

딥러닝 스터디

밑바닥 부터 시작하는 딥러닝2

김제우

딥러닝스터디

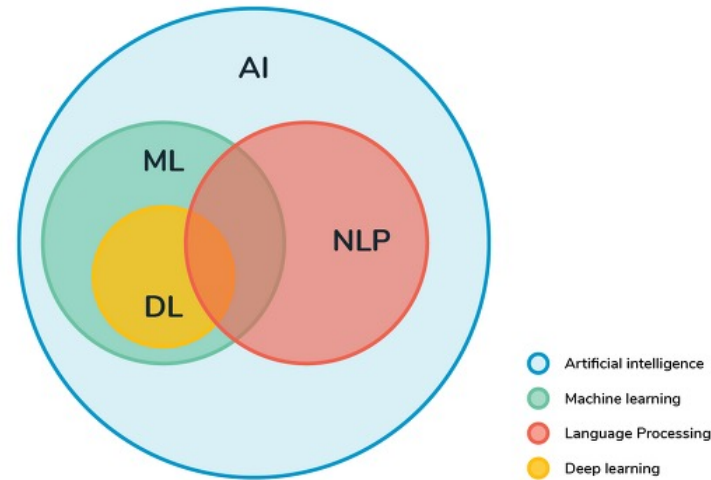
목차

2. 자연어와 단어의 분산 표현

2. 자연어와 단어의 분산 표현





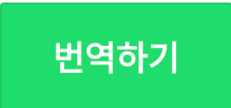





자연어 처리

자연어 (Natural Language) : 영어, 한국어 등 평소에 쓰는 말
컴퓨터가 우리의 말을 알아듣게(이해하게) 만드는 것



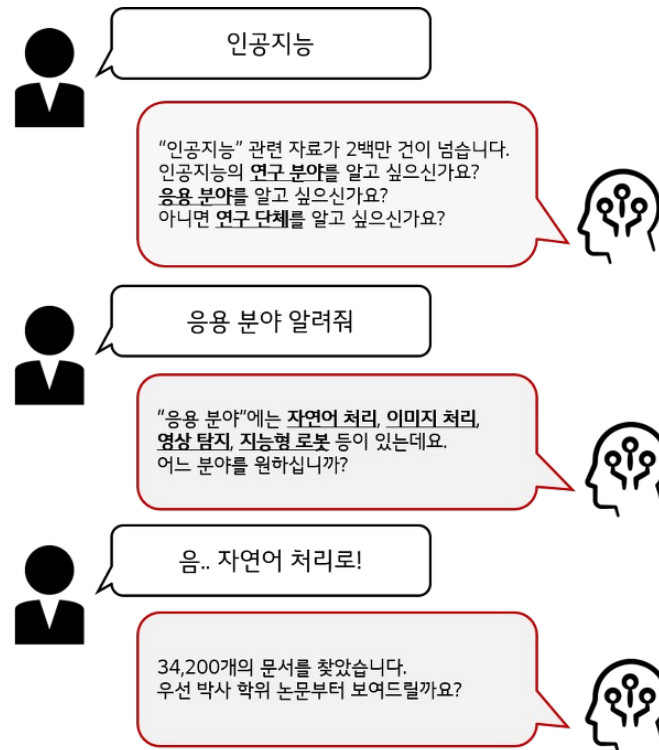
2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 기계 번역 (Machine translation)

한국어 감지 ▼	↔	영어 ▼
우리는 딥러닝 <u>스터디</u> 를 한다. ×		We do deep learning studies. 위 두 딥 러닝 스터디즈.
16 / 5000		번역 수정 번역 평가
    		    

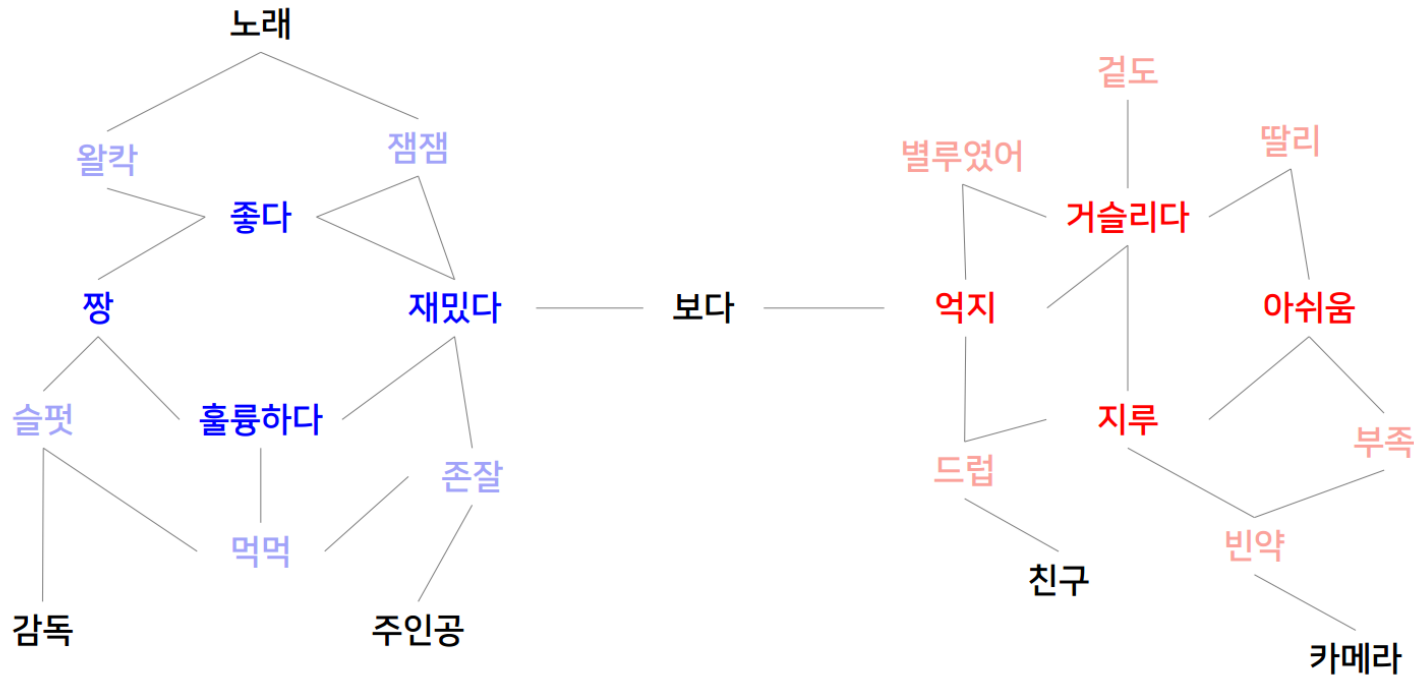
2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 질의 응답 (Question Answering)



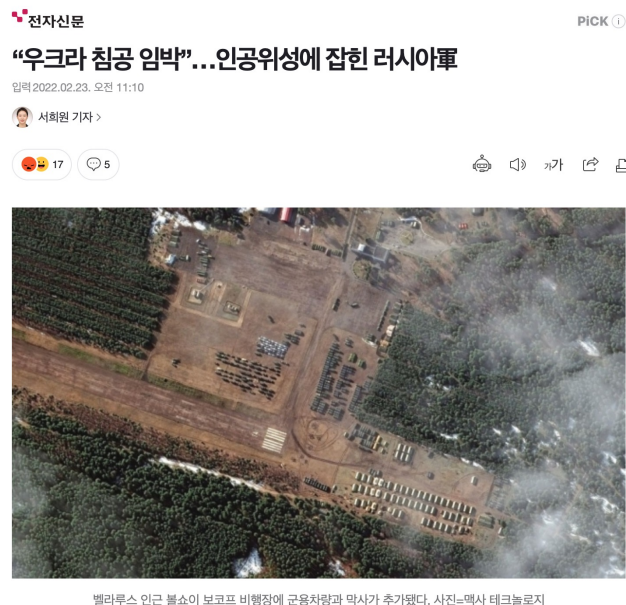
2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 감정 분석 (Sentimental Analysis)



2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 요약 (Summarization)



우크라이나 돈바스 지역에 '평화유지'를 명목으로 군대를 파병하기로 한 러시아가 벨라루스 등 접경지역에 병력을 보강하고 있는 것으로 확인됐다. 미국 위성업체 맥사(MAXAR) 테크놀로지의 위성사진을 인용해 로이터

본문 요약봇 ?

자동 추출 기술로 요약된 내용입니다. 요약 기술의 특성상 본문의 주요 내용이 제외될 수 있어, 전체 맥락을 이해하기 위해서는 기사 본문 전체보기를 권장합니다.

“우크라 침공 임박”...인공위성에 잡힌 러시아군

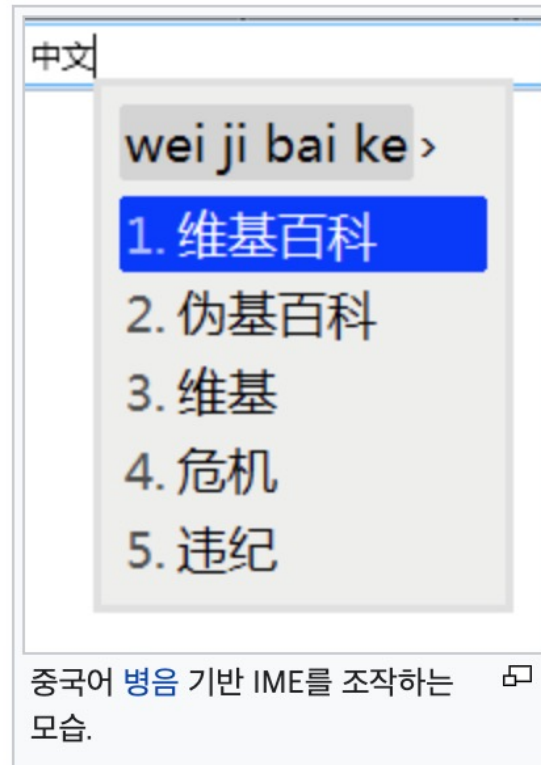
우크라이나 돈바스 지역에 '평화유지'를 명목으로 군대를 파병하기로 한 러시아가 벨라루스 등 접경지역에 병력을 보강하고 있는 것으로 확인됐다.

한편, 이날 조 바이든 미국 대통령은 러시아의 우크라이나 침공이 시작됐다고 판단하고, 러시아를 대상으로 한 첫 제재 조치를 단행했다.

앞서 블라디미르 푸틴 러시아 대통령이 우크라이나 동부 돈바스 지역의 분리주의 공화국들의 독립을 승인하고 이 곳에 병력 투입을 명령한 것이 '침공'이라는 설명이다.

2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 입력 전환기 (IME)



2.1 자연어 처리(NLP)란?

- 자연어 처리의 분야(task)
 - 기계 번역 (Machine Translation)
 - 질의응답 (Question Answering)
 - 감정분석 (Sentimental Analysis)
 - 요약 (Summarization)
 - 입력기 전환(IME)
 - 구문 상태 해석(Dialogue State Tracking)
 - 관계 추출(Relation Extraction)
 - 개체명 인식(Named Entity Recognition)

2.1.1 단어의 의미

- 말은 '문자'로 구성되며 말의 의미는 '단어'로 구성된다.
- 단어의 의미를 이해시키는 방법 3가지
 - 시소러스(유의어 사전)를 활용한 기법
 - 통계 기반 기법
 - 추론 기반 기법(word2vec) -> 다음 장

2.2 시소러스

- 동의어 사전(유의어 사전)

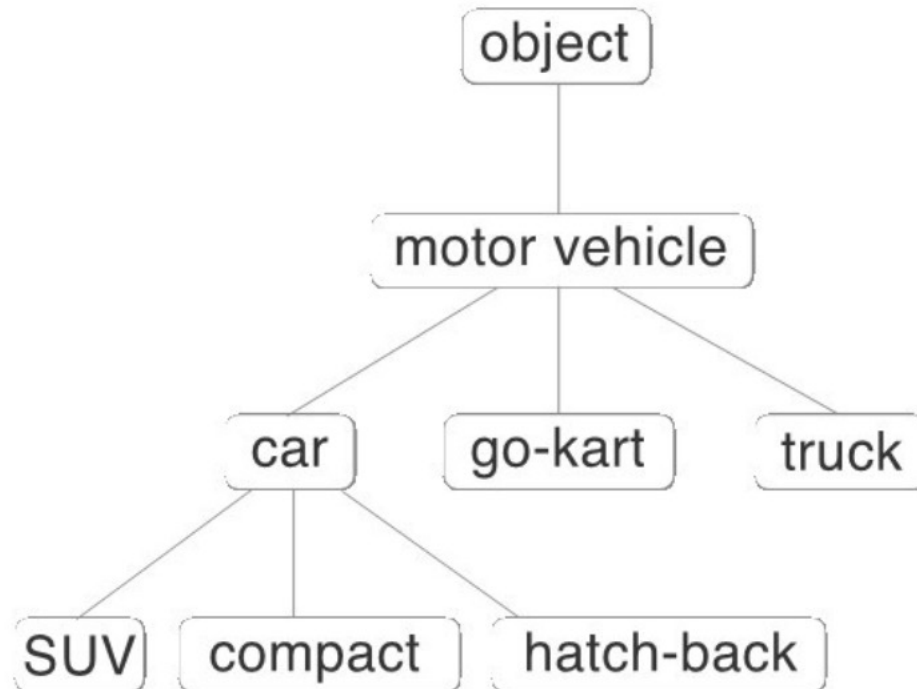
그림 2-1 동의어의 예: “car”, “auto”, “automobile” 등은 “자동차”를 뜻하는 동의어다.

car = auto automobile machine motorcar

2.2 시소러스

- 동의어 사전(유의어 사전)

그림 2-2 단어들을 의미의 상·하위 관계에 기초해 그래프로 표현한다(문헌 [14]를 참고하여 그림).



2.2 시소러스

- 동의어 사전(유의어 사전)

모든 단어의 네트워크를 이용하여 컴퓨터에게 단어사이의 관계를 가르침

2.2.1 WordNet

- 가장 유명한 시소러스
- 프린스턴 대학교 (1985~)
- NLTK 모듈
- 자세히 설명하지는 않음

2.2.2 시소러스의 문제점

- 동의어와 계층 구조 등의 관계가 정의돼 있음
- 사람이 수작업으로 레이블링 하는 방식의 문제점
 - 시대 변화에 대응하기 어렵다.
 - heavy에 심각하다는 뜻은 예전에는 쓰지 않았음
 - 사람을 쓰는 비용은 크다.
 - 현존하는 영단어는 1000만개 이상
 - WordNet에 등록된 단어는 약20만개
 - 단어의 미묘한 차이를 표현할 수 없다.
 - Vintage와 Retro의 의미는 같지만 용법이 다른 것 처럼 미묘한 차이를 표현할 수 없음
- 사람이 손수 단어를 연결 짓는 작업을 해결하기 위해 나온 통계 기반 기법!

2.3 통계 기반 기법

- 말뭉치(corpus)를 이용함.
 - 자연어 처리 연구를 염두에 두고 수집된 대량의 텍스트 데이터
 - 사람이 쓴 글이기 때문에 '지식'이 담겨 있다고 볼 수 있음.

분야	설명	수량
뉴스	뉴스 텍스트	80만 문장
정부 웹사이트/저널	정부/지자체 홈페이지, 간행물	10만 문장
법률	행정 규칙, 자치 법규	10만 문장
한국문화	한국 역사, 문화 콘텐츠	10만 문장
구어체	자연스러운 구어체 문장	40만 문장
대화체	상황/시나리오 기반 대화 세트	10만 문장
합계		160만 문장

2.3.1 파이썬으로 말뭉치 전처리하기

You say goodbye and I say hello.

you say goodbye and i say hello .

['you', 'say', 'goodbye', 'and', 'i', 'say', 'hello', '.']

2.3.1 파이썬으로 말뭉치 전처리하기

```
['you', 'say', 'goodbye', 'and', 'i', 'say', 'hello', '.']
```

```
id_to_word = {0: 'you', 1: 'say', 2: 'goodbye', 3: 'and', 4: 'i', 5: 'hello', 6: '.'}
```

```
word_to_id = {'you': 0, 'say': 1, 'goodbye': 2, 'and': 3, 'i': 4, 'hello': 5, '.': 6}
```

```
corpus = [0, 1, 2, 3, 4, 1, 5, 6]
```

2.3.1 파이썬으로 말뭉치 전처리하기

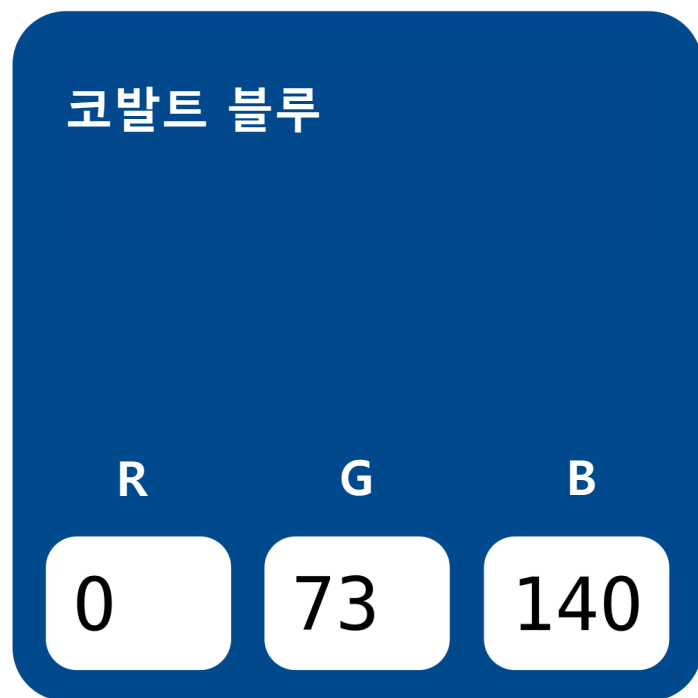
```
# 말뭉치 전처리 함수
def preprocess(text):
    text = text.lower() # 모든 문자 소문자로 변환
    text = text.replace('.', ' .') # 공백 만들기
    words = text.split(' ') # 공백을 기준으로 분할

    word_to_id = {}
    id_to_word = {}

    for word in words:
        if word not in word_to_id:
            new_id = len(word_to_id)
            word_to_id[word] = new_id
            id_to_word[new_id] = word
    corpus = np.array([word_to_id[w] for w in words]) # 단어 ID, 넘파이 배열로 변환

    return corpus, word_to_id, id_to_word
```

2.3.2 단어의 분산 표현



색의 벡터화

단어를 벡터로 표현해보려는 작업
분산 표현 distributional representation

2.3.3 분포 가설

- 단어의 의미는 주변 단어에 의해 형성된다.
- 단어 자체에는 의미가 없고 '맥락context'이 의미를 형성한다.

윈도우 크기가 2인 맥락

you say **goodbye** and i say hello



맥락을 통한 의미 형성

2.3.3 분포 가설

- 단어의 의미는 주변 단어에 의해 형성된다.
- 단어 자체에는 의미가 없고 '맥락context'이 의미를 형성한다.

상황에 따라서는 왼쪽만 볼 수도, 오른쪽만 볼 수 있지만 이 책에서는 좌우 동수인 맥락만 사용

you say **goodbye** and i say hello



맥락을 통한 의미 형성

2.3.4 동시발생 행렬

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0

[0, 1, 0, 0, 0, 0, 0]

you say goodbye and i say hello

2.3.4 동시발생 행렬

	you	say	goodbye	and	i	hello	.
say	1	0	1	0	1	1	0

[1, 0, 1, 0, 1, 1, 0]

you say goodbye and i say hello .

2.3.4 동시발생 행렬

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

모든 단어에 대해 수행 -> 동시발생 행렬

2.3.5 벡터 간 유사도

- 벡터 사이의 유사도를 측정하는 방법
 - 벡터의 내적
 - 유클리드 거리
- 단어 벡터의 유사도를 구할 때 많이 사용하는 코사인 유사도

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

2.3.5 벡터 간 유사도

- 코사인 유사도
- 두 벡터의 방향이 완전히 동일하면 1, 90도면 0, 180도면 -1
- 이 값이 얼마나 1에 가까운가

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

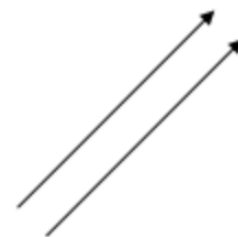
L2 노름 : 원점에서 A점 B점까지의 직선거리



코사인 유사도 : -1



코사인 유사도 : 0



코사인 유사도 : 1

2.3.5 벡터 간 유사도

- 코사인 유사도

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

```
# 코사인 유사도
def cos_similarity(x,y,eps=1e-8):
    nx = x/np.sqrt(np.sum(x**2)+eps) # x의 정규화
    ny = y/np.sqrt(np.sum(y**2)+eps) # y의 정규화
    return np.dot(nx,ny)
```

2.3.5 벡터 간 유사도

- 코사인 유사도

```
# 단어 벡터 유사도 구하기
# you와 i의 유사도 구하기

text2 = "You say goodbye and I say hello."
corpus, word_to_id, id_to_word = preprocess(text2)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)

c0 = C[word_to_id['you']] # 'you'의 단어 벡터
c1 = C[word_to_id['i']] # 'i'의 단어 벡터

print(cos_similarity(c0, c1))
>> 0.7071067758832467 # 유사성이 높다
```

전체 단어의 개수

동시 발생 행렬

동시 발생 행렬 중 'you'의 벡터

동시 발생 행렬 중 'i'의 벡터

2.3.6 유사 단어의 랭킹 표시

1. 검색어의 단어 벡터를 꺼낸다.
2. 검색어의 단어 벡터와 다른 모든 단어 벡터와의 코사인 유사도를 구한다.
3. 계산한 코사인 유사도 결과를 기준으로 값이 높은 순서대로 출력한다.

검색어 →

	you	say	goodbye	and	i	hello	.
you	0	1	0	0	0	0	0
say	1	0	1	0	1	1	0
goodbye	0	1	0	1	0	0	0
and	0	0	1	0	1	0	0
i	0	1	0	1	0	0	0
hello	0	1	0	0	0	0	1
.	0	0	0	0	0	1	0

2.3.6 유사 단어의 랭킹 표시

```
def most_similar(query, word_to_id, id_to_word, word_matrix, top=5):  
    # 1. 검색어를 꺼낸다.  
    if query not in word_to_id:  
        print('%s(을)를 찾을 수 없습니다.' % query) → 검색어(query) 꺼냄  
        return  
  
    print('[검색어]:'+query +'\n')  
    query_id = word_to_id[query] # 검색 단어의 id 지정  
    query_vec = word_matrix[query_id] # 유사도 행렬의 id의 행  
  
    # 2. 코사인 유사도 계산  
    vocab_size = len(id_to_word)  
    similarity = np.zeros(vocab_size) # 0벡터 초기화 → 다른 벡터들과 코사인 유사도 계산  
    for i in range(vocab_size):  
        similarity[i] = cos_similarity(word_matrix[i], query_vec) # 각 단어와의 코사인 유사도 계산  
  
    # 3. 코사인 유사도를 기준으로 내림차순으로 출력  
    count = 0  
    for i in (-1*similarity).argsort(): # 큰 수로 나열 → 내림차순 정렬  
        if id_to_word[i] == query:  
            continue  
        print('%s: %s' % (id_to_word[i], similarity[i]))  
  
        count += 1  
        if count >= top:  
            return
```


2.3.6 유사 단어의 랭킹 표시

```
# 검색어 you와 유사도가 높은 순서
```

```
text2 = "You say goodbye and I say hello." → 문장
corpus, word_to_id, id_to_word = preprocess(text2)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)
```

```
most_similar('you', word_to_id, id_to_word, C, top=5) → 검색어를 활용한 유사도 연산
```

```
#결과
```

```
[검색어:]you
```

```
goodbye: 0.7071067758832467
i: 0.7071067758832467
hello: 0.7071067758832467
say: 0.0
and: 0.0
```

→ 'you'와의 유사단어 랭킹 상위5개 결과

2.4 통계 기반 기법 개선하기

2.4.1 상호정보량

- '발생 횟수'라는 것은 좋은 특징은 아님
- car 와 drive는 깊은 연관성이 있지만 여러 문장에서 빈도수로만 본다면 the 랑 더 많이 붙어있을 가능성이 높다는 문제점이 있음
- the와 같은 고빈도 단어들이랑 연관성이 높다는 문제점을 보완하고자 함.
- 점별 상호정보량(PMI - Pointwise Mutual Information)

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

2.4.1 상호정보량

- 점별 상호정보량(PMI - Pointwise Mutual Information)

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)}$$

$P(x)$ 는 x 가 일어날 확률

$P(y)$ 는 y 가 일어날 확률

$P(x, y)$ 는 x 와 y 가 동시에 일어날 확률

2.4.1 상호정보량

- 점별 상호정보량(PMI - Pointwise Mutual Information)

$$\text{PMI}(x, y) = \log_2 \frac{P(x, y)}{P(x)P(y)} = \log_2 \frac{\frac{C(x, y)}{N}}{\frac{C(x)}{N} \frac{C(y)}{N}} = \log_2 \frac{C(x, y) \cdot N}{C(x)C(y)}$$

$C(x, y)$ 는 단어 x 와 y 가 동시발생하는 횟수
 $C(x)$ 와 $C(y)$ 는 각각 단어 x 와 y 의 등장 횟수
 N 은 말뭉치에 포함된 단어의 수

단어가 동시에 보이는 횟수를 단어가 단독으로 등장하는 횟수로 나눔
PMI의 의미 : 같이는 많이 나오고 단독으로는 잘 안나오는 단어 연관 지수

2.4.1 상호정보량

- 점별 상호정보량(PMI - Pointwise Mutual Information)

the는 1000번 car는 20번 drive는 10번 등장했을때

$$\text{PMI}(\text{"the"}, \text{"car"}) = \log_2 \frac{10 \cdot 10000}{1000 \cdot 20} \approx 2.32$$

the와 car의 동시 발생횟수가 10일때 PMI

$$\text{PMI}(\text{"car"}, \text{"drive"}) = \log_2 \frac{5 \cdot 10000}{20 \cdot 10} \approx 7.97$$

the와 drive의 동시 발생횟수가 5일때 PMI

2.4.1 상호정보량

- 양의 상호정보량 (PPMI - Positive PMI)
- 실제 구현에서는 PPMI를 사용

$$\text{PPMI}(x, y) = \max(0, \text{PMI}(x, y))$$

$\log 0$ 은 마이너스 무한대가 되기 때문에 PMI가 음수 일 때는 0으로 취급

2.4.1 상호정보량

- 양의 상호정보량 (PPMI - Positive PMI)

```
text2 = "You say goodbye and I say hello."
corpus, word_to_id, id_to_word = preprocess(text2)
vocab_size = len(word_to_id)
C = create_co_matrix(corpus, vocab_size)
W = ppmi(C) # return M

np.set_printoptions(precision=3) # 유효 자릿수 3자리 표시
print("동시발생 행렬")
print(C)
print('-'*50)
print("PPMI")
print(W)
```

동시발생 행렬

```
[[0 1 0 0 0 0 0]
 [0 0 1 0 1 1 0]
 [0 1 0 1 0 0 0]
 [0 0 1 0 1 0 0]
 [0 1 0 1 0 0 0]
 [0 1 0 0 0 0 1]
 [0 0 0 0 1 0 0]]
```

PPMI

```
[[0.  inf 0.  0.  0.  0.  0.]
 [0.  0.  0.7 0.  0.7 0.7 0.]
 [0.  0.7 0.  1.7 0.  0.  0.]
 [0.  0.  1.7 0.  1.7 0.  0.]
 [0.  0.7 0.  1.7 0.  0.  0.]
 [0.  0.7 0.  0.  0.  0.  2.7]
 [0.  0.  0.  0.  0.  2.7 0.]]
```

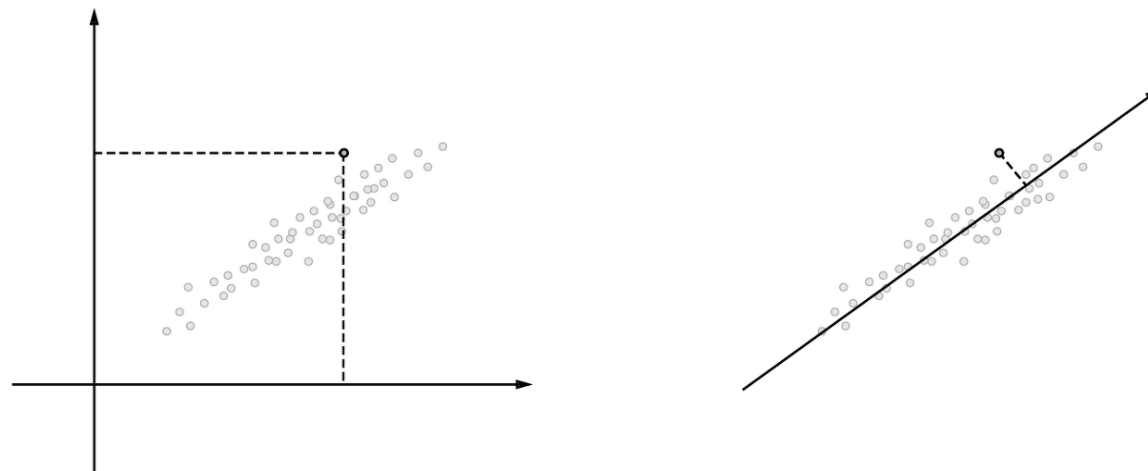
2.4.1 상호정보량

- PPMI의 문제점
- 말뭉치의 어휘 수가 증가함에 따라 각 단어 벡터의 차원 수가 증가함.
- 말뭉치가 10만 개라면 그 벡터의 차원 수도 똑같이 10만이 된다.
- 또한 행렬의 대부분의 원소가 0이다.
- 이런 벡터는 노이즈에 약하고 견고하지 못하다는 약점이 있음

2.4.2 차원 감소

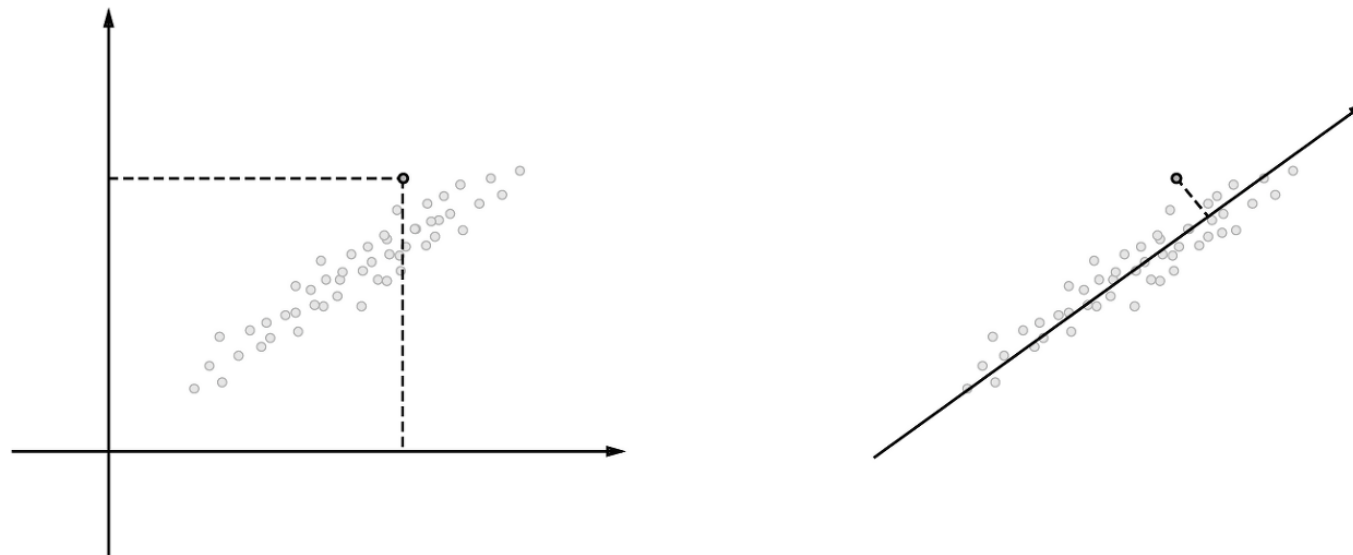
- 차원 감소 dimensionality reduction은 문자 그대로 벡터의 차원을 줄이는 방법을 말합니다.
- 중요한 정보는 최대한 유지하면서 줄이는 게 핵심
- 데이터의 분포를 고려해 중요한 '축'을 찾는 일을 수행

그림 2-8 그림으로 이해하는 차원 감소: 2차원 데이터를 1차원으로 표현하기 위해 중요한 축(데이터를 넓게 분포시키는 축)을 찾는다.



2.4.2 차원 감소

그림 2-8 그림으로 이해하는 차원 감소: 2차원 데이터를 1차원으로 표현하기 위해 중요한 축(데이터를 넓게 분포시키는 축)을 찾는다.



각 데이터점의 값은 새로운 축으로 사영된 값으로 변한다.

2.4.2 차원 감소

- 희소 벡터(Sparse vector)
 - 원소의 대부분이 0인 행렬
[0,0,1,0,1,0]
- 밀집 벡터(Dense vector)
 - 원소 대부분이 0이 아닌 행렬
[1.5,2,4,3.2,6,2]

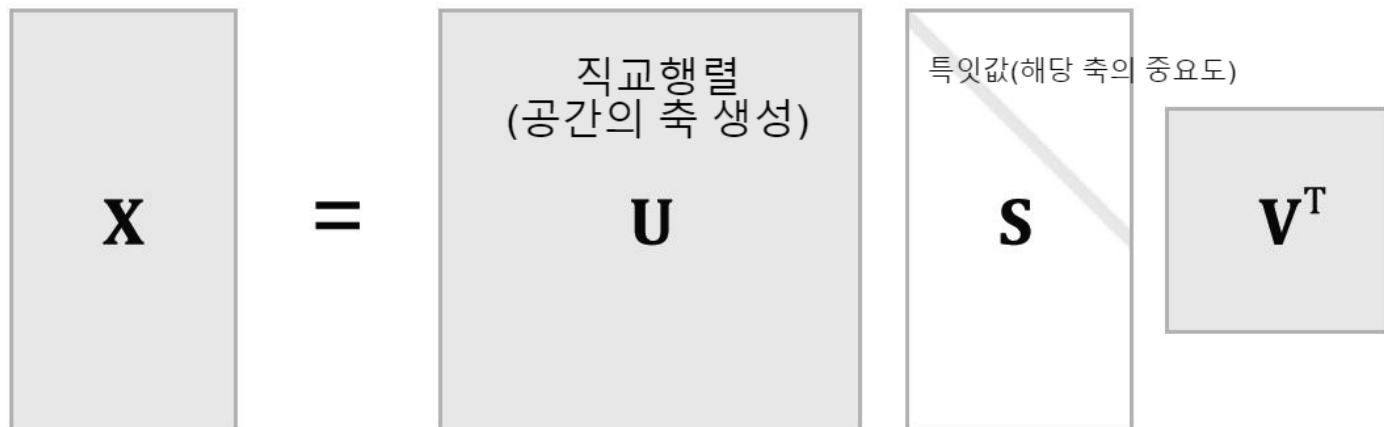
2.4.2 차원 감소

- 특잇값 분해 (SVD)

$$X = USV^T$$

임의의 행렬 X 를 USV 세 개로 분해함
 U 와 V 는 직교 행렬(orthogonal matrix)
 S 는 대각 행렬(diagonal matrix)

그림 2-9 SVD에 의한 행렬의 변환(행렬의 '흰 부분'은 원소가 0임을 뜻함)

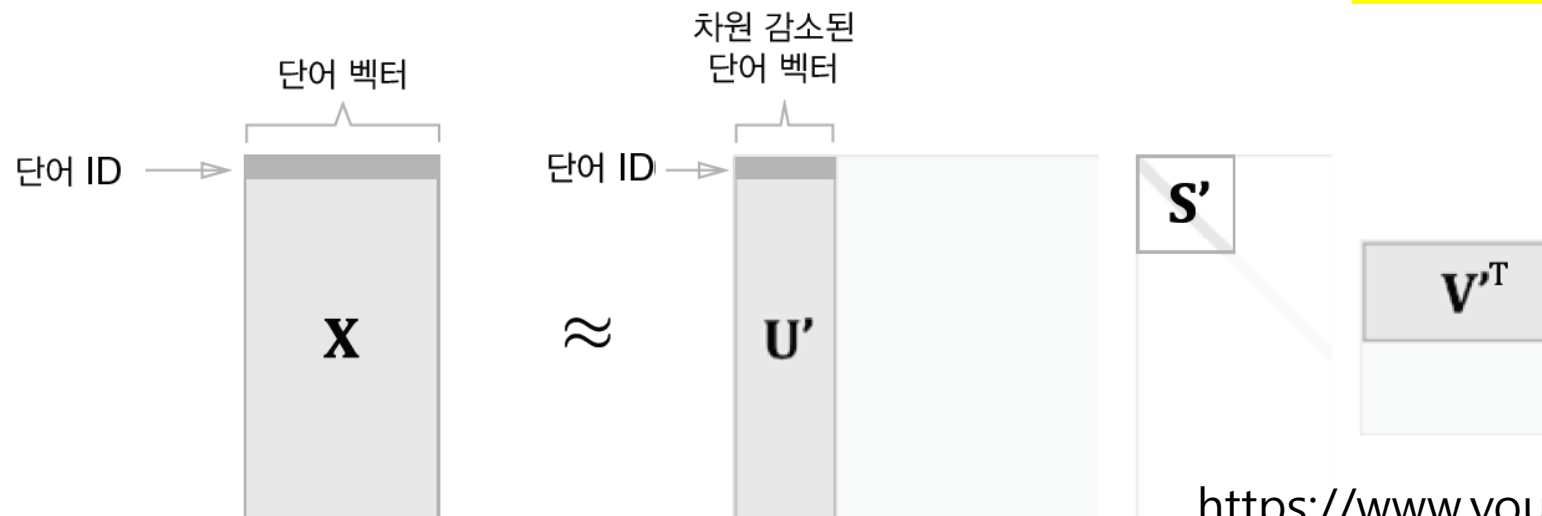


2.4.2 차원 감소

- 특잇값분해 (SVD)

$$X = USV^T$$

그림 2-10 SVD에 의한 차원 감소



행렬 S 에서 특잇값이 작다면 중요도가 낮다는 뜻이므로
행렬 U 에서 여분의 열 벡터를 깎아내어
원래의 행렬을 근사할 수 있음

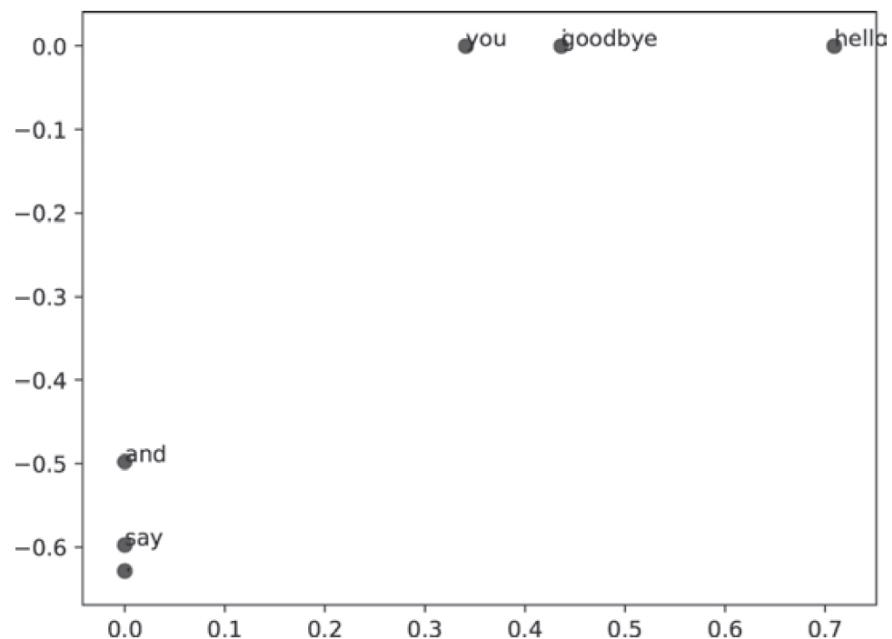
2.4.3 SVD에 의한 차원 감소

`U, S, V = np.linalg.svd(W)`

희소벡터인 W 를 밀집벡터 U 로 바꿈.
 $U[0,:2] \rightarrow$ 2차원 축소

2.4.3 SVD에 의한 차원 감소

그림 2-11 동시발생 행렬에 SVD를 적용한 후, 각 단어를 2차원 벡터로 변환해 그린 그래프("i"와 "goodbye"가 겹쳐 있음)



말뭉치가 너무 작아서 결과를 그대로 믿기 어렵다.
-> 대규모 말뭉치를 이용하여 평가

2.4.4 PTB 데이터셋

- 펜 트리뱅크 (PTB) 말뭉치
- 주어진 기법의 품질을 측정하는 벤치마크
- 한 문장이 하나의 줄로 저장되어 있음.
- 하나의 큰 시계열 데이터로 취급함.

```
1 consumers may want to move their telephones a little closer to the tv set
2 <unk> <unk> watching abc 's monday night football can now vote during <unk> for the greatest play in N years from
   among four or five <unk> <unk>
3 two weeks ago viewers of several nbc <unk> consumer segments started calling a N number for advice on various
   <unk> issues
4 and the new syndicated reality show hard copy records viewers ' opinions for possible airing on the next day 's show
5 interactive telephone technology has taken a new leap in <unk> and television programmers are racing to exploit the
   possibilities
6 eventually viewers may grow <unk> with the technology and <unk> the cost
```


2.5 정리

- WordNet 등의 시소러스를 이용하면 유의어를 얻거나 단어 사이의 유사도를 측정하는 등 유용한 작업을 할 수 있다.
- 시소러스 기반 기법은 시소러스를 작성하는 데 엄청난 인적 자원이 든다거나 새로운 단어에 대응하기 어렵다는 문제가 있다.
- 현재는 말뭉치를 이용해 단어를 벡터화 하는 방식이 주로 쓰인다.
- 최근의 단어 벡터화 기법들은 대부분 '단어의 의미는 주변 단어에 의해 형성된다'는 분포 가설에 기초한다.
- 통계 기반 기법은 말뭉치 안의 각 단어에 대해서 그 단어의 주변 단어의 빈도를 집계한다(동시발생 행렬)

2.5 정리

- 동시발생 행렬을 PPMI 행렬로 변환하고 다시 차원을 감소시킴으로써, 거대한 '희소벡터'를 작은 '밀집벡터'로 변환할 수 있다.
- 단어의 벡터 공간에서는 의미가 가까운 단어는 그 거리도 가까울 것으로 기대된다.