

Lecture 02:

NLP problems and word representations

MADE, Moscow

01.04.2020

Radoslav Neychev

Word Representations

- NLP: introduction
- Text Preprocessing
- Feature Extraction: classical approach
 - Bag-of-Words
 - Bag-of-Ngramms
 - TF-IDF
- Tea / Coffee break (optional)
- Word Embeddings

Natural Language Processing: Introduction

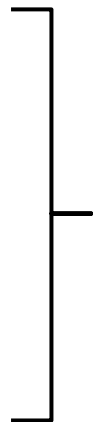


Natural Language Processing



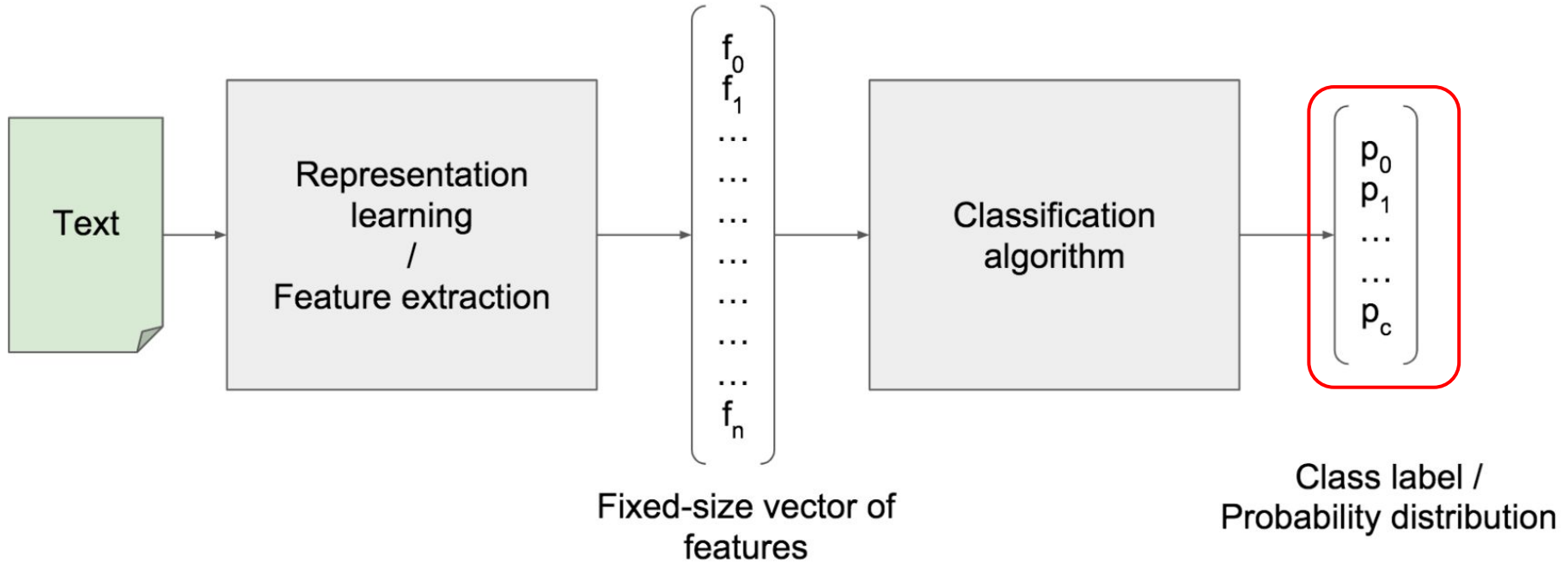
Popular NLP tasks

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



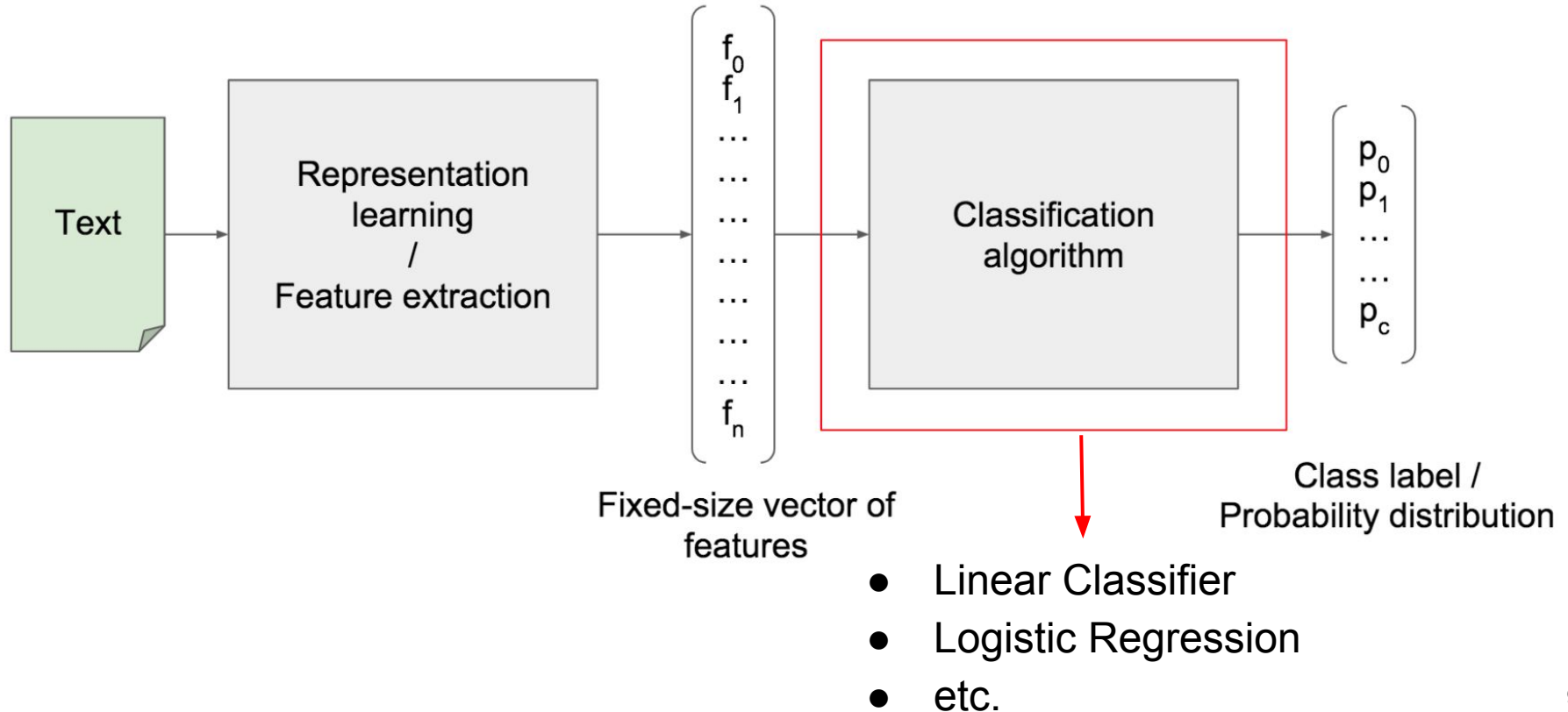
Text classification tasks

Text classification in general

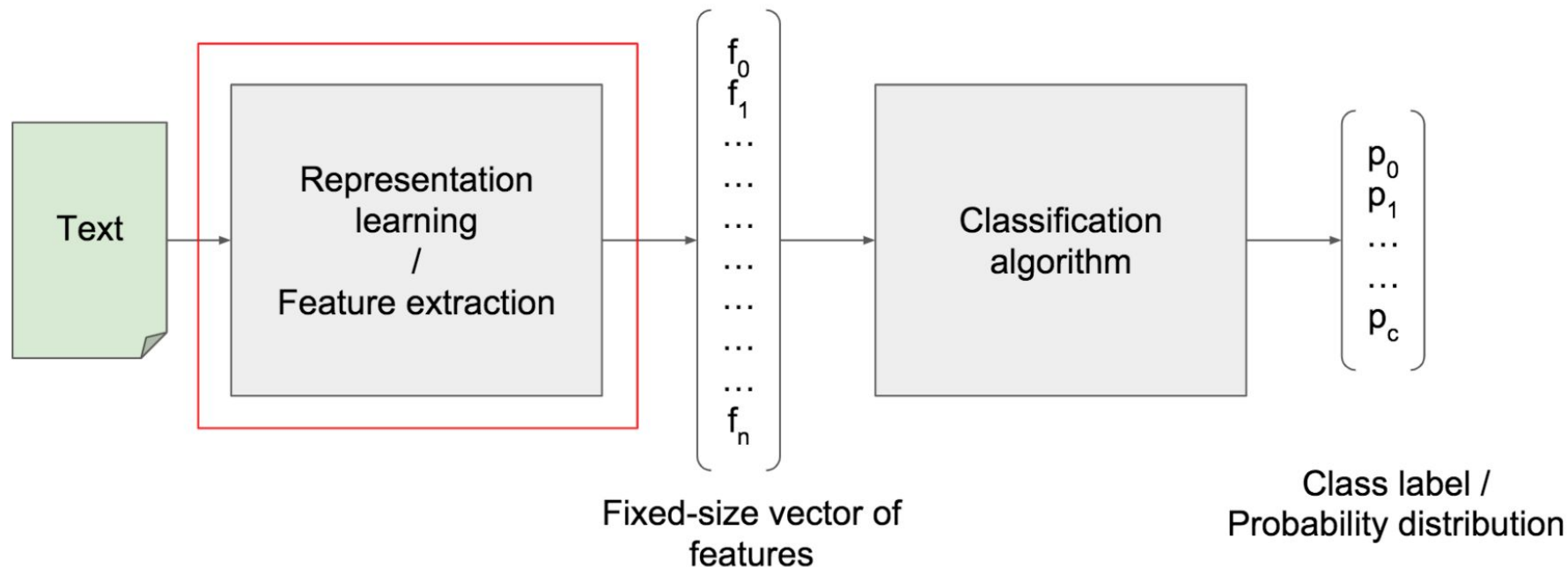


- Discrete labels:
 - Binary
 - spam filtering, sentiment analysis
 - Multi-class
 - categorization of items by its description
 - Multi-label
 - #hashtag prediction
- Continuous labels:
 - Predict product price by its description

Text classification in general

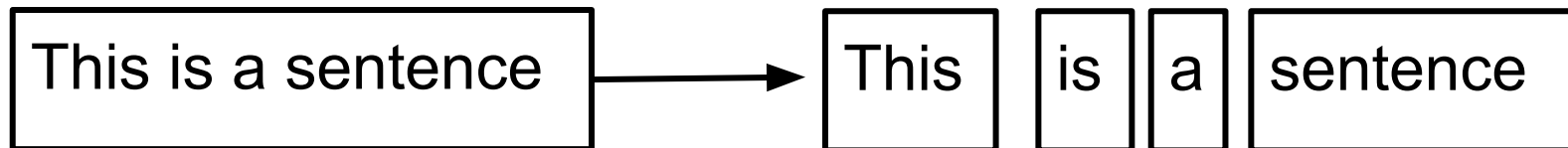


Text classification in general



Text Preprocessing

- Tokenization: split the input into tokens

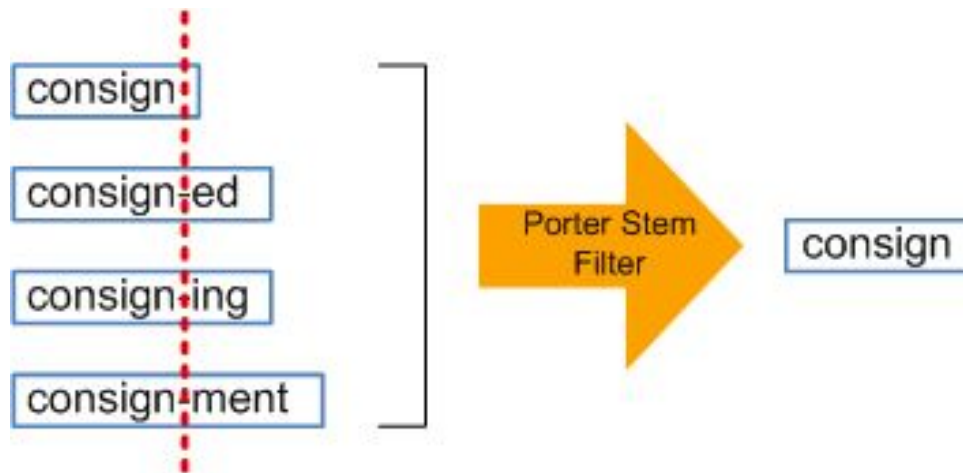


- Token normalization

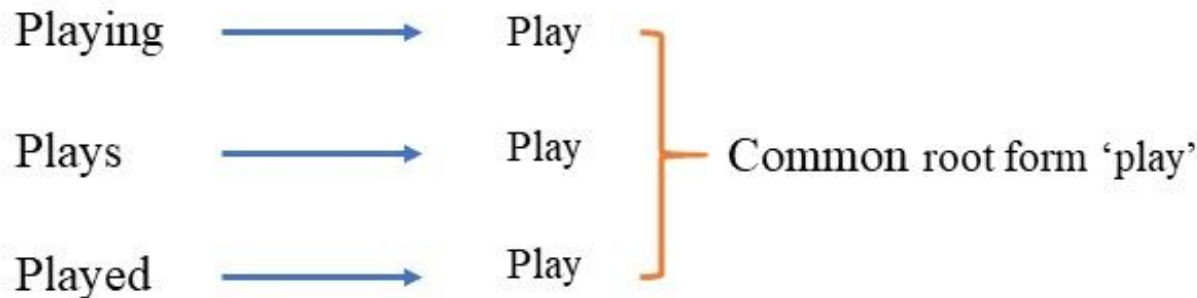
Dog, dogs → dog

Bark, barks → bark

- 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

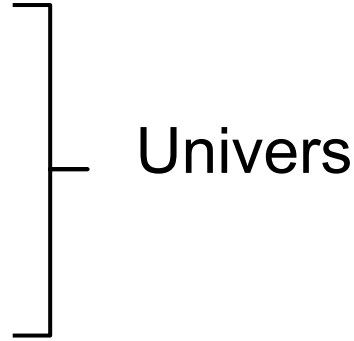
Lancaster stemmer

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

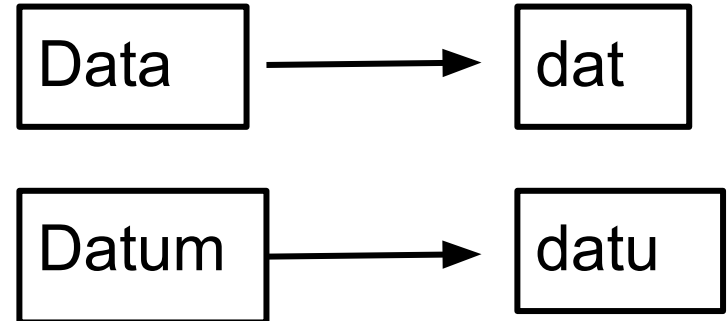
- Porter's stemmer:
 - **Heuristics, applied one-by-one:**
 - SSES - SS (dresses - dress)
 - IES - I (ponies - poni)
 - S - <empty> (dogs - dog)
 - **What's wrong?**
 - **Overstemming and understemming**

Overstemming

- University
- Universal
- Universities
- Universe

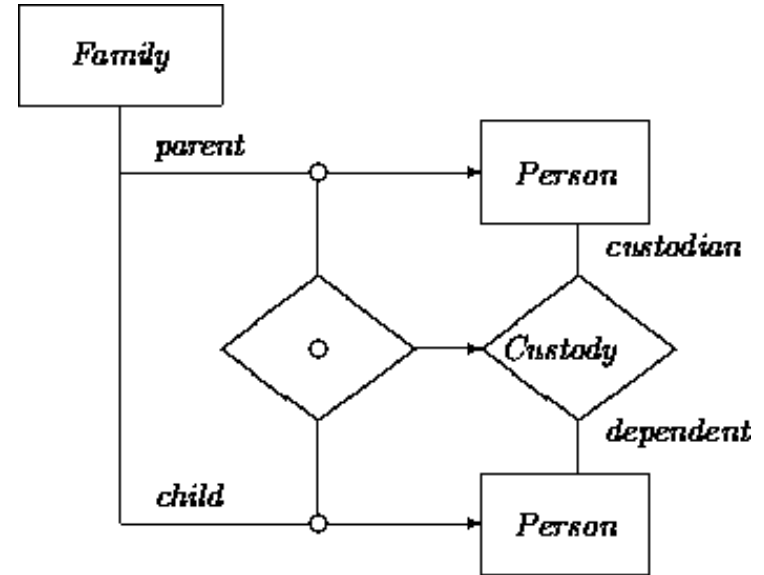
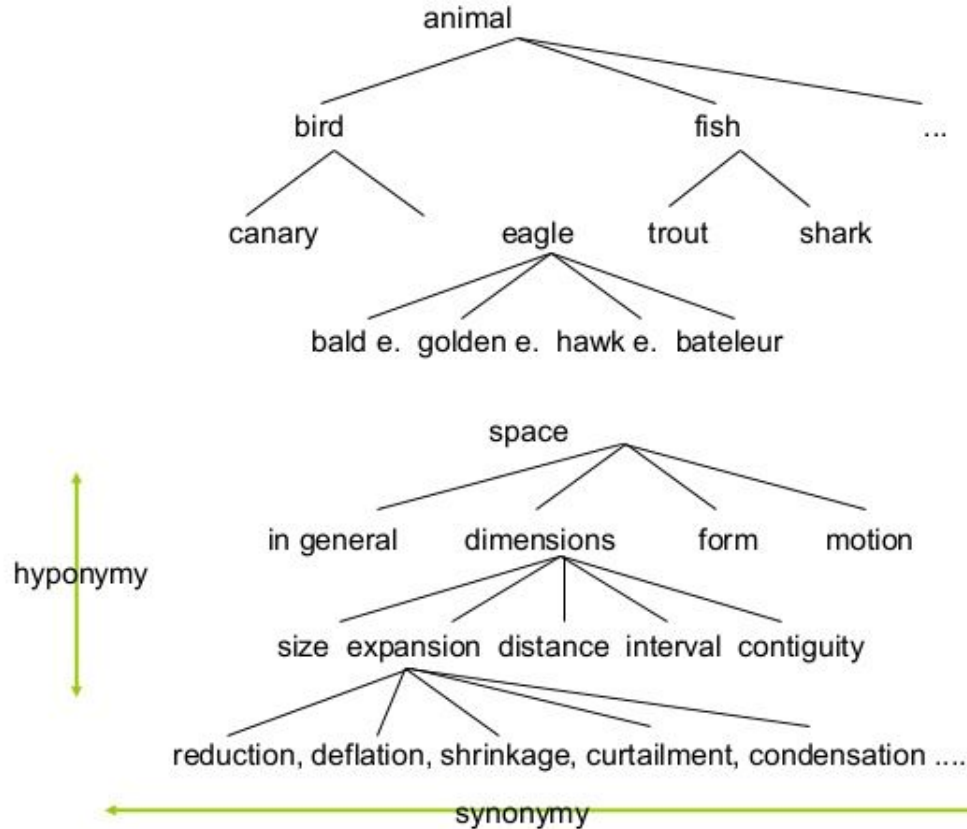


Understemming



- 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?



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)

What's left?

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

Feature extraction

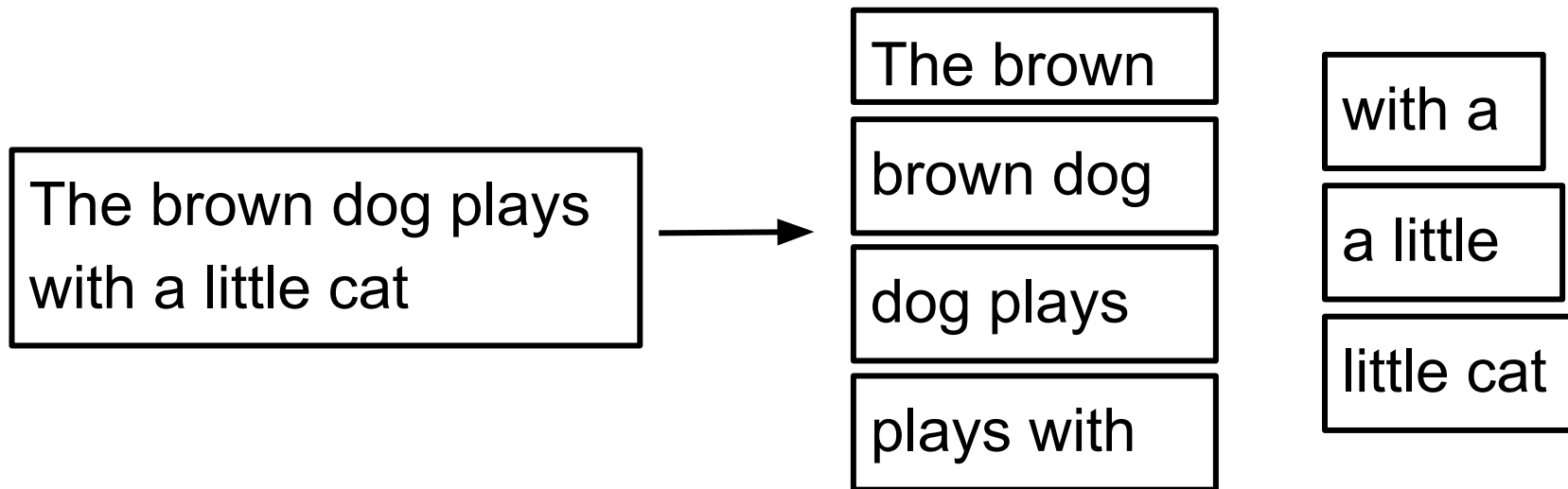
Once again: Bag-of-Words

the dog is on the table

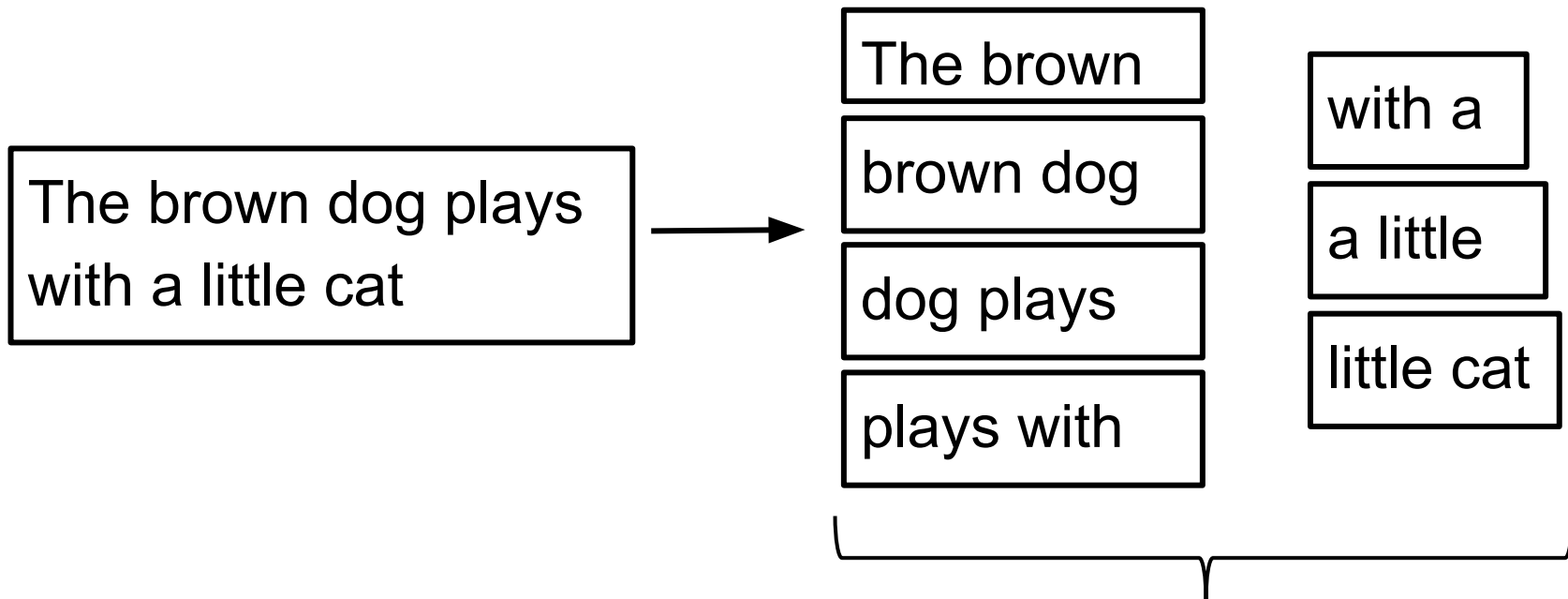
0	0	1	1	0	1	1	1
are	cat	dog	is	now	on	table	the

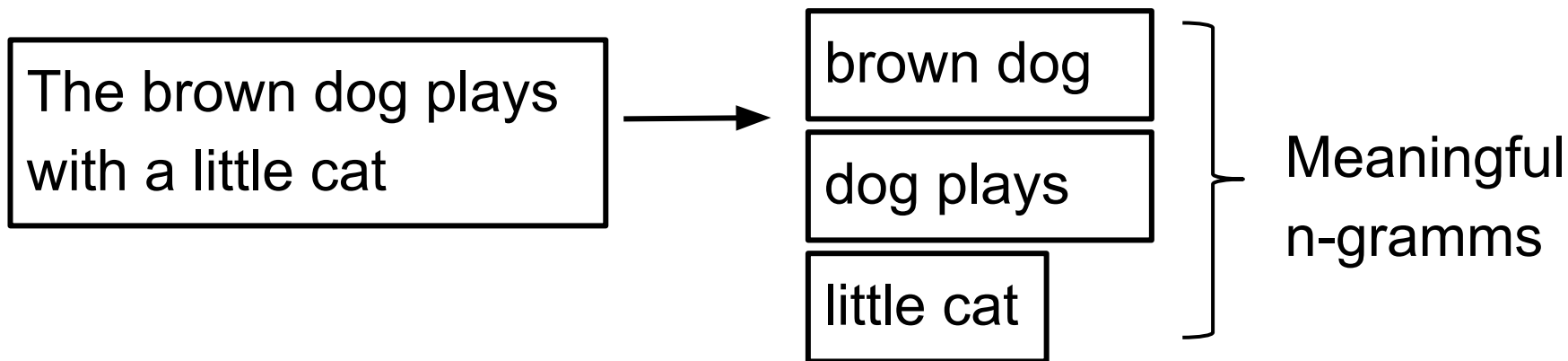
- Problems:
 - No information about words order
 - Word vectors are huge and very sparse
 - Word vectors are not normalized

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



Bag-of-Words





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

How to detect meaningful n-gramms?

- 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

- Cooccurrence 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)**

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

- Use statistics:
 - T-criterion

$$t = \frac{\overline{x} - \mu}{\sqrt{\frac{s^2}{N}}}$$

H_0 : 'social media' occurs with probability:

$$\mu = P(\text{social})P(\text{media}) = \frac{C(\text{social})(\text{media})}{N^2}$$

H_a : 'social media' does not occur with such a probability

- Use statistics:
 - Chi-squared

$$\chi^2 = \sum_{ij} \frac{(O_{ij} - E_{ij})^2}{E_{ij}}$$

$$E(\text{social media}) = \frac{C(\text{social})}{N} \cdot \frac{C(\text{media})}{N} \cdot N$$

O_{ij} from table

	w1 = social	w1 != social
w2 = media	C(social media)	C(x media) where x could be any word
w2 != media	C(social x) where x could be any word	C(any pair not starting with social or ending with media)

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)

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

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

- **Inverse Data 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.

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

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
```



Word Embeddings

- **One-hot vectors:**

Rome = $[1, 0, 0, 0, 0, 0, \dots, 0]$

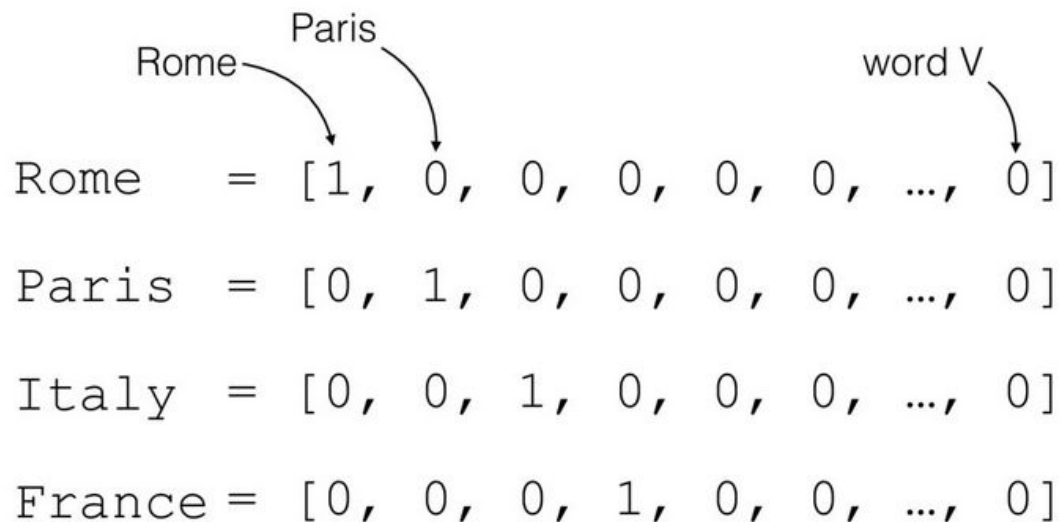
Paris = $[0, 1, 0, 0, 0, 0, \dots, 0]$

Italy = $[0, 0, 1, 0, 0, 0, \dots, 0]$

France = $[0, 0, 0, 1, 0, 0, \dots, 0]$

One-hot vectors

Rome = [1, 0, 0, 0, 0, 0, ..., 0]
Paris = [0, 1, 0, 0, 0, 0, ..., 0]
Italy = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]



The diagram illustrates one-hot vectors for four words: Rome, Paris, Italy, and France. Each word is associated with a vector of length V, where only one element is 1 and the rest are 0. Arrows indicate the mapping: Rome points to the first element (1), Paris to the second (1), Italy to the third (1), and France to the fourth (1). An arrow labeled 'word V' points to the last element of the vectors, which is 0 for all words shown.

Problems:

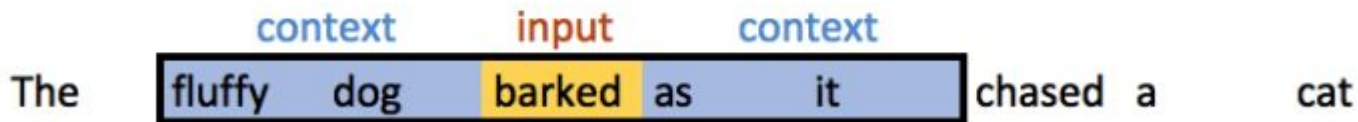
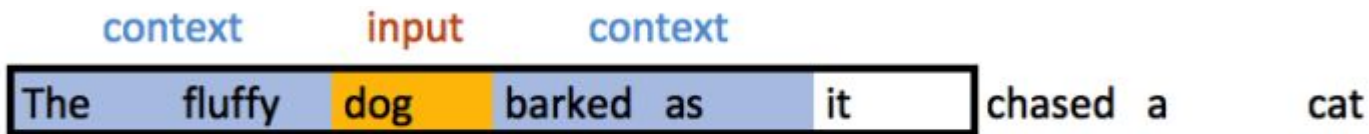
- Huge vectors
- VERY sparse
- No semantics or word similarity information included

Distributional semantics

Does vector similarity imply semantic similarity?

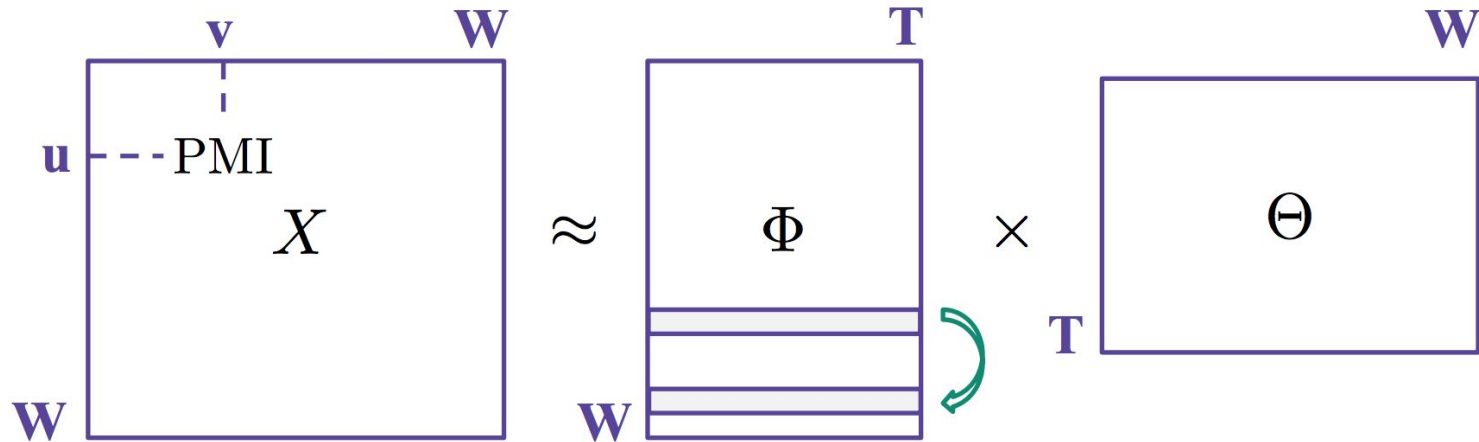
“You shall know a word by the company it keeps”

Firth, 1957



Word representations via matrix factorization

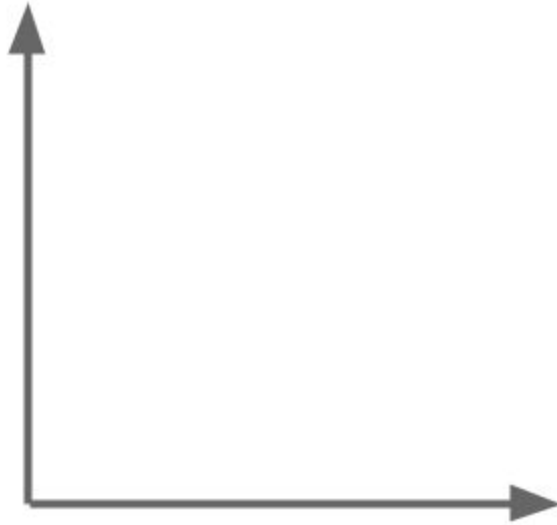
- **Input: PMI, word cooccurrences, etc.**
- **Method: dimensionality reduction (SVD)**
- **Output: word similarities**



Why not to learn word vectors?

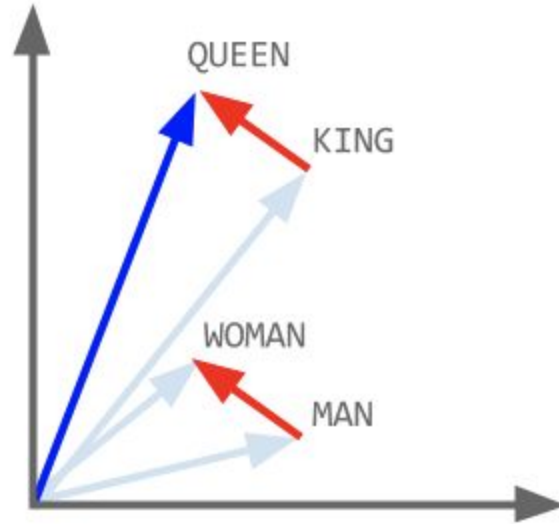
Embeddings: intuition

What is king - man + woman?

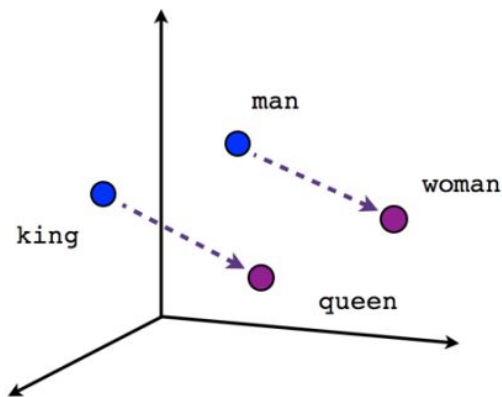


Embeddings: intuition

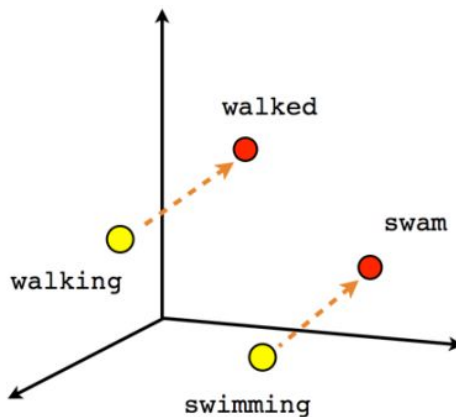
So $\text{king} - \text{man} + \text{woman} = \text{queen!}$



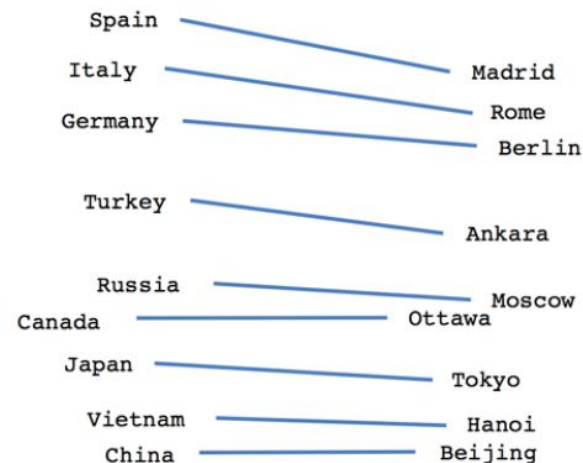
- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings



Male-Female



Verb tense

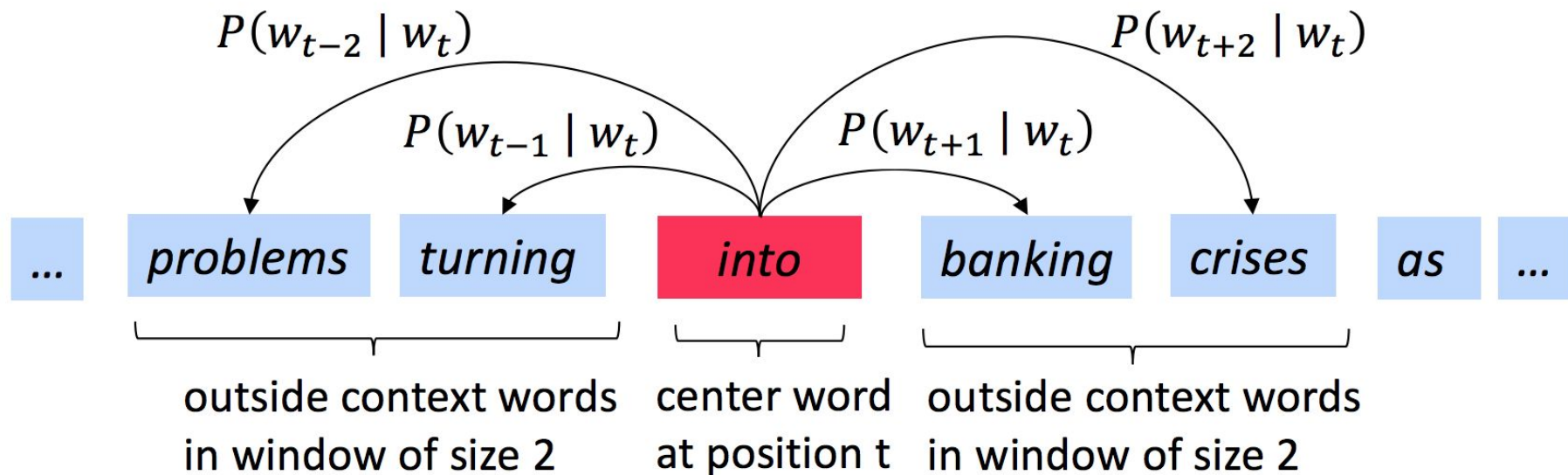


Country-Capital

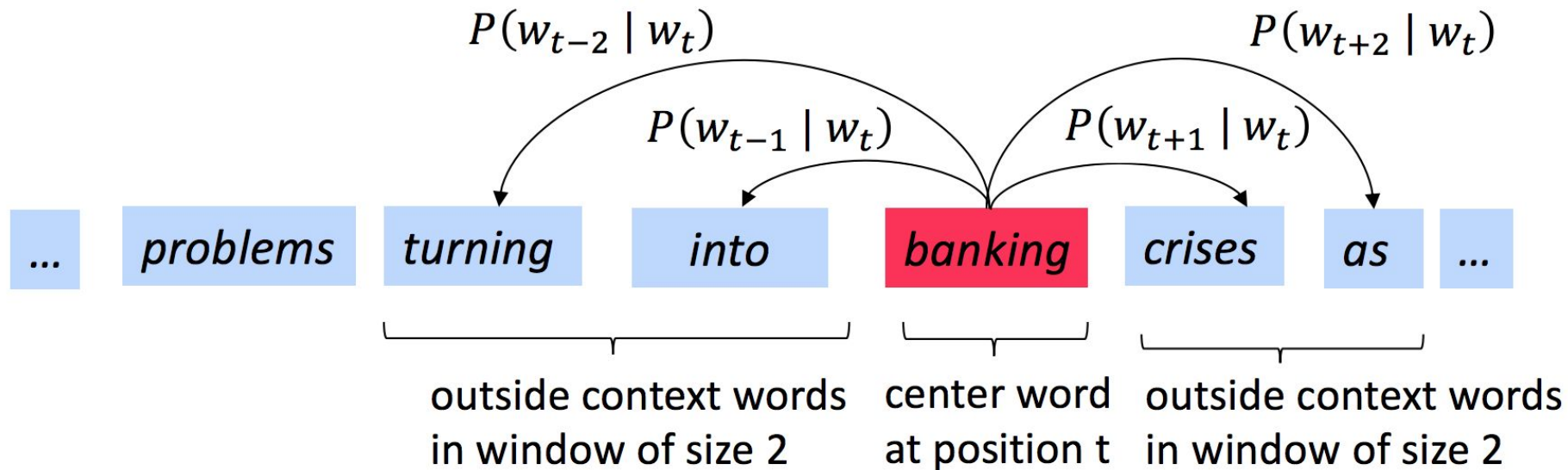
Embeddings: word2vec

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		

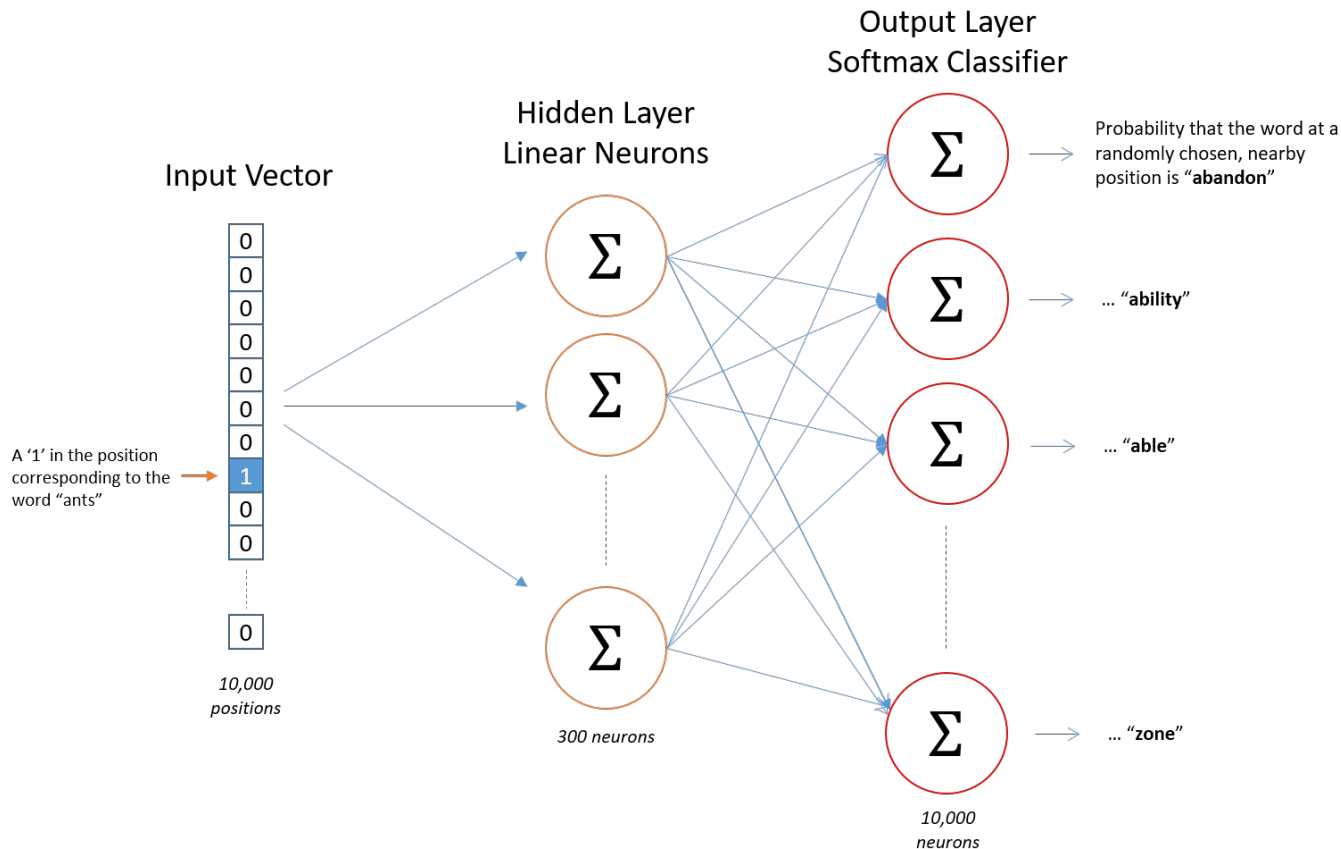
Embeddings: word2vec



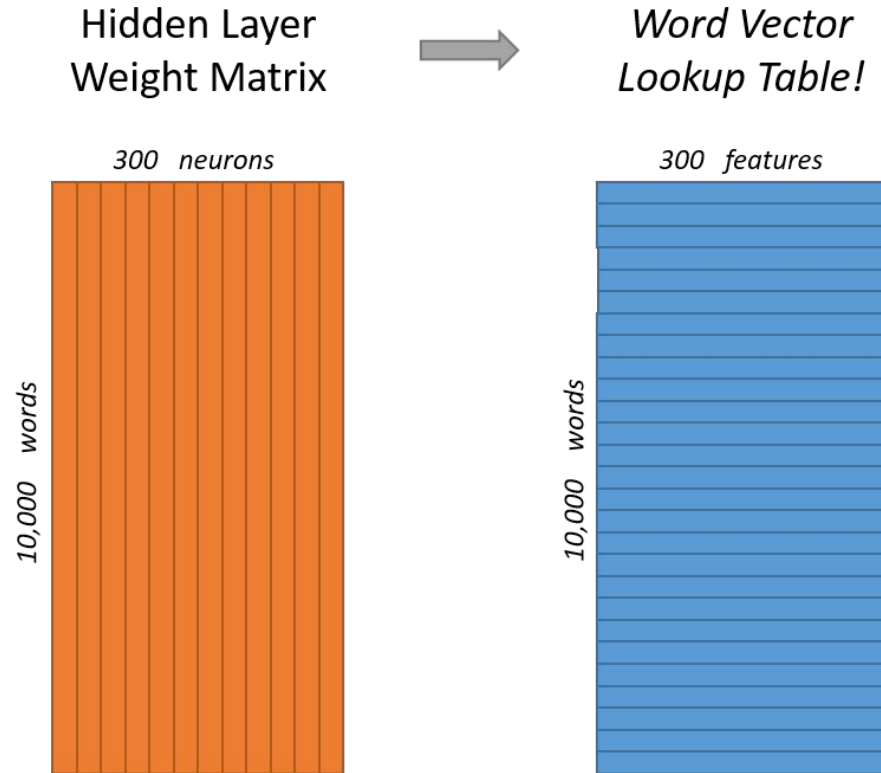
Embeddings: word2vec



Embeddings: word2vec



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?

Basic approaches:

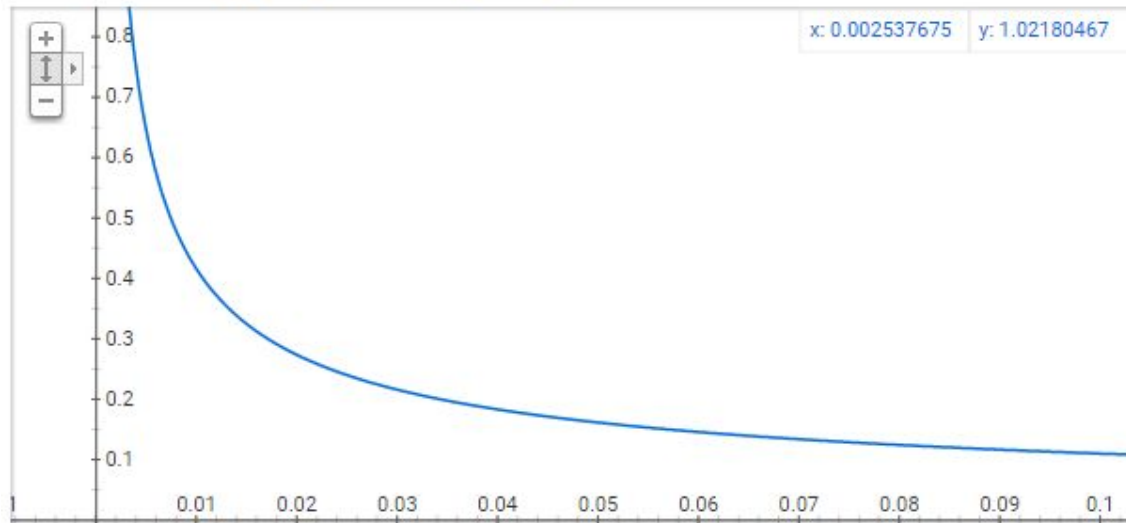
1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update only a small percentage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

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

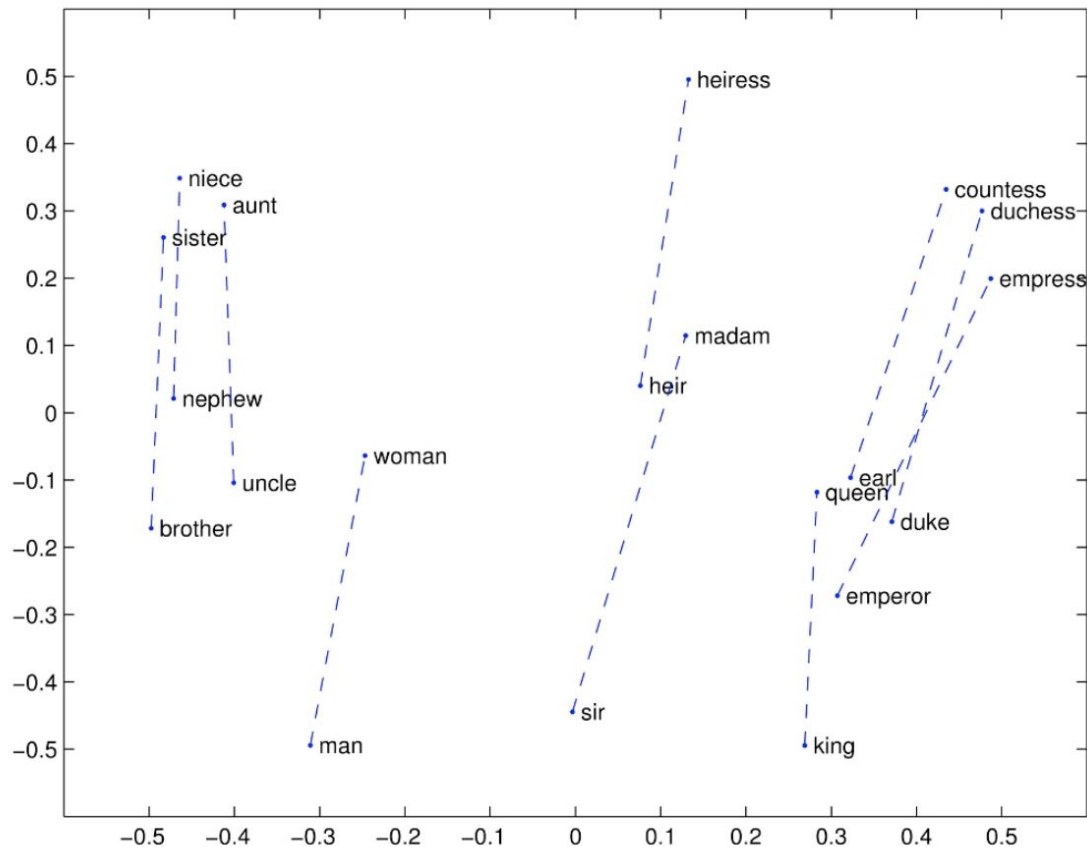
Embeddings: negative sampling

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

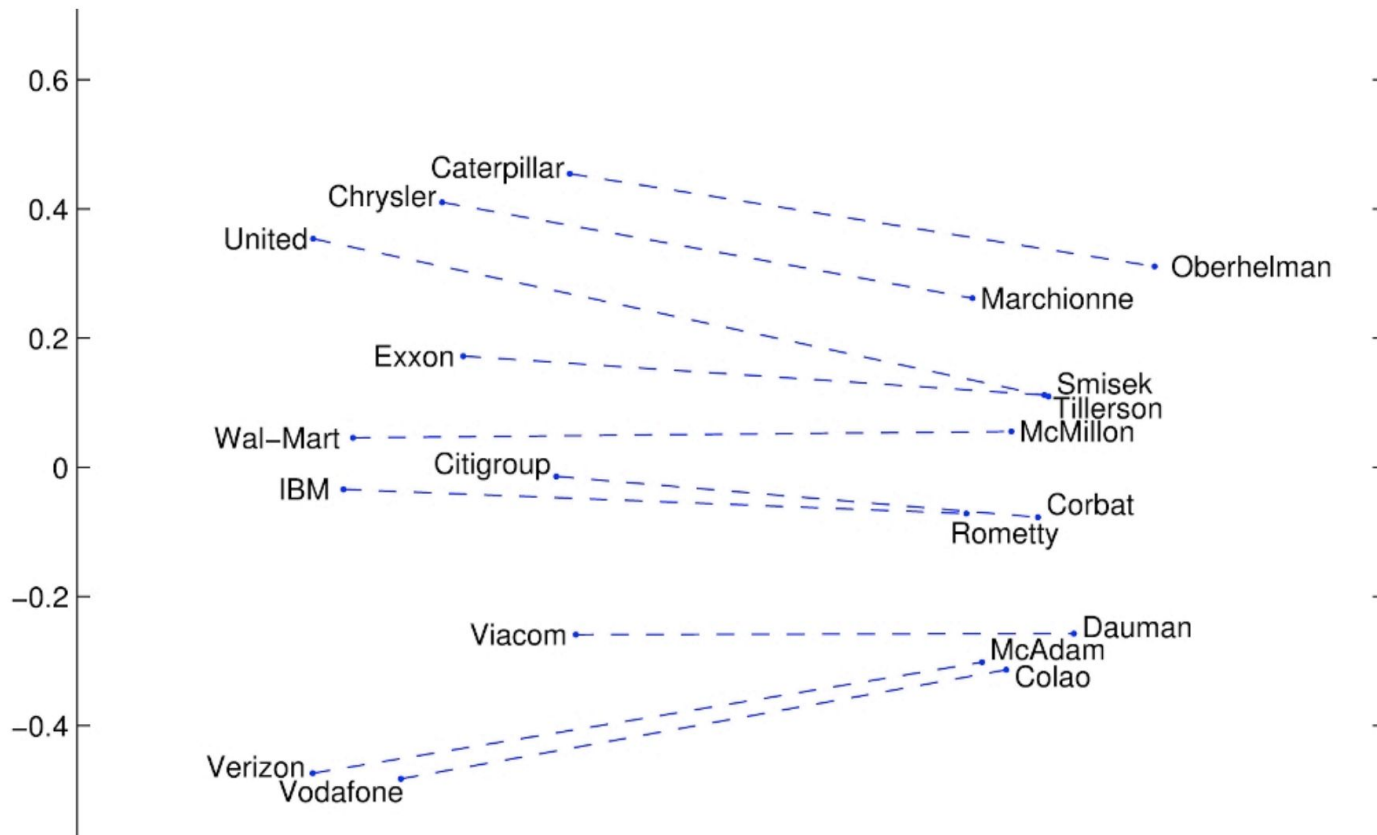
More frequent words are selected to be negative samples more often. The probability for a selecting a word is just it's weight divided by the sum of weights for all words.

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

GloVe Visualizations



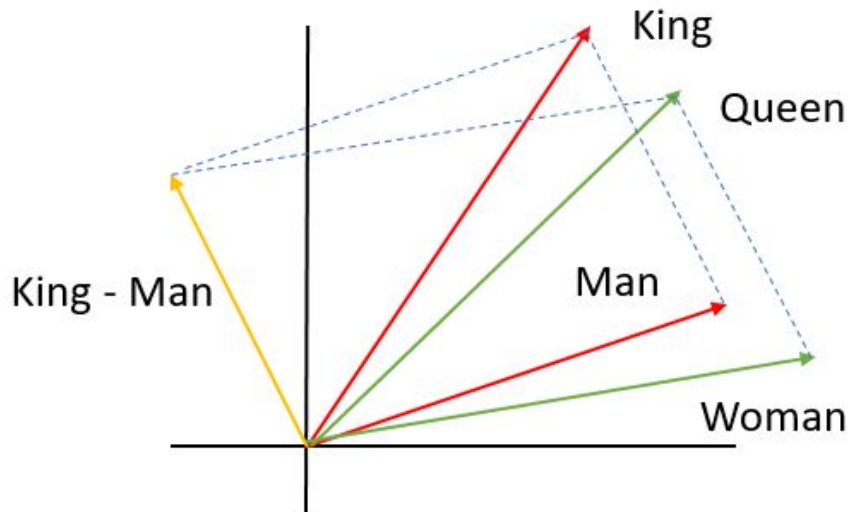
GloVe Visualizations: Company - CEO

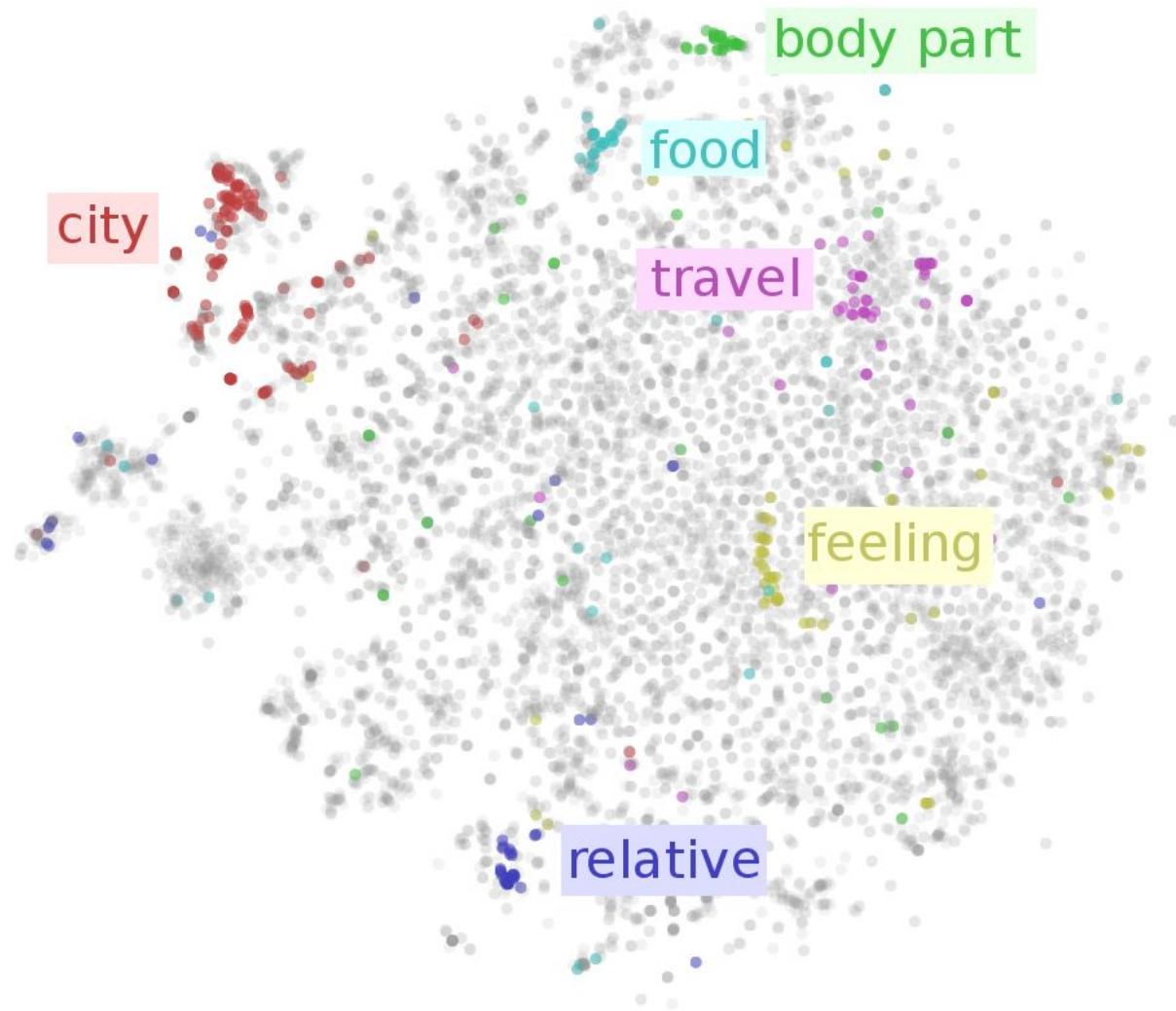


Word2vec: word analogies

King - man + woman = queen
↓ ↓ ↓ ↓
 x y y' $target$

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





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)

Word2vec: two models

Continuous BOW (CBOW)

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

Predict center word from
(bag of) context words

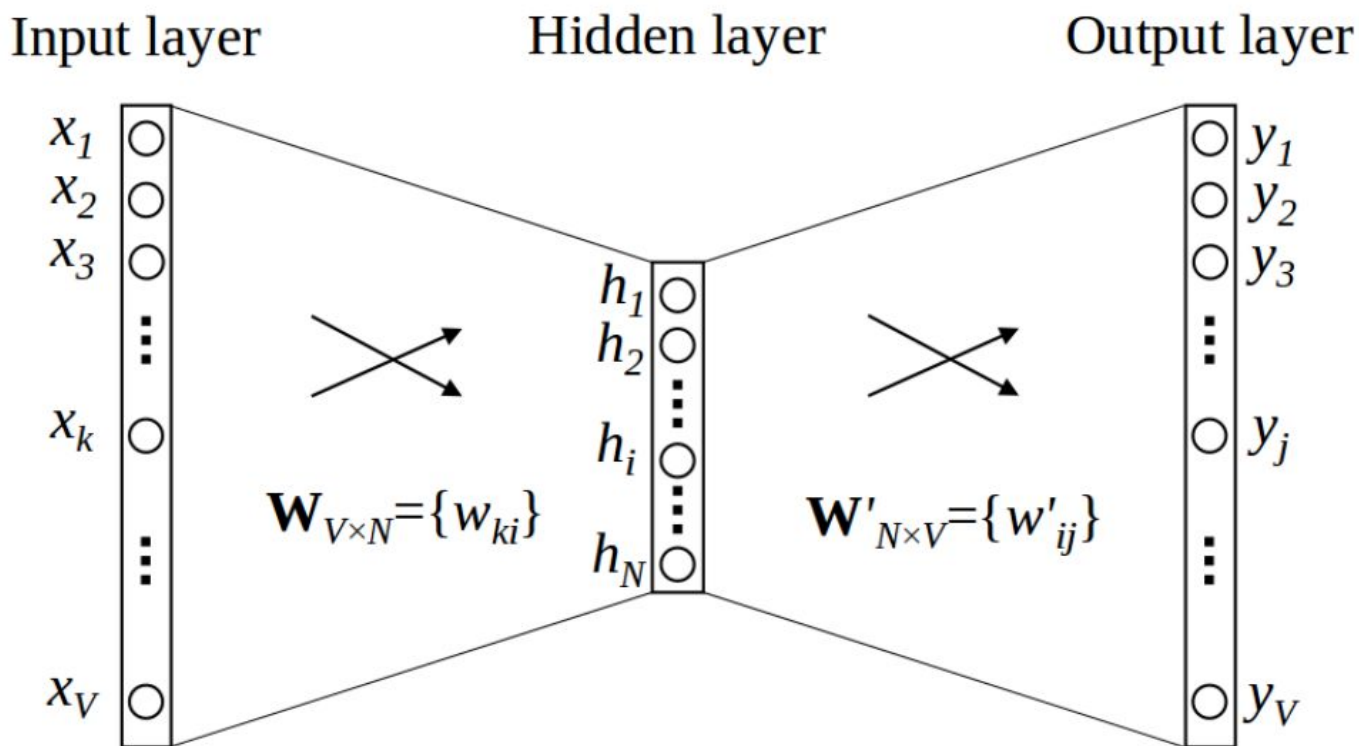
- Predicting one word each time
- Relatively fast

Skip-gram

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

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

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



Skip-gram

