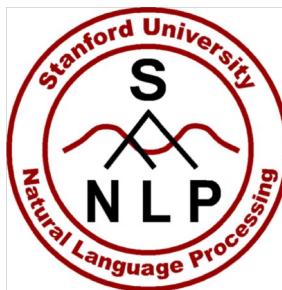


Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning
Lecture 2: Word Vectors and Word Senses



Lecture Plan

Lecture 2: Word Vectors and Word Senses

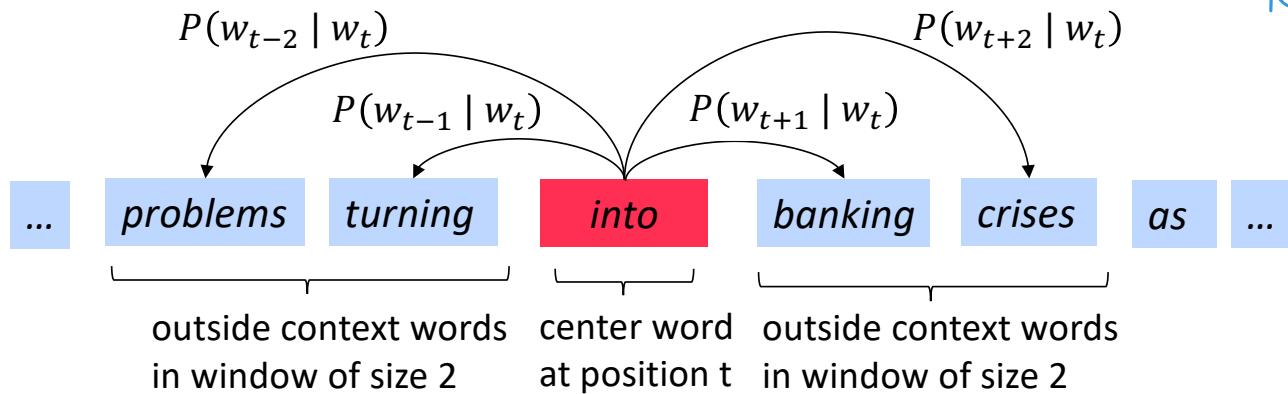
1. Finish looking at word vectors and word2vec (12 mins)
2. Optimization basics (8 mins)
3. Can we capture this essence more effectively by counting? (15m)
4. The GloVe model of word vectors (10 min)
5. Evaluating word vectors (15 mins)
6. Word senses (5 mins)
7. The course (5 mins)

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

1. Review: Main idea of word2vec

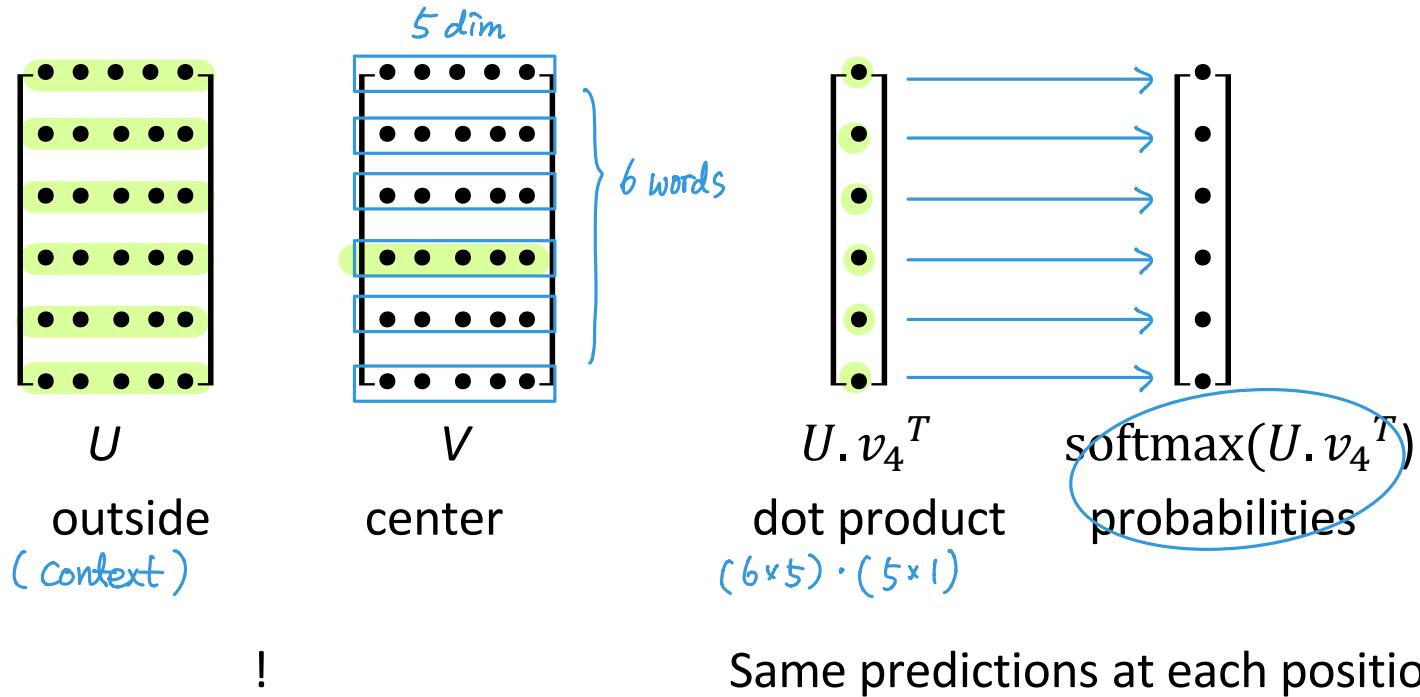
- Iterate through each word of the whole corpus
- Predict surrounding words using word vectors

CBOW
skipgram



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

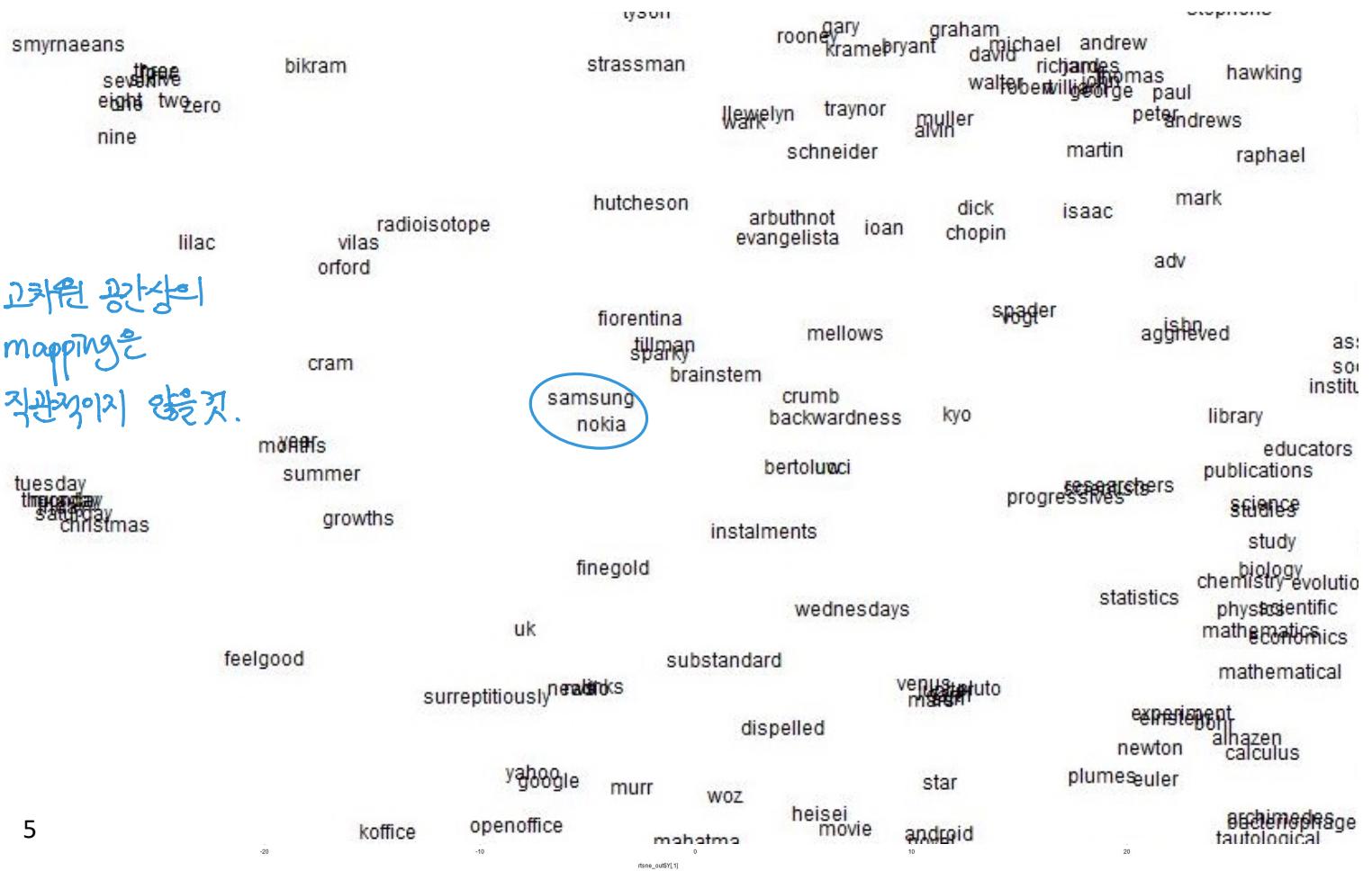
Word2vec parameters and computations



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

- 우리가 원하는 것
→ context에 나온 법한 단어들이 높은 확률을 부여하는 것.
⇒ ISSUE : 'the', 'and' 등의 문법적인 단어는 높은 확률을 가지 것. (추후 다른 예제)

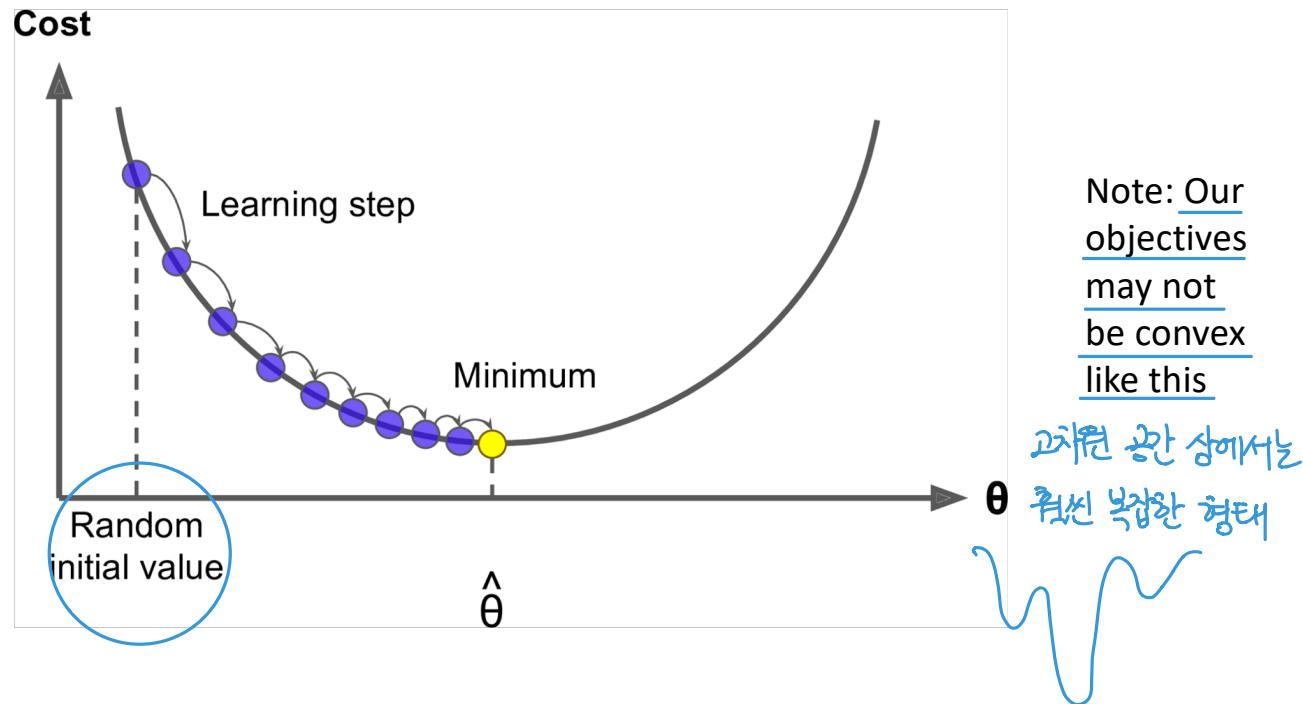
Word2vec maximizes objective function by putting similar words nearby in space



* 고차원 공간상의
mapping은
직관적이지 않을 것.

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.



Gradient Descent

→ 모든 data의 gradient를 계산 후 update
(data가 커지면 속도 매우 느림)

- Update equation (in matrix notation):

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

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

→ data 중 random sample
하나는 봄이 update (빠른 속도)

- 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)** & mini batch SGD
 - Repeatedly sample windows, and update after each one
- Algorithm:

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

Stochastic gradients with word vectors!

각 window 단위로!

- 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!

window size = m 일 경우

center word
왼쪽 m 개
오른쪽 m 개 } $2m + 1$

예) window size = 10

($2d_{\text{dim}} \times 2$) 크기의 word vector를

update하기 위해 ($2d_{\text{dim}} \times$ 수백만) 크기의

matrix가 필요



sparsity로 인한

계산 비효율성

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

Stochastic gradients with word vectors!

- We might only update the word vectors that actually appear!
 - ① 실제로 등장한 row에 대해서만 gradient 계산
- 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

② 명시적인 Index를 부여하여 빠른 mapping 진행

$$|V| \begin{bmatrix} \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}^d$$

- 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 : 자정된 window에 등장하지 않은 단어 (negative sample)

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.

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

- 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)
• **특징:** true pair에 대한 high probability .

The skip-gram model with negative sampling (HW2)

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

maximize likelihood of w_{t+j} given w_t

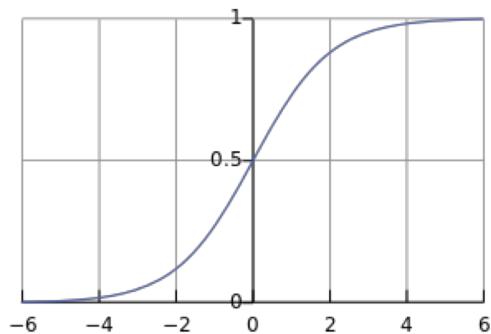
$$J_t(\theta) = \underbrace{\log \sigma(u_o^T v_c)}_{\text{Binary Softmax}} + \underbrace{\sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]}_{\text{Negative Sampling}}$$

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

$\rightarrow (\text{Word}) \cdot (\text{context}) \Rightarrow \text{high prob.}$



유사한 공간에 위치 \rightarrow 벡터값 부호 같을 확률 높음 \rightarrow 나적 시 큰 값

The skip-gram model with negative sampling (HW2)

- Notation more similar to class and HW2:

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

- We take k negative samples (using word probabilities)
Maximize probability that real outside word appears,
minimize prob. that random words appear around center word
- $P(w) = U(w)^{3/4} / Z$,
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
(empirical)

왜 word2vec에서는 co-occurrence 정보를 사용하지 않나?

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

뒤에서 답 확인

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”

입력 데이터에 특잇값 분해를 수행해

데이터의 차원 수를 줄여 계산 효율성을 키우는 한편,
행간에 숨어 있는 (latent) 의미를 이끌어내기 위한 방법론

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
for simplicity
- Symmetric (irrelevant whether left or right context)
for simplicity
- 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.

Symmetric
&
Sparse

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

But why not capture co-occurrence counts directly? ← 이 질문에 대한 답변

Problems with simple co-occurrence vectors

Increase in size with vocabulary

① Vocab 크기가 너무 크다.

Very high dimensional: requires a lot of storage

② 고차원 공간은 메모리를 많이 차지

Subsequent classification models have sparsity issues

③ sparsity 문제

→ Models are less robust

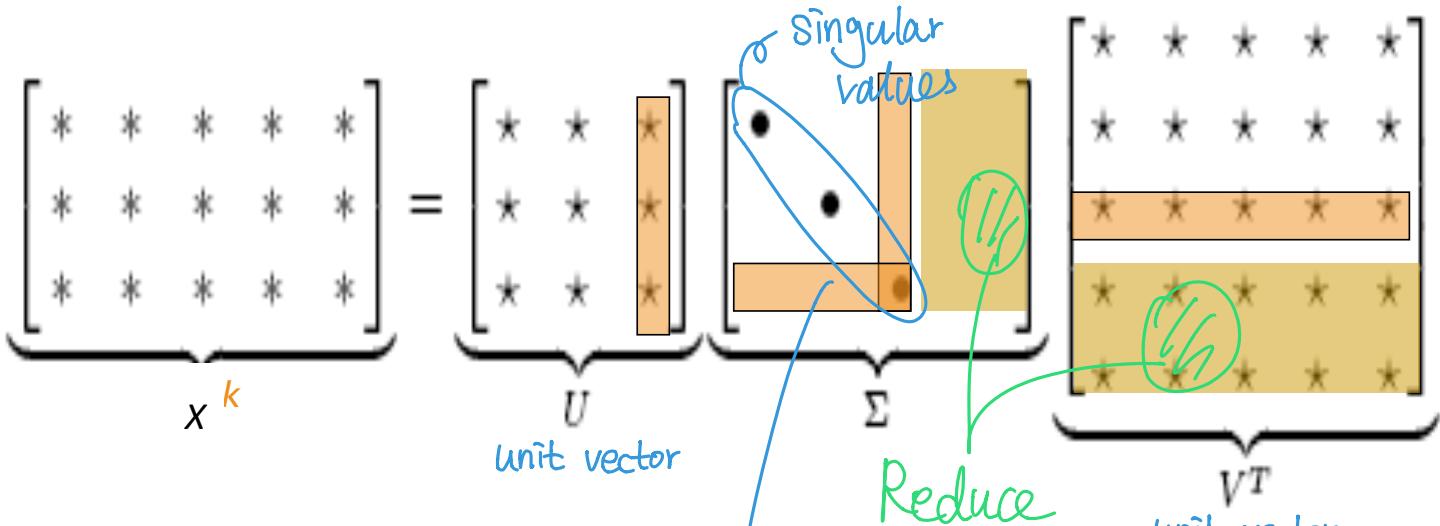
Solution: Low dimensional vectors

- 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.

가장 큰 2개만 사용 \Rightarrow 고차원으로 축소

* Eigen Vector & EigenValue

$$\begin{array}{l} \textcircled{1} \quad \vec{v} \neq 0 \\ \textcircled{2} \quad L(v) = \lambda v \end{array} \quad \left. \begin{array}{l} \lambda: \text{eigenvalue (scalar)} \\ v: \text{eigenvector} \end{array} \right.$$

* Singular Value (특잇값)

$m \times n$ 행렬 A에 대해

$A A^T$ 의 고윳값이 $\lambda_1, \lambda_2, \dots, \lambda_n$ 이라면

$\sigma_1 = \sqrt{\lambda_1}, \sigma_2 = \sqrt{\lambda_2}, \dots$ 가 A의 특잇값이다.

* Singular Value Decomposition (특잇값분해)

영행렬이 아닌 임의의 $m \times n$ 벡터 A를 다음과 같이 분해

$$A = U \Sigma V^T$$

U & V: Orthonormal

Σ : 주대각선분이 A의 특잇값 (나머지 성분은 0)

크기는 $m \times n$

* Reduced SVD

대각행렬에서 원소가 0인 부분과
대응하는 특이벡터를 삭제

```

import numpy as np

X = np.array([[0, 2, 2, 0, 0, 0, 0, 0, 0, 0], # I
              [2, 0, 0, 1, 0, 0, 0, 0, 0, 0], # ate
              [2, 0, 0, 0, 0, 0, 1, 0, 1, 0], # like
              [0, 1, 0, 0, 1, 1, 0, 0, 0, 0], # a
              [0, 0, 0, 1, 0, 0, 0, 0, 0, 1], # banana
              [0, 0, 0, 1, 0, 0, 0, 0, 0, 1], # cherry
              [0, 0, 1, 0, 0, 0, 0, 1, 0, 0], # deep
              [0, 0, 0, 0, 0, 0, 1, 0, 0, 1], # learning
              [0, 0, 1, 0, 0, 0, 0, 0, 0, 1], # NLP
              [0, 0, 0, 0, 1, 1, 0, 1, 1, 0]]) # .

```

10 차원으로
 표준화된 word vec
 (original)

```

# X_svd
U, S, V_t = np.linalg.svd(X)

```

```

k = 5
theta = sum(S[:k]) / sum(S)
print(f'theta : {theta}')
=====
theta : 0.8174988844487284

```

```

U = U[:, :k]
=====
[[-0.58951321,  0.63338858, -0.07381218, -0.31561873,  0.08921915], # I
 [-0.43880494, -0.4679372 , -0.0950592 , -0.15540716,  0.40200628], # ate
 [-0.49906814, -0.51894471,  0.18296896, -0.18902671, -0.32887242], # like
 [-0.21718772,  0.19140752,  0.37405398,  0.29204662,  0.48061906], # a
 [-0.12612756, -0.06426321, -0.39796596,  0.39641327,  0.19296524], # banana
 [-0.12612756, -0.06426321, -0.39796596,  0.39641327,  0.19296524], # cherry
 [-0.19421905,  0.18666537,  0.02954001,  0.04266901, -0.41679693], # deep
 [-0.11890892, -0.06274144, -0.25333285,  0.28215586, -0.35443303], # learning
 [-0.21471749,  0.16369162, -0.31774506,  0.17599977, -0.30080158], # NLP
 [-0.184132 ,  0.00884932,  0.57389513,  0.57316285, -0.16426787]] # .

```

5차원으로 축소된
 word vec

```

US = np.dot(U, np.diag(S[:k]))
X_hat = np.dot(US, V_t)
X_hat = np.round(X_hat)
=====

[[ 0., 2., 2., -0., 0., 0., -0., 0., -0., 0.],
 [ 2., 0., -0., 1., -0., -0., 0., -0., 0., 0.],
 [ 2., -0., 0., 0., 0., 0., 1., 0., 1., -0.],
 [-0., 1., 0., 0., 1., 1., -0., 0., 0., -0.],
 [ 0., -0., 0., 1., 0., 0., 0., -0., -0., 1.],
 [ 0., -0., 0., 1., 0., 0., 0., -0., -0., 1.],
 [-0., 0., 1., -0., 0., 0., 0., 0., 0., 0.],
 [ 0., -0., 0., 0., -0., 0., 0., 0., 0., 1.],
 [-0., 0., 1., 0., -0., -0., 0., 0., 0., 1.],
 [ 0., 0., -0., -0., 1., 0., 0., 1., 0., 0.]]

```

Original과 거의
 유사하게 복원

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]])
```

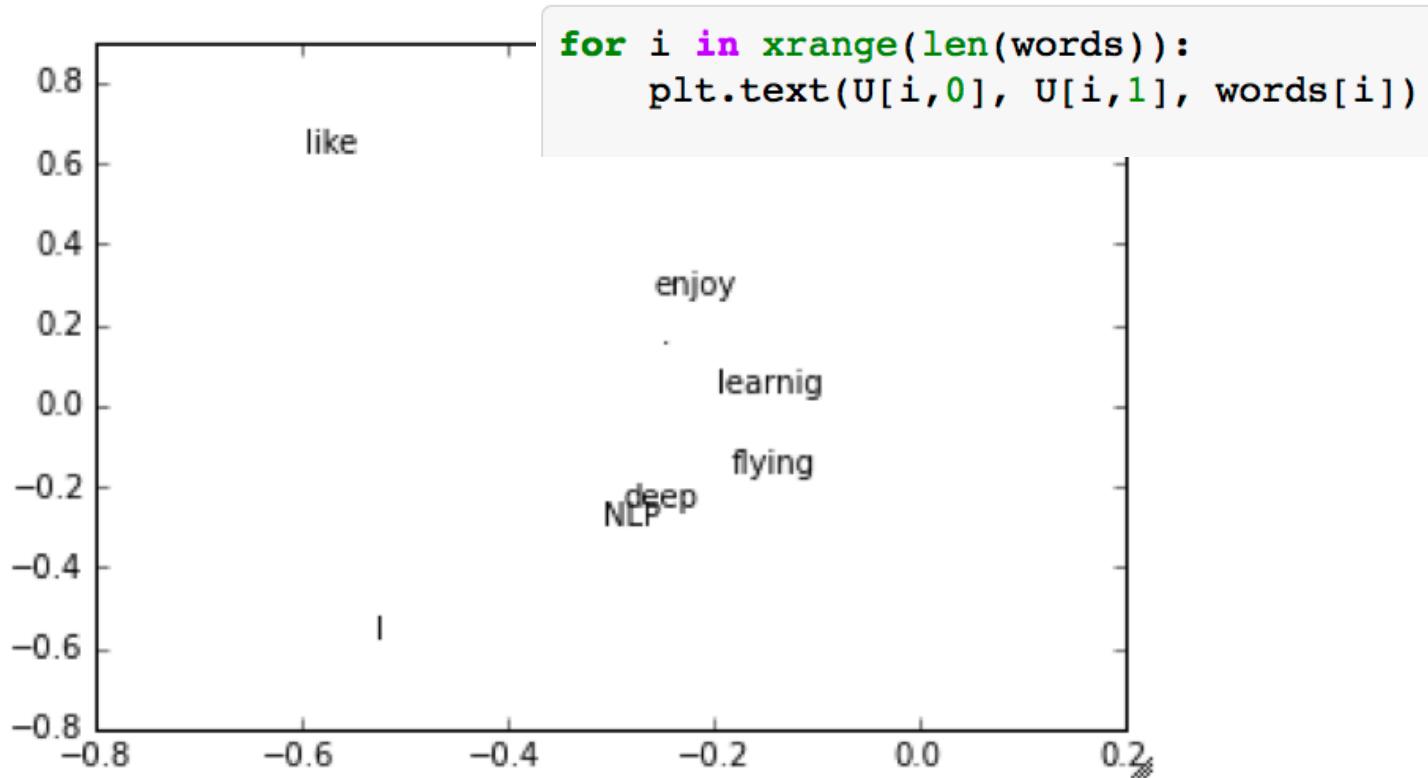


```
U, s, Vh = la.svd(X, full_matrices=False)
```

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

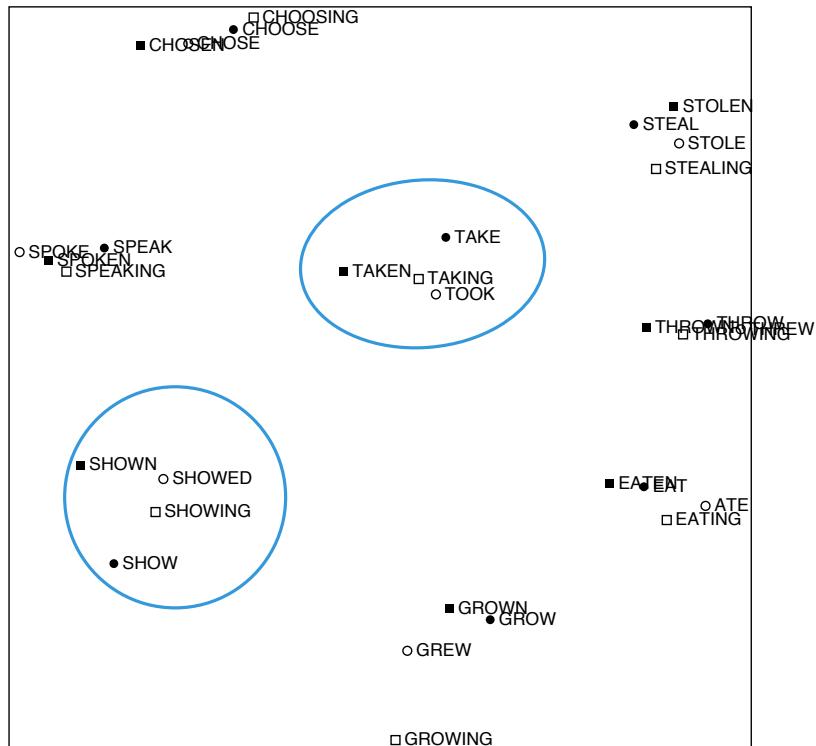


Hacks to X (several used in Rohde et al. 2005)

Scaling the counts in the cells can help **a lot** (예: log scale)

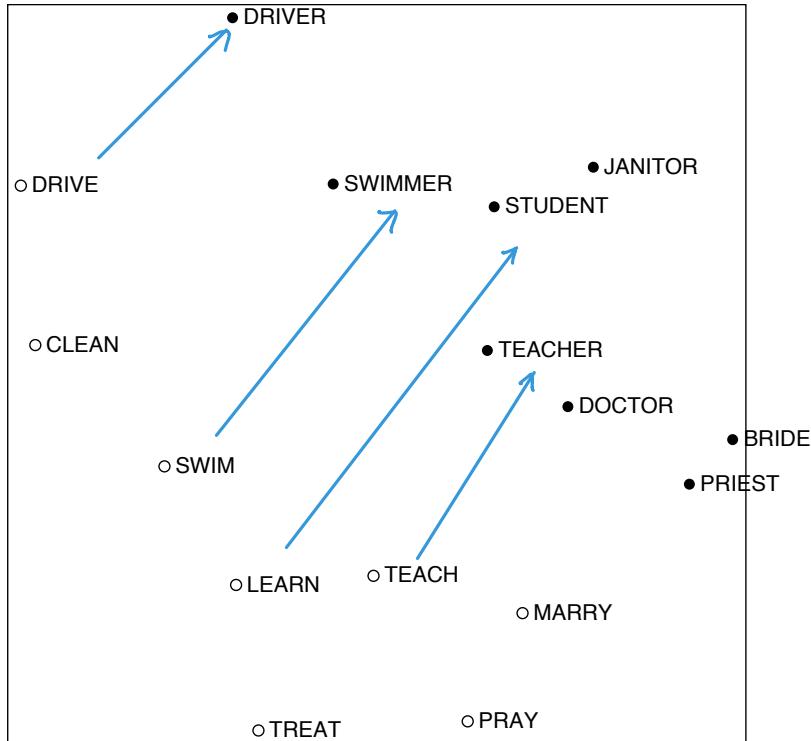
- Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
 - $\min(X, t)$, with $t \approx 100$ → 100 이상으로 넘어가지 않도록 (ceiling)
 - Ignore them all
- Ramped windows that count closer words more
center word와 가까운 수록 높은 가중치
- Use Pearson correlations instead of counts, 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



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

→ 'the', 'and' 등의 단어 자체가 많은 경우

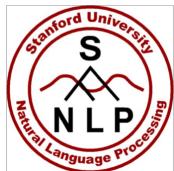
Neural Network Method

- 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

각 sample마다 update를 진행하기 때문에
co-occurrence 같은 통계정보 활용 불가

* Glove = Co-occurrence 정보 + word2vec 장점.



Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

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

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$ $P_{\text{ice } x}$	large	small	large	small
$P(x \text{steam})$ $P_{\text{steam } x}$	small	large	large	small
$\frac{P(x \text{ice})}{P(x \text{steam})}$	large	small	~ 1	~ 1

$$* P_{ij} = P(w_j | w_i) = \frac{x_{ij}}{x_i}$$

$\rightarrow j$ 번째 단어가 i 번째 단어와 같은 context에 등장한 확률



Encoding meaning in vector differences

[Pennington, Socher, and Manning, EMNLP 2014]

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

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{fashion}$
$P(x \text{ice})$	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



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

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

Input : vector diff

Output : ratio

GloVe

Pennington et al. (2014)

여러 번역

- Formulation

- ✓ Express the relationship among three words using a function F

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}} = \frac{P(k|i)}{P(k|j)}$$

세 단어의 관계를
나타내는 임의의 함수

세 단어의 관계는
분수로 표현 가능

- ✓ Relationship between w_i and w_j is expressed by subtraction

$$F(\underline{w_i - w_j}, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

- ✓ Inner product is used to link \tilde{w}_k with w_i and w_j

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}$$

두 단어 간 차이

context word

link를 만들기 위해 dot product
(하나의 scalar value 만들기)

GloVe

Pennington et al. (2014)

- Homomorphism → 학수 F가 input/output을 어떻게 mapping하는가?

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



✓ Want to preserve $\frac{P(k|ice)}{P(k|steam)}$ using $F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$

inverse

$$\frac{\frac{P(solid|ice)}{P(solid|steam)}}{=} F\left(\frac{(ice - steam)^T solid}{}\right)$$

↓ x ⊕

$$\frac{\frac{P(solid|steam)}{P(solid|ice)}}{=} F\left(\frac{(steam - ice)^T solid}{}\right)$$

GloVe

Pennington et al. (2014)

* 입력을 덧셈에 대한 역원으로 바꿔주면, 합수의 출력은 곱셈에 대한 역원으로 mapping

- Homomorphism

$$F((\text{ice} - \text{steam})^T \text{solid}) = \frac{P(\text{solid}|\text{ice})}{P(\text{solid}|\text{steam})} = \frac{1}{F((\text{steam} - \text{ice})^T \text{solid})}$$

Input

$$(\text{ice} - \text{steam})^T \text{solid} = -(\text{steam} - \text{ice})^T \text{solid}$$

inverse element of addition 덧셈에 대한 역원 (합수)

F의
output

$$F((\text{ice} - \text{steam})^T \text{solid}) = \frac{1}{F((\text{steam} - \text{ice})^T \text{solid})}$$

inverse element of multiplication 곱셈에 대한 역원 (역수)

✓ Homomorphism preserves an operation, which in turn preserves the inverse element

✓ Need a homomorphism from $(\mathbb{R}, +)$ to $(\mathbb{R}_{>0}, \times)$

위 조건을 만족 시키려면

이 가정 필요

입력 실수공간 상에서의 덧셈

→ 출력 > 0 공간 상에서의 곱셈

GloVe

Pennington et al. (2014)

- Homomorphism

- ✓ Function F: homomorphism that maps $(\mathbb{R}, +)$ to $(\mathbb{R}_{>0}, \times)$

$$w_i^T \tilde{w}_k = \underline{(w_i - w_j)^T \tilde{w}_k} + \underline{w_j^T \tilde{w}_k}$$

$$\begin{aligned} F(w_i^T \tilde{w}_k) &= F\left((w_i - w_j)^T \tilde{w}_k + w_j^T \tilde{w}_k\right) \text{ input에서의 합성} \\ &= F\left((w_i - w_j)^T \tilde{w}_k\right) \otimes F(w_j^T \tilde{w}_k) \text{ F output} \\ &\quad \text{에서의 곱셈} \end{aligned}$$

대수구조는 그대로 유지

$$\underline{F((w_i - w_j)^T \tilde{w}_k)} = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} = \frac{P_{ik}}{P_{jk}}$$

즉, $f(a+b) = f(a) \cdot f(b)$ 을 만족하는

- ✓ Finally, we can drive that $F(x) = \exp(x)$ 합수 f 가 필요.

⇒ 지수합수가 이를 만족하는
가장 간단한 형태.

GloVe

Pennington et al. (2014)

- Solution

✓ We know that $F(x) = \exp(x)$ and $F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} = \frac{P_{ik}}{P_{jk}}$

$$\exp(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{\exp(w_i^T \tilde{w}_k)}{\exp(w_j^T \tilde{w}_k)}$$

$P_{ik} = \frac{X_{ik}}{X_i}$



$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$



$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \underline{\log X_i}$$

$$w_i^T \tilde{w}_k = \log X_{ik} - b_i - \tilde{b}_k$$

상수항으로 처리

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log X_{ik}$$

(단어 i, k 임베딩의 내적) + (i와 k의 상수항) = $\log (i\text{와 }k\text{가 동시에 발생한 빈도})$

GloVe

Pennington et al. (2014)

- Objective Function

- ✓ A least squared objective function

$$J = \sum_{i,j=1}^V \left(\underbrace{w_i^T \tilde{w}_j + b_i + \tilde{b}_j}_{\text{param to learn}} - \underbrace{\log X_{ij}}_{\text{실제 관측값}} \right)^2$$
$$\Rightarrow J = \sum_{i,j=1}^V f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

where f has the following desiderata:

- ① $f(0) = 0$
- ② $f(x)$ should be non-decreasing so that rare co-occurrences are not overweighted.
- ③ $f(x)$ should be relatively small for large values of x , so that frequent co-occurrences are not overweighted. 빈도가 높은 단어의 가중치 작게

GloVe

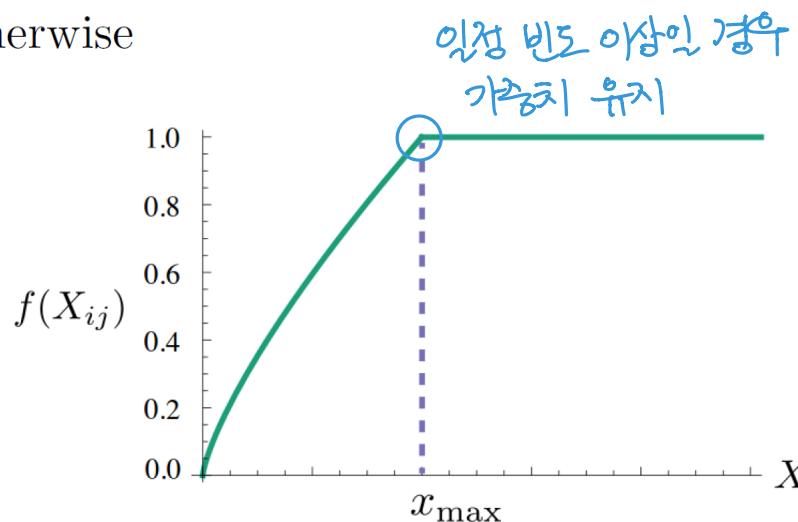
Pennington et al. (2014)

- Objective Function

✓ A least squared objective function

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

where $f(x) = \begin{cases} \left(\frac{x}{x_{\max}}\right)^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}$





Combining the best of both worlds

GloVe [Pennington et al., EMNLP 2014]

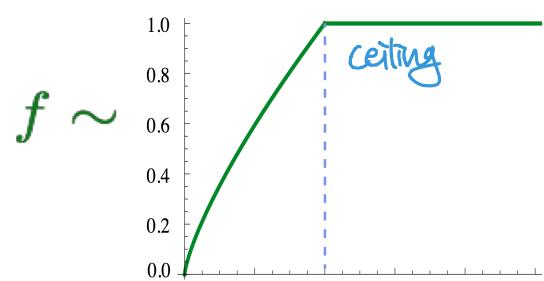


뛰어서 다른 것

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

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

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



GloVe results

Nearest words to
frog:

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



litoria



rana



leptodactylidae



eleutherodactylus

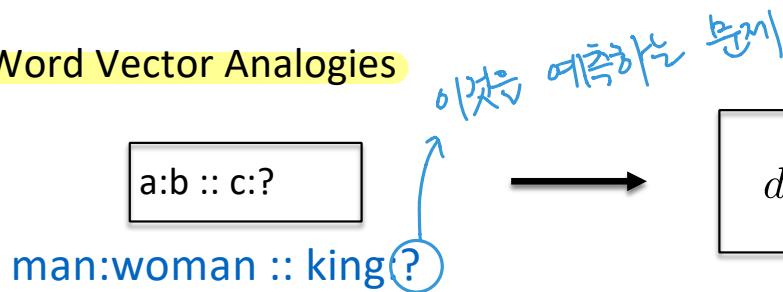
How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic: → 강의에서 더 짧은 문장으로 다룬.
 - 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 모든 구조의 문제인지, embedding의 문제인지 판단 어려움.
 - If replacing exactly one subsystem with another improves accuracy → Winning!

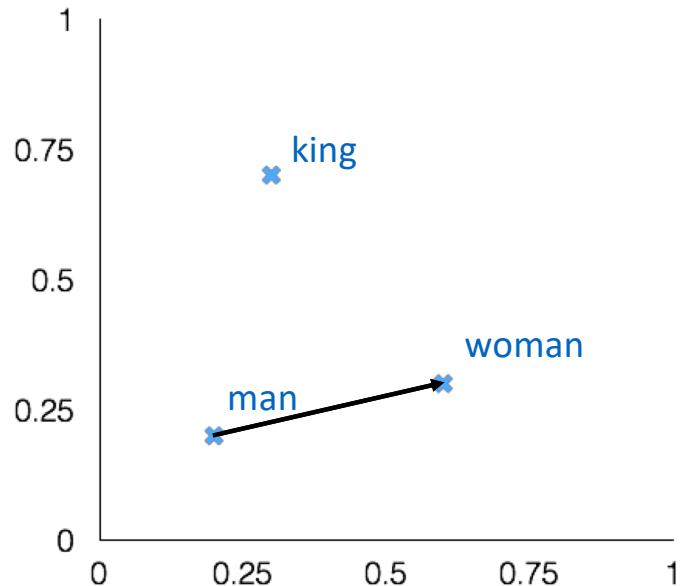
예) web search, question answering, phone dialog system, etc.

Intrinsic word vector evaluation

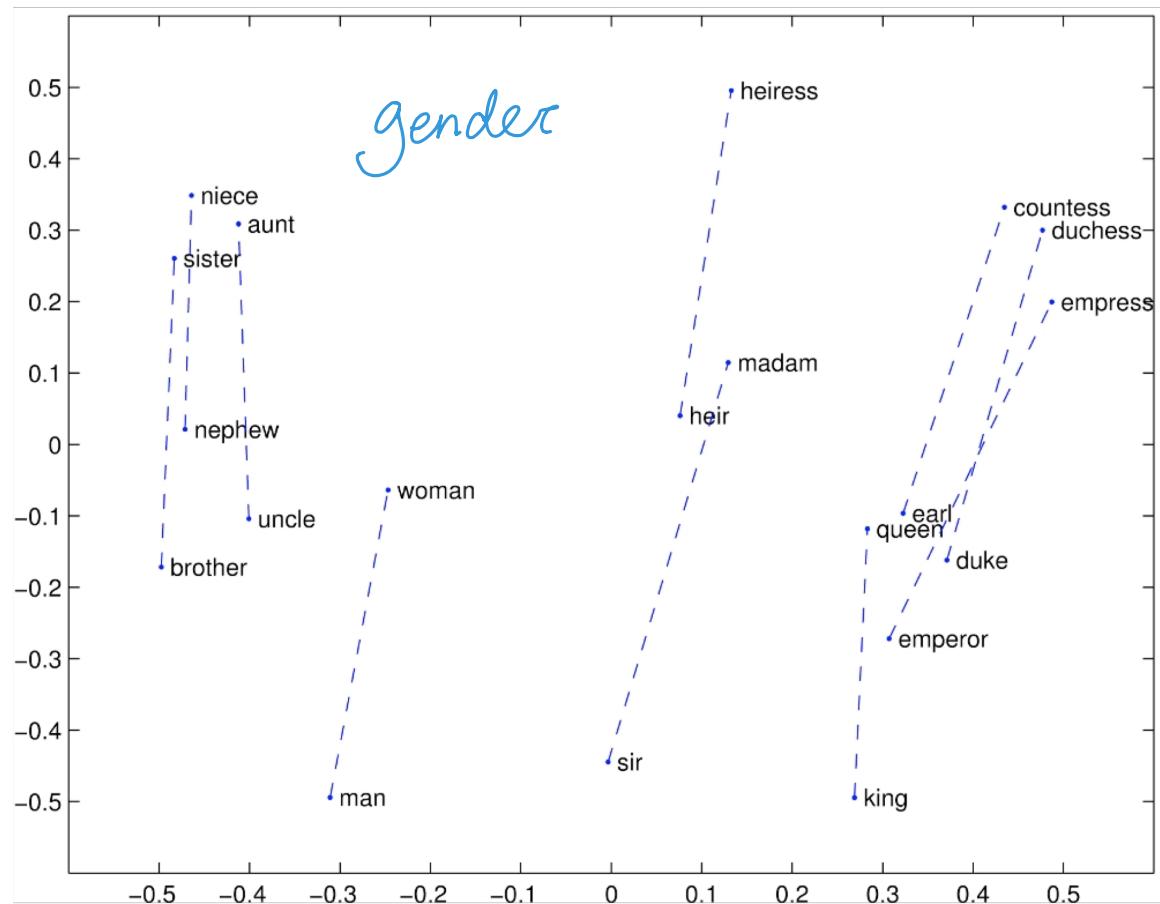
- Word Vector Analogies



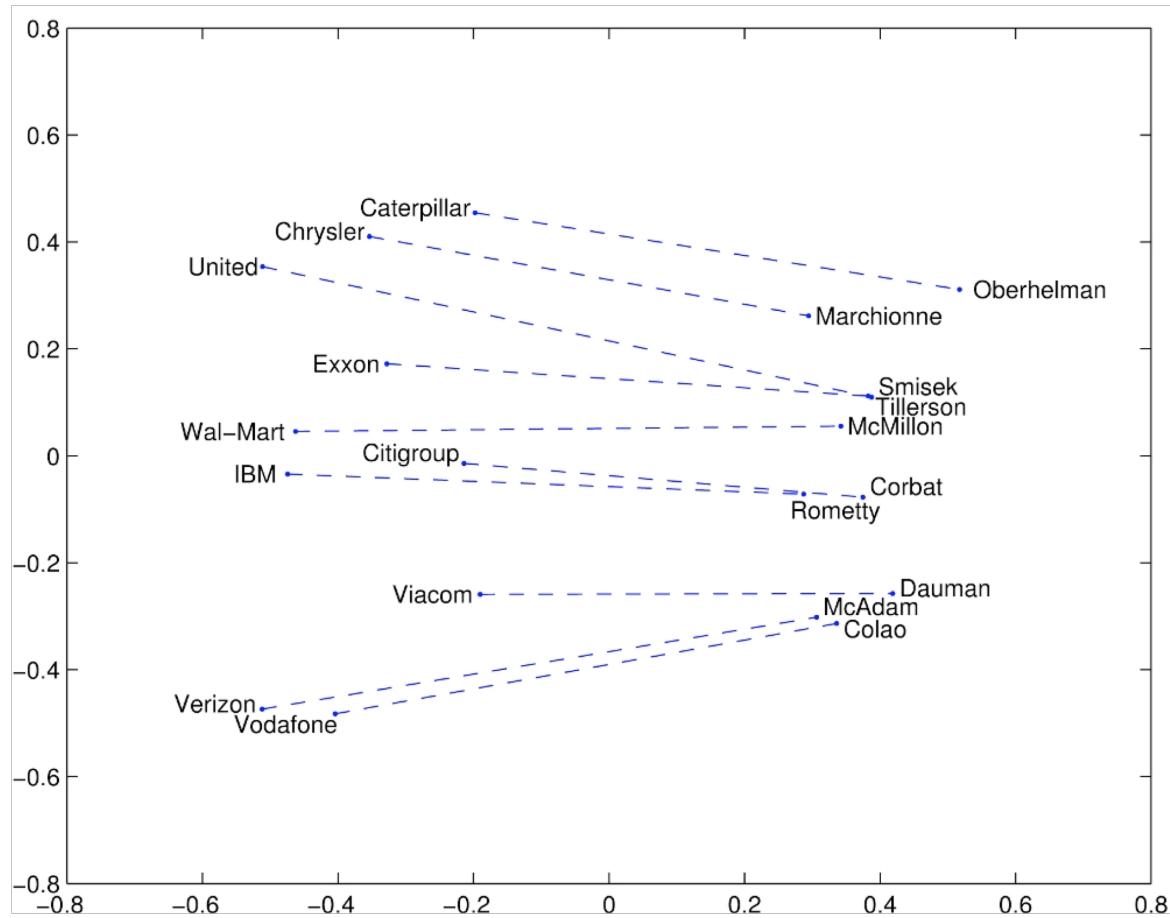
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



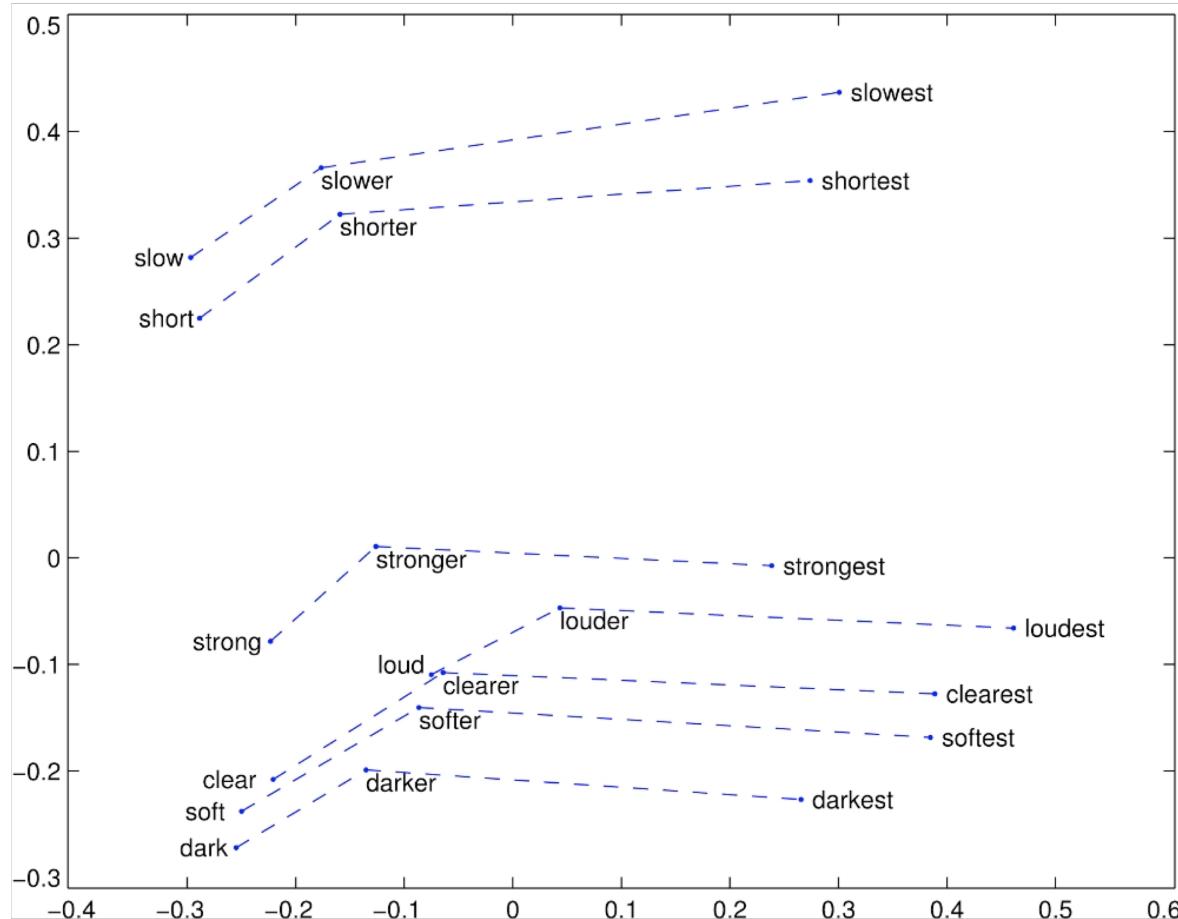
Glove Visualizations



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

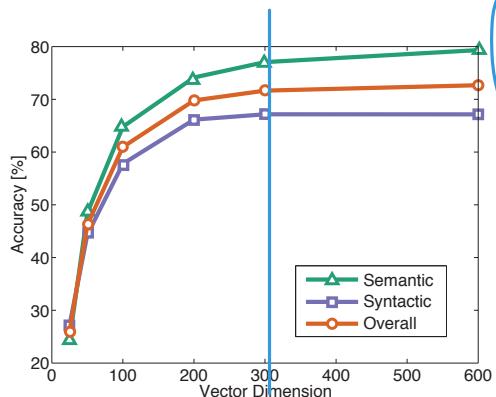
- 많은 양의 text 있을 때
→ bigger dim better
- text 예시 많을수록 better

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>

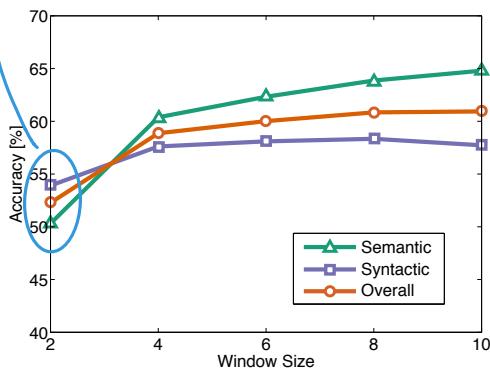
Analogy evaluation and hyperparameters

Syntactic (구문, 문법적) 효과는 local하기 때문에

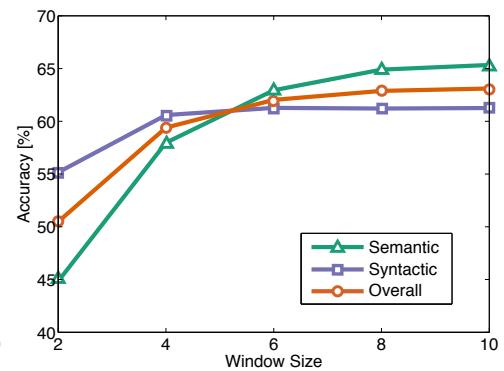
작은 window에서 더 효과가 좋다.



(a) Symmetric context



(b) Symmetric context



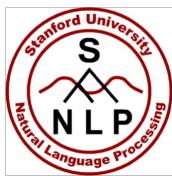
(c) Asymmetric context

Dimensionality

Window size

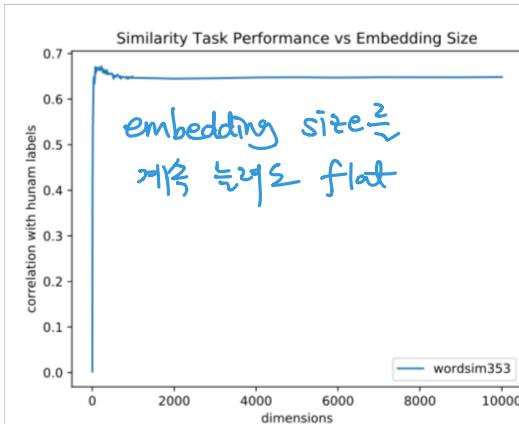
Window size

- 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



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

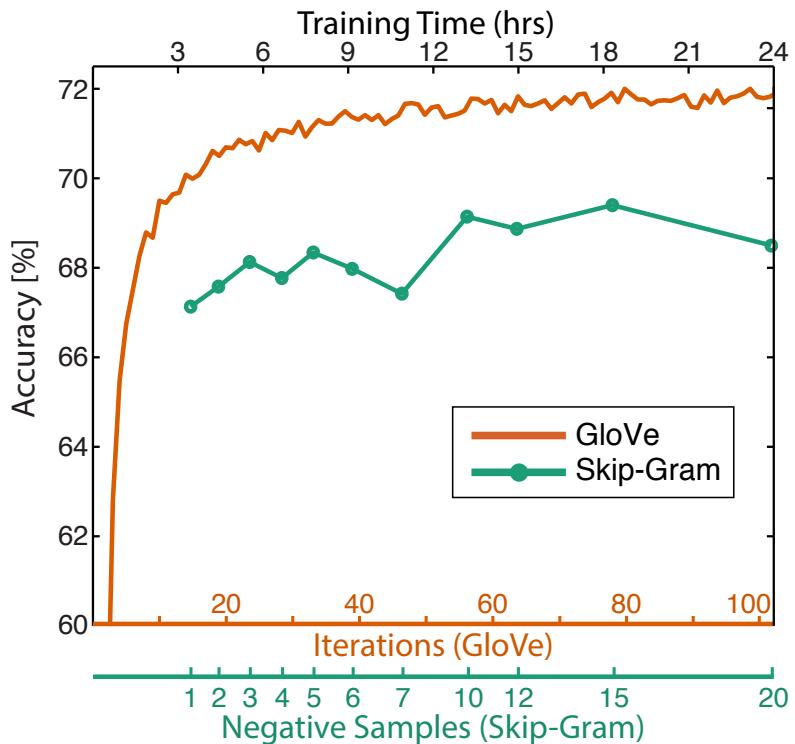
Using matrix perturbation theory, reveal a fundamental bias-variance trade-off in dimensionality selection for word embeddings

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

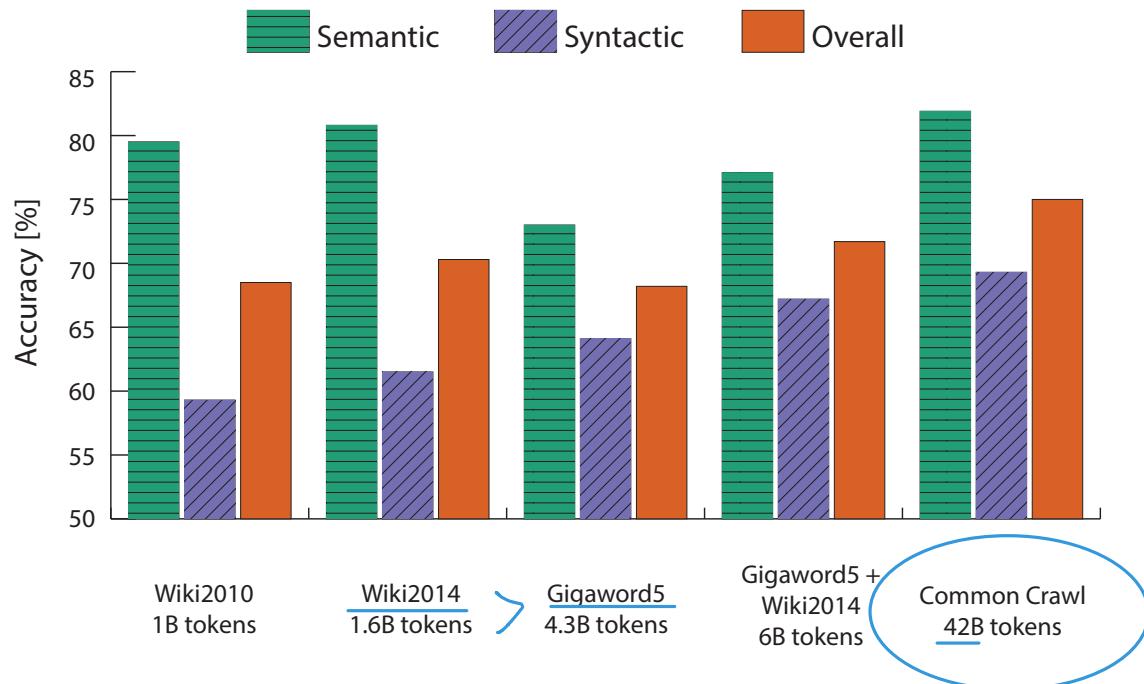
Analogy evaluation and hyperparameters

- More training time helps



Analogy evaluation and hyperparameters

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



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)	두 단어의 유사도는 1~10 척도로 표기
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
<hr/>	
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	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

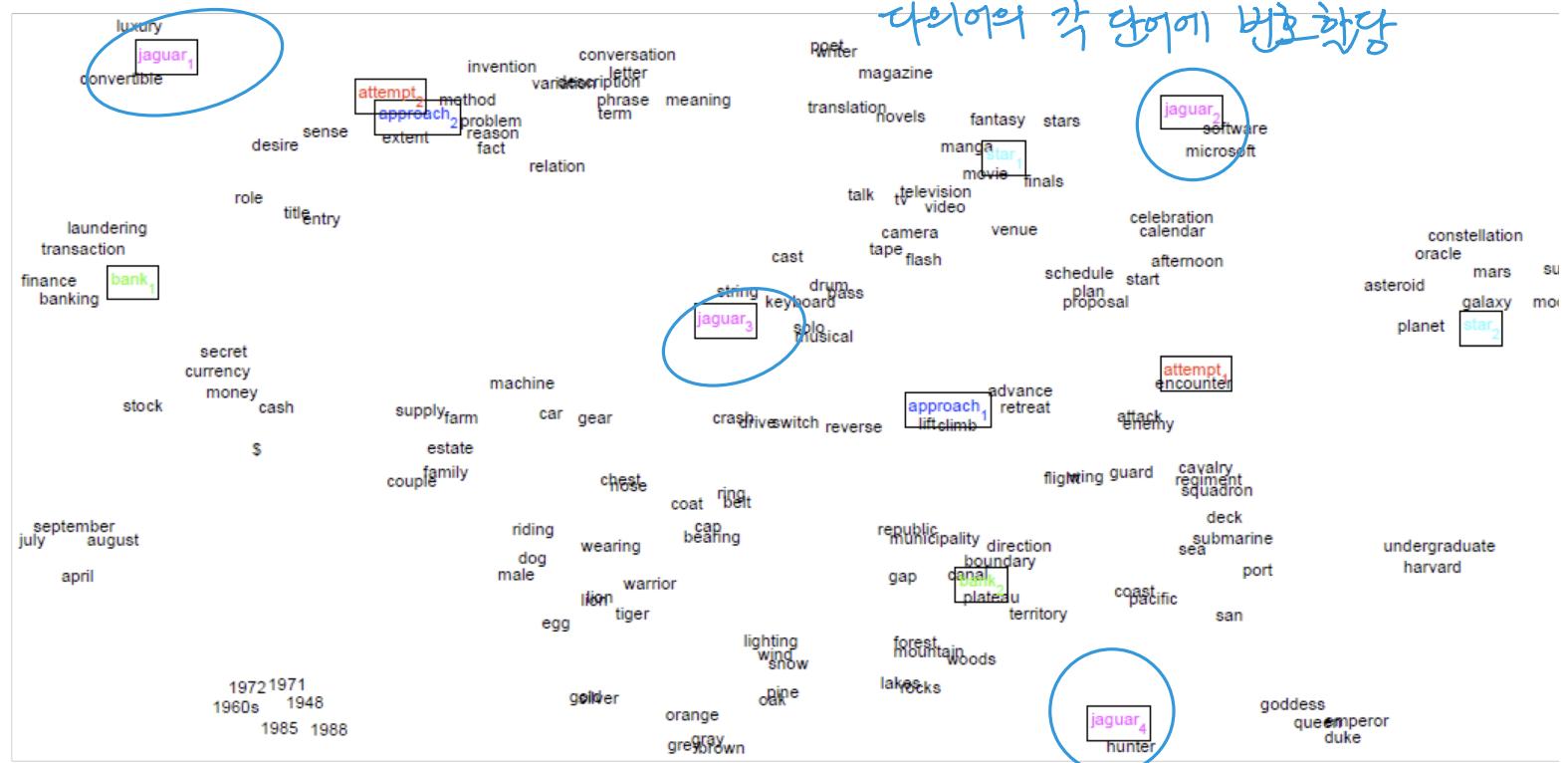
- Most words have lots of meanings!
 - Especially common words
 - Especially words that have existed for a long time
- Example: pike
- Does one vector capture all these meanings or do we have a mess?

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



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
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$ 각자의 뜻을 가진 단어의 가중합.
- 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)
→ 단어가 가질 수 있는 의미보다 더 많은 차원으로 구성
→ 이 중 k 개의 vectors로 단어를 표현
 m (전체): 2000
 k : 5

		tie	가 좋은 결과	
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!

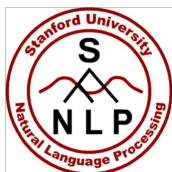
Course plan: coming weeks

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

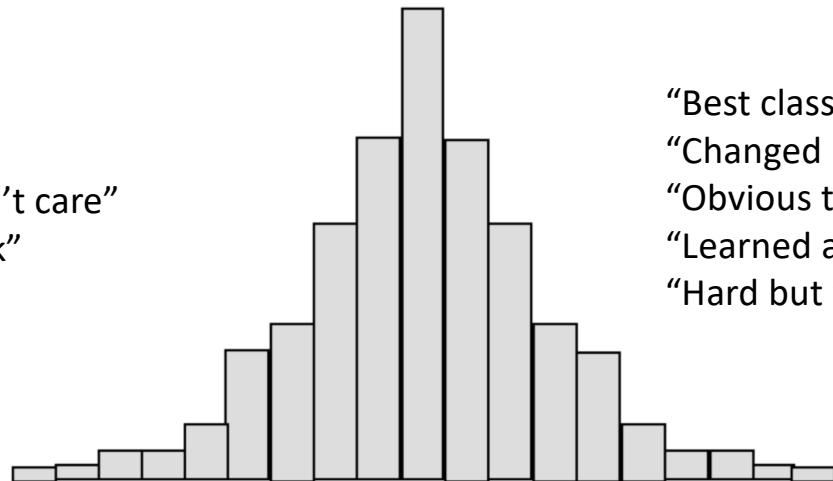
- We learn about putting syntactic structure (dependency parses) over sentence (this is HW3!)
- We develop the notion of the probability of a sentence (a probabilistic language model) and why it is really useful



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?