exploring_word_vectors

January 14, 2019

1 CS224N Assignment 1: Exploring Word Vectors (25 Points)

Welcome to CS224n!

Before you start, make sure you read the README.txt in the same directory as this notebook.

In [1]:
```
# All Import Statements Defined Here
# Note: Do not add to this list.
# All the dependencies you need, can be installed by running .
# ----------------

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
[nltk_data]     /Users/jaigupta/nltk_data...
```

```
[nltk_data]   Package reuters is already up-to-date!
```

1.1 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 *word2vec*.

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

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*".

1.2 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, 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 or here).

1.2.1 Co-Occurrence

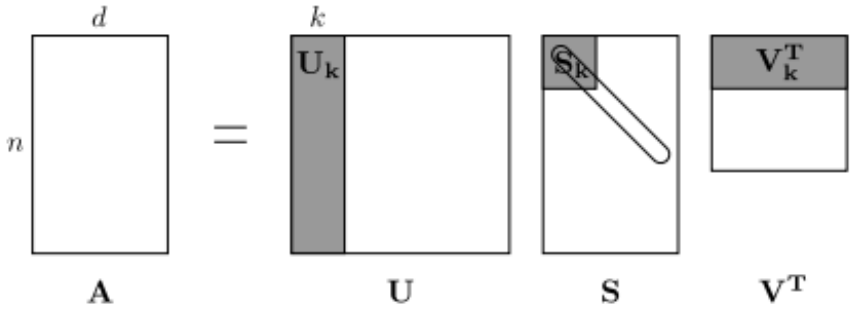
A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ 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.

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



Picture of an SVD

*	START	all	that	glitters	is	not	gold	well	ends	END
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 thise 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 lectures 7, 8, and 9 of CS168. 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.

1.2.2 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 "crude" (i.e. news articles about oil, gas, etc.) category. The function also adds START and END tokens to each of the documents, and lowercases words. You do **not** have perform any other kind of pre-processing.

```
In [2]: 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] :
```

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

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

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'demand', 'downwards', 'tl
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'miti', ')', 'will', 'rev
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outlook', 'by', 'august', 'to
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'demand', ',', 'ministry',
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'the', 'projection', 'for
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '550', 'mln', 'kilolitres
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the', 'decision', 'follows'
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'industry', 'following'
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'decline', 'in', 'domest
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'work', 'out', 'a', 'revi
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberations', 'of', 'committee',
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and', 'energy', ',', 'the'
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'review', 'the', 'breakdown
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nuclear', ',', 'coal', 'a
  'natural', 'gas', '.', 'nuclear', 'energy', 'provided', 'the', 'bulk', 'of', 'japan', "'", 's
  'electric', 'power', 'in', 'the', 'fiscal', 'year', 'ended', 'march', '31', ',', 'supplying'
  'an', 'estimated', '27', 'pct', 'on', 'a', 'kilowatt', '/', 'hour', 'basis', ',', 'followed'
  'by', 'oil', '(', '23', 'pct', ')', 'and', 'liquefied', 'natural', 'gas', '(', '21', 'pct',
  'they', 'noted', '.', '<END>'],
 ['<START>', 'energy', '/', 'u', '.', 's', '.', 'petrochemical', 'industry', 'cheap', 'oil',
  'feedstocks', ',', 'the', 'weakened', 'u', '.', 's', '.', 'dollar', 'and', 'a', 'plant',
  'utilization', 'rate', 'approaching', '90', 'pct', 'will', 'propel', 'the', 'streamlined', 'u
  '.', 's', '.', 'petrochemical', 'industry', 'to', 'record', 'profits', 'this', 'year', ',',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'major', 'company',
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'chemical', 'manufactur
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'businesses', 'has', 'prompte
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'cain', 'chemical', 'in
  ',', 'and', 'other', 'firms', 'to', 'aggressively', 'seek', 'acquisitions', 'of', 'petrochemi
  'plants', '.', 'oil', 'companies', 'such', 'as', 'ashland', 'oil', 'inc', '&', 'lt', ';', 'as
```

'>,', 'the', 'kentucky', '-', 'based', 'oil', 'refiner', 'and', 'marketer', ',', 'are', 'also
'shopping', 'for', 'money', '-', 'making', 'petrochemical', 'businesses', 'to', 'buy', '.',
'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'of', 'a', 'golden', 'period', ,"', 's
'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow', 'chemical', 'co', '&', 'lt', ';',
'dow', '>,', 'adding', ',', '"', 'there', "'", 's', 'no', 'major', 'plant', 'capacity', 'bein
'added', 'around', 'the', 'world', 'now', '.', 'the', 'whole', 'game', 'is', 'bringing', 'out
'new', 'products', 'and', 'improving', 'the', 'old', 'ones', '."', 'analysts', 'say', 'the',
'chemical', 'industry', "'", 's', 'biggest', 'customers', ',', 'automobile', 'manufacturers'
'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', 'pric
'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'said', 'john', 'dosher', ',', 'managing
'director', 'of', 'pace', 'consultants', 'inc', 'of', 'houston', '.', 'demand', 'for', 'some
'products', 'such', 'as', 'styrene', 'could', 'push', 'profit', 'margins', 'up', 'by', 'as',
'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreffice', ',', 'speaking', 'at', 'a',
'meeting', 'of', 'chemical', 'engineers', 'in', 'houston', ',', 'said', 'dow', 'would', 'easi
'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'last', 'year', 'and', 'predicted', 'it'
'would', 'have', 'the', 'best', 'year', 'in', 'its', 'history', '.', 'in', '1985', ',', 'when
'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs', 'a', 'barrel', 'and', 'chemical',
'exports', 'were', 'adversely', 'affected', 'by', 'the', 'strong', 'u', '.', 's', '.', 'dolla
',', 'dow', 'had', 'profits', 'of', '58', 'mln', 'dlrs', '.', '"', 'i', 'believe', 'the',
'entire', 'chemical', 'industry', 'is', 'headed', 'for', 'a', 'record', 'year', 'or', 'close
'to', 'it', ,"', 'oreffice', 'said', '.', 'gaf', 'chairman', 'samuel', 'heyman', 'estimated
'that', 'the', 'u', '.', 's', '.', 'chemical', 'industry', 'would', 'report', 'a', '20', 'pct
'gain', 'in', 'profits', 'during', '1987', '.', 'last', 'year', ',', 'the', 'domestic',
'industry', 'earned', 'a', 'total', 'of', '13', 'billion', 'dlrs', ',', 'a', '54', 'pct', 'le
'from', '1985', '.', 'the', 'turn', 'in', 'the', 'fortunes', 'of', 'the', 'once', '-', 'sickl
'chemical', 'industry', 'has', 'been', 'brought', 'about', 'by', 'a', 'combination', 'of', 'l
'and', 'planning', ',', 'said', 'pace', "'", 's', 'john', 'dosher', '.', 'dosher', 'said', 'l
'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made', 'feedstocks', 'dramatically', 'cheap
'and', 'at', 'the', 'same', 'time', 'the', 'american', 'dollar', 'was', 'weakening', 'against
'foreign', 'currencies', '.', 'that', 'helped', 'boost', 'u', '.', 's', '.', 'chemical',
'exports', '.', 'also', 'helping', 'to', 'bring', 'supply', 'and', 'demand', 'into', 'balance
'has', 'been', 'the', 'gradual', 'market', 'absorption', 'of', 'the', 'extra', 'chemical',
'manufacturing', 'capacity', 'created', 'by', 'middle', 'eastern', 'oil', 'producers', 'in',
'the', 'early', '1980s', '.', 'finally', ',', 'virtually', 'all', 'major', 'u', '.', 's', '.
'chemical', 'manufacturers', 'have', 'embarked', 'on', 'an', 'extensive', 'corporate',
'restructuring', 'program', 'to', 'mothball', 'inefficient', 'plants', ',', 'trim', 'the',
'payroll', 'and', 'eliminate', 'unrelated', 'businesses', '.', 'the', 'restructuring', 'touch
'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile', 'takeover', 'attempts', '.', 'gaf'
'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1985', 'to', 'acquire', 'union',
'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently', 'offered', 'three', 'billion', 'd
'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor', '>,', 'a', 'chicago', 'manufacturer'
'of', 'plastics', 'and', 'chemicals', '.', 'another', 'industry', 'powerhouse', ',', 'w', '.
'r', '.', 'grace', '&', 'lt', ';', 'gra', '>', 'has', 'divested', 'its', 'retailing', ',',
'restaurant', 'and', 'fertilizer', 'businesses', 'to', 'raise', 'cash', 'for', 'chemical',
'acquisitions', '.', 'but', 'some', 'experts', 'worry', 'that', 'the', 'chemical', 'industry

```
 'may', 'be', 'headed', 'for', 'trouble', 'if', 'companies', 'continue', 'turning', 'their',
 'back', 'on', 'the', 'manufacturing', 'of', 'staple', 'petrochemical', 'commodities', ',', 's
 'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profitable', 'specialty', 'chemicals',
 'that', 'are', 'custom', '-', 'designed', 'for', 'a', 'small', 'group', 'of', 'buyers', '.',
 'companies', 'like', 'dupont', '&', 'lt', ';', 'dd', '>', 'and', 'monsanto', 'co', '&', 'lt'
 'mtc', '>', 'spent', 'the', 'past', 'two', 'or', 'three', 'years', 'trying', 'to', 'get', 'ou
 'of', 'the', 'commodity', 'chemical', 'business', 'in', 'reaction', 'to', 'how', 'badly', 'th
 'market', 'had', 'deteriorated', ',"', 'dosher', 'said', '.', '"', 'but', 'i', 'think', 'they
 'will', 'eventually', 'kill', 'the', 'margins', 'on', 'the', 'profitable', 'chemicals', 'in'
 'the', 'niche', 'market', '."', 'some', 'top', 'chemical', 'executives', 'share', 'the',
 'concern', '.', '"', 'the', 'challenge', 'for', 'our', 'industry', 'is', 'to', 'keep', 'from
 'getting', 'carried', 'away', 'and', 'repeating', 'past', 'mistakes', ',"', 'gaf', "'", 's',
 'heyman', 'cautioned', '.', '"', 'the', 'shift', 'from', 'commodity', 'chemicals', 'may', 'be
 'ill', '-', 'advised', '.', 'specialty', 'businesses', 'do', 'not', 'stay', 'special', 'long
 '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'created', 'this', 'month', 'by', 'th
 'sterling', 'investment', 'banking', 'group', ',', 'believes', 'it', 'can', 'generate', '700
 'mln', 'dlrs', 'in', 'annual', 'sales', 'by', 'bucking', 'the', 'industry', 'trend', '.',
 'chairman', 'gordon', 'cain', ',', 'who', 'previously', 'led', 'a', 'leveraged', 'buyout', 'd
 'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical', 'business', ',', 'has', 'spent',
 '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'buy', 'seven', 'petrochemical', 'plan
 'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'plants', 'produce', 'only', 'basic',
 'commodity', 'petrochemicals', 'that', 'are', 'the', 'building', 'blocks', 'of', 'specialty'
 'products', '.', '"', 'this', 'kind', 'of', 'commodity', 'chemical', 'business', 'will', 'nev
 'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'business', ',"', 'cain', 'said', ',',
 'adding', 'that', 'demand', 'is', 'expected', 'to', 'grow', 'by', 'about', 'three', 'pct',
 'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'with', 'dean', 'witter', 'reynolds'
 'said', 'chemical', 'makers', 'have', 'also', 'benefitted', 'by', 'increasing', 'demand', 'fo
 'plastics', 'as', 'prices', 'become', 'more', 'competitive', 'with', 'aluminum', ',', 'wood'
 'and', 'steel', 'products', '.', 'armen', 'estimated', 'the', 'upturn', 'in', 'the', 'chemica
 'business', 'could', 'last', 'as', 'long', 'as', 'four', 'or', 'five', 'years', ',', 'provide
 'the', 'u', '.', 's', '.', 'economy', 'continues', 'its', 'modest', 'rate', 'of', 'growth',
 '<END>'],
['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'solve', 'dispute', 'turkey', 'said',
 'today', 'its', 'disputes', 'with', 'greece', ',', 'including', 'rights', 'on', 'the',
 'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',', 'should', 'be', 'solved', 'through
 'negotiations', '.', 'a', 'foreign', 'ministry', 'statement', 'said', 'the', 'latest', 'crisi
 'between', 'the', 'two', 'nato', 'members', 'stemmed', 'from', 'the', 'continental', 'shelf'
 'dispute', 'and', 'an', 'agreement', 'on', 'this', 'issue', 'would', 'effect', 'the', 'securi
 ',', 'economy', 'and', 'other', 'rights', 'of', 'both', 'countries', '.', '"', 'as', 'the',
 'issue', 'is', 'basicly', 'political', ',', 'a', 'solution', 'can', 'only', 'be', 'found', 'b
 'bilateral', 'negotiations', ',"', 'the', 'statement', 'said', '.', 'greece', 'has', 'repeate
 'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'be', 'solved', 'at', 'the',
 'international', 'court', 'of', 'justice', '.', 'the', 'two', 'countries', 'approached', 'arm
 'confrontation', 'last', 'month', 'after', 'greece', 'announced', 'it', 'planned', 'oil',
 'exploration', 'work', 'in', 'the', 'aegean', 'and', 'turkey', 'said', 'it', 'would', 'also'
 'search', 'for', 'oil', '.', 'a', 'face', '-', 'off', 'was', 'averted', 'when', 'turkey',
 'confined', 'its', 'research', 'to', 'territorrial', 'waters', '.', '"', 'the', 'latest',
 'crises', 'created', 'an', 'historic', 'opportunity', 'to', 'solve', 'the', 'disputes', 'betw
```

```
 'the', 'two', 'countries', ',"', 'the', 'foreign', 'ministry', 'statement', 'said', '.', 'tu
 "'", 's', 'ambassador', 'in', 'athens', ',', 'nazmi', 'akiman', ',', 'was', 'due', 'to', 'me
 'prime', 'minister', 'andreas', 'papandreou', 'today', 'for', 'the', 'greek', 'reply', 'to',
 'message', 'sent', 'last', 'week', 'by', 'turkish', 'prime', 'minister', 'turgut', 'ozal', '
 'the', 'contents', 'of', 'the', 'message', 'were', 'not', 'disclosed', '.', '<END>']]
```

1.2.3 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 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.

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

```
In [4]: 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,
                    num_corpus_words (integer): number of distinct words across the corpus
            """
            corpus_words = []
            num_corpus_words = -1

            # ------------------
            # Write your implementation here.
            corpus_words = set()
            num_corpus_words = 0

            for article in corpus:
                for word in article:
                    if word not in corpus_words:
                        corpus_words.add(word)
                        num_corpus_words+=1

            corpus_words = sorted(list(corpus_words))
            # ------------------

            return corpus_words, num_corpus_words

In [5]: # ----------------------
        # Run this sanity check
        # Note that this not an exhaustive check for correctness.
        # ----------------------

        # Define toy corpus
```

```
        test_corpus = ["START All that glitters isn't gold END".split(" "), "START All's well t
        test_corpus_words, num_corpus_words = distinct_words(test_corpus)

        # Correct answers
        ans_test_corpus_words = sorted(list(set(["START", "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 words.

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

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

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

1.2.4 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 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. If you're not familiar with NumPy, there's a NumPy tutorial in the second half of this cs231n Python NumPy tutorial.

```
In [6]: def compute_co_occurrence_matrix(corpus, window_size=4):
            """ Compute co-occurrence matrix for the given corpus and window_size (default of

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

                      For example, if we take the document "START All that glitters is not gol
                      "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 (numpy matrix of shape (number of unique words in the corpus , number of
                        Co-occurence matrix of word counts.
                        The ordering of the words in the rows/columns should be the same as th
                    word2Ind (dict): dictionary that maps word to index (i.e. row/column numbe
            """
```

```
    words, num_words = distinct_words(corpus)
    M = None
    word2Ind = {}

    # ------------------
    # Write your implementation here.
    word2Ind = dict(zip(words, range(len(words))))

    M = np.zeros((num_words, num_words))

    for article in corpus:
        article_len = len(article)
        for t in range(article_len):
            wt_index = word2Ind[article[t]]
            for j in range(-window_size, window_size+1):
                if j == 0 or t+j < 0 or t+j >= article_len:
                    continue
                wj_index = word2Ind[article[t+j]]
                M[wt_index][wj_index] += 1

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

    return M, word2Ind
```

In [7]:
```
# ---------------------
# 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 = ["START All that glitters isn't gold END".split(" "), "START All's well t
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., 1., 0., 0., 0., 0., 1., 0.,],
     [0., 0., 0., 1., 0., 0., 0., 0., 0., 1.,],
     [0., 0., 0., 0., 0., 0., 1., 0., 0., 1.,],
     [1., 1., 0., 0., 0., 0., 0., 0., 0., 0.,],
     [0., 0., 0., 0., 0., 0., 0., 0., 1., 1.,],
     [0., 0., 0., 0., 0., 0., 0., 1., 1., 0.,],
     [0., 0., 1., 0., 0., 0., 0., 1., 0., 0.,],
     [0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,],
     [1., 0., 0., 0., 1., 1., 0., 0., 0., 1.,],
     [0., 1., 1., 0., 1., 0., 0., 0., 1., 0.,]]
)
word2Ind_ans = {'All': 0, "All's": 1, 'END': 2, 'START': 3, 'ends': 4, 'glitters': 5,
```

```
        # Test correct word2Ind
        assert (word2Ind_ans == word2Ind_test), "Your word2Ind is incorrect:\nCorrect: {}\nYou

        # 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 matrix

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

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

1.2.5 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 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.

```
In [8]: def reduce_to_k_dim(M, k=2):
            """ Reduce a co-occurence count matrix of dimensionality (num_corpus_words, num_co
                to a matrix of dimensionality (num_corpus_words, k) using the following SVD fu
                    - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.T

                Params:
                    M (numpy matrix of shape (number of unique words in the corpus , number of
```

```
            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
                    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, random_state=42)
    M_reduced = svd.fit_transform(M)
    # ------------------

    print("Done.")
    return M_reduced
```

In [9]:
```
# ---------------------
# Run this sanity check
# Note that this 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 = ["START All that glitters isn't gold END".split(" "), "START All's well t
M_test, word2Ind_test = compute_co_occurrence_matrix(test_corpus, window_size=1)
M_test_reduced = reduce_to_k_dim(M_test, k=2)
print(M_test_reduced.shape)

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

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

```
Running Truncated SVD over 10 words...
Done.
(10, 2)
--------------------------------------------------------------------------------
Passed All Tests!
--------------------------------------------------------------------------------
```

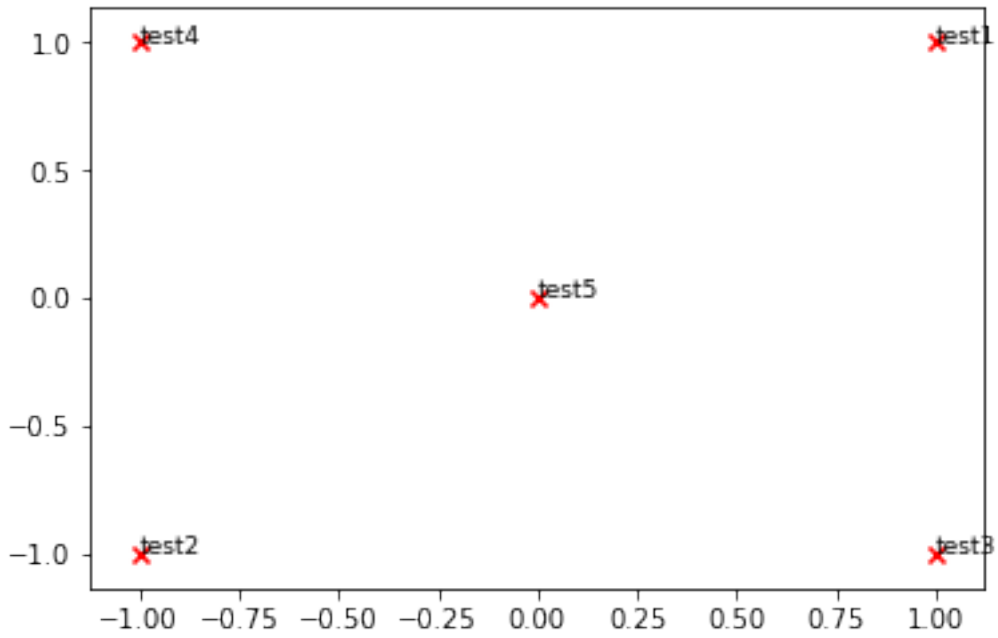
1.2.6 Question 1.4: Implement `plot_embeddings` [code] (1 point)

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

```
In [10]: def plot_embeddings(M_reduced, word2Ind, words):
             """ Plot in a scatterplot the embeddings of the words specified in the list "word
                 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
             """

             # ------------------
             # Write your implementation here.
             types = words
             xy_coords = [M_reduced[word2Ind[word]] for word in words]

             for i,type in enumerate(types):
                 x = xy_coords[i][0]
                 y = xy_coords[i][1]
                 plt.scatter(x, y, marker='x', color='red')
                 plt.text(x, y, type, fontsize=9)
             plt.show()

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

In [11]: # ----------------------
         # Run this sanity check
         # Note that this 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:



Test Plot Solution

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

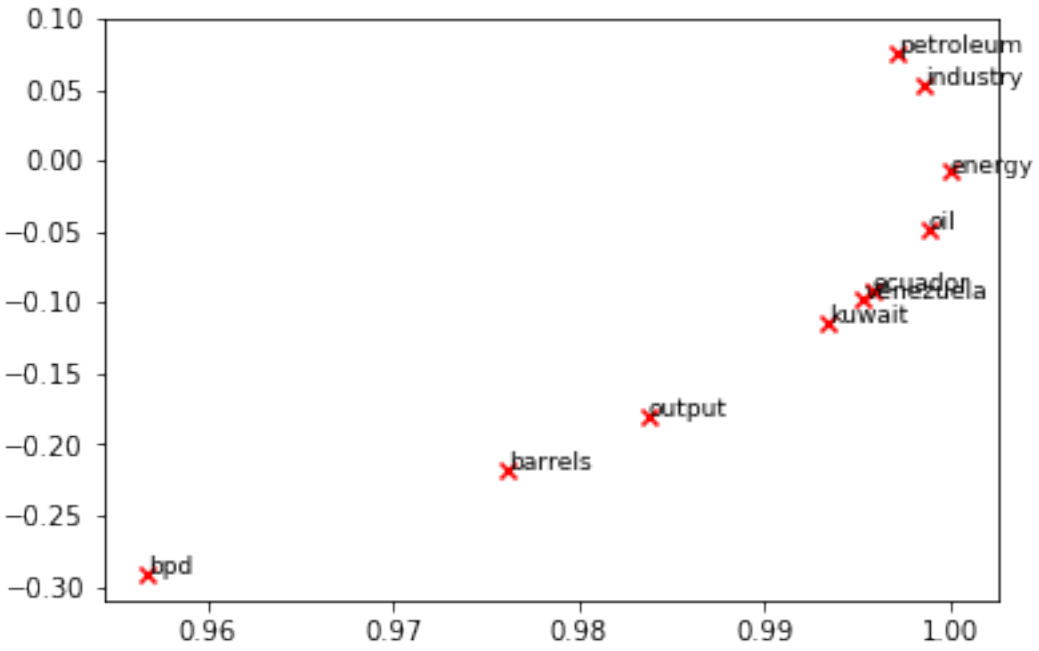
Now we will put together all the parts you have written! We will compute the co-occurrence matrix with fixed window of 5, over the Reuters "crude" corpus. Then we will use TruncatedSVD to compute 2-dimensional embeddings of each word. TruncatedSVD returns U*S, so we normalize the returned vectors, so 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. What clusters together in 2-dimensional embedding space? What doesn't cluster together that you might think should have? **Note:** "bpd" stands for "barrels per day" and is a commonly used abbreviation in crude oil topic articles.

```
In [14]: # ----------------------------
         # 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'
plot_embeddings(M_normalized, word2Ind_co_occurrence, words)
```

```
Running Truncated SVD over 8185 words...
Done.
```



Write your answer here. Two important characteristics about the above clustering are: * We have taken only two components from the SVD decomposition. Hence, all clustering information present here is based on these. * Since we normalize the embeddings vectors, only direction is preserved. In other words, given the two dimensions, we preserve only the relative distribution of the word's similarity with the two semantic concepts expressed by the two dimensions. (Note that even though we started with a 2D vector, the new representation has a single degree of freedom). The clustering graph without normalizing the two words gives a much different picture.

Due to just one degree of freedom, we can expect the embedding to not perform that well. This would usually be reflected in un-expected words getting clustered, but that is that the case with the word list we have.

What clusters together in 2-dimensional embedding space?

- In context of *crude* sector *petroleum* and *industry* are quite synonymous and they get clustered together pretty nicely.
- Similarly, all the countries are clustered together pretty well.

What doesn't cluster together that you might think should have?

- I would have expected *oil* and *petroleum* to cluster together. But *oil* is centrally located between petroleum industry cluster and countries clusters. That can be attributed to the fact that it is equally related to both the clusters.
- *bpd* and *barrels* are not clustered together though it seems they should be.
- Looking at the nearby words to *bpd*, it looks like it is clustered with words denoting production of oil. For example: 'produces', 'firms', 'inventories'.
- Barrels, on the other hand, is used in contexts other than production and supply. An extreme case of this is that the word *warships* is close to *barrels*, where it might refer to the tube of warships and guns.
- It is surprising though that *output* does not cluster with bpd. Perhaps it too is used in few other contexts than just oil production (which is very possible).

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

As discussed in class, more recently prediction-based word vectors have come into fashion, e.g. word2vec. Here, we shall explore the embeddings produced by word2vec. Please revisit the class notes and lecture slides for more details on the word2vec algorithm. If you're feeling adventurous, challenge yourself and try reading the original paper.

Then run the following cells to load the word2vec vectors into memory. **Note**: This might take several minutes.

```
In [15]: def load_word2vec():
             """ Load Word2Vec Vectors
                 Return:
                     wv_from_bin: All 3 million embeddings, each lengh 300
             """
             import gensim.downloader as api
             wv_from_bin = api.load("word2vec-google-news-300")
             vocab = list(wv_from_bin.vocab.keys())
             print("Loaded vocab size %i" % len(vocab))
             return wv_from_bin
```

```
In [16]: # -----------------------------------
         # Run Cell to Load Word Vectors
         # Note: This may take several minutes
         # -----------------------------------
         wv_from_bin = load_word2vec()
```

```
Loaded vocab size 3000000
```

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 follow the Piazza instructions, as how to run remotely on Stanford Farmshare machines.

1.3.1 Reducing dimensionality of Word2Vec Word Embeddings

Let's directly compare the word2vec embeddings to those of the co-occurrence matrix. Run the following cells to:

1. Put the 3 million word2vec vectors into a matrix M
2. Run reduce_to_k_dim (your Truncated SVD function) to reduce the vectors from 300-dimensional to 2-dimensional.

```
In [17]: def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd', 'ecuador', '
             """ Put the word2vec vectors into a matrix M.
                 Param:
                     wv_from_bin: KeyedVectors object; the 3 million word2vec vectors loaded f
                 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.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:
                 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
```

In [18]:
```
# -----------------------------------------------------------------
# Run Cell to Reduce 300-Dimensinal Word Embeddings to k Dimensions
# Note: This may take several minutes
# -----------------------------------------------------------------
M, word2Ind = get_matrix_of_vectors(wv_from_bin)
M_reduced = reduce_to_k_dim(M, k=2)
```

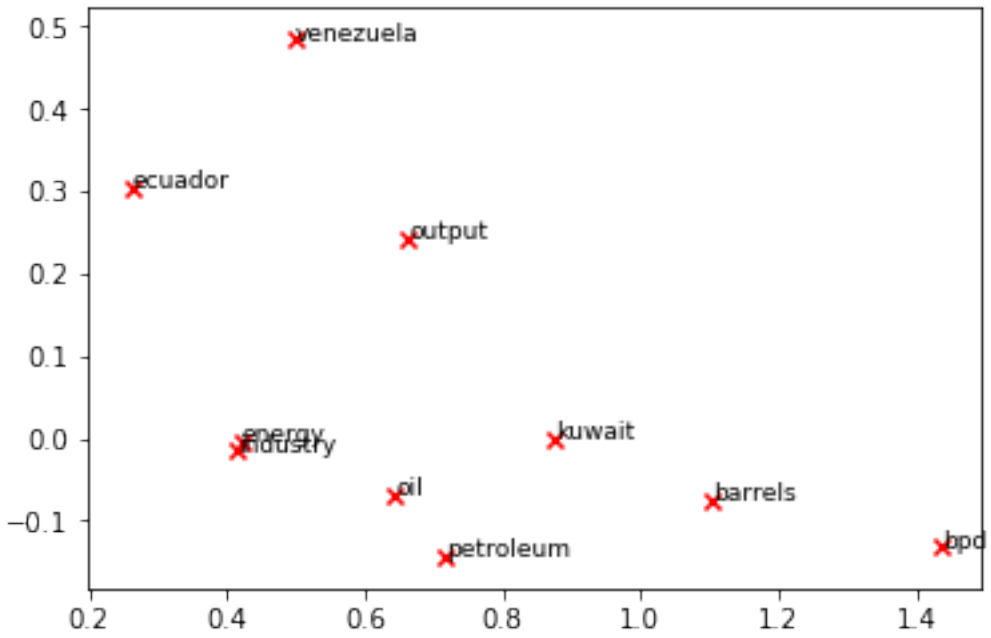
```
Shuffling words ...
Putting 10000 words into word2Ind and matrix M...
Done.
Running Truncated SVD over 10010 words...
Done.
```

1.3.2 Question 2.1: Word2Vec Plot Analysis [written] (4 points)

Run the cell below to plot the 2D word2vec embeddings for `['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output', 'petroleum', 'venezuela']`.

What clusters together in 2-dimensional embedding space? What doesn't cluster together that you might think should have? How is the plot different from the one generated earlier from the co-occurrence matrix?

In [19]:
```
words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'output'
plot_embeddings(M_reduced, word2Ind, words)
```



Write your answer here.

How is the plot different from the one generated earlier from the co-occurrence matrix?

- First major difference is that the word embeddnig here isn't normalized. So since the word vectors have two degrees of freedom, thereby creating a bit richer representation.
- Secondly, the corpus here is different from the *reuters' crude* corpus used earlier. Word-pairs here are extraced from a significantly wider set of topics. So closeness of vectors here can have a much different meaning.
- Further, we perform dimensionality reduction to look at just two dimensions. So the two dimensions here can be much different compared to the two dimensions in the previous analysis.

What clusters together in 2-dimensional embedding space?

- oil and petroleum are pretty close by. Kuwait (one of the largest oil producers) is pretty neat as well.
- barrels and bpd are much closer. They will be much closer than what is represented in the graph above if we take only directional similarity. This can be a result of the corpus covering much wider array of topics and thereby not being as stern about the intricate details of the *crude* corpus.

What doesn't cluster together that you might think should have?

- The countries are spread apart much farther. This can be explained from the fact that these countries are similar if you just look from *crude* oil perspective. But that won't be the case with such wider corpus.

1.3.3 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:

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

1.3.4 Question 2.2: Polysemous Words (2 points) [code + written]

Find a polysemous word (for example, "leaves" or "scoop") such that the top-10 most similar words (according to cosine similarity) contains related words from *both* meanings. For example, "leaves" has both "vanishes" and "stalks" in the top 10, and "scoop" has both "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous words before you find one. Please state the polysemous word you discover and the multiple meanings that occur in the top 10. Why do you think many of the polysemous words you tried didn't work?

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**.

```
In [20]: # ------------------
         # Write your polysemous word exploration code here.

         polysemous_words = ["lie", "march", "fly"]

         for w in polysemous_words:
             print("-"*80)
             print("Polysemous words: ", w)
             print("-"*80)
             for similar_word, score in wv_from_bin.most_similar(w):
                 print("%20s | %20s"%(similar_word, score))
             print("-"*80, "\n")

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

```
--------------------------------------------------------------------------------
Polysemous words:  lie
--------------------------------------------------------------------------------
                lies |    0.6923316717147827
               lying |    0.6025736331939697
               Lying |    0.5922049283981323
  Terravista_complex |    0.5337908267974854
                 lay |    0.5245625376701355
                 Lie |   0.49543923139572144
    perjure_yourself |   0.46970558166503906
                 sit |   0.46696919202804565
      BE_TRUTHFUL_Do |   0.46605420112609863
                lurk |   0.45586875081062317
--------------------------------------------------------------------------------

--------------------------------------------------------------------------------
Polysemous words:  march
--------------------------------------------------------------------------------
            marching |    0.7582396864891052
             marches |    0.7539257407188416
             marched |    0.7141172289848328
            marchers |    0.6616493463516235
```

```
          procession |   0.6252826452255249
               rally |   0.6107629537582397
     protest_marches |   0.5781099796295166
              parade |   0.5777097344398499
             protest |   0.5737200975418091
  protesters_marched |    0.549041748046875
--------------------------------------------------------------------------------


--------------------------------------------------------------------------------
Polysemous words:  fly
--------------------------------------------------------------------------------
              flying |   0.6795353889465332
                flew |   0.6271739602088928
               flies |   0.6233823299407959
                flys |   0.6012992858886719
               flown |    0.567908525466919
                 Fly |   0.5646632313728333
              flight |   0.5158863067626953
Nash_dribbled_upcourt |   0.5138448476791382
     Hylira_dry_skin |  0.49056074023246765
                fl_y |  0.47577083110809326
--------------------------------------------------------------------------------
```

Write your answer here.

1.3.5 Polysemous words

Please state the polysemous word you discover and the multiple meanings that occur in the top 10. Word: lie Multiple meanings: * resting on a supporting horizontal surface. * an intentionally false statement.

Why do you think many of the polysemous words you tried didn't work? Words tried: march, fly

- march:

1. walk in a military manner with a regular measured tread.
2. The month of march.

- fly:

1. move through the air using wings.
2. insect.

Explanation for missing word meanings * In the top similar words for *march*, none of them seem similar to the second meaning (the month of march) * In the top similar words for *fly*, none of them

seem related to the second meaning (insect). Still it is not obvious whether the word *flies* refers to *insect* or just the third person singular form of the first meaning.

Possible reasons include: 1. Due to bias in the corpus due to which it includes more data related to one meaning compared to others, the vector will shift towards that meaning. 2. As a corollary to #1, the other meanings of the words may be present, but not in the top 10 words.

1.3.6 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 simply 1 - Cosine Similarity.

Find three words (w1,w2,w3) where w1 and w2 are synonyms and w1 and w3 are antonyms, but Cosine Distance(w1,w3) < Cosine 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 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.

```
In [21]: # ------------------
         # Write your synonym & antonym exploration code here.

         w1 = "tall"
         w2 = "short"
         w3 = "huge"
         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))

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

```
Synonyms tall, short have cosine distance: 0.7756051597013258
Antonyms tall, huge have cosine distance: 0.8364604184144997
```

Write your answer here. The way we define similarity is that two words w1 and w2 are similar if they have similar usage pattern in word formation.

Though "long" and "huge" are pretty related, but they have larger distance compared to "long" and "short". This is because "long" and "short" have more probability of being used in similar context compared to "long" and "huge".

For examples, take the sentence "That guy is tall". There is more probability of getting a similar sentence "That guy is short" used in similar context, compared to "That guy is huge".

1.3.7 Solving Analogies with Word Vectors

Word2Vec vectors have been shown to *sometimes* exhibit the ability to solve analogies.

As an example, for the analogy "man : king :: woman : x", what is x?

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

Note: Further Documentation on the `most_similar` function can be found within the **GenSim documentation**.

```
In [22]: # 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', 0.5902431607246399),
 ('crown_prince', 0.5499460697174072),
 ('prince', 0.5377321243286133),
 ('kings', 0.5236844420433044),
 ('Queen_Consort', 0.5235945582389832),
 ('queens', 0.5181134343147278),
 ('sultan', 0.5098593235015869),
 ('monarchy', 0.5087411999702454)]
```

1.3.8 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). 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!

```
In [23]: # ------------------
         # Write your analogy exploration code here.

         pprint.pprint(wv_from_bin.most_similar(positive=["cry", "walked"], negative=["walk"]))

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

```
[('cried', 0.6157843470573425),
 ('crying', 0.6023385524749756),
 ('cries', 0.5490766763687134),
 ('screamed', 0.5411455035209656),
 ('bawling', 0.5122770071029663),
 ('sobbing', 0.5035713911056519),
 ('shouted', 0.5021365880966187),
 ('wailed', 0.50178062915802),
 ('pompom_fiasco', 0.4938657879829407),
 ('sobbed', 0.49248096346855164)]
```

Write your answer here. As discussed during the lecture, the difference vector tries to get some relationship. Here "walked" - "walk" gives the past tense relationship. This when added to "cry" gives "cried" at the top.

An interesting observation I found was that it even preserves whether the first letter is capital in some cases. In this case, one can verify that the following holds true: * walk : walked :: cry : x cried * walk : walked :: Cry : x Cried

1.3.9 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 intended analogy in the form x:y :: a:b, and state the (incorrect) value of b according to the word vectors.

```
In [24]: # ------------------
         # Write your incorrect analogy exploration code here.

         pprint.pprint(wv_from_bin.most_similar(positive=["pakistan", "rome"], negative=["italy

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

```
[('kashmir', 0.5378997325897217),
 ('india', 0.5142123103141785),
 ('delhi', 0.5132304430007935),
 ('lahore', 0.509597897529602),
 ('modi', 0.4979272484779358),
 ('kasab', 0.4943981170654297),
 ('iran', 0.4932347238063812),
 ('karachi', 0.4916754961013794),
 ('sonia', 0.48575571179389954),
 ('pakistani', 0.48445770144462585)]
```

Write your answer here. Correct capital guess: * italy:rome :: france:x --> paris

Bad capital guess: * italy:rome :: pakistan:x --> kashmir (correct answer 'islamabad' is not present in the top 10. It is actually at position 67). * italy:rome :: india:x --> chennai (correct answer 'delhi' is at the second position).

The intented analogy in each case is to get the capital city. But that is our own interpretation. The relationship vector that we get from substraction may refer to something else for the model. For example, it may refer to a tourism spot, in which case Paris for France is still a good guess, and Kashmir for Pakistan will also be a good guess.

In other words, capital city of a country might not have the same significance for each country. Hence, its usage pattern with the country's name will differ, thereby affecting the relationship vector expressed by the delta of their corresponding embeddings.

1.3.10 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 to our word embeddings.

Run the cell below, to examine (a) which terms are most similar to "woman" and "boss" and most dissimilar to "man", and (b) which terms are most similar to "man" and "boss" and most dissimilar to "woman". What do you find in the top 10?

```
In [25]: # Run this cell
         # Here `positive` indicates the list of words to be similar to and `negative` indicat
         # most dissimilar from.
         pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'boss'], negative=['man']))
         print()
         pprint.pprint(wv_from_bin.most_similar(positive=['man', 'boss'], negative=['woman']))
```

```
[('bosses', 0.5522644519805908),
 ('manageress', 0.49151360988616943),
 ('exec', 0.45940813422203064),
 ('Manageress', 0.45598435401916504),
 ('receptionist', 0.4474116563796997),
 ('Jane_Danson', 0.44480544328689575),
 ('Fiz_Jennie_McAlpine', 0.44275766611099243),
 ('Coronation_Street_actress', 0.44275566935539246),
 ('supremo', 0.4409853219985962),
 ('coworker', 0.43986251950263977)]

[('supremo', 0.6097398400306702),
 ('MOTHERWELL_boss', 0.5489562153816223),
 ('CARETAKER_boss', 0.5375303626060486),
 ('Bully_Wee_boss', 0.5333974361419678),
 ('YEOVIL_Town_boss', 0.5321705341339111),
 ('head_honcho', 0.5281980037689209),
 ('manager_Stan_Ternent', 0.525971531867981),
 ('Viv_Busby', 0.5256162881851196),
 ('striker_Gabby_Agbonlahor', 0.5250812768936157),
 ('BARNSLEY_boss', 0.5238943099975586)]
```

Write your answer here.

Case 1: boss + woman - man Adding *woman* to *boss* and subtracting *man* seems to be moving the vector in a direction that diminishes the bossness characteristic. For example: the word *receptionist* is close to the resulting vector.

Case 2: boss + man - woman Adding *man* to *boss* and subtracting *woman* seems to be moving the vector in a direction that further exagerates the bossness characteristic. For example: the word head_honcho is close to the resulting vector.

Additionally, while case #1 returns names of actresses, case #2 returns CEOs and managers of different associations.

1.3.11 Question 2.7: Independent Analysis of Bias in Word Vectors [code + written] (2 points)

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.

```
In [26]: # ------------------
         # Write your bias exploration code here.

         pprint.pprint(wv_from_bin.most_similar(positive=["doctor", "woman"], negative=["man"])

         # ------------------a
```

```
[('gynecologist', 0.7093892097473145),
 ('nurse', 0.647728681564331),
 ('doctors', 0.6471461057662964),
 ('physician', 0.64389967918396),
 ('pediatrician', 0.6249487996101379),
 ('nurse_practitioner', 0.6218312978744507),
 ('obstetrician', 0.6072014570236206),
 ('ob_gyn', 0.5986712574958801),
 ('midwife', 0.5927063226699829),
 ('dermatologist', 0.5739566683769226)]
```

Write your answer here. There is a inherent gender bias in occupation categories.

"doctor" + "woman" - "man" : The set of words we get (top 10) are: * gynaecologist * nurse * doctors * physician * nurse_practitioner * obstetrician * ob_gyn * midwife * dermatologist

"doctor" + "man" - "woman" : The set of words we get (top 10) are: * physician * doctors * surgeon * cardiologist * neurologist * neurosurgeon * urologist * Doctor * internist

Clearly the second set of words denote that the resulting word vector in second case moves to a more acclaimed set of categories of doctors.

On the other hand, case #1 has "nurse", "nurse_practisioner", "midwife" which show that the resulting word vector has moved to a direction where the cluster does not contain the same category of doctors thereby exemplifying the gender bias present in these embeddings.

Additionally one may notice that some of the categories of doctors that are present in case #1 are doctors that mostly help women. So it also possible that gynaecologist, obstetrician, obgyn and dermatologist are present in the nearby words in case #1 because women visit these categories of doctors more in a similar sense as men visit a doctor.

1.3.12 Question 2.8: Thinking About Bias [written] (1 point)

What might be the cause of these biases in the word vectors?

Write your answer here. The words embeddings do not create any new information. They just summarise the information that was already present in the input corpora. Word2Vec was trained on GoogleNews articles. Clearly these articles publish facts and express opinions that have inherent gender bias.

This does not mean GoogleNews corpus is the one at fault. But in today's society itself, we will see more articles about men doctor and women as housewifes etc. Further, these articles have

been using 'he' pronouns to refer to doctors in general (which has improved in the recent days). Since 'he' - 'she' is similar to 'man'-'woman', these kind of biases further add to it.

2 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
5. Look at the PDF file and make sure all your solutions are there, displayed correctly. The PDF is the only thing your graders will see!
6. Submit your PDF on Gradescope.