CS290D – Advanced Data Mining

Instructor: Xifeng Yan
Computer Science
University of California at Santa Barbara

Word Embeddings

Lecturer: Yu Su
Computer Science
University of California at Santa Barbara

Source of slides

- Richard Socher's course in Stanford
- Thomas Mikolov's invited talk at the Deep Learning workshop in NIPS'13

How to let a computer understand meaning?

A cat sits on a mat.

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

How to let a computer understand meaning?

- Symbolic/discrete solution: use a taxonomy 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 discrete representation

- Great as resource but missing nuances
  - e.g. **synonyms**: adept, expert, good, practiced, proficient, skillful?
- Missing new words (hard to keep up to date):
  wicked, badass, nifty, crack, ace, wizard, ninjia
- Subjective, sometimes hard to reach agreement
- Requires human labor to create and adapt
- Hard to compute accurate word similarity →

Problems with this discrete representation

- The vast majority of rule-based and statistical NLP work regards words as atomic symbols
- In vector space terms, this is a vector with one 1 and a lot of zeroes

[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)

- We call this a “one-hot” representation. Its problem:

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

Distributional 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"

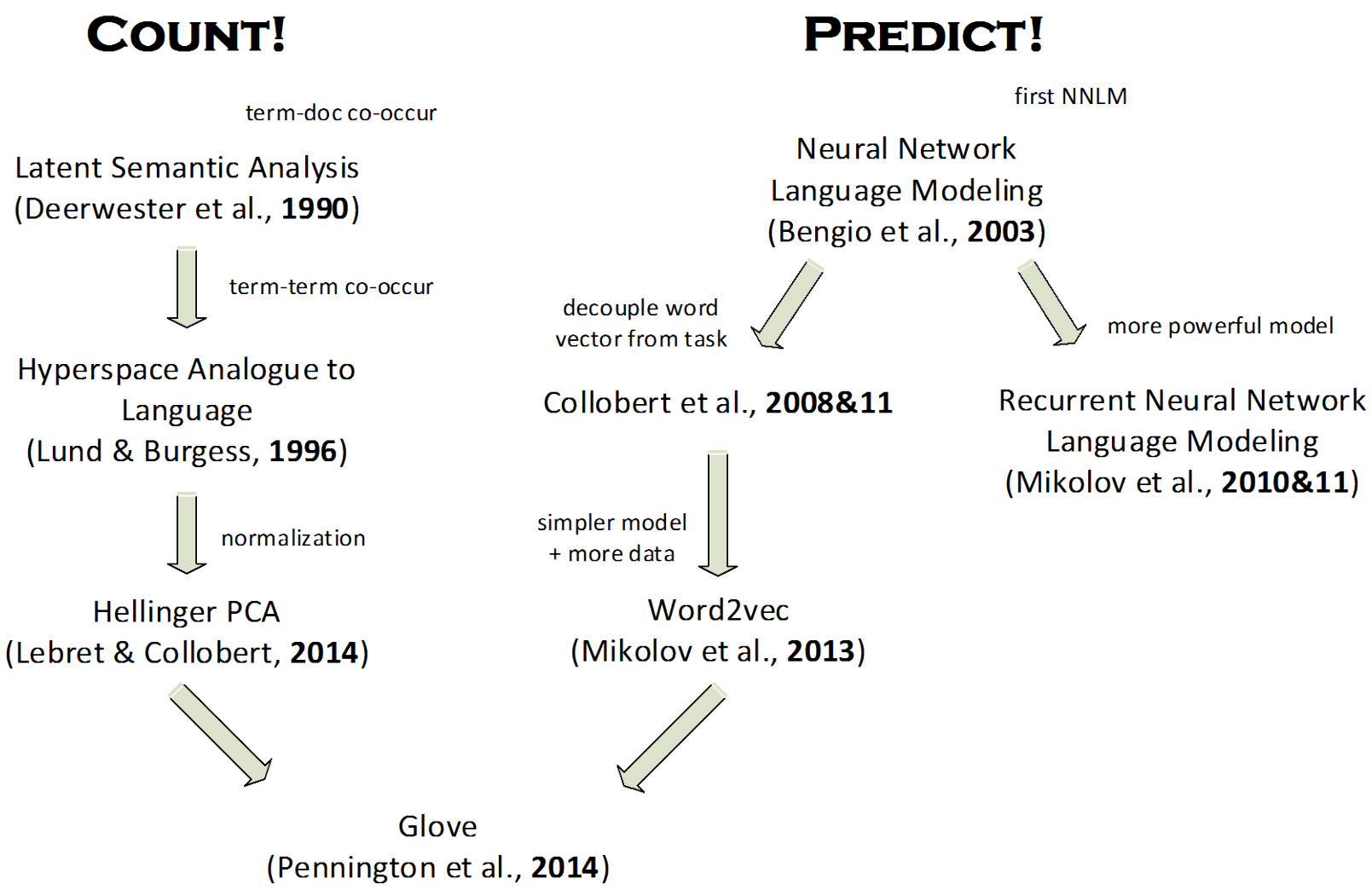
(J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical

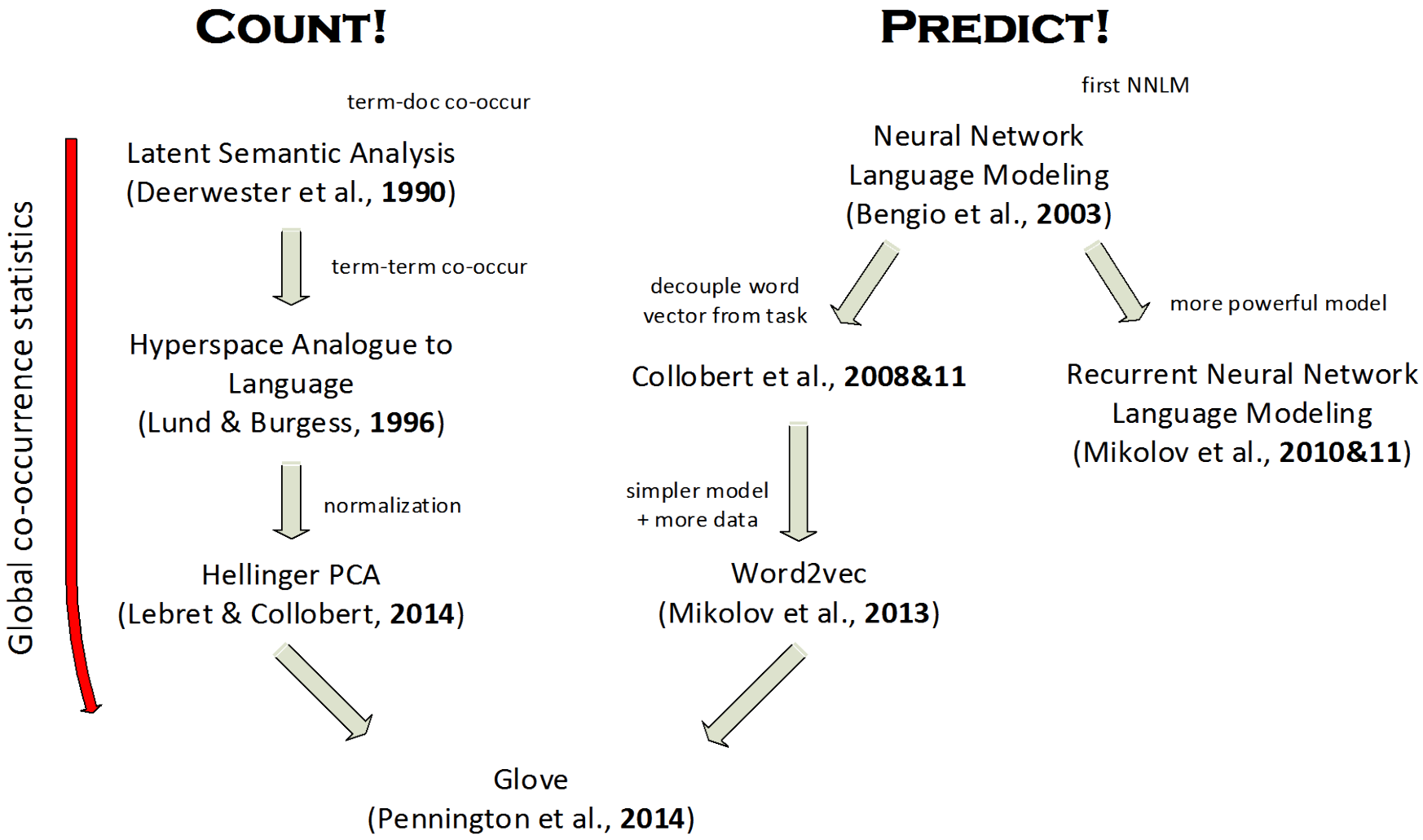
government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

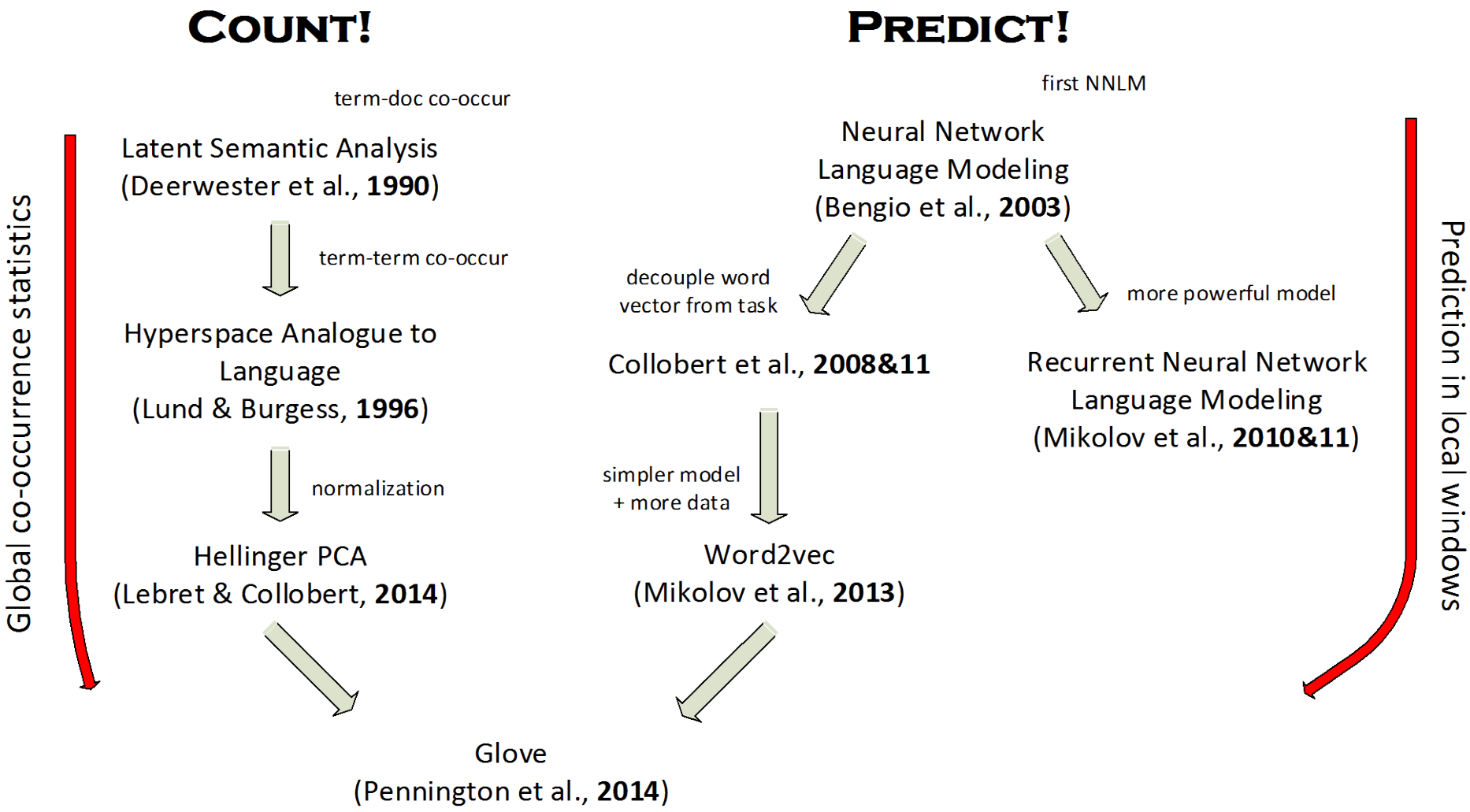
History of word embeddings



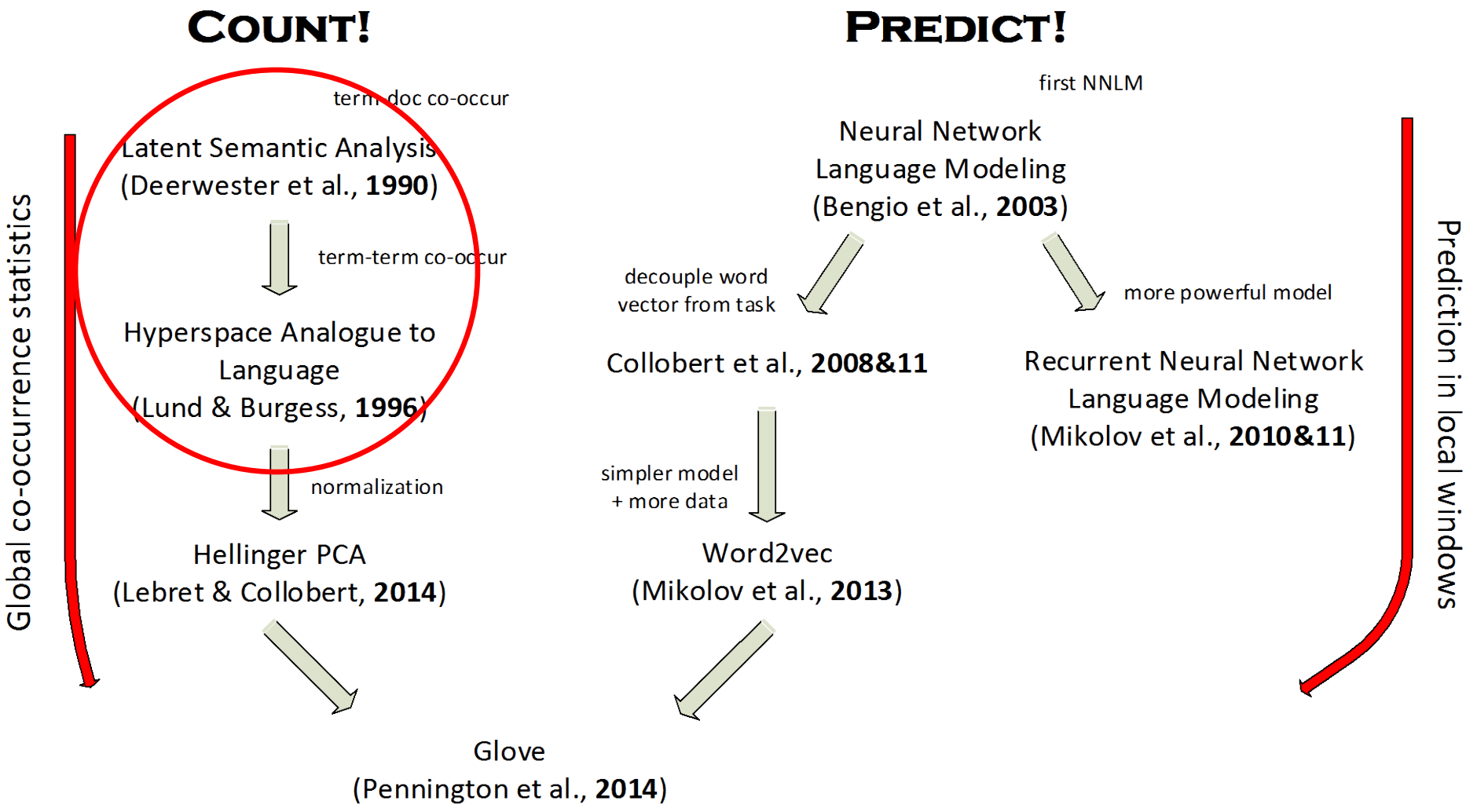
History of word embeddings



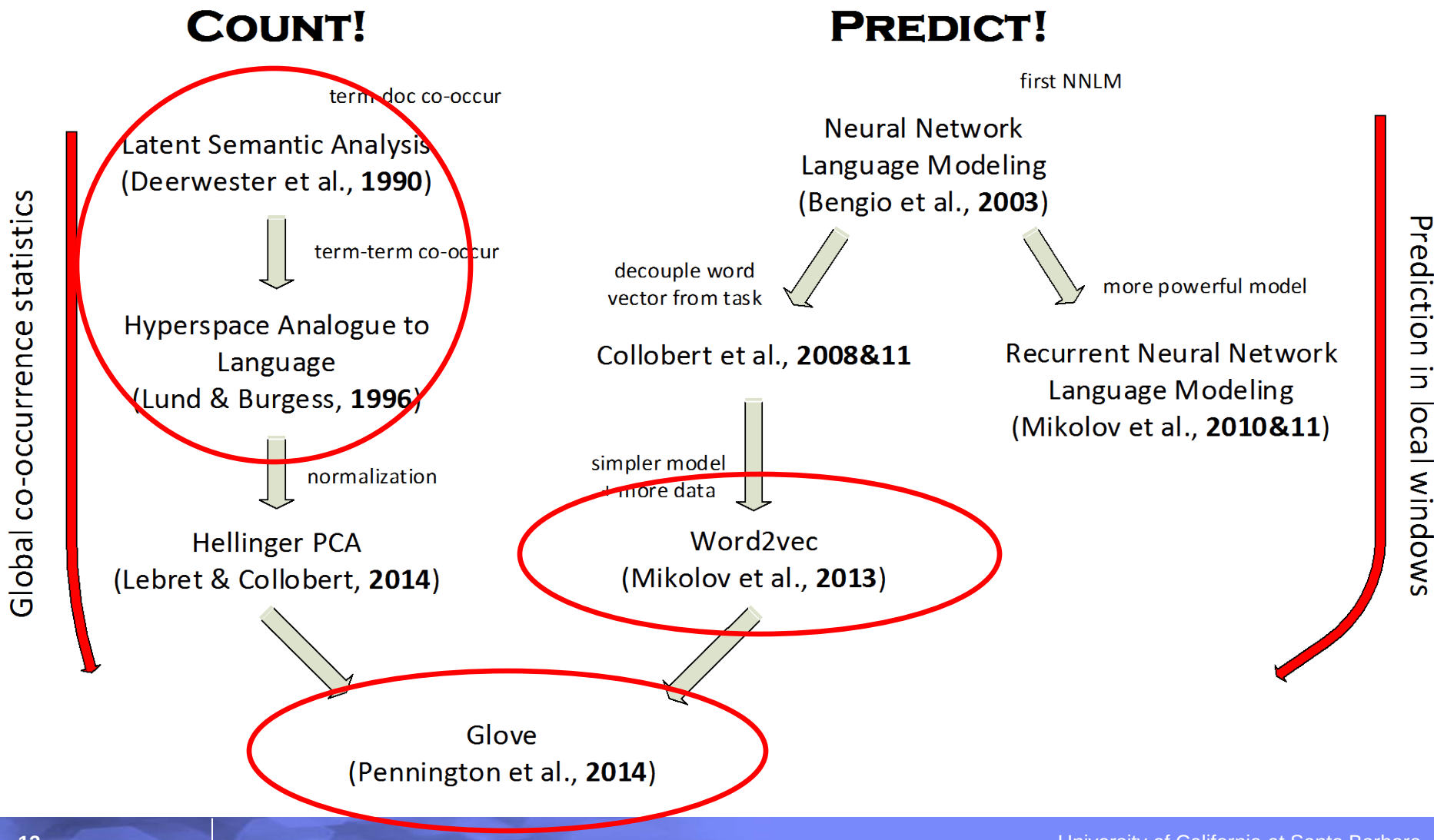
History of word embeddings



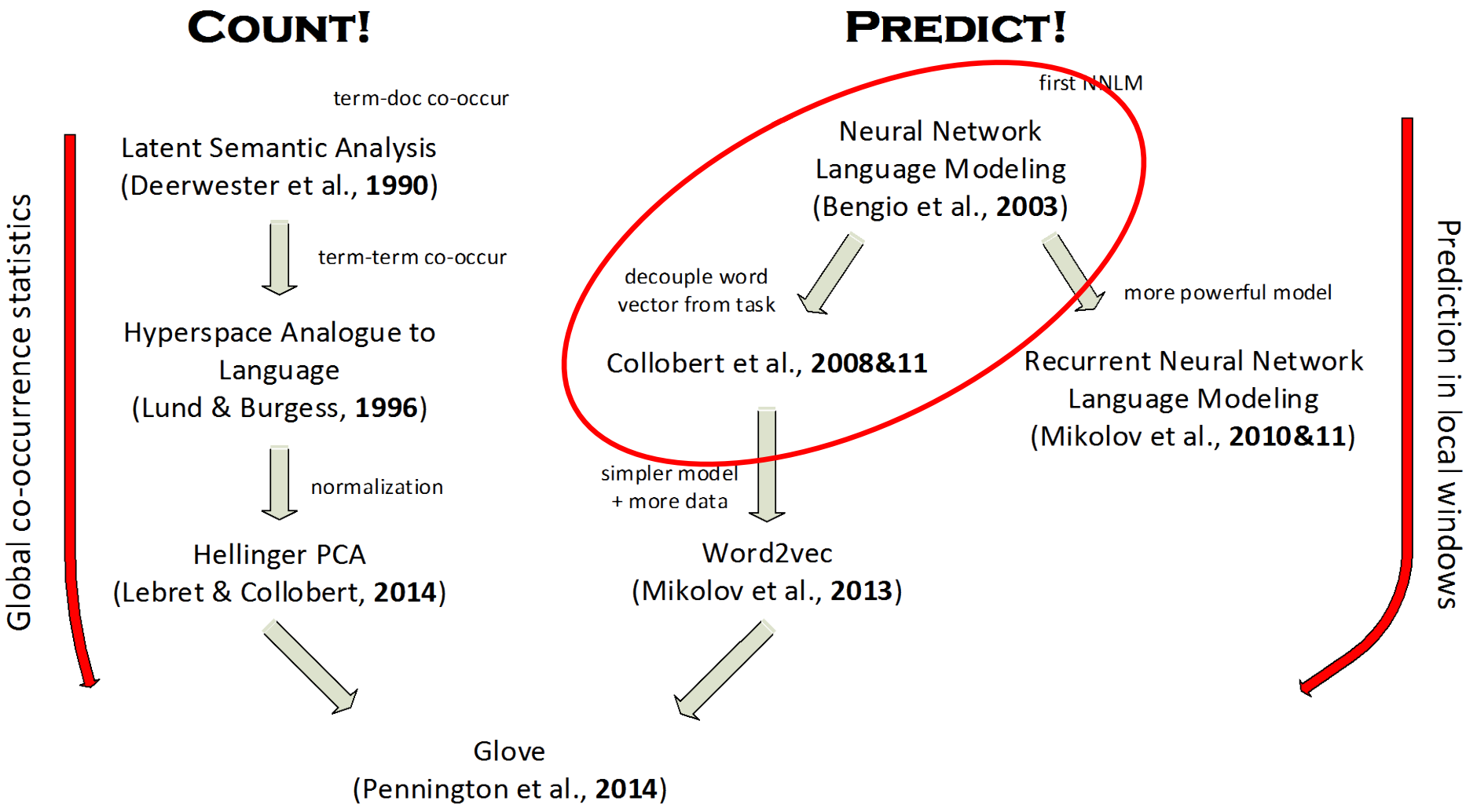
History of word embeddings



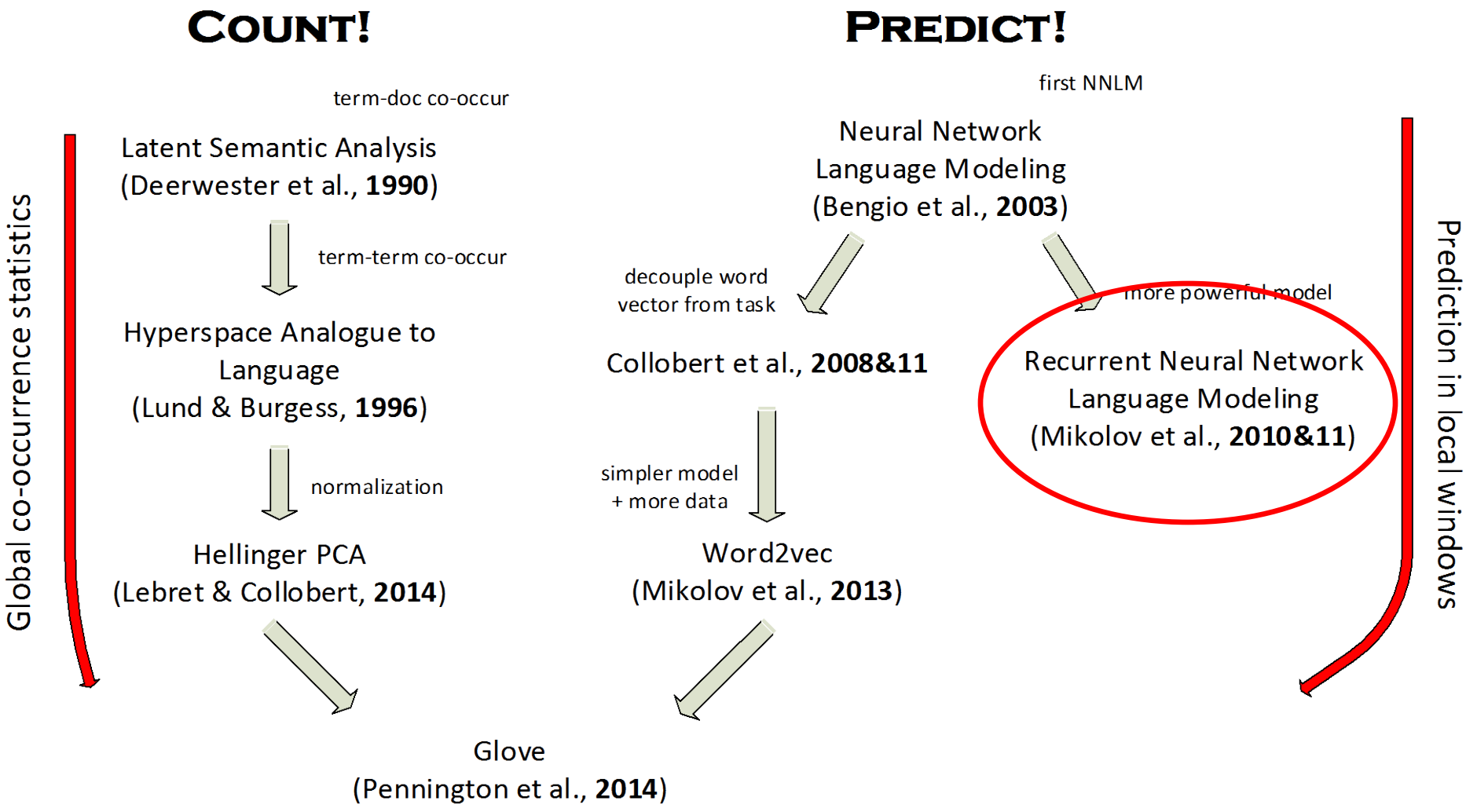
History of word embeddings



History of word embeddings



History of word embeddings



How to make neighbors represent words?

- Answer: With a co-occurrence matrix X
- 2 options: full document (term-doc) vs window (term-term)
- Word-doc co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to *Latent Semantic Analysis*
- Window allows us to capture both syntactic (POS) and semantic information →

Window based co-occurrence matrix: toy example

- Window length 1 (typically 5-10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

Window based co-occurrence matrix: toy example

- Example corpus:
  - 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

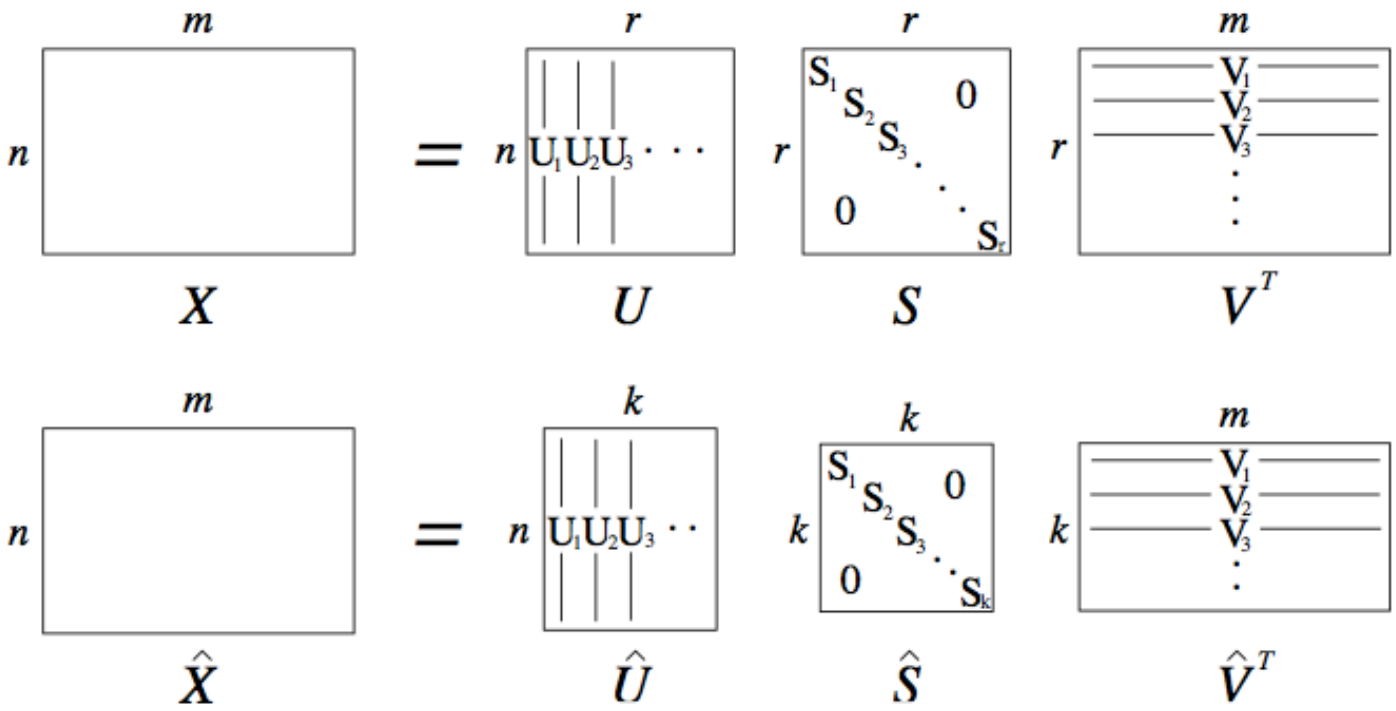
→ Models are less robust

Solution: Low dimensional 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

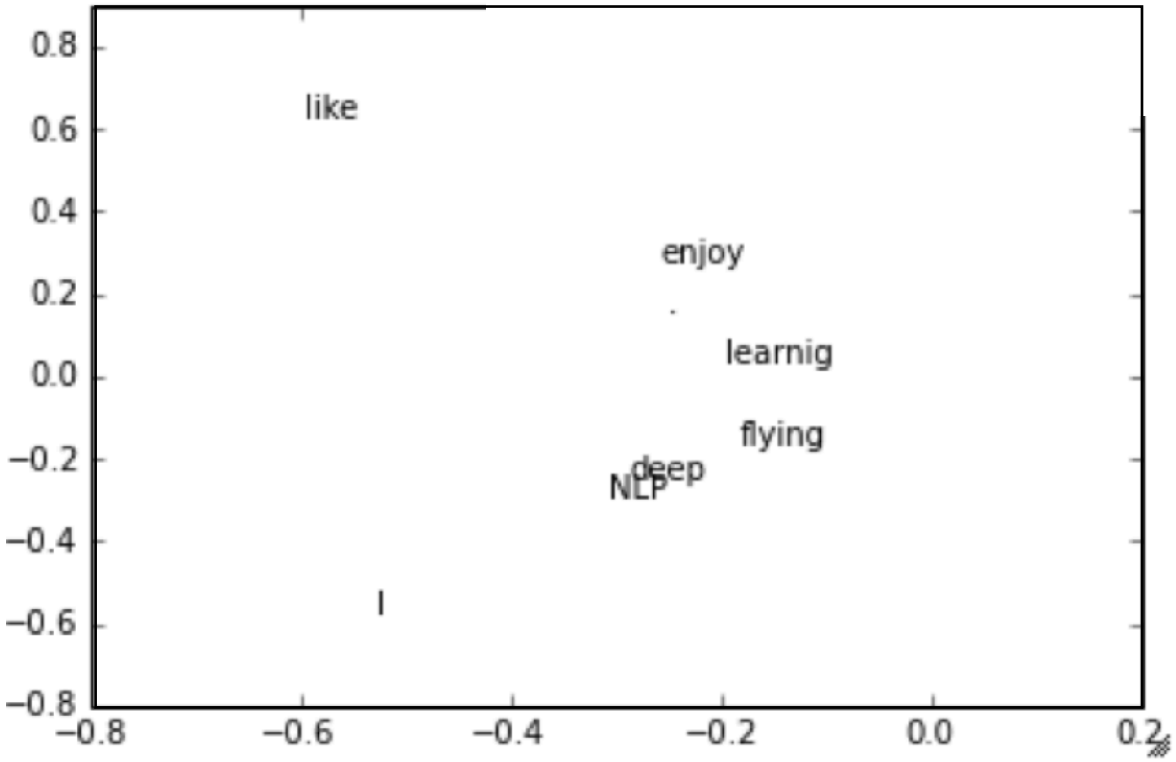
- Singular Value Decomposition (SVD) on X



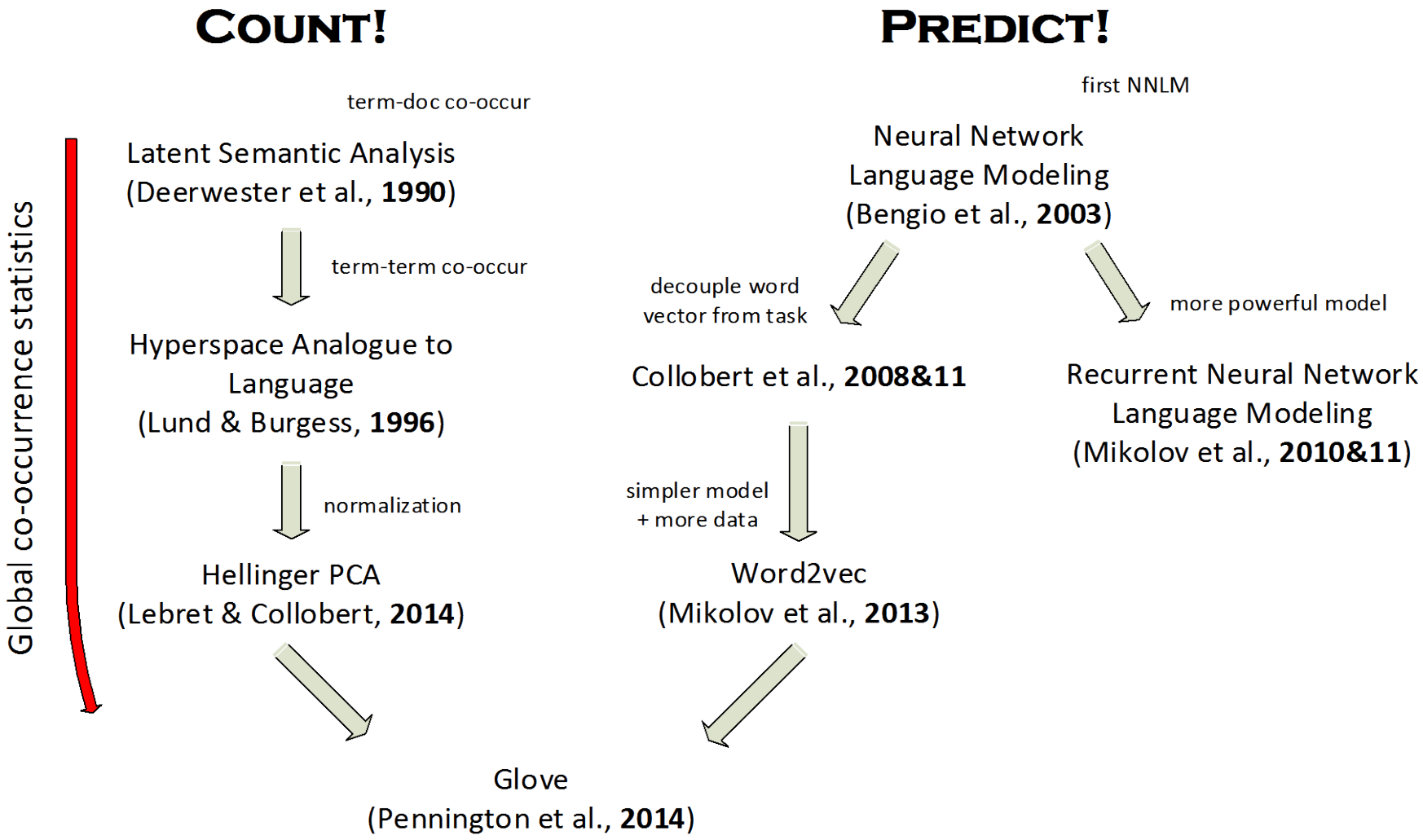
$\hat{X}$ is the best rank k approximation to X, in terms of least squares.

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 biggest singular values



History of word embeddings

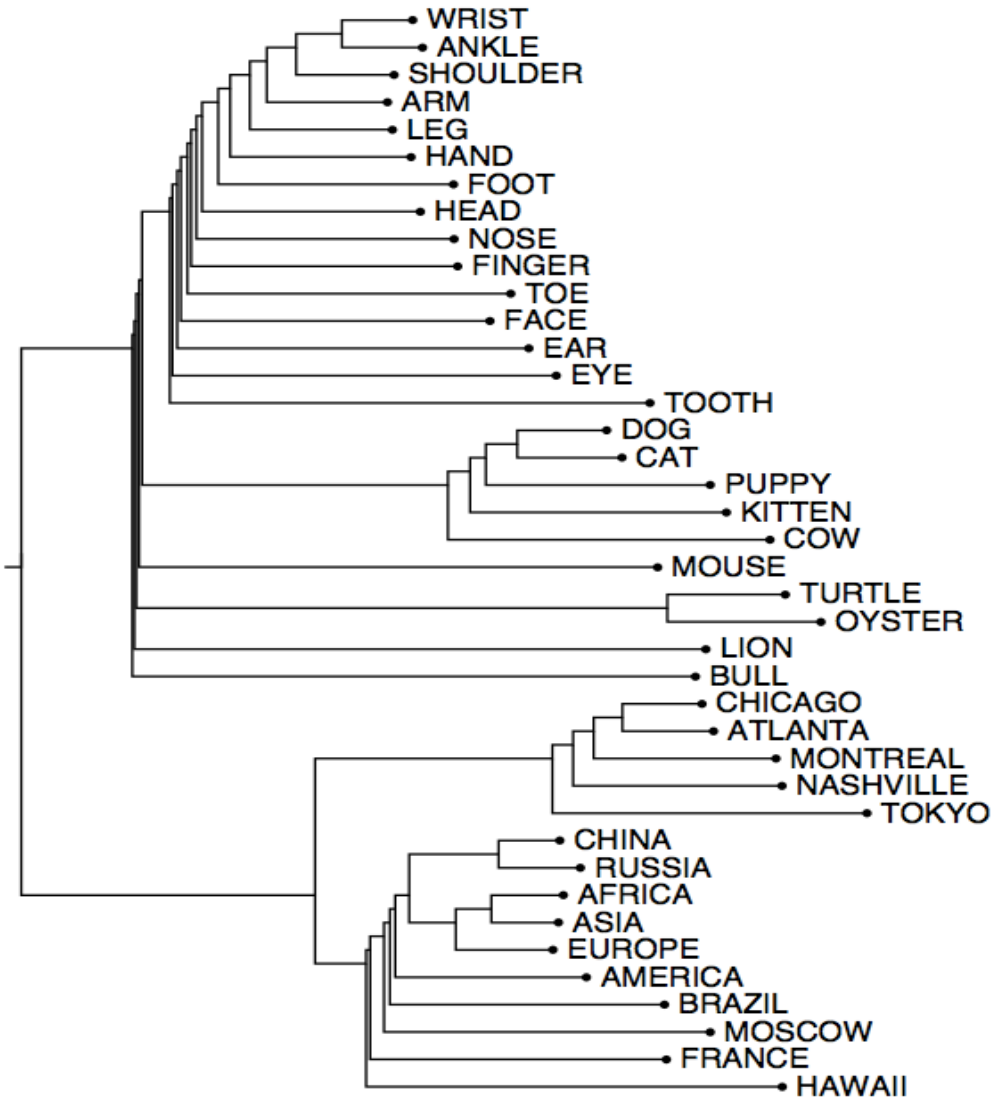


Word meaning is defined in terms of vectors

- A word is represented as a dense vector

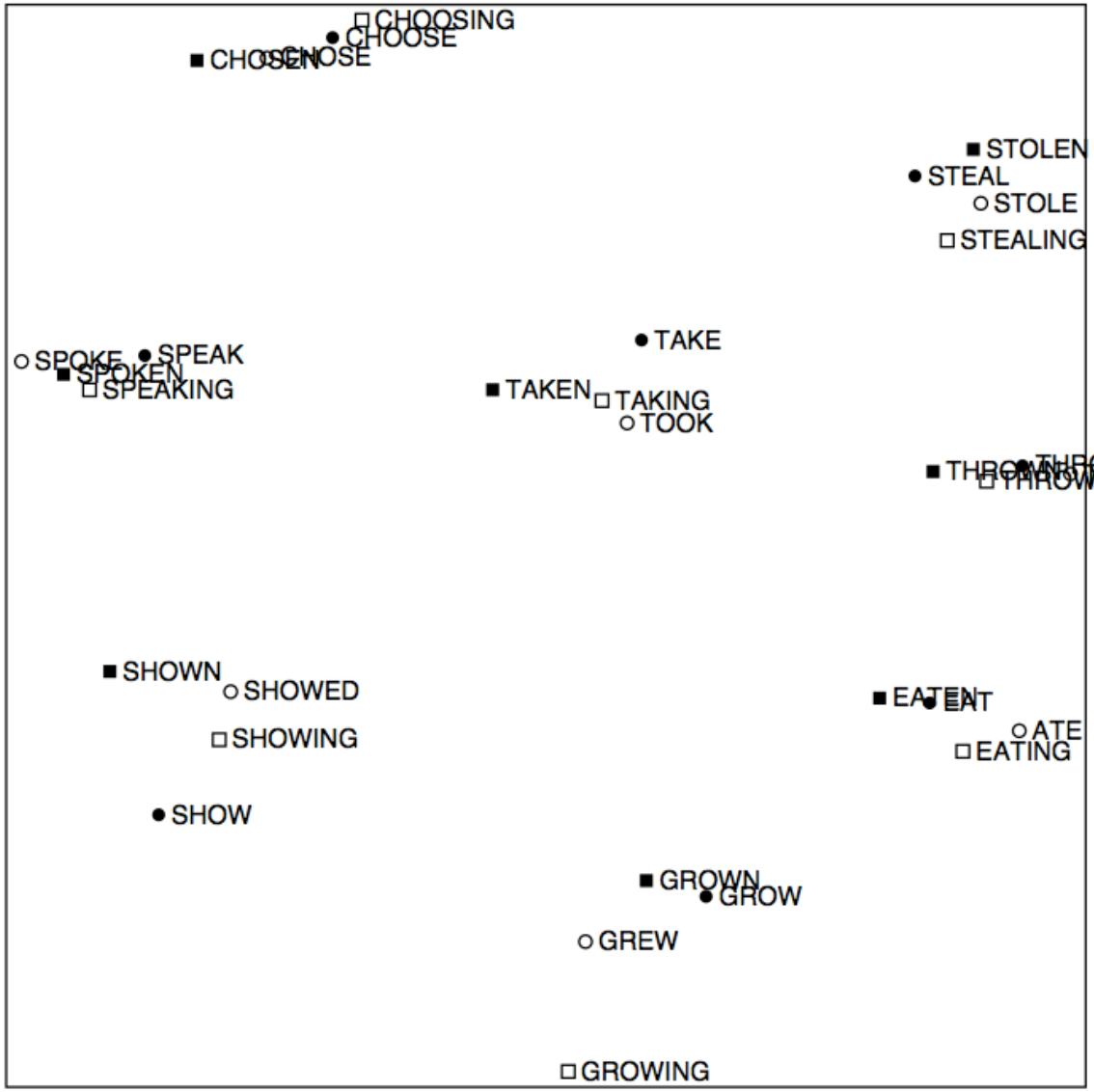
$$\textit{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Interesting semantic patters emerge in the vectors



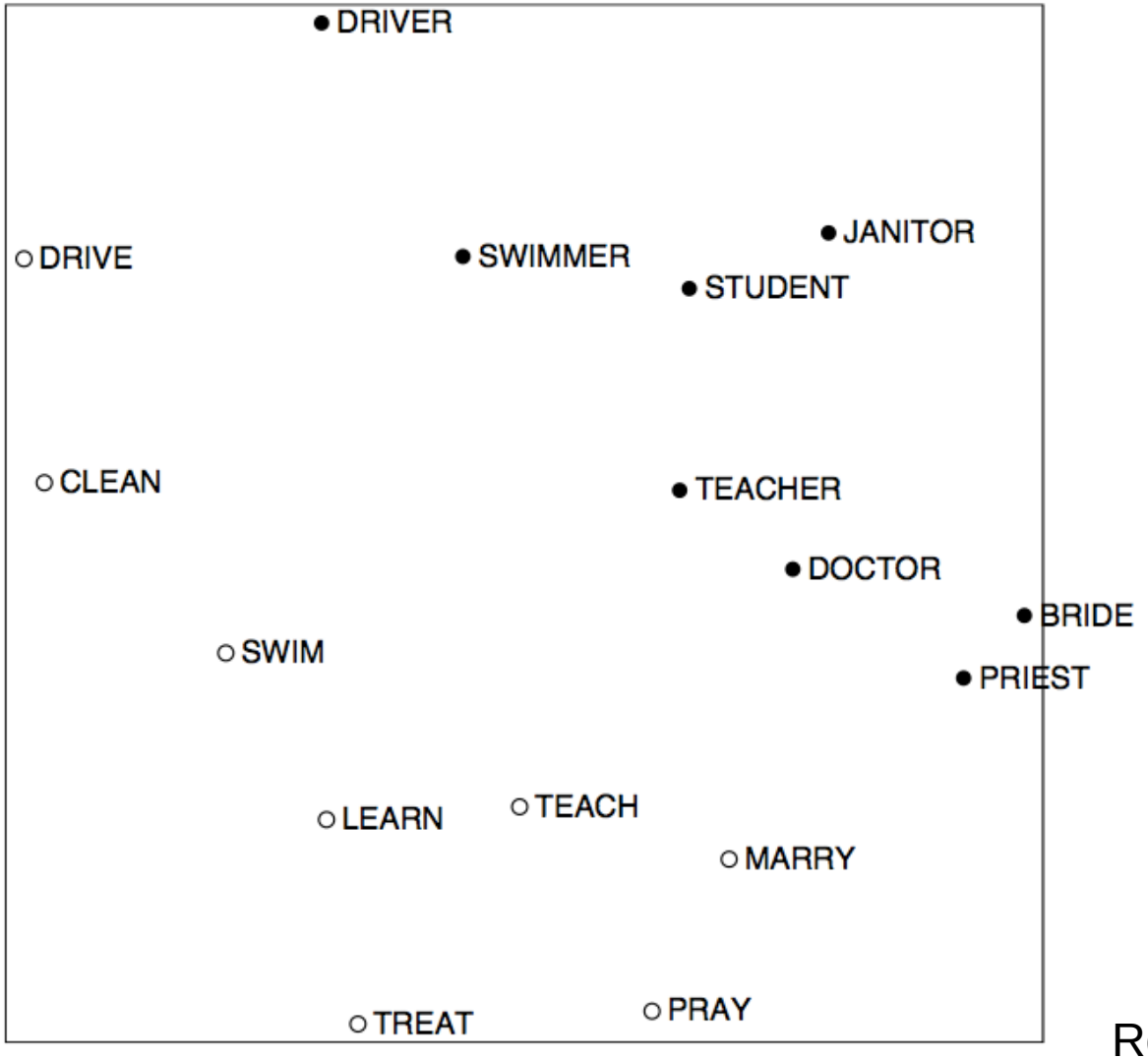
Rohde et al., 2005

Interesting semantic patters emerge in the vectors



Rohde et al., 2005

Interesting semantic patters emerge in the vectors



Rohde et al., 2005

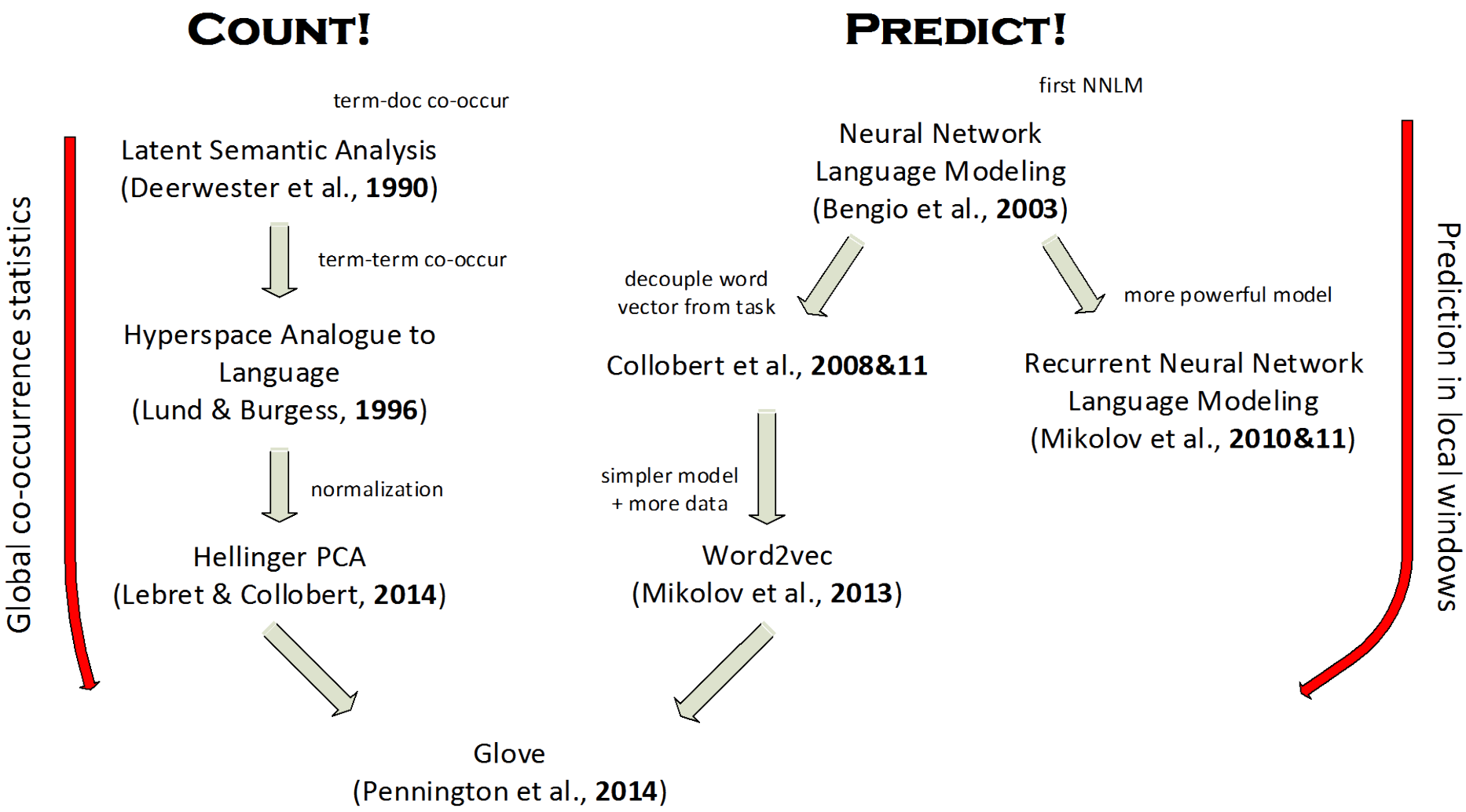
Problems with SVD

- Naïve 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

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

Word2vec

- The simplest NN-like model to train word embeddings
- **Skip-gram: given the central word, predict surrounding words**
- Continuous Bag-of-words (CBOW): given the surrounding words, predict the central word

Skip-gram

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

- Softmax: the simplest formulation for $p(w_{t+j}|w_t)$:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)}$$

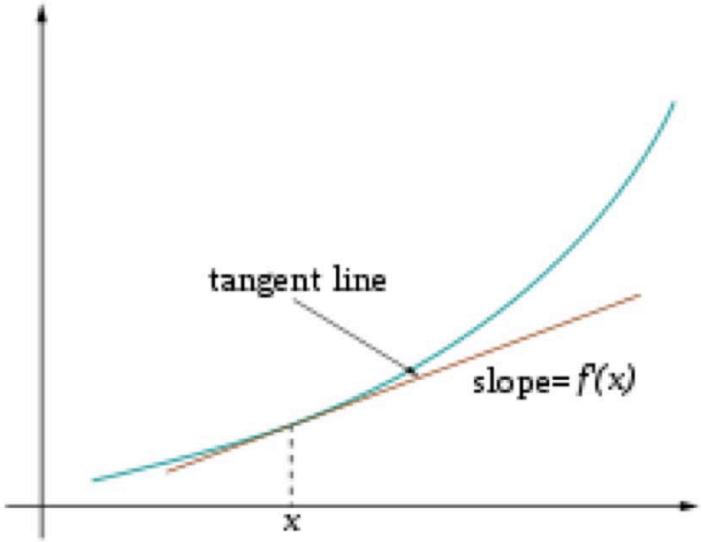
- *v* and *v'* are the "input" and "output" vectors of *w* (each words has two vectors!)
- *W* is the vocabulary size

Cost/Objective functions

We will optimize (maximize or minimize) our objective/cost functions

For now: minimize→gradient descent

Refresher with trivial example: (from Wikipedia)
Find a local minimum of the function
$f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$.




```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```

Derivation of gradients

$$\log p(o \mid c) = v_o^T v_c - \log(\sum_w \exp(v_w^T v_c))$$

See whiteboard!

$$\frac{\partial \log p(o \mid c)}{\partial v_c} = v_o - \sum_w p(w \mid c) v_w$$

Problem of the basic skip-gram

- 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

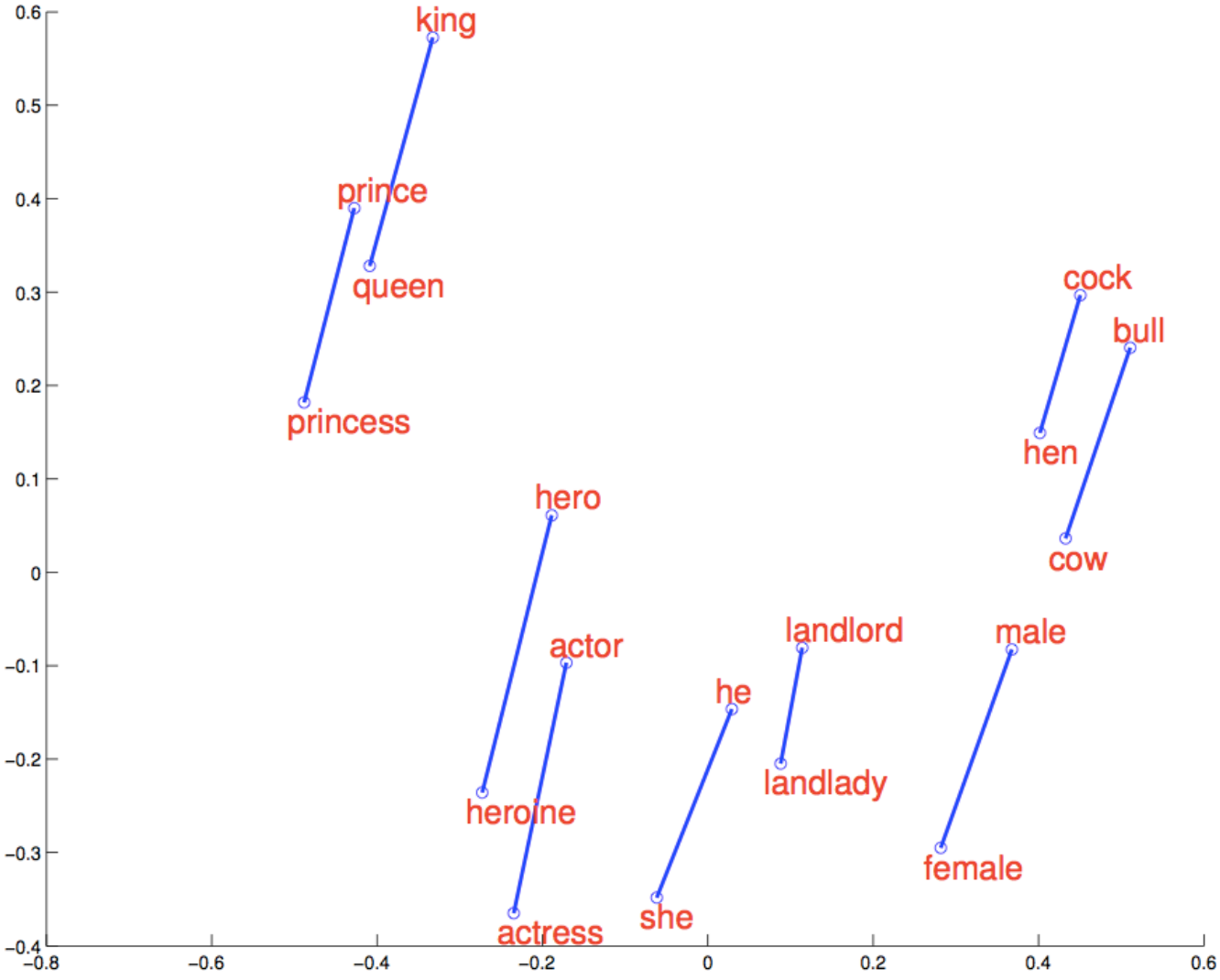
- There are multiple degrees of similarity among words:
  - **KING** is similar to **QUEEN** as **MAN** is similar to **WOMAN**
  - **KING** is similar to **KINGS** as **MAN** is similar to **MEN**

- 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

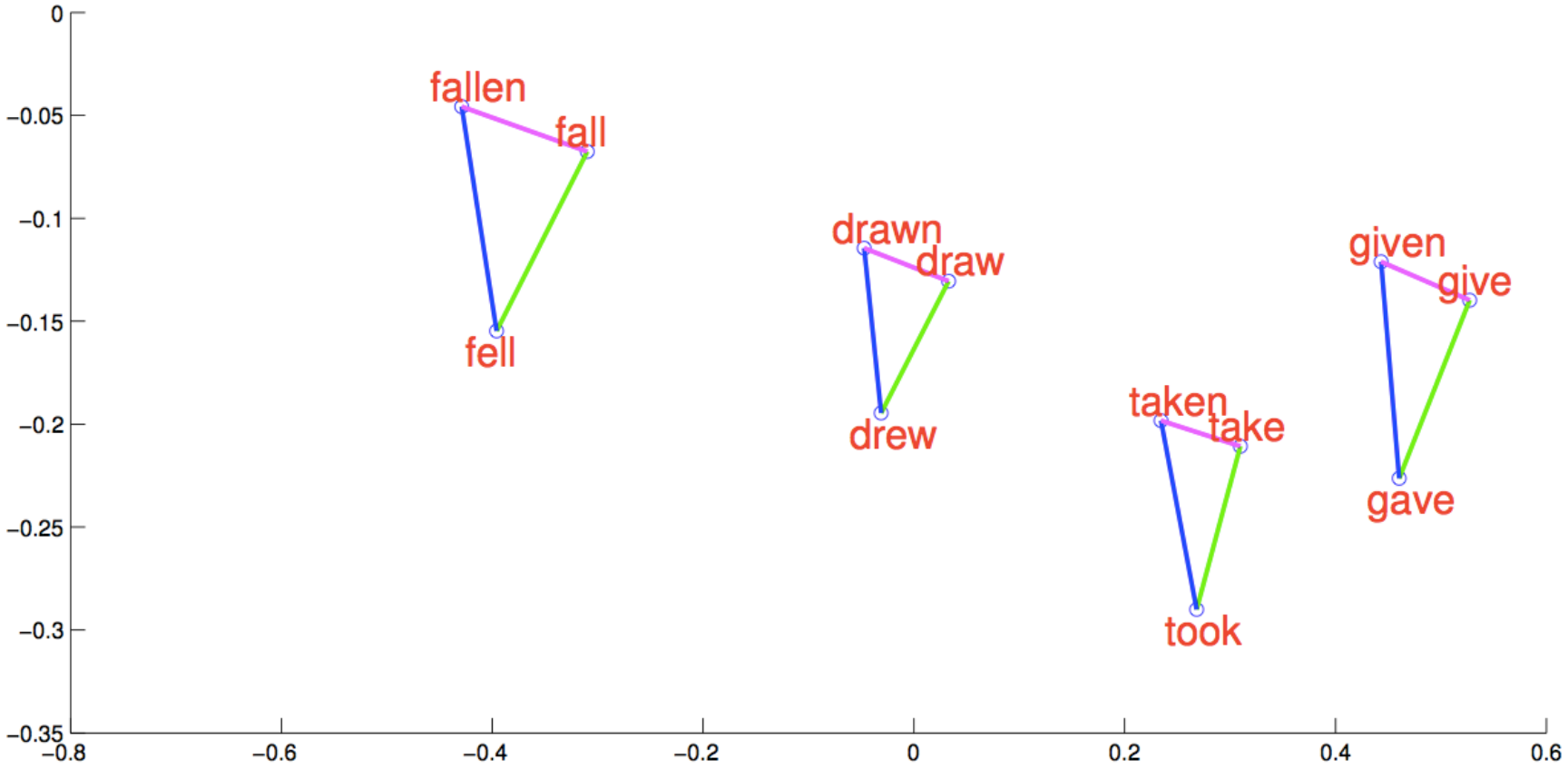
Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Visualization of regularities in word vector space



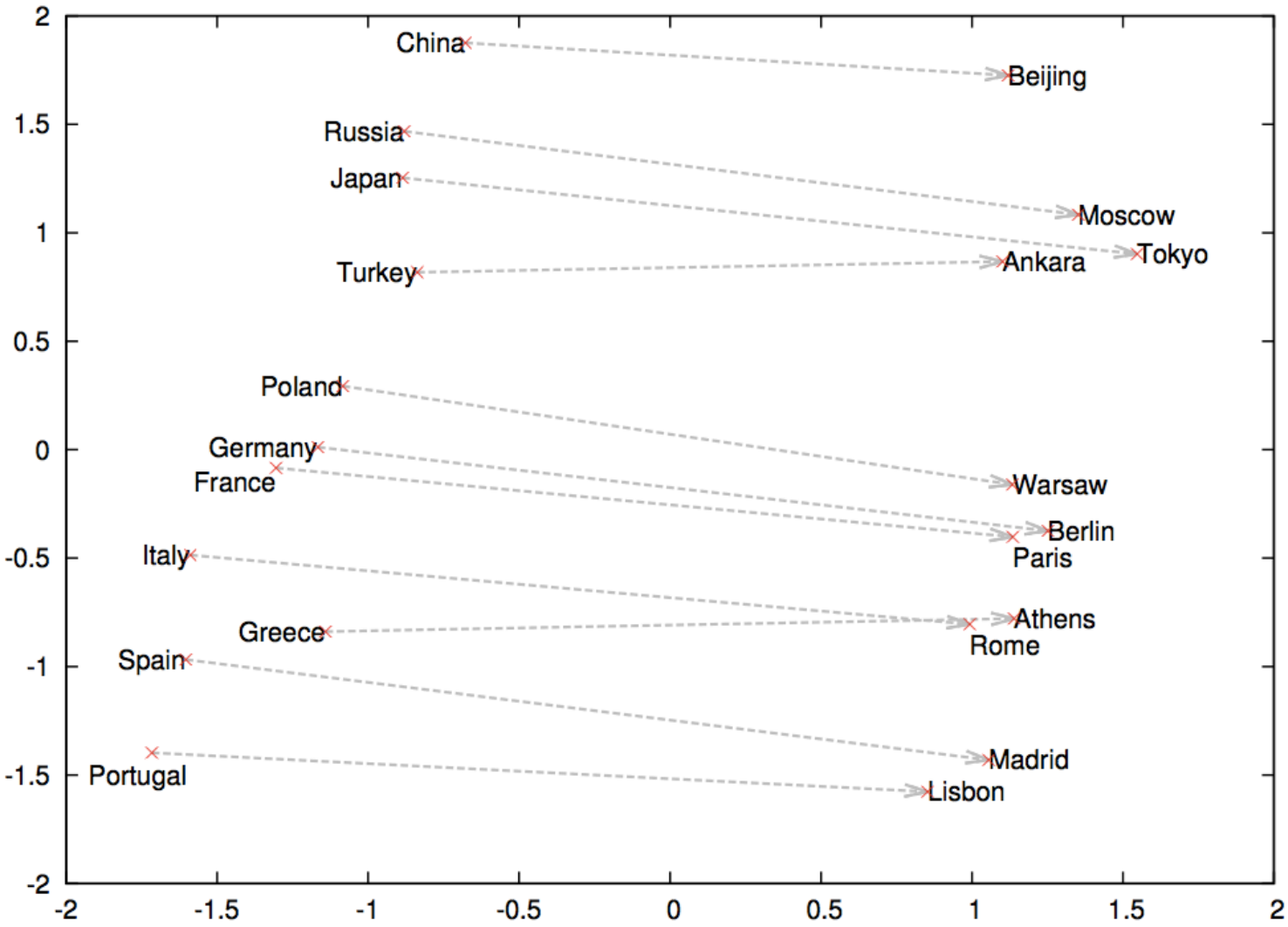
generated by PCA

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

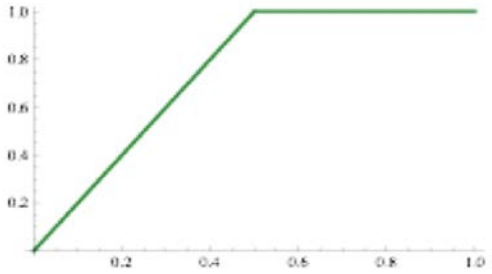
- Efficient usage of global statistics
- Primarily used to capture word similarity

Prediction

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

Combining the two worlds: Glove

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



- Fast Training
- Scalable to huge corpora (840 billion words)
- Good performance even with small corpus, and small vectors

Glove results

Nearest words to frog:

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

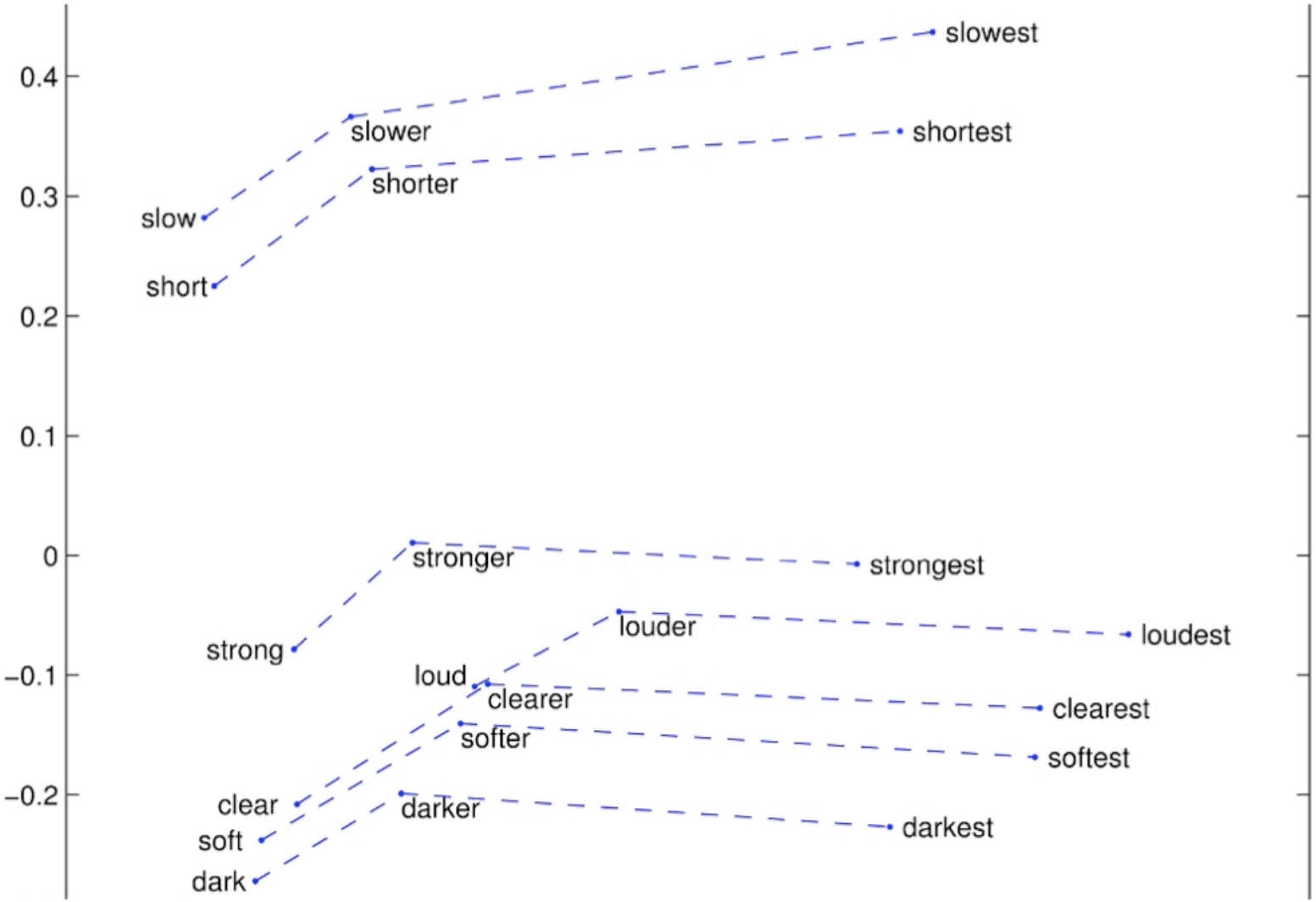
litoria

leptodactylidae

rana

eleutherodactylus

Glove results



Project ideas

- Count based: Non-linear dimensionality reduction on X
  - What's the intrinsic structure of language? Linear? Probably not.
  - Non-linear dimensionality reduction on X to reveal non-linear structure of language
  - Very challenging
    - Which algorithm to use?
    - Single sub-structure or multiple sub-structured scattered in the space?
    - How to represent relations computationally?
    - How to evaluate?
- Prediction based: incorporate prior linguistic knowledge
  - We already know a whole lot about language: synonyms, hypernyms, etc.
  - combine to learn linguistically regularized word embeddings
  - Markov logic networks or regularization

Project ideas

- Prediction based: Phrase/sentence/paragraph embeddings
  - Recently under active investigation (e.g., Le & Mikolov 2014)
  - Still in “baby steps”
  - The meaning of a paragraph vector is not very clear
  - What relations exist between two sentences?

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/