DSA4213 Natural Language Processing for Data Science

Doudou Zhou (ddzhou@nus.edu.sg)

Lecture 1 Introduction and Word Embedding

Lecture Plan

Lecture 1: Introduction and Word Embedding

1. The course
2. Introduction to natural language processing
3. Word embedding
   - Introduction
   - Word2vec
   - Co-occurrence based word embeddings
   - SPPMI-SVD
   - GloVe
4. Evaluating word vectors
5. Word senses and word sense ambiguity
6. Assignment 1

Learning goals

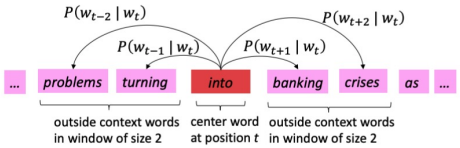
1. The foundations of the effective modern methods for deep learning applied to NLP
   - Basics first: word vectors, feed-forward networks, recurrent networks, attention
   - Then key methods used in NLP in recent years: encoder-decoder models, transformers, pretraining, post-training, efficient adaptation, interpretability, language model agents, etc.
2. A big picture understanding of human languages and the difficulties in understanding and producing them via computers
3. An understanding of and **ability to build systems** (in PyTorch) for some of the major problems in NLP:
   - Word meaning, machine translation, question answering

Learning goals

Word vectors

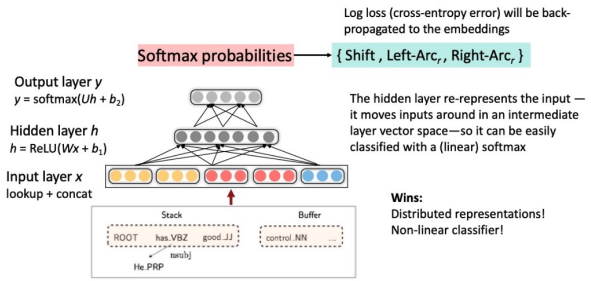
Word2Vec Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



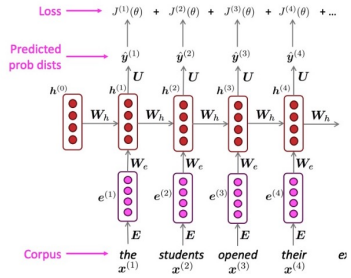
Neural NLP

(A simple feed-forward neural network multi-class classifier)

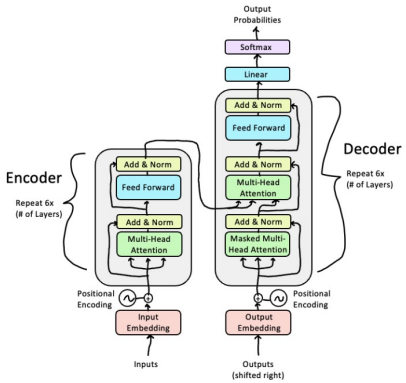


LMs/RNNs/LSTMs

Training an RNN Language Model



Transformers



Pretraining, Post-training

What kinds of things does pretraining teach?

There's increasing evidence that pretrained models learn a wide variety of things about the statistical properties of language. Taking our examples from the start of class:

- *Stanford University is located in _____, California.* [Trivia]
- *I put ___ fork down on the table.* [syntax]
- *The woman walked across the street, checking for traffic over ___ shoulder.* [coreference]
- *I went to the ocean to see the fish, turtles, seals, and _____.* [lexical semantics/topic]
- *Overall, the value I got from the two hours watching it was the sum total of the popcorn and the drink. The movie was ___.* [sentiment]
- Iroh went into the kitchen to make some tea. Standing next to Iroh, Zuko pondered his destiny. Zuko left the _____. [some reasoning – this is harder]
- I was thinking about the sequence that goes 1, 1, 2, 3, 5, 8, 13, 21, ____ [some basic arithmetic; they don't learn the Fibonnaci sequence]
- Models also learn – and can exacerbate racism, sexism, all manner of bad biases.

… and more

Benchmarking
Reasoning
Responsible NLP
Brain-Computer Interfaces
…

4

Course work and grading policy

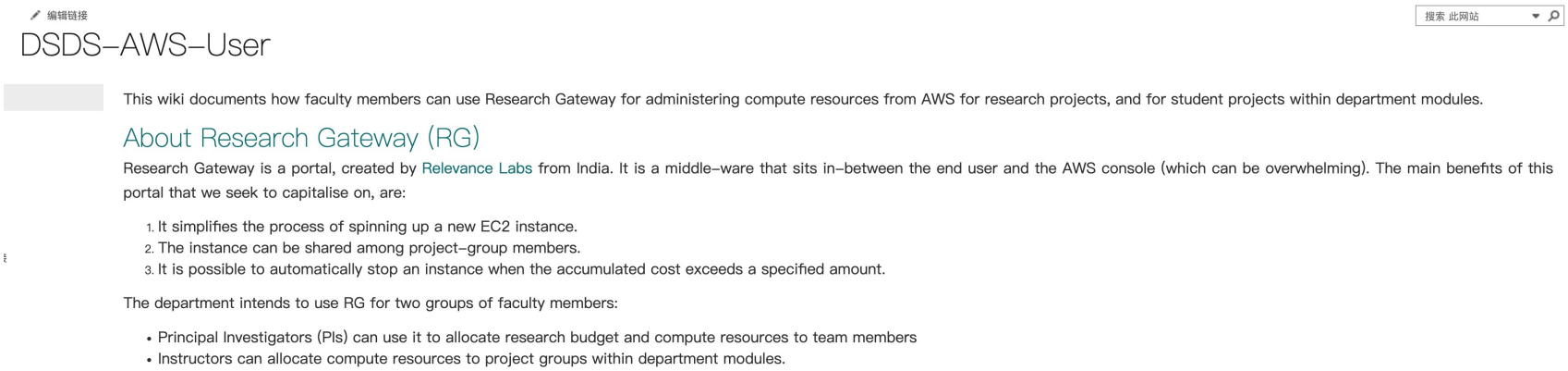
- Class participation **5%**
- Quizzes/Tests **20%**
- Assignments **30%**
- Project/Group Project **45%**: 10% proposal report; 25% final report; 10% presentation

Project/Group Project

- Topic: **Methods/Applications of NLP**
  - Fine-tuning a pre-trained language model for a downstream task
  - Evaluating performance of prompting vs. finetuning
  - Building a chatbot using large language models
  - Named Entity Recognition (NER) on medical text
  - Multimodal analysis with text and image data (e.g., CLIP)
  - Literature review of a specific NLP subfield (e.g., RAG, RLHF)
  - ...
- Team size: 1 to 4 members.
  - Larger groups are expected to undertake more extensive work.
  - You can use any language/packages

Project/Group Project

- Computing Resources: Each group may be allocated a budget of 100 USD for using an Amazon EC2 instance through *Reseach Gateway*.



编辑链接

搜索 此网站

DSDS–AWS–User

This wiki documents how faculty members can use Research Gateway for administering compute resources from AWS for research projects, and for student projects within department modules.

About Research Gateway (RG)

Research Gateway is a portal, created by Relevance Labs from India. It is a middle–ware that sits in–between the end user and the AWS console (which can be overwhelming). The main benefits of this portal that we seek to capitalise on, are:

1. It simplifies the process of spinning up a new EC2 instance.
2. The instance can be shared among project–group members.
3. It is possible to automatically stop an instance when the accumulated cost exceeds a specified amount.

The department intends to use RG for two groups of faculty members:

- Principal Investigators (PIs) can use it to allocate research budget and compute resources to team members
- Instructors can allocate compute resources to project groups within department modules.

Resources

- All class assignments can be finished by Python (using NumPy and PyTorch).
- If you need to remind yourself of Python, or you are not very familiar with NumPy:
  - Stanford Python Review Session (slides)
  - Stanford Python Review Session (code)
  - Stanford PyTorch Tutorial (code)
- Online resources
  - Natural Language Processing with Deep Learning (Stanford)
  - Stanford CS224N Video

Introduction to Natural Language Processing

What is Natural language processing (NLP)

Natural Language Processing (NLP) is a field of artificial intelligence and computational linguistics that focuses on the interaction between computers and human (natural) languages. The goal of NLP is to enable computers to understand, interpret, generate, and manipulate human language in a way that is both meaningful and useful. NLP combines elements of computer science, linguistics, machine learning, and artificial intelligence to work with both written and spoken language.

- Can you think of NLP Applications?

Neural machine translation was an early big success of Neural NLP

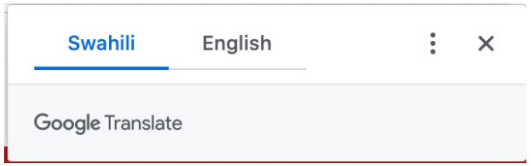
TUKO

BEST DIGITAL NEWS PLATFORM

https://kiswahili.tuko.co.ke/

Malawi yawapoteza mawaziri 2 kutokana na maafa ya COVID-19

TUKO.co.ke imefahamishwa kuwa waziri wa serikali ya mitaa Lingson Belekanyama na mwenzake wa uchukuzi Sidik Mia walifariki dunia ndani ya saa mbili tofauti.



Malawi loses 2 ministers due to COVID-19 disaster

TUKO.co.ke has been informed that local government minister Lingson Belekanyama and his transport counterpart Sidik Mia died within two separate hours.

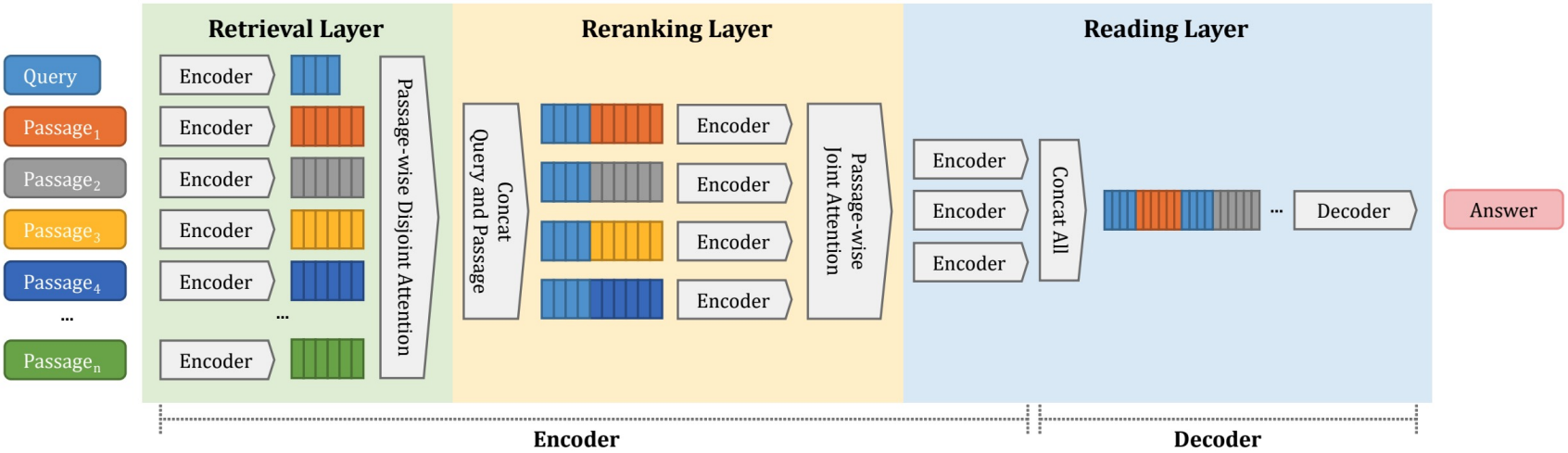
Free-text question answering: Next gen search

when did Kendrick lamar's first album come out?

July 2, 2011

These are my leftover songs you all can have them. I'm going to put my best out. My best effort. I'm trying to look for an album in 2012."[44] In June 2011, Lamar released "Ronald Reagan Era (His Evils)", a cut from *Section.80*, featuring Wu-Tang Clan leader RZA.[45] On July 2, 2011, Lamar released *Section.80*, his first independent album. The album features guest appearances from GLC, Colin Munroe, Schoolboy Q, and Ab-Soul, while the production was handled by Top Dawg in-house

E.g., YONO (Lee et al. 2021, https://arxiv.org/pdf/2112.07381.pdf) uses a T5-Large model fine-tuned for QA 3 times to run entire QA pipeline



Sentiment analysis

Twitter

V_FSD: RT @FreeMemesKids: Love my new **fidget spinner** https://t.co/IcA6Ui6fsV

MeqzEdits: @23Duckk Oh no the **fidget spinner**?

fffaraa: @ibrahimyussop Omg.. not the **fidget spinner**

boybrunch: RT @a1andar: The Born This Way album cover except Gaga is a **fidget spinner**

eve_bertie: I bought a **fidget spinner** and I think it's the best thing I've ever done

Hamza0207: I broke this little girls **fidget spinner** and I legit feel so bad ??

sentiment140.com

GPT-2 (2019): An early step towards foundation models

PROMPT (HUMAN-WRITTEN)	*A train carriage containing controlled nuclear materials was stolen in Cincinnati today. Its whereabouts are unknown.*
MODEL COMPLETION	The incident occurred on the downtown train line, which runs from Covington and Ashland stations. In an email to Ohio news outlets, the U.S. Department of Energy said it is working with the Federal Railroad Administration to find the thief. "The theft of this nuclear material will have significant negative consequences on public and environmental health, our workforce and the economy of our nation," said Tom Hicks, the U.S. Energy Secretary, in a statement. "Our top priority is to secure the theft and ensure it doesn't happen again." The stolen material was taken from the University of Cincinnati's Research Triangle Park nuclear research site, according to a news release from Department officials .

ChatGPT, GPT-4, and more

A

Hey please draft a polite mail to explain my boss Jeremy that I would not be able to come to office for next 2 days because my 9 year song Peter is angry with me that I am not giving him much time. So I have plans to take him out for a trip. I will return and complete my work by meeting deadline.

Dear Jeremy,

I hope this email finds you well. I'm writing to let you know that I will not be able to come into the office for the next two days.

My 9-year-old son Peter is feeling upset with me because he feels like I haven't been giving him enough attention lately. In order to make it up to him, I've decided to take him on a little trip to help us reconnect.

I understand that this may cause some inconvenience, but I promise to make up for any lost time by working extra hard to meet any deadlines when I return.

Thank you for your understanding.

Best regards,
[Your Name]

User What is unusual about this image?

Source: Barnorama

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Summary

- Core NLP tasks
  - Tokenisation, Segmentation
  - Language Modelling
  - Machine Translation
  - Syntactic and Semantic Parsing
  - Document Classification
  - Information Extraction
  - Question Answering
  - Sentiment analysis

How do we have usable meaning of words in a computer?

How do we have usable meaning of words in a computer?

- Definition: meaning (Webster dictionary)
  - the idea that is represented by a word, phrase, etc.
  - the idea that a person wants to express by using words, signs, etc.
  - the idea that is expressed in a work of writing, art, etc.
- Challenges:
  - **Ambiguity:** Words can have multiple meanings based on context.
  - **Complex Grammar:** Human languages have complex and varied grammatical rules.
  - **Context Understanding:** Understanding implied meanings and sarcasm.
- Previously commonest NLP solution: Use, e.g., WordNet, a thesaurus containing lists of synonym sets and hypernyms ("is a" relationships)

One-hot vector representation

- In traditional NLP, we regard words as discrete symbols:
  - $\{\text{hotel}, \text{conference}, \text{motel}\}$ - a localist representation
  - Such symbols for words can be represented by one-hot vectors:

$$\text{hotel} = [1, 0, 0]$$
$$\text{conference} = [0, 1, 0]$$
$$\text{motel} = [0, 0, 1]$$

  - Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But

$$\text{hotel} = [1, 0, 0]$$
$$\text{motel} = [0, 0, 1]$$

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But

$$\text{hotel} = [1, 0, 0]$$
$$\text{motel} = [0, 0, 1]$$

- These two vectors are orthogonal
- There is no natural notion of similarity for one-hot vectors!

Representing words by their context

- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by
  - "You shall know a word by the company it keeps" (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of w to build up a representation of w.

Representing words by their context

- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by
  - "You shall know a word by the company it keeps" (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- We use the many contexts of w to build up a representation of w.

...advancements in statistics have led to improved methods for analyzing...
...statistics is crucial for making data-driven decisions in healthcare...
...students pursuing a degree in statistics often learn both theory and application...
...the role of statistics in machine learning is foundational...

- These context words will represent statistics.

Word vectors

We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product.

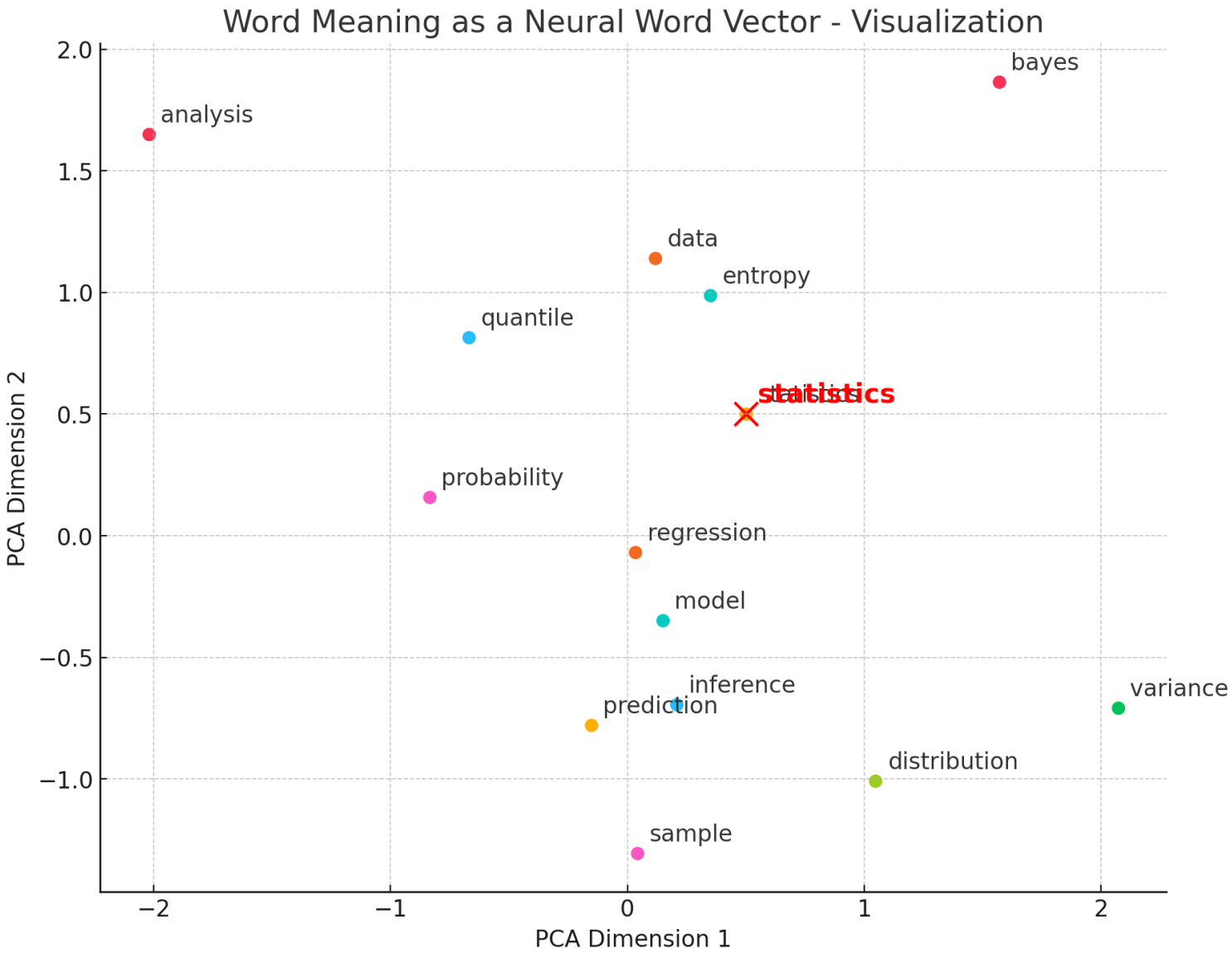
$$\mathbf{v}_{\text{statistics}} = (0.124, 0.453, 0.156, -0.454, -0.859)^\top$$
$$\mathbf{v}_{\text{math}} = (0.104, 0.493, 0.126, -0.404, -0.409)^\top$$

- Cosine similarity of two word vectors:

$$\cos(\mathbf{v}_w, \mathbf{v}_c) = \frac{\mathbf{v}_w^\top \mathbf{v}_c}{\|\mathbf{v}_w\|_2 \|\mathbf{v}_c\|_2}$$

- Note: word vectors are also called (word) embeddings or (neural) word representations. They are a distributed representation.

Word meaning as a neural word vector – visualization



Word2vec: Overview

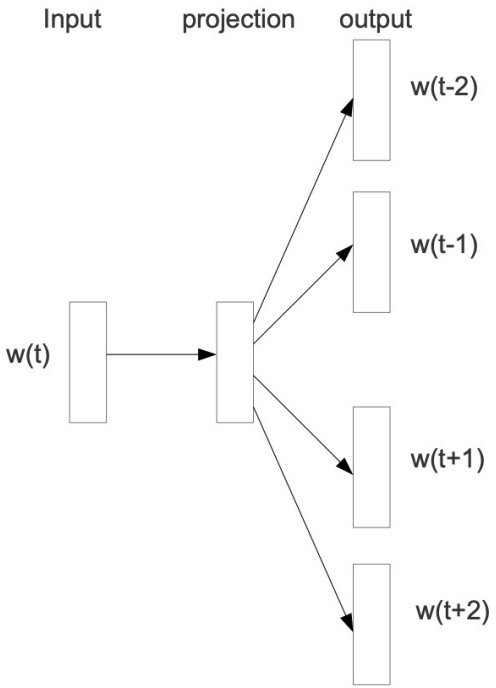


Figure: Skip-gram model ((Mikolov et al., 2013a))

Word2vec is a framework for learning word vectors ((Mikolov et al., 2013a,b))

Idea:

- We have a large corpus ("body") of text: a long list of words

Word2vec: Overview

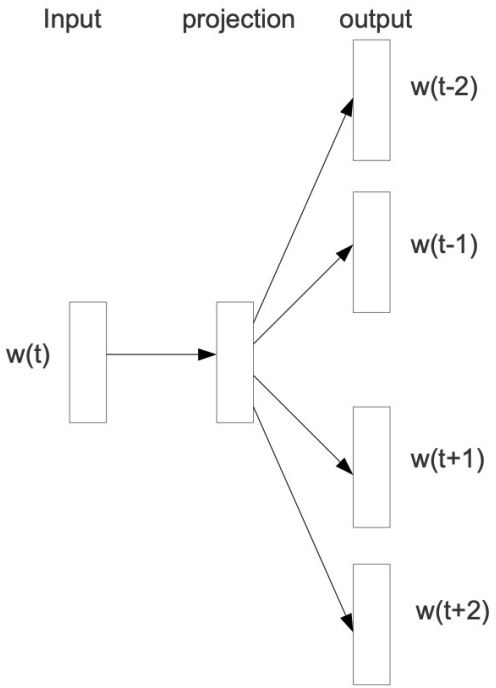


Figure: Skip-gram model ((Mikolov et al., 2013a))

Word2vec is a framework for learning word vectors ((Mikolov et al., 2013a,b))

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector

Word2vec: Overview

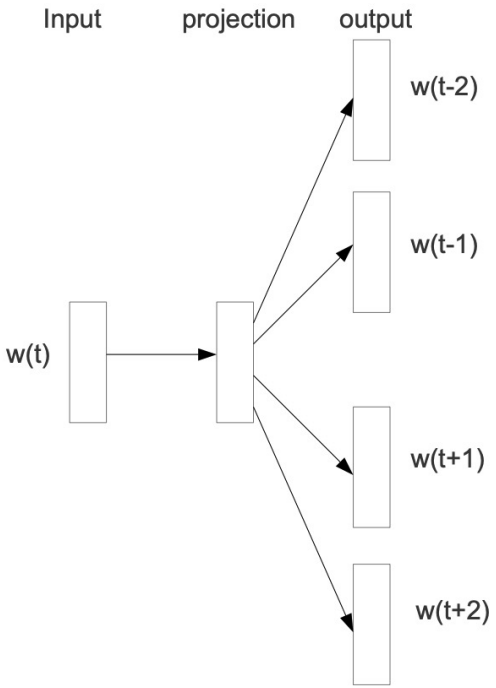


Figure: Skip-gram model ((Mikolov et al., 2013a))

Word2vec is a framework for learning word vectors ((Mikolov et al., 2013a,b))

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o

Word2vec: Overview

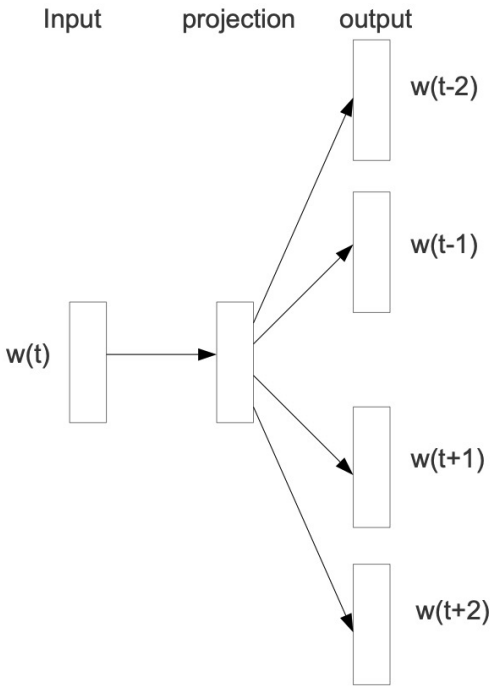


Figure: Skip-gram model ((Mikolov et al., 2013a))

Word2vec is a framework for learning word vectors ((Mikolov et al., 2013a,b))

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)

Word2vec: Overview

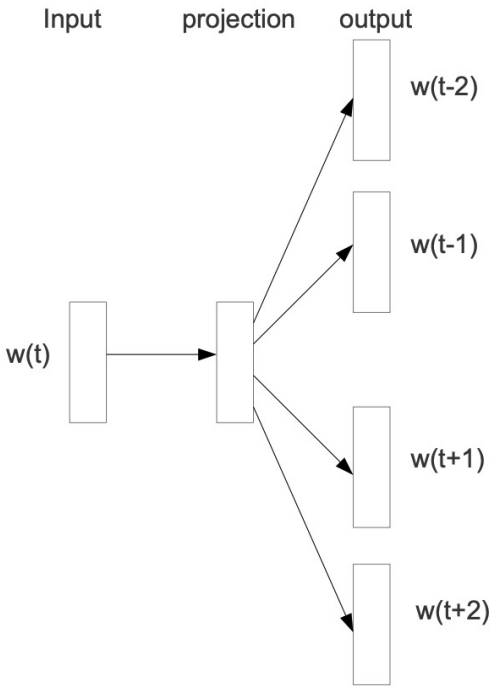


Figure: Skip-gram model ((Mikolov et al., 2013a))

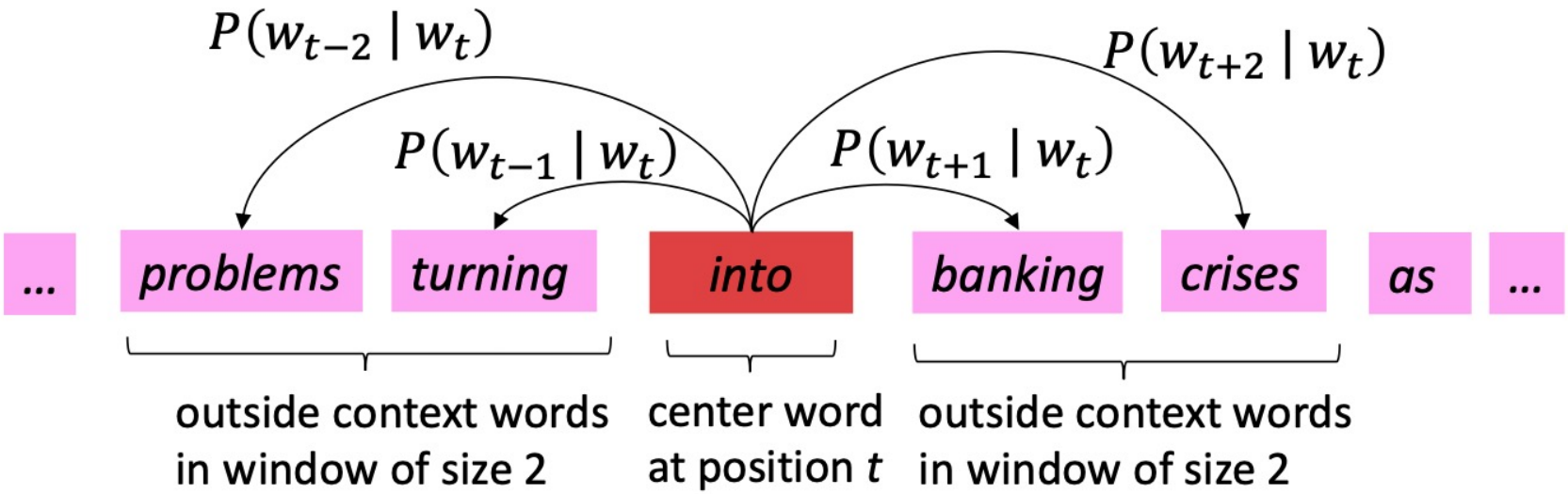
Word2vec is a framework for learning word vectors ((Mikolov et al., 2013a,b))

Idea:

- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

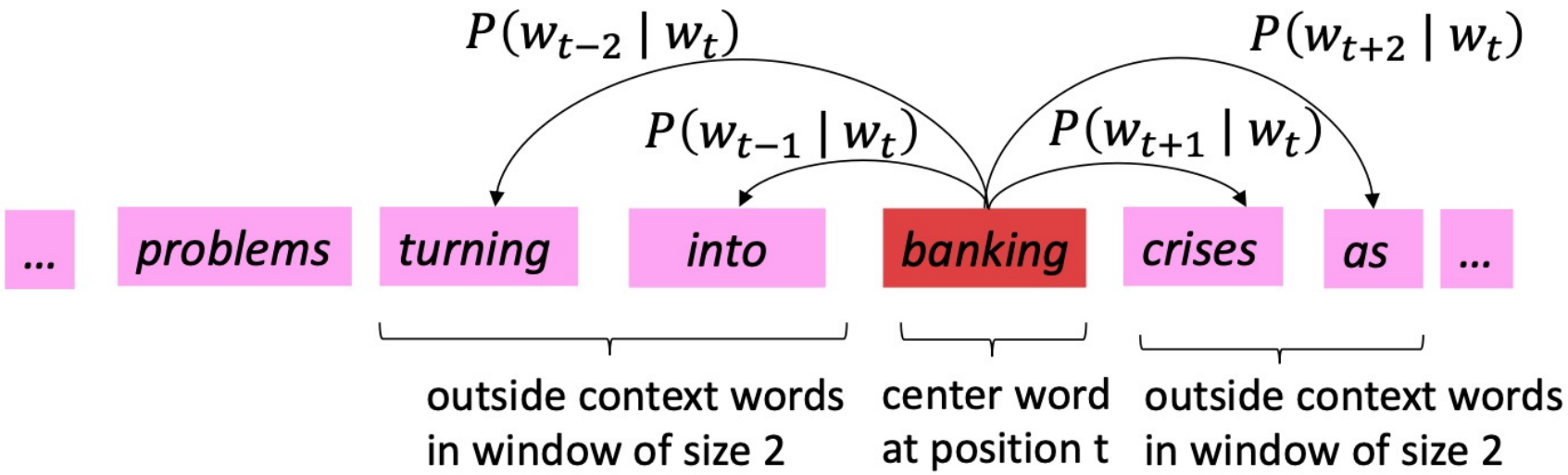
Word2vec: Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: Overview

Example windows and process for computing $P(w_{t+j} \mid w_t)$



Word2vec: objective function

For each position $t = 1, \ldots, T$, predict context words within a window of fixed size m, given center word w_t.

Likelihood

$$L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \neq 0} P(w_{t+j} \mid w_t; \theta)$$

- θ is all variables to be optimized: the word embeddings

The objective function $J(\theta)$ is the (average) negative log likelihood (sometimes called a cost or loss function):

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two vectors per word* w:
  - v_w, when w is a center word
  - u_w, when w is a context word

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0} \log P(w_{t+j} \mid w_t; \theta)$$

- **Question:** How to calculate $P(w_{t+j} \mid w_t; \theta)$?
- **Answer:** We will *use two vectors per word* w:
  - v_w, when w is a center word
  - u_w, when w is a context word
- These word vectors are subparts of the big vector of all parameters θ
- Then for a center word c and a context word o:

$$P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V} \exp(u_w^T v_c)}$$

Word2vec: prediction function

$$\text{Probability Calculation:} \quad P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

1. Dot product compares similarity of o and c: $u^T v = \sum_{i=1}^{n} u_i v_i$
   Larger dot product = larger probability
2. Exponentiation makes anything positive
3. Normalize over entire vocabulary to give probability distribution

Word2vec: prediction function

$$\text{Probability Calculation:} \quad P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

1. Dot product compares similarity of o and c: $u^T v = \sum_{i=1}^{n} u_i v_i$
   Larger dot product = larger probability
2. Exponentiation makes anything positive
3. Normalize over entire vocabulary to give probability distribution

This is an example of the softmax function $\mathbb{R}^n \to (0,1)^n$:

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

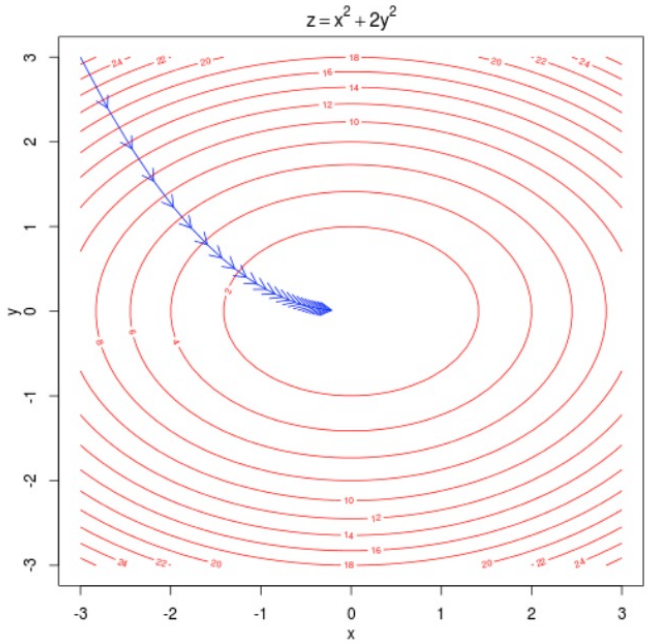
- The softmax function maps arbitrary values x_i to a probability distribution p_i
- **"max"** because it amplifies the probability of the largest x_i
- **"soft"** because it still assigns some probability to smaller x_i
- Frequently used in Deep Learning. Sort of a weird name because it returns a distribution!

To train the model: optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- **Recall:** θ represents **all** the model parameters, in one long vector
- In our case, with d-dimensional vectors and V-many words, we have $\Rightarrow$
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{\text{aardvark}} \\ v_a \\ \vdots \\ v_{\text{zebra}} \\ u_{\text{aardvark}} \\ u_a \\ \vdots \\ u_{\text{zebra}} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

Stochastic gradient descent

- **Problem:** $J(\theta)$ is a function of **all** windows in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is **very** expensive to compute
- You would wait a very long time before making a single update!
- **Very** bad idea for pretty much all neural nets!
- **Solution:** Stochastic gradient descent (SGD)
  - Repeatedly sample windows, and update after each one
- Algorithm:

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

Word2vec algorithm family: more details

- **Why two vectors?** $\rightarrow$ Easier optimization. Average both at the end
  - But can implement the algorithm with just one vector per word... and it helps a bit
- **Two model variants:**
  1. Skip-grams (SG)
     Predict context ("outside") words (position independent) given center word
  2. Continuous Bag of Words (CBOW)
     Predict center word from (bag of) context words

Word2vec algorithm family: more details

- **Why two vectors?** → Easier optimization. Average both at the end
  - But can implement the algorithm with just one vector per word... and it helps a bit
- **Two model variants:**
  1. Skip-grams (SG)
     Predict context ("outside") words (position independent) given center word
  2. Continuous Bag of Words (CBOW)
     Predict center word from (bag of) context words
- We presented: Skip-gram model
- **Loss functions for training:**
  1. Naïve softmax (simple but expensive loss function, when many output classes)
  2. More optimized variants like hierarchical softmax
  3. Negative sampling
- So far, we explained naïve softmax

Skip-gram Model with Negative Sampling

- The normalization term is computationally expensive (when many output classes):

$$P(o \mid c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)} \quad \text{[A big sum over many words]}$$

- Hence, in standard Word2vec, you implement the skip-gram model with negative sampling
- **Idea:** train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

Skip-gram model with negative sampling

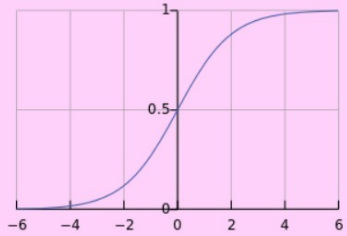
- We take K negative samples (using word probabilities*)
- Maximize probability of real outside word; minimize probability of random words
- Using notation consistent with this class, we minimize (Sigmoid rather than softmax):

$$J_{\text{neg-sample}}(u_o, v_c, U) = -\log \sigma(u_o^T v_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-u_k^T v_c)$$

- The logistic/sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- *Sample with $P(w) = U(w)^{3/4}/Z$, the unigram distribution $U(w)$ raised to the $3/4$ power
  - The power makes less frequent words be sampled a bit more often

Stochastic gradients with negative sampling

- We iteratively take gradients at each window for SGD
- In each window, we only have at most $2m+1$ words plus $2km$ negative words with negative sampling, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla v_{\text{like}} \\ \vdots \\ \nabla u_l \\ \vdots \\ \nabla u_{\text{learning}} \end{bmatrix} \in \mathbb{R}^{2dV}$$

- We might only update the word vectors that actually appear!

Python code

I'm teaching word2vec to students with little python background. Could you generate some simple data and code for me to illustrate the usage of the word2vec?

Here's a simple example to demonstrate Word2Vec to students with little Python background. This code uses the `gensim` library, which is beginner-friendly for working with Word2Vec.

Example Task: Analyze a Small Dataset of Sentences

Step 1: Install Necessary Library

First, ensure your students have `gensim` installed. Use:

bash 复制代码

```
pip install gensim
```

Python code

```
In [1]: from gensim.models import Word2Vec
   ...: from gensim.models.word2vec import LineSentence

In [2]: sentences = [
   ...:     ["I", "love", "natural", "language", "processing"],
   ...:     ["Word2Vec", "is", "a", "popular", "model", "for", "embeddings"],
   ...:     ["We", "can", "learn", "word", "relationships"],
   ...:     ["Word", "embeddings", "capture", "semantic", "meanings"],
   ...:     ["I", "enjoy", "teaching", "Word2Vec", "to", "students"]
   ...: ]

In [3]: model = Word2Vec(sentences, vector_size=10, window=3, min_count=1, sg=1)

In [4]: # Step 3: Explore the Word2Vec model

In [5]: print("Vocabulary in the model:")
Vocabulary in the model:

In [6]: print(list(model.wv.index_to_key))
['I', 'embeddings', 'Word2Vec', 'love', 'natural', 'language', 'processing', 'is', 'a', 'popular', 'model', 'for', 'students', 'to', 'can',
'learn', 'word', 'relationships', 'Word', 'capture', 'semantic', 'meanings', 'enjoy', 'teaching', 'We']

In [7]: # Step 4: Get the vector representation of a word

In [8]: print("\nVector representation of the word 'Word2Vec':")

Vector representation of the word 'Word2Vec':

In [9]: print(model.wv['Word2Vec'])
[ 0.07311766  0.05070262  0.06757693  0.00762866  0.06350891 -0.03405366
 -0.00946401  0.05768573 -0.07521638 -0.03936104]

In [10]: # Step 5: Find most similar words

In [11]: print("\nWords most similar to 'language':")

Words most similar to 'language':

In [12]: print(model.wv.most_similar("language"))
[('word', 0.5916882753372192), ('enjoy', 0.5116440057754517), ('We', 0.4622342586517334), ('semantic', 0.38148820400238037), ('Word',
0.30411970615386963), ('popular', 0.2914133369922638), ('a', 0.27560877799987793), ('natural', 0.24965371191501617), ('model',
0.22384946048259735), ('capture', 0.17258583009243011)]
```

Training Word2Vec on Movie Reviews

```
In [4]:
   ...: nltk.download('movie_reviews')
   ...: sentences = [list(movie_reviews.words(fileid)) for fileid in movie_reviews.fileids()]
[nltk_data] Downloading package movie_reviews to
[nltk_data]     /Users/doudou/nltk_data...
[nltk_data]   Package movie_reviews is already up-to-date!

In [5]:
   ...: model = Word2Vec(sentences, vector_size=50, window=5, min_count=5, sg=1)
   ...:
   ...: # Step 3: Explore the trained model
   ...: print("Vocabulary in the model:")
   ...: print(list(model.wv.index_to_key)[:10])  # Display top 10 words in the vocabulary
Vocabulary in the model:
[',', 'the', '.', 'a', 'and', 'of', 'to', "'", 'is', 'in']

In [6]:
   ...: print("\nWords most similar to 'good':")
   ...: print(model.wv.most_similar("good"))

Words most similar to 'good':
[('decent', 0.8692122101783752), ('great', 0.8277547955513), ('bad', 0.8179364800453186), ('wonderful', 0.7874459028244019), ('terrible',
0.7806883454322815), ('nice', 0.7732791900634766), ('perfect', 0.766955554485321), ('fantastic', 0.7631729245185852), ('passable',
0.757935106754303), ('lousy', 0.7437241077423096)]
```

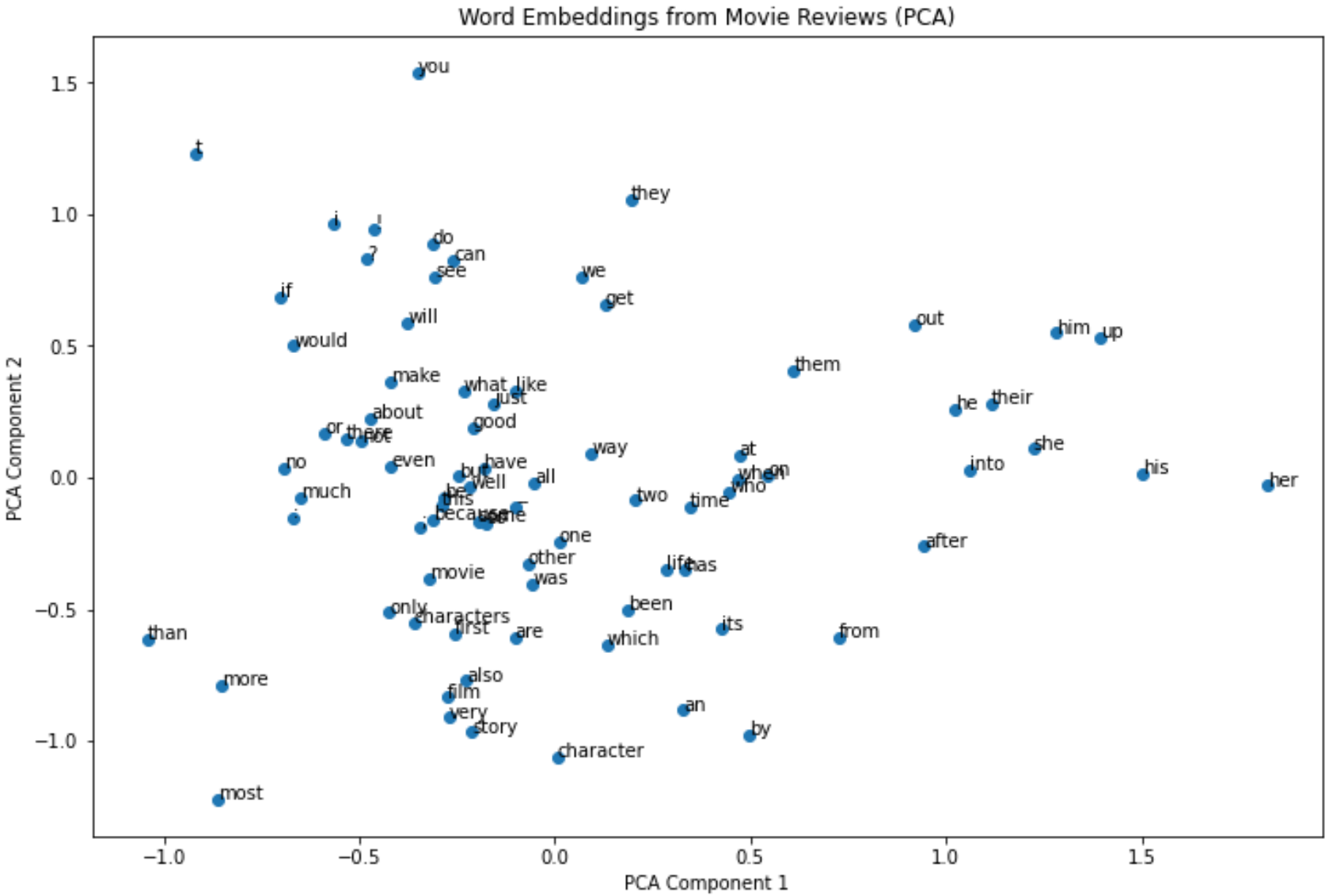
Training Word2Vec on Movie Reviews

- Visualization by PCA

```
In [7]:
   ...: from sklearn.decomposition import PCA
   ...: import matplotlib.pyplot as plt
   ...:
   ...: # Get the embeddings and corresponding words
   ...: words = list(model.wv.index_to_key[20:100])  # Limit to 50 words for clarity
   ...: embeddings = model.wv[words]
   ...:
   ...: # Use PCA to reduce dimensionality
   ...: pca = PCA(n_components=2)
   ...: reduced_embeddings = pca.fit_transform(embeddings)
   ...:
   ...: # Plot the embeddings
   ...: plt.figure(figsize=(12, 8))
   ...: plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1], marker='o')
   ...:
   ...: for i, word in enumerate(words):
   ...:     plt.annotate(word, xy=(reduced_embeddings[i, 0], reduced_embeddings[i, 1]), fontsize=10)
   ...:
   ...: plt.title("Word Embeddings from Movie Reviews (PCA)")
   ...: plt.xlabel("PCA Component 1")
   ...: plt.ylabel("PCA Component 2")
   ...: plt.show()
```

Training Word2Vec on Movie Reviews

- Visualization by PCA



Why not capture co-occurrence counts directly?

There's something weird about iterating through the whole corpus (perhaps many times). Why don't we just accumulate all the statistics of what words appear near each other?

Why not capture co-occurrence counts directly?

There's something weird about iterating through the whole corpus (perhaps many times). Why don't we just accumulate all the statistics of what words appear near each other?

Building a co-occurrence matrix X:

- 2 options: windows vs. full document
- Window: Similar to word2vec, use a window around each word $\rightarrow$ captures some syntactic and semantic information ("word space")
- Word-document co-occurrence: Co-occurrence matrix will give general topics (all sports terms will have similar entries), leading to "Latent Semantic Analysi" ("document space")

Example: window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)

Example corpus:

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

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

In-class practice: window-based co-occurrence matrix

- **Context window size:** 2 (words within a distance of 2 are considered co-occurring)
- **Symmetric window:** both left and right context are used

Example corpus:

- cats chase mice.
- dogs chase cats.
- mice eat cheese.
- cheese scares dogs.

Task: Construct the co-occurrence matrix using the above corpus and a window size of 2.

Answer: window-based co-occurrence matrix (window = 2)

Example corpus:

- cats chase mice
- dogs chase cats
- mice eat cheese
- cheese scares dogs

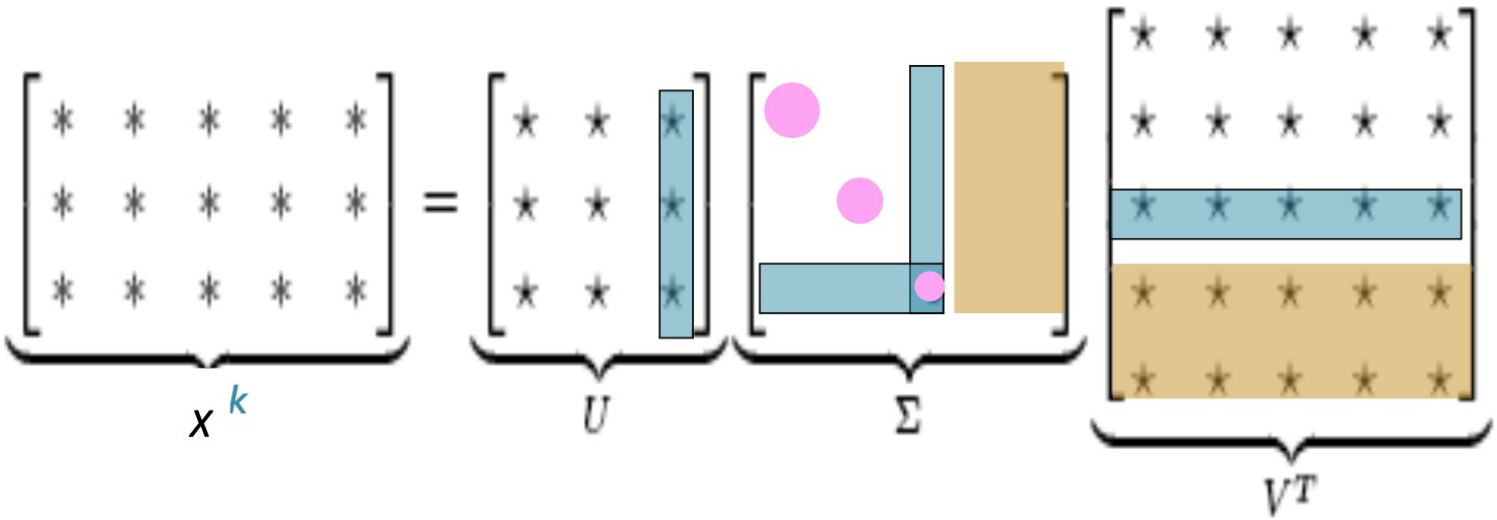
counts	cats	chase	mice	dogs	eat	cheese	scares
cats	0	2	1	1	0	0	0
chase	2	0	1	1	0	0	0
mice	1	1	0	0	1	1	0
dogs	1	1	0	0	0	1	1
eat	0	0	1	0	0	1	0
cheese	0	0	1	1	1	0	1
scares	0	0	0	1	0	1	0

Co-occurrence Vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues $\rightarrow$ Models are less robust
- Low-dimensional vectors
  - Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
  - Usually 25–1000 dimensions, similar to word2vec
  - How to reduce the dimensionality?

Classic method: dimensionality reduction

- Singular Value Decomposition of co-occurrence matrix $\mathbf{X}$:
- Factorizes $\mathbf{X}$ into $\mathbf{U\Sigma V}^T$, where U and V are orthonormal (unit vectors and orthogonal)

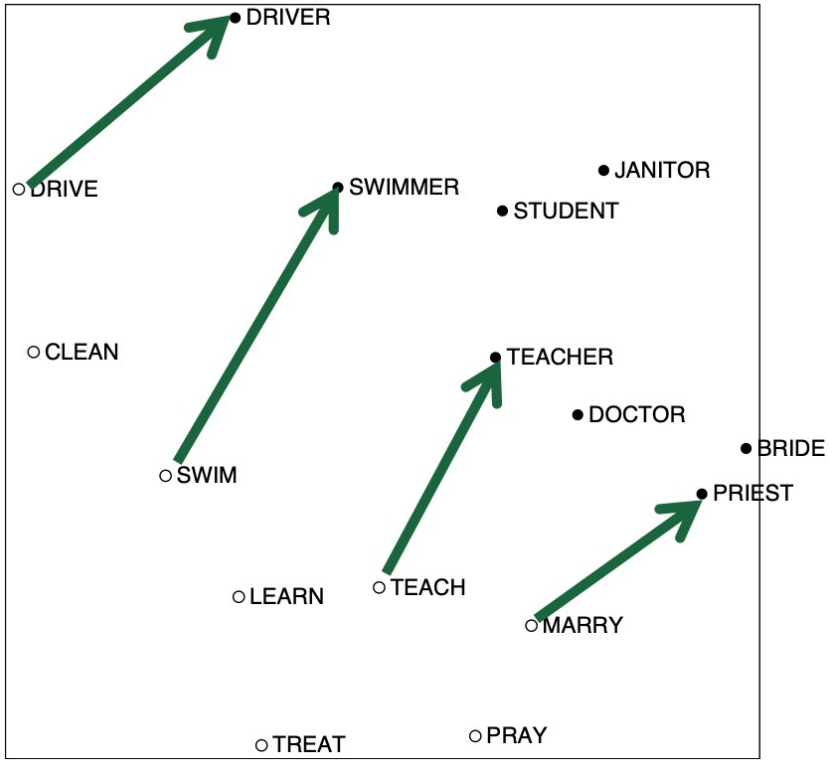


- Retain only k singular values, in order to generalize.
- $\widehat{\mathbf{X}}$ is the best rank k approximation to $\mathbf{X}$, in terms of least squares.
- Classic linear algebra result. Expensive to compute for large matrices.

Hacks to X (several used in Rohde et al. 2005 in COALS)

- Running an SVD on raw counts doesn't work well!!!
- Scaling the counts in the cells can help **a lot**
  - Problem: function words (the, he, has) are too frequent $\rightarrow$ syntax has too much impact. Some fixes:
    - log the frequencies
    - min(X, t), with $t \approx 100$
    - Ignore the function words
- Ramped windows that count closer words more than further away words
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Interesting semantic patterns emerge in the scaled vectors



- A meaning component (doer of event) becomes a linear meaning component in the space!
- This is the COALS model from Rohde et al. ms., 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

Neural word embedding as implicit matrix factorization

- Levy and Goldberg ((2014)) analyzed skip-gram with negative-sampling (SGNS), and showed that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant.

Neural word embedding as implicit matrix factorization

- Levy and Goldberg ((2014)) analyzed skip-gram with negative-sampling (SGNS), and showed that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant.
- Pointwise mutual information is an information-theoretic association measure between a pair of discrete outcomes x and y, defined as

$$\mathrm{PMI}(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

- $\mathrm{PMI}(x, y)$ measures the association between a word w and a context c:

$$\mathrm{PMI}(w, c) = \log \frac{\sharp(w, c) \cdot |D|}{\sharp(w)\sharp(c)}$$

Neural word embedding as implicit matrix factorization

- Levy and Goldberg ((2014)) analyzed skip-gram with negative-sampling (SGNS), and showed that it is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant.
- Pointwise mutual information is an information-theoretic association measure between a pair of discrete outcomes x and y, defined as

$$\mathrm{PMI}(x, y) = \log \frac{P(x, y)}{P(x)P(y)}$$

- $\mathrm{PMI}(x, y)$ measures the association between a word w and a context c:

$$\mathrm{PMI}(w, c) = \log \frac{\sharp(w, c) \cdot |D|}{\sharp(w)\sharp(c)}$$

- The use of PMI as a measure of association in NLP was introduced by Church and Hanks ((1990)) and widely adopted for word similarity tasks ((Dagan et al., 1994; Turney, 2001; Turney and Pantel, 2010))

Positive PMI and shifted positive PMI

- A sparse and consistent alternative from the NLP literature is to use the positive PMI (PPMI) metric:

$$\mathrm{PPMI}(w, c) = \max\big(\mathrm{PMI}(w, c), 0\big)$$

- Shifted PPMI (SPPMI), a novel association metric which, to the best of our knowledge, was not explored in the NLP and word similarity communities:

$$\mathrm{SPPMI}_k(w, c) = \max\big(\mathrm{PMI}(w, c) - \log k, 0\big)$$

- Low-rank approximation:

$$\mathrm{SPPMI}_k \approx \mathbf{U}\mathbf{U}^\top$$

- The so-called SPPMI-SVD algorithm.
- Generative model for PMI-based embeddings ((Arora et al., 2016))

Performance

Method	PMI$-\log k$	SPPMI	SVD			SGNS		
			$d = 100$	$d = 500$	$d = 1000$	$d = 100$	$d = 500$	$d = 1000$
$k = 1$	0%	0.00009%	26.1%	25.2%	24.2%	31.4%	29.4%	7.40%
$k = 5$	0%	0.00004%	95.8%	95.1%	94.9%	39.3%	36.0%	7.13%
$k = 15$	0%	0.00002%	266%	266%	265%	7.80%	6.37%	5.97%

Table 1: Percentage of deviation from the optimal objective value (lower values are better). See 5.1 for details.

WS353 (WordSim) [13]			MEN (WordSim) [4]			Mixed Analogies [20]			Synt. Analogies [22]		
Representation		Corr.	Representation		Corr.	Representation		Acc.	Representation		Acc.
SVD	(k=5)	0.691	SVD	(k=1)	0.735	SPPMI	(k=1)	0.655	SGNS	(k=15)	0.627
SPPMI	(k=15)	0.687	SVD	(k=5)	0.734	SPPMI	(k=5)	0.644	SGNS	(k=5)	0.619
SPPMI	(k=5)	0.670	SPPMI	(k=5)	0.721	SGNS	(k=15)	0.619	SGNS	(k=1)	0.59
SGNS	(k=15)	0.666	SPPMI	(k=15)	0.719	SGNS	(k=5)	0.616	SPPMI	(k=5)	0.466
SVD	(k=15)	0.661	SGNS	(k=15)	0.716	SPPMI	(k=15)	0.571	SVD	(k=1)	0.448
SVD	(k=1)	0.652	SGNS	(k=5)	0.708	SVD	(k=1)	0.567	SPPMI	(k=1)	0.445
SGNS	(k=5)	0.644	SVD	(k=15)	0.694	SGNS	(k=1)	0.540	SPPMI	(k=15)	0.353
SGNS	(k=1)	0.633	SGNS	(k=1)	0.690	SVD	(k=5)	0.472	SVD	(k=5)	0.337
SPPMI	(k=1)	0.605	SPPMI	(k=1)	0.688	SVD	(k=15)	0.341	SVD	(k=15)	0.208

Table 2: A comparison of word representations on various linguistic tasks. The different representations were created by three algorithms (SPPMI, SVD, SGNS) with $d = 1000$ and different values of k.

Encoding meaning components in vector differences

- **Glove: Global vectors for word representation** proposed by Pennington et al. ((2014)).
- **Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components
- We want to capture them as linear meaning components in a word vector space!

	$x =$ solid	$x =$ gas	$x =$ water	$x =$ random
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~ 1	small

Encoding meaning components in vector differences

- **Glove: Global vectors for word representation** proposed by Pennington et al. ((2014)).
- **Crucial insight:** Ratios of co-occurrence probabilities can encode meaning components
- We want to capture them as linear meaning components in a word vector space!

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{fashion}$
$P(x \mid \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x \mid \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

GloVe

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?
A: Log-bilinear model:

$$w_i \cdot w_j = \log P(i \mid j)$$

With vector differences:

$$w_x \cdot (w_a - w_b) = \log \frac{P(x \mid a)}{P(x \mid b)}$$

Loss:

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \left(w_i^T \widetilde{w}_j + b_i + \widetilde{b}_j - \log X_{ij} \right)^2$$

- Fast training
- Scalable to huge corpora

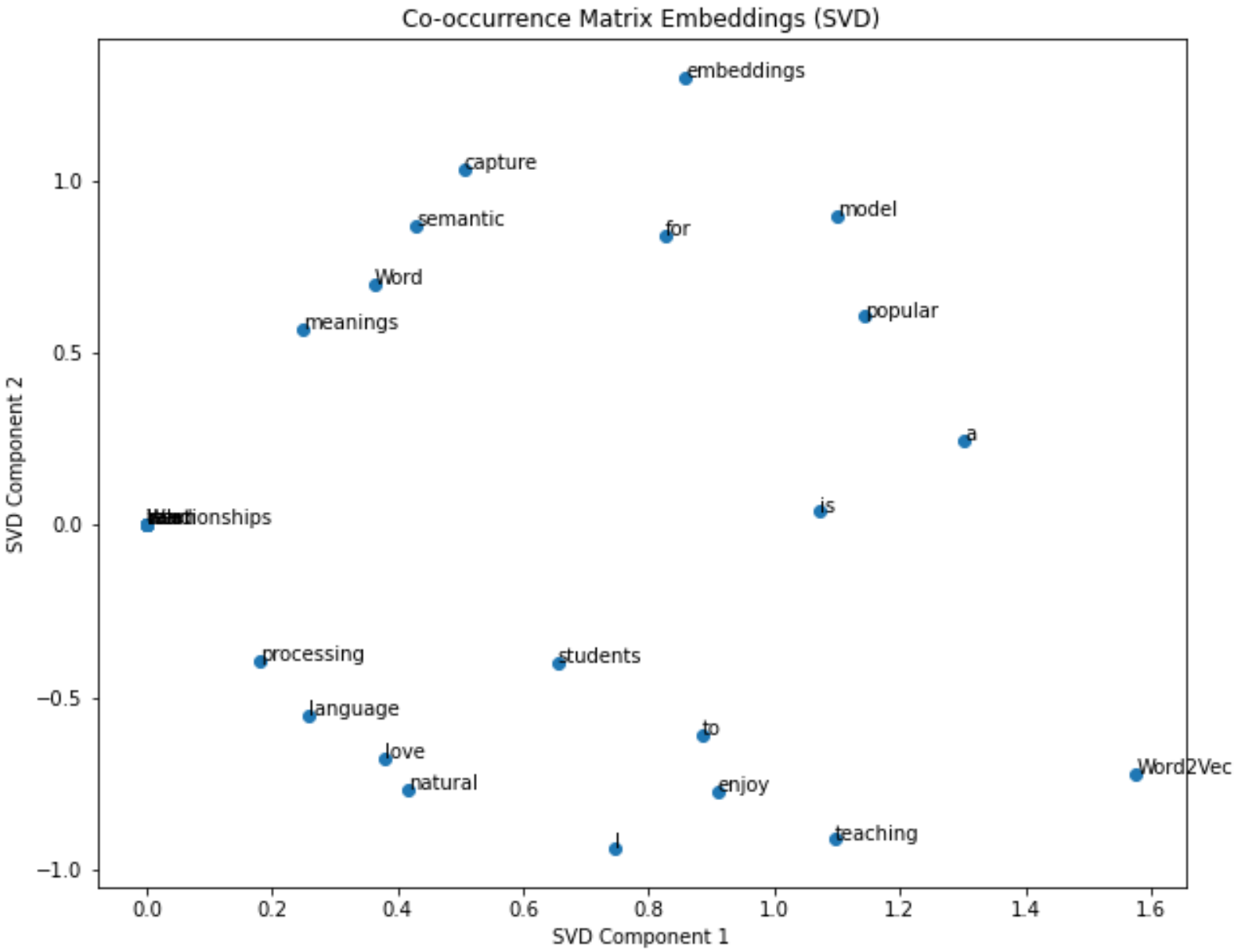
Python code (Co-occurrence-based embedding)

```
In [1]:
   ...: import numpy as np
   ...: import pandas as pd
   ...: from sklearn.decomposition import TruncatedSVD
   ...: import matplotlib.pyplot as plt
   ...: from collections import Counter
   ...: from itertools import combinations
   ...:
   ...: # Example dataset
   ...: sentences = [
   ...:     ["I", "love", "natural", "language", "processing"],
   ...:     ["Word2Vec", "is", "a", "popular", "model", "for", "embeddings"],
   ...:     ["We", "can", "learn", "word", "relationships"],
   ...:     ["Word", "embeddings", "capture", "semantic", "meanings"],
   ...:     ["I", "enjoy", "teaching", "Word2Vec", "to", "students"]
   ...: ]
   ...:
   ...: # Step 1: Build the vocabulary
   ...: vocab = {word: idx for idx, word in enumerate(set(word for sentence in sentences for word in sentence))}
   ...: print(vocab)
   ...:
   ...: # Step 2: Build the co-occurrence matrix
   ...: vocab_size = len(vocab)
   ...: print(vocab_size)
   ...:
   ...: co_matrix = np.zeros((vocab_size, vocab_size), dtype=np.float32)
{'popular': 0, 'Word': 1, 'learn': 2, 'Word2Vec': 3, 'love': 4, 'a': 5, 'can': 6, 'word': 7, 'enjoy': 8, 'processing': 9, 'is': 10, 'language':
11, 'relationships': 12, 'semantic': 13, 'for': 14, 'students': 15, 'capture': 16, 'We': 17, 'to': 18, 'model': 19, 'meanings': 20, 'I': 21,
'natural': 22, 'teaching': 23, 'embeddings': 24}
25
```

Python code (Co-occurrence-based embedding)

```
In [2]:
   ...: window_size = 2  # Only consider words within 2 positions on either side
   ...:
   ...: # Step 4: Populate the co-occurrence matrix with the window
   ...: for sentence in sentences:
   ...:     sentence_length = len(sentence)
   ...:     for idx, word in enumerate(sentence):
   ...:         word_idx = vocab[word]
   ...:
   ...:         # Define the context window
   ...:         start = max(0, idx - window_size)
   ...:         end = min(sentence_length, idx + window_size + 1)
   ...:
   ...:         # Update co-occurrence counts for words in the window
   ...:         for context_idx in range(start, end):
   ...:             if idx != context_idx:  # Skip the word itself
   ...:                 context_word_idx = vocab[sentence[context_idx]]
   ...:                 co_matrix[word_idx, context_word_idx] += 1
   ...:
   ...: # Step 3: Reduce dimensionality using SVD
   ...: svd = TruncatedSVD(n_components=2)
   ...: reduced_embeddings = svd.fit_transform(co_matrix)
   ...:
   ...: # Step 4: Visualize
   ...: words = list(vocab.keys())
   ...: plt.figure(figsize=(10, 8))
   ...: plt.scatter(reduced_embeddings[:, 0], reduced_embeddings[:, 1])
   ...: for i, word in enumerate(words):
   ...:     plt.annotate(word, (reduced_embeddings[i, 0], reduced_embeddings[i, 1]))
   ...: plt.title("Co-occurrence Matrix Embeddings (SVD)")
   ...: plt.xlabel("SVD Component 1")
   ...: plt.ylabel("SVD Component 2")
   ...: plt.show()
```

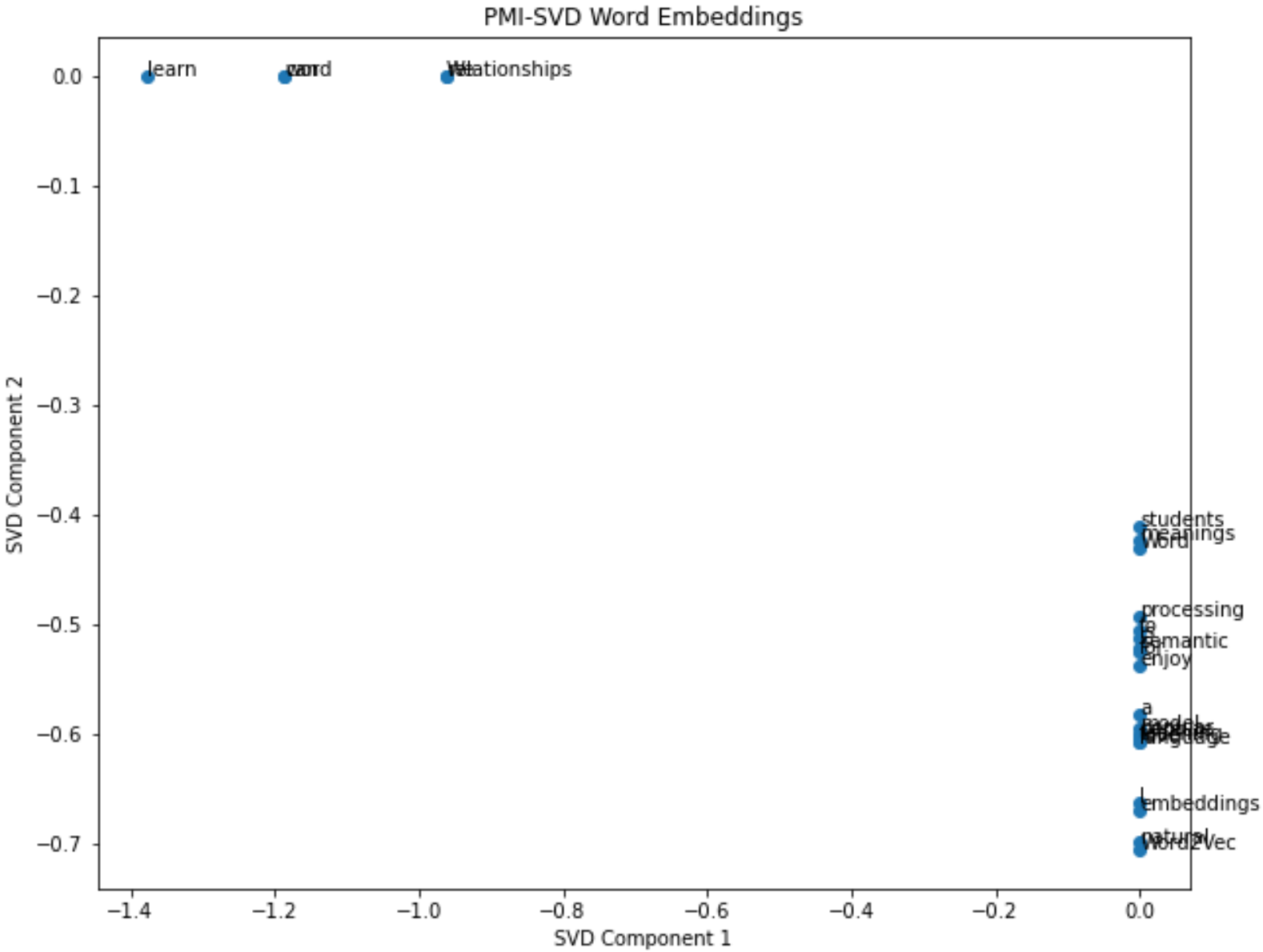
Python code (Co-occurrence-based embedding)



Python code (PMI-SVD embedding)

```
In [3]:
   ...:
   ...: co_occurrence_sum = np.sum(co_matrix)
   ...: p_word = np.sum(co_matrix, axis=1) / co_occurrence_sum
   ...: pmi_matrix = np.zeros_like(co_matrix)
   ...:
   ...: for i in range(vocab_size):
   ...:     for j in range(vocab_size):
   ...:         if co_matrix[i, j] > 0:
   ...:             pmi = np.log((co_matrix[i, j] / co_occurrence_sum) / (p_word[i] * p_word[j]))
   ...:             pmi_matrix[i, j] = max(pmi, 0)  # Positive PMI
   ...:
   ...: # Step 2: Apply SVD on the PMI matrix
   ...: import numpy as np
   ...: U, Sigma, Vt = np.linalg.svd(pmi_matrix, full_matrices=False)
   ...:
   ...: # Extract the top-k components
   ...: k = 2  # Number of dimensions
   ...: U_k = U[:, :k]                # First k columns of U
   ...: Sigma_k = np.diag(Sigma[:k])  # Top k singular values as a diagonal matrix
   ...: V_k = Vt[:k, :]               # First k rows of V^T
   ...:
   ...: # Compute U_k Sigma_k^{1/2}
   ...: Sigma_k_sqrt = np.sqrt(Sigma_k)
   ...: pmi_embeddings = U_k @ Sigma_k_sqrt
   ...:
   ...:
   ...: # Step 3: Visualize
   ...: plt.figure(figsize=(10, 8))
   ...: plt.scatter(pmi_embeddings[:, 0], pmi_embeddings[:, 1])
   ...: for i, word in enumerate(words):
   ...:     plt.annotate(word, (pmi_embeddings[i, 0], pmi_embeddings[i, 1]))
   ...: plt.title("PMI-SVD Word Embeddings")
   ...: plt.xlabel("SVD Component 1")
   ...: plt.ylabel("SVD Component 2")
   ...: plt.show()
```

Python code (PMI-SVD embedding)



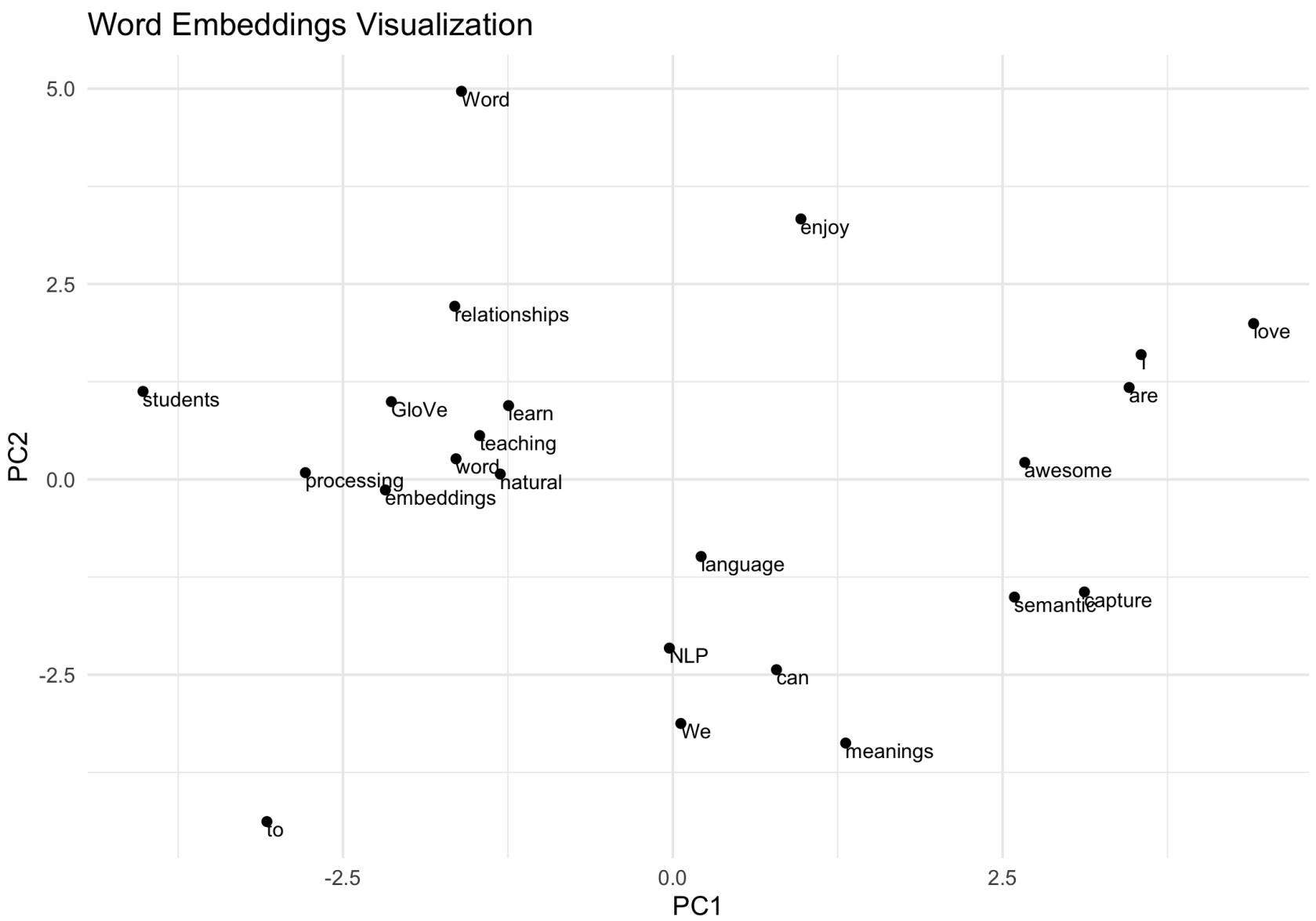
R code (GloVe)

```
> library(text2vec)
> # Tokenize the sentences
> tokens <- word_tokenizer(sentences)
> # Create an iterator over the tokens
> it <- itoken(tokens, progressbar = FALSE)
> # Build the vocabulary
> vocab <- create_vocabulary(it)
> # Prune the vocabulary (optional: remove infrequent or frequent terms)
> vocab <- prune_vocabulary(vocab, term_count_min = 1)
> # Create a term-co-occurrence matrix (TCM)
> tcm <- create_tcm(it, vectorizer = vocab_vectorizer(vocab), skip_grams_window = 5)
> # Define the GloVe model
> glove <- GlobalVectors$new(rank = 50, x_max = 10)  # rank = embedding dimensions
> # Fit the GloVe model
> word_vectors <- glove$fit_transform(tcm, n_iter = 20, convergence_tol = 0.01)
INFO  [14:38:52.747] epoch 1, loss 0.1139
INFO  [14:38:52.774] epoch 2, loss 0.0748
INFO  [14:38:52.779] epoch 3, loss 0.0513
INFO  [14:38:52.780] epoch 4, loss 0.0363
INFO  [14:38:52.780] epoch 5, loss 0.0263
INFO  [14:38:52.781] epoch 6, loss 0.0194
INFO  [14:38:52.782] epoch 7, loss 0.0145
INFO  [14:38:52.783] epoch 8, loss 0.0109
INFO  [14:38:52.784] epoch 9, loss 0.0083
INFO  [14:38:52.785] epoch 10, loss 0.0064
INFO  [14:38:52.786] epoch 11, loss 0.0049
INFO  [14:38:52.786] epoch 12, loss 0.0038
INFO  [14:38:52.787] epoch 13, loss 0.0030
INFO  [14:38:52.788] epoch 14, loss 0.0023
INFO  [14:38:52.789] epoch 15, loss 0.0018
INFO  [14:38:52.790] epoch 16, loss 0.0015
INFO  [14:38:52.791] epoch 17, loss 0.0012
INFO  [14:38:52.792] epoch 18, loss 0.0009
INFO  [14:38:52.793] epoch 19, loss 0.0007
```

R code (GloVe)

```
> # Combine word and context embeddings (optional)
> word_vectors <- word_vectors + t(glove$components)
> # Retrieve embedding for a specific word
> word <- "love"
> word_embedding <- word_vectors[word, ]
> print(word_embedding)
 [1] -0.55416492  0.43191020 -0.58748525  0.39225934  0.47539131  0.27782424
 [7] -0.32605123 -0.19630244 -0.05444455 -0.24727631 -0.45432924  0.40637144
[13] -0.39984916 -0.15407270 -0.38818867  0.30600208 -0.58661460  0.41452079
[19]  0.22671995  0.35373962 -0.82961680 -0.04348827 -0.77532951 -0.72930203
[25]  0.08347611  0.03764002 -0.19398189 -0.36651540  0.29629118  0.40022792
[31]  0.21066985  0.43317064  0.12095259 -0.16691826  0.09775695  0.22341988
[37] -0.36932726 -0.52612566 -0.44224834 -0.32543007  0.13502970 -0.55782452
[43] -0.10243091 -0.48402157  0.35128499 -0.71189768 -0.71381330 -0.24417511
[49] -0.06171226  0.14878652
> library(ggplot2)
> # Perform PCA to reduce dimensions to 2D
> pca <- prcomp(word_vectors, center = TRUE, scale. = TRUE)
> word_vectors_pca <- data.frame(pca$x[, 1:2])
> word_vectors_pca$word <- rownames(word_vectors)
> # Plot the embeddings
> ggplot(word_vectors_pca, aes(x = PC1, y = PC2, label = word)) +
+   geom_point() +
+   geom_text(aes(label = word), hjust = 0, vjust = 1, size = 3) +
+   theme_minimal() +
+   labs(title = "Word Embeddings Visualization", x = "PC1", y = "PC2")
```

R code (GloVe)


Word Embeddings Visualization
5.0
2.5
0.0
-2.5
PC2
-2.5
0.0
2.5
PC1
Word
enjoy
relationships
love
I
are
students
GloVe
learn
teaching
word
natural
processing
embeddings
awesome
language
semantic
capture
NLP
can
We
meanings
to

Suggested reading

1. Two papers about word2vec: Mikolov et al. ((2013a,b))
2. Glove: Pennington et al. ((2014))
3. SPPMI-SVD: Levy and Goldberg ((2014))
4. Application of the word embeddings in healthcare: Beam et al. ((2020))

Summary

- Raw text data based algorithms
  - Skip-gram
  - CBOW
- Co-occurrence matrix based algorithms
  - Co-occurrence and SVD
  - SPPMI-SVD
  - Glove
- More complex deep learning algorithms to be covered in the future
  - RNN
  - BERT
- Pros:
- Cons:

Summary

- Raw text data based algorithms
  - Skip-gram
  - CBOW
- Co-occurrence matrix based algorithms
  - Co-occurrence and SVD
  - SPPMI-SVD
  - Glove
- More complex deep learning algorithms to be covered in the future
  - RNN
  - BERT
- Pros:
- Cons:

For some types of data, such as the electronic health records (EHR) data, only the co-occurrence based algorithms are feasible.

How to evaluate word vectors?

- A general concept of evaluation (in NLP): Intrinsic vs. extrinsic
- **Intrinsic:**
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- **Extrinsic:**
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy $\rightarrow$ Winning!

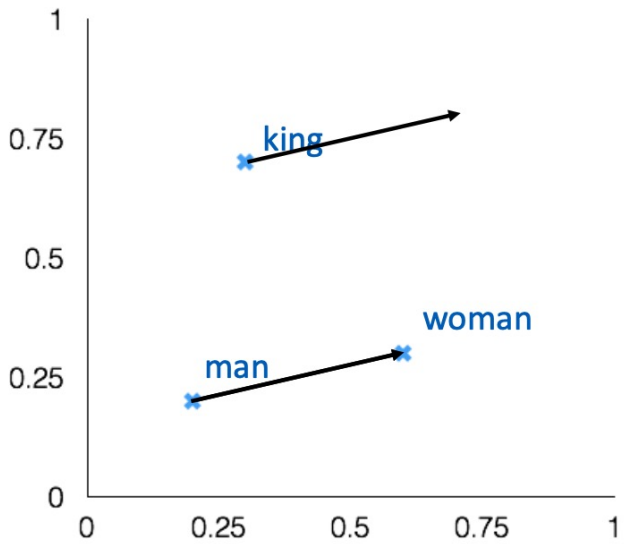
Intrinsic word vector evaluation

- Word Vector Analogies

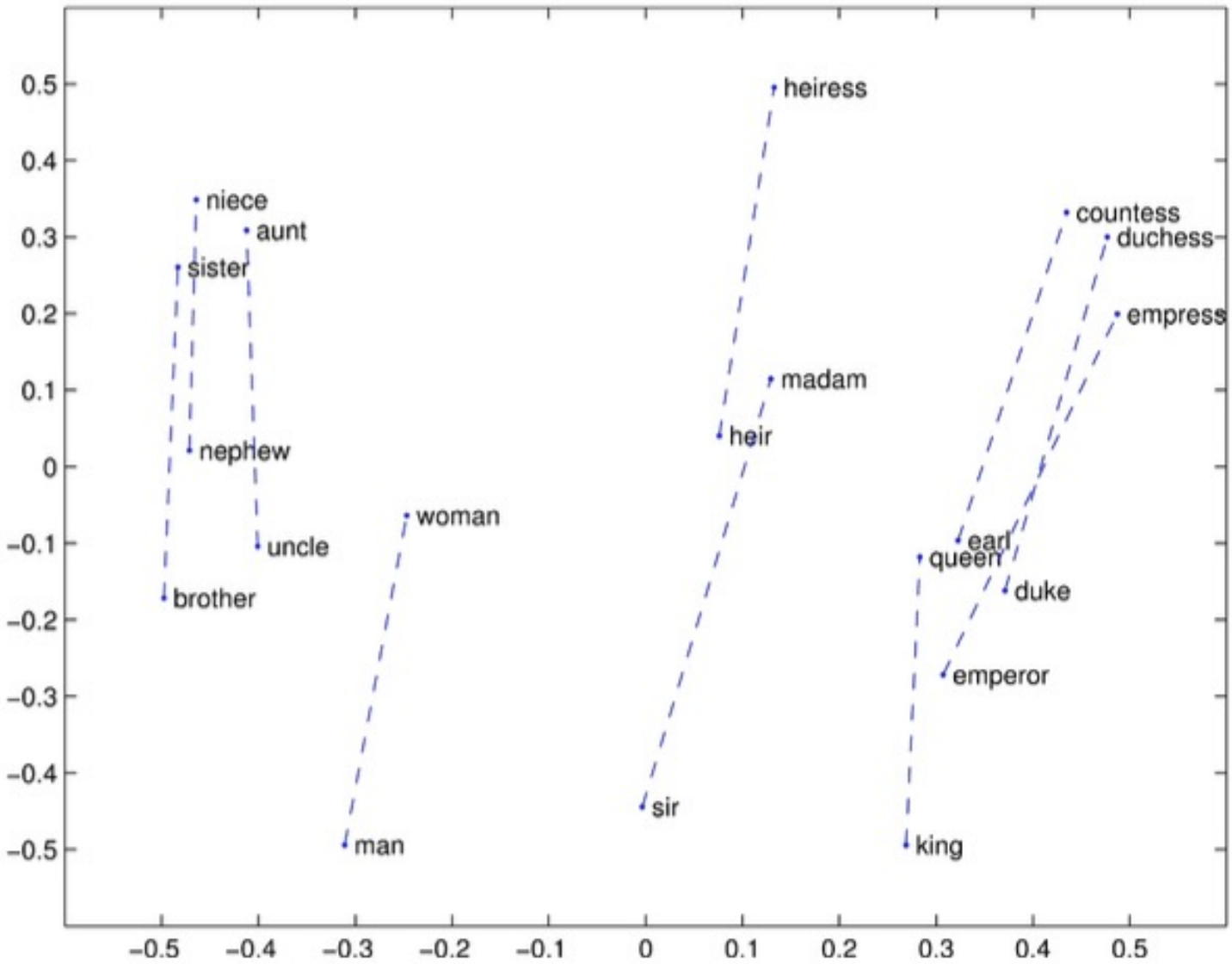
$$\boxed{a : b :: c : ?} \longrightarrow \boxed{d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}}$$

  - Example: man : woman :: king : ?

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search (!!!)
- Problem: What if the information is there but not linear?



GloVe visualization



Meaning similarity: another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353
  https://gabrilovich.com/resources/data/wordsim353/wordsim353.html

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Table: WordSim353 dataset examples

Correlation evaluation

- Word vector distances and their correlation with human judgments

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.8	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	59.6	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

Table: Word vector correlation evaluation with human judgments

Extrinsic word vector evaluation

- One example where good word vectors should help directly: **Named Entity Recognition (NER)**
  - Identifying references to a person, organization, or location.
  - Example: *Chris Manning lives in Palo Alto.*

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	88.7	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	**88.3**	**82.9**	**82.2**

Table: Word Vector Evaluation for Named Entity Recognition

Word senses and word sense ambiguity

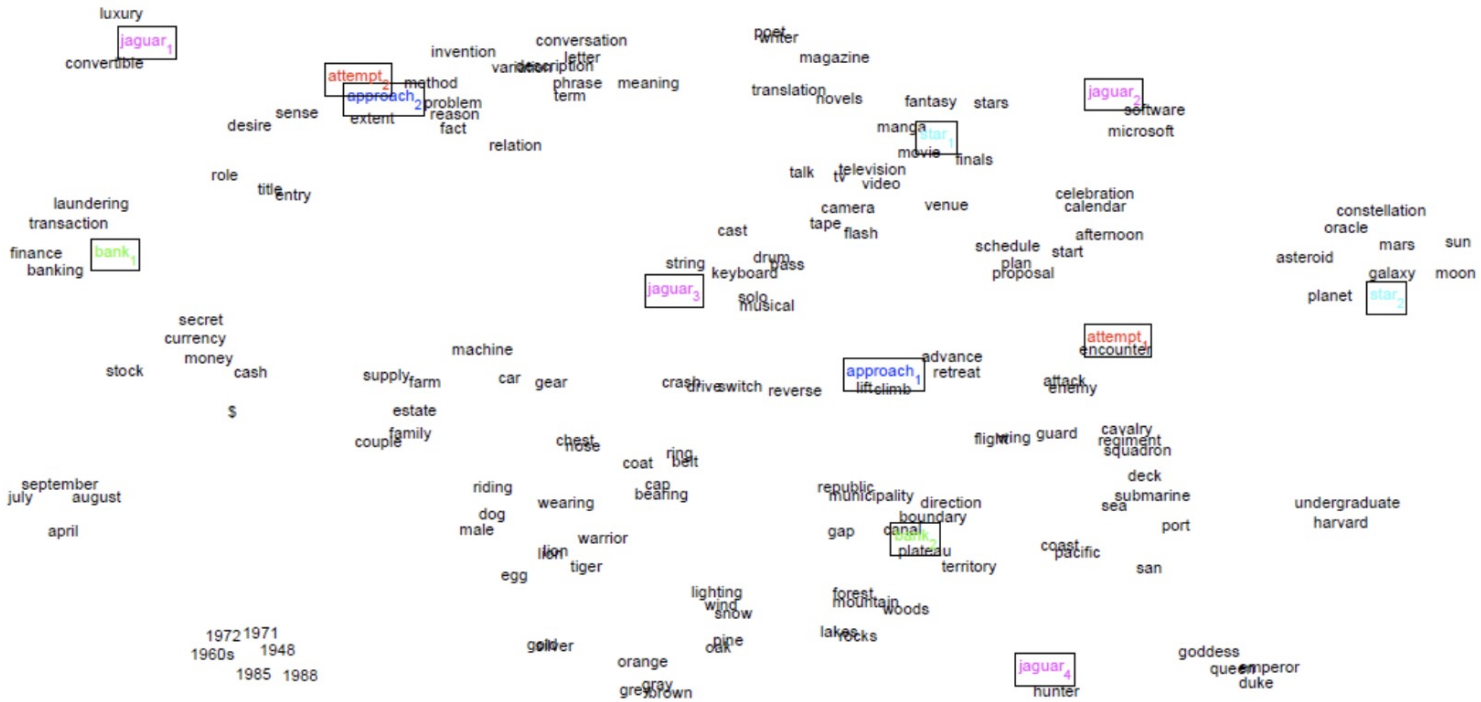
- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time
- **Example: pike**
- Does one vector capture all these meanings or do we have a mess?

Multiple meanings of "Pike"

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, "pike" means to pull out from doing something:
  - *I reckon he could have climbed that cliff, but he piked!*

Improving word representations via global context and multiple word prototypes ((Huang et al., 2012))

- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters bank_1, bank_2, etc.



Linear algebraic structure of word senses ((Arora et al., 2018))

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like Word2Vec:

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$$

- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f
- **Surprising result:**
  - Due to ideas from *sparse coding*, you can actually separate the senses (provided that they are relatively common)!

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

Assignment 1: objective

- Understand and implement key word embedding algorithms
- Train embeddings using real corpora
- Compare model outputs via qualitative and optional quantitative analyses
- Practice scientific reporting and reproducible research

Assignment overview

- **Individual assignment**
- **Report**: PDF with embedded figures and appendix for code
- **Deliverables**:
  - Algorithm explanation
  - Data preprocessing
  - Model training + visualization
  - Comparison and analysis

Algorithms to implement

- Skip-gram
- SPPMI-SVD
- GloVe

Algorithms to implement

- Skip-gram
- SPPMI-SVD
- GloVe

Each model should be trained on the same corpus and compared fairly.

- Corpus option: anything that you feel interesting, for example, English Wikipedia, https://www.gutenberg.org/, ...

Required analysis

- **Nearest neighbors** for selected words
- **Qualitative evaluation**: Are the similar words reasonable?
- **t-SNE/UMAP/PCA visualization** of embedding space
- **Comparison across models**: what differs, what patterns arise?

Bonus opportunities

- Benchmark evaluation (e.g., WordSim-353, analogy tasks)
- Using embeddings for text classification or clustering
- Discovering interesting phenomena: semantic drift, gender bias, etc.

Submission details

- **Deadline**: `[Aug 22, 11:00 pm]`
- **Format**: `YourName_Assignment1.pdf`
- **Include**:
  - PDF report
  - Code in appendix
- **Submission via**: Canvas

Final notes

- Keep experiments reproducible
- Explain results clearly and concisely
- Bonus points reward curiosity and initiative
- Have fun learning about semantic representations!

Questions?

References I

S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski. A latent variable model approach to pmi-based word embeddings. *Transactions of the Association for Computational Linguistics*, 4:385–399, 2016.

S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski. Linear algebraic structure of word senses, with applications to polysemy. *Transactions of the Association for Computational Linguistics*, 6:483–495, 2018.

A. L. Beam, B. Kompa, A. Schmaltz, I. Fried, G. Weber, N. Palmer, X. Shi, T. Cai, and I. S. Kohane. Clinical concept embeddings learned from massive sources of multimodal medical data. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, volume 25, page 295, 2020.

K. Church and P. Hanks. Word association norms, mutual information, and lexicography. *Computational linguistics*, 16(1):22–29, 1990.

I. Dagan, F. Pereira, and L. Lee. Similarity-based estimation of word cooccurrence probabilities. *arXiv preprint cmp-lg/9405001*, 1994.

References II

E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th annual meeting of the association for computational linguistics (Volume 1: Long papers)*, pages 873–882, 2012.

O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. *Advances in neural information processing systems*, 27, 2014.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *International Conference on Learning Representations*, 2013a.

T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013b.

References III

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

P. D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In *European conference on machine learning*, pages 491–502. Springer, 2001.

P. D. Turney and P. Pantel. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37: 141–188, 2010.