

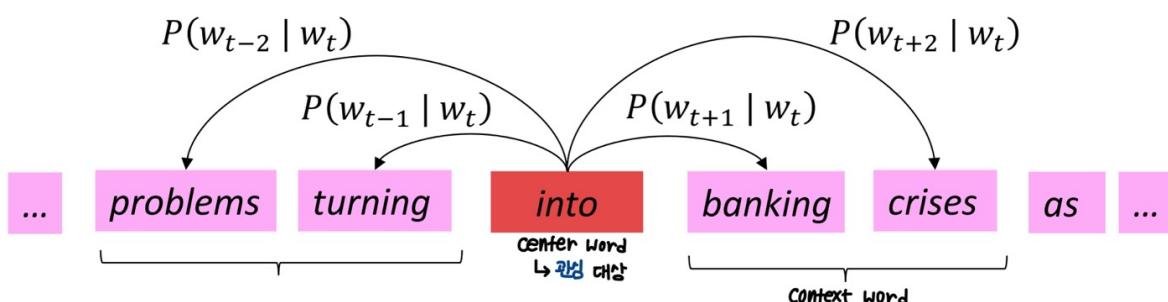


2. Word Vectors, Word Senses, and Neural Classifiers

1. Word2Vec

- 텍스트 모델의 핵심은 텍스트를 컴퓨터가 인식할 수 있도록 숫자 형태의 벡터 또는 행렬로 변환하는 것임
- 두 가지 벡터화 방식
 - One Hot Encoding(= sparse representation)
 - 장점: 단어 벡터가 0 또는 1의 값으로 구성되어 과정이 단순함
 - 단점: 텍스트 분석에서 중요한 부분 중 하나인 단어 간의 유사성을 반영하지 못함
 - Word Embedding(= dense representation)
 - One-hot Encoding의 단점 보완
 - 현대의 대부분의 자연어 처리 기법들은 (ex. Word2vec, GloVe, FastText...) Word Embedding 방식을 기반으로 발전

Word2Vec의 전체 과정

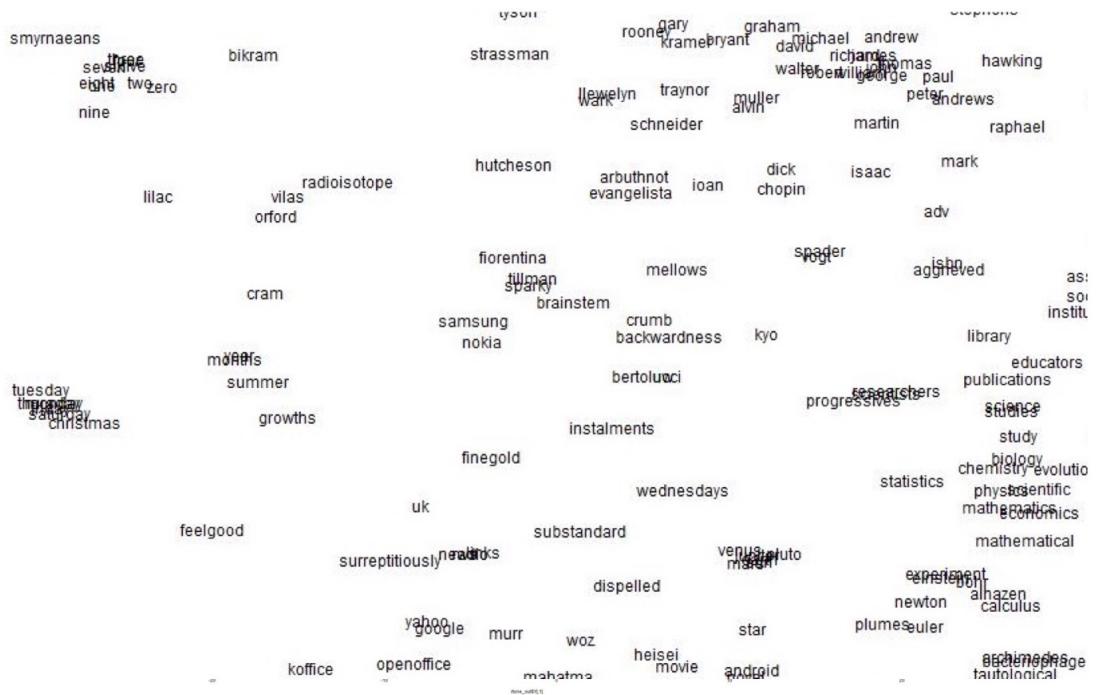


- 주요 feature
 - 중심 단어(center word)
 - 맥락 단어(context word)
 - 윈도우(window)

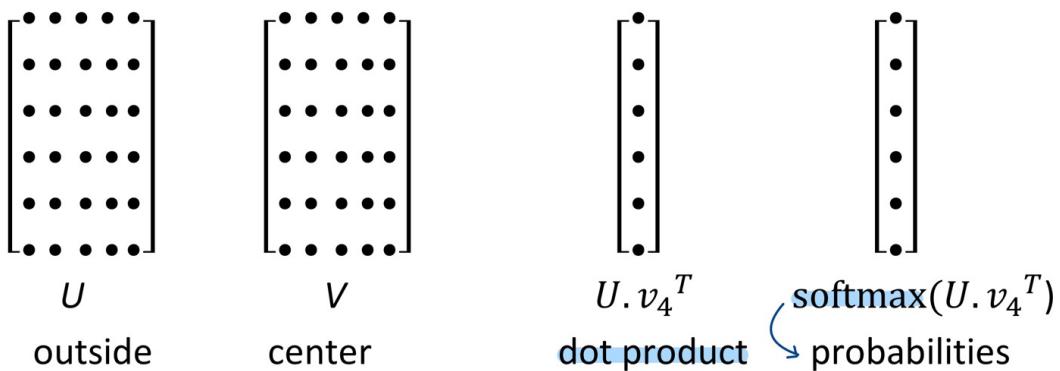
- Idea

 - “의미적으로 유사성을 가진 단어들은 서로 가까운 위치에 존재한다.”

⇒ 목적 함수(objective function)를 최대화



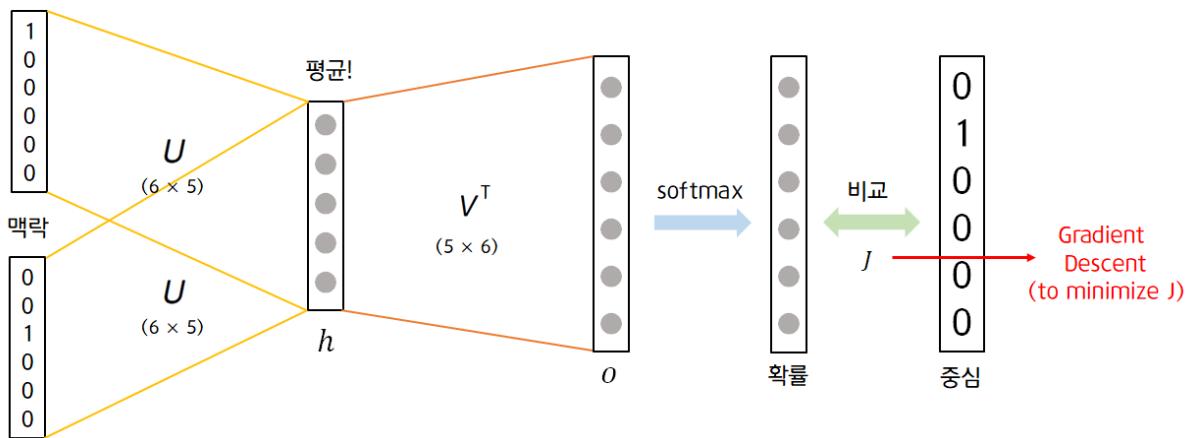
 - 중심 단어를 기준으로 양쪽으로 윈도우 크기만큼의 단어를 맥락 단어로 설정 → 원핫 인코딩을 통해 중심 단어 벡터와 맥락 단어 벡터를 생성 → 입력 벡터/출력 벡터로 사용



 - 맥락 벡터는 기본적으로 윈도우 개수의 2배만큼 생성됨

- 맥락 벡터 생성

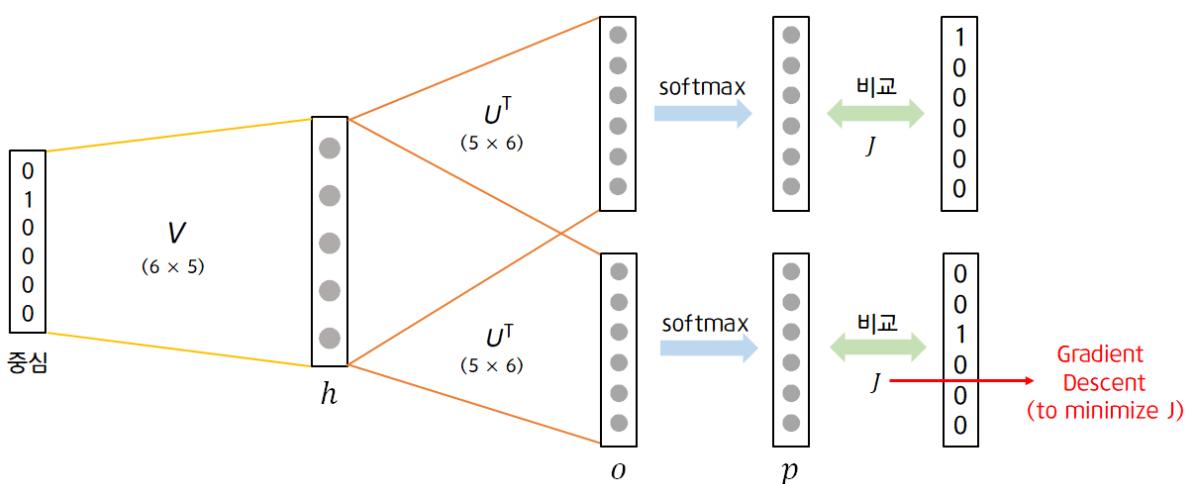
1. CBOW(Continuous Bag of Words)



- 입력 벡터 = 맥락 벡터, 출력 벡터 = 중심 벡터

- 맥락 단어를 통해 중심 단어를 예측

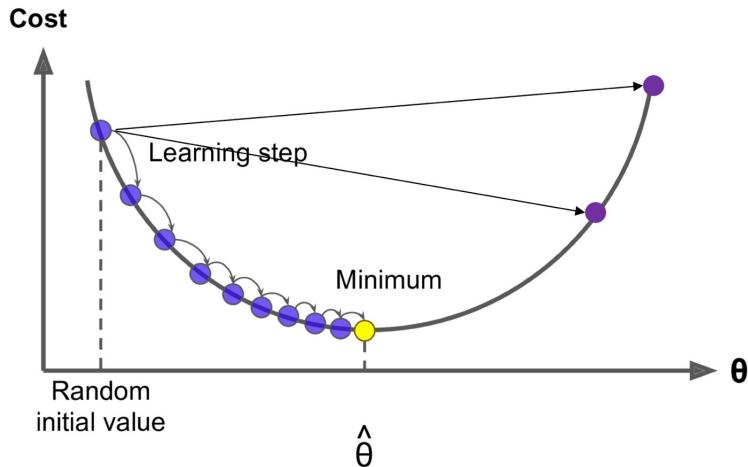
2. SG(Skip-grams)



- 입력 벡터 = 맥락 벡터, 출력 벡터 = 중심 벡터

- 주어진 중심 단어를 통해 맥락 단어를 예측하는 방식

2. 최적화: 경사 하강법



2-1. 경사 하강법

- 비용함수 $J(\theta)$ 를 최소화
 - 경사 하강법은 θ 를 변화시키며 비용함수를 최소화하는 알고리즘
- 아이디어)** 현재 θ 에서의 비용 함수 계산 → 점진적으로 감소 → 음의 기울기 방향으로 이동 → 반복
- 수식

Gradient Descent

- Update equation (in matrix notation):

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

\uparrow

$\alpha = \text{step size or learning rate}$

- 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
  
```

8

2-2. 확률적(stochastic) 경사 하강법

- Gradient Descent는 **전체** 데이터에 대해 계산이 이루어짐
 - 계산량이 너무 많다는 단점
- ⇒ 대신 **Stochastic Gradient Descent**(또는 mini-batch)를 활용

- 하지만 One-Hot Encoding 혹은 Word2vec의 입·출력 벡터로 사용되는 0과 1으로 이루어진 sparse한 vector는 stochastic gradient descent를 사용했을 때 문제가 발생
 - 0에 해당되는 위치에서는 계산이 이루어지더라도 계속해서 0이기 때문에 실제로 gradient가 update되지 않는데, 불필요한 계산이 수반되기 때문

$$\nabla_{\theta} J_t(\theta) = \begin{bmatrix} 0 & \xrightarrow{\text{계산 솔루션}} \text{어짜피 } 0 \text{임} \\ \vdots & \Rightarrow \text{무의미하다!} \\ \nabla v_{like} & \\ \vdots & \\ 0 & \\ \nabla u_I & \\ \vdots & \\ \nabla u_{learning} & \\ \vdots & \end{bmatrix} \in \mathbb{R}^{2dV}$$



그럼 실제로 나타나는 벡터만 업데이트하면 되지 않을까?

⇒

Negative Sampling

2-3. Negative Sampling

- 실제로 등장한 (= 0이 아닌) 행에 대해서만 gradient 계산하고 sparse한 matrix를 add/subtract 함으로써 gradient를 update하는 방식
 - ⇒ 기존의 다중분류를 이진분류로 근사시켜 모델을 효율적으로 만들기

negative sampling의 목적 함수

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

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

c : 중심 벡터

o : 맥락 벡터

k : 노이즈 벡터 (랜덤하게 선택된 벡터, 실제 맥락 벡터가 아님)

u : 중심 벡터와 hidden layer 사이의 가중치

v : 맥락 벡터와 hidden layer 사이의 가중치

⇒ True pair(중심 벡터, 맥락 벡터)의 경우 코사인 유사도가 클수록 확률이 높고, Noise pair(중심 벡터, 맥락 벡터)의 경우 코사인 유사도가 작을수록 확률이 높다고 해석할 수 있음

⇒ True pair는 중심 벡터와 맥락 벡터가 가까이 있을수록(코사인 유사도가 클수록) 손실이 0에 가깝고, Noise pair는 중심 벡터와 노이즈 벡터가 멀리 있을수록(코사인 유사도가 작을수록) 손실이 0에 가까움을 의미

3. Co-occurrence matrix

- Skip-gram은 중심 단어를 기준으로 맥락 단어가 등장할 확률을 계산
 - 원도우 개수를 아무리 크게 늘려도, global co-occurrence statistics(ex> 전체 단어의 동반 출현 빈도 수)와 같은 통계 정보는 내포할 수 없음
 - 벡터의 값들은 중심 단어가 given일 때 각 값의 개별적인 등장 확률을 의미하기 때문

⇒ count-based의 Co-occurrence matrix가 등장

3-1. Window based co-occurrence matrix

- 한 문장을 기준으로 윈도우에 각 단어가 몇 번 등장하는지를 세어 구성
- 대칭(symmetric) 행렬
- 예시

- 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

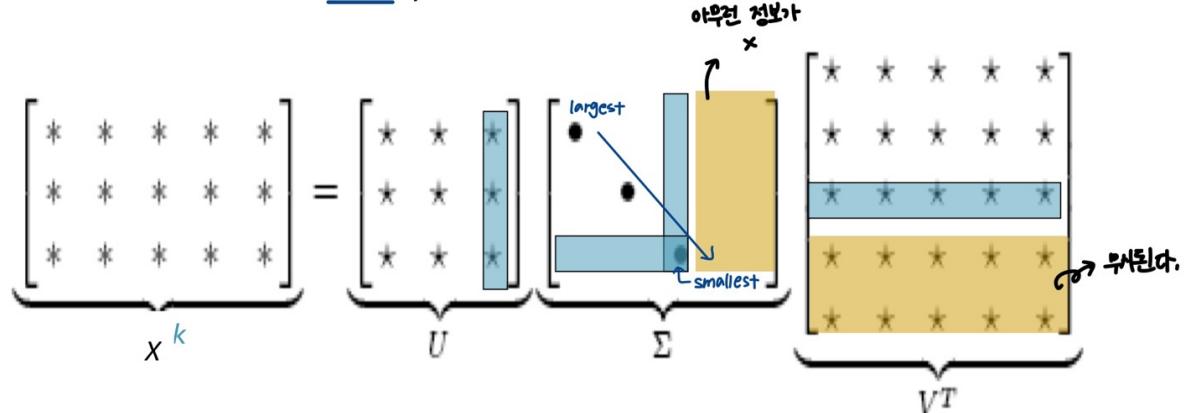
3-2. Word-Document matrix

- 한 문서를 기준으로 각 단어가 몇 번 등장하는지를 세어 구성
 - 문서에 있는 많은 단어들 중 빈번하게 등장하는 특정 단어가 존재한다는 것을 전제
- LSA(잠재적 의미 분석)를 가능하게 하는 기법
 - ex> 문서 간 유사도 측정 등
- 그러나 이와 같은 count-based matrix는 단어의 개수가 증가할수록 차원이 폭발적으로 증가
 - ⇒ SVD 또는 LSA 등을 이용하여 차원을 축소시킨 후 사용
 - ⇒ 대부분의 정보를 작은 차원의 행렬 내에 포함시킴

3-3. SVD (Singular Value Decomposition)

- 다음과 같이 행렬을 분해

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



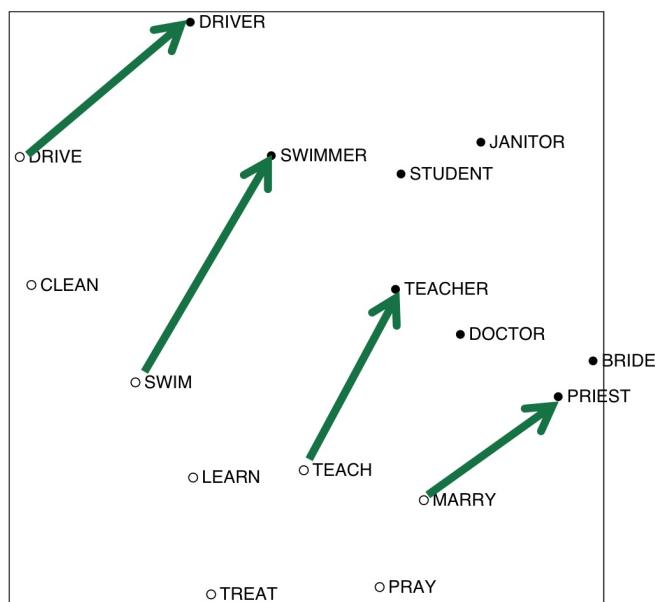
Retain only k singular values, in order to generalize.

\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.

↳ 작은 행렬은 속도

3-4. Scaling the Counts

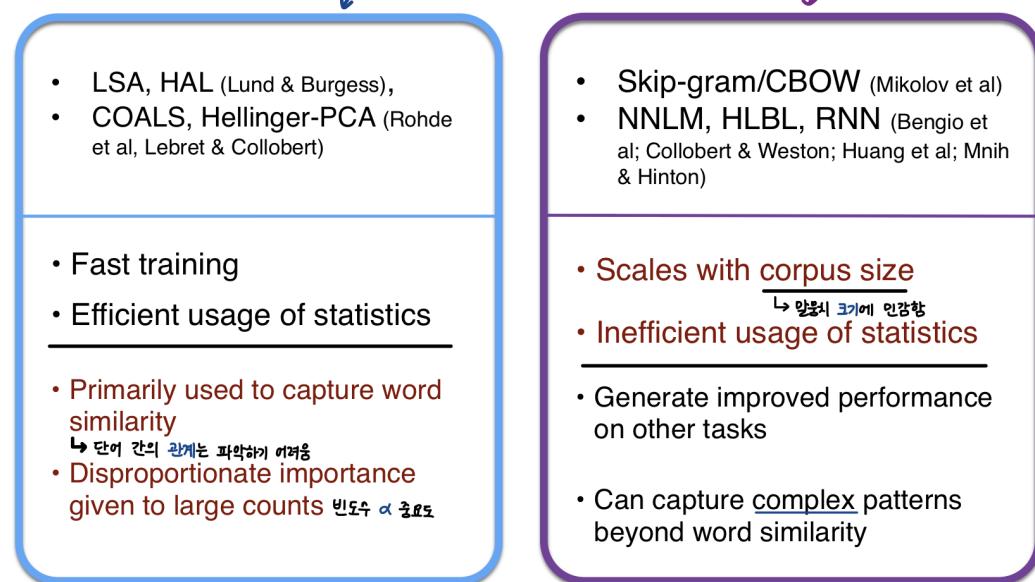


- 각 단어에 대한 카운트를 조정하는 것은 학습에 도움이 됨
- 자주 등장하는 단어들(ex> the, he, has)은 다른 단어들보다 더 큰 임팩트를 가지게 됨
 - 일정 카운트 이상 등장하는 단어들은 모두 무시하는 등의 방법으로 조정
- 같은 윈도우 안의 주변 단어라고 해도 중심 단어와 더 가까운 단어와 덜 가까운 단어 사이의 중요도를 차등으로 주는 방법도 고려
- 단순 집계 대신 피어슨 상관관계를 사용하고, 음수 값은 0으로 설정하는 방법도 존재

4. GloVe (Global Vectors for Word Representation)

4-1. Count Based vs Direct Prediction

5. Towards GloVe: Count based vs. direct prediction



- 각 방법의 뚜렷한 장단점이 존재하였고, 각 기법들의 장점만을 갖춘 새로운 방법이 등장
⇒ **GloVe**

4-2. GloVe

Idea

- 임베딩 된 단어 벡터 간 유사도 측정을 수월하게 하자!
(word2vec의 장점)
- 말뭉치 전체의 통계 정보를 반영하자!
(co-occurrence matrix의 장점)

주요 insight

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

▲ 전체 학습 말뭉치 중 두 단어가 동시에 등장한 빈도를 각각 센 다음 전체 말뭉치의 단어 개수로 나눈 동시 등장 확률을 나타낸 표

- 임베딩 된 두 단어 벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적 함수를 정의

Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

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

목적 함수

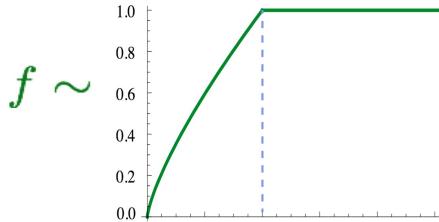
$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^V f\left(\begin{matrix} \text{동시 등장 빈도} \\ \text{가중치 합} \end{matrix}\right) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij} \right)^2$$

- X : 동시 등장 행렬
 - 희소 행렬일 가능성이 높음

- 동시 등장 행렬에서 동시 등장 빈도의 값 X_{ik} 의 값이 굉장히 낮은 경우에는 거의 도움이 되지 않음

$\Rightarrow X_{ik}$ 값에 영향을 받는 가중치 함수 $f(X_{ik})$ 를 목적 함수 J 에 도입



적용 결과

Nearest words to
frog:

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



litoria

leptodactylidae

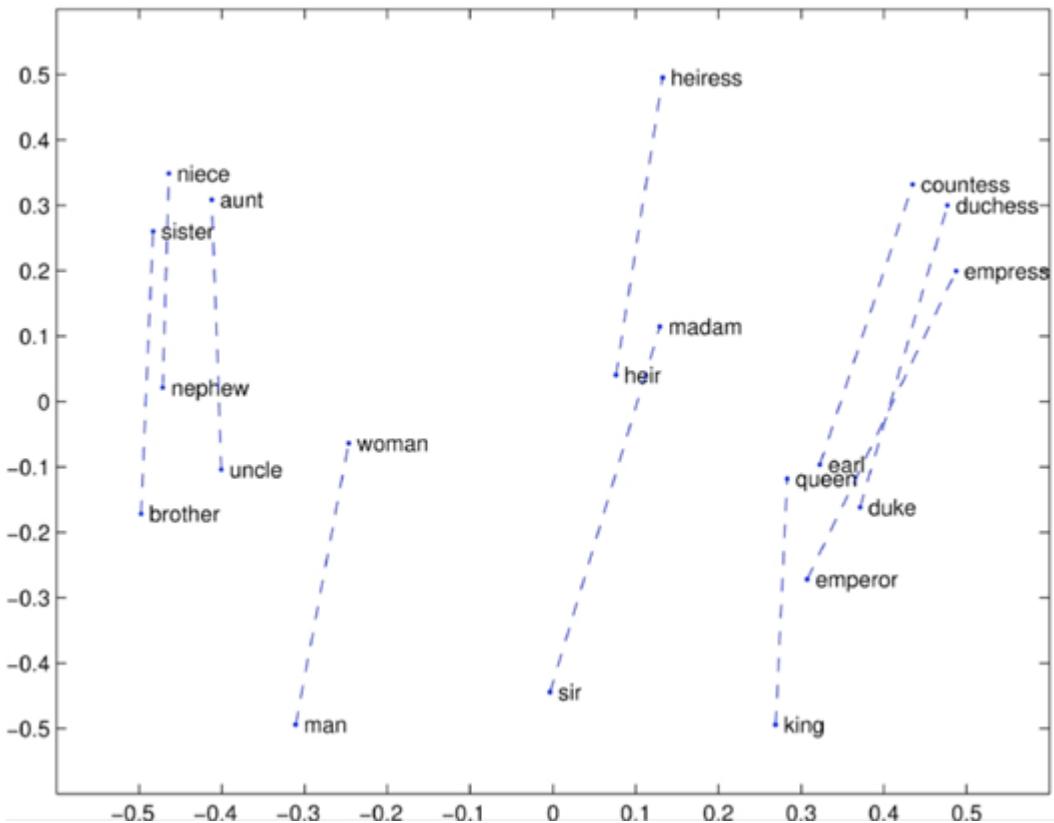


rana



eleutherodactylus

- 유사한 관계에 있는 단어들끼리 비슷한 Linear Property를 갖게 됨



5. Word embedding Evaluation

5-1. NLP의 일반적인 평가 방식

1. 내적(intrinsic) 평가

- 평가를 위한 데이터에 적용하기 위한 성능 평가
- word analogy(유사)
 - a:b :: c:? 에서 ?에 들어갈 단어를 유추하는 문제

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

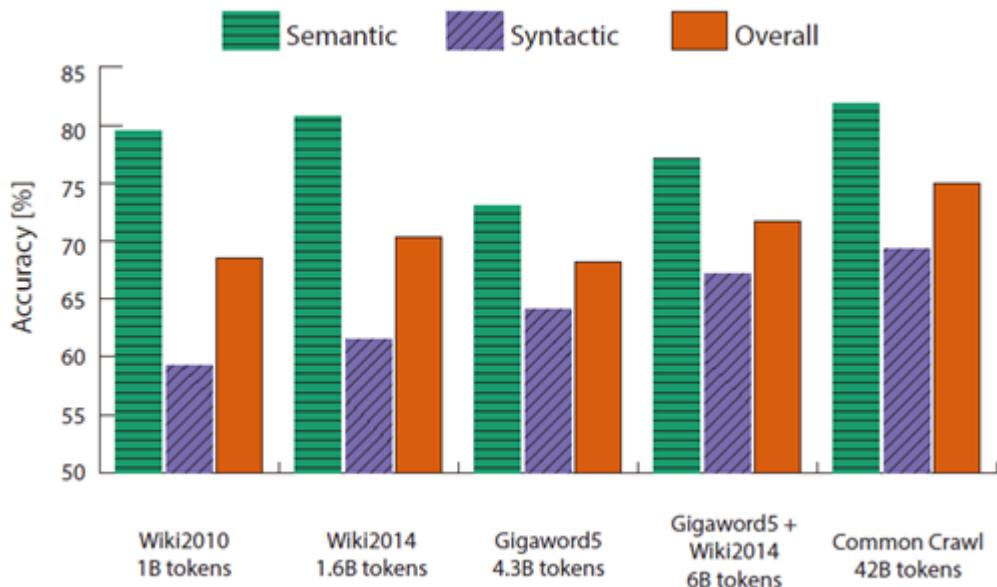
d를 찾는 문제

-
- 단어 간의 유사성을 측정
 - 빠른 계산 속도
 - 해당 시스템에 대한 이해가 용이

- 현실에서 해당 시스템이 유용할 지는 알 수 없음

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 [†]	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

GloVe는 내적 평가에서 좋은 성능을 보이고 있음



텍스트 내에 정보가 많은 Wiki data를 활용했을 때 성능이 좋음

-
- 일련의 단어 쌍을 미리 구성한 후에 사람이 평가한 점수와 단어 벡터 간 코사인 유사도 사이의 상관관계를 계산해 단어 임베딩의 품질을 평가하는 방식도 활용할 수 있음

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

유사도 높음

유사도 낮음

- Word vector distances and their correlation with human judgments 주제 비슷한 데이터

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

good!

2. 외적(extrinsic) 평가

- 실제 문제에 직접 적용하기 위한 성능 평가
- 각종 자연어 처리 system에서 임베딩을 직접 사용하여 시스템의 성능을 측정
- 느린 계산 속도
- 해당 시스템 자체의 문제인지 다른 시스템의 문제인지 또는 둘의 상호작용이 문제인지를 점검하기 어려움

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

GloVe는 외적 평가에서 좋은 성능을 보이고 있음

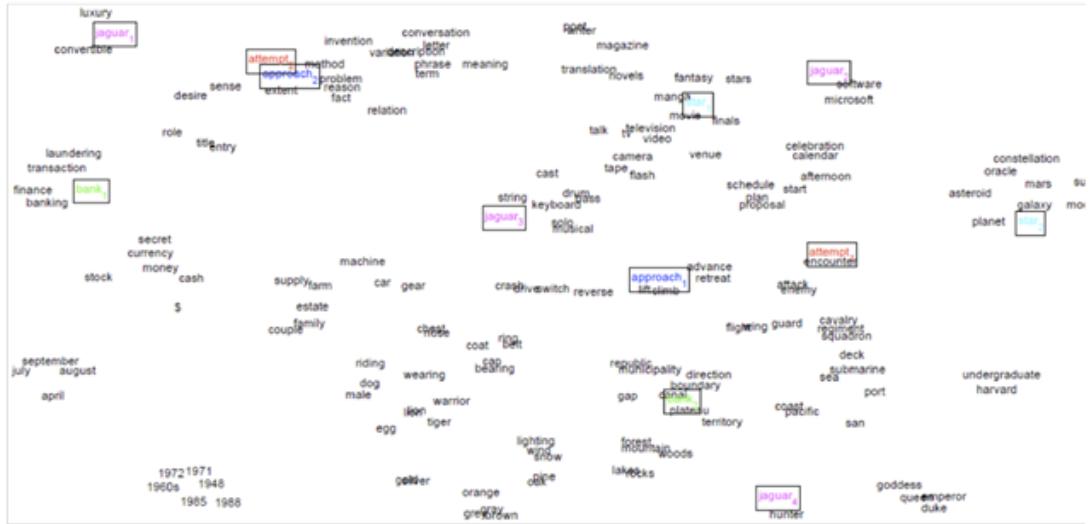
6. 어감과 어감 모호성

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!*
- 하나의 단어가 여러 의미를 가지는 경우(= 다의어)
 - 이러한 경우엔 NLP에선 어떻게 해결할까?
- 해결책

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

- 특정 단어의 윈도우들을 클러스터링(ex. bank1, bank2, bank3 ...) 한 후, 단어들을 bank1, bank2, bank3 를 중심으로 다시 임베딩



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

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

- 각 의미에 가중치를 부여하고 선형 결합을 통해 새로운 단어 벡터를 생성
- 해당 단어 벡터를 가지고 유사 단어들끼리 clustering 했을 때 상당히 결과가 좋았음
 - 내적인 의미까지 잘 파악해서 분류했음을 의미