NLP : CS224N (2주차)

<Review>

지난 시간에는 Word2Vec 모델에 대해 배웠습니다.

Word2Vec 모델은 task에 따라 두 가지 알고리즘으로 작동됩니다.

> 💡 1) 주변에 있는 단어들을 통해, 중간에 있는 단어 예측 → CBOW
> 2) 중간에 있는 단어를 통해, 주변에 있는 단어들을 예측 → Skip-gram

두 알고리즘은 서로 다른 task를 수행할 뿐 작동방식은 같습니다.

> 💡 1) One Hot vector 형태의 입력값을 입력받아
> 2) weight_input과 곱해서 hidden layer를 만들고
> 3) hidden layer 값을 weight_output과 곱해서 score를 만든 뒤
> 4) score에 softmax를 취해서 “각 단어가 나올 확률”로 이해
> 5) 추정된 단어와 정답 단어의 Loss를 통해 backpropagation 방식으로 weight를 업데이트
> 6) 1)~5) 과정을 문장의 모든 단어에 대해서 수행

<Gradient Descent>

$$\theta\, new = \theta - \alpha \nabla_\theta J(\theta)$$

Loss를 최소화하는 Parameter(세타) 를 찾기 위해 Gradient Descent를 사용합니다.

하지만 Gradient Descent는 전체 데이터에 대해 계산이 이루어지기 때문에 계산량이 너무 많다는 단점이 있습니다.

그래서 Gradient Descent 대신 **Stochastic Gradient Descent** 를 사용한다고 강의에서는 말하고 있습니다

하지만 One-hot encoding 벡터는 sparse하기 때문에, SGD로 1개(혹은 몇 개)를 골라 업데이트 하려고 해도

0에 해당하는 위치에서는 계산 결과도 0이 되기 때문에 실제로 theta가 업데이트되지 않는 문제가 발생합니다.

$$\nabla_\theta J(\theta) = \begin{bmatrix} 0 \\ \dots \\ \nabla_{v_{sat}} \\ 0 \\ \dots \\ \nabla_{u_{mat}} \end{bmatrix}$$

그래서 사용하는 방법이 Negative sampling인데요.

강의에서는 Skip-gram model with Negative sampling을 소개하고 있습니다.

<SGNS : Skip-gram with Negative sampling>

첫째, 왜 사용할까요?

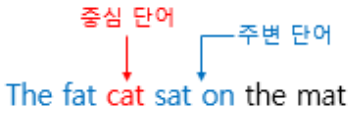
Word2Vec의 출력층에서는 소프트맥스 함수를 거쳐온 벡터(벡터의 크기 = 단어집합의 크기)와 실제값인 One-hot vector의 오차를 구하고 이로부터 임베딩 테이블에 있는 모든 단어에 대한 임베딩 벡터 값을 업데이트합니다.

따라서 단어 개수가 많아질수록 계산 복잡도 역시 높아지고, 이는 모델 학습 속도 저하를 유발합니다. 전체 단어 집합이 아니라 일부 단어 집합에만 집중하기 위해 사용하는 방법이 Negative sampling이다 이해하시면 됩니다.

둘째, 그러면 Negative sampling이 뭘까요?

한 줄로 설명하면 “softmax 확률값 계산시 전체 단어 대상으로 구하지 않고 몇몇 단어만으로 계산”하는 것입니다.

이런 문장이 있다고 하겠습니다.



Skip-gram은 중심 단어로부터 주변 단어를 예측하는 모델이었습니다. 입력은 중심 단어, 모델의 예측은 주변 단어인 구조입니다.

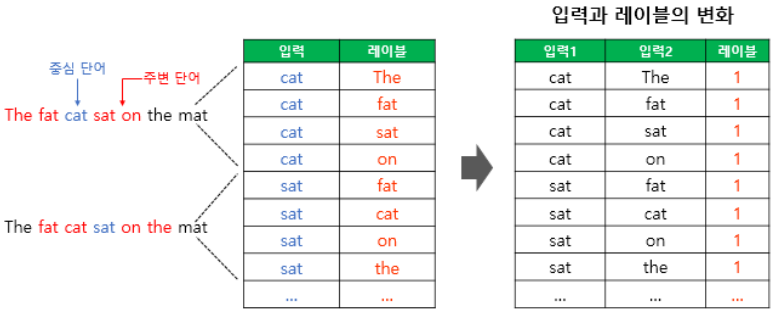
SGNS는 다음과 같이 중심 단어와 주변 단어가 모두 입력이 되고, 이 두 단어가 실제로 윈도우 크기 내에 존재하는 이웃 관계인지 그 확률을 예측합니다.

그래서 아래 사진처럼 데이터의 구조를 좀 바꿔줘야 하는데요

Skip-gram은 기본적으로 중심 단어를 입력, 주변 단어를 레이블로 합니다.

하지만 SGNS에서는 기존의 Skip-gram 데이터셋에서 중심 단어와 주변 단어를 각각 입력1, 입력2로 둡니다.

이 둘은 실제로 윈도우 크기 내에서 이웃 관계였므로 레이블은 1로 합니다.

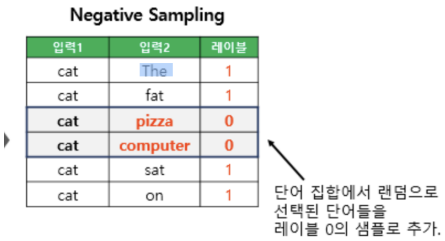


입력	레이블
cat	The
cat	fat
cat	sat
cat	on
sat	fat
sat	cat
sat	on
sat	the
...	...

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	sat	1
cat	on	1
sat	fat	1
sat	cat	1
sat	on	1
sat	the	1
...	...	...

이제 주변 단어가 아닌 (레이블=0) 단어들도 준비를 해주는데요.
전체 단어 집합 중에서 레이블이 0인 아무 단어들을 가져와서 넣고 레이블은 0으로 지정해줍니다.

이 과정에서 레이블이 0인 단어, 즉 윈도우 내에 존재하지 않는 단어를 negative sample이라고 하기 때문에 SGNS라고 부릅니다.

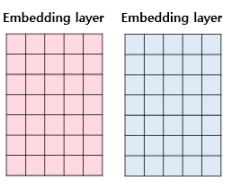


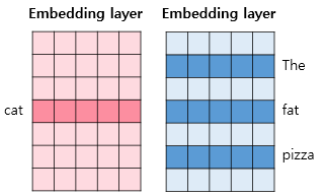
입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	pizza	0
cat	computer	0
cat	sat	1
cat	on	1

이제 두 개의 임베딩 테이블을 준비합니다. 두 임베딩 테이블은 훈련 데이터의 단어 집합의 크기를 가지므로 크기가 같습니다.

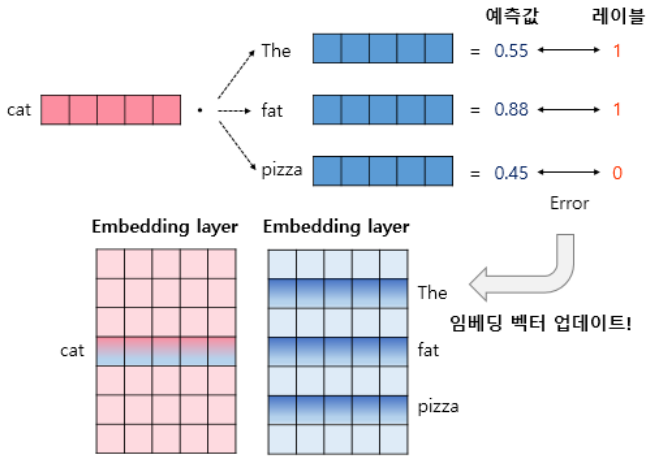
첫 테이블은 입력 1인 중심 단어의 테이블을 위한 임베딩 테이블이고,
다른 하나는 입력 2인 주변 단어의 테이블을 위한 임베딩 테이블입니다.

입력1	입력2	레이블
cat	The	1
cat	fat	1
cat	pizza	0
cat	computer	0
cat	sat	1
cat	on	1
cat	cute	1
cat	mighty	0
...	...	...





각 임베딩 테이블을 통해 테이블 룩업하여 임베딩 벡터로 변환되었다면 그 후의 연산과 업데이트는 아래와 같습니다.



중심 단어와 주변 단어의 내적값을 이 모델의 예측값으로 하고, 레이블과의 오차로부터 역전파하여 중심 단어와 주변 단어의 임베딩 벡터 값을 업데이트하는 겁니다.

셋째, 작동 방식을 이해했다면 이제 강의에서 사용하는 수식을 살펴봅시다.

$$maximize \qquad \mathbf{J}_t(\theta) = log\,\sigma(u_o^T v_c) + \sum_{k=1}^{K} log\,\sigma(-u_j^T v_c)$$

$$minimize \qquad \mathbf{J}_{neg-sample}(\mathbf{u}_o, \mathbf{v}_c, U) = -log\,\sigma(u_o^T v_c) - \sum_{k=1}^{K} log\,\sigma(-u_j^T v_c)$$

c: 중심 벡터

o: 맥락 벡터

k: 노이즈 벡터 (랜덤하게 선택된 벡터. 실제 맥락벡터가 아님.)

u: 중심 벡터와 hidden layer 사이의 가중치

v: 맥락 벡터와 hidden layer 사이의 가중치

괄호 안에 있는 두 벡터의 내적은 중심 벡터와 맥락 벡터간 유사도를 의미합니다. 이를 이용하여 위 목적함수를 해석해보면 True pair (우변의 왼쪽 항)의 경우 유사도가 클수록 확률이 높아 비용이 0에 가깝고, Noise pair (우변의 오른쪽 항)의 경우 유사도가 작을수록 확률이 높아 비용이 0에 가깝다고 해석할 수 있습니다.

<Co-occurrence matrix>

skip-gram은 중심 단어를 기준으로 맥락 단어가 등장할 확률을 계산합니다. 벡터의 값들은 중심 단어가 given일 때 각 값의 개별적인 등장 확률을 의미하기 때문에, global한(중심 단어가 not given) 출현 빈도를 파악하기 어렵습니다.
그래서 나온 것이 Co-occurrence matrix입니다.

크게 두 종류로 나뉘는데요.
1) Window based co-occurrence matrix (단어-문맥 행렬)과 2) Word-Document matrix (단어-문서 행렬)입니다.

1) Window based co-occurrence matrix는 행과 열을 전체 단어 집합의 단어들로 구성하고, i 단어의 윈도우 크기(Window Size) 내에서 k 단어가 등장한 횟수를 i행 k열에 기재한 행렬을 말합니다.

아래 3개의 문장으로 구성된 텍스트 데이터가 있다고 하면, Co-occurrence matrix는 아래와 같습니다.

I like deep learning
I like NLP

I enjoy flying

카운트	I	like	enjoy	deep	learning	NLP	flying
I	0	2	1	0	0	0	0
like	2	0	0	1	0	1	0
enjoy	1	0	0	0	0	0	1
deep	0	1	0	0	1	0	0
learning	0	0	0	1	0	0	0
NLP	0	1	0	0	0	0	0
flying	0	0	1	0	0	0	0

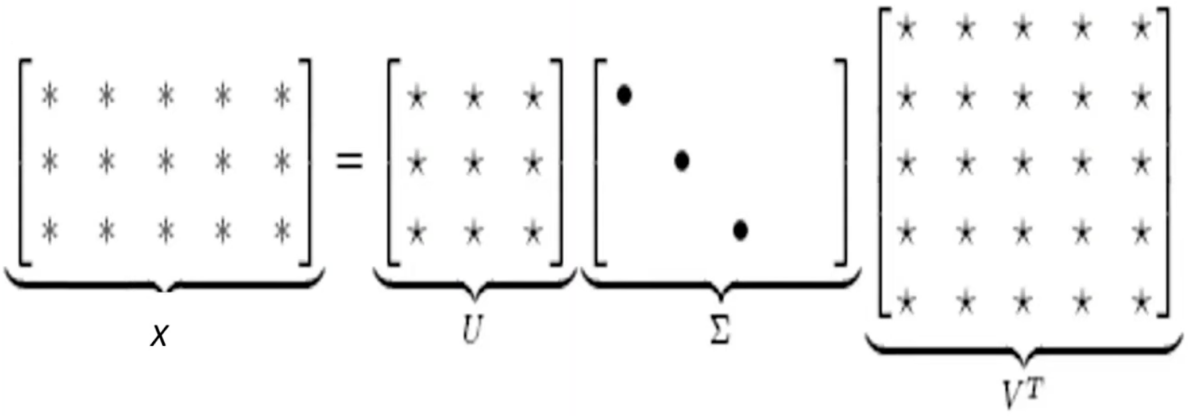
2) Word-Document matrix는 한 문서를 기준으로 각 단어가 몇 번 등장하는 지를 세어 구성합니다.

	문서1	문서2	문서3
I	1	1	1
like	1	1	0
enjoy	0	0	1
deep	1	0	0
learning	1	0	0
NLP	0	1	0
flying	0	0	1

하지만 이렇게 만든 Co-occurrence matrix는 단어의 수가 많아짐에 따라 차원이 너무 커집니다. 그리고 sparse합니다.

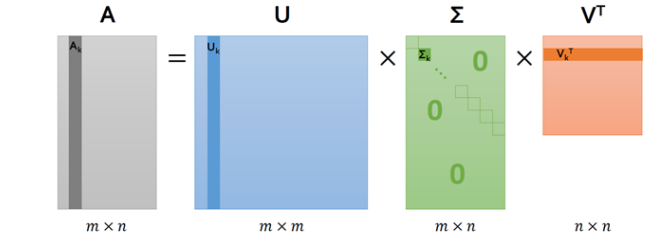
그래서 우리는 차원을 축소하기 위해 서로 관련이 있는 (=중요한) 단어들만 남기고 지우고 싶은 겁니다. (25-1000차원 정도)

<SVD (Singular Value Decomposition; 특이값 분해)>



강의에서 가져온 SVD 설명 자료

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



정방 행렬이 아니어도 가능!
(↔ 고유값 분해)

$UU^T = I$
orthogonal

Diagonal

$VV^T = I$
orthogonal

$\Sigma_k = \sqrt{\lambda_k}$

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

SVD 이해를 위한 보충 자료

아래 사진처럼 SVD를 통해 Co-occurrence matrix를 3개의 행렬의 곱으로 표현을 할 수가 있는데요.

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

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

U_k V_k^T

$UU^T = I$ orthogonal

Diagonal Matrix (대각행렬)

$VV^T = I$ orthogonal

$\Sigma_k = \sqrt{\lambda_k}$

λ_k : 행렬 AA^T 의 kth 고유값 (eigen value)

아래 사진처럼 U에서 특정 columns만 가져오더라도 원래의 Co-occurrence matrix에 가까운 행렬을 구할 수 있습니다.

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

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

λ_k : 내림차순. Non-zero

U_k V_k^T

$U'U'^T = I$ orthogonal

Diagonal Matrix (대각행렬)

$V'V'^T = I$ orthogonal

$\Sigma_k = \sqrt{\lambda_k}$

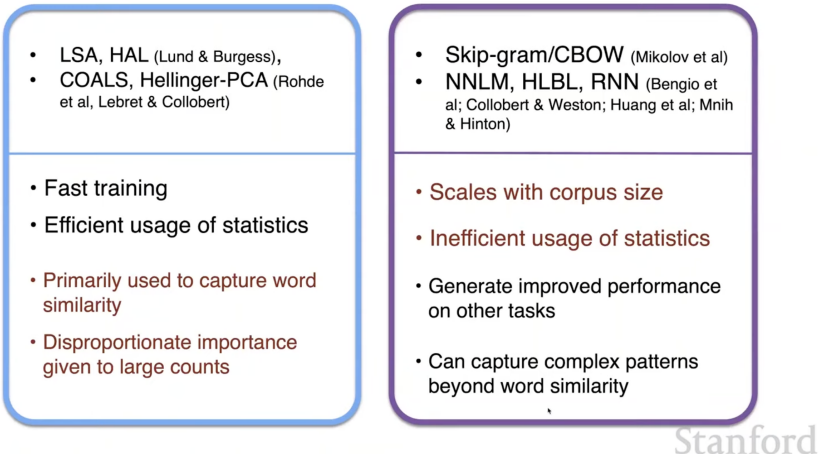
λ_k : 행렬 AA^T 의 kth 고유값 (eigen value)

참고로, SVD를 할 때에는 function words(the, he, has, …)들은 너무 자주 나오기 때문에
→ 등장 빈도에 log를 씌우거나, 등장 횟수를 100까지만 세고 그 뒤로는 count를 안하거나, 없는 단어로 취급한다고 합니다.

지금까지 Count-based 기법과 Direct prediction 기법에 해당되는 여러가지 방법들을 살펴보았습니다.

Count-based : ex. Co-occurrence matrix. 코퍼스의 전체적인(global) 등장 빈도를 고려하기는 하지만,
왕:남자 = 여왕:? (정답은 여자)와 같은 단어 의미의 유추 작업에는 성능이 떨어집니다.

Direct prediction : ex. Word2Vec. 단어 유사성을 파악할 수 있을 뿐만 아니라+ 단어의 유추 작업도 가능하지만,
사용자가 지정한 윈도우 내 단어만 고려하기 때문에 코퍼스의 전체적인 통계 정보를 반영하지 못합니다.



지금 위 그림 왼쪽에 LSA라는 말이 나오는데 SVD를 토픽 모델링에 적용하는 것을 LSA라고 부른다고 이해하시고,
LSA가 나오면 그냥 Count-based 방식 중 하나라고 생각하시면 될 듯 합니다.

아무튼, 두 기법들의 장점만을 갖춘 새로운 방법이 등장하였는데 이것이 바로 **GloVe**입니다.

<GloVe (Global Vectors for Word Representation)>

한마디로 말하면…
"임베딩된 단어벡터 간 유사도 측정을 수월하게 하면서도 (Word2Vec의 장점)
말뭉치 전체의 통계 정보를 더 잘 반영해보자 (co-occurrence matrix의 장점)"입니다.

그거… 어떻게 할건데? 생각이 들지만 일단 따라가봅시다

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

k=solid 열이 8.9로 1보다 훨씬 크기 때문에, solid는 steam보다는 ice와 관련이 있는 단어라 보는 거고요.
k=gas 열이 8.5/100로 1보다 훨씬 작기 때문에, gas는 ice보다는 steam과 관련이 있는 단어라 보는 것입니다.
k=water, fashion처럼 ice, steam과 별로 관련이 없는 단어의 경우에는 1에 근접한 확률이 나옵니다.

이제 우리의 목적은 F(ice, steam | solid) = 8.9인 함수 F를 찾는 것입니다.
정확하게 표현해보자면, solid가 주어졌을 때, ice와 steam이라는 두 벡터를 내적한 값이 8.9가 되는 함수입니다.
이걸 좀 일반화해서 있어보이게 말하면

→ "임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의 동시 등장확률 로그값이 되도록 목적함수를 정의합니다"

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

우리가 찾는 F는 위 사진처럼 표현할 수 있을텐데요, 지금 F는 세 단어 벡터를 입력받지만, 우리의 목적은 두 단어벡터의 내적을 입력받는 함수 F를 찾는 것이므로 아래처럼 표현을 바꿔줍니다.

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

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

wi, wj, wk의 관계를 알아보기 위해 wi-wj와 wk 내적

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

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

임베딩된 두 단어벡터의 내적은 전체 말뭉치의 동시등장확률이 되도록 하려는 목적이 있었으므로 우변 P를 F로 표현해줍니다.

그러면 이렇게 표현된 F는 어떤 함수가 되어야 할까요?
1) F에 들어가는 단어가 서로 바뀌어도 같은 값을 반환해야 합니다. (ice, steam은 언제든 solid, gas의 위치에 갈 수 있기 때문)
2) Co-occurrence matrix는 대칭행렬이므로 F는 이러한 성질을 반영해야 합니다.
3) F는 homomorphism을 만족해야 합니다. F(X)+F(Y) = F(X+Y) 정도로 표현할 수 있습니다.
1)2)3)을 아래에 정리해보면

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

이런 성질을 만족하는 함수는 바로바로… **지수함수**라고 하네요 (대박) → F를 exp로 바꿔주면 아래처럼 됩니다.

$$exp(w_i^T \tilde{w}_k - w_j^T \tilde{w}_k) = \frac{exp(w_i^T \tilde{w}_k)}{exp(w_j^T \tilde{w}_k)}$$
$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

가장 아래줄에서 i와 k가 서로로 바뀌더라도 같은 결과값을 얻기 위해
즉 logXi = logXk를 위해 logXi = logXk = bi+bk로 표현해줍니다. (이제 i와 k가 바뀌어도 같습니다.)

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

아래 사진처럼 이항을 해주면

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

→ 좌변은 unknown parameter로 이루어진 추정값, 그리고 우변은 실제값입니다.
우변 log Xik는, 특정 윈도우 사이즈를 두고 말뭉치 전체에서 단어별 등장 빈도를 구한 co-occurrence matrix에 로그를 취해준 행렬로, 쉽게 말해 그냥 우리가 이미 알고 있는 정보라고 생각하시면 됩니다.

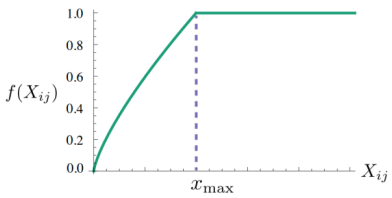
이제 추정값과 실제값이 있으니, 오차를 구할 수 있고 이 말은 loss function(object function)을 설정할 수 있다는 말이 됩니다.

오차는 Regression에서 LSE 방식과 동일하게, 오차에 제곱을 해서 다 더해주고요, 이것이 J가 됩니다.

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

GloVe 연구팀은 목적함수에 아래와 같은 모양의 $f(x)$를 추가했습니다. 아래 오른쪽 초록색 그래프를 보니 $f(x)$는 Xij가 특정 값 이상으로 튀는 경우(학습 말뭉치에서 지나치게 빈도가 높게 나타나는 단어) 이를 방지하고자 하는 장치라고 이해하시면 됩니다.

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



GloVe는 속도가 빠르며, 말뭉치 크기가 성능에 미치는 영향을 제한할 수 있으며, 그렇기 때문에 작은 말뭉치나 작은 벡터에서도 좋은 성능을 가진다는 장점이 있습니다.

GloVe에 대해 자세히 설명한 글이 있어 첨부합니다. (제가 설명드린 내용의 출처입니다. 이해가 안되시면 가서 보시면 될듯해요)

https://ratsgo.github.io/from frequency to semantics/2017/04/09/glove/

<Word embedding Evaluation>

이제는 이런 단어 임베딩 모델들을 어떤 방식으로 평가할 수 있는지 간단하게 보겠습니다.

우선 내적 평가와 외적 평가의 정의와 내용은 다음과 같습니다.

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

1) Extrinsic evaluation

실제 현실 문제 (real task)에 직접 적용하여 성능을 평가하는 방식입니다. Glove는 외적 평가에서 좋은 성능을 보입니다.

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**

2) Intrinsic evaluation

Intrinsic evaluation의 예시는 많이 있는데, GloVe는 거의 모든 평가에서 좋은 성능을 보여줍니다.

▼ Word analogy

analogy는 유사를 의미합니다. 즉 a:b :: c:? 에서 ?에 들어갈 단어를 유추하는 문제입니다.
ex. man:woman :: king:?

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

식으로 표현하면 위 사진의 식을 만족하는 d를 찾는 문제입니다.

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

<Semantic>

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

<Syntactic>

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 보였습니다.

▼ Correlation

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10 10점 만점에 10점
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

일련의 단어 쌍을 미리 구성한 후에 사람이 평가한 점수와, 단어 벡터 간 코사인 유사도 사이의 상관관계를 계산해 단어 임베딩의 품질을 평가하는 방식입니다.

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!

이 때 GloVe는 좋은 성능을 보입니다.

▼ 출처

https://wikidocs.net/22885

https://velog.io/@tobigs-text1314/CS224n-Lecture-2-Word-Vectors-and-Word-Senses

▼ 페이지모음

세타