

□

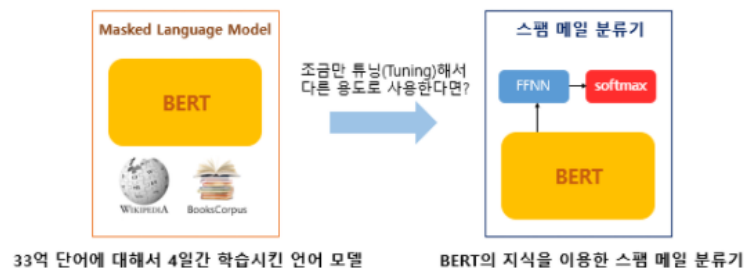
# BERT(Bidirectional Encoder Representations from Transformers)

## Ch17. BERT(Bidirectional Encoder Representations from Transformers)

트랜스포머(transformer)의 등장 이후, RNN 계열의 신경망인 LSTM, GRU는 트랜스포머로 대체되어가는 추세이다. 특히 트랜스포머로 부터 파생된 BERT를 기반한 ALBERT, RoBERTa, ELECTRA와 같은 모델들이 널리 쓰이고 있다.

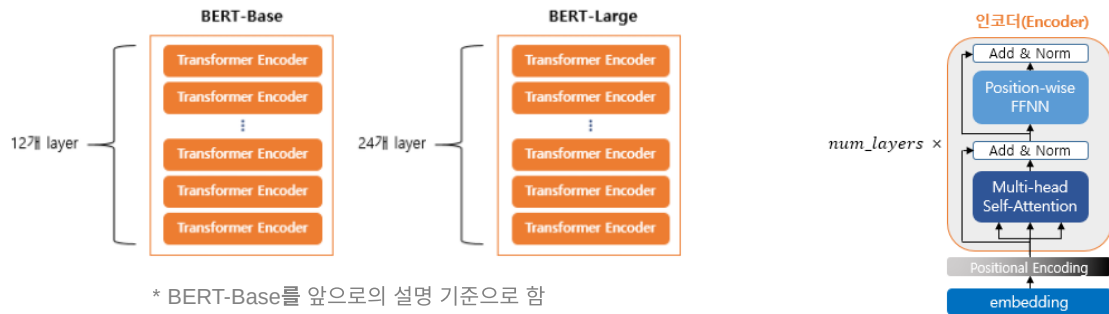
### 1. BERT의 개요

- 2018년에 구글이 공개한 트랜스포머 기반의 사전 훈련된 모델
- 위키피디아(25억 단어)와 BooksCorpus(8억 단어)와 같은 레이블이 없는 텍스트 데이터로 사전 훈련
- 주로 **fine-tuning**을 거쳐 여러 task에 이용
  - 사전 훈련된 BERT 모델 + 추가적 훈련 + 하이퍼 파라미터 재조정



### 2. BERT의 구조&크기

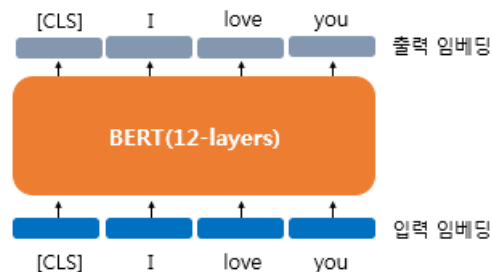
- BERT의 기본 구조 : 트랜스포머의 인코더를 쌓아올린 것
- Open AI GPT-1과 성능을 비교하기 위해서 GPT-1과 동등한 크기로 만든 **BERT-Base**와 BERT의 최대 성능을 보여주기 위해 만들어진 **BERT-Large**가 존재



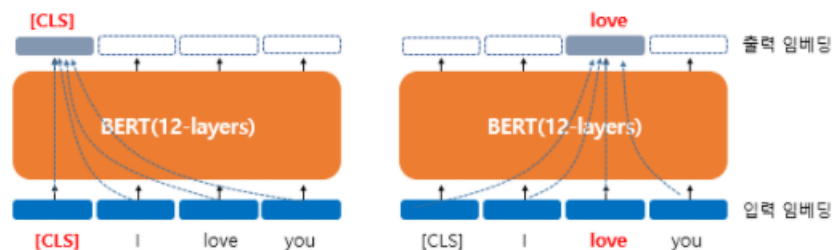
Aa 속성	≡ 인코더 층 수(L)	≡ d_model의 크기(D)	≡ 셀프 어텐션 헤드 수(A)
<u>초기 트랜스포머</u>	6	512	
<u>BERT-Base</u>	12	768	12
<u>BERT-Large</u>	24	1024	16

### 3. BERT의 문맥을 반영한 임베딩(Contextual Embedding)

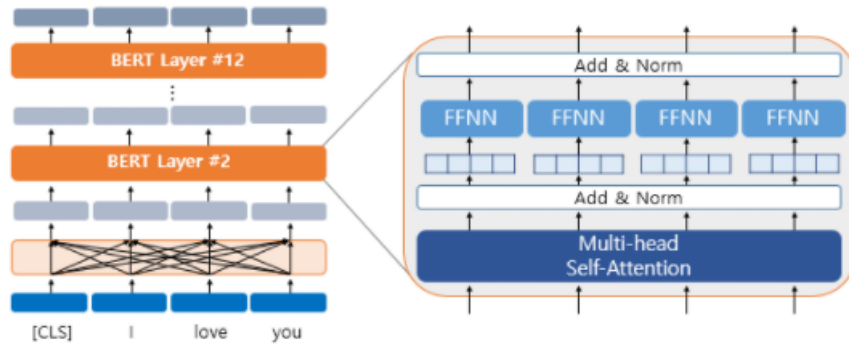
- **BERT의 입력:** 단순한 임베딩 층(Embedding layer)를 지난 임베딩 벡터들  
BERT-Base에서는 단어 임베딩 크기가 768이므로 모든 단어들은 768차원의 임베딩 벡터가 되어 BERT의 입력값으로 사용된다.
- **BERT의 출력:** 각 단어에 대한 **문맥 정보를 포함한** 768차원의 임베딩 벡터들



- BERT의 입력에 사용되는 각 임베딩 벡터는 단순히 자신의 임베딩 값이었지만 BERT의 연산을 거쳐 출력되면 문맥의 모든 단어 정보를 담은 임베딩 벡터로 바뀐다.



- 이렇게 하나의 단어가 모든 단어(문맥)를 참고하는 연산은 BERT의 모든 인코더 계층에서 이루어진다.
- 모든 인코더 계층에서 문맥 참고가 가능한 이유는 인코더 계층 내부에 존재하는 **셀프 어텐션 + 포지션 와이즈 피드 포워드 신경망** 때문이다.



## 4. BERT의 서브워드 토크나이저 : WordPiece Model(WPM)

### 1) 서브워드 토크나이저

- 단어 집합의 한계

아무리 많은 단어를 학습한다고 해도 신조어, 단어 크기의 한계 등으로 인해 Out-Of-Vocabulary가 발생함

- 서브워드 분리 이유

하나의 단어를 더 작은 단위의 의미를 가진 서브워드로 쪼개 단어 집합의 한계를 뛰어 넘고자 함

### 2) WordPiece Model(WPM)

- BERT는 서브워드 토크나이저 중 WordPiece 토크나이저를 사용

- WordPiece의 토큰화 수행 방식

1. BERT는 이미 훈련 데이터로부터 만들어진 단어 집합을 가지고 있는 상태이다.
2. 토큰이 단어 집합에 존재 → 해당 토큰 분리 X
3. 토큰이 단어 집합에 존재 X → 해당 토큰을 서브 워드로 분리

- 예시

만약 BERT의 단어 집합에 embedding이 존재하지 않고 대신 em, ##bed, ##ding, #s라는 서브 워드들이 존재한다면 em, ##bed, ##ding, #s 을 토큰으로 삼는다.

(##은 해당 서브워드가 단어의 중간부터 등장하는 서브워드라는 것을 알려주기 위한 기호)

- 사용 방법

BertTokenizer를 import해 문장을 토큰화 할 수 있으며 BERT의 단어 집합의 크기는 30,522이다.

```
import pandas as pd
from transformers import BertTokenizer

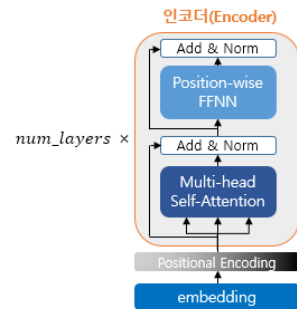
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased") # Bert-base의 토크나이저
result = tokenizer.tokenize('Here is the sentence I want embeddings for.')
print(result)

# ['here', 'is', 'the', 'sentence', 'i', 'want', 'em', '##bed', '##ding', '##s', 'for', '.']
```

## 5. 포지션 임베딩(Position Embedding)

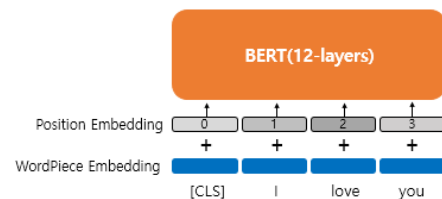
### 1) Positional Encoding

- 트랜스포머에서 단어의 위치정보를 표현하는 방법
- 사인 함수와 코사인 함수를 사용하여 위치에 따라 다른 값을 가지는 행렬을 만들어 이를 단어 벡터들과 더해 위치를 표현



### 2) Position Embedding

- BERT에서 단어의 위치정보를 표현하는 방법
- 위치정보를 학습을 위한 새로운 임베딩 계층 사용
- 실제 BERT에서는 문장의 최대 길이를 512로 하고 있으므로, 총 512개의 포지션 임베딩 벡터가 학습되어 이용된다.

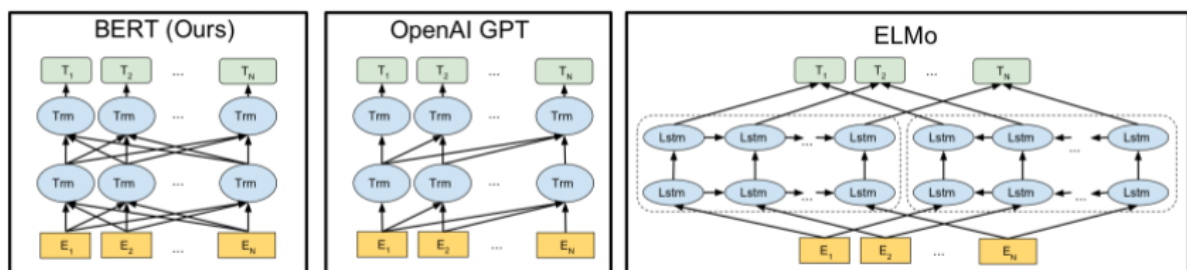


즉, BERT는 다음과 같은 두 임베딩 계층을 이용한다.

- 단어 집합의 크기가 30,522개인 단어 벡터를 위한 임베딩 층
- 문장의 최대 길이가 512이므로 512개의 포지션 벡터를 위한 임베딩 층

## 6. BERT의 사전 훈련(Pre-training)

### 1) BERT와 기존 모델의 pre-training 차이점



BERT	OpenAI GPT	ELMo
<ul style="list-style-type: none"> <li>트랜스포머의 인코더를 이용해 이전 단어와 다음 단어들로부터 단어 예측</li> <li>양방향 언어 모델</li> </ul>	<ul style="list-style-type: none"> <li>트랜스포머의 디코더를 이용해 이전 단어들로부터 다음 단어를 예측</li> <li>단방향 언어 모델</li> </ul>	<ul style="list-style-type: none"> <li>정방향 LSTM과 역방향 LSTM을 각각 훈련</li> <li>양방향 언어 모델</li> </ul>

BERT의 사전 훈련 방법은 크게 마스크드 언어 모델(Masked Language Model)과 다음 문장 예측(Next sentence prediction, NSP)이 있다. 특히 마스크드 언어 모델을 통해 BERT는 양방향성을 얻을 수 있었다.

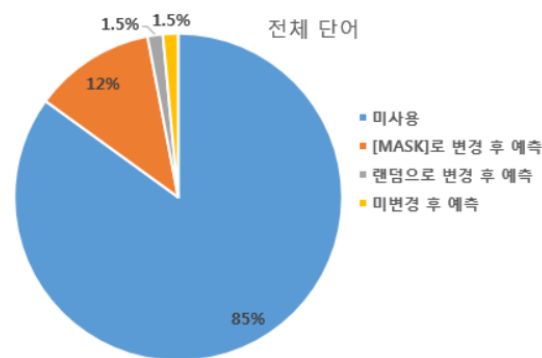
## 2) 마스크드 언어 모델(Masked Language Model, MLM)

- 마스크를 사용하는 이유

마스크 없이 문장을 학습할 경우 양방향 학습을 해버리면 문장의 전체 정보를 학습하는 것과 다를 없게 된다. 즉, 이미 문단을 통채로 암기하고 있는 것과 마찬가지다. 따라서 특정 부분에 오는 단어를 예측하는 것은 의미 없는 일이 되버린다. 하지만 마스크가 있는 상태라면 애초에 문단 전체 정보를 다 알 수 없으므로 양방향 학습을 해도 문제가 없다.

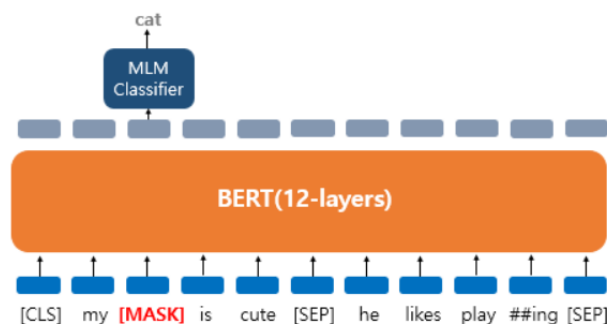
- BERT의 마스크 방법

1. 입력 텍스트의 15%에 해당하는 단어를 랜덤으로 고른다.
2. 1번 중, 80%의 단어들은 [MASK]로 변경한다.
3. 1번 중, 10%의 단어들은 랜덤으로 단어가 변경된다.
4. 1번 중, 10%의 단어들은 동일하게 둔다.



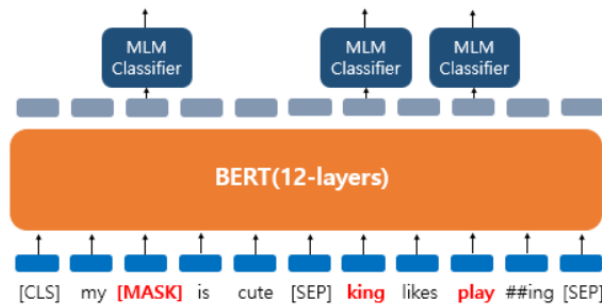
이렇게 하는 이유는 [MASK]만 사용할 경우에는 [MASK] 토큰이 파인 튜닝 단계에서는 나타나지 않으므로 사전 학습 단계와 파인 튜닝 단계에서의 불일치가 발생하기 때문이다.

- 예시1 : 'My **dog** is cute. he likes playing' ⇒ 'My **[Mask]** is cute. he likes playing'



- 'dog' 토큰을 [MASK]로 변경

- BERT 모델은 [MASK]의 원래 단어를 예측하는 것이 목적
- 오직 'dog' 위치의 출력층의 벡터만이 사용 (출력층에 있는 다른 위치의 벡터들은 예측과 학습에 사용 X)
- 출력층에서는 예측을 위해 단어 집합의 크기만큼의 밀집층(Dense layer) + 소프트맥스 함수 사용
- 예시2 : 'My **dog** is cute. **he** likes **playing**' ⇒ 'My **[Mask]** is cute. **king** likes **playing**'



- 'dog' 토큰을 [MASK]로 변경 + 'he'는 랜덤 단어 'king'으로 변경 + 'play'는 변경되진 않았지만 예측에 사용
- BERT 모델은 [MASK]와 king, play에 대해 모두 원래 단어가 무엇이었는지 예측하는 것이 목적

### 3) 다음 문장 예측(Next Sentence Prediction, NSP)

- BERT는 두 개의 문장을 준 후에 이 문장이 이어지는 문장인지 아닌지를 맞추는 방식을 사용

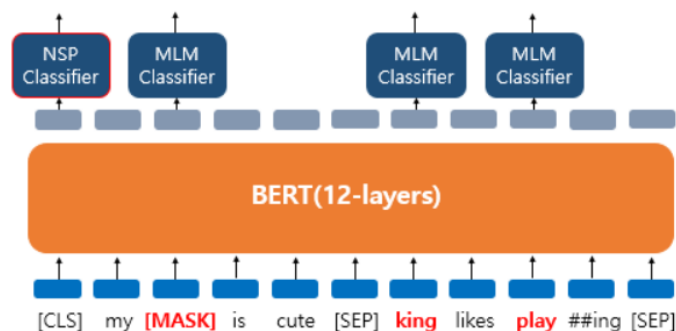
#### ▼ 이어지는 문장 vs. 이어지지 않는 문장

이어지는 문장 : The man went to the store. + He bought a gallon of milk.


이어지지 않는 문장 : Sentence A : The man went to the store. + dogs are so cute.

- 다음
- 50:50 비율로 실제 이어지는 두 개의 문장과 랜덤으로 이어붙인 두 개의 문장을 주고 훈련
- 입력에서 첫번째 문장과 두번째 문장의 끝에 [SEP]라는 특별 토큰을 사용해서 문장을 구분
- 두 문장이 실제 이어지는 문장인지 아닌지를 [CLS] 토큰의 위치의 출력층에서 이진 분류 문제로 학습

| [CLS]은 이 외에도 모든 분류 문제에서 이용된다.

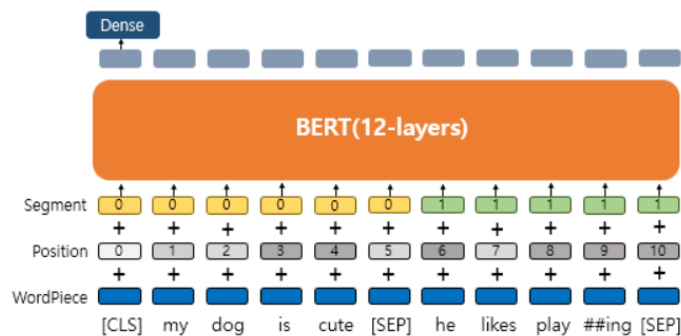


BERT가 언어 모델 외에도 다음 문장 예측이라는 태스크를 학습하는 이유는 BERT가 풀고자 하는 태스크 중에서는 QA(Question Answering)나 NLI(Natural Language Inference)와 같이 두 문장의 관계를 이해하는 것이 중요한 태스크들이 있기 때문이다.

 마스크드 언어 모델과 다음 문장 예측은 따로 학습하는 것이 아니라 위의 그림처럼 loss를 합하여 학습이 동시에 이루어진다.

## 7. 세그먼트 임베딩(Segment Embedding)

- QA 등과 같은 두 개의 문장 입력이 필요한 태스크를 풀기 위해 이용
- 문장 구분을 위해 BERT는 첫번째 문장에는 0, 두번째 문장에는 1을 더해주는 방식의 세그먼트 임베딩 사용



 이 경우, 결론적으로 BERT는 총 3개의 임베딩 층을 사용

- 1) WordPiece Embedding : 실질적인 입력이 되는 워드 임베딩
- 2) Position Embedding : 위치 정보를 학습하기 위한 임베딩
- 3) Segment Embedding : 두 개의 문장을 구분하기 위한 임베딩

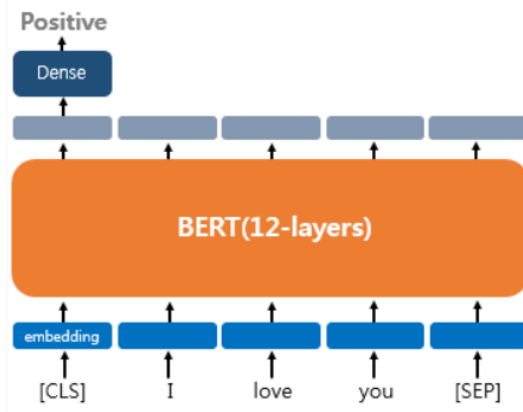
주의할 점은 일반적으로 설명의 편의를 위해 QA task에서 BERT는 두 개의 구분된 문장을 입력 받는다고 설명하지만 실제로는 두 개의 문단이 될 수도 있다. 즉 Q와 A로 나뉘지 기만 한다면 Q와 A는 각각 문장, 문단, 텍스트 중 어느 것이든 가능하다.

만약 QA task가 아니라 문장 구분을 할 필요가 없는데도 segment embedding 계층이 있는 BERT 모델을 사용할 경우, segment embedding 값을 모두 0으로 주면 된다.

## 8. BERT를 파인 튜닝(Fine-tuning)하기

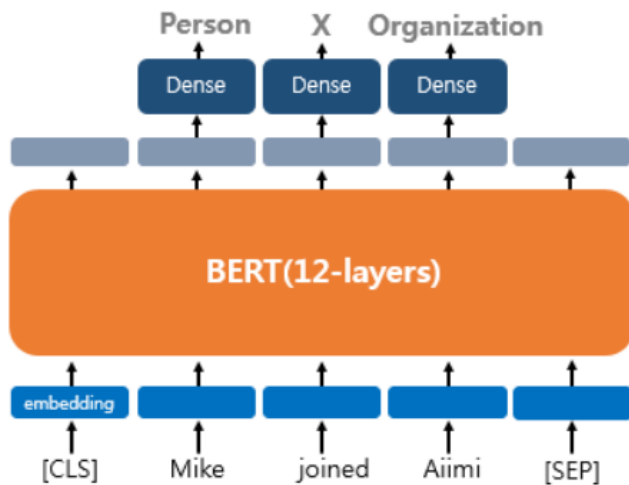
이렇게 다양한 기법들을 통해 pre-training된 BERT는 fine-tuning을 통해 여러 task에 이용될 수 있다.

### 1) 하나의 텍스트에 대한 텍스트 분류 유형(Single Text Classification)



- 영화 리뷰 감성 분류, 로이터 뉴스 분류 등과 같이 입력된 문서에 대해서 분류를 하는 유형
- **분류 문제**를 풀기 위해 문서의 시작에 **[CLS]** 라는 토큰을 입력
- [CLS] 토큰의 위치의 출력층에서 밀집층(Dense layer) 또는 완전 연결층(fully-connected layer)이라고 불리는 층들을 추가하여 분류에 대한 예측 실행

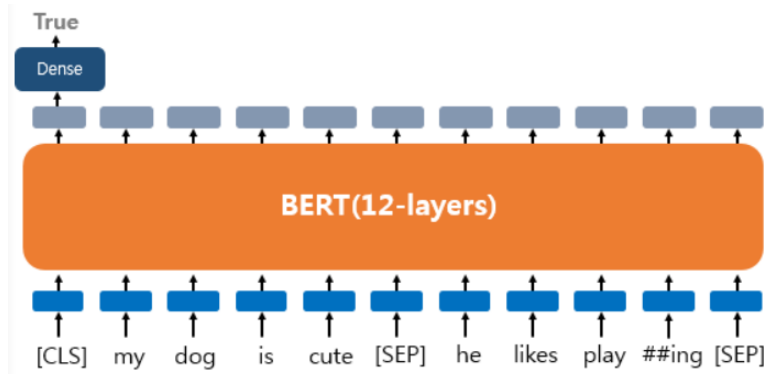
## 2) 하나의 텍스트에 대한 태깅 작업(Tagging)



- 문장의 각 단어에 품사를 태깅하는 품사 태깅 작업, 개체를 태깅하는 개체명 인식 작업 등
- 출력층에서는 입력 텍스트의 각 토큰의 위치에 밀집층을 사용하여 분류에 대해 예측 실행

## 3) 텍스트의 쌍에 대한 분류 또는 회귀 문제(Text Pair Classification or Regression)





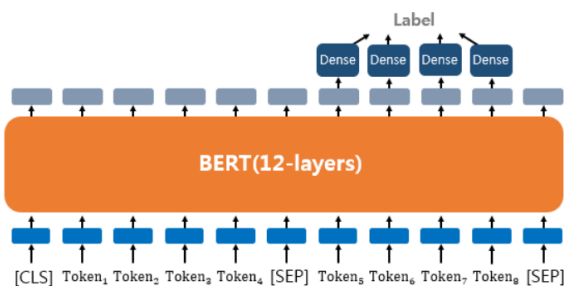
- 자연어 추론 문제 등에 이용

#### ▼ 자연어 추론 문제

두 문장이 주어졌을 때, 하나의 문장이 다른 문장과 논리적으로 어떤 관계에 있는지를 분류하는 것. 모순 관계(contradiction), 함의 관계(entailment), 중립 관계(neutral) 등이 존재

- 입력 텍스트가 1개가 아니므로, [SEP] 토큰과 세그먼트 임베딩(0, 1)을 모두 사용하여 문서를 구분

## 4) 질의 응답(Question Answering)



- BERT로 QA를 풀기 위해서 질문과 본문이라는 두 개의 텍스트의 쌍을 입력
- 대표적인 데이터셋 : SQuAD(Stanford Question Answering Dataset) v1.1
  - 예시

Q : "강우가 떨어지도록 영향을 주는 것은?"

A : "기상학에서 강우는 대기 수증기가 응결되어 중력의 영향을 받고 떨어지는 것을 의미합니다. 강우의 주요 형태는 이슬비, 비, 진눈깨비, 눈, 싸락눈 및 우박이 있습니다."

BERT에게 기대되는 답 : "중력"

## 9. BERT의 하이퍼파라미터

- 100만 step 훈련 ≈ (총 합 33억 단어 코퍼스에 대해 40 에포크 학습)
- 옵티마이저 : 아담(Adam)
- 학습률(learning rate) : 10<sup>-4</sup>
- 가중치 감소(Weight Decay) : L2 정규화로 0.01 적용

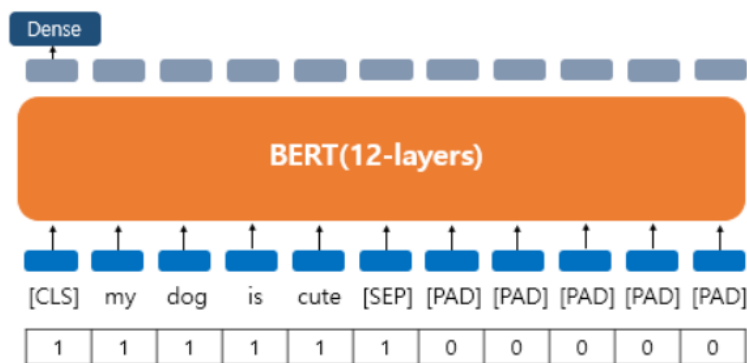
- 드롭 아웃 : 모든 레이어에 대해서 0.1 적용
- 활성화 함수 : relu 함수가 아닌 gelu 함수
- 배치 크기(Batch size) : 256

## 10. 어텐션 마스크(Attention Mask)

- BERT를 실제로 실습하게 되면 BERT가 어텐션 연산을 할 때, 불필요하게 패딩 토큰에 대해서 어텐션을 하지 않도록 실제 단어와 패딩 토큰을 구분할 수 있도록 알려주는 입력값이 필요하며 이를 어텐션 마스크라 한다.
- 어텐션 마스크는 다음과 같은 값을 가진다.

1 : 실제 단어. 어텐션 마스크 X

0 : 패딩 토큰. 어텐션 마스크 O

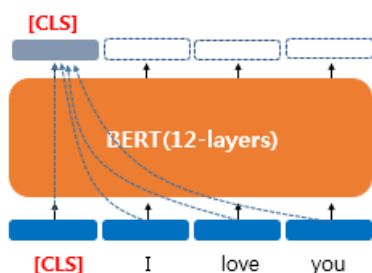


## Ch17-7. 센텐스 버트(Sentence BERT, SBERT)

Sentence BERT란 문장 자체를 하나의 벡터로 임베딩 해주는 BERT를 말한다.

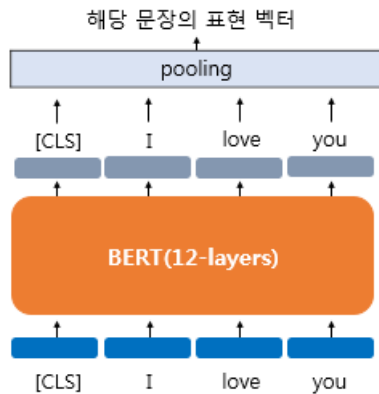
### 1. BERT의 문장 임베딩 방법

#### 1) [CLS] 토큰 이용



- BERT로 텍스트 분류 문제를 풀 때, [CLS] 토큰의 출력 벡터를 출력층(다음 입력층)의 입력으로 사용했던 이유는 [CLS] 토큰이 입력된 문장에 대한 총체적 표현이라고 간주하고 있기 때문이다.
- 다시 말해 [CLS] 토큰 자체를 입력 문장의 벡터로 간주할 수 있다.

## 2) “[CLS] 토큰 + 모든 출력 벡터”의 Pooling 이용



- 출력된 단어 벡터들에 Pooling을 적용한 벡터를 문장 벡터로 간주할 수 있다.
- 이때 Pooling 계층에 Average Pooling과 Max Pooling을 적용할 수 있다.

Average Pooling : 문장의 모든 단어의 의미를 반영

: 문장의 중요한 단어를 위주로 의미 반영

Max Pooling

## 2. SBERT(센텐스버트, Sentence-BERT)

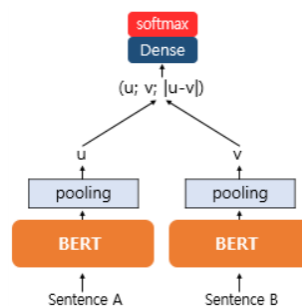
- BERT의 문장 임베딩의 성능을 우수하게 개선시킨 모델
- 위에서 언급한 BERT의 문장 임베딩을 응용하여 BERT를 파인 튜닝

### 1) 문장 쌍 분류 태스크로 파인 튜닝

- 대표적으로 NLI(Natural Language Inferencing) 문제가 있다.
- NLI는 두 개의 문장이 수반(entailment) 관계인지, 모순(contradiction) 관계인지, 중립(neutral) 관계인지를 맞추는 문제를 말한다.

문장 A	문장 B	레이블
A lady sits on a bench that is against a shopping mall.	A person sits on the seat.	Entailment
A lady sits on a bench that is against a shopping mall.	A woman is sitting against a building.	Entailment
A lady sits on a bench that is against a shopping mall.	Nobody is sitting on the bench.	Contradiction
Two women are embracing while holding to go packages.	The sisters are hugging goodbye while holding to go packages after just eating lunch.	Neutral

- NLI 데이터 학습을 위한 SBERT의 구조



전체 문장 임베딩 벡터 크기가  $n$ 이라면 Dense layer에 입력되는 벡터의 크기는  $u, v, |u - v|$ 를 이어붙인  $3n$ 이 된다.

실제 레이블과의 오차를 줄이는 방식으로 학습된다.

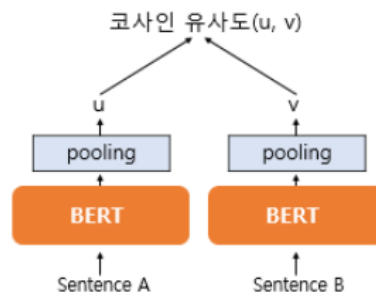
## 2) 문장 쌍 회귀 태스크로 파인 튜닝

- 대표적으로 STS(Semantic Textual Similarity) 문제가 있다.
- STS란 두 개의 문장으로부터 의미적 유사성을 구하는 문제를 말한다.

문장 A	문장 B	레이블
A plane is taking off.	An air plane is taking off.	5.00
A man is playing a large flute.	A man is playing a flute.	3.80
A man is spreading shredded cheese on a pizza.	A man is spreading shredded cheese on an uncoo...	3.80
Three men are playing chess.	Two men are playing chess.	2.60
A man is playing the cello.	A man seated is playing the cello.	4.25

\* 여기서 레이블은 두 문장의 유사도 점수로 0~5의 값을 가진다.

- STS 데이터 학습을 위한 SBERT 구조



두 문장의 임베딩 벡터의 코사인 유사도 값과 레이블 유사도간의 평균 제곱 오차(Mean Squared Error, MSE)를 최소화하는 방식으로 학습한다.