

Word2Vec

FastText



Word2Vec & FastText 실습

■ Word Embedding 활용

- Word Embedding을 현실에서 그대로 사용 하는 경우는 많이 없음. 특히 NLP 분야에서는…
→ ex) 단어 유사도
- 주요 Model에서는 Corpus를 sequence로 변환 후 학습이 대다수
: sequence는 vector Ex) count-based vector
- 그러나, Word Embedding에서 사용되는 이론을 바탕으로 작은 규모의 모델을 만드는 경우는 있음.

■ 실습

- Gensim
: <https://radimrehurek.com/gensim/index.html#install>
: 2013년 Google
- FastText
: <https://github.com/facebookresearch/fastText/>
: http://bit.ly/fasttext_win
: 2016년 facebook

임베딩(embedding)이란?

■ 개념

- Sparce한 one-hot Encoding을 Dense로 표현하기 위해 Dense한 Embedding Vector을 곱함 즉, Dense한 Vector로 표현하기 위한 Embedding Vector를 찾는 것

$$\bullet \mathbf{x}_{embedding} = \mathbf{W}_{embedding} * \mathbf{x}_{one_hot}$$

■ 예시

"blue"	"blues"	"orange"	"oranges"		"blue"	"blues"	"orange"	"oranges"
[1	[0	[0	[0		0.48	0.46	-0.11	-0.13
0	1	0	0	Embedding	0.45	0.42	-0.23	-0.21
0	0	1	1		-0.2	-0.19	0.76	0.72
0	0	0
.
.
0]	0]	0]	0]		0.01]	0.03]	-0.21]	-0.3]
10000x1	10000x1	10000x1	10000x1		250x1	250x1	250x1	250x1

- blue와 blues는 비슷한 값, orange와 oranges는 비슷한 값
- <https://word2vec.kr/search/>
: 미국 - 뉴욕 + 서울 = 한국

Word2Vec 사전 지식

■ Monte Carlo 평균

- $f(x) \rightarrow$ Model, $P(x) \rightarrow$ 확률분포, $\mathbb{E} \rightarrow$ 기대값
: n이 많을 수록(=sampling이 많을 수록) 실제에 근사

$$\mathbb{E}_{x \sim P(x)} [f(x)] \approx \frac{1}{n} \sum_{i=1}^n f(x_i), \text{ where } x_i \sim P(x).$$

■ Maximum Likelihood Estimation

- θ 는 확률값으로 이루어진 집합
- Deep Learning에서는 Log Likelihood로 문제 해결
: 덧셈이 곱셈보다 속도가 빠름

$$\prod_{i=1}^n P_\theta(x=x_i) \quad \Rightarrow \quad \sum_{i=1}^n \log P_\theta(x=x_i)$$

- Maximum Likelihood와 Cross-entropy는 같은 개념이며, 책마다 혼용하여 사용
: 추후에 재 확인 하겠음.

Word2Vec(CBOW, Skip-gram)

■ CBOW, Skip-gram

- CBOW ...> 입력 : 주변단어, 출력 : 중심단어
- Skip-gram ...> 입력 : 중심단어, 출력 : 주변단어
- window ...> 중심단어와 주변단어의 합이며 embedding 할 word 수
: 책마다 다르게 표현하나 여기서는 총합
: 옆 그림에서는 windows = 5

중심 단어	주변 단어
[1, 0, 0, 0, 0, 0]	[0, 0, 1, 0, 0, 0]
[0, 1, 0, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0]
[0, 0, 1, 0, 0, 0]	[1, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0]	[0, 1, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0], [0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0]	[0, 0, 1, 0, 0, 0], [0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1]	[0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 1, 0], [0, 0, 0, 0, 0, 1]

출처 : <https://wikidocs.net/book/2155>

■ Training Strategy

- Skip-gram
: 옆 그림처럼 하나씩 training

$$p(o|c)$$

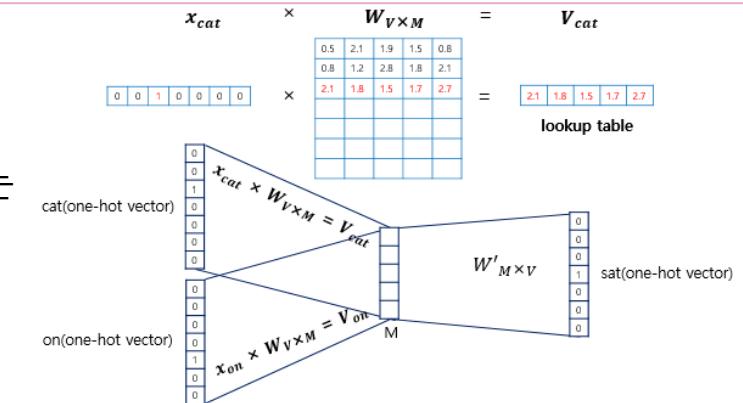
...> C : 중심단어 vectors
O : 주변단어 vectors

Source Text	Training Samples
The quick brown fox jumps over the lazy dog.	(the, quick) (the, brown)
The quick brown fox jumps over the lazy dog.	(quick, the) (quick, brown) (quick, fox)
The quick brown fox jumps over the lazy dog.	(brown, the) (brown, quick) (brown, fox) (brown, jumps)
The quick brown fox jumps over the lazy dog.	(fox, quick) (fox, brown) (fox, jumps) (fox, over)

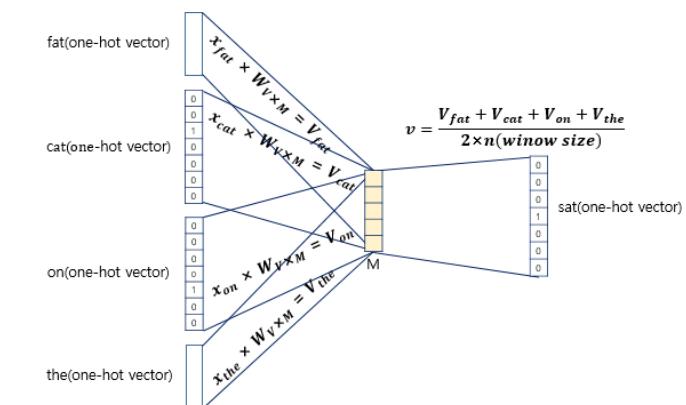
Word2Vec(CBOW, Skip-gram)

- 은닉층 1개
→ 투사층(projection layer)
- 활성화 함수 없음

- 학습 : 가중치(W, W')를 업데이트 하는 것
- input-vector, output-vector : one-hot vector
- projection layer : input-vector, output-vector와는 다른 차원, W 는 input-vector보다는 작은 차원, W' 은 output-vector와 같은 차원
→ Auto-Encoder와 유사
- lookup table
: W 중 특정 sample과 같은 값이며 학습 후
embedding vector
- CBOW은 평균으로 계산한 값으로 저장
Skip-gram은 평균이 필요 없음



출처 : <https://wikidocs.net/book/2155>

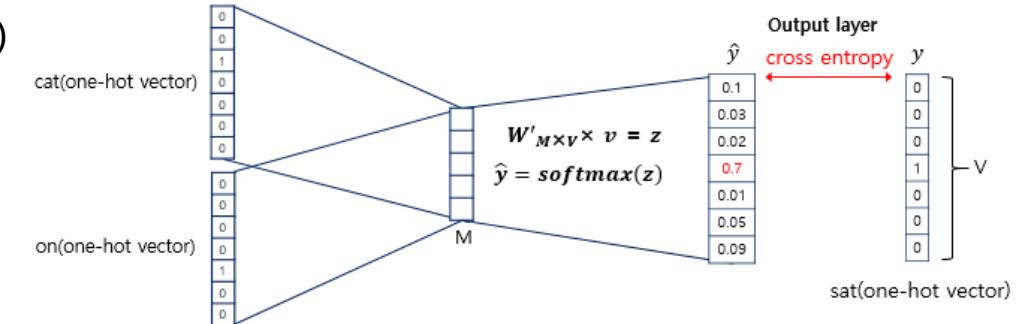


Word2Vec(CBOW, Skip-gram)

■ 최종 모델

- $\hat{y} = \text{softmax}(W^* \text{ 평균 벡터})$
- loss function은 cross entropy

: Classification 문제로 바꿈



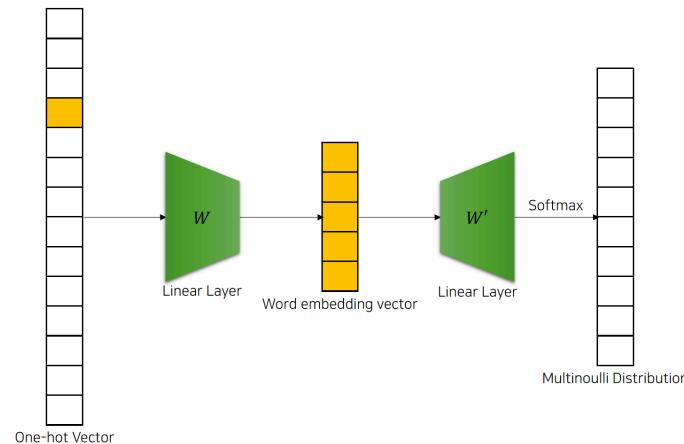
출처 : <https://wikidocs.net/book/2155>

- Back Propagation로 W 와 W' 을 업데이트
- Skip-gram은 CBOW와 동일한 이론(?)
- Skip-gram이 CBOW보다 성능이 우수
- 최종 학습된 모델
 - W 와 W' 행렬을 가지는 벡터이며 embedding vector라고 함
 - 활성화 함수가 없는 두개의 Linear 함수(W , W')

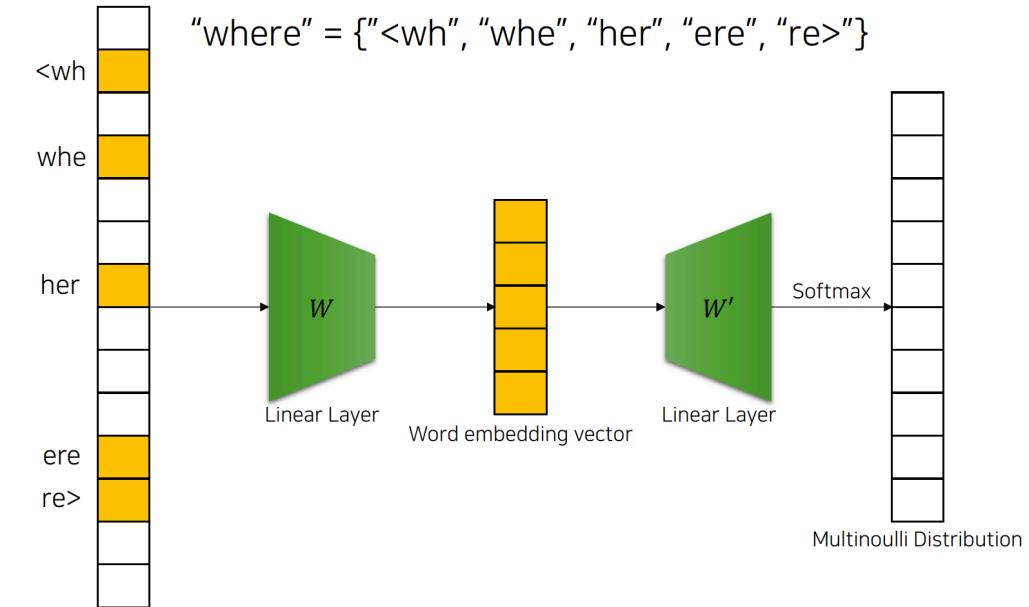
FastText

■ Skip-gram 업그레이드 버전

[skip-gram]



- Skip-gram의 단점
 - : OoV에 매우 취약
 - : 저 빈도 단어에 대해 예측 어려움
- **subword**
 - : word를 window 크기로 나누어서 학습



- input : skip-gram과는 다르게 one-hot은 아님
- word embedding vector = subword embedding vector 합
→ 중심단어가 특정 subword 구성되어 있을 때 주변단어를 예측하는 것