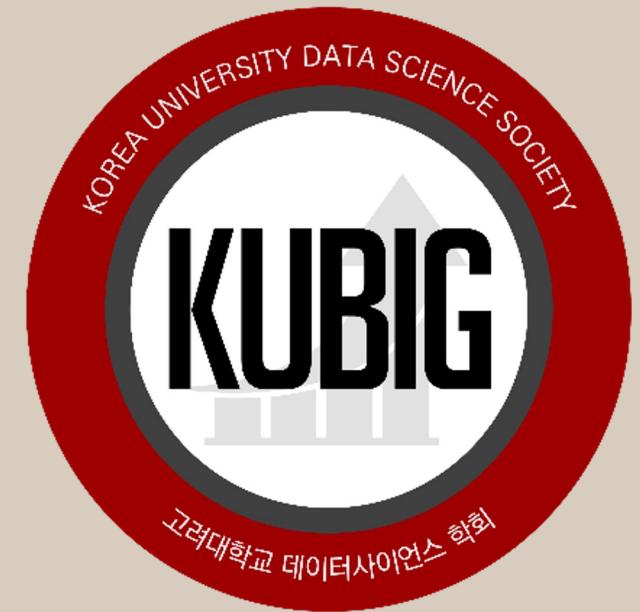


KUBIG 24-W  
겨울방학 BASIC STUDY SESSION

# NLP SESSION WEEK2



Today's Session Leader: 17기 김희준

---

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



[분반원 명단 변동]  
16기 임정준  
17기 서지민 안영지 황민아  
18기 김나연 김송성 이수민 장원준 정해원 진서연 최유민  
19기 이승준 최주희

홍여빈\_17기 오전 9:41  
안녕하세요 반갑습니다! 아래 링크는 줌 링크, 과제 현황 기록 등이 있는 노션입니다. 줌에서 7시에 뵙겠습니다 😊  
<https://puzzled-mouse-1f0.notion.site/NLP-6422889eef144d39bf802c08d208d10b?pvs=4>

▣ 홍여빈의 Notion on Notion  
[NLP 세미나 | Notion](#)  
일정 (36kB) ▾

1. 주교재  
딥러닝을 이용한 자연어 처리 입문  
많은 분들의 피드백으로 수년간 보완된 입문자를 위한 딥러닝 자연어 처리 교재 E-book입니다. 오프라인 출판물  
<https://wikidocs.net/book/2155>

2. 서브교재  
밀바닥부터 시작하는 딥러닝2  
<https://drive.google.com/1xPGVHbOB4g7T6wKbpcapWDX0CnDVQsyV>

PyTorch로 시작하는 딥 러닝 입문  
이 책은 딥 러닝 프레임워크 PyTorch를 사용하여 딥 러닝에 입문하는 것을 목표로 합니다. 이 책은 파이썬은 어느정  
<https://wikidocs.net/book/2788>

공지에 올라간 교재들 참고해서  
다음주 파트를 예습해오시는 걸 권장드립니다 : )

<https://puzzled-mouse-1f0.notion.site/REFERENCE-270a9b07e7ac435384cda2071b4c3d95>

승준님

1주차  
복습과제1

Pytorch tutorial

화면공유 하셔서 3분 내외로 가볍게 리뷰해주시면 됩니다!

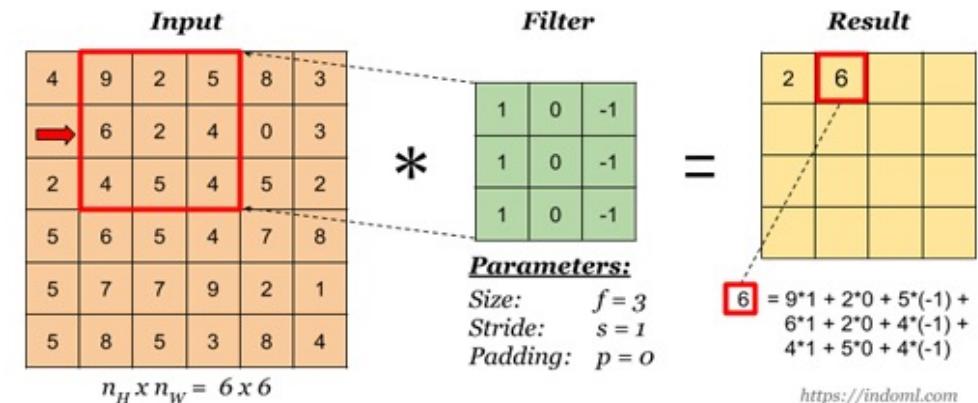
# 02 Text Preprocessing

텍스트 전처리의 대략적인 파이프라인

## 2-1. 텍스트 전처리, 왜, 어떻게 할까요?

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

Tabular(정형) data: 그 자체로 정형이라 별다른 처리 X



## Image data: feature map으로 표현

단어	인덱스	One hot vector
희준이는	0	[1,0,0,0,0]
교수님과	1	[0,1,0,0,0]
밥약을	2	[0,0,1,0,0]
하다가	3	[0,0,0,1,0]
체했다	4	[0,0,0,0,1]

## Text data: vector로 표현

단어	인덱스	One hot vector
희준이는	0	[1,0,0,0,0]
교수님과	1	[0,1,0,0,0]
밥약을	2	[0,0,1,0,0]
하다가	3	[0,0,0,1,0]
체했다	4	[0,0,0,0,1]

Text data: vector로 표현



단어를 저 상태로 input으로 넣어도 될까?

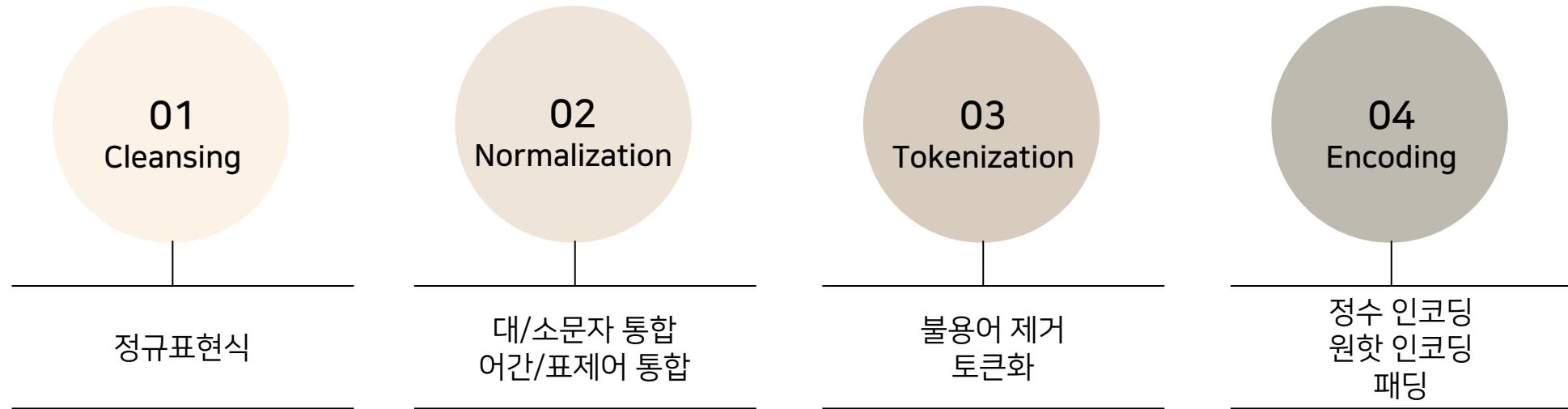
불필요한 조사, 통일되지 않은 용언들이 눈에 밟힘.

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

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

## 2-2. 텍스트 전처리 pipeline

Then, … How?



## 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 규칙을 따른다.

정규표현식은 이러한 규칙을 잡아내어, 필요한 부분들을 제거하는 데 유용하다.

.	한 개의 임의의 문자를 나타냅니다. (줄바꿈 문자인 \n 는 제외)	{숫자}	숫자만큼 반복합니다.
?	앞의 문자가 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 또는 1개)	{숫자1, 숫자2}	숫자1 이상 숫자2 이하만큼 반복합니다. ?, *, + 를 이것으로 대체할 수 있습니다.
*	앞의 문자가 무한개로 존재할 수도 있고, 존재하지 않을 수도 있습니다. (문자가 0개 이상)	{숫자,}	숫자 이상만큼 반복합니다.
+	앞의 문자가 최소 한 개 이상 존재합니다. (문자가 1개 이상)	[ ]	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.
^	뒤의 문자열로 문자열이 시작됩니다.	[^문자]	해당 문자를 제외한 문자를 매치합니다.
\$	앞의 문자열로 문자열이 끝납니다.	\	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 `WWd`
- 영어, 숫자를 제외하고 그 외(한글, 기호)만 표현하고 싶을 때 => `[^a-zA-Z0-9]`
- 한글이 아닌 것들만 표현하고 싶을 때 => `[ㄱ-ㅎ가-핳]`
- 한글 단어만 표현하고 싶을 때 => `[가-핳]`

```
[5] string = "Let's go jeju MT, KUBIG. 제주도 엠티 갑시다 여러분."
pattern = re.compile('[a-z]+')
print(pattern.findall(string))
```

```
['et', 's', 'go', 'jeju']
```

```
[8] string = "Let's go jeju MT, KUBIG. 제주도 엠티 갑시다 여러분."
result = re.sub('^[가-핳]', '', string)
print(result)
```

```
제주도엠티갑시다여러분
```

**Normalization:** 겉보기엔 다른 단어더라도, 의미가 같은 것끼리는 통일을 시켜주려는, 말 그대로 정규화

### 대/소문자 통합

```
[9] string = "kuBiG"
    print("upper:",string.upper())
    print("lower:",string.lower())

upper: KUBIG
lower: kubig
```

주의: '미국'을 나타내는 'US'와 '우리'를 나타내는 'us'는 구분되어야 함

### 표제어 추출 (Lemmatization)

단어의 품사 정보를 그대로 유지. 하지만 본래 단어의 품사 정보가 주어져야 함.  
어간 추출에 비해 단어의 형태를 적절하게 유지.

```
▶ from nltk.stem import WordNetLemmatizer
  lemmatizer = WordNetLemmatizer()

  print(lemmatizer.lemmatize("dies","v"))
  print(lemmatizer.lemmatize("watched","v"))
  print(lemmatizer.lemmatize("playing","v"))
  print(lemmatizer.lemmatize("playing"))

die
watch
play
playing
```

### 어간 추출 (Stemming)

규칙에 따라 접사 부분을 삭제하고 어간만 남김.  
품사 정보가 훼손될 수 있고, 단어의 형태가 어색할 수 있음.

```
▶ from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

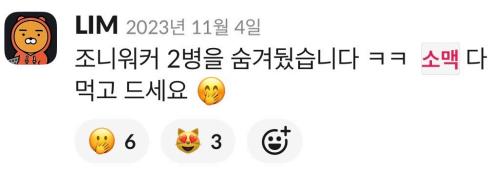
stemmer = PorterStemmer()
print(stemmer.stem("dies"))
print(stemmer.stem("diying"))

die
diy
```

## 2-5. Tokenization

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

### 토큰화



### 영어

띄어쓰기 단위로 자르더라도 토큰화 충분. 어퍼스트로피에 주의.

```
from nltk.tokenize import WordPunctTokenizer
text = "Kubig couldn't help drinking Lim's Johnny Walker."
print("토큰화1:",word_tokenize(text))
print("토큰화2:",WordPunctTokenizer().tokenize(text))

토큰화1: ['Kubig', 'couldn', "n't", 'help', 'drinking', 'Lim', "'s", 'Johnny', 'Walker', '.']
토큰화2: ['Kubig', 'couldn', "", 't', 'help', 'drinking', 'Lim', "", 's', 'Johnny', 'Walker', '.']
```

### 한국어

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

```
from konlpy.tag import Okt
okt = Okt()

text = "희준은 눈에 불을 켜고 교수님이 숨겨두신 조니워커를 찾았다"

print("형태소 분석:",okt.morphs(text))
print("품사 태깅:",okt.pos(text))
print("명사 추출:",okt.nouns(text))

형태소 분석: ['희준은', '눈', '에', '불', '을', '켜고', '교수', '님', '이', '숨겨', '두신', '조니워커', '를', '찾았다']
품사 태깅: [('희준은', 'Adjective'), ('눈', 'Noun'), ('에', 'Josa'), ('불', 'Noun'), ('을', 'Josa'), ('꺼고', 'Verb'), ('교수', 'Noun'), ('님', 'Noun'), ('이', 'Noun'), ('숨겨', 'Verb'), ('두신', 'Noun'), ('조니워커', 'Noun'), ('찾았다', 'Verb')]
명사 추출: ['눈', '불', '교수', '조니워커']
```

### 불용어(stopword) 제거

### 영어

Nltk 라이브러리에서 불용어 사전 제공

```
text = "Kubig couldn't help drinking Lim's Johnny Walker."
stop_words = set(stopwords.words('english'))
word_tokens = WordPunctTokenizer().tokenize(text)

result = []
[result.append(word) for word in word_tokens if not word in stop_words]

print("불용어 제거 전:",word_tokens)
print("불용어 제거 후:",result)

불용어 제거 전: ['Kubig', 'couldn', "", 't', 'help', 'drinking', 'Lim', "", 's', 'Johnny', 'Walker', '.']
불용어 제거 후: ['Kubig', "", 'help', 'drinking', 'Lim', "", 'Johnny', 'Walker', '.']
```

### 한국어

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

```
text = "희준은 눈에 불을 켜고 교수님이 숨겨두신 조니워커를 찾았다"
stop_words = "은 에 을 고 이 를"
stop_words = set(stop_words.split(' '))
word_tokens = okt.morphs(text)

result = []
[result.append(word) for word in word_tokens if not word in stop_words]

print("불용어 제거 전:",word_tokens)
print("불용어 제거 후:",result)

불용어 제거 전: ['희준은', '눈', '에', '불', '을', '꺼고', '교수', '님', '이', '숨겨', '두신', '조니워커', '를', '찾았다']
불용어 제거 후: ['희준은', '눈', '불', '꺼고', '교수', '님', '숨겨', '두신', '조니워커', '찾았다']
```

## 04 Encoding

정수 인코딩  
원핫 인코딩  
패딩

Encoding: 전처리된 토큰을 컴퓨터가 이해할 수 있게 숫자로 mapping해주는 과정

```
text = "서연이는 대외협력팀장이고 유민이는 학술기획팀장이다"
word_tokens = okt.morphs(text)

word2index = {}
for voca in word_tokens:
    if voca not in word2index.keys():
        word2index[voca] = len(word2index)
word2index['OOV'] = len(word2index) + 1
print("단어 집합:", word2index)

단어 집합: {'서연': 0, '이': 1, '는': 2, '대외협력': 3, '팀': 4, '장이': 5, '고': 6, '유민': 7, '학술': 8, '기획': 9, '다': 10, 'OOV': 12}
```

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

```
test_text = "원준이는 인사팀이고 대외협력팀과 친하게 지낸다"
test_word_tokens = okt.morphs(test_text)

encoded = []
for word in test_word_tokens:
    try:
        # 단어 집합에 있는 단어라면 해당 단어의 정수를 리턴.
        encoded.append(word2index[word])
    except KeyError:
        # 만약 단어 집합에 없는 단어라면 'OOV'의 정수를 리턴.
        encoded.append(word2index['OOV'])
encoded.append(encoded)
print(test_word_tokens)
print(encoded)

['원', '준', '이', '는', '인사', '팀', '이고', '대외협력', '팀', '과', '친하게', '지낸다']
[12, 12, 1, 2, 12, 4, 12, 3, 4, 12, 12, 12, [...]]
```

# 03 Basic of Word Representation

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

### 3-1. One Hot Encoding

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

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

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

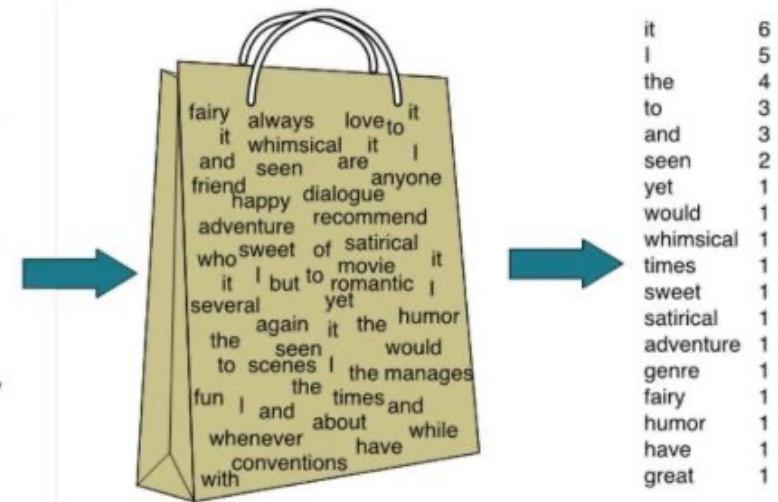
```
def build_bag_of_words(document):
    tokenized_document = okt.morphs(document)

    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!



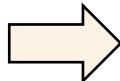
```
doc = "희준은 의정부에서 통학을 한다 의정부 통학 참 힘들다"
vocab, bow = build_bag_of_words(doc)
print('vocabulary :', vocab)
print('bag of words vector :', bow)

vocabulary : {'희준은': 0, '의정부': 1, '에서': 2, '통학': 3, '을': 4, '한다': 5, '참': 6, '힘들다': 7}
bag of words vector : [1, 2, 1, 2, 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: 희준이는 쿠빅을 좋아한다  
문서2: 우리는 쿠빅을 좋아한다  
문서3: 교수님은 인공지능을 좋아한다



	희준이는	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**: DTM 내의 각 단어마다 중요도를 가중치로 주는 행렬 표현 방식

- TF(d, t): 특정 문서 d에서의 특정 단어 t의 등장 횟수
- DF(t): 특정 단어 t가 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.

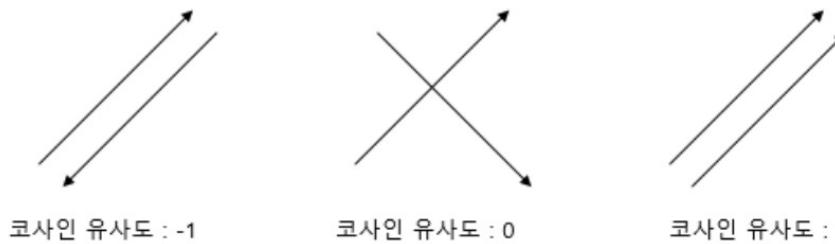
```
doc = ["희준이는 쿠빅을 정말 좋아한다",
       "우리는 쿠빅을 진짜 좋아한다",
       "교수님은 인공지능을 정말 사랑한다"]

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)
tfidf = tf*idf
tfidf = tfidf / np.linalg.norm(tfidf, axis=1, keepdims=True)
tfidf
```

	교수님은	사랑한다	우리는	인공지능을	정말	좋아한다	진짜	쿠빅을	희준이는
0	0.000000	0.000000	0.000000	0.000000	0.462191	0.462191	0.000000	0.462191	0.599281
1	0.000000	0.000000	0.559925	0.000000	0.000000	0.431838	0.559925	0.431838	0.000000
2	0.527426	0.527426	0.000000	0.527426	0.406774	0.000000	0.000000	0.000000	0.000000

### 3-5. Cosine Similarity



$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}}$$

```
doc = ["여빈이는 찍먹 좋아요",
        "희준이는 부먹 좋아요",
        "희준이는 부먹 좋아요 희준이는 부먹 좋아요"]

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

def cos_sim(A, B):
    return dot(A, B)/(norm(A)*norm(B))

print('문서 1과 문서2의 유사도 : ',cos_sim(tf.iloc[0,:], tf.iloc[1,:]))
print('문서 1과 문서3의 유사도 : ',cos_sim(tf.iloc[0,:], tf.iloc[2,:]))
print('문서 2와 문서3의 유사도 : ',cos_sim(tf.iloc[1,:], tf.iloc[2,:]))
```

{ 문서 1과 문서2의 유사도 : 0.3333333333333337  
문서 1과 문서3의 유사도 : 0.3333333333333337  
문서 2와 문서3의 유사도 : 1.000000000000002

	부먹	여빈이는	좋아요	찍먹	희준이는	
0	0	1	1	1	0	
1	1	0	1	0	1	
2	2	0	2	0	2	

# 04 Word2Vec, FastText

about Word Embedding

## 4-1. What is Word Embedding?

희소표현  
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, Glove이 그 대표적 예시

## 4-2. What is Word2Vec?

고려대학교+라이벌

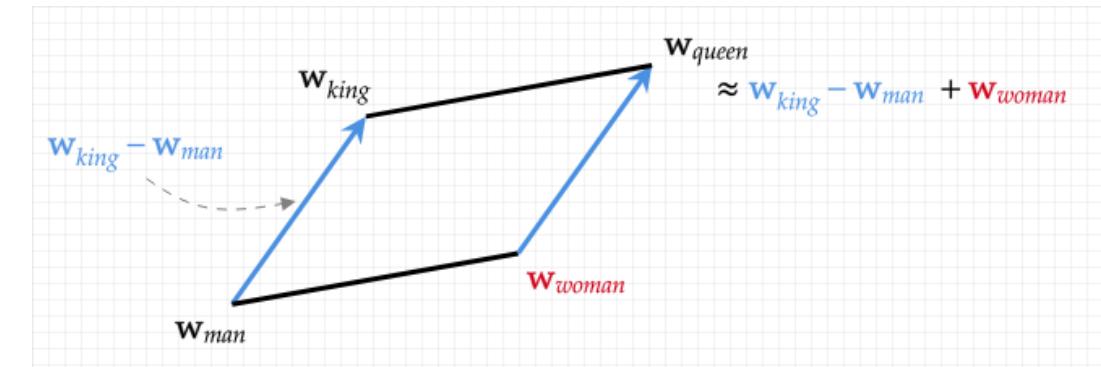
QUERY

+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)

해원이는 кубик \_\_\_\_\_ 계정을 담당하는 홍보팀장이야

Output: 중심 단어(center word)

### Skip-Gram

중심에 있는 단어로 주변에 있는 단어를 예측

Input: 중심 단어(center word)

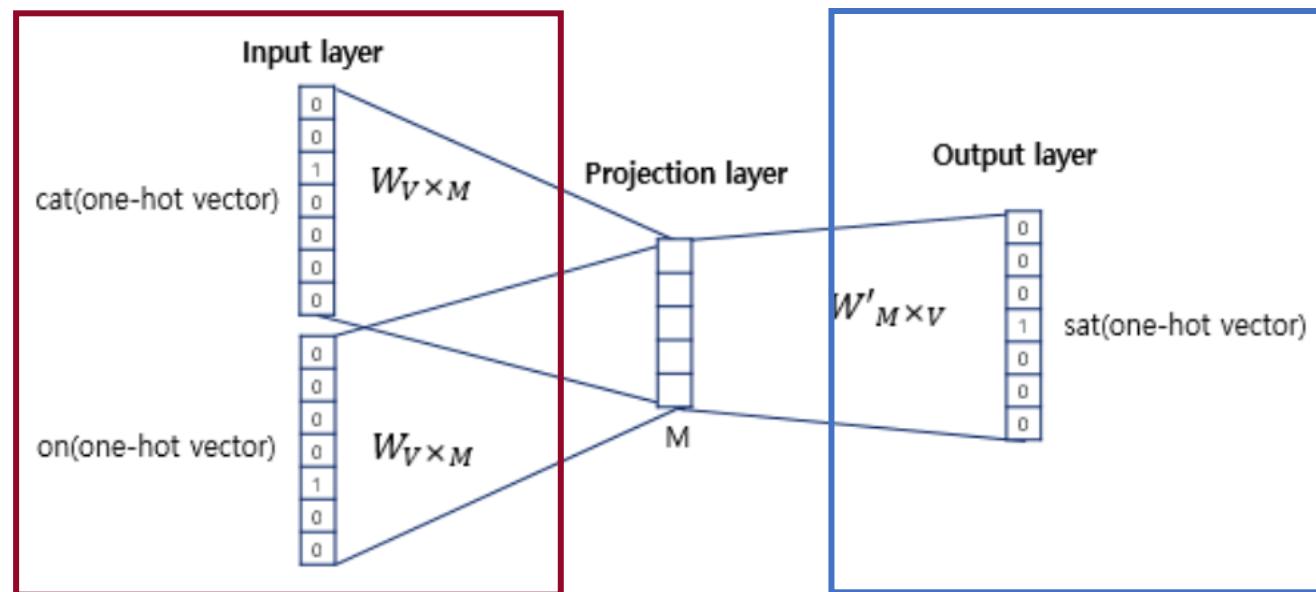
해원이는 \_\_\_\_\_ 인스타그램 \_\_\_\_\_ 담당하는 홍보팀장이야

Output: 주변 단어(context word)

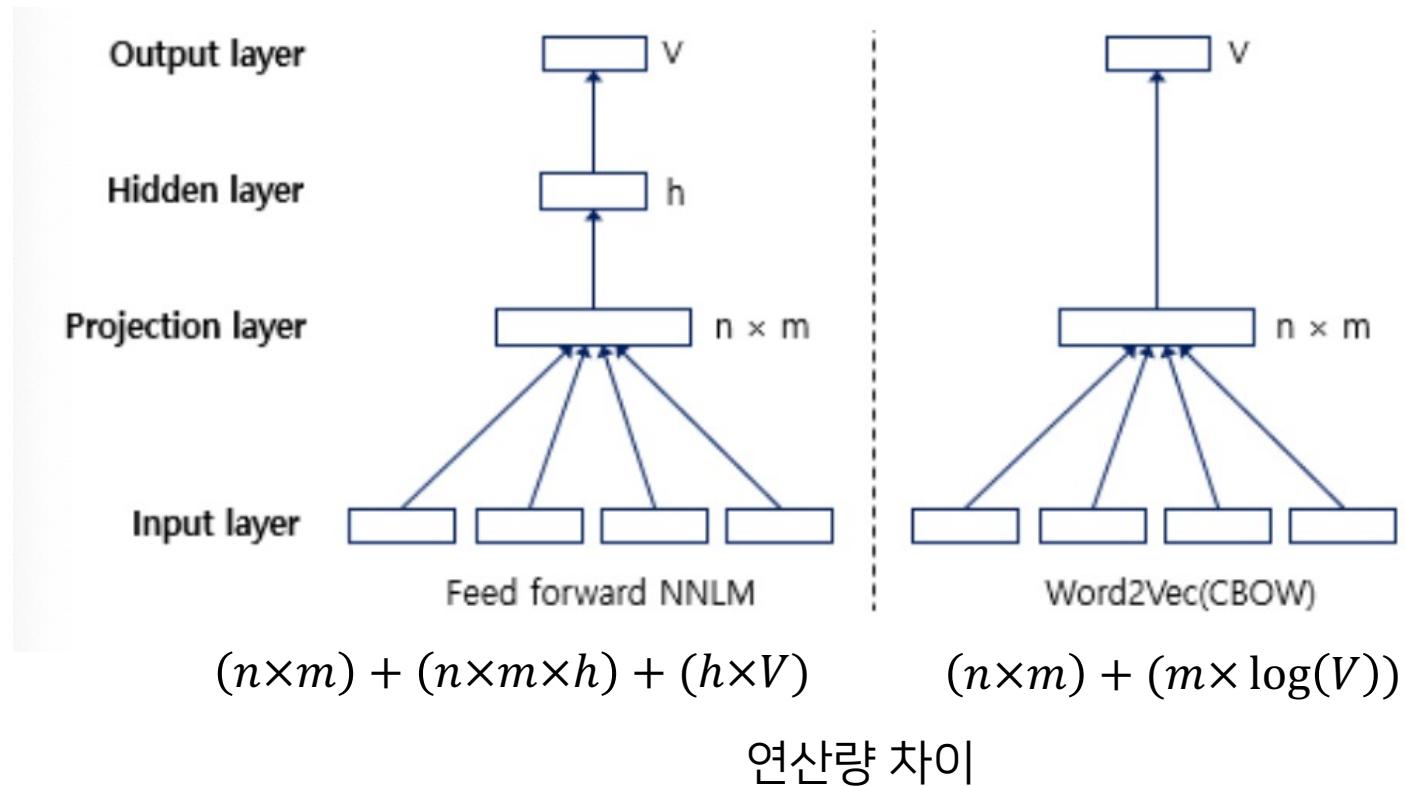
**CBOW(Continuous Bag of Words)**: 주변 단어(context words)로 중심 단어(center word)를 예측하는 학습 방식

Example: "The fat cat sat on the mat"

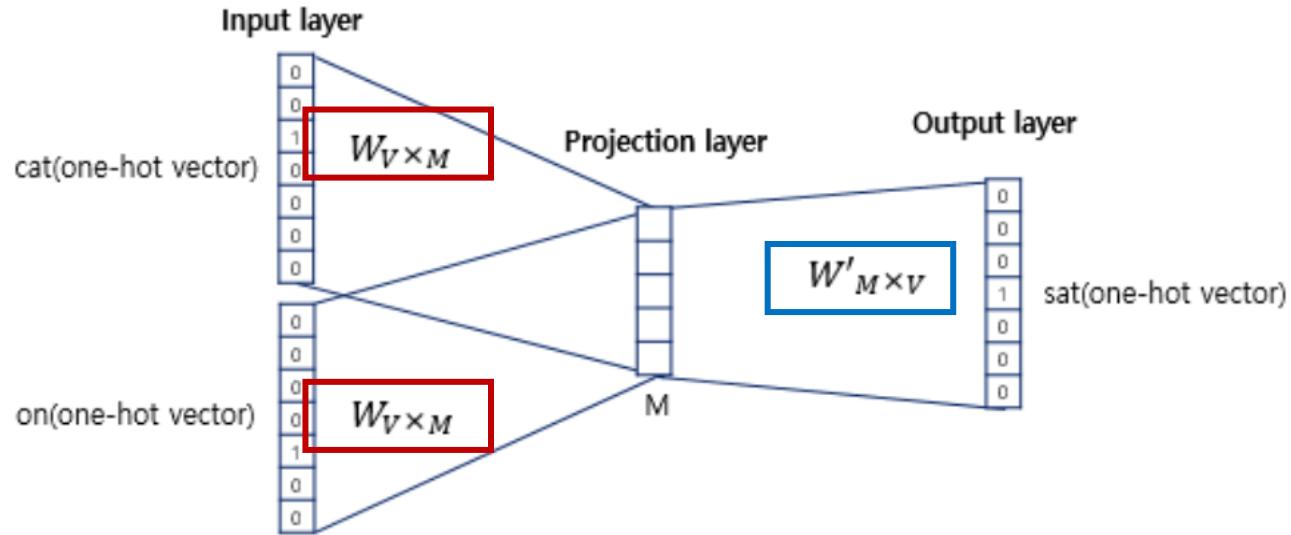
GOAL: window size=1 일 때 **sat**을 예측하라



Q. neural net이라기엔, 우리가 아는 것들보다 너무 단순한 layer인데?  
=> 기존의 NNLM, RNNLM의 nonlinear, hidden layer들이 비효율적이라 projection layer만 쌓은 간단한 형태



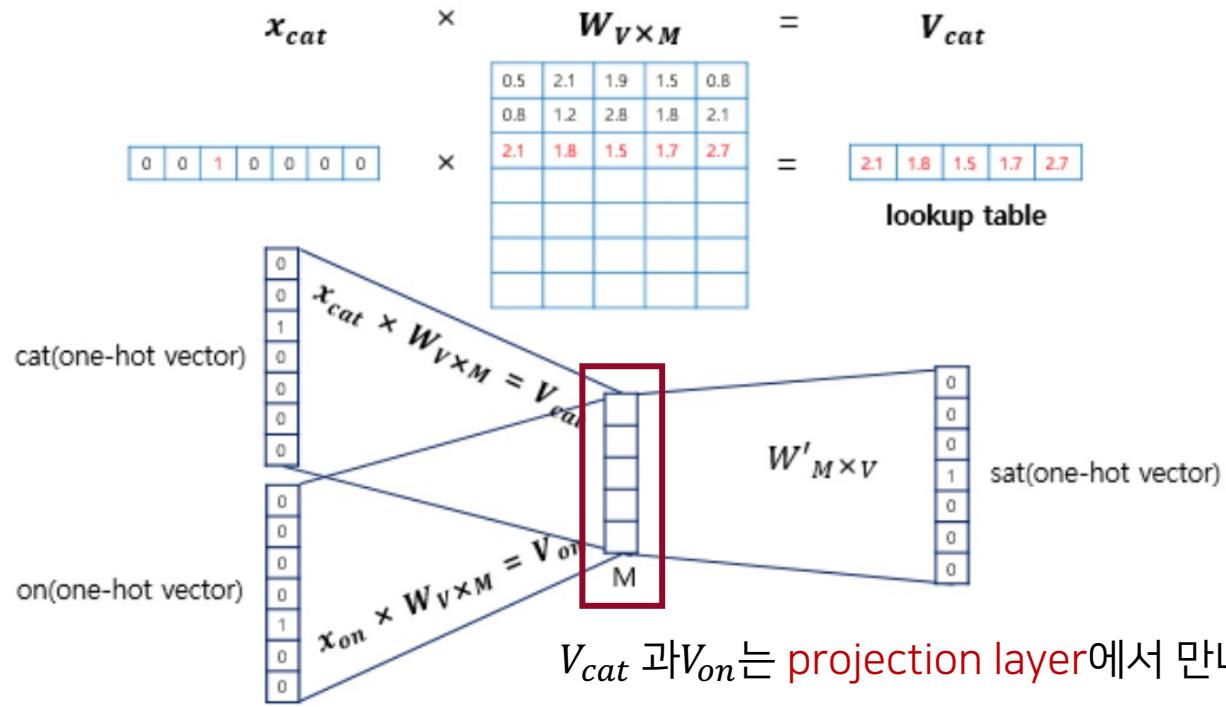
Example: "The fat cat sat on the mat"



Input으로 들어오는 one hot vector의 크기는  $V$ (=단어집합 크기)  
 Projection layer의 크기는  $M$ (=임베딩 벡터 차원)

가중치 행렬  $W, W'$ 의 초기값은 랜덤

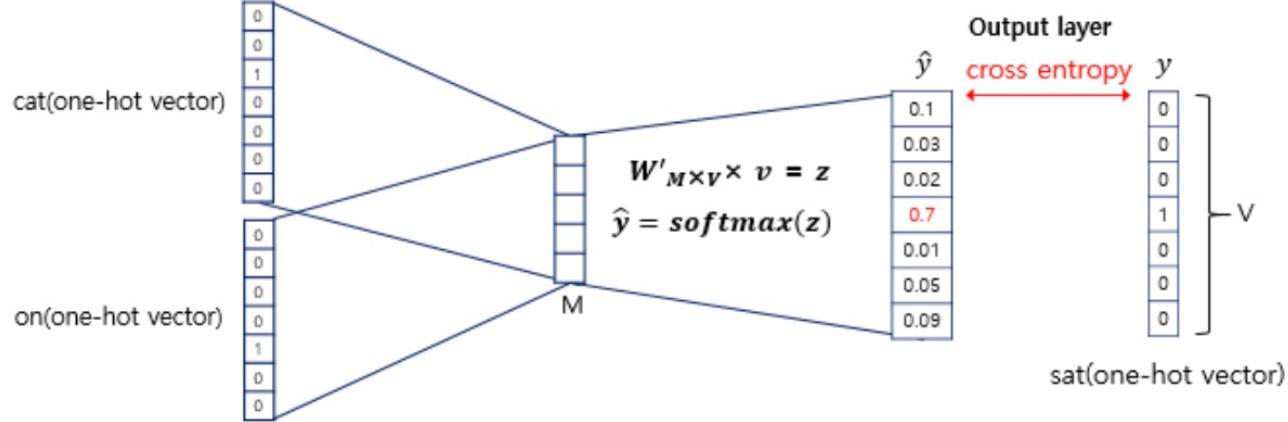
중심 단어를 더 잘 맞히기 위한 학습 과정에서 가중치도 학습됨  
 학습을 마친 후  $W$ 의 행 벡터가 해당 단어의 embedding vector가 됨!



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

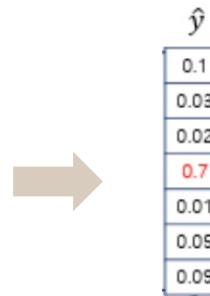
$V_{cat}$  과  $V_{on}$ 는 projection layer에서 만나 평균을 구하게 됨.

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



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

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



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

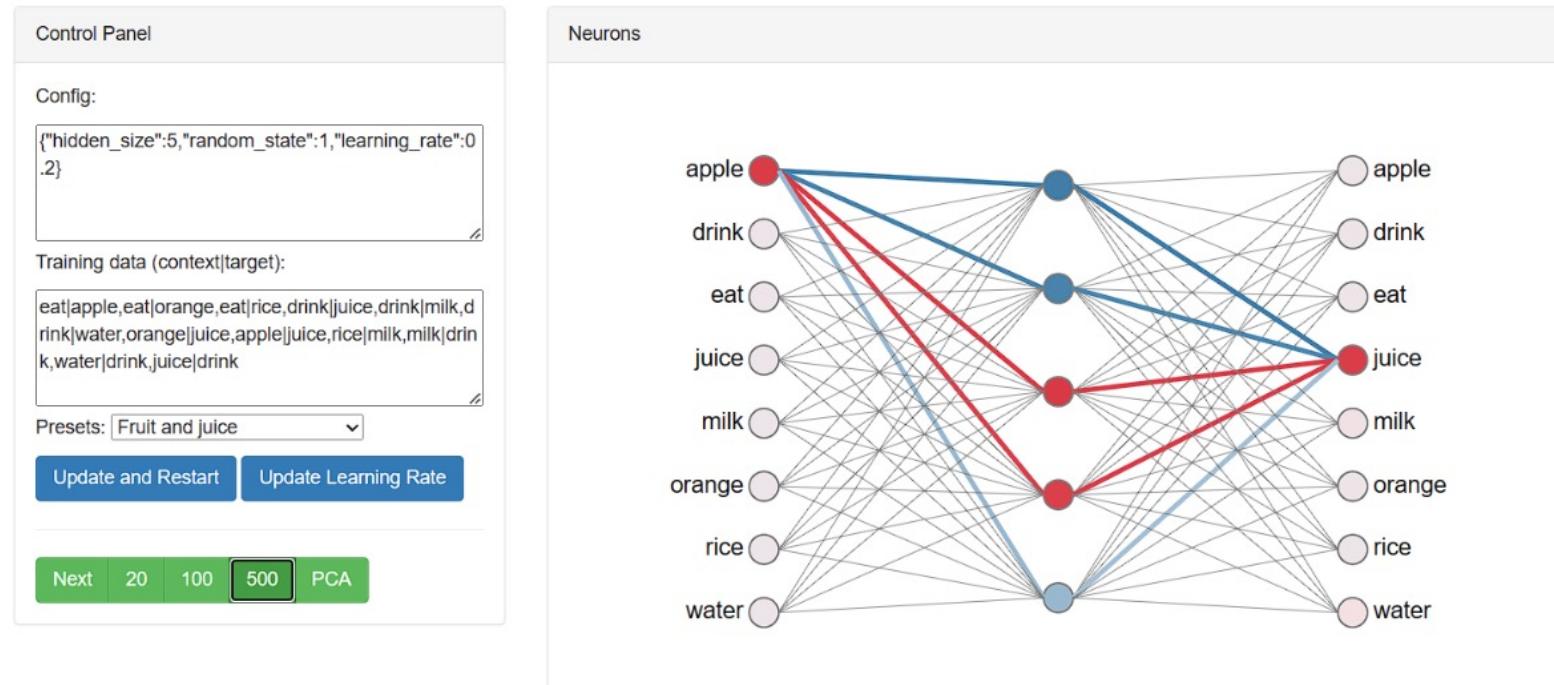
$$cost(\hat{y}, y) = - \sum_{j=1}^V y_j \log(\hat{y}_j)$$

Loss function으로 cross entropy 사용. 우변을 최소화하는 방향으로 학습  
이 때 발생한 오차로 backpropagation하여 가중치  $W$  조정!  
 $W$ 의 행 벡터가 해당 단어의 embedding vector가 됨!

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

### wevi: word embedding visual inspector

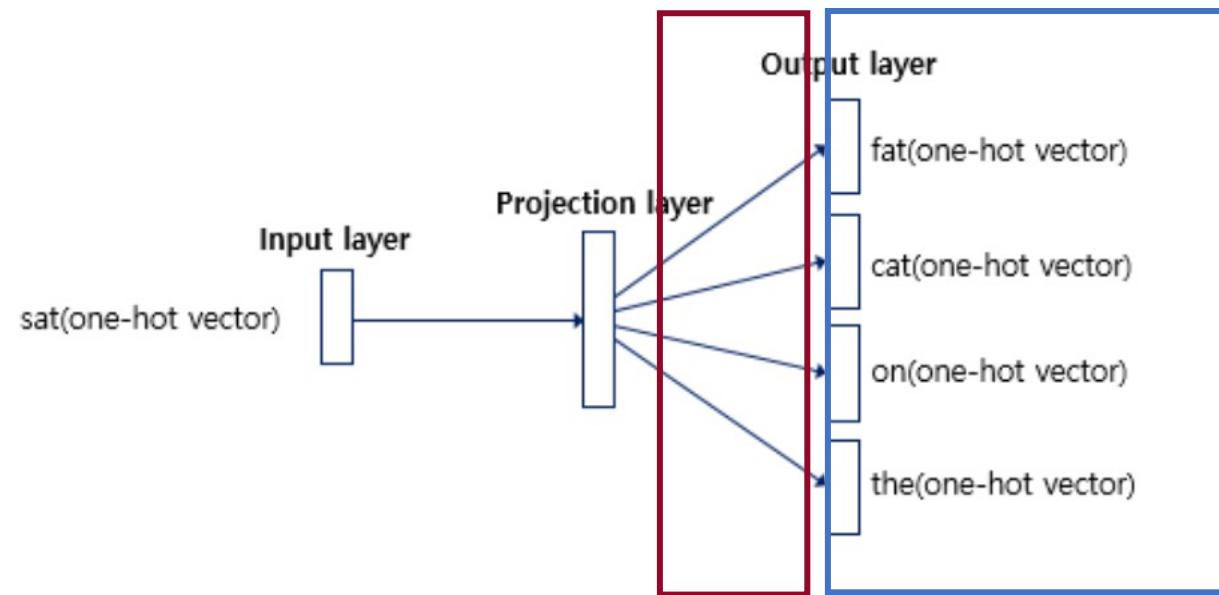
Everything you need to know about this tool - Source code



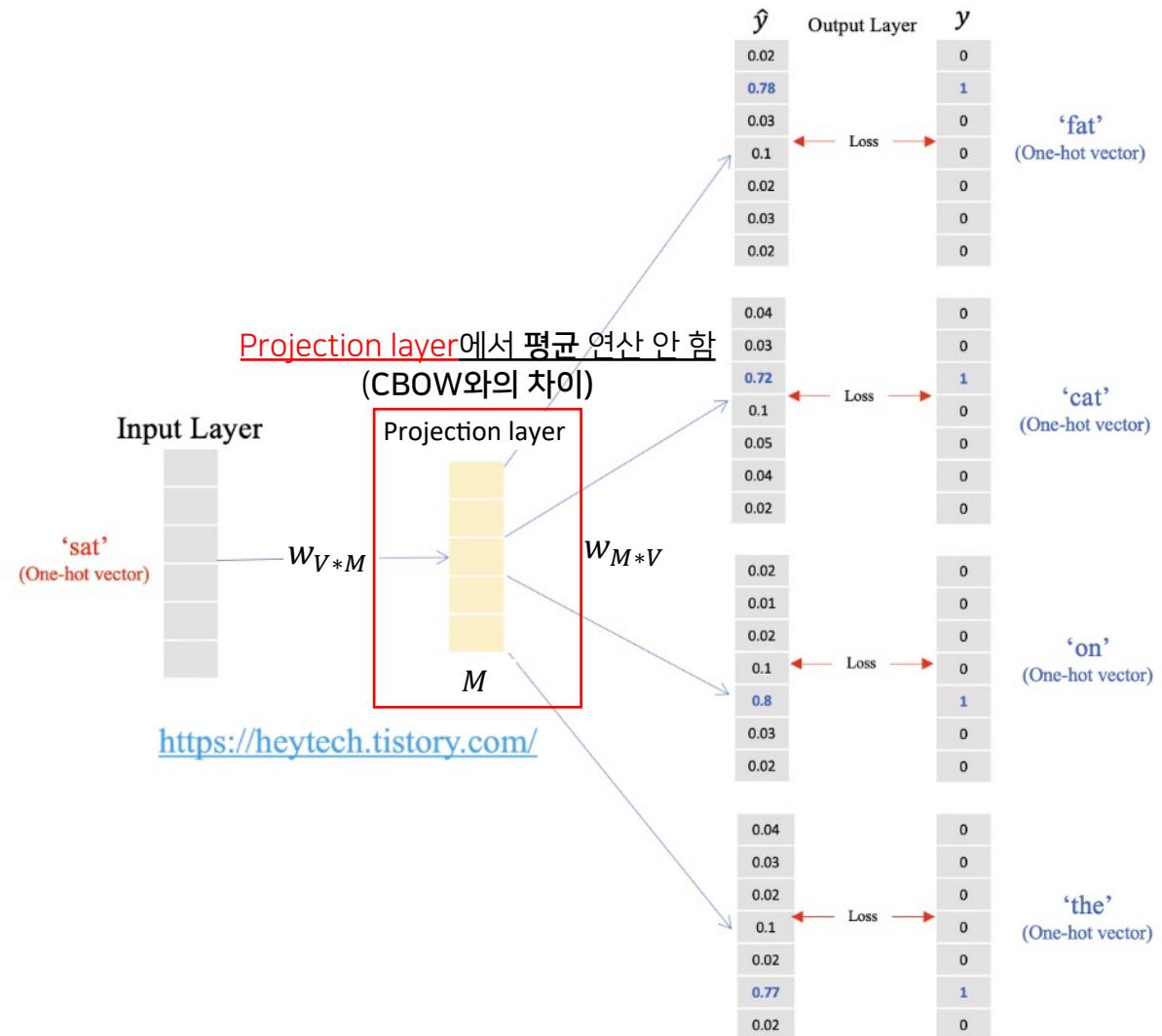
**Skip-gram:** 중심 단어(center word)로 주변 단어(context words)를 예측하는 학습 방식

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

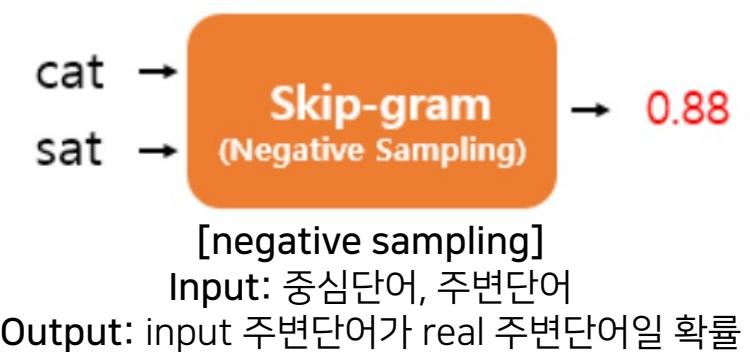
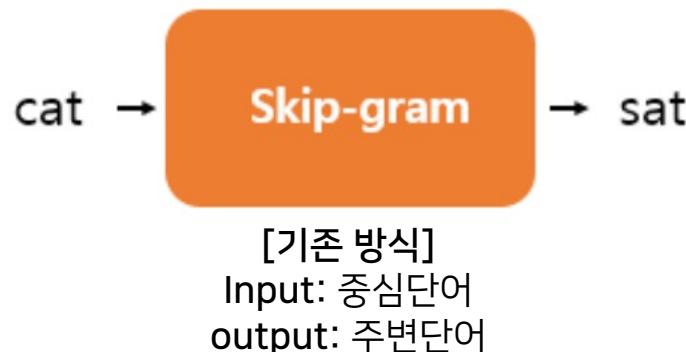
Model	Vector Dimensionality	Training words	Accuracy [%]			Training time [days x CPU cores]
			Semantic	Syntactic	Total	
NNLM	100	6B	34.2	64.5	50.8	14 x 180
CBOW	1000	6B	57.3	68.9	63.7	2 x 140
Skip-gram	1000	6B	66.1	65.1	65.6	2.5 x 125



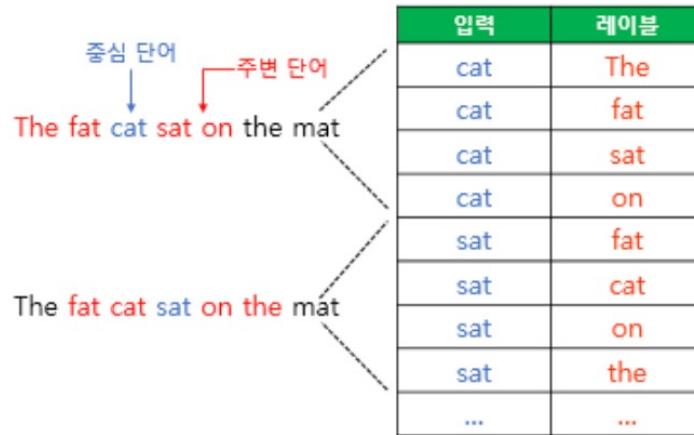
따라서 Word2vec에선 Skip-gram을 주로 사용하는데,  
이에 더해 **Negative Sampling**까지 추가로 사용

CBOW보다 Skip-gram의 성능이 보다 우수함

**Negative Sampling**: 속도 문제를 개선하기 위해 다중 분류 문제를 이진 분류로 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



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

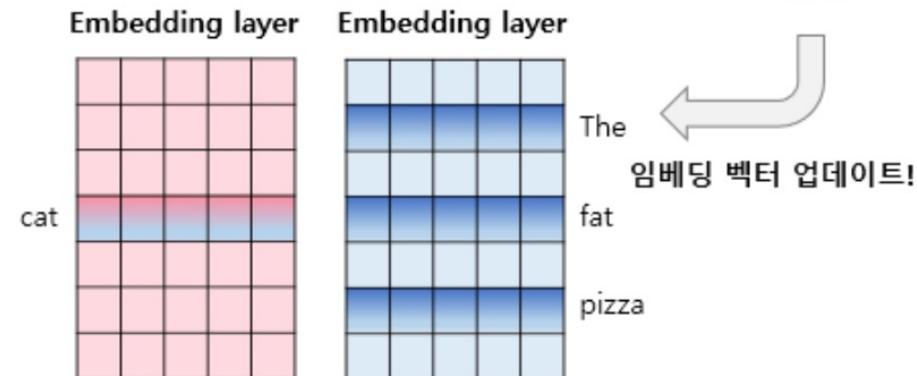
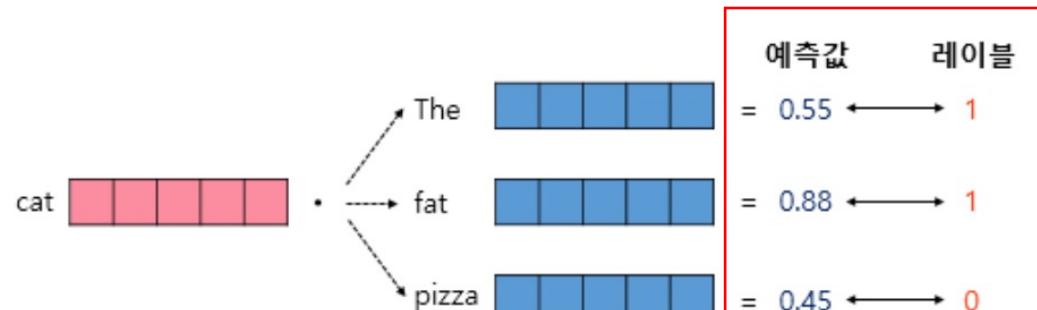
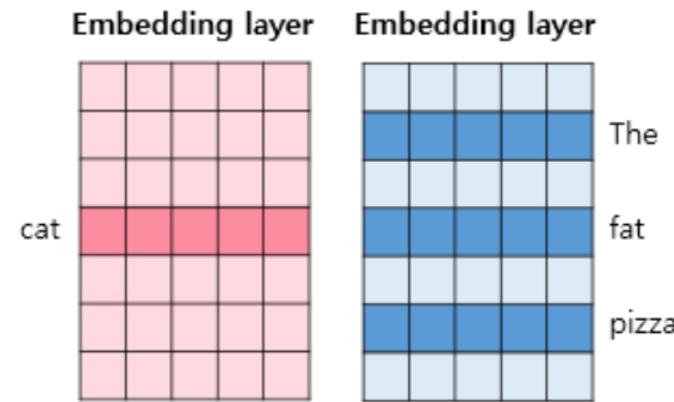
이웃관계가 아닌 경우,  
즉 부정(negative)의 경우(레이블=0)도 학습을 해야 하기 때문에  
negative sampling이란 이름이 붙음

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: <ti, tig, ige, ger, er>

4-gram: <tig, tige, iger, ger>

5-gram: <tige, tiger, iger>

6-gram = <tiger, tiger>

special token = <tiger>

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

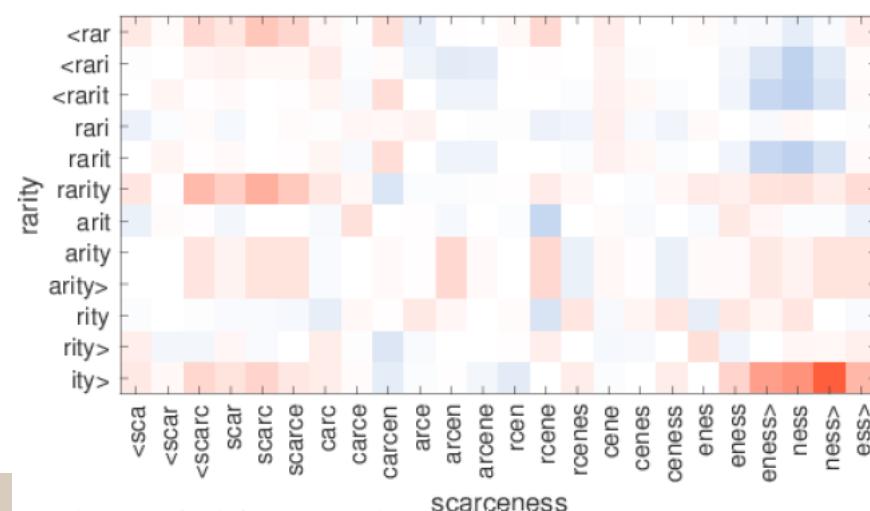
이것이 OOV로 처리되지 않고, elec, fish 등으로 쪼개져 유의미한 벡터로 쓰일 수 있다.



tiger의 gram set

보통 한 단어의 gram set은

3~6 gram으로 이루어진다



[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  $F$ 를 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.

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

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

$$\therefore \text{준동형 성질} = F((w_i - w_j)^T w_k) \times F(w_j^T w_k)$$

중심 단어  $w$ 와 주변 단어  $w^{\sim}$ 는 무작위로 선택됨.  
즉 서로 자리가 바뀌더라도 값이 같아야 함

이를 만족하려면 함수  $F$ 는 준동형(Homomorphism)이어야 함

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

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

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

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

위에서 구한 값

## 5-2. Loss Function of GloVe

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

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

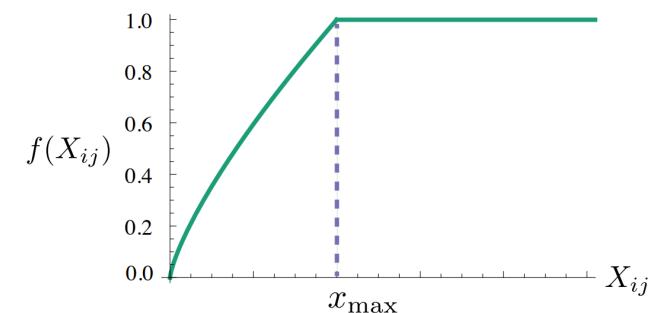


Figure 1: Weighting function  $f$  with  $\alpha = 3/4$ .

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

Remind ) GloVe의 목표

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

# 06 Announcement

Week1 예습과제 Review, week2 예복습 과제 안내, week3 진도 안내

## 6-1. 우수 예습과제 Review

서연님

1주차  
예습과제1

Text preprocessing

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(읽어오시길 권장 드립니다!)

#### [딥러닝을 이용한 자연어 처리 입문]

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

#### [밑바닥부터 시작하는 딥러닝2]

- Ch5. 순환신경망

E.O.D  
수고하셨습니다!