

# NLP: Natural Language Processing

단어 임베딩; 텍스트 유사도



## 단어 임베딩

단어 임베딩으로 얻은 결과가 밀집된 정보를 가지고 있고 공간의 낭비가 적다는 것을 알 수 있습니다. 이런 결과가 가능한 이유는 각 단어 간의 유사도를 계산했



# 텍스트 유사도

예시)

“시리야, 김치찌개의 레시피가 뭐야?”

“시리야, 김치찌개 어떻게 만들어?”

=> 김치찌개의 요리 방법

결과)

“레시피가 뭐야”

<-> “어떻게 만들어”

=> 엄연히 다른 문장이지만 같은 의도의 질문

=> 같은 내용의 답변

원-핫-인코딩

[ 0 or 1 ]

과학  $[ \circ \circ \circ \circ \circ \circ \circ \circ \circ \circ 1 \circ \circ \circ \circ ]^T$   
공학  $[ \circ \circ \circ \circ \circ \circ \circ 1 \circ \circ \circ \circ \circ \circ ] = \circ$

희소 표현  
희소 벡터  
(Sparse vector)

1번째 인덱스:  $[0, 1, 0, 0, 0, \dots, 0, 0]$

2번째 인덱스:  $[0, 0, 1, 0, 0, \dots, 0, 0]$

3번째 인덱스:  $[0, 0, 0, 1, 0, \dots, 0, 0]$

...

마지막 인덱스:  $[0, 0, 0, 0, 0, \dots, 0, 1]$

인덱스(=단어 수)가 늘어날수록,  
각 단어를 표현하는 벡터의 차원 수가 늘어남.

단어의 인덱스만  
표현 가능

= 컴퓨터 혼절


과학  $[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]^T$   
공학  $[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]^T =$



임베딩

[ float ]

<https://velog.io/@cjkangme/%EC%9E%90%EC%97%B0%EC%96%B4-%EC%8A%A4%ED%84%B0%EB%94%94-09.-%EC%9B%8C%EB%93%9C-%EC%9E%84%EB%B2%A0%EB%94%A9>



임의 지정  
'어떤 단어 A': [0.1, 1.5, -2.4, ...]

분산 표현  
=> 분포 가설

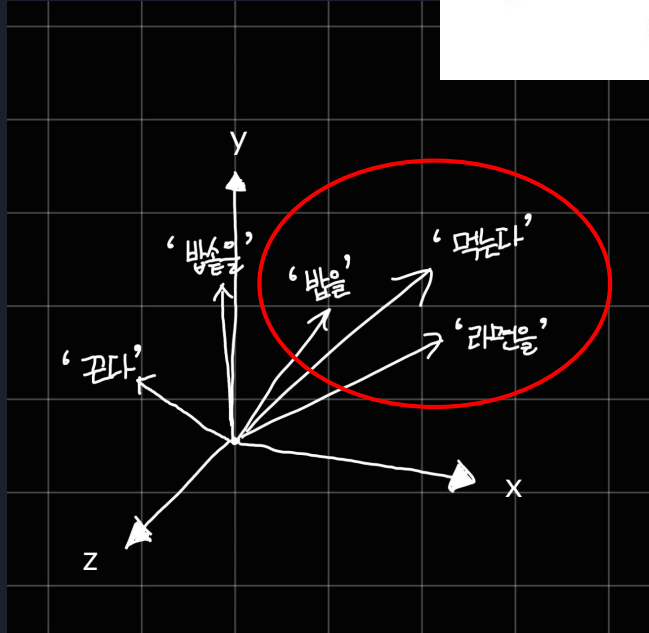
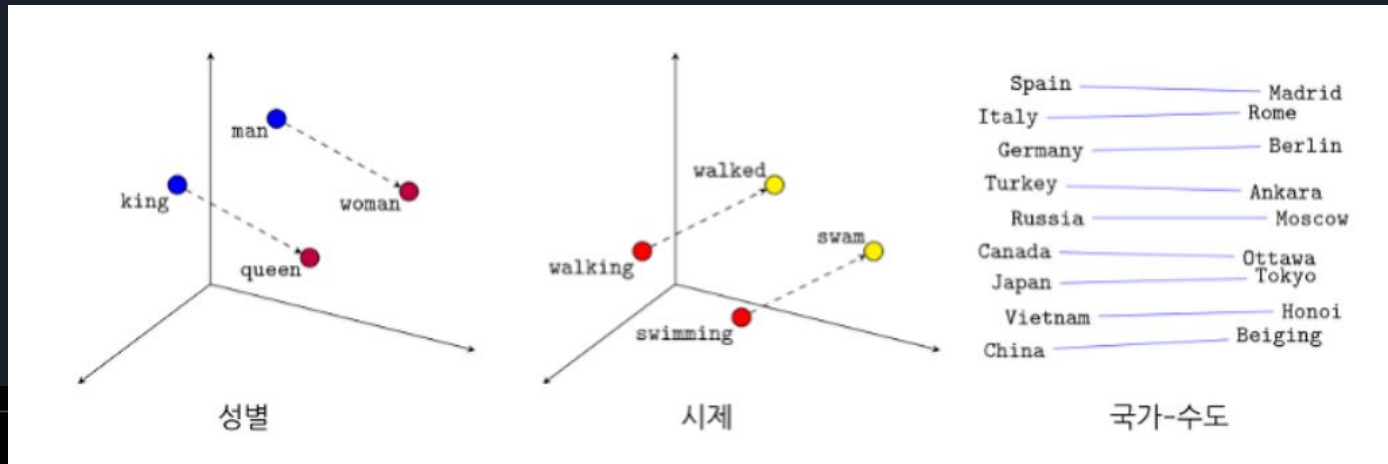
비슷한 문맥에서 등장하는  
단어들은 비슷한 의미를  
가짐.

밀집 표현  
↔ 희소 표현

벡터의 차원을 임의로 지정  
벡터 원소가 실수값  
학습 필요

word2vec

원-핫-인코딩된 단어가  
투사층을 통과하고  
오차 역전파를 적용한  
학습 결과 = 임베딩 벡터



O

“밥을 먹는다”

“라면을 먹는다”

“밥술에 밥을 짓는다”

“밥술을 끈다”

X

“끈다 먹는다”

“라면을 끈다”

“밥술을 라면”



중심 단어  
주변 단어

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

The fat cat sat on the mat

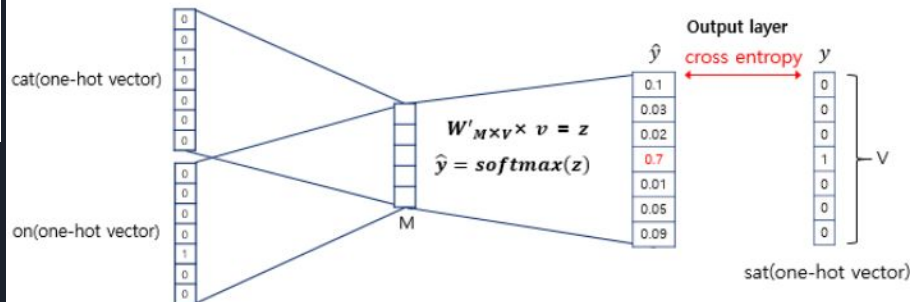
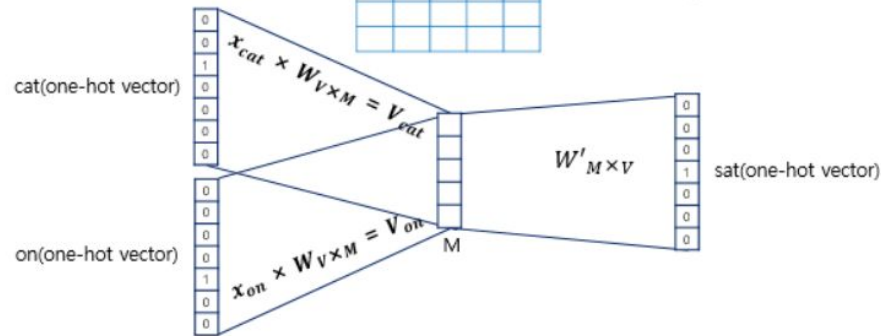
The fat cat sat on the mat

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

전에 이 가중치 행렬  $W$ 와  $W'$ 는 랜덤 값을 가지게 됩니다. CBOW는 주변 단어로 중심 단어를 더 정확히 맞추기 위해 계속해서 이  $W$ 와  $W'$ 를 학습해가는 구조입니다.

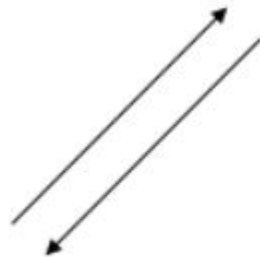
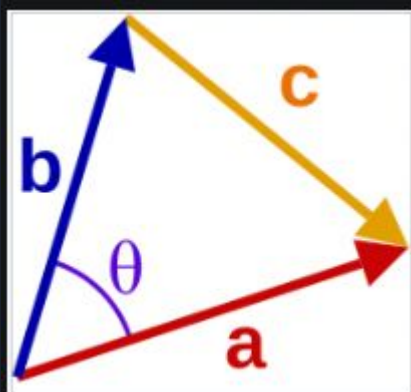
$$x_{cat} \times W_{V \times M} = V_{cat}$$

lookup table

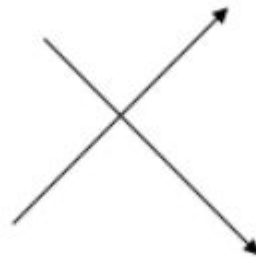


## 텍스트 유사도 측정을 위한 다양한 방법 : 코사인 유사도

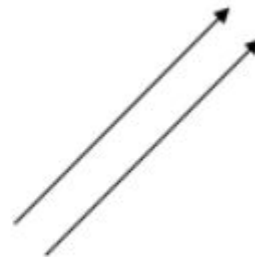
벡터의 내적을 이용하여 텍스트 유사도를 구함.




코사인 유사도 : -1



코사인 유사도 : 0



코사인 유사도 : 1



## 텍스트 유사도 측정을 위한 다양한 방법 : 유클리디언 유사도(L2 Distance)

두 벡터의 성분의 좌표, 즉 두 점 사이 거리를 구함.

측정값이 1을 넘을 수 있다는 단점.

+ 맨하탄 유사도(L1 Distance)

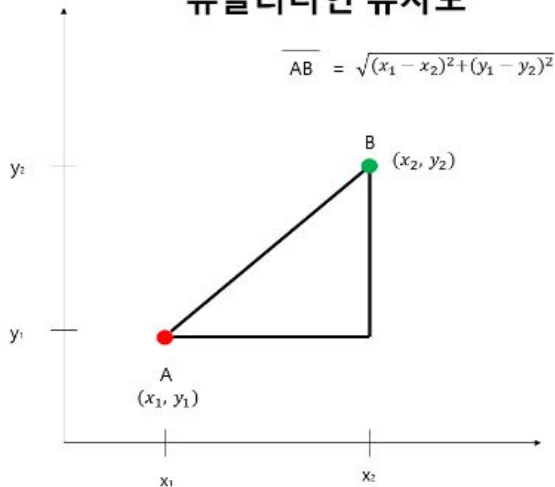
$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2}$$

유클리디언 유사도 공식

# 텍스트 유사도 측정을 위한 다양한 방법

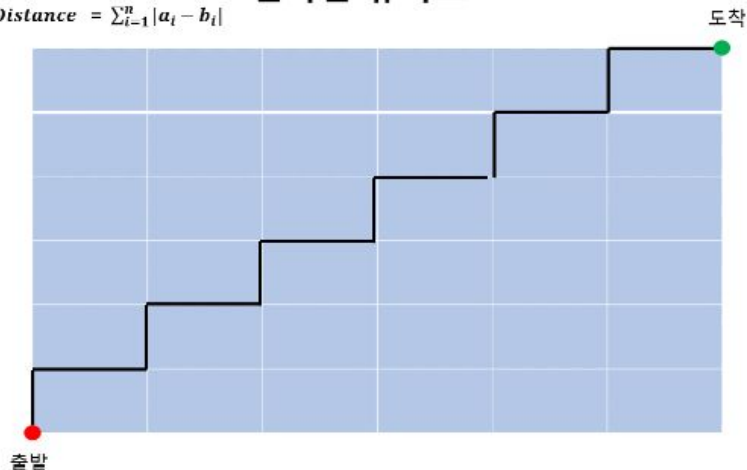
## : 유클리디언 유사도(L2 Distance)


### 유클리디언 유사도



### 맨하탄 유사도

$$MaDistance = \sum_{i=1}^n |a_i - b_i|$$





## 텍스트 유사도 측정을 위한 다양한 방법 : 자카드 유사도(Jaccard Similarity)

벡터를 사용하지 않고 겹치는 원소 비율을 활용함.

문장을 단어의 집합으로 만들어  
두 집합의 교집합, 합집합을 이용하여 유사도를 구함.

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

# 텍스트 유사도 측정을 위한 다양한 방법 : n-gram 유사도

n개의 연속적인 단어 배열로  
문장 간의 유사도를 따지는 방식

## 2-gram 유사도

A

12월 철수는 패딩을 구입하러  
상점으로 이동했다.

12월

철수

패딩

구입

상점

이동

B

12월 철수는 패딩을 환불하러  
백화점으로 이동했다.

12월

철수

패딩

환불

백화점

이동

	(12월, 철수)	(철수, 패딩)	(패딩, 구입)	(구입, 상점)	(상점, 이동)	(이동)	빈도
A	1	1	1	1	1	1	6
B	1	1	0	0	1	1	4

$$\text{Similarity} = \frac{tf(A, B)}{token(A)} = \frac{4}{6} = 0.66$$

# 텍스트 유사도 코드 동작

라이브러리 TF-IDF 그리고 Word2Vec

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

*# 객체 생성*

```
tfidf_vectorizer = TfidfVectorizer()
```

*# 문장 벡터화 진행*

```
tfidf_matrix = tfidf_vectorizer.fit_transform(sentence)
```

*# 각 단어*

```
text = tfidf_vectorizer.get_feature_names()
```

*# 각 단어의 벡터 값*

```
idf = tfidf_vectorizer.idf_
```

*# 라이브러리 불러 오기*

```
from tqdm import tqdm_notebook as tqdm
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
from gensim.models.word2vec import Word2Vec
```

```
from soynlp.tokenizer import RegexTokenizer
```

```
import pandas as pd
```

```
import numpy as np
```

```
import math
```



## 출처

<https://velog.io/@acdongpgm/%EC%B1%97%EB%B4%87-Tokenzie-%EC%99%80-Sparse-Vector%ED%9D%AC%EC%86%8C%EB%B2%A1%ED%84%B0> 토큰화와 희소 표현

<https://velog.io/@glad415/%EC%9E%84%EB%B2%A0%EB%94%A9Embedding%EC%9D%B4%EB%9E%80> 임베딩이란?

<https://thegap.tistory.com/90> 희소 표현 / 밀집 표현 / 임베딩

<https://soyoung-new-challenge.tistory.com/34> 텍스트 유사도 코딩

<https://wikidocs.net/24603> 위키독스\_코사인 유사도와 활용