

Word vectors

Radoslav Neychev

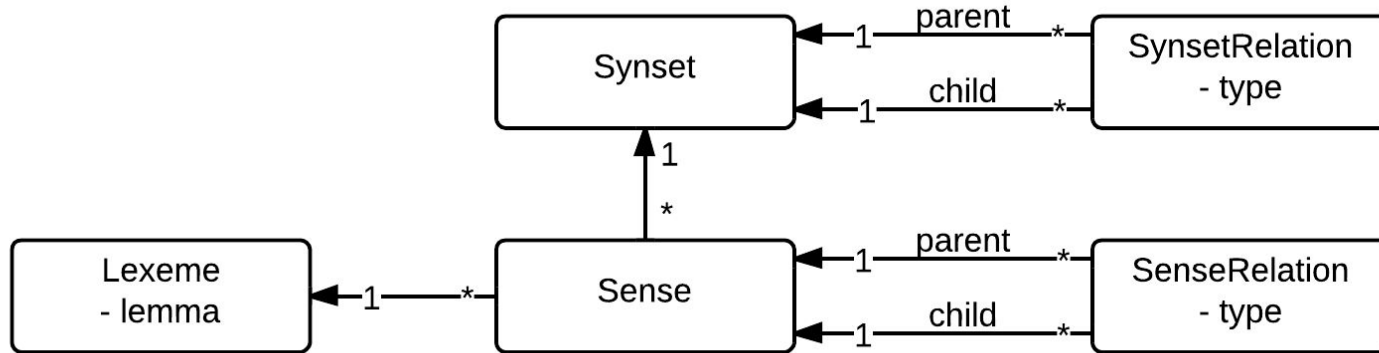
21.10.2019, MIPT
Moscow, Russia

Outline

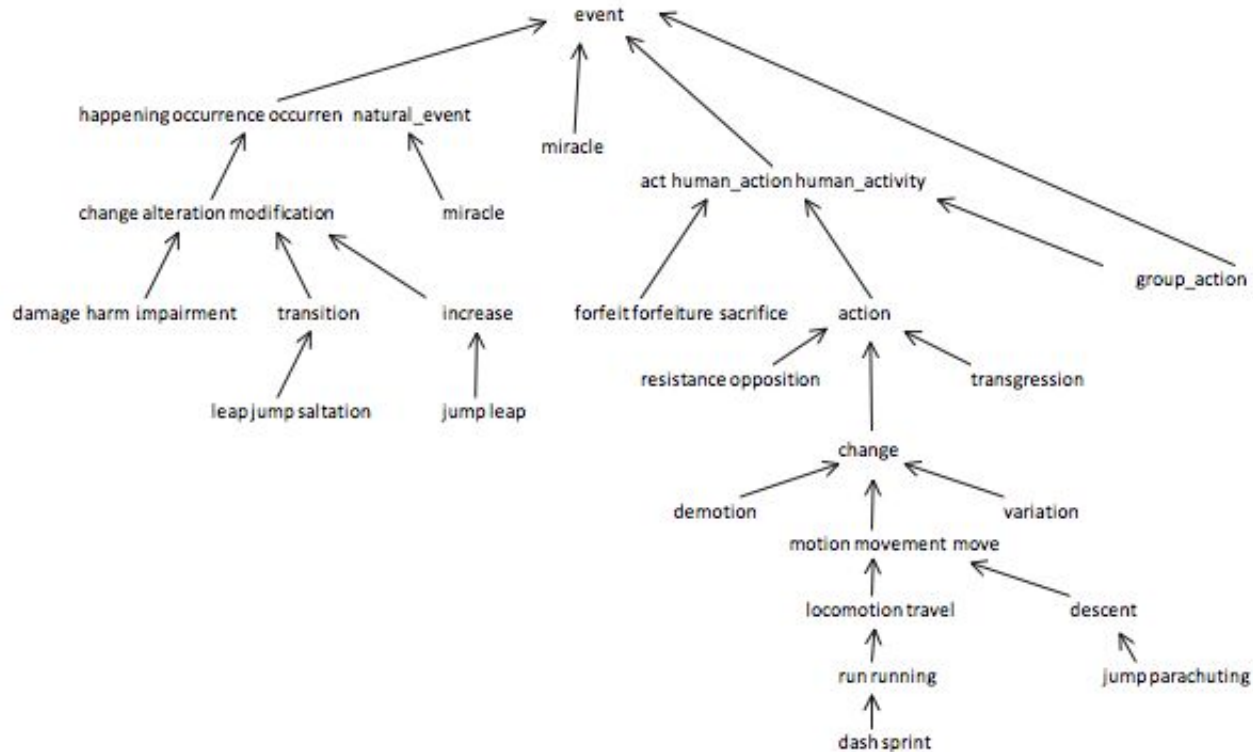
1. Discrete representations.
2. Matrix of co-occurrence.
3. SVD.
4. Embeddings (GloVe, word2vec).
5. Examples.

How to represent text in a computer?

Use a taxonomy like WordNet that has hypernyms (is-a) relationships and synonym sets



How to represent text in a computer: WordNet



Discrete representations: problems

- Missing new words
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity

Discrete representations: one-hot encoding

"a"	"abbreviations"		"zoology"	"zoom"
1	0		0	0
0	1		0	1
0	0		0	0
⋮	⋮	⋮	⋮	⋮
0	0		0	0
0	0		1	0
0	0		0	1

$$s(Q, D) = \sum_w tf_{w,Q} \cdot \frac{tf_{w,D}}{tf_{w,D} + \frac{k|D|}{avg|D|}} \cdot \log \frac{|C|}{df_w}$$

If word is repeated in the query, it's probably important
 Repetitions of query words in the document → good
 Rare words more important
 The more query words we match, the better.
 Σ over the vocabulary
 Repetitions of same word less important than different words.
 Except in very long documents

TF - term frequency

IDF - Inversed Document Frequency

TF-IDF: make it simple

$$\text{tf}(\text{"this"}, d_1) = \frac{1}{5} = 0.2$$

$$\text{tf}(\text{"this"}, d_2) = \frac{1}{7} \approx 0.14$$

$$\text{idf}(\text{"this"}, D) = \log\left(\frac{2}{2}\right) = 0$$



$$\text{tfidf}(\text{"this"}, d_1, D) = 0.2 \times 0 = 0$$

$$\text{tfidf}(\text{"this"}, d_2, D) = 0.14 \times 0 = 0$$



Word 'this' is not very
informative

Document 1		Document 2	
Term	Term Count	Term	Term Count
this	1	this	1
is	1	is	1
a	2	another	2
sample	1	example	3

One of the most successful ideas of statistical NLP:

“You shall know a word by the company it keeps”

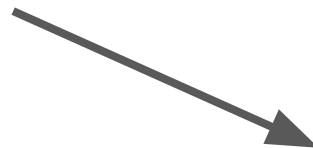
(J. R. Firth 1957: 11)

Words cooccurrences

Finding N-grams in a text



Word-document
cooccurrence matrix

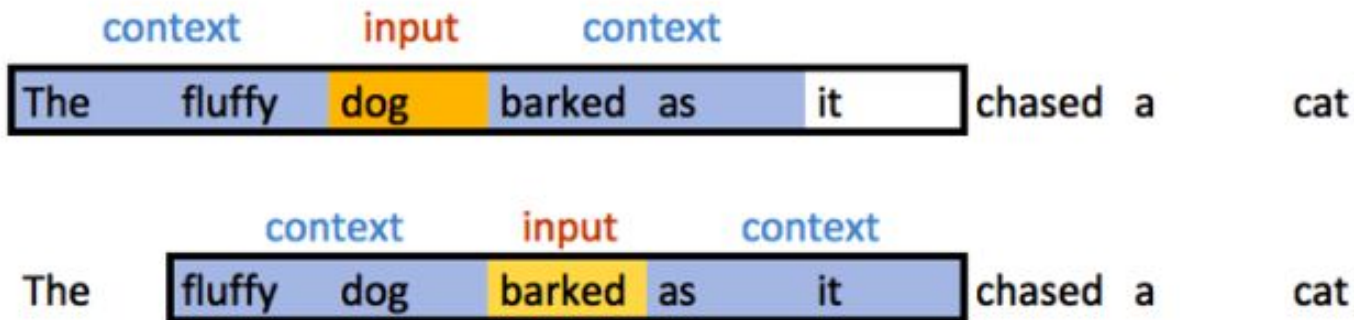


Window around
each word

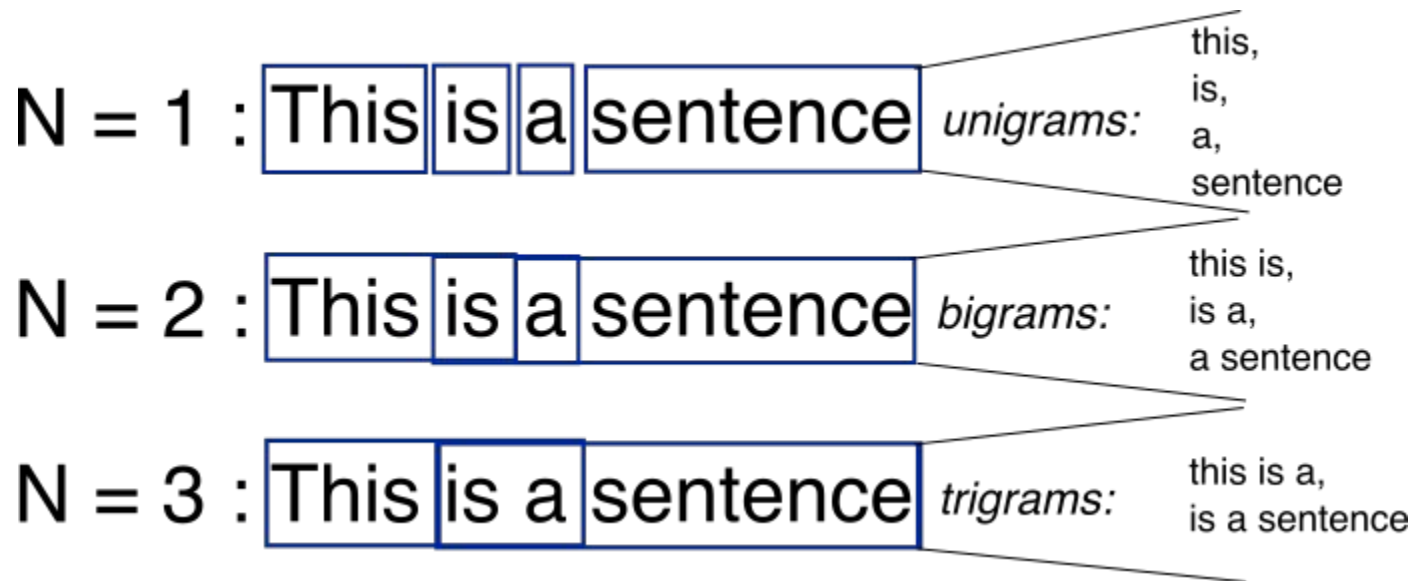
Word-document cooccurrence matrix

$$X = \begin{matrix} & \begin{matrix} I & like & enjoy & deep & learning & NLP & flying & . \end{matrix} \\ \begin{matrix} I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{matrix} & \left[\begin{array}{cccccccc} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array} \right] \end{matrix}$$

Words cooccurrences: sliding window



Words cooccurrences: n-grams



Cooccurrence vectors: problems

- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues



Models are less robust

Reducing dimensionality: SVD of cooccurrence matrix

Item x subject matrix
(ISM)


	S1	S2	S3	S4	S5
dog	1	1	1	1	1
cat	1	1	0	1	0
cow	0	0	1	0	1
lion	0	0	1	1	0
tiger	1	1	0	0	1

Singular decomposition
analysis (SVD)

$$C_{m \times n} = U_{m \times r} \times \Sigma_{r \times r} \times V'_{r \times n}$$

Item vectors Singular values Subject vectors

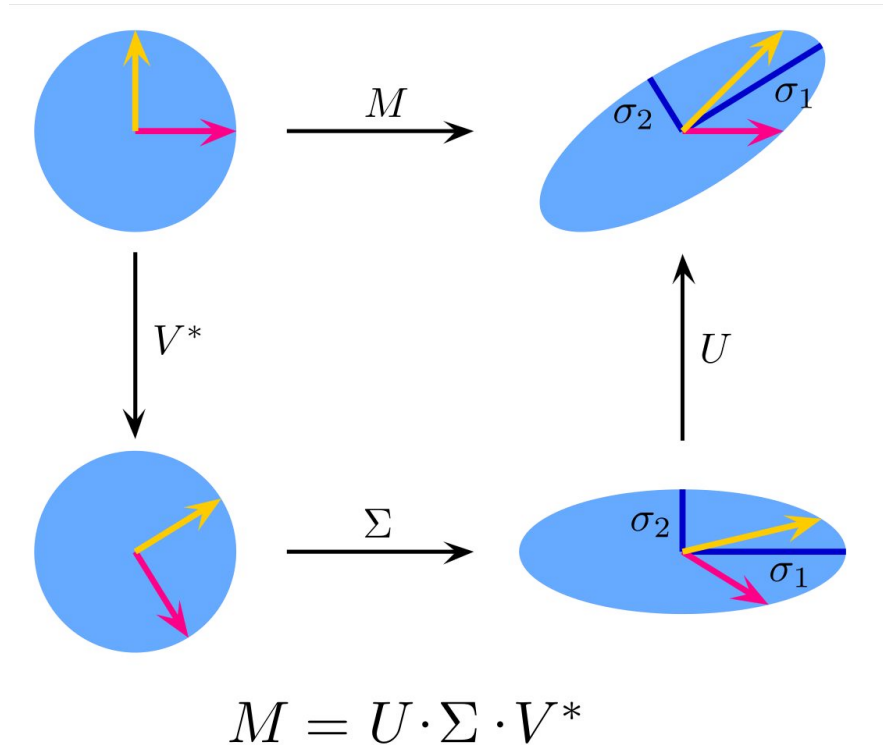
Reducing dimensions
from r to k



$$\tilde{C}_{m \times n} = U_{m \times k} \times \Sigma_{k \times k} \times V'_{k \times n}$$

The diagram shows the reduction of dimensions from r to k . The matrix $U_{m \times r}$ is reduced to $U_{m \times k}$, the matrix $\Sigma_{r \times r}$ is reduced to $\Sigma_{k \times k}$, and the matrix $V'_{r \times n}$ is reduced to $V'_{k \times n}$. The reduced matrices are shown with dashed lines and pink highlights to indicate the selected columns and rows.

SVD: intuition

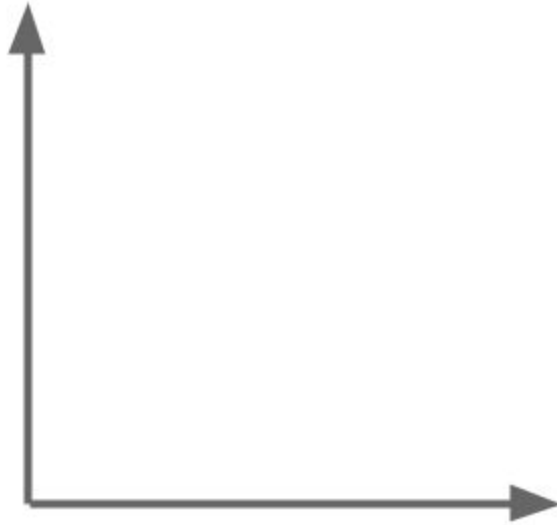


SVD: problems

- Computational cost scales quadratically for $n \times m$ matrix:
 $O(mn^2)$ flops (when $n < m$)
- Hard to incorporate new words or documents
- Different learning regime than other DL models

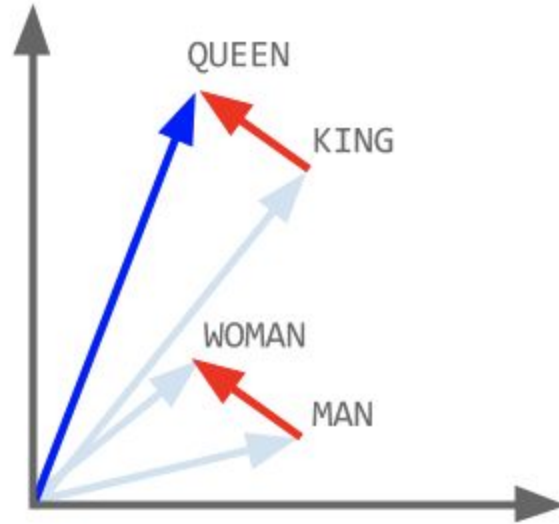
Embeddings: intuition

What is king - man + woman?



Embeddings: intuition

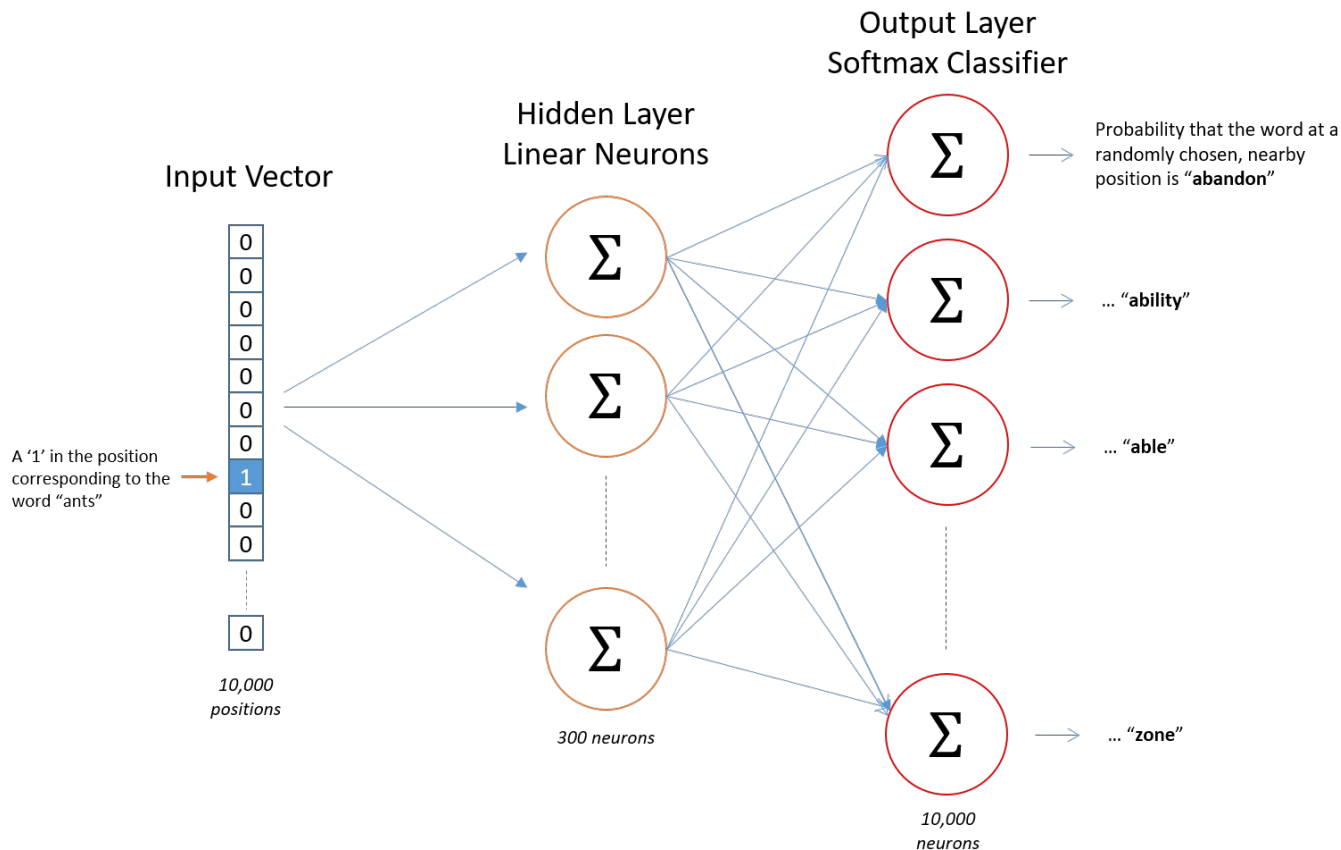
So $\text{king} - \text{man} + \text{woman} = \text{queen!}$



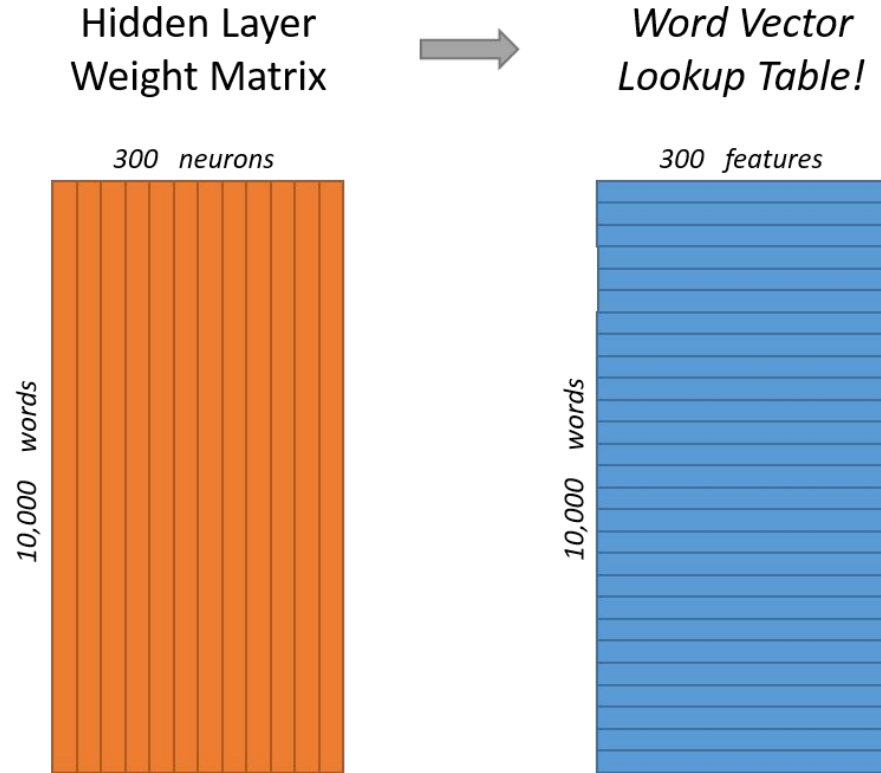
Embeddings: word2vec

Source Text	Training Samples					
<table><tr><td>The</td><td>quick</td><td>brown</td></tr></table> fox jumps over the lazy dog. ➡	The	quick	brown	(the, quick) (the, brown)		
The	quick	brown				
The <table><tr><td>quick</td><td>brown</td><td>fox</td></tr></table> jumps over the lazy dog. ➡	quick	brown	fox	(quick, the) (quick, brown) (quick, fox)		
quick	brown	fox				
The quick <table><tr><td>brown</td><td>fox</td><td>jumps</td></tr></table> over the lazy dog. ➡	brown	fox	jumps	(brown, the) (brown, quick) (brown, fox) (brown, jumps)		
brown	fox	jumps				
The <table><tr><td>quick</td><td>brown</td><td>fox</td><td>jumps</td><td>over</td></tr></table> the lazy dog. ➡	quick	brown	fox	jumps	over	(fox, quick) (fox, brown) (fox, jumps) (fox, over)
quick	brown	fox	jumps	over		

Embeddings: word2vec



Embeddings: word2vec



Embeddings: word2vec

- Word vectors with 300 components
- Vocabulary of 10,000 words.
- Weight matrix with $300 \times 10,000 = 3$ million weights each!

Training is too long and computationally expensive

How to fix this?

Embeddings: word2vec

Basic approaches:

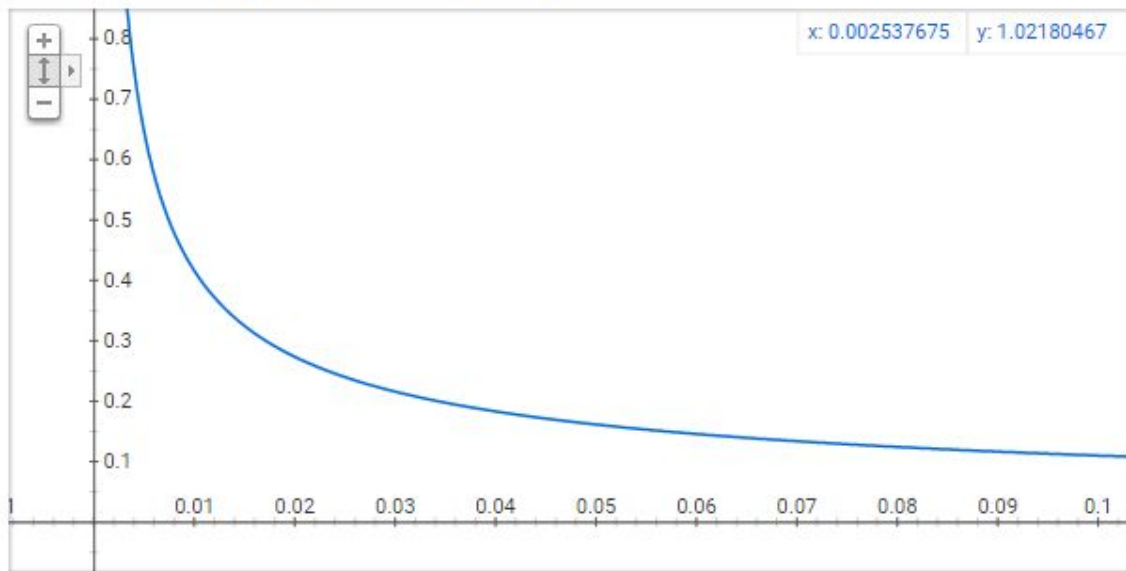
1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage of the model’s weights.

Embeddings: word2vec

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the whole text,

Graph for $(\sqrt{x/0.001}+1)*0.001/x$



$P(w_i)$ is the probability of *keeping* the word:

$$P(w_i) = \left(\sqrt{\frac{z(w_i)}{0.001}} + 1 \right) \cdot \frac{0.001}{z(w_i)}$$

Source: <http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/>

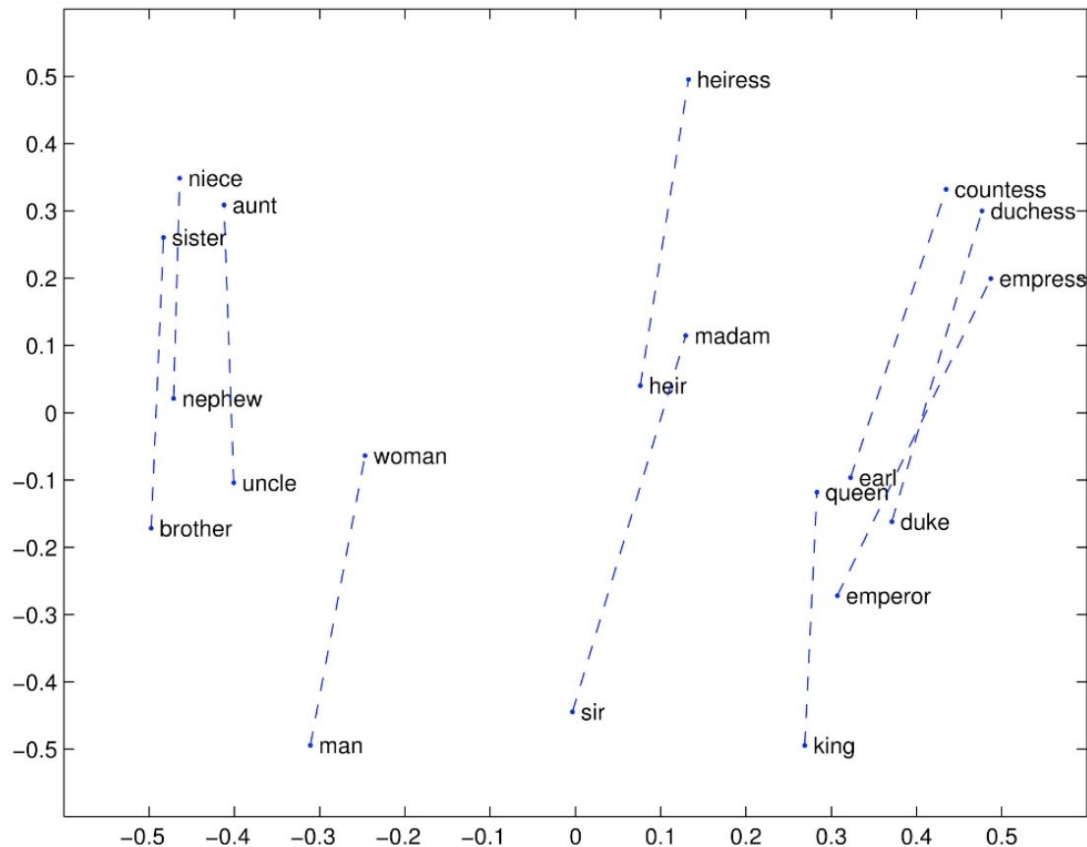
Embeddings: negative sampling

Negative Sampling idea: only few words error is computed. All other words has zero error, so no updates by the backprop mechanism.

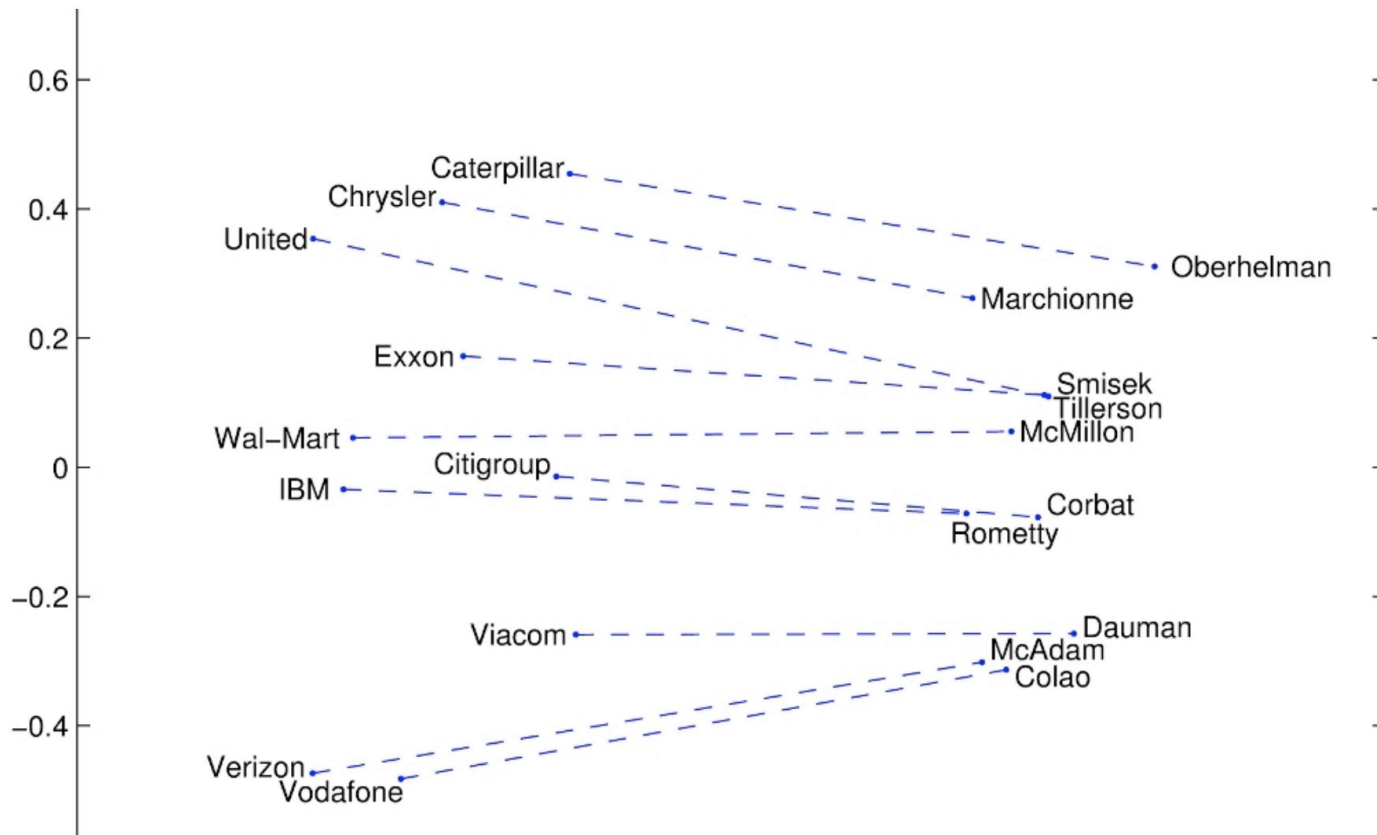
More frequent words are selected to be negative samples more often. The probability for a selecting a word is just it's weight divided by the sum of weights for all words.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^n (f(w_j)^{3/4})}$$

GloVe Visualizations



GloVe Visualizations: Company - CEO



Word vectors are simply vectors of numbers that represent the meaning of a word

Approaches:

- One-hot encoding
- Bag-of-words models
- Counts of word / context co-occurrences
- TF-IDF
- Predictions of context given word (skip-gram neural network models, e.g. word2vec)