INF5820
Distributional Semantics: Extracting Meaning from Data

Lecture 2
Distributional and distributed: inner mechanics of modern word embedding models

Andrey Kutuzov
andreku@ifi.uio.no

2 November 2016

Contents

Brief recap

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];

Brief recap

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):

Brief recap

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);

Brief recap

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.

Brief recap

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.

Brief recap

- Distributional models are based on distributions of word co-occurrences in large training corpora;

Brief recap

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

Brief recap

- Distributional models are based on distributions of 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);

Brief recap

- Distributional models are based on distributions of 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);
- particular vector components (features) are not directly related to any particular semantic 'properties';

Brief recap

- Distributional models are based on distributions of 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);
- particular vector components (features) are not directly related to any particular semantic ‘properties’;
- words occurring in similar contexts have similar vectors;

Brief recap

- Distributional models are based on distributions of 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);
- particular vector components (features) are not directly related to any particular semantic 'properties';
- words occurring in similar contexts have similar vectors;
- one can find nearest semantic associates of a given word by calculating cosine similarity between vectors.

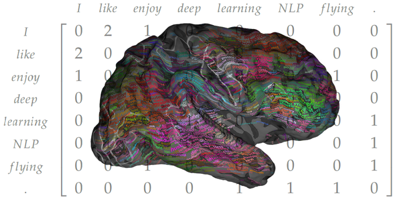
Brief recap

Nearest semantic associates

Brief recap

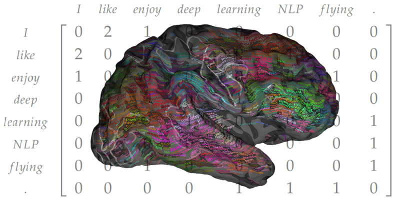
Nearest semantic associates

Brain *(from a model trained on English Wikipedia)*:



Brief recap

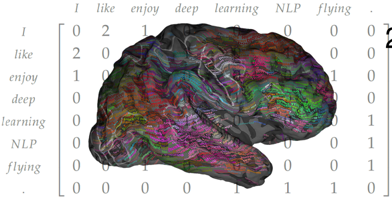
Nearest semantic associates



Brain *(from a model trained on English Wikipedia)*:

1. *cerebral* 0.74

Nearest semantic associates

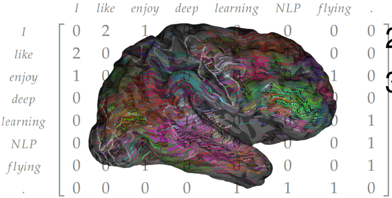


Brain *(from a model trained on English Wikipedia)*:

1. *cerebral* 0.74
2. *cerebellum* 0.72

Brief recap

Nearest semantic associates

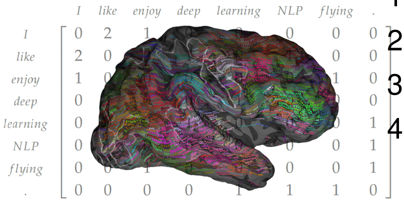


Brain *(from a model trained on English Wikipedia)*:

1. *cerebral* 0.74
2. *cerebellum* 0.72
3. *brainstem* 0.70

Brief recap

Nearest semantic associates

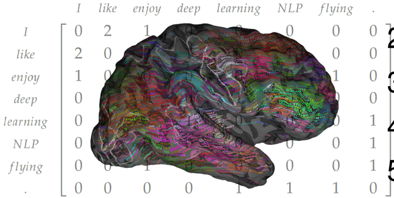


Brain *(from a model trained on English Wikipedia)*:

1. *cerebral* 0.74
2. *cerebellum* 0.72
3. *brainstem* 0.70
4. *cortical* 0.68

Brief recap

Nearest semantic associates



Brain *(from a model trained on English Wikipedia)*:

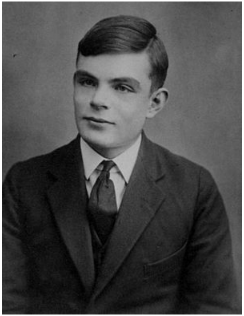
1. *cerebral* 0.74
2. *cerebellum* 0.72
3. *brainstem* 0.70
4. *cortical* 0.68
5. *hippocampal* 0.66
6. ...

Brief recap

Works with multi-word entities as well

Brief recap

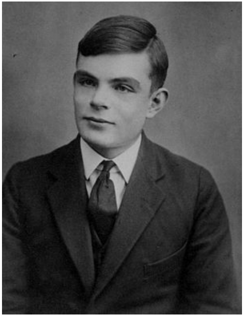
Works with multi-word entities as well



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

Brief recap

Works with multi-word entities as well

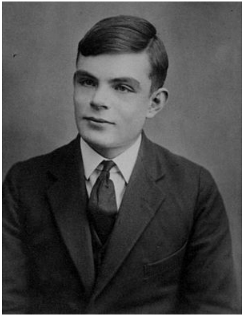


Alan_Turing *(from a model trained on Google News corpus (2013))*:

1. *Turing* 0.68

Brief recap

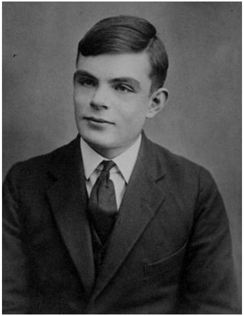
Works with multi-word entities as well



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

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

Works with multi-word entities as well

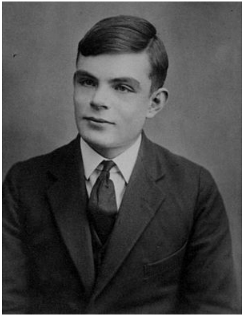


Alan_Turing *(from a model trained on Google News corpus (2013))*:

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

Brief recap

Works with multi-word entities as well

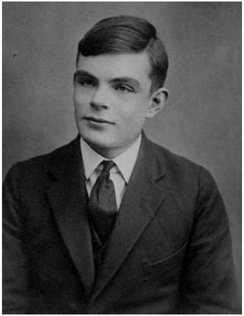


Alan_Turing *(from a model trained on Google News corpus (2013))*:

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

Brief recap

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. ...

Contents

Count-based distributional models

Traditional distributional models are known as count-based.

Count-based distributional models

Traditional distributional models are known as count-based.

How to construct a good count-based model

1. compile full co-occurrence matrix on the whole corpus;

Count-based distributional models

Traditional distributional models are known as count-based.

How to construct a good count-based model

1. compile full co-occurrence matrix on the whole corpus;
2. weigh absolute frequencies with positive point-wise mutual information (PPMI) association measure;

Count-based distributional models

Traditional distributional models are known as count-based.

How to construct a good count-based model

1. compile full co-occurrence matrix on the whole corpus;
2. weigh absolute frequencies with positive point-wise mutual information (PPMI) association measure;
3. factorize the matrix with singular value decomposition (SVD) to reduce dimensionality and arrive from sparse to dense vectors.

For more details, see [Bullinaria and Levy, 2007] and methods like Latent Semantic Indexing (LSI) or Latent Semantic Analysis (LSA).

Count-based distributional models

1. Matrix compilation

For each target word t we count how many times each context word c appeared in a pre-defined window around this target word.

Count-based distributional models

1. Matrix compilation

For each target word t we count how many times each context word c appeared in a pre-defined window around this target word.
The result is a vector of conditional probabilities
$p(c|t)$
for each target word.

Count-based distributional models

1. Matrix compilation

For each target word t we count how many times each context word c appeared in a pre-defined window around this target word.
The result is a vector of conditional probabilities
$p(c|t)$
for each target word.
The matrix of these vectors constitutes vector semantic space (VSM).

Count-based distributional models

1. Matrix compilation

For each target word t we count how many times each context word c appeared in a pre-defined window around this target word.
The result is a vector of conditional probabilities
$p(c|t)$
for each target word.
The matrix of these vectors constitutes vector semantic space (VSM).

Now we have to scale and weigh absolute frequency counts.

2. Probabilities weighting

PPMI (Positive point-wise mutual information) association measure seems to be the optimal choice. Let's recall:

Count-based distributional models

2. Probabilities weighting

PPMI (Positive point-wise mutual information) association measure seems to be the optimal choice. Let's recall:

$$PPMI(t, c) = \max(\log_2 \frac{p(t, c)}{p(t) * p(c)}, 0) \qquad (1)$$

where $p(t)$ – probability of t word in the whole corpus,
$p(c)$ – probability of c word in the whole corpus,
$p(t, c)$ – probability of t and c occurring together.

Count-based distributional models

2. Probabilities weighting

PPMI (Positive point-wise mutual information) association measure seems to be the optimal choice. Let's recall:

$$PPMI(t,c) = \max(\log_2 \frac{p(t,c)}{p(t) * p(c)}, 0) \quad (1)$$

where $p(t)$ – probability of t word in the whole corpus,
$p(c)$ – probability of c word in the whole corpus,
$p(t,c)$ – probability of t and c occurring together.
As a result, we pay less attention to random 'noise' co-occurrences.

3. Matrix factorization

To reduce the number of dimensions in the VSM, we can use one of many matrix factorization methods. The idea is to generate a lower-rank approximation of the original matrix (to truncate it), maximally retaining the relations between the vectors. It essentially means to find the most important dimensions of the data set, along which most variation happens.

Count-based distributional models

3. Matrix factorization

To reduce the number of dimensions in the VSM, we can use one of many matrix factorization methods. The idea is to generate a lower-rank approximation of the original matrix (to truncate it), maximally retaining the relations between the vectors. It essentially means to find the most important dimensions of the data set, along which most variation happens.

The most popular method to generate matrix approximations of any given rank k is Singular Value Decomposition or SVD, based on extracting so called *singular values* of the initial matrix. Other methods include PCA, factor analysis, etc, but truncated SVD is probably most widely used in NLP.

3. Matrix factorization

As a result, each word vector is now transformed into a dense embedding of k dimensions (typically hundreds), thus significantly reducing the dimensionality and often improving the models' performance.

3. Matrix factorization

As a result, each word vector is now transformed into a dense embedding of k dimensions (typically hundreds), thus significantly reducing the dimensionality and often improving the models' performance.
Matrix factorization can be easily performed in Python using, for example, Numpy: *numpy.linalg.svd*

3. Matrix factorization

As a result, each word vector is now transformed into a dense embedding of k dimensions (typically hundreds), thus significantly reducing the dimensionality and often improving the models' performance.
Matrix factorization can be easily performed in Python using, for example, Numpy: *numpy.linalg.svd*
Problem: SVD is often computationally expensive, especially for large vocabularies. The alternative is given by the predict(ive) models.

Contents

Predictive distributional models: Word2Vec revolution

Machine learning

Predictive distributional models: Word2Vec revolution

Machine learning

- Some problems are so complex that we can't formulate exact algorithms for them. We do not know ourselves how our brain does this.

Predictive distributional models: Word2Vec revolution

Machine learning

- Some problems are so complex that we can't formulate exact algorithms for them. We do not know ourselves how our brain does this.
- To solve such problems, one can use machine learning: attempts to build programs which learn to make correct decisions on some training material and improve with experience;

Machine learning

- Some problems are so complex that we can't formulate exact algorithms for them. We do not know ourselves how our brain does this.
- To solve such problems, one can use machine learning: attempts to build programs which learn to make correct decisions on some training material and improve with experience;
- One of popular machine learning approaches for language modeling – artificial neural networks.

Predictive distributional models: Word2Vec revolution

- Machine learning based distributional models are often called predict models.

Predictive distributional models: Word2Vec revolution

- Machine learning based distributional models are often called predict models.
- In the count models we count co-occurrence frequencies and use them as word vectors; in the predict models it is vice versa:

Predictive distributional models: Word2Vec revolution

- Machine learning based distributional models are often called predict models.
- In the count models we count co-occurrence frequencies and use them as word vectors; in the predict models it is vice versa:
- We try to find (to learn) for each word such a vector/embedding that it is maximally similar to the vectors of its paradigmatic neighbors and minimally similar to the vectors of the words which in the training corpus are not second-order neighbors of the given word.

When using artificial neural networks, such learned vectors are called neural embeddings.

Predictive distributional models: Word2Vec revolution

How brain works

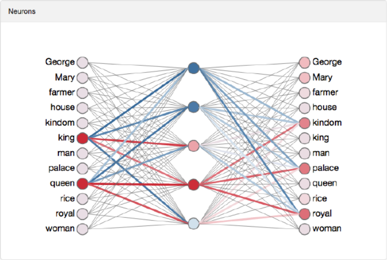
How brain works

There are 10^{11} neurons in our brain, with 10^4 connections each.

Predictive distributional models: Word2Vec revolution

How brain works

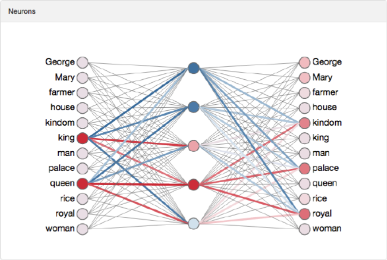
There are 10^{11} neurons in our brain, with 10^4 connections each.
Neurons receive differently expressed signals from other neurons.
Neuron reacts depending on the input.



Predictive distributional models: Word2Vec revolution

How brain works

There are 10^{11} neurons in our brain, with 10^{4} connections each.
Neurons receive differently expressed signals from other neurons.
Neuron reacts depending on the input.



Artificial neural networks try to imitate this process.

Predictive distributional models: Word2Vec revolution

Imitating the brain with artificial neural networks

Predictive distributional models: Word2Vec revolution

Imitating the brain with artificial neural networks

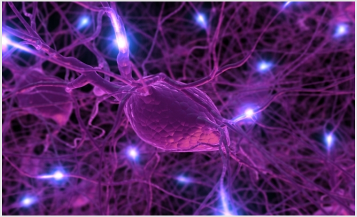
There is evidence that concepts are stored in brain as neural activation patterns.

Predictive distributional models: Word2Vec revolution

Imitating the brain with artificial neural networks

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'; each of them can be more or less activated (expressed).

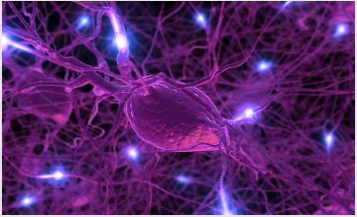


Predictive distributional models: Word2Vec revolution

Imitating the brain with artificial neural networks

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'; each of them can be more or less activated (expressed).



Concepts are represented by vectors of n dimensions (aka neurons), and each neuron is responsible for many concepts or rough 'semantic components'.

Predictive distributional models: Word2Vec revolution

In 2013, Google's Tomas Mikolov et al. published a paper called 'Efficient Estimation of Word Representations in Vector Space'; they also made available the source code of *word2vec* tool, implementing their algorithms, and a distributional model trained on large Google News corpus.

Predictive distributional models: Word2Vec revolution

In 2013, Google's Tomas Mikolov et al. published a paper called 'Efficient Estimation of Word Representations in Vector Space'; they also made available the source code of *word2vec* tool, implementing their algorithms, and a distributional model trained on large Google News corpus.

- [Mikolov et al., 2013]
- `https://code.google.com/p/word2vec/`

Predictive distributional models: Word2Vec revolution

In 2013, Google's Tomas Mikolov et al. published a paper called 'Efficient Estimation of Word Representations in Vector Space'; they also made available the source code of *word2vec* tool, implementing their algorithms, and a distributional model trained on large Google News corpus.

- [Mikolov et al., 2013]
- `https://code.google.com/p/word2vec/`

Mikolov modified already existing algorithms (especially from [Bengio et al., 2003] and work by R. Collobert), and explicitly made learning good embeddings the final aim of the model training.

Predictive distributional models: Word2Vec revolution

In 2013, Google's Tomas Mikolov et al. published a paper called 'Efficient Estimation of Word Representations in Vector Space'; they also made available the source code of *word2vec* tool, implementing their algorithms, and a distributional model trained on large Google News corpus.

- [Mikolov et al., 2013]
- `https://code.google.com/p/word2vec/`

Mikolov modified already existing algorithms (especially from [Bengio et al., 2003] and work by R. Collobert), and explicitly made learning good embeddings the final aim of the model training.

word2vec turned out to be very fast and efficient.

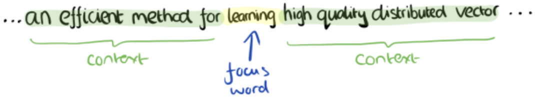
NB: it actually features two different algorithms: Continuous Bag-of-Words (CBOW) and Continuous Skipgram.

Predictive distributional models: Word2Vec revolution

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

Predictive distributional models: Word2Vec revolution

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

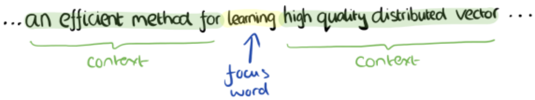


Learning good vectors

During the 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).

Predictive distributional models: Word2Vec revolution

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

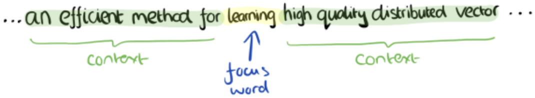


Learning good vectors

During the 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.

Predictive distributional models: Word2Vec revolution

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



Learning good vectors

During the 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.

It is important that prediction here is not an aim in itself: it is just a proxy to learn vector representations good for other downstream tasks.

Predictive distributional models: Word2Vec revolution

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

Predictive distributional models: Word2Vec revolution

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

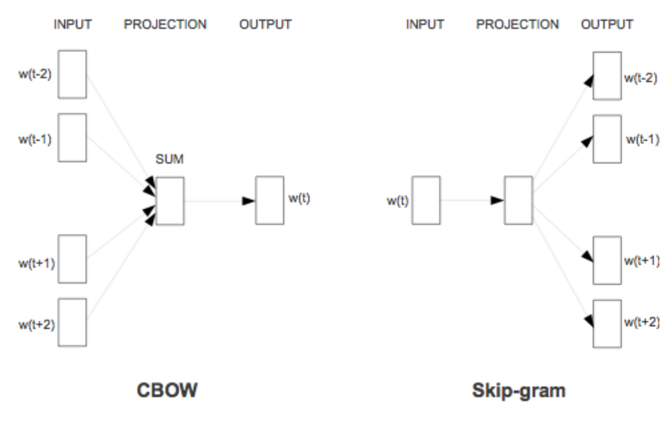
Predictive distributional models: Word2Vec revolution

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

Predictive distributional models: Word2Vec revolution

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



Predictive distributional models: Word2Vec revolution

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple, with a single hidden/projection layer.

Predictive distributional models: Word2Vec revolution

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

Predictive distributional models: Word2Vec revolution

It is clear that none of these algorithms is actually **deep** learning. Neural network is very simple, with a single hidden/projection layer.

The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, 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:

$$C = -log(p(w_t | cw_1 ... cw_j)) \tag{2}$$

Predictive distributional models: Word2Vec revolution

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple, with a single hidden/projection layer.
The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, 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:

$$C = -log(p(w_t|cw_1 ... cw_j)) \tag{2}$$

or for SkipGram

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

Predictive distributional models: Word2Vec revolution

It is clear that none of these algorithms is actually **deep** learning.
Neural network is very simple, with a single hidden/projection layer.
The training objective is to maximize the probability of observing the correct output word(s) w_t given the context word(s) $cw_1 ... cw_j$, 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:

$$C = -log(p(w_t|cw_1 ... cw_j)) \tag{2}$$

or for SkipGram

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

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

Predictive distributional models: Word2Vec revolution

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.

Predictive distributional models: Word2Vec revolution

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 and then turned into probabilities using softmax.

Predictive distributional models: Word2Vec revolution

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 and then turned into probabilities using softmax.

During the training, we are updating 2 weight matrices: of context vectors (from the input to the hidden layer) and of output vectors (from hidden layer to the output). As a rule, they share the same lexicon, and only output vectors are used in practical tasks.

Predictive distributional models: Word2Vec revolution

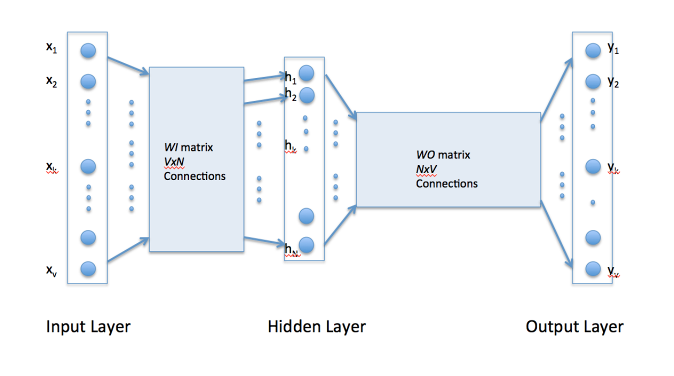
CBOW and SkipGram training algorithms

‘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]

Predictive distributional models: Word2Vec revolution

CBOW and SkipGram training algorithms

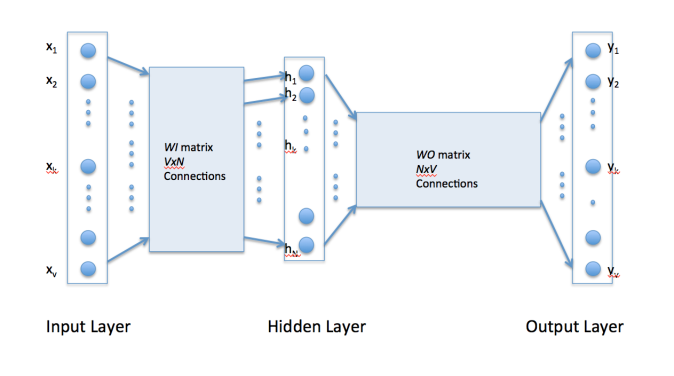
‘the **vector** of a word w is “dragged” back-and-forth by the **vectors** of w’s co-occurring words, as if there are physical strings between w and its neighbors...like gravity, or force-directed graph layout.’ [Rong, 2014]



Predictive distributional models: Word2Vec revolution

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/`

Predictive distributional models: Word2Vec revolution

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 and calculate their dot products with the input word(s).

Predictive distributional models: Word2Vec revolution

Selection of learning material

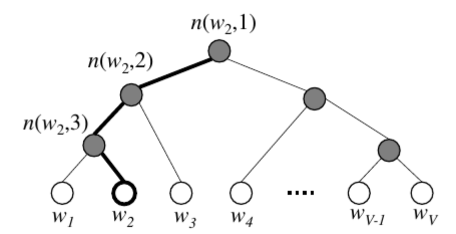
At each training instance, to find out whether the prediction is true, we have to iterate over all words in the vocabulary and calculate their dot products with the input word(s).

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

1. Hierarchical softmax;
2. Negative samping.

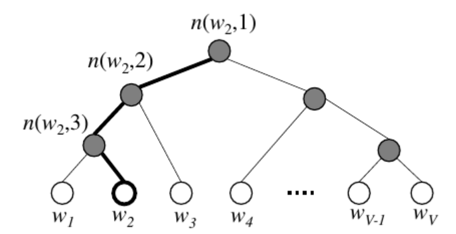
Predictive distributional models: Word2Vec revolution

Hierarchical softmax



Predictive distributional models: Word2Vec revolution

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 size V, the complexity of each prediction is $O(log(V))$ instead of $O(V)$.

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;

Predictive distributional models: Word2Vec revolution

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

Predictive distributional models: Word2Vec revolution

Things are complicated

Predictive distributional models: Word2Vec revolution

Things are complicated

Model performance hugely depends on training settings (hyperparameters):

Predictive distributional models: Word2Vec revolution

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).

Predictive distributional models: Word2Vec revolution

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.

Predictive distributional models: Word2Vec revolution

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.

Predictive distributional models: Word2Vec revolution

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;

Predictive distributional models: Word2Vec revolution

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);

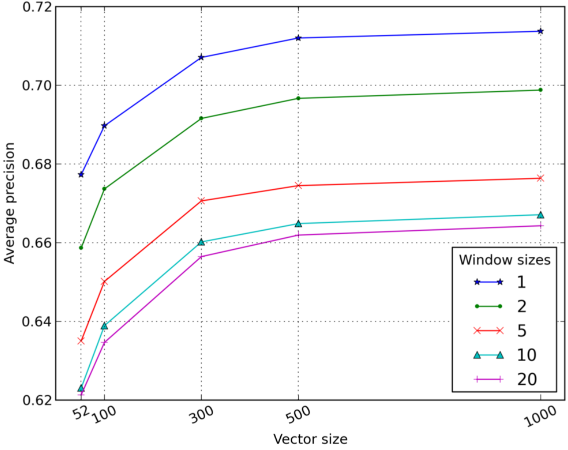
Predictive distributional models: Word2Vec revolution

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...

Predictive distributional models: Word2Vec revolution



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

Contents

The followers: GloVe and the others

In the next two years after 2013 Mikolov's paper, there was a lot of follow-up research:

- Christopher Mannning and other folks at Stanford released GloVe – a slightly different version of the same approach [Pennington et al., 2014];

The followers: GloVe and the others

In the next two years after 2013 Mikolov's paper, there was a lot of follow-up research:

- Christopher Mannning and other folks at Stanford released GloVe – a slightly different version of the same approach [Pennington et al., 2014];
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients [Levy and Goldberg, 2014];

The followers: GloVe and the others

In the next two years after 2013 Mikolov's paper, there was a lot of follow-up research:

- Christopher Mannning and other folks at Stanford released GloVe – a slightly different version of the same approach [Pennington et al., 2014];
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients [Levy and Goldberg, 2014];
- The same people showed that much of amazing performance of SkipGram is due to the choice of hyperparameters, but it is still very robust and computationally efficient [Levy et al., 2015];

The followers: GloVe and the others

In the next two years after 2013 Mikolov's paper, there was a lot of follow-up research:

- Christopher Mannning and other folks at Stanford released GloVe – a slightly different version of the same approach [Pennington et al., 2014];
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients [Levy and Goldberg, 2014];
- The same people showed that much of amazing performance of SkipGram is due to the choice of hyperparameters, but it is still very robust and computationally efficient [Levy et al., 2015];
- Le and Mikolov proposed Paragraph Vector: an algorithm to learn distributed representations not only for words but also for paragraphs or documents [Le and Mikolov, 2014];

The followers: GloVe and the others

In the next two years after 2013 Mikolov's paper, there was a lot of follow-up research:

- Christopher Mannning and other folks at Stanford released GloVe – a slightly different version of the same approach [Pennington et al., 2014];
- Omer Levy and Yoav Goldberg from Bar-Ilan University showed that SkipGram implicitly factorizes word-context matrix of PMI coefficients [Levy and Goldberg, 2014];
- The same people showed that much of amazing performance of SkipGram is due to the choice of hyperparameters, but it is still very robust and computationally efficient [Levy et al., 2015];
- Le and Mikolov proposed Paragraph Vector: an algorithm to learn distributed representations not only for words but also for paragraphs or documents [Le and Mikolov, 2014];
- These approaches were implemented in third-party open-source software, for example, Gensim or TensorFlow.

The followers: GloVe and the others

GlobalVectors: a global log-bilinear regression model for unsupervised learning of word embeddings

- GloVe is an attempt to combine the global matrix factorization (count) models and local context window (predict) models.

The followers: GloVe and the others

GlobalVectors: a global log-bilinear regression model for unsupervised learning of word embeddings

- GloVe is an attempt to combine the global matrix factorization (count) models and local context window (predict) models.
- It employs on global co-occurrence counts by factorizing the log of co-occurrence matrix

The followers: GloVe and the others

GlobalVectors: a global log-bilinear regression model for unsupervised learning of word embeddings

- GloVe is an attempt to combine the global matrix factorization (count) models and local context window (predict) models.
- It employs on global co-occurrence counts by factorizing the log of co-occurrence matrix
- Non-zero elements are stochastically sampled from the matrix, and the model iteratively trained on them.

The followers: GloVe and the others

GlobalVectors: a global log-bilinear regression model for unsupervised learning of word embeddings

- GloVe is an attempt to combine the global matrix factorization (count) models and local context window (predict) models.
- It employs on global co-occurrence counts by factorizing the log of co-occurrence matrix
- Non-zero elements are stochastically sampled from the matrix, and the model iteratively trained on them.
- The objective is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.

The followers: GloVe and the others

GlobalVectors: a global log-bilinear regression model for unsupervised learning of word embeddings

- GloVe is an attempt to combine the global matrix factorization (count) models and local context window (predict) models.
- It employs on global co-occurrence counts by factorizing the log of co-occurrence matrix
- Non-zero elements are stochastically sampled from the matrix, and the model iteratively trained on them.
- The objective is to learn word vectors such that their dot product equals the logarithm of the words' probability of co-occurrence.
- Code and pre-trained embeddings available at `http://nlp.stanford.edu/projects/glove/`.

References I

Baroni, M., Dinu, G., and Kruszewski, G. (2014).
Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors.
In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 238–247, Baltimore, USA.

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.

References II

Le, Q. V. and Mikolov, T. (2014).
Distributed representations of sentences and documents.
In *ICML*, volume 14, pages 1188–1196.

Levy, O. and Goldberg, Y. (2014).
Neural word embedding as implicit matrix factorization.
In *Advances in neural information processing systems*, pages 2177–2185.

Levy, O., Goldberg, Y., and Dagan, I. (2015).
Improving distributional similarity with lessons learned from word embeddings.
Transactions of the Association for Computational Linguistics, 3:211–225.

References III

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.

Rong, X. (2014).
word2vec parameter learning explained.
arXiv preprint arXiv:1411.2738.

Questions?

INF5820
Distributional Semantics: Extracting Meaning from Data
Lecture 2
Distributional and distributed: inner mechanics of modern word embedding models

Homework: play with
`http://ltr.uio.no/semvec`,
install *Gensim* library for *Python* (`http://radimrehurek.com/gensim/`).

Contents

In the next week

Practical aspects of training and using distributional models

- Models hyperparameters;
- Models evaluation;
- Models' formats;
- Off-the-shelf tools to train and use models.