8. Latent & Semantic Retrieval

Prof. Dr. Goran Glavaš

Center for AI and Data Science (CAIDAS)
Fakultät für Mathematik und Informatik
Universität Würzburg

After this lecture, you'll...

- Know about retrieval models that go beyond term matching
- Understand different models for capturing semantics of texts
- Know what Latent Semantic Analysis/Indexing is
- Understand how to use Topic Modeling in IR
- Know what word embeddings are and how to exploit them in IR

Outline

- Recap of Lecture #7
- Beyond term matching
- Latent Semantic Analysis/Indexing
- Probabilistic Topic Modeling for IR
- Word Embeddings for IR

Recap of the previous lecture

- Improving recall of IR systems
  - **Q:** When does recall matter more then precision in IR?
  - **Q:** Which are global and which are local methods for improving recall?
- Relevance feedback
  - **Q:** What is relevance feedback?
  - **Q:** How do we incorporate relevance feedback into probabilistic retrieval?
  - **Q:** How does Rocchio algorithm work?
  - **Q:** What is pseudo-relevance feedback? How does Relevance model for pseudo-relevance feedback work? Compare Rocchio algorithm and Relevance model.
- Query expansion
  - **Q:** Name and explain different query expansion methods?
  - **Q:** How does thesaurus-based query expansion work?
  - **Q:** How may we automatically build a thesaurus?

Rocchio algorithm

- We are given only a handful of relevance feedback annotations
- Thus, we re-estimate the query by combining
  1. Centroid of relevant documents
  2. Centroid of non-relevant documents
  3. Initial query vector q_0

$$q_m = \alpha \cdot q_0 + \left(\beta \cdot \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j \right) - \left(\gamma \cdot \frac{1}{|D_{nr}|} \sum_{d_j \in D_{nr}} d_j \right)$$

- D_r is the set of vectors of known relevant documents (different from C_r)
- D_{nr} is the set of vectors of known non-relevant documents (different from C_{nr})
- α, β, and γ are weights, determining the contribution of each component (set beforehand or empirically)
- New query moves towards the relevant and away from non-relevant documents

Relevance model (Lavrenko, 2001)

- Input
  - Initial query q_0
  - Top K documents in the ranking for initial query – d_1, d_2, ..., d_K
  - Relevance probabilities of top ranked documents for the initial query – $P(d_i|q_0)$
- Output
  - A distribution of terms denoting how well they describe the initial query q_0
  - An importance/probability of term w for q_0 query is computed as follows:

$$P(w|q_0) = \sum_{i=1}^{K} P(w|d_i) \cdot P(d_i|q_0)$$

  - Rank the terms in decreasing order of $P(w|q_0)$, take top N terms and combine them into a weighted expansion query q_{PRF}

Relevance model vs. Rocchio algorithm

- Let's compare Lavrenko's relevance model with Rocchio algorithm
  - Assume Rocchio considers top K initially ranked documents as relevant (D_r) and does not consider non-relevant documents ($\gamma = 0$)
- Lavrenko's relevance model

$$q' = \lambda \cdot q_0 + (1 - \lambda) \cdot q_{RF}$$

$$q' = \lambda \cdot q_0 + (1 - \lambda) \cdot q_{RF}$$

- Rocchio algorithm

$$q_m = \alpha \cdot q_0 + \beta \cdot \frac{1}{|D_r|} \sum_{d_j \in D_r} d_j$$

Rocchio uses all terms, **RM uses only top N terms**

Rocchio computes simple average, **RM weighted average with document relevances for query $P(d_i|q_0)$ as weight**

Rocchio uses TF-IDF weights, **RM uses $P(w|d_i)$**

Thesaurus-based query expansion

- Manually producing a thesaurus is time-consuming and expensive
  - Additionally, it needs to be constantly updated to reflect changes in the domain

- **Automated thesaurus generation**
  - Generating thesaurus by detecting similarity/relatedness of terms in a large corpora
  - **Distributional hypothesis** – words are similar if they occur in similar contexts
    - E.g., „apple” is similar to „pear” as you can both harvest, peel, prepare and eat both
  - **Related words** – words that often co-appear are semantically related
    - E.g., „pilot” and „airplane”

Outline

- Recap of Lecture #7
- Beyond term matching
- Latent Semantic Analysis/Indexing
- Probabilistic Topic Modeling for IR
- Word Embeddings for IR

Beyond term matching in IR

- All IR models we considered so far were based on **term overlap** between the query and documents

- We were estimating the amount and importance of term overlap and ranked the documents according to these estimates

- Often, there is a **lexical gap** between the query and relevant documents

- E.g., query: *„bad hombre”*
  - Relevant document:
  - „These are *terrible dudes*, drug smugglers and other criminals”

Beyond term matching in IR

- We are interested in capturing semantics **beyond discrete terms**
  - „bad hombre" has similar meaning as „terrible dude"

- We must represent documents and queries **semantically**
  - So that semantically similar words and phrases have similar representations

- Discrete bag-of-words representations **do not meet** this requirement
  - With discrete terms – all words are equally similar/distant
    - d(„dog", „cat") = d(„dog", „space")

  - Vectors of texts with no lexical overlap will be dissimilar
  - *cos(bow(„bad hombre"), bow(„terrible dude"))* = **0**

Beyond term matching in IR

- Latent and semantic IR models all represent texts with **semantic vectors**
  - Able to **bridge** the **lexical gap** between query and documents
  - Models have different theoretical underpinnings but they all produce **numeric vectors** to represent the **meaning** of portions of text
    - Words, phrases, sentences, paragraphs, documents

- Semantic representations of text typically derived from large corpus, exploiting the **distributional hypothesis**:
  - **„You shall know the meaning of the word by the company it keeps”** (Harris, 1954)
  - E.g., „dog” and „cat” will tend to co-occur with the similar sets of words (e.g., „eat”, „pet”, „cuddle”, „friend”).

Beyond term matching in IR

- Latent and semantic models used in IR that we will cover

1. Latent Semantic Analysis (LSA)
   - Often called Latent Semantic Indexing (LSI) when used for IR
   - Decomposition of word-document co-occurrence matrix

2. Probabilistic Topic Modeling for IR
   - Generative model assuming that documents and words are probabilistic distributions over a set of latent topics

3. Text Embeddings
   - Also based on distributional hypothesis, but do not count co-occurrences
   - Start from random vectors and update them based on observations in large corpora

Beyond term matching in IR

- Latent vs. Term-based IR models
  - Use latent/semantic models when
    1. Query terms do not need to be exactly matched
    2. Recall is as important as precision
    3. There are many relevant documents with lexical gap wrt. to query
  - Use term-based IR models
    1. Query terms need to be exactly matched
    2. Recall (i.e., retrieving all relevant documents) is not so important
    3. There are many relevant documents, most of which are expected to have significant lexical overlap with the query

Outline

- Recap of Lecture #7
- Beyond term matching
- Latent Semantic Analysis/Indexing
- Topic Modeling for IR
- Word Embeddings for IR

Latent Semantic Indexing

- Assume we have a collection of N documents and a vocabulary of M words

- We start by building a word-document occurrence matrix **A** od dimensions **M x N**
  - Rows correspond to words
  - Columns correspond to documents
  - Elements **A**[*i*, *j*] contain information about the occurrence of word *i* in document *j*
    - Can be binary indicators of occurrence, raw frequency, or TF-IDF weights

- Rows of the occurrence matrix A are distributional vectors of words
  - These vectors are of a large dimension N (we assume large document collections)
  - Distributional vectors of words are **sparse** – on average the word appears only in a small subset of all documents in the collection
- Columns of **A** are also sparse vectors (of size M) representing documents

Latent Semantic Indexing

- Toy example:
  - Collection of 6 documents, d1–d3 about *politics* and d4 – d6 about *sports*
  - Three groups of words corresponding to prominent topics: *politics*, *sport*, and *other*
  - Ocurrence matrix contains raw occurrence frequency

$$A = \begin{array}{l} \\ \textit{president} \\ \textit{minister} \\ \textit{speech} \\ \textit{law} \\ \textit{ball} \\ \textit{score} \\ \textit{player} \\ \textit{run} \\ \textit{person} \\ \textit{piano} \\ \textit{mouse} \end{array} \begin{array}{c} \begin{array}{cccccc} d1 & d2 & d3 & d4 & d5 & d6 \end{array} \\ \begin{pmatrix} 3 & 2 & 0 & 1 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 & 0 \\ 2 & 5 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 1 \\ 0 & 0 & 0 & 4 & 0 & 2 \\ 0 & 0 & 0 & 3 & 2 & 3 \\ 0 & 0 & 1 & 1 & 4 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Latent Semantic Indexing

- **Latent Semantic Indexing (LSI) –** IR model based on matrix factorization, namely Singular Value Decomposition (SVD) of the word-document occurrence matrix

- We decompose the sparse word-document occurrence into factor matrices which we use to obtain dense vector representations of words and documents

- Obtained dense vectors better capture meaning of words and documents
  - Comparing dense vectors of words better captures their semantic similarity than comparing their sparse distributional vectors
  - Comparing dense vectors of documents captures semantic similarity between documents beyond term overlap

LSI – Singular Value Decomposition

- Given a matrix **A** (with non-negative elements), the Singular Value Decomposition finds orthogonal matrices **U** and **V** and a rectangular diagonal matrix **Σ** such that:

$$A = U\Sigma V^T$$

- Matrix **U** is of dimensions **M** x **M**
- Matrix **V** is of dimensions **N** x **N**
- Matrix **Σ** is of dimensions **M** x **N**
- U and V are orthogonal: $\mathbf{U^T U = I}$, $\mathbf{V^T V = I}$
- Values of the diagonal matrix **Σ** are singular values of the original matrix **A**
- Let *r* be the rank of matrix **A**

LSI – Singular Value Decomposition

- We apply SVD to the word-document occurrence matrix **A**

$$\underset{\mathbf{A}}{\begin{bmatrix} \begin{bmatrix} \\ d_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ d_N \\ \\ \end{bmatrix} \end{bmatrix}} = \underset{\mathbf{U}}{\begin{bmatrix} \begin{bmatrix} \\ u_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ u_r \\ \\ \end{bmatrix} \end{bmatrix}} \cdot \underset{\mathbf{\Sigma}}{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix}} \cdot \underset{\mathbf{V}^T}{\begin{bmatrix} \begin{bmatrix} \\ \hat{d}_1 \\ \\ \end{bmatrix} \dots \begin{bmatrix} \\ \hat{d}_N \\ \\ \end{bmatrix} \end{bmatrix}}$$

- Each document d_i can be written as a linear combination (i.e., weighted sum) of elements of column vectors u_1, ..., u_r (*r* is the rank of **A**)
- Typically, $\sigma_1 > \sigma_2 > \dots > \sigma_r$ – thus the first components of columns vectors in V^T have more influence than the later ones

LSI – SVD Example

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

- The first column („topic”) seems to have weights of large magnitude for *politics* terms, and the second column for *sports* terms

LSI – SVD Example

- Useful to look at columns of the matrix ΣV^T to see scaled topic weights for each document

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

- As expected, the first three documents have large-magnitude weights for the „politics" topic, and the second other three for the „sports" topics

Latent Semantic Indexing

- **Goal**: reduce the dimensionality of word and document vectors and obtain dense semantic vectors of terms and documents

- We reduce the size of the matrix $\mathbf{\Sigma}$ with singular values
  - We keep only the top K largest singular values: $\sigma_1, \ldots, \sigma_k$
  - We denote the reduced matrix with $\mathbf{\Sigma}_k$
  - Dense vectors for terms and documents will be then be of dimension K

- By reducing the rank of the matrix with singular values, we are effectively retaining only the K most prominent „topics"
  - Retained topics carry the most of the „meaning"
  - The topics/dimensions we discard are assumed to be noise

LSI reduction – example

- This leaves us with the best possible approximation of rank A_K ($K = 2$ in our example) of the original term-document occurrence matrix A

Dense vectors of terms

U_K

Dense vectors of documents

$$
\begin{array}{l}
president \\ minister \\ speech \\ law \\ ball \\ score \\ player \\ run \\ person \\ piano \\ mouse
\end{array}
\begin{bmatrix}
-0.43 & 0.13 \\
-0.53 & 0.25 \\
-0.58 & 0.33 \\
-0.12 & -0.05 \\
-0.22 & -0.51 \\
-0.26 & -0.62 \\
-0.22 & -0.40 \\
-0.03 & -0.06 \\
-0.11 & -0.03 \\
-0.10 & -0.02 \\
-0.09 & -0.08
\end{bmatrix}
\cdot
\begin{array}{c}
\begin{array}{cccccc} d_1 & d_2 & d_3 & d_4 & d_5 & d_6 \end{array} \\
\begin{bmatrix}
-4.66 & -4.37 & -2.71 & -2.37 & -1.51 & -1.65 \\
2.01 & 2.12 & 0.49 & -4.23 & -2.93 & -3.35
\end{bmatrix}
\end{array}
\Sigma_K V^T_K
$$

- A_K has the same dimensions as original A ($M \times N$)
- U_K is of size $M \times K$, and $\Sigma_K V^T_K$ of size $K \times N$

Latent Semantic Indexing

- In practice, we don't compute A_K
  - A_K is not a sparse matrix – it's explicit computation is computationally expensive!
  - We don't need to have A_K to compare pairs of terms or pairs of documents

- Term comparison is performed by comparing rows of U_K
  - sim(„president", „minister") = cos([-0.43, 0.13], [-0.53, 0.25])
  - sim(„president", „player") = cos([-0.43, 0.13], [-0.22, -0.40])

- Document comparison is performed by comparing columns of $\Sigma_K V^T_K$
  - sim(d_1, d_2) = cos([-4.66, 2.01], [-4.37, 2.12])
  - sim(d_4, d_6) = cos([-2.37, -4.23], [-1.65, -3.35])

- **Q:** Do we need to compute complete SVD, i.e., find all singular values of **A**?

Latent Semantic Indexing

- We have shown how to obtain latent representations (i.e., dense vectors) for terms and documents in the collection using SVD

- **Q:** How do we compute the dense vector for the query?
  1. Compute the sparse vector **q** of the query (e.g., TF-IDF vector)
  2. Project the sparse vector **q** into the dense topic space of documents **q’** (i.e., $\Sigma_K q_K$)

$$\mathbf{q'} = \mathbf{U}_K^T \mathbf{q}$$

- LSI ranks the documents in decreasing order of similarity (cosine) of their dense vectors and the dense vector of the query
  - I.e., $\cos([\Sigma_K V^T_K]^i, U_K^T q)$

Outline

- Recap of Lecture #7
- Beyond term matching
- Latent Semantic Analysis/Indexing
- Probabilistic Topic Modeling for IR
- Word Embeddings for IR

Topic Models for IR

- LSI has one prominent shortcoming
  - Latent topics are numerically justified – SVD ensures the best lower-dimensional approximation (i.e., with minimum loss)
  - But LSI latent topics are often not interpretable by humans – topics often contain high weights for seemingly unrelated terms
    - E.g., a topic with high weights for: *hobbit*, *umbrella*, *cinnamon*

- Alternative: induce latent topics in a probabilistic framework
  - Probabilistic LSA (pLSA)
  - Latent Dirichlet Allocation (LDA)
  - Dynamic Topic Models

Multinomial Distribution

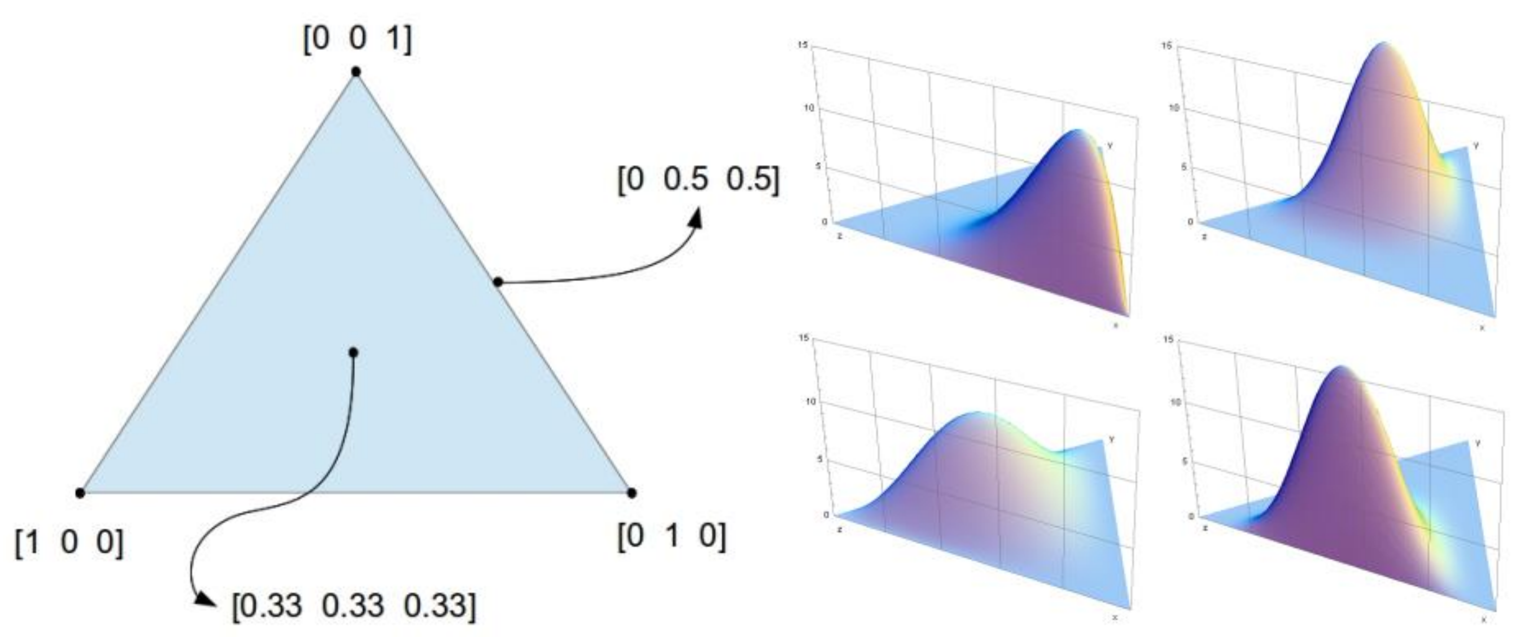
- A **multinomial (categorical) distribution** is a probability distribution over a discrete (finite) set of possible events

- We dealt with multinomial distributions when we discussed language models
  - P(w), probability of the word appearing in a language
  - E.g., P(„frodo”) = 0.1, P(„hobbit”) = 0.2, P(„house”) = 0.4, P(„see”) = 0.3

- The multinomial distribution over N terms, which we denote with $Mult_K(\boldsymbol{\vartheta})$ is parametrized by the vector $\boldsymbol{\vartheta}$ of N – 1 probabilities
  - Probabilities of the distribution must sum to 1, so we can compute the last probability from the given N – 1

Dirichlet Distribution

- **Dirichlet distribution** is a probability distribution over all vectors of length K that sum up to 1
  - A meta-distribution, a probability distribution over multinomial distributions
  - Denoted with $Dir_K(\alpha)$ Dirichlet distribution is parametrized with a parameter vector α
  - A sample $\boldsymbol{\vartheta}$ drawn from the Dirichlet distribution $Dir_K(\alpha)$ can be used to parametrize the multinomial distribution – $Mult_K(\boldsymbol{\theta})$



Latent Dirichlet Allocation

- **Latent Dirichlet Allocation (LDA)** is a latent topic model that assumes that the collection of documents was generated by a particular Dirichlet distribution
  - Collection of M documents, vocabulary of N terms, K latent topics

- Each of the K latent topics is a concrete multinomial distribution over terms
- For each position in each of the M document we obtain the observed word by:
  1. Randomly selecting one of the topics (from the Dirichlet distribution)
  2. Randomly select the term from the multinomial distribution of the topic that was randomly selected in the step 1

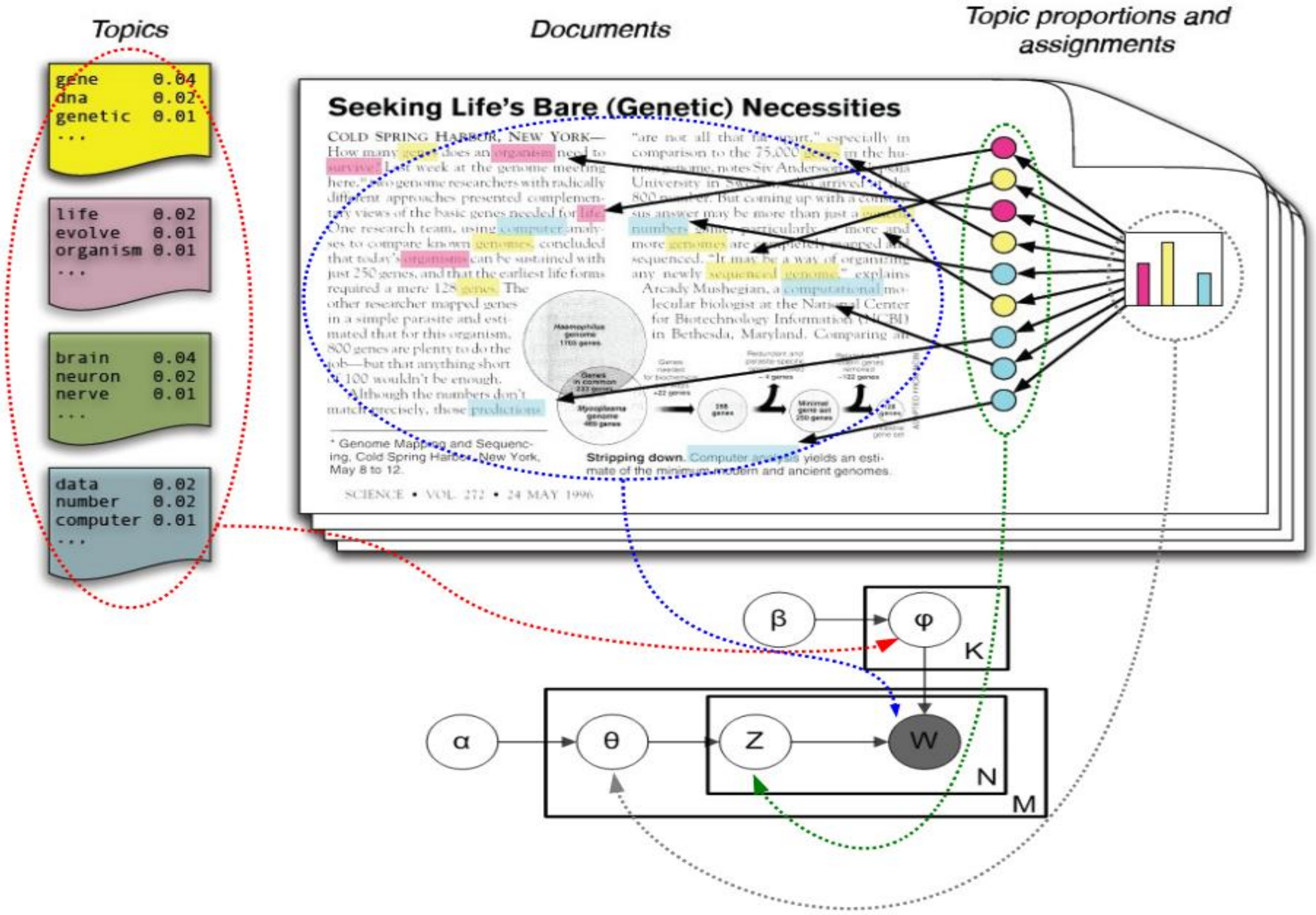
- Vocabulary of N terms
  - Each topic is a concrete multinomial distribution with N – 1 parameters

LDA – Generative View

1. For each topic k (k = 1, ..., K):
   - Draw parameters of a multinomial distribution $\boldsymbol{\varphi}_k$ (over terms) for topic k from a Dirichlet distribution $Dir_N(\beta)$

2. For each document d in the collection:
   - Draw parameters of a multinomial distribution of topics for the document d, $\boldsymbol{\theta}_d$, from a Dirichlet distribution $Dir_K(\alpha)$
   - For each term position w_{dn} in the document d:
     a) Draw a topic assignment (i.e., a concrete multinomial distribution over terms) z_{dn} from $Mult_K(\boldsymbol{\theta}_d)$
     b) Draw a concrete term w_{dn} from the multinomial distribution over terms of the topic z_{dn} (drawn in a)), $Mult_N(\boldsymbol{\varphi} z_{dn})$

LDA – Generative View



LDA – Parameters and estimation

- Parameters of the LDA are variables/probabilities that we cannot directly observe
- Probabilities of all multinomial distributions that are sampled in the generative algorithm
  1. Term probabilities (vector of N probabilities) for each of the K latent topics

     $\boldsymbol{\varphi}_k$ for k = 1, ..., K (so, total of K * N parameters)
  2. Topic probabilities (vectors of K probabilities) for each of the M documents

     $\boldsymbol{\theta}_d$ for d = 1, ..., M (so, total of M * K parameters)
- **Optimization** (learning model's parameters):
  1. Start from random multinomial distributions
  2. Update parameters to maximize probability of observed terms in documents
     - Direct maximization is intractable
     - Approximate inference (maximization) via
       - (1) variational methods or (2) sampling methods

Latent Dirichlet Allocation

- Once the model is trained (parameters optimized based on observed text), we represent documents and terms as follows:
  1. Document d – simply the multinomial distribution vector over topics for that documents, $\boldsymbol{\theta}_d$
  2. Term t_i ($i = 1, ..., N$) – for each of the K topics we take the probability of t_i from the multinomial distribution (over terms) of that topic $[\boldsymbol{\varphi}_k]^i$,. term's probability in multinomial distributions of all topics
     - **Q:** Are term vectors obtained this way probability distributions?
- Computing the representation for the query:
  - Query vector also needs to be represented as a multinomial distribution over topics
  - Easier inference:
    1. We know the probabilities of terms over topics
    2. We only need to estimate the multinomial distribution of topics given the query

Latent Dirichlet Allocation

- The topics are generally interpretable – the terms with largest probabilities within the multinomial distribution of the topic tend to be semantically related
- Example – topics obtained on 1.8M New York times articles:

music	book	art	game	show
band	life	museum	knicks	film
songs	novel	show	nets	television
rock	story	exhibition	points	movie
album	books	artist	team	series
jazz	man	artists	season	says
pop	stories	paintings	play	life
song	love	painting	games	man
singer	children	century	night	character
night	family	works	coach	know

theater	clinton	stock	restaurant	budget
play	bush	market	sauce	tax
production	campaign	percent	menu	governor
show	gore	fund	food	county
stage	political	investors	dishes	mayor
street	republican	funds	street	billion
broadway	dole	companies	dining	taxes
director	presidential	stocks	dinner	plan
musical	senator	investment	chicken	legislature
directed	house	trading	served	fiscal

Outline

- Recap of Lecture #7
- Beyond term matching
- Latent Semantic Analysis/Indexing
- Probabilistic Topic Modeling for IR
- Word Embeddings for IR

Word Embeddings

- **Word embeddings** are dense semantic vector representations of words
  - Unlike LSI, not based on counting (co)-ocurrences, but on predicting representation vectors of words based on context (surrounding words)
- Assume a vocabulary of N words
  - Sparse representation of each term is the so-called **one-hot encoding** vector that has only one non-zero element (denoting the term) and all other zeros
    - One-hot encoding vectors are highly-dimensional (size of vocabulary)
    - If we compare sparse vectors of terms, all terms are equally dissimilar (no overlap)
  - Dense representation of the term is the real-valued vector of dimension orders of magnitude lower than the size of vocabulary
    - We want real values in dense vectors of words to somehow capture meaning of words
    - LSI and LDA provide word vectors that can, to some extent, capture semantic properties of words
    - Prediction-based vectors, called **word embeddings**, have been shown to better capture the meaning of words than LSI and LDA vectors

Word Embeddings

- **Predictive models** for deriving dense word vectors try to predict
  1. The word in focus from its context or
  2. The context from the word in focus

- Popular models
  1. Skip-Gram (predicts context from the word) (Mikolov et al., '13)
  2. CBOW (predicts the word from the context) (Mikolov et al., '13)
  3. GloVe (count-based, makes global optimization) (Pennington et al., '14)

[Mikolov et al., '13] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Advances in neural information processing systems (pp. 3111-3119).

[Pennington et al., '14] Pennington, J., Socher, R., & Manning, C. D. (2014, October). Glove: Global Vectors for Word Representation. In EMNLP (Vol. 14, pp. 1532-1543).

Continuous Bag-of-Words (CBOW)

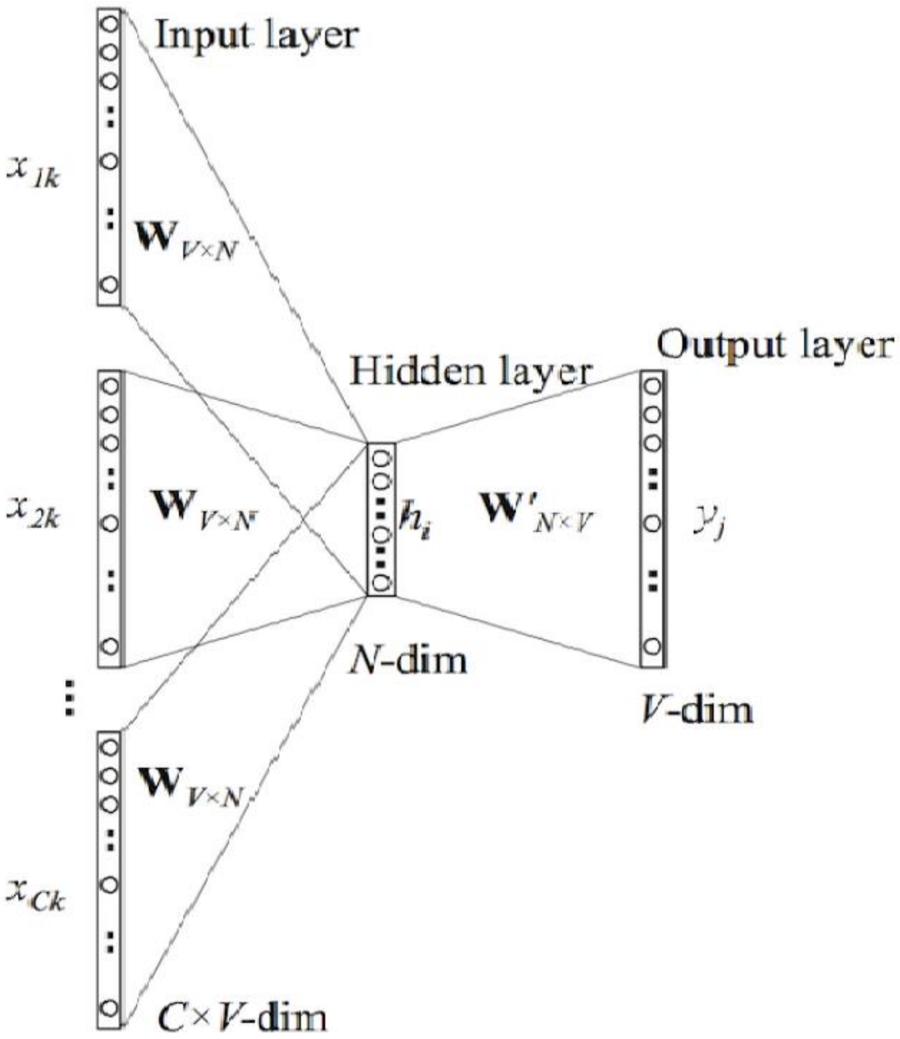
- Each word from the vocabulary of the large corpus is represented with **two dense vectors** of size N << V (size of vocabulary):
  1. **Center vector** – represents the word when it is in the focus

     E.g., „*carries*" in „*hobbit Frodo carries blue sword*"
  2. **Context vector** – represents the word when it is in the context of the center word

     E.g., „*carries*" in „*Frodo carries blue sword home*"

- Each context represented by aggregating one-hot vectors of words
- **Idea:** Given the context, predict the center word
  - E.g., given „hobbit Frodo blue sword" predict „*carries*"

Continuous Bag-of-Words (CBOW)



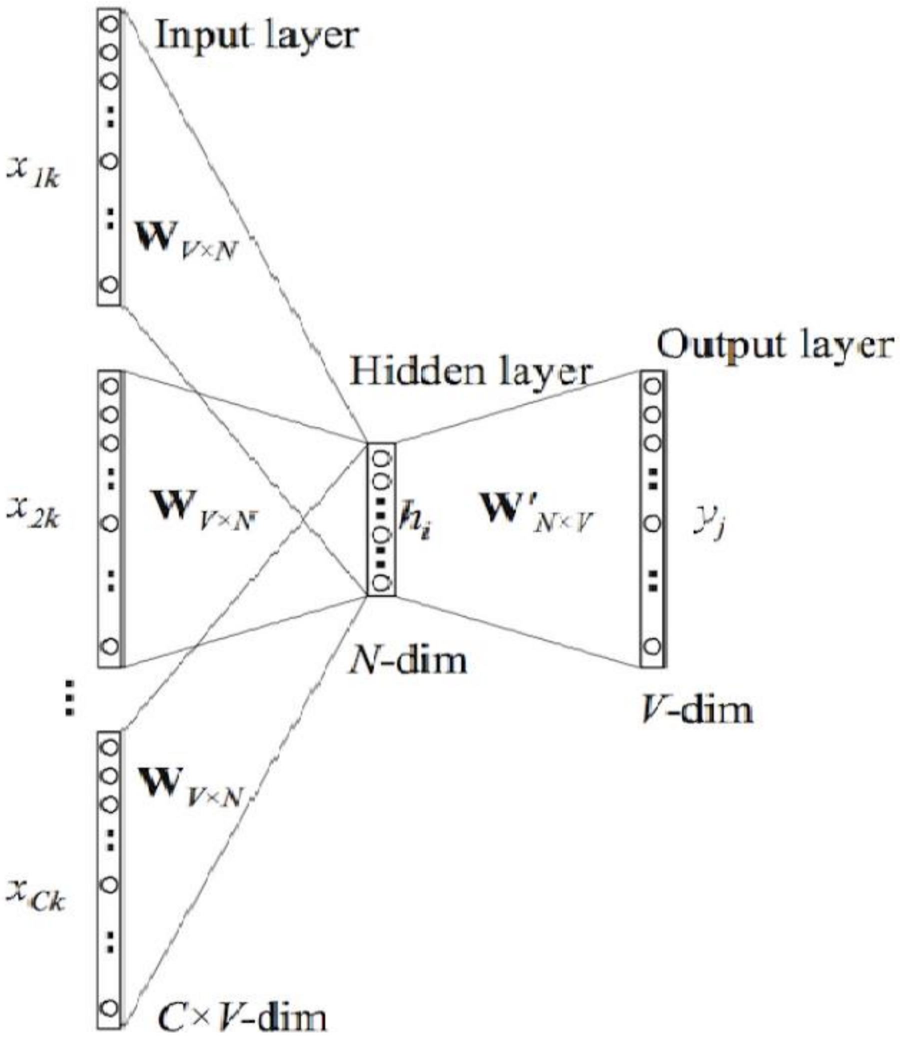
- Context consists of C words, with corresponding one-hot vectors
  - $x_{1k}, x_{2k}, \ldots, x_{Ck}$
- One-hot vectors transformed to dense vectors using input matrix **W** (V x N)
- Dense context vector h is obtained as:

$$h = \frac{1}{C}\boldsymbol{W}\left(\sum_{i=1}^{C} x_{ik}\right)$$

- Dense context vector h is then multiplied with the output matrix **W'** (N x V)

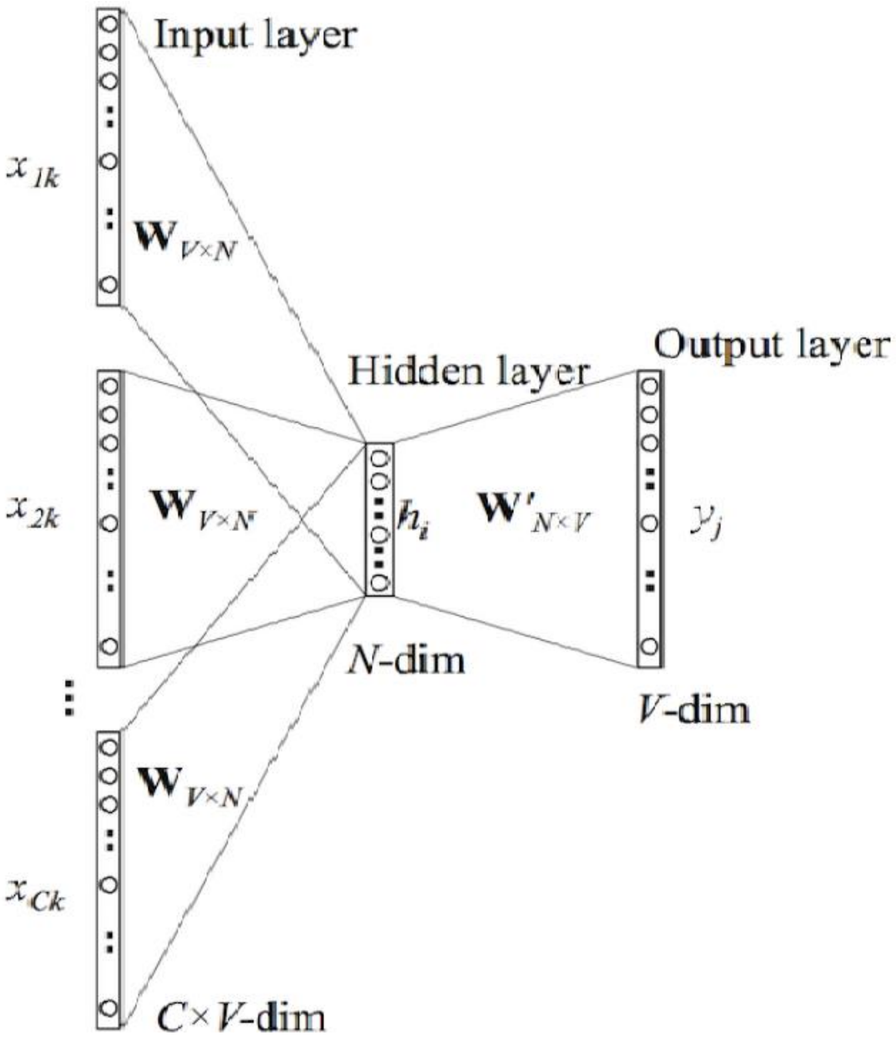
$$y_k = softmax(h^T \boldsymbol{W'})$$

Continuous Bag-of-Words (CBOW)



- Output vector y needs to be as similar as possible to one-hot vector of center word
- Parameters of the model are elements of **W** and **W'**
  - Each row of **W** is the **dense context vector** of one vocabulary word
  - Each column of **W'** is the **dense center vector** of one vocabulary word
- Dense representation (**embedding**) of the *i*-th vocabulary term is concatenation of
  1. *i*-th row of **W** and
  2. *i*-th column of **W'**

Continuous Bag-of-Words (CBOW)

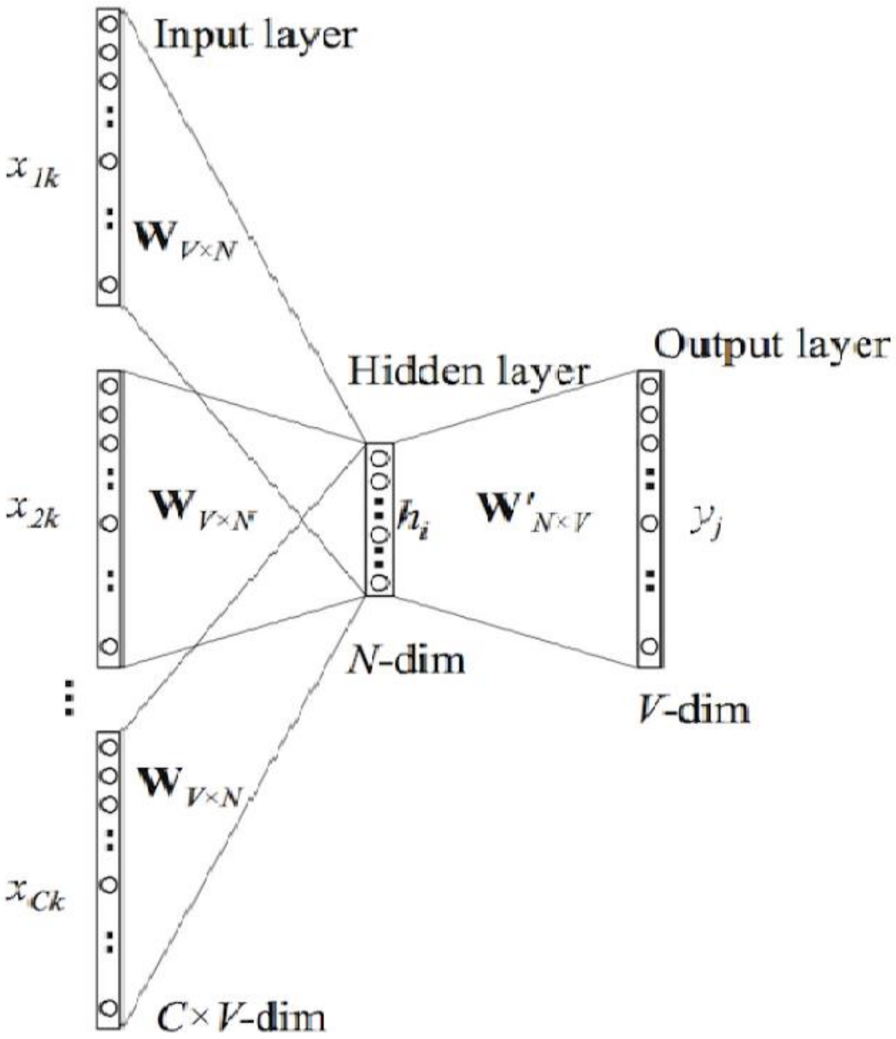


- **Q:** How do we optimize the model, i.e., learn „good" matrices **W** and **W'**?

- We prepare many examples of contexts
  1. **Positive contexts** – actual sequences of C words from a large corpus
  2. **Negative contexts** – fake artificial sequences not observed in the context
     - Obtained by replacing the center word with a random word from the vocabulary
     - Expected output vectors for negative contexts are zero vectors

- We start from random values in **W** and **W'**

Continuous Bag-of-Words (CBOW)



- For each context (i.e., „training example”), positive and negative, we compare
  1. The predicted output vector y_k
  2. One-hot vector of the center word t_k

- The difference between y_k and t_k is the prediction **error** of the model
  - Errors are propagated backwards to update **W** and **W’** using an algorithm called **backpropagation**
  - The bigger the error, the bigger the update of values in **W** and **W’**

Word embeddings

- Word embedding models like CBOW, Skip-Gram, and GloVe yield dense vectors with some **very nice semantic properties**
- They capture semantic similarity between words much better than word vectors obtained via LSI or LDA

Airplane

word	cosine
plane	0.835
airplanes	0.777
aircraft	0.764
planes	0.734
jet	0.716
airliner	0.707
jetliner	0.706

Cat

word	cosine
cats	0.810
dog	0.761
kitten	0.746
feline	0.732
puppy	0.707
pup	0.693
pet	0.689

Dog

word	cosine
dogs	0.868
puppy	0.811
pit_bull	0.780
pooch	0.763
cat	0.761
pup	0.741
canines	0.722

Word embeddings

- Word embeddings also capture **semantic analogies** between pairs of words

- e(*Germany*) – e(*Berlin*) ≈ e(„*Italy*”) – e(„*Rome*”)
- This allows for knowledge inferences like: *king* – *man* + *woman* = *queen*

Information retrieval based on word embeddings

- Word embeddings are learned on a huge external corpus of text (e.g., Wikipedia)
  - I.e., Word embeddings do not depend on our retrieval collection
- Thus, deriving word embeddings is an „offline” step we perform before retrieval

- To use word embeddings in retrieval, we need to derive dense document/query vectors from word embedding vectors
- Embeddings of a larger unit of text (phrases, sentences, paragraphs, documents):
  - Typically computed by aggregating word embeddings
  - There are also models that learn to directly predict embedding vectors of larger text units (Kiros et al., ’15; Reimers & Gurevych, 2019)

[Kiros et al., ‘15] Kiros, R., Zhu, Y., Salakhutdinov, R. R., Zemel, R., Urtasun, R., Torralba, A., & Fidler, S. (2015). Skip-thought vectors. In *Advances in neural information processing systems* (pp. 3294-3302).

[Reimers, N., & Gurevych ‘19] Reimers, N., & Gurevych, I. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP-IJCNLP) (pp. 3982-3992).

Information retrieval based on word embeddings

- Let document d contain terms $t_1, \ldots, t_N$ and let $e(t)$ be the word embedding of the term t
- The aggregate embedding vector of the document d, to be used for retrieval, is computed as weighted average of word embeddings:

$$e(d) = \frac{\sum_{i=1}^{N} w_i * e(t_i)}{\sum_{i=1}^{N} w_i}$$

- Weight w_i determines how much the word embedding of term t_i contributes to the aggregate embeddings
  - As usual, we would want more frequent/common words to contribute less
  - Thus, TF-IDF scores are often used as weights, i.e., $w_i = tf(t_i, d) * idf(t_i)$

Now you...

- Know about retrieval models that go beyond term matching
- Understand different models for capturing semantics of texts
- Know what Latent Semantic Analysis/Indexing is
- Understand how to use Topic Modeling in IR
- Know what word embeddings are and how to exploit them in IR