Advanced Topics in Machine Learning

Alejo Nevado-Holgado

Lecture 10 (NLP 2) - Embeddings 2

V 0.6 (15 Feb 2020 - minor improvements after lecture)

Course structure

- **Introduction:** What is NLP. Why it is hard. Why NNs work well ← Lecture 9 (NLP 1)
- **Word representation:** How to represent the meaning of individual words
  - Old technology: One-hot representations, synsets ← Lecture 9 (NLP 1)
  - Embeddings: First trick that boosted the performance of NNs in NLP ← Lecture 9 (NLP 1)
    - Word2vec: Single layer NN. CBOW and skip-gram ← Lecture 10 (NLP 2)
    - Co-occurrence matrices: Basic counts and SVD improvement ← Lecture 10 (NLP 2)
    - Glove: Combining word2vec and co-occurrence matrices idea ← Lecture 10 (NLP 2)
    - Evaluating performance of embeddings ← Lecture 10 (NLP 2)
- **Named Entity Recognition (NER):** How to find words of specific meaning within text
  - Multilayer NNs: Margin loss. Forward- and back-propagation ← Lecture 11 (NLP 3)
  - Better loss functions: margin loss, regularisation ← Lecture 11 (NLP 3)
  - Better initializations: uniform, xavier ← Lecture 11 (NLP 3)
  - Better optimizers: Adagrad, RMSprop, Adam... ← Lecture 11 (NLP 3)

Course structure

- **Language modelling:** How to represent the meaning of full pieces of text
  - Old technology: N-grams ← Lecture 12 (NLP 4)
  - Recursive NNs language models (RNNs) ← Lecture 12 (NLP 4)
  - Evaluating performance of language models ← Lecture 12 (NLP 4)
  - Vanishing gradients: Problem. Gradient clipping ← Lecture 13 (NLP 5)
  - Improved RNNs: LSTM, GRU ← Lecture 13 (NLP 5)
- **Machine translation:** How to translate text
  - Old technology: Georgetown–IBM experiment and ALPAC report ← Lecture 16 (NLP 6)
  - Seq2seq: Greedy decoding, encoder-decoder, beam search ← Lecture 16 (NLP 6)
  - Attention: Simple attention, transformers, reformers ← Lecture 16 (NLP 6)
  - Evaluating performance: BLEU ← Lecture 16 (NLP 6)

Embeddings: word2vec

- Word2vec very successfully implemented word embeddings using this context-meaning idea.
  - We start with a very large corpus of text (e.g. all of Wikipedia)
  - Every word is represented by a vector in $\mathbb{R}^n$ space (n~200 dimentions)
  - You have a model (e.g. a NN) that tries to predict the vector of a word (i.e. the central word) given the vectors of the words around it (i.e. its context). In probability terms, the NN models the probability $P(w_c \mid w_{c-3}, w_{c-2}, w_{c-1}, w_{c+1}, w_{c+2}, w_{c+3})$
  - Go through each central word - context pair in the corpus
  - In each iteration, modify the NN and vectors a little bit for words with similar contexts to have similar vectors
  - Repeat last 2 steps many times

Embeddings: word2vec

There are two versions of word2vec:

➢ CBOW:

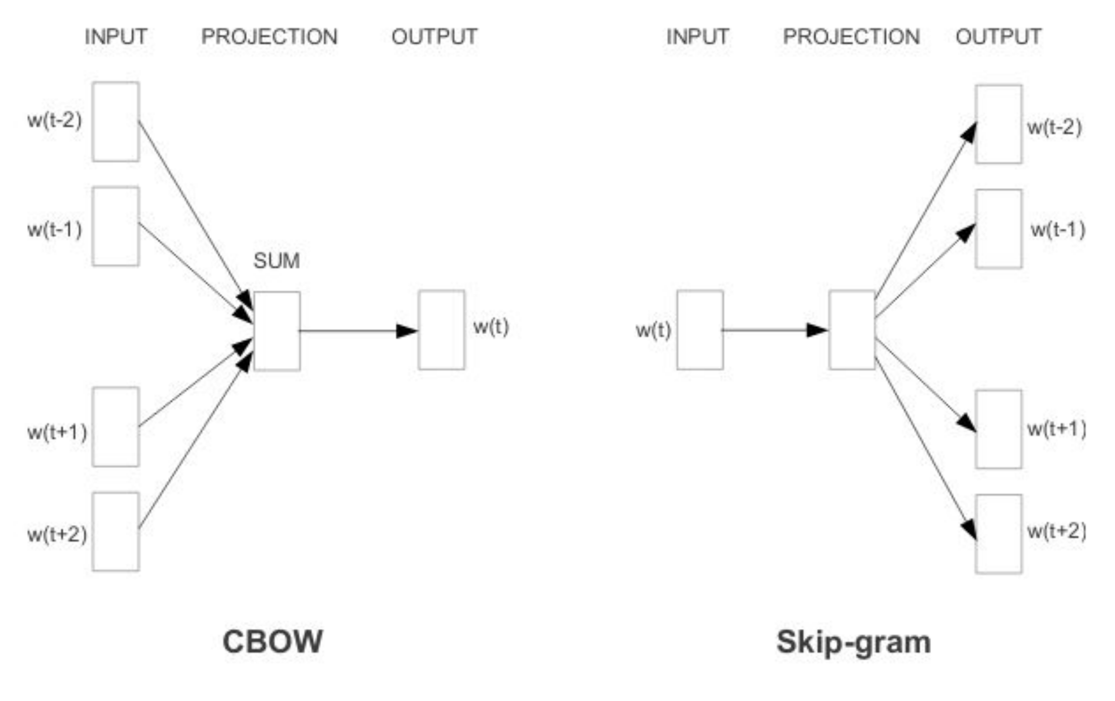
context → centre

A NN learns to model the central word P(w_c | w_{c-3} …)

➢ Skip-gram:

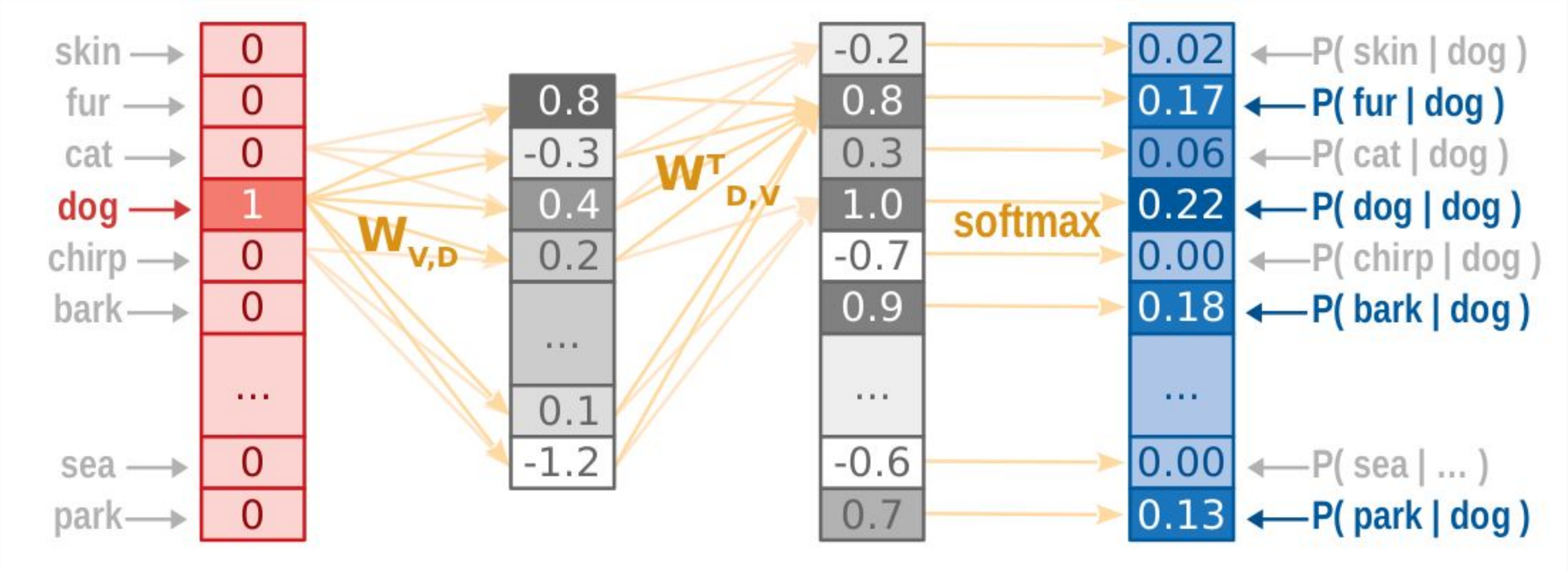
centre → context

A NN learns to model the context P(w_{c+i} | w_c)

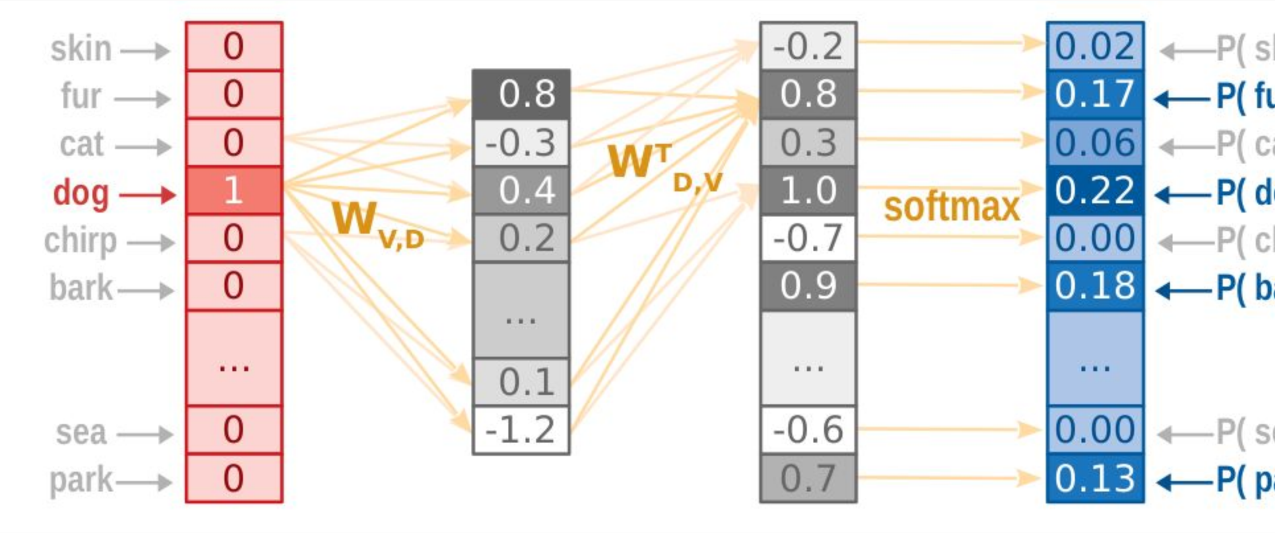


Word2vec: Skip-gram

- The NN uses a single hidden layer, a single weight matrix (W_{VD}), the transpose of this weight matrix (W^T_{VD}), and a single activation function (softmax)



Word2vec: Skip-gram



$$\underset{V}{softmax}\left([x]_V \times [W]_{VD} \times [W^T]_{DV}\right) = [y]_V$$

$$softmax(x) = \frac{\exp(x)}{\sum \exp(x)}$$

$$\frac{\exp\left([x]_V \times [W]_{VD} \times [W^T]_{DV}\right)}{\sum_V \exp\left([x]_V \times [W]_{VD} \times [W^T]_{DV}\right)} = [y]_V$$

$$\frac{\exp(\vec{w}_y \times \vec{w}_x)}{\sum_i \exp(\vec{w}_i \times \vec{w}_x)} = y(v)$$

$$\begin{bmatrix} 0 & 0 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ 10 & 12 & 19 \\ 11 & 18 & 25 \end{bmatrix} = \begin{bmatrix} 10 & 12 & 19 \end{bmatrix}$$

Word2vec: learning

- OK, awesome, then we use a NN to implement the model... but how do we learn the model? How do we find the values of W_{VD}? This is the matrix that contains all our vector embeddings
- We apply stochastic gradient descent (SGD) on a very very large corpus:
  - Go through each central word - context pair in the corpus
  - In each iteration, modify the NN and vectors a little bit for words with similar contexts to have similar vectors
  - Repeat last 2 steps many times
- But there is a problem: The denominator of $\frac{\exp\left(\underset{V}{[x]}\times\underset{VD}{[W]}\times\underset{DV}{[W^T]}\right)}{\sum_V \exp\left(\underset{V}{[x]}\times\underset{VD}{[W]}\times\underset{DV}{[W^T]}\right)} = \underset{V}{[y]}$ ..., or: $\frac{\exp(\vec{w}_y \times \vec{w}_x)}{\sum_i \exp(\vec{w}_i \times \vec{w}_x)} = y(v)$

Word2vec: learning

- The normalization factor is computationally too expensive:

$$\frac{\exp\left(\underset{V}{[x]}\times\underset{VD}{[W]}\times\underset{DV}{[W^T]}\right)}{\sum_{V}\exp\left(\underset{V}{[x]}\times\underset{VD}{[W]}\times\underset{DV}{[W^T]}\right)}=\underset{V}{[y]} \qquad \frac{\exp(\vec{w}_y\times\vec{w}_x)}{\sum_{i}\exp(\vec{w}_i\times\vec{w}_x)}=y(v)$$

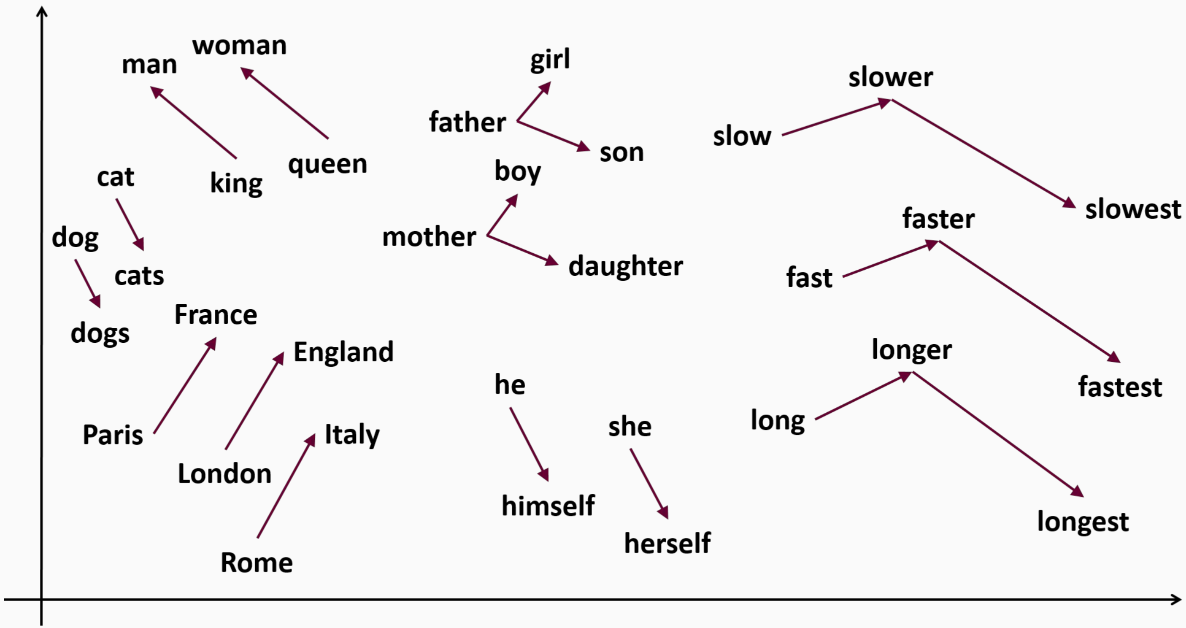
- To bypass this problem, rather than calculating the denominator exactly, we calculate and approximation of it by using “negative sampling”
- **Idea:** train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word).
- We take N negative samples (using word probabilities).
- Maximize probability that true word is predicted by the NN from its pair, minimize probability that noise word is predicted by the NN from its pair.
- “Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)

Word2vec

➢ The main end result of word2vec (and other embedding algorithms) is that words of similar meaning end up being place nearby to each other in $\mathbb{R}^D$ space

Word2vec

- A surprising side result, is that specific directions in $\mathbb{R}^D$ space also encode meaning
- Both properties are very useful for building further ML on top of the embeddings



Co-occurrence matrices

- **Idea:** But if all what word2vec is doing is extracting the meaning of words from their context, why don't we simply count how often words appear together?
- This is called a "co-occurrence matrix", and it also gets quite far capturing the meaning of words
- As the NN in word2vec, this very simple matrix of co-occurrence counts can be calculated:
  - Go through each central word - context pair in the corpus (context window length is commonly anything between 1 and 5)
  - In each iteration, update in the row of the count matrix corresponding to the central word by adding +1 in the columns corresponding to the context words
  - Repeat last 2 steps many times

Co-occurrence matrices

- Example corpus:
  - … after a few days the fur of the dog was unkept and dirty, and this spread ...
  - … you will soon realise that walks in the park are dog’s priority, and they will ...
  - … he was worried that his neighbour’s dog kept barking during all night ...
  - … the warden had a labrador of brown fur who kept chasing squirrels in ...
  - … breeders recommend to daily take your labrador to the park to tempter ....
  - … at the end of the day, a labrador barks less than other breeds, but he also …

Co-occurrence matrices

	...	cat	dog	labrador	fur	park	bark	...
...	...							
cat		23	4	0	12	0	0	
dog		4	28	23	13	22	28	
labrador		0	23	25	16	23	22	
fur		12	13	16	16	0	0	
park		0	22	23	0	21	3	
bark		0	28	22	0	3	16	
...								...

Co-occurrence matrices

- While word2vec was able to encode the meaning of the words in a matrix W_{VD} of size $\mathbf{V \times D}$, the co-occurrence C_{VV} matrix has size $\mathbf{V \times V}$
  - But there are some problems with this basic approach:
  - The co-occurrence matrix increase very fast (as V^2) with the size of the vocabulary
  - This requires a lot of storage
  - This requires a lot of computation (e.g. to calculate distance between words)
  - Rare words will have very few counts
  - ML algorithms built on top of the co-occurrence matrix have sparsity problems
- Three are some methods to ameliorate these problems

Co-occurrence matrices

- **Idea:** Store the most important information of $\mathbb{R}^V$ in a fixed small number of dimensions (usually 25-1000, as in word2vec), rather than in V dimensions.
- There are many methods to reduce dimensionality while preserving the most important information: Principal Component Analysis, Independent Component Analysis…
- One commonly used to reduce C_{VV} is Singular Value Decomposition (SVD):

$$\underset{mn}{[M]} = \underset{mm}{[U]} \times \underset{mn}{[\Sigma]} \times \underset{nn}{[V]}$$

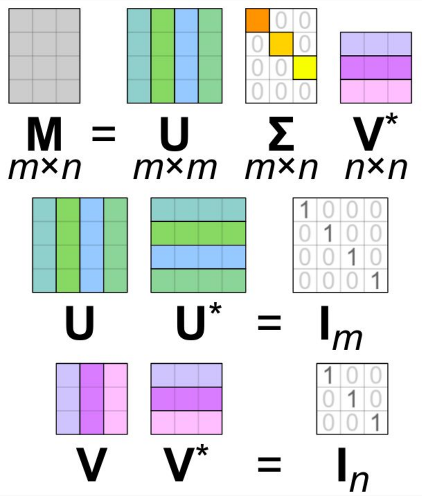
$$\underset{VV}{[C]} = \underset{VV}{[U]} \times \underset{VV}{[\Sigma]} \times \underset{VV}{[V]}$$

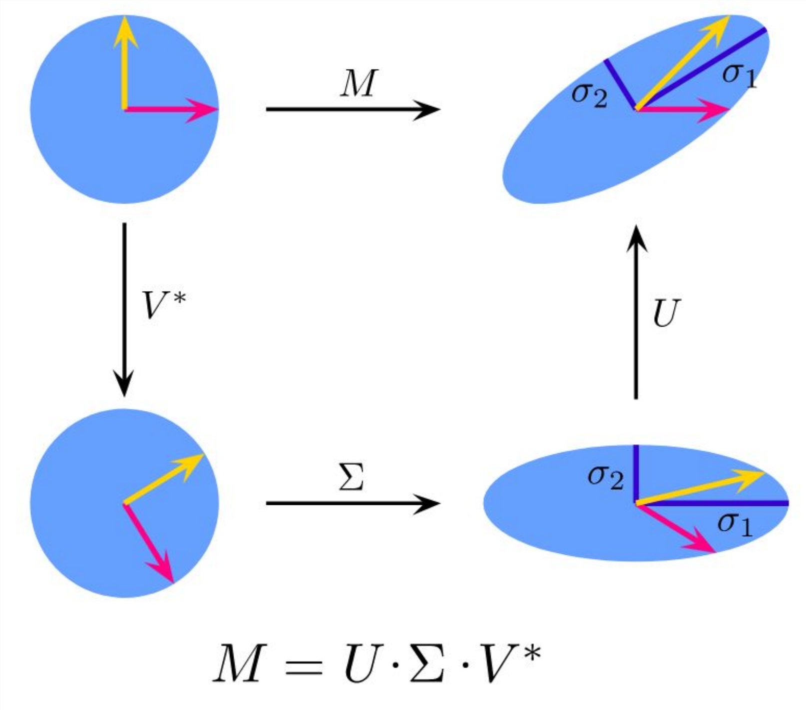
- Why do we use this very complicated method?

Co-occurrence matrices

➢ If M_{mn} were a linear transformation "M: $\mathbb{R}^V \rightarrow \mathbb{R}^V$" then:

- U = rotates the basis of M: $\mathbb{R}^V \rightarrow \mathbb{R}^V$
- Σ = rescales the basis
- V = rotates the basis again





Co-occurrence matrices

- And the magic comes from the fact that, if (1st) we decompose M_{mn} in its singular values

  $C_{vv} \rightarrow U_{vv} \times \Sigma_{vv} \times V_{vv}$

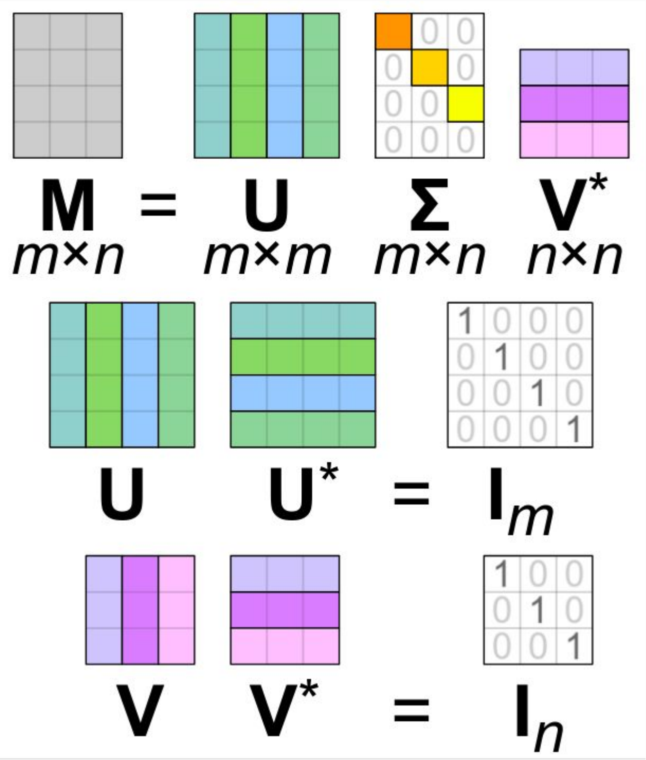
  (2nd) keep only the 'd' largest of such singular values

  $C_{vv} \rightarrow U_{vv} \times \Sigma_{vv} \times V_{vv} \rightarrow U_{vv} \times \Sigma_{vd} \times V_{dd}$

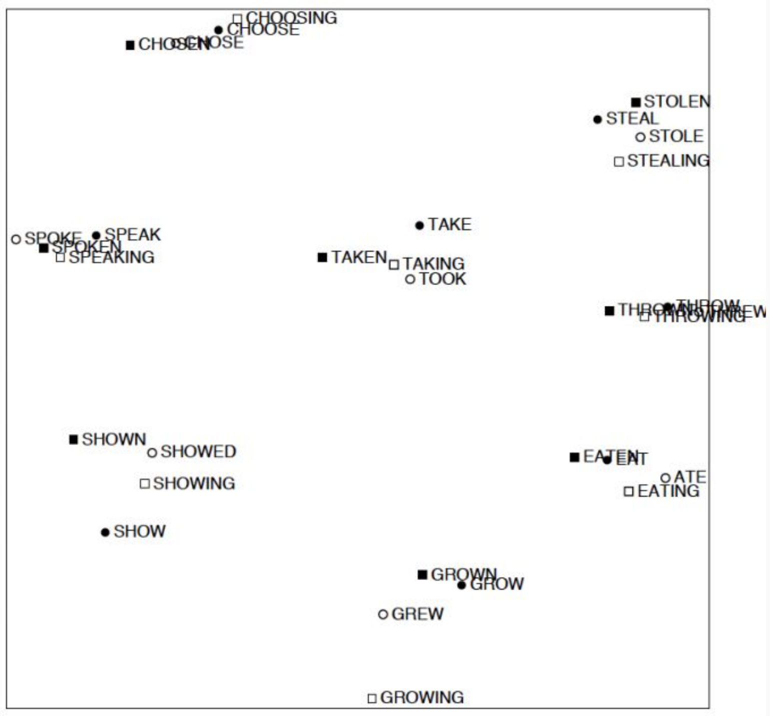
  and (3rd) recalculate M_{mm}

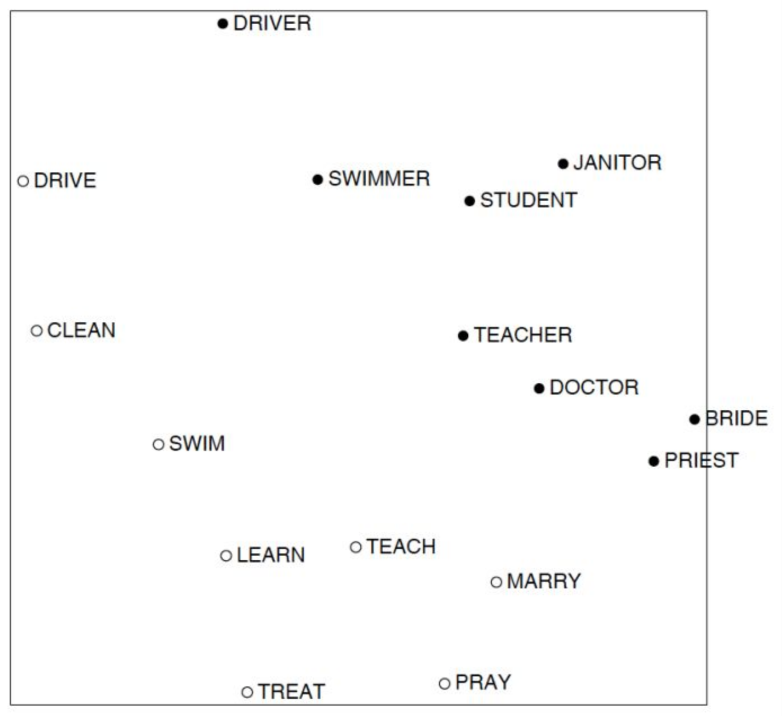
  $U_{vv} \times \Sigma_{vd} \times V_{dd} \rightarrow C_{vd}$

  We obtain the closest approximation of M_{mm} according to mean square error



Co-occurrence matrices





“An improved model of semantic similarity based on lexical co-occurrence” Rohde, 2005

Co-occurrence matrices

- ➢ Co-occurrence models:
  - LSA, HAL (Lund & Burges)
  - COALS, Hellinger-PCA (Rohde, Lebert & Collobert
- ➢ Pros and cons:
  - Fast training
  - Efficient use of statistics
  - Primarily used to capture word similarity
  - Disproportionate importance given to large counts

- ➢ NN based models:
  - Skip-gram, CBOW (Mikolov)
  - NNLM, HLBL, RNN (Bengio, Collobert & Weston, Huang, Mnih & Hinton)
- ➢ Pros and cons:
  - Not too fast training
  - Inefficient use of statistics
  - Gives improved performance on other tasks
  - Can capture complex patterns beyond word similarity

Glove

- Rather than word counts C_{vv}, people often used the probabilities of one word appearing in the context of another: $P(v_1|v_2)$
- Both word counts in C_{vv} and probabilities $P(v_1|v_2)$ depend very strongly on the frequency of words: frequent words will have much larger counts and probabilities
- The authors of Glove suggest that ratios of probabilities between words are much better suited to create good embeddings
- The authors of Glove introduce 2 further heuristic arguments:
  - The distance between words $d(v_1,v_2)$ should be a linear function
  - The Distance between words should be symmetric between context and central words. Namely $d(v_1,v_2)$ when v_1 is a central word and v_2 a context word should be the same than $d(v_1,v_2)$ when v_1 is a context word and v_2 a central word

Glove

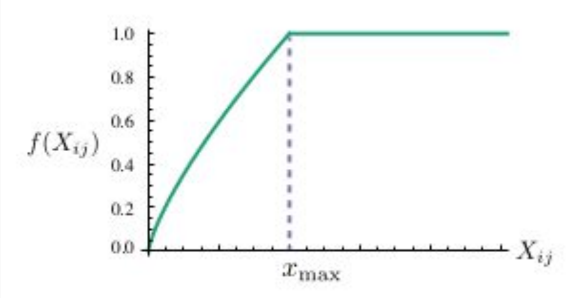
- The embeddings w_i that best fulfill those three rules, are those whose scalar product ($w^T_i w_k$) approximates ($\log(C_{vv})$) minus two constant values that depend only on the 2 words being multiplied (b_i & b_j):

$$w^T_i w_j + b_i + b_j \sim \log(\ [C_{vv}](i,j)\ )$$

- You can obtain these embeddings w_i for all words by minimising the error function:

$$J = \sum_{i,j=1}^{V} ramp(\ \underset{V,V}{[C]}(i,j)\ )\left(\underset{E}{[e_i]} \cdot \underset{E}{[e_j]} + b_i + b_j - log(\ \underset{V,V}{[C]}(i,j)\ )\right)^2$$

- The function ramp(...) is defined heuristically →



Glove

➢ You can obtain these embeddings wi for all words by minimising the error function:

$$J = \sum_{i,j=1}^{V} ramp(\underset{V,V}{[C]}(i,j))\left(\underset{E}{[e_i]} \cdot \underset{E}{[e_j]} + b_i + b_j - log(\underset{V,V}{[C]}(i,j))\right)^2$$

➢ The advantages of Glove:

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

Evaluating quality of embeddings

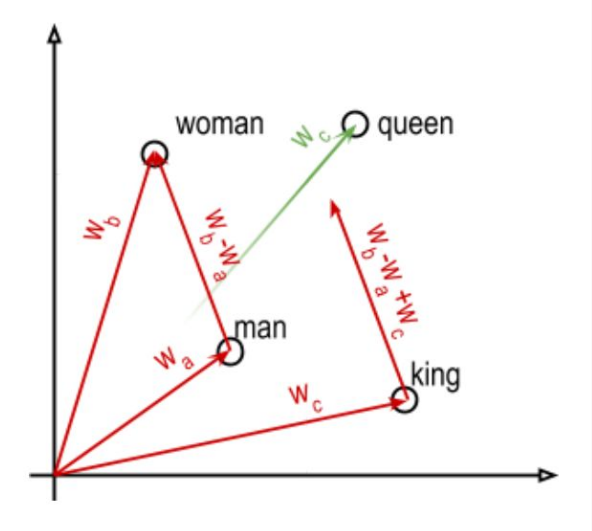
- **Intrinsic methods**: Measure some statistical property of the embeddings that should correlate with quality (e.g. similar words should be close to each other)
  - Fast to compute
  - Helps to understand the system
  - The method does not fully ensure that the embeddings are going to perform well when sent to another real world task
- **Extrinsic methods**: Use the embeddings in a real NN and on a real task to evaluate embeddings (e.g. named entity recognition)
  - Slow to compute
  - Unclear what part of the performance on the real task comes from the embedding and which part comes from the rest of the NN… and which from the embeddings - NN interaction

Evaluating quality of embeddings

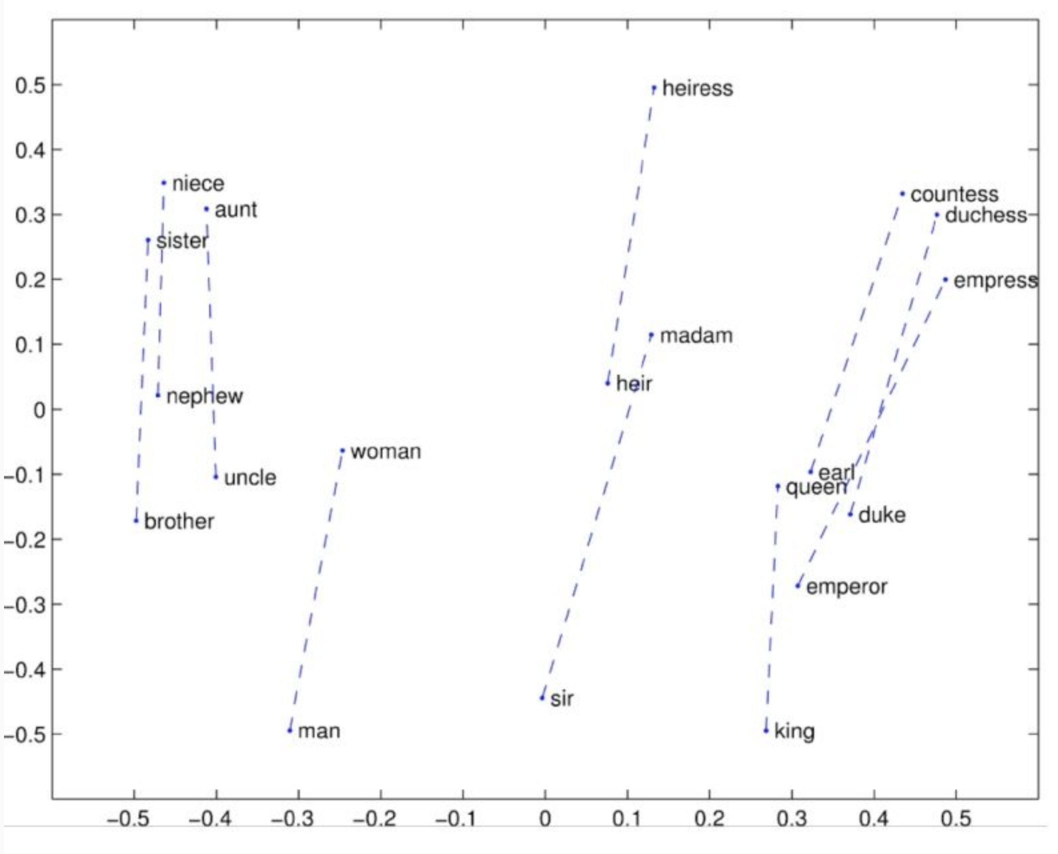
Intrinsic methods:

- ➢ Word Vector Analogies:
  - *a is to b* as *c is to x*
  - *man is to woman* as *king is to x*
- ➢ Evaluates word vectors by how well their difference vector (w_b-w_a) captures meaning consistently when moved to another word (w_c)
- ➢ Discards the input words from the search
- ➢ Problem: What if the information is there but it is nonlinear

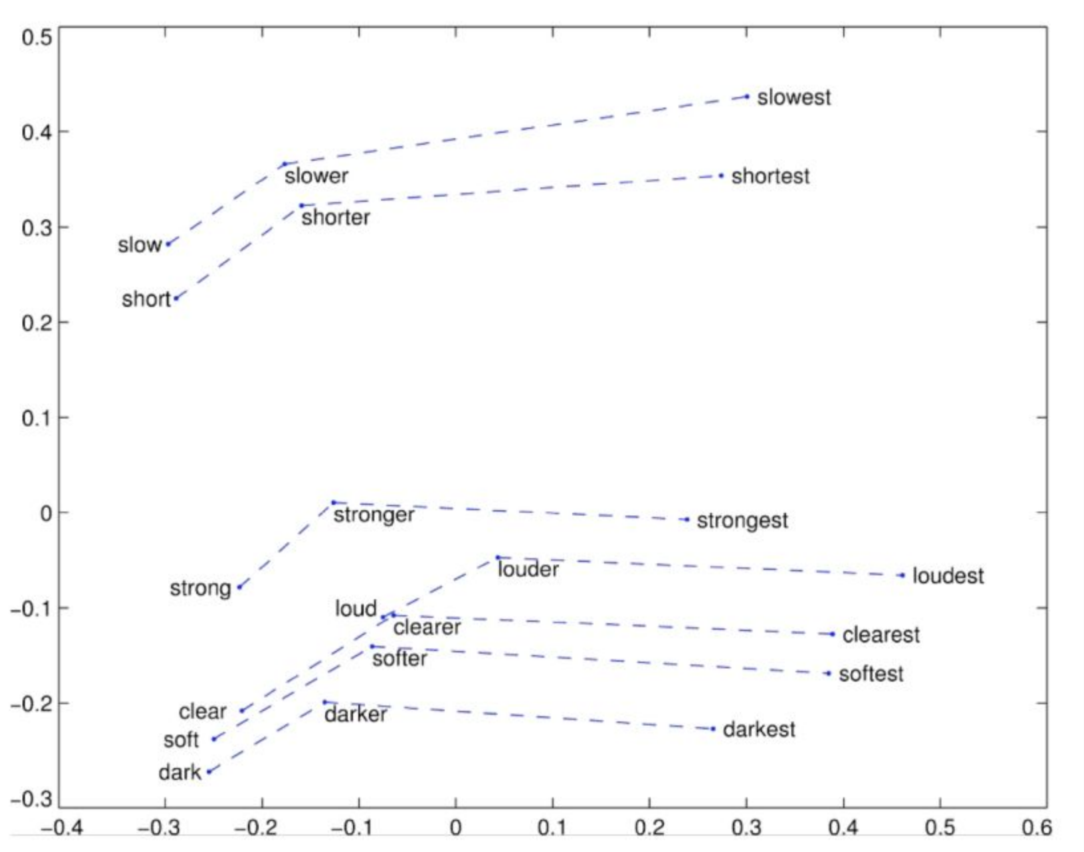
$$x = \underset{d}{argmax} \frac{(w_b - w_a + w_c)^T w_d}{||w_b - w_a + w_c||}$$



Evaluating quality of embeddings



Evaluating quality of embeddings



Evaluating quality of embeddings

Intrinsic methods:

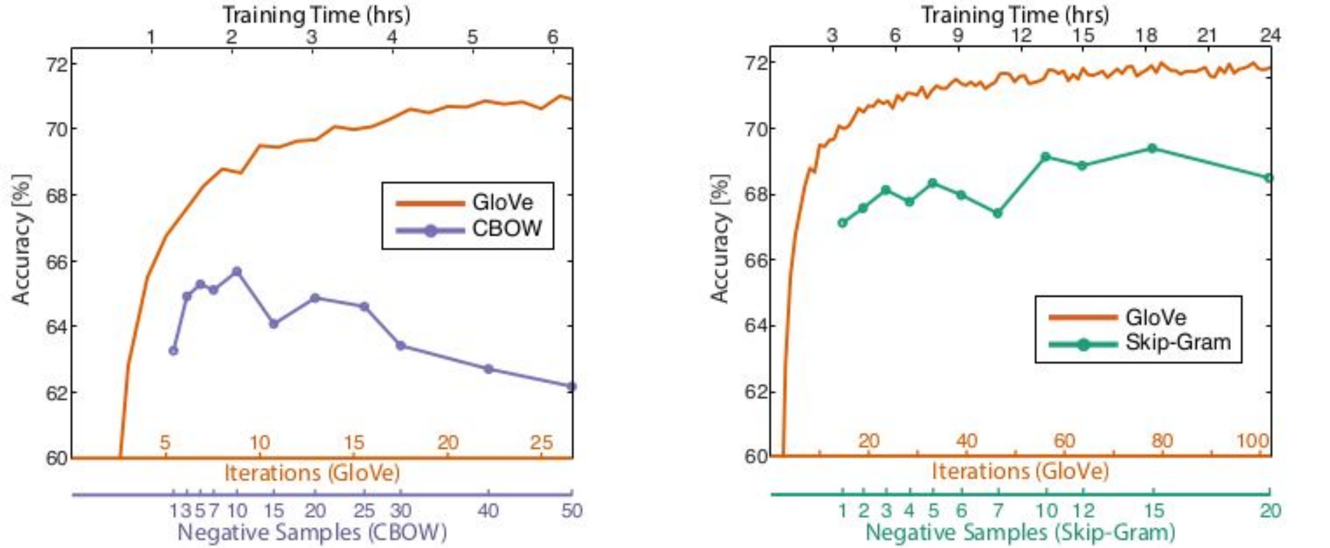
➢ There are datasets available to run intrinsic evaluation:

- https://github.com/nicholas-leonard/word2vec/questions-words.txt → Word relationships
- https://github.com/nicholas-leonard/word2vec/blob/master/questions-phrases.txt → phrase relationships
- http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/ → Word similarities

Athens Greece Bangkok Thailand
Fresno California Anchorage Alaska
free freely usual usually
clear unclear certain uncertain

d(cup, coffee) → 6.6
d(cup, article) → 2.4
d(Noon, string) → 0.5
d(Midday, noon) → 0.3

Evaluating quality of embeddings



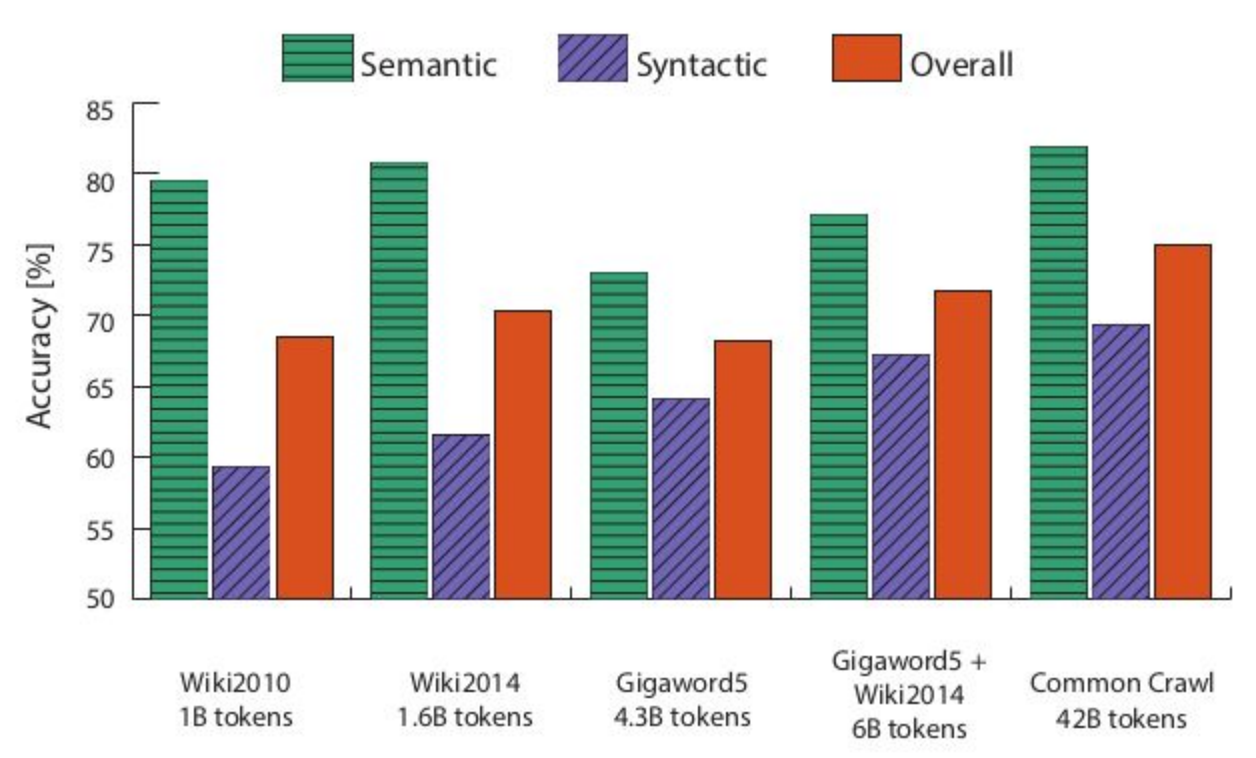
(a) GloVe vs CBOW

(b) GloVe vs Skip-Gram

Figure 4: Overall accuracy on the word analogy task as a function of training time, which is governed by the number of iterations for GloVe and by the number of negative samples for CBOW (a) and skip-gram (b). In all cases, we train 300-dimensional vectors on the same 6B token corpus (Wikipedia 2014 + Gigaword 5) with the same 400,000 word vocabulary, and use a symmetric context window of size 10.

→ Glove: Global vectors for word representation.
Pennington et al. EMNLP. 2014

Evaluating quality of embeddings



→ Glove: Global vectors for word representation. Pennington et al. EMNLP. 2014

Evaluating quality of embeddings

Extrinsic methods:

- ➢ Typical benchmark tasks:
  - Named Entity Recognition
  - Parts Of Speech tagging
  - Sentiment analysis
  - Translation
  - … basically, anything meaningful

Table 4: F1 score on NER task with 50d vectors. *Discrete* is the baseline without word vectors. We use publicly-available vectors for HPCA, HSMN, and CW. See text for details.

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

→ Glove: Global vectors for word representation. Pennington et al. EMNLP. 2014

Course structure

- **Introduction:** What is NLP. Why it is hard. Why NNs work well ← Lecture 9 (NLP 1)
- **Word representation:** How to represent the meaning of individual words
  - Old technology: One-hot representations, synsets ← Lecture 9 (NLP 1)
  - Embeddings: First trick that boosted the performance of NNs in NLP ← Lecture 9 (NLP 1)
    - Word2vec: Single layer NN. CBOW and skip-gram ← Lecture 10 (NLP 2)
    - Co-occurrence matrices: Basic counts and SVD improvement ← Lecture 10 (NLP 2)
    - Glove: Combining word2vec and co-occurrence matrices idea ← Lecture 10 (NLP 2)
    - Evaluating performance of embeddings ← Lecture 10 (NLP 2)
- **Named Entity Recognition (NER):** How to find words of specific meaning within text
  - Multilayer NNs: Margin loss. Forward- and back-propagation ← Lecture 11 (NLP 3)
  - Better loss functions: margin loss, regularisation ← Lecture 11 (NLP 3)
  - Better initializations: uniform, xavier ← Lecture 11 (NLP 3)
  - Better optimizers: Adagrad, RMSprop, Adam... ← Lecture 11 (NLP 3)

Course structure

- ➢ **Language modelling:** How to represent the meaning of full pieces of text
  - Old technology: N-grams ← Lecture 12 (NLP 4)
  - Recursive NNs language models (RNNs) ← Lecture 12 (NLP 4)
  - Evaluating performance of language models ← Lecture 13 (NLP 5)
  - Vanishing gradients: Problem. Gradient clipping ← Lecture 13 (NLP 5)
  - Improved RNNs: LSTM, GRU ← Lecture 13 (NLP 5)

- ➢ **Machine translation:** How to translate text
  - Old technology: Georgetown–IBM experiment and ALPAC report ← Lecture 16 (NLP 6)
  - Seq2seq: Greedy decoding, encoder-decoder, beam search ← Lecture 16 (NLP 6)
  - Attention: Simple attention, transformers, reformers ← Lecture 16 (NLP 6)
  - Evaluating performance: BLEU ← Lecture 16 (NLP 6)

Literature

➢ Papers =

- GloVe: Global vectors for word representation, Pennington et al., 2014. http://nlp.stanford.edu/pubs/glove.pdf
- Improving distributional similarity with lessons learned from word embeddings", Levy et al., 2015. http://www.aclweb.org/anthology/Q15-1016
- Evaluation methods for unsupervised word embeddings, Schnabel et al., 2015. http://www.aclweb.org/anthology/D15-1036
- A latent variable model approach to PMI-based word embeddings, Arora et al., 2016. http://aclweb.org/anthology/Q16-1028
- Linear algebraic structure of word senses, with applications to polysemy, Arora et al., 2017, https://transacl.org/ojs/index.php/tacl/article/viewFile/1346/320
- "On the dimensionality of word embedding", Yin et al., 2018. https://arxiv.org/pdf/1812.04224