

# **Word embeddings**

-----

## **an introduction and applications**

**P. Meladianos, A. Tixier, M. Vazirgiannis**

November 2017

**“a word is defined by “the company it keeps”  
(Firth, 1957)**

# Language model

- Goal: determine  $P(s = w_1 \dots w_k)$  in some domain of interest

$$P(s) = \prod_{i=1}^k P(w_i | w_1 \dots w_{i-1})$$

e.g.,  $P(w_1 w_2 w_3) = P(w_1) P(w_2 | w_1) P(w_3 | w_1 w_2)$

- Traditional n-gram language model assumption:  
“the probability of a word depends only on **context** of  $n - 1$  previous words”

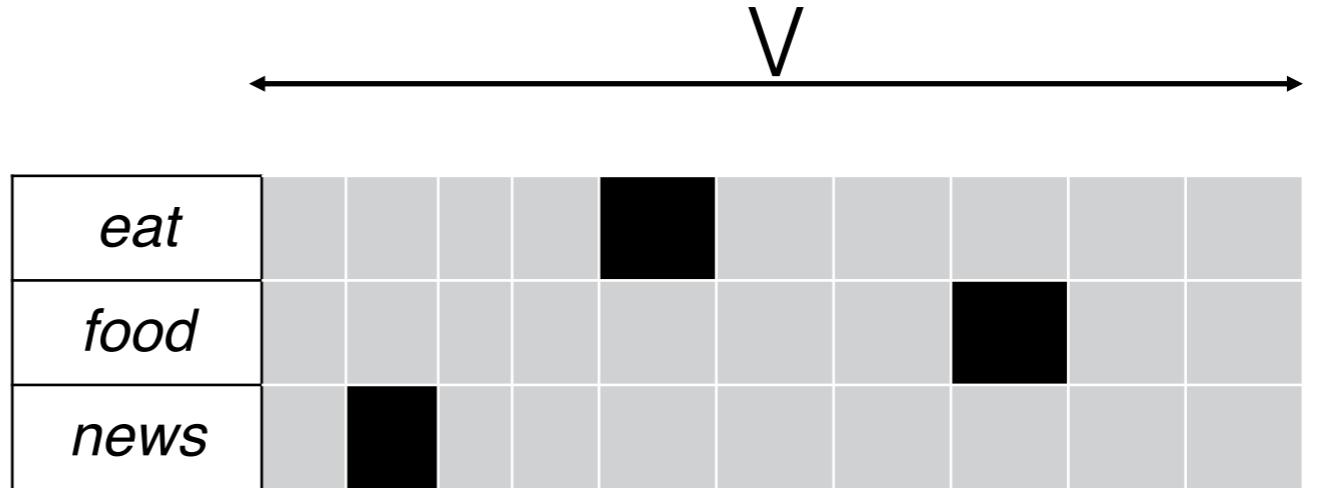
$$\Rightarrow \hat{P}(s) = \prod_{i=1}^k P(w_i | w_{i-n+1} \dots w_{i-1})$$

- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute  $\hat{P}(w_i | w_{i-n+1} \dots w_{i-1}) = \frac{\#w_{i-n+1} \dots w_{i-1} w_i}{\#w_{i-n+1} \dots w_{i-1}}$  on training corpus
  2. smooth to avoid zero probabilities

# Representing Words

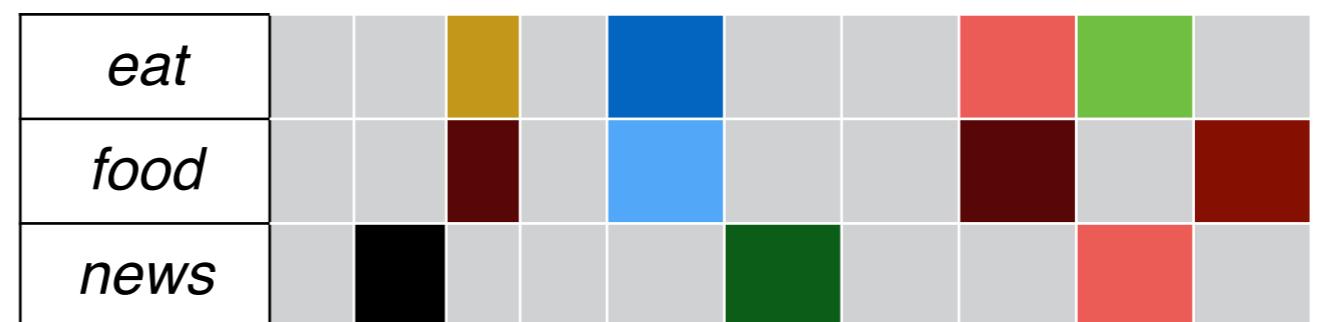
## ➤ One-hot vector

- high dimensionality
- sparse vectors
- dimensions= $|V|$  ( $10^6 < |V|$ )
- unable to capture semantic similarity between words



## ➤ Distributional vector

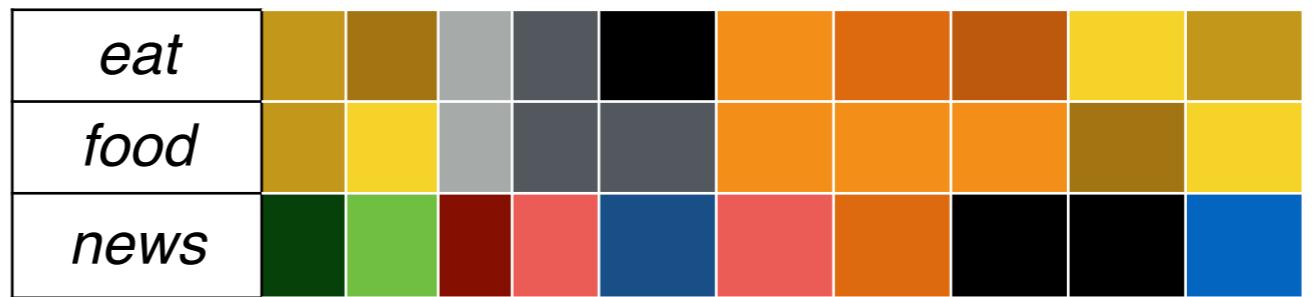
- words that occur in similar contexts, tend to have similar meanings
- each word vector contains the frequencies of all its neighbors
- dimensions= $|V|$
- computational complexity for ML algorithms



# Representing Words

## ➤ Word embeddings

- store the same contextual information in a low-dimensional vector
- **densification** (sparse to dense)
- **compression**
  - dimensionality reduction
  - dimensions=m  
 $100 < m < 500$
- able to capture semantic similarity between words
- learned vectors (unsupervised)
- Learning methods
  - [SVD](#)
  - [word2vec](#)
  - [GloVe](#)

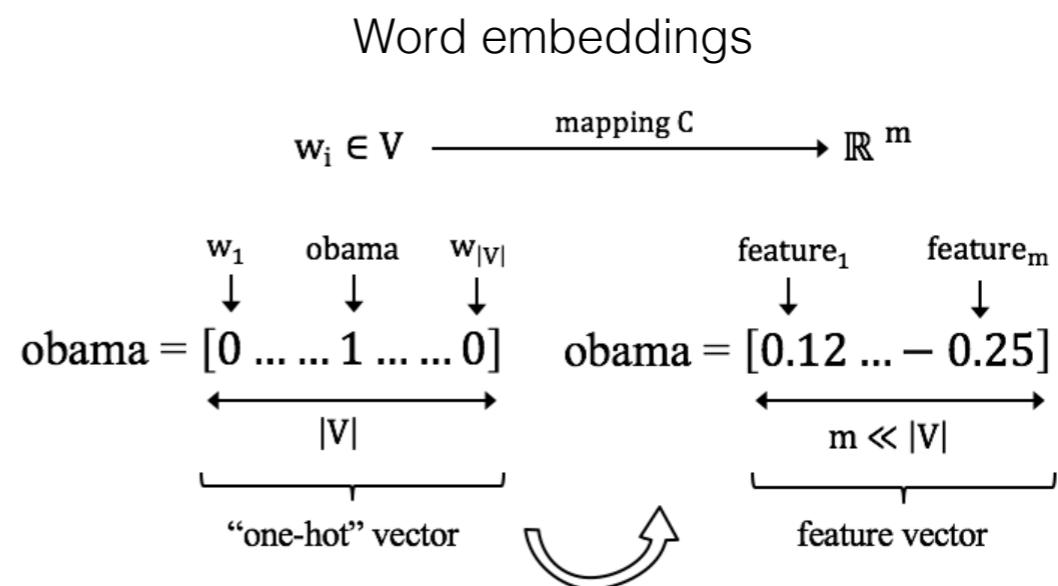


# Example

- We should assign similar probabilities (discover similarity) to *Obama speaks to the media in Illinois* and the *President addresses the press in Chicago*
- This does not happen because of the “one-hot” vector space representation

One hot

$$\begin{aligned}
 \text{obama} &= [0 \ 0 \ 0 \ 0 \dots 0 \ 1 \ 0 \ 0] \\
 \text{president} &= [0 \ 0 \ 0 \ 1 \dots 0 \ 0 \ 0 \ 0] \\
 \text{speaks} &= [0 \ 0 \ 1 \ 0 \dots 0 \ 0 \ 0 \ 0] \\
 \text{addresses} &= [0 \ 0 \ 0 \ 0 \dots 0 \ 0 \ 1 \ 0] \\
 \text{illinois} &= [1 \ 0 \ 0 \ 0 \dots 0 \ 0 \ 0 \ 0] \\
 \text{chicago} &= [0 \ 1 \ 0 \ 0 \dots 0 \ 0 \ 0 \ 0]
 \end{aligned}
 \quad \left. \begin{array}{l} \xrightarrow{\text{obama}, \text{president}} = \vec{0} \\ \xrightarrow{\text{speaks}, \text{addresses}} = \vec{0} \\ \xrightarrow{\text{illinois}, \text{chicago}} = \vec{0} \end{array} \right\}$$



# SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a  $|V| \times |V|$  word co-occurrence matrix  $X$
- Apply SVD  $X = USV^T$
- Take first  $k$  columns of  $U$
- Use the  $k$ -dimensional vectors as representations for each word
- Able to capture semantic and syntactic similarity

# Latent Semantic Indexing (LSI) -I

---

- Finding similarity with exact keyword matching is problematic.
  - Using SVD we process the initial document-term document.
  - Then we choose the  $k$  larger singular values. The resulting matrix is of order  $k$  and is the most similar to the original one based on the Frobenius norm than any other  $k$ -order matrix.
-

# Latent Semantic Indexing (LSI) - II

---

- The initial matrix is SVD decomposed as:  $A=ULV^T$
- Choosing the top-k singular values from L we have:

$$A_k = U_k L_k V_k^T,$$

- $L_k$  square  $k \times k$  - top-k singular values of the diagonal in matrix L,
- $U_k$ ,  $m \times k$  matrix - first k columns in U (left singular vectors)
- $V_k^T$ ,  $k \times n$  matrix - first k lines of  $V^T$  (right singular vectors)

Typical values for  $k \sim 200-300$  (empirically chosen based on experiments appearing in the bibliography)

---

# LSI capabilities

- Term to term similarity:  $A_k A_k^T = U_k L_k^2 U_k^T$ 
  - Where  $A_k = U_k L_k V_t$
- Document-document similarity:  $A_k^T A_k = V_k L_k^2 V_k^T$
- Term document similarity (as an element of the transformed – document matrix)
- Extended query capabilities transforming initial query  $q$  to  $q_n$   $q_n = q^T U_k L_k^{-1}$
- Thus  $q_n$  can be regarded a line in matrix  $V_k$

# LSI – an example

---

## LSI application on a term – document matrix

C1: Human machine Interface for Lab ABC computer application

C2: A survey of user opinion of computer system response time

C3: The EPS user interface management system

C4: System and human system engineering testing of EPS

C5: Relation of user-perceived response time to error measurements

M1: The generation of random, binary unordered trees

M2: The intersection graph of path in trees

M3: Graph minors IV: Widths of trees and well-quasi-ordering

M4: Graph minors: A survey

- The dataset consists of 2 classes, 1st: “human – computer interaction” (c1-c5) 2nd: related to graph (m1-m4). After feature extraction the titles are represented as follows.

# LSI – an example

# LSI – an example

$$A = U L V^T$$

A =

# LSI – an example

$$A = U L V^T$$

$U =$

0.22	-0.11	0.29	-0.41	-0.11	-0.34	0.52	-0.06	-0.41	0	0	0
0.20	-0.07	0.14	-0.55	0.28	0.50	-0.07	-0.01	-0.11	0	0	0
0.24	0.04	-0.16	-0.59	-0.11	-0.25	-0.30	0.06	0.49	0	0	0
0.40	0.06	-0.34	0.10	0.33	0.38	0.00	0.00	0.01	0	0	0
0.64	-0.17	0.36	0.33	-0.16	-0.21	-0.17	0.03	0.27	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.27	0.11	-0.43	0.07	0.08	-0.17	0.28	-0.02	-0.05	0	0	0
0.30	-0.14	0.33	0.19	0.11	0.27	0.03	-0.02	-0.17	0	0	0
0.21	0.27	-0.18	-0.03	-0.54	0.08	-0.47	-0.04	-0.58	0	0	0
0.01	0.49	0.23	0.03	0.59	-0.39	-0.29	0.25	-0.23	0	0	0
0.04	0.62	0.22	0.00	-0.07	0.11	0.16	-0.68	0.23	0	0	0
0.03	0.45	0.14	-0.01	-0.30	0.28	0.34	0.68	0.18	0	0	0

# LSI – an example

$$A = U L V^T$$

L =

# LSI – an example

$$A = U L V^T$$

$V =$

0.20	-0.06	0.11	-0.95	0.05	-0.08	0.18	-0.01	-0.06
0.61	0.17	-0.50	-0.03	-0.21	-0.26	-0.43	0.05	0.24
0.46	-0.13	0.21	0.04	0.38	0.72	-0.24	0.01	0.02
0.54	-0.23	0.57	0.27	-0.21	-0.37	0.26	-0.02	-0.08
0.28	0.11	-0.51	0.15	0.33	0.03	0.67	-0.06	-0.26
0.00	0.19	0.10	0.02	0.39	-0.30	-0.34	0.45	-0.62
0.01	0.44	0.19	0.02	0.35	-0.21	-0.15	-0.76	0.02
0.02	0.62	0.25	0.01	0.15	0.00	0.25	0.45	0.52
0.08	0.53	0.08	-0.03	-0.60	0.36	0.04	-0.07	-0.45

# LSI – an example

Choosing the 2 largest singular values we have

0.22	-0.11
0.20	-0.07
0.24	0.04
0.40	0.06
0.64	-0.17
0.27	0.11
0.27	0.11
0.30	-0.14
0.21	0.27
0.01	0.49
0.04	0.62
0.03	0.45

$$L_k = \begin{bmatrix} 3.34 & 0 \\ 0 & 2.54 \end{bmatrix}$$

$$U_k = \begin{bmatrix} 0.20 & 0.61 & 0.46 & 0.54 & 0.28 & 0.00 & 0.02 & 0.02 & 0.08 \\ -0.06 & 0.17 & -0.13 & -0.23 & 0.11 & 0.19 & 0.44 & 0.62 & 0.53 \end{bmatrix}$$

$$V_k^T =$$

# LSI (2 singular values)

$A_k =$

	C1	C2	C3	C4	C5	M1	M2	M3	M4
human	0.16	0.40	0.38	0.47	0.18	-0.05	-0.12	-0.16	-0.09
Interface	0.14	0.37	0.33	0.40	0.16	-0.03	-0.07	-0.10	-0.04
Computer	0.15	0.51	0.36	0.41	0.24	0.02	0.06	0.09	0.12
User	0.26	0.84	0.61	0.70	0.39	0.03	0.08	0.12	0.19
System	0.45	1.23	1.05	1.27	0.56	-0.07	-0.15	-0.21	-0.05
Response	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
Time	0.16	0.58	0.38	0.42	0.28	0.06	0.13	0.19	0.22
EPS	0.22	0.55	0.51	0.63	0.24	-0.07	-0.14	-0.20	-0.11
Survey	0.10	0.53	0.23	0.21	0.27	0.14	0.31	0.44	0.42
Trees	-0.06	0.23	-0.14	-0.27	0.14	0.24	0.55	0.77	0.66
Graph	-0.06	0.34	-0.15	-0.30	0.20	0.31	0.69	0.98	0.85
Minors	-0.04	0.25	-0.10	-0.21	0.15	0.22	0.50	0.71	0.62

## LSI Example

---

- Query: “human computer interaction” retrieves documents:  $c_1, c_2, c_4$  but *not*  $c_3$  and  $c_5$ .
- If we submit the same query (based on the transformation shown before) to the transformed matrix we retrieve (using cosine similarity) all  $c_1-c_5$  even if  $c_3$  and  $c_5$  have no common keyword to the query.
- According to the transformation for the queries we have:

# Query transformation

---

	query
human	1
Interface	0
computer	1
User	0
System	0
Response	0
Time	0
EPS	0
Survey	0
Trees	0
Graph	0
Minors	0

$$q = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

# Query transformation

---

$$q^T = [1 \quad 0 \quad 1 \quad 0 \quad 0]$$

$$U_k = \begin{array}{|c|c|} \hline 0.22 & -0.11 \\ \hline 0.20 & -0.07 \\ \hline 0.24 & 0.04 \\ \hline 0.40 & 0.06 \\ \hline 0.64 & -0.17 \\ \hline 0.27 & 0.11 \\ \hline 0.27 & 0.11 \\ \hline 0.30 & -0.14 \\ \hline 0.21 & 0.27 \\ \hline 0.01 & 0.49 \\ \hline 0.04 & 0.62 \\ \hline 0.03 & 0.45 \\ \hline \end{array}$$

$$L_k = \begin{array}{|c|c|} \hline 0.3 & 0 \\ \hline 0 & 0.39 \\ \hline \end{array}$$

$$q_n = q^T U_k L_k = \begin{array}{|c|c|} \hline 0.138 & - \\ \hline & 0.0273 \\ \hline \end{array}$$

# Query transformation

---

Map  
docs to  
the 2  
dim  
space  
 $V_k L_k =$

0.20	-0.06
0.61	0.17
0.46	-0.13
0.54	-0.23
0.28	0.11
0.00	0.19
0.01	0.44
0.02	0.62
0.08	0.53

3.34	0
0	2.54

0.67	-0.15
2.04	0.43
1.54	-0.33
1.80	-0.58
0.94	0.28
0.00	0.48
0.03	1.12
0.07	1.57
0.27	1.35

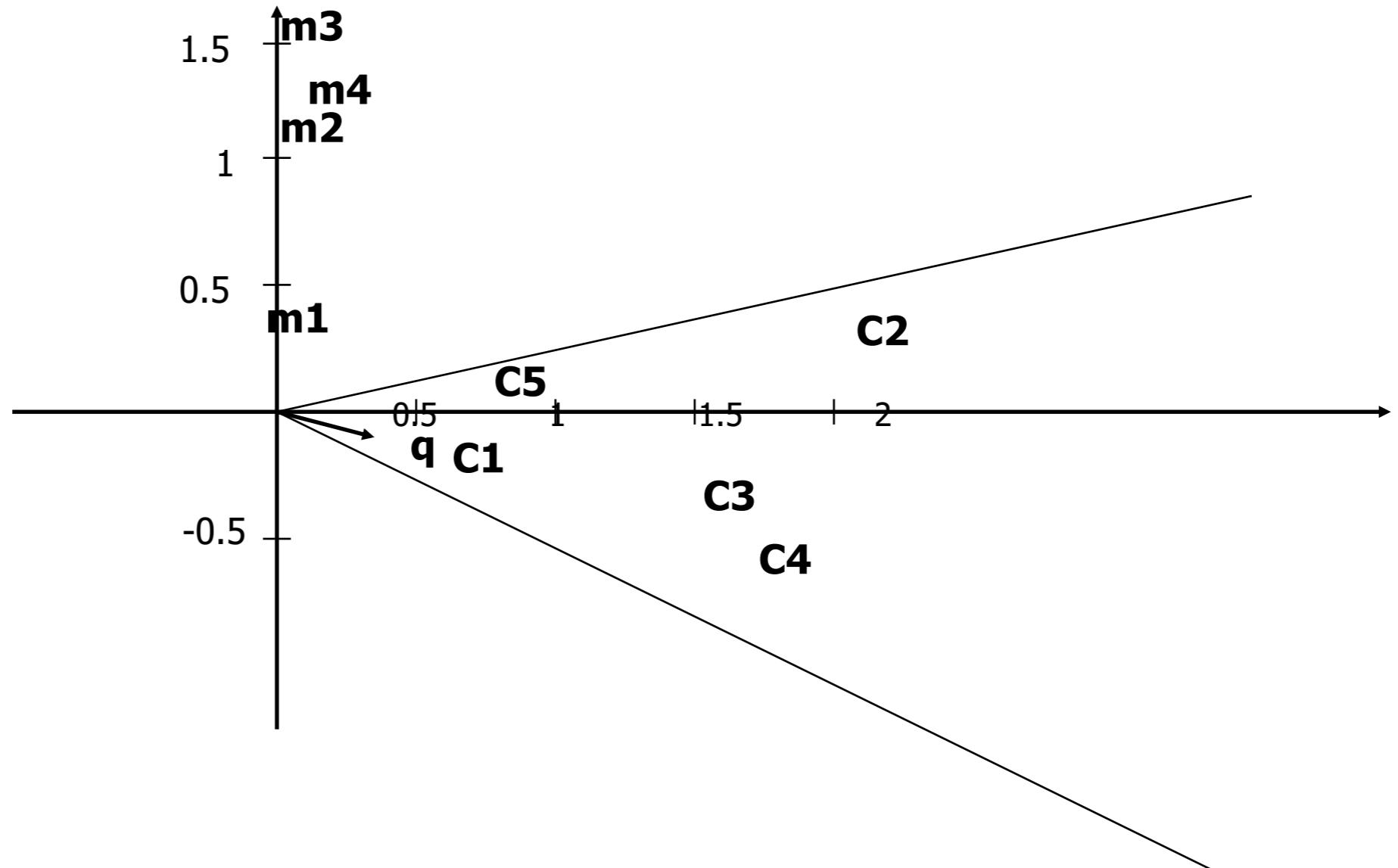
$$q_n L_k = \begin{array}{|c|c|} \hline 0.138 & - \\ \hline & 0.0273 \\ \hline \end{array}$$

$$\begin{array}{|c|c|} \hline 3.34 & 0 \\ \hline 0 & 2.54 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 0.46 & -0.069 \\ \hline \end{array}$$

# Query transformation

- The cosine similarity matrix of query vector to the documents is:

query	
C1	0.99
C2	0.94
C3	0.99
C4	0.99
C5	0.90
M1	-0.14
M2	-0.13
M3	-0.11
M4	0.05

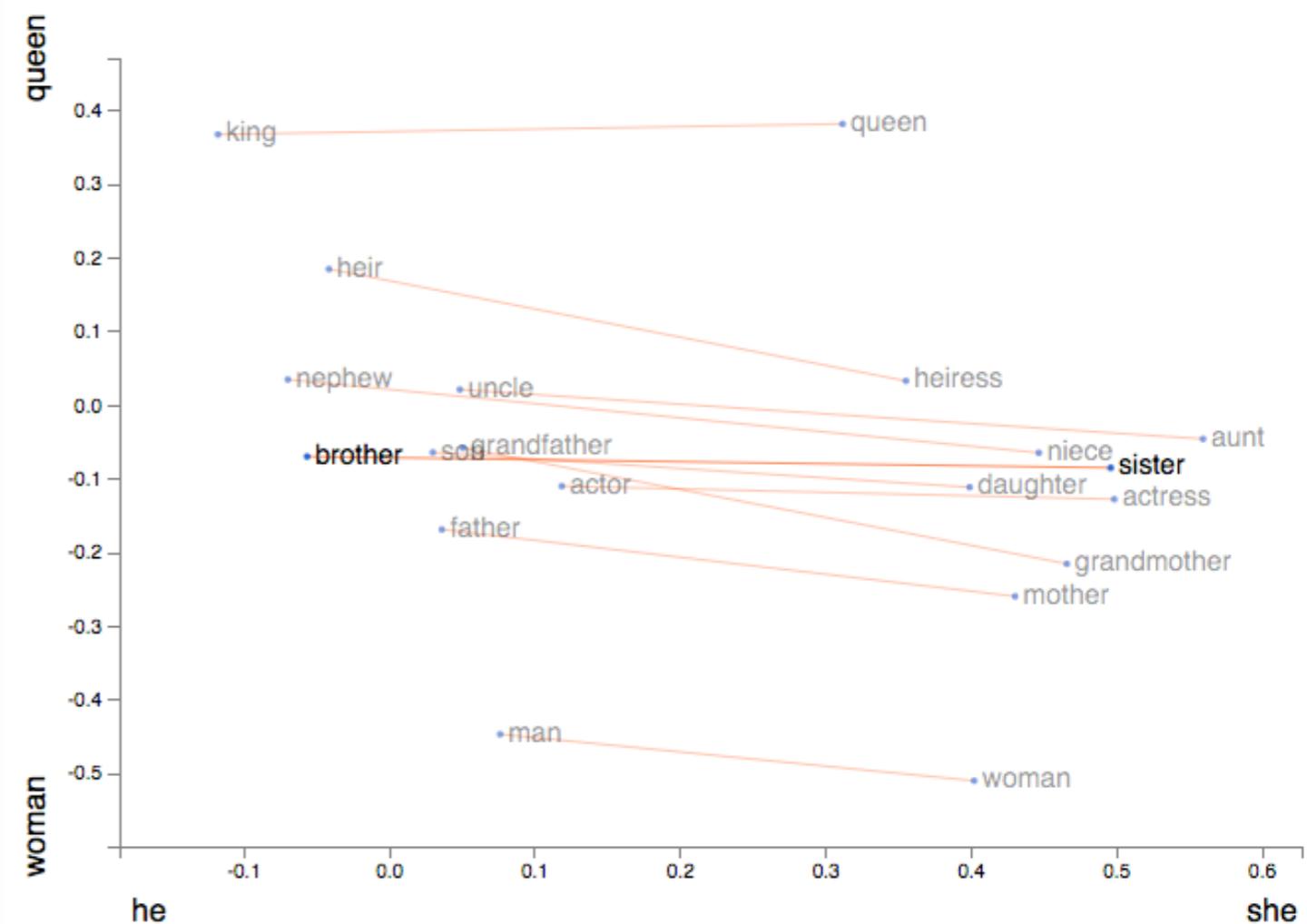


# SVD problems

- The dimensions of the matrix change when dictionary changes
- The whole decomposition must be re-calculated when we add a word
- Sensitive to the imbalance in word frequency
- Very high dimensional matrix
- Not suitable for millions of words and documents
- Quadratic cost to perform SVD
- Solution: Directly calculate a low-dimensional representation

# Word analogy

- Words with similar meaning end up laying close to each other
- Words that share similar contexts may be analogous
  - Synonyms
  - Antonyms
  - Names
  - Colors
  - Places
  - Interchangeable words
- Vector arithmetics to work with analogies
- i.e. **king - man + woman = queen**



<https://lamyiowce.github.io/word2viz/>

# But why?

- what's an analogy?

$$\frac{p(w'|man)}{p(w'|woman)} \approx \frac{p(w'|king)}{p(w'|queen)}$$

Assume PMI is approximated by a low rank approximation of the co-occurrence matrix.

1.  $PMI(w', w) \approx v_w v_{w'}^T$  \*inner product\*
2. Isotropic:  $E_{w'}[(v_{w'} v_u)]^2 = \|v_u\|^2$

Then

3.  $\operatorname{argmin}_w E_{w'} [\ln \frac{p(w'|w)}{p(w'|queen)} - \ln \frac{p(w'|man)}{p(w'|woman)}]^2$
4.  $\operatorname{argmin}_w E_{w'} [(PMI(w'|w) - PMI(w'|queen)) - (PMI(w'|man) - PMI(w'|woman))]^2$
5.  $\operatorname{argmin}_w \|(v_w - v_{queen}) - (v_{man} - v_{woman})\|^2$
6.  $v_w \approx v_{queen} - v_{woman} + v_{man}$  which is an analogy!

- Arora et al (ACL 2016) shows that if (2) holds then (1) holds as well
- So we need to construct vectors from co-occurrence that satisfy (2)
- $d < |V|$  in order to have isotropic vectors

# Learning Word Vectors

- Corpus containing a sequence of  $T$  training words

- Objective:  $f(w_t, \dots, w_{t-n+1}) = \hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$

- Decomposed in two parts:

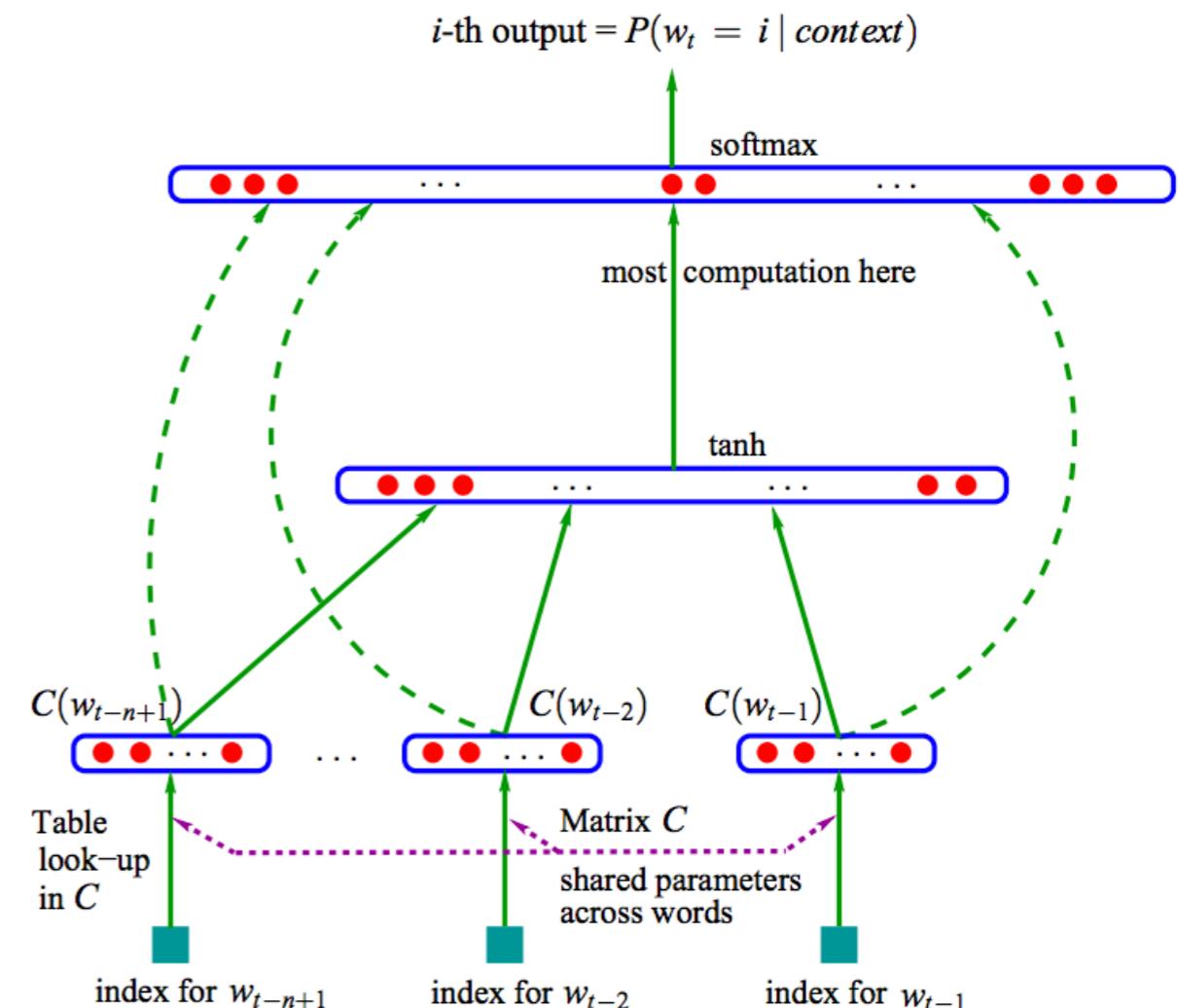
$$w_i \in V \xrightarrow{\text{mapping } C} \mathbb{R}^m$$

- Mapping **C** (1-hotv => lower dimensions)
- Mapping any **g** s.t. (estimate prob  $t+1 | t$  previous)

$$f(w_{t-1}, \dots, w_{t-n+1}) = g(C(w_{t-1}), \dots, C(w_{t-n+1}))$$

- $C(i)$  is the  $i$ -th word feature vector (Word embedding)

- Objective function:  $J = \frac{1}{T} \sum f(w_t, \dots, w_{t-n+1})$

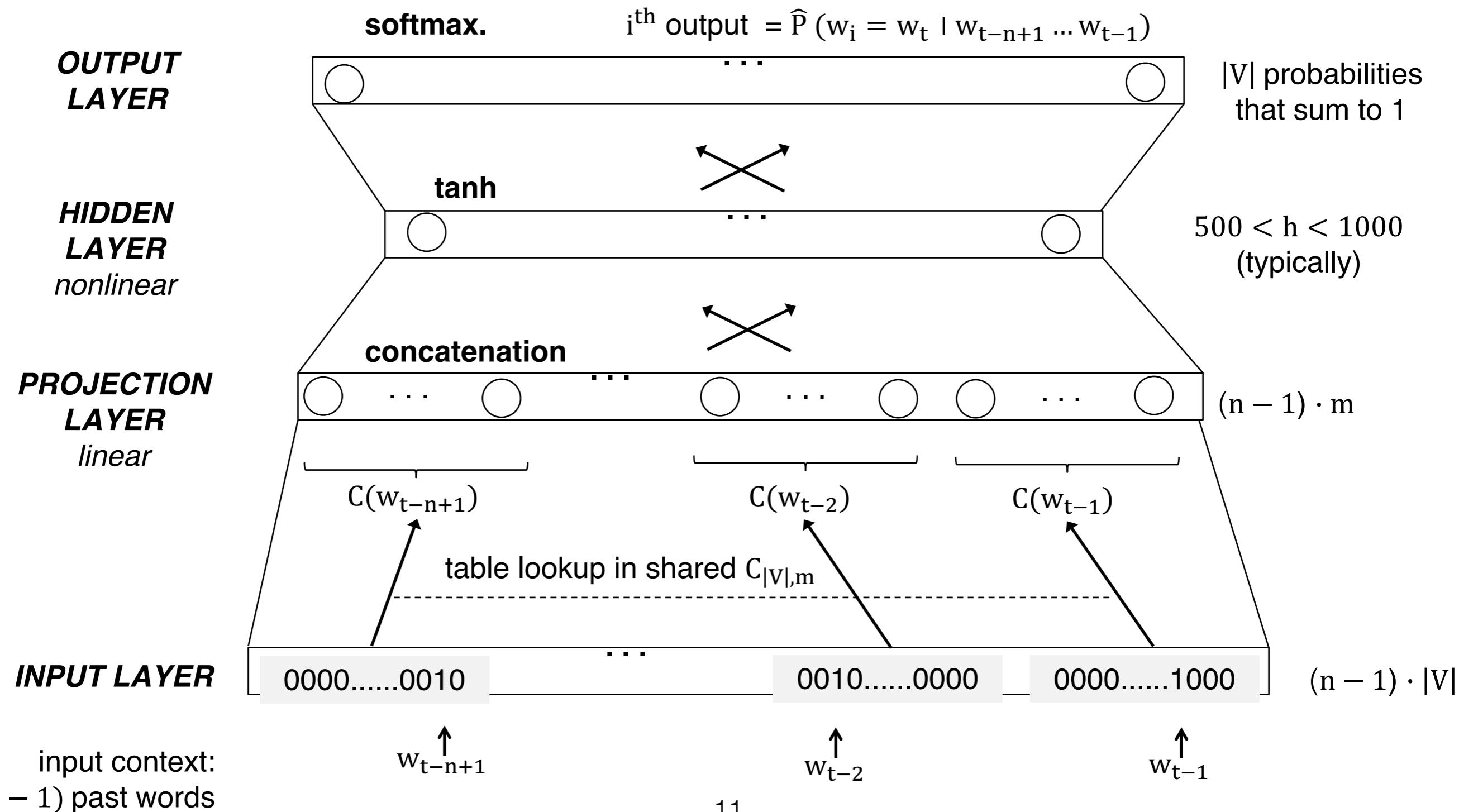


[Bengio, Yoshua, et al. "A neural probabilistic language model."](#)  
[The Journal of Machine Learning Research 3 \(2003\): 1137-1155.](#)

# Neural Net Language Model

For each training sequence:

input = (context, target) pair:  $(w_{t-n+1} \dots w_{t-1}, w_t)$   
 objective: minimize  $E = -\log \hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$



# Objective function

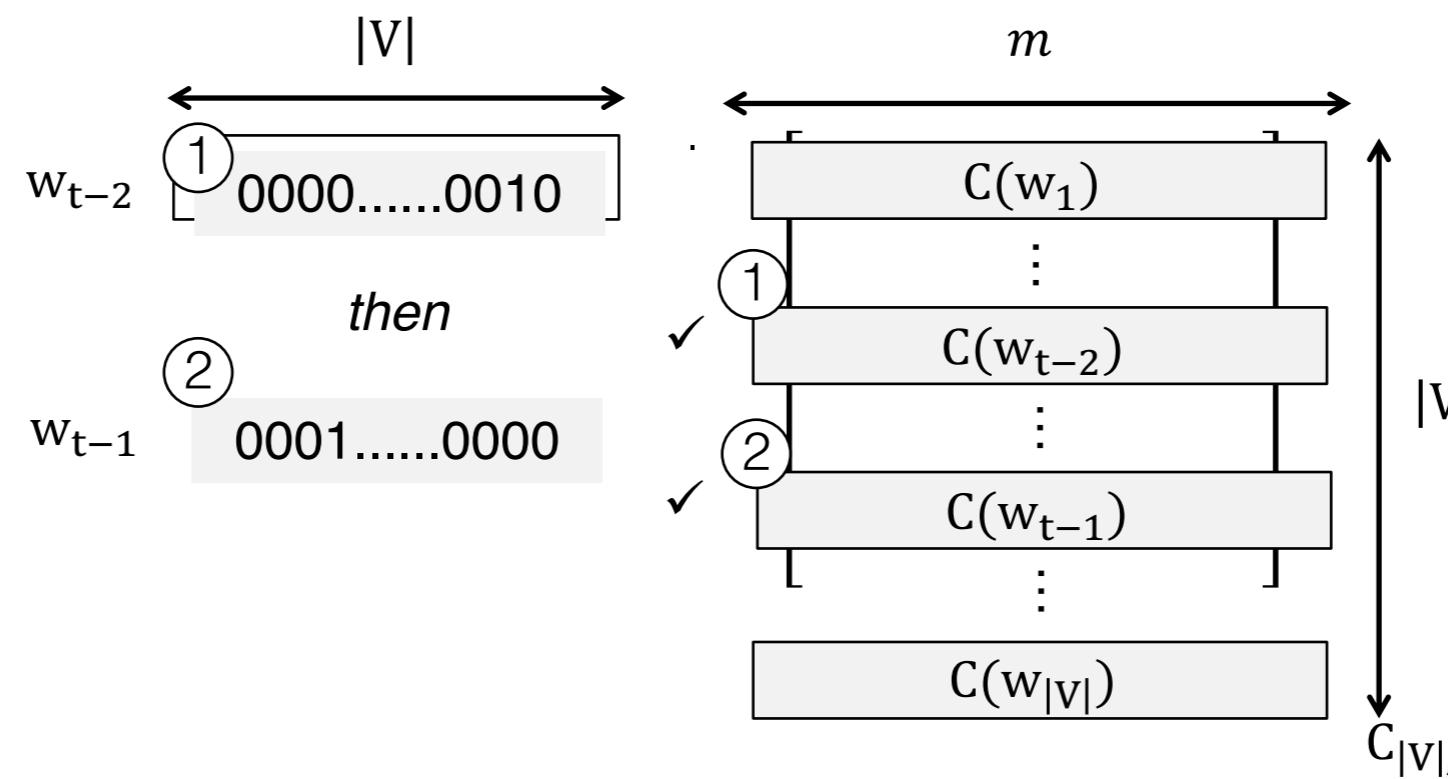
- $E = -\log \hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$
- a probability between 0 and 1.
- On this support, the log is negative  $\Rightarrow -\log$  term positive.
- makes sense to try to minimize it.
  - Probability of word given the context be as high as possible (1 for a perfect prediction).
  - case the error is equal to 0 (global minimum).

p	log(p)	-log(p)
0,7	<b>-0,15490196</b>	<b>0,15490196</b>
0,2	<b>-0,698970004</b>	<b>0,698970004</b>

# NNLM Projection layer

- Performs a simple table lookup in  $C_{|V|,m}$ : concatenate the rows of the shared mapping matrix  $C_{|V|,m}$  corresponding to the context words

Example for a two-word context  $w_{t-2}w_{t-1}$ :



Concatenate ① and ② →  $C(w_{t-2})$   $C(w_{t-1})$

- $C_{|V|,m}$  is **critical**: it contains the weights that are tuned at each step. After training, it contains what we're interested in: the **word vectors**

# NNLM hidden/output layers and training

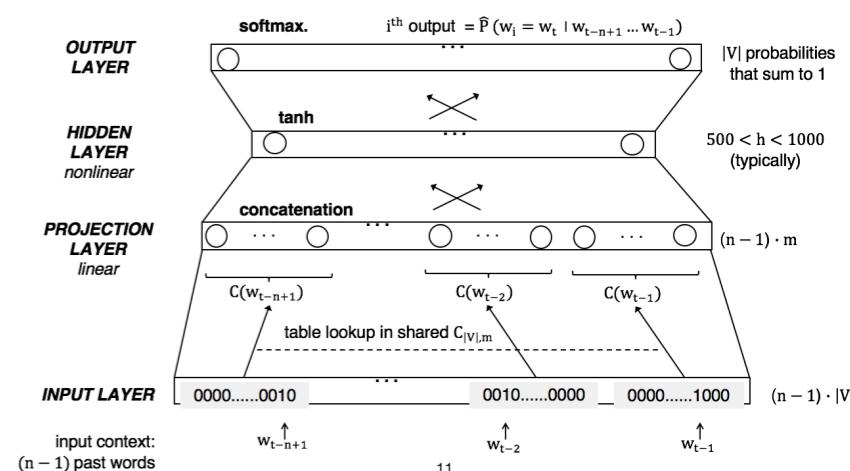
- Softmax (log-linear classification model) is used to output positive numbers that sum to one (a multinomial probability distribution):

for the  $i^{\text{th}}$  unit in the output layer:  $\hat{P}(w_i = w_t \mid w_{t-n+1} \dots w_{t-1}) = \frac{e^{y_{w_i}}}{\sum_{i'=1}^{|V|} e^{y_{w_{i'}}}}$

Where:

- $y = b + U \cdot \tanh(d + H \cdot x)$
- $\tanh$  : nonlinear squashing (link) function
- $x$  : concatenation  $C(w)$  of the context weight vectors seen previously
- $b$  : output layer biases ( $|V|$  elements)
- $d$  : hidden layer biases ( $h$  elements). Typically  $500 < h < 1000$
- $U$  :  $|V| * h$  matrix storing the *hidden-to-output* weights
- $H$  :  $(h * (n - 1)m)$  matrix storing the *projection-to-hidden* weights
- $\rightarrow \theta = (b, d, U, H, C)$

- Complexity per training sequence:  $n * m + n * m * h + h * |V|$   
computational bottleneck: **nonlinear hidden layer** ( $h * |V|$  term)



- **Training** is performed via stochastic gradient descent (learning rate  $\varepsilon$ ):

$$\theta \leftarrow \theta + \varepsilon \cdot \frac{\partial E}{\partial \theta} = \theta + \varepsilon \cdot \frac{\partial \log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})}{\partial \theta}$$

(weights are initialized randomly, then updated via backpropagation)

# NNLM facts

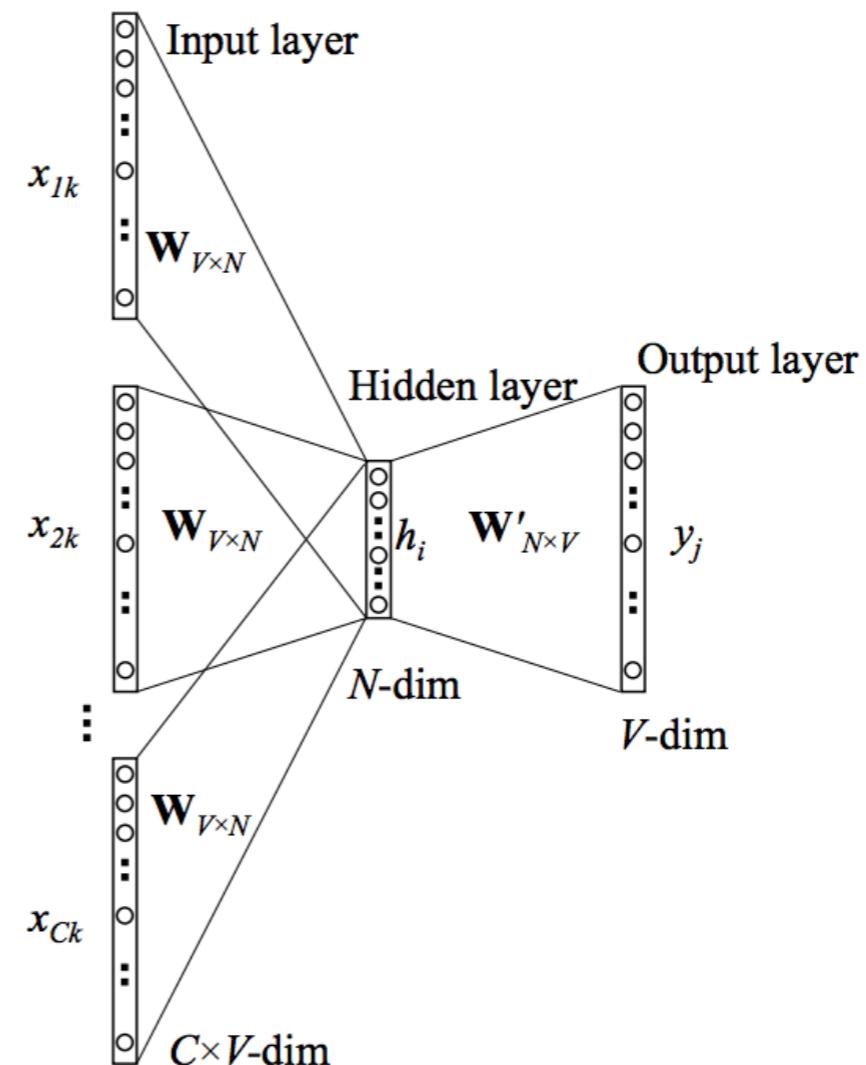
- tested on Brown (1.2M words,  $|V| \approx 16K$ ) and AP News (14M words,  $|V| \approx 150K$  reduced to 18K) corpuses
- Brown:  $h = 100$ ,  $n = 5$ ,  $m = 30$
- AP News:  $h = 60$ ,  $n = 6$ ,  $m = 100$ , **3 week** training using **40 cores**
- 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs
- in terms of test set *perplexity*: geometric average of  $1/\hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$
- Due to **complexity**, NNLM can't be applied to large data sets → poor performance on rare words
- Bengio et al. (2003) initially thought their main contribution was a more accurate LM. They let the interpretation and use of the word vectors as **future work**
- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

# Word2Vec

- Mikolov et al. in 2013
- Key idea of word2vec: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a **simpler (shallower) model** to be trained on **much larger amounts of data**
- no hidden layer (leads to 1000X speedup)
- projection layer is shared (not just the weight matrix) - C
- context: words from both history & future:
  - Two algorithms for learning words vectors:
    - **CBOW**: from context predict target
    - **Skip-gram**: from target predict context

# CBOW

- continuous bag-of-words
- continuous representations whose order is of no importance
- uses the surrounding words to predict the center word
- n-words before and after the target word



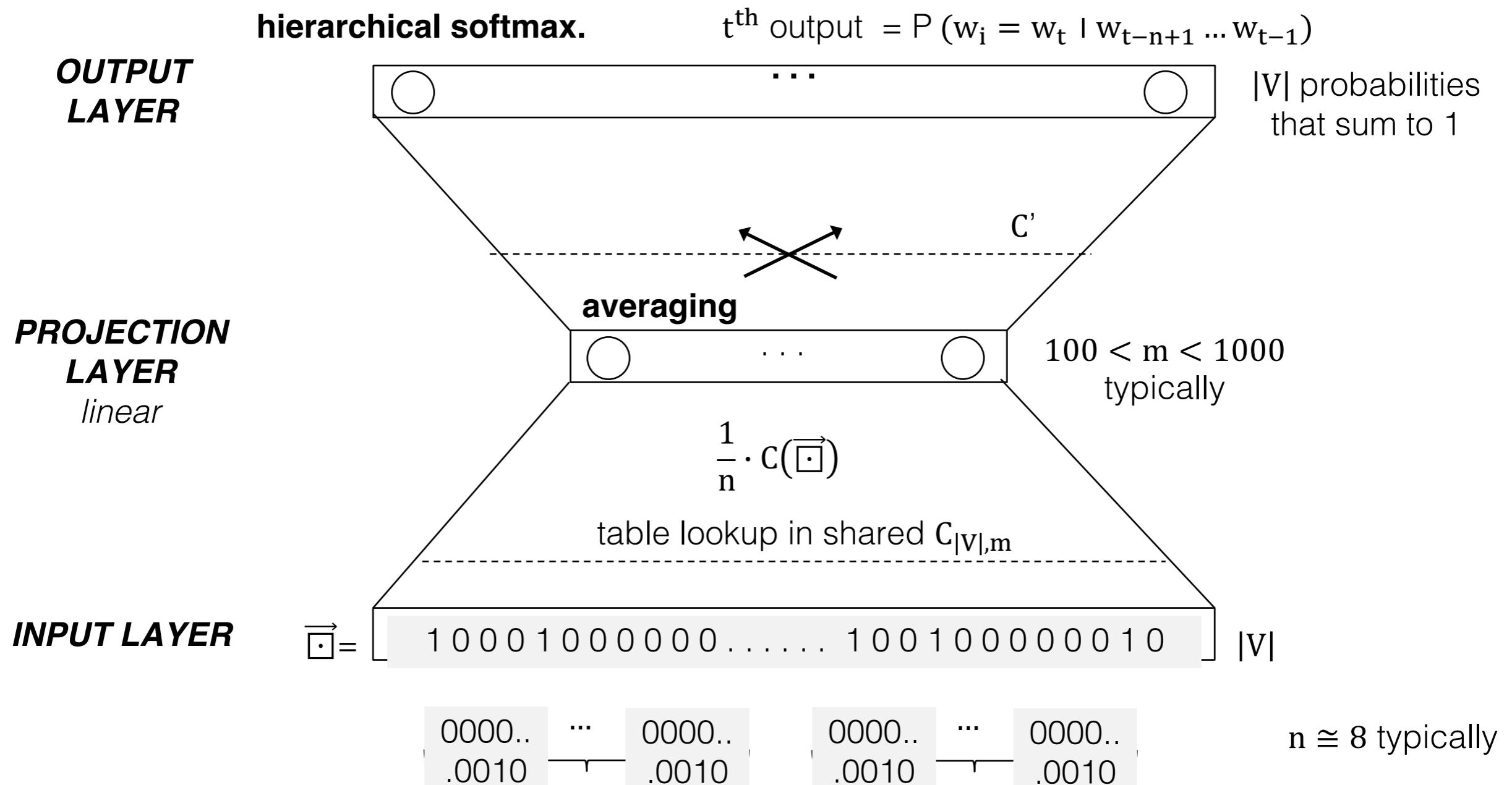
Efficient Estimation of Word Representations in Vector Space- Mikolov et al.

# Continuous Bag-of-Words (CBOW)

For each training sequence:

input = (context, target) pair:  $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

objective: minimize  $-\log \hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$



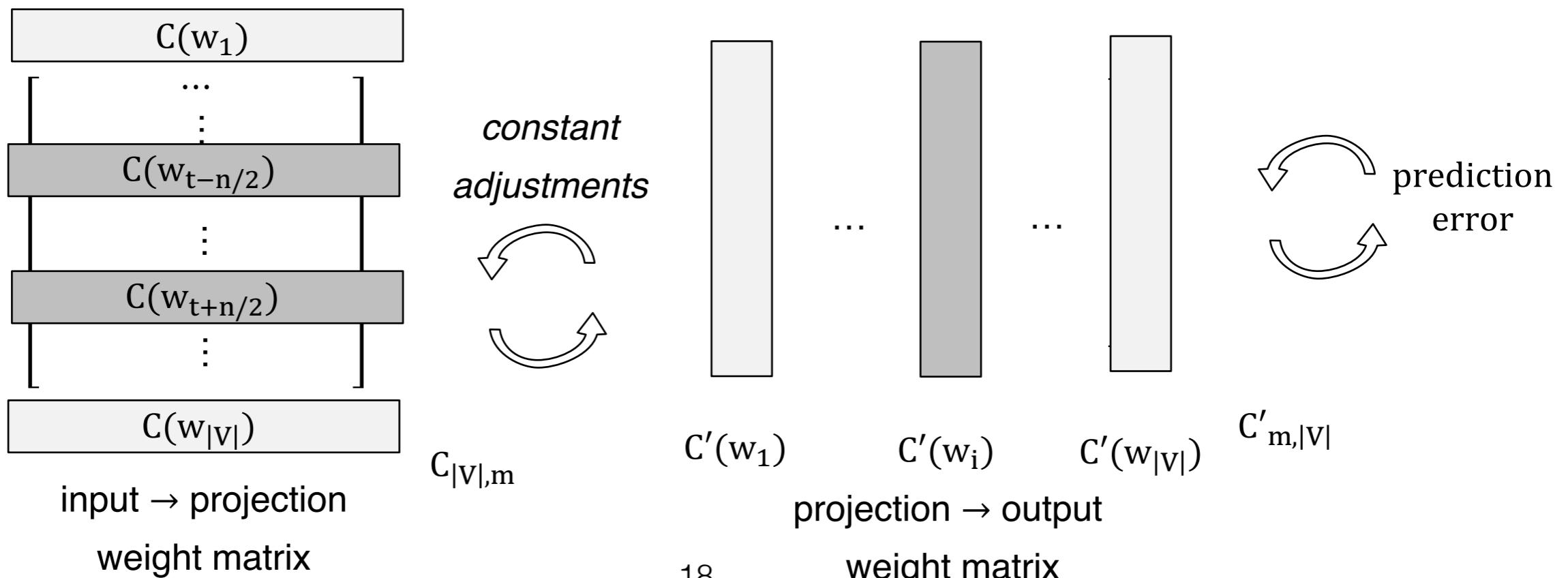
input context:  $n/2$  history words:  $w_{t-\frac{n}{2}} \dots w_{t-1}$

17

$n/2$  future words:  $w_{t+1} + \dots + w_{t+\frac{n}{2}}$

# Weight updating

- For each (context, target= $w_t$ ) pair, only the word vectors from matrix  $C$  corresponding to the context words are updated
- Recall that we compute  $P(w_i = w_t | \text{context}) \forall w_i \in V$ . We compare this distribution to the true probability distribution (1 for  $w_t$ , 0 elsewhere)
- **Back propagation**
- If  $P(w_i = w_t | \text{context})$  is **overestimated** (i.e.,  $> 0$ , happens in potentially  $|V| - 1$  cases), some portion of  $C'(w_i)$  is **subtracted** from the context word vectors in  $C$ , proportionally to the magnitude of the error
- Reversely, if  $P(w_i = w_t | \text{context})$  is **underestimated** ( $< 1$ , happens in potentially 1 case), some portion of  $C'(w_i)$  is **added** to the context word vectors in  $C$   
→ at each step the words move away or get closer to each other in the feature space → clustering



# Skip-gram

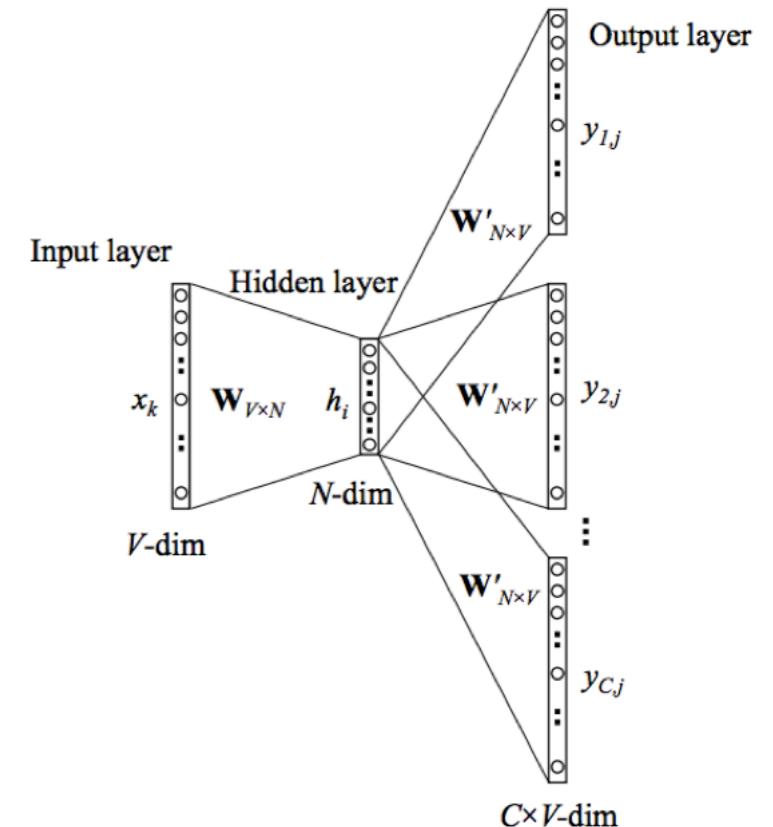
- skip-gram uses the center word to predict the surrounding words
- instead of computing the probability of the target word  $w_t$  given its previous words, we calculate the probability of the surrounding word  $w_{t+j}$  given  $w_t$

$$\text{➤ } p(w_{t+j} | w_t) = \frac{\exp(v_{w_t}^T v'_{w_{t+j}})}{\sum_{w \in V} \exp(v_{w_t}^T v'_{w_{t+j}})}$$

➤  $v_{w_t}^T$  is a column of  $\mathbf{W}_{VxN}$  and  $v'_{w_{t+j}}$  is a column of  $\mathbf{W}'_{NxV}$

➤ Objective function

$$J = \frac{1}{T} \sum_{t=1}^T \sum_{-n \leq j \leq n} \log p(w_{t+j} | w_t)$$



Efficient Estimation of Word Representations in  
Vector Space- Mikolov et al. 2013

# word2vec facts

- Complexity is  $n * m + m * \log|V|$  (Mikolov et al. 2013a)
- $n$ : size of the context window ( $\sim 10$ )  $n \times m$ : dimensions of the projection layer,  $|V|$  size of the vocabulary
- On Google news 6B words training corpus, with  $|V| \sim 10^6$ :
  - CBOW with  $m = 1000$  took **2 days** to train on **140 cores**
  - Skip-gram with  $m = 1000$  took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for  $m = 100$  only!  
(note that  $m = 1000$  was not reasonably feasible on such a large training set)
- word2vec training speed  $\cong 100K\text{-}5M$  words/s
- Quality of the word vectors:
  - $\nearrow$  significantly with **amount of training data** and **dimension of the word vectors** ( $m$ ), with diminishing relative improvements
  - measured in terms of accuracy on 20K semantic and syntactic association tasks.  
e.g., words in **bold** have to be returned:

Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: <b>Greece</b>	walking: <b>walked</b>	easy: <b>easiest</b>	brother: <b>sister</b>	ethical: <b>unethical</b>

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%
- References: <http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd>  
<https://code.google.com/p/word2vec/>

# GloVe

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

- Ratio of co-occurrence probabilities best distinguishes relevant words

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} \quad \rightarrow \quad w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

- Cast this into a least square problem:

- $X$  co-occurrence matrix
- $f$  weighting function,
- $b$  bias terms
- $w_i$  = word vector
- $\tilde{w}_j$  = context vector

$$J = \sum_{i,j=1}^V f(X_{ij}) \left( w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases} .$$

model that utilizes

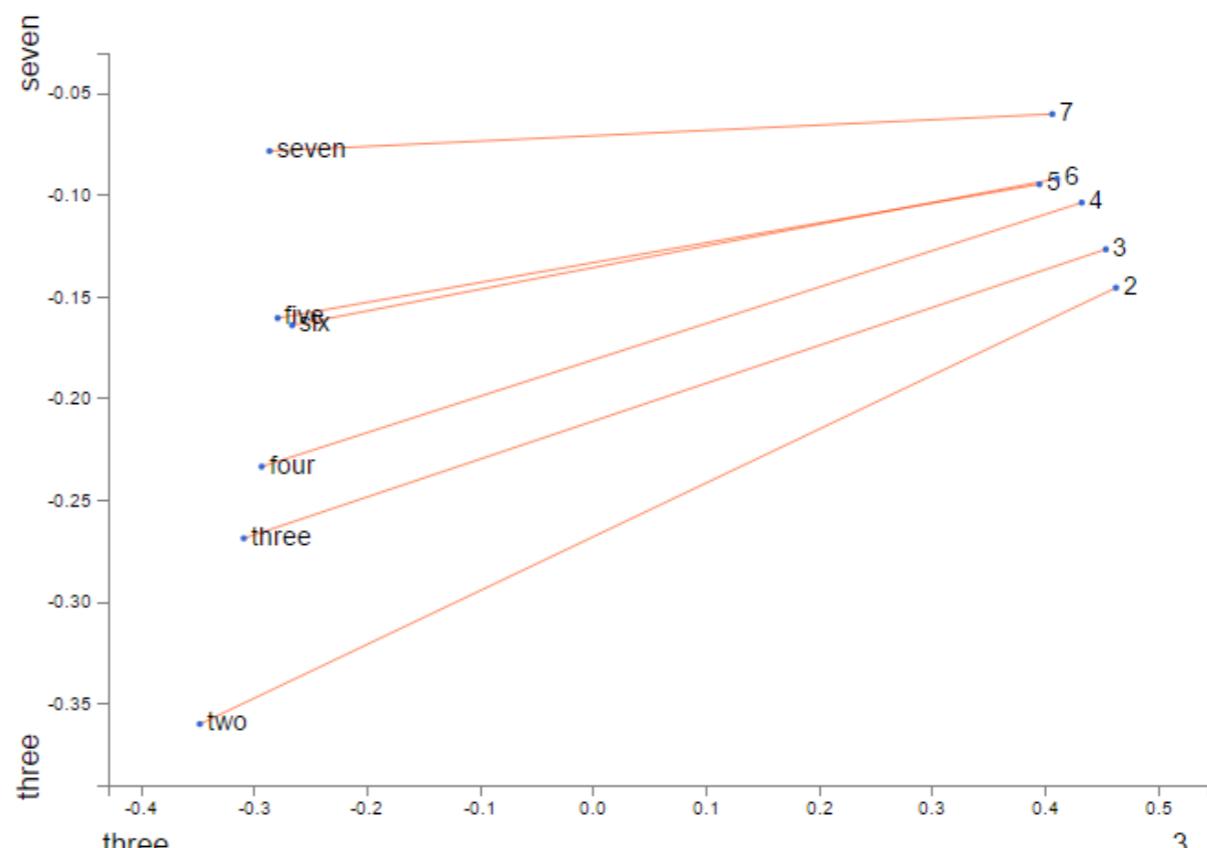
- count data
- bilinear prediction-based methods like word2vec

# Which is better?

- Open question
- SVD vs word2vec vs GloVe
- All based on co-occurrence
- Levy, O., Goldberg, Y., & Dagan, I. (2015)
  - SVD performs best on similarity tasks
  - Word2vec performs best on analogy tasks
  - *No single algorithm consistently outperforms the other methods*
  - *Hyperparameter tuning is important*
  - 3 out of 6 cases, tuning hyperparameters is more beneficial than increasing corpus size
  - word2vec outperforms GloVe on all tasks
  - *CBOW is worse than skip-gram on all tasks*

# Applications

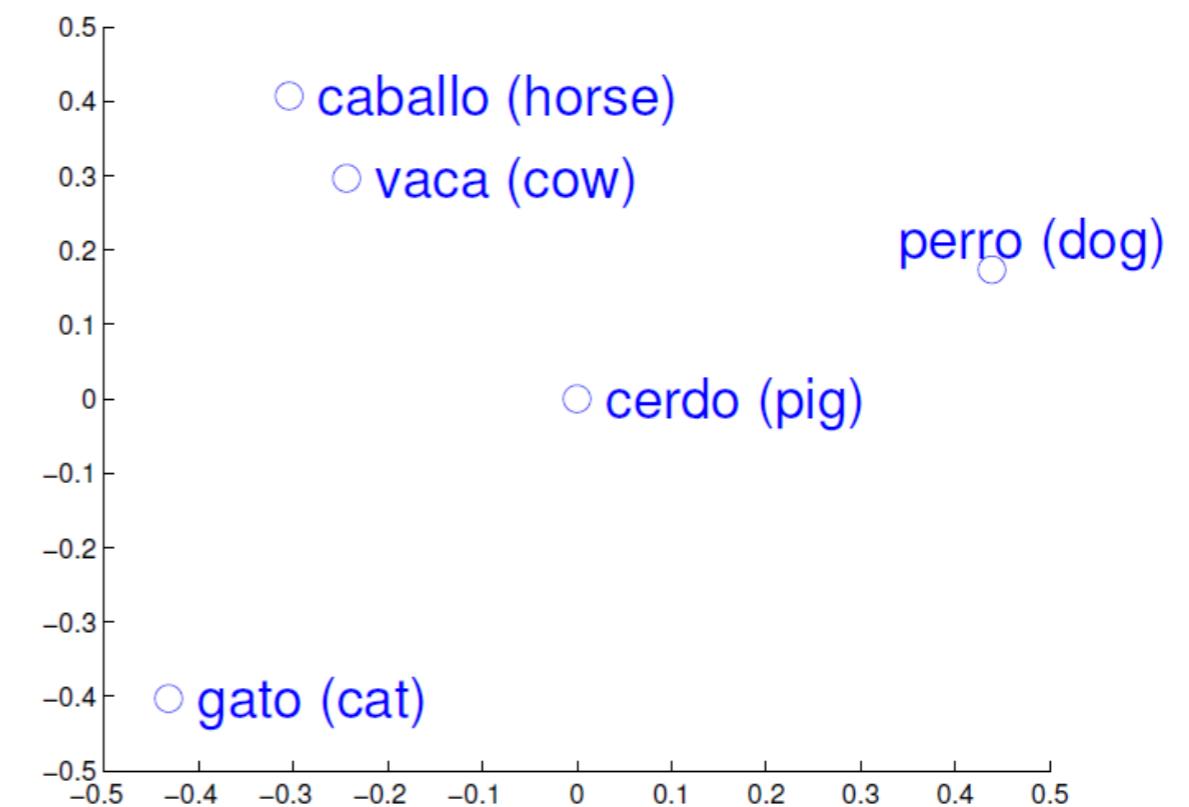
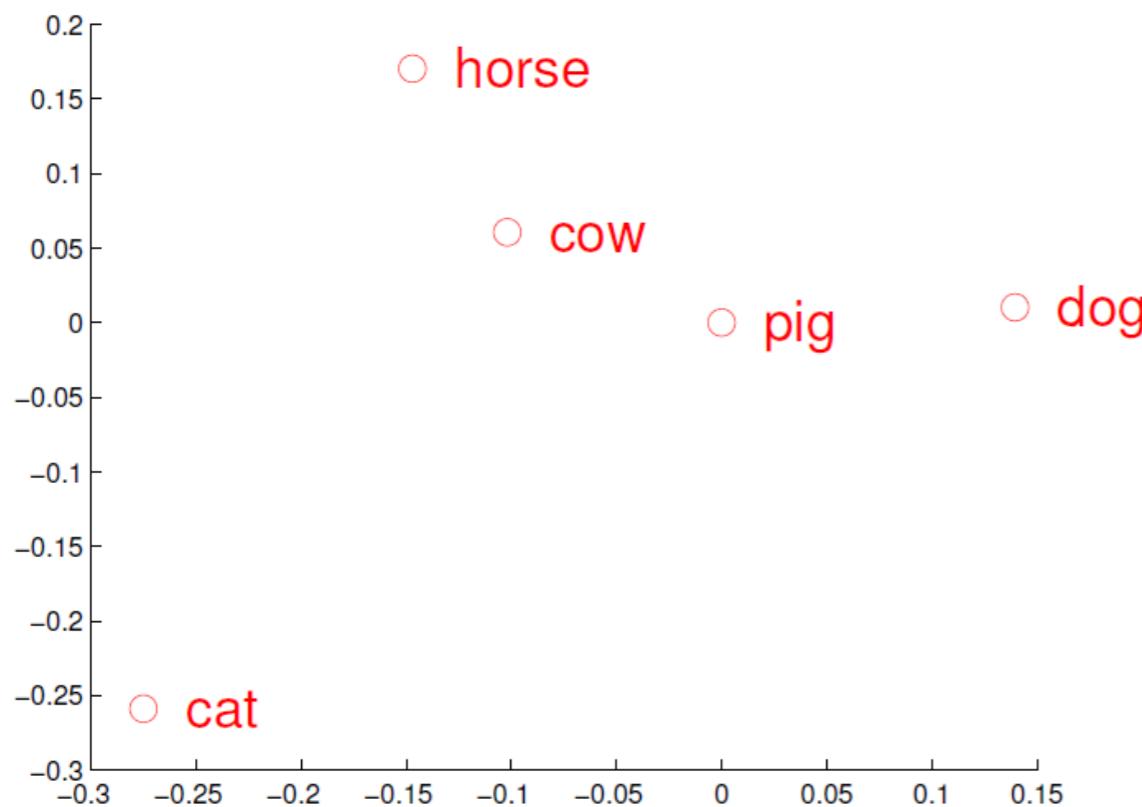
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification



<https://lamiowce.github.io/word2viz/>

# Applications

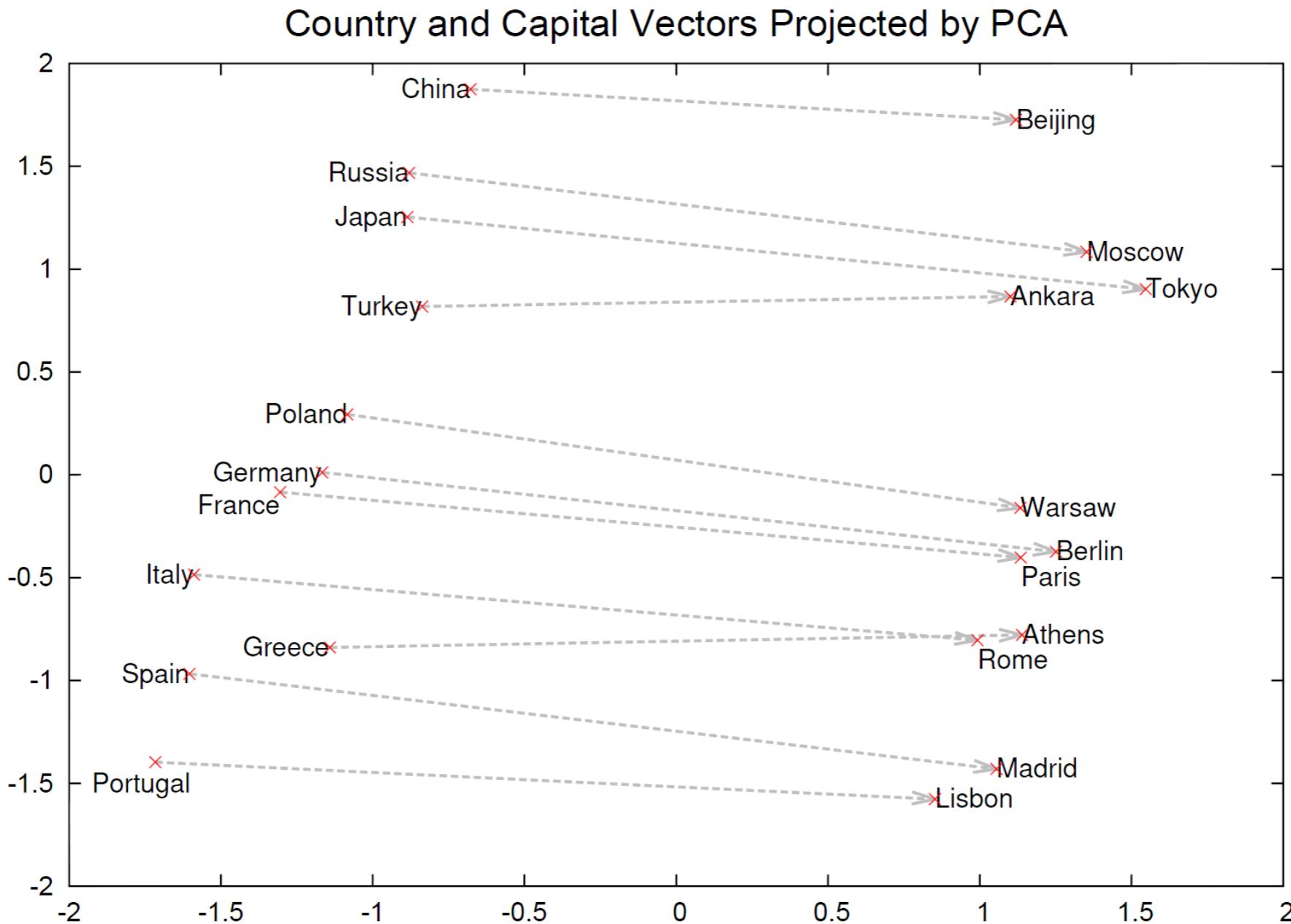
- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval...
- Example for English to Spanish machine translation:



About 90% reported accuracy (Mikolov et al. 2013c)

[Mikolov, T., Le, Q. V., & Sutskever, I. \(2013\). Exploiting similarities among languages for machine translation. \*arXiv preprint arXiv:1309.4168\*.](https://arxiv.org/abs/1309.4168)

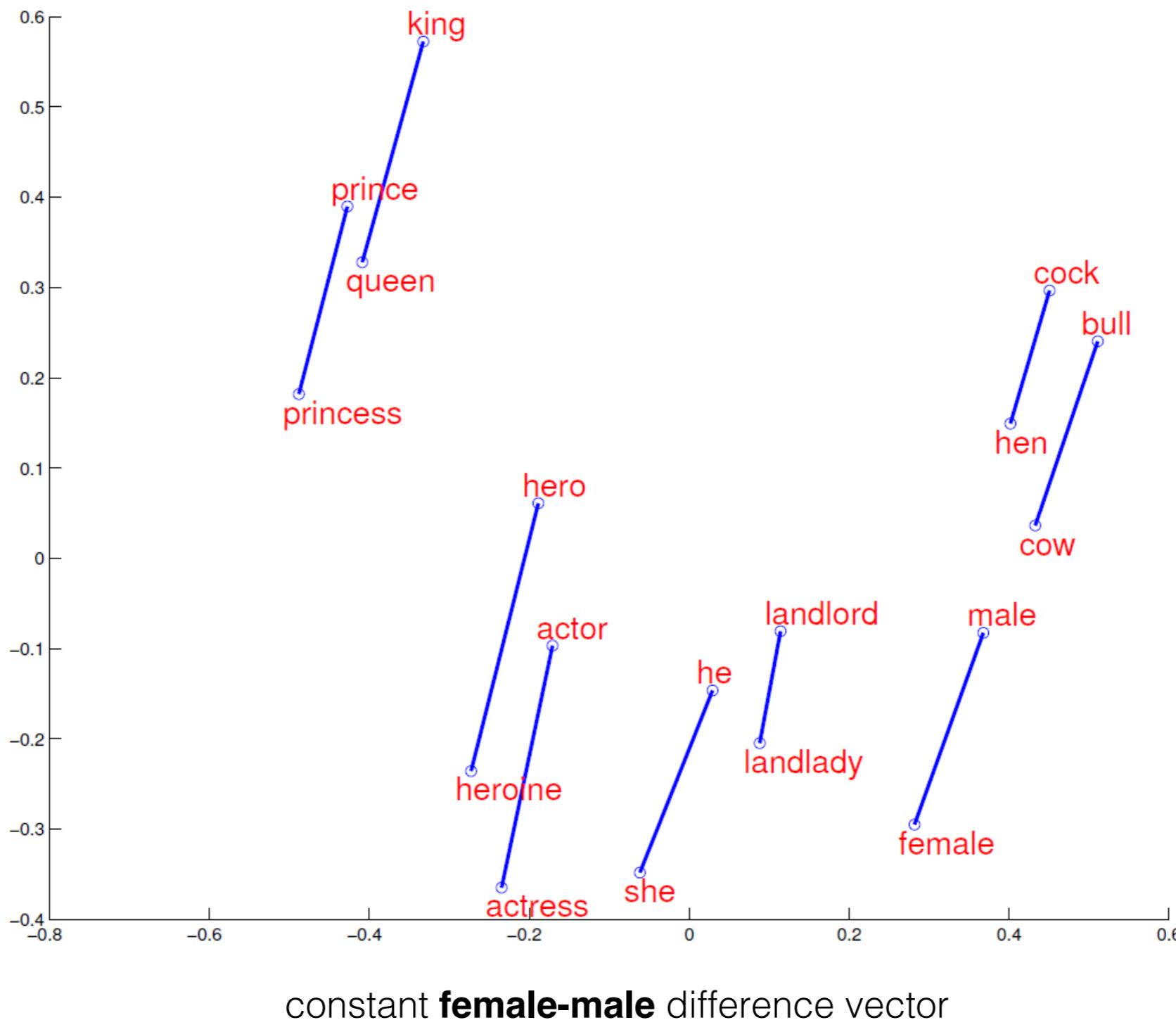
# Remarkable properties of word vectors



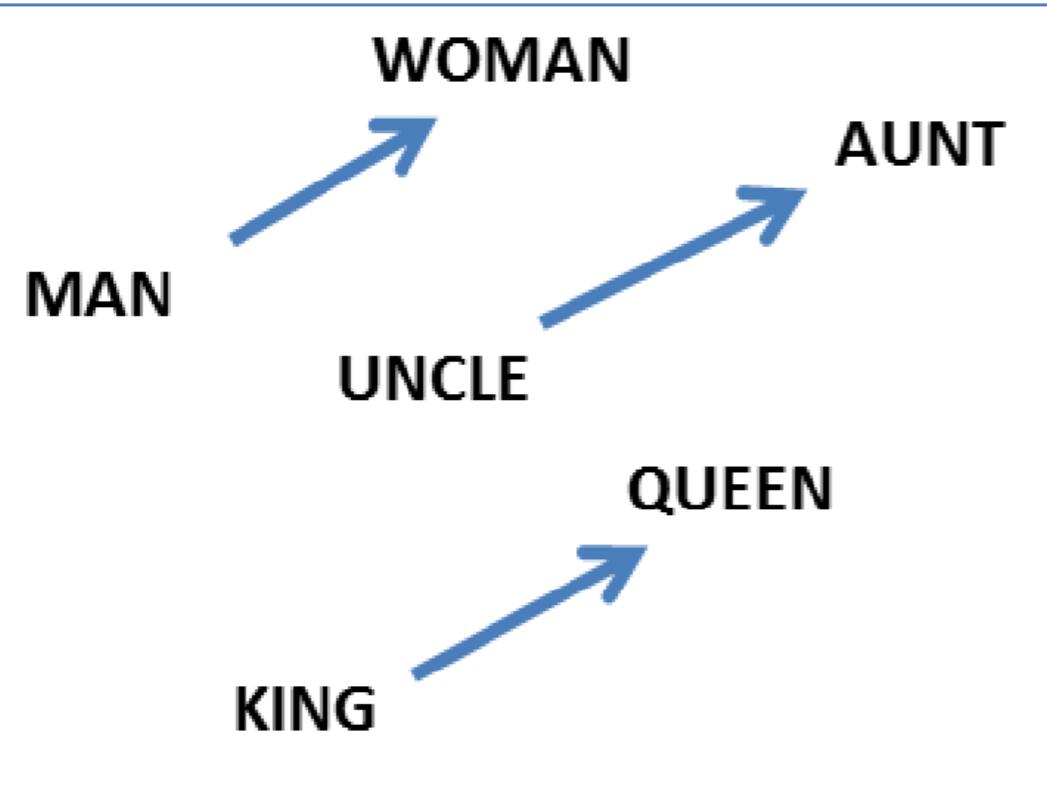
regularities between words are encoded in the difference vectors  
e.g., there is a constant **country-capital** difference vector

Mikolov et al. (2013b)  
Distributed representations of  
words and phrases and their  
compositionality

# Remarkable properties of word vectors



# Remarkable properties of word vectors



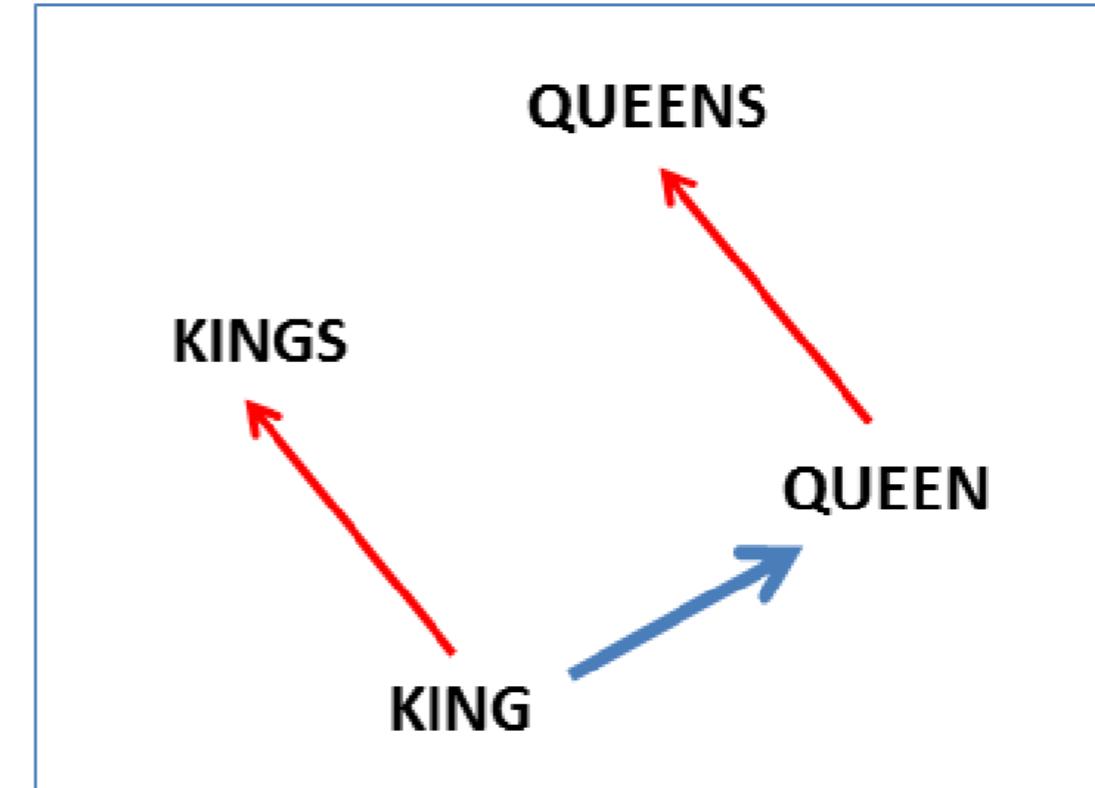
constant **male-female** difference vector

- Vector operations are supported and make intuitive sense:

$$w_{king} - w_{man} + w_{woman} \cong w_{queen}$$

$$w_{paris} - w_{france} + w_{italy} \cong w_{rome}$$

$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$



constant **singular-plural** difference vector

$$w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$$

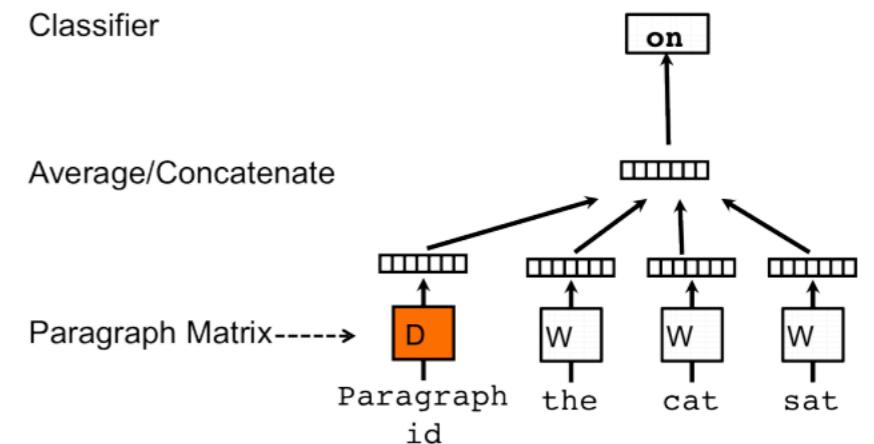
$$w_{his} - w_{he} + w_{she} \cong w_{her}$$

$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online [demo](#) (scroll down to end of tutorial)

# Distributed Representations of Sentences and Documents

- **Doc2vec**
- Paragraph or document vectors
- Capable of constructing representations of input sequences of variable length
- Represent each document by a dense vector
- Trained to predict words in the document
- paragraph vector and word vectors are averaged or concatenated to predict the next word in a context
- can be thought of as another word shared across all contexts in document

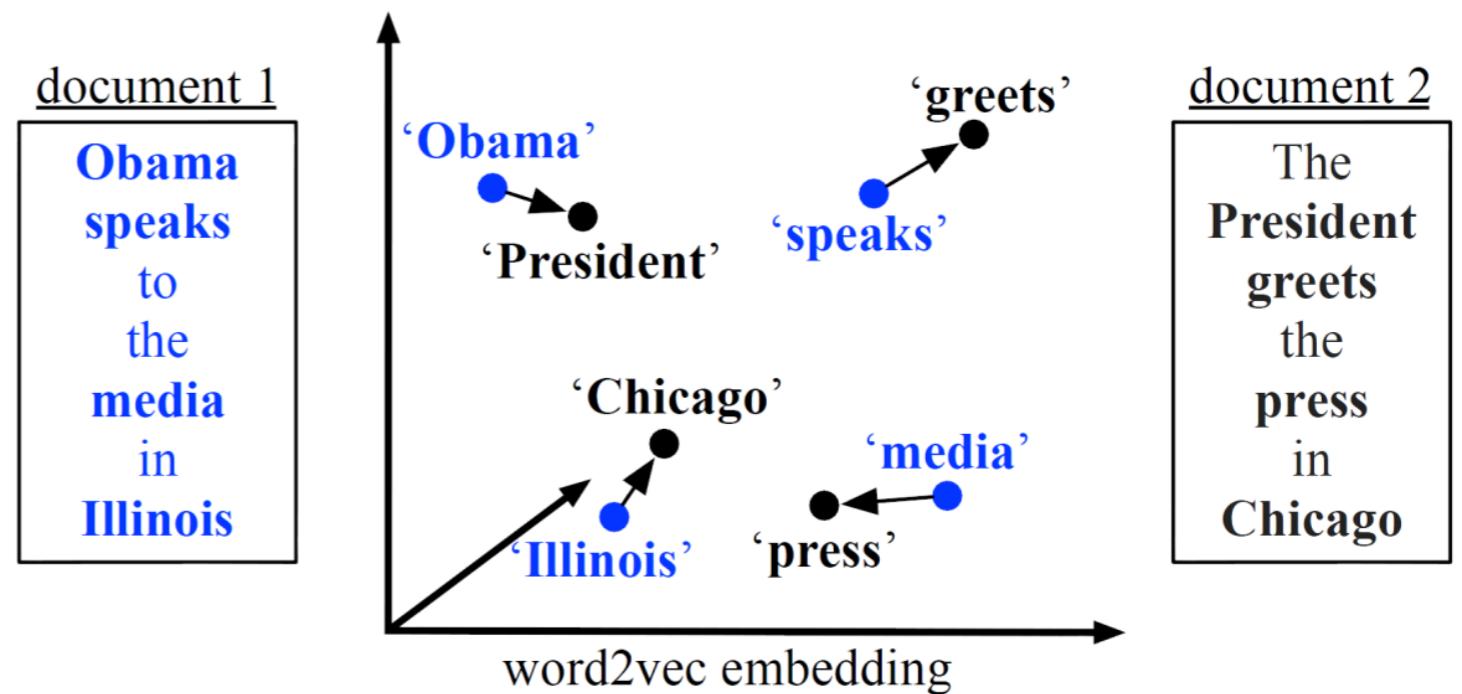


Model	Error rate (Positive/ Negative)	Error rate (Fine- grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 2013b)	17.6%	56.8%
Matrix Vector-RNN (Socher et al., 2013b)	17.1%	55.6%
Recursive Neural Tensor Network (Socher et al., 2013b)	14.6%	54.3%
Paragraph Vector	<b>12.2%</b>	<b>51.3%</b>

[https://cs.stanford.edu/~quocle/paragraph\\_vector.pdf](https://cs.stanford.edu/~quocle/paragraph_vector.pdf)

# Word Mover's distance

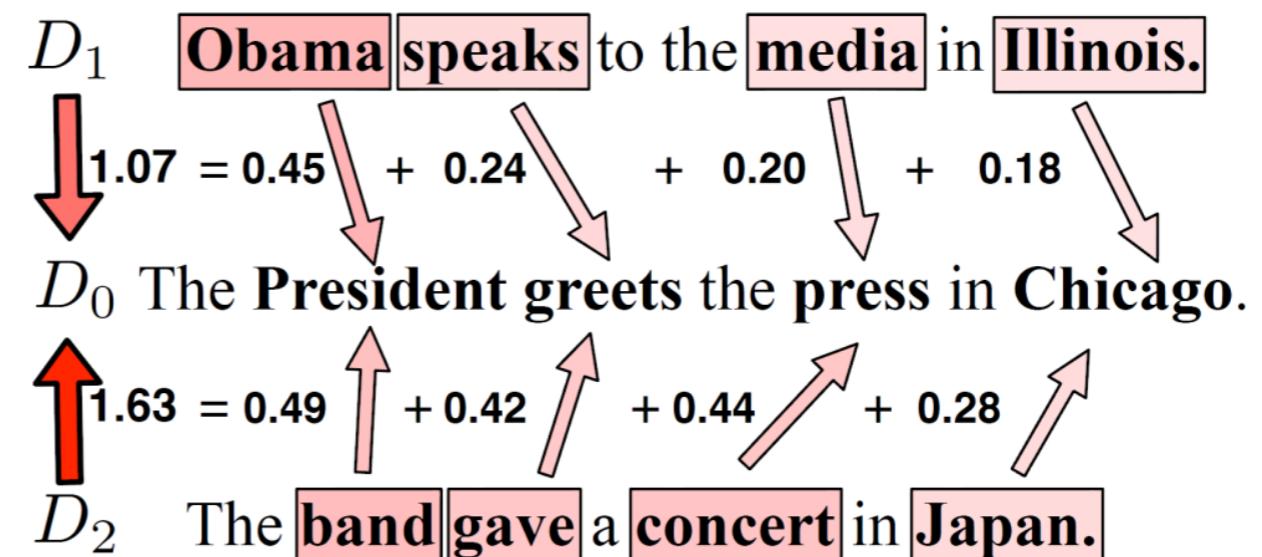
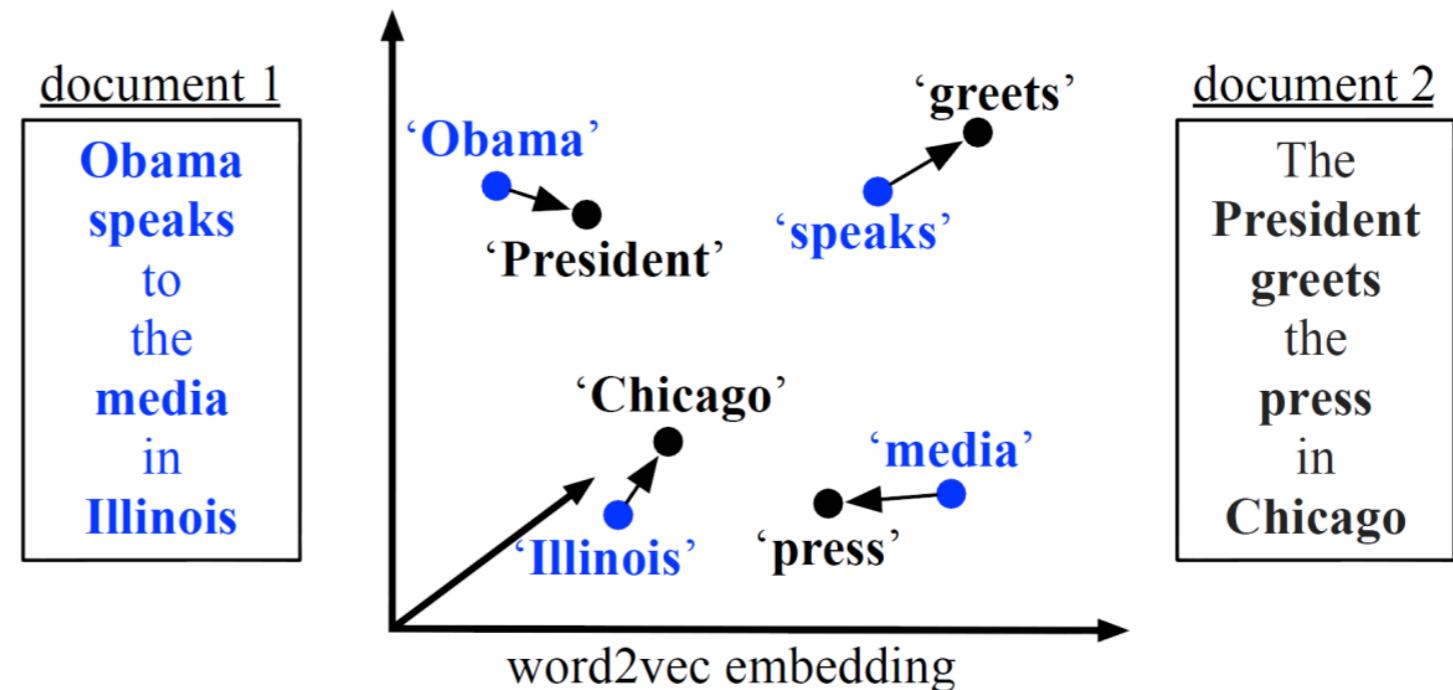
- “Edit” distance of 2 documents
- Based on word embedding representations
- Incorporate semantic similarity between individual word pairs into the document distance metric
- Based on “travel cost” between two words
- Calculates the cost of moving  $d$  to  $d'$
- hyper-parameter free
- highly interpretable
- high retrieval accuracy



“minimum cumulative distance that all words in document 1 need to travel to exactly match document 2”

# Word Mover's distance example

With the BOW representation  $D_1$  and  $D_2$  are at equal distance from  $D_0$ . Word embeddings allow to capture the fact that  $D_1$  is closer.



[Kusner, M. J., Sun, E. Y., Kolkin, E. N. I., & EDU, W. From Word Embeddings To Document Distances. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.](#)

# WMD

$d_i = \frac{c_i}{\sum_{j=1}^n c_j}$ . Normalized frequency of word  $i$

$c(i, j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2$  the word embeddings distance among words  $i, j$

- Assume documents  $\mathbf{d}, \mathbf{d}'$ .
- Assume each word  $i$  from  $\mathbf{d}$  can be transformed into any word  $j$  in  $\mathbf{d}'$
- $T_{ij} \geq 0$  denotes how much of word  $i$  in  $\mathbf{d}$  travels to word  $j$  in  $\mathbf{d}'$ .
- To transform  $\mathbf{d}$  entirely into  $\mathbf{d}'$  : entire outgoing flow from word  $i$  equals  $d_i$  .
- Transportation problem:

$$\min_{\mathbf{T} \geq 0} \sum_{i,j=1}^n \mathbf{T}_{ij} c(i, j) \quad \sum_j \mathbf{T}_{ij} = d_i.$$

$$\sum_i \mathbf{T}_{ij} = d'_j$$

- subject to:  $\sum_{j=1}^n \mathbf{T}_{ij} = d_i \quad \forall i \in \{1, \dots, n\}$
- $\sum_{i=1}^n \mathbf{T}_{ij} = d'_j \quad \forall j \in \{1, \dots, n\}.$

- Learn parameters  $\mathbf{T}_{ij}$  then the distance is:  $\sum_{i,j=1}^n \mathbf{T}_{ij} c(i, j)$

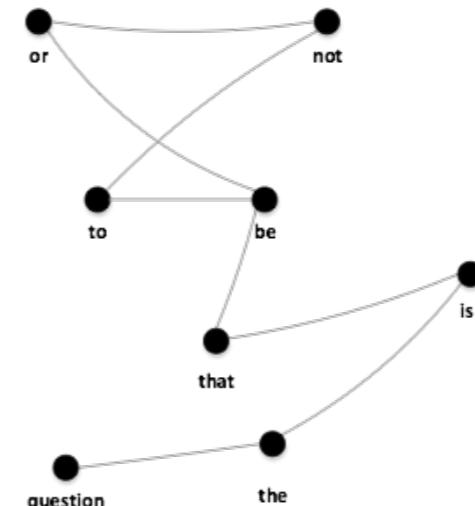
# Gaussian Document Representation from Word Embeddings

**EACL 2017**

# Document representation

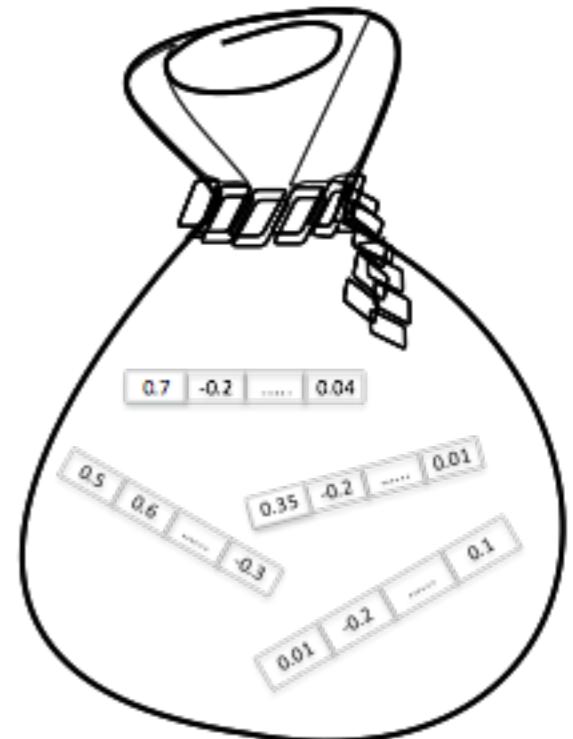
- Bag-of-Words
  - Document-term matrix
  - high dimensionality
  - sparse vectors
  - dimensions=|V|    |V|>10^6
  - unable to capture semantic similarity
- Graph-of-Words
  - captures co-occurrence
  - graph algorithms and techniques
  - graph kernels

	T1	T2	...Tn-1	Tn
D1	■	■	■	■
D2	■	■	■	■
D3	■	■	■	■



# Bag-of-Vectors

- Distributed representations of words
  - store contextual information in a low-dimensional vector
  - dimensionality reduction
  - dimensions=m       $100 < m < 500$
  - able to capture semantic similarity between words
  - many learning methods (word2vec, GloVe, SVD)
- Document is represented by a bag-of-vectors
- Goal: meaningful document representations and distance metrics based on the representations of their words



# Related work

- Centroid of vectors *[Lebret and Collobert, 2015]*
- Paragraph Vector *[Mikolov, 2014]*
  - vector representations for paragraphs by inserting an additional memory vector in the input layer.
- Word Mover's Distance *[Kusner, 2015]*
  - cumulative edit distance between two documents
- CNN for document classification *[Kim, 2014]*
  - Use the high quality embeddings as input for Convolutional Neural Network

# Gaussian Document Representation from Word Embeddings

- Assumption that words  $w$  present in a document are i.i.d. samples drawn from a multivariate Gaussian distribution
- Document is represented as a multivariate Gaussian distribution
  - mean vector  $\mu = \frac{1}{|d|} \sum_{w \in d} w$
  - covariance matrix  $\Sigma = \frac{1}{|d|} \sum_{w \in d} (w - \mu)(w - \mu)^T$
- Words contained in the vocabulary, but not contained in the embeddings model are randomly initialized

# Document Similarity

- centroid similarity

$$sim(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2) = \frac{\boldsymbol{\mu}_1 \cdot \boldsymbol{\mu}_2}{\|\boldsymbol{\mu}_1\| \|\boldsymbol{\mu}_2\|}$$

- covariance similarity

$$sim(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2) = \frac{\sum \boldsymbol{\Sigma}_1 \circ \boldsymbol{\Sigma}_2}{\|\boldsymbol{\Sigma}_1\|_F \times \|\boldsymbol{\Sigma}_2\|_F}$$

$(\cdot \circ \cdot)$  is the Hadamard  
or element-wise product

- similarity between two documents

$$sim(d_1, d_2) = \alpha(sim(\boldsymbol{\mu}_1, \boldsymbol{\mu}_2)) + (1 - \alpha)(sim(\boldsymbol{\Sigma}_1, \boldsymbol{\Sigma}_2)) \quad \alpha \in [0, 1]$$

- valid kernel function

# Why Gaussian?

- Each document follows a distribution described by its topic
- Word embeddings capture lexico-semantic regularities
- Words with similar syntactic and semantic properties are found to be close to each other in the embedding space
- Semantically related words are localized in space
- Gaussian distributions capture a notion of centrality in space
- Gaussian parameterization justified
  - analytic convenience
  - Euclidean distances between embeddings correlate with semantic similarity

# Experiments

## Datasets

Dataset	# training examples	# test examples	# classes	vocabulary size	<i>word2vec</i> size
Reuters	5,485	2,189	8	23,585	15,587
Amazon	8,000	CV	4	39,133	30,526
TREC	5,452	500	6	9,513	9,048
Snippets	10,060	2,280	8	29,276	17,067
BBCSport	348	389	5	14,340	13,390
Polarity	10,662	CV	2	18,777	16,416
Subjectivity	10,000	CV	2	21,335	17,896
Twitter	3,115	CV	3	6,266	4,460

## Baselines

- BOW-SVM
- NBSVM
- Centroid-SVM
- WMD-KNN
- CNN

# Results

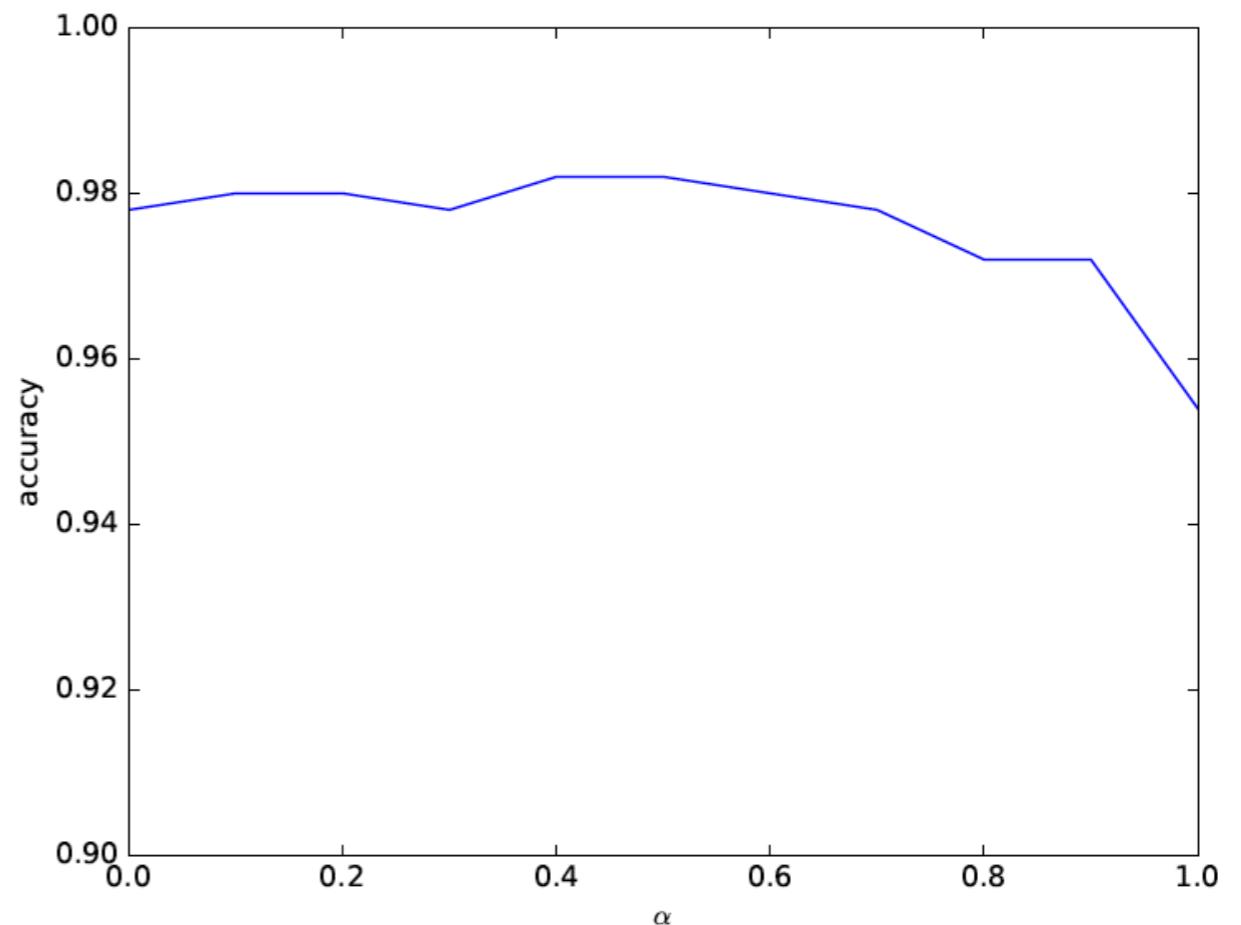
Dataset Method	Reuters		Amazon		TREC		Snippets	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
BOW (binary)	0.9571	0.8860	0.9126	0.9127	0.9660	0.9692	0.6171	0.5953
Centroid	0.9676	0.9171	0.9311	0.9312	0.9540	0.9586	0.8123	0.8170
WMD	0.9502	0.8204	0.9200	0.9201	0.9240	0.9336	0.7417	0.7388
NBSVM	<b>0.9712</b>	0.9155	0.9486	0.9486	0.9780	0.9805	0.6474	0.6357
CNN	0.9707	0.9297	0.9448	0.9449	0.9800	0.9800	<b>0.8478</b>	<b>0.8466</b>
Gaussian	<b>0.9712</b>	<b>0.9388</b>	<b>0.9498</b>	<b>0.9497</b>	<b>0.9820</b>	<b>0.9841</b>	0.8224	0.8244

Dataset Method	BBCSport		Polarity		Subjectivity		Twitter	
	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score	Accuracy	F1-score
BOW (binary)	0.9640	0.9690	0.7615	0.7614	0.9004	0.9004	0.7467	0.6205
Centroid	0.9923	0.9915	0.7783	0.7782	0.9100	0.9100	0.7361	0.5727
WMD	0.9871	0.9866	0.6642	0.6639	0.8604	0.8603	0.7031	0.4436
NBSVM	0.9871	0.9892	<b>0.8698</b>	<b>0.8698</b>	<b>0.9369</b>	<b>0.9368</b>	<b>0.7852</b>	0.6191
CNN	0.9486	0.9461	0.8037	0.8031	0.9315	0.9314	0.7549	0.6137
Gaussian	<b>0.9974</b>	<b>0.9974</b>	0.8021	0.8020	0.9310	0.9310	0.7534	<b>0.6443</b>

# Results

- Parameter a sensitivity
- TREC dataset
- Centroid performance drops significantly
- Highest accuracy a=0.5



# Conclusion

- Model each document as a Gaussian distribution based on the embeddings of its words
- Similarity between two documents based on the similarity of their distributions
- Empirical evaluation demonstrates the effectiveness of the approach across a range of data
- Performance gain is attributed to the high quality of the embeddings and the ability to effectively utilize them

# References

- Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A Neural Probabilistic Language Model. *The Journal of Machine Learning Research*, 3, 1137–1155. <http://doi.org/10.1162/153244303322533223>
- Mikolov, T., Corrado, G., Chen, K., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space. *Proceedings of the International Conference on Learning Representations (ICLR 2013)*, 1–12.
- Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. *NIPS*, 1–9.
- Collobert, R., & Weston, J. (2008). A unified architecture for natural language processing. *Proceedings of the 25th International Conference on Machine Learning - ICML '08*, 20(1), 160–167. <http://doi.org/10.1145/1390156.1390177>
- Kim, Y., Jernite, Y., Sontag, D., & Rush, A. M. (2016). Character-Aware Neural Language Models. *AAAI*. Retrieved from <http://arxiv.org/abs/1508.06615>
- Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the Limits of Language Modeling. Retrieved from <http://arxiv.org/abs/1602.02410>
- Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural Language Processing (almost) from Scratch. *Journal of Machine Learning Research*, 12 (Aug), 2493–2537. Retrieved from <http://arxiv.org/abs/1103.0398>
- Chen, W., Grangier, D., & Auli, M. (2015). Strategies for Training Large Vocabulary Neural Language Models, 12. Retrieved from <http://arxiv.org/abs/1512.04906>

# More References

- Levy, O., Goldberg, Y., & Dagan, I. (2015). Improving Distributional Similarity with Lessons Learned from Word Embeddings. *Transactions of the Association for Computational Linguistics*, 3, 211–225. Retrieved from <https://tacl2013.cs.columbia.edu/ojs/index.php/tacl/article/view/570>
- Pennington, J., Socher, R., & Manning, C. D. (2014). Glove: Global Vectors for Word Representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, 1532–1543. <http://doi.org/10.3115/v1/D14-1162>
- Baroni, M., Dinu, G., & Kruszewski, G. (2014). Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. *ACL*, 238–247. <http://doi.org/10.3115/v1/P14-1023>
- Levy, O., & Goldberg, Y. (2014). Neural Word Embedding as Implicit Matrix Factorization. *Advances in Neural Information Processing Systems (NIPS)*, 2177–2185. Retrieved from <http://papers.nips.cc/paper/5477-neural-word-embedding-as-implicit-matrix-factorization>
- Hamilton, W. L., Clark, K., Leskovec, J., & Jurafsky, D. (2016). Inducing Domain-Specific Sentiment Lexicons from Unlabeled Corpora. *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. Retrieved from <http://arxiv.org/abs/1606.02820>
- Hamilton, W. L., Leskovec, J., & Jurafsky, D. (2016). Diachronic Word Embeddings Reveal Statistical Laws of Semantic Change. *arXiv Preprint arXiv:1605.09096*.

# References- blogs

- Sebastian Ruder blog series on Word Embeddings, <http://sebastianruder.com/>
- Andy Jones blog on word2vec, <http://andyljones.tumblr.com/post/111299309808/why-word2vec-works>
- Arora et al, <https://arxiv.org/pdf/1502.03520v7.pdf>
- Piotr Migdał , <http://p.migdal.pl/2017/01/06/king-man-woman-queen-why.html>