NLP Adv Assignment

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

들어가기 전

- BERT : Bidirectional Encoder Representations from Transformers
  - Bidirectional : 양방향
  - Encoder : 입력값을 숫자로
  - Transformer : 2017년에 구글에서 공개한 인코더(입력값을 양방향으로 처리)/디코더(입력값을 단방향(왼→오)으로 처리) 구조를 가진 딥러닝 모델
- **문맥을 양방향으로 이해해서 숫자의 형태로 바꿔주는 딥러닝 모델**

✔️ **GPT-1** 이란?

- 2018년 Open AI에서 Transformer의 디코더 구조를 사용해서 만든 자연어 처리 모델
- Generative Training으로 학습된 Language 모델이 얼마나 자연어 처리 능력이 우수한지 보여주는 모델
- 문장을 데이터로, 단어를 하나씩 읽어가면서 다음 단어 예측(왼→오)
- 비지도 학습
- 따라서 GPT 학습시에 가장 중요한건 엄청난 양의 데이터

- 구글 왈) BERT 논문을 통해 GPT-1의 Transformer Decoder를 사용한 자연어 처리 능력은 문장을 처리하는데 부족함이 있다. 질의 및 응답 능력은 문맥 이해 능력이 상당히 중요한데 단순히 왼→오 로 읽어나가는 방식으로는 문맥 이해에 약점이 있다.
- 구글은 양방향으로 문맥을 이해할 수 있는 Language 모델을 **BERT**라는 이름으로 발표

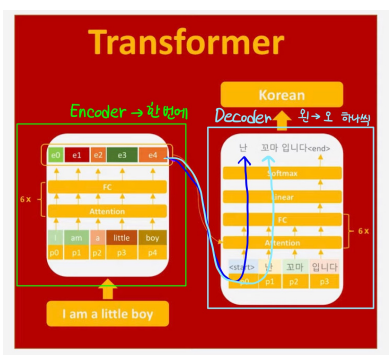
✔️ **Transformer** 란?

- 입력값이 인코더에 입력됨 → 각 토큰들은 positional encoding과 더해짐 → Encoder는 이 값들의 행렬 계산을 통해 Attention Vector를 생성함
- Attention Vector는 토큰의 의미를 구하기 위해 사용됨.(단어 하나만을 가지고는 문자가 행위를 말하는지 명사 하나를 의미하는지 모름)
- Attention Vector를 통해 각각의 토큰들은 문장 속의 모든 토큰을 봄으로써 각 토큰의 정확한 의미를 모델에 전달
- Attention Vector는 Fully Connected Layer(FC) 로 전송됨. 이와 동일한 과정이 6번 진행
- 최종 출력값은 Transformer의 Decoder 입력값으로 사용됨.

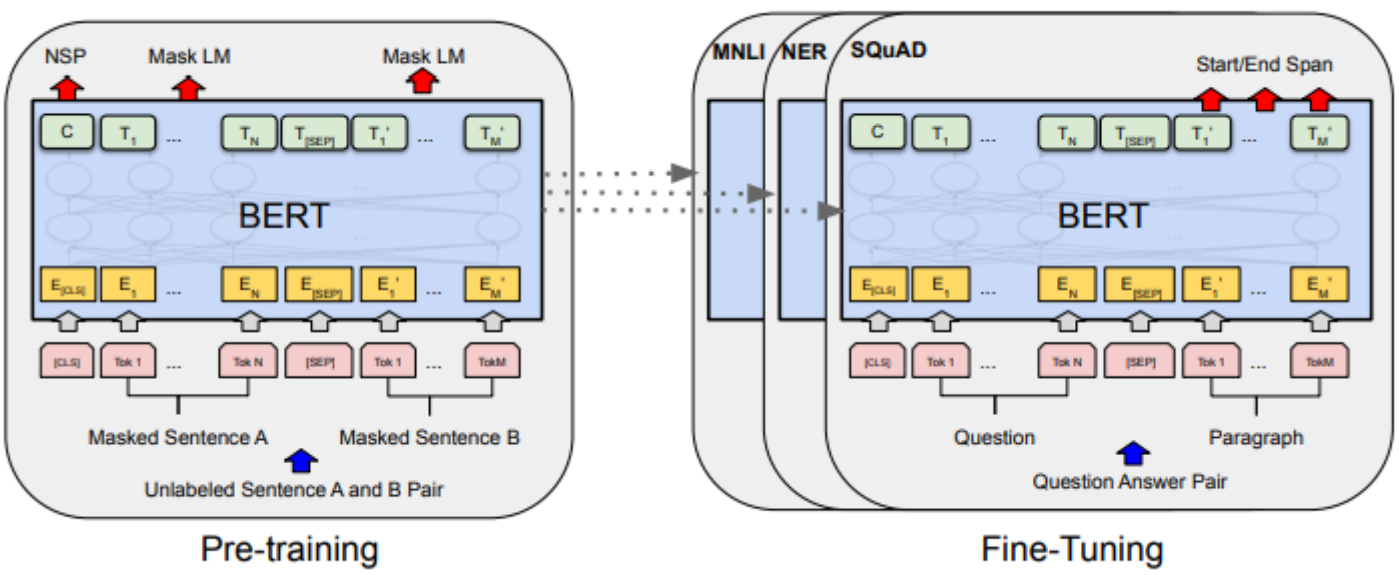
✅ **Encoder**는 모든 토큰을 한 번에 계산한다. 왼→오 로 하나씩 읽어가는 과정이 없다.

✅ **Decoder**는 왼쪽에서 오른쪽으로 출력값을 생성한다. Decoder는 이전 생성된 디코더의 출력값과 인코더의 출력값을 사용해서 현재의 출력값을 생성함.

- Decoder 역시 Attention Vector를 만들고 Fully Conneted Layer로 보내는 작업이 6번 진행됨.
- End Token이라는 special token이 나올때까지 반복

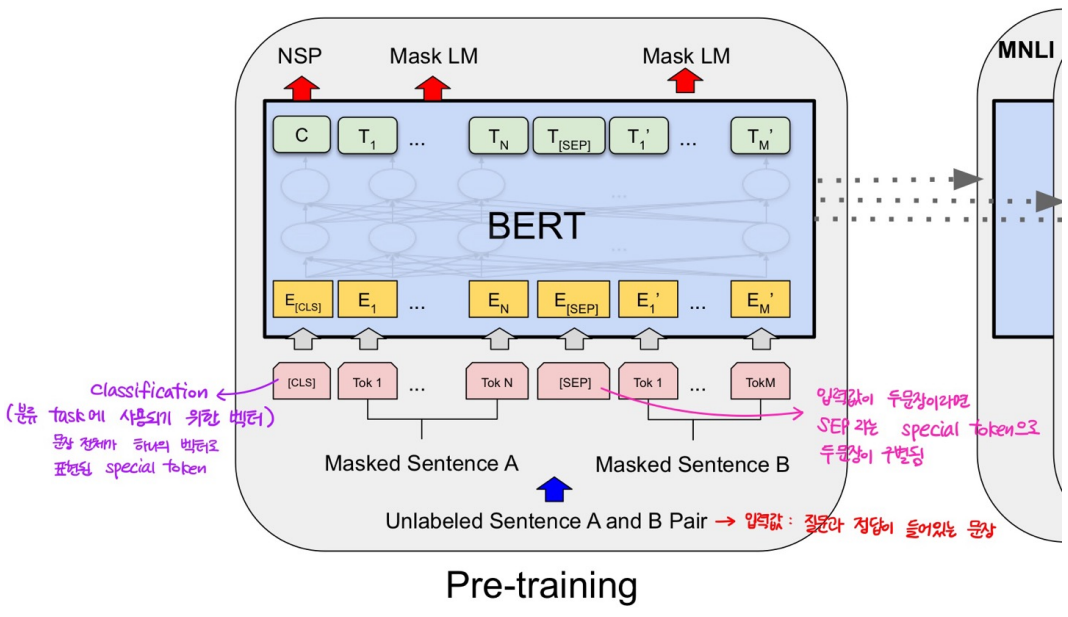


BERT

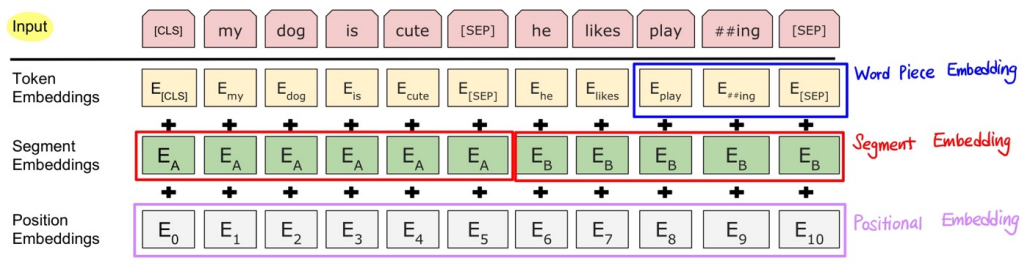


- BERT는 문장을 그대로 학습하되 가려진 단어(mask token)를 예측하도록 학습함
  - EX) 오늘 <mask> 메뉴는 뭐야? → <mask>를 예측
- BERT는 크게 Pre-training과 Fine-Tuning으로 나뉜다.
- BERT는 입력값으로 두문장도 입력받을 수 있음(질의 및 응답)
- BERT는 한문장 or 두문장의 학습 데이터를 통해 토큰 간의 상관관계 뿐 아니라 문장 간의 상관관계도 학습함.
- Fine Tuning을 하기 위해 만들어졌다.

Pre-training



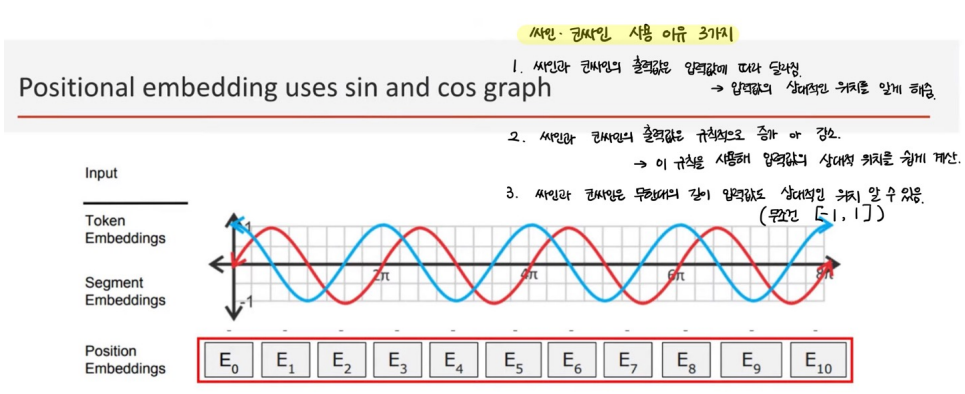
- 입력 토큰들이 position 임베딩, segment 임베딩과 더해진다.



- BERT는 **WordPiece Embedding**을 사용해 문장을 token 단위로 분리한다.
  - WordPiece Embedding은 단순히 띄어쓰기로 token을 나누는 것보다 효과적으로 토큰을 구분한다.
    - playing은 하나의 단어이지만, play+ing로 토큰이 나뉨.
  - 장점1) play: 놀다 & ing: 무언가를 하는 중이다 -> 딥러닝 모델에게 각각의 의미를 명확하게 전달 가능.
  - 장점2) 쪼개서 입력할 경우 신조어 or 오탈자의 경우에도 구별이 되어(ex. Googling = Google + ing) 딥러닝 단계에서 봤을 법한 단어들이 입력됨. -> 흔치 않은 단어들에 대한 예측이 향상된다.

- **Segment Embedding**은 두 개의 문장이 입력될 경우 각각의 문장에 서로 다른 숫자들을 더해준다.
  - 딥러닝 모델에게 두 개의 다른 문장이 있다는 것일 알려주기 위한 임베딩

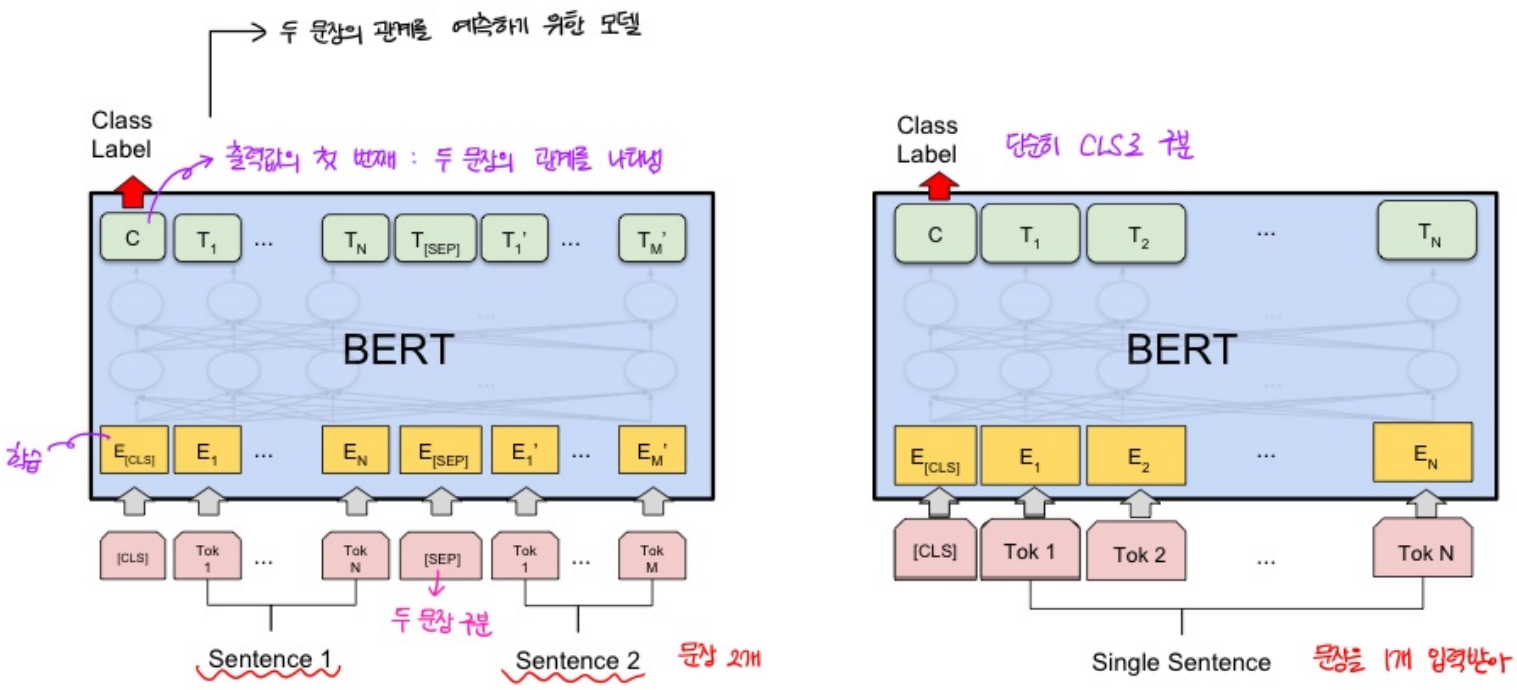
- **Positional Embedding**은 tokent들의 상대적 위치 정보를 알려줌.
  - 딥러닝 모델은 positional embedding으로 E1 → E2 → E3 이런 식으로 위치를 알 수가 있음.
  - positional embedding은 **싸인, 코싸인 함수** 이용 → **토큰의 상대적 위치**를 알고, 쉽게 계산하기 위함



  - 그럼 절대적 위치는 왜 사용하지 않는걸까?
    - 절대적 위치를 사용할 경우, 최장 길이의 문장을 준비해놓아야 함. 학습시에 사용했던 최장 길이의 문장보다 더 긴 문장은 받을 수가 없다.
    - 따라서 상대적 위치가 positional embedding에서 선호됨.

Fine-Tuning

- pre-training된 BERT는 인터넷에서 쉽게 다운받을 수 있다. 개발자가 알아야 하는 부분은 fine-tuning이다.
- 사용 목적에 따른 fine-tuning이 다르다.
- 예제



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

BERT의 성능

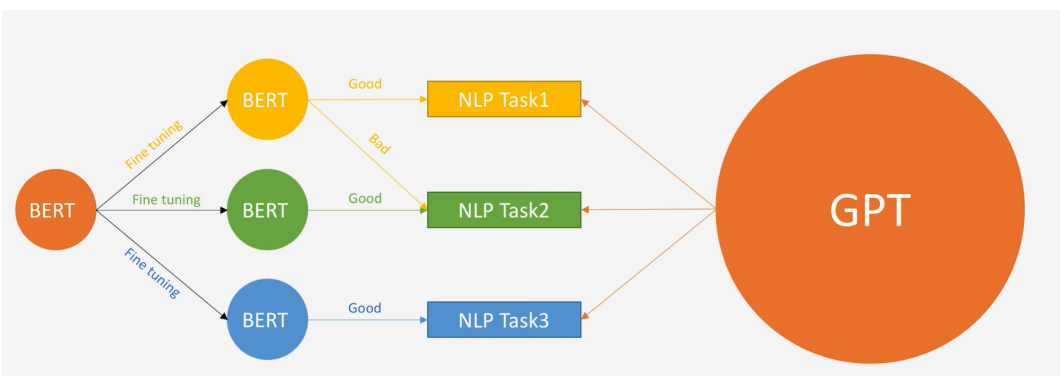
System	MNLI-(m/mm) 392k	QQP 363k	QNLI 108k	SST-2 67k	CoLA 8.5k	STS-B 5.7k	MRPC 3.5k	RTE 2.5k	**Average** -
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.8	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	87.4	91.3	45.4	80.0	82.3	56.0	75.1
$BERT_{BASE}$	84.6/83.4	71.2	90.5	93.5	52.1	85.8	88.9	66.4	79.6
$BERT_{LARGE}$	**86.7/85.9**	**72.1**	**92.7**	**94.9**	**60.5**	**86.5**	**89.3**	**70.1**	**82.1**

BERT vs GPT

BERT **GPT**

- Bi directional(양방향) LM
- Loves Fine-Tuning
- 각각의 다른 자연어 처리를 위해 따로 fine tuning이 필요함
- 모델 크기 작음
- 상대적으로 비용과 시간은 적게 필요하지만, fine-tuning을 개발자가 해줘야 하고, 별도의 시간과 비용이 듦

- Left to Right(단방향) LM
- Hates Fine-Tuning
- pre-training(선행학습)된 모델 자체
- 모델 크기가 매우 큼
- 한번 학습하는데 많은 비용과 시간 필요



주황색 동그라미는 pre-training을 나타낸 것

참고자료

1810.04805.pdf

https://www.youtube.com/watch?v=30SvdoA6ApE