Representation Learning for Text and Applications

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

M. Vazirgiannis

Ecole Polytechnique & AUEB

Scholar: https://tinyurl.com/y7ulzoqt

November 2019

Language model

- Goal: determine $\mathrm{P}(s = \mathrm{w}_1 \dots \mathrm{w}_\mathrm{k})$ in some domain of interest

$$\mathrm{P}(\mathrm{s}) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_1 \dots \mathrm{w}_{\mathrm{i}-1}\right)$$

e.g., $\mathrm{P}\left(\mathrm{w}_1 \mathrm{w}_2 \mathrm{w}_3\right) = \mathrm{P}\left(\mathrm{w}_1\right) \mathrm{P}\left(\mathrm{w}_2 \mid \mathrm{w}_1\right) \mathrm{P}\left(\mathrm{w}_3 \mid \mathrm{w}_1 \mathrm{w}_2\right)$

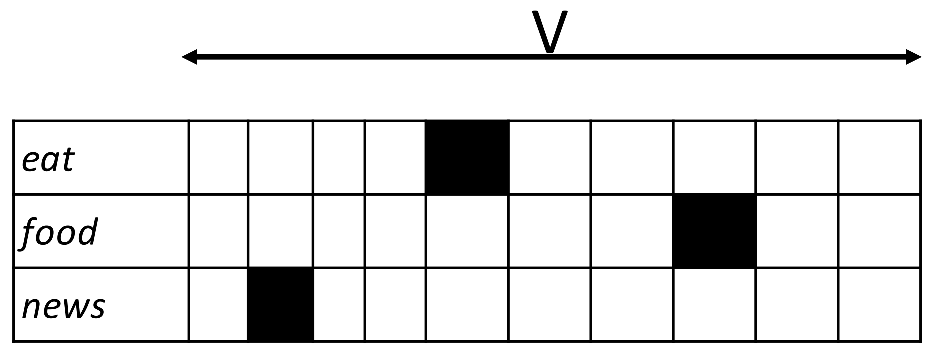
- Traditional n-gram language model assumption:
  "the probability of a word depends only on **context** of $\mathrm{n}-1$ previous words"

$$\Rightarrow \widehat{\mathrm{P}}(\mathrm{s}) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}\right)$$

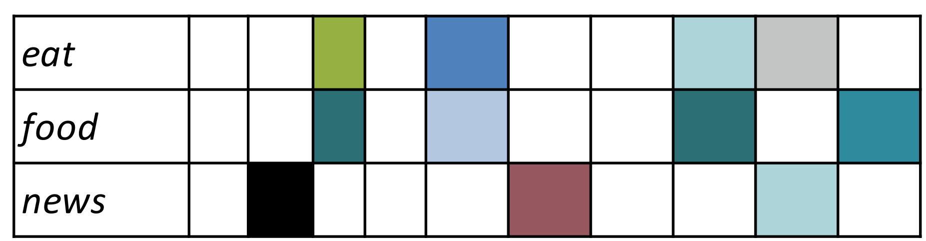
- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\widehat{\mathrm{P}}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}\right) = \frac{\# \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1} \mathrm{w}_\mathrm{i}}{\# \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \dots \mathrm{w}_{\mathrm{i}-1}}$ 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

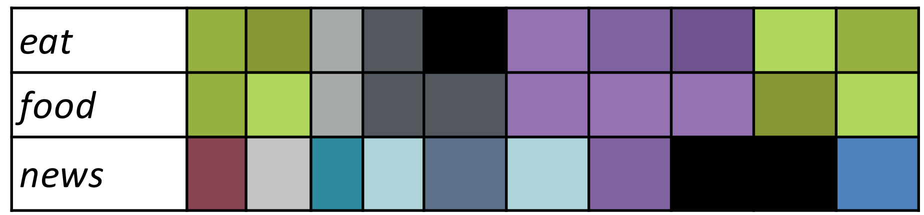


- **Distributional vector**
  - words that occur in similar contexts, tend to have similar meanings
  - each word vector contains the frequencies of all its neighbors
  - dimensions=|V|
  - computational complexity for ML algorithms



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

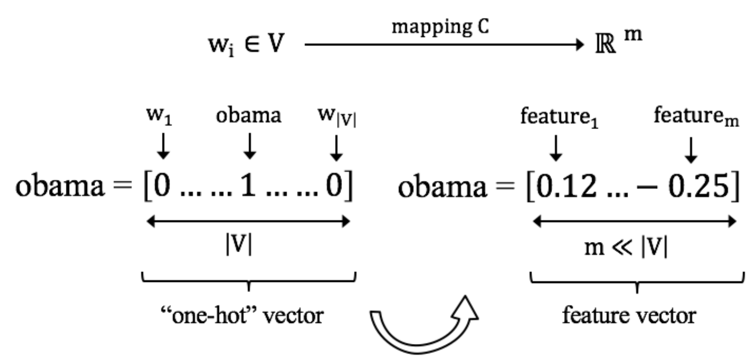
One hot

$$\left.\begin{array}{r} \text{obama} = [0\,0\,0\,0\ \ldots\ 0\,1\,0\,0] \\ \text{president} = [0\,0\,0\,1\ \ldots\ 0\,0\,0\,0] \end{array}\right\} \overrightarrow{\text{obama}}.\overrightarrow{\text{president}} = \vec{0}$$

$$\left.\begin{array}{r} \text{speaks} = [0\,0\,1\,0\ \ldots\ 0\,0\,0\,0] \\ \text{addresses} = [0\,0\,0\,0\ \ldots\ 0\,0\,1\,0] \end{array}\right\} \overrightarrow{\text{speaks}}.\overrightarrow{\text{addresses}} = \vec{0}$$

$$\left.\begin{array}{r} \text{illinois} = [1\,0\,0\,0\ \ldots\ 0\,0\,0\,0] \\ \text{chicago} = [0\,1\,0\,0\ \ldots\ 0\,0\,0\,0] \end{array}\right\} \overrightarrow{\text{illinois}}.\overrightarrow{\text{chicago}} = \vec{0}$$

Word embeddings



SVD word embeddings

- Dimensionality reduction on co-occurrence matrix
- Create a |V|x|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

SVD application - Latent Structure in documents

- Documents are represented based on the Vector Space Model
- Vector space model consists of the keywords contained in a document.
- In many cases baseline keyword based performs poorly – not able to detect synonyms.
- Therefore document clustering is problematic
- Example where of keyword matching with the query: “IDF in computer-based information look-up”

	access	document	retrieval	information	theory	database	indexing	computer
Doc1	x	x	x			x	x	
Doc2				x	x			x
Doc3			x	x				x

Indexing by Latent Semantic Analysis (1990) Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, Journal of the American Society of Information Science

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:

$A_k=U_kL_kV_k^T$,

- L_k square kxk - top-k singular values of the diagonal in matrix L,
- $U_{k,}$ mxk matrix - first k columns in U (left singular vectors)
- $V_k^{T,}$ kxn matrix - first k lines of V^T (right singular vectors)

Typical values for κ~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 Ak=UkLkVt
- - 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 $\quad q_n = q^T U_k L_k^{-1}$
- - 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.

Indexing by Latent Semantic Analysis (1990) Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, Richard Harshman, Journal of the American Society of Information Science

LSI – an example

	C1	C2	C3	C4	C5	M1	M2	M3	M4
human	1	0	0	1	0	0	0	0	0
Interface	1	0	1	0	0	0	0	0	0
computer	1	1	0	0	0	0	0	0	0
User	0	1	1	0	1	0	0	0	0
System	0	1	1	2	0	0	0	0	0
Response	0	1	0	0	1	0	0	0	0
Time	0	1	0	0	1	0	0	0	0
EPS	0	0	1	1	0	0	0	0	0
Survey	0	1	0	0	0	0	0	0	1
Trees	0	0	0	0	0	1	1	1	0
Graph	0	0	0	0	0	0	1	1	1
Minors	0	0	0	0	0	0	0	1	1

LSI – an example

$A=ULV^T$

A=

1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
0	1	1	0	1	0	0	0	0
0	1	1	2	0	0	0	0	0
0	1	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0
0	0	1	1	0	0	0	0	0
0	1	0	0	0	0	0	0	1
0	0	0	0	0	1	1	1	0
0	0	0	0	0	0	1	1	1
0	0	0	0	0	0	0	1	1

LSI – an example

$A = ULV^T$

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

$A=ULV^T$

L=

3.34	0	0	0	0	0	0	0	0
0	2.54	0	0	0	0	0	0	0
0	0	2.35	0	0	0	0	0	0
0	0	0	1.64	0	0	0	0	0
0	0	0	0	1.50	0	0	0	0
0	0	0	0	0	1.31	0	0	0
0	0	0	0	0	0	0.85	0	0
0	0	0	0	0	0	0	0.56	0
0	0	0	0	0	0	0	0	0.36
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

LSI – an example

$A=ULV^T$

$V=$

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

$U_k=$

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

$L_k=$

3.34	0
0	2.54

$V_k^T=$

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

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=

1
0
1
0
0
0
0
0
0
0
0
0

Query transformation

$q^T=$

1	0	1	0	0	0	0	0	0	0	0	0

$U_k=$

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

$L_k=$

0.334	0
0	0.254

$q_n=q^TU_kL_k=$

0.138	-0.0273

Query transformation

Map docs to the 2 dim space $V_kL_k=$

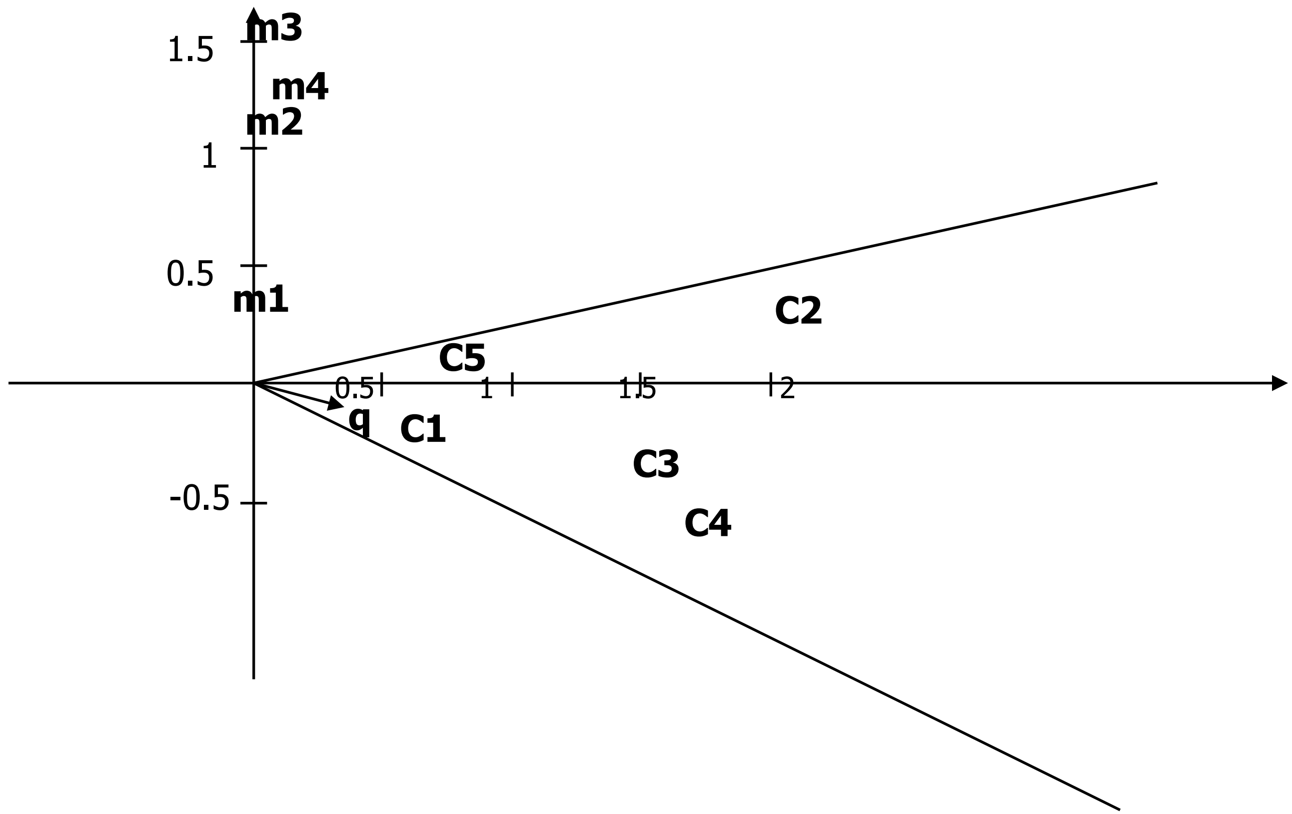
$$\begin{bmatrix} 0.20 & -0.06 \\ 0.61 & 0.17 \\ 0.46 & -0.13 \\ 0.54 & -0.23 \\ 0.28 & 0.11 \\ 0.00 & 0.19 \\ 0.01 & 0.44 \\ 0.02 & 0.62 \\ 0.08 & 0.53 \end{bmatrix} \begin{bmatrix} 3.34 & 0 \\ 0 & 2.54 \end{bmatrix} = \begin{bmatrix} 0.67 & -0.15 \\ 2.04 & 0.43 \\ 1.54 & -0.33 \\ 1.80 & -0.58 \\ 0.94 & 0.28 \\ 0.00 & 0.48 \\ 0.03 & 1.12 \\ 0.07 & 1.57 \\ 0.27 & 1.35 \end{bmatrix}$$

$$q_nL_k = \begin{bmatrix} 0.138 & -0.0273 \end{bmatrix} \begin{bmatrix} 3.34 & 0 \\ 0 & 2.54 \end{bmatrix} = \begin{bmatrix} 0.46 & -0.069 \end{bmatrix}$$

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



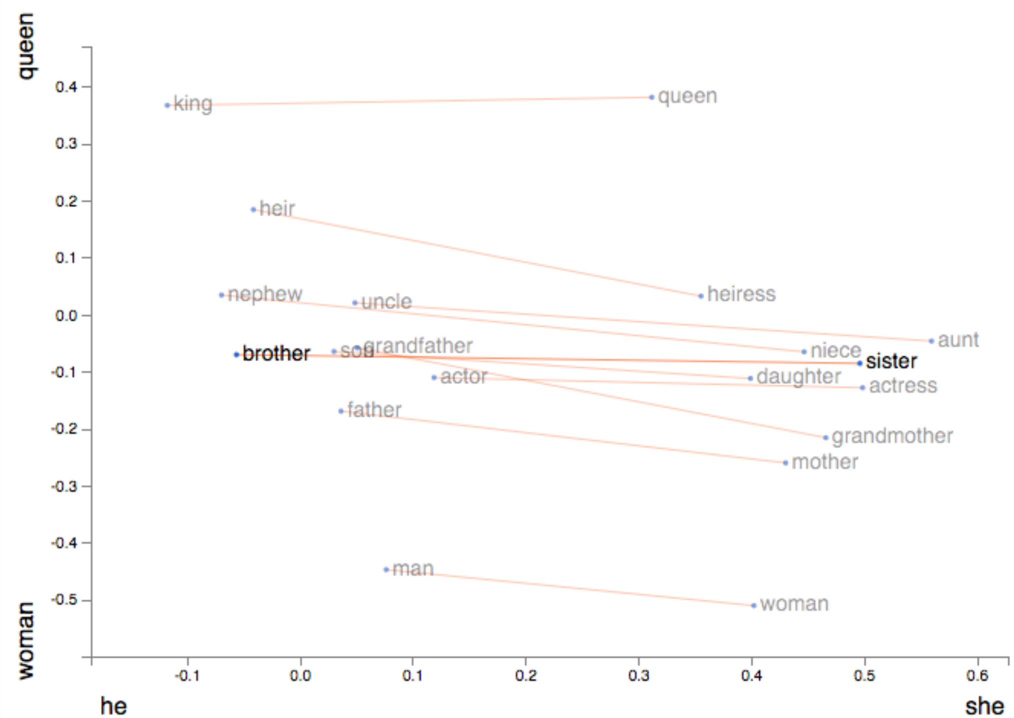
SVD problems

- 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. $PMI(w', w) \approx v_w v_{w'}$ *inner product*
2. Isotropic: $E_{w'}[(v_{w'} v_u)]^2 = ||v_u||^2$

Then

3. $argmin_w \, E_{w'}[ln \frac{p(w'|w)}{p(w'|queen)} - ln \frac{p(w'|man)}{p(w'|woman)}]^2$
4. $argmin_w \, E_{w'}[\big(PMI(w'|w) - PMI(w'|queen)\big) - (PMI(w'|man) - PMI(w'|woman))]^2$
5. $argmin_w ||\big(v_w - v_{queen}\big) - (v_{man} - v_{woman})||^2$
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

A Latent Variable Model Approach to PMI-based Word Embeddings, ***Arora et al (ACL 2016)***

Learning Word Vectors

- Corpus containing a sequence of T training words
- Objective: $f(w_t, \ldots, w_{t-n+1}) = \hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$
- Decomposed in two parts:

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

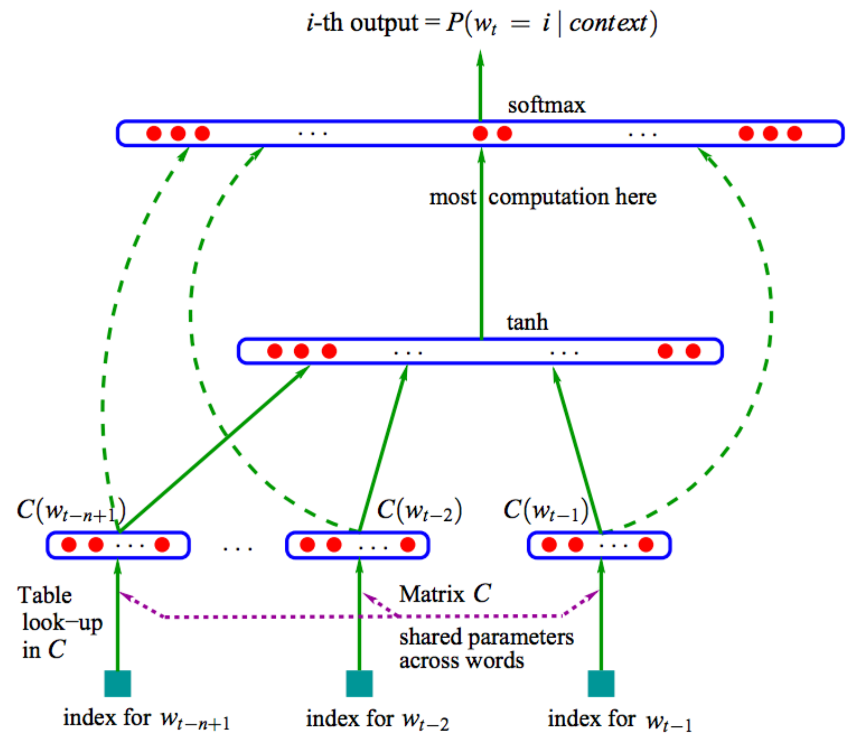
  - Mapping **C** (1-hotv => lower dimensions)
  - Mapping any **g** s.t. (estimate prob t+1 | t previous)

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

- C(i) is the i-th word feature vector

  (Word embedding)

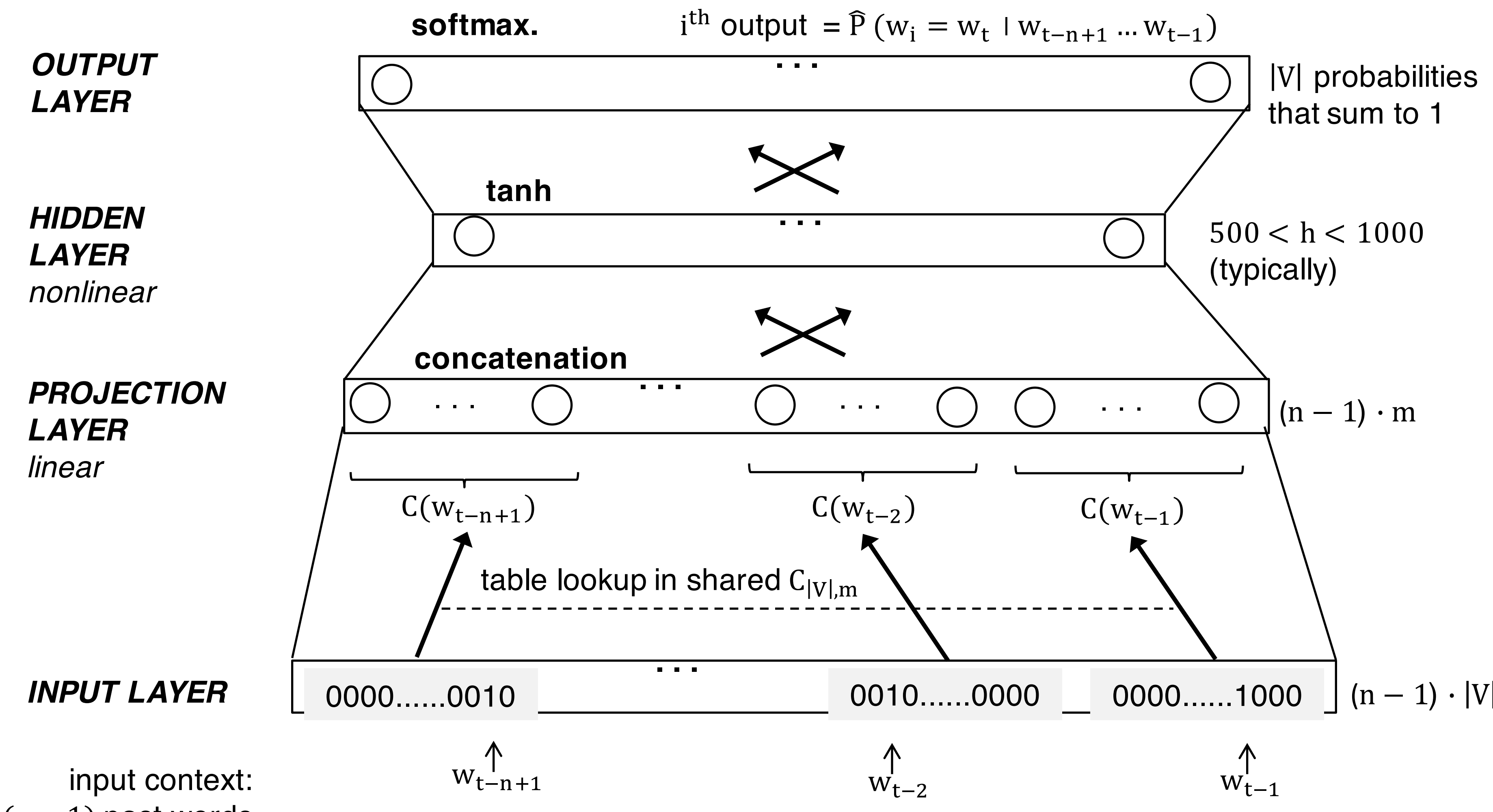
- Objective function: $J = \frac{1}{T} \sum f(w_t, \ldots, w_{t-n+1})$



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 $E = -\log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$



Objective function

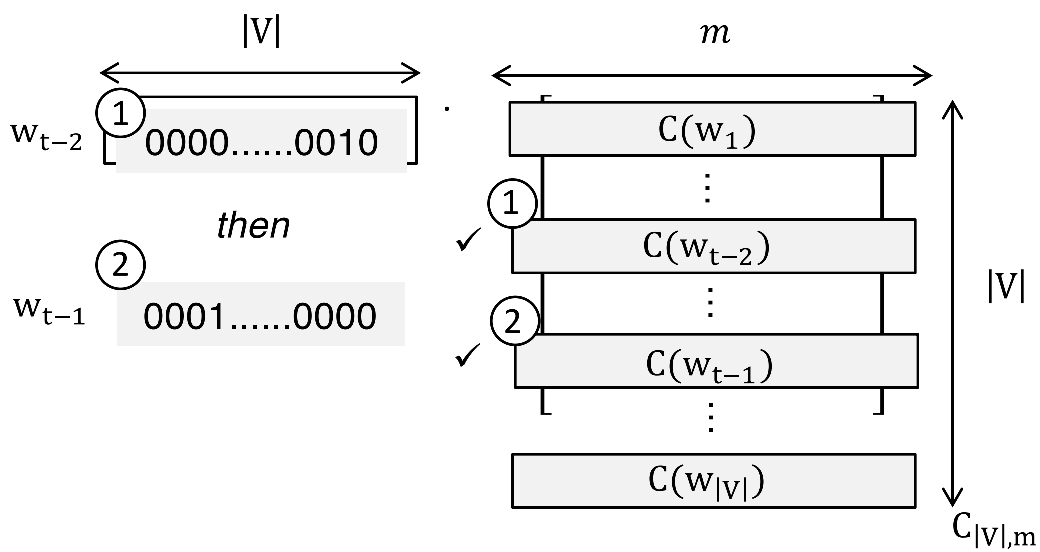
- $\mathrm{E} = -\log \widehat{\mathrm{P}}(\mathrm{w_t} \mid \mathrm{w_{t-n+1}} \dots \mathrm{w_{t-1}})$
- a probability between 0 and 1.
- On this support, the log is negative => –log term positive.
- makes sense to try to minimize it.
- Probability of word given the context be as high as possible (1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

p	log(p)	-log(p)
0,7	-0,15490196	0,15490196
0,2	-0,698970004	0,698970004

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):

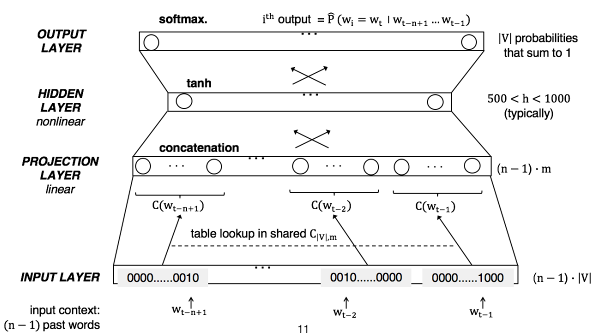
  for the i^{th} unit in the output layer: $\widehat{\mathrm{P}}(\mathrm{w_i} = \mathrm{w_t} \mid \mathrm{w_{t-n+1}} \dots \mathrm{w_{t-1}}) = \dfrac{e^{y_{w_i}}}{\sum_{i'=1}^{|V|} e^{y_{w_{i'}}}}$

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

  $\rightarrow \boldsymbol{\theta} = (\boldsymbol{b}, \boldsymbol{d}, \boldsymbol{U}, \boldsymbol{H}, \boldsymbol{C})$

  - Complexity per training sequence: $\mathrm{n} * \mathrm{m} + \mathrm{n} * \mathrm{m} * \mathrm{h} + \mathbf{h} * |\mathbf{V}|$

    computational bottleneck: **nonlinear hidden layer** ($\mathrm{h} * |\mathrm{V}|$ term)



- **Training** is performed via stochastic gradient descent (learning rate ε):

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

(weights are initialized randomly, then updated via backpropagation)

NNLM facts

- tested on Brown (1.2M words, $|\mathrm{V}| \cong 16\mathrm{K}$) and AP News (14M words, $|\mathrm{V}| \cong 150\mathrm{K}$ reduced to 18K) corpuses

- Brown: $\mathrm{h} = 100$, $\mathrm{n} = 5$, $\mathrm{m} = 30$

- AP News: $\mathrm{h} = 60$, $\mathrm{n} = 6$, $\mathrm{m} = 100$, **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 $1/\widehat{\mathrm{P}}(\mathrm{w_t} \mid \mathrm{w_{t-n+1}} \ldots \mathrm{w_{t-1}})$

- 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**

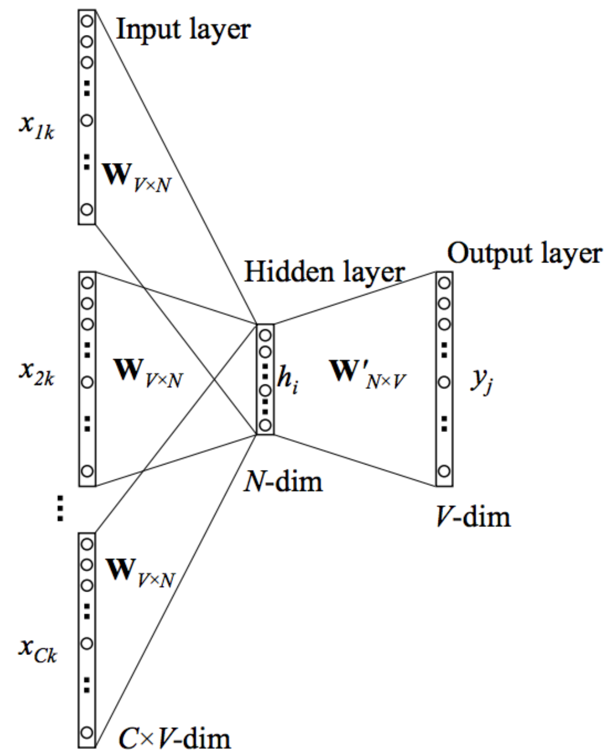
- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

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

Continuous Bag-of-Words (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

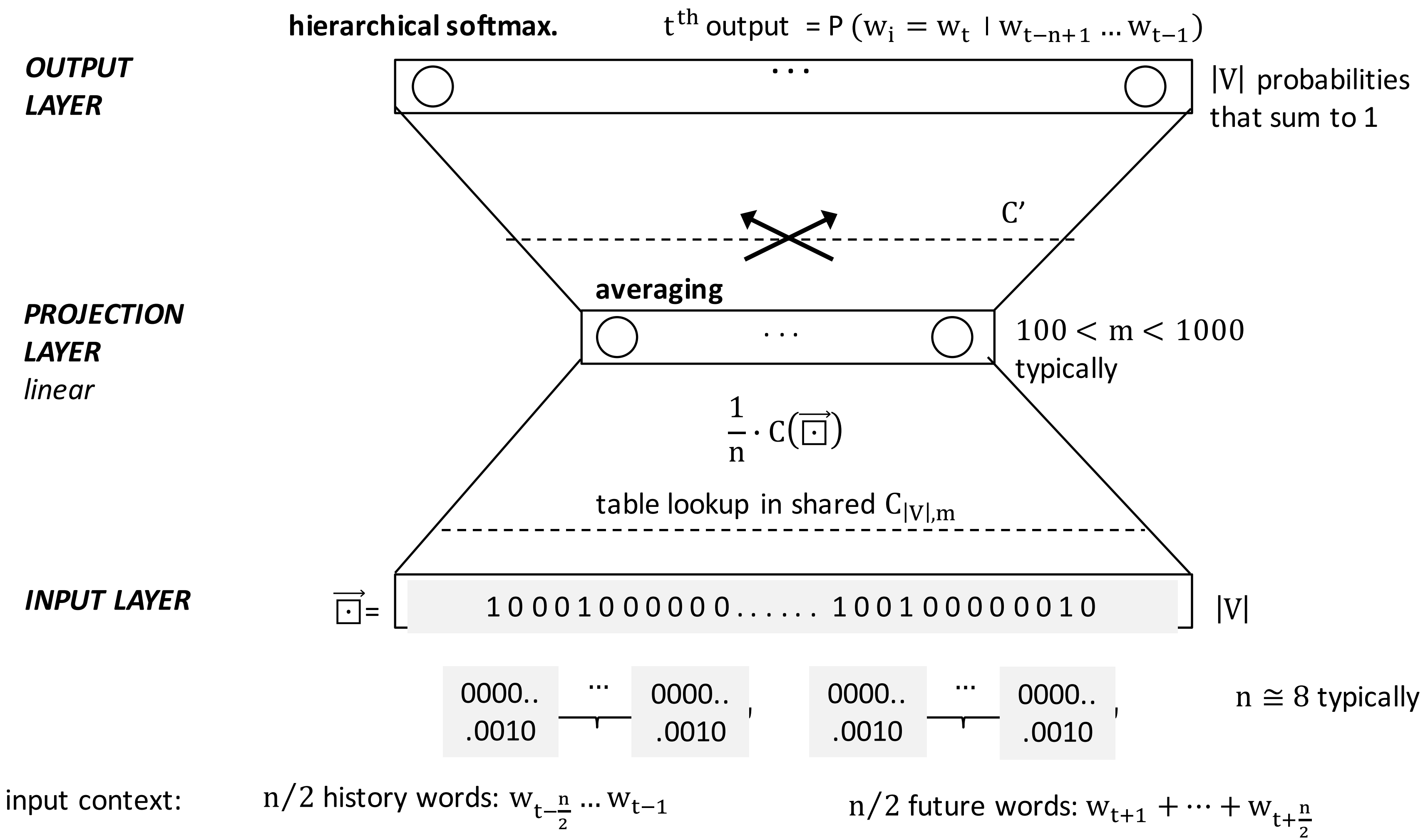


Efficient Estimation of Word Representations in Vector Space- Mikolov et al. 2013

Continuous Bag-of-Words (CBOW)

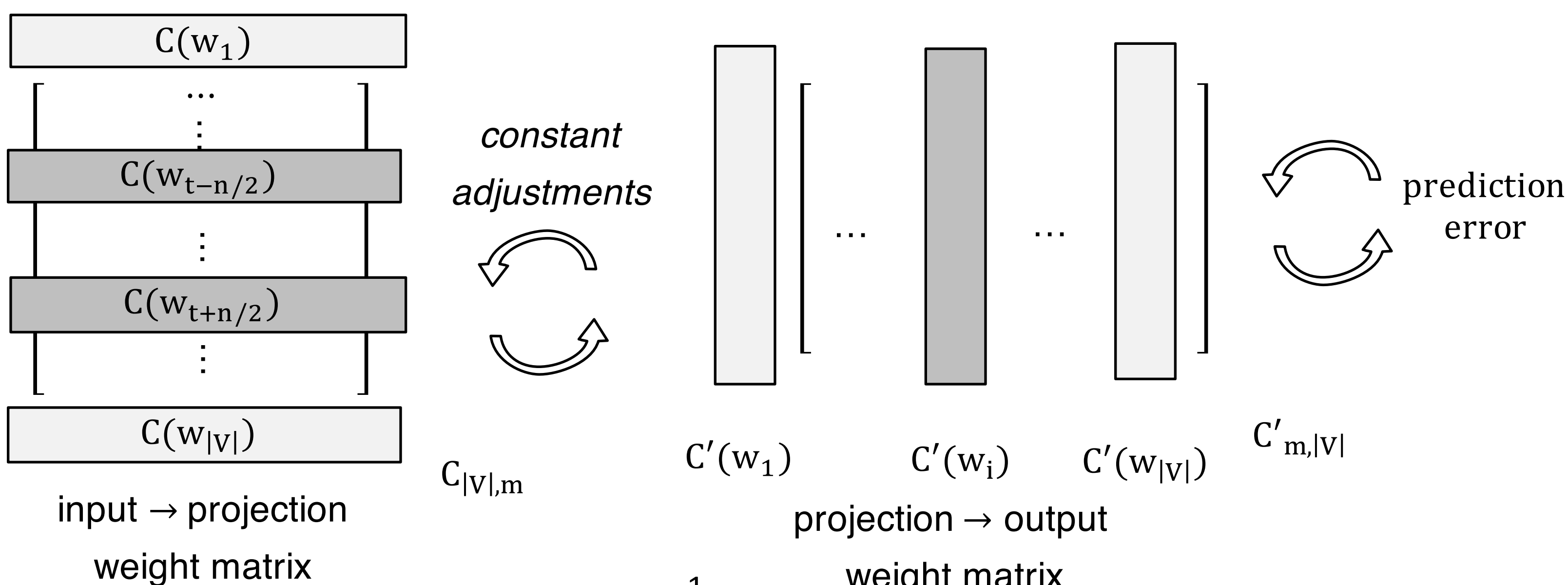
For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

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



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 $(w_i = w_t \mid \text{context})\ \forall\, w_i \in V$. We compare this distribution to the true probability distribution (1 for w_t, 0 elsewhere)
- **Back propagation**
- If P $(w_i = w_t \mid \text{context})$ 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 $(w_i = w_t \mid \text{context})$ 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

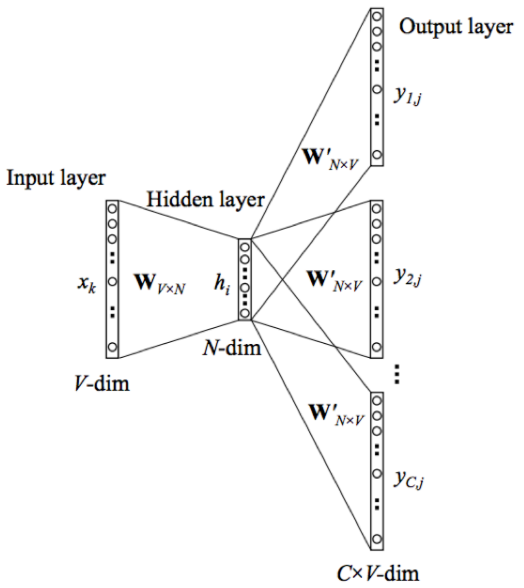
- instead of computing the probability of the target word w_t given its previous words, we calculate the probability of the surrounding word w_{t+j} given w_t

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

- $\boldsymbol{v^T}_{wt}$ is a column of $\boldsymbol{W}_{VxN}$ and $\boldsymbol{v}'_{w_{t+j}}$ is a column of $\boldsymbol{W'}_{NxV}$

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

- Objective function



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

Word2vec facts

- Complexity is $\mathbf{n} * \mathbf{m} + \mathbf{m} * \mathbf{log}|\mathbf{V}|$ (Mikolov et al. 2013a)
- **n**:size of the context window (~10) **nxm**: dimensions of the projection layer, **|V|** size of the vocabulary
- On Google news 6B words training corpus, with $|\mathbf{V}| \sim 10^6$:
  - CBOW with $\mathrm{m} = 1000$ took **2 days** to train on **140 cores**
  - Skip-gram with $\mathrm{m} = 1000$ took **2.5 days** on **125 cores**
  - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $\mathrm{m} = 100$ only!
  (note that $\mathrm{m} = 1000$ was not reasonably feasible on such a large training set)
- word2vec training speed ≅ 100K-5M words/s

- Quality of the word vectors:
  - ↗ 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: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%

- References: http://www.scribd.com/doc/285890694/NIPS-DeepLearning Workshop-NNforText#scribd
  https://code.google.com/p/word2vec/

GloVe

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

- Ratio of co-occurrence probabilities best distinguishes relevant words

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

- *Cast this into a lease square problem:*

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

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

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

model that utilizes
- count data
- bilinear prediction-based methods like word2vec

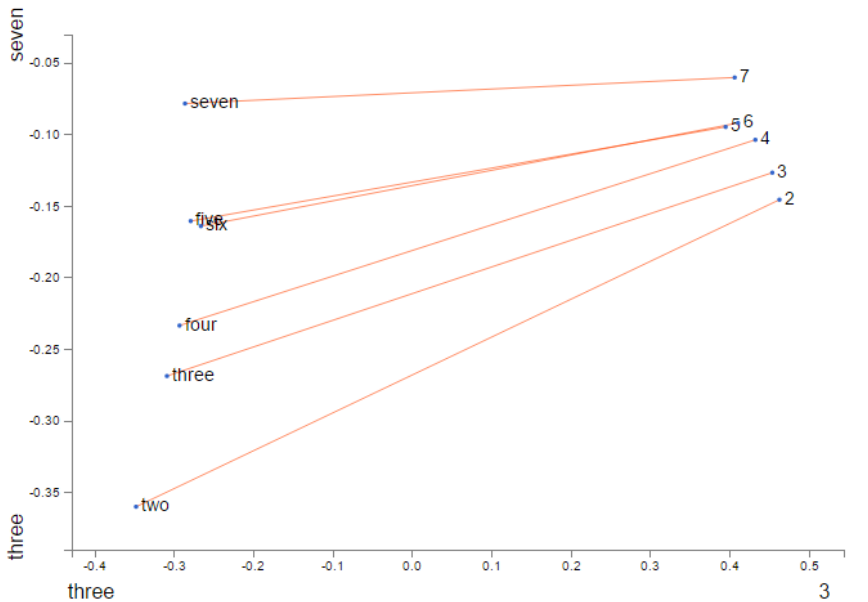
https://nlp.stanford.edu/pubs/glove.pdf

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

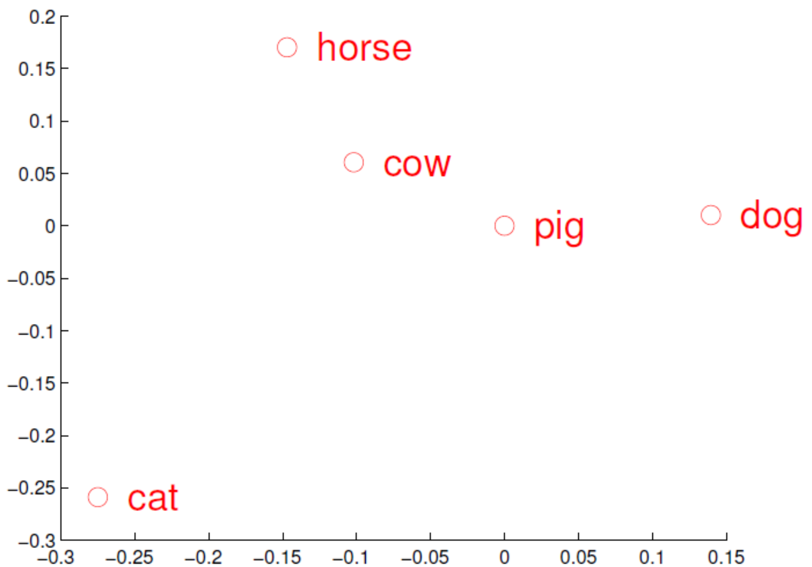
- Word analogies
- Find similar words
  - Semantic similarity
  - Syntactic similarity
- POS tagging
- Similar analogies for different languages
- Document classification

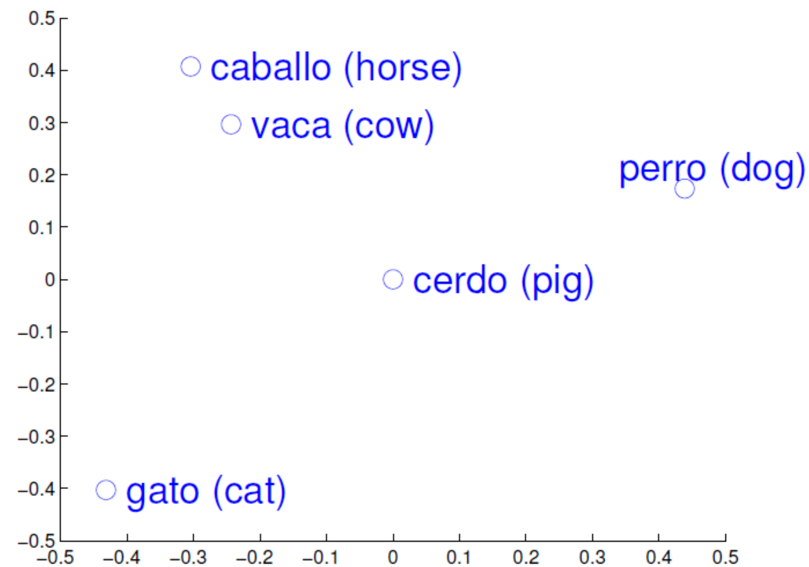


https://lamyiowce.github.io/word2viz/

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:

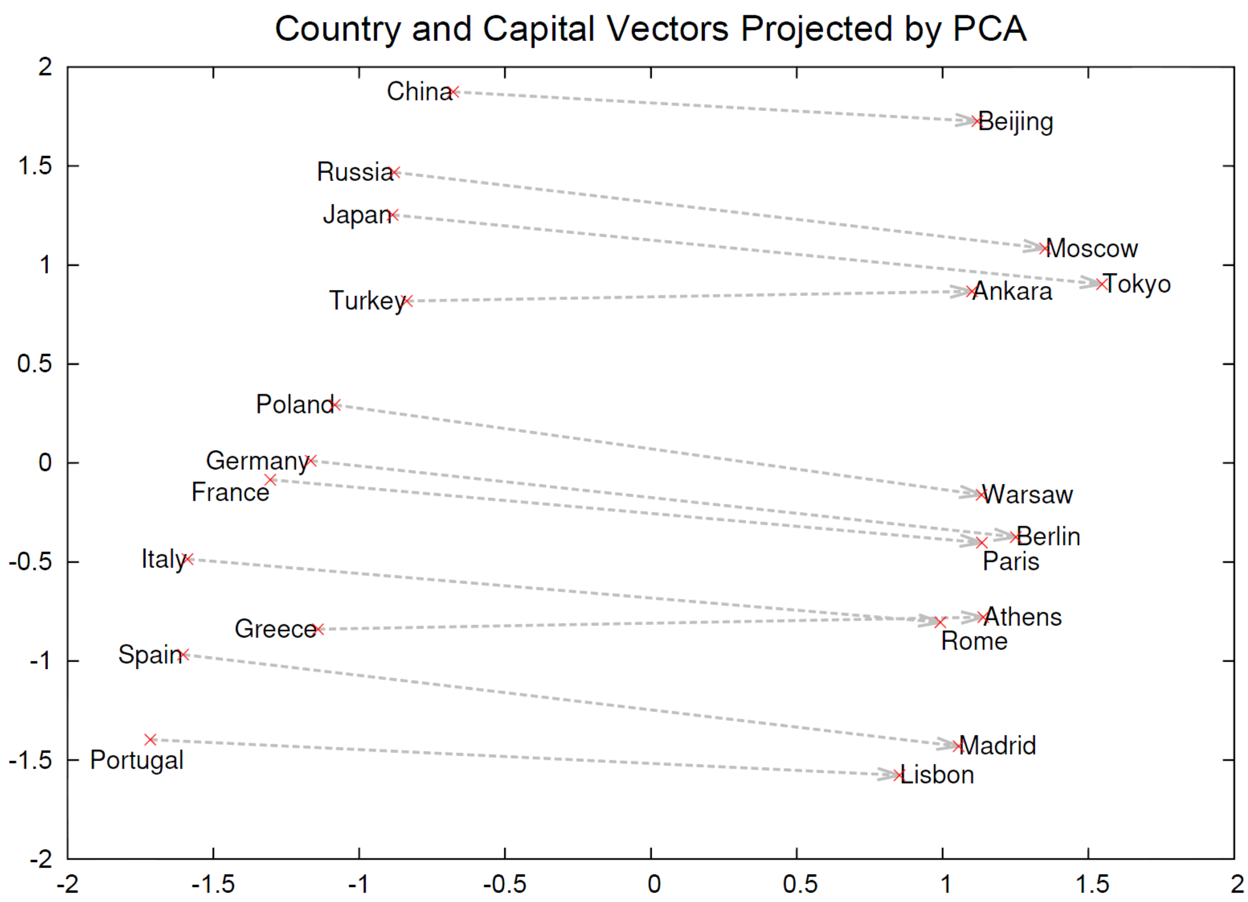




About 90% reported accuracy (Mikolov et al. 2013c)

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

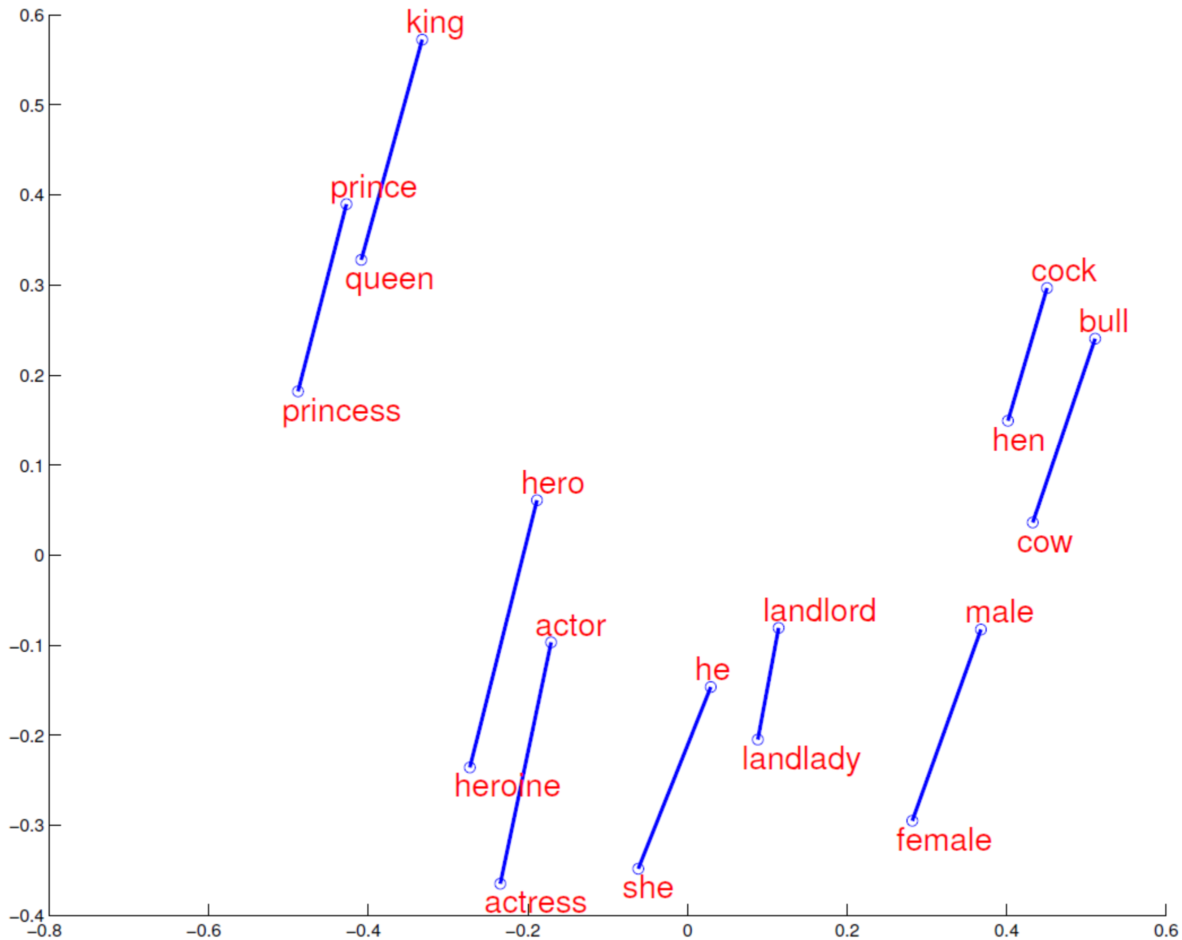
Remarkable properties of word vectors



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

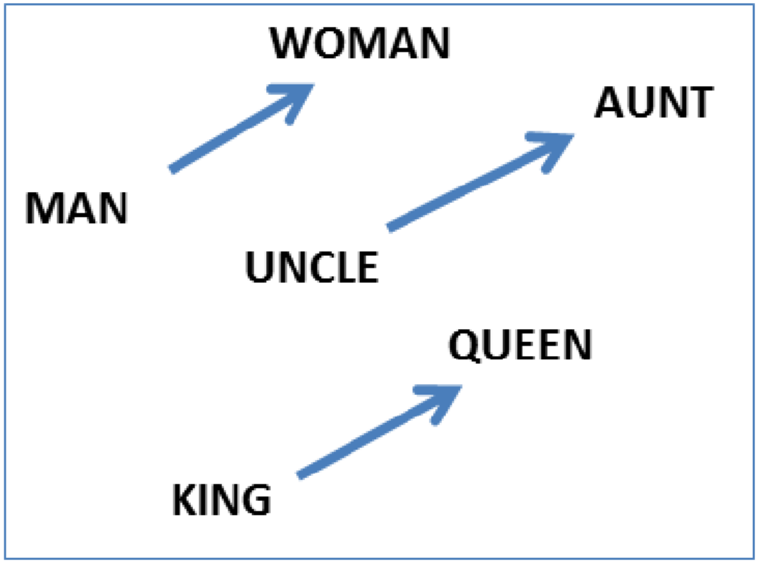
Remarkable properties of word vectors



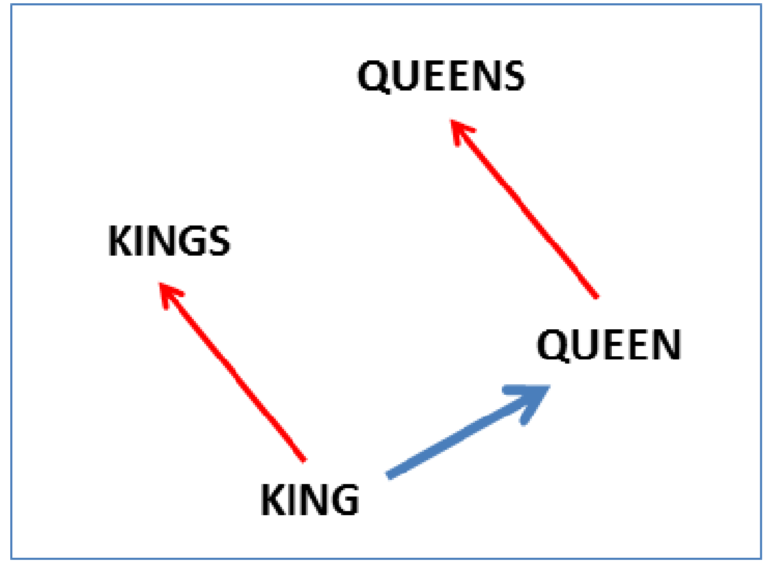
constant **female-male** difference vector

http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd

Remarkable properties of word vectors



constant **male-female** difference vector



constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

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

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

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

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

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

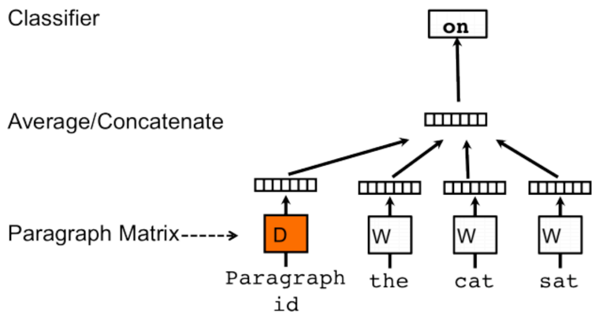
$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online demo (scroll down to end of tutorial)

http://rare-technologies.com/word2vec-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

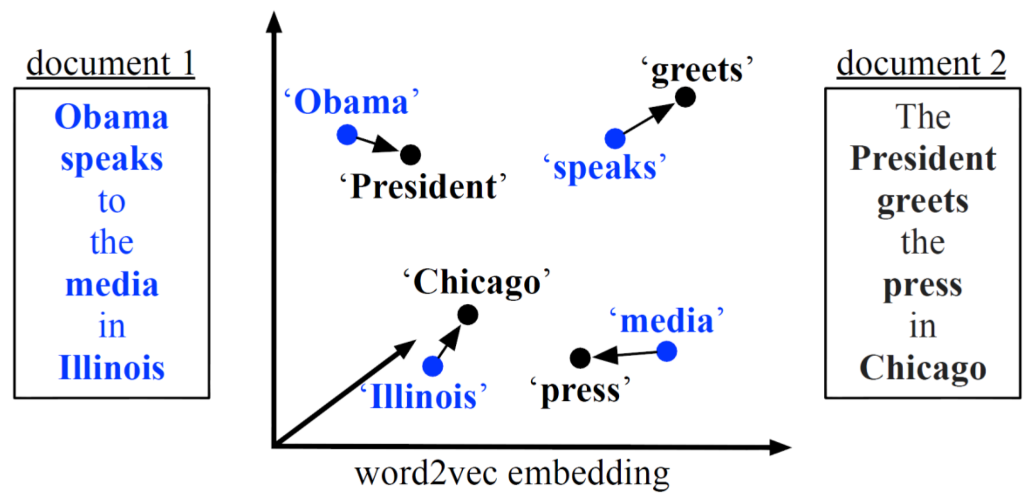


Model	Error rate (Positive/ Negative)	Error rate (Fine-grained)
Naïve Bayes (Socher et al., 2013b)	18.2 %	59.0%
SVMs (Socher et al., 2013b)	20.6%	59.3%
Bigram Naïve Bayes (Socher et al., 2013b)	16.9%	58.1%
Word Vector Averaging (Socher et al., 2013b)	19.9%	67.3%
Recursive Neural Network (Socher et al., 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	**12.2%**	**51.3%**

https://cs.stanford.edu/~quocle/**paragraph_vector**.pdf

Word Mover’s distance

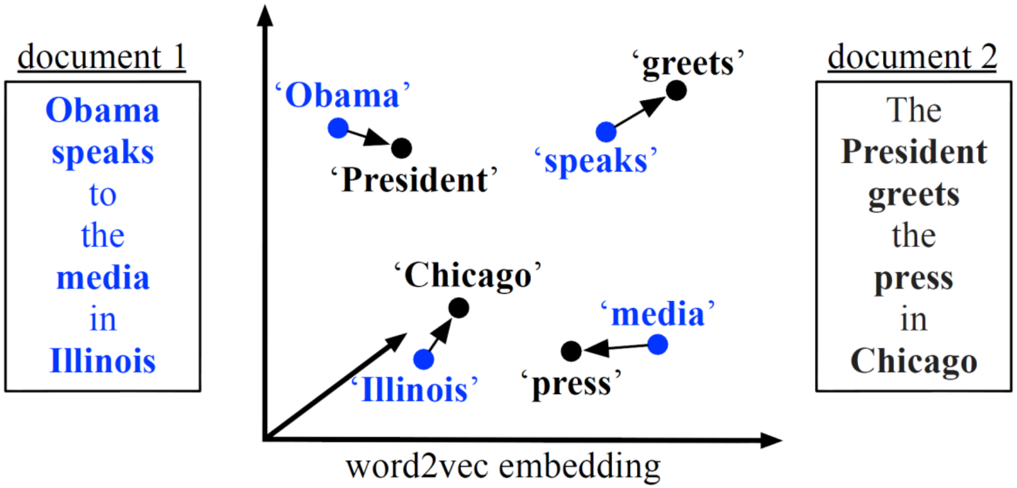
- “Edit” distance of 2 documents
- Based on word embedding representations
- Incorporate semantic similarity between individual word pairs into the document distance metric
- Based on “travel cost” between two words
- Calculates the cost of moving d to d’
- hyper-parameter free
- highly interpretable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

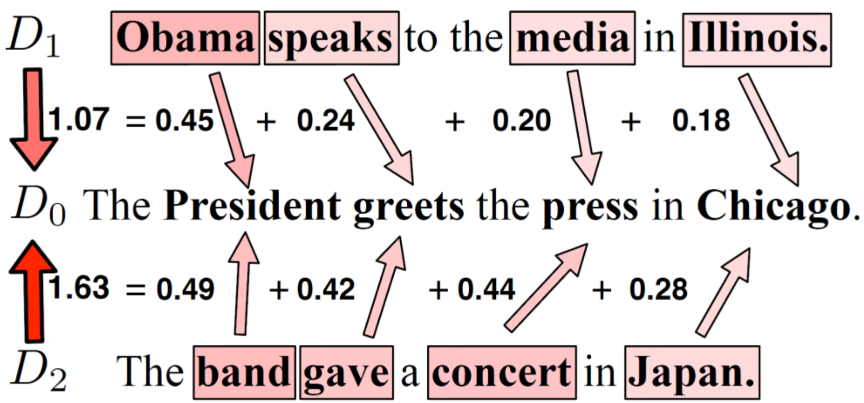


“minimum cumulative distance that all words in document 1 need to travel to exactly match document 2”

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.]

Word Mover's distance computation

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

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

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

$$\sum_j \mathbf{T}_{ij} = d_i.$$

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

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

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

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

Representation Learning for Greek

- Prototype and resources

http://archive.aueb.gr:7000

- Paper: Word Embeddings from Large-Scale Greek Web Content

https://arxiv.org/abs/1810.06694

ΕΥΧΑΡΙΣΤΙΕΣ …!

Google Scholar: https://bit.ly/2rwmvQU

Twitter: @mvazirg

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

References and online resources

- **Artificial neural networks: A tutorial**, AK Jain, J Mao, KM Mohiuddin - Computer, 1996
- introduction from a coder's perspective: http://karpathy.github.io/neuralnets/
- http://cs231n.github.io/
- online book: http://neuralnetworksanddeeplearning.com/index.html
- history of neural nets: http://stats.stackexchange.com/questions/182734/what-is-the-difference-between-a-neural-network-and-a-deep-neural-network
- nice blog post on neural nets applied to NLP: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/
- **A Primer on Neural Network Models for Natural Language Processing, Y. Goldberg,** *u.cs.biu.ac.il/~yogo/nnlp.pdf*