CS 6770 Natural Language Processing

Word Embeddings (I):

Distributed Hypothesis, Latent Semantic Analysis

Yangfeng Ji

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

UVA ILP
UNIVERSITY OF VIRGINIA
INFORMATION AND LANGUAGE PROCESSING LAB

Overview

1. Distributional Hypothesis
2. Latent Semantic Analysis
3. Singular Value Decomposition
4. An Example
5. Evaluation Methods
6. Further Extension

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 great 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

Bag-of-words Representation

Consider the two example sentences

- `I love coffee and their cappuccino is great.`
- `I prefer black coffee.`
- `Which one do you prefer, cappuccino or black coffee?`
- `I think latte is better.`

A bag-of-words representation is a way to represent natural language texts with numeric vectors, which consists of three steps

1. Tokenize texts
2. Build a vocabulary
3. Represent texts as numeric vectors

Word-document Matrix

In general, for a corpus of d documents over a vocabulary $\mathcal{V}$, the cooccurence matrix is defined as $\mathbf{C}$,

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

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 (Cont.)

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

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

Word-document Matrix (Cont.)

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

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

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 (Cont.)

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

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

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 $\{w_k\}$ 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

- $w_{\text{coffee}}^{\mathsf{T}} w_{\text{cappuccino}}$
- $w_{\text{coffee}}^{\mathsf{T}} w_{\text{latte}}$

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_{\text{coffee}}^{\mathsf{T}} w_{\text{cappuccino}}$
- $w_{\text{coffee}}^{\mathsf{T}} w_{\text{latte}}$

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

The Sparsity Issue in Representations

Compute the dot product of the following two pairs

- $w_{\text{coffee}}^{\top} w_{\text{cappuccino}}$
- $w_{\text{coffee}}^{\top} w_{\text{latte}}$

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

- 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

- $w_{\text{coffee}}^{\top} w_{\text{cappuccino}}$
- $w_{\text{coffee}}^{\top} w_{\text{latte}}$

	x_1	x_2	x_3	x_4
i	1	1	0	1
love	1	0	0	0
prefer	0	1	0	0
think	0	0	0	1
coffee	1	1	1	0
cappuccino	1	0	1	0
latte	0	0	0	1

- 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

Map $\boldsymbol{C}$ to a lower-dimensional matrix $\boldsymbol{W} \in \mathbb{R}^{v \times k}$ while preserving as much information as possible

Two Constraints

Map $\boldsymbol{C}$ to a lower-dimensional matrix $\boldsymbol{W} \in \mathbb{R}^{v \times k}$ while preserving as much information as possible

New numeric representations of words

(a) should have low-dimensional dense vectors

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

Singular Value Decomposition

Singular Value Decomposition (SVD)

Using SVD, the word-document matrix C can be decomposed into a multiplication of three matrices

$$C = U_0 \cdot \Sigma_0 \cdot V_0^\top. \tag{3}$$

- $U_0 \in \mathbb{R}^{v \times v}$ is an orthonormal matrix
- $V_0 \in \mathbb{R}^{d \times d}$ is an orthonormal matrix
- $\Sigma_0 \in \mathbb{R}^{v \times d}$ is a diagonal matrix — each component on the diagonal is called a singular value

SVD: Example

Given a matrix C as

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

The decomposition is

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

To obtain a low-dimensional approximation of 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

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

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

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

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

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

- Option 2: remove the second singular value

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

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

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 C as

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

The decomposition is

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

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

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

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 C

$$C = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \dots & u_v \\ | & & | \end{bmatrix}}_{U_0} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_? \end{bmatrix}}_{\Sigma_0} \cdot \underbrace{\begin{bmatrix} — & v_1 & — \\ & \vdots & \\ — & v_d & — \end{bmatrix}}_{V_0^\top} \tag{10}$$

As U_0 and V_0 are both orthonormal matrices, Σ_0 is the only one that reflects the "magnitude" of matrix C.

SVD in General Form

The full decomposition of matrix C

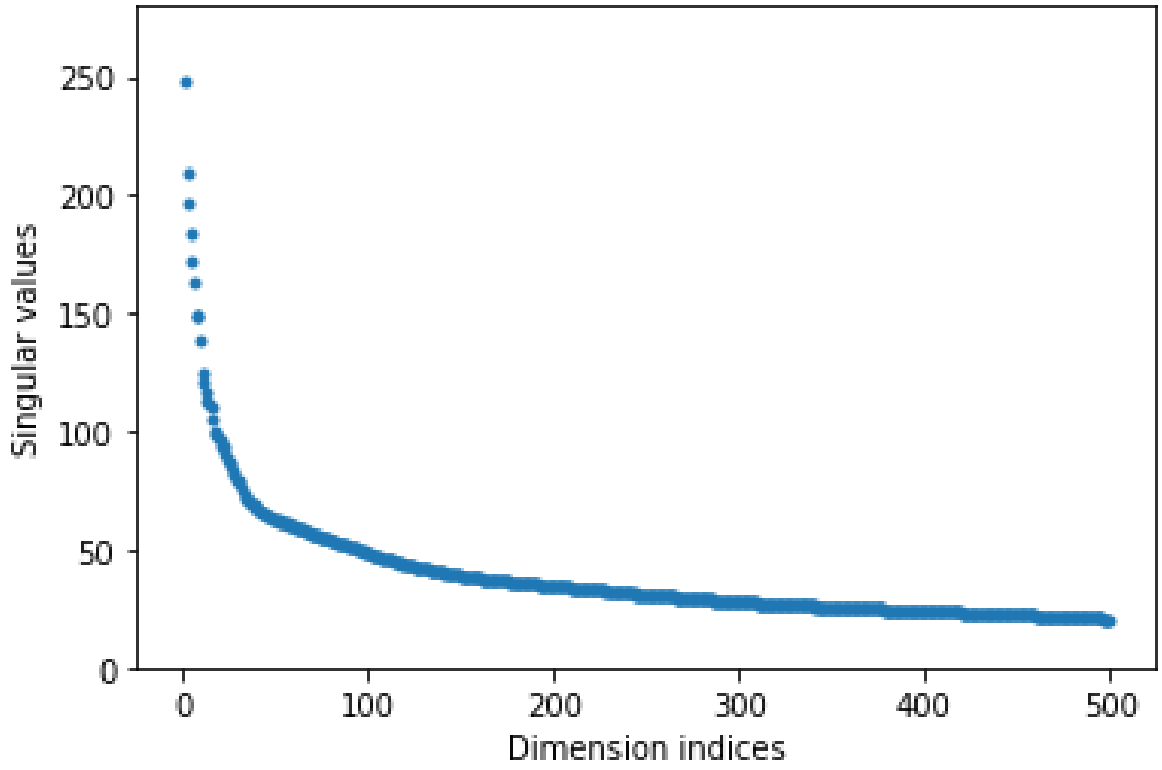
$$C = \underbrace{\begin{bmatrix} | & & | \\ u_1 & \dots & u_v \\ | & & | \end{bmatrix}}_{U_0} \cdot \underbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_? \end{bmatrix}}_{\Sigma_0} \cdot \underbrace{\begin{bmatrix} — & v_1 & — \\ & \vdots & \\ — & v_d & — \end{bmatrix}}_{V_0^\top} \tag{10}$$

As U_0 and V_0 are both orthonormal matrices, Σ_0 is the only one that reflects the "magnitude" of matrix C.

For a large-scale sparse matrix, the singular values in Σ_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

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

Lower-dimensional Word Representations

Given

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

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

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

What if we multiply $\boldsymbol{U}$ on both side of equation 12?

LSA Algorithm

- Construct the word-document matrix $\boldsymbol{C} \in \mathbb{R}^{v \times d}$ from the corpus
- Perform SVD on $\boldsymbol{C}$ and get $\boldsymbol{U}_0$, $\boldsymbol{\Sigma}_0$ and $\boldsymbol{V}_0$
- Pick a k and construct $\boldsymbol{U}$, $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$
- Construct low-dimensional word representations $\boldsymbol{W} = \boldsymbol{U} \cdot \boldsymbol{\Sigma}$

An Example

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

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

$$\cos(\boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j}) = \frac{\langle \boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j} \rangle}{\|\boldsymbol{v}_{w_i}\|_2 \cdot \|\boldsymbol{v}_{w_j}\|_2} \tag{14}$$

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

`http://wordvec.colorado.edu/`

Word Analogy

- It is sometimes referred as *linguistic regularity* [Mikolov et al., 2013]
- 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., 2013]
- 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., 2013]
- 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: $(\boldsymbol{v}_{w_a} - \boldsymbol{v}_{w_b})^\mathsf{T}(\boldsymbol{v}_{w_c} - \boldsymbol{v}_{w_d})$

Word Analogy: Examples

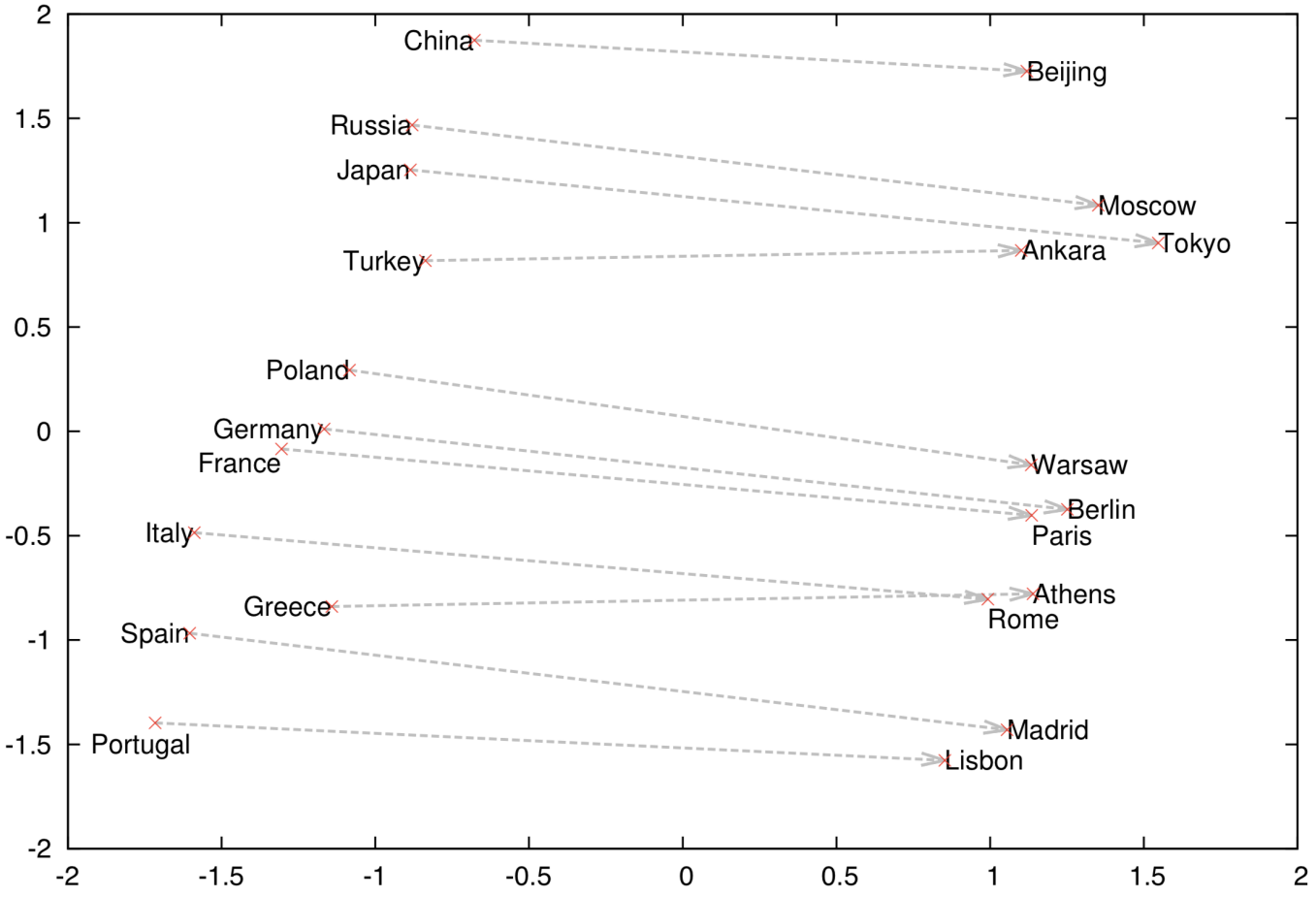


Figure: Word analogy examples.

Word Similarity: Problems (I)

Similarity and relatedness: which pair is closer?

- `train, car`
- `coffee, cup`

Word Similarity: Problems (I)

Similarity and relatedness: which pair is closer?

- `train, car`
- `coffee, cup`

In WS-353, the similarity between `coffee` and `cup` is higher than `train` and `car`.

Word Similarity: Problems (II)

Frequency effects of cosine similarity

- prevents the bias introduced by the norm of a vector 3
- pairs of words that have similar frequency will be closer in the embedding space
  - higher similarity of two words can be given by cosine similarity then they should be based on their word meaning

Word Similarity: Problems (III)

Inability to account for polysemy (one word has multiple meanings)

$$\cos(\boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j}) = \frac{\langle \boldsymbol{v}_{w_i}, \boldsymbol{v}_{w_j} \rangle}{\|\boldsymbol{v}_{w_i}\| \cdot \|\boldsymbol{v}_{w_j}\|} \tag{15}$$

- Encode them into different dimensions?

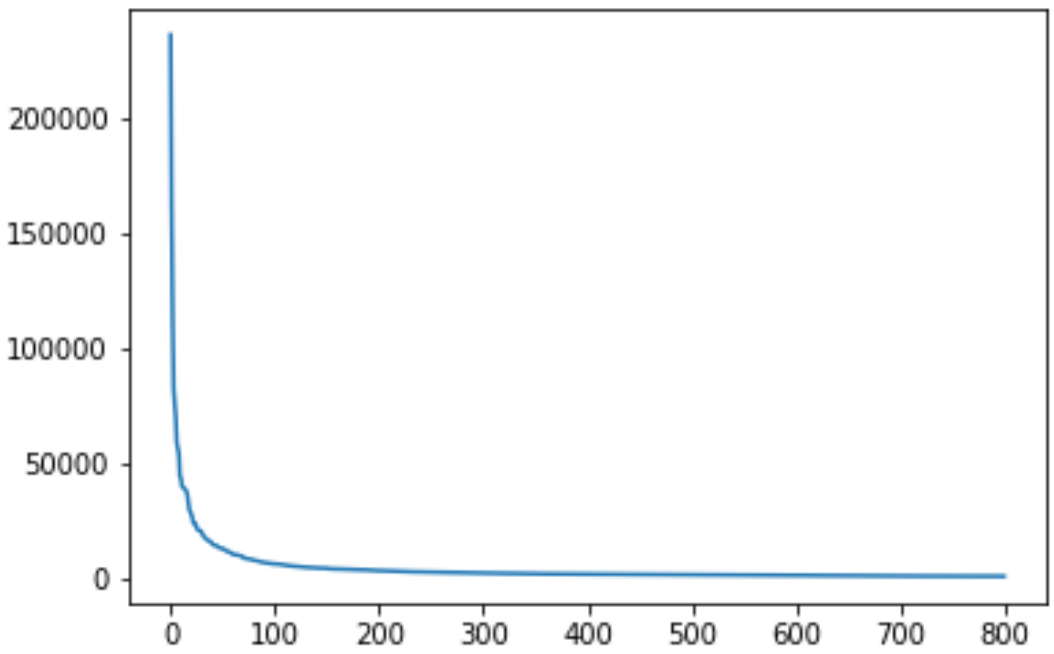
Extrinsic Evaluation

- 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, 2014] and many other works)

Further Extension

Re-weighting: Motivation

Word frequency in the descreasing order



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

Re-weighting: TF-IDF

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

$$\text{tf}_{w,d} = \#(w, d) \tag{16}$$

Re-weighting: TF-IDF

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

$$\text{tf}_{w,d} = \#(w, d) \tag{16}$$

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

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{17}$$

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

$$\text{tf}_{w,d} = \#(w, d) \tag{16}$$

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

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{17}$$

  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

$$c_{w,d} = \text{tf}_{w,d} \cdot \text{idf}_w \tag{18}$$

Re-weighting: TF-IDF

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

$$\text{tf}_{w,d} = \#(w, d) \tag{16}$$

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

$$\text{idf}_w = \log_{10} \frac{N}{\text{df}_w} \tag{17}$$

  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

$$c_{w,d} = \text{tf}_{w,d} \cdot \text{idf}_w \tag{18}$$

- 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.

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.

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. (2014).

One vector is not enough: Entity-augmented distributional semantics for discourse relations.

arXiv preprint arXiv:1411.6699.

Jurafsky, D. and Martin, J. (2019).

Speech and language processing.

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

Linguistic regularities in continuous space word representations.

In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Schnabel, T., Labutov, I., Mimno, D., and Joachims, T. (2015).

Evaluation methods for unsupervised word embeddings.

In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 298–307.