Advanced Text Representations Part 2

Overview

1. Representations in IR
2. Latent semantic indexing
3. Neural network based representations
   - CBOW, skip-gram NNL models and BERT language model
   - Application: word similarity
   - Modeling sequences: recurrent neural networks [see Chapter on Question answering]
   - Open issues
4. Use of word vectors in retrieval models

1. Representations in IR

Representations in IR

1. Unstructured representations

- E.g., text represented as an unordered set of terms (the so-called bag-of-words representation)
- Considerable oversimplication : ignoring syntax and semantics
- Despite oversimplifying, satisfactory retrieval performance

2. Weakly-structured representations

- Certain content has more or additional importance (other content might be downscaled or ignored): e.g., tagged by controlled language terms, e.g., named entities, entity-relationships, opinions, etc. [see chapter on Information extraction]

Representations in IR

3. Latent representations

- Discovering latent concepts/topics in the data
- Inducing structured semantically-aware representations of words, images, queries, documents, etc.
- **Latent semantic indexing**, latent Dirichlet allocation, **word embeddings**, **document embeddings**, …

Representations in IR

The representations are used in **retrieval models**, often as a combination of different representations:

- Vector space model
- Probabilistic retrieval models such as a language model

Possibly combined with additional structured metadata, users' tags, subject classes [see chapter on Categorization], etc.

Vector representations

1. **Local representations**: 1 hot encoding or vector of a word or other content item (used in bag-of-words representation of document): sparse representation

2. **Distributed representations**: representing a word or other content item by its attribute vector that usually integrates context:
   - Sparse representations
   - Dense representations

One-hot representations

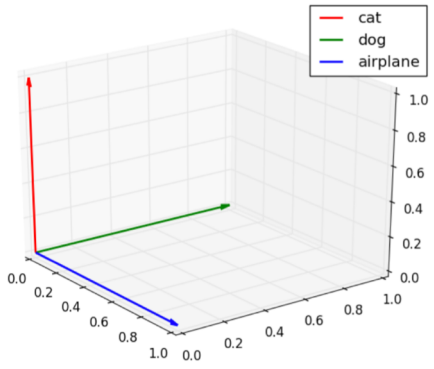
- One-hot = 1-of-*N*:
  - E.g., word is represented as a vector with *N* (= size of vocabulary) components with one component activated and the others zero
- Simple way how to encode discrete concepts, such as words
- There is no notion of similarity between words: every item is very distinct
- Example:

  Vocabulary = (airplane, dog, cat)

  airplaine = [1 0 0]

  dog = [0 1 0]

  cat = [0 0 1]



Bag-of-words representation

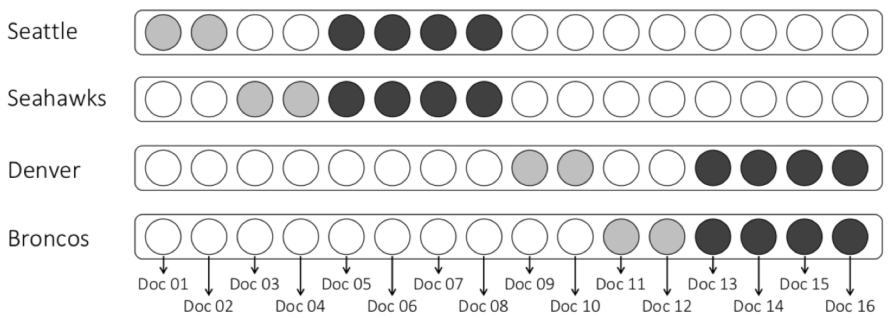
Taking into account term frequency = sum of one-hot codes

Ignores order of words: e.g.,

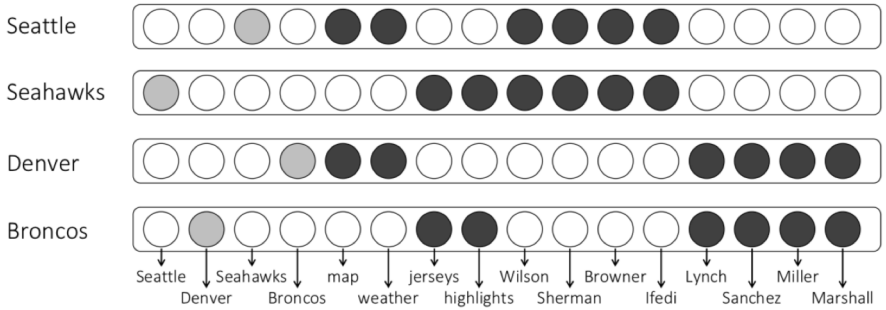
- Vocabulary = (Monday, Tuesday, is, a, today)
- Monday Monday = [2 0 0 0 0]
- today is a Monday = [1 0 1 1 1]
- today is a Tuesday = [0 1 1 1 1]
- is a Monday today = [1 0 1 1 1]

Can be extended to bag-of-*n*-grams to capture local ordering of words

Sparse distributed representations



(a) "In-documents" features



(b) "Neighbouring terms" features

[Mitra & Craswell Intro NIR 2017]

Dense distributed representations

Each word is represented by a dense vector, a point in a vector space

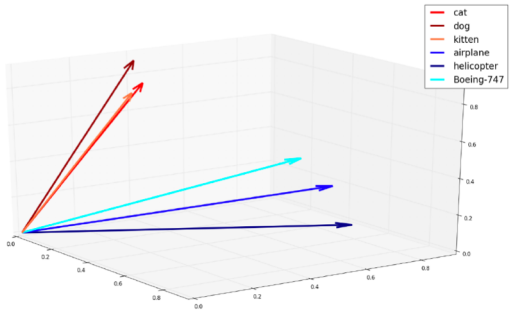
The dimension of the semantic representation *d* is usually much

smaller than the size of the vocabulary ($d \ll N$), and all dimensions

contain real-valued numbers (usually normalized between −1 and 1)

$\mathbf{v}_{word} = [\ldots\ 0.3.\ -0.5\ \ 0.1\ \ \ldots]$

In such a space we want also to represent documents



Dense distributed representations

Examples of dense distributed representations obtained e.g., with:

- Latent semantic indexing
- Neural networks

2. Latent semantic indexing

Latent semantic indexing (LSI)

Latent semantic indexing (LSI) or **Latent semantic analysis**

- To represent queries and documents with vectors:
  - Mapping the term vectors of documents and query into a low dimensional space which is associated with statistical **concepts**
  - This would allow the retrieval of documents even when they are not indexed by the query index terms
- To represent words with vectors into a low dimensional space (less common)

LSI = method for dimensionality reduction using method of Singular Value Decomposition (SVD) [Deerwester et al. 1990]

Singular Value Decomposition example

Given the 4 x 2 matrix **A**

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 2 & -2 \\ -2 & 2 \\ -1 & -1 \end{bmatrix}$$

Compute singular value decomposition: $\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$

Compute $\mathbf{A}^T\mathbf{A}$

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 1 & 2 & -2 & -1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 2 & -2 \\ -2 & 2 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 10 & -6 \\ -6 & 10 \end{bmatrix}$$

Singular Value Decomposition example

This matrix has two eigenvalues. Sort them from large to small.

$\lambda_1 = 16$ $\lambda_2 = 4$

Search for each eigenvalue the normed eigenvector.

$$\lambda_1 = 16: \begin{bmatrix} -6 & -6 \\ -6 & -6 \end{bmatrix} \mathbf{V}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \mathbf{V}_1 = \begin{bmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{bmatrix}$$

$$\lambda_2 = 4: \begin{bmatrix} 6 & -6 \\ -6 & 6 \end{bmatrix} \mathbf{V}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \mathbf{V}_2 = \begin{bmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{bmatrix}$$

The orthogonal matrix $\boldsymbol{V}$ with in its columns $\mathbf{V}_1$ and $\mathbf{V}_2$ are thus found.

Singular Value Decomposition example

The singular values are: $\sigma_1 = \sqrt{\lambda_1} = 4 \qquad \sigma_2 = \sqrt{\lambda_2} = 2$

So matrix $\sum$ is also computed.

Computations matrix **U** based on $\mathbf{AA}^T$ cf. above

$\mathbf{A = U\sum V^T}$

$$\begin{bmatrix} 1 & 1 \\ 2 & -2 \\ -2 & 2 \\ -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 1/\sqrt{2} & 1/\sqrt{2} & 0 \\ 1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \\ -1/\sqrt{2} & 0 & 0 & 1/\sqrt{2} \\ 0 & -1/\sqrt{2} & 1/\sqrt{2} & 0 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 2 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{2} & -1/\sqrt{2} \\ 1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}$$

Latent semantic indexing

= application of Singular Value Decomposition (SVD) to the

term-document matrix $\mathbf{A}_{txm}$

where

t = number of unique index terms in the collection

m = the number of unique documents in the collection

$$\mathbf{A}_{t\times m} =$$

	$d1$	$d2$	$d3$	$d4$	$d5$	$d6$
president	3	2	0	1	0	0
minister	4	1	3	0	0	0
speech	2	5	1	0	0	0
law	0	0	2	0	0	1
ball	0	0	0	4	0	2
score	0	0	0	3	2	3
player	0	0	1	1	4	1
run	0	0	0	0	1	0
person	1	0	0	0	0	1
piano	0	1	0	0	1	0
mouse	0	0	1	1	0	0

Latent semantic indexing

A is decomposed with SVD as the product of 3 matrices:

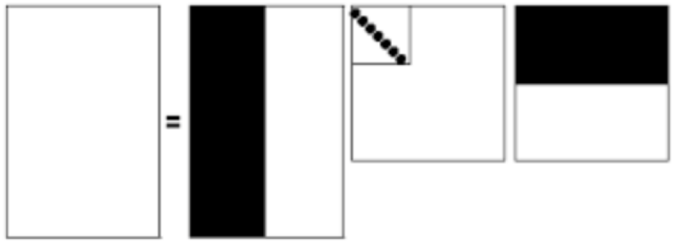
$$\mathbf{A} = \mathbf{U}\sum\mathbf{V}^T \quad \text{where}$$

- $\mathbf{U}_{txt}$ = orthogonal matrix of eigenvectors (left singular vectors) derived from the term-to-term correlation matrix given by $\mathbf{AA}^T$
- $\mathbf{V}_{mxm}$= orthogonal matrix of eigenvectors (right singular vectors) derived from the document-to-document correlation matrix given by $\mathbf{A}^T\mathbf{A}$
- $\sum$ = diagonal matrix with the singular values of **A**: indicates the importance of the corresponding singular vectors in matrices **U** and **V**

$$
\mathbf{U} = \begin{bmatrix}
-0.43 & 0.13 & 0.22 & -0.01 & -0.55 & -0.09 & \\
-0.53 & 0.25 & -0.28 & 0.62 & -0.09 & -0.07 & \\
-0.58 & 0.33 & 0.18 & -0.56 & 0.37 & 0.06 & \\
-0.12 & -0.05 & -0.19 & 0.28 & 0.64 & 0.26 & \\
-0.22 & -0.51 & 0.53 & 0.17 & 0.10 & -0.32 & \\
-0.26 & -0.62 & 0.08 & -0.05 & -0.03 & 0.41 & \cdots \\
-0.22 & -0.40 & -0.69 & -0.25 & -0.12 & -0.21 & \\
-0.03 & -0.06 & -0.18 & -0.11 & -0.12 & -0.07 & \\
-0.11 & -0.03 & 0.02 & 0.13 & -0.18 & 0.60 & \\
-0.10 & -0.02 & -0.12 & -0.29 & 0.01 & -0.06 & \\
-0.09 & -0.08 & 0.01 & 0.16 & 0.26 & -0.47 &
\end{bmatrix}
\begin{matrix}
president \\ minister \\ speech \\ law \\ ball \\ score \\ player \\ run \\ person \\ piano \\ mouse
\end{matrix}
$$

$$\Sigma V^T = \begin{bmatrix} -4.66 & -4.37 & -2.71 & -2.37 & -1.51 & -1.65 \\ 2.01 & 2.12 & 0.49 & -4.23 & -2.93 & -3.35 \\ -0.06 & 0.92 & -1.70 & 1.90 & -2.90 & 0.44 \\ 1.45 & -2.48 & 1.75 & 0.43 & -1.51 & 0.34 \\ -1.44 & 0.68 & 1.53 & -0.09 & -0.64 & 0.46 \\ 0.19 & 0.02 & -0.32 & -0.82 & -0.16 & 1.25 \\ & & & \dots & & \end{bmatrix}$$

Latent semantic indexing



Reduced vector space:

- Select k dimensions as basis for the reduced vector space
- $k < t$ and $k < m$ (k = usually 50-300)
- Singular values are ordered by size: keep the k largest values and keep the corresponding columns from the **U** and **V** matrices: $\mathbf{U}_k$ and $\mathbf{V}_k$

The product of the resulting matrices: $\mathbf{U_k \Sigma_k V_k^T} = \widehat{\mathbf{A}}$

$\widehat{\mathbf{A}}$ represents **A** as in a lower k dimensional space (rank k): in such a way that the representations in the original space are changed as little as possible when measured by the sum of squares of the differences

U

$$\begin{bmatrix} -0.43 & 0.13 & 0.22 & -0.01 & -0.55 & -0.09 \\ -0.53 & 0.25 & -0.28 & 0.62 & -0.09 & -0.07 \\ -0.58 & 0.33 & 0.18 & -0.56 & 0.37 & 0.06 \\ -0.12 & -0.05 & -0.19 & 0.28 & 0.64 & 0.26 \\ -0.22 & -0.51 & 0.53 & 0.17 & 0.10 & -0.32 \\ -0.26 & -0.62 & 0.08 & -0.05 & -0.03 & 0.41 \cdots \\ -0.22 & -0.40 & -0.69 & -0.25 & -0.12 & -0.21 \\ -0.03 & -0.06 & -0.18 & -0.11 & -0.12 & -0.07 \\ -0.11 & -0.03 & 0.02 & 0.13 & -0.18 & 0.60 \\ -0.10 & -0.02 & -0.12 & -0.29 & 0.01 & -0.06 \\ -0.09 & -0.08 & 0.01 & 0.16 & 0.26 & -0.47 \end{bmatrix}$$

ΣV^T

$$\cdot \begin{bmatrix} -4.66 & -4.37 & -2.71 & -2.37 & -1.51 & -1.65 \\ 2.01 & 2.12 & 0.49 & -4.23 & -2.93 & -3.35 \\ -0.06 & 0.92 & -1.70 & 1.90 & -2.90 & 0.44 \\ 1.45 & -2.48 & 1.75 & 0.43 & -1.51 & 0.34 \\ -1.44 & 0.68 & 1.53 & -0.09 & -0.64 & 0.46 \\ 0.19 & 0.02 & -0.32 & -0.82 & -0.16 & 1.25 \\ & & & \cdots & & \end{bmatrix}$$

$\mathbf{U}_k$ $\mathbf{\Sigma}_k \mathbf{V}_k^{\mathrm{T}}$

$$\begin{bmatrix} -0.43 & 0.13 \\ -0.53 & 0.25 \\ -0.58 & 0.33 \\ -0.12 & -0.05 \\ -0.22 & -0.51 \\ -0.26 & -0.62 \\ -0.22 & -0.40 \\ -0.03 & -0.06 \\ -0.11 & -0.03 \\ -0.10 & -0.02 \\ -0.09 & -0.08 \end{bmatrix} \cdot \begin{bmatrix} -4.66 & -4.37 & -2.71 & -2.37 & -1.51 & -1.65 \\ 2.01 & 2.12 & 0.49 & -4.23 & -2.93 & -3.35 \end{bmatrix}$$

Determining a value for *k* is difficult – generally a value which works well on a development set is chosen

Latent semantic indexing

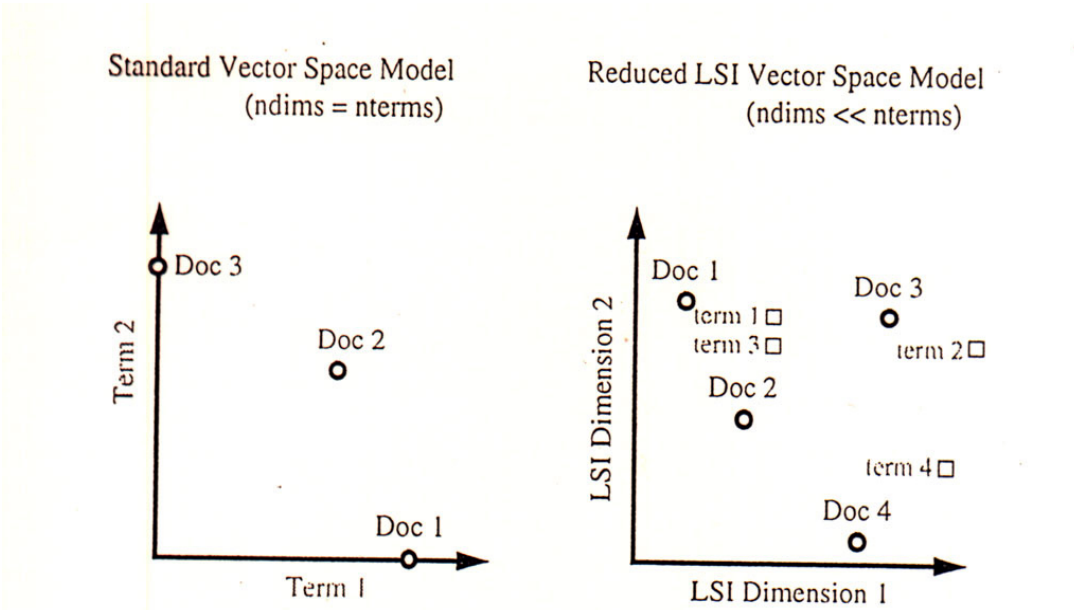


Figure 1 Term representations in the standard vector vs. reduced LSI vector models.

[Grefenstette 1998]

Latent semantic indexing

Document comparison can be done by comparing columns of $\boldsymbol{\Sigma}_k \mathbf{V}_k^T$

A **query vector** $\mathbf{q}$ (or any document not used during training $\mathbf{d}_j$) can be projected into the topic space of documents where they can be compared

LSI for representing documents and queries

Query and (usually) documents are mapped to the lower dimensional space

- In reduced vector space: term in $\mathbf{U}_k$ is represented as linear combination of documents: "synonyms grouped"
- **L** contains weighted singular vectors in $\mathbf{U}_k$, where the weights are the corresponding singular values in $\sum_k$:

$$\mathbf{L} = \mathbf{U}_k \boldsymbol{\Sigma}_k^{-1}$$

-1: inverted values of the singular values

The **similarity** *sim* ($\boldsymbol{d}_j, \boldsymbol{q}$) of the document $\boldsymbol{d}_j$ to the query $\boldsymbol{q}$ is often defined as:

$$\boldsymbol{sim}(\mathbf{d}_j, \mathbf{q}) = \boldsymbol{cos}\left(\mathbf{L}^{\mathrm{T}}\mathbf{d}_j, \mathbf{L}^{\mathrm{T}}\mathbf{q}\right) = \boldsymbol{cos}\left(\hat{\mathbf{d}}_j, \hat{\mathbf{q}}\right)$$

LSI for representing words

Assuming all words occur in the document collection used for training:

- Bag-of-words representation: each row of matrix **A** represents a word/term: similarity between 2 words can be computed as the cosine of 2 row vectors
- LSI: each row of matrix $\mathbf{U}_k$ represents a word/term in the semantic space with k dimensions: similarity between 2 words can be computed as the cosine of 2 row vectors of $\mathbf{U}_k$ weighted by the singular values in $\sum_k$

Latent semantic indexing

The LSI approach makes 3 claims:

- The semantic information can be derived from a word-document co-occurrence matrix => word context = document (although considering smaller contexts is possible)
- Dimensionality reduction is an essential part of this derivation
- The words and documents can be represented as points in the Euclidean space

How to define k = number of dimensions ?

3. Neural network based representations

Neural network based representations

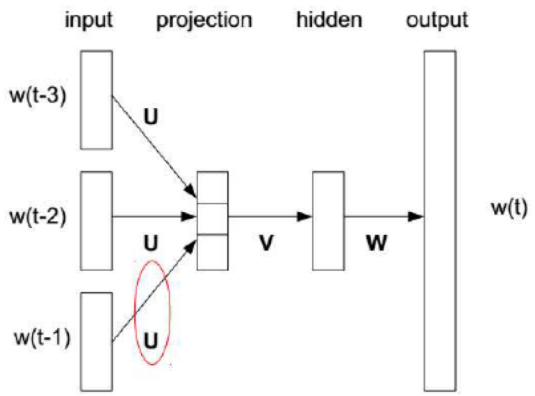
Motivation:

- The main motivation is to come up with more precise way how to represent and model words, documents and other content
- Inspired by the success in computer vision

Word vectors

Neural net based word vectors were traditionally trained as part of neural network language models [Bengio et al. 2003]

This models consists of input layer, projection layer, hidden layer and output layer



Tomas Mikolov, COLING 2014

Word vectors = word embeddings

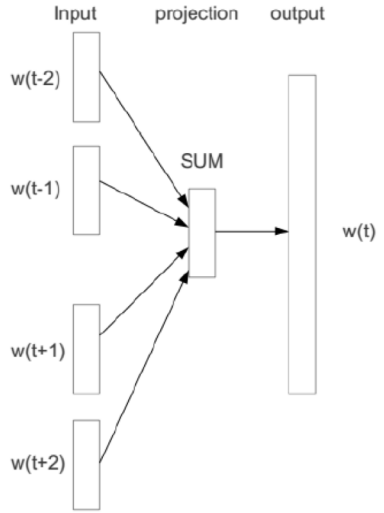
Word embedding = distributed representation of the word in the form of a vector: **trained with its local contexts**

- Each word is associated with a real valued vector in k-dimensional space (usually $k = 50 – 1000$)
- Usually computed with a neural network
- Capture linguistic regularities (e.g., syntax) in the word vector space
- Various architectures to train the vectors: most famous:
  - CBOW
  - Skip-gram NN-LM

Word vectors = word embeddings

CBOW model:

- Input: words within short window **without their position**
- Output: the current word is predicted
- The hidden layer implements a linear function: efficient computation

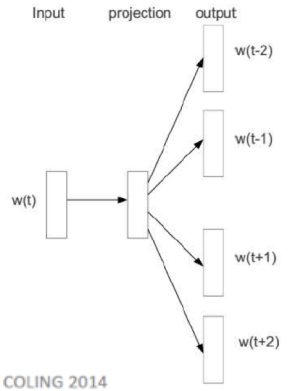


Tomas Mikolov, COLING 2014

Word vectors = word embeddings

Skip-gram NNLM model:

- Input: the current word
- Output: the words within a short window is predicted, the network predicts the surrounding words, **not their position**
- The hidden layer implements a linear function: efficient computation



Tomas Mikolov, COLING 2014

Skip-gram NNLM

- Input: one-hot coding: i.e., number of inputs = size of the vocabulary: every neuron represents a word
- Input is fed to the hidden layer without activation function: projection layer
  - Hidden layer typically contains around 300 neurons:

$$\mathbf{o}_{hidden} = \mathbf{V}_{hidden}\ \mathbf{o}_{in}$$

V = interconnection matrix containing the weights

- Output layer has again the size of the vocabulary
- Using softmax activation function on the output layer:

$$\mathbf{o}_{out} = softmax - vector(\mathbf{V}_{out}\mathbf{o}_{hidden}) = softmax - vector(\mathbf{V}_{out}\mathbf{V}_{hidden}\mathbf{o}_{in})$$

Skip-gram NNLM

The training of this model is computationally expensive because of the normalization of the softmax function (denominator sums over the whole vocabulary)

=> two efficient approximations have been proposed:

- **Negative sampling**
- Hierarchical softmax

Skip-gram NNLM

Negative sampling:

- Instead of propagating the signal from the hidden layer to the whole output layer, only the output neuron that represents the positive class + few randomly sampled negatives are evaluated
- For each positive training example (word w and its surroundings), k negative choice examples are generated (w with random surrounding words)
- Training: maximizing output in case of positive example and minimize in case of noisy input

Word vectors: training

Subsampling = to not give importance to frequent words (e.g., stopwords): words are randomly discarded with a probability depending on their frequency of occurrence:

$$P_{discard}(w_i) = 1 - \left(\sqrt{\frac{trh}{freq(w_i)}}\right)$$

$$P_{discard}(w_i) = 1 - \left(\sqrt{\frac{trh}{freq(w_i)}} + \frac{trh}{freq(w_i)}\right)$$

Used in word2vec tool

With trh = threshold frequency and $freq(w_i)$ the frequency of w_i in the training collection

Word vectors: training

- What are the word vectors?
  - Commonly: **columns of the $\mathbf{V}_{hidden}$ matrix**: word vector has reduced dimension
  - Alternative: columns of the $\mathbf{V}_{out}$ matrix (vector has size of the vocabulary)
  - Trained model can also be used as language/prediction model

- How to choose *cs* (= window context size left and right of the word)?
- Non-linearity does not seem to improve performance of the above word embedding models, thus the hidden layer does not use a non-linear activation function in word2vec

Alternatives

Alternative ways to compute embeddings or language models (but computational more complex):

- Neural network based:
  - Neural Net Language Models (NN-LMs) [Bengio et al. 2006]
  - Convolutional Nets for tagging (SENNA) [Collobert & Weston 2008]
  - Recurrent NN-LMs [Mikolov et al. 2010]
  - Recursive NNs [Socher et al. 2011]
  - Paragraph Vector [Le & Mikolov 2014]
- Bayesian network based:
  - Latent Words Language Model [Deschacht & Moens 2009, Deschacht et al. 2012]

Recent alternative language models

Neural models pretrained on a language modeling task:

- ELMo [Peters et al. 2017]
- OpenAI GPT [Radford et al. 2018]
- BERT [Devlin et al., 2018]

- Have achieved impressive results on various natural language processing tasks such as question-answering and natural language inference

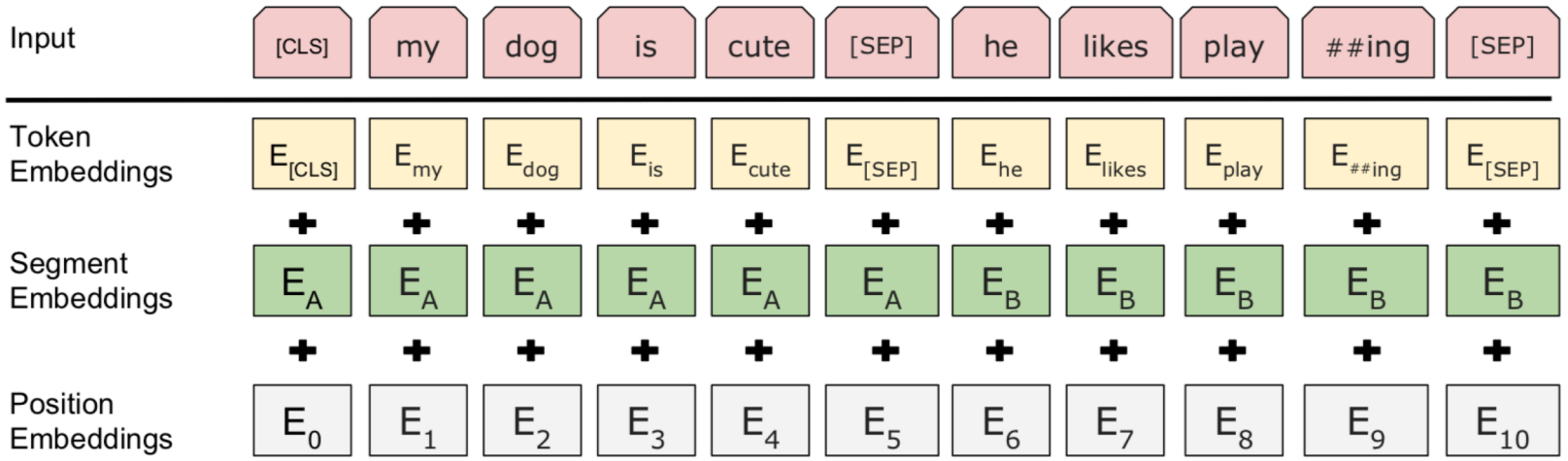


Figure 2: BERT input representation. The input embeddings is the sum of the token embeddings, the segmentation embeddings and the position embeddings.

[Devlin et al., 2018]

Word vectors: comparison

- Comparison of models in terms of training time and accuracy in word similarity tasks: note that almost all models are trained on different datasets
- Skip-gram NNLM and CBOW are trained on the Google 20K questions dataset (8869 semantic and 10675 syntactic questions)

Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

[Mikolov et al . 2013b]

Tomas Mikolov, COLING 2014

Word vectors

- The choice of training corpus is usually more important than the choice of the technique itself
- The crucial component of any successful model thus should be low computational complexity
- Optimized code for computing the CBOW and skip-gram models has been published as **word2vec project**: https://code.google.com/p/word2vec/

Word vectors

Used in many tasks such as word similarity, categorization, word similarity, etc.

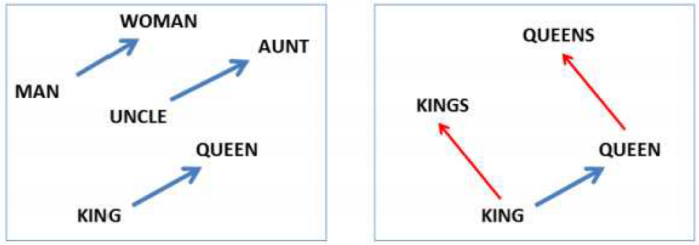
Word similarity

Because a word w_i is represented as a vector $\mathbf{v}_i$ capturing its context: similarity and distance between two words can be computer with classic similarity or distance metrics respectively: e.g.,

$$cos(\mathbf{v}_i, \mathbf{v}_j) = \frac{\mathbf{v}_i \cdot \mathbf{v}_j}{\|\mathbf{v}_i\|\|\mathbf{v}_j\|}$$

Word similarity

- It was shown that word vectors capture many syntactic and semantic properties such as gender, tense, plurarity, lexical semantics, etc.
- E.g., shown by simple vector operations: king – man + woman ≈ queen



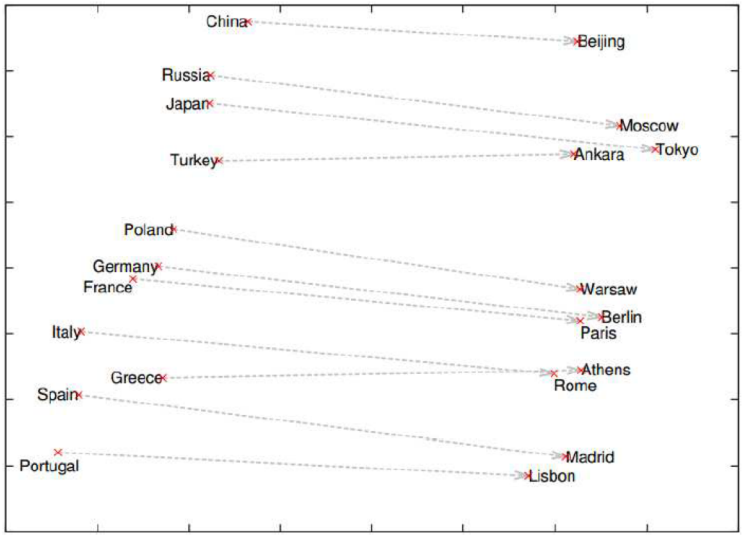
Tomas Mikolov, COLING 2014

Word similarity

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Tomas Mikolov, COLING 2014

Word similarity



Tomas Mikolov, COLING 2014

Visualization using PCA

Open issues

How many dimensions should we allocate for each word?

- **There are no theoretical bounds or even established best-practices** [Goldberg & Levy 2015]
- In current research, the dimensionality of word-embedding vectors range between about 50 to a few hundreds, and, in some extreme cases, thousands
- One experiments with a few different sizes, and chooses a good trade-off between speed and task accuracy
- But, need for theoretical foundations

Open issues

Which architecture and corresponding parameters?

- Should the vector for "dog as previous-word" be the same as the vector of "dog as next-word"? Or should we assign them two distinct vectors?
- **Integrating sequence information** = especially important for fine-grained text representations (e.g., used in information extraction)
  - Deep learning solutions proposed as recurrent neural network (RNN) and long short-term memory (LSTM) variants
  - Sequence embeddings
- Which input information?
- But architecture and parameters tuned empirically, again **need for theoretical foundations**

Open issues

How to learn **embeddings of larger units, e.g., for text: phrases and sentences, documents**?

- Idea to assign unique tokens to phrase or sentence, but not feasible given the variety of sentences
- Train on different variants of a sentence that express the same meaning (e.g., COCO MS dataset), but limited
- Other forms of compositionality apart from simple additions, subtractions and component-wise multiplication of vectors components, but again **need for theoretical foundations**
- Empirical evidence: best unsupervised models: average of the (e.g., tf-idf weighted) word embeddings

4. Use of word vectors in retrieval models

Word vectors in IR

New area of research, but closely aligned with previous research on vector space models (cf. tutorial at WSDM 2017)

- Open issues:
  - **How to represent document and queries based on word vectors ?** [Le & Mikolov ICML 2014]
    - E.g., experiments with training of document vectors and paragraph vectors (cf. LSA with word embeddings), but not yet a successful and principled way
  - **How to build similarity or distance metrics that operate on sets of word vectors ?**
    - E.g., word mover's distance, but computationally complex and not always accurate

Retrieval models that make use of word vectors

Vector (space) model (VSM): document and query are represented as term vectors with term weights ≥ 0 in a t-dimensional space:

$\mathbf{d}_i = [w_{1i}\ w_{2i} \ldots w_{ti}]$

$\mathbf{q} = [w_{1q}\ w_{2q} \ldots w_{tq}]$

where t = the number of features (here terms) measured

Now for each w_i we have learned a corresponding word vector $\mathbf{v}_i$

How to represent $\mathbf{d}_i$ and $\mathbf{q}$ based on the word vectors?

Retrieval models that make use of word vectors

Centroid model used in vector (space) model: document and query are represented as term vectors with term weights ≥ 0 in a *t*-dimensional space:

$\mathbf{d'}_i = (\mathbf{v}_1 + \mathbf{v}_2 \ \ldots \ + \mathbf{v}_p)/\ p$

$\mathbf{q'} = (\mathbf{v}_1 + \mathbf{v}_2 \ \ldots \ + \mathbf{v}_q)/\ q$

where p = number of words in $\mathbf{d}_i$

where q = number of words in $\mathbf{q}$

Possible combine with traditional bag-of-words model: e.g.,

$$(1-\alpha)\, cos(\mathbf{d}_j, \mathbf{q}) + \alpha\, cos(\mathbf{d}'_j, \mathbf{q}')$$

Word mover's distance

Figure 1. An illustration of the *word mover's distance*. All non-stop words (***bold***) of both documents are embedded into a *word2vec* space. The distance between the two documents is the minimum cumulative distance that all words in document 1 need to travel to exactly match document 2. (Best viewed in color.)

[Kusner et al. ICML 2015]

Word mover's distance

WMD = distance between two text documents d_x and d_y that takes into account the alignments between words

Let $|x|$ and $|y|$ be the number of distinct words in d_x and d_y

Let $\mathbf{f}_x \in \mathbb{R}^{|x|}, \mathbf{f}_y \in \mathbb{R}^{|y|}$ denote the normalized frequency vectors of each word in the documents d_x and d_y respectively so that : $\mathbf{f}_x^T\mathbf{I} = \mathbf{f}_y^T\mathbf{I} = 1$

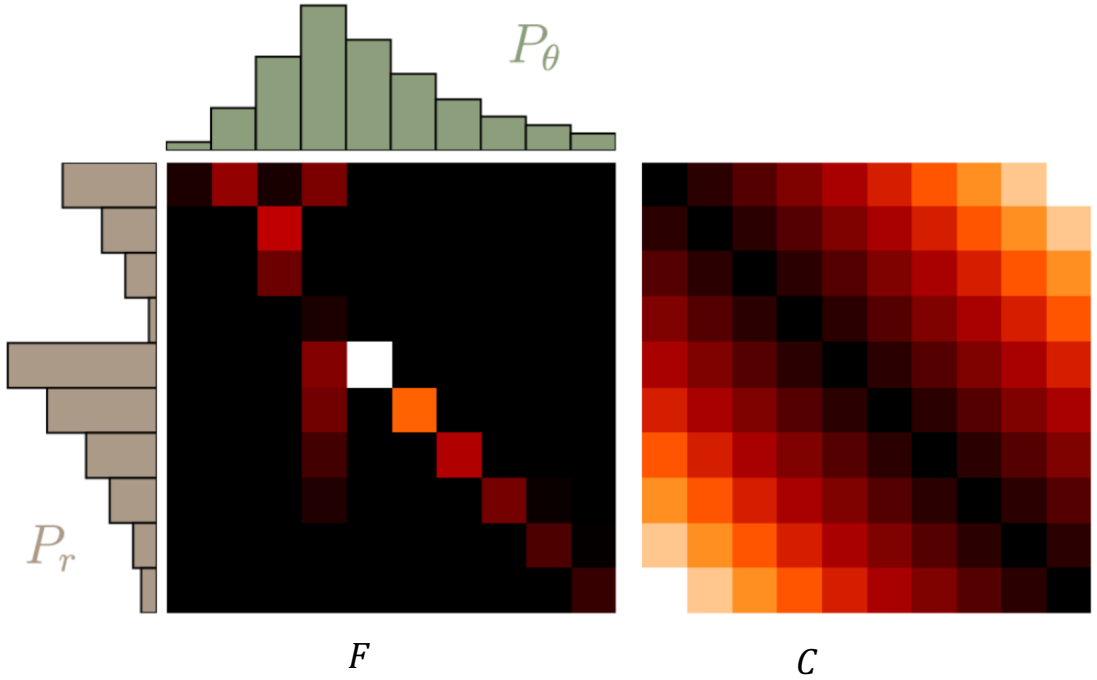
Word mover's distance

WMD: Moves words from one document d_1 to its nearest neighboring word of another document d_2.
Figure adapted from [Niculae 2015]

Word mover's distance

WMD words flow in the direction of the arrows when one document has more context than the other. The ticker arrows indicate the flow the closest word, when the thin arrows indicate excess flow. Figure adapted from [Kusner et al. 2015]

Word mover's distance



Adapted from: https://vincentherrmann.github.io/blog/wasserstein/

Word mover's distance

Then the WMD distance between documents d_x and d_y is defined as:

$$WMD(d_x, d_y) = \min_{F \in \mathbb{R}_+^{|x| \times |y|}} \langle C, F \rangle$$

subject to, $F\ \mathbf{I} = \mathbf{f}_x$, $F^{\mathrm{T}}\ \mathbf{I} = \mathbf{f}_y$

where

$F =$ transportation flow matrix with F_{ij} denoting the amount of flow traveling from the i-th word x_i in d_x to the j-th word y_j in d_y

$C =$ transportation cost matrix with $C_{ij} = dist(\mathbf{v}_{x_i}, \mathbf{v}_{y_j}) =$ distance between the two word embedding vectors $\mathbf{v}_{x_i}$ and $\mathbf{v}_{y_j}$ (e.g., Euclidean distance)

Retrieval models that make use of word vectors

- **Neural translation language model:**

$$P(w_i|d_j) = \sum_{u \in d_j} P(w_i|u)P(u|d_j)$$

where $P(w_i|u)$ is e.g., computed based on corresponding word vectors $\mathbf{w}_i$ and $\mathbf{u}$

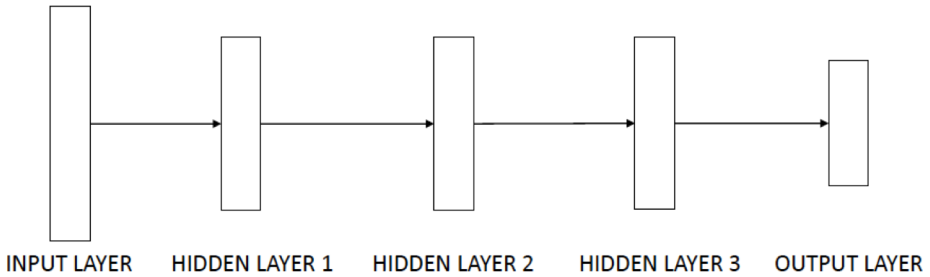
$$P(w_i|u) = \frac{cos(\mathbf{w}_i, \mathbf{u})}{\sum_{\mathbf{u}' \in \boldsymbol{Voc}} cos(\mathbf{w}_i, \mathbf{u}')}$$

considering all query word – document word pairs, where $\boldsymbol{Voc}$ is the set of all vocabulary words

[Zuccon et al. ADCS 2015]

Deep learning in IR

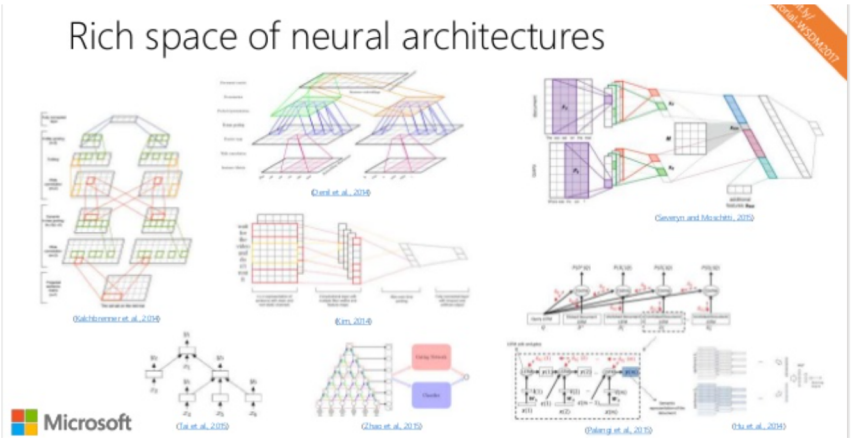
Whenever we try to learn complex function that is a composition of simpler functions, it may be beneficial to use deep architecture (often done in computer vision and natural language processing)



Deep learning in IR

Useful when we have annotated training data:

- Learning to rank
- Text categorization
- Information extraction
- …



[Mitra & Craswell WSDM 2017]

Learning to rank

Up till now unsupervised retrieval models and content representations: usually used as first filter

Learning to rank for IR uses **supervised training data** such as human relevance labels and click data to train towards an IR objective:

=> direct supervision is used to optimize the model for a ranking task: usually used as a second step in the ranking

- A classifier is trained:
  - Pointwise approach: a binary or graded relevance label is predicted for each document
  - Pairwise approach: the preference ranking of two documents is predicted
  - Listwise approach: a ranked relevance list of documents is predicted

What have we learned?

- Document and word representations based on Latent semantic indexing
- Understanding the basics of word representations obtained with neural networks
- Integration of the obtained vector representations in vector space and language retrieval models
- Learning good distributed representations of text is important for dealing with a vocabulary mismatch between query and document, but exact matching is also important to deal with rare words and specific information needs

Main references

Deerwester, S., Dumais, S., Furnas, G., Landauer, T. & Harshman, R. (1990) Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6), 391—407.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation*. Prentice Hall.

Goldberg, Y. (2015). *A Primer on Neural Network Models for Natural Language Processing*. Technical report: http://www.cs.biu.ac.il/~yogo/nnlp.pdf.

Kusner, M.,J., Sun, Y., Kolkin, N.I. & Weinberger, K.Q. (2015). From word embeddings to document distances. In *Proceedings ICML* 2015.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013a). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. & Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. In *Proceedings NIPS*.

Mitra, B. & Craswell, N. (2017). *An Introduction to Neural Information Retrieval* (Foundations and Trends in Information Retrieval).