



9주차 - KLUE

한국어 언어 모델 학습 및 다중 과제 튜닝

1. 인공지능과 자연어 처리

인공지능의 근본은 NLP.

현대에서는 다양한 자연어 처리 application 이 개발되었습니다.

다양한 자연어처리 application		
<ul style="list-style-type: none">• 문서 분류• 문법, 오타 교정• 정보 추출• 음성 인식 결과 보정• 음성 합성 텍스트 보정• 정보 검색• 요약문 생성• 기계 번역• 질의 응답	<ul style="list-style-type: none">• 기계 독해• 챗봇• 형태소 분석• 개체명 분석• 구문 분석• 감성 분석• 관계 추출• 의도 파악• 이미지 캡셔닝	<ul style="list-style-type: none">• 텍스트 압축• 패러프레이징• 주요 키워드 추출• 빈칸 맞추기• 발음기호 변환• 소셜 생성• 텍스트 기반 게임• 오픈 도메인 QA• 가설 검증

인간의 자연어 처리

- 대화의 단계
 - 화자는 자연어 형태로 객체를 인코딩
 - 메시지의 전송

- 청자는 자연어를 객체로 디코딩

화자는 청자가 이해할 수 있는 방법으로 인코딩

청자는 본인 지식을 바탕으로 디코딩

컴퓨터의 자연어 처리

- 대화의 단계
 - 인코더는 벡터 형태로 자연어를 인코딩
 - 메시지의 전송
 - 디코더는 벡터를 자연어로 디코딩

자연어를 컴퓨터가 이해할 수 있게 수학적으로 어떻게 인코딩할 수 있는지 알아보자.

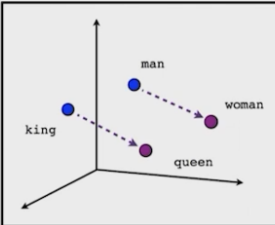
자연어 단어 임베딩

- 자연어를 어떻게 좌표 평면 위에 표현해야 하는가?
- 단순한 표현 방법은 원-핫 인코딩 (Sparse representation)

다만 sparse해져서 단어가 가지는 의미를 공간에 표현이 불가능하다.

- word2vec : 자연어(단어)의 의미를 벡터 공간에 임베딩.
 - 한 단어의 주변 단어를 통해 그 단어의 의미를 파악
 - 단어에 대한 DENSE VECTOR를 얻을 수 있음.

Word2Vec



- 단어가 가지는 의미 자체를 다차원 공간에 '벡터화'하는 것
- 중심 단어의 주변 단어들을 이용해 중심단어를 추론하는 방식으로 학습

장점

- 단어간의 유사도 측정에 용이
- 단어간의 관계 파악에 용이
- 벡터 연산을 통한 추론이 가능 (e.g. 한국 - 서울 + 도쿄 = ?)

단점

- 단어의 subword information 무시 (e.g. 서울 vs 서울시 vs 고양시)
- Our of vocabulary (OOV)에서 적용 불가능

FastText

• 한국어는 다양한 용언 형태를 가짐

• Word2Vec의 경우, 다양한 용언 표현들이 서로 독립된 vocab으로 관리

동사 원형: **모르다**

모르네	모르기까지	모르겠으나	몰라야	몰랐다면	몰랐었으나
모르데	모르기를	모르겠으면	몰라요	몰랐다면	몰랐었으면
모르지	모르기는	모르겠으면서	몰라라	몰랐을	몰랐었으면서
모르더라	모르기도	모르겠거나	몰랐다	몰랐을까	몰랐었거나
모르리라	모르기만	모르겠거든	몰랐네	몰랐을지	몰랐었거든
모르는구나	모르는	모르겠는데	몰랐지	몰랐을지도	몰랐었는데
모르잖아	모르면	모르겠지만	몰랐더라	몰랐어	몰랐었지만
모르려나	모른	모르겠더라도	몰랐으리라	몰랐어도	몰랐었더라도
모르니	모른다	모르겠다가도	몰랐구나	몰랐어야	몰랐었다가도
모르고	모른다면	모르겠던	몰랐잖아	몰랐어요	몰랐었던
모르나	모른다만	모르겠다면	몰랐으려나	몰랐더라면	몰랐었다면
모르면	모른답시고	모르겠다만	몰랐으니	몰랐더라도	몰랐었다만
모르면서	모르겠다	모를까	몰랐거나	몰랐겠다	몰랐었어
모르거든	모르겠네	모를지	몰랐거든	몰랐겠네	몰랐었어도
모르는데	모르겠지	모를지도	몰랐는데	몰랐겠지	몰랐었어서
모르지만	모르겠더라	모를수록	몰랐지만	몰랐겠더라	몰랐었어야
모르더라도	모르겠구나	몰라	몰랐더라도	몰랐겠구나	몰랐었어요
모르다가도	모르겠니	몰라도	몰랐다가도	몰랐겠니	몰랐었더라면
모르기조차	모르겠고	몰라서	몰랐던	몰랐겠고	몰랐었더라도

알고리즘 자체는 word2vec과 유사하나, 단어를 N-Gram으로 나누어서 학습을 진행.

n그램의 범위가 2-5 일때, 단어를 다음과 같이 분리하여 학습함.

ex) assumption = as,ss,su, ...ass,sssu,sum,ump,mpt,...assumption

이때, n-gram 으로 나뉜 단어는 사전에 들어가지 않으며, 별도의 n-gram 벡터를 형성.

Fasttext

- Facebook research에서 공개한 open source library (<https://research.fb.com/fasttext/>, fasttext.cc)
- C++11

Training

- 기존의 word2vec과 유사하나, 단어를 n-gram으로 나누어 학습을 수행
- n-gram의 범위가 2-5일 때, 단어를 다음과 같이 분리하여 학습함
"assumption" = {as, ss, su, ..., ass, ssu, sum, ump, mpt, ..., ption, assumption}
- 이 때, n-gram으로 나뉜 단어는 사전에 들어가지 않으며, 별도의 n-gram vector를 형성함

Testing

- 입력 단어가 vocabulary에 있을 경우, word2vec과 마찬가지로 해당 단어의 word vector를 return함
- 만약 OOV일 경우, 입력 단어의 n-gram vector들의 합산을 return함

모든 단어를 n-gram으로 분리를 한 후, 모든 엔그램 벡터를 합산한 후 평균을 통해 단어벡터를 획득

오탈자, oov, 등장횟수 적은 학습 단어에 대해 강세.

다만, 이러한 word embedding 방식은 동형어, 다의어에 대해 임베딩 성능이 좋지 않음.

주변 단어를 통해 학습이 이루어지기에, 문맥을 고려할 수 없음.

모델이란?

[IT용어]모형(model)

1. ① 어떤 상황이나 물체 등 연구 대상 주제를 도면이나 사진 등 화상을 사용하거나 수식이나 악보와 같은 기호를 사용하여 표현한 것. 표현 양식에 따라서 화상 모형(graphical model)이나 기호 모형(symbolic model)이라고 하고, 특히 수학적 기호와 수식을 사용하여 표현한 것을 수학적 모형이라고 한다. 분자(molecule)의 도식 모형, 우주의 물질 분포의 수학적 모형, 기업 경영의 표 계산(숫자적) 모형 등이 있다. 모형을 변경하거나 조작하여 그것이 변형, 수정 또는 조건의 변화에 의해 어떻게 달라지는가를 알아낼 수 있다.

- 모델의 종류
 - 일기예보 모델, 데이터 모델, 비즈니스 모델, 물리 모델, 분자 모델 등
- 모델의 특징
 - 자연 법칙을 컴퓨터로 모사함으로써 시뮬레이션이 가능
 - 이전 state를 기반으로 미래의 state를 예측할 수 있음 (e.g. 습도와 바람 세기 등으로 내일 날씨 예측)
 - 즉, 미래의 state를 올바르게 예측하는 방식으로 모델 학습이 가능함

현재 state가 피쳐가 되어, 미래 state를 예측하는 방식으로 학습이 이루어진다.

전통적 모델

마르코프 체인.

기본적인 확률 모델.

RNN 네트워크 : 앞선 단어들의 문맥을 고려해서 만들어진 최종 출력 벡터. - 문맥 벡터
출력된 벡터 값에 대해 CLASSIFICATION LAYER를 붙이면 문장 분류를 위한 신경망 모델.

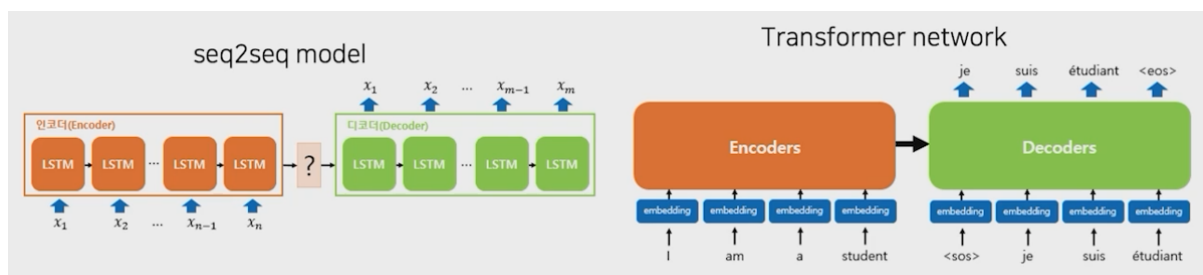
- RNN 구조의 문제점
 - 입력 시퀀스의 길이가 매우 긴 경우, 처음에 나온 토큰에 대한 정보가 희석.
 - 고정된 문맥 벡터 사이즈로 인해 긴 시퀀스에 대한 정보를 함축하기 어려움
 - 모든 토큰이 영향을 미치니, 중요하지 않은 토큰도 영향을 줌.

어텐션

- 인간의 정보처리 시 모든 정보를 받아들이는 것이 아니므로 중요한 것에만 집중한다.
- 문맥에 따라 동적으로 할당되는 인코드의 어텐션 가중치로 인한 동적 문맥 벡터를 획득.

처음엔 RNN과 섞었는데, 그러니 성능이 올라가긴 했지만 여전히 RNN이 순차적으로 연산이 이뤄짐에 따라 연산 속도가 너무도 떨어졌다.

셀프-어텐션 모델이 그런 의미에서 나오게 되었다.



02. 자연어 전처리

자연어 처리의 단계

- 전처리란?
 - 원시 데이터를 기계학습 모델이 학습하는데 적합하게 만드는 프로세스
 - 학습에 사용될 데이터를 수집&가공하는 과정
 - TASK의 성능을 가장 확실하게 올릴 수 있는 방법입니다.
- 자연어 처리의 단계
 - 태스크 설계
 - 필요 데이터 수집
 - 통계학적 분석
 - 토큰 개수 → 아웃라이어 제거
 - 빈도 확인 → 사전(dictionary) 정의
 - 전처리
 - 개행문자 제거
 - 특수문자 제거
 - 공백 제거
 - 중복 표현 제어
 - 이메일, 링크 제거
 - 제목 제거
 - 불용어 (의미 없는 용어) 제거
 - 조사 제거
 - 띄어쓰기, 문장 분리 보정
 - 태깅
 - 악성댓글인지 아닌지 판별
 - 토큰나이징
 - 토큰의 단위를 확인, 자연어를 어떤 단위로 살펴볼 것인가
 - 어절 토큰나이징
 - 형태소 토큰나이징

- WordPiece tokenizing
- 모델 설계
- 모델 구현
- 성능 평가
- 완료

대소문자의 변환

함수	설명
upper()	모두 대문자로 변환
lower()	모두 소문자로 변환
capitalize()	문자열의 첫 문자를 대문자로 변환
title()	문자열에서 각 단어의 첫 문자를 대문자로 변환
swapcase()	대문자와 소문자를 서로 변환

편집, 치환

함수	설명
strip()	좌우 공백을 제거
rstrip()	오른쪽 공백을 제거
lstrip()	왼쪽 공백을 제거
replace(a, b)	a를 b로 치환

분리, 결합

함수	설명
split()	공백으로 분리
split('wt')	탭을 기준으로 분리
''.join(s)	리스트 s에 대하여 각 요소 사이에 공백을 두고 결합
lines.splitlines()	라인 단위로 분리

구성 문자열 판별

함수	설명
isdigit()	숫자 여부 판별
isalpha()	영어 알파벳 여부 판별
isalnum()	숫자 혹은 영어 알파벳 여부 판별
islower()	소문자 여부 판별
isupper()	대문자 여부 판별
isspace()	공백 문자 여부 판별
startswith('hi')	문자열이 hi로 시작하는지 여부 파악
endswith('hi')	문자열이 hi로 끝나는지 여부 파악

검색

함수	설명
count('hi')	문자열에서 hi가 출현한 빈도 리턴
find('hi')	문자열에서 hi가 처음으로 출현한 위치 리턴, 존재하지 않는 경우 -1
find('hi', 3)	문자열의 index에서 3번부터 hi가 출현한 위치 검색
rfind('hi')	문자열에서 오른쪽부터 검사하여 hi가 처음으로 출현한 위치 리턴, 존재하지 않는 경우 -1
index('hi')	find와 비슷한 기능을 하지만 존재하지 않는 경우 예외발생
rindex('hi')	rfind와 비슷한 기능을 하지만 존재하지 않는 경우 예외발생

한국어 토큰화

토큰화

- 주어진 데이터를 토큰이라 불리는 단위로 나누는 작업
- 토큰이 되는 기준은 다를 수 있음 (어절, 단어, 형태소, 음절, 자소)

문장 토큰화

- 문장 분리

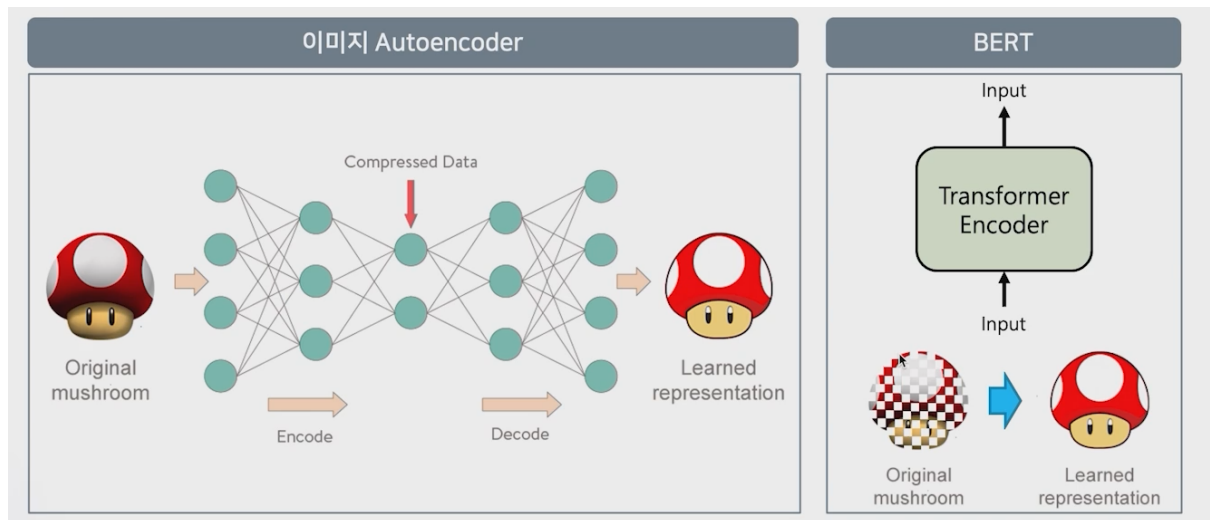
단어 토큰화

- 구두점 분리, 단어 분리

Bert 언어모델

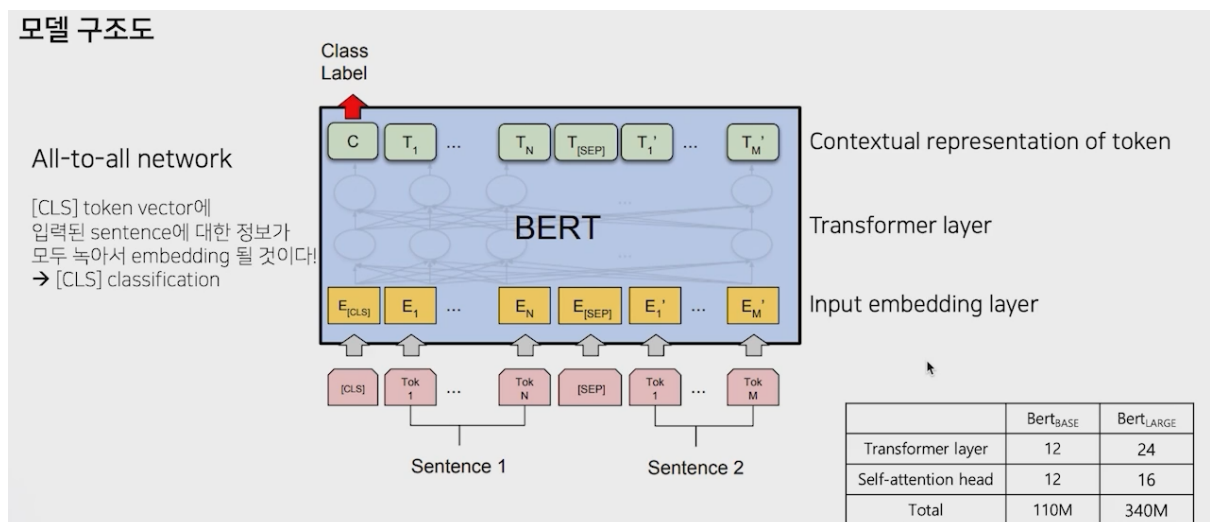
Bert 소개

전통적인 모델에서 발전한 모델이다.



BERT는 셀프-어텐션을 사용한 모델로서, 입력된 정보를 다시 입력된 정보로 돌리는 것을 목표로 합니다만, MASKED 라는 기술을 사용합니다. 중간중간 마스킹을 하여서 원본 이미지를 복원하기 어렵게 만들어둡니다. mask된 자연어를 원본 자연어로 학습하는 것을 목표로 움직이는 것입니다.

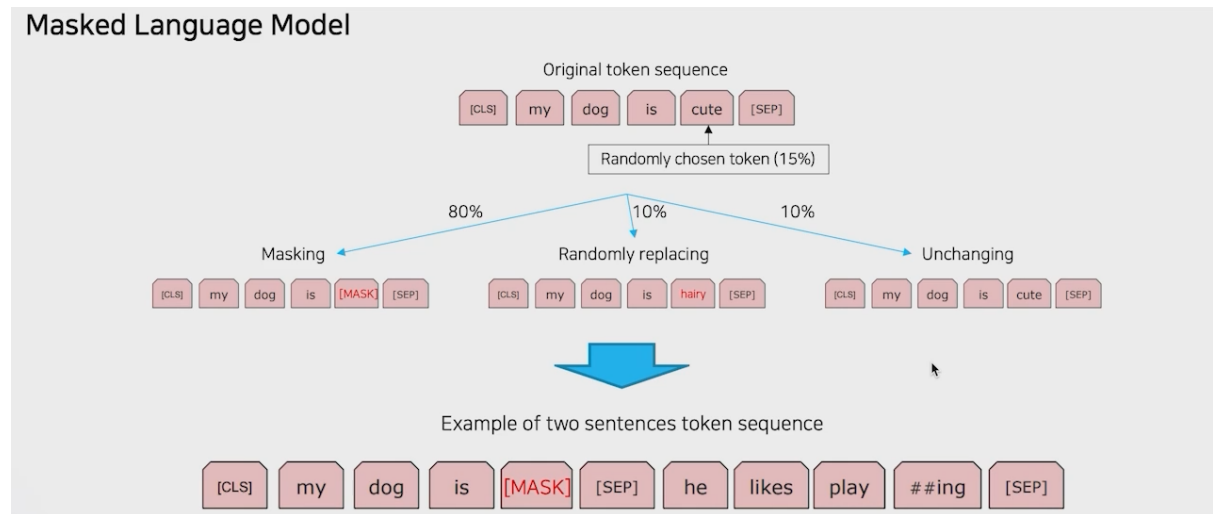
GPT-2 같은 경우엔 원본 이미지를 특정한 시퀀스로 잘라둡니다. 모델은 next를 알아내는 것을 목표로 합니다.



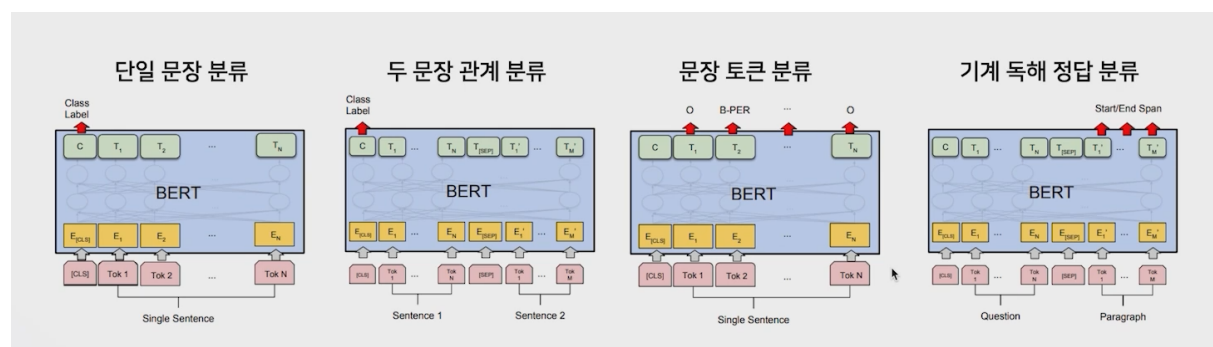
입력은 문장 2개를 부착한 채 들어오게 되지만, 12개 레이어가 all-to-all로 연결이 되어있고, bert 내부의 트랜스포머가 all-to-all로 연결이 되어있고,

cls 토큰 벡터에 입력된 문장의 정보가 모두 녹아든다고 가정을 한다고 한다면, 다만 조금 쉽지 않은 것 같습니다.

워드피스 토크나이징을 사용하지만, 빈도수 기반으로 토크나이징을 합니다. 다음 문장 또는 랜덤



수많은 태스크를 네가지 모델로 가능하다.



CLS 토크는 입력된 센텐스의 정보가 녹아있다고 분류할 수 있다만..

태스크 별로 인풋 센텐스만 바뀐다. 그저 토크가 다른 것 뿐이다.

버트 자체는 변하지 않는 다는 것이다.

Subject - 주어

Object - 목적어

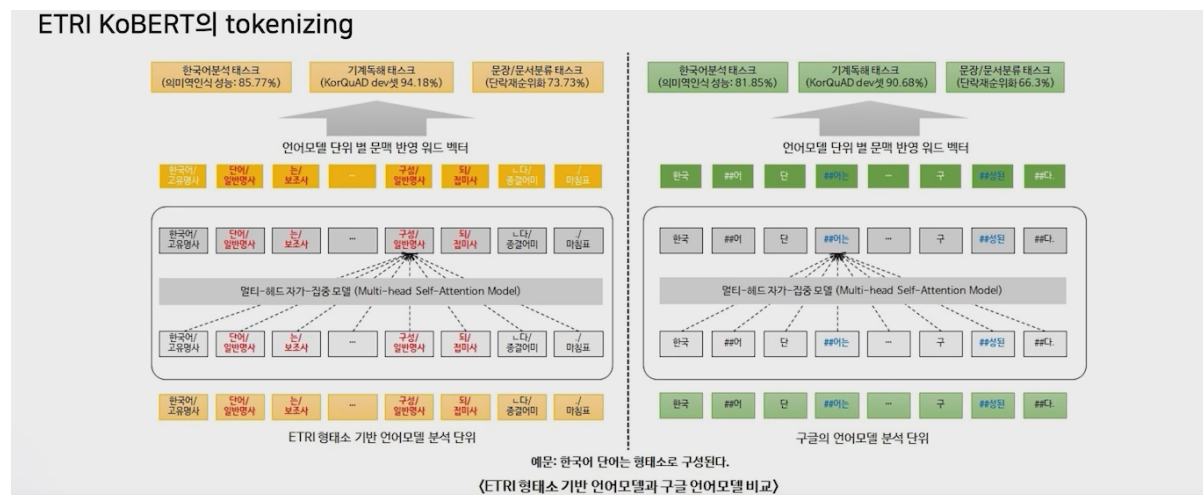
두 단어는 어떤 관계를 가지고 있는가를 확인하는 것입니다만...

개체명 분석, 기계 독해...!

KLUE : 성능 비교를 위한 데이터 셋

기계 독해 : 토큰나이징 한계 때문에 자주 문제가 발생한다만, 잘못 붙은 VOCAB으로 인해 점수가 낮을 수 밖에 없다고 한다.

음절단위 토큰나이징이 꽤나 괜찮은 것 같기도...? 이것을 시도해봐야 할 것 같다.



먼저 형태소 단위로 쪼개고, 이후 이를 워드피스 토큰라이저를 사용하여 진행한 듯 하다. → 이게 상당히 성능이 괜찮은 듯 하다. 이후에도 다른 모델에서도 자주 사용했다.

워드피스 토큰라이저만 사용한 경우에도 뭐 어느정도 성능이 나온다고 하는 것 같다.

전처리 시에 토큰을 조금 더 붙여서 나타내는 경우에도 성능이 오른다는 듯.

BERT Pre-Training

이미 있는거 쓰지, 왜 새로 학습할까?

도메인 특화 Task의 경우, 도메인 특화 된 학습 데이터만 사용하는 것이 성능이 더 좋다!

1. 토큰라이저 만들기

2. 데이터셋 확보

먼저, 문서단위로 잘 잘리고 있는지 알아야한다.

ex. 이순신의 마지막 문장이 문재인의 첫 문장과 이어져서는 학습이 되질 않는다.

1. 먼저 readline으로 문장을 읽고, 양쪽의 공백을 strip으로 날린다.
2. 이중 띄어쓰기가 발견되면, 나왔던 문장들을 모아 하나의 문서로 묶는다.
3. 즉, 문단 단위로 데이터를 저장한다.

3. Next sentence prediction

4. Masking

KLUE 데이터셋

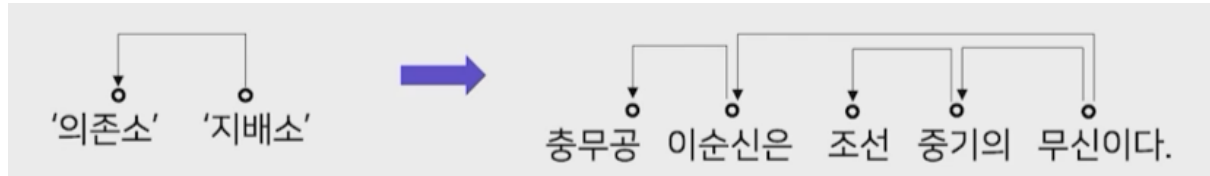
한국어 자연어 이해 벤치마크 (Korean Language Understanding Evaluation, KLUE)

살면서? 직면하게 되는 모든 자연어 TASK 유형!

- 문장 분류
- 관계 추출
- 문장 유사도
- 자연어 추론
- 개체명 인식
- 품사 태깅
- 질의 응답
- 목적형 대화
- 의존 구문 분석

의존 구문 분석

단어들 사이의 관계를 분석하는 task



1. 특징

- 지배소 : 의미의 중심이 되는 요소
- 의존소 : 지배소가 갖는 의미를 보완해주는 요소 (수식)
- 어순과 생략이 자유로운 한국어와 같은 언어에서 주로 연구된다.

2. 분류 규칙

- 지배소는 후위언어 이다. 즉 지배소는 항상 의존소보다 뒤에 위치한다.
- 각 의존소의 지배소는 하나이다.
- 교차 의존 구조는 없다.

3. 분류 방법

- Sequence labeling 방식으로 처리 단계를 나눈다.
- 앞 어절에 의존소가 없고 다음 어절이 지배소인 어절을 삭제하며 의존 관계를 만든다.

0단계	어절	충무공	이순신은	조선	중기의	무신이다.
	의존여부					

충무공 이순신은 조선 중기의 무신이다.

1단계	어절	충무공	이순신은	조선	중기의	무신이다.
	의존여부	의존	-	의존	의존	-

충무공 이순신은 조선 중기의 무신이다.

2단계	어절	충무공	이순신은	조선	중기의	무신이다.
	의존여부	의존	의존	의존	의존	-

충무공 이순신은 조선 중기의 무신이다.

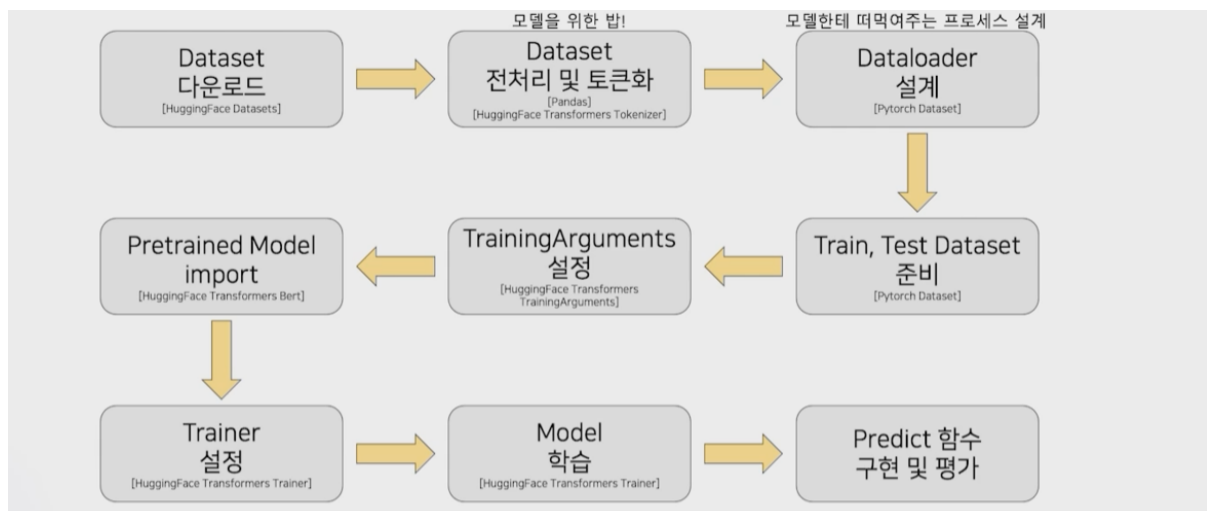
3단계	어절	충무공	이순신은	조선	중기의	무신이다.
	의존여부	의존	의존	의존	의존	-

충무공 이순신은 조선 중기의 무신이다.

복잡한 자연어 형태를 그래프로 구조화해서 표현이 가능하고, 이는 곧 각 대상에 대한 정보 추출을 가능케 한다.

- input_ids : sequence token을 입력
- attention_mask : [0,1]로 구성된 마스크이며 패딩 토큰을 구분
- token_type_ids : [0,1]로 구성되었으며 입력의 첫 문장과 두번째 문장 구분
- position_ids : 각 입력 시퀀스의 임베딩 인덱스
- inputs_embeds : input_ids대신 직접 임베딩 표현을 할당
- labels : loss 계산을 위한 레이블
- Next_sentence_label : 다음 문장 예측 loss 계산을 위한 레이블

- input_ids : 토큰의 vocab id
- attention_mask : 마스크나 패딩 되면 0, 안된 건 1로 처리.
- token_type_ids : 세그먼트 별 0, 1
- position_ids : 포지셔널 임베딩
- inputs_embeds
- labels
- Next_sentence_label



두 문장 관계 분류 태스크

1. NLI

- 언어 모델이 자연어의 맥락을 이해할 수 있는지 검증하는 TASK
- 전체 문장과 가설 문장을 함의, 모순, 중립으로 분류.

2. 두 문장 관계 분류 모델 학습

모델은 더욱 어려운 문제를 풀 때 더 효용성이 좋다. csv를 하나 따로 만들어보자.

3. 문장 토큰 관계 분류 task

NER

개체명 인식은 문맥을 파악해서 인명, 기관명, 지명과 같은 문장 또는 문서에서 특정한 의미를 가지고 있는 단어, 또는 어구 등을 인식하는 과정을 의미한다.

```
[20] texts = ["이수지가 저수지에 갔는데 이 수지가 저수지에 간 걸까 저 수지가 저수지에 간걸까 그 수지가 저수지에 간 걸까 하며",
             "이수지는 고민했는데 고민 끝에 이수지의 마이웨이를 부르며 불쾌지수가 올라가며 저수지를 떠나 경기도 수지구의 한 학원으로 달려가더니",
             "저수함을 배워서 잘 사용하여 주식 수지를 맞아 \나 이수지, 바로 고단수지! 수지맞았다!\\"하며 행복해했다."]

[21] from pororo import Pororo
     ner = Pororo(task="ner", lang="ko")

[22] for text in texts:
     for pred in ner(text):
         if pred[1] != "O":
             print("{}({})".format(pred[0], pred[1]), end="")
         else:
             print(pred[0], end="")
     print()

이수지(PERSON)가 저수지에 갔는데 이 수지가 저수지에 간 걸까 저 수지가 저수지에 간걸까 그 수지가 저수지에 간 걸까 하며
이수지(PERSON)는 고민했는데 고민 끝에 이수지(PERSON)의 마이웨이를 부르며 불쾌지수가 올라가며 저수지를 떠나 경기도(LOCATION) 수지구(LOCATION)의 한 학원으로 달려가더니
저수함을 배워서 잘 사용하여 주식 수지를 맞아 \나 이수지(PERSON), 바로 고단수지! 수지맞았다!\\"하며 행복해했다.
```

같은 단어라도 문맥에서 다양한 개체로 사용됨

pororo를 사용해 태깅하기도 좋을 것 같다.

태깅 관련 모듈로서는 굉장히 흡족할 것 같습니다.

kor_ner

- 한국해양대학교 자연어 처리 연구실에서 공개한 한국어 NER 데이터셋
- 일반적으로, NER 데이터셋은 pos tagging 도 함께 존재

```
## 1
## 스포츠동아 김민정 기자 ricky337@donga.com
## <스포츠동아:ORG> <김민정:PER> 기자 <ricky337@donga.com:POH>
스포츠      스포츠      NNG      B-ORG
동아      동아      NNP      I-ORG
          O
김민정      김민정      NNP      B-PER
          O
기자      기자      NNG      O
          O
ricky      ricky      SL      B-POH
337      337      SN      I-POH
@      @      SW      I-POH
donga      donga      SL      I-POH
.      .      SF      I-POH
com      com      SL      I-POH
```

해당 데이터셋에서의 문장 번호

문장 원본

문장에서 Entity가 표시된 문장

형태소 토큰	형태소 토큰	POS 태그	Entity 태그
...

학습은 역시 각각의 토큰에 Classification layer를 부착하는 과정으로 이루어진다.

한국어는 음절단위로 잘라내는 것을 추천드린다고 하십니다.