# IN5550: Neural Methods in Natural Language Processing
## Sub-lecture 5.4
## *Word embeddings: miscellaneous*

Andrey Kutuzov

University of Oslo

17 February 2022



# Contents




1. Training word embeddings
2. The followers
3. Practical aspects
4. Demo web service and pre-trained vectors repository
5. Next group session: February 22

# Training word embeddings

## Prediction-based 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]

# Training word embeddings

## Prediction-based 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/`

# Contents




1. Training word embeddings
2. The followers
3. Practical aspects
4. Demo web service and pre-trained vectors repository
5. Next group session: February 22

# The followers

In the years after Mikolov's 2013 paper, there was a lot of follow-up works (to mention only a few):

- [Pennington et al., 2014] released GloVe – a hybrid approach combining count matrices and ML;

# The followers

In the years after Mikolov's 2013 paper, there was a lot of follow-up works (to mention only a few):

- [Pennington et al., 2014] released GloVe – a hybrid approach combining count matrices and ML;
- [Levy and Goldberg, 2014] showed that SkipGram implicitly factorizes a count-based word-context PPMI matrix (never explicitly constructing it);

# The followers

In the years after Mikolov's 2013 paper, there was a lot of follow-up works (to mention only a few):

- [Pennington et al., 2014] released GloVe – a hybrid approach combining count matrices and ML;
- [Levy and Goldberg, 2014] showed that SkipGram implicitly factorizes a count-based word-context PPMI matrix (never explicitly constructing it);
- [Le and Mikolov, 2014] proposed Paragraph Vector ('doc2vec'): an algorithm to learn dense representations not only for words but also for paragraphs or documents;

# The followers

In the years after Mikolov's 2013 paper, there was a lot of follow-up works (to mention only a few):

- [Pennington et al., 2014] released GloVe – a hybrid approach combining count matrices and ML;
- [Levy and Goldberg, 2014] showed that SkipGram implicitly factorizes a count-based word-context PPMI matrix (never explicitly constructing it);
- [Le and Mikolov, 2014] proposed Paragraph Vector ('doc2vec'): an algorithm to learn dense representations not only for words but also for paragraphs or documents;
- After moving to Facebook, Mikolov and other folks released fastText, which learns embeddings using subword data [Bojanowski et al., 2017];

# The followers

In the years after Mikolov's 2013 paper, there was a lot of follow-up works (to mention only a few):

- [Pennington et al., 2014] released GloVe – a hybrid approach combining count matrices and ML;
- [Levy and Goldberg, 2014] showed that SkipGram implicitly factorizes a count-based word-context PPMI matrix (never explicitly constructing it);
- [Le and Mikolov, 2014] proposed Paragraph Vector ('doc2vec'): an algorithm to learn dense representations not only for words but also for paragraphs or documents;
- After moving to Facebook, Mikolov and other folks released fastText, which learns embeddings using subword data [Bojanowski et al., 2017];
- These approaches were implemented in third-party open-source software: Gensim, PyTorch, TensorFlow...

# The followers



## fastText

- A vector is learned for each character n-gram

# The followers



## fastText

- A vector is learned for each character n-gram
- Word are represented as the sum of their character n-grams vectors.

# The followers



## fastText

- A vector is learned for each character n-gram
- Word are represented as the sum of their character n-grams vectors.
- Partially solves the out-of-vocabulary (OOV) words problem:

# The followers



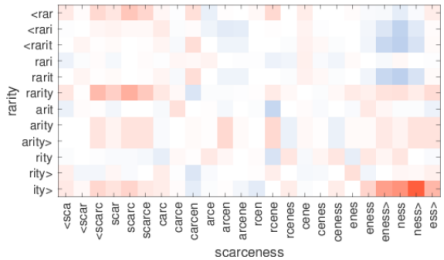
## fastText

- A vector is learned for each character n-gram
- Word are represented as the sum of their character n-grams vectors.
- Partially solves the out-of-vocabulary (OOV) words problem:
- unknown words still contain known n-grams.

# The followers

## fastText

- A vector is learned for each character n-gram
- Word are represented as the sum of their character n-grams vectors.
- Partially solves the out-of-vocabulary (OOV) words problem:
- unknown words still contain known n-grams.
- `https://fasttext.cc/`
- `http://vectors.nlpl.eu/repository/`



# The followers



Word (or subword) embeddings are now used as an input to neural network models:

# The followers



Word (or subword) embeddings are now used as an input to neural network models:

- feed-forward networks,

# The followers



Word (or subword) embeddings are now used as an input to neural network models:

- feed-forward networks,
- convolutional networks (CNNs)

# The followers



Word (or subword) embeddings are now used as an input to neural network models:

- feed-forward networks,
- convolutional networks (CNNs)
- recurrent networks (RNNs, you will use them in the Obligatory 2),

# The followers



Word (or subword) embeddings are now used as an input to neural network models:

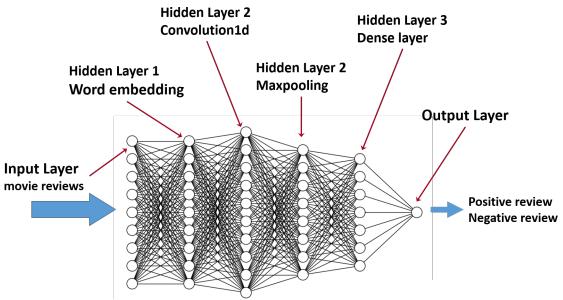
- feed-forward networks,
- convolutional networks (CNNs)
- recurrent networks (RNNs, you will use them in the Obligatory 2),
- transformers, etc.

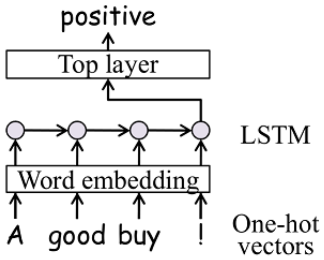
# The followers

Word (or subword) embeddings are now used as an input to neural network models:

- feed-forward networks,
- convolutional networks (CNNs)
- recurrent networks (RNNs, you will use them in the Obligatory 2),
- transformers, etc.

Embeddings themselves can be updated at the training time along with the rest of the network weights or 'frozen' (protected from updating).



# Contents




1. Training word embeddings
2. The followers
3. Practical aspects
4. Demo web service and pre-trained vectors repository
5. Next group session: February 22

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);
2. word2vec original C code [Le and Mikolov, 2014] (`https://code.google.com/archive/p/word2vec/`)

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);
2. word2vec original C code [Le and Mikolov, 2014] (`https://code.google.com/archive/p/word2vec/`)
3. word2vec implementation in *TensorFlow* (`https://www.tensorflow.org/tutorials/word2vec`);

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);
2. word2vec original C code [Le and Mikolov, 2014] (`https://code.google.com/archive/p/word2vec/`)
3. word2vec implementation in *TensorFlow* (`https://www.tensorflow.org/tutorials/word2vec`);
4. word2vec implementation in *PyTorch* (`https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`);

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);
2. word2vec original C code [Le and Mikolov, 2014] (`https://code.google.com/archive/p/word2vec/`)
3. word2vec implementation in *TensorFlow* (`https://www.tensorflow.org/tutorials/word2vec`);
4. word2vec implementation in *PyTorch* (`https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`);
5. fastText official implementation by *Facebook* [Bojanowski et al., 2017] (`https://fasttext.cc/`);

# Practical aspects



## Software to train word embeddings on your data

1. *Gensim* library for Python, including word2vec and fastText implementations (`https://github.com/RaRe-Technologies/gensim`);
2. word2vec original C code [Le and Mikolov, 2014] (`https://code.google.com/archive/p/word2vec/`)
3. word2vec implementation in *TensorFlow* (`https://www.tensorflow.org/tutorials/word2vec`);
4. word2vec implementation in *PyTorch* (`https://pytorch.org/tutorials/beginner/nlp/word_embeddings_tutorial.html`);
5. fastText official implementation by *Facebook* [Bojanowski et al., 2017] (`https://fasttext.cc/`);
6. GloVe reference implementation [Pennington et al., 2014] (`http://nlp.stanford.edu/projects/glove/`).

# Practical aspects



**Pre-trained word embedding models can come in several formats:**

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

# Practical aspects



**Pre-trained word embedding models can come in several formats:**

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

# Practical aspects



**Pre-trained word embedding models can come in several formats:**

1. Simple text format, `*.txt` or `*.vec`:
   - words and sequences of values representing their vectors, one word per line;
   - the first line gives information on the number of words in the model and the vector size.
2. The same in the binary form, `*.bin`.
3. fastText `*.bin` files additionally contain vectors for subword units;

# Practical aspects

**Pre-trained word embedding models can come in several formats:**

1. Simple text format, `*.txt` or `*.vec`:
   - words and sequences of values representing their vectors, one word per line;
   - the first line gives information on the number of words in the model and the vector size.
2. The same in the binary form, `*.bin`.
3. fastText `*.bin` files additionally contain vectors for subword units;
4. Gensim format:
   - uses *NumPy* arrays;
   - stores additional information (training weights, hyperparameters, word frequency, etc).

# Practical aspects

**Pre-trained word embedding models can come in several formats:**

1. Simple text format, `*.txt` or `*.vec`:
   - words and sequences of values representing their vectors, one word per line;
   - the first line gives information on the number of words in the model and the vector size.
2. The same in the binary form, `*.bin`.
3. fastText `*.bin` files additionally contain vectors for subword units;
4. Gensim format:
   - uses *NumPy* arrays;
   - stores additional information (training weights, hyperparameters, word frequency, etc).

*Gensim* works with all these formats (also if compressed). Recommended for the Obligatory 2.
On Saga: `module load nlpl-gensim/3.8.3-gomkl-2019b-Python-3.7.4`



**To sum up**

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



**To sum up**

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



**To sum up**

- Distributional models are based on distributions of word co-occurrences in large training corpora;
- they represent lexical meanings as dense vectors (embeddings);
- the models are also distributed: meaning is expressed via values of multiple vector entries;



**To sum up**

- Distributional models are based on distributions of word co-occurrences in large training corpora;
- they represent lexical meanings as dense vectors (embeddings);
- the models are also distributed: meaning is expressed via values of multiple vector entries;
- particular vector entries (features) are not directly related to any particular semantic 'properties', and thus not directly interpretable;



**To sum up**

- Distributional models are based on distributions of word co-occurrences in large training corpora;
- they represent lexical meanings as dense vectors (embeddings);
- the models are also distributed: meaning is expressed via values of multiple vector entries;
- particular vector entries (features) are not directly related to any particular semantic 'properties', and thus not directly interpretable;
- words occurring in similar contexts have similar vectors.

# Contents




1. Training word embeddings
2. The followers
3. Practical aspects
4. Demo web service and pre-trained vectors repository
5. Next group session: February 22

# Demo web service and pre-trained vectors repository



## Distributional semantic models for English and Norwegian online

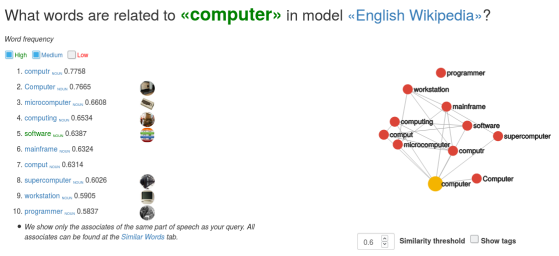
You can try the *WebVectors* service developed by our Language Technology group

# Demo web service and pre-trained vectors repository

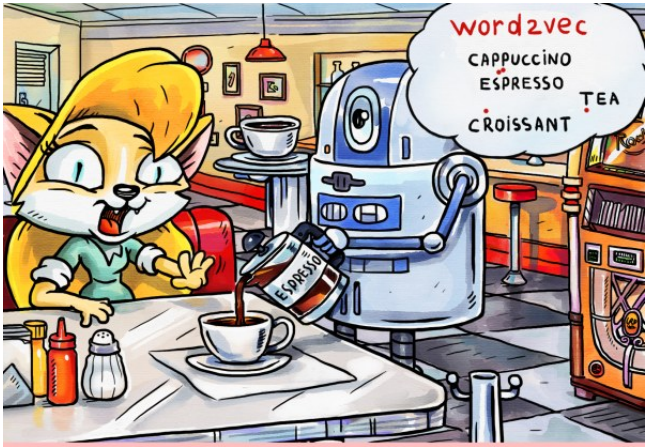
## Distributional semantic models for English and Norwegian online

You can try the *WebVectors* service developed by our Language Technology group

`http://vectors.nlpl.eu/explore/embeddings/`
`http://vectors.nlpl.eu/repository/`



# Demo web service and pre-trained vectors repository




- Espresso? But I ordered a cappuccino!
- Don't worry, the cosine distance between them is so small that they are almost the same thing.

*(by Dmitry Malkov)*

# Contents




1. Training word embeddings
2. The followers
3. Practical aspects
4. Demo web service and pre-trained vectors repository
5. Next group session: February 22

# Next group session: February 22



- Discussing Obligatory 1
- Training *word2vec* embeddings using *PyTorch*.

# References I


Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017).
Enriching word vectors with subword information.
Transactions of the Association of Computational Linguistics, 5:135–146.

Le, Q. and Mikolov, T. (2014).
Distributed representations of sentences and documents.
In International Conference on Machine Learning, 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.

Pennington, J., Socher, R., and Manning, C. D. (2014).
Glove: Global vectors for word representation.
In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543.

# References II




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