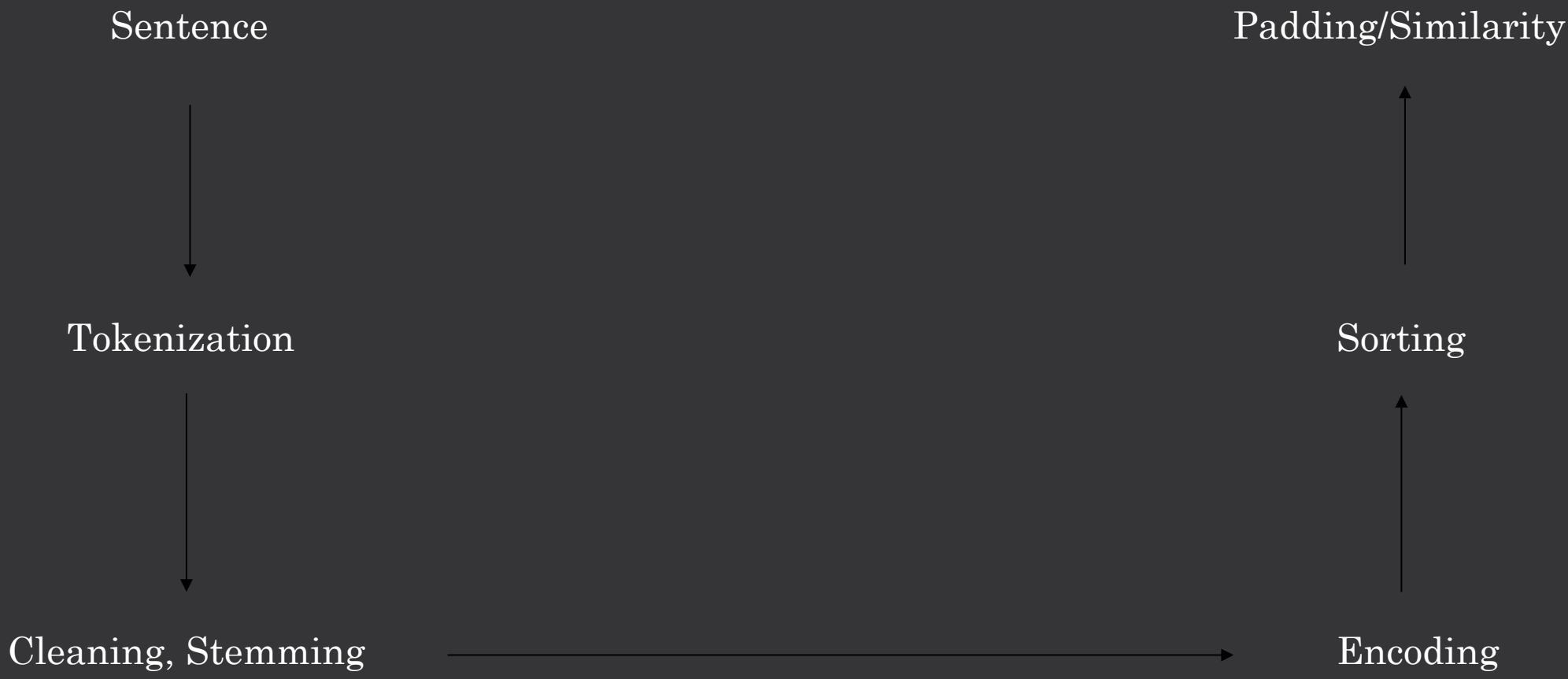
자연어 처리 기초

권범윤

목차

자연어 처리 기초


Sentence
Tokenization
Cleaning, Stemming
Encoding
Sorting
Padding/Similarity

-Tokenization

주어진 문장에서 “ 의미 부여 " 가 가능한 단위를 찾는다. (기본적으로 띄어쓰기로 나눔)

Ex) Machine Learning methods including ANN have been applied in compound activity prediction for a long time.

문장 토큰화 : 문장 단위로 의미를 나누기 (기본적으로 마침표 기준으로 나눔)

Ex) My professor in looking for a student who is fluent on the Python Programming. Yet, he also Wants a person also capable of dealing with Pytorch.

-Tokenization

*토큰화가 어려운 예시

Ex) 어제 삼성 라이온즈가 기아 타이거즈를 5:3으로 꺾고 위닝 시리즈를 거두었습니다.

-> 구두점이나 특수문자를 전부 제거하는 작업만으로는 불가능하다.

* 표준 토큰화 : Treebank Tokenization

```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
text = "Model-based RL don't need a value function for the policy"
print(tokenizer.tokenize(text))
```

```
['Model-based', 'RL', 'do', "n't", 'need', 'a', 'value', 'function', 'for', 'the', 'policy']
```

-Cleaning(정제)

목적에 맞추어서 노이즈를 제거

1. 대문자 vs 소문자 ex) US us , Mike mike

2. 출현 빈도수가 적은 단어의 제거 ex) Floras and faunas (Plants and animals)

3. 길이가 짧은 단어, 지시(대) 명사, 관사의 제거 ex) I , a(n)

-Stemming(추출)

어간(Stem) : 단어의 의미를 담은 핵심
접사(Affix) : 단어에 추가 용법을 부여

Ex) lectures, playing, kindness

Porter Algorithm : 대포적인 Stemming 방법

Formalize

Tolerance

Electrical

organization

-Lemmatization(표제어 추출)

is , are -> be

having -> have

어간 추출 vs 표제어 추출

표제어 추출은 단어의 품사 정보 O
어간 추출은 품사 정보 X

Having -> have (표제어 추출)
Having -> hav (어간 추출)

- 불용어(Stopword)

문장에서 대세로 작용하지 않는, 중요도가 낮은 단어 제거

1. 불용어 (stopword) 목록을 받아온다.
2. 정제할 문장을 토큰화(tokenize) 한다.
3. 토큰화된 각 단어마다:
4. 단어가 불용어 목록에 없는 경우 -> 정제 결과에 추가
5. 단어가 불용어 목록에 있는 경우 -> Pass

Ex) We Should all study hard for the exam.

- 불용어(Stopword)

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

input_sentence = "We Should all study hard for the exam."
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(input_sentence)
result = []
for w in word_tokens:
    if w not in stop_words:
        result.append(w)

print(word_tokens)
print(result)
```

```
['We', 'Should', 'all', 'study', 'hard', 'for', 'the', 'exam', '.']
['We', 'Should', 'study', 'hard', 'exam', '.']
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

-Integer Encoding(정수 인코딩)

1. 문장의 토큰화 – 불용어 및 대문자 제거 과정을 거친다.
2. 빈 단어 dictionary vocab={}를 만든다.
3. 토큰화 된 각 단어에 대해서 :
4. 단어가 vocab에 속해 있는 경우 -> vocab[단어] +=1
5. 단어가 vocab에 속해 있지 않은 경우 -> vocab[단어] = 0

-Integer Encoding(정수 인코딩)

```
raw_text = "A barber is a person. a barber is good person. a barber is huge person. he Knew A Secret! The Secret He Kept is huge secret. Huge secret. His barber kept his word. a barber kept
# 문장 토큰화
sentences = sent_tokenize(raw_text)
print(sentences)

vocab = {}
preprocessed_sentences = []
stop_words = set(stopwords.words('english'))

for sentence in sentences:
    # 단어 토큰화
    tokenized_sentence = word_tokenize(sentence)
    result = []
    for word in tokenized_sentence:
        word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄인다.
        if word not in stop_words: # 단어 토큰화 된 결과에 대해서 불용어를 제거한다.
            if len(word) > 2: # 단어 길이가 2이하인 경우에 대하여 추가로 단어를 제거한다.
                result.append(word)
                if word not in vocab:
                    vocab[word] = 0
                vocab[word] += 1
    preprocessed_sentences.append(result)
print(preprocessed_sentences)
print('단어 집합 :',vocab)
vocab_sorted = sorted(vocab.items(), key = lambda x:x[1], reverse = True)
print(vocab_sorted)
word_to_index = {}
i = 0
for (word, frequency) in vocab_sorted :
    if frequency > 1 : # 빈도수가 작은 단어는 제외.
        i = i + 1
        word_to_index[word] = i
print(word_to_index)
```

```
단어 집합 : {'barber': 8, 'person': 3, 'good': 1, 'huge': 5, 'knew': 1, 'secret': 6, 'kept': 4, 'word': 2, 'keeping': 2, 'driving': 1, 'crazy': 1, 'went': 1, 'mountain': 1}
[('barber', 8), ('secret', 6), ('huge', 5), ('kept', 4), ('person', 3), ('word', 2), ('keeping', 2), ('good', 1), ('knew', 1), ('driving', 1), ('crazy', 1), ('went', 1), ('mountain', 1)]
{'barber': 1, 'secret': 2, 'huge': 3, 'kept': 4, 'person': 5, 'word': 6, 'keeping': 7}
```

-Padding(Zero-Padding)

1. 문장들에 대해서 정수 인코딩을 거친다.

2. 각 문장에 대해서 해당 문장의 길이가 가장 긴 문장의 길이보다 작을 경우 나머지 부분에 0 을 추가

[1,3]	->	[1,3,0,0,0,0]
[2,4,8]	->	[2,4,8,0,0,0]
[8,6,1,9]	->	[8,6,1,9,0,0]
[4,7]	->	[4,7,0,0,0,0]
[8,4,8,1,3,7]	->	[8,4,8,1,3,7]

-One-hot Encoding

0	(apple)	->	[1,0,0,0,0,0]
1	(like)	->	[0,1,0,0,0,0]
2	(banana)	->	[0,0,1,0,0,0]
3	(eat)	->	[0,0,0,1,0,0]
4	(hate)	->	[0,0,0,0,1,0]
5	(fruit)	->	[0,0,0,0,0,1]

Problem ?
1. 저장 공간
2. 유사도

-Word2vec Encoding

- 단어의 유사성을 인코딩에 반영

- 인코딩 벡터가 비슷하다
  = 단어가 유사하다.

Ex)

Korean
Japanese
English
Math
Science
Social Studies

-Statistical Language Model(SLM)

- 전통적인 접근 방법을 이용하는 언어 모델 (카운트 기반의 접근)

An adorable little boy 다음에 is 가 나올 확률 :

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

-한계점

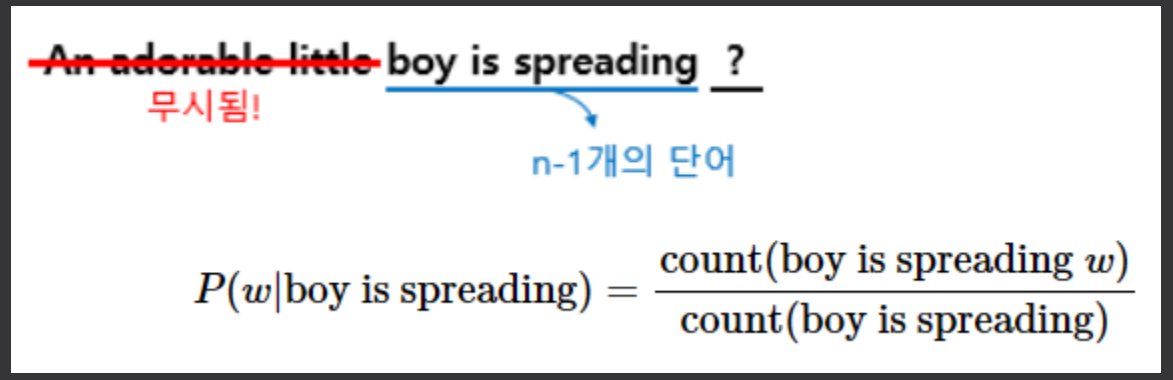
언어 모델의 확률 분포를 현실에서의 확률 분포와 같게 하려면 방대한 데이터가 필요하다.

- N-gram 언어 모델(N-gram Language model)

- N-gram 기법을 이용한 언어 모델

Ex) An adorable little boy is spreading smiles

Unigrams : an, adorable, little, boy, is, spreading, smiles
Bigrams : an adorable, adorable little, little boy, boy is, is spreading, spreading smiles
Trigrams : an adorable little, adorable little boy, little boy is , boy is spreading, is spreading smiles



-한계점

1. 일부만 참고함으로써 확률을 높일 수는 있지만 완전히 극복할 수는 없다.

2. N은 선택하는 것은 trade-off 문제
   - n을 높일 수록 정확해지지만 모델 사이즈가 커진 다는 문제 발생. (최대 5가 넘지 않도록 권장)

-Perplexity(PPL)

$$PP(W) = P(w_1 w_2 ... w_N)^{-\frac{1}{N}}$$

$$= \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$$

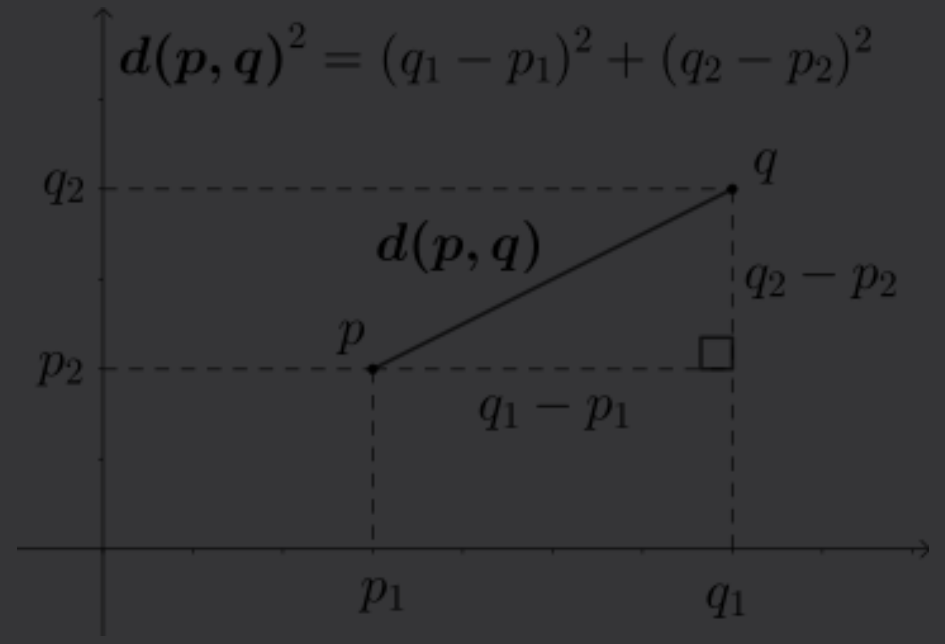
Model	Perplexity
Interpolated Kneser-Ney 5-gram (Chelba et al., 2013)	67.6
RNN-1024 + MaxEnt 9-gram (Chelba et al., 2013)	51.3
RNN-2048 + BlackOut sampling (Ji et al., 2015)	68.3
Sparse Non-negative Matrix factorization (Shazeer et al., 2015)	52.9
LSTM-2048 (Jozefowicz et al., 2016)	43.7
2-layer LSTM-8192 (Jozefowicz et al., 2016)	30
Ours small (LSTM-2048)	43.9
Ours large (2-layer LSTM-2048)	39.8

-코사인 유사도(Cosine Similarity)

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

-	바나나	사과	저는	좋아요
문서1	0	1	1	1
문서2	1	0	1	1
문서3	2	0	2	2

-이 외의 유사도 기법

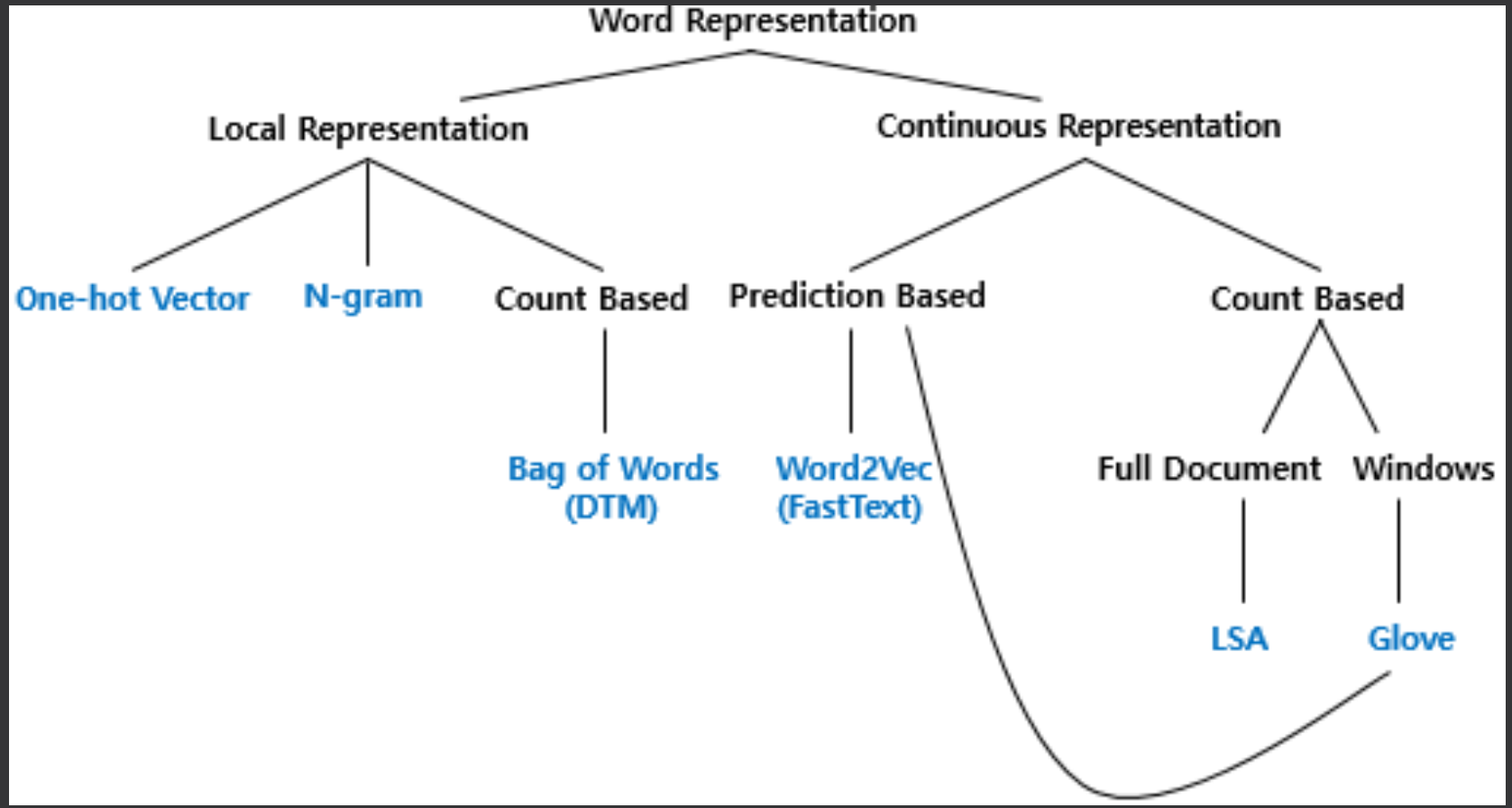


유클리드 유사도

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

자카드 유사도

-단어 표현의 카테고리화


Word Representation
Local Representation
Continuous Representation
One-hot Vector
N-gram
Count Based
Prediction Based
Count Based
Bag of Words
(DTM)
Word2Vec
(FastText)
Full Document
Windows
LSA
Glove

-Bag of Words(BoW)

- 단어들의 순서는 전혀 고려하지 않고 , 단어들의 출현 빈도에만 집중하는 데이터 수치화 표현 방법

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0

-Document-Term Matrix(DTM)

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 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

※ 한계점 ※

1) 희소 표현(대부분의 값이 0인 표현, 많은 양의 저장 공간과 계산 복잡도를 요구)

2) 단순 빈도 수 기반 접근

-TF-IDF (Term Frequency-Inverse Document Frequency)

DTM 내의 각 단어들마다 중요한 정도를 가중치로 부여하는 방법

tf(d ,t) : 특정 문서 d에서의 특정 단어 t의 등장 횟수 (DTM과 동일)

df (t) : 특정 단어 t가 등장한 문서의 수

idf(d, t) : df(t)에 반비례하는 수

$$TF(t,d) = \frac{number\ of\ times\ t\ appears\ in\ d}{total\ number\ of\ terms\ in\ d}$$

$$IDF(t) = log\frac{N}{1+df}$$

$$TF-IDF(t,d) = TF(t,d) * IDF(t)$$

TF-IDF(Term Frequency-Inverse Document Frequency)

문서1 : 먹고 싶은 사과

문서2 : 먹고 싶은 바나나

문서3 : 길고 노란 바나나 바나나

문서4 : 저는 과일이 좋아요

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

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
싶은	ln(4/(2+1)) = 0.287682
저는	ln(4/(1+1)) = 0.693147
좋아요	ln(4/(1+1)) = 0.693147

-	과일이	길고	노란	먹고	바나나	사과	싶은	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

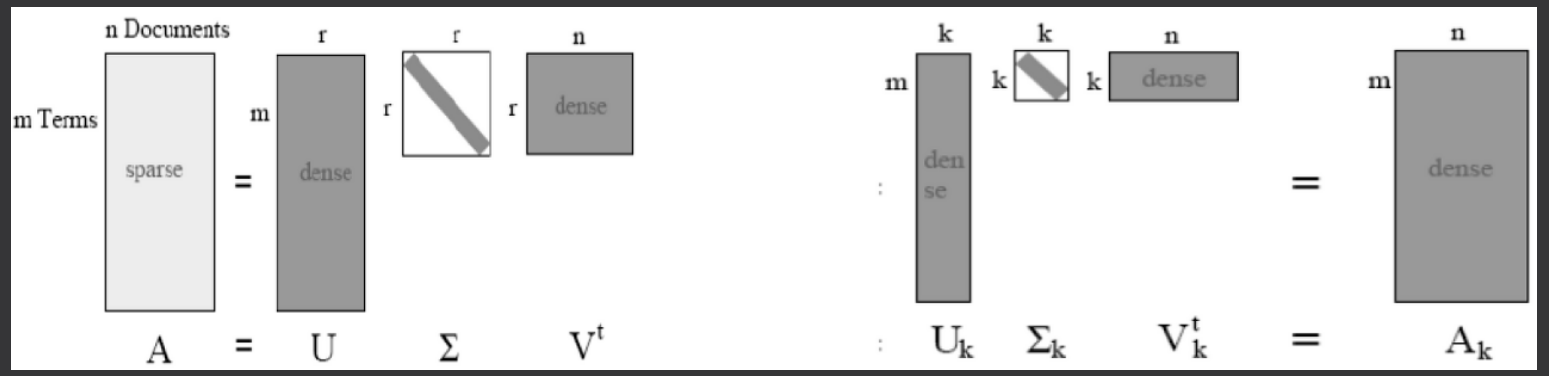
-워드 임베딩(Word Embedding)

-기존 정수 인코딩(Integer Encoding)의 한계?
- 단어 사이의 연관성을 파악하기 어려움

- 원 핫 인코딩 (One-hot Encoding)의 한계?
  - 메모리
  - 희소 표현(Sparse Representation)

- 밀집 표현(Dense Representation)
  - One-hot Encoding의 희소 표현 문제를 보완
  - 벡터의 차원을 원하는 대로 설정할 수 있음
  - 데이터를 이용해서 표현을 학습함.

-잠재 의미 분석(Latent semantic Analysis, LSA)



Full SVD

Truncated SVD

출력 예시

```
[[0 0 0 1 0 1 1 0 0]
 [0 0 0 1 1 0 1 0 0]
 [0 1 1 0 2 0 0 0 0]
 [1 0 0 0 0 0 0 1 1]]
[[ 0.   -0.17 -0.17  1.08  0.12  0.62  1.08 -0.   -0.  ]
 [ 0.    0.2   0.2   0.91  0.86  0.45  0.91  0.    0.  ]
 [ 0.    0.93  0.93  0.03  2.05 -0.17  0.03  0.    0.  ]
 [ 0.    0.    0.    0.    0.    0.    0.    0.    0.  ]]
```

LSA와 같은 통계 기반 방식

- 장점 : 말뭉치 전체의 통계 정보를 활용
- 단점 : 단어/문서 간 유사도 측정이 힘듦.

-워드투벡터(Word2Vec)

1. CBOW (Continuous Bag Of Words)
    - 주변 단어를 활용해 중심 단어를 예측

2. Skip - Gram
    - 중심 단어를 활용해 주변 단어를 예측

Word2Vec과 같은 임베딩 방식

- 장점 : 유의어와 같이 단어 간 유사도 기반의 Task에 강함

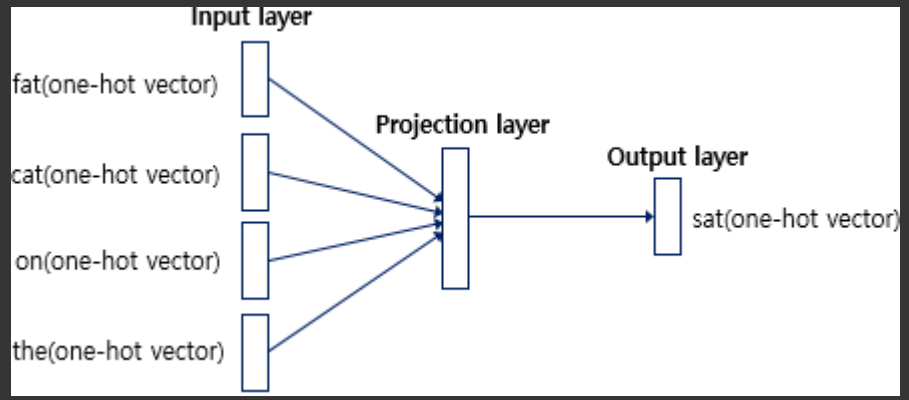
- 단점 : 주변 일부 단어와의 연관성만 학습하므로, 전체의 정보를 반영하지 못함.

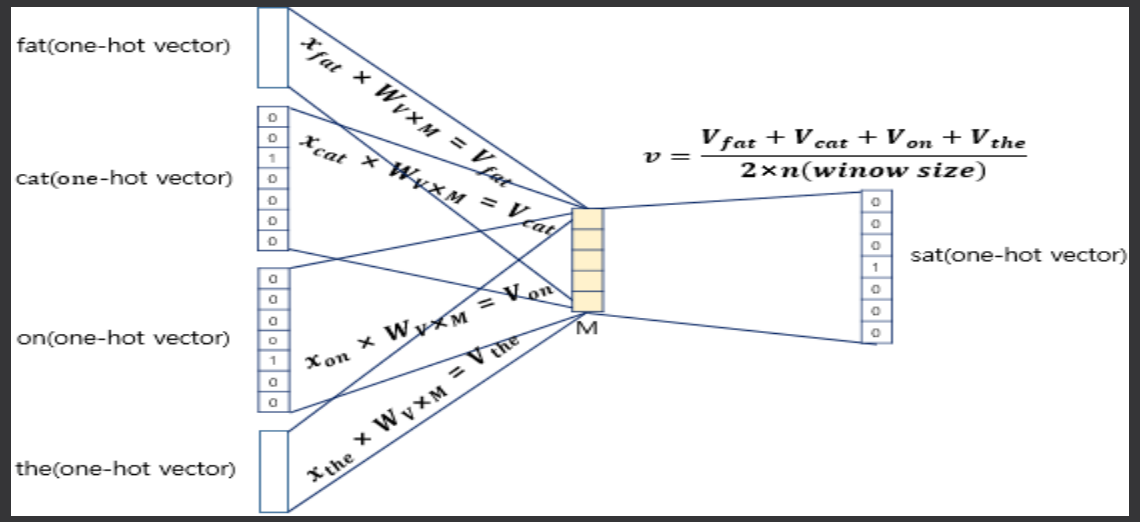
CBOW(Continuous Bag Of Words)

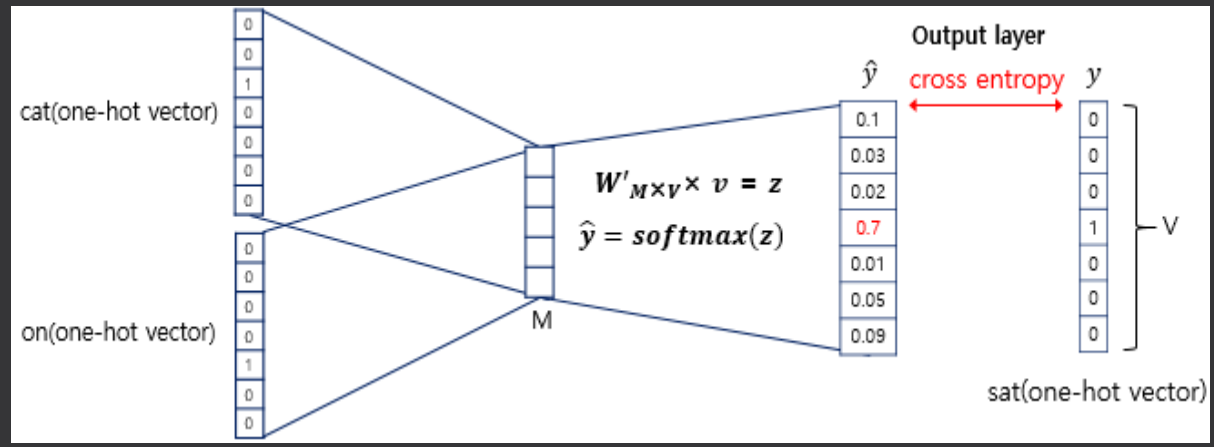
-CBOW (Continuous Bag Of Words)

- 주변 단어를 활용해 중심 단어를 예측

The fat cat sat on the mat.







-Skip - gram

- Skip – Gram
  - 중심 단어가 주어졌을 때 주변단어가 나타날 확률이 높아지도록 학습

- 전반적으로 Skip-gram 방식이 CBOW 보다 성능이 좋음.
- 한 번에 여러 단어를 예측하기 때문에 비효율적, 최근에는 negative sampling 방법 사용