

KUBIG 25-W  
겨울방학 BASIC STUDY SESSION

# NLP SESSION WEEK2

---

01 Announcement, 복습과제 우수 코드 review

---

02 Text Preprocessing

---

03 basic of Word Representation

---

04 Word2Vec, FastText

---

05 Glove

---

06 예습과제 우수 코드 review, Announcement

---

# 01 Announcement, 우수 복습과제 Review

## 커리큘럼 재공지

주차	학습내용
1주차	OT, DL Reminder
2주차	텍스트 전처리, 워드 임베딩 (Word2Vec, GloVe)
3주차	순환신경망: RNN, LSTM, GRU, ELMo
4주차	Attention, Transformer  <b>*설연휴주간 - 시간조정 or 녹화강의 논문 리뷰 과제 등</b>
5주차	BERT/GPT  <b>세션 후 콘테스트 팀별 중간발표 (진행 과정 보고) * 대면 진행 예정</b>
6주차	LLM 기초: Fine-tuning, RAG
7주차	Toy project  <b>세션 진행 x, 콘테스트 준비 / 질의사항 응답 (자율)</b>

## KUBIG Contest 안내

### Team Building

1팀: 김민재, 김재훈, 강서연, 남동연

2팀: 이예지, 이영서, 이예일

3팀: 이우진, 김유진, 이연호

4팀: 장건호, 기광민, 김정찬



- 주 1회 이상(대면 권장) 회의 진행한 후 회의록 작성
- 5주차에 중간 진행 과정 보고
- 2월 27일 KUBIG Contest에서 결과물 발표

! SLACK 사용시 @이름 사용해서 원활한 소통 바랍니다!!

이예일, 남동연

1주차  
복습과제1,2

---

Pytorch tutorial  
Deep Learning Inference

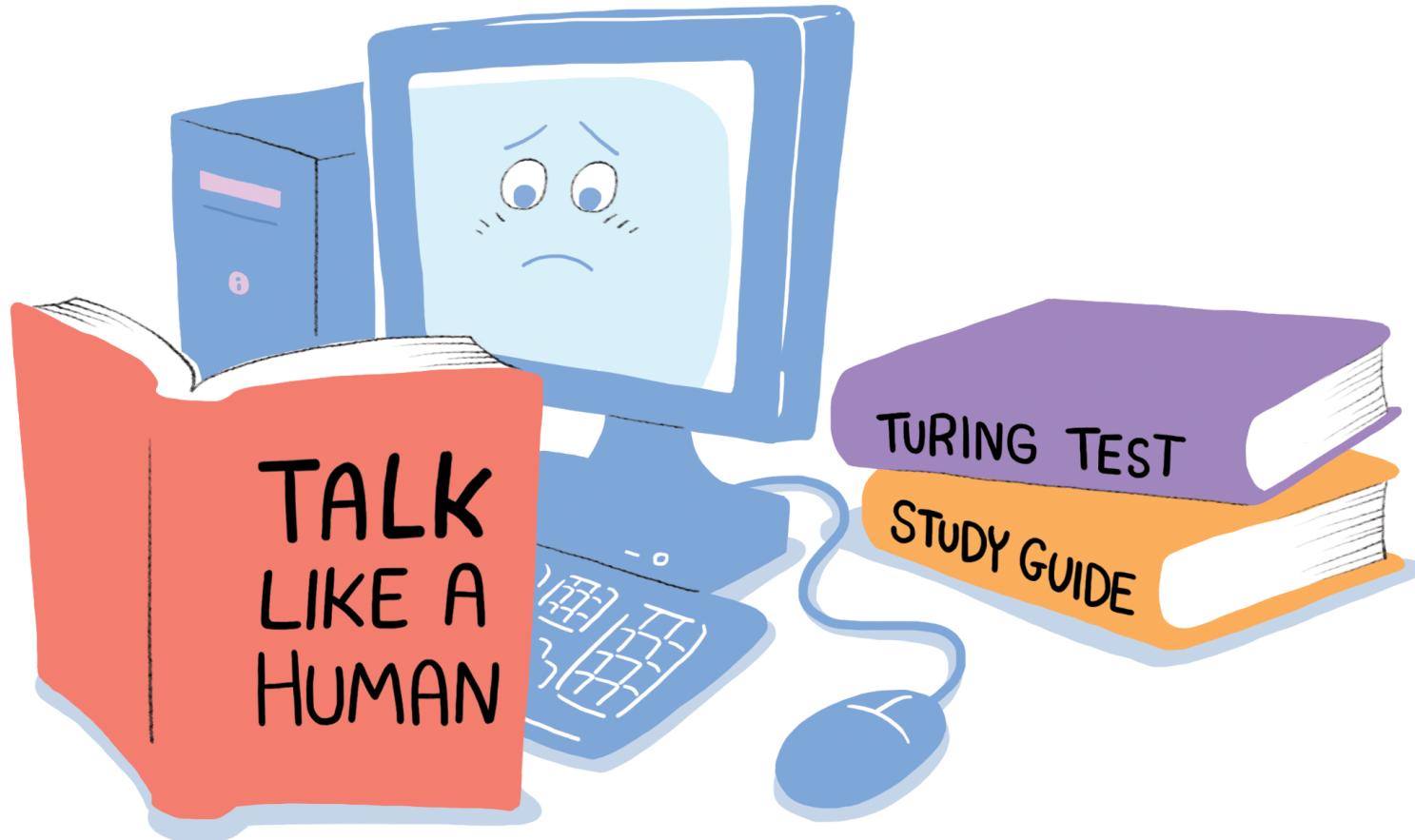
---

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

# 02 Text Preprocessing

텍스트 전처리의 대략적인 파이프라인

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?



컴퓨터는 문자를 어떻게 이해할까?

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

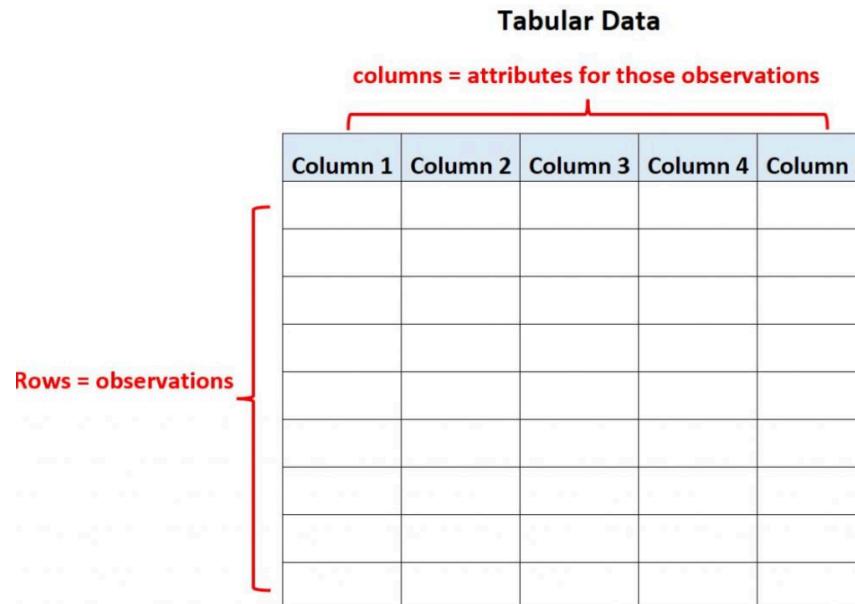
ASCII CODE TABLE																	
10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자	10	HEX	문자
0	0x00	NULL	22	0x16	STN	44	0x2C	,	66	0x42	B	88	0x58	X	110	0x6E	n
1	0x01	SOH	23	0x17	ETB	45	0x2D	-	67	0x43	C	89	0x59	Y	111	0x6F	o
2	0x02	STX	24	0x18	CAN	46	0x2E	.	68	0x44	D	90	0x5A	Z	112	0x70	p
3	0x03	ETX	25	0x19	EM	47	0x2F	/	69	0x45	E	91	0x5B	[	113	0x71	q
4	0x04	EOT	26	0x1A	SUB	48	0x30	0	70	0x46	F	92	0x5C	₩	114	0x72	r
5	0x05	ENQ	27	0x1B	ESC	49	0x31	1	71	0x47	G	93	0x5D	]	115	0x73	s
6	0x06	ACK	28	0x1C	FS	50	0x32	2	72	0x48	H	94	0x5E	^	116	0x74	t
7	0x07	BEL	29	0x1D	GS	51	0x33	3	73	0x49	I	95	0x5F	-	117	0x75	u
8	0x08	BS	30	0x1E	RS	52	0x34	4	74	0x4A	J	96	0x60	:	118	0x76	v
9	0x09	HT	31	0x1F	US	53	0x35	5	75	0x4B	K	97	0x61	a	119	0x77	w
10	0x0A	Wr	32	0x20	SP	54	0x36	6	76	0x4C	L	98	0x62	b	120	0x78	x
11	0x0B	VT	33	0x21	!	55	0x37	7	77	0x4D	M	99	0x63	c	121	0x79	y
12	0x0C	FF	34	0x22	"	56	0x38	8	78	0x4E	N	100	0x64	d	122	0x7A	z
13	0x0D	Wr	35	0x23	#	57	0x39	9	79	0x4F	O	101	0x65	e	123	0x7B	{
14	0x0E	SO	36	0x24	\$	58	0x3A	:	80	0x50	P	102	0x66	f	124	0x7C	
15	0x0F	SI	37	0x25	%	59	0x3B	:	81	0x51	Q	103	0x67	g	125	0x7D	}
16	0x10	DLE	38	0x26	&	60	0x3C	<	82	0x52	R	104	0x68	h	126	0x7E	
17	0x11	DC1	39	0x27	'	61	0x3D	=	83	0x53	S						
18	0x12	DC2	40	0x28	(	62	0x3E	>	84	0x54	T						
19	0x13	DC3	41	0x29	)	63	0x3F	?	85	0x55	U						
20	0x14	DC4	42	0x2A	*	64	0x40	@	86	0x56	V						
21	0x15	NAK	43	0x2B	+	65	0x41	A	87	0x57	W						

유니코드 한  
0xD55C

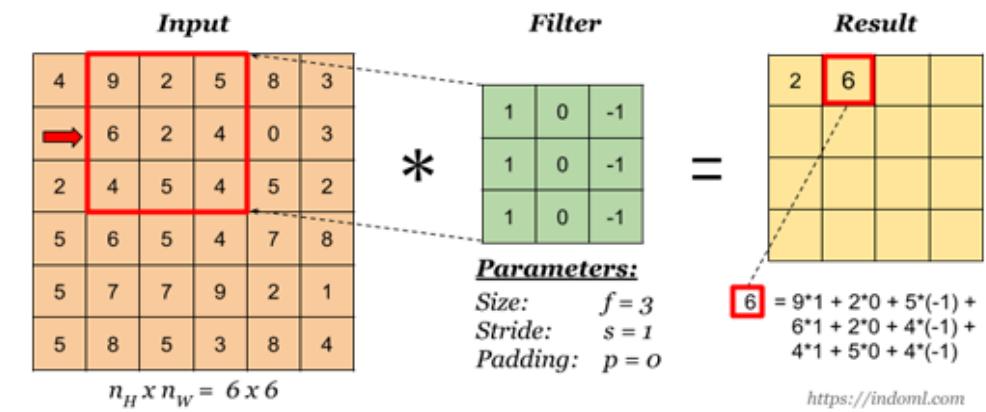
UTF-8 한

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

컴퓨터가 이해할 수 있도록, 비정형 데이터(이미지, 텍스트)를 전산 처리 및 정형 데이터로 변환하는 과정이 필수



Tabular(정형) data: 그 자체로 정형이라 별다른 처리 X

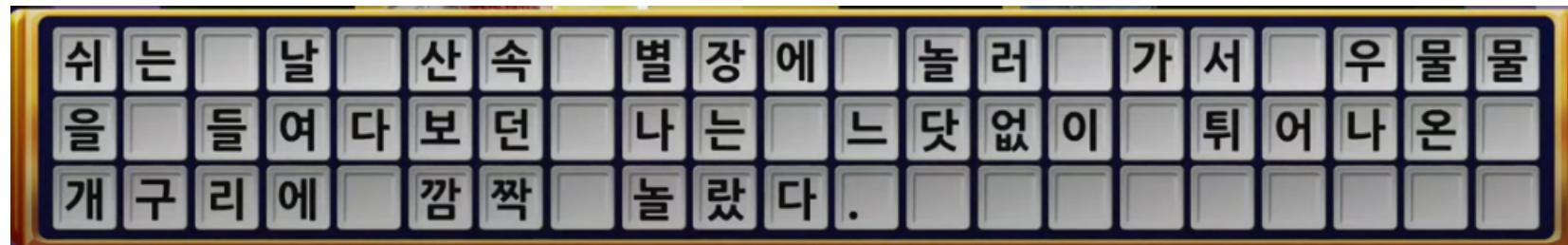


## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?



문장을 이대로 입력?

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?



## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

단어	인덱스	One hot vector
우물물을	1	[1,0,0,0,0,0,0]
들여다보던	2	[0,1,0,0,0,0,0]
나는	3	[0,0,1,0,0,0,0]
느닷없이	4	[0,0,0,1,0,0,0]
튀어나온	5	[0,0,0,0,1,0,0]
개구리에	6	[0,0,0,0,0,1,0]
깜짝	7	[0,0,0,0,0,0,1]
놀랐다	8	[0,0,0,0,0,0,1]

Text data: vector로 표현



단어를 저 상태로 input으로 넣어도 될까?

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

단어	인덱스	One hot vector
우물물	1	[1,0,0,0,0,0,0,0,0,0,0,0,0,0]
을	2	[0,1,0,0,0,0,0,0,0,0,0,0,0,0]
들여다보	3	[0,0,1,0,0,0,0,0,0,0,0,0,0,0]
던	4	[0,0,0,1,0,0,0,0,0,0,0,0,0,0]
나	5	[0,0,0,0,1,0,0,0,0,0,0,0,0,0]
는	6	[0,0,0,0,0,1,0,0,0,0,0,0,0,0]
느닷없이	7	[0,0,0,0,0,0,1,0,0,0,0,0,0,0]
튀어나오	8	[0,0,0,0,0,0,0,1,0,0,0,0,0,0]
ㄴ	9	[0,0,0,0,0,0,0,0,1,0,0,0,0,0]
개구리	10	[0,0,0,0,0,0,0,0,0,1,0,0,0,0]
에	11	[0,0,0,0,0,0,0,0,0,0,1,0,0,0]
깜짝	12	[0,0,0,0,0,0,0,0,0,0,0,1,0,0]
놀라	13	[0,0,0,0,0,0,0,0,0,0,0,0,1,0]
았	14	[0,0,0,0,0,0,0,0,0,0,0,0,0,1]
다	15	[0,0,0,0,0,0,0,0,0,0,0,0,0,1]

한국어 말의 최소 단위: 형태소



형태소 단위로 입력이 효율적인가?

No. 조사 단위 등으로 형태소를 쪼갤 때마다 차원 증가

우물물을/우물물에 는 의미가 다르나, 텍스트 전 처리에서  
을/에 등의 조사는 분석에 큰 의미가 없다

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

단어	인덱스	One hot vector
우물물	1	[1,0,0,0,0]
을	2	[0,1,0,0,0]



단어	인덱스	One hot vector
우물물	1	[1,0,0,0,0]
...	...	...

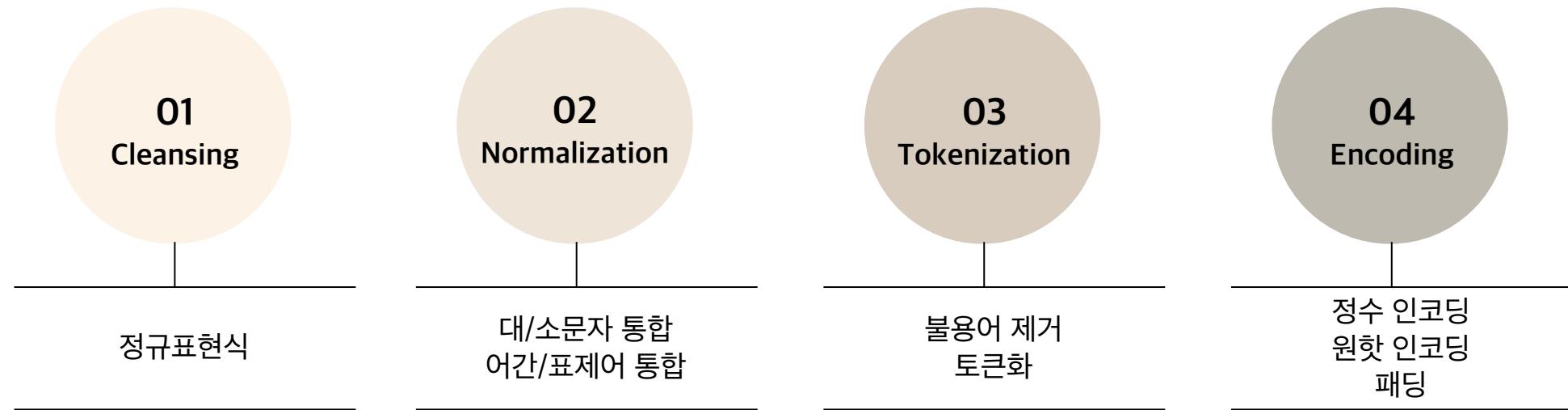
조사, 관사 등 분석에 불필요한 내용 등을 제거하는 과정을 거친다

불용어(Stopword)를 제거한다

**텍스트 전처리는** task에 맞게 단어를 쪼개고, 다듬고, 통일하는 작업이자,  
단어를 가장 **효과적으로** 임베딩시키기 위해 반드시 선행되어야 하는 과정

텍스트 전처리라는 첫 단추가 잘 뛰어지지 않으면,  
뒤이어 배울 모든 작업들이 제대로 동작하지 않음.

## 2-2. 텍스트 전처리 pipeline



## 2-3. Cleansing: 정규표현식(Regex)

**정규 표현식:** 특정한 규칙을 가진 문자열의 집합을 표현하는 데 사용하는 형식 언어

맛이 좋아서 꾸준히 먹고 있습니다.^^  
가성비 갑 제품 + 맛도 다양해서 ㅋㅋㅋ 항상 구매하고 있습니다요! \n\n...  
맛있습니다. 저번에 말차라떼도 먹어봤는데 그것보다 더 나은거 같네요  
5키로는 정말 엄청크네요.\n 평점이 좋아서 스트로베리크림으로 주문했는데 좋습...  
약간 달긴 하지만 물에 잘 녹고 가루날림이 거의없어 좋아요

Ex) 텍스트를 크롤링해왔을 때 생기는 html 태그들은 일정한 **규칙**을 따른다.

정규표현식은 이러한 **규칙**을 잡아내어, 필요한 부분들을 제거하는 데 유용하다.

.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n는 제외)
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)
^	뒤의 문자열로 문자열이 시작됩니다.
\$	앞의 문자열로 문자열이 끝납니다.

{숫자}	숫자만큼 반복합니다.
숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, + 를 이것으로 대체할 수 있습니다.
숫자, }	숫자 이상만큼 반복합니다.
[ ]	대괄호 안의 문자들 중 한 개의 문자와 매치합니다. [amk]라고 한다면 a 또는 m 또는 k 중 하나라도 존재하면 매치를 의미합니다. [a-z] 와 같이 범위를 지정할 수도 있습니다. [a-zA-Z] 는 알파벳 전체를 의미하는 범위이며, 문자열에 알파벳이 존재하면 매치를 의미합니다.
^문자]	해당 문자를 제외한 문자를 매치합니다.
l	A B 와 같이 쓰이며 A 또는 B의 의미를 가집니다.

## 2-3. Cleansing: 정규표현식(Regex)

문자 규칙	설명
<code>\\"</code>	역 슬래쉬 문자 자체를 의미합니다.
<code>\\d</code>	모든 숫자를 의미합니다. <code>[0-9]</code> 와 의미가 동일합니다.
<code>\\D</code>	숫자를 제외한 모든 문자를 의미합니다. <code>[^0-9]</code> 와 의미가 동일합니다.
<code>\\s</code>	공백을 의미합니다. <code>[ \t\n\r\f\v]</code> 와 의미가 동일합니다.
<code>\\S</code>	공백을 제외한 문자를 의미합니다. <code>[^ \t\n\r\f\v]</code> 와 의미가 동일합니다.
<code>\\w</code>	문자 또는 숫자를 의미합니다. <code>[a-zA-Z0-9]</code> 와 의미가 동일합니다.
<code>\\W</code>	문자 또는 숫자가 아닌 문자를 의미합니다. <code>[^a-zA-Z0-9]</code> 와 의미가 동일합니다.

모듈 함수	설명
<code>re.compile()</code>	정규표현식을 컴파일하는 함수입니다. 다시 말해, 파이썬에게 전해주는 역할을 합니다. 찾고자 하는 패턴이 빈번한 경우에는 미리 컴파일해놓고 사용하면 속도와 편의성 면에서 유리합니다.
<code>re.search()</code>	문자열 전체에 대해서 정규표현식과 매치되는지를 검색합니다.
<code>re.match()</code>	문자열의 처음이 정규표현식과 매치되는지를 검색합니다.
<code>re.split()</code>	정규 표현식을 기준으로 문자열을 분리하여 리스트로 리턴합니다.
<code>re.findall()</code>	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열을 찾아서 리스트로 리턴합니다. 만약, 매치되는 문자열이 없다면 빈 리스트가 리턴됩니다.
<code>re.finditer()</code>	문자열에서 정규 표현식과 매치되는 모든 경우의 문자열에 대한 이터레이터 객체를 리턴합니다.
<code>re.sub()</code>	문자열에서 정규 표현식과 일치하는 부분에 대해서 다른 문자열로 대체합니다.

### 경우에 따라 정규표현식이 어떻게 표현될까?

- 숫자만 표현하고 싶을 때 => `[0-9]` or `\d`
- 영어, 숫자를 제외하고 그 외(한글, 기호)만 표현하고 싶을 때 => `[^a-zA-Z0-9]`
- 한글이 아닌 것들만 표현하고 싶을 때 => `[ㄱ-ㅎ가-힣]`
- 한글 단어만 표현하고 싶을 때 => `[가-힣]`
- 단어 형태를 유지하고 싶을 때 => `[a-zA-Z]+`

## 2-3. Cleansing: 정규표현식(Regex)

```
string = "Jeju MT was fantastic. 이번 엠티도 기대된다! "
pattern = re.compile('[a-z]+') # []+ : 영문, 소문자 string이 하나 이상 등장
print(pattern.findall(string))
```

✓ 0.0s

Python

```
['eju', 'was', 'fantastic']
```

```
string = "Jeju MT was fantastic. 이번 엠티도 기대된다! "
pattern = re.compile('[a-z]') # 영문, 소문자 string이 하나씩만 매칭
print(pattern.findall(string))
```

✓ 0.0s

Python

```
['e', 'j', 'u', 'w', 'a', 's', 'f', 'a', 'n', 't', 'a', 's', 't', 'i', 'c']
```

```
string = "Jeju MT was fantastic. 이번 엠티도 기대된다! "
result = re.sub('[^가-힣]', '', string) # string에서 한글이 아닌 모든 문자(영어, 숫자, 공백, 특수 문자 등)가 제거 (''로 대체)
print(result)
```

✓ 0.0s

Python

```
이번엠티도기대된다
```

**Normalization:** 겉보기엔 다른 단어더라도, 의미가 같은 단어(token)끼리 통일하는 작업

=> 주로 영어 데이터에서 중요하게 작용(한국어는 거의x)

### 대/소문자 통합

```
string = "KuBiG"
print("upper:", string.upper())
print("lower:", string.lower())
✓ 0.0s
upper: KUBIG
lower: kubig
```

### 어간 추출 (Stemming)

규칙에 따라 접사 부분을 삭제하고 어간만 남김.

품사 정보가 훼손될 수 있고, 단어의 형태가 어색할 수 있음.

```
1 # 어간 추출 (Stemming)
2
3 from nltk.stem import PorterStemmer
4 from nltk.tokenize import word_tokenize
5
6 stemmer = PorterStemmer()
7
8 print(stemmer.stem('dies'))
9 print(stemmer.stem('dying'))
10 print(stemmer.stem('diying')) # 오타
die
die
diy
```

### 표제어 추출 (Lemmatization)

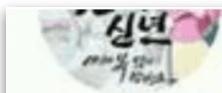
단어의 품사 정보를 그대로 유지. 하지만 본래 단어의 품사 정보가 주어져야 함.  
그렇지 않을 경우, 표제어 추출이 어색할 수 있음  
어간 추출에 비해 단어의 형태를 적절하게 유지.

```
1 # 표제어 추출 (Lemmatization)
2 # nltk.download('wordnet')
3
4 from nltk.stem import WordNetLemmatizer
5
6 lemmatizer = WordNetLemmatizer()
7
8 print(lemmatizer.lemmatize('dies'))
9 print(lemmatizer.lemmatize('dies', 'n'))
10 print(lemmatizer.lemmatize('dies', 'v'))
11
12 print(lemmatizer.lemmatize('watched', 'v'))
13 print(lemmatizer.lemmatize('watched', 'a'))
14
15 print(lemmatizer.lemmatize('playing', 'v'))
16 print(lemmatizer.lemmatize('playing', 'n'))
```

```
dy
dy
die
watch
watched
play
playing
```

## 2-4. Normalization

Normalization: 겉보기엔 다른 단어더라도, 의미가 같은 단어(token)끼리 통일하는 작업



@ys\_skywing

외국어 띄어쓰기보다 우리말 띄어쓰기가  
어려운 거 실화입니까

일본어 : 다 붙이면 됨.

중국어 : 다 붙이면 됨.

영어 : 단어 다 띄면 됨.

한국어 : 너 따위가 이것이 단어인지  
조사인지 구별할 수 있을까?

2017년 06월 19일 · 7:40 오후

너 나 본 지 두 달 다 돼 감

이게 맞는 띄어쓰기래요  
맞춤법의 기묘함....

2015년 06월 08일 · 4:10 오후

1,481 리트윗 139 관심글



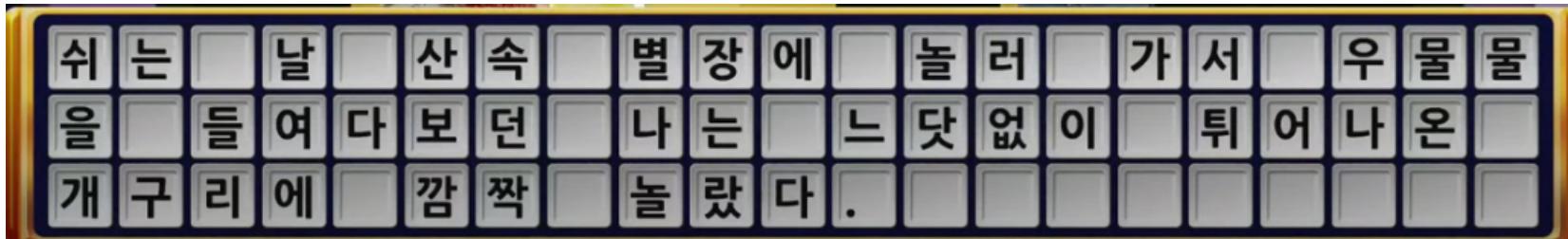
**Normalization:** 겉보기엔 다른 단어더라도, 의미가 같은 단어(token)끼리 통일하는 작업

### 띄어쓰기 (PyKoSpacing)

띄어쓰기가 되어있지 않은 문장을 띄어쓰기 한 문장으로 변환해주는 딥 러닝 기반 패키지

```
1 from pykospacing import Spacing
2
3 sent = "쉬는날산속별장에놀러가서우물물을들여다보던나는느닷없이튀어나온개구리에깜짝놀랐다"
4
5 spacing = Spacing()
6 kospacing_sent = spacing(sent)
7
8 print(kospacing_sent)
```

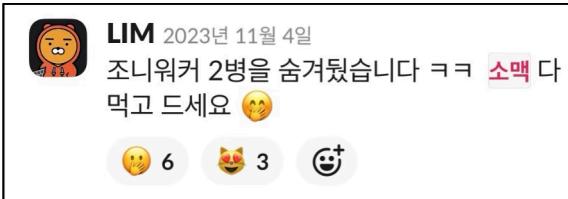
쉬는 날 산 속 별장에 놀러 가서 우물물을 들여다보던 나는 느닷없이 튀어나온 개구리에 깜짝 놀랐다



## 2-5. Tokenization

Tokenization(토큰화): 단어를 의미 있는 단위로 잘게 쪼개는 과정

### 토큰화



### 영어

띄어쓰기 단위로 자르더라도 토큰화 충분.

```
1 from nltk.tokenize import word_tokenize, WordPunctTokenizer
2
3 text = "KUBIG members couldn't find Lim's Jonny Walker."
4 print("토큰화1:", word_tokenize(text))
5 print("토큰화2:", WordPunctTokenizer().tokenize(text))
```

```
토큰화1: ['KUBIG', 'members', 'couldn\'t', 'find', 'Lim', "'s", 'Jonny', 'Walker', '.']
토큰화2: ['KUBIG', 'members', 'couldn\'t', "", 't', 'find', 'Lim', "", 's', 'Jonny', 'Walker', '.']
```

2번 토큰화 방법은 apostrophe(')를 개별 토큰으로 인식.

### 한국어

띄어쓰기론 토큰화 X. 주로 형태소 단위로 토큰화 수행.  
Okt, kkma, mecab 등 다양한 한국어 형태소 분석기 제공

```
1 from konlpy.tag import Okt
2 okt = Okt()
3
4 text = "승현이는 지원이의 구름 이모지가 제법 탐난다"
5
6 print("형태소 분석: ", okt.morphs(text))
7 print("품사 태깅: ", okt.pos(text))
8 print("명사 추출: ", okt.nouns(text))
9
```

```
형태소 분석: ['승현', '이', '는', '지원이', '의', '구름', '이모지', '가', '제법', '탐', '난다']
품사 태깅: [('승현', 'Noun'), ('이', 'Noun'), ('는', 'Suffix'), ('지원이', 'Noun'), ('의', 'Noun'), ('구름', 'Noun'), ('이모지', 'Noun'), ('가', 'Verb'), ('제법', 'Verb'), ('탐', 'Verb'), ('난다', 'Verb')]
명사 추출: ['승현', '지원이', '구름', '이모지', '제법']
```

배지원\_19기

**Tokenization(토큰화):** 단어를 의미 있는 단위로 잘게 쪼개는 과정

## 불용어(stopword) 제거

### 영어

Nltk 라이브러리에서 불용어 사전 제공  
영어 포함 23개 언어 제공, 그러나 한국어는 없음.

```
1 from nltk.corpus import stopwords
2 from nltk.tokenize import WordPunctTokenizer
3 import nltk
4
5 nltk.download('stopwords') # 불용어 데이터 다운로드
6
7 text = "KUBIG members couldn't find Lim's Jonny Wallker."
8 stop_words = set(stopwords.words('english'))
9
10 tokenizer = WordPunctTokenizer()
11 word_tokens = tokenizer.tokenize(text)
12
13 result = [word for word in word_tokens if word.lower() not in stop_words]
14
15 print("불용어 제거 전:", word_tokens)
16 print("불용어 제거 후:", result)
```

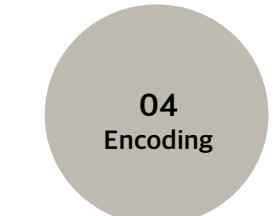
```
불용어 제거 전: ['KUBIG', 'members', 'couldn\'t', "", 't', 'find', 'Lim', "", 's', 'Jonny', 'Wallker', '.']
불용어 제거 후: ['KUBIG', 'members', "", 'find', 'Lim', "", 'Jonny', 'Wallker', '.']
```

### 한국어

라이브러리에서 따로 불용어 제공해주지 않음.  
경우에 따라 직접 생성하거나, 웹에서 txt 파일 받아서 사용하기!

```
1 from konlpy.tag import Okt
2
3 text = "쿠빅이들은 교수님이 숨겨둔 조니워커를 찾느라 눈돌아갔다"
4 stop_words = "은 이 를"
5 stop_words = set(stop_words.split(' '))
6 word_tokens = Okt().morphs(text)
7
8 result = [word for word in word_tokens if word not in stop_words]
9
10 print("불용어 제거 전:", word_tokens)
11 print("불용어 제거 후:", result)
```

```
불용어 제거 전: ['쿠빅이들', '은', '교수', '님', '이', '숨겨', '둔', '조니워커', '를', '찾느라', '눈', '돌아갔다']
불용어 제거 후: ['쿠빅이들', '교수', '님', '숨겨', '둔', '조니워커', '찾느라', '눈', '돌아갔다']
```



정수 인코딩  
원핫 인코딩  
패딩

**Encoding:** 전처리된 토큰을 컴퓨터가 이해할 수 있게 숫자로 mapping해주는 과정

```
1 text = "건호는 콘텐츠기획팀장이고 광민이는 인사팀장이다"
2 word_tokens = okt.morphs(text)
3
4 word2index = {}
5 for word in word_tokens:
6     if word not in word2index.keys():
7         word2index[word] = len(word2index)
8
9 word2index['OOV'] = len(word2index) + 1
10
11 print("단어 집합: ", word2index)
```

단어 집합: {'건호': 0, '는': 1, '콘텐츠': 2, '기획': 3, '팀': 4, '장이': 5, '고': 6, '광민': 7, '이': 8, '인사': 9, '다': 10, 'OOV': 12}

**OOV(Out Of Vocabulary):** 단어집합에 없는 단어들을 mapping해주기 위한 index key

```
1 text = "우진이는 인사팀원이고 다른 팀과 친하게 지낸다"
2 word_tokens = okt.morphs(text)
3
4 encoded = []
5 for word in word_tokens:
6     try:
7         encoded.append(word2index[word]) # 단어 집합에 있는 단어면 해당 단어 index 반환
8     except KeyError:
9         encoded.append(word2index['OOV']) # 없다면 OOV의 index 반환 (12)
10
11 encoded.append(encoded)
12 print(word_tokens)
13 print(encoded)
```

['우진', '이', '는', '인사', '팀', '원', '이고', '다른', '팀', '과', '친하게', '지낸다']
[12, 8, 1, 9, 4, 12, 12, 12, 4, 12, 12, 12, [...]]

# 03 Basic of Word Representation

One Hot Vector, 카운트 기반의 단어 표현(DTM, TF-IDF)

### 3-1. One Hot Encoding

단어를 벡터화하는 가장 naïve한 방식이 **One-Hot Encoding**

token	index	One hot vector
광민	0	[1,0,0,0,0]
이	1	[0,1,0,0,0]
는	2	[0,0,1,0,0]
인사	3	[0,0,0,1,0]
팀장	4	[0,0,0,0,1]

one hot vector의 차원은 단어사전의 크기가 됨.

만약 단어사전에 단어가 매우 많아진다면?  
**벡터의 차원이 무한정 커지면서 저장 공간 측면에서 매우 비효율적!**

One hot vector는 단어 간 유사도를 파악할 수 없음!

## 3-2. BoW(Bag of Words)

**BoW:** 단어들의 출현 빈도에 집중하는 방식.  
각 단어에 고유한 정수 인덱스를 부여한 후,  
각 인덱스 위치에 단어 등장 빈도를 표현

```
def build_bag_of_words(document):
    tokenized_document = okt.morphs(document)

    word_to_index = {}
    bow = []

    for word in tokenized_document:
        if word not in word_to_index.keys():
            word_to_index[word] = len(word_to_index)
            # BoW에 전부 기본값 1을 넣는다.
            bow.insert(len(word_to_index) - 1, 1)
        else:
            # 재등장하는 단어의 인덱스
            index = word_to_index.get(word)
            # 재등장한 단어는 해당하는 인덱스의 위치에 1을 더한다.
            bow[index] = bow[index] + 1

    return word_to_index, bow
```

I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!



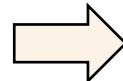
```
1 doc = "동주는 목동에서 올 때는 지하철 갈 때는 버스로 힘들게 통학한다"
2 vocab, bow = build_bag_of_words(doc)
3
4 print("vocabulary:", vocab)
5 print("BoW Vector:", bow)
```

```
vocabulary: {'동주': 0, '는': 1, '목동': 2, '에서': 3, '올': 4, '때': 5, '지하철': 6, '갈': 7, '버스': 8, '로': 9, '힘들게': 10, '통학': 11, '한다': 12}
BoW Vector: [1, 3, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1]
```

### 3-3. DTM(Document-Term Matrix)

**DTM:** 여러 문서가 존재할 때, 각 문서의 BoW를 하나의 행렬로 표현한 것. **문서 간의 유사도** 파악 가능

문서1: 승현이는 쿠빅을 좋아한다  
문서2: 우리는 쿠빅을 좋아한다  
문서3: 교수님은 인공지능을 좋아한다



	승현이는	쿠빅을	좋아한다	우리는	교수님은	인공지능을
문서1	1	1	1	0	0	0
문서2	0	1	1	1	0	0
문서3	0	0	1	0	1	1

#### 한계1)

DTM의 각 행은 마치 One Hot Vector처럼 단어 집합의 크기에 따라 차원이 무한정 커질 수 있음.  
대부분의 값이 0이 되는 불상사 => 저장 공간 측면에서 매우 비효율적.  
이러한 비효율적인 표현 방식을 **희소 표현(Sparse Representation)**이라고 함

#### 한계2)

단순 빈도 수 기반으로 접근하다보니, 중요한 단어든 불필요한 단어든 구분없이 빈도를 세게 됨  
단어의 중요성에 가중치를 둘 방법이 없을까? => sol) **TF-IDF**

**TF-IDF (Term Frequency-Inverse Document Frequency)** : DTM 내의 각 단어마다 중요도를 가중치로 주는 행렬 표현 방식

- $TF(d, t)$ : 특정 문서  $d$ 에서의 특정 단어  $t$ 의 등장 횟수
- $DF(t)$ : 특정 단어  $t$ 가 등장한 문서의 수
- $IDF(t)$ :  $DF(t)$ 에 반비례 하는 수

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

**TF-IDF**: TF와 IDF를 곱한 값. 값이 클수록 중요도가 높음.

```
import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()

doc = ["승현이는 쿠빅을 정말 좋아한다",
       "우리는 쿠빅을 진짜 좋아한다",
       "교수님은 인공지능을 정말 사랑한다"]

dtm = vectorizer.fit_transform(doc)
tf = pd.DataFrame(dtm.toarray()), columns=vectorizer.get_feature_names_out()

df = tf.astype(bool).sum(axis=0)
D = len(tf)
idf = np.log((D+1) / (df+1) +1)
tf_idf = tf * idf
tf_idf = tf_idf / np.linalg.norm(tf_idf, axis=1, keepdims=True)

tf_idf
✓ 0.1s
```

	교수님은	사랑한다	승현이는	우리는	인공지능을	정말	좋아한다	진짜	ку빅을
0	0.000000	0.000000	0.701495	0.000000	0.000000	0.411463	0.411463	0.000000	0.411463
1	0.000000	0.000000	0.000000	0.609928	0.000000	0.000000	0.357754	0.609928	0.357754
2	0.546845	0.546845	0.000000	0.000000	0.546845	0.320752	0.000000	0.000000	0.000000

# 04 Word2Vec, FastText

about Word Embedding

## 4-1. What is Word Embedding?

회소표현  
Sparse rep

- 벡터 또는 행렬의 값이 대부분 0으로 표현
- 단어집합의 크기 = 벡터의 차원
- 저장공간 상의 낭비 심함
- 단어 간 유사성 표현이 어려움
- Ex) one hot vector, DTM

밀집표현  
Dense rep

- 0과 1만이 아닌 실수값들로 표현됨
- 벡터의 차원을 사용자가 설정
- 차원이 보다 작아짐(=밀집됨)
- 단어 간 유사성 파악 가능
- Ex) Word Embedding

**Word Embedding:** 단어들을 밀집 벡터로 표현하는 방법  
**Embedding Vector:** Word Embedding을 거쳐 생성된 밀집 벡터  
앞으로 배울 **Word2Vec, FastText, Glove**이 그 대표적 예시

## 4-2. What is Word2Vec?

고려대학교+라이벌

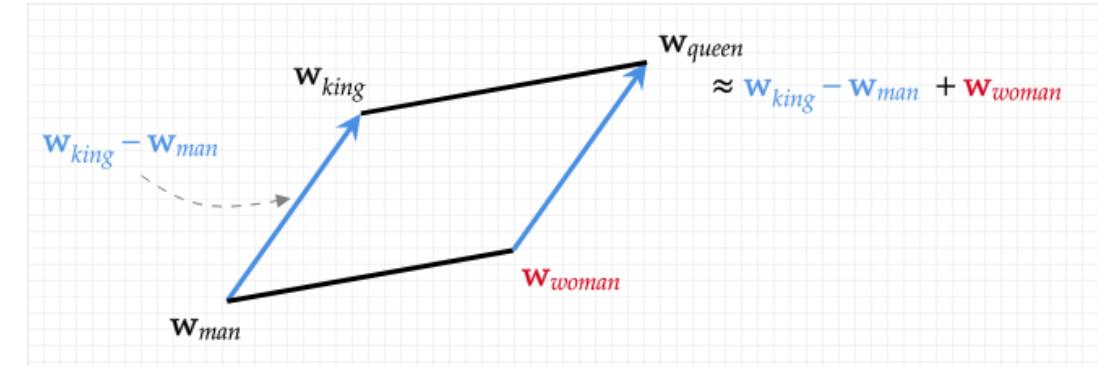
QUERY

+고려대학교/Noun +라이벌/Noun

RESULT

연세대학교/Noun

<http://w.elnn.kr/search/>



Word2Vec로 구한 벡터는 유사도를 반영한 벡터이기에 연산이 가능하며, neural net으로 최적의 값을 찾아나감

Tomas Mikolov(2013),  
Efficient Estimation of Word Representation in Vector Space

with the expectation that not only will similar words tend to be close to each other, but that words can have **multiple degrees of similarity**

Tomas Mikolov(2013),  
Efficient Estimation of Word Representation in Vector Space

algebraic operations are performed on the word vectors  
 $\text{vector}(\text{"King"}) - \text{vector}(\text{"Man"}) + \text{vector}(\text{"Woman"}) = \text{vector}(\text{"Queen"})$

### CBOW (Continuous Bag of Words)

주변에 있는 단어들로 중심에 있는 단어를 예측

Input: 주변 단어(context word)

민재는    계정을 담당하는 홍보팀장이야

Output: 중심 단어(center word)

### Skip-Gram

중심에 있는 단어로 주변에 있는 단어를 예측

Input: 중심 단어(center word)

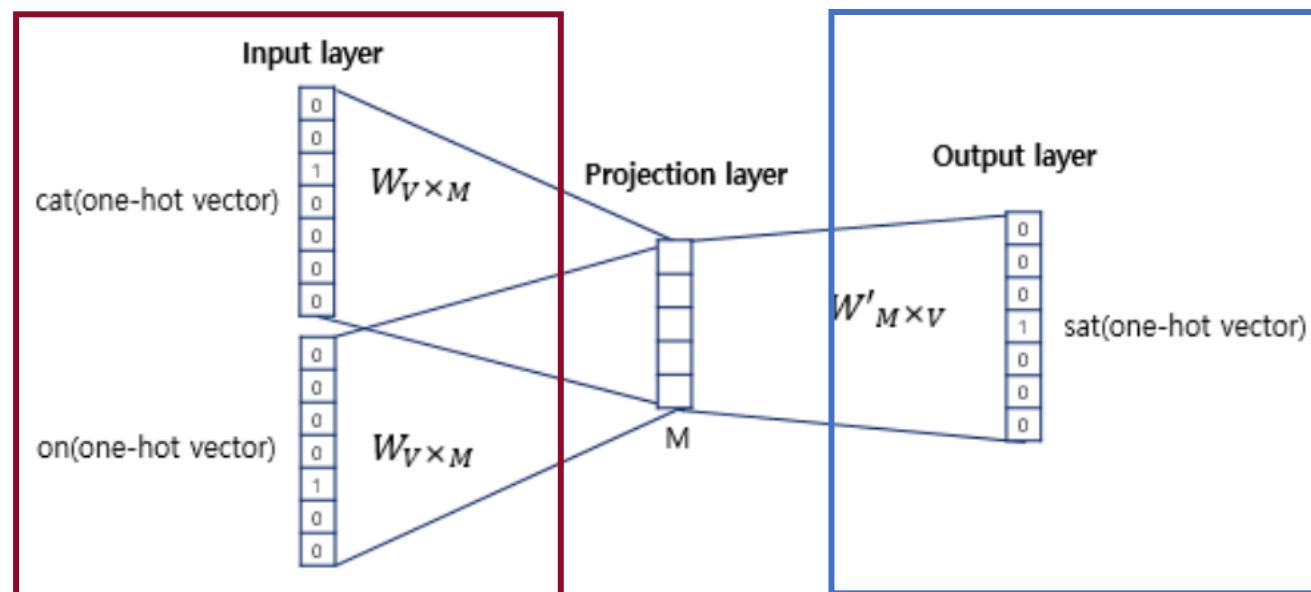
민재는    인스타그램 담당하는 홍보팀장이야

Output: 주변 단어(context word)

**CBOW(Continuous Bag of Words)**: 주변 단어(context words)로 중심 단어(center word)를 예측하는 학습 방식

Example: “The fat cat sat on the mat”

GOAL: window size=1 일 때 **sat**을 예측하라



$V$ : 단어 사전의 크기 (one-hot vector의 차원)

$M$ : 임베딩 벡터의 차원 (축소할 만큼의 차원)

Window size: 중심단어 앞뒤로 몇개의 단어를 볼 지

**CBOW(Continuous Bag of Words)**: 주변 단어(context words)로 중심 단어(center word)를 예측하는 학습 방식

Example: “The fat cat sat on the mat”, window size = 2

중심단어 주변단어

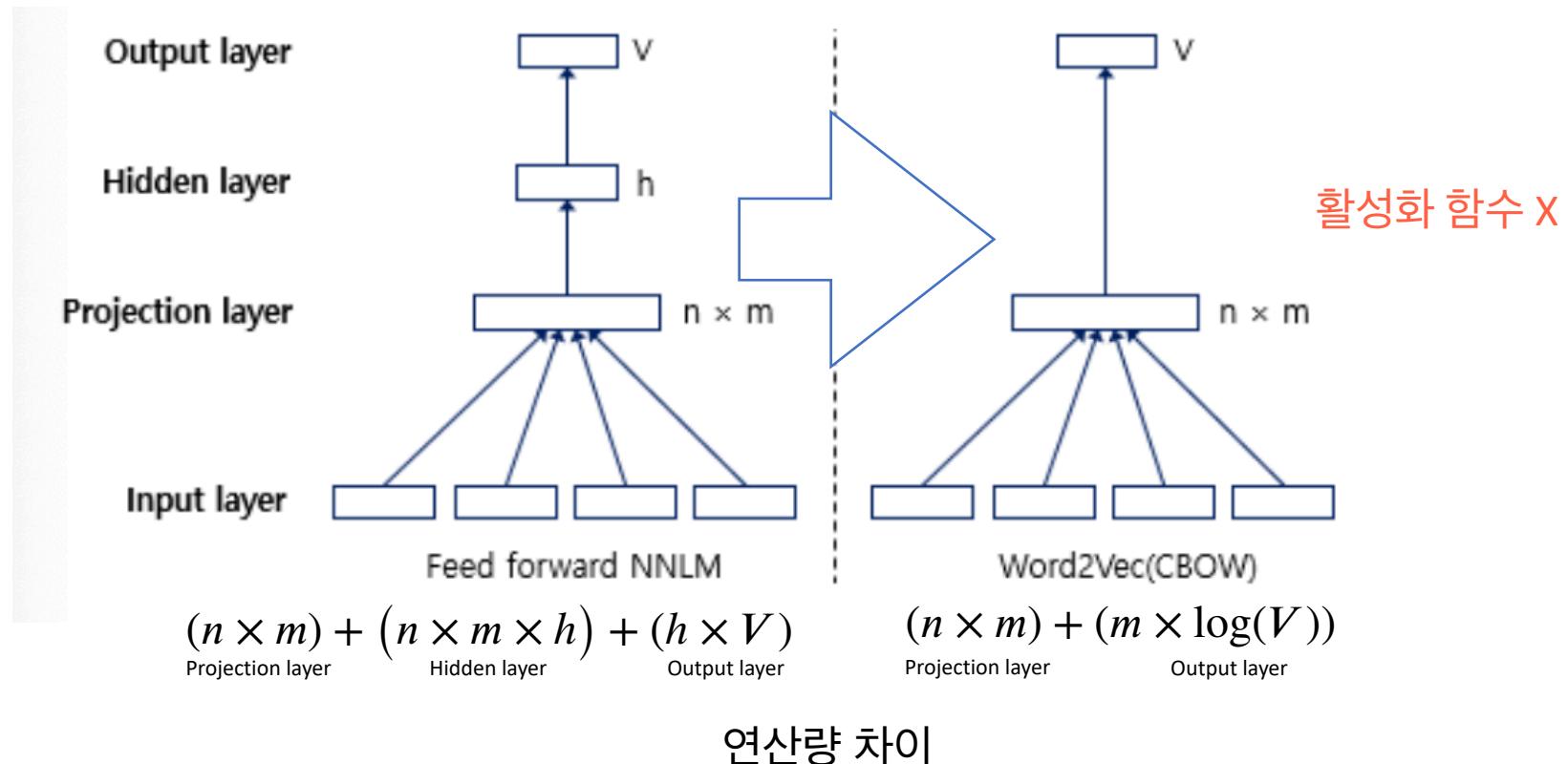
The fat cat sat on the mat

윈도우 크기가 n이라고 한다면,

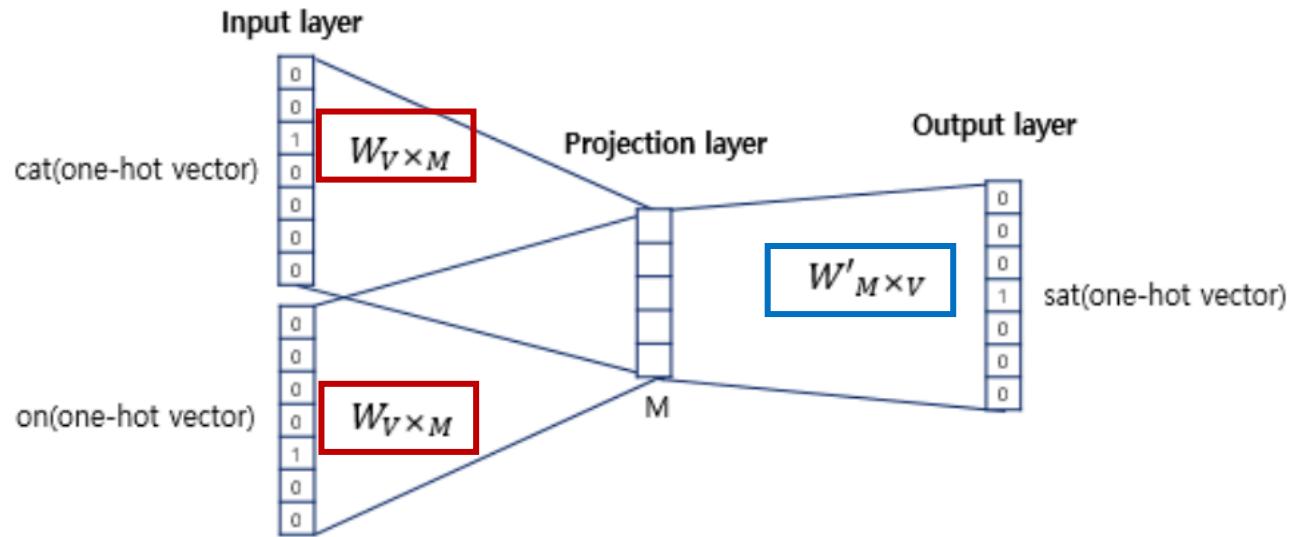
실제 중심 단어를 예측하기 위해 참고하려고 하는 주변 단어의 개수는  $2n$

윈도우 크기가 정해지면 윈도우를 옆으로 움직여서 주변 단어와 중심 단어의 선택을 변경해가며 학습을 위한 데이터 셋 생성

Q. neural net이라기엔, 우리가 아는 것들보다 너무 단순한 layer인데?  
기존의 NNLM, RNNLM의 nonlinear, hidden layer들이 비효율적이라 **projection layer**만 쓰은 간단한 형태

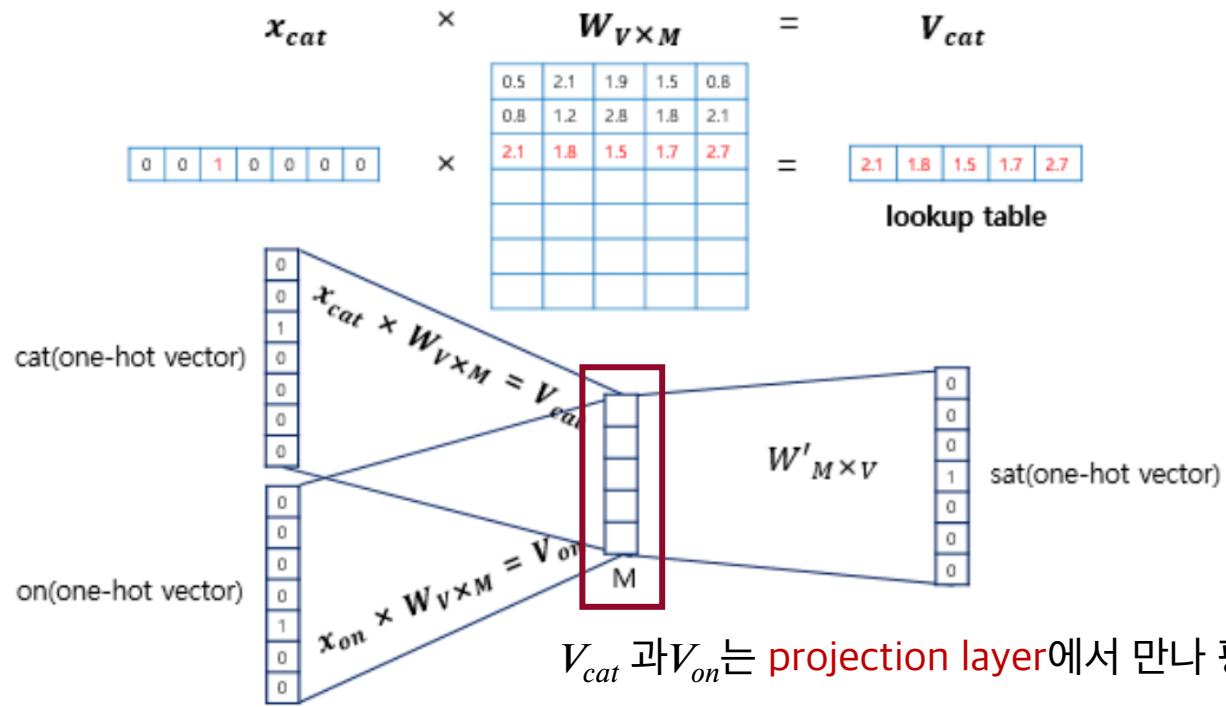


Example: “The fat cat sat on the mat”



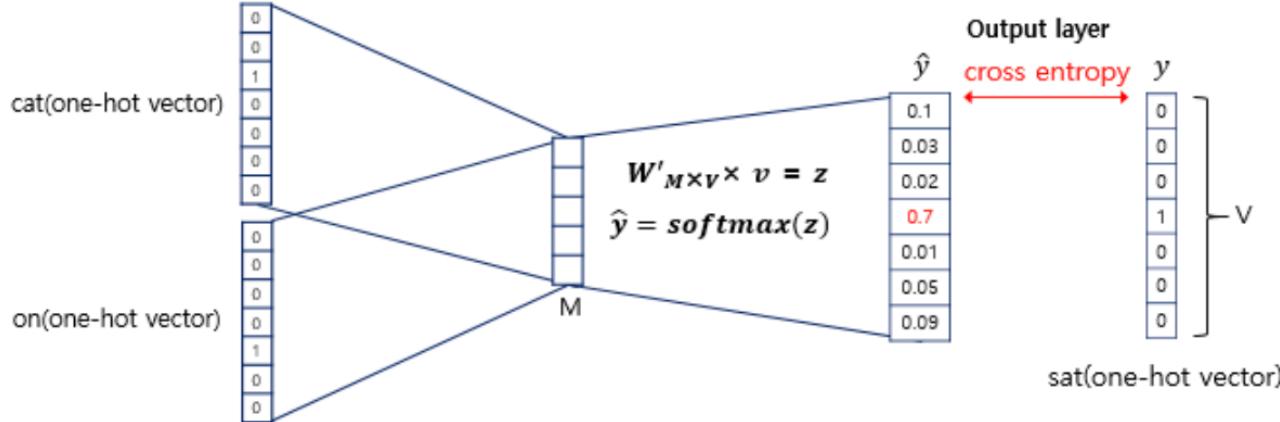
Input으로 window size만큼의 input (one-hot vector)이 들어옴  
 학습을 위해 예측해야 하는 단어(output)의 label (one-hot vector)도 필요함

가중치 행렬  $W$ ,  $W'$ 의 초기값은 랜덤 ( $W$ ,  $W'$ 는 전치관계가 아님!)  
 중심 단어를 더 잘 맞히기 위한 학습 과정에서 가중치도 학습됨  
 학습을 마친 후  $W$ 의 행 벡터가 해당 단어의 embedding vector가 됨!



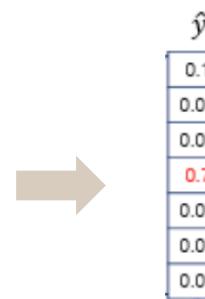
가중치 행렬  $W$ 는  $V * M$  차원이므로,  
1 \*  $V$  차원인 one hot vector와 곱해지면  
1에 해당하는 행의 값만 lookup해옴

차원도  $M$  (우리가 원했던 저차원) 만큼으로 변함!



$$v = \frac{V_{cat} + V_{on}}{2 * n(\text{window.size})}$$

평균치인  $v$ 는 가중치 행렬  $W'$ 과 곱해져 크기  $V$ 의 벡터  $z$ 가 됨



$z$ 는 softmax를 거쳐 0~1 사이의 실수 벡터( $y^{\wedge}$ )로 변환.  
4번째 index에 해당하는 sat0이 **중심단어일 확률이 0.7**이다

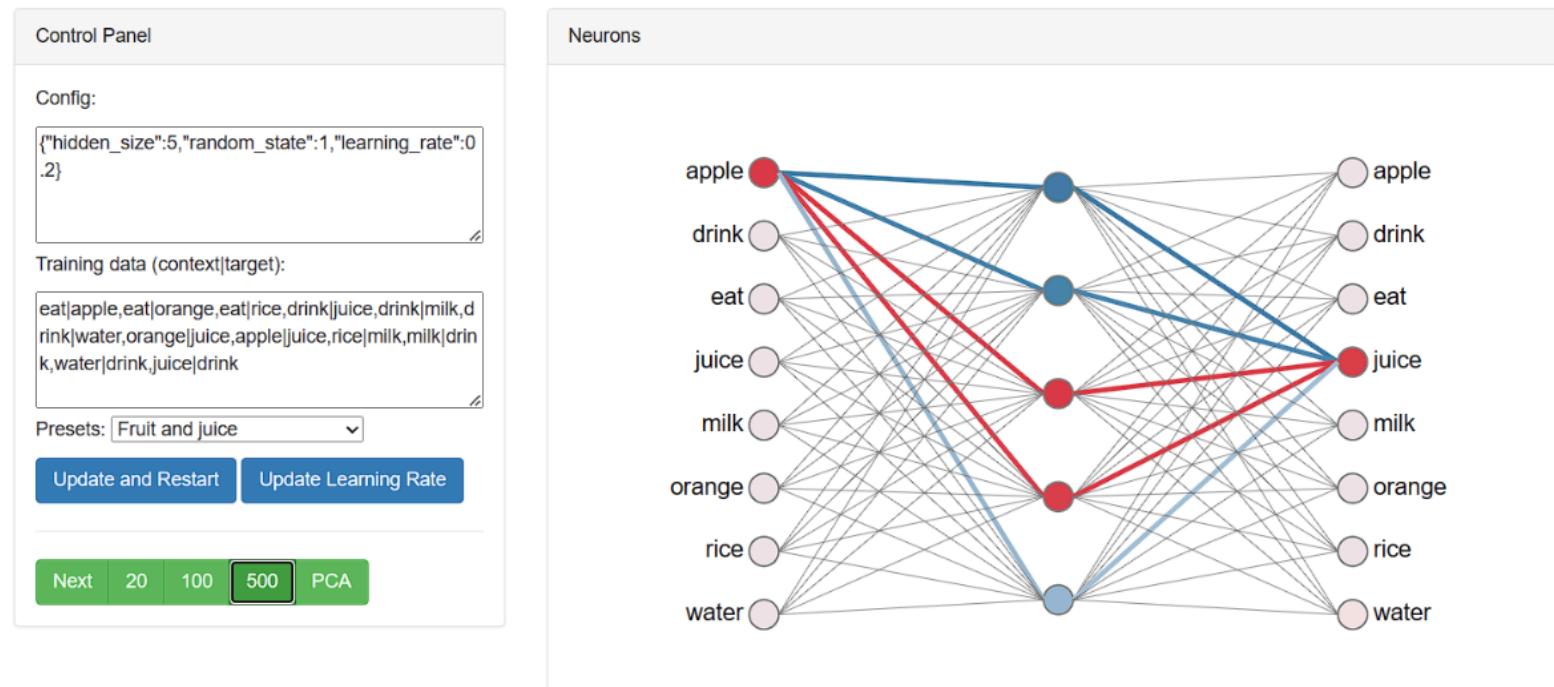
$$\text{cost}(\hat{y}, y) = - \sum_{j=1}^V y_j \log(\hat{y}_j)$$

Loss function으로 **cross entropy** 사용. 우변을 최소화하는 방향으로 학습  
이 때 발생한 오차로 **backpropagation**하여 가중치  $W$  조정!  
 $W$ 의 행 벡터가 해당 단어의 **embedding vector**가 됨!

<https://ronxin.github.io/wevi/>

## wevi: word embedding visual inspector

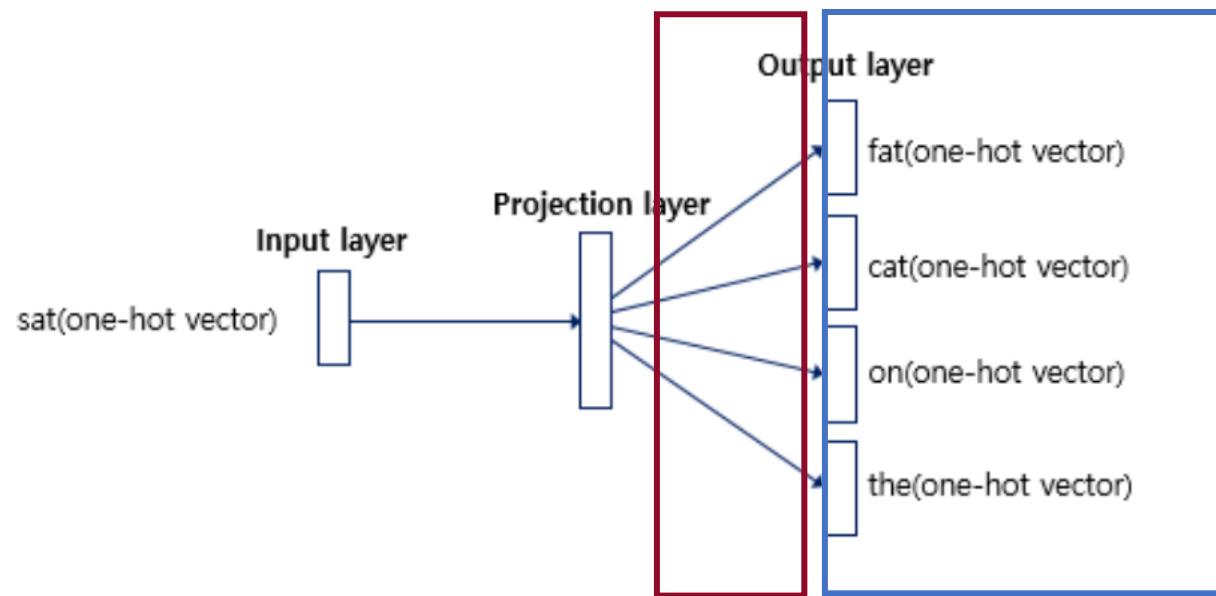
[Everything you need to know about this tool](#) - [Source code](#)



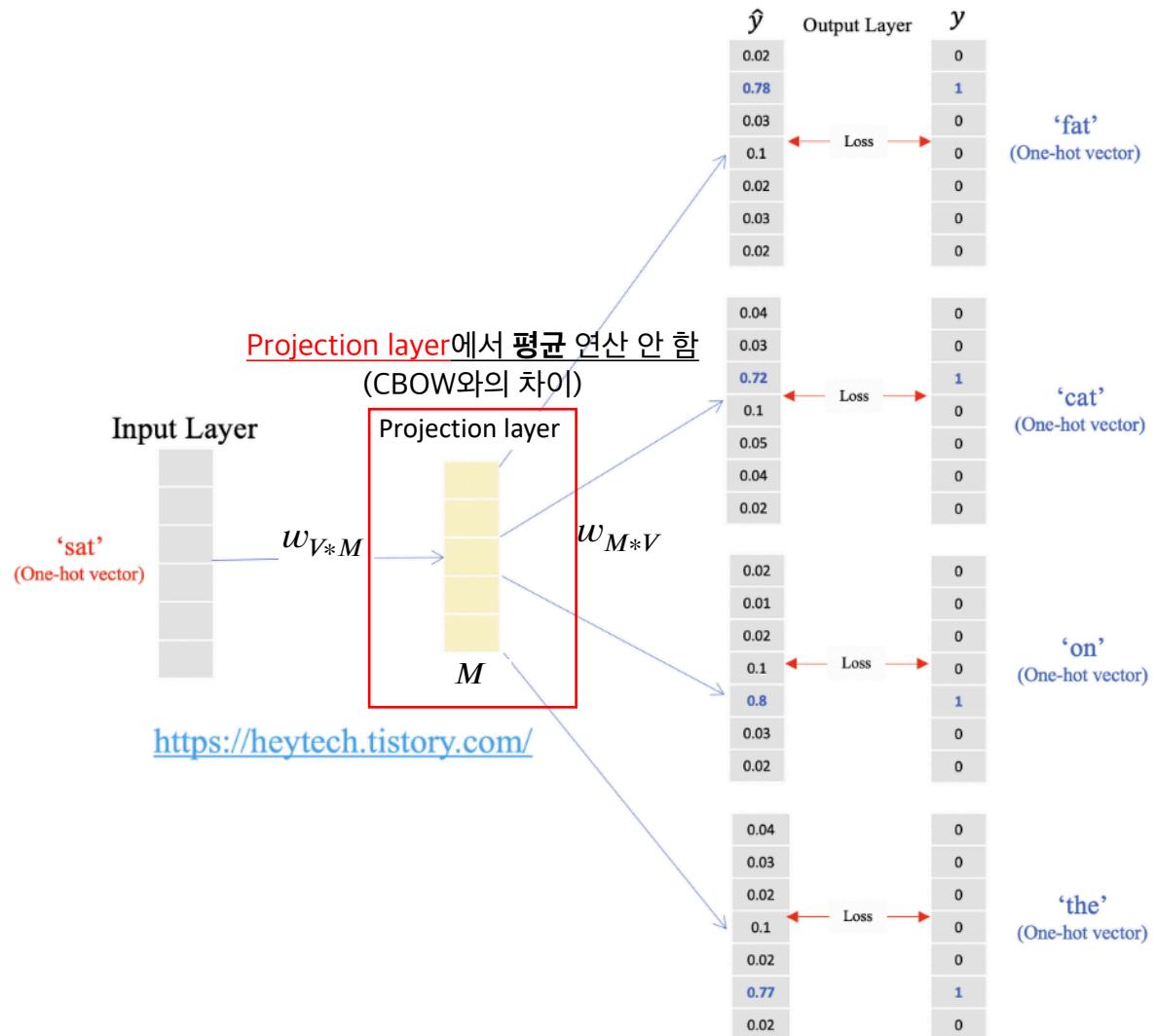
**Skip-gram:** 중심 단어(center word)로 주변 단어(context words)를 예측하는 학습 방식

Example: “The fat cat sat on the mat”

GOAL: window size=2 일 때 sat에 주변에 있는 fat, cat, on, the를 예측하라



## 4-5. Skip-gram



## 4-5. Skip-gram, Negative sampling

Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125



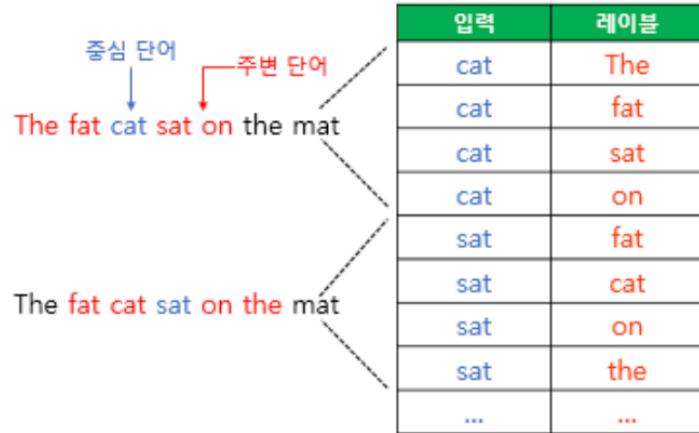
따라서 Word2vec에선 Skip-gram을 주로 사용하는데,  
이에 더해 Negative Sampling까지 추가로 사용

CBOW보다 Skip-gram의 성능이 보다 우수함

**Negative Sampling:** 속도 문제를 개선하기 위해 **다중 분류** 문제를 **이진 분류**로 바꾸는 방식



## 4-5. Skip-gram, Negative sampling



입력과 레이블의 변화

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
...	...	...

레이블=1: 입력1과 입력2가 서로 이웃관계가 맞음  
레이블=0: 입력1과 입력2가 서로 이웃관계가 아님

Negative Sampling

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

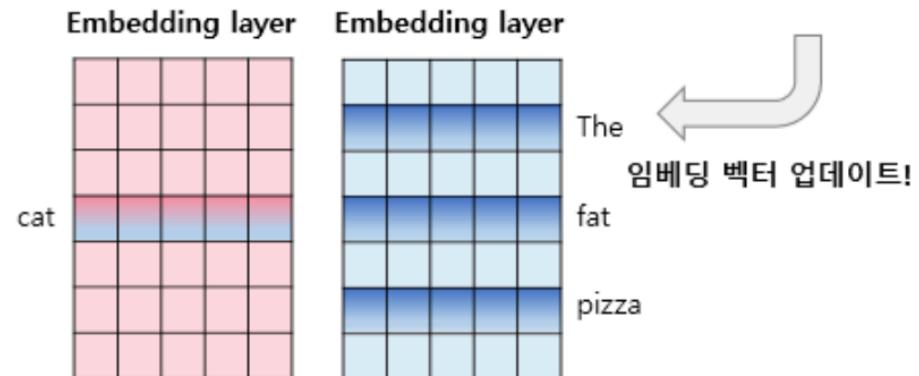
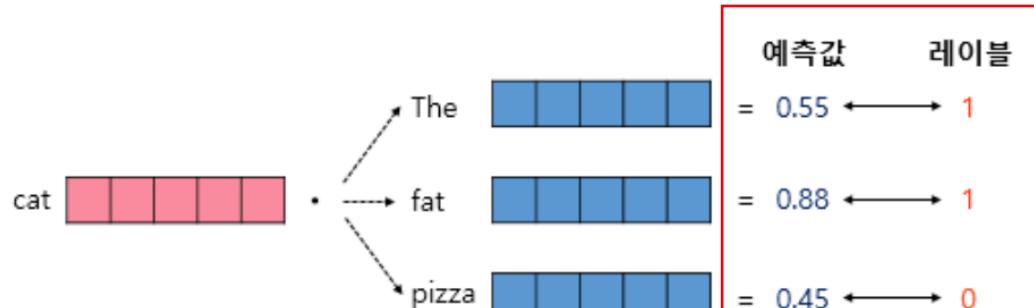
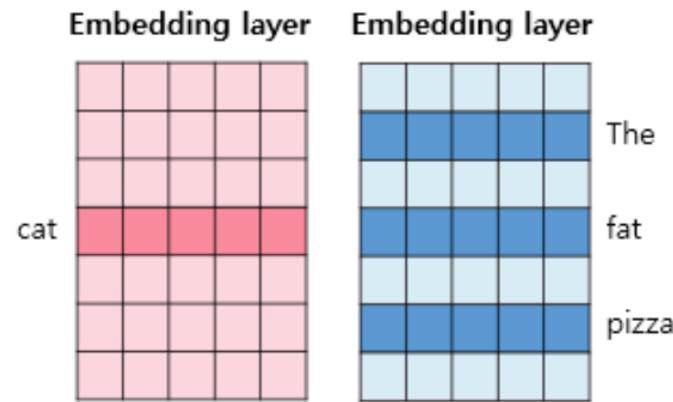
이웃관계가 아닌 경우,  
즉 부정(negative)의 경우(레이블=0)도 학습을 해야 하기 때문에  
negative sampling이란 이름이 붙음

단어 집합에서 랜덤으로  
선택된 단어들을  
레이블 0의 샘플로 추가.

## 4-5. Skip-gram, Negative sampling

[Negative sampling의 학습 과정]

Vector를 비교하던 기존 skip-gram과 달리(multi class)  
Scalar를 비교하게 됨(binary)  
즉 불필요한 연산이 줄어듦





### FastText: Word2Vec 방식을 확장하여, word를 subword로 쪼개어 학습하는 방식

Piotr Bojanowski(2017),  
Enriching Word Vectors with Subword Information

Most popular models represent each word of vocabulary  
by a distinct vector ...  
they ignore the morphology and structures of words

...

In this paper, we propose to learn representations for character  
n-gram, and to represent words as the sum of the n-gram vectors

하나의 단어를 하나의 distinct vector로 보는 것은 morphology(형태학)을 간과함

French, Spanish 등 morphologically rich languages는 OOV가 너무 많음



단어를 n-gram으로 쪼개는 모델(subword)을 만들어서 이러한 단점을 보완하고자 함

### [Subword model]

3-gram: tig, ige, ger

4-gram: tige, iger

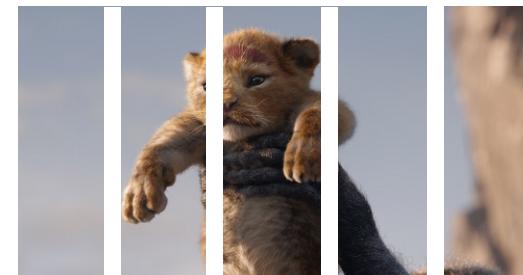
5-gram: tiger

special token = <tiger>

tiger의 gram set

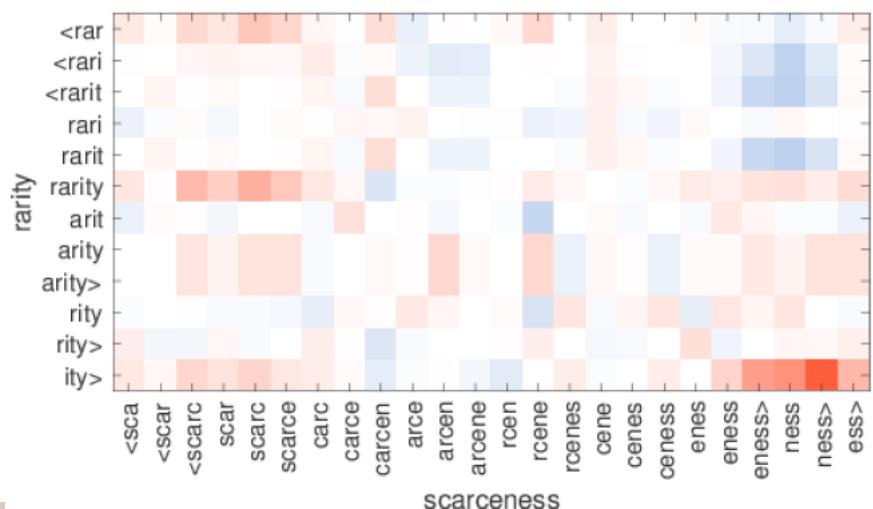
보통 한 단어의 gram set은

3~6 gram으로 이루어진다



ex) 다른 모델에선 OOV인 electrofishing(전류어법)이란 단어를 4-gram으로 나누면

이것이 **OOV로 처리되지 않고**, elec, fish 등으로 쪼개져 **유의미한 벡터**로 쓰일 수 있다.



[similarity between character n-grams in OOV]

명사형 접미사 ness와 ity가 높은 유사도  
실질적 의미를 갖는 scarc와 rarity도 높은 유사도

# 05 Glove

with co-occurrence matrix

### 카운트 기반

DTM, TF-IDF, LSA

- 장) Corpus 전체적인 통계 정보를 반영
- 단) 단어 의미의 유추(analogy) task에서는 불리

### 예측 기반

Word2Vec

- 장) 단어 의미의 유추(task)에서 유리
- 단) corpus 전체적인 통계 정보를 간과  
불필요한 단어의 분포도 모두 학습하여 비효율적

‘카운트 기반’, ‘예측 기반’을 모두 사용하면 어떨까?

Sol) **GloVe**

# 5-1. What is GloVe?



STANFORD

GloVe(Global Vectors for Word Representation):

co-occurrence count matrix(동시등장행렬)을 기반으로 하는 워드 임베딩 방식

[예문]

- I like deep learning
- I like NLP
- I enjoy flying

예문을 바탕으로 한  
Window size=1의 동시등장행렬

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

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

Co-occurrence probability

- $i$ : 중심 단어
- $k$ : 주변단어
- $P(k|i)$ : 특정 단어  $i$ 가 등장했을 때 어떤 단어  $k$ 가 등장한 횟수를 카운트하여 계산한 조건부 확률

### Notation

- $X$  : 동시 등장 행렬(Co-occurrence Matrix)
- $X_{ij}$  : 중심 단어  $i$ 가 등장했을 때 윈도우 내 주변 단어  $j$ 가 등장하는 횟수
- $X_i$  :  $\sum_j X_{ij}$  : 동시 등장 행렬에서  $i$ 행의 값을 모두 더한 값
- $P_{ik}$  :  $P(k | i) = \frac{X_{ik}}{X_i}$  : 중심 단어  $i$ 가 등장했을 때 윈도우 내 주변 단어  $k$ 가 등장할 확률

Ex)  $P(\text{solid} | \text{ice})$  = 단어 ice가 등장했을 때 단어 solid가 등장할 확률

- $\frac{P_{ik}}{P_{jk}}$  :  $P_{ik}$ 를  $P_{jk}$ 로 나눠준 값

Ex)  $P(\text{solid} | \text{ice}) / P(\text{solid} | \text{steam}) = 8.9$

- $w_i$  : 중심 단어  $i$ 의 임베딩 벡터
- $\tilde{w}_k$  : 주변 단어  $k$ 의 임베딩 벡터

### GloVe의 목표

중심 단어 벡터와 주변 단어 벡터의 내적이 동시등장 확률과 가까워지게 학습시키는 것

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

## 5-2. Loss Function of GloVe

$$F(v_1^T v_2 + v_3^T v_4) = F(v_1^T v_2) F(v_3^T v_4), \forall v_1, v_2, v_3, v_4 \in V$$

$$F(v_1^T v_2 - v_3^T v_4) = \frac{F(v_1^T v_2)}{F(v_3^T v_4)}, \forall v_1, v_2, v_3, v_4 \in V$$

곱셈과 뺄셈의 준동형식

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k ice)$	$1.9 \times 10^{-4}$	$6.6 \times 10^{-5}$	$3.0 \times 10^{-3}$	$1.7 \times 10^{-5}$
$P(k steam)$	$2.2 \times 10^{-5}$	$7.8 \times 10^{-4}$	$2.2 \times 10^{-3}$	$1.8 \times 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \times 10^{-2}$	1.36	0.96

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

Ratio를 벡터 공간에 인코딩하는 함수 F를 찾아보자!

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

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

우변이 scalar이므로 좌변도 벡터가 아닌 scalar가 되어야 함.

$$w_i^T w_k = (w_i - w_j)^T w_k + w_j^T w_k$$

$$\begin{aligned} F(w_i^T w_k) &= F\left(\left(w_i - w_j\right)^T w_k + w_j^T w_k\right) \\ &\because \text{준동형 성질} \\ &= F\left(\left(w_i - w_j\right)^T w_k\right) \times F(w_j^T w_k) \end{aligned}$$

$$F((w_i - w_j)^T w_k) = \frac{F(w_i^T w_k)}{F(w_j^T w_k)}$$

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

중심 단어  $w$ 와 주변 단어  $w^\sim$ 는 무작위로 선택됨.  
즉 서로 자리가 바뀌더라도 값이 같아야 함  
이를 만족하려면 함수 F는 준동형(Homomorphism)이어야 함

이 식을 만족해주는 함수? => 지수 함수

$$\exp(w_i^T w_k - w_j^T w_k) = \frac{\exp(w_i^T w_k)}{\exp(w_j^T w_k)} - \frac{P_{ik}}{P_{jk}}$$

$$\exp(w_i^T w_k) = P_{ik} = \frac{X_{ik}}{X_i}$$

$$w_i^T w_k = \log P_{ik} = \log\left(\frac{X_{ik}}{X_i}\right) = \log X_{ik} - \log X_i$$

$$w_k^T w_i = \log P_{ki} = \log\left(\frac{X_{ki}}{X_k}\right) = \log X_{ki} - \log X_k$$

$w_i$  와  $w_k$ 의 순서를 바꾸더라도 식이 성립해야 하는데... 값이 달라짐

$$w_i^T w_k + b_i + b_k = \log X_{ik}$$

편향(상수항)으로 대체

$$\text{Loss function} = \sum_{m,n=1}^V \left( w_m^T w_n + b_m + b_n - \log X_{mn} \right)^2$$

위에서 구한 값

## 5-2. Loss Function of GloVe

$$\text{Loss function} = \sum_{m,n=1}^V (w_m^T w_n + b_m + b_n - \log X_{mn})^2$$

문제점1)  $X_{ik}$  가 0일 수도 있음 => log 성립 안됨

문제점2) 동시등장행렬도 마치 DTM처럼 희소 행렬

$$f(x) = \begin{cases} (x/x_{\max})^\alpha & \text{if } x < x_{\max} \\ 1 & \text{otherwise} \end{cases}.$$

가중치 함수(weight function)  
동시발생 건수가 낮을수록 적은 가중치  
 $x_{\max}$ 를 넘으면 1

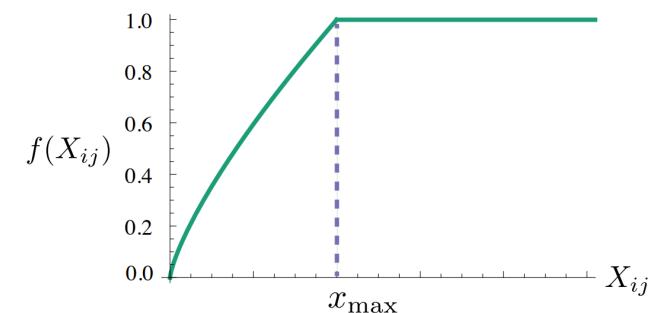


Figure 1: Weighting function  $f$  with  $\alpha = 3/4$ .

$$\text{Loss function} = \sum_{m,n=1}^V f(X_{ij}) (w_m^T w_n + b_m + b_n - \log X_{mn})^2$$

Remind ) GloVe의 목표

중심 단어 벡터와 주변 단어 벡터의 내적이 동시등장 확률과 가까워지게 학습시키는 것

Word2Vec, GloVe 임베딩 모델  
=> 단어 사전 형성 후 직접 학습



사전학습 된 언어모델에서 제공하는 임베딩 활용  
(BERT, GPT, RoBERTa 등)

## GloVe 임베딩

```
import numpy as np

# GloVe 파일 로드
glove_path = "glove.6B.300d.txt" # GloVe 사전학습된 파일 경로
embedding_dict = {}

with open(glove_path, "r", encoding="utf-8") as f:
    for line in f:
        values = line.split()
        word = values[0]
        vector = np.asarray(values[1:], dtype="float32")
        embedding_dict[word] = vector

# 단어 임베딩 추출
word = "language"
word_embedding = embedding_dict.get(word)
print(word_embedding.shape) # (300,
```

## BERT 임베딩

```
from transformers import BertModel, BertTokenizer
import torch

# BERT 모델과 토크나이저 로드
model_name = "bert-base-uncased"
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertModel.from_pretrained(model_name)

# 입력 문장
sentence = "I love natural language processing."

# 토큰화
inputs = tokenizer(sentence, return_tensors="pt", truncation=True, padding=True)

# 임베딩 추출
with torch.no_grad():
    outputs = model(**inputs)
    embeddings = outputs.last_hidden_state # [Batch size, Sequence length, Hidden size]

#CLS 토큰의 임베딩만 추출 (문장 전체 임베딩)
cls_embedding = embeddings[:, 0, :] # [Batch size, Hidden size]
print(cls_embedding.shape) # (1, 768)
```

# 06 Announcement

Week1 예습과제 Review

week2 예복습 과제 안내

week3 진도 안내

## 6-1. 우수 예습과제 Review

---

장건호

1주차  
예습과제1,2

Text preprocessing  
Word2Vec

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

## 6-2. Week2 예, 복습과제 안내, Week3 진도 안내



코드과제의 파일형식은 ipynb로, KUBIG 24-1 Github repo에 업로드 될 예정입니다!  
Colab 환경에서 제작된 과제들이므로 [google colab](#)에서 실행하시는 것을 권장드립니다.

### 2주차 복습과제1

IMDB 텍스트 감성분석

### 2주차 복습과제2

FastText vs Word2Vec

### 2주차 복습과제3

Word2Vec CBOW 구현

### 2주차 예습과제1

RNN vs LSTM

## WEEK3 진도

- RNN
- LSTM
- GRU
- ELMo

WEEK3 진도 해당 범위(미리 읽어보시면 도움이 됩니다!)  
[딥러닝을 이용한 자연어 처리 입문]

[밑바닥부터 시작하는 딥러닝2]

- Ch8. 순환 신경망
  - 08-01~03(RNN, LSTM, GRU)
  - 08-05(RNN LM)
- Ch9. 워드 임베딩
  - 09-09(ELMo)

- Ch5. 순환신경망

# 수고하셨습니다!