

Lecture 5: Word Embeddings

Contents

- Imports and LOs
- Sneak peek
- 1. Motivation and context
- 2. Word representations
- 3. Dense representations
- ? ? Questions for you
- 4. More word2vec
- ? ? Questions for you
- Final comments, summary, and reflection
- Resources
- Fun tools



DSCI 563 Unsupervised Learning

UBC Master of Data Science program, 2024-25

Imports and LOs

Imports

```
import os
import random
import sys
import time

import numpy as np
import pandas as pd

sys.path.append(os.path.join(os.path.abspath("."), "code"))

import matplotlib.pyplot as plt
from comat import CooccurrenceMatrix
from preprocessing import MyPreprocessor
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.feature_extraction.text import CountVectorizer
# from support_functions import *

plt.rcParams["font.size"] = 16
import matplotlib.cm as cm

%matplotlib inline
pd.set_option("display.max_colwidth", 0)
```

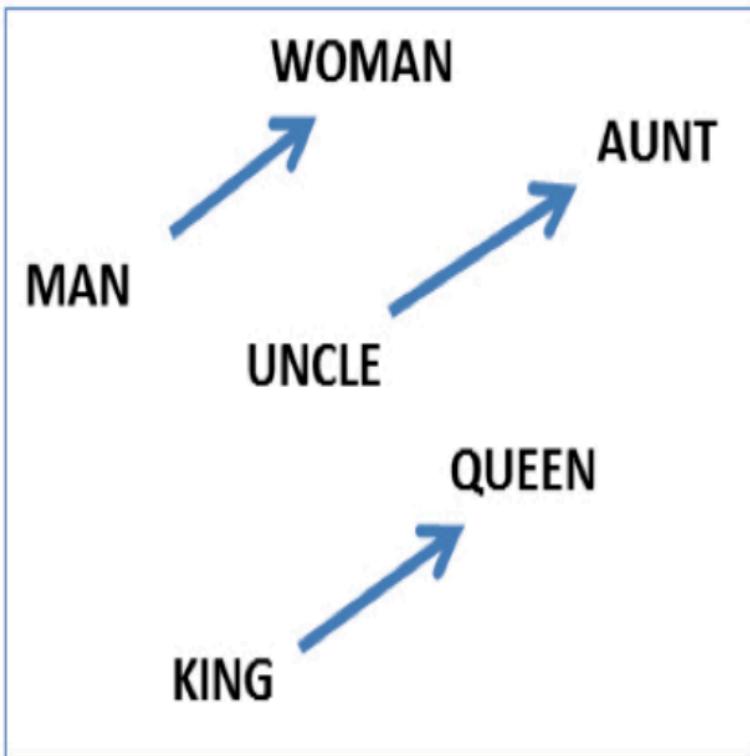
Learning outcomes

From this lecture, students are expected to be able to:

- Explain the general idea of a vector space model.
- Explain the difference between different word representations: one-hot encoding, term-term co-occurrence matrix representation, and word2vec representation.
- Explain the skip-gram model at a high level.
- Load and use pre-trained word embeddings to find word similarities and analogies.
- Train your own word vectors with [Gensim](#).
- Use word2vec models for word similarity and analogies.
- Demonstrate biases in word2vec models and learn to watch out for such biases in pre-trained embeddings.

Sneak peek

- In 2013, Tomas Mikolov et al. published the paper [Efficient Estimation of Word Representations in Vector Space](#) by which introduced word2vec and has since received over 46,000 citations.
- word2vec generates word embeddings that capture relationships between words. Since its publication, this idea has been widely adopted and applied in various text-based data science applications.
- A key example from the paper: **MAN : KING :: WOMAN : ?** (MAN is to KING as WOMAN is to what?)
 - What word relates to **WOMAN** in the same way **KING** relates to **MAN**?
- Perform a simple algebraic operations with the vector representation of words.
 $\vec{X} = \vec{\text{KING}} - \vec{\text{MAN}} + \vec{\text{WOMAN}}$
- Search in the vector space for the word closest to \vec{X} measured by cosine distance.



(Credit: Mikolov et al. 2013)

In this lecture, I'll provide context and intuition for word embeddings.

1. Motivation and context

Activity: Discuss the following questions with your neighbour 🤔

- Why are low-dimensional numeric representations created by PCA, LSA, and NMF useful?
- If you wanted to reduce the dimensionality of text data, which method (PCA, LSA, NMF) would you choose, and why?
- Do large language models, such as ChatGPT, "understand" your questions to some extent and provide useful responses? If so, how do you think they achieve this?
- What is required for a machine to "understand" language?

1.1 Why word embeddings?

- Last week, we discussed representing text with LSA or TruncatedSVD.
- So far, we have focused on sentence and document-level representations.
- But what about words? Can we represent words in a way that captures their meaning?
- How do humans recognize relationships between words (e.g., **awesome** and **delightful**)? Can we teach machines to do the same?
- Words are the basic semantic units of text. To capture the meaning of text, we must first capture the meaning of words—including their relationships!💡

Activity: Context and word meaning

- Pair up with the person next to you and try to guess the meanings of two made-up words: **flibbertigibbet** and **groak**.

1. The plot twist was totally unexpected, making it a **flibbertigibbet** experience.
2. Despite its **groak** special effects, the storyline captivated my attention till the end.
3. I found the character development rather **groak**, failing to evoke empathy.
4. The cinematography is **flibbertigibbet**, showcasing breathtaking landscapes.
5. A **groak** narrative that could have been saved with better direction.
6. This movie offers a **flibbertigibbet** blend of humour and action, a must-watch.
7. Sadly, the movie's potential was overshadowed by its **groak** pacing.
8. The soundtrack complemented the film's theme perfectly, adding to its **flibbertigibbet** charm.
9. It's rare to see such a **flibbertigibbet** performance by the lead actor.
10. Despite high expectations, the film turned out to be quite **groak**.
11. **Flibbertigibbet** dialogues and a gripping plot make this movie stand out.
12. The film's **groak** screenplay left much to be desired.

Attributions: Thanks to ChatGPT!

- How did you infer the meaning of the words **flibbertigibbet** and **groak**?
- Which specific words or phrases in the context helped you infer the meaning of these imaginary words?

What you did in the above activity is referred to as **distributional hypothesis**.

You shall know a word by the company it keeps.

Firth, 1957

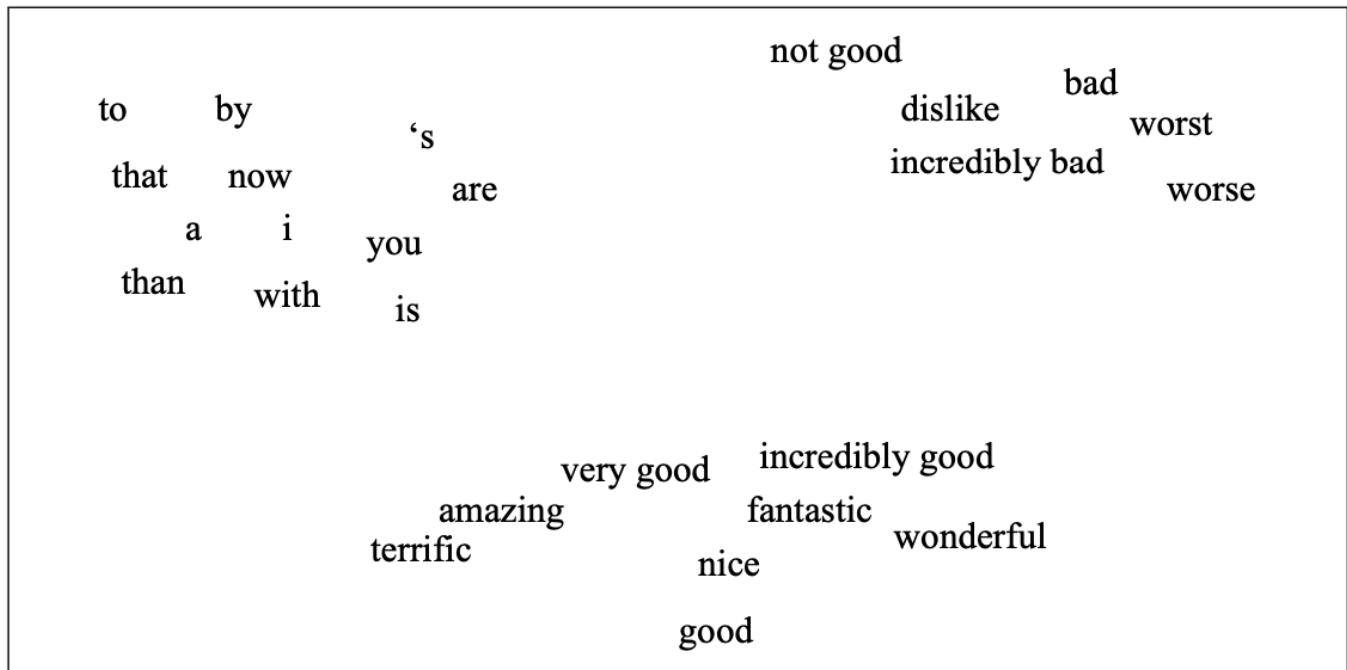
If A and B have almost identical environments we say that they are **synonyms**.
Harris, 1954

Example:

- The plot twist was totally unexpected, making it a **flibbertigibbet** experience.
- The plot twist was totally unexpected, making it a **delightful** experience.

1.2 Word representations: intro

- A standard way to represent meanings of words is by placing them into a vector space.
- Distances between words in the vector space indicate relationships between them.



(Attribution: [Jurafsky and Martin 3rd edition](#))

- Word meaning has been a favourite topic of philosophers for centuries.
- An example from legal domain: [Are hockey gloves "gloves, mittens, mitts" or "articles of plastics"?](#)

Canada (A.G.) v. Igloo Vikski Inc. was a tariff code case that made its way to the SCC (Supreme Court of Canada). The case disputed the definition of hockey gloves as either "gloves, mittens, or mitts" or as "other articles of plastic."



In ML and Natural Language Processing (NLP) we are interested in

- Modeling word meaning that allows us to
 - draw useful inferences to solve meaning-related problems
 - find relationship between words, e.g., which words are similar, which ones have positive or negative connotations

Example: Word similarity

- Suppose you are carrying out sentiment analysis.
- Consider the sentences below.

S1: This movie offers a **flibbertigibbet** blend of humour and action, a must-watch.

S2: This movie offers a **delightful** blend of humour and action, a must-watch.

- Here we would like to capture similarity between **flibbertigibbet** and **delightful** in reference to sentiment analysis task.

How are word embeddings related to unsupervised learning?

- They are closely related to dimensionality reduction and extracting meaningful representations from raw data.
- The word2vec algorithm is an unsupervised (or semi-supervised) method; we do not need any labeled data but we use running text as supervision signal.

2. Word representations

Activity: Brainstorm ways to represent words (~2 mins)

- Suppose you are building a question answering system and you are given the following question and three candidate answers.
- What kind of relationship between words would we like our representation to capture in order to arrive at the correct answer?

Question: How **tall** is Machu Picchu?

Candidate 1: Machu Picchu is 13.164 degrees south of the equator.

Candidate 2: The official height of Machu Picchu is 2,430 m.

Candidate 3: Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.

- Let's explore different ways to represent words.

- First, let's look at two simplistic word representations
 - One-hot representation
 - Term-term co-occurrence matrix

2.1 Simplest representation: One-hot representation of words

- Example: Consider the sentence

How tall is Machu_Picchu ?

- What is the one-hot representation for the word *tall*?
 - Vocabulary size = 5 and index of the word *tall* = 1
 - One-hot vector for *tall*: [0 1 0 0 0]
- Build **vocabulary** containing all unique words in the corpus.
- One-hot representation of a word is a vector of length V such that the value at word index is 1 and all other indices is 0.

```
def get_onehot_encoding(word, vocab):  
    onehot = np.zeros(len(vocab), dtype="float64")  
    onehot[vocab[word]] = 1  
    return onehot
```

```

from sklearn.metrics.pairwise import cosine_similarity

def print_cosine_similarity(df, word1, word2):
    """
    Returns similarity score between word1 and word2
    Arguments
    -----
    df -- (pandas.DataFrame)
        Dataframe containing word representations
    word1 -- (array)
        Representation of word1
    word2 -- (array)
        Representation of word2

    Returns
    -----
    None. Returns similarity score between word1 and word2 with the given repr
    """
    vec1 = df.loc[word1].values.reshape(1, -1)
    vec2 = df.loc[word2].values.reshape(1, -1)
    sim = cosine_similarity(vec1, vec2)
    print(
        "The dot product between %s and %s: %0.2f and cosine similarity is: %0
        % (word1, word2, np.dot(vec1.flatten(), vec2.flatten()), sim[0][0])
    )

```

Vocabulary and one-hot encoding

```

corpus = (
    """how tall is machu_picchu ? the official height of machu_picchu is 2,430
)
unique_words = list(set(corpus.split()))
unique_words.sort()
vocab = {word: index for index, word in enumerate(unique_words)}
print("Size of the vocabulary: %d" % (len(vocab)))
print(vocab)

```

```

Size of the vocabulary: 12
{'.': 0, '2,430': 1, '?': 2, 'height': 3, 'how': 4, 'is': 5, 'm': 6, 'machu_pi

```

```

data = {}
for word in vocab:
    data[word] = get_onehot_encoding(word, vocab)
ohe_df = pd.DataFrame(data).T
ohe_df

```

	0	1	2	3	4	5	6	7	8	9	10	11
.	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2,430	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
?	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
height	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
how	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
is	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0
m	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0
machu_picchu	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
of	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
official	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
tall	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
the	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0

```
print_cosine_similarity(ohe_df, "tall", "height")
print_cosine_similarity(ohe_df, "tall", "official")
```

The dot product between tall and height: 0.00 and cosine similarity is: 0.00
The dot product between tall and official: 0.00 and cosine similarity is: 0.00

Problem with one-hot encoding

- We would like the word representation to capture the similarity between *tall* and *height* and so we would like them to have bigger dot product or bigger cosine similarity (normalized dot product).
- The problem with one-hot representation of words is that there is no inherent notion of relationship between words and the dot product between similar and non-similar words is zero.

$$\vec{tall} \cdot \vec{height} = 0$$

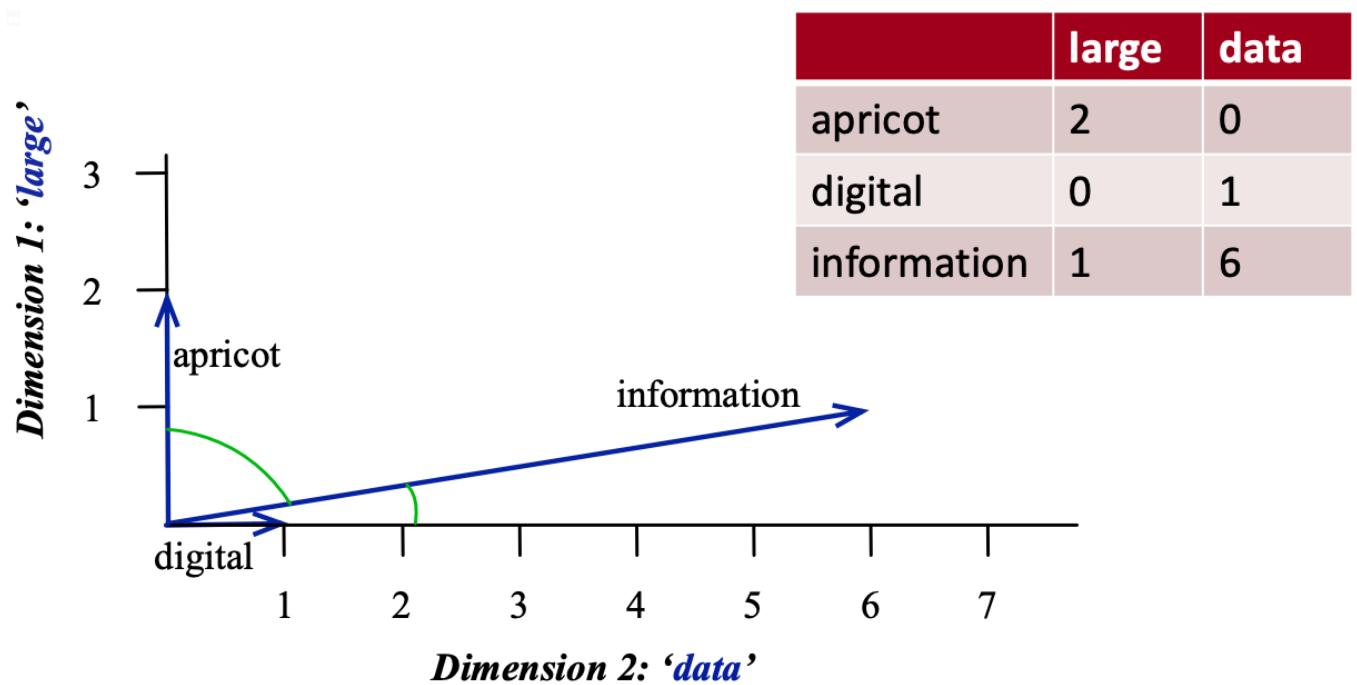
Need a representation that captures relationships between words.

- We will be looking at two such representations.
 1. Sparse representation with **term-term co-occurrence matrix**
 2. Dense representation with **word2vec skip-gram model**

2.2 Term-term co-occurrence matrix

- So far we have been talking about documents and we created document-term co-occurrence matrix (e.g., bag-of-words representation of text).
- We can also do this with words. The idea is to go through a corpus of text, keeping a count of all of the words that appear in context of each word (within a window).

Visualizing word vectors and similarity



(Credit: Jurafsky and Martin 3rd edition)

- The similarity is calculated using dot products between word vectors.
 - Example: $\vec{\text{digital}} \cdot \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
 - Higher the dot product more similar the words.
- The similarity is calculated using dot products between word vectors.
 - Example: $\vec{\text{digital}} \cdot \vec{\text{information}} = 0 \times 1 + 1 \times 6 = 6$
 - Higher the dot product more similar the words.

- We can also calculate a normalized version of dot products. \$

$$\text{similarity}_{\text{cosine}}(w_1, w_2) = \frac{w_1 \cdot w_2}{\|w_1\|_2 \|w_2\|_2} \$$$

```
corpus = [
    "How tall is Machu Picchu?",
    "Machu Picchu is 13.164 degrees south of the equator.",
    "The official height of Machu Picchu is 2,430 m.",
    "Machu Picchu is 80 kilometres (50 miles) northwest of Cusco.",
    "It is 80 kilometres (50 miles) northwest of Cusco, on the crest of the mo
]
sents = MyPreprocessor(corpus)
```

```
cm = CooccurrenceMatrix(
    sents, window_size=2
) # Let's build term-term co-occurrence matrix for our text.
comat = cm.fit_transform()
vocab = cm.get_feature_names()
df = pd.DataFrame(comat.todense(), columns=vocab, index=vocab, dtype=np.int8)
df.head()
```

	tall	machu	picchu	13.164	degrees	south	equator	official	height
tall	0	1	1	0	0	0	0	0	0
machu	1	0	5	1	0	0	0	1	1
picchu	1	5	0	1	1	0	0	0	1
13.164	0	1	1	0	1	1	0	0	0
degrees	0	0	1	1	0	1	1	0	0

5 rows × 32 columns

```
print_cosine_similarity(df, "tall", "height")
print_cosine_similarity(df, "tall", "official")
```

The dot product between tall and height: 2.00 and cosine similarity is: 0.82
 The dot product between tall and official: 1.00 and cosine similarity is: 0.50

- We are getting non-zero cosine similarity now and we are able to capture some similarities between words now.

- That said similarities do not make much sense in the toy example above because we're using a tiny corpus.
- To find meaningful patterns of similarities between words, we need a large corpus.
- Let's try a bit larger corpus and check whether the similarities make sense.

```
import wikipedia
from nltk.tokenize import sent_tokenize, word_tokenize

corpus = []

queries = [
    "Machu Picchu",
    # "Everest",
    "Sequoia sempervirens",
    "President (country)",
    "Politics Canada",
]

for i in range(len(queries)):
    sents = sent_tokenize(wikipedia.page(queries[i]).content)
    corpus.extend(sents)
print("Number of sentences in the corpus: ", len(corpus))
```

Number of sentences in the corpus: 785

```
sents = MyPreprocessor(corpus)
cm = CooccurrenceMatrix(sents)
comat = cm.fit_transform()
vocab = cm.get_feature_names()
df = pd.DataFrame(comat.todense(), columns=vocab, index=vocab, dtype=np.int8)
df
```

	machu	picchu	15th-century	inca	citadel	located	eastern	cordilleran
machu	0	85	1	3	1	2	1	1
picchu	85	0	1	6	3	1	1	1
15th-century	1	1	0	1	1	1	0	1
inca	3	6	1	0	3	1	1	1
citadel	1	3	1	3	0	1	1	1
...
boondoggles	0	0	0	0	0	0	0	1
elephants	0	0	0	0	0	0	0	1
campaigning	0	0	0	0	0	0	0	1
compared	0	0	0	0	0	0	0	1
textbook	0	0	0	0	0	0	0	1

4242 rows × 4242 columns

```
print_cosine_similarity(df, "tall", "height")
print_cosine_similarity(df, "tall", "official")
```

The dot product between tall and height: 19.00 and cosine similarity is: 0.31
The dot product between tall and official: 0.00 and cosine similarity is: 0.00

3. Dense representations

- The goal is to learn general purpose embeddings that are useful for common tasks involving text data.

Sparse vs. dense word vectors

- Term-term co-occurrence matrix representation is long and sparse.
 - length $|V|$ is usually large (e.g., > 50,000)
 - most elements are zero
- OK because there are efficient ways to deal with sparse matrices.
- Learn short (~100 to 1000 dimensions) and dense vectors.
- Short vectors are usually easier to train with ML models (less weights to train).
- They may generalize better.
- In practice they work much better!

3.1 What can we do with these word representations

- Before looking at how to create dense word representations let's see how they look like and what can we do with them.
- Below I am loading word vectors trained on Google News corpus.

```
# It'll take a while to run this when you try it out for the first time.
import gensim.downloader as api

google_news_vectors = api.load("word2vec-google-news-300")
```

```
ModuleNotFoundError                                     Traceback (most recent call last)
Cell In[13], line 2
      1 # It'll take a while to run this when you try it out for the first time
----> 2 import gensim.downloader as api
      3 google_news_vectors = api.load("word2vec-google-news-300")

ModuleNotFoundError: No module named 'gensim'
```

```
print("Size of vocabulary: ", len(google_news_vectors))
```

Size of vocabulary: 3000000

- `google_news_vectors` above has 300 dimensional word vectors for 3,000,000 unique words from Google news.

- Let's examine word vector for the word UBC.

```
google_news_vectors["UBC"][:20] # Representation of the word UBC
```

```
array([-0.3828125 , -0.18066406,  0.10644531,  0.4296875 ,  0.21582031,
       -0.10693359,  0.13476562, -0.08740234, -0.14648438, -0.09619141,
       0.02807617,  0.01409912, -0.12890625, -0.21972656, -0.41210938,
      -0.1875     , -0.11914062, -0.22851562,  0.19433594, -0.08642578],
      dtype=float32)
```

```
google_news_vectors["UBC"].shape
```

```
(300,)
```

Indeed it is a short and a dense vector!

Finding similar words

- Given word w , search in the vector space for the word closest to w as measured by cosine distance.

```
google_news_vectors.most_similar("UBC")
```

```
[('UVic', 0.7886475920677185),
 ('SFU', 0.7588528394699097),
 ('Simon_Fraser', 0.7356574535369873),
 ('UFV', 0.688043475151062),
 ('VIU', 0.6778583526611328),
 ('Kwantlen', 0.6771429181098938),
 ('UBCO', 0.6734487414360046),
 ('UPEI', 0.673112690448761),
 ('UBC_Okanagan', 0.6709133386611938),
 ('Lakehead_University', 0.6622507572174072)]
```

```
google_news_vectors.most_similar("information")
```

```
[('info', 0.7363681793212891),  
 ('infomation', 0.680029571056366),  
 ('infor_mation', 0.673384964466095),  
 ('informaiton', 0.6639008522033691),  
 ('informa_tion', 0.660125732421875),  
 ('informationon', 0.633933424949646),  
 ('informationabout', 0.6320978999137878),  
 ('Information', 0.6186580657958984),  
 ('informaion', 0.6093292832374573),  
 ('details', 0.6063088774681091)]
```

If you want to extract all documents containing words similar to **information**, you could use this information.

Google News embeddings also support multi-word phrases.

```
google_news_vectors.most_similar("British_Columbia")
```

```
[('BC', 0.7640386819839478),  
 ('Alberta', 0.7285023331642151),  
 ('Ontario', 0.7031311392784119),  
 ('Vancouver', 0.6976038217544556),  
 ('Lower_Mainland', 0.6730170249938965),  
 ('Saskatchewan', 0.6690970063209534),  
 ('Manitoba', 0.6569438576698303),  
 ('Canada', 0.6478375792503357),  
 ('Kamloops', 0.6449972987174988),  
 ('Nanaimo_BC', 0.6426822543144226)]
```

Finding similarity scores between words

```
google_news_vectors.similarity("Canada", "hockey")
```

0.27610135

```
google_news_vectors.similarity("Japan", "hockey")
```

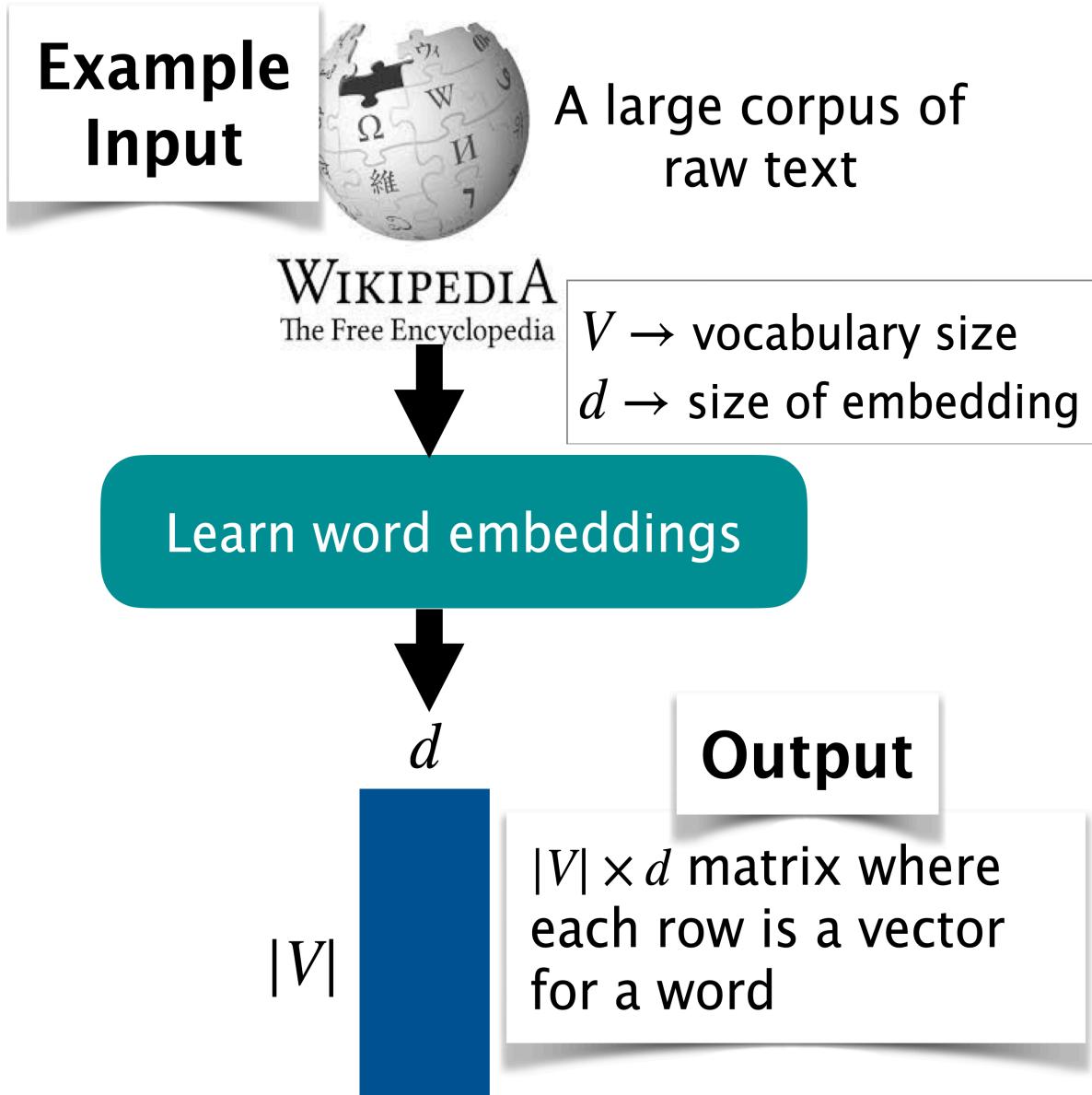
0.0019627889

```
word_pairs = [
    ("height", "tall"),
    ("height", "official"),
    ("pineapple", "mango"),
    ("pineapple", "juice"),
    ("sun", "robot"),
    ("GPU", "hummus"),
]
for pair in word_pairs:
    print(
        "The similarity between %s and %s is %0.3f"
        % (pair[0], pair[1], google_news_vectors.similarity(pair[0], pair[1]))
    )
```

The similarity between height and tall is 0.473
The similarity between height and official is 0.002
The similarity between pineapple and mango is 0.668
The similarity between pineapple and juice is 0.418
The similarity between sun and robot is 0.029
The similarity between GPU and hummus is 0.094

We are getting reasonable word similarity scores!!

3.2 Creating dense representations



Two classes of approaches

- **LSA (count-based/matrix factorization.)**
 - LSA can generate dense word embeddings.
 - Works well for small datasets.
- **word2vec (prediction-based)**
 - A popular alternative for creating dense embeddings using neural networks.
 - Part of a family of algorithms designed for word representation.

Goal for this lecture: Build intuition for these approaches.

word2vec task

Remember fill in the blank puzzles in your highschool tests?

Add freshly squeezed ___ juice to your smoothie.

1. pineapple
2. scarf
3. PCA
4. earthquake

Another slightly non-intuitive way to think about this is what would be the context words given the target word **pineapple**?

Add freshly ___ **pineapple** ___ to your smoothie.

word2vec learns meaningful word representations by learning to solve large number of such fill in the blank puzzles.

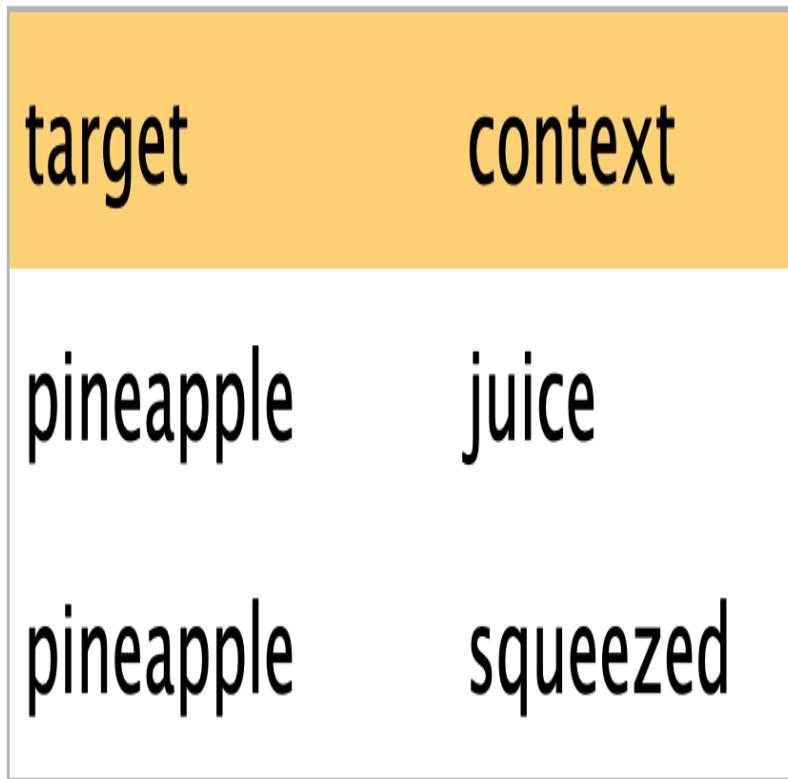
- Based on this intuition, there are two primary algorithms
 - Continuous bag of words (CBOW)
 - **Skip-gram**
- Two moderately efficient training methods
 - Hierarchical softmax
 - Negative sampling

word2vec: Skip-gram model

- We are going to talk about the inefficient Skip-gram model, as it's enough to get an intuition.
- A neural network model to obtain short and dense representations of words.
- A simple architecture with
 - an input layer
 - a linear hidden layer (without any activation function)
 - an output layer with softmax

- In skip-gram we work on an “auxiliary” supervised machine learning word prediction task of prediction context words given the target word.
- We train a neural network for this task and the learned weights are our word vectors.
- So we are not actually interested in making prediction about the context words.
- Our goal is to learn meaningful weights (meaningful representation of the input) in the process.

Add freshly squeezed\$_{context}\$ pineapple\$_{target}\$ juice\$_{context}\$ to your smoothie.

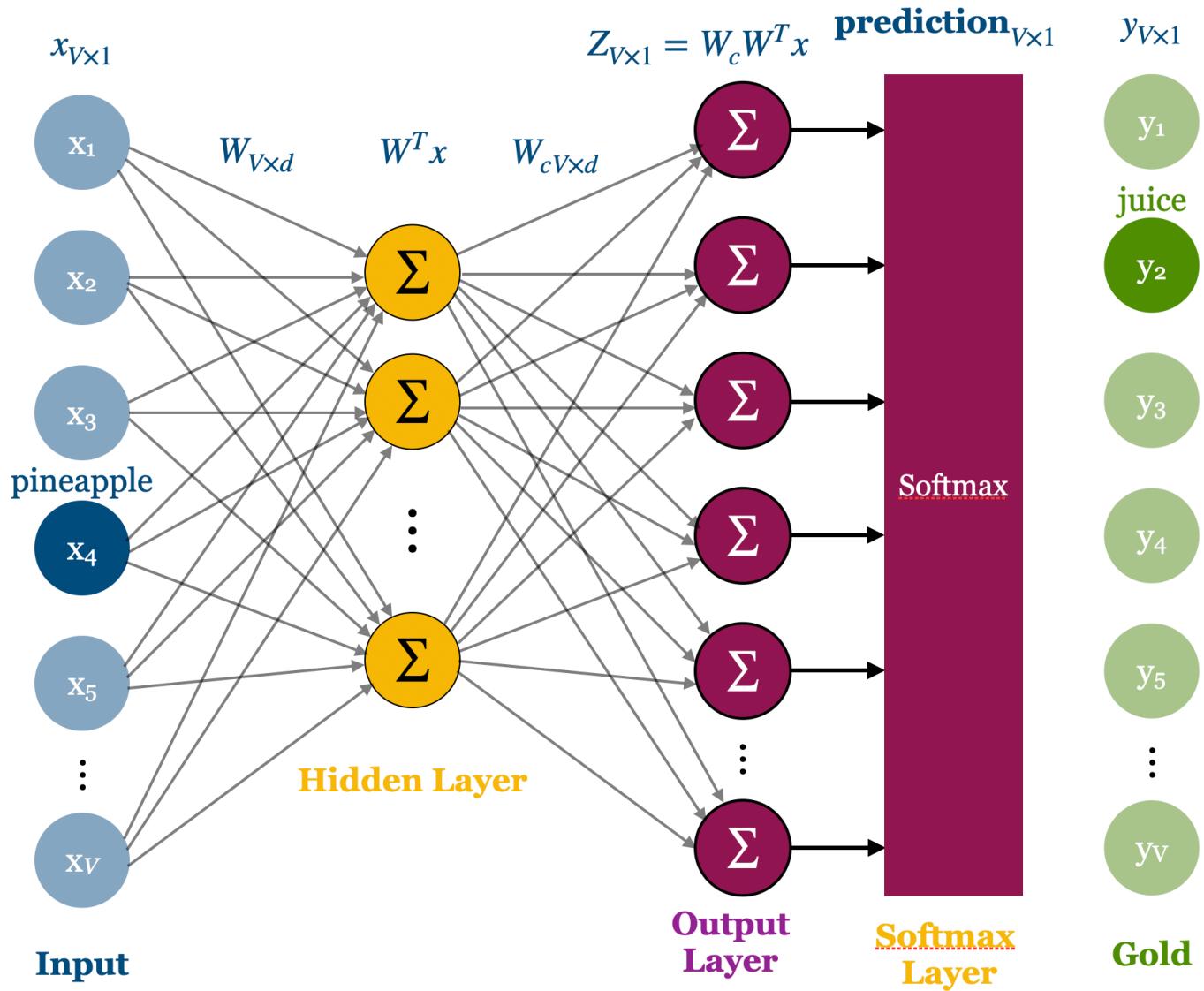


- So in the example above given the target word **pineapple**
 - what's the probability that a randomly picked context word is **juice**.
 - what's the probability that a randomly picked context word is **squeezed**.
- Given a **target word** (i.e., center word) word, predict **context words** (i.e., surrounding words).
- Note that we are using “target” in a different sense here compared to how we use it in supervised machine learning.

A simple neural network architecture ([Source](#))

- an input layer

- a linear hidden layer (without any activation function)
- an output layer with softmax



The Skip-gram Model Example



Add freshly squeezed_{context} pineapple_{target} juice_{context} to your smoothie .

index = 9292 index = 8000 index = 5428

- Size of the vocabulary $V = 10,000$
- Size of the embedding $d = 300$
- Window size 1

Parameters of the model

Input x

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{V \times 1} \xrightarrow{\text{index } 8000} \begin{bmatrix} 0.02 & -0.12 & \cdots & 0.4 \\ \vdots & \vdots & \ddots & \vdots \\ -0.13 & 0.03 & \cdots & -0.2 \\ \vdots & \vdots & \ddots & \vdots \\ -0.28 & 0.22 & \cdots & 0.05 \end{bmatrix}_{V \times d}$$

One-hot encoding of the word
pineapple

Word embedding matrix

Hidden layer output $W^T x$

$$\begin{bmatrix} 0.13 \\ 0.03 \\ \vdots \\ -0.2 \end{bmatrix}_{d \times 1}$$

Select the row of W corresponding to the target word **pineapple**

Shared context matrix

$$\begin{bmatrix} 0.42 & -0.04 & \cdots & 0.03 \\ \vdots & \vdots & \ddots & \vdots \\ 0.3 & -0.5 & \cdots & 0.16 \\ \vdots & \vdots & \ddots & \vdots \\ 0.06 & -0.93 & \cdots & 0.11 \end{bmatrix}_{V \times d} \xleftarrow{\text{index } 5428} \begin{bmatrix} 0.19 \\ \vdots \\ 2.3 \\ \vdots \\ -0.02 \end{bmatrix}_{V \times 1}$$

Prob. dist of $W_C W^T x$

Output layer output
 $W_C W^T x$

Prediction

Softmax layer output "Gold"

$$\begin{bmatrix} 0.011 \\ \vdots \\ 0.21 \\ \vdots \\ 0.002 \end{bmatrix}_{V \times 1} \xrightarrow{\text{index } 5428} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{V \times 1}$$

One-hot encoding of the word
juice

The Skip-gram Model Example



Add freshly squeezed_{context} pineapple_{target} juice_{context} to your smoothie .

index = 9292 index = 8000 index = 5428

- Size of the vocabulary $V = 10,000$
- Size of the embedding $d = 300$
- Window size 1

Parameters of the model

Input x

$$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{V \times 1} \xrightarrow{\text{index } 8000} \begin{bmatrix} 0.02 & -0.12 & \cdots & 0.4 \\ \vdots & \vdots & \ddots & \vdots \\ -0.13 & 0.03 & \cdots & -0.2 \\ \vdots & \vdots & \ddots & \vdots \\ -0.28 & 0.22 & \cdots & 0.05 \end{bmatrix}_{V \times d}$$

One-hot encoding of the word
pineapple

Word embedding matrix

Hidden layer output $W^T x$

$$\begin{bmatrix} 0.13 \\ 0.03 \\ \vdots \\ -0.2 \end{bmatrix}_{d \times 1}$$

Select the row of W corresponding to the target word **pineapple**

Shared context matrix

$$\begin{bmatrix} 0.42 & -0.04 & \cdots & 0.03 \\ \vdots & \vdots & \ddots & \vdots \\ -0.18 & 0.45 & \cdots & 0.26 \\ \vdots & \vdots & \ddots & \vdots \\ 0.06 & -0.93 & \cdots & 0.11 \end{bmatrix}_{V \times d} \xleftarrow{\text{index } 9292} \begin{bmatrix} 0.19 \\ \vdots \\ 0.45 \\ \vdots \\ -0.02 \end{bmatrix}_{V \times 1}$$

Prob. dist of $W_C W^T x$

Output layer output
 $W_C W^T x$

Prediction

Softmax layer output "Gold"

$$\begin{bmatrix} 0.011 \\ \vdots \\ 0.012 \\ \vdots \\ 0.002 \end{bmatrix}_{V \times 1} \xrightarrow{\text{index } 9292} \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}_{V \times 1}$$

One-hot encoding of the word
squeezed

Skip-gram objective

- Consider the conditional probabilities $p(w_c|w_t)$ and set the parameters θ of $p(w_c|w_t; \theta)$ so as to maximize the corpus probability.

$$\arg \max_{\theta} \prod_{(w_c, w_t) \in D} p(w_c | w_t; \theta)$$

- $w_t \rightarrow$ target word
- $w_c \rightarrow$ context word
- $D \rightarrow$ the set of all target and context pairs from the text
- $V \rightarrow$ vocabulary
- Model the conditional probability using softmax of the dot product.
 - Higher the dot product higher the probability and vice-versa.

$$P(w_c | w_t; \theta) = \frac{e^{w_c \cdot w_t}}{\sum_{c' \in V} e^{w_{c'} \cdot w_t}}$$

- Substituting the conditional probability with the softmax of dot product: \$
$$\arg \max_{\theta} \prod_{(w_c, w_t) \in D} P(w_c | w_t; \theta) \approx \prod_{(w_c, w_t) \in D} \frac{e^{w_c \cdot w_t}}{\sum_{c' \in V} e^{w_{c'} \cdot w_t}} \$$$
- Assumption: Maximizing this objective on a large corpus will result in meaningful embeddings for all words in the vocabulary.
- This model is computationally inefficient, and optimizations exist:
 - **Negative sampling:** Instead of computing probabilities for all words, the model updates only a small number of negative samples (randomly selected words that are unlikely to be in the context).
 - **Hierarchical softmax:** Instead of computing softmax over the entire vocabulary, words are arranged in a binary tree structure, allowing predictions to be made in $\log(V)$ time instead of $\mathcal{O}(V)$.

Main hyperparameters of the model

- Dimensionality of the word vectors
- Window size
 - shorter window: more syntactic representation
 - longer window: more semantic representation
 - Mikolov et al. (2015) suggest setting this parameter in the range 5 to 20 for small training datasets and in the range 2 to 5 for large training datasets.

(Optional) Parameters to learn

- Given a corpus with vocabulary of size V , where a word w_i is identified by its index $i \in 1, \dots, V$, learn a vector representation for each w_i by predicting the words that appear in its context.
- Learn the following parameters of the model
 - Suppose $V = 10,000$, $d = 300$, the number of parameters to learn are 6,000,000!

$$\theta = \begin{bmatrix} aardvark_t \\ aback_t \\ \dots \\ zymurgi_t \\ aardvark_c \\ aback_c \\ \dots \\ zymurgi_c \end{bmatrix} \in R^{2dV}$$

?? Questions for you

Exercise 5.1 Select all of the following statements which are **True** (iClicker)

- (A) Word representation created by term-term co-occurrence matrix are long and sparse whereas the ones created by word2vec are short and dense.
- (B) You could pass term-term co-occurrence matrix to **TruncatedSVD** or LSA to get short and dense representations.
- (C) The word2vec algorithm does not require any manually labeled training data.

- (D) When training a word2vec model, it is fine if we do poorly on the fake word prediction task because in the end we only care about the learned weights of the model.
- (E) Given the following table (word 1, word 2) are more similar than (word 1, word 3) in terms of dot products.

	data	fashion	model
word 1	10	0	8
word 2	0	8	3
word 3	6	1	4

💡 V's Solutions



4. More word2vec

4.1 Training word2vec embeddings

- [Original C code](#)
- [GitHub version of the code](#)
- [Gensim](#), an open source Python library that provides a Python interface for word2vec family of algorithms

You can install it as follows:

```
> conda activate 563  
> pip install gensim
```

```
sentences = [["cat", "say", "meow"], ["dog", "say", "woof"]]
sentences
```

```
[['cat', 'say', 'meow'], ['dog', 'say', 'woof']]
```

We need a corpus to be preprocessed and stored as

- A list of lists.
- Each inner list corresponds to a sequence of tokens from a sentence. In other words, each sentence is tokenized and stored as a list of tokens.

Let's look at the word vector of the word *cat*.

```
from gensim.models import Word2Vec
model = Word2Vec(sentences, vector_size=10, window=5, min_count=1) # train wor
```

```
model.wv["cat"]
```

```
array([-0.0960355,  0.05007293, -0.08759586, -0.04391825, -0.000351,
       -0.00296181, -0.0766124,  0.09614743,  0.04982058,  0.09233143],
      dtype=float32)
```

What's the most similar word to the word *cat*?

```
model.wv.most_similar("cat")
```

```
[('say', -0.22418655455112457),
 ('dog', -0.3207966387271881),
 ('meow', -0.36627137660980225),
 ('woof', -0.5381841659545898)]
```

Let's try it out on our small wiki corpus.

```
len(corpus)
```

785

corpus[:4]

```
['Machu Picchu is a 15th-century Inca citadel located in the Eastern Cordillera.  

 'Often referred to as the "Lost City of the Incas", it is the most familiar image  

 'It is located in the Machupicchu District within the Urubamba Province above  

 'The Urubamba River flows past it, cutting through the Cordillera and creating
```

```
sents = MyPreprocessor(corpus)  

wiki_model = Word2Vec(sents, vector_size=100, window=5, min_count=2)
```

wiki_model.wv["tree"][0:10]

```
array([ 0.00714502, -0.00508035, -0.0054067 ,  0.0026014 ,  0.00891363,  

       -0.01356727,  0.0056368 ,  0.02251533, -0.01142082, -0.00982451],  

      dtype=float32)
```

wiki_model.wv.most_similar("tree")

```
[('redwood', 0.5844029188156128),  

 ('president', 0.5573535561561584),  

 ('trees', 0.5283204913139343),  

 ('government', 0.4909103810787201),  

 ('picchu', 0.48084592819213867),  

 ('inca', 0.47769996523857117),  

 ('georgia', 0.47641751170158386),  

 ('ft', 0.4606384336948395),  

 ('state', 0.4584057927131653),  

 ('====', 0.4545993208885193)]
```

- This is good. But if you want good and meaningful representations of words you need to train models on a large corpus such as the whole Wikipedia, which is computationally intensive.
- So instead of training our own models, we use the **pre-trained embeddings**. These are the word embeddings people have trained embeddings on huge corpora and made them available for us to use.

Let's try out Google news pre-trained word vectors.

4.2 Pre-trained embeddings

- Training embeddings is computationally expensive
- For typical corpora, the vocabulary size is greater than 100,000.
- If the size of embeddings is 300, the number of parameters of the model is $2 \times 30,000,000$.
- So people have trained embeddings on huge corpora and made them available.

A number of pre-trained word embeddings are available. The most popular ones are:

- [word2vec](#)
 - trained on several corpora using the word2vec algorithm
- [wikipedia2vec](#)
 - pretrained embeddings for 12 languages
- [GloVe](#)
 - trained using [the GloVe algorithm](#)
 - published by Stanford University
- [fastText pre-trained embeddings for 294 languages](#)
 - trained using [the fastText algorithm](#)
 - published by Facebook

Let's try Google News pre-trained embeddings.

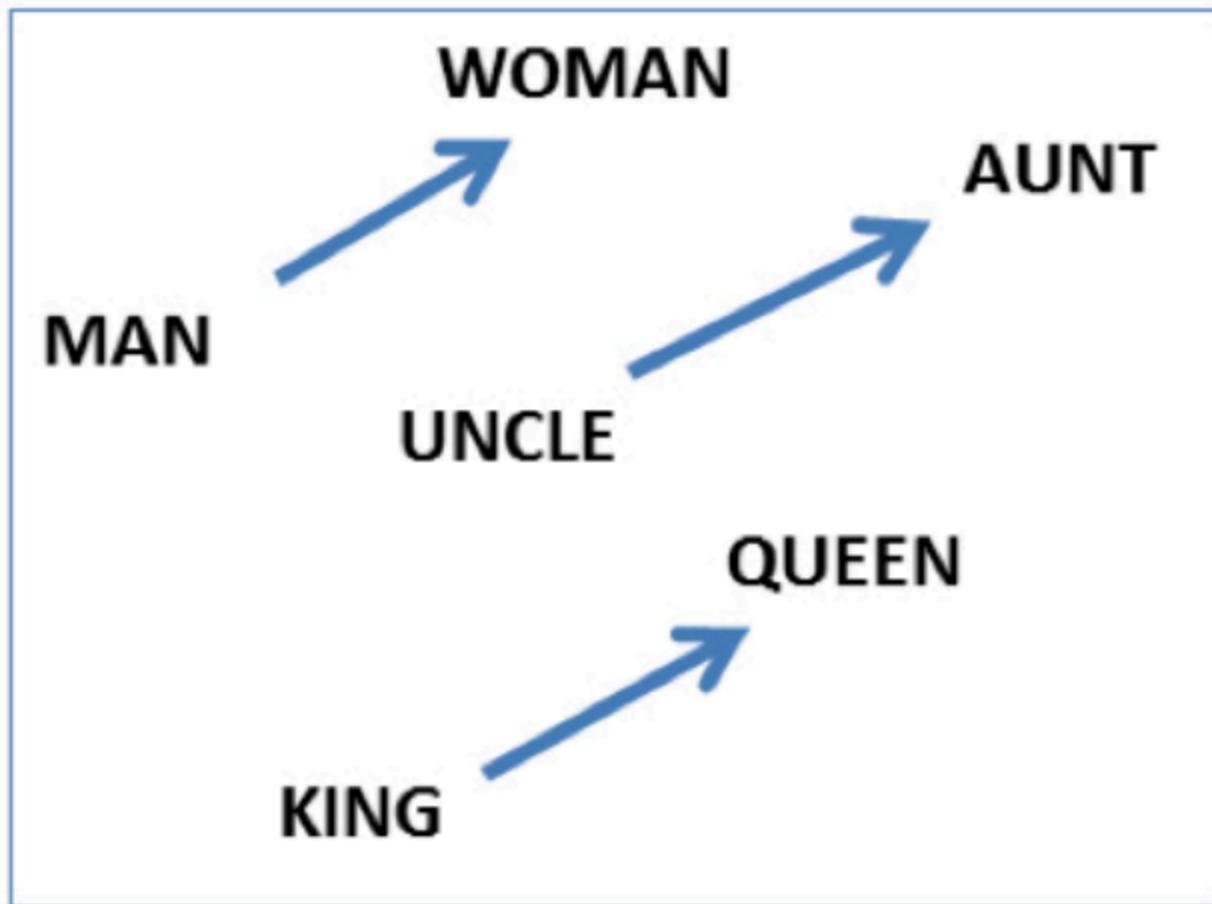
- You can download pre-trained embeddings from their original source.
- [Gensim](#) provides an api to conveniently load them.

```
google_news_vectors = api.load("word2vec-google-news-300")
```

4.3 Success of word2vec

- This analogy example often comes up when people talk about word2vec, which was used by the authors of this method.

- **MAN : KING :: WOMAN : ?**
 - What is the word that is similar to **WOMAN** in the same sense as **KING** is similar to **MAN**?
- Perform a simple algebraic operations with the vector representation of words.
 $\vec{X} = \vec{\text{KING}} - \vec{\text{MAN}} + \vec{\text{WOMAN}}$
- Search in the vector space for the word closest to \vec{X} measured by cosine distance.



(Credit: Mikolov et al. 2013)

```

def analogy(word1, word2, word3, model=google_news_vectors):
    """
    Returns analogy word using the given model.

    Parameters
    -----
    word1 : (str)
        word1 in the analogy relation
    word2 : (str)
        word2 in the analogy relation
    word3 : (str)
        word3 in the analogy relation
    model :
        word embedding model

    Returns
    -----
    pd.DataFrame
    """
    print("%s : %s :: %s : ?" % (word1, word2, word3))
    sim_words = model.most_similar(positive=[word3, word2], negative=[word1])
    return pd.DataFrame(sim_words, columns=["Analogy word", "Score"])

```

```
analogy("man", "king", "woman")
```

```
man : king :: woman : ?
```

	Analogy word	Score
0	queen	0.711819
1	monarch	0.618967
2	princess	0.590243
3	crown_prince	0.549946
4	prince	0.537732
5	kings	0.523684
6	Queen_Consort	0.523595
7	queens	0.518113
8	sultan	0.509859
9	monarchy	0.508741

```
analogy("Montreal", "Canadiens", "Vancouver")
```

Montreal : Canadiens :: Vancouver : ?

	Analogy word	Score
0	Canucks	0.821327
1	Vancouver_Canucks	0.750401
2	Calgary_Flames	0.705471
3	Leafs	0.695783
4	Maple_Leafs	0.691617
5	Thrashers	0.687504
6	Avs	0.681716
7	Sabres	0.665307
8	Blackhawks	0.664625
9	Habs	0.661023

```
analogy("Toronto", "UofT", "Vancouver")
```

Toronto : UofT :: Vancouver : ?

	Analogy word	Score
0	SFU	0.579245
1	UVic	0.576921
2	UBC	0.571431
3	Simon_Fraser	0.543464
4	Langara_College	0.541347
5	UVIC	0.520495
6	Grant_MacEwan	0.517273
7	UFV	0.514150
8	Ubyssey	0.510421
9	Kwantlen	0.503807

```
analogy("Gauss", "mathematician", "Bob_Dylan")
```

Gauss : mathematician :: Bob_Dylan : ?

	Analogy word	Score
0	singer_songwriter_Bob_Dylan	0.520782
1	poet	0.501191
2	Pete_Seeger	0.497143
3	Joan_Baez	0.492307
4	sitarist_Ravi_Shankar	0.491968
5	bluesman	0.490930
6	jazz_musician	0.489593
7	Joni_Mitchell	0.487740
8	Billie_Holiday	0.486664
9	Johnny_Cash	0.485722

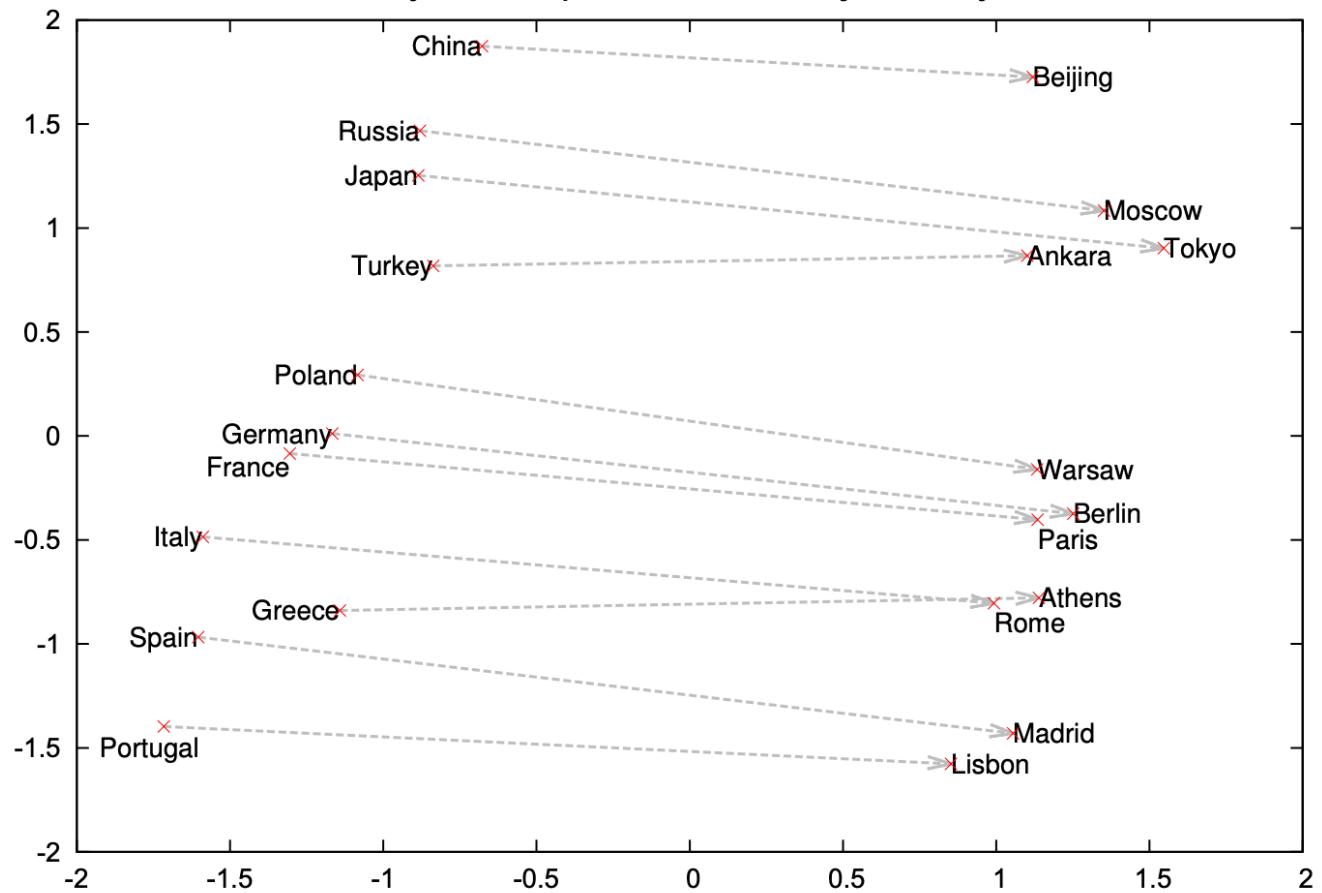
```
analogy("Canada", "maple", "China")
```

Canada : maple :: China : ?

	Analogy word	Score
0	bamboo	0.513054
1	Chinese	0.496991
2	osmanthus	0.496046
3	bean_curd	0.489001
4	zhou	0.483002
5	teak_mahogany	0.478263
6	star_anise	0.477919
7	Longjing_tea	0.472245
8	Hu	0.471446
9	Longjing	0.470437

So you can imagine these models being useful in many meaning-related tasks.

Country and Capital Vectors Projected by PCA



(Credit: Mikolov et al. 2013)

Examples of semantic and syntactic relationships

Type of relationship	Word Pair 1		Word Pair 2	
Common capital city	Athens	Greece	Oslo	Norway
All capital cities	Astana	Kazakhstan	Harare	Zimbabwe
Currency	Angola	kwanza	Iran	rial
City-in-state	Chicago	Illinois	Stockton	California
Man-Woman	brother	sister	grandson	granddaughter
Adjective to adverb	apparent	apparently	rapid	rapidly
Opposite	possibly	impossibly	ethical	unethical
Comparative	great	greater	tough	tougher
Superlative	easy	easiest	lucky	luckiest
Present Participle	think	thinking	read	reading
Nationality adjective	Switzerland	Swiss	Cambodia	Cambodian
Past tense	walking	walked	swimming	swam
Plural nouns	mouse	mice	dollar	dollars
Plural verbs	work	works	speak	speaks

(Credit: Mikolov 2013)

4.4 Implicit biases and stereotypes in word embeddings

```
analogy("man", "computer_programmer", "woman")
```

```
man : computer_programmer :: woman : ?
```

	Analogy word	Score
0	homemaker	0.562712
1	housewife	0.510505
2	graphic_designer	0.505180
3	schoolteacher	0.497949
4	businesswoman	0.493489
5	paralegal	0.492551
6	registered_nurse	0.490797
7	saleswoman	0.488163
8	electrical_engineer	0.479773
9	mechanical_engineer	0.475540



- Embeddings reflect gender stereotypes present in broader society.
- They may also amplify these stereotypes because of their widespread usage.
- See the paper [Man is to Computer Programmer as Woman is to](#)

Most of the modern embeddings are de-biased for some obvious biases.

4.5 Other popular methods to get embeddings

fastText

- NLP library by Facebook research
- Includes an algorithm which is an extension to word2vec
- Helps deal with unknown words elegantly
- Breaks words into several n-gram subwords
- Example: trigram sub-words for *berry* are *ber*, *err*, *rry*
- Embedding(*berry*) = embedding(*ber*) + embedding(*err*) + embedding(*rry*)

(Optional) GloVe: Global Vectors for Word Representation

- Starts with the co-occurrence matrix
 - Co-occurrence can be interpreted as an indicator of semantic proximity of words
- Takes advantage of global count statistics
- Predicts co-occurrence ratios
- Loss based on word frequency

? ? Questions for you

Exercise 5.2 Select all of the following statements which are **True** (iClicker)

- (A) Suppose you learn word embeddings for a vocabulary of 20,000 words using word2vec. Then each dense word embedding associated with a word is of size 20,000 to make sure that we capture the full range of meaning of that word.
- (B) If you try to get word vector for a word outside the vocabulary of your word2vec model, it will throw an error.

- (C) A model trained with fastText is likely to contain representation for words even when they are not in the vocabulary.
- (D) Suppose I have a HUGE corpus of biomedical text, which contains many domain-specific words and abbreviations. It makes sense to train our own embeddings to get word representations of the words in the corpus rather than using pre-trained Google News embeddings.
- (E) Suppose my corpus gets updated very often, and the new text tends to contain many unknown words. It makes sense to either train fastText or use pre-trained embeddings trained using fastText.

V's Solutions!



Final comments, summary, and reflection

Word embeddings

- Word embeddings give a representation of individual words in a vector space so that
 - Distance between words in this vector space indicate the relationship between them.
- Word representations
 - One-hot encoding
 - not able to capture similarities between words
 - Term-term co-occurrence matrix
 - long and sparse representations
 - word2vec
 - short and dense representations

word2vec

- word2vec is a recent alternative to LSA.
- One of the most widely used and influential algorithms in the last decade.
- Skipgram model predicts surrounding words (context words) given a target word.
- We are not interested in the prediction task itself but we are interested in the learned weights which we use as word embeddings.
- Freely available code and pre-trained models.
- Generally do not train these models yourself. It's very likely that there are pre-trained embeddings available for the domain of your interest.

Pre-trained embeddings

- Available for many different languages in a variety of domains.
- Finally beware of biases and stereotypes encoded in these word embeddings!

Resources

- [Embedding Projector](#)
- [Word embedding workshop by Rachel Thomas](#)
- [Distributed representations of words and phrases and their compositionality](#)
- [Efficient estimation of word representations in vector space](#)
- [Linguistic regularities in continuous space word representations](#)
- [Enriching Word Vectors with Subword Information](#)

Fun tools

[wevi: word embedding visual inspector](#)