CS310 Natural Language Processing
自然语言处理
Lecture 02 - Word Vectors

Instructor: Yang Xu

主讲人：徐炀

xuyang@sustech.edu.cn

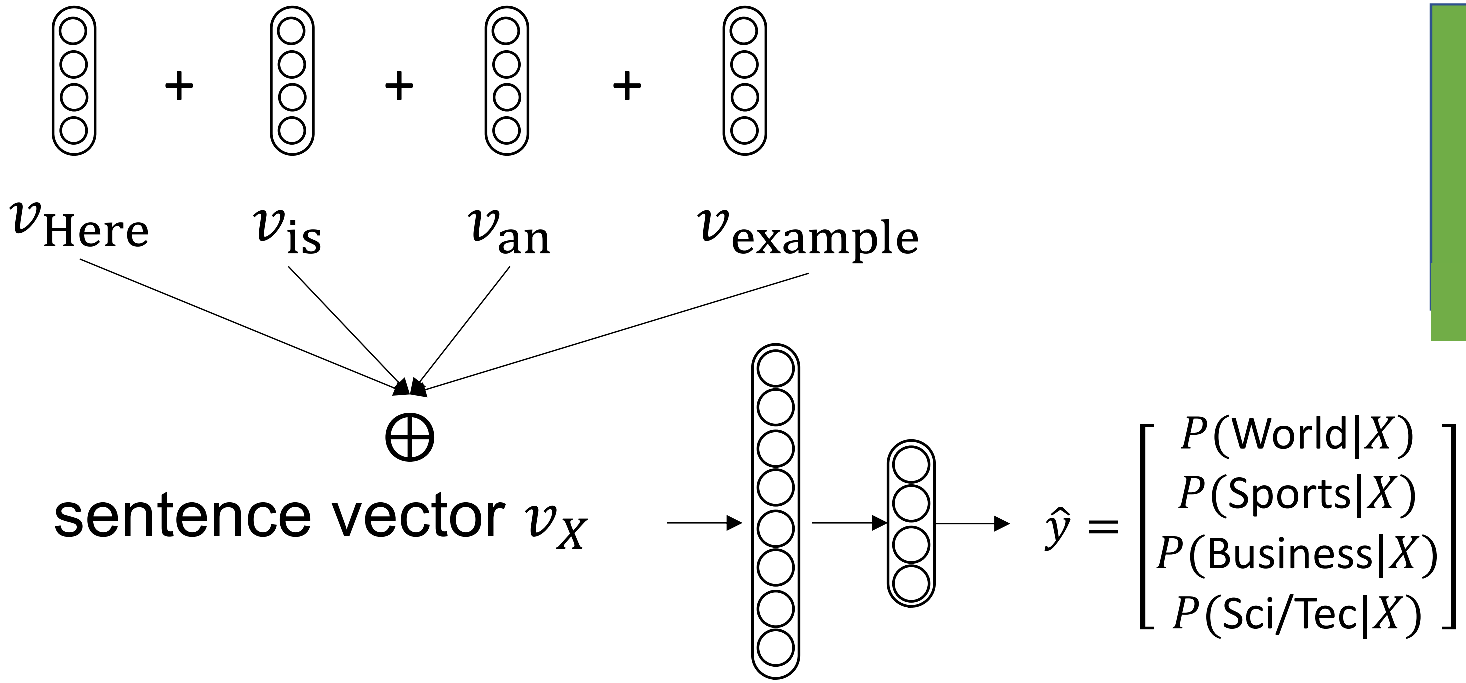
Content

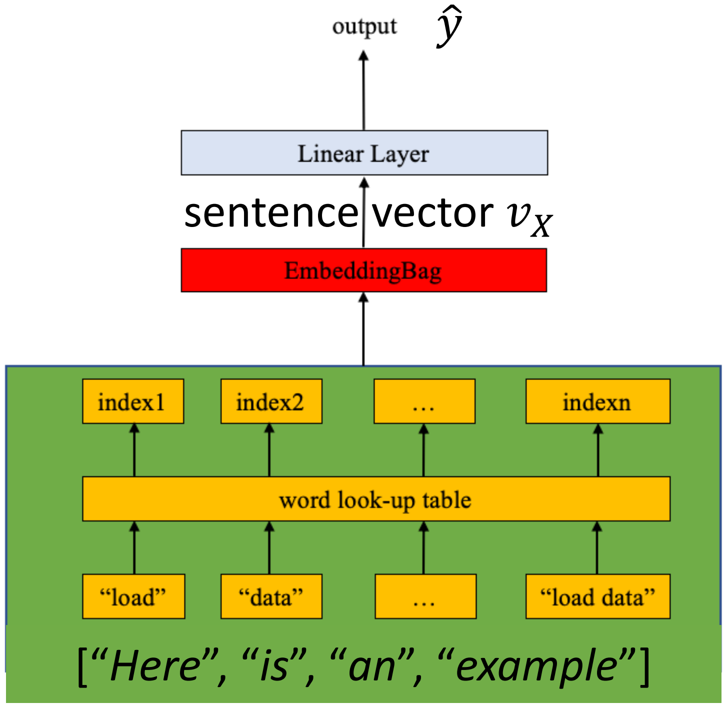
- **Motivation**
- Documents and Counts-based Method
- Neural Network-based Method -- word2vec
- Evaluation and Applications

Recap: Bag-of-Words Neural Networks

Task: News text classification

X: ["Here", "is", "an", "example"]





Naïve method: one-hot vectors

- Words as discrete symbols ⇔ localist representations
- **One-hot** vectors

$$\text{Vocabulary (10k)} = \begin{bmatrix} a \\ about \\ all \\ \vdots \\ zoo \end{bmatrix}$$

Apple [0 0 0 0 0 0 0 0 **1** 0 0 0 0 0 0 0 0 ... 0]

Orange [0 0 0 0 0 0 0 0 0 0 0 0 0 0 **1** 0 0 0 ... 0]

I would like some apple juice

I would like some orange ______

One-hot vector is not helpful

Distance between any pair of words is **constant:**

Euclidean distance = $\sqrt[2]{(1-0)^2+(1-0)^2}$

Cosine distance = 0

Ideally ⇒ real-valued word vectors

	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.98	0.97	0.00	-0.01
Royal	0.01	0.02	0.93	0.98	-0.01	0.00
Age	0.03	0.02	0.72	0.68	0.03	0.02
Food	0.00	0.00	0.01	0.02	0.95	0.97

main difference: gender

main difference: gender

$$e_{Man} = \begin{bmatrix} -1 \\ 0.01 \\ 0.03 \\ 0.0 \end{bmatrix} \quad e_{Woman} = \begin{bmatrix} 1 \\ 0.02 \\ 0.02 \\ 0.0 \end{bmatrix} \quad e_{Man} - e_{Woman} = \begin{bmatrix} -2 \\ -0.01 \\ 0.01 \\ 0.00 \end{bmatrix} \quad e_{King} - e_{Queen} = \begin{bmatrix} -1.95 \\ -0.05 \\ 0.04 \\ -0.01 \end{bmatrix}$$

With real-valued dense vectors, word similarity can be computed more accurately

Content

- Motivation
- **Documents and Counts-based Method**
  - **LSA and TF-IDF**
- Neural Network-based Method -- word2vec
- Evaluation and Applications

Documents and Word Counts

- **Goal**: Derive word vectors from a collection of documents
- without annotation -- unsupervised/self-supervised

- **Notations**:
- $\boldsymbol{x}$ is the collection of C documents
- $\boldsymbol{x_c}$ is the cth document in the corpus
- ℓ_c is the length of $\boldsymbol{x_c}$ (in # of tokens)
- N is the total number of tokens (words), $N = \sum_{c=1}^{C} \ell_c$

Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/vectors.pdf

Build Word-Document Matrix (term-document matrix)[1]

- Build matrix $\mathbf{A} \in \mathbb{R}^{V\times C}$, which contains the count of each word in each document
- **Example**:

x_1: 学 而 时 习 之
x_2: 学 而 不 思 则 罔
x_3: 思 而 不 学 则 殆

Entry $\mathbf{A}_{v,c} = \text{count}_{x_c}(v)$, count of word v in the cth document

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

[1] https://en.wikipedia.org/wiki/Term-document_matrix

Q: Can we directly use this matrix?

- **Example**
- 《论语》前十篇内容，8664字，267章

$C = 267$

$V = 8664$

	x_1	x_2	x_3	⋯
子				⋯
曰				⋯
学				⋯
⋮	⋮	⋮	⋮	

$$v(\text{子}) = [2., 2., 1., 1., 1., 2., 1., 2., \cdots, 0., 1.]$$

$$v(\text{曰}) = [1., 1., 1., 1., 1., 1., 2., 1., \cdots, 0., 1.]$$

$$v(\text{学}) = [1., 0., 0., 0., 0., 1., 2., 1., \cdots, 0., 0.]$$

Most of them are same numbers.
Are they really necessary?

Q: How do we interpret the matrix?

- What is the expected occurrence of word v in document c?
- Under a simple assumption, the chance of word v to occur at any position is $\frac{\text{count}_{\boldsymbol{x}}(v)}{N}$, (where $\text{count}_{\boldsymbol{x}}(v)$ is the count of v over all documents)
- So the expected occurrence of v in a document of length ℓ_c is $\frac{\text{count}_{\boldsymbol{x}}(v)}{N} \cdot \ell_c$
- Consider the **ratio** of *observed* count of v in document c, $\text{count}_{x_c}(v)$, to the expected count $\frac{\text{count}_{\boldsymbol{x}}(v)}{N} \cdot \ell_c$

Intuition of surprise in word

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

$\text{count}_{\boldsymbol{x}}(\text{学}) = 1 + 1 + 1 = 3$

Expected count of 学 in x_1 is $\frac{\text{count}_{\boldsymbol{x}}(\text{学})}{N} \cdot \ell_1 = \frac{3}{17} \cdot 5 \approx 0.88$

The observed count of 学 in x_1 is $\text{count}_{\boldsymbol{x_1}}(\text{学}) = 1$

The **surprise** of seeing 学 in x_1 is:

$$\log \frac{\text{observed}}{\text{expected}} = \log \frac{\text{count}_{\boldsymbol{x_1}}(\text{学})}{\frac{\text{count}_{\boldsymbol{x}}(\text{学})}{N} \cdot \ell_1} \approx \log \frac{1}{0.88} \approx 0.125$$

Intuition of surprise in word

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

$\text{count}_{\boldsymbol{x}}(习) = 1 + 0 + 0 = 1$

Expected count of 习 in x_1 is $\frac{\text{count}_{\boldsymbol{x}}(习)}{N} \cdot \ell_1 = \frac{1}{17} \cdot 5 \approx 0.29$

The observed count of 习 in x_1 is $\text{count}_{\boldsymbol{x}_1}(习) = 1$

The **surprise** of seeing 习 in x_1 is:

$$\log \frac{\text{observed}}{\text{expected}} = \log \frac{\text{count}_{\boldsymbol{x}_1}(习)}{\frac{\text{count}_{\boldsymbol{x}}(习)}{N} \cdot \ell_1} \approx \log \frac{1}{0.29} \approx 1.223$$

> surprise of 学

Pointwise Mutual Information

- From matrix $\mathbf{A} \in \mathbb{R}^{V \times C}$, derive positive **pointwise mutual information**

$$[\mathbf{A}]_{v,c} = \left[\log \frac{\text{count}_{\boldsymbol{x}_c}(v)}{\frac{\text{count}_{\boldsymbol{x}}(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{N \cdot \text{count}_{\boldsymbol{x}_c}(v)}{\text{count}_{\boldsymbol{x}}(v) \cdot \ell_c}\right]_+ \quad \text{where } [x]_+ = \max(0, x)$$

More examples:

$$[\mathbf{A}]_{学,2} = \log \frac{17 \cdot 1}{3 \cdot 6} \approx -0.057 \to 0$$

rounded to 0 because of max()

$$[\mathbf{A}]_{思,2} = \log \frac{17 \cdot 1}{2 \cdot 6} \approx 0.348$$

Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/vectors.pdf

Meaning of PMI

Random variable ***A*** and ***B***

$$\mathsf{PMI}(a,b) = \log \frac{p(A=a, B=b)}{p(A=a)\cdot p(B=b)} = \log \frac{p(A=a \mid B=b)}{p(A=a)} = \log \frac{p(B=b \mid A=a)}{p(B=b)}$$

Mutual Information (MI):
The amount of information each r.v. offers about the other.
I.e., how much do we know about **B** by knowing about **A**

$$[\mathbf{A}]_{v,c} = \left[\log \frac{\mathrm{count}_{\boldsymbol{x}_c}(v)}{\frac{\mathrm{count}_{\boldsymbol{x}}(v)}{N}\cdot \ell_c}\right]_+ = \left[\log \frac{\frac{\mathrm{count}_{\boldsymbol{x}_c}(v)}{\ell_c}}{\frac{\mathrm{count}_{\boldsymbol{x}}(v)}{N}}\right]_+$$

Local probability

Global probability

Example:

$$\log \frac{\mathrm{count}_{\boldsymbol{x}_1}(\text{习})}{\frac{\mathrm{count}_{\boldsymbol{x}}(\text{习})}{N}\cdot \ell_1} \approx \log \frac{1}{0.29} \approx 1.223$$

is high, which means we learn a lot about the global meaning of "习" by reading x_1

How much do we know about the global meaning of v by knowing about its local meaning in document c

Slide credit: https://nasmith.github.io/NLP-winter23/assets/slides/vectors.pdf

Pointwise Mutual Information

$$PMI = [\mathbf{A}]_{v,c} = \left[\log \frac{\text{count}_{x_c}(v)}{\frac{\text{count}_x(v)}{N} \cdot \ell_c}\right]_+$$

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

x_1: 学 而 时 习 之
x_2: 学 而 不 思 则 罔
x_3: 思 而 不 学 则 殆

PMIs highlight the most informative words

	x_1	x_2	x_3
学	0	0	0
而	0	0	0
不	0	0	0
思	0	0	0
则	0	0	0
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

Properties of PMI

- If a word v has nearly same frequency in every document, then its row $[\mathbf{A}]_{v,*}$ will be nearly all zeros
- If a word v only occurs in one document c, then its PMI will be large and positive
- Thus, PMI is sensitive to **rare** words; usually need to smooth the frequencies by filtering rare words

Reflection

- Can we directly use word-document matrix $\mathbf{A} \in \mathbb{R}^{V \times C}$ (or smoothed PMI $[\mathbf{A}]$) to represent word meanings?
- For example, can we use the row vectors as input features for a neural text classifier?
- What are the advantages/disadvantages?

Improvement: Latent Semantic Analysis

(Deerwester et al., 1990)

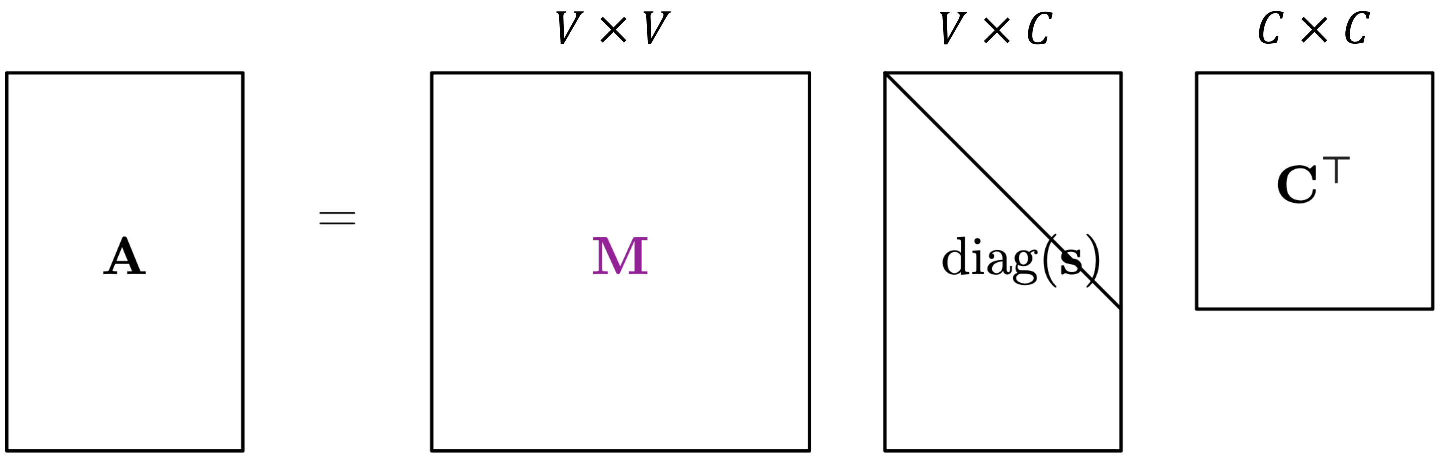
- LSA seeks to find a more compact (low rank) representation of word-document matrix $\mathbf{A}$

$$\underset{V \times C}{\mathbf{A}} \approx \hat{\mathbf{A}} = \underset{V \times d}{\mathbf{M}} \times \underset{d \times d}{\mathrm{diag}(\boldsymbol{s})} \times \underset{d \times C}{\mathbf{C}^\top}$$

- Can be solved by applying singular value decomposition to $\mathbf{A}$, and then truncating to d dimensions ($\hat{\mathbf{A}}$)
- $\mathbf{M}$ contains left singular vectors of $\mathbf{A}$
- $\mathbf{C}$ contains right singular vectors of $\mathbf{A}$
- $\boldsymbol{s}$ are singular values of $\mathbf{A}$

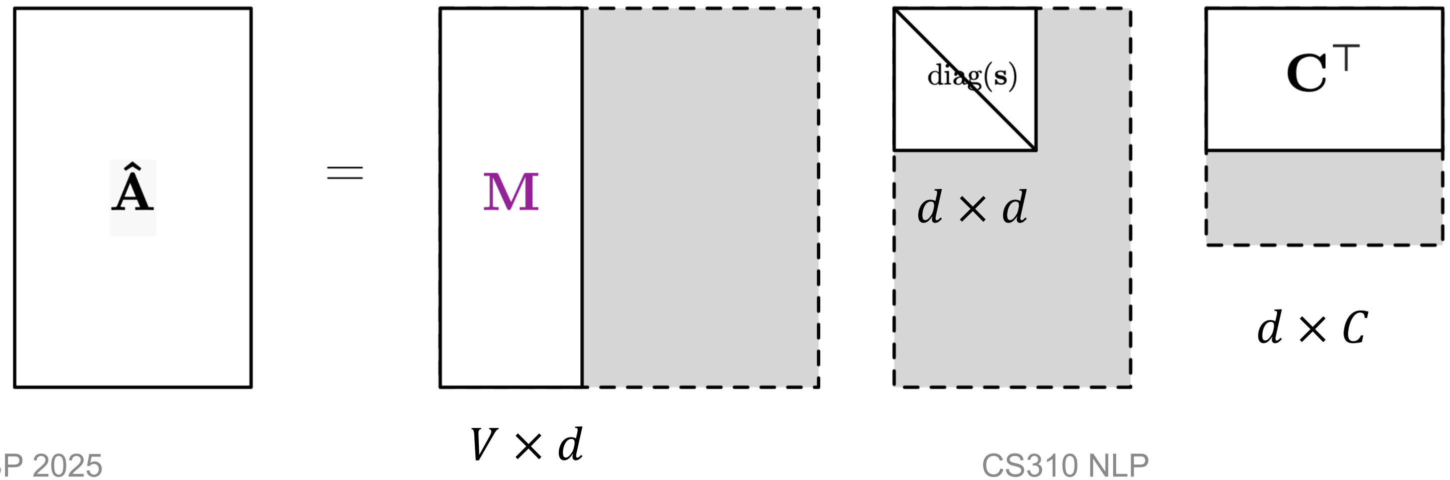
SVD and Truncated SVD

SVD:



- $\mathbf{M}$ and $\mathbf{C}$ are unitary, i.e., $\mathrm{MM}^\top = \mathrm{I}$ and $\mathrm{CC}^\top = \mathrm{I}$
- $\mathrm{diag}(\mathbf{s})$ only has non-zero elements at diagonal
- $\mathbf{M}$ are eigenvectors of $\mathbf{AA}^\top$
- $\mathbf{C}$ are eigenvectors of $\mathbf{A}^\top\mathbf{A}$
- $\mathbf{s}^2$ are eigenvalues

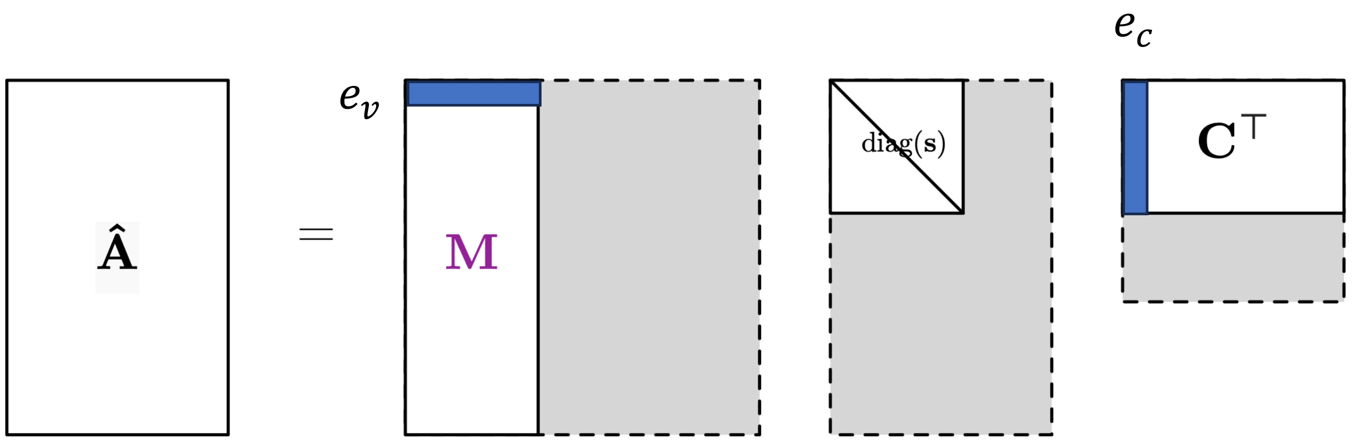
SVD truncated at d dimensions:



- Truncated: keeping only top d singular values in $\mathbf{s}$
- corresponding d columns in $\mathbf{M}$ and $\mathbf{C}$

Truncated SVD => word vectors

$$\mathbf{A} \approx \hat{\mathbf{A}} = \mathbf{M} \times \mathrm{diag}(\mathbf{s}) \times \mathbf{C}^\top$$

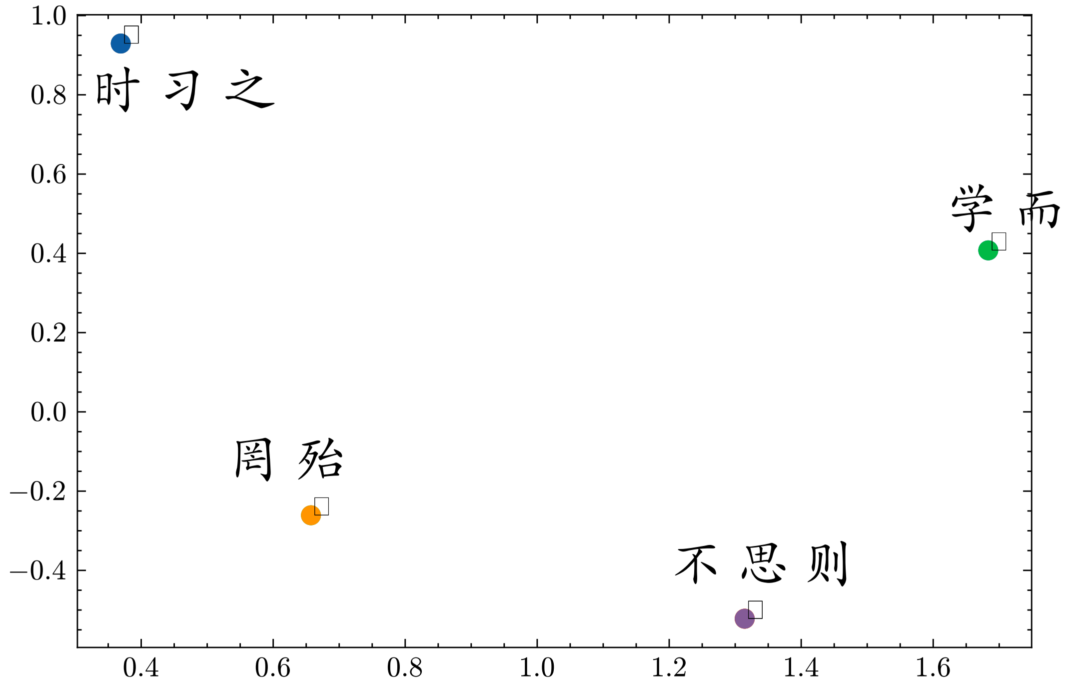


- vth row in $\mathbf{M}$ is the embedding vector for word v
- cth column in $\mathbf{C}$ is the embedding vector for document c

- $\mathbf{M}$ contains useful word vectors ("embeddings") of d dimensions
- $\mathbf{C}$ contains document vectors

LSA Example $d = 2$

- Word vectors $\mathbf{M}$ plotted
- Note that some words are in the same spot. Why?



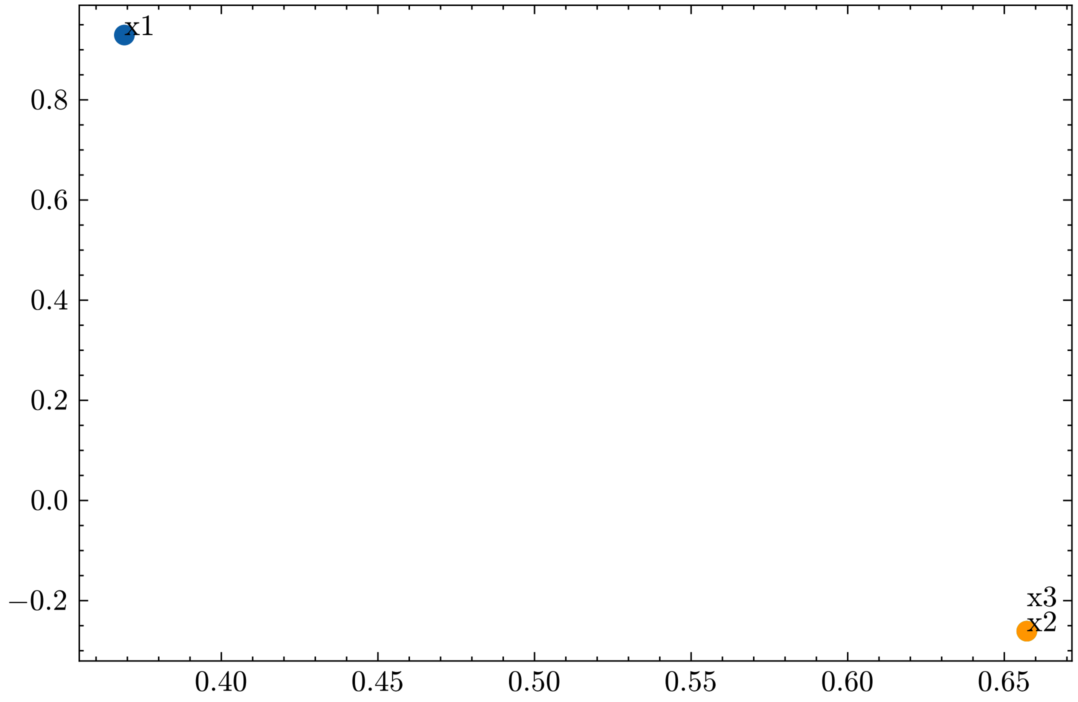
$$\mathbf{A} \approx \hat{\mathbf{A}}$$
$$= \mathbf{M} \times \mathrm{diag}(\mathbf{s}) \times \mathbf{C}^\top$$

$\mathbf{A} =$

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

LSA Example $d = 2$

- Document vectors $\mathbf{C}$ plotted
- Note that documents x_2 and x_3 are in the same spot. Why?



$\mathbf{A} =$

	x_1	x_2	x_3
学	1	1	1
而	1	1	1
不	0	1	1
思	0	1	1
则	0	1	1
时	1	0	0
习	1	0	0
之	1	0	0
罔	0	1	0
殆	0	0	1

LSA Summarized

- It creates a mapping of words and documents into the same low-dimensional space.
- Bag-of-words assumption (Salton et al., 1975):
  - A document is nothing more than the distribution of words it contains.
- Distributional hypothesis (Harris, 1954; J.R. Firth, 1957):
  - Words' meanings are nothing more than the distribution of *contexts* (here, documents) they occur in.
  - Words that occur in similar contexts have similar meanings.
- Word-document matrix $\mathbf{A}$ is sparse and noisy; LSA "fills in" the zeroes and tries to eliminate the noise.
- It finds the best rank-d approximation to $\mathbf{A}$.

Adapted from: https://nasmith.github.io/NLP-winter23/assets/slides/vectors.pdf

Content

- Motivation
- **Documents and Counts-based Method**
  - **LSA and TF-IDF**
- Neural Network-based Method -- word2vec
- Evaluation and Applications

TF-IDF

- Background: Find the most relevant document among a collection of documents, using a query

tf: term frequency
idf: inverse document frequency

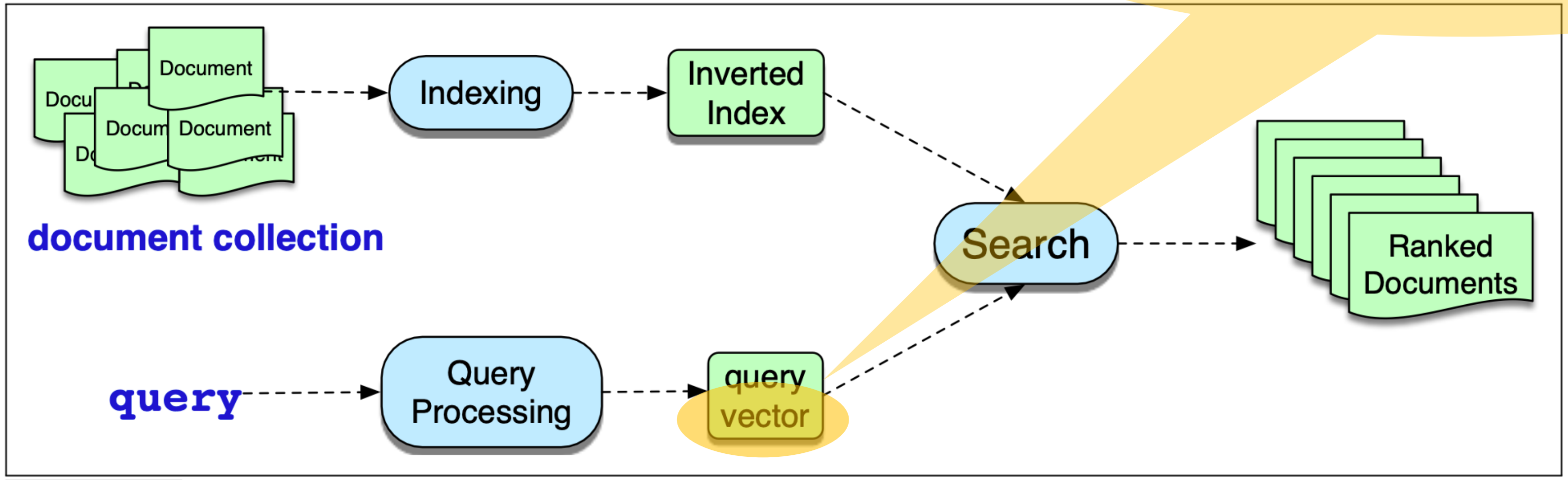


Figure 14.1 The architecture of an ad hoc IR system.

How to match a document a query?

- Compute a term weight for each document term
- **tf**: term frequency
- **idf**: inverse document frequency
- **tf-idf ≜ tf × idf** (product of the two)

term t; document d

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

- **tf**: words that occur more often in a document are likely to be informative about the document's content
- Use the $\log_{10}$ of word frequency count rather than raw count
- *Why? A word appearing 100 times doesn't make it 100 times more likely*

Tf-idf

$$\text{tf}_{t,d} = \begin{cases} 1 + \log_{10} \text{count}(t,d) & \text{if } \text{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

term t; document d

term occurs 0 times in document: tf = 0
term occurs 1 times in document: tf = 1
term occurs 10 times in document: tf = 2, …

- **document frequency** df_t of a term t is the number of documents it occurs in
- Terms that occur in only **a few** documents are useful for discriminating those documents from the rest of the collection;
- terms that occur across the entire collection aren't as helpful (*the, a, an,* …)
- **inverse document frequency** or **idf** is defined as:

$$\text{idf}_t = \log_{10} \frac{N}{\text{df}_t}$$

N: total number of documents
The fewer documents in which t occurs, the higher idf_t

Inverse document frequency example

- Some idf values for some words in the corpus of Shakespeare plays

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

Extremely informative words that occur in only one play like *Romeo*

good or *sweet* tare completely non-discriminative since they occur in all 37 plays

Scoring with tf-idf

- We can score document d by the cosine of its vector $\vec{\boldsymbol{d}}$ with the query vector $\vec{\boldsymbol{q}}$:

$$\text{score}(q,d) = \cos(\vec{\boldsymbol{q}}, \vec{\boldsymbol{d}}) = \frac{\vec{\boldsymbol{q}} \cdot \vec{\boldsymbol{d}}}{|\vec{\boldsymbol{q}}| \cdot |\vec{\boldsymbol{d}}|}$$

- in which $\vec{\boldsymbol{q}}$ and $\vec{\boldsymbol{d}}$ are vectors of query length n, whose values are the **tf-idf** values (normalized):

$$\vec{\boldsymbol{q}} = \frac{[\text{tfidf}(t_1, q), \dots, \text{tfidf}(t_n, q)]}{\sqrt{\sum_{t \in q} \text{tfidf}^2(t, q)}}$$

$$\vec{\boldsymbol{d}} = \frac{[\text{tfidf}(t_1, d), \dots, \text{tfidf}(t_n, d)]}{\sqrt{\sum_{t \in d} \text{tfidf}^2(t, d)}}$$

$$\text{score}(q,d) = \sum_{t_i \in q} \frac{\text{tfidf}(t_i, q)}{\sqrt{\sum_{t \in q} \text{tfidf}^2(t, q)}} \cdot \frac{\text{tfidf}(t_i, d)}{\sqrt{\sum_{t \in d} \text{tfidf}^2(t, q)}}$$

Tf-idf scoring example

- A collection of 4 nano documents

Query:	sweet love
Doc 1:	Sweet sweet nurse! Love?
Doc 2:	Sweet sorrow
Doc 3:	How sweet is love?
Doc 4:	Nurse!

Query vector $\vec{\boldsymbol{q}} = (0.383, 0.924)$

Query						
word	**cnt**	**tf**	**df**	**idf**	**tf-idf**	**n'lized** = tf-idf/$\lvert q \rvert$
sweet	1	1	3	0.125	0.125	0.383
nurse	0	0	2	0.301	0	0
love	1	1	2	0.301	0.301	0.924
how	0	0	1	0.602	0	0
sorrow	0	0	1	0.602	0	0
is	0	0	1	0.602	0	0

$|q| = \sqrt{.125^2 + .301^2} = .326$

Tf-idf scoring example

Query vector $\vec{q} = (0.383, 0.924)$

			Document 1		
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**× q**
sweet	2	1.301	0.163	0.357	**0.137**
nurse	1	1.000	0.301	0.661	0
love	1	1.000	0.301	0.661	**0.610**
how	0	0	0	0	0
sorrow	0	0	0	0	0
is	0	0	0	0	0

$|d_1| = \sqrt{.163^2 + .301^2 + .301^2} = .456$

			Document 2		
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**×q**
sweet	1	1.000	0.125	0.203	**0.0779**
nurse	0	0	0	0	0
love	0	0	0	0	**0**
how	0	0	0	0	0
sorrow	1	1.000	0.602	0.979	0
is	0	0	0	0	0

$|d_2| = \sqrt{.125^2 + .602^2} = .615$

$\vec{d}_1 = (0.357, 0.661)$

$\vec{d}_2 = (0.203)$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.747}$

$\text{score}(\vec{q}, \vec{d}_1) = \mathbf{0.0779}$

Therefore, d_1 is more relevant

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Table of Content

- Motivation
- Documents and Counts-based Method
- **Neural Network-based Method -- word2vec**
- Evaluation and Applications

Motivation: Distributional semantics

- 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)
- When a word w appears in a text, its local **context** is the set of words that co-occur within a fixed-size window

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

The meaning of "banking" is represented by these context words

Example from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

In What Form of Representation?

- **Goal**: Obtain a dense vector for each word, so that word sense similarity can be computed via vector distance, such as dot product

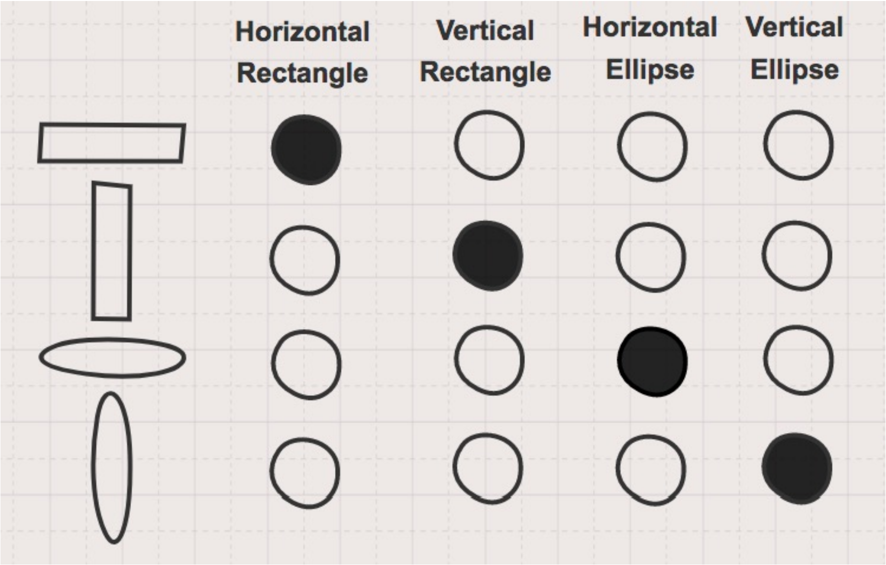
$$e_{apple} = \begin{bmatrix} 0.00 \\ -0.01 \\ 0.03 \\ 0.95 \\ \dots \\ 0.21 \end{bmatrix} \quad e_{orange} = \begin{bmatrix} -0.01 \\ 0.00 \\ 0.02 \\ 0.97 \\ \dots \\ 0.22 \end{bmatrix} \Bigg\}$$

Common dimension size: 100-d,200-d,300-*d*, …

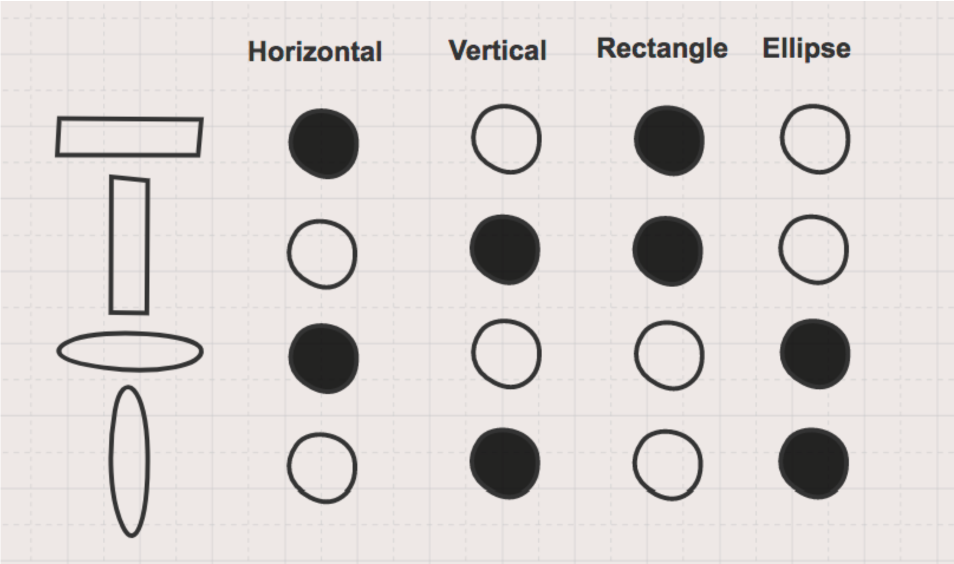
These dense word vectors are also called word **embeddings (嵌入)**
(which implies the idea of placing or mapping words into some continuous vector space)

Intuition: One-hot vs. Distributed repr.

One-hot representation



Distributed representation



The individual dimensions of a word embedding do not have concrete "meanings"

$$E_{Orange} = \begin{bmatrix} -0.01 \\ 0.00 \\ 0.02 \\ 0.97 \\ \dots \\ 0.22 \end{bmatrix}$$

For instance, e_{orange} It does NOT mean
1st dimension -0.01 is for "animalness"
4th dimension 0.97 is for "fruitness"
They are only meaningful when compared to other words

Images source: https://www.oreilly.com/ideas/how-neural-networks-learn-distributed-representations

Question: How to obtain word embeddings?

- An effective and efficient method: Word2vec (Mikolov et al. 2013 a&b)
- **Basic Idea**:
- Given a corpus as a list of words
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the **similarity of word vectors** between c and o to **compute the probability** of o given c, i.e., conditional probability $P(o|c)$ (or vice versa)
- **Maximize this probability** by keep adjusting the word vectors

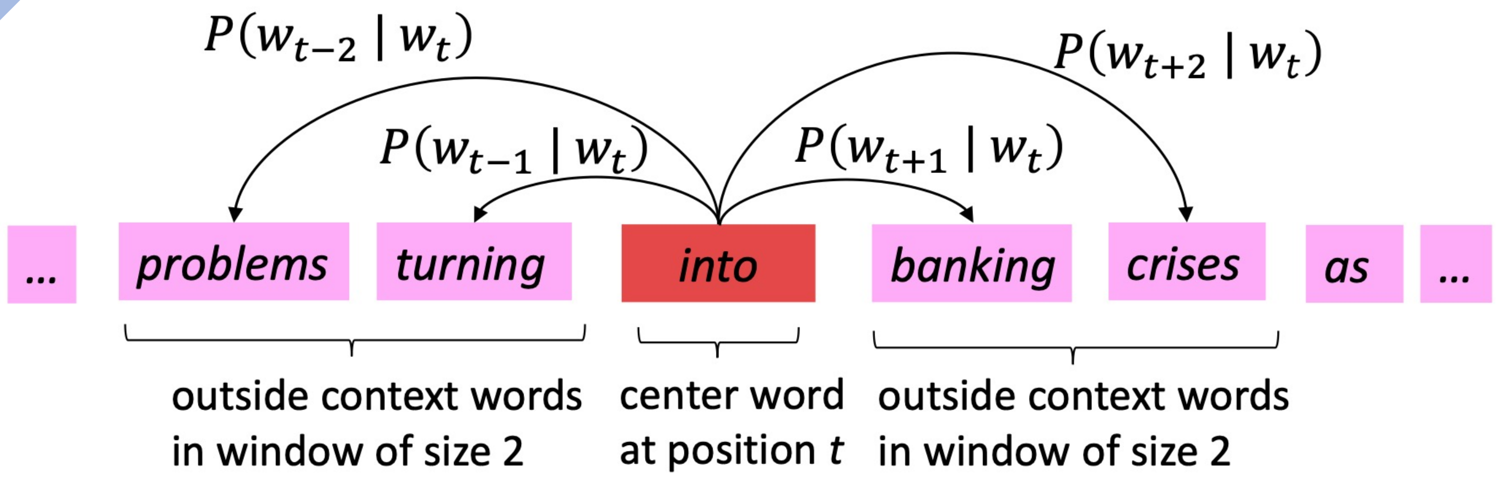
使用中心词c与上下文词o的词向量相似度来计算条件概率 (|)（或者反过来）

Two architectures of Word2vec

Word2vec:
- **Skip-gram**: Maximize $P(o|c)$
- Continuous Bag-of-words (**CBOW**): Maximize $P(c|o)$

- “o” for outside (context) words
- “c” for center word

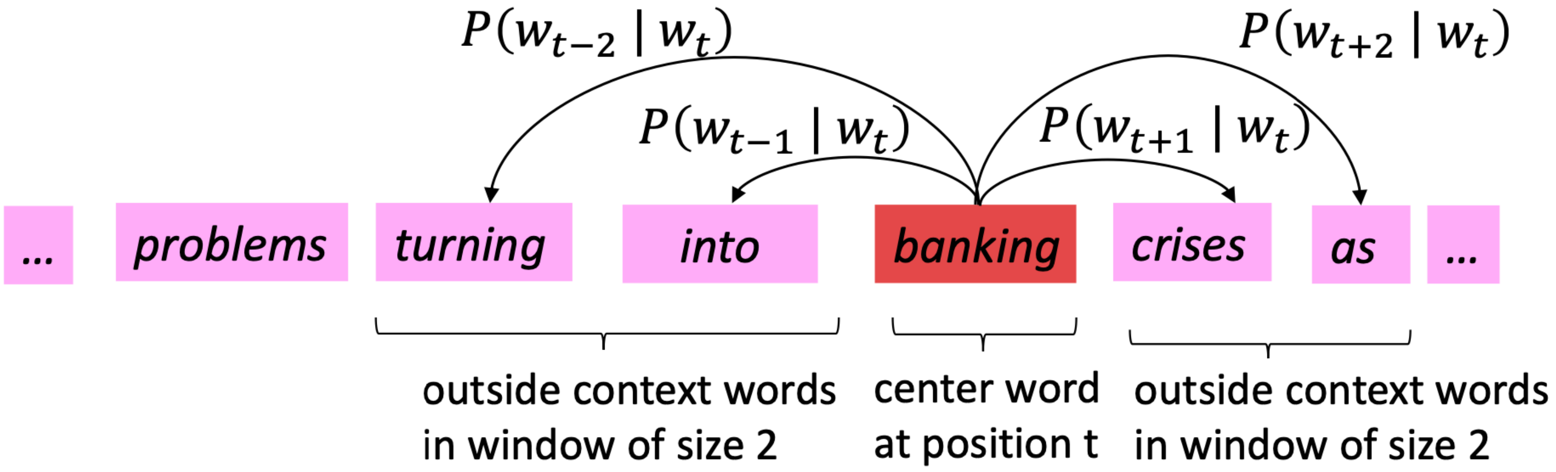
Compute probability $P(w_{t+j}|w_t)$, for $j \in \{-2,-1,1,2\}$ when window size is 2



Example from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Use A Moving Window $t \leftarrow t + 1$

Skip-gram: Compute probability $P(w_{t+j}|w_t)$, for $j \in \{-2, -1, 1, 2\}$ when window size is 2



Example from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Continuous Bag-of-Words (CBOW)

$P(w_t | w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$

Aggregate all context words as if a bag of words

I would like **some orange** **juice** **with some** white chocolate chips

2 outside context words | center word | 2 outside context words

Compute only one probability at position t:
$P(w_t | w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$, for window size 2

Word2vec Objective Function (Skip-gram as example)

- Given a data set of T tokens, for each position $t = 1, \ldots, T$, we compute the conditional probability $P(w_{t+j}|w_t)$, for $j \in \{-m, \ldots, m\}$, with window size m
- Then the *likelihood* of data is:

$$\mathcal{L}(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j}|w_t; \theta)$$

θ denotes model parameters, that is, all the word **embeddings** to be learned!

- The objective function (cost/loss) is the negative log-likelihood

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

Question: How to compute $P(w_{t+j}|w_t;\theta)$?

- **Solution**: Use *two* vectors per word w
- When w is a center word, its vector is v_w
- When w is a context (outside) word, its vector is u_w
- Then the conditional probability of context word o given center word c can be computed using **softmax** function:

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

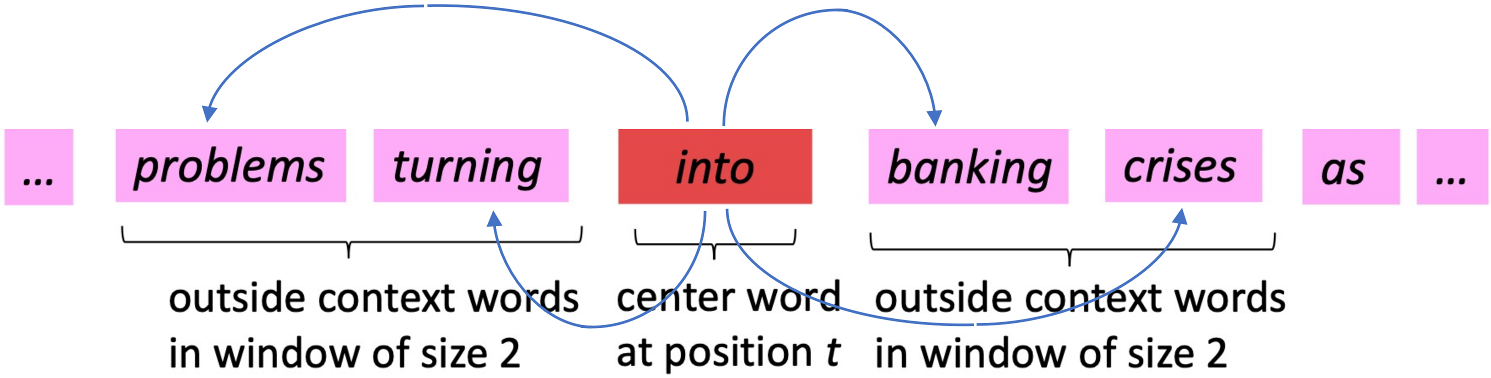
Dot product measures the similarity between o and c

Normalized over the entire vocabulary

Compute probabilities using softmax

$$P(problems|into) = \frac{\exp(u_{problems}^{\top} v_{into})}{\sum_{w \in V} \exp(u_w^{\top} v_{into})}$$

$$P(banking|into) = \frac{\exp(u_{banking}^{\top} v_{into})}{\sum_{w \in V} \exp(u_w^{\top} v_{into})}$$



$$P(turning|into) = \frac{\exp(u_{turning}^{\top} v_{into})}{\sum_{w \in V} \exp(u_w^{\top} v_{into})}$$

$$P(crises|into) = \frac{\exp(u_{crises}^{\top} v_{into})}{\sum_{w \in V} \exp(u_w^{\top} v_{into})}$$

Example from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Number of Parameters

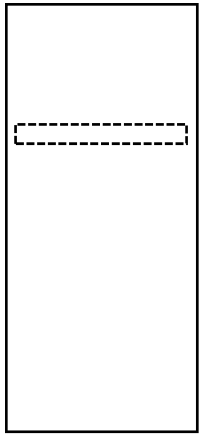
- Because *two* vectors are used per word w: v_w and u_w
- => Two parameter tables, or, embedding matrices

Usually we keep the target table $\mathbf{V}$ as the trained embeddings

Total # of params:
2 * [vocab-size] * [emb-size]

emb size

vocab size

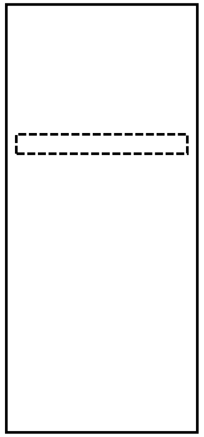


j th row: v_j
target vector
for word w_j

Table $\mathbf{V}$ contains all parameters for **center** vectors

emb size

j th row: u_j
context vector
for word w_j



vocab size

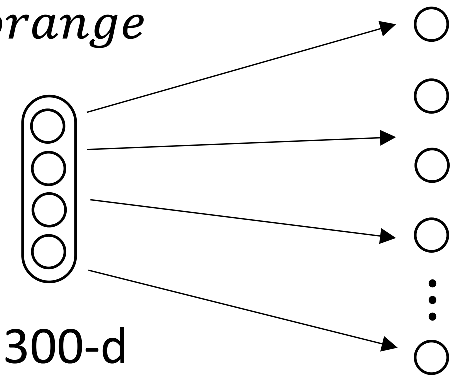
Table $\mathbf{U}$ contains all parameters for **context** vectors

Problem with Softmax

				center	context					
I	*would*	*like*	*some*	*orange*	*juice*	*with*	*some*	*white*	*chocolate*	*chips*

$$P(juice|orange) = \frac{\exp(u_{juice}^{\top} v_{orange})}{\sum_{w \in V} \exp(u_w^{\top} v_{orange})}$$

v_{orange}



300-d

$$z_1 = u_a^{\top} v_{orange}$$

$$z_2 = u_{about}^{\top} v_{orange}$$

$$z_3 = u_{all}^{\top} v_{orange}$$

$$z_j = u_{juice}^{\top} v_{orange}$$

$$z_{10000} = u_{zoo}^{\top} v_{orange}$$

For a vocabulary of 10,000 words

Needs 10,000 times of dot product to compute the denominator

To Overcome Softmax

For vocabulary size $|V| = 10k$

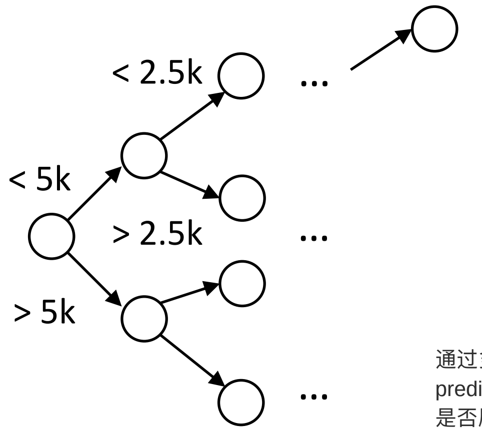
Solutions

1. Hierarchical softmax
2. **Negative sampling**

Make binary predictions instead:

$$P\left(o < \frac{|V|}{2} \middle| c\right)$$

The probability of word o belongs to the 1st half of vocabulary



word o

通过多步二进制预测（binary predictions），每一步判断当前单词是否属于词汇表的某一部分

Multiple steps of binary predictions until word o is found

Time complexity $O(\log(|V|))$

Then $P(o|c) = P(o < 5k|c) \cdot P(o < 2.5k|c) \cdot P(o < 1.25k|c) \ldots$

product of probabilities along the path

Reference: http://ruder.io/word-embeddings-softmax/

Solution 2: Negative sampling intuition

window size = 2 window size = 2

center

I would like some orange juice with some white chocolate chips

Intuition: Given a center word, predict if a *randomly sampled* word is its context or not (within a fixed window)

Center (x)	*Outside Word* (x)	*Label* (y)
orange	*juice*	*1*
orange	*king*	*0*
orange	*the*	*0*
orange	*of*	*0*
orange	*book*	*0*

Step 1: Pick a context word within the window

Positive sample

Step 2: Randomly pick *k* words from the entire vocabulary that do not appear in the window

Negative samples

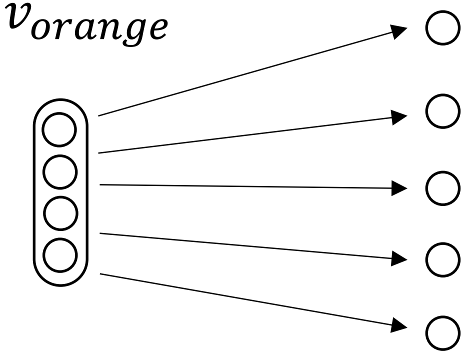
Negative sampling intuition

c Center	o Outside	y Context or not
orange	*juice*	1
orange	*king*	0
orange	*the*	0
orange	*of*	0
orange	*book*	0

Instead of using softmax: $P(o|c) = \frac{\exp(u_o \cdot v_c)}{\sum_{j=1\ldots|V|} \exp(u_j \cdot v_c)} = \hat{y}_t$

Use **logistic regression**: $P(y=1|c,o) = \boldsymbol{\sigma}(u_o \cdot v_c)$

v_{orange}



$\hat{y}_1 = P(juice = 1)\; y_1 = 1$

$\hat{y}_2 = P(king = 1)\; y_2 = 0$

$\hat{y}_3 = P(the = 1)\; y_3 = 0$

$\hat{y}_4 = P(of = 1)\; y_4 = 0$

$\hat{y}_5 = P(book = 1)\; y_5 = 0$

k+1 times of logistic regression

Negative Sampling: Objective Function (loss)

- For token at position t, maximize the log-likelihood:

Word o is the positive sample

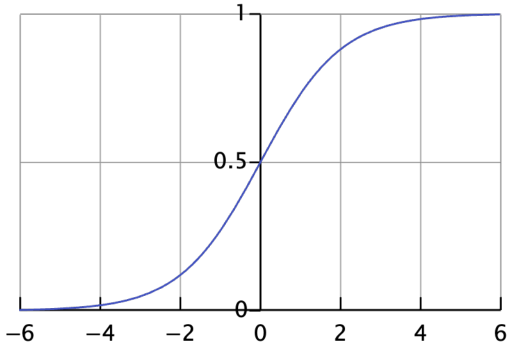
$$J_t(\theta) = \log \sigma(u_o^\top v_c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P(w)} \left[\log \sigma(-u_{w_i}^\top v_c) \right]$$

The k words w_i $(i = 1 \dots k)$ are the negative samples

- Sigmoid function $\sigma(u_o^\top \boldsymbol{v_c})$ outputs the probability of o in the context window of c

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

a monotone increasing function



Maximizing this term will push the dot product $u_o^\top v_c$ to **larger** values, i.e., making o and c closer in semantic space

Maximizing this term will push the dot product $u_{w_i}^\top v_c$ to **smaller** values, i.e., making w_i and c farther apart in semantic space

Loss ⇒ negation of objective

$$\mathcal{L}(\theta) = -J_t(\theta) = -\left[\log\sigma(u_o^\top v_c) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P(w)}\left[\log\sigma\left(-u_{w_i}^\top v_c\right)\right]\right]$$

(Simplify the subscripts)

$$\mathcal{L}(\theta) = -\left[\log\sigma\left(u_{\text{pos}} \cdot v\right) + \sum_{i=1}^{k} \log\sigma\left(-u_{\text{neg}_i} \cdot v\right)\right]$$

u_{pos}, u_{neg}, and v are all learnable parameters

Need to derive derivatives: $\dfrac{\partial\mathcal{L}}{\partial u_{\text{pos}}}, \dfrac{\partial\mathcal{L}}{\partial u_{\text{neg}_i}}, \dfrac{\partial\mathcal{L}}{\partial v}$

Loss: derivatives

$$\mathcal{L}(\theta) = -\left[\log\sigma(u_{\text{pos}} \cdot v) + \sum_{i=1}^{k} \log\sigma(-u_{\text{neg}_i} \cdot v)\right]$$

Using the knowledge:

$$\frac{\partial\sigma(z)}{\partial z} = \sigma(z)(1 - \sigma(z))$$

$$\frac{\partial\mathcal{L}}{\partial u_{\text{pos}}} = (\sigma(u_{\text{pos}} \cdot v) - 1)v$$

$$\frac{\partial\mathcal{L}}{\partial u_{\text{neg}_i}} = \left(\sigma(u_{\text{neg}_i} \cdot v)\right)v$$

$$\frac{\partial\mathcal{L}}{\partial v} = (\sigma(u_{\text{pos}} \cdot v) - 1)u_{\text{pos}} + \sum_{i=1}^{k} \sigma(u_{\text{neg}_i} \cdot v)u_{\text{neg}_i}$$

Gradients update with SGD

- Start with randomly initialized U and V matrices, and do the updates

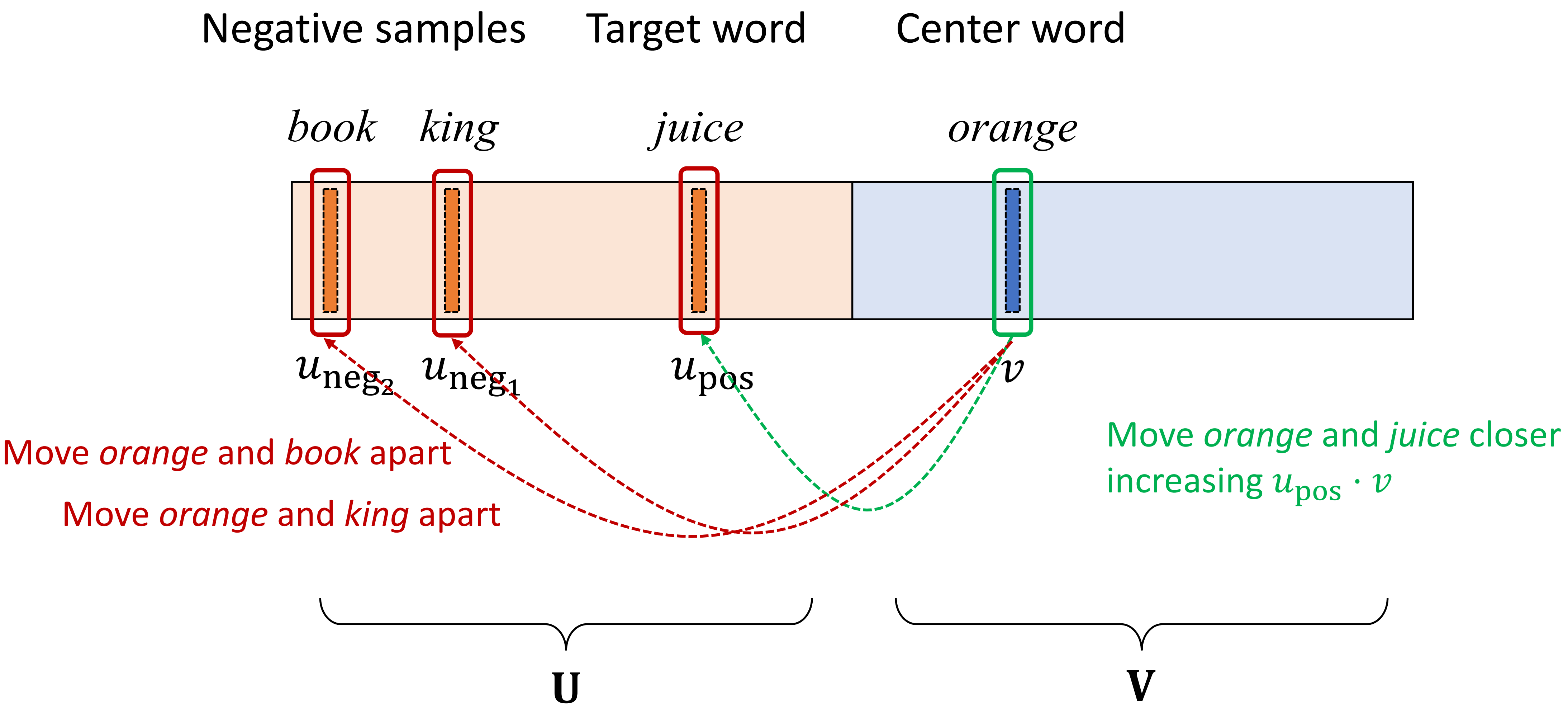
$$u_{\text{pos}}^{t+1} = u_{\text{pos}}^{t} - \alpha\left(\sigma\left(u_{\text{pos}}^{t} \cdot v^{t}\right) - 1\right)v^{t}$$

$$u_{\text{neg}_i}^{t+1} = u_{\text{neg}_i}^{t} - \alpha\left(\sigma\left(u_{\text{neg}_i}^{t} \cdot v^{t}\right)\right)v^{t}$$

$$v^{t+1} = v^{t} - \alpha\left[\left(\left(\sigma\left(u_{\text{pos}}^{t} \cdot v^{t}\right) - 1\right) - 1\right)u_{\text{pos}}^{t} + \sum_{i=1}^{k}\sigma\left(u_{\text{neg}_i}^{t} \cdot v^{t}\right)u_{\text{neg}_i}^{t}\right]$$

α is the learning rate

Intuition: One step of gradient descent



More details: choosing window size

- **Small windows** (k = 2) : nearest words are syntactically similar words in same taxonomy
  - Nearest neighbor of *Hogwarts*: Sunnydale, Evernight, Blandings (Other school names)

- **Large windows** (k = 5) : nearest words are related words in same semantic field
  - Nearest neighbor of *Hogwarts*: Dumbledore, half-blood, Malfoy (entities in the HP world)

More Details : Sample less frequent words

- Maximize probability that *real* outside word appears;
- Minimize probability that *random* words appear around center word
- Sample from the distribution $P(w) = \frac{U(w)^{\frac{3}{4}}}{Z}$, the unigram frequency distribution $U(w)$ raised to the ¾ power (Z is normalization term)
- The power makes less frequent words be sampled more often
- $0.9^{3/4} \approx 0.924$ => a 2.7% increase in chance being sampled
- $0.1^{3/4} \approx 0.178$ => a 77.8% increase in chance being sampled

Why word2vec works?

- Levy and Goldberg, 2014, Neural Word Embedding as Implicit Matrix Factorization

Why Skip-gram negative sampling (**SGNS**) works?

Consider the product $\boldsymbol{V} \cdot \boldsymbol{U}^{\top} = \boldsymbol{M}$

entry $M_{ij} = f(v_i \cdot u_j)$ (f is unknown)

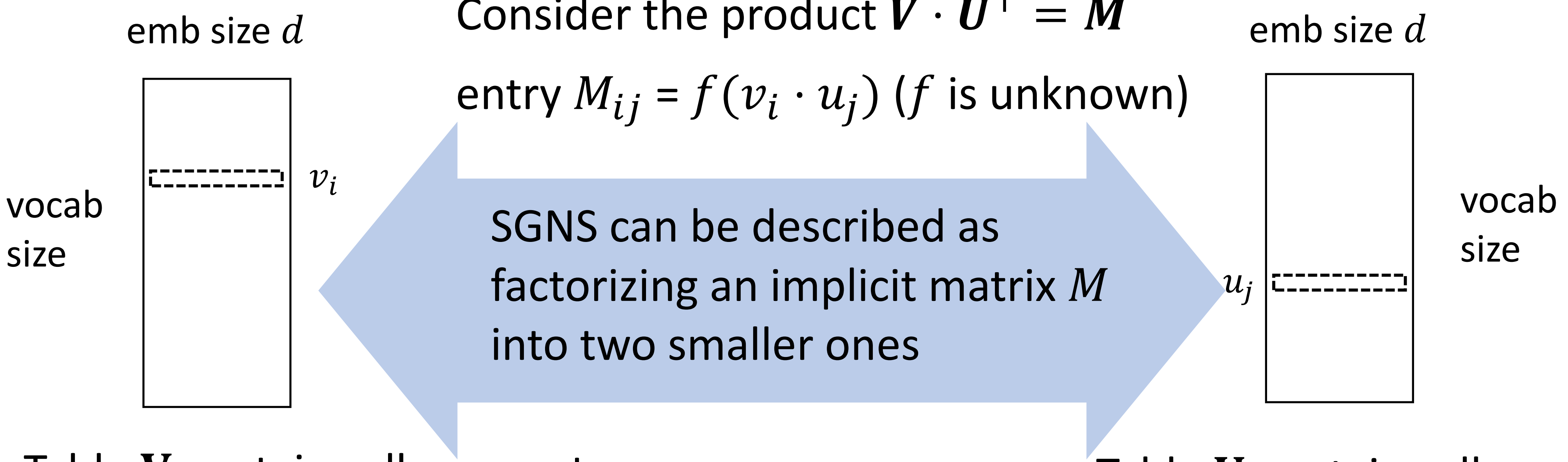


Table $\mathbf{V}$ contains all parameters for **center** vectors

Table $\mathbf{U}$ contains all parameters for **context** vectors

Word2vec as implicit matrix factorization

Equations from Levy and Goldberg, 2014

- How to find the unknown function $M_{ij} = f(v_i \cdot u_j)$?
- Start from the SGNS loss:

$$\ell = \sum_{w\in V_W} \sum_{c\in V_C} \#(w,c) \left(\log \sigma(\vec{w} \cdot \vec{c})\right) + \sum_{w\in V_W} \sum_{c\in V_C} \#(w,c) \left(k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]\right)$$

$$= \sum_{w\in V_W} \sum_{c\in V_C} \#(w,c) \left(\log \sigma(\vec{w} \cdot \vec{c})\right) + \sum_{w\in V_W} \#(w) \left(k \cdot \mathbb{E}_{c_N \sim P_D} [\log \sigma(-\vec{w} \cdot \vec{c}_N)]\right)$$

This eq from Levy and Goldberg (2014) use different terms:

$w\in V_W \Rightarrow v_c \in V$

$c\in V_C \Rightarrow u_o \in U$

#(w, c) denotes the number of times the pair (w,c) appears in D

P_D is the prob. distr. to sample NSs

Word2vec as implicit matrix factorization

Equations from Levy and Goldberg, 2014

- Explicitly express the expectation term:

$$\mathbb{E}_{c_N \sim P_D}\left[\log \sigma(-\vec{w} \cdot \vec{c}_N)\right] = \sum_{c_N \in V_C} \frac{\#(c_N)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}_N)$$

$$= \frac{\#(c)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}) + \sum_{c_N \in V_C \setminus \{c\}} \frac{\#(c_N)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c}_N)$$

- Get the local loss for a specific (w, c) pair:

$$\ell(w, c) = \#(w, c) \log \sigma(\vec{w} \cdot \vec{c}) + k \cdot \#(w) \cdot \frac{\#(c)}{|D|} \log \sigma(-\vec{w} \cdot \vec{c})$$

Word2vec as implicit matrix factorization

Equations from Levy and Goldberg, 2014

- Let $x = w \cdot c$, and calculate the partial derivative:

$$\frac{\partial \ell}{\partial x} = \#(w,c) \cdot \sigma(-x) - k \cdot \#(w) \cdot \frac{\#(c)}{|D|} \cdot \sigma(x)$$

- Compare the derivative to zero, with some simplification:

$$e^{2x} - \left(\frac{\#(w,c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} - 1 \right) e^{x} - \frac{\#(w,c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} = 0$$

Word2vec as implicit matrix factorization

Equations from Levy and Goldberg, 2014

- If we let $y = e^x$, it becomes a quadratic equation of y
- and the solution is:

$$y = \frac{\#(w,c)}{k \cdot \#(w) \cdot \frac{\#(c)}{|D|}} = \frac{\#(w,c) \cdot |D|}{\#w \cdot \#(c)} \cdot \frac{1}{k}$$

numerator: among all occurrences of c, the chance of w co-occurs with it

- Substituting y with e^x and x with $w \cdot c$:

$$\log\left(\frac{\#(w,c)/\#(c)}{\#(w)/|D|}\right)$$

$$\vec{w} \cdot \vec{c} = \log\left(\frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)} \cdot \frac{1}{k}\right) = \log\left(\frac{\#(w,c) \cdot |D|}{\#(w) \cdot \#(c)}\right) - \log k$$

denominator: prior (global) probability of w

It is exactly the point-wise mutual information (PMI) between w and c!

Word2vec SGNS related to PMI

- Recall PMI from LSA

$$[\mathbf{A}]_{v,c} = \left[\log \frac{\text{count}_{x_c}(v)}{\frac{\text{count}_x(v)}{N} \cdot \ell_c}\right]_+ = \left[\log \frac{\frac{\text{count}_{x_c}(v)}{\ell_c}}{\frac{\text{count}_x(v)}{N}}\right]_+$$

Local probability

Global probability

- Finally, we can describe the matrix M that SGNS is factorizing:

$$M_{ij}^{\text{SGNS}} = W_i \cdot C_j = \vec{w}_i \cdot \vec{c}_j = PMI(w_i, c_j) - \log k$$

- When k = 1, SGNS is factorizing a word-context matrix, in which the unknown association function between w and c is $f(w, c) = PMI(w, c)$
- When k > 1, SGNS is factorizing a *shifted* PMI matrix

GloVe: From a different perspective

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.

- Start from the word-word co-occurrence counts X
  - X_{ij} is the number of times word j occurs in the context of word i

$$\begin{array}{cc} & j = \text{steam} \\ i = \text{ice} & \begin{bmatrix} & \vdots & \\ \cdots & X_{ij} & \cdots \\ & \vdots & \end{bmatrix} \end{array}$$

Let $X_i = \sum_k X_{ik}$ be the number of times any word occurs in the context of word i

Let $P_i = P(j|i) = X_{ij}/X_i$ be the probability that word j occurs in the context of word i

GloVe: a showcase

Pennington, J., Socher, R., & Manning, C. (2014)

- Consider i = ice and j = steam

$$
\begin{array}{cc}
 & \begin{array}{cc} k = \text{solid} & k = \text{gas} \end{array} \\
\begin{array}{c} i = \text{ice} \\ j = \text{steam} \end{array} & \begin{bmatrix} P_{ik} & P_{ik} \\ P_{jk} & P_{jk} \end{bmatrix}
\end{array}
$$

The relationship between ice and steam can be examined by studying the *ratio of their co-occurrence probabilities* with various **probe** words k

E.g., for $k = $ solid (related to ice but not steam)

we expect the ratio $\frac{P_{ik}}{P_{jk}}$ to be large, as $P_{ik} \gg P_{jk}$

For $k = $ gas (related to steam but not ice)

we expect the ratio $\frac{P_{ik}}{P_{jk}}$ to be small, as $P_{ik} \ll P_{jk}$

GloVe: a showcase

Pennington, J., Socher, R., & Manning, C. (2014)

- For words k that are either related to both, say water
- or to neither, say fashion

$$\begin{array}{c c} & \begin{array}{cc} k = \text{water} & k = \text{fashion} \end{array} \\ \begin{array}{c} i = \text{ice} \\ j = \text{steam} \end{array} & \begin{bmatrix} \cdots & P_{ik} & P_{ik} \\ \cdots & P_{jk} & P_{jk} \end{bmatrix} \end{array}$$

The ratio should $\frac{P_{ik}}{P_{jk}}$ be close to one, as $P_{ik} \approx P_{jk}$

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

GloVe: introduce the basic idea

- The appropriate starting point for learning word vectors should be the **ratios of co-occurrence probabilities** rather than the probabilities themselves.
- The most general embedding model takes the form:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \Rightarrow F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- The final form takes several steps further following a few desiderata

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}} \Rightarrow F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} \Rightarrow F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

$$\Downarrow$$

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i)$$

GloVe

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.

Cost function：

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w^{\top}{}_i \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2$$

Dot product of two embeddings

Frequency counts of word i and j co-occur (within a fixed window)

Basic idea: words that appear together more often (larger X_{ij}) should have closer meanings (larger dot product)

Advantages: Fast training; scalable to large corpra

GloVe: interesting connection to word2vec

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.

- Choice for the weighting-function $f(\cdot)$

Cost function： $J = \sum_{i,j=1}^{V} \boldsymbol{f(X_{ij})}(w^{\top}{}_i\tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2$

1. $f(0) = 0$, as $x \to 0$, it should vanish fast so that $\lim_{x\to 0} f(x) \log x$ is finite
2. $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted
3. $f(x)$ should be small for larger x, so that frequent co-occurrences ("of", "the") are not overweighted

GloVe: interesting connection to word2vec

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.

- Choice for the weighting-function $f(\cdot)$

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w^{\top}{}_i \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij}))^2$$

$$f(x) = \begin{cases} (x/x_{\max})^{\alpha} & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$$

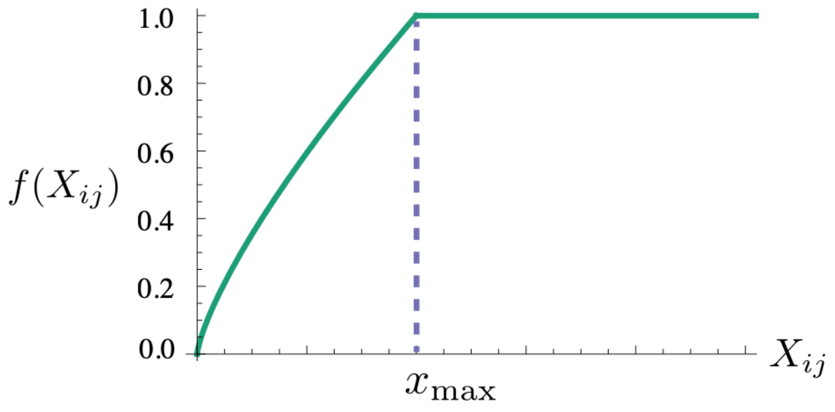


Figure 1: Weighting function f with $\alpha = 3/4$.

It is the similar fractional power scaling used for negative sampling in word2vec

$$P(w) = \frac{U(w)^{\frac{3}{4}}}{Z} \qquad \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P(w)} \left[\log \sigma\left(-u_{w_i}^{\top} v_c\right)\right]$$

Content

- Motivation
- Documents and Counts-based Method
- Neural Network-based Method -- word2vec
- **Evaluation and Applications**

General Evaluation in NLP

- Intrinsic (内在的) vs. Extrinsic (外在的)
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Not clear if really helpful unless correlation to real task is found
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction with other subsystems

Adapted from: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1224/

Evaluate Word Vectors (Embeddings)

- Intrinsic task: Word semantic similarity task

$d_1 = \cos(v_{book}, v_{library})$, cosine similarity

Word1	Word2	Human score	Cosine distance
book	library	7.46	d1
bank	money	8.12	d2
wood	forest	7.73	d3
professor	cucumber	0.31	d4
…	…	…	…

Correlation between the two columns are used to evaluate the quality of word embeddings

Evaluate Word Vectors (Embeddings)

- Intrinsic task: Word analogy task

Question: What is to “King” as “woman” to “man”?

$$v_{Man} - v_{Woman} \approx v_{King} - v_w \quad w = ?$$

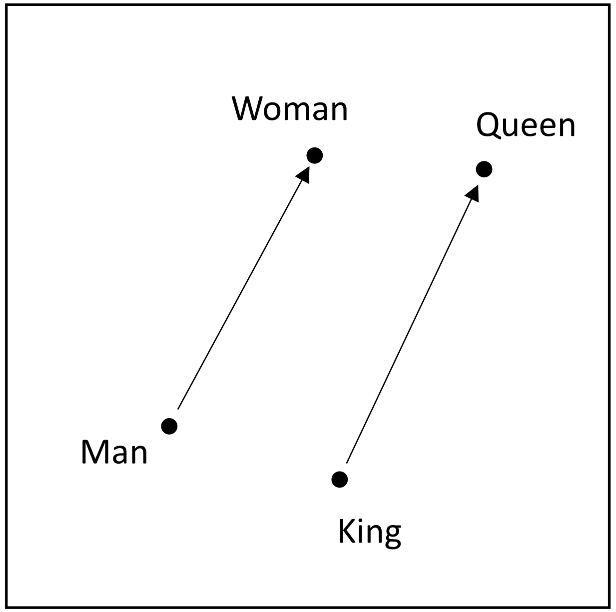
Find the word w so that:

$$\arg\max_{w} sim(v_w, v_{King} - v_{Man} + v_{Woman})$$

Here, $sim()$ is a similarity function, for example, cosine similarity

$$sim(u, v) = \frac{u^T v}{\|u\| \|v\|}$$

Finding the most similar vector v_w will hopefully pick up the word w = Queen



Word Analogy Task (as an interesting application)

Capital-common-countries:

Athens Greece Baghdad **Iraq**
Athens Greece Bangkok **Thailand**
Athens Greece Beijing **China**
Athens Greece Berlin **Germany**
…

Family:

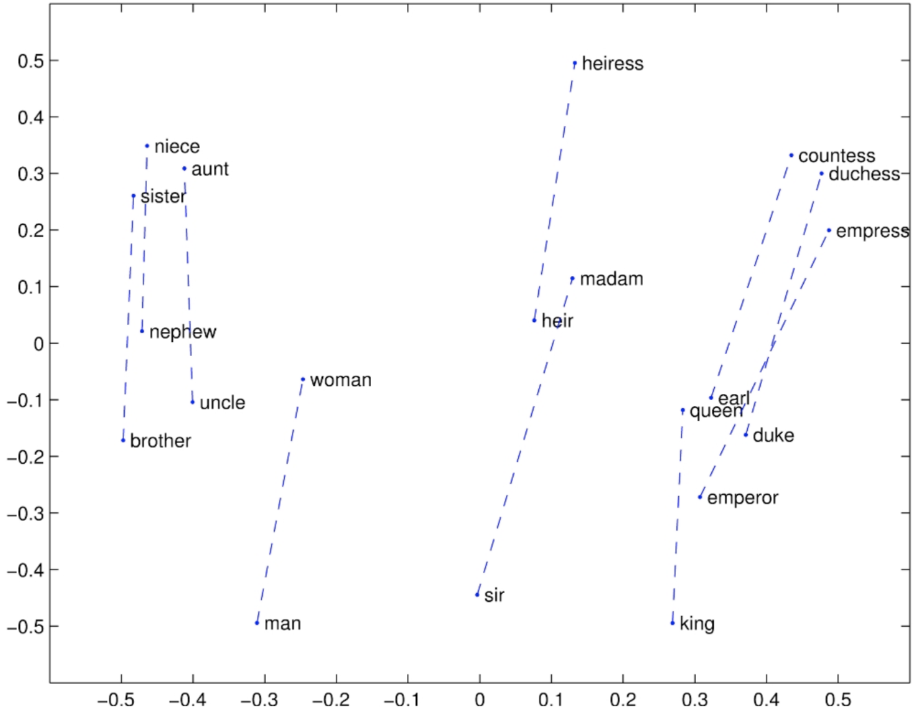
boy girl brother **sister**
boy girl brothers **sisters**
boy girl dad **mom**
boy girl father **mother**
…

Comparative:

bad worse big bigger
bad worse bright brighter
bad worse cheap cheaper
bad worse cold colder
…

City-in-state:

Chicago Illinois Houston **Texas**
Chicago Illinois Philadelphia **Pennsylvania**
Chicago Illinois Phoenix **Arizona**
Chicago Illinois Dallas **Texas**
…



70 - 80 % accuracy reported in Mikolov et al., 2013

Figure from: Pennington et al. (2014). Glove: Global vectors for word representation.

How to use embeddings

- Load pretrained word2vec/glove embeddings to initialize parameters

Binary or text format

```
import torch
import torch.nn as nn
import numpy as np
from gensim.models import KeyedVectors

# Load pretrained embeddings (Word2Vec format)
word_vectors = KeyedVectors.load_word2vec_format("path/to/word2vec.bin", binary=True)

# Define vocabulary (example: words mapped to indices)
vocab = {"hello": 0, "world": 1, "goodbye": 2}  # Example vocab
vocab_size = len(vocab)
embedding_dim = word_vectors.vector_size  # Must match pretrained embedding size

# Initialize an embedding matrix
embedding_matrix = np.zeros((vocab_size, embedding_dim))

# Fill the embedding matrix with pretrained word vectors
for word, idx in vocab.items():
    if word in word_vectors:
        embedding_matrix[idx] = word_vectors[word]
    else:
        embedding_matrix[idx] = np.random.normal(scale=0.6, size=(embedding_dim,))  # F

# Convert to torch tensor
embedding_tensor = torch.tensor(embedding_matrix, dtype=torch.float)

# Initialize nn.Embedding with pretrained weights
embedding_layer = nn.Embedding.from_pretrained(embedding_tensor, freeze=False)  # Set f
```

Fun Application: Emoji2vec

Eisner, B., Rocktäschel, T., Augenstein, I., Bošnjak, M., & Riedel, S. (2016). emoji2vec: Learning emoji representations from their description. *arXiv preprint arXiv:1609.08359*.

References

- Gerard Salton, Anita Wong, and Chung-Shu Yang. A vector space model for automatic indexing. Communications of the ACM, 18(11):613–620, 1975.
- Zellig Harris. Distributional structure. Word, 10(23):146–162, 1954.
- J. R. Firth. A synopsis of linguistic theory 1930–1955. In Studies in Linguistic Analysis, pages 1–32. Blackwell, 1957.
- Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. Journal of the American Society for Information Science, 41(6):391–407, 1990.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In Proceedings of ICLR, 2013a. URL http://arxiv.org/pdf/1301.3781.pdf.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In NeurIPS, 2013b.
- Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation.
- Eisner, B., Rocktäschel, T., Augenstein, I., Bošnjak, M., & Riedel, S. (2016). emoji2vec: Learning emoji representations from their description. arXiv preprint arXiv:1609.08359.