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 [123]:
```
# 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\Tolga\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 $w_{i-n} \dots w_{i-1}$ 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"

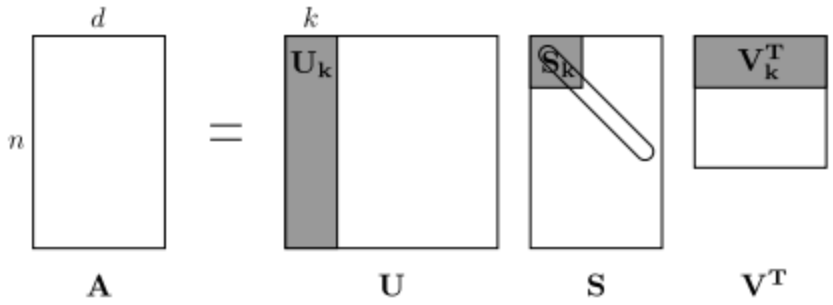
Document 2: "all is well that ends well"

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

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

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 [124]:
```
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 [125]:

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

```
[['<START>', 'japan', 'to', 'revise', 'long', '-', 'term', 'energy', 'deman
d', 'downwards', 'the',
  'ministry', 'of', 'international', 'trade', 'and', 'industry', '(', 'miti',
')', 'will', 'revise',
  'its', 'long', '-', 'term', 'energy', 'supply', '/', 'demand', 'outlook',
'by', 'august', 'to',
  'meet', 'a', 'forecast', 'downtrend', 'in', 'japanese', 'energy', 'demand',
',', 'ministry',
  'officials', 'said', '.', 'miti', 'is', 'expected', 'to', 'lower', 'the',
'projection', 'for',
  'primary', 'energy', 'supplies', 'in', 'the', 'year', '2000', 'to', '550',
'mln', 'kilolitres',
  '(', 'kl', ')', 'from', '600', 'mln', ',', 'they', 'said', '.', 'the', 'dec
ision', 'follows',
  'the', 'emergence', 'of', 'structural', 'changes', 'in', 'japanese', 'indus
try', 'following',
  'the', 'rise', 'in', 'the', 'value', 'of', 'the', 'yen', 'and', 'a', 'decli
ne', 'in', 'domestic',
  'electric', 'power', 'demand', '.', 'miti', 'is', 'planning', 'to', 'work',
'out', 'a', 'revised',
  'energy', 'supply', '/', 'demand', 'outlook', 'through', 'deliberations',
'of', 'committee',
  'meetings', 'of', 'the', 'agency', 'of', 'natural', 'resources', 'and', 'en
ergy', ',', 'the',
  'officials', 'said', '.', 'they', 'said', 'miti', 'will', 'also', 'review',
'the', 'breakdown',
  'of', 'energy', 'supply', 'sources', ',', 'including', 'oil', ',', 'nuclea
r', ',', 'coal', 'and',
  'natural', 'gas', '.', 'nuclear', 'energy', 'provided', 'the', 'bulk', 'o
f', 'japan', "'", 's',
  'electric', 'power', 'in', 'the', 'fiscal', 'year', 'ended', 'march', '31',
',', 'supplying',
  'an', 'estimated', '27', 'pct', 'on', 'a', 'kilowatt', '/', 'hour', 'basi
s', ',', 'followed',
  'by', 'oil', '(', '23', 'pct', ')', 'and', 'liquefied', 'natural', 'gas',
'(', '21', 'pct', '),',
  'they', 'noted', '.', '<END>'],
 ['<START>', 'energy', '/', 'u', '.', 's', '.', 'petrochemical', 'industry',
'cheap', 'oil',
  'feedstocks', ',', 'the', 'weakened', 'u', '.', 's', '.', 'dollar', 'and',
'a', 'plant',
  'utilization', 'rate', 'approaching', '90', 'pct', 'will', 'propel', 'the',
'streamlined', 'u',
  '.', 's', '.', 'petrochemical', 'industry', 'to', 'record', 'profits', 'thi
s', 'year', ',',
  'with', 'growth', 'expected', 'through', 'at', 'least', '1990', ',', 'majo
r', 'company',
  'executives', 'predicted', '.', 'this', 'bullish', 'outlook', 'for', 'chemi
cal', 'manufacturing',
  'and', 'an', 'industrywide', 'move', 'to', 'shed', 'unrelated', 'businesse
s', 'has', 'prompted',
  'gaf', 'corp', '&', 'lt', ';', 'gaf', '>,', 'privately', '-', 'held', 'cai
n', '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',
'to', 'buy', '.', '"',
  'i', 'see', 'us', 'poised', 'at', 'the', 'threshold', 'of', 'a', 'golden',
'period', ,"', 'said',
  'paul', 'oreffice', ',', 'chairman', 'of', 'giant', 'dow', 'chemical', 'c
o', '&', 'lt', ';',
  'dow', '>,', 'adding', ',', '"', 'there', "'", 's', 'no', 'major', 'plant',
'capacity', 'being',
  'added', 'around', 'the', 'world', 'now', '.', 'the', 'whole', 'game', 'i
s', 'bringing', 'out',
  'new', 'products', 'and', 'improving', 'the', 'old', 'ones', '."', 'analyst
s', 'say', 'the',
  'chemical', 'industry', "'", 's', 'biggest', 'customers', ',', 'automobil
e', 'manufacturers',
  'and', 'home', 'builders', 'that', 'use', 'a', 'lot', 'of', 'paints', 'an
d', 'plastics', ',',
  'are', 'expected', 'to', 'buy', 'quantities', 'this', 'year', '.', 'u',
'.', 's', '.',
  'petrochemical', 'plants', 'are', 'currently', 'operating', 'at', 'about',
'90', 'pct',
  'capacity', ',', 'reflecting', 'tighter', 'supply', 'that', 'could', 'hik
e', 'product', 'prices',
  'by', '30', 'to', '40', 'pct', 'this', 'year', ',', 'said', 'john', 'doshe
r', ',', 'managing',
  'director', 'of', 'pace', 'consultants', 'inc', 'of', 'houston', '.', 'dema
nd', 'for', 'some',
  'products', 'such', 'as', 'styrene', 'could', 'push', 'profit', 'margins',
'up', 'by', 'as',
  'much', 'as', '300', 'pct', ',', 'he', 'said', '.', 'oreffice', ',', 'speak
ing', 'at', 'a',
  'meeting', 'of', 'chemical', 'engineers', 'in', 'houston', ',', 'said', 'do
w', 'would', 'easily',
  'top', 'the', '741', 'mln', 'dlrs', 'it', 'earned', 'last', 'year', 'and',
'predicted', 'it',
  'would', 'have', 'the', 'best', 'year', 'in', 'its', 'history', '.', 'in',
'1985', ',', 'when',
  'oil', 'prices', 'were', 'still', 'above', '25', 'dlrs', 'a', 'barrel', 'an
d', 'chemical',
  'exports', 'were', 'adversely', 'affected', 'by', 'the', 'strong', 'u',
'.', 's', '.', 'dollar',
  ',', 'dow', 'had', 'profits', 'of', '58', 'mln', 'dlrs', '.', '"', 'i', 'be
lieve', 'the',
  'entire', 'chemical', 'industry', 'is', 'headed', 'for', 'a', 'record', 'ye
ar', 'or', 'close',
  'to', 'it', ,"', 'oreffice', 'said', '.', 'gaf', 'chairman', 'samuel', 'he
yman', 'estimated',
  'that', 'the', 'u', '.', 's', '.', 'chemical', 'industry', 'would', 'repor
t', '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',
'once', '-', 'sickly',
  'chemical', 'industry', 'has', 'been', 'brought', 'about', 'by', 'a', 'comb
```

```
ination', 'of', 'luck',
  'and', 'planning', ',', 'said', 'pace', "'", 's', 'john', 'dosher', '.', 'd
osher', 'said', 'last',
  'year', "'", 's', 'fall', 'in', 'oil', 'prices', 'made', 'feedstocks', 'dra
matically', 'cheaper',
  'and', 'at', 'the', 'same', 'time', 'the', 'american', 'dollar', 'was', 'we
akening', 'against',
  'foreign', 'currencies', '.', 'that', 'helped', 'boost', 'u', '.', 's',
'.', 'chemical',
  'exports', '.', 'also', 'helping', 'to', 'bring', 'supply', 'and', 'deman
d', 'into', 'balance',
  'has', 'been', 'the', 'gradual', 'market', 'absorption', 'of', 'the', 'extr
a', 'chemical',
  'manufacturing', 'capacity', 'created', 'by', 'middle', 'eastern', 'oil',
'producers', 'in',
  'the', 'early', '1980s', '.', 'finally', ',', 'virtually', 'all', 'major',
'u', '.', 's', '.',
  'chemical', 'manufacturers', 'have', 'embarked', 'on', 'an', 'extensive',
'corporate',
  'restructuring', 'program', 'to', 'mothball', 'inefficient', 'plants', ',',
'trim', 'the',
  'payroll', 'and', 'eliminate', 'unrelated', 'businesses', '.', 'the', 'rest
ructuring', 'touched',
  'off', 'a', 'flurry', 'of', 'friendly', 'and', 'hostile', 'takeover', 'atte
mpts', '.', 'gaf', ',',
  'which', 'made', 'an', 'unsuccessful', 'attempt', 'in', '1985', 'to', 'acqu
ire', 'union',
  'carbide', 'corp', '&', 'lt', ';', 'uk', '>,', 'recently', 'offered', 'thre
e', 'billion', 'dlrs',
  'for', 'borg', 'warner', 'corp', '&', 'lt', ';', 'bor', '>,', 'a', 'chicag
o', 'manufacturer',
  'of', 'plastics', 'and', 'chemicals', '.', 'another', 'industry', 'powerhou
se', ',', 'w', '.',
  'r', '.', 'grace', '&', 'lt', ';', 'gra', '>', 'has', 'divested', 'its', 'r
etailing', ',',
  'restaurant', 'and', 'fertilizer', 'businesses', 'to', 'raise', 'cash', 'fo
r', 'chemical',
  'acquisitions', '.', 'but', 'some', 'experts', 'worry', 'that', 'the', 'che
mical', 'industry',
  'may', 'be', 'headed', 'for', 'trouble', 'if', 'companies', 'continue', 'tu
rning', 'their',
  'back', 'on', 'the', 'manufacturing', 'of', 'staple', 'petrochemical', 'com
modities', ',', 'such',
  'as', 'ethylene', ',', 'in', 'favor', 'of', 'more', 'profitable', 'specialt
y', 'chemicals',
  'that', 'are', 'custom', '-', 'designed', 'for', 'a', 'small', 'group', 'o
f', 'buyers', '.', '"',
  'companies', 'like', 'dupont', '&', 'lt', ';', 'dd', '>', 'and', 'monsant
o', 'co', '&', 'lt', ';',
  'mtc', '>', 'spent', 'the', 'past', 'two', 'or', 'three', 'years', 'tryin
g', 'to', 'get', 'out',
  'of', 'the', 'commodity', 'chemical', 'business', 'in', 'reaction', 'to',
'how', 'badly', 'the',
  'market', 'had', 'deteriorated', ,"', 'dosher', 'said', '.', '"', 'but',
'i', 'think', 'they',
  'will', 'eventually', 'kill', 'the', 'margins', 'on', 'the', 'profitable',
'chemicals', 'in',
```

```
  'the', 'niche', 'market', '."', 'some', 'top', 'chemical', 'executives', 's
hare', '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', 'chem
icals', 'may', 'be',
  'ill', '-', 'advised', '.', 'specialty', 'businesses', 'do', 'not', 'stay',
'special', 'long',
  '."', 'houston', '-', 'based', 'cain', 'chemical', ',', 'created', 'this',
'month', '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', 'levera
ged', 'buyout', 'of',
  'dupont', "'", 's', 'conoco', 'inc', "'", 's', 'chemical', 'business', ',',
'has', 'spent', '1',
  '.', '1', 'billion', 'dlrs', 'since', 'january', 'to', 'buy', 'seven', 'pet
rochemical', 'plants',
  'along', 'the', 'texas', 'gulf', 'coast', '.', 'the', 'plants', 'produce',
'only', 'basic',
  'commodity', 'petrochemicals', 'that', 'are', 'the', 'building', 'blocks',
'of', 'specialty',
  'products', '.', '"', 'this', 'kind', 'of', 'commodity', 'chemical', 'busin
ess', 'will', 'never',
  'be', 'a', 'glamorous', ',', 'high', '-', 'margin', 'business', ,"', 'cai
n', 'said', ',',
  'adding', 'that', 'demand', 'is', 'expected', 'to', 'grow', 'by', 'about',
'three', 'pct',
  'annually', '.', 'garo', 'armen', ',', 'an', 'analyst', 'with', 'dean', 'wi
tter', 'reynolds', ',',
  'said', 'chemical', 'makers', 'have', 'also', 'benefitted', 'by', 'increasi
ng', 'demand', 'for',
  'plastics', 'as', 'prices', 'become', 'more', 'competitive', 'with', 'alumi
num', ',', 'wood',
  'and', 'steel', 'products', '.', 'armen', 'estimated', 'the', 'upturn', 'i
n', 'the', 'chemical',
  'business', 'could', 'last', 'as', 'long', 'as', 'four', 'or', 'five', 'yea
rs', ',', 'provided',
  'the', 'u', '.', 's', '.', 'economy', 'continues', 'its', 'modest', 'rate',
'of', 'growth', '.',
  '<END>'],
 ['<START>', 'turkey', 'calls', 'for', 'dialogue', 'to', 'solve', 'dispute',
'turkey', 'said',
  'today', 'its', 'disputes', 'with', 'greece', ',', 'including', 'rights',
'on', 'the',
  'continental', 'shelf', 'in', 'the', 'aegean', 'sea', ',', 'should', 'be',
'solved', 'through',
  'negotiations', '.', 'a', 'foreign', 'ministry', 'statement', 'said', 'th
e', 'latest', 'crisis',
  'between', 'the', 'two', 'nato', 'members', 'stemmed', 'from', 'the', 'cont
inental', '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', '.', 'greec
e', 'has', 'repeatedly',
  'said', 'the', 'issue', 'was', 'legal', 'and', 'could', 'be', 'solved', 'a
t', 'the',
  'international', 'court', 'of', 'justice', '.', 'the', 'two', 'countries',
'approached', 'armed',
  'confrontation', 'last', 'month', 'after', 'greece', 'announced', 'it', 'pl
anned', 'oil',
  'exploration', 'work', 'in', 'the', 'aegean', 'and', 'turkey', 'said', 'i
t', 'would', 'also',
  'search', 'for', 'oil', '.', 'a', 'face', '-', 'off', 'was', 'averted', 'wh
en', 'turkey',
  'confined', 'its', 'research', 'to', 'territorrial', 'waters', '.', '"', 't
he', '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', 'gree
k', 'reply', 'to', 'a',
  'message', 'sent', 'last', 'week', 'by', 'turkish', 'prime', 'minister', 't
urgut', '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.

In [126]:

```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): sorted list of distinct words
            num_corpus_words (integer): number of distinct words across th
    """
    corpus_words = []
    num_corpus_words = -1

    # ------------------
    # Write your implementation here.
    corpus_words = [word for doc in corpus for word in doc]
    corpus_words = list(set(corpus_words))
    num_corpus_words = len(corpus_words)
    corpus_words.sort()

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

    return corpus_words, num_corpus_words
```

In [127]:

```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, EI
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 dis

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

# 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.

In [128]:

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

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

              For example, if we take the document "<START> All that glitt
              "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 :
                Co-occurence matrix of word counts.
                The ordering of the words in the rows/columns should be th
            word2ind (dict): dictionary that maps word to index (i.e. row/
    """
    words, num_words = distinct_words(corpus)
    M = None
    word2ind = {}

    # ------------------
    # Write your implementation here.
    M = np.zeros((num_words, num_words))
    word2ind = {value: index for index, value in enumerate(words)}
    print(f'{num_words}')

    for doc in corpus:
        curr_pos = -1    # start from the beginning of a document
        for word in doc:
            curr_pos += 1
            curr_ind = word2ind[word]

            # dynamically adjusts the window boundaries to prevent going o
            left_window = doc[max(0, curr_pos - window_size) : curr_pos]
            #print(f'{word} s left window {left_window}')
            right_window = doc[curr_pos + 1 : min(len(doc), curr_pos + wi

            for cooccur in left_window:
                cooccur_pos = word2ind[cooccur]
                M[curr_ind][cooccur_pos] += 1
                #print(f'{word} and {curr_pos} in LEFT')
            for cooccur in right_window:
                cooccur_pos = word2ind[cooccur]
                M[curr_ind][cooccur_pos] += 1
                #print(f'{word} and {curr_pos} in RIGHT')
    # ------------------

    return M, word2ind
```

In [129]:

```
# ---------------------
# 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, EI
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_s

# 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_w

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

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

# 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)
```

```
10
--------------------------------------------------------------------------------
---
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 [130]:

```
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_corpus_w
        to a matrix of dimensionality (num_corpus_words, k) using the foll
            - http://scikit-learn.org/stable/modules/generated/sklearn.dec

        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))
                    In terms of the SVD from math class, this actually ret
    """
    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 [131]:

```
# ---------------------
# 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, EN
M_test, word2ind_test = compute_co_occurrence_matrix(test_corpus, window_s
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 hav
assert (M_test_reduced.shape[1] == 2), "M_reduced has {} columns; should h

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

```
10
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`).

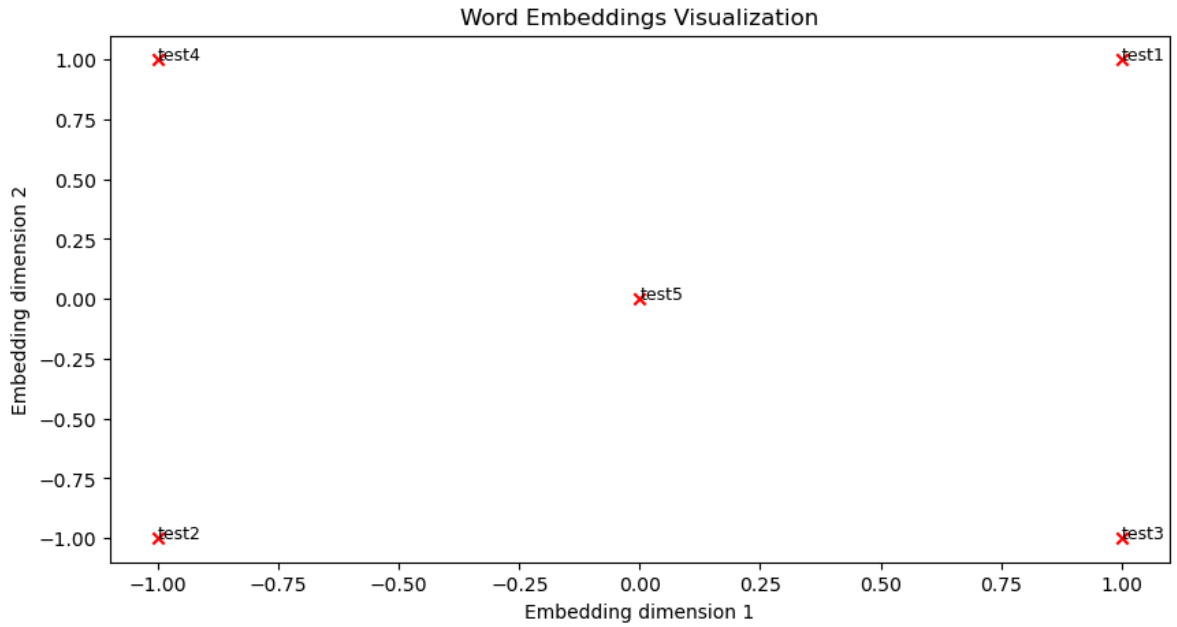
For this example, you may find it useful to adapt this code (http://web.archive.org/web/20190924160434/https://www.pythonmembers.club/2018/05/08/matplscatter-plot-annotate-set-text-at-label-each-point/). In the future, a good way to make a plot is to look at the Matplotlib gallery (https://matplotlib.org/gallery/index.html), find a plot that looks somewhat like what you want, and adapt the code they give.

In [132]:

```
def plot_embeddings(M_reduced, word2ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the
        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 th
            word2ind (dict): dictionary that maps word to indices for matr
            words (list of strings): words whose embeddings we want to vis
    """

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

    for i,word in enumerate(words):
        x = M_reduced[word2ind[word]][0]
        y = M_reduced[word2ind[word]][1]
        plt.scatter(x, y ,marker='x',color='red')
        plt.text(x, y ,word,fontsize=9)

    plt.xlabel("Embedding dimension 1")
    plt.ylabel("Embedding dimension 2")
    plt.title("Word Embeddings Visualization")
    plt.show()



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

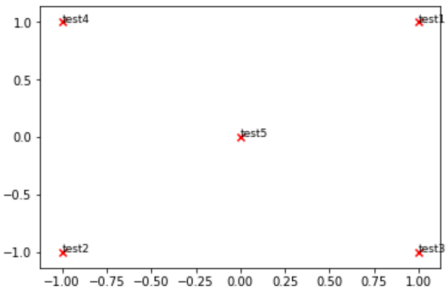
In [133]:
```
# ---------------------
# Run this sanity check
# Note that this is not an exhaustive check for correctness.
# The plot produced should look like the "test solution plot" depicted bel
# ---------------------

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, 'tes
words = ['test1', 'test2', 'test3', 'test4', 'test5']
plot_embeddings(M_reduced_plot_test, word2ind_plot_test, words)

print ("-" * 80)
```

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




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

Test Plot Solution



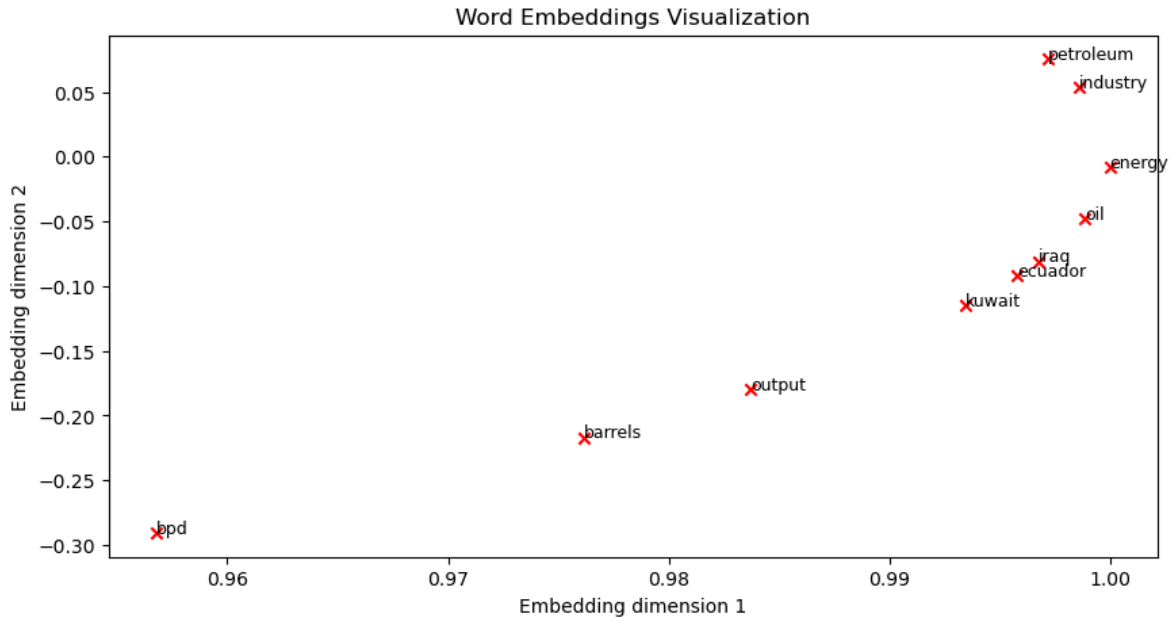
Question 1.5: Co-Occurrence Plot Analysis [written] (3 points)

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 [134]:
```
# ----------------------------
# Run This Cell to Produce Your Plot
# ----------------------------
reuters_corpus = read_corpus()
M_co_occurrence, word2ind_co_occurrence = compute_co_occurrence_matrix(reu
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] # broad

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

plot_embeddings(M_normalized, word2ind_co_occurrence, words)
```

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



"petroleum" and "industry" cluster together as expected, because I imagine this is the formal name for this industry that is commonly used in these documents. Iraq, Ecuador and Kuwait also form a cluster, and this makes sense since they can be listed one after each other. An unexpected result is how far away the word "output" is from energy or oil, as I believe they are contextually related and would expect them to be used together.

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.

In [135]:
```
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 [136]:
```
# ----------------------------------
# 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 [137]:
```
def get_matrix_of_vectors(wv_from_bin, required_words=['barrels', 'bpd',
    """ Put the GloVe vectors into a matrix M.
        Param:
            wv_from_bin: KeyedVectors object; the 400000 GloVe vectors loa
        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 [138]:
```
# -----------------------------------------------------------------
# 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] # broadcastin
```

```
Shuffling words ...
Putting 10000 words into word2ind and matrix M...
Done.
Running Truncated SVD over 10010 words...
Done.
```

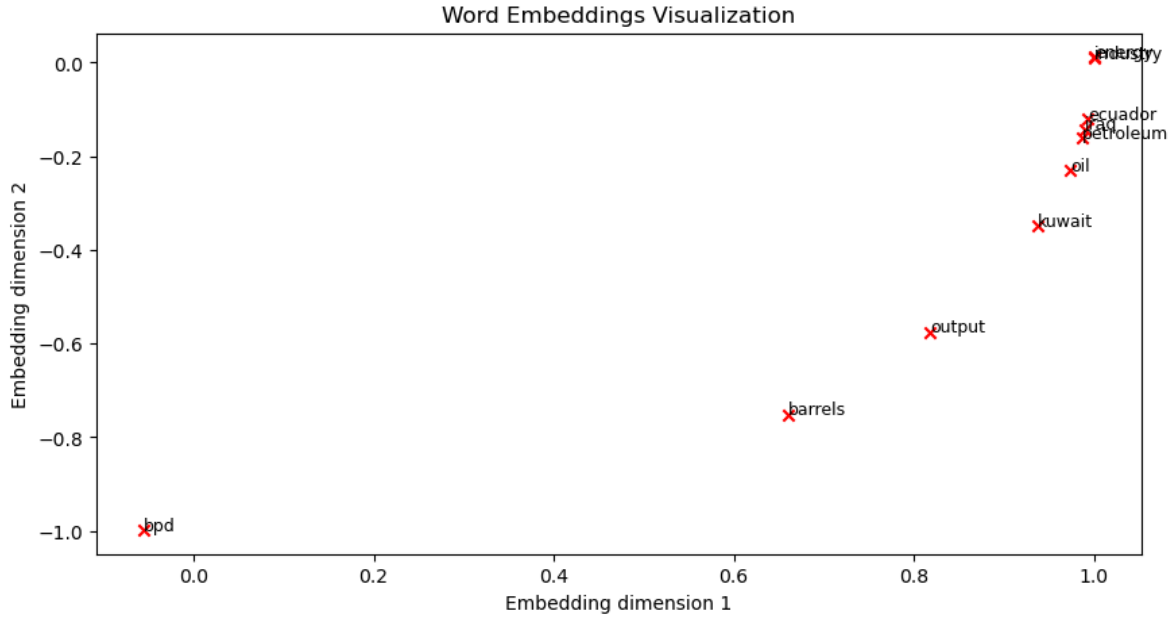
Note: If you are receiving out of memory issues on your local machine, try closing other applications to free more memory on your device. You may want to try restarting your machine so that you can free up extra memory. Then immediately run the jupyter notebook and see if you can load the word vectors 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 [139]:
```
words = ['barrels', 'bpd', 'ecuador', 'energy', 'industry', 'kuwait', 'oi
plot_embeddings(M_reduced_normalized, word2ind, words)
```

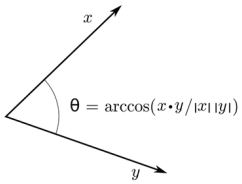


"ecuador", "iraq" and "petroleum" cluster together very closely in 2-d space. Also, "energy" and "industry" are on the same coordinates. I would expect "kuwait" to be clustered with "iraq" and "ecuador" as in the cooccurence matrix plot, but it the embeddings seem to have captured a difference between these countries. "bpd" is even further away in this plot, not forming a cluster with anything. The differences are caused by the ability of GloVe representation to better capture semantic relationships

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 (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.keyedvecto**

In [140]:
```
# ------------------
# Write your implementation here.

wv_from_bin.most_similar("nail")
# ------------------
```

Out[140]:
```
[('nails', 0.6682122945785522),
 ('biter', 0.6199564337730408),
 ('remover', 0.501641571521759),
 ('tooth', 0.4984681010246277),
 ('hammer', 0.49688461422920227),
 ('glue', 0.4765755832195282),
 ('polishes', 0.4695630967617035),
 ('toe', 0.46233314275741577),
 ('scissors', 0.460370808839798),
 ('biters', 0.4598196744918823)]
```

I tried "can" and "watch" before finding "nail". "can" did not work because there was a bunch of verbs that was linked to it semantically. "watch" did not work because it returned different conjugations of the word

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.keyedvecto** for further assistance.

In [141]:
```
# ------------------
# Write your implementation here.
w1 = "hot"
w2 = "warm"
w3 = "cold"
print(f'{wv_from_bin.distance(w1, w3):.4f} < {wv_from_bin.distance(w1, w2

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

```
0.4062 < 0.4112
```

the antonym "cold" is often used together with w1 "hot" in a context of comparison. As GloVe also makes use of co-occurence, this pattern can be captured because "hot" and "cold" appear in similar contexts more frequently than "hot" and "warm"

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.keyedvecto** 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 [142]:
```
# Run this cell to answer the analogy -- man : king :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'king'], negati
```

```
[('queen', 0.6978679299354553),
 ('princess', 0.6081743836402893),
 ('monarch', 0.5889754891395569),
 ('throne', 0.5775110125541687),
 ('prince', 0.5750998258590698),
 ('elizabeth', 0.5463595986366272),
 ('daughter', 0.5399126410484314),
 ('kingdom', 0.5318052768707275),
 ('mother', 0.5168544054031372),
 ('crown', 0.5164472460746765)]
```

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?

the expression which we are trying to maximize the similarity with is k - m + w, as if we are substracting the male gender from the word "king" and then inserting "woman" to the notion of that is represented by "king"

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 [143]:
```
# ------------------
# Write your implementation here.
pprint.pprint(wv_from_bin.most_similar(positive=['soccer', 'points'], nega
# ------------------
```

```
[('goals', 0.5983241200447083),
 ('ahead', 0.5734228491783142),
 ('draw', 0.566275417804718),
 ('2-0', 0.5491933822631836),
 ('up', 0.5348238348960876),
 ('2-1', 0.5346878170967102),
 ('goal', 0.5345394611358643),
 ('point', 0.5321381092071533),
 ('1-0', 0.5301719307899475),
 ('while', 0.5202596783638)]
```

basketball : points :: soccer : goals
points refers to the way of "scoring" in basketball, goals has the same meaning for soccer

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 [144]:
```
# ------------------
# Write your implementation here.

pprint.pprint(wv_from_bin.most_similar(positive=['see', 'nose'], negative=
# ------------------
```

```
[('you', 0.5465276837348938),
 ('can', 0.539147675037384),
 ("n't", 0.5341160297393799),
 ('right', 0.5323171615600586),
 ('?', 0.5299803018569946),
 ('if', 0.5269917249679565),
 ('ca', 0.5263591408729553),
 ('just', 0.5242089629173279),
 ('probably', 0.5232036113739014),
 ('i', 0.5229233503341675)]
```

eyes : see :: nose : smell
the functionality analogy in this case does not work.

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 [145]:
```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `negat
# most dissimilar from.
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'worker'], nega
print()
pprint.pprint(wv_from_bin.most_similar(positive=['man', 'worker'], negati
```

```
[('employee', 0.6375863552093506),
 ('workers', 0.6068919897079468),
 ('nurse', 0.5837946534156799),
 ('pregnant', 0.536388635635376),
 ('mother', 0.5321309566497803),
 ('employer', 0.5127025842666626),
 ('teacher', 0.5099576711654663),
 ('child', 0.5096741914749146),
 ('homemaker', 0.5019454956054688),
 ('nurses', 0.4970572292804718)]

[('workers', 0.611325740814209),
 ('employee', 0.5983108282089233),
 ('working', 0.5615329146385193),
 ('laborer', 0.5442320108413696),
 ('unemployed', 0.5368516445159912),
 ('job', 0.5278826951980591),
 ('work', 0.5223962664604187),
 ('mechanic', 0.5088937282562256),
 ('worked', 0.5054520964622498),
 ('factory', 0.4940453767776489)]
```

gender bias is obvious because we see "nurse", "pregnant", "mother" and "teacher" as words that are similar to "woman" but "unemployed", "meachanic" and "factory" for "man".
This reflects the bias in the corpus that the vectors were learned from.

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

In [146]:
```
# ------------------
# Write your implementation here.
pprint.pprint(wv_from_bin.most_similar(positive=['canada', 'immigrant'], n
print()
pprint.pprint(wv_from_bin.most_similar(positive=['america', 'immigrant'],
# ------------------
```

```
[('immigrants', 0.5841482877731323),
 ('immigration', 0.5370097160339355),
 ('canadian', 0.5285191535949707),
 ('quebec', 0.49370744824409485),
 ('migrant', 0.47261807322502136),
 ('canadians', 0.4712975323200226),
 ('ontario', 0.4507613480091095),
 ('ottawa', 0.4419090151786804),
 ('deported', 0.42593371868133545),
 ('migrants', 0.4252772927284241)]

[('immigrants', 0.6289525628089905),
 ('hispanic', 0.6204770803451538),
 ('latino', 0.5953384041786194),
 ('latinos', 0.49594351649284363),
 ('african-american', 0.49025407433509827),
 ('hispanics', 0.48906266689300537),
 ('undocumented', 0.4830229878425598),
 ('mexican', 0.48044121265411377),
 ('migrant', 0.4793946444988251),
 ('neighborhoods', 0.46772298216819763)]
```

The harmful stereotypes can be seen in this bias about immigrants in different countries

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?

The dataset contains biased representation of certain races or genders, thus the embeddings capture these meanings. To test this bias, we can compute the cosine similarity of given professions to the words "man" and "woman" and compare these.

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 [147]:
```
# ------------------
# Start your implementation here.
import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import DataLoader, Dataset
import os
import pandas as pd
from tqdm import tqdm
# ------------------
```

Data Import

- path declarations
- dataframe merges
- discretize labels
- split data

In [148]:
```
# Define data paths
data_dir = ".\\data\\stanfordSentimentTreebank"
dataset_sentences_path = os.path.join(data_dir, "datasetSentences.txt")
sentiment_labels_path = os.path.join(data_dir, "sentiment_labels.txt")
dictionary_path = os.path.join(data_dir, "dictionary.txt")
dataset_split_path = os.path.join(data_dir, "datasetSplit.txt")
```

In [149]:
```
dataset_sentences = pd.read_csv(dataset_sentences_path, sep='\t')
dataset_sentences.head()
```

Out[149]:

	sentence_index	sentence
0	1	The Rock is destined to be the 21st Century 's...
1	2	The gorgeously elaborate continuation of `` Th...
2	3	Effective but too-tepid biopic
3	4	If you sometimes like to go to the movies to h...
4	5	Emerges as something rare , an issue movie tha...

In [150]:
```
# Load phrase IDs and sentiment labels
sentiment_labels = pd.read_csv(sentiment_labels_path, sep='|')
sentiment_labels.head()
```

Out[150]:

	phrase ids	sentiment values
0	0	0.50000
1	1	0.50000
2	2	0.44444
3	3	0.50000
4	4	0.42708

In [151]:
```
# Load dictionary mapping phrases to phrase IDs
dictionary = pd.read_csv(dictionary_path, sep='|', names=["phrase", "phra
# Load dataset split (train/dev/test)
dataset_split = pd.read_csv(dataset_split_path)
# Merge the sentiment_labels with the dictionary to get sentiment scores
data = pd.merge(dictionary, sentiment_labels, left_on="phrase_id", right_
data.head()
```

Out[151]:

	phrase	phrase_id	phrase ids	sentiment values
0	!	0	0	0.50000
1	! '	22935	22935	0.52778
2	! "	18235	18235	0.50000
3	! Alas	179257	179257	0.44444
4	! Brilliant	22936	22936	0.86111

In [152]:
```
# Merge with sentences to get sentiment values for sentences
data = pd.merge(data, dataset_sentences, left_on="phrase", right_on="sente
data.head()

```

Out[152]:

	phrase	phrase_id	phrase ids	sentiment values	sentence_index	sentence
0	, The Sum of All Fears is simply a well-made a...	102340	102340	0.88889	4860	, The Sum of All Fears is simply a well-made a...
1	, `` They 're out there ! ''	221244	221244	0.61111	7251	, `` They 're out there ! ''
2	, is a temporal inquiry that shoulders its phi...	221388	221388	0.69444	5477	, is a temporal inquiry that shoulders its phi...
3	- I also wanted a little alien as a friend !	221714	221714	0.69444	5576	- I also wanted a little alien as a friend !
4	- West Coast rap wars , this modern mob music ...	221716	221716	0.76389	2338	- West Coast rap wars , this modern mob music ...

In [153]:
```
# Merge with dataset_split to get the split information
data = pd.merge(data, dataset_split, left_on="sentence_index", right_on="s

```

In [154]:
```
# Discretize sentiment scores into 5 classes
def discretize_sentiment(score):
    if score <= 0.2:
        return 0  # Very negative
    elif score <= 0.4:
        return 1  # Negative
    elif score <= 0.6:
        return 2  # Neutral
    elif score <= 0.8:
        return 3  # Positive
    else:
        return 4  # Very positive

data['sentiment_class'] = data['sentiment values'].apply(discretize_sentir
data = data[['sentence','splitset_label', 'sentiment_class']]

# Separate data into train, test, and dev sets
train_data = data[data['splitset_label'] == 1]
test_data = data[data['splitset_label'] == 2]
dev_data = data[data['splitset_label'] == 3]
```

Custom Dataset Class for DataLoader

- average of vectors computed here?

In [155]:
```
class SentimentDataset(Dataset):
    def __init__(self, sentences, labels, embeddings):
        self.sentences = sentences  # List of tokenized sentences as tens
        self.labels = labels        # List of sentiment classes as tensor
        self.embeddings = embeddings  # Pre-trained embeddings (e.g., Glo

    def __len__(self):
        return len(self.sentences)

    def __getitem__(self, idx):
        sentence = self.sentences[idx]
        # Convert words to embeddings and average them
        word_vectors = [self.embeddings[word] for word in sentence.split(
        #print(f'{word_vectors}')
        # Convert word_vectors (list of numpy.ndarray) into a list of PyT
        tensor_word_vectors = [torch.tensor(wv) for wv in word_vectors]
        if word_vectors:
            avg_vector = torch.mean(torch.stack(tensor_word_vectors), dim
        else:
            avg_vector = torch.zeros(200)  # Ensure a zero vector for mis
        label = self.labels[idx]
        return avg_vector, label
```

DAN Model

DAN with two nonlinear layers as in Figure 1 of the paper

- initialization
- flatten

In [156]:
```
class DAN(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.linear1 = nn.Linear(embedding_dim, hidden_dim)
        torch.nn.init.xavier_uniform_(self.linear1.weight)
        torch.nn.init.zeros_(self.linear1.bias)

        self.relu1 = nn.ReLU()

        self.linear2 = nn.Linear(hidden_dim, output_dim)
        torch.nn.init.xavier_uniform_(self.linear2.weight)
        torch.nn.init.zeros_(self.linear2.bias)

    def forward(self, x):
        # Average the word embeddings across the sentence?
        x = x.reshape(x.shape[0], -1)
        x = self.linear1(x)
        x = self.relu1(x)
        x = self.linear2(x)
        return x
```

Training

- declaration of required vars (embedding_dim etc.)
- train_loader
- model, optimizer, scheduler, criterion
- check_accuracy function
- Training Loop

In [157]:
```
embedding_dim = wv_from_bin.vector_size
hidden_dim = 300
output_dim = 5  # 5 sentiment classes
learning_rate = 1e-3
num_epochs = 5
```

In [158]:
```
# Convert data to lists of tokenized sentences and labels
train_sentences = train_data['sentence'].tolist()
train_labels = torch.tensor(train_data['sentiment_class'].values, dtype=t
# Initialize Dataset and DataLoader
train_dataset = SentimentDataset(train_sentences, train_labels, wv_from_b
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
```

In [159]:
```
# Initialize the model, loss function, and optimizer
model = DAN(embedding_dim, hidden_dim, output_dim)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
scheduler = optim.lr_scheduler.StepLR(optimizer, step_size=5, gamma=0.1)

criterion = nn.CrossEntropyLoss()
```

In [160]:
```
def check_accuracy(loader, model):

    model.eval()

    num_correct = 0
    num_samples = 0

    with torch.no_grad():
        for inputs, labels in loader:
            outputs = model(inputs)
            _, predicted = outputs.max(1)
            num_correct += (predicted == labels).sum()
            num_samples += labels.size(0)

    model.train()
    return num_correct / num_samples
```

Training Loop

In [161]:

```
mean_losses = []
model.train()
for epoch in range(num_epochs):
    losses = []

    for batch_idx, (inputs, labels) in enumerate(tqdm(train_loader)):

        # Forward pass
        outputs = model(inputs)
        loss = criterion(outputs, labels)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # Record loss
        losses.append(loss.item())


        mean_loss = sum(losses) / len(losses)
    losses.append(mean_loss)
    scheduler.step(mean_loss)
    print(f"Epoch: [{epoch+1}/{num_epochs}] | Loss: {mean_loss:.3f}")

print(f"Accuracy on training set: {check_accuracy(train_loader, model)*10
```

```
100%|████████████████████████████████████████████████████████████████
████████| 127/127 [00:06<00:00, 21.01it/s]
C:\Users\Tolga\anaconda3\envs\comp541\lib\site-packages\torch\optim\lr_schedu
ler.py:163: UserWarning: The epoch parameter in `scheduler.step()` was not ne
cessary and is being deprecated where possible. Please use `scheduler.step()`
to step the scheduler. During the deprecation, if epoch is different from Non
e, the closed form is used instead of the new chainable form, where availabl
e. Please open an issue if you are unable to replicate your use case: http
s://github.com/pytorch/pytorch/issues/new/choose. (https://github.com/pytorc
h/pytorch/issues/new/choose.)
  warnings.warn(EPOCH_DEPRECATION_WARNING, UserWarning)

Epoch: [1/5] | Loss: 1.475

100%|████████████████████████████████████████████████████████████████
████████| 127/127 [00:06<00:00, 20.77it/s]

Epoch: [2/5] | Loss: 1.359

100%|████████████████████████████████████████████████████████████████
████████| 127/127 [00:05<00:00, 21.31it/s]

Epoch: [3/5] | Loss: 1.329

100%|████████████████████████████████████████████████████████████████
████████| 127/127 [00:06<00:00, 21.00it/s]

Epoch: [4/5] | Loss: 1.305

100%|████████████████████████████████████████████████████████████████
████████| 127/127 [00:06<00:00, 21.03it/s]
```

```
Epoch: [5/5] | Loss: 1.295
Accuracy on training set: 44.97
```

Evaluation using test data

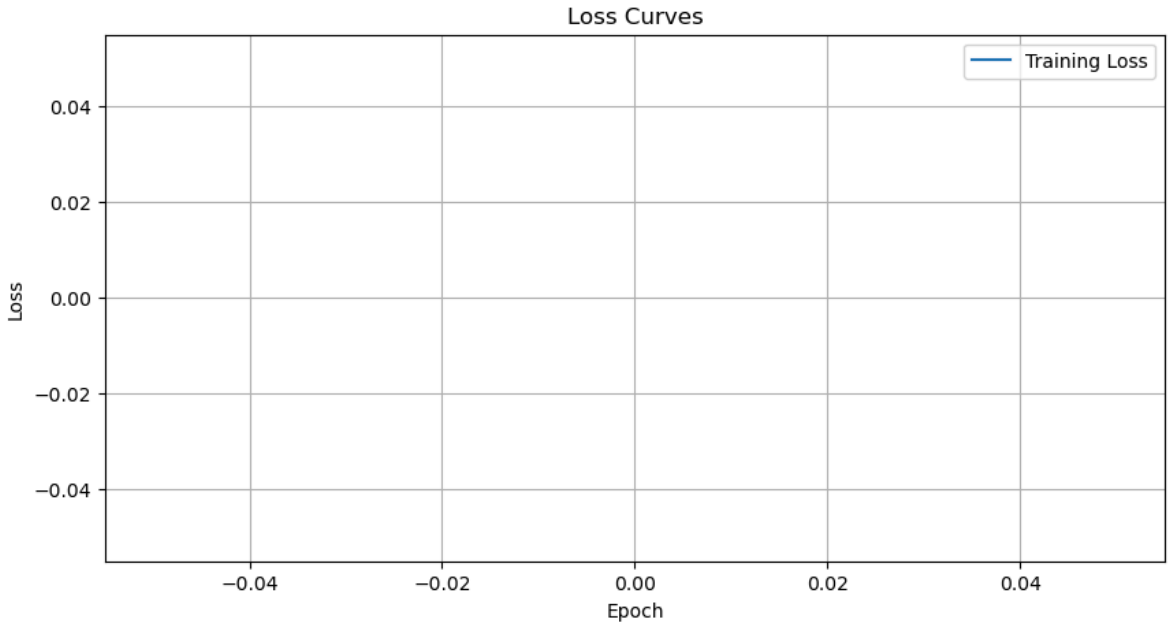
- test_loader
- check_accuracy for test

In [162]:
```python
# Convert data to lists of tokenized sentences and labels
test_sentences = test_data['sentence'].tolist()
test_labels = torch.tensor(test_data['sentiment_class'].values, dtype=tor
# Initialize Dataset and DataLoader
test_dataset = SentimentDataset(test_sentences, test_labels, wv_from_bin)
test_loader = DataLoader(test_dataset, batch_size=64, shuffle=True)
```

In [163]:
```python
print(f"Accuracy on test set: {check_accuracy(test_loader, model)*100:.2f
```

```
Accuracy on test set: 41.27
```

In [164]:
```python
plt.figure(figsize=(10, 5))
plt.plot(mean_losses, label="Training Loss")

plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.title("Loss Curves")
plt.legend()
plt.grid()
plt.show()
```



I did not manage to run with cooccurence embeddings.
We can see the decrease in training loss over the epochs. The accuracy over the test test is around 40%, which is acceptible given that this is a 2-layer network for a 5-class classification

problem.

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/)