CSCE 670 - Information Storage and Retrieval

Week 8: word2vec, Neural Ranking

Yu Zhang

yuzhang@tamu.edu

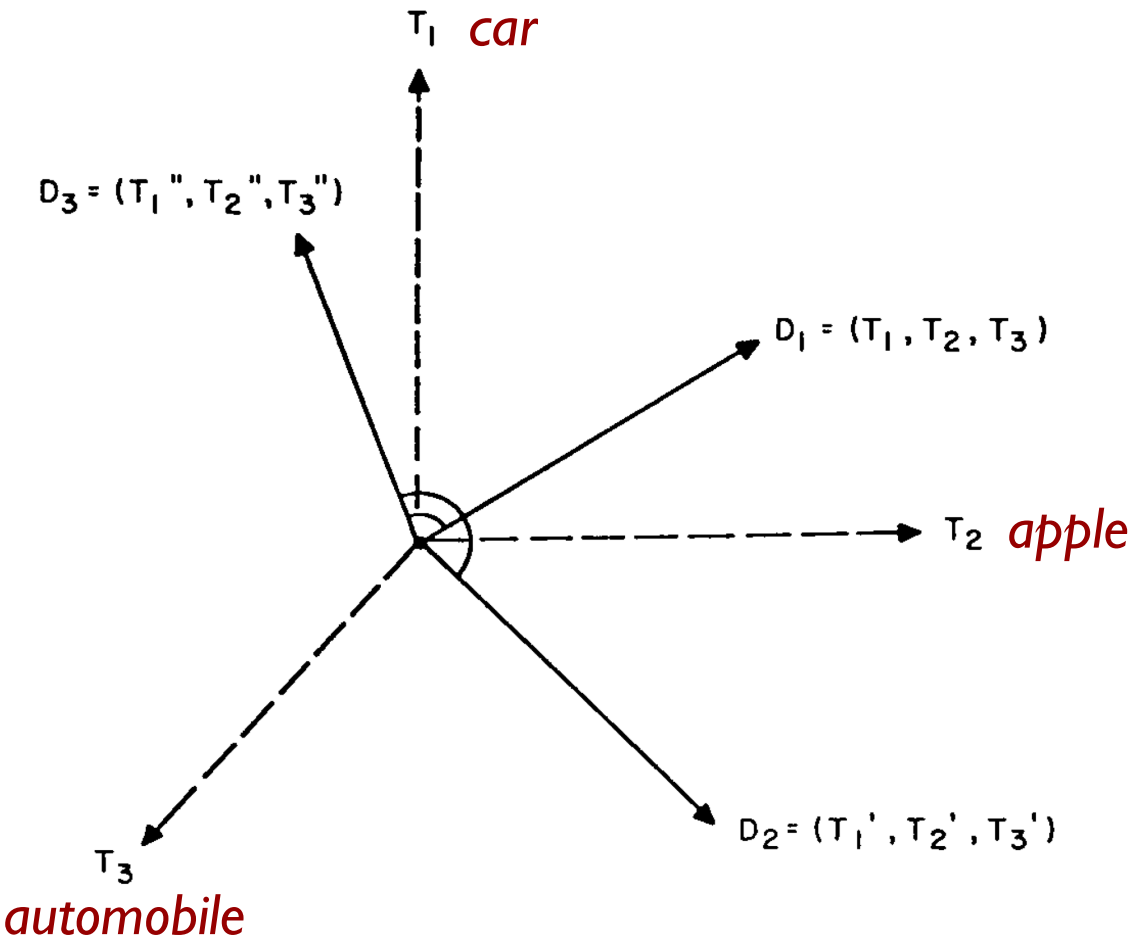
Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html

Adapted from the slides by Prof. James Caverlee

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.



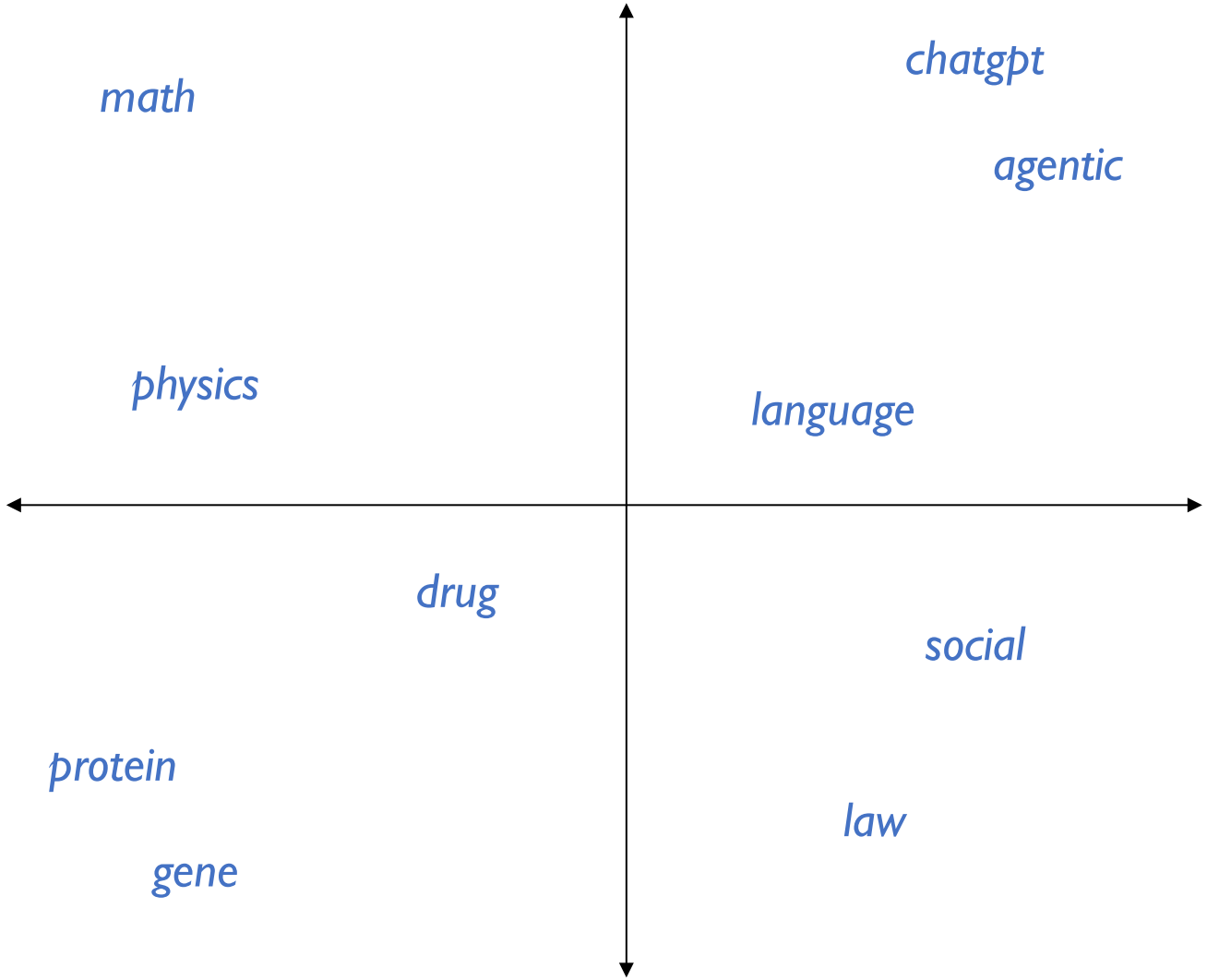
Recap: Limitations of "Count-based" Vectors

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

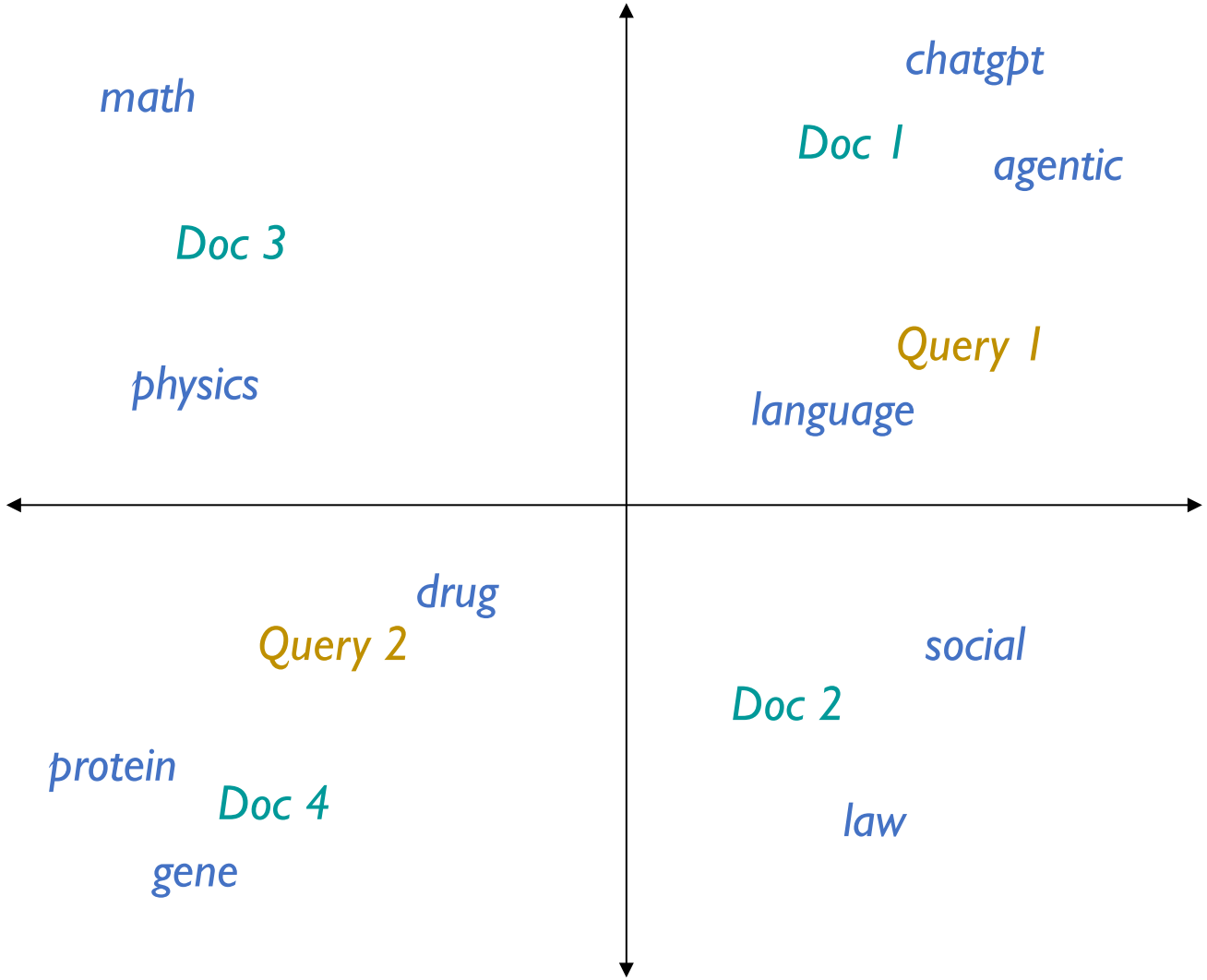
- Limitation 1: Doc 1 and Doc 2 are totally irrelevant according to Boolean, TF-IDF, and BM25.
  - Because we directly use the corresponding row to represent each document.
  - The inner product or cosine similarity between them is 0.
- Limitation 2: Vectors are sparse (with lots of zeros)
  - Zero values 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

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

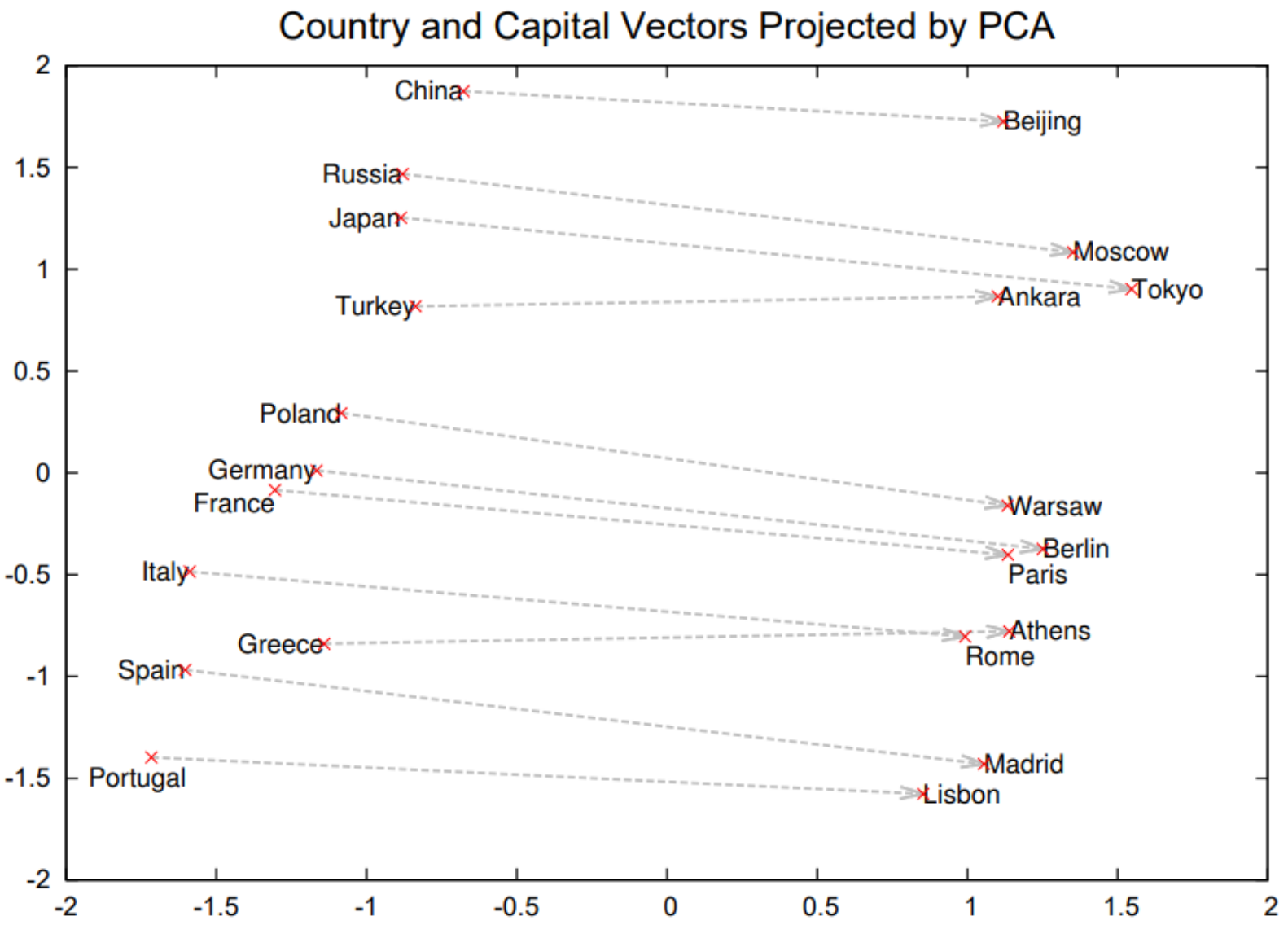
Distributed representations of **words** and **phrases** and their compositionality [PDF] neurips.cc

T Mikolov, I Sutskever, K Chen… - Advances in neural …, 2013 - proceedings.neurips.cc

… We show how to train **distributed representations** of **words** and **phrases** with the Skip-gram model and demonstrate that these **representations** exhibit linear structure that makes precise …

☆ Save 99 Cite Cited by 49518 Related articles All 46 versions »

2-dimensional Visualization of 1000-dimensional word2vec Vectors



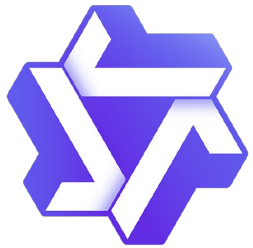
Idea: Distributional Hypothesis [Harris, 1954]

- 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: 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*
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 e_w of all words w in the vocabulary $\mathcal{V}$

w	*abandon*	…	*chatgpt*	*learning*	*openai*	*proposes*	…	*zucchini*
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

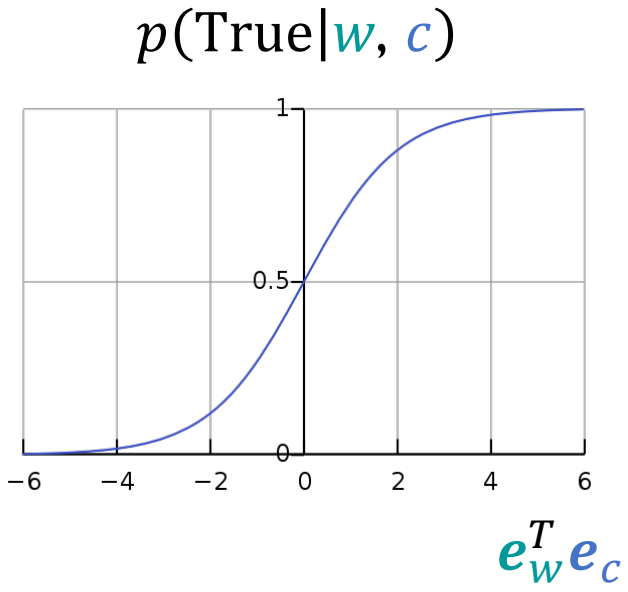
More Details about the Learning Task

- What is the learning objective?

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

  - 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.

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



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.

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

real word-context pair

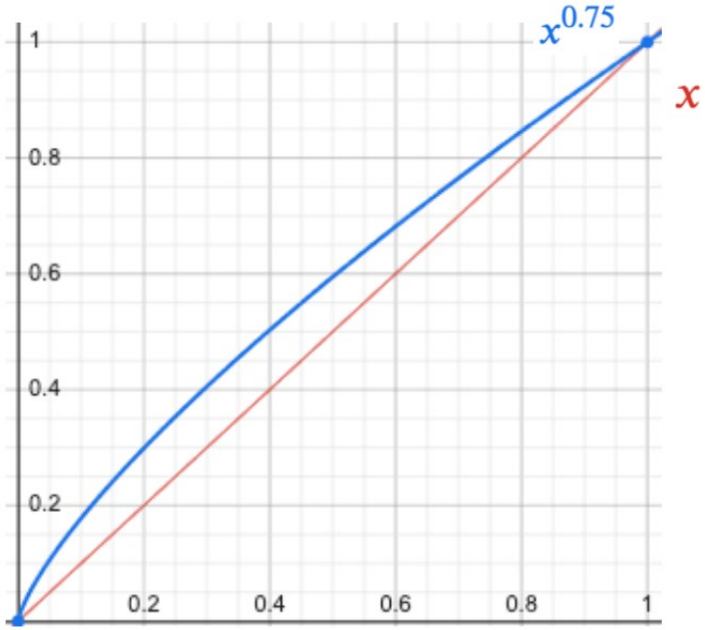
negative word-context pair

More Details about the Learning Task

- What is the learning objective?

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

  - 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?

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

  - 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

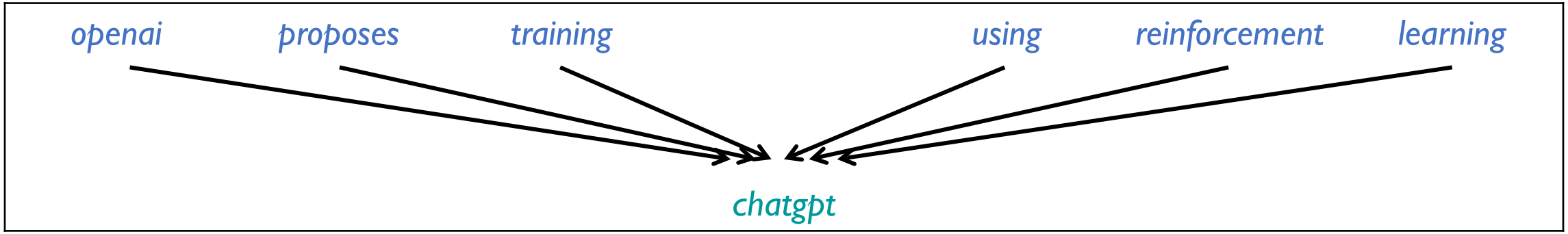
- Each word w may appear multiple times in the corpus.
- Each time, w may have a different context c.
- Our objective requires that w be close not just to one context, but to all the contexts in the corpus.

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

Questions?

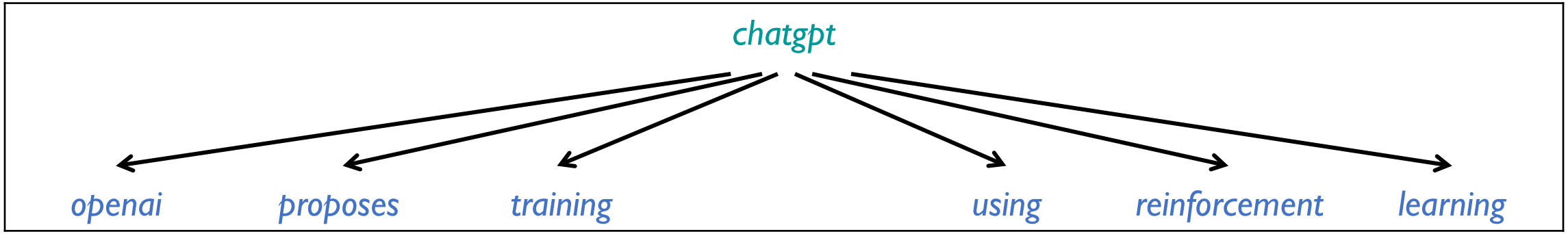
Continuous Bag-of-Words (CBOW)

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



Skip-Gram

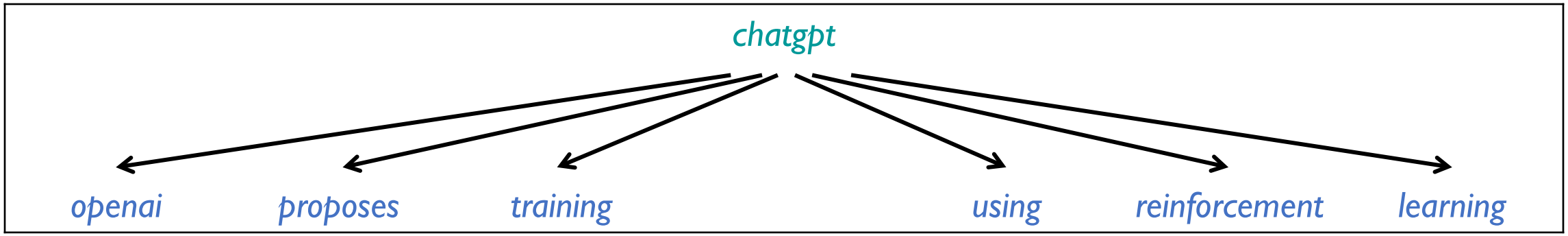
- word2vec has two variants
- The other one is Skip-Gram
  - Idea: using word to predict context



- 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



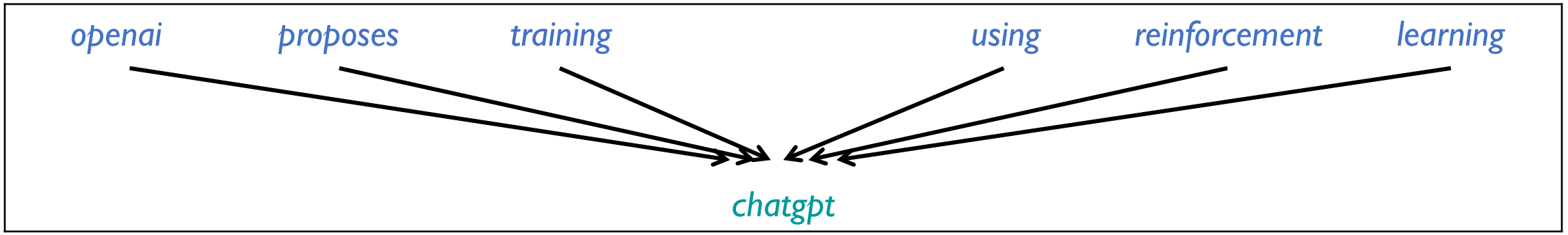
- Learning objective:

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

$\text{Context}(w)$: $\pm k$ 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

Copy code

```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

- [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

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

- 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 $\left[\log\left(\mathrm{PMI}(w,x)\right) - \log b\right]_{w\in\mathcal{V},x\in\mathcal{V}}$

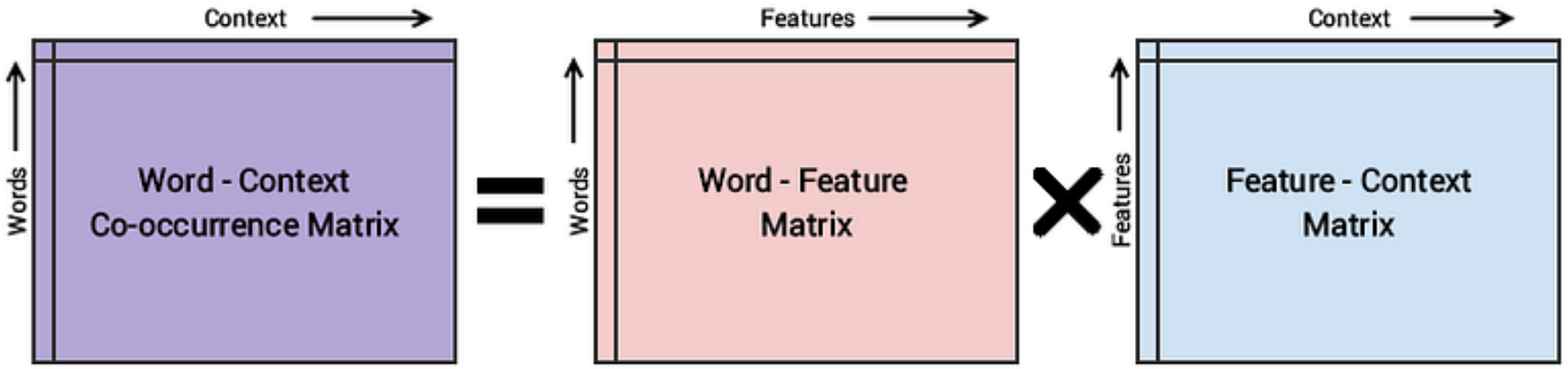
- Note 2: What if we explicitly factorize this matrix?
  - Better than word2vec [Levy and Goldberg, NIPS 2014]
  - But much slower!

Other Word Embedding Techniques

GloVe [Pennington et al., EMNLP 2014]

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

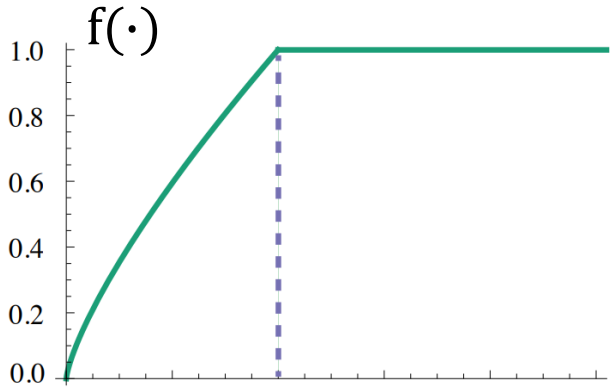
$$\log\left(\#(w,x)\right) = b_w + \tilde{b}_x + \boldsymbol{e}_w^T\tilde{\boldsymbol{e}}_x$$



- Learning objective:

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

$\mathrm{f}(\#(w,x))$: 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

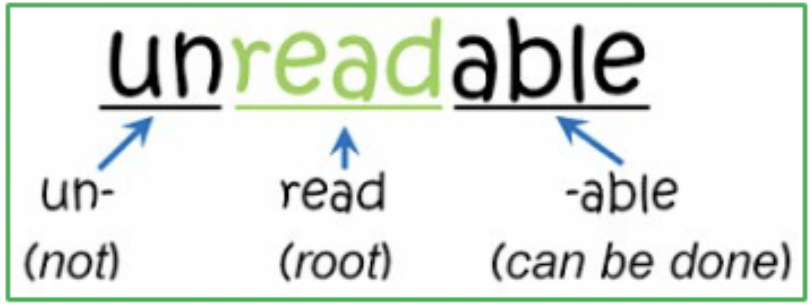
This year, following the usual practice, we chose a NeurIPS paper from 10 years ago to receive the Test of Time Award, and **"Distributed Representations of Words and Phrases and their Compositionality"** by Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean, won.

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



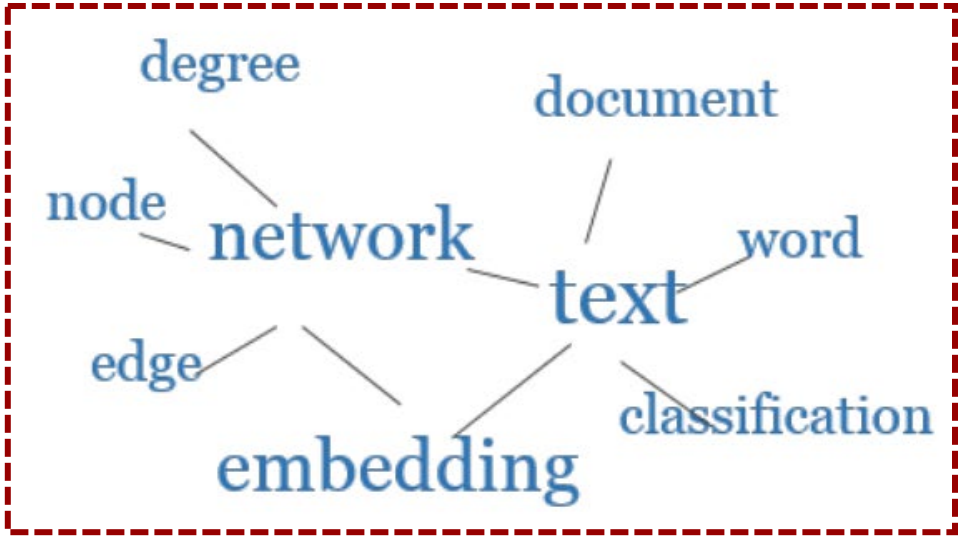
Word Sums from Morphemes

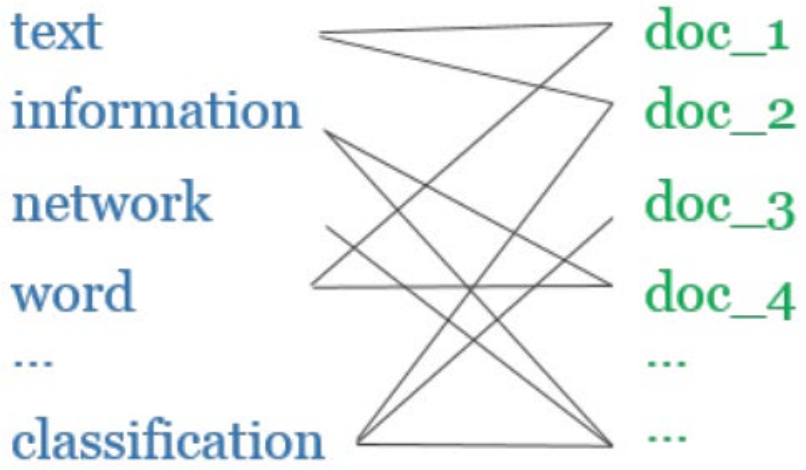
dis + rupt	→	disrupt
dis + rupt + ive	→	disruptive
dis + rupt + ive + ly	→	disruptively
cor + rupt + ion	→	corruption
cor + rupt + ible	→	corruptible

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.

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

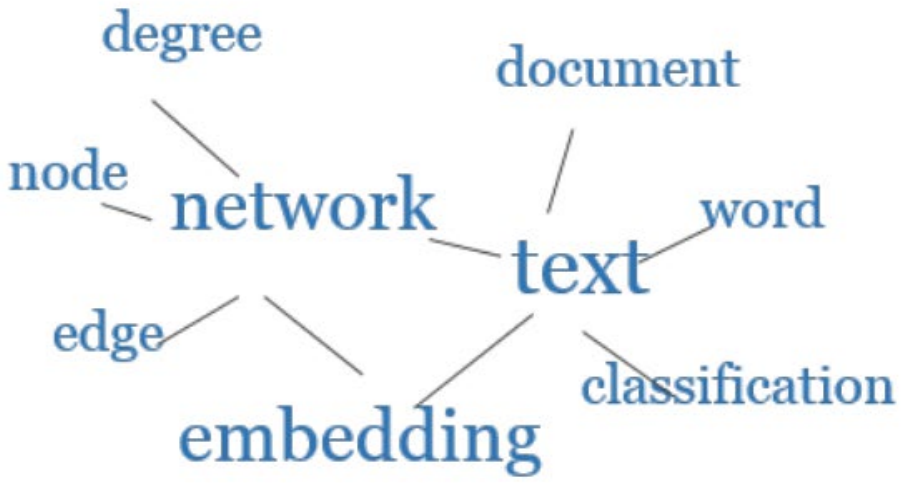


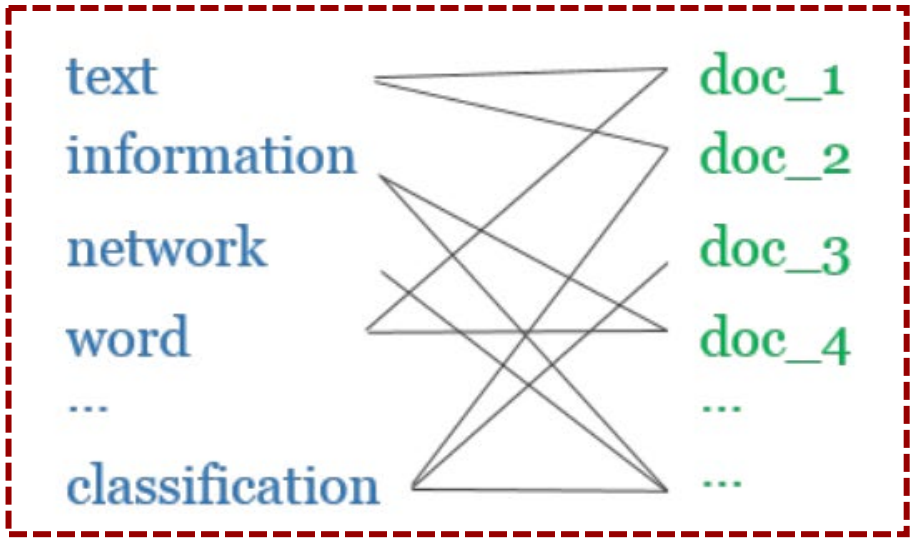


PTE [Tang et al., KDD 2015]

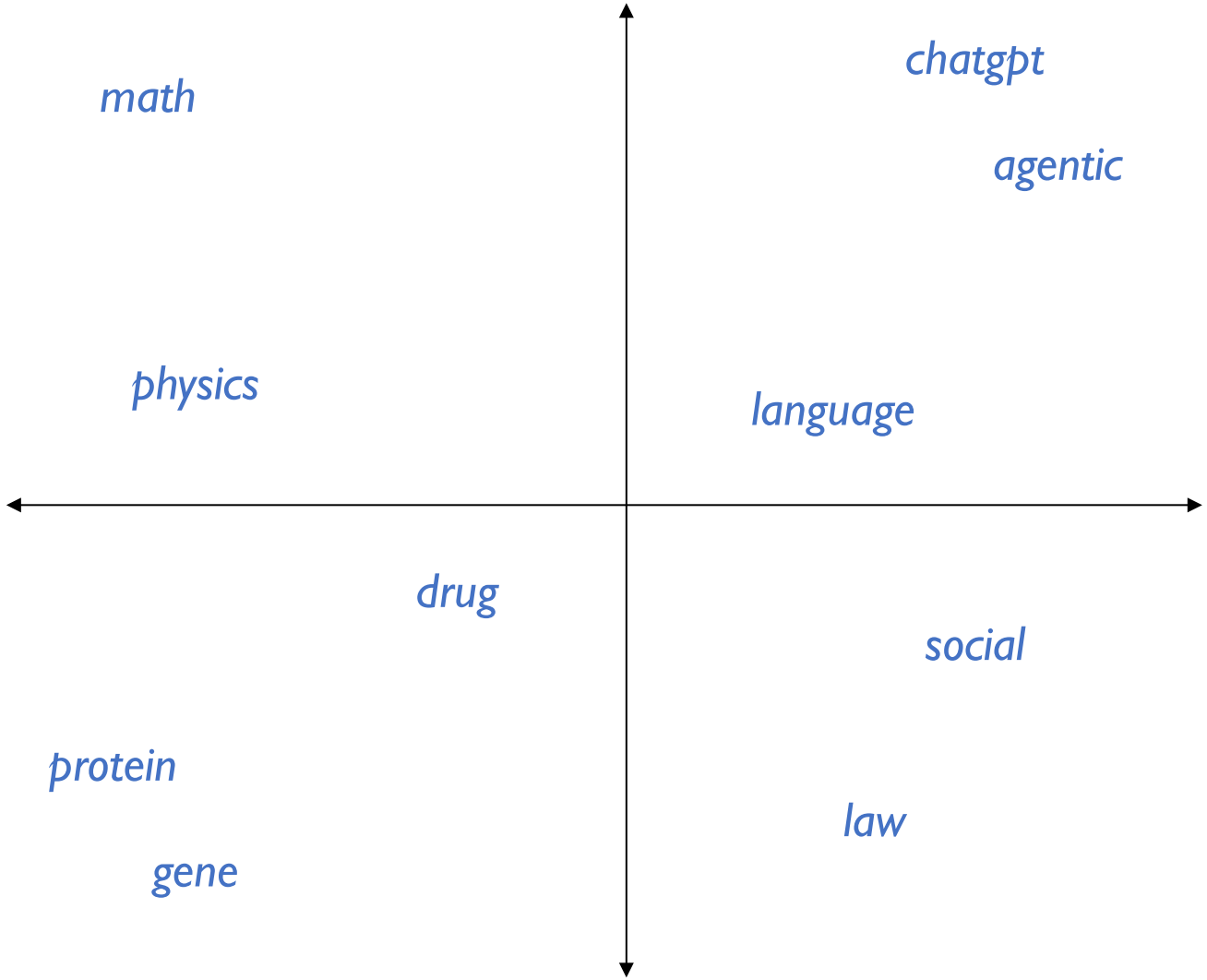
- Consider word embedding from the graph perspective
  - Each document is also represented as a node, and an edge is added between a word and a document if the word occurs in that document.

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

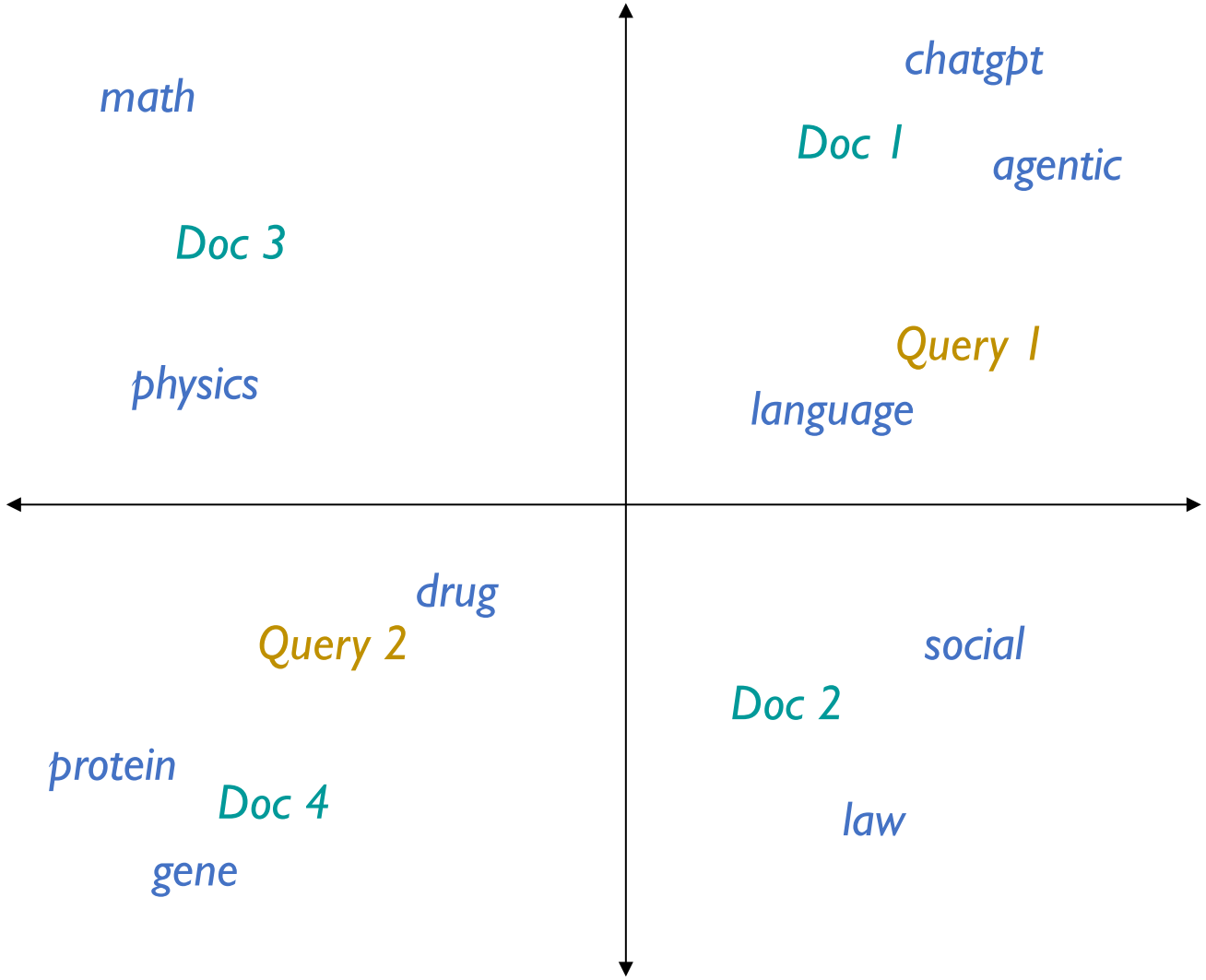




Summary: We have done this!



But how do we achieve this?



Neural Ranking with Word Embeddings

Simplest Solution: Taking the Average/Sum

- Query q:"*large language models*"

$$\boldsymbol{e}_q = \frac{1}{|q|}\sum_{w \in q} \boldsymbol{e}_w = \frac{1}{3}\left(\boldsymbol{e}_{\text{large}} + \boldsymbol{e}_{\text{language}} + \boldsymbol{e}_{\text{models}}\right)$$

- Document d:"*openai proposes training chatgpt using reinforcement learning*"

$$\boldsymbol{e}_d = \frac{1}{|d|}\sum_{w \in d} \boldsymbol{e}_w = \frac{1}{7}\left(\boldsymbol{e}_{\text{openai}} + \boldsymbol{e}_{\text{proposes}} + \boldsymbol{e}_{\text{training}} + \cdots\right)$$

- Skip out-of-vocabulary words (if there are any)
- Can also take the sum instead of the average

- $\text{score}(q, d) = \cos(\boldsymbol{e}_q, \boldsymbol{e}_d)$ or $\boldsymbol{e}_q^T \boldsymbol{e}_d$

DESM [Nalisnick et al., WWW 2016]

Improving Document Ranking with Dual Word Embeddings

Eric Nalisnick
University of California
Irvine, USA
enalisni@uci.edu

Bhaskar Mitra
Microsoft
Cambridge, UK
bmitra@microsoft.com

Nick Craswell, Rich Caruana
Microsoft
Redmond, USA
nickcr, rcaruana@microsoft.com

ABSTRACT

This paper investigates the popular neural word embedding method *Word2vec* as a source of evidence in document ranking. In contrast to NLP applications of word2vec, which tend to use only the input embeddings, we retain both the input and the output embeddings, allowing us to calculate a different word similarity that may be more suitable for document ranking. We map the query words into the *input* space and the document words into the *output* space, and compute a relevance score by aggregating the cosine similarities across all the query-document word pairs. We postulate that the proposed *Dual Embedding Space Model* (DESM) provides evidence that a document is *about* a query term, in addition to and complementing the traditional term frequency based approach.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: H.3.3 Information Search and Retrieval

Keywords: Document ranking; Word embeddings; Word2vec

Albuquerque is the most populous city in the U.S. state of New Mexico. The high-altitude city serves as the county seat of Bernalillo County, and it is situated in the central part of the state, straddling the Rio Grande. The city population is 557,169 as of the July 1, 2014, population estimate from the United States Census Bureau, and ranks as the 32nd-largest city in the U.S. The Metropolitan Statistical Area (or MSA) has a population of 902,797 according to the United States Census Bureau's most recently available estimate for July 1, 2013.

(a)

Allen suggested that they could program a BASIC interpreter for the device; after a call from Gates claiming to have a working interpreter, MITS requested a demonstration. Since they didn't actually have one, Allen worked on a simulator for the Altair while Gates developed the interpreter. Although they developed the interpreter on a simulator and not the actual device, the interpreter worked flawlessly when they demonstrated the interpreter to MITS in Albuquerque, New Mexico in March 1975; MITS agreed to distribute it, marketing it as Altair BASIC.

(b)

Dual Embedding Space Model

- Idea: Each word w can have two embeddings
  - $\boldsymbol{e}_w$: when w serves as the "word" in embedding learning
  - $\tilde{\boldsymbol{e}}_w$: when w serves as the "context" in embedding learning
- The learning objective of Skip-Gram becomes

$$\max_{\{\boldsymbol{e}_v, \tilde{\boldsymbol{e}}_v | v \in \mathcal{V}\}} \sum_{d \in D} \sum_{w \in d} \sum_{x \in \text{Context}(w)} \frac{\exp(\boldsymbol{e}_w^T \tilde{\boldsymbol{e}}_x)}{\sum_{v \in \mathcal{V}} \exp(\boldsymbol{e}_w^T \tilde{\boldsymbol{e}}_v)}$$

- Analogy:
  - If you factorize a user-item matrix, you get user embeddings and item embeddings in the same space
  - If you factorize a word-word matrix, you get word embeddings (as the word) and word embeddings (as the context) in the same space

How to use Dual Embeddings?

- Strategy 1 ("IN-IN"): Discard $\tilde{e}_w$
  - score$(w, x) = e_w^T e_x$
- Strategy 2 ("IN-OUT"): Keep both e_w and $\tilde{e}_w$
  - score$(w, x) = e_w^T \tilde{e}_x$

yale		**seahawks**		**eminem**	
IN-IN	IN-OUT	IN-IN	IN-OUT	IN-IN	IN-OUT
yale	yale	seahawks	seahawks	eminem	eminem
harvard	faculty	49ers	highlights	rihanna	rap
nyu	alumni	broncos	jerseys	ludacris	featuring
cornell	orientation	packers	tshirts	kanye	tracklist
tulane	haven	nfl	seattle	beyonce	diss
tufts	graduate	steelers	hats	2pac	performs

Which strategy is better?

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77
LSA	22.41*	28.25*	44.24*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*
DESM (IN-OUT, trained on queries)	**25.02***	**31.14***	**47.89***

- Why?
  - Assume the query is "*yale*". Which of the following sentence is more relevant?
  - "*He has collaborated with researchers from Yale, Harvard, NYU, and Cornell on several interdisciplinary projects.*"
  - "*Many Yale faculty members are proud alumni who continue to mentor current graduate students.*"

Similarity Calculation in DESM

- Precompute document embedding

$$\tilde{\boldsymbol{e}}_d = \frac{1}{|d|} \sum_{w \in d} \frac{\tilde{\boldsymbol{e}}_w}{\|\tilde{\boldsymbol{e}}_w\|}$$

- Average the similarity between each query word and the document

$$\text{score}(q, d) = \frac{1}{|q|} \sum_{w \in q} \cos(\boldsymbol{e}_w, \tilde{\boldsymbol{e}}_d)$$

- Any more complicated solutions?

Experiments

- Train word2vec (CBOW) from either
  - 600 million Bing queries
  - 342 million web document sentences
- Test on 7,741 randomly sampled Bing queries
  - 5-level evaluation (Perfect, Excellent, Good, Fair, Bad)
- Two approaches
  - Use DESM model to rerank top results from BM25
  - Use DESM alone or a mixture model of it and BM25

Ranking All Documents

- DESM alone performs poorly
- Even BM25 + DESM does not significantly outperform BM25
  - Recall the "Neural Hype" mentioned in the lecture of BM25

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	21.44	26.09	37.53
LSA	04.61*	04.63*	04.83*
DESM (IN-IN, trained on body text)	06.69*	06.80*	07.39*
DESM (IN-IN, trained on queries)	05.56*	05.59*	06.03*
DESM (IN-OUT, trained on body text)	01.01*	01.16*	01.58*
DESM (IN-OUT, trained on queries)	00.62*	00.58*	00.81*
BM25 + DESM (IN-IN, trained on body text)	21.53	26.16	37.48
BM25 + DESM (IN-IN, trained on queries)	**21.58**	26.20	37.62
BM25 + DESM (IN-OUT, trained on body text)	21.47	26.18	37.55
BM25 + DESM (IN-OUT, trained on queries)	21.54	**26.42***	**37.86***

Reranking Top-k Results by BM25

- Better at reranking somewhat relevant documents by judging their fine-grained relevance to the query

	Explicitly Judged Test Set		
	NDCG@1	NDCG@3	NDCG@10
BM25	23.69	29.14	44.77
LSA	22.41*	28.25*	44.24*
DESM (IN-IN, trained on body text)	23.59	29.59	45.51*
DESM (IN-IN, trained on queries)	23.75	29.72	46.36*
DESM (IN-OUT, trained on body text)	24.06	30.32*	46.57*
DESM (IN-OUT, trained on queries)	**25.02***	**31.14***	**47.89***

Questions?

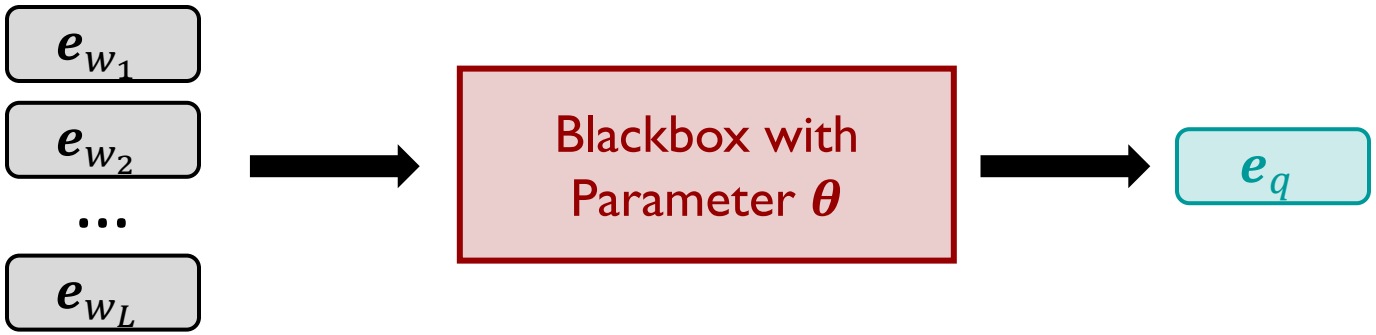
Learning to Rank with Word Embeddings

- DESM does not use any relevant (q, d) pairs to train the ranker!
  - Self-supervised word embedding learning + similarity computation (without any learnable parameters)

- What we have some relevant (q, d) pairs to perform training (i.e., learning to rank)?
  - Parameters?
  - Learning objective?
  - Optimization technique? – Will not go into detail here. You can think of it as gradient descent, although in reality it involves more sophisticated tricks

Learning to Rank with Word Embeddings

- Parameters: How to aggregate word embeddings to query / document embedding?
  - $\boldsymbol{e}_q = \mathrm{f}_{\boldsymbol{\theta}}(\{\boldsymbol{e}_w | w \in q\}),\ \boldsymbol{e}_d = \mathrm{f}_{\boldsymbol{\theta}}(\{\tilde{\boldsymbol{e}}_w | w \in d\})$
  - $\boldsymbol{\theta}$ denotes all parameters of the model



  - A simple (but overly ideal) example:
    - Assume all queries and documents have 10 words (e.g., q is $w_1 w_2 \ldots w_{10}$)
    - $\boldsymbol{e}_q = \sum_{i=1}^{10} \theta_i \boldsymbol{e}_{w_i}$

Learning to Rank with Word Embeddings

- Parameters: How to aggregate word embeddings to query / document embedding?
  - In reality:

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

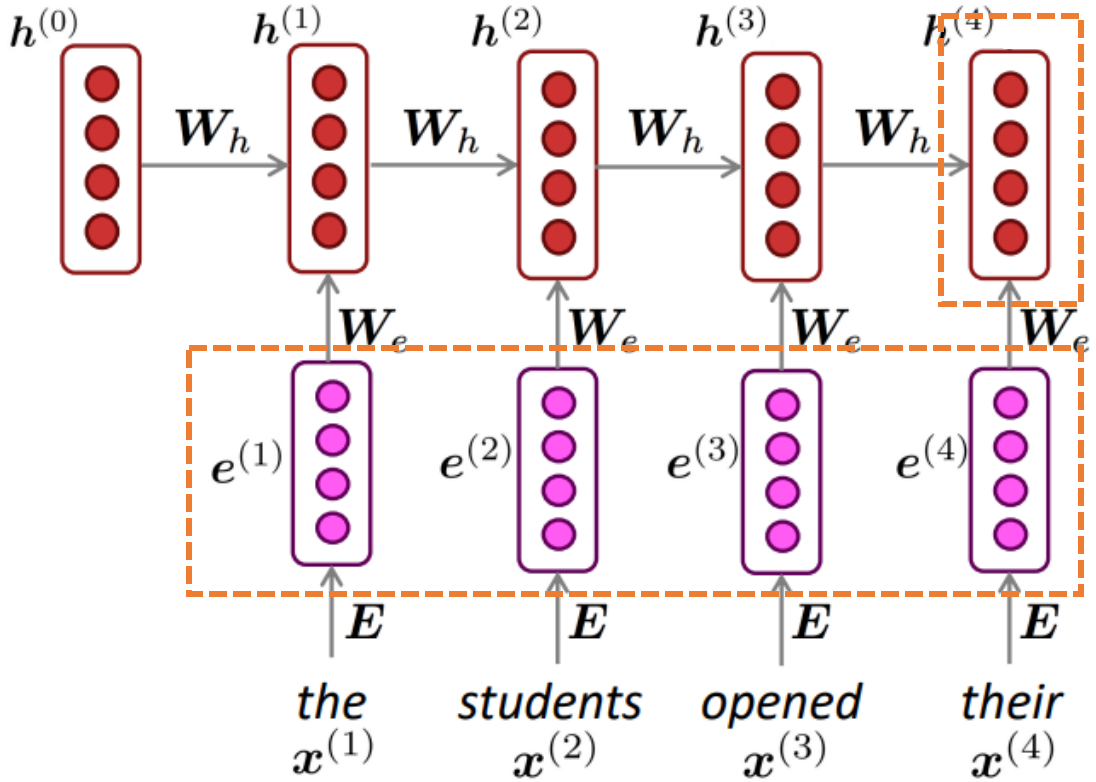
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



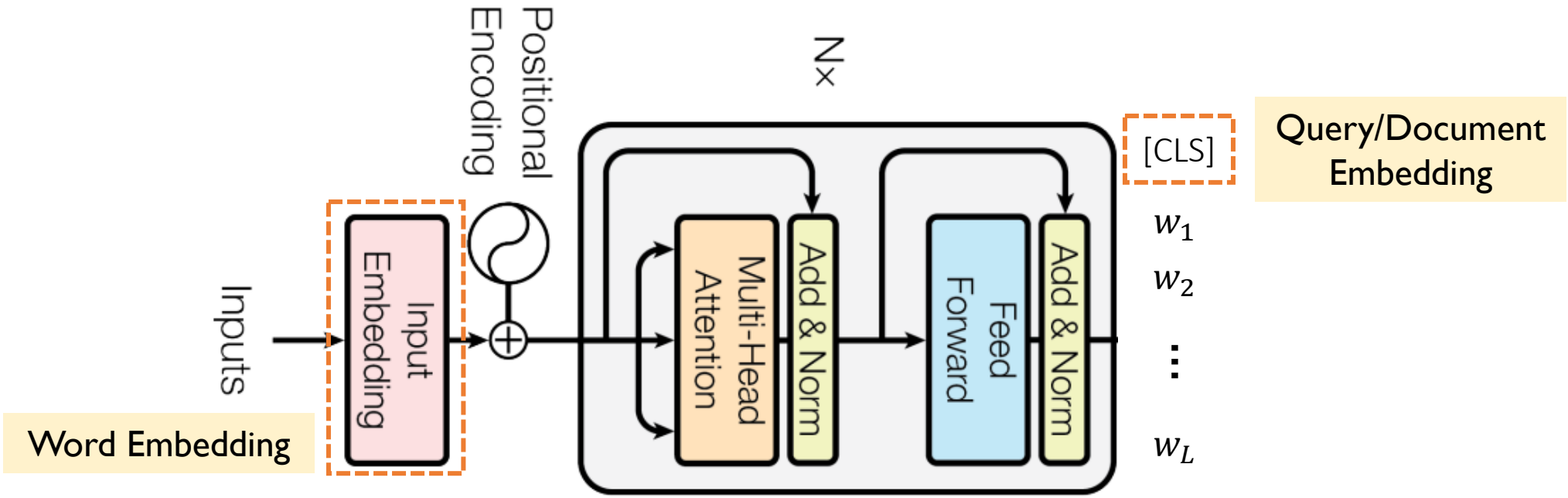
Query/Document Embedding

Word Embedding

Recurrent Neural Network: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf

Learning to Rank with Word Embeddings

- Parameters: How to aggregate word embeddings to query / document embedding?
  - In reality:

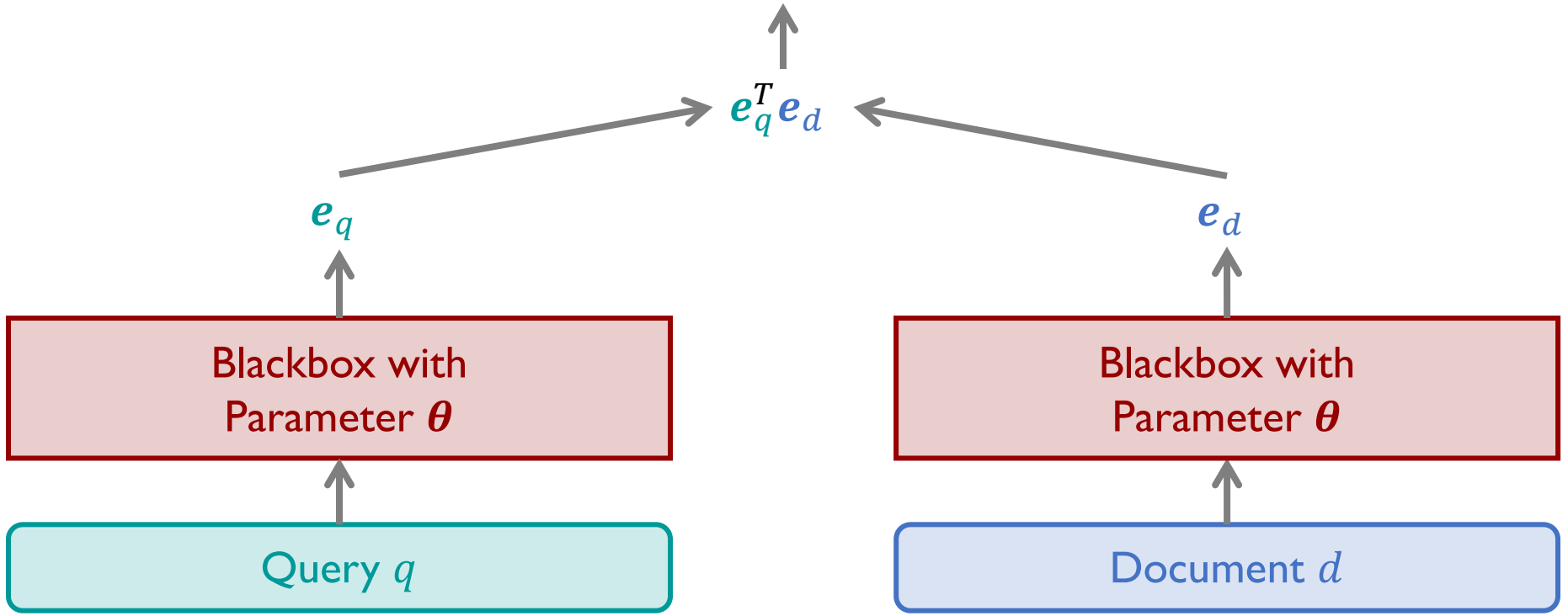


Transformer Encoder / BERT: https://arxiv.org/pdf/1706.03762

Learning to Rank with Word Embeddings

- Learning objective: A simple strategy is binary classification (i.e., whether q and d is relevant)

$$\hat{y}_{q,d} = p(\text{relevant}|q,d) = \text{Sigmoid}\left(\boldsymbol{e}_q^T \boldsymbol{e}_d\right) = \frac{1}{1+\exp(-\boldsymbol{e}_q^T \boldsymbol{e}_d)}$$



Learning to Rank with Word Embeddings

- Learning objective: A simple strategy is binary classification (i.e., whether q and d is relevant)

$$\hat{y}_{q,d} = p(\text{relevant}|q,d) = \text{Sigmoid}\left(\boldsymbol{e}_q^T \boldsymbol{e}_d\right) = \frac{1}{1+\exp(-\boldsymbol{e}_q^T \boldsymbol{e}_d)}$$

  - If q and d are relevant, then the ground truth $y_{q,d} = 1$
  - If q and d are irrelevant, then the ground truth $y_{q,d} = 0$
  - Cross-entropy loss: $\mathcal{L} = -y_{q,d}\log\hat{y}_{q,d} - \left(1-y_{q,d}\right)\log(1-\hat{y}_{q,d})$

  - Pulling relevant $\boldsymbol{e}_q$ and $\boldsymbol{e}_d$ closer
  - Pushing irrelevant $\boldsymbol{e}_q$ and $\boldsymbol{e}_d$ apart

  - Is this strategy pointwise or pairwise learning to rank?
  - What if we want to perform pairwise learning to rank?

Dense Passage Retrieval [Karpukhin et al., EMNLP 2020]

Dense Passage Retrieval for Open-Domain Question Answering

Vladimir Karpukhin[*], **Barlas Oğuz**[*], **Sewon Min**[†], **Patrick Lewis**,
Ledell Wu, Sergey Edunov, Danqi Chen[‡], **Wen-tau Yih**
Facebook AI [†]University of Washington [‡]Princeton University
{vladk, barlaso, plewis, ledell, edunov, scottyih}@fb.com
sewon@cs.washington.edu
danqic@cs.princeton.edu

Abstract

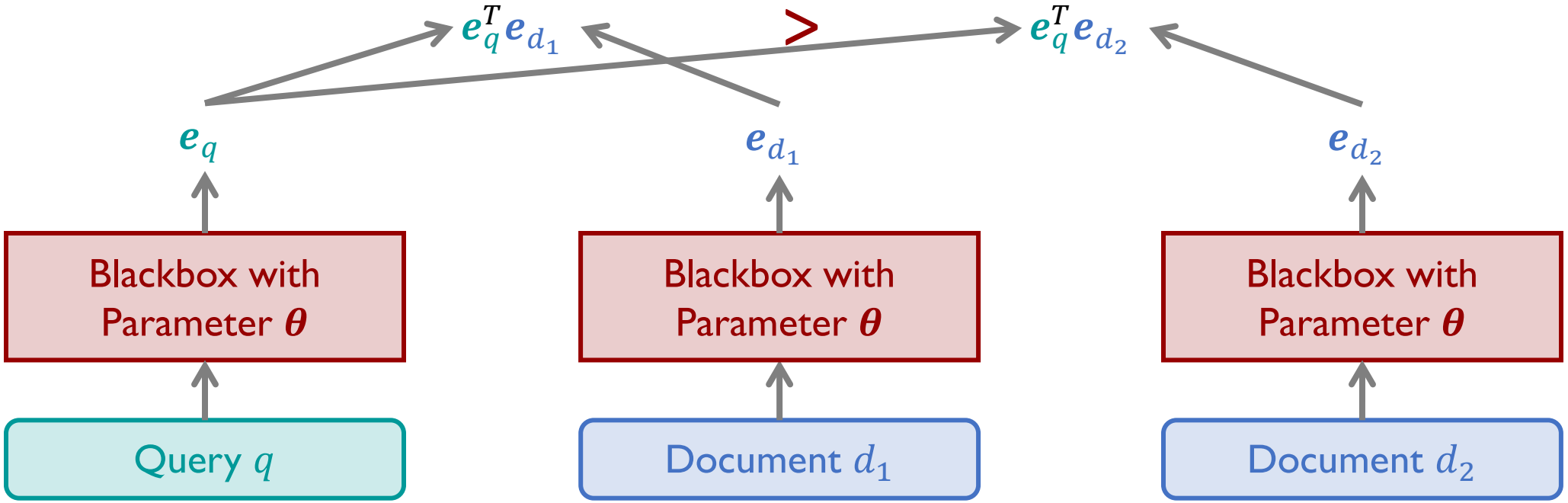
Open-domain question answering relies on efficient passage retrieval to select candidate contexts, where traditional sparse vector space models, such as TF-IDF or BM25, are the de facto method. In this work, we show that retrieval can be practically implemented us-

Retrieval in open-domain QA is usually implemented using TF-IDF or BM25 (Robertson and Zaragoza, 2009), which matches keywords efficiently with an inverted index and can be seen as representing the question and context in high-dimensional, sparse vectors (with weighting). Conversely, the *dense*, latent semantic encoding is *com-*

Learning to Rank with Word Embeddings

- Learning objective (Pairwise): Given a query q and two documents d_1 and d_2, the ground truth tells us that d_1 is more relevant to q than d_2

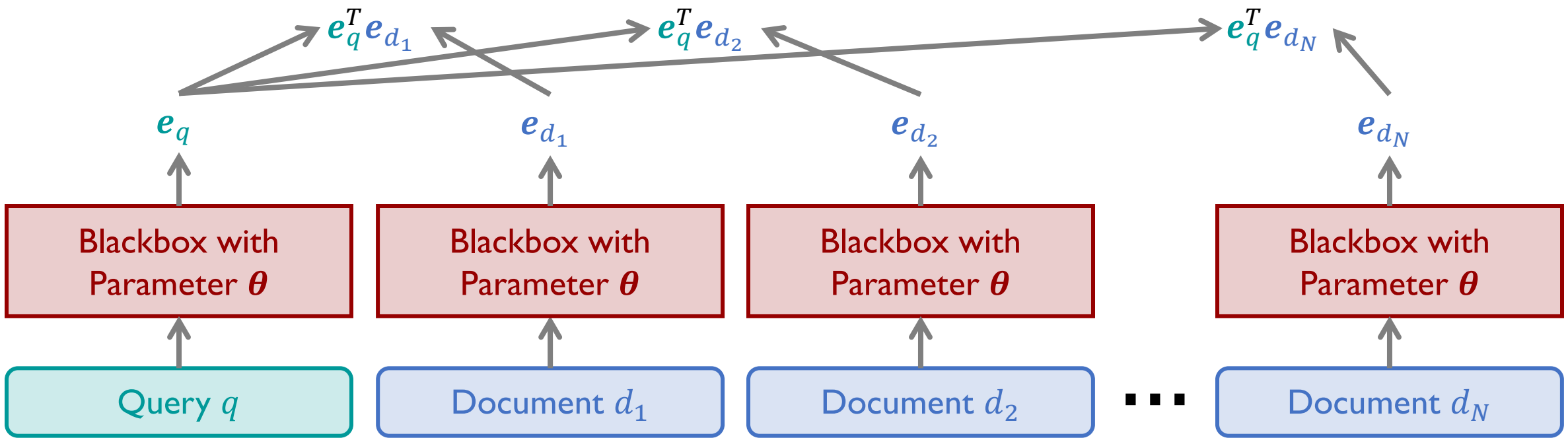
$$\text{Sigmoid}\left(\boldsymbol{e}_q^T\boldsymbol{e}_{d_1} - \boldsymbol{e}_q^T\boldsymbol{e}_{d_2}\right) = \frac{\exp(\boldsymbol{e}_q^T\boldsymbol{e}_{d_1})}{\exp\left(\boldsymbol{e}_q^T\boldsymbol{e}_{d_1}\right) + \exp(\boldsymbol{e}_q^T\boldsymbol{e}_{d_2})}$$



Learning to Rank with Word Embeddings

- Learning objective: Given a query q and N documents $(d_1, d_2, \ldots, d_N)$, the ground truth tells us that d_1 is the most relevant to q among these documents

$$\frac{\exp(\boldsymbol{e}_q^T \boldsymbol{e}_{d_1})}{\sum_{i=1}^{N} \exp(\boldsymbol{e}_q^T \boldsymbol{e}_{d_i})}$$



Contrastive Learning

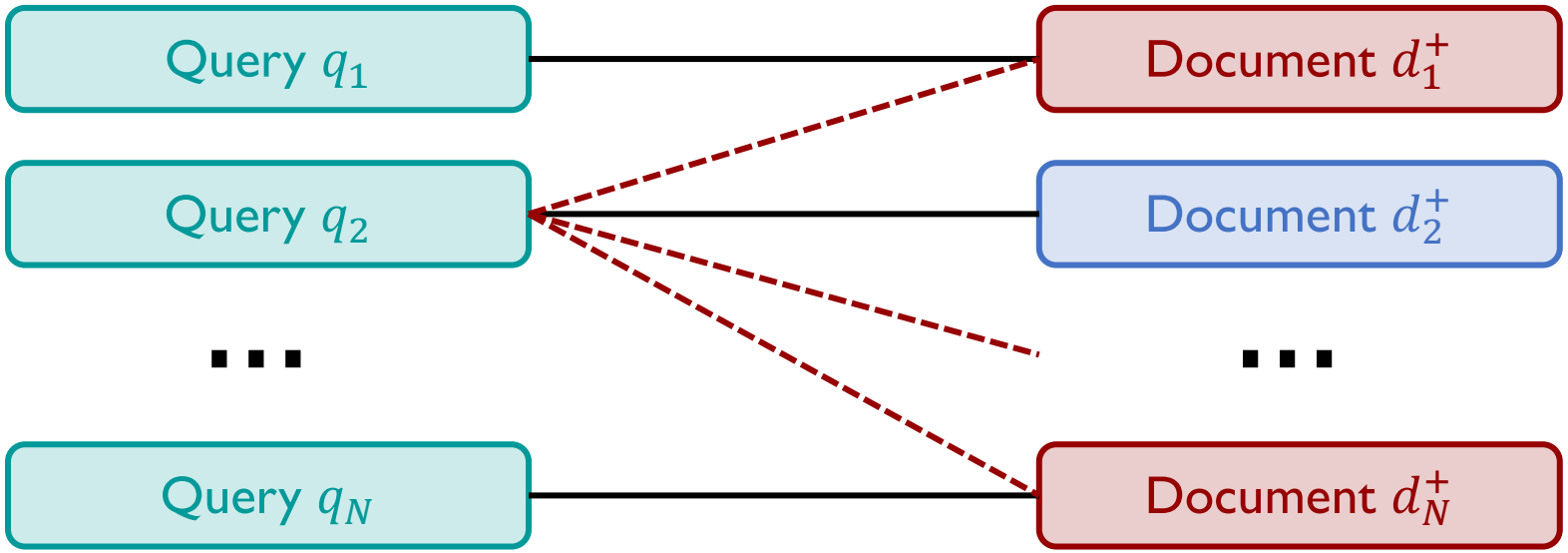
- A query q
- A positive sample d^+
- One or more negative samples $d_1^-, d_2^-, \ldots, d_{N-1}^-$
- Learning objective:

$$\max \frac{\exp\left(\text{score}(q, d^+)\right)}{\exp\left(\text{score}(q, d^+)\right) + \sum_{i=1}^{N-1} \exp\left(\text{score}(q, d_i^-)\right)}$$

- Different choices of $\text{score}(q, d)$
  - $\text{score}(q, d) = \boldsymbol{e}_q^T \boldsymbol{e}_d$
  - $\text{score}(q, d) = \frac{\cos(\boldsymbol{e}_q, \boldsymbol{e}_d)}{\tau}$
    - τ is a hyperparameter that is usually smaller than 1 (e.g., 0.05)

Contrastive Learning

- In-batch negative samples
  - Feed N (query, positive document) pairs together as a batch into GPU
  - No need to explicitly sample negative documents for each query
  - The positive documents of other queries in the same batch will be used as negative documents of the current query

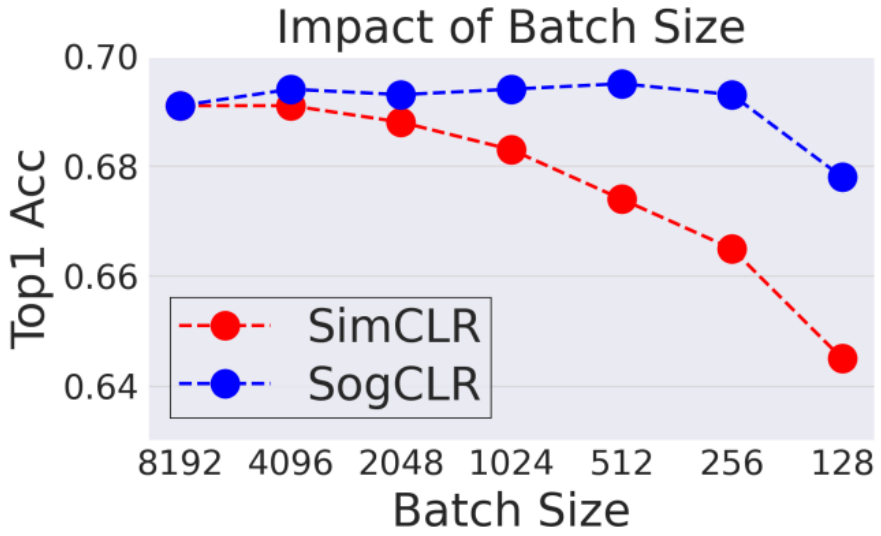


Contrastive Learning

- The performance of contrastive learning improves as the batch size N increases
- For Dense Passage Retrieval,

N	8	32	128
Performance	80.8	82.1	83.1

- In other contrastive learning studies,

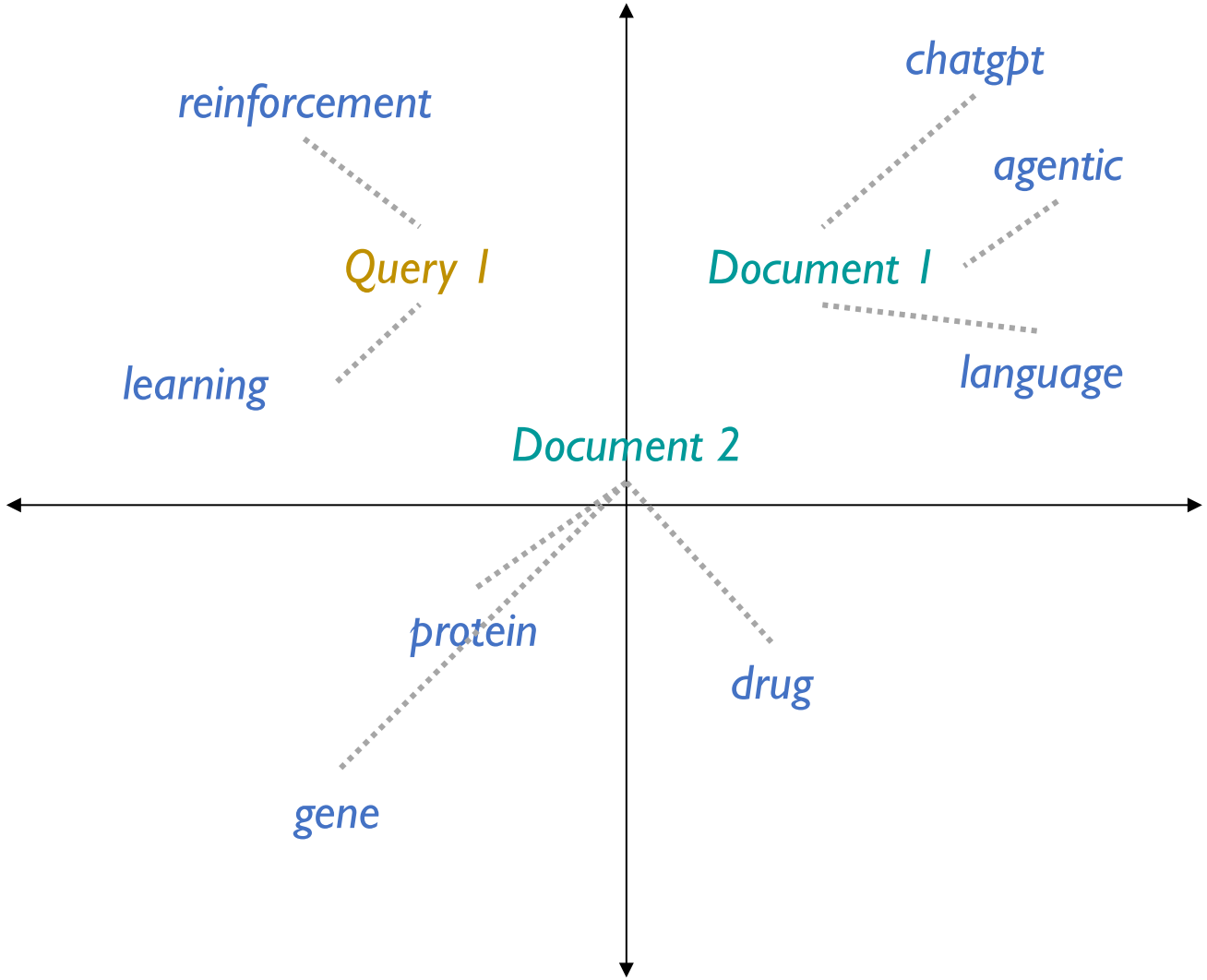


Contrastive Learning

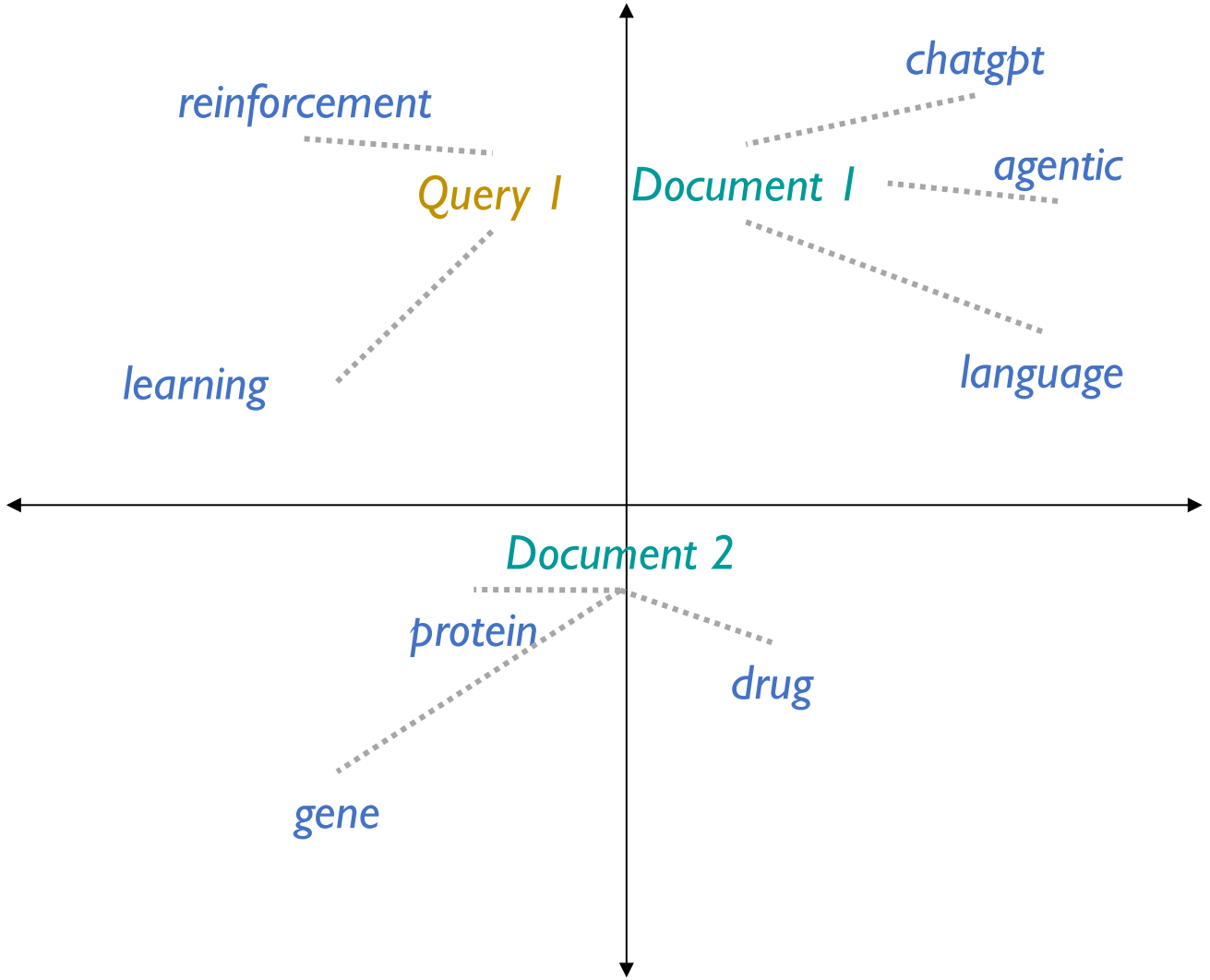
- Contrastive learning can be applied to scenarios far beyond retrieval
- Example 1: Citation Recommendation
  - Query: A research paper
  - Positive Document: A research paper cited by the query paper
  - Usage: When you write a new paper, the model can suggest possible citations
- Example 2: Community Question Answering
  - Query: A question from Stack Overflow
  - Positive Document: The answer with the most upvotes
  - Usage: When you post a new question on Stack Overflow, the model can suggest possible existing answers

Visualization of the Contrastive Learning Process

Suppose *Document 1* is the positive document of *Query 1*; *Document 2* is a negative document of *Query 1*.



Visualization of the Contrastive Learning Process

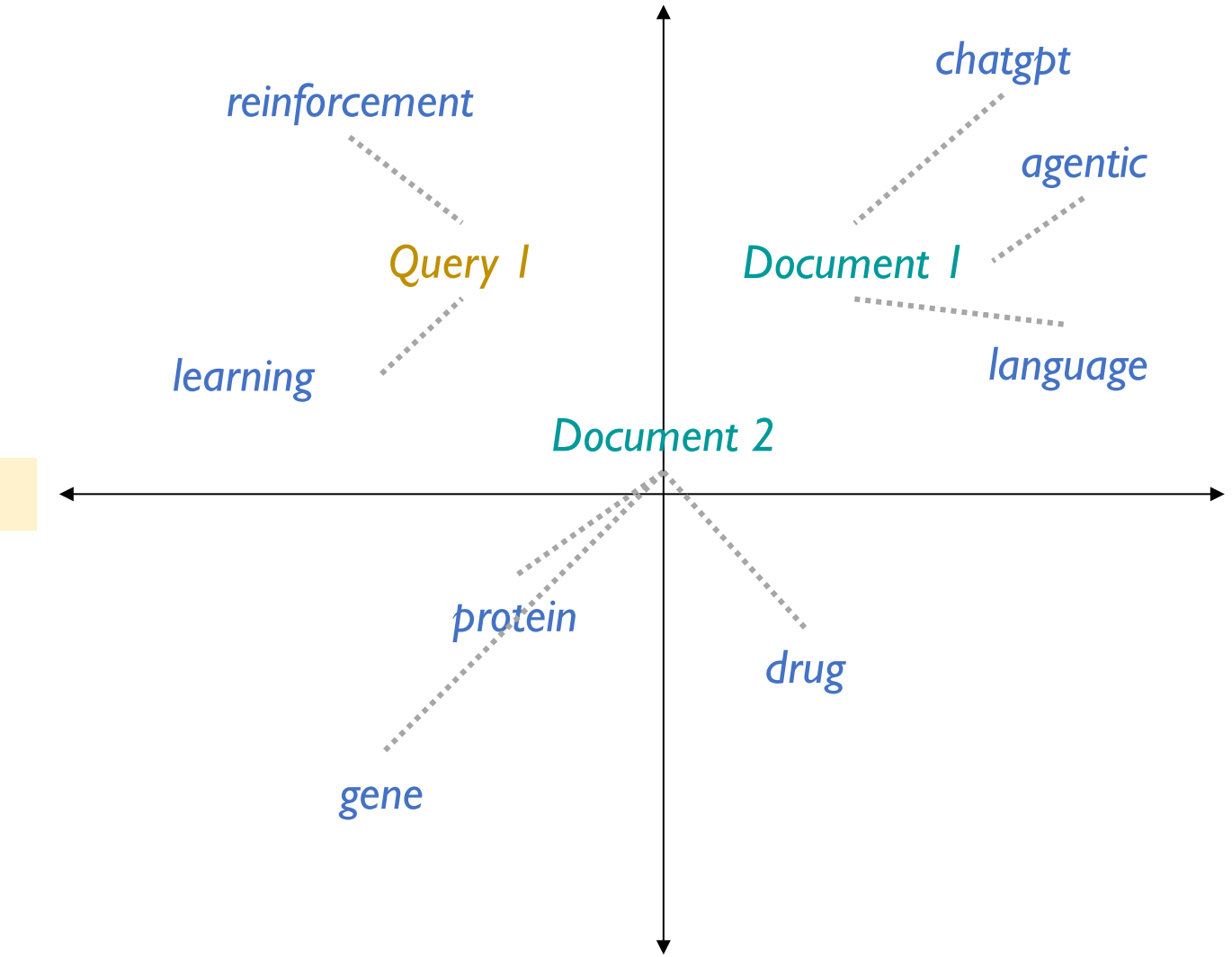


Contrastive learning pulls relevant (*query*, *document*) closer and pushes irrelevant (*query*, *document*) apart

But word embeddings do not change because they are not part of the model parameters θ

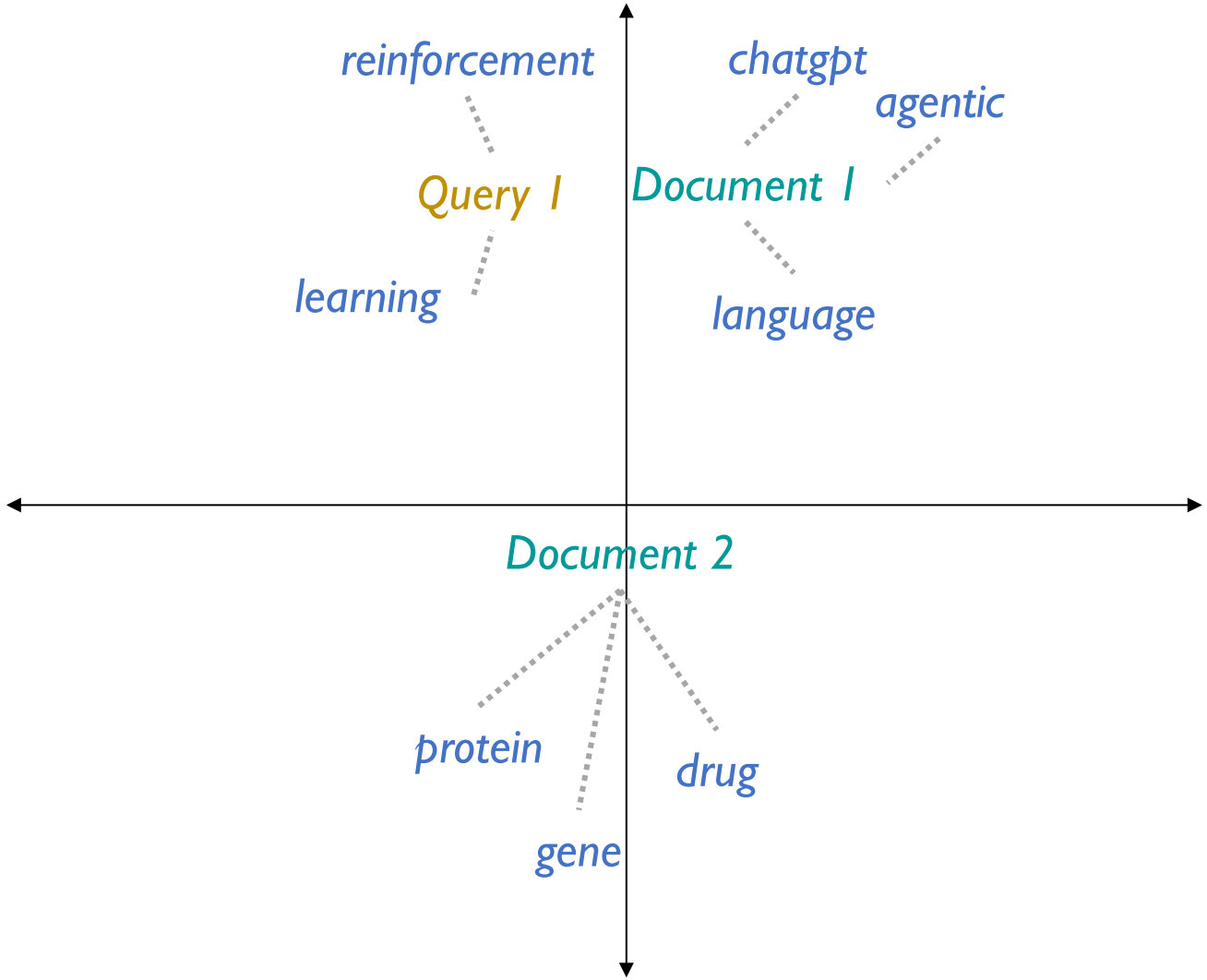
Can we also include word embeddings as model parameters?

Visualization of the Contrastive Learning Process



Before update

Visualization of the Contrastive Learning Process



After the model is trained on this sample, word embeddings are updated as well!

In practice, word embeddings will be included as model parameters if the training data (for contrastive learning) is large enough or domain-specific

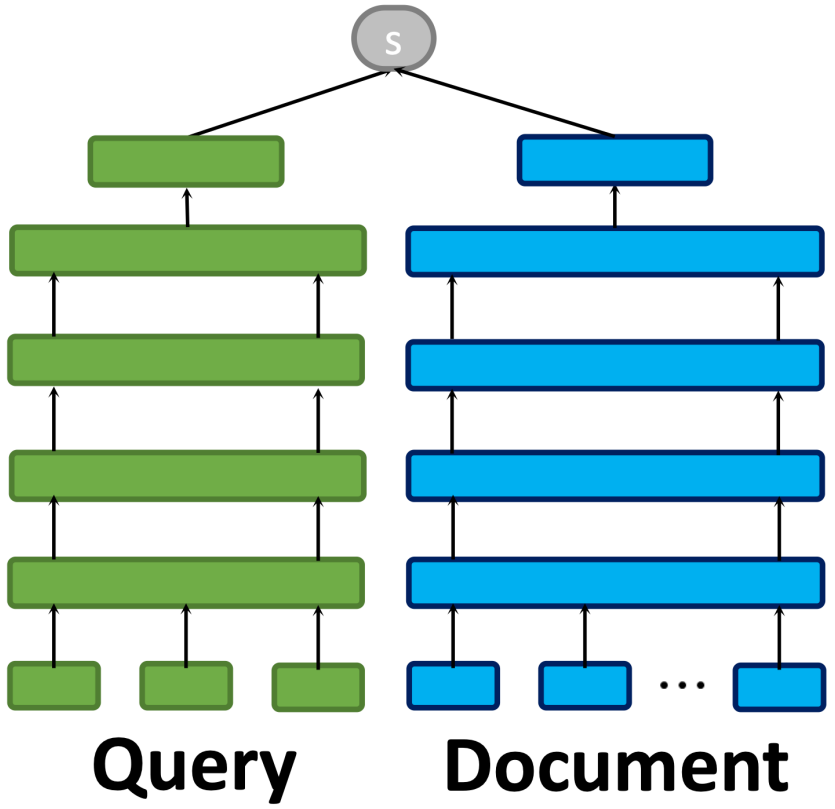
Questions?

Modeling Query-Document Interaction

- We have introduced
  - Average of word2vec
  - Dense Passage Retrieval (contrastive learning)

- They independently encode the query and the document into vectors and calculate their similarity.

- However, the importance of a query word may vary across different documents; the importance of a document word may also vary across different queries.



Modeling Query-Document Interaction

- Document: "*a comprehensive survey of scientific large language models and their applications in scientific discovery*"
- Query: "*chatgpt*"
- Document: "*a comprehensive survey of scientific large language models and their applications in scientific discovery*"
- Query: "*drug discovery*"
- Document: "*a comprehensive survey of scientific large language models and their applications in scientific discovery*"
- Capture interactions between each query word and each document word when predicting their relevance

Duet [Mitra et al., WWW 2017]

Learning to Match using Local and Distributed Representations of Text for Web Search

Bhaskar Mitra*[1,2], Fernando Diaz[1], and Nick Craswell[1]

[1]Microsoft, {bmitra, fdiaz, nickcr}@microsoft.com
[2]University College London, {bhaskar.mitra.15}@ucl.ac.uk

ABSTRACT

Models such as latent semantic analysis and those based on neural embeddings learn *distributed* representations of text, and match the query against the document in the latent semantic space. In traditional information retrieval models, on the other hand, terms have discrete or *local* representations, and the relevance of a document is determined by the exact matches of query terms in the body text. We hypothesize that matching with distributed representations complements matching with traditional local representations, and that a combination of the two is favourable. We propose a novel document ranking model composed of two separate deep neural networks, one that matches the query and the document using a local representation, and another that matches the query and the document using learned distributed representations. The two networks are jointly trained as part of a single neural network. We show that this combination or 'duet' performs significantly better than either neural network individually on a Web page ranking task, and

The President of the United States of America (POTUS) is the elected head of state and head of government of the United States. The president leads the executive branch of the federal government and is the commander in chief of the United States Armed Forces. Barack Hussein Obama II (born August 4, 1961) is an American politician who is the 44th and current President of the United States. He is the first African American to hold the office and the first president born outside the continental United States.

(a) Local model

The President of the United States of America (POTUS) is the elected head of state and head of government of the United States. The president leads the executive branch of the federal government and is the commander in chief of the United States Armed Forces. Barack Hussein Obama II (born August 4, 1961) is an American politician who is the 44th and current President of the United States. He is the first African American to hold the office and the first president born outside the continental United States.

(b) Distributed model

Conv-KNRM [Dai et al., WSDM 2018]

Convolutional Neural Networks for Soft-Matching N-Grams in Ad-hoc Search

Zhuyun Dai
Language Technologies Institute
Carnegie Mellon University
zhuyund@cs.cmu.edu

Chenyan Xiong
Language Technologies Institute
Carnegie Mellon University
cx@cs.cmu.edu

Jamie Callan
Language Technologies Institute
Carnegie Mellon University
callan@cs.cmu.edu

Zhiyuan Liu
Department of Computer Science and Technology
Tsinghua University
liuzy@tsinghua.edu.cn

ABSTRACT

This paper presents Conv-KNRM, a Convolutional Kernel-based Neural Ranking Model that models n-gram soft matches for ad-hoc search. Instead of exact matching query and document n-grams, Conv-KNRM uses Convolutional Neural Networks to represent n-grams of various lengths and soft matches them in a unified embedding space. The n-gram soft matches are then utilized by the kernel pooling and learning-to-rank layers to generate the final ranking score. Conv-KNRM can be learned end-to-end and fully optimized from user feedback. The learned model's generalizability is investigated by testing how well it performs in a related domain with small amounts of training data. Experiments on English search logs, Chinese search logs, and TREC Web track tasks demonstrated consistent advantages of Conv-KNRM over prior neural IR methods and feature-based methods.

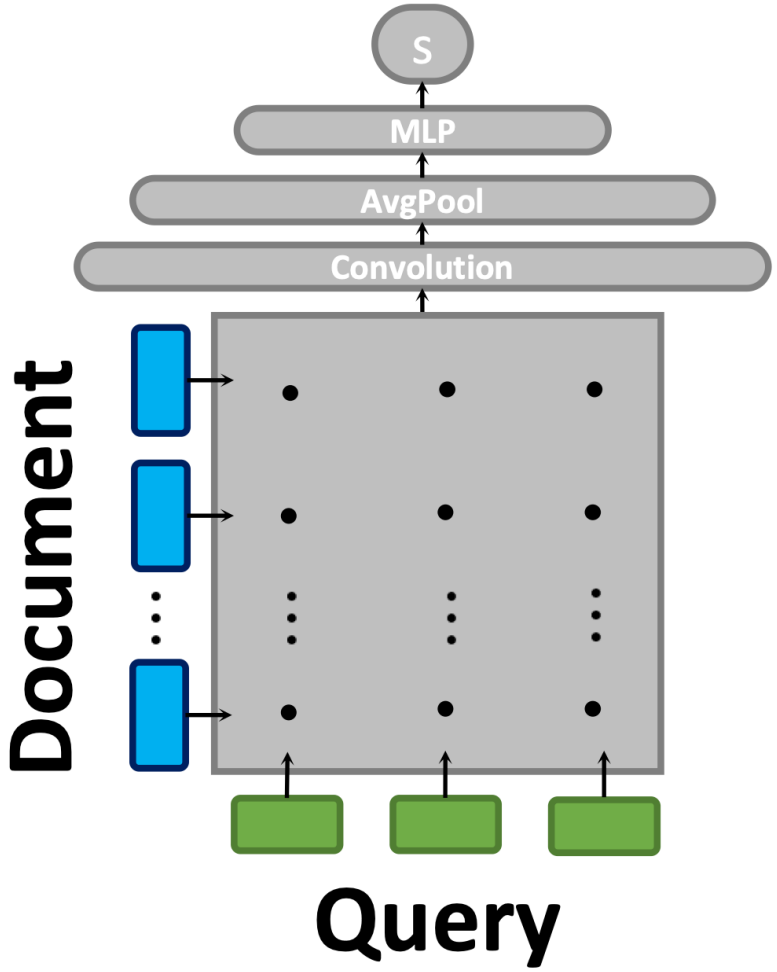
hand, the query and document often match at n-grams, such as phrases [18], concepts [2], and entities [28]; how to effectively model n-gram soft-matches remains an open question in neural IR.

This paper presents a new Convolutional Kernel-based Neural Ranking Model(Conv-KNRM). We first embed words in continuous vectors (embeddings), and then employ Convolutional Neural Networks (CNN) to compose adjacent words' embeddings to n-gram embeddings. In the n-gram embedding space, soft-matching n-grams is as simple as calculating the similarity of two n-grams' embeddings. The current state-of-the-art kernel pooling and learning-to-rank techniques are then used to combine the n-gram soft-matches to the final ranking score [29].

The CNN is the key to modeling n-grams. Typical IR approaches treat n-grams as discrete terms and use them the same as unigrams. For example, a document bigram 'white_house' is one term, has its

Modeling Query-Document Interaction

- Step 1: Get the embedding (e.g., word2vec) of each query word and each document word
- Step 2: Build a query-document interaction matrix (e.g., store the cosine similarity of each pair of words)
- Step 3: Reduce this dense matrix to a score using convolution layers and linear layers
  - Imagine how binary image classification is performed (i.e., predicting whether an image is a cat or a dog from its pixel matrix)
  - We are predicting whether a query-document pair is relevant from its interaction matrix
- Step 4: Learn these neural layers from training data



But …

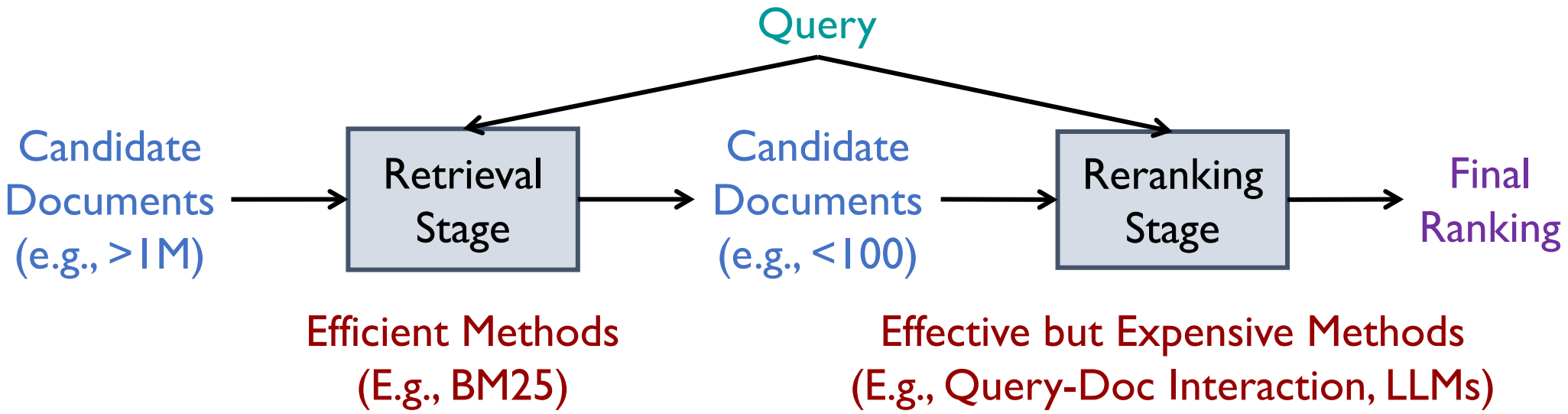
- Query-Document Interaction models are expensive!
- We cannot aggregate embeddings of document words until we see the query!
- By contrast,
  - For simple average:
    - $\boldsymbol{e}_q = \frac{1}{|q|}\sum_{w \in q} \boldsymbol{e}_w, \quad \boldsymbol{e}_d = \frac{1}{|d|}\sum_{w \in d} \boldsymbol{e}_w, \quad \text{score}(q, d) = \cos(\boldsymbol{e}_q, \boldsymbol{e}_d)$ or $\boldsymbol{e}_q^T \boldsymbol{e}_d$
    - We can precompute $\boldsymbol{e}_d$ for all documents
  - For DESM,
    - $\tilde{\boldsymbol{e}}_d = \frac{1}{|d|}\sum_{w \in d} \frac{\tilde{\boldsymbol{e}}_w}{\|\tilde{\boldsymbol{e}}_w\|}, \quad \text{score}(q, d) = \frac{1}{|q|}\sum_{w \in q} \cos(\boldsymbol{e}_w, \tilde{\boldsymbol{e}}_d)$
    - We can precompute $\tilde{\boldsymbol{e}}_d$ for all documents

But …

- Query-Document Interaction models are expensive!
- We cannot aggregate embeddings of document words until we see the query!
- By contrast,
  - For Dense Passage Retrieval:
    - $\boldsymbol{e}_q = \mathrm{f}_{\boldsymbol{\theta}}(\{\boldsymbol{e}_w | w \in q\}), \quad \boldsymbol{e}_d = \mathrm{f}_{\boldsymbol{\theta}}(\{\tilde{\boldsymbol{e}}_w | w \in d\}), \quad \mathrm{score}(q, d) = \boldsymbol{e}_q^T \boldsymbol{e}_d$
    - We can precompute $\boldsymbol{e}_d$ for all documents
- Imagine you have 1,000 queries and 1,000 documents:
  - How many times do you need to call $\mathrm{f}_{\boldsymbol{\theta}}(\cdot)$ to obtain the score between each query and each document?
  - How many times do you need to call a Query-Document Interaction model to obtain the score between each query and each document?

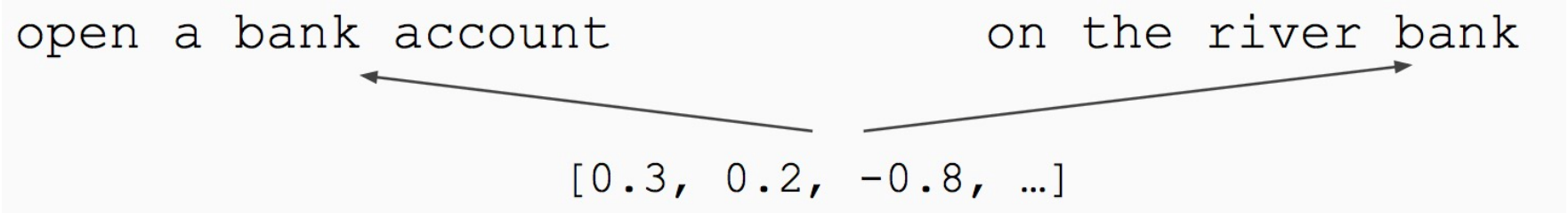
Retrieval-Reranking Paradigm

- We want to use effective but expensive ranking models …
- … only for a more fine-grained ranking of the most relevant documents.



Limitations of word2vec-based Ranking

- Word embeddings are applied in a context-free manner!



- Is it possible to have different vectors of the same word given different contexts?

Transformer [Vaswani et al., NIPS 2017]

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gom University of Tor aidan@cs.toront

Illia Polos illia.polosukh

Attention is all you need [PDF] neurips.cc

A Vaswani, N Shazeer, N Parmar... - Advances in neural ..., 2017 - proceedings.neurips.cc

... to attend to **all** positions in the decoder up to and including that position. **We need** to prevent ... **We** implement this inside of scaled dot-product **attention** by masking out (setting to $-\infty$) ...

☆ Save 99 Cite Cited by 231902 Related articles All 71 versions »

BERT [Devlin et al., NAACL 2019]

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com

Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both

Bert: Pre-training of deep bidirectional transformers for language understanding [PDF] aclanthology.org

J Devlin, MW Chang, K Lee... - Proceedings of the 2019 ..., 2019 - aclanthology.org

... **BERT** and its detailed implementation in this section. There are two steps in our framework: **pre-training** and fine... of **BERT** by evaluating two **pretraining** objectives using exactly the same ...

☆ Save 99 Cite Cited by 161948 Related articles All 21 versions »

tional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal

Thank You!

Course Website: https://yuzhang-teaching.github.io/CSCE670-S26.html