Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 2: Word Vectors and Word Senses

Lecture Plan

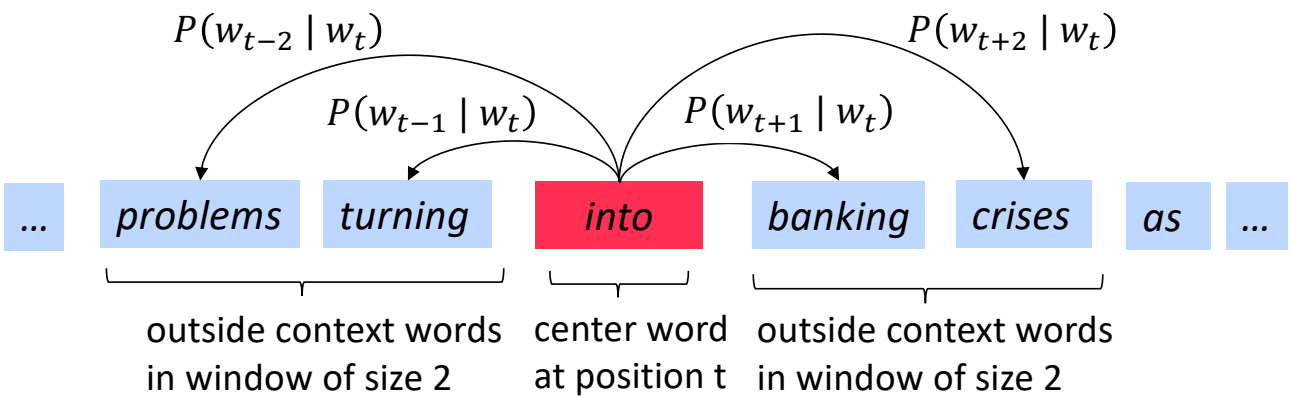
Lecture 2: Word Vectors and Word Senses

1. Finish looking at word vectors and word2vec (12 mins)
2. Optimization basics (8 mins)
3. Can we capture this essence more effectively by counting? (15m)
4. The GloVe model of word vectors (10 min)
5. Evaluating word vectors (15 mins)
6. Word senses (5 mins)
7. The course (5 mins)

Goal: be able to read word embeddings papers by the end of class

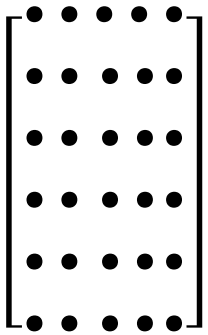
1. Review: Main idea of word2vec

- Iterate through each word of the whole corpus
- Predict surrounding words using word vectors



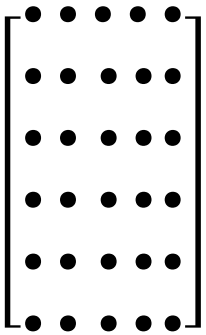
- $P(o|c) = \dfrac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$
- Update vectors so you can predict well

Word2vec parameters and computations



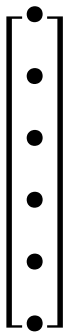
U

outside



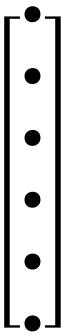
V

center



$U.v_4{}^T$

dot product



$\text{softmax}(U.v_4{}^T)$

probabilities

Word vectors are represented as rows- In pytorch, tensorflow etc. Unlike in maths classes.

!

Same predictions at each position

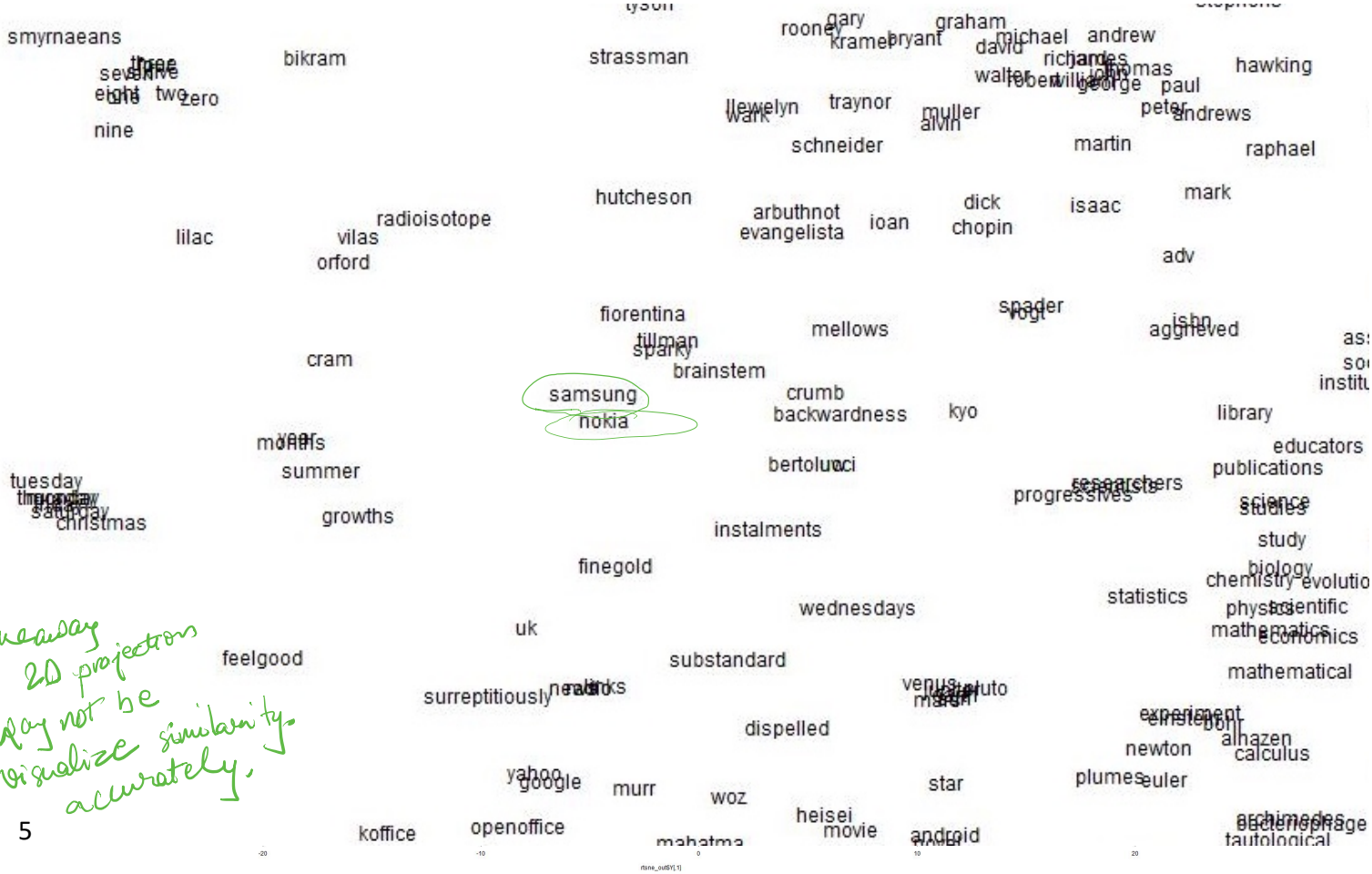
Note: the, I, a and other high frequency words that may dominate occurence.

One Fix: Cut off top K high frequency words?

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (fairly often)

This simple approach can capture meanings.. Example, Jupyter notebook - analogy ( ).

Word2vec maximizes objective function by putting similar words nearby in space

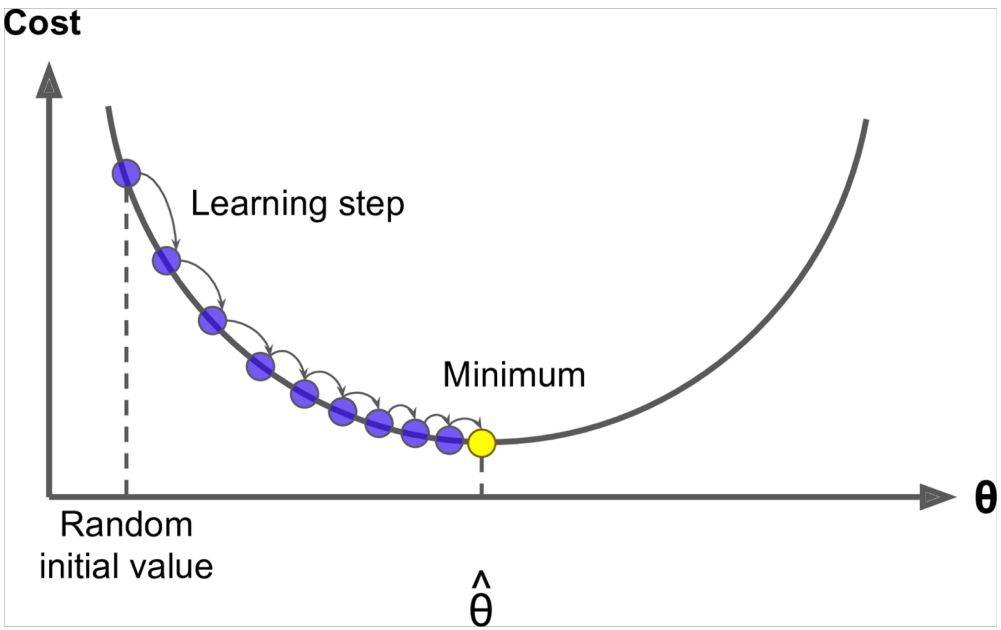


Takeaway 2D projections may not be visualize similarity accurately.

2. Optimization: Gradient Descent

- We have a cost function $J(\theta)$ we want to minimize
- **Gradient Descent** is an algorithm to minimize $J(\theta)$
- Idea: for current value of θ, calculate gradient of $J(\theta)$, then take small step in the direction of negative gradient. Repeat.

In high Dim, non-convex.



Note: Our objectives may not be convex like this

Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

we can overshoot if α is too small

- Update equation (for a single parameter):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- Algorithm:

```
while True:
    theta_grad = evaluate_gradient(J,corpus,theta)
    theta = theta - alpha * theta_grad
```

Stochastic Gradient Descent

- Problem: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!

- **Very** bad idea for pretty much all neural nets!
- Solution: **Stochastic gradient descent (SGD)**
  - Repeatedly sample windows, and update after each one
- Algorithm:

→ noisy estimate of gradient → gradually approach minima.
→ Each update different center word.

```
while True:
    window = sample_window(corpus)
    theta_grad = evaluate_gradient(J,window,theta)
    theta = theta - alpha * theta_grad
```

In practice mini-batch → 32, 64, 128 etc. (power of 2)

Stochastic gradients with word vectors!

- Iteratively 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!

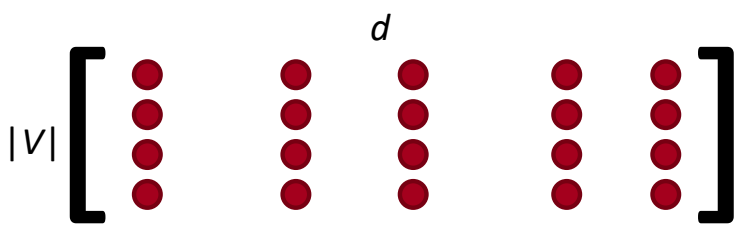
$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

→ very sparse parameter update. (due to window size. 100~150 from millions)

Stochastic gradients with word vectors!

- We might only update the word vectors that actually appear!

- Solution: either you need sparse matrix update operations to only update certain rows 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!

1b. Word2vec: More details

Why two vectors? → Easier optimization. Average both at the end

- But can do it with just one vector per word

Two model variants:

1. Skip-grams (SG)

   Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

   Predict center word from (bag of) context words

We presented: **Skip-gram model**

Additional efficiency in training:

1. Negative sampling

So far: Focus on **naïve softmax** (simpler, but expensive, training method)

The skip-gram model with negative sampling (HW2)

- The normalization factor is too computationally expensive.

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

If million vocab.?
→ expensive.
→ i.e. low.

- Hence, in standard word2vec and HW2 you 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 several noise pairs (the center word paired with a random word)

Higher probability to one appear around a center word
Lower " " " randomly sampled word.

The skip-gram model with negative sampling (HW2)

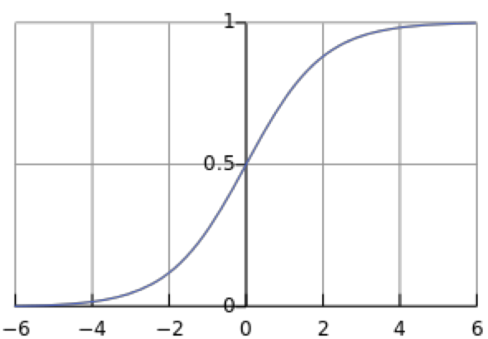
- From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)
- Overall objective function (they maximize): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

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

Notation might be different -

binary softmax.

- 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 →



The skip-gram model with negative sampling (HW2)

- Notation more similar to class and HW2:

randomly chosen K words.

$$J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$$

K = 10, 15

[negate → sigmoid is symmetric].

- We take *k* negative samples (using word probabilities)
- Maximize probability that real outside word appears, minimize prob. that random words appear around center word

How to sample? Random? Unigram distribution (count based distribution)

decreasing common words & increasing rare words (hyperparameter)

- $P(w)=U(w)^{3/4}/Z$, → Normalization
  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

3. But why not capture co-occurrence counts directly?

→ same thing as word2vec [window] but counting based

Traditional NLP (before 2013 NN)

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.

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

similarity bet^n word vectors. to see if two words have similar meaning.? works well but.

Problems with simple co-occurrence vectors

Increase in size with vocabulary

Very high dimensional: requires a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

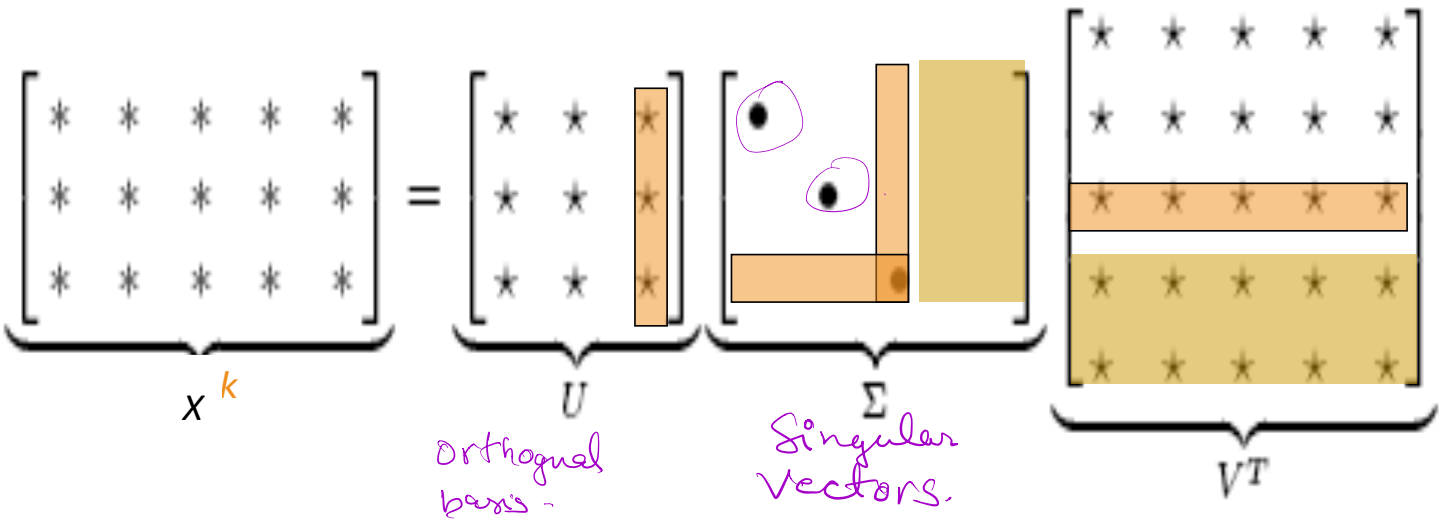
Solution: Low dimensional vectors

- 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 (HW1)

Singular Value Decomposition of co-occurrence matrix *X*

Factorizes *X* into $U\Sigma V^T$, where *U* and *V* are orthonormal



Retain only *k* singular values, in order to generalize.

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

Classic linear algebra result. Expensive to compute for large matrices.

Simple SVD word vectors in Python

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

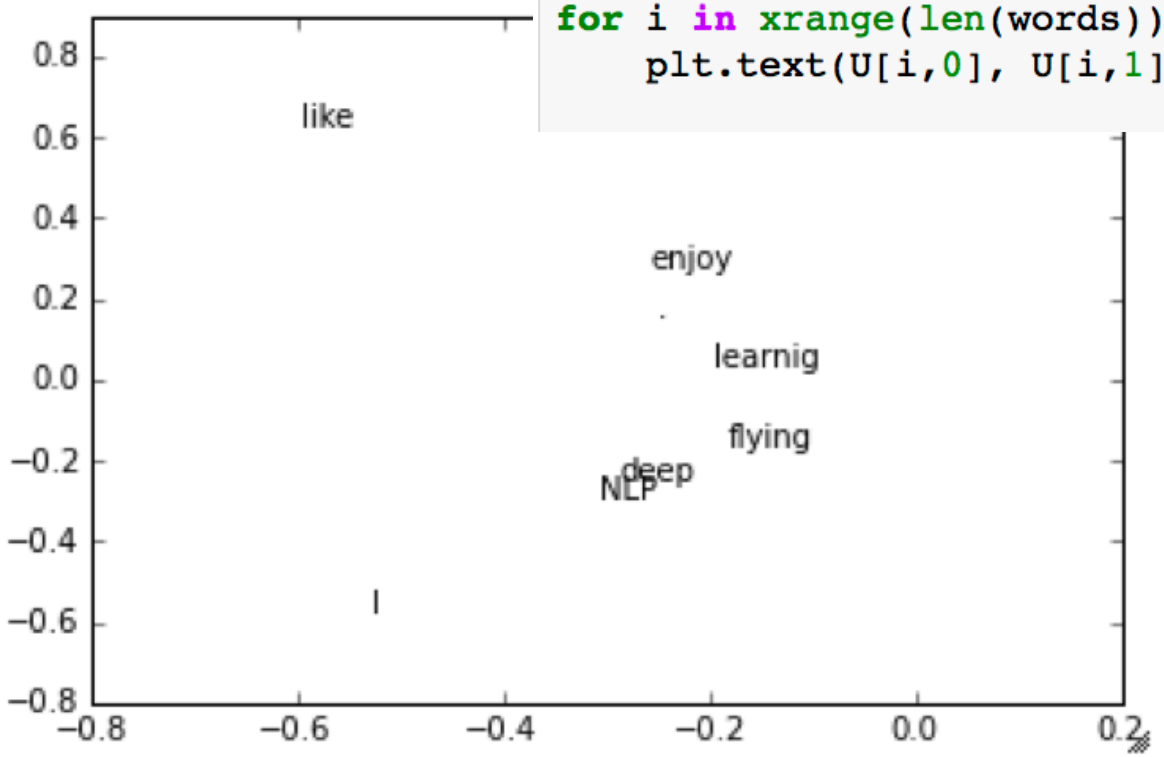
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],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



Latent Semantic analysis / Indexing (LSA).

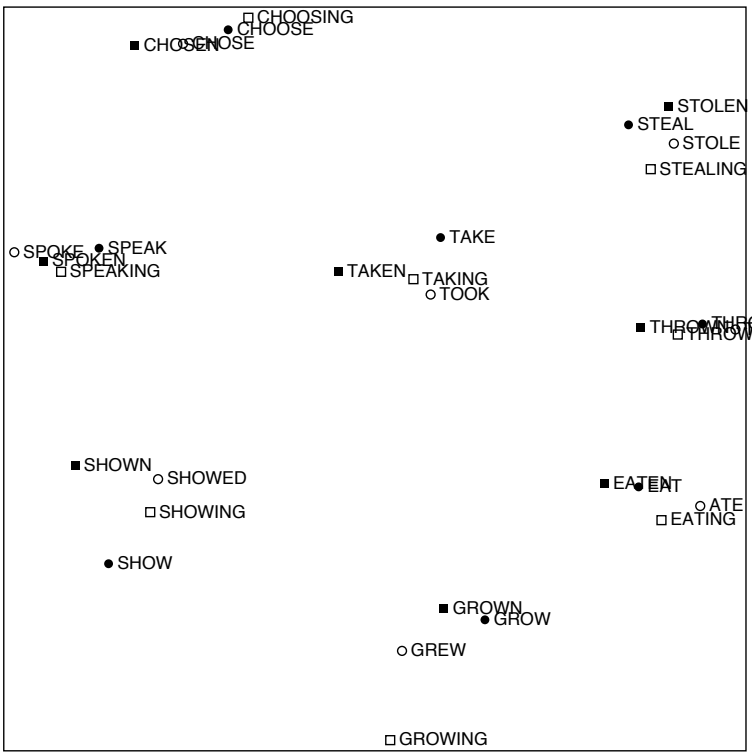
→ worked a bit but not well.

Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help ***a lot***

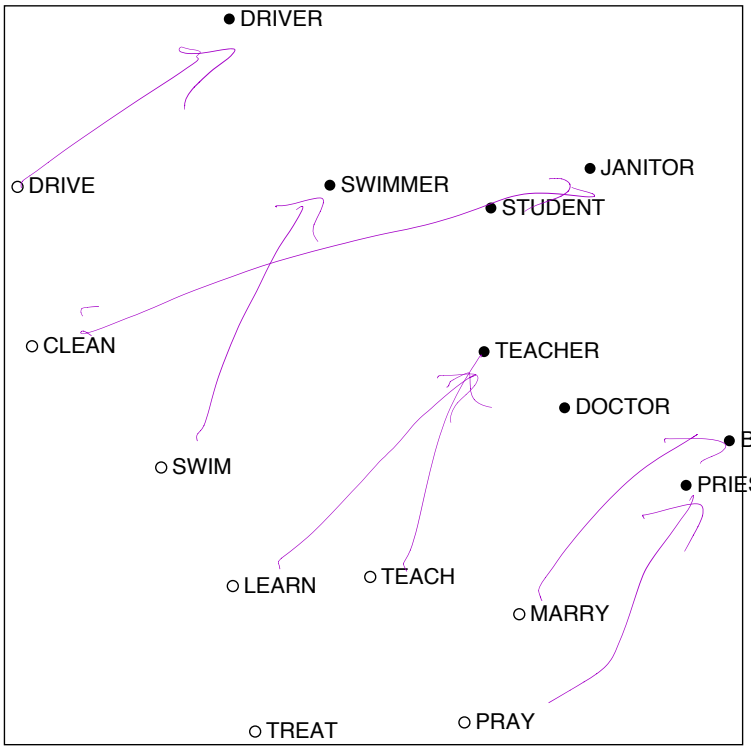
- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
  - min(X,t), with t ≈ 100
  - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Interesting syntactic patterns emerge in the vectors



COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Interesting semantic patterns emerge in the vectors



Verb → Noun.
(doer of verb).

Linearity property in vector space → will do better in analogy.

COALS model from
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. ms., 2005

Count based vs. direct prediction

Counting & transforming count.

- 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)

- 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

word.

	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, Socher, and Manning, EMNLP 2014]

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

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

Ratio of co-occurrence probabilities. (should be linear.

Encoding meaning in vector differences

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$ [to make linear]

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

Combining the best of both worlds GloVe [Pennington et al., EMNLP 2014]

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

dot product should be similar to log

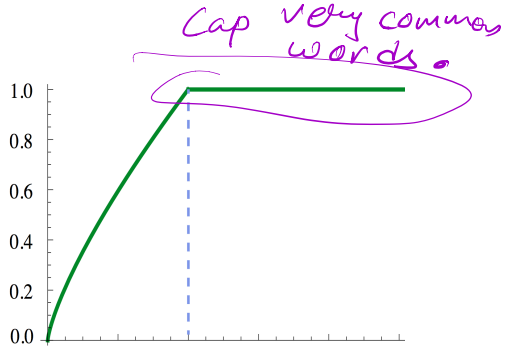
to regularize. Overall common or rare words

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

squared loss.

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

$f \sim$



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 (work on real system) Web search, QA.
  - 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

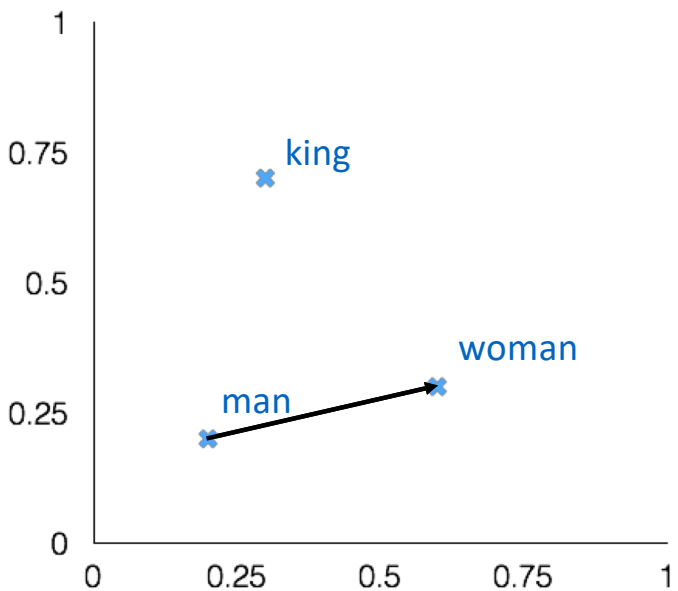
- Word Vector Analogies

a:b :: c:?

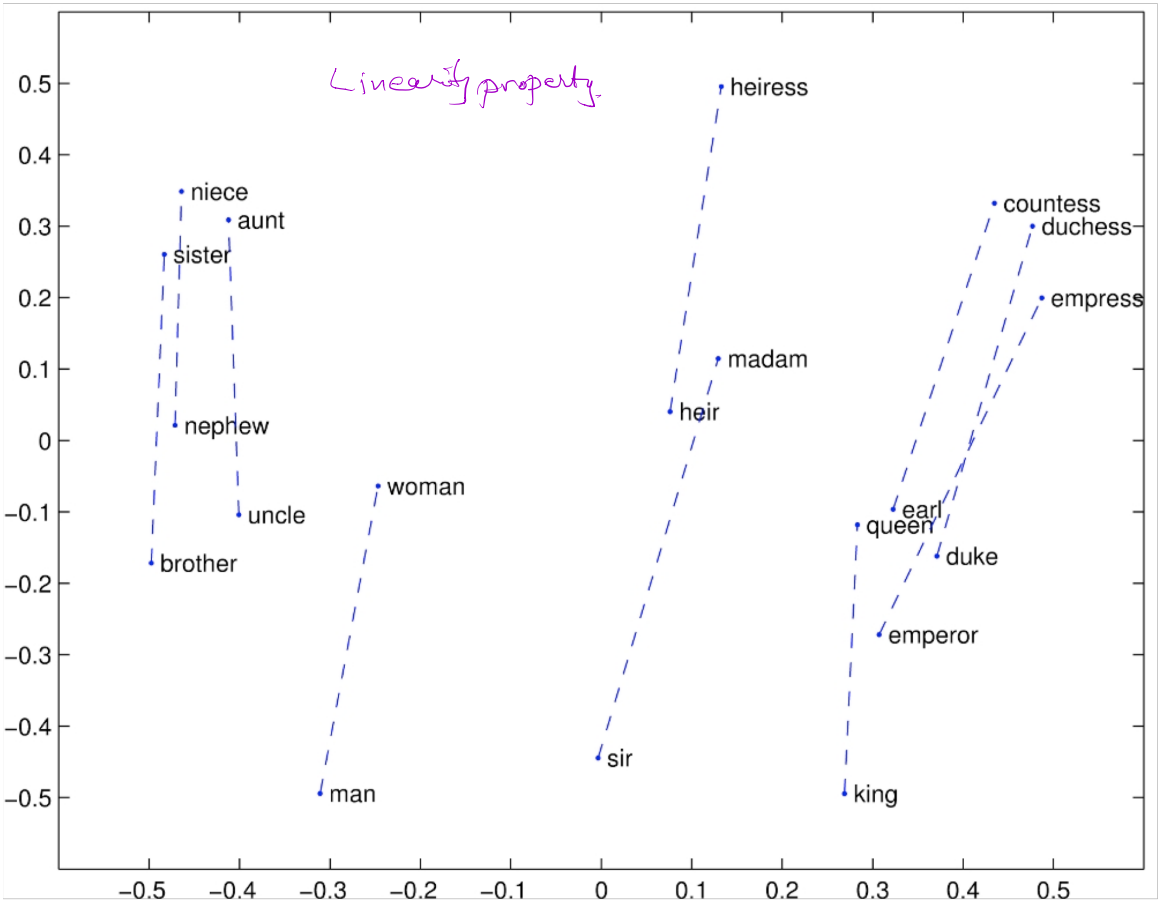
man:woman :: king:?

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

- 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?

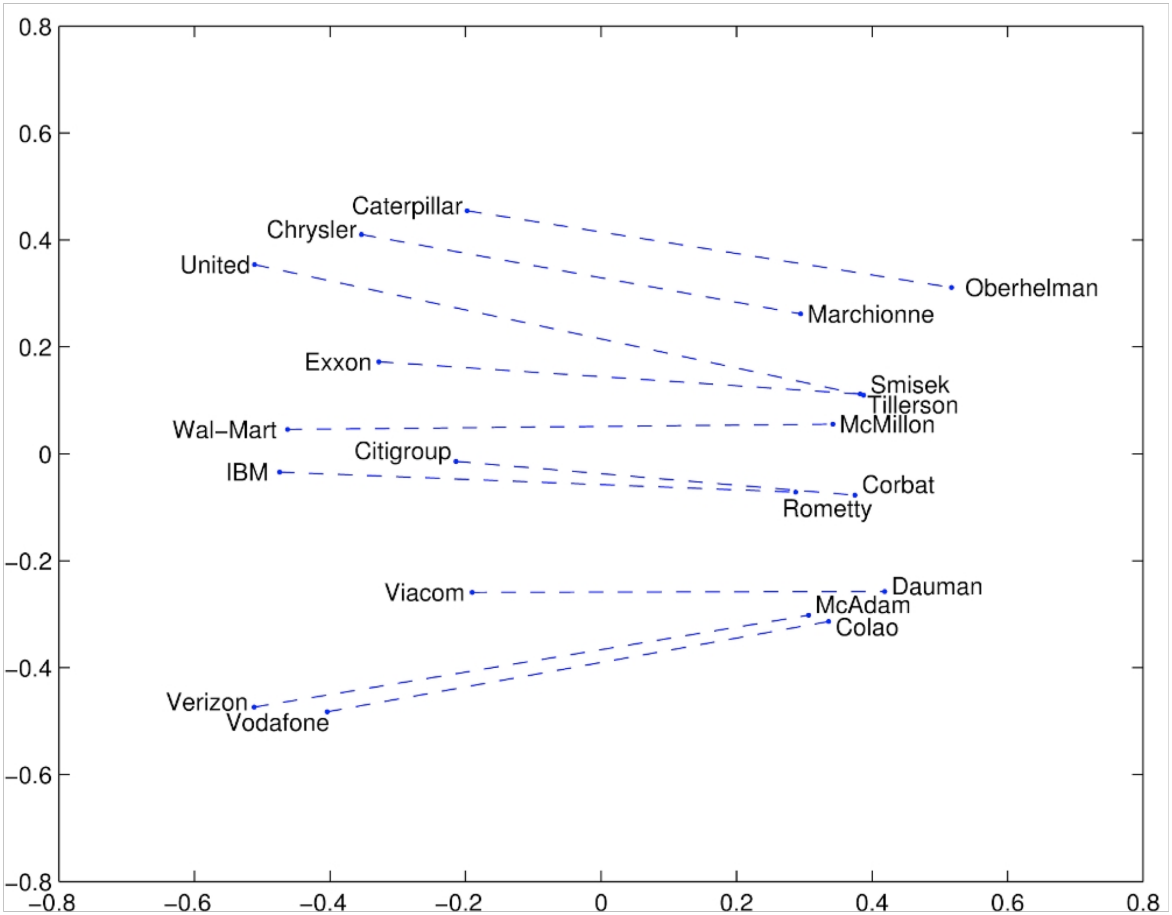


Glove Visualizations



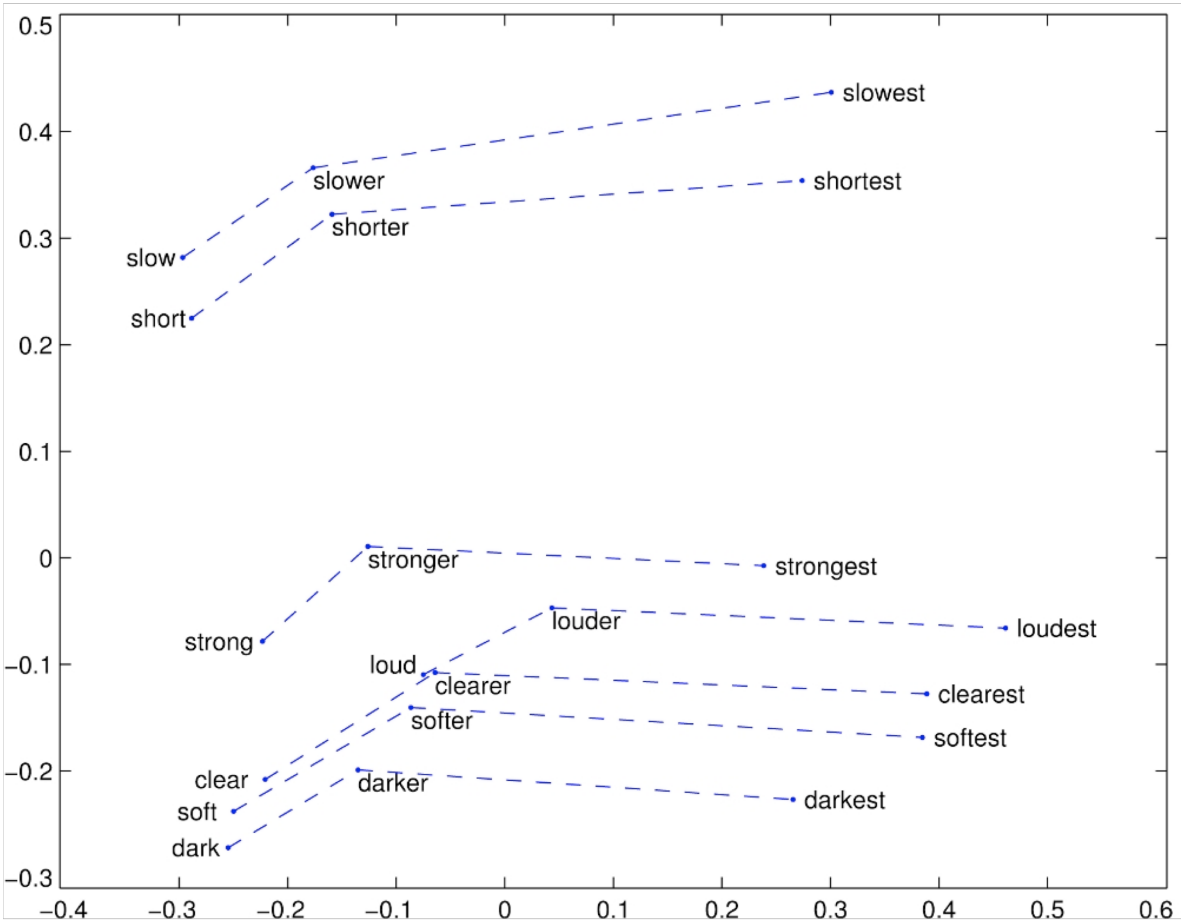
Gender relation

Glove Visualizations: Company - CEO



Company
CEO

Glove Visualizations: Superlatives



Superlative

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

: 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

Analogy evaluation and hyperparameters

- Glove word vectors evaluation

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**

manipulation of the count before SVD,

Bigger Dim ⇒ Better

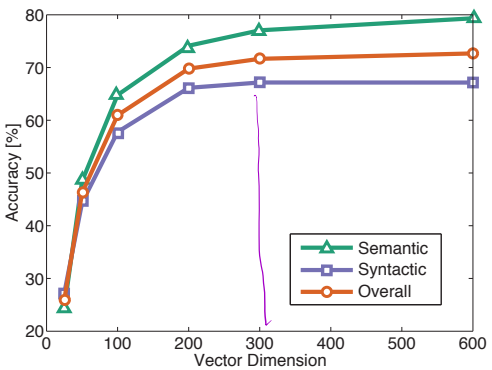
Corpus Size.

Analogy evaluation and hyperparameters

This is why ppl use 300 dim.

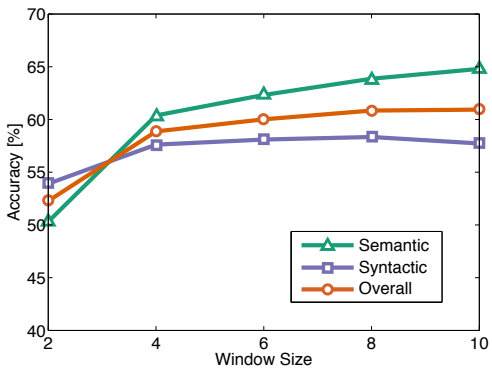
Syntactic are locals.

If context on one size, results not good.



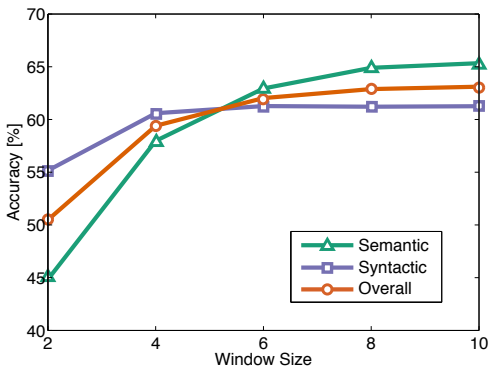
(a) Symmetric context

Dimensionality



(b) Symmetric context

Window size



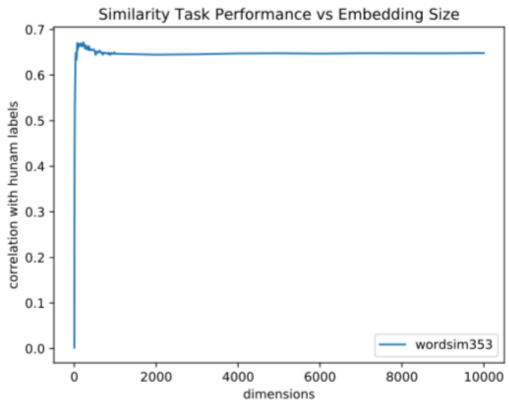
(c) Asymmetric context

Window size

- Good dimension is ~300
- Asymmetric context (only words to the left) are not as good
- But this might be different for downstream tasks!
- Window size of 8 around each center word is good for Glove vectors

On the Dimensionality of Word Embedding

[Zi Yin and Yuanyuan Shen, NeurIPS 2018]



(b) WordSim353 Test

performance stays same as you increase dim

https://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf

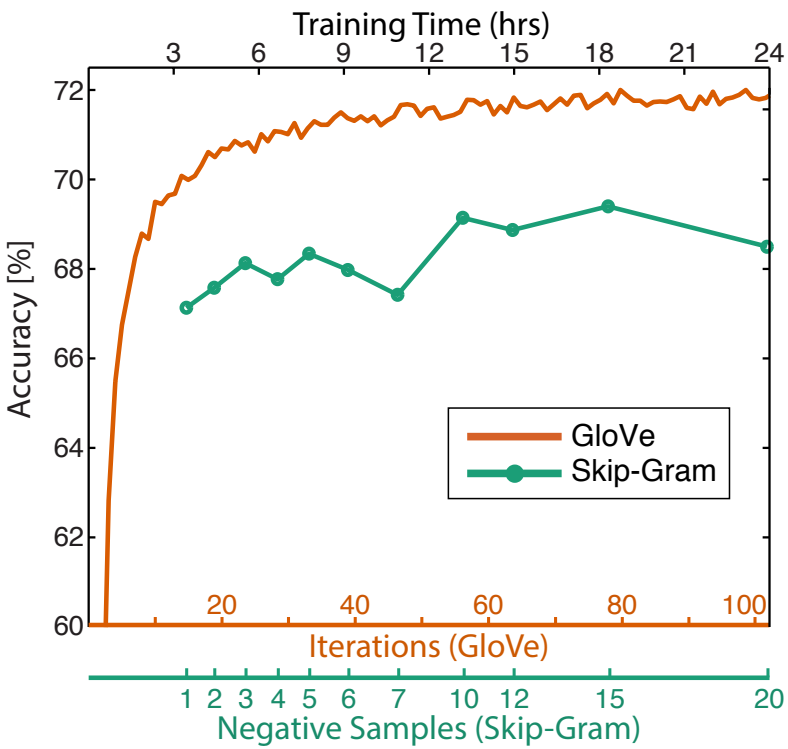
Using matrix perturbation theory, reveal a fundamental bias-variance trade-off in dimensionality selection for word embeddings

Table 3: PIP loss minimizing dimensionalities and intervals for GloVe on Text8 corpus

Surrogate Matrix	arg min	+5% interval	+10% interval	+20% interval	+50% interval	WS353	MT771	Analogy
GloVe (log-count)	719	[290,1286]	[160,1663]	[55,2426]	[5,2426]	220	860	560

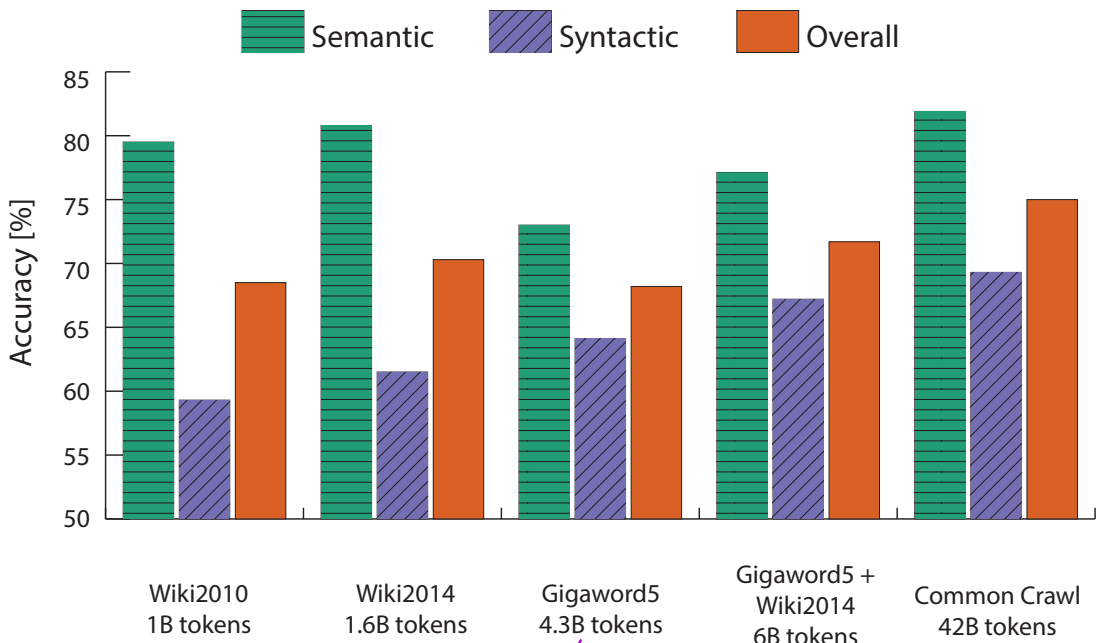
Analogy evaluation and hyperparameters

- More training time helps



Analogy evaluation and hyperparameters

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



of news paper article.

Not much improvement

Makes Intuitive sense. Wikipedia has association information. while news data don't.

Another intrinsic word vector evaluation

(Comparing with human).

- 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
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

Psychology.

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)

Word senses and word sense ambiguity

- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time

clinton → Bill? hillary? (ambiguity)

Time for old words with lots of meanings. may not be true for technical words with one meaning.

- Example: pike
  - → fish
  - → large spear.
  - → gymnastic move.
  - → road

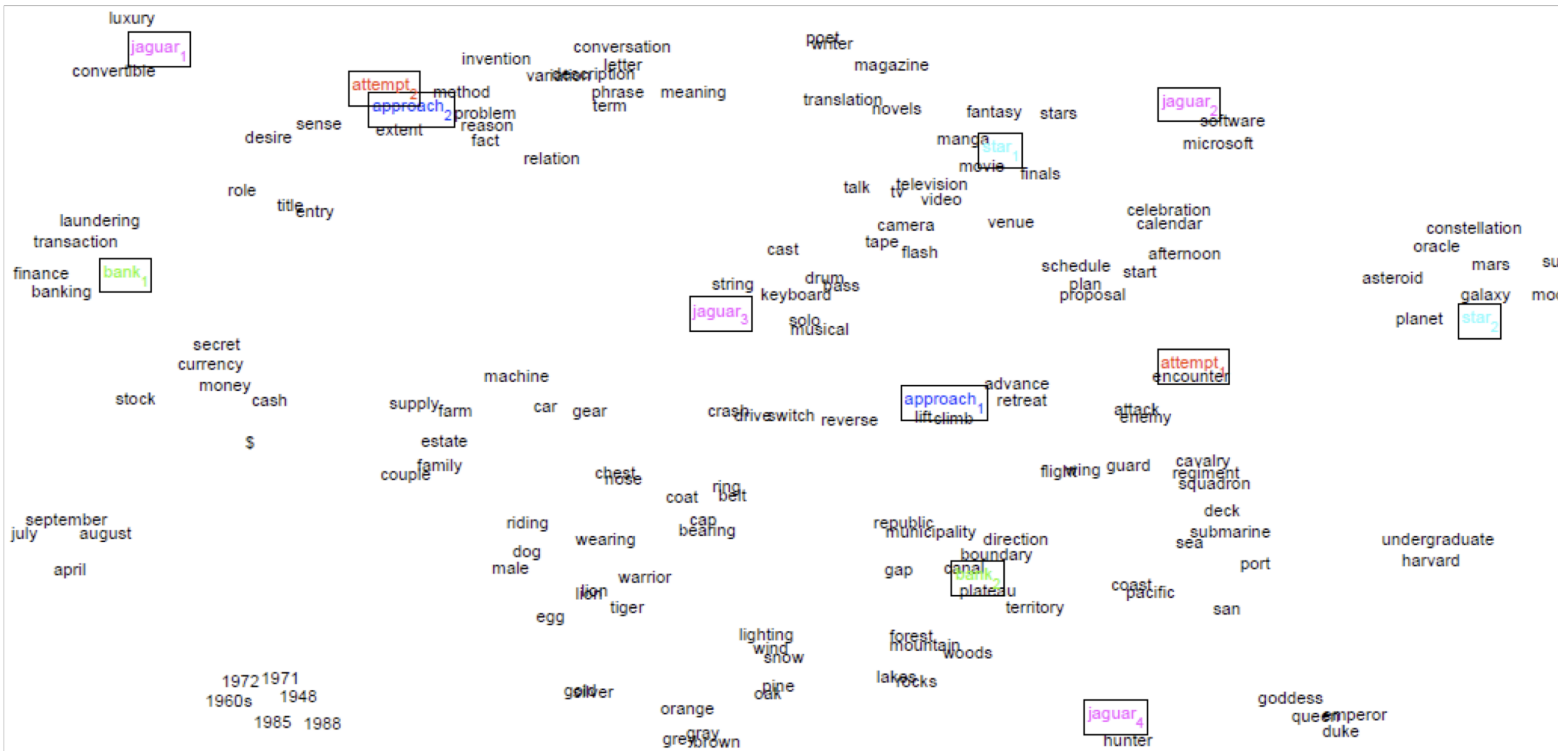
- Does one vector capture all these meanings or do we have a mess?

pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!

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



Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$
- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f (weighted by frequency).
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)

word senses.

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

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**

You can use word vectors to any task & it helps improve performance.

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

Course plan: coming weeks

Week 2: Neural Net Fundamentals

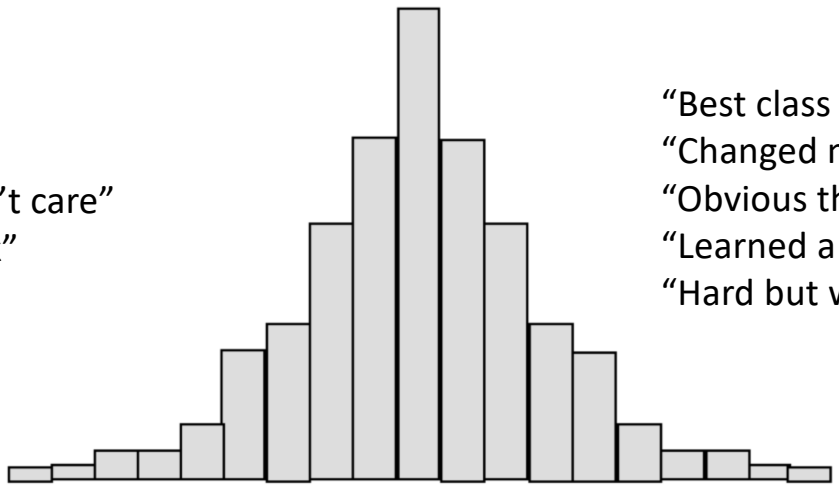
- We concentrate on understanding (deep, multi-layer) neural networks and how they can be trained (learned from data) using backpropagation (the judicious application of calculus)
- We'll look at an NLP classifier that adds context by taking in windows around a word and classifies the center word (not just representing it across all windows)!

Week 3: We learn some natural language processing

- We learn about putting syntactic structure (dependency parses) over sentence (this is HW3!)
- We develop the notion of the probability of a sentence (a probabilistic language model) and why it is really useful

A note on your experience 😀

"Terrible class"
"Don't take it"
"Instructors don't care"
"Too much work"



"Best class at Stanford"
"Changed my life"
"Obvious that instructors care"
"Learned a ton"
"Hard but worth it"

- This is a hard, advanced, graduate level class
- I and all the TAs really care about your success in this class
- Give Feedback. Take responsibility for holes in your knowledge
- **Come to office hours/help sessions**

Office Hours / Help sessions

- **Come to office hours/help sessions!**
  - Come to discuss final project ideas as well as the homeworks
  - Try to come early, often and off-cycle
- **Help sessions:** daily, at various times, see calendar
  - First one is tonight: 6:30–9:00pm
  - Gates Basement B21 (and B30) – bring your student ID
  - Attending in person: Just show up! Our friendly course staff will be on hand to assist you
  - SCPD/remote access: Use queuestatus
- **Chris's office hours:**
  - Mon 1–3pm. Come along this Monday?