

Natural Language Processing with Deep Learning

CS224N/Ling284



Lecture 3: More Word Vectors

Christopher Manning and Richard Socher

Organization. See Calendar

Coding Session!

Wednesday, 6:30 - 9:30pm at Charles B. Thornton Center
for Engineering and Management, room 110

Career Fair

Wednesday 11am - 4pm

My first project advice office hour is 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 (window's center) word

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

- Then take gradients at each such window for SGD

Stochastic gradients with word vectors!

- 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

$$d \quad [\begin{array}{c:c:c:c} \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots \end{array}]^{|V|}$$

- 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.

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$$

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

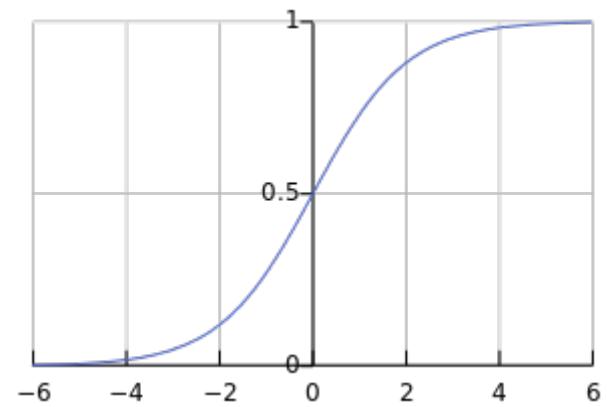
Ass 1: The skip-gram model and negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function: $J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

- Where k is the number of negative samples and we use,

- The sigmoid function! $\sigma(x) = \frac{1}{1+e^{-x}}$
(we'll become good friends soon)
- So we maximize the probability
of two words co-occurring in first log
 \rightarrow



Ass 1: The skip-gram model and negative sampling

- Slightly clearer notation:

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

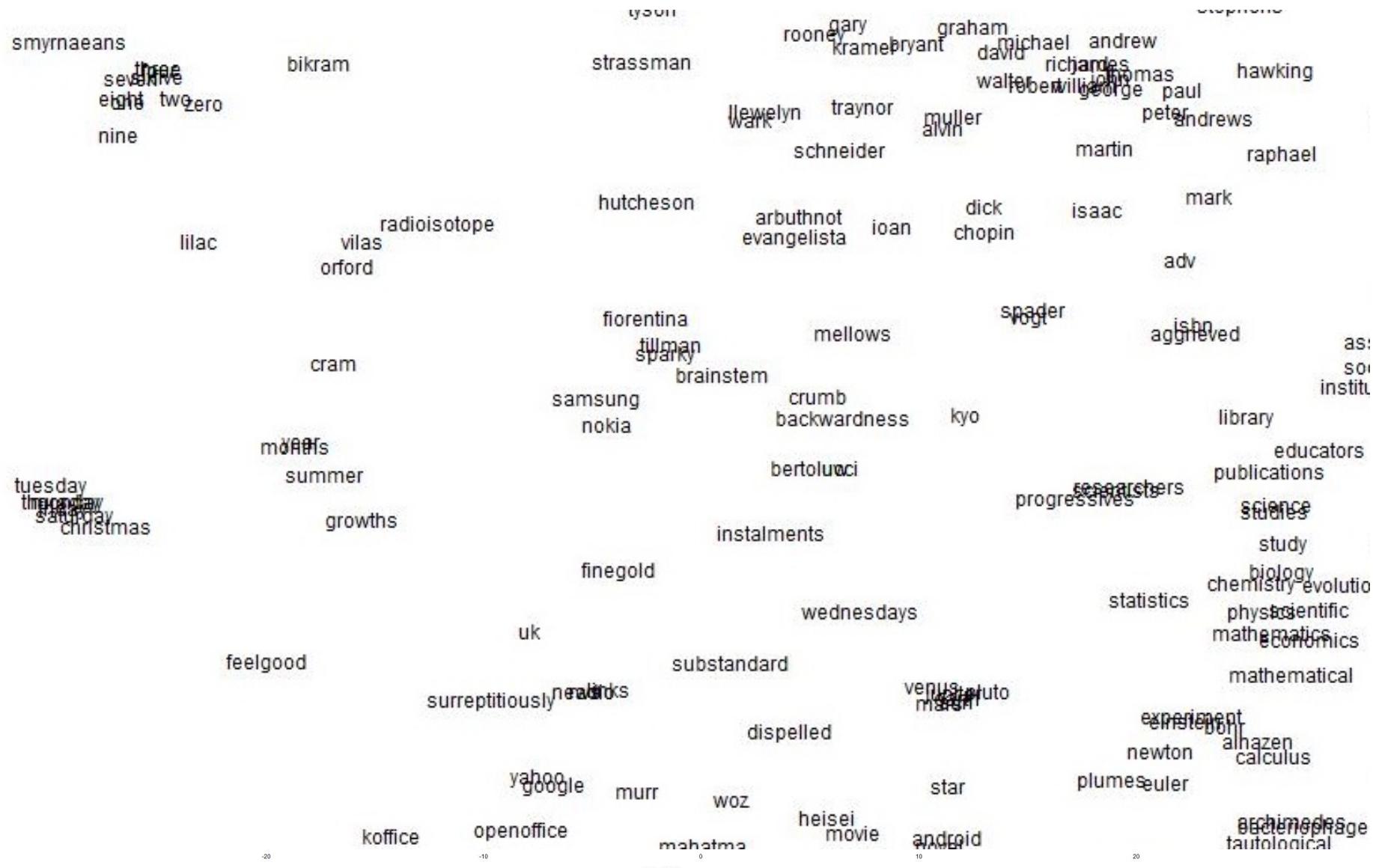
- Maximize probability that real outside word appears, minimize prob. that random words appear around center word
- $P(w) = U(w)^{3/4} / Z$,
the unigram distribution $U(w)$ raised to the $3/4$ power
(We provide this function in the starter code).
- The power makes less frequent words be sampled more often

Ass 1: The continuous bag of words model

- Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model
- To make assignment slightly easier:

Implementation of the CBOW model is not required (you can do it for a couple of bonus points!), but you do have to do the written problem on CBOW.

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 cooccurrence of words one at a time
- Why not capture cooccurrence **counts** directly?

Yes we can!

With a co-occurrence matrix X

- 2 options: full document vs. windows
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
- Instead: Similar to word2vec, use window around each word → captures both syntactic (POS) and semantic information

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.

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 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 .

$$\begin{matrix} & m \\ n & \boxed{} \end{matrix} = n \begin{matrix} r \\ \boxed{| | |} \\ U_1 U_2 U_3 \dots \end{matrix} r \begin{matrix} r \\ S_1 S_2 S_3 \dots \\ 0 \\ \vdots \\ S_r \end{matrix} r \begin{matrix} m \\ V_1 \\ V_2 \\ V_3 \\ \vdots \end{matrix}$$

X U S V^T

$$\begin{matrix} & m \\ n & \boxed{} \end{matrix} = n \begin{matrix} k \\ \boxed{| | |} \\ U_1 U_2 U_3 \dots \end{matrix} k \begin{matrix} k \\ S_1 S_2 S_3 \dots \\ 0 \\ \vdots \\ S_k \end{matrix} k \begin{matrix} m \\ V_1 \\ V_2 \\ V_3 \\ \vdots \end{matrix}$$

\hat{X} \hat{U} \hat{S} \hat{V}^T

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

Simple SVD word vectors in Python

Corpus:

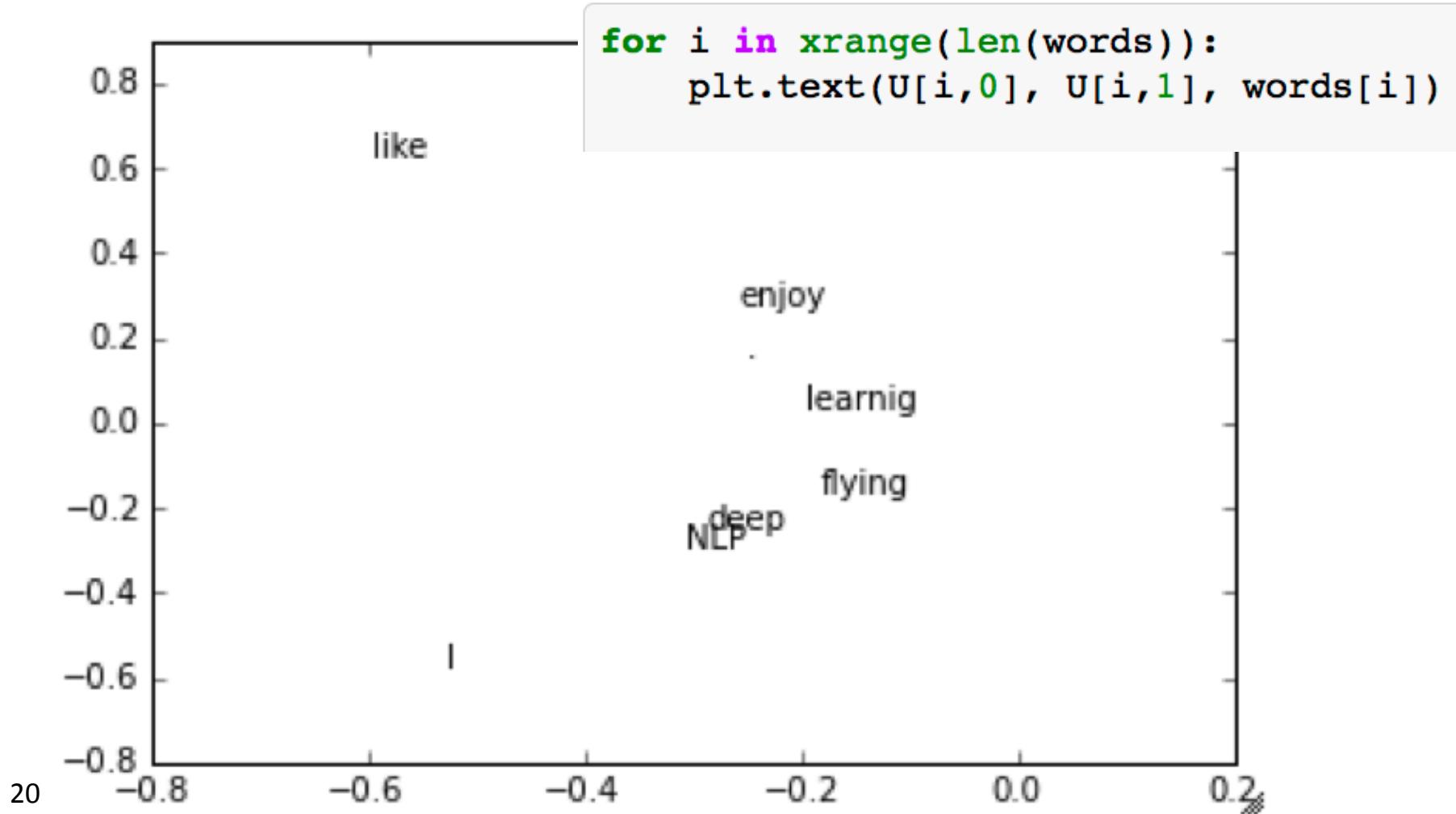
I like deep learning. I like NLP. I enjoy flying.

```
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.

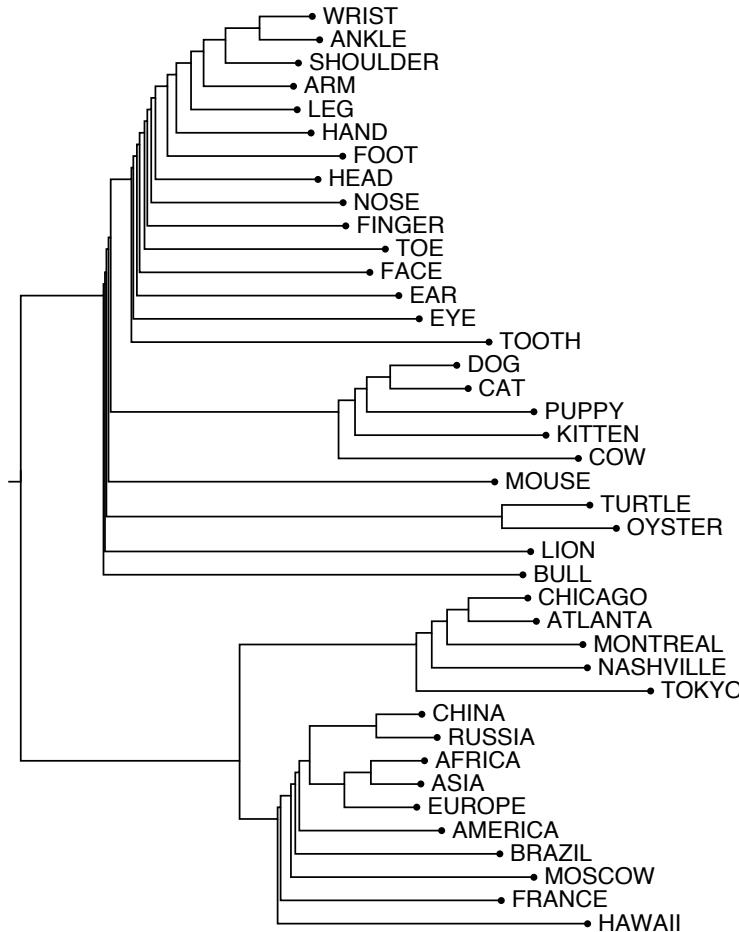
Printing first two columns of U corresponding to the 2 biggest singular values



Hacks to X

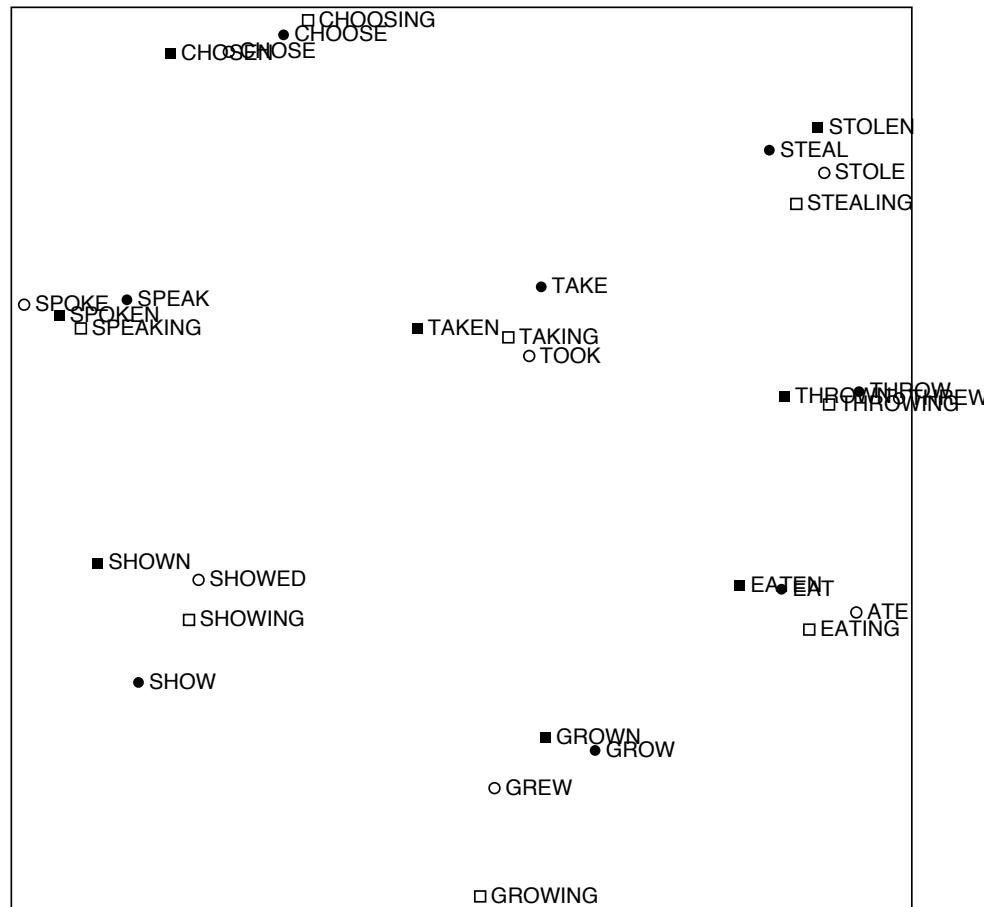
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
 - $\min(X, t)$, with $t \sim 100$
 - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- +++

Interesting semantic patterns emerge in the vectors



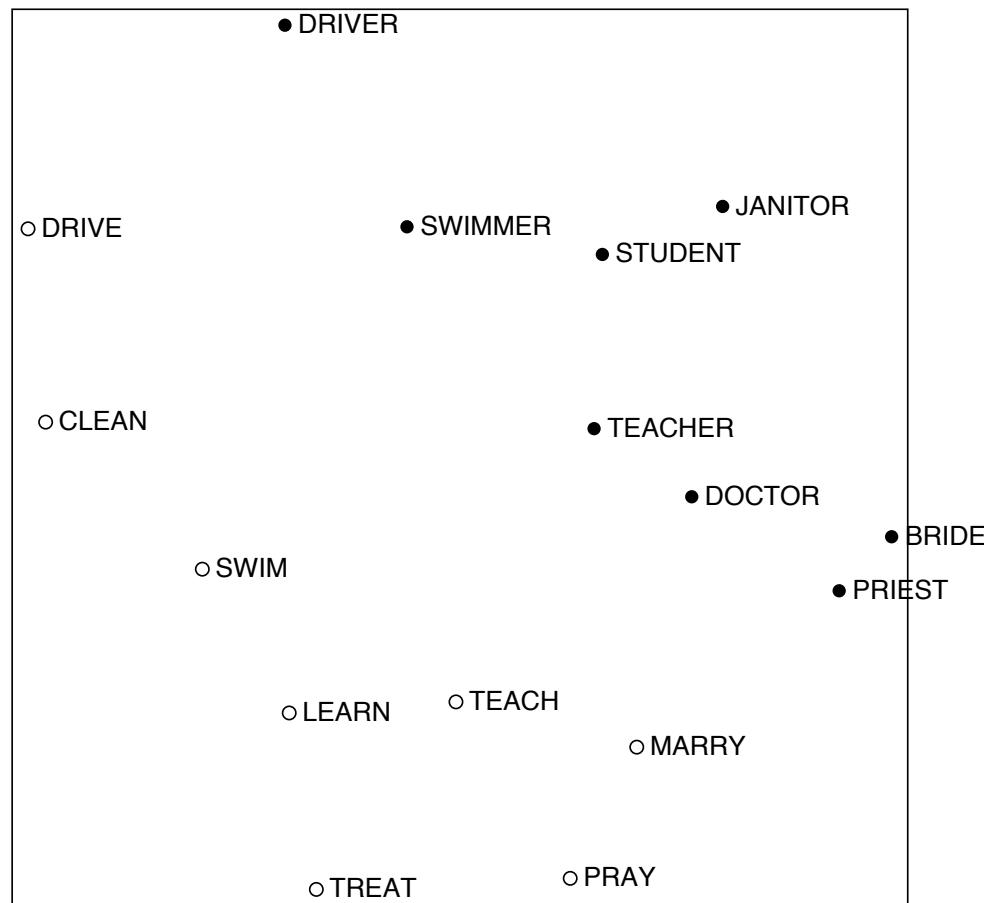
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting syntactic 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

Problems with SVD

Computational cost scales quadratically for $n \times m$ matrix:

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

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models

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 large counts

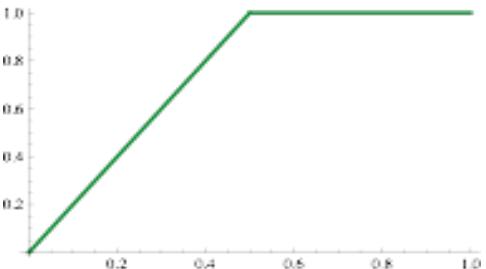
NNLM, HLBL, RNN, Skip-gram/
CBOW, (Bengio et al; Collobert & Weston;
Huang et al; Mnih & Hinton; Mikolov et al;
Mnih & Kavukcuoglu)

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

Combining the best of both worlds: GloVe

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

$f \sim$



- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors

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:

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

- 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



rana



leptodactylidae



eleutherodactylus

Linear Algebraic Structure of Word Senses, with Applications to Polysemy

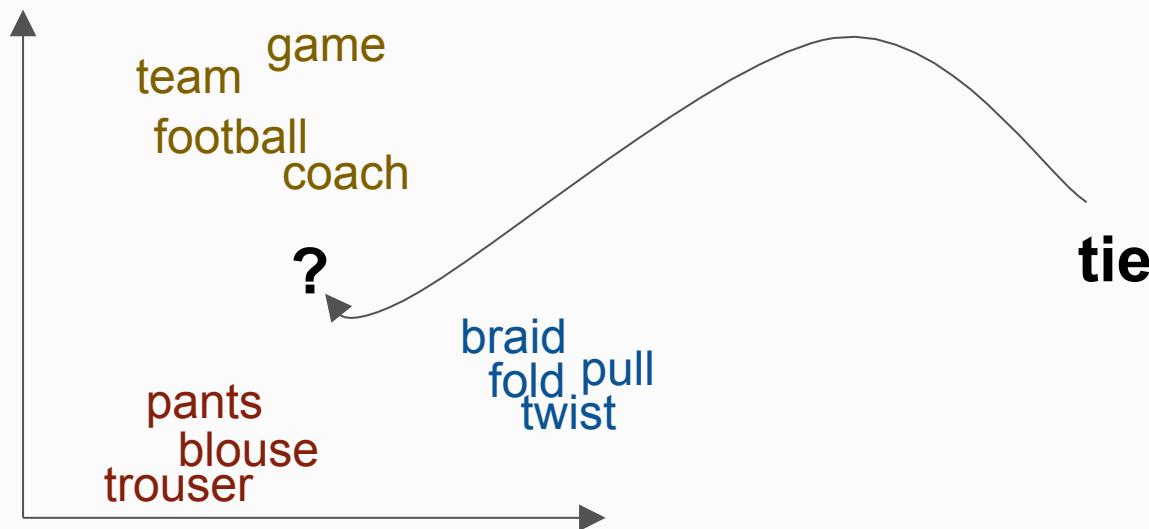
Sanjeev Arora, Yuanzhi Li, Yingyu Liang, Tengyu Ma, Andrej Risteski

Presented by: **Arun Chaganty**

Word vectors encode similarity.



What about polysemy?



1. Polysemous vectors are superpositioned.



2. Senses can be recovered by sparse coding

$$v = \sum_{i=0}^D \alpha_i A_i + \eta$$

Word vector Context vectors (~2000)
 >Selectors (< 5) noise

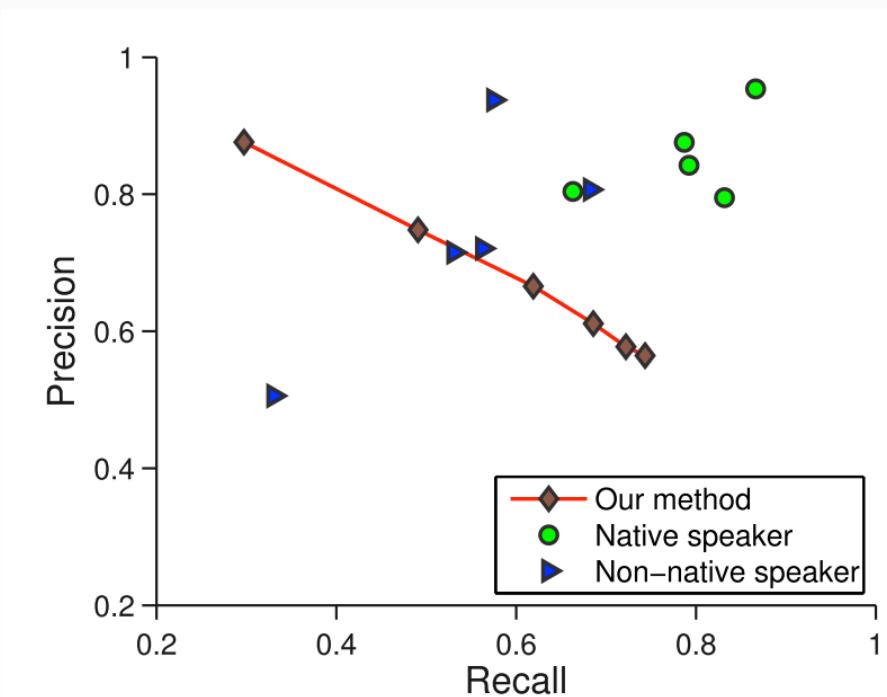
2. Senses can be recovered by sparse coding

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

3. Senses recovered are non-native English level

tie

1. Trouzers, blouse, pants
2. Season, teams, winning
3. Computer, mouse, keyboard
4. Bulb, light, flash



Summary

Word vectors can capture polysemy!

Word vectors are linear superposition of each sense vector.

Sense/context vectors can be recovered by sparse coding.

The senses recovered are about as good as a non-native English speakers!

Thanks!

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

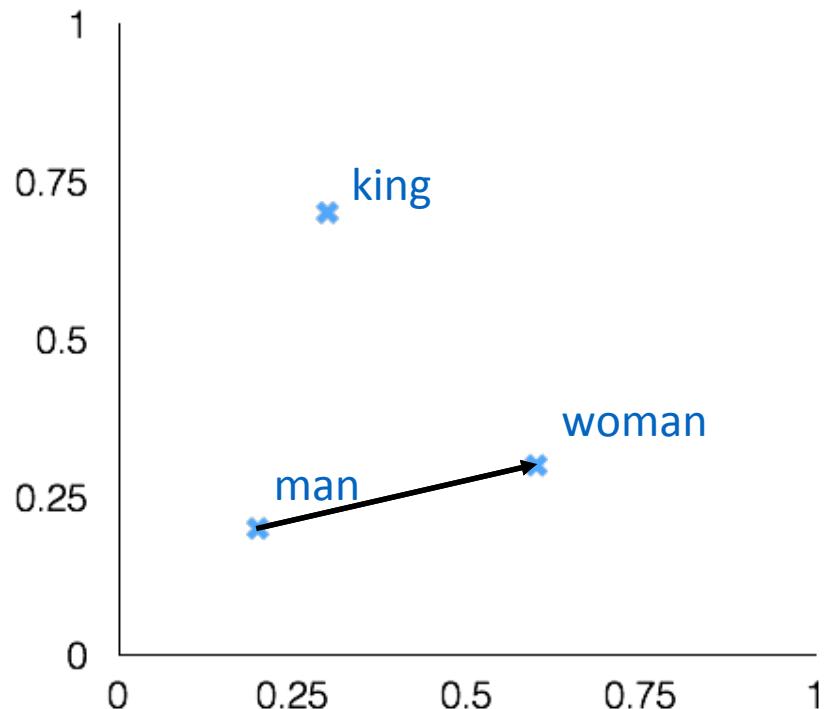
a:b :: c:?



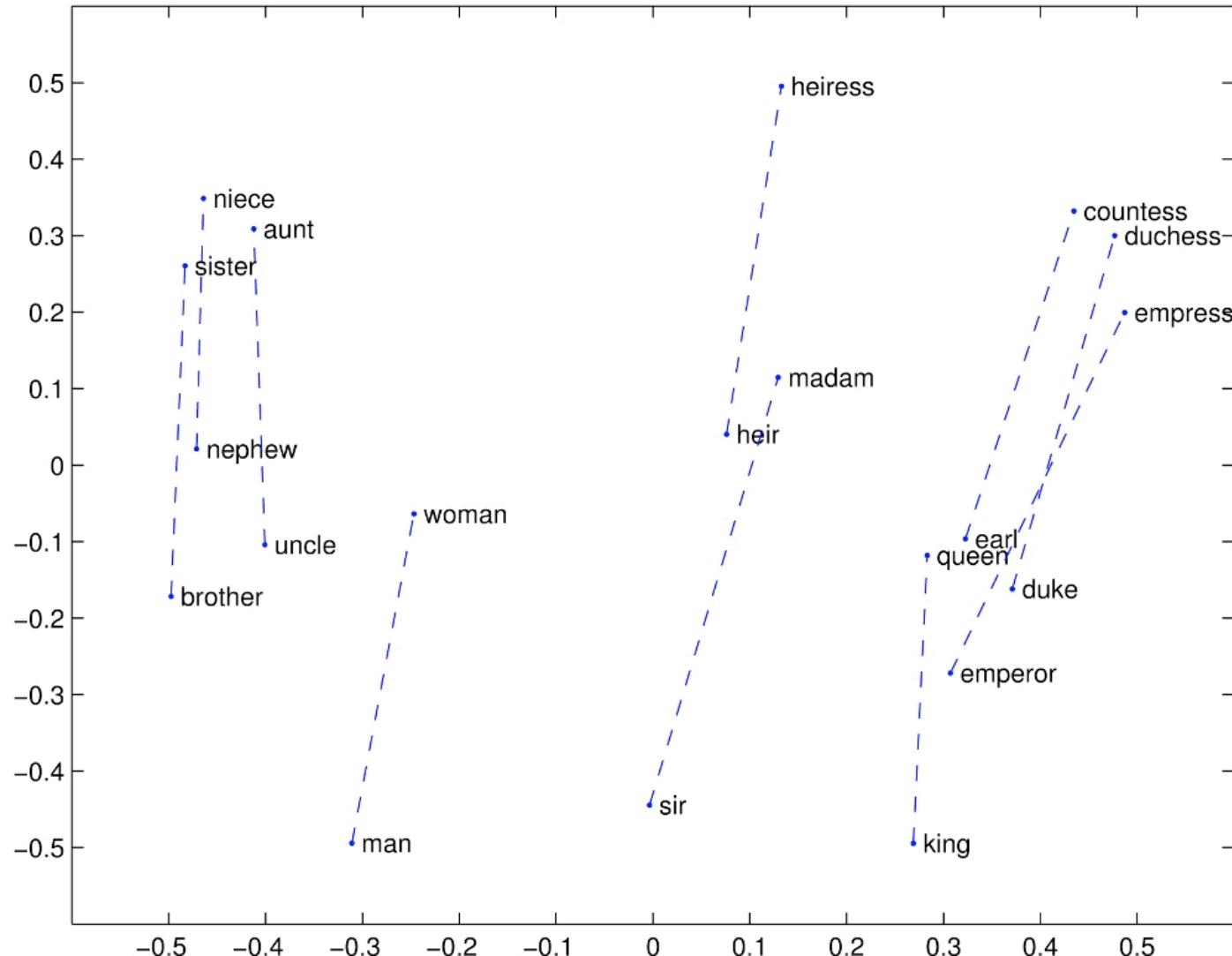
man:woman :: king:?

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

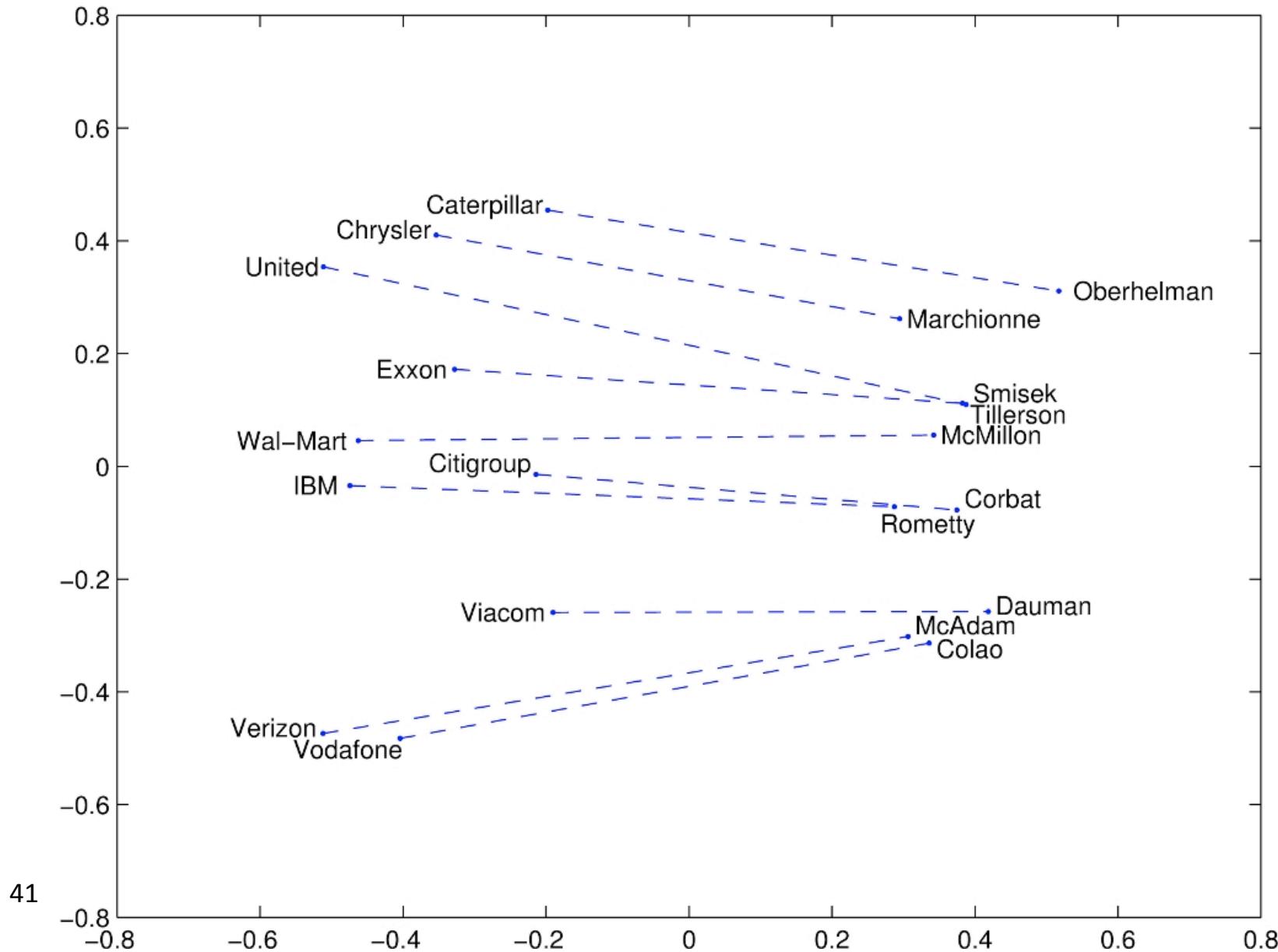
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



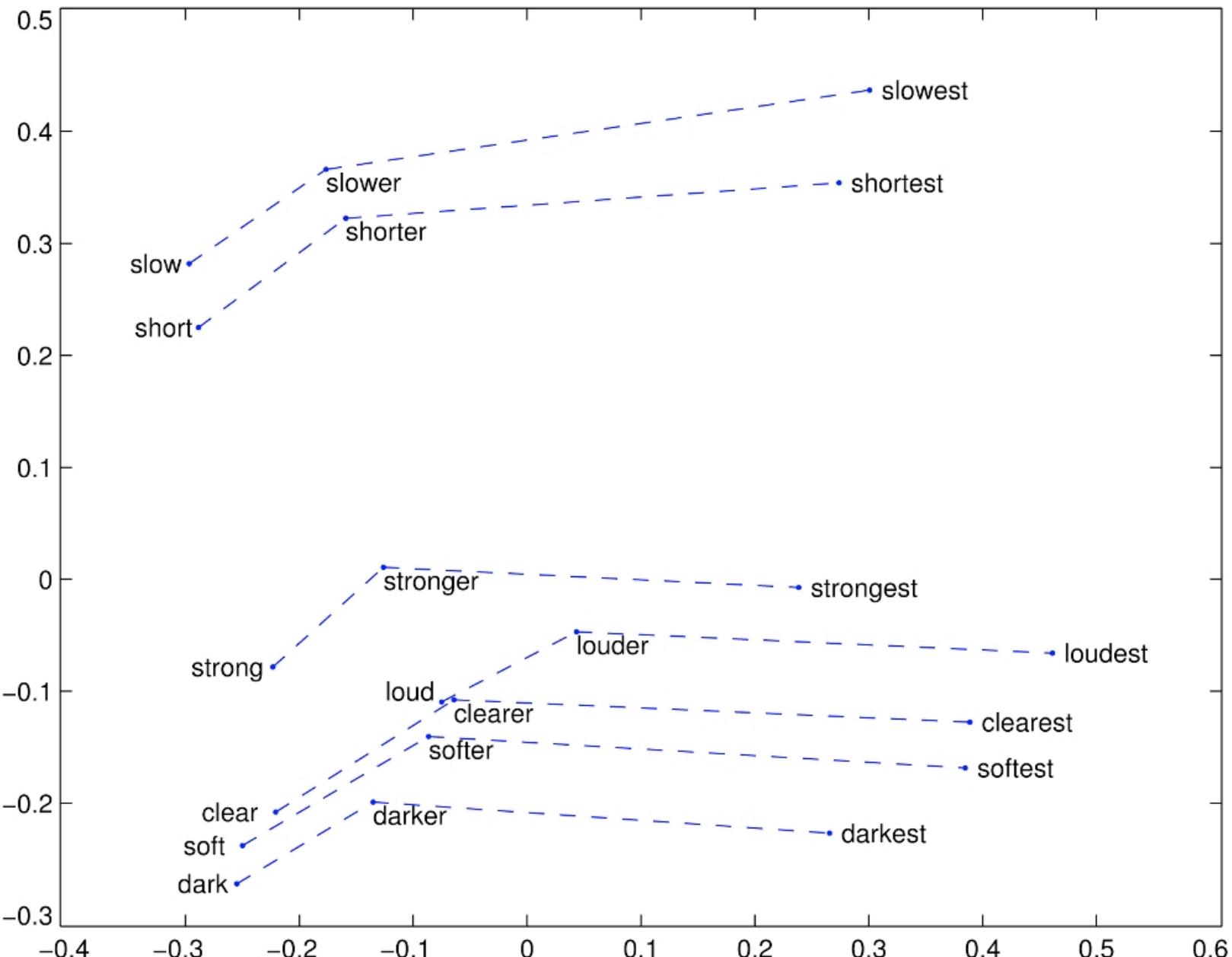
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Other fun word2vec analogies

<i>Expression</i>	<i>Nearest token</i>
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

problem: can change

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

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

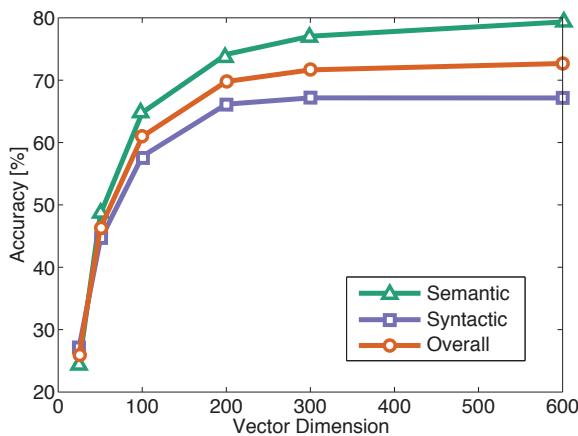
Analogy evaluation and hyperparameters

- Very careful analysis: Glove word vectors

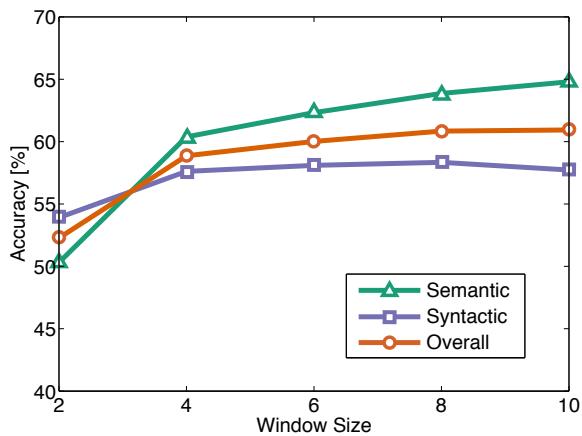
Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	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	<u>67.4</u>	65.7
SG [†]	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
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	<u>81.9</u>	<u>69.3</u>	<u>75.0</u>

Analogy evaluation and hyperparameters

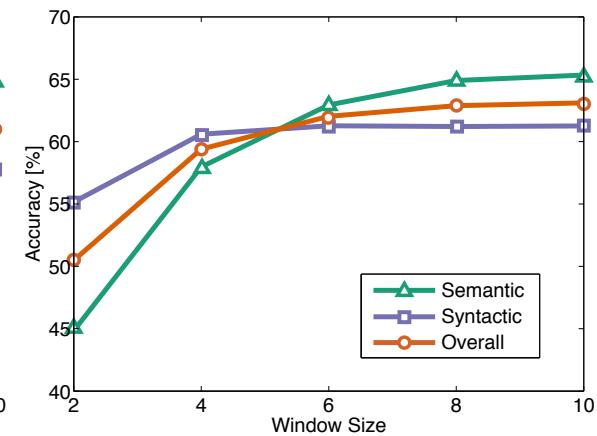
- Asymmetric context (only words to the left) are not as good



(a) Symmetric context



(b) Symmetric context

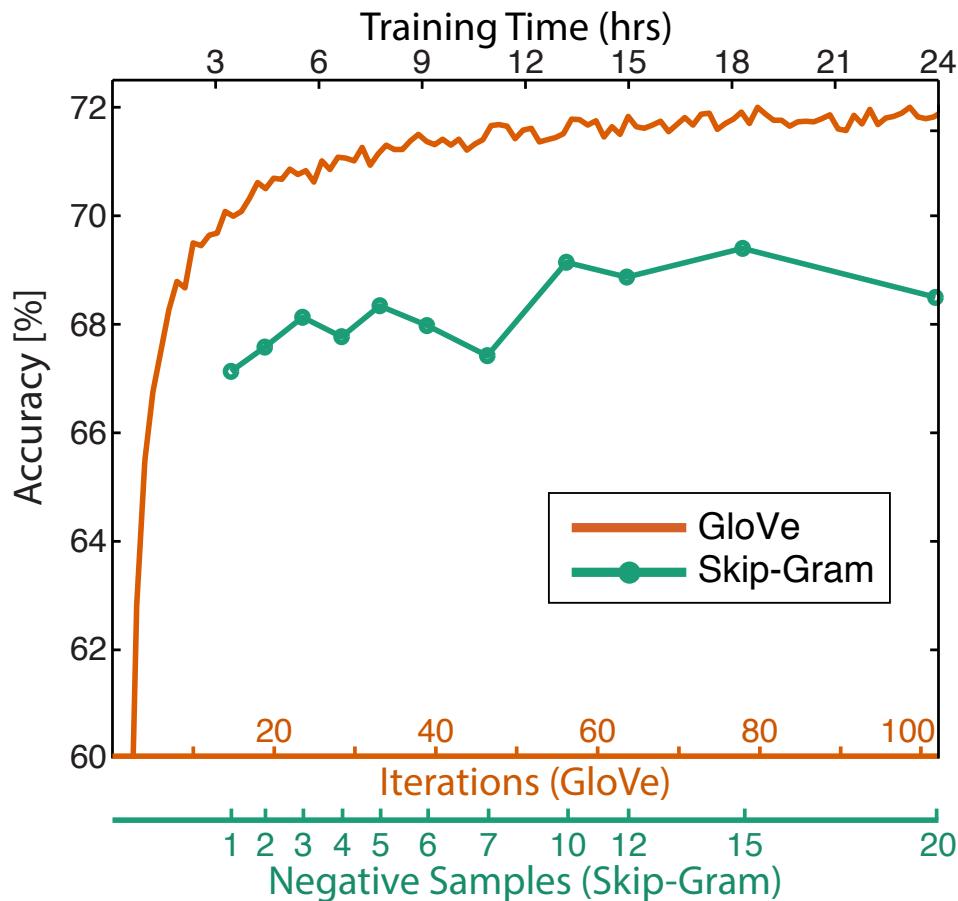


(c) Asymmetric context

- 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

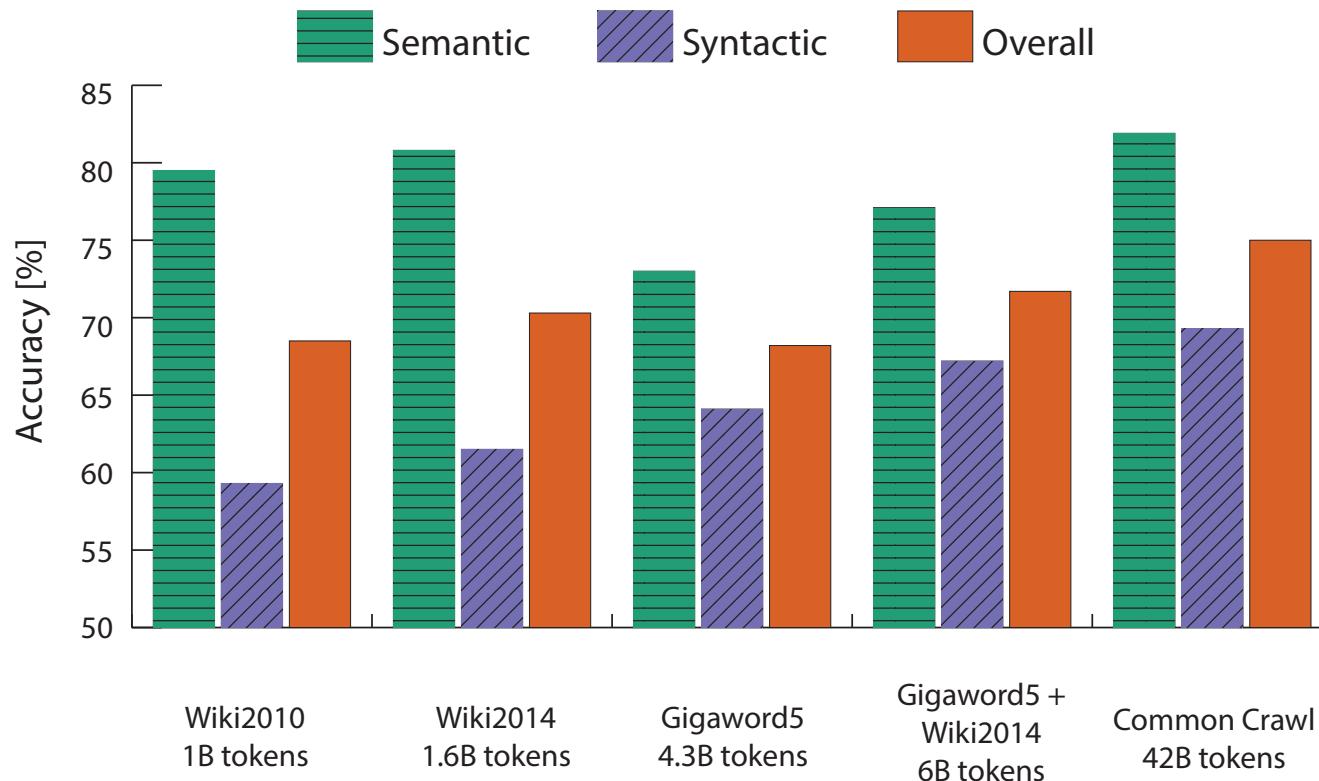
Analogy evaluation and hyperparameters

- 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	<u>72.7</u>	75.1	56.5	37.0
CBOW [†]	6B	57.2	<u>65.6</u>	68.2	57.0	32.5
SG [†]	6B	62.8	<u>65.2</u>	69.7	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	<u>53.9</u>	<u>38.1</u>
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

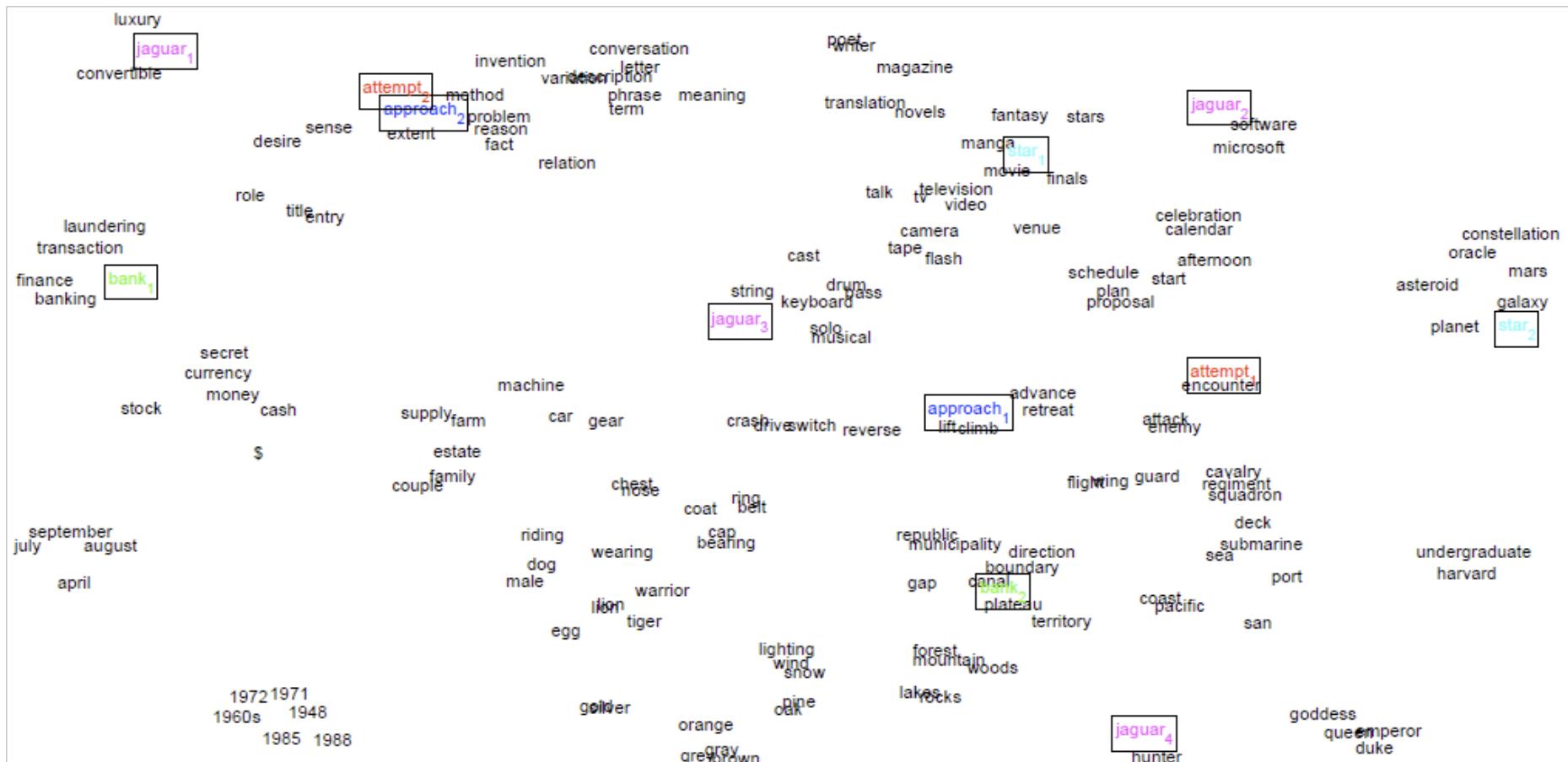
- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)

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 lecture:

- Word classification
- With context
- Neural networks
- So amazing. Wow!