Natural Language Processing

ToBig's 10기 박규리

Word Vectors and Word Senses

Stanford CS224N
: NLP with Deep Learning | Winter 2019

Word Vectors and Word Senses

단어를 그대로 컴퓨터에 input할 수 없다

단어를 벡터화 시키자(숫자로 만들자)

*토크나이징 된 토큰을 단어라고 치환하겠음

Word Vectors and Word Senses

$$w^{aardvark} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{a} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, w^{at} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, \cdots \; w^{zebra} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

그래서 처음에 시도한 것은 one-hot vector

but 글자 수만큼 차원이 늘어나는 문제,

단어 하나하나의 의미가 반영 안되는 문제

Word Vectors and Word Senses

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

The resulting counts matrix will then be:

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

I랑 like이 동시에 출현한 횟수 2번

이번에는 Co-occurrence Matrix 제작

한 document에 어떤 단어들이

동시 출현했는지 알 수 있음

Word Vectors and Word Senses

이번에는 Co-occurrence Matrix 제작

이것도 차원이 너무 많으므로

SVD(특이값 분해)로 차원축소

k-dimensional representation of every word 가능

$$\underset{|V|}{\begin{bmatrix} \\ \hat{X} \\ \\ \end{bmatrix}} = \underset{|V|}{\begin{bmatrix} | & | & \\ u_1 & u_2 & \cdots \\ | & | & \end{bmatrix}} \underset{k}{\begin{bmatrix} \sigma_1 & 0 & \cdots \\ 0 & \sigma_2 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}} \underset{|V|}{\begin{bmatrix} - & v_1 & - \\ - & v_2 & - \\ & \vdots & \end{bmatrix}}$$

Word Vectors and Word Senses

문제점 발생

- The matrix is extremely sparse since most words do not co-occur.
- The matrix is very high dimensional in general ($\approx 106 \times 106$)
- Quadratic cost to train (i.e. to perform SVD)
- Requires the incorporation of some hacks on X to account for the drastic imbalance in word frequency

Word Vectors and Word Senses

해결책

• Ignore function words such as "the", "he", "has", etc.

• Apply a ramp window – i.e. weight the co-occurrence count based on distance between the words in the document.

• Use Pearson correlation and set negative counts to 0 instead of using just raw count.

Word Vectors and Word Senses

해결책

• Ignore function words such as "the", "he", "has", etc.

• Apply a ramp window – i.e. weight the co-occurrence count based on distance between the words in the document.

• Use Pearson correlation and set negative counts to 0 instead of using just raw count.

iteration based methods solve many of these issues in a far more elegant manner.
하지만 우리는 조금 더 우아한 방법으로 풀어보자

Word Vectors and Word Senses

Word2vec

Word Vectors and Word Senses

A good language model will give this sentence a high probability because this is a completely valid sentence, syntactically and semantically

Word Vectors and Word Senses

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=1}^{n} P(w_i)$$

단어의 등장이 서로 독립적인 확률을 가지고 있으면 이 식으로 확률을 구해도 됨

그러나 앞에 있던 단어에 뒤의 단어가 영향을 많이 받는다는 사실을 알고 있음

그래서 밑의 조건부 확률 식이 더 정확함

$$P(w_1, w_2, \cdots, w_n) = \prod_{i=2}^{n} P(w_i|w_{i-1})$$

그러나 이것도 무리가 많이 가는 방법이라

확률을 학습하는 좀 더 좋은 모델을 알아보겠음

this would require computing and storing global information about a massive dataset.

Continuous Bag of Words Model (CBOW)

Word Vectors and Word Senses

notation

Notation for CBOW Model:

- w_i: Word i from vocabulary V
- $\mathcal{V} \in \mathbb{R}^{n \times |V|}$: Input word matrix
- v_i: i-th column of $\mathcal{V}$, the input vector representation of word w_i
- $\mathcal{U} \in \mathbb{R}^{|V| \times n}$: Output word matrix
- u_i: i-th row of $\mathcal{U}$, the output vector representation of word w_i

CBOW Model:

Predicting a center word from the surrounding context

For each word, we want to learn 2 vectors

- v: (input vector) when the word is in the context
- u: (output vector) when the word is in the center

Word Vectors and Word Senses

1. We generate our one hot word vectors for the input context of size m : $(x^{(c-m)}, \ldots, x^{(c-1)}, x^{(c+1)}, \ldots, x^{(c+m)} \in \mathbb{R}^{|V|})$.

input으로 쓸 one hot vector 만듬

2. We get our embedded word vectors for the context ($v_{c-m} = \mathcal{V}x^{(c-m)}$, $v_{c-m+1} = \mathcal{V}x^{(c-m+1)}$, …, $v_{c+m} = \mathcal{V}x^{(c+m)} \in \mathbb{R}^n$)

context에 따라 임베딩

3. Average these vectors to get $\hat{v} = \frac{v_{c-m}+v_{c-m+1}+\ldots+v_{c+m}}{2m} \in \mathbb{R}^n$

다른 모든 단어 다 학습

4. Generate a score vector $z = \mathcal{U}\hat{v} \in \mathbb{R}^{|V|}$. As the dot product of similar vectors is higher, it will push similar words close to each other in order to achieve a high score.

비슷한 단어는 similar하게 학습

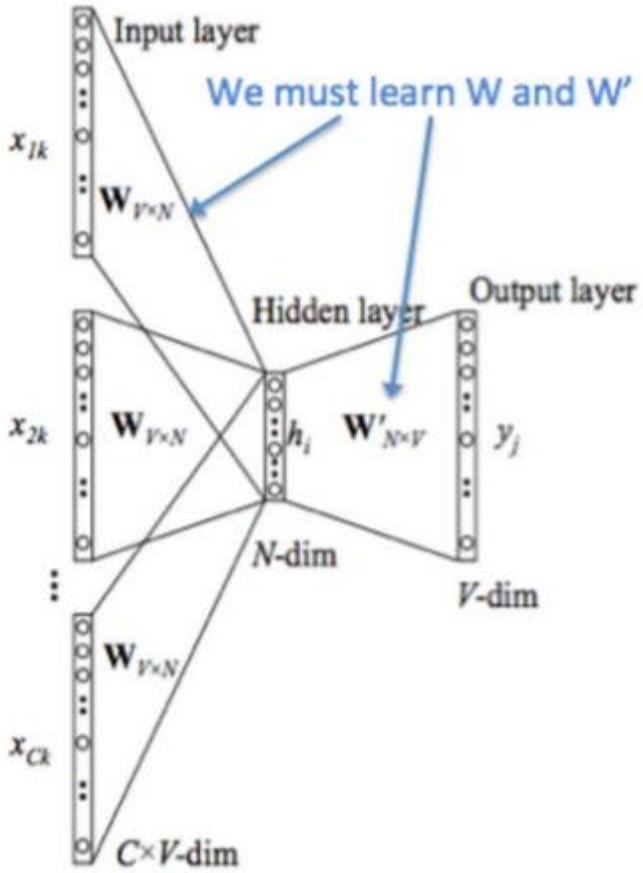
5. Turn the scores into probabilities $\hat{y} = \text{softmax}(z) \in \mathbb{R}^{|V|}$.

softmax로 확률값 도출

6. We desire our probabilities generated, $\hat{y} \in \mathbb{R}^{|V|}$, to match the true probabilities, $y \in \mathbb{R}^{|V|}$, which also happens to be the one hot vector of the actual word.

실제 y는 one hot vector
y햇(추측값)은 true가 될
확률로 접근한다

Word Vectors and Word Senses



hidden layer의 weight를 학습하는 방식

ex)

나는 학교에 간다

'학교' : y_j

나는 , 에, 간다

: x_{1k}, x_{2k}, x_{3k}...

*이러면 학교와 고등학교가 비슷한 벡터가 된다는 원리...

Word Vectors and Word Senses

$$H(\hat{y}, y) = -y_i \log(\hat{y}_i)$$

두 vector의 분포의 거리를 측정할 loss function 필요

cross entropy

$$\begin{aligned}
\text{minimize } J &= -\log P(w_c | w_{c-m}, \ldots, w_{c-1}, w_{c+1}, \ldots, w_{c+m}) \\
&= -\log P(u_c | \hat{v}) \\
&= -\log \frac{\exp(u_c^T \hat{v})}{\sum_{j=1}^{|V|} \exp(u_j^T \hat{v})} \\
&= -u_c^T \hat{v} + \log \sum_{j=1}^{|V|} \exp(u_j^T \hat{v})
\end{aligned}$$

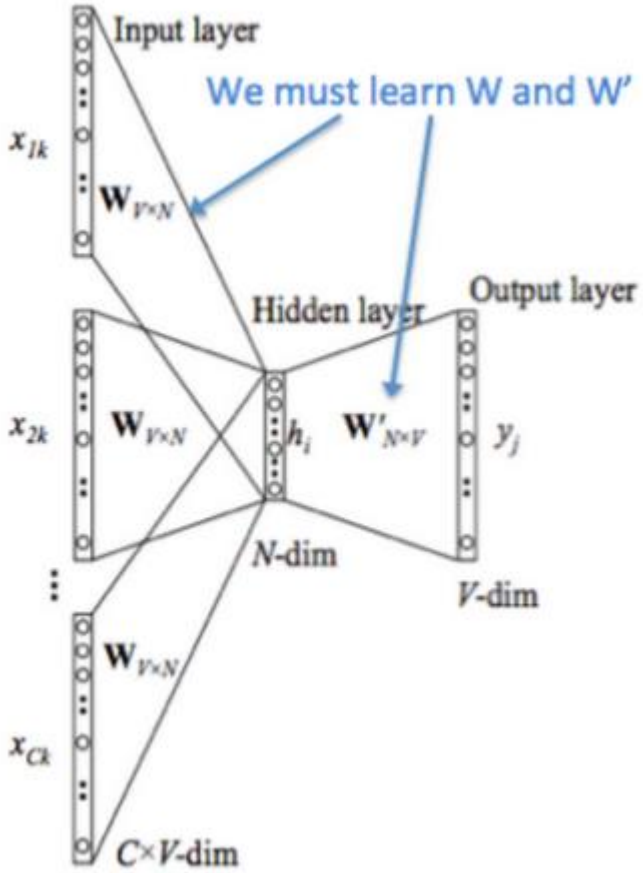
optimization objective

Wc-m…Wc-1…Wc+m일 때 Wc의 확률 극대화

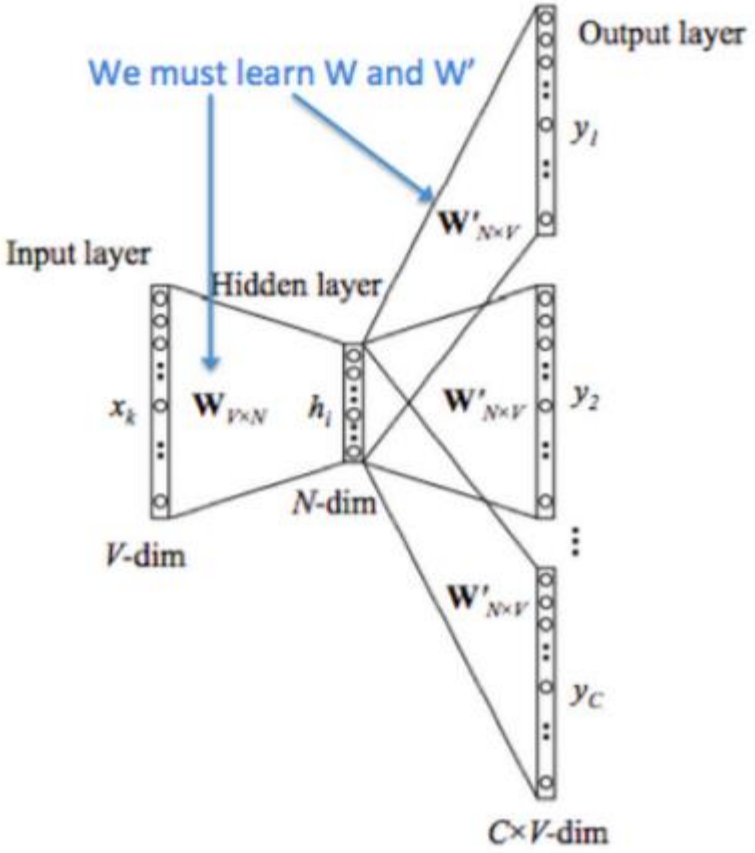
Skip-Gram Model

Word Vectors and Word Senses

CBOW



Skip-Gram



Word Vectors and Word Senses

$$
\begin{aligned}
\text{minimize } J &= -\log P(w_{c-m},\ldots,w_{c-1},w_{c+1},\ldots,w_{c+m}|w_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} P(w_{c-m+j}|w_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} P(u_{c-m+j}|v_c) \\
&= -\log \prod_{j=0,j\neq m}^{2m} \frac{\exp(u_{c-m+j}^T v_c)}{\sum_{k=1}^{|V|}\exp(u_k^T v_c)} \\
&= -\sum_{j=0,j\neq m}^{2m} u_{c-m+j}^T v_c + 2m\log\sum_{k=1}^{|V|}\exp(u_k^T v_c)
\end{aligned}
$$

Negative Sampling

Word Vectors and Word Senses

*전체 vocab사이즈만큼 단어 하나당 p(o|c) 계산 해야됨

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^{V} \exp(u_w^T v_c)}$$

문제점

Note that the summation over |V| is computationally huge!

vocabulary V 갯수만큼 전부 계산하기에는 너무 많다!

Word Vectors and Word Senses

해결책

instead of looping over the entire vocabulary, we can just sample several negative examples! We "sample" from a noise distribution (Pn(w)) whose probabilities match the ordering of the frequency of the vocabulary.

샘플링해서 계산하자!

Word Vectors and Word Senses

이 때 실제 target으로 사용하는 단어의 경우 반드시 계산을 해야하므로

이를 'positive sample' 이라고 부르고, 나머지 'negative sample' 들을 어떻게 뽑느냐가 문제가 된다.

이 뽑는 방법을 어떻게 결정하느냐에 따라 Negative sampling의 성능도 달라지고,

이는 보통 실험적으로 결정한다.

Word Vectors and Word Senses

Let's denote by $P(D = 1|w, c)$ the probability that (w, c) came from the corpus data.

Correspondingly, $P(D = 0|w, c)$ will be the probability that (w, c) did not come from the corpus data.

we build a new objective function that tries to maximize the probability of a word and context being in the corpus data if it indeed is, and maximize the probability of a word and context not being in the corpus data if it indeed is not.

corpus data에서 w와 c가 동시에 나올지 안 나올지에 대한 확률을 최적으로 만듬

Word Vectors and Word Senses

k개만 샘플링할 경우 **k개의 단어를 정해진 규칙 안에서 랜덤으로 선택**하면 된다.

이때, **고의적으로 여러개의 오답이 될만한 후보를 랜덤하게 선택한 후**

확률값에 negative를 취하는 방법을 **negative sampling**이라 부르고

이 negative sampling으로 뽑아 계산된 값을 손실함수에 추가하는 형식으로 적용 가능하다.

Word Vectors and Word Senses

정답이 될수있는 후보쌍의 확률값을 최대로 만들고, 오답이 될수있는 후보쌍의 확률을 최소로 한다

$$\theta = \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

$$= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)})$$

$$= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})$$

Word Vectors and Word Senses

$$
\begin{aligned}
\theta &= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \underset{\theta}{\operatorname{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)}) \\
&= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})
\end{aligned}
$$

(a). θ = **$\arg_\theta \max$ $\pi P(D=1|W,C,\theta)$**: center word (C), context word (W)가 주어졌을때 W,C쌍이 학습셋 D이 **나올 확률**을 최대로 만드는 θ를 찾기 (skipgram)

(b). θ = **$\arg_\theta \max$ $\pi P(D=0|W,C,\theta)$**: center word (C), context word (W)가 주어졌을때 W,C쌍이 학습셋 D에 **나오지 않을 확률**을 최대로 만드는 θ를 찾기 (negative sampling)

(c). $\theta = \arg_\theta \max \pi_D P(D=1|W,C,\theta) * \pi_{D'} P(D=0|W,C,\theta)$

$= \arg_\theta \max \pi_D P(D=1|W,C,\theta) * \pi_{D'} P(1-P(D=1|W,C,\theta))$

:결국, 정답쌍은 최대로 오답쌍은 최소로만드는 θ찾기

Word Vectors and Word Senses

비용함수

Note that maximizing the likelihood is the same as minimizing the negative log likelihood

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)$$

Note that D~ is a "false" or "negative" corpus. Where we would have sentences like "stock boil fish is toy". Unnatural sentences that should get a low probability of ever occurring. We can generate D~ on the fly by randomly sampling this negative from the word bank.

Word Vectors and Word Senses

skip gram with negative sampling

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

이부분이 바뀜

(negative sampling)

To compare with the regular softmax loss for skip-gram

$$-u_{c-m+j}^T v_c + \log \sum_{k=1}^{|V|} \exp(u_k^T v_c)$$

원래꺼

center word(v_c)와 context word(u_o)의 곱이 크면 손실함수 J가 0에 가까워진다

center word(v_c)와 context word(u_o)가 같이 나올 확률이 높으면 손실값(loss) J=0에 가까워진다

Word Vectors and Word Senses

skip gram with negative sampling

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

To compare with the regular softmax loss for skip-gram

$$-u_{c-m+j}^T v_c + \log \sum_{k=1}^{|V|} \exp(u_k^T v_c)$$

오답쌍 후보들(j~P(w))은 어떻게 뽑을까? (어떤 분포로)

원래꺼

"j~P(w)"은 랜덤 추출을 위한 "정해진 규칙"

Word Vectors and Word Senses

In the above formulation, {u˜k |k = 1 . . . K} are sampled from Pn(w). Let's discuss what Pn(w) should be. While there is much discussion of what makes the best approximation, what seems to work best is the Unigram Model raised to the power of 3/4.

보통 Negative Sampling에서 샘플들을 뽑는 것은 'Noise Distribution' 을 정의하고 그 분포를 이용하여 단어들을 일정 갯수 뽑아서 사용하는데, 논문에서는 여러 분포를 실험적으로 사용해본 결과 'Unigram Distribution의 3/4 승' 분포가 이 제일 좋았다

(자주 출몰하지 않는 단어를 뽑기 위해서 이 분포가 좋다고 함... is,the같은거 뽑으면 안대니깐)

Global Vectors for Word Representation (GloVe)

Word Vectors and Word Senses

The other set of methods are shallow window-based (e.g. the skip-gram and the CBOW models), which learn word embeddings by making predictions in local context windows.

기존 방식은 window size내에 있는 단어들만 고려해 predict한다는 한계점

아무리 window size를 늘린다해도, skipgram은 전체 단어의 동반출현 빈도수와 같은 통계 정보를 내포하지 못한다.

Word Vectors and Word Senses

GloVe consists of a weighted least squares model that trains on

global word-word co-occurrence counts and thus makes efficient use of statistics

word간의 동시출현 횟수 또한 고려해보자

$$P_{ij} = P(w_j | w_i) = X_{ij}$$

Xi be the probability of j appearing in the context of word i.

Word Vectors and Word Senses

GloVe 연구팀은 임베딩된 두 단어벡터의 내적이 말뭉치 전체에서의

동시 등장확률 로그값이 되도록 목적함수를 정의

임베딩된 단어벡터 간 유사도 측정을 수월하게 하면서도 말뭉치 전체의 통계 정보를 좀 더 잘 반영해보자

Word Vectors and Word Senses

$$Q_{ij} = \frac{\exp(\vec{u}_j^T \vec{v}_i)}{\sum_{w=1}^{W} \exp(\vec{u}_w^T \vec{v}_i)}$$

동시에 얼마나 많이 나왔나

$$J = -\sum_{i=1}^{W}\sum_{j=1}^{W} X_{ij} \log Q_{ij}$$

손실함수((weighted) least square objective)

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2$$

$$\hat{P}_{ij} = X_{ij} \text{ and } \hat{Q}_{ij} = \exp(\vec{u}_j^T \vec{v}_i)$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\log(\hat{P})_{ij} - \log(\hat{Q}_{ij}))^2$$

$$= \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} f(X_{ij})(\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

Word Vectors and Word Senses

$$Q_{ij} = \frac{\exp(\vec{u}_j^T \vec{v}_i)}{\sum_{w=1}^{W} \exp(\vec{u}_w^T \vec{v}_i)}$$

동시에 얼마나 많이 나왔나

$$J = -\sum_{i=1}^{W}\sum_{j=1}^{W} X_{ij} \log Q_{ij}$$

손실함수((weighted) least square objective)

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2$$

$$\hat{P}_{ij} = X_{ij} \text{ and } \hat{Q}_{ij} = \exp(\vec{u}_j^T \vec{v}_i)$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\log(\hat{P})_{ij} - \log(\hat{Q}_{ij}))^2$$

$$= \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} f(X_{ij})(\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

Xij 와 exp(UjVi)가 비슷해야 손실값 J=0에 가깝다는 해석이 된다.

다시 말해 Xij = (중심어에 대한 j번째 context단어가 많이 출현했다면)

exp(UjVi)의 값이 커야 J=0에 가까워진다.

Word Vectors and Word Senses

손실함수((weighted) least square objective)

$$Q_{ij} = \frac{\exp(\vec{u}_j^T \vec{v}_i)}{\sum_{w=1}^{W} \exp(\vec{u}_w^T \vec{v}_i)}$$

$$J = -\sum_{i=1}^{W}\sum_{j=1}^{W} X_{ij} \log Q_{ij}$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\hat{P}_{ij} - \hat{Q}_{ij})^2$$

$$\hat{P}_{ij} = X_{ij} \text{ and } \hat{Q}_{ij} = \exp(\vec{u}_j^T \vec{v}_i)$$

동시에 얼마나 많이 나왔나

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\log(\hat{P})_{ij} - \log(\hat{Q}_{ij}))^2$$

$$= \sum_{i=1}^{W}\sum_{j=1}^{W} X_i (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

$$\hat{J} = \sum_{i=1}^{W}\sum_{j=1}^{W} f(X_{ij}) (\vec{u}_j^T \vec{v}_i - \log X_{ij})^2$$

Xij는 가중치값인데 is나 the같은게 gradient decent에 너무 영향을 많이 줄 수 있어 f(Xij)로 바꿈

Word Vectors and Word Senses

word embedding이 학습하는 원리

우리는 앞서 word embedding을 학습시킬때 중심어 C와 주변 단어들 O를 활용했다. king의 주변어는 royal, familiy, emperor 등 이 예상되고 이런 단어는 마찬가지로 queen의 주변단어로 자주 등장할 것이다. 하지만, king과 king을 구분할 수 있는 주변단어 O가 몇가지 있을 수 있는데.. 아마 다음과 같은 문장들로 부터 파생 될꺼다. "the king was a man", "the queen was a woman" 또한, king은 man과 가까운단어 he랑도 자주 등장할것이다. "he will be a man", "he will be a king", "he will be a uncle" 이처럼 중심어 king은 man, he, uncle과 문법적, 그리고 의미적으로 woman, she보다 상대적으로 더 많이 등장할것이고 뿐만아니라 문장에서 king의 자리는 man, he, uncle로 대체 될 수 있다. 즉 상대적으로 she, woman보다 주변 단어 O가 비슷하다는 이야기다. (물론 "she is the wife of the king"과 같은 문장이 나올 수 있으나 상대적으로 생각해야한다)

결과적으로, word embedding은 위에서 설명한 의미적 문법적 정보를 단어의 동반출현 빈도를 통해 상대적으로 학습한다.

Word Vectors and Word Senses

읽어보면 좋을 자료

word2vec 등 embedding이 아직 생소하신 분들 강추! (일러스트 그림이 예술)

https://jalammar.github.io/illustrated-word2vec/?fbclid=IwAR0plv8YFoTCMYpoqInRo-7w5EuU83bTR0Jad2dtSSaojx0M0saa3b0uvBY

스탠포드 강의 노트

http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes01-wordvecs1.pdf

http://web.stanford.edu/class/cs224n/readings/cs224n-2019-notes02-wordvecs2.pdf

Word Vectors and Word Senses

읽어보면 좋을 자료

ratsgo

https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/03/30/word2vec/

https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/04/09/glove/

shuuki4

https://shuuki4.wordpress.com/2016/01/27/word2vec-%EA%B4%80%EB%A0%A8-%EC%9D%B4%EB%A1%A0-%EC%A0%95%EB%A6%AC/

들어주셔서 감사합니다.