Natural Language Processing with Deep Learning

CS224N/Ling284

Christopher Manning

Lecture 2: Word Vectors and Word Senses

Lecture Plan

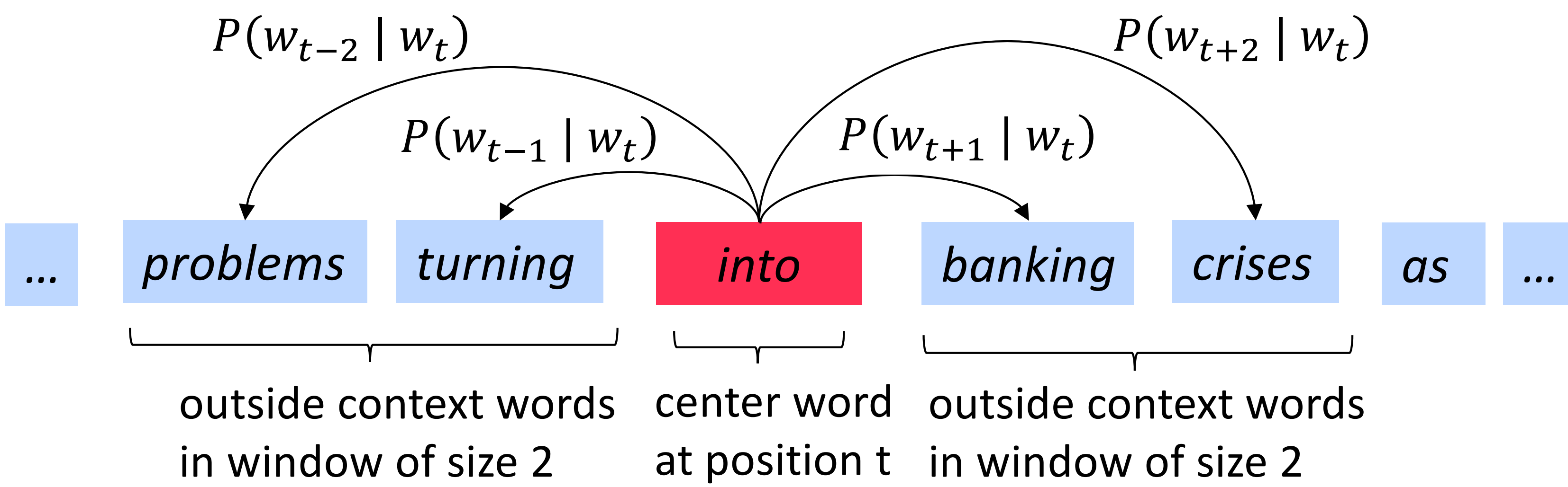
Lecture 2: Word Vectors and Word Senses

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

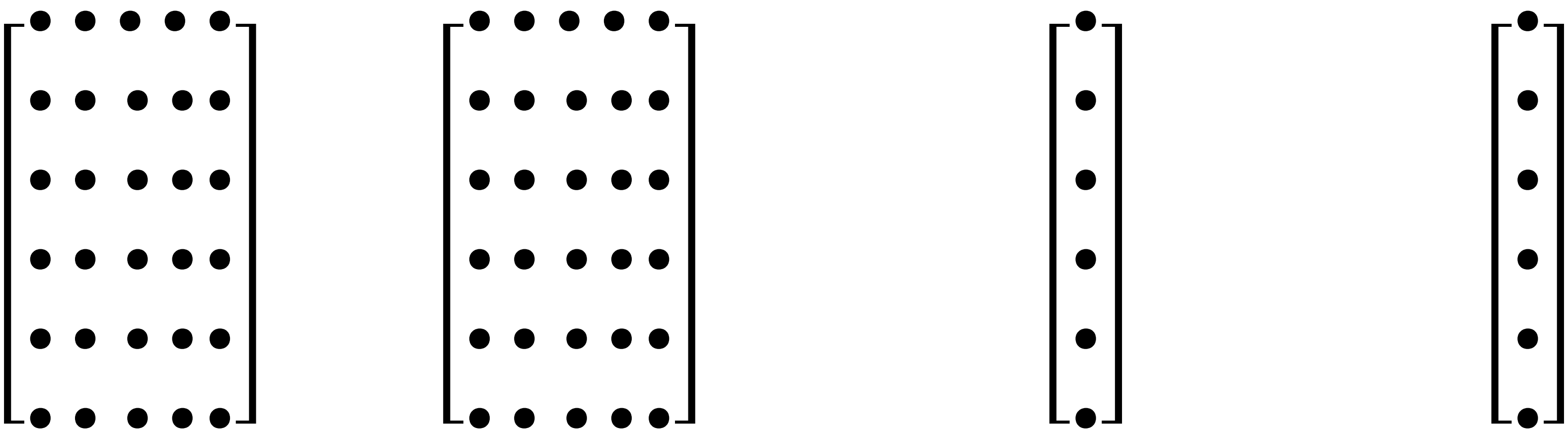


- $$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

그냥 softmax와 같다.

- Update vectors so you can predict well

Word2vec parameters and computations



U outside

V center

$U.{v_4}^T$ dot product

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

6 words
5 dimension

!

Same predictions at each position

위 그림과 같이 tensorflow나 pytorch 등 모든 딥러닝 라이브러리에서도 row 방향으로 word vector를 저장하고 있다.

stopwords는 모든 단어와의 dot product 값이 높게 측정되므로 삭제하는 것이 좋다.

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

Word2vec maximizes objective function by putting similar words nearby in space

nokia가 Finland와 가까이에 있길 예상했을텐데...
2 dim에서는 이런 다양한 측면을 포함하기 굉장히 어렵다.
high dimensional vector space는 매우 unintuitive하다. (because a word can be close to lots of other words in different directions)

Vector space는 단순히 similarity space뿐만 아니라, certain meaning이 포함된 direction space를 가질 것이다.

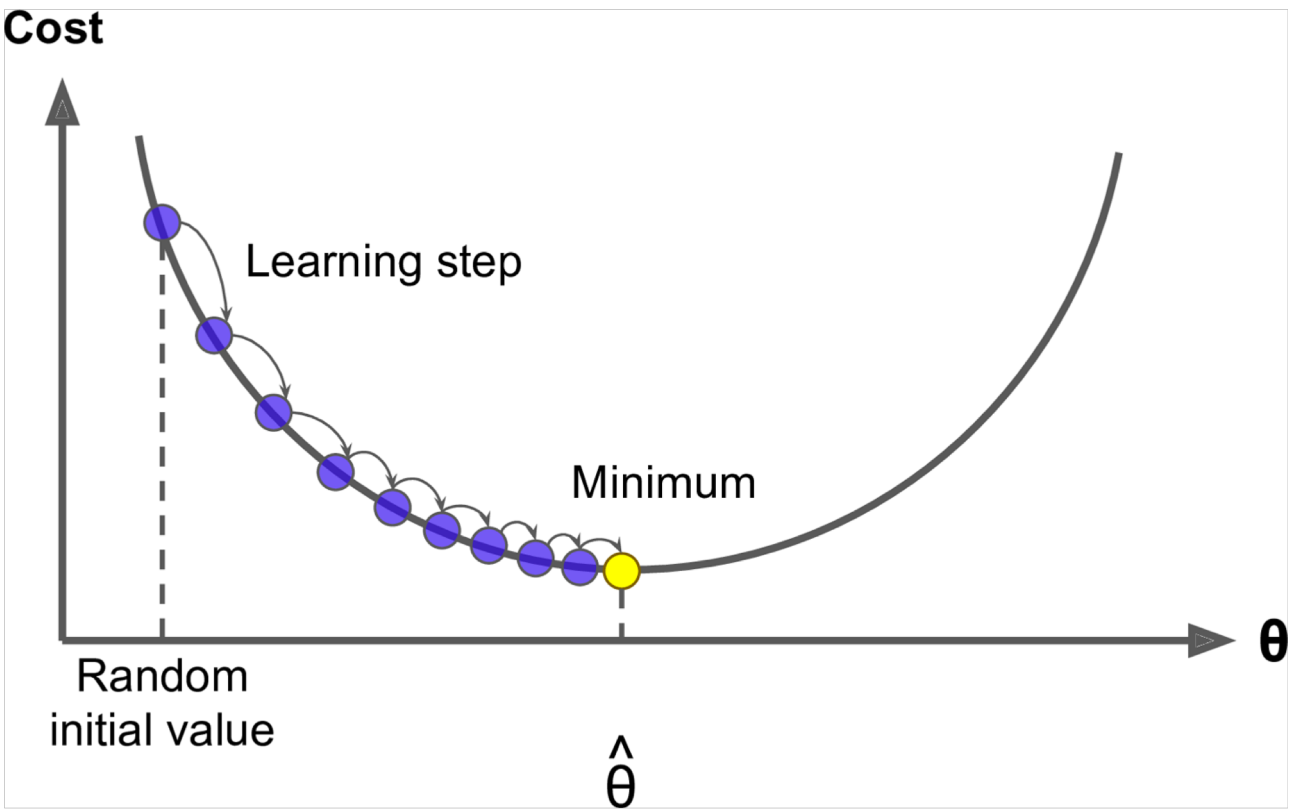
즉, 가까이에 그룹핑되는 영역에 관심을 가지고, 또한, 같은 방향의 라인에 있는 단어들에게도 관심을 가져야 한다 (멀리 있더라도).

이를 analogy로 확인할 수 있는데...
analogy('man', 'king', 'woman')
> 'queen'
analogy('tall', 'tallest', 'long')
> 'longest'

semantic과 함께 syntactic fact도 확인할 수 있다.

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.



Note: Our objectives may not be convex like this

Gradient Descent

- Update equation (in matrix notation):

$$\theta^{new} = \theta^{old} - \underbrace{\alpha}_{\alpha \text{ = step size or learning rate}} \nabla_\theta J(\theta)$$

α = *step size* or *learning rate*

- 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:

mini - batch를 하는 것과 같다.
1. 한 개만 고려할 때보다 노이지가 덜 하고
2. GPU의 능력(병렬)을 활용할 수 있다.

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

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!

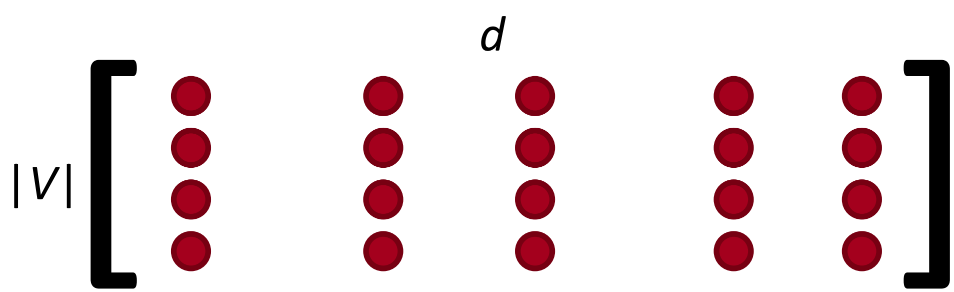
SGD, mini - batch를 하므로
gradient가 0인 케이스가 매우 많다.

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

Stochastic gradients with word vectors!

- We 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

dot product는 symmetric 하므로, 두 개의 벡터가 있어도 크게 상관없다. 두 벡터는 거의 비슷해질 것이다.

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.

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

- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**

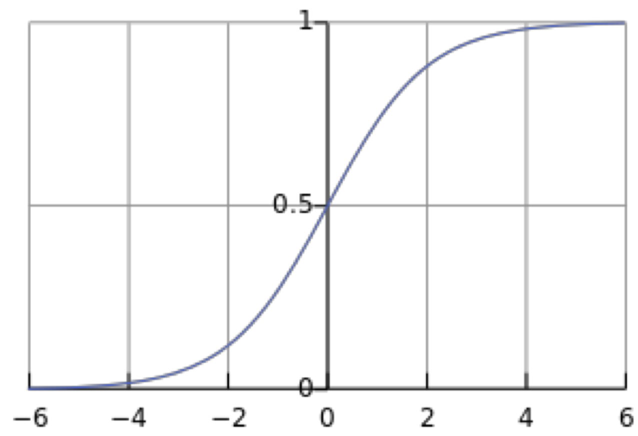
- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)

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): $J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J_t(\theta)$

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

- 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
(K=10 or 15)

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

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

3/4 power의 의미
decreasing how often you sample very common words and increasing how often you sample rare words

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

sample word의 start point는 unigram distribution을 따르는 것이다. (count independent words)

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?

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"

Document 전체를 context라고 생각하면, 데이터 locality를 잃어버린 것과 마찬가지로 된다. 이미지 데이터와 비교해보면 이미지에서 locality는 critical한 부분을 가진다. Text 데이터 역시 locality는 POS의 의미를 지닌다.

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

Sparse matrix

Problems with simple co-occurrence vectors

Increase in size with vocabulary

New word가 생기면,
the whole matrix를 업데이트해야 한다.

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

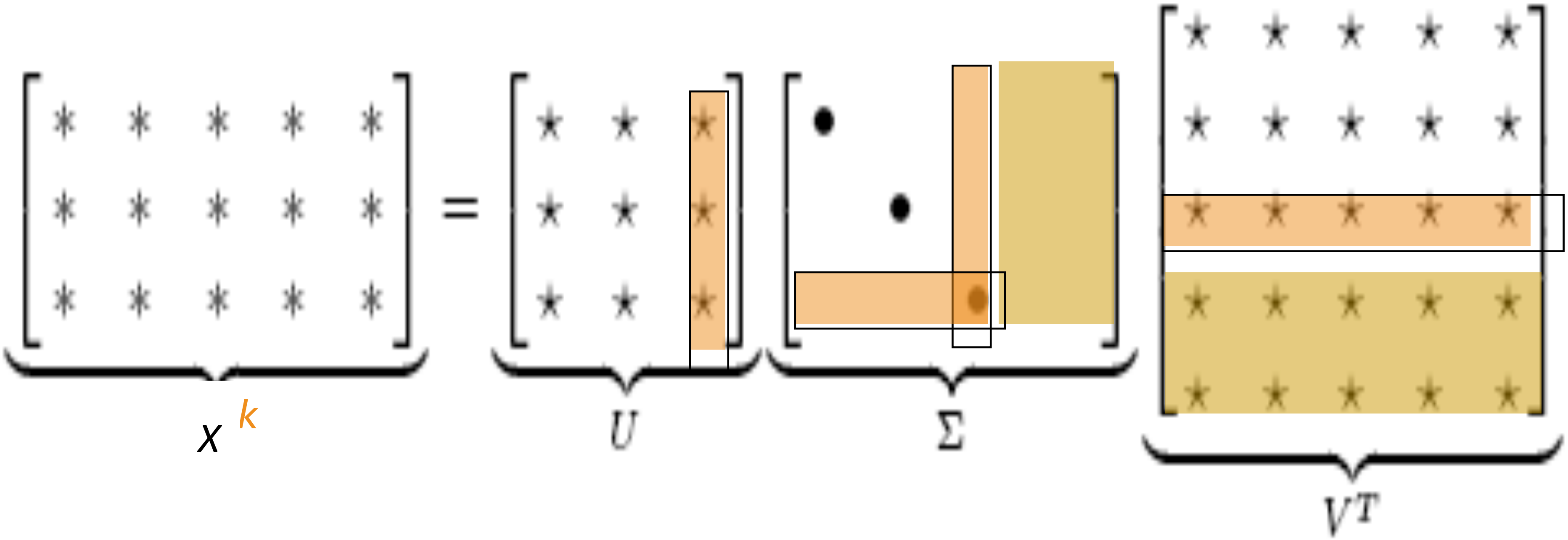
- Usually 25–1000 dimensions, similar to word2vec

- How to reduce the dimensionality?

Method 1: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X Factorizes X into $U\Sigma V^{\mathrm{T}}$, 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.

We can represent the original co - occurrence matrix and find to best - length k approximation to that, in terms of Least Squares by using SVD.
Orthonomal columns in left singular vectors and orthonomal rows in right singular vectors. And along diagonal, singular values, which are sorted.
Each of these columns basically describes principle component.
Actually, diagonal matrix is automatically ordered and the first singular value is the most important axis variation in your data in the first column of U_1
Instead of representing X, all of the orthonomal columns and rows and we can just take top - k. That allow us to reduce noise.

Simple SVD word vectors in Python

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

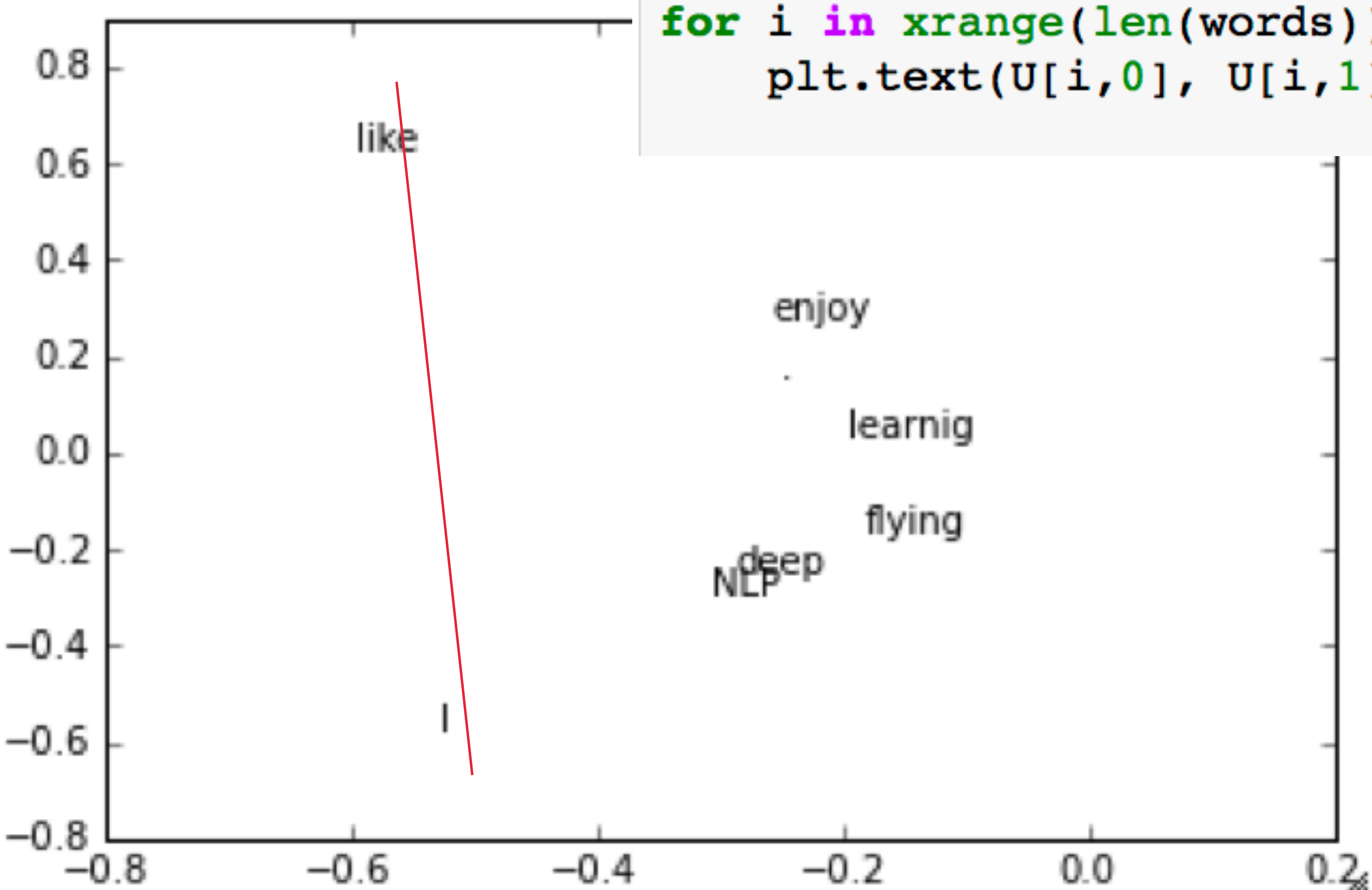
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],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



Like와 I는 공통된 세로 축을 가지므로 비슷한 linear relationship을 가지는데 이는 Like와 I가 frequency가 높아서 일수도 있다.

예전에는
이러한 작업 - LSA approximations
을 통해 Information Retrieval을 많이 했다.

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

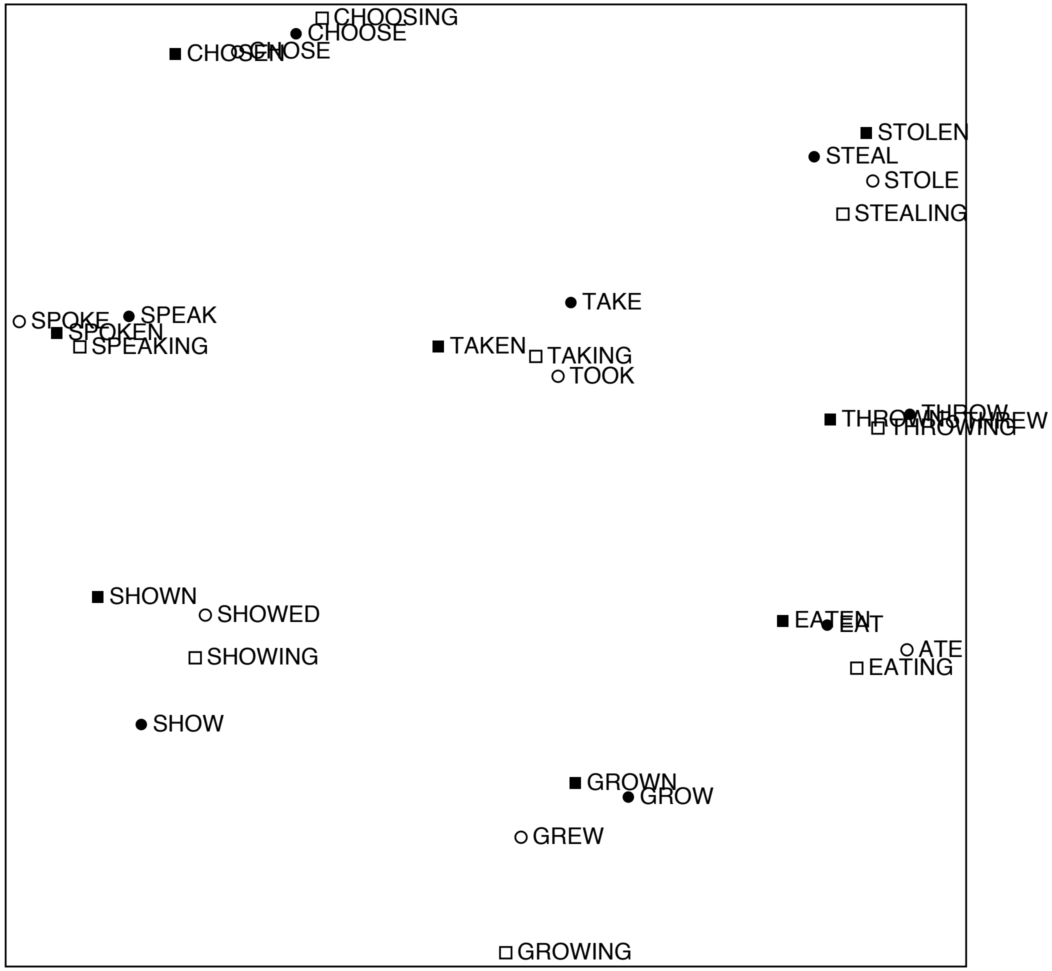
Ramped windows:
In co - occurrence matrix, counts with weight of them. 즉, 기존의 co - occurrence matrix와 같이 count만 고려하는게 아니고, center word와의 거리에 따라 weight도 부여해준다. 만약 window - size가 1이면 weight는 언제나 동일할 것이다.

- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

- Pearson correlation will be negative for most words. That is, the most words don ' t appear in the context of other words. (팬더는 뉴클리어밤이랑 왠만하면 같이 나오지 않는다.)
- Pearson correlation의 결과는 [- 1, 1]을 가지는 대칭행렬이 완성된다.

이러한 transformed counts 는 유용한 word vector를 제공해준다.
이와 같이 raw counts에서 different form들로 변형되는 점이 Word2vec에서도 사용되고 있다.

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

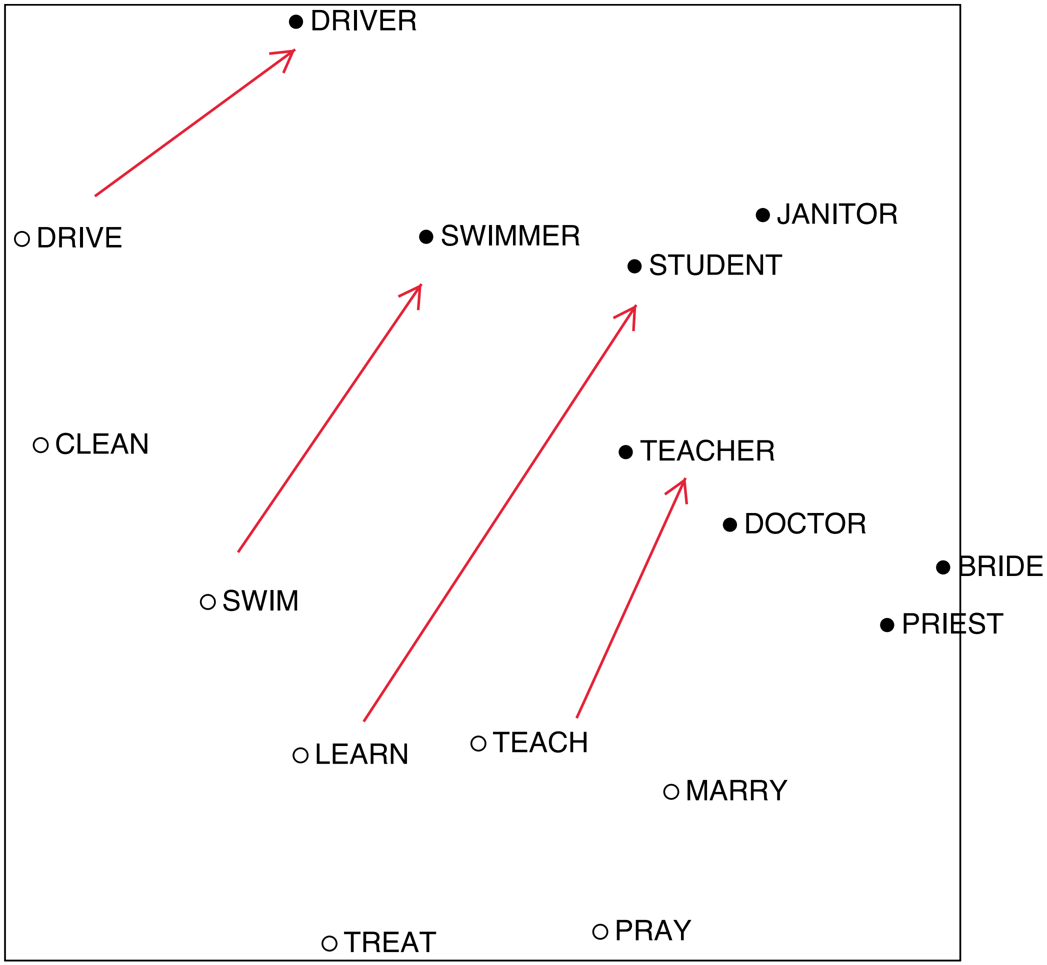
Interesting semantic patterns emerge in the vectors

다음 carefully - constructed space
에서 linear components인
semantic vectors가 있다.

semantic direction이 있다.

이렇듯 conventional method도 좋은
word vector를 가지고 있었다.

Co - occurrence로부터 시작된 word
vector는 Glove의 출발점이 된다.



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

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)

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

SVD - based 방법론들은 모든 단어들의 횟수가 미리 정해져 있어야 하는 반면, Word2Vec 방법론은 횟수를 전부 알지는 못해도 sampling을 통해서 횟수를 확률로 표현한다. 그리고 점진적으로 모델을 train해서 정확도를 높여가는 방식이다. (sampling이 거의 population을 흉내 낼 때는 SVD와 같이 모든 횟수를 추정할 수 있을 것이다. 그리고 좋은 sample 몇 개만 가져도 정확한 확률을 가질 수 있을 것이다.)

앞의 두 아이디어를 결합.
neural net method의 좋은 점과
count matrix의 좋은 점.

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 = 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

large와 small이라는 정보만으로는
부족하다.

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

ratio of co - occurrence probabilities = dimension of meaning

ratio of co - occurrence probabilities가 count보다 더 의미적으로 보이므로(값의 크기가 크다), 이를 vector space에 어떻게 encoding할 지 고민해보자.

water와 fashion은 거의 1에 수렴하므로 dimension of meaning이 아니다.

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

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]

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

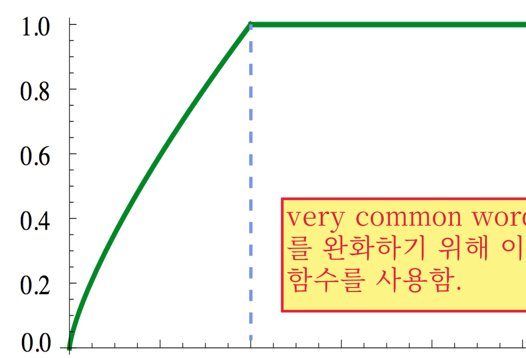
녹색 두 term이 같아지면 좋다.

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

b는 두 단어에 대한 bias term이다.

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

$f \sim$



very common word pairs를 완화하기 위해 이러한 f 함수를 사용함.

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

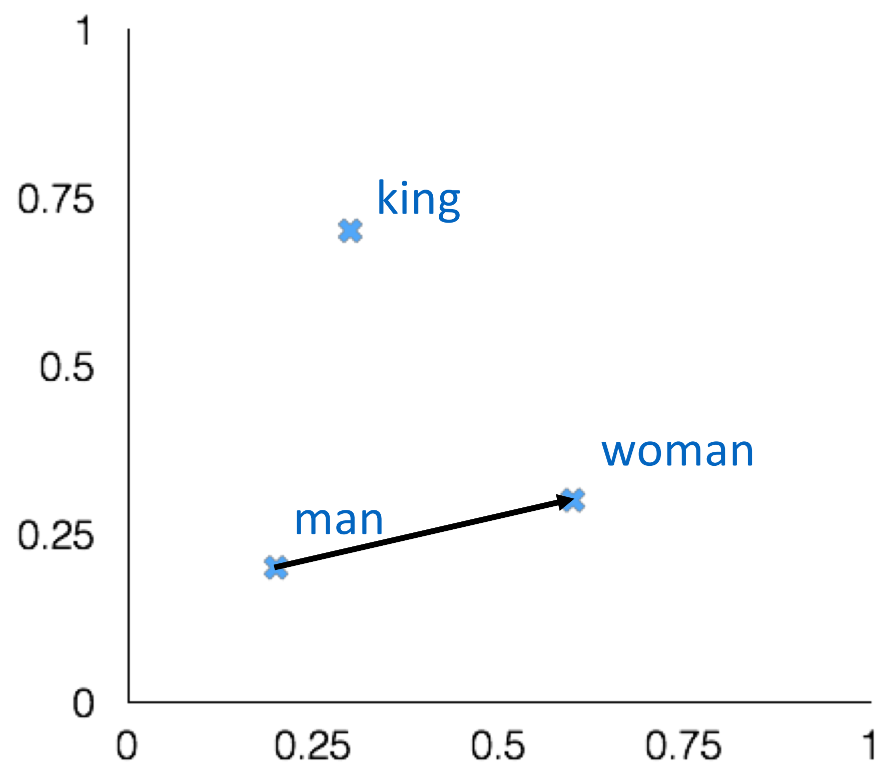
- Word Vector Analogies

a:b :: c:?

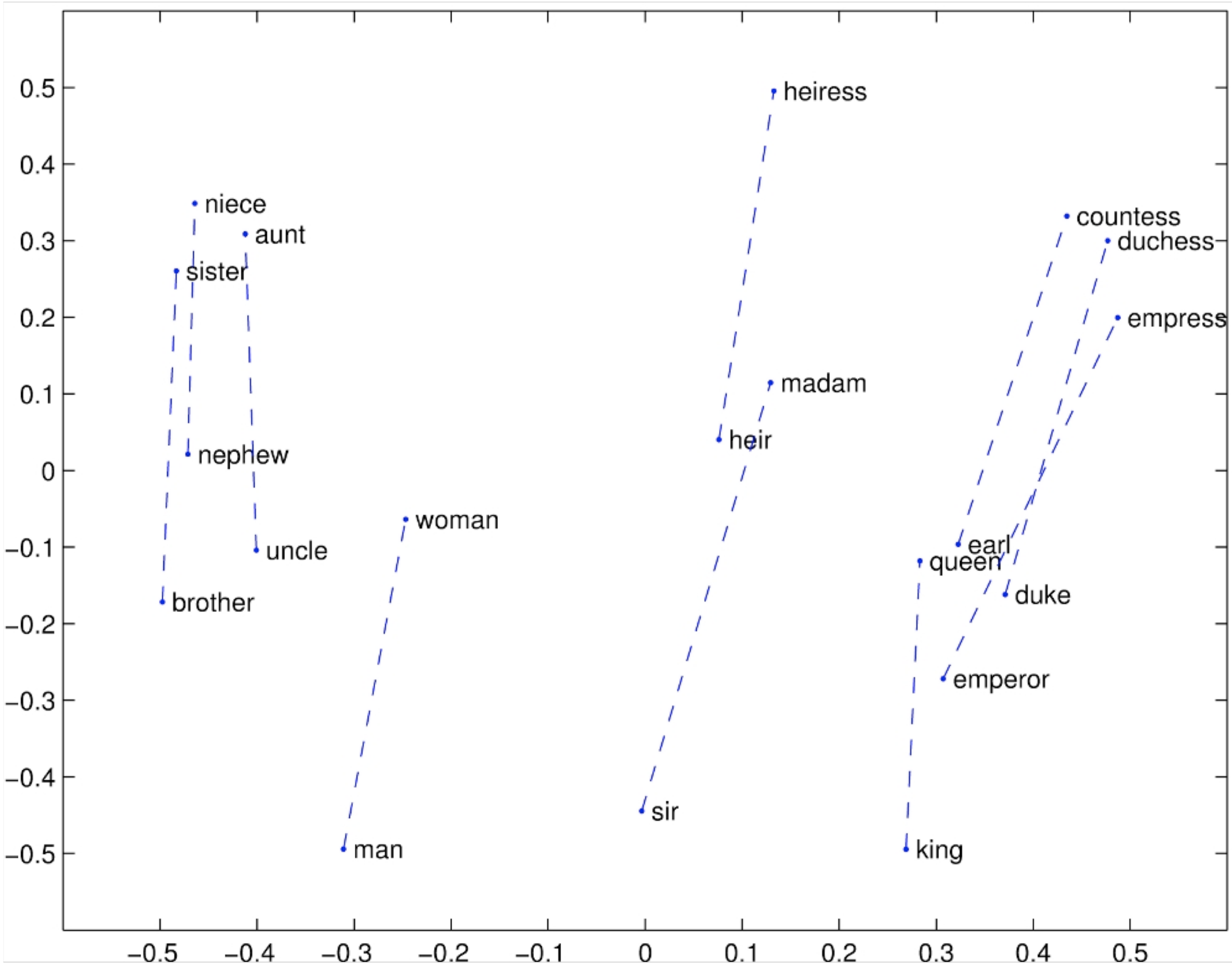
man:woman :: king:?

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

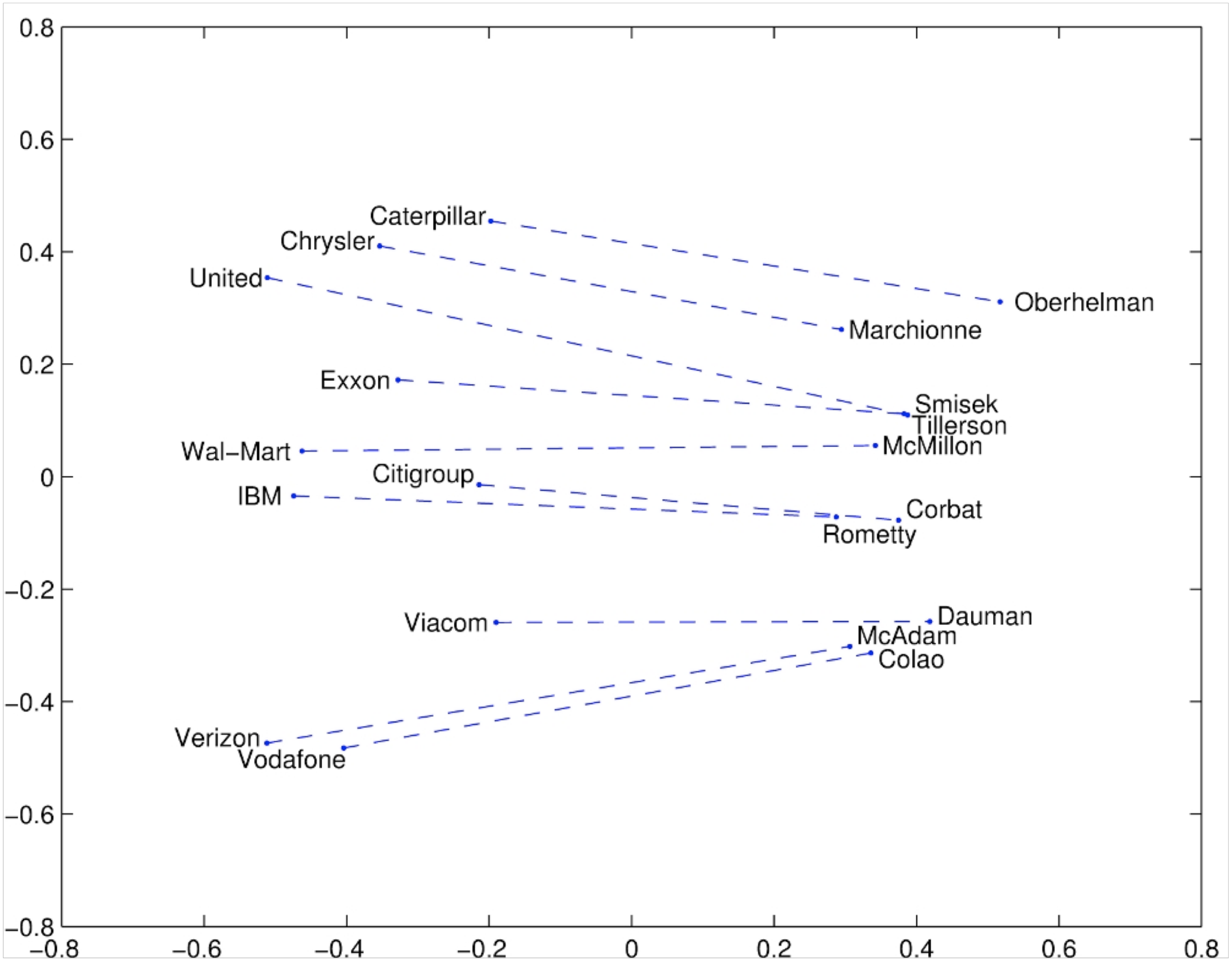
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



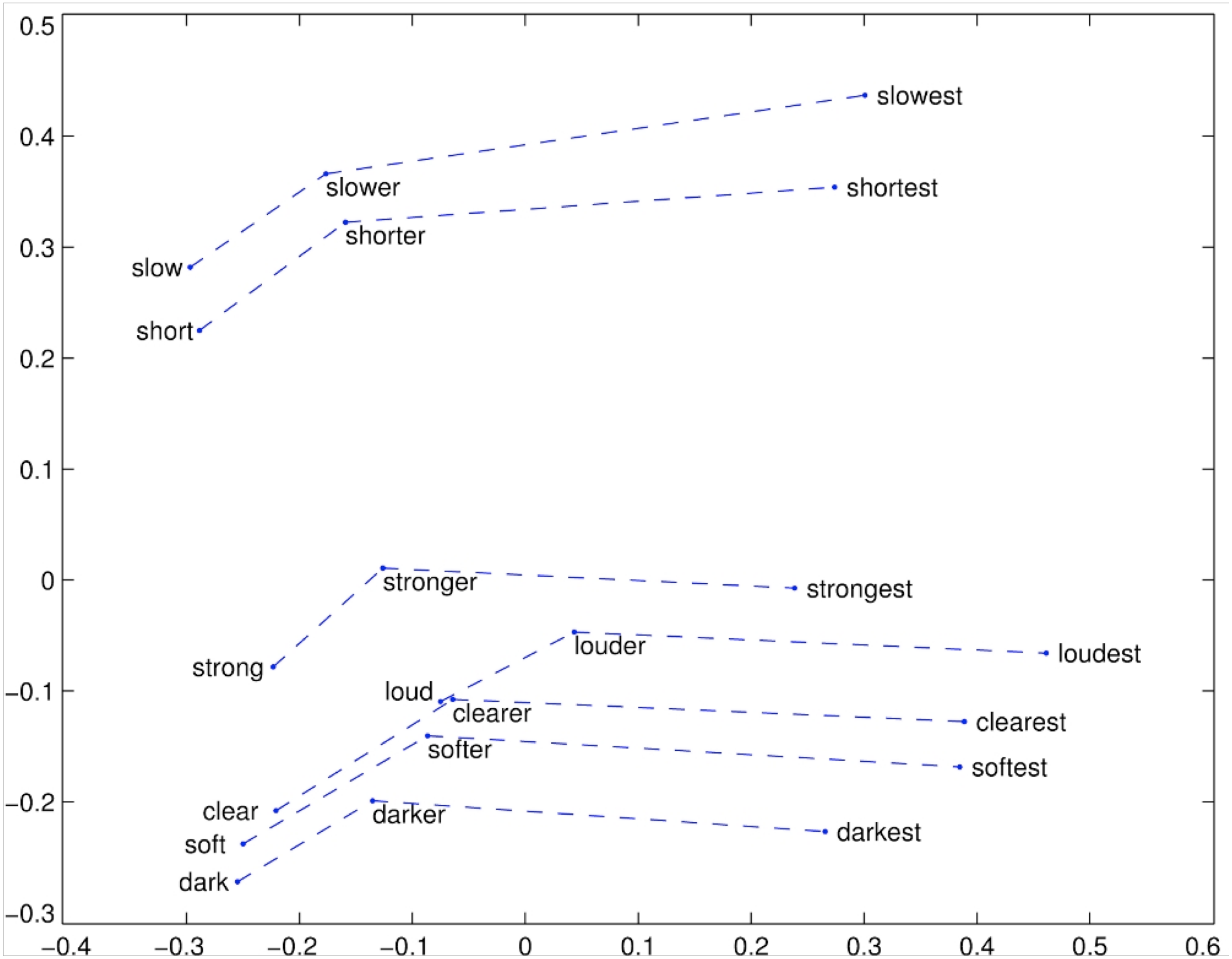
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



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

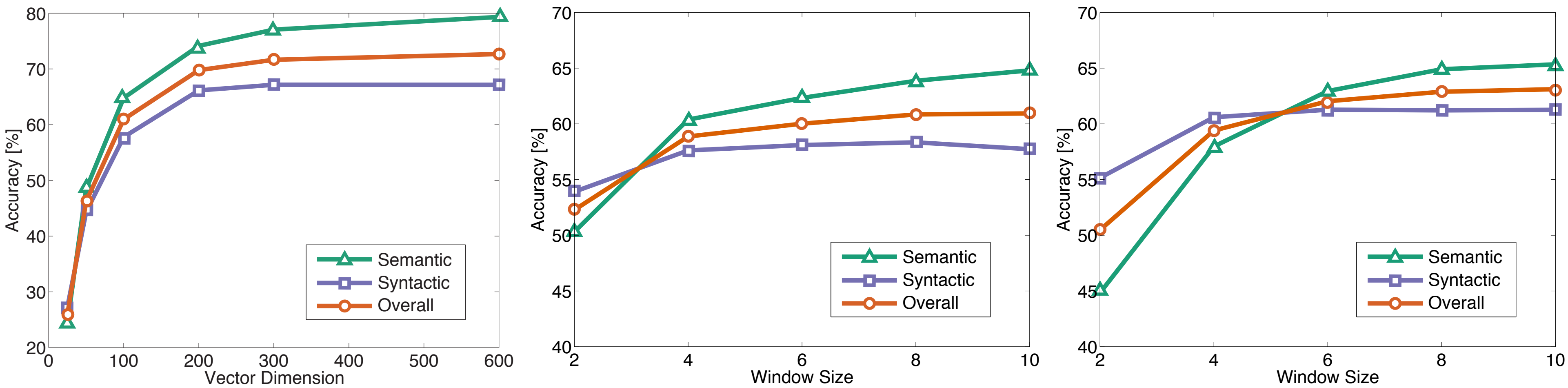
: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

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	<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



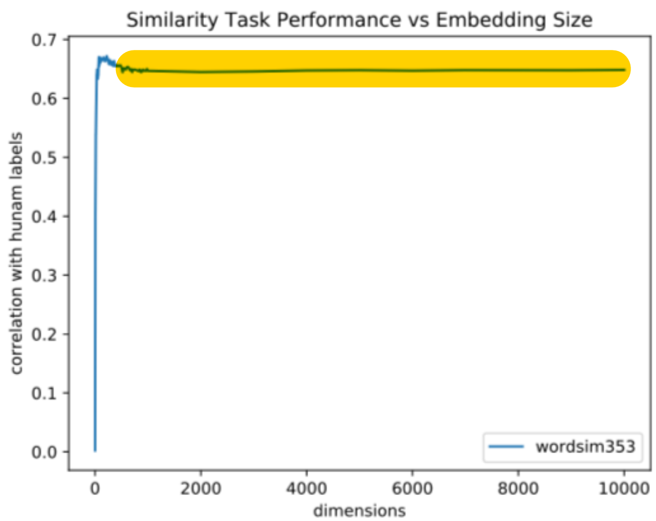
(a) Symmetric context (b) Symmetric context (c) Asymmetric context

Dimensionality Window size 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

dim이 300에서부터 증가해서 10000까지 성능이 flat한 것을 알 수 있다. 일반적인 생각으로는 dim이 매우 크면 학습해야 되는 데이터가 더 많아지지 않는 이상 성능은 더 떨어질 것으로 생각되는데..

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

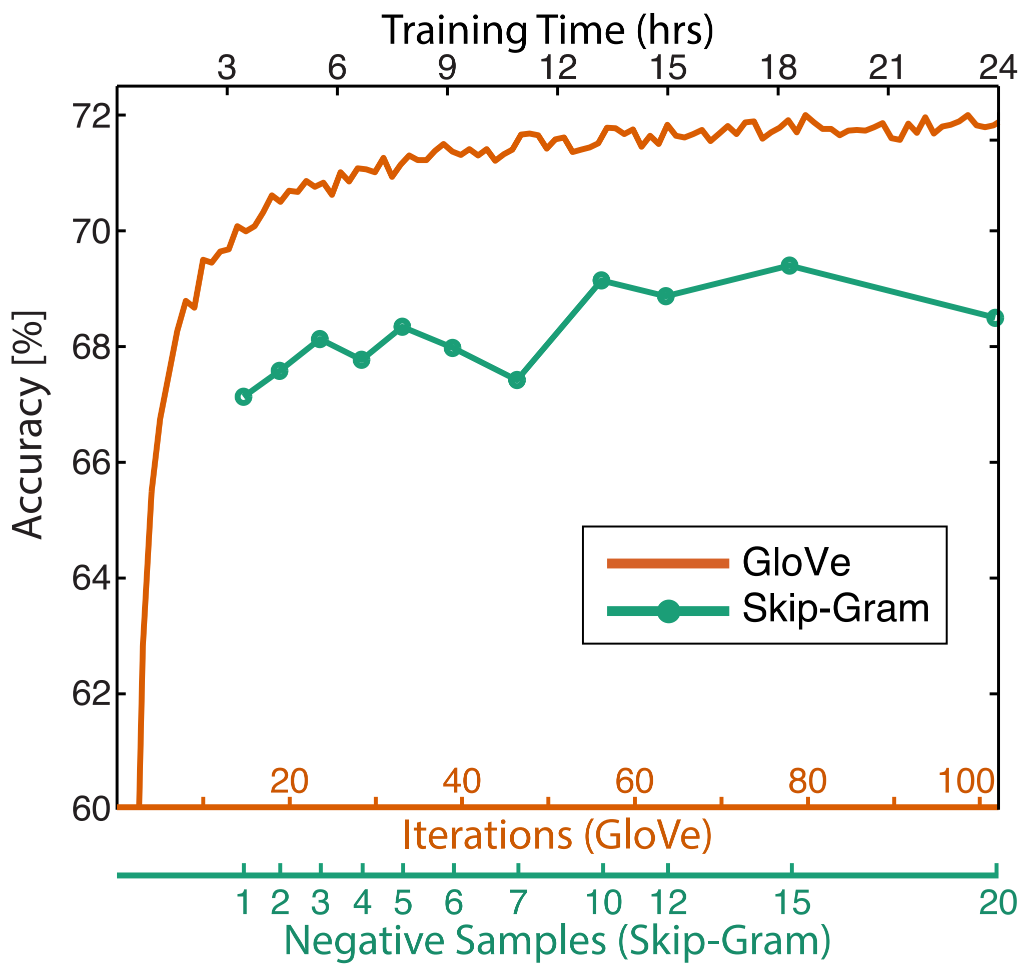
Using matrix perturbation theory, reveal a fundamental bias-variance trade-off in dimensionality selection for word embeddings

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

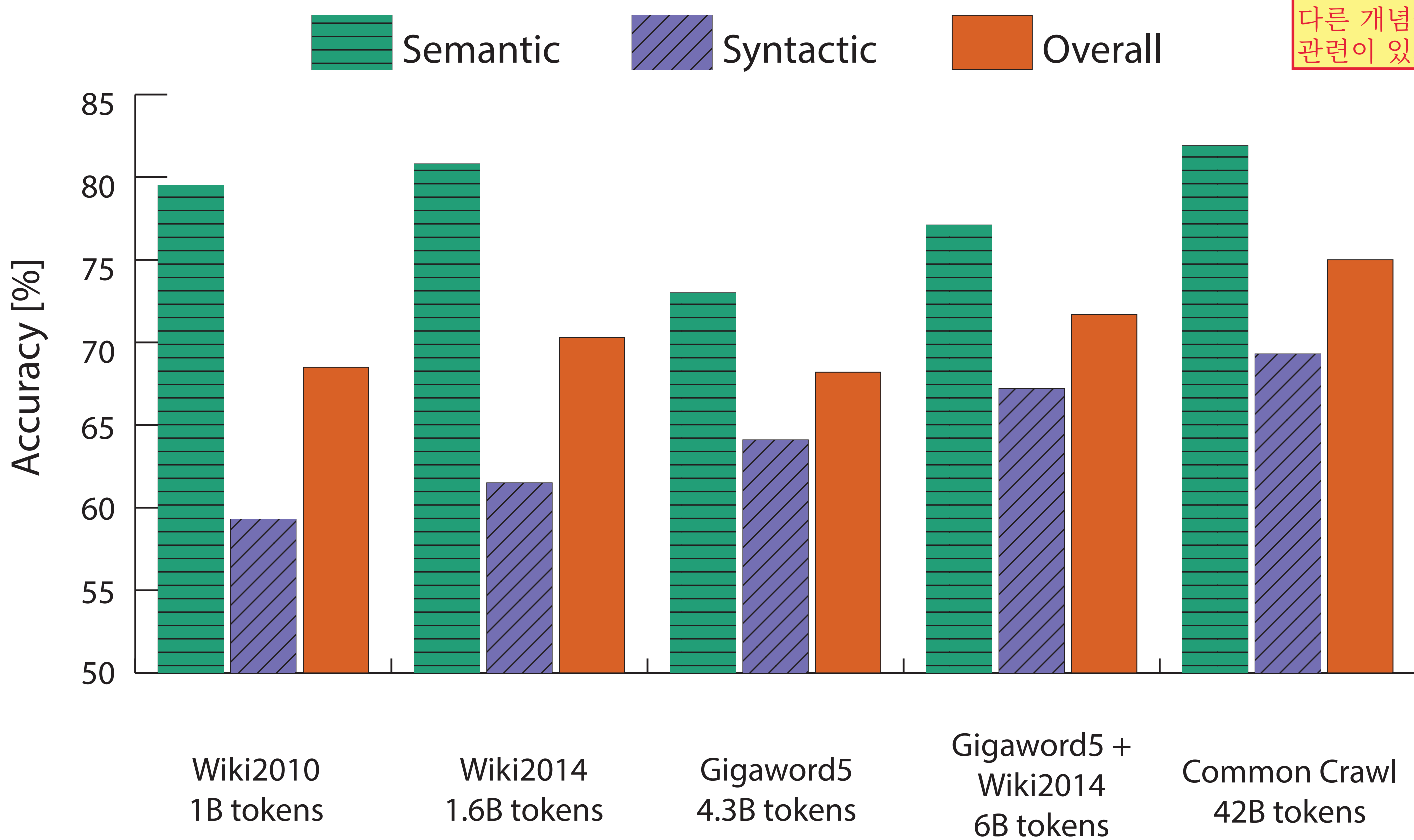
- More training time helps



Analogy evaluation and hyperparameters

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

위키피디아는 주로 개념을 설명하는 글이기 때문에.. 이는 다른 개념들과 서로 밀접하게 관련이 있고.. 그러다보니..



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
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	65.6	68.2	57.0	32.5
SG†	6B	62.8	65.2	69.7	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	<u>**75.9**</u>	<u>**83.6**</u>	<u>**82.9**</u>	<u>**59.6**</u>	<u>**47.8**</u>
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

- Example: pike

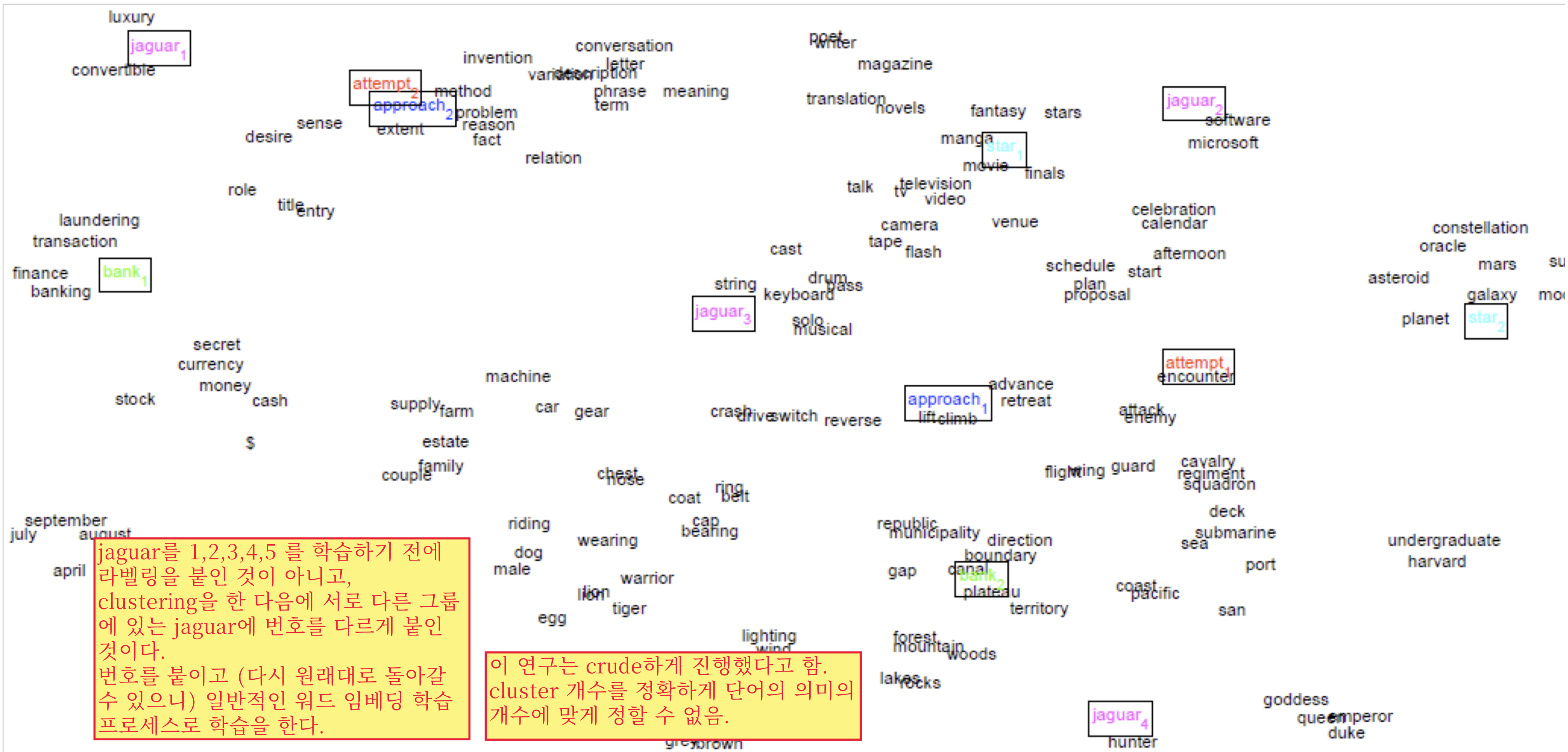
- Does one vector capture all these meanings or do we have a mess?

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



jaguar를 1,2,3,4,5 를 학습하기 전에 라벨링을 붙인 것이 아니고, clustering을 한 다음에 서로 다른 그룹에 있는 jaguar에 번호를 다르게 붙인 것이다.
번호를 붙이고 (다시 원래대로 돌아갈 수 있으니) 일반적인 워드 임베딩 학습 프로세스로 학습을 한다.

이 연구는 crude하게 진행했다고 함. cluster 개수를 정확하게 단어의 의미의 개수에 맞게 정할 수 없음.

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
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)

tie는 아래와 같이
5개의 meaning component를 가지는
것을 superposition으로 추출했다.

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**

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

Course plan: coming weeks

Week 2: Neural Net Fundamentals

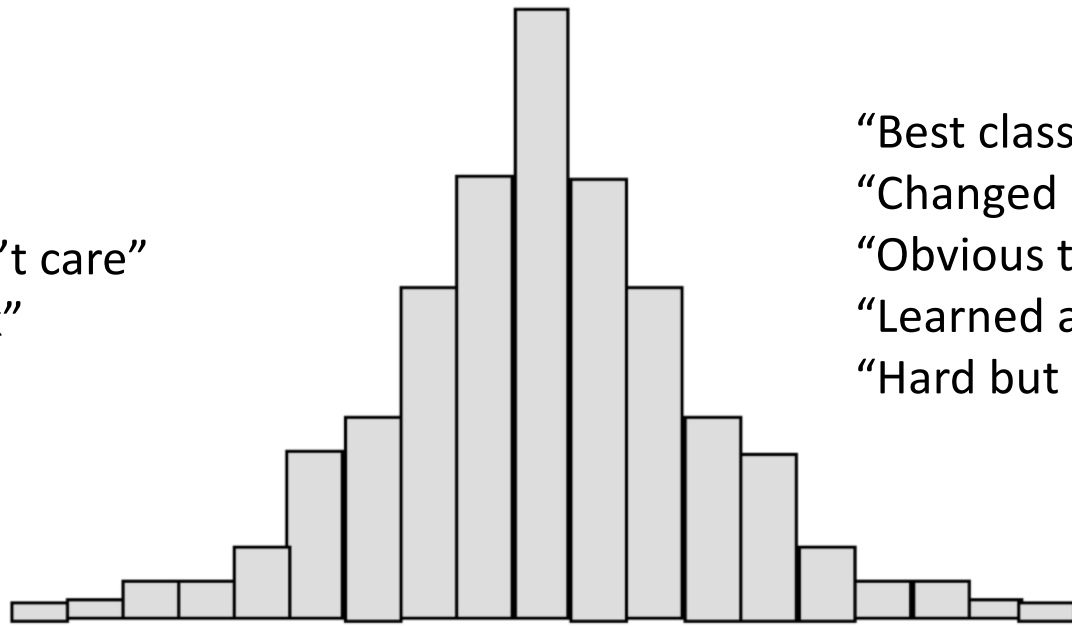
- We concentrate on understanding (deep, multi-layer) neural networks and how they can be trained (learned from data) using backpropagation (the judicious application of calculus)
- We'll look at an NLP classifier that adds context by taking in windows around a word and classifies the center word (not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