Lesson 2

Word embeddings

План

- Word representations recall
- Word embeddings:
  - Word2Vec
  - GloVe
  - FastText
- Sentence embeddings
- Languages similarities

Recall: one-hot encoding

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

mother = [0…0, 1, 0...0]

cat = [1, 0...0]

Recall: one-hot encoding

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

mother = [0…0, 1, 0...0]

cat = [1, 0...0]

Problems:

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

Recall: context embeddings

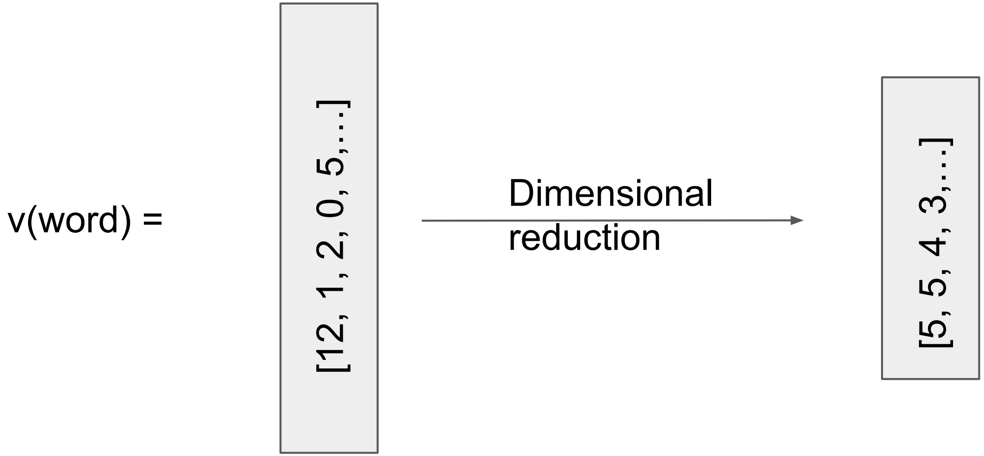
1. Marie rode a ______
2. ______ wheel was punctured
3. The ______ has a beautiful white frame

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

Recall: context embeddings

v(word_i)[j] = count(co-occurrences word_i with word_j)

Example: v(word) = [12,1,2,0,5,...]

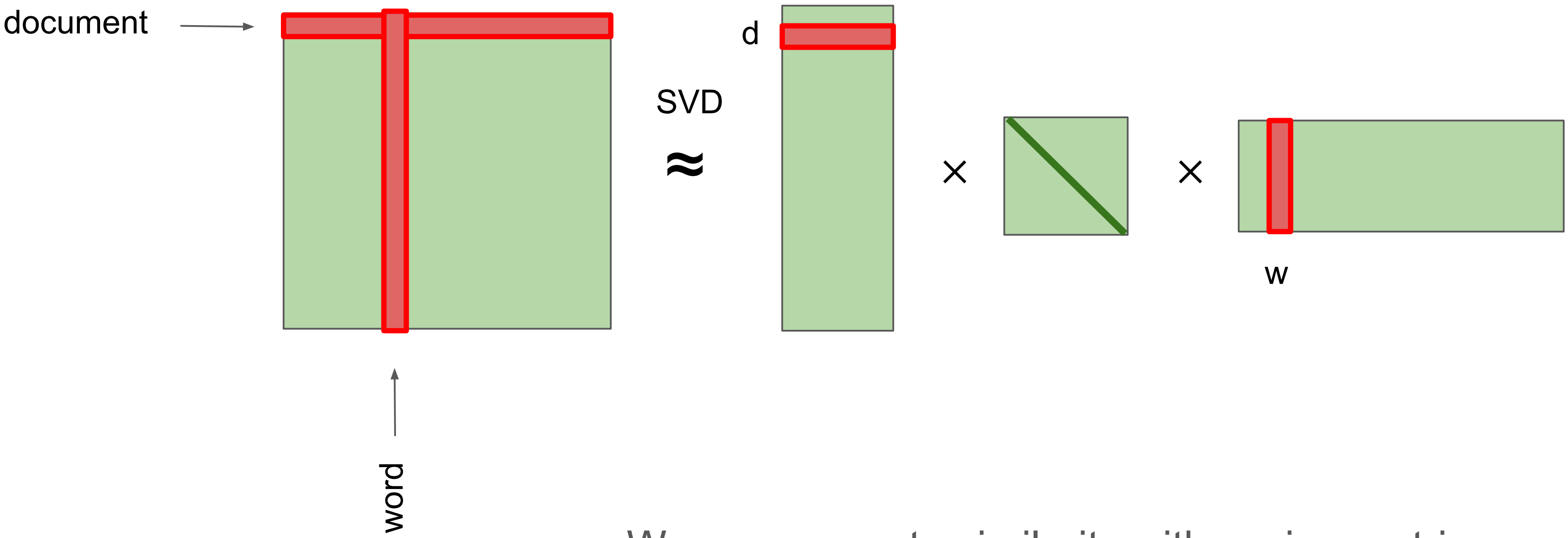


Recall: context embeddings

Problems:

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

Recall: latent semantic analysis



We can compute similarity with cosine metric.

Recall: latent semantic analysis

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

Recall: 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

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

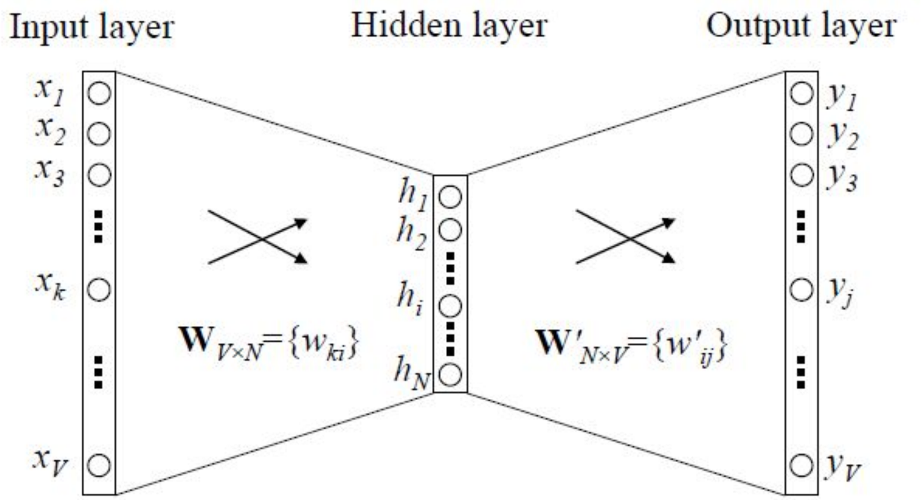
But how about to **learn** those word representations?

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**

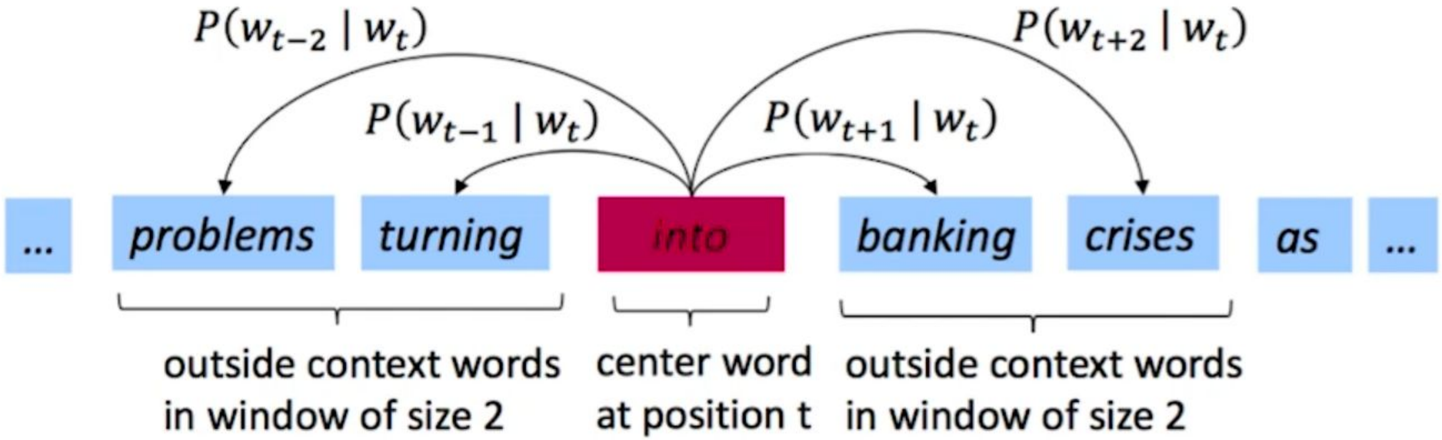
Word2Vec



- ➔ Predict context words given a word (or vice versa)
- ➔ maximize probability of seeing word and its context together

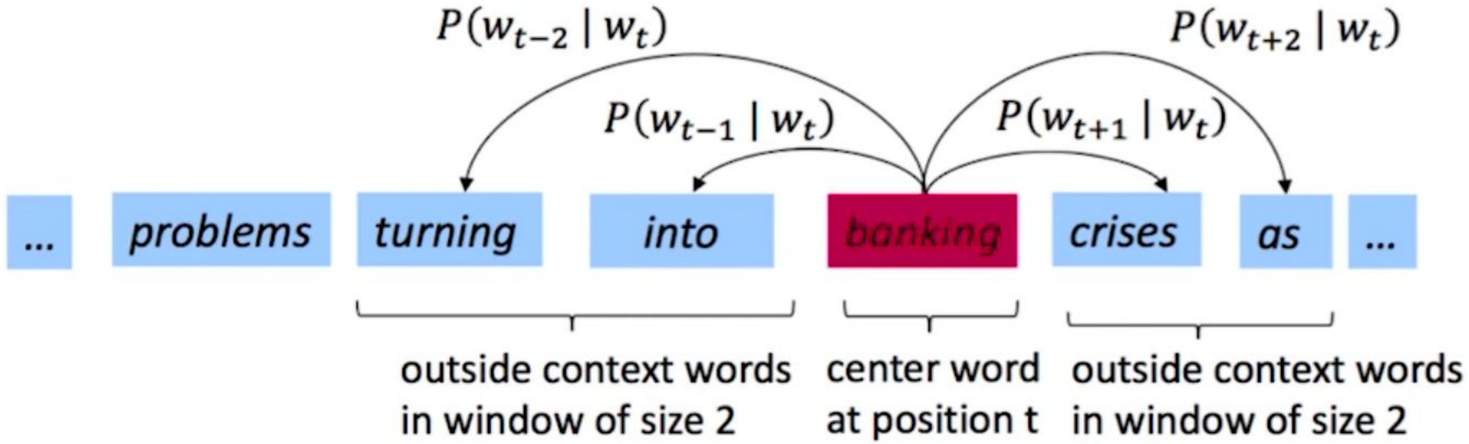
Word2Vec

Going through text corpus by sliding windows



Word2Vec

Going through text corpus by sliding windows



Word2Vec objective function

We want to maximize probability of context words given the center word (log-likelihood):

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

Word2Vec objective function

OR we want to minimize negative log-likelihood:

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

Word2Vec objective function

OR we want to minimize negative log-likelihood:

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

How to calculate $P(w_{t+j} \mid w_t , \theta)$?

Word2Vec objective function

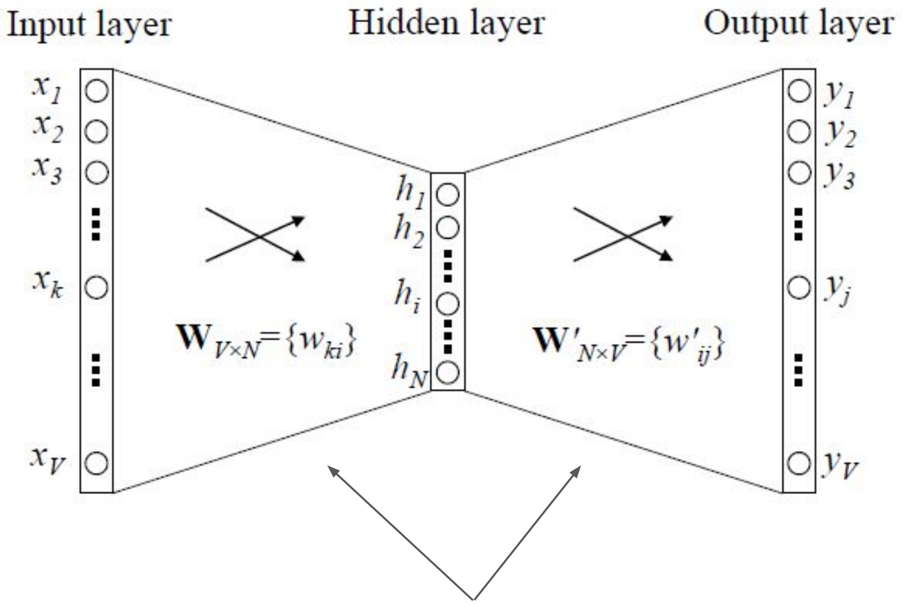
Given a center word **c** and a context word **o**:

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

- 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”

Word2Vec

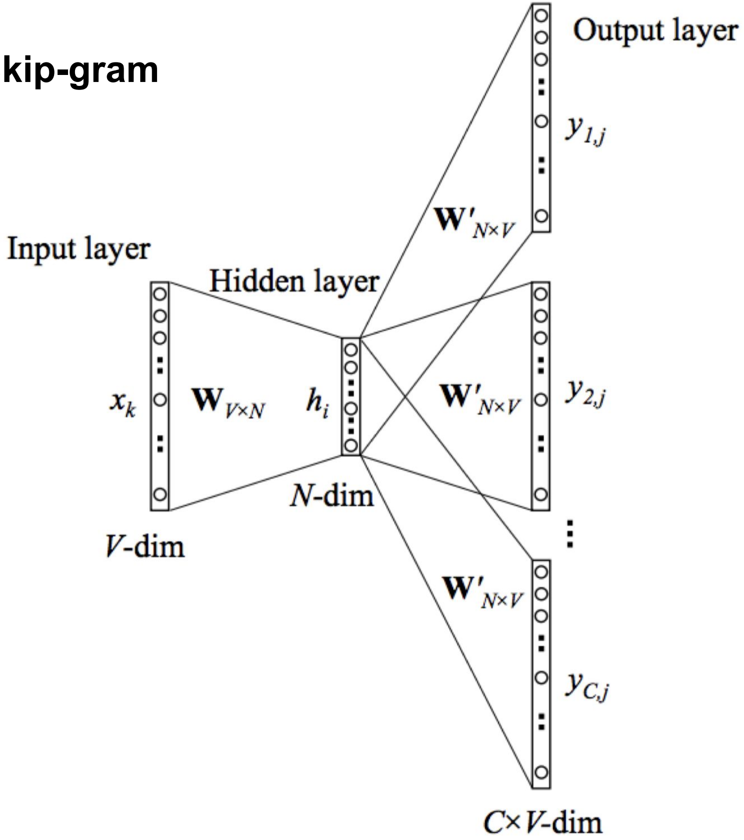


θ (матрицы векторов)

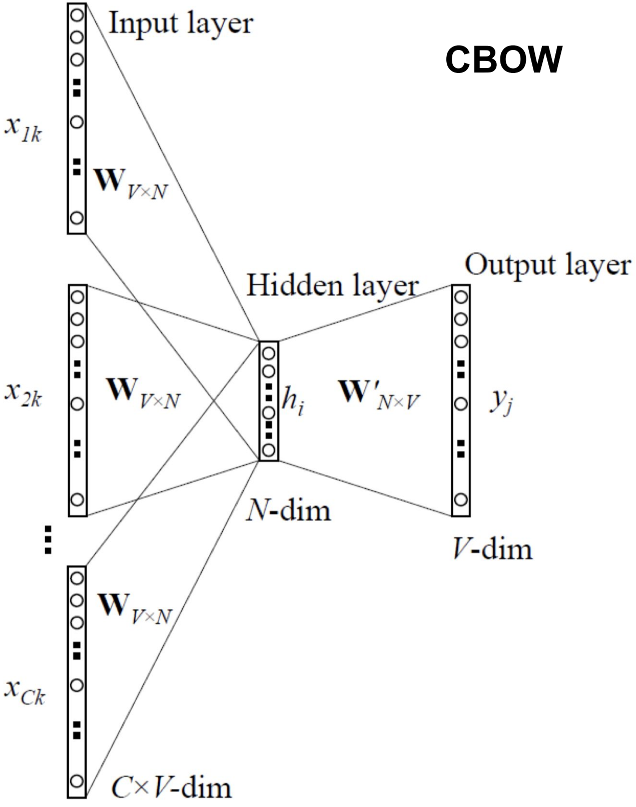
$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Word2Vec

Skip-gram



CBOW



Word2Vec

Problem:

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

Still big sum to compute

Word2Vec

Problem:

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

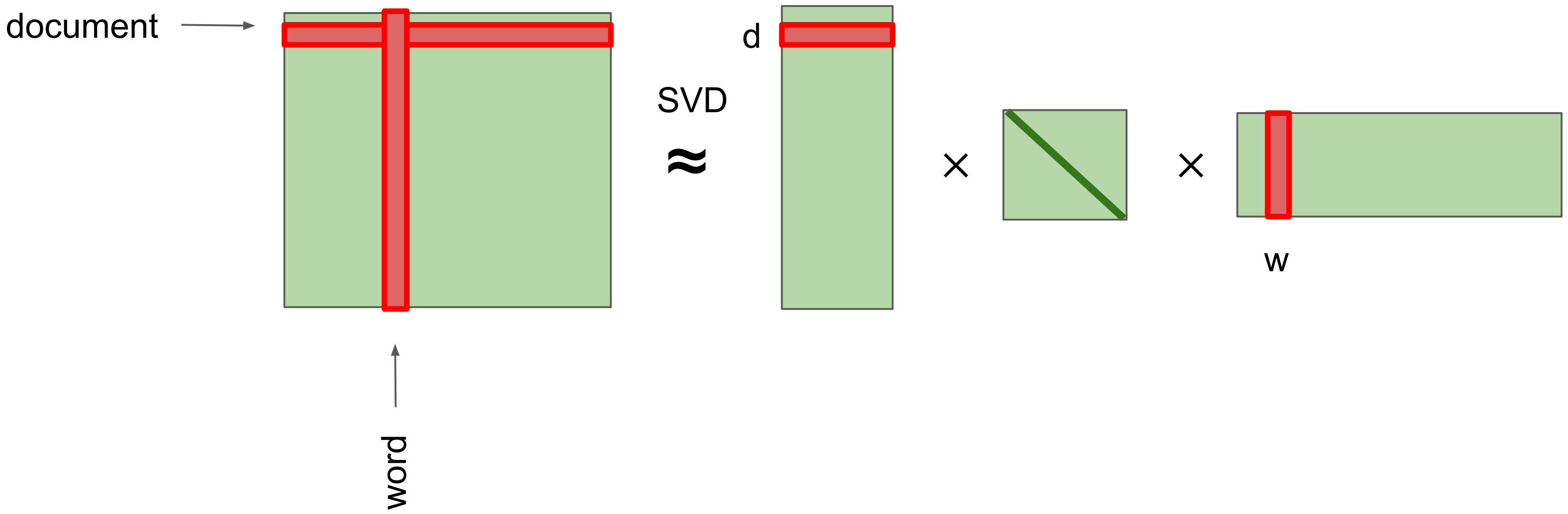
Still big sum to compute

Possible solutions:

- Hierarchical softmax
- Negative sampling

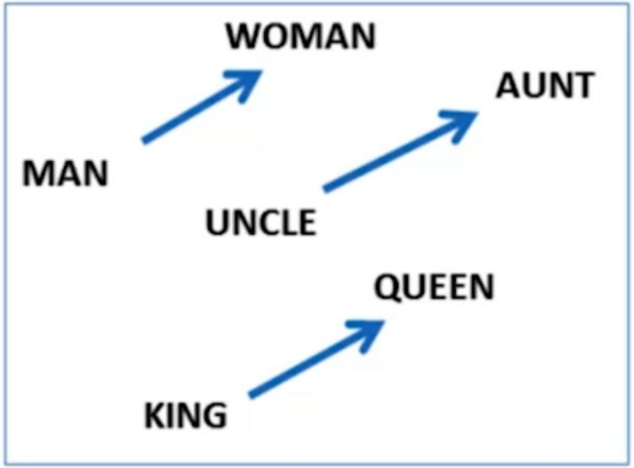
Word2Vec vs SVD

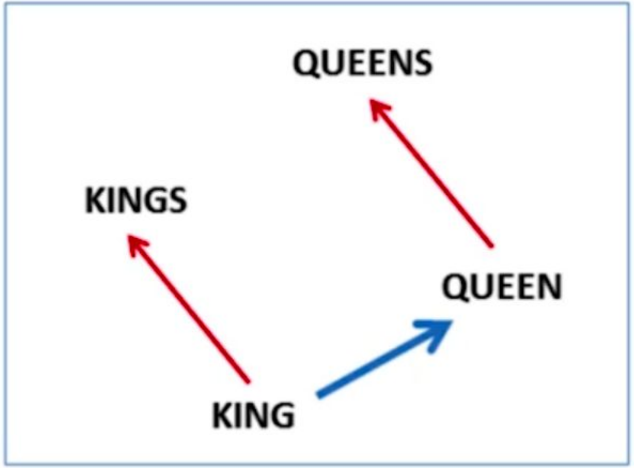
Word2Vec with negative sampling ≈ matrix factorization

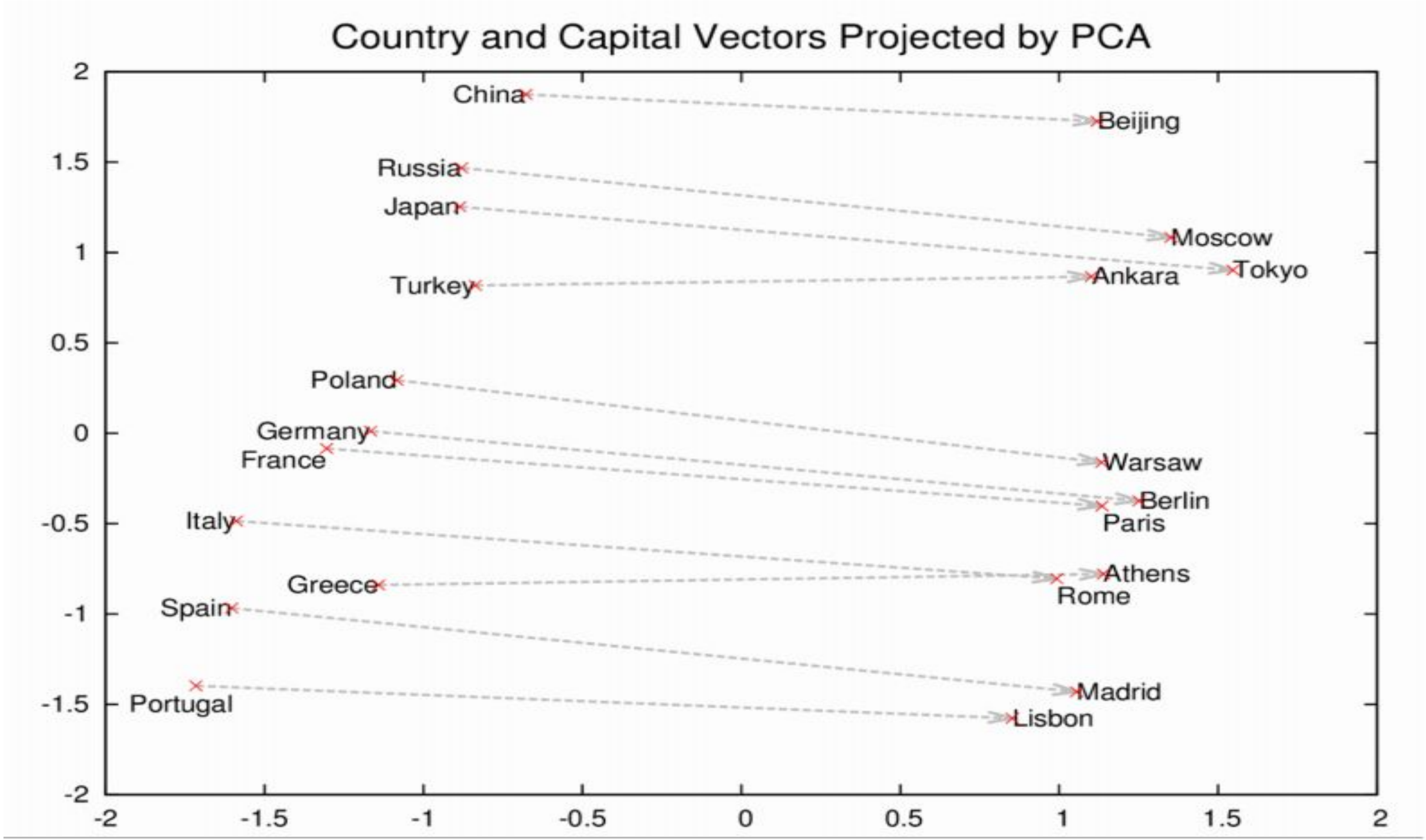


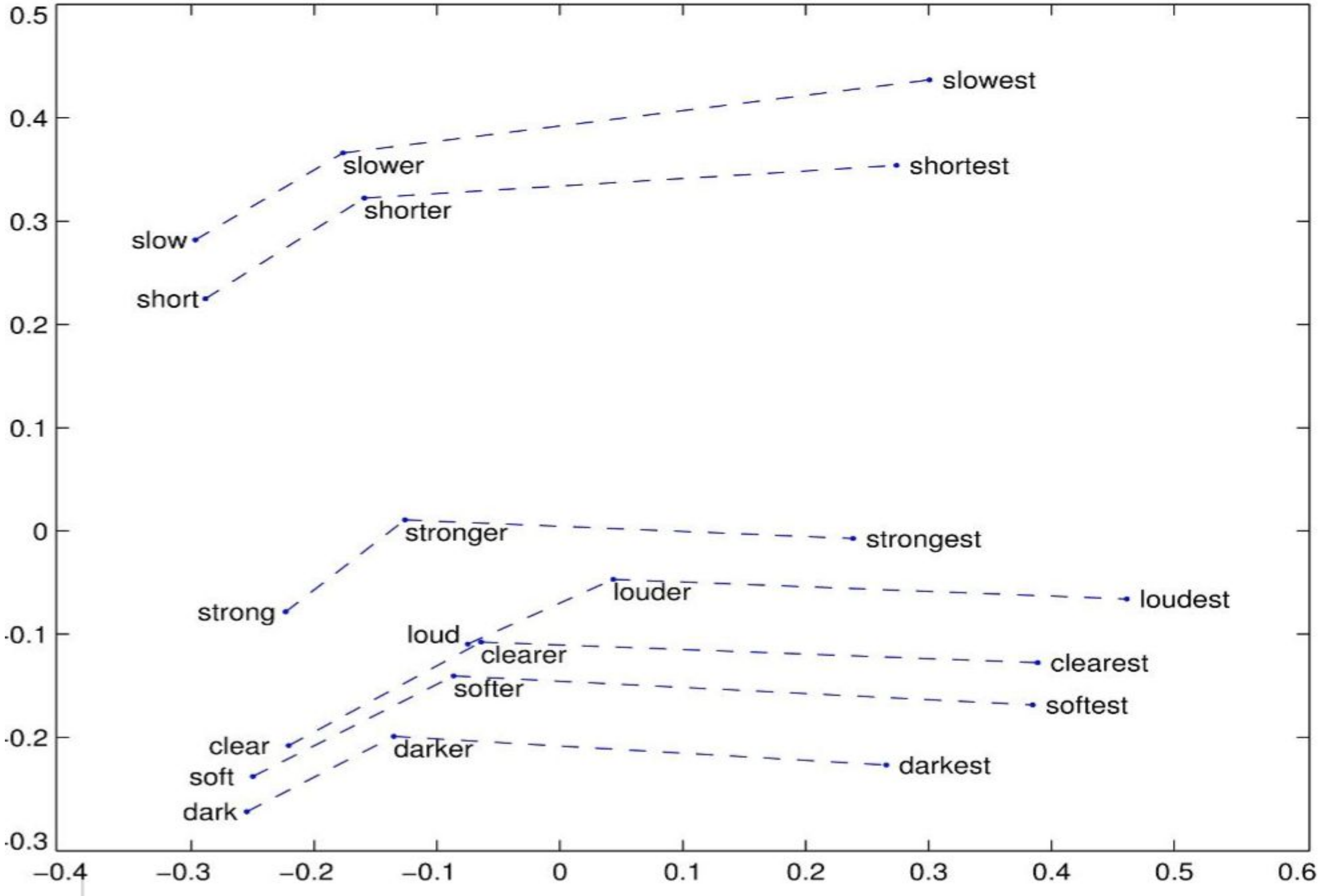
Word2Vec

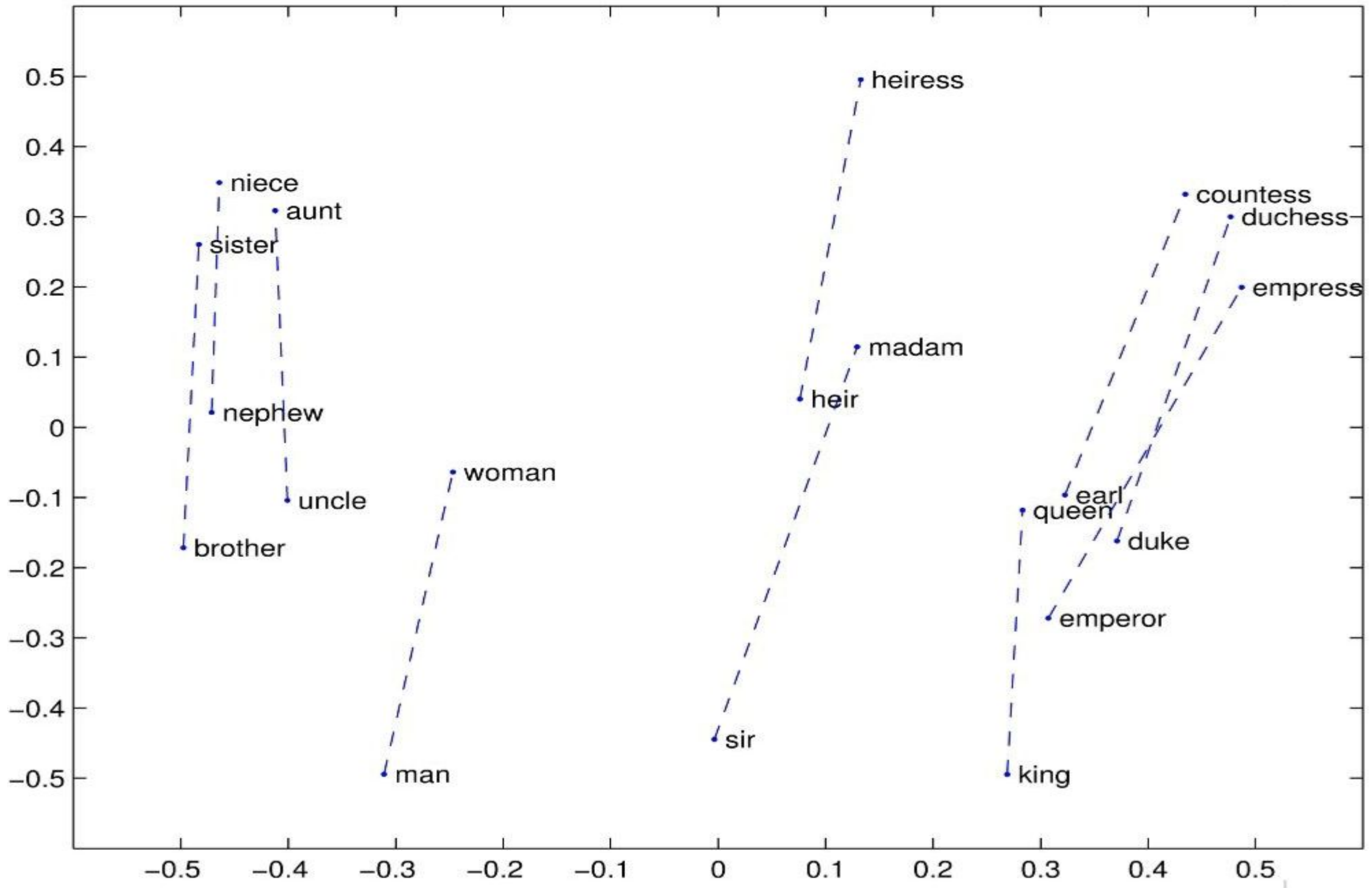
v(king) - v(man) + v(woman) ≈ v(queen)






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


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
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6


heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
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

GloVe

Before training count occurencies of pairs [$word_i$, word□] in corpus

Compute probabilities P_i□ = P ([$word_i$, word□])

Objective function:

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

GloVe

Before training count occurencies of pairs [word$_i$, word□] in corpus

Compute probabilities P_{i}□ = P ([word$_i$, word□])

Objective function:

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

Closely related to
co-occurence matrix

Close to the idea of factorizing
co-occurence 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>
- 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

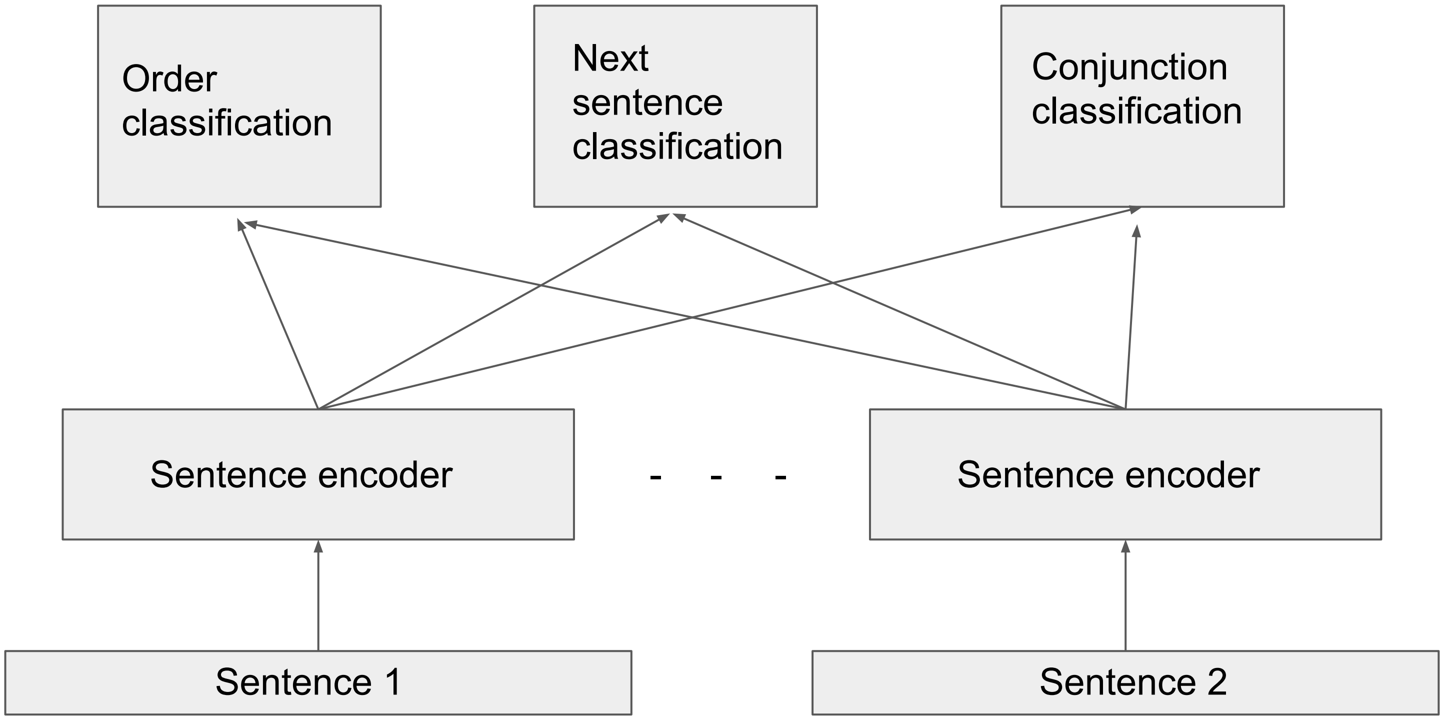
What we did before:

- Predicted context using word

What we'll do now:

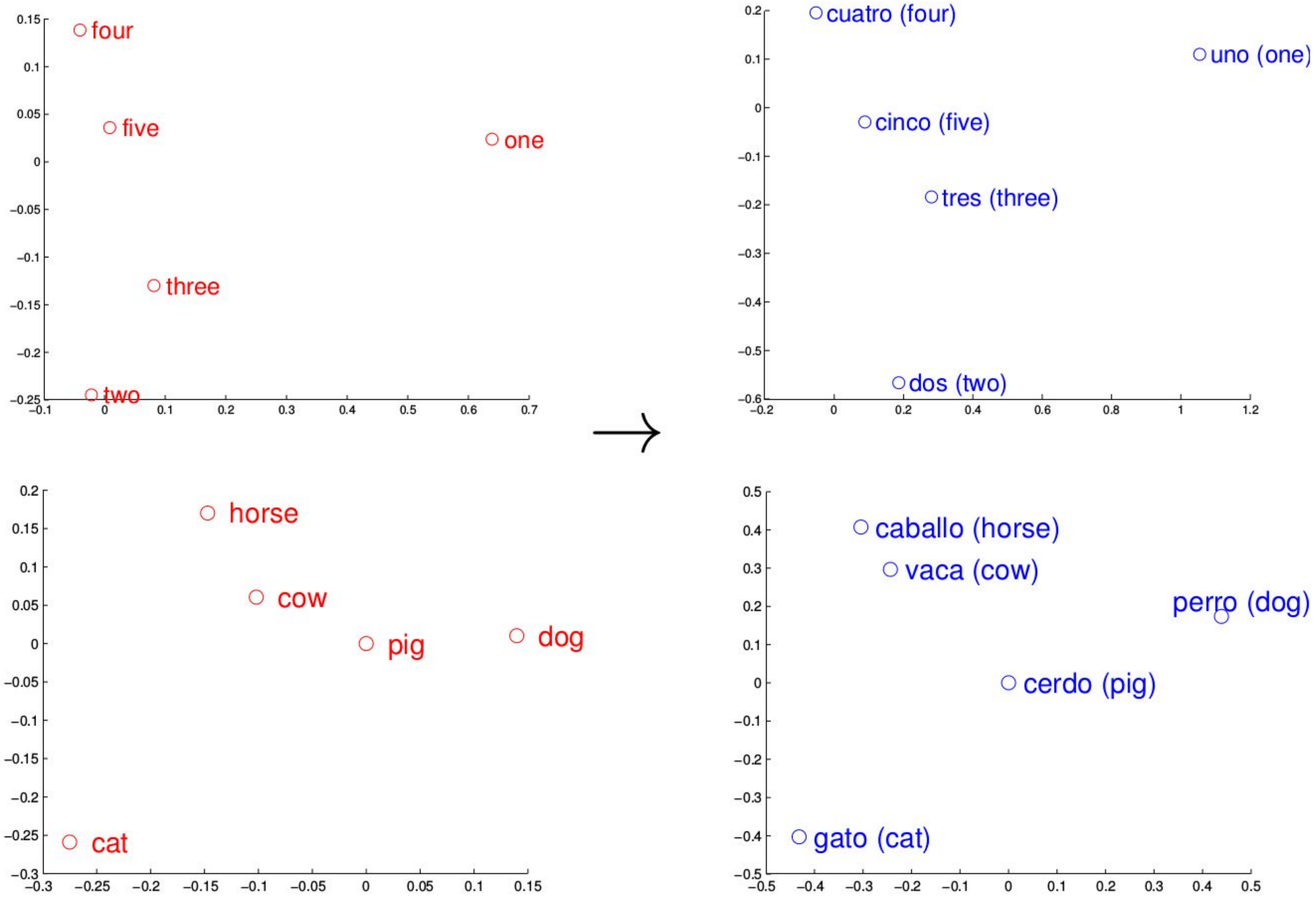
- Predict binary ordering of sentences (does this sentence go next or before?)
- Can this sentence be the next or not? (Next sentence classifier)
- Conjunction prediction

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?

Use embeddings

When you have small text data for your task