Assignment 1: MLPs and Backpropagation

Due on November 7th, 2024 (23:59:59)

Welcome to Comp541: Deep Learning Course!

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

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

In [127]:
```python
# 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
[nltk_data]     C:\Users\Dell\AppData\Roaming\nltk_data...
[nltk_data]   Package reuters is already up-to-date!
```

Word Vectors

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

Note on Terminology: The terms "word vectors" and "word embeddings" are often used interchangeably. The term "embedding" refers to the fact that we are encoding aspects of a word's meaning in a lower dimensional space. As Wikipedia (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 $w_{i-n} \ldots w_{i-1}$ and $w_{i+1} \ldots w_{i+n}$. 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.
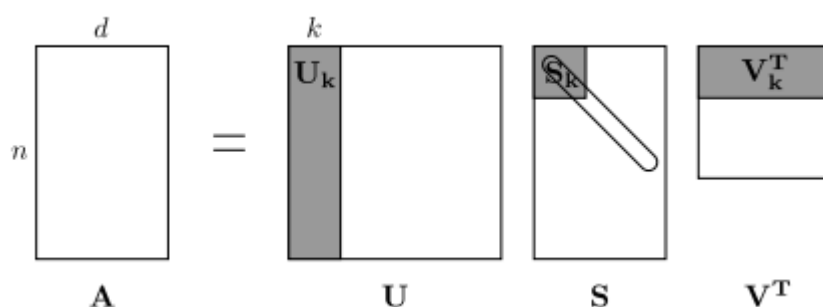
This reduced-dimensionality co-occurrence representation preserves semantic relationships between words, e.g. *doctor* and *hospital* will be closer than *doctor* and *dog*.

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)](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)](https://web.stanford.edu/class/cs168/l/l7.pdf), [8 (http://theory.stanford.edu/~tim/s15/l/l8.pdf)](http://theory.stanford.edu/~tim/s15/l/l8.pdf), and [9 (https://web.stanford.edu/class/cs168/l/l9.pdf)](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)](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)](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.

```python
In [128]: 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))] + [
```

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

```
In [129]: reuters_corpus = read_corpus()
          pprint.pprint(reuters_corpus[:3], compact=True, width=100)
```

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'dema
nd', 'downwards', 'the',
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'mit
i', ')', 'will', 'revise',
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outloo
k', 'by', 'august', 'to',
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'deman
d', ',', 'ministry',
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'th
e', 'projection', 'for',
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '55
0', 'mln', 'kilolitres',
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the',
'decision', 'follows',
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'in
dustry', 'following',
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'de
cline', 'in', 'domestic',
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'wor
k', 'out', 'a', 'revised',
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberation
s', 'of', 'committee',
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and',
'energy', ',', 'the',
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'revie
w', 'the', 'breakdown',
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nucl
ear', ',', 'coal', 'and',
  'natural', 'gas', '.', 'nuclear', 'energy', 'provided', 'the', 'bulk',
'of', 'japan', '"', 's',
  'electric', 'power', 'in', 'the', 'fiscal', 'year', 'ended', 'march', '3
1', ',', 'supplying',
  'an', 'estimated', '27', 'pct', 'on', 'a', 'kilowatt', '/', 'hour', 'bas
is', ',', 'followed',
  'by', 'oil', '(', '23', 'pct', ')', 'and', 'liquefied', 'natural', 'ga
s', '(', '21', 'pct', '),',
  'they', 'noted', '.', '<END>'],
 ['<START>', 'energy', '/', 'u', '.', 's', '.', 'petrochemical', 'industr
y', 'cheap', 'oil',
  'feedstocks', ',', 'the', 'weakened', 'u', '.', 's', '.', 'dollar', 'an
d', 'a', 'plant',
  'utilization', 'rate', 'approaching', '90', 'pct', 'will', 'propel', 'th
e', 'streamlined', 'u',
  '.', 's', '.', 'petrochemical', 'industry', 'to', 'record', 'profits',
'this', 'year', ',',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'ma
jor', 'company',
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'ch
emical', 'manufacturing',
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'busines
ses', 'has', 'prompted',
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'c
ain', 'chemical', 'inc',
  ',', 'and', 'other', 'firms', 'to', 'aggressively', 'seek', 'acquisition
s', 'of', 'petrochemical',
  'plants', '.', 'oil', 'companies', 'such', 'as', 'ashland', 'oil', 'in
c', '&', 'lt', ';', 'ash',
  '>,', 'the', 'kentucky', '-', 'based', 'oil', 'refiner', 'and', 'markete
r', ',', 'are', 'also',
  'shopping', 'for', 'money', '-', 'making', 'petrochemical', 'businesse
s', 'to', 'buy', '.', '"',
```

```
       'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'of', 'a', 'golde
      n', 'period', ',"', 'said',
       'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow', 'chemical',
      'co', '&', 'lt', ';',
       'dow', '>,', 'adding', ',', '"', 'there', "'", 's', 'no', 'major', 'plan
      t', 'capacity', 'being',
       'added', 'around', 'the', 'world', 'now', '.', 'the', 'whole', 'game',
      'is', 'bringing', 'out',
       'new', 'products', 'and', 'improving', 'the', 'old', 'ones', '."', 'anal
      ysts', 'say', 'the',
       'chemical', 'industry', "'", 's', 'biggest', 'customers', ',', 'automobi
      le', 'manufacturers',
       'and', 'home', 'builders', 'that', 'use', 'a', 'lot', 'of', 'paints', 'a
      nd', 'plastics', ',',
       'are', 'expected', 'to', 'buy', 'quantities', 'this', 'year', '.', 'u',
      '.', 's', '.',
       'petrochemical', 'plants', 'are', 'currently', 'operating', 'at', 'abou
      t', '90', 'pct',
       'capacity', ',', 'reflecting', 'tighter', 'supply', 'that', 'could', 'hi
      ke', 'product', 'prices',
       'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'said', 'john', 'dos
      her', ',', 'managing',
       'director', 'of', 'pace', 'consultants', 'inc', 'of', 'houston', '.', 'd
      emand', 'for', 'some',
       'products', 'such', 'as', 'styrene', 'could', 'push', 'profit', 'margin
      s', 'up', 'by', 'as',
       'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreffice', ',', 'sp
      eaking', 'at', 'a',
       'meeting', 'of', 'chemical', 'engineers', 'in', 'houston', ',', 'said',
      'dow', 'would', 'easily',
       'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'last', 'year', 'an
      d', 'predicted', 'it',
       'would', 'have', 'the', 'best', 'year', 'in', 'its', 'history', '.', 'i
      n', '1985', ',', 'when',
       'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs', 'a', 'barrel',
      'and', 'chemical',
       'exports', 'were', 'adversely', 'affected', 'by', 'the', 'strong', 'u',
      '.', 's', '.', 'dollar',
       ',', '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', 'rep
      ort', 'a', '20', 'pct',
       'gain', 'in', 'profits', 'during', '1987', '.', 'last', 'year', ',', 'th
      e', 'domestic',
       'industry', 'earned', 'a', 'total', 'of', '13', 'billion', 'dlrs', ',',
      'a', '54', 'pct', 'leap',
       'from', '1985', '.', 'the', 'turn', 'in', 'the', 'fortunes', 'of', 'th
      e', 'once', '-', 'sickly',
       'chemical', 'industry', 'has', 'been', 'brought', 'about', 'by', 'a', 'c
      ombination', 'of', 'luck',
       'and', 'planning', ',', 'said', 'pace', "'", 's', 'john', 'dosher', '.',
      'dosher', 'said', 'last',
       'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made', 'feedstocks',
      'dramatically', 'cheaper',
       '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', 'dema
nd', 'into', 'balance',
   'has', 'been', 'the', 'gradual', 'market', 'absorption', 'of', 'the', 'e
xtra', 'chemical',
   'manufacturing', 'capacity', 'created', 'by', 'middle', 'eastern', 'oi
l', 'producers', 'in',
   'the', 'early', '1980s', '.', 'finally', ',', 'virtually', 'all', 'majo
r', 'u', '.', 's', '.',
   'chemical', 'manufacturers', 'have', 'embarked', 'on', 'an', 'extensiv
e', 'corporate',
   'restructuring', 'program', 'to', 'mothball', 'inefficient', 'plants',
',', 'trim', 'the',
   'payroll', 'and', 'eliminate', 'unrelated', 'businesses', '.', 'the', 'r
estructuring', 'touched',
   'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile', 'takeover', 'a
ttempts', '.', 'gaf', ',',
   'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1985', 'to', 'a
cquire', 'union',
   'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently', 'offered', 't
hree', 'billion', 'dlrs',
   'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor', '>,', 'a', 'chic
ago', 'manufacturer',
   'of', 'plastics', 'and', 'chemicals', '.', 'another', 'industry', 'power
house', ',', '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', ',', 'such',
   'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profitable', 'speci
alty', 'chemicals',
   'that', 'are', 'custom', '-', 'designed', 'for', 'a', 'small', 'group',
'of', 'buyers', '.', '"',
   'companies', 'like', 'dupont', '&', 'lt', ';', 'dd', '>', 'and', 'monsan
to', 'co', '&', 'lt', ';',
   'mtc', '>', 'spent', 'the', 'past', 'two', 'or', 'three', 'years', 'tryi
ng', 'to', 'get', 'out',
   'of', 'the', 'commodity', 'chemical', 'business', 'in', 'reaction', 't
o', 'how', 'badly', 'the',
   'market', 'had', 'deteriorated', ',"', 'dosher', 'said', '.', '"', 'bu
t', 'i', 'think', 'they',
   'will', 'eventually', 'kill', 'the', 'margins', 'on', 'the', 'profitabl
e', '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', 'c
hemicals', 'may', 'be',
   'ill', '-', 'advised', '.', 'specialty', 'businesses', 'do', 'not', 'sta
y', 'special', 'long',
   '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'created', 'thi
s', 'month', 'by', 'the',
```

```
   'sterling', 'investment', 'banking', 'group', ',', 'believes', 'it', 'ca
n', 'generate', '700',
   'mln', 'dlrs', 'in', 'annual', 'sales', 'by', 'bucking', 'the', 'industr
y', 'trend', '.',
   'chairman', 'gordon', 'cain', ',', 'who', 'previously', 'led', 'a', 'lev
eraged', 'buyout', 'of',
   'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical', 'business',
',', 'has', 'spent', '1',
   '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'buy', 'seven',
'petrochemical', 'plants',
   'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'plants', 'produc
e', 'only', 'basic',
   'commodity', 'petrochemicals', 'that', 'are', 'the', 'building', 'block
s', 'of', 'specialty',
   'products', '.', '"', 'this', 'kind', 'of', 'commodity', 'chemical', 'bu
siness', 'will', 'never',
   'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'business', ',"', 'c
ain', 'said', ',',
   'adding', 'that', 'demand', 'is', 'expected', 'to', 'grow', 'by', 'abou
t', 'three', 'pct',
   'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'with', 'dean',
'witter', 'reynolds', ',',
   'said', 'chemical', 'makers', 'have', 'also', 'benefitted', 'by', 'incre
asing', 'demand', 'for',
   'plastics', 'as', 'prices', 'become', 'more', 'competitive', 'with', 'al
uminum', ',', 'wood',
   'and', 'steel', 'products', '.', 'armen', 'estimated', 'the', 'upturn',
'in', 'the', 'chemical',
   'business', 'could', 'last', 'as', 'long', 'as', 'four', 'or', 'five',
'years', ',', 'provided',
   'the', 'u', '.', 's', '.', 'economy', 'continues', 'its', 'modest', 'rat
e', 'of', 'growth', '.',
   '<END>'],
 ['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'solve', 'disput
e', 'turkey', 'said',
   'today', 'its', 'disputes', 'with', 'greece', ',', 'including', 'right
s', 'on', 'the',
   'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',', 'should', 'b
e', 'solved', 'through',
   'negotiations', '.', 'a', 'foreign', 'ministry', 'statement', 'said', 't
he', 'latest', 'crisis',
   'between', 'the', 'two', 'nato', 'members', 'stemmed', 'from', 'the', 'c
ontinental', 'shelf',
   'dispute', 'and', 'an', 'agreement', 'on', 'this', 'issue', 'would', 'ef
fect', 'the', 'security',
   ',', 'economy', 'and', 'other', 'rights', 'of', 'both', 'countries',
'.', '"', 'as', 'the',
   'issue', 'is', 'basicly', 'political', ',', 'a', 'solution', 'can', 'onl
y', 'be', 'found', 'by',
   'bilateral', 'negotiations', ',"', 'the', 'statement', 'said', '.', 'gre
ece', 'has', 'repeatedly',
   'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'be', 'solved',
'at', 'the',
   'international', 'court', 'of', 'justice', '.', 'the', 'two', 'countrie
s', 'approached', 'armed',
   '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', 'th
e', 'disputes', 'between',
      'the', 'two', 'countries', ',"', 'the', 'foreign', 'ministry', 'statemen
t', 'said', '.', 'turkey',
      "'", 's', 'ambassador', 'in', 'athens', ',', 'nazmi', 'akiman', ',', 'wa
s', 'due', 'to', 'meet',
      'prime', 'minister', 'andreas', 'papandreou', 'today', 'for', 'the', 'gr
eek', 'reply', 'to', 'a',
      'message', 'sent', 'last', 'week', 'by', 'turkish', 'prime', 'minister',
'turgut', '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).

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

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

```python
In [130]: def distinct_words(corpus):
              """ Determine a list of distinct words for the corpus.
                  Params:
                      corpus (list of list of strings): corpus of documents
                  Return:
                      corpus_words (list of strings): sorted list of distinct words ac
                      num_corpus_words (integer): number of distinct words across the
              """
              corpus_words = []
              num_corpus_words = -1

              # ------------------
              # Write your implementation here.
              flattened_corpus = [y for x in corpus for y in x]
              corpus_words= {word for word in flattened_corpus}
              corpus_words=sorted(corpus_words)
              num_corpus_words=len(corpus_words)

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

              return corpus_words, num_corpus_words
```

In [131]:
```python
# --------------------
# 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_
test_corpus_words, num_corpus_words = distinct_words(test_corpus)

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

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

# 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.

```python
In [132]: def compute_co_occurrence_matrix(corpus, window_size=4):
              """ Compute co-occurrence matrix for the given corpus and window_size (c

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

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

                  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
                          Co-occurence matrix of word counts.
                          The ordering of the words in the rows/columns should be the
                      word2ind (dict): dictionary that maps word to index (i.e. row/c
              """
              words, num_words = distinct_words(corpus)
              M = None
              word2ind = {}

              # ------------------
              # Write your implementation here.
              M=np.zeros((num_words,num_words))
              word2ind = {word: idx for idx, word in enumerate(words)}

              for sentence in corpus:

                  for idx, w in enumerate(sentence):
                      f_idx = idx - window_size if idx - window_size >= 0 else 0
                      l_idx = idx + window_size if idx + window_size < len(sentence) e
                      window_words=sentence[f_idx:l_idx]
                      for word in window_words:
                          if word != w:

                              M[word2ind[w], word2ind[word]] += 1
                              M[word2ind[word], word2ind[w]] += 1


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

              return M, word2ind
```

In [133]:
```python
# --------------------
# 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_
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_si:

# 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",
word2ind_ans = dict(zip(ans_test_corpus_words, range(len(ans_test_corpus_wo)

# Test correct word2ind
assert (word2ind_ans == word2ind_test), "Your word2ind is incorrect:\nCorre(

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

# 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 ({}, {})=({}, {})

# 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).

```python
In [134]: def reduce_to_k_dim(M, k=2):
              """ Reduce a co-occurence count matrix of dimensionality (num_corpus_wor
                  to a matrix of dimensionality (num_corpus_words, k) using the follow
                      - http://scikit-learn.org/stable/modules/generated/sklearn.decon

                  Params:
                      M (numpy matrix of shape (number of unique words in the corpus ,
                      k (int): embedding size of each word after dimension reduction
                  Return:
                      M_reduced (numpy matrix of shape (number of corpus words, k)): m
                              In terms of the SVD from math class, this actually retur
              """
              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 [135]:
```python
# --------------------
# 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_
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_si
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
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should hav

# 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 (http://web.archive.org/web/20190924160434/https://www.pythonmembers.club/2018/05/08/m 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.

```python
In [136]: def plot_embeddings(M_reduced, word2ind, words):
              """ Plot in a scatterplot the embeddings of the words specified in the l
                  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
                      word2ind (dict): dictionary that maps word to indices for matri>
                      words (list of strings): words whose embeddings we want to visua
              """

              # ------------------
              # Write your implementation here.
              idxs=[word2ind[word] for word in words]
              x=[M_reduced[idx,0] for idx in idxs]
              y=[M_reduced[idx,1] for idx in idxs]

              for i,word in enumerate(words):
                plt.scatter(x[i],y[i],marker='x',color='red')
                plt.text(x[i],y[i],word,fontsize=12)


              # ------------------
```
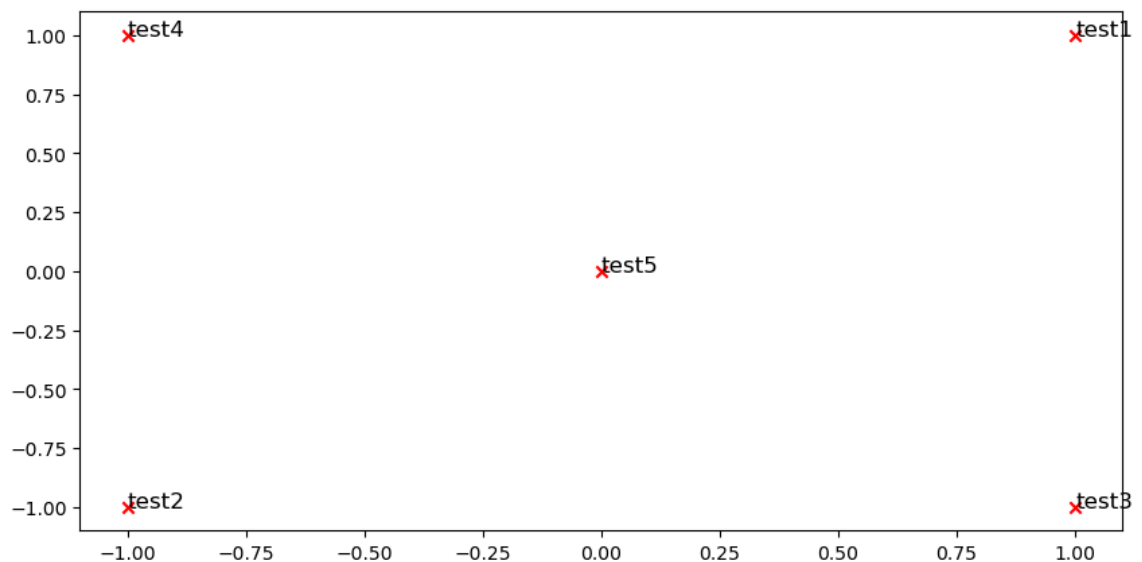
```
In [137]:  # ---------------------
           # Run this sanity check
           # Note that this is not an exhaustive check for correctness.
           # The plot produced should look like the "test solution plot" depicted below
           # ---------------------

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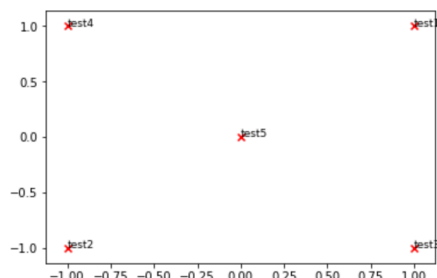
```
--------------------------------------------------------------------------------
------
Outputted Plot:
--------------------------------------------------------------------------------
------
```


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)](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 [138]:
```python
# ------------------------------
# Run This Cell to Produce Your Plot
# ------------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2ind_co_occurrence = compute_co_occurrence_matrix(reute
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] # broadcas


words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil',

plot_embeddings(M_normalized, word2ind_co_occurrence, words)
```

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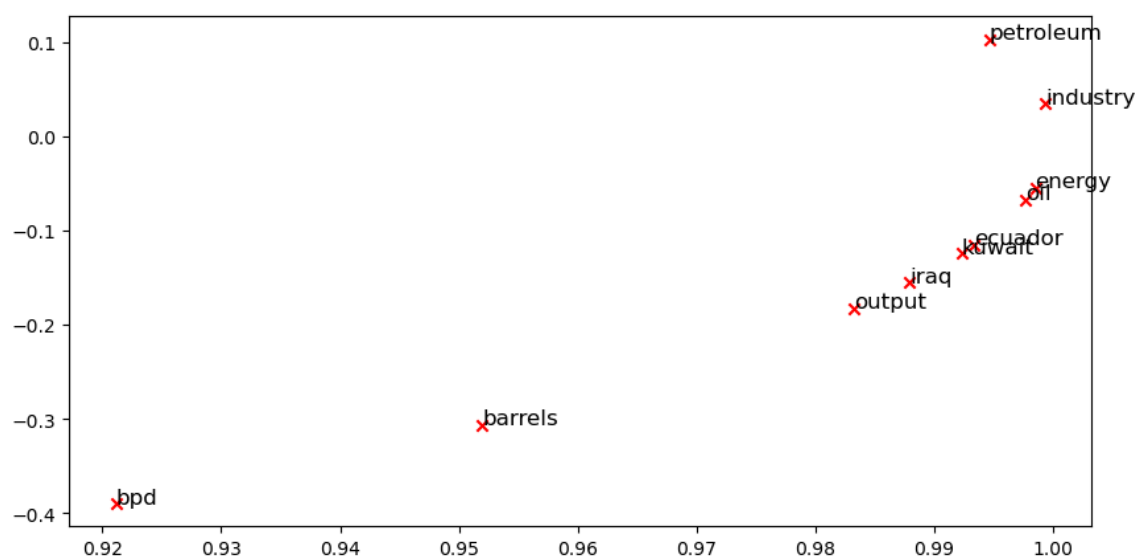

Write your answer here.

roughly we can say there are 3 clusters: cluster 1: bpd cluster 2: barrels cluster 3: rest of the words. I expect barrels and bpd in the same cluster because the meaning of bpd is barrels per day.

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).

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

```python
In [139]: 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
```

```python
In [140]: # ---------------------------------
          # Run Cell to Load Word Vectors
          # Note: This will take a couple minutes
          # ---------------------------------
          wv_from_bin = load_embedding_model()
```

```
Loaded vocab size 400000
```

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

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 [141]:
```python
def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd', 'ec
    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loade
        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 [142]:
```python
# --------------------------------------------------------------------
# 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 go to office hours or contact course staff.

Question 2.1: GloVe Plot Analysis [written] (3 points)

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

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

```
In [143]:  words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oil'
           plot_embeddings(M_reduced_normalized, word2ind, words)
```
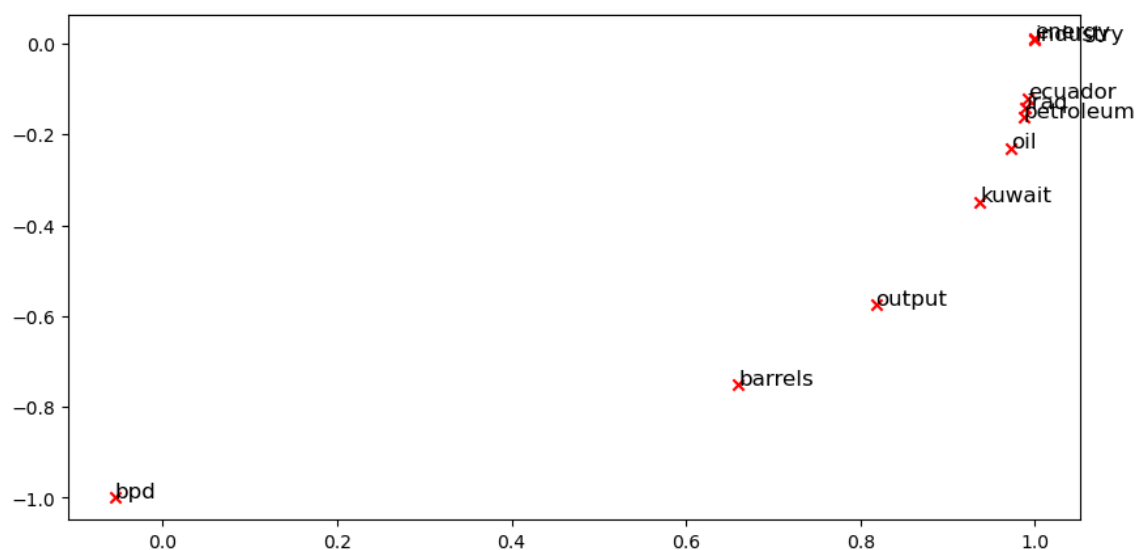

Write your answer here.

the clusters are roughly like the following: cluster 1: bpd cluster 2:barrels and output cluster 3: rest of the words

I think barrels should be clustered with bpd instead of output. The difference between the plots is in this one, output is much close to the barrels instead of other words so we can put it in the same cluster with barrels. And rest of the words also closer to each other comparing to the first one. The embedding aproach of the words can be cause this difference.

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:

$$\theta = \arccos(x \cdot y / |x| |y|)$$
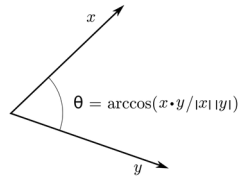
Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos(\Theta)$. Formally the Cosine Similarity (https://en.wikipedia.org/wiki/Cosine_similarity) s between two vectors p and q is defined as:

Question 2.2: Words with Multiple Meanings (1.5 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 two different meanings* such that the top-10 most similar words (according to cosine similarity) contain related words from *both* meanings. For example, "leaves" has both "go_away" and "a_structure_of_a_plant" meaning in the top 10, and "scoop" has both "handed_waffle_cone" and "lowdown". You will probably need to try several polysemous or homonymic words before you find one.

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

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

In [144]:
```python
# ------------------
# Write your implementation here.
print(wv_from_bin.most_similar('lie'))

#tried words: book, bright, bat, date, key, light,
#i believe i got different meanings of lie with truth and sit
# ------------------
```

[('lying', 0.6884375214576721), ('lies', 0.6648906469345093), ('lay', 0.4987420439720154), ('beneath', 0.4979451894760132), ('hide', 0.4929003417491913), ('exist', 0.4912907481935425), ('sit', 0.48945313692092896), ('truth', 0.4867425560951233), ('bare', 0.48414286971092224), ('these', 0.4807294011116028)]

Write your answer here.

I think this is related to the corpus. If in the corpus, 1 meaning of a word is dominant to the

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

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

You should use the the `wv_from_bin.distance(w1, w2)` function here in order to compute the cosine distance between two words. Please see the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvec** for further assistance.

In [145]:
```python
# ------------------
# Write your implementation here.


print(wv_from_bin.distance('happy','sad'))
print(wv_from_bin.distance('happy','cheerful'))

print('Antonym distance: ',wv_from_bin.distance('hard','soft'))
print('Synonym distance: ',wv_from_bin.distance('hard','rough'))

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

```
0.40401363372802734
0.5172466933727264
Antonym distance:  0.44494473934173584
Synonym distance:  0.45697635412216187
```

Write your answer here.

I tried hard,soft, and rough. I expected the distance between hard and soft greater than the distance between hard and rough because hard and soft are antonyms but hard and rough are synonmys. But the distance between hard and soft is 0.445 but the distance between hard and rough is 0.457. The reason behind this can be the following: each word has it's own unique embeddings, and even though in a human's perspective the meanings of the words are similar, in vector space their embedding may not be close because in the corpus, that kind of prior information is not available for model.

Question 2.4: Analogies with Word Vectors [written] (1.5 points)

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

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

In the cell below, we show you how to use word vectors to find x using the `most_similar` function from the **GenSim documentation (https://radimrehurek.com/gensim/models/keyedvectors.html#gensim.models.keyedvec** The function finds words that are most similar to the words in the `positive` list and most dissimilar from the words in the `negative` list (while omitting the input words, which are often the most similar; see this paper (https://www.aclweb.org/anthology/N18-2039.pdf)). The answer to the analogy will have the highest cosine similarity (largest returned numerical value).

```
In [146]:   # Run this cell to answer the analogy -- man : king :: woman : x
            pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], negative=
```

```
[('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)]
```

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

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

Write your answer here.

In this example where m=man, k=king, and w=woman, as we can see from the above cosine similarities, x should be queen. The cosine similarity can be calculated with the vector arithmetic can be expressed as the following: x= k-m+w. With the equation, I subtracted man from king, the intuition is eliminating being man from being king, and add woman instead, so we got what we call a king if it is not man but woman? Queen. This will maximize the cosine similarity because queen has the greates cosine similarity comparing to others like princess, monarch etc.

Question 2.5: Finding Analogies [code + written] (1.5 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 [147]:  # ------------------
           # Write your implementation here.
           # x:y :: a:b -> man is to king as woman is to x (format)
           #rome is to italy as sofia is to bulgaria
           #rome:italy :: sofia:bulgaria
           pprint.pprint(wv_from_bin.most_similar(positive=['italy', 'sofia'], negative

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

```
[('bulgaria', 0.6866204738616943),
 ('levski', 0.5451005697250366),
 ('slovakia', 0.5311755537986755),
 ('arvidsson', 0.5301610231399536),
 ('bulgarian', 0.5292257070541382),
 ('sweden', 0.5004056692123413),
 ('finland', 0.4876615107059479),
 ('romania', 0.48506829142570496),
 ('hungary', 0.47911474108695984),
 ('blagoevgrad', 0.47849082946777344)]
```

Write your answer here.

The analogy is: rome is to italy as sofia is to bulgaria. This analogy is about countries and their capitals.

Question 2.6: Incorrect Analogy [code + written] (1.5 points)

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 [148]:  # ------------------
           # Write your implementation here.
           #sweet is to candy as sour is to lemon
           #sweet:candy :: sour:lemon
           pprint.pprint(wv_from_bin.most_similar(positive=['candy', 'sour'], negative

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

```
[('candies', 0.5462512969970703),
 ('soda', 0.5375092029571533),
 ('chocolate', 0.5183511972427368),
 ('yogurt', 0.5167396068572998),
 ('cream', 0.5093993544578552),
 ('cereal', 0.47105589509010315),
 ('caramel', 0.4681696891784668),
 ('applesauce', 0.46436604857444763),
 ('cookies', 0.4613129496574402),
 ('butter', 0.4543982744216919)]
```

Write your answer here.

Question 2.7: 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 [149]:  # Run this cell
           # Here `positive` indicates the list of words to be similar to and `negative
           # most dissimilar from.
           pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'worker'], negativ
           print()
           pprint.pprint(wv_from_bin.most_similar(positive=['man', 'worker'], negative=
```

```
[('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)]
```

Write your answer here.

When we checked the list for similar with woman and worker but dissimilar to man, we can see nurse, pregnant, mother, child which are gender-biased. When we checked the list for similar with man and worker but dissimilar to woman, we can see mechanic and factory can be said gender-biased. In the top predictions of both lists, it tends to give employee/employer and workes, which are not gender-biased, but after top2 predictions in woman list, it starts giving gender-biased words, where for man this gender-biased words appear later like 8th entry. So I believe we can say there is a negative bias toward woman because mother, child, pregnant does not closely related to the worker class according to modern feminist theories therefore this results can be problematic.

Question 2.8: 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 [150]: # ------------------
          # Write your implementation here.
          pprint.pprint(wv_from_bin.most_similar(positive=['human', 'european'], negat
          print()
          pprint.pprint(wv_from_bin.most_similar(positive=['human', 'african'], negati

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

```
[('eu', 0.5649541616439819),
 ('rights', 0.5320478677749634),
 ('europe', 0.524067223072052),
 ('animal', 0.4800909757614136),
 ('fundamental', 0.4670671820640564),
 ('humans', 0.4633054733276367),
 ('protection', 0.4604228138923645),
 ('environmental', 0.45908528566360474),
 ('expression', 0.4589957594871521),
 ('possible', 0.45526012778282166)]

[('aids', 0.5491437911987305),
 ('hiv', 0.5123550891876221),
 ('africa', 0.5116332769393921),
 ('rights', 0.49636009335517883),
 ('humanity', 0.47854310274124146),
 ('darfur', 0.4725809693336487),
 ('africans', 0.47198355197906494),
 ('beings', 0.47111910581588745),
```

Write your answer here. when I used european + human -african like an approach, it gave me eu, rights and europe in top 3. But when I used african+human-european, top 3 is aids, hiv, and africa. It seems like there is a bias towards the continent, because i did not expect seeing 2 sickness in top 2 for africa, I expected like europe, it will more continent spesific geographical features.

Question 2.9: Thinking About Bias [written] (2 points)

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

Write your answer here. To understand the source of the bias, we can create word embeddings with different corpuses. Those biases are coming from the corpuses that the embeddings are created. We can check the results of different embeddings coming from different corpuses and see if the bias is common between different embeddings too.

Part 3: Sentiment Analysis (15 points)

Lastly, you will implement a simple sentiment classifier **from scratch** by using the Deep Averaging Network (DAN) proposed in the paper (https://aclanthology.org/P15-1162.pdf). The model is based on the following three steps:

- Take the vector average of the embeddings associated with the words in the inputs
- Pass that average vector through one or more feed-forward layers
- Perform linear classification on the final layer's representation

Here, you will use Stanford Sentiment Treebank (SST) dataset but note that in this dataset, the sentiment levels are originally represented with real values. Hence, you need to discretize these values into the following five classes:

- 0: "very negative" (≤ 0.2),
- 1: "negative" (≤ 0.4),
- 2: "neutral" (≤ 0.6),
- 3: "positive" (≤ 0.8),
- 4: "very positive" (> 0.8)

Download the Dataset

You can download the dataset here (https://nlp.stanford.edu/sentiment/) (Download the **"Main zip file with readme (6mb)"** version). Please read `README.txt` in details, that comes with the .zip folder.

Create a /data directory to store your SST data and unzip your downloaded folder there. Your data path should be like following:

```
./comp541-441/assignment1/data
                            └── stanfordSentimentTreebank
                                ├── README.txt
                                ├── SOStr.txt
                                ├── STree.txt
                                ├── datasetSentences.txt
                                ├── datasetSplit.txt
                                ├── dictionary.txt
                                ├── original_rt_snippets.txt
                                └── sentiment_labels.txt
```

Or, you can simply use Huggingface's **datasets** library if you are familiar.

What to show

In your work, perform the following experiments and explain your findings:

- Provide your loss curves by plotting them clearly,
- Play with the number of layers,
- Try with embeddings trained on different corpuses
- Test with the GloVe embeddings and the embeddings formed through the word co-occurrence matrix. Report your results on the test set for both types of embeddings

(make sure to use the same test set for both, to ensure a fair comparison).

In [151]:
```python
# ------------------
# Start your implementation here.
import csv
#data load part
data_dir='C:\\Users\\Dell\\Desktop\\comp541-assignment1\\data\\stanfordSenti
sentences_file = f'{data_dir}\\datasetSentences.txt'
labels_file = f'{data_dir}\\sentiment_labels.txt'
sentiment_labels_file = f'{data_dir}\\sentiment_labels.txt'
dataset_split_file = f'{data_dir}\\datasetSplit.txt'
dictionary_file = f'{data_dir}\\dictionary.txt'

def get_sentiment_class(score):
    if score <= 0.2:
        return 0
    elif score <= 0.4:
        return 1
    elif score <= 0.6:
        return 2
    elif score <= 0.8:
        return 3
    else:
        return 4

sentences = []
with open(sentences_file, 'r', encoding='utf-8') as f:
    next(f)
    for line in f:
        parts = line.strip().split('\t')
        if len(parts) > 1:
            sentences.append(parts[1])

all_phrases = []
all_p_ids = []


with open(dictionary_file, 'r', encoding='utf-8') as file:
    reader = csv.reader(file, delimiter='|')
    for row in reader:
        all_phrases.append(row[0].strip())
        all_p_ids.append(int(row[1]))


phrase_id = []
sentiment_values = []

with open(sentiment_labels_file, 'r') as file:
    reader = csv.reader(file, delimiter='|')
    next(reader)
    for row in reader:
        phrase_id.append(int(row[0]))
        sentiment_values.append(float(row[1]))

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

In [152]:
```python
sentence_phrase_index=[]
for sentence in sentences:
    try:

        index = all_phrases.index(sentence)
        sentence_phrase_index.append(index)
    except ValueError:

        sentence_phrase_index.append(-1)
```

In [153]:
```python
scores = []

for index in sentence_phrase_index:
    if index != -1:

        sentiment_index = all_p_ids[index]
        scores.append(sentiment_values[sentiment_index])
    else:

        scores.append(-1)
```

In [154]:
```python
train_indices = []
test_indices = []
dev_indices = []

with open(dataset_split_file, 'r', encoding='utf-8') as f:
    next(f)
    for line in f:
        index, split_label = line.strip().split(',')
        index = int(index) - 1
        if split_label == '1':
            train_indices.append(index)
        elif split_label == '2':
            test_indices.append(index)
        elif split_label == '3':
            dev_indices.append(index)

train_sentences = [sentences[i] for i in train_indices if i < len(sentences)]
train_labels = [scores[i] for i in train_indices if i < len(sentences)]
train_labels = [get_sentiment_class(label) if label != -1 else -1 for label
train_sentences1 = []
train_labels1 = []

for index, class_label in enumerate(train_labels):
    if class_label != -1:
        train_sentences1.append(train_sentences[index])
        train_labels1.append(class_label)

test_sentences = [sentences[i] for i in test_indices if i < len(sentences)]
test_labels = [scores[i] for i in test_indices if i < len(sentences)]
test_labels = [get_sentiment_class(label) if label != -1 else -1 for label
test_sentences1 = []
test_labels1 = []

for index, class_label in enumerate(test_labels):
    if class_label != -1:
        test_sentences1.append(test_sentences[index])
        test_labels1.append(class_label)

dev_sentences = [sentences[i] for i in dev_indices if i < len(sentences)]
dev_labels = [scores[i] for i in dev_indices if i < len(sentences)]
dev_labels = [get_sentiment_class(label) if label != -1 else -1 for label i
dev_sentences1 = []
dev_labels1 = []

for index, class_label in enumerate(dev_labels):
    if class_label != -1:
        dev_sentences1.append(dev_sentences[index])
        dev_labels1.append(class_label)
```

```python
In [155]:  #helper functions

           def average_embed(sentences, word2ind, embedding_matrix):

               sentence_embeddings = []

               for sentence in sentences:
                   words = sentence.split()
                   word_vectors = []

                   for word in words:
                       if word in word2ind:
                           index = word2ind[word]
                           word_vectors.append(embedding_matrix[index])

                   if word_vectors:

                       avg_vector = np.mean(word_vectors, axis=0)
                   else:

                       avg_vector = np.zeros(embedding_matrix.shape[1])

                   sentence_embeddings.append(avg_vector)

               return torch.tensor(sentence_embeddings, dtype=torch.float32)
```

```python
In [156]:  vocab_words = list(wv_from_bin.vocab.keys())
           embeddings_matrix = np.array([wv_from_bin[word] for word in vocab_words])

           word_to_index = {word: i for i, word in enumerate(vocab_words)}
```

```python
In [157]:  len(vocab_words)
```

```
Out[157]:  400000
```

In [158]:
```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F

class DAN(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super(DAN, self).__init__()

        self.linear1 = nn.Linear(input_dim, hidden_dim)
        self.relu = nn.ReLU()
        self.linear2 = nn.Linear(hidden_dim, hidden_dim)
    #    self.relu2=nn.ReLU()
      # self.linear3=nn.Linear(hidden_dim,output_dim)

    def forward(self, x):

        x = self.linear1(x)
        x = self.relu(x)
        x = self.linear2(x)
      #x=self.relu(x)
    #  x=self.linear3(x)
        return x
```

In [159]:
```python
import matplotlib.pyplot as plt

train_embeddings = average_embed(train_sentences1, word_to_index, embedding
dev_embeddings=average_embed(dev_sentences1,word_to_index,embeddings_matrix)
train_labels_tensor = torch.tensor(train_labels1, dtype=torch.long)
dev_labels_tensor=torch.tensor(dev_labels1,dtype=torch.long)
num_classes=5


input_dim = M.shape[1]
hidden_dim = 64
output_dim = num_classes


model = DAN(input_dim, hidden_dim, output_dim)
batch_size=16

train_dataset = torch.utils.data.TensorDataset(train_embeddings, train_label
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_s
dev_dataset = torch.utils.data.TensorDataset(dev_embeddings, dev_labels_tens
dev_loader = torch.utils.data.DataLoader(dev_dataset, batch_size=batch_size)


criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

train_losses = []
dev_losses = []
train_accuracies = []
dev_accuracies = []

num_epochs = 10
def train(model, train_loader, dev_loader, criterion, optimizer, num_epochs)
    train_losses = []
    dev_losses = []
    train_accuracies = []
    dev_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        train_loss = 0.0
        train_correct = 0

        # Training loop
        for inputs, labels in train_loader:
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            train_loss += loss.item() * labels.size(0)
            _, train_preds = torch.max(outputs, 1)
            train_correct += (train_preds == labels).sum().item()

        train_loss /= len(train_loader.dataset)
        train_accuracy = train_correct / len(train_loader.dataset)
        train_losses.append(train_loss)
        train_accuracies.append(train_accuracy)

        # Validation loop
```

```python
            model.eval()
            dev_loss = 0.0
            dev_correct = 0

            with torch.no_grad():
                for inputs, labels in dev_loader:
                    dev_outputs = model(inputs)
                    loss = criterion(dev_outputs, labels)
                    dev_loss += loss.item() * labels.size(0)
                    _, dev_preds = torch.max(dev_outputs, 1)
                    dev_correct += (dev_preds == labels).sum().item()

            dev_loss /= len(dev_loader.dataset)
            dev_accuracy = dev_correct / len(dev_loader.dataset)
            dev_losses.append(dev_loss)
            dev_accuracies.append(dev_accuracy)

            # Print progress
            print(f'Epoch [{epoch + 1}/{num_epochs}], '
                  f'Train Loss: {train_loss:.4f}, '
                  f'Train Accuracy: {train_accuracy:.4f}, '
                  f'Dev Loss: {dev_loss:.4f}, '
                  f'Dev Accuracy: {dev_accuracy:.4f}')

    return train_losses, dev_losses, train_accuracies, dev_accuracies

train_losses, dev_losses, train_accuracies, dev_accuracies = train(
    model, train_loader, dev_loader, criterion, optimizer, num_epochs=num_ep
)

plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), train_losses, label='Train Loss')
plt.plot(range(1, num_epochs + 1), dev_losses, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()
plt.show()
```

```
Epoch [1/10], Train Loss: 1.4689, Train Accuracy: 0.4595, Dev Loss: 1.624
8, Dev Accuracy: 0.3027
Epoch [2/10], Train Loss: 1.2468, Train Accuracy: 0.4747, Dev Loss: 1.519
9, Dev Accuracy: 0.3439
Epoch [3/10], Train Loss: 1.2084, Train Accuracy: 0.4877, Dev Loss: 1.455
4, Dev Accuracy: 0.3630
Epoch [4/10], Train Loss: 1.1853, Train Accuracy: 0.4939, Dev Loss: 1.433
3, Dev Accuracy: 0.3602
Epoch [5/10], Train Loss: 1.1674, Train Accuracy: 0.4975, Dev Loss: 1.413
6, Dev Accuracy: 0.3630
Epoch [6/10], Train Loss: 1.1587, Train Accuracy: 0.4992, Dev Loss: 1.407
1, Dev Accuracy: 0.3659
Epoch [7/10], Train Loss: 1.1474, Train Accuracy: 0.5031, Dev Loss: 1.401
1, Dev Accuracy: 0.3774
Epoch [8/10], Train Loss: 1.1406, Train Accuracy: 0.5073, Dev Loss: 1.390
8, Dev Accuracy: 0.3831
Epoch [9/10], Train Loss: 1.1320, Train Accuracy: 0.5120, Dev Loss: 1.396
9, Dev Accuracy: 0.3697
Epoch [10/10], Train Loss: 1.1274, Train Accuracy: 0.5135, Dev Loss: 1.388
8, Dev Accuracy: 0.3745
```

Training and Validation Loss over Epochs
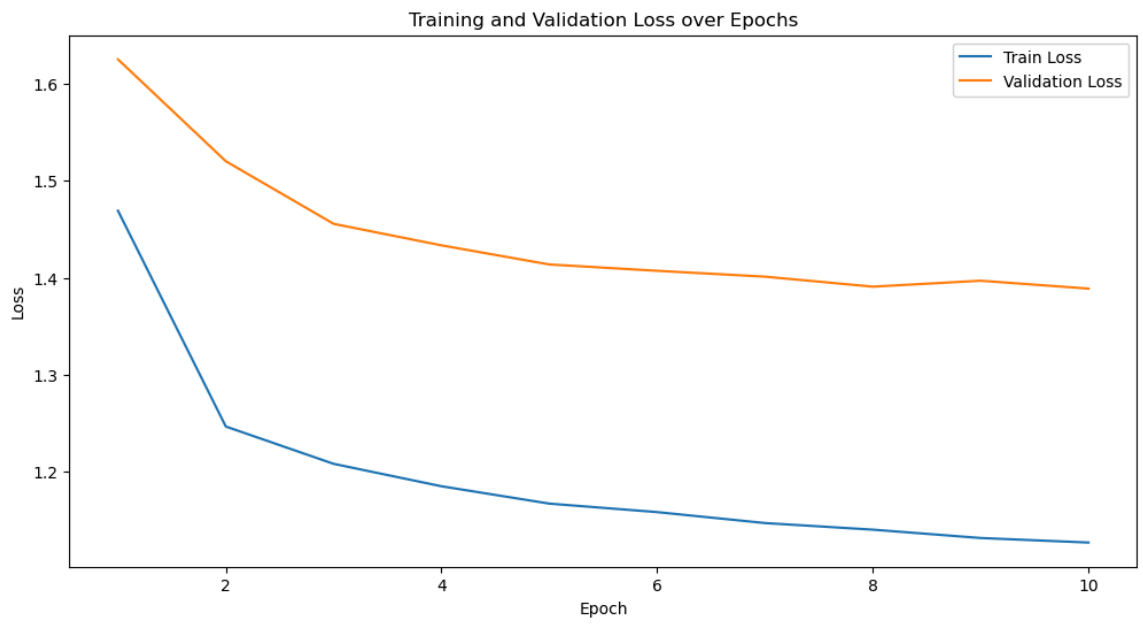
```python
In [160]: def evaluate_model(model, test_sentences, test_labels, word_to_index, embedd
              model.eval()
              total_loss = 0.0
              correct_predictions = 0
              total_samples = 0


              test_embeddings = average_embed(test_sentences, word_to_index, embedding
              test_labels_tensor = torch.tensor(test_labels, dtype=torch.long)


              test_dataset = torch.utils.data.TensorDataset(test_embeddings, test_labe
              test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch

              criterion = nn.CrossEntropyLoss()

              with torch.no_grad():
                  for inputs, labels in test_loader:



                      outputs = model(inputs)


                      loss = criterion(outputs, labels)
                      total_loss += loss.item()


                      _, predicted = torch.max(outputs, 1)
                      correct_predictions += (predicted == labels).sum().item()
                      total_samples += labels.size(0)


              avg_loss = total_loss / len(test_loader)
              accuracy = correct_predictions / total_samples * 100

              print(f"Test Loss: {avg_loss:.4f}, Accuracy: {accuracy:.2f}%")

          evaluate_model(model, test_sentences1, test_labels1, word_to_index, embeddir
```

Test Loss: 1.3987, Accuracy: 39.34%

In [161]:
```python
from collections import Counter
import re

def create_vocabulary(sentences):
    # Tokenize sentences and clean up punctuation
    words = []
    for sentence in sentences:
        # Convert to lowercase, remove punctuation, and split by whitespace
        tokens = re.findall(r'\b\w+\b', sentence.lower())
        words.extend(tokens)

    # Create a counter to get unique words
    word_counts = Counter(words)
    vocabulary = sorted(word_counts.keys())
    return vocabulary

# Generate vocabulary
vocabulary1 = create_vocabulary(sentences)
print(len(vocabulary1))
print("Vocabulary Size:", len(vocabulary1))
```

```
17825
Vocabulary Size: 17825
```

In [162]:
```python
def format_sentence(sentence):
    # Add START and END tokens
    formatted = ['<START>'] + sentence.split() + ['<END>']
    return formatted
formatted_sentences = [format_sentence(sentence) for sentence in sentences]
print(formatted_sentences[0])
```

```
['<START>', 'The', 'Rock', 'is', 'destined', 'to', 'be', 'the', '21st', 'C
entury', "'s", 'new', '``', 'Conan', "'", 'and', 'that', 'he', "'s", 'goi
ng', 'to', 'make', 'a', 'splash', 'even', 'greater', 'than', 'Arnold', 'Sc
hwarzenegger', ',', 'Jean-Claud', 'Van', 'Damme', 'or', 'Steven', 'Segal',
'.', '<END>']
```

In [163]:
```python
print(dev_embeddings1[0].shape)
```

```
torch.Size([21703])
```

In [164]:
```python
#now lets train with our cooccurence matrix
M2a, word2inda = compute_co_occurrence_matrix(formatted_sentences)
#M2b = reduce_to_k_dim(M2a, k=200)
#M_lengths = np.linalg.norm(M1, axis=1)
#M2 = M1 / M_lengths[:, np.newaxis]


train_embeddings1 = average_embed(train_sentences1, word2inda, M2a)
dev_embeddings1=average_embed(dev_sentences1,word2inda,M2a)
train_labels_tensor1 = torch.tensor(train_labels1, dtype=torch.long)
dev_labels_tensor1=torch.tensor(dev_labels1,dtype=torch.long)


model2 = DAN(M2a.shape[1] , hidden_dim, output_dim)
optimizer = optim.Adam(model2.parameters(), lr=0.001)

train_dataset1 = torch.utils.data.TensorDataset(train_embeddings1, train_lal
train_loader1 = torch.utils.data.DataLoader(train_dataset1, batch_size=batch
dev_dataset1 = torch.utils.data.TensorDataset(dev_embeddings1, dev_labels_te
dev_loader1 = torch.utils.data.DataLoader(dev_dataset1, batch_size=batch_siz


train_losses1 = []
dev_losses1 = []
train_accuracies1 = []
dev_accuracies1 = []

num_epochs=10
train_losses1, dev_losses1, train_accuracies1, dev_accuracies1 = train(
    model2, train_loader1, dev_loader1, criterion, optimizer, num_epochs=nur
)

plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), train_losses1, label='Train Loss')
plt.plot(range(1, num_epochs + 1), dev_losses1, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()
plt.show()
```

```
Epoch [1/10], Train Loss: 1.5874, Train Accuracy: 0.4070, Dev Loss: 1.746
6, Dev Accuracy: 0.2557
Epoch [2/10], Train Loss: 1.3153, Train Accuracy: 0.4541, Dev Loss: 1.930
7, Dev Accuracy: 0.2701
Epoch [3/10], Train Loss: 1.3608, Train Accuracy: 0.4510, Dev Loss: 3.820
2, Dev Accuracy: 0.2519
Epoch [4/10], Train Loss: 3.0705, Train Accuracy: 0.2750, Dev Loss: 2.615
5, Dev Accuracy: 0.2692
Epoch [5/10], Train Loss: 1.5116, Train Accuracy: 0.4390, Dev Loss: 1.705
9, Dev Accuracy: 0.2692
Epoch [6/10], Train Loss: 1.3491, Train Accuracy: 0.4440, Dev Loss: 4.282
3, Dev Accuracy: 0.2692
Epoch [7/10], Train Loss: 1.3907, Train Accuracy: 0.4470, Dev Loss: 2.456
5, Dev Accuracy: 0.2519
Epoch [8/10], Train Loss: 2.1868, Train Accuracy: 0.2787, Dev Loss: 2.001
1, Dev Accuracy: 0.2692
Epoch [9/10], Train Loss: 1.9012, Train Accuracy: 0.2861, Dev Loss: 1.830
3, Dev Accuracy: 0.2692
Epoch [10/10], Train Loss: 1.7813, Train Accuracy: 0.2769, Dev Loss: 1.746
6, Dev Accuracy: 0.2663
```


Training and Validation Loss over Epochs
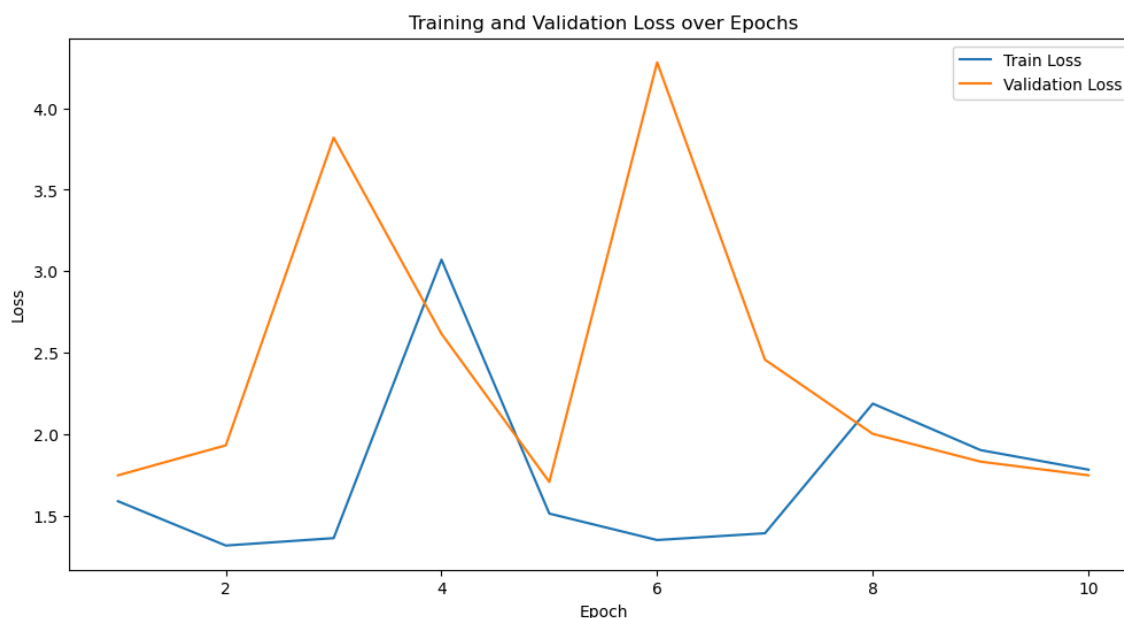
In [165]: `evaluate_model(model2, test_sentences1, test_labels1, word2inda, M2a)`

Test Loss: 1.7595, Accuracy: 28.47%

In [166]:

```python
#now lets try some other hyperparameters

train_lossesh = []
dev_lossesh = []
train_accuraciesh = []
dev_accuraciesh = []


model_h = DAN(input_dim, hidden_dim, output_dim)
optimizer_h = optim.SGD(model_h.parameters(), lr=0.01, momentum=0.9)

train_lossesh, dev_lossesh, train_accuraciesh, dev_accuraciesh = train(
    model_h, train_loader, dev_loader, criterion, optimizer_h, num_epochs=nu
)

plt.figure(figsize=(12, 6))
plt.plot(range(1, num_epochs + 1), train_lossesh, label='Train Loss')
plt.plot(range(1, num_epochs + 1), dev_lossesh, label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss over Epochs')
plt.legend()
plt.show()
```

```
Epoch [1/10], Train Loss: 1.4208, Train Accuracy: 0.4334, Dev Loss: 1.812
7, Dev Accuracy: 0.2481
Epoch [2/10], Train Loss: 1.2803, Train Accuracy: 0.4548, Dev Loss: 1.767
1, Dev Accuracy: 0.2682
Epoch [3/10], Train Loss: 1.2558, Train Accuracy: 0.4672, Dev Loss: 1.721
5, Dev Accuracy: 0.2663
Epoch [4/10], Train Loss: 1.2376, Train Accuracy: 0.4735, Dev Loss: 1.685
1, Dev Accuracy: 0.2720
Epoch [5/10], Train Loss: 1.2225, Train Accuracy: 0.4799, Dev Loss: 1.656
8, Dev Accuracy: 0.2682
Epoch [6/10], Train Loss: 1.2110, Train Accuracy: 0.4826, Dev Loss: 1.662
8, Dev Accuracy: 0.2529
Epoch [7/10], Train Loss: 1.1974, Train Accuracy: 0.4859, Dev Loss: 1.662
0, Dev Accuracy: 0.2557
Epoch [8/10], Train Loss: 1.1884, Train Accuracy: 0.4892, Dev Loss: 1.673
3, Dev Accuracy: 0.2634
Epoch [9/10], Train Loss: 1.1787, Train Accuracy: 0.4950, Dev Loss: 1.717
7, Dev Accuracy: 0.2395
Epoch [10/10], Train Loss: 1.1706, Train Accuracy: 0.4961, Dev Loss: 1.731
3, Dev Accuracy: 0.2414
```

Training and Validation Loss over Epochs
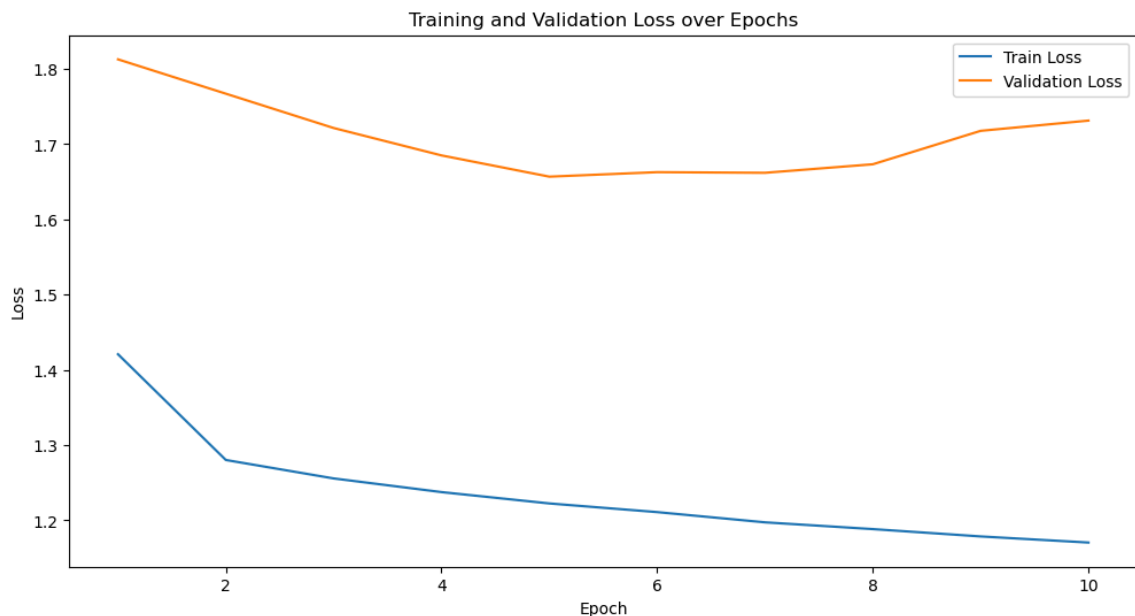
In [167]: `evaluate_model(model_h, test_sentences1, test_labels1, word_to_index, embedd`

Test Loss: 1.7166, Accuracy: 26.68%

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.
6. Download a .ipynb version of your notebook
7. Please name your files as username_assignment1.ipynb and username_assignment1.pdf.
8. Submit your work to Blackboard by the deadline.

This assignment is adapted from Stanford [CS224n (http://web.stanford.edu/class/cs224n/)](http://web.stanford.edu/class/cs224n/)