14기 심화세미나

ToBig's 14기 강의자

이혜린

Word Vectors and Word Senses

Contents

Unit 01 | Details of word2vec

- Text Model의 핵심!

  → 텍스트를 컴퓨터가 다룰 수 있게 숫자 (벡터, 행렬) 로 변환하는 것

  1. one hot encoding (Sparse representation)

  - 벡터가 0 또는 1의 값으로 구성됨 → 단어 간의 의미(유사성 등)을 반영하지 못함

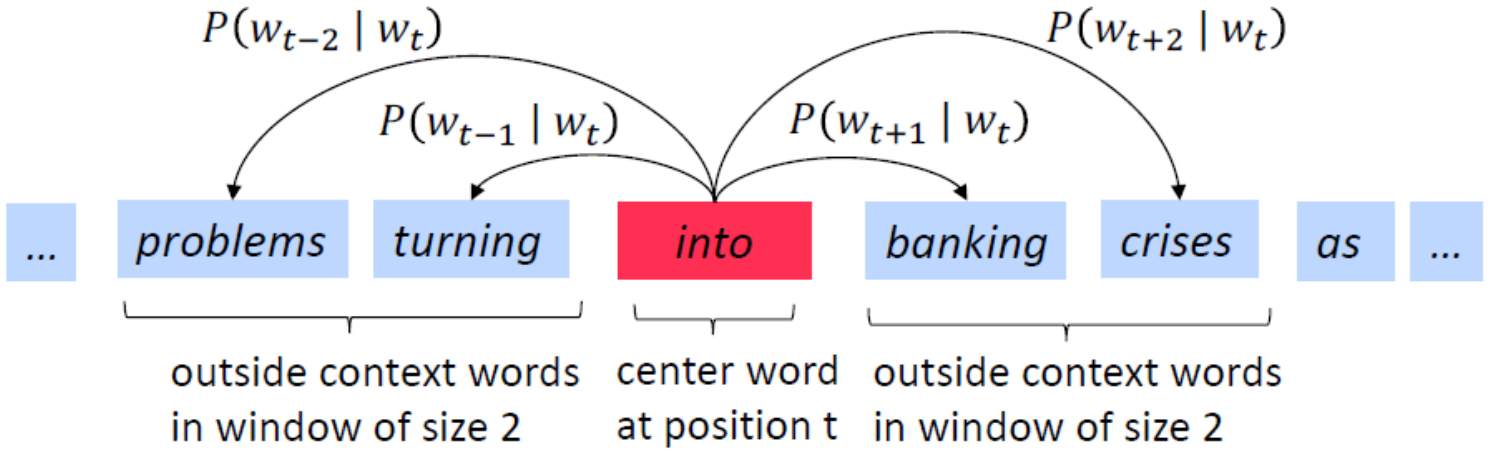
  2. 단어 임베딩 (Dense representation)

  - 단어를 실수로 구성된 벡터로 변환. 단어의 의미를 반영할 수 있도록 함.
  - 현대의 자연어 처리 기법들의 기반이 되는 방법
  - 차원의 저주 방지, 더 높은 일반화 수준 가능 (유사한 단어가 입력되었을 때 비슷한 출력 결과를 도출)
  - 모델 : word2vec, GloVe, FastText 등

Unit 01 | Details of word2vec

- word2vec Google

  Skip-gram 위주로 설명하도록 하겠습니다 :)

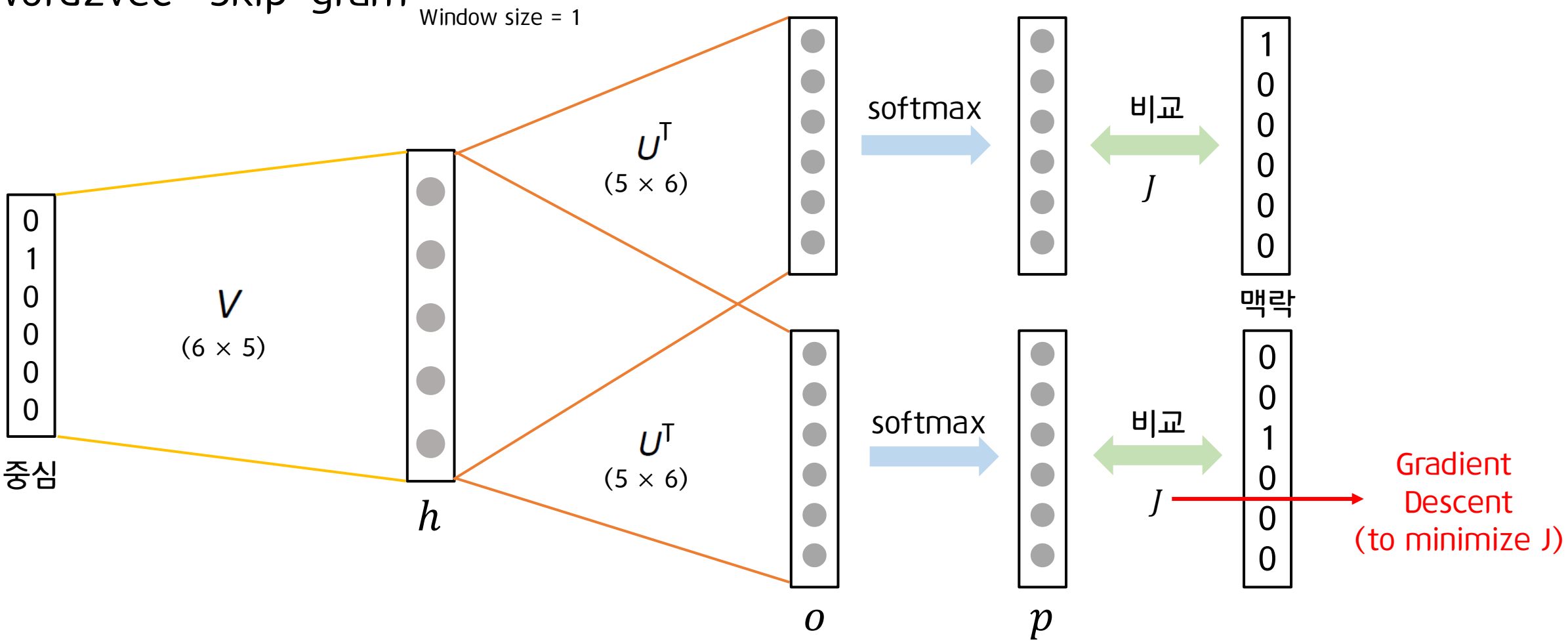


$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m \le j \le m \\ j \neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

m : 윈도우 개수

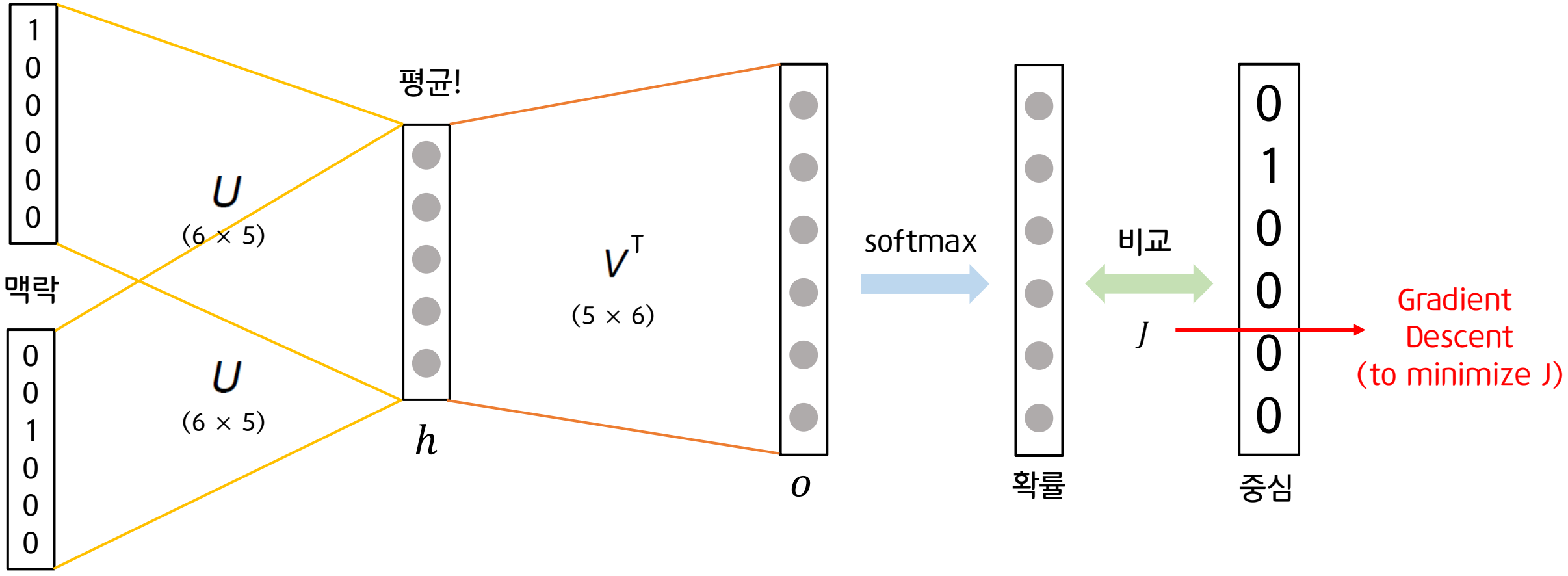
Unit 01 | Details of word2vec

- word2vec : Skip-gram



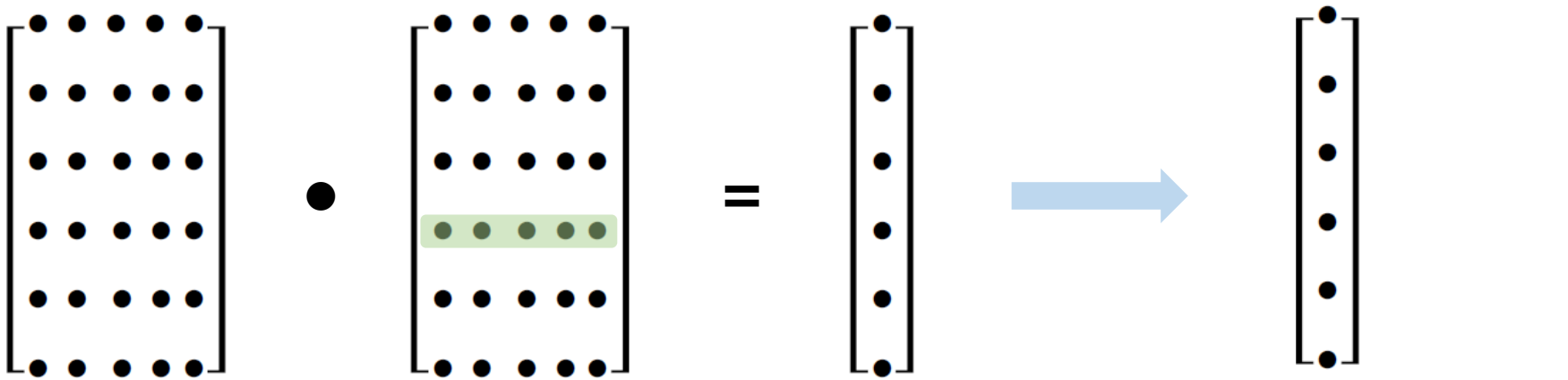
Unit 01 | Details of word2vec

- word2vec : CBOW (Continuous Bag of Words) Window size = 1



Unit 01 | Details of word2vec

- Parameters of word2vec



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

o : 맥락 단어 c : 중심 단어

Unit 01 | Details of word2vec

- Problem of Stochastic Gradient Descent $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$

Gradient descent는 전체 window에 대해서 진행 → 계산량 too much → SGD 이용 (또는 mini-batch도 가능)

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

그러나 SGD를 사용할 때에도 문제는 발생!

가중치 행렬 U, V의 dimension = (d x V)
parameter의 개수 = d x V + d x V = 2dV

이 중 0이 아닌 값을 갖는 건 많아도 2m+1 개 → 불필요한 계산 발생

즉 $\nabla_\theta J_t(\theta)$ 는 sparse한 vector

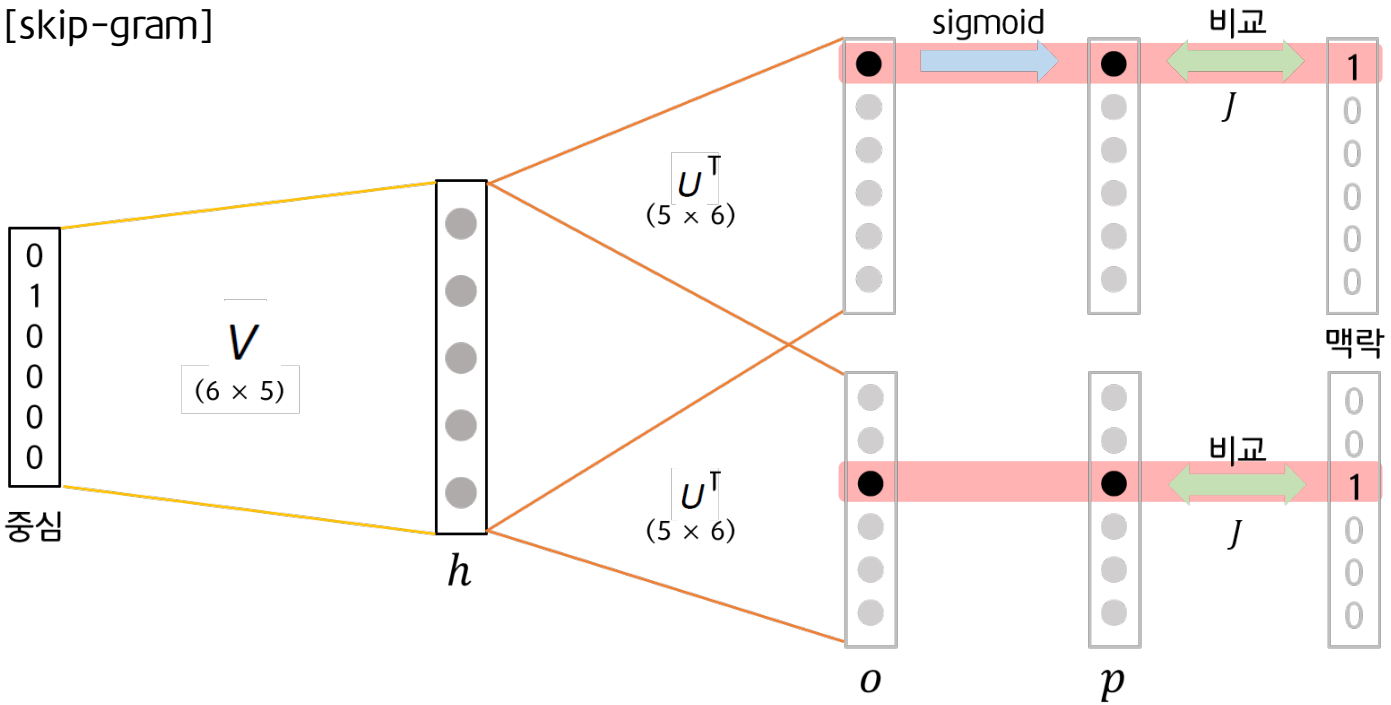
이처럼 feature가 많아서 발생하는 sparsity를 해결하기 위해
실제로 등장한 (=값이 0이 아닌) 행에 대해서만 gradient 계산하고 sparse한 matrix를 add, subtract 함으로써 gradient를 update

Unit 01 | Details of word2vec

- Negative Sampling

SGD에 이어서 word2vec의 효율성을 높일 수 있는 (= 계산량을 감소시킬 수 있는) 또 다른 방법.
기존의 다중 분류를 이진 분류로 근사 시킨다!

[skip-gram]



은닉층과 출력층의 가중치 행렬의 내적을 정답 벡터에서 값이 있는 부분에 해당되는 부분만을 대상으로 진행

하지만 이 경우, 긍정적 예(정답)에 대해서만 학습이 진행되기 때문에 k개의 부정적 예를 sampling하여 학습

Unit 01 | Details of word2vec

- Negative Sampling

Train 'binary logistic regressions for a true pair (center word and word in its context window)'

vs 'several noise pairs (the center word paired with a random word)'

[overall object function]

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

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

σ(양수) σ(음수)

o : 맥락 단어 c : 중심 단어

Unit 01 | Details of word2vec

- Negative Sampling

Train 'binary logistic regressions for a true pair (center word and word in its context window)'

vs 'K noise pairs (the center word paired with a random word)'

↑ negative sampling을 통해 뽑은 K개의 negative samples

[negative loglikelihood]

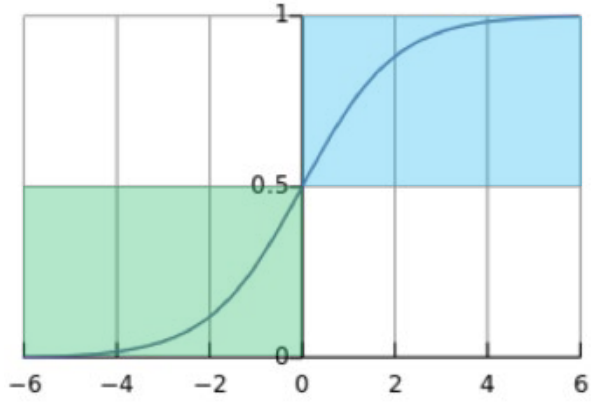
minimize $J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$

σ(양수) σ(음수)

True pair : $u^T v$ (코사인 유사도) 가 클수록 확률(=시그모이드 값)이 큼

Noise pair : $u^T v$ (코사인 유사도) 가 작을수록 확률(=시그모이드 값)이 큼

[Sigmoid ftn]



+ 일 때 $0.5 \leq p < 1$
- 일 때 $0 \leq p < 0.5$

Unit 01 | Details of word2vec

- Negative Sampling

Train 'binary logistic regressions for a true pair (center word and word in its context window)'

vs 'K noise pairs (the center word paired with a random word)'

negative sampling을 통해 뽑은 K개의 negative samples

[negative loglikelihood]

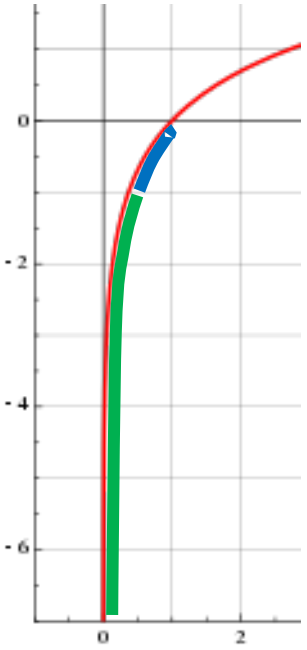
minimize $J_{neg-sample}(\boldsymbol{o}, \boldsymbol{v}_c, \boldsymbol{U}) = -\log(\sigma(\boldsymbol{u}_o^\top \boldsymbol{v}_c)) - \sum_{k=1}^{K} \log(\sigma(-\boldsymbol{u}_k^\top \boldsymbol{v}_c))$

σ(양수) σ(음수)

True pair : $u^T v$ 가 클수록 확률(=시그모이드 값)이 큼 → 로그 값이 0에 가까움

Noise pair : $u^T v$ 가 작을수록 확률(=시그모이드 값)이 큼 → 로그 값이 0에 가까움

[자연로그 ftn]



Unit 01 | Details of word2vec

- Negative Sampling

Train 'binary logistic regressions for a true pair (center word and word in its context window)'

vs 'K noise pairs (the center word paired with a random word)'

↑ negative sampling을 통해 뽑은 K개의 negative samples

[negative loglikelihood]

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

σ(양수) σ(음수)

True pair : $u^T v$ 가 클수록 (중심 단어와 맥락 단어가 같이 나올 확률이 클수록) 손실 J가 0에 가까움

Noise pair : $u^T v$ 가 작을수록 (중심 단어와 랜덤 단어가 같이 나올 확률이 작을수록) 손실 J가 0에 가까움

Unit 01 | Details of word2vec

- Negative Sampling

[negative sampling 방법]

K : small data의 경우 5-20개, big data의 경우 2-5개로 지정해야 성능이 좋음

Sampling 기준 : 해당 단어가 출현할 확률 (given)

$$P'(w_i) = \frac{P(w_i)^{0.75}}{\sum_j^n P(w_j)^{0.75}} = \frac{U(w_i)^{0.75}}{Z}$$

^0.75 : 낮은 확률의 단어가 (조금 더) 쉽게 샘플링 되게 해준다!

p = [0.7 0.29 0.01] → p^0.75 = [0.6419 0.3315 0.0265]

Unit 01 | Details of word2vec

- Subsampling Frequent words

말뭉치에서 자주 등장하는 단어는 학습량을 확률적으로 감소시킴 (등장빈도 만큼 업데이트 될 기회가 많으므로)

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

$f(w_i)$: 해당 단어 빈도 / 전체 단어 수 (=단어가 corpus에 등장한 비율)

t : 보통 0.000001 (연구자가 결정)

자주 등장하는 단어는 $p(w_i)$ 만큼 학습에서 제외. 드물게 등장하는 단어는 거의 나올 때 마다 학습.

Unit 02 | co-occurrence matrix

- co-occurrence counts matrix

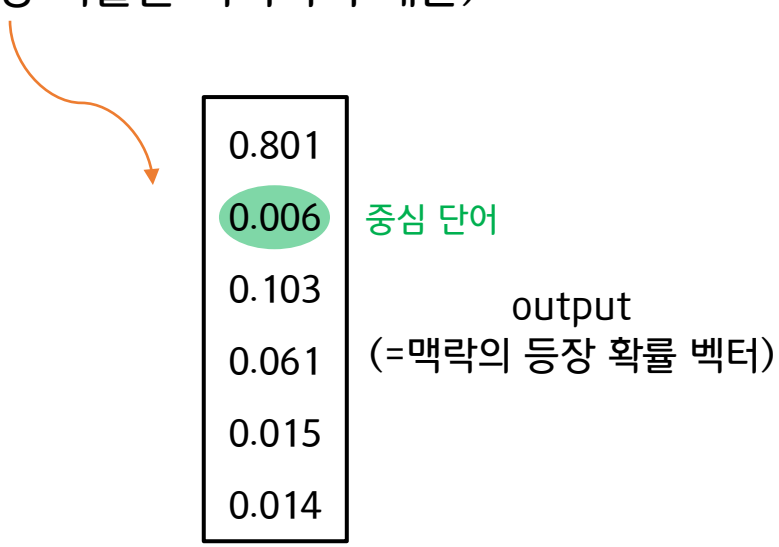
Skip-gram

- 중심 단어를 기준으로 맥락 단어가 등장할 확률을 계산
- 윈도우 개수를 아무리 크게 늘려도, global co-occurrence statistics (전체 단어의 동반출현 빈도수)와 같은 통계 정보는 내포할 수 없음 (벡터의 값들은 중심 단어가 given일 때 각 값의 개별적인 등장 확률을 의미하기 때문)

→ count-based의 co-occurrence matrix 생성

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

Window size = 1



Unit 02 | co-occurrence matrix

- co-occurrence counts matrix

Co-occurrence matrix : 동반출현 빈도수에 대한 행렬

1) Window based co-occurrence matrix (단어-문맥 행렬)

- word2vec과 비슷한 방식. 한 문장을 기준으로 윈도우에 각 단어가 몇 번 등장하는 지 count
- Syntatic (POS; post (noun) post office (noun) other synonyms) and semantic 정보 획득 가능

- I like an apple.
- I like you.

⋮

	I	like	an	apple	you	.
I	0	2	0	0	0	0
like	2	0	1	0	1	0
an	0	1	0	1	0	0
apple	0	0	1	0	0	1
you	0	1	0	0	0	1
.	0	0	0	1	1	0

...

Window size = 1

Unit 02 | co-occurrence matrix

- co-occurrence counts

Co-occurrence matrix : 동반출현 빈도수에 대한 행렬

2) Word-Document matrix (단어-문서 행렬)

- 한 문서를 기준으로 각 단어가 몇 번 등장하는 지 count
- 문서에 있는 많은 단어들 중 빈번하게 등장하는 특정 단어가 존재한다는 것을 전제
- LSA (Latent Semantic Analysis; 잠재적 의미 분석) 가능하게 함 (ex. 문서 간 유사도 측정 등)

-	doc1	doc2	doc3
나	1	0	0
는	1	1	2
학교	1	1	0
에	1	1	0
가	1	1	0
ㄴ	1	0	0
다	1	0	1
영희	0	1	1
좋	0	0	1

Unit 02 | co-occurrence matrix

- co-occurrence counts

그러나 이와 같은 count-based matrix는

- 단어의 개수가 증가할수록 차원이 폭발적으로 증가

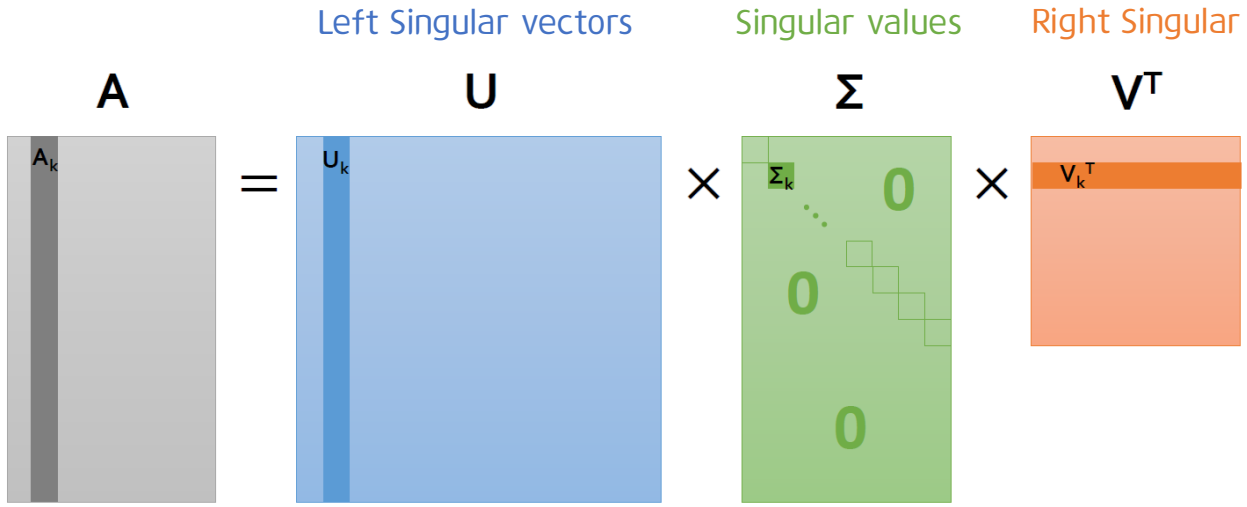
→ SVD (Singular Value Decomposition; 특이값 분해), LSA 등을 이용하여 차원을 축소시킨 후 사용

→ 대부분의 정보를 작은 차원의 행렬안에 포함시킬 수 있음 (like 주성분 분석)

Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

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

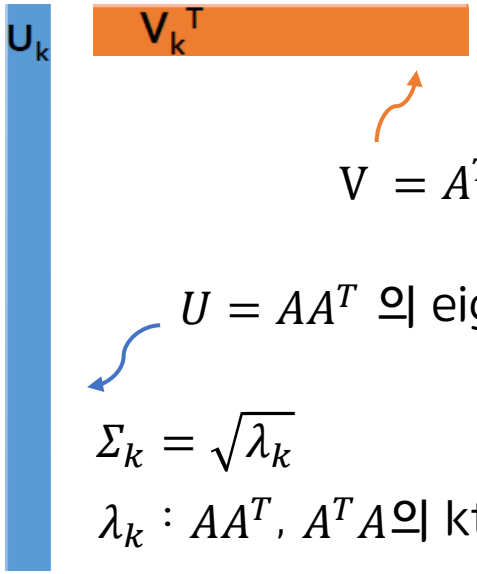


정방 행렬이 아니어도 가능!
(↔ 고유값 분해)

$UU^T = I$
orthogonal

Diagonal

$VV^T = I$
orthogonal



$V = A^T A$의 eigenvector 행렬

$U = AA^T$ 의 eigenvector 행렬

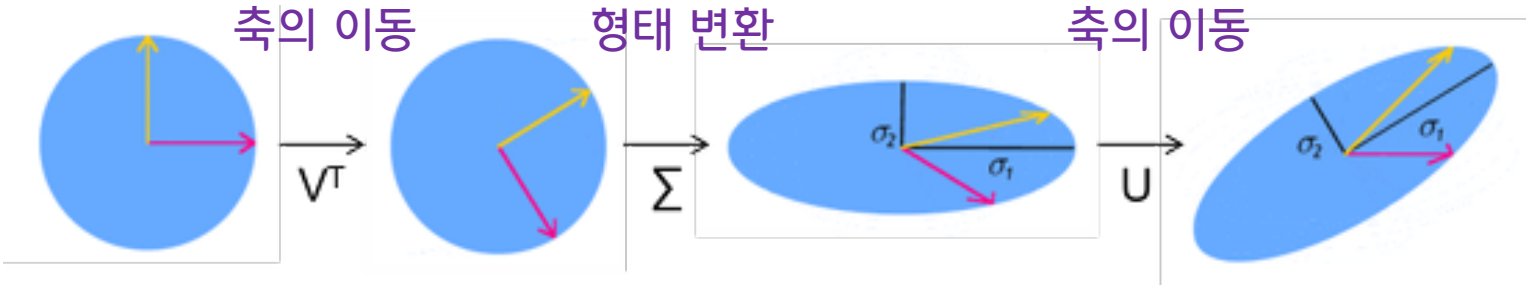
$\Sigma_k = \sqrt{\lambda_k}$

λ_k : AA^T, $A^T A$의 kth 고유값 (eigen value)

Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

$$A = U\Sigma V^T \longrightarrow Ax = U\Sigma V^T x$$



x $V^T x$ $\Sigma V^T x$ $U\Sigma V^T x$

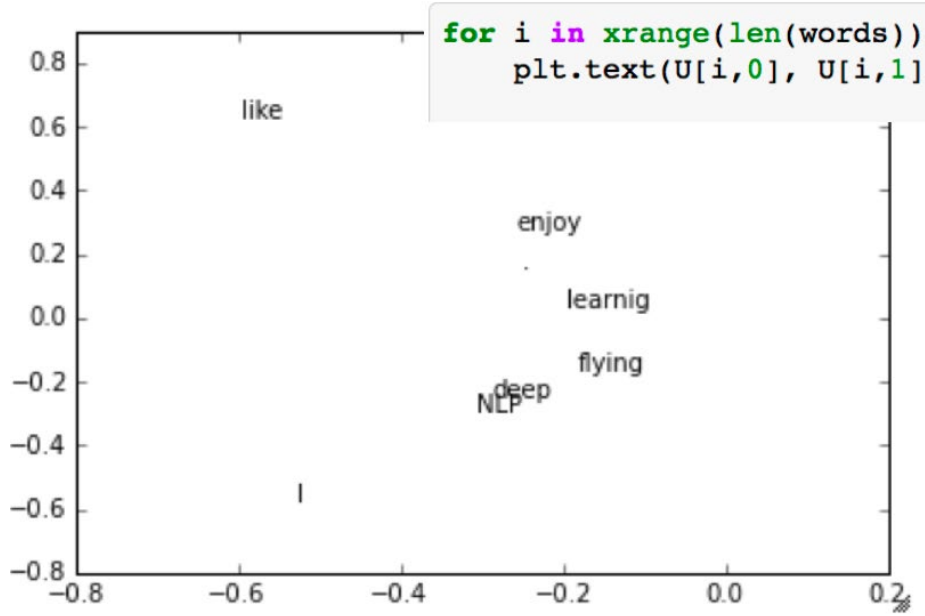
Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

```
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)
```

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



Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

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

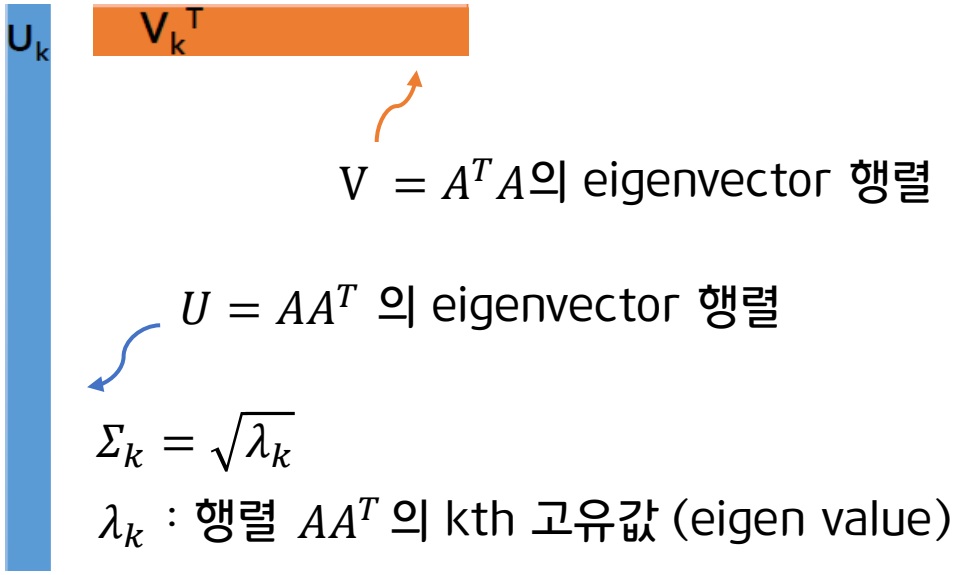
$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 2 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -0.17 & 0.27 & -0.40 \\ -0.63 & -0.41 & -0.03 \\ -0.32 & 0.37 & 0.21 \\ -0.32 & 0.37 & 0.21 \\ -0.32 & 0.37 & 0.21 \\ -0.17 & 0.27 & -0.40 \\ -0.33 & -0.12 & -0.52 \\ -0.30 & -0.29 & 0.49 \\ -0.15 & -0.39 & -0.13 \end{bmatrix} \begin{bmatrix} 3.61 & 0 & 0 \\ 0 & 2.04 & 0 \\ 0 & 0 & 1.34 \end{bmatrix} \begin{bmatrix} -0.63 & -0.53 & -0.57 \\ 0.56 & 0.20 & -0.80 \\ -0.54 & 0.83 & -0.17 \end{bmatrix}$$

λ_k : 내림차순. Non-zero

$UU^T = I$
orthogonal

Diagonal Matrix (대각행렬)

$VV^T = I$
orthogonal



Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

truncated SVD

$$A' = U'\Sigma'V'^T$$

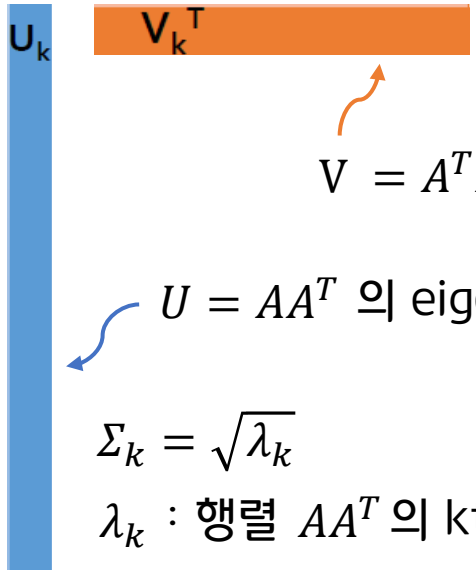
$$\begin{bmatrix} 0.71 & 0.44 & -0.09 \\ 0.97 & 1.04 & 1.99 \\ 1.15 & 0.76 & 0.04 \\ 1.15 & 0.76 & 0.04 \\ 1.15 & 0.76 & 0.04 \\ 0.71 & 0.45 & -0.09 \\ 0.62 & 0.58 & 0.88 \\ 0.36 & 0.45 & 1.11 \\ -0.09 & 0.14 & 0.97 \end{bmatrix} = \begin{bmatrix} -0.17 & 0.27 \\ -0.63 & -0.41 \\ -0.32 & 0.37 \\ -0.32 & 0.37 \\ -0.32 & 0.37 \\ -0.17 & 0.27 \\ -0.33 & -0.12 \\ -0.30 & -0.29 \\ -0.15 & -0.39 \end{bmatrix} \begin{bmatrix} 3.61 & 0 \\ 0 & 2.04 \end{bmatrix} \begin{bmatrix} -0.63 & -0.53 & -0.57 \\ 0.56 & 0.20 & -0.80 \end{bmatrix}$$

λ_k : 내림차순. Non-zero

$U'U'^T = I$ orthogonal

Diagonal Matrix (대각행렬)

$V'V'^T = I$ orthogonal



$V = A^T A$의 eigenvector 행렬

$U = AA^T$ 의 eigenvector 행렬

$\Sigma_k = \sqrt{\lambda_k}$

λ_k : 행렬 AA^T 의 kth 고유값 (eigen value)

Unit 02 | co-occurrence matrix

- SVD (Singular Value Decomposition; 특이값 분해)

truncated SVD 를 이용한 LSA

열 : 문서

$$X_1' = U'^T A' = U'^T U' \Sigma' V'^T : \begin{bmatrix} -2.28 & -1.90 & -2.07 \\ 1.14 & 0.42 & -1.64 \end{bmatrix}$$

$$X_2' = A'V' = U'\Sigma' V'^T V' = \begin{bmatrix} -0.63 & 0.56 \\ -2.30 & -0.84 \\ -1.16 & 0.76 \\ -1.16 & 0.76 \\ -1.16 & 0.76 \\ -0.63 & 0.56 \\ -1.20 & -0.24 \\ -1.10 & -0.60 \\ -0.57 & -0.80 \end{bmatrix}$$

행 : 단어

열 : 문서

-	doc1	doc2	doc3
나	1	0	0
는	1	1	2
학교	1	1	0
에	1	1	0
가	1	1	0
ㄴ	1	0	0
다	1	0	1
영희	0	1	1
좋	0	0	1

행 : 단어

Unit 02 | co-occurrence matrix

- Hacks to X

그러나 여전히 문제점은 있을 유… → functional words (a, the, …) 이 너무 자주 등장한다!

이를 보완하기 위해,

- 불용어 (stop words) 제거
- min(X, t), t = 100 ~ ; count가 100이 넘는 단어들은 다 100으로 처리 (t값은 연구자 마음~ 데이터의 max값에 따라 적절히 선택하면 될 것 같아요!)
- Window size에서 단어의 거리를 반영하는 ramped window 사용 (중심 단어에서 가까울수록 큰 가중치 부여)
- 단순 count가 아닌, Pearson correlation 사용 (negative value는 0으로 변환)

Unit 02 | co-occurrence matrix

- Count based vs. direct prediction

Count-based

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

- 빠른 훈련 속도
- 효율적으로 통계정보 사용

- 주로 단어 유사성 여부만을 파악하는 데에 사용 (단어 간 관계는 파악 불가)
- 큰 빈도수에 과도하게 중요도 부여

Direct prediction

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

- 높은 수준의 성능
- 단어 유사성 이상의 복잡한 패턴을 파악

- 말뭉치 크기가 성능에 영향을 미침
- 효율적으로 통계정보를 사용하지 못함

GloVe

통계정보를 포함한
Direct prediction embedding

Unit 03 | GloVe

- GloVe (Global Vectors for Word Representation)

임베딩된 단어벡터 간 유사도 측정을 수월하게 하면서 … word2vec 의 장점

말뭉치 전체의 통계 정보를 반영하자! … co-occurrence matrix의 장점

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

임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

X_{ik} ; 전체 말뭉치 중 사용자가 정한 window 내에, i번째 단어와 k번째 단어가 동시에 등장하는 횟수

$X_i = \sum_k X_{ik}$; 전체 말뭉치 중 사용자가 정한 window 내에, i번째 단어가 등장하는 횟수

$P_{ik} = P(k|i) = \frac{X_{ik}}{X_i}$; i번째 단어 (context word) 주변에 k번째 단어가 등장할 조건부 확률

Unit 03 | GloVe

- GloVe

Probability 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}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

$$\frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9 \times 10^{-4}}{2.2 \times 10^{-5}} = 8.9$$

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input　　F의 output

$$F(w_{ice}, w_{steam}, w_{solid}) = \frac{P_{ice,solid}}{P_{steam,solid}} = \frac{P(solid|ice)}{P(solid|steam)} = \frac{1.9 \times 10^{-4}}{2.2 \times 10^{-5}} = 8.9$$

8.9라는 결과를 만들어주는 함수 F를 찾아야 한다!

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

i번째 단어 (맥락 단어)가 있을 때 k번째 단어가 등장할 조건부 확률

i번째 단어 (맥락 단어)가 있을 때 j번째 단어가 등장할 조건부 확률

두 단어벡터의 내적이 input으로 들어가게 만들어야 한다!

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input　　F의 output

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

세 단어 벡터의 함수 → 두 단어 벡터의 함수

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

두 단어 벡터의 함수 → 두 단어 벡터의 내적

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

동시 발생 확률의 비를, 단어 벡터 스페이스 내에 선형으로 표현!

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input　　F의 output

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

조건부 확률 → 함수 F 형태로 변환

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

$F(\mathrm{A} - \mathrm{B}) = \frac{F(\mathrm{A})}{F(\mathrm{B})}$ 의 형태 → F는 지수함수

$e^{\mathrm{A}-\mathrm{B}} = e^{\mathrm{A}}/e^{\mathrm{B}}$

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

$$exp(w_i{}^T \tilde{w}_k) = P_{ik} \longrightarrow w_i{}^T \tilde{w}_k = log P_{ik}$$

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input　　F의 output

$$w_i^T \tilde{w}_k = logP_{ik} = logP(k|i) = logX_{ik} - logX_i$$

근데… 함수 F는 Homomorphism(준동형)을 만족해야 한다. 그 이유를 추측해보기론… ㅠㅜ

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

co-occurrence matrix = co−occurrence matrix^T

$$w_i^T \tilde{w}_k = w_k^T \tilde{w}_i$$
$$logP_{ik} = logP_{ki}$$

단어벡터 내적 = 단어벡터 내적^T

$$logX_{ik} - logX_i = logX_{ki} - logX_k$$

이 성립해야 하는데, 그렇지 않다
이를 보정하기 위해 상수항을 더해준다

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input　　F의 output

$$w_i{}^T \tilde{w}_k = logX_{ik} - logX_i = logX_{ik} - b_i - \tilde{b_k}$$

$$w_i{}^T \tilde{w}_k + b_i + \tilde{b_k} = logX_{ik}$$

동시 등장 횟수에 대한 정보를 담고 있는
co-occurrence matrix

$$\underset{w_i, \tilde{w}_k, b_i, \tilde{b_k}}{\text{argmin}} \quad J = \sum_{i,j=1}^{V} (w_i{}^T \tilde{w}_k + b_i + \tilde{b_k} - logX_{ik})^2$$

Unit 03 | GloVe

- GloVe

Crucial Insight : 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의

F의 input F의 output

$$\underset{w_i, \tilde{w}_j, b_i, \tilde{b}_j}{\text{argmin}} \quad J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2 \quad , \quad f(X_{ij}) = \begin{cases} \left(\frac{x}{x_{max}}\right)^{\alpha} & if\ x < x_{max} \\ 1, & otherwise \end{cases}$$

특정 단어가 지나치게 빈도수가 높아서
X_{ij} 가 튀는 현상을 방지 하기 위해 추가한 함수

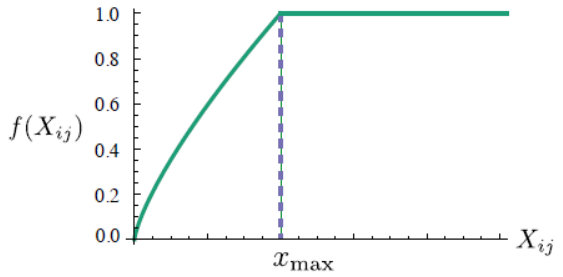


Figure 1: Weighting function f with $\alpha = 3/4$.

- 빠른 훈련
- 말뭉치 크기가 성능에 미치는 영향을 제한할 수 있음
- 작은 말뭉치, 작은 벡터에도 좋은 성능

Unit 03 | GloVe

- GloVe Results : Frog

Nearest words to frog:

개구리들 1. frogs
두꺼비 2. toad
청개구리 3. litoria
긴발가락개구리과… 4. leptodactylidae
개구리 5. rana
도마뱀 6. lizard
가는발가락개구리과… 7. eleutherodactylus

litoria

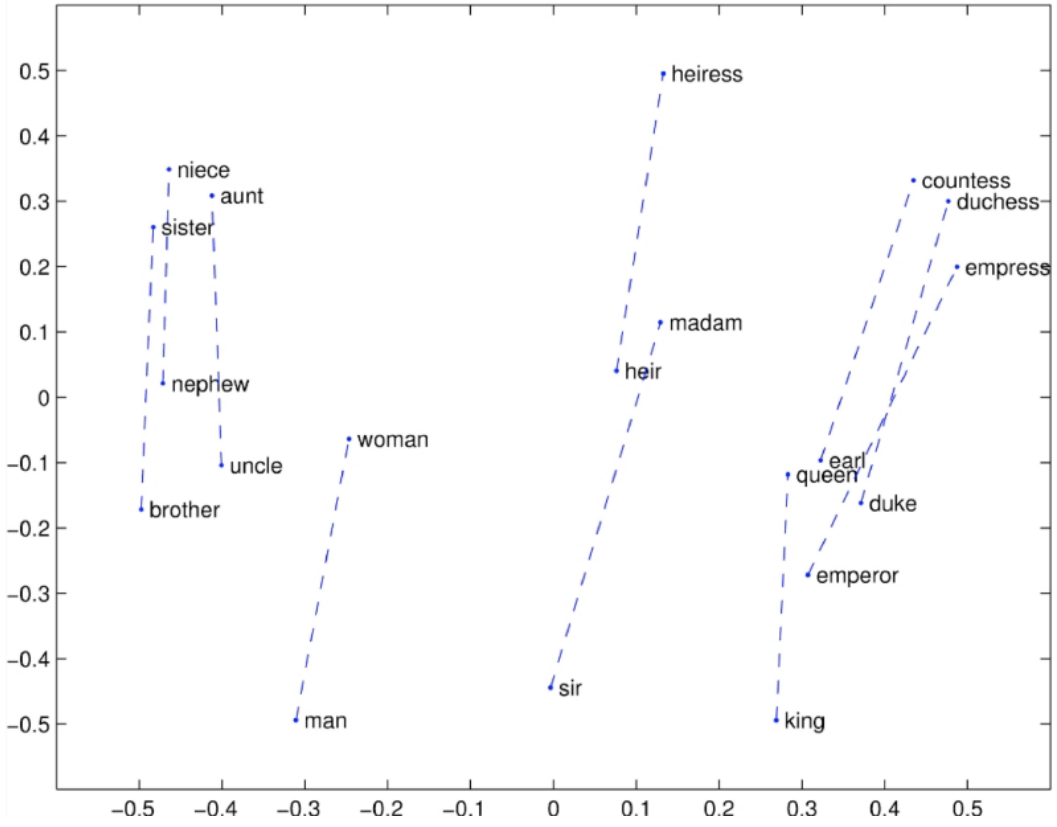
leptodactylidae

rana

eleutherodactylus

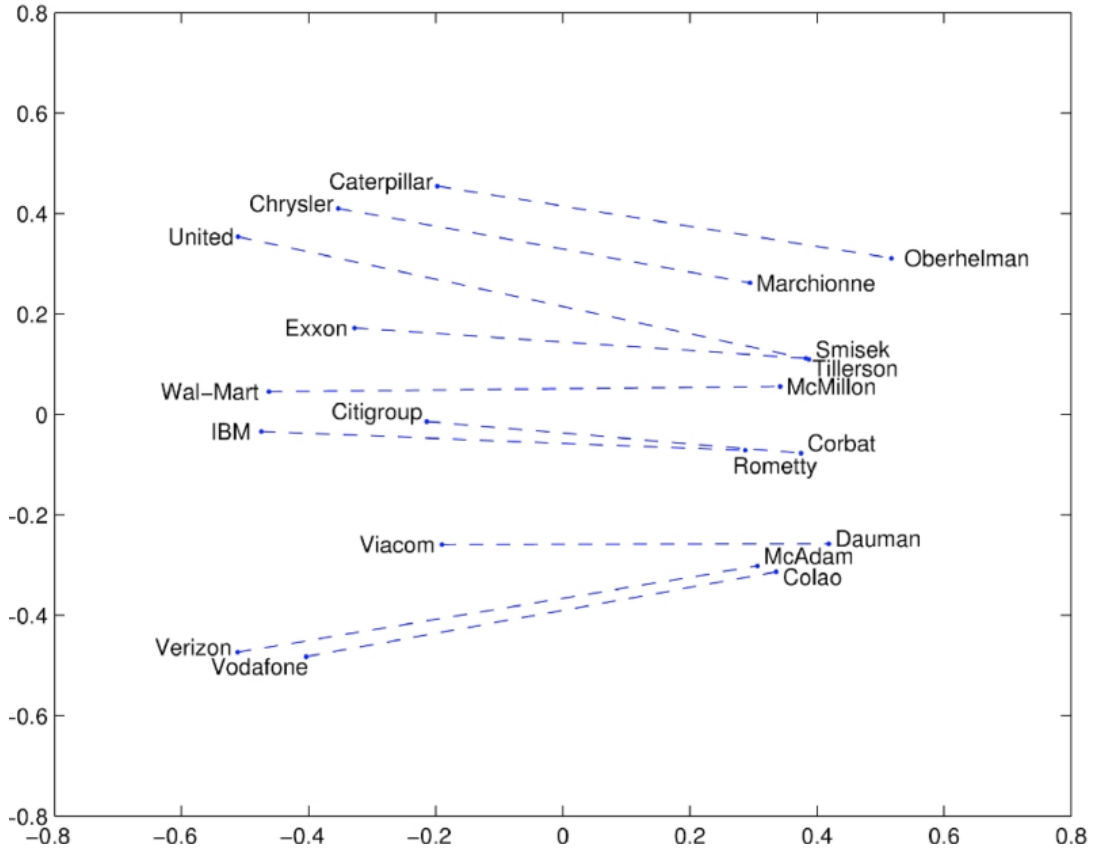
Unit 03 | GloVe

- GloVe Results : opposite



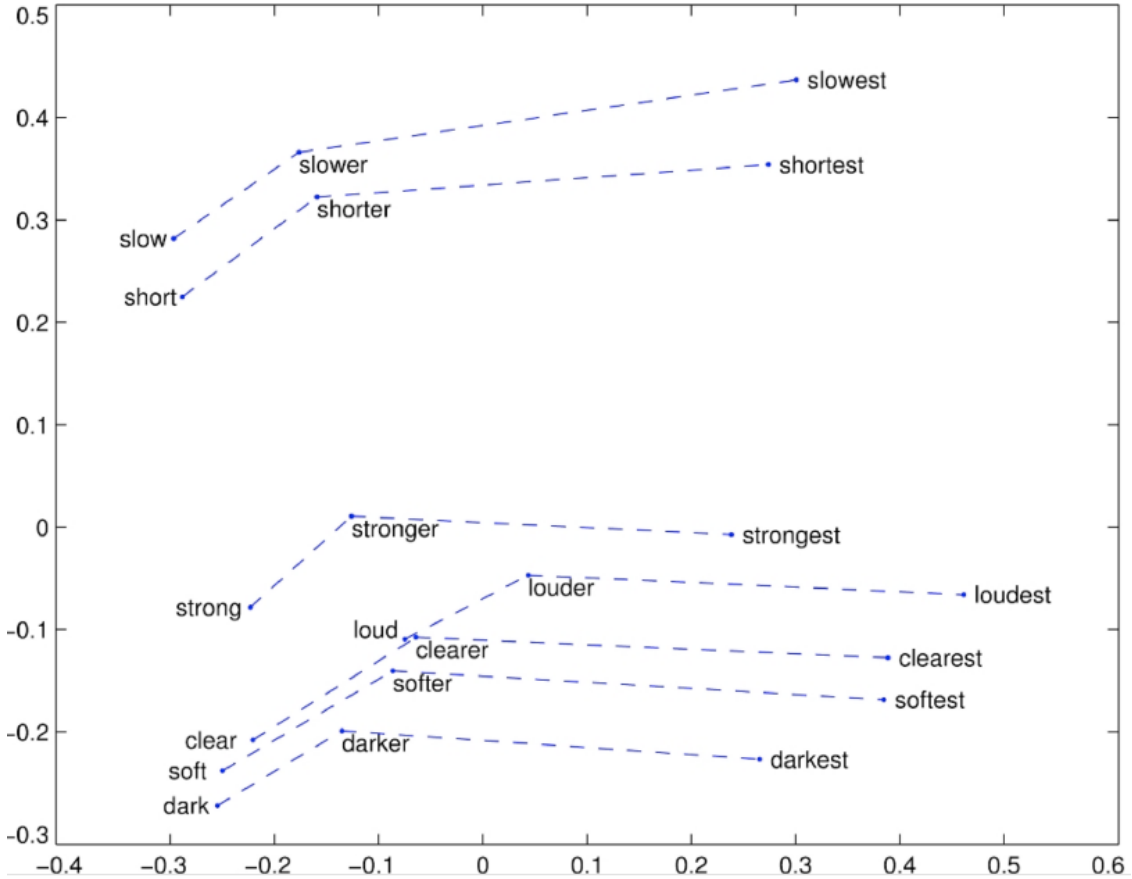
Unit 03 | GloVe

- GloVe Results : company – CEO



Unit 03 | GloVe

- GloVe Results : Superlatives



Unit 03 | GloVe

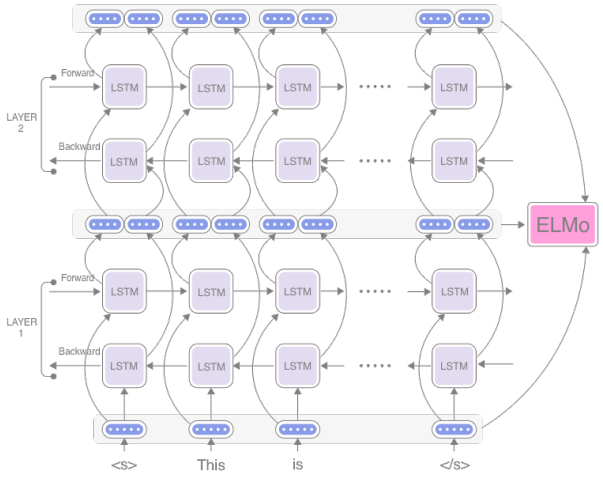
- Embedding models : FastText, ELMo

FastText (2016)

- word2vec과 유사. But 원래 단어를 부분 단어(subword)로 표현한다는 차이가 존재
- word2vec은 단어를 쪼갤 수 없는 것으로 생각. 그러나 FastText는 하나의 단어도 부분 단어로 쪼갤 수 있다고 생각.
- 학습하지 않은 단어에 대해서 유사한 단어를 찾아낼 수 있음

ELMo (2018)

- 단어의 의미는 전에 오는지, 후에 오는지에 따라서도 변화함
- 전체 문장을 한 번 훑고, 문장 전체를 고려한 word embedding을 진행
- 같은 단어라도 문맥에 따라 뜻이 달라짐 (동적)
- 다의어로 인한 어려움을 해결할 수 있음



Unit 04 | Evaluation

- Word embedding Evaluation

평가 분류	정의	내용
내적 (intrinsic) 평가	평가를 위한 데이터(subtask)에 적용하여 성능을 평가	- 단어 간의 유사성(similarity)를 측정 - 계산 속도가 빠름 - 해당 시스템을 이해하기 좋음 - 현실에서 해당 시스템이 유용한지 알 수 없음
외적 (extrinsic) 평가	실제 현실 문제 (real task)에 직접 적용하여 성능을 평가	- 각종 자연어처리 system에 embedding을 직접 사용하여 시스템의 성능을 측정 - 계산 속도가 느림 - 해당 시스템이 문제인지, 다른 시스템이 문제인지, 아니면 둘의 교호작용이 문제인지 알 수 없음

Unit 04 | Model Evaluation

- Extrinsic Evaluation
  - 실제 현실 문제 (real task)에 직접 적용하여 성능을 평가
  - Glove는 좋은 성능을 보임

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**

Unit 04 | Model Evaluation

- Intrinsic Evaluation : word analogy

  Word analogy 유추

  - a:b :: c:? → ?에 들어갈 단어를 유추하는 문제
  - ex. man:woman :: king:?

$$d = \underset{i}{argmax} \frac{(w_b - w_a + w_c)^T w_i}{\|w_b - w_a + w_c\|}$$ 를 만족하는 d를 탐색

Unit 04 | Evaluation

- Intrinsic Evaluation : word analogy

[Semantic examples : city-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

[Syntactic examples : gram4-supelative]

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

하나의 도시가 여러 개의 이름을 갖는 경우 → hard

Unit 04 | Evaluation

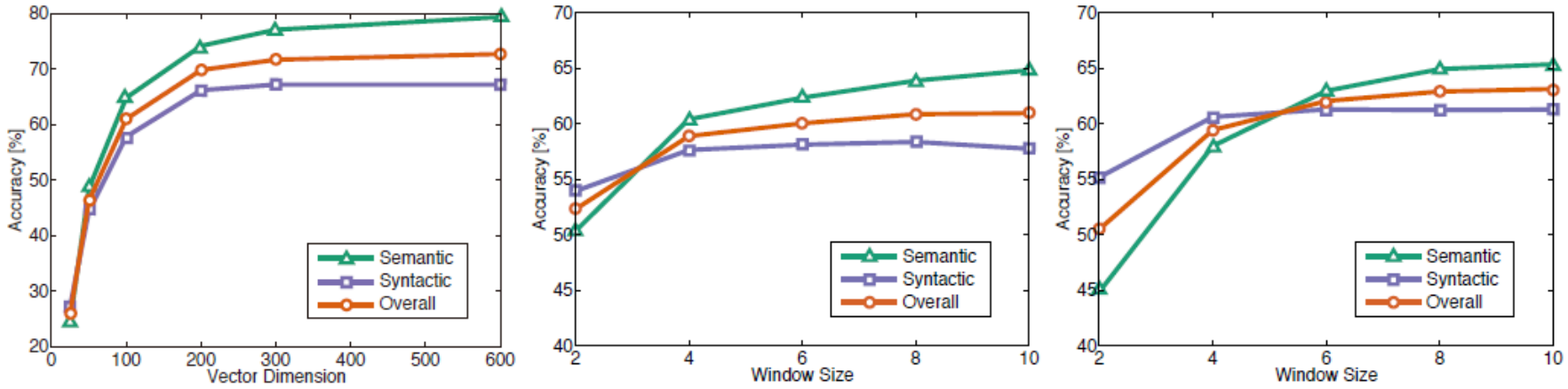
- Intrinsic Evaluation : word analogy

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**

Dimension, corpus size 등을 다르게 하면서 여러 임베딩 모델에 대해서 analogy 분석을 진행해본 결과, GloVe는 좋은 성능을 보였다

Unit 04 | Evaluation

- Intrinsic Evaluation : word analogy



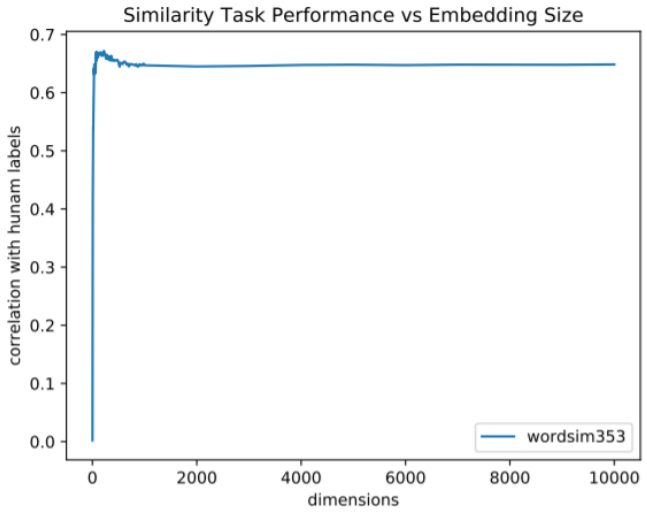
(a) Symmetric context (b) Symmetric context (c) Asymmetric context

- symmetric : window가 양방향 ; asymmetric : window가 중심단어 기준 왼쪽에
- (a) : window size가 10개일 때, dimension은 300으로 두는 게 accuracy가 best
- 맥락 단어를 중심 단어기준 양방향에서 찾았을 때 성능이 더 좋음 (왼쪽에서만 찾으면 성능이 별로 좋지않음)
- (b) (c) : dimension이 100일 때, window size는 8개가 best

Unit 04 | Evaluation

- Intrinsic Evaluation : word analogy

[over-parametrization]



(b) WordSim353 Test

GloVe : dimension이 크게 증가해도 모델 성능이 크게 떨어지지 않음 → GloVe의 장점 중 하나

Unit 04 | Evaluation

- Intrinsic Evaluation : word analogy

  [bias-variance trade-off in dimensionality]

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

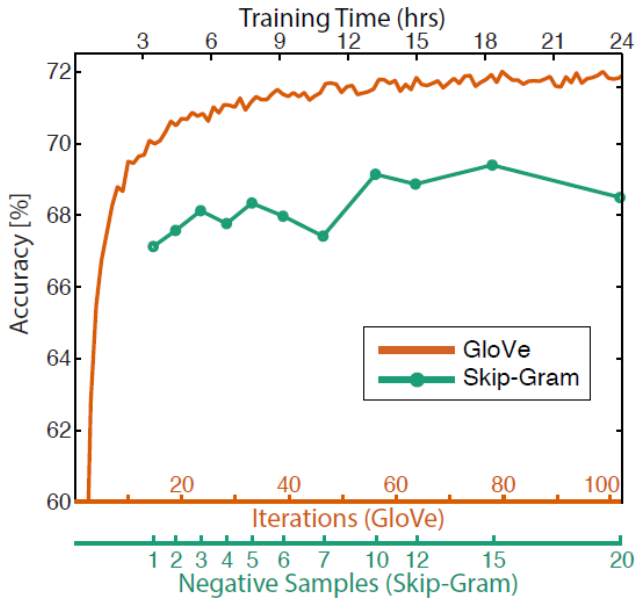
Surrogate Matrix	arg min	+5% interval	+10% interval	+20% interval	+50% interval	WS353	MT771	Analogy
GloVe (log-count)	719	[290,1286]	[160,1663]	[55,2426]	[5,2426]	220	860	560

- PIP Loss : 단어 임베딩 사이의 dissimilarity를 나타내는 지표
- p% sub-optimal interval : dimension이 k이고, k가 interval 안에 존재하면, k차원 임베딩의 PIP loss는 최적 임베딩에 비해 p% 커지는 것을 의미

Unit 04 | Evaluation

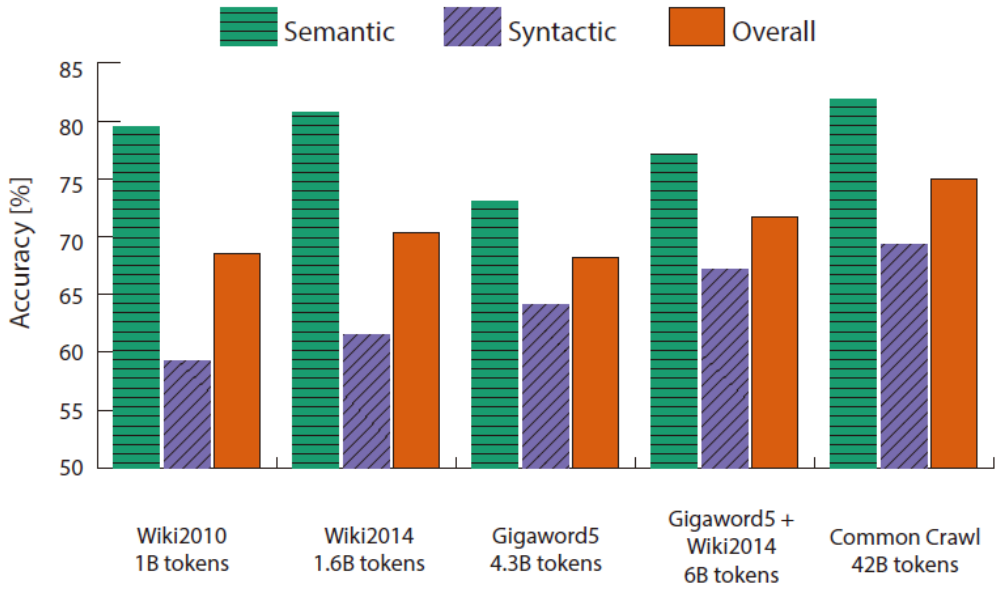
- Intrinsic Evaluation : word analogy

[Training time]



훈련을 오래시킬수록 성능 향상

[Informations in Text]



Gigawords : article
Wiki : dictionary?

텍스트 내에 정보가 많을수록 성능 향상
Wiki data를 활용했을 때 성능이 좋음

Unit 04 | Evaluation

- Intrinsic Evaluation : Correlation
  - 일련의 단어 쌍을 미리 구성한 후에 사람이 평가한 점수와, 단어 벡터 간 코사인 유사도 사이의, 상관관계를 계산해 단어 임베딩의 품질을 평가

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

10점 만점에 10점

5 similarity datasets

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

GloVe : good performance!

Unit 04 | Evaluation

- Word senses and word sense ambiguity
  - 다의어는 어떻게? (like pipe…)

pike

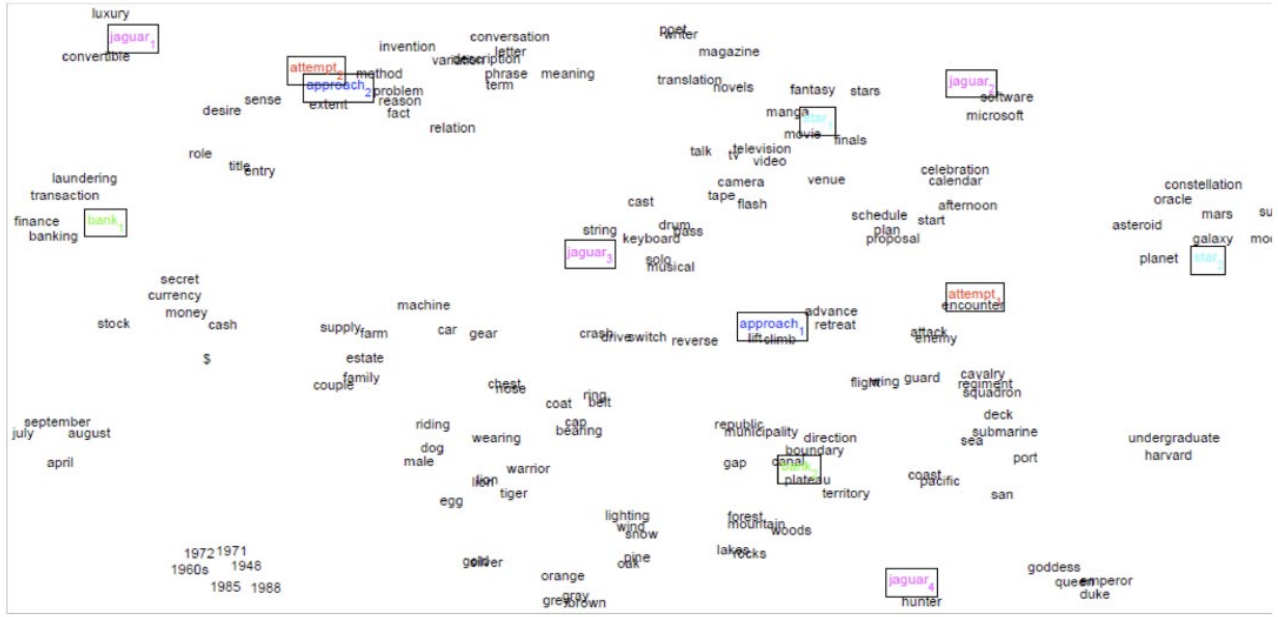
- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something: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

  - Sol1 : Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)
  - Sol2 : Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

Unit 04 | Evaluation

- Word senses and word sense ambiguity

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



특정 단어의 윈도우들을 클러스터링

ex. bank1, bank2, bank3 …

like 다의어…

단어들을 bank1, bank2, bank3 를 중심으로 다시 임베딩~

Unit 04 | Evaluation

- Word senses and word sense ambiguity

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

- 각 의미에 가중치를 부여하여 선형결합 → 새로운 단어 벡터 생성

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}, \quad \alpha_1 = \frac{f_1}{f_1+f_2+f_3}, \text{ etc., for frequency } f$$

- 이 단어벡터를 가지고 유사 단어들 끼리 묶어봤을 때 상당히 결과가 좋았다고 합니다~ (내적인 의미까지 잘 파악해서 분류!)

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

추가질문

1. 언어별 word2vec 방식이 어떻게 차이나는지, 어순도 다른데 그 점을 어떻게 고려하는지
→ word2vec은 단어 임베딩 모델이기 때문에 단어 하나씩을 끊어서 처리한다고 보시면 되는데요! 그렇기에 사실 어순이 그렇게 중요하지는 않죠…! (물론 단어 위치에 따라서 단어의 의미가 조금씩 달라지는 경우가 있긴 한데, 이 때는 word2vec이 아닌 문장의 문맥을 반영해줄 수 있는 ELMo 모델을 사용하시면 될 것 같구요!)

2. Count based와 direct prediction 성능차이 예시가 있는지
→ 찾아본 결과 count-based model과 direct prediction model의 장단점이 명확한 편이기 때문에 , 같은 데이터를 사용해서 두 모델의 성능을 직접적으로 비교한 예시는 잘 없는 것 같아요~

3. Glove의 속도가 빠른 이유
→ word2vec은 input으로 맥락단어나 중심단어의 원핫벡터를 대입한 후, 각 중심단어의 윈도우크기에 비례하게 모델 내에서 계산을 처리합니다. 그러나 glove의 경우 이미 계산된 co-occurrence matrix를 모델의 input값으로 넣고 처리하기 때문에 모델 구조상 glove의 처리속도가 더 빠를 수 밖에 없는 것으로 개인적으로는 생각합니다~ (논문에서는 “테스트를 해보니 glove의 처리속도 및 성능이 더 좋았다” 라고 결과론적으로 서술되어 있어서 제가 개인적으로 생각해봤습니다 ㅎㅎ)

참고문헌

https://eda-ai-lab.tistory.com/122?category=706160
https://yjjo.tistory.com/11
https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/06/pcasvdlsa/
https://bskyvision.com/59
https://darkpgmr.tistory.com/106
https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/09/glove/
https://medium.com/@omicro03/%EC%9E%90%EC%97%B0%EC%96%B4%EC%B2%98%EB%A6%AC-nlp-16%EC%9D%BC%EC%B0%A8-word-embedding-%EC%A0%95%EB%A6%AC-e65c0f4bfe2f

Efficient Estimation of Word Representations in Vector Space (2013)
Distributed Representations of Words and Phrases and their Compositionality (2013)
Glove: Global vectors for word representation (2014)
On the Dimensionality of Word Embedding (2018)
…

들어주셔서 감사합니다.