Word embeddings

Radoslav Neychev, neychev@bigdatateam.org
ML Instructor, BigDataTeam
Assistant Professor, MIPT
LinkedIn profile

09.12.2020, online

MEGAFON

BIGDATA TEAM

Outline

- NLP: introduction
- Text Preprocessing
- Feature Extraction: classical approach
  - Bag-of-Words
  - Bag-of-Ngramms
  - TF-IDF
- Word Embeddings

Natural Language Processing:

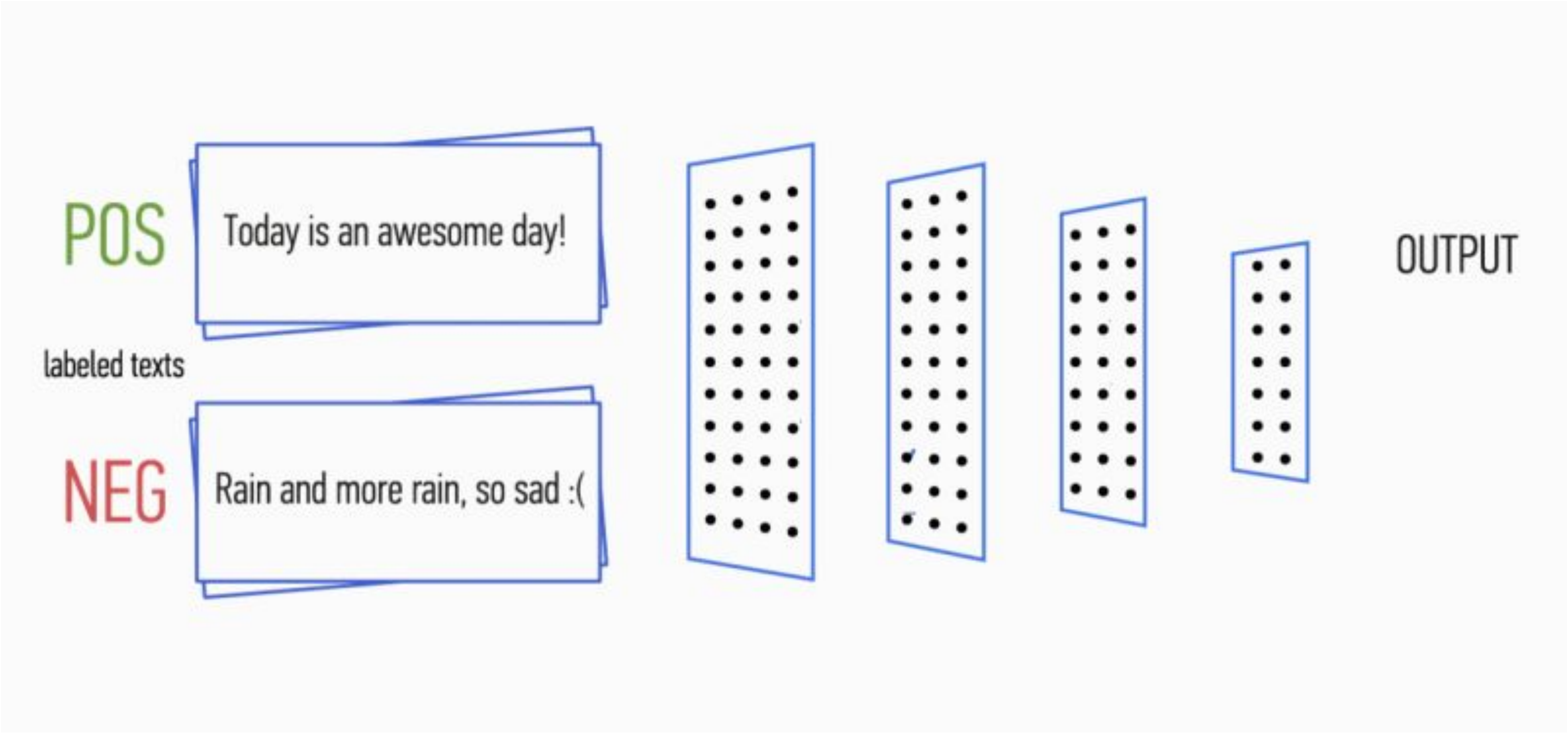
Introduction

Popular NLP tasks

- Sentiment analysis
- Spam filtering
- Fake news detection
- Topic prediction
- #hashtag prediction

Text classification tasks

Example: sentiment analysis

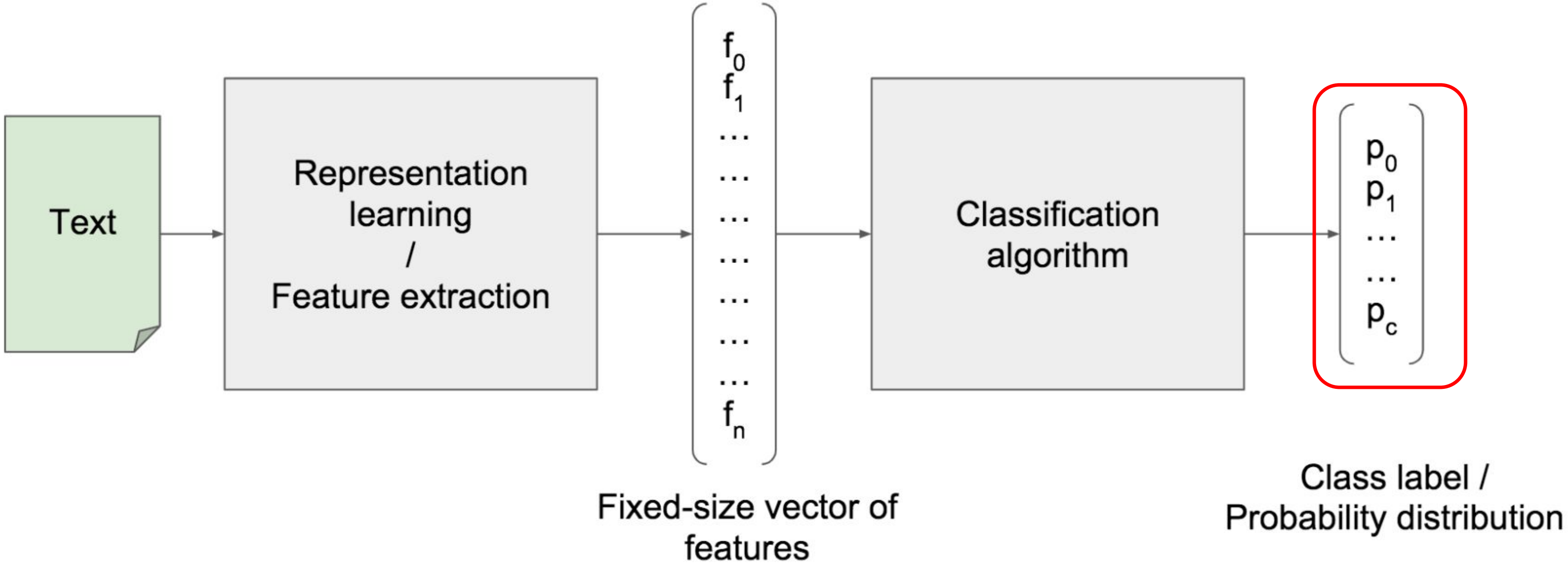


Source: habr.com

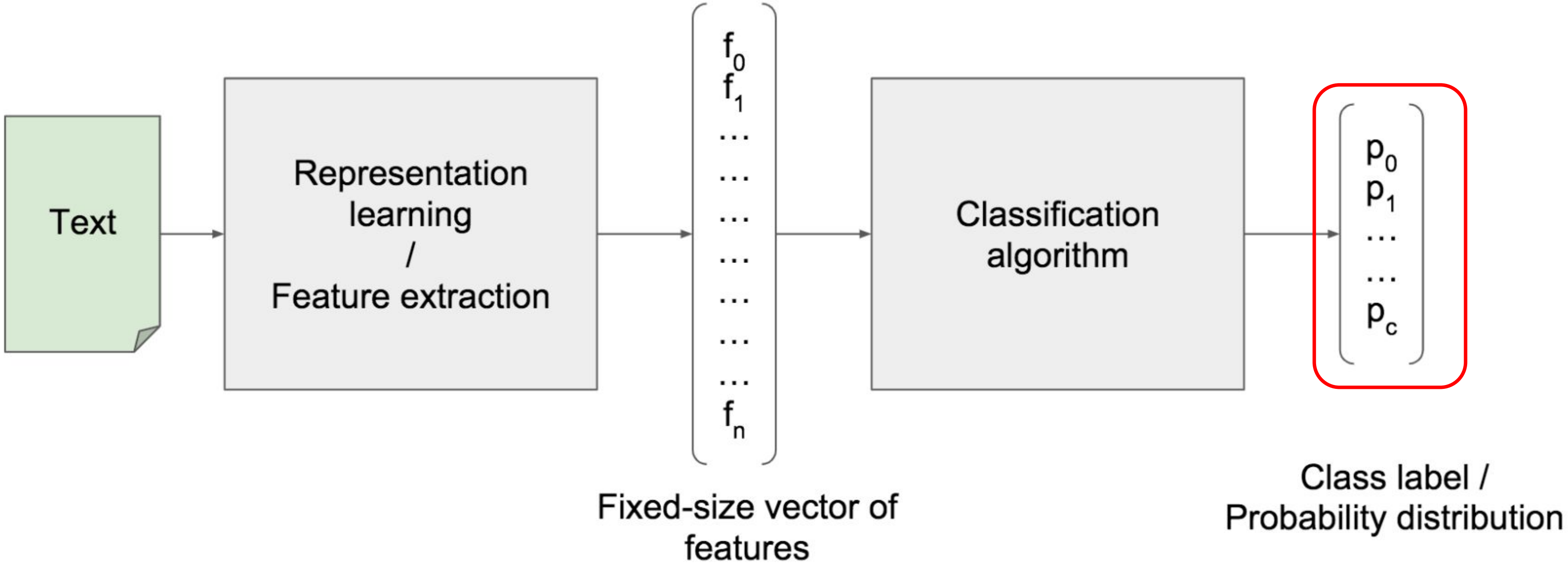
Text label kinds

- Discrete labels:
  - Binary
    - spam filtering, sentiment analysis
  - Multi-class
    - categorization of items by its description
  - Multi-label
    - #hashtag prediction
- Continuous labels:
  - Predict product price by its description

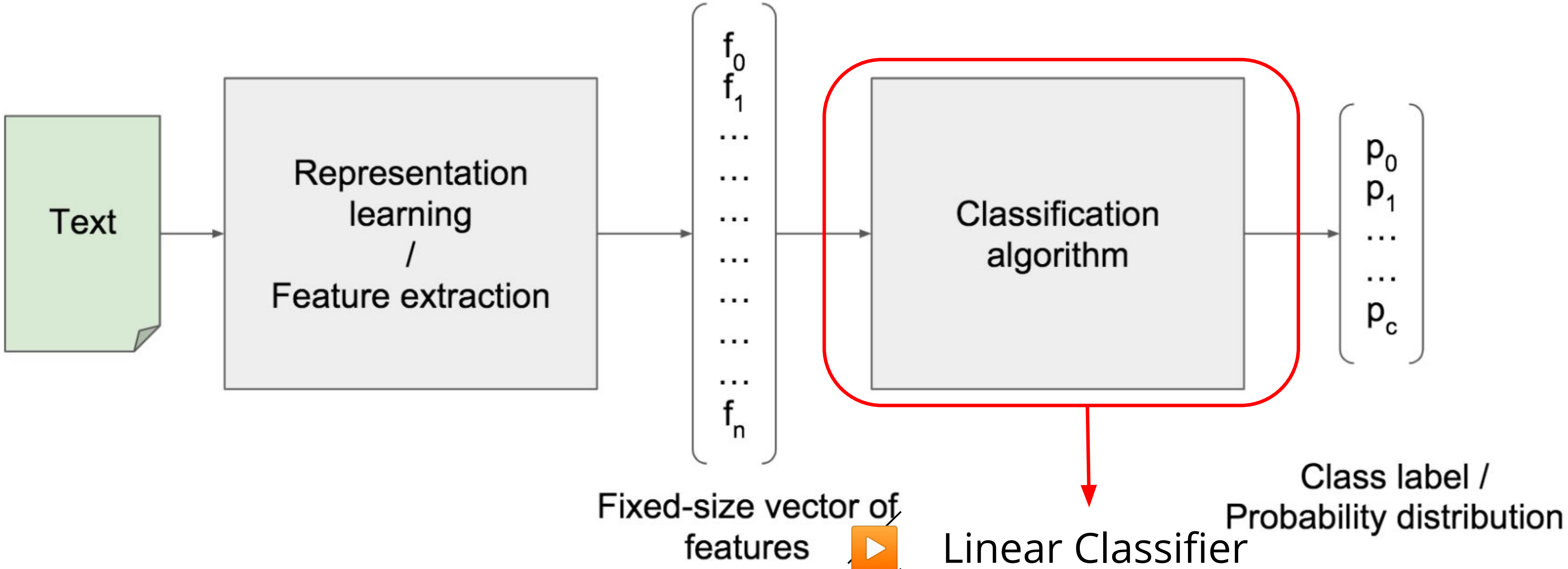
Text classification in general



Text classification in general

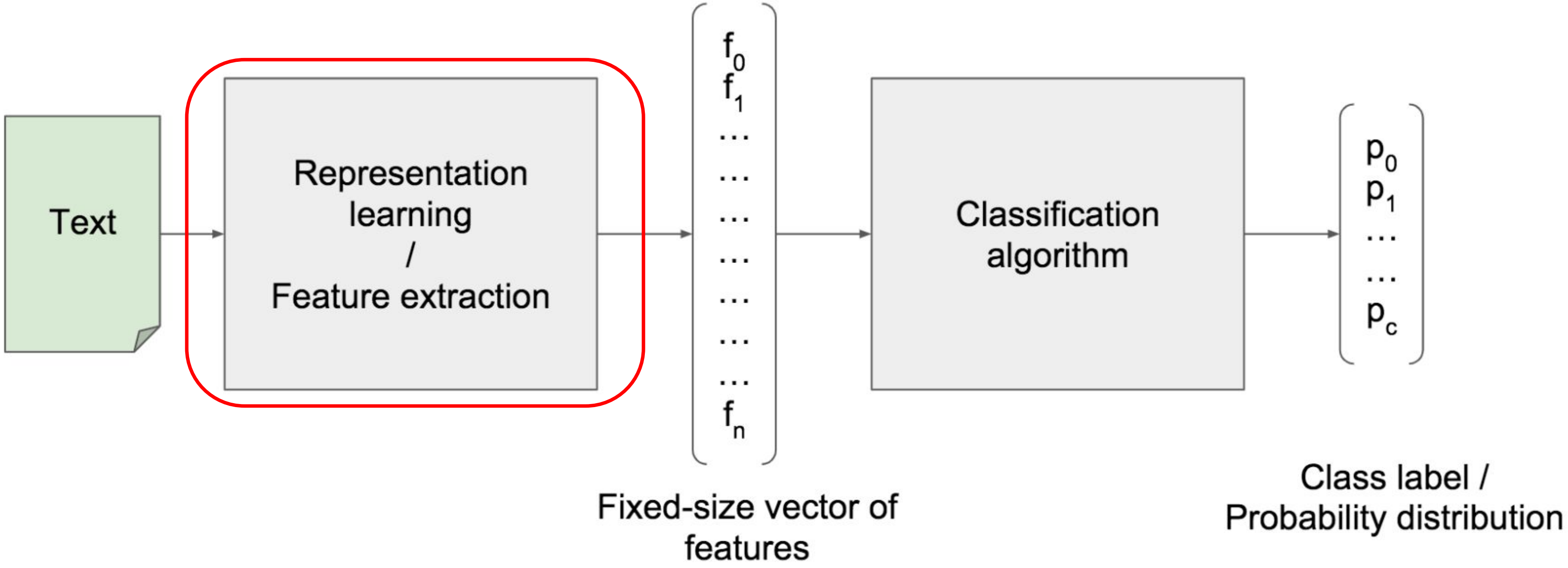


Text classification in general



- Linear Classifier
- Logistic Regression
- etc.

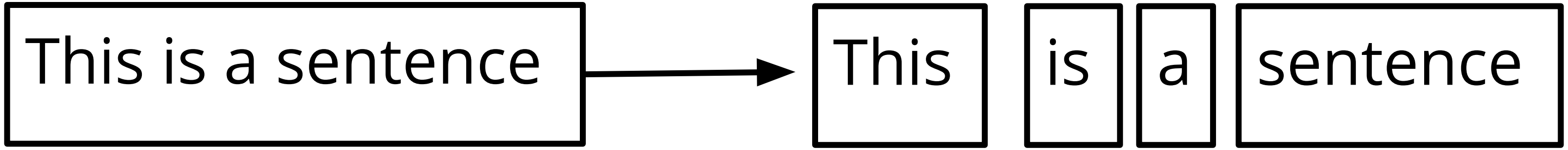
Text classification in general



Feature extraction

Preprocessing

- Tokenization: split the input into tokens



Bag-of-Words

Problems:

- No information about words order
- Word vectors are huge and very sparse
- Word vectors are not normalized
- Same words can take different forms

BIGDATA TEAM

Text Preprocessing

Preprocessing

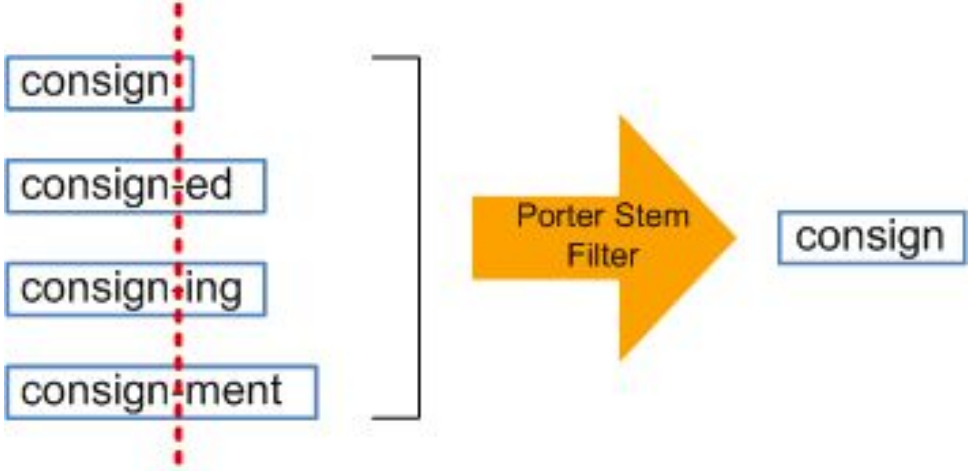
- Token normalization

Dog, dogs → dog

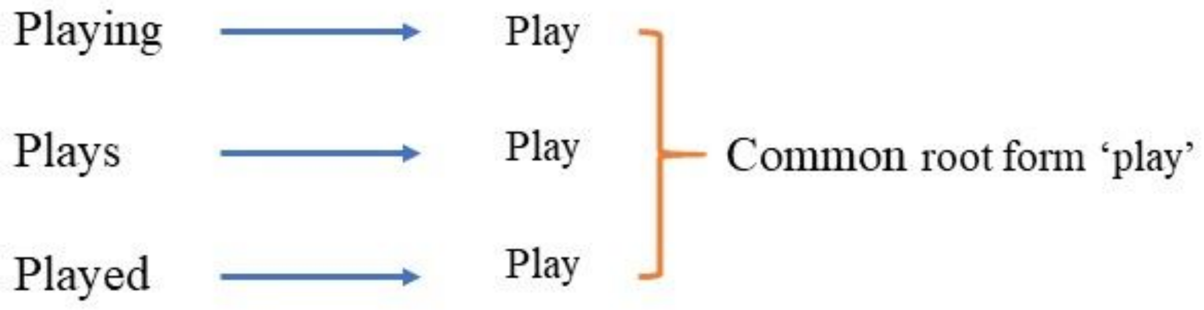
Bark, barks → bark

Preprocessing

- Token normalization:
  - **Stemming**: removing and replacing suffixes to get to the root of the word (**stem**)



- Token normalization:
  - **Stemming**: removing and replacing suffixes to get to the root of the word (**stem**)
  - **Lemmatization**: to get base or dictionary form of a word (**lemma**)



Stemming: Porter vs Lancaster

Porter stemmer

- Published in 1979
- Base starting option

Snowball stemmer (Porter 2)

- Based on Porter
- More aggressive
- Most popular option now

Lancaster stemmer

- Published in 1990
- The most aggressive
- Easy adding of your own rules

Stemming example

- Porter’s stemmer:
  - **Heuristics, applied one-by-one:**
    - SSES - SS (dresses - dress)
    - IES - I (ponies - poni)
    - S - <empty> (dogs - dog)
  - **What’s wrong?**
    - **Overstemming and understemming**

Overstemming

- University
- Universal
- Universities
- Universe

→ Univers

Understemming

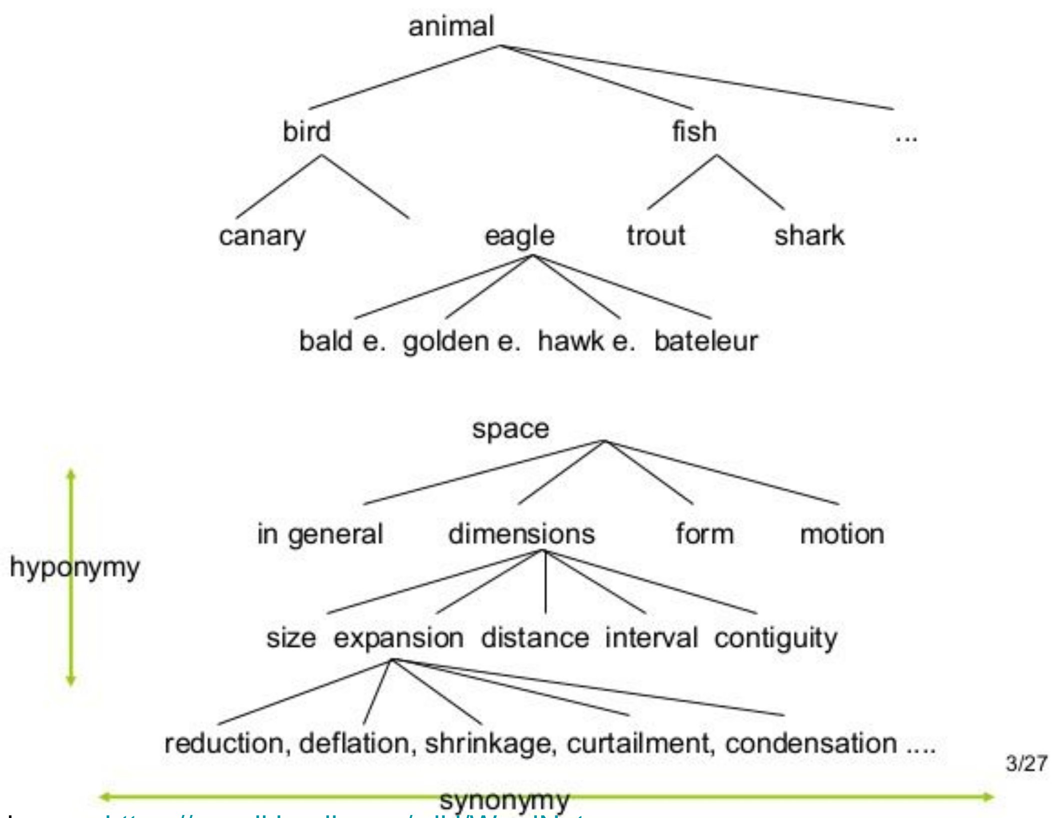
Data → dat

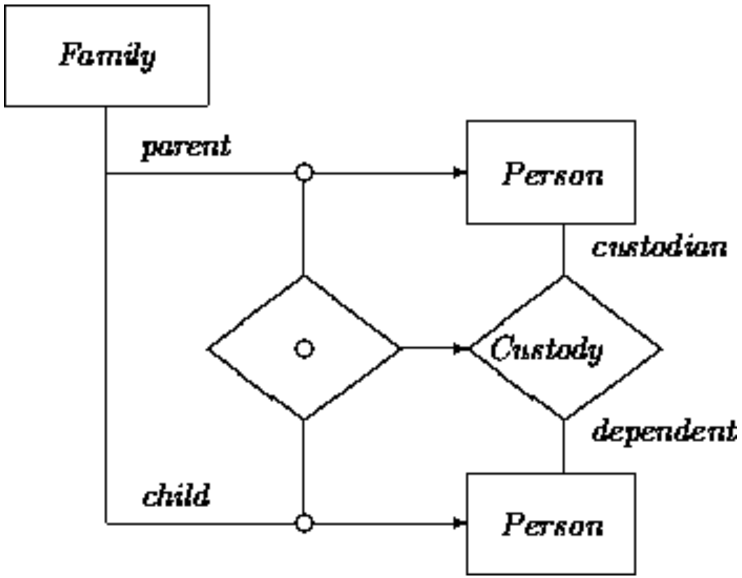
Datum → datu

Lemmatization

- Lemmatizer from NLTK:
  - Tries to resolve word to its dictionary form
  - Based on **WordNet** database
  - For the best results feed part-of-speech tagger

BTW, what is WordNet?





Read more: https://en.wikipedia.org/wiki/WordNet

Handful tools for preprocessing

- NLTK
  - nltk.stem.SnowballStemmer
  - nltk.stem.PorterStemmer
  - nltk.stem.WordNetLemmatizer
  - nltk.corpus.stopwords
- BeautifulSoup (for parsing HTML)
- Regular Expressions (import re)
- Pymorphy2

What’s left?

- Capital Letters
- Punctuation
- Contractions (e.g, etc.)
- Numbers (dates, ids, page numbers)
- Stop-words (“the”, “is”, etc.)
- Tags

Bag-of-Words

- How to improve BOW?
  - Use n-gramms instead of words!

The brown dog plays with a little cat

→

The brown

brown dog

dog plays

plays with

with a

a little

little cat

Bag-of-Words

The brown dog plays with a little cat

The brown

brown dog

dog plays

plays with

with a

a little

little cat

Do we need all this bigramms?

Bag-of-Words

The brown dog plays with a little cat →

brown dog

dog plays

little cat

Meaningful n-gramms

Meaningful n-gramms are often called **collocations**

How to detect meaningful n-gramms?

Collocations: first step

Delete:

- High-frequency n-gramms
  - Articles, prepositions
  - Auxiliary verbs (to be, to have, etc.)
  - General vocabulary
- Low-frequency n-gramms
  - Typos
  - Combinations that occur 1-2 times in a text

TF-IDF

- **Term Frequency (tf):** gives us the frequency of the word in each document in the corpus.

$$\mathrm{tf}(t,d) = f_{t,d}$$

- **Inverse Document Frequency (idf):** used to calculate the weight of rare words across all documents in the corpus. The words that occur rarely in the corpus have a high IDF score.

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

N: total number of documents in the corpus $N = |D|$

$|\{d \in D : t \in d\}|$: number of documents where the term t appears

TF-IDF example

- *Sentence A:* The car is driven on the road.
- *Sentence B:* The truck is driven on the highway.

(each sentence is a separate document)

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7			
Car	1/7	0			
Truck	0	1/7			
Is	1/7	1/7			
Driven	1/7	1/7			
On	1/7	1/7			
The	1/7	1/7			
Road	1/7	0			
Highway	0	1/7			

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0		
Car	1/7	0	log(2/1)=0.3		
Truck	0	1/7	log(2/1)=0.3		
Is	1/7	1/7	log(2/2)=0		
Driven	1/7	1/7	log(2/2)=0		
On	1/7	1/7	log(2/2)=0		
The	1/7	1/7	log(2/2)=0		
Road	1/7	0	log(2/1)=0.3		
Highway	0	1/7	log(2/1)=0.3		

TF-IDF example

Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0	0	0
Car	1/7	0	log(2/1)=0.3	0.043	0
Truck	0	1/7	log(2/1)=0.3	0	0.043
Is	1/7	1/7	log(2/2)=0	0	0
Driven	1/7	1/7	log(2/2)=0	0	0
On	1/7	1/7	log(2/2)=0	0	0
The	1/7	1/7	log(2/2)=0	0	0
Road	1/7	0	log(2/1)=0.3	0.043	0
Highway	0	1/7	log(2/1)=0.3	0	0.043

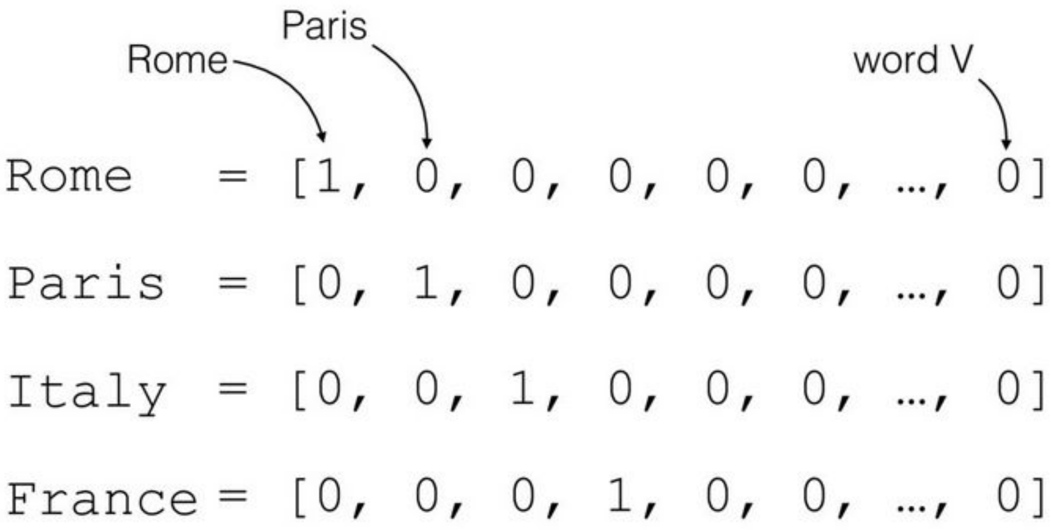
TF-IDF example

```
from sklearn.feature_extraction.text
import TfidfVectorizer
```

BIGDATA TEAM

Word Embeddings

One-hot vectors:



Problems:

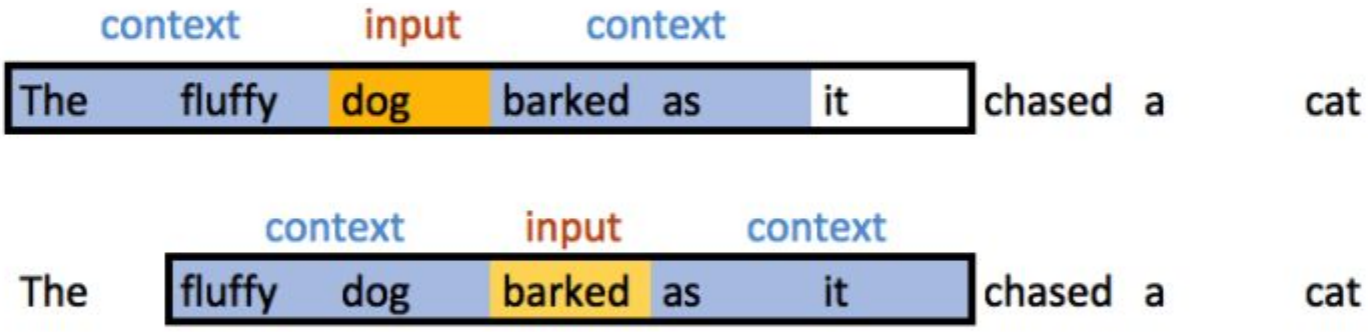
- Huge vectors
- VERY sparse
- No semantics or word similarity information included

Distributional semantics

Does vector similarity imply semantic similarity?

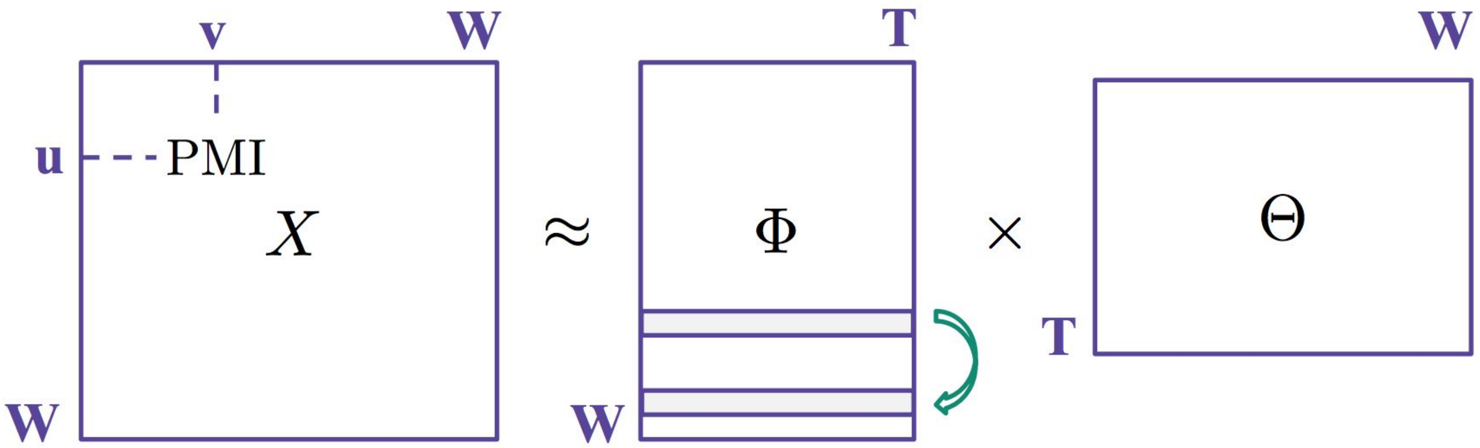
"You shall know a word by the company it keeps"

Firth, 1957



Word representations via matrix factorization

- **Input: PMI, word coocurrences, etc.**
- **Method: dimensionality reduction (SVD)**
- **Output: word similarities**



Collocations: first step

Delete:

- High-frequency n-gramms
  - Articles, prepositions
  - Auxiliary verbs (to be, to have, etc.)
  - General vocabulary
- Low-frequency n-gramms
  - Typos
  - Combinations that occur 1-2 times in a text

Collocations: context is all you need

- Coocurrence counters in a window of fixed size
  - n_{uv} states for the number of times we've seen word *u* and word *v* together in the window
- Better solution: Pointwise Mutual Information (PMI)

$$PMI = log\frac{p(u,v)}{p(u)p(v)} = log\frac{n_{uv}n}{n_u n_v}$$

- Much better solution: **Positive PMI (pPMI)**

$$pPMI = \max(0, PMI)$$

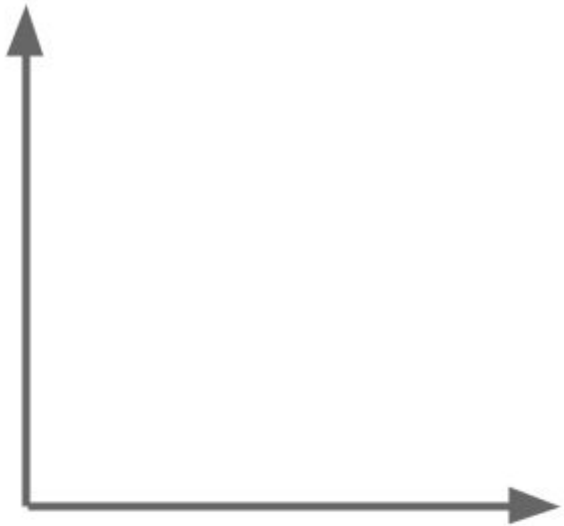
Frequency With Filter	PMI	T-test With Filter	Chi-Sq Test
(front, desk)	(universal, studios)	(front, desk)	(wi, fi)
(great, location)	(howard, johnson)	(great, location)	(cracker, barrel)
(friendly, staff)	(cracker, barrel)	(friendly, staff)	(howard, johnson)
(hot, tub)	(santa, barbara)	(hot, tub)	(la, quinta)
(clean, room)	(sub, par)	(continental, breakfast)	(front, desk)
(hotel, staff)	(santana, row)	(free, breakfast)	(universal, studios)
(continental, breakfast)	(e, g)	(great, place)	(santa, barbara)
(nice, hotel)	(elk, springs)	(parking, lot)	(santana, row)
(free, breakfast)	(times, square)	(customer, service)	(, more)
(great, place)	(ear, plug)	(desk, staff)	(flat, screen)
(desk, staff)	(la, quinta)	(walk, distance)	(french, quarter)
(parking, lot)	(fire, pit)	(comfortable, bed)	(elk, springs)
(customer, service)	(san, clemente)	(nice, hotel)	(walking, distance)

BIGDATA TEAM

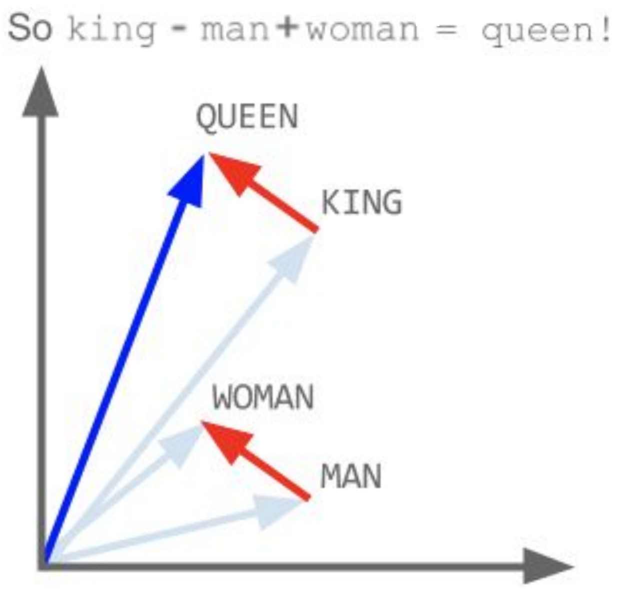
Why not to learn word vectors?

Embeddings: intuition

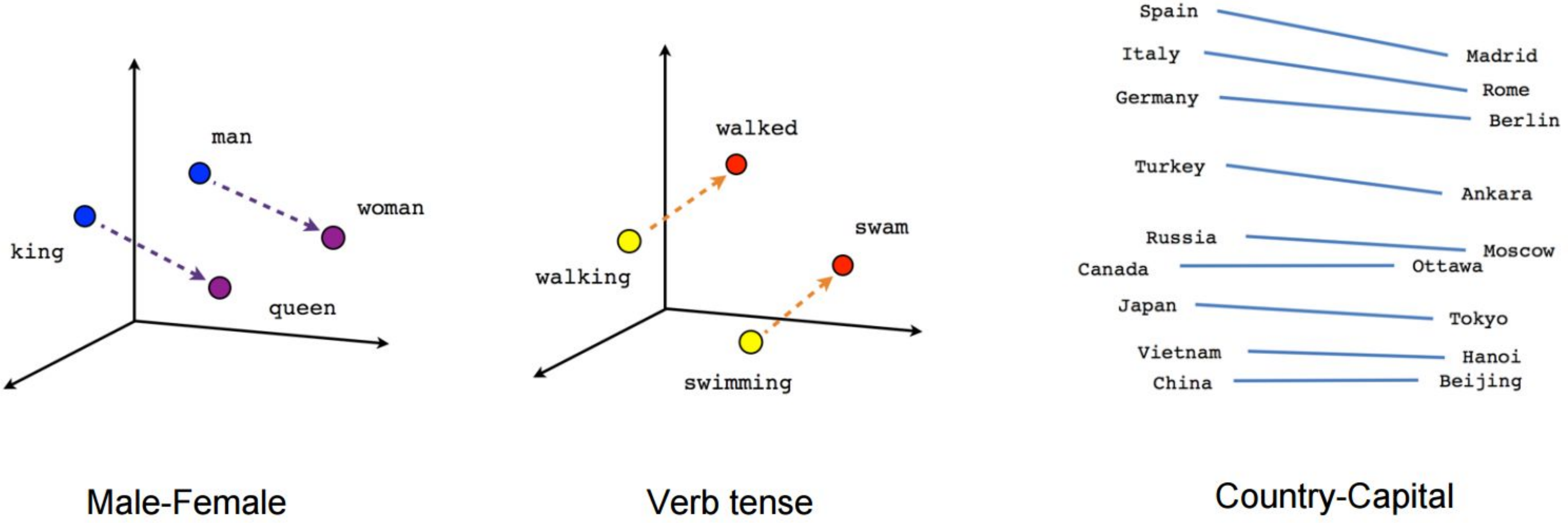
What is king - man+woman?



Embeddings: intuition



- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings

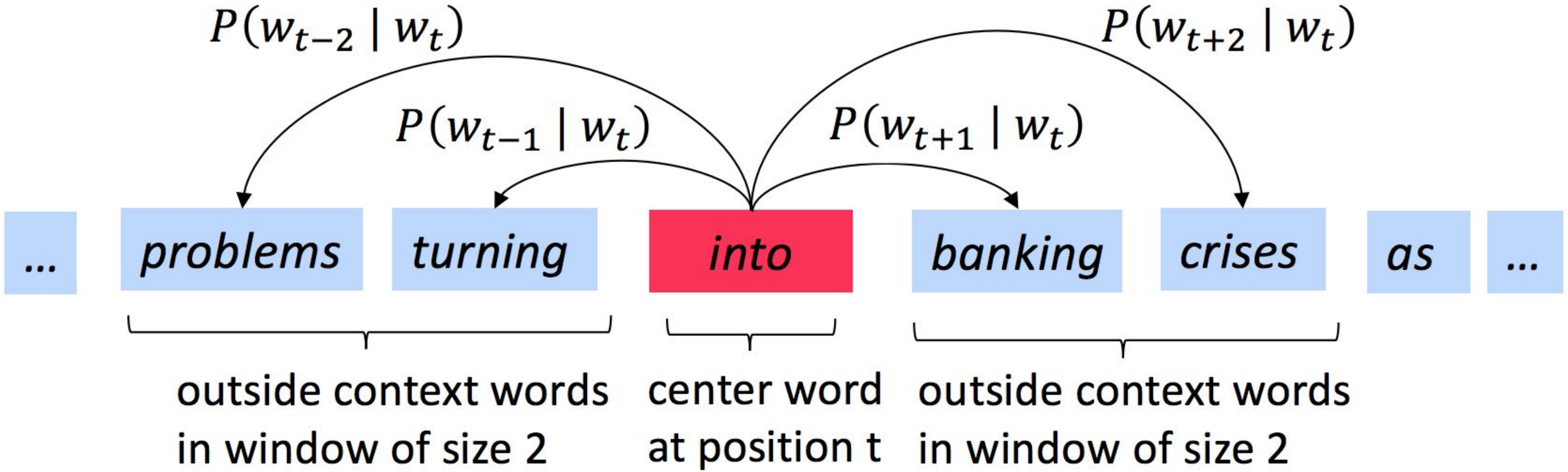


Embeddings: word2vec

Source Text	Training Samples
The quick brown fox jumps over the lazy dog. →	(the, quick) (the, brown)
The **quick** brown fox jumps over the lazy dog. →	(quick, the) (quick, brown) (quick, fox)
The quick **brown** fox jumps over the lazy dog. →	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown **fox** jumps over the lazy dog. →	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

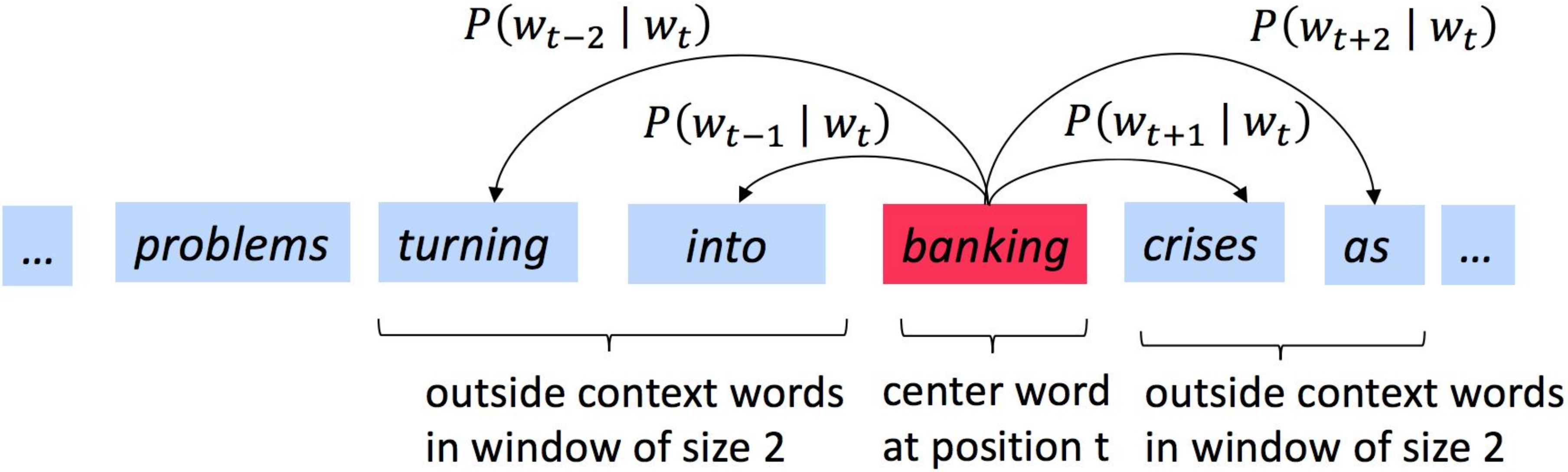
Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



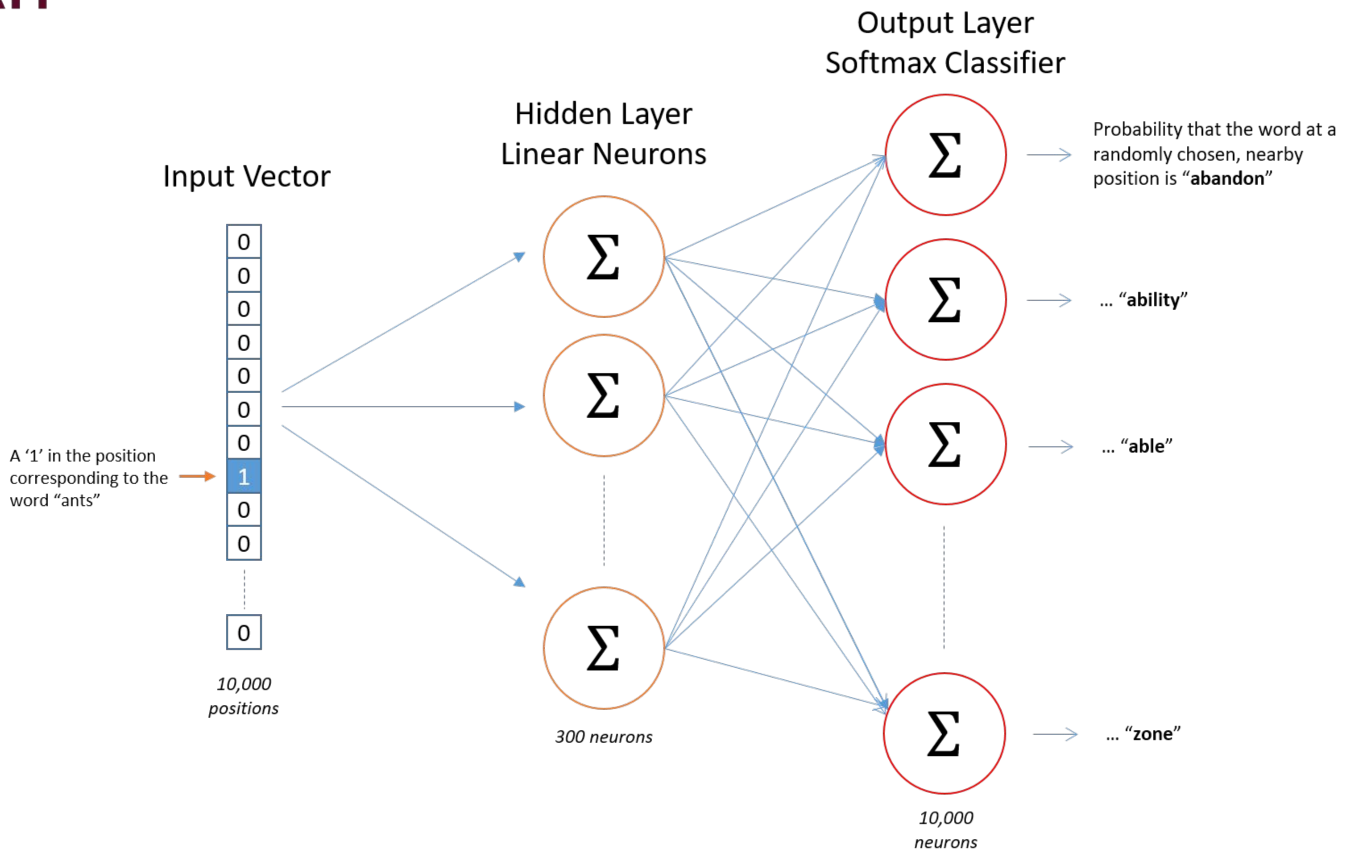
Source: CS224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

Embeddings: word2vec



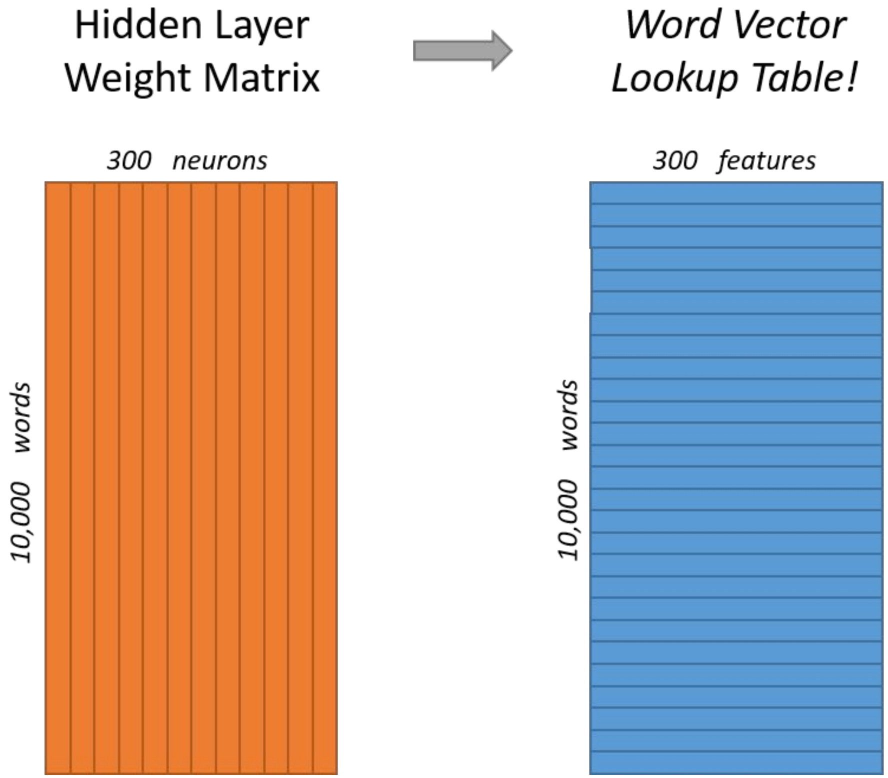
Source: CS224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec



Source: http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/

Embeddings: word2vec

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

Training is too long and computationally expensive

How to fix this?

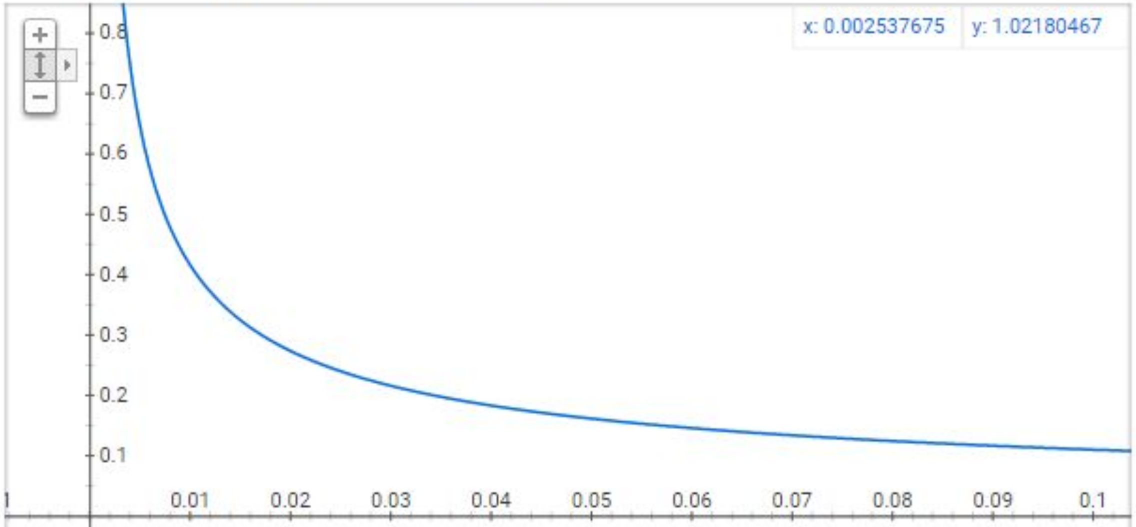
Basic approaches:

1. **1.** Treating common word pairs or phrases as single "words" in their model.
2. **2.** Subsampling frequent words to decrease the number of training examples.
3. **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.

Subsampling frequent words.

w_i is the word, $z(w_i)$ is the fraction of this word in the 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 have 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.

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

Word2vec: two models

Continuous BOW (CBOW)

$$p(w_i | w_{i-h}, \ldots, w_{i+h})$$

Predict center word from (bag of) context words

- ▶️ Predicting one word each time
- ▶️ Relatively fast

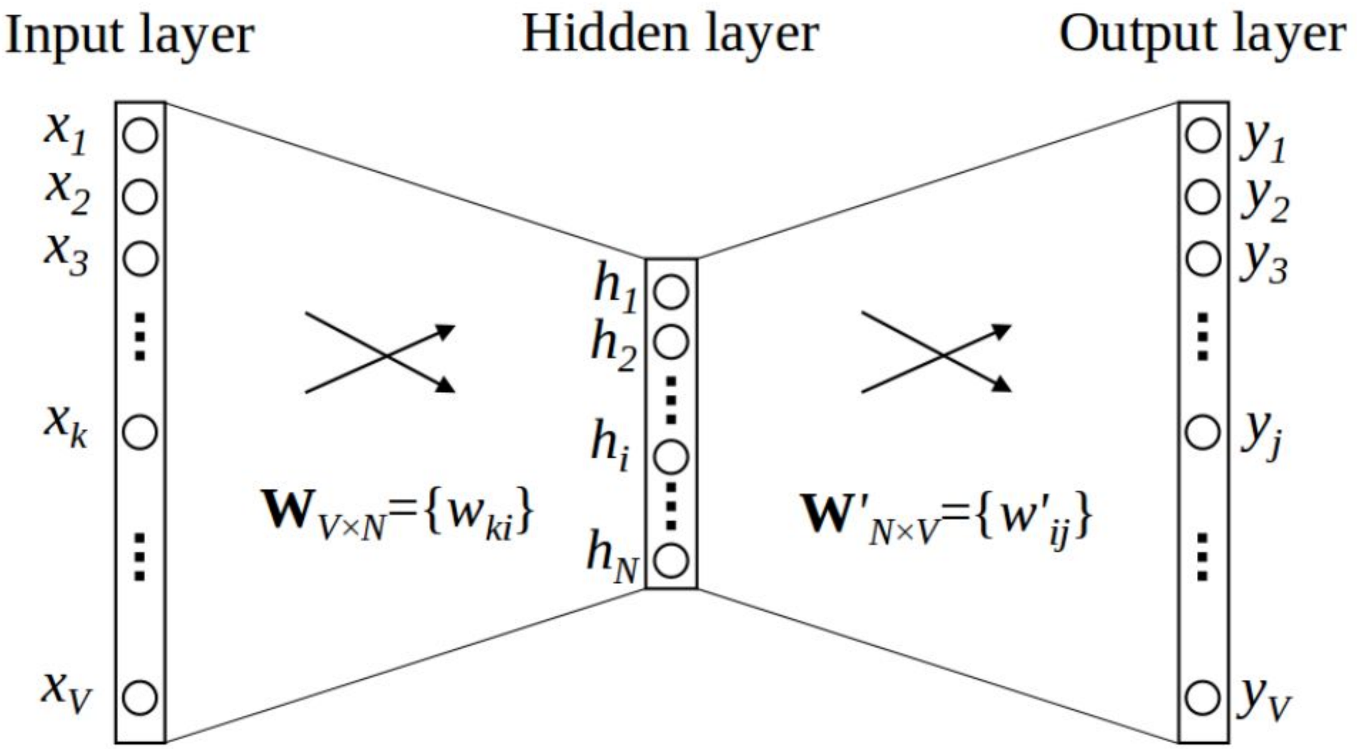
Skip-gram

$$p(w_{i-h}, \ldots w_{i+h} | w_i)$$

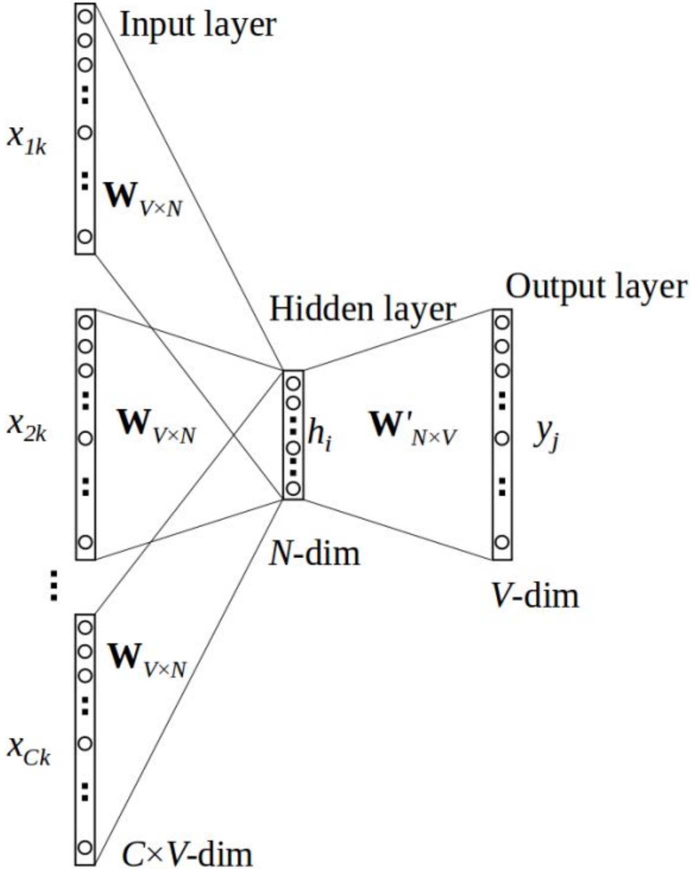
Predict context (”outside”) words (position independent) given center word

- ▶️ Predicting context by one word
- ▶️ Much slower
- ▶️ Better with infrequent words

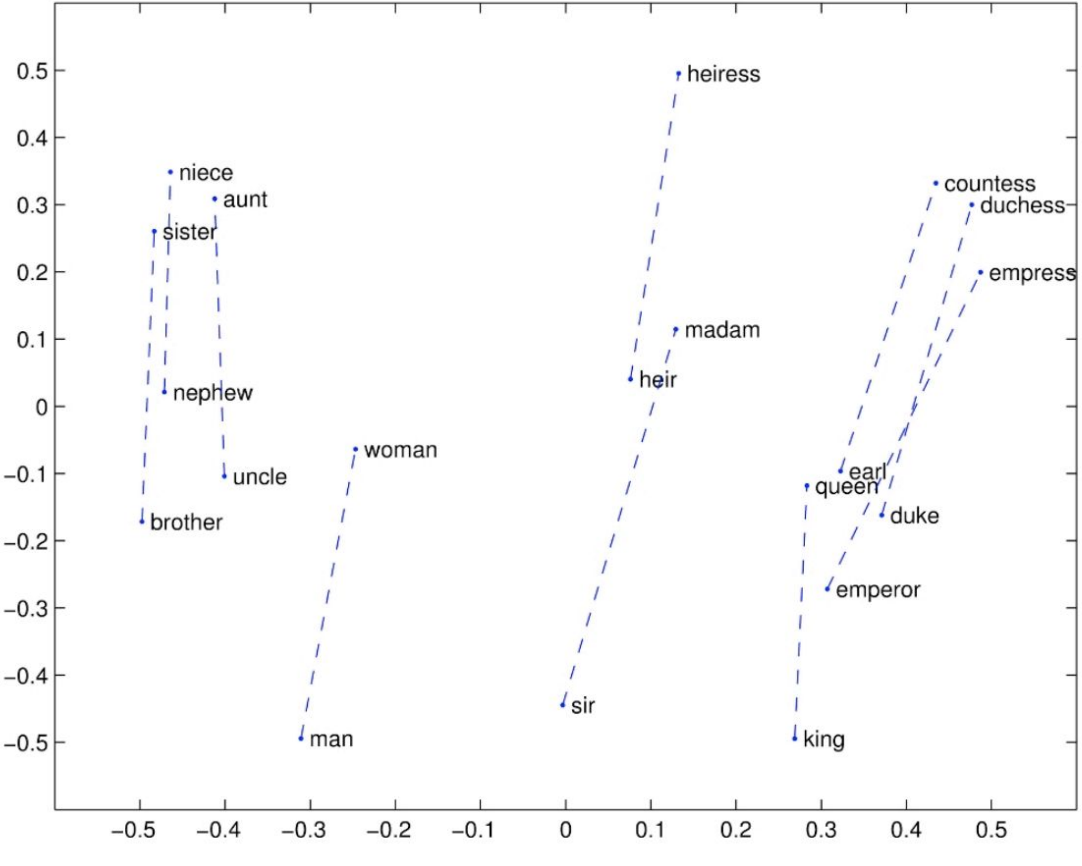
CBOW


Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$

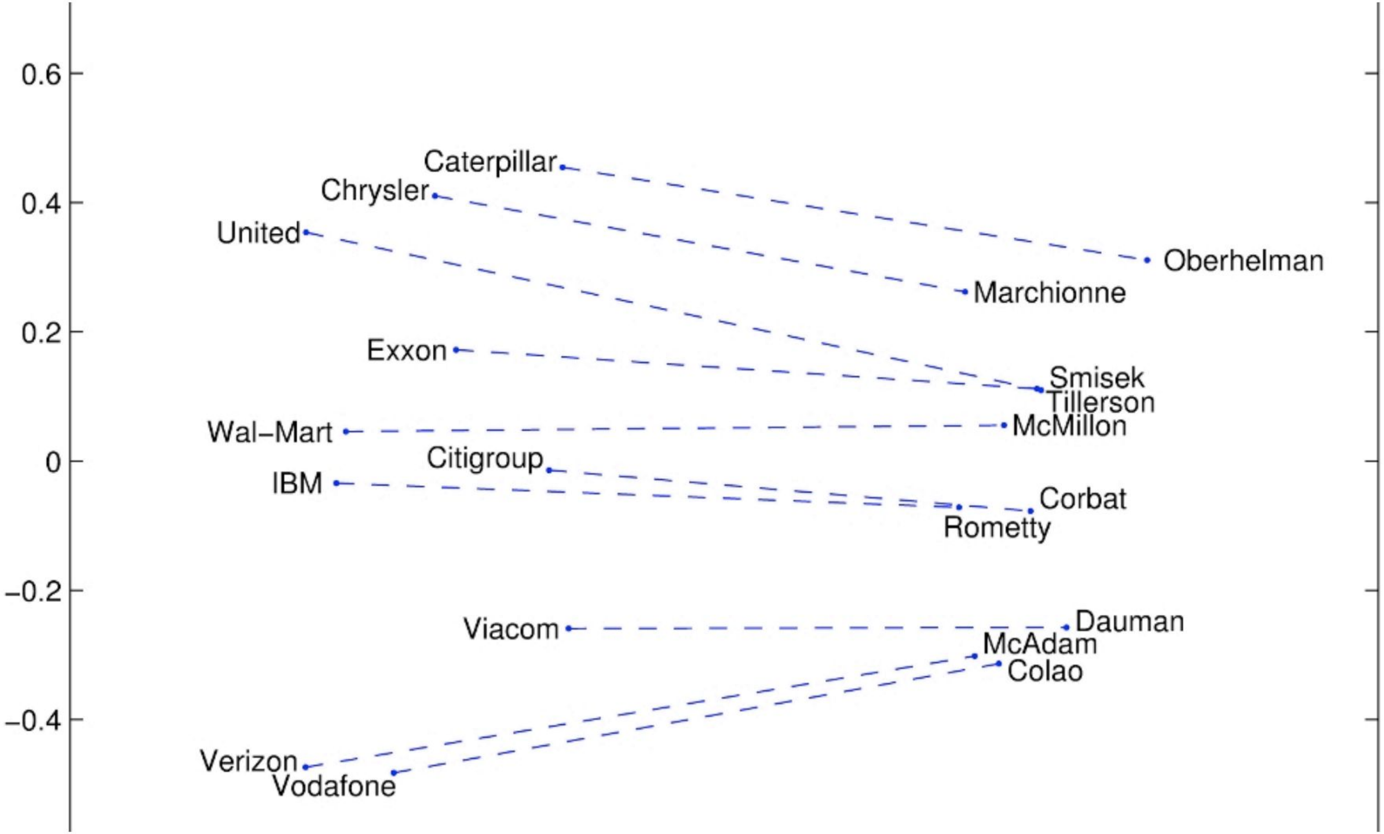
Skip-gram


Input layer
x_{1k}
$\mathbf{W}_{V\times N}$
x_{2k}
$\mathbf{W}_{V\times N}$
x_{Ck}
$\mathbf{W}_{V\times N}$
$C\times V$-dim
Hidden layer
h_i
N-dim
$\mathbf{W}'_{N\times V}$
Output layer
y_j
V-dim

GloVe Visualizations



GloVe Visualizations: Company - CEO

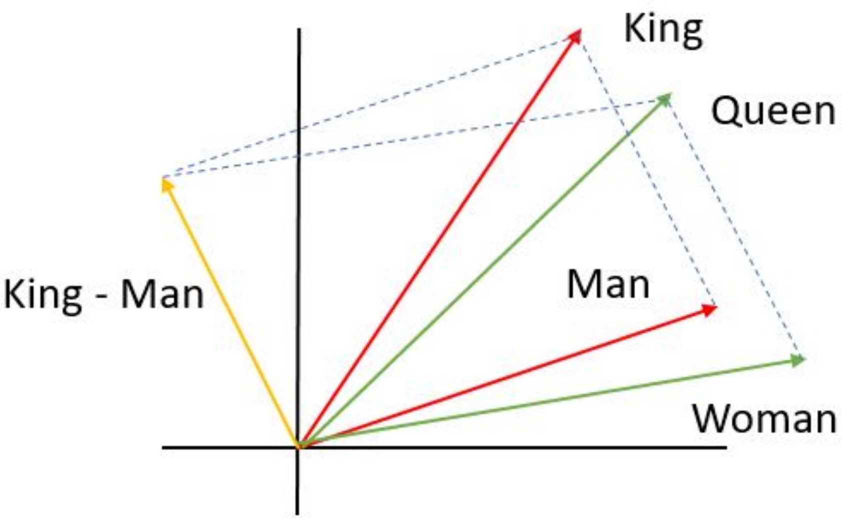


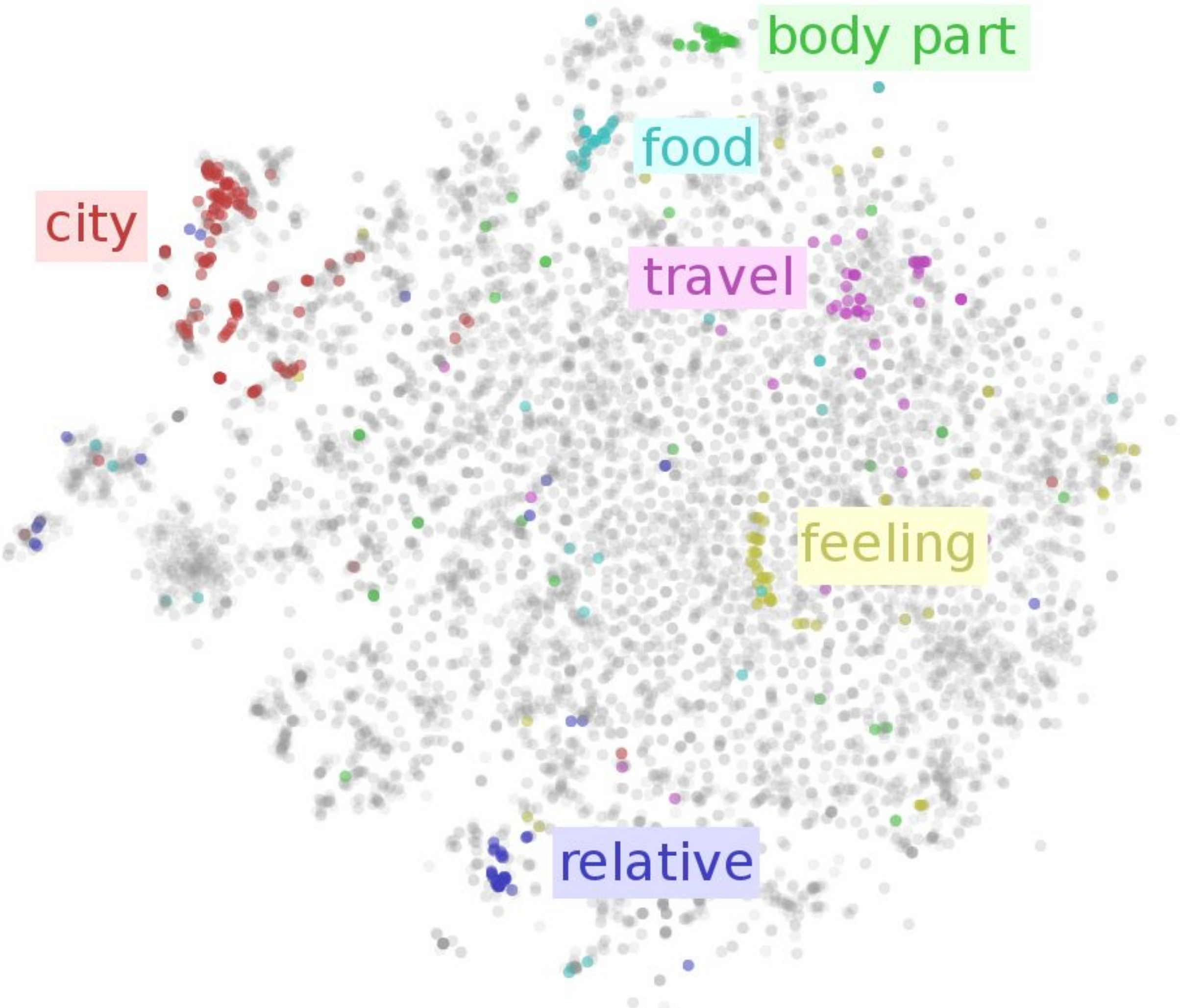
Word2vec: word analogies

King - man + woman = queen

$$\downarrow \quad x \qquad \downarrow \quad y \qquad \downarrow \quad y' \qquad \downarrow \quad target$$

$$\cos(x - y + y', target) \longrightarrow \max_{target}$$




body part
food
city
travel
feeling
relative

Conclusion

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)

Backup