Anthony Galczak - HW1

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 find many provided codes in the notebook. We highly encourage you to read and understand the provided codes as part of the learning :-)

In [184]:
```
# 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 /home/anthony/nltk_data...
[nltk_data]   Package reuters is already up-to-date!
```

Word Vectors

Word Vectors are often used as a fundamental component for downstream NLP tasks, e.g. question answering, text generation, translation, etc., so it is important to build some intuitions as to their strengths and weaknesses. Here, you will explore two types of word vectors: those derived from *co-occurrence matrices*, and those derived via *GloVe*.

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 (https://en.wikipedia.org/wiki/Word_embedding) states, "*conceptually it involves a mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension*".

Part 1: Count-Based Word Vectors (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 (https://en.wikipedia.org/wiki/John_Rupert_Firth))

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 (http://web.stanford.edu/class/cs124/lec/vectorsemantics.video.pdf) or here (https://medium.com/data-science-group-iitr/word-embedding-2d05d270b285)).

Co-Occurrence

A co-occurrence matrix counts how often things co-occur in some environment. Given some word w_i occurring in the document, we consider the *context window* surrounding w_i. Supposing our fixed window size is n, then this is the n preceding and n subsequent words in that document, i.e. words $w_{i-n} \dots w_{i-1}$ and $w_{i+1} \dots w_{i+n}$. We build a *co-occurrence matrix* M, which is a symmetric word-by-word matrix in which M_{ij} is the number of times w_j appears inside w_i's window among all documents.

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

Document 1: "all that glitters is not gold"

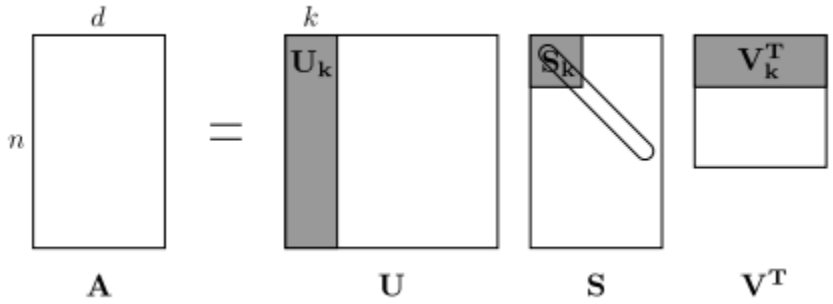
Document 2: "all is well that ends well"

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

Note: In NLP, we often add `<START>` and `<END>` tokens to represent the beginning and end of sentences, paragraphs or documents. In 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 (https://davetang.org/file/Singular_Value_Decomposition_Tutorial.pdf). If you want to learn more thoroughly about PCA or SVD, feel free to check out lectures 7 (https://web.stanford.edu/class/cs168/l/l7.pdf), 8 (http://theory.stanford.edu/~tim/s15/l/l8.pdf), and 9 (https://web.stanford.edu/class/cs168/l/l9.pdf) 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 (https://en.wikipedia.org/wiki/Singular_value_decomposition#Truncated_SVD) — then there are reasonably scalable techniques to compute those iteratively.

Plotting Co-Occurrence Word Embeddings

Here, we will be using the Reuters (business and financial news) corpus. If you haven't run the import cell at the top of this page, please run it now (click it and press SHIFT-RETURN). The corpus consists of 10,788 news documents totaling 1.3 million words. These documents span 90 categories and are split into train and test. For more details, please see https://www.nltk.org/book/ch02.html (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 to perform any other kind of pre-processing.

In [185]:
```
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_TO
```

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

In [186]:
```
reuters_corpus = read_corpus()
pprint.pprint(reuters_corpus[:3], compact=True, width=100)
```

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'demand',
'downwards', 'the',
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'miti',
')', 'will', 'revise',
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outlook', 'b
y', 'august', 'to',
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'demand',
',', 'ministry',
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'the', 'pr
ojection', 'for',
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '550', 'm
ln', 'kilolitres',
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the', 'decis
ion', 'follows',
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'industr
y', 'following',
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'declin
e', 'in', 'domestic',
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'work',
'out', 'a', 'revised',
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberations', 'o
f', 'committee',
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and', 'ener
gy', ',', 'the',
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'review',
'the', 'breakdown',
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nuclear',
',', 'coal', 'and',
  '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', 'c
heap', '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', 'thi
s', 'year', ',',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'major',
'company',
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'chemica
l', 'manufacturing',
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'businesses',
'has', 'prompted',
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'cain',
'chemical', 'inc',
  ',', 'and', 'other', 'firms', 'to', 'aggressively', 'seek', 'acquisitions',
'of', 'petrochemical',
  'plants', '.', 'oil', 'companies', 'such', 'as', 'ashland', 'oil', 'inc',
'&', 'lt', ';', 'ash',
  '>,', 'the', 'kentucky', '-', 'based', 'oil', 'refiner', 'and', 'marketer',
```

```
',', 'are', 'also',
  'shopping', 'for', 'money', '-', 'making', 'petrochemical', 'businesses', 't
o', 'buy', '.', '"',
  'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'of', 'a', 'golden', 'p
eriod', ,"', 'said',
  'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow', 'chemical', 'co',
'&', 'lt', ';',
  'dow', '>,', 'adding', ',', '"', 'there', "'", 's', 'no', 'major', 'plant',
'capacity', 'being',
  'added', 'around', 'the', 'world', 'now', '.', 'the', 'whole', 'game', 'is',
'bringing', 'out',
  'new', 'products', 'and', 'improving', 'the', 'old', 'ones', '."', 'analyst
s', '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', '9
0', 'pct',
  'capacity', ',', 'reflecting', 'tighter', 'supply', 'that', 'could', 'hike',
'product', 'prices',
  'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'said', 'john', 'dosher',
',', 'managing',
  'director', 'of', 'pace', 'consultants', 'inc', 'of', 'houston', '.', 'deman
d', 'for', 'some',
  'products', 'such', 'as', 'styrene', 'could', 'push', 'profit', 'margins', 'u
p', 'by', 'as',
  'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreffice', ',', 'speakin
g', 'at', 'a',
  'meeting', 'of', 'chemical', 'engineers', 'in', 'houston', ',', 'said', 'do
w', 'would', 'easily',
  'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'last', 'year', 'and', 'p
redicted', 'it',
  'would', 'have', 'the', 'best', 'year', 'in', 'its', 'history', '.', 'in', '1
985', ',', 'when',
  'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs', 'a', 'barrel', 'an
d', 'chemical',
  'exports', 'were', 'adversely', 'affected', 'by', 'the', 'strong', 'u', '.',
's', '.', 'dollar',
  ',', 'dow', 'had', 'profits', 'of', '58', 'mln', 'dlrs', '.', '"', 'i', 'beli
eve', 'the',
  'entire', 'chemical', 'industry', 'is', 'headed', 'for', 'a', 'record', 'yea
r', 'or', 'close',
  'to', 'it', ,"', 'oreffice', 'said', '.', 'gaf', 'chairman', 'samuel', 'heym
an', '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', 'leap',
  'from', '1985', '.', 'the', 'turn', 'in', 'the', 'fortunes', 'of', 'the', 'on
ce', '-', 'sickly',
  'chemical', 'industry', 'has', 'been', 'brought', 'about', 'by', 'a', 'combin
ation', 'of', 'luck',
  'and', 'planning', ',', 'said', 'pace', "'", 's', 'john', 'dosher', '.', 'dos
her', 'said', 'last',
  'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made', 'feedstocks', 'drama
tically', 'cheaper',
  'and', 'at', 'the', 'same', 'time', 'the', 'american', 'dollar', 'was', 'weak
```

```
ening', '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', 'extr
a', 'chemical',
  'manufacturing', 'capacity', 'created', 'by', 'middle', 'eastern', 'oil', 'pr
oducers', 'in',
  'the', 'early', '1980s', '.', 'finally', ',', 'virtually', 'all', 'major',
'u', '.', 's', '.',
  'chemical', 'manufacturers', 'have', 'embarked', 'on', 'an', 'extensive', 'co
rporate',
  'restructuring', 'program', 'to', 'mothball', 'inefficient', 'plants', ',',
'trim', 'the',
  'payroll', 'and', 'eliminate', 'unrelated', 'businesses', '.', 'the', 'restru
cturing', 'touched',
  'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile', 'takeover', 'attemp
ts', '.', 'gaf', ',',
  'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1985', 'to', 'acquir
e', 'union',
  'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently', 'offered', 'thre
e', 'billion', 'dlrs',
  'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor', '>,', 'a', 'chicago',
'manufacturer',
  'of', 'plastics', 'and', 'chemicals', '.', 'another', 'industry', 'powerhous
e', ',', 'w', '.',
  'r', '.', 'grace', '&', 'lt', ';', 'gra', '>', 'has', 'divested', 'its', 'ret
ailing', ',',
  'restaurant', 'and', 'fertilizer', 'businesses', 'to', 'raise', 'cash', 'fo
r', 'chemical',
  'acquisitions', '.', 'but', 'some', 'experts', 'worry', 'that', 'the', 'chemi
cal', 'industry',
  'may', 'be', 'headed', 'for', 'trouble', 'if', 'companies', 'continue', 'turn
ing', 'their',
  'back', 'on', 'the', 'manufacturing', 'of', 'staple', 'petrochemical', 'commo
dities', ',', 'such',
  'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profitable', 'specialt
y', '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', 'out',
  'of', 'the', 'commodity', 'chemical', 'business', 'in', 'reaction', 'to', 'ho
w', 'badly', 'the',
  'market', 'had', 'deteriorated', ,"', 'dosher', 'said', '.', '"', 'but',
'i', 'think', 'they',
  'will', 'eventually', 'kill', 'the', 'margins', 'on', 'the', 'profitable', 'c
hemicals', 'in',
  'the', 'niche', 'market', '."', 'some', 'top', 'chemical', 'executives', 'sha
re', 'the',
  'concern', '.', '"', 'the', 'challenge', 'for', 'our', 'industry', 'is', 't
o', 'keep', 'from',
  'getting', 'carried', 'away', 'and', 'repeating', 'past', 'mistakes', ,"',
'gaf', "'", 's',
  'heyman', 'cautioned', '.', '"', 'the', 'shift', 'from', 'commodity', 'chemic
als', 'may', 'be',
  'ill', '-', 'advised', '.', 'specialty', 'businesses', 'do', 'not', 'stay',
'special', 'long',
  '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'created', 'this', 'm
```

```
onth', 'by', 'the',
  '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', 'leverage
d', 'buyout', 'of',
  'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical', 'business', ',',
'has', 'spent', '1',
  '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'buy', 'seven', 'petro
chemical', 'plants',
  'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'plants', 'produce', 'o
nly', 'basic',
  'commodity', 'petrochemicals', 'that', 'are', 'the', 'building', 'blocks', 'o
f', 'specialty',
  'products', '.', '"', 'this', 'kind', 'of', 'commodity', 'chemical', 'busines
s', 'will', 'never',
  'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'business', ,"', 'cain',
'said', ',',
  'adding', 'that', 'demand', 'is', 'expected', 'to', 'grow', 'by', 'about', 't
hree', 'pct',
  'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'with', 'dean', 'witt
er', 'reynolds', ',',
  'said', 'chemical', 'makers', 'have', 'also', 'benefitted', 'by', 'increasin
g', 'demand', 'for',
  'plastics', 'as', 'prices', 'become', 'more', 'competitive', 'with', 'aluminu
m', ',', 'wood',
  'and', 'steel', 'products', '.', 'armen', 'estimated', 'the', 'upturn', 'in',
'the', 'chemical',
  'business', 'could', 'last', 'as', 'long', 'as', 'four', 'or', 'five', 'year
s', ',', 'provided',
  'the', 'u', '.', 's', '.', 'economy', 'continues', 'its', 'modest', 'rate',
'of', 'growth', '.',
  '<END>'],
 ['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'solve', 'dispute', 't
urkey', 'said',
  'today', 'its', 'disputes', 'with', 'greece', ',', 'including', 'rights', 'o
n', 'the',
  'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',', 'should', 'be', 's
olved', 'through',
  'negotiations', '.', 'a', 'foreign', 'ministry', 'statement', 'said', 'the',
'latest', 'crisis',
  'between', 'the', 'two', 'nato', 'members', 'stemmed', 'from', 'the', 'contin
ental', 'shelf',
  'dispute', 'and', 'an', 'agreement', 'on', 'this', 'issue', 'would', 'effec
t', 'the', 'security',
  ',', 'economy', 'and', 'other', 'rights', 'of', 'both', 'countries', '.',
'"', 'as', 'the',
  'issue', 'is', 'basicly', 'political', ',', 'a', 'solution', 'can', 'only',
'be', 'found', 'by',
  'bilateral', 'negotiations', ,"', 'the', 'statement', 'said', '.', 'greece',
'has', 'repeatedly',
  'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'be', 'solved', 'at',
'the',
  'international', 'court', 'of', 'justice', '.', 'the', 'two', 'countries', 'a
pproached', 'armed',
  'confrontation', 'last', 'month', 'after', 'greece', 'announced', 'it', 'plan
ned', 'oil',
  'exploration', 'work', 'in', 'the', 'aegean', 'and', 'turkey', 'said', 'it',
'would', 'also',
  'search', 'for', 'oil', '.', 'a', 'face', '-', 'off', 'was', 'averted', 'whe
n', 'turkey',
```

```
  'confined', 'its', 'research', 'to', 'territorrial', 'waters', '.', '"', 'th
e', 'latest',
  'crises', 'created', 'an', 'historic', 'opportunity', 'to', 'solve', 'the',
'disputes', 'between',
  'the', 'two', 'countries', ',"', 'the', 'foreign', 'ministry', 'statement',
'said', '.', 'turkey',
  "'", 's', 'ambassador', 'in', 'athens', ',', 'nazmi', 'akiman', ',', 'was',
'due', 'to', 'meet',
  'prime', 'minister', 'andreas', 'papandreou', 'today', 'for', 'the', 'greek',
'reply', 'to', 'a',
  'message', 'sent', 'last', 'week', 'by', 'turkish', 'prime', 'minister', 'tur
gut', 'ozal', '.',
  'the', 'contents', 'of', 'the', 'message', 'were', 'not', 'disclosed', '.',
'<END>']]
```

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 (https://coderwall.com/p/rcmaea/flatten-a-list-of-lists-in-one-line-in-python) may be useful to flatten a list of lists. If you're not familiar with Python list comprehensions in general, here's more information (https://python-3-patterns-idioms-test.readthedocs.io/en/latest/Comprehensions.html).

You may find it useful to use Python sets (https://www.w3schools.com/python/python_sets.asp) to remove duplicate words.

In [187]:
```python
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 cor
            num_corpus_words (integer): number of distinct words across the corpu
    """
    corpus_words = []
    num_corpus_words = -1

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

    # List comprehension to flatten the corpus
    corpus_words = [l for word_list in corpus for l in word_list]

    # Removing duplicates via set, then sorting a new list of the set.
    corpus_words = sorted(list(set(corpus_words)))

    num_corpus_words = len(corpus_words)
    # ------------------

    return corpus_words, num_corpus_words
```

In [188]:
```
# ---------------------
# 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
test_corpus_words, num_corpus_words = distinct_words(test_corpus)

# Correct answers
ans_test_corpus_words = sorted([START_TOKEN, "All", "ends", "that", "gold", "All'
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 w

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

# 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 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 (http://cs231n.github.io/python-numpy-tutorial/).

In [189]:

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

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

              For example, if we take the document "<START> All that glitters is
              "All" will co-occur with "<START>", "that", "glitters", "is", and "

        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 c
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should be the same
            word2Ind (dict): dictionary that maps word to index (i.e. row/column
    """
    words, num_words = distinct_words(corpus)
    M = None
    word2Ind = {}

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

    # Initialize M and make the word2Ind dictionary.
    M = np.zeros((num_words, num_words))
    index = 0
    for word in words:
        word2Ind[word] = index
        index += 1

    # Loop over each doc in the corpus
    for doc in corpus:
        doc_len = len(doc)

        # Loop over each doc, incrementing each occurrence of a context word.
        for i in range(doc_len):
            center_word = doc[i]

            left_index = i - 1
            right_index = i + 1

            while left_index >= 0 and left_index > i - window_size - 1:
                M[word2Ind[center_word], word2Ind[doc[left_index]]] += 1
                left_index -= 1

            while right_index < doc_len and right_index < i + window_size + 1:
                M[word2Ind[center_word], word2Ind[doc[right_index]]] += 1
                right_index += 1

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

    return M, word2Ind
```

In [190]:
```
# ---------------------
# 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
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
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: {

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

# 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 m

# 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 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 (https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html)](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html).

In [191]:
```
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_corpus_words, n
        to a matrix of dimensionality (num_corpus_words, k) using the following S
            - http://scikit-learn.org/stable/modules/generated/sklearn.decomposit

        Params:
            M (numpy matrix of shape (number of unique words in the corpus , numb
            k (int): embedding size of each word after dimension reduction
        Return:
            M_reduced (numpy matrix of shape (number of corpus words, k)): matrix
                    In terms of the SVD from math class, this actually returns U
    """
    n_iters = 10     # Use this parameter in your call to `TruncatedSVD`
    M_reduced = None
    print("Running Truncated SVD over %i words..." % (M.shape[0]))

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

    svd = TruncatedSVD(n_components = k, n_iter = n_iters)
    M_reduced = svd.fit_transform(M)

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

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

In [192]:
```
# ---------------------
# 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
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 {}".f
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should have {}"

# 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 Matplotlib (`plt`).

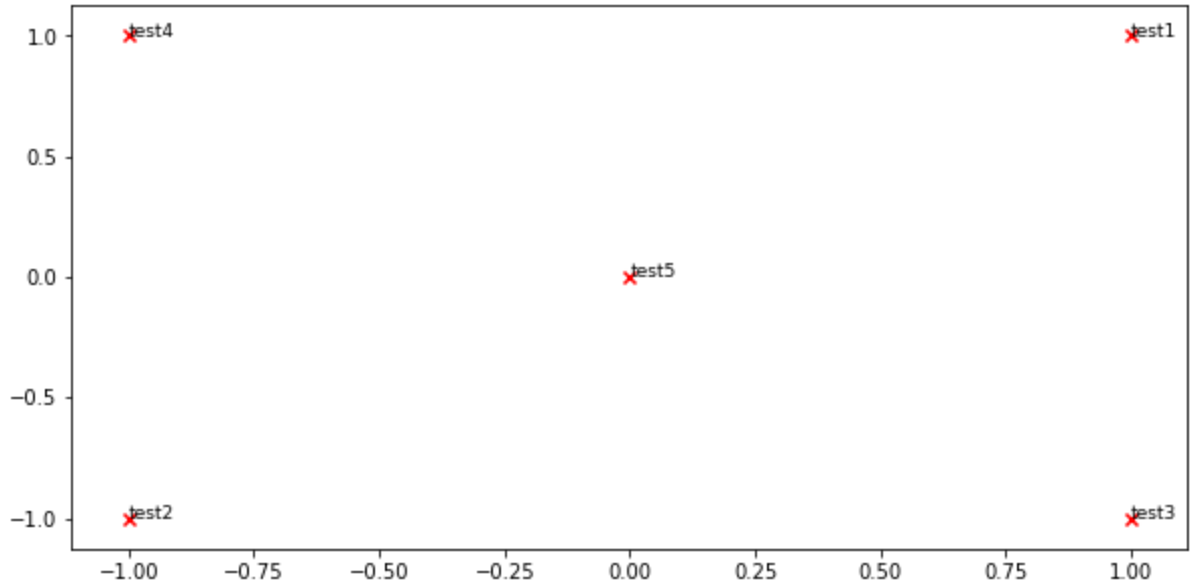
For this example, you may find it useful to adapt this code (https://www.pythonmembers.club/2018/05/08/matplotlib-scatter-plot-annotate-set-text-at-label-each-point/). In the future, a good way to make a plot is to look at the Matplotlib gallery (https://matplotlib.org/gallery/index.html), find a plot that looks somewhat like what you want, and adapt the code they give.

In [193]:

```
def plot_embeddings(M_reduced, word2Ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the list '
        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 corpu
            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.

    # Get the first column as x coords and second column as y coords.
    x_coords = M_reduced[:,0]
    y_coords = M_reduced[:,1]

    for i, type in enumerate(words):
        x = x_coords[i]
        y = y_coords[i]
        plt.scatter(x, y, marker='x', color='red')
        plt.text(x, y, type, fontsize=9)

    plt.show()

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

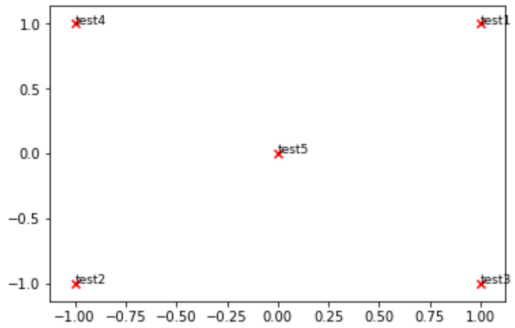
In [194]:
```
# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness.
# The plot produced should look like the "test solution plot" depicted below.
# ---------------------

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:
```




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

Test Plot Solution



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

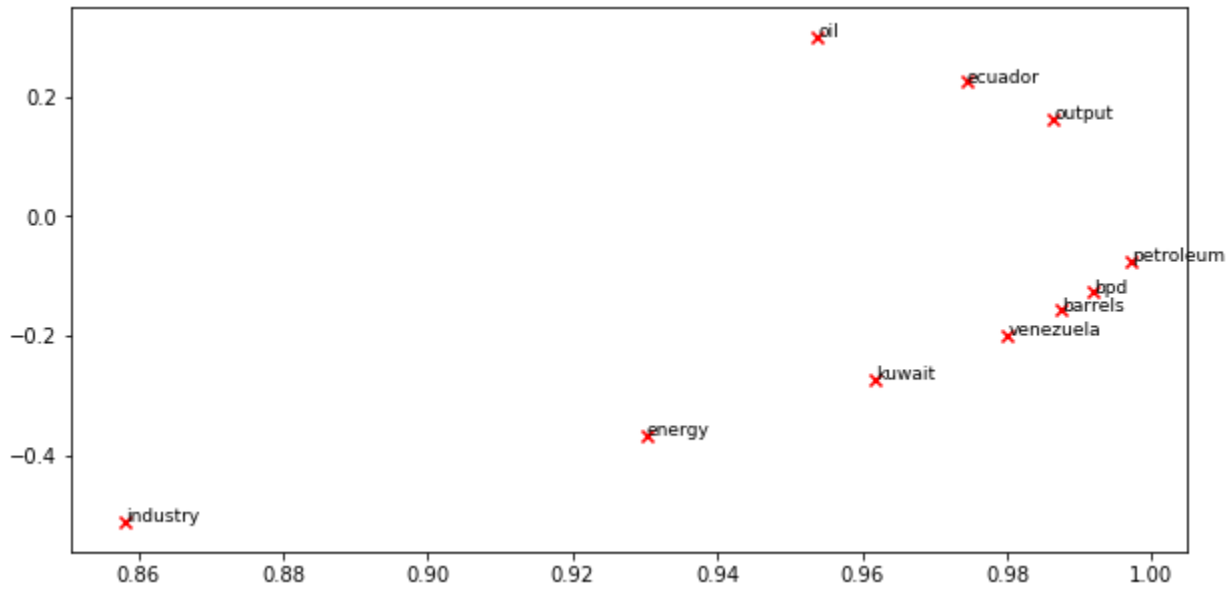
Now we will put together all the parts you have written! We will compute the co-occurrence matrix with fixed window of 4 (the default window size), over the Reuters "crude" (oil) corpus. Then we will use TruncatedSVD to compute 2-dimensional embeddings of each word. TruncatedSVD returns U*S, so we

need to normalize the returned vectors, so that all the vectors will appear around the unit circle (therefore closeness is directional closeness). **Note**: The line of code below that does the normalizing uses the NumPy concept of *broadcasting*. If you don't know about broadcasting, check out Computation on Arrays: Broadcasting by Jake VanderPlas (https://jakevdp.github.io/PythonDataScienceHandbook/02.05-computation-on-arrays-broadcasting.html).

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 [195]:
```
# ----------------------------
# Run This Cell to Produce Your Plot
# ----------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2Ind_co_occurrence = compute_co_occurrence_matrix(reuters_co
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', 'out

plot_embeddings(M_normalized, word2Ind_co_occurrence, words)
```

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



There is a cluster around petroleum, bpd, barrels, and to some extent, Venezuela. The only other thing that could potentially be a cluster is Ecuador and output, but I don't think so. I would expect "oil" to be in a cluster with petroleum and barrels. Industry and energy are related to these words, but they seem so far away. Additionally, it seems like the regions Venezuela, Kuwait, Ecuador should all be in a cluster, but they aren't.

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

As discussed in class, more recently prediction-based word vectors have demonstrated better performance, such as word2vec and GloVe (which also utilizes the benefit of counts). Here, we shall explore the embeddings produced by GloVe. Please revisit the class notes and lecture slides for more details on the

word2vec and GloVe algorithms. If you're feeling adventurous, challenge yourself and try reading [GloVe's original paper (https://nlp.stanford.edu/pubs/glove.pdf)](https://nlp.stanford.edu/pubs/glove.pdf).

Then run the following cells to load the GloVe vectors into memory. **Note**: If this is your first time to run these cells, i.e. download the embedding model, it will take about 15 minutes to run. If you've run these cells before, rerunning them will load the model without redownloading it, which will take about 1 to 2 minutes.

In [196]:
```
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
```

In [197]:
```
# ----------------------------------
# Run Cell to Load Word Vectors
# Note: This will take several minutes
# ----------------------------------
wv_from_bin = load_embedding_model()
```

```
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 running out of memory, we will work with a sample of 10000 GloVe vectors instead. Run the following cells to:

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

In [198]:
```
def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd', 'ecuador
    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loaded fro
        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
```

In [199]:
```
# -----------------------------------------------------------------
# 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 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

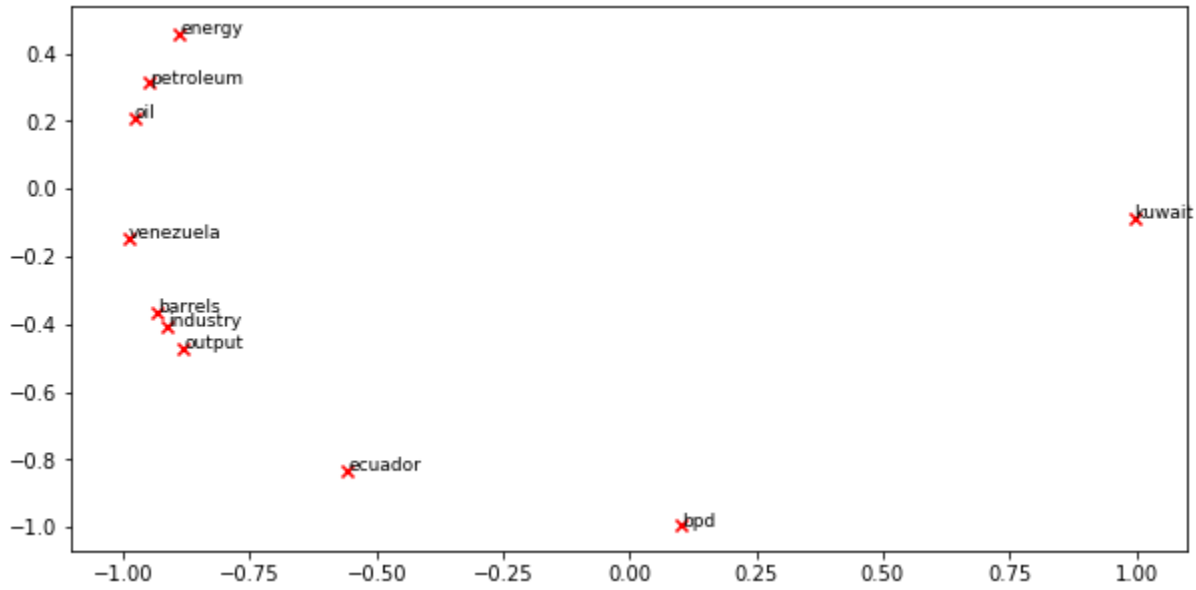
vectors properly. If you still have problems with loading the embeddings onto your local machine after this, please follow the Piazza instructions, as how to run remotely on Stanford Farmshare machines.

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

Run the cell below to plot the 2D GloVe 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? What is a possible reason for causing the difference?

In [200]:
```
words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil', 'out
plot_embeddings(M_reduced_normalized, word2Ind, words)
```

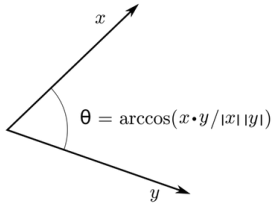


There are clusters around: "energy, petroleum, oil" and "barrels, industry, output". There still isn't a cluster around Venezuela, Ecuador, and Kuwait. This is surprising, but I think it may be a result of the dimensionality reduction. We don't know the true distance for the countries in the high-dimensional space. An alternative idea is that there isn't enough documents to relate the regions together for meaning. Additionally, bpd was left out of the "barrels, industry, output" cluster. However, the clusters make a lot more sense than the co-occurrence matrix plot.

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 (http://mathworld.wolfram.com/L1-Norm.html) and L2 (http://mathworld.wolfram.com/L2-Norm.html) 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 (https://en.wikipedia.org/wiki/Cosine_similarity) s between two vectors p and q is defined as:

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

Question 2.2: Words with Multiple Meanings (2 points) [code + written]

Polysemes and homonyms are words that have more than one meaning (see this wiki page (https://en.wikipedia.org/wiki/Polysemy) to learn more about the difference between polysemes and homonyms). Find a word with at least 2 different meanings such that the top-10 most similar words (according to cosine similarity) contain related words from *both* meanings. For example, "leaves" has both "vanishes" and "stalks" in the top 10, and "scoop" has both "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous or homonymic words before you find one. Please state the word you discover and the multiple meanings that occur in the top 10. Why do you think many of the polysemous or homonymic words you tried didn't work (i.e. the top-10 most similar words only contain **one** of the meanings of the words)?

Note: You should use the `wv_from_bin.most_similar(word)` function to get the top 10 similar words. This function ranks all other words in the vocabulary with respect to their cosine similarity to the given word. For further assistance please check the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvectors.FastTextK**

In [201]:
```
# ------------------
# Write your implementation here.

wv_from_bin.most_similar("bear")


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

Out[201]:
```
[('bears', 0.6849423050880432),
 ('grizzly', 0.6034084558486938),
 ('wolf', 0.590702474117279),
 ('stearns', 0.5704878568649292),
 ('lion', 0.5357851982116699),
 ('bearing', 0.5106834173202515),
 ('dog', 0.5077008605003357),
 ('big', 0.49132290482521057),
 ('bore', 0.4897792339324951),
 ('deer', 0.4877018928527832)]
```

First, I'll discuss examples of some failures. Orange only netted me things like "yellow", "black", "colors", no mention of fruit. "Book" only gave me things about published works like author, wrote, essay. No mention of restaurant reservations. I did find a word that was a good example of a polyseme. "Critical" gave me obvious things like crucial, important, particular, but also gave me "critics". This is a related, but slightly different meaning for critical.

"Bear" gave me 2 distinct meanings in the top 10 similarities. Of course, we have the common definition of a bear, which gave grizzly, bears, lion, etc. But then we have "stearns", which is a very specific word in this context. This refers to Bear Stearns, the financial company that most believe led to the 2008 housing crisis. An interesting word here is "bearing", which has a totally distinct meaning from all the words here, but isn't directly related to bear.

There was a lot of words that just had one meaning in the top 10 similarities. When we are looking at a word and calculating it's distance, we are only putting that word in the space once. Something like "calf" could have 2 distinct representations, i.e. two entries, thus two distances. One would be around "cattle", "cow", "farm", and the other would be around "thigh", "leg". But since we only have one word representation in this space, we only get the physical leg meaning of a calf when getting the top 10 similar words.

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 (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvectors.FastTextK** for further assistance.

In [202]:
```
    # ------------------
    # Write your implementation here.

    # Synonym
    print(wv_from_bin.distance("right", "correct"))

    # Antonym
    print(wv_from_bin.distance("right", "wrong"))


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

```
0.5864298343658447
0.3679667115211487
```

The task given here was very interesting. I interpreted it as: we want to find synonyms that aren't super strong and find antonyms that are often seen with each other. I thought about how often people say "right or wrong", or at least say right and wrong in the same sentence. I don't think "correct" is used quite as often as right is for the same sort of contexts. It's possible that it may be further confusing for the system that right is often seen with a direction like "left". This is an unintuitive result from GloVe, but makes sense giving what sort of information we are feeding into it.

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 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 (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvectors.FastTextK**

In [203]:
```
# Run this cell to answer the analogy -- man : king :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], negative=['mar
```

```
[('queen', 0.6978678703308105),
 ('princess', 0.6081745028495789),
 ('monarch', 0.5889754891395569),
 ('throne', 0.5775108933448792),
 ('prince', 0.5750998258590698),
 ('elizabeth', 0.5463595986366272),
 ('daughter', 0.5399125814437866),
 ('kingdom', 0.5318052172660828),
 ('mother', 0.5168544054031372),
 ('crown', 0.5164473056793213)]
```

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 [204]:
```
    # ------------------
    # Write your implementation here.

    # Dwarf is to small as giant is to large
    pprint.pprint(wv_from_bin.most_similar(positive=['small','giant'],
                                           negative=['dwarf']))

    # This also works...
    # Woman is to man as girl is to boy
    # pprint.pprint(wv_from_bin.most_similar(positive=['man','girl'], negative=['

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

```
[('large', 0.6522384881973267),
 ('big', 0.626799464225769),
 ('huge', 0.6061473488807678),
 ('companies', 0.6018587946891785),
 ('holding', 0.5931851863861084),
 ('company', 0.5874850749969482),
 ('largest', 0.5823867917060852),
 ('a', 0.5609161853790283),
 ('biggest', 0.5529043674468994),
 ('smaller', 0.5460005402565002)]
```

A dwarf, a short person, is to small as giant, a tall person, is to large.

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 [205]:
```
    # ------------------
    # Write your implementation here.

    # Sprinting is to running as jogging is to walking
    # walking is to jogging as running is to sprinting
    pprint.pprint(wv_from_bin.most_similar(positive=['jogging','running'],
                                           negative=['walking']))


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

```
[('ran', 0.5216119289398193),
 ('errands', 0.395122766494751),
 ('run', 0.39265263080596924),
 ('amok', 0.3728637099266052),
 ('clarett', 0.37244486808776855),
 ('line', 0.3719397187232971),
 ('portis', 0.3650514781475067),
 ('sprints', 0.3584376275539398),
 ('scrimmage', 0.35078656673431396),
 ('sprinting', 0.3496767580509186)]
```

I initially did "sprinting is to running as jogging is to walking", and the system did guess walking. However, I think walking is way too common of a word to assume that wouldn't work. So "walking is to jogging as running is to sprinting" works here. Ran is the past-tense of run, not a "faster version of running", which would give sprinting. However, it is interesting that sprints and sprinting is further down the list.

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 embeddings. Bias can be dangerous because it can reinforce stereotypes through applications that employ these models.

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

In [206]:
```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `negative` inc
# most dissimilar from.
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'worker'], negative=['m
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'worker'], negative=['wom
```

```
[('employee', 0.6375863552093506),
 ('workers', 0.6068919897079468),
 ('nurse', 0.5837947130203247),
 ('pregnant', 0.5363885760307312),
 ('mother', 0.5321309566497803),
 ('employer', 0.5127025842666626),
 ('teacher', 0.5099577307701111),
 ('child', 0.5096741914749146),
 ('homemaker', 0.5019455552101135),
 ('nurses', 0.4970571994781494)]

[('workers', 0.611325740814209),
 ('employee', 0.5983108878135681),
 ('working', 0.5615329742431641),
 ('laborer', 0.5442320108413696),
 ('unemployed', 0.5368517637252808),
 ('job', 0.5278826951980591),
 ('work', 0.5223963260650635),
 ('mechanic', 0.5088937282562256),
 ('worked', 0.5054520964622498),
 ('factory', 0.4940453767776489)]
```

Both lists have gender bias in them. The list for woman has things like "teacher", "homemaker", "nurse" and the man list has "mechanic", "laborer", "unemployed". Men are commonly associated with jobs such as mechanic and laborer. This illustrates the importance of looking at our models and analyzing them for bias.

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

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

In [207]:
```
    # ------------------
    # Write your implementation here.

    pprint.pprint(wv_from_bin.most_similar(positive=['boy', 'toy'],
                                           negative=['girl']))

    pprint.pprint(wv_from_bin.most_similar(positive=['girl', 'toy'],
                                           negative=['boy']))

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

```
[('toys', 0.71570885181427),
 ('hasbro', 0.5164632797241211),
 ('robot', 0.4731711149215698),
 ('pet', 0.4670490324497223),
 ('manufacturer', 0.46681636571884155),
 ('mattel', 0.4582391381263733),
 ('lego', 0.4581176638603210 4),
 ('miniature', 0.4441472887992859),
 ('makers', 0.4429824948310852),
 ('manufactured', 0.4427534341812134)]
[('toys', 0.7094953060150146),
 ('doll', 0.5932915210723877),
 ('dolls', 0.570662260055542),
 ('barbie', 0.5407705903053284),
 ('mattel', 0.532855212688446),
 ('accessories', 0.5206909775733948),
 ('hasbro', 0.49227219820022583),
 ('jewelry', 0.47385698556900024),
 ('lego', 0.4690813422203064),
 ('apparel', 0.4613623321056366)]
```

Notice that toys for boys come up with robots, manufactured, makers. Toys for girls comes up with dolls, barbie, jewelry, apparel. There is a clear gender bias in toys here.

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 how bias get into the word vectors. How might you be able to investigate/test these causes?

This embedding is ran on the "wiki-gigaword5" data set. The bulk of this is from news articles and not from Wikipedia. When a particular journalist writes an article about nursing, they are much more likely to include "woman" in the article than "man". I would argue that the same holds true for news articles about toys. It is much more likely that "boy" is included when talking about robot toys, rather than "girl". The fact that bias is inside our embeddings is what sort of data that we are bringing in. My hypothesis is that the more "objective" an incoming corpus is, the less bias will exist.

Investigating the causes of the bias is as simple as performing experiments on a subset of the data. For example, we may want re-run our analysis for just Wikipedia data and see how egregious the bias is. What quantitative effects does adding news articles into our data set do? From there, we can even narrow down specific sources of bias. What if Fox News is a higher source of bias than Reuters? These are the sort of questions I would want to answer when doing an investigation for bias.

An aside on this whole topic is that we need to be clear and up front about bias with our downstream consumers. In other words, it is extremely important to be transparent about what sorts of bias appear in analysis of our data sets. Eliminating bias entirely is likely impossible, but doing the investigation and making it available to the public is a step in the right direction.

Submission Instructions

1. Click the Save button at the top of the Jupyter Notebook.
2. Select Cell -> All Output -> Clear. This will clear all the outputs from all cells (but will keep the content of all cells).
3. Select Cell -> Run All. This will run all the cells in order, and will take several minutes.
4. Once you've rerun everything, select File -> Download as -> PDF via LaTeX (If you have trouble using "PDF via LaTex", you can also save the webpage as pdf. Make sure all your solutions especially the coding parts are displayed in the pdf, it's okay if the provided codes get cut off because lines are not wrapped in code cells).
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.

In [ ]: