Word2vec and others buzzwords: unsupervised machine learning approach to distributional semantics

Andrey Kutuzov

Language Technology Group, University of Oslo

November 13, 2015
AINL Conference, Saint-Petersburg

Table of Contents

Distributional semantics: how to model meaning?

Tiers of linguistic analysis

Distributional semantics: how to model meaning?

Tiers of linguistic analysis

Computational linguistics can comparatively easy model lower tiers of language:

- graphematics

Distributional semantics: how to model meaning?

Tiers of linguistic analysis

Computational linguistics can comparatively easy model lower tiers of language:

- graphematics
- phonetics

Distributional semantics: how to model meaning?

Tiers of linguistic analysis

Computational linguistics can comparatively easy model lower tiers of language:

- graphematics
- phonetics
- morphology

Distributional semantics: how to model meaning?

Tiers of linguistic analysis

Computational linguistics can comparatively easy model lower tiers of language:

- graphematics
- phonetics
- morphology
- syntax

Distributional semantics: how to model meaning?

But how to represent meaning?

Distributional semantics: how to model meaning?

But how to represent meaning?

- Semantics is difficult to represent formally.

Distributional semantics: how to model meaning?

But how to represent meaning?

- Semantics is difficult to represent formally.
- It generally means to invent machine-readable word representations with the following constraint: words which are similar in their sense should possess mathematically similar representations.

Distributional semantics: how to model meaning?

But how to represent meaning?

- Semantics is difficult to represent formally.
- It generally means to invent machine-readable word representations with the following constraint: words which are similar in their sense should possess mathematically similar representations.
- *«Светильник»* must be similar to *«лампа»* but not to *«кипятильник»*, even though their surface form suggests the opposite.

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Distributional semantics: how to model meaning?

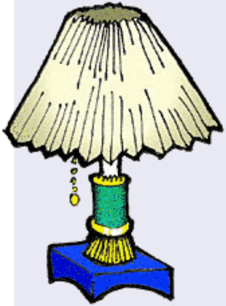
Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. *«Лампа»* concept can be expressed by any sequence of letters or sounds:

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. «*Лампа*» concept can be expressed by any sequence of letters or sounds:

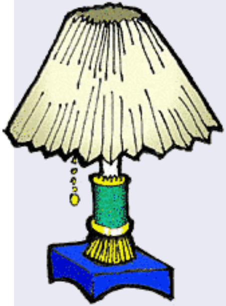


- lantern

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. *«Лампа»* concept can be expressed by any sequence of letters or sounds:

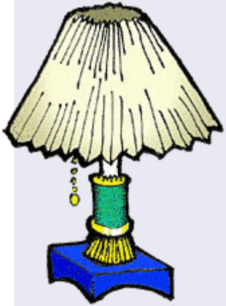


- lantern
- лампа

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. *«Лампа»* concept can be expressed by any sequence of letters or sounds:

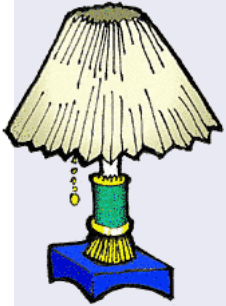


- lantern
- лампа
- lucerna

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. *«Лампа»* concept can be expressed by any sequence of letters or sounds:

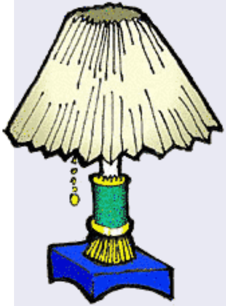


- lantern
- лампа
- lucerna
- гэрэл

Distributional semantics: how to model meaning?

Arbitrariness of a linguistic sign

Unlike many other signs, words do not possess a direct link between form and meaning. *«Лампа»* concept can be expressed by any sequence of letters or sounds:



- lantern
- лампа
- lucerna
- гэрэл
- ...

Distributional semantics: how to model meaning?

Possible data sources

Distributional semantics: how to model meaning?

Possible data sources

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

Distributional semantics: how to model meaning?

Possible data sources

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

- Building ontologies (knowledge-based approach). Top-down.

Distributional semantics: how to model meaning?

Possible data sources

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

- Building ontologies (knowledge-based approach). Top-down.
- Extracting semantics from usage patterns in text corpora (distributional approach). Bottom-up .

Distributional semantics: how to model meaning?

Possible data sources

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

- Building ontologies (knowledge-based approach). Top-down.
- Extracting semantics from usage patterns in text corpora (distributional approach). Bottom-up .

We are interested in the second approach: semantics can be derived from the contexts a given word takes.

Distributional semantics: how to model meaning?

Possible data sources

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

- Building ontologies (knowledge-based approach). Top-down.
- Extracting semantics from usage patterns in text corpora (distributional approach). Bottom-up .

We are interested in the second approach: semantics can be derived from the contexts a given word takes.

'You shall know a word by the company it keeps.' (Firth 1957)

Distributional semantics: how to model meaning?

Possible data sources

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

- Building ontologies (knowledge-based approach). Top-down.
- Extracting semantics from usage patterns in text corpora (distributional approach). Bottom-up .

We are interested in the second approach: semantics can be derived from the contexts a given word takes.

'You shall know a word by the company it keeps.' (Firth 1957)
Word meaning is typically defined by lexical co-occurrences in a large training corpus: distributional semantics models (DSMs).

Table of Contents

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

- Multi-dimensional semantic space.

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

- Multi-dimensional semantic space.
- Contexts are axes (dimensions) in this space.

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

- Multi-dimensional semantic space.
- Contexts are axes (dimensions) in this space.
- Words are vectors or points in this space.

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

- Multi-dimensional semantic space.
- Contexts are axes (dimensions) in this space.
- Words are vectors or points in this space.
- In case of lexical co-occurrences, words are both.

Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

- Multi-dimensional semantic space.
- Contexts are axes (dimensions) in this space.
- Words are vectors or points in this space.
- In case of lexical co-occurrences, words are both.
- With large corpora, we have tens of millions of dimensions (axes, words).

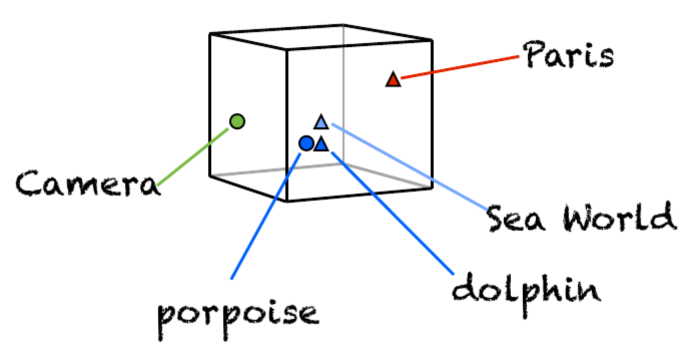
Traditional count-based DSMs

In count models, semantics of particular words is represented as vectors of real values denoting frequency of their co-occurrences with contexts.

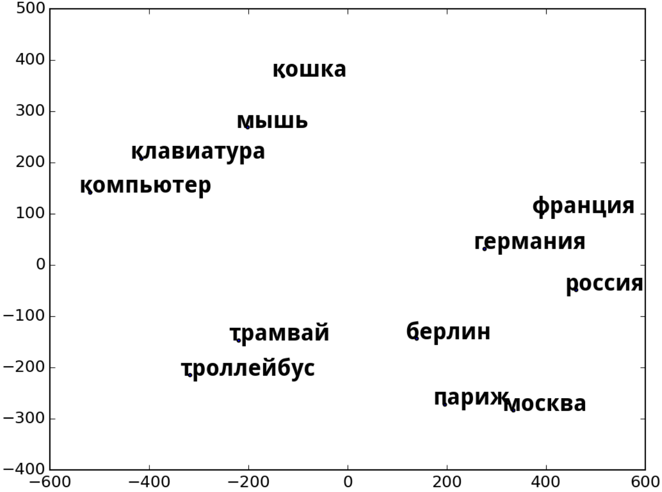
- Multi-dimensional semantic space.
- Contexts are axes (dimensions) in this space.
- Words are vectors or points in this space.
- In case of lexical co-occurrences, words are both.
- With large corpora, we have tens of millions of dimensions (axes, words).
- But the vectors are very sparse, most components are zero.

Traditional count-based DSMs

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



Traditional count-based DSMs



Traditional count-based DSMs

Semantic similarity between words is usually represented through cosine similarity of their corresponding vectors.

Traditional count-based DSMs

Semantic similarity between words is usually represented through cosine similarity of their corresponding vectors.

- Similarity lowers as angle between word vectors grows.

Traditional count-based DSMs

Semantic similarity between words is usually represented through cosine similarity of their corresponding vectors.

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

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

Traditional count-based DSMs

Of course one can somehow weight absolute frequency of co-occurrences to improve quality.

Traditional count-based DSMs

Of course one can somehow weight absolute frequency of co-occurrences to improve quality.
For example, Dice coefficient:

$$Dice(w,w') = \frac{2c(w,w')}{c(w) + c(w')} \tag{2}$$

where $c(w)$ – absolute frequency of w word,
$c(w')$ – absolute frequency of w' word
$c(w,w')$ – frequency of w and w' occurring together (collocation).

Traditional count-based DSMs

Of course one can somehow weight absolute frequency of co-occurrences to improve quality.
For example, Dice coefficient:

$$Dice(w,w') = \frac{2c(w,w')}{c(w) + c(w')} \tag{2}$$

where $c(w)$ – absolute frequency of w word,
$c(w')$ – absolute frequency of w' word
$c(w,w')$ – frequency of w and w' occurring together (collocation).
...or other weighting coefficients: log-likelihood, (positive) pointwise mutual information (PMI), etc.

Traditional count-based DSMs

Count-based models do have disadvantages:

Traditional count-based DSMs

Count-based models do have disadvantages:

- Curse of dimensionality: vector sizes are huge (generally equal to vocabulary size).

Traditional count-based DSMs

Count-based models do have disadvantages:

- Curse of dimensionality: vector sizes are huge (generally equal to vocabulary size).
- We do not know what part of our vectors is really useful, and what part is noise. They are derived from corpora 'as is'.

Traditional count-based DSMs

Count-based models do have disadvantages:

- Curse of dimensionality: vector sizes are huge (generally equal to vocabulary size).
- We do not know what part of our vectors is really useful, and what part is noise. They are derived from corpora 'as is'.
- Dimensionality reduction techniques like PCA or Singular Value Decomposition (SVD) partially address these issues...
- ...but make the whole thing even more computationally expensive and effectively forbid online training.

Traditional count-based DSMs

Count-based models do have disadvantages:

- Curse of dimensionality: vector sizes are huge (generally equal to vocabulary size).
- We do not know what part of our vectors is really useful, and what part is noise. They are derived from corpora 'as is'.
- Dimensionality reduction techniques like PCA or Singular Value Decomposition (SVD) partially address these issues...
- ...but make the whole thing even more computationally expensive and effectively forbid online training.

Are there any other way to get high-quality vectors?

Traditional count-based DSMs

Count-based models do have disadvantages:

- Curse of dimensionality: vector sizes are huge (generally equal to vocabulary size).
- We do not know what part of our vectors is really useful, and what part is noise. They are derived from corpora 'as is'.
- Dimensionality reduction techniques like PCA or Singular Value Decomposition (SVD) partially address these issues...
- ...but make the whole thing even more computationally expensive and effectively forbid online training.

Are there any other way to get high-quality vectors?
Learn them from the data!

Table of Contents

Predict models

- This is the approach employed by the so called predict models in distributional semantics.

Predict models

- This is the approach employed by the so called predict models in distributional semantics.
- With count models, we first calculate all words' co-occurrences with other words and treat these frequencies as vectors.

Predict models

- This is the approach employed by the so called predict models in distributional semantics.
- With count models, we first calculate all words' co-occurrences with other words and treat these frequencies as vectors.
- With predict models, we directly learn vectors which maximize similarity between contextual neighbors found in the data, while minimizing similarity for unseen contexts.

Predict models

- This is the approach employed by the so called predict models in distributional semantics.
- With count models, we first calculate all words' co-occurrences with other words and treat these frequencies as vectors.
- With predict models, we directly learn vectors which maximize similarity between contextual neighbors found in the data, while minimizing similarity for unseen contexts.
- Initial vectors are generated randomly and then gradually converge to (hopefully) optimal values, as we move through the training corpus with a sliding window.

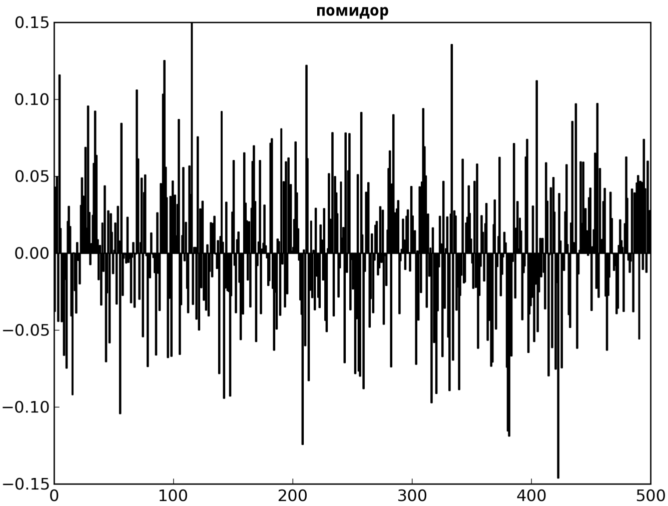
Predict models

- This is the approach employed by the so called predict models in distributional semantics.
- With count models, we first calculate all words' co-occurrences with other words and treat these frequencies as vectors.
- With predict models, we directly learn vectors which maximize similarity between contextual neighbors found in the data, while minimizing similarity for unseen contexts.
- Initial vectors are generated randomly and then gradually converge to (hopefully) optimal values, as we move through the training corpus with a sliding window.
- Target vector size is set at the beginning of training process and typically is hundreds of components.

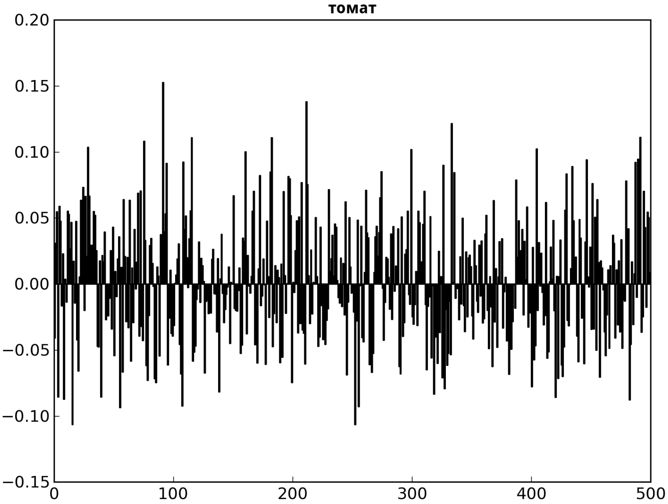
Predict models

- This is the approach employed by the so called predict models in distributional semantics.
- With count models, we first calculate all words' co-occurrences with other words and treat these frequencies as vectors.
- With predict models, we directly learn vectors which maximize similarity between contextual neighbors found in the data, while minimizing similarity for unseen contexts.
- Initial vectors are generated randomly and then gradually converge to (hopefully) optimal values, as we move through the training corpus with a sliding window.
- Target vector size is set at the beginning of training process and typically is hundreds of components.
- Thus, dense vectors (embeddings) are produced naturally, without additionall dimensionality reduction step.

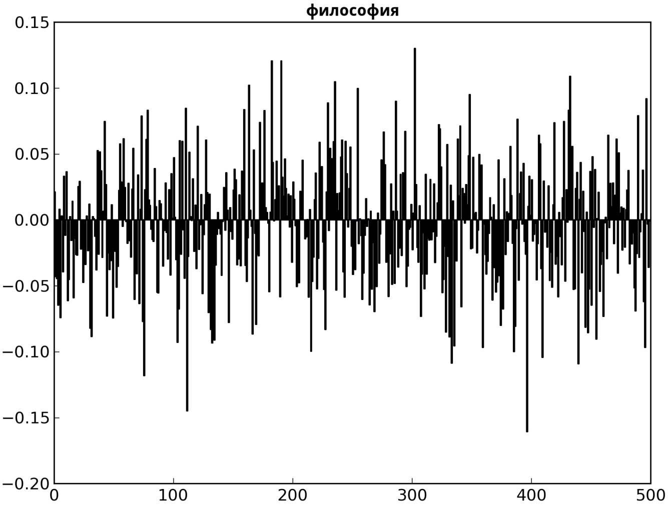
Predict models



Predict models



Predict models



Predict models

Brief recap

In count models (including PPMI matrices, LSA and others), vector size initially is equal to the size of our vocabulary (for example, one million)

Predict models

Brief recap

In count models (including PPMI matrices, LSA and others), vector size initially is equal to the size of our vocabulary (for example, one million)

While in predict models, vector size is arbitrarily set up before training (for example, about 500)

Predict models

Brief recap

In count models (including PPMI matrices, LSA and others), vector size initially is equal to the size of our vocabulary (for example, one million)

While in predict models, vector size is arbitrarily set up before training (for example, about 500)

In count models, vector components are absolute co-occurrence counts (may be, weighted and factorized).

Predict models

Brief recap

In count models (including PPMI matrices, LSA and others), vector size initially is equal to the size of our vocabulary (for example, one million)

While in predict models, vector size is arbitrarily set up before training (for example, about 500)

In count models, vector components are absolute co-occurrence counts (may be, weighted and factorized).

In predict models, vectors are first initialized randomly, and then gradually converge to optimum values, so that vectors for words with similar contexts are similar. We never actually count co-occurrence frequencies.

Online training is possible: we can update the model with new data any time!

Table of Contents

Going neural

- Detail of learning can differ from one algorithm to another;

Going neural

- Detail of learning can differ from one algorithm to another;
- Particularly, we can either simply change the current word vector with the vectors of its neighbors (as in Random Indexing)...

Going neural

- Detail of learning can differ from one algorithm to another;
- Particularly, we can either simply change the current word vector with the vectors of its neighbors (as in Random Indexing)...
- ...or we can employ machine learning and consider each training instance as a prediction problem: we want to predict current word with the help of its contexts (or vice versa).

Going neural

- Detail of learning can differ from one algorithm to another;
- Particularly, we can either simply change the current word vector with the vectors of its neighbors (as in Random Indexing)...
- ...or we can employ machine learning and consider each training instance as a prediction problem: we want to predict current word with the help of its contexts (or vice versa).
- The outcome of the prediction determines whether we change the current word vector and in what direction.

Going neural

- Detail of learning can differ from one algorithm to another;
- Particularly, we can either simply change the current word vector with the vectors of its neighbors (as in Random Indexing)...
- ...or we can employ machine learning and consider each training instance as a prediction problem: we want to predict current word with the help of its contexts (or vice versa).
- The outcome of the prediction determines whether we change the current word vector and in what direction.

That's where neural networks come into play.

Going neural

Imitating the brain

Going neural

Imitating the brain

- 10^{11} neurons in the brain, 10^4 links connected to each.

Going neural

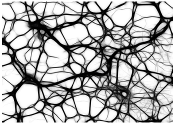
Imitating the brain

- 10^{11} neurons in the brain, 10^4 links connected to each.
- Neurons receive signals with different weights from other neurons.

Going neural

Imitating the brain

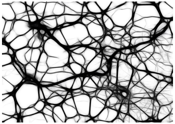
- 10^{11} neurons in the brain, 10^4 links connected to each.
- Neurons receive signals with different weights from other neurons.
- Then they produce output depending on signals received.



Going neural

Imitating the brain

- 10^{11} neurons in the brain, 10^4 links connected to each.
- Neurons receive signals with different weights from other neurons.
- Then they produce output depending on signals received.



Artificial neural networks attempt to imitate this process.

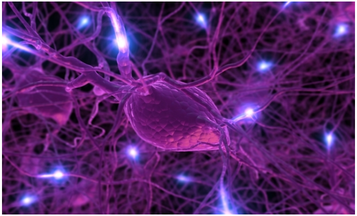
Going neural

There is evidence that concepts are stored in brain as neural activation patterns.

Going neural

There is evidence that concepts are stored in brain as neural activation patterns.

Very similar to vector representations! Meaning is a set of distributed 'semantic components' which can be more or less activated.



Each word is represented by a vector of n dimensions (aka neurons), and each neuron is responsible for many concepts or wide 'semantic components'.

Table of Contents

What about word2vec?

In 2013, Tomas Mikolov et al published a paper '*Efficient Estimation of Word Representations in Vector Space*', and released *word2vec* tool to train neural embeddings on large text corpora.

What about word2vec?

In 2013, Tomas Mikolov et al published a paper '*Efficient Estimation of Word Representations in Vector Space*', and released *word2vec* tool to train neural embeddings on large text corpora.

- http://arxiv.org/abs/1301.3781
- https://code.google.com/p/word2vec/

What about word2vec?

In 2013, Tomas Mikolov et al published a paper '*Efficient Estimation of Word Representations in Vector Space*', and released *word2vec* tool to train neural embeddings on large text corpora.

- `http://arxiv.org/abs/1301.3781`
- `https://code.google.com/p/word2vec/`

Mikolov smartly modified already existing algorithms: removed hidden layer from neural networks, used hierarchical softmax and negative sampling, found good combination of hyperparameters.
As a a result, *word2vec* learns word vectors orders of magnitude faster than earlier NNLMs, with comparable or better performance.

What about word2vec?

Now it is very easy to train *word2vec*-like models on large corpora and get meaningul vectors. One can produce lists of semantically 'similar' words:

What about word2vec?

Now it is very easy to train *word2vec*-like models on large corpora and get meaningul vectors. One can produce lists of semantically 'similar' words:

динозавр

1. *мамонт* 0.397899210453
2. *рептилия* 0.360172241926
3. *млекопитающее* 0.328677803278
4. *ящерица* 0.326320767403
5. *птеродактиль* 0.320571988821

What about word2vec?

Now it is very easy to train *word2vec*-like models on large corpora and get meaningul vectors. One can produce lists of semantically 'similar' words:

динозавр

1. *мамонт* 0.397899210453
2. *рептилия* 0.360172241926
3. *млекопитающее* 0.328677803278
4. *ящерица* 0.326320767403
5. *птеродактиль* 0.320571988821

Values after items, of course, represent cosine similarity between respective words' vectors and **'динозавр'** vector.

What about word2vec?

- *word2vec* in fact features two algorithms: Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram);

What about word2vec?

- *word2vec* in fact features two algorithms: Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram);
- Conceptually similar but differ in important details;

What about word2vec?

- *word2vec* in fact features two algorithms: Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram);
- Conceptually similar but differ in important details;
- Shown to outperform traditional count DSMs in various semantic tasks for English (Baroni et al. 2014).

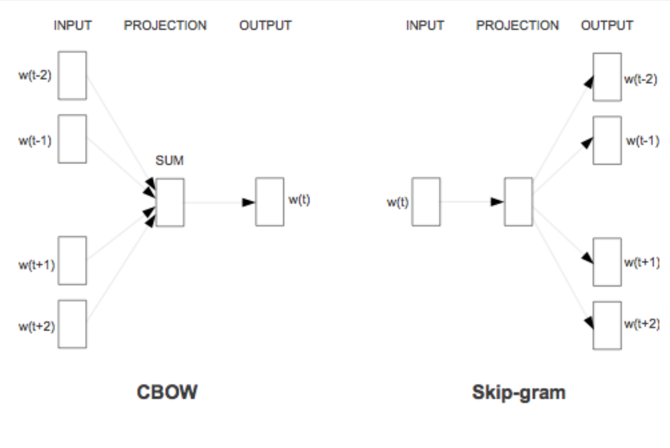
What about word2vec?

- *word2vec* in fact features two algorithms: Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram);
- 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.

What about word2vec?

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



What about word2vec?

It is clear that none of these algorithms is actually **deep** learning. Neural network is very simple: only input, projection and output layers.

What about word2vec?

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple: only input, projection and output layers.
The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 \ldots cw_j$, with regard to its current embedding (set of neural weights).

What about word2vec?

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

Neural network is very simple: only input, projection and output layers.

The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, with regard to its current embedding (set of neural weights).

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

$$C = -logp(w_t | cw_1 ... cw_j) \tag{3}$$

What about word2vec?

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple: only input, projection and output layers.
The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, with regard to its current embedding (set of neural weights).
Cost function C for CBOW is the negative log probability (cross-entropy) of the correct answer:

$$C = -logp(w_t | cw_1 ... cw_j) \tag{3}$$

or for SkipGram

$$C = -\sum_j logp(cw'_j | w_t) \tag{4}$$

What about word2vec?

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple: only input, projection and output layers.
The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, with regard to its current embedding (set of neural weights).
Cost function C for CBOW is the negative log probability (cross-entropy) of the correct answer:

$$C = -logp(w_t | cw_1 ... cw_j) \tag{3}$$

or for SkipGram

$$C = -\sum_j logp(cw_j' | w_t) \tag{4}$$

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

What about word2vec?

Prediction for each training instance is basically:

- CBOW: average vector for all context words. We check how close it is to the current word vector.
- SkipGram: current word vector. We check how close it is to each of the context words vector.

What about word2vec?

In the next two years after the original paper, there was a lot of follow-up research on this:

- Christopher Mannning and other folks at Stanford released Glove – a slightly different version of the same approach;

What about word2vec?

In the next two years after the original paper, there was a lot of follow-up research on this:

- Christopher Mannning and other folks at Stanford released Glove – a slightly different version of the same approach;
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients;

What about word2vec?

In the next two years after the original paper, there was a lot of follow-up research on this:

- Christopher Mannning and other folks at Stanford released Glove – a slightly different version of the same approach;
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients;
- The same people showed that much of amazing performance of SkipGram is due to choice of hyperparameters, but it is still very robust and computationally efficient;

What about word2vec?

In the next two years after the original paper, there was a lot of follow-up research on this:

- Christopher Mannning and other folks at Stanford released Glove – a slightly different version of the same approach;
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients;
- The same people showed that much of amazing performance of SkipGram is due to choice of hyperparameters, but it is still very robust and computationally efficient;
- Le and Mikolov proposed Paragraph Vector: an algorithm to learn such distributed representations not only for words but also for paragraphs or documents;

What about word2vec?

In the next two years after the original paper, there was a lot of follow-up research on this:

- Christopher Mannning and other folks at Stanford released Glove – a slightly different version of the same approach;
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients;
- The same people showed that much of amazing performance of SkipGram is due to choice of hyperparameters, but it is still very robust and computationally efficient;
- Le and Mikolov proposed Paragraph Vector: an algorithm to learn such distributed representations not only for words but also for paragraphs or documents;
- These approaches were implemented in open-source software, for example, Gensim framework for Python.

What about word2vec?

Things are complicated

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

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

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

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

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

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

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

1. CBOW or skip-gram algorithm. Needs further research; SkipGram is generally better (but slower). CBOW is better on small corpora (less than 100 mln tokens).
2. Vector size: how many distributed semantic features (dimensions) we use to describe a lemma. 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;

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

1. CBOW or skip-gram algorithm. Needs further research; SkipGram is generally better (but slower). CBOW is better on small corpora (less than 100 mln tokens).
2. Vector size: how many distributed semantic features (dimensions) we use to describe a lemma. 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;

What about word2vec?

Things are complicated

Model performance hugely depends on training settings:

1. CBOW or skip-gram algorithm. Needs further research; SkipGram is generally better (but slower). CBOW is better on small corpora (less than 100 mln tokens).
2. Vector size: how many distributed semantic features (dimensions) we use to describe a lemma. 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;
6. Number of iterations on our training data, etc...

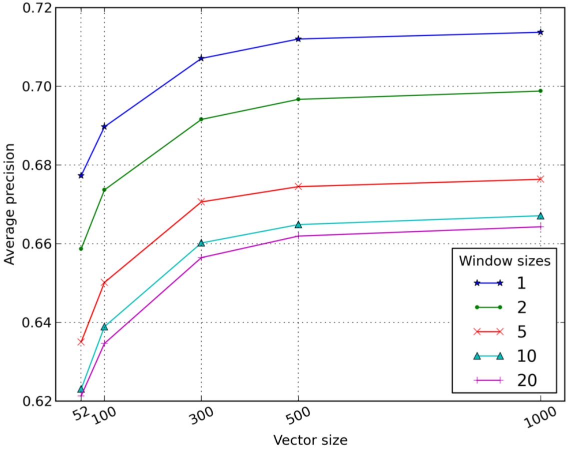
What about word2vec?

There is no silver bullet:
set of optimal hyperparameters is unique for each particular task.

What about word2vec?

There is no silver bullet:
set of optimal hyperparameters is unique for each particular task.
CBOW likes hierarchical softmax and SkipGram likes negative sampling (no less than 10 samples). Otherwise, performance drops drastically.

What about word2vec?



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

Table of Contents

Linguistic details

Performance also critically depends on the quality of training data: not only raw volume, but balanced coverage of the language.

- In our experiments with Russian, models trained on academic Russian National Corpus outperformed their competitors, often with vectors of lower dimensionality.
  Very impressive, considering it is much smaller in size (100m words versus corpora of billions of words).

Linguistic details

Performance also critically depends on the quality of training data: not only raw volume, but balanced coverage of the language.

- In our experiments with Russian, models trained on academic Russian National Corpus outperformed their competitors, often with vectors of lower dimensionality.
  Very impressive, considering it is much smaller in size (100m words versus corpora of billions of words).
- The corpus seems to be representative of the Russian language: balanced linguistic evidence for all major vocabulary tiers.

Linguistic details

Performance also critically depends on the quality of training data: not only raw volume, but balanced coverage of the language.

- In our experiments with Russian, models trained on academic Russian National Corpus outperformed their competitors, often with vectors of lower dimensionality.
  Very impressive, considering it is much smaller in size (100m words versus corpora of billions of words).
- The corpus seems to be representative of the Russian language: balanced linguistic evidence for all major vocabulary tiers.
- Little or no noise and junk fragments.

Linguistic details

Performance also critically depends on the quality of training data: not only raw volume, but balanced coverage of the language.

- In our experiments with Russian, models trained on academic Russian National Corpus outperformed their competitors, often with vectors of lower dimensionality.
  Very impressive, considering it is much smaller in size (100m words versus corpora of billions of words).
- The corpus seems to be representative of the Russian language: balanced linguistic evidence for all major vocabulary tiers.
- Little or no noise and junk fragments.

Linguistic details

Performance also critically depends on the quality of training data: not only raw volume, but balanced coverage of the language.

- In our experiments with Russian, models trained on academic Russian National Corpus outperformed their competitors, often with vectors of lower dimensionality.
  Very impressive, considering it is much smaller in size (100m words versus corpora of billions of words).
- The corpus seems to be representative of the Russian language: balanced linguistic evidence for all major vocabulary tiers.
- Little or no noise and junk fragments.

...rules!

Linguistic details

Our best-performing models submitted to RUSSE evaluation task (more about this tomorrow):

Linguistic details

Our best-performing models submitted to RUSSE evaluation task (more about this tomorrow):

Track	**hj**	**rt**	**ae**	**ae2**
Rank	2	5	5	4
Training settings	CBOW on **Ruscorpora** + CBOW on **Web**	CBOW on **Ruscorpora** + CBOW on **Web**	Skip-gram on **News**	CBOW on **Web**
Score	0.7187	0.8839	0.8995	0.9662

Linguistic details

Our best-performing models submitted to RUSSE evaluation task (more about this tomorrow):

Track	hj	rt	ae	ae2
Rank	2	5	5	4
Training settings	CBOW on **Ruscorpora** + CBOW on **Web**	CBOW on **Ruscorpora** + CBOW on **Web**	Skip-gram on **News**	CBOW on **Web**
Score	0.7187	0.8839	0.8995	0.9662

- **Russian National Corpus** is better in semantic relatedness tasks;
- larger **Web** and **News** corpora provide good training data for association tasks.

Linguistic details

A bunch of observations

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

Linguistic details

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.

Linguistic details

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.
- It helps to augment words with PoS tags before training (*'стать_N'*). As a result, your model can resolve morphological ambiguity.

Linguistic details

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.
- It helps to augment words with PoS tags before training (*'стать_N'*). As a result, your model can resolve morphological ambiguity.
- Remove your stop words yourself. Statistical downsampling implemented in *word2vec* algorithms can easily deprive you of valuable text data.

Table of Contents

What the model knows: inter-word relations

Models trained on large corpora possess an interesting property: algebraic operations on vectors reflect semantic relations between words.

What the model knows: inter-word relations

Models trained on large corpora possess an interesting property: algebraic operations on vectors reflect semantic relations between words.

Operations

If we subtract *France* vector from *Paris* vector and add *Germany* vector, we will get a vector for which the nearest one in the model will be *Berlin*.

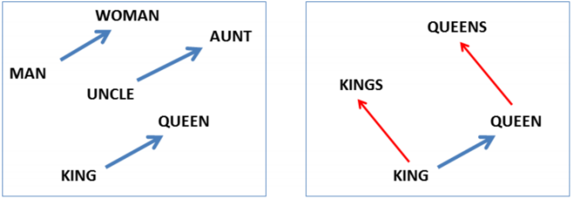
What the model knows: inter-word relations

Models trained on large corpora possess an interesting property: algebraic operations on vectors reflect semantic relations between words.

Operations

If we subtract *France* vector from *Paris* vector and add *Germany* vector, we will get a vector for which the nearest one in the model will be *Berlin*.

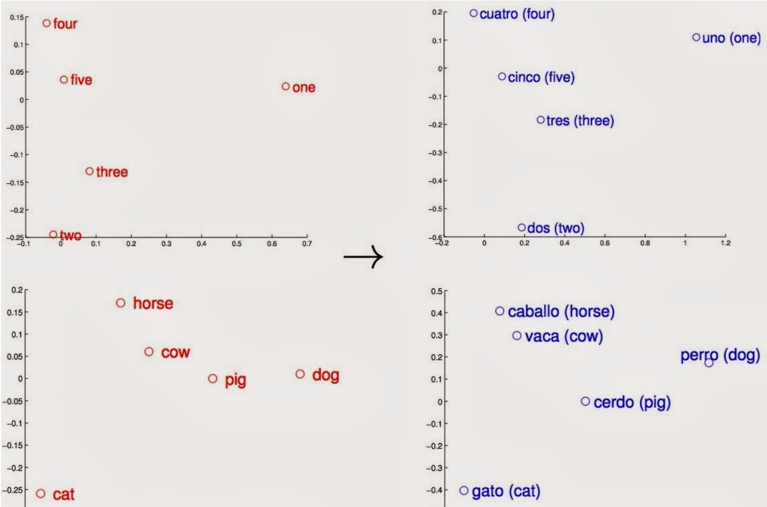
This paves way for many sense-related applications.



(Mikolov et al., NAACL HLT, 2013)

What the model knows: inter-word relations

Semantic structures are reproduced even in different languages (Mikolov 2013):



What the model knows: inter-word relations

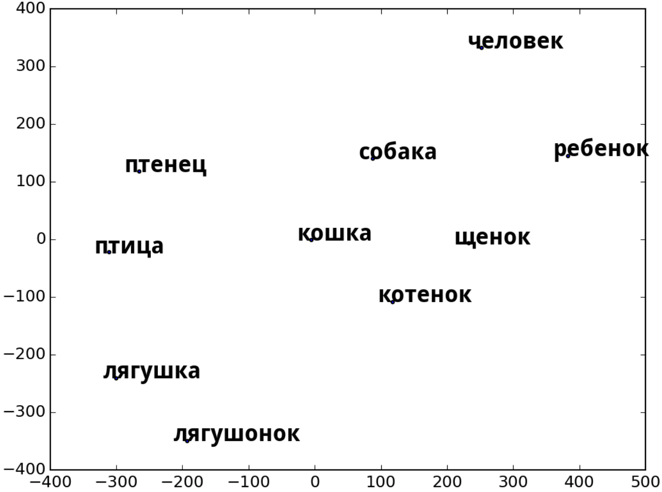


Table of Contents

Distributional semantic models for Russian on-line

As a sign of reverence to *RusCorpora* project, we launched *RusVectōrēs* web service:

RusVectores

Distributional semantic models for Russian on-line

As a sign of reverence to *RusCorpora* project, we launched *RusVectōrēs* web service:

`http://ling.go.mail.ru/dsm`

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*'крыло'* - *'самолет'* + *'машина'* = *'колесо'*);

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*‘крыло’* - *‘самолет’* + *‘машина’* = *‘колесо’*);
- Generate visualizations of word vectors and their relations to each other;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*'крыло'* - *'самолет'* + *'машина'* = *'колесо'*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*‘крыло’* - *‘самолет’* + *‘машина’* = *‘колесо’*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;
- Choose one of four models trained on different corpora;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*'крыло'* - *'самолет'* + *'машина'* = *'колесо'*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;
- Choose one of four models trained on different corpora;
- ... or upload your own corpus and have a model trained on it;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*‘крыло’* - *‘самолет’* + *‘машина’* = *‘колесо’*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;
- Choose one of four models trained on different corpora;
- ... or upload your own corpus and have a model trained on it;
- Every lemma in every model is identified by a unique URI:

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*‘крыло’* - *‘самолет’* + *‘машина’* = *‘колесо’*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;
- Choose one of four models trained on different corpora;
- ... or upload your own corpus and have a model trained on it;
- Every lemma in every model is identified by a unique URI:
  - *http://ling.go.mail.ru/dsm/ru/ruscorpora/разум* ;

RusVectores

`http://ling.go.mail.ru/dsm`:

- Find nearest semantic neighbors of Russian words;
- Compute cosine similarity between pairs of words;
- Perform algebraic operations on word vectors (*'крыло'* - *'самолет'* + *'машина'* = *'колесо'*);
- Generate visualizations of word vectors and their relations to each other;
- Optionally limit results to particular parts-of-speech;
- Choose one of four models trained on different corpora;
- ... or upload your own corpus and have a model trained on it;
- Every lemma in every model is identified by a unique URI:
    - *http://ling.go.mail.ru/dsm/ru/ruscorpora/разум* ;
- *Creative Commons Attribution license*;
- More to come (API)!

RusVectores

Used training corpora

Corpus	Size, tokens	Size, documents	Size, lemmas
News	1300 mln	9 mln	166 thousand
Web	620 mln	9 mln	750 thousand
Ruscorpora	107 mln	≈70 thousand	400 thousand
Ruscorpora+Russian Wikipedia	280 mln	≈1 mln	600 thousand

RusVectores

Used training corpora

Corpus	Size, tokens	Size, documents	Size, lemmas
News	1300 mln	9 mln	166 thousand
Web	620 mln	9 mln	750 thousand
Ruscorpora	107 mln	≈70 thousand	400 thousand
Ruscorpora+Russian Wikipedia	280 mln	≈1 mln	600 thousand

Lemmatized with MyStem 3.0, disambiguation turned on.
Stop-words and single-word sentences removed.

RusVectores

Play with it!

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

Table of Contents

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;
- Corpora comparison (including diachronic) via comparing models trained on these corpora;

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;
- Corpora comparison (including diachronic) via comparing models trained on these corpora;
- Clustering vector representations of words to get coarse semantic classes (*inter alia*, useful in NER recognition);

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;
- Corpora comparison (including diachronic) via comparing models trained on these corpora;
- Clustering vector representations of words to get coarse semantic classes (*inter alia*, useful in NER recognition);
- Using neural embeddings in search engines industry: query expansion, semantic hashing of documents, etc

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;
- Corpora comparison (including diachronic) via comparing models trained on these corpora;
- Clustering vector representations of words to get coarse semantic classes (*inter alia*, useful in NER recognition);
- Using neural embeddings in search engines industry: query expansion, semantic hashing of documents, etc
- Recommendation systems;

What else can be done with distributional semantic models

- Comprehensive study of differences between models trained with different hyper-parameters;
- Corpora comparison (including diachronic) via comparing models trained on these corpora;
- Clustering vector representations of words to get coarse semantic classes (*inter alia*, useful in NER recognition);
- Using neural embeddings in search engines industry: query expansion, semantic hashing of documents, etc
- Recommendation systems;
- Sentiment analysis;
- Event extraction, etc...

What else can be done with distributional semantic models

What it hot in the field

1. Finding out what particular neighbors were most important for training word vector: essentially, it is a question why these two words are similar.

What else can be done with distributional semantic models

What it hot in the field

1. Finding out what particular neighbors were most important for training word vector: essentially, it is a question why these two words are similar.
2. Compositional distributional semantics: how predict models can represent whole texts not with a simple sum or mean of word vectors?

What else can be done with distributional semantic models

What it hot in the field

1. Finding out what particular neighbors were most important for training word vector: essentially, it is a question why these two words are similar.
2. Compositional distributional semantics: how predict models can represent whole texts not with a simple sum or mean of word vectors?
3. Grounding: how to blend vector spaces for words and images or even sounds?

What else can be done with distributional semantic models

What it hot in the field

1. Finding out what particular neighbors were most important for training word vector: essentially, it is a question why these two words are similar.
2. Compositional distributional semantics: how predict models can represent whole texts not with a simple sum or mean of word vectors?
3. Grounding: how to blend vector spaces for words and images or even sounds?
4. Monitoring model dynamics as the model is trained with new data (temporal dimension added).

Table of Contents

Further reference

To read

1. Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." arXiv preprint arXiv:1301.3781 (2013).
2. Baroni, Marco, Georgiana Dinu, and Germán Kruszewski. "Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors." ACL 2014, Vol. 1.
3. Omer Levy, Yoav Goldberg, and Ido Dagan. "Improving Distributional Similarity with Lessons Learned from Word Embeddings". TACL 2015.
4. Le, Quoc V., and Tomas Mikolov. "Distributed representations of sentences and documents." arXiv preprint arXiv:1405.4053 (2014).
5. Pennington, Jeffrey, Richard Socher, and Christopher D. Manning. "Glove: Global vectors for word representation." EMNLP 2014: 1532-1543.
6. Kutuzov, Andrey and Andreev, Igor. "Texts in, meaning out: neural language models in semantic similarity task for Russian." Proceedings of the Dialog 2015 Conference, Moscow, Russia

Further reference

To lay hands on

1. https://code.google.com/p/word2vec/
2. http://radimrehurek.com/2014/02/word2vec-tutorial/
3. http://ling.go.mail.ru/dsm (for Russian)

Table of Contents

Q and A

Thank you!

Questions are welcome.

Word2vec and others buzzwords: unsupervised machine learning approach to distributional semantics

Andrey Kutuzov (andreku@ifi.uio.no)

Language Technology Group
University of Oslo

13 November 2015, AINL, Saint Petersburg