Introduction to

Information Retrieval

CS276: Information Retrieval and Web Search

Christopher Manning and Pandu Nayak

Lecture 15: Distributed Word Representations for Information Retrieval

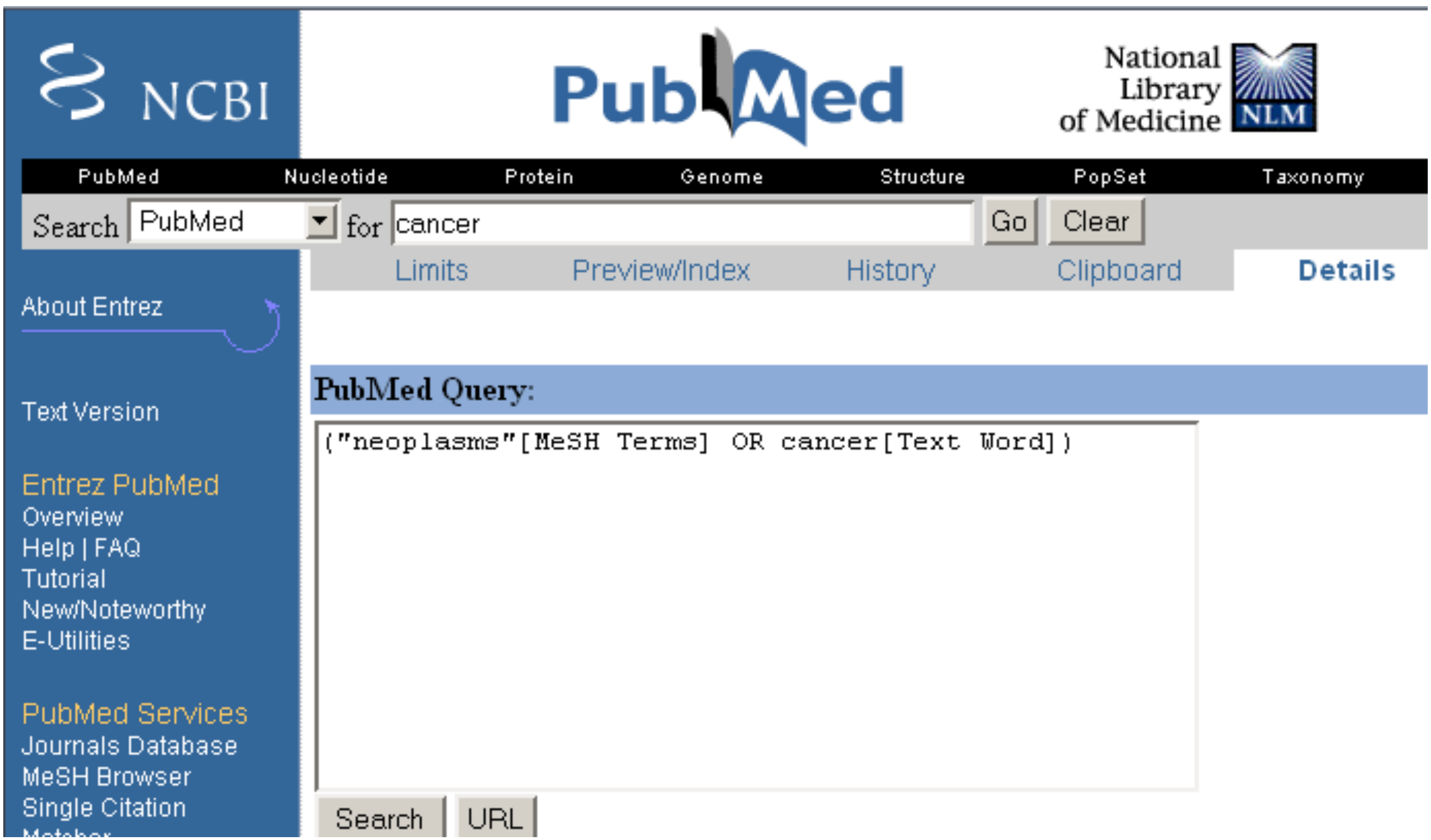
How can we more robustly match a user's query intent?

- If user searches for [Dell notebook battery size], we would like to match documents discussing "Dell laptop battery capacity"
- If user searches for [Seattle motel], we would like to match documents containing "Seattle hotel"
- Problem is that our query and document vectors are **orthogonal**
- motel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0]^T$
  hotel $[0\ 0\ 0\ 0\ 0\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 0] = 0$

How can we more robustly match a user's query intent?

- Use of anchor text may solve this by providing human authored synonyms, but not for new or less popular web pages, or non-hyperlinked collections
- Relevance feedback could allow us to capture this if we get near enough to matching documents with these words
- We can also fix this with information on word similarities:
  - A manual thesaurus of synonyms
  - A measure of word similarity
    - Calculated from a big document collection
    - Calculated by query log mining (common on the web)

Example of manual thesaurus



Thesaurus-based query expansion

- For each term, *t*, in a query, expand the query with synonyms and related words of *t* from the thesaurus
  - feline → feline cat
- May weight added terms less than original query terms.
- Generally increases recall
- Widely used in many science/engineering fields
- May significantly decrease precision, particularly with ambiguous terms.
  - "interest rate" → "interest rate fascinate evaluate"
- There is a high cost of manually producing a thesaurus
  - And for updating it for scientific changes

Search log query expansion

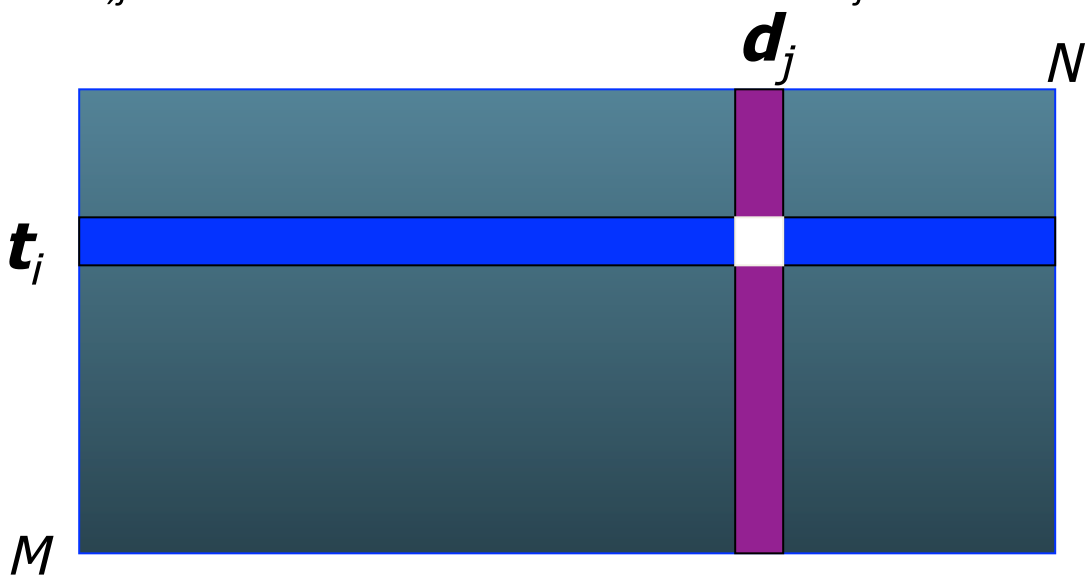
- Context-free query expansion ends up problematic
  - [light hair] ≈ [fair hair]
    - So expand [light] ⇒ [light fair]
  - But [bed light price] ≠ [bed fair price]
- You can learn query context-specific rewritings from search logs by attempting to identify the same user making a second attempt at the same user need
  - [Hinton word vector]
  - [Hinton word embedding]
- In this context, [vector] ≈ [embedding]
  - But not when talking about a *disease vector* or C++!

Automatic Thesaurus Generation

- Attempt to generate a thesaurus automatically by analyzing the collection of documents
- Fundamental notion: similarity between two words
- Definition 1: Two words are similar if they co-occur with similar words.
- Definition 2: Two words are similar if they occur in a given grammatical relation with the same words.
- You can harvest, peel, eat, prepare, etc. apples and pears, so apples and pears must be similar.
- Co-occurrence based is more robust, grammatical relations are more accurate. ⟸ Why?

Co-occurrence Thesaurus

- Simplest way to compute one is based on term-term similarities in $C = AA^T$ where A is term-document matrix.
- $w_{i,j}$ = (normalized) weight for (t_i,$\boldsymbol{d}_j$)



- For each t_i, pick terms with high values in C

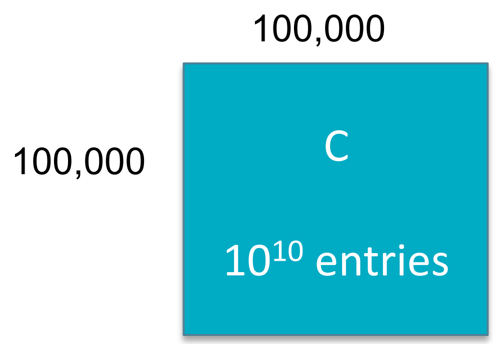
What does C contain if A is a term-doc incidence (0/1) matrix?

Automatic thesaurus generation example

Word	Nearest neighbors
absolutely	absurd, whatsoever, totally, exactly, nothing
bottomed	dip, copper, drops, topped, slide, trimmed
captivating	shimmer, stunningly, superbly, plucky, witty
doghouse	dog, porch, crawling, beside, downstairs
makeup	repellent, lotion, glossy, sunscreen, skin, gel
mediating	reconciliation, negotiate, cease, conciliation
keeping	hoping, bring, wiping, could, some, would
lithographs	drawings, Picasso, Dali, sculptures, Gauguin
pathogens	toxins, bacteria, organisms, bacterial, parasites
senses	grasp, psyche, truly, clumsy, naïve, innate

Automatic Thesaurus Generation Issues

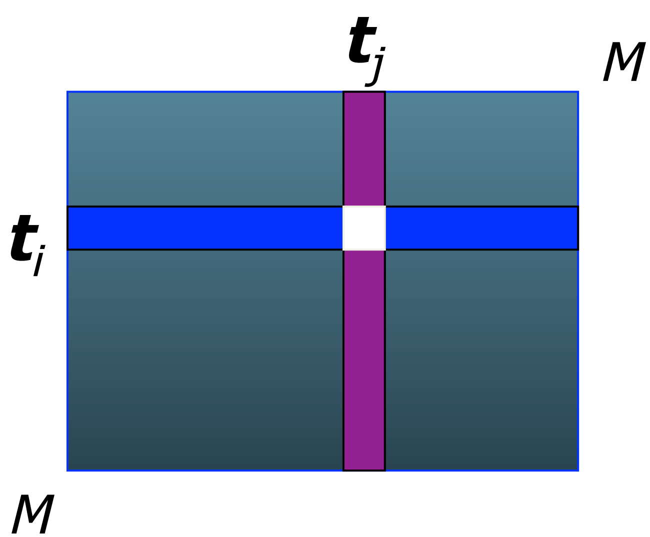
- Quality of associations is usually a problem
- Sparsity



- Term ambiguity may introduce irrelevant statistically correlated terms.
  - "planet earth facts" → "planet earth soil ground facts"
- Since terms are highly correlated anyway, expansion may not retrieve many additional documents.

Co-occurrence Thesaurus

- Since terms are highly correlated anyway, expansion may not retrieve many additional documents
- Really want "second order" similarity: terms that appear in similar term contexts – perhaps local window, not whole doc
- Simplest way to compute one is a term-term matrix $D = CC^T$ based on term-term similarities in $C = AA^T$ where A is term-document matrix. For each t_i, pick terms with high values in D



What does D contain if A was a term-doc incidence (0/1) matrix?

Can you directly learn term relations?

- Basic IR is scoring on $q^T d$
- No treatment of synonyms; no machine learning
- Can we learn parameters W to rank via $q^T W d$
- Problem is again sparsity – W is huge > 10^{10}

Is there a better way?

- Idea:
  - Can we learn a low dimensional representation of a words in $\mathbb{R}^d$ such that dot products $u^T v$ express word similarity?
  - We could still if we want to include a "translation" matrix between vocabularies (e.g., cross-language): $u^T W v$
    - But now W is small!
  - Supervised Semantic Indexing (Bai et al. *Journal of Information Retrieval* 2009) shows successful use of learning W for information retrieval
- But we'll develop direct similarity in this class

Distributional similarity based representations

- You can get a lot of value by representing a word by means of its neighbors
- "You shall know a word by the company it keeps"
  - (J. R. Firth 1957: 11)
- One of the most successful ideas of modern statistical NLP

government debt problems turning into banking crises as has happened in
saying that Europe needs unified banking regulation to replace the hodgepodge

- ↖ These words will represent *banking* ↗

Solution: Low dimensional vectors

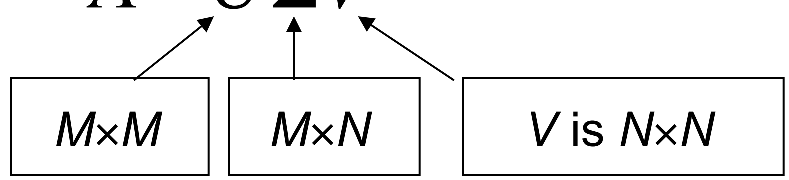
- The number of topics that people talk about is small (in some sense)
  - Clothes, movies, politics, …
- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually around 25 – 1000 dimensions

- How to reduce the dimensionality?
  - Go from big, sparse co-occurrence count vector to low dimensional "word embedding"

Traditional Way: Singular Value Decomposition

For an $M \times N$ matrix $\mathbf{A}$ of rank r there exists a factorization (Singular Value Decomposition = **SVD**) as follows:

$$A = U \Sigma V^T$$



(Not proven here. See *IIR* chapter 18)

Singular Value Decomposition

$$A = U\Sigma V^T$$

M×*M* | *M*×*N* | *V* is *N*×*N*

- $AA^T = (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)(V\Sigma U^T) = U\Sigma^2U^T$

The columns of ***U*** are orthogonal eigenvectors of $\boldsymbol{AA^T}$.

The columns of ***V*** are orthogonal eigenvectors of $\boldsymbol{A^TA}$.

Eigenvalues $\lambda_1 \ldots \lambda_r$ of $\boldsymbol{AA^T}$ are the eigenvalues of $\boldsymbol{A^TA}$.

$$\sigma_i = \sqrt{\lambda_i}$$

$$\Sigma = diag(\sigma_1 \ldots \sigma_r)$$

← *Singular values*

Low-rank Approximation

- SVD can be used to compute optimal **low-rank approximations**.
- Approximation problem: Find $\boldsymbol{A_k}$ of rank $\boldsymbol{k}$ such that

$$A_k = \min_{X:rank(X)=k} \|A - X\|_F$$

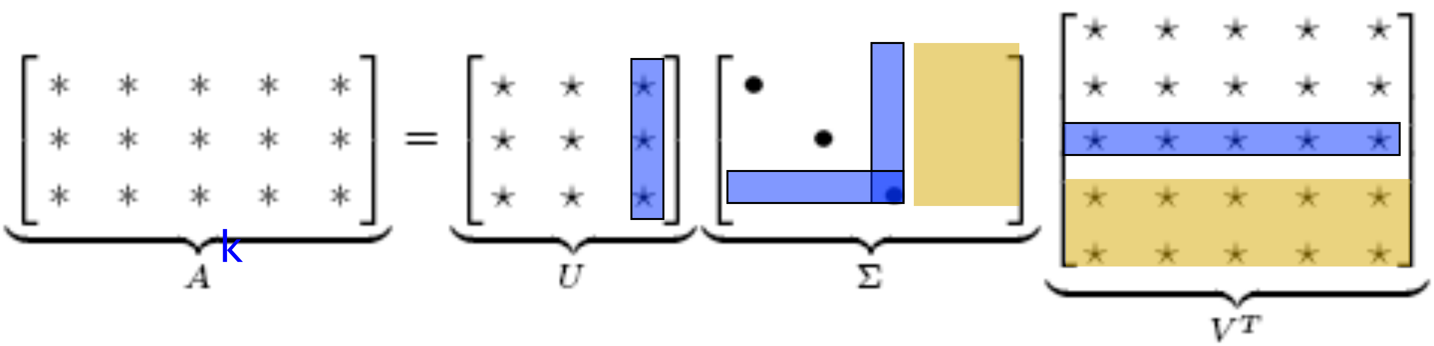
← *Frobenius norm*

$$\|\mathsf{A}\|_F \equiv \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}\,.$$

A_k and X are both $m \times n$ matrices.
Typically, want $k << r$.

Reduced SVD

- If we retain only k singular values, and set the rest to 0, then we don't need the matrix parts in color
- Then Σ is $k \times k$, U is $M \times k$, V^T is $k \times N$, and A_k is $M \times N$
- This is referred to as the reduced SVD
- It is the convenient (space-saving) and usual form for computational applications
- It's what Matlab gives you



Approximation error

- How good (bad) is this approximation?
- It's the best possible, measured by the Frobenius norm of the error:

$$\min_{X:rank(X)=k} \|A - X\|_F = \|A - A_k\|_F = \sigma_{k+1}$$

where the σ_i are ordered such that $\sigma_i \geq \sigma_{i+1}$.

Suggests why Frobenius error drops as *k* increases.

Latent Semantic Indexing via the SVD

Latent Semantic Indexing (LSI)

- Perform a **low-rank approximation** of **document-term matrix** (typical rank **100–300**)
- General idea
  - Map documents (*and* terms) to a low-dimensional representation.
  - Design a mapping such that the low-dimensional space reflects **semantic associations** (latent semantic space).
  - Compute document similarity based on the **inner product** in this **latent semantic space**

LSA Example

- A simple example term-document matrix (binary)

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

LSA Example

- Example of $C = U\Sigma V^T$: The matrix U

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

LSA Example

- Example of $C = U\Sigma V^T$: The matrix Σ

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

LSA Example

- Example of $C = U\Sigma V^T$: The matrix V^T

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

LSA Example: Reducing the dimension

U	1	2	3	4	5
ship	−0.44	−0.30	0.00	0.00	0.00
boat	−0.13	−0.33	0.00	0.00	0.00
ocean	−0.48	−0.51	0.00	0.00	0.00
wood	−0.70	0.35	0.00	0.00	0.00
tree	−0.26	0.65	0.00	0.00	0.00

Σ_2	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.00	0.00	0.00	0.00	0.00	0.00
4	0.00	0.00	0.00	0.00	0.00	0.00
5	0.00	0.00	0.00	0.00	0.00	0.00

Original matrix C vs. reduced $C_2 = U\Sigma_2 V^T$

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

C_2	d_1	d_2	d_3	d_4	d_5	d_6
ship	0.85	0.52	0.28	0.13	0.21	−0.08
boat	0.36	0.36	0.16	−0.20	−0.02	−0.18
ocean	1.01	0.72	0.36	−0.04	0.16	−0.21
wood	0.97	0.12	0.20	1.03	0.62	0.41
tree	0.12	−0.39	−0.08	0.90	0.41	0.49

Performing the maps

- Each row and column of *A* gets mapped into the *k*-dimensional LSI space, by the SVD.
- Claim – this is not only the mapping with the best (Frobenius error) approximation to *A*, but in fact *improves* retrieval.
- A query *q* is also mapped into this space, by

$$q_k = q^T U_k \Sigma_k^{-1}$$

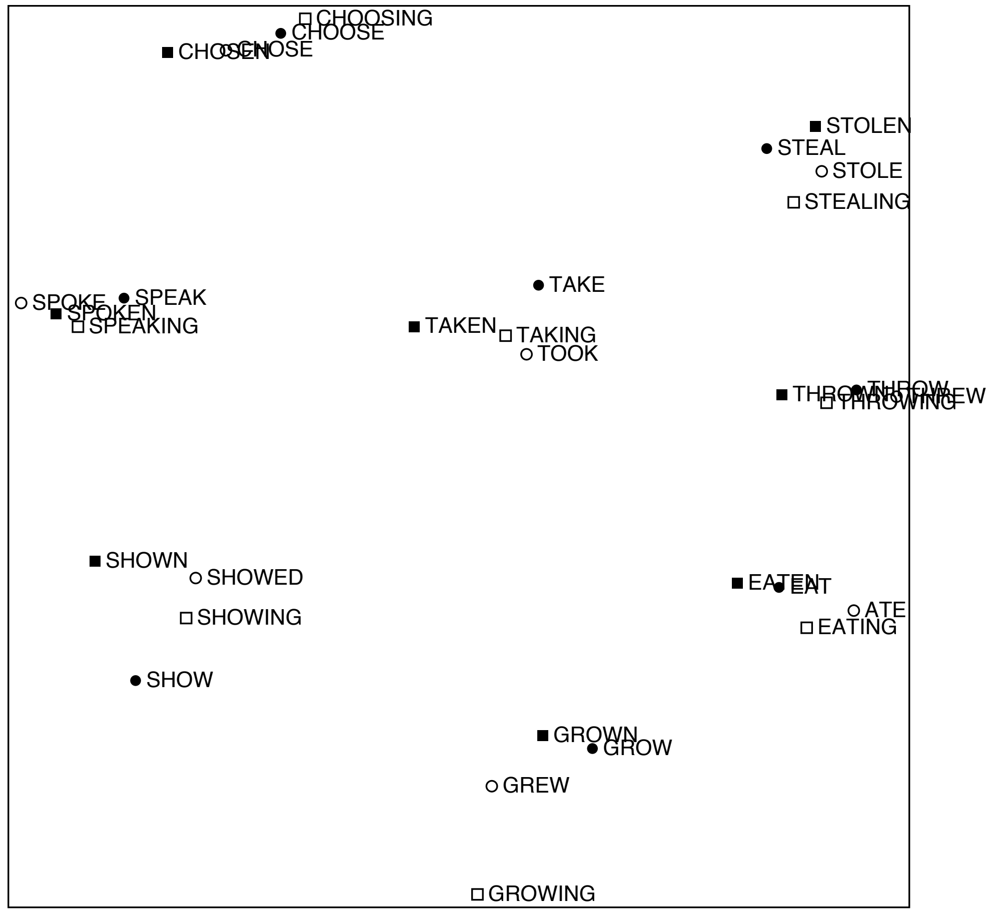
  - Query NOT a sparse vector.

“NEURAL EMBEDDINGS”

Idea: Directly learn low-dimensional word vectors

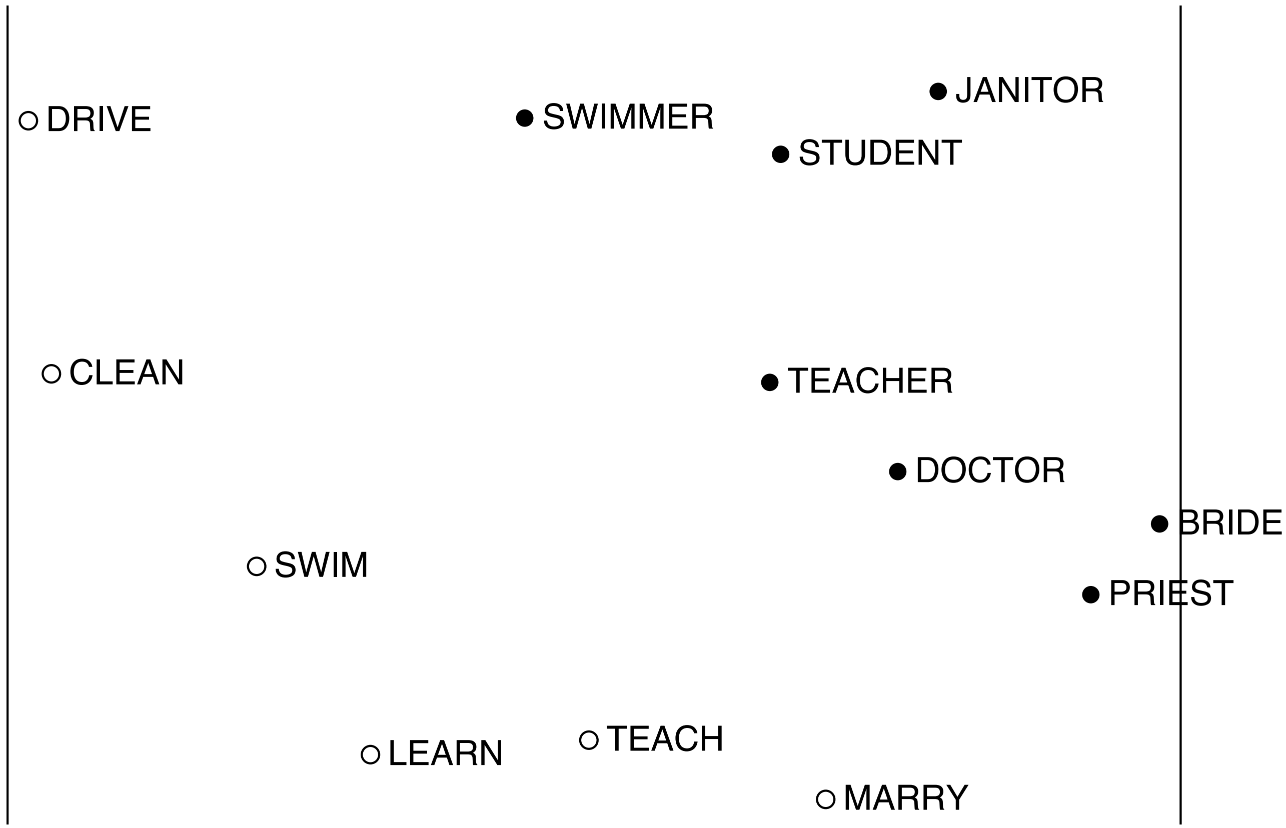
- Old idea. Relevant for this lecture & deep learning:
  - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  - A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - A recent, even simpler and faster model: word2vec (Mikolov et al. 2013) → intro now

Interesting semantic patterns emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting semantic patterns emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

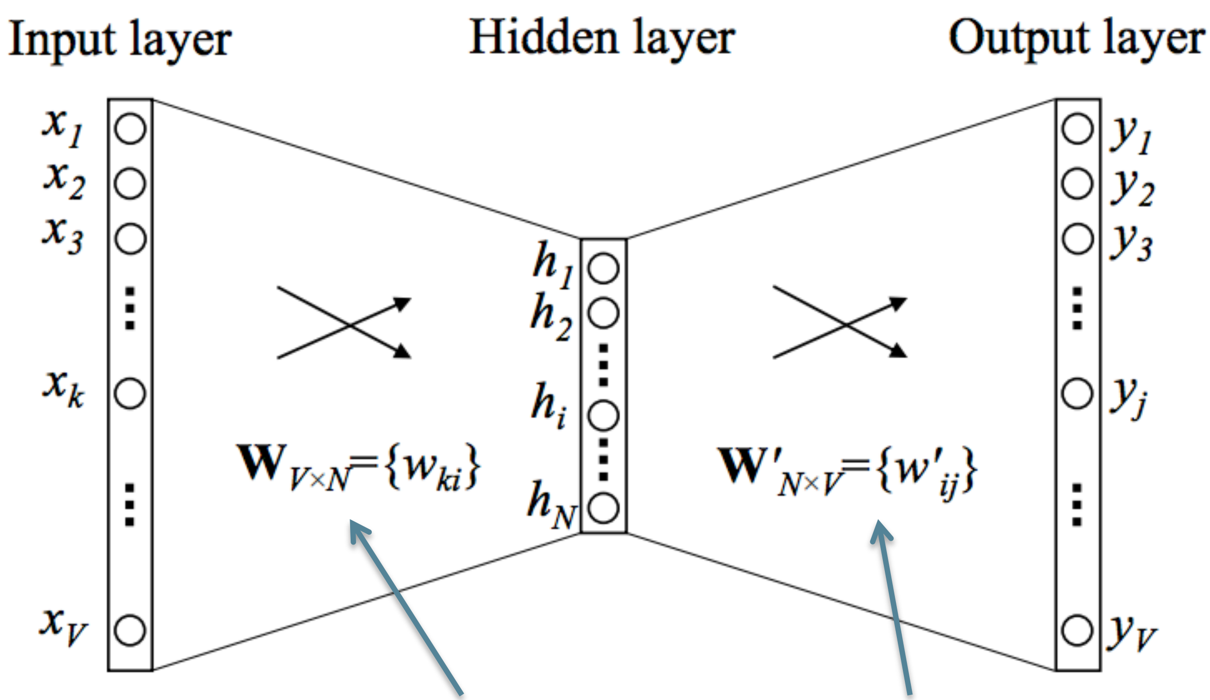
Main Idea of word2vec

- Instead of capturing co-occurrence counts directly, predict surrounding words of every word
- Faster and can easily incorporate a new sentence/ document or add a word to the vocabulary
- Two variants:
  - CBOW: Predict target from bag of words context
  - Skipgram: Predict context words from target (position-independent)

Details of 1 word context CBOW

- Objective function: Maximize the log probability of any target word given a context word

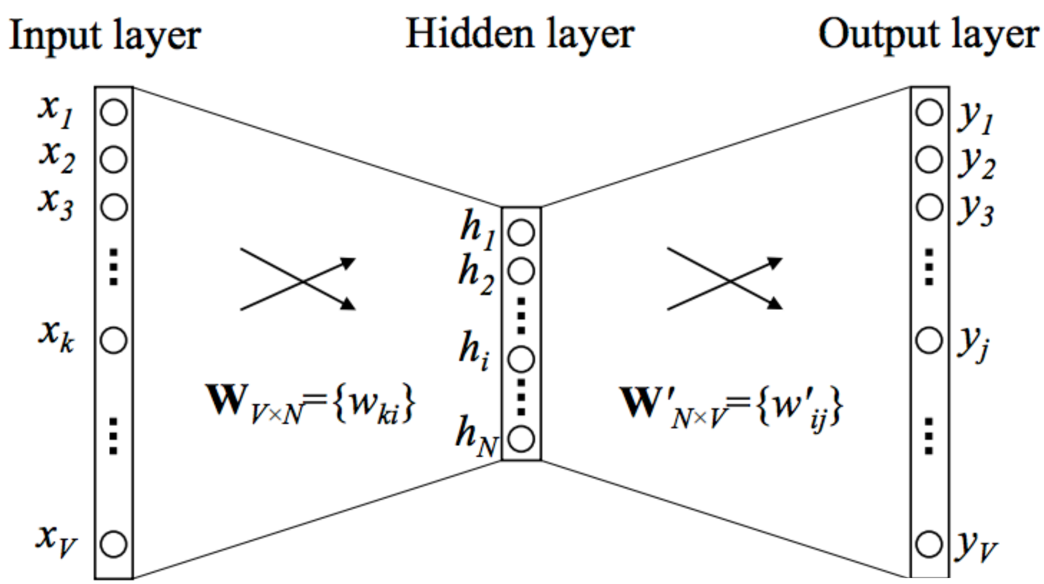
One-hot vector



Attempts to predict one-hot vector

These matrices have word vectors!

CBOW model (one context word)



$$\mathbf{h} = \mathbf{x}^T \mathbf{W} = \mathbf{W}_{(k,\cdot)} := \mathbf{v}_{w_I},$$

Word score $u_j = {\mathbf{v}'_{w_j}}^T \cdot \mathbf{h}$

$$p(w_j|w_I) = y_j = \frac{\exp(u_j)}{\sum_{j'=1}^{V} \exp(u_{j'})} = \frac{\exp\left({\mathbf{v}'_{w_O}}^T \mathbf{v}_{w_I}\right)}{\sum_{j'=1}^{V} \exp\left({\mathbf{v}'_{w'_j}}^T \mathbf{v}_{w_I}\right)}$$

CBOW model

Want to maximize

$$
\begin{aligned}
p(w_O|w_I) &= \max y_{j^*} \\
&= \max \log y_{j^*} \\
&= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E,
\end{aligned}
$$

- Do this by differentiating wrt each variable and walking downhill to minimize *E*. Remember:

$$\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

- Chain rule: If *y* = *f*(*u*) and *u* = *g*(*x*), i.e. y=f(g(x)), then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

CBOW model

Want to maximize

$$
\begin{aligned}
p(w_O|w_I) &= \max y_{j^*} \\
&= \max \log y_{j^*} \\
&= u_{j^*} - \log \sum_{j'=1}^{V} \exp(u_{j'}) := -E,
\end{aligned}
$$

$$\frac{\partial E}{\partial u_j} = y_j - t_j := e_j \qquad \text{where } t_j = \mathbb{1}(j = j^*)$$

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

CBOW model: Stochastic gradient descent updates

$$\frac{\partial E}{\partial w'_{ij}} = \frac{\partial E}{\partial u_j} \cdot \frac{u_j}{\partial w'_{ij}} = e_j \cdot h_i$$

$${w'_{ij}}^{(\text{new})} = {w'_{ij}}^{(\text{old})} - \eta \cdot e_j \cdot h_i$$

where $\eta > 0$ is the learning rate

$${\mathbf{v}'_{w_j}}^{(\text{new})} = {\mathbf{v}'_{w_j}}^{(\text{old})} - \eta \cdot e_j \cdot \mathbf{h}$$

CBOW model: W matrix

$$\frac{\partial E}{\partial h_i} = \sum_{j=1}^{V} \frac{\partial E}{\partial u_j} \cdot \frac{\partial u_j}{\partial h_i} = \sum_{j=1}^{V} e_j \cdot w'_{ij} := \mathrm{EH}_i$$

$$h_i = \sum_{k=1}^{V} x_k \cdot w_{ki}$$

$$\frac{\partial E}{\partial w_{ki}} = \frac{\partial E}{\partial h_i} \cdot \frac{\partial h_i}{\partial w_{ki}} = \mathrm{EH}_i \cdot x_k$$

$$\frac{\partial E}{\partial \mathbf{W}} = \mathbf{x} \cdot \mathrm{EH}$$

$$\mathbf{v}_{w_I}^{(\mathrm{new})} = \mathbf{v}_{w_I}^{(\mathrm{old})} - \eta \cdot \mathrm{EH}$$

Training regime

- Start with small, random vectors for words
- Iteratively go through millions of words in contexts
  - Work out prediction, work out error
  - Backpropagate error to update word vectors
  - Repeat
- Result is dense vectors for all words

$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Word similarity in word2vec

Linear Relationships in word2vec

These representations are *very good* at encoding similarity and dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

  Syntactically

  - $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
  - Similarly for verb and adjective morphological forms

  Semantically (Semeval 2012 task 2)

  - $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
  - $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Word Analogies

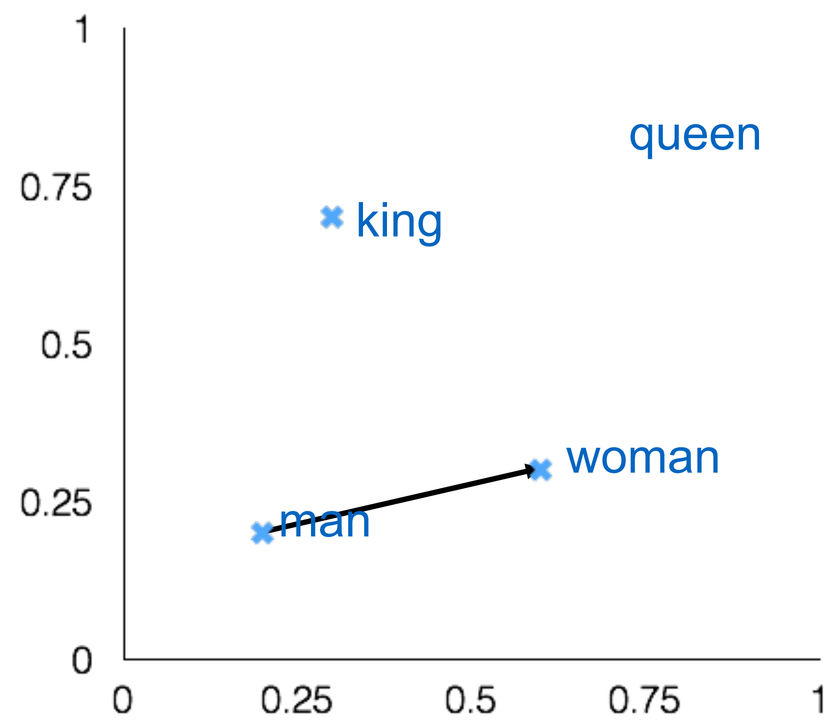
Test for linear relationships, examined by Mikolov et al.

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

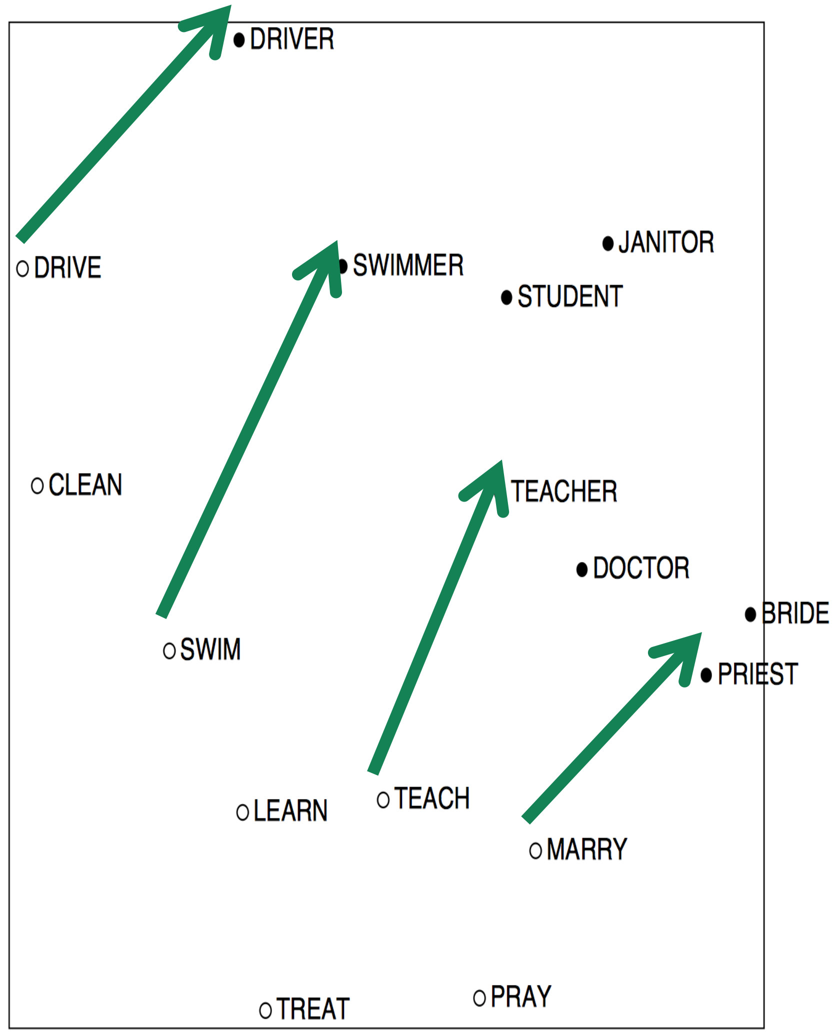
man:woman :: king:?

+	king	[0.30 0.70]
–	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]



COALS model (count-modified LSA)

[Rohde, Gonnerman & Plaut, ms., 2005]



Count based vs. direct prediction

LSA, HAL (Lund & Burgess),
COALS (Rohde et al),
Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to small counts

- NNLM, HLBL, RNN, word2vec Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight:

Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Encoding meaning in vector differences

[Pennington et al., EMNLP 2014]

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = fashion
$P(x \mid \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x \mid \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

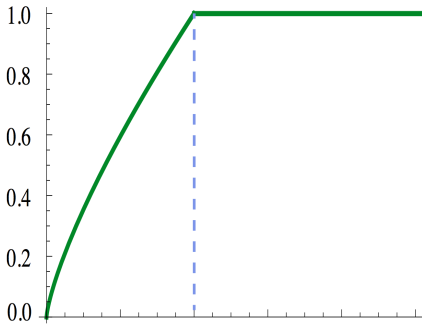
GloVe: A new model for learning word representations [Pennington et al., EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

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

$f \sim$



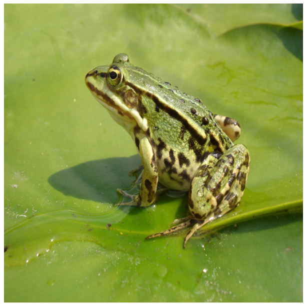
Word similarities

Nearest words to frog:

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus

litoria

leptodactylidae



rana

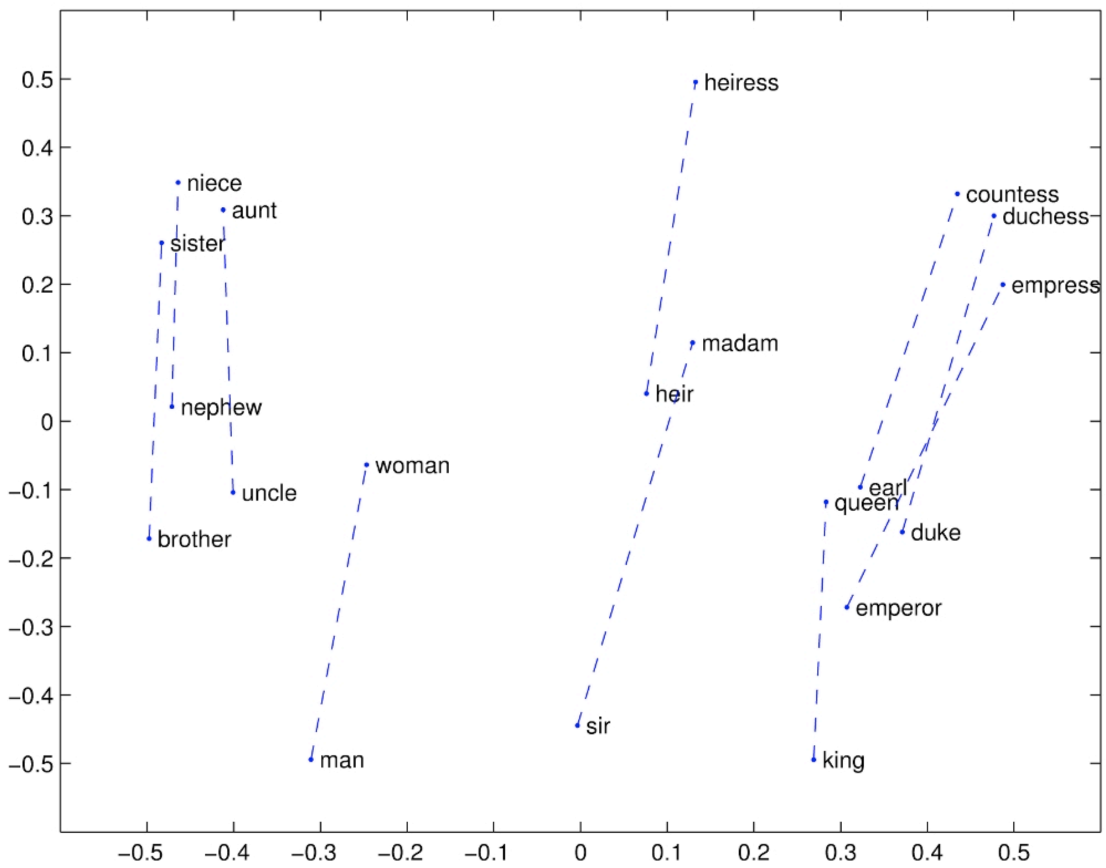
eleutherodactylus

http://nlp.stanford.edu/projects/glove/

Word analogy task [Mikolov, Yih & Zweig 2013a]

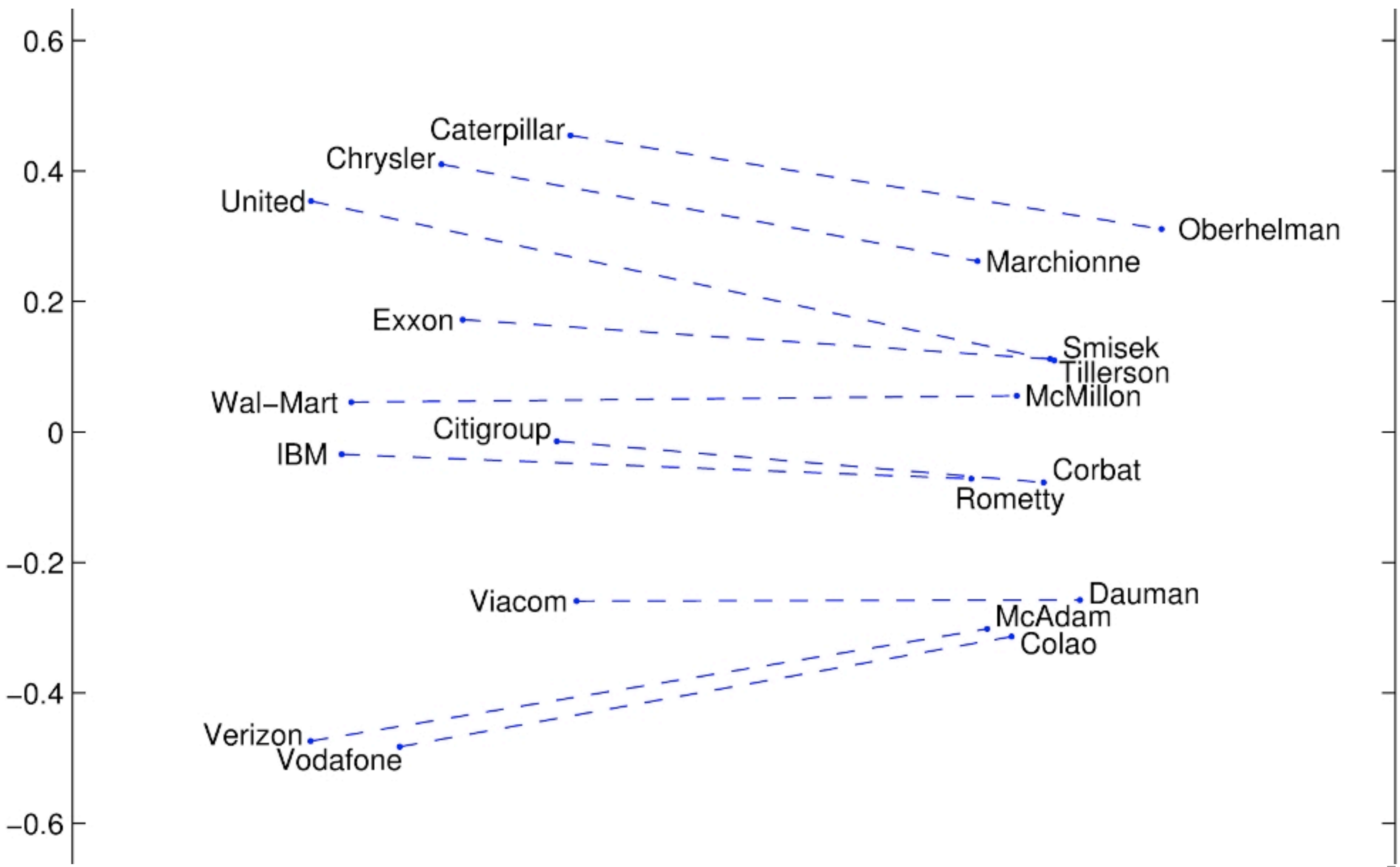
Model	**Dimensions**	**Corpus size**	**Performance (Syn + Sem)**
CBOW (Mikolov et al. 2013b)	300	1.6 billion	36.1

Glove Visualizations

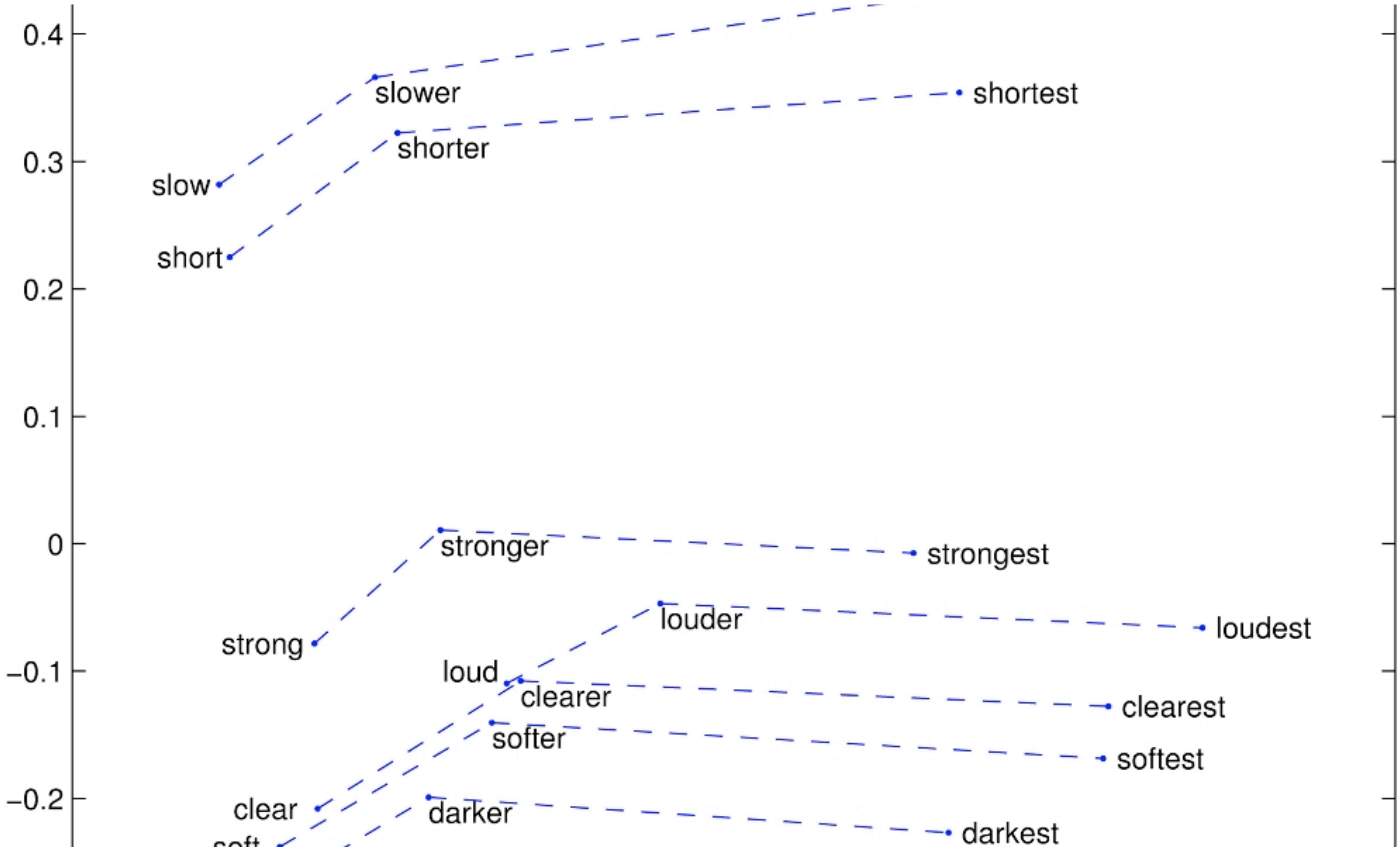


http://nlp.stanford.edu/projects/glove/

Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Word embeddings

Word embeddings are currently the hot new technology

Lots of applications whenever knowing word similarity helps prediction:

- Synonym handling in search
- Ad serving
- Language models
- Machine translation
- Sentiment analysis
- …