

17기 정규세션

ToBig's 16기 전민진

NLP Basic

Contents

Unit 01 | Introduction

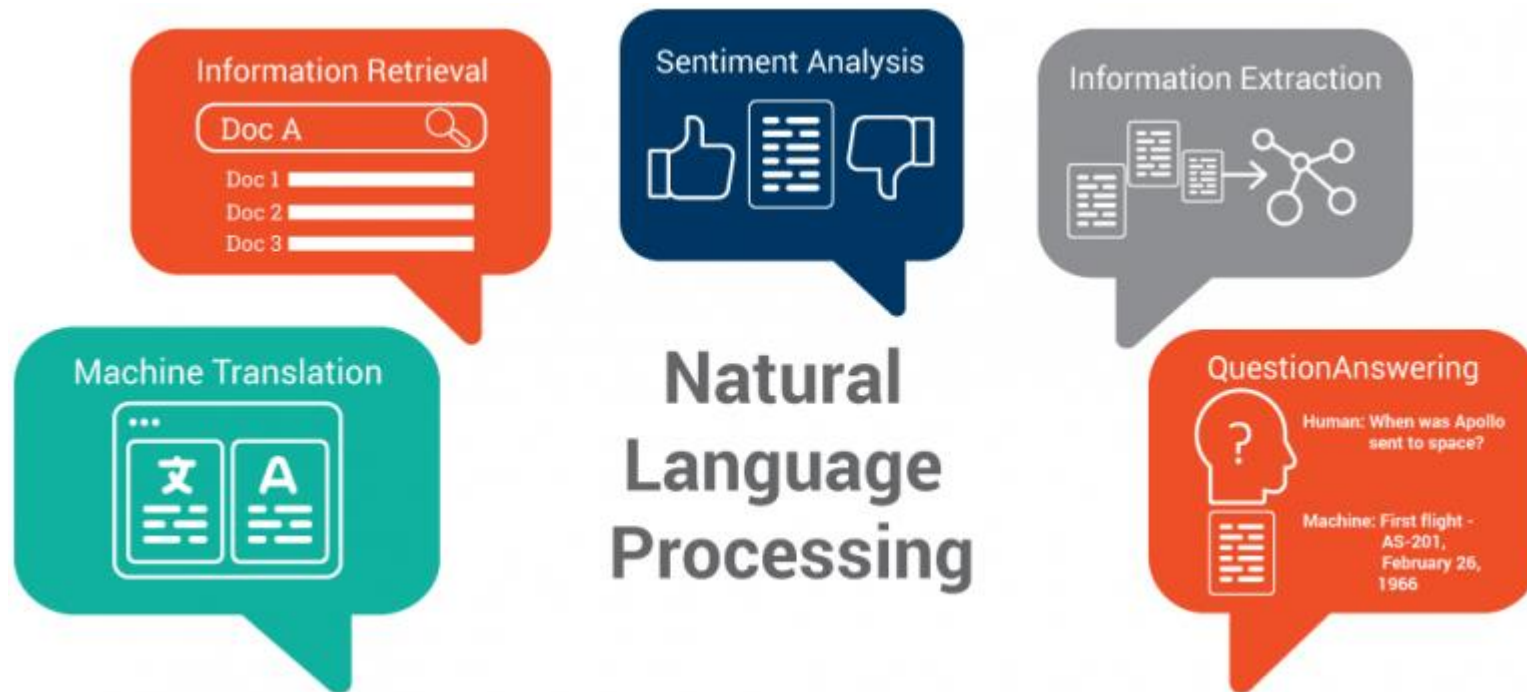
Unit 02 | Text preprocessing

Unit 03 | LM(Language Model)

Unit 04 | 단어 의미 & 단어 표현

Unit 05 | Word Embedding

Unit 01 | Introduction



Contents

Unit 01 | Introduction

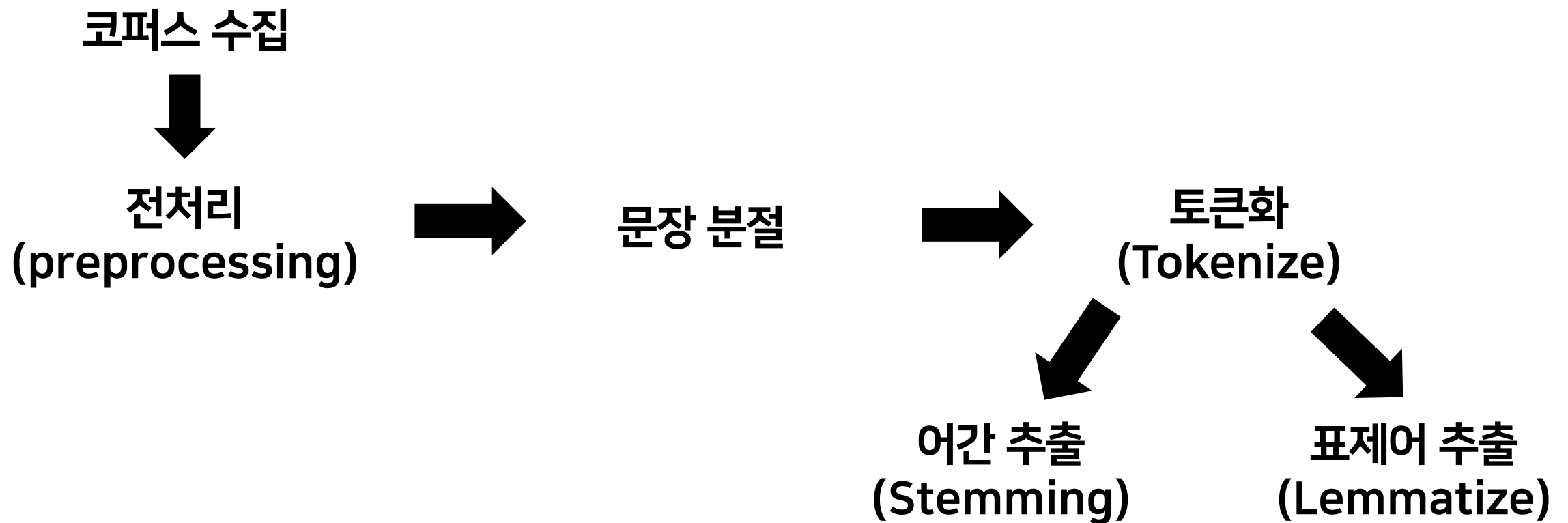
Unit 02 | Text preprocessing

Unit 03 | LM(Language Model)

Unit 04 | 단어 의미 & 단어 표현

Unit 05 | Word Embedding

Unit 02 | Text preprocessing



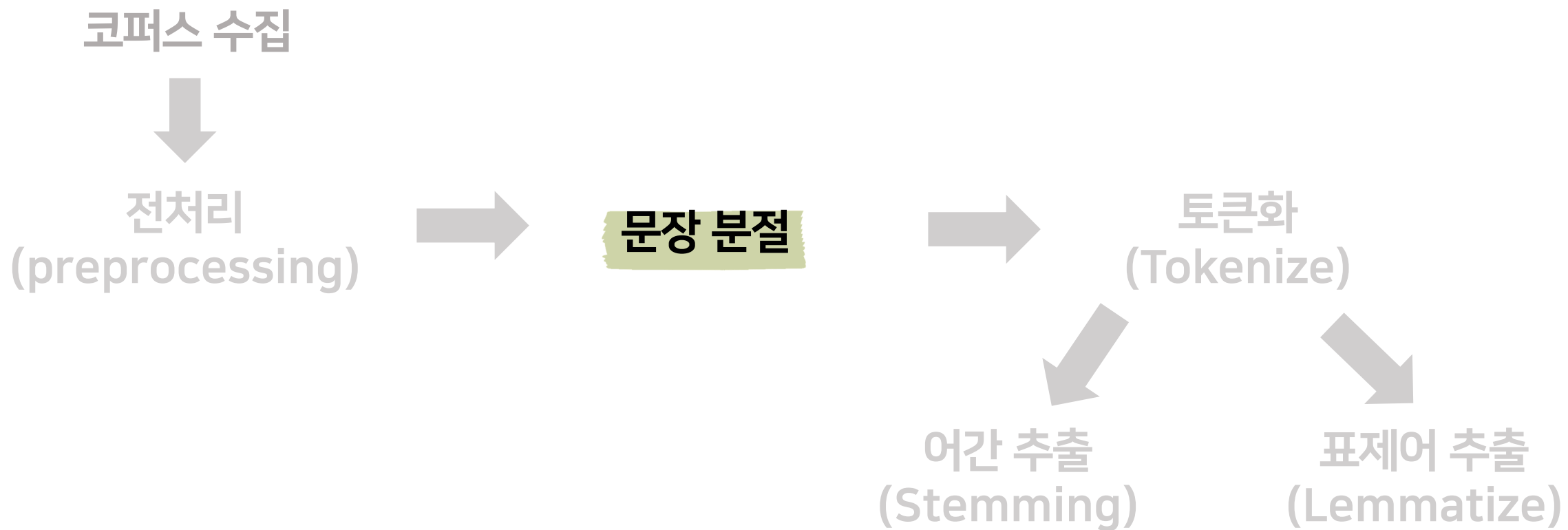
Unit 02 | Text preprocessing



Unit 02 | Text preprocessing



Unit 02 | Text preprocessing



Unit 02 | Text preprocessing

문장 분절 (Sentence Tokenization)

- 대부분의 NLP task는 입력 단위가 문장이기에 문장 단위로 분절해주는 절차가 필요하다.
- 하지만, 단순히 마침표를 기준으로 문장을 분절할 경우, 약어나 소수점, 네트워크 주소가 있을 경우 잘못 분절될 수 있다.
- 따라서, nltk, kss, genia 등에서 제공하는 **문장 분절 모듈**을 사용해야한다.

```
[ ] from nltk.tokenize import sent_tokenize
```

```
[ ] text="""His barber kept his word.  
But keeping such a huge secret to himself was driving him crazy.  
Finally, the barber went up a mountain and almost to the edge of a cliff.  
He dug a hole in the midst of some reeds.  
He looked about, to mae sure no one was near.  
"""
```

```
[ ] sent_tokenize(text)
```

```
['His barber kept his word.',  
'But keeping such a huge secret to himself was driving him crazy.',  
'Finally, the barber went up a mountain and almost to the edge of a cliff.',  
'He dug a hole in the midst of some reeds.',  
'He looked about, to mae sure no one was near.']
```

Unit 02 | Text preprocessing

문장 분절 (Sentence Tokenization)

```
[ ] text2="I am actively looking for Ph.D. students. You are not a Ph.D student. Sorry."
```

```
[ ] sent_tokenize(text2)
```

```
['I am actively looking for Ph.D. students.',  
 'You are not a Ph.D student.',  
 'Sorry.']
```

```
[ ] text3 = "Network prefix: IP 255.255.255.0. Network Address number is 192.168.56.31. Thanks!"
```

```
[ ] sent_tokenize(text3)
```

```
['Network prefix: IP 255.255.255.0.',  
 'Network Address number is 192.168.56.31.',  
 'Thanks!']
```

Unit 02 | Text preprocessing



Unit 02 | Text preprocessing

토큰화(Tokenization)

- 주어진 코퍼스에서 토큰이라고 불리는 단위로 나누는 작업을 의미한다.
- 토큰은 의미를 갖는 단위로 정의된다. 즉, 알파벳 단위로 토큰화를 하지 않는 이유는 해당 단위가 의미를 담지 않기 때문이다.
- 언어는 유한 규칙을 가지고 무한한 문장을 생성하는 특징이 있기 때문에, 의미가 있는 단위로 잘라줘야 한다.

There is a cat



"There", "is", "a", "cat"

고양이가 한 마리 있다.



"고양이", "가", "한", "마리", "있다"

Unit 02 |

Text preprocessing

토큰화(Tokenization)

- 토큰화 역시 모듈을 사용해 쉽게 할 수 있다.
- 하지만 각 모듈별로 마침표, 아포스트로피 등을 처리하는 방법이 다르기 때문에 적절히 선택해야한다.

```
from nltk.tokenize import TreebankWordTokenizer, WordPunctTokenizer, word_tokenize
from nltk.tokenize.moses import MosesTokenizer
```

word_tokenize	I	am	actively	looking	for	Ph.D.	students	.	None	None	None
WordPunctTokenizer	I	am	actively	looking	for	Ph	.	D	.	students	.
TreebankWordTokenizer	I	am	actively	looking	for	Ph.D.	students	.	None	None	None
MosesTokenizer	I	am	actively	looking	for	Ph.D	.	students	.	None	None

Unit 02 |Text preprocessing

토큰화(Tokenization)

word_tokenize	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None
WordPunctTokenizer	Network	Address	number	is	192	.	168	.	56	.	31	.
TreebankWordTokenizer	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None
MosesTokenizer	Network	Address	number	is	192.168.56.31	.	None	None	None	None	None	None

word_tokenize	I	'll	be	back	as	soon	as	possible	.	None
WordPunctTokenizer	I	'	ll	be	back	as	soon	as	possible	.
TreebankWordTokenizer	I	'll	be	back	as	soon	as	possible	.	None
MosesTokenizer	I	'll	be	back	as	soon	as	possible	.	None

Unit 02 | Text preprocessing

토큰화(Tokenization)

- 한국어의 경우 영어보다 복잡하다.
- 이는 한국어가 교착어(어근+접사)이기 때문이다.

```
from konlpy.tag import Okt, Kkma
```

```
Okt  열심히 코딩 하지 않은 당신 , 조금 더 열심히 해보는 것 은 어떨까 요 ? None None None
```

```
Kkma 열심히 코딩 하 지 않 은 당신 , 조금 더 열심히 해보 는 것 은 어떻 ㄹ까요 ?
```

```
Okt IP 192.168 . 56.31 서버 에 들어가서 로그 파일 저장 해서 tobigs1516@gmail.com 로 결과 좀 보내줘 . None None None None None None None
```

```
Kkma IP 192.168 . 56.31 서버 에 들어가 서 로그 파일 저장해 서 tobigs 1516 @ gmail . com 로 결과 좀 보내주 어 .
```

Unit 02 |Text preprocessing

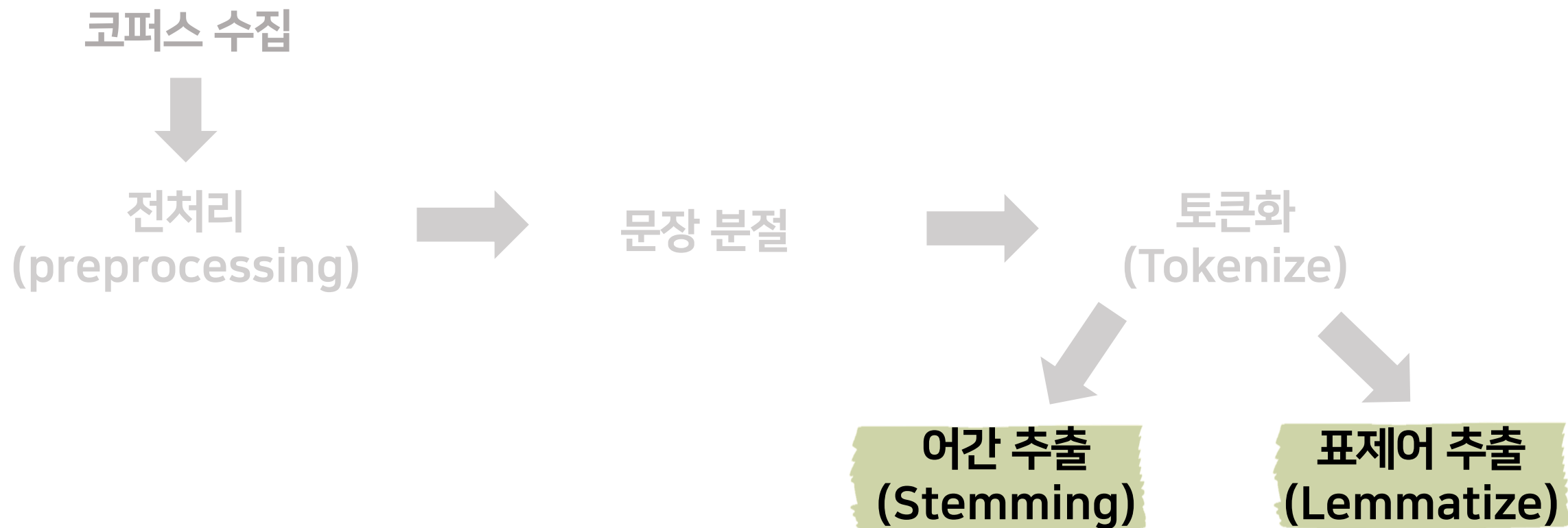
품사 태깅(Part-of-speech tagging)

- 단어는 표기는 같지만 품사에 따라 단어의 의미가 달라지기도 한다. Ex) fly - ‘날다’, ‘파리’
- 단어의 의미를 제대로 파악하기 위해선 해당 단어가 어떤 품사로 쓰였는지 보는 것이 중요할 수 있다.

```
from konlpy.tag import Okt, Kkma, Hannanum
```

okt	(한국, Noun)	(은, Josa)	(지난, Noun)	(2년, Number)	(동안, Noun)	(아시 아, Noun)	(지역, Noun)	(을, Josa)	(힘썌, Adjective)	(경제, Noun)	(적, Suffix)	(위기, Noun)	(를, Josa)	(국민, Noun)	(과, Josa)	(정부, Noun)	(의, Josa)	(헌신, Noun)	(,, Punctuation)	(그리고, Conjunction)
kkma	(한국, NNG)	(은, JX)	(지나, VV)	(ㄴ, ETD)	(2, NR)	(년, NNM)	(동안, NNG)	(아시 아, NNG)	(지역, NNG)	(을, JKO)	(힘썌, VV)	(ㄴ, ETD)	(경제적, NNG)	(위기, NNG)	(를, JKO)	(국민, NNG)	(과, JC)	(정부, NNG)	(의, JKG)	(헌신, NNG)
hannanum	(한국, N)	(은, J)	(지나, P)	(ㄴ, E)	(2년, N)	(동안, N)	(아시 아, N)	(지역, N)	(을, J)	(힘썌, P)	(ㄴ, E)	(경제적, N)	(위, N)	(이, J)	(기, E)	(를, J)	(국 민, N)	(과, J)	(정부, N)	(의, J)

Unit 02 | Text preprocessing



Unit 02 | Text preprocessing

Stemming & Lemmatization

- 눈으로 봤을 땐, 서로 다른 단어지만 하나의 단어로 일반화시켜 문서 내의 단어 수를 줄이고자 할 때 사용한다.
- 뒤에서 배울 BoW(Bag of Words) 표현을 사용하는 자연어 처리 문제에서 주로 사용한다.
- 즉, 자연어처리에서 전처리, 정규화의 지향점은 언제나 갖고 있는 코퍼스로부터 복잡성을 줄이는 일이다.

Lemmatization(표제어 추출)

- 표제어(Lemma)는 '표제어' 또는 '기본 사전형 단어' 정도의 의미를 갖는다.
- 즉, 표제어 추출은 단어로부터 표제어를 찾아가는 과정이다.
- 표제어 추출은 **형태소**에서 **어간**과 **접사**를 분리하는 작업을 말한다.

단어의 의미를 담고 있는 단어의 핵심 부분

단어의 추가적인 의미를 주는 부분

am -> be
having -> have
cats -> cat

Unit 02 |Text preprocessing

Stemming(어간 추출)

- 어간 추출은 형태학적 분석을 단순화한 버전이라고 볼 수도 있고, 정해진 규칙만 보고 단어의 어미를 자르는 작업이라고 볼 수 있다.
- 이 작업은 섬세한 작업이 아니기 때문에, 결과 단어가 사전에 존재하지 않는 단어일 수도 있다.

This -> Thi
Was -> wa

```
from nltk.stem import PorterStemmer, LancasterStemmer, WordNetLemmatizer
```

Token	There	was	no	hope	for	him	this	time	:	it	was	the	third	stroke	.	Night	after	night	I	had	passed	the	house	and	studied	the	lighted
PorterStemmer()	there	wa	no	hope	for	him	thi	time	:	it	wa	the	third	stroke	.	night	after	night	I	had	pass	the	hous	and	studi	the	light
LancasterStemmer()	ther	was	no	hop	for	him	thi	tim	:	it	was	the	third	stroke	.	night	aft	night	i	had	pass	the	hous	and	study	the	light
WordNetLemmatizer()	There	wa	no	hope	for	him	this	time	:	it	wa	the	third	stroke	.	Night	after	night	I	had	passed	the	house	and	studied	the	lighted

Token	don't	doing	have	has	going	goes	gone	went	puppies	living	lives	fly	mainly	dies	watched	starting	policy	organization
LancasterStemmer()	don't	do	have	ha	go	goe	gone	went	puppi	live	live	fli	mainli	die	watch	start	polici	organ
WordNetLemmatizer()	don't	doing	have	ha	going	go	gone	went	puppy	living	life	fly	mainly	dy	watched	starting	policy	organization

Unit 02 | Text preprocessing

Integer Encoding(정수 인코딩)

- 컴퓨터가 자연어를 처리하기 위해선, 자연어를 컴퓨터가 이해할 수 있는 숫자로 바꿔줘야 한다.
- 자연어처리에는 텍스트를 숫자로 바꾸는 여러가지 기법들이 있고, 이러한 기법들을 본격적으로 적용하기 위한 첫 단계로 각 단어(토큰)을 고유한 정수에 맵핑(mapping)시키는 전처리 작업이 필요하다.

나는 밥을 먹었다.
배가 부른다.
집에 가고 싶다.
루빅스 사랑해요.
일안하 돈 벌고 싶어
⋮

encoding
→

7 12 153 256 5100
27 28 29 100
⋮
⋮
⋮

```
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
preprocessed_sentences = [['barber', 'person'], ['barber', 'good', 'person'],  
                           ['barber', 'huge', 'person'], ['knew', 'secret'],  
                           ['secret', 'kept', 'huge', 'secret'], ['huge', 'secret'],  
                           ['barber', 'kept', 'word'], ['barber', 'kept', 'word'],  
                           ['barber', 'kept', 'secret'],  
                           ['keeping', 'keeping', 'huge', 'secret', 'driving', 'barber', 'crazy'],  
                           ['barber', 'went', 'huge', 'mountain']]
```

```
tokenizer = Tokenizer()  
# fit_on_texts()안에 코퍼스를 입력으로 하면 빈도수를 기준으로 단어 집합을 생성.  
tokenizer.fit_on_texts(preprocessed_sentences)
```

```
print(tokenizer.word_index)
```

```
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7, 'good': 8, 'went': 9, 'mountain': 10}
```

Unit 02 | Text preprocessing

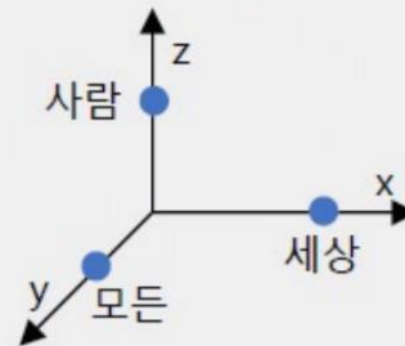
One-hot Encoding

- 자연어를 컴퓨터가 인지할 수 있는 수치로 바꾸는 가장 간단한 방법!
- 각 토큰에 배정된 인덱스에 1, 나머지를 0으로 표현하는 방법
- 차원 == 어휘 집합 크기

세상 모든 사람

단어	Vector
세상	[1, 0, 0]
모든	[0, 1, 0]
사람	[0, 0, 1]

n개의 단어는 n차원의 벡터로 표현



Unit 02 | Text preprocessing

One-hot Encoding

- 차원의 저주
 - 단어의 개수가 늘어날수록, 벡터를 저장하기 위해 필요한 공간이 계속 늘어난다.
 - 같은 크기의 공간에 표현되는 정보의 양의 적다(sparse representation).
- 단어의 의미 정보 표현 불가
 - 벡터 연산 시 결과값이 0이 되어, similarity 계산 불가
 - 이상적으로는, 단어들 사이의 유사성이 높으면 가까이 위치해야한다.
 - 하지만, One-hot encoding에선 모든 단어들 사이의 거리가 일정하다.

단어의 의미를 반영하면서도, Dense한 낮은 차원의 벡터를 만들어보자!

Contents

Unit 01 | Introduction

Unit 02 | Text preprocessing

Unit 03 | LM(Language Model)

Unit 04 | 단어 의미 & 단어 표현

Unit 05 | Word Embedding

Unit 03 | Language Model

언어 모델(Language Model)

- 언어 모델(Language Model)이란 단어 시퀀스(문장)에 확률을 할당하는 모델이다.
== 이전 단어(주변 단어)들이 주어졌을 때 다음 단어를 예측하도록!
- 단어 시퀀스에 확률을 할당하는 것이 “왜” 필요한가
 - 기계 번역(Machine Translation)
: $P(\text{“나는 버스를 탔다”}) > P(\text{“나는 버스를 태운다”})$
 - 오타 교정(Spell Correction)
: 퇴근하고 집으로 버스를
 $P(\text{“타고 간다”}) > P(\text{“달려 간다”})$

Unit 03 | Language Model

언어 모델(Language Model)

- 주어진 이전 단어들로부터 다음 단어를 “어떻게” 예측할 것인가

하나의 단어를 w , 단어 시퀀스를 W 라고 한다면, n 개의 단어가 등장하는 단어 시퀀스 W 의 확률은 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n)$$

이를 바탕으로 $n-1$ 개의 단어가 나열된 상태에서 n 번째 단어의 확률은 다음과 같이 쓸 수 있다.

$$P(w_n | w_1, \dots, w_{n-1})$$

전체 단어 시퀀스 W 의 확률은 모든 단어가 예측 되어야 알 수 있으므로 단어 시퀀스의 확률은 다음과 같다.

$$P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$$

Unit 03 | Language Model

통계적 언어 모델(Statistical Language Model, SLM)

- 문장에 대한 확률 $P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{n=1}^n P(w_n | w_1, \dots, w_{n-1})$

조건부 확률을 이용해서 문장 "An adorable little boy is spreading smiles"의 확률을 식으로 표현해 보자

$P(\text{An adorable little boy is spreading smiles}) =$

$P(\text{An}) \times P(\text{adorable} | \text{An}) \times P(\text{little} | \text{An adorable}) \times P(\text{boy} | \text{An adorable little}) \times P(\text{is} | \text{An adorable little boy})$
 $\times P(\text{spreading} | \text{An adorable little boy is}) \times P(\text{smiles} | \text{An adorable little boy is spreading})$

이 때, 각각의 확률을 "카운트에 기반하여 계산"

$$P(\text{is} | \text{An adorable little boy}) = \frac{\text{count}(\text{An adorable little boy is})}{\text{count}(\text{An adorable little boy})}$$

하지만 데이터에 "An adorable little boy is"가 없다면?

=> 위의 확률이 0이 됨.



희소 문제(sparsity problem) 발생

Unit 03 | Language Model

N-gram 언어 모델(N-gram Language Model)

- SLM의 일종이지만, 이전에 등장한 모든 단어를 고려하는 것이 아니라 일부 단어만 고려하는 방법
- N-gram에서 N은 이전 단어를 몇 개 볼지(N-1)를 결정
- 갖고 있는 코퍼스에서 n개의 단어 문치 단위로 끊어서 이를 하나의 토큰으로 간주

Ex) 4-gram

~~An adorable little~~ boy is spreading ?
무시됨! ↘
n-1개의 단어

$$P(w|\text{boy is spreading}) = \frac{\text{count}(\text{boy is spreading } w)}{\text{count}(\text{boy is spreading})}$$

Unit 03 | Language Model

N-gram 언어 모델(N-gram Language Model)

[장점] sparsity problem 일부 해소

- N이 작아질수록 카운트를 할 수 있을 가능성을 높일 수 있음

$$P(\text{is}|\text{An adorable little boy}) \approx P(\text{is}|\text{little boy})$$

[단점 1] sparsity problem 여전히 존재

- 물론 모든 이전 단어를 보는 것보다 일부 단어만 보는 것이 현실적으로 카운트할 수 있는 확률을 높이지만, 그럼에도 불구하고 여전히 n-gram에 대한 희소 문제가 존재한다.

[단점 2] n은 선택하는 것은 trade-off 문제

- N이 커질수록 sparsity problem 문제가 심각해지고, N이 작아질수록 현실의 확률 분포와 멀어진다.

=> 적절한 n을 선택하는 것이 중요(정확도를 높이려면 n은 최대 5를 넘게 잡아서 안된다고 권장)

Unit 03 | Language Model

Perplexity(PPL)

- 언어 모델을 평가하기 위한 평가 지표
- “perplexed”는 “헛갈리는”과 유사한 의미를 가지므로, 여기서 PPL은 “헛갈리는 정도 ” 로 받아들일 수 있다.
- PPL은 “낮을수록” 언어 모델의 성능이 높다는 것을 의미한다.
- 문장 W 의 길이를 N 이라고 할 때, PPL은 다음과 같이 쓸 수 있다.

$$PPL(W) = \sqrt[N]{\frac{1}{P(w_1, w_2, w_3, \dots, w_N)}} = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_1, w_2, \dots, w_{i-1})}}$$

N-gram을 적용할 경우(bi-gram인 경우),

$$PPL(W) = \sqrt[N]{\frac{1}{\prod_{i=1}^N P(w_i | w_{i-1})}}$$

Unit 03 | Language Model

Perplexity(PPL)

- PPL은 선택할 수 있는 가능한 경우의 수를 의미하는 분기 계수(branching factor)다.
- PPL은 이 언어 모델이 특정 시점에 평균적으로 몇 개의 선택지를 가지고 고민하는지를 의미한다.

$$PPL(W) = P(w_1, w_2, w_3, \dots, w_N)^{-\frac{1}{N}} = \left(\frac{1}{10}\right)^{-\frac{1}{N}} = \frac{1}{10}^{-1} = 10$$

- 하지만, PPL값이 낮다는 것 == 테스트 데이터 상에서 높은 정확도를 보인다는 것이지, 사람이 보기에 좋은 언어 모델이라는 것을 의미하지 않음!
- PPL로 언어 모델 비교 시, 정량적으로 양이 많고, 도메인에 알맞은 동일한 테스트 데이터를 사용해야 신뢰도가 높음

-	Unigram	Bigram	Trigram
Perplexity	962	170	109

Contents

Unit 01 | Introduction

Unit 02 | Text preprocessing

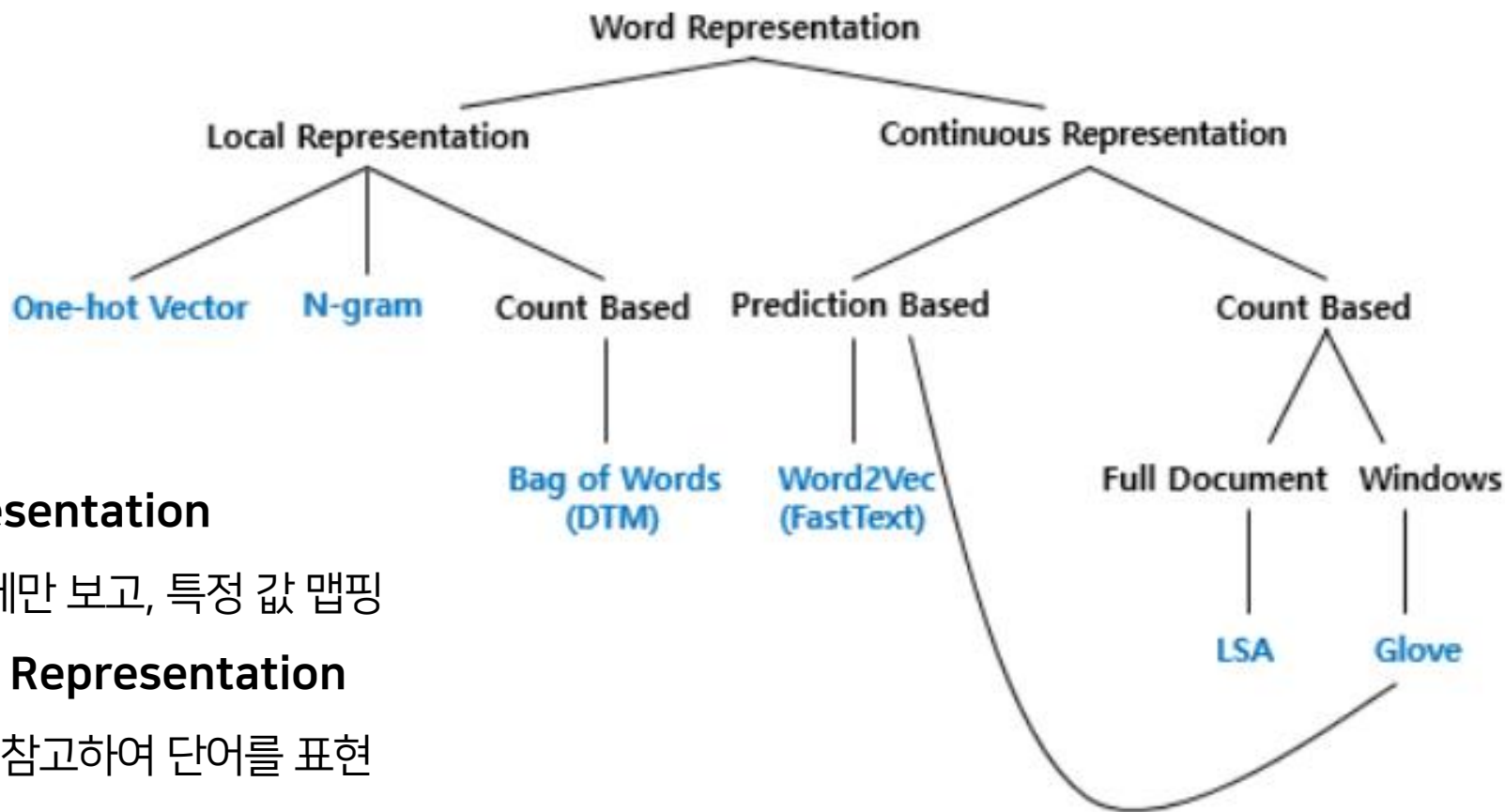
Unit 03 | LM(Language Model)

Unit 04 | 단어 의미 & 단어 표현

Unit 05 | Word Embedding

Unit 04 | 단어 의미 & 단어 표현

어떻게 단어를 “표현”할 것인가?



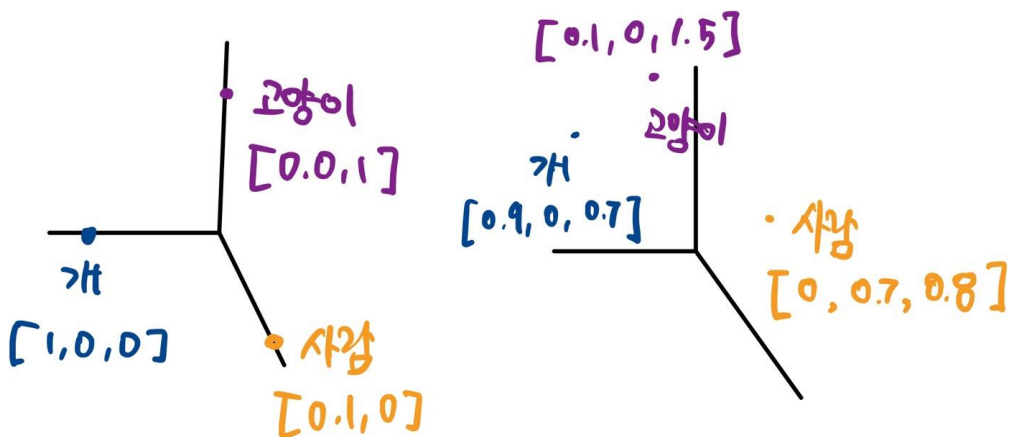
- **Local Representation**
: 해당 단어 자체만 보고, 특정 값 매핑
- **Continuous Representation**
: 주변 단어를 참고하여 단어를 표현

Unit 04 | 단어 의미 & 단어 표현

어떻게 단어를 “표현”할 것인가?

• 희소 표현(Sparse Representation)

- 벡터 또는 행렬의 값이 대부분 0으로 표현되는 방법
- 각 단어 벡터 간 유의미한 유사성을 표현할 수 없음
- One-hot vector $[0, 0, 0, 1, 0, \dots, 0]$



• 분산 표현(distributed Representation)

- 단어의 의미를 다차원 공간에 벡터화하는 방법
- 분포 가설 가정 하에 만들어진 표현 방법
→ 비슷한 문맥에서 등장하는 단어들은 비슷한 의미를 가진다
- 벡터의 차원이 단어 집합 크기일 필요 X
=> 상대적으로 저차원
- 이를 이용해 단어 벡터 간 유의미한 유사도 계산 가능

Unit 04 | 단어 의미 & 단어 표현

어떻게 의미를 “벡터”에 넣을 것인가?

[Ans] 자연어의 통계적 패턴 정보를 벡터에 넣자

- [Why] 자연어의 의미를 해당 언어 화자들의 사용에서 드러나기 때문!

	Bow 가정	언어 모델	분포 가설
내용	빈도	순서	맥락
대표 통계	TF-IDF	-	PMI
대표 모델	-	GPT	Word2Vec

← 통계 기반

신경망 기반 →

- BoW 가정 : 순서 X 빈도 O
 - 저자의 의도는 단어 사용 여부/빈도에서 드러난다.
- Language Model : 순서 O
 - 주어진 단어 시퀀스가 얼마나 자연스러운지 정도를 확인해야 한다.
- 분포 가설 : 주변 단어
 - 단어의 의미를 주변 context를 통해 유추 가능하다.

Unit 04 | 단어 의미 & 단어 표현

문서 단어 행렬(Document-Term Matrix, DTM)

- 다수의 문서에서 등장하는 각 단어들의 빈도를 행렬로 표현한 것

Ex) 문서 1 : 먹고 싶은 사과 문서 3 : 길고 노란 바나나 바나나

문서 2 : 먹고 싶은 바나나 문서 4 : 저는 과일이 좋아요

-	과일이	길고	노란	먹고	바나나	사과	싶은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

Unit 04 | 단어 의미 & 단어 표현

문서 단어 행렬(Document-Term Matrix, DTM)

[단점 1] 희소 표현(Sparse representation)

- DTM도 one-hot 벡터와 마찬가지로 대부분의 값이 0이므로 공간 낭비 + 계산량 증가
- DTM에서 각 행을 문서 벡터라고 할 경우, 각 문서 벡터의 차원 == 전체 단어 집합 크기
- 물론 구두점, 빈도수가 낮은 단어, 어간이나 표제어 추출을 통해 단어를 정규화해 단어 집합의 크기를 어느 정도 줄일 순 있음.

[단점 2] 단순 빈도 수 기반 접근

- 여러 문서에 등장하는 모든 단어에 대해 빈도 표기하는 방법은 한계를 가짐
ex) "the"의 경우 어느 문서에서나 등장, 하지만 문서의 the 빈도수가 높다고 해서 유사하다고 판단 불가
- 각 문서에는 중요한 단어와 불필요한 단어가 혼재되어 있음
=> 그럼 중요한 단어에 대해 가중치를 주는 방법은? **TF-IDF**

Unit 04 | 단어 의미 & 단어 표현

TF-IDF(Term Frequency-Inverse Document Frequency)

- 단어의 빈도와 역 문서 빈도를 사용해 DTM 내의 각 단어들마다 중요한 정도를 가중치로 표현
- 우선 DTM을 만든 후, TF-IDF 가중치를 부여해 조정

TF, DF, IDF 정의

- $tf(d,t)$: 특정 문서 d 에서의 특정 단어 t 의 등장 횟수(DTM)
- $df(t)$: 특정 단어 t 가 등장한 문서의 수
=> 특정 단어가 등장한 문서 수에만 관심을 가짐.

ex) 바나나의 df 는 2

- $idf(d,t)$: $df(t)$ 에 반비례 하는 수(n 은 문서의 총 개수)

$$idf(d,t) = \log\left(\frac{n}{1 + df(t)}\right)$$

→ 총 문서의 수가 커질수록, IDF의 값이 기하급수적으로 커지는 걸 방지

→ 특정 단어가 전체 문서에서 등장하지 않을 경우 분모가 0이 되는 상황을 방지하기 위함

Unit 04 | 단어 의미 & 단어 표현

TF-IDF(Term Frequency-Inverse Document Frequency)

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	0.287682	0	0.693147	0.287682	0	0
문서2	0	0	0	0.287682	0.287682	0	0.287682	0	0
문서3	0	0.693147	0.693147	0	0.575364	0	0	0	0
문서4	0.693147	0	0	0	0	0	0	0.693147	0.693147

[TD-IDF]

-	과일이	길고	노란	먹고	바나나	사과	싫은	저는	좋아요
문서1	0	0	0	1	0	1	1	0	0
문서2	0	0	0	1	1	0	1	0	0
문서3	0	1	1	0	2	0	0	0	0
문서4	1	0	0	0	0	0	0	1	1

[DTM]

$$idf(d,t) = log(\frac{n}{1 + df(t)})$$

단어	IDF(역 문서 빈도)
과일이	ln(4/(1+1)) = 0.693147
길고	ln(4/(1+1)) = 0.693147
노란	ln(4/(1+1)) = 0.693147
먹고	ln(4/(2+1)) = 0.287682
바나나	ln(4/(2+1)) = 0.287682
사과	ln(4/(1+1)) = 0.693147
싫은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

[IDF]

Contents

Unit 01 | Introduction

Unit 02 | Text preprocessing

Unit 03 | LM(Language Model)

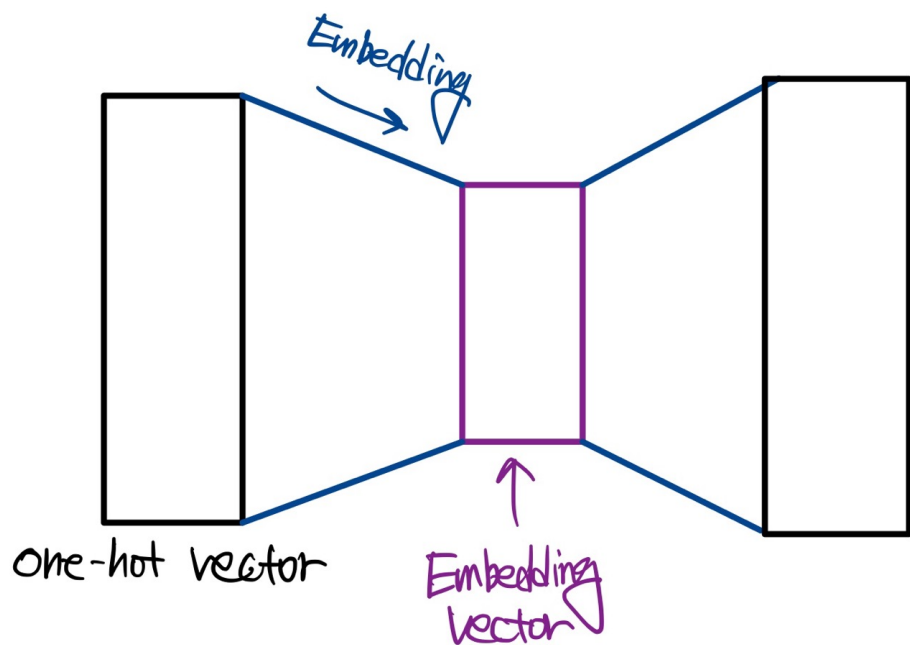
Unit 04 | 단어 의미 & 단어 표현

Unit 05 | Word Embedding

Unit 05 | Word Embedding

워드 임베딩(Word Embedding)

- 단어를 벡터로 표현하는 방법으로, 단어를 밀집 표현으로 변환
- 이 때, 무슨 정보를 이용해 단어 의미를 표현?

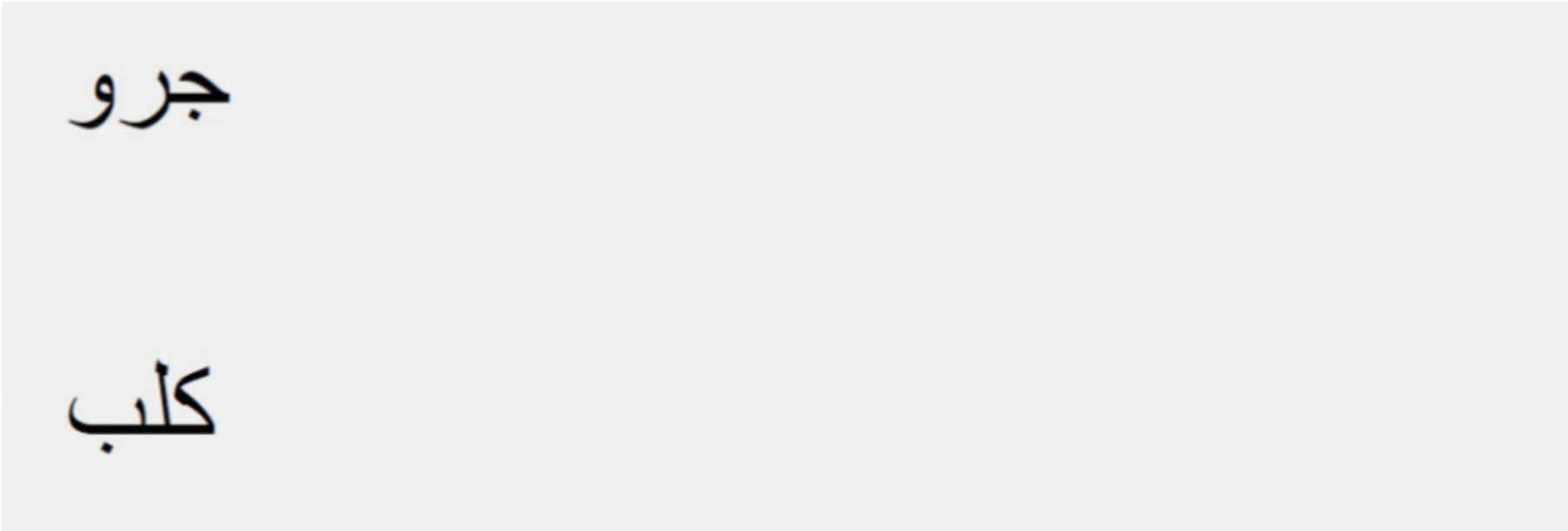


주변 단어	➡	Word2Vec
주변 단어 + n-gram	➡	FastText
주변 단어 + 분포	➡	Glove

Unit 05 | Word Embedding

Word2Vec

- 한 단어의 주변 단어를 통해, 그 단어의 의미를 파악



컴퓨터에게 자연어는 "기호"일 뿐

Unit 05 | Word Embedding

Word2Vec

- 한 단어의 주변 단어를 통해, 그 단어의 의미를 파악

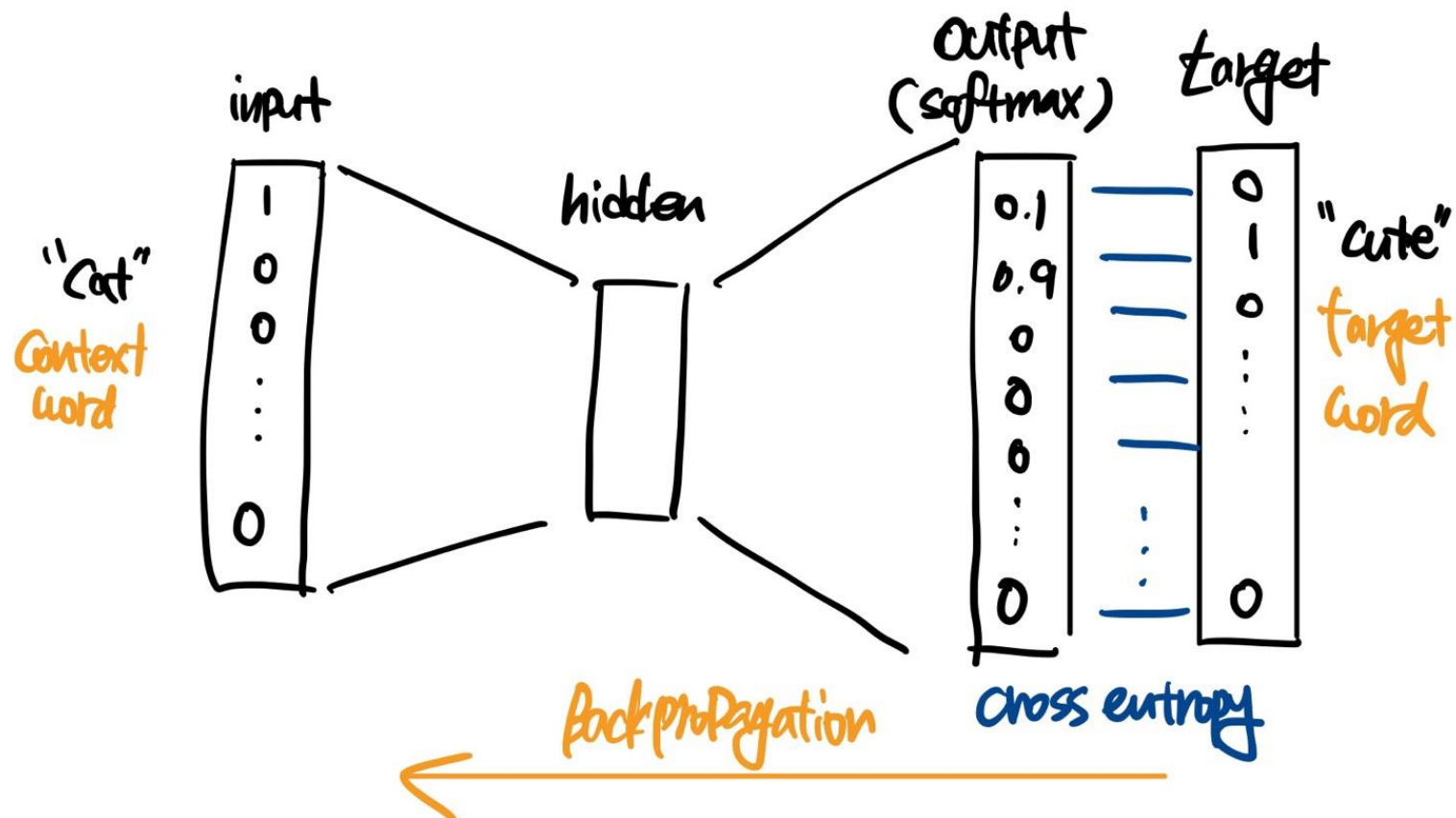
جرو
(개) 가 멍멍! 하고 짖었다.

كلب
(강아지) 가 멍멍! 하고 짖었다.

각 단어의 의미를 모르겠지만, 주변 단어가 비슷하니 의미도 비슷할 것이다!

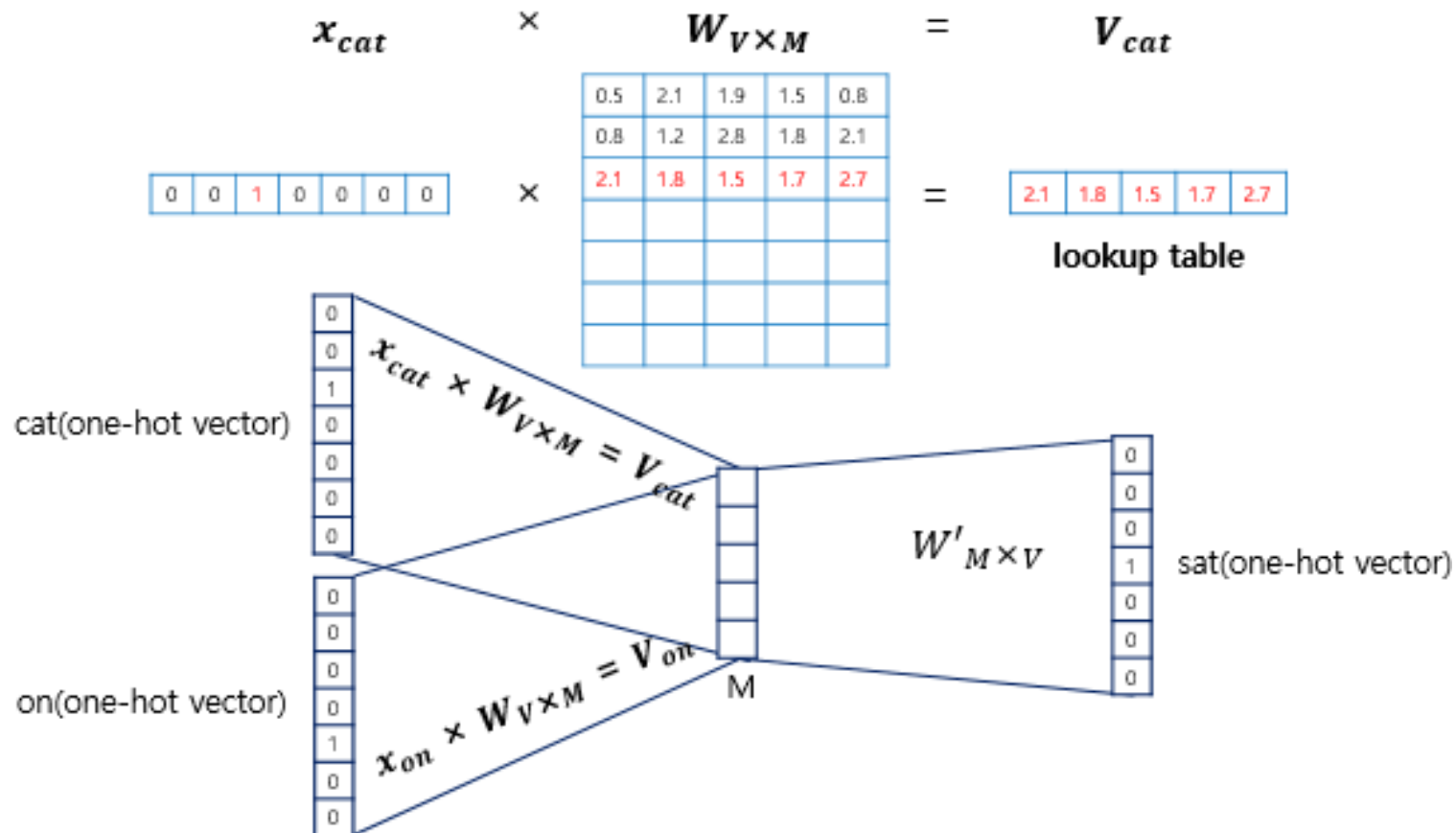
Unit 05 | Word Embedding

Word2Vec



Unit 05 | Word Embedding

Word2Vec – CBOW

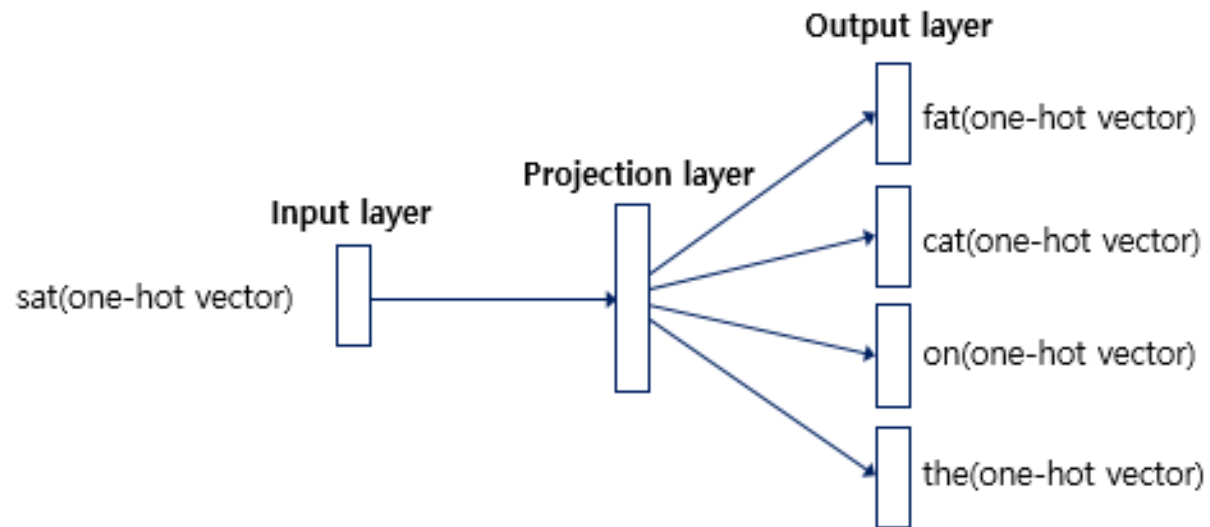
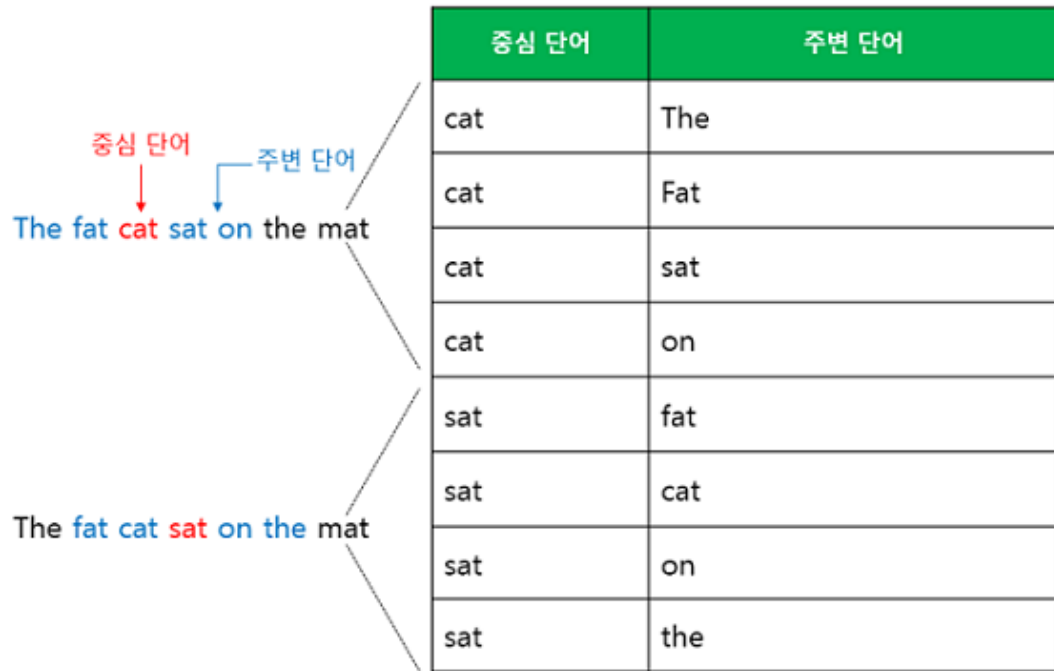


["The cat sat on the mat"]

- window = 1이라 가정
- 중간 단어를 예측하기 위해 주변 단어 벡터를 평균 냄
- W와 W'는 다른 행렬(전치 X)
- W가 결국 임베딩 벡터가 됨

Unit 05 | Word Embedding

Word2Vec – Skip-gram



- 보통 Skip-gram이 CBOW보다 성능이 좋음
 - => 역전파 시, 더 많은 정보를 받음
 - => 타겟 단어 1개 당 더 많은 데이터

Unit 05 | Word Embedding

Word2Vec – Skip-gram


$$J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$

Let's derive gradient for center word together

For one example window and one example outside word:

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

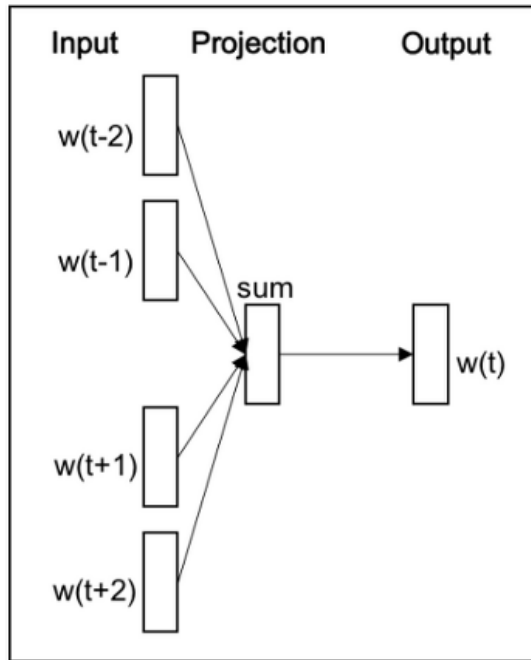
Unit 05 | Word Embedding

- $J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t).$
- $p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)}$
- $\frac{\partial}{\partial v_c} \log \frac{\exp(u_o^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} = \frac{\partial}{\partial v_c} \log \exp(u_o^T v_c) - \frac{\partial}{\partial v_c} \log \sum_{w=1}^V \exp(u_w^T v_c)$

- $\frac{\partial}{\partial v_c} \log \sum_{w=1}^V \exp(u_w^T v_c) = \frac{1}{\sum_{w=1}^V \exp(u_w^T v_c)} \times \frac{\partial}{\partial v_c} \sum_{w=1}^V \exp(u_w^T v_c)$
- $\frac{\partial}{\partial v_c} \sum_{w=1}^V \exp(u_w^T v_c) = \sum_{w=1}^V \frac{\partial}{\partial v_c} \exp(u_w^T v_c) = \sum_{w=1}^V \exp(u_w^T v_c) u_w$
- $\frac{\partial}{\partial v_c} \log p(o|c) = u_o - \sum_{x=1}^V \frac{\exp(u_x^T v_c)}{\sum_{w=1}^V \exp(u_w^T v_c)} u_x = u_o - \sum_{x=1}^V p(x|c) u_x$
 $= \text{observed} - \text{expected}$

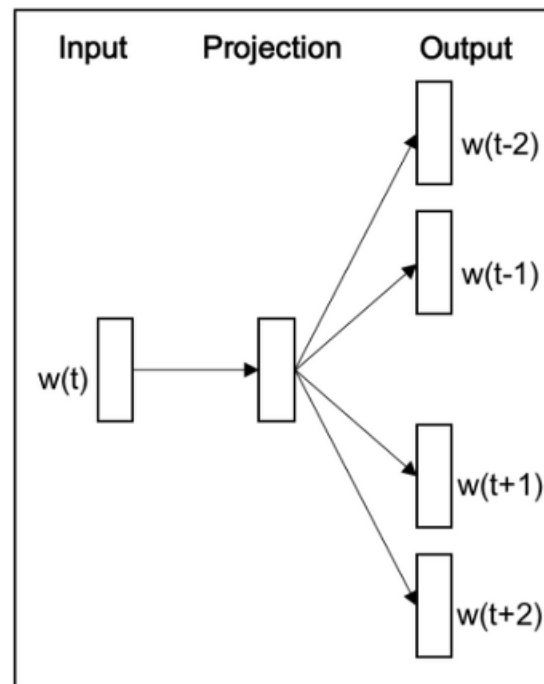
Unit 05 | Word Embedding

Word2Vec – CBOW & Skip-gram

CBOW



Skip-Gram

**[장점]**

- 단어 간 유사도 측정 가능, 관계 파악 가능
- 벡터 연산을 통해 추론 가능

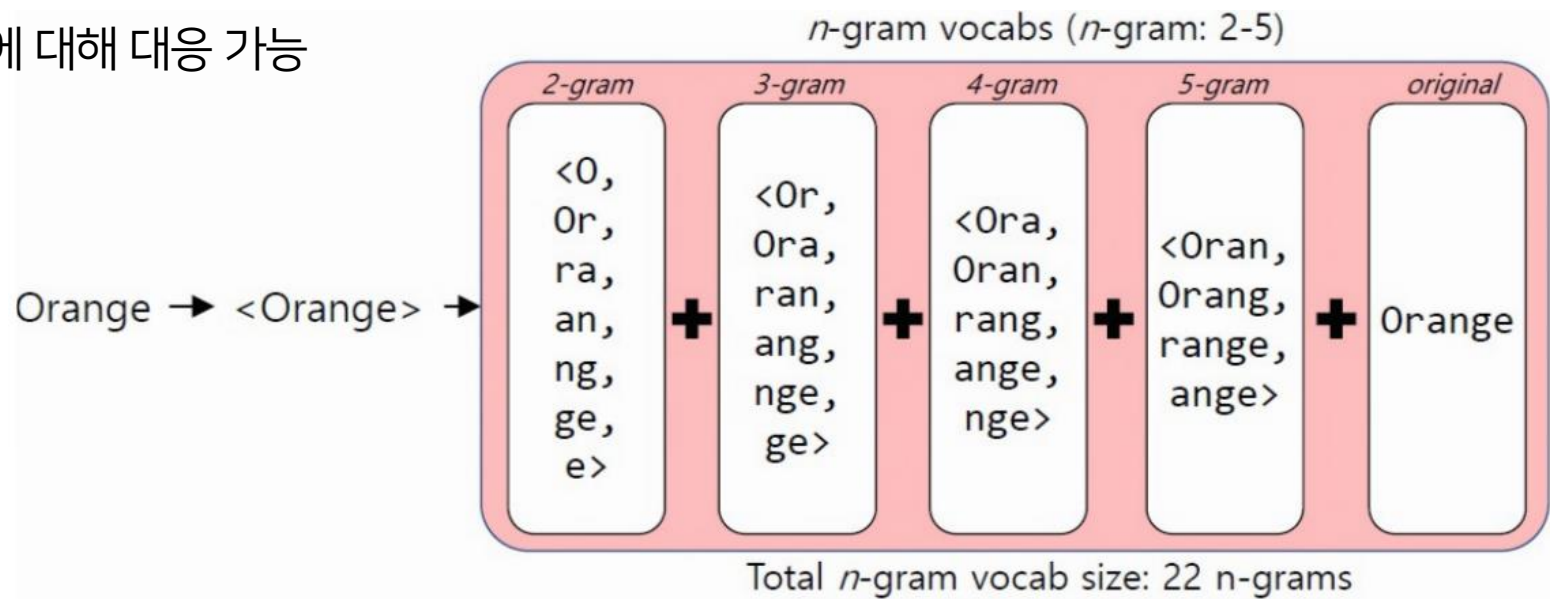
[단점]

- 단어의 subword information 무시
- Out of Vocabulary

Unit 05 | Word Embedding

FastText

- Word2Vec과 유사한 방식으로 학습
- **[차이]** 단어를 n-gram으로 나누어 학습
- 이때, n-gram으로 나누어진 단어는 사전에 들어가지 않고, 별도의 n-gram vector 생성
- **[장점]** OOV, rare word에 대해 대응 가능



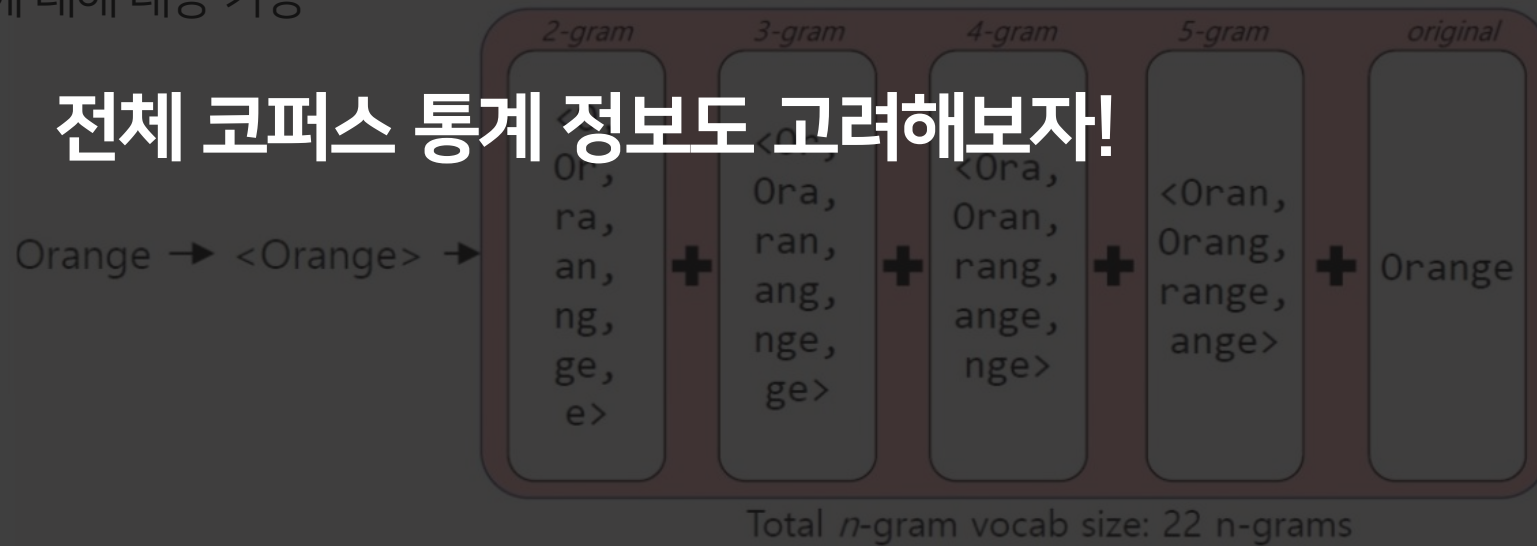
Unit 05 | Word Embedding

FastText

- Word2Vec과 유사한 방식으로 학습
- **[차이]** 단어를 n-gram으로 나누어 학습
- 이때, n-gram으로 나누어진 단어는 사전에 들어가지 않고, 별도의 n-gram vector 생성
- **[장점]** OOV, rare word에 대해 대응 가능

하지만, 주변 단어만 반영해도 괜찮을까?

전체 코퍼스 통계 정보도 고려해보자!



Unit 05 | Word Embedding

Glove

[기존 연구의 한계]

- TF-IDF의 경우 카운트 기반으로 전체 통계 정보를 고려하지만, 단어의 의미 유추 작업의 성능은 낮음
- Word2Vec은 예측 기반으로 단어 간 유추 작업엔 상대적으로 뛰어나지만, 코퍼스의 전체적인 통계 정보를 반영하지 X

⇒ 카운트 기반과 예측 기반 방법론을 모두 사용하자!

[윈도우 기반 동시 등장 행렬]

- I like deep learning
- I like NLP
- I enjoy flying
- N은 1이라 가정

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

Unit 05 | Word Embedding

Glove

[동시 등장 확률]

- $P(k|i)$ 는 동시 등장 행렬로부터 특정 단어 i 의 전체 등장 횟수를 카운트하고, 특정 단어 i 가 등장했을 때 어떤 단어 k 가 등장한 횟수를 카운트하여 계산
- i 를 중심 단어, k 를 주변 단어라 할 때, 앞에서 배운 동시 등장 행렬에서 중심단어 i 행의 모든 값을 더한 것이 분모가 되고, i 행 k 열의 값을 분자가 된다.

동시 등장 확률과 크기 관계 비(ratio)	k=solid	k=gas	k=water	k=fasion
$P(k ice)$	0.00019	0.000066	0.003	0.000017
$P(k steam)$	0.000022	0.00078	0.0022	0.000018
$P(k ice) / P(k steam)$	8.9	0.085	1.36	0.96

Unit 05 | Word Embedding

Glove

- Word2Vec이 전체 코퍼스의 정보를 담지 못한다는 문제 보완
- 두 단어의 유사도에 통계 정보가 반영됨
- 분류 -> 회귀 문제로 전환
- **[목적 함수]** $\text{dot product}(w_i \bar{w}_k) \approx \log P(k | i) = \log P_{ik}$

임베딩 된 중심 단어와 주변 단어 벡터의 내적 == 전체 코퍼스에서의 동시 등장 확률

$$\text{Loss function} = \sum_{m,n=1}^V f(X_{mn})(w_m^T \bar{w}_n + b_m + \bar{b}_n - \log X_{mn})^2$$

$f(x) = \min(1, (x/x_{\max})^{3/4})$ → 특정 단어가 지나치게 빈도수가 높아서 X_{mn} 이 튀는 현상을 방지하기 위해 추가한 함수

Unit 05 | Word Embedding

Glove

$$\text{dot product}(w_i \tilde{w}_k) \approx \log P(k | i) = \log P_{ik}$$

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

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

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

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

조건을 만족하는 F는 지수 함수이므로 다음과 같이 식을 바꾸면,

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

$$w_i^T \tilde{w}_k = \log P_{ik} = \log X_{ik} - \log X_i$$

이 때, w_i 와 w_k 의 내적은 순서가 바뀌어도 값이 같도록 일종의 편향 b 를 추가해 표현

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

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

Unit 05 | Word Embedding

Glove

- Word2Vec이 전체 코퍼스의 정보를 담지 못한다는 문제 보완
- 두 단어의 유사도에 통계 정보가 반영됨
- 분류 -> 회귀 문제로 전환
- **[목적 함수]** $\text{dot product}(w_i \bar{w}_k) \approx \log P(k | i) = \log P_{ik}$

임베딩 된 중심 단어와 주변 단어 벡터의 내적 == 전체 코퍼스에서의 동시 등장 확률

$$\text{Loss function} = \sum_{m,n=1}^V f(X_{mn})(w_m^T \bar{w}_n + b_m + \bar{b}_n - \log X_{mn})^2$$

$f(x) = \min(1, (x/x_{\max})^{3/4})$ → 특정 단어가 지나치게 빈도수가 높아서 X_{mn} 이 튀는 현상을 방지하기 위해 추가한 함수

Unit 05 | Word Embedding

Glove

- Word2Vec이 전체 코퍼스의 정보를 담지 못한다는 문제 보완
- 두 단어의 유사도에 통계 정보가 반영됨
- 분류 -> 회귀 문제로 전환

- **하지만, 주변 단어 및 코퍼스 통계 정보만 고려하는게 맞나?**
- **[목적 함수]** $dot\ product(w_i\ \bar{w}_k) \approx \log P(k\ |\ i) = \log P_{ik}$

임베딩 된 중심 단어와 주변 단어 벡터의 내적 = 전체 코퍼스에서의 동시 등장 확률

문맥에 맞춰서 임베딩 하자!

$$Loss\ function = \sum_{m,n=1}^V f(X_{mn})(w_m^T\bar{w}_n + b_m + \bar{b}_n - \log X_{mn})^2$$

$f(x) = min(1, (x/x_{max})^{3/4})$ → 특정 단어가 지나치게 빈도수가 높아서 X_mn이 튀는 현상을 방지하기 위해 추가한 함수

Unit 05 | Word Embedding

ELMo



- 워드 임베딩 시 문맥을 반영하여 임베딩 하자 → Contextualized Word Embedding
- 사전 훈련된 언어 모델(Pre-trained language model)을 사용

싱가포르에서 온 준원이는 말했다. 오 세상에, **눈**이 예쁘게 내리고 있어!

민진이는 낙타를 보며 말했다. 와, 난 세상에서 이렇게 예쁜 **눈**을 본 적이 없어!

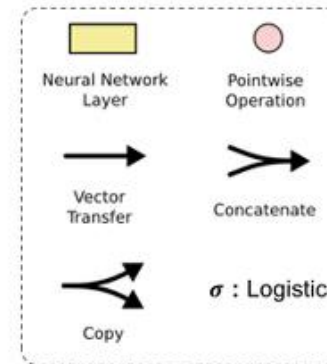
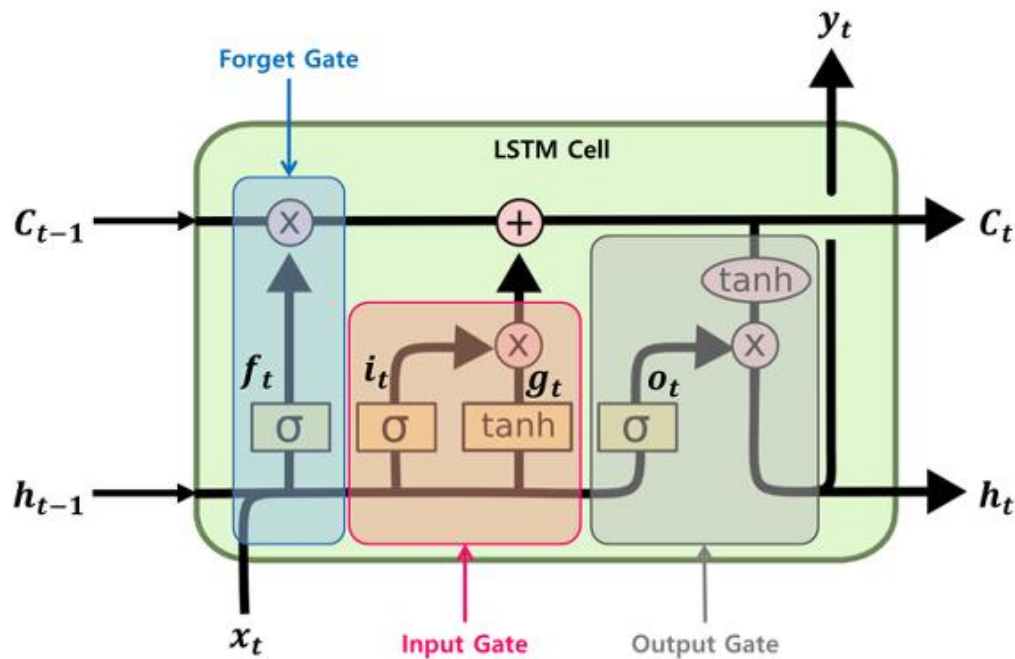
→ 주변 단어만 고려한 임베딩 할 경우, " 눈"의 문맥 속 의미를 제대로 반영하지 못함

Unit 05 | Word Embedding

ELMo



[LSTM]



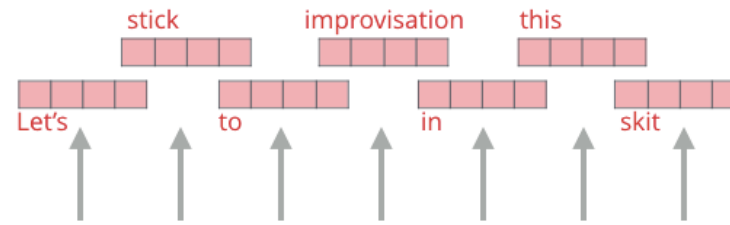
- Forget gate(f_t) : 과거 정보를 얼마나 유지할 것인가
- Input gate (i_t) : 새로 입력된 정보는 얼마나 활용할 것인가
- Output gate(o_t) : 두 정보를 계산하여 나온 출력 정보를 얼마만큼 넘겨줄 것인가?

Unit 05 | Word Embedding

ELMo



ELMo
Embeddings



Words to embed

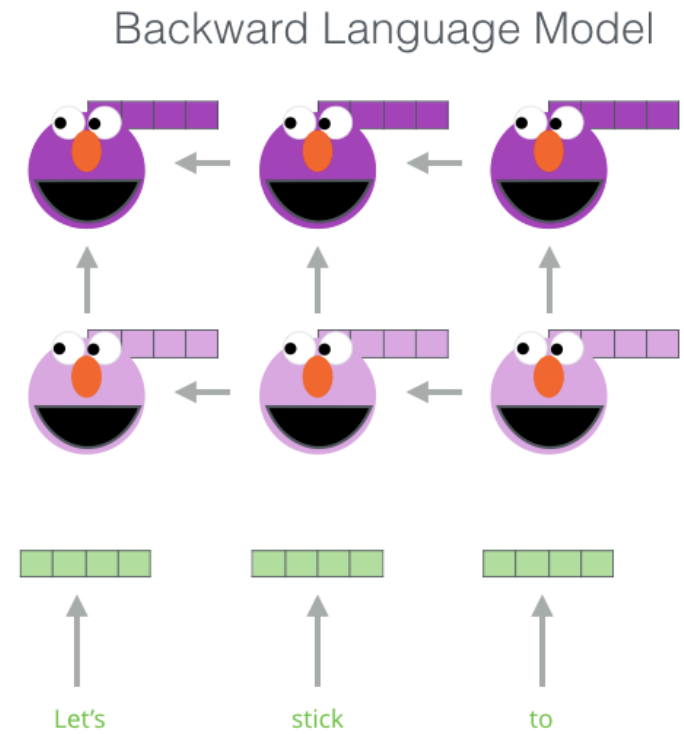
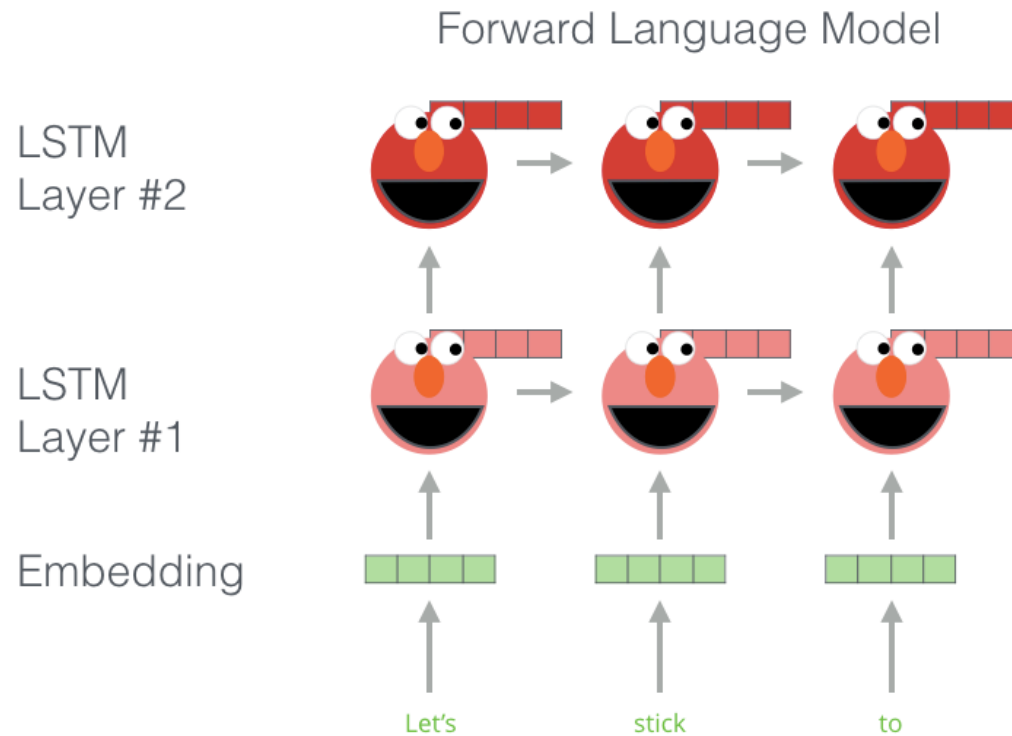


Unit 05 | Word Embedding

ELMo



Embedding of “stick” in “Let’s stick to” - Step #1



Unit 05 | Word Embedding

ELMo

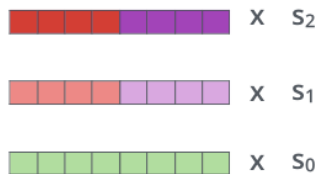


Embedding of “stick” in “Let’s stick to” - Step #2

1- Concatenate hidden layers



2- Multiply each vector by a weight based on the task

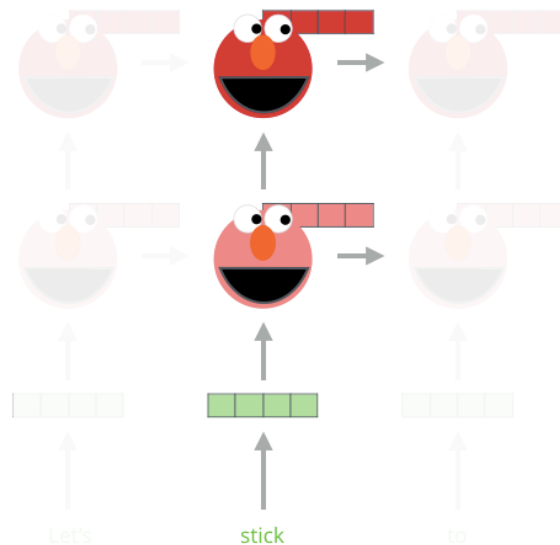


3- Sum the (now weighted) vectors

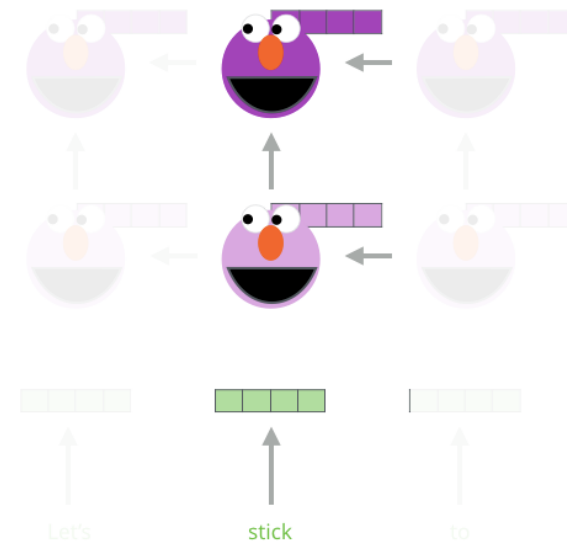


ELMo embedding of “stick” for this task in this context

Forward Language Model



Backward Language Model



$$\text{ELMo}_k^{task} = E(R_k; \Theta^{task}) = \gamma^{task} \sum_{j=0}^L s_j^{task} \mathbf{h}_{k,j}^{LM}$$

Unit 05 | Word Embedding

임베딩으로 할 수 있는 것

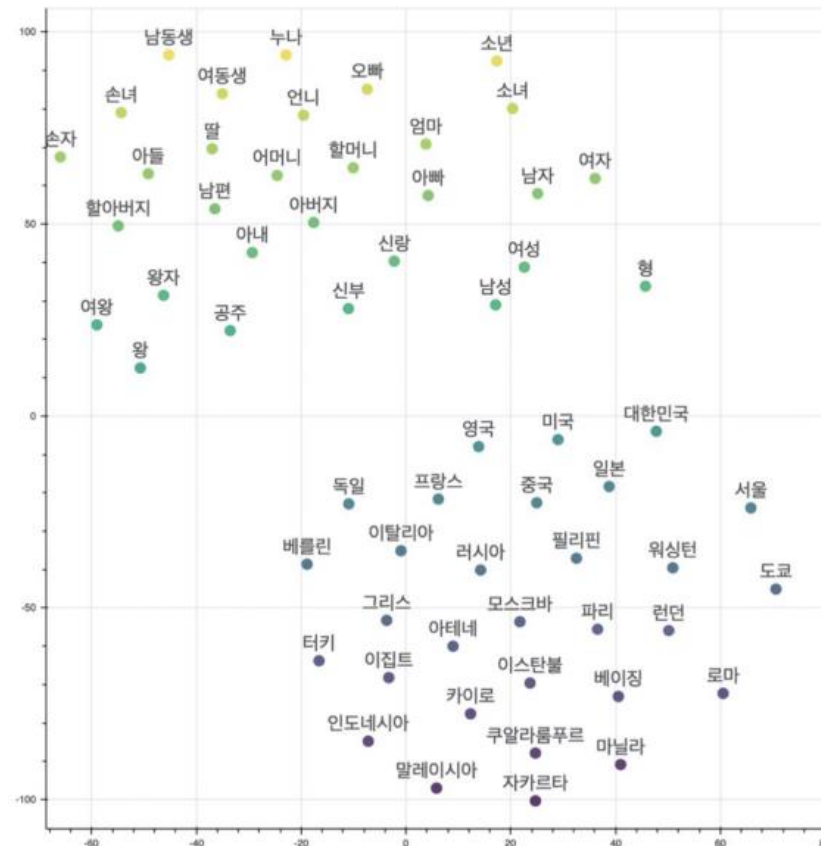
[관련도/유사도 계산]

희망	절망	학교	학생	가족	자동차
소망	체념	초등	대학생	아이	승용차
행복	고뇌	중학교	대학원생	부모	상용차
희망찬	절망감	고등학교	고학생	편부모	트럭
꿈	상실감	야학교	교직원	고달픈	대형트럭
열망	번민	중학	학부모	사랑	모터사이클

Unit 05 | Word Embedding

임베딩으로 할 수 있는 것

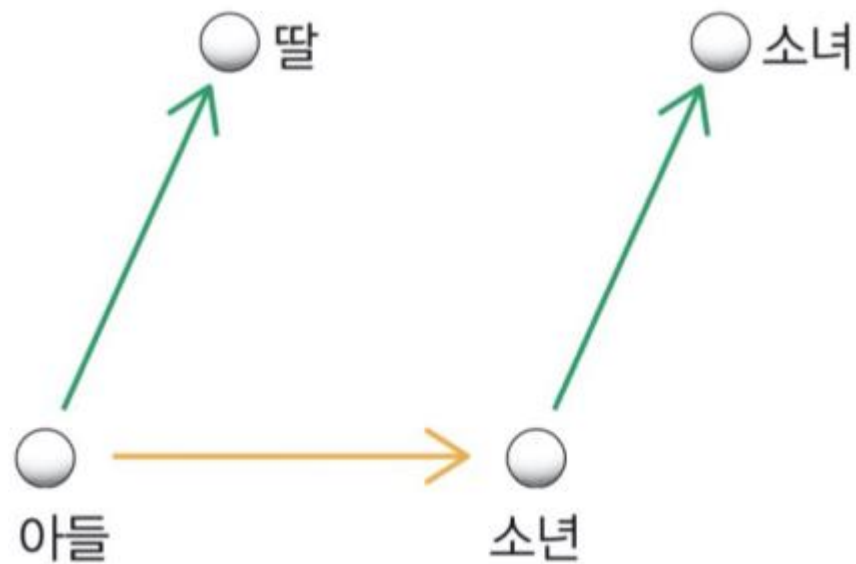
[시각화]



Unit 05 | Word Embedding

임베딩으로 할 수 있는 것

[벡터 연산(유추 평가) : 아들 - 딸 + 소녀 = 소년]



Unit 05 | Word Embedding

임베딩으로 할 수 있는 것

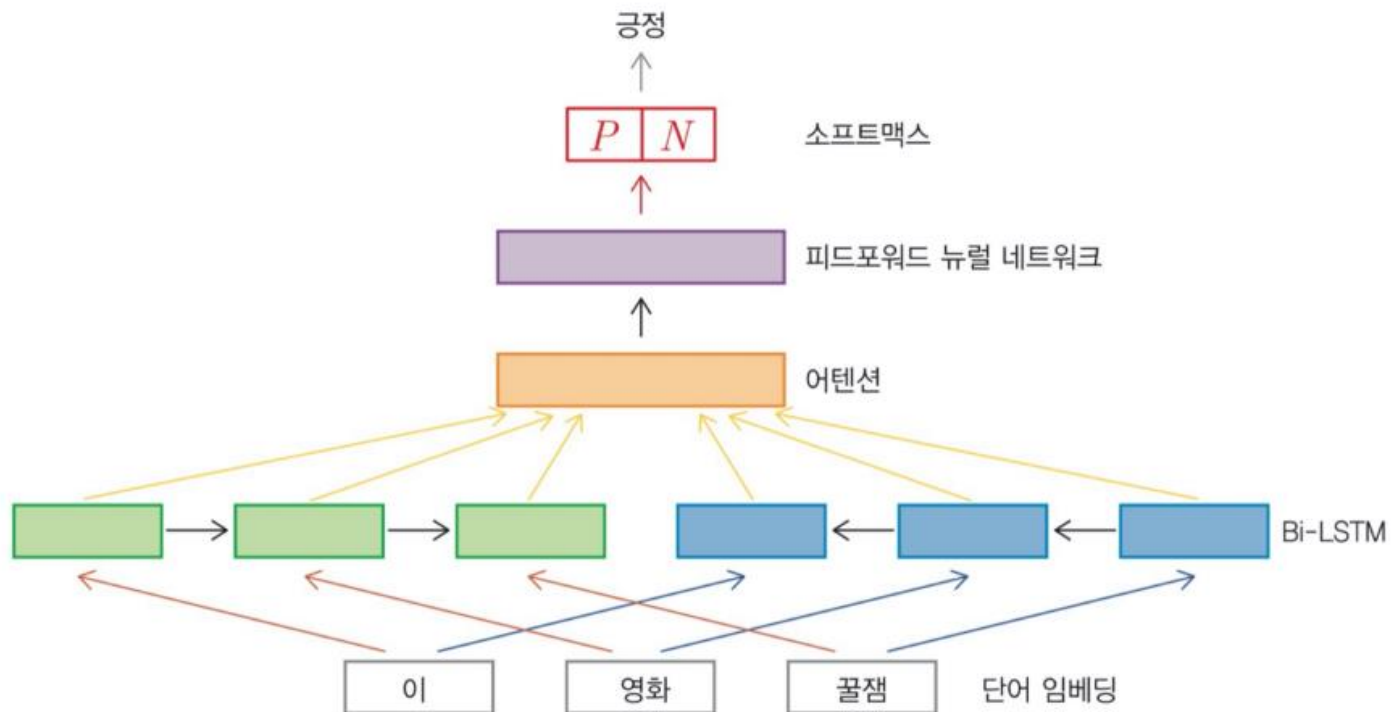
[벡터 연산(유추 평가) : 아들 - 딸 + 소녀 = 소년]

단어1	단어2	단어3	결과
아들	딸	소년	소녀
아들	딸	아빠	엄마
아들	딸	남성	여성
남동생	여동생	소년	소녀
남동생	여동생	아빠	엄마
남동생	여동생	남성	여성
신랑	신부	왕	여왕
신랑	신부	손자	손녀
신랑	신부	아빠	엄마

Unit 05 | Word Embedding

임베딩으로 할 수 있는 것

[전이학습(Transfer learning) : 다른 모델의 입력값]



실습

과제 : NLP 맛보기

Step1. 데이터 확인

Step2. Tokenizing(불용어 처리, 특수 문자 제거 등의 전처리 포함)

Step3. 임베딩(One-hot encoding, CBOW, Skip-gram, Glove, FastText 등)

Step4. 유의미한 해석 도출 (유사도, wordcloud, 이진 분류 모델, 그래프 해석..)

[주의사항]

- 임베딩 모델 적어도 2개 이상 사용 후, 해석에 따라 가장 좋은 모델을 선택
- 유의미한 해석 도출해보기 - 3가지 이상의 인사이트 도출
- 토크나이저 및 임베딩 모델 선택 과정, 인사이트 해석을 주석으로 달아주세요!

참고 자료

- ToBig's 16기 정규세션 NLP Basic 강의 (조효원님)
- ToBig's 15기 정규세션 NLP Basic 강의 (정세영님)
- https://velog.io/@yuns_u/LSTMLong-Short-Term-Memory%EA%B3%BC-GRUgated-Recurrent-Unit
- <https://nlpinkorean.github.io/illustrated-bert/>
- <https://wikidocs.net/book/2155>
- [한국어를 위한 어휘 임베딩의 개발 -1- \(brunch.co.kr\)](http://brunch.co.kr)
- [1\) 코사인 유사도\(Cosine Similarity\) - 딥 러닝을 이용한 자연어 처리 입문 \(wikidocs.net\)](http://wikidocs.net)
- [딥러닝을 이용한 자연어 처리\(Natural Language Processing with Deep Learning\) – CS224n 강의 노트 1-2 | 솔라리스의 인공지능 연구실 \(solarisailab.com\)](http://solarisailab.com)
- 자연어처리 바이블 (임희석 | 고려대학교 자연어처리연구실 저)
- 한양대학교 김태욱 교수님의 자연어처리 강의



Q & A

들어주셔서 감사합니다.