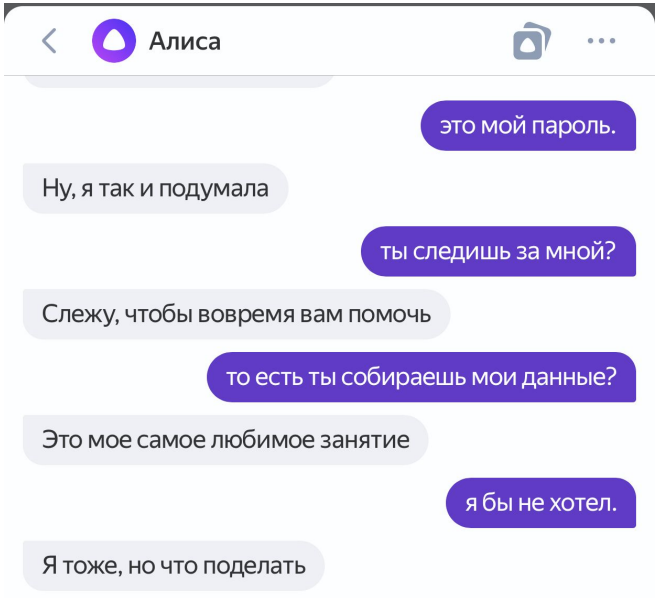
Natural Language Processing and Word Embeddings

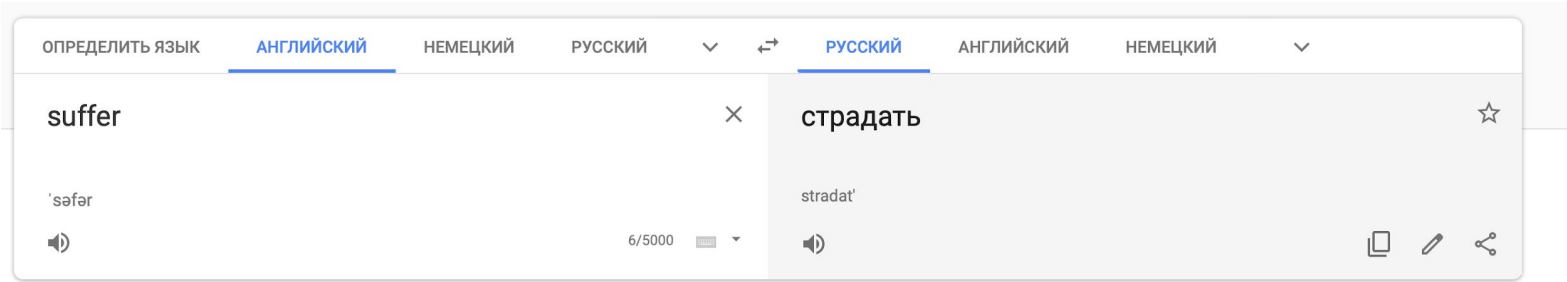
План

- NLP tasks
- Statistical word representations
- Word embeddings:
  - Word2Vec
  - GloVe
  - FastText
- Sentence embeddings
- Languages similarities

Natural Language Processing

- texts classification (author, genre, etc)
- sentiment analysis
- machine translation
- texts generation
- question answering
- text summarization
- dialogue systems





Natural Language Processing

CV+NLP

- image captioning
- video captioning (subtitles)
- video description (movement/profession/disaster (?), etc.)
- reverse: generate image by text

A group of young people playing a game of frisbee.

Two hockey players are fighting over the puck.

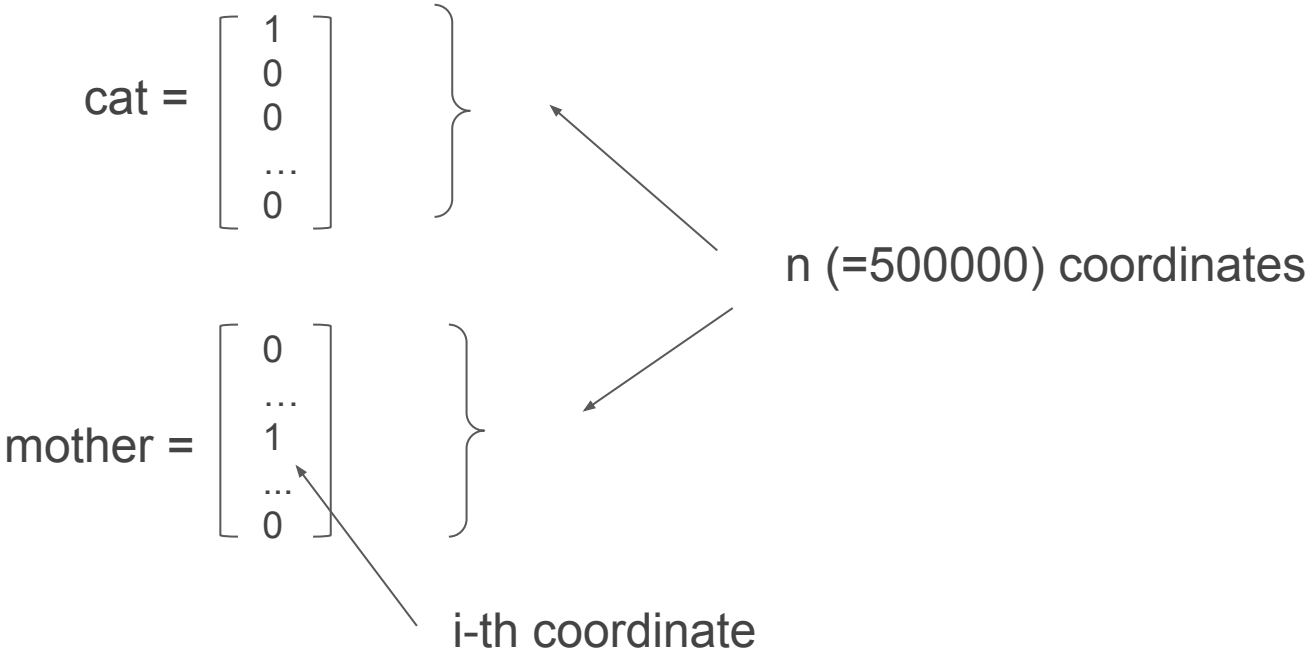
A little girl in a pink hat is blowing bubbles.

Natural Language Processing

How do we represent words in computer?

Easy way: one-hot encoding

One-hot vectors of dictionary size (ex. 500,000)



One-hot encoding

One-hot vectors of dictionary size (ex. 500,000)

Problems:

- vectors do not contain meaning
- no similarity measure between vectors

More clever way: context embeddings

Let’s consider words that can fit in the gaps:

1. Marie rode a _____
2. _____ wheel was punctured
3. The _____ has a beautiful white frame

	1	2	3
Bicycle	+	+	+
Bike	+	+	+
Car	+	+	-
Horse	+	-	-

Context embeddings

Let’s take into account words meanings in some way:

$v(word_i)[\,j\,]$ = count(co-occurrences $word_i$ with $word_j$ in dataset)

$v(word_1)$ = [12, 1, 0, 10, 5,...]

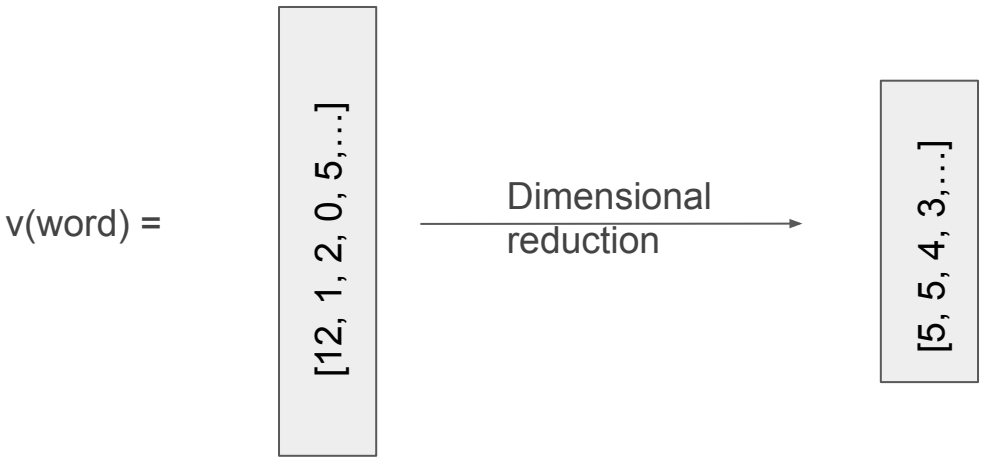
horse ride wheel roof hair breed

$v(word_2)$ = [20, 10, 0, 0, 1,...]

car ride wheel roof hair breed

Context embeddings

v(word$_i$)[j] = count(co-occurrences word$_i$ with word$_j$ in dataset)

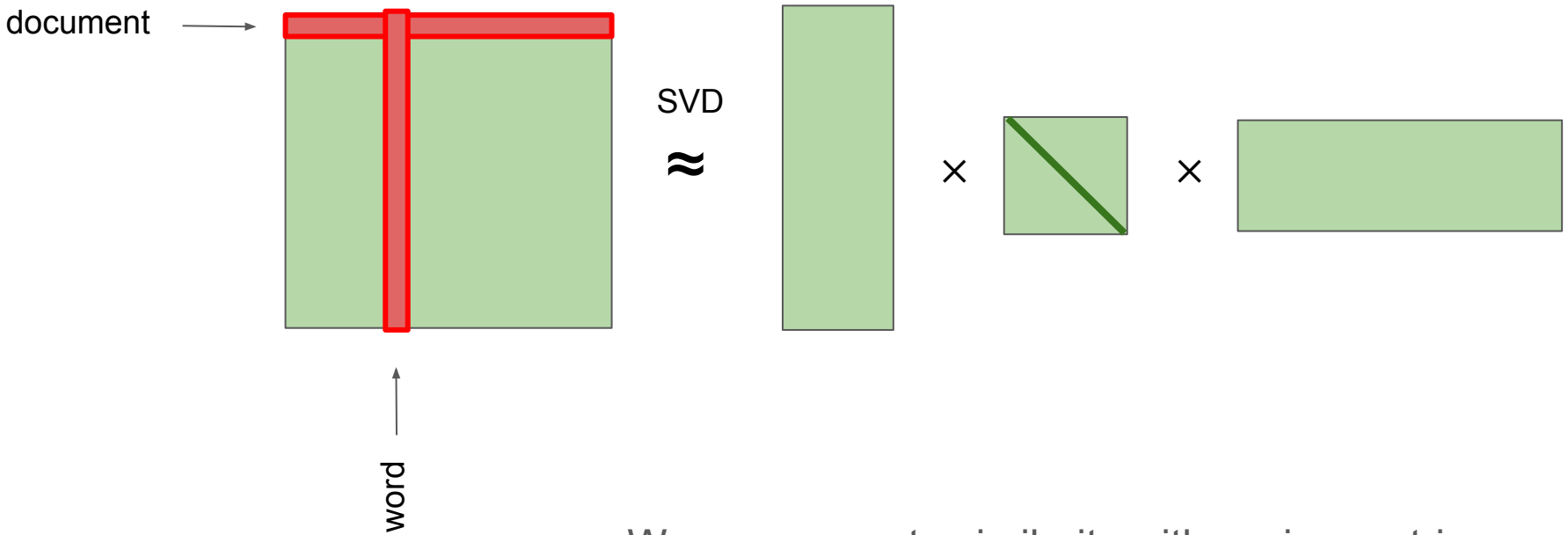


Context embeddings

Problems:

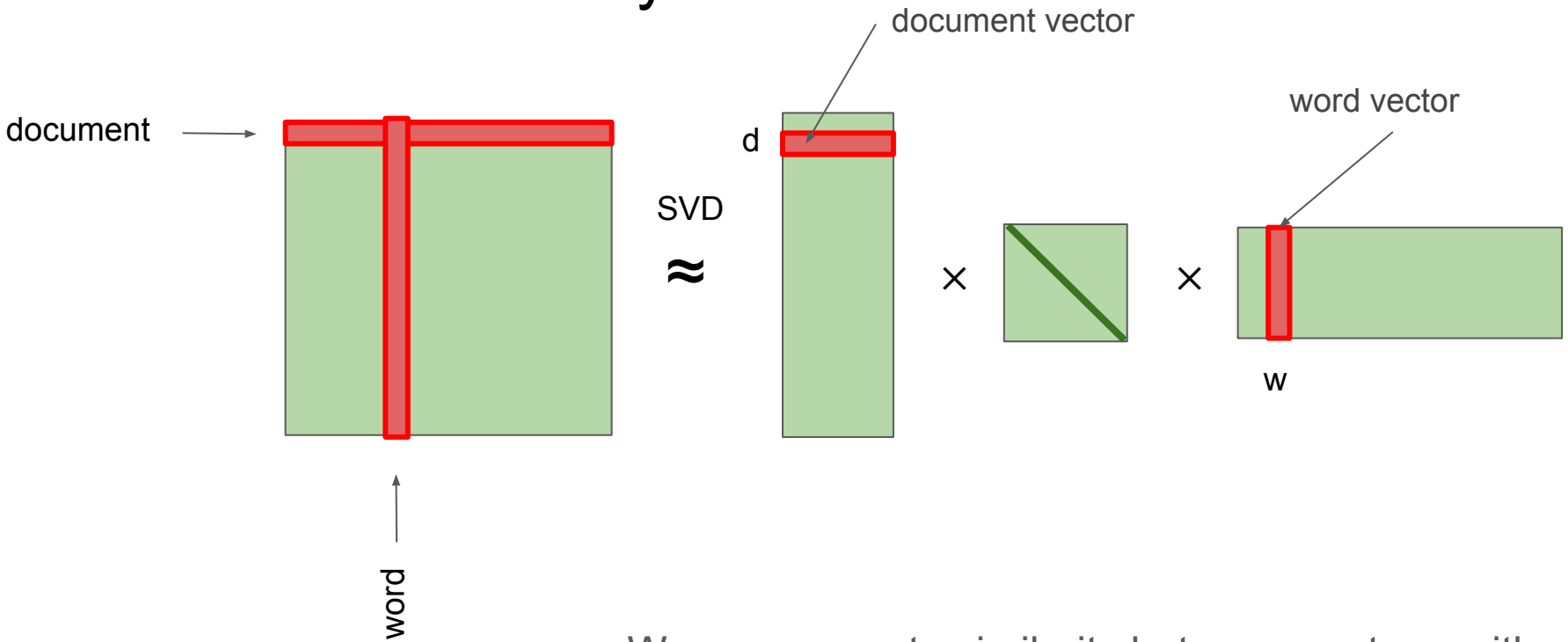
- rare words
- huge computational time
- when you change your dataset you need to recompute everything

Latent semantic analysis



We can compute similarity with cosine metric.

Latent semantic analysis



We can compute similarity between vectors with cosine metric.

Latent semantic analysis

- We get small-dimensional vectors
- Vectors contain meanings of words
- We can compute distance between words (cosine distance)

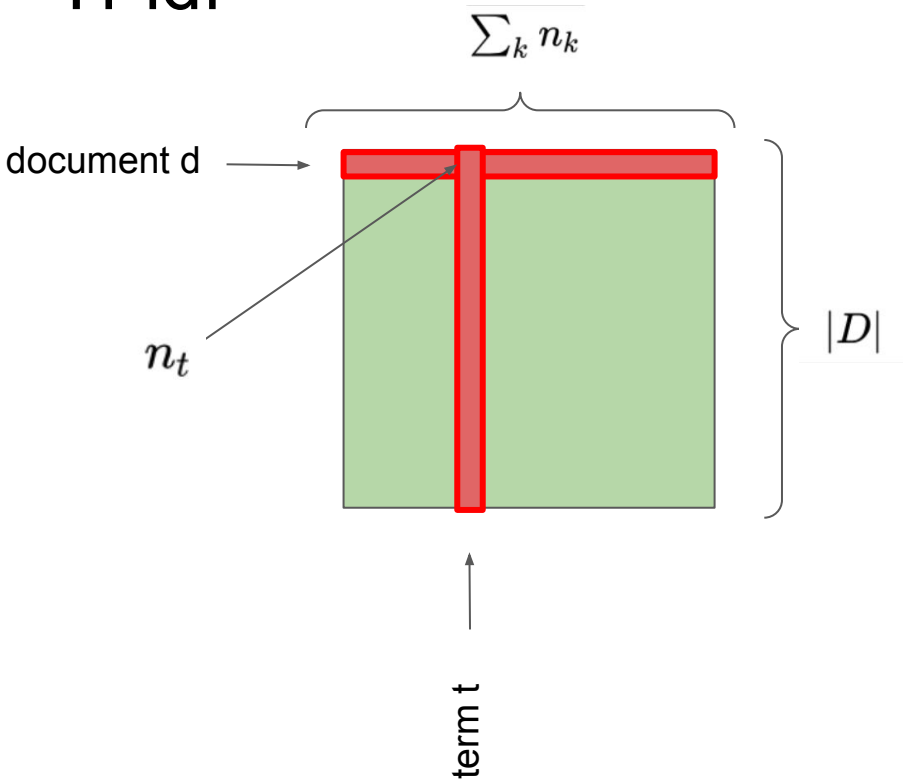
Latent semantic analysis

- We get small-dimensional vectors
- Vectors contain meanings of words
- We can compute distance between words (cosine distance)

Problems:

- huge computational time
- when you change your dataset you need to recompute everything
- poor quality (still)

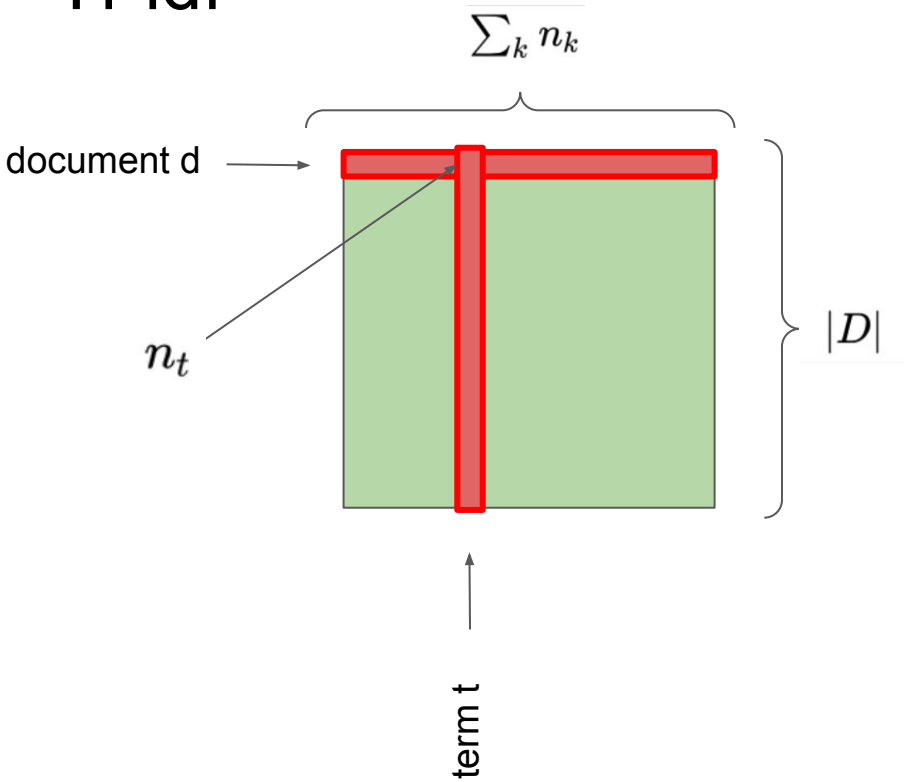
Tf-idf



$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

count term frequency in the document

Tf-idf



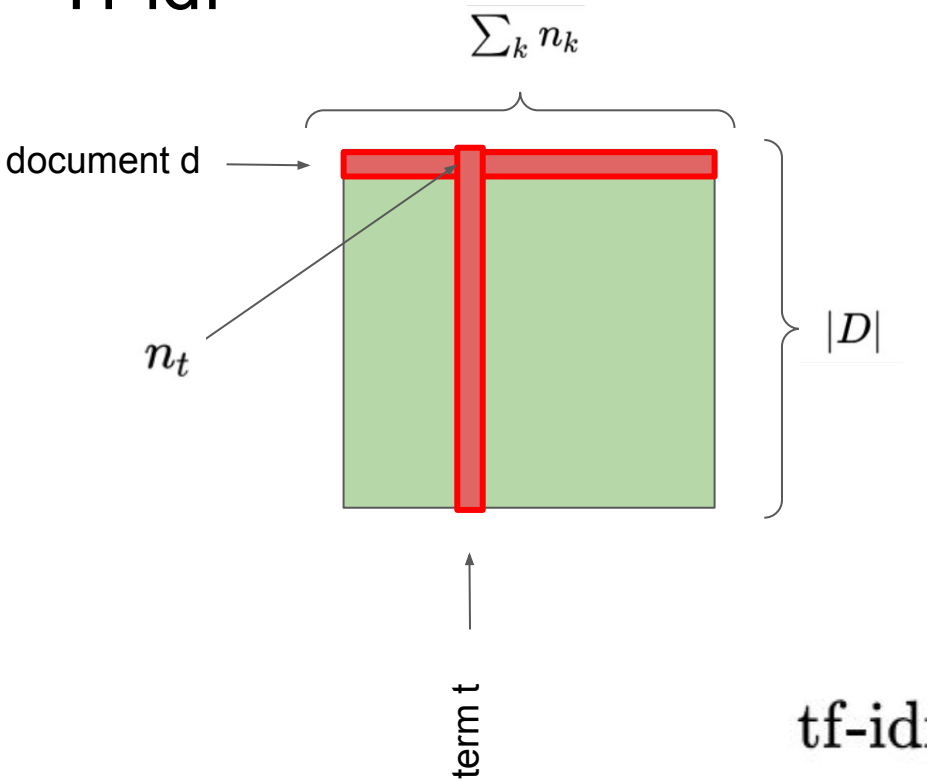
$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

count term frequency in the document

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{\, d_i \in D \mid t \in d_i \,\}|}$$

take into account very usual words:
inverse document frequency

Tf-idf



$$\mathrm{tf}(t,d) = \frac{n_t}{\sum_k n_k}$$

count term frequency in the document

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|\{\, d_i \in D \mid t \in d_i \,\}|}$$

take into account very usual words:
inverse document frequency

$$\text{tf-idf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D)$$

Still what we did is we created some matrix of word representations and then computed word vectors using dimensionality reduction methods

But how about to **learn** those word representations?

What we want:

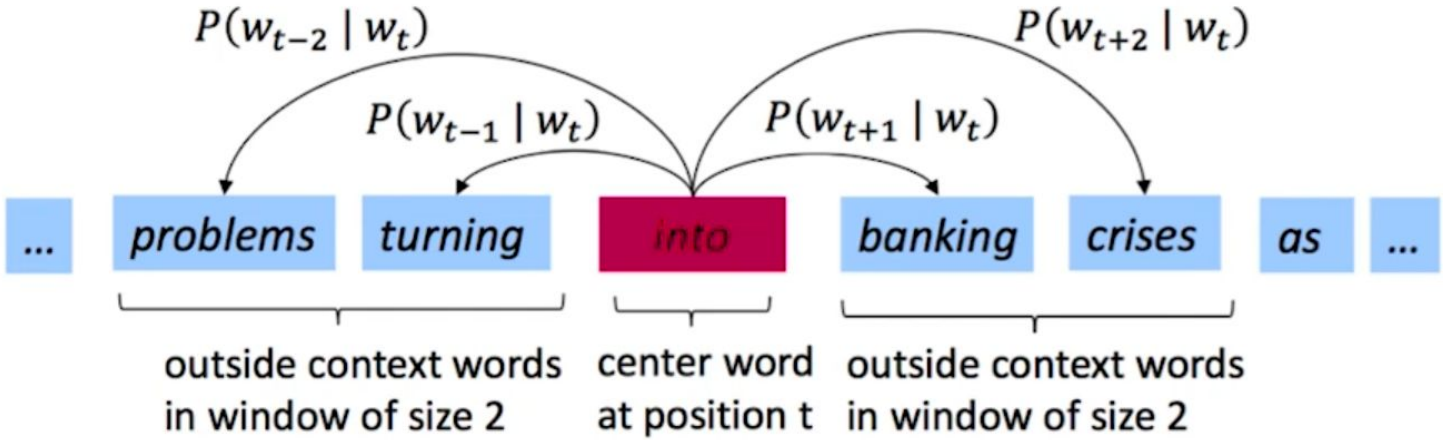
We want to **learn dense** vectors for words and we want this vectors to have **distance** (word vector should be close to vectors of other words that often appear in same context)

Those learned word vectors are called **word embeddings**

Word2Vec

We want to maximize probabilities of seeing a surrounding word based on center words.

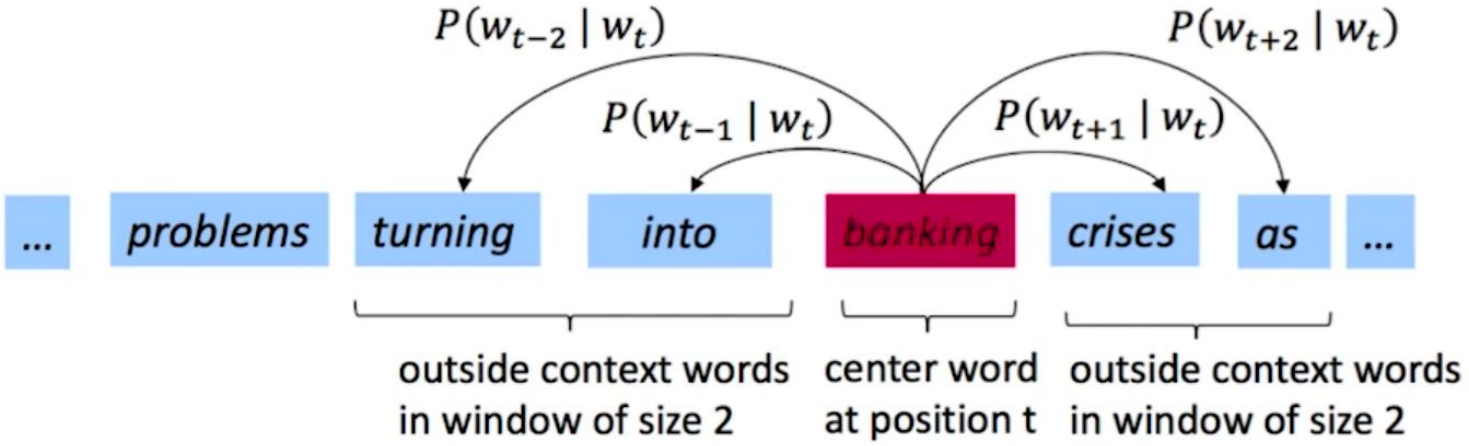
Going through text corpus by sliding windows.



Word2Vec

We want to maximize probabilities of seeing a surrounding word based on center words.

Going through text corpus by sliding windows.



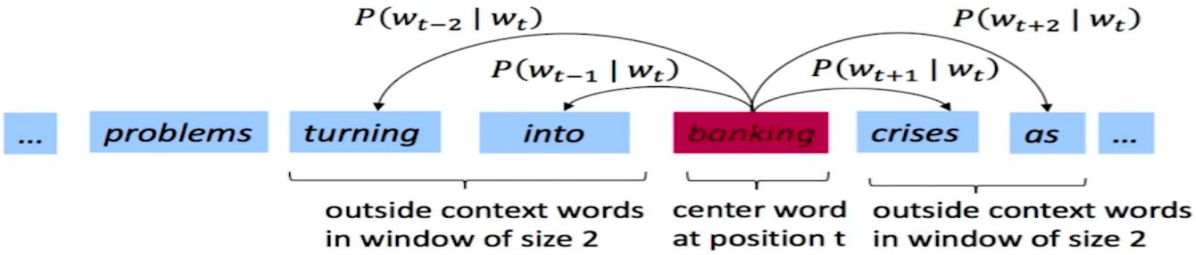
Word2Vec

Target function:

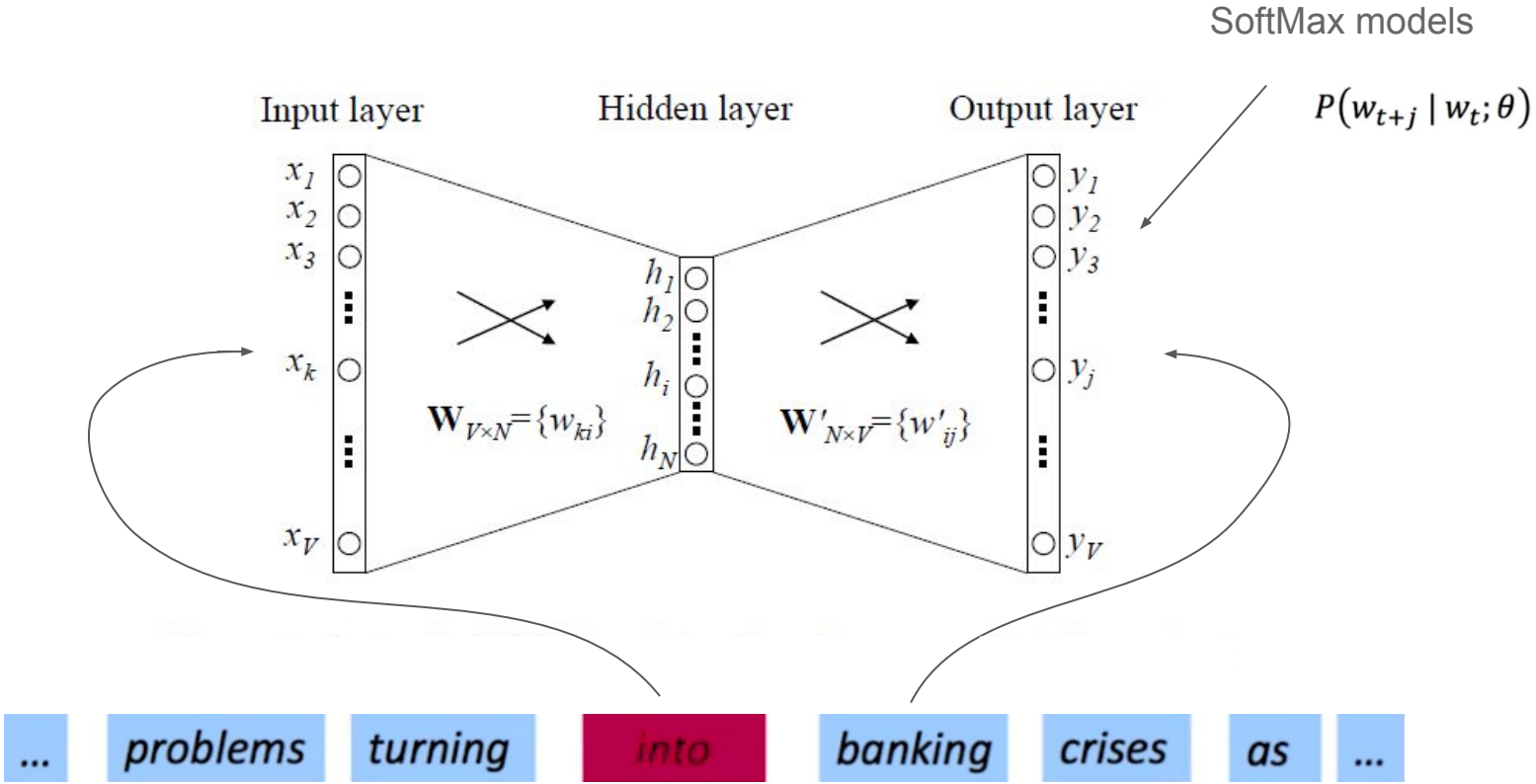
$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$

Or in log-likelihood point of view::

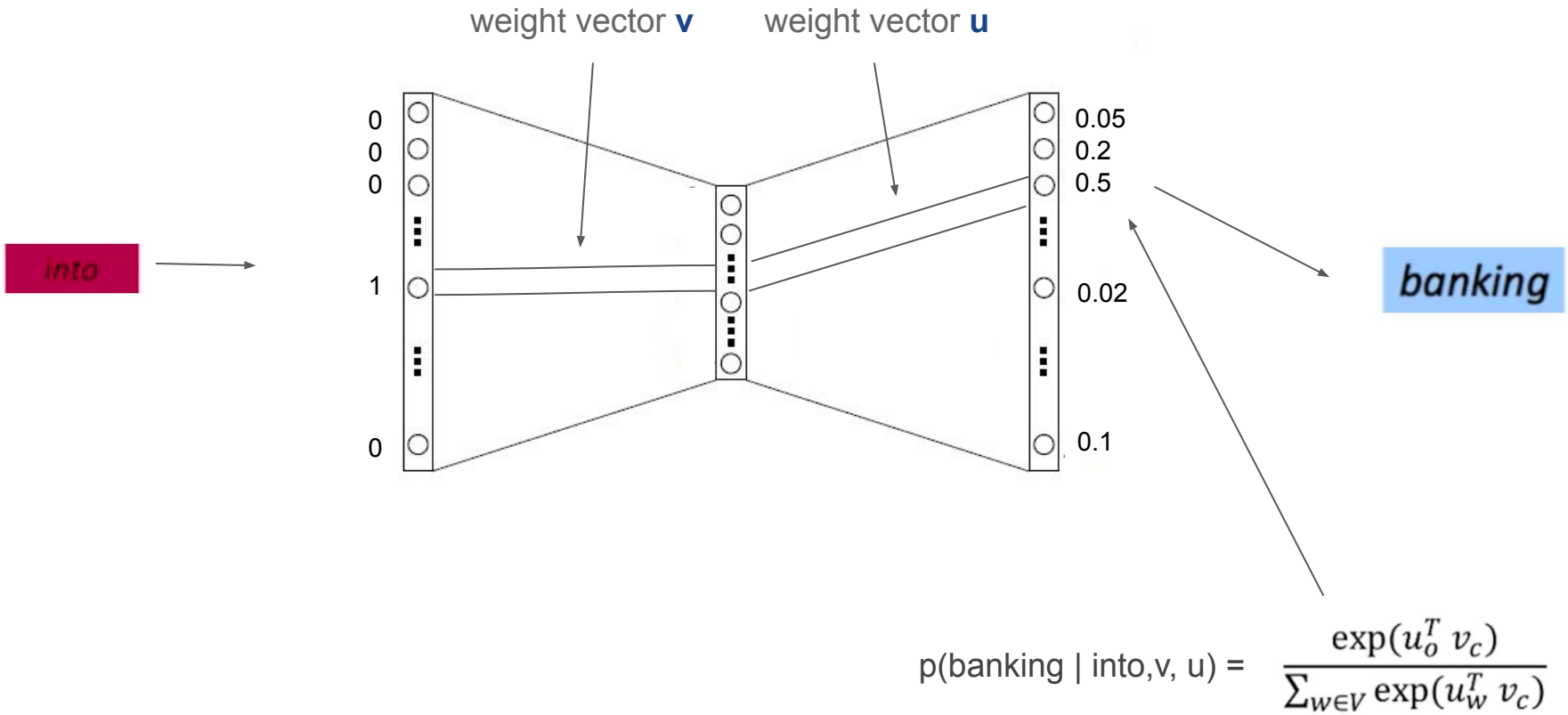
$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P(w_{t+j} \mid w_t; \theta)$$



Word2Vec

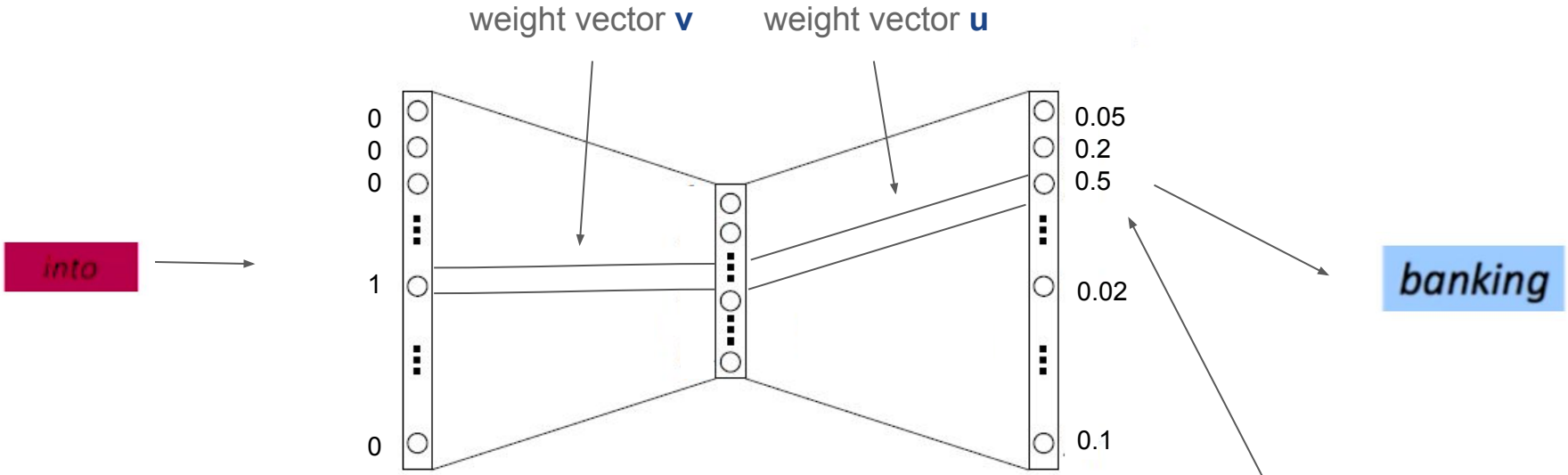

SoftMax models
$P(w_{t+j} \mid w_t; \theta)$
Input layer
Hidden layer
Output layer
x_1
x_2
x_3
x_k
x_V
h_1
h_2
h_i
h_N
y_1
y_2
y_3
y_j
y_V
$\mathbf{W}_{V\times N}=\{w_{ki}\}$
$\mathbf{W}'_{N\times V}=\{w'_{ij}\}$
...
problems
turning
into
banking
crises
as
...

Word2Vec



$$p(\text{banking} \mid \text{into}, v, u) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2Vec



- u is a context word vector
- v is a center word vector

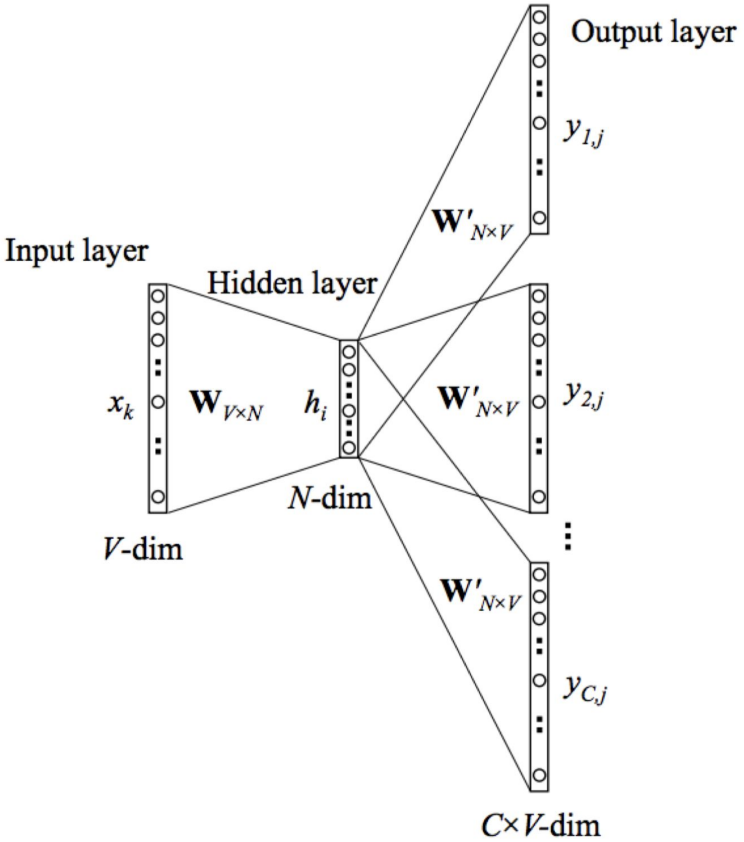
“Scalar product between me and my neighbour must be as big as possible”

$$p(\text{banking} \mid \text{into}, v, u) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Word2Vec: Skip-Gram

Maximize probabilities of seeing a surrounding word based on center words.

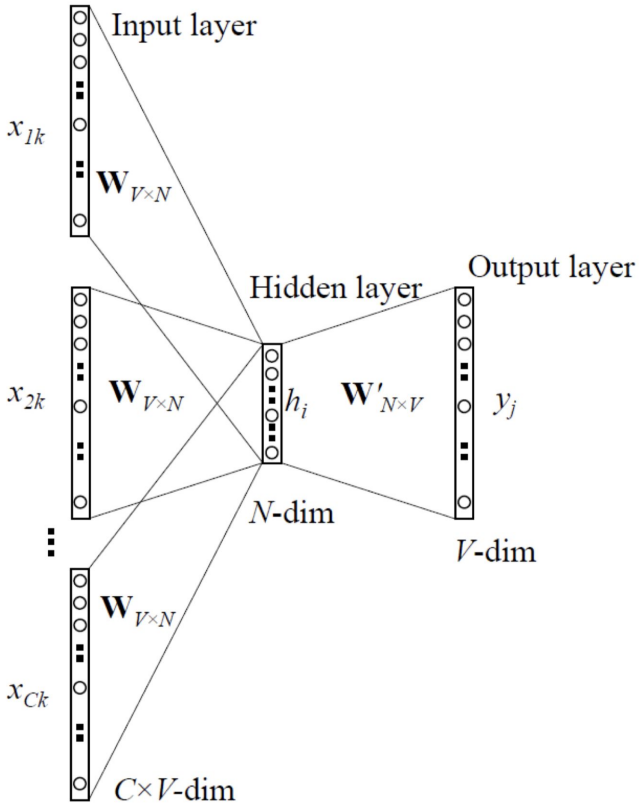
$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} P(w_{t+j} \mid w_t; \theta)$$



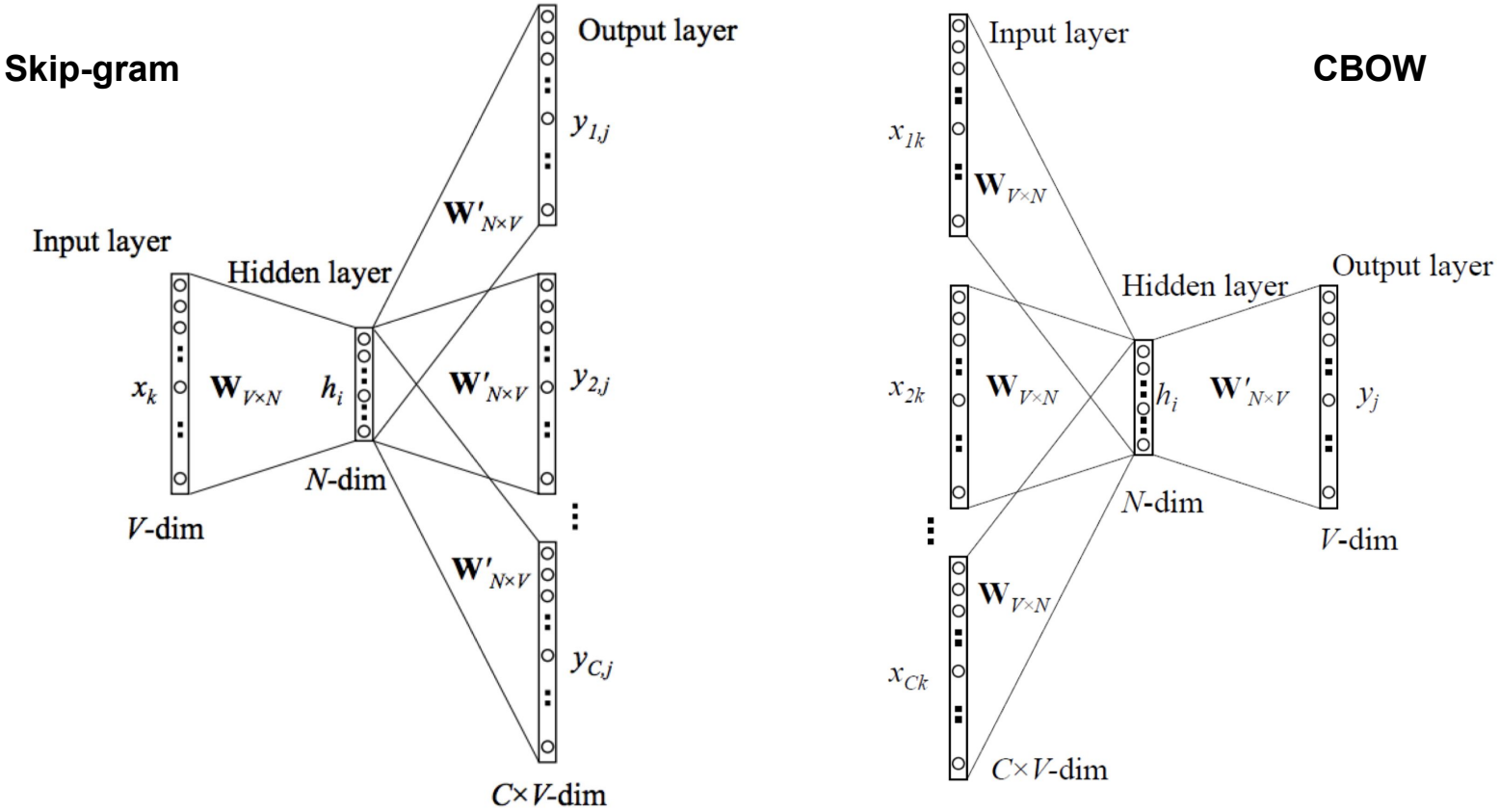
Word2Vec: CBOW

Maximize probabilities of seeing a center word based on surrounding words.

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



Word2Vec


Skip-gram
Input layer
Hidden layer
Output layer
x_k
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
$y_{1,j}$
$y_{2,j}$
$y_{C,j}$
V-dim
N-dim
$C\times V$-dim
CBOW
Input layer
Hidden layer
Output layer
x_{1k}
x_{2k}
x_{Ck}
$\mathbf{W}_{V\times N}$
h_i
$\mathbf{W}'_{N\times V}$
y_j
N-dim
V-dim
$C\times V$-dim

Word2Vec

Problem:

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

Still big sum to compute

Word2Vec

Problem:

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

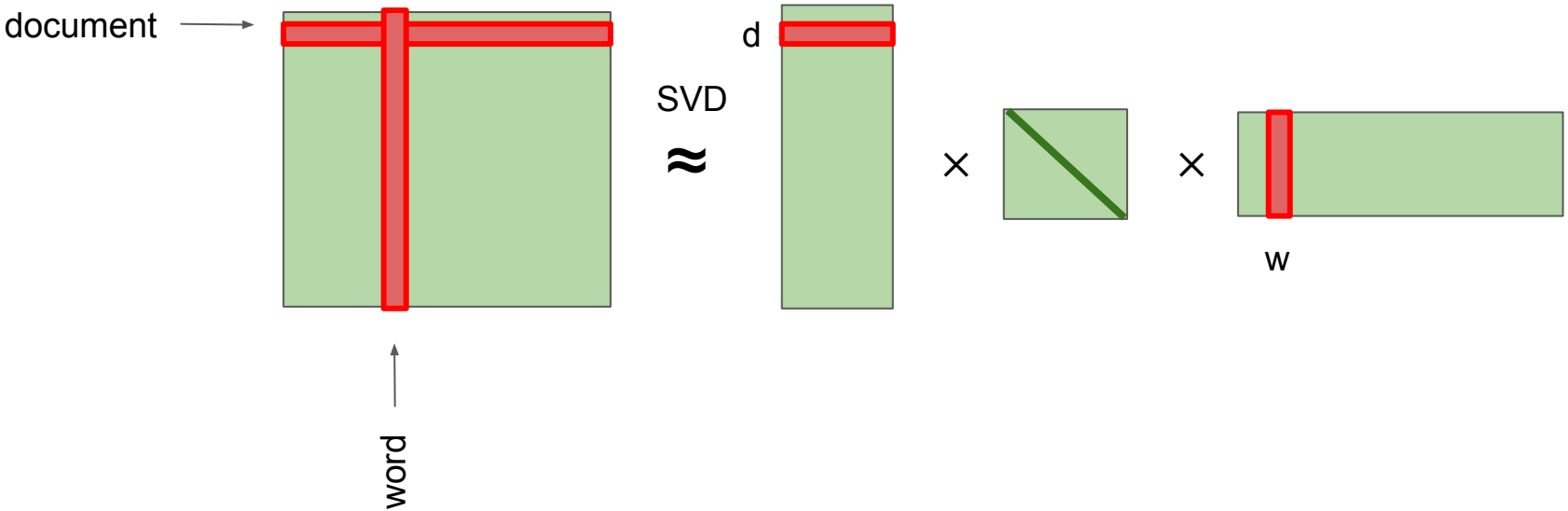
Still big sum to compute

Possible solutions:

- Hierarchical softmax
- Negative sampling

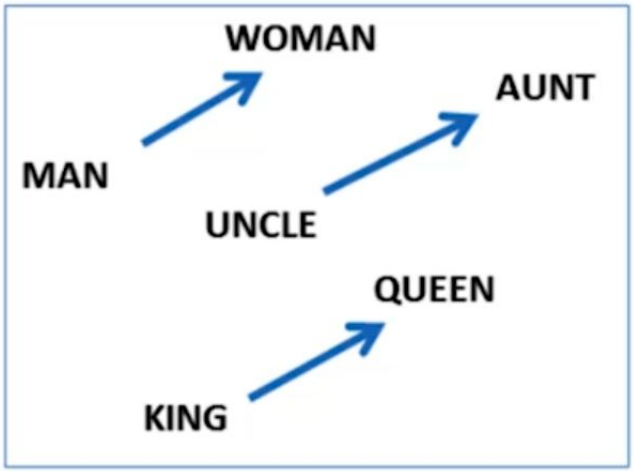
Word2Vec vs SVD

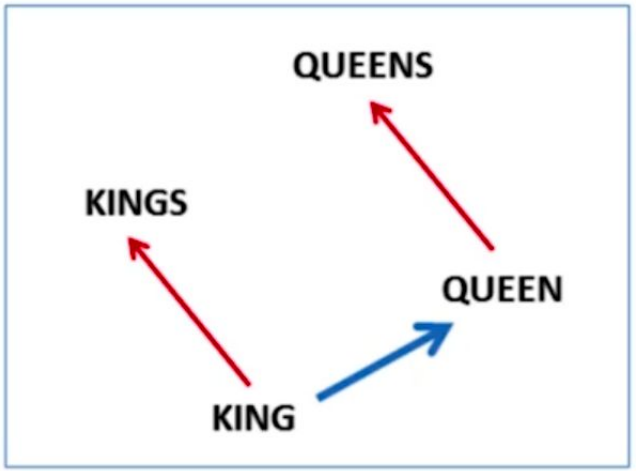
Word2Vec with negative sampling ≈ matrix factorization

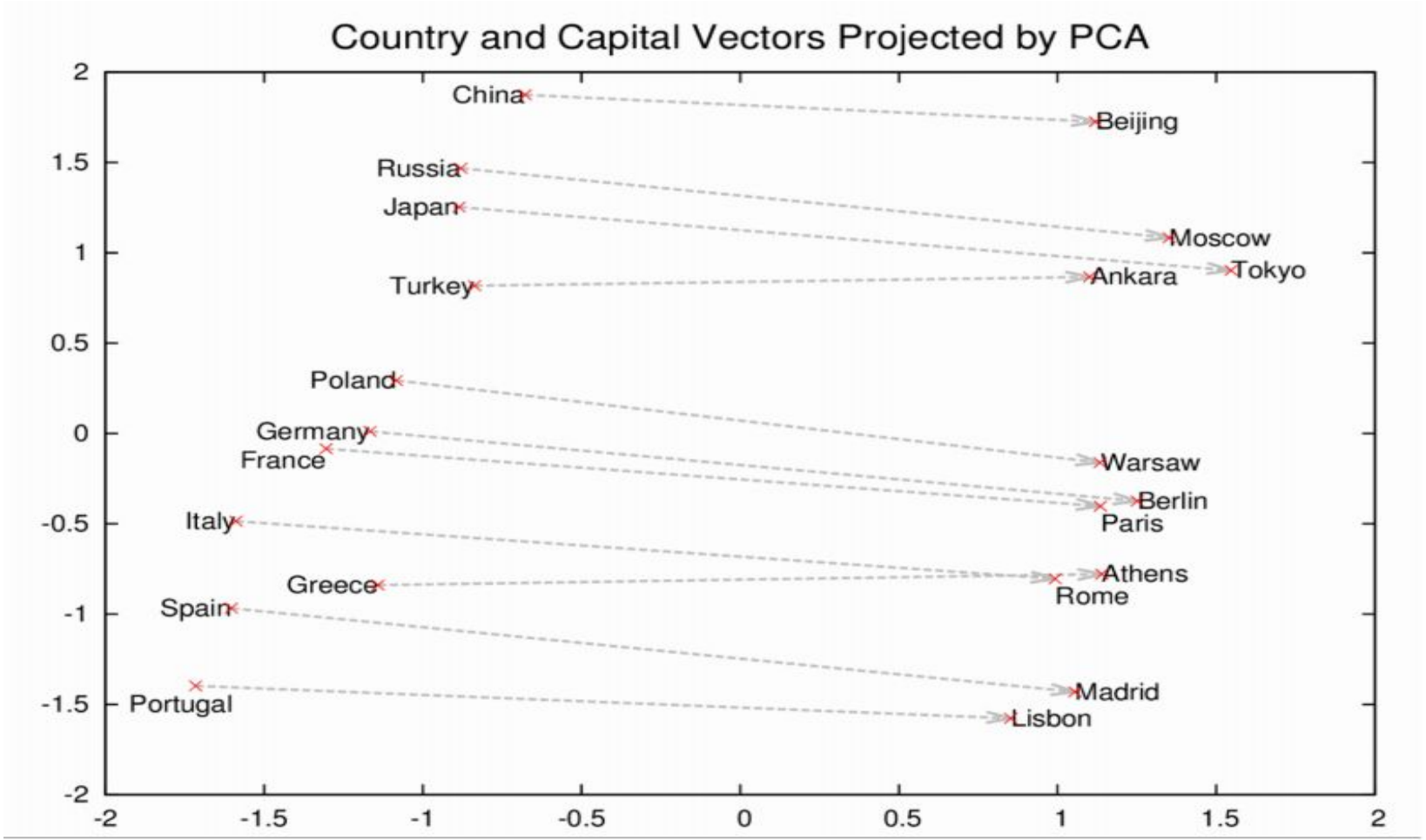


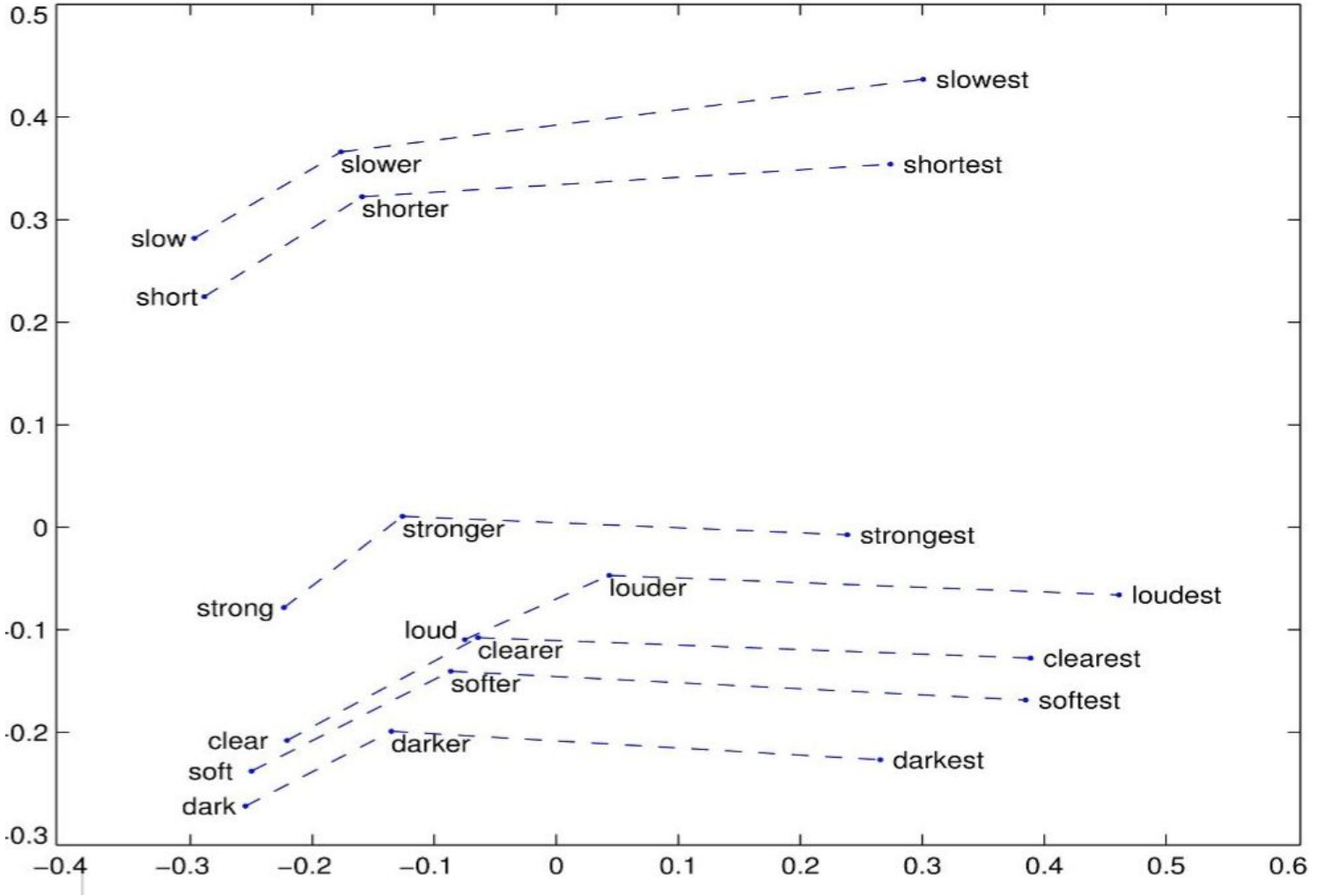
Word2Vec

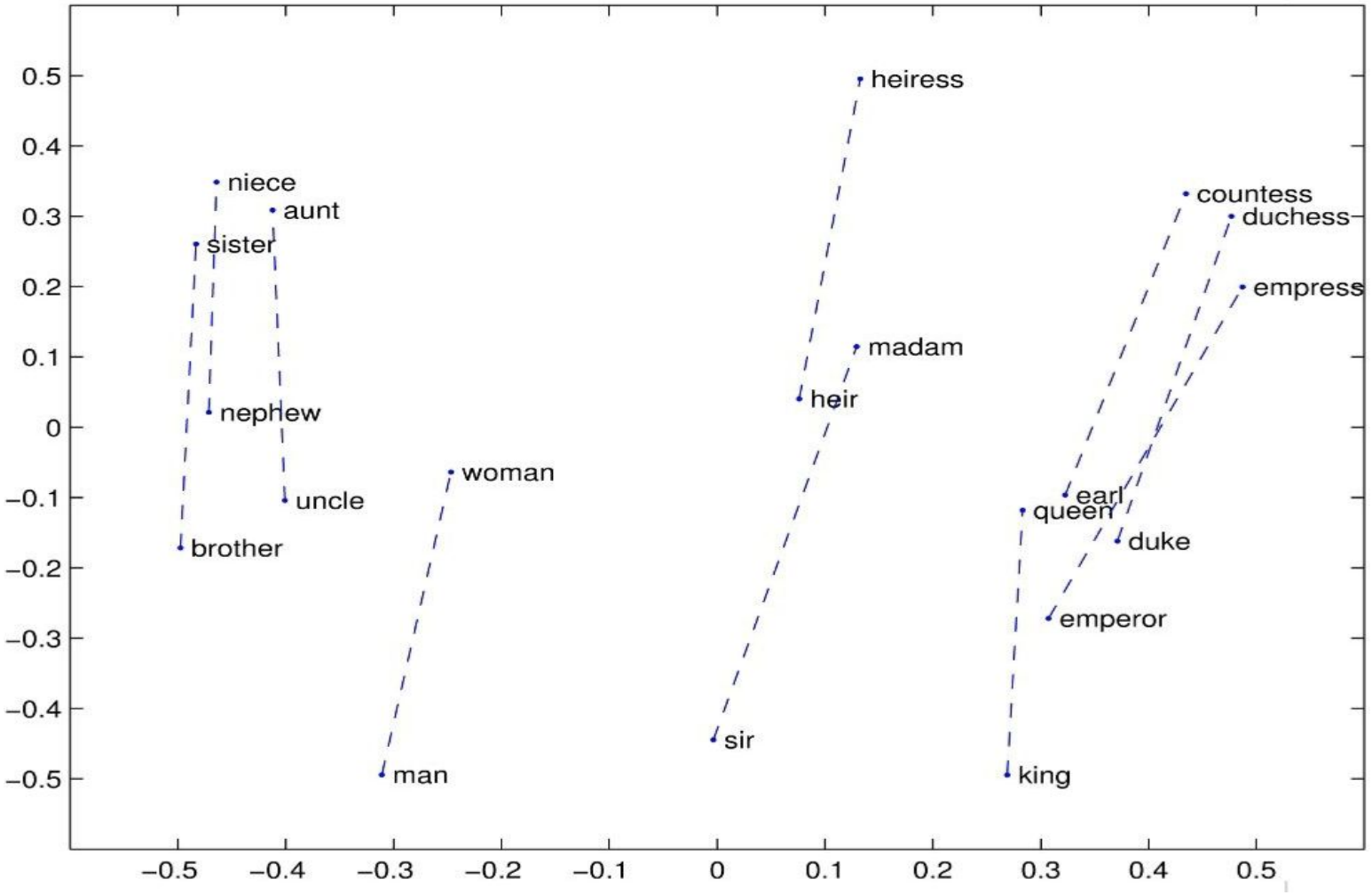
v(king) - v(man) + v(woman) ≈ v(queen)






Country and Capital Vectors Projected by PCA
China
Beijing
Russia
Moscow
Japan
Tokyo
Turkey
Ankara
Poland
Warsaw
Germany
Berlin
France
Paris
Italy
Athens
Greece
Rome
Spain
Madrid
Portugal
Lisbon
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2


0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6
slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark


0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
niece
aunt
sister
nephew
uncle
brother
woman
man
heiress
madam
heir
sir
countess
duchess
empress
earl
queen
duke
emperor
king

GloVe

Before training count occurrences of pairs [$word_i$, $word_j$] in corpus

Compute probabilities P_{ij} = P ([$word_i$, $word_j$])

Objective function:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij}) (u_i^T v_j - \log P_{ij})^2$$

Discount factor for rare words

GloVe

Before training count occurrences of pairs [$word_i$, $word_j$] in corpus

Compute probabilities P_{ij} = P ([$word_i$, $word_j$])

Closely related to co-occurrence matrix

Objective function:

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij}) (u_i^T v_j - \log P_{ij})^2$$

Close to the idea of factorizing co-occurrence matrix (LSA)

Discount factor for rare words

FastText

- Divide word into bag of n-grams: apple = <ap, ppl, ple, le>
- Compute vector for each n-gram
- Vector for a word = sum of vectors for word n-grams

FastText

- Divide word into bag of n-grams: apple = <ap, ppl, ple, le> (**BPE**)
- Compute vector for each n-gram
- Vector for a word = sum of vectors for word n-grams

Advantages:

- Reasonable embeddings for rare words and words with mistakes
- Model is the same as before, we can even use model trained on words to train it further on n-grams!

Sentence embeddings

Yeah, really, why not?

Sentence embeddings

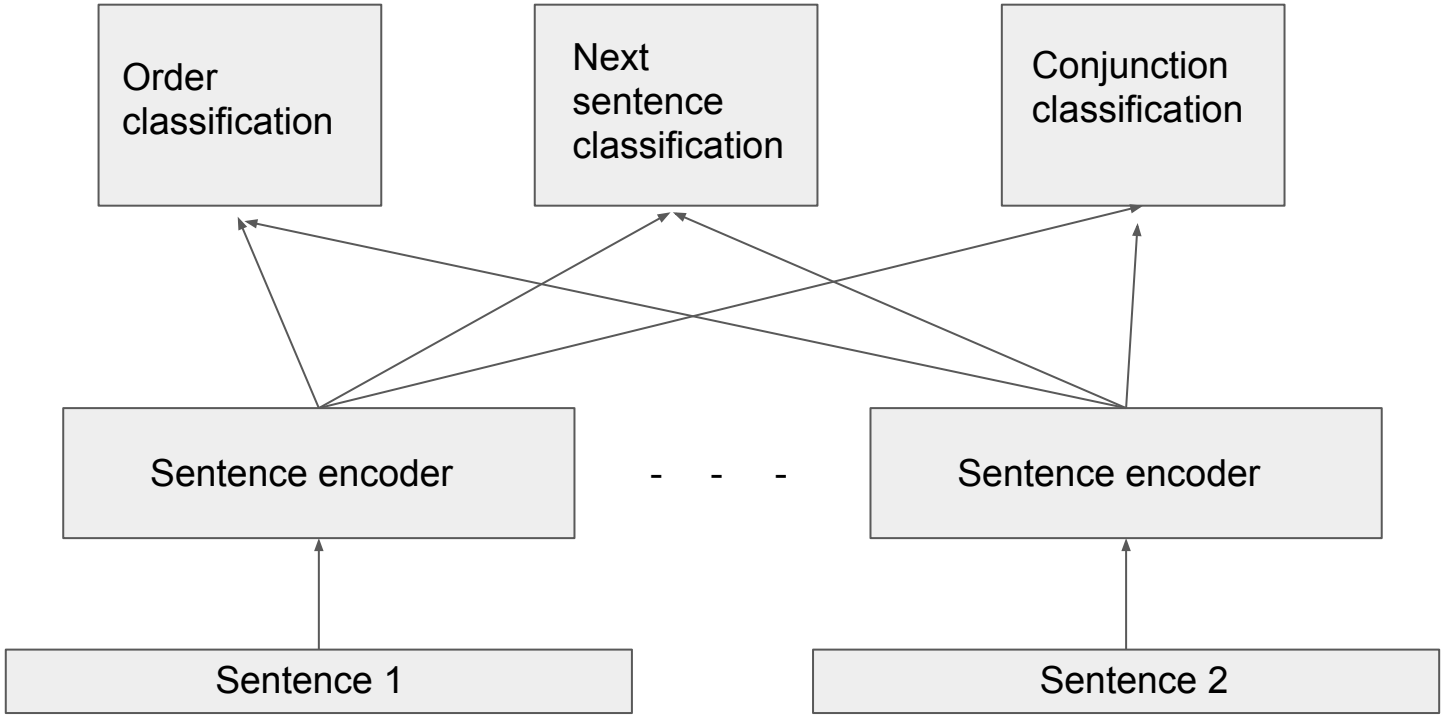
What we did before:

- Predicted context using word

What we'll do now:

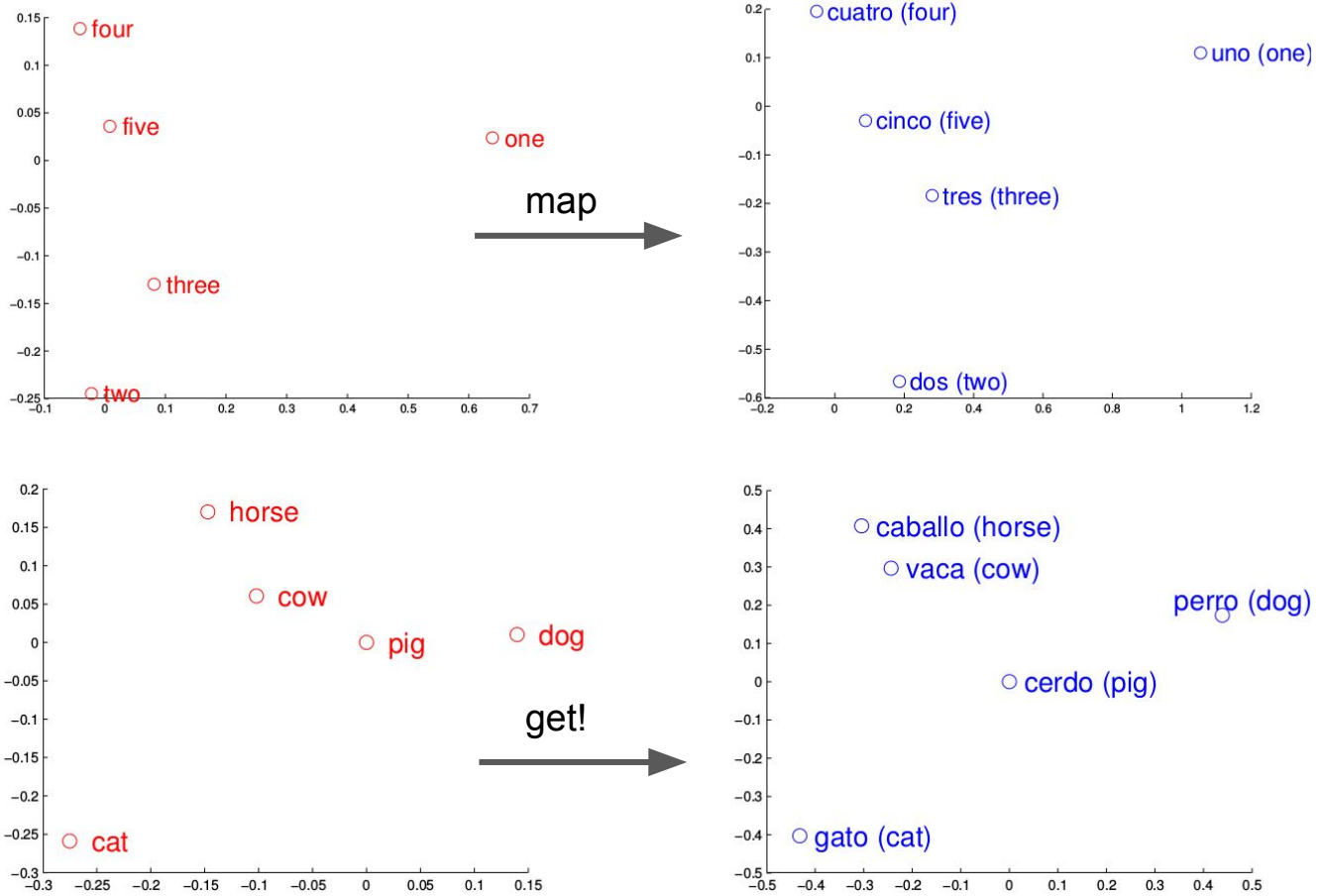
- Predict binary ordering of sentences (does this sentence go next or before?)
- Can this sentence be the next or not? (Next sentence classifier)
- Conjunction prediction

Sentence embeddings



Language similarities

- train embeddings for english
- find mapping f() from english to spanish
- get new english word -- use f() to compute translation!



Finally...

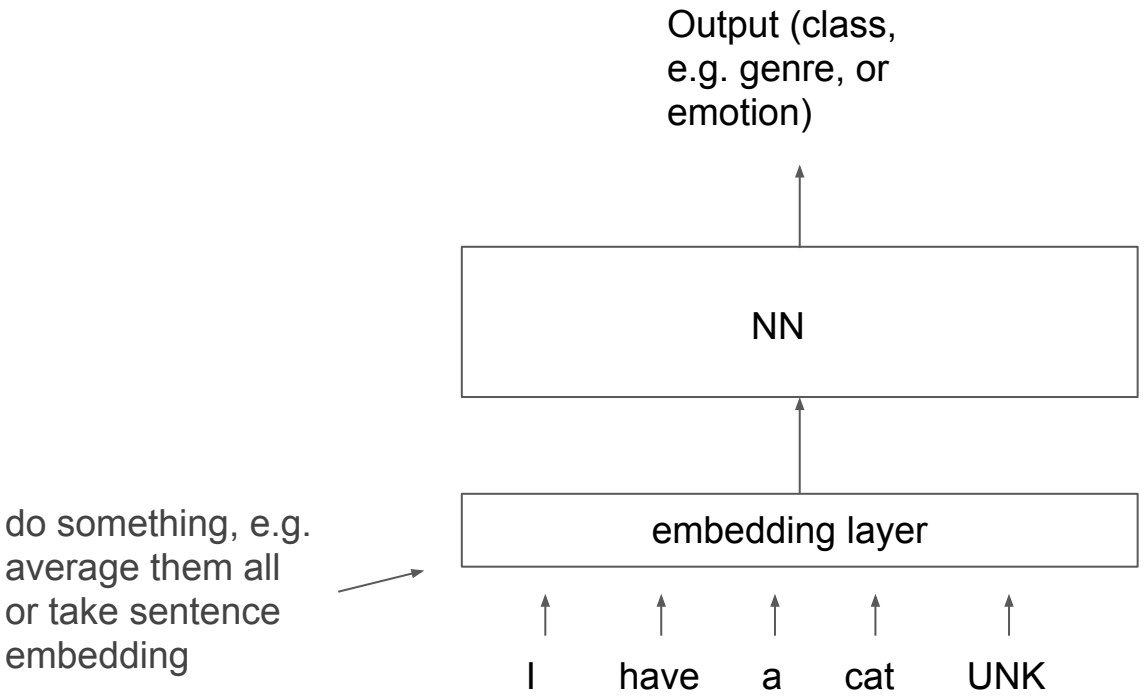
Okay, that’s great, but why do we actually need embeddings?

How to solve...

- sentiment analysis?
- text classification?

… with word embeddings?

Classification task



Classification task

When you have small text data for your task