exam 개인학습

Lec 1

AI summary Try on this page

Type something...

01) Introduction to word vectors

01_환경설정

CS224N Assignments

Solutions for cs224n(Deep Learning For Natural Language Processing) course by stanford university

Prerequisites

1. Install Anaconda
2. go to assignmentX where X is either 1, 2, 3 using a Terminal:

```
$ cd \path\to\assignment1
```

3. create a python 2.7 environnment using

```
$ conda env -n cs224n python=2.7 anaconda
```

4. activate your environment using (add source before activate if you're working with Linux/Mac)

```
$ activate cs224n
```

5. install the dependencies using requirements.txt

```
$ pip install -r requirements.txt
```

6. Don't forget to deactivate your environment when you're done (add source before deactivate if you're on Linux/Mac)

```
$ deactivate cs224n
```

- 아나콘다 가상환경을 설정해줘야하기 때문에 폴더위치를 옮겨서 진행해보자.

```
(base) C:₩Users₩user>cd C:Downloads
```

```
(base) C:₩Users₩user₩Downloads>cd C:a1

(base) C:₩Users₩user₩Downloads₩a1>dir
 C 드라이브의 볼륨: OS
 볼륨 일련 번호: B217-3C45

 C:₩Users₩user₩Downloads₩a1 디렉터리

2024-12-04  오후 01:17    <DIR>          .
2024-12-04  오후 01:17    <DIR>          ..
2024-12-04  오후 01:17    <DIR>          student
2024-12-04  오후 01:17    <DIR>          __MACOSX
               0개 파일                   0 바이트
               4개 디렉터리  322,080,669,696 바이트 남음

(base) C:₩Users₩user₩Downloads₩a1>
```

- a1 폴더에 들어와서 student로 들어간다.

```
(base) C:₩Users₩user₩Downloads₩a1>cd C:student

(base) C:₩Users₩user₩Downloads₩a1₩student>dir
 C 드라이브의 볼륨: OS
 볼륨 일련 번호: B217-3C45

 C:₩Users₩user₩Downloads₩a1₩student 디렉터리

2024-12-04  오후 01:17    <DIR>          .
2024-12-04  오후 01:17    <DIR>          ..
2024-12-04  오후 01:17             8,196 .DS_Store
2024-12-04  오후 01:17               197 env.yml
2024-12-04  오후 01:17            52,976 exploring_word_vectors.ipynb
2024-12-04  오후 01:17    <DIR>          imgs
2024-12-04  오후 01:17             1,306 README.md
               4개 파일              62,675 바이트
               3개 디렉터리  322,145,746,944 바이트 남음

(base) C:₩Users₩user₩Downloads₩a1₩student>
```

- `conda env create -f env.yml` 을 입력해서 `env.yml` 에 의존하는 환경을 만든다.
- 그 후 `conda activate cs224n` 을 입력하여 새로운 환경을 활성화한다.

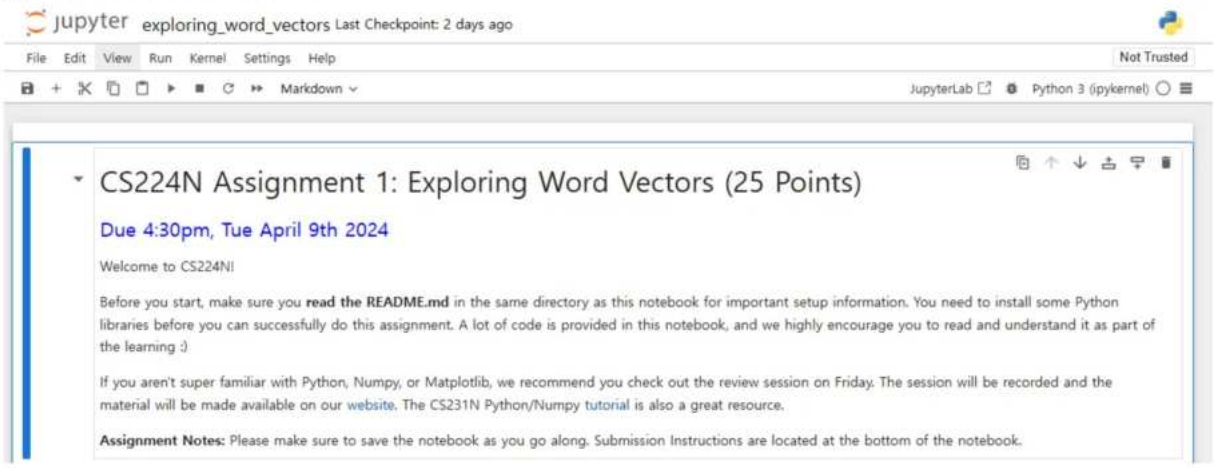
```
(base) C:₩Users₩user₩Downloads₩a1₩student>conda activate cs224n

(cs224n) C:₩Users₩user₩Downloads₩a1₩student>
```

```
cs224n) C:₩Users₩user₩Downloads₩a1₩student>python -m ipykernel install --user --name cs224n
nstalled kernelspec cs224n in C:₩Users₩user₩AppData₩Roaming₩jupyter₩kernels₩cs224n
```

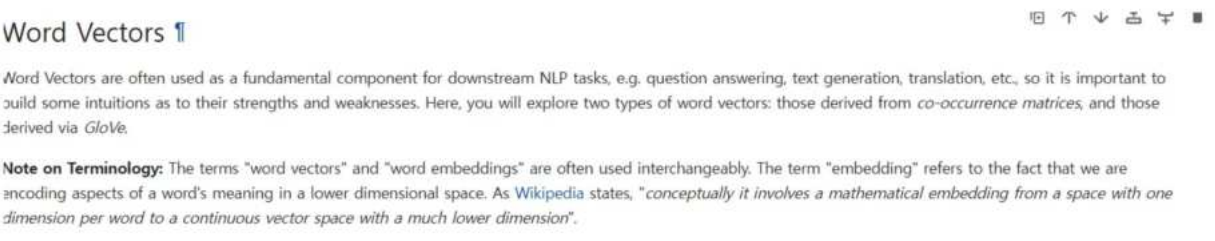
- Ipython을 설치해서 jupyter 환경을 만들어준다.
- 이제 `jupyter notebook exploring_word_vectors.ipynb` 을 실행해보자.

```
cs224n) C:₩Users₩user₩Downloads₩a1₩student>jupyter notebook exploring_word_vectors.ipynb
₩ 2024-12-07 11:46:51.817 ServerApp] A `_jupyter_server_extension_points` function was not found in jupyter_lsp. In
tead, a `_jupyter_server_extension_paths` function was found and will be used for now. This function name will be d
precated in future releases of Jupyter Server.
```

- 그러면 이제 jupyter 서버가 열려서 코딩을 할 수 있게 된다.



- 이제부터 본격적으로 과제에 들어가보자.



- Word vectors는 NLP에 근본적인 구성요소로 우리는 2가지 word vector를 볼 것이다.
- 하나는 co-occurence matrices(동시 발생 matrix), 다른 하나는 GloVe이다.

02_Part 1: Count-Based Word Vectors(10 points)

Part 1: Count-Based Word Vectors (10 points)

Most word vector models start from the following idea:

You shall know a word by the company it keeps (Firth, J. R. 1957:11)

Many word vector implementations are driven by the idea that similar words, i.e., (near) synonyms, will be used in similar contexts. As a result, similar words will often be spoken or written along with a shared subset of words, i.e., contexts. By examining these contexts, we can try to develop embeddings for our words. With this intuition in mind, many "old school" approaches to constructing word vectors relied on word counts. Here we elaborate upon one of those strategies, *co-occurrence matrices* (for more information, see here or here).

- 대부분의 word vector들은 synonyms(동의어)를 기반으로 동작하였다.
- 결과적으로 비슷한 단어는 비슷한 맥락에서 사용된다는 것이다.
- 과거에는 word counts, 단어의 빈도수를 기반으로 word vector를 구성하였다.
- 대표적인 방법이 co-occurrence matrices, 동시 발생 행렬로 서로가 동시에 나온 빈도수를 행렬에 저장하는 것이다.
- 이 행렬은 대칭적이지만 매우 sparse하다는 단점이 있다.

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

- window size = 1인 상황에서 한번 살펴보자.
- all that glitters is not gold, all is well that ends well
- 두 문장에서 all이 <START>와 붙어있기 때문에 count = 2가 되었다.
- 그리고 that과 is가 각각 1번씩 붙어있었기 때문에 count = 1이 되었다.
- 이 과정을 반복하여 행렬을 완성시켜주었다.

The matrix rows (or columns) provide word vectors based on word-word co-occurrence, but they can be large. To reduce dimensionality, we employ Singular Value Decomposition (SVD), akin to PCA, selecting the top k principal components. The SVD process decomposes the co-occurrence matrix A into singular values in the diagonal S matrix and new, shorter word vectors in U_k.

This dimensionality reduction maintains semantic relationships; for instance, *doctor* and *hospital* will be closer than *doctor* and *dog*.

For those unfamiliar with eigenvalues and SVD, a beginner-friendly introduction to SVD is available here. Additional resources for in-depth understanding include lectures 7, 8, and 9 of CS168, providing high-level treatment of these algorithms. For practical implementation, utilizing pre-programmed functions from Python packages like numpy, scipy, or sklearn is recommended. While applying full SVD to large corpora can be memory-intensive, scalable techniques such as Truncated SVD exist for extracting the top k vector components efficiently.

- 이 행렬은 매우 크기 때문에 차원을 줄여줄 필요가 있다.
- 그래서 우리는 SVD, PCA를 사용할 것이다.
- 주성분 K개만 뽑아서 차원을 축소시킨다.
- 차원 축소는 의미적 관계를 유지시킨다.
- Full SVD는 비용이 너무 비싸기 때문에 우리는 Truncated SVD를 사용하여 효율적으로 k개만 추출할 것이다.

Plotting Co-Occurrence Word Embeddings

Here, we will be using the Large Movie Review Dataset. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. We provide a set of 25,000 highly polar movie reviews for training, and 25,000 for testing. There is additional unlabeled data for use as well. We provide a `read_corpus` function below that pulls out the text of a movie review from the dataset. 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.

- 이번 과제에서 사용할 데이터셋은 영화 감상평이다.
- 이 데이터셋은 이진 감정 분류로
- 25,000 Train, 25,000 Test, Unlabeled된 data도 존재한다.
- read_corpus함수를 통해 dataset에서 review text를 추출할 수 있게 한다. 또한 <START> 와 <END> 토큰을 양끝에 넣을 것이다.

- 그래서 우리는 따로 전처리를 할 필요가 없다.

2.1 read_corpus

```
def read_corpus():
    """ Read files from the Large Movie Review Dataset.
        Params:
            category (string): category name
        Return:
            list of lists, with words from each of the processed files
    """
    files = imdb_dataset["train"]["text"][:NUM_SAMPLES]
    return [[START_TOKEN] + [re.sub(r'[^\w]', '', w.lower()) for w in f.split(" ")] + [END_TOKEN] for f in files]
```

- `re.sub` : 문자열을 대체하는 함수
- ₩w: 문자, ^: not
- files에서 문장 단위로 f를 뽑는다.
- 문장 f에서 공백 단위로 단어 w를 뽑는다.
- `-` : range
- `[\w]` = `[0-9a-zA-Z]` → 0~9사이 or a~z사이 or A~Z 사이 → 문자나 숫자만 입력을 받겠다는 의미.

r prefix

- String and Bytes prefix에 대한 공식 문서

아래 예제를 보면 아주 직관적이고 쉽게 이해할 수 있다!

```
print("Hello World!\n")
print(r"Hello World!\n")
```

```
>> Hello World!
>>
>> Hello World!\n
```

문자열 앞에 붙이는 `r`은 `raw string`의 의미를 가지며, 구체적인 의미는 `\`을 **탈출 문자로 보지 않고**, 그냥 아무 역할도 하지 않는 평범한 문자열로 간주하여 처리하겠다는 뜻이다.

- 앞에 붙이는 `r` 은 raw string으로 ₩을 탈출문자로 보지 않고, 그냥 아무 역할도 하지 않은 평범한 문자열로 간주하여 처리하겠다는 뜻이다.
- **정리하면 각 단어에서 숫자나 문자가 아닌 것을 공백으로 바꿔주는 작업을 하고 [START]토큰과 [END]토큰을 양끝에 삽입하겠다는 함수이다.**

```
imdb_corpus = read_corpus()
pprint.pprint(imdb_corpus[:3], compact=True, width=100)
print("corpus size: ", len(imdb_corpus[0]))
```

```
[['<START>', 'i', 'rented', 'i', 'am', 'curiousyellow', 'from', 'my', 'video', 'store', 'because',
  'of', 'all', 'the', 'controversy', 'that', 'surrounded', 'it', 'when', 'it', 'was', 'first',
  'released', 'in', '1967', 'i', 'also', 'heard', 'that', 'at', 'first', 'it', 'was', 'seized',
  'by', 'us', 'customs', 'if', 'it', 'ever', 'tried', 'to', 'enter', 'this', 'country', 'therefore',
  'being', 'a', 'fan', 'of', 'films', 'considered', 'controversial', 'i', 'really', 'had', 'to',
  'see', 'this', 'for', 'myselfbr', 'br', 'the', 'plot', 'is', 'centered', 'around', 'a', 'young',
  'swedish', 'drama', 'student', 'named', 'lena', 'who', 'wants', 'to', 'learn', 'everything',
  'she', 'can', 'about', 'life', 'in', 'particular', 'she', 'wants', 'to', 'focus', 'her',
  'attentions', 'to', 'making', 'some', 'sort', 'of', 'documentary', 'on', 'what', 'the', 'average',
  'swede', 'thought', 'about', 'certain', 'political', 'issues', 'such', 'as', 'the', 'vietnam',
  'war', 'and', 'race', 'issues', 'in', 'the', 'united', 'states', 'in', 'between', 'asking',
  'politicians', 'and', 'ordinary', 'denizens', 'of', 'stockholm', 'about', 'their', 'opinions',
  'on', 'politics', 'she', 'has', 'sex', 'with', 'her', 'drama', 'teacher', 'classmates', 'and',
  'married', 'menbr', 'br', 'what', 'kills', 'me', 'about', 'i', 'am', 'curiousyellow', 'is',
  'that', '40', 'years', 'ago', 'this', 'was', 'considered', 'pornographic', 'really', 'the', 'sex',
  'and', 'nudity', 'scenes', 'are', 'few', 'and', 'far', 'between', 'even', 'then', 'its', 'not',
  'shot', 'like', 'some', 'cheaply', 'made', 'porno', 'while', 'my', 'countrymen', 'mind', 'find',
  'it', 'shocking', 'in', 'reality', 'sex', 'and', 'nudity', 'are', 'a', 'major', 'staple', 'in',
  'swedish', 'cinema', 'even', 'ingmar', 'bergman', 'arguably', 'their', 'answer', 'to', 'good',
```

- `pprint`를 이용하면 문자열을 보기 좋게 출력할 수 있다.
- compact = True로 하여 문자열을 줄을 띄우지 않고 일자로 출력해주었다.

2.2 Question 1.1: Implement `distinct_words` (2 points)

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 use `for` loops to process the input `corpus` (a list of list of strings), but try using Python list comprehensions (which are generally faster). In particular, this may be useful to flatten a list of lists. If you're not familiar with Python list comprehensions in general, here's more information.

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 to remove duplicate words.

- corpus에서 발생하는 distinct words를 만드는 함수이다.
- 단어는 중복될 수 없고 sorted되어 있어야 한다.

```
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 across the corpus
            n_corpus_words (integer): number of distinct words across the corpus
    """
    corpus_words = []
    n_corpus_words = -1

    # ------------------
    # Write your implementation here.
    for f in corpus:
        for word in f:
            if word not in corpus_words:
                corpus_words.append(word)
    corpus_words.sort()
    n_corpus_words = len(corpus_words)
    # ------------------

    return corpus_words, n_corpus_words
```

- 반복문을 돌리면서 리스트 내에 중복으로 있는지 확인하고 추가해주었다.
- sort로 정렬을 해주었다.
- 간단하게 list comprehensions을 이용하여 코드를 작성하면 다음과 같다.

```
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 across the corpus
            n_corpus_words (integer): number of distinct words across the corpus
    """
    corpus_words = []
    n_corpus_words = -1

    # ------------------
    # Write your implementation here.
    corpus_words = sorted(list(set(word for docs in corpus for word in docs)))
    n_corpus_words = len(corpus_words)
    # ------------------

    return corpus_words, n_corpus_words
```

- set을 이용하여 중복을 제거해주고 sorted를 이용하여 정렬해주었다.
- list comprehension은 다중 루프도 지원해주기 때문에 처음 for문에는 큰 덩어리를 쪼개주고 그 다음에 세부적인 덩어리를 쪼개는 for문으로 작성한다.
- corpus → docs → word

2.3 Question 1.2: Implement `compute_co_occurrence_matrix` (3 points)

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

Write a method that constructs a co-occurrence matrix for a certain window-size n (with a default of 4), considering words n before and n after the word in the center of the window. Here, we start to use `numpy (np)` to represent vectors, matrices, and tensors. If you're not familiar with NumPy, there's a NumPy tutorial in the second half of this cs231n Python NumPy tutorial.

- window-size가 n(default = 4)인 co-occurence matrix를 만드는 method를 구축할 것이다.
- n개 이전의 단어와 n개 이후의 단어를 중심 단어를 비교해서 행렬에 count해준다.
- 대칭 행렬이기 때문에 꼭 반대도 값을 넣어주어야 한다.
- numpy를 사용하여 진행할 것이다.

```
""" Compute co-occurrence matrix for the given corpus and window_size (default of 4).
```

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

          For example, if we take the document "<START> All that glitters is not gold <END>" with window size of 4,
          "All" will co-occur with "<START>", "that", "glitters", "is", and "not".

    Params:
        corpus (list of list of strings): corpus of documents
        window_size (int): size of context window
    Return:
        M (a symmetric numpy matrix of shape (number of unique words in the corpus , number of unique words in the corpus)):
            Co-occurence matrix of word counts.
            The ordering of the words in the rows/columns should be the same as the ordering of the words given by the distinct_words function.
        word2ind (dict): dictionary that maps word to index (i.e. row/column number) for matrix M.
    """
```

- 문장 단위로 진행할 것이다.
- 순서는 이전에 만든 distinct_words function의 순서를 따라갈 것이다.
- 인덱스가 넘어가지 않게 주의하자.
- `M` 은 co-occurence matrix이고, `word2ind` 는 단어를 index로 바꿔주는 dictionary이다. M을 만들기 위해 사용될 것이다.
- 앞에서 만든 distnct_words function을 이용해서 단어 순서를 인덱스로 부여해주자.
- 0 ~ `n_corpus_words - 1` 까지 인덱스를 부여할 것이다.
- 2가지를 이용하여 코드를 작성해보자.

```
{'<END>': 0, '<START>': 1, 'All': 2, "All's": 3, 'ends': 4, 'glitters': 5, 'gold': 6, "isn't": 7, 'that': 8, 'well': 9}
```

- `word2ind` 은 위와 같이 word → index로 mapping되어있다.

```
['<END>', '<START>', 'All', "All's", 'ends', 'glitters', 'gold', "isn't", 'that', 'well']
```

- 단어의 순서는 올바르게 인덱싱 된 것을 확인할 수 있다.
- 같은 문장에서 여러번 겹친다고 여러번 count해서는 안된다.
- ex) `[START]` 와 `All` 이 [START]가 중심일 때 1번, All이 중심일 때 1번 count되어서 2번 중복 카운트가 되면 안된다.

```python
# word2ind
corpus_words, n_corpus_words = distinct_words(corpus)
for i in range(n_corpus_words):
    word2ind[corpus_words[i]] = i

# M(Co-occurence matrix of word counts)
M = np.zeros((n_corpus_words, n_corpus_words))

for docs in corpus:
    for j in range(len(docs)):
        # center word기준으로 +- window size만큼 떨어져있는거 구하기
        # 인덱스 범위에 유의하자.
        center_word = docs[j]
        center_idx = word2ind[center_word] # 중심단어 인덱스 부여해주기.
        # +-1, +-2
        # 근처에 있으면 co-ocurrence matrix에다가 +=1 해주기
        for k in range(1, window_size + 1):
            if j - k >= 0: # 0을 벗어나지는 않았는가?
                outer_word = docs[j - k]
                outer_idx = word2ind[outer_word]
                M[center_idx][outer_idx] += 1
            if j + k < len(docs): # 마지막 인덱스를 넘어가지는 않았는가?
                outer_word = docs[j + k]
                outer_idx = word2ind[outer_word]
                M[center_idx][outer_idx] += 1
```

- 처음에는 중복을 고려해서 양쪽에다가 +=1을 해주었는데 그렇게 되니 중복으로 카운트가 되는 문제가 발생하였다.
- 그래서 center를 기준으로만 카운트를 진행해서 작성하였더니 test를 통과할 수 있었다.

2.4 Question 1.3: Implement `reduce_to_k_dim` (1 point)

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.

- SVD를 사용하여 상위 k개의 components를 추출하는 method이다.
- Truncated SVD를 사용하여 진행해보자.

```
def reduce_to_k_dim(M, k=2):
    """ Reduce a co-occurence count matrix of dimensionality (num_corpus_words, num_corpus_words)
        to a matrix of dimensionality (num_corpus_words, k) using the following SVD function from Scikit-Learn:
            - http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

        Params:
            M (numpy matrix of shape (number of unique words in the corpus , number of unique words in the corpus)): co-occurence matrix of word counts
            k (int): embedding size of each word after dimension reduction
        Return:
            M_reduced (numpy matrix of shape (number of corpus words, k)): matrix of k-dimensioal word embeddings.
                    In terms of the SVD from math class, this actually returns U * S
```

- `(num_corpus_words, num_corpus_words)` → `(num_corpus_words, k)`

```
# ------------------
# Write your implementation here.
truncatedSVD = TruncatedSVD(n_components = k, n_iter = n_iters)
M_reduced = truncatedSVD.fit_transform(M)
# ------------------
```

- `TruncatedSVD` 객체를 하나 만들고 `fit_transform` 을 통해서 행렬을 차원축소 시켜준다.

2.5 Question 1.4: Implement `plot_embeddings` (1 point)

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

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

For this example, you may find it useful to adapt this code. In the future, a good way to make a plot is to look at the Matplotlib gallery, find a plot that looks somewhat like what you want, and adapt the code they give.

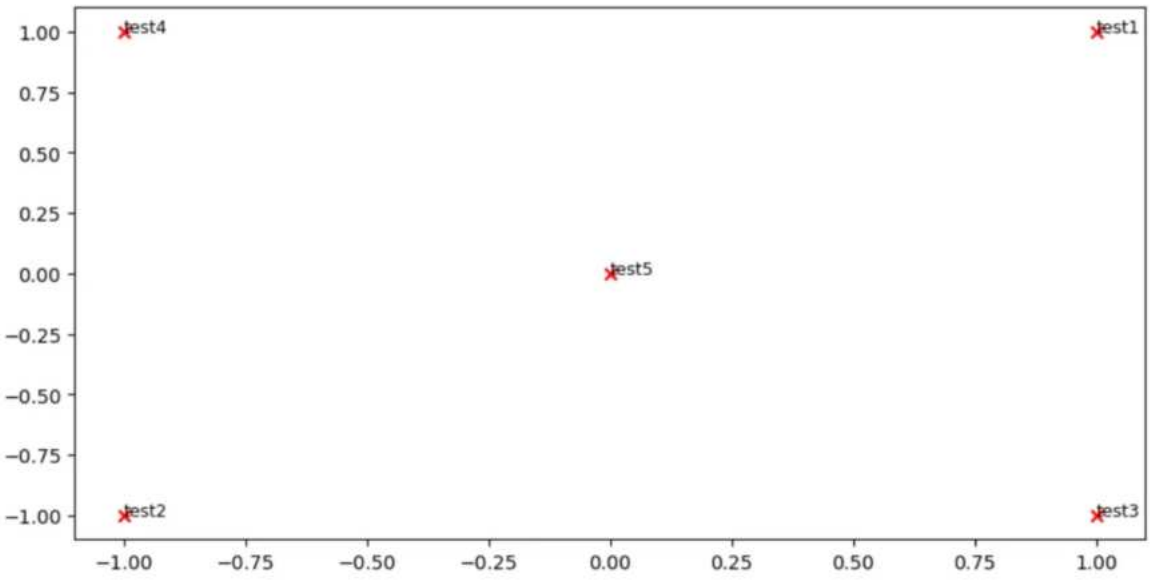
- `Matplotlib.pyplot` 를 이용해서 2차원 그래프를 그릴 것이다.

```
def plot_embeddings(M_reduced, word2ind, words):
    """ Plot in a scatterplot the embeddings of the words specified in the list "words".
        NOTE: do not plot all the words listed in M_reduced / word2ind.
        Include a label next to each point.

        Params:
            M_reduced (numpy matrix of shape (number of unique words in the corpus , 2)): matrix of 2-dimensioal word embeddings
            word2ind (dict): dictionary that maps word to indices for matrix M
            words (list of strings): words whose embeddings we want to visualize
    """
```

- Scatterplot(산점도)를 그릴 것이다.
- `M_reduced` : word embedding 각각 2차원을 가진 행렬
- `word2ind` : word → index
- `words` : 시각화해야하는 단어

```
    # ------------------
    # Write your implementation here.
    for i, word in enumerate(words):
        idx = word2ind[word]
        x = M_reduced[idx][0]
        y = M_reduced[idx][1]
        plt.scatter(x, y, marker = 'x', color = 'red')
        plt.text(x, y, word, fontsize = 9)
    plt.show()
```

- `M_reduced` 를 통해서 좌표를 뽑아주고 scatter를 그려준다.
- 그 후 `plt.text` 를 통해서 text를 적어준다.



- 결과물을 다음과 같이 나오게 된다.

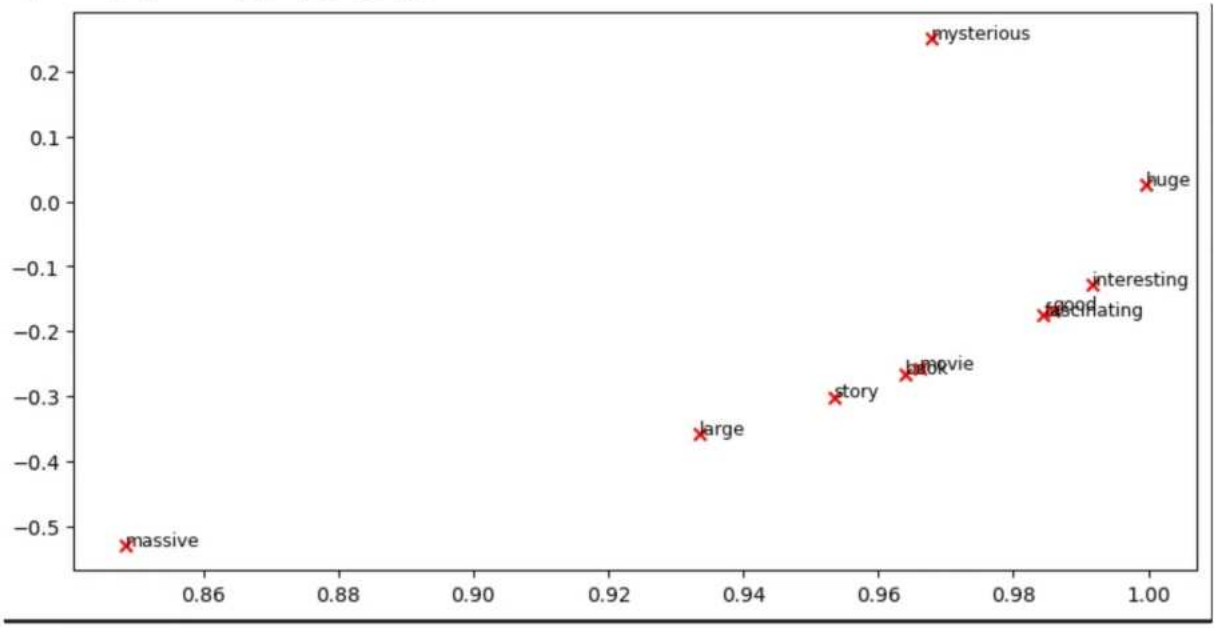
2.5 Question 1.4: Implement `plot_embeddings` (1 point)

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 Large Movie Review 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.

Run the below cell to produce the plot. It can take up to a few minutes to run.

- window size = 4인 co-occurence matrix를 계산하고 TruncatedSVD를 사용하여 각각의 단어를 2차원 embedding vector로 만들어주었다.
- return된 vector를 normalize를 하여 unit circle주위에 그려질 수 있게 하였다.



a. Find at least two groups of words that cluster together in 2-dimensional embedding space. Give an explanation for each cluster you observe.

Write your answer here.

b. What doesn't cluster together that you might think should have? Describe at least two examples.

Write your answer here.

a. [interesting, fascinating]이 같은 무리이고, [book, movie]가 같은 무리이다.

b. mysterious한 것과 massive한 것은 동떨어져 있다.

03_Part 2: Prediction-Based Word Vectors (15 points)

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.

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.

- 더 나은 퍼포먼스를 위해서는 word2vec와 GloVe가 있다.
- 우리는 여기서 GloVe를 사용할 것이다.

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
P(k | ice)	큰 값	작은 값	큰 값	작은 값
P(k | steam)	작은 값	큰 값	큰 값	작은 값
P(k | ice) / P(k | steam)	큰 값	작은 값	1에 가까움	1에 가까움

- GloVe는 임베딩된 중심 단어와 주변 단어 벡터의 내적이 전체 Corpus에서의 동시 등장 확률이 되도록 만드는 것이다.

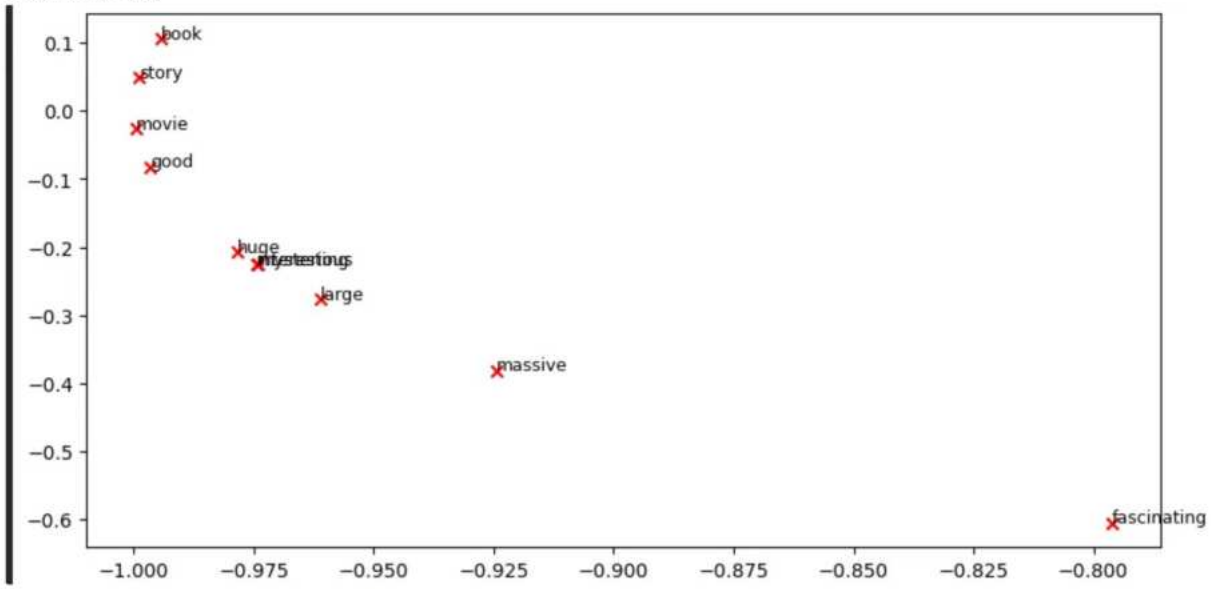
$$exp(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

- 두 벡터를 내적한 값에 exp를 취하면 확률이 된다는 사실을 저번 정리를 통해 알게 되었다.
- X: co-occurence matrix

```
def load_embedding_model():
    """ Load GloVe Vectors
        Return:
            wv_from_bin: All 400000 embeddings, each length 200
    """
    import gensim.downloader as api
    wv_from_bin = api.load("glove-wiki-gigaword-200")
    print("Loaded vocab size %i" % len(list(wv_from_bin.index_to_key)))
    return wv_from_bin
wv_from_bin = load_embedding_model()
```

- `gensim.downloader as api` 에서 glove를 model를 load한다.

3.1 Glove Plot analysis(3 points)



a. What is one way the plot is different from the one generated earlier from the co-occurrence matrix? What is one way it's similar?

Write your answer here.

b. Why might the GloVe plot (question_2.1.png) differ from the plot generated earlier from the co-occurrence matrix (question_1.5.png)?

Write your answer here.

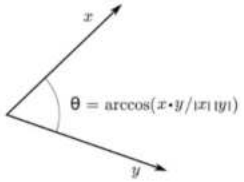
a. x좌표가 양수 → 음수로 변하였고 y좌표도 음수 → 양수로 변하였다.

b. GloVe는 wikipedia를 기반으로 학습을 진행하여 scale에 차이가 생겼다. 또한 fascinating이 동떨어져있다.

Cosine Similarity

Now that we have word vectors, we need a way to quantify the similarity between individual words, according to these vectors. One such metric is cosine-similarity. We will be using this to find words that are "close" and "far" from one another.

We can think of n-dimensional vectors as points in n-dimensional space. If we take this perspective L1 and L2 Distances help quantify the amount of space "we must travel" to get between these two points. Another approach is to examine the angle between two vectors. From trigonometry we know that:



Instead of computing the actual angle, we can leave the similarity in terms of $similarity = cos(\Theta)$. Formally the Cosine Similarity s between two vectors p and q is defined as:

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

- cosin similarity는 두 벡터가 얼마나 닮았는지를 확인하는 지표이다.
- 1에 가까울수록 닮았고, 0에 가까울수록 닮지 않았다.

3.2 Question 2.2: Words with Multiple Meanings (1.5 points)

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 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 most 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**.

- 다의어와 동음이의어는 한가지 이상의 의미를 가지는 단어이다.
- 적어도 2개의 다른 의미를 가진 단어와 비슷한 의미를 가진 단어 상위 10개를 찾아보자.

In [59]:
```
### SOLUTION BEGIN
wv_from_bin.most_similar('close')
### SOLUTION END
```

Out[59]:
```
[('up', 0.6898429989814758),
 ('closed', 0.6854836940765381),
 ('down', 0.6749923229217529),
 ('while', 0.6741016507148743),
 ('just', 0.6566178798675537),
 ('but', 0.6522998213768005),
 ('closing', 0.6294093728065491),
 ('point', 0.6282009482383728),
 ('far', 0.6234282851219177),
 ('time', 0.6206083893775940)]
```

SOLUTION BEGIN

Homonym: **close** (*near* / *not open*). Related words:

Meaning 1	Meaning 2
up	closed
while	down
but	just
point	closing
far	time

- 단어가 여러개의 의미를 가지고 있으면 잘 동작하지 않는다.
- 몇몇 동음이의어에 경우 하나의 의미가 다른 하나보다 빈번하게 쓰이기 때문이다.

3.3 Question 2.3: Synonyms & Antonyms (2 points)

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** for further assistance.

- w1. w2는 동의어이고, w1과 w3는 반의어이다.

[59]:
```
# ------------------
# Write your implementation here.
w1 = 'tall'
w2 = 'high'
w3 = 'short'
w1_w2_dist = wv_from_bin.distance(w1, w2)
w1_w3_dist = wv_from_bin.distance(w1, w3)

print("Synonyms {}, {} have cosine distance: {}".format(w1, w2, w1_w2_dist))
print("Antonyms {}, {} have cosine distance: {}".format(w1, w3, w1_w3_dist))
# ------------------
```

```
Synonyms tall, high have cosine distance: 0.6528714299201965
Antonyms tall, short have cosine distance: 0.6098593175411224
```

- 동의어끼리의 코사인 유사도가 반의어끼리의 코사인유사도보다 더 크다.

3.4 Question 2.4: Analogies with word Vectors

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 : grandfather :: woman : x" (read: man is to grandfather 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**. 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). The answer to the analogy will have the highest cosine similarity (largest returned numerical value).

60]:
```
# Run this cell to answer the analogy -- man : grandfather :: woman : x
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'grandfather'], negative=['man']))
```

```
[('grandmother', 0.7608445286750793),
 ('granddaughter', 0.7200808525085449),
 ('daughter', 0.7168302536010742),
 ('mother', 0.7151536345481873),
 ('niece', 0.7005682587623596),
 ('father', 0.6659888029098511),
 ('aunt', 0.6623408794403076),
 ('grandson', 0.6618767380714417),
 ('grandparents', 0.6446609497070312),
 ('wife', 0.6445354223251343)]
```

- word vector 유추를 한번 보자.
- `most_similar` 함수는 이 x를 구할 수 있게 해준다.

Let m, g, w, and x denote the word vectors for `man`, `grandfather`, `woman`, and the answer, respectively. Using **only** vectors m, g, 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 `grandfather` and the answer?

- m: man, g: grandfather, w: woman, x
- **x = w + g - m**
- 남자인 부분을 빼고 여자인 부분을 더하면 grandmother로 유추할 수 있다.

3.5 Question 2.5: Finding Analogies(1.5 points)

Question 2.5: Finding Analogies [code + written] (1.5 points)

a. For the previous example, it's clear that "grandmother" completes the analogy. But give an intuitive explanation as to why the `most_similar` function gives us words like "granddaughter", "daughter", or "mother?

Write your answer here.

b. 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!

- granddaughter, daughter, mother이 grandmother과 같이 쓰이는 경우가 많았기 때문이다.

b. 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!

```
# For example: x, y, a, b = ("", "", "", "")
# ----------------
# Write your implementation here.

x, y, a, b = 'tall', 'short', 'high', 'low'
# Test the solution
assert wv_from_bin.most_similar(positive=[a, y], negative=[x])[0][0] == b
print(f"{x} is to {y} as {a} is to {b}")
# ----------------
```

```
tall is to short as high is to low
```

- tall, short, high, low도 같은 관계로 볼 수 있다.

3.6 Question 2.6: Incorrect Analogy(1.5 points)

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

a. Below, we expect to see the intended analogy "hand : glove :: foot : **sock**", but we see an unexpected result instead. Give a potential reason as to why this particular analogy turned out the way it did?

```
pprint.pprint(wv_from_bin.most_similar(positive=['foot', 'glove'], negative=['hand']))
```

```
[('45,000-square', 0.4922032058238983),
 ('15,000-square', 0.4649604558944702),
 ('10,000-square', 0.45447564125061035),
 ('6,000-square', 0.44975772500038147),
 ('3,500-square', 0.4441334009170532),
 ('700-square', 0.44257497787475586),
 ('50,000-square', 0.43563973903656006),
 ('3,000-square', 0.43486514687538147),
 ('30,000-square', 0.4330596923828125),
 ('footed', 0.4323687553405761 7)]
```

- 그냥.

b. Find another 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 the previous example, this would be '**45,000-square**').

```
# For example: x, y, a, b = ("", "", "", "")
# ----------------
# Write your implementation here.

x, y, a, b = 'cow', 'calf', 'cat', 'kitten'
# ----------------
pprint.pprint(wv_from_bin.most_similar(positive=[a, y], negative=[x]))
assert wv_from_bin.most_similar(positive=[a, y], negative=[x])[0][0] != b
```

```
[('ankle', 0.5550614595413208),
 ('knee', 0.5175143480300903),
 ('groin', 0.5135608315467834),
 ('thigh', 0.5085046887397766),
 ('hamstring', 0.5065605640411377),
 ('tendon', 0.4663308262825012),
 ('achilles', 0.46237698197364807),
 ('sprained', 0.46101343631744385),
 ('legs', 0.4600679278373718),
```

- cow: calf :: cat : kitten → ankle

3.7 Question 2.7: Guided Analysis of Bias in Word Vectors(1 point)

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 "man" and "profession" and most dissimilar to "woman" and (b) which terms are most similar to "woman" and "profession" and most dissimilar to "man". 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.

```
# Run this cell
# Here `positive` indicates the list of words to be similar to and `negative` indicates the list of words to be
# most dissimilar from.

pprint.pprint(wv_from_bin.most_similar(positive=['man', 'profession'], negative=['woman']))
print()
pprint.pprint(wv_from_bin.most_similar(positive=['woman', 'profession'], negative=['man']))
```

```
[('reputation', 0.5250177383422852),
 ('professions', 0.5178037881851196),
 ('skill', 0.49046966433525085),
 ('skills', 0.4900550842285156),
 ('ethic', 0.4897659420967102),
 ('business', 0.4875851273536682),
 ('respected', 0.485920250415802),
 ('practice', 0.482104629278183),
 ('regarded', 0.4778572618961334),
 ('life', 0.4760662019252777)]
```

- man: skill, reputation, business
- woman: teaching, nursing, vocation
- 성과 관련한 직업이 나오는 것을 확인할 수 있다.

단어장

synonym: 동의어

encapsulating: 캡슐화

akin to: ~와 유사하다

pull out: 추출하다, 빼다

polysemy: 다의어

homonym: 동음이의어