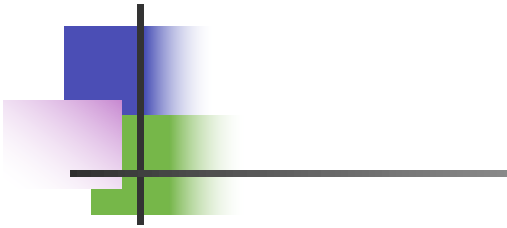
BERT

Sang Yup Lee



BERT (Bidirectional Encoder Representations from Transformers)

BERT

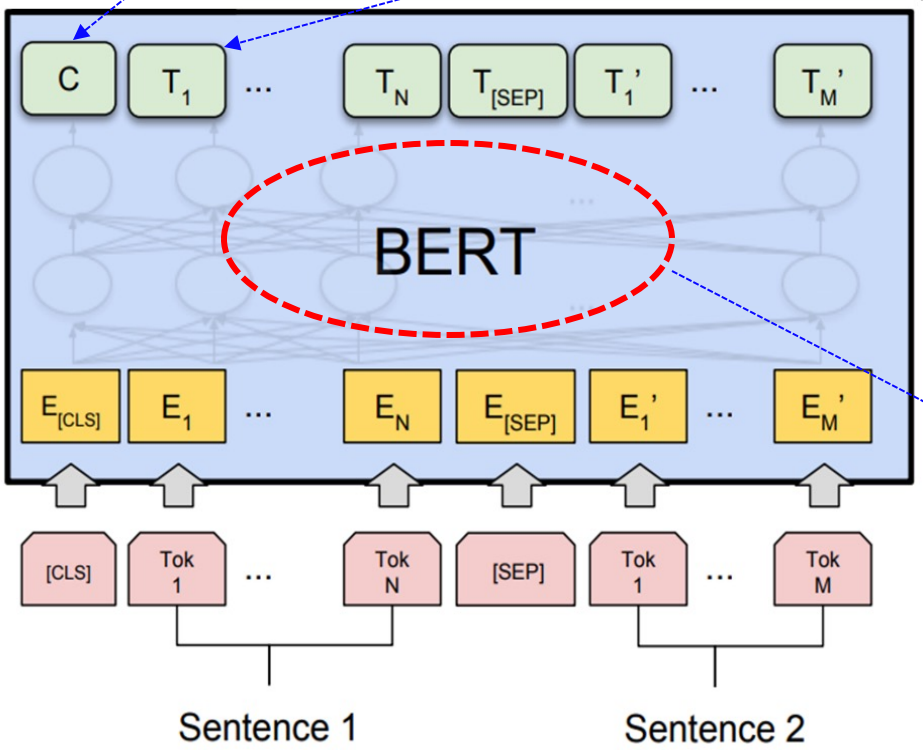
- BERT의 주 목적
  - Text 분석에서의 전이학습 (transfer learning)
  - 학습 데이터
    - BooksCorpus (800M words), English Wikipedia (2,500M words)
  - 위의 학습 데이터를 이용해서 모형의 파라미터를 학습 => downstream task에 따라서 fine tuning 혹은 feature-based 방법으로 사용 가능
- BERT의 구조
  - Transformer의 encoder 부분만 사용
  - 2가지 형태
    - $\text{BERT}_{\text{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M)
    - $\text{BERT}_{\text{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M)
    - L = encoder block의 수, H = 임베딩 벡터 또는 hidden state 벡터의 차원의 수, A = multi-head attention에서 사용된 attention의 수

Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

BERT의 구조

입력 데이터의
전체적인 특성정보를
담고 있음

Ti 는 i번째 token에 대한 hidden state 정보를 의미



- 두개의 문장으로 구성된 입력 (하나의 문장도 입력 가능), 데이터를 구성하고 있는 토큰들을 임베딩 벡터로 표현하여 입력 받음 (입력된 문장을 토큰으로 구분할 때는 WordPiece tokenization 방법을 사용)
- 두개의 토큰 추가: [CLS], [SEP]
- BERT를 통해 각 토큰에 대한 hidden state 출력

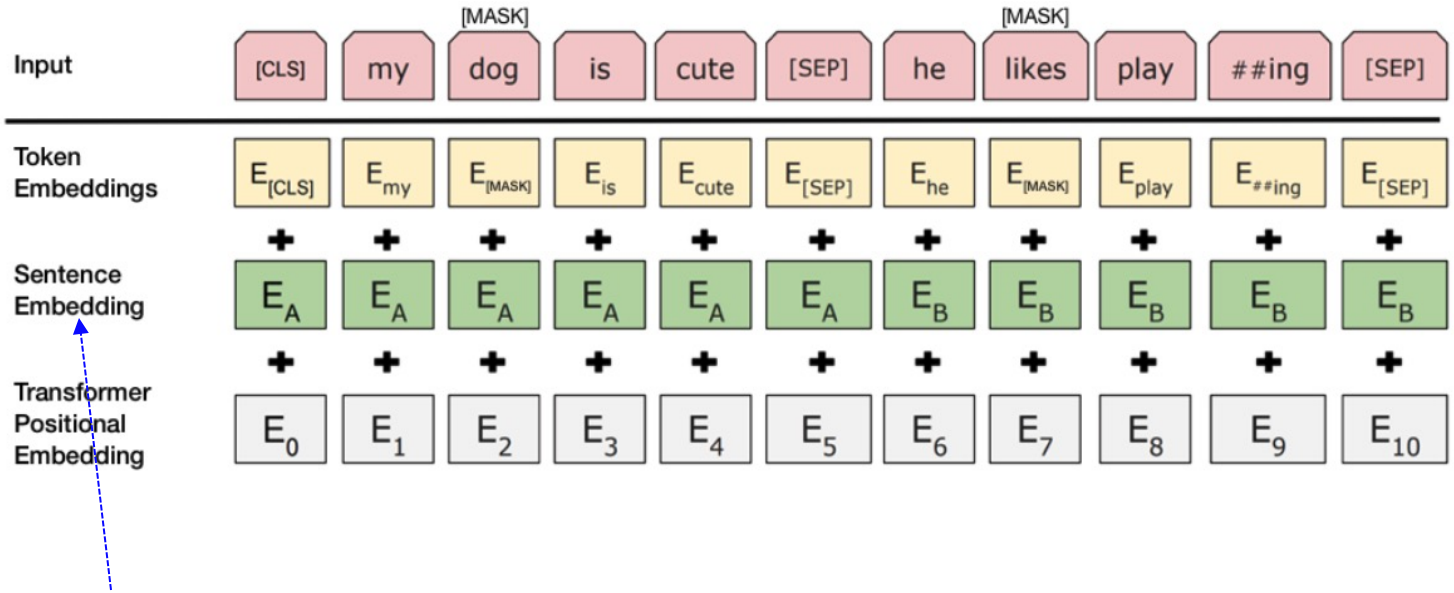
여러개의 encoder block으로 구성

BERT의 구조

- 새로운 토큰
  - [CLS]: 입력 시퀀스 데이터 전체의 정보를 반영하기 위한 토큰
    - 이 토큰에 대한 hidden state는 보통 분류 등의 목적에 사용
  - [SEP]: 두 개의 문장을 구분하기 위한 목적

BERT의 구조

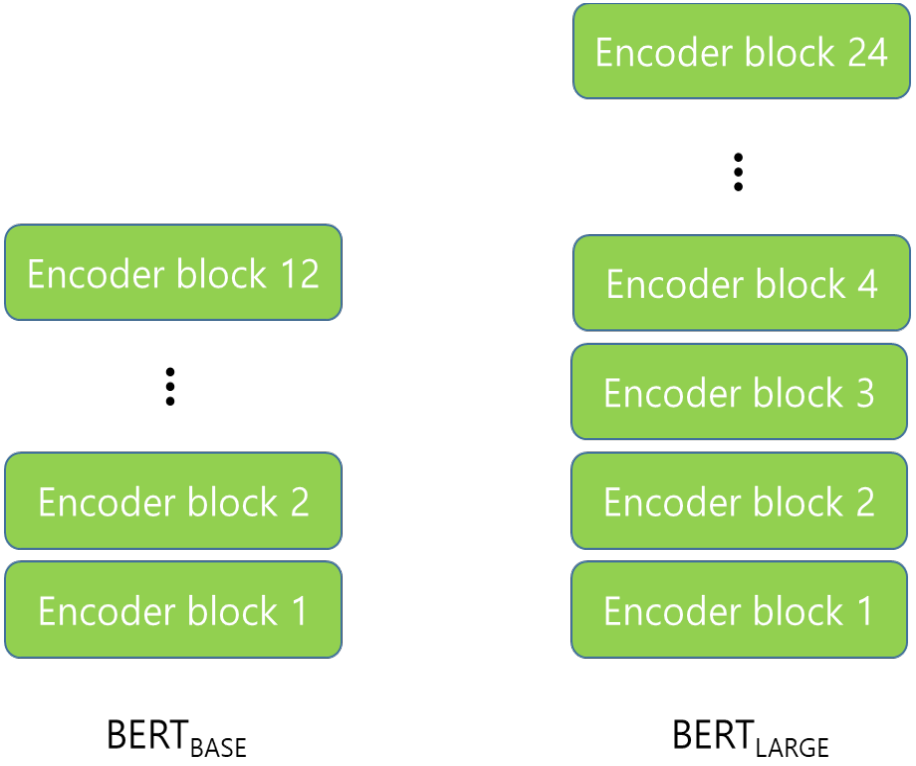
- BERT에 입력되는 토큰의 정보: 3가지



각 토큰이 입력된 두개의 문장 (즉, A와 B 문장)
중에서 어떤 문장에 속하는지를 나타내기 위해 사용

BERT의 구조

- BERT의 내부 구조



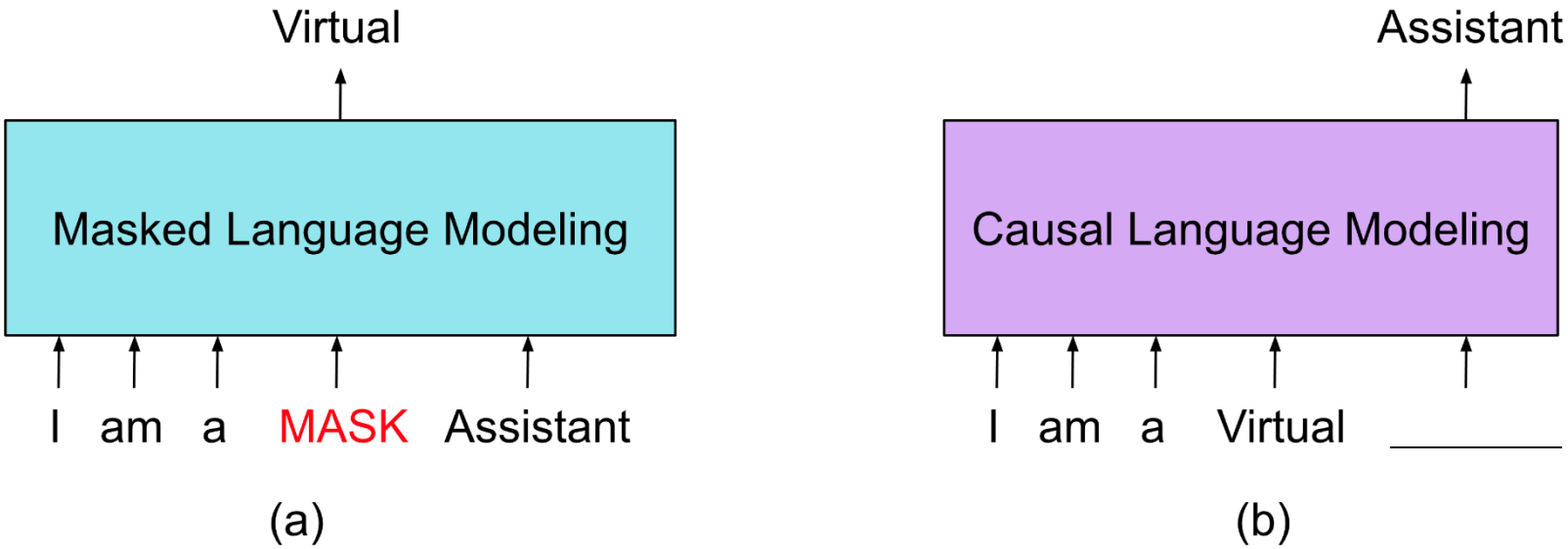
각 encoder block은 Transformer의 encoder block 과 동일한 구조
하지만, 각 토큰의 hidden state 벡터는 768차원

BERT 학습

- 아래 두 가지 작업을 수행하면서 토큰들의 임베딩 정보와 모형의 파라미터를 학습
- 작업의 종류
  - Masked language model (MLM)
    - 입력된 데이터를 구성하는 토큰 중에서 일부의 토큰을 비워놓고 (mask처리하고) 해당 단어를 맞히는 작업
  - Next Sentence Prediction (NSP)
    - 입력된 두 개의 문장이 서로 연속된 문장인지 그렇지 않은지를 맞히는 것
    - 분류의 문제라고 생각할 수 있음

BERT 학습

- Masked LM (a) vs. Causal LM (b)



BERT 학습

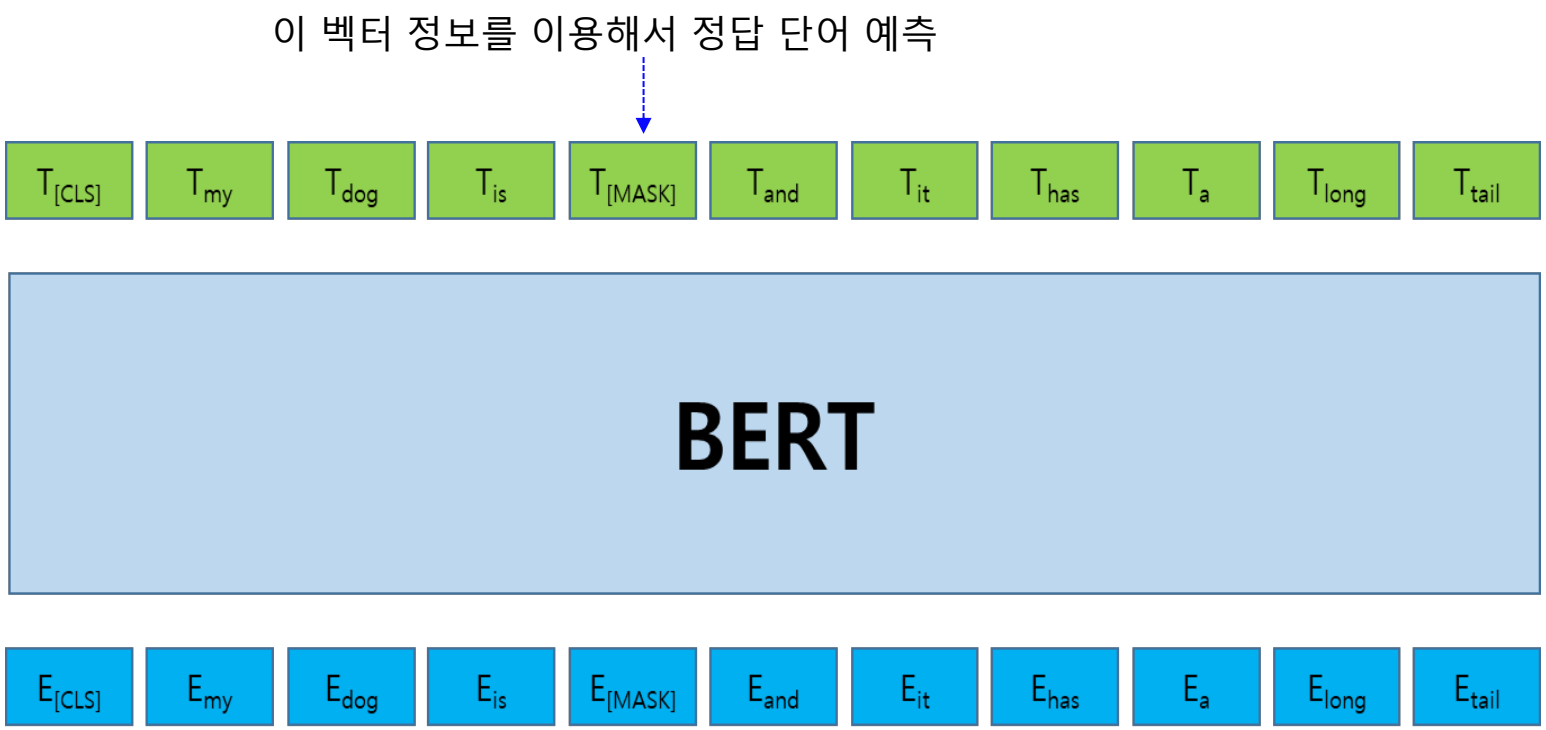
- MLM
  - 입력되는 데이터를 구성하는 토큰 중에서 15%를 랜덤하게 선택하고, 해당 토큰들에 대해 아래 작업 수행
    - 마스킹 방법
      - 이 중 80% 만 실제로 [MASK]로 대체
      - 10%는 임의의 단어로 대체
      - 10%는 원래 단어로 대체
    - 주된 이유 ⇒ 실제 분석을 위한 fine-tuning시에는 [MASK] 토큰이 사용되지 않음, 이러한 mismatch를 어느 정도 해결하기 위해서
  - 대체된 토큰에 대해서 BERT에서 출력되는 hidden state 벡터에 FNN과 softmax 층을 적용하여 비용함수 계산

BERT 학습

- MLM 과정의 구체적인 예
  - 예) 입력 시퀀스 → 'my dog is cute and it has a short tail‘
    - 입력 시퀀스를 구성하고 있는 토큰들 중에서 masking을 위해 임의로 선택된 15%에 해당하는 토큰이 'cute'라고 가정
    - 해당 문장이 학습에서 사용되는 정도를 100이라고 가정하면,
      - 그중 80%를 [MASK] 토큰으로 대체: 예) 'cute' → [MASK]
      - 10%를 임의의 토큰으로 대체: 예) ‘cute' → 'your'
      - 10%를 원래의 토큰으로 대체: 예) ‘cute' → ‘cute‘
      - 해당 토큰에 대한 hidden state vector를 이용해서 예측

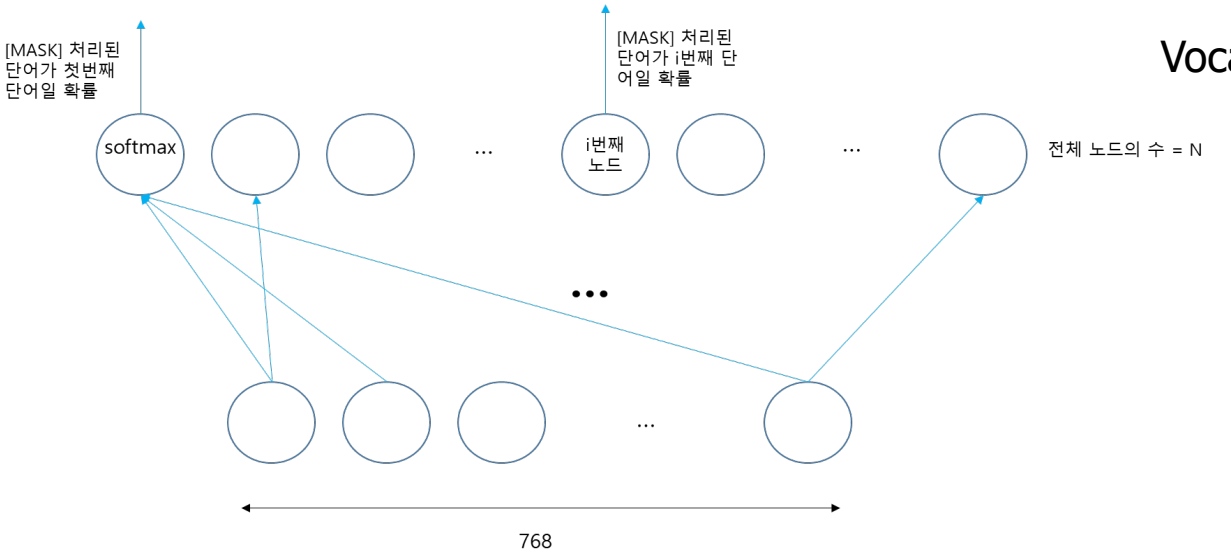
BERT 학습

- 입력 데이터: [CLS] my dog is [MASK] and it has a long tail [SEP]
- BERT를 통해서 각 토큰에 대한 hidden state 정보가 출력



BERT 학습

- MLM 과정의 구체적인 예 (cont'd)
  - 앞의 결과 중에서 [MASK] 토큰에 대한 결과인 T[MASK]를 softmax 활성화 함수의 입력값으로 전달
    - $T_{[MASK]}$는 768차원의 벡터



Vocabulary size = N

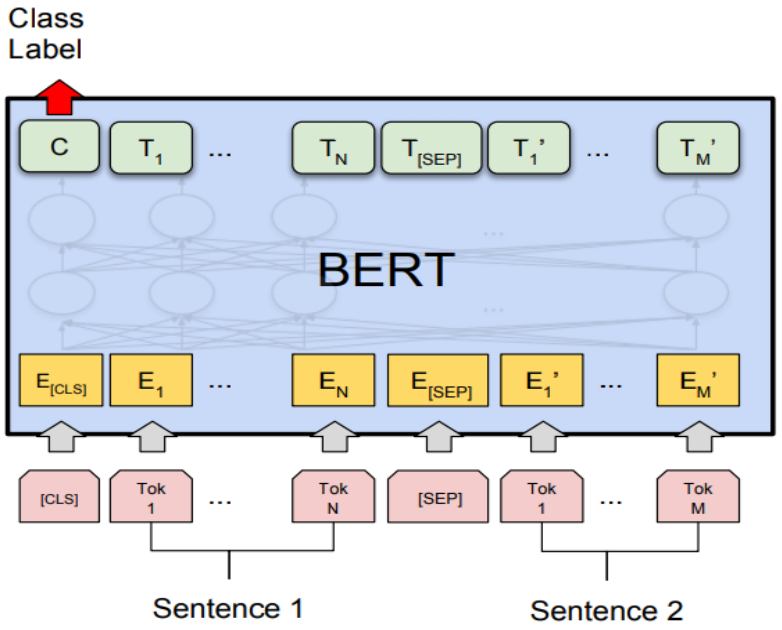
BERT 학습

- Next Sentence Prediction (NSP)
  - 두 개의 문장을 입력 받아서, 실제로 두 개의 문장이 연속된 문장들인지, 그렇지 않은지를 예측
  - 이진 분류 문제 (Binary classification)
    - 연속된 문장이면, Label = 'isnext‘
    - 아니라면, Label = ‘notnext’

Example	Label
[CLS] the man went to [MASK] store [SEP] he bought a gallon [MASK] milk [SEP]	IsNext
[CLS] the man [MASK] to the store [SEP] penguin [MASK] are flight ##less birds [SEP]	NonNext

BERT 학습

- Next Sentence Prediction (cont'd)
  - 학습데이터의 50%는 실제로 연결된 두문장으로 구성이 되고, 나머지 50%에 대해서는 두 번째 문장으로 랜덤하게 선택된 문장이 사용
  - [CLS]의 hidden state인 C가 사용됨



BERT fine-tuning

- Use of the pre-trained BERT model for several downstream tasks
  - Examples of such downstream tasks used in the paper
    - Paraphrasing
    - Natural language inference, a.k.a., entailment
      - 자연어 추론이란?
        문장 A(premise)가 주어졌을 때 또 다른 문장 B(hypothesis)가 문장 A에 의해 추론될 수 있는 것인지(entailment), 문장 A와 모순된 것인지(contradiction), 아니면 참/거짓을 알 수 없는 것인지(neutral)인지를 판단하는 것을 의미
    - Question-answering
    - Text classification (e.g., sentiment analysis)
  - 사용된 학습데이터셋
    - GLUE (General Language Understanding Evaluation), SQuAD (Stanford Question Answering Dataset), SWAG (Situations With Adversarial Generations, for NLI task)
      - GLUE는 여러개의 dataset 포함 ⇒ Entailment 관련 데이터셋인 MNLI (Multi-Genre Natural Language Inference), 질의-응답 데이터셋인 QNLI (Question Natural Language Inference), 감성분석 데이터셋인 SST-2 (Stanford Sentiment Treebank) 등

BERT fine-tuning

- 각 토큰의 어떠한 hidden state 벡터 정보를 사용하는가?
  - $\text{BERT}_{\text{BASE}}$의 경우, 12개의 encoder block를 사용
  - 각 encoder block에서 각 토큰에 대해서 768차원의 hidden state 벡터 정보를 출력 ⇒ 그렇다면, 각 단어마다 (혹은 토큰마다) 12개의 hidden state 벡터가 존재
    - 그럼 최종적으로 각 단어의 임베딩 벡터로 우리는 무엇을 사용하면 될까요?
    - BERT 논문에서는 마지막 4개의 encoder block에서 출력하는 hidden state 정보들을 사용하였음

BERT에서의 단어 embedding

- Context-based embedding
  - Word2vec이나 FastText의 경우도 문맥 정보 (즉, 주변 단어 정보)를 사용하기는 하나, 한계 존재
    - 즉, 동음이의어를 구분하지 못함
    - 예) '나는 과일 중에서 **사과**를 좋아합니다'에서의 사과와 '내가 어제일에 대해서 너에게 **사과** 할게'에서의 사과는 다른 의미를 갖는 단어들이지만 word2vec 경우는 동일한 임베딩 정보를 출력
  - BERT의 경우는 단어의 의미에 따라 다른 embedding vector (즉, **hidden state vector**) 출력
    - 주변 단어의 정보를 사용하는 self-attention 방법을 사용하여 hidden state 벡터 정보를 계산하기 때문
    - 추가적으로 positional embedding 정보도 사용

Python coding

- BERT를 사용해서 단어 임베딩 정보 얻기
  - Hugging Face에서 제공하는 Transformers 사용하기
    - https://huggingface.co/transformers/
    - pip install transformers
  - Transformer 기반의 알고리즘 구현
  - BERT의 경우에도 downstream task가 무엇인지에 따라서 사용할 수 있는 클래스가 별도로 존재
    - https://huggingface.co/transformers/model_doc/bert.html
  - Python code: "BERT_word_embeddings_TF.ipynb" 참고

Python coding

- BERT_word_embeddings_TF.ipynb
  - 입력 텍스트 데이터 준비
    - text = "These days word embeddings are important.“
    - Tokenization
      - WordPiece Tokenizer 사용
      - tokenized_text = tokenizer.tokenize(text)
      - 결과 ⇒ ['these', 'days', 'word', 'em', '##bed', '##ding', '##s', 'are', 'important', '.']
      - embeddings ⇒ 'em', '##bed', '##ding', '##s‘
        - WordPiece는 30000개 정도의 토큰으로 구성된 어휘사전을 사용

Python coding

- BERT_word_embeddings_TF.ipynb
  - WordPiece tokenizer 작동 방식
    - 일단 하나의 단어가 어휘 사전에 있는지를 확인
    - 만약 없다면 해당 단어를 subword로 split, 그렇게 쪼갠 결과인 단어의 일부가 해당 어휘 사전에 있다면 그 것을 하나의 토큰으로 간주
    - 만약 그렇게 해서도 찾지 못한 단어 혹은 단어의 일부는 더 쪼개서 개별 문자로 구분
    - 그렇게 찾아진 subword나 character가 다른 단어의 일부인 경우에는 앞에 ##를 붙여서 표현

Python coding

- BERT_word_embeddings_TF.ipynb
  - 동음이의어의 임베딩 벡터가 다르게 됨

text1 = "After stealing money from the **bank** vault, the **bank** robber was seen fishing on the Mississippi river **bank**."

inputs = tokenizer(text1, return_tensors="tf", max_length=100, padding='max_length')

inputs.keys()
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])

- input_ids: 토큰 아이디
- token_type_ids: 문장 아이디
- attention_mask: 토큰 or 마스킹

Python coding

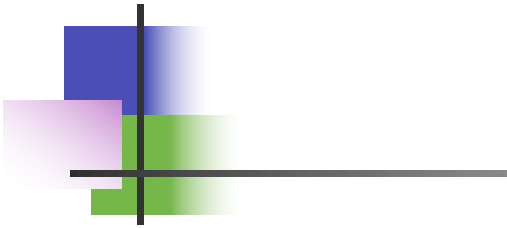
- BERT를 이용한 감성분석: 영어
  - 방법1: Feature-based approach
    - BERT_En_movie_review_sentiment_feature_based.ipynb
    - 이를 위해서는 Tensorflow 기반의 BERT 모형 사용 (TFBertModel)
  - 방법2: Fine-tuning approach
    - BERT_En_movie_reviews_sentiment_fine_tuning.ipynb

Python coding

- BERT를 이용한 감성분석: 한글
  - 크게 2가지 방법 존재
    - 방법1: BERT의 다국어(multilingual) 버전 이용하는 것
      - BERT_multi_Kor_movie_reviews_sentiment_feature_based.ipynb
      - BERT_multi_Kor_movie_reviews_sentiment_fine_tuning.ipynb
    - 방법2: 한글 학습데이터를 이용해서 학습한 모형을 사용하는 것
      - KoBERT, KcBERT, KLUE-BERT 등
        - BERT_Kor_movie_reviews_sentiment_fine_tuning_KcBERT.ipynb
        - BERT_Kor_movie_reviews_sentiment_fine_tuning_KLUE.ipynb
      - KcELECTRA
        - ELECTRA를 설명한 후 코드 설명

Variants of BERT

- 다음과 같은 변형된 버전 존재
  - ALBERT: A Lite version of BERT
    - Cross-layer parameter sharing
    - Factorized embedding layer parameterization
  - RoBERTa: Robustly Optimized BERT-Pretraining Approach
    - 정적 마스킹 (static masking)이 아닌 동적 마스킹 (dynamic masking) 방법 사용
    - 더 많은 학습 데이터 사용
    - NSP (next sentence prediction) task를 사용하지 않음
    - WordPiece tokenizer가 아닌 byte-level Byte-Pair Encoding tokenizer 사용
  - ELECTRA: Efficiently Learning an Encoder that Classifies Token Replacements Accurately
    - MLM 대신 Replaced Token Detection (RTD)이라고 하는 작업을 통해서 학습
    - NSP 작업은 학습에서 사용하지 않음
  - DeBERTa
    - 토큰의 인코딩과 위치 인코딩을 더하지 않고 구분해서 사용 (Disentangled Attention)
  - 지식 증류 기반
    - DistilBERT 등



Q & A