IN5550: Neural Methods in Natural Language Processing
Sub-lecture 5.3
Machine learning based distributional models

Andrey Kutuzov

University of Oslo

17 February 2022



Contents



1. Machine learning based distributional models
   - Word2Vec revolution

Machine learning based distributional models



Main contemporary approaches to produce word embeddings

1. *Word-context* co-occurrence matrices, factorized by *SVD* (count-based approaches);

Machine learning based distributional models



Main contemporary approaches to produce word embeddings

1. *Word-context* co-occurrence matrices, factorized by *SVD* (count-based approaches);
2. Predict(ive) approaches using *neural language models*, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (word2vec):

Machine learning based distributional models



Main contemporary approaches to produce word embeddings

1. *Word-context* co-occurrence matrices, factorized by *SVD* (count-based approaches);
2. Predict(ive) approaches using *neural language models*, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (word2vec):
   - Continuous Bag-of-Words (CBOW),
   - Continuous Skip-Gram (skipgram);
   - fastText [Bojanowski et al., 2017];

Machine learning based distributional models



Main contemporary approaches to produce word embeddings

1. *Word-context* co-occurrence matrices, factorized by *SVD* (count-based approaches);
2. Predict(ive) approaches using *neural language models*, introduced in [Bengio et al., 2003] and [Mikolov et al., 2013] (word2vec):
   - Continuous Bag-of-Words (CBOW),
   - Continuous Skip-Gram (skipgram);
   - fastText [Bojanowski et al., 2017];
3. Hybrid approaches like Global Vectors for Word Representation (GloVe) [Pennington et al., 2014].

1 is not used much any more.

2 and 3 are being right now increasingly replaced with contextualized models; more on that in later lectures.

Machine learning based distributional models



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

Machine learning based distributional models



- 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:

Machine learning based distributional models



- 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 a vector/embedding such that it is...
  - maximally similar to the vectors of its paradigmatic neighbors,
  - minimally similar to the vectors of the words which in the training corpus are not paradigmatic neighbors of the given word.

Machine learning based distributional models



- 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 a vector/embedding such that it is...
  - maximally similar to the vectors of its paradigmatic neighbors,
  - minimally similar to the vectors of the words which in the training corpus are not paradigmatic neighbors of the given word.
- We do it using word co-occurrences in corpora as a supervision signal...
  - ...as we have already seen in neural language models.

Word2Vec revolution



Word2Vec revolution



- In 2013, Tomas Mikolov et al. from Google published a paper 'Efficient Estimation of Word Representations in Vector Space' [Mikolov et al., 2013];

Word2Vec revolution



- In 2013, Tomas Mikolov et al. from Google published a paper 'Efficient Estimation of Word Representations in Vector Space' [Mikolov et al., 2013];
- they also made available the source code of *word2vec* tool, implementing their algorithms...

Word2Vec revolution



- In 2013, Tomas Mikolov et al. from Google published a paper 'Efficient Estimation of Word Representations in Vector Space' [Mikolov et al., 2013];
- they also made available the source code of *word2vec* tool, implementing their algorithms...
- `https://code.google.com/p/word2vec/`

Word2Vec revolution



- In 2013, Tomas Mikolov et al. from Google published a paper 'Efficient Estimation of Word Representations in Vector Space' [Mikolov et al., 2013];
- they also made available the source code of *word2vec* tool, implementing their algorithms...
- `https://code.google.com/p/word2vec/`
- ...and a distributional model trained on the large Google News corpus
- Just a set of vectors for English words.

Word2Vec revolution



- Mikolov slightly modified already existing neural language modeling algorithms ([Bengio et al., 2003], [Collobert and Weston, 2008]),

Word2Vec revolution



- Mikolov slightly modified already existing neural language modeling algorithms ([Bengio et al., 2003], [Collobert and Weston, 2008]),
- explicitly made learning good word embeddings the final aim of the language model training, not just a byproduct.

Word2Vec revolution



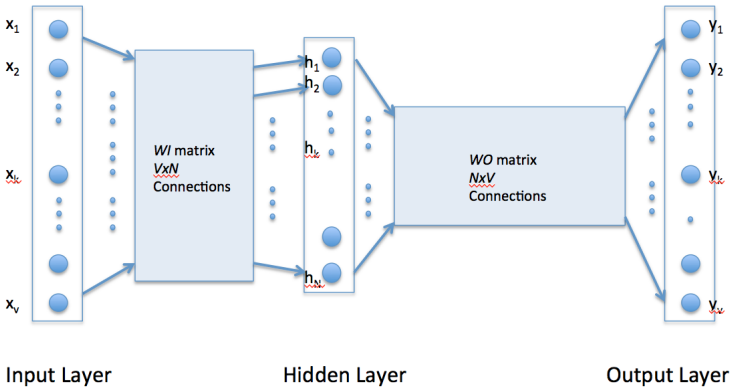
- Mikolov slightly modified already existing neural language modeling algorithms ([Bengio et al., 2003], [Collobert and Weston, 2008]),
- explicitly made learning good word embeddings the final aim of the language model training, not just a byproduct.
- *word2vec* turned out to be very fast and efficient.

Word2Vec revolution



- Mikolov slightly modified already existing neural language modeling algorithms ([Bengio et al., 2003], [Collobert and Weston, 2008]),
- explicitly made learning good word embeddings the final aim of the language model training, not just a byproduct.
- *word2vec* turned out to be very fast and efficient.
- It actually featured two different algorithms:
  1. Continuous Bag-of-Words (CBOW),
  2. Continuous Skipgram.
- Both use a very simple feed-forward neural network.
- No non-linear activations in the hidden layer, can hardly be considered a neural network at all!
- But this is why it is so fast.

Word2Vec revolution





Model initializing

First, each word in the vocabulary V is assigned 2 vectors of uniformly sampled random numbers (as a *word* and as a *context*) of a pre-defined size N. Thus, we have two weight matrices:



Model initializing

First, each word in the vocabulary V is assigned 2 vectors of uniformly sampled random numbers (as a *word* and as a *context*) of a pre-defined size N. Thus, we have two weight matrices:

- input matrix W^I with word vectors between input and hidden layers;

Model initializing

First, each word in the vocabulary V is assigned 2 vectors of uniformly sampled random numbers (as a *word* and as a *context*) of a pre-defined size N. Thus, we have two weight matrices:

- input matrix W^I with word vectors between input and hidden layers;
- output matrix W^O with context vectors between hidden and output layers.

Model initializing

First, each word in the vocabulary V is assigned 2 vectors of uniformly sampled random numbers (as a *word* and as a *context*) of a pre-defined size N. Thus, we have two weight matrices:

- input matrix W^I with word vectors between input and hidden layers;
- output matrix W^O with context vectors between hidden and output layers.
- The W^I shape is vocabulary size $\times$ vector size ($|V| \times N$)
- The W^O shape is the transposition of W^I: vector size $\times$ vocabulary size ($N \times |V|$).

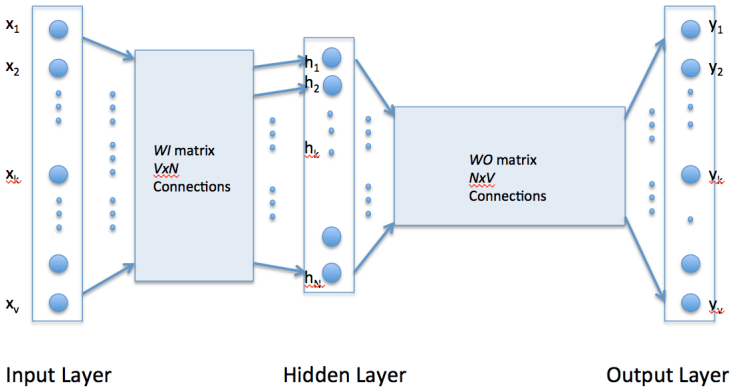
Model initializing

First, each word in the vocabulary V is assigned 2 vectors of uniformly sampled random numbers (as a *word* and as a *context*) of a pre-defined size N. Thus, we have two weight matrices:

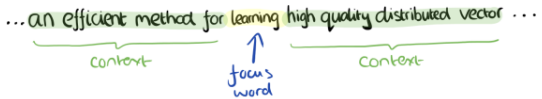
- input matrix W^I with word vectors between input and hidden layers;
- output matrix W^O with context vectors between hidden and output layers.
- The W^I shape is vocabulary size $\times$ vector size ($|V| \times N$)
- The W^O shape is the transposition of W^I: vector size $\times$ vocabulary size ($N \times |V|$).

What happens next?

Word2Vec revolution



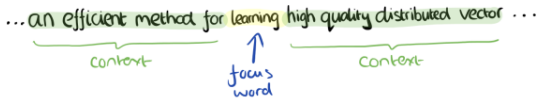
Word2Vec revolution



Learning good vectors

- While training, we move through the corpus with a sliding window.

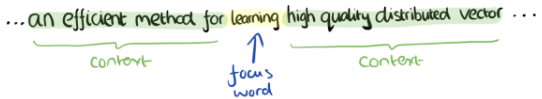
Word2Vec revolution



Learning good vectors

- While training, we move through the 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.

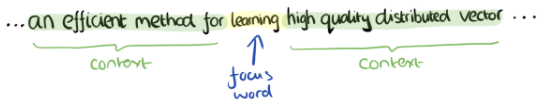
Word2Vec revolution



Learning good vectors

- While training, we move through the 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.
- NB: the context is now not only previous words (as in classic LMs), but also the next ones ('symmetric context').

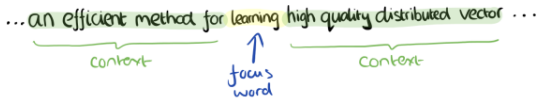
Word2Vec revolution



Learning good vectors

- While training, we move through the 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.
- NB: the context is now not only previous words (as in classic LMs), but also the next ones ('symmetric context').
- The outcomes of the predictions determine the direction of adjusting the weights/entries of the current word vectors. Gradually, vectors converge to (hopefully) optimal values.

Word2Vec revolution



Learning good vectors

- While training, we move through the 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.
- NB: the context is now not only previous words (as in classic LMs), but also the next ones ('symmetric context').
- The outcomes of the predictions determine the direction of adjusting the weights/entries of the current word vectors. Gradually, vectors converge to (hopefully) optimal values.

Prediction not an aim in itself: it is just a proxy to learn vector representations for words, useful for other downstream tasks.

Word2Vec revolution



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

Word2Vec revolution



- Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram) are conceptually similar but differ in some details;
- At training time, CBOW learns to predict current word based on its context...

Word2Vec revolution



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

Word2Vec revolution

- Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram) are conceptually similar but differ in some details;
- 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.
- NB: context (window size and nature) can be defined in many different ways;

Word2Vec revolution



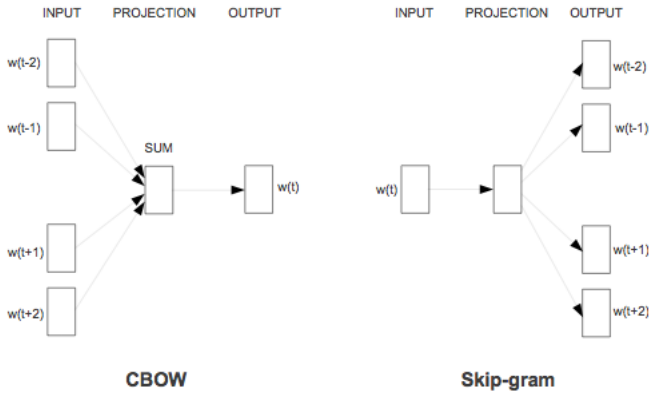
- Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram) are conceptually similar but differ in some details;
- At training time, CBOW learns to predict current word based on its context...
- ...while Skip-Gram learns to predict context based on the current word.
- NB: context (window size and nature) can be defined in many different ways;
- this greatly influences the resulting models.

Word2Vec revolution



- Continuous Bag-of-words (CBOW) and Continuous Skip-gram (skip-gram) are conceptually similar but differ in some details;
- 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.
- NB: context (window size and nature) can be defined in many different ways;
- this greatly influences the resulting models.
- Both CBOW and Skip-Gram outperformed traditional count DSMs in various semantic tasks [Baroni et al., 2014]

Word2Vec revolution



Continuous Bag-of-Words and Continuous Skip-Gram:
two algorithms in the *word2vec* paper [Mikolov et al., 2013]

What we are updating during the training?

- weight matrix of input vectors W^I (from the input layer to the hidden layer)
- weight matrix of output vectors W^O (from the hidden layer to the output layer)
- they share the same vocabulary, and only the input vectors are used in practical tasks.



What we are updating during the training?

- weight matrix of input vectors W^I (from the input layer to the hidden layer)
- weight matrix of output vectors W^O (from the hidden layer to the output layer)
- they share the same vocabulary, and only the input vectors are used in practical tasks.

At each training instance, the input for the prediction is:

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

What we are updating during the training?

- weight matrix of input vectors W^I (from the input layer to the hidden layer)
- weight matrix of output vectors W^O (from the hidden layer to the output layer)
- they share the same vocabulary, and only the input vectors are used in practical tasks.

At each training instance, the input for the prediction is:

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

Reminder: this 'closeness' is calculated with the help of dot product or cosine similarity and then turned into probabilities using softmax.

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.
- The training objective is to maximize the probability of observing the correct output word $\boldsymbol{w}_t$ given the context words $\boldsymbol{c}_1 ... \boldsymbol{c}_j$, with regard to their current embeddings (sets of weights in the matrices).

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.
- The training objective is to maximize the probability of observing the correct output word $\boldsymbol{w}_t$ given the context words $\boldsymbol{c}_1 ... \boldsymbol{c}_j$, with regard to their current embeddings (sets of weights in the matrices).
- Loss function L is cross-entropy:

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.
- The training objective is to maximize the probability of observing the correct output word $\boldsymbol{w}_t$ given the context words $\boldsymbol{c}_1 ... \boldsymbol{c}_j$, with regard to their current embeddings (sets of weights in the matrices).
- Loss function L is cross-entropy:

For CBOW:

$$L = -\log\left(P\left(\boldsymbol{w}_t \middle| \sum_{i=1}^{j} \boldsymbol{c}_i\right)\right) \tag{1}$$

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.
- The training objective is to maximize the probability of observing the correct output word $\boldsymbol{w}_t$ given the context words $\boldsymbol{c}_1 ... \boldsymbol{c}_j$, with regard to their current embeddings (sets of weights in the matrices).
- Loss function L is cross-entropy:

For CBOW:

$$L = -\log\left(P\left(\boldsymbol{w}_t \middle| \sum_{i=1}^{j} \boldsymbol{c}_i\right)\right) \tag{1}$$

For SkipGram:

$$L = -\sum_{i=1}^{j} \log\left(P(\boldsymbol{c}_i|\boldsymbol{w}_t)\right) \tag{2}$$

Word2Vec revolution



- The network architecture in *word2vec* is very simple, with a single hidden/projection layer between the input and the output layers.
- The training objective is to maximize the probability of observing the correct output word $\boldsymbol{w}_t$ given the context words $\boldsymbol{c}_1 ... \boldsymbol{c}_j$, with regard to their current embeddings (sets of weights in the matrices).
- Loss function L is cross-entropy:

For CBOW:

$$L = -\log\left(P\left(\boldsymbol{w}_t \middle| \sum_{i=1}^{j} \boldsymbol{c}_i\right)\right) \tag{1}$$

For SkipGram:

$$L = -\sum_{i=1}^{j} \log\left(P(\boldsymbol{c}_i|\boldsymbol{w}_t)\right) \tag{2}$$

and the learning itself is implemented with stochastic gradient descent.



Selection of learning material

To evaluate each prediction, at the output layer we have to take all words in V, calculate their dot products with the current prediction, and then produce a probability distribution with softmax.



Selection of learning material

To evaluate each prediction, at the output layer we have to take all words in V, calculate their dot products with the current prediction, and then produce a probability distribution with softmax.

But softmax is expensive! And we have millions and billions of training instances!



Selection of learning material

To evaluate each prediction, at the output layer we have to take all words in V, calculate their dot products with the current prediction, and then produce a probability distribution with softmax.

But softmax is expensive! And we have millions and billions of training instances!

That's why *word2vec* employs one of these two smart tricks:

1. Hierarchical softmax (rarely used nowadays);
2. Negative sampling.



Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:



Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:

- do not take all words in the vocabulary;



Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:

- do not take all words in the vocabulary;
- take your true prediction and sample 5...15 random 'noise' words from the vocabulary;



Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:

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



Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:

- do not take all words in the vocabulary;
- take your true prediction and sample 5...15 random 'noise' words from the vocabulary;
- these words serve as negative examples.
- The task boils down to binary classification (real or random word pair?).
- Dealing with 15 word pairs is much faster than applying softmax to the full vocabulary.

Negative sampling (dominant nowadays)

The idea of negative sampling or noise-contrastive estimation (NCE) is quite simple:

- do not take all words in the vocabulary;
- take your true prediction and sample 5...15 random 'noise' words from the vocabulary;
- these words serve as negative examples.
- The task boils down to binary classification (real or random word pair?).
- Dealing with 15 word pairs is much faster than applying softmax to the full vocabulary.

SkipGram model with Negative Sampling is often called simply 'SGNS'.
Still often seen in NLP papers.

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.

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.

References II


Collobert, R. and Weston, J. (2008).
A unified architecture for natural language processing: Deep neural networks with multitask learning.
In Proceedings of the 25th international conference on Machine learning, pages 160–167. ACM.


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.