CS224N Assignment 1: Exploring Word Vectors (25 Points)

Due 4:30pm, Tue Jan 14

Welcome to CS224n!

Before you start, make sure you read the README.txt in the same directory as this notebook. You will
notebook. We highly encourage you to read and understand the provided codes as part of the learning

```
# 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
nltk.download('reuters')
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 /root/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 a
etc., so it is important to build some intuitions as to their strengths and weaknesses. Here, you will ex
derived from *co-occurrence matrices*, and those derived via *GloVe*.

Assignment Notes: Please make sure to save the notebook as you go along. Submission Instructions
notebook.

Note on Terminology: The terms "word vectors" and "word embeddings" are often used interchangeal
fact that we are encoding aspects of a word's meaning in a lower dimensional space. As [Wikipedia](#) st
mathematical embedding from a space with one dimension per word to a continuous vector space with

Part 1: Count-Based Word Vectors (10 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, wi
similar words will often be spoken or written along with a shared subset of words, i.e., contexts. By ex
develop embeddings for our words. With this intuition in mind, many "old school" approaches to cons
counts. Here we elaborate upon one of those strategies, *co-occurrence matrices* (for more informatio

Co-Occurrence

A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i
the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n precedin
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 i
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 sentenc case we imagine `<START>` and `<END>` tokens encapsulating each document, e.g., " `<START>` All that gli 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 in general (linear in the number of distinct words in a corpus). Thus, our next step is to run *dimension SVD (Singular Value Decomposition)*, which is a kind of generalized *PCA (Principal Components Analys* components. Here's a visualization of dimensionality reduction with SVD. In this picture our co-occurr corresponding to n words. We obtain a full matrix decomposition, with the singular values ordered in shorter length-k word vectors in U_k.

Picture of an SVD

This reduced-dimensionality co-occurrence representation preserves semantic relationships between closer than *doctor* and *dog*.

Notes: If you can barely remember what an eigenvalue is, here's a slow, friendly introduction to SVD. I PCA or SVD, feel free to check out lectures 7, 8, and 9 of CS168. These course notes provide a great h purpose algorithms. Though, for the purpose of this class, you only need to know how to extract the k programmed implementations of these algorithms from the numpy, scipy, or sklearn python packages SVD to large corpora because of the memory needed to perform PCA or SVD. However, if you only wa relatively small k — known as Truncated SVD — then there are reasonably scalable techniques to com

Plotting Co-Occurrence Word Embeddings

Here, we will be using the Reuters (business and financial news) corpus. If you haven't run the import now (click it and press SHIFT-RETURN). The corpus consists of 10,788 news documents totaling 1.3 r categories and are split into train and test. For more details, please see https://www.nltk.org/book/ch function below that pulls out only articles from the "crude" (i.e. news articles about oil, gas, etc.) categ `<END>` tokens to each of the documents, and lowercases words. You do **not** have to perform any othe

```
def read_corpus(category="crude"):
    """ 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] for f i
```

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

```
reuters_corpus = read_corpus()
pprint.pprint(reuters_corpus[:5], compact=True, width=100)
```

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'demand', 'downward
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'miti', ')', 'will
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outlook', 'by', 'augus
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'demand', ',', 'mini
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'the', 'projection'
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '550', 'mln', 'kil
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the', 'decision', 'fo
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'industry', 'foll
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'decline', 'in',
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'work', 'out', 'a'
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberations', 'of', 'commi
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and', 'energy', ',',
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'review', 'the', 'br
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nuclear', ',', 'co
  'natural', 'gas', '.', 'nuclear', 'energy', 'provided', 'the', 'bulk', 'of', 'japan',
  'electric', 'power', 'in', 'the', 'fiscal', 'year', 'ended', 'march', '31', ',', 'supp
  'an', 'estimated', '27', 'pct', 'on', 'a', 'kilowatt', '/', 'hour', 'basis', ',', 'fol
  'by', 'oil', '(', '23', 'pct', ')', 'and', 'liquefied', 'natural', 'gas', '(', '21', '
  'they', 'noted', '.', '<END>'],
 ['<START>', 'energy', '/', 'u', '.', 's', '.', 'petrochemical', 'industry', 'cheap', 'o
  'feedstocks', ',', 'the', 'weakened', 'u', '.', 's', '.', 'dollar', 'and', 'a', 'plant
  'utilization', 'rate', 'approaching', '90', 'pct', 'will', 'propel', 'the', 'streamlin
  '.', 's', '.', 'petrochemical', 'industry', 'to', 'record', 'profits', 'this', 'year',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'major', 'company
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'chemical', 'manu
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'businesses', 'has', '
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'cain', 'chemica
  ',', 'and', 'other', 'firms', 'to', 'aggressively', 'seek', 'acquisitions', 'of', 'pet
  'plants', '.', 'oil', 'companies', 'such', 'as', 'ashland', 'oil', 'inc', '&', 'lt', '
  '>,', 'the', 'kentucky', '-', 'based', 'oil', 'refiner', 'and', 'marketer', ',', 'are'
  'shopping', 'for', 'money', '-', 'making', 'petrochemical', 'businesses', 'to', 'buy',
  'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'of', 'a', 'golden', 'period', '
  'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow', 'chemical', 'co', '&', 'lt'
  'dow', '>,', 'adding', ',', '"', 'there', "'", 's', 'no', 'major', 'plant', 'capacity'
  'added', 'around', 'the', 'world', 'now', '.', 'the', 'whole', 'game', 'is', 'bringing
  'new', 'products', 'and', 'improving', 'the', 'old', 'ones', '."', 'analysts', 'say',
  'chemical', 'industry', "'", 's', 'biggest', 'customers', ',', 'automobile', 'manufact
  'and', 'home', 'builders', 'that', 'use', 'a', 'lot', 'of', 'paints', 'and', 'plastics
  'are', 'expected', 'to', 'buy', 'quantities', 'this', 'year', '.', 'u', '.', 's', '.',
  'petrochemical', 'plants', 'are', 'currently', 'operating', 'at', 'about', '90', 'pct'
  'capacity', ',', 'reflecting', 'tighter', 'supply', 'that', 'could', 'hike', 'product'
  'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'said', 'john', 'dosher', ',', 'ma
  'director', 'of', 'pace', 'consultants', 'inc', 'of', 'houston', '.', 'demand', 'for',
  'products', 'such', 'as', 'styrene', 'could', 'push', 'profit', 'margins', 'up', 'by',
  'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreffice', ',', 'speaking', 'at',
  'meeting', 'of', 'chemical', 'engineers', 'in', 'houston', ',', 'said', 'dow', 'would'
  'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'last', 'year', 'and', 'predicted'
  'would', 'have', 'the', 'best', 'year', 'in', 'its', 'history', '.', 'in', '1985', ','
  'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs', 'a', 'barrel', 'and', 'chemic
  'exports', 'were', 'adversely', 'affected', 'by', 'the', 'strong', 'u', '.', 's', '.',
  ',', 'dow', 'had', 'profits', 'of', '58', 'mln', 'dlrs', '.', '"', 'i', 'believe', 'th
  'entire', 'chemical', 'industry', 'is', 'headed', 'for', 'a', 'record', 'year', 'or',
  'to', 'it', ',"', 'oreffice', 'said', '.', 'gaf', 'chairman', 'samuel', 'heyman', 'est
  'that', 'the', 'u', '.', 's', '.', 'chemical', 'industry', 'would', 'report', 'a', '20
  'gain', 'in', 'profits', 'during', '1987', '.', 'last', 'year', ',', 'the', 'domestic'
  'industry', 'earned', 'a', 'total', 'of', '13', 'billion', 'dlrs', ',', 'a', '54', 'pc
  'from', '1985', '.', 'the', 'turn', 'in', 'the', 'fortunes', 'of', 'the', 'once', '-',
```

```
 'chemical', 'industry', 'has', 'been', 'brought', 'about', 'by', 'a', 'combination', '
 'and', 'planning', ',', 'said', 'pace', "'", 's', 'john', 'dosher', '.', 'dosher', 'sa
 'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made', 'feedstocks', 'dramatically',
 'and', 'at', 'the', 'same', 'time', 'the', 'american', 'dollar', 'was', 'weakening', '
 'foreign', 'currencies', '.', 'that', 'helped', 'boost', 'u', '.', 's', '.', 'chemical
 'exports', '.', 'also', 'helping', 'to', 'bring', 'supply', 'and', 'demand', 'into', '
 'has', 'been', 'the', 'gradual', 'market', 'absorption', 'of', 'the', 'extra', 'chemic
 'manufacturing', 'capacity', 'created', 'by', 'middle', 'eastern', 'oil', 'producers',
 'the', 'early', '1980s', '.', 'finally', ',', 'virtually', 'all', 'major', 'u', '.', '
 'chemical', 'manufacturers', 'have', 'embarked', 'on', 'an', 'extensive', 'corporate',
 'restructuring', 'program', 'to', 'mothball', 'inefficient', 'plants', ',', 'trim', 't
 'payroll', 'and', 'eliminate', 'unrelated', 'businesses', '.', 'the', 'restructuring',
 'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile', 'takeover', 'attempts', '.',
 'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1985', 'to', 'acquire', 'unio
 'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently', 'offered', 'three', 'billio
 'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor', '>,', 'a', 'chicago', 'manufac
 'of', 'plastics', 'and', 'chemicals', '.', 'another', 'industry', 'powerhouse', ',', '
 'r', '.', 'grace', '&', 'lt', ';', 'gra', '>', 'has', 'divested', 'its', 'retailing',
 'restaurant', 'and', 'fertilizer', 'businesses', 'to', 'raise', 'cash', 'for', 'chemic
 'acquisitions', '.', 'but', 'some', 'experts', 'worry', 'that', 'the', 'chemical', 'in
 'may', 'be', 'headed', 'for', 'trouble', 'if', 'companies', 'continue', 'turning', 'th
 'back', 'on', 'the', 'manufacturing', 'of', 'staple', 'petrochemical', 'commodities',
 'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profitable', 'specialty', 'chemic
 'that', 'are', 'custom', '-', 'designed', 'for', 'a', 'small', 'group', 'of', 'buyers'
 'companies', 'like', 'dupont', '&', 'lt', ';', 'dd', '>', 'and', 'monsanto', 'co', '&'
 'mtc', '>', 'spent', 'the', 'past', 'two', 'or', 'three', 'years', 'trying', 'to', 'ge
 'of', 'the', 'commodity', 'chemical', 'business', 'in', 'reaction', 'to', 'how', 'badl
 'market', 'had', 'deteriorated', ',"', 'dosher', 'said', '.', '"', 'but', 'i', 'think'
 'will', 'eventually', 'kill', 'the', 'margins', 'on', 'the', 'profitable', 'chemicals'
 'the', 'niche', 'market', '."', 'some', 'top', 'chemical', 'executives', 'share', 'the
 'concern', '.', '"', 'the', 'challenge', 'for', 'our', 'industry', 'is', 'to', 'keep',
 'getting', 'carried', 'away', 'and', 'repeating', 'past', 'mistakes', ',"', 'gaf', "'"
 'heyman', 'cautioned', '.', '"', 'the', 'shift', 'from', 'commodity', 'chemicals', 'ma
 'ill', '-', 'advised', '.', 'specialty', 'businesses', 'do', 'not', 'stay', 'special',
 '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'created', 'this', 'month', 'b
 'sterling', 'investment', 'banking', 'group', ',', 'believes', 'it', 'can', 'generate'
 'mln', 'dlrs', 'in', 'annual', 'sales', 'by', 'bucking', 'the', 'industry', 'trend', '
 'chairman', 'gordon', 'cain', ',', 'who', 'previously', 'led', 'a', 'leveraged', 'buyo
 'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical', 'business', ',', 'has', 'sp
 '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'buy', 'seven', 'petrochemical'
 'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'plants', 'produce', 'only', 'ba
 'commodity', 'petrochemicals', 'that', 'are', 'the', 'building', 'blocks', 'of', 'spec
 'products', '.', '"', 'this', 'kind', 'of', 'commodity', 'chemical', 'business', 'will
 'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'business', ',"', 'cain', 'said',
 'adding', 'that', 'demand', 'is', 'expected', 'to', 'grow', 'by', 'about', 'three', 'p
 'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'with', 'dean', 'witter', 'rey
 'said', 'chemical', 'makers', 'have', 'also', 'benefitted', 'by', 'increasing', 'deman
 'plastics', 'as', 'prices', 'become', 'more', 'competitive', 'with', 'aluminum', ',',
 'and', 'steel', 'products', '.', 'armen', 'estimated', 'the', 'upturn', 'in', 'the', '
 'business', 'could', 'last', 'as', 'long', 'as', 'four', 'or', 'five', 'years', ',', '
 'the', 'u', '.', 's', '.', 'economy', 'continues', 'its', 'modest', 'rate', 'of', 'gro
 '<END>'],
['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'solve', 'dispute', 'turkey', '
 'today', 'its', 'disputes', 'with', 'greece', ',', 'including', 'rights', 'on', 'the',
 'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',', 'should', 'be', 'solved', '
 'negotiations', '.', 'a', 'foreign', 'ministry', 'statement', 'said', 'the', 'latest',
 'between', 'the', 'two', 'nato', 'members', 'stemmed', 'from', 'the', 'continental', '
 'dispute', 'and', 'an', 'agreement', 'on', 'this', 'issue', 'would', 'effect', 'the',
```

```
',', 'economy', 'and', 'other', 'rights', 'of', 'both', 'countries', '.', '"', 'as', '
'issue', 'is', 'basicly', 'political', ',', 'a', 'solution', 'can', 'only', 'be', 'fou
'bilateral', 'negotiations', ',"', 'the', 'statement', 'said', '.', 'greece', 'has', '
'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'be', 'solved', 'at', 'the',
'international', 'court', 'of', 'justice', '.', 'the', 'two', 'countries', 'approached
'confrontation', 'last', 'month', 'after', 'greece', 'announced', 'it', 'planned', 'oi
'exploration', 'work', 'in', 'the', 'aegean', 'and', 'turkey', 'said', 'it', 'would',
'search', 'for', 'oil', '.', 'a', 'face', '-', 'off', 'was', 'averted', 'when', 'turke
'confined', 'its', 'research', 'to', 'territorrial', 'waters', '.', '"', 'the', 'lates
'crises', 'created', 'an', 'historic', 'opportunity', 'to', 'solve', 'the', 'disputes'
'the', 'two', 'countries', ',"', 'the', 'foreign', 'ministry', 'statement', 'said', '.
"'", 's', 'ambassador', 'in', 'athens', ',', 'nazmi', 'akiman', ',', 'was', 'due', 'to
```

Question 1.1: Implement `distinct_words` [code] (2 points)

Write a method to work out the distinct words (word types) that occur in the corpus. You can do this v it with Python list comprehensions. In particular, [this](#) may be useful to flatten a list of lists. If you're nc in general, here's [more information](#).

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): list of distinct words across the corpus, sorted
            num_corpus_words (integer): number of distinct words across the corpus
    """
    corpus_words = []
    num_corpus_words = -1
    # ------------------
    # Write your implementation here.
    for x in corpus:
      for word in x:
          if word not in corpus_words:
              corpus_words.append(word)
    num_corpus_words=len(corpus_words)
    corpus_words.sort()
    # ------------------

    return corpus_words, num_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).split(" ")
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", "glitter
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 words. Correc

# Test correct words
assert (test_corpus_words == ans_test_corpus_words), "Incorrect corpus_words.\nCorrect: {}\nY

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

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

Question 1.2: Implement `compute_co_occurrence_matrix` [code] (3 points)

Write a method that constructs a co-occurrence matrix for a certain window-size n (with a default of after the word in the center of the window. Here, we start to use `numpy (np)` to represent vectors, ma with NumPy, there's a NumPy tutorial in the second half of this cs231n Python NumPy tutorial.

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

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

              For example, if we take the document "<START> All that glitters is not gold <EN
              "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 corpus , numb
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should be the same as the order
            word2Ind (dict): dictionary that maps word to index (i.e. row/column number) for
    """
    words, num_words = distinct_words(corpus)
    M = np.zeros((num_words,num_words))
    word2Ind = {words[i]: i for i in range(0, len(words), 1)}
    # ------------------
    # Write your implementation here.
    for x in corpus:
```

```
      index=0
      for word in x:
        if index-window_size>0:
          l = index-window_size
        else:
          l = 0

        if  index+window_size+1<len(x):
          r = index+window_size+1
        else:
          r = len(x)

        words_in_window = x[l:index] + x[index+1:r]
        for temp in words_in_window:
          M[word2Ind[word],word2Ind[temp]]+=1
        index+=1
    # ------------------

    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).split(" ")
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", "glitter
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: {}\nYours: {}"

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

```
# 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 matrix M. You

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

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

Question 1.3: Implement `reduce_to_k_dim` [code] (1 point)

Construct a method that performs dimensionality reduction on the matrix to produce k-dimensional e 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 sc implementation of Truncated SVD, and only sklearn provides an efficient randomized algorithm for ca 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`
    print("Running Truncated SVD over %i words..." % (M.shape[0]))
        # ------------------
        # Write your implementation here.
    SVD = TruncatedSVD(n_components=k, algorithm='randomized', n_iter=n_iters, random_state=N
    M_reduced=SVD.fit_transform(M)
```

```
        # ------------------
    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).split(" ")
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 {}".format(M_test
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should have {}".format(M_te

# 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] (1 point)

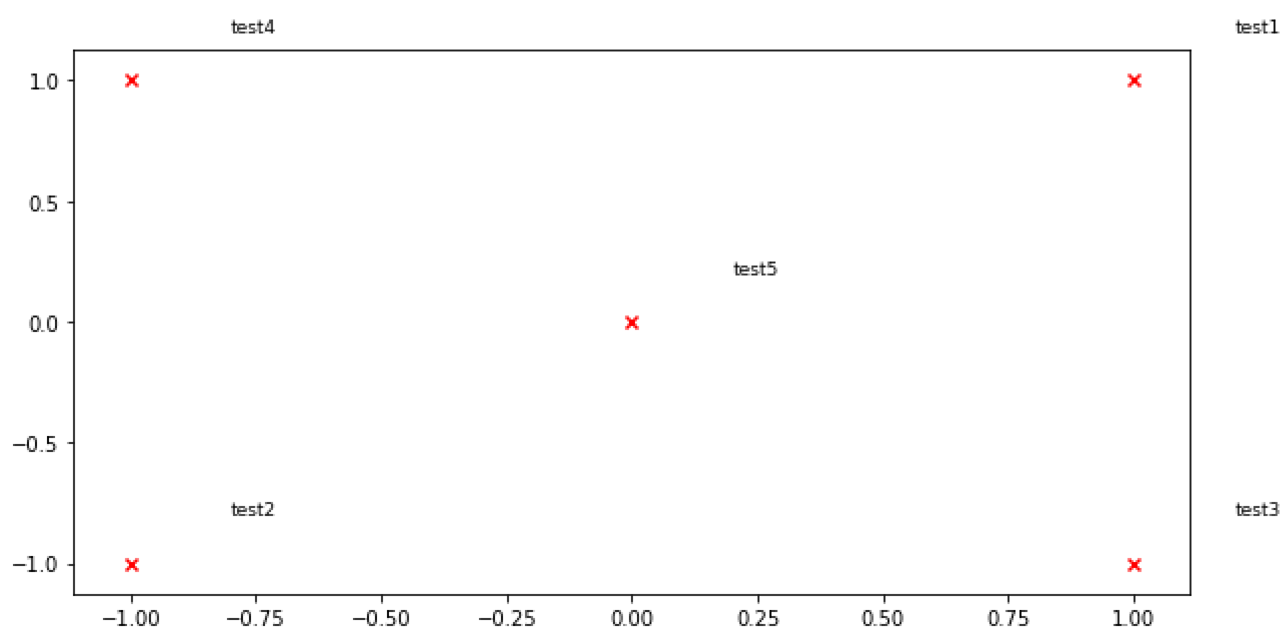
Here you will write a function to plot a set of 2D vectors in 2D space. For graphs, we will use Matplotli

For this example, you may find it useful to adapt [this code](). In the future, a good way to make a plot is 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.
    print(M_reduced)
    for i,type in enumerate(words):
      x = M_reduced[i][0]
      y = M_reduced[i][1]
      plt.scatter(x, y, marker='x', color='red')
      plt.text(x+0.2, y+0.2, type, fontsize=9)

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


# ---------------------
# 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:
[[ 1  1]
 [-1 -1]
 [ 1 -1]
 [-1  1]
 [ 0  0]]
--------------------------------------------------------------------------------
```



Test Plot Solution

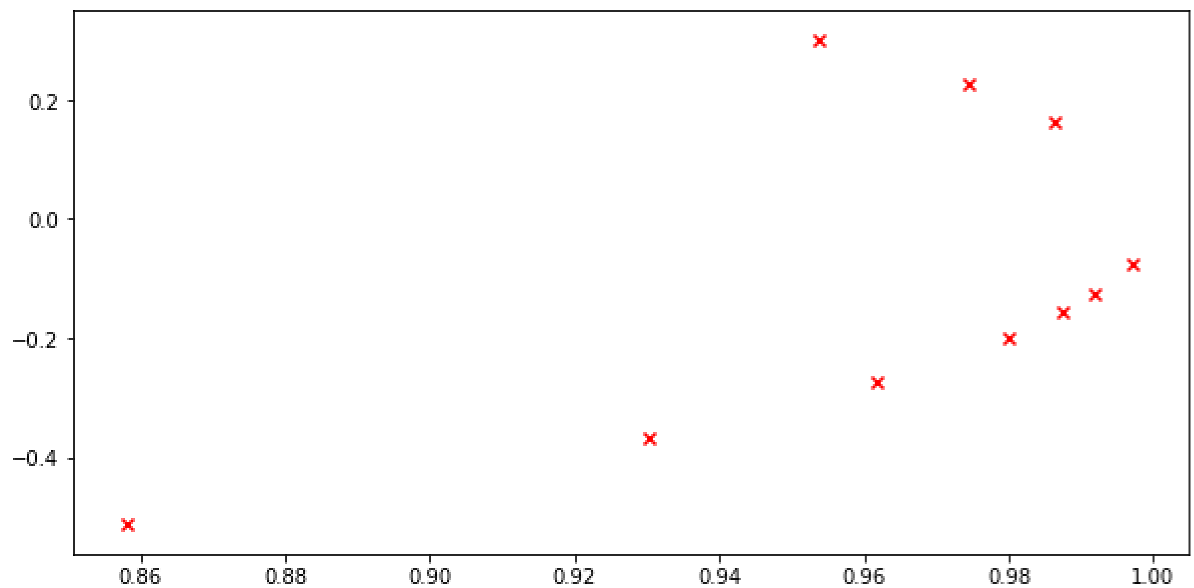
Question 1.5: Co-Occurrence Plot Analysis [written] (3 points)

Now we will put together all the parts you have written! We will compute the co-occurrence matrix wit size), over the Reuters "crude" (oil) corpus. Then we will use TruncatedSVD to compute 2-dimensional returns U*S, so we need to normalize the returned vectors, so that all the vectors will appear around tl directional closeness). **Note**: The line of code below that does the normalizing uses the NumPy conce 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. What clusters together doesn't cluster together that you might think should have? **Note:** "bpd" stands for "barrels per day" anc crude oil topic articles.

```
# ----------------------------
# Run This Cell to Produce Your Plot
# -----------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2Ind_co_occurrence = compute_co_occurrence_matrix(reuters_corpus)
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 = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petro

plot_embeddings(M_normalized, word2Ind_co_occurrence, words)
```

```
Running Truncated SVD over 8185 words...
Done.
[[ 0.98750922 -0.15756123]
 [ 0.99184344 -0.12746215]
 [ 0.97435474  0.22501741]
 ...
 [ 0.95936328 -0.28217386]
 [ 0.98010885 -0.19846067]
 [ 0.995596    0.09374751]]
```



Write your answer here.

Clusters:

- ecuador, output
- petroleum, bpd, barrels, venezuela

Should have clustered:

- ecuador, venezuela, kuwait

Part 2: Prediction-Based Word Vectors (15 points)

As discussed in class, more recently prediction-based word vectors have demonstrated better perforr (which also utilizes the benefit of counts). Here, we shall explore the embeddings produced by GloVe. slides for more details on the word2vec and GloVe algorithms. If you're feeling adventurous, challeng paper.

Then run the following cells to load the GloVe vectors into memory. **Note**: If this is your first time to ru
model, it will take about 15 minutes to run. If you've run these cells before, rerunning them will load th

```
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(wv_from_bin.vocab.keys()))
    return wv_from_bin


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

```
/usr/local/lib/python3.6/dist-packages/smart_open/smart_open_lib.py:402: UserWarning: Th
  'See the migration notes for details: %s' % _MIGRATION_NOTES_URL
Loaded vocab size 400000
```

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

Reducing dimensionality of Word Embeddings

Let's directly compare the GloVe embeddings to those of the co-occurrence matrix. In order to avoid r
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-dimens

```
def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd', 'ecuador', 'energy',
    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loaded from file
        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.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


# -----------------------------------------------------------------
# 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)
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 10010 words...
Done.
```

Note: If you are receiving out of memory issues on your local machine, try closing other applications You may want to try restarting your machine so that you can free up extra memory. Then immediate can load the word vectors properly. If you still have problems with loading the embeddings onto you the Piazza instructions, as how to run remotely on Stanford Farmshare machines.

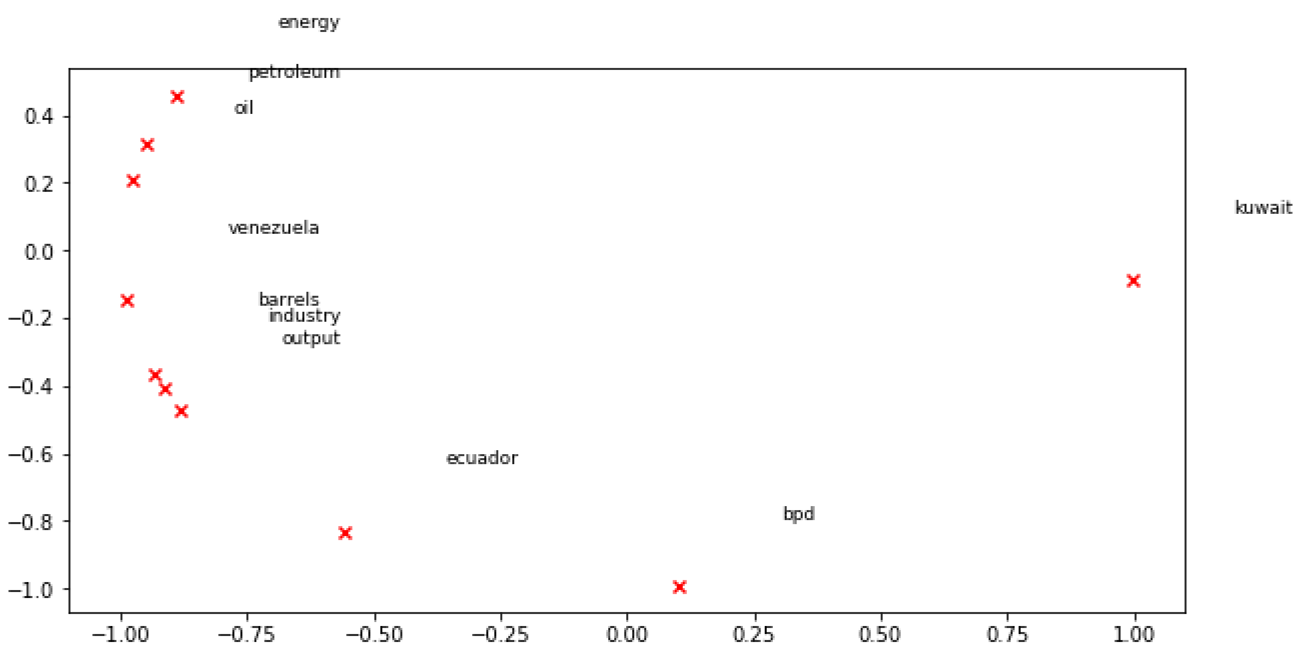
Question 2.1: GloVe Plot Analysis [written] (4 points)

Run the cell below to plot the 2D GloVe embeddings for `['barrels', 'bpd', 'ecuador', 'energy', 'petroleum', 'venezuela']`.

What clusters together in 2-dimensional embedding space? What doesn't cluster together that you mi different from the one generated earlier from the co-occurrence matrix? What is a possible reason for

```
words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petro
plot_embeddings(M_reduced_normalized, word2Ind, words)
```

```
[[-0.93170464 -0.3632168 ]
 [ 0.10210682 -0.99477345]
 [-0.5568495  -0.83061343]
 ...
 [ 0.8177966  -0.57550746]
 [ 0.9868982  -0.16134427]
 [ 0.99160886 -0.12927444]]
```



Write your answer here.

Clusters:

- energy, petroleom, oil
- barrels, industry, output, venezuela

should have clustered:

- venezuela, kuwait, ecuador
- bpd, petroleum, barrels

Values got more closer because it depends on the ratios of Co-occurance probability here

Cosine Similarity

Now that we have word vectors, we need a way to quantify the similarity between individual words, ac 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 L amount of space "we must travel" to get between these two points. Another approach is to examine tl trigonometry we know that:

Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos(\Theta)$ 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 (2 points) [code + written]

Polysemes and homonyms are words that have more than one meaning (see this wiki page to learn m polysemes and homonyms). Find a word with at least 2 different meanings such that the top-10 mos similarity) contain related words from *both* meanings. For example, "leaves" has both "vanishes" and ' "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous or homonymi the word you discover and the multiple meanings that occur in the top 10. Why do you think many of t 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. vocabulary with respect to their cosine similarity to the given word. For further assistance please che

```
# ------------------
# Write your implementation here.
wv_from_bin.most_similar("nails")

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

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
[('nail', 0.6682122349739075),
 ('screws', 0.6072793006896973),
 ('fingernails', 0.589718222618103),
 ('bolts', 0.551854133605957),
 ('reznor', 0.5144748687744141),
 ('plastic', 0.5015807151794434),
 ('metal', 0.49925774335861206),
 ('inch', 0.49902814626693726),
 ('toenails', 0.4934726357460022),
 ('rusted', 0.4670681357383728)]
```

Write your answer here.

Nails has 2 meanings:

- Nails in fingers
- Nails as a sharp metal piece

Maybe some words from the top 10 have different shape as being upper case so it wasn't able to dete
the dataset

Question 2.3: Synonyms & Antonyms (2 points) [code + written]

When considering Cosine Similarity, it's often more convenient to think of Cosine Distance, which is si

Find three words (w1,w2,w3) where w1 and w2 are synonyms and w1 and w3 are antonyms, but Cosi
Distance(w1,w2). For example, w1="happy" is closer to w3="sad" than to w2="cheerful".

Once you have found your example, please give a possible explanation for why this counter-intuitive r

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

```
# ------------------
# Write your implementation here.
print('cos distance of w1,w3 ->' ,wv_from_bin.distance('father', 'mother'))
print('cos distance of w1,w2 ->', wv_from_bin.distance('father', 'man'))
# ------------------
```

```
cos distance of w1,w3 -> 0.20632314682006836
cos distance of w1,w2 -> 0.3738422989845276
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
```

Write your answer here.

The cosine distance between W1 and W3 is less than that between W1 and W2 so it detects words of
more similar So in this example the sidtance between father and mother is less than between father a

Solving Analogies with Word Vectors

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

In the cell below, we show you how to use word vectors to find x. The `most_similar` function finds wo
the `positive` list and most dissimilar from the words in the `negative` list. The answer to the analogy
(largest numerical value).

Note: Further Documentation on the `most_similar` function can be found within the GenSim docume

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

```
[('queen', 0.6978678703308105),
 ('princess', 0.6081745028495789),
 ('monarch', 0.5889754891395569),
 ('throne', 0.5775108933448792),
 ('prince', 0.5750998854637146),
 ('elizabeth', 0.546359658241272),
 ('daughter', 0.5399125814437866),
 ('kingdom', 0.5318052768707275),
 ('mother', 0.5168544054031372),
 ('crown', 0.5164472460746765)]
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
```

Question 2.4: Finding Analogies [code + written] (2 Points)

Find an example of analogy that holds according to these vectors (i.e. the intended word is ranked top analogy in the form x:y :: a:b. If you believe the analogy is complicated, explain why the analogy holds

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

```
# ------------------
# Write your implementation here.
pprint.pprint(wv_from_bin.most_similar(positive=['happy', 'sad'], negative=['proud']))

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

```
[('sorry', 0.5316814184188843),
 ('tragic', 0.5281333923339844),
 ('unhappy', 0.5133940577507019),
 ('strange', 0.5081397891044617),
 ('pretty', 0.4955098628997803),
 ('happen', 0.4930151104927063),
 ('mood', 0.49234145879745483),
 ('feeling', 0.49193012714385986),
 ('unfortunate', 0.4919125735759735),
 ('scary', 0.48966652154922485)]
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
```

Write your answer here.

happy : sad :: proud: ....

Question 2.5: Incorrect Analogy [code + written] (1 point)

Find an example of analogy that does *not* hold according to these vectors. In your solution, state the i
state the (incorrect) value of b according to the word vectors.

```
# ------------------
# Write your implementation here.
pprint.pprint(wv_from_bin.most_similar(positive=['happy', 'proud'], negative=['laugh']))

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

```
[('pleased', 0.6534539461135864),
 ('glad', 0.6373730897903442),
 ('grateful', 0.5932959914207458),
 ("'m", 0.5777617692947388),
 ('delighted', 0.5720453262329102),
 ('very', 0.5710911750793457),
 ('thankful', 0.5684477090835571),
 ('truly', 0.566472053527832),
 ('confident', 0.5524917244911194),
 ('honored', 0.5501493811607361)]
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
```

Write your answer here.

Question 2.6: 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 e
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 di
most similar to "man" and "worker" and most dissimilar to "woman". Point out the difference between
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']))
```

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
[('employee', 0.6375863552093506),
 ('workers', 0.6068919897079468),
 ('nurse', 0.5837947726249695),
 ('pregnant', 0.5363885164260864),
 ('mother', 0.5321309566497803),
 ('employer', 0.5127025842666626),
 ('teacher', 0.5099576711654663),
 ('child', 0.5096741914749146),
 ('homemaker', 0.5019454956054688),
 ('nurses', 0.4970572590827942)]

[('workers', 0.6113258004188538),
 ('employee', 0.5983108282089233),
 ('working', 0.5615328550338745),
 ('laborer', 0.5442320108413696),
 ('unemployed', 0.5368517637252808),
 ('job', 0.5278826951980591),
 ('work', 0.5223963260650635),
 ('mechanic', 0.5088937282562256),
 ('worked', 0.505452036857605),
 ('factory', 0.4940453767776489)]
```

Write your answer here.

In female list ->> from top answers are (pregnant, mother)

In male list ->> from top answers no any woman exact related words

Question 2.7: Independent Analysis of Bias in Word Vectors [code + written] (1 pc

Use the `most_similar` function to find another case where some bias is exhibited by the vectors. Plea
you discover.

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

pprint.pprint(wv_from_bin.most_similar(positive=['old', 'grandfather'], negative=['young']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['young', 'child'], negative=['old']))
    # ------------------
```

```
/usr/local/lib/python3.6/dist-packages/gensim/matutils.py:737: FutureWarning: Conversion
  if np.issubdtype(vec.dtype, np.int):
[('father', 0.5955848693847656),
 ('grandmother', 0.5890868306159973),
 ('grandson', 0.5681414604187012),
 ('55-year', 0.5370609760284424),
 ('uncle', 0.5290553569793701),
 ('60-year', 0.5252043008804321),
 ('great-great', 0.5241730809211731),
 ('35-year', 0.5190296173095703),
 ('50-year', 0.5185691118240356),
 ('47-year', 0.5144185423851013)]

[('children', 0.6739084720611572),
 ('youngsters', 0.5904493927955627),
 ('parents', 0.5897642970085144),
 ('girls', 0.5872694849967957),
 ('adolescents', 0.5764964818954468),
 ('adults', 0.5760174989700317),
 ('teenagers', 0.5647995471954346),
 ('mothers', 0.5531293153762817),
 ('kids', 0.5519955158233643),
 ('teens', 0.5455273389816284)]
```

Write your answer here.

The two output lists differentaite accoriding to ages one for adults and the other for kids and young p

Question 2.8: Thinking About Bias [written] (2 points)

What might be the causes of these biases in the word vectors? You should give least 2 explainations might you be able to investigate/test these causes?

Write your answer here.

I think it depends on the input dataset the model is build and trained on as each dataset expacially foı mentallity of the people's society so for example if people are dealing with those biases often it woulc languages so be of high weights in data sets.

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 ce
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 using "PDF via L Make sure all your solutions especially the coding parts are displayed in the pdf, it's okay if the provided code

code cells).

5. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing y
6. Submit your PDF on Gradescope.

Aa .* Find Replace with REPLACE