LLM

Tokenizer & Embedding

DeepShark Lab

학부연구생 박정규

목차 안내

LLM Architecture Overview

LLM의 Arcitecture 요약

Input Text

사용자가 자연어를 입력
모델은 이 문장을 그대로 계산할 수 없다

Tokenizer

입력 문장을 작은 단위인 Token으로 분할

Token IDs

분할된 Token은 Tokenizer의
vocabulary에 따라 고유한 숫자 ID로 전환

LLM Architecture Overview

LLM의 Arcitecture 요약

Embedding Layer

Token ID는 다시 Embedding vector로 변환

Transformer Blocks

Transformer block에서 Token 벡터들을 처리하며 토큰 간의 관계를 학습

Output Probability

Transformer를 지나고 다음 토큰이 무엇일지에 대한 확률로 변환 후 확률이 가장 높은 Token이 출력으로 선택

Generated Text

선택된 Token의 ID는 사람이 읽을 수 있는 자연어로 변환

Tokenizer

자연어 문장을 LLM이 처리할 수 있는 형태로 변환
GPT -2 Tokenizer (BPE) 사용한 예시

Before

자연어 문장

```
Original Text:
We are DeepShark Family
```

▶

After

Token ID

```
Tokens:
['We', 'Ġare', 'ĠDeep', 'Sh', 'ark', 'ĠFamily']

Token IDs:
[1135, 389, 10766, 2484, 668, 7884]

Token ↔ Token ID:
We              -> 1135
Ġare            -> 389
ĠDeep           -> 10766
Sh              -> 2484
ark             -> 668
ĠFamily         -> 7884
```

Tokenizer

What is token?

1. Word
2. Subword
3. Character
4. symbol

Token은 문장을 나눈 조각

단어, 단어의 일부, 글자, 특수문자등 기준에
따라 문장이 Token으로 분할 된다

LLM은 문장 전체를 계산하지 않고,
이 Token들을 이용하여 문장을 이해한다

Tokenizer

Word-Level Tokenization 의 한계

01
Voca Size

형태변화가 많은 언어를 모두 다른 Token으로 저장하면 사전의 사이즈가 너무 커진다

02
Out of Voca

Tokenizer가 보지 못한 단어가 많아질 수 있다

03
Rare word

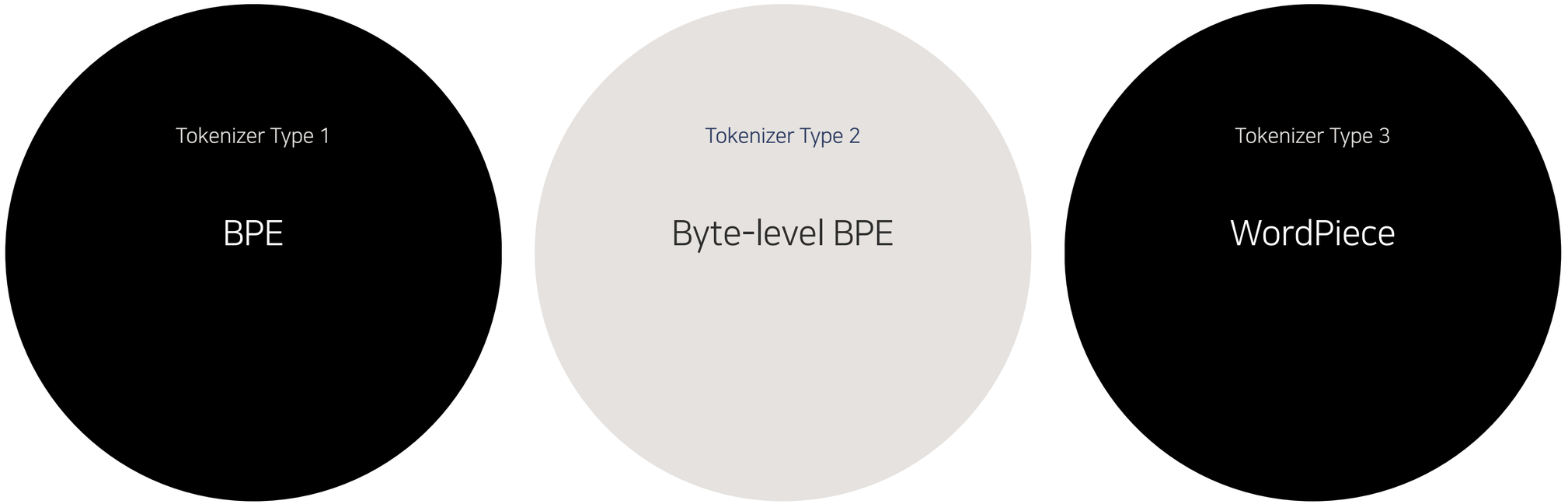
처음보는 단어를 처리하기 어려워 신조어, 고유명사, 도메인 전문용어에 취약하다

04
cost

사전의 Size가 커지면서 embedding parameter 수가 크게 증가하여 메모리 사용량과 계산 비용이 커진다

Tokenizer

대표적인 Tokenizer



Tokenizer

Byte Pair Encoding

Step 1

모든 단어를 character 단위로 나눈다
player → p l a y e r
played → p l a y e d
playing → p l a y i n g

Step 2

자주 등장하는 쌍을 찾는다
p + l
l + a
a + y

Step 3

자주 등장하는 쌍을 하나의 새 토큰으로 합친다
p + l → pl
pl + a → pla
pla + y → play

Step 4

원하는 vocabulary 크기가 될때까지 반복 후 저장
playing → play + ing
played → play + ed
player → play + er

player, played, playing을 각각 다른 단어로 저장하지 않고, 조각으로 저장하여 모델의 이해도를 효과적으로 높일 수 있다

희귀한 단어나, 신조어등 모델이 처음보는 단어도 subword로 쪼개어 이해할 수 있다

BPE 사용 모델 : GPT-3, Qwen 계열 모델 등

Tokenizer

Byte-level Byte Pair Encoding

Step 1

Raw Text를 UTF-8 byte 단위로 변환한다

Step 2

byte 쌍 중 자주 등장하는 쌍을 찾는다

Step 3

자주 등장하는 쌍을 하나의 토큰으로 합친다

Step 4

원하는 vocabulary 크기가 될때까지 반복 후 저장

BBPE는 byte 단위로 시작하기 때문에 거의 모든 문자를 표현할 수 있다

영어 뿐만 아니라, 한자, 이모지, 특수문자와 같은 다양한 문자를 처리할 수 있다

BBPE 사용 모델 : GPT-2, GPT 계열 모델, DeepSeek-V2, DeepSeek-V3

Tokenizer

WordPiece

Step 1

모든 단어를 작은 단위로 나눈다
play → p l a y
player → p l a y e r
played → p l a y e d
playing → p l a y i n g

Step 2

자주 등장하거나 단어를 잘 설명할 수 있는 조각을 후보로 둔다
play
##er
##ed
##ing

Step 3

단어를 잘 표현할 수 있는 subword를 vocabulary에 저장
play, ##er, ##ed, ##ing

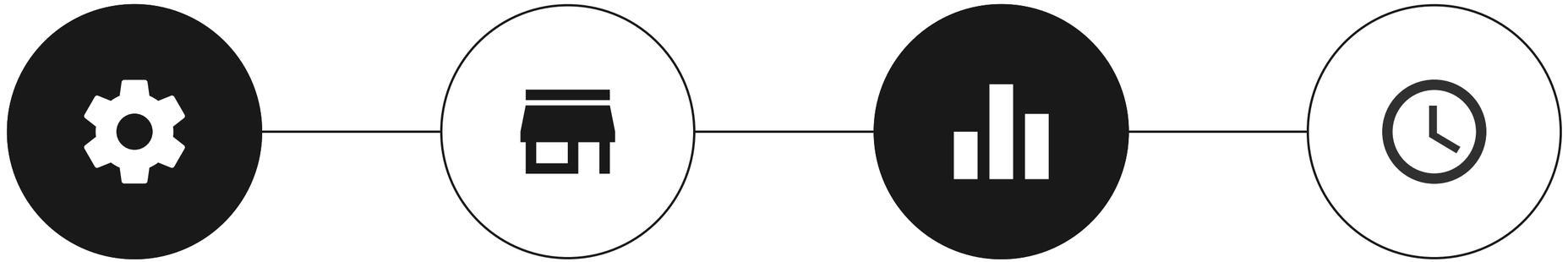
BPE와 마찬가지로 Voca의 사이즈를 줄일 수 있고, 처음보는 단어에 대해서도 처리할 수 있다.

단어의 형태 변화를 처리하기 좋다

WordPiece 사용 모델 : BERT 계열 모델

Embedding Layer

Token의 ID가 비슷해도 의미적으로 가깝다는 의미는 아니기 때문에, ID를 이용하여 벡터공간에 단어들을 학습시켜 비슷한 의미의 단어는 비슷한 방향에 위치시키도록 한다.



Tokenization

문장을 Token으로 나눈다

Token ID Mapping

나눈 Token들을 ID로 변환한다

Embedding Lookup

Embedding Layer에서 ID를 이용하여 벡터로 변환한다

Transformer Input

변환된 벡터들을 Transformer에 입력한다

Embedding Layer

1	Token Embedding	Token ID를 dense vector로 변환
2	Positional Embedding	토큰의 순서 정보를 벡터로 추가
3	Contextual Embedding	문맥에 따라 같은 토큰도 다른 벡터를 가질 수 있음

Embedding Layer

Token Embedding

Token Embedding

- Token Embedding은 이산적인 Token ID를 연속적인 dense vector로 변환한다.
- 각 토큰 ID는 하나의 학습 가능한 Embedding vector와 연결된다
- 이 벡터들이 Transformer 모델의 실제 입력으로 사용된다

Example

- Text : We are Deepshark
- Tokenizer : [“We”, “are”, “Deep”, “Shark”]
- Token ID : [1023, 389, 8214, 23091]
- Token Embedding

1023 → [0.12, -0.31, 0.08, ...]
389 → [-0.44, 0.27, 0.91, ...]
8214 → [0.63, -0.10, -0.52, ...]
23091 → [-0.19, 0.78, 0.35, ...]

Embedding Layer

Positional Embedding

Positional Embedding

- Positional Embedding은 토큰의 순서 정보를 벡터로 추가한다
- Transformer는 입력을 병렬로 처리하기 때문에 위치 정보가 필요하다
- 최종 입력 벡터는 Token Embedding + Positional Embedding 형태이다

Example

- Text : We are Deepshark
- Tokenizer : [“We”, “are”, “Deep”, “Shark”]
- Token ID : [1023, 389, 8214, 23091]
- Token Embedding : $[E_0, E_1, E_2, E_3]$
- Positional Embedding : $[P_0, P_1, P_2, P_3]$
- Final input : $[E_0+P_0, E_1+P_1, E_2+P_2, E_3+P_3]$

Embedding Layer

Contextual Embedding

Contextual Embedding

- 같은 토큰도 문맥에 따라 다른 의미를 가질 수 있기 때문에 사용한다
- 주변 문맥을 반영한 토큰 벡터 표현이다
- Token Embedding과 Positional Embedding을 거친 후 Transformer에서 Self-Attention을 통해 주변 토큰의 정보를 반영한다. 이과정에서 문맥이 반영된 Embedding이 만들어진다.

Example

- Text : We are Deepshark
- Tokenizer : [“We”, “are”, “Deep”, “Shark”]
- Token ID : [1023, 389, 8214, 23091]
- Token Embedding : [E_0, E_1, E_2, E_3]
- Positional Embedding : [P_0, P_1, P_2, P_3]
- Self-Attention
- Contextual Embedding

Tokenizer & Embedding

발표를 마칩니다

DeepShark Lab

학부연구생 박정규