CSCE 670 - Information Storage and Retrieval

Lecture 15: Word Embedding, word2vec, GloVe

Yu Zhang

yuzhang@tamu.edu

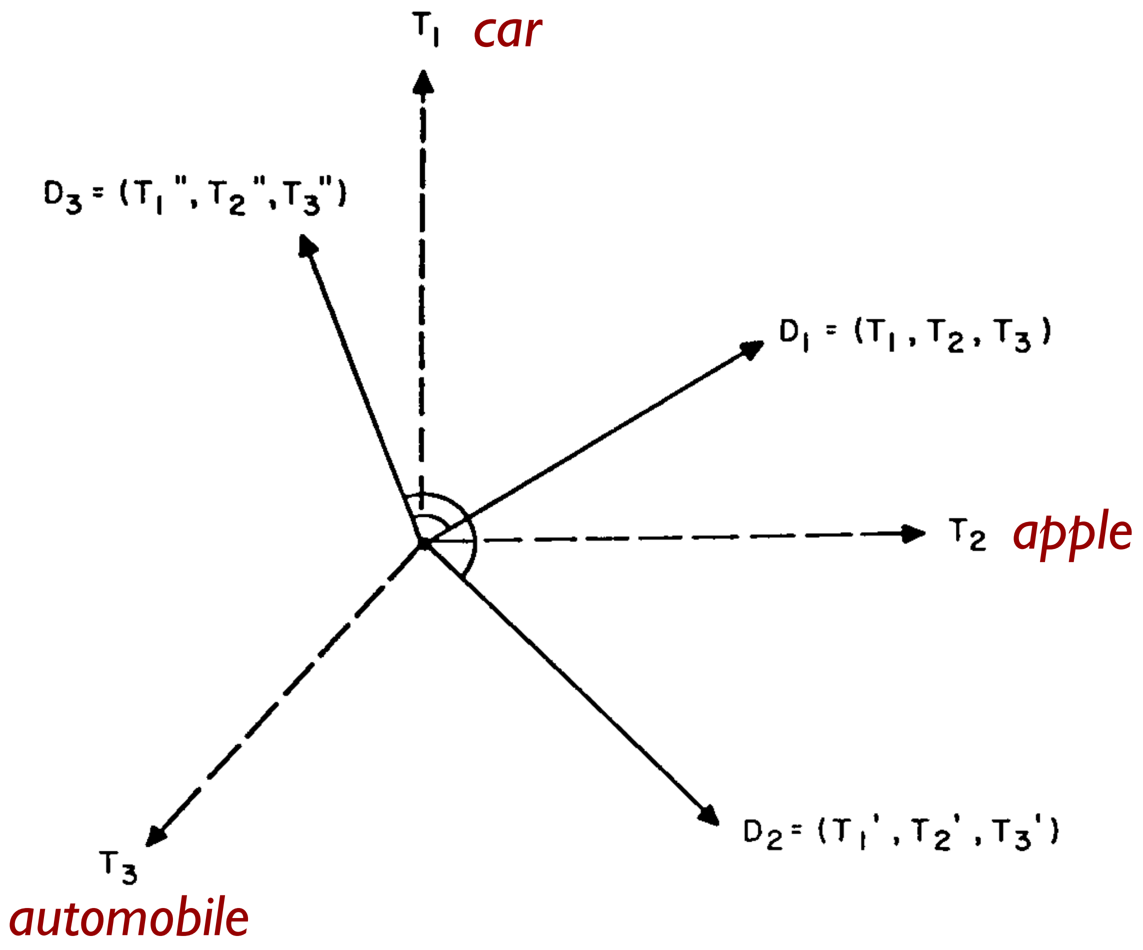
October 16, 2025

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html

Recap: Limitations of "Count-based" Vectors

- Each dimension represents a word in the vocabulary
  - In this case, we assume that any two distinct words are orthogonal to each other!
  - cos(*car*, *apple*) = 0
  - cos(*car*, *automobile*) = 0
  - cos(*car*, *car*) = 1
- However, the meanings of "*car*" and "*automobile*" are very similar, so we should have
  - cos(*car*, *automobile*) > cos(*car*, *apple*)

Fig. 1. Vector representation of document space.

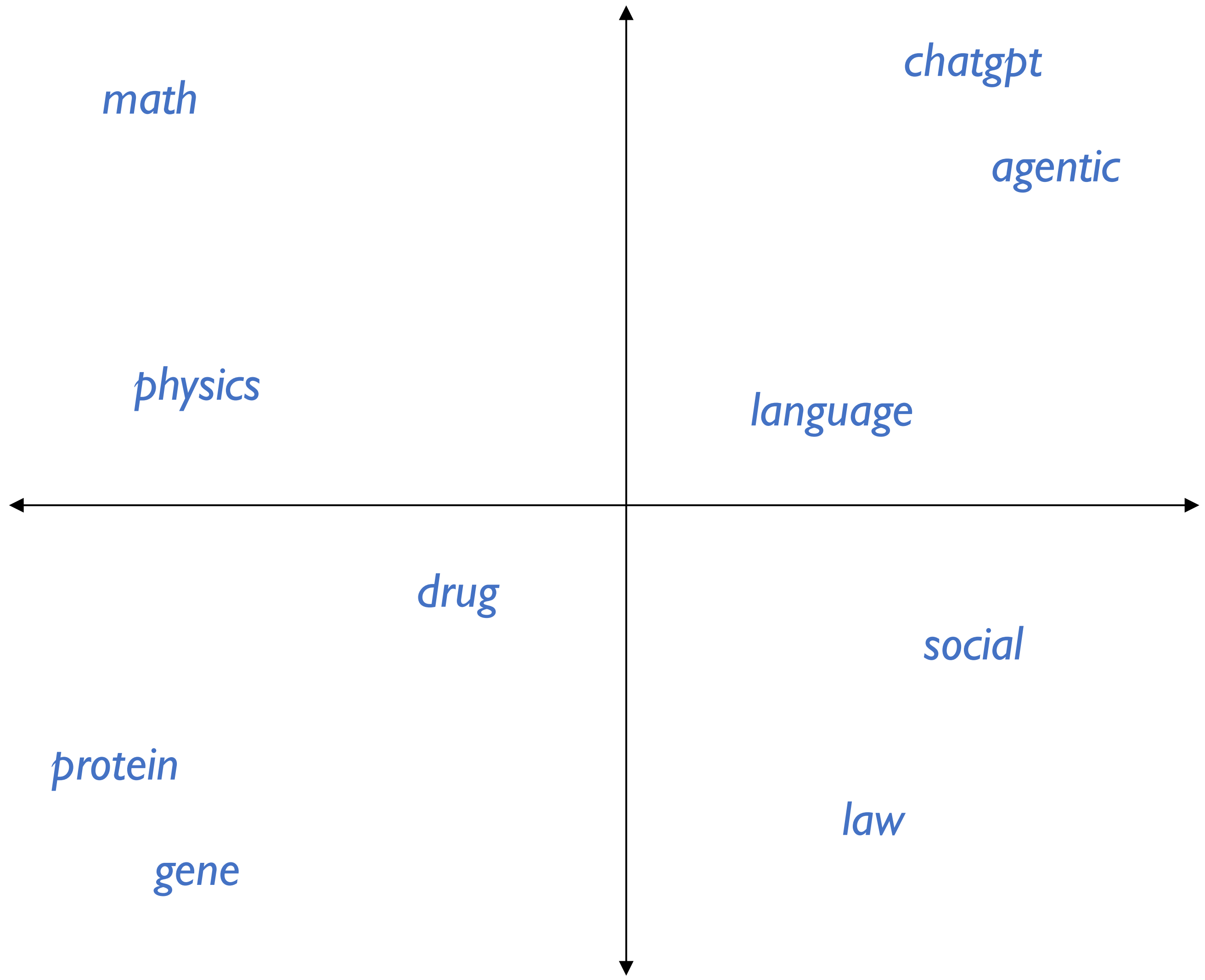


Recap: Limitations of "Count-based" Vectors

	abandon	…	*chatgpt*	*generative*	*language*	*models*	…	*zucchini*
Doc 1	0	…	1	1	0	0	…	0
Doc 2	0	…	0	0	1	1	…	0

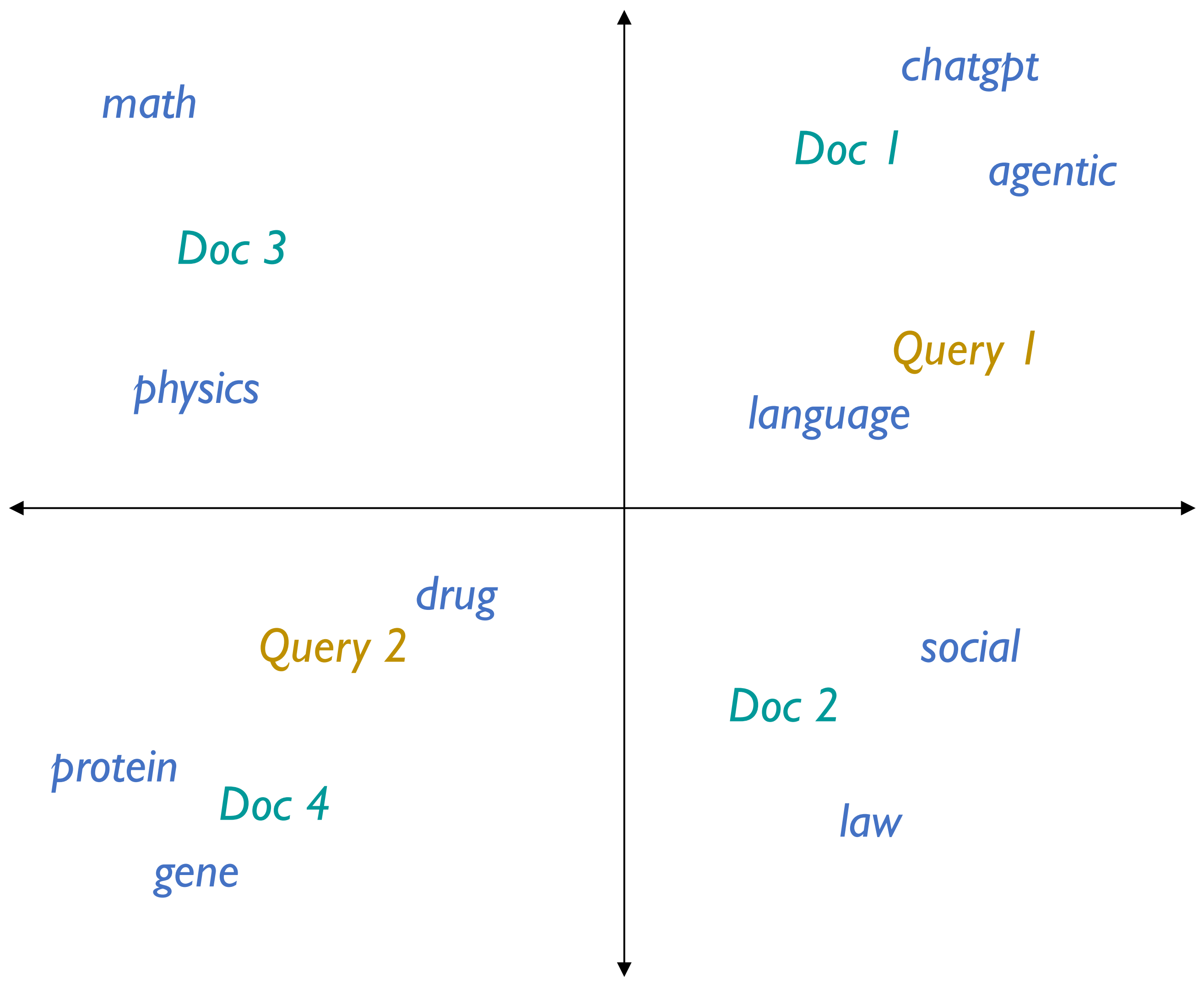
- Limitation 1: Doc 1 and Doc 2 are totally irrelevant according to Boolean, TF-IDF, and BM25.
  - Because we directly use the corresponding row to represent each document.
  - The inner product or cosine similarity between them is 0.
- Limitation 2: Vectors are sparse (with lots of zeros)
  - Zero values in the vectors do not carry any semantics.
- Limitation 3: Vectors are long (with many dimensions)
  - Vector dimension = vocabulary size (usually > 10K)
  - "Curse of dimensionality": Metrics (e.g., cosine) become less meaningful in high dimensions.

We hope the words are distributed in the vector space in this way!

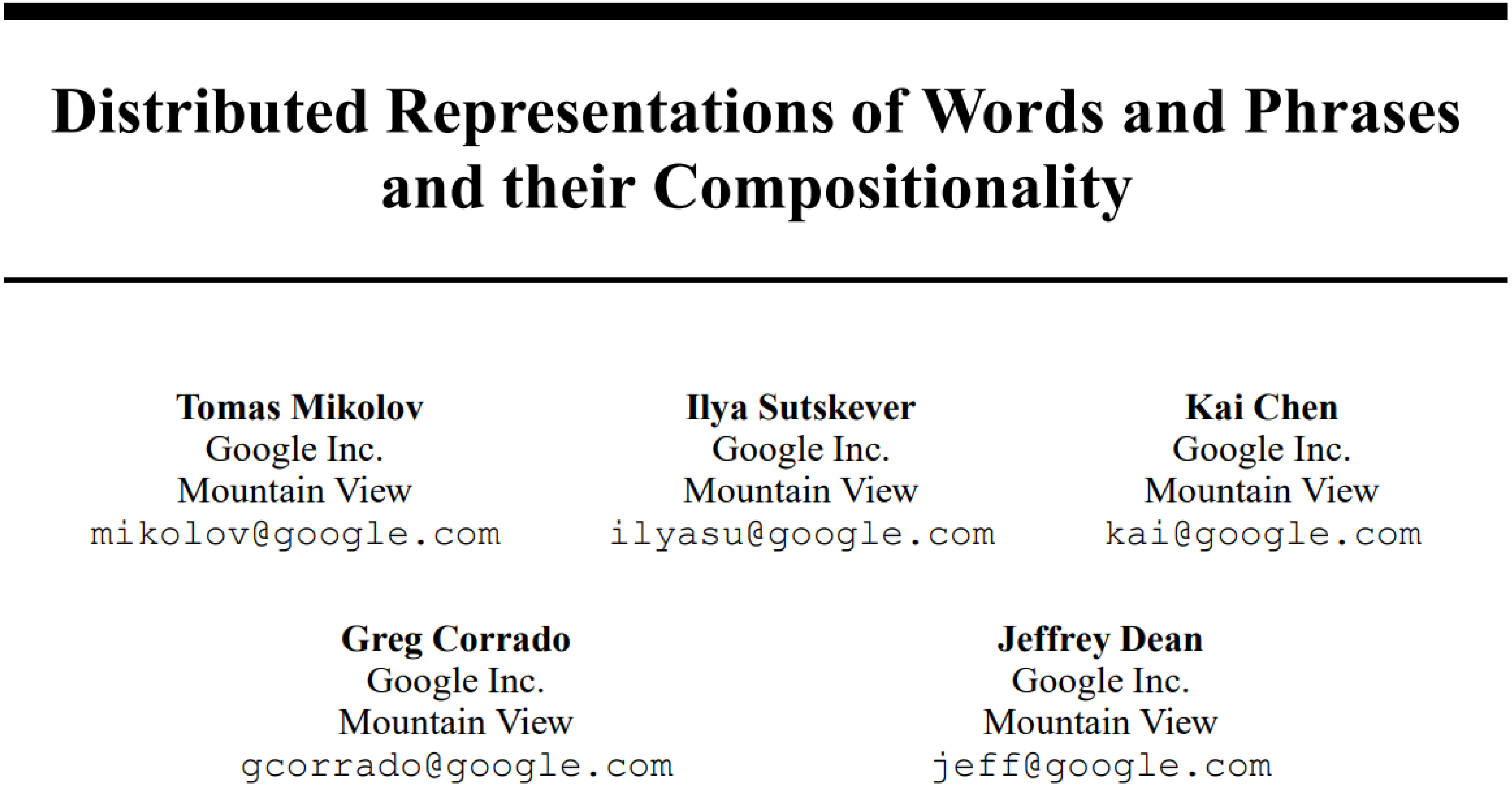


Each dimension represents a latent factor rather than an explicit word.

And queries and documents in this way!



word2vec [Mikolov et al., NIPS 2013]



Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

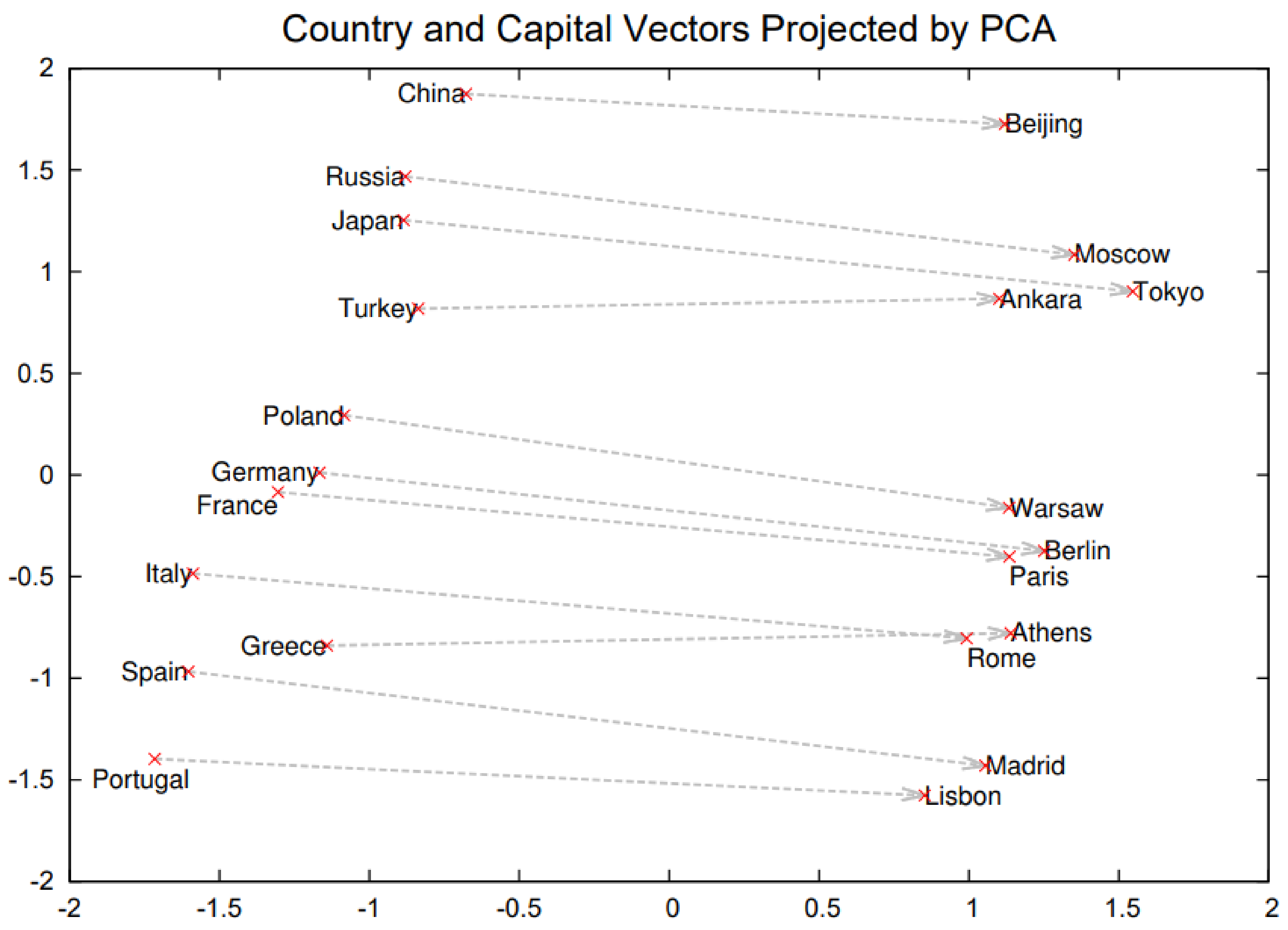
Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

2-dimensional Visualization of 1000-dimensional word2vec Vectors



Idea: Distributional Hypothesis [Harris, 1954]

- A word's meaning is largely defined by its context.
- Words that occur in similar contexts tend to have similar meanings.
- Example:
  - Suppose we don't know the meaning of "*ong choy*" but see the following:
    - *ong choy is delicious sautéed with garlic*
    - *ong choy is superb over rice*
    - … *ong choy leaves with salty sauces*
  - And we've seen the following contexts:
    - … *spinach sautéed with garlic over rice*
    - … *chard stems and leaves are delicious*
    - … *collard greens and other salty leafy greens*

ong choy = *water spinach*

Idea: Distributional Hypothesis [Harris, 1954]

- A word's meaning is largely defined by its context.
- Words that occur in similar contexts tend to have similar meanings.
- Example:
  - *openai proposes training chatgpt using reinforcement learning*
  - *alibaba proposes training qwen using reinforcement learning*

chatgpt is semantically similar to *qwen*

How to leverage the Distributional Hypothesis?

- Hypothesis: A word's meaning is largely defined by its context.
- Task: Mask a word and predict it using its context.

- Assume each word w is represented by a vector $\boldsymbol{e}_w$ (also known as embedding)

$\boldsymbol{e}_{\text{openai}}$	$\boldsymbol{e}_{\text{proposes}}$	$\boldsymbol{e}_{\text{training}}$		$\boldsymbol{e}_{\text{using}}$	$\boldsymbol{e}_{\text{reinforcement}}$	$\boldsymbol{e}_{\text{learning}}$
openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			?			

- Step 1: Sum (or average) the embedding vectors of context words → $\boldsymbol{e}_c$
- Step 2: Find the word in the entire vocabulary whose embedding is most similar to $\boldsymbol{e}_c$

How to leverage the Distributional Hypothesis?

- Example

[0.3, -0.7]	[−0.9, 0.4]	[0.8, 0.2]		[−0.2, −0.6]	[0.5, 0.9]	[−0.7, −0.1]
openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			chatgpt			

- Step 1: $\boldsymbol{e}_c$ = [0.3, -0.7] + [-0.9, 0.4] + [0.8, 0.2] + [-0.2, -0.6] + [0.5, 0.9] + [-0.7, -0.1]
  = [-0.2, 0.1]
- Step 2: Learn embeddings so that "*chatgpt*" has the largest inner product with [-0.2, 0.1] among all words in the entire vocabulary

w	*abandon*	…	*chatgpt*	*learning*	*openai*	*proposes*	…	*zucchini*
$\boldsymbol{e}_w$	[0.1, 0]	…	[-0.2, 0.2]	[-0.7, 0.1]	[0.3, -0.7]	[-0.9, 0.4]	…	[-0.1, -0.3]

More Details about the Learning Task

- What are the parameters to be learned?
  - Embedding vectors $\boldsymbol{e}_w$ of all words w in the vocabulary $\mathcal{V}$

w	*abandon*	…	*chatgpt*	*learning*	*openai*	*proposes*	…	*zucchini*
$\boldsymbol{e}_w$	[0.1, 0]	…	[-0.2, 0.2]	[-0.7, 0.1]	[0.3, -0.7]	[-0.9, 0.4]	…	[-0.1, -0.3]

  - If there are 10,000 words in the vocabulary, and the embedding space has 100 dimensions, how many parameters are there in total?
    - $10{,}000 \times 100 = 1{,}000{,}000$

More Details about the Learning Task

- What is the learning objective?

[0.3, -0.7]	[−0.9, 0.4]	[0.8, 0.2]		[−0.2, −0.6]	[0.5, 0.9]	[−0.7, −0.1]
openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			chatgpt			

- Maximizing the probability that "*chatgpt*" is the center word w, given that the context c is "*openai proposes training ____ using reinforcement learning*":
  - $p(w|c) \propto \exp(\boldsymbol{e}_w^T \boldsymbol{e}_c)$
  - In other words,

$$p(w|c) = \frac{\exp(\boldsymbol{e}_w^T \boldsymbol{e}_c)}{\sum_{v \in \mathcal{V}} \exp(\boldsymbol{e}_v^T \boldsymbol{e}_c)}$$

Softmax: typically employed as the output layer for multiclass classification tasks, just as Sigmoid is used in binary classification

More Details about the Learning Task

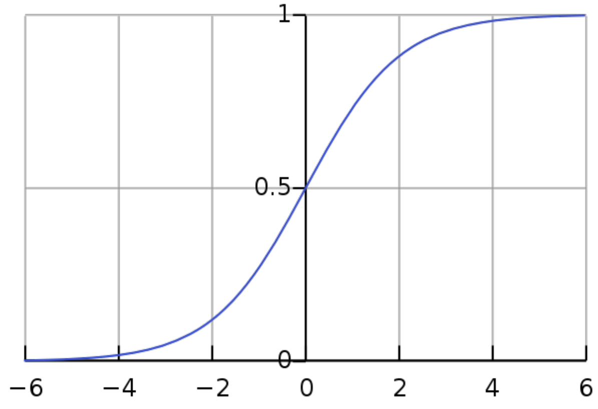
- What is the learning objective?

$$p(w|c) = \frac{\exp(\boldsymbol{e}_w^T \boldsymbol{e}_c)}{\boxed{\sum_{v \in \mathcal{V}}} \exp(\boldsymbol{e}_v^T \boldsymbol{e}_c)}$$

  - But summing over the entire vocabulary is too expensive!
  - Negative sampling: Randomly sample a few negative terms from the vocabulary to form a negative set $\mathcal{N}$
    - In practice, $|\mathcal{N}|$ is usually between 5 and 10.
  - Formulate a binary classification task to predict whether (w, c) is a real word-context pair.

$$p(\text{True}|w, c) = \text{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c) = \frac{1}{1 + \exp(-\boldsymbol{e}_w^T \boldsymbol{e}_c)}$$

$p(\text{True}|w, c)$



$\boldsymbol{e}_w^T \boldsymbol{e}_c$

More Details about the Learning Task

- What is the learning objective?
  - Negative sampling: Randomly sample a few negative terms from the vocabulary to form a negative set $\mathcal{N}$
  - Formulate a binary classification task to predict whether (w, c) is a real word-context pair.

$$p(\text{True}|w, c) = \text{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c) = \frac{1}{1 + \exp(-\boldsymbol{e}_w^T \boldsymbol{e}_c)}$$

  - Maximize the binary classification probability for real word-context pairs, and minimize for negative (random) pairs.

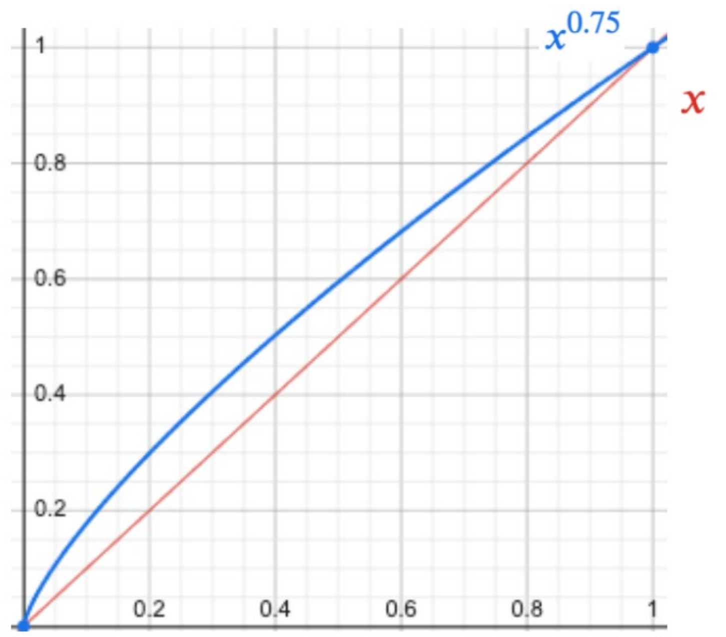
$$\max_{\{\boldsymbol{e}_v | v \in \mathcal{V}\}} \underbrace{\log\left(\text{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c)\right)}_{\text{real word-context pair}} + \sum_{v \in \mathcal{N}} \underbrace{\log\left(\text{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_c)\right)}_{\text{negative word-context pair}}$$

More Details about the Learning Task

- What is the learning objective?

$$\max_{\{\boldsymbol{e}_v \mid v \in \mathcal{V}\}} \log\big(\mathrm{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c)\big) + \sum_{v \in \mathcal{N}} \log\big(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_c)\big)$$

- How to sample negatives?
  - Based on the (power-smoothed) unigram distribution
  - $p_{\text{neg}}(v) \propto \left(\frac{\#(v)}{\sum_{v \in \mathcal{V}} \#(v)}\right)^{0.75}$
  - Rare words get a bit boost in sampling probability



More Details about the Learning Task

- What is the learning objective?

$$\max_{\{\boldsymbol{e}_v \mid v \in \mathcal{V}\}} \log\big(\mathrm{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c)\big) + \sum_{v \in \mathcal{N}} \log\big(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_c)\big)$$

- How to optimize the objective?
  - Gradient descent
  - Given the following task, which parameters (word embeddings) will be updated by gradient descent?

openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			chatgpt			

More Details about the Learning Task

- What is the training data?
  - Before answering this question, we first need to define "context".
  - In word2vec, context refers to $\pm k$ words.
  - Example: $k = 3$

openai	*proposes*	*training*	*[MASK]*	*using*	*reinforcement*	*learning*
			↓			
			chatgpt			

  - *chatgpt learns from vast amounts of text data and generates responses that sound natural and coherent while adapting to different topics and user intents through continuous optimization and large scale training*

  - In practice, k is usually between 5 and 10.

More Details about the Learning Task

- What is the training data?
  - Each word together with its context can be used as training data.
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - *chatgpt learns from vast amounts of text data and generates responses …*
  - …

Final Objective

$$\max_{\{\boldsymbol{e}_v | v \in \mathcal{V}\}} \sum_{d \in D} \sum_{w \in d} \left(\log\left(\text{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c)\right) + \sum_{v \in \mathcal{N}} \log\left(\text{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_c)\right) \right)$$

- D: a corpus (the larger, the better!)
- d: a document in the corpus

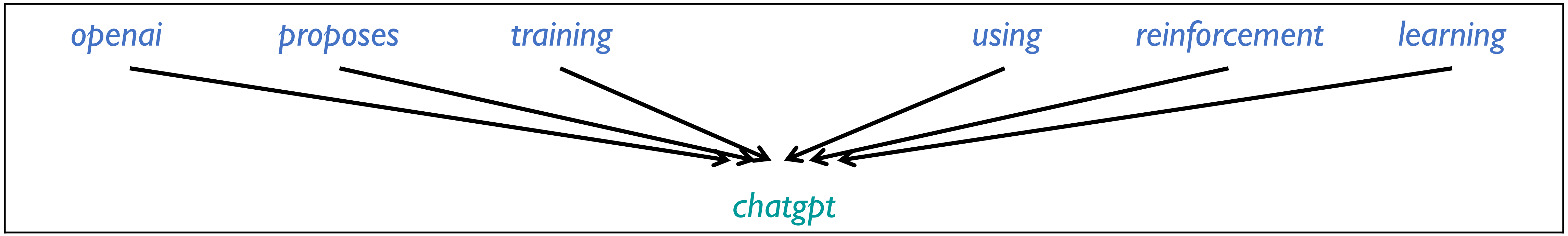
- Each word w may appear multiple times in the corpus.
- Each time, w may have a different context c.
- Our objective requires that w be close not just to one context, but to all the contexts in the corpus.

- Can you estimate roughly where the embedding of "*and*" lies in the vector space?
    - Close to the center of all word embeddings

Questions?

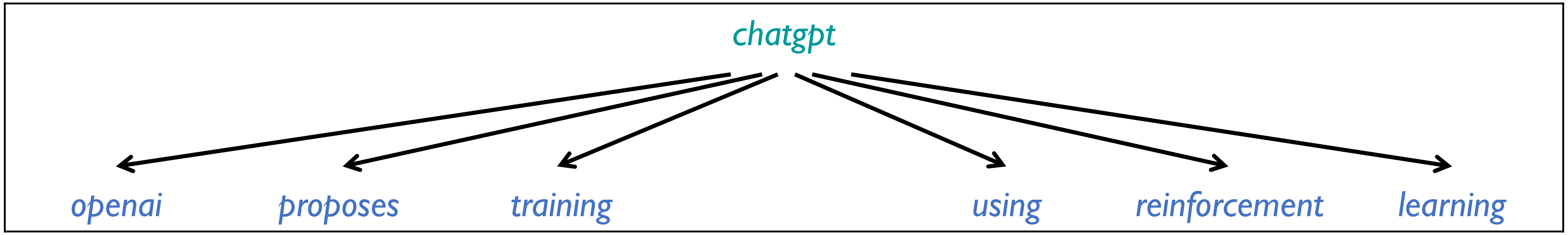
Continuous Bag-of-Words (CBOW)

- word2vec has two variants
- The one we just introduced is Continuous Bag-of-Words (CBOW)
  - Idea: using context to predict word



Skip-Gram

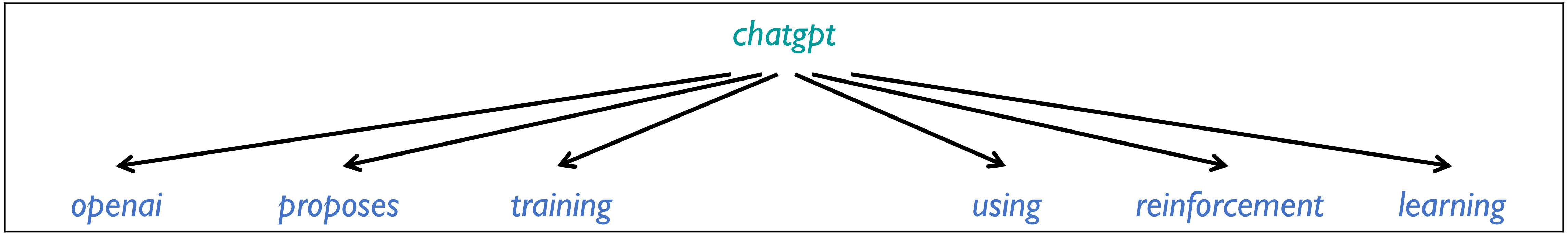
- word2vec has two variants
- The other one is Skip-Gram
  - Idea: using word to predict context



- Use word embedding to predict the sum of context embeddings?
  - Hard to create negative samples!
- Use word embedding to predict each context word!

Skip-Gram

- Idea: using word to predict context



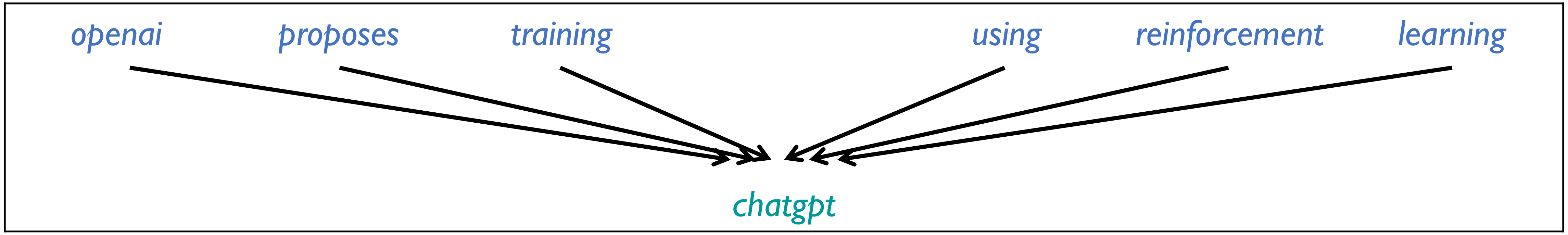
- Learning objective:

$$\max_{\{\boldsymbol{e}_v | v \in \mathcal{V}\}} \sum_{d \in D} \sum_{w \in d} \sum_{x \in \mathrm{Context}(w)} \left(\log\left(\mathrm{Sigmoid}(\boldsymbol{e}_x^T \boldsymbol{e}_w)\right) + \sum_{v \in \mathcal{N}} \log\left(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_w)\right) \right)$$

$\mathrm{Context}(w)$: $\pm k$ words of w in the text

Continuous Bag-of-Words (CBOW)

- Idea: using context to predict word



- Learning objective:

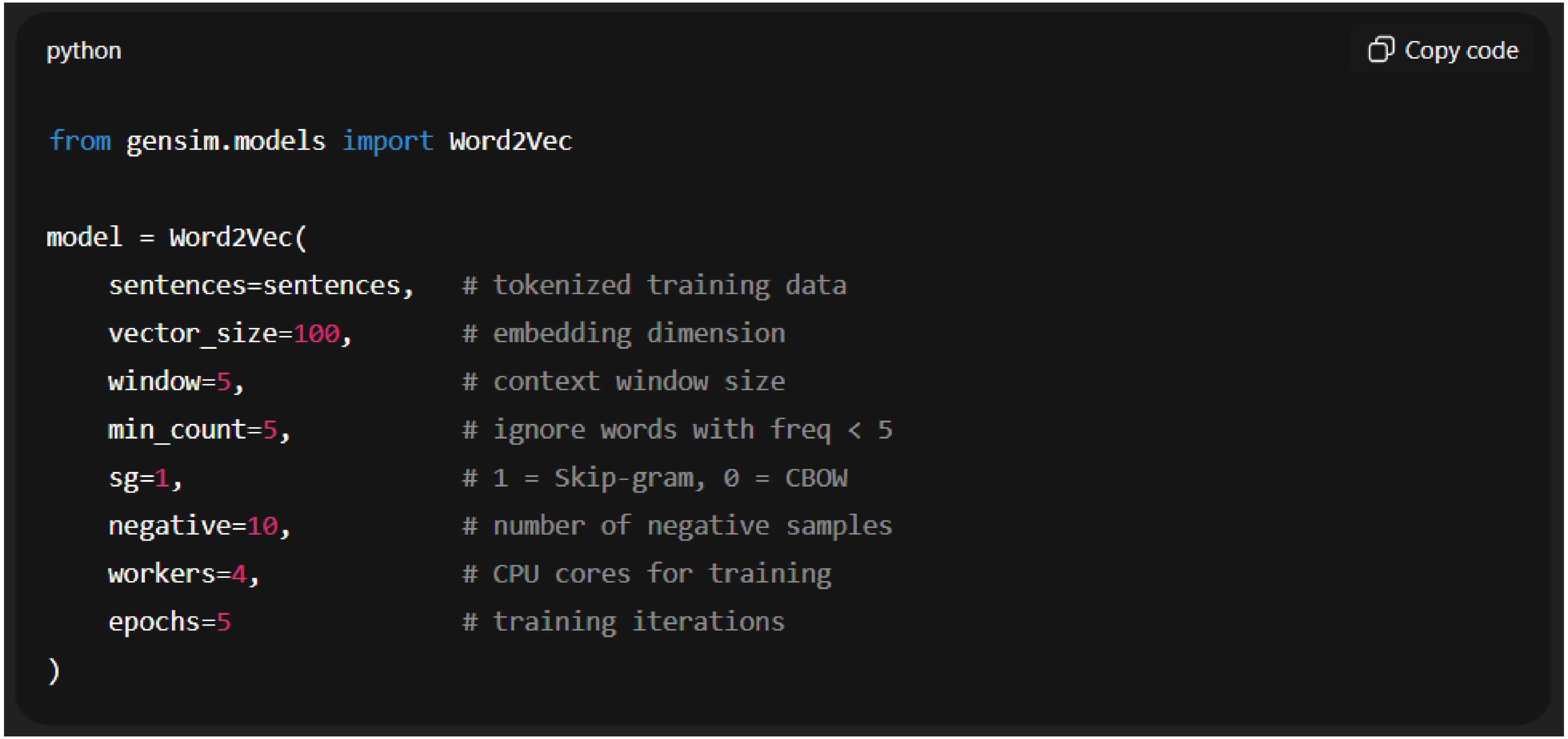
$$\max_{\{\boldsymbol{e}_v | v \in \mathcal{V}\}} \sum_{d \in D} \sum_{w \in d} \left(\log\left(\text{Sigmoid}(\boldsymbol{e}_w^T \boldsymbol{e}_c)\right) + \sum_{v \in \mathcal{N}} \log\left(\text{Sigmoid}(-\boldsymbol{e}_v^T \boldsymbol{e}_c)\right) \right)$$

where $\boldsymbol{e}_c = \sum_{x \in \text{Context}(w)} \boldsymbol{e}_x$

word2vec Hyperparameters

- Context window size k (usually 5-10)
  - Smaller k learns from immediately nearby words – more syntactic information
  - Larger k learns from longer-ranged contexts – more semantic/topical information
- Word embedding dimension d (usually 100-300)
  - Larger d provides richer vector semantics
  - Extremely large d suffers from inefficiency and curse of dimensionality
- Number of negative samples $b = |\mathcal{N}|$ (usually 5-10)
  - Larger b usually makes training more stable but also more costly
- Minimum word count (usually 5-10)
  - Any word that occurs fewer times than this threshold is ignored
  - Setting the threshold too high may remove useful but uncommon words; setting it too low may make training slower and embeddings noisier.

Python Implementation



```python
from gensim.models import Word2Vec

model = Word2Vec(
    sentences=sentences,   # tokenized training data
    vector_size=100,       # embedding dimension
    window=5,              # context window size
    min_count=5,           # ignore words with freq < 5
    sg=1,                  # 1 = Skip-gram, 0 = CBOW
    negative=10,           # number of negative samples
    workers=4,             # CPU cores for training
    epochs=5               # training iterations
)
```

word2vec is Self-Supervised Learning

- Self-supervised learning: a model learns to predict parts of its input from other parts of the same input
  - No human-labeled data
  - The model seeks supervision from the unlabeled data itself

- Examples:
  - word2vec – predict a word from its $\pm k$ words in a sentence

    chatgpt learns from vast amounts of text data and generates responses …

  - Encoder-based language models (e.g., BERT) – predict a token from all other tokens in the input sequence

    chatgpt learns from vast amounts of text data and generates responses …

  - Decoder-based language models (e.g., ChatGPT) – predict a token from all previous tokens

    chatgpt learns from vast amounts of text data and generates responses …

word2vec as Matrix Factorization (Extended Content)

- [Levy and Goldberg, NIPS 2014]:
  - If we use the Skip-Gram variant,
  - and the negative sampling strategy is $p_{\text{neg}}(v) \propto \frac{\#(v)}{\sum_{v \in \mathcal{V}} \#(v)}$,

    (not raised to the power of 0.75!)
  - then word2vec is implicitly factorizing the matrix

$$\left[\log \frac{\#(w, x) \cdot |\mathcal{D}|}{\#(w) \cdot \#(x) \cdot b}\right]_{w \in \mathcal{V}, x \in \mathcal{V}}$$

$\#(w, x)$: number of times w and x co-occur in a context window

$\#(w)$: number of times w occurs in the corpus D

$|D|$: number of words in the corpus ($= \sum_{w \in \mathcal{V}} \#(w)$)

word2vec as Matrix Factorization (Extended Content)

- Proof:

$$\sum_{d\in D}\sum_{w\in d}\sum_{x\in \mathrm{Context}(w)}\left(\log\left(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\right)+\sum_{v\in\mathcal{N}}\log\left(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T\boldsymbol{e}_w)\right)\right)$$

$$=\sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}}\#(w,x)\log\left(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\right)+\sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}}\#(w,x)\sum_{v\in\mathcal{N}}\log\left(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T\boldsymbol{e}_w)\right)$$

$$=\sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}}\#(w,x)\log\left(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\right)+\sum_{w\in\mathcal{V}}\#(w)\sum_{v\in\mathcal{N}}\log\left(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T\boldsymbol{e}_w)\right)$$

Let's look at the second term from the perspective of expectation:

$$=\sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}}\#(w,x)\log\left(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\right)+\sum_{w\in\mathcal{V}}\#(w)\cdot b\sum_{v\in\mathcal{V}}\frac{\#(v)}{|D|}\log\left(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T\boldsymbol{e}_w)\right)$$

word2vec as Matrix Factorization (Extended Content)

$$\sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}}\#(w,x)\log\big(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\big)+\sum_{w\in\mathcal{V}}\#(w)\cdot b\sum_{v\in\mathcal{V}}\frac{\#(v)}{|D|}\log\big(\mathrm{Sigmoid}(-\boldsymbol{e}_v^T\boldsymbol{e}_w)\big)$$

- If we focus on a specific pair of (w,x) in this objective:

$$J=\#(w,x)\log\big(\mathrm{Sigmoid}(\boldsymbol{e}_x^T\boldsymbol{e}_w)\big)+\frac{\#(w)\cdot b\cdot\#(x)}{|D|}\log\big(\mathrm{Sigmoid}(-\boldsymbol{e}_x^T\boldsymbol{e}_w)\big)$$

- Let $\theta=\boldsymbol{e}_x^T\boldsymbol{e}_w$. To maximize J, we need to have

$$0=\frac{\partial J}{\partial\theta}=\#(w,x)\big(1-\mathrm{Sigmoid}(\theta)\big)+\frac{\#(w)\cdot b\cdot\#(x)}{|D|}\big(1-\mathrm{Sigmoid}(-\theta)\big)$$

- Substitute $\mathrm{Sigmoid}(\theta)=\frac{1}{1+\exp(-\theta)}$ into above

word2vec as Matrix Factorization (Extended Content)

$$\exp(2\theta) - \left(\frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)\cdot b} - 1\right)\exp(\theta) - \frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)\cdot b} = 0$$

- Two solutions:

$$\exp(\theta) = -1$$

$$\exp(\theta) = \frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)\cdot b}$$

- The first solution is invalid, so

$$\boldsymbol{e}_x^T\boldsymbol{e}_w = \theta = \log\left(\frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)\cdot b}\right)$$

word2vec as Matrix Factorization (Extended Content)

$$\boldsymbol{e}_x^T \boldsymbol{e}_w = \log\left(\frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)\cdot b}\right)$$

- Note 1: In information theory, the pointwise mutual information is defined as
  - $\mathrm{PMI}(w,x) = \frac{p(w,x)}{p(w)p(x)}$
  - $\mathrm{PMI}(w,x) = \frac{\frac{\#(w,x)}{|D|}}{\frac{\#(w)}{|D|}\cdot\frac{\#(x)}{|D|}} = \frac{\#(w,x)\cdot|D|}{\#(w)\cdot\#(x)}$
- So Skip-Gram is implicitly factorizing $\left[\log\big(\mathrm{PMI}(w,x)\big) - \log b\right]_{w\in\mathcal{V},x\in\mathcal{V}}$

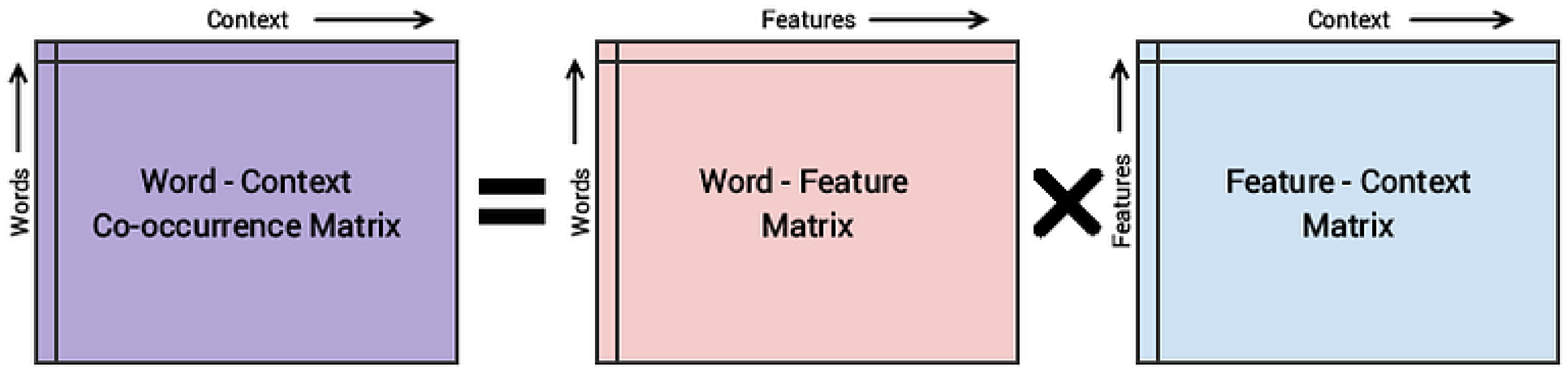
- Note 2: What if we explicitly factorize this matrix?
  - Better than word2vec [Levy and Goldberg, NIPS 2014]
  - But much slower!

Other Word Embedding Techniques

GloVe [Pennington et al., EMNLP 2014]

- Factorizing $[\log(\#(w,x))]_{w\in\mathcal{V},x\in\mathcal{V}}$ while taking bias into account

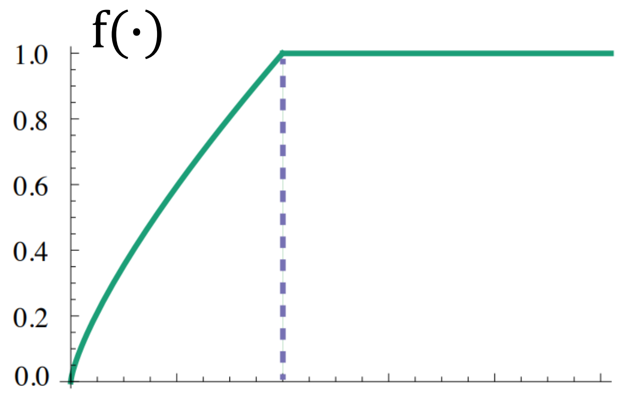
$$\log(\#(w,x)) = b_w + \tilde{b}_x + \boldsymbol{e}_w^T\tilde{\boldsymbol{e}}_x$$



- Learning objective:

$$J = \sum_{w\in\mathcal{V}}\sum_{x\in\mathcal{V}} \mathrm{f}(\#(w,x))\left(b_w + \tilde{b}_x + \boldsymbol{e}_w^T\tilde{\boldsymbol{e}}_x - \log(\#(w,x))\right)^2$$

$\mathrm{f}(\#(w,x))$: weight word-context pairs according to their co-occurrence

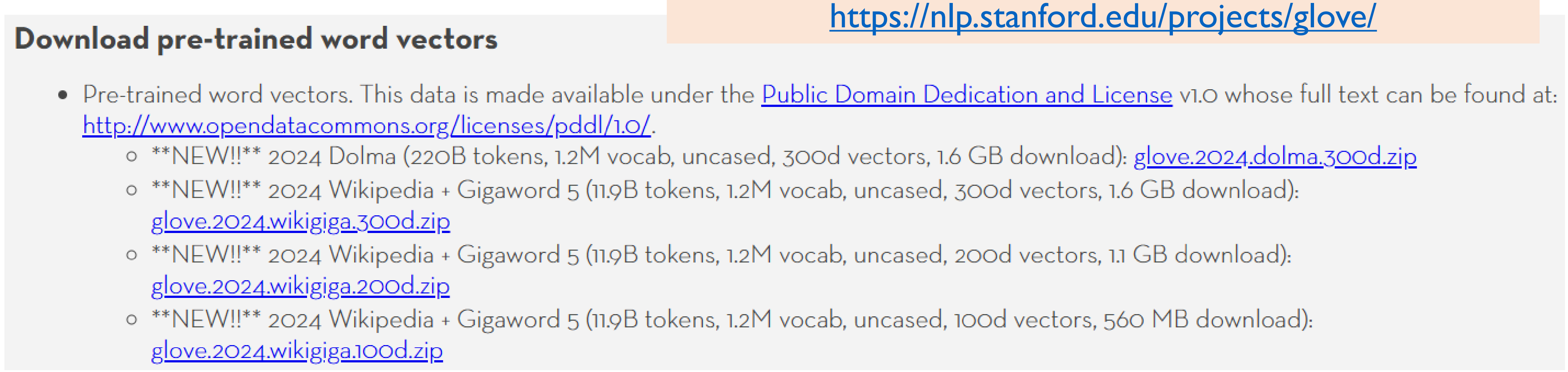


word2vec vs. GloVe

- According to [Pennington et al., EMNLP 2014]:

Model	Dimension	Semantic	Syntactic	Total
SVD	300	6.3	8.1	7.3
CBOW	300	63.6	67.4	65.7
Skip-Gram	300	73.0	66.0	69.1
GloVe	300	77.4	67.0	71.7

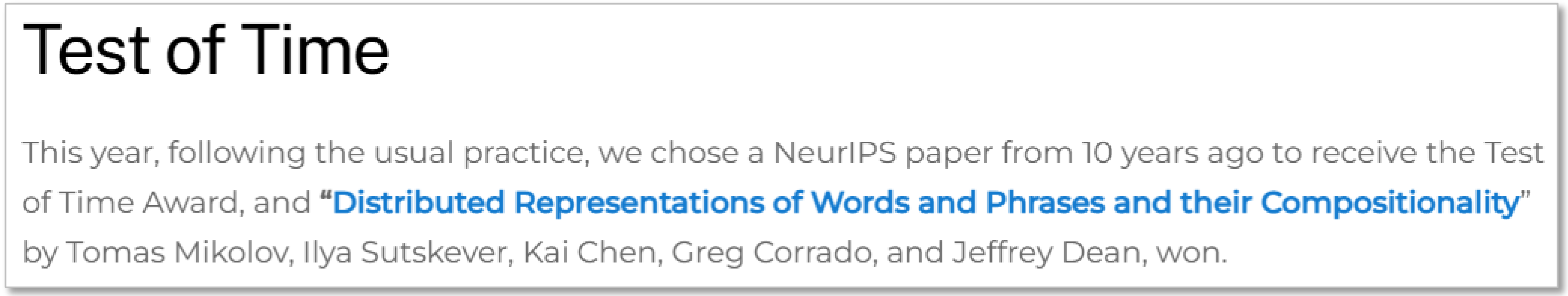
https://nlp.stanford.edu/projects/glove/



Download pre-trained word vectors

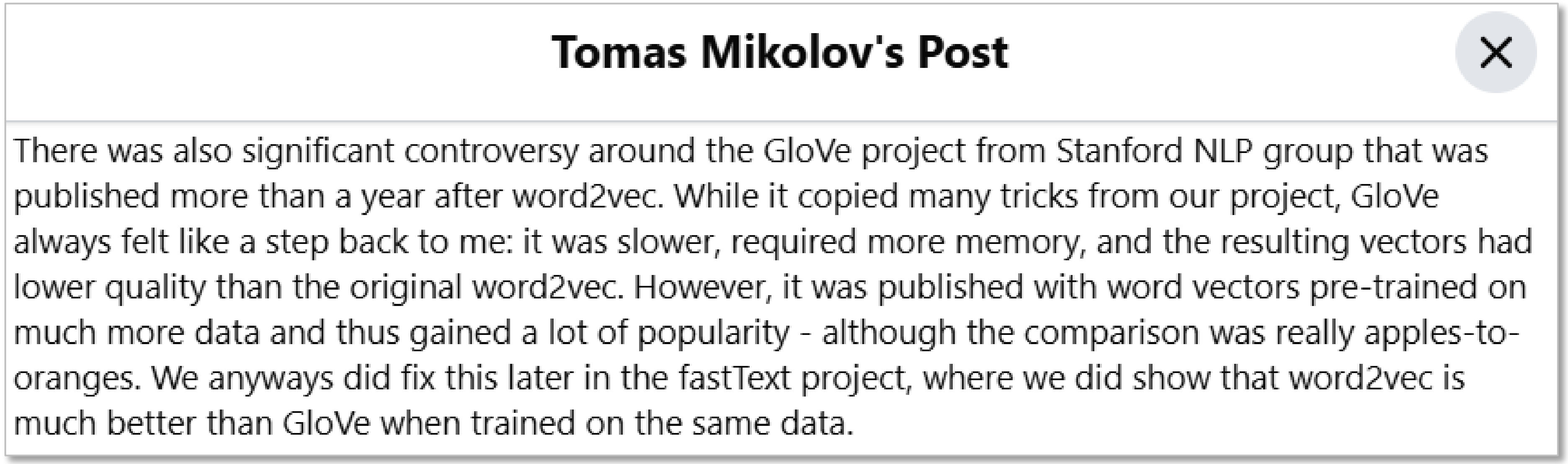
- Pre-trained word vectors. This data is made available under the Public Domain Dedication and License v1.0 whose full text can be found at: http://www.opendatacommons.org/licenses/pddl/1.0/.
  - **NEW!!** 2024 Dolma (220B tokens, 1.2M vocab, uncased, 300d vectors, 1.6 GB download): glove.2024.dolma.300d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 300d vectors, 1.6 GB download): glove.2024.wikigiga.300d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 200d vectors, 1.1 GB download): glove.2024.wikigiga.200d.zip
  - **NEW!!** 2024 Wikipedia + Gigaword 5 (11.9B tokens, 1.2M vocab, uncased, 100d vectors, 560 MB download): glove.2024.wikigiga.100d.zip

word2vec vs. GloVe



Test of Time

This year, following the usual practice, we chose a NeurIPS paper from 10 years ago to receive the Test of Time Award, and "**Distributed Representations of Words and Phrases and their Compositionality**" by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, won.

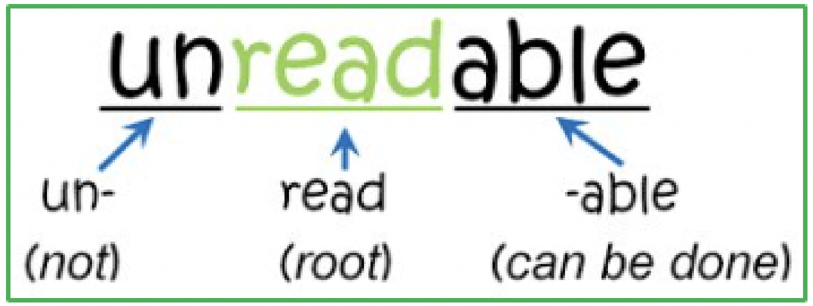


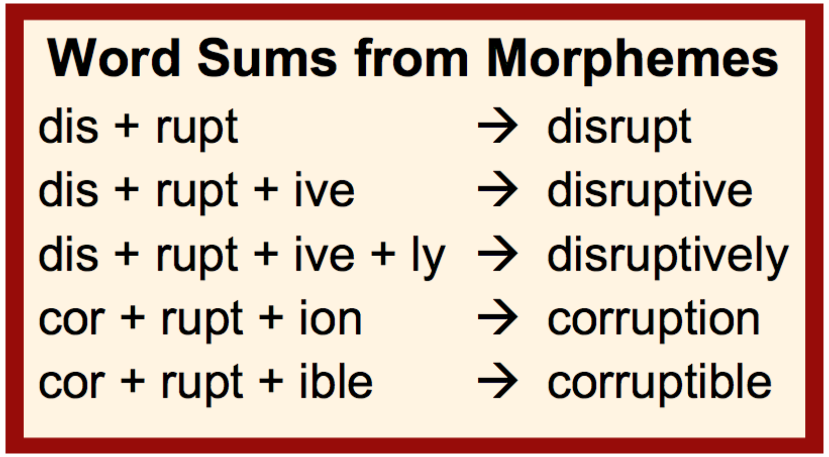
Tomas Mikolov's Post

There was also significant controversy around the GloVe project from Stanford NLP group that was published more than a year after word2vec. While it copied many tricks from our project, GloVe always felt like a step back to me: it was slower, required more memory, and the resulting vectors had lower quality than the original word2vec. However, it was published with word vectors pre-trained on much more data and thus gained a lot of popularity - although the comparison was really apples-to-oranges. We anyways did fix this later in the fastText project, where we did show that word2vec is much better than GloVe when trained on the same data.

fastText [Bojanowski et al., TACL 2017]

- Motivation: treating each word as a whole ignores the internal structure of words
- Solution: representing words with character N-grams
- Example (assume character trigrams):
  - The word "*where*" will be decomposed into: <*wh*, *whe*, *her*, *ere*, *re*>
  - The word "*her*" will be represented as <*her*>
- Each word is represented by the sum of the vectors of its character N-grams
- Use the same training objective as word2vec
- Benefit: more robust representations for rare words



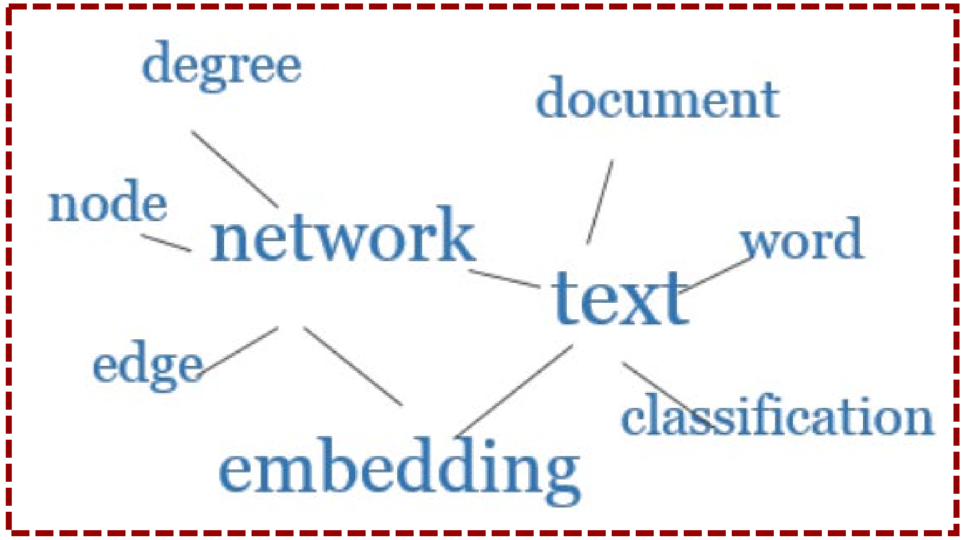

Word Sums from Morphemes

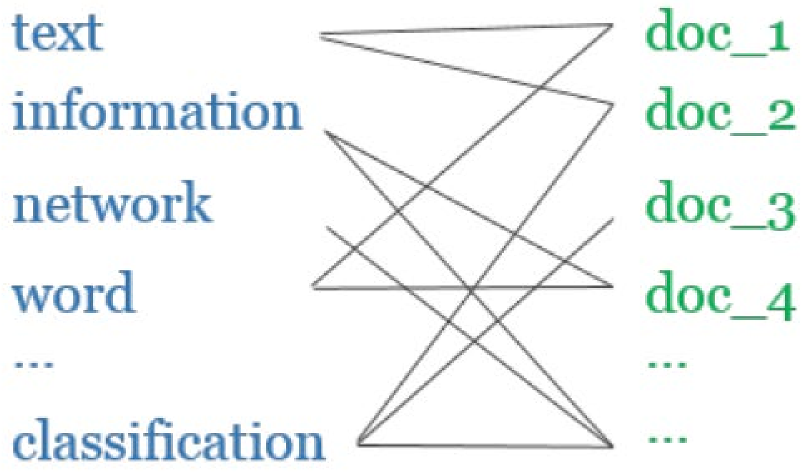
dis + rupt	→ disrupt
dis + rupt + ive	→ disruptive
dis + rupt + ive + ly	→ disruptively
cor + rupt + ion	→ corruption
cor + rupt + ible	→ corruptible

PTE [Tang et al., KDD 2015]

- Consider word embedding from the graph perspective
    - Each word is represented as a node, and an edge is added between two words if they co-occur within a context window.

$$\max \sum_{(w,x)\in\mathcal{E}} \frac{\exp(\boldsymbol{e}_w^T \tilde{\boldsymbol{e}}_x)}{\sum_{w'\in\mathcal{V}} \exp(\boldsymbol{e}_{w'}^T \tilde{\boldsymbol{e}}_x)}$$

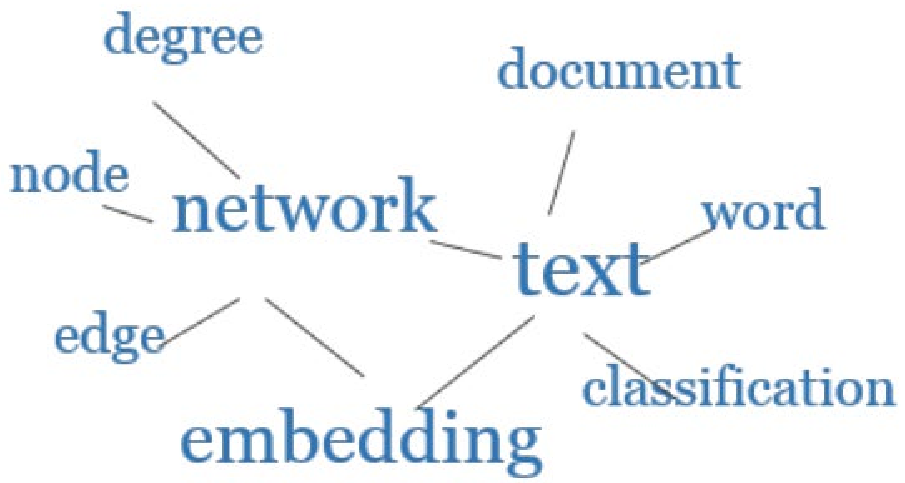


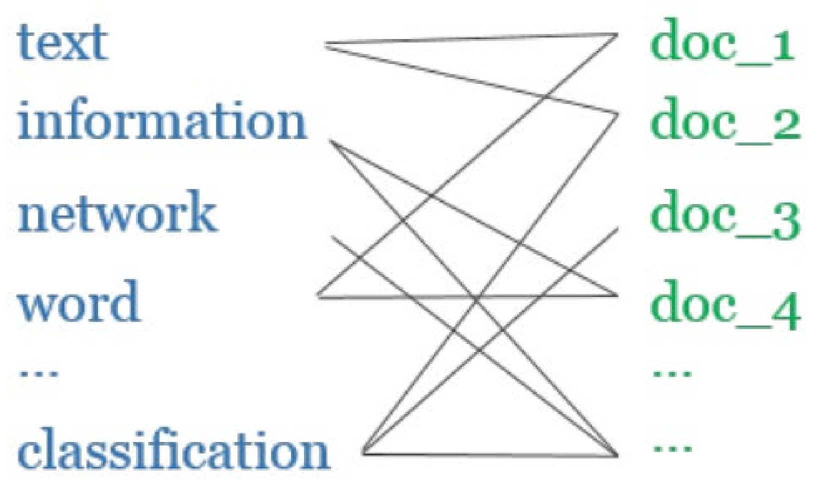


PTE [Tang et al., KDD 2015]

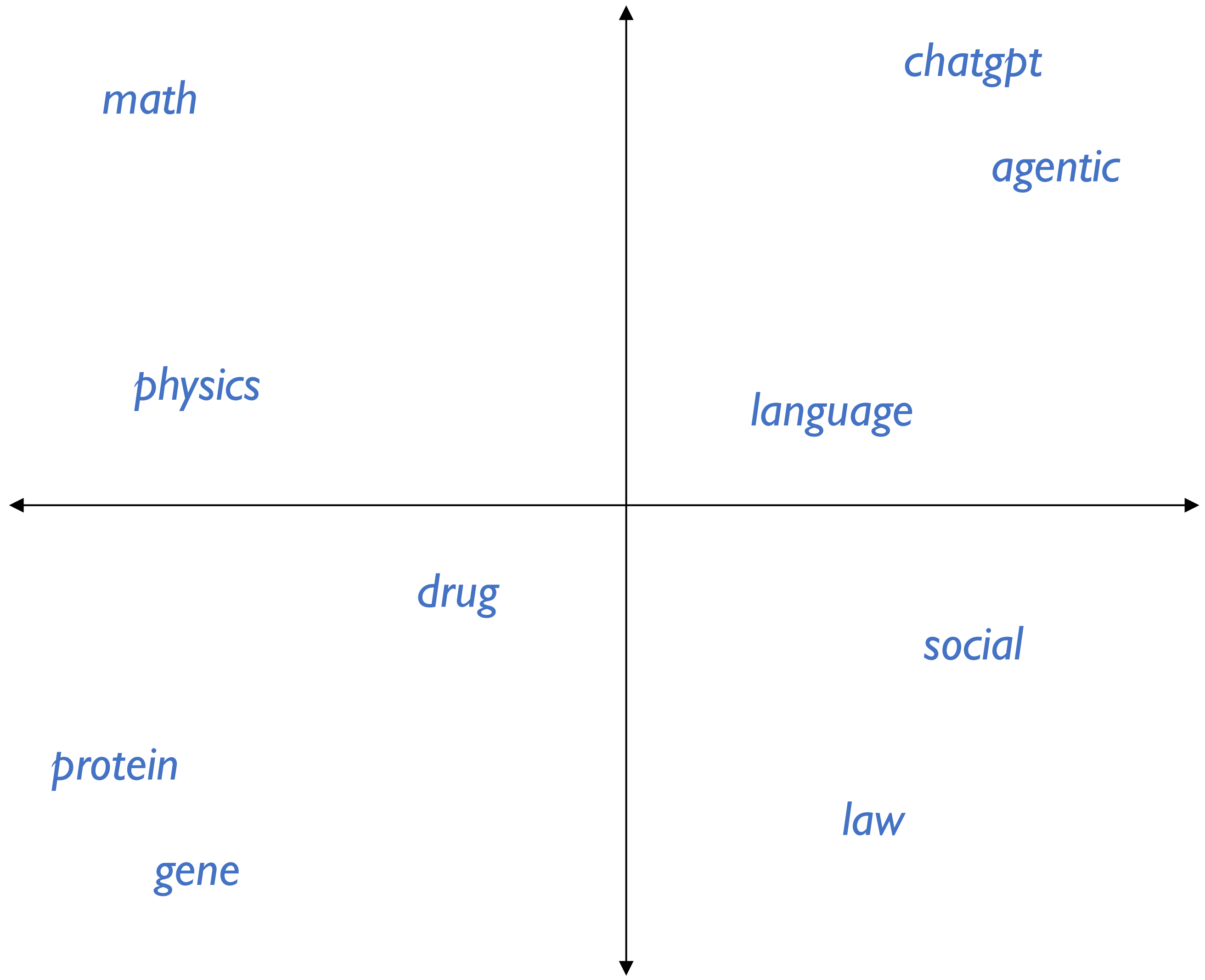
- Consider word embedding from the graph perspective
  - Each document is also represented as a node, and an edge is added between a word and a document if the word occurs in that document.

$$\max \sum_{(w,x)\in\mathcal{E}} \frac{\exp(\boldsymbol{e}_w^T \tilde{\boldsymbol{e}}_x)}{\sum_{w'\in\mathcal{V}} \exp(\boldsymbol{e}_{w'}^T \tilde{\boldsymbol{e}}_x)} + \sum_{(w,d)\in\mathcal{E}} \frac{\exp(\boldsymbol{e}_w^T \boldsymbol{e}_d)}{\sum_{w'\in\mathcal{V}} \exp(\boldsymbol{e}_{w'}^T \boldsymbol{e}_d)}$$

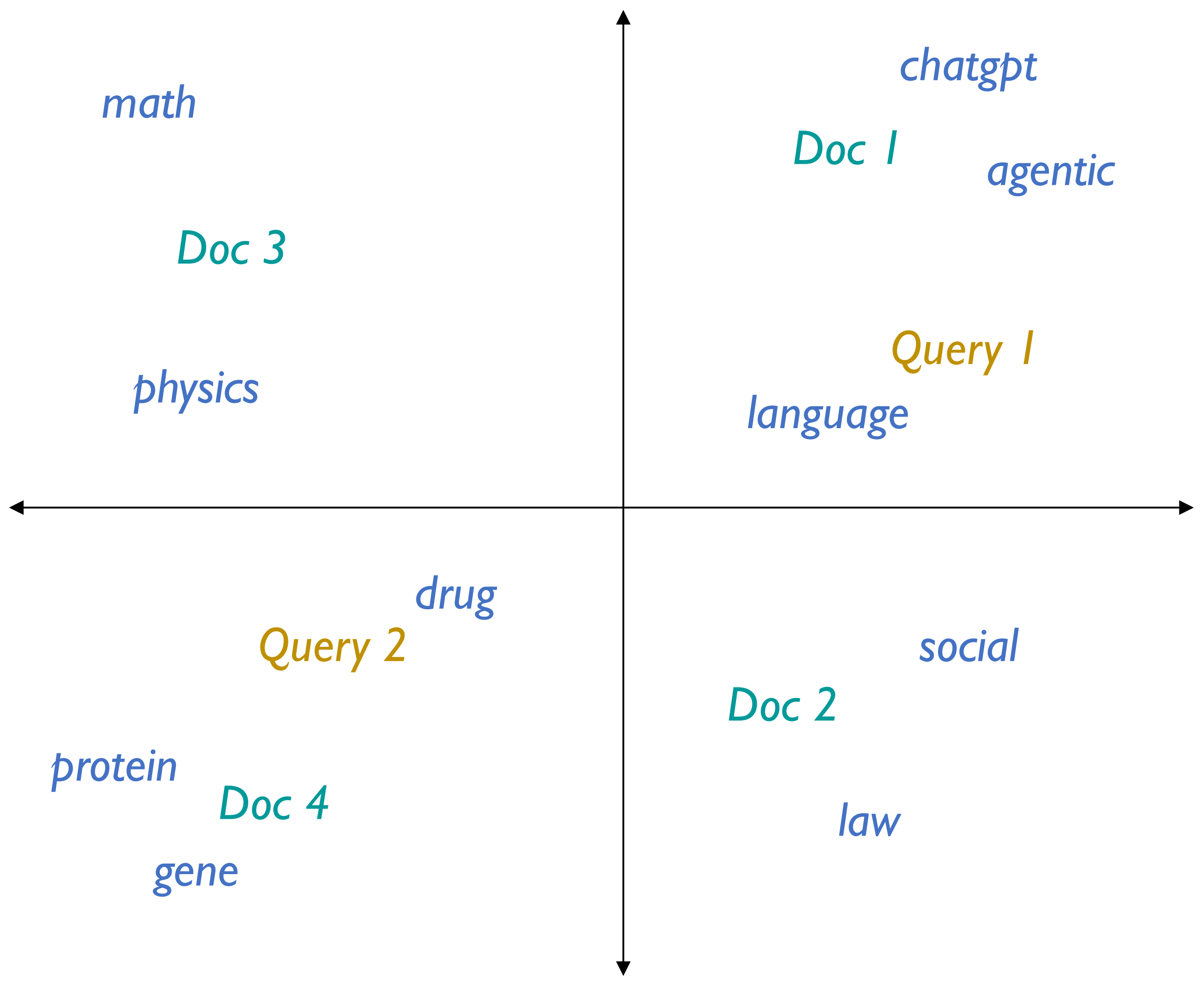




Summary: We have done this!



Next Lecture



Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-F25.html