

Distributional Word Embeddings in Text Classification

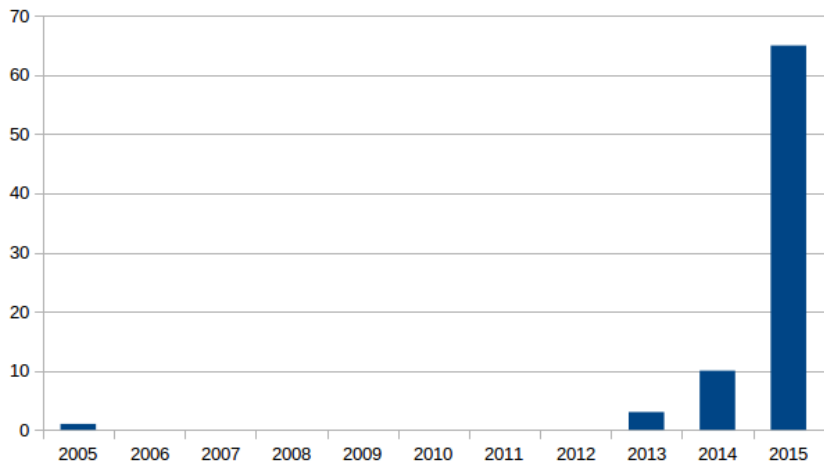
Andrey Kutuzov
University of Oslo
Language Technology Group

September 1, 2016



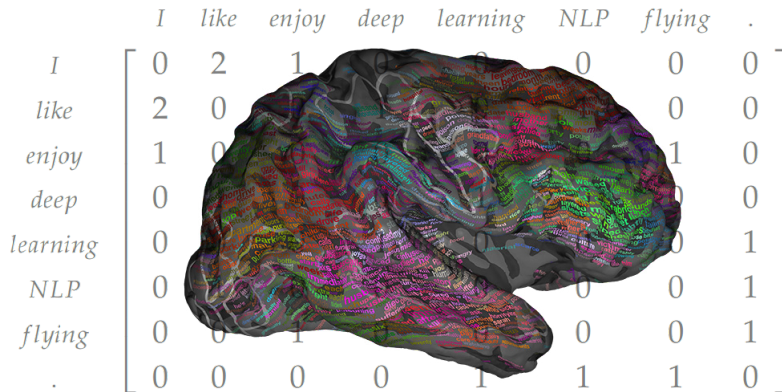
Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations
- 5 Model formats
- 6 What can we find in the models?
- 7 More than words: representing texts

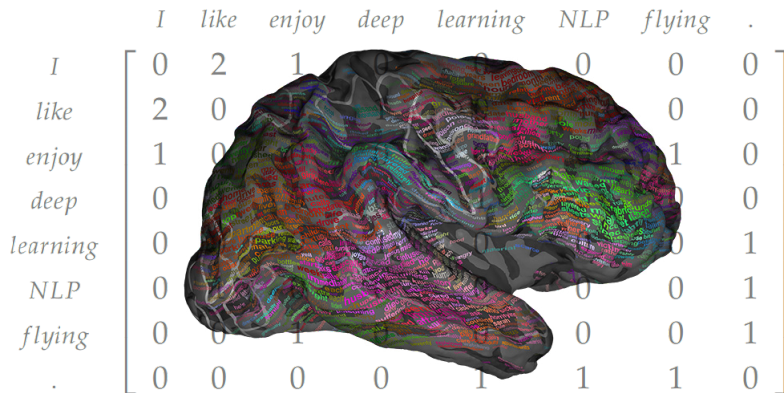


Number of publications on word embeddings in Association for Computational Linguistics Anthology (<http://aclanthology.info/>)

Mapping words in brain

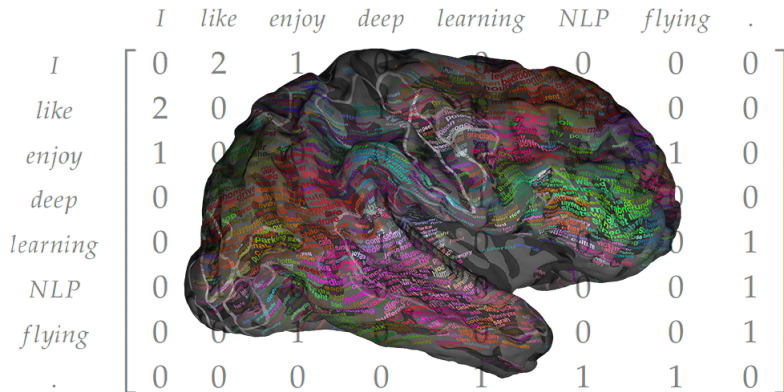


Mapping words in brain



We want a **machine** to imitate human brain and **understand meaning of words**.

Mapping words in brain



We want a **machine** to imitate human brain and **understand meaning of words**.

Then, it will be able to classify texts, among other things.

How to design it?

Contents

- 1 Our motivation
- 2 TL:DR**
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations
- 5 Model formats
- 6 What can we find in the models?
- 7 More than words: representing texts

Distributional semantic models for Russian

Russian National Corpus web site is <http://ruscorpora.ru>. To acknowledge their efforts, we launched *RusVectōrēs* web service ('Russian vectors' in Latin)

Distributional semantic models for Russian

Russian National Corpus web site is <http://ruscorpora.ru>. To acknowledge their efforts, we launched *RusVectōrēs* web service ('Russian vectors' in Latin)

<http://ling.go.mail.ru/dsm>

Distributional semantic models for English (and Norwegian)

<http://ltr.uio.no/semvec>

You can entertain yourself during the tutorial :-)

Later we will look closer at the features of these services.

Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings**
- 4 Existing approaches and implementations
- 5 Model formats
- 6 What can we find in the models?
- 7 More than words: representing texts

Tiers of linguistic analysis

Tiers of linguistic analysis

Computational linguistics can comparatively easily model lower tiers of language:

- ▶ **graphematics** – how words are spelled

Tiers of linguistic analysis

Computational linguistics can comparatively easily model lower tiers of language:

- ▶ **graphematics** – how words are spelled
- ▶ **phonetics** – how words are pronounced

Tiers of linguistic analysis

Computational linguistics can comparatively easily model lower tiers of language:

- ▶ **graphematics** – how words are spelled
- ▶ **phonetics** – how words are pronounced
- ▶ **morphology** – how words inflect

Tiers of linguistic analysis

Computational linguistics can comparatively easily model lower tiers of language:

- ▶ **graphematics** – how words are spelled
- ▶ **phonetics** – how words are pronounced
- ▶ **morphology** – how words inflect
- ▶ **syntax** – how words interact in sentences

Distributional hypothesis and word embeddings

But how to represent meaning?

But how to represent meaning?

- ▶ **Semantics** is difficult to represent formally.

Distributional hypothesis and word embeddings

But how to represent meaning?

- ▶ **Semantics** is difficult to represent formally.
- ▶ We need machine-readable word **representations**.
- ▶ Words which are **similar in their meaning** should possess **mathematically similar representations**.

Distributional hypothesis and word embeddings

But how to represent meaning?

- ▶ **Semantics** is difficult to represent formally.
- ▶ We need machine-readable word **representations**.
- ▶ Words which are **similar in their meaning** should possess **mathematically similar representations**.
- ▶ *'Judge'* is similar to *'court'* but not to *'kludge'*, even though their surface form suggests the opposite.

Distributional hypothesis and word embeddings

But how to represent meaning?

- ▶ **Semantics** is difficult to represent formally.
- ▶ We need machine-readable word **representations**.
- ▶ Words which are **similar in their meaning** should possess **mathematically similar representations**.
- ▶ ‘*Judge*’ is similar to ‘*court*’ but not to ‘*kludge*’, even though their surface form suggests the opposite.
- ▶ Why so?

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Distributional hypothesis and word embeddings

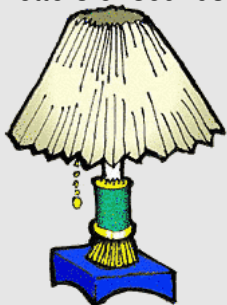
Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:

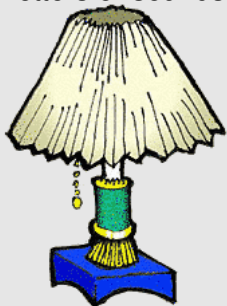


► **lantern**

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:

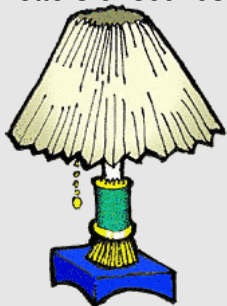


- ▶ lantern
- ▶ lykt

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:



- ▶ lantern
- ▶ lykt
- ▶ ЛАМПА

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:

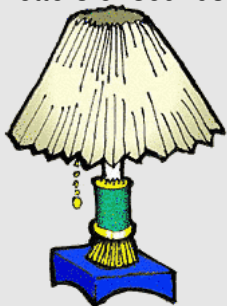


- ▶ lantern
- ▶ lykt
- ▶ ЛАМПА
- ▶ lucerna

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:

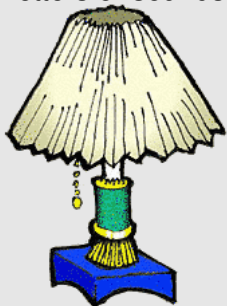


- ▶ lantern
- ▶ lykt
- ▶ лампа
- ▶ lucerna
- ▶ гэрэл

Distributional hypothesis and word embeddings

Arbitrariness of a linguistic sign

Unlike **road signs**, words do not possess a direct link between form and meaning. '*Lantern*' concept can be expressed by any sequence of letters or sounds:



- ▶ lantern
- ▶ lykt
- ▶ лампа
- ▶ lucerna
- ▶ гэрэл
- ▶ ...

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

1. **Manually building ontologies** (knowledge-based approach). Works top-down: from abstractions to real texts.

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

1. **Manually building ontologies** (knowledge-based approach). Works top-down: from abstractions to real texts.
2. **Extracting semantics from usage patterns in text corpora** (distributional approach). Works bottom-up: from real texts to abstractions.

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

1. **Manually building ontologies** (knowledge-based approach). Works top-down: from abstractions to real texts.
2. **Extracting semantics from usage patterns in text corpora** (distributional approach). Works bottom-up: from real texts to abstractions.

The **second** approach is today's topic.

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

1. **Manually building ontologies** (knowledge-based approach). Works top-down: from abstractions to real texts.
2. **Extracting semantics from usage patterns in text corpora** (distributional approach). Works bottom-up: from real texts to abstractions.

The **second** approach is today's topic.

Meaning is actually a sum of contexts: *'You shall know a word by the company it keeps'* [Firth, 1957]

Distributional hypothesis and word embeddings

How we can make a computer understand this?

Possible data sources

The methods of computationally representing semantic relations in natural languages fall into two large groups:

1. **Manually building ontologies** (knowledge-based approach). Works top-down: from abstractions to real texts.
2. **Extracting semantics from usage patterns in text corpora** (distributional approach). Works bottom-up: from real texts to abstractions.

The **second** approach is today's topic.

Meaning is actually a sum of contexts: *'You shall know a word by the company it keeps'* [Firth, 1957]

Distributional semantics models (DSMs) are built upon lexical co-occurrences in a large training corpus (**lots** of natural texts).

Distributional hypothesis and word embeddings

In **distributional semantics**, meanings of particular words are represented as vectors of real values derived from **frequency of their co-occurrences with other words in the training corpus**.

Distributional hypothesis and word embeddings

In **distributional semantics**, meanings of particular words are represented as vectors of real values derived from **frequency of their co-occurrences with other words in the training corpus**.

- ▶ Words are **axes** (dimensions) in multi-dimensional semantic space.

Distributional hypothesis and word embeddings

In **distributional semantics**, meanings of particular words are represented as vectors of real values derived from **frequency of their co-occurrences with other words in the training corpus**.

- ▶ Words are **axes** (dimensions) in multi-dimensional semantic space.
- ▶ At the same time, words are **vectors** or points in this space.

Distributional hypothesis and word embeddings

A simple example of a symmetric **co-occurrence matrix**:

	vector	meaning	hamster	corpus	weasel	animal
<i>vector</i>	0	10	0	8	0	0
<i>meaning</i>	10	0	1	15	0	0
<i>hamster</i>	0	1	0	0	20	14
<i>corpus</i>	8	15	0	0	0	2
<i>weasel</i>	0	0	20	0	0	21
<i>animal</i>	0	0	14	2	21	0

Distributional hypothesis and word embeddings

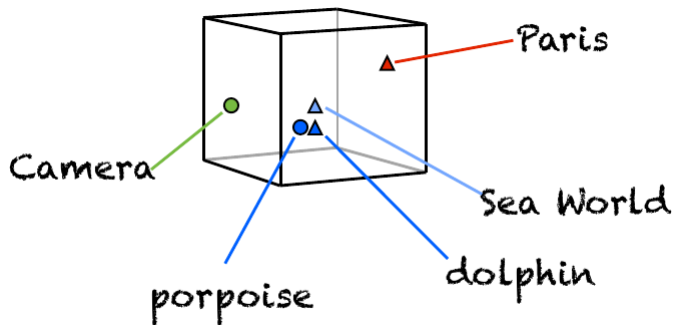
A simple example of a symmetric **co-occurrence matrix**:

	vector	meaning	hamster	corpus	weasel	animal
<i>vector</i>	0	10	0	8	0	0
<i>meaning</i>	10	0	1	15	0	0
<i>hamster</i>	0	1	0	0	20	14
<i>corpus</i>	8	15	0	0	0	2
<i>weasel</i>	0	0	20	0	0	21
<i>animal</i>	0	0	14	2	21	0

We produced meaningful representations in a completely **unsupervised** way!

Distributional hypothesis and word embeddings

Similar words are close to each other in the space defined by their typical co-occurrences



Curse of dimensionality

Curse of dimensionality

- ▶ With large corpora, we can end up with **millions of dimensions** (axes, words).

Curse of dimensionality

- ▶ With large corpora, we can end up with **millions of dimensions** (axes, words).
- ▶ But the vectors are very **sparse**, most components are zero.

Distributional hypothesis and word embeddings

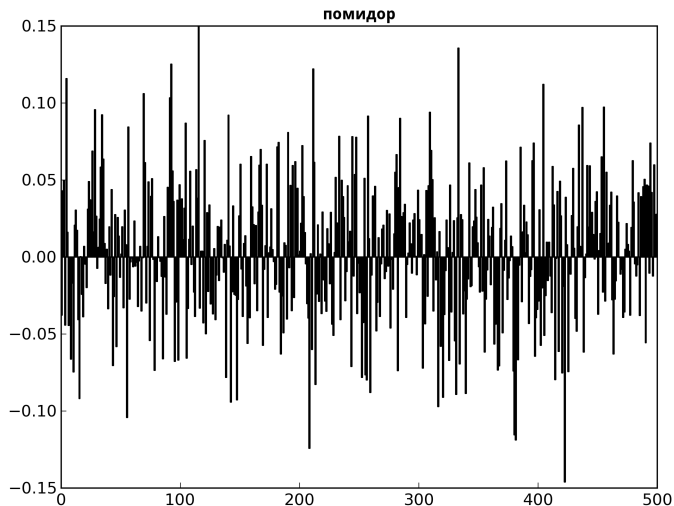
Curse of dimensionality

- ▶ With large corpora, we can end up with **millions of dimensions** (axes, words).
- ▶ But the vectors are very **sparse**, most components are zero.
- ▶ One can **reduce vector sizes** to some reasonable values, and still retain meaningful relations between them.

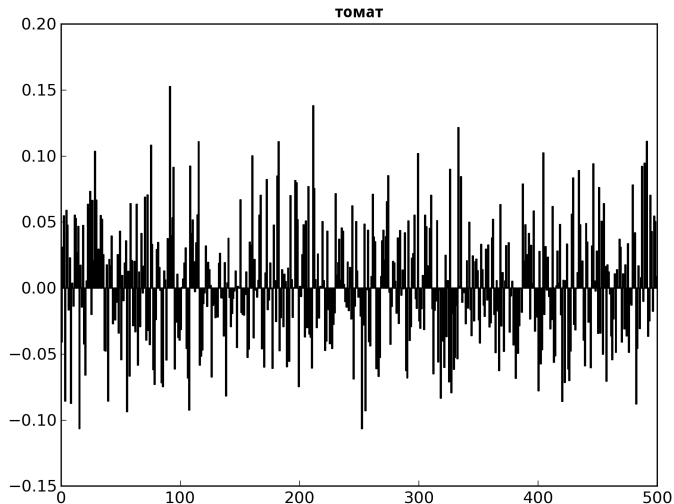
Curse of dimensionality

- ▶ With large corpora, we can end up with **millions of dimensions** (axes, words).
- ▶ But the vectors are very **sparse**, most components are zero.
- ▶ One can **reduce vector sizes** to some reasonable values, and still retain meaningful relations between them.
- ▶ Such dense vectors are called '**word embeddings**'.

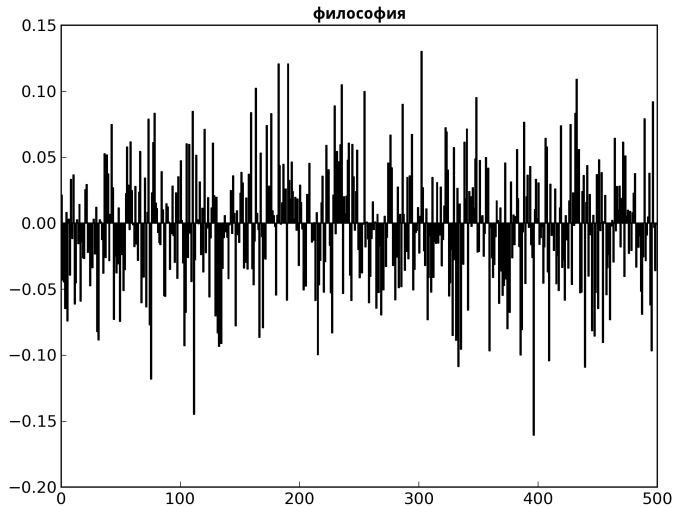
Distributional hypothesis and word embeddings



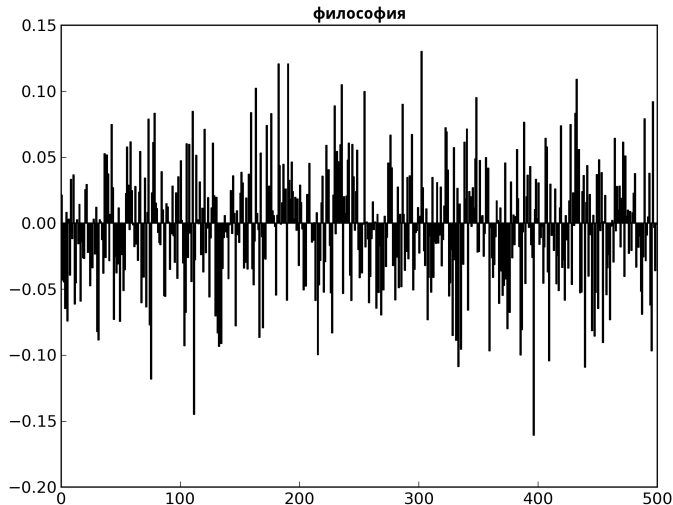
Distributional hypothesis and word embeddings



Distributional hypothesis and word embeddings



Distributional hypothesis and word embeddings



Can we prove that **tomatoes** are more similar to each other than to **philosophy**?

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

- ▶ Similarity lowers as **angle between word vectors grows**.

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

- ▶ Similarity lowers as **angle between word vectors grows**.
- ▶ Similarity grows as **the angle lessens**.

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

- ▶ Similarity lowers as **angle between word vectors grows**.
- ▶ Similarity grows as **the angle lessens**.

$$\cos(w1, w2) = \frac{\vec{V}(w1) \times \vec{V}(w2)}{|\vec{V}(w1)| \times |\vec{V}(w2)|} \quad (1)$$

$$\cos(\text{tomat}, \text{philosophy}) = 0.00698$$

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

- ▶ Similarity lowers as **angle between word vectors grows**.
- ▶ Similarity grows as **the angle lessens**.

$$\cos(w1, w2) = \frac{\vec{V}(w1) \times \vec{V}(w2)}{|\vec{V}(w1)| \times |\vec{V}(w2)|} \quad (1)$$

$$\cos(\text{tomat}, \text{philosophy}) = 0.00698$$

$$\cos(\text{pomidor}, \text{philosophy}) = -0.03429$$

Distributional hypothesis and word embeddings

Semantic similarity between words is usually measured by **cosine similarity** of their corresponding vectors.

- ▶ Similarity lowers as **angle between word vectors grows**.
- ▶ Similarity grows as **the angle lessens**.

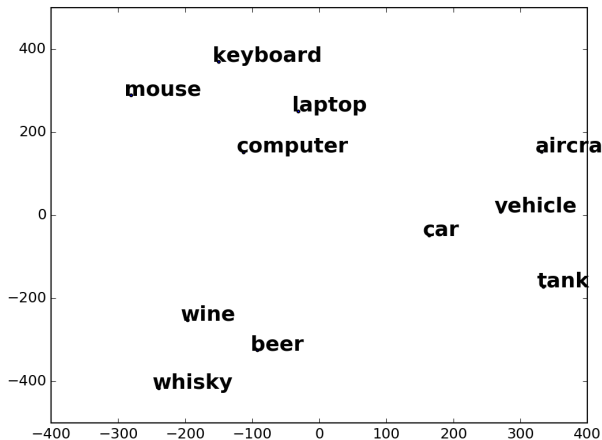
$$\cos(w1, w2) = \frac{\vec{V}(w1) \times \vec{V}(w2)}{|\vec{V}(w1)| \times |\vec{V}(w2)|} \quad (1)$$

$$\cos(\text{tomat}, \text{philosophy}) = 0.00698$$

$$\cos(\text{pomidor}, \text{philosophy}) = -0.03429$$

$$\cos(\text{tomat}, \text{pomidor}) = 0.65049$$

Distributional hypothesis and word embeddings



Embeddings reduced to 2 dimensions and visualized by t-SNE algorithm

[Van der Maaten and Hinton, 2008]

Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations**
- 5 Model formats
- 6 What can we find in the models?
- 7 More than words: representing texts

Existing approaches and implementations

Main approaches to produce word embeddings

1. Point-wise mutual information (**PMI**) association matrices, factorized by **SVD** (so called *count-based models*) [Bullinaria and Levy, 2007];

Existing approaches and implementations

Main approaches to produce word embeddings

1. Point-wise mutual information (**PMI**) association matrices, factorized by **SVD** (so called *count-based models*) [Bullinaria and Levy, 2007];
2. *Predictive models* using **artificial neural networks**, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (**word2vec**):

Existing approaches and implementations

Main approaches to produce word embeddings

1. Point-wise mutual information (**PMI**) association matrices, factorized by **SVD** (so called *count-based models*) [Bullinaria and Levy, 2007];
2. *Predictive models* using **artificial neural networks**, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (**word2vec**):
 - ▶ Continuous Bag-of-Words (**CBOW**),
 - ▶ Continuous Skip-Gram (**skipgram**);

Existing approaches and implementations

Main approaches to produce word embeddings

1. Point-wise mutual information (**PMI**) association matrices, factorized by **SVD** (so called *count-based models*) [Bullinaria and Levy, 2007];
2. *Predictive models* using **artificial neural networks**, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (**word2vec**):
 - ▶ Continuous Bag-of-Words (**CBOW**),
 - ▶ Continuous Skip-Gram (**skipgram**);
3. Global Vectors for Word Representation (**GloVe**) [Pennington et al., 2014];
4. ...etc

Two last approaches became super popular in the recent years and boosted almost all areas of natural language processing.

Existing approaches and implementations

Main approaches to produce word embeddings

1. Point-wise mutual information (**PMI**) association matrices, factorized by **SVD** (so called *count-based models*) [Bullinaria and Levy, 2007];
2. *Predictive models* using **artificial neural networks**, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (**word2vec**):
 - ▶ Continuous Bag-of-Words (**CBOW**),
 - ▶ Continuous Skip-Gram (**skipgram**);
3. Global Vectors for Word Representation (**GloVe**) [Pennington et al., 2014];
4. ...etc

Two last approaches became super popular in the recent years and boosted almost all areas of natural language processing.

Their principal difference from previous methods is that they actively employ **machine learning**.

Existing approaches and implementations

First, each word in the vocabulary receives a **random initial vector** of a pre-defined size. What happens next?

Existing approaches and implementations

First, each word in the vocabulary receives a **random initial vector** of a pre-defined size. What happens next?

...an efficient method for learning high quality distributed vector ...

The diagram shows the sentence "...an efficient method for learning high quality distributed vector ...". A blue arrow points to the word "learning", which is labeled "focus word". Two green brackets are drawn under the words "method" and "distributed", both labeled "context".

Learning good vectors

During training, we move through the training corpus with a **sliding window**.

Existing approaches and implementations

First, each word in the vocabulary receives a **random initial vector** of a pre-defined size. What happens next?

...an efficient method for learning high quality distributed vector ...

Diagram illustrating the concept of learning a distributed vector. The text "...an efficient method for learning high quality distributed vector ..." is shown with green brackets underneath. The left bracket is labeled "context" and the right bracket is labeled "context". A blue arrow points upwards from the word "learning" to the word "vector".

Learning good vectors

During training, we move through the training corpus with a **sliding window**.

Each instance (word in running text) is a prediction problem: the objective is to **predict the current word with the help of its contexts** (or vice versa).

Existing approaches and implementations

First, each word in the vocabulary receives a **random initial vector** of a pre-defined size. What happens next?

...an efficient method for learning high quality distributed vector ...

The diagram shows the text "...an efficient method for learning high quality distributed vector ...". Two green brackets are drawn under the text: one under "an efficient method for learning" and another under "high quality distributed vector". A blue arrow points upwards from the word "learning" to the word "vector".

Learning good vectors

During training, we move through the training corpus with a **sliding window**.

Each instance (word in running text) is a prediction problem: the objective is to **predict the current word with the help of its contexts** (or vice versa).

The outcome of the prediction determines whether we **adjust** the current word vector and in what direction.

Existing approaches and implementations

First, each word in the vocabulary receives a **random initial vector** of a pre-defined size. What happens next?

...an efficient method for learning high quality distributed vector ...

context focus word context

Learning good vectors

During training, we move through the training corpus with a **sliding window**.

Each instance (word in running text) is a prediction problem: the objective is to **predict the current word with the help of its contexts** (or vice versa).

The outcome of the prediction determines whether we **adjust** the current word vector and in what direction.

Gradually, vectors **converge** to (hopefully) optimal values.

Existing approaches and implementations

- ▶ **Continuous Bag-of-words** (CBOW) and **Continuous Skip-gram** (skip-gram) are conceptually similar but differ in important details;

Existing approaches and implementations

- ▶ **Continuous Bag-of-words** (CBOW) and **Continuous Skip-gram** (skip-gram) are conceptually similar but differ in important details;
- ▶ Shown to outperform traditional count DSMs in various semantic tasks for English (Baroni et al. 2014).

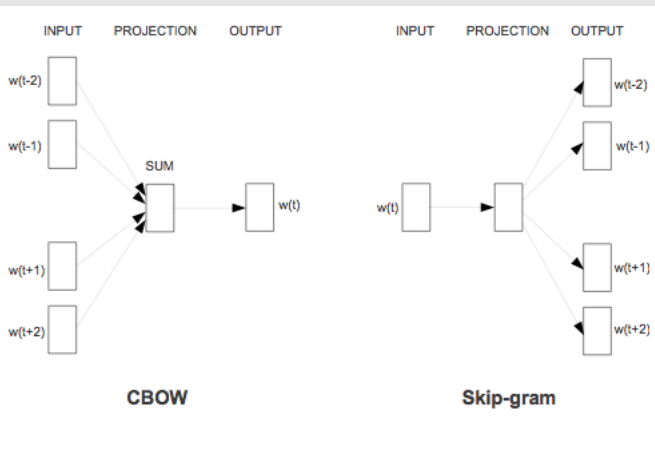
Existing approaches and implementations

- ▶ **Continuous Bag-of-words** (CBOW) and **Continuous Skip-gram** (skip-gram) are conceptually similar but differ in important details;
- ▶ Shown to outperform traditional count DSMs in various semantic tasks for English (Baroni et al. 2014).

At training time, **CBOW** learns to predict current word based on its context, while **Skip-Gram** learns to predict context based on the current word.

Existing approaches and implementations

Continuous Bag-of-Words and Continuous Skip-Gram: two algorithms in *word2vec* paper



Existing approaches and implementations

It is clear that **none of these algorithms is actually deep learning**.
Neural network is very simple, with only one hidden layer.

Existing approaches and implementations

It is clear that **none of these algorithms is actually deep learning**. Neural network is very simple, with only one hidden layer. The training objective is to maximize the **probability of observing the correct output word(s) w_t given the context word(s) $cw_1 \dots cw_j$** , with regard to their current embeddings (sets of neural weights).

Existing approaches and implementations

It is clear that **none of these algorithms is actually deep learning**.

Neural network is very simple, with only one hidden layer.

The training objective is to maximize the **probability of observing the correct output word(s) w_t given the context word(s) $cw_1 \dots cw_j$** , with regard to their current embeddings (sets of neural weights).

Cost function C for CBOW is the negative log probability (cross-entropy) of the correct answer:

$$C = -\log(p(w_t | cw_1 \dots cw_j)) \quad (2)$$

Existing approaches and implementations

It is clear that **none of these algorithms is actually deep learning**. Neural network is very simple, with only one hidden layer. The training objective is to maximize the **probability of observing the correct output word(s) w_t given the context word(s) $cw_1 \dots cw_j$** , with regard to their current embeddings (sets of neural weights). **Cost function C** for CBOW is the negative log probability (cross-entropy) of the correct answer:

$$C = -\log(p(w_t | cw_1 \dots cw_j)) \quad (2)$$

or for SkipGram

$$C = -\sum_{i=1}^j \log(p(cw_i | w_t)) \quad (3)$$

Existing approaches and implementations

It is clear that **none of these algorithms is actually deep learning**.

Neural network is very simple, with only one hidden layer.

The training objective is to maximize the **probability of observing the correct output word(s) w_t given the context word(s) $cw_1 \dots cw_j$** , with regard to their current embeddings (sets of neural weights).

Cost function C for CBOW is the negative log probability (cross-entropy) of the correct answer:

$$C = -\log(p(w_t | cw_1 \dots cw_j)) \quad (2)$$

or for SkipGram

$$C = -\sum_{i=1}^j \log(p(cw_i | w_t)) \quad (3)$$

and the learning itself is implemented with **stochastic gradient descent** and (optionally) adaptive learning rate.

Existing approaches and implementations

Prediction for each training instance is basically:

- ▶ **CBOW**: **average vector for all context words**. We check whether the current word vector is the closest to it among all vocabulary words.
- ▶ **SkipGram**: **current word vector**. We check whether each of context words vector is the closest to it among all vocabulary words.

Existing approaches and implementations

Prediction for each training instance is basically:

- ▶ **CBOW**: **average vector for all context words**. We check whether the current word vector is the closest to it among all vocabulary words.
- ▶ **SkipGram**: **current word vector**. We check whether each of context words vector is the closest to it among all vocabulary words.

Reminder: this 'closeness' is calculated with the help of **cosine similarity**.

After the training, we have 2 weight matrices: of context vectors and of output vectors. As a rule, only **output vectors** are used in practical tasks.

Existing approaches and implementations

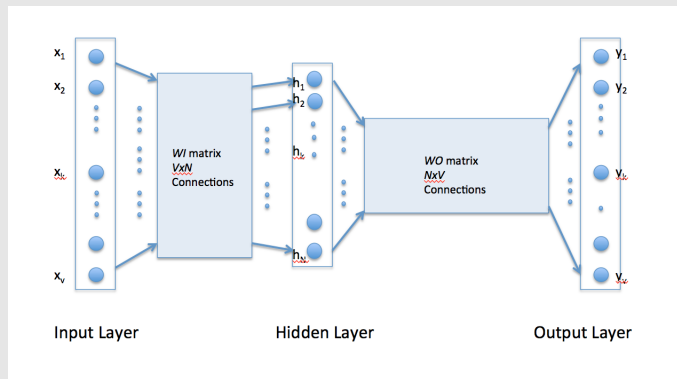
CBOW and SkipGram training algorithms

'the **vector** of a word **w** is “dragged” back-and-forth by the **vectors** of **w's** co-occurring words, as if there are physical strings between **w** and its neighbors...like gravity, or force-directed graph layout.' [Rong, 2014]

Existing approaches and implementations

CBOW and SkipGram training algorithms

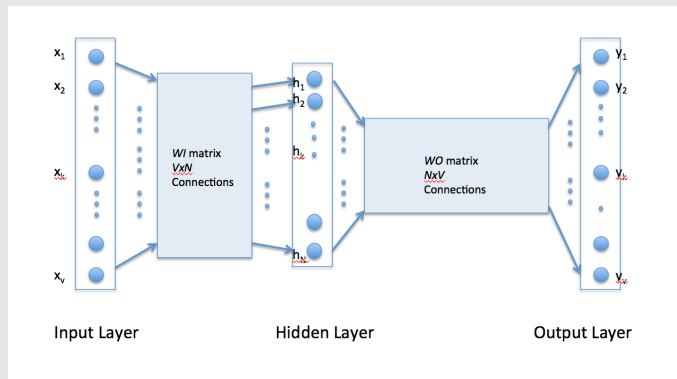
‘the **vector** of a word **w** is “dragged” back-and-forth by the **vectors** of **w**’s co-occurring words, as if there are physical strings between **w** and its neighbors...like gravity, or force-directed graph layout.’ [Rong, 2014]



Existing approaches and implementations

CBOW and SkipGram training algorithms

‘the **vector** of a word **w** is “dragged” back-and-forth by the **vectors** of **w**’s co-occurring words, as if there are physical strings between **w** and its neighbors...like gravity, or force-directed graph layout.’ [Rong, 2014]



Useful demo of *word2vec* algorithms: <https://ronxin.github.io/wevi/>

Existing approaches and implementations

Selection of learning material

At each training instance, to find out whether the prediction is true, we have to **iterate over all words in the vocabulary**.

Existing approaches and implementations

Selection of learning material

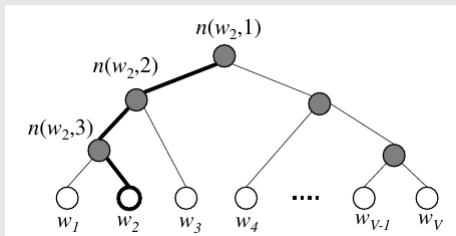
At each training instance, to find out whether the prediction is true, we have to **iterate over all words in the vocabulary**.

This is not feasible. That's why *word2vec* uses one of these two smart tricks:

1. **Hierarchical softmax**;
2. **Negative sampling**.

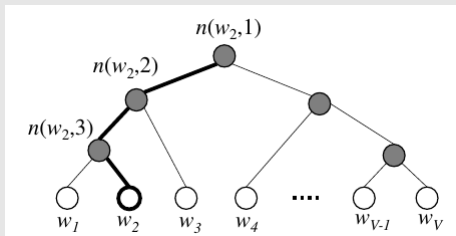
Existing approaches and implementations

Hierarchical softmax



Existing approaches and implementations

Hierarchical softmax



Calculate **joint probability of all items in the binary tree path** to the true word. This will be the probability of choosing the right word.

Now for vocabulary V , the complexity of each prediction is $O(\log(V))$ instead of $O(V)$.

Existing approaches and implementations

Negative sampling

The idea of **negative sampling** is even simpler:

Negative sampling

The idea of **negative sampling** is even simpler:

- ▶ do not iterate over all words in the vocabulary;

Negative sampling

The idea of **negative sampling** is even simpler:

- ▶ do not iterate over all words in the vocabulary;
- ▶ take your true word and sample **5...15 random 'noise' words** from the vocabulary;

Negative sampling

The idea of **negative sampling** is even simpler:

- ▶ do not iterate over all words in the vocabulary;
- ▶ take your true word and sample **5...15 random 'noise' words** from the vocabulary;
- ▶ these words serve as **negative examples**.

Negative sampling

The idea of **negative sampling** is even simpler:

- ▶ do not iterate over all words in the vocabulary;
- ▶ take your true word and sample **5...15 random 'noise' words** from the vocabulary;
- ▶ these words serve as **negative examples**.

Calculating probabilities for 15 words is of course much faster than iterating over all the vocabulary

Existing approaches and implementations

Things are complicated

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings
(**hyperparameters**):

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).
2. **Vector size**: how many distributed semantic features (dimensions) we use to describe a word. The more is not always the better.

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).
2. **Vector size**: how many distributed semantic features (dimensions) we use to describe a word. The more is not always the better.
3. **Window size**: context width and influence of distance. **Topical** (associative) or **functional** (semantic proper) models.

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).
2. **Vector size**: how many distributed semantic features (dimensions) we use to describe a word. The more is not always the better.
3. **Window size**: context width and influence of distance. **Topical** (associative) or **functional** (semantic proper) models.
4. **Frequency threshold**: useful to get rid of long noisy lexical tail;

Existing approaches and implementations

Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).
2. **Vector size**: how many distributed semantic features (dimensions) we use to describe a word. The more is not always the better.
3. **Window size**: context width and influence of distance. **Topical** (associative) or **functional** (semantic proper) models.
4. **Frequency threshold**: useful to get rid of long noisy lexical tail;
5. **Selection of learning material**: hierarchical softmax or negative sampling (used more often);

Existing approaches and implementations

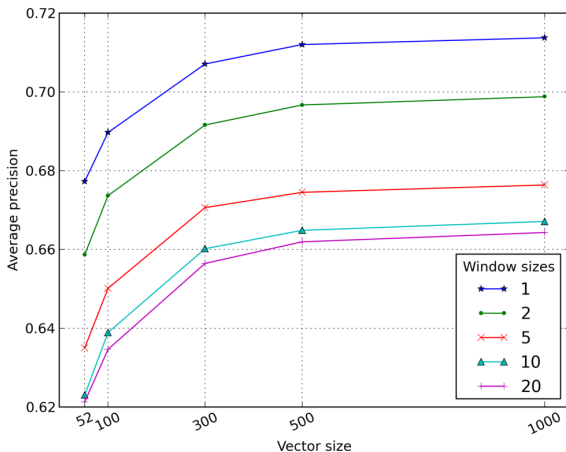
Things are complicated

Model performance hugely depends on training settings

(**hyperparameters**):

1. **CBOW** or **skip-gram** algorithm. Needs further research; SkipGram is generally better (but slower). CBOW seems to be better on small corpora (less than 100 mln tokens).
2. **Vector size**: how many distributed semantic features (dimensions) we use to describe a word. The more is not always the better.
3. **Window size**: context width and influence of distance. **Topical** (associative) or **functional** (semantic proper) models.
4. **Frequency threshold**: useful to get rid of long noisy lexical tail;
5. **Selection of learning material**: hierarchical softmax or negative sampling (used more often);
6. **Number of iterations** on our training data, etc...

Existing approaches and implementations



Model performance in **semantic relatedness** task depending on context width and vector size.

Existing approaches and implementations

How do we evaluate trained models?

Existing approaches and implementations

How do we evaluate trained models?

- ▶ **Semantic relatedness** (what is the association degree?):
 - ▶ Rubenstein and Goodenough dataset (1965)
 - ▶ WordSim 353 dataset (2002)
 - ▶ MEN dataset (2013)
 - ▶ SimLex-999 dataset (includes Russian since 2015)

Existing approaches and implementations

How do we evaluate trained models?

- ▶ **Semantic relatedness** (what is the association degree?):
 - ▶ Rubenstein and Goodenough dataset (1965)
 - ▶ WordSim 353 dataset (2002)
 - ▶ MEN dataset (2013)
 - ▶ SimLex-999 dataset (includes Russian since 2015)
- ▶ **Synonym detection** (what is most similar?):
 - ▶ TOEFL dataset (1997)

Existing approaches and implementations

How do we evaluate trained models?

- ▶ **Semantic relatedness** (what is the association degree?):
 - ▶ Rubenstein and Goodenough dataset (1965)
 - ▶ WordSim 353 dataset (2002)
 - ▶ MEN dataset (2013)
 - ▶ SimLex-999 dataset (includes Russian since 2015)
- ▶ **Synonym detection** (what is most similar?):
 - ▶ TOEFL dataset (1997)
- ▶ **Concept categorization** (what groups with what?):
 - ▶ ESSLI 2008 dataset
 - ▶ Battig dataset (2010)

Existing approaches and implementations

How do we evaluate trained models?

- ▶ **Semantic relatedness** (what is the association degree?):
 - ▶ Rubenstein and Goodenough dataset (1965)
 - ▶ WordSim 353 dataset (2002)
 - ▶ MEN dataset (2013)
 - ▶ SimLex-999 dataset (includes Russian since 2015)
- ▶ **Synonym detection** (what is most similar?):
 - ▶ TOEFL dataset (1997)
- ▶ **Concept categorization** (what groups with what?):
 - ▶ ESSLI 2008 dataset
 - ▶ Battig dataset (2010)
- ▶ **Analogical inference** (A is to B as C is to ?):
 - ▶ Google Analogy dataset (2013)
 - ▶ Many domain-specific datasets inspired by Google Analogy

Existing approaches and implementations

How do we evaluate trained models?

- ▶ **Semantic relatedness** (what is the association degree?):
 - ▶ Rubenstein and Goodenough dataset (1965)
 - ▶ WordSim 353 dataset (2002)
 - ▶ MEN dataset (2013)
 - ▶ SimLex-999 dataset (includes Russian since 2015)
- ▶ **Synonym detection** (what is most similar?):
 - ▶ TOEFL dataset (1997)
- ▶ **Concept categorization** (what groups with what?):
 - ▶ ESSLI 2008 dataset
 - ▶ Battig dataset (2010)
- ▶ **Analogical inference** (A is to B as C is to ?):
 - ▶ Google Analogy dataset (2013)
 - ▶ Many domain-specific datasets inspired by Google Analogy
- ▶ **Correlation with manually crafted linguistic features**:
 - ▶ QVEC (2015)

Subject to many discussions! The topic of a special workshop at ACL2016:

<https://sites.google.com/site/repevalacl16/>

Main frameworks and toolkits

1. *Dissect* (<http://clic.cimec.unitn.it/composes/toolkit/>);

Main frameworks and toolkits

1. *Dissect* (<http://clic.cimec.unitn.it/composes/toolkit/>);
2. **word2vec** original C code
(<https://word2vec.googlecode.com/svn/trunk/>)

Main frameworks and toolkits

1. *Dissect* (<http://clic.cimec.unitn.it/composes/toolkit/>);
2. **word2vec** original C code
(<https://word2vec.googlecode.com/svn/trunk/>)
3. *Gensim* framework for Python, including **word2vec** implementations
(<http://radimrehurek.com/gensim/>);

Existing approaches and implementations

Main frameworks and toolkits

1. *Dissect* (<http://clic.cimec.unitn.it/composes/toolkit/>);
2. **word2vec** original C code
(<https://word2vec.googlecode.com/svn/trunk/>)
3. *Gensim* framework for Python, including **word2vec** implementations
(<http://radimrehurek.com/gensim/>);
4. **word2vec** implementations in *Google's TensorFlow*
(<https://www.tensorflow.org/tutorials/word2vec/>);

Existing approaches and implementations

Main frameworks and toolkits

1. *Dissect* (<http://clic.cimec.unitn.it/composes/toolkit/>);
2. **word2vec** original C code
(<https://word2vec.googlecode.com/svn/trunk/>)
3. *Gensim* framework for Python, including **word2vec** implementations
(<http://radimrehurek.com/gensim/>);
4. **word2vec** implementations in *Google's TensorFlow*
(<https://www.tensorflow.org/tutorials/word2vec/>);
5. **GloVe** reference implementation
(<http://nlp.stanford.edu/projects/glove/>).

A bunch of observations

- ▶ **Wikipedia** is not the best training corpus: fluctuates wildly depending on hyperparameters. Perhaps, too specific language.

Existing approaches and implementations

A bunch of observations

- ▶ **Wikipedia** is not the best training corpus: fluctuates wildly depending on hyperparameters. Perhaps, too specific language.
- ▶ Normalize you data: **lowercase**, **lemmatize**, merge **multi-word entities**.

Existing approaches and implementations

A bunch of observations

- ▶ **Wikipedia** is not the best training corpus: fluctuates wildly depending on hyperparameters. Perhaps, too specific language.
- ▶ Normalize your data: **lowercase**, **lemmatize**, merge **multi-word entities**.
- ▶ It helps to **augment words with PoS tags** before training ('стать_ N'). As a result, your model becomes aware of morphological ambiguity.

Existing approaches and implementations

A bunch of observations

- ▶ **Wikipedia** is not the best training corpus: fluctuates wildly depending on hyperparameters. Perhaps, too specific language.
- ▶ Normalize your data: **lowercase**, **lemmatize**, merge **multi-word entities**.
- ▶ It helps to **augment words with PoS tags** before training ('СТАТЬ_ N'). As a result, your model becomes aware of morphological ambiguity.
- ▶ Remove your **stop words** yourself. Statistical downsampling implemented in *word2vec* algorithms can easily deprive you of valuable text data.

Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations
- 5 Model formats**
- 6 What can we find in the models?
- 7 More than words: representing texts

Model formats

Models can come in several formats:

1. Simple **text format**: words and sequences of values representing their vectors, one word per line; first line gives information on the number of words in the model and vector size.

Model formats

Models can come in several formats:

1. Simple **text format**: words and sequences of values representing their vectors, one word per line; first line gives information on the number of words in the model and vector size.
2. The same in the **binary form**.

Model formats

Models can come in several formats:

1. Simple **text format**: words and sequences of values representing their vectors, one word per line; first line gives information on the number of words in the model and vector size.
2. The same in the **binary form**.
3. **Gensim binary format**: uses *NumPy* matrices saved via Python pickles; stores a lot of additional information (input vectors, training algorithm, word frequency, etc).

Gensim works with all of these formats.

Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations
- 5 Model formats
- 6 What can we find in the models?**
- 7 More than words: representing texts

What can we find in the models?

- ▶ **Distributional** models are based on word co-occurrences in large training corpora;

What can we find in the models?

- ▶ **Distributional** models are based on word co-occurrences in large training corpora;
- ▶ they represent words as dense lexical vectors (**embeddings**);

What can we find in the models?

- ▶ **Distributional** models are based on word co-occurrences in large training corpora;
- ▶ they represent words as dense lexical vectors (**embeddings**);
- ▶ the models are also **distributed**: each word is represented as **multiple activations** (not a one-hot vector);

What can we find in the models?

- ▶ **Distributional** models are based on word co-occurrences in large training corpora;
- ▶ they represent words as dense lexical vectors (**embeddings**);
- ▶ the models are also **distributed**: each word is represented as **multiple activations** (not a one-hot vector);
- ▶ words occurring in similar contexts have **similar vectors**;

What can we find in the models?

- ▶ **Distributional** models are based on word co-occurrences in large training corpora;
- ▶ they represent words as dense lexical vectors (**embeddings**);
- ▶ the models are also **distributed**: each word is represented as **multiple activations** (not a one-hot vector);
- ▶ words occurring in similar contexts have **similar vectors**;
- ▶ one can find nearest **semantic associates** of a given word by calculating **cosine similarity** between vectors.

What can we find in the models?

Nearest semantic associates

вектор:

1. параметр 0.433
2. диполь 0.423
3. переменная 0.423
4. координата 0.413
5. плоскость 0.410
6. направление 0.404
7. ...

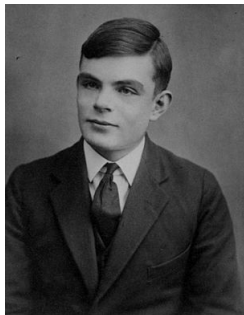
(From a model trained on the Russian National Corpus)

What can we find in the models?

Works with multi-word entities as well

What can we find in the models?

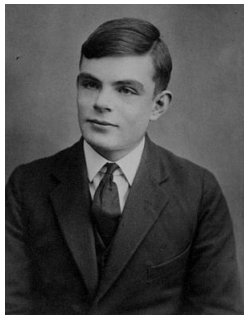
Works with multi-word entities as well



Alan_Turing (*from a model trained on Google News corpus (2013)*):

What can we find in the models?

Works with multi-word entities as well

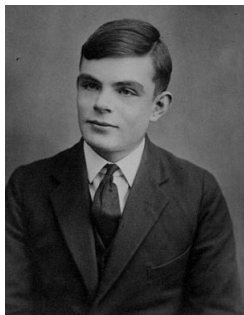


Alan_Turing (*from a model trained on Google News corpus (2013)*):

1. *Turing* 0.68

What can we find in the models?

Works with multi-word entities as well

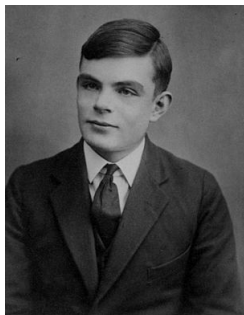


Alan_Turing (from a model trained on Google News corpus (2013)):

1. *Turing* 0.68
2. *Charles_Babbage* 0.65

What can we find in the models?

Works with multi-word entities as well

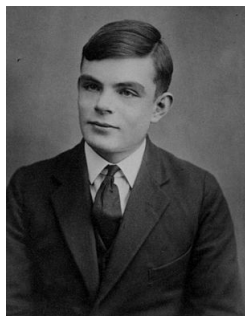


Alan_Turing (from a model trained on Google News corpus (2013)):

1. *Turing* 0.68
2. *Charles_Babbage* 0.65
3. *mathematician_Alan_Turing* 0.62

What can we find in the models?

Works with multi-word entities as well

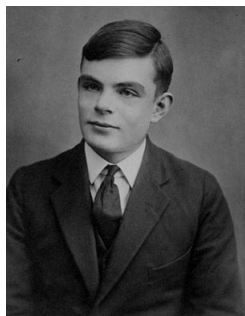


Alan_Turing (from a model trained on Google News corpus (2013)):

1. *Turing* 0.68
2. *Charles_Babbage* 0.65
3. *mathematician_Alan_Turing* 0.62
4. *pioneer_Alan_Turing* 0.60

What can we find in the models?

Works with multi-word entities as well

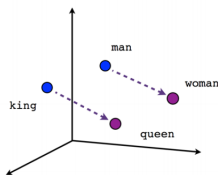


Alan_Turing (from a model trained on Google News corpus (2013)):

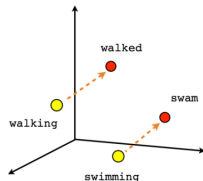
1. *Turing* 0.68
2. *Charles_Babbage* 0.65
3. *mathematician_Alan_Turing* 0.62
4. *pioneer_Alan_Turing* 0.60
5. *On_Computable_Numbers* 0.60
6. ...

What can we find in the models?

One can apply simple **algebraic operations** to word vectors (addition, subtraction, finding average vector for a group of words). They reflect semantic relationships between words.



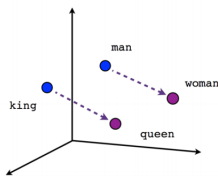
Male-Female



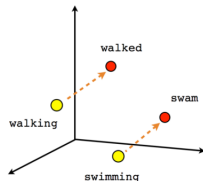
Verb tense

What can we find in the models?

One can apply simple **algebraic operations** to word vectors (addition, subtraction, finding average vector for a group of words). They reflect semantic relationships between words.



Male-Female

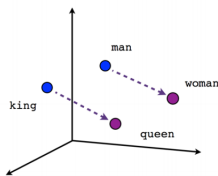


Verb tense

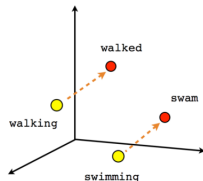
самолёт is to крыло as машина is to ?

What can we find in the models?

One can apply simple **algebraic operations** to word vectors (addition, subtraction, finding average vector for a group of words). They reflect semantic relationships between words.



Male-Female

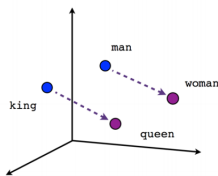


Verb tense

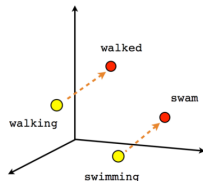
самолёт is to крыло as машина is to ? (колесо)

What can we find in the models?

One can apply simple **algebraic operations** to word vectors (addition, subtraction, finding average vector for a group of words). They reflect semantic relationships between words.



Male-Female



Verb tense

самолёт is to крыло as машина is to ? (колесо)

This paves way for many sense-related applications.

What can we find in the models?



googlenews model

1. [jehadis](#) 0.53280
2. [Naxalites](#) 0.52525
3. [Kashmiri_militant](#)
0.52517
4. [LeT](#) 0.51489
5. [Lashkar_e_Tayyaba](#)
0.51067

What can we find in the models?



1. Muslim_Brotherhood
0.56775
2. Egyptians 0.56694
3. Mubarak 0.56404
4. Hamas 0.55456
5. Egyptian 0.53355

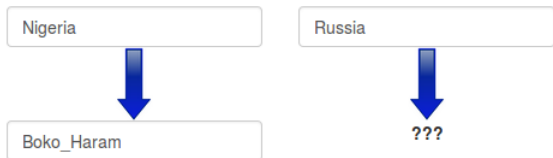
What can we find in the models?



googlenews model

1. [Kremlin](#) 0.57884
2. [Basayev](#) 0.55851
3. [Moscow](#) 0.55125
4. [Chechen_separatist_rebels](#)
0.52799
5. [Chechen_rebel](#)
0.52108

What can we find in the models?



googlenews model

1. [Kremlin](#) 0.57884
2. [Basayev](#) 0.55851
3. [Moscow](#) 0.55125
4. [Chechen_separatist_rebels](#) 0.52799
5. [Chechen_rebel](#) 0.52108

Try yourself at

<http://ltr.uio.no/semvec/calculator>,

<http://ling.go.mail.ru/dsm/calculator>

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;
- ▶ Using embeddings to help other NLP tasks: e.g., **parsing**;

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;
- ▶ Using embeddings to help other NLP tasks: e.g., **parsing**;
- ▶ Finding better ways to **evaluate** models;

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;
- ▶ Using embeddings to help other NLP tasks: e.g., **parsing**;
- ▶ Finding better ways to **evaluate** models;
- ▶ Learning **bilingual** or **multilingual** word embeddings;

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;
- ▶ Using embeddings to help other NLP tasks: e.g., **parsing**;
- ▶ Finding better ways to **evaluate** models;
- ▶ Learning **bilingual** or **multilingual** word embeddings;
- ▶ **Diachronic word embeddings**: discovering semantic shifts happening over time;

What can we find in the models?

Hot topics in word embeddings now

(based on impressions from this year's ACL conference)

- ▶ Going beyond just synonyms and similarity: detecting **hyponyms**, **hypernyms** and **antonyms**;
- ▶ Using embeddings to help other NLP tasks: e.g., **parsing**;
- ▶ Finding better ways to **evaluate** models;
- ▶ Learning **bilingual** or **multilingual** word embeddings;
- ▶ **Diachronic word embeddings**: discovering semantic shifts happening over time;
- ▶ ...and of course going beyond words: **sentence embeddings**.

Contents

- 1 Our motivation
- 2 TL:DR
- 3 Distributional hypothesis and word embeddings
- 4 Existing approaches and implementations
- 5 Model formats
- 6 What can we find in the models?
- 7 More than words: representing texts**

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.
- ▶ Can they help in detecting **similar texts**?

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.
- ▶ Can they help in detecting **similar texts**?
- ▶ Yes they can!

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.
- ▶ Can they help in detecting **similar texts**?
- ▶ Yes they can!
- ▶ Basically, we just need a way to combine **word vectors** into **phrase/sentence/document vectors**.

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.
- ▶ Can they help in detecting **similar texts**?
- ▶ Yes they can!
- ▶ Basically, we just need a way to combine **word vectors** into **phrase/sentence/document vectors**.
- ▶ For example, they can be 300-dimensional (as in the models).

More than words: representing texts

- ▶ Word embeddings allow to trace semantic similarity **at word level**.
- ▶ Can they help in detecting **similar texts**?
- ▶ Yes they can!
- ▶ Basically, we just need a way to combine **word vectors** into **phrase/sentence/document vectors**.
- ▶ For example, they can be 300-dimensional (as in the models).
- ▶ After the documents are represented as vectors, **classification/clustering becomes trivial**.

More than words: representing texts

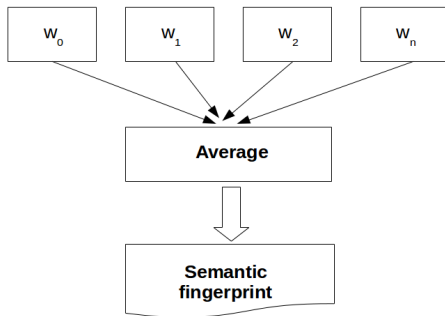
Simple but efficient: semantic fingerprints

- ▶ Whole document can be represented as average vector \vec{S} over vectors of all words $w_0 \dots n$ in it: 'semantic fingerprints'.

More than words: representing texts

Simple but efficient: semantic fingerprints

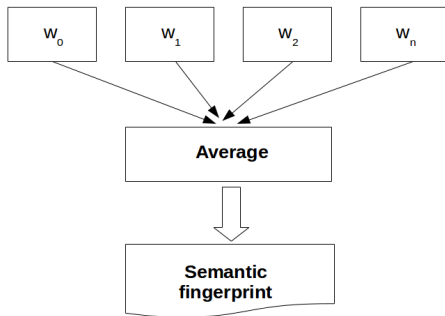
- ▶ Whole document can be represented as average vector \vec{S} over vectors of all words $w_0 \dots n$ in it: **'semantic fingerprints'**.



More than words: representing texts

Simple but efficient: semantic fingerprints

- ▶ Whole document can be represented as average vector \vec{S} over vectors of all words $w_0 \dots n$ in it: **'semantic fingerprints'**.



$$\vec{S} = \frac{1}{n} * \sum_{i=0}^n \vec{w}_n \quad (4)$$

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;
- ▶ With **semantic fingerprints** approach, we take advantage of 'semantic features' learned during the model training;

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;
- ▶ With **semantic fingerprints** approach, we take advantage of 'semantic features' learned during the model training;
- ▶ **Topically connected words collectively increase or decrease expression of the corresponding semantic components;**

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;
- ▶ With **semantic fingerprints** approach, we take advantage of 'semantic features' learned during the model training;
- ▶ **Topically connected words collectively increase or decrease expression of the corresponding semantic components;**
- ▶ Thus, **topical words automatically become more important than noise words;**
- ▶ **Semantic fingerprints** are precise enough to reveal topical differences between documents in corpora;

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;
- ▶ With **semantic fingerprints** approach, we take advantage of 'semantic features' learned during the model training;
- ▶ **Topically connected words collectively increase or decrease expression of the corresponding semantic components**;
- ▶ Thus, **topical words automatically become more important than noise words**;
- ▶ **Semantic fingerprints** are precise enough to reveal topical differences between documents in corpora;
- ▶ **Semantic fingerprints** work fast and can reuse already trained models.

More than words: representing texts

Why semantic fingerprints are so cool?

- ▶ **Generalized representations** do not depend on particular words;
- ▶ With **semantic fingerprints** approach, we take advantage of 'semantic features' learned during the model training;
- ▶ **Topically connected words collectively increase or decrease expression of the corresponding semantic components**;
- ▶ Thus, **topical words automatically become more important than noise words**;
- ▶ **Semantic fingerprints** are precise enough to reveal topical differences between documents in corpora;
- ▶ **Semantic fingerprints** work fast and can reuse already trained models.

See more in [Kutuzov et al., 2016] (shameless plug :)).

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;
- ▶ the algorithm takes as an input sentences/texts tagged with (possibly unique) identifiers;

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;
- ▶ the algorithm takes as an input sentences/texts tagged with (possibly unique) identifiers;
- ▶ learns distributed representations for these multi-word entities, such that similar sentences have similar vectors, etc...;

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;
- ▶ the algorithm takes as an input sentences/texts tagged with (possibly unique) identifiers;
- ▶ learns distributed representations for these multi-word entities, such that similar sentences have similar vectors, etc...;
- ▶ implemented in *Gensim* under the name *doc2vec*;

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;
- ▶ the algorithm takes as an input sentences/texts tagged with (possibly unique) identifiers;
- ▶ learns distributed representations for these multi-word entities, such that similar sentences have similar vectors, etc...;
- ▶ implemented in *Gensim* under the name *doc2vec*;
- ▶ not much studied;

More than words: representing texts

We can also learn vectors for texts during training

- ▶ [Mikolov et al., 2013] proposed **Paragraph Vector**;
- ▶ the algorithm takes as an input sentences/texts tagged with (possibly unique) identifiers;
- ▶ learns distributed representations for these multi-word entities, such that similar sentences have similar vectors, etc...;
- ▶ implemented in *Gensim* under the name *doc2vec*;
- ▶ not much studied;
- ▶ very memory-hungry (but one can reduce the number of tags)!

More than words: representing texts

Other possibilities

- ▶ [Taddy, 2015] proposed **classifying texts by inverting distributional models:**

Other possibilities

- ▶ [Taddy, 2015] proposed **classifying texts by inverting distributional models**:
- ▶ employs **Bayes rule** to calculate the likelihood of a model given a sentence;

Other possibilities

- ▶ [Taddy, 2015] proposed **classifying texts by inverting distributional models**:
- ▶ employs **Bayes rule** to calculate the likelihood of a model given a sentence;
- ▶ for example, models trained on positive and negative reviews;

More than words: representing texts

Other possibilities

- ▶ [Taddy, 2015] proposed **classifying texts by inverting distributional models**:
- ▶ employs **Bayes rule** to calculate the likelihood of a model given a sentence;
- ▶ for example, models trained on positive and negative reviews;
- ▶ implemented in *Gensim* (for hierarchical softmax models only);

Other possibilities

- ▶ [Taddy, 2015] proposed **classifying texts by inverting distributional models**:
- ▶ employs **Bayes rule** to calculate the likelihood of a model given a sentence;
- ▶ for example, models trained on positive and negative reviews;
- ▶ implemented in *Gensim* (for hierarchical softmax models only);
- ▶ drawback: you need separate models for each of your classes.

More than words: representing texts




There can be many other ways to employ word embeddings in classification tasks!

More than words: representing texts




There can be many other ways to employ word embeddings in classification tasks!

We will try to work with the **semantic fingerprints** approach.




References I

-  Bengio, Y., Ducharme, R., and Vincent, P. (2003).
A neural probabilistic language model.
Journal of Machine Learning Research, 3:1137–1155.
-  Bullinaria, J. A. and Levy, J. P. (2007).
Extracting semantic representations from word co-occurrence
statistics: A computational study.
Behavior research methods, 39(3):510–526.
-  Firth, J. (1957).
A synopsis of linguistic theory, 1930-1955.
Blackwell.

References II

-  Kutuzov, A., Kopotev, M., Ivanova, L., and Sviridenko, T. (2016). Clustering comparable corpora of Russian and Ukrainian academic texts: Word embeddings and semantic fingerprints. In *Proceedings of the Ninth Workshop on Building and Using Comparable Corpora, held at LREC-2016*, pages 3–10. European Language Resources Association.
-  Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems 26*.
-  Pennington, J., Socher, R., and Manning, C. D. (2014). GloVe: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543.

References III

-  Rong, X. (2014).
word2vec parameter learning explained.
arXiv preprint arXiv:1411.2738.
-  Taddy, M. (2015).
Document classification by inversion of distributed language
representations.
In *Proceedings of the 53rd Annual Meeting of the Association for
Computational Linguistics and the 7th International Joint
Conference on Natural Language Processing (Volume 2: Short
Papers)*, pages 45–49, Beijing, China.
-  Van der Maaten, L. and Hinton, G. (2008).
Visualizing data using t-SNE.
Journal of Machine Learning Research, 9(2579-2605):85.



Thank you for your attention!
Questions are welcome.

**Distributed Word Embeddings
in Text Classification**

Andrey Kutuzov (andreku@ifi.uio.no)
Language Technology Group
University of Oslo

*ISMW-FRUCT school
Saint-Petersburg, Russia*