Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 2: Word embeddings

Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 2 月 25 日

Slides adapted in part from UPenn CIS 7000: LLMs, Stanford CS224n: Natural Language Processing with Deep Learning, and CS324 LLMs.

What is a word (Cambridge dictionary)

*Meaning of **word** in English*

word

noun

UK /wɜːd/ **US** /wɜːd/

word *noun* (LANGUAGE UNIT)

Add to word list

A1 [C]

a single unit of language that has meaning and can be spoken or written:

- *Your essay should be no more than two thousand words long.*
- *Some words are more difficult to spell than others.*
- **word for** *What's the word for bikini in French?*
- **find the right word** *It's sometimes difficult to find exactly the right word to express what you want to say.*

What is the meaning of "meaning"?

- Cambridge dictionary: The meaning of something is what it expresses or represents.
- Symbol $\Leftrightarrow$ idea of thing
- For example, when we say "Cat", it means

Role of dictionaries in classic NLP

- **Word definitions:** Dictionaries provide meanings of words to help machines understand vocabulary.
- **Part-of-Speech (POS) Tagging (词性标注)**: Dictionaries categorize words (e.g., noun, verb) for syntactic analysis.
- **Morphological Analysis (形态分析)**: Dictionaries help identify word roots, prefixes, and suffixes.
- **Synonyms (同义词) and Antonyms （反义词）**: Used for tasks like paraphrasing and semantic analysis.

WordNet

- WordNet: A Lexical Database for English(George A. Miller, 1995; Christiane Fellbaum, 1998)
- The website for WordNet is `https://wordnet.princeton.edu/`.

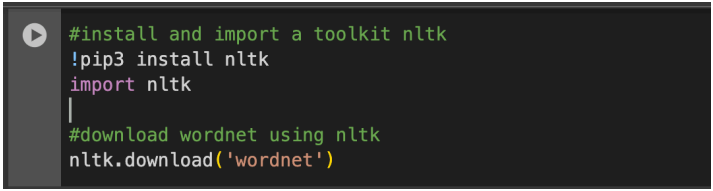
Install WordNet using nltk

- First, install the `nltk` package (here I adopt `pip3`):

  ```
  pip3 install nltk
  ```

- Then, use the following Python code to download WordNet:

  ```
  import nltk
  nltk.download('wordnet')
  ```



An example using Google's colab.

What is NLTK?

- The Natural Language Toolkit (NLTK) is a powerful Python library for NLP.
- It provides easy-to-use interfaces to over 50 corpora and lexical resources, making it a valuable tool for linguistic research and text analysis.
- NLTK includes a comprehensive suite of text processing tools for tasks such as tokenization, stemming, tagging, parsing, classification, and semantic reasoning.

WordNet contains a set of synonyms

- Using WordNet, we can find synonyms of a certain word (here, I use "mean" as an example:

```
from nltk.corpus import wordnet as wn
print(wn.synsets("mean"))
```

- The output includes synsets (sets of synonyms), such as "entail.v.01":
  - "entail" is the synonym.
  - "v" indicates the part of speech (verb in this case).
  - "01" is the sense number, as a word can have multiple meanings (senses).

```
from nltk.corpus import wordnet as wn
wn.synsets("mean")
```

```
[Synset('mean.n.01'),
 Synset('mean.v.01'),
 Synset('entail.v.01'),
 Synset('mean.v.03'),
 Synset('intend.v.01'),
 Synset('mean.v.05'),
 Synset('think_of.v.04'),
 Synset('mean.v.07'),
 Synset('average.s.01'),
 Synset('hateful.s.02'),
 Synset('base.s.05'),
 Synset('mean.s.04'),
 Synset('beggarly.s.01'),
 Synset('mean.s.06'),
 Synset('beggarly.s.02'),
 Synset('bastardly.s.02')]
```

```python
synsets = wn.synsets('mean')
for synset in synsets:
    print(f"Synset: {synset.name()}")
    print(f"Definition: {synset.definition()}")
    print(f"Examples: {synset.examples()}")
    print(f"POS: {synset.pos()}")
    print()
```

```
Synset: mean.n.01
Definition: an average of n numbers computed by adding some function of the numbers and dividing by some function of n
Examples: []
POS: n

Synset: mean.v.01
Definition: mean or intend to express or convey
Examples: ['You never understand what I mean!', 'what do his words intend?']
POS: v

Synset: entail.v.01
Definition: have as a logical consequence
Examples: ['The water shortage means that we have to stop taking long showers']
POS: v
```

Reorganizing it Using a Dictionary

```python
# Define a mapping of part-of-speech tags to their full names
part_of_speech_mapping = {
    'n': 'noun',
    'v': 'verb',
    's': 'adj (s)',
    'a': 'adj',
    'r': 'adv'
}

# Iterate through each synset for the word "mean"
for synset in wn.synsets("mean"):
    # Get the full part-of-speech name using the mapping
    pos_name = part_of_speech_mapping[synset.pos()]

    # Extract all lemma names (synonyms) for the current synset
    lemma_names = [lemma.name() for lemma in synset.lemmas()]

    # Join the lemma names into a comma-separated string
    lemma_names_str = ", ".join(lemma_names)

    # Print the part-of-speech and the corresponding lemma names
    print(f"{pos_name}: {lemma_names_str}")
```

(a) Code demonstrating dictionary usage

```
noun: mean, mean_value
verb: mean, intend
verb: entail, imply, mean
verb: mean, intend, signify, stand_for
verb: intend, mean, think
verb: mean
verb: think_of, have_in_mind, mean
verb: mean
adj (s): average, mean
adj (s): hateful, mean
adj (s): base, mean, meanspirited
adj (s): mean
adj (s): beggarly, mean
adj (s): mean, mingy, miserly, tight
adj (s): beggarly, mean
adj (s): bastardly, mean
```

(b) Output of the code

Figure: Reorganizing it using a dictionary

Inspired by Stanford CS224N

Hypernyms（上位词）and hyponymy（下位词）

0 秒

```python
from nltk.corpus import wordnet as wn

# Get the first synset for the word "dog"
dog = wn.synsets('dog')[0]

# Get hypernyms (more general terms)
print("Hypernyms:")
for hypernym in dog.hypernyms():
    print(hypernym.name())

# Get hyponyms (more specific terms)
print("\nHyponyms:")
for hyponym in dog.hyponyms():
    print(hyponym.name())
```

```
Hypernyms:
canine.n.02
domestic_animal.n.01

Hyponyms:
corgi.n.01
leonberg.n.01
cur.n.01
pug.n.01
dalmatian.n.02
pooch.n.01
lapdog.n.01
```

Limitations of using lexical resources

- Limited Coverage: WordNet focuses on general language and may lack specialized vocabulary for domains like medicine, law, or technology.
- Lack of Dynamic Knowledge (static): WordNet is not updated frequently, which means it may not reflect current language usage or emerging meanings of words. (e.g., "卧龙凤雏").
- Bias and Subjectivity: The creation of WordNet involves human judgment, which can introduce biases or subjective interpretations of word meanings and relationships.
- Requires significant human effort for creation and maintenance.
- Computationally expensive for large-scale tasks.

Recall an example of word embeddings

Place your text below:

It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness, it was the epoch of belief, it was the epoch of incredulity, it was the season of Light, it was the season of Darkness, it was the spring of hope, it was the winter of despair, we had everything before us, we had nothing before us, we were all going direct to Heaven, we were all going direct the other way—in short, the period was so far like the present period, that some of its noisiest authorities insisted on its being received, for good or for evil, in the superlative degree of comparison only.

Specify model and parameters to generate dataset:

Model	Skip-gram

```
Input: of
[0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
Output: times
[0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
```

Here we convert each word to a numerical vector.

Represent words using one-hot vectors

- One-hot vector: one element is 1 and the rest are 0s (e.g., [0,0,1,0,0]).
- Words can be represented as one-hot vectors by treating them as discrete symbols.
- Example: for a vocabulary {dog, cat, dogs}, one-hot encoding yields

$$\text{dog} = [1, 0, 0], \qquad \text{dogs} = [0, 1, 0], \qquad \text{cat} = [0, 0, 1].$$

- The dimension of each vector equals the size of the vocabulary (total number of words), which may be 50K+ in practice.
- Limitation: can not measure the similarity between words. (how?)

Similarity between words

- For 2 vectors a and b, the similarity between them can be measured by the angle between them.
- Mathematically, this is expressed using the cosine similarity: $\cos(a, b) = \frac{a^T b}{\|a\|_2 \cdot \|b\|_2}$, here $\|\cdot\|_2$ denotes the Euclidean norm of a vector.
- When using one-hot encoding, the vectors of any two distinct words are orthogonal, and the cosine similarity is 0.
- For example, $\cos(\text{dog}, \text{dogs}) = 0$, even though "dogs" is the plural form of "dog" and the two words are semantically closely related.

Representing Words Using Dense Vectors

- To measure the similarity between words, we represent each word using a dense vector.
- For example:

$$\text{dog} = [0.6, 0.8, 0], \qquad \text{dogs} = [0.6, 0.7, \sqrt{0.15}].$$

- The cosine similarity between these vectors is $\cos(\text{dog}, \text{dogs}) = 0.92$ (close to 1, representing high similarity).
- These dense word vectors are referred to as word embeddings or word representations.

Google pretrained word embeddings using Word2Vec

- We first use a pre-trained word embedding model developed by Google using the Word2Vec model (Tomas Mikolov et al., 2013), known as "GoogleNews-vectors-negative300".
- Each embedding in this model is a 300-dimensional vector.
- Using this model, we obtain

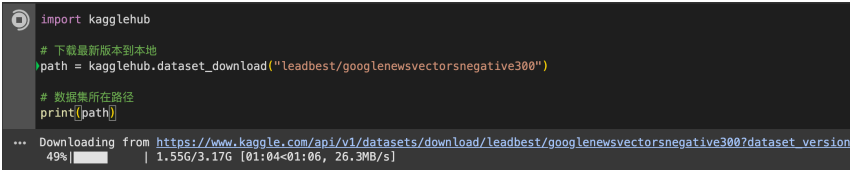
$$\text{dog} = [0.051, -0.022, \cdots, -0.355, 0.223]$$
$$\text{dogs} = [-0.021, -0.012, \cdots, -0.127, 0.122].$$

- The cosine similarity between the words "dog" and "dogs" is calculated to be 0.868049, indicating a high degree of semantic similarity.

Download a pretrained model using Kaggle Hub

- Kaggle Hub provides a convenient way to download pretrained embeddings.
- Use the following Python code to download the "GoogleNews-vectors-negative300" model and display its download path:

```
import kagglehub
path = kagglehub.dataset_download("leadbest/googlenewsvectorsnegative300")
print(path)
```



Load the pretrained model

- Here we use a library called gensim which can load the Word2Vec format

```
import os
import gensim
file_path = os.path.join(path, "GoogleNews-vectors-negative300.bin")
model = gensim.models.KeyedVectors.load_word2vec_format(file_path, binary=True)
```

Obtaining Word Embeddings

- Once the model is loaded, we can obtain the embeddings as follows

```
dog = model['dog']
dogs = model['dogs']
print(f'dog={dog}, dogs={dogs}')
```

```
dog = model['dog']
dogs = model['dogs']

print(f'dog={dog}, dogs={dogs}')
```

```
dog=[ 5.12695312e-02 -2.23388672e-02 -1.72851562e-01  1.61132812e-01
 -8.44726562e-02  5.73730469e-02  5.85937500e-02 -8.25195312e-02
 -1.53808594e-02 -6.34765625e-02  1.79687500e-01 -4.23828125e-01
 -2.25830078e-02 -1.66015625e-01 -2.51464844e-02  1.07421875e-01
 -1.99218750e-01  1.59179688e-01 -1.87500000e-01 -1.20117188e-01
  1.55273438e-01 -9.91210938e-02  1.42578125e-01 -1.64062500e-01
 -8.93554688e-02  2.00195312e-01 -1.49414062e-01  3.20312500e-01
  3.28125000e-01  2.44140625e-02 -9.71679688e-02 -8.20312500e-02
 -3.63769531e-02 -8.59375000e-02 -9.86328125e-02  7.78198242e-03
 -1.34277344e-02  5.27343750e-02  1.48437500e-01  3.33984375e-01
  1.66015625e-02 -2.12890625e-01 -1.50756836e-02  5.24902344e-02
 -1.07421875e-01 -8.88671875e-02  2.49023438e-01 -7.03125000e-02
 -1.59912109e-02  7.56835938e-02 -7.03125000e-02  1.19140625e-01
  2.29492188e-01  1.41601562e-02  1.15234375e-01  7.50732422e-03
```

Pretrained Word Similarity

- Using the pretrained model, you can find the most similar words to a given word: `model.most_similar('dog')`

```
0 秒
model.most_similar('dog')

[('dogs', 0.8680489659309387),
 ('puppy', 0.8106428384780884),
 ('pit_bull', 0.780396044254303),
 ('pooch', 0.7627376914024353),
 ('cat', 0.7609457969665527),
 ('golden_retriever', 0.7500901818275452),
 ('German_shepherd', 0.7465174198150635),
 ('Rottweiler', 0.7437615394592285),
 ('beagle', 0.7418621778488159),
 ('pup', 0.740691065788269)]
```

Install required packages

- Install requires packages using pip3 or anaconda

  ```
  pip3 install nltk gensim scikit-learn matplotlib numpy
  ```

- nltk gensim are NLP packages.
- scikit-learn is a library for machine learning.
- matplotlib is a python library for data visualizations.
- numpy is a fundamental package for scientific computing in Python.

Input your text

- Input your text as follows.

```
paragraphs = [
    "I am currently an assistant professor at The Paula and Gregory Chow Institute for
    Wang Yanan Institute for Studies in Economics (WISE), and the School of Economics (
    "From 2022 to 2024, I served as a visiting scholar
    in the Department of Statistics and Data Science at the Wharton School, University
    "During this time, I had the privilege of working under the joint supervision of
    Professor Guang Cheng from the University of California, Los Angeles,
    and Professor Weijie Su from the University of Pennsylvania.",
    "Prior to that, I completed my Ph.D. at The Hong Kong Polytechnic University,
    where I was advised by Professor Xin Guo from the University of Queensland and Prof
    "My research interests primarily lie in machine learning,
    with a focus on foundation models, deep learning, federated learning, learning theo
    "Additionally, I am interested in privacy-preserving data science,
    specifically differential privacy and its applications to machine learning.",
    "Finally, I am also interested in synthetic data generation, particularly in the co
    More details about my work and publications can be found on my publications page ar
]
```

Tokenize your text

- Tokenize your text using tokenization tools from `nltk.tokenize` (details will be covered in the tokenization session).

```
import nltk
from nltk.tokenize import word_tokenize
nltk.download('punkt_tab')
tokenized_paragraphs = [word_tokenize(para.lower()) for para in paragraphs]
print(tokenized_paragraphs)
```

Outputs of tokenization

```python
import nltk
from nltk.tokenize import word_tokenize

# Download NLTK data (only needed once)
nltk.download('punkt_tab')

# Tokenize each paragraph
tokenized_paragraphs = [word_tokenize(para.lower()) for para in paragraphs]
print(tokenized_paragraphs)
```

```
[['i', 'am', 'currently', 'an', 'assistant', 'professor', 'at', 'the', 'paula', 'and', 'gregory', 'chow', 'institute', 'for', 'studies', 'in', 'economics',
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
```

Train a Word2Vec model using your text

- Train a Word2Vec model using your text via gensim.

```
from gensim.models import Word2Vec
model = Word2Vec(sentences=tokenized_paragraphs,
vector_size=100, window=5, min_count=1, workers=4)
```

Get the embeddings trained by your text

```
import numpy as np

word_vectors = model.wv


word = "professor"
embedding = model.wv[word]

print(f"Embedding for '{word}': {embedding}")
print(f"Shape of embedding: {np.array(embedding).shape}")
```

Outputs

```python
# Get the vector for a specific word
import numpy as np
# Get all word vectors
word_vectors = model.wv

# Get the embedding for a specific word
word = "professor"
embedding = model.wv[word]

print(f"Embedding for '{word}': {embedding}")
print(f"Shape of embedding: {np.array(embedding).shape}")  # (vector_size,)
```

```
Embedding for 'professor': [-0.0051361  -0.00659803 -0.00782192  0.0082907  -0.00194871 -0.00697739
 -0.00405444  0.00530236 -0.00291081 -0.00385124  0.00161313 -0.00290492
 -0.00157205  0.00115802 -0.00293672  0.00845319  0.00390112 -0.00996939
  0.0062117  -0.00691239  0.00084195  0.00444398 -0.00509481 -0.00214931
  0.00816469 -0.00429123 -0.00770628  0.00919645 -0.00221356 -0.00475558
  0.00860605  0.00433254  0.00440554  0.00912381 -0.00848435  0.00534433
  0.00203672  0.00415794  0.00172313  0.00437222  0.00453679  0.00607285
 -0.00324986 -0.00462353 -0.00038177  0.00248753 -0.00331359  0.00600793
  0.00425319  0.00783358  0.00258004  0.00803349 -0.00140195  0.00807743
  0.00376856 -0.00798078 -0.00392784 -0.00246213  0.00488229 -0.00079537
 -0.00276272  0.0077916   0.00939347 -0.00160628 -0.00521087 -0.00459212
 -0.00487027 -0.00955932  0.00121374 -0.00413175  0.00248223  0.0056771
 -0.00397891 -0.00959309  0.00162366 -0.00671032  0.00247748 -0.00377529
  0.00704106  0.00066801  0.00351811 -0.00272078 -0.00180958  0.00774603
  0.00135684 -0.00591442 -0.00764247  0.00123657  0.00655957  0.00558367
 -0.00891899  0.00867222  0.00409578  0.00750228  0.00987858 -0.007242
```

Save your model

- Use the following code to save the trained Word2Vec model. The os library is used to handle directory creation and file paths (interacting with your operating system).

```
import os

save_path = "/path/to/your/directory/file"

os.makedirs(os.path.dirname(save_path), exist_ok=True)

model.save(save_path)
```

An example

```python
import os

# Specify the path where you want to save the model
save_path = "/content/drive/MyDrive/word2vec_model/word2vec_model"

# Ensure the directory exists
os.makedirs(os.path.dirname(save_path), exist_ok=True)

# Save the model to the specified path
model.save(save_path)
```

Load your model

- The saved Word2Vec model can be loaded using the following code:

```
load_path = "/path/to/your/directory/file"


loaded_model = Word2Vec.load(load_path)

vector = loaded_model.wv['professor']
print(vector)
```

Example

```
load_path = "/content/drive/MyDrive/word2vec_model/word2vec_model"

# Load the model
loaded_model = Word2Vec.load(load_path)

# Use the loaded model
vector = loaded_model.wv['professor']
print(vector)
```

```
[-0.0051361  -0.00659803 -0.00782192  0.0082907  -0.00194871 -0.00697739
 -0.00405444  0.00530236 -0.00291081 -0.00385124  0.00161313 -0.00290492
 -0.00157205  0.00115802 -0.00293672  0.00845319  0.00390112 -0.00996939
  0.0062117  -0.00691239  0.00084195  0.00444398 -0.00509481 -0.00214931
  0.00816469 -0.00429123 -0.00770628  0.00919645 -0.00221356 -0.00475558
  0.00860605  0.00433254  0.00440554  0.00912381 -0.00848435  0.00534433
  0.00203672  0.00415794  0.00172313  0.00437222  0.00453679  0.00607285
 -0.00324986 -0.00462353 -0.00038177  0.00248753 -0.00331359  0.00600793
  0.00425319  0.00783358  0.00258004  0.00803349 -0.00140195  0.00807743
  0.00376856 -0.00798078 -0.00392784 -0.00246213  0.00488229 -0.00079537
 -0.00276272  0.0077916   0.00939347 -0.00160628 -0.00521087 -0.00459212
 -0.00487027 -0.00955932  0.00121374 -0.00413175  0.00248223  0.0056771
 -0.00397891 -0.00959309  0.00162366 -0.00671032  0.00247748 -0.00377529
```

Visualize your model

- Use t-SNE (t-Distributed Stochastic Neighbor Embedding) to visualize the trained word embeddings in a 2d space.

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import numpy as np


words = list(model.wv.index_to_key)
vectors = [model.wv[word] for word in words]


tsne = TSNE(n_components=2, random_state=0, perplexity=5)
vectors_2d = tsne.fit_transform(np.array(vectors))
```

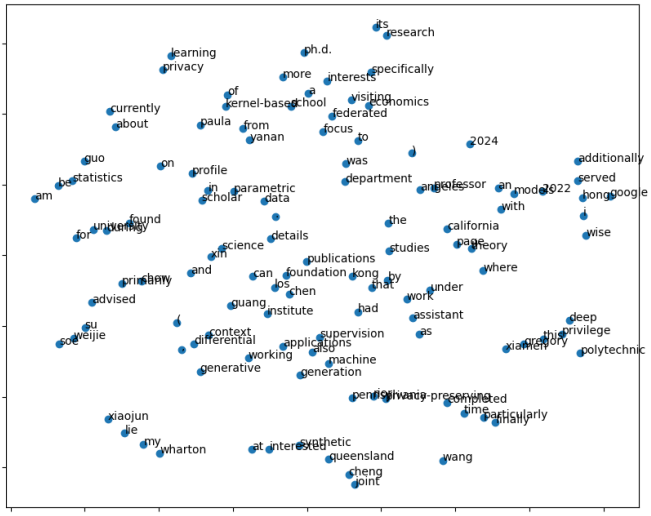
Plot the 2d embeddings

```
plt.figure(figsize=(10, 8))
plt.scatter(vectors_2d[:, 0], vectors_2d[:, 1])


for i, word in enumerate(words):
    plt.annotate(word, xy=(vectors_2d[i, 0], vectors_2d[i, 1]))

plt.show()
```

Result


its
research
ph.d.
learning
privacy
specifically
more
interests
of
a
visiting
kernel-based
school
economics
currently
federated
about
paula
from
focus
yanan
to
2024
guo
was
additionally
on
profile
department
professor
served
statistics
in
parametric
scholar
data
model
2022
hong
kong
with
found
the
california
for
details
wise
science
studies
page
theory
publications
and
can
foundation
kong
by
where
primary
los
that
under
chen
advised
guang
work
institute
had
assistant
deep
su
context
supervision
as
privilege
weijie
differential
applications
also
machine
generative
generation
polytechnic
xiaojun
completed
time
finally
wharton
synthetic
queensland
wang
cheng
joint

Assignment 1

- Train a Word2Vec model using your own text data. Ensure your code includes detailed comments explaining the purpose of each line.
- Visualize the trained Word2Vec model to gain insights into the word embeddings.
- **Optional:** Prompt any LLM for guidance on training a Word2Vec model. Identify any potential bugs or inaccuracies in their instructions and debug them.

The idea of Word2Vec (Mikolov et al. 2013)

- For each position t in the text, identify a center word (c) and its context words (o).
- Use the similarity between word vectors of c and o to calculate the probability of o given c (or vice versa).
- Updating the vectors to maximize the probability.

An example of a window

- Consider "The quick brown fox jumps over the lazy dog".
- A window determines how many words to the left and right of the center word are considered as its context.
- For instance, with a window size of 2, the context words for the center word "jumps" are "brown fox" (left) and "over the" (right).

A Probability Model

- For a center word x_t and its context word x_{t+j}, compute the conditional probability $p(x_{t+j} \mid x_t)$.
- Let the window size be M. Then, j can take integer values from $-M$ to M.
- The likelihood of the model is given by:

$$L(p) = \prod_{t=1}^{T} \prod_{-M \leq j \leq M} p(x_{t+j} \mid x_t).$$

- Objective: Find p that maximizes the likelihood.

Two Questions (Next Lecture)

- How to parameterize p?
- How to find p that maximizes the likelihood?