Machine Learning
Lecture 11: Embeddings

Iurii Efimov

girafe
ai

Outline

1. NLP introduction
2. Text preprocessing
3. Feature extraction:
   a. Bag-of-Words
   b. Bag-of-Ngrammes
   c. TF-IDF
4. Word embeddings
5. Word2vec

Natural Language Processing

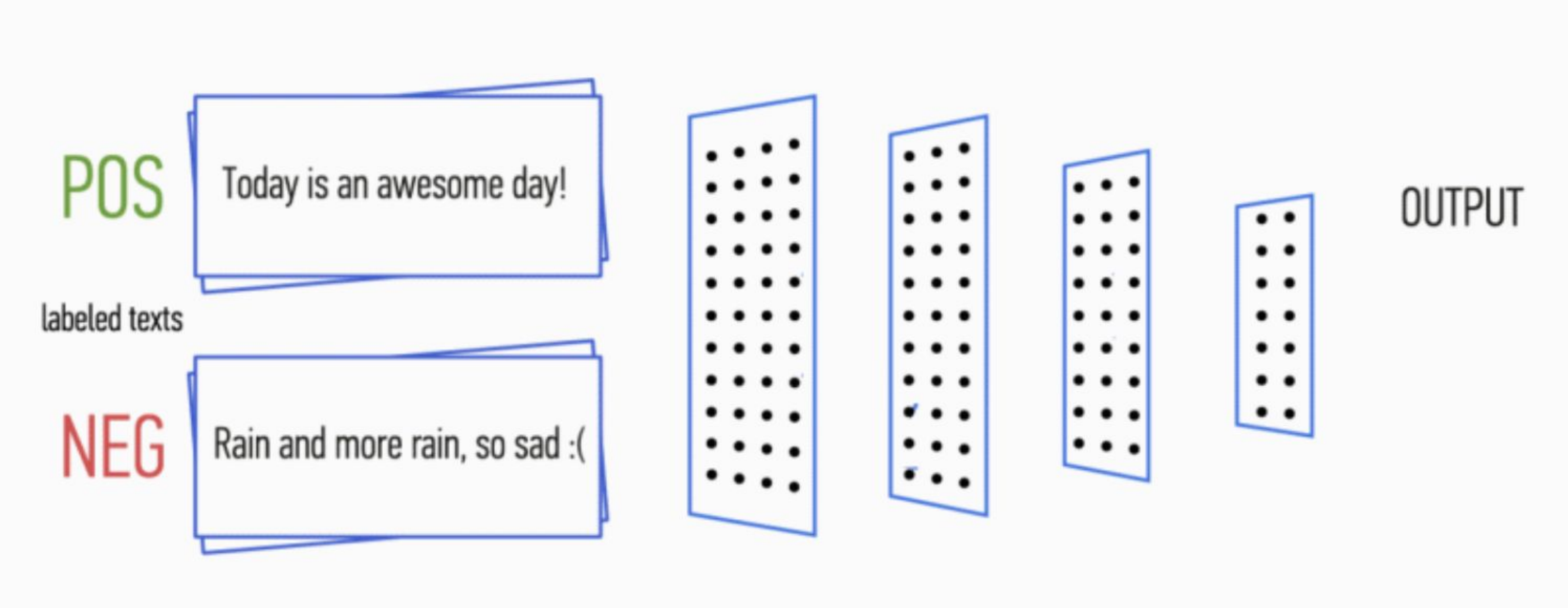
girafe
ai

01

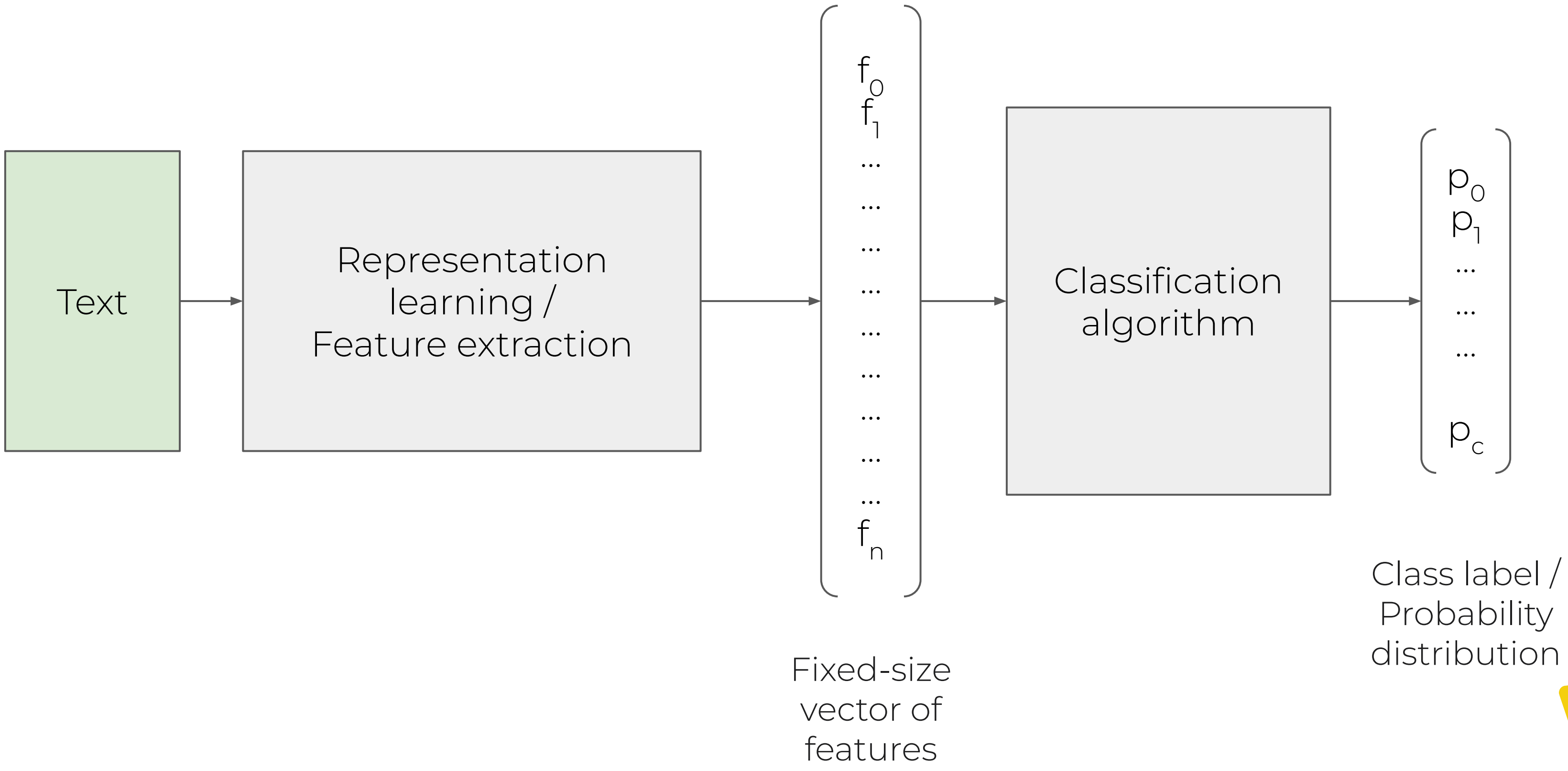
Natural language processing

- Text classification tasks:
  - Sentiment analysis
  - Spam filtering
  - Fake news detection
  - Topic prediction
  - #hashtag prediction

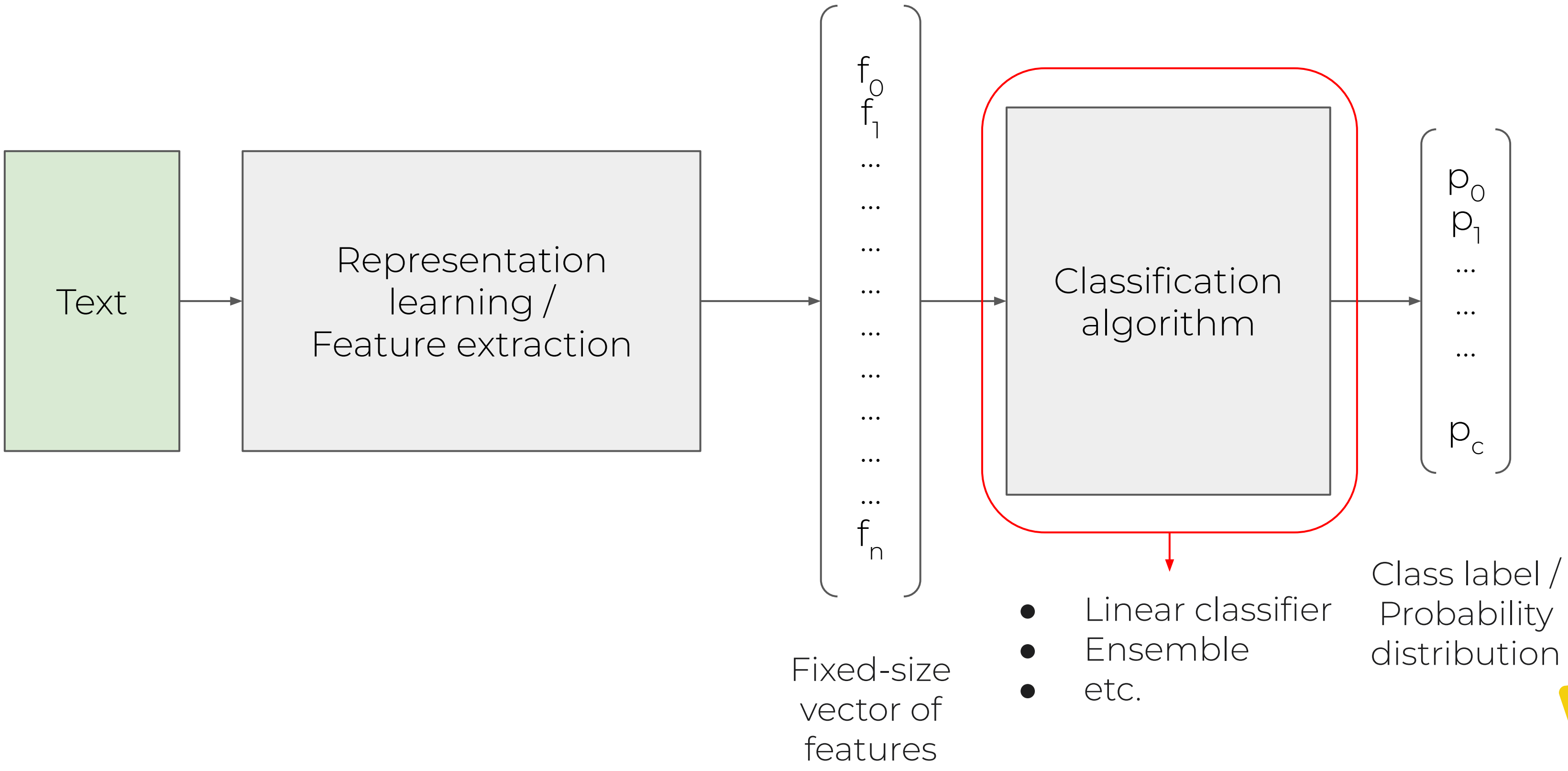
Example: sentiment analysis



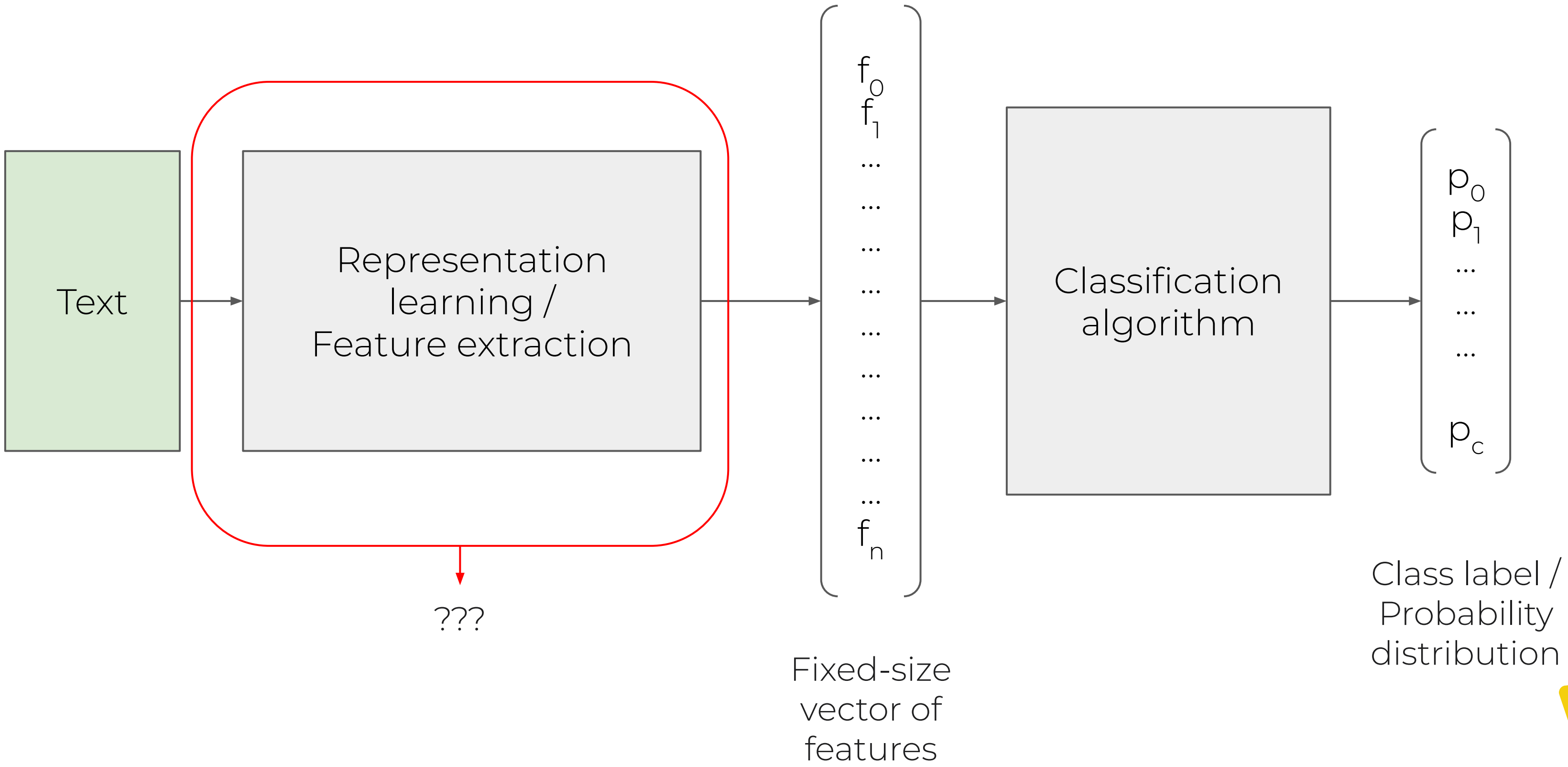
Text classification in general



Text classification in general



Text classification in general



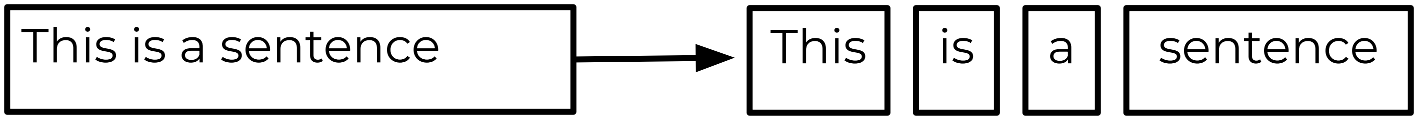
Text preprocessing

girafe
ai

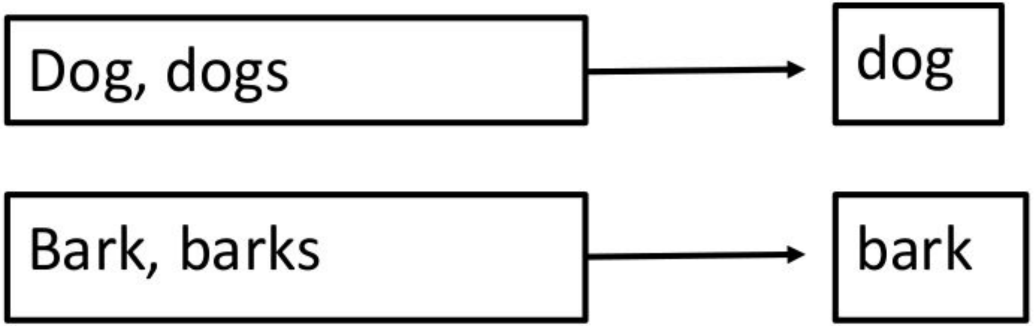
02

Tokenization

- Split the input into tokens
- Tokens:
  - words
  - symbol
  - morpheme
  - …



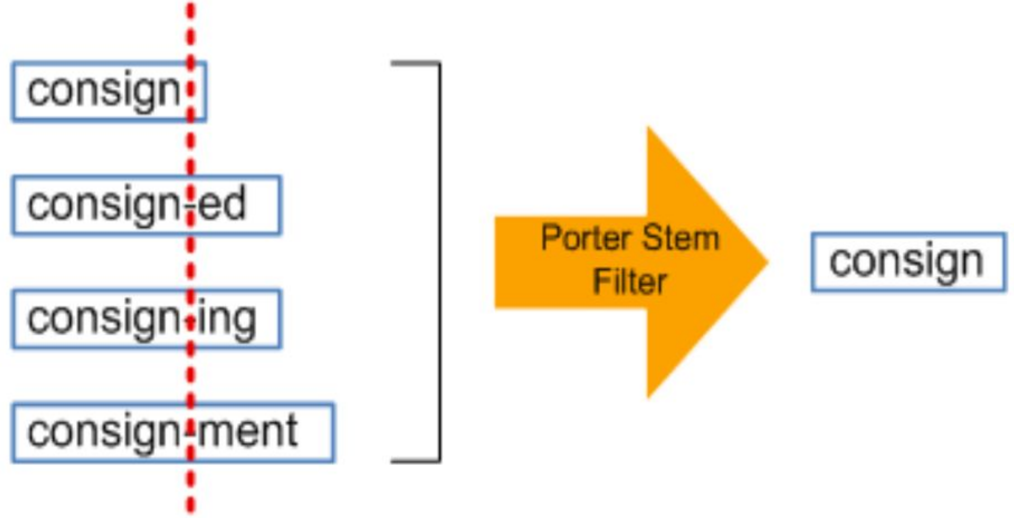
Token normalization



Token normalization

Stemming:

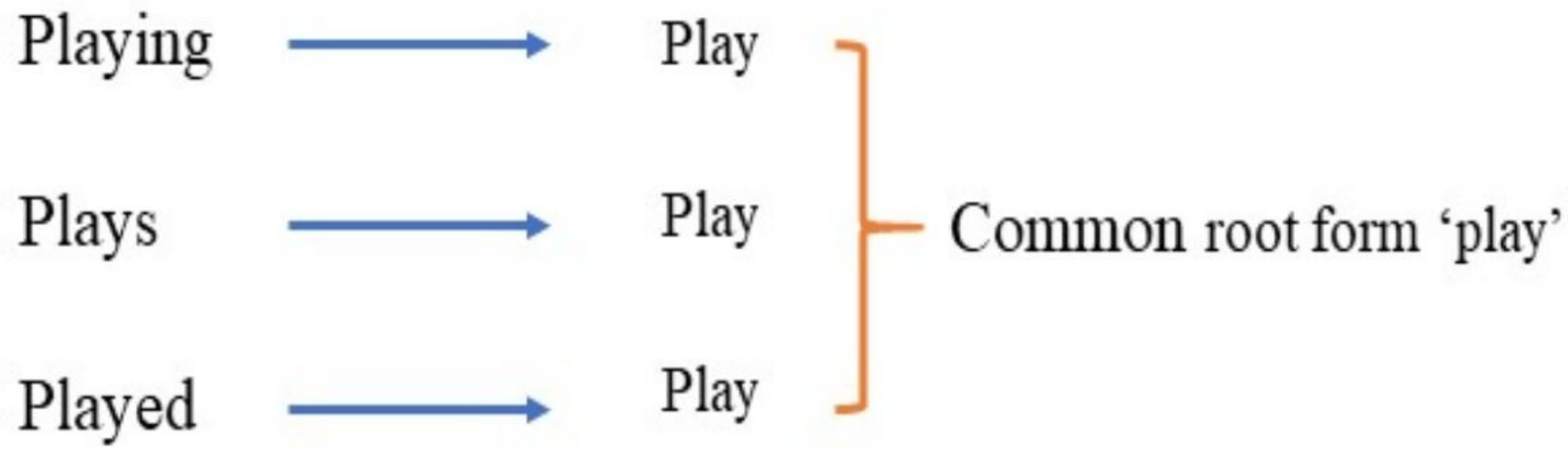
- removing and replacing suffixes
- get to the root of the word (stem)



Token normalization

Lemmatization:

- Get base or dictionary form of a word (lemma)



Stemming: Porter vs Lancaster

Porter stemmer

- Published in 1979
- Base starting option

Lancaster stemmer

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

Snowball stemmer (Porter 2)

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

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

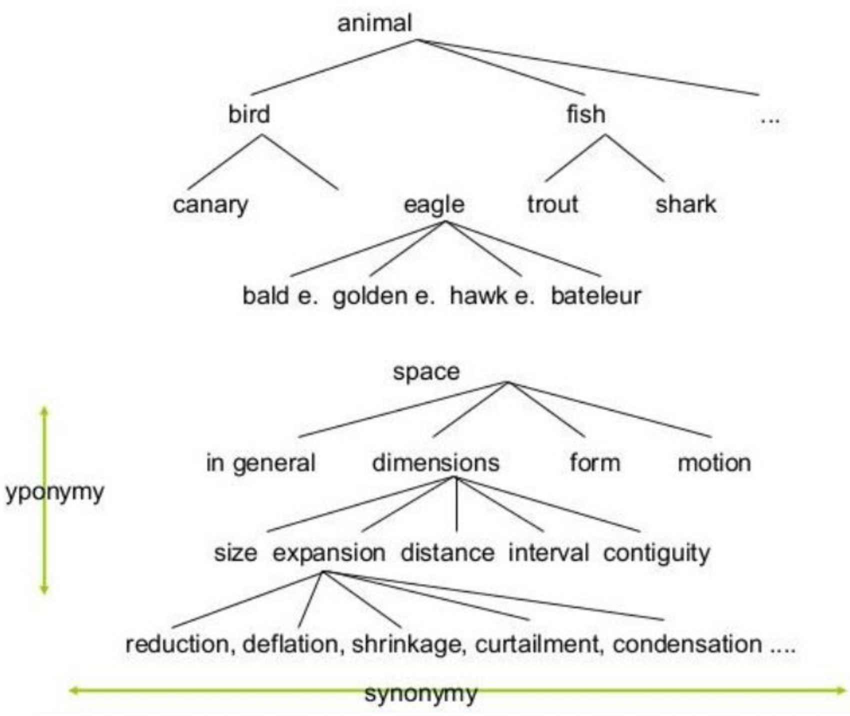
Data → dat

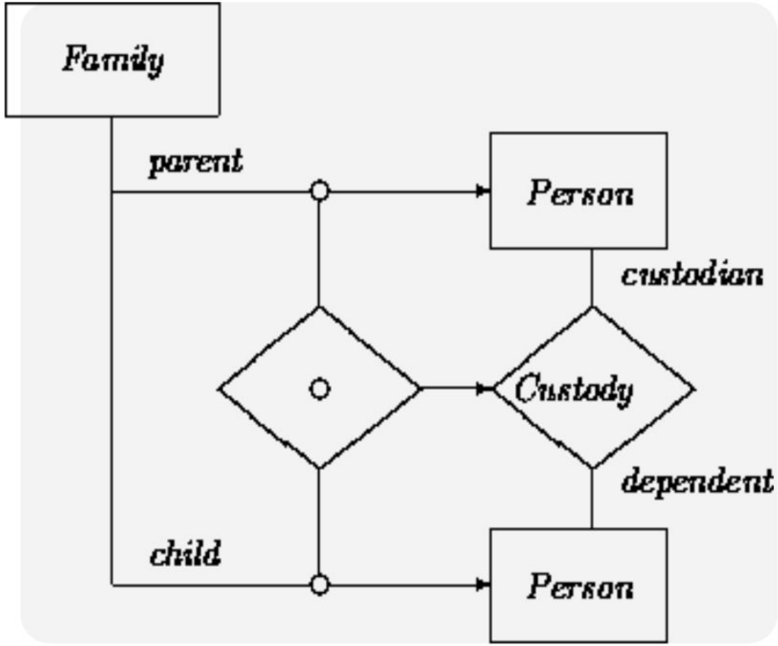
Datum → datu

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

What is WordNet?





link

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

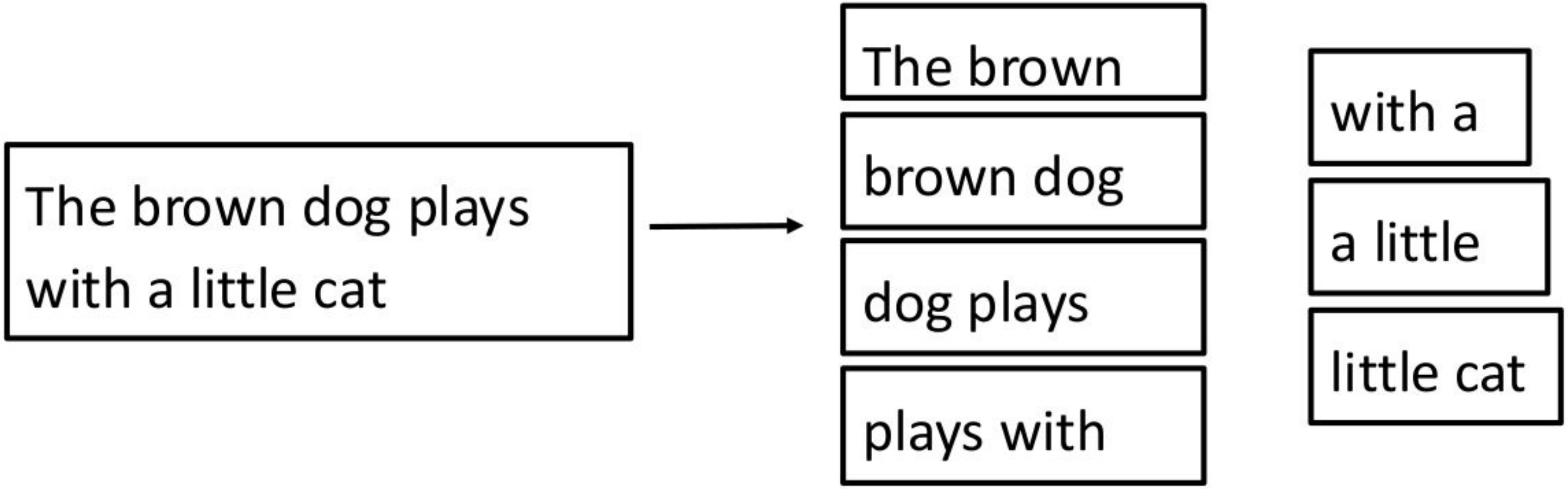
What’s left

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

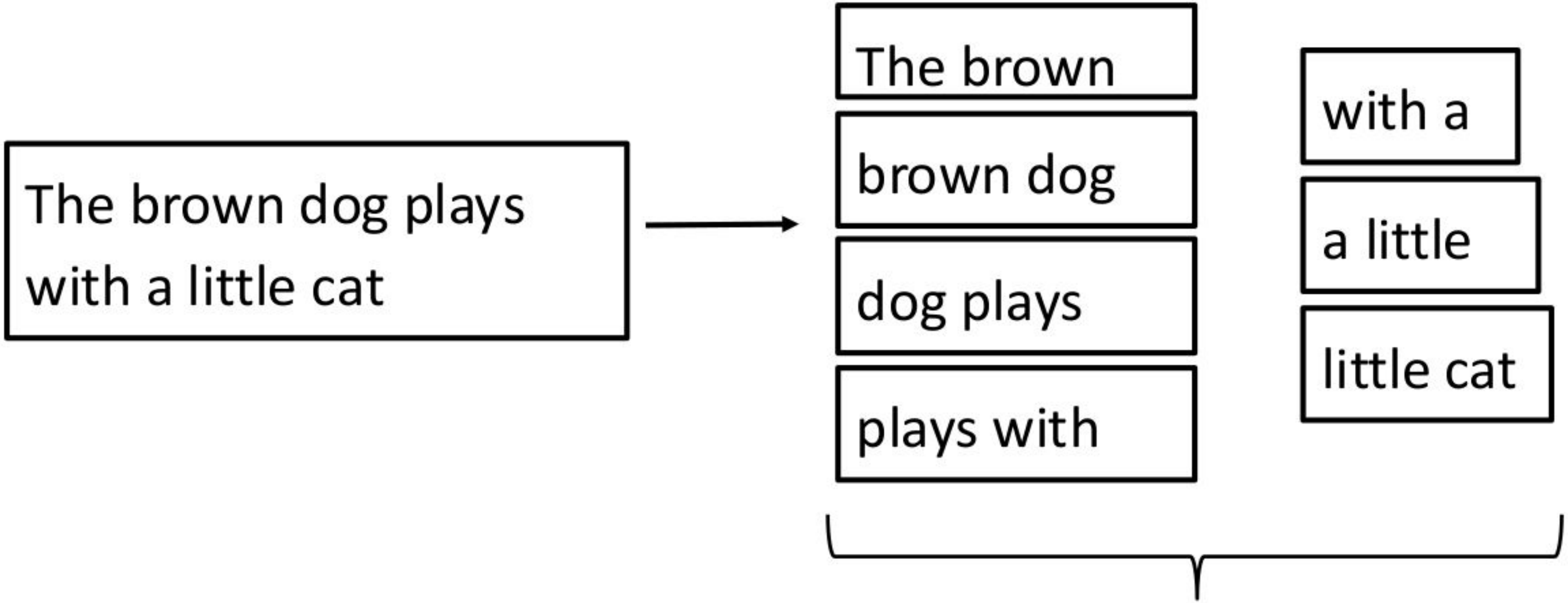
Bag-of-words

How to improve BOW?

- Use n-gramms instead of words!

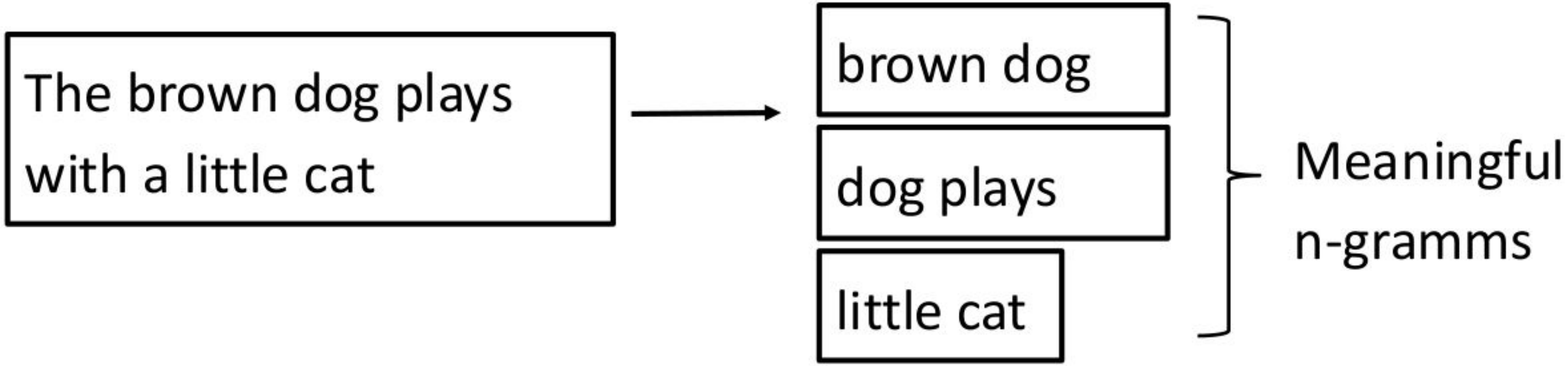


Bag-of-words



Do we need all this bigramms?

Bag-of-words



Meaningful n-gramms are often called collocations

How to detect meaningful n-gramms?

Collocations: first step

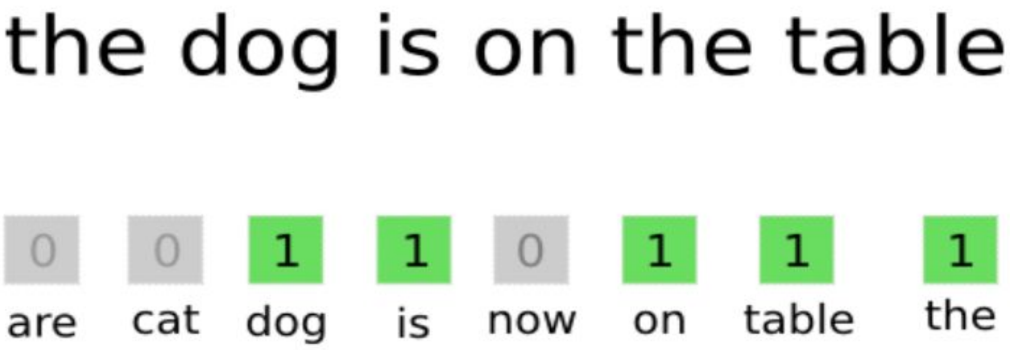
- **Delete:**
  - High-frequency n-gramms
    - Articles, prepositions
    - Auxiliary verbs (to be, to have, etc.)
    - General vocabulary
  - Low-frequency n-gramms
    - Typos
    - Combinations that occur 1-2 times in a text

Feature extraction

girafe
ai

03

Bag-of-words



Problems

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

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

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

04

One-hot vectors

Problems:

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

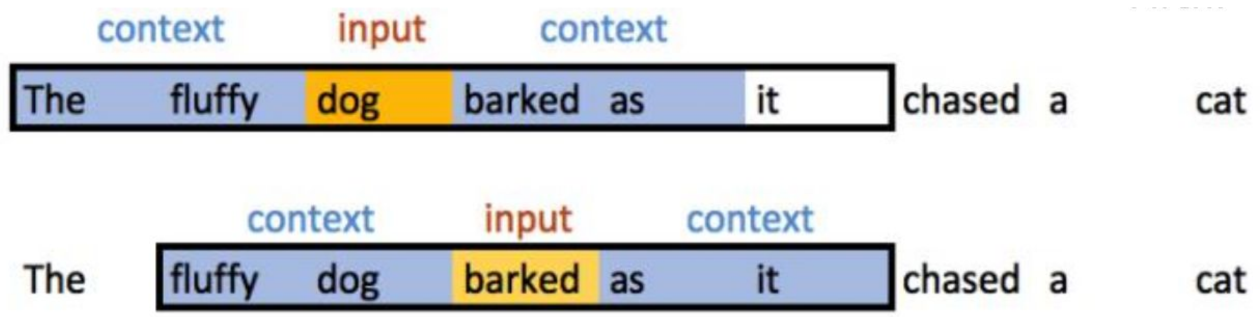
```
       Rome Paris                          word V
Rome   = [1, 0, 0, 0, 0, 0, ..., 0]
Paris  = [0, 1, 0, 0, 0, 0, ..., 0]
Italy  = [0, 0, 1, 0, 0, 0, ..., 0]
France = [0, 0, 0, 1, 0, 0, ..., 0]
```

Distributional semantics

Does vector similarity imply semantic similarity?

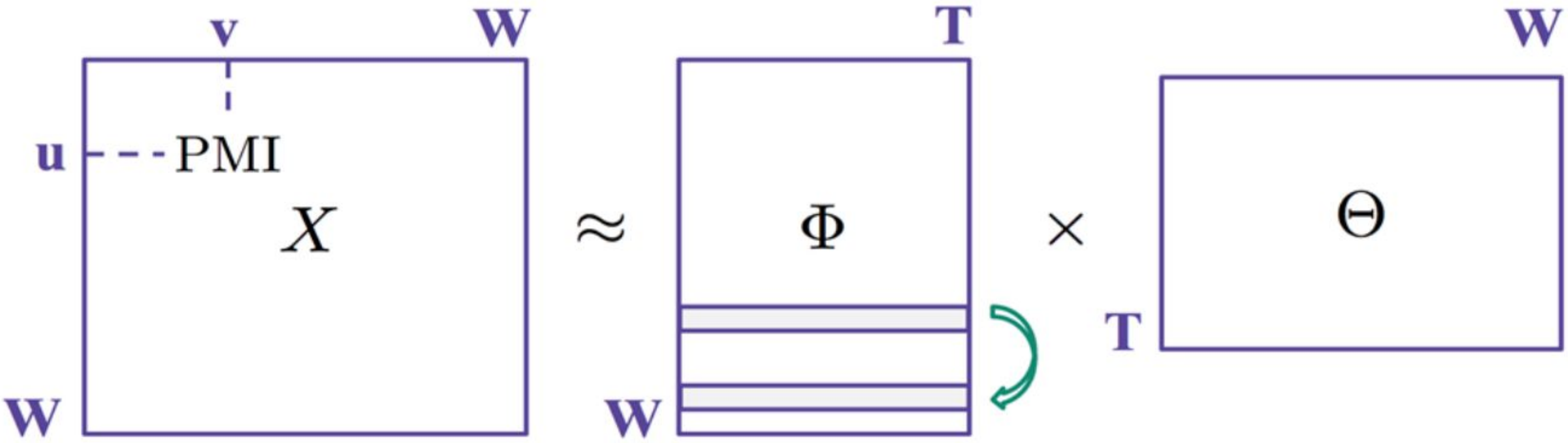
“You shall know a word by the company it keeps”

Firth



Matrix factorization

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



Collocations: first step

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

Collocations: context is all you need

- Co-Occurrence counters in a window of fixed size
  - 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)

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

Learning word vectors

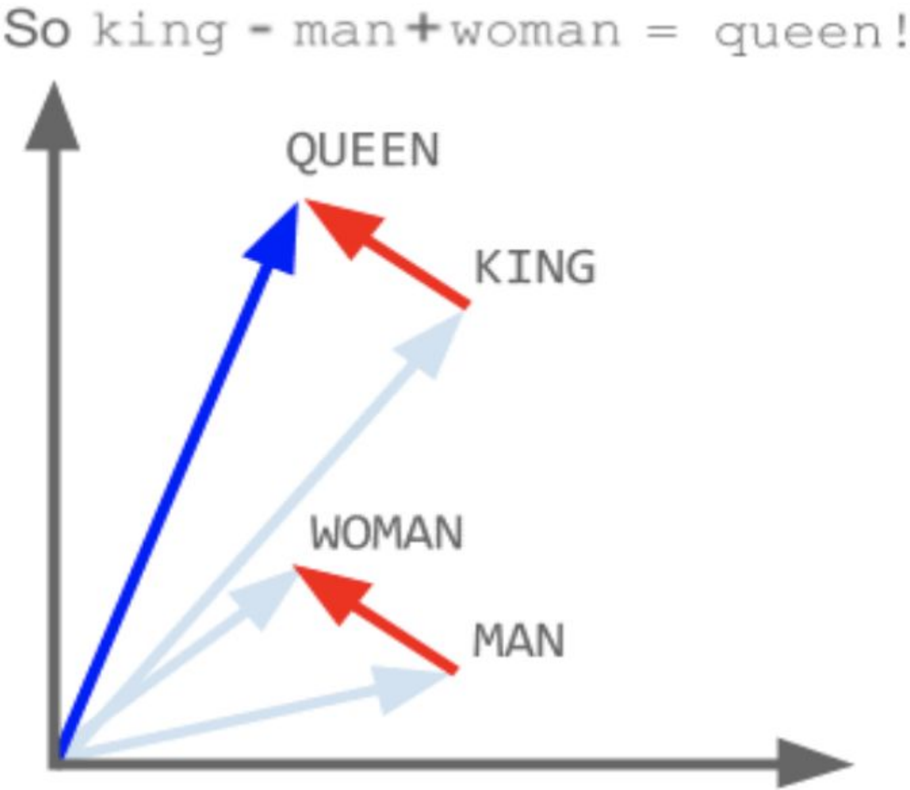
girafe
ai

05

Embeddings: intuition

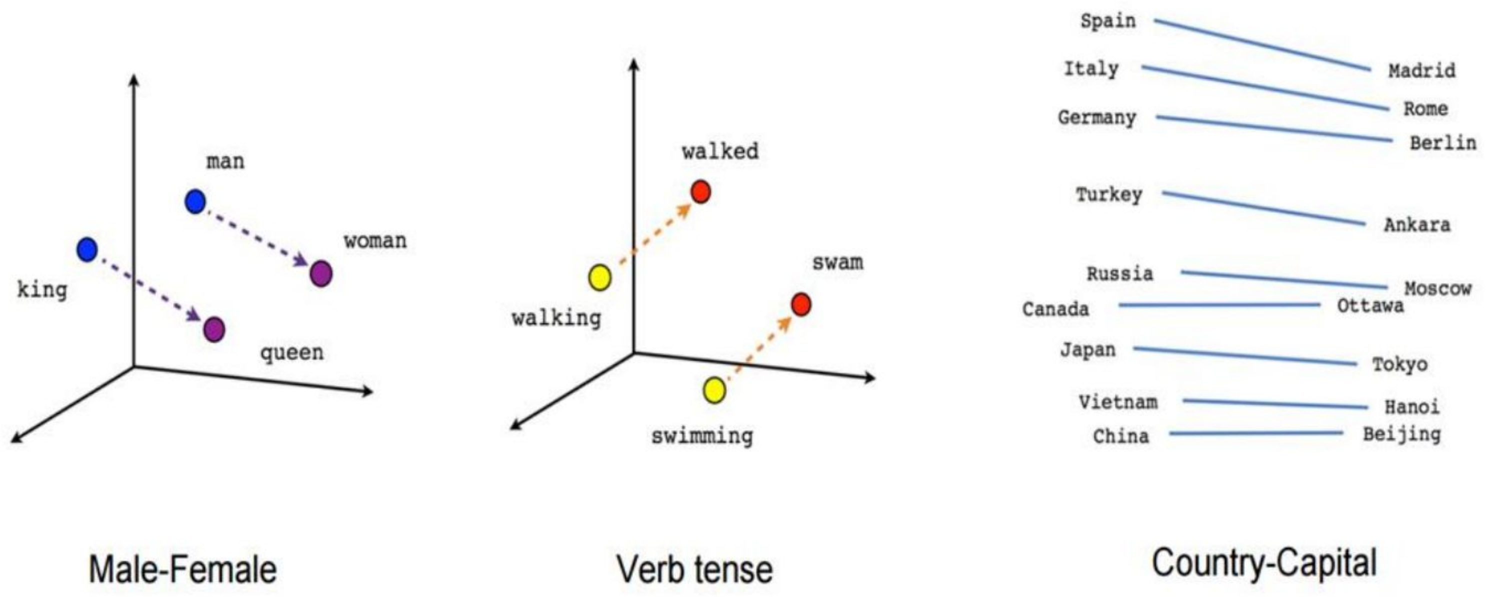
What is king - man + woman?

Embeddings: intuition



Word2vec

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

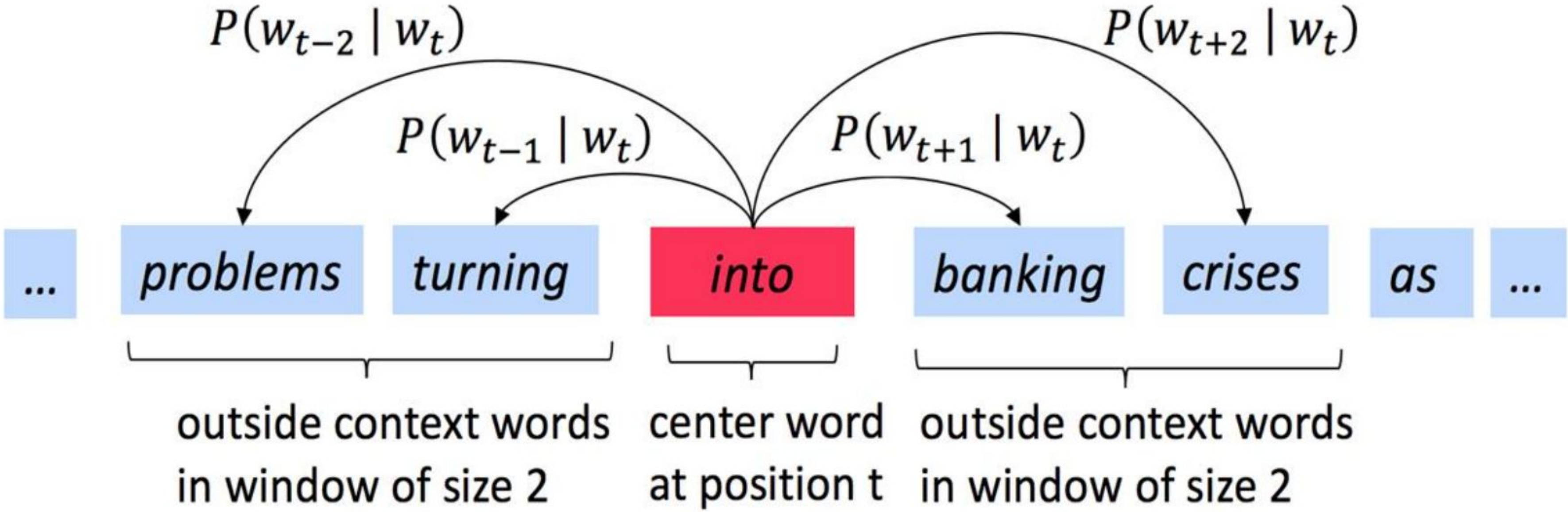


Word2vec

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)

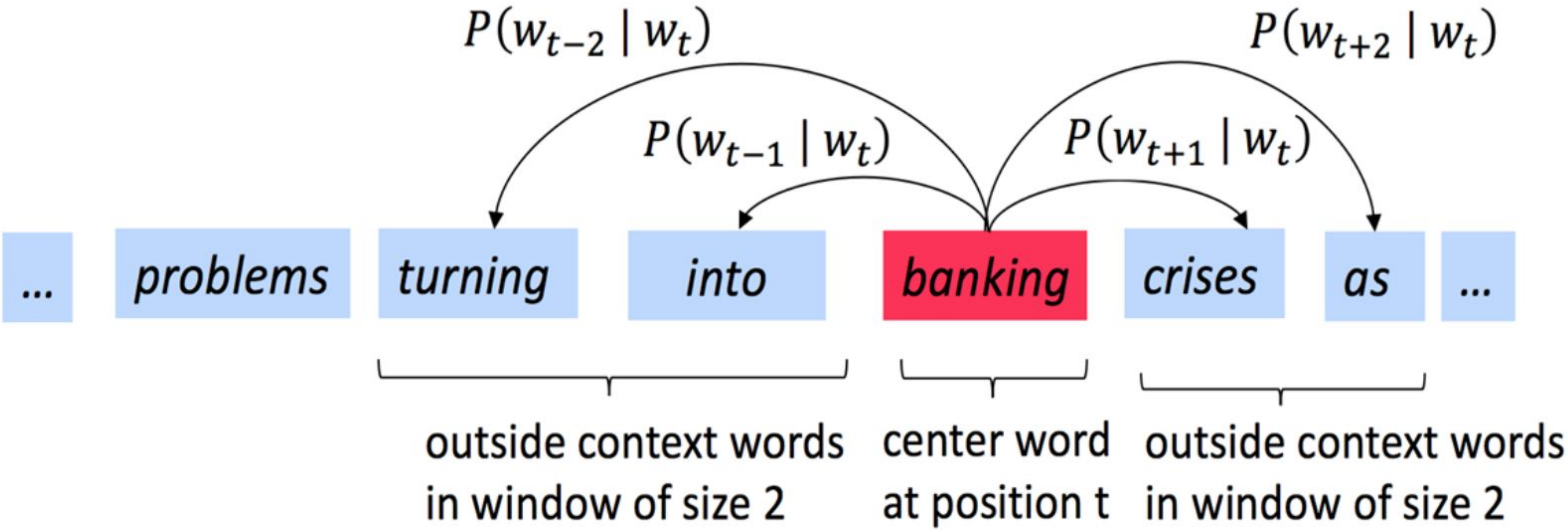
link

Word2vec



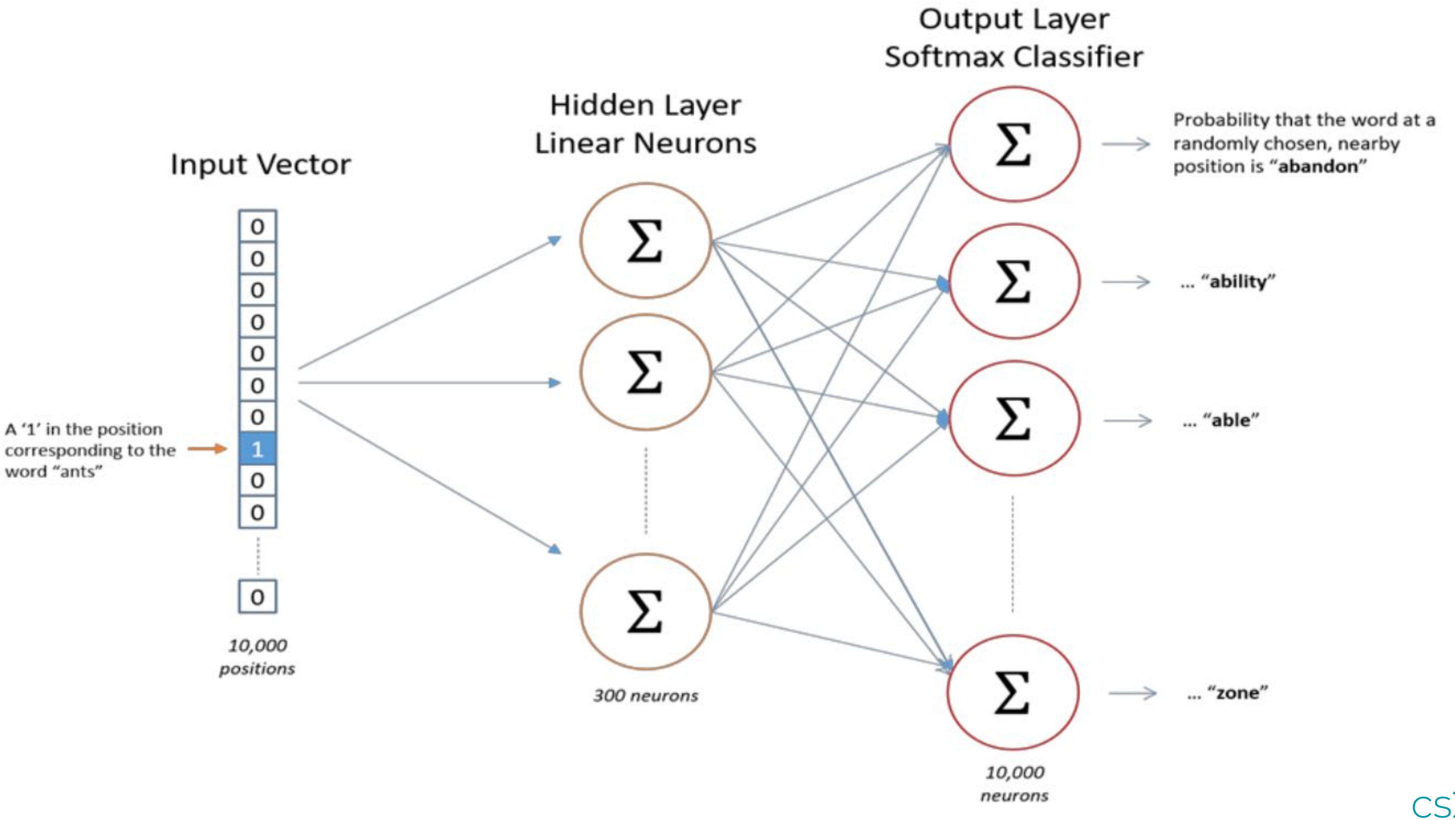
$P(w_{t-2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
$P(w_{t+2} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words
in window of size 2
center word
at position t
outside context words
in window of size 2

Word2vec



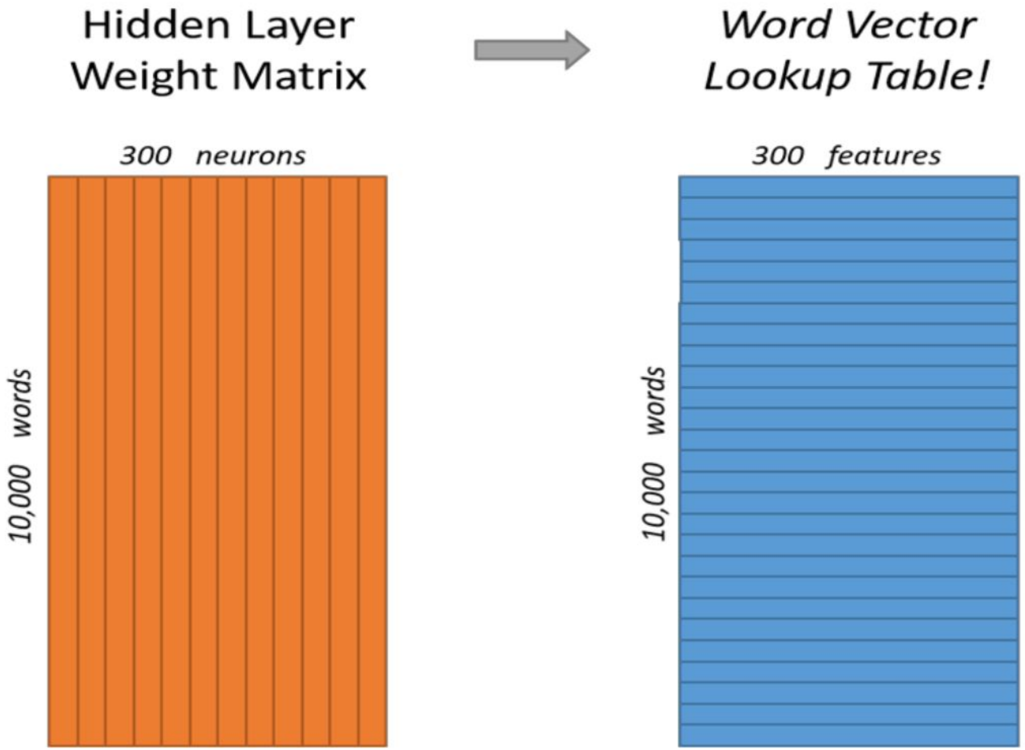
$P(w_{t-2} \mid w_t)$
$P(w_{t-1} \mid w_t)$
$P(w_{t+1} \mid w_t)$
$P(w_{t+2} \mid w_t)$
...
problems
turning
into
banking
crises
as
...
outside context words in window of size 2
center word at position t
outside context words in window of size 2

Word2vec



Output Layer
Softmax Classifier
Hidden Layer
Linear Neurons
Input Vector
0
0
0
0
0
0
0
1
0
0
0
A '1' in the position corresponding to the word "ants"
10,000 positions
300 neurons
10,000 neurons
Probability that the word at a randomly chosen, nearby position is "abandon"
... "ability"
... "able"
... "zone"

Word2vec



Hidden Layer
Weight Matrix
300 neurons
10,000 words
Word Vector
Lookup Table!
300 features
10,000 words

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 IT?

Word2vec

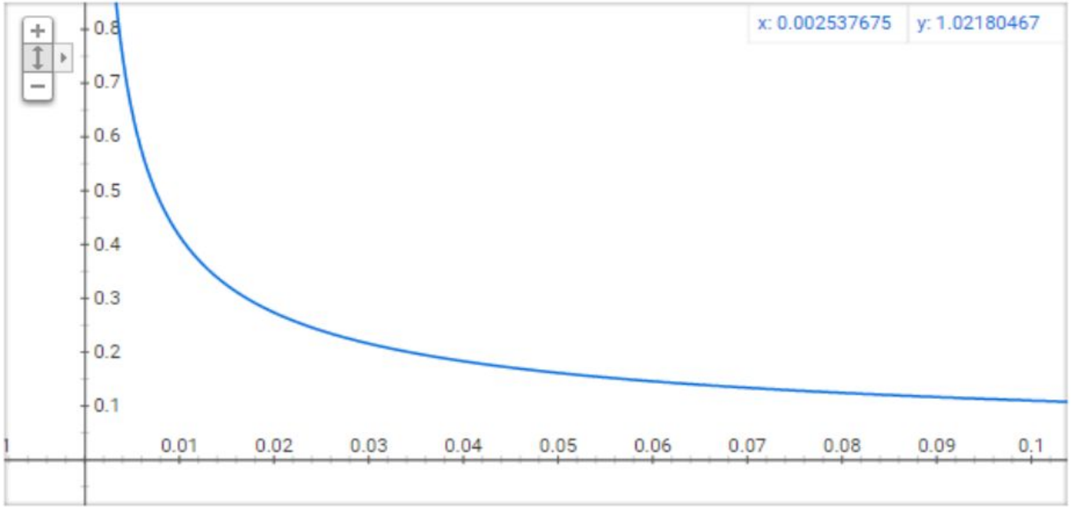
Basic approaches:
1. Treating common word pairs or phrases as single “words” in their model.
2. Subsampling frequent words to decrease the number of training examples.
3. Modifying the optimization objective with a technique they called “Negative Sampling”, which causes each training sample to update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

Word2vec: Subsampling

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:

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

Word2vec: Negative Sampling

- Error is computed only for a few rods. 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)
Predict center word from (bag of) context words

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

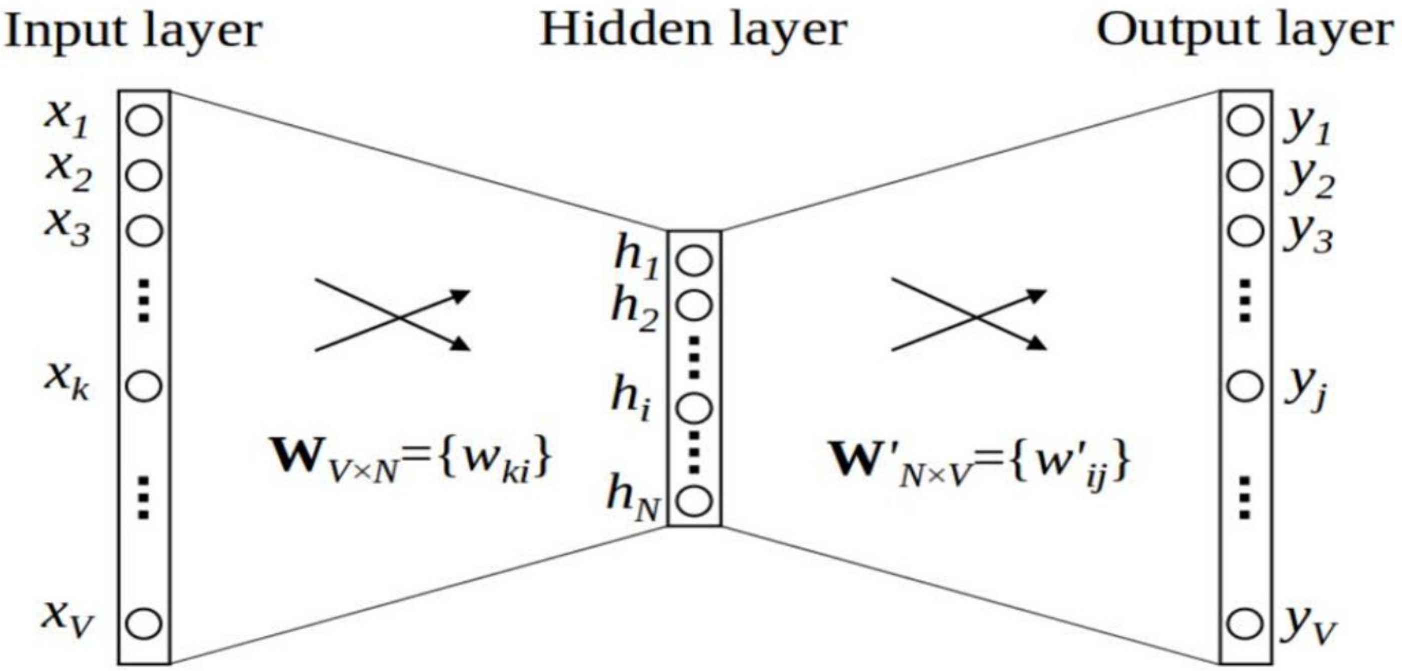
- Predicting one word each
- time
- Relatively fast

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

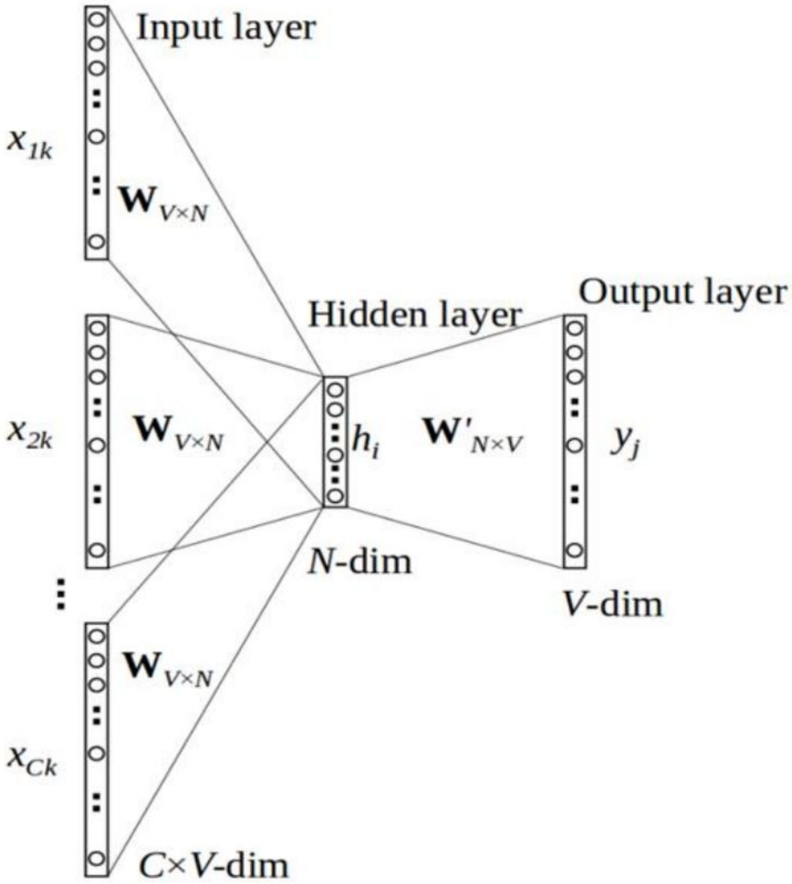
$$p(w_{i-h}, \ldots w_{i+h} | w_i)$$

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

Word2vec: Skip-gram



Word2vec: CBOW



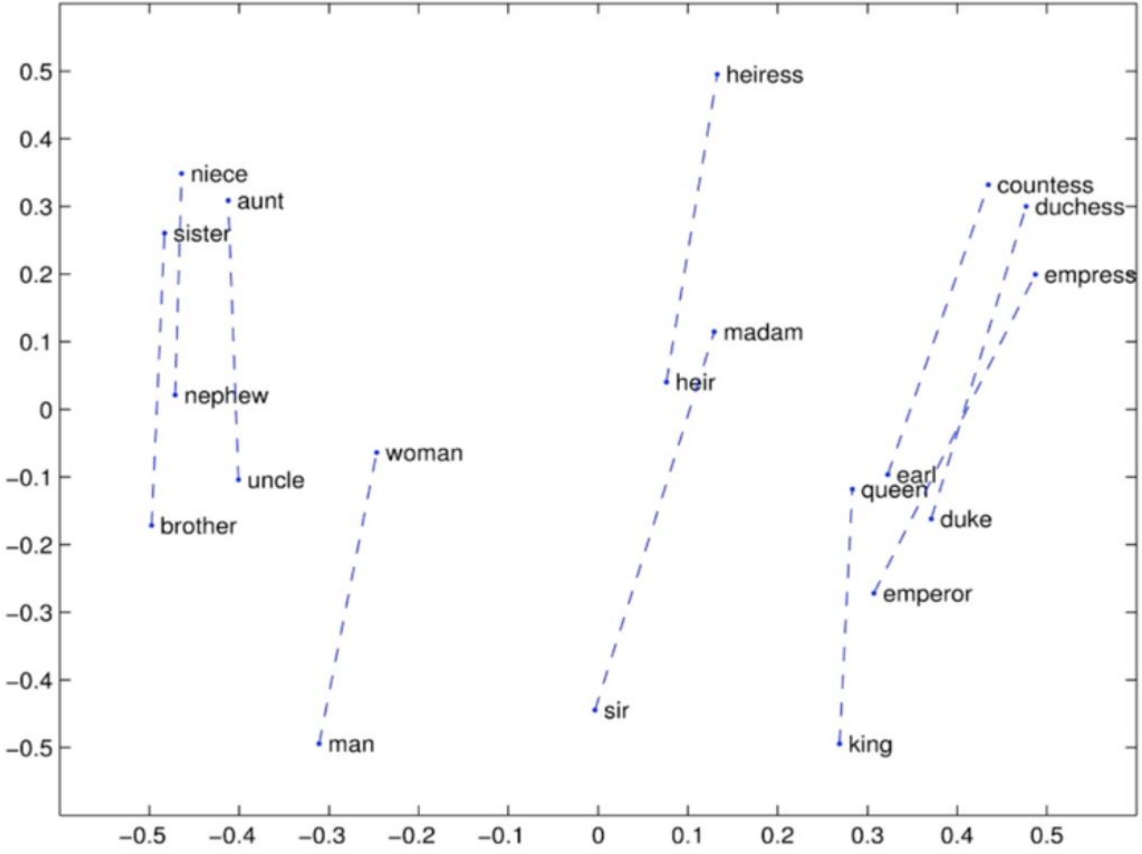
Word2vec: Contrastive

Simplify the approaches above:

Consider only one **positive pair** of words and **a few negative pairs**.

$$\log\sigma(v_{w_o}^{\top} v_{w_t}) + \sum_{\substack{i=0 \\ w_i \in P_n(\omega)}}^{k} \log\sigma(-v_{w_i}^{\top} v_{w_t})$$

GloVe: visualizations

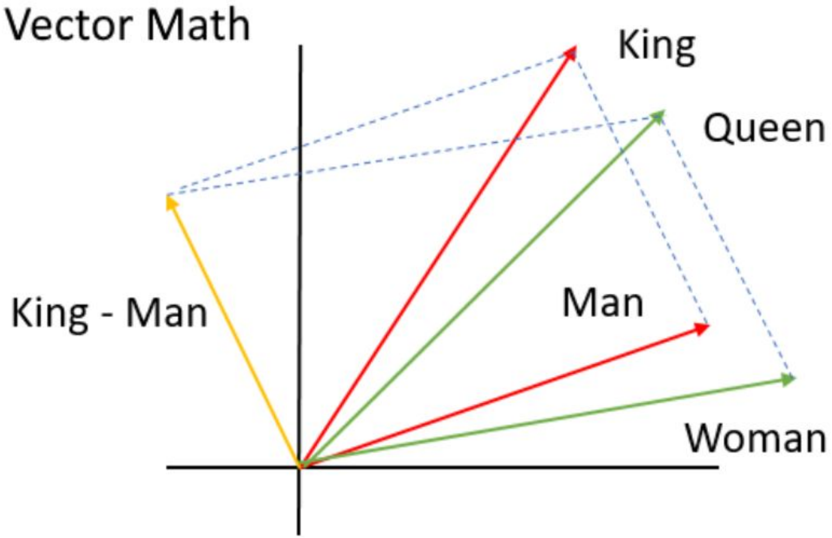


Word2vec: word analogies

King - man + woman = queen

$$\begin{array}{cccc} \text{King} & \text{man} & \text{woman} & \text{queen} \\ \downarrow & \downarrow & \downarrow & \downarrow \\ x & y & y' & target \end{array}$$

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



Summary

- 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)

Revise

1. NLP introduction
2. Text preprocessing
3. Feature extraction:
    a. Bag-of-Words
    b. Bag-of-Ngrammes
    c. TF-IDF
4. Word embeddings
5. Word2vec

Thanks for attention!

Questions?

girafe
ai