CSE 5243 INTRO. TO DATA MINING

Word Embedding

Yu Su, CSE@The Ohio State University

Source of slides

- Richard Socher's course in Stanford
- Tomas Mikolov's invited talk at the Deep Learning workshop in NIPS'13
- Dan Jurafsky's lecture about language modeling
- Tutorial at EMNLP'14: Embedding Methods for NLP
- Tutorial at ACL'14: New Directions in Vector Space Models of Meaning
- Manaal Faruqui's talk at NAACL'15: Retrofitting Word Vectors to Semantic Lexicons

How to let a computer understand meaning?

A cat sits on a mat.

#_$@^_&*^&_()_@_+@^=

Knowledge Representation

- Machine understandable representation of knowledge
- **Symbolic** solution, e.g., semantic lexicons like WordNet

hypernyms of 'panda' (is-a relation)

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

synonym sets of 'good'

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good

Problems with this symbolic representation

- Great as resource but missing nuances
  - e.g. **synonyms**: adept, expert, good, practiced, proficient, skillful?
- Requires human labor to create and adapt
- Subjective, sometimes hard to reach agreement
- Missing new words (hard to keep up to date):

  wicked, badass, nifty, crack, ace, wizard, ninjia

Problems with this symbolic representation

- Words are distinctive atomic symbols
- In vector space terms, this is a vector with one 1 and a lot of zeroes. We call this the "one-hot" representation.

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

- No way to capture word similarity

motel [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0] AND
hotel [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0] = 0

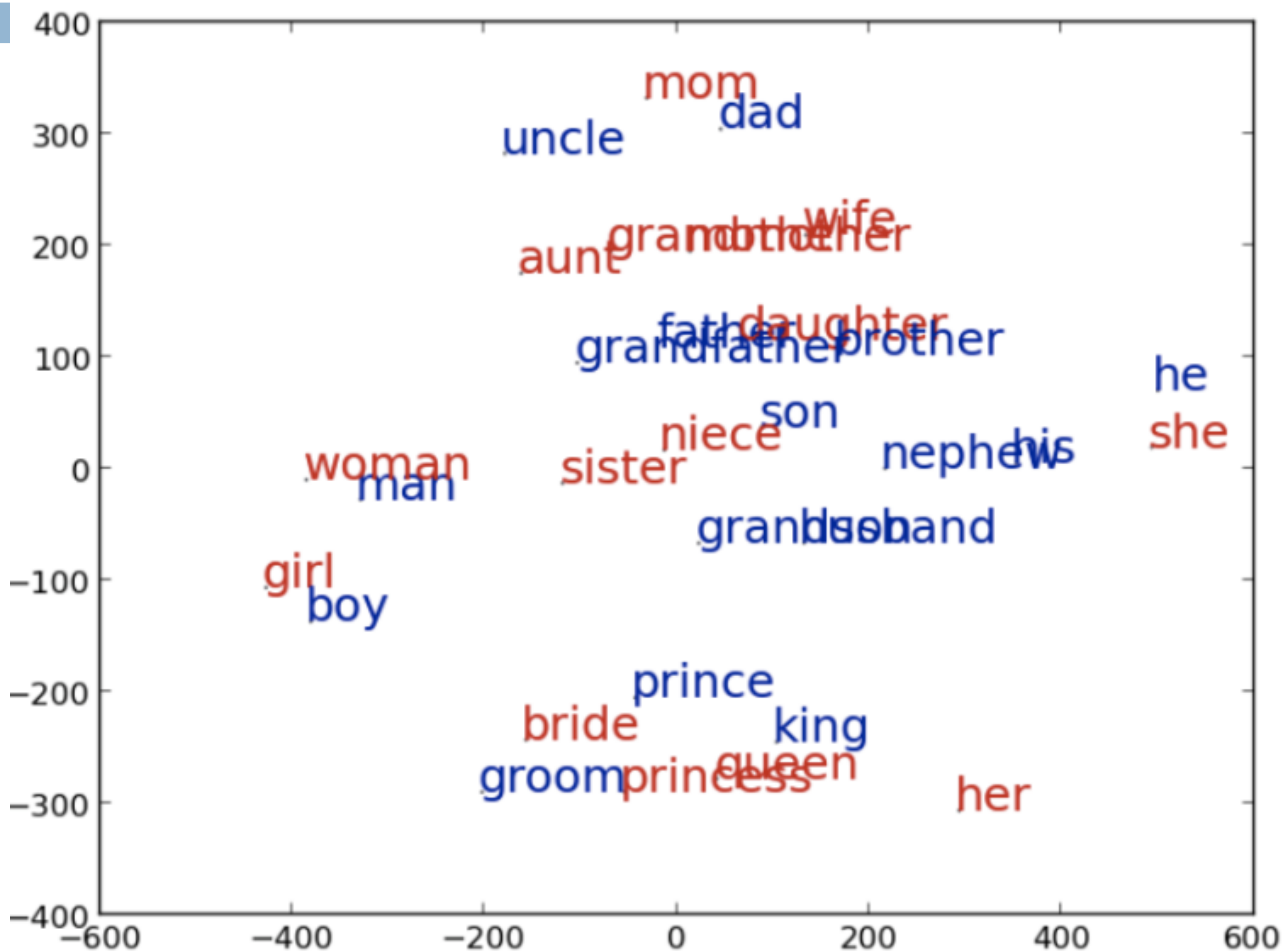
Is there another (probably better) way to represent the meaning of words? →

Statistical solution: word embedding

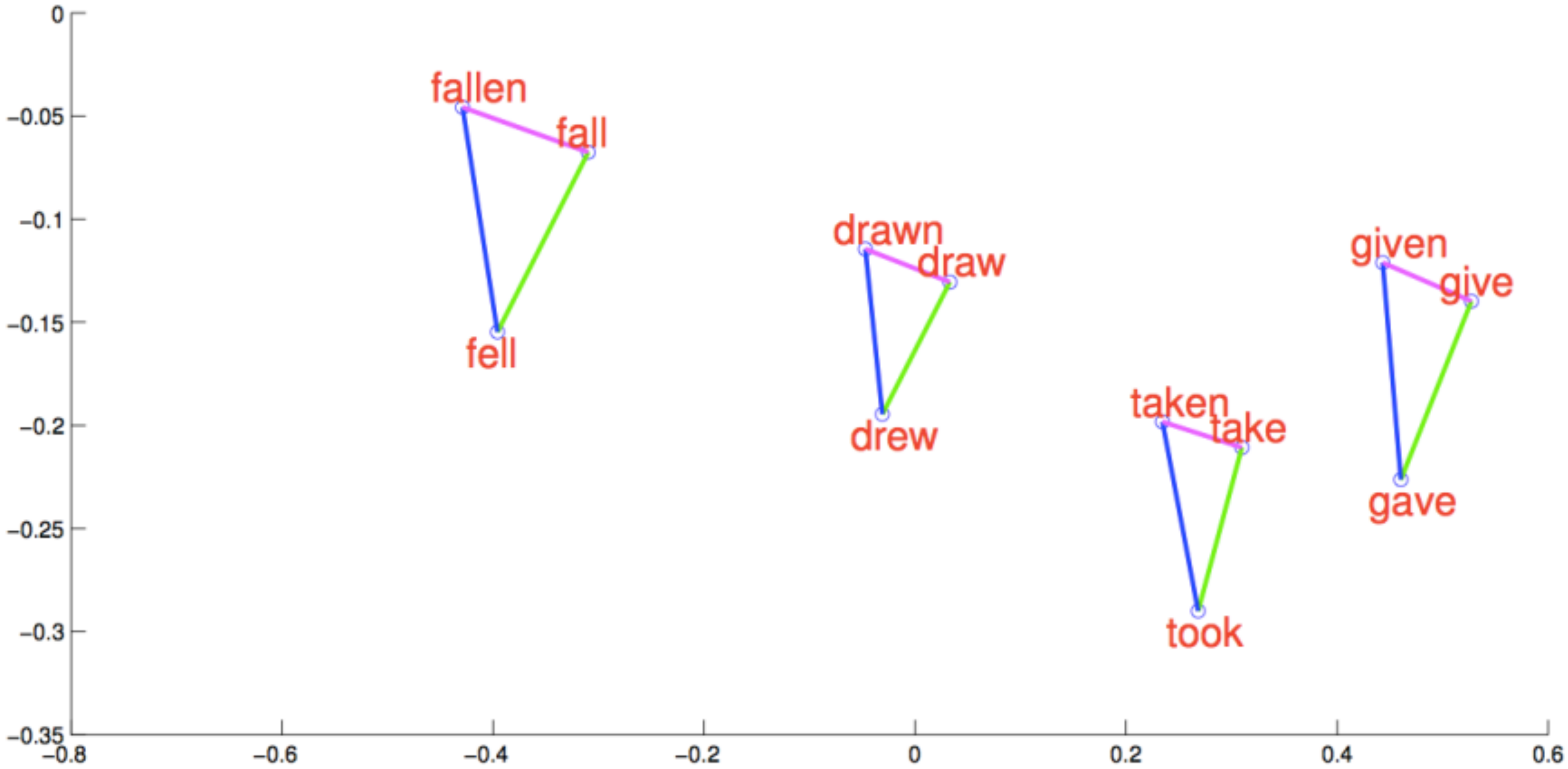
- Each word is represented as a dense vector
- Each dimension captures more information

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

(Expected) regularities in word vector space

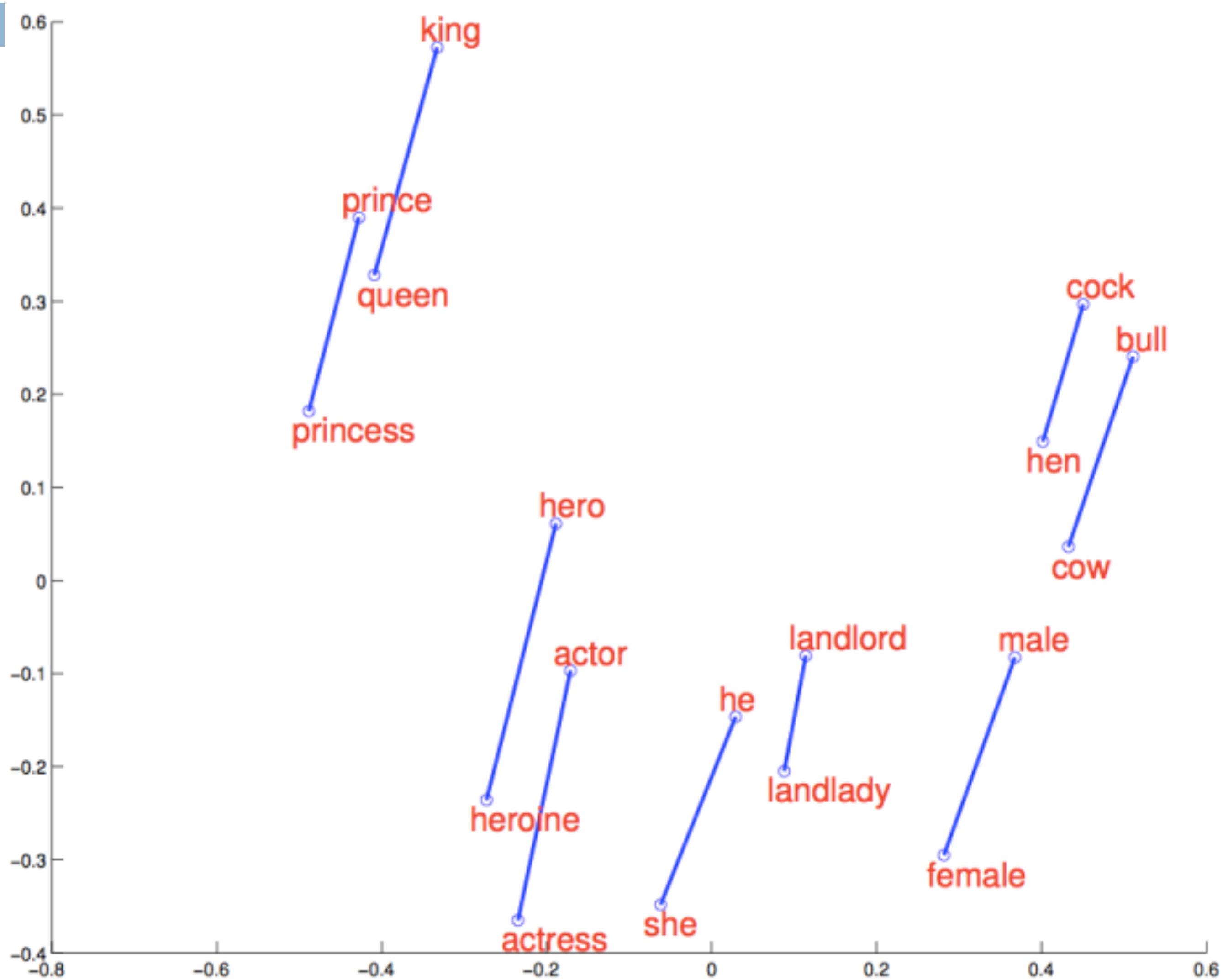


(Expected) regularities in word vector space



generated by PCA

(Expected) regularities in word vector space



generated by PCA

Q: How to generate word embedding?

A: Distributional semantics

Distributional semantics

- You can get a lot of value by representing a word by means of its neighbors (context)

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

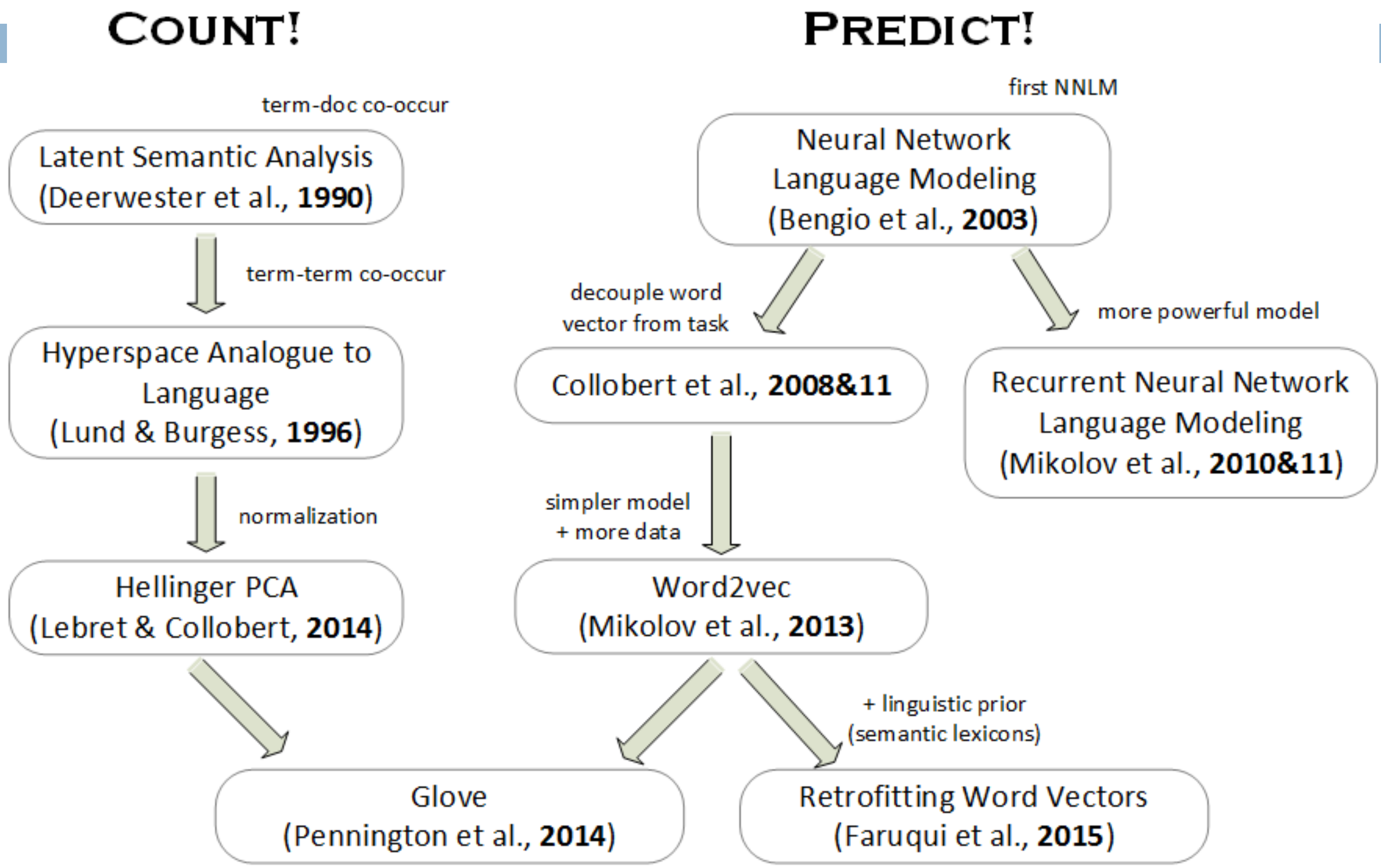
- One of the most successful ideas of modern statistical NLP

government debt problems turning into **banking** crises as has happened in

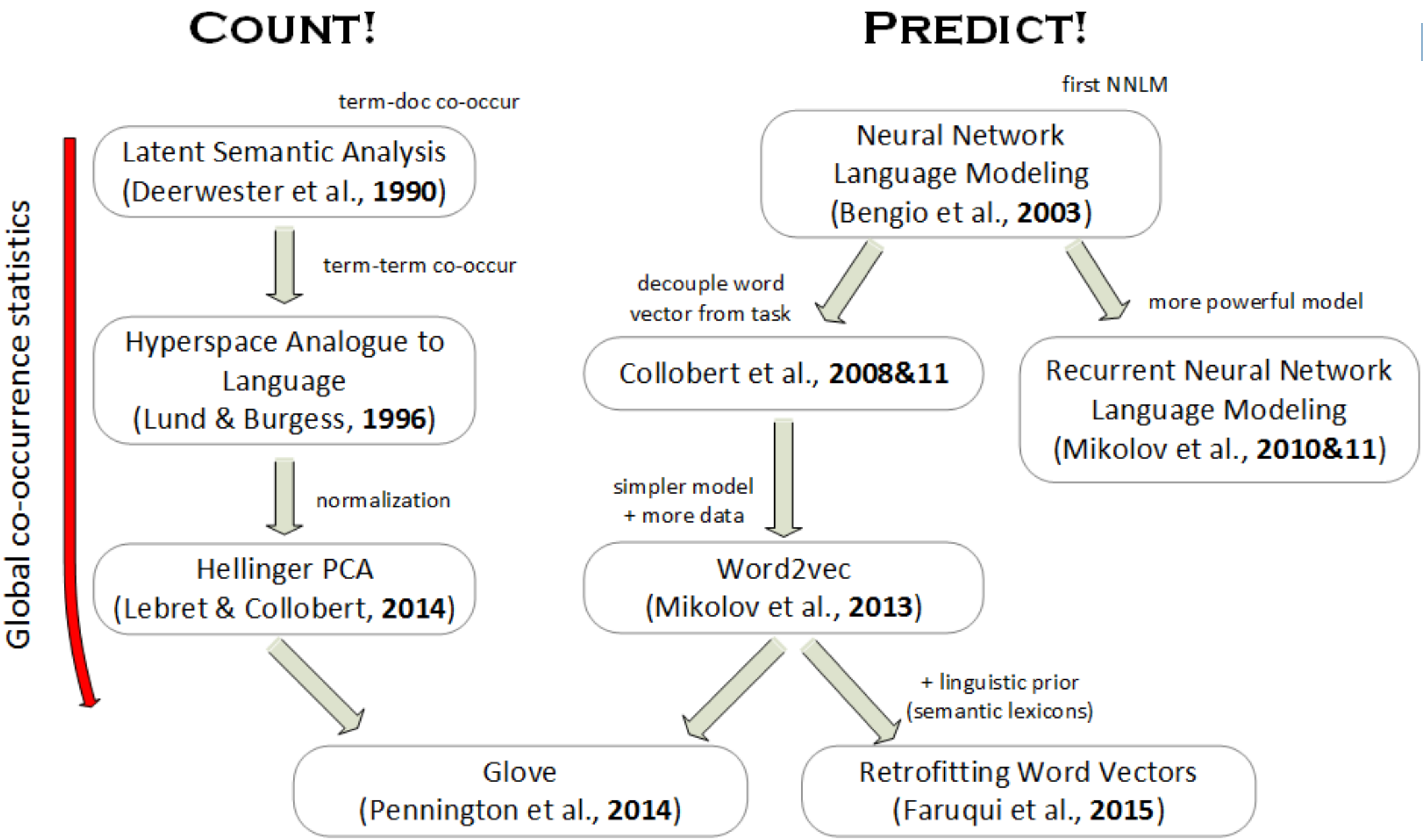
saying that Europe needs unified **banking** regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

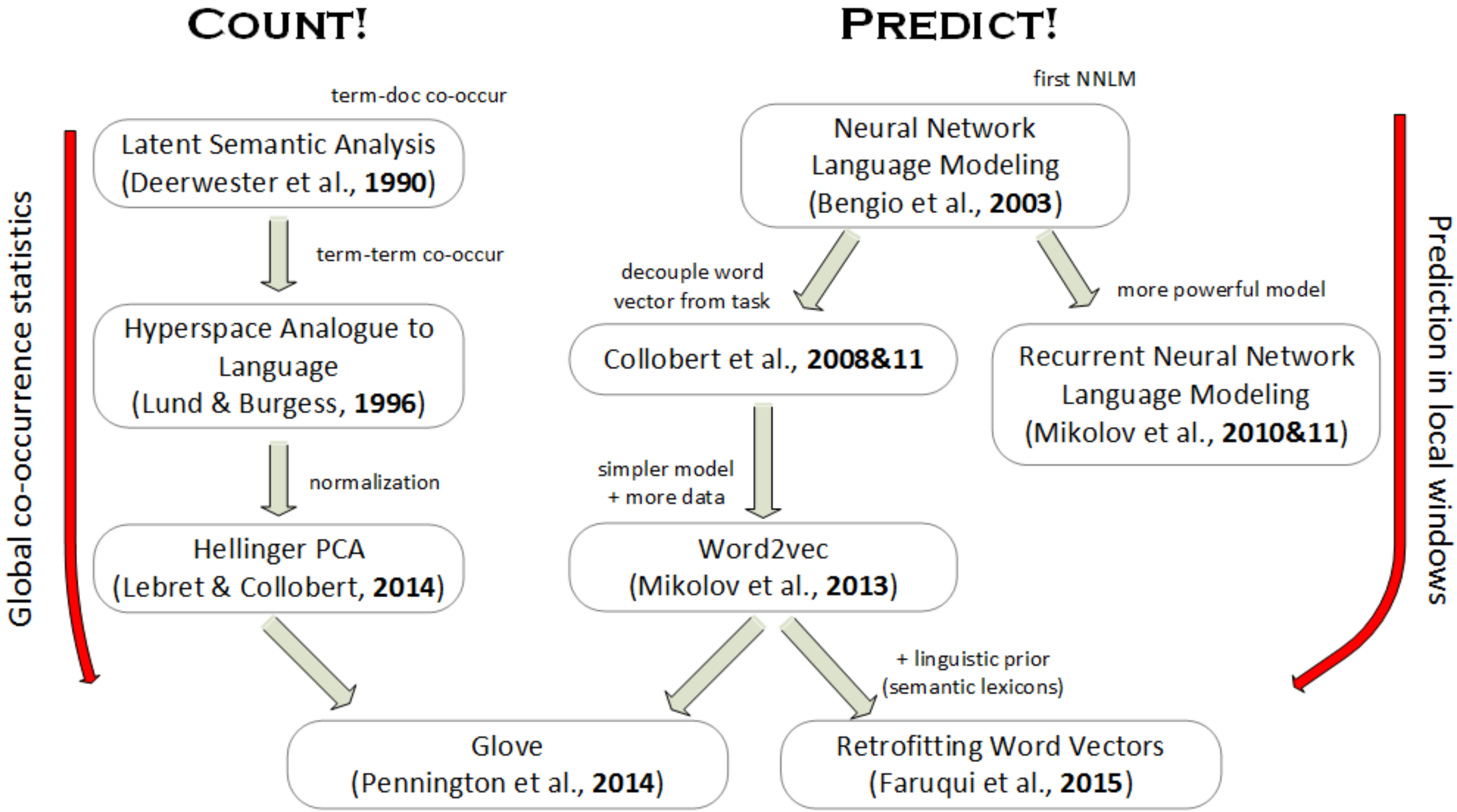
History of word embedding



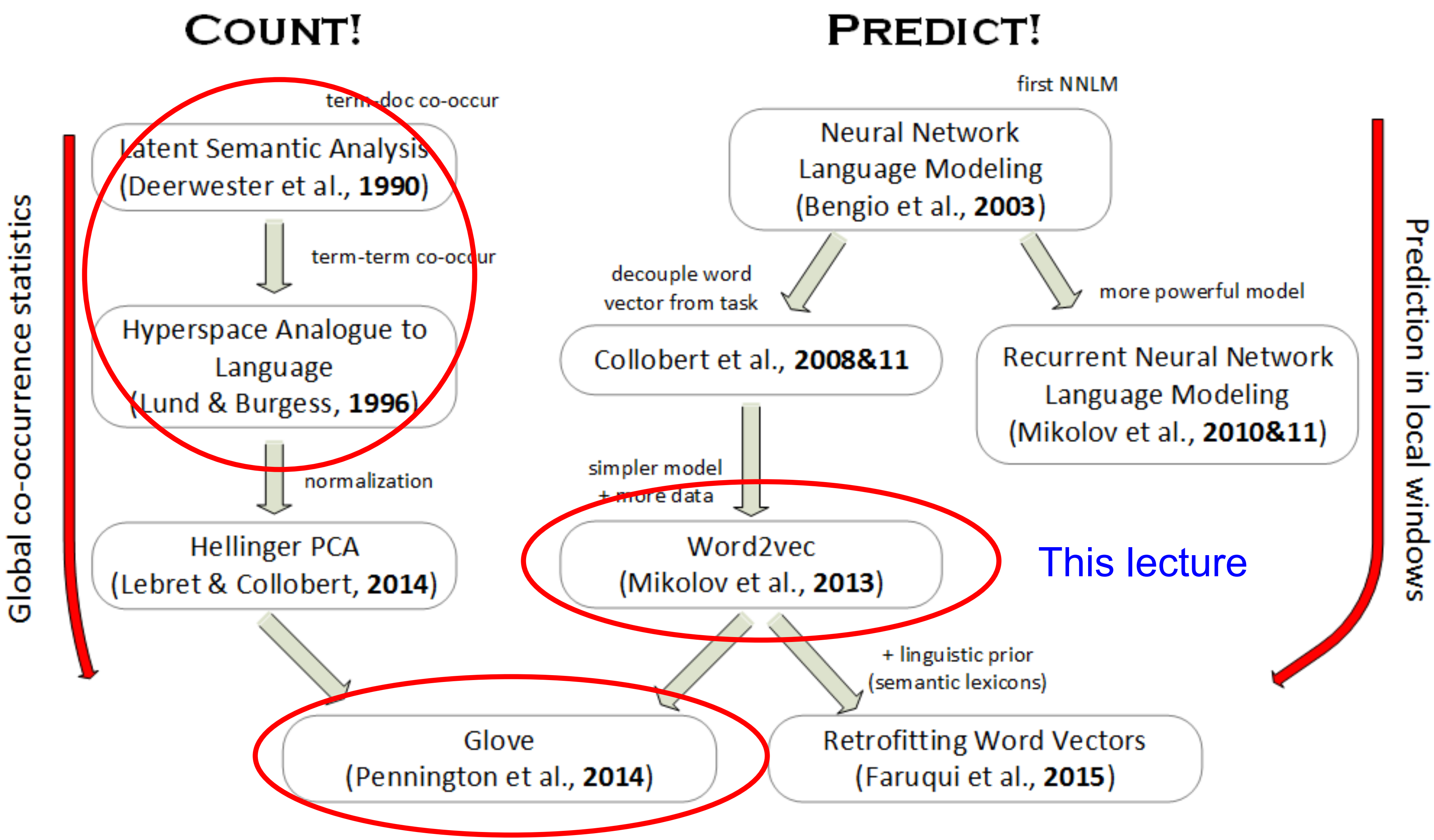
History of word embedding



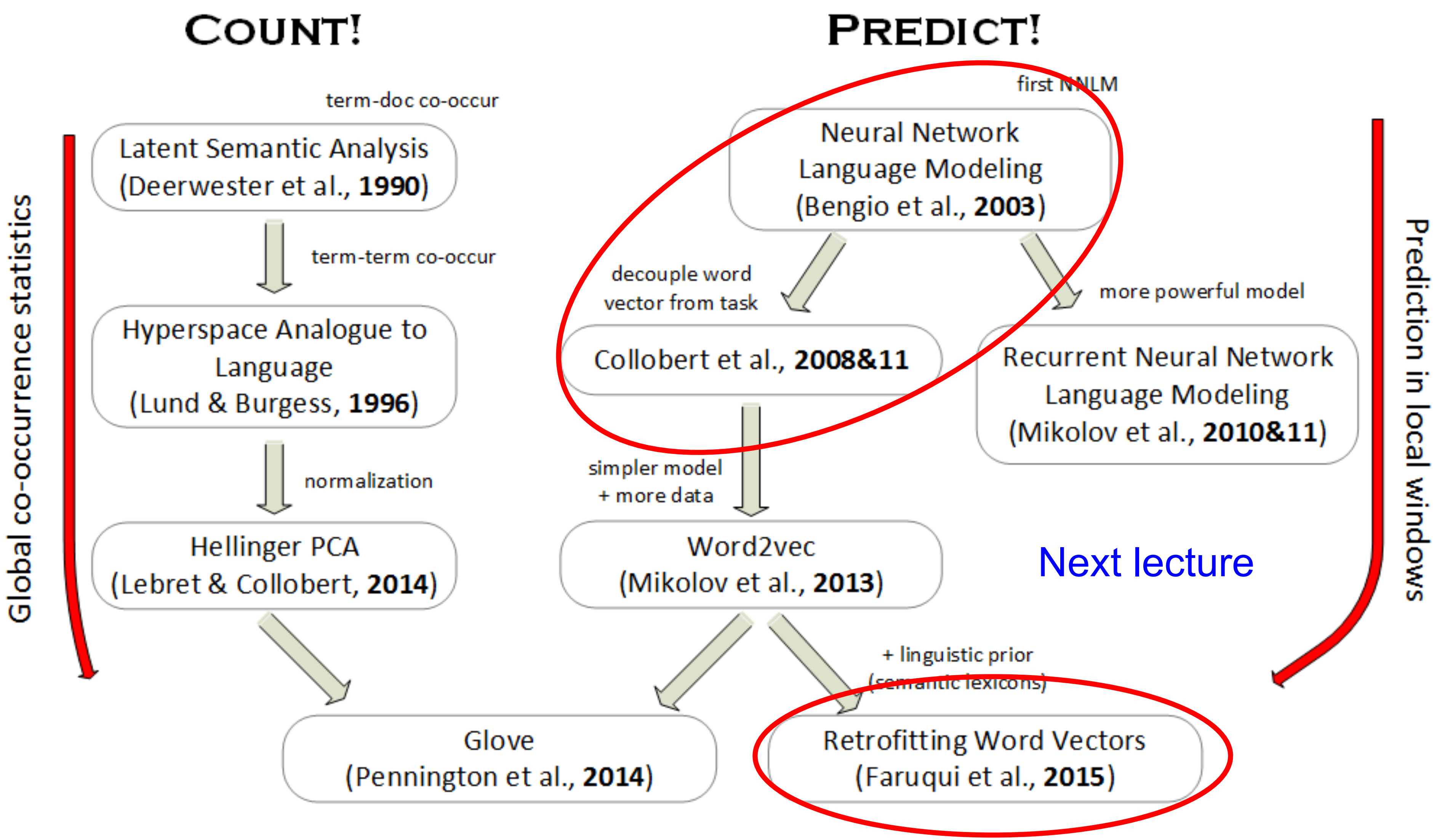
History of word embedding



History of word embedding



History of word embedding



Count-based methods: Build global context matrix X

- Choice of context: Full document vs. Local window

- Full document:
  - Context = all the words in the same doc
  - Word-doc occurrence matrix

- Local window
  - Context = words within a certain distance
  - Word-word co-occurrence matrix

Word-doc occurrence matrix

Terms	Docs 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
data	1	1	0	0	2	0	0	0	0	0	1	2	1	1	1	0	1	0	0	0
examples	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
introduction	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
mining	0	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0	0	0	0	0
network	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1
package	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

- Word-doc occurrence matrix will give general topics, e.g., all sports words will have similar entries
- Lead to *Latent Semantic Analysis*

Word-word co-occurrence matrix

window size = 2

window size = 1

I am studying computer science in UCSB .

central word

- Window allows us to capture both syntactic and semantic information →

Word-word co-occurrence matrix: toy example

- Example corpus (window size=1):
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

Problems with simple co-occurrence vectors

- Increase in size with vocabulary
- Very high dimensional: require a lot of storage
- Subsequent classification models have sparsity issues

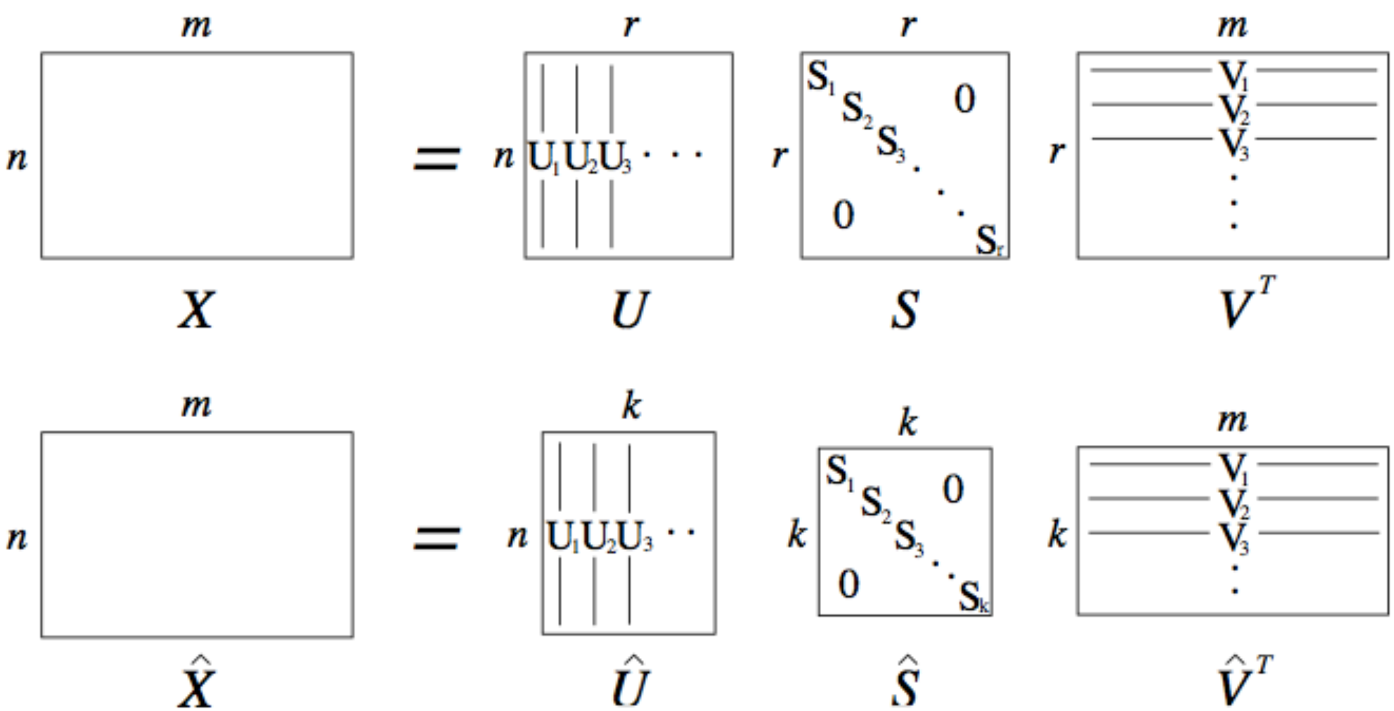
→ Models are less robust

Solution: Low-dimensional dense vectors

- 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?

Method 1: Dimensionality Reduction on X

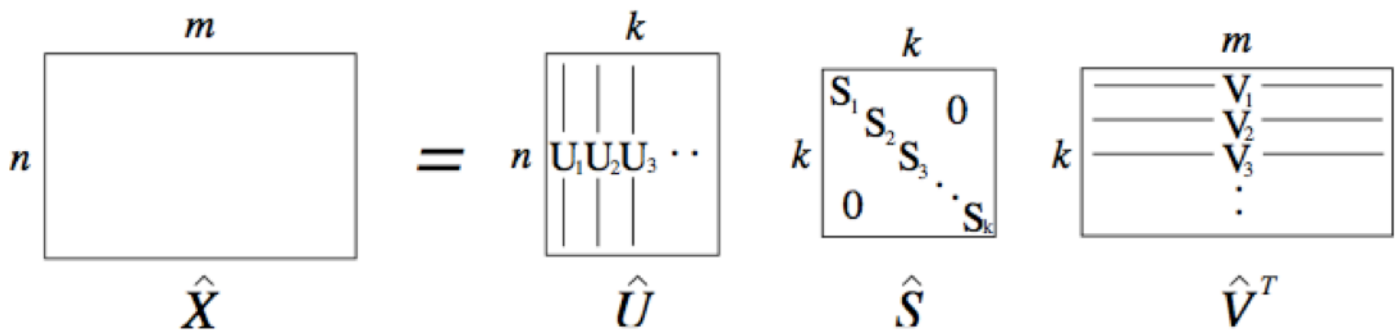
- Singular Value Decomposition (SVD) on X

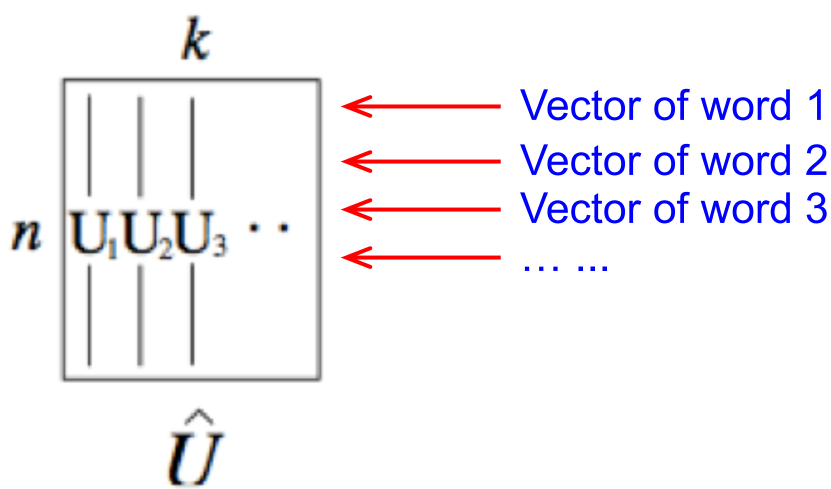


$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

Method 1: Dimensionality Reduction on X

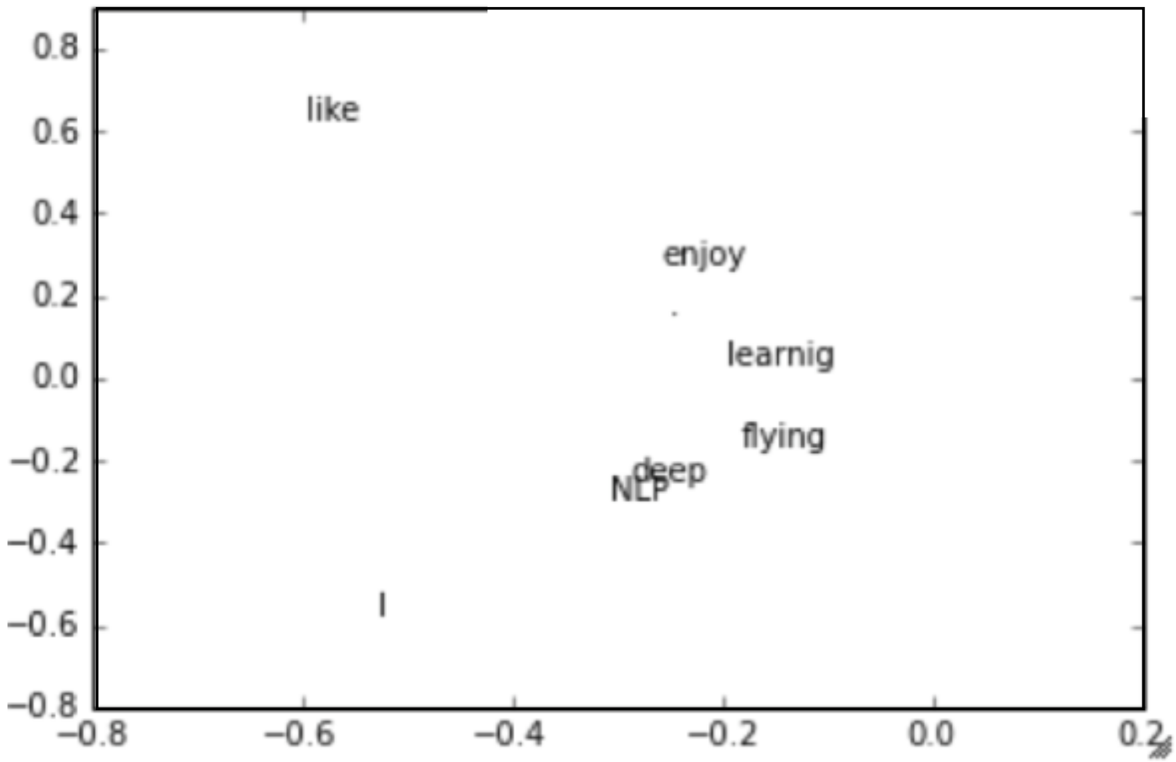
- Singular Value Decomposition (SVD) on X



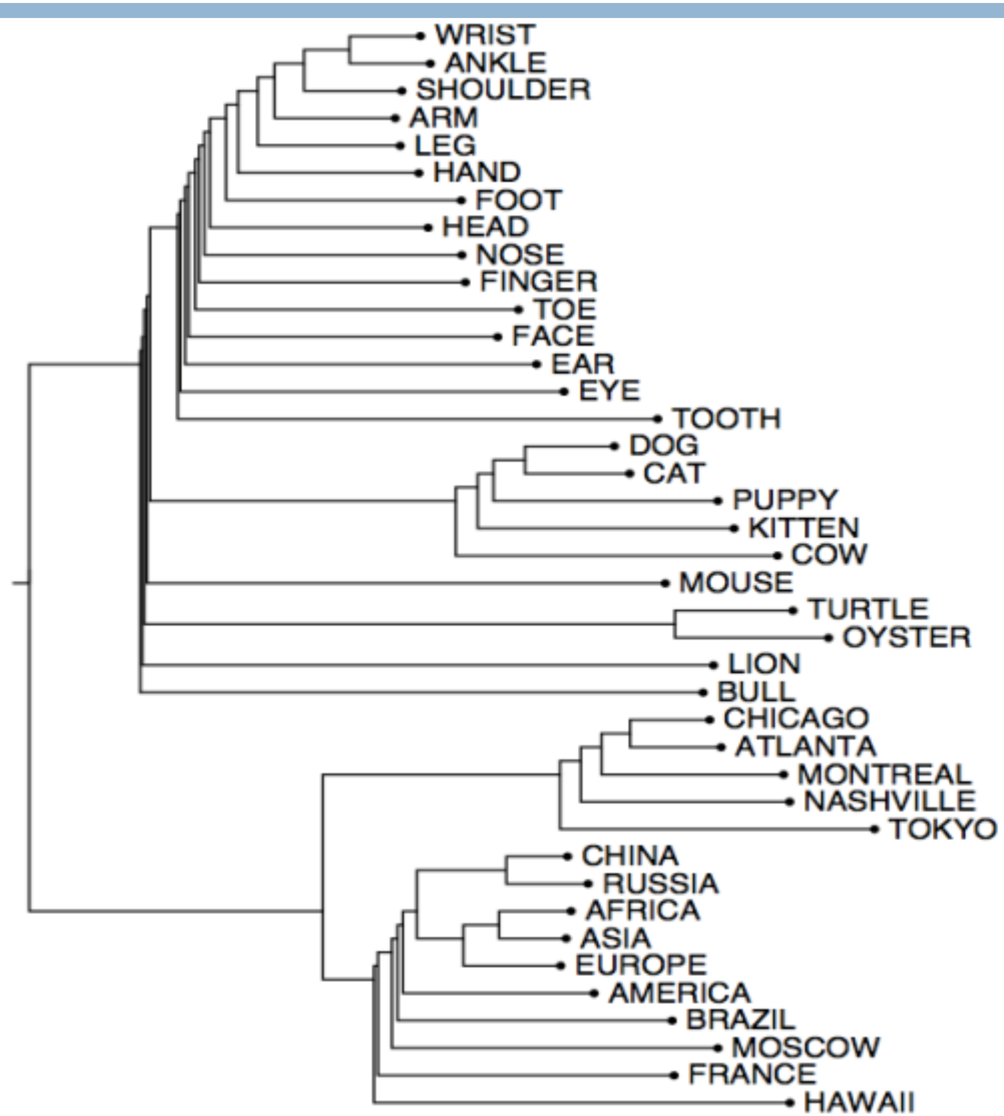


Simple SVD on X

- Corpus: I like deep learning. I like NLP. I enjoy flying.
- Print the first two columns of U corresponding to the 2 largest singular values

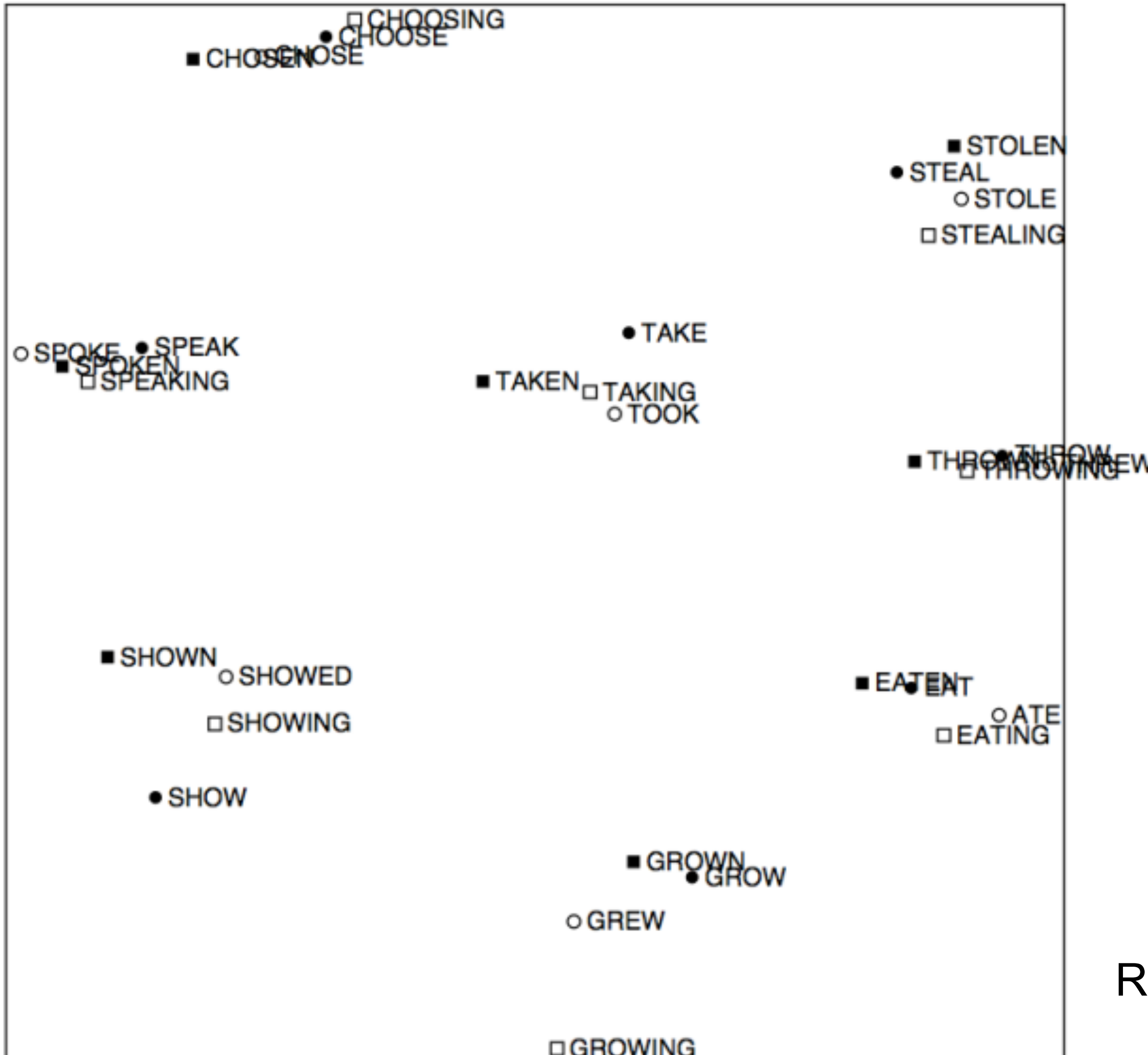


Interesting patterns emerge in the vectors



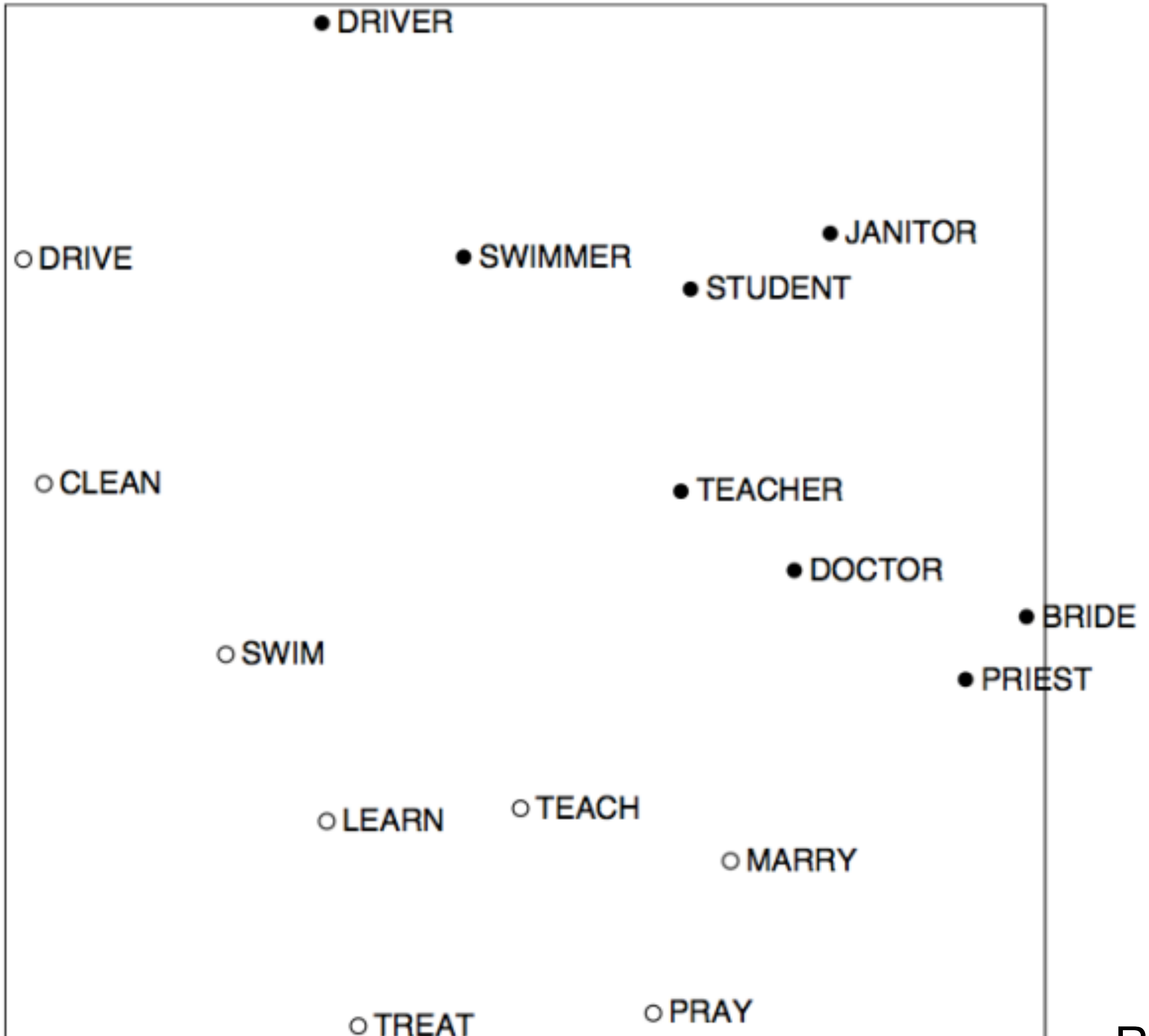
Rohde et al., 2005

Interesting patterns emerge in the vectors



Rohde et al., 2005

Interesting patterns emerge in the vectors



Rohde et al., 2005

Problems with SVD

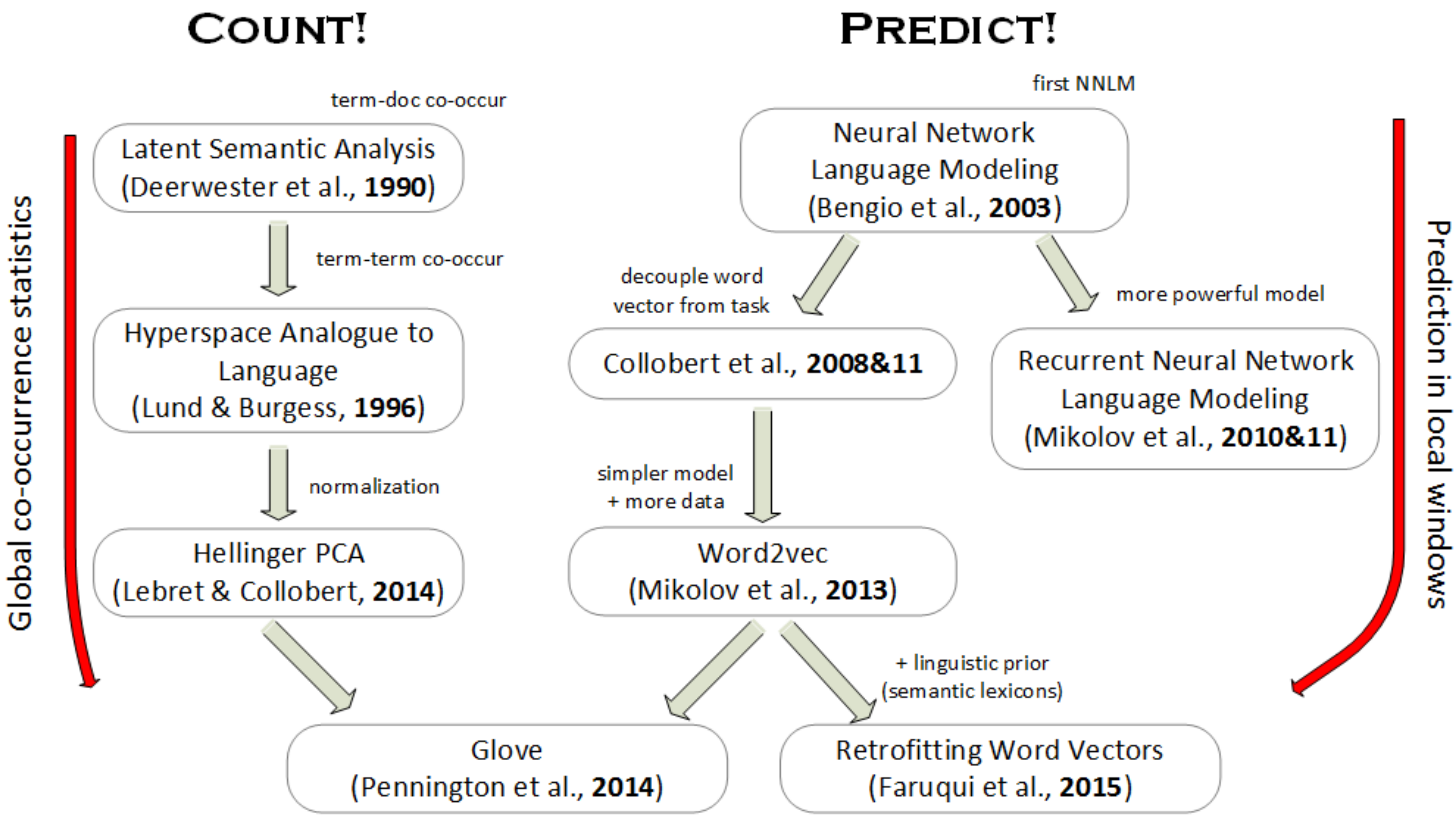
- Naive implementation: Computational cost scales quadratically for n x m matrix: $O(mn^2)$ when $n<m$

→ Bad for millions of words or documents

→ More efficient approximate solutions exist, though

- Hard to incorporate new words or documents
  - Changing a single entry has a global effect
  - Need to do it again…

Method 2: Directly learn low-dimensional word vectors



Main idea

- Instead of capturing global co-occurrence counts directly
- Sequentially scan local windows and do prediction
- Easily incorporate a new sentence/document or add a word to the vocabulary

Word2vec

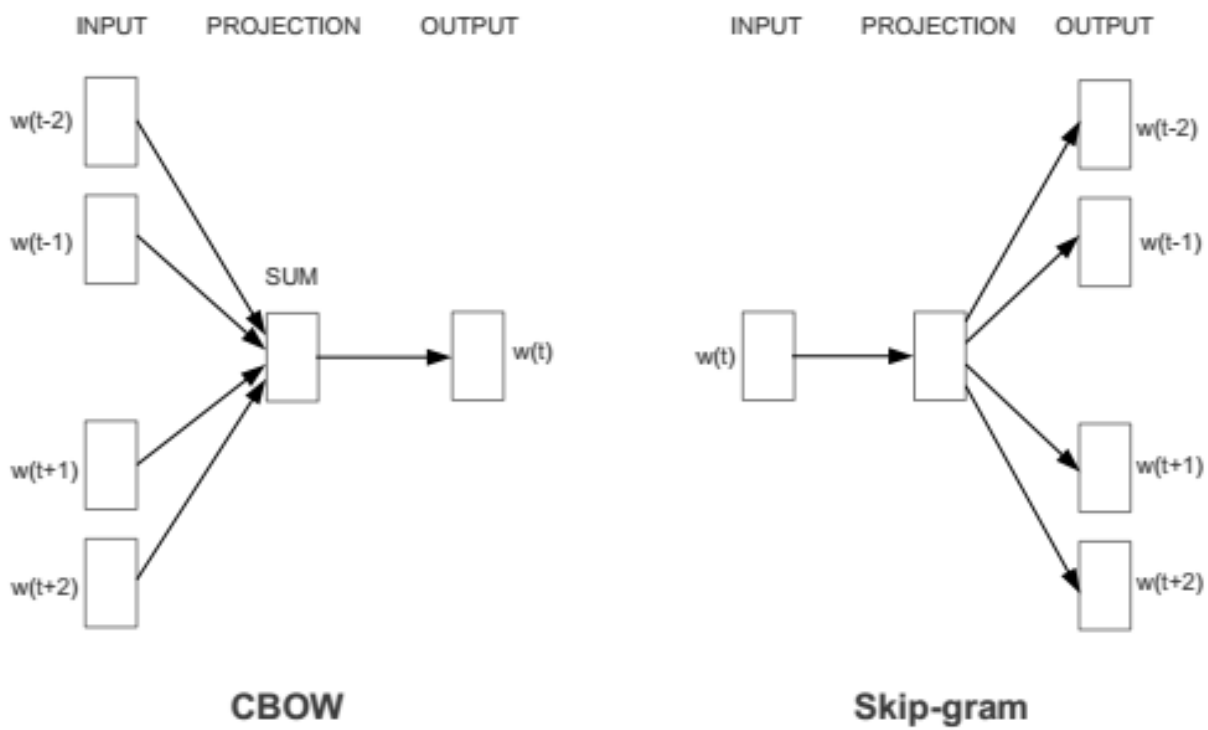
- The simplest NN-like model to learn word embedding

- Skip-gram: given the central word, predict surrounding words

- Continuous Bag-of-words (CBOW): given the surrounding words, predict the central word

$$\text{I}\ \overbrace{\text{am studying computer science in}}^{\text{window size = 2}}\ \text{UCSB .}$$

central word



Skip-gram

- Given the central word, predict surrounding words in a window of size c

- Objective function: Maximize the log probability of the surrounding words given the current central word:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{-c\leq j\leq c, j\neq 0} \log p(w_{t+j} | w_t)$$

Skip-gram

- Given the central word I, predict a surrounding word O

- Softmax: the simplest formulation for $p(O \mid I)$:

$$p(O \mid I) = \frac{\exp({v'_O}^T v_I)}{\sum_{w \in V} \exp({v'_w}^T v_I)}$$

- *v* and *v'* are the "input" and "output" vectors of words (each word has two vectors!)
- *V* is the whole vocabulary

Derivation of gradients

$$\log p(O \mid I) = v'^{T}_{O} v_I - \log\left(\sum_{w} \exp(v'^{T}_{w} v_I)\right)$$

Try to derive it by yourself!

1. Note that all *v*s are vectors
2. The chain rule is your good friend

$$\frac{\partial \log p(O \mid I)}{\partial v'_O} = v_I$$

$$\frac{\partial \log p(O \mid I)}{\partial v_I} = v'_O - \sum_{w} p(w \mid I) v'_w$$

Skip-gram naive implementation: step by step

Input: a text corpus, dimensionality k

Output: two k-dimensional vectors for each word

- Convert the corpus into a single string of words
- A single epoch: scan from the first word to the last word, for each window with central word I:
  - For each context word O, compute $\frac{\partial \log p(O \mid I)}{\partial v_I}$ and $\frac{\partial \log p(O \mid I)}{\partial v'_O}$
  - Update v_I and v'_O using stochastic gradient ascent
- Repeat the above step

Problem of the naive implementation

- With large vocabularies this objective function is not scalable and would train too slowly! → Why?

- Solutions: Approximate the normalization or

- Just sample a few negative words (not in context) to contrast with the positive word (in context)

- Will talk about them in the next lecture

Linguistic regularities in word vector space

- The resulting distributed representations of words contain surprisingly a lot of syntactic and semantic information

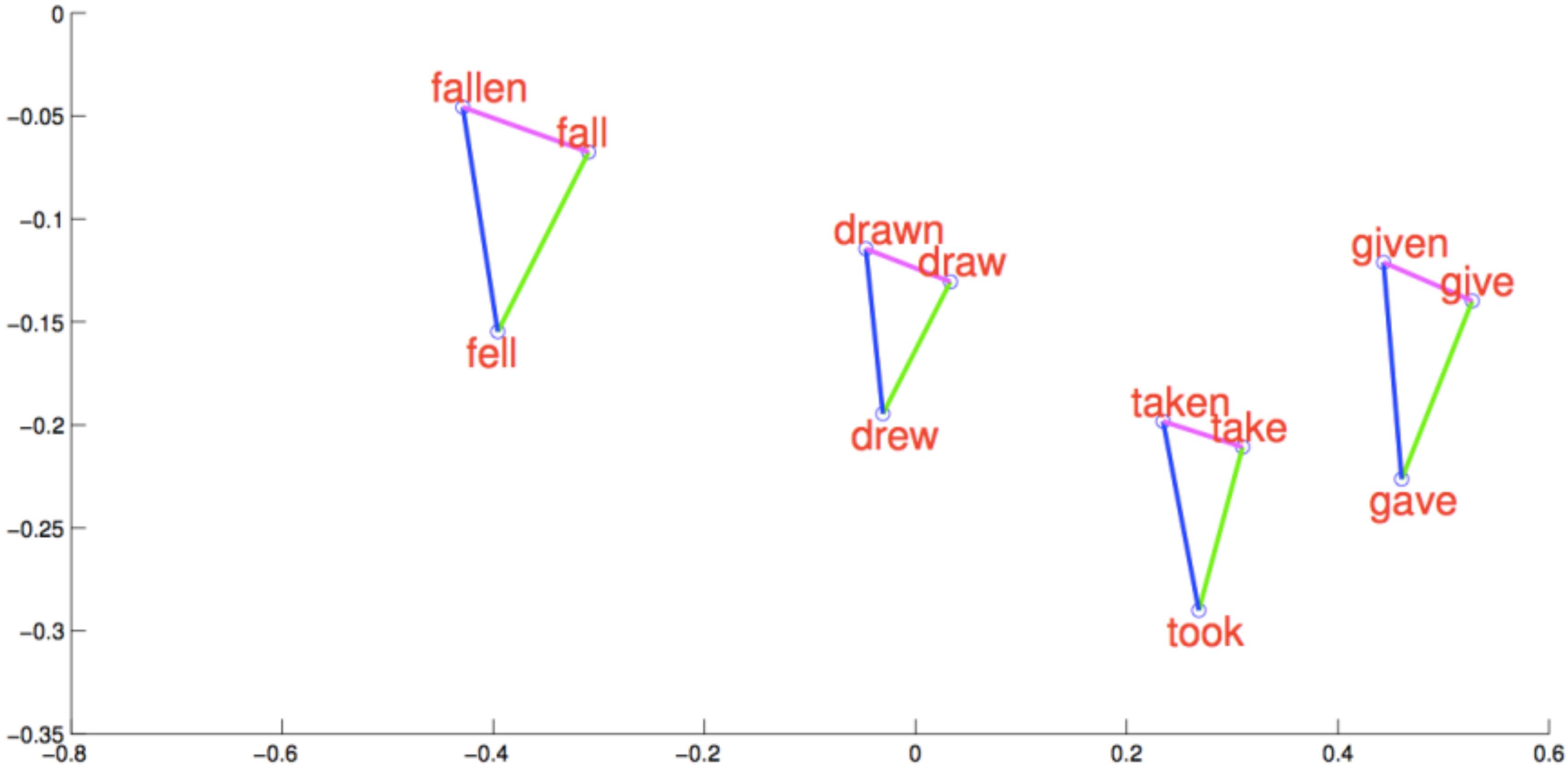
- There are multiple degrees of similarity among words:
  - **KING** is similar to **QUEEN** as **MAN** is similar to **WOMAN**
  - **KING** is similar to **KINGS** as **MAN** is similar to **MEN**

- Simple vector operations with the word vectors provide very intuitive results
  - $V_{KING} - V_{QUEEN} \approx V_{MAN} - V_{WOMAN}$
  - $V_{KING} - V_{KINGS} \approx V_{MAN} - V_{MEN}$

Linguistic regularities in word vector space

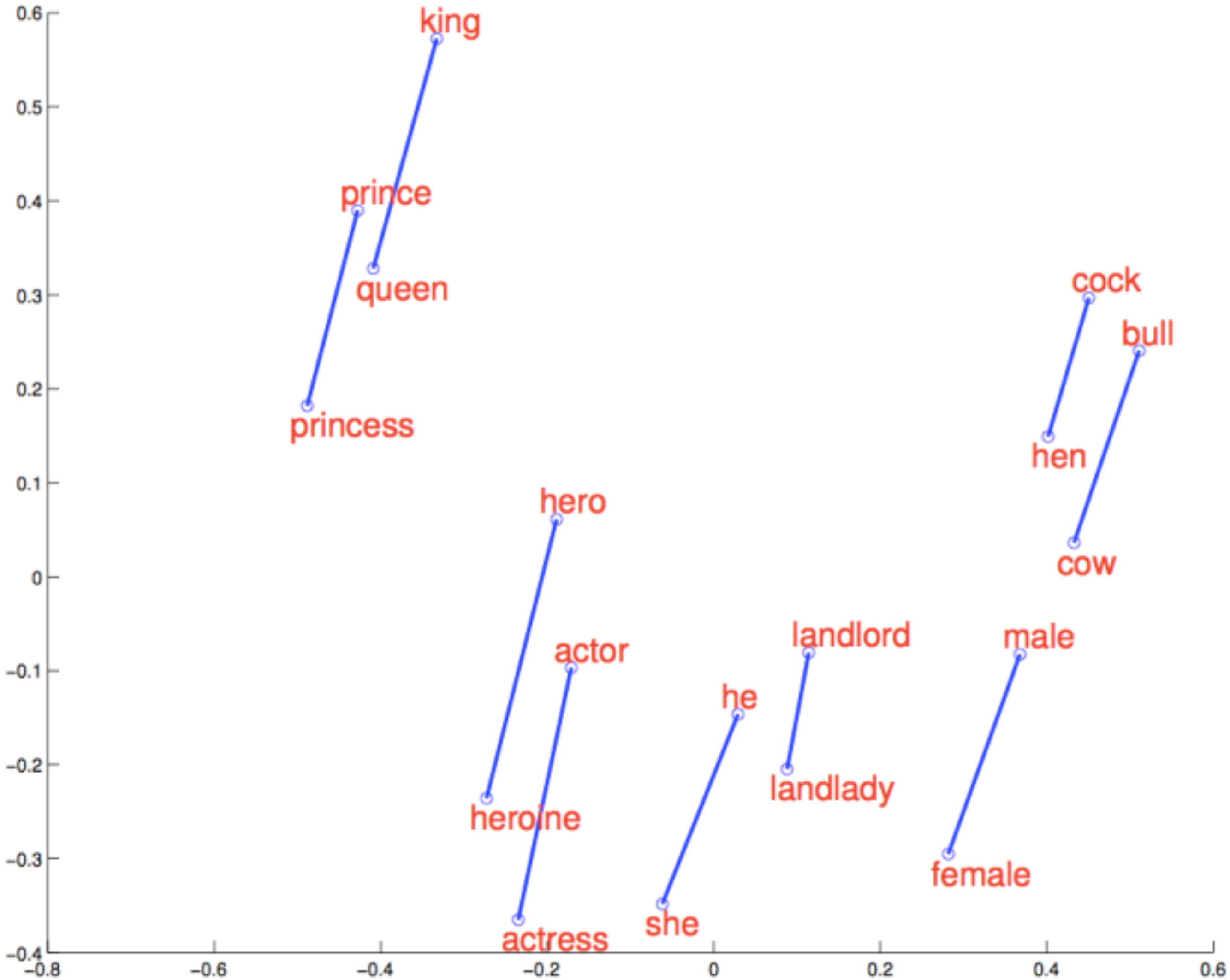
Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Visualization of regularities in word vector space



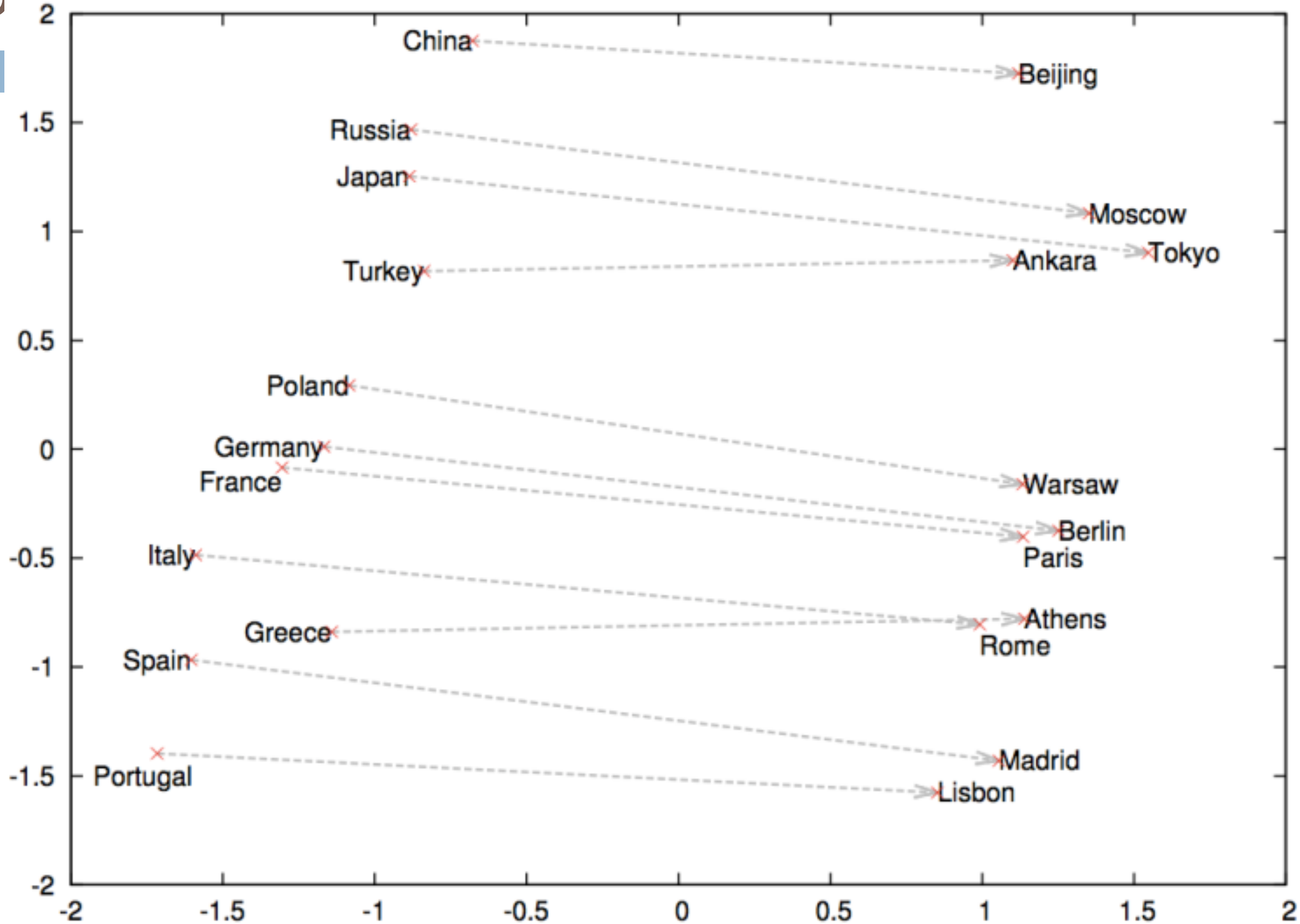
generated by PCA

Visualization of regularities in word vector space



generated by PCA

Visualization of regularities in word vector space



generated by PCA

Count based vs. prediction based

Count

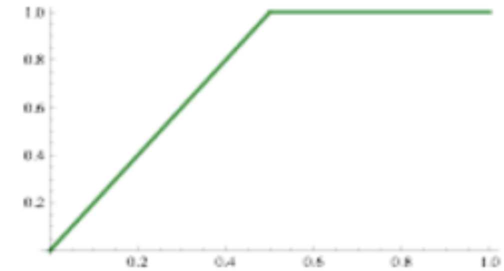
- Efficient usage of global statistics
- Primarily used to capture word similarity

Prediction

- Inefficient usage of global statistics
- Improved performance on other tasks
- Can capture richer relations between words

Combining the two worlds: GloVe (EMNLP'14)

$$J = \frac{1}{2}\sum_{ij} f(P_{ij})\left(w_i \cdot \tilde{w}_j - \log P_{ij}\right)^2 \qquad f \sim$$



- P_{ij} is the number of co-occurrences of word i and word j
- f is just a weighting function
- Fast training: $\sim O(|C|^{0.8})$, $|C|$ is the corpus size
- Scalable to huge corpora (840 billion words)

Glove results

Nearest words to
frog:

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

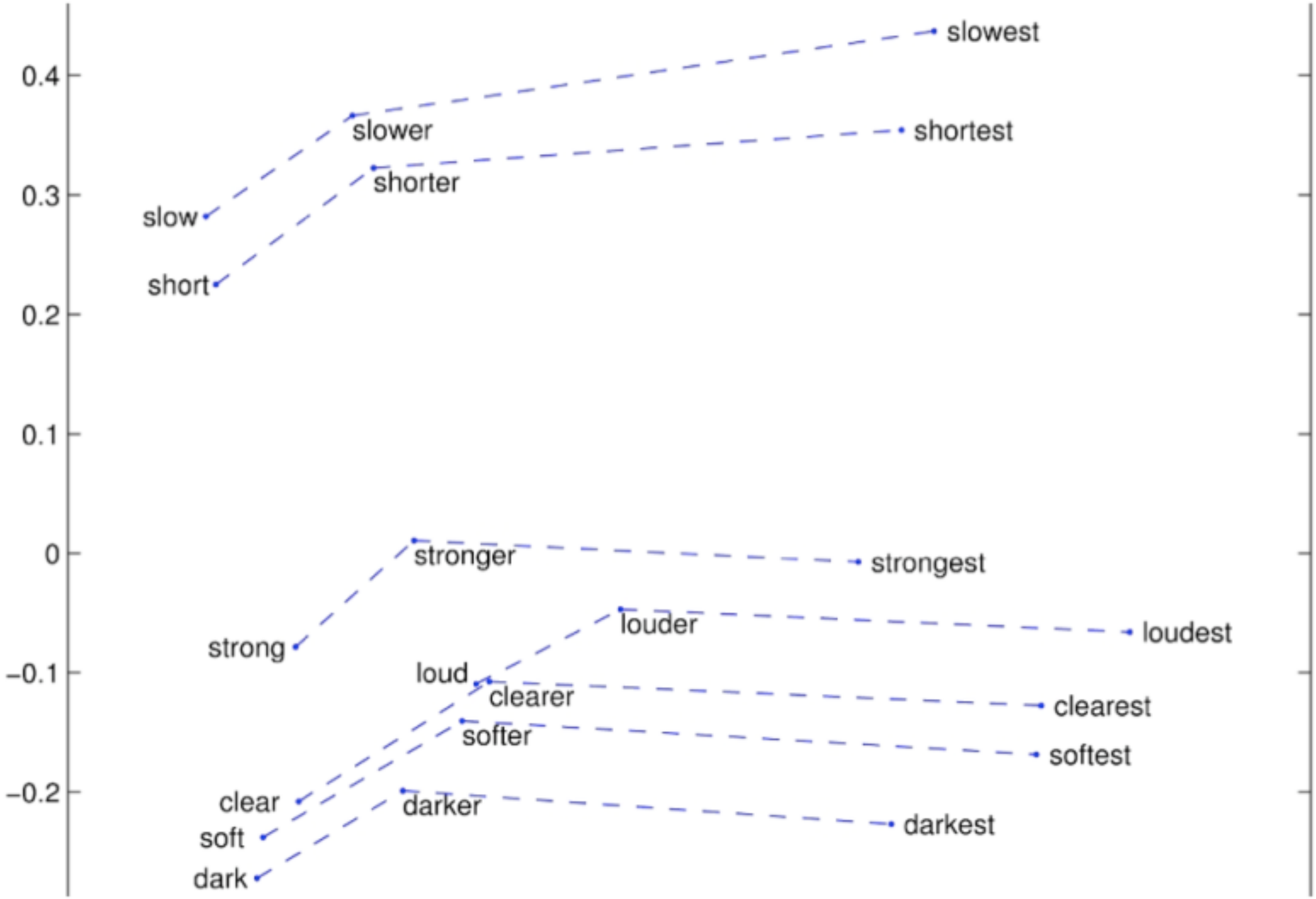
litoria

leptodactylidae

rana

eleutherodactylus

Glove results



Resources

- Word2vec: https://code.google.com/p/word2vec/
  - including codes, training/testing sets and pre-trained vectors

- Glove: http://nlp.stanford.edu/projects/glove/
  - including codes, training/testing sets and pre-trained vectors

- Dimensionality reduction:
  - Tapkee for C++: http://jmlr.org/papers/v14/lisitsyn13a.html
  - Scikit-learn for Python: http://scikit-learn.org/stable/