ML course

Advanced ML

Neychev Radoslav

2019, Moscow, Russia

Course syllabus:

3 main blocks:

Course syllabus:

3 main blocks:

1. Natural Language Processing
    a. Language models
    b. Text generation
    c. Neural machine translation

Course syllabus:

3 main blocks:

1. Natural Language Processing
2. Reinforcement Learning
   a. Simple approaches to non-gradient optimization
   b. Q-learning, SARSA
   c. DQN
   d. REINFORCE, AAC

Course syllabus:

3 main blocks:

1. Natural Language Processing
2. Reinforcement Learning

Course syllabus:

3 main blocks:

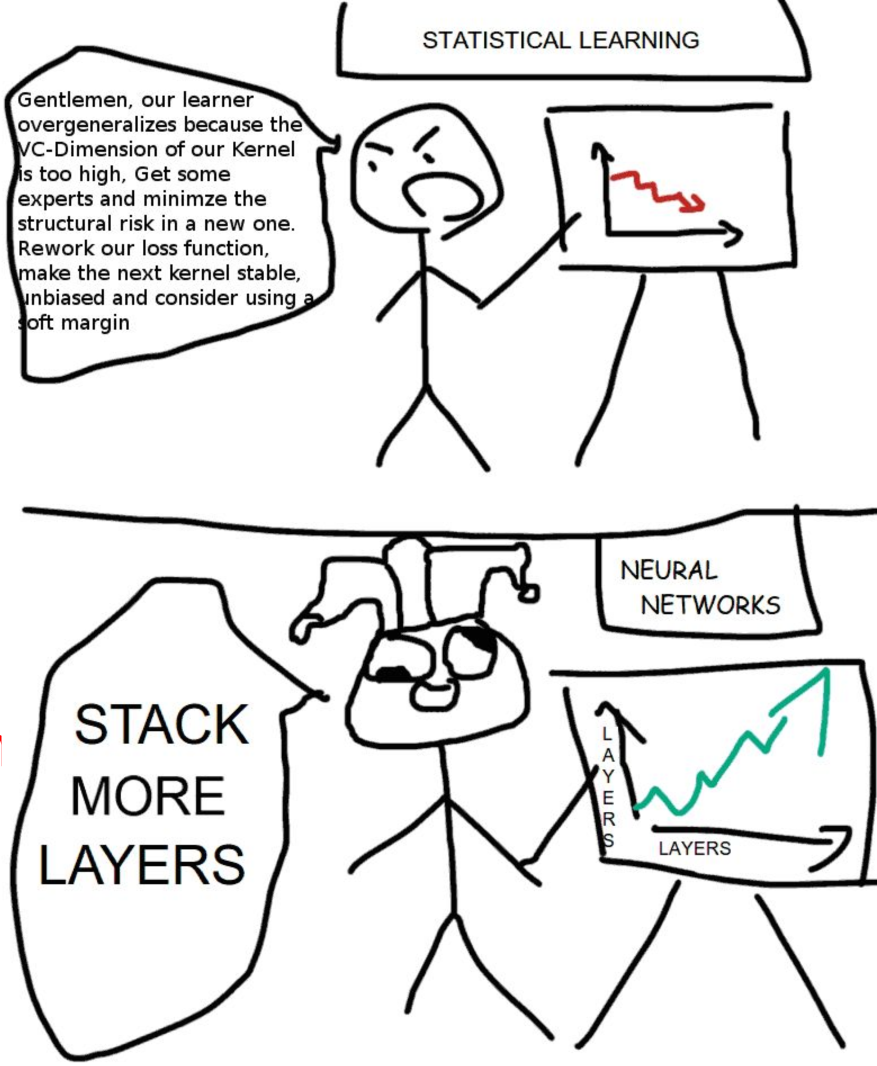
1. Natural Language Processing
2. Reinforcement Learning
3. DL & CV interesting cases
    a. GAN, VAE
    b. Style transfer
    c. Object detection & segmentation

Course syllabus:

3 main blocks:

1. Natural Language Processing
2. Reinforcement Learning
3. DL & CV interesting cases

All flavored with Deep Learnin



Rules of play

1. Homeworks:
   a. Labs:
      i. Modular structure with several milestones
   b. Tiny homeworks
      i. Provided each week.
2. Tests:
   a. Small tests at the beginning of each day
3. Opportunities
   a. Internships/Interviews in tech companies (if it works :)
   b. Fun

Technical stuff

- Python 3.6+
  - Miniconda is recommended for env managing
- Supported platforms: Linux/macOS/docker
  - Anything else on your own risk
- Course chat in Telegram
- All materials are available at github

This course is using materials and generally based on such courses as:

- Stanford:
  - CS224n Natural Language Processing
  - CS234n Reinforcement Learning
- Yandex School of Data Analysis:
  - Practical RL
  - NLP course
- Berkeley:
  - CS188x Intro to AI
  - CS294-112 Deep Reinforcement Learning

Special thanks to the teams for developing the materials and making them available online

Word Representations

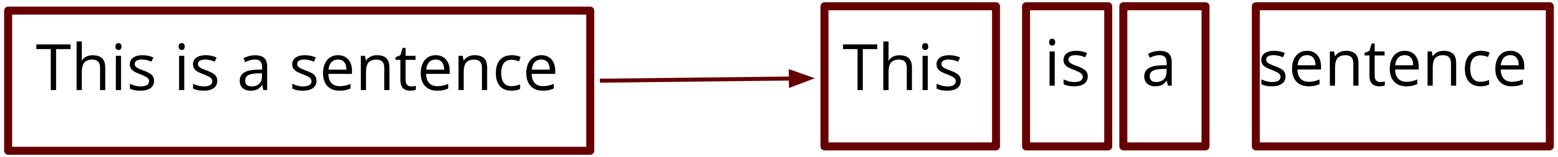
Agenda

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

Text Preprocessing

Preprocessing

- Tokenization: split the input into tokens



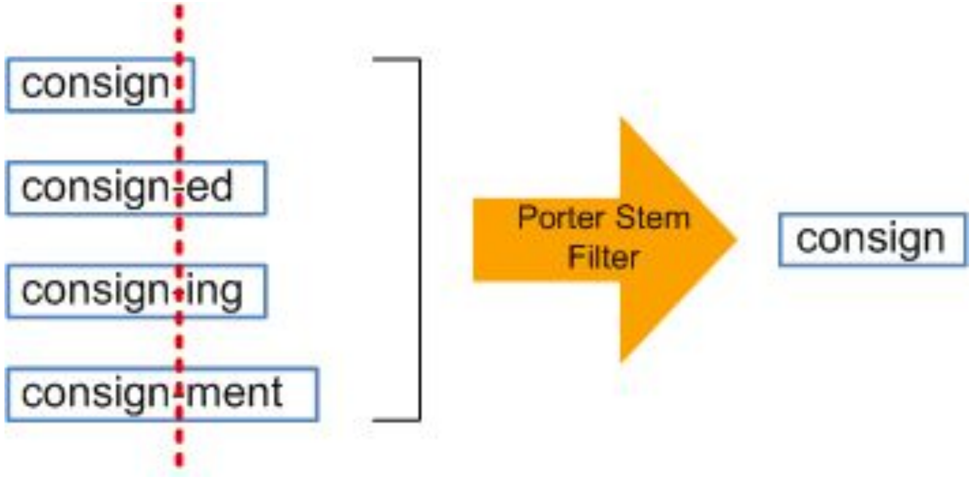
Preprocessing

- Token normalization

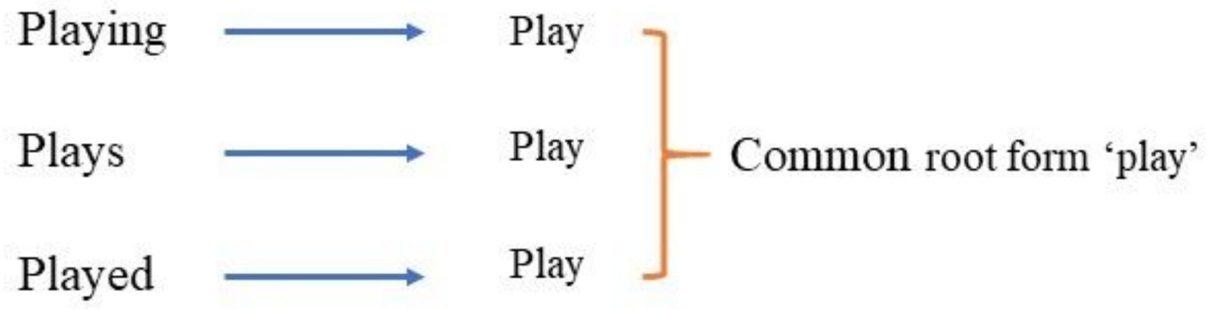
Dog, dogs → dog

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

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?**

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

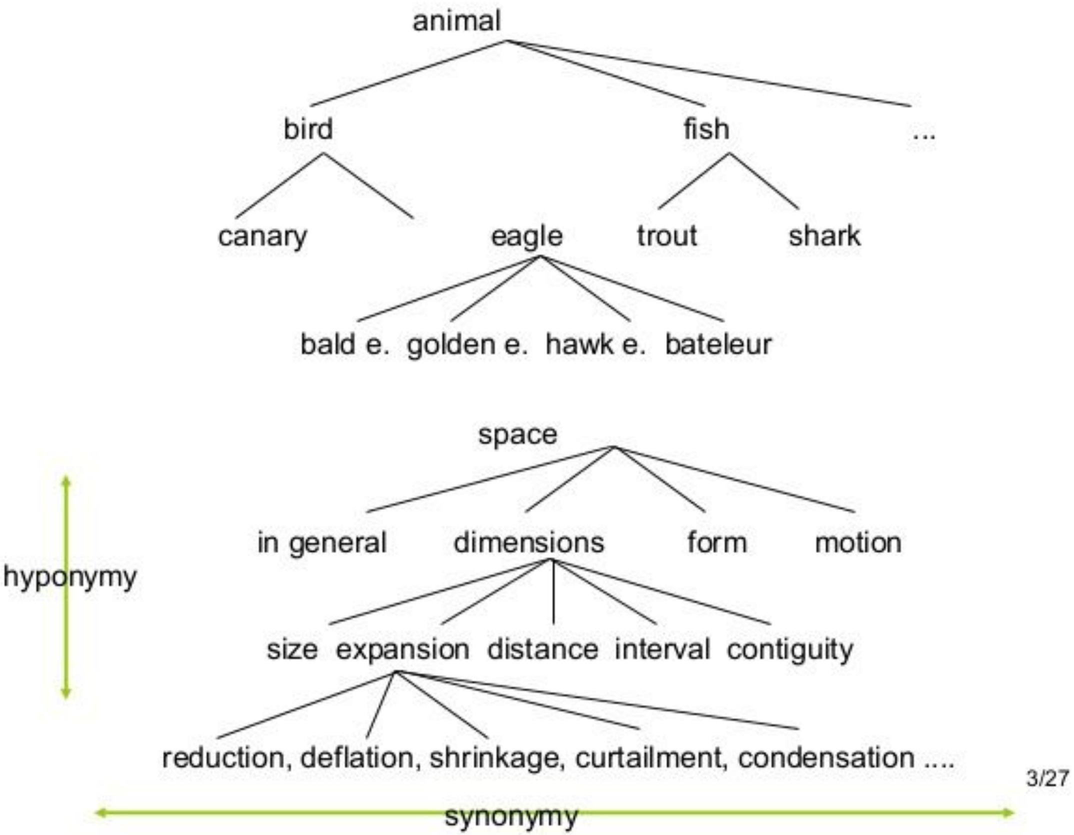
Data → dat

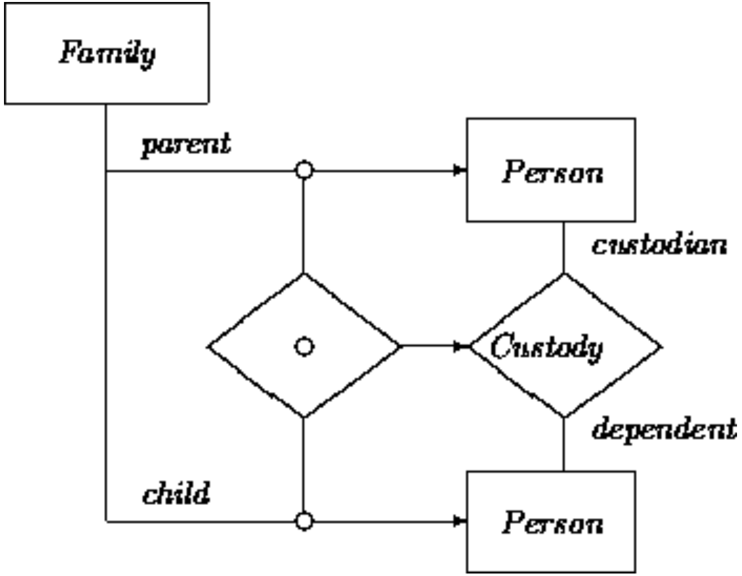
Datum → datu

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?





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

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

Bag-of-Words

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

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

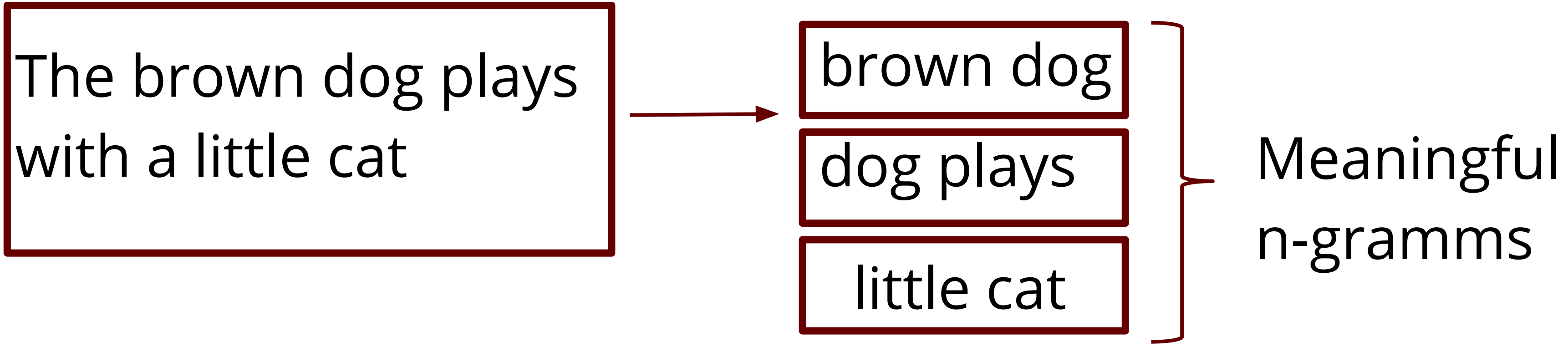
with a

a little

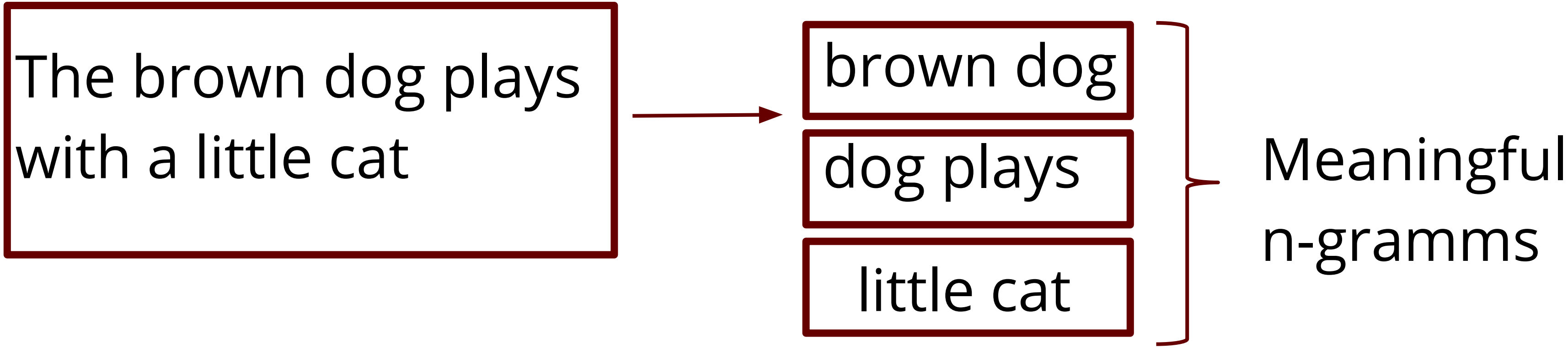
little cat

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

Bag-of-Words



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

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

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

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)**

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

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)**

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

- Use statistics:
  - T-criterion

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

H_0: 'social media' occurs with probability:

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

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

Collocations

- Use statistics:
  - Chi-squared

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

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

$$O_{ij}\ \ from\ \ table$$

	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)

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 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.

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

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

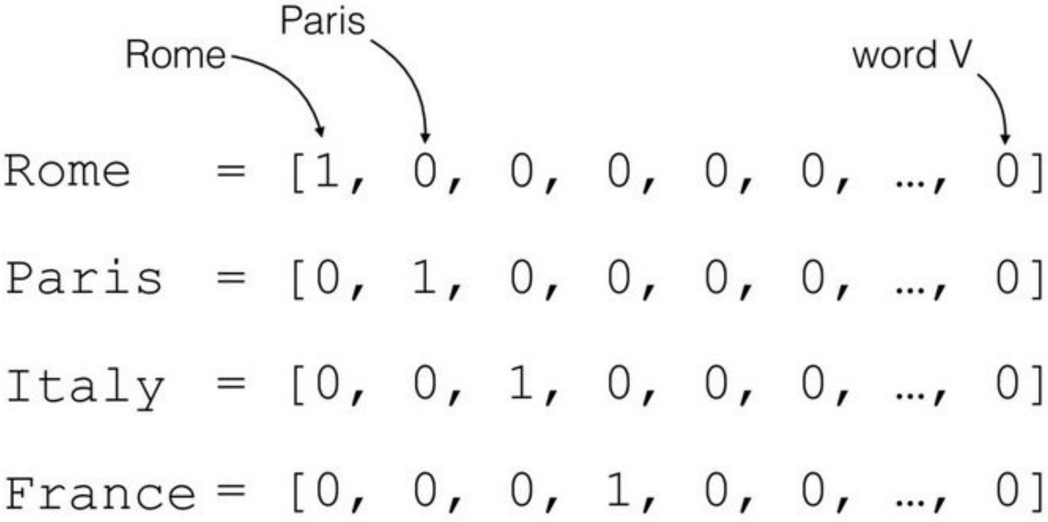
Word	TF		IDF	TF * IDF	
	A	B		A	B
The	1/7	1/7	log(2/2)=0	0	0
Car	1/7	0	log(2/1)=0.3	0.043	0
Truck	0	1/7	log(2/1)=0.3	0	0.043
Is	1/7	1/7	log(2/2)=0	0	0
Driven	1/7	1/7	log(2/2)=0	0	0
On	1/7	1/7	log(2/2)=0	0	0
The	1/7	1/7	log(2/2)=0	0	0
Road	1/7	0	log(2/1)=0.3	0.043	0
Highway	0	1/7	log(2/1)=0.3	0	0.043

TF-IDF example: much easier

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

Word Embeddings

- **One-hot vectors:**



One-hot vectors

$$\begin{array}{lcccccccc}
 & \text{Rome} & \text{Paris} & & & & & & \text{word V} \\
\texttt{Rome} = & [1, & 0, & 0, & 0, & 0, & 0, & \dots, & 0] \\
\texttt{Paris} = & [0, & 1, & 0, & 0, & 0, & 0, & \dots, & 0] \\
\texttt{Italy} = & [0, & 0, & 1, & 0, & 0, & 0, & \dots, & 0] \\
\texttt{France} = & [0, & 0, & 0, & 1, & 0, & 0, & \dots, & 0]
\end{array}$$

Problems:

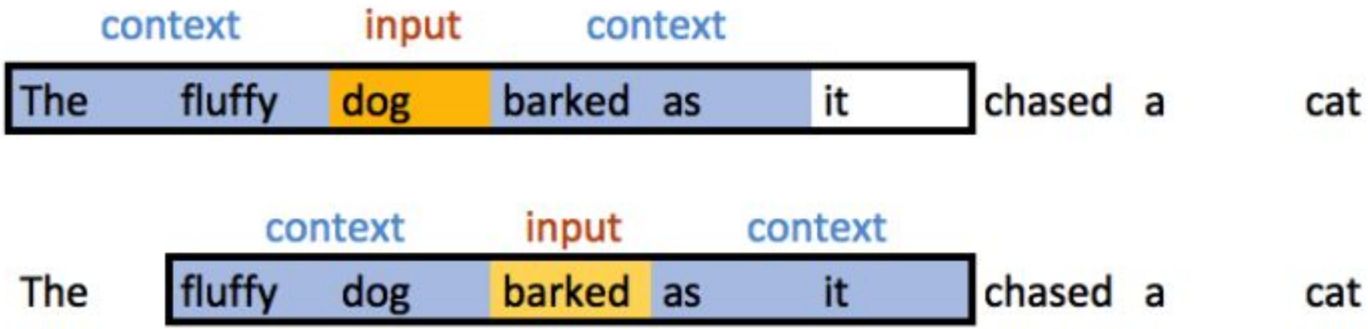
- Huge vectors
- VERY sparse
- No semantics or word similarity information included

Distributional semantics

Does vector similarity imply semantic similarity?

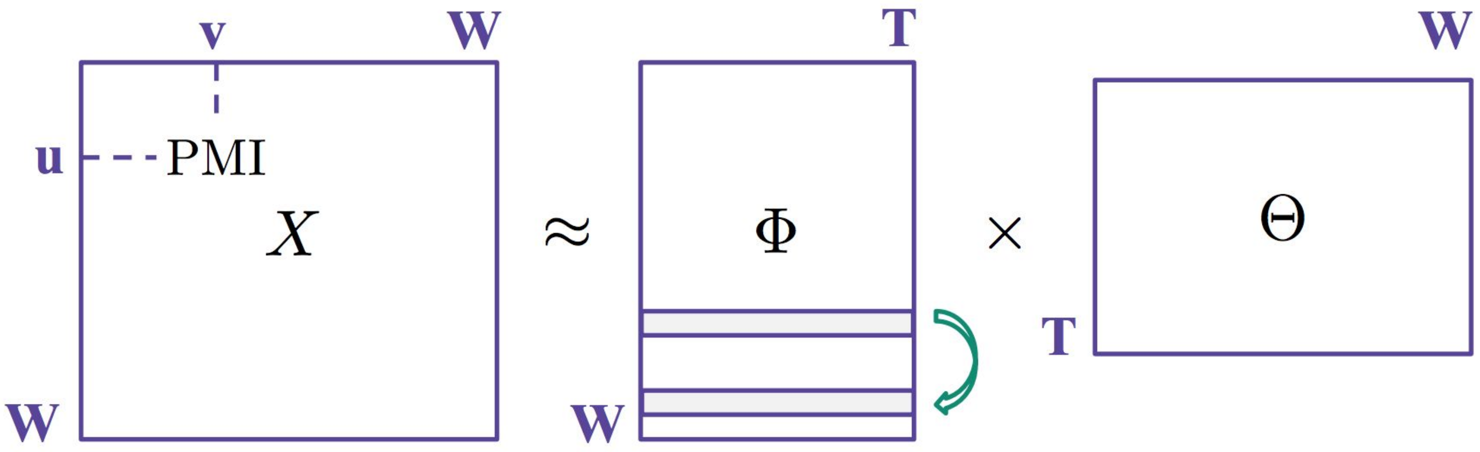
"You shall know a word by the company it keeps"

Firth, 1957



Word representations via matrix factorization

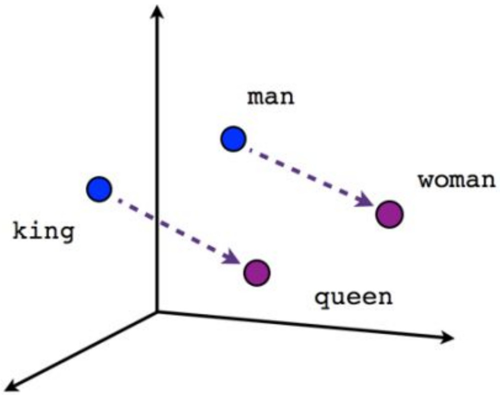
- **Input: PMI, word coocurrences, etc.**
- **Method: dimensionality reduction (SVD)**
- **Output: word similarities**



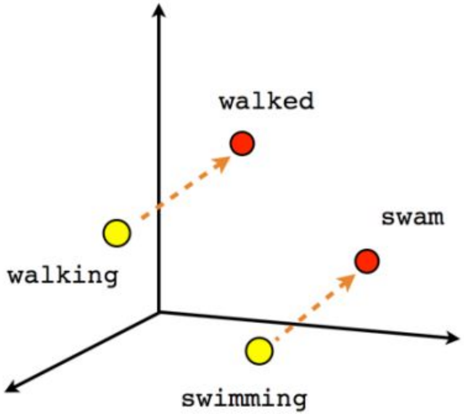
Why not to learn word vectors?

Word2vec

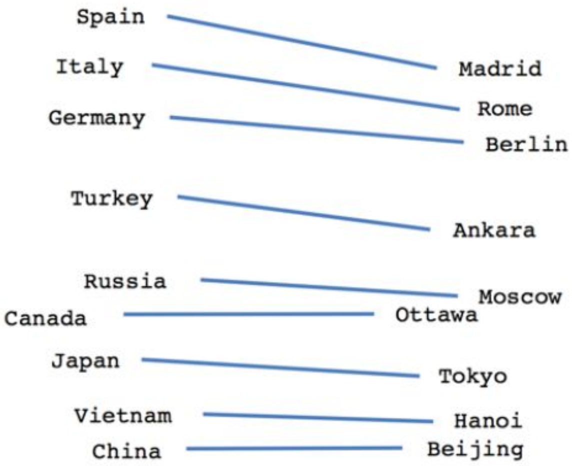
- **Word2vec** (Mikolov et al. 2013) - a framework for learning word embeddings



Male-Female



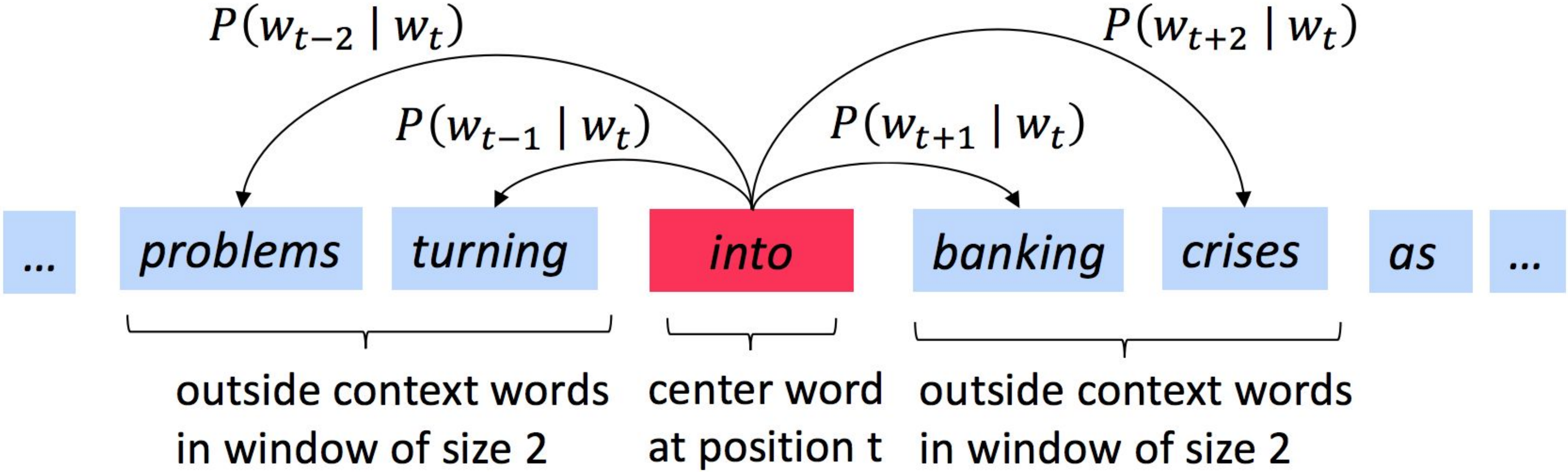
Verb tense



Country-Capital

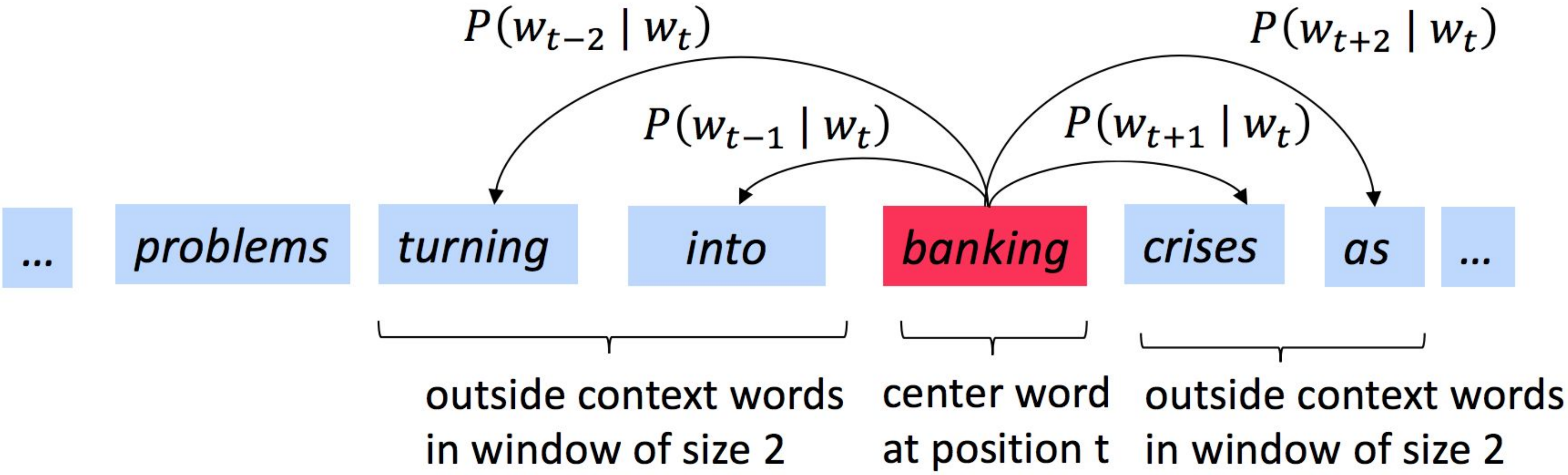
- Main idea:
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context (“outside”) words o
  - Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

Word2vec



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

Word2vec



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

Word2vec

For each position t predict context words within a window of fixed size m, given center word.

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \ne 0}} P\left(w_{t+j} \mid w_t; \theta\right)$$

Word2vec

For each position t predict context words within a window of fixed size m, given center word.

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

Let’s get rid of multiplication:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$

For each position t predict context words within a window of fixed size m, given center word.

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\neq 0}} \underbrace{\log P\left(w_{t+j} \mid w_t; \theta\right)}_{\text{How to calculate this?}}$$

For each position t predict context words within a window of fixed size m, given center word.

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \underbrace{\log P\left(w_{t+j} \mid w_t; \theta\right)}_{\text{How to calculate this?}}$$

We will use two types of vectors:

- v_w when w is a central word
- u_w when w is a context word

Word2vec

We will use two types of vectors:

- v_w when w is a central word
- u_w when w is a context word

Then for a center word c and a context word o:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2vec

- v_w when w is a central word
- u_w when w is a context word

Exponentiation makes anything positive

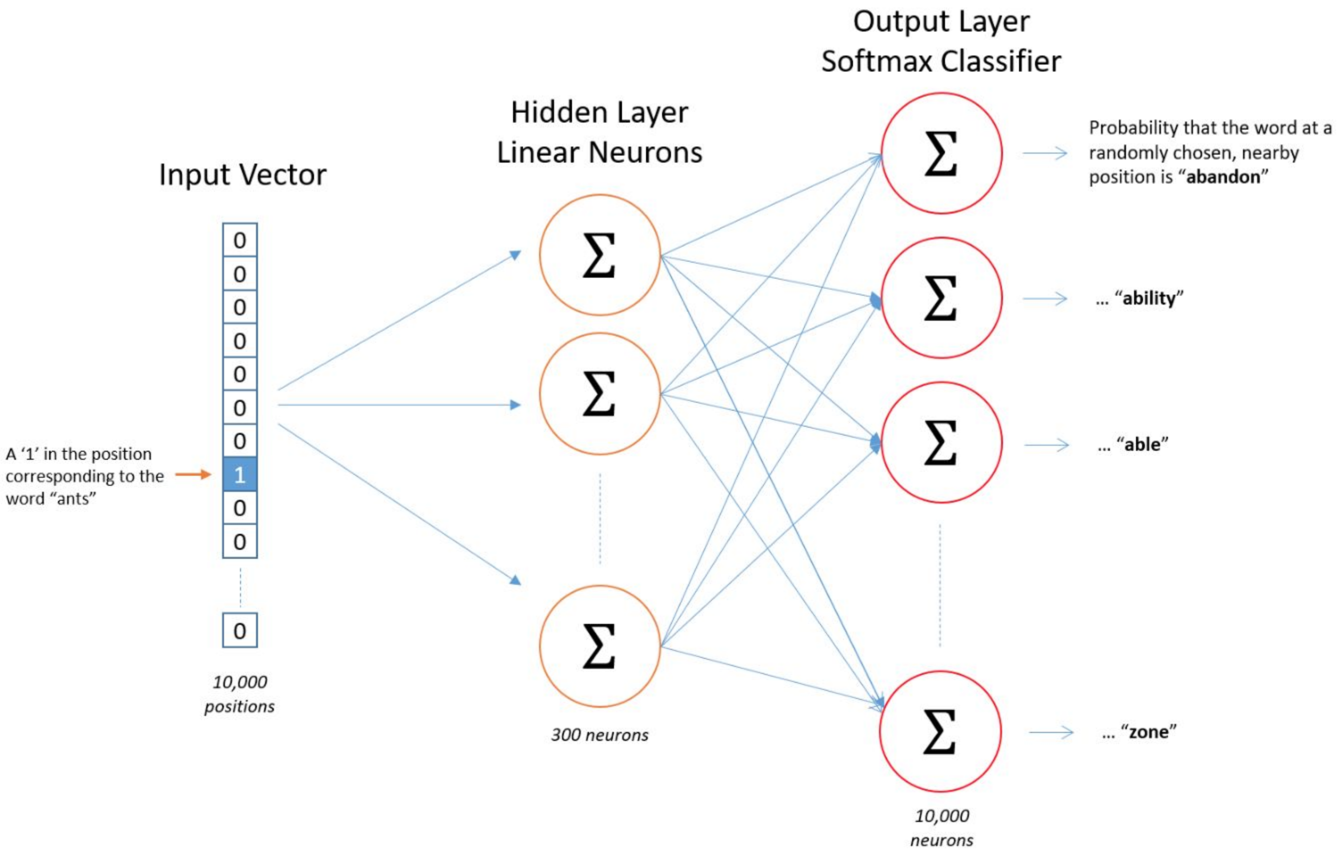
$$P(o|c) = \frac{\exp(\boxed{u_o^T v_c})}{\boxed{\sum_{w \in V} \exp(u_w^T v_c)}}$$

Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

Normalize over entire vocabulary
to give probability distribution

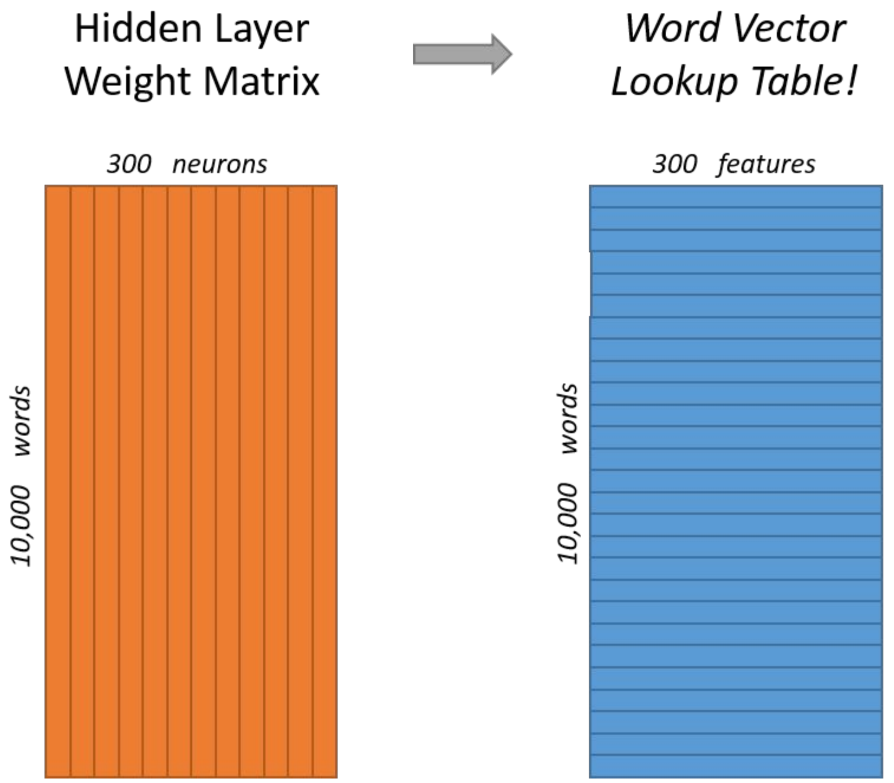
CS224n: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf

Word2vec: architecture



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

Word2vec



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

Gradient descent

- θ represents all the parameters of the model

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

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

Word2vec: two models

Continuous BOW (CBOW)

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

Predict center word from (bag of) context words

Skip-gram

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

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

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

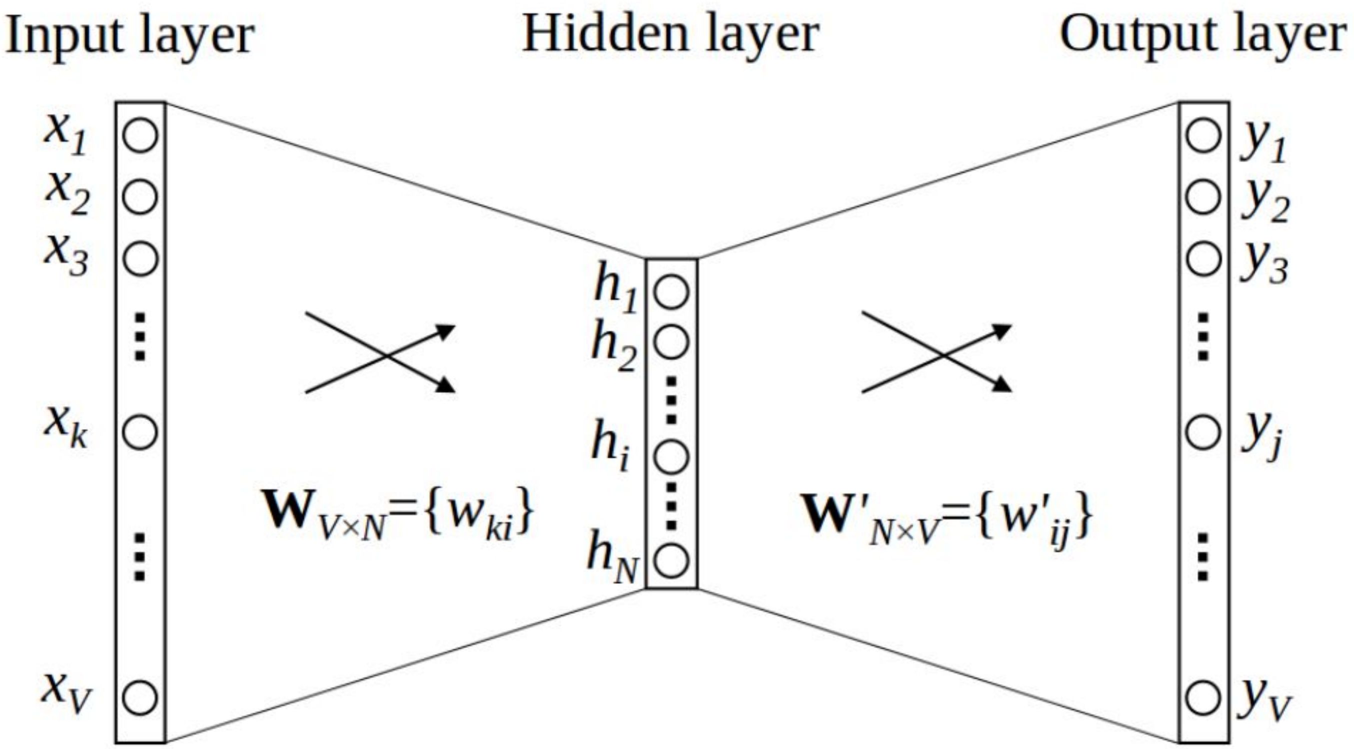
Skip-gram

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

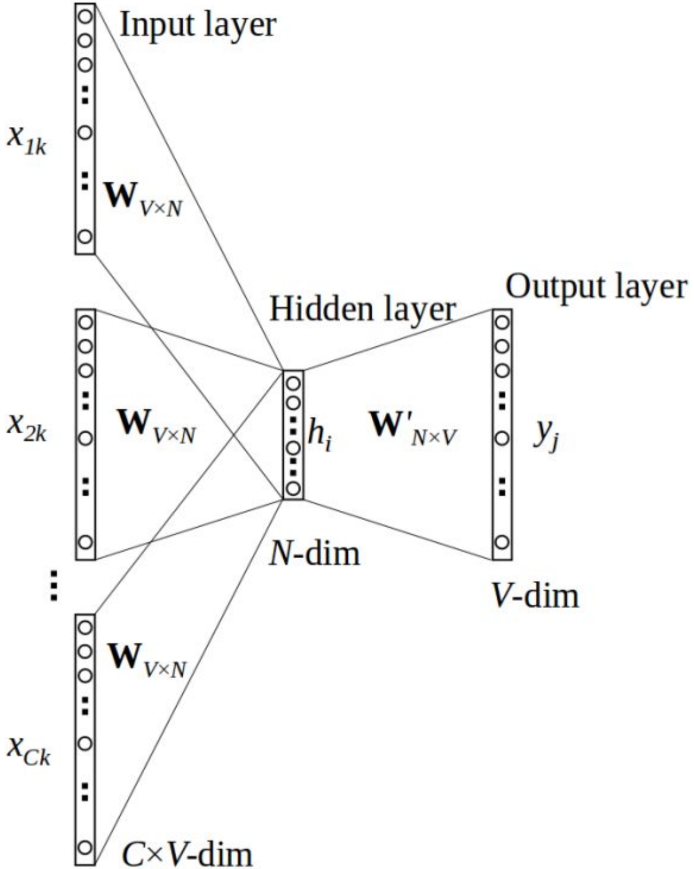
Predict context (”outside”) words (position independent) given center word

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

Word2vec improvements

- Subsampling frequent words
- Negative sampling

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

Subsampling frequent words

What’s the problem with words like “the”?

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

Subsampling frequent words

What’s the problem with words like “the”?

- “the” appears in the context of pretty much every word.
- We will have many more samples of (“the”, …) than we need to learn a good vector for “the”.

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

Subsampling frequent words

What’s the problem with words like “the”?

- “the” appears in the context of pretty much every word.
- We will have many more samples of (“the”, ...) than we need to learn a good vector for “the”.

Let’s remove frequent words with some probability!

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

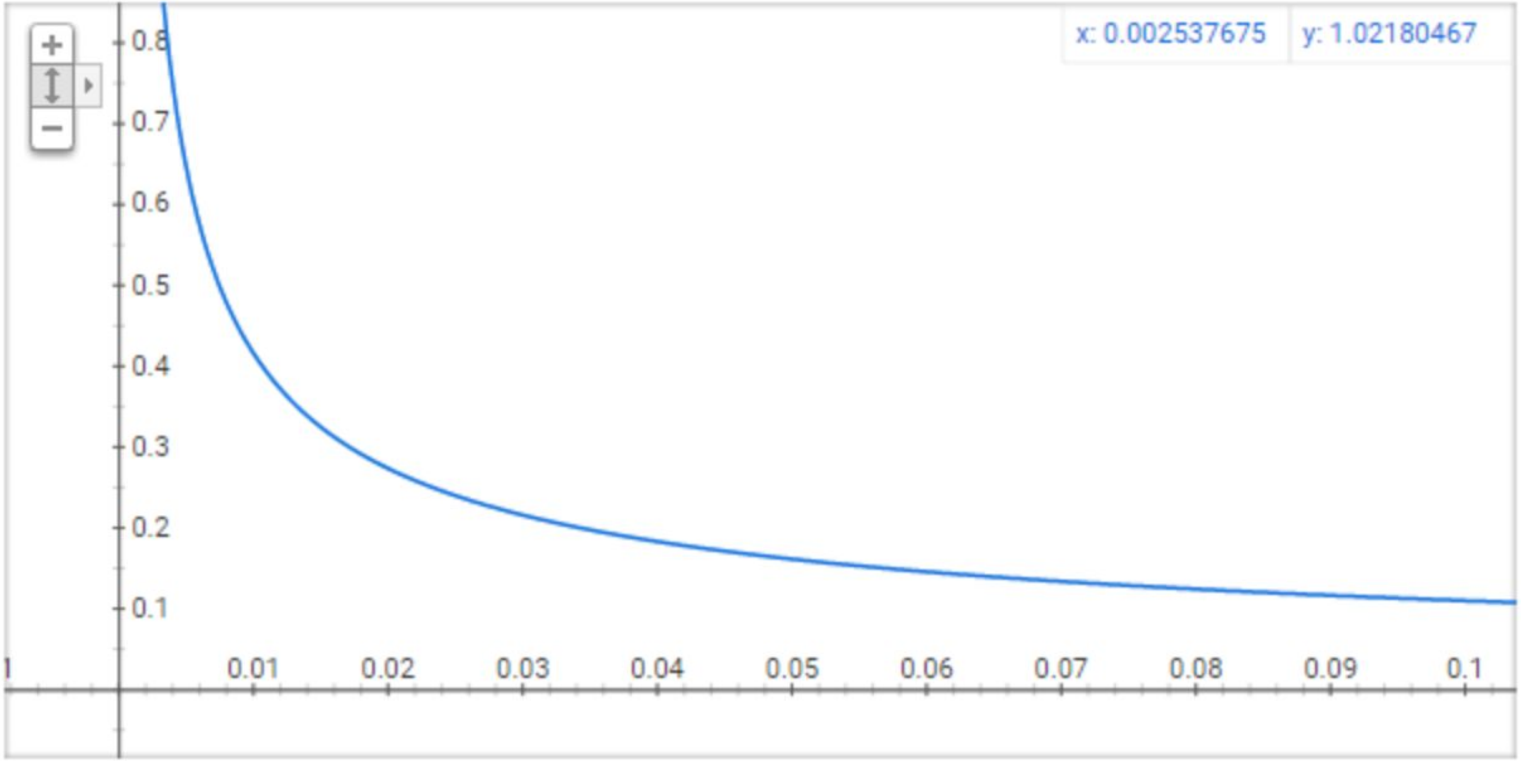
Subsampling frequent words

The probability of keeping the word:

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

$z(w_i)$ is the fraction of the total words in the corpus that are that word

Graph for (sqrt(x/0.001)+1)*0.001/x



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

Subsampling frequent words

$P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.

- This means that only words which represent more than 0.26% of the total words will be subsampled.

$P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.

$P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.

- That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

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

Negative sampling

- each training sample will tweak all of the weights in the neural network
- negative sampling addresses this by having each training sample only modify a small percentage of the weights

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

Negative sampling

- randomly select just a small number of “negative” words (“negative” word is one for which we want the network to output 0)
- update the weights for all our “positive” words

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

Negative sampling

- more frequent words are more likely to be selected as negative samples

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

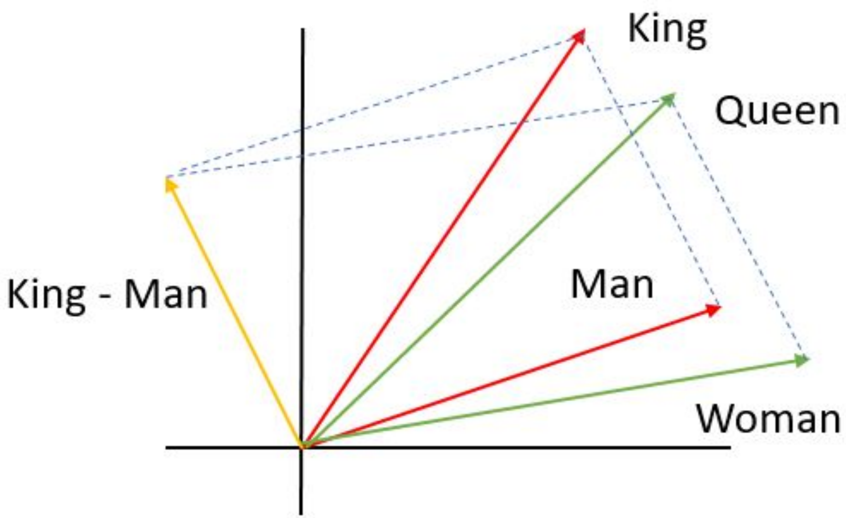
http://mccormickml.com/2017/01/11/word2vec-tutorial-part-2-negative-sampling/

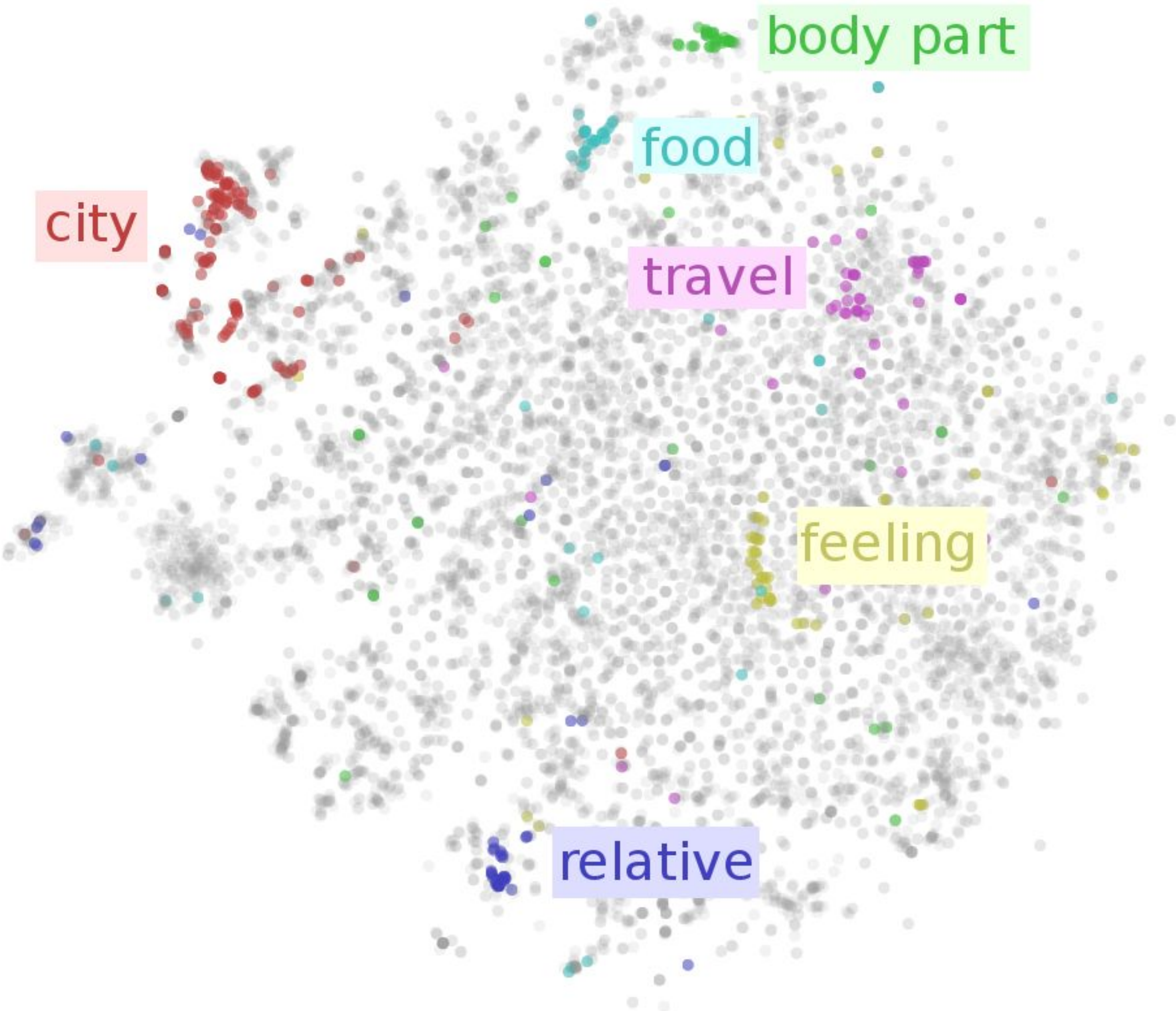
Word2vec: word analogies

King - man + woman = queen

$$x \qquad y \qquad y' \qquad target$$

$$\cos\left(x-y+y', target\right) \to \max_{target}$$




body part
food
city
travel
feeling
relative

Backup