Natural Language Processing

언어 모델: BERT 모델

강사 양석환

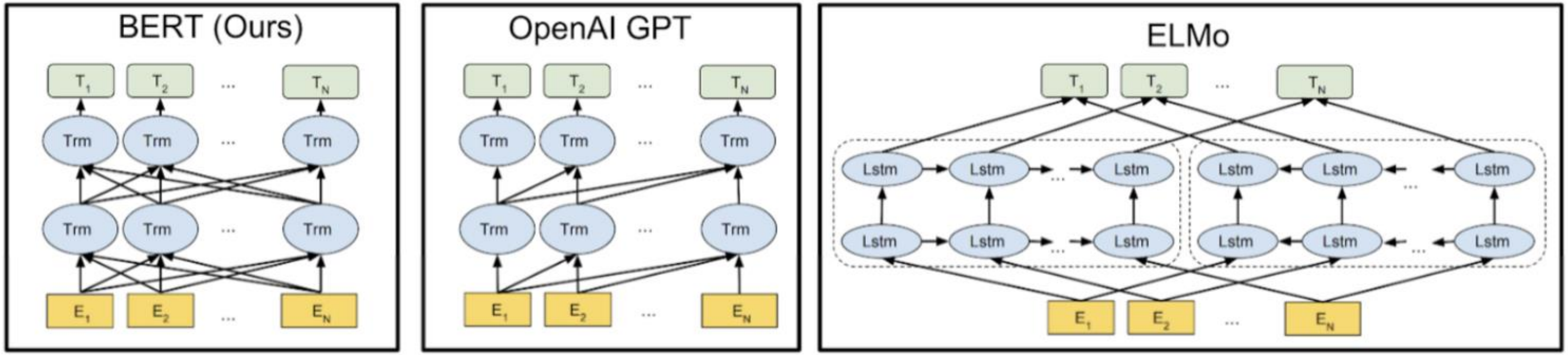
BERT 모델 개요

BERT 개요

- BERT (Bidirectional Encoder Representation Transformers)
  - 2018년 Google이 발표한 자연어 처리를 위한 딥 러닝 모델
    - BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding (https://arxiv.org/abs/1810.04805)
  - 비지도 사전 학습을 한 모델에 추가로 하나의 완전 연결 계층만 추가한 후 미세 조정을 통해 총 11개의 자연어 처리 문제에서 최고의 성능을 보여줌
  - 영상인식 계열에 비하여 발전이 늦은 언어처리 딥 러닝의 한계를 돌파하는 계기가 될 것으로 기대 받음

BERT 개요

- **양방향성의 사전 학습 모델**
  - 기존 모델인 GPT, ELMo 등과 달리 양방향성을 가진 사전 학습 모델이며
  - 이로 인하여 기존 모델보다 뛰어난 성능을 보임
  - GPT: 단방향, ELMo: 단방향 모델 2개를 반대 방향으로 결합시켜 만든 (불완전) 양방향
- **모델의 방향성 시각화**



BERT 개요

- BERT는 사전 학습 언어 모델이며
  - 문맥을 고려한 임베딩 모델이라고도 부름

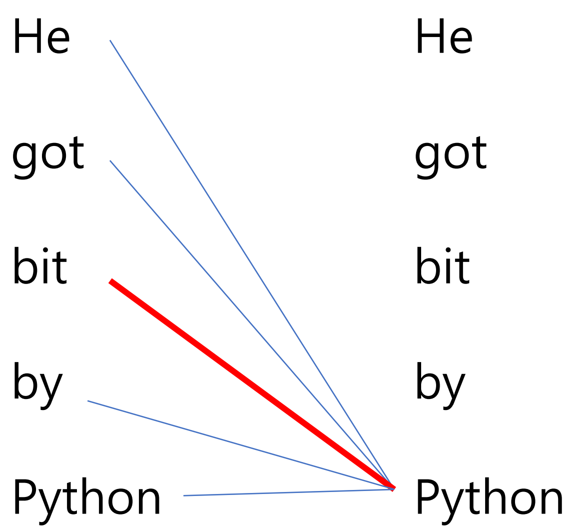
- 문맥 기반의 임베딩 모델과 문맥 독립의 임베딩 모델 비교
  - He got bit by Python (파이썬이 그를 물었다).
    뱀 종류
  - Python is my favorite programming language (내가 가장 좋아하는 프로그래밍 언어는 파이썬 이다).
    프로그래밍 언어 종류

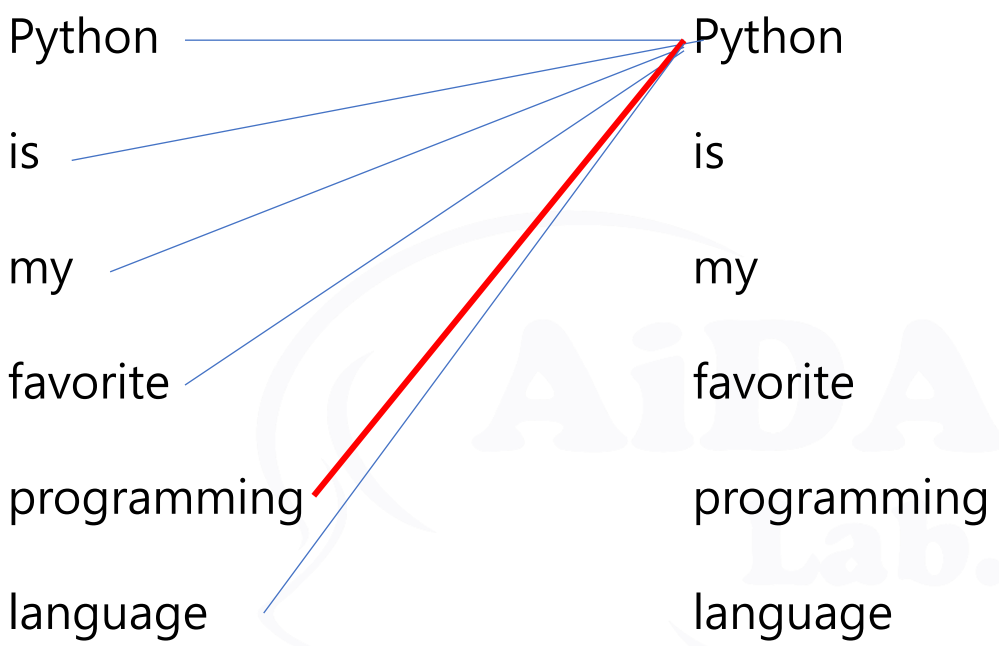
Word2Vec: 두 단어를 동일한 표현으로 임베딩 → 문맥 독립 모델

BERT: 두 단어에 대하여 서로 다른 임베딩을 제공 → 문맥 기반 모델

BERT 개요

- **“Python”과 다른 모든 단어의 관계**





- **BERT는 문장의 각 단어를 문장의 다른 모든 단어와 연결시키고 그 관련성을 인지하여 임베딩 수행**

BERT 개요

- 트랜스포머 모델을 기반으로 하며, 트랜스포머 모델의 인코더만 사용
- 트랜스포머 인코더는 양방향으로 문장을 읽을 수 있으므로 양방향

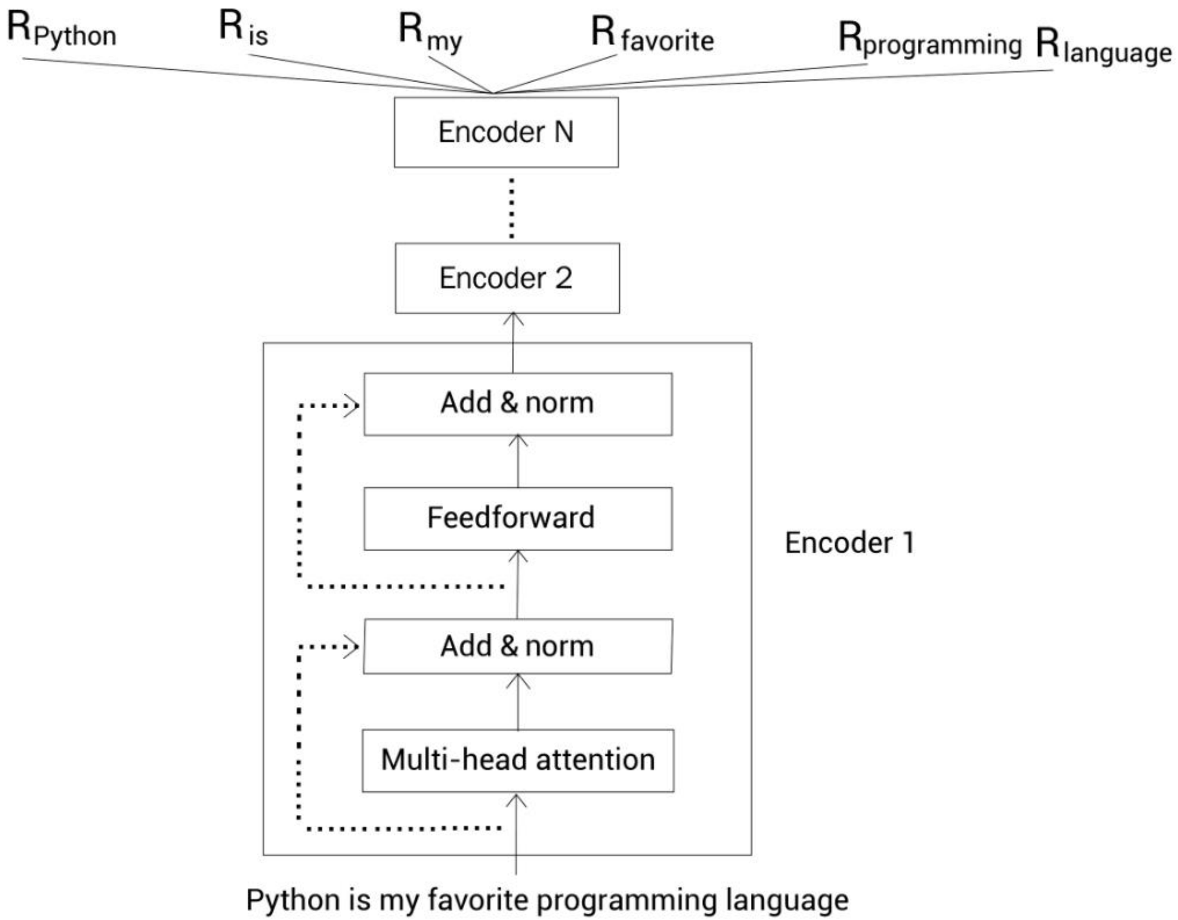
  → BERT는 양방향 인코더 표현을 사용

  - 앞의 예시에서...
    - "He got bit by Python" 문장을 트랜스포머 인코더에 입력으로 제공
    - 문장의 각 단어에 대한 문맥 표현(임베딩)을 출력으로 가져옴
    - 인코더에 문장을 입력하면 → 인코더는 멀티 헤드 어텐션 메커니즘으로 문장의 각 단어의 문맥을 이해 → 문장에 있는 각 단어의 문맥 표현을 출력으로 반환

문장의 각 단어를 문장의 다른 모든 단어와 연결해 관계 및 문맥을 고려해 의미를 학습함

BERT 개요

- BERT에 입력된 문장의 각 단어 표현 출력



BERT 모델의 구조

• BERT-base와 BERT-large의 두 가지 구성의 모델 제시

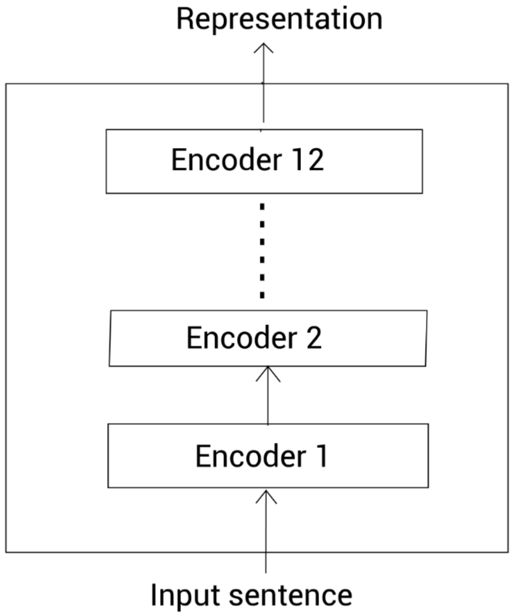
12개의 인코더 레이어가 스택처럼 쌓인 형태로 구성

인코더의 FFNN: 768개 차원의 은닉 유닛으로 구성

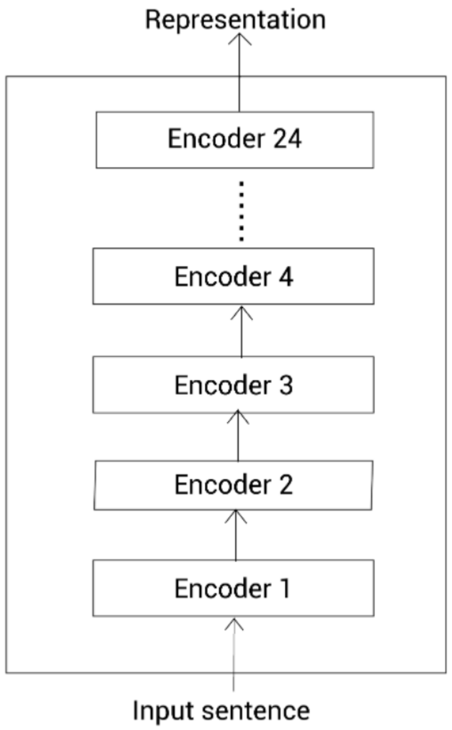
BERT-base에서 얻은 표현크기: 768

L: 인코더 레이어 수
A: 어텐션 헤드
H: 은닉유닛

BERT-base는
L=12, A=12, H=768
총변수의 수=1억1천만개



BERT-base



BERT-large

24개의 인코더 레이어가 스택처럼 쌓인 형태로 구성

모든 인코더는 16개의 어텐션 헤드 사용

인코더의 FFNN: 1024개 차원의 은닉 유닛으로 구성

BERT-large에서 얻은 표현크기: 1024

BERT-large는
L=24, A=16, H=1024
총변수의 수=3억1천만개

BERT의 구조

- 두 가지 표준 구조 외에도 다른 조합으로 BERT 구축 가능
  - 예시
    - BERT-tiny: L=2, A=2, H=128
    - BERT-mini: L=4, A=4, H=256
    - BERT-small: L=4, A=8, H=512
    - BERT-medium: L=8, A=8, H=512
  - 컴퓨팅 리소스가 제한된 환경 → 작은 BERT가 적합
  - 그러나 BERT-base, BERT-large와 같은 표준 구조가 더 정확한 결과를 제공하기때문에 가장 널리 사용됨

BERT를 사용할 때 접하는 문제

- 입력 문장에 대한 적절한 표현을 생성하게 하려면 BERT를 어떻게 학습시켜야 하는가?
- 학습에는 어떤 데이터셋을 사용해야 하는가?
- 학습 방법은 무엇인가?

BERT에서의 사전 학습

- 사전 학습
  - 특정 태스크에 대한 방대한 데이터셋으로 모델을 학습시키고 저장
  - 새로운 태스크가 주어지면 임의의 가중치로 모델을 초기화하는 대신 이미 학습된 모델(사전 학습된 모델)의 가중치로 모델을 초기화
  - 새로운 태스크에 따라 가중치를 조정(미세 조정, Fine-Tuning)

- BERT의 사전 학습
  - 입력 표현과 MLM, NSP의 두 가지 태스크를 이용하여 사전 학습 수행

BERT에서의 사전 학습

- BERT의 입력 표현
  - BERT의 입력은 다음 세 가지의 임베딩의 합으로 구성
    - 토큰 임베딩
    - 세그먼트 임베딩
    - 위치 임베딩

BERT에서의 사전 학습

- **BERT에서 사용하는 토크나이저: WordPiece Tokenizer**
  - **하위 단어(SubWord) 토큰화 알고리즘을 기반으로 함**

Let us start pretraining the model (모델 사전 학습을 시작하자).
tokens = [let, us, start, pre, ##train, ##ing, the, model]

  - **pretraining → pre + ##train + ##ing로 분할됨**
  - **BERT는 WordPiece Tokenizer를 사용해서 토큰화할 때, 단어가 어휘사전에 있는지 확인 → 있으면 그대로 토큰으로 사용,**

    → 없으면 SubWord로 분할하여 다시 어휘사전 확인 → ...

개별 문자에 도달할 때까지 반복 → OOV(Out of Vocabulary)의 단어 처리에 효과적

BERT에서의 사전 학습

- **토큰 임베딩**

토큰 임베딩의 변수들은 사전학습이 진행되면서 학습됨

입력 문장
A: Paris is a beautiful city (파리는 아름다운 도시다).
B: I love paris (나는 파리를 좋아한다).

입력된 문장을 모두 토큰화하여 토큰들을 추출

tokens = [Paris, is, a, beautiful, city, I, love, Paris]

첫 번째 문장의 시작부분에 [CLS] 토큰 추가

tokens = [[CLS], Paris, is, a, beautiful, city, I, love, Paris]

모든 문장 끝에 [SEP] 토큰 추가

tokens = [[CLS], Paris, is, a, beautiful, city , [SEP], I, love, Paris, [SEP]]

토큰 임베딩 레이어를 사용해 임베딩 변환

Input	[CLS]	Paris	is	a	beautiful	city	[SEP]	I	love	Paris	[SEP]
Token embeddings	E_{CLS}	E_{Paris}	E_{is}	E_{a}	$E_{beautiful}$	E_{city}	$E_{[SEP]}$	E_{I}	E_{love}	E_{Paris}	$E_{[SEP]}$

BERT에서의 사전 학습

- **세그먼트 임베딩**

주어진 두 문장을 구별하는데 사용됨

A: Paris is a beautiful city (파리는 아름다운 도시다).

B: I love paris (나는 파리를 좋아한다).

토큰화

tokens = [[CLS], Paris, is, a, beautiful, city , [SEP], I, love, Paris, [SEP]]

토큰과는 별도로 두 문장을
구분하기 위해 모델에 일종의
지표를 제공함

Input	[CLS]	Paris	is	a	beautiful	city	[SEP]	I	love	Paris	[SEP]
Segment embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B

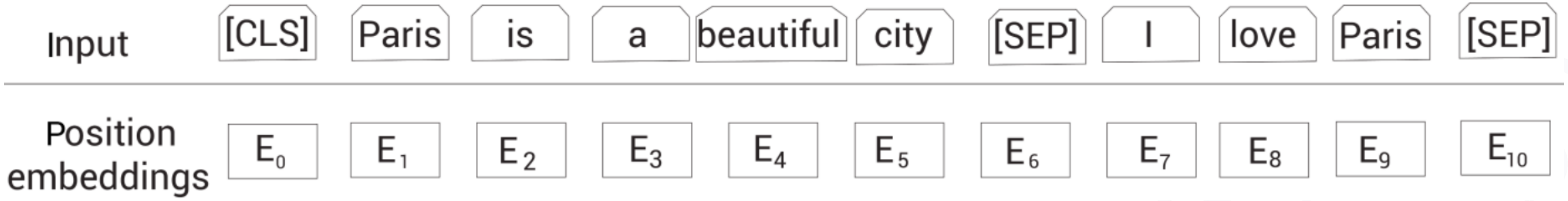
문장이 하나만 있다면 모든 문장이 E_A에 매핑됨

BERT에서의 사전 학습

- **위치 임베딩**
  - **트랜스포머 모델은**
    - 어떤 반복 메커니즘도 사용하지 않고 모든 단어를 병렬 처리

      → 단어 순서와 관련된 정보 제공이 필수 → 위치 인코딩 사용
  - **BERT는 본질적으로 트랜스포머의 인코더이므로…**
    - BERT에 데이터를 직접 입력하기 전에 문장에서 단어(토큰)의 위치에 대한 정보 제공 요구

      → 위치 임베딩 레이어를 사용해 각 토큰에 대한 위치 임베딩 출력을 확보



BERT에서의 사전 학습

- **최종 입력 데이터 표현**
  - **입력 문장 → 토큰으로 변환 → 토큰 임베딩 → 세그먼트 임베딩 → 위치임베딩 → 최종 임베딩 결과(세가지 임베딩의 합산) 확보 → BERT에 입력**

Input	[CLS]	Paris	is	a	beautiful	city	[SEP]	I	love	Paris	[SEP]
Token embeddings	$E_{[CLS]}$	E_{Paris}	E_{is}	E_{a}	$E_{beautiful}$	E_{city}	$E_{[SEP]}$	E_{I}	E_{love}	E_{Paris}	$E_{[SEP]}$
	+	+	+	+	+	+	+	+	+	+	+
Segment embeddings	E_A	E_A	E_A	E_A	E_A	E_A	E_A	E_B	E_B	E_B	E_B
	+	+	+	+	+	+	+	+	+	+	+
Position embeddings	E_0	E_1	E_2	E_3	E_4	E_5	E_6	E_7	E_8	E_9	E_{10}

BERT의 사전 학습

- BERT는 두 가지의 태스크에 대하여 사전 학습을 수행함
  - 마스크 언어 모델링 (Masked Language Modeling, MLM)
  - 다음 문장 예측 (Next Sentence Prediction, NSP)

언어 모델링이란?

임의의 문장을 주고 단어를 순서대로 보면서 다음 단어를 예측하도록 모델을 학습시키는 것

언어 모델링은 크게 두 가지로 분류됨

- 자동 회귀 언어 모델링 (Auto-Regressive Language Modeling)
- 자동 인코딩 언어 모델링 (Auto-Encoding Language Modeling)

BERT의 사전 학습

- 언어 모델링
  - 자동 회귀 언어 모델링

모델은 공백을 예측해야 함

Paris is a beautiful city. I love Paris. → Paris is a beautiful ____. I love Paris.

- 전방(→) 예측 (Forward(Left to Right) Prediction)
  - 예측을 위하여 문장의 왼쪽에서 오른쪽으로 공백까지 모든 단어를 읽음

    → Paris is a beautiful ____.
- 후방(←) 예측 (Backward(Right to Left) Prediction)
  - 예측을 위하여 문장의 오른쪽에서 왼쪽으로 공백까지 모든 단어를 읽음

    → ____. I love Paris.
- 자동 회귀 언어 모델은 원래 단방향이므로 한 방향으로만 문장을 읽음

BERT의 사전 학습

- **자동 인코딩 언어 모델링**
  - 전방(→) 예측, 후방(←) 예측을 모두 활용함. 즉, 예측을 하면서 양방향으로 문장을 읽음
  - 자동 인코딩 언어 모델은 본질적으로 양방향 시스템

    Paris is a beautiful ____. I love Paris

  - 양방향으로 문장을 읽으면 문장 이해 측면에서 더 명확해지므로 더욱 정확한 결과를 제공함

마스크 언어 모델링(MLM)

- BERT: 자동 인코딩 언어 모델로 예측을 위해 문장을 양방향으로 읽음

- 마스크 언어 모델링은
  - 빈칸 채우기 태스크(Cloze Task)라고도 부름
  - 주어진 입력 문장에서 전체 단어의 15%를 무작위로 마스킹하고
  - 마스크 된 단어를 예측하도록 모델을 학습시키는 것
  - 모델은 마스크 된 단어를 예측하기 위해 양방향으로 문장을 읽고 예측을 시도

마스크 언어 모델링(MLM)

- **마스크 언어 모델링의 동작**

Paris is a beautiful city. I love Paris.

토큰화

tokens = [Paris, is, a, beautiful, city, I, love, Paris]

[CLS], [SEP] 토큰 추가

tokens = [[CLS], Paris, is, a, beautiful, city , [SEP], I, love, Paris, [SEP]]

토큰(단어)의 15%를
무작위로 마스킹

tokens = [[CLS], Paris, is, a, beautiful, [MASK] , [SEP], I, love, Paris, [SEP]]

마스킹 후 [MASK]
토큰으로 교체

- **마스킹-대체 후 마스크 된 토큰을 예측하기 위한 BERT를 학습시킴**

문제점 발생: 위와 같은 방식으로 토큰을 마스킹하면 사전학습과 파인 튜닝 사이에 불일치가 발생함

마스크 언어 모델링(MLM)

- **MLM에서 사전학습과 파인 튜닝 간 불일치 문제의 원인**
  - **Process를 살펴보면**

1. 먼저 [MASK] 토큰을 예측해서 BERT를 사전학습 시킨다.
2. 학습 후에는 감정 분석과 같은 하위 작업(Downstream Task)을 위해 사전 학습된 BERT를 파인 튜닝한다.
3. 그런데 파인 튜닝에는 입력에 [MASK] 토큰이 없다.

→ 이 때문에 BERT가 사전 학습되는 방식과 파인 튜닝에 사용되는 방식 간에 불일치 발생

마스크 언어 모델링(MLM)

- **마스킹 언어 모델링의 문제점개선 방안**
  - **토큰을 마스킹하면 사전학습과 파인 튜닝 사이에 불일치가 발생함**

    → 문제 극복을 위해 80-10-10% 규칙 적용

    - 15% 중 80%의 토큰(실제 단어)을 [MASK] 토큰으로 교체

      tokens = [[CLS], Paris, is, a, beautiful, [MASK] , [SEP], I, love, Paris, [SEP]]

    - 15% 중 10%의 토큰(실제 단어)을 임의의 토큰(임의 단어)으로 교체

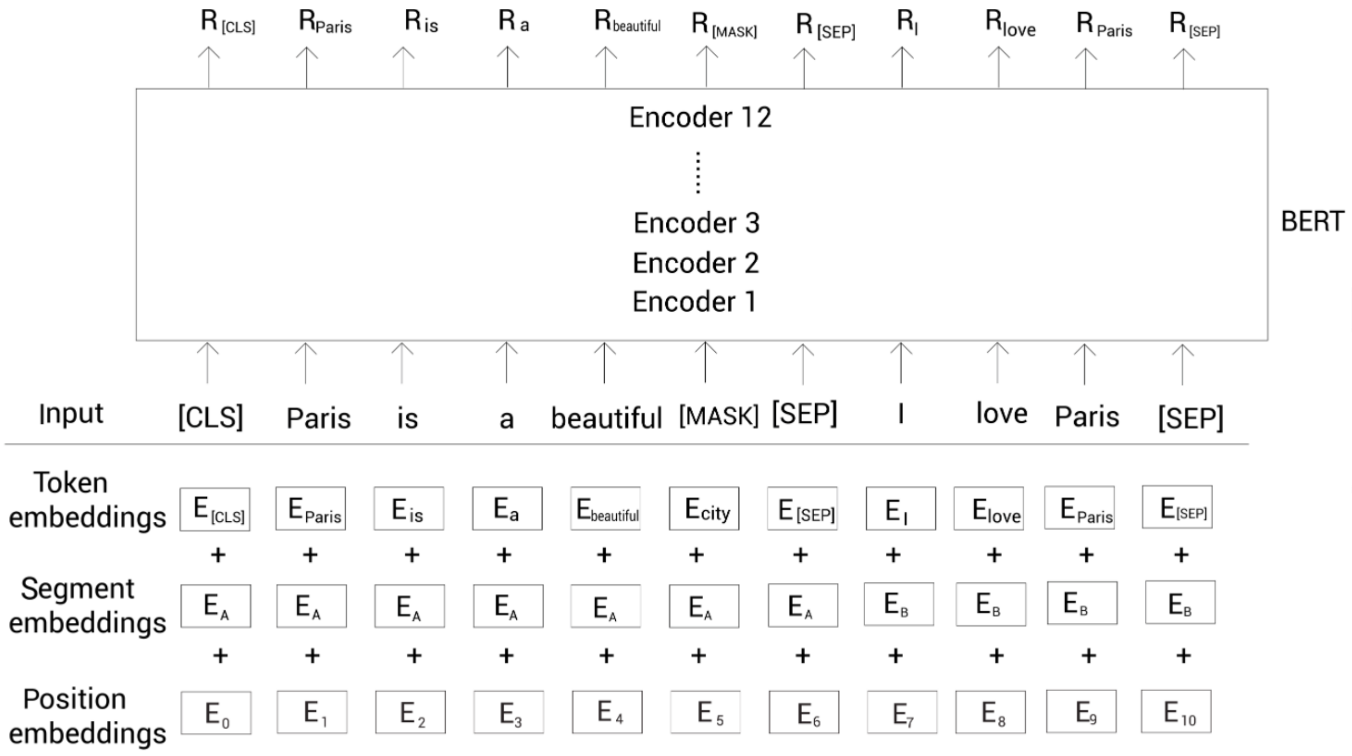
      tokens = [[CLS], Paris, is, a, beautiful, love, [SEP], I, love, Paris, [SEP]]

    - 15% 중 나머지 10%의 토큰은 어떤 변경도 하지 않음

      tokens = [[CLS], Paris, is, a, beautiful, city, [SEP], I, love, Paris, [SEP]]

마스크 언어 모델링(MLM)

- 토큰화 및 마스킹 후에
  - 입력 토큰을 토큰, 세그먼트, 위치 임베딩 레이어에 입력해서 입력 임베딩 확보
  - 입력 임베딩을 BERT에 제공

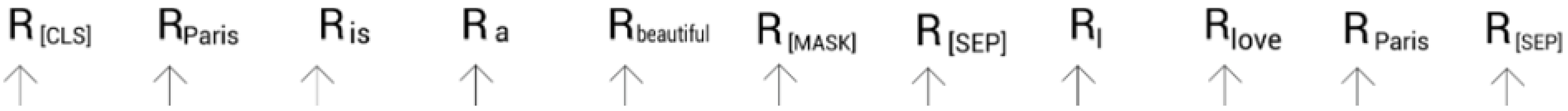


- BERT는 입력을 받은 다음 각 토큰의 표현 벡터를 출력으로 반환
- $R_{[CLS]}$: [CLS] 토큰의 표현 벡터
- R_{Paris}: Paris 토큰의 표현 벡터
- 예제에서는 BERT-base 모델 사용
  - 12개의 인코더 레이어
  - 12개의 어텐션 헤드
  - 768개의 은닉 유닛

  → 각 토큰의 표현 벡터 크기=768

마스크 언어 모델링(MLM)

- **각 토큰의 표현 R을 얻은 후**

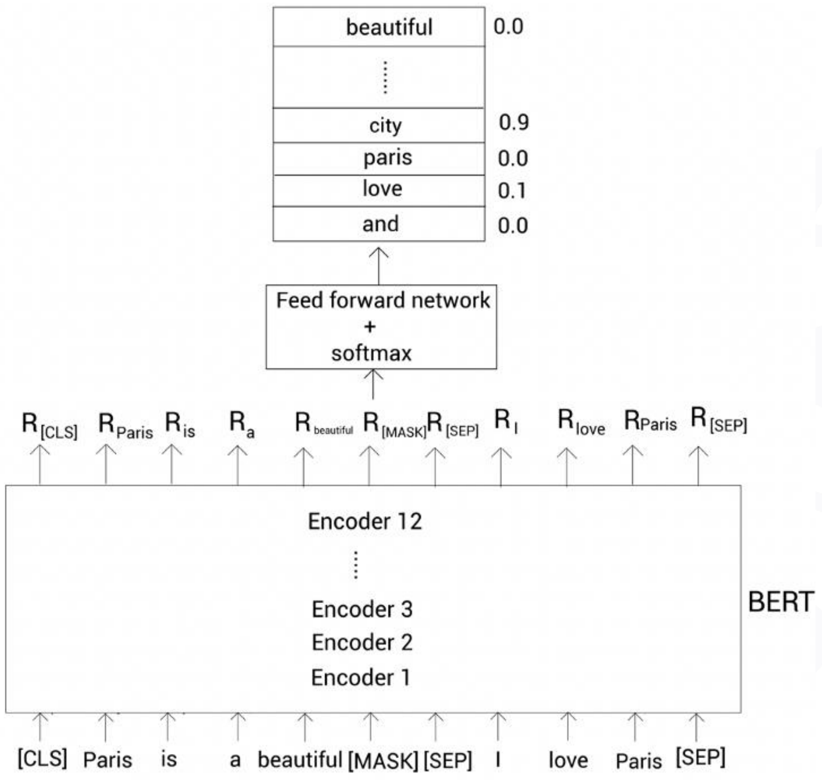


- 마스크된 토큰은 어떻게 예측하는가?

1. BERT에서 반환된 마스크된 토큰 $R_{[MASK]}$ 표현을
2. Softmax 활성화를 통해 피드포워드 네트워크(FFN)에 입력
3. FFN은 $R_{[MASK]}$단어가 마스크된 단어가 될 확률 반환

   → 그림에서는 "city"라는 단어가 마스크된 단어일

   확률이 높음 → 마스크된 단어는 "city"라고 예측



마스크 언어 모델링(MLM)

- **학습 초기의 경우**
  - BERT의 FFN 및 인코더 계층의 가중치가 최적이 아님 → 모델의 올바른 확률 반환 불가능
  - 역전파를 통한 일련의 반복 학습을 거치면서 BERT의 FFN 및 인코더 계층의 가중치 업데이트 반복
    → 최적의 가중치 학습

마스크 언어 모델링(MLM)

- **전체 단어 마스킹(Whole Word Masking, WWM)**

Let us start pretraining the model.

토큰화

tokens = [let, us, start, pre, ##train, ##ing, the, model]

[CLS], [SEP] 토큰 추가

tokens = [[CLS], let, us, start, pre, ##train, ##ing, the, model, [SEP]]

토큰의 15%를 무작위로 마스킹
→ let, ##train 마스킹

##train은 pretraining의 하위 단어

tokens = [[CLS], [MASK], us, start, pre, [MASK], ##ing, the, model, [SEP]]

WWM에서는 하위 단어가 마스킹되면
→ 해당 하위 단어와 관련된 모든 단어를
마스킹하면서 마스크 비율(15%)을 유지하려고 함

tokens = [[CLS], [MASK], us, start, [MASK], [MASK], [MASK], the, model, [SEP]]

하위 단어와 관련된 모든 단어를 마스킹하는 동안 마스크 비율(15%)을 초과하면 → 다른 단어의 마스킹 무시

tokens = [[CLS], let, us, start, [MASK], [MASK], [MASK], the, model, [SEP]]

- **마스킹 후 토큰을 BERT에 입력, 마스크된 토큰을 예측하도록 모델 학습 수행**

다음 문장 예측(NSP)

- 다음 문장 예측(Next Sentence Prediction, NSP)는
  - BERT 학습에 사용되는 Task이며,
  - 이진 분류 Task임
  - NSP Task에서는
    - BERT에 두 문장을 입력하고
    - 두 번째 문장이 첫 번째 문장의 다음 문장인지 예측함

다음 문장 예측(NSP)

- **NSP Task**

A: She cooked pasta (그녀가 파스타를 요리했다).
B: It was delicious (맛있었다).

B 문장은 A 문장의 후속 문장(이어지는 문장) → "isNext" 표시
(B 문장이 A 문장의 다음 문장임을 알 수 있게 함)

A: Turn the radio on (라디오 켜줘).
B: She bought a new hat (그녀는 새 모자를 샀다).

B 문장은 A 문장의 후속 문장(이어지는 문장)이 아님 → "notNext" 표시
(B 문장이 A 문장의 다음 문장이 아님을 알 수 있게 함)

- **NSP Task에서 모델의 목표는**
  - 문장 쌍이 isNext 범주에 속하는지 여부를 예측하는 것
    → 문장 쌍을 BERT에 입력하고 B 문장이 A 문장 다음에 오는지 여부를 예측하도록 학습
    → 모델은 B 문장이 A 문장에 이어지면 isNext 반환, 그렇지 않으면 notNext 반환
    → NSP는 본질적으로 이진 분류 태스크

다음 문장 예측(NSP)

- **NSP Task의 목적은**
  - NSP 태스크를 수행함으로써 모델은 두 문장 사이의 관계를 파악할 수 있음

    → 두 문장 간의 관계를 이해하는 것은

    → 질문-응답(QA), 감정 분류, 유사문장 탐지와 같은 하위작업(Downstream Task)에 유용
- **NSP Task를 위한 데이터셋 확보**
  - 어떠한 말뭉치에서도 데이터셋 확보 가능
  - 예시: 2개의 문서에서
    - isNext 클래스: 한 문서에서 연속된 두 문장을 isNext로 표시
    - notNext 클래스: 한 문서에서 한 문장을, 임의의 문서에서 다른 문장을 가져와 notNext로 표시
  - isNext 클래스와 notNext 클래스의 비율을 50:50으로 유지하여 클래스가 균형을 이루게 함

다음 문장 예측(NSP)

- **NSP Task를 수행하기 위한 BERT 학습 방법**

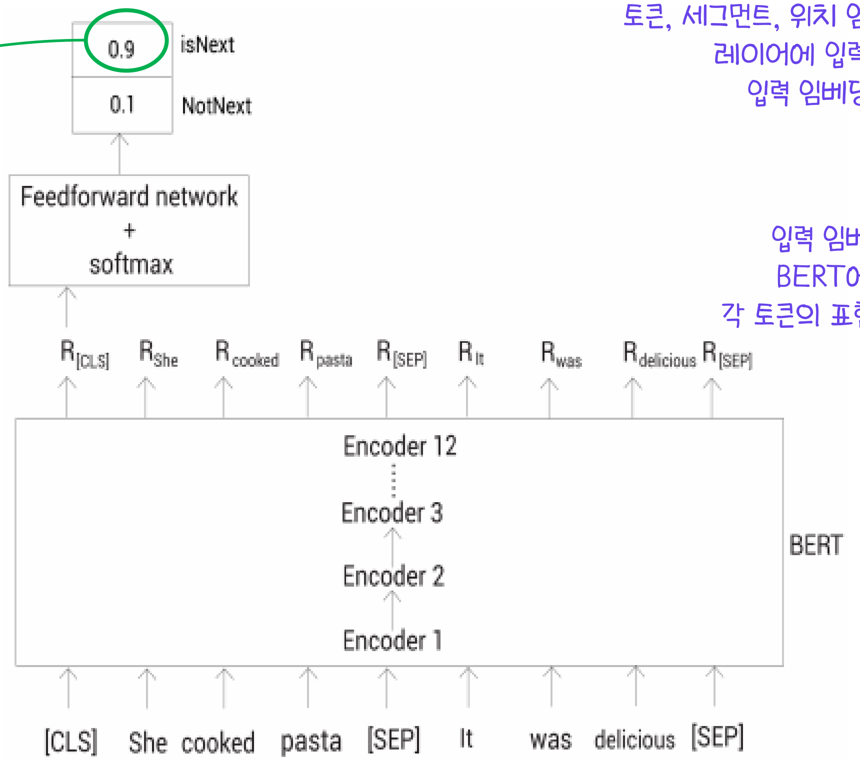
문장 쌍	레이블
She cooked pasta (그녀는 파스타를 요리했다) It was delicious (맛있었다)	isNext
Jack loves songwriting (잭은 작곡을 좋아한다) He wrote a new song (그는 새 노래를 썼다)	isNext
Birds fly in the sky (새들은 하늘을 난다) He was reading (그는 읽고 있었다)	notNext
Turn the radio on (라디오 켜줘) She bought a new hat (그녀는 새 모자를 샀다)	notNext

토큰화

tokens = [She, cooked, pasta, It, was, delicious]

[CLS], {SEP]
토큰 추가

tokens = [[CLS], She, cooked, pasta, [SEP], It, was, delicious, [SEP]]



입력 문장 쌍이
isNext 클래스에
속할 확률이 높음

토큰, 세그먼트, 위치 임베딩
레이어에 입력하고
입력 임베딩 반환

입력 임베딩을
BERT에 넣어
각 토큰의 표현 획득

다음 문장 예측(NSP)

- **NSP의 이진 분류 수행**
  - **현재 가지고 있는 데이터는 "문장 쌍에서의 각 토큰의 표현 뿐"**

    → 이러한 표현을 기반으로 문장 쌍을 어떻게 분류하는가?
  - **분류를 수행하려면…**
    - **[CLS] 토큰 표현을 가져와 Softmax 함수를 통해 FFN에 입력 → isNext/notNext의 확률값 반환**
    - **왜 [CLS] 토큰만 포함시키는가?**
      - [CLS] 토큰은 기본적으로 모든 토큰의 집계 표현을 보유하고 있음

        → 문장 전체에 대한 표현을 담고 있음

        → 따라서 다른 모든 토큰의 표현을 무시하고 [CLS] 토큰 표현인 $R_{[CLS]}$를 가져와서 확률을 반환하는 Softmax 함수를 사용해 FFN에 공급할 수 있음

다음 문장 예측(NSP)

- **학습 초기의 경우**
  - MLM과 마찬가지로...
  - BERT의 FFN 및 인코더 계층의 가중치가 최적이 아님 → 모델의 올바른 확률 반환 불가능
  - 역전파를 통한 일련의 반복 학습을 거치면서 BERT의 FFN 및 인코더 계층의 가중치 업데이트 반복
    → 최적의 가중치 학습

BERT 사전 학습의 절차

- BERT 사전 학습의 데이터셋
  - 토론토 책 말뭉치(Toronto BookCorpus) + 위키피디아 데이터셋
- 문장 샘플링
  - 말뭉치에서 두 문장(A, B) 샘플링
    - A, B 문장의 총 토큰 수의 합은 512 이하(작거나 같음)여야 함
    - 두 문장을 샘플링 할 때
      - 50%는 B 문장이 A 문장의 후속 문장이 되도록 샘플링
      - 나머지 50%는 B 문장을 A 문장의 후속 문장이 아닌 것으로 샘플링

BERT 사전 학습의 절차

- **예시**

A: We enjoyed the game (우리는 게임을 즐겼다).
B: Turn the radio on (라디오 켜줘).

tokens = [[CLS], we, enjoyed, the, game, [SEP], turn, the radio, on, [SEP]]

tokens = [[CLS], we, enjoyed, the, [MASK], [SEP], turn, the radio, on, [SEP]]

80-10-10% 규칙에 따라
토큰의 15%를 무작위로 마스킹
→ game 마스킹

- **이후, 토큰을 BERT에 입력하고**
- **마스크된 토큰을 예측하기 위해 모델 학습**
- **동시에 B 문장이 A 문장의 후속 문장인지 여부 분류**

MLM과 NSP 작업을 동시에 사용해서 BERT 학습

BERT 사전 학습의 절차

- **학습 설정**
  - **총 100만 스텝 학습**
  - **각 스텝 당 배치 크기: 256**
  - **학습률:** $lr = 1e - 4, \beta_1 = 0.9, \beta_2 = 0.999$
  - **Optimizer: Adam 사용**
  - **웜업(Warmup): 1만 스텝**
  - **Dropout 확률: 0.1 → 모든 레이어에 적용**
  - **활성화 함수: GELU(Gaussian Error Linear Unit) →** $GELU(x) = x\Phi(x)$

웜업 스텝이란?

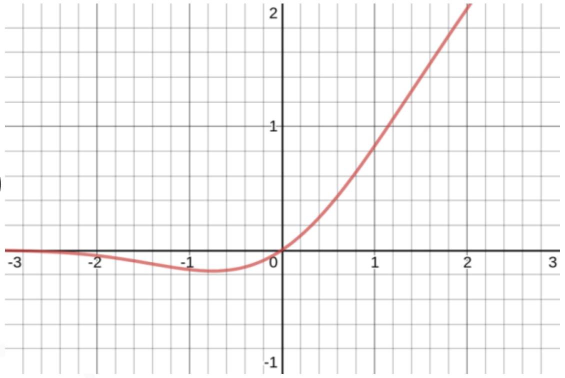
학습 스케줄링의 일부. 학습률이 1e-4 이고 웜업 스텝이 1만 스텝이라고 가정하면

초기 1만 스텝은 학습률이 0에서 1e-4로 선형 증가하는 것을 의미함

1만 스텝 후에는 수렴에 가까워 짐에 따라 학습률을 선형적으로 감소시킴

$\Phi(x)$: 표준 가우시안 누적 분포. GELU함수는 다음 수식의 근사치임

$$GELU(x) = 0.5x\left(1 + tanh\left[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)\right]\right)$$



GELU Activation Function

하위 단어 토큰화 알고리즘

- 하위 단어 토큰화
  - OOV (Out of Vocabulary) 단어의 처리에 매우 효과적이기 때문에
  - BERT, GPT-3 등 다양한 최신 자연어 모델에서 널리 사용 중

- (일반적인) 단어 수준 토큰화 과정
  - 학습 데이터셋 준비 → 학습 데이터셋에서 어휘 사전 구축
    → 데이터셋의 텍스트를 공백으로 분할 → 모든 고유 단어를 어휘 사전에 추가
    → 어휘 사전을 이용하여 입력 텍스트를 토큰화

하위 단어 토큰화 알고리즘

- **단어 수준 토큰화의 예시**
  - 어휘 사전: vocabulary = [game, the, I, played, walked, enjoy]
  - 입력 문장: I played the game
  - 입력 문장에서 단어 확보: [I, played, the, game] → 어휘 사전에 단어가 있는지 확인
  - 모든 단어가 있으면 주어진 문장에 대한 최종 토큰은 tokens=[I, played, the, game]
  - 입력 문장: I enjoyed the game → [I, enjoyed, the, game] → enjoyed가 어휘사전에 없음
  - 없는 단어가 있으면 주어진 문장에 대한 최종 토큰은 tokens=[I, <UNK>, the, game]

OOV문제를 피하기위해 없는 단어를 무작정 추가하여 어휘사전의 크기를 키우면
메모리 부족, 성능 문제 야기, 또한 그렇게 추가해도 <UNK>는 여전히 존재

→ 하위 토큰화 알고리즘 도입으로 해결 가능

하위 단어 토큰화 알고리즘

- 하위 단어 토큰화의 동작
  - 어휘 사전: vocabulary = [game, the, I, played, walked, enjoy]
  - 하위 단어 토큰화에서는 단어를 하위 단어로 분할
    - played → [play, ed], walked → [walk, ed]
  - 하위 단어로 분할 후, 어휘 사전에 추가
    - vocabulary = [game, the, I, play, walk, ed, enjoy]
  - 입력문장: I enjoyed the game → [I, enjoyed, the, game] → enjoyed 없음
    → enjoyed를 하위 단어로 분할 → [enjoy, ed] → 어휘 사전에 있음
    → token = [I, enjoy, ##ed, the, game]

하위 단어 토큰화 알고리즘

- **## (해시 기호): 하위 단어라는 표시**
  - ##ed : 하위 단어.
  - enjoy에는 왜 ##을 붙이지 않는가?
    - 단어의 시작 부분에 해당하는 하위 단어에는 추가하지 않음
    - ## 기호는 하위 단어이고 그 앞에 다른 단어가 있음을 나타내기 위해 추가됨
- **어휘 사전의 다른 단어들은 왜 나누지 않는가? 분할할 단어와 분할하지 않을 단어는 어떻게 결정하는가?**

  → 하위 단어 토큰화 알고리즘에서 정의됨

하위 단어 토큰화 알고리즘

- 어휘 사전 생성에 사용되는 하위 단어 토큰화 알고리즘
  - 바이트 쌍 인코딩 (Byte Pair Encoding)
  - 바이트 수준 바이트 쌍 인코딩 (Byte-level Byte Pair Encoding)
  - 워드피스 (WordPiece)

하위 단어 토큰화 알고리즘

- 바이트 쌍 인코딩 (Byte Pair Encoding, BPE)
  - 데이터셋에서 빈도수와 함께 추출된 단어가
    - (cost, 2), (best, 2), (menu, 1), (men, 1), (camel, 1) 이라고 가정하면
  - 모은 단어를 문자로 나누고 문자 시퀀스로 만듦 →
  - 어휘 사전 크기 정의 → 크기 14로 가정

    : 14개의 토큰으로만 어휘 사전을 생성함을 의미 →

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u

어휘사전 → 크기가 11

하위 단어 토큰화 알고리즘

- **어휘 사전에 새 토큰을 추가하려면**
  - 먼저 가장 빈도수가 큰 기호 쌍을 식별
  - 가장 빈번한 기호 쌍을 병합해 어휘 사전에 추가 → 어휘 사전 크기에 도달할 때까지 반복

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u, st

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u, st

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u, st, me

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u, st, me, men

vocabulary ={a, b, c, e, l, m, n, o, s, t, u, st, me, men}

하위 단어 토큰화 알고리즘

- **BPE 수행 단계**

1. 빈도수와 함께 주어진 데이터셋에서 단어 추출
2. 어휘 사전 크기 정의
3. 단어를 문자 시퀀스로 분할
4. 문자 시퀀스의 모든 고유 문자를 어휘 사전에 추가
5. 빈도가 높은 기호 쌍을 선택하고 병합
6. 어휘 사전 크기에 도달할 때까지 앞 다섯 단계 반복

하위 단어 토큰화 알고리즘

- **BPE로 토큰화 하기**

vocabulary ={a, b, c, e, l, m, n, o, s, t, u, st, me, men}

  - **이 어휘 사전을 어떻게 사용할 것인가?**
    - **가정: 입력 텍스트는 mean 이라는 한 단어로만 구성되어 있다.**
    - **어휘 사전에서 “mean”이라는 단어가 있는지 확인한다. → 없음**
    - **“mean”을 분할한다 → “me”, ”an”**
    - **하위 단어가 어휘 사전에 있는지 확인한다. → “me”는 있지만 “an”은 없음**
    - **하위 단어 “an”을 분할한다 → “a”, “n”**
    - **하위 단어가 어휘 사전에 있는지 확인한다. → 모두 있음**
    - **최종 토큰은** tokens =[me, a, n]

하위 단어 토큰화 알고리즘

- 새로운 가정: 입력 텍스트는 bear라는 한 단어로만 구성되어 있다.
- 어휘 사전에서 “bear”라는 단어가 있는지 확인한다. → 없음
- “bear”를 분할한다 → “be”, ”ar”
- 하위 단어가 어휘 사전에 있는지 확인한다. → “be”는 있지만 “ar”은 없음
- 하위 단어 “ar”을 분할한다 → “a”, “r”
- 하위 단어가 어휘 사전에 있는지 확인한다. → “a”는 있지만 “r”은 없음
- 개별 문자 수준까지 내려왔기때문에 더이상 분할할 수 없음 → “r”은 <UNK> 토큰으로 교체
- 최종 토큰은 tokens =[be, a, <UNK>]

- 그런데 BPE는 OOV 문제를 잘 해결한다고 했는데…

  → 예시의 어휘사전이 작아서 그럼. 원래의 큰 어휘사전에는 거의 다 있음

  - men 이라는 단어가 입력된다면 → 어휘사전에 있음 → 최종 토큰은 tokens =[men]

하위 단어 토큰화 알고리즘

- **바이트 수준 바이트 쌍 인코딩 (Byte-level Byte Pair Encoding, BBPE)**
  - **BPE와 유사하지만 문자 수준 시퀀스 대신 바이트 수준의 시퀀스를 사용함**
  - **예시**
    - 가정: 입력 텍스트는 "best"라는 단어로만 구성되어 있다.
    - BPE에서는 단어를 문자 시퀀스로 변환: best
    - BBPE에서는 단어를 byte 시퀀스로 변환: best → 62 65 73 74

      각 유니코드 문자는 2바이트로 변환되므로 단일문자는 1~4바이트까지 될 수 있음
      - 한자 단어 예시: 你好 → e4 bd a0 e5 a5 bd
  - **BBPE → 다국어 설정에서 매우 유용하다. (특히 다국어 OOV 단어처리)**

하위 단어 토큰화 알고리즘

- 워드피스 (WordPiece)
  - BPE와 유사하게 동작하지만
    - BPE
      1. 주어진 데이터셋에서 먼저 단어의 빈도수를 추출하고
      2. 단어를 문자 시퀀스로 분할
      3. 빈도수가 높은 기호 쌍을 병합
      4. 어휘 사전 크기에 도달할 때까지 반복적으로 고빈도 기호 쌍 병합
    - WordPiece
      - 빈도에 따라 기호 쌍을 병합하지 않고 가능도(likelihood)를 기준으로 기호 쌍을 병합
      - 따라서 주어진 학습 데이터에 대해 학습된 언어 모델 가능도가 높은 기호 쌍을 병합

하위 단어 토큰화 알고리즘

- **예시**
  - BPE에서는 s, t가 4번 발생했기 때문에 병합되었으나
  - WordPiece에서는 빈도가 아닌 가능도에 따라 병합함
    1. 먼저 모든 기호 쌍에 대해 언어모델(주어진 학습 세트에서 학습된)의 가능도 확인
    2. 가능도가 가장 높은 기호 쌍 병합: $\frac{p(st)}{p(s)p(t)}$
    3. 가능도가 높으면 기호 쌍을 병합하고 어휘 사전에 추가
    4. 이런 식으로 모든 기호 쌍의 가능도를 계산하고 최대 가능도를 가진 것을 병합하여 어휘 사전에 추가

문자 시퀀스	빈도수
cost	2
best	2
menu	1
men	1
camel	1

a, b, c, e, l, m, n, o, s, t, u

- **WordPiece 수행 단계**
  1. 빈도수와 함께 주어진 데이터셋에서 단어 추출
  2. 어휘 사전 크기 정의
  3. 단어를 문자 시퀀스로 분할
  4. 문자 시퀀스의 모든 고유 문자를 어휘 사전에 추가
  5. 주어진 데이터셋(학습셋)에서 언어 모델 빌드
  6. 학습셋에서 학습된 언어 모델의 최대 가능도를 가진 기호 쌍을 선택하고 병합
  7. 어휘 사전 크기에 도달할 때까지 앞 여섯 단계 반복

하위 단어 토큰화 알고리즘

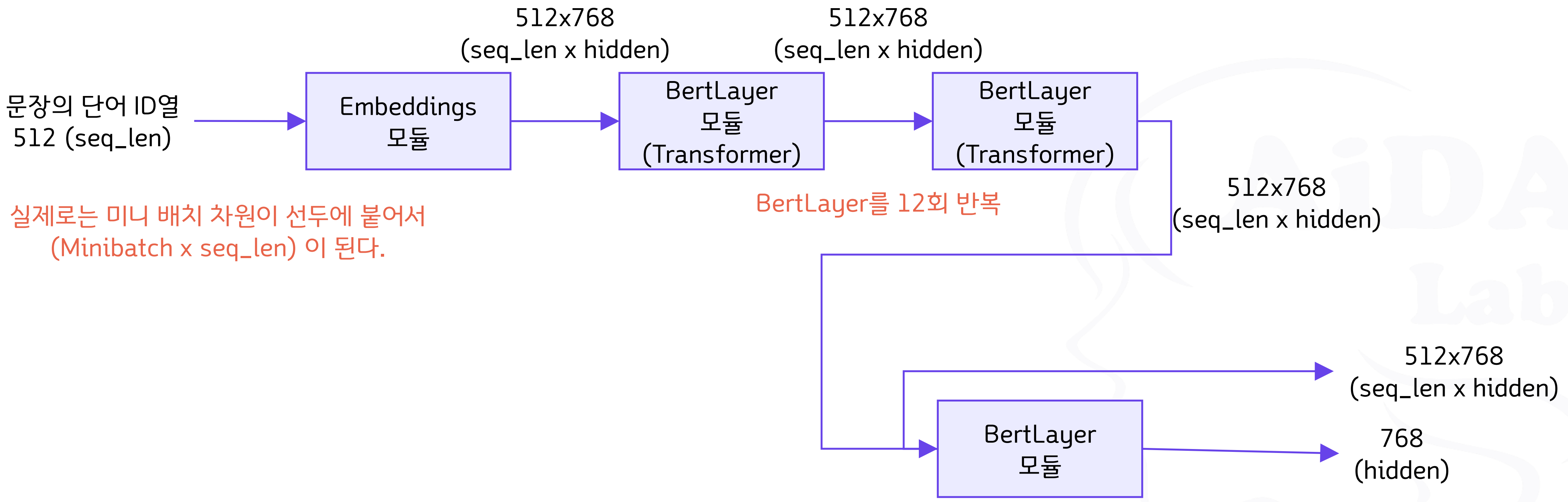
- **WordPiece로 토큰화 하기**
  - 어휘사전 vocabulary ={a, b, c, e, l, m, o, s, t, u, st}
  - 가정: 입력 텍스트가 하나의 단어 “stem”으로만 구성되어 있다.
  - 어휘사전 검색 → “stem” 없음 → 하위 단어 분할 → [st, ##em]
  - 어휘사전 검색 → “st”는 있지만 “em”은 없음 → 하위 단어 “em” 분할 → [st, ##e, ##m]
  - 어휘사전 검색 → e, m 모두 있음
  - 최종 토큰: tokens = [st, ##e, ##m]

  - 이런 식으로 WordPiece 하위 단어 토큰화 알고리즘을 사용하여 어휘 사전 생성, 토큰화에 어휘 사전 활용 등을 수행할 수 있음

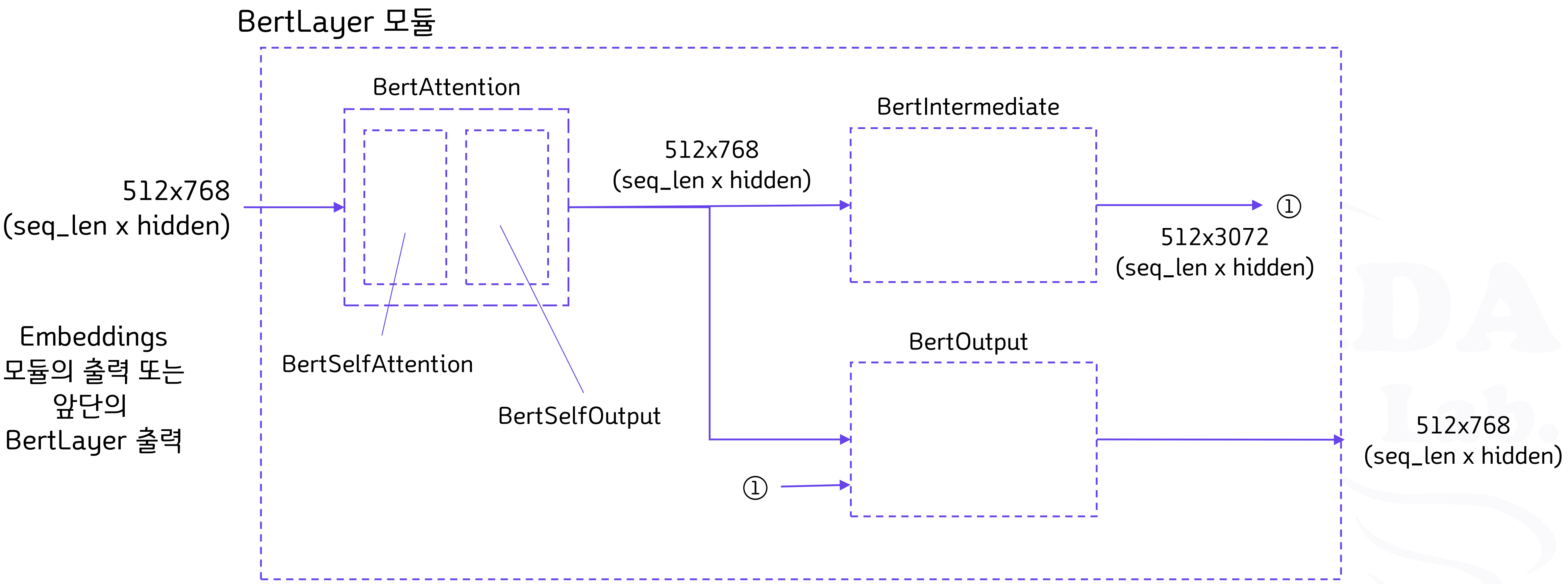
BERT 모델의 프로세스

BERT 모델의 구성

- **BERT-Base 모델 기준으로 학습**
- **BERT 모델 구조**



BertLayer 모듈 구성



- **BERT 모델 프로세스**
  - 문장을 단어 ID로 하는 ID열(길이는 seq_len=512) → 입력 → Embeddings 모듈 전달
  - Embeddings 모듈은 ID 열을 단어의 특징량 벡터로 변환

    → 단어와 특징량 벡터의 위치 정보를 나타내는 Positional Embedding 추가

    : BERT-Base에서 사용하는 특징량 벡터의 차원 수는 768

    → (모델 구조도에서 hidden으로 표시)

BERT 모델 구조

- Embedding 모듈의 출력 텐서인 (seq_len x hidden) = 512 x 768 → BertLayer로 전달
- BertLayer 모듈: Self-Attention을 이용하여 특징량 변환 수행 → 모듈을 총 12회 반복: 출력 텐서 크기는 입력 텐서와 같은 512 x 768
- 12회 반복된 BertLayer 모듈의 출력 텐서(512 x 768(에서 첫 단어의 특징량(1 x 768)을 → BertPooler 모듈에 입력

BERT 모델 구조

- 출력 텐서의 첫 단어를 [CLS]로 설정 → 문장의 클래스 분류 등에 사용하기 위한 입력 문장 전체의 특징량을 가지는 부분으로 활용
- 선두 단어의 특징량을 BertPooler 모듈로 변환
- 최종 출력 텐서 (2가지)
  - 12회의 BertLayer 모듈에서 출력된 (seq_len x hidden)=(512x768) 텐서
  - 선두 단어 [CLS]의 특징량(BertPooler 모듈의 출력)인 크기 768의 텐서

BERT 모델의 사전 작업

- BERT는 네트워크 모델을 두 종류의 언어 작업으로 사전 학습함
  - MLM (Masked Language Model)
  - NSP (Next Sentence Prediction)

BERT 모델의 사전 작업

- MLM
  - CBOW 모델의 확장판 작업
  - CBOW 모델: 문장 중 한 단어를 마스크하여 알 수 없게 하고, 마스크 단어의 앞뒤(약 5단어씩) 정보로 마스크된 단어를 추정하는 모델
  - 입력의 512 단어 중 여러 단어를 마스크하여 마스크된 단어 앞뒤 몇 단어를 지정하지 않고 마스크되지 않은 단어 모두를 사용하여 마스크된 단어를 추정함으로써 해당 단어의 특징량 벡터를 획득하는 작업

BERT 모델의 사전 작업

- NSP
  - BERT 모델은 사전 학습에서 두 개의 텍스트 데이터를 입력함 (512 단어로 두 문장 구성)
  - 두 문장은 [SEP]로 구분되며 지도 데이터 내에서 두 개의 패턴으로 준비됨
    - 연속적으로 존재하며 의미 있고 관계가 깊은 문장
    - 전혀 관계가 없고 문맥의 연결이 없는 두 문장
  - BertPooler 모듈에서 출력된 선두 단어 [CLS]의 특징량으로 입력된 두 개의 문장이 어떤 패턴인지 추론

BERT 모델의 사전 작업

- **BERT 모델의 두 종류의 사전 학습의 예**
  - **MLM**

    일부 단어가 마스크 된 문장을 입력하고 해당 단어가 Vocabulary의 어떤 낱말인지 맞히기

    (문장) [CLS] I accessed the 마스크 account [SEP] We play soccer at the bank of the 마스크 [SEP]

    답: bank 답: river

  - **NSP**

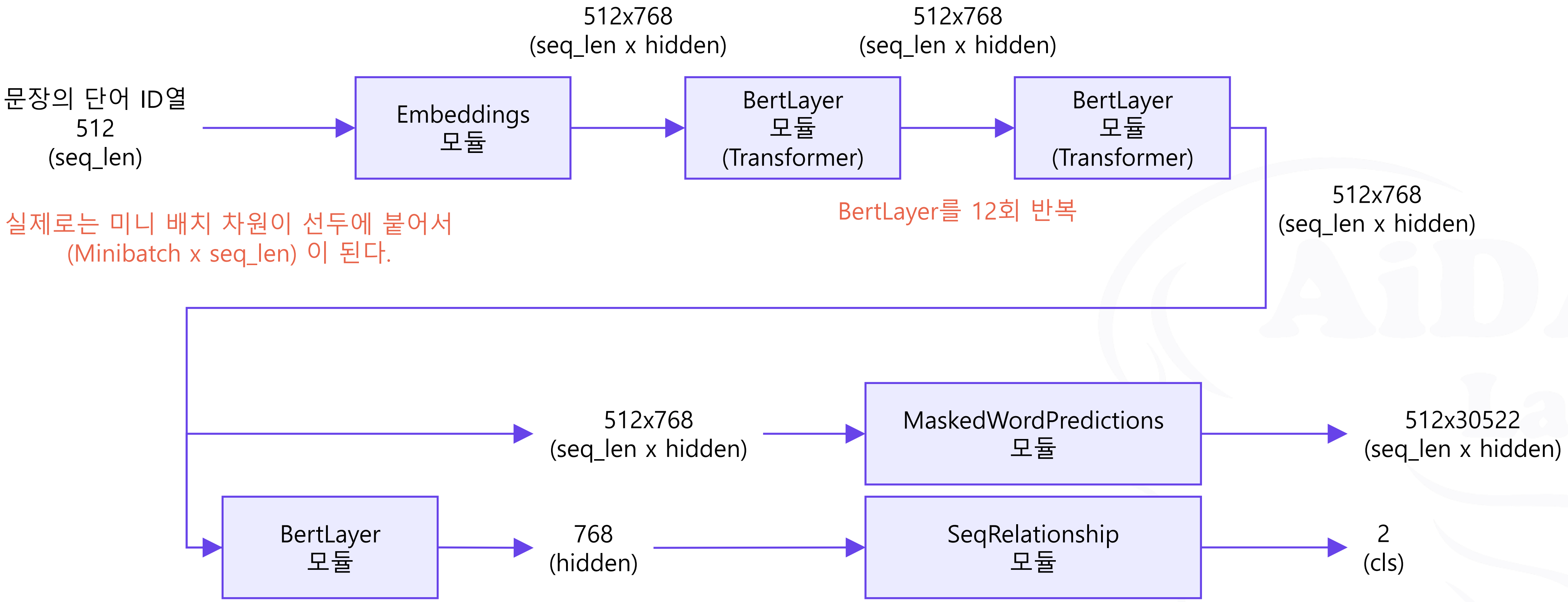
    두 문장을 입력하고 두 문장의 의미가 연결되어 있는지 맞히기

    (문장) [CLS] I accessed the bank account [SEP] We play soccer at the bank of the river. [SEP]

    답은 관계성이 없음

BERT 모델의 사전 작업

- **사전 학습을 실시하는 BERT 모델 구조**



BERT 모델의 사전 작업

- 두 언어 작업을 해결하기 위하여 모듈 연결
  - 기본 모델에 MaskedWordPredictions 모듈과 SeqRelationship 모듈을 붙여 두 종류의 사전 작업인 MLM과 NSP를 잘 수행할 수 있도록 기본 모델 학습

BERT 모델의 사전 작업

- MaskedWordPredictions 모듈
  - BertLayer 출력(seq_len x hidden)=(512x768)을 입력하고 (seq_len x vocab_size) = (512x30,522) 출력
  - vocab_size (30,522)는 BERT의 vocabulary 전체의 단어 수(영어의 경우)
  - 입력된 512 단어가 전체 vocabulary 단어의 어느 것인지 (512x30522)에 대하여 소프트맥스 함수를 계산하여 도출
  - 실제로 추정하는 것은 입력 단어 512개 전체가 아닌 마스크된 알 수 없는 단어 뿐

BERT 모델의 사전 작업

- SeqRelationship 모듈
  - BertPooler 모듈에서 출력된 선두 단어 [CLS]의 특징량 벡터를 전결합층에 입력하여 클래스의 수가 2인 분류를 실행
  - 전 결합층의 출력 크기 2 → 아래의 2 패턴 중 어느 쪽인지 판정하기 위함
    - 연속적으로 존재하며 의미 있고 관계가 깊은 문장
    - 전혀 관계가 없고 문맥의 연결이 없는 두 문장

BERT의 특징

- BERT의 세가지 특징
  - 문맥에 의존한 단어 벡터 표현을 만들 수 있게 되었다.
  - 자연어 처리 작업에서 파인 튜닝이 가능해 졌다.
  - Attention에 의한 설명과 시각화가 간편해 졌다.

BERT의 특징

- 문맥에 의존한 단어 벡터 표현을 만들 수 있게 되었다.
  - 어떤 언어라도 단어의 의미가 단 하나인 경우는 적음
    - (예) bank: 은행, 강변 이라는 의미가 있음
  - 다양한 의미를 가진 각 단어들은 문맥에 따라 단어의 의미가 바뀜
  - BERT는 문맥에 맞는 단어의 벡터 표현이 가능함

BERT의 특징

- **BERT는 12단 Transformer를 사용**
  - Embedding 모듈에서 단어ID를 단어 벡터로 변환할 때는 은행의 bank와 강변의 bank는 동일한 길이(768)의 단어 벡터
  - 12단의 Transformer를 거치는 동안 단어 bank의 위치에 있는 특징량 벡터는 변화함
  - 변화 결과, 12단 째의 출력인 단어, bank의 위치에 있는 특징량 벡터는 최종적으로 은행 bank와 강변 bank가 서로 다른 벡터가 됨
  - 여기서 말하는 특징량 벡터란, 사전 학습의 MLM이 풀어놓은 특징량 벡터
  - 문장 중의 단어 bank와 그 주변 단어와의 관계성을 바탕으로 하여 Transformer의 Self-Attention 처리로 작성됨
  - 동일한 단어도 주변 단어와의 관계성에 따라 문맥에 맞는 단어 벡터가 생성됨

BERT의 특징

- 자연어 처리 작업에서 파인 튜닝이 가능해 졌다.
  - BERT를 기반으로 다양한 자연어 처리를 수행하려면
    - 두 언어 작업에서 사전 학습한 가중치 파라미터를 BERT 모델의 가중치로 설정
    - BERT 모델 구조 그림에서 나타낸 (seq_len x hidden)=(512x768) 텐서와 (hidden)=(768)의 두 텐서를 출력
    - 두 텐서를 실행하고 싶은 자연어 처리 작업에 맞춘 어댑터 모듈에 투입
    - 작업에 따른 출력 획득
    - (예) 긍정적/부정적 감정 분석의 경우, 어댑터 모듈로 하나의 전결합층을 추가하는 것 만으로 문자의 판정이 가능해짐

BERT의 특징

- 모델을 학습할 때, 기반이 되는 BERT와 어댑터 모듈의 전결합층 양쪽 모두를 파인 튜닝으로 학습
- BERT의 출력에 어댑터 모듈을 연결하여 다양한 자연어 처리 작업 수행 가능
- Object Detection을 위한 SSD 모델, 자세 추정을 위한 OpenPose 모델에서 사용된 기반 네트워크인 VGG와 같은 역할을 BERT가 수행
- 적은 문서 데이터로도 성능 좋은 모델의 작성이 가능함
- 자연어 처리 작업도 화상 작업처럼 전이학습 및 파인 튜닝을 적용할 수 있게 된 것이 BERT가 주목받은 요인의 하나임

BERT의 특징

- BERT는 어떻게 화상 작업의 기본 모델인 VGG와 같은 전이학습 및 파인 튜닝의 기반 역할을 수행할 수 있을까?
  - VGG 모델과 같이 화상 처리에서 화상 분류가 가능한 네트워크는 물체 감지나 시맨틱 분할에도 유효함
  - BERT도 사전작업 MLM을 풀 수 있는 단어를 문맥에 맞는 특징량 벡터로 변환할 수 있는 능력이 단어의 의미를 정확하게 파악할 수 있게 함
  - 사전작업 NSP로 문장이 의미 있게 연결되었는지 여부를 판정할 수 있는 능력이 문장의 의미를 이해할 수 있게 함
  - 단어와 문장의 의미를 이해할 수 있도록 사전학습을 하고 있으므로 자연어 처리 작업인 감정 분석 등에도 응용이 가능해짐

BERT의 특징

- 선구적인 범용 언어 모델의 사례를 만듦
  - 단어와 문장의 의미를 제대로 파악해야 하는 사전 작업의 수행
  - 사전 작업으로 학습한 가중치를 기반으로, 어댑터를 자연어 처리 작업에 맞게 교체하여 파인 튜닝을 수행

→ 이러한 처리의 흐름이 자연어 처리에서의 하나의 표준이 될 것으로 기대됨

BERT의 특징

- Attention에 의한 설명과 시각화가 간편해 졌다.
  - Attention: 결과에 영향을 준 단어의 위치정보
  - Attention을 시각화 함으로써 인간이 추론 결과를 설명하기가 쉬워짐

BERT 모델 활용하기

BERT 활용하기

- 사전 학습된 BERT를 사용하는 방법을 알아보자
  - 사전 학습된 BERT 모델의 구성 살펴보기
  - 사전 학습된 BERT 모델을 특징 추출기로 사용하는 방법
  - 허깅페이스의 트랜스포머 라이브러리 살펴보기
  - 사전 학습된 BERT 모델에서 임베딩 추출하기
  - BERT 모델의 모든 인코더 레이어에서 임베딩 추출하기
  - 하위 작업을 위해 사전 학습된 BERT 모델을 파인 튜닝 하는 방법

사전 학습된 BERT 모델 탐색

- 이전 수업 내용
  - MLM, NSP를 사용하여 BERT를 사전 학습 시키는 방법

    → 그러나 BERT를 처음부터 사전 학습 시키는 것은 많은 계산 비용이 요구됨

    → 사전 학습된 공개 BERT 모델을 이용하는 것이 효과적

  - Google에서 제공하는 BERT 모델을 찾아서 활용하자

사전 학습된 BERT 모델 탐색

- Google에서 제공하는 사전 학습된 BERT 모델
  - https://github.com/google-research/bert
    - 해당 URL에서 제공하는 오른쪽과 같은 테이블에서 원하는 각 모델을 선택하여 사전 학습된 BERT 모델을 다운로드
    - Cased/Uncased 구분된 모델도 제공
      - Cased: 토큰에 대소문자가 모두 존재하는 모델
      - Uncased: 토큰에 대하여 소문자화를 수행한 모델
    - WWM(Whole Word Masking) 방법으로 학습된 모델도 제공

	H=128	H=256	H=512	H=768
L=2	2/128 (BERT-Tiny)	2/256	2/512	2/768
L=4	4/128	4/256 (BERT-Mini)	4/512 (BERT-Small)	4/768
L=6	6/128	6/256	6/512	6/768
L=8	8/128	8/256	8/512 (BERT-Medium)	8/768
L=10	10/128	10/256	10/512	10/768
L=12	12/128	12/256	12/512	12/768 (BERT-Base)

L: 인코더 레이어 수
H: 은닉 유닛의 크기 (표현의 크기)

사전 학습된 BERT 모델 탐색

- 사전 학습된 모델의 활용법
  - 임베딩을 추출해서 특징 추출기로 사용
  - 사전 학습된 BERT 모델을 텍스트 분류, 질문-응답 등과 같은 하위 작업에 맞게 파인 튜닝해서 사용

사전 학습된 BERT에서 임베딩 추출하기

- 다음의 문장에서 각 단어의 문맥 임베딩을 추출해보자
  - 입력 문장: I love Paris (나는 파리를 사랑한다).

- 문장에서 각 단어의 문맥 임베딩을 추출하려면
  1. 문장을 토큰화하고
  2. 사전 학습된 BERT에 토큰을 입력하여
  3. 토큰에 대한 임베딩을 반환

토큰 수준(단어 수준) 표현 외에도
문장 수준의 표현을 얻을 수도 있음

사전 학습된 BERT에서 임베딩 추출하기

- 사전 학습된 BERT에서 단어 수준 및 문장 수준 임베딩 추출하기
  - 감정 분석 작업을 위한 데이터셋 예시

문장	레이블
I love Paris	1
Sam hated the movie	0
It was a great day	1
The song is not good	0
...	...
We loved the game	1

1: 긍정적인 감정
0: 부정적인 감정

- 데이터셋은 텍스트 데이터 → 모델에 직접 입력 불가
  → 벡터화가 우선 되어야 함

- 벡터화 방안
  - TF-IDF, Word2Vec 등 다양한 방법 사용 가능(문맥 독립적)
  - BERT 모델을 이용한 벡터화(문맥 기반 임베딩)

사전 학습된 BERT에서 임베딩 추출하기

- **BERT 모델을 이용한 벡터화 방법**

I love Paris

1. **워드피스 토크나이저로 문장 토큰화**

tokens=[I, love, Paris] → tokens=[[CLS], I, love, Paris, [SEP]]

2. **학습셋의 모든 문장을 토큰화**
3. **그런데 모든 토큰의 길이를 동일하게 유지해야 함 → 토큰 길이를 7로 유지**

tokens=[[CLS], I, love, Paris, [SEP], [PAD], [PAD]]

사전 학습된 BERT에서 임베딩 추출하기

4. [PAD] 토큰은 실제 토큰이 아니라는 것을 모델에게 이해시키기

→ 어텐션 마스크 적용

```
attention_mask=[ 1, 1, 1, 1, 1, 0, 0 ]
```

[PAD] 토큰이 있는 위치에만 "0" 설정

5. 모든 토큰을 고유한 토큰 ID에 매핑

```
token_ids=[ 101, 1045, 2293, 3000, 102, 0, 0 ]
```

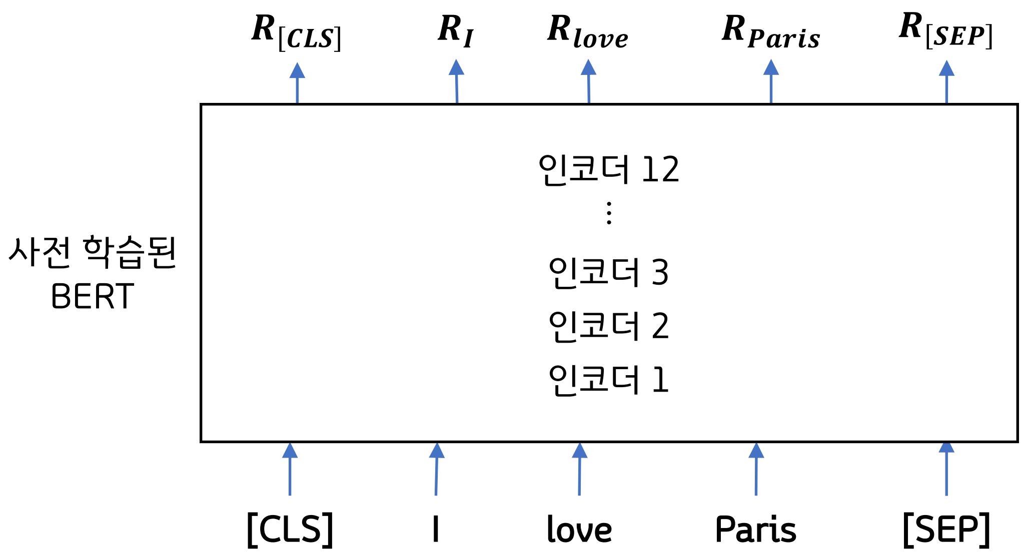
id 값은 임의 값이므로 신경쓰지 말것

- ID 101: [CLS]
- ID 1045: I
- ID 3000: Paris
- ID 102: [SEP]

어텐션 마스크, token_ids를 사전 학습된 BERT 모델에 입력

→ 각 토큰의 벡터 표현(임베딩) 획득

사전 학습된 BERT에서 임베딩 추출하기



- 토큰을 입력으로 공급하면
- 인코더 1은 모든 토큰의 표현을 계산해서 인코더 2로 보냄
- 인코더 2는 인코더 1이 계산한 표현을 입력으로 가져와서 다음 인코더인 인코더 3으로 전송
- ... 반복 ...
- 최종 인코더인 인코더 12는 문장에 있는 모든 토큰의 최종 표현 벡터(임베딩)를 반환

- 사전 학습된 BERT-base 모델을 사용한다면 각 토큰의 표현 크기는 768

사전 학습된 BERT에서 임베딩 추출하기

- 전체 문장의 표현은 어떻게 얻을 수 있나?
  - 문장 시작 부분에 있는 [CLS] 토큰의 표현은 전체 문장의 집계 표현을 보유
    → 다른 모든 토큰의 임베딩을 무시하고 [CLS] 토큰의 임베딩을 가져와서 문장의 표현으로 할당할 수 있음
  - "I love Paris" 문장의 표현은 [CLS] 토큰에 해당하는 $R_{[CLS]}$의 표현 벡터
  - 유사한 방식으로 학습셋의 모든 문장의 벡터 표현 계산 가능
  - 학습셋의 모든 문장의 문장 표현을 얻은 후 → 해당 표현을 입력으로 제공
    → 분류기 학습 → 감정 분석 작업 수행

사전 학습된 BERT에서 임베딩 추출하기

- 문장의 표현을 얻는 효율적인 방법
  - [CLS] 토큰의 표현을 문장 표현으로 사용하는 것은 항상 좋은 방식은 아님.
    적절하지 않은 경우도 있음
  - 문장의 표현을 얻는 효율적인 방법은
    → 모든 토큰의 표현을 평균화 또는 풀링하는 것

허깅페이스 트랜스포머

- 허깅페이스(Hugging Face)
  - 자연어 기술의 민주화(?)를 추구하는 조직
  - 오픈 소스 트랜스포머 라이브러리 → 자연어 처리 커뮤니티에서 많은 인기
- 허깅페이스 트랜스포머 라이브러리
  - 라이브러리 내에 100개 이상의 언어로 사전 학습된 수천 개의 모델 포함
  - 파이토치, 텐서플로 모두와 호환됨

BERT 모델 활용 예시

질문-응답 시스템

- 질문-응답 태스크에서는
  - 데이터셋: 질문에 대한 응답이 포함된 단락과 함께 질문이 제공됨
  - 태스크의 목표:주어진 질문에 대한 단락에서 답을 추출하는 것
  - BERT의 입력: 질문-단락 쌍
    - BERT에 질문과 응답을 담은 단락을 입력하고
    - 단락에서 응답을 추출해야 함
  - BERT의 출력
    - 단락에서 응답에 해당하는 텍스트의 범위 반환

질문-응답 시스템

- **예시**

질문	면역 체계는 무엇입니까?
단락	면역 체계는 질병으로부터 보호하는 유기체 내의 다양한 생물학적 구조와 과정의 시스템입니다. 제대로 기능하려면 면역 체계가 바이러스에서 기생충에 이르기까지 병원균으로 알려진 다양한 물질을 탐지하고 유기체의 건강한 조직과 구별해야 합니다.
응답	질병으로부터 보호하는 유기체 내의 다양한 생물학적 구조와 과정의 시스템입니다.

- **해결 방안**
  - **모델: 주어진 단락의 답을 포함하는 텍스트 범위의 시작과 끝의 인덱스를 이해해야 함**

면역 체계는 질병으로부터 보호하는 유기체 내의 다양한 생물학적 구조와 과정의 시스템입니다.
제대로 기능하려면 면역 체계가 바이러스에서 기생충에 이르기까지 병원균으로 알려진 다양한 물질을 탐지하고 유기체의 건강한 조직과 구별해야 합니다.

질문-응답 시스템

- 고민해야 할 것은?
  - 답을 포함하는 텍스트 범위의 시작과 끝의 인덱스를 어떻게 찾을까?
  - 단락 내 답의 시작과 끝 토큰(단어)의 확률을 구하면 쉽게 답을 추출할 수 있을 것인가?
  - 어떻게 하면 이렇게 동작하게 할 수 있는가?

질문-응답 시스템

- **방안**
  - 시작 벡터 S와 끝 벡터 E라는 2개의 벡터를 사용. 시작 및 끝 벡터의 값은 학습이 되는 값
  - 단락 내 각 토큰이 응답의 시작 토큰이 될 확률 계산

$$P_i = \frac{e^{S \cdot R_i}}{\sum_j e^{S \cdot R_i}}$$

- 각 토큰 i 에 대하여 R_i 토큰 표현 벡터와 시작 벡터 S 간의 내적을 계산한다.
- 내적 $S \cdot R_i$에 $softmax$ 함수를 적용하고 확률 획득

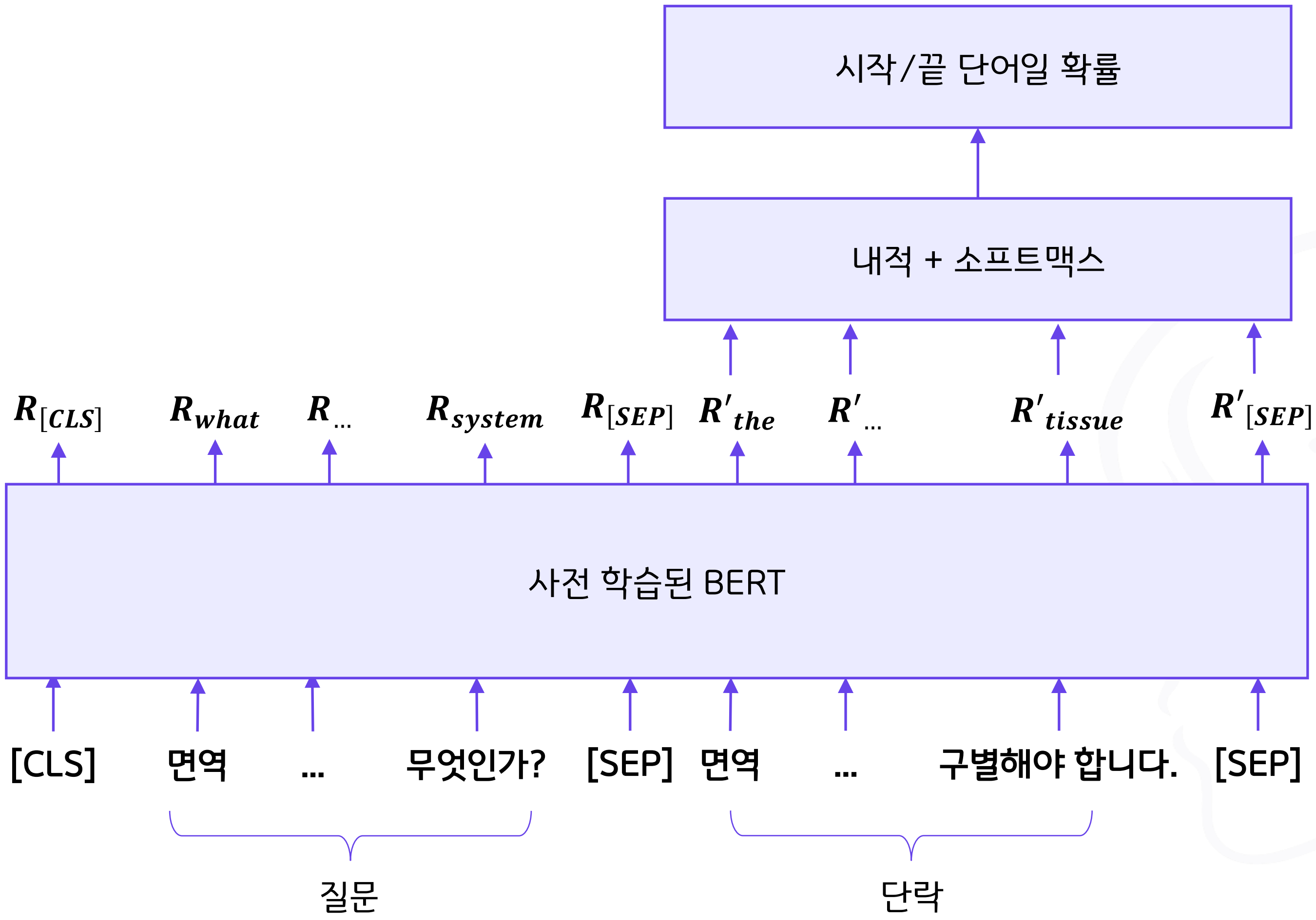
  - 다음으로 시작 토큰이 될 확률이 높은 토큰의 인덱스를 선택하여 시작 인덱스 계산
  - 유사한 방식으로 단락의 각 토큰이 응답의 끝 토큰이 될 확률을 계산

$$P_i = \frac{e^{E \cdot R_i}}{\sum_j e^{E \cdot R_i}}$$

- 각 토큰 i 에 대하여 R_i 토큰 표현 벡터와 끝 벡터 E 간의 내적을 계산한다.
- 내적 $E \cdot R_i$에 $softmax$ 함수를 적용하고 확률 획득

  - 다음으로 끝 토큰이 될 확률이 높은 토큰의 인덱스를 선택하여 끝 인덱스 계산

- 질문-응답을 위한 사전 학습된 BERT 파인 튜닝



- **자연어 추론(Natural Language Inference, NLI)**
  - **NLI에서 모델은**
    - 가정이
    - 주어진 전제에 대하여 참인지 거짓인지 중립인지 여부를
    - 결정하는 태스크
  - **모델의 목표**
    - 주어진 문장 쌍(전제, 가설 쌍)에 대해서 참, 거짓, 중립 여부를 결정하는 것

- **샘플 NLI 데이터**

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

전제: He is playing (그는 놀고 있다).

가설: He is sleeping (그는 자고 있다).

tokens=[[CLS], He is playing [SEP], He is sleeping , [SEP]]

- **사전 학습된 BERT에 토큰 입력, 각 토큰의 임베딩 획득**
- **[CLS] 토큰의 $R_{[CLS]}$ 표현 벡터를 가져와 분류기(FFN, Softmax)에 입력**
- **분류기는 문장이 참, 거짓, 중립일 확률 반환**

개체명 인식

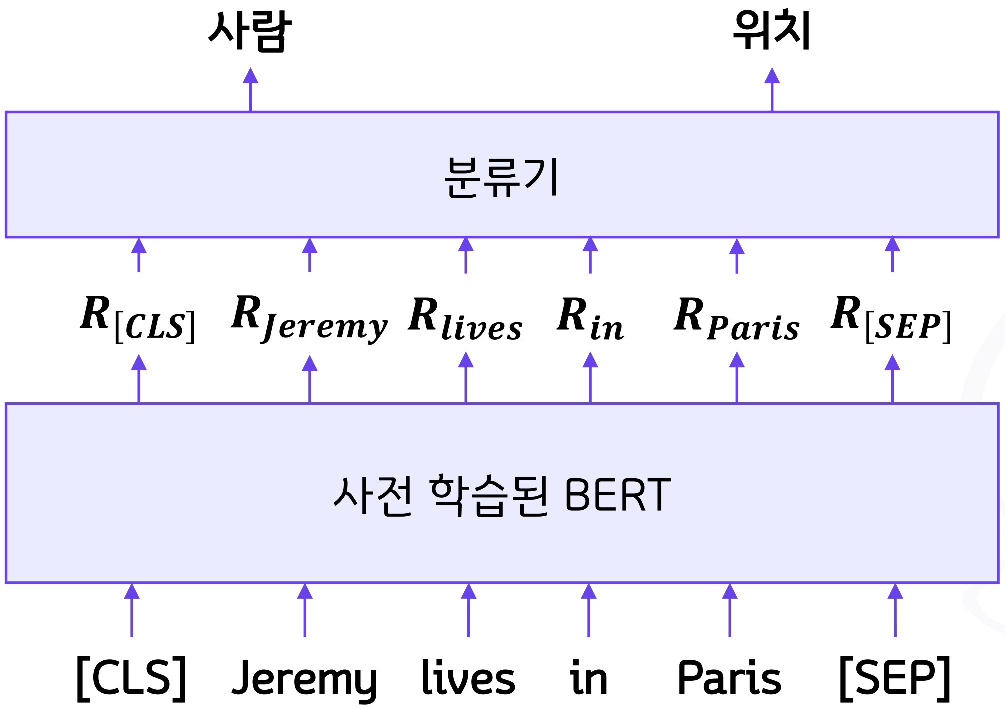
- 개체명 인식 (Named Entity Recognition, NER)
  - 목표: 개체명을 미리 정의된 범주로 분류하는 것
    - 예: Jeremy lives in Paris (제레미는 파리에 산다) → "제레미: 사람, 파리: 위치"로 분류
- BERT 모델의 파인 튜닝으로 NER 수행하기
  - 문장 토큰화 → [CLS][SEP] 추가

    → 사전 학습된 BERT 모델에 토큰 입력 → 모든 토큰의 표현 벡터 획득

    → 토큰 표현을 분류기(FFN+Softmax 함수)에 입력

    → 분류기가 개체명이 속한 범주를 반환

개체명 인식


사람
위치
분류기
$R_{[CLS]}$ R_{Jeremy} R_{lives} R_{in} R_{Paris} $R_{[SEP]}$
사전 학습된 BERT
[CLS] Jeremy lives in Paris [SEP]

THANK YOU