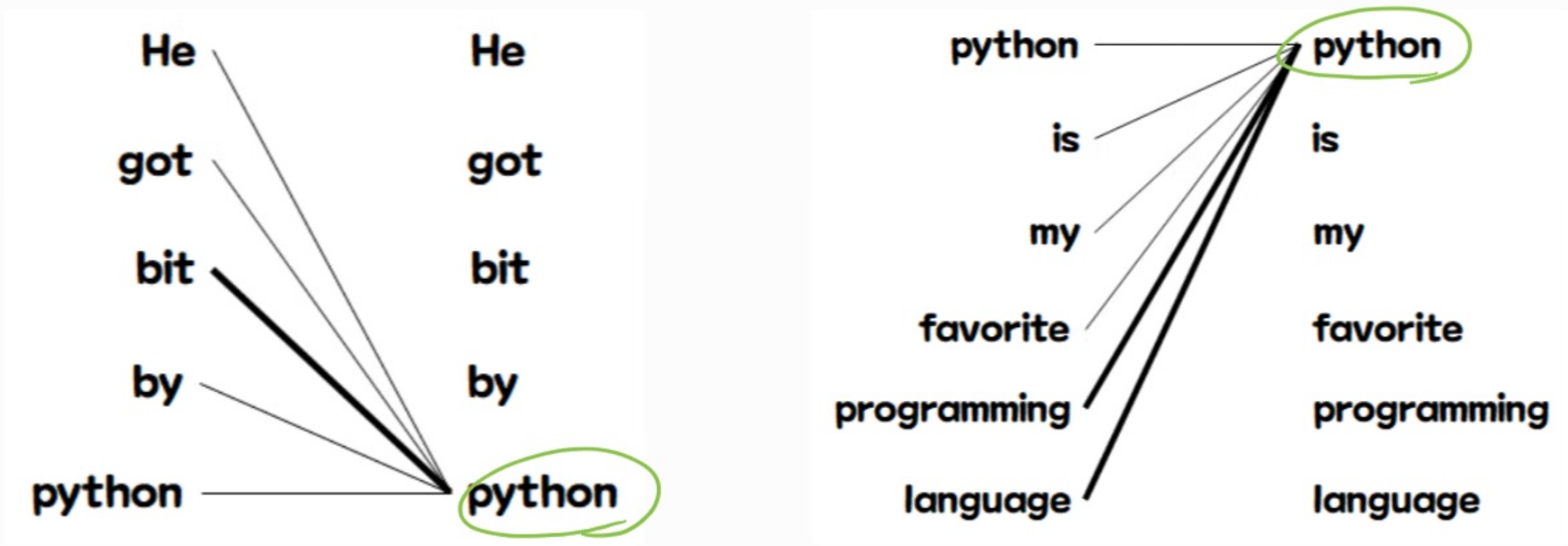
BERT 정리

: 구글에서 발표한 고성능 "임베딩" 모델

"문맥"(context)을 고려한 모델 (⇒ 특별한 점)

A: He got bit by python

B: python is my favorite programming language



비록 같은 단어 이지만, "문맥"상 A문장은 뱀(Python), B 문장 프로그램 언어(Python)

BERT는 문장 03 부터 문맥 기반의 동적 임베딩을 생성한다.

BERT 동작.

· Bert는 트랜스포머의 인코더 만 사용한다.

→ 인코더의 출력은 잘 정제(표현)된 Input text의 임베딩.

BERT 사전학습

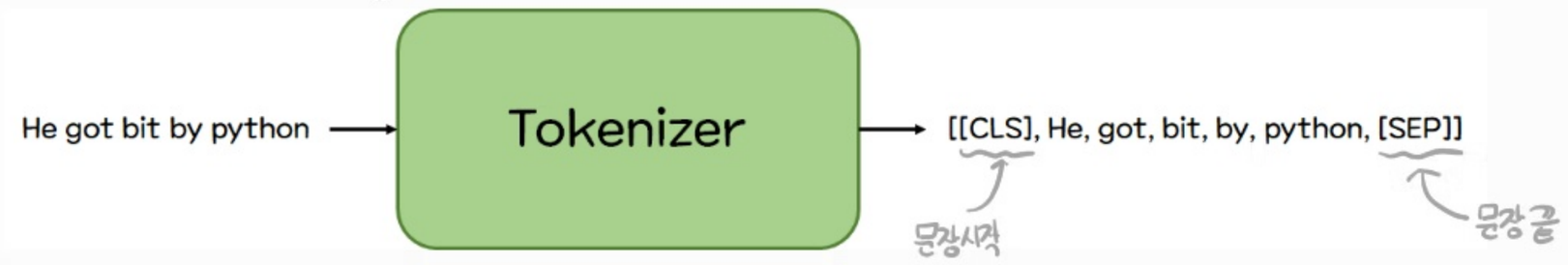
- Bert는 MLM(마스킹된 언어 모델링 = Masked language modeling) 과
  NSP(다음 문장 예측 = Next sentence prediction) 2가지 태스크를 통해 사전 학습됨.
- 우리는 잘 학습된 Bert를 fine-tunning 하면 됨.

학습 과정

입력 데이터 ⟶ 임베딩 (3가지)

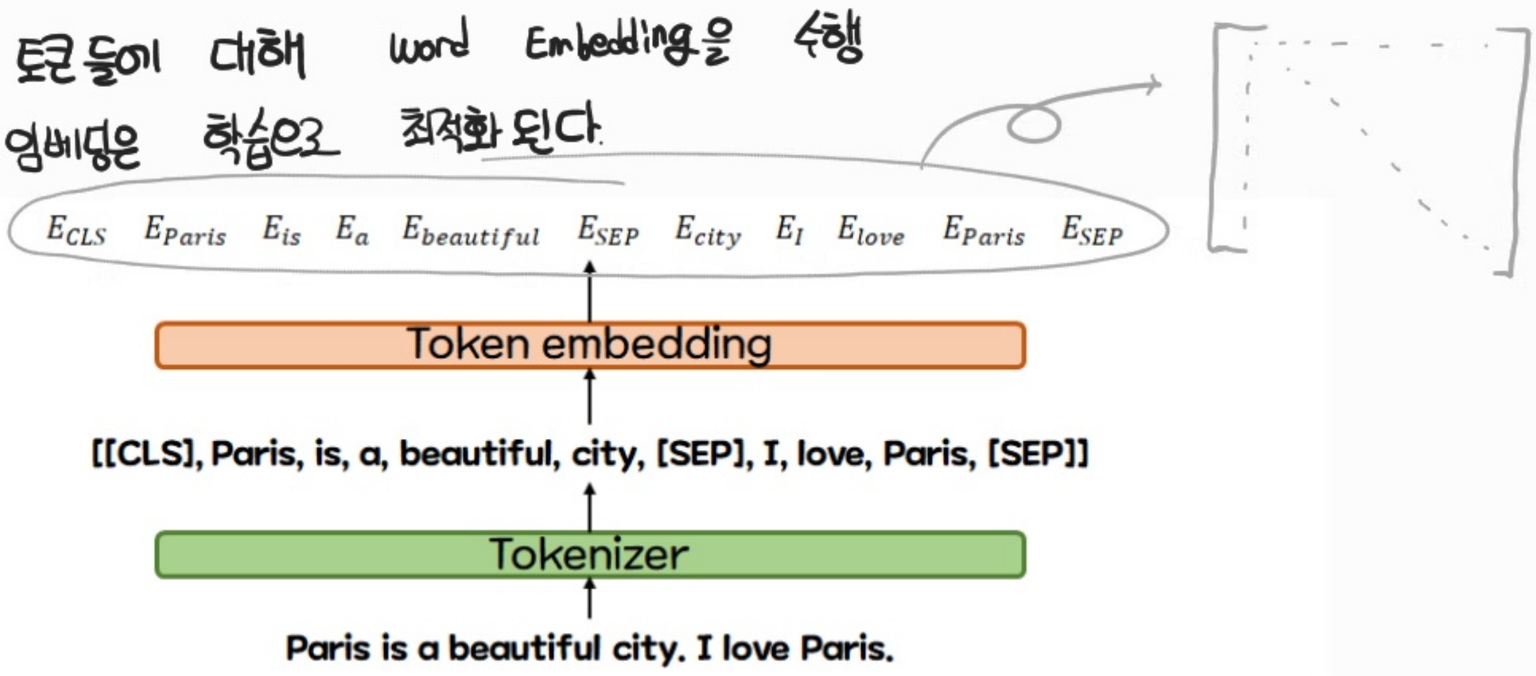
Tokenization (토큰화)

입력 문장을 토큰 단위로 분할 하는 작업
↳ 단어 단위 or 단어구 단위 등등.



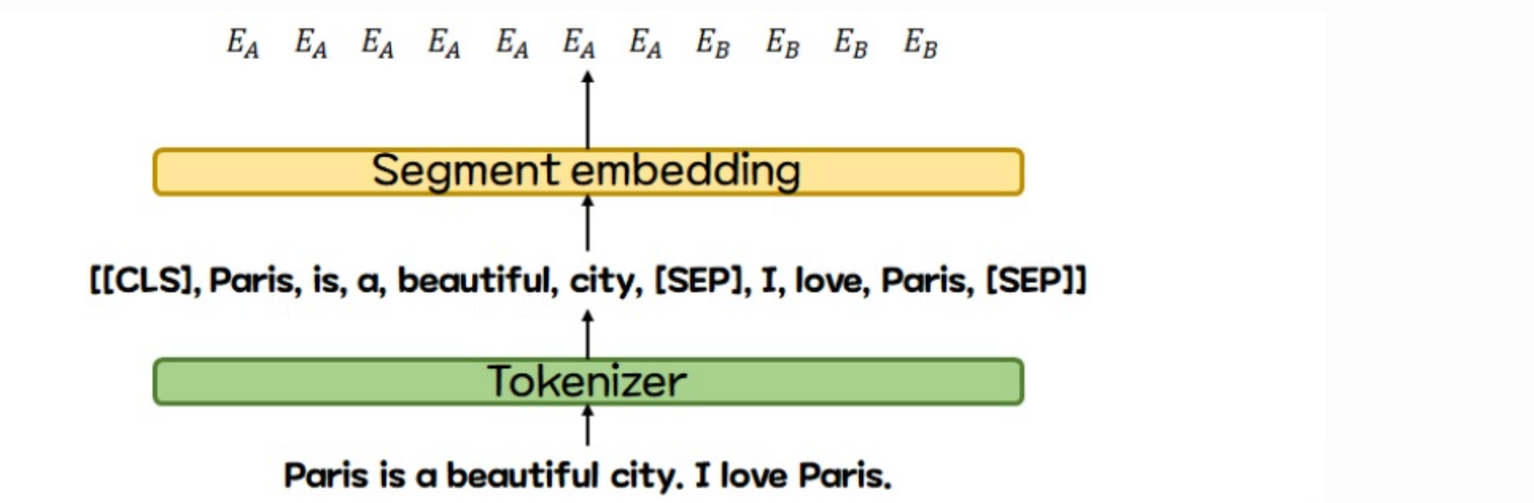
토큰 임베딩 (Token embedding)

1. 주어진 입력 문장에 대해 Tokenization
2. 해당 토큰들에 대해 word Embedding을 수행
3. 토큰 임베딩은 학습으로 최적화 된다.



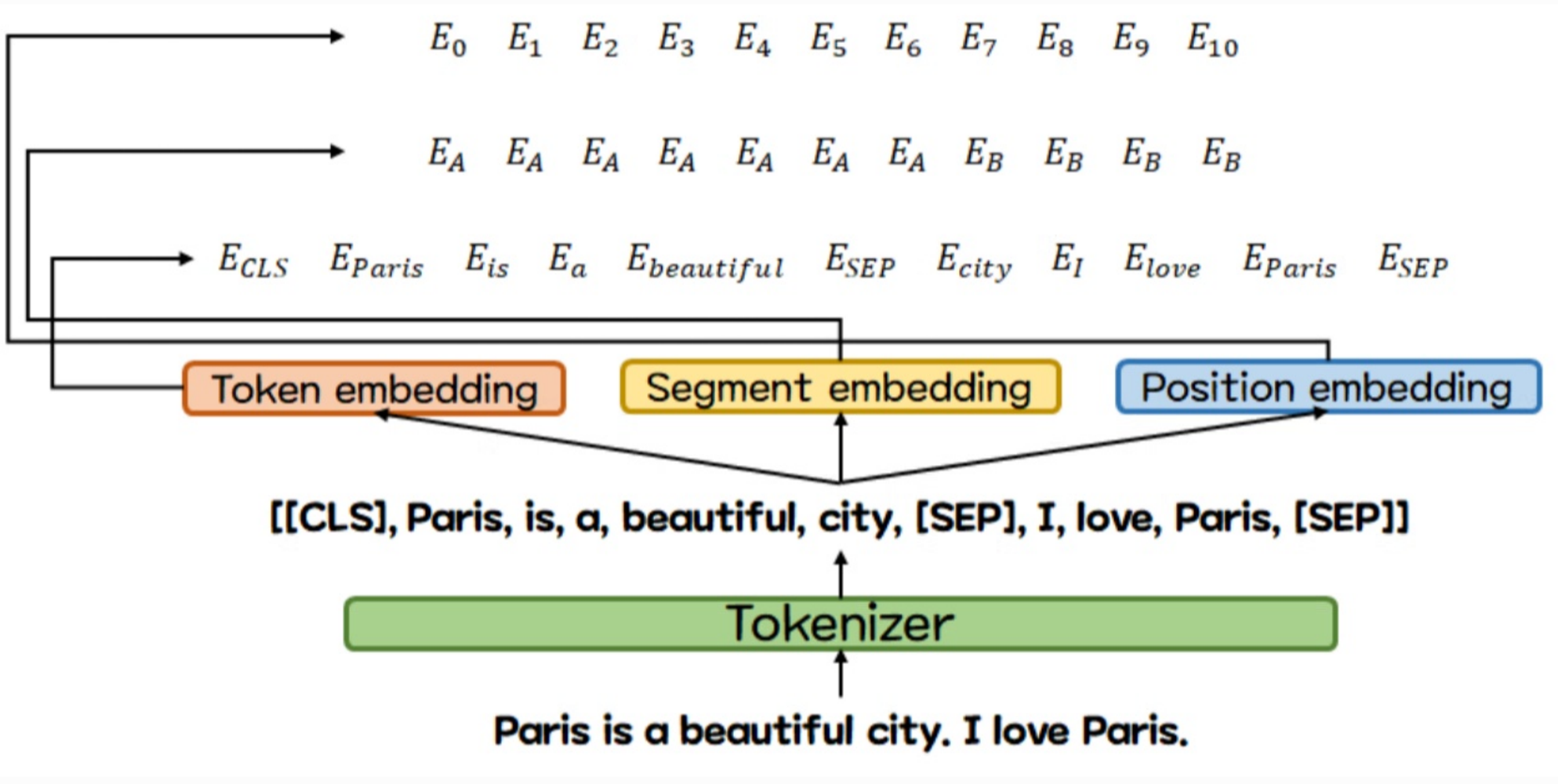
세그먼트 임베딩 (segment embedding)

- 문장을 구분하는 임베딩.
- 문장 구분을 위해 모델에게 일종의 지표를 제공



위치 임베딩 (Position embedding)

- 순서정보를 담는 임베딩.



Bert의 Tokenizer (워드피스 토크나이저)

- 어휘 사전 외의 (OOV)의 단어를 "쪼개서" 토큰화

ex) pretraining ⇒ [Pre, ##train, ##ing]

↳ 앞 단어의 종속을 표시

사전학습 전략

1. MLM (마스크 언어 모델링)

언어 모델링은 임의의 문장이 주어지고 단어를 보며, 다음 단어를 예측 하도록 모델을 학습시키는 것

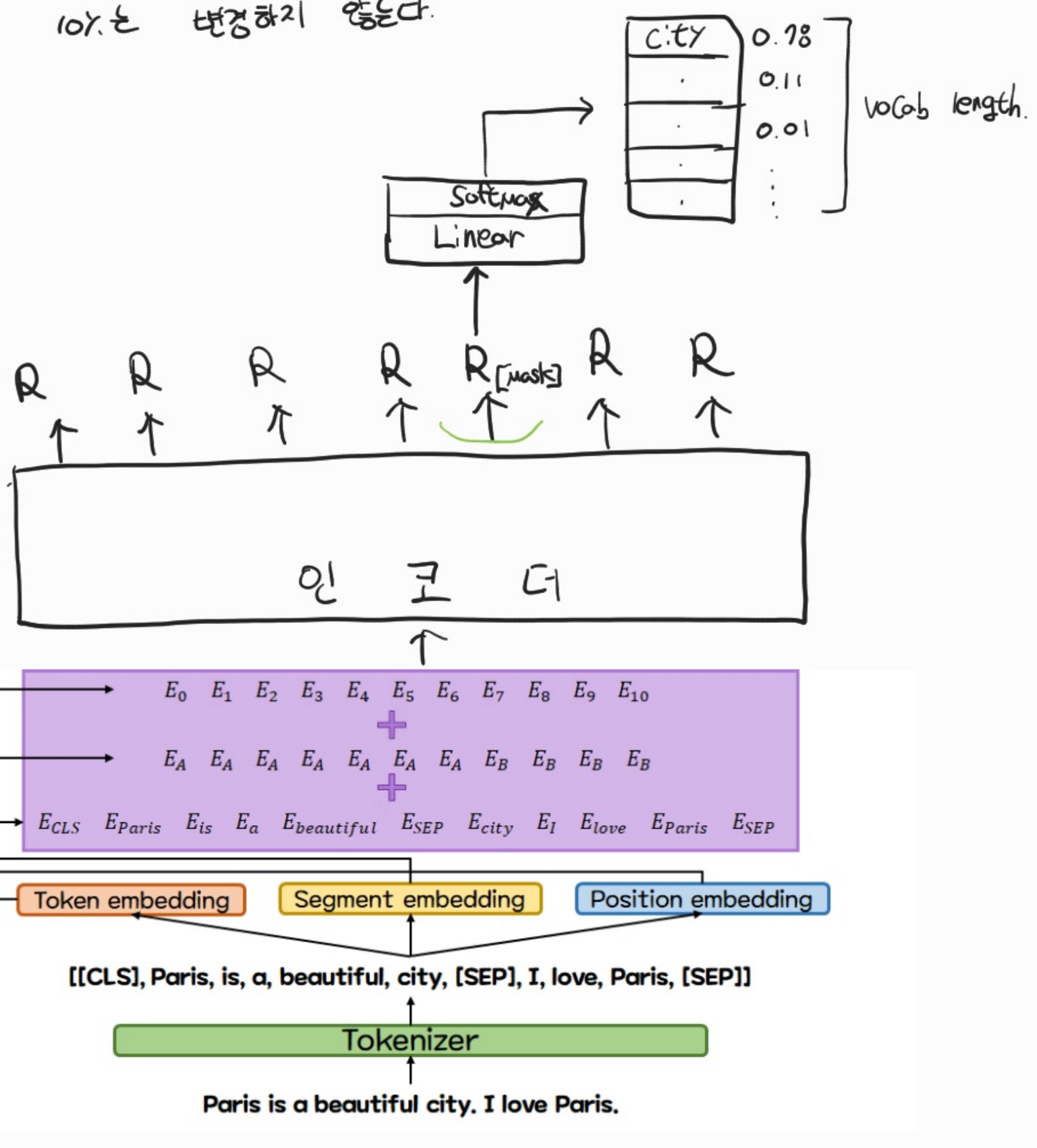
① 토큰의 15%를 무작위로 마스킹

② 80-10-10 규칙

15% 중 80%를 [MASK] 토큰으로 교체

15% 중 10%를 임의의 토큰으로 교체

15% 중 10%는 변경하지 않는다.

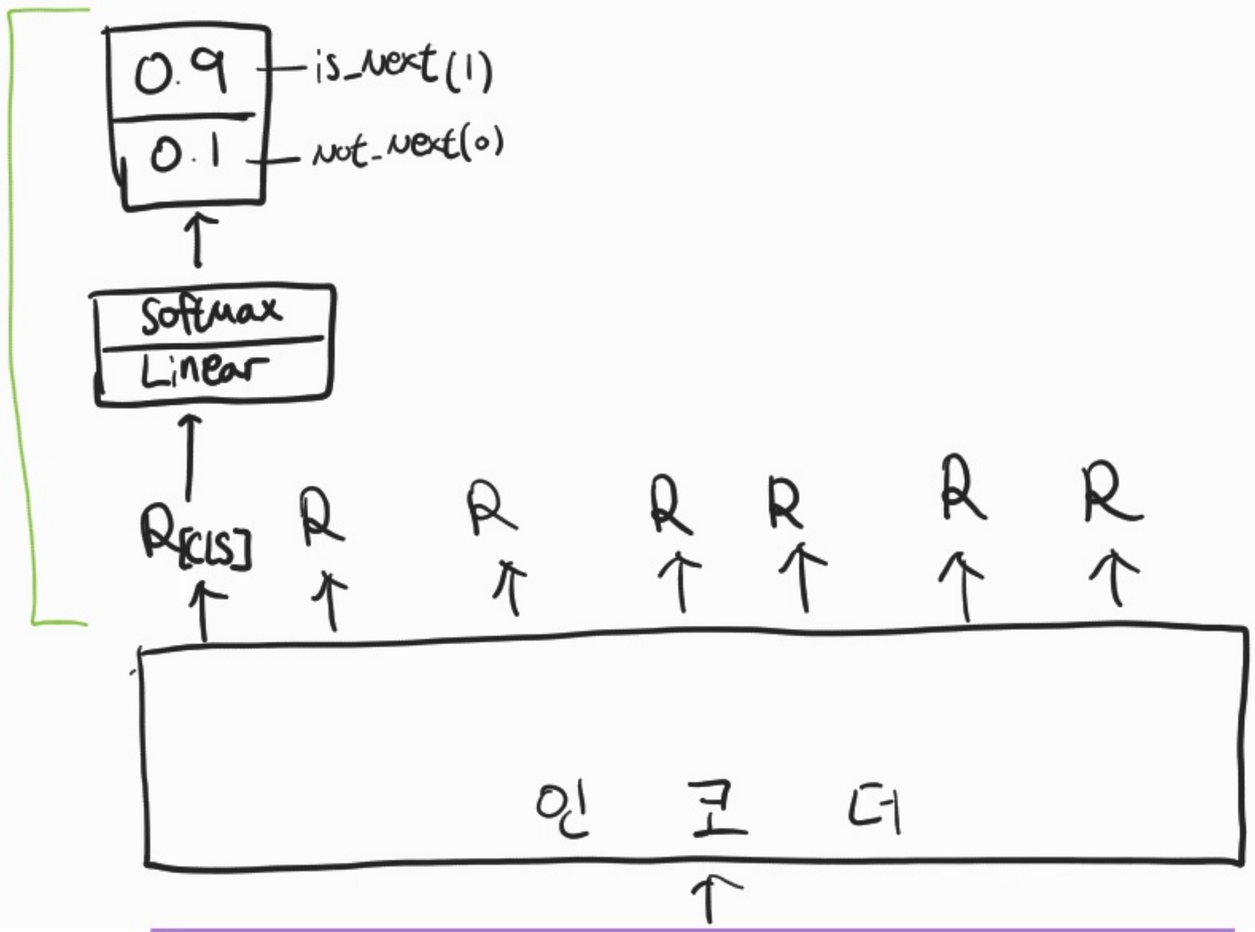


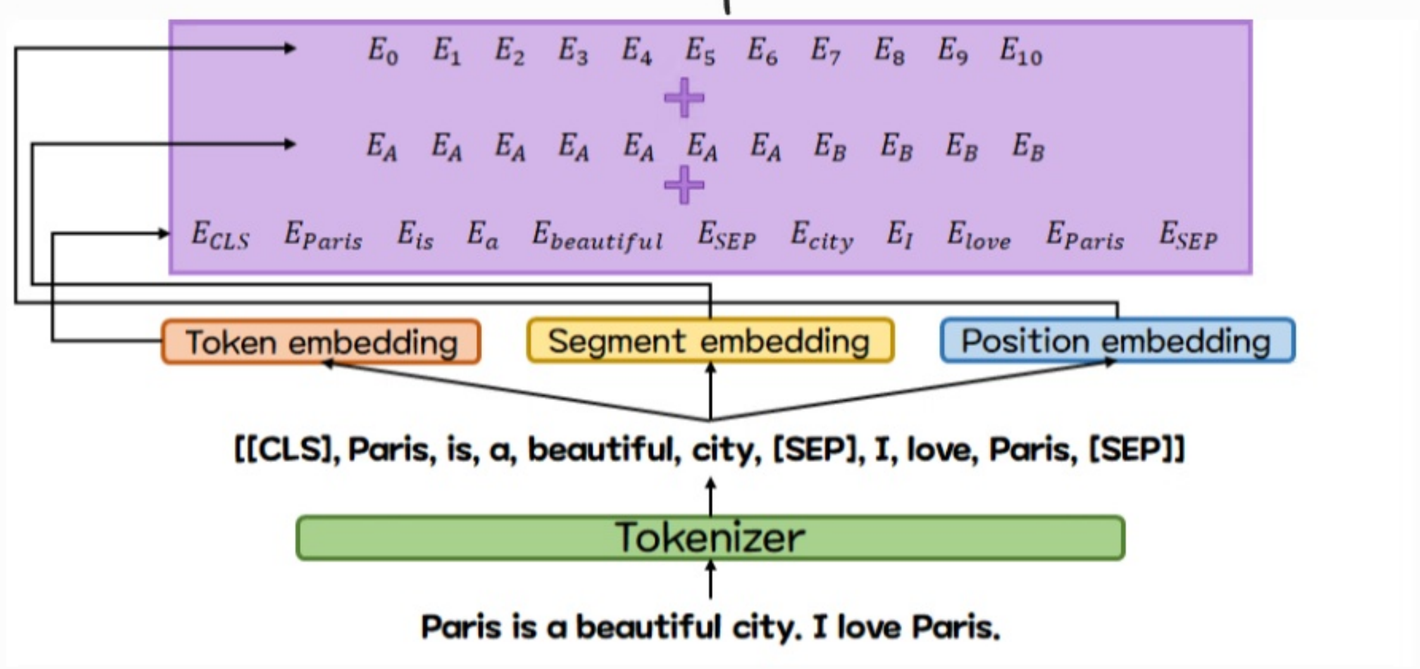
2. NSP (다음 문장 예측)

- 이진 분류로 볼수 있다.

문장 쌍	레이블
She cooked pasta(그녀는 파스타를 요리했다) It was delicious(맛있었다)	isNext 1
Jack loves songwriting(잭은 작곡을 좋아한다) He wrote a new song(그는 새 노래를 썼다)	isNext 1
Birds fly in the sky(새들은 하늘을 난다) He was reading(그는 읽고 있었다)	NotNext 0
Turn the radio on(라디오 켜줘) She bought a new hat(그녀는 새 모자를 샀다)	NotNext 0

- Tokenization 과 embedding, encoding 과정은 동일.





- why $R_{[CLS]}$ 인가?

  $R_{[CLS]}$는 기본적으로 모든 토큰의 기본적인 집계 표현을 보유

- Activation 함수로는 Gelu을 사용

어휘사전 생성.

1. 바이트 쌍 인코딩(BPE)
   단어를 글자로 나누고 글자의 출현 빈도를 기준으로 vocabulary에 추가.
2. 바이트 수준 바이트 쌍 인코딩(BBPE)
   BPE와 동일 하지만 글자를 Byte 단위로 맞춤 a ⇒ 0x61
3. 워드 피스
   출현 빈도가 아닌 가능도(likehood) $\frac{P(st)}{P(s)P(t)}$

BERT 활용

앞선 내용은 MLM과 NSP를 통해 Bert를 학습 시킨다고 언급

학습된 Bert는 ① 임베딩 추출을 통한 특징 추출기로 사용, ② 텍스트 분류, ③ 질문-응답 같은 곳에서 활용 된다.

허깅페이스 트랜스포머

트랜스포머 오픈라이브러리.

```
pip install transformer == version
```

임베딩 추출하기.

```
!pip install transformer == 3.5.1 → transform 설치

import transformser import BertModel, BertTokenizer
import torch

sentence = "I love Paris"

tokens = tokenizer.toeknize(setence) → Input text token화
# tokens = ['[CLS]', 'i', 'love', 'paris', '[SEP]', '[PAD]', '[PAD]']
attention_mask = [1, 1, 1, 1, 1, 0, 0] → 주목할 토큰 mask

token_ids = tokenizer.convert_tokens_to_ids(tokens) → 토큰을 고유한 ID에 매핑
# token_ids = [101, 4045, 2293, 3000, 102, 0, 0]

token_ids = torch.tensor(token_ids).unsqueeze(0)
attention_mask = torch.tensor(attention).unsqueeze(0) → Torch.Tensor

model = BertModel.from_pretrained('bert-base-uncased') → model import

hidden_rep, cls_head = model(token_ids, attention_mask = attention_mask) → 임베딩 값 return
```

hidden_rep = Size [1, 7, 768] # [batch_size, sequence_len, hidden_size]

hidden_rep[0][0] = <CLS> 토큰의 임베딩 벡터 값 = [1, 768] ← 같은?

cls_head = Size [1, 768] # [Batch_Size, hidden Size]

또한, 모든 encoder의 출력 값을 모두 받아 올수도 있다. (책 참조 P.120~122)

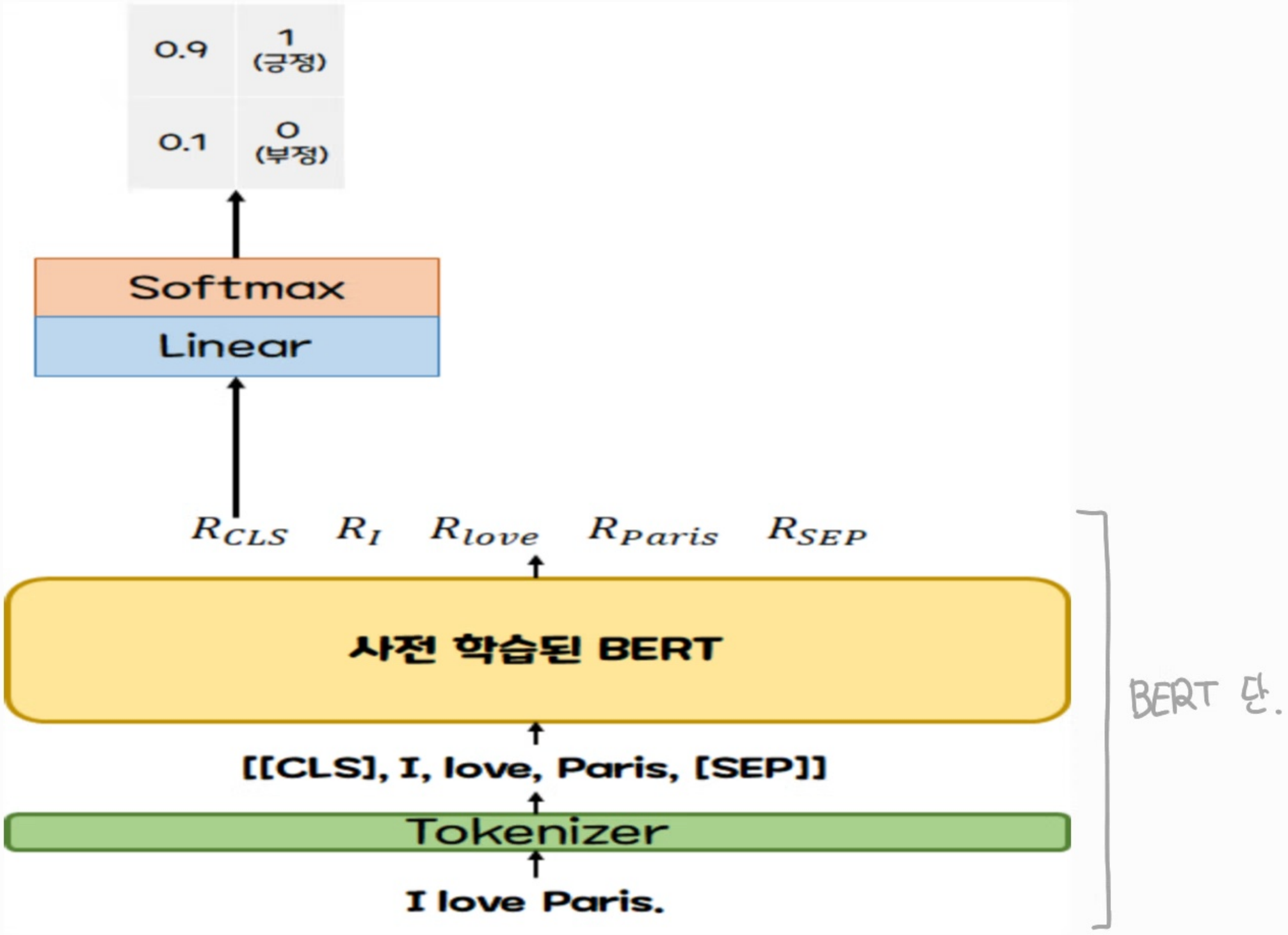
BERT Fine-tuning

1. 텍스트 분류

사전 학습된 Bert 모델을 통해서 텍스트의 감정을 분류

이전 설명한 Bert로부터 입력 Text의 토큰을 대상하고 표현 벡터를 통해 수행 → 여기서 사용되는 표현은 <CLS> 토큰에 대한 임베딩이다.

→ 모든 토큰에 대한 집계 표현.



2. 자연어 추론

아래와 같은 테스크가 있을때..

전제와 가정이 주어질때
가정의 참/거짓을 판단하는 문제

Input		Label
전제	가설	레이블
그는 놀고 있다	그는 자고 있다	거짓
여러 남성이 플레이하는 축구 게임	몇몇 남자들이 스포츠를 하고 있다	참
웃고 있는 노인과 청년	두 남자가 바닥에서 놀고 있는 강아지들을 보고 웃고 있다	중립

3. 질문 - 응답

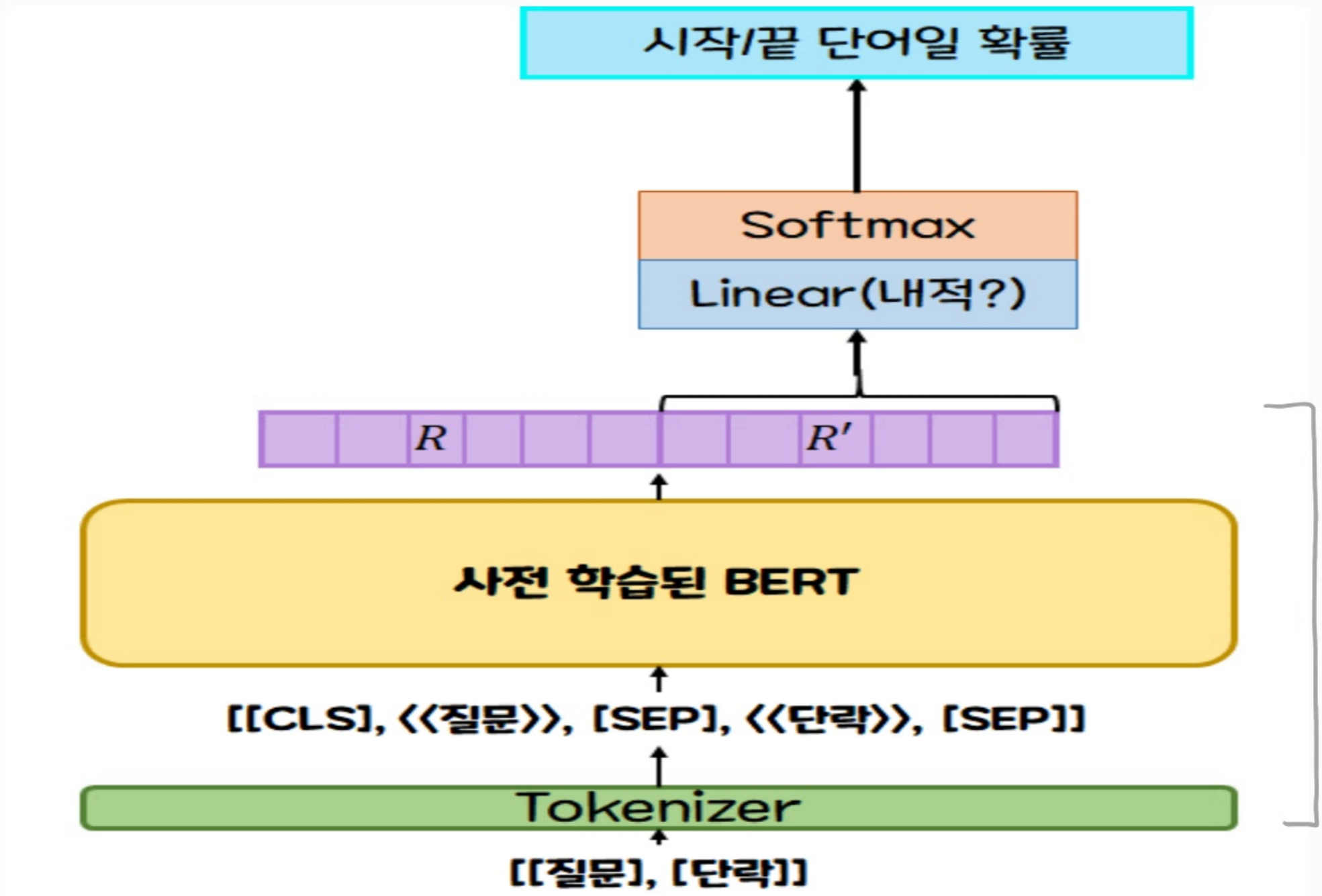
질문과 단락이 주어질때 질문에 대한 응답을 단락에서 찾아 대당하는 문제

↳ 단락이 가지고 있는 응답 내용중 시작과 끝 인덱스를 찾는다.

↳ 확률이 높은 것을 찾는다.

시작/끝 확률? ⇒ 시작 : $\frac{e^{S \cdot R_i}}{\sum e^{S \cdot R_i}}$ ↳ i번째 토큰이 시작 토큰일 확률

끝 : $\frac{e^{E \cdot R_i}}{\sum e^{E \cdot R_i}}$ ↳ i번째 토큰이 끝 토큰일 확률.



4. 개체 명 인식(NER)