

# Tensorflow를 활용한 딥러닝 자연어 처리 입문. 1강

# 자연어 처리를 배우는 이유

- 점점 시장이 커질 수 밖에 없는 분야. 커리어의 우위를 가질 수 있다.
- 회사의 도메인과 상관없이 거의 모든 분야에서 활용이 가능.
- 배우면 배울 수록 재밌음.
- 성능 좋은 모델들이 오픈소스로 계속해서 외부에 공개되고 있는 중!

가성비가 좋은 공부!



# 선수 지식

- 약간의 Python 실력 (for문, if문, 리스트 등을 다뤄본 경험)
- 선형 회귀, 로지스틱 회귀, 경사 하강법에 대해서 들어봤다.
- 그렇지 않다면 2강이 시작 되기 전까지 공부를 해오시는 것을 권장.

동영상 강의 링크 : <https://youtu.be/BS6O0zOGX4E>

선형 회귀 : <https://wikidocs.net/21670>

로지스틱 회귀 : <https://wikidocs.net/22881>

# 선수 지식

- 약간의 Python 실력 (for문, if문, 리스트 등을 다뤄본 경험)
- 선형 회귀, 로지스틱 회귀, 경사 하강법에 대해서 들어봤다.
- 그렇지 않다면 2강이 시작 되기 전까지 공부를 해오시는 것을 권장.
- **케라스와 텐서플로우가 처음이시라면 매주 복습 필수.**

동영상 강의 링크 : <https://youtu.be/BS6O0zOGX4E>

선형 회귀 : <https://wikidocs.net/21670>

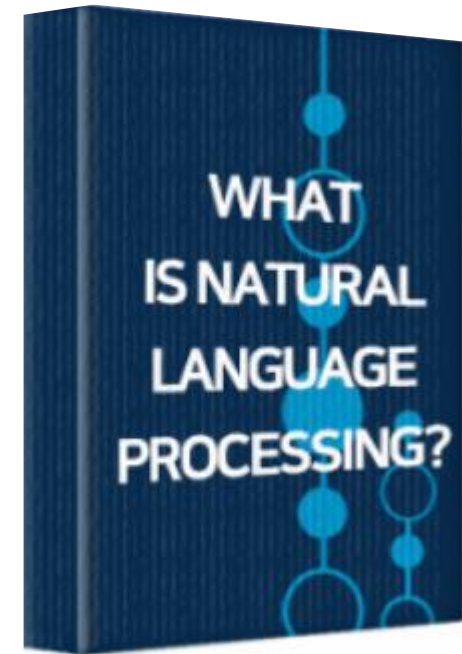
로지스틱 회귀 : <https://wikidocs.net/22881>

# 자연어 처리를 배울 때 참고하면 좋은 자료들

- 1. Ratsgo 블로그  
: <https://ratsgo.github.io/blog/>
- 2. 한국어 임베딩 (출판 교재)  
[https://book.naver.com/bookdb/book\\_detail.nhn?bid=15431390](https://book.naver.com/bookdb/book_detail.nhn?bid=15431390)
- 3. lovit 블로그  
: <https://lovit.github.io/>
- 4. Jay Alammар 블로그 (영어)  
: <http://jalammar.github.io/>
- 5. 고려대학교 DSBA 연구실  
: 유튜브 채널 '**KoreaUniv DSBA**'  
[youtube.com/channel/UCPq01cgCcEwhXI7BvcwlQyg](https://youtube.com/channel/UCPq01cgCcEwhXI7BvcwlQyg)
- 6. 스탠포드 대학교 강의 CS224N  
<http://web.stanford.edu/class/cs224n/>
- 7. 자연어 처리 오픈 카톡방  
<https://open.kakao.com/o/gkBgzwC>

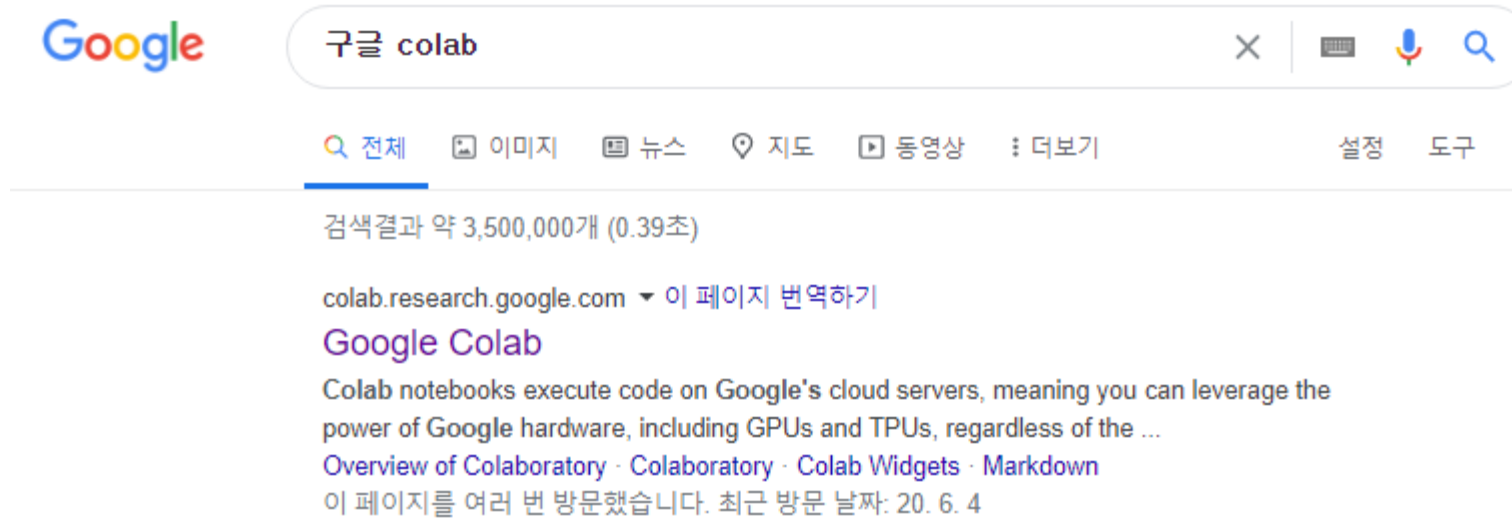
# 앞으로 배우게 될 내용

- **Text preprocessing for NLP & Language Model**
- Basic Tensorflow & Vectorization
- Word Embedding (Word2Vec, FastText, GloVe)
- Text Classification (using RNN & CNN)
- Chatbot with Deep Learning
- Sequence to Sequence
- Attention Mechanism
- Transformer & BERT



참고 자료 : <https://wikidocs.net/book/2155>

# 실습 환경 : Colab



- 구글 검색창에 구글 colab이라고 검색해봅시다.
- 설치가 필요하지 않으며 GPU를 사용 가능한 무료 개발 환경
- colab 사용 방법 : [https://tykimos.github.io/2019/01/22/colab\\_getting\\_started/](https://tykimos.github.io/2019/01/22/colab_getting_started/)

# Text preprocessing for NLP



# Text preprocessing for NLP

- Tokenization (Word & Sentence)
- Cleaning & Normalization
- Stopwords
- Build your vocabulary
- Integer Encoding
- Padding
- Vectorization

# Text preprocessing for NLP

- Tokenization (Word & Sentence)
- Cleaning & Normalization
- Stopwords
- Build your vocabulary
- Integer Encoding
- Padding
- Vectorization

수업이 끝나고 이 키워드로부터 내용이 연상되는지 떠올려보세요.

# Tokenization (eng)

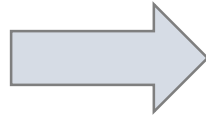
# Tokenization (eng)

- 기계에게 어느 구간까지가 문장이고, 단어인지를 알려주어야 합니다.
- 문장 토큰화, 단어 토큰화, subword 토큰화 등. 다양한 단위의 토큰화가 존재.

## 원문

His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to make sure no one was near.

sentence  
tokenization



# Tokenization (eng)

- 기계에게 어느 구간까지가 문장이고, 단어인지를 알려주어야 합니다.
- 문장 토큰화, 단어 토큰화, subword 토큰화 등. 다양한 단위의 토큰화가 존재.

## 원문

His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to mae sure no one was near.

sentence  
tokenization



['His barber kept his word.',  
'But keeping such a huge secret to himself was driving him crazy.',  
'Finally, the barber went up a mountain and almost to the edge of a cliff.',  
'He dug a hole in the midst of some reeds.',  
'He looked about, to mae sure no one was near.']

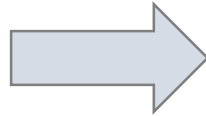
# Tokenization (eng)

- 기계에게 어느 구간까지가 문장이고, 단어인지를 알려주어야 합니다.
- 문장 토큰화, 단어 토큰화, subword 토큰화 등. 다양한 단위의 토큰화가 존재.

## 원문

His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to make sure no one was near.

word  
tokenization



# Tokenization (eng)

- 기계에게 어느 구간까지가 문장이고, 단어인지를 알려주어야 합니다.
- 문장 토큰화, 단어 토큰화, subword 토큰화 등. 다양한 단위의 토큰화가 존재.

## 원문

His barber kept his word. But keeping such a huge secret to himself was driving him crazy. Finally, the barber went up a mountain and almost to the edge of a cliff. He dug a hole in the midst of some reeds. He looked about, to mae sure no one was near.

word  
tokenization



['His', 'barber', 'kept', 'his', 'word', '.', 'But', 'keeping', 'such', 'a', 'huge', 'secret', 'to', 'himself', 'was', 'driving', 'him', 'crazy', '.', 'Finally', ',', 'the', 'barber', 'went', 'up', 'a', 'mountain', 'and', 'almost', 'to', 'the', 'edge', 'of', 'a', 'cliff', '.', 'He', 'dug', 'a', 'hole', 'in', 'the', 'midst', 'of', 'some', 'reeds', '.', 'He', 'looked', 'about', ',', 'to', 'mae', 'sure', 'no', 'one', 'was', 'near', '.']

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?



# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

다음의 세 문장으로부터 단어를 구분하는 작업을 해봅시다.

We're Avengers!!


We are Avengers!!

We are Avengers

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!! 

We are Avengers!! 

We are Avengers 

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!! →

We are Avengers!! →

We are Avengers →

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	→	['We're', 'Avengers!!']
We are Avengers!!	→	['We', 'are', 'Avengers!!']
We are Avengers	→	['We', 'are', 'Avengers']

*세 문장은 다른 결과*

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	—————→	['We're', 'Avengers!!]
We are Avengers!!	—————→	['We', 'are', 'Avengers!!]
We are Avengers	—————→	['We', 'are', 'Avengers']

특수 문자는 제거하고 해본다면?

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	→	['We're', 'Avengers!!]
We are Avengers!!	→	['We', 'are', 'Avengers!!]
We are Avengers	→	['We', 'are', 'Avengers']

특수 문자는 제거하고 해본다면?

\$45.55

Ph.D

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	—————→	['We're', 'Avengers!!]
We are Avengers!!	—————→	['We', 'are', 'Avengers!!]
We are Avengers	—————→	['We', 'are', 'Avengers']

특수 문자는 제거하고 해본다면?

\$45.55	제거 —————→
Ph.D	—————→

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	—————→	['We're', 'Avengers!!]
We are Avengers!!	—————→	['We', 'are', 'Avengers!!]
We are Avengers	—————→	['We', 'are', 'Avengers']

특수 문자는 제거하고 해본다면?

\$45.55	————— <sup>제거</sup> —————→	45 55
Ph.D	—————→	Ph D



# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?  
만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!!	—————→	['We're', 'Avengers!!]
We are Avengers!!	—————→	['We', 'are', 'Avengers!!]
We are Avengers	—————→	['We', 'are', 'Avengers']

특수 문자는 제거하고 해본다면?

\$45.55	————— <sup>제거</sup> —————→	45 55	—————→	['45', '55']
Ph.D	—————→	Ph D	—————→	['Ph', 'D']

*본래 의미 상실*

# Word Tokenization (eng)

문장 내에서 단어를 어떻게 구분할까?

만약 띄어쓰기로 단어를 구분한다면?

We're Avengers!! → ['We're', 'Avengers!!']

We are Avengers!! → ['We', 'are', 'Avengers!!']

We are Avengers!!

특수 :

**결국 상황에 맞는 전처리와  
토큰라이저의 선택이 성능을 좌우한다.**

\$45.55  $\xrightarrow{\text{세거}}$  45 55  $\rightarrow$  ['45', '55']

Ph.D  $\rightarrow$  Ph D  $\rightarrow$  ['Ph', 'D']

**본래 의미 상실**

# 단어 : TreebankWordTokenizer (eng)

영어권 언어의 단어 토크나이저 중 하나인 TreebankWordTokenizer.  
표준 토큰화 규칙인 Penn Treebank Tokenization를 따른다.

**규칙 1. 하이픈으로 구성된 단어는 하나로 유지한다.**

**규칙 2. doesn't와 같이 아포스트로피로 '접어'가 함께하는 단어는 분리해준다.**

Starting a home-based restaurant may be an ideal. it doesn't have a food chain or restaurant of their own.



['Starting', 'a', 'home-based', 'restaurant', 'may', 'be', 'an', 'ideal.', 'it', 'does', "n't", 'have', 'a', 'food', 'chain', 'or', 'restaurant', 'of', 'their', 'own', '.']

# 단어 : TreebankWordTokenizer (eng)

영어권 언어의 단어 토크나이저 중 하나인 TreebankWordTokenizer.  
표준 토큰화 규칙인 Penn Treebank Tokenization를 따른다.

규칙 1. 하이픈으로 구성된 단어는 하나로 유지한다.

규칙 2. doesn't와 같이 아포스트로피로 '접어'가 함께하는 단어는 분리해준다.

Starting a **home-based** restaurant may be an ideal. it **doesn't** have a food chain or restaurant of their own.



['Starting', 'a', '**home-based**', 'restaurant', 'may', 'be', 'an', 'ideal.', 'it', '**does**', '**"n't"**', 'have', 'a', 'food', 'chain', 'or', 'restaurant', 'of', 'their', 'own', '.']

# 문장 : Sentence Tokenization (eng)

문장 단위로 토큰화할 필요가 있는 경우가 있다.

이 경우 온점(.)이나 !나 ?로 구분하면 되지 않을까?란 생각을 하게된다.

방금 본 예제에 적용해보자.

Starting a home-based restaurant may be an ideal. it doesn't have a food chain or restaurant of their own.



['Starting a home-based restaurant may be an ideal',  
'it doesn't have a food chain or restaurant of their own']

**잘 되는 것 같은데?**

# 문장 : Sentence Tokenization (eng)

!나 ?는 꽤 명확한 구분자가 되지만 온점을 문장 구분자(Boundary)로 판단하는 것은 생각보다 잘 동작하지 않는다.

반례

Since I'm actively looking for Ph.D. students, I get the same question a dozen times every year.



['Since I'm actively looking for **Ph**', '**D**', 'students, I get the same question a dozen times every year']

제대로 된 결과라면 Ph.D.는 분리되어서는 안 된다.

# 문장 : Sentence Tokenization (kor)

!나 ?는 꽤 명확한 구분자가 되지만 온점을 문장 구분자(Boundary)로 판단하는 것은 생각보다 잘 동작하지 않는다.

반례

IP 192.168.56.31 서버에 들어가서 로그 파일 저장해서 ukairia777@gmail.com로 결과 좀 보내줘. 그리고나서 점심 먹으러 가자.



['IP 192', '168', '56', '31 서버에 들어가서 로그 파일 저장해서 ukairia777@gmail', 'com로 결과 좀 보내줘', '그리고나서 점심 먹으러 가자 ']

**문장 구분조차 보다 섬세한 규칙이 필요하다.**

# 문장 : NLTK Sentence Tokenizer (eng)

영어권 언어의 경우 다양한 Sentence Tokenizer가 있는데,  
그 중에서 NLTK Sentence Tokenizer의 동작을 보자.

앞서 잘 되지 않았던 문장

Since I'm actively looking for Ph.D. students, I get the same question a dozen times every year.



['I am actively looking for Ph.D. students.', 'and you are a Ph.D student.']

**단순 온점을 기준으로 구분하지 않으며 제대로 동작한다.**



# Tokenization (kor)

# 문장 : KSS(Korean Sentence Splitter)

한국어의 경우 문장 토큰화 패키지는 KSS 사용을 권장.

## 동작 예시

'딥 러닝 자연어 처리가 재미있기는 합니다. 그런데 문제는 영어보다 한국어로 할 때 너무 어려워요. 농담아니예요. 이제 해보면 알걸요?'



['딥 러닝 자연어 처리가 재미있기는 합니다.', '그런데 문제는 영어보다 한국어로 할 때 너무 어려워요.', '농담아니예요.', '이제 해보면 알걸요?']

링크 : <https://github.com/likejazz/korean-sentence-splitter>

# 한국어는 영어보다 자연어 처리가 훨씬 어렵다.

1. 한국어는 교착어이다.
2. 한국어는 띄어쓰기가 잘 지켜지지 않는다.
3. 한국어는 어순이 그렇게 중요하지 않다.
4. 한자어라는 특성상 하나의 음절조차도 다른 의미를 가질 수 있다.
5. 주어가 손쉽게 생략된다.
6. 데이터가 영어에 비해 너무 부족하다.
7. 그 외 여러가지 이유 기타 등등등...

# 한국어는 띄어쓰기가 잘 지켜지지 않는다.

한국어는 띄어쓰기가 어렵고, 지키지 않더라도 쉽게 읽을 수 있다.  
대부분의 데이터에서 띄어쓰기가 잘 지켜지지 않는 경향이 있다.

제가 이렇게 띄어쓰기를 전혀 하지 않고 글을 썼다고 하더라도 글을 쉽게 이해할 수 있습니다.

반면 영어는 띄어쓰기를 하지 않으면 읽기 어려운 언어의 특성으로 인해 띄어쓰기가 꽤 엄격하게 지켜지는 편이다.

**tobeornottobethatisthequestion**

# 한국어는 띄어쓰기가 잘 지켜지지 않는다.

한국어는 띄어쓰기가 어렵고, 지키지 않더라도 쉽게 읽을 수 있다.  
대부분의 데이터에서 띄어쓰기가 잘 지켜지지 않는 경향이 있다.

제가 이렇게 띄어쓰기를 전혀 하지 않고 글을 썼다고 하더라도 글을 쉽게 이해할 수 있습니다.

반면 영어는 띄어쓰기를 하지 않으면 읽기 어려운 언어의 특성으로 인해 띄어쓰기가 꽤 엄격하게 지켜지는 편이다.

`tobeornottobethatisthequestion`

결국 띄어쓰기를 보장해주어야 하는 전처리가 필요할 수도 있다.

# 띄어쓰기 보정 : KoSpacing

- KoSpacing

잘못된 띄어쓰기를 보정해주는 딥 러닝 기반의 패키지.

오지호는극중두얼굴의사나이성준역을맡았다.성준은국내유일의태백권전승자를가리는결전의날을앞두고20년간동고동락한사형인진수(정의육분)를찾으러속세로내려온인물이다.

동작 예시



오지호는 극중 두 얼굴의 사나이 성준 역을 맡았다. 성준은 국내 유일의 태백권 전승자를 가리는 결전의 날을 앞두고 20년간 동고동락한 사형인 진수(정의육 분)를 찾으러 속세로 내려온 인물이다.

링크 : <https://github.com/haven-jeon/KoSpacing>

# 한국어는 주어 생략은 물론 어순도 중요하지 않다.

같은 의미의 문장을 다음과 같이 자유롭게 쓸 수 있다.  
3번의 경우 주어까지 생략했다.

1. 나는 운동을 했어. 체육관에서.
2. 나는 체육관에서 운동을 했어.
3. (나는) 체육관에서 운동했어.
4. 나는 운동을 체육관에서 했어.

# 한국어는 주어 생략은 물론 어순도 중요하지 않다.

같은 의미의 문장을 다음과 같이 자유롭게 쓸 수 있다.  
3번의 경우 주어까지 생략했다.

1. 나는 운동을 했어. 체육관에서.
2. 나는 체육관에서 운동을 했어.
3. (나는) 체육관에서 운동했어.
4. 나는 운동을 체육관에서 했어.

뒤에서 배우겠지만 다음 단어를 예측하는 Language Model에게는 매우 혼란스러운 상황



# 한국어는 교착어이다.

교착어란 실질적인 의미를 가지는 어간에 조사나, 어미와 같은 문법 형태소들이 결합하여 문법적인 기능이 부여되는 언어를 말한다.

가령, 한국어에는 영어에는 없는 '은, 는, 이, 가, 를' 등과 같은 조사가 존재한다.

## 예시문

'**사과**가 건강에 좋다고 하더라구. **사과**에 비타민이 많다잖아. 그래서 **사과**를 사러 근처 슈퍼에 갔더니 **사과** 가격이 글썄 3개에 5천원이라고 해서 사지 않았어.'

이 예시문을 띄어쓰기 단위로 토큰화를 할 경우에는 '사과가', '사과에', '사과를', '사과'가 전부 다른 단어로 간주된다.

# 형태소 분석기

교착어인 한국어의 특성으로 인해 한국어는 토큰나이저로 형태소 분석기를 사용하는 것이 보편적이다.

## 예시문

'**사과**가 건강에 좋다고 하더라구. **사과**에 비타민이 많다잖아. 그래서 **사과**를 사러 근처 슈퍼에 갔더니 **사과** 가격이 글썄 3개에 5천원이라고 해서 사지 않았어.'



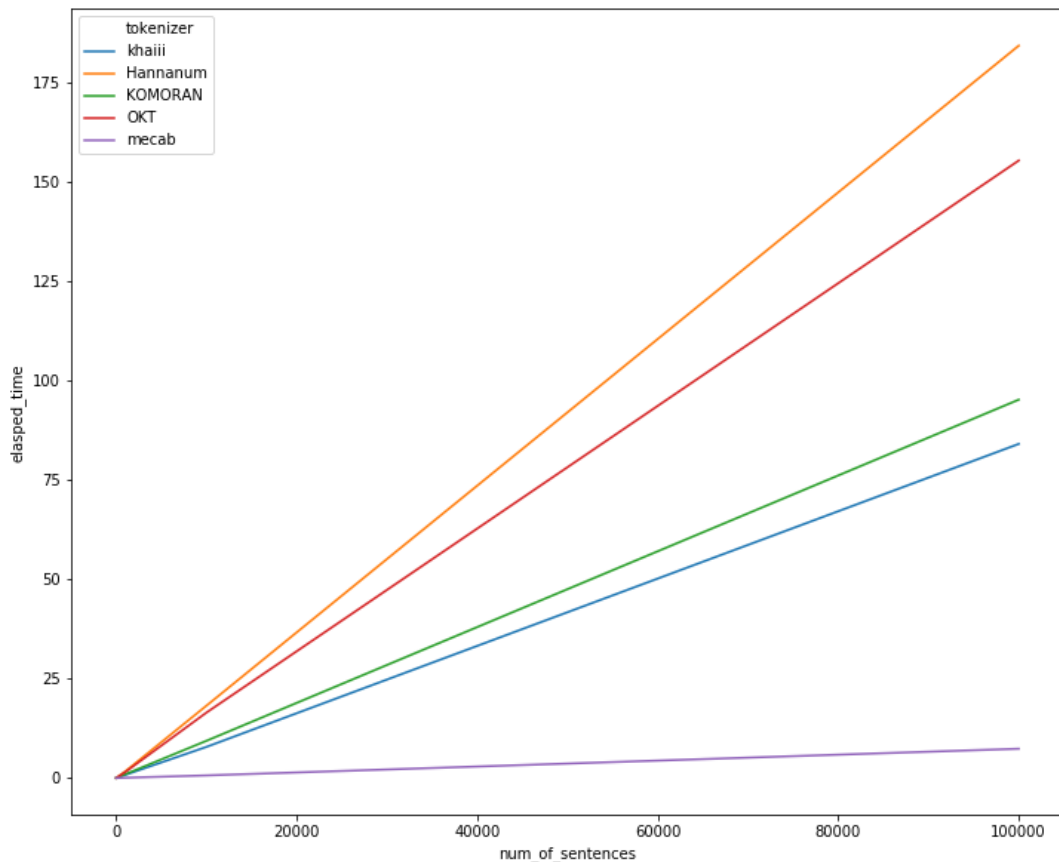
## 형태소 분석기 중 OKT 사용

['**사과**', '가', '건강', '에', '좋다고', '하더라', '구', '.', '**사과**', '에', '비타민', '이', '많다잖아', '.', '그래서', '**사과**', '를', '사러', '근처', '슈퍼', '에', '갔더니', '**사과**', '가격', '이', '글썄', '3', '개', '에', '5천원', '이라고', '해서', '사지', '않았어', '.']

# 형태소 분석기

다양한 형태소 분석기가 존재하므로 원하는 Task에 맞는 형태소 분석기를 선택.

- Mecab
- Kkma
- Okt
- Hannanum
- Khaii
- Soynlp



- Mecab은 연산 속도가 빠르고 분석 성능도 준수하여 선호도가 높음.
- Khaii는 가장 최근에 나온 분석기로 딥 러닝 기반 형태소 분석기.
- KOMORAN은 오타자에 강건함.
- Soynlp는 학습 기반으로 복합 명사를 잘 추출해낼 수 있음.

속도 비교 그래프 출처 :

<https://iostream.tistory.com/144>

# 형태소 분석기 결과 비교

## 임포트

```
from konlpy.tag import *  
  
hannanum = Hannanum()  
kkma = Kkma()  
komoran = Komoran()  
okt = Okt()  
mecab = Mecab()
```

## 입력 문장

“열심히 코딩한 당신, 연휴에는 여행을 가봐요”

## 상이한 결과

한나눔 : ['열심히', '코딩', '하', 'ㄴ', '당신', ',', '연휴', '에는', '여행', '을', '가', '아', '보', '아']  
꼬꼬마 : ['열심히', '코딩', '하', 'ㄴ', '당신', ',', '연휴', '에', '는', '여행', '을', '가보', '아요']  
꼬모란 : ['열심히', '코', '딩', '하', 'ㄴ', '당신', ',', '연휴', '에', '는', '여행', '을', '가', '아', '보', '아요']  
Okt : ['열심히', '코딩', '한', '당신', ',', '연휴', '에는', '여행', '을', '가봐요']  
메캅 : ['열심히', '코딩', '한', '당신', ',', '연휴', '에', '는', '여행', '을', '가', '봐요']

# Clearning and Norm

# Cleaning and Normalization

- **Cleaning(정제) : 불필요한 데이터를 제거하는 일**

- 정규 표현식을 이용한 노이즈 데이터 제거
- 인코딩 문제 해결
- 등장 빈도가 적은 단어 제거 : 등장 빈도가 2회 이하
- 길이가 짧은 단어 제거(영어의 경우) : I, by, at 등...
- 불용어 제거 : I, at, for, by, at, 은, 는, 이, 가

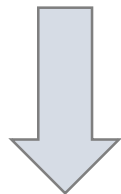
- **Normalization(정규화) : 같은 의미를 갖고 있다면 하나로 통일하여 복잡도를 줄임.**

- am, are, were, was → be (lemmatization)
- has, had → have (lemmatization)
- 10, 159, 123 → num (숫자가 중요하지 않을 경우)
- ㅋㅋ, ㅋㅋㅋ, ㅋㅋㅋㅋㅋㅋ → ㅋㅋ
- Hmmmmm, Hmmmm, hmmm → hmm
- 대소문자 통합

# Cleaning

## 정규 표현식(Regular Expression)을 이용한 정제의 예

'Hmm...포스터보고 초딩영화인줄.....오버연기조차 가볍지 않구나!!'



`.replace("[^ㄱ-ㅎㅌ-ㅣ가-힣 ]","")`

**한국어를 제외하고 모두 제거**

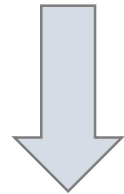
'포스터보고 초딩영화인줄오버연기조차 가볍지 않구나'

# Cleaning

이 수업에서 다루지는 않지만 꼭 공부하셔야 됩니다.

## 정규 표현식(Regular Expression)을 이용한 정제의 예

'Hmm...포스터보고 초딩영화인줄.....오버연기조차 가볍지 않구나!!'



`.replace("[^ㄱ-ㅎㅌ-ㅣ가-힣 ]","")`

한국어를 제외하고 모두 제거

'포스터보고 초딩영화인줄오버연기조차 가볍지 않구나'



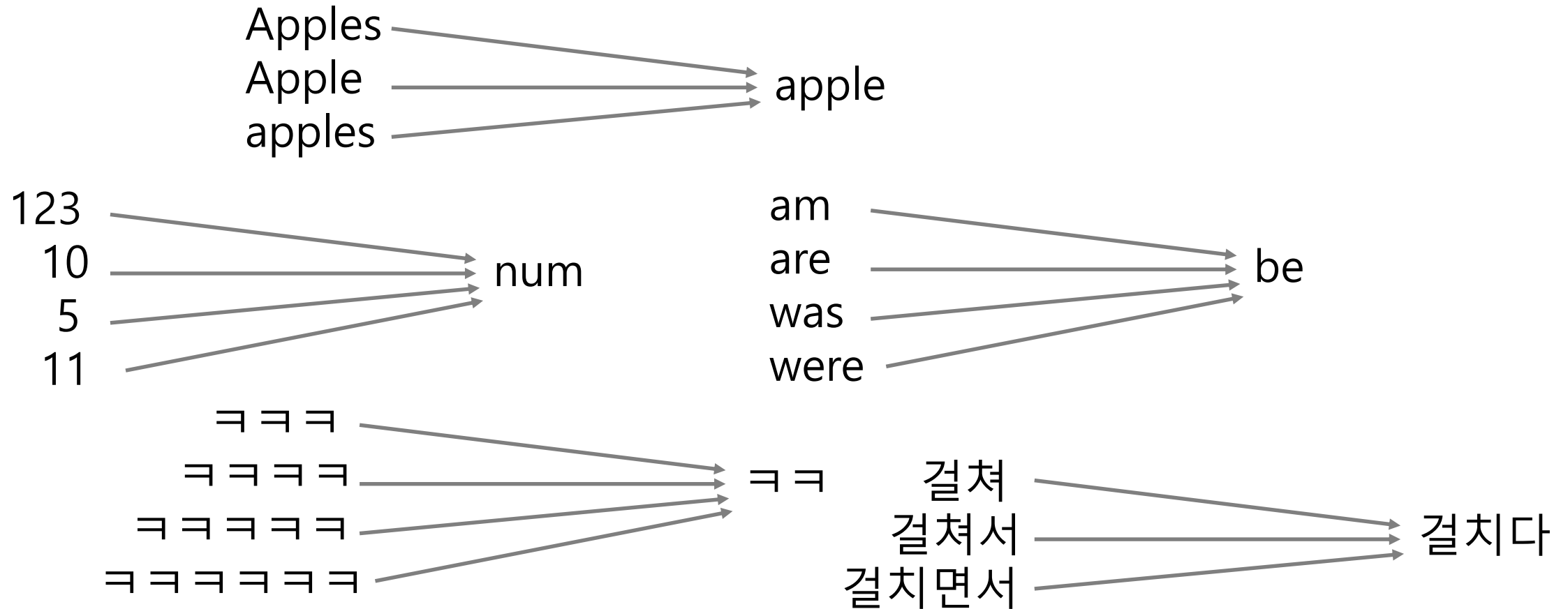
# Cleaning – 불용어 제거(Removing Stopwords)

**자주 등장하지만 실제 분석을 하는데는 거의 기여하는 바가 없는 단어들.**

- 영어의 경우 'the'는 거의 모든 텍스트 데이터에서 등장 빈도수가 가장 높은 단어지만 실제 의미는 갖고있지 않으므로 대표적인 불용어.
- NLTK에서 정의한 영어 불용어로는 'i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've" ... 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't" 등이 존재.
- 한국어의 경우 '그럼', '위하', '때', '있', '그것', '사실', '경우', '어떤', '은', '를' 등이 존재.
- 불용어는 절대적인 기준이 아니라 어떤 데이터인지, 어떤 문제를 푸는지, 어떤 토크나이저를 사용하는지에 따라서 달라진다.

# Normalization

다르게 표기된 단어들을 하나로 통합 (Normalization)



# Cleaning and Normalization example (Eng)

```
text = sent_tokenize(text) # 문장 토큰화
print(text)

sentences = []
stop_words = set(stopwords.words('english')) # NLTK 불용어

for i in text:
    sentence = word_tokenize(i) # 단어 토큰화
    result = []

    for word in sentence:
        word = word.lower() # 모든 단어를 소문자화하여 단어의 개수를 줄입니다.
        if word not in stop_words: # 불용어를 제거합니다.
            if len(word) > 2: # 단어 길이가 2미하인 경우에 대하여 추가로 단어를 제거합니다.
                result.append(word)
    sentences.append(result)
print(sentences)
```

# Normalization (Kor)

다르게 표기된 문자들을 하나로 통합 (Normalization)  
정규 표현식 또는 `soynlp.normalizer`를 사용.

```
from soynlp.normalizer import *
```

```
print(emoticon_normalize('알ㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_repeats=2))  
print(emoticon_normalize('알ㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠ', num_repeats=2))  
print(emoticon_normalize('알ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_repeats=2))  
print(emoticon_normalize('알ㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋㅋ이영화존잼쓰ㅠㅠㅠㅠ', num_repeats=2))
```

```
아ㅋㅋ 영화존잼쓰 ㅠㅠ  
아ㅋㅋ 영화존잼쓰 ㅠㅠ  
아ㅋㅋ 영화존잼쓰 ㅠㅠ  
아ㅋㅋ 영화존잼쓰 ㅠㅠ
```

```
print(repeat_normalize('와하하하하하하하하하하', num_repeats=2))  
print(repeat_normalize('와하하하하하하하', num_repeats=2))  
print(repeat_normalize('와하하하하하', num_repeats=2))
```

```
와하하하  
와하하하  
와하하하
```

# 맞춤법 검사기 (Kor)

## hanspell.spell.checker는 맞춤법 및 띄어쓰기 정정 기능 제공

```
from hanspell import spell_checker

sent = "맞춤법 틀리면 외 앞뒤?"
spelled_sent = spell_checker.check(sent)

hanspell_sent = spelled_sent.checked
print(hanspell_sent)
```

맞춤법 틀리면 왜 안돼?

```
sent = "오지호는극중두얼굴의사나이성준역을맡았다.성준은국내유일의태백권전승자를가리는결전의날을앞두고20년간동고동락한사형인진수(정의욱분)를찾으러속세로내려온인물이다."
spelled_sent = spell_checker.check(sent)

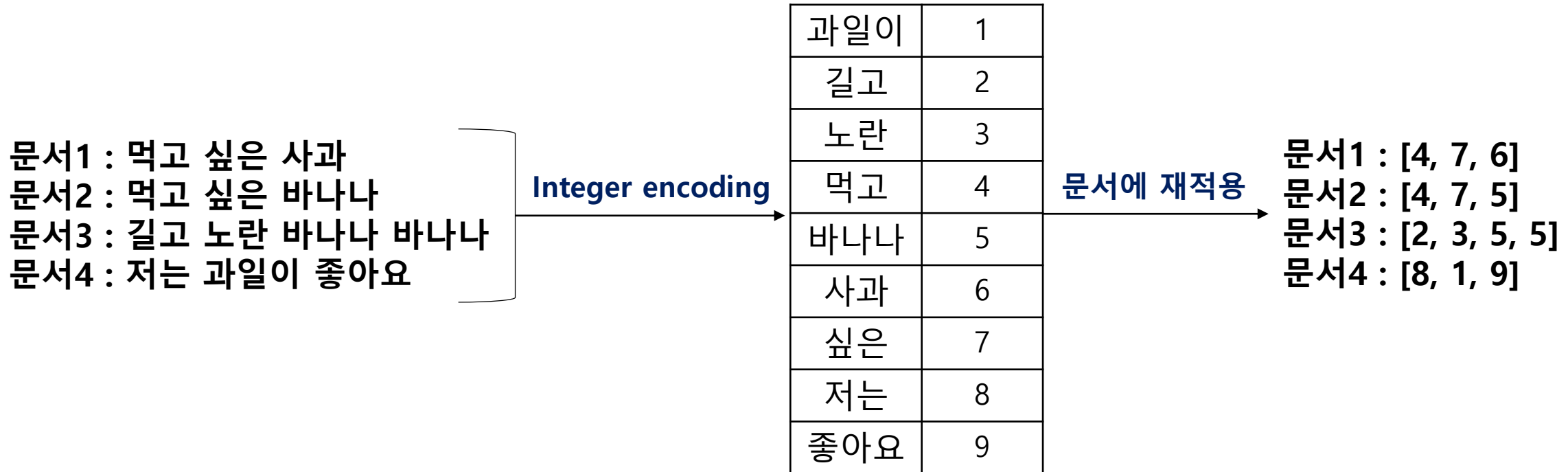
hanspell_sent = spelled_sent.checked
print(hanspell_sent)
print(kospacing_sent) # 앞서 사용한 kospacing 패키지에서 얻은 결과
```

오지호는 극 중 두 얼굴의 사나이 성준 역을 맡았다. 성준은 국내 유일의 태백권 전승자를 가리는 결전의 날을 앞두고 20년간 동고동락한 사형인 진수(정의욱 분)를 찾으러 속세로 내려온 인물이다.  
오지호는 극중 두 얼굴의 사나이 성준 역을 맡았다. 성준은 국내 유일의 태백권 전승자를 가리는 결전의 날을 앞두고 20년간 동고동락한 사형인 진수(정의욱 분)를 찾으러 속세로 내려온 인물이다.

# 텍스트의 수치화

# Integer Encoding

- 단어 토큰화 또는 형태소 토큰화를 수행했다면 각 단어에 고유한 정수를 부여한다.
- 이때 중복은 허용하지 않는다.
- 중복이 허용되지 않는 모든 단어들의 집합을 단어 집합(Vocabulary)이라고 한다.



# Padding

- 모든 문장에 대해서 정수 인코딩을 수행하였을 때 길이는 서로 다를 수 있다.
- 이때 가상의 단어를 추가하여 길이를 맞춰준다.
- 이렇게 하면 기계가 이를 병렬 연산할 수 있다.

과일이	1
길고	2
노란	3
먹고	4
바나나	5
사과	6
싶은	7
저는	8
좋아요	9

문서에 재적용

문서1 : [4, 7, 6]  
문서2 : [4, 7, 5]  
문서3 : [2, 3, 5, 5]  
문서4 : [8, 1, 9]

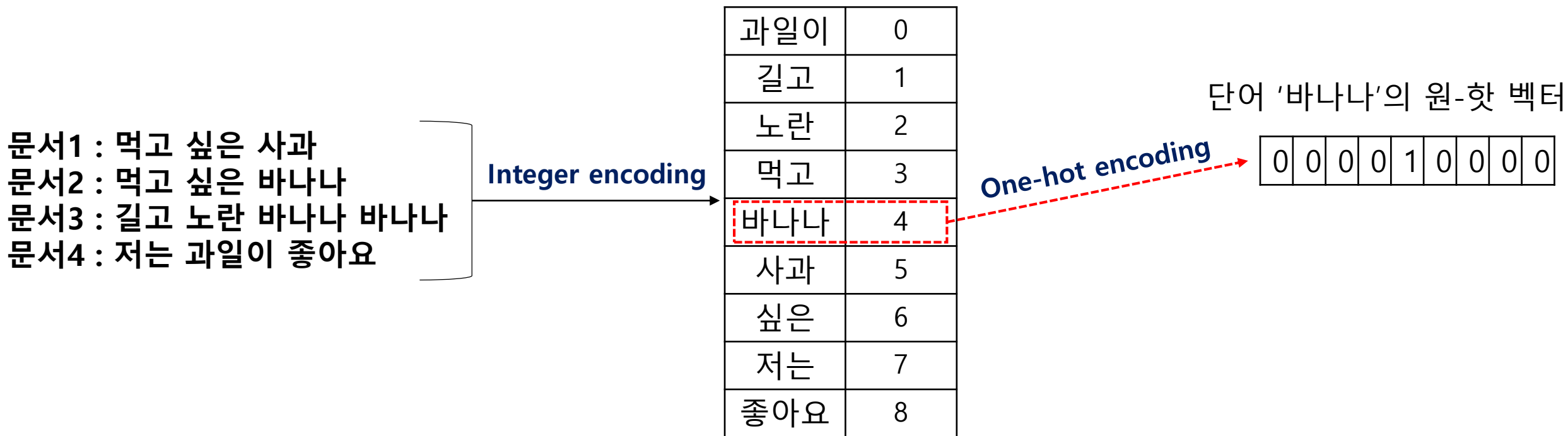
0번 단어 'PAD'추가

문서1 : [4, 7, 6, 0]  
문서2 : [4, 7, 5, 0]  
문서3 : [2, 3, 5, 5]  
문서4 : [8, 1, 9, 0]



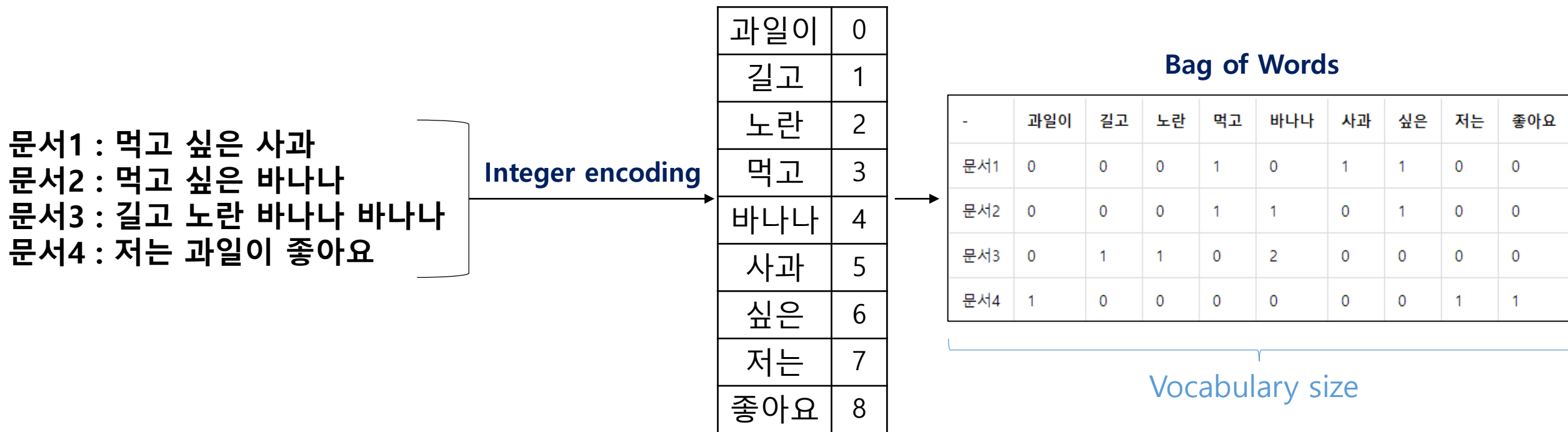
# Vectorization : One-Hot encoding

- 워드 임베딩에 대해서 다루기 전에 원-핫 인코딩이란 방법부터 알아본다.
- 원-핫 인코딩은 전체 단어 집합의 크기(중복은 카운트하지 않은 단어들의 개수)를 벡터의 차원으로 가진다.
- 각 단어에 고유한 정수 인덱스를 부여하고, 해당 인덱스의 원소는 1, 나머지 원소는 0을 가지는 벡터로 만든다.



# Vectorization : Document Term Matrix, DTM

- **DTM**은 마찬가지로 벡터가 단어 집합의 크기를 가지며, 대부분의 원소가 0을 가진다.
- 각 단어는 고유한 정수 인덱스를 가지며, 해당 단어가 등장 횟수를 해당 인덱스의 값으로 가진다.



# 구글 Colab 노트북 실습

# Language Model

# Language Model

- Language Model, 줄여서 LM이란 단어 시퀀스에 확률을 할당하는 모델.
- 단어 시퀀스에 확률을 할당하는 이유? LM을 통해 더 그럴듯한 문장을 선택할 수 있으니까!

*아래에서  $P$ 는 확률(Probability)을 나타낸다.*

## 기계 번역(Machine Translation)

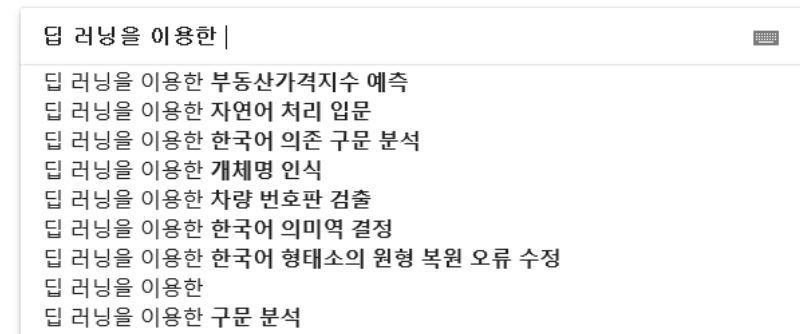
-  $P(\text{나는 버스를 탔다}) > P(\text{나는 버스를 태운다})$

## 음성 인식(Spell Correction)

-  $P(\text{나는 메롱을 먹는다}) < P(\text{나는 메론을 먹는다})$

## 오타 교정(Spell Correction)

-  $P(\text{선생님이 달려갔다}) > P(\text{선생님이 잘려갔다})$



<검색 엔진에서의 언어 모델이 동작하는 예 : 다음 단어 예측>

# Language Model

- Language Model은 어떤 문장이 가장 그럴듯한지를 판단한다.
- Language Model은 주어진 단어 시퀀스 다음에 어떤 단어가 등장해야 자연스러운지를 판단한다.

## 1. 전체 단어 시퀀스의 확률

-  $P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = n$ 개의 단어가 동시에 등장할 확률

## 2. 이전 단어들이 주어졌을 때 다음 단어의 등장 확률

다섯번째 단어의 등장 확률은 다음과 같이 표현할 수 있다.

$$P(w_5 | w_1, w_2, w_3, w_4)$$

일반화하면 전체 단어 시퀀스의 확률은 아래와 같이 정의할 수 있다.

-  $P(W) = P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$

# Language Model

- Language Model은 어떤 문장이 가장 그럴듯한지를 판단한다.
- Language Model은 주어진 단어 시퀀스 다음에 어떤 단어가 등장해야 자연스러운지를 판단한다.

## 예문으로 보는 단어 시퀀스 확률

$$\begin{aligned} P(W) &= P(w_1, w_2, w_3, w_4, w_5, \dots, w_n) \\ &= \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1}) \end{aligned}$$

위 수식에 따라서  $P(\text{"its water is so transparent"}) =$

$$\begin{aligned} &P(\text{its}) \times P(\text{water} | \text{its}) \times P(\text{is} | \text{its water}) \\ &\times P(\text{so} | \text{its water is}) \times P(\text{transparent} | \text{its water is so}) \end{aligned}$$

**그래, 수식은 알았는데 그래서 도대체 어떻게 계산하는데?**

# Statistical Language Model

- 통계적 방법 : 훈련 데이터에서 단순 카운트하고 나누는 것은 어떨까?

$$P(\text{the | its water is so transparent that}) \\ = \frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

통계적 언어 모델의 접근 방법 :

Its water is so transparent that 다음에 the가 나올 확률은 전체 훈련 데이터에서  
Its water is so transparent that the의 등장 횟수를 카운트한 것을 분자로  
Its water is so transparent that의 등장 횟수를 카운트한 것을 분모로 한다.



# Statistical Language Model

- 통계적 방법 : 훈련 데이터에서 단순 카운트하고 나누는 것은 어떨까?

$$P(\text{the | its water is so transparent that}) \\ = \frac{\text{Count}(\text{its water is so transparent that the})}{\text{Count}(\text{its water is so transparent that})}$$

훈련 데이터가 정-말 방대하지 않은 이상 제대로 카운트할 수 있는 경우가 거의 없다.

통계적 언어 모델의 접근 방법 :

Its water is so transparent that 다음에 the가 나올 확률은 전체 훈련 데이터에서  
Its water is so transparent that the의 등장 횟수를 카운트한 것을 분자로  
Its water is so transparent that의 등장 횟수를 카운트한 것을 분모로 한다.

# Statistical Language Model

훈련 데이터가 정-말 방대하지 않은 이상 제대로 카운트할 수 있는 경우가 거의 없다.



The screenshot shows a Google search interface. The search bar contains the text "그 물은 투명했다". Below the search bar, there are tabs for "전체" (All), "이미지" (Images), "동영상" (Videos), "뉴스" (News), "지도" (Maps), and "더보기" (More). The search results show approximately 5,970,000 results in 0.46 seconds. The first result is titled "'투명한' 전극 만드는 방법 찾았다 - 헬로디디" (Method for making 'transparent' electrodes found - Hello Didi) from www.hellodd.com, dated May 31, 2018. The second result is titled "보이지 않는 금속전극 개발, 투명망토 나올까 - Sciencetimes" (Development of invisible metal electrodes, will the invisibility cloak come out - Sciencetimes) from https://www.sciencetimes.co.kr, dated May 31, 2018. The third result is titled "투명 망토 현실로?...韓 연구진, 99% 투명한 금속 전극 개발 - IT ..." (Invisibility cloak becomes reality?... Korean researchers develop 99% transparent metal electrodes - IT ...) from it.chosun.com, dated May 31, 2018.

방대한 데이터의 집합체인 구글 검색 엔진에서조차  
'그 물은 투명했다' 라는 간단한 문장이  
포함된 결과가 나오지 않는다.

언어라는 것은 너무나 가능한 경우의 수가 많다.

단순 카운트 방법으로 언어를 모델링하기에는  
그만큼의 방대한 훈련 데이터를 가지기 어렵다.

# N-gram Language Model

- n-gram은 n개의 연속적인 단어 나열을 의미하며 n은 사용자가 정하는 값이다.
- 이전 단어들의 주어졌을 때 다음 단어의 등장 확률의 추정을 앞의 n-1개의 단어에만 의존한다.

' An adorable little boy is spreading ' 다음에 나올 단어를 예측하고자 하며 n이 4일 때 (예시)

~~An adorable little~~ boy is spreading ?  
무시됨!  
n-1개의 단어

$$P(w|\text{boy is spreading}) = \frac{\text{count}(\text{boy is spreading } w)}{\text{count}(\text{boy is spreading})}$$

통계적 언어 모델

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i | w_1, \dots, w_{i-1})$$

N-gram 언어 모델 (if n=4)

$$P(w_1, w_2, \dots, w_n) \approx \prod_{i=1}^n P(w_i | w_{i-3}, w_{i-2}, w_{i-1})$$

# Bigram Language Model (n=2)

- Bigram Language Model로 생성한 문장은 다음 단어 생성이 오직 이전 단어에만 의존한다.

## Bigram Language Model이 생성한 문장

texaco, rose, one, in, this, issue, is, pursuing, growth, in,  
a, boiler, house, said, mr., gurria, mexico, 's, motion,  
control, proposal, without, permission, from, five, hundred,  
fifty, five, yen

outside, new, car, parking, lot, of, the, agreement, reached

this, would, be, a, record, november

# Bigram Language Model (n=2)

- n=2이면 Bigram Language Model이라고 하며 오직 이전 단어 하나만 고려하여 카운트하여 확률 추정.
- n이 작으면 알토당토 않는 문장이 되버리지만 n이 크면 카운트하기가 어려워짐. 적절한 n 선정이 중요.

## 1. 다음 단어 등장 확률 추정

$$P(w_i | w_1 w_2 \dots w_{i-1}) \approx P(w_i | w_{i-1})$$



## 2. 확률 추정은 훈련 데이터에서 카운트에 기반함.

$$P(w_i | w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})}$$

## 3. 전체 단어 시퀀스의 확률 추정

$$P(w_1, w_2, \dots, w_n) = \prod_{i=1}^n P(w_i | w_{i-1})$$



# Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.

$\langle s \rangle$ 와  $\langle /s \rangle$ 는 special token으로  $\langle s \rangle$ 는 문장의 시작,  $\langle /s \rangle$ 는 문장의 끝을 의미한다.

$\langle s \rangle$  I am Sam  $\langle /s \rangle$

$\langle s \rangle$  Sam I am  $\langle /s \rangle$

$\langle s \rangle$  I do not like green eggs and ham  $\langle /s \rangle$

계산식

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(\text{I} | \langle s \rangle) = ?$$

$$P(\text{Sam} | \langle s \rangle) = ?$$

# Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.

<s>와 </s>는 special token으로 <s>는 문장의 시작, </s>는 문장의 끝을 의미한다.

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

계산식

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(\text{I} | \text{<s>}) = \frac{2}{3} = .67$$

$$P(\text{Sam} | \text{<s>}) = \frac{1}{3} = .33$$

# Bigram Language Model Example

다음과 같이 3개의 문장을 가진 훈련 데이터가 있다고 하자.

$\langle s \rangle$ 와  $\langle /s \rangle$ 는 special token으로  $\langle s \rangle$ 는 문장의 시작,  $\langle /s \rangle$ 는 문장의 끝을 의미한다.

$\langle s \rangle$  I am Sam  $\langle /s \rangle$

$\langle s \rangle$  Sam I am  $\langle /s \rangle$

$\langle s \rangle$  I do not like green eggs and ham  $\langle /s \rangle$

계산식

$$P(w_i | w_{i-1}) = \frac{c(w_{i-1}, w_i)}{c(w_{i-1})}$$

$$P(\text{I} | \langle s \rangle) = \frac{2}{3} = .67 \quad P(\text{Sam} | \langle s \rangle) = \frac{1}{3} = .33 \quad P(\text{am} | \text{I}) = \frac{2}{3} = .67$$

$$P(\langle /s \rangle | \text{Sam}) = \frac{1}{2} = 0.5 \quad P(\text{Sam} | \text{am}) = \frac{1}{2} = .5 \quad P(\text{do} | \text{I}) = \frac{1}{3} = .33$$



# Sparsity Problem

- N-gram 언어 모델은 앞의 단어를 n-1개만 고려하므로 n이 너무 작으면 **장기 의존성 문제 발생**.
- n이 커지면 훈련 데이터 내에서 카운트 자체를 하지 못해서 제대로 된 모델링이 되지 않음.
- 카운트 자체를 하지 못할 경우 분자 또는 분모가 0이 되는 상황 발생.

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

카운트가 0이면?

카운트가 0이면?

충분한 데이터를 관측하지 못하여  
언어를 정확히 모델링하지 못하는 문제를  
**희소 문제(sparsity problem)**라고 합니다.

# Sparsity Problem

- N-gram 언어 모델은 앞의 단어를 n-1개만 고려하므로 n이 너무 작으면 장기 의존성 문제 발생.
- n이 커지면 훈련 데이터 내에서 카운트 자체를 하지 못해서 제대로 된 모델링이 되지 않음.
- 카운트 자체를 하지 못할 경우 분자 또는 분모가 0이 되는 상황 발생.

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

카운트가 0이면?

카운트가 0이면?

충분한 데이터를 관측하지 못하여  
언어를 정확히 모델링하지 못하는 문제를  
희소 문제(sparsity problem)라고 합니다.

이를 위해 백오프, 스무딩과 같은 대안들이 나왔습니다. 이는 여기서 다루지 않겠습니다.

# Sparsity Problem

- N-gram 언어 모델은 앞의 단어를 n-1개만 고려하므로 n이 너무 작으면 **장기 의존성 문제 발생**.
- n이 커지면 훈련 데이터 내에서 카운트 자체를 하지 못해서 제대로 된 모델링이 되지 않음.
- 카운트 자체를 하지 못할 경우 분자 또는 분모가 0이 되는 상황 발생.

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

카운트가 0이면?

카운트가 0이면?

충분한 데이터를 관측하지 못하여  
언어를 정확히 모델링하지 못하는 문제를  
**희소 문제(sparsity problem)**라고 합니다.

**결국 인공 신경망 언어모델(NNLM)의 시대로!**

# Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룹니다.
- 인공 신경망 언어 모델은 희소 문제(sparsity problem)를 Word Embedding으로 해결하였습니다.

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있습니다.

귀여운  $\approx$  사랑스러운

강아지  $\approx$  고양이

좋아해  $\approx$  사랑해

# Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룹니다.
- 인공 신경망 언어 모델은 희소 문제(sparsity problem)를 Word Embedding으로 해결하였습니다.

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있습니다.

귀여운  $\approx$  사랑스러운

강아지  $\approx$  고양이

좋아해  $\approx$  사랑해

이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능합니다.

# Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룹니다.
- 인공 신경망 언어 모델은 희소 문제(sparsity problem)를 Word Embedding으로 해결하였습니다.

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있습니다.

귀여운  $\approx$  사랑스러운

강아지  $\approx$  고양이

좋아해  $\approx$  사랑해

이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능합니다.

훈련 데이터 : 귀여운 강아지 좋아해  $\longrightarrow$

# Neural Network Language Model (예고)

- 인공 신경망 언어 모델에 대해서는 3강 Word Embedding 챕터에서 다룹니다.
- 인공 신경망 언어 모델은 희소 문제(sparsity problem)를 Word Embedding으로 해결하였습니다.

Word Embedding을 사용하면 단어 벡터 간 유사도를 얻을 수 있습니다.

귀여운  $\approx$  사랑스러운

강아지  $\approx$  고양이

좋아해  $\approx$  사랑해

이로부터 훈련 데이터에 없던 시퀀스라도 생성이 가능합니다.

훈련 데이터 : 귀여운 강아지 좋아해  $\longrightarrow$  사랑스러운 고양이 사랑해 (훈련 데이터에 없던 시퀀스)