Tokenizer

SentencePiece & BPE (Byte Pair Encoding)

Tokenizer란?

정의: 텍스트를 모델이 처리할 수 있는 토큰(token) 시퀀스로 변환해, 언어와 모델을 연결하는 번역기

1) 분해: 텍스트 → 토큰
2) 매핑: 토큰 → 정수 ID
3) 복원: 정수 ID → 텍스트

Tokenizer 필요성

- 희소성 완화: 단어 수준 → 서브워드로 분해해 OOV (Out of Vocabulary) 거의 제거
- 길이/메모리 절감: 자주 등장 서브워드에 짧은 ID 부여 → 효율증가
- 학습 안정성: 일관된 분해/결합 규칙 → 모델 입력 분포 안정
- 디코딩 안정성: 재결합 가능한 토큰 설계(공백·Unicode 처리) 필수

Tokenizer 종류

1) 문자 단위(Character): 한 글자씩. OOV (Out of Vocabulary) 없으나 문장 시퀀스 길어짐.
2) 바이트 단위(Byte/Byte-level): 어떤 문자이든 1–4바이트로 안전하게 처리(이모지 포함).
3) Subword (+byte fallback 현대 LLM 표준): BPE (Byte Pair Encoding), WordPiece, Unigram LM
4) 단어/공백 기반(Whitespace/Rule-based): 공백·문장부호로 분할. 간단하지만 OOV 잦음.
5) 형태소 기반(Morphological): 한국어/일본어 등 교착어 세밀 처리

BPE (Byte Pair Encoding)

BPE (Byte Pair Encoding)

핵심 아이디어

- 코퍼스에서 가장 자주 붙어 나오는 인접 쌍을 반복적으로 병합해 서브워드 사전을 만든다.
- 병합 횟수 ≈ vocab_size 증가

학습 루프(Training) 개요

1) 초기 토큰: 문자/바이트 단위(공백은 _로 표시)
2) 모든 인접 쌍 빈도를 센다
3) 최다 빈도 쌍 (X,Y) 선택 → 새 심볼 “XY” 추가, 규칙 X Y → XY 기록
4) 코퍼스 전역에서 모든 X Y를 동시에 XY로 치환
5) 쌍 빈도 재계산, 목표 병합 횟수까지 반복

산출물

- 어휘(초기 문자/바이트 + 병합으로 생긴 서브워드 + 스페셜)
- 규칙 리스트(merge ops): 예) w e → we, we s → wes, …

BPE 병합 예제

- 코퍼스: _lowest, _lower, _newest, _wider

Step 0) 문자 단위

- _ l o w e s t / _ l o w e r / _ n e w e s t / _ w i d e r
- 빈도 상위: (w,e)=3, (l,o)=2, (o,w)=2, (e,s)=2, (s,t)=2, (e,r)=2

Step 1) (w, e) → we 추가, 전역 치환

- 이후 (e,s)는 사라짐(we 내부의 e는 독립 토큰 아님)
- 새 후보 예: (l,o)=2, (o,we)=2, (we,s)=2, (s,t)=2

Step 2) (we,s) → wes

Step 3) (wes,t) → west

Step 4) (l,o) → lo

Step 5) (lo,west) → lowest

Step 6) (we,r) → wer → (lo,wer) → lower

Encoding

학습 때 만들어 둔 병합 규칙(merge ops) 을 우선순위 순서대로 적용하면서, 가능한 한 긴 서브워드가 되도록 탐욕적으로 병합

- 빈도 재집계 없음(훈련에서 이미 끝남): 정해진 규칙만 적용
- 입력단위는 문장(하나의 문자열) 전체: 공백은 특수 기호 _ 로 표시해 단어 경계 보존하기 때문에 인코딩은 문장 단위로 돌지만 실질적으로는 단어별로 처리
- 예시 입력: `lowest` → `_ l o w e s t` 로 나누고, 학습시에 만들어진 병합 규칙을 greedy하게 적용, 더 이상 적용할 규칙이 없으면 중단하고 최종 토큰 시퀀스를 출력

① 입력은 문장(문자열), 공백은 _로 처리 → 단어 경계가 보존됨.
② 인코딩은 훈련 때의 병합 규칙 순서를 따라 탐욕적으로 병합.
③ 하나의 인코딩 패스에서 빈도를 새로 세지 않음(그건 훈련 때 이미 끝남).
④ _ 덕분에 토큰이 보통 단어 시작에서 시작하고, 디코딩 시 가역성 보장

Hyperparameter

- vocab_size: 8k–50k(모델/속도/메모리 트레이드오프)
- 초기 알파벳: 문자 또는 바이트 레벨(다국어/이모지 강건)
- 정규화: NFKC(nmt_nfkc) 권장, 한국어는 character_coverage≈0.9995
- byte_fallback=true + hard_vocab_limit=false(바이트 토큰 256개)

- LLM 미세조정·서빙: 32k(경량) 또는 50k(조금 여유).
- 다국어 LLM 사전학습: 100k 전후(언어 다양성↑), 단 토큰 효율/속도 trade-off 확인.
- n-gram 실험: 8k~16k부터 시작(32k는 비용이 급증).

Sentencepiece

2

SentencePiece: 구글 오픈소스 서브워드 토크나이저

- 2018~: SentencePiece 공개 및 논문 발표(구글 번역 등에서 서브워드 대중화)
- BPE & Unigram 동시 지원으로 연구·산업 표준 도구로 자리잡음
  - BPE: 단순·빠름·결정적, GPT류 등 광범위 채택
  - Unigram: 확률 모델 기반, 분절 유연·잡음 강건(T5 등에서 채택)
  - 이후 LLM 대중화와 함께 다국어·대규모 코퍼스에서 사실상 기본 선택지
- 특징
  - OOV최소화, 다국어/이모지 대응, 디코딩 가역성 높음
  - 토큰 고효율 → 학습/추론 저비용
- 핵심 옵션
  - character_coverage(CJK 권장 0.9995~0.9999)
  - normalization_rule_name(예: nmt_nfkc)
  - byte_fallback=true(UNK 방지, 바이트 안전망)
  - hard_vocab_limit=false(256 바이트 토큰 등 여유)

학습 – Python API

- 설치: pip install sentencepiece
- 데이터 준비
  - UTF-8, 중복 제거/클린징, NFKC 정규화 고려
  - 대용량은 --input_sentence_size + 셔플 샘플링
- BPE 예시(32k, byte_fallback 포함)

```
import sentencepiece as spm
args = [
  '--input=corpus.txt',
  '--model_prefix=spm_bpe32k',
  '--vocab_size=32000',
  '--model_type=bpe',
  '--normalization_rule_name=nmt_nfkc',
  '--character_coverage=0.9995',
  '--byte_fallback=true',          # UNK 대신 바이트 토큰으로 안전 처리
  '--hard_vocab_limit=false',      # 바이트 토큰 256개 추가 여지
'--input_sentence_size=2000000',   # 문자 수 상한, 0이면 전체 corpus의미
  '--shuffle_input_sentence=true'  # 무작위로 문장 수만큼 비중복 샘플링
]
spm.SentencePieceTrainer.Train(' '.join(args))
# 산출물: spm_bpe32k.model, spm_bpe32k.vocab
```

인코딩/디코딩(Encoding/Decoding)

```
import sentencepiece as spm
sp = spm.SentencePieceProcessor(model_file='spm_bpe32k.model')


text = "안녕하세요🙂 전각Ａ와 희귀문자𠮷 테스트"
pieces = sp.encode(text, out_type=str)   # 서브워드 시퀀스 (예: ['▁안녕하세요', ...])
ids    = sp.encode(text, out_type=int)   # 정수 ID 시퀀스
back1  = sp.decode(pieces)               # 서브워드 → 텍스트
back2  = sp.decode(ids)                  # ID → 텍스트
```

byte_fallback 효과

미등록 문자 → <0x..> 바이트 토큰으로 분해 → 정보 손실 없이 복원 가능

BPE 인코딩 동작(탐욕적)

학습 시 생성한 병합 규칙 순서(merge rank) 를 따라 가장 긴 서브워드로 병합, 더 적용할 규칙이 없으면 중단

설계 & 평가

- 선택 가이드
  - BPE: 단순·빠름·결정적 → 대부분의 LLM, 코드·영문 중심
  - Unigram: 확률 모델, 분절 유연·잡음 강건 → 다국어/노이즈 많은 데이터
  - vocab_size: 32k(경량), 50k(여유), 다국어는 100k± 검토 / n-gram 실험은 8–16k
- 평가 지표 & 목표
  - 문자/토큰 비율(↓ 좋음), 토큰 길이 전체 중 95%에 해당하는 길이(짧을수록), <unk> 비율(0에 수렴)
  - 라운드트립(encode→decode 원문 동일) 1k 샘플 100% 통과
  - (모델 기준) VAL loss/PPL 비교, 학습·추론 토큰 수(비용) 비교

예제

```
import sentencepiece as spm, numpy as np, random
sp = spm.SentencePieceProcessor(model_file="spm_bpe32k.model")
texts = [l.strip() for l in open("valid.txt", encoding="utf-8") if l.strip()][:10000]

lens = [len(sp.encode(t, out_type=int)) for t in texts]
print({"n":len(lens), "avg":float(np.mean(lens)), "p95":float(np.percentile(lens,95))})

# 라운드트립 정확도
samples = random.sample(texts, min(1000, len(texts)))
ok = sum(sp.decode(sp.encode(t, out_type=int)) == t for t in samples)
print({"roundtrip_ok": ok, "total": len(samples)})

# (byte_fallback 모델일 때) 바이트 토큰 사용률 – byte_fallback사용하면 <unk>비율은 0
pieces = [sp.encode(t, out_type=str) for t in samples]
byte_ratio = sum(sum(p.startswith("<0x") for p in ps) for ps in pieces) / sum(len(ps) for
ps in pieces)
print({"byte_piece_ratio": byte_ratio})
```

Hugging Face 연동

```
pip install -U transformers tokenizers sentencepiece
from transformers import T5Tokenizer
tok = T5Tokenizer(
    vocab_file="spm_bpe32k.model",
    unk_token="<unk>", bos_token="<s>", eos_token="</s>",
    additional_special_tokens=["[CLS]","[SEP]","[MASK]"]
)
print(tok("안녕하세요😳"))
```