CS 6501 Natural Language Processing

Word Embeddings

Yangfeng Ji

Information and Language Processing Lab
Department of Computer Science
University of Virginia

Overview

Distributional Hypothesis

Distributional Hypothesis

The starting point of building word semantic representations:

Distributional hypothesis

Words that occur in the similar contexts tend to have similar meanings

[Jurafsky and Martin, 2019, Chap 06]

Distributional Hypothesis

The starting point of building word semantic representations:

Distributional hypothesis

Words that occur in the similar contexts tend to have similar meanings

Examples

- `to have a splendid time in Rome`
- `to have a wonderful time in Rome`

[Jurafsky and Martin, 2019, Chap 06]

Another Example

Consider the following examples, although we do not know what exactly words are missing, to some extent we can still guess the meanings of those missing words

- `_____ is delicious sauteed with garlic.`
- `_____ is superb over rice.`
- `... _____ leaves with salty sauces ...`

[Jurafsky and Martin, 2019]

Latent Semantic Analysis

Word-document Matrix

For a corpus of d documents over a vocabulary $\mathcal{V}$, the cooccurence matrix is defined as **C**,

$$\begin{aligned} \boldsymbol{C} &= [c_{ij}] \in \mathbb{R}^{v \times d} \\ &= \begin{bmatrix} c_{1,1} & \dots & c_{1,d} \\ \vdots & \ddots & \vdots \\ c_{v,1} & \dots & c_{v,d} \end{bmatrix} \end{aligned} \quad (1)$$

where

- $v = |\mathcal{V}|$ is the size of vocab
- d is the number of the documents
- c_{ij} is the count of word i in document j

Word-document Matrix

Consider the following toy example, where we have eight documents and a vocabulary with eight words

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

Word-document Matrix

Consider the following toy example, where we have eight documents and a vocabulary with eight words

Word	Documents 1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

Two views of this matrix

- Each column $\boldsymbol{d}_i$ is a document (BoW) representation (same as the one used in logistic regression)

Word-document Matrix

Consider the following toy example, where we have eight documents and a vocabulary with eight words

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

Two views of this matrix

- Each column $\boldsymbol{d}_i$ is a document (BoW) representation (same as the one used in logistic regression)
- Each row $\boldsymbol{w}_k$ is a word representation (by considering a context is a *whole* document)

Word Similarity

Now, with the numeric representations of words, we can calculate word similarity numerically

- We can use row vectors $\{\boldsymbol{w}_k\}$ to represent words by considering each document as a context,

Word Similarity

Now, with the numeric representations of words, we can calculate word similarity numerically

- We can use row vectors $\{\boldsymbol{w}_k\}$ to represent words by considering each document as a context,
- A typical way of measuring word similarity is using cosine values, for two word representations $\boldsymbol{w}_k$ and $\boldsymbol{w}_{k'}$, we have

$$\text{cos-sim}(\boldsymbol{w}_k, \boldsymbol{w}_{k'}) = \frac{\boldsymbol{w}_k^\top \boldsymbol{w}_{k'}}{\|\boldsymbol{w}_k\|_2 \cdot \|\boldsymbol{w}_{k'}\|_2} \tag{2}$$

where

- $\boldsymbol{w}_k^\top \boldsymbol{w}_{k'} = \sum_{i=1} w_{k,i} w_{k',i}$
- $\|\boldsymbol{w}_k\|_2 = \sqrt{\langle \boldsymbol{w}_k, \boldsymbol{w}_k \rangle}$

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_1^\top w_2$
- $w_2^\top w_3$

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_1^\top w_2 = 0$
- $w_2^\top w_3 = 0$

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_1^\top w_2 = 0$
- $w_2^\top w_3 = 0$

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

- The sparsity issue will get even worse when we have a large vocab, say, 10K or 50K words

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_1^\top w_2$= 0
- $w_2^\top w_3$= 0

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

- The sparsity issue will get even worse when we have a large vocab, say, 10K or 50K words
- This motivates us to find a way of compressing these sparse *raw* vectors

Two Constraints

New numeric representations of words

(a) should have low-dimensional dense vectors

Two Constraints

New numeric representations of words

(a) should have low-dimensional dense vectors

(b) should contain similar information as the original sparse vectors

Two Constraints

New numeric representations of words

(a) should have low-dimensional dense vectors

(b) should contain similar information as the original sparse vectors

Matrix decomposition on **C** will help us to identify low-dimensional representations

Singular Value Decomposition (SVD)

Using SVD, the word-document matrix $\boldsymbol{C}$ can be decomposed into a multiplication of three matrices

$$\boldsymbol{C} = \boldsymbol{U}_0 \cdot \boldsymbol{\Sigma}_0 \cdot \boldsymbol{V}_0^{\top}. \tag{3}$$

- $\boldsymbol{U}_0 \in \mathbb{R}^{v \times v}$ is an orthonormal matrix
- $\boldsymbol{V}_0 \in \mathbb{R}^{d \times d}$ is an orthonormal matrix
- $\boldsymbol{\Sigma}_0 \in \mathbb{R}^{v \times d}$ is a diagonal matrix — each component on the diagonal is called a singular value

SVD: Example

Given a matrix $\boldsymbol{C}$ as

$$\boldsymbol{C} = \begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \end{bmatrix} \tag{4}$$

The decomposition is

$$\boldsymbol{U} = \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 5.46 & 0 \\ 0 & 0.37 \end{bmatrix} \quad \boldsymbol{V}^{\top} = \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix} \tag{5}$$

To obtain a low-dimensional approximation of $\boldsymbol{C}$, we can remove one of the singular values. But what matters is which one we are going to remove?

SVD for Low-dimensional Approximation

- Option 1: remove the first singular value

$$C_1 = \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0.37 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix}$$

SVD for Low-dimensional Approximation

- Option 1: remove the first singular value

$$C_1 = \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0.37 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix}$$

SVD for Low-dimensional Approximation

- Option 1: remove the first singular value

$$
\begin{aligned}
C_1 &= \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0.37 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix} \\
&= 0.37 \cdot \begin{bmatrix} -0.91 \\ 0.40 \end{bmatrix} \cdot [0.82 \quad -0.58] = \begin{bmatrix} -0.28 & 0.20 \\ 0.12 & -0.09 \end{bmatrix}
\end{aligned}
$$

SVD for Low-dimensional Approximation

- Option 1: remove the first singular value

$$C_1 = \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0.37 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix}$$

$$= 0.37 \cdot \begin{bmatrix} -0.91 \\ 0.40 \end{bmatrix} \cdot [0.82 \quad -0.58] = \begin{bmatrix} -0.28 & 0.20 \\ 0.12 & -0.09 \end{bmatrix}$$

- Option 2: remove the second singular value

$$C_2 = \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 5.46 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix}$$

$$= 5.46 \cdot \begin{bmatrix} -0.40 \\ -0.91 \end{bmatrix} \cdot [-0.58 \quad -0.82] = \begin{bmatrix} 1.26 & 1.79 \\ 2.88 & 4.07 \end{bmatrix}$$

SVD for Low-dimensional Approximation

- Option 1: remove the first singular value

$$
\begin{aligned}
C_1 &= \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 0 & 0 \\ 0 & 0.37 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix} \\
&= 0.37 \cdot \begin{bmatrix} -0.91 \\ 0.40 \end{bmatrix} \cdot [0.82 \quad -0.58] = \begin{bmatrix} -0.28 & 0.20 \\ 0.12 & -0.09 \end{bmatrix}
\end{aligned}
$$

- Option 2: remove the second singular value

$$
\begin{aligned}
C_2 &= \begin{bmatrix} -0.40 & -0.91 \\ -0.91 & 0.40 \end{bmatrix} \cdot \begin{bmatrix} 5.46 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} -0.58 & -0.82 \\ 0.82 & -0.58 \end{bmatrix} \\
&= 5.46 \cdot \begin{bmatrix} -0.40 \\ -0.91 \end{bmatrix} \cdot [-0.58 \quad -0.82] = \begin{bmatrix} 1.26 & 1.79 \\ 2.88 & 4.07 \end{bmatrix}
\end{aligned}
$$

Therefore, $\|C - C_1\|_F > \|C - C_2\|_F$. In other words, removing the smaller singular value creates a better low-dimensional approximation.

SVD: Example (Cont.)

Given a matrix $\boldsymbol{C}$ as

$$\boldsymbol{C} = \begin{bmatrix} 1.0 & 2.0 \\ 3.0 & 4.0 \\ 5.0 & 6.0 \end{bmatrix} \tag{6}$$

The decomposition is

$$\boldsymbol{U} = \begin{bmatrix} 0.23 & -0.88 & 0.41 \\ 0.52 & -0.24 & -0.82 \\ 0.82 & 0.40 & 0.41 \end{bmatrix} \tag{7}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 9.53 & 0 \\ 0 & 0.51 \\ 0 & 0 \end{bmatrix} \tag{8}$$

$$\boldsymbol{V}^{\top} = \begin{bmatrix} 0.62 & 0.78 \\ 0.78 & -0.62 \end{bmatrix} \tag{9}$$

The maximum number of non-zero singular values is $\min(v, d)$, where v and d are the numbers of rows and columns respectively.

SVD in General Form

The full decomposition of matrix $\boldsymbol{C}$

$$\boldsymbol{C} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_v \\ | & & | \end{bmatrix}}_{\boldsymbol{U}_0} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_? \end{bmatrix}}_{\boldsymbol{\Sigma}_0} \cdot \underbrace{\begin{bmatrix} — & \boldsymbol{v}_1 & — \\ & \vdots & \\ — & \boldsymbol{v}_d & — \end{bmatrix}}_{\boldsymbol{V}_0^\top} \tag{10}$$

As $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$ are both orthonormal matrices, $\boldsymbol{\Sigma}_0$ is the only one that reflects the "magnitude" of matrix $\boldsymbol{C}$.

SVD in General Form

The full decomposition of matrix $\boldsymbol{C}$

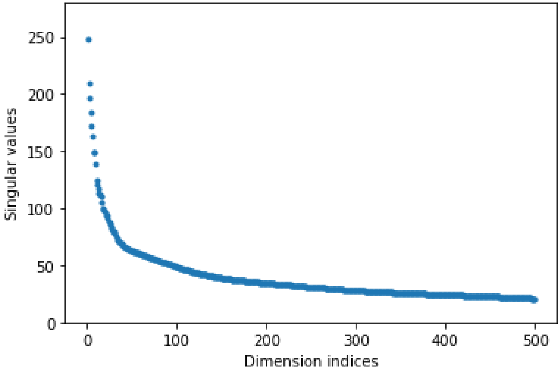
$$\boldsymbol{C} = \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_v \\ | & & | \end{bmatrix}}_{\boldsymbol{U}_0} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_? \end{bmatrix}}_{\boldsymbol{\Sigma}_0} \cdot \underbrace{\begin{bmatrix} — & \boldsymbol{v}_1 & — \\ & \vdots & \\ — & \boldsymbol{v}_d & — \end{bmatrix}}_{\boldsymbol{V}_0^\top} \tag{10}$$

As $\boldsymbol{U}_0$ and $\boldsymbol{V}_0$ are both orthonormal matrices, $\boldsymbol{\Sigma}_0$ is the only one that reflects the "magnitude" of matrix $\boldsymbol{C}$.

For a large-scale sparse matrix, the singular values in $\boldsymbol{\Sigma}_0$ often have big differences.

Singular Values

A real example: C with about 9K words and 71.8K sentences is constructed from a dataset used in the demo code. The following plot shows the first/top 500 singular values in the decreasing order.



With the index $\rightarrow 9K$, the singular values are close to 0.

SVD for Approximation

With SVD, we can approximate $\boldsymbol{C}$ only keep the first k singular values in $\boldsymbol{\Sigma}_0$, as $\boldsymbol{\Sigma}$

$$\boldsymbol{C} \approx \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_k \\ | & & | \end{bmatrix}}_{\boldsymbol{U}} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}}_{\boldsymbol{\Sigma}} \cdot \underbrace{\begin{bmatrix} - & \boldsymbol{v}_1 & - \\ & \vdots & \\ - & \boldsymbol{v}_k & - \end{bmatrix}}_{\boldsymbol{V}^\top} \tag{11}$$

where $\boldsymbol{U} \in \mathbb{R}^{v \times k}$, $\boldsymbol{V} \in \mathbb{R}^{d \times k}$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$.

SVD for Approximation

With SVD, we can approximate $\boldsymbol{C}$ only keep the first k singular values in $\boldsymbol{\Sigma}_0$, as $\boldsymbol{\Sigma}$

$$\boldsymbol{C} \approx \underbrace{\begin{bmatrix} | & & | \\ \boldsymbol{u}_1 & \dots & \boldsymbol{u}_k \\ | & & | \end{bmatrix}}_{\boldsymbol{U}} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_k \end{bmatrix}}_{\boldsymbol{\Sigma}} \cdot \underbrace{\begin{bmatrix} — & \boldsymbol{v}_1 & — \\ & \vdots & \\ — & \boldsymbol{v}_k & — \end{bmatrix}}_{\boldsymbol{V}^\top} \tag{11}$$

where $\boldsymbol{U} \in \mathbb{R}^{v \times k}$, $\boldsymbol{V} \in \mathbb{R}^{d \times k}$ and $\boldsymbol{\Sigma} \in \mathbb{R}^{k \times k}$.

For the previous case, we can pick $k \in [200, 400]$ without worrying about losing too much information.

Lower-dimensional Word Representations

Given

$$\boldsymbol{C} \approx \boldsymbol{U} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{V}^{\top} \tag{12}$$

to construct low-dimensional word representation, we can multiply $\boldsymbol{V}$ on both side of equation 12 and then have

$$\boldsymbol{W} = \boldsymbol{U} \cdot \boldsymbol{\Sigma} \approx \boldsymbol{C} \cdot \boldsymbol{V} \in \mathscr{R}^{v \times k} \tag{13}$$

Dataset

We collected the dataset from the abstracts of NLP papers from the arXiv website. Some example sentences from the dataset

- `The author uses the entropy of the ideal Bose-Einstein gas to minimize losses in computer-oriented languages.`
- `In this paper, current dependency based treebanks are introduced and analyzed.`
- `The model of semantic concept lattice for data mining of microblogs has been proposed in this work.`

This dataset includes about 1.6M tokens.

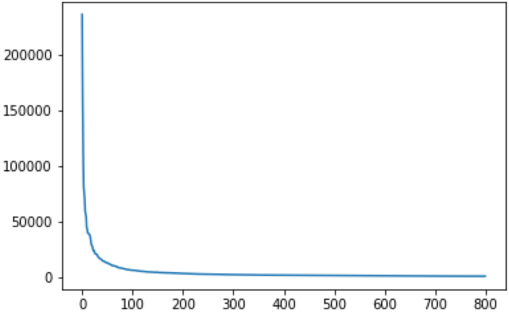
Results

- The size of the matrix C: 8909 words, 71K sentences
- Word embedding dimension: 50
- Word similarity is calculated by the cosine value between two word vectors

natural	embeddings
processing	word
language	contextualized
understanding	glove
nlu	sense
fundamental	embedding
nlg	vectors
vision	disambiguation
sign	analogy

Re-weighting: Motivation

Word frequency in the descreasing order



Top words: `the`, `and`, `to`, `was`, `it`

Re-weighting: TF-IDF

- Term frequency $\mathrm{tf}_{w,d}$: the number of the word w in the document d

$$\mathrm{tf}_{w,d} = \#(w, d) \tag{14}$$

Re-weighting: TF-IDF

- Term frequency $\text{tf}_{w,d}$: the number of the word w in the document d

$$\text{tf}_{w,d} = \#(w, d) \tag{14}$$

- Document frequency df_w: the number of documents that the word w occurs in
- Inverse document frequency

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{15}$$

where N is the total number of documents

Re-weighting: TF-IDF

- Term frequency $\text{tf}_{w,d}$: the number of the word w in the document d

$$\text{tf}_{w,d} = \#(w, d) \tag{14}$$

- Document frequency df_w: the number of documents that the word w occurs in
- Inverse document frequency

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{15}$$

where N is the total number of documents

- TF-IDF weighted value: for word w in document d, the corresponding value in the matrix $\mathbf{C}$ is

$$c_{w,d} = \text{tf}_{w,d} \cdot \text{idf}_w \tag{16}$$

Re-weighting: TF-IDF

- Term frequency $\text{tf}_{w,d}$: the number of the word w in the document d

$$\text{tf}_{w,d} = \#(w, d) \tag{14}$$

- Document frequency df_w: the number of documents that the word w occurs in
- Inverse document frequency

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{15}$$

where N is the total number of documents

- TF-IDF weighted value: for word w in document d, the corresponding value in the matrix $\mathbf{C}$ is

$$c_{w,d} = \text{tf}_{w,d} \cdot \text{idf}_w \tag{16}$$

- Factorize the weighted matrix using SVD

Context Window Size

Distributional hypothesis

Words that occur in the similar contexts tend to have similar meanings

Word	Documents							
	1	2	3	4	5	6	7	8
w_1	0	1	0	0	0	0	0	0
w_2	0	0	1	0	0	3	0	0
w_3	1	0	0	2	0	0	5	0
w_4	3	0	0	1	1	0	2	0
w_5	0	1	3	0	1	2	1	0
w_6	1	2	0	0	0	0	1	0
w_7	0	1	0	1	0	1	0	1
w_8	0	0	0	0	0	7	0	0

Are w_i and w_j similar to each other, when they appear in the same documents but far away from each other?

Context Window Size (II)

Just under a week ago, Apple released a supplemental update to macOS Catalina with various bug fixes and performance improvements. Now, Apple has made a revised version of that same supplemental update available to users.

On its developer website, Apple says that a new version of the macOS Catalina supplemental update has been released today. If you installed the original supplemental update released last week, you might not even receive today's revised version with Apple focusing on people who hadn't yet installed the initial supplemental update.

The release notes for today's update, build 19A603, are exactly the same as last week's:

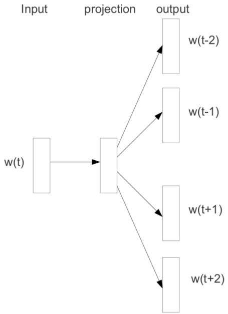
- Improves installation reliability of macOS Catalina on Macs with low disk space
- Fixes an issue that prevented Setup Assistant from completing during some installations
- Resolves an issue that prevents accepting iCloud Terms and Conditions when multiple iCloud accounts are logged in
- Improves the reliability of saving Game Center data when playing Apple Arcade games offline

The revised version of the macOS Catalina supplemental update likely includes very minor changes and fixes. Apple is also currently beta testing macOS Catalina 10.15.1, which may have provided our first look at the forthcoming 16-inch MacBook Pro.

The Skip-gram Model

The Skip-gram Model

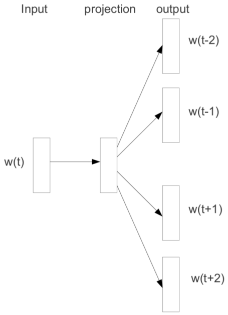
Instead of using matrix decomposition, a different strategy of learning word embeddings is using a word w_t to predict its surrounding words w_{t+i}



[Mikolov et al., 2013a]

The Skip-gram Model

Instead of using matrix decomposition, a different strategy of learning word embeddings is using a word w_t to predict its surrounding words w_{t+i}



In probabilistic form, we need

$$P(w_{t+i} \mid w_t) = ? \tag{17}$$

[Mikolov et al., 2013a]

Skip-gram

One way of finding a better word representation is to make sure it has the potential to predict its surrounding words

$$P(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^\top \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_t})} \tag{18}$$

where $i \in \{-c, \ldots, -1, 1, \ldots, c\}$ and c is the window size.

Skip-gram

One way of finding a better word representation is to make sure it has the potential to predict its surrounding words

$$P(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^\top \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_t})} \tag{18}$$

where $i \in \{-c, \ldots, -1, 1, \ldots, c\}$ and c is the window size.

- $t = 6, c = 2$

Skip-gram

One way of finding a better word representation is to make sure it has the potential to predict its surrounding words

$$P(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^\top \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_t})} \tag{18}$$

where $i \in \{-c, \dots, -1, 1, \dots, c\}$ and c is the window size.

- $t = 6, c = 2$
- Usually, larger window size c gives better quality of word representations, but it also causes large computational complexity.

Skip-gram

One way of finding a better word representation is to make sure it has the potential to predict its surrounding words

$$P(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^\top \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_t})} \tag{18}$$

where $i \in \{-c, \ldots, -1, 1, \ldots, c\}$ and c is the window size.

- $t = 6, c = 2$
- Usually, larger window size c gives better quality of word representations, but it also causes large computational complexity.
- Unlike LSA, the skip-gram model always considers local context.

Word Vectors vs. Context Vectors

Distinguish a word as target (input) and context (output):

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{19}$$

The definition in equation 19 requires two sets of parameters for the same vocabulary

- $\boldsymbol{v}_w$: word vector (as input)
- $\boldsymbol{u}_w$: context vector (as output)

Word Vectors vs. Context Vectors

Distinguish a word as target (input) and context (output):

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{19}$$

The definition in equation 19 requires two sets of parameters for the same vocabulary

- $\boldsymbol{v}_w$: word vector (as input)
- $\boldsymbol{u}_w$: context vector (as output)

Quiz

Why we need two vectors for a word?

Word Vectors vs. Context Vectors

Distinguish a word as target (input) and context (output):

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{19}$$

The definition in equation 19 requires two sets of parameters for the same vocabulary

- $\boldsymbol{v}_w$: word vector (as input)
- $\boldsymbol{u}_w$: context vector (as output)

Quiz

Why we need two vectors for a word? Assume we only use one set of the parameter $\{\boldsymbol{v}_w\}$

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{v}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{v}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{20}$$

Word Vectors vs. Context Vectors

Distinguish a word as target (input) and context (output):

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{19}$$

The definition in equation 19 requires two sets of parameters for the same vocabulary

- $\boldsymbol{v}_w$: word vector (as input)
- $\boldsymbol{u}_w$: context vector (as output)

Quiz

Why we need two vectors for a word? Assume we only use one set of the parameter $\{\boldsymbol{v}_w\}$

$$p(w_{t+i} \mid w_t; \boldsymbol{\theta}) = \frac{\exp(\boldsymbol{v}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{v}_{w'}^{\top} \boldsymbol{v}_{w_t})} \tag{20}$$

A trivial solution that maximize the (log-)probability is $\boldsymbol{v}_{w_{t+i}} = \boldsymbol{v}_w$, which means all words will have the exactly same embedding.

Objective Function

The objective function of a skip-gram model is defined as

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\le i\le c;i\neq 0}\log p(w_{t+i}\mid w_t) \tag{21}$$

Objective Function

The objective function of a skip-gram model is defined as

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq i\leq c;i\neq 0}\log p(w_{t+i}\mid w_t) \tag{21}$$

Each log probability is defined as

$$\log p(w_{t+i}\mid w_t) \quad = \quad \log\frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top}\boldsymbol{v}_{w_t})}{\sum_{w'\in\mathcal{V}}\exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_t})}$$

Objective Function

The objective function of a skip-gram model is defined as

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq i\leq c; i\neq 0}\log p(w_{t+i}\mid w_t) \tag{21}$$

Each log probability is defined as

$$\begin{aligned}\log p(w_{t+i}\mid w_t) &= \log\frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top}\boldsymbol{v}_{w_t})}{\sum_{w'\in\mathcal{V}}\exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_t})}\\ &= \boldsymbol{u}_{w_{t+i}}^{\top}\boldsymbol{v}_{w_t} - \log\sum_{w'\in\mathcal{V}}\exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_t})\end{aligned}$$

Objective Function

The objective function of a skip-gram model is defined as

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq i\leq c;i\neq 0}\log p(w_{t+i}\mid w_t) \tag{21}$$

Each log probability is defined as

$$\begin{aligned}\log p(w_{t+i}\mid w_t) &= \log\frac{\exp(\boldsymbol{u}_{w_{t+i}}^{\top}\boldsymbol{v}_{w_t})}{\sum_{w'\in\mathcal{V}}\exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_t})}\\ &= \boldsymbol{u}_{w_{t+i}}^{\top}\boldsymbol{v}_{w_t}-\log\sum_{w'\in\mathcal{V}}\exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_t})\end{aligned}$$

Essentially, this is learning a classifier over a huge number of classes. In practice, the vocab size could be 10K, 50K or even bigger, the normalization of prediction probability is the major bottleneck.

Negative Sampling

Review what have discussed so far

- The ultimate goal is learning word representations instead of a classifier
- The normalization of prediction probability is computationally expensive

Negative Sampling

Review what have discussed so far

- The ultimate goal is learning word representations instead of a classifier
- The normalization of prediction probability is computationally expensive

To reduce the computational complexity, we can replace

$$\log p(w_{t+i} \mid w_t) = \boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t} - \log \sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})$$

with the following function as objective

$$\log \sigma(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t}) - \sum_{i=1}^{k} \log \sigma(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})\Big|_{w' \sim p_n(w)} \tag{22}$$

where k is the number of negative samples and $\sigma(\cdot)$ is the Sigmoid function (the one used for binary classification in lecture 02)

Basic Training Procedure

Example with $t = 6$, $i = 1$, and $k = 3$

`... finding a better word representation ...`

w_6	w_7	negative samples
better	word	larger
		cause
		window

Basic Training Procedure

Example with $t = 6$, $i = 1$, and $k = 3$

```
... finding a better word representation ...
```

w_6	w_7	negative samples
better	word	larger
		cause
		window

For a given word w_t and i

1. Treat its neighboring context word w_{t+i} as positive example
2. Randomly sample k other words from the vocab as negative examples
3. Optimize Equation 22 to update both $\boldsymbol{v}_{\cdot}$ and $\boldsymbol{u}_{\cdot}$

Two Factors in Negative Sampling

There are two factors that can affect the model performance [Mikolov et al., 2013a]

$$\log \sigma(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t}) - \sum_{i=1}^{k} \log \sigma(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})\Big|_{w' \sim p_n(w)} \qquad (23)$$

Two Factors in Negative Sampling

There are two factors that can affect the model performance [Mikolov et al., 2013a]

$$\log \sigma(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t}) - \sum_{i=1}^{k} \log \sigma(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})\Big|_{w' \sim p_n(w)} \tag{23}$$

- The size of negative samples k
  - $5 \leq k \leq 20$ works better for small datasets
  - $2 \leq k \leq 5$ is enough for large datasets

Two Factors in Negative Sampling

There are two factors that can affect the model performance [Mikolov et al., 2013a]

$$\log \sigma(\boldsymbol{u}_{w_{t+i}}^{\top} \boldsymbol{v}_{w_t}) - \sum_{i=1}^{k} \log \sigma(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_t})\Big|_{w' \sim p_n(w)} \quad (23)$$

- The size of negative samples k
  - $5 \leq k \leq 20$ works better for small datasets
  - $2 \leq k \leq 5$ is enough for large datasets
- Noisy distribution $p_n(w)$
  - $p_n(w) \propto \text{unigram-distribution}(w)^{\frac{3}{4}}$

Results

- Context window size: 3
- Word embedding dimension: 50
- Epochs of training: 3

natural	embeddings
processing	contextualized
nlp	embedding
nl	representations
language	vectors
understanding	elmo
nlu	static
nlg	word
fundamental	polyglot

Word Embeddings: GloVe

Glove

The motivation of GloVe [**?**] is to find a balance between the methods based on

- global matrix factorization (e.g., LSA) and
- local context windows (e.g., Skip-gram).

Word-to-word Co-occurrence Matrix

- Define $\mathbf{X}$ with $X_{i,j}$ denotes the frequency of word j appears in the context of word i

$$\mathbf{X} = \begin{bmatrix} \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ X_{i,1} & \dots & X_{i,j-1} & X_{i,j} & X_{i,j+1} & \dots & X_{i,V} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \end{bmatrix} \qquad (24)$$

Each row corresponds one target word, each column corresponds one context word.

Word-to-word Co-occurrence Matrix

- Define $\mathbf{X}$ with $X_{i,j}$ denotes the frequency of word j appears in the context of word i

$$\mathbf{X} = \begin{bmatrix} \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ X_{i,1} & \cdots & X_{i,j-1} & X_{i,j} & X_{i,j+1} & \cdots & X_{i,V} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix} \quad (24)$$

  Each row corresponds one target word, each column corresponds one context word.

- Empirical probability estimation of w_j given w_i

$$Q(w_j \mid w_i) = \frac{X_{ij}}{X_i} \quad (25)$$

  where $X_i = \sum_j X_{i,j}$

Probability Estimation via Word Embeddings

Another way to estimate the probability of w_j given w_i is

$$P(w_j \mid w_i) = \frac{\exp(\boldsymbol{u}_{w_j}^\top \boldsymbol{v}_{w_i})}{\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_i})} \tag{26}$$

with $\boldsymbol{u}_\cdot$ and $\boldsymbol{v}_\cdot$ are two sets of parameters (embeddings) associated with words, similar to the Skip-gram model.

GloVe

The basic idea is to learn $\{\boldsymbol{v}_\cdot\}$ and $\{\boldsymbol{u}_\cdot\}$, such that

$$Q(w_j \mid w_i) \approx P(w_j \mid w_i) \tag{27}$$

or

$$\log Q(w_j \mid w_i) \approx \log P(w_j \mid w_i) \tag{28}$$

GloVe

The basic idea is to learn $\{\boldsymbol{v}_{\cdot}\}$ and $\{\boldsymbol{u}_{\cdot}\}$, such that

$$Q(w_j \mid w_i) \approx P(w_j \mid w_i) \tag{27}$$

or

$$\log Q(w_j \mid w_i) \approx \log P(w_j \mid w_i) \tag{28}$$

More specific

$$\log(X_{ij}) - \log(X_i) \approx \boldsymbol{u}_{w_j}^{\top} \boldsymbol{v}_{w_i} - \log \sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_i}) \tag{29}$$

GloVe (II)

Starting point:

$$\log(X_{ij}) - \log(X_i) \approx \boldsymbol{u}_{w_j}^{\top} \boldsymbol{v}_{w_i} - \log \sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top} \boldsymbol{v}_{w_i}) \tag{30}$$

GloVe (II)

Starting point:

$$\log(X_{ij}) - \log(X_i) \approx \boldsymbol{u}_{w_j}^\top \boldsymbol{v}_{w_i} - \log \sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_i}) \tag{30}$$

In order to find the best approximation, we could formulate this as a optimization problem

$$\left\{ \log(X_{ij}) - \log(X_i) - \boldsymbol{u}_{w_j}^\top \boldsymbol{v}_{w_i} + \log \sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^\top \boldsymbol{v}_{w_i}) \right\}^2 \tag{31}$$

GloVe (II)

Starting point:

$$\log(X_{ij}) - \log(X_i) \approx \boldsymbol{u}_{w_j}^{\top}\boldsymbol{v}_{w_i} - \log\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_i}) \tag{30}$$

In order to find the best approximation, we could formulate this as a optimization problem

$$\left\{\log(X_{ij}) - \log(X_i) - \boldsymbol{u}_{w_j}^{\top}\boldsymbol{v}_{w_i} + \log\sum_{w' \in \mathcal{V}} \exp(\boldsymbol{u}_{w'}^{\top}\boldsymbol{v}_{w_i})\right\}^2 \tag{31}$$

It can be further simplified as (Eq. 16 in [**?**])

$$\left\{\log(X_{ij}) - \boldsymbol{u}_{w_j}^{\top}\boldsymbol{v}_{w_i}\right\}^2 \tag{32}$$

if we only consider the unnormalized version of P and Q.

Objective Function

The overall objective function is defined as

$$\sum_{w_i} \sum_{w_j} (\log(X_{ij}) - \boldsymbol{u}_{w_j}^{\mathsf{T}} \boldsymbol{v}_{w_i})^2 \tag{33}$$

Objective Function

The overall objective function is defined as

$$\sum_{w_i} \sum_{w_j} (\log(X_{ij}) - \boldsymbol{u}_{w_j}^{\top} \boldsymbol{v}_{w_i})^2 \tag{33}$$

The objective function is further refined by discouraging high-frequency words as

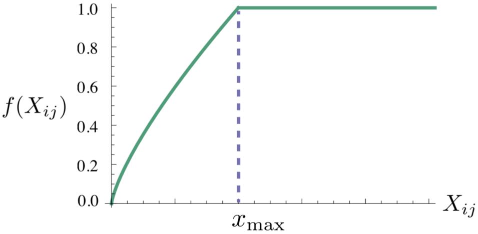
$$\sum_{w_i} \sum_{w_j} f(X_{ij})(\log(X_{ij}) - \boldsymbol{u}_{w_j}^{\top} \boldsymbol{v}_{w_i})^2 \tag{34}$$

Down-weighting

Weighting function:

$$f(x) = \begin{cases} (\frac{x}{x_{\max}})^a & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} \tag{35}$$

where $a = 3/4$.



Skip-gram as Implicit Matrix Factorization

[**?**] shows that skip-gram with negative sampling can be viewed as an implicit matrix factorization over a word-word co-occurrence matrix weighted by point-wise mutual information (PMI).

$$\boldsymbol{u}_{w_j}^{\top}\boldsymbol{v}_{w_i} \approx \mathrm{PMI}(w_i, w_j) - \log k \tag{36}$$

where $\mathrm{PMI}(w_i, w_j)$ is the mutual information of $P(w_i)$ and $P(w_j)$ with *a given window size* and k is the number of negative samples.

Skip-gram as Implicit Matrix Factorization (II)

The definition of $\mathrm{PMI}(w_i, w_j)$ is

$$\mathrm{PMI}(w_i, w_j) = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \log P(w_j \mid w_i) - \log P(w_j) \tag{37}$$

Skip-gram as Implicit Matrix Factorization (II)

The definition of $\text{PMI}(w_i, w_j)$ is

$$\text{PMI}(w_i, w_j) = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \log P(w_j \mid w_i) - \log P(w_j) \tag{37}$$

Combine 36 and 37, we have

$$\begin{aligned} \boldsymbol{u}_{w_j}^{\top} \boldsymbol{v}_{w_i} &\approx \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} - \log k \\ &= \log P(w_j \mid w_i) - \log P(w_j) - \log k \\ &= \log(X_{ij}) - \log(X_i) - \log(X_j) + \log D - \log k \end{aligned} \tag{38}$$

Similar to Eq. 8 in [**?**].

Essentially,

A unified framework

$$\boldsymbol{u}_{w_j}^{\top}\boldsymbol{v}_{w_i} \approx \log(X_{ij}) + g(\mathbf{X}) \tag{39}$$

Essentially,

A unified framework

$$\boldsymbol{u}_{w_j}^{\mathsf{T}} \boldsymbol{v}_{w_i} \approx \log(X_{ij}) + g(\mathbf{X}) \tag{39}$$

Which one matters?

- $g(\mathbf{X})$, or
- Implicit/explicit optimization, or
- Other tricks (down-sampling, hyper-parameters, etc.)

Evaluation Methods

Overview

- Intrinsic Evaluation[1]
  - Word similarity
  - Word analogy
  - Word intrusion
- Extrinsic Evaluation
  - Evaluating based on a downstream task, such as text classification

[1] http://bionlp-www.utu.fi/wv_demo/

Word Similarity

Let w_i and w_j be two words, and $\boldsymbol{v}_{w_i}$ and $\boldsymbol{v}_{w_j}$ be the corresponding word embeddings, word similarity can be obtained by computing their cosine similarity between $\boldsymbol{v}_{w_i}$ and $\boldsymbol{v}_{w_j}$ as

$$\cos(\boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j}) = \frac{\langle \boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j} \rangle}{\|\boldsymbol{v}_{w_i}\|_2 \cdot \|\boldsymbol{v}_{w_j}\|_2} \tag{40}$$

Examples

$Word_1$	$Word_2$	Similarity score [0,10]
love	sex	6.77
stock	jaguar	0.92
money	cash	9.15
development	issue	3.97
lad	brother	4.46

Figure: Sample word pairs along with their human similarity judgment from WS-353 [Faruqui et al., 2016].

Datasets

Available word similarity datasets

Dataset	Word pairs	Reference
RG	65	Rubenstein and Goodenough (1965)
MC	30	Miller and Charles (1991)
WS-353	353	Finkelstein et al. (2002)
YP-130	130	Yang and Powers (2006)
MTurk-287	287	Radinsky et al. (2011)
MTurk-771	771	Halawi et al. (2012)
MEN	3000	Bruni et al. (2012)
RW	2034	Luong et al. (2013)
Verb	144	Baker et al. (2014)
SimLex	999	Hill et al. (2014)

Figure: Word similarity datasets [Faruqui et al., 2016].

Word Similarity

the basis for other intrinsic evaluations

Word Analogy

- It is sometimes referred as *linguistic regularity* [Mikolov et al., 2013b]
- The basic setup

$$w_a : w_b = w_c :?$$

where $w_{a,b,c}$ are words and w_a, w_b are related under a certain linguistic relation

Word Analogy

- It is sometimes referred as *linguistic regularity* [Mikolov et al., 2013b]
- The basic setup

$$w_a : w_b = w_c :?$$

where $w_{a,b,c}$ are words and w_a, w_b are related under a certain linguistic relation

- Example
  - Semantic: `love` : `like` = `hate` :?
  - Syntactic: `quick` : `quickly` = `happy` :?
  - Gender: `king` : `man` = `queen` :?
  - Others: `Beijing` : `China` = `Paris` :?

Word Analogy

- It is sometimes referred as *linguistic regularity* [Mikolov et al., 2013b]
- The basic setup

$$w_a : w_b = w_c :?$$

  where $w_{a,b,c}$ are words and w_a, w_b are related under a certain linguistic relation
- Example
  - Semantic: `love` : `like` = `hate` :?
  - Syntactic: `quick` : `quickly` = `happy` :?
  - Gender: `king` : `man` = `queen` :?
  - Others: `Beijing` : `China` = `Paris` :?
- Calculation: $(v_{w_a} - v_{w_b})^{\mathsf{T}}(v_{w_c} - v_{w_d})$

Word Analogy: Examples

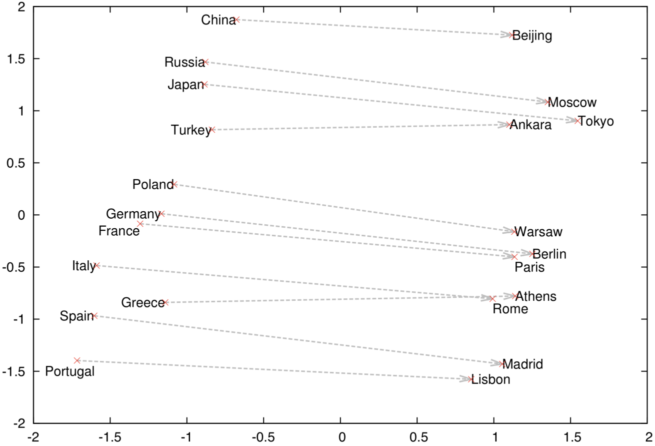


Figure: Word analogy examples.

Word Intrusion

From [Faruqui et al., 2014]

`naval, industrial, technological, marine, identity`

- constructed from word embeddings
- evaluated by human annotators

Extrinsic Evaluation

- Implicit assumption: there is a consistent, global ranking of word embedding quality, and that higher quality embeddings will necessarily improve results on *any* downstream task.
- Unfortunately, this assumption does not hold in general [Schnabel et al., 2015].
- Examples
  - empirical results show that it may not be able give much help to syntactic parsing [Andreas and Klein, 2014]
  - adding surface-form features always help ([Ji and Eisenstein, 2014a] and many other works)

Further Discussion

Gender Bias

$$v_{\texttt{man}} - v_{\texttt{woman}} \approx v_{\texttt{computer programmer}} - v_{\texttt{homemaker}} \quad (41)$$

$$v_{\texttt{father}} - v_{\texttt{mother}} \approx v_{\texttt{doctor}} - v_{\texttt{nurse}} \quad (42)$$

[Bolukbasi et al., 2016]

Example

[Bolukbasi et al., 2016]

Problem

- Word embeddings from either Word2vec or GloVe encode not just semantic information

Problem

- Word embeddings from either Word2vec or GloVe encode not just semantic information
- In some applications, we want to emphasize one particular aspect of linguistic information
  - Semantic information [Faruqui et al., 2014, Mrksic et al., 2016]
  - Discourse information [Ji and Eisenstein, 2014b]

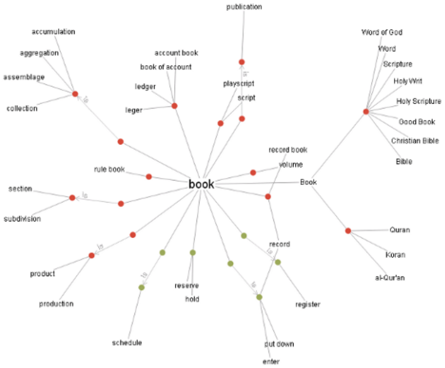
Problem

- Word embeddings from either Word2vec or GloVe encode not just semantic information
- In some applications, we want to emphasize one particular aspect of linguistic information
  - Semantic information [Faruqui et al., 2014, Mrksic et al., 2016]
  - Discourse information [Ji and Eisenstein, 2014b]
- Solutions
  - fine-tuning word embeddings with certain constraints [Faruqui et al., 2014, Mrksic et al., 2016]
  - learning from supervision information [Ji and Eisenstein, 2014b]

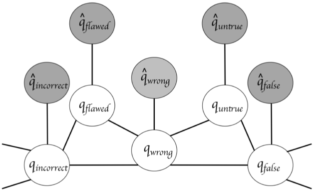
Retrofitting

Retrofitting with WordNet [Miller, 1995]

- $\Omega = (V, E)$ be a semantic graph over words, where V is the node set with each element as a word, and E is the edge set with each edge representing a semantic relation between two words.



Retrofitting (II)



- The goal is to learn word embeddings $\{\tilde{v}\}$ such that $\tilde{v}_i$ and $\tilde{v}_j$ are close enough if $(i, j) \in E$.
- In addition, $\{\tilde{v}\}$ should also satisfy the constraint from original word embeddings, such that $\tilde{v}_i$ and $\tilde{v}_i$ are close enough for every word in $\mathcal{V}$.

$$\Psi(\tilde{\mathbf{V}}) = \sum_{i=1}^{|\mathcal{V}|} \left[\alpha_i \| v_i - \tilde{v}_i \|^2 + \sum_{(i,j)\in E} \beta_{ij} \| \tilde{v}_i - \tilde{v}_j \|^2 \right] \tag{43}$$

Counter-fitting

Inject antonymy and synonymy constraints into word embedding space to improve the embeddings' capability for judging semantic similarity

	east	expensive	British
Before	west	pricey	American
	north	cheaper	Australian
	south	costly	Britain
	southeast	overpriced	European
	northeast	inexpensive	England
After	eastward	costly	Brits
	eastern	pricy	London
	easterly	overpriced	BBC
	-	pricey	UK
	-	afford	Britain

Table 1: Nearest neighbours for target words using GloVe vectors before and after counter-fitting

[Mrksic et al., 2016]

Learning from Supervision Signal

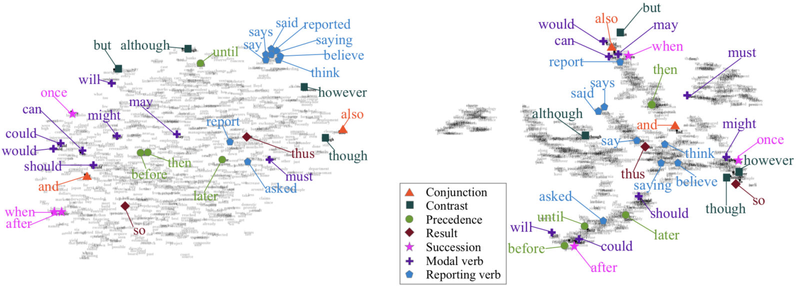


Figure: (Left) Word embeddings learned with supervision signal; (Right) Unsupervised word embeddings.

Reference

Andreas, J. and Klein, D. (2014).
How much do word embeddings encode about syntax?
In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, volume 2, pages 822–827.

Bolukbasi, T., Chang, K.-W., Zou, J. Y., Saligrama, V., and Kalai, A. T. (2016).
Man is to computer programmer as woman is to homemaker? debiasing word embeddings.
In *Advances in Neural Information Processing Systems*, pages 4349–4357.

Faruqui, M., Dodge, J., Jauhar, S. K., Dyer, C., Hovy, E., and Smith, N. A. (2014).
Retrofitting word vectors to semantic lexicons.
arXiv preprint arXiv:1411.4166.

Faruqui, M., Tsvetkov, Y., Rastogi, P., and Dyer, C. (2016).
Problems with evaluation of word embeddings using word similarity tasks.
arXiv preprint arXiv:1605.02276.

Ji, Y. and Eisenstein, J. (2014a).
One vector is not enough: Entity-augmented distributional semantics for discourse relations.
arXiv preprint arXiv:1411.6699.

Ji, Y. and Eisenstein, J. (2014b).
Representation learning for text-level discourse parsing.
In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 13–24.

Jurafsky, D. and Martin, J. (2019).
Speech and language processing.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013a).
Distributed representations of words and phrases and their compositionality.
In *Advances in neural information processing systems*, pages 3111–3119.

Mikolov, T., Yih, W.-t., and Zweig, G. (2013b).