Vector Space Models of Semantics

Word and Sentence Embeddings and Topic Models

Língua Natural e Sistemas Conversacionais

Luiz Faria

Adapted from Natural Language Processing course from HSE University

Distributional Semantics

Distributional Semantics

- How to represent the meaning of words or documents?
- We can represent this meaning by vectors, in such a way that similar words will have similar vectors, and similar documents will have similar vectors
- Some applications:
  - To search in documents
  - To get a representation of the hierarchical structure of some area, from concepts and examples of these concepts
  - To find the most similar documents
  - ...

Word Similarities

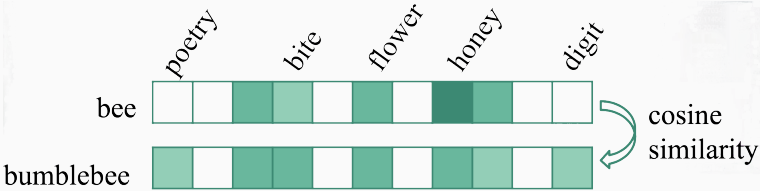
- First order co-occurrences

  syntagmatic associates / relatedness (bee and honey)

- Second order co-occurrences

  paradigmatic parallels / similarity (bee and bumblebee)

Syntagmatic associates: the terms often co-occur together in some contexts
Paradigmatic parallells: the term would co-occur with similar words in their contexts



Schutze, H., & Pedersen, J. (1993). A vector model for syntagmatic and paradigmatic relatedness. In Making Sense of Words: Proceedings of the Conference, pp. 104-113, Oxford, England.

Distributional Hypothesis

"You shall know a word by the company it keeps."

- Firth, 1957.
  - Use a sliding window of a fixed size
  - Compute word co-occurrences n_{uv}

To find if two words are similar, we can compute word co-occurrences. But they can be biased because of too popular words in the vocabulary, like stop words. So you need to find a way to penalize too popular words.

Distributional Hypothesis

"You shall know a word by the company it keeps."

- Firth, 1957.
  - Use a sliding window of a fixed size
  - Compute word co-occurrences n_{uv}
  - Better: Pointwise Mutual Information:

$$PMI = \log \frac{p(u,v)}{p(u)p(v)} = \log \frac{n_{uv}n}{n_u n_v}$$

PMI: We use the individual counts of the words in the denominator. The goal is to verify if these two words are randomly co-occurrent or not. In the first formula, the numerator has the joint probability of the words. And in the denominator has the joint probability in the case that the two random variables are independent. If the words were independent, we get 1 for this fraction. And in case of dependent words that occur too much together, we will get something more.

Distributional Hypothesis

"You shall know a word by the company it keeps."

- Firth, 1957.
  - Use a sliding window of a fixed size
  - Compute word co-occurrences n_{uv}
  - Better: Pointwise Mutual Information:

$$PMI = \log \frac{p(u,v)}{p(u)p(v)} = \log \frac{n_{uv}n}{n_u n_v}$$

  - Even better: positive Pointwise Mutual Information:

$$pPMI = max(0, PMI)$$

With PMI, in case of words that never co-occur together, or those words that co-occur really rare, we will get negative values for *log*. So we can take the maximum of the PMI and 0 (pPMI).

Any Problems Here?

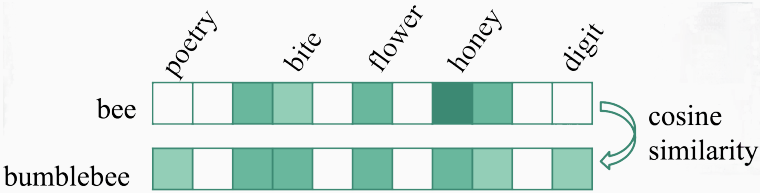
- First order co-occurrences

  syntagmatic associates / relatedness (bee and honey)

- Second order co-occurrences

  paradigmatic parallels / similarity (bee and bumblebee)

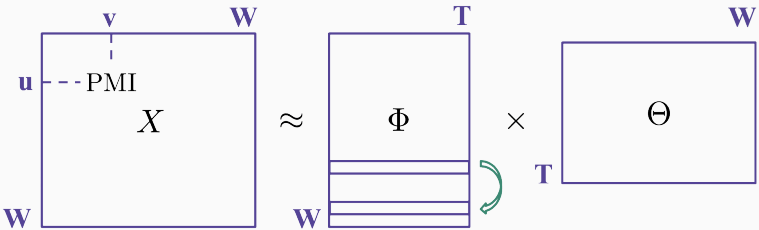
What other problems we have here? If we want to measure a cosine similarity between these long, sparse vectors, we conclude that these vectors are too long and too sparse. So we will try to reduce its dimensionality.



Schutze, H., & Pedersen, J. (1993). A vector model for syntagmatic and paradigmatic relatedness. In Making Sense of Words: Proceedings of the Conference, pp. 104-113, Oxford, England.

Vector Space Models of Semantics

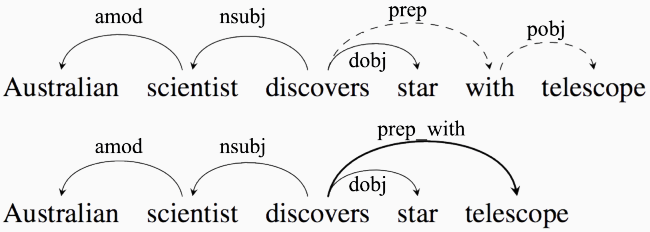
- Input (X): word-word co-occurrences (counts, PMI, ...)
- Method: dimensionality reduction (SVD - Singular Value Decomposition, ...)
- Output: similarity between vector representations of words



Turnay, P.D., Pantel, P.: from Frequency to Meaning: Vector Space Models of Semantics, 2010.

X will be factorized such that $T = 300$, for instance. There are different methods to do this factorization, and we will get into them later. What we need to know now is that we going to compare the rows of ϕ matrix instead of the original sparse rows of X matrix. This way we will get some measure of whether the words are similar, and this will be the output of our model. So far we have looked into how our words occur with other words from a sliding window. So we had some contexts, which would be words from a sliding window.

What is Context?



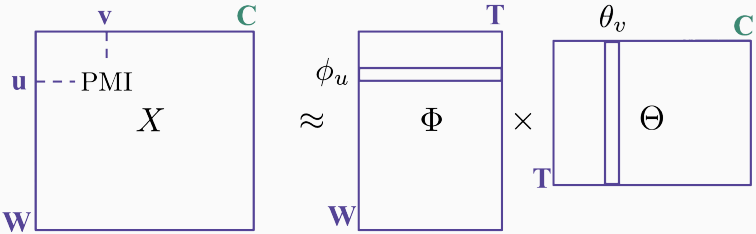
WORD	CONTEXTS
australian	scientist/amod^{-1}
scientist	australian/amod, discovers/nsubj^{-1}
discovers	scientist/nsubj, star/dobj, telescope/prep_with
star	discovers/dobj^{-1}
telescope	discovers/prep_with^{-1}

However, we can have a more complicated notion of the contexts. For example, we can have some syntax parses. This will allow us to discover syntactic dependencies between the words. The co-occurrence of some words can be related with this type of relationship between them. For example, “Australian” has co-occurred with scientist as a modifier.

Omer Levy, Yoav Goldber, Dependency-Based Word Embeddings, ACL-2014.

What is Context?

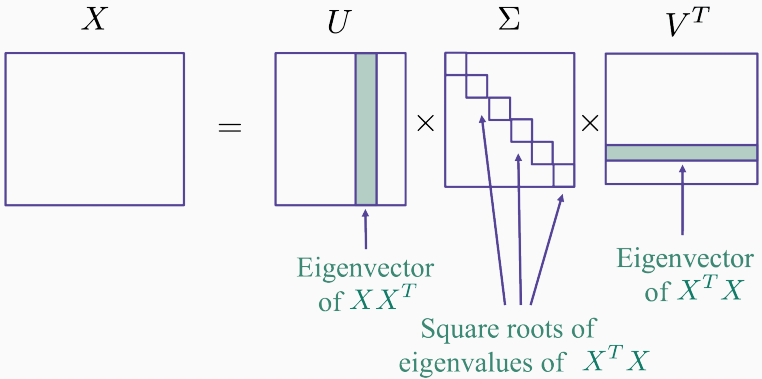
- C is a vocabulary of contexts, e.g. word/dependency
- But usually contexts are words form a sliding window
- Then $W = C$ and X is a symmetric matrix



So in this model, the contexts are words plus the type of the relationship. And in this case we will have not a square matrix, but some matrix of words by contexts. And for the contexts we will have the vocabulary of those word and class of modifier units. The syntax can really help to understand what is important to local context and what is not. What is just some random co-occurrences that are near, but that are not meaningful.

Explicit and Implicit Matrix Factorization

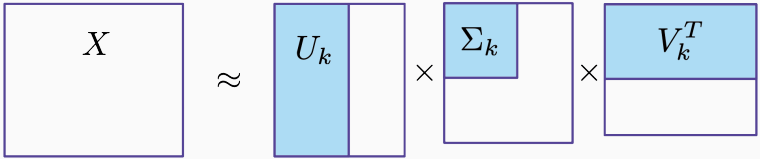
Singular Value Decomposition (SVD)



From linear algebra we know that every matrix can be factorized into those three matrices. The left and the right matrices will contain left and right eigenvectors. The metrics in the middle will be diagonal and the values on the diagonal will be related to eigenvalues. Those values on the diagonal will be sorted in the decreasing order. The number of those values on the diagonal will be the number of non-zero eigenvalues of X^TX, corresponding to the matrix rank.

Truncated SVD

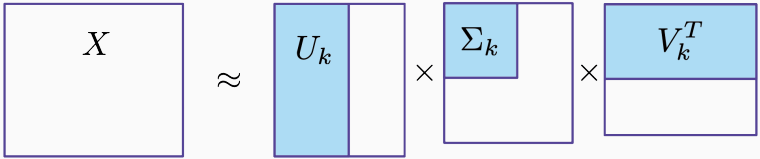
Keep only the first k components: $\hat{X}_k = U_k \Sigma_k V_k^T$



Once the diagonal values of the middle matrix are sorted, we can keep only the first k components. This corresponds to the blue regions of the matrices, corresponding to a good approximation of the original X matrix.

Truncated SVD

Keep only the first k components: $\hat{X}_k = U_k \Sigma_k V_k^T$



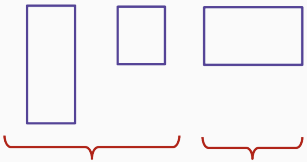
The blue part is the best approximation of rank k for X matrix, in terms of the loss that defined in the expression (a squared loss).

It's the best approximation of rank k in term of Frobenius norm:

$$||X - \hat{X}||_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} (x_{ij} - \hat{x}_{ij})^2}$$

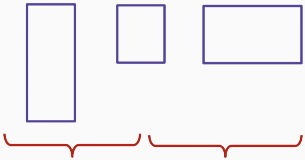
How to Use It?

Option 1:



$$\Phi = U_k \Sigma_k \qquad \Theta = V_k^T$$

Option 2:

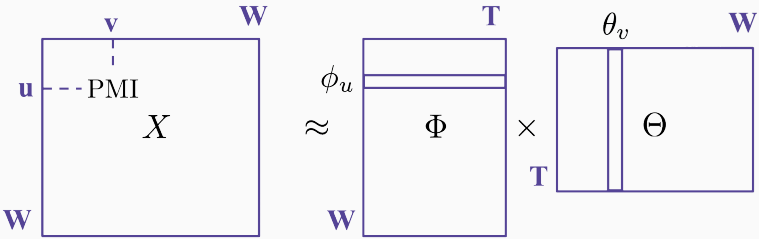


$$\Phi = U_k \sqrt{\Sigma_k} \qquad \Theta = \sqrt{\Sigma_k} V_k^T$$

Our goal is to get a factorization in two matrices but we got three. We can use some heuristics to get only two.
Option 1: The first and second matrices will be combined in the ϕ matrix, and the last one will be θ.
Option 2: The diagonal matrix in between should be honestly split between the two matrices ϕ and θ. The squared root is applied and one squared root goes to the left and another squared root goes to the right.

Vector Space Models of Semantics

- **Input:** word-word co-occurrences (counts, PMI, ...)
- **Method:** dimensionality reduction (SVD, ...)
- **Output:** similarity between vector representations of words



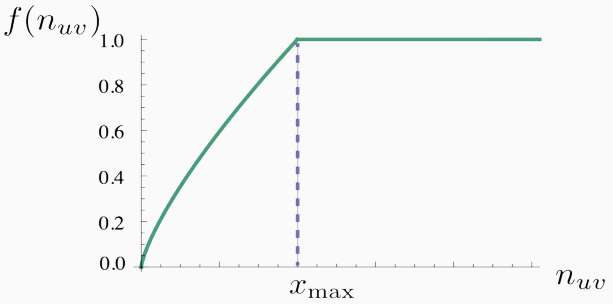
Summarizing: to build models of distributional semantics, we start with a word co-ocurrence matrix filled with PMI values. SVD can factorize the matrix X into ϕ and θ matrices. ϕ_u and θ_u vectors will be our embeddings. If we multiply ϕ_u and θ_u, we get the value of PMI in the X matrix.

Turnay, P.D., Pantel, P.: from Frequency to Meaning: Vector Space Models of Semantics, 2010.

Weighted Squared Loss: GloVe

Fill X with $\log n_{uv}$ and try another objective:

$$\sum_{u \in W} \sum_{v \in W} f(n_{uv}) \left(\langle \phi_u, \theta_v \rangle + b_u + b'_v - \log n_{uv}\right)^2 \to \min_{\phi_u, \theta_v, b_u, b'_v}$$



Pennington et. al. GloVe: Global Vectors for Word Representation, 2014.

GloVe: another approach to get the factorization of X

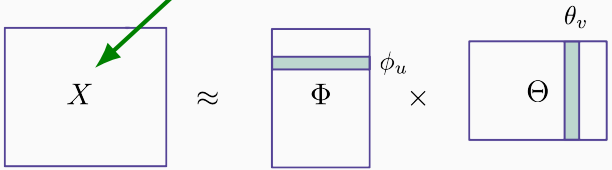
The goal is to estimate the parameters allowing to minimize the loss function using stochastic gradient descent, instead of using algebraic factorization. Every element of the matrix X is treated as an example. So we take an element, we perform one step of gradient descent and then update the parameters and take another element, and proceed in this way.

The f function provides weights, so if the word concurrences are higher, then this particular element of the matrix is more important to be approximated exactly. The f function stops increasing because very frequent words maybe corresponds to stop words that are not important. The green part corresponds to the factorization task. The red part is the original matrix.

Skip-gram Negative Sampling (SGNS) as Implicit Matrix Factorization

SGNS objective is maximized when $\langle \phi_u, \theta_v \rangle$ is equal to shifted Pointwise Mutual Information:

$$sPMI = \log \frac{n_{uv} n}{n_u n_v} - \log k$$



Another approach to get matrix X factorization

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

Word2vec and Doc2vec

Word2vec

Two architectures:

- CBOW (Continuous Bag-of-words):

$$p(w_i|w_{i-h}, \dots w_{i+h})$$

- Continuous Skip-gram:

$$p(w_{i-h}, \dots w_{i+h}|w_i)$$

Two ways to avoid softmax:

- Negative sampling
- Hierarchical softmax

Open-source and fast: `code.google.com/archive/p/word2vec/`

Different implementations of word2vec software package offer different architectures. One variant would be continuous bag-of-words. It means that we try to predict one word given the context words. Another option would be to do vice versa and predict context words given some words and this one will be called skip-gram.
Then softmax computation is usually too slow. So there are some alternatives: negative sampling and hierarchical softmax.

Evaluation: Word Similarities

How do we test that *similar* words have *similar vectors*?

- Linguists know a lot about what is “similar”
- We can use *human judgements* for word pairs
- Compare *Spearman’s correlation* between two lists:

tiger	tiger	10.00
media	radio	7.42
tiger	cat	7.37
train	car	6.31
...	...	...

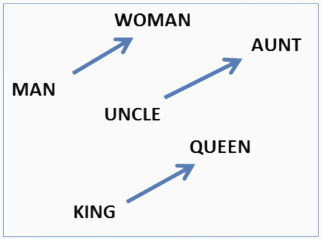
tiger	tiger	$\cos(\phi_u, \phi_v)$
media	radio	...
tiger	cat	...
train	car	...
...	...	...

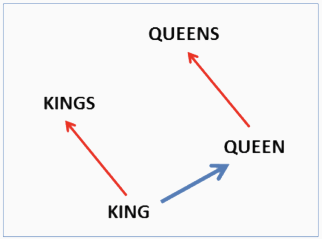
To get a measure of similarity between words we can apply cosine similarity to vectors embeddings. How can we test the model? We can use human judgements. There are data sets provided by linguists that look (left table). For example, they say that tiger and tiger are super similar. And media and radio are also similar, but not to that extent, and so on. We can compare these measures with the similarities obtained by the model. This comparing can be made with Spearman’s correlation. Alternatively, we can use extrinsic evaluation. For example, we could build a ranking system and then apply word similarities, compute the quality of the ranking system, and use this to evaluate the model.

Evaluation: Word Analogies

- In cognitive science well known as *relational similarity* (vs. *attributional similarity*)
- a : a' is as b : b' (man : woman is as king : ?)

$$\cos(b - a + a', x) \rightarrow \max_{x}$$





We can perform arithmetic operations over these vectors. Like king minus man plus woman, we get some other vector, and the closest word for this vector will be queen. This way, we can understand relations between the words. We can understand that man to woman is related in the same way as king to queen. This is called relational similarity. On the contrary, the similarity that we have been discussing up to this moment was called attributional similarity.

Gentner, D. Structure-mapping: A theoretical framework for analogy. Cognitive Science, 1983.
Mikolov et. al. Linguistic Regularities in Continuous Space Word Representations, 2013.

Word Similarity Task Performance

- For word similarity task, count-based methods (PPMI, SVD) perform on par with predictive methods (GloVe, SGNS)

win	Method	WordSim Similarity	WordSim Relatedness	Bruni et al. MEN	Radinsky et al. M. Turk
2	PPMI	.732	**.699**	.744	.654
	SVD	.772	.671	**.777**	.647
	SGNS	**.789**	.675	.773	**.661**
	GloVe	.720	.605	.728	.606
5	PPMI	.732	**.706**	.738	**.668**
	SVD	.764	.679	**.776**	.639
	SGNS	**.772**	.690	.772	.663
	GloVe	.745	.617	.746	.631

How to evaluate word analogies task? Usually we rely on human judgments. The table presents the performance of different models on these two tasks. The methods perform really similar. Different columns correspond to different datasets of word pairs. Old methods like SVD are not much worse than very recent methods like SGNS.

win is the width of the window for co-occurrences collection

Levy et. al. Improving distributional similarity with lessons learned from word embeddings, 2015.

Word Similarity Task Performance

- Word analogy task is solved with 70% average accuracy

win	Method	Google Add/Mul	MSR Add/Mul
2	PPMI	.552/.677	.306/.535
	SVD	.554/.591	.408/.468
	SGNS	.676/**.689**	.617/**.644**
	GloVe	.649/.666	.540/.591
5	PPMI	.518/.649	.277/.467
	SVD	.532/.569	.369/.424
	SGNS	.692/**.714**	.605/**.645**
	GloVe	.700/.712	.541/.599

Two data sets, one from Google and another from Microsoft Research. For Google data set the accuracy is about 70%, which means that in 70%, we can guess the right word correctly. For example, we can guess that king minus man plus woman is equal to queen.

Add is the way of analogy solving that we discussed. Mull is a modification.

Levy et. al. Improving distributional similarity with lessons learned from word embeddings, 2015.

Paragraph2vec aka doc2vec

DM (Distributed Memory):

$$p(w_i|w_{i-h}, \dots w_{i+h}, d)$$

DBOW (Distributed Bag Of Words):

$$p(w_{i-h}, \dots w_{i+h}|d)$$

paragraph2vec and doc2vec are names for the same model. Paragraph2vec name goes from the paper. Previously in word2vec, we had two architectures: we produce contexts given some focus word, or a focus word given some contexts. Now we will treat documents the same way as we treated words. So we have some document d in some fixed vocabulary of the documents and then we will build embeddings for the documents. Now there are again two architectures. The first, DM, stands for providing the probabilities of focus words, given everything we have. And DBOW architecture stands for providing the probability of the context given the document. So the last one is somehow similar to skip-gram model. But instead of the focus words, we condition on the document. This model can be used to provide documents similarities.

Doc2vec Usage

The doc2vec model may be used in the following way:

- Training phase:
  - a set of documents is required
  - a word vector is generated for each word, and a document vector d is generated for each document
  - the model also trains weights for a softmax hidden layer
- Inference phase: a new document may be presented, and all weights are fixed to calculate the document vector d

Evaluation: Document Similarities

How do we test that similar documents have similar vectors?

- arXiv triplets: paper A, similar paper B, dissimilar paper C
- Measure the accuracy of guessing the dissimilar paper

Title	Cosine Similarity
Evaluating Neural Word Representations in Tensor-Based Compositional Settings	0.771
Polyglot: Distributed Word Representations for Multilingual NLP	0.764
Lexicon Infused Phrase Embeddings for Named Entity Resolution	0.757
A Convolutional Neural Network for Modelling Sentences	0.747
Distributed Representations of Words and Phrases and their Compositionality	0.740
Convolutional Neural Networks for Sentence Classification	0.735
SimLex-999: Evaluating Semantic Models With (Genuine) Similarity Estimation	0.735
Exploiting Similarities among Languages for Machine Translation	0.731
Efficient Estimation of Word Representations in Vector Space	0.727
Multilingual Distributed Representations without Word Alignment	0.721

How can we test that we document similarities provided by the model are good? We need a test set. This data set provides triplets from arXiv papers. In each triplet, two documents are similar and third one is dissimilar. The task is to predict which is the dissimilar one.

arXiv nearest neighbours to the paper “Distributed Representations of Sentences and Documents” using Paragraph Vectors

Andrew Dai, Cristopher Olah, Quoc Le. Document Embedding with Paragraph Vectors, CoRR, 2015.

Summary

Methods:

- *word2vec*: SGNS, CBOW, ...
- *doc2vec*: DBOW, DM, ...
- Python library for both: `https://radimrehurek.com/gensim/`

Evaluation:

- Word similarity and analogy
- Document similarity
- *Interpretability of the components*
- *Geometry of the embeddings space*

Count-based and predictive approaches are not so different

Word Analogies

Word Analogy Task

- Previously we saw how word2vec can represent words
- word2vec model is trained in an unsupervised manner
  - it means that the model sees texts and obtain word vectors
- However, there are some problems, especially for world analogies task

An Interesting Property of word2vec

Learn word vectors by predicting their contexts (or vice-versa).

Obtain vectors that solve word analogies:

- king – man + woman = queen
- Moscow – Russia + France = Paris

Demo: `https://rare-technologies.com/word2vec-tutorial/`

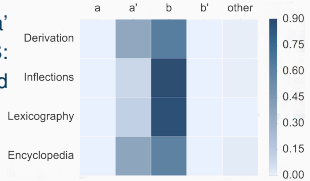
Word vectors have interesting properties. For example, if we take the vector for king, we'll subtract the vector for man and add the vector for woman, you will get a result vector. The closest word to this vector will be queen. The model could understand the meaning of the language, even though we did not have this in the data explicitly.

Analogy Task

$$\cos b - a + a', x \rightarrow \max_{x \notin \{a, a', b\}}$$

king - man + woman = king:

Figure: the result of a - a' + b calculation on BATS: source vectors a, a', and b are not excluded



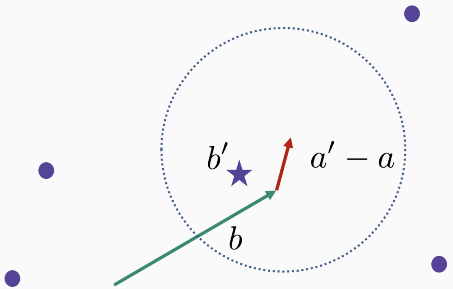
The Fig. shows that if we do not exclude the source vectors, b is the most likely to be predicted; in derivational and encyclopedic categories a' is also possible in under 30% of cases. b' is as unlikely to be predicted as a, or any other vector.

Rogers et. al. The (Too Many) Problems of Analogical Reasoning with Word Vectors, 2017.

How this of the closest word is performed? We want to maximize cosine similarity between the result of the expression and all the candidates in our space but we exclude three candidates from this search: king, man and woman do not participate in this search.
Let's now perform the maximization in the whole space. The picture shows what would be the closest neighbour to the arithmetic expression in case of the search. The color shows the ratio. The names on the left correspond to different categories of analogies. The example about king will fall into Encyclopedia category. So when we do king - man + woman we get a vector that will be close to the b vector (king). It also can be close to a' vector (women). But never close to b' vector, which is the target vector (queen). In about 80% of different analogies, we find a vector which is close to b instead of the target b´ vector!

The Closest Neighbour of b is Good Enough?

For the plural noun category in the Google test set:
70% accuracy by taking the closest neighbour of the vector b!



Why when we exclude a, a' and b vectors, we find b' vectors and if we do not exclude them, we end up with b vector?
The shift vector a'-a seems to be close to 0. So when we add this shift vector to b we find the closest neighbour, which is actually b. But once b is excluded, the next closest neighbor is indeed b'. So maybe we can use a much more simple method to do this, for instance, the closest neighbour. Well, some people tried that and they said that for one particular category of analogies, the plural category which is apple to apples is same as orange to oranges. The strategy to take the closest neighbour of b results in 70% accuracy, a good accuracy with a dumb approach.

Linzen. Issues in evaluating semantic spaces using word analogies, 2016.

BATS Dataset

Inflectional morphology:

- *student:students, strong:stronger, follow:following, . . .*

Derivational morphology:

- *bake:baker, edit:editable, home:homeless, . . .*

Lexicographic semantics:

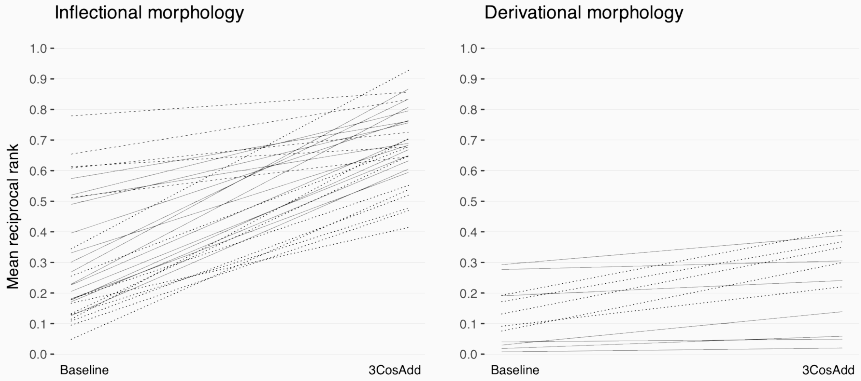
- Hypernyms, meronyms : peach:fruit, sea:water, . . .
- Antonyms, synonyms: up:down, clean:dirty, cry:scream

Encyclopedic semantics:

- Animals: cat:kitten, dog:bark, . . .
- Geography: Athens:Greece, Peru:Spanish, . . .
- People: Lincoln:president, Lincoln:American, . . .
- Other: blood:red, actor:actress, . . .

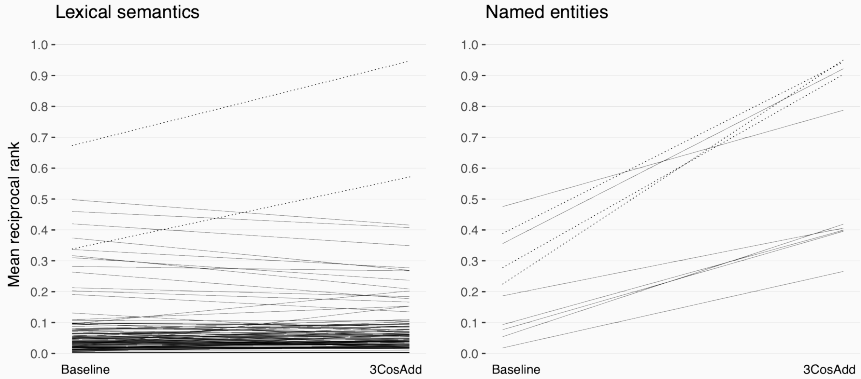
Different categories of analogies. How word2vec performs for different analogies? In the next slide word2vec is compared with a very simple baseline. The baseline would be to just to take the closest neighbour to one of the query words. For example, one line could correspond to apple to apples is the same as orange to oranges. The left point for every line is the performance of the baseline. And the right point of every line is the performance of word2vec. So it means that horizontal lines show you that word2vec is not better than base line. When the line has a high slope, it means that word2vec does a good job.

Performance by Categories



Finley et. al. What Analogies Reveal about Word Vectors and their Compositionality, 2017.

Performance by Categories



Finley et. al. What Analogies Reveal about Word Vectors and their Compositionality, 2017.

Summary

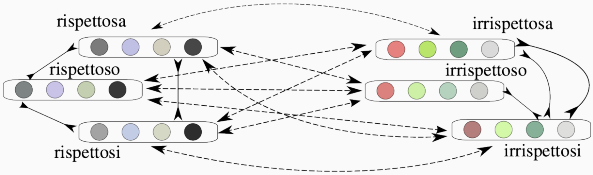
- *word2vec* works fine for word similarities
- But there are many questions with word analogies
- Be careful about hype!

Why Words? From Character to Sentence Embeddings

Morphology Can Help

Problems:

- Languages with rich morphology
- Low frequency words, OOV words



Use morphology to improve word embeddings

Sometimes it is necessary go to the sub-word level. We can use the rich morphology of some languages to improve the models with the help of linguistics. For example, some prefixes can change the meaning of the word to the opposite one - antonyms (such as relevant, irrelevant). On the other hand, there are suffixes that do not change the semantics of the words a lot. The paper referred presents the ideia that we can put the words that have some not important morphological changes together in the space, and on the opposite, let us try to have the embeddings for words that have some prefixes that change the meaning of the word completely. We can do this with some regularization of your model, or adding some loss, to make sure that you have this additional constraints.

I. Vulic et. al. Morph-fitting: Fine-Tuning Word Vector Spaces with Simple Language-Specific Rules, 2017.

FastText

Represent a word as a bag of character n-grams, e.g. for $n = 3$:

G_{where}: _wh, whe, her, ere, re_, _where_

Model a word vector as a sum of sub-word vectors:

SGNS: $sim(u, v) = \langle \phi_u, \theta_v \rangle$

FastText: $sim(u, v) = \sum_{g \in G_v} \langle \phi_u, \theta_g \rangle$

Code and pre-trained embeddings:

`https://fasttext.cc/`

P. Bojanowsky et al. Enriching Word Vectors with Subword Information, 2016.

But sometimes we don't have linguists to tell us what are the morphological patterns in the language. What can we do then? In this case, we can try to have a more brute-force approach. This would be to go to character n-grams. The FastText model proposed by Facebook research, is based on the following idea. Let us represent a word by a set of character n-grams, and include the word itself in this set as well. In the example, we have n-grams of size 3. Usually, we will have several n-values, like n from 3 to 6. So we will have n-grams of different length in the same set to represent the word. In methods like SGNS (Skip-Gram Negative Sampling) the similarity between words is represented by the product $\langle \phi_u, \theta_v \rangle$. But now, the words are represented by set of vectors. Since each n-gram character is represented by a vector, we can sum all those vectors. The FastText model provides a way to represent sub-word unions.

Sent2vec

First ideas:

- Average pre-trained word vectors (*word2vec, GloVe, etc*)
- Maybe use TF-IDF weights for averaging

We can build embeddings to represent sentences. Some simple ideas. First, we can use pre-trained word embeddings, from Word2vec, and average them to obtain the embedding of the sentence. An alternative would be consider a weighted average (e.g. TF-IDF weights). However these approaches use vectors trained with some other objectives, and they might not suit well for our task.

Sent2vec

Sent2vec:

- Learn sentence embedding as a sum of sub-sentence units:

$$sim(u, s) = \frac{1}{|G_s|} \sum_{g \in G_s} \langle \phi_u, \theta_g \rangle$$

where G_s is a set of word n-grams for the sentence s

Code and embeddings: `https://github.com/epfml/sent2vec`

Another idea would be to represent sentences as a sum of sub-sentence units. In Sent2vec, we represent the similarity between word and the sentence, and our training data will be those words that occur in some sentences. We have a positive pair, word occurred in a sentence. The negative example will be some word that occurs in some other sentence. So we assume that they are not similar. The similarity will be the sum over sub-unions. These unions are word n-grams – a BOW n-grams will represent a sentence. This model is similar to FastText model, but instead of character n-grams to represent words, you have word n-grams to represent sentences. Instead of a sum, we have an average (the sum divided by the size of the set).

Pagliardini et. al. Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features, 2017.

StarSpace

General framework:

- In different levels of the language we can apply similar ideas – StarSpace model apply this approach for all these levels
- The idea is that we have entities represented by features: words represented by character n-grams or sentences by word n-grams

Applications:

- Ranking, e.g. ranking web documents given a query
- Collaborative filtering-based recommendation
- Embedding graphs, e.g. Freebase

Wu et. al. StarSpace: Embed All The Things! 2017

StarSpace

- Content-based recommendation
  - Example: recommendation of movies – from a set of users represented as a set of bag of items that they like (e.g. a bag of movies), the model will learn how to embed users and movies in the same space.
- Text classification, e.g. sentiment analysis
  - From a set of documents as a BOW and corresponding labels, for example, sentiment labels, the model try to learn to produce correct labels for documents. The similarity measure between the documents and the label should be high if this label can be found in the supervised data for this document, and low vice-versa. The model learns to get the embeddings for labels, documents, and words in the same space.
- Learning word, sentence or document embeddings

Code and tutorials: `github.com/facebookresearch/Starspace`

StarSpace

Mode 3 (sentence embeddings):

Use case: learn pairwise similarity from collections of similar objects, e.g. sentence similarity

Data format: each line is a collection of similar sentences (supervised data):

sent1_word1 sent1_word2 ... <tab> sent2_word1 sent2_word2 ...

Training:

- Each sentence is represented as a bag of features (words or n-grams). They are embedded to predict sentence similarity
- Similar sentence pairs are taken from the collections
- Dissimilar sentence pairs are sampled at random
- As result of training we get embeddings for words and sentences

Deep Learning

The main trends in DL to sentence representation:

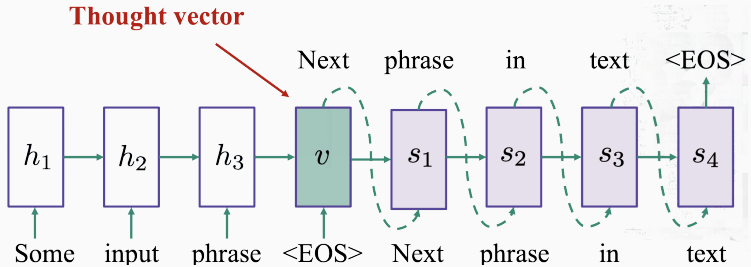
- Recurrent Neural Networks (sequence modelling, popular in NLP)
- Convolutional Neural Networks (much faster than RNN)
- Recursive Neural Networks (use hierarchical structure)

Linguistic structure is back:

- Morphology can help to build word embeddings
- Recursive Neural Networks, Tree-LSTMs, DAG-LSTMs, etc. use syntax to help building the representation of the sentence, span annotations, co-reference links ...

Skip-thought Vectors

- Predict next sentence
  - the sentence is encoded with a Recurrent NN and get a hidden representation, called thought-vector (embedding of the sentence); from this vector, we try to generate the next sentence with some language model. This vector will be the input of a conditional NN model.
- RNN encoder-decoder architecture



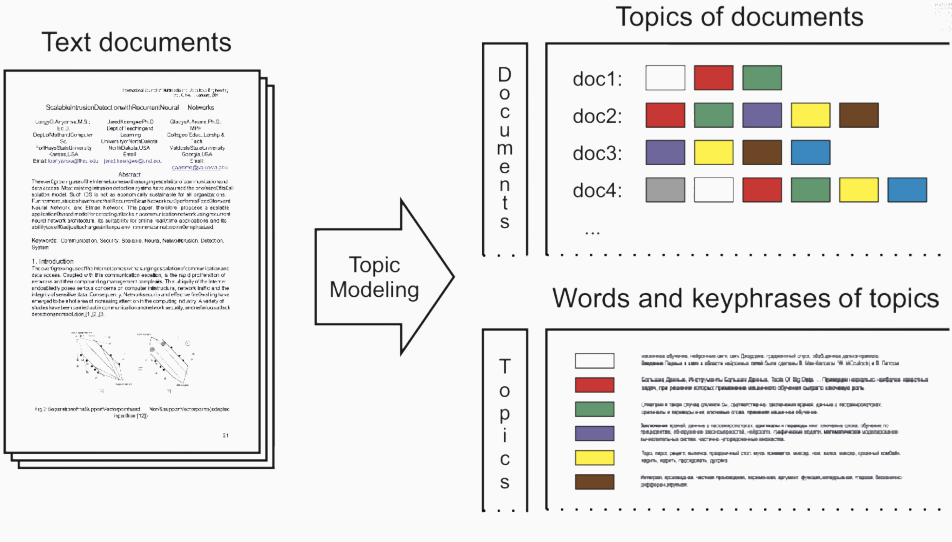
Kiros et. al. Skip-Thought Vectors, 2015, https://github.com/ryankiros/skip-thoughts.

Topic Modeling

From Texts to Topics

- Topic modeling is an alternative way to build vector representations for your document collections
- From a text collection we want to build some hidden representation
- Each document is described by topics, and each topic is described by words (e.g. topic weather can be described by sky, rain, sun)
- This problem can described as soft-biclustering: *clustering* because we cluster both words and documents; *soft* because we build probability distributions to softly assign words and documents to topics

From Texts to Topics



The Formal Task

Given:

- Collection of texts as bags-of-words:

  n_{wd} is a count of the word w in the document d

Find:

- Probabilities of word in topics:

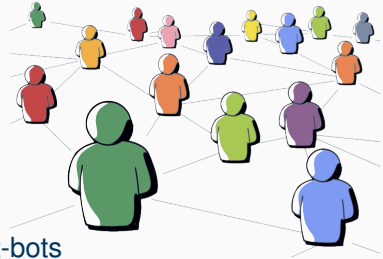
  $\phi_{wt} = p(w|t)$ ⟵ **Definition of a topic**

- Probabilities of topics in documents:

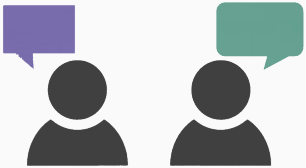
  $\theta_{td} = p(t|d)$

Where Do We Need TM?

- Exploration and navigation through large text collections
- Social network analysis



- Dialogue manager in chat-bots



Why Do We Need TM?

Topic models provide hidden semantic representation of texts

Many more applications:

- Categorization and classification of texts
- Document segmentation and summarization
- News flows aggregation and analysis
- Recommender systems
- Image captioning
- Bioinformatics (genome annotation)
- Exploratory search
- . . .

Generative Model of Texts

Probabilistic Latent Semantic Analysis (PLSA):

$$p(w|d) = \sum_{t \in T} p(w|t,d)p(t|d) = \sum_{t \in T} p(w|t)p(t|d)$$

Law of total probability

$p(w) = \sum_{t \in T} p(w|t)p(t)$

Assumption of conditional independence

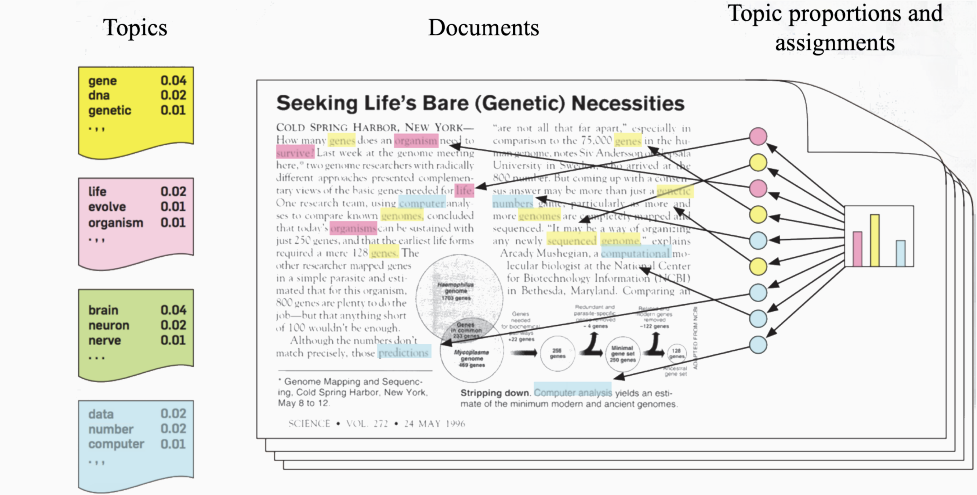
$p(w|t,d) = p(w|t)$

Generative model of text
From a probability distribution of topics for the document we first decide what would be the topic for the next word. After decided about the next topic, we can draw a certain word from the probability distribution for this topic. So, this model assumes that the text is generated not by authors, not just by handwriting, but by some probability procedure.

Notation:

- *w – word*
- *d – document*
- *t – topic*

Generative Model of Texts

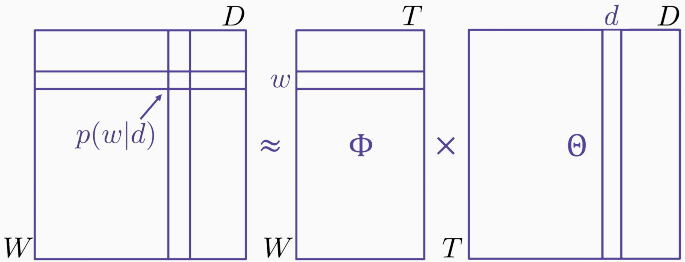


David Blei, Probabilistic topic models, 2012.

Matrix Way of Thinking

Probabilistic Latent Semantic Analysis:

$$p(w|d) = \sum_{t \in T} p(w|t)p(t|d) = \sum_{t \in T} \phi_{wt}\theta_{td}$$



The first matrix represents a probability distributions of words in documents. These probabilities can be computed by counting how many times each word occurs in each document and then normalize these counts. Now you need to factorize this real matrix into two matrices of parameters ϕ and θ. Matrix ϕ is about probability distributions over words and θ matrix contains probability distributions over topics. Every column in matrix θ is a probability distribution. This corresponds to the matrix form of the formula.

Approaches to Topic Modelling

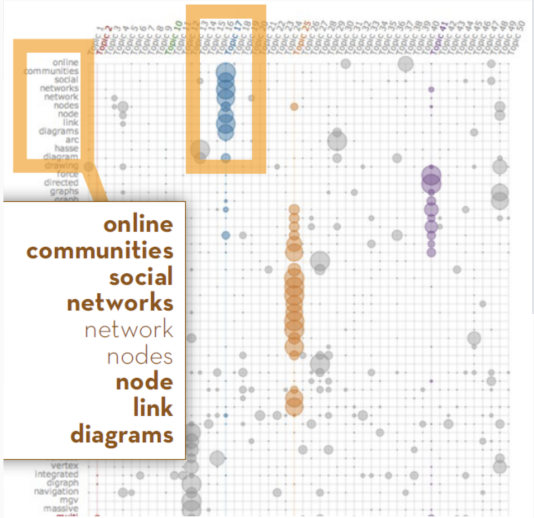
Approaches to Topic Modelling

- Bayesian methods and graphical models
  - Bayesian inference to provide new topic models
- Hierarchical topic models
  - To build a hierarchy of topics
- Dynamic topic models
  - Models that consider that topics can evolve over time - the probability of the topics changes
- Multilingual topic models
  - Topics are not dependent on the language - the same topic can be expressed in different languages with different terms - the model is trained on parallel data (documents about the same topic but expressed with different terms)

Libraries for Topic Modelling

- **BigARTM** is an open-source library for Additive Regularization of Topic Models, `http://bigartm.org`
- **Gensim** is a library of text analysis for Python, `https://radimrehurek.com/gensim`
- **MALLET** is a library of text analysis for Java, `http://mallet.cs.umass.edu`
- **Vowpal Wabbit** has a fast implementation of online LDA, `http://hunch.net/~vw/`

Topic Visualization



The probability distributions are hard to represent in such a way that people can understand it. This is an example of how to visualize ϕ metrics. We have words by topic's metrics and we can see that we group those words that correspond to every certain topic together so that we can see that this blue topic is about these terms.

J. Chuang, C. D. Manning, J. Heer – Termite: Visualization Techniques For Assessing Textual Topic Models, 2012

Text Visualization

380 ways to visualize: `http://textvis.lnu.se`