Natural Language Processing with Deep Learning

CS224N/Ling284

Lecture 3: More Word Vectors

Richard Socher

Organization. See Calendar

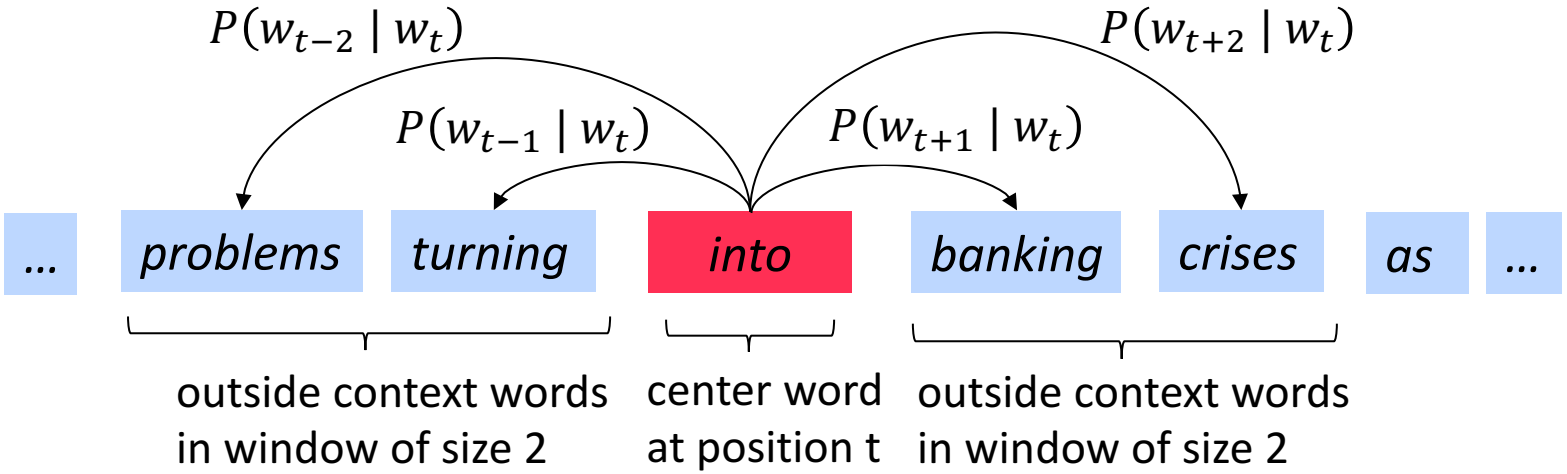
- Project advice office hours are today

A Deeper Look at Word Vectors

- Finish word2vec
- What does word2vec capture?
- How could we capture this essence more effectively?
- How can we analyze word vectors?

Review: Main ideas of word2vec

- Go through each word of the whole corpus
- Predict surrounding words of each word



This demonstrates skip-gram method, like a moving window (kind of CNN with 1*k kernel)

- $P(o|c) = \dfrac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

Stochastic gradients with word vectors!

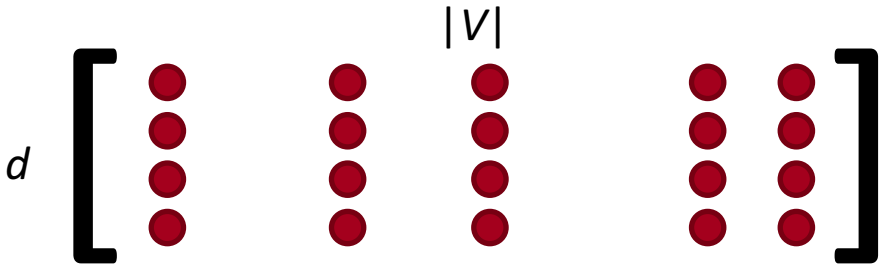
- Then take gradients at each such window for SGD
- But in each window, we only have at most $2m + 1$ words, so $\nabla_\theta J_t(\theta)$ is very sparse!

$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Stochastic gradients with word vectors!

- We may as well only update the word vectors that actually appear!

- Solution: either you need sparse matrix update operations to only update certain columns of full embedding matrices *U* and *V,* or you need to keep around a hash for word vectors



- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!

Approximations: Assignment 1

- The normalization factor is too computationally expensive.

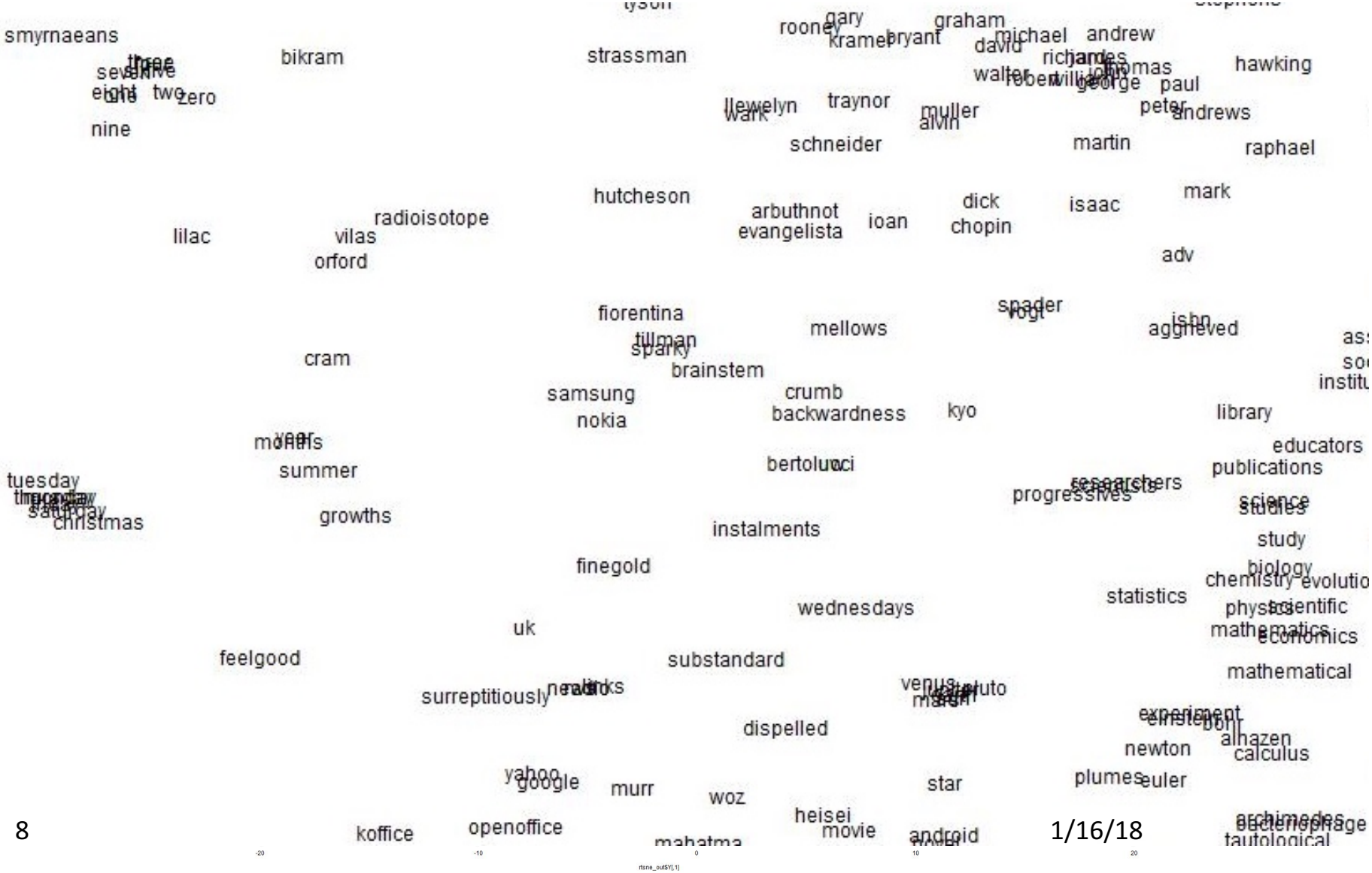
- $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

- Hence, in Assignment 1, you will implement the skip-gram model with **negative sampling**

  Randomly pick irrelevant words and pair up with the center word

- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus a couple of noise pairs (the center word paired with a random word)

Word2vec improves objective function by putting similar words nearby in space



Summary of word2vec

- Go through each word of the whole corpus
- Predict surrounding words of each word

- This captures co-occurrence of words one at a time

- Why not capture co-occurrence **counts** directly?

Yes we can!

With a co-occurrence matrix X

- 2 options: windows vs full document
- Window: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information

- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”

Example: Window based co-occurrence matrix

- Window length 1 (more common: 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

- Example corpus:
  - I like deep learning.
  - I like NLP.
  - I enjoy flying.

It's already a wordvec!!

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

If online learning is implemented, the counting vector would keep changing, which is not good for model implementation

Very high dimensional: require a lot of storage

One new word => one more dimension, no matter how many of them have similar meanings

Subsequent classification models have sparsity issues

→ Models are less robust

Solution: Low dimensional vectors

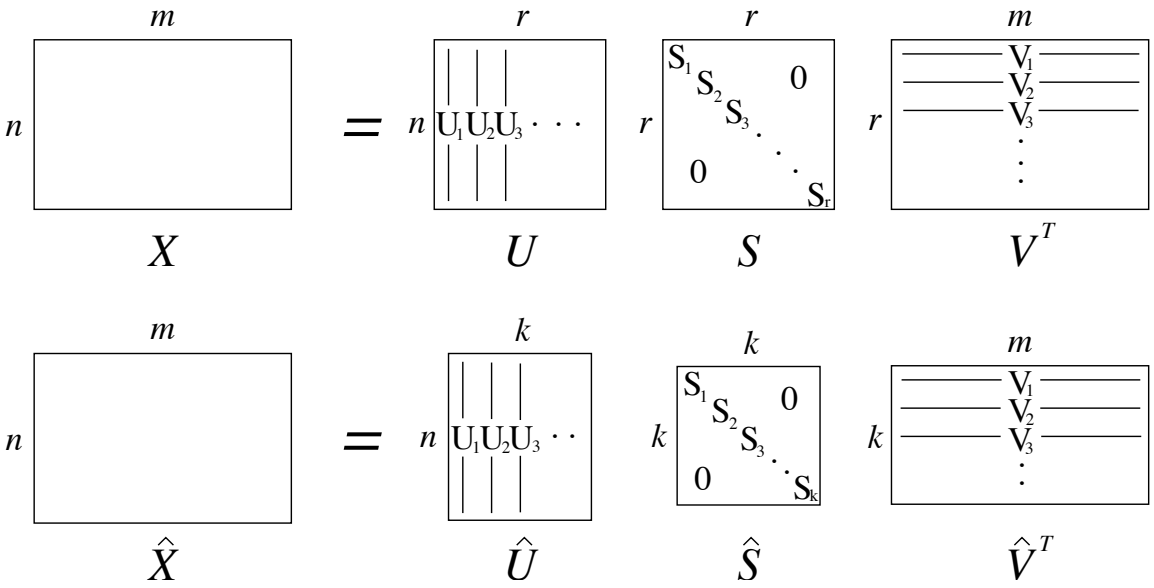
- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector

- Usually 25 – 1000 dimensions, similar to word2vec

- How to reduce the dimensionality?

Method 1: Dimensionality Reduction on X

Singular Value Decomposition of co-occurrence matrix *X*.



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

Simple SVD word vectors in Python

Corpus:
I like deep learning. I like NLP. I enjoy flying.

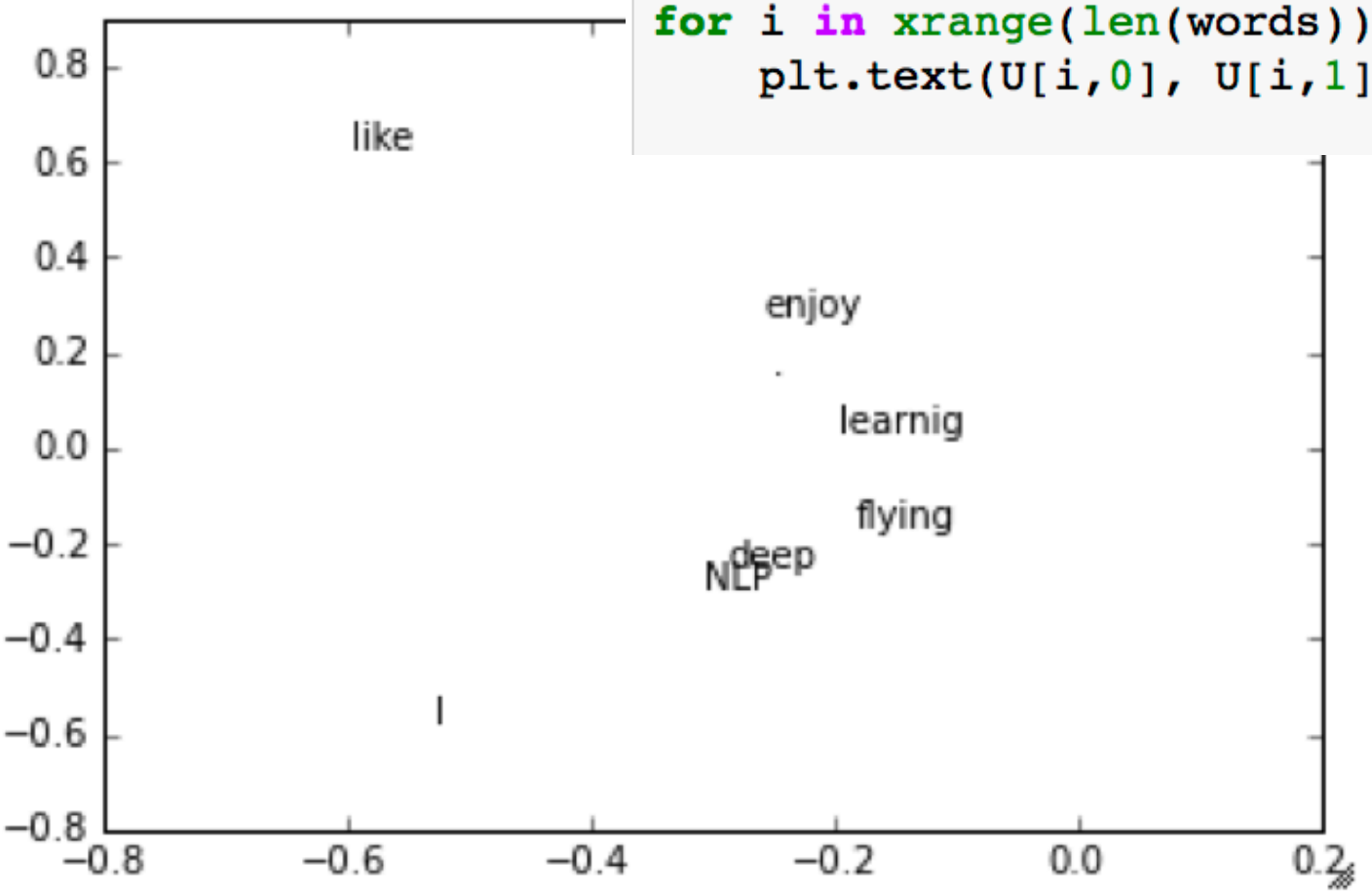
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [0,0,1,0,0,0,0,1],
              [0,0,0,0,1,1,1,0]])

U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

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

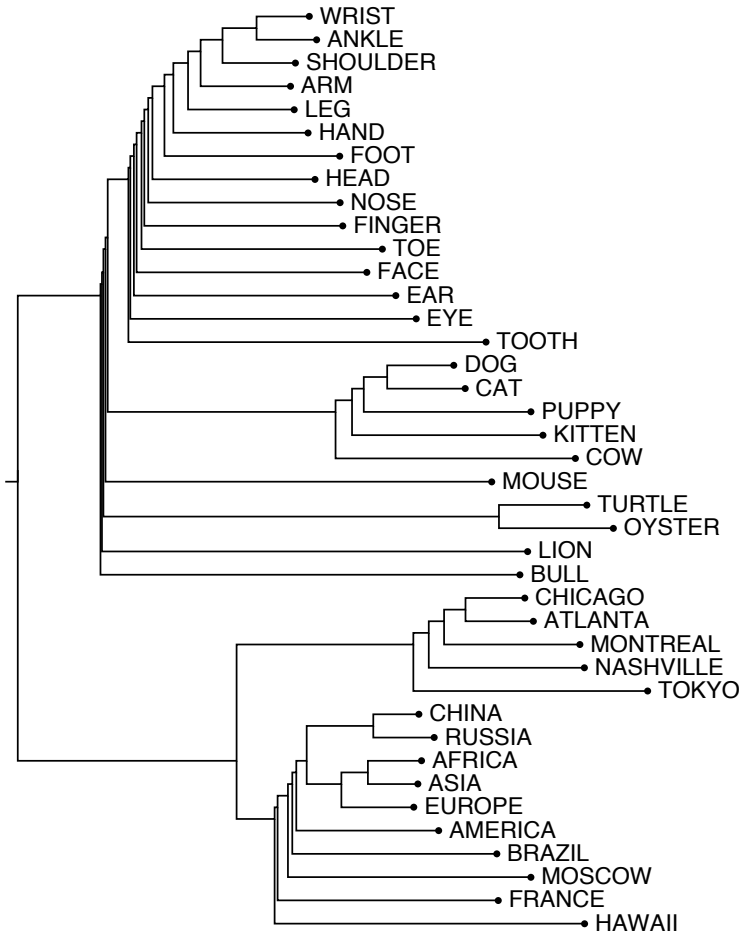
```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Hacks to X

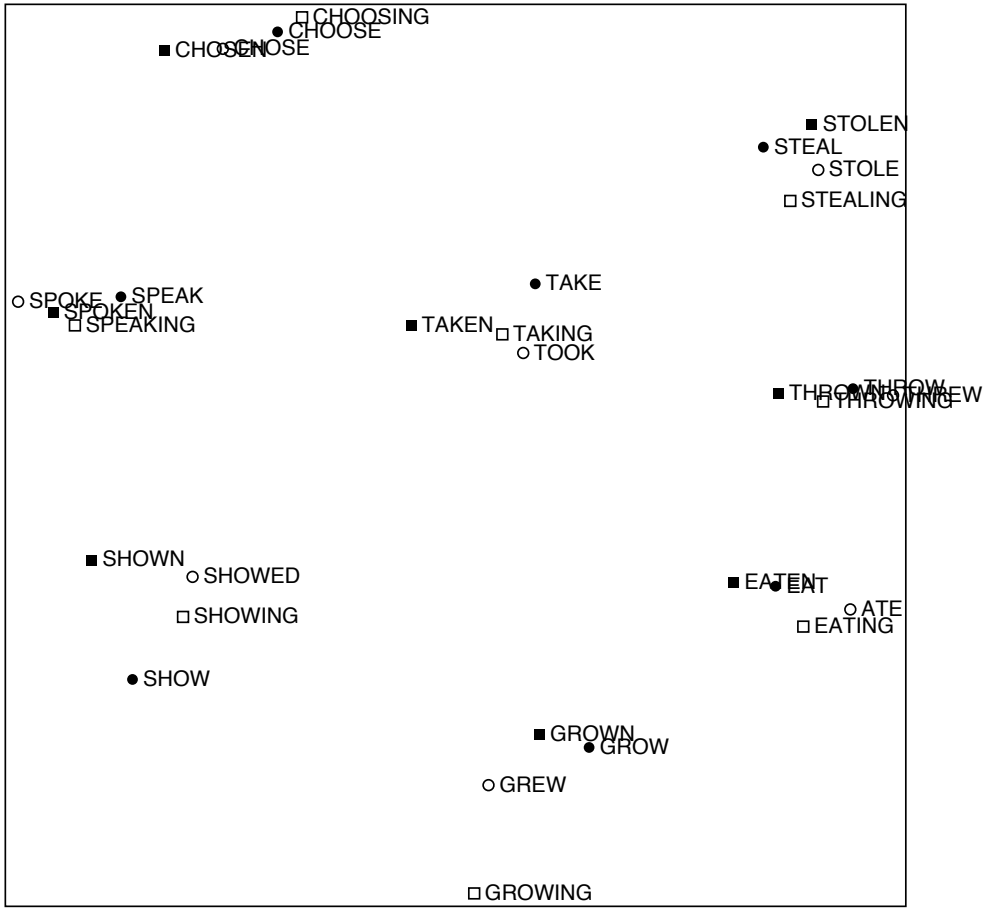
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t~100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- +++

Interesting semantic patters emerge in the vectors



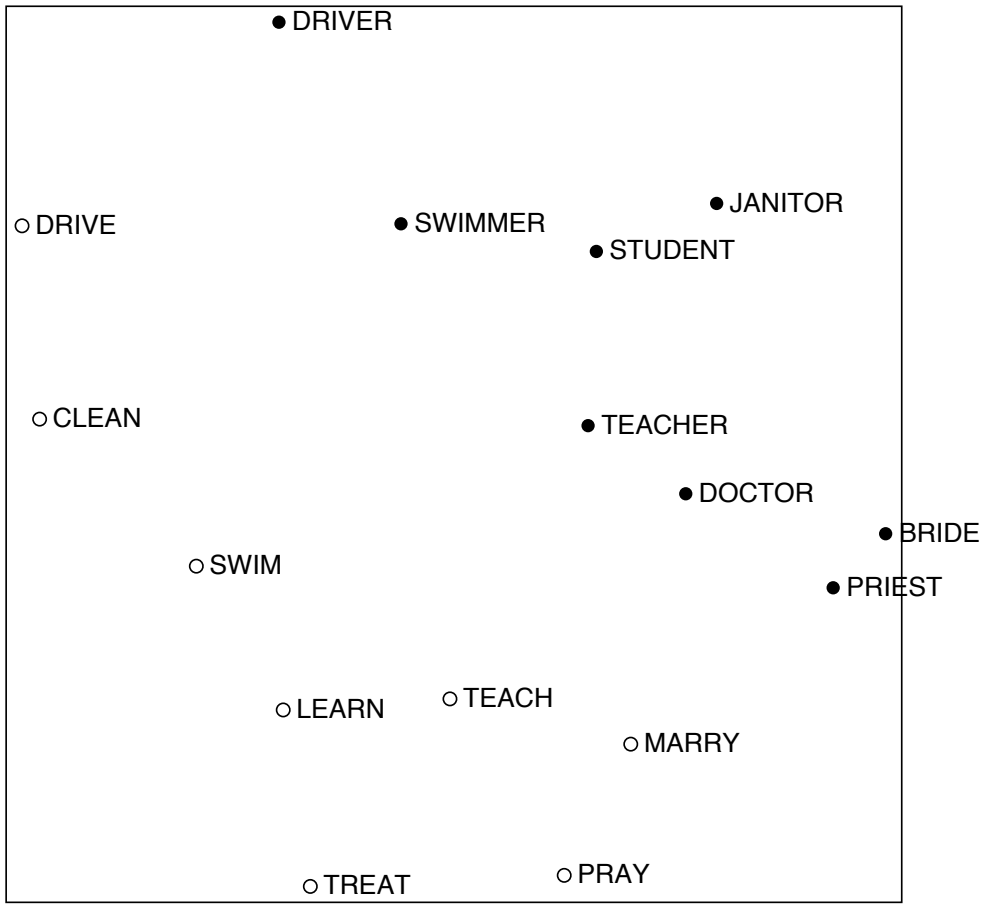
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting syntactic patters emerge in the vectors



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

Interesting semantic patters emerge in the vectors



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

Problems with SVD

Computational cost scales quadratically for n x m matrix:

O(mn^2) flops (when n<m)

→ Bad for millions of words or documents

For every new word, we need to re-train the model

Hard to incorporate new words or documents

Different learning regime than other DL models

Count based vs direct prediction

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

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

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

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

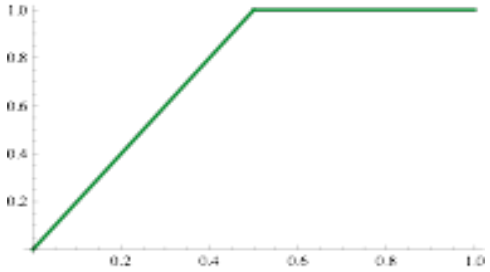
Combining the best of both worlds: GloVe

Skip-gram: slide and tries to capture co-occurences one window at a time
GloVe: tries to capture the counts of the overall statistics of how often the words occur together

$$J(\theta) = \frac{1}{2} \sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

u: vectors of all words (row)
v: vectors of all words (col)

- Fast training
- Scalable to huge corpora

$f \sim$ 

- Good performance even with small corpus, and small vectors
- By Pennington, Socher, Manning (2014)

What to do with the two sets of vectors?

- We end up with U and V from all the vectors u and v (in columns)

- Both capture similar co-occurrence information. It turns out, the best solution is to simply sum them up:

$$X_{final} = U + V$$

Based on research try and the best outcome it gets, maybe not in other language models (Germany, French, or Chinese)

- One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation,* Pennington et al. (2014)

Glove results

Nearest words to frog:

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

litoria

leptodactylidae

rana

eleutherodactylus

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy → Winning!

Intrinsic word vector evaluation

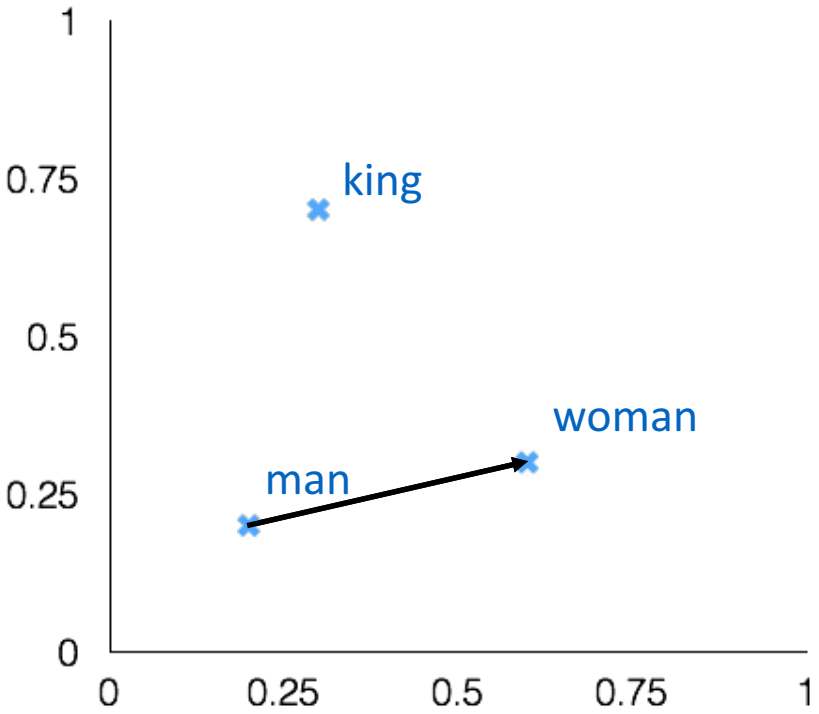
- Word Vector Analogies

a:b :: c:?

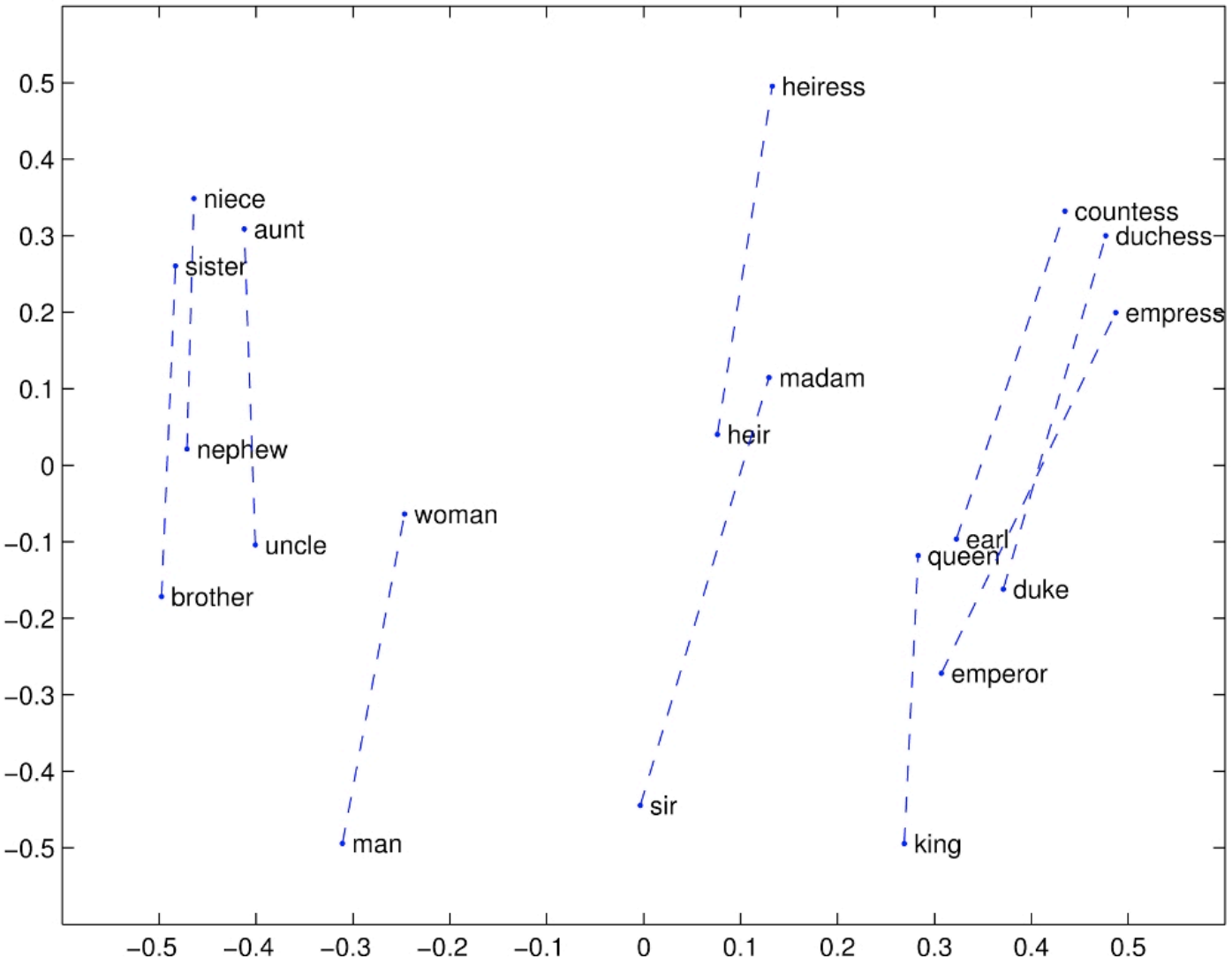
man:woman :: king:?

$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$

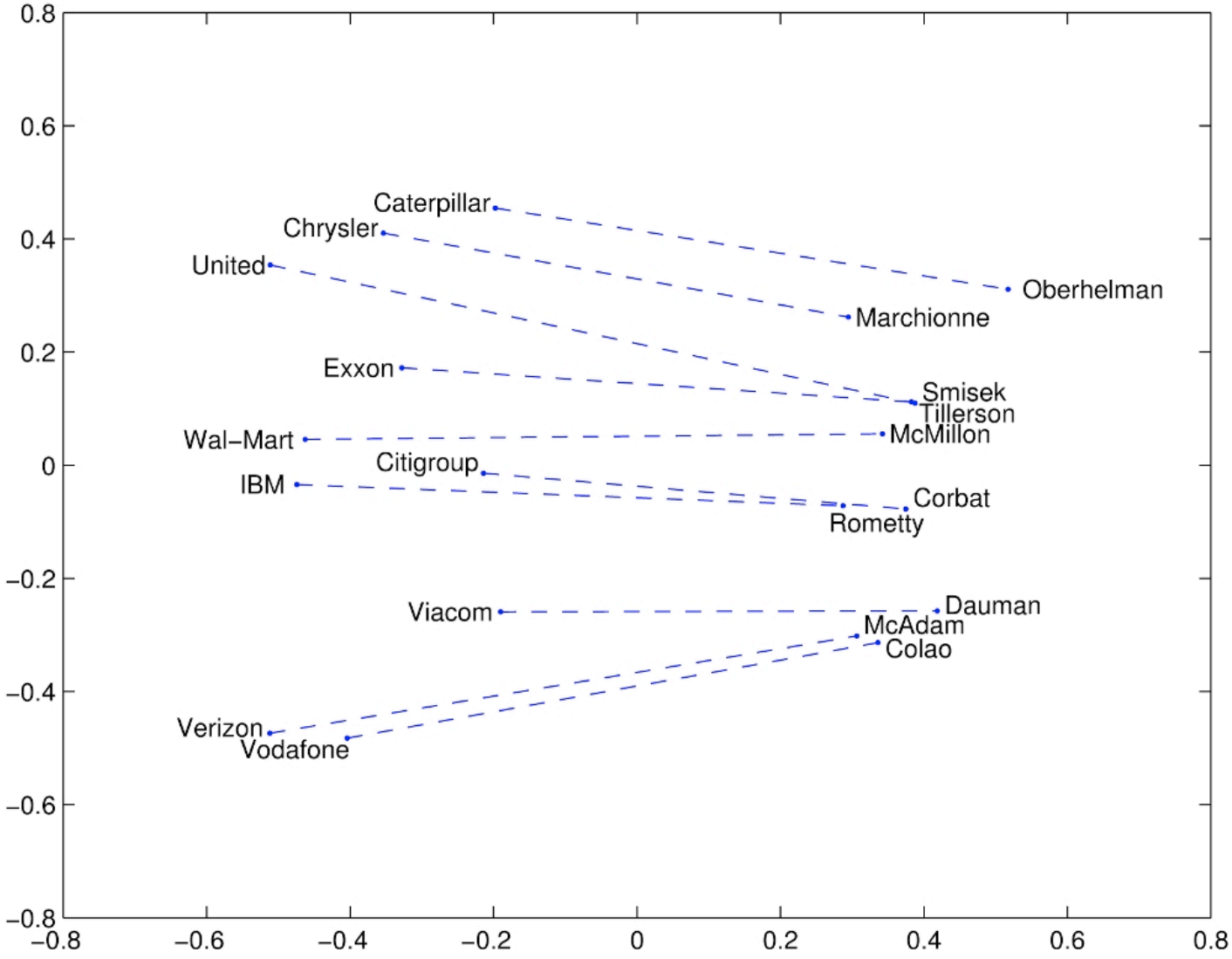
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



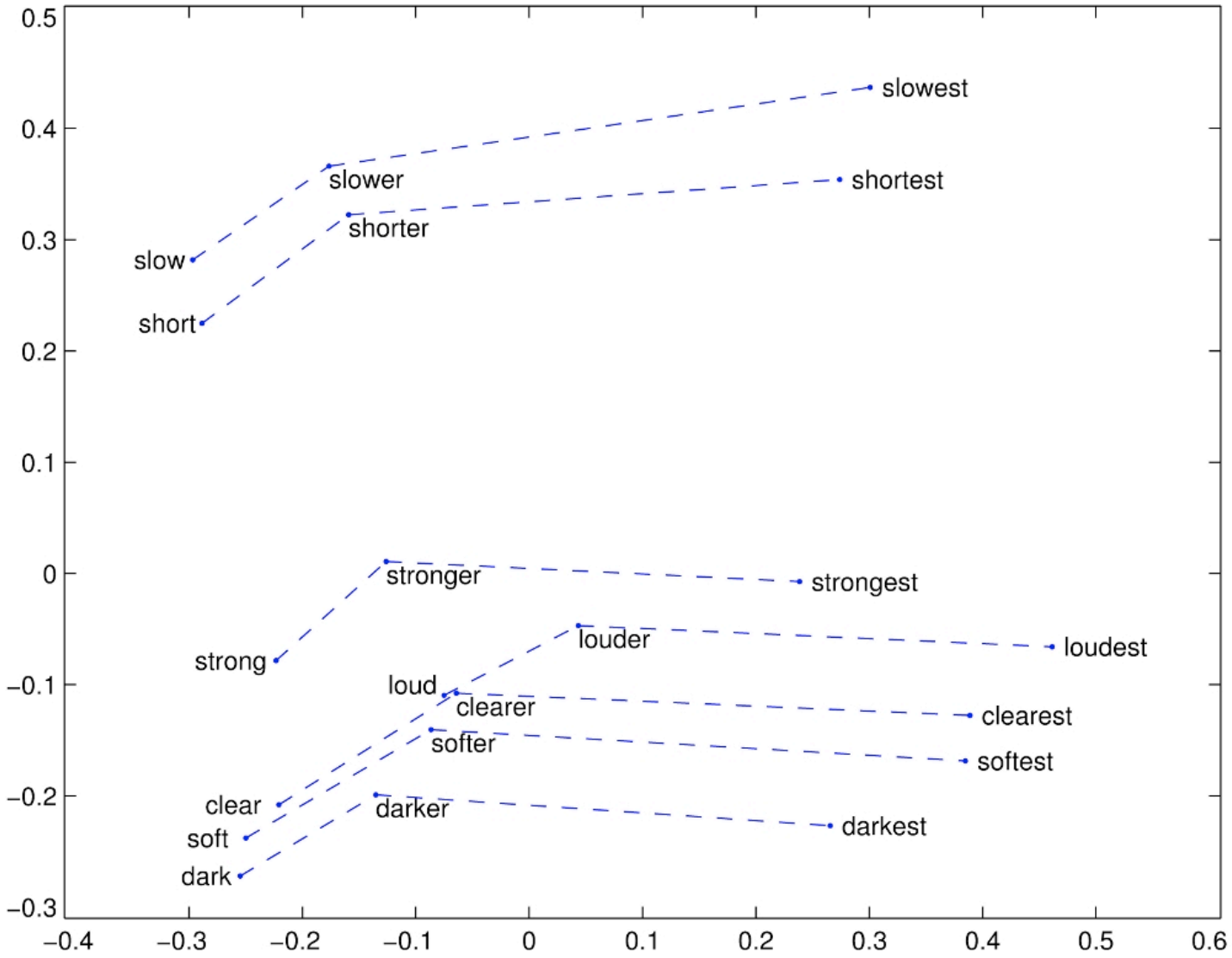
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Other fun word2vec analogies

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

Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

: city-in-state
Chicago Illinois Houston Texas
Chicago Illinois Philadelphia Pennsylvania
Chicago Illinois Phoenix Arizona
Chicago Illinois Dallas Texas
Chicago Illinois Jacksonville Florida
Chicago Illinois Indianapolis Indiana
Chicago Illinois Austin Texas
Chicago Illinois Detroit Michigan
Chicago Illinois Memphis Tennessee
Chicago Illinois Boston Massachusetts

problem: different cities may have same name

Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from

: capital-world
Abuja Nigeria Accra Ghana
Abuja Nigeria Algiers Algeria
Abuja Nigeria Amman Jordan
Abuja Nigeria Ankara Turkey
Abuja Nigeria Antananarivo Madagascar
Abuja Nigeria Apia Samoa
Abuja Nigeria Ashgabat Turkmenistan
Abuja Nigeria Asmara Eritrea
Abuja Nigeria Astana Kazakhstan

problem: can change

Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from

: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from

: gram7-past-tense
dancing danced decreasing decreased
dancing danced describing described
dancing danced enhancing enhanced
dancing danced falling fell
dancing danced feeding fed
dancing danced flying flew
dancing danced generating generated
dancing danced going went
dancing danced hiding hid
dancing danced hitting hit

Analogy evaluation and hyperparameters

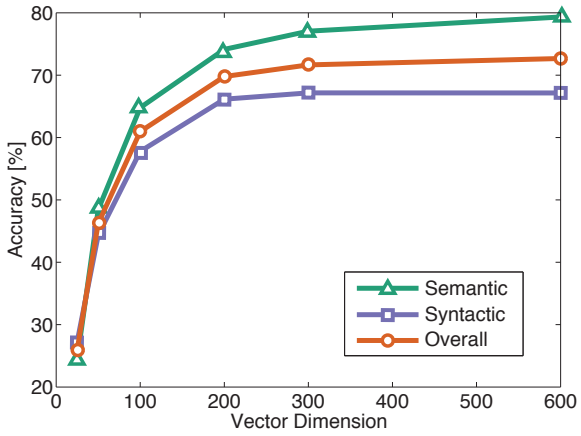
- Very careful analysis: Glove word vectors

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**81.9**	**69.3**	**75.0**

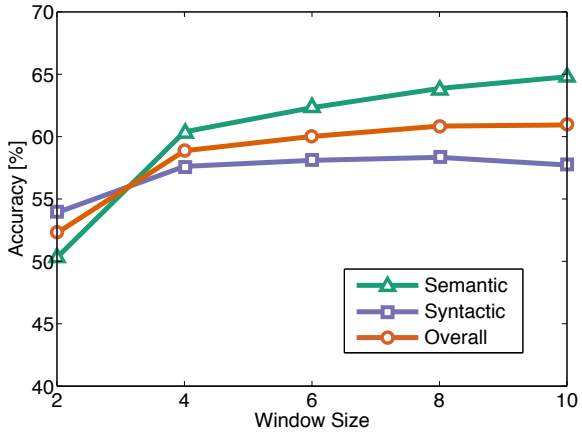
More data, more accurate;
More dimensions do not guarantee more accurate!

Analogy evaluation and hyperparameters

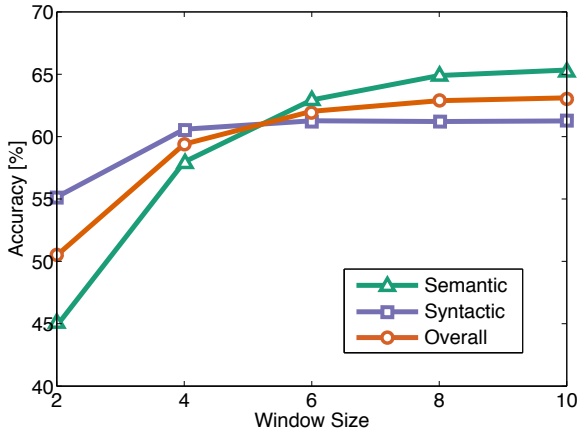
- Asymmetric context (only words to the left) are not as good



(a) Symmetric context



(b) Symmetric context



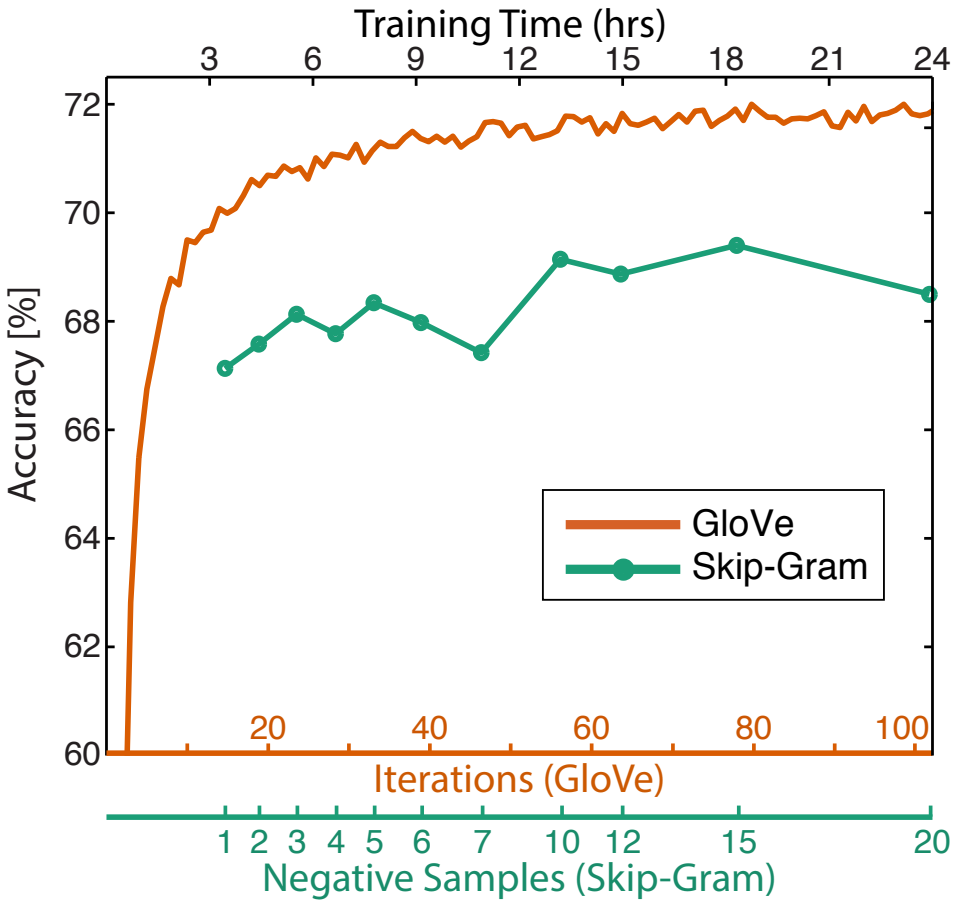
(c) Asymmetric context

- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!

- Window size of 8 around each center word is good for Glove vectors

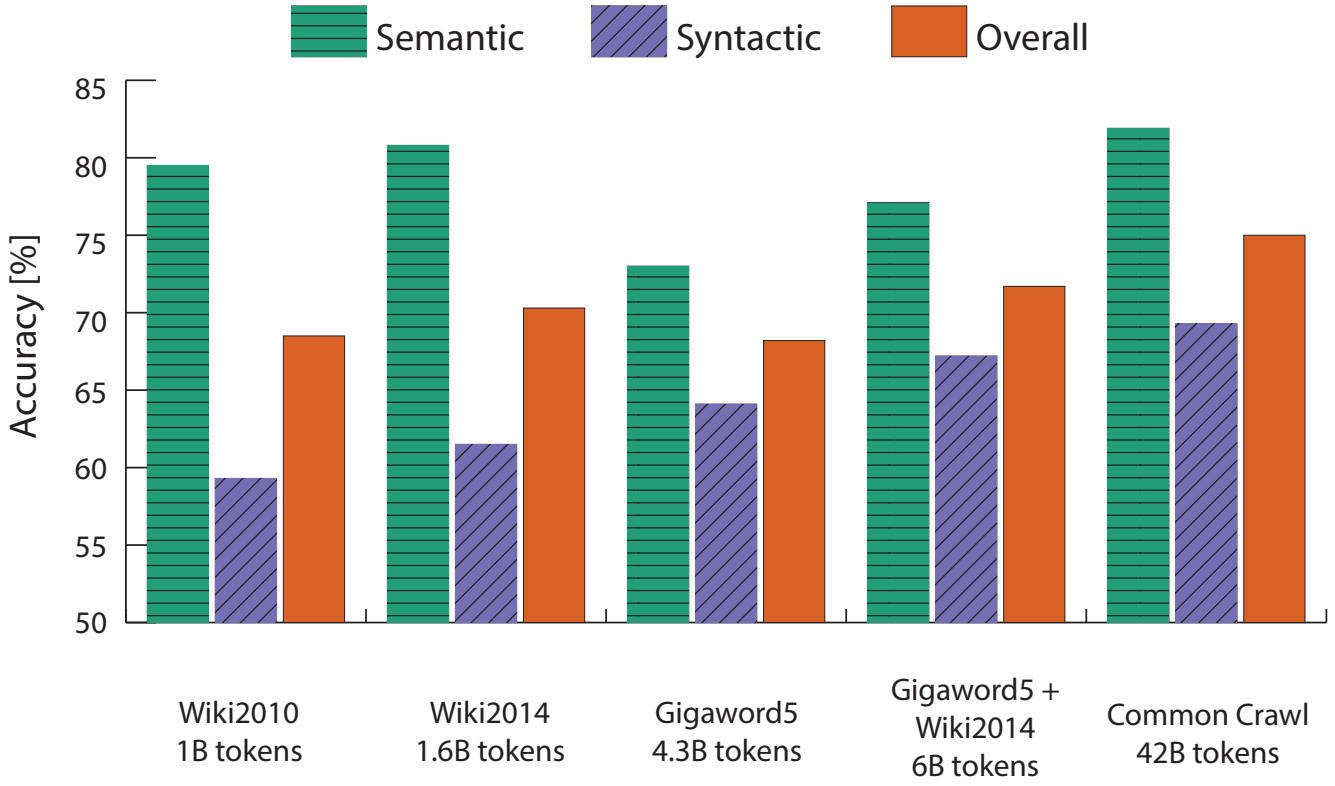
Analogy evaluation and hyperparameters

- More training time helps



Analogy evaluation and hyperparameters

- More data helps, Wikipedia is better than news text!



Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353
  http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10.00
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Closest words to “Sweden” (cosine similarity)

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Correlation evaluation

- Word vector distances and their correlation with human judgments

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	72.7	75.1	56.5	37.0
CBOW†	6B	57.2	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	**75.9**	**83.6**	**82.9**	**59.6**	**47.8**
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)

But what about ambiguity?

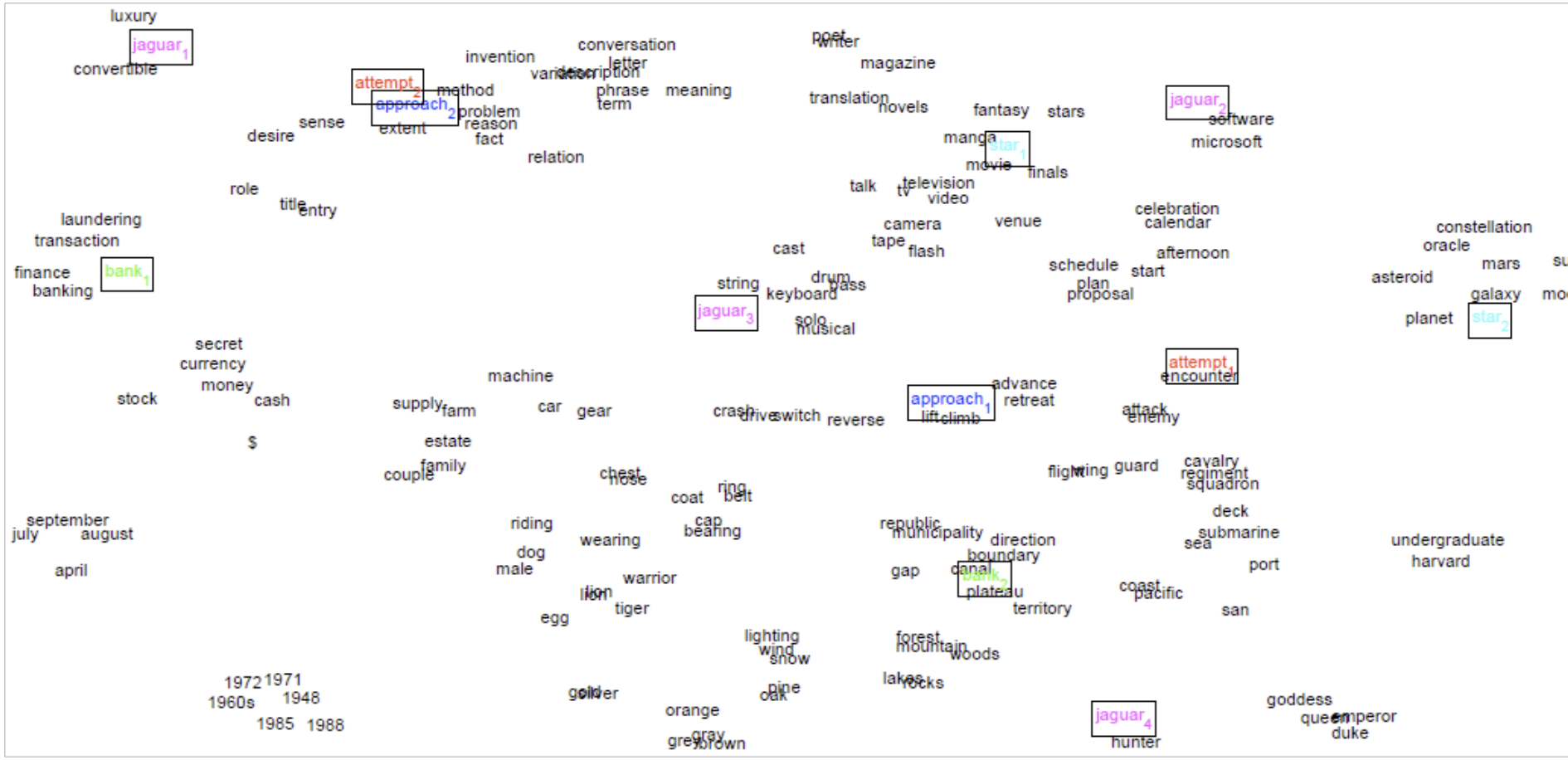
- You may hope that one vector captures both kinds of information (run = verb and noun) but then vector is pulled in different directions

- Alternative described in: *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)

- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters $bank_1$, $bank_2$, etc

But what about ambiguity?

- *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)



Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this class

- One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	**88.7**	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	**93.2**	88.3	**82.9**	**82.2**

- Next: How to use word vectors in neural net models!

Next level up: Word and Window classification

- Let's add context by taking in windows and classifying the center word of that window (and not just representing it across all windows)!

- Possible: Softmax and cross entropy error or **max-margin loss**

- Next class!