

## Natural Language Processing with DeepLearning

### week 2

#### Review - Stochastic Gradient Descent)

: 전체 corpus에 대해 gradient를 계산하는 대신 하나의 단어 또는 작은 batch에 대해 gradient를 계산

=> 학습이 Gradient Descent보다 빠름 대부분 NN 학습에서 SGD 사용

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

word2vec의 입출력 형태: One-Hot Encoding

=> SGD에 사용하면 각 윈도우에 해당하는 단어들을 기반으로 업데이트되므로 나머지 수 많은 단어들은 업데이트되지 않음

ex) 윈도우 사이즈가 20이면 5개 단어에 대한 gradient 정보를 갖지만 나머지 단어에 대한 gradient 정보 없음 => sparse한 gradient update를 하게 되어 불필요한 계산이 이루어짐.

#### word2vec: More details

- 1) Skip Grams + Negative sampling (additional efficiency in training)
  - center word로 context word 예측
- 2) Continuous Bag of Words(CBOW)
  - context word로 center word 예측

#### 1. SGNS

일반적으로는 naive softmax 사용, 그러나 이러한 방법은 분모에서 모든 단어에 대해 내적 계산을 해야 하므로 computationally expensive함

ex)

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

**Main Idea** : true pair(center word, context word)와 noise pairs(center word, negative word) 이진분류하는 binary logistic regressions 학습

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

<Skip gram 목적함수>

이 목적함수를 maximize하여 최적화하는데, softmax 대신 logistic/sigmoid function 사용함

=> 목적함수 최대화 하려면 window 안에 있는 outside word와 center word의 내적값은 커야함

- + center word와 negative word와 negative word의 내적값은 음수가 나오므로 음수일 경우 sigmoid 함수 통과하면 값이 작아져서 sigmoid 함수의 대칭성을 이용해 내적값에 음수를 취해줌  
(\* negative word: random sampling으로 추출한 window에 해당하지 않는 단어)

<negative likelihood (목적함수 최대화)>

$$J_{\text{neg-sample}}(\mathbf{u}_o, \mathbf{v}_c, U) = -\log \sigma(\mathbf{u}_o^T \mathbf{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(-\mathbf{u}_k^T \mathbf{v}_c)$$

=> positive word와 center word의 내적값은 크게, negative word와 center word의 내적값은 작게끔 학습한다.

**How to do Negative Sampling?** : 단어 sampling 할 때, 균일한 확률로 하지 않고, a, the와 같이 많이쓰이는 단어들에 대해서는 제한을 주기위한 함수를 사용한다.

### co-occurrence matrix

- window length 1 (일반적으로는 5-10)
- 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

=> 비슷한 의미 가진 단어끼리 비슷한 벡터를 가지는 걸 알 수 있음

(ex) I like, you like, I enjoy, you enjoy 반복 => I와 you는 유사한 벡터 일것이다!

**문제 :**

- 1) 성능이 나쁘진 않지만 sparsity 문제 있음
- 2) 차원이 매우 커지므로 희소성이 커짐
- 3) noisy하고 less robust해지는 경향이 있음

=> 고정된 차원수로 단어를 표현하여 저차원 벡터를 사용하는 것이 더 나은 결과를 보임 (일반적으로 25-100의 차원수 사용)

따라서 co-occurrence matrix의 차원을 SVD, LSA, COALS등 선형대수적 접근인 특이값 분해를 통해 축소함

## Glove

: 각각 장단점을 가진 선형대수 기반의 co-occurrence matrix와 neural updating 알고리즘을 connect하기 위해 개발

<ul style="list-style-type: none"> <li>• LSA, HAL (Lund &amp; Burgess),</li> <li>• COALS, Hellinger-PCA (Rohde et al, Lebret &amp; Collobert)</li> </ul> <ul style="list-style-type: none"> <li>• Fast training</li> <li>• Efficient usage of statistics</li> </ul> <p><b>Primarily used to capture word similarity</b></p> <p><b>Disproportionate importance given to large counts</b></p>	<ul style="list-style-type: none"> <li>• Skip-gram/CBOW (Mikolov et al)</li> <li>• NNLM, HLBL, RNN (Bengio et al; Collobert &amp; Weston; Huang et al; Mnih &amp; Hinton)</li> </ul> <ul style="list-style-type: none"> <li>• Scales with corpus size</li> <li>• Inefficient usage of statistics</li> </ul> <p>Generate improved performance on other tasks</p> <p>Can capture complex patterns beyond word similarity</p>
--	---

### co-occurrence matrix (count-based)

장점

- 빠른 훈련 속도

파악

- 효율적으로 통계정보 사용

단점

- 주로 단어 유사성 여부만을 파악하여

미침

단어 간 관계는 파악할 수 없음

사용 못함

- 빈도수가 클수록 과도한 중요성을 부여하여 불균형 함

### neural updating algorithm

장점

- 단어 유사성 이상의 복잡한 패턴을

- 높은 수준의 성능

단점

- corpus의 크기가 성능에 영향을

- 비교적 효율적으로 통계정보

\*\* Crucial insight: vector의 뼈심이나 덧셈을 통한 유추를 가능하게 하려면 해당 단어의 component를 알아야 하는데, 이를 알아내기 위해 통계정보인 co-occurrence probabilities의 비율 활용 가능

(\* 유추: 얼음-고체=물)

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

: ice이라는 단어에 대해 solid과 water라는 단어는 유사성이 높음

steam이라는 단어에 대해서는 solid과의 유사성이 낮고 gas와 water라는 단어와 유사성이 높음

ice와 steam 간의 관계를 알아내기 위해서 ice에 대한 단어의 확률과 steam에 대한 단어의 확률의 비율을 사용!!

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

: 수치가 위와 같이 주어진다면, ice와 solid의 유사도에 steam과 solid의 유사도를 나누었을 때 8.9라는 높은 수치가 도출됨

그러나, water나 fashion과 같이 두 단어에 대해 유사성의 정도가 비슷한 경우에는 1에 가까운 수치가 나오므로 이러한 co-occurrence 확률의 비율을 통해 단어의 components를 알아내!!

### Encoding meaning in vector differences

: 동시 출현 발생 확률의 비율을 meaning component로 어떻게 활용하는 것인지 확인

#### Log-bilinear model

: 두 단어 간의 내적을 co-occurrence probabilities의 로그에 근사하도록 함

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

#### Log-bilinear model with vector differences

: a와 b 단어 벡터 간의 차이와  $w_x$ (x단어)와의 유사도가 위에서 보았던 동시 출현 확률의 비율의 로그값에 대응하도록 함

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

=> GloVe 모델의 목적함수는?

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

: 위에서 설명한 것처럼 두 단어의 내적과 동시 출현 확률의 비율에 로그를 취한 값이 유사하도록,

두 단어의 dot product와 co-occurrence 확률의 로그 간 오차제곱을 최소화함  
앞의 f 함수: 단어 빈도수에 대해 scaling

### Evaluate word vectors

- intrinsic evaluation : 평가를 위한 substack(데이터)에 모델을 적용하여 성능을 평가  
-> 계산 속도가 빠르지만 현실에서 유용하게 쓰일 수 있을지 알 수 없음

- extrinsic evaluation: 실제 현실 문제에 직접 적용하여 성능을 평가  
-> 각종 NLP task에 embedding 결과를 직접 적용하여 성능을 측정 가능, 하지만 성능의 결과가 임베딩 모델 때문인지 다른 요소 때문인지 알 수 없으며 계산 속도가 느림

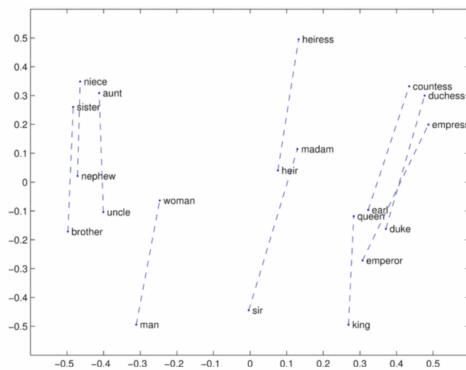
## Intrinsic word vector evaluation

\*\* Word Vector Analogies (단어 유추) : ?에 해당하는 단어 유추하는 것을 의미

$$\begin{array}{c} \boxed{a:b :: c: ?} \\ \text{man:woman :: king: ?} \end{array} \longrightarrow \boxed{d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}}$$

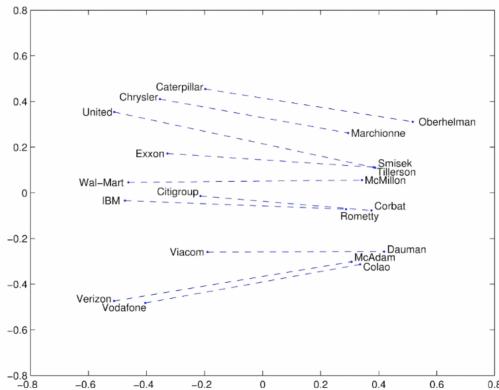
### ex1) GloVe Visualizations

: 유사한 단어간에 강한 선형 관계를 보이므로 단어를 빼고 더함으로써 단어 유추 가능



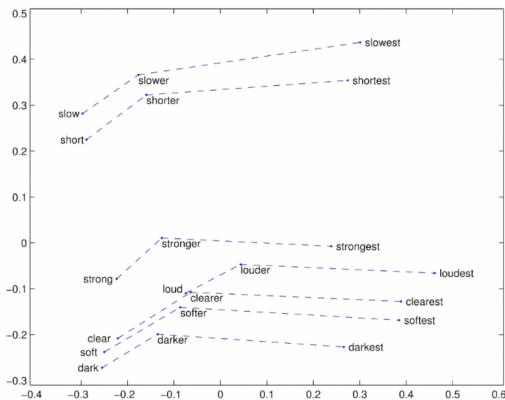
### ex2) GloVe Visualizations: Company- CEO

: 회사와 CEO에 대한 데이터를 학습한 결과 회사와 CEO의 관계도 선형적이기 때문에 단어의 유추가 가능



### ex3) GloVe Visualizations: Comparatives and Superlatives

: 원형-비교급-최상급에도 대략 선형 component로 구성된 것을 확인할 수 있고, 의미론적, 구문론적 외에도 문법에 대해서도 학습함을 알 수 있음



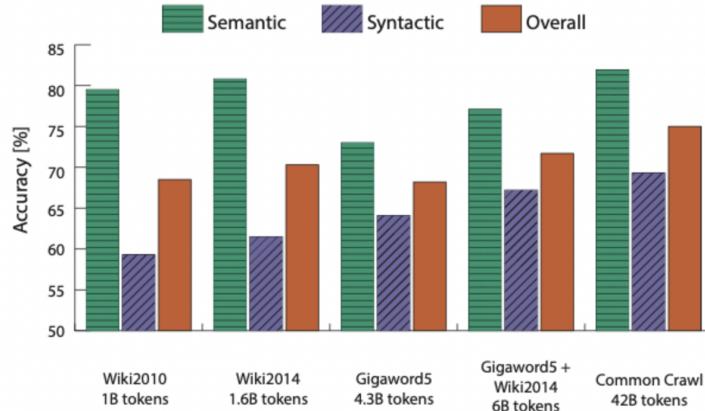
-Analogy evaluation

Model	Dim.	Size	Sem.	Syn.	Tot.
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 <sup>†</sup>	300	6B	63.6	67.4	65.7
SG <sup>†</sup>	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>

: 유추에 대한 semantics(의미적) / syntaxics(문법적) / Total(전체) 평가?

GloVe > word2vec > co-occurrence matrix 순으로 성능이 좋음

하지만 GloVe가 가장 높게 나온 이유는 평가에 더 유리한 데이터를 학습했기 때문이라 함

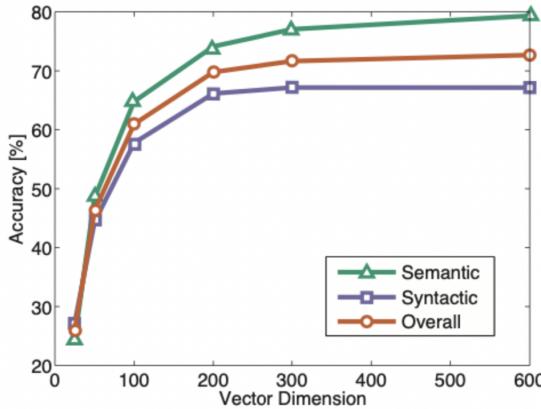


- GloVe는 일부 Wikipedia 데이터에 대해 학습했지만 word2vec은 뉴스 데이터들을 위주로 학습했기 때문에 의미론적으로 성능이 좋을 수 밖에 없음

- Wikipedia는 백과사전 데이터의 특성상 의미론적인 데이터를 많이 포함하고 있기 때문에 wikipedia 데이터를 학습한 경우 semantic적으로 성능이 높아짐

- article 데이터인 Gigaword를 학습했을 때의 의미론적 성능은 wikipedia를 학습했을 때에 비해 1/4 정도 떨어지는 것을 보아 wikipedia를 학습하면 의미론적 성능이 높아진다는 것을 증명

\*\* 하지만 어떤 데이터든 무조건 많은 데이터를 학습했을 때 가장 성능이 좋다 !!



- + 벡터의 차원이 300차원 정도일 때 가장 성능이 좋았음 (GloVe 뿐 아니라 Word2Vec의 임베딩 모델에 적용)

### Another intrinsic word vector evaluation

: 모델의 단어 간 유사성에 대한 판단과 인간의 판단을 비교하는 intrinsic evaluation

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

지난 수십년 동안 단어의 유사성에 대해 인간이 0-10 사이로 점수를 매겨 수집한 데이터를 활용해 모델의 성능을 평가함

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 <sup>†</sup>	6B	57.2	65.6	68.2	57.0	32.5
SG <sup>†</sup>	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

: 인간의 판단과 word vector 거리의 상관관계를 통해 평가한 결과,  
GloVe > Word2Vec > co-occurrence matrix 순으로 성능이 좋다!!

### Extrinsic word vector evaluation

: 좋은 vector를 사용하면 NLP task의 성능이 좋아지므로 GloVe를 통해 구한 Embedding vector가 NLP task에 얼마나 도움이 되는지를 평가함

- Named Entity Recognition(개체명 인식)에 대한 성능 평가

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	<b>88.7</b>	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	<b>93.2</b>	88.3	<b>82.9</b>	<b>82.2</b>

: GloVe > word2vec > co-occurrence matrix 순으로 성능이 좋음!!