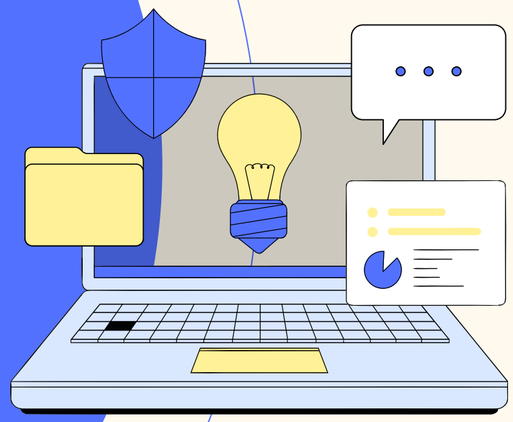
KOREA IT ACADEMY

WordPiece Tokenizer

WORDPIECE TOKENIZER
이론 설명 및 논문 분석 발표

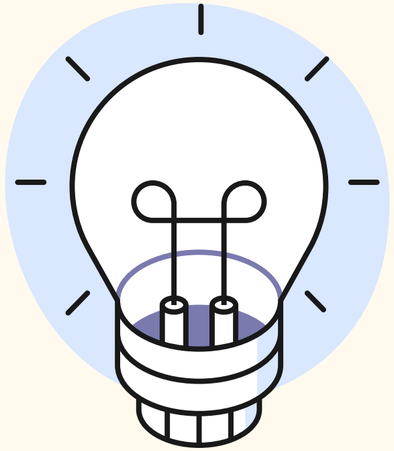
홍준표 김동휘 양문기 유아람 이정흔

KOREA IT ACADEMY



목차

워드피스 모델



WordPiece Model 이란?

WordPiece Model(WPM)은 구글에서 주로 기계 번역 및 음성 인식과 같은 자연어 처리(Natural Language Processing, NLP) 태스크에서 언어 모델의 성능을 향상시키기 위해 도입 되었으며. Byte Pair Encoder(BPE)의 변형 알고리즘으로, 주어진 텍스트 데이터에서 학습된 언어 모델의 우도(Likelihood)를 최대화하는 방식으로 토큰을 생성하는 알고리즘을 이용하는 모델입니다.

WPM은 텍스트 내에서 우도가 가장 높은 쌍을 반복적으로 찾아내어 이를 병합하여 어휘를 동적으로 확장하며, 이러한 방식은 기계 번역 및 음성 인식과 같은 자연어 처리에서 OOV(Out-Of-Vocabulary) 문제를 해결하는데 도움을 준다.

워드피스 모델

WordPiece Model(WPM)의 디코딩 개선과정

초기의 WPM은 길고 희귀한 쿼리에서 발생하는 공백 해결 문제를 처리하기 어려웠습니다. Google은 이에 대한 해결책으로 디코딩 중에 해당 문제를 처리할 수 있도록 WPM을 수정하였습니다.

입력 데이터 처리

원본 언어 모델 데이터에는 입력된 값 그대로 사용되며, WPM으로 LM 데이터를 분할할 때 공백이 보이면 각 단위의 앞뒤에 공백 마커(_, ~)를 붙이고, 그렇지 않으면 붙이지 않는다.

LM과 사전 재구축

위에서 정리한 인벤토리를 기반으로 언어 모델(LM)과 사전을 다시 구축한다.

디코딩 시 공백 삽입 위치 보존

디코딩 시 모델에 따라 최적의 경로가 선택되어 공백을 넣을 위치와 넣지 않을 위치가 보존된다.

BPE

BPE란?

데이터 압축 알고리즘

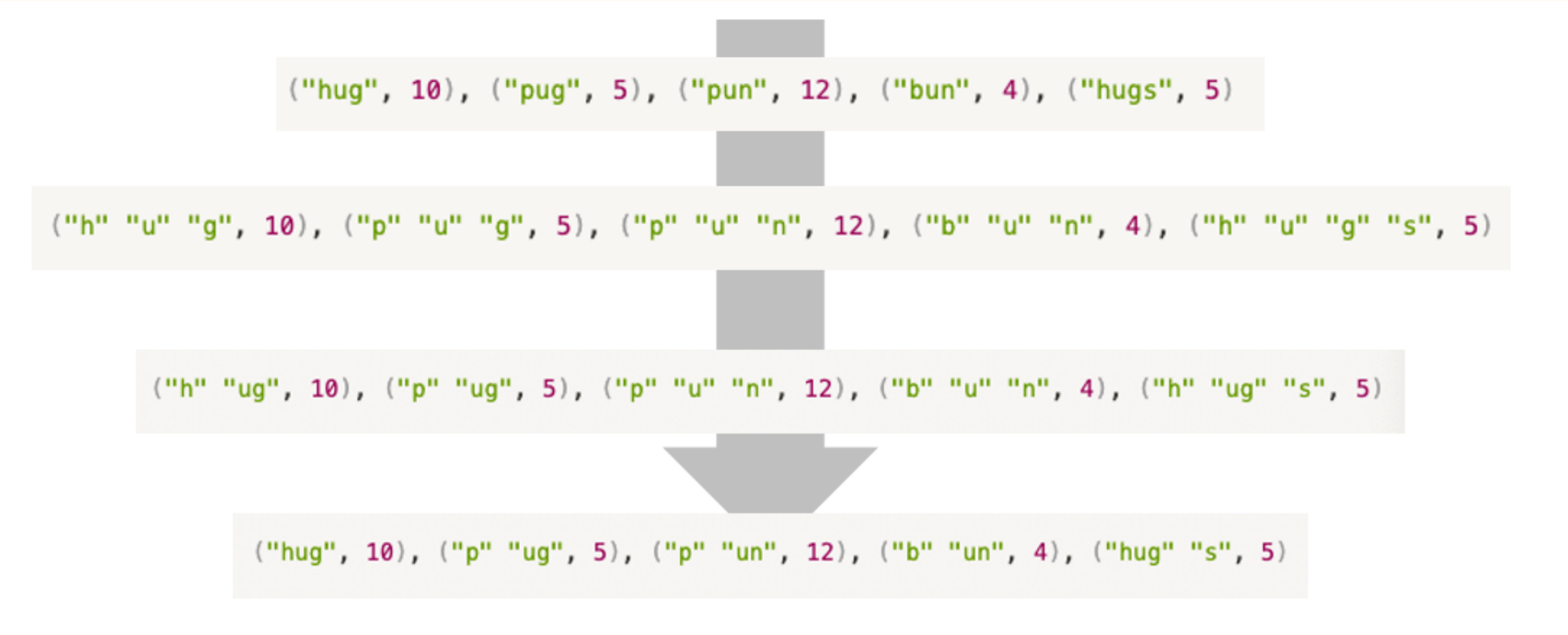
연속된 시퀀스에서 가장 빈번하게 등장하는 두 **바이트 쌍**을 묶어서 하나의 토큰으로 압축하는 기술

NMT(인공신경망기계번역)의 SUBWORD TOKENIZER

OOV를 해결하기 위해 BPE를 서브워드 토크나이저로서 사용

BPE 토크나이저

1. 각 단어를 바이트 단위로 나눔
2. 빈도가 높은 바이트쌍을 조합하여 하나의 토큰으로 생성
3. 이 과정을 원하는 만큼 반복



BPE 한계점

희귀하거나 특수한 패턴에 대한 대응이 어렵다.

WPT는 훈련 데이터의 패턴을 반영하여 동적으로 어휘를 생성함.

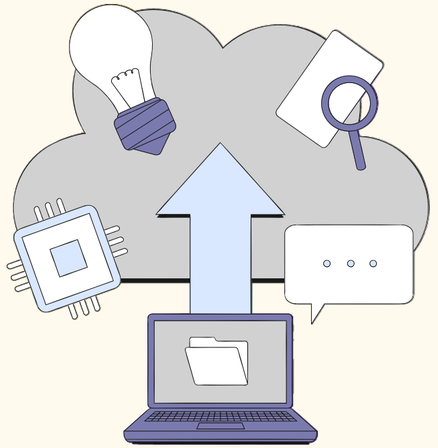
한국어의 경우 자음 모음 단위로 서브워드가 나눠지면 모델이 복잡해짐.

한국어 단어는 보통 높은 음절 수를 가진다. WPT는 음절 수가 많은 단어도 의미적인 부분으로 나누어 처리하면서, 모델이 장문의 단어에 대해 더 효과적으로 학습할 수 있다.

이외에도 다의어성 문제, 단어경계 문제가 있음.

결국 더 좋은 알고리즘이 필요

Wordpiece Tokenizer (1)



번역 모델의 최선 학습을 위한 데이터 전처리 알고리즘

- 일본어, 한국어 음성 검색 모델을 완성시키기 위해 구글에서 처음 개발한 알고리즘 [JAPANESE AND KOREAN VOICE SEARCH - Google Inc., USA]
- 모델의 Input 값을 조정하여 (데이터 전처리) 성공적인 모델의 학습을 겨냥

〈1〉

초기화

- 단어 단위 인벤토리를 기본 유니코드 및 ASCII 코드로 초기화
- 디코딩 시의 편의를 위해 공백에 “_” 삽입

〈2〉

모델 학습

- 주어진 단어 단위 인벤토리를 학습 데이터로 사용하여 학습

〈3〉

인벤토리 수정

- 두 개의 단위를 결합하여 인벤토리를 하나씩 증가
- 우도를 가장 증가시키는 쌍을 찾아 결합
- 한계에 도달하거나 우도 증가율이 특정 임계값 아래로 떨어질 때 까지 반복

Wordpiece Tokenizer (1)

Wordpiece Tokenzier 의 훈련 세부과정

단어 유닛 목록을 ASCII, 22,000개 일본어 11,000개 한국어를 유니코드 형태로 초기화

전세계의 모든 문자를 포함할 수 있는 Unicode 형식으로 단어 유닛 목록을 초기화

위 과정을 통해 만들어진 유닛 목록을 사용하여 학습 데이터에 대해 LANGUAGE MODEL 구축

초기화된 단어 유닛 목록을 사용하여 훈련 데이터에 대한 언어 모델을 구축합니다. 여기서 언어 모델로는 N-gram 알고리즘 중에서 3~5-grams을 사용하며, Katz back-off language model을 활용

모델에 결합한 2개의 단어 유닛을 추가했을 때 LIKELIHOOD가 높은 PAIR를 새로운 단어 유닛으로 생성

특정한 단어 단위의 한계에 도달하거나, likelihood의 증가량이 특정 임계점보다 낮아질 때까지 생성된 단어 유닛을 계속해서 추가하고 언어 모델을 업데이트하여 훈련을 반복

미리 결정한 단어 단위의 한계에 도달하거나 LIKELIHOOD의 증가량이 특정 임계점보다 떨어질 때까지 해당 과정을 반복

특정한 단어 단위의 한계에 도달하거나, likelihood의 증가량이 특정 임계점보다 낮아질 때까지 생성된 단어 유닛을 계속해서 추가하고 언어 모델을 업데이트하여 훈련을 반복

Wordpiece Tokenizer (2)

공백 디코딩 시나리오

01. 모든 공백 제거

" " → ""

예제

입력: "Hello World"
출력: "HelloWorld"

02. 이중 공백 마커를 공백으로 변경

"__" → " "

예제

입력: "OpenAI__GPT-3.5"
출력: "OpenAI GPT-3.5"

03. 남은 단일 공백 마커 제거

"_" → ""

예제

입력: "Data_Science_is_amazing"
출력: "DataScienceisamazing"

이러한 과정을 통해 WPM의 디코딩 과정이 최적화되어, 다양한 형태의 공백이 효과적으로 처리되고 최종 디코딩 결과가 보다 정교해지게 된다.

Wordpiece Tokenizer (2)

예시

아래 코퍼스에서 makers, over 는 모두 자주 이용되기 때문에 그 자체를 units 으로 이용하지만, Jet 은 자주 등장하지 않는 단어이기 때문에 subword units 인 ‘J’ 와 ‘et’ 으로 나눈다.

Word : Jet makers feud over seat width with big orders at stake
Wordpieces: _J et __makers __fe ud __over __seat __width __with __big __orders __at __stake

Underbar("_") 는 문장 생성, 혹은 subwords 부터의 문장 복원을 위한 특수기호 입니다. Underbar 없이 subwords 를 띄어두면, 본래 띄어쓰기와 구분이 되지 않기 때문에 혼란을 줄 수 있어 Underbar를 이용하여 띄어쓰기를 구분한다.

문장을 복원하는 코드는 간단합니다. 띄어쓰기 기준으로 나눠진 tokens 을 concatenation 한 뒤, _로 다시 나눠 tokenize 하거나 _ 를 빈 칸으로 치환하여 문장으로 복원합니다.

```
def recover(tokens):
    sent = ''.join(tokens)
    sent = sent.replace('_', ' ')
    return sent
```

Wordpiece Tokenizer (3)

IMPORTANT POINT

전처리 이후 개발자가 원하는 Token의 개수를 가진 가장 작은 Size의 Corpus 생성

Greedy Algorithm 을 통하여 매 선택 가장 높은 가중치를 지닌 쌍을 생성

때문에 한 번의 알고리즘 수행에서 가장 많은 단어를 설명할 수 있는 Token Pair가 생성

$$score = \frac{freq_{pair}}{freq_{first} \times freq_{second}}$$

GREEDY ALGORITHM

문제를 해결하는 데 있어서 매 순간마다 가장 최적인 선택을 하는 방식의 알고리즘

Greedy Algorithm 은 전역 최적해(global optimum)를 찾는 것이 목표가 아니라, 각 단계에서의 로컬 최적해(local optimum)를 찾는 방법

Likelihood

확률 분포의 모수가, 어떤 확률변수의 표집값과 일관되는 정도를 나타내는 값

식 1

$$\mathcal{O}_{\mathrm{ML}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \log P_\theta(Y^{*(i)} \mid X^{(i)}) \ .$$

모델의 파라미터를 나타내는 θ에 대해 최대우도 추정(maximum-likelihood estimation)을 수행하는 손실 함수로 모델이 주어진 입력에 대해 정답을 생성할 확률을 계산해 모델이 실제로 관측된 정답 시퀀스에 가까운 출력을 생성하도록 하는 것을 목표

문제점 디코딩 중에 발생하는 오류에 대해 학습하지 못해 훈련 및 테스트 절차 간 상당한 불일치가 있을 수 있는데 식1은 그것을 반영하지 않는다

식 2

$$\mathcal{O}_{\mathrm{RL}}(\boldsymbol{\theta}) = \sum_{i=1}^{N} \sum_{Y \in \mathcal{Y}} P_\theta(Y \mid X^{(i)}) \ r(Y, Y^{*(i)}).$$

출력 시퀀스 Y와 실제 출력 시퀀스Y* 사이의 관계를 나타내는 보상 함수를 추가하여 모델이 학습 중에 출력 시퀀스를 예측할 때 디코딩 중에 발생하는 오류를 고려하여 학습하는 방법을 사용하여 모델의 성능을 측정하는 방법을 사용

$r(Y, Y^{*(i)})$ 모델이 예측한 출력과 실제 출력 간의 일치 정도를 평가하며, 일치 정도가 높을수록 높은 보상을 부여

해결방안

학습을 안정화하기 위해 식1과 식2를 활용하여 계산해 선형조합한 공식을 사용하여 해당 공식이 최대가 되는 방법으로 모델을 개선

α : 0.017으로 설정

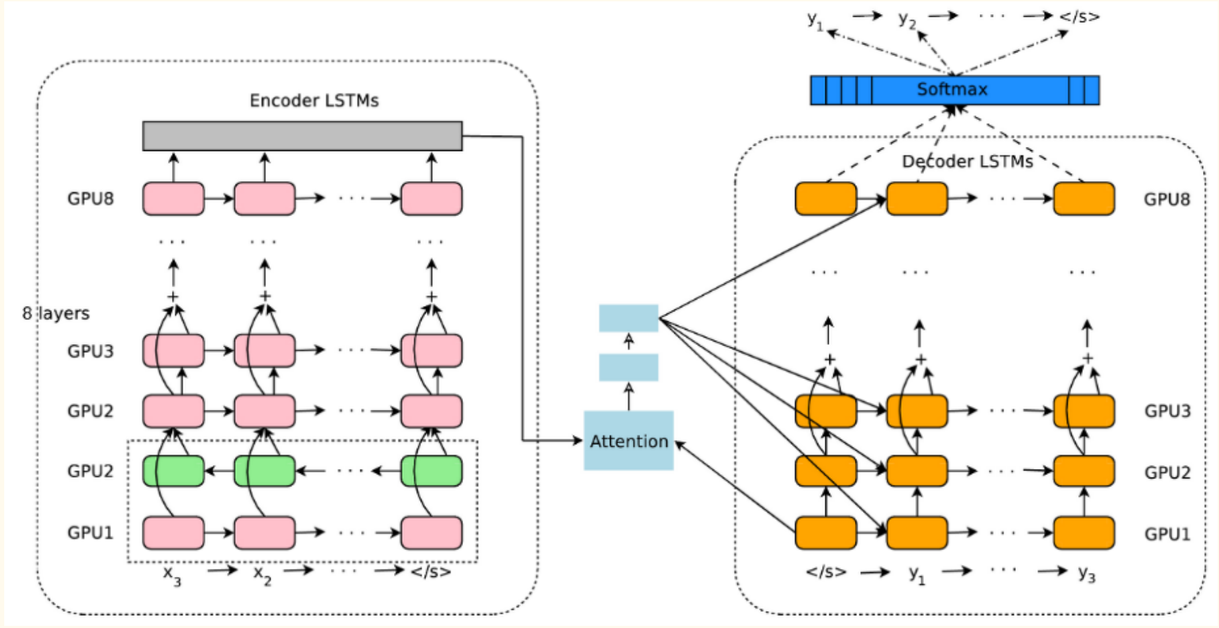
최종 Likelihood 식

$$\mathcal{O}_{\mathrm{Mixed}}(\boldsymbol{\theta}) = \alpha * \mathcal{O}_{\mathrm{ML}}(\boldsymbol{\theta}) + \mathcal{O}_{\mathrm{RL}}(\boldsymbol{\theta})$$

Wordpiece Tokenizer in GMNT

변형 및 차이점

단어의 시작 부분에만 특수 기호를 사용하고 양쪽 끝에는 사용하지 않음
데이터에 따라 기본 문자 수를 제한하고 나머지는 알 수 없는 특수 문자로 mapping 하여 희귀 단어에 의한 오염을 방지



예제

Google Colab 에서 구현한
Wordpiece Tokenizer 간단 예제

https://colab.research.google.com/drive/1bVmgniEFEEszURbUcdFmh62AX0jLsBzh

- 적용
- 예시 문장

```
ex_sentence = "bugs hg hn puugs bun"
WPTed(ex_sentence) = ["bu", "##gs", "h", "##g", "h", "##n", "p", "##u", "##
```

경청해 주셔서 감사합니다

KOREA IT ACADEMY

PRODUCED BY 2-JO
GUNG GUEM HAN JUM ??
MULL EO BWA YO !!