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

Due 4:30pm, Tue Jan 14

Welcome to CS224n!

Before you start, make sure you read the README.txt in the same directory as this notebook. You will find many provided codes in the notebook. We highly encourage you to read and understand the provided codes as part of the learning :)

```
# If you don't have environment variables set
# Check out https://docs.wandb.com/quickstart#2-log-from-your-script
# and https://docs.wandb.com/quickstart#3-view-your-visualized,-organized,-and-collaborative-machine-learning-work
# for more information about setup for wandb.
# Uncomment the line below to log in to wandb.
# !wandb login

# Output:
# wandb: Current SDK version is 0.10.11
# wandb: Current API version is 1.3.0
# wandb: If you have trouble connecting to W&B servers, set a timeout with export WANDB_HTTP_TIMEOUT=<seconds>
# wandb: Enter your choice: (Press enter for default [auto])
# wandb: You chose "never"
# wandb: Logging disabled.
# wandb: Run data will not be synced permanently.
# wandb: To enable syncing, run "wandb on".
# wandb: To turn off this message, run: "wandb off".
```

```
# If you don't have environment variables set
# Check out https://docs.wandb.com/quickstart#2-log-from-your-script
# and https://docs.wandb.com/quickstart#3-view-your-visualized,-organized,-and-collaborative-machine-learning-work
# for more information about setup for wandb.
# Uncomment the line below to log in to wandb.
# !wandb login

# Output:
# wandb: Current SDK version is 0.10.11
# wandb: Current API version is 1.3.0
# wandb: If you have trouble connecting to W&B servers, set a timeout with export WANDB_HTTP_TIMEOUT=<seconds>
# wandb: Enter your choice: (Press enter for default [auto])
# wandb: You chose "never"
# wandb: Logging disabled.
# wandb: Run data will not be synced permanently.
# wandb: To enable syncing, run "wandb on".
# wandb: To turn off this message, run: "wandb off".
```

## 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 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* statistics, and those derived via *word2vec*.

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

**Note on Terminology:** Throughout the assignment, we will refer to the vector representation of a word as a word vector or simply a vector. Also, in the context of word vectors, we will sometimes use the terms embedding and embedding matrix interchangeably with word vectors. Finally, occasionally 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 date from the mid 2000s. Intuitively, they rely on the hypothesis that words that occur in similar contexts tend to have similar meanings.

For one such family of models, we can distinguish two ways to define a word's context: one where it includes all other words in the document, and another where it includes only a window of a few words (typically five) on either side of the word. For example, given the sentences:

```
I enjoy flying.
I like NLP.
I like deep learning.
```

The resulting counts matrix (with rows and columns alphabetically ordered) is:

| | I | like | enjoy | flying | . | NLP | deep | learning |
|---|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| . | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| learning | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

This row-by-column co-occurrence matrix is symmetric (since the relation "w1 is in the context of w2" implies the converse), and thus is also a common form of word-word co-occurrence matrix we have considered in class.

Your task will be to implement this co-occurrence matrix, and then to investigate properties of the word vectors it yields. Keep in mind that this is a small-scale corpus for illustrative purposes, so we don't expect to see the results we've been discussing in class. But we do expect you to see some evidence of the properties we've been discussing in class, and to be able to explain why the word vectors we are creating have the properties they do.

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

**Note on Terminology:** Throughout the assignment, we will refer to the vector representation of a word as a word vector or simply a vector. Also, in the context of word vectors, we will sometimes use the terms embedding and embedding matrix interchangeably with word vectors. Finally, occasionally 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 date from the mid 2000s. Intuitively, they rely on the hypothesis that words that occur in similar contexts tend to have similar meanings.

For one such family of models, we can distinguish two ways to define a word's context: one where it includes all other words in the document, and another where it includes only a window of a few words (typically five) on either side of the word. For example, given the sentences:

```
I enjoy flying.
I like NLP.
I like deep learning.
```

The resulting counts matrix (with rows and columns alphabetically ordered) is:

| | I | like | enjoy | flying | . | NLP | deep | learning |
|---|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| . | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| learning | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

This row-by-column co-occurrence matrix is symmetric (since the relation "w1 is in the context of w2" implies the converse), and thus is also a common form of word-word co-occurrence matrix we have considered in class.

Your task will be to implement this co-occurrence matrix, and then to investigate properties of the word vectors it yields. Keep in mind that this is a small-scale corpus for illustrative purposes, so we don't expect to see the results we've been discussing in class. But we do expect you to see some evidence of the properties we've been discussing in class, and to be able to explain why the word vectors we are creating have the properties they do.

```
# All HW2 imports
import numpy as np
import sklearn.decomposition as skd
import matplotlib
import matplotlib.pyplot as plt
import gensim.downloads.api as gd
import gensim.models.keyvectors as gmk
import gensim.scripts.glove2word2vec as gg2w
import os
import urllib.error
import sklearn.manifold as skm

# Matplotlib configuration
%matplotlib inline
%config InlineBackend.figure_format = 'retina'

# Check version
import sys
import platform
print("Python version: {}".format(sys.version))
print("Platform: {}".format(platform.platform()))
print("Numpy version: {}".format(np.__version__))
import matplotlib
import sklearn
import gensim
import urllib
import os
import os.path as op

# Output:
# Python version: 3.7.9 (default, Aug 27 2020, 21:59:41) [MSC v.1916 64 bit (AMD64)]
# Platform: Windows-10-10.0.19041-SP0
# Numpy version: 1.19.2
```

## 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* statistics, and those derived via *word2vec*.

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

**Note on Terminology:** Throughout the assignment, we will refer to the vector representation of a word as a word vector or simply a vector. Also, in the context of word vectors, we will sometimes use the terms embedding and embedding matrix interchangeably with word vectors. Finally, occasionally 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 date from the mid 2000s. Intuitively, they rely on the hypothesis that words that occur in similar contexts tend to have similar meanings.

For one such family of models, we can distinguish two ways to define a word's context: one where it includes all other words in the document, and another where it includes only a window of a few words (typically five) on either side of the word. For example, given the sentences:

```
I enjoy flying.
I like NLP.
I like deep learning.
```

The resulting counts matrix (with rows and columns alphabetically ordered) is:

| | I | like | enjoy | flying | . | NLP | deep | learning |
|---|---|---|---|---|---|---|---|---|
| I | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 0 |
| like | 2 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| enjoy | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| flying | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| . | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| NLP | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| deep | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| learning | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |

This row-by-column co-occurrence matrix is symmetric (since the relation "w1 is in the context of w2" implies the converse), and thus is also a common form of word-word co-occurrence matrix we have considered in class.

Your task will be to implement this co-occurrence matrix, and then to investigate properties of the word vectors it yields. Keep in mind that this is a small-scale corpus for illustrative purposes, so we don't expect to see the results we've been discussing in class. But we do expect you to see some evidence of the properties we've been discussing in class, and to be able to explain why the word vectors we are creating have the properties they do.

```
# Ignore this.
# Set up some ignore patterns.
import os
import os.path as op
import urllib.error
import urllib.request
import sklearn.manifold as skm

# Output:
# Python version: 3.7.9 (default, Aug 27 2020, 21:59:41) [MSC v.1916 64 bit (AMD64)]
# Platform: Windows-10-10.0.19041-SP0
# Numpy version: 1.19.2
```

### Part 1.1: Implement `pairs_to_mat(pairs)` [code] (2 points)

Write a method to serve as the Gensim corpus input format for your co-occurrence matrix. Your code should be a generator function that yields a document at a time. Each document should be a list of strings, where each string is a word in the document. Your code should ignore the document ID, which will be the first element of the input tuple.

```
# Example: Convert pairs into a single document.
# Ignore the document ID.
# Output: ['I', 'like', 'enjoy', 'flying', '.']
# Output: ['I', 'like', 'NLP', '.']
# Output: ['I', 'like', 'deep', 'learning', '.']
```

```
# Ignore this.
# Set up some ignore patterns.
import os
import os.path as op
import urllib.error
import urllib.request
import sklearn.manifold as skm

# Output:
# Python version: 3.7.9 (default, Aug 27 2020, 21:59:41) [MSC v.1916 64 bit (AMD64)]
# Platform: Windows-10-10.0.19041-SP0
# Numpy version: 1.19.2
```

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

As discussed in class, more recently word vectors have been derived from neural network models, or more generally from machine learning models. For this part of the assignment, we will be using the popular library Gensim to load and use pre-trained word vectors from a state-of-the-art neural network word2vec model trained on Google News articles. Gensim also provides many other word vector models, including ones that we will use later in the course, such as GloVe and fastText. For a list of the pre-trained word vectors that are available in Gensim, see https://github.com/RaRe-Technologies/gensim-data. Loading a pre-trained word vector model in Gensim is very easy, as the following code shows:

```
# Download the word2vec-google-news-300 model.
# This will take about 2 minutes the first time you run it.
# Subsequent runs should be much faster.
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Download the word2vec-google-news-300 model.
# This will take about 2 minutes the first time you run it.
# Subsequent runs should be much faster.
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

### Part 2.1: Hyponymy and Antonyms [code + written] (5 points)

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

As an example, for the analogy "man : woman :: king : queen", in some cases the following operation applies:

$$king - man + woman = queen$$

In this exercise, we show you how to see whether this property holds in the word2vec vectors you just loaded.

Note: Further Documentation on the `wv_from_pretrained` function can be found within the Gensim documentation.

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

### Part 2.2: Synonyms & Antonyms [code + written] (5 points)

When considering Cosine Similarity, it often becomes convenient to think of Cosine Distance, which is simply 1 - Cosine Similarity.

Find three words (w1,w2,w3) where w1 and w2 are synonyms and w1 and w3 are antonyms, but Cosine Distance(w1, w3) < Cosine Distance(w1, w2). For the purposes of this question, you should use the word2vec vectors you loaded above.

Hint: You can find the cosine distance between two words given their word vectors by using the `scipy.spatial.distance.cosine` function.

```
# Output: 0.3790276348590851
# Output: 0.3790276348590851
```

```
# Output: 0.3790276348590851
# Output: 0.3790276348590851
```

### Part 2.3: Analogies [code + written] (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 x1 : x2 :: y1 : y2. For example, if you think "man : woman :: king : queen" is the analogy you're looking for, you would write:

Note: You may have to try many analogies to find one that works!

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

### Part 2.4: Incorrect Analogy [code + written] (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 x1 : x2 :: y1 : y2. For example, if you think "man : woman :: king : queen" is the analogy you're looking for, you would write:

Note: You may have to try many analogies to find one that works!

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

### Part 2.5: Bias [code + written] (5 points)

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

As an example, for the analogy "man : woman :: king : queen", in some cases the following operation applies:

$$king - man + woman = queen$$

In this exercise, we show you how to see whether this property holds in the word2vec vectors you just loaded.

Note: Further Documentation on the `wv_from_pretrained` function can be found within the Gensim documentation.

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

### Part 2.6: Bias [code + written] (5 points)

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

As an example, for the analogy "man : woman :: king : queen", in some cases the following operation applies:

$$king - man + woman = queen$$

In this exercise, we show you how to see whether this property holds in the word2vec vectors you just loaded.

Note: Further Documentation on the `wv_from_pretrained` function can be found within the Gensim documentation.

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

```
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
# Output: [('queen', 0.7118192315292), ('monarch', 0.6189675331115), ('princess', 0.5777117609
```

## Submission Instructions

1. Click the Save button at the top of the screen.
2. Select File -> Download -> Clean. This will clear all the outputs from all cells (but will keep the contents of all cells).
3. Select File -> Download -> Notebook (.ipynb) to download your ipynb file.
4. Rename the downloaded file to a_1.ipynb if it is not already named that. Then, upload the file to the Assignment 1 Submission Gradescope assignment. Make sure that the notebook you submit is named a_1.ipynb and not something else (such as a_1 (1).ipynb).
5. Unzip your code folder and rename it to a_1_code. Then, zip your code folder (not just the files in it) as a_1_code.zip and upload it to the Assignment 1 Code Submission Gradescope assignment.

Submit your PDF on Gradescope.