Artficial Intelligence Laboratory

Ch2. Fine-Tuning BERT Models

Transformers for NLP

정보융합공학과 AI전공
정주경

부산대학교
PUSAN NATIONAL UNIVERSITY

목 차

The architecture of BERT

BERT(Bidirectional Encoder Representations from Transformers)

- Transformer 모델 중 encoder 부분만을 사용하여 encoder를 쌓아 올린 구조
- 문맥을 양방향으로 이해하는 딥러닝 모델
- 많은 데이터와 많은 컴퓨팅 자원 문제를 해결하기 위해

해결법이 유사한 문제를 다루는 모델(pre-trained model)을 기반으로 해결하고자 하는 문제(downstream task)를 해결할 수 있도록 추가 학습(fine-tuning)

- 대형 corpus에서 unsupervised learning으로 pre-training을 구축하고 supervised learning으로 fine-tuning해서 downstream nlp task에 적용하는 semi-supervised learning모델

BERT의 크기

L : 트랜스포머 인코더 층의 수

D : d_model의 크기

A : 셀프 어텐션 헤드의 수

- BERT-Base : L = 12, D = 768, A = 12 : 110M개의 파라미터
- BERT-Large : L = 24, D = 1024, A = 16 : 340M개의 파라미터
- Transformer : L = 6, D = 512, A = 8

BERT-Base는 GPT-1과 동등한 크기로 성능 비교를 위해

BERT-Large는 BERT의 최대 성능을 보이기 위해

The architecture of BERT

Encoder와 Decoder의 차이

- Encoder : 양방향으로 문맥 이해
- Decoder: 왼쪽에서 오른쪽으로 문맥 이해

BERT 모델은 디코더 스택이없어 masked multi-head attention sub-layer가 없다.

Masked multi-head attention layer는 예측해야할 출력을 보지 못하도록 현재 위치를 넘어선 모든 토큰들을 mask

ex)

The cat sat on it because it was a nice rug.

단어 “it”에 도달하면 encoder의 입력부분은

The cat sat on it<masked sequence>

하지만 “it”이 참조하는 것이 뭔지 알려면, “rug”에 도달해서 전체 문장을 봐야한다.

BERT는 self-attention을 이용해서 모든 토큰을 한번에 계산하여 양방향으로 문맥을 이해

BERT Input Representation

BERT의 서브워드 토크나이저 : WordPiece

BERT는 단어보다 더 작은 단위로 쪼개는 서브워드 토크나이저를 사용.

서브워드 토크나이저 : 자주 등장하는 단어는 단어 집합에 추가하지만, 자주 등장하지 않는 단어의 경우에는 더 작은 단위인 서브워드로 분리되어 단어 집합에 추가

BERT에서 토큰화 수행 방식

1. 토큰이 단어 집합에 존재 -> 해당 토큰을 분리하지 않는다.
2. 토큰이 단어 집합에 존재X -> 해당 토큰을 서브워드로 분리, 해당 토큰의 첫 번째 서브워드를 제외한 나머지 서브워드들은 앞에 '##'를 붙인 것을 토큰

2가지 장점

1. Playing : Play(놀다) + ##ing(현재 진행)처럼 의미를 명확히 전달
2. 신조어 또는 오탈자가 있는 흔치 않는 단어들에 대한 예측이 향상 ex) texting, googling

BERT Input Representation

Position Embedding

- **Transformer**

Positional encoding 방법을 통해 단어의 위치 정보를 표현

=> Sin함수와 cos함수를 사용하여 위치에 따라 다른 값을 가지는 행렬을 만들어 이를 단어 벡터들과 더하는 방법

- **BERT**

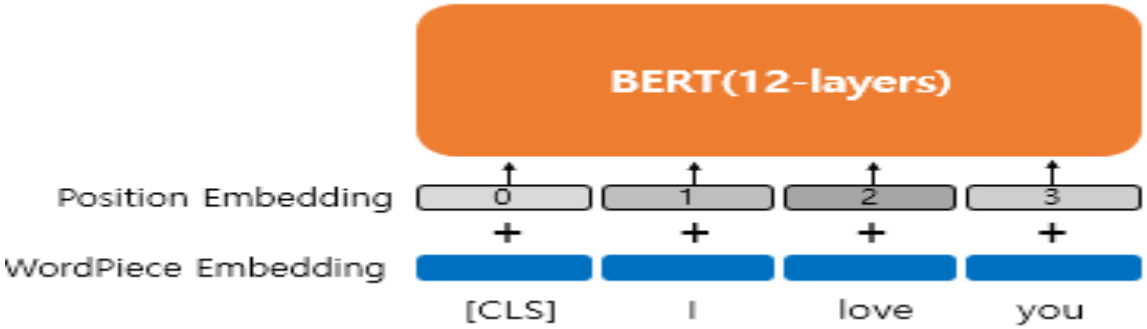
Sin함수와 cos함수로 만드는 것이 아닌 학습을 통해서 얻는 Position Embedding 방법 사용

위치 정보를 위한 Embedding Layer를 하나 더 사용.

가령 문장의 길이가 4라면 4개의 포지션 임베딩 벡터를 학습.

BERT의 입력마다 포지션 임베딩 벡터를 더해줌

실제 BERT에서는 문장의 최대 길이가 512로 총 512개의 포지션 임베딩 벡터가 학습



BERT Input Representation

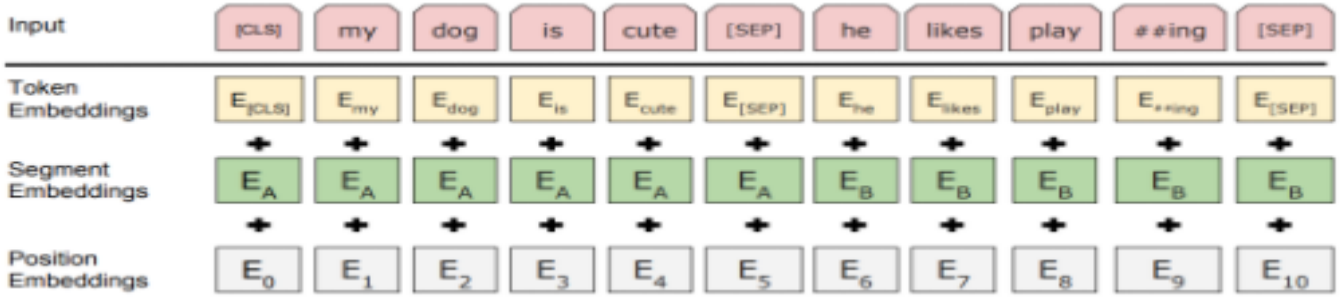
Segment Embedding

QA 등과 같은 두 개의 문장 입력이 필요한 task.

문장 구분을 위해서 BERT는 Segment Embedding 이라는 Embedding layer를 사용.

첫 번째 문장에는 Sentence 0 임베딩, 두 번째 문장에는 Sentence 1 임베딩을 더해주는 방식.

임베딩 벡터는 두 개만 사용



- **Token Embedding : 실질적인 입력이 되는 워드 임베딩. 임베딩 벡터의 종류는 단어 집합의 크기로 30,522개**
- **Segment Embedding : 두 개의 문장을 구분하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 개수인 2개**
- **Position Embedding : 위치 정보를 학습하기 위한 임베딩. 임베딩 벡터의 종류는 문장의 최대 길이 512개**

여기서 문장이라는 것은 실제로는 두 종류의 텍스트, 두 개의 문서가 될 수 있다.

BERT Input Representation

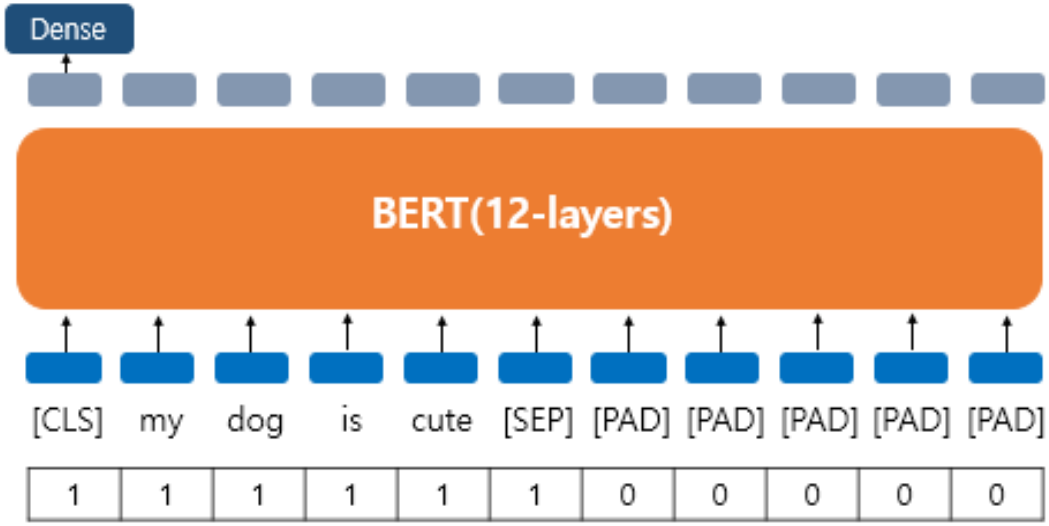
Attention Mask

BERT를 실제로 실습하게 되면 Attention mask sequence 입력이 추가로 필요

불필요하게 패딩 토큰에 대해서 attention을 하지 않도록 실제 단어와 패딩 토큰을 구분할 수 있도록 알려주는 입력

숫자 1은 해당 토큰을 실제 단어이므로 마스킹을 하지 않는다

숫자 0은 해당 토큰은 패딩 토큰이므로 마스킹을 한다



Pretraining and Fine-tuning a BERT model

Step 1 Pretraining

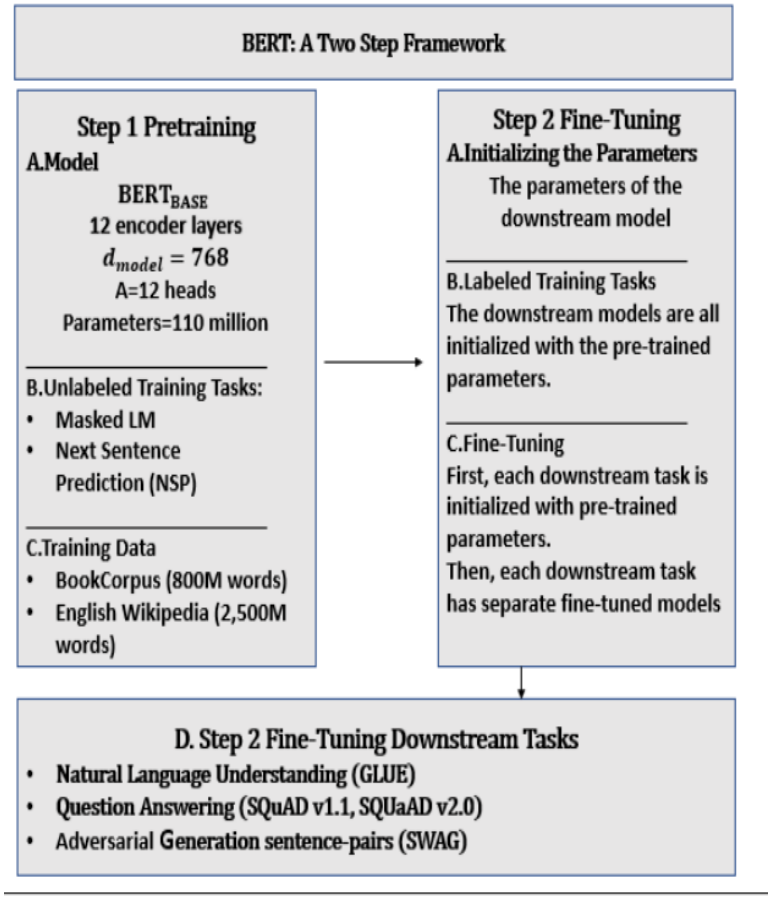
- Unlabled data로부터 모델을 사전학습시키는 단계
- 모델의 아키텍쳐 정의: layers와 head와 dimension의 숫자
- Masked Laguage Modeling(MLM), NSP task
- Training data

=> 위키피디아(25억 단어) + BooksCorpus(8억 단어)

Step 2 Fine-Tuning

- Pretrained된 BERT모델의 학습된 parameter들을 사용하여 선택한 downstream 모델 초기화
- Recognizing Textual Entailment (RTE), Question Answerting(SQuAD v1.1, SQuAD v2.0)등과 같은 특정한 downstream task에 대한 파라미터 파인튜닝

fine-tuning할 BERT model은
The Corpus of Linguistic Acceptability(CoLA)
에서 학습되었다.



Pre-training BERT

Masked Language Modeling

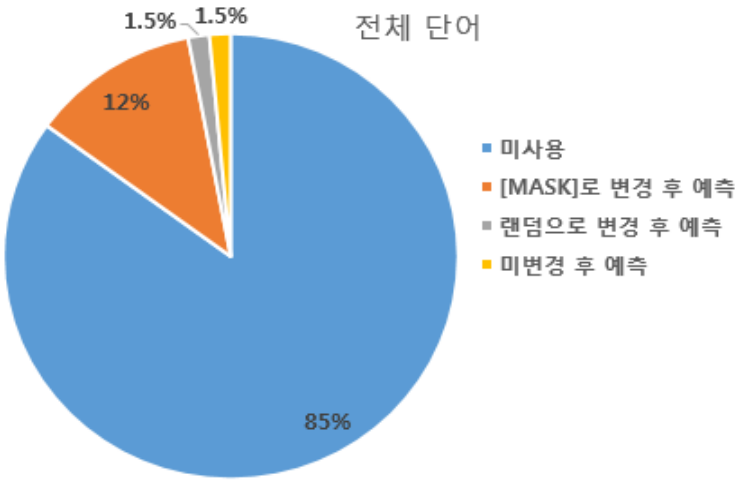
입력 텍스트의 15%의 단어를 랜덤하게 masking -> self-attention한 후
Masked token 예측하여 전체적인 문장에 대한 문맥 정보를 파악

- The cat sat on it because it was a nice rug.
- The cat sat on it [Mask] it was a nice rug.

선택된 15%의 단어들은 다시 다음과 같은 비율로 규칙이 적용

- 80%의 단어들은 [Mask]로 변경
- 10%의 단어들은 랜덤으로 단어가 변경
- 10%의 단어들은 미변경

Why? [Mask]만 사용할 경우에 [Mask]토큰이
파인 튜닝 단계에서는 나타나지 않으므로
사전 학습 단계와 파인 튜닝 단계에서의 불일치가 발생
모든 토큰을 [Mask]로 사용하지 않는다.



Pre-training BERT

다음 문장 예측 (Next Sentence Prediction, NSP)

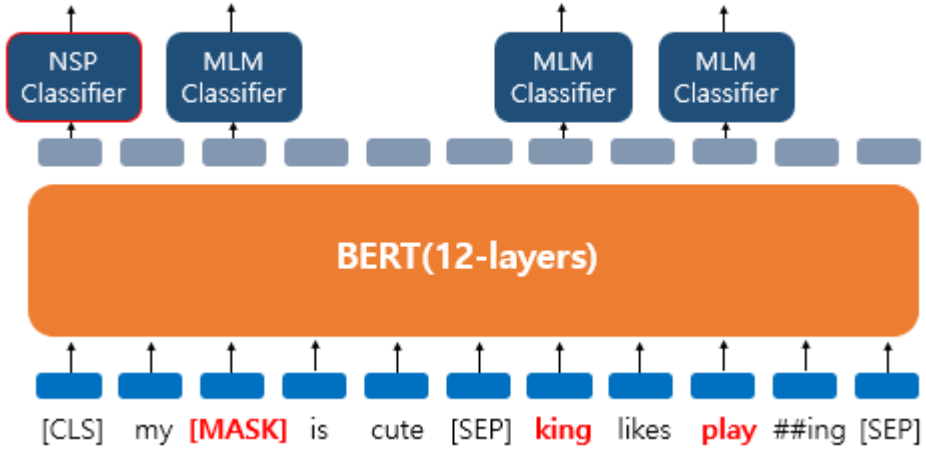
두 문장이 이어지는 문장인지 아닌지를 맞추는 방식의 훈련

50:50의 비율로 실제 이어지는 두 개의 문장과 랜덤으로 이어붙인 두 개의 문장 훈련

- [CLS] : 두 문장이 실제 이어지는 문장인지 아닌지를 여부를 예측하기 위해 첫 번째 sequence의 시작 부분에 추가. 출력층에서 이진 분류 문제를 풀도록 추가된 특별 토큰
- [SEP] : sequence의 끝을 표시하는 separation 토큰. 문장을 구분

MLM model과 NSP는 따로 학습하는 것이 아닌 loss를 합하여 동시에 학습

다음 문장 예측이라는 task를 학습하는 이유는 QA(Question Answering)나 NLI(Natural Language Inference)와 같이 두 문장의 관계를 이해하는 것이 중요한 task들이 있기 때문



Pre-training BERT

Detail

실제 Pre-training 단계는 다음과 같이 진행된다

Corpus에서 문장들을 뽑아내 두 문장의 sequence로 만들고, 각각 A, B를 Segment Embedding으로 부여한다. 50%의 확률로 B 문장은 실제로 A문장에 이어지는 문장이고(IsNext), 50%의 확률로 A문장에 이어지지 않는 문장이다(NotNext). 이후 Word Piece Tokenization을 한 뒤, Masking을 수행한다.

Hyperparameters는 다음과 같다.

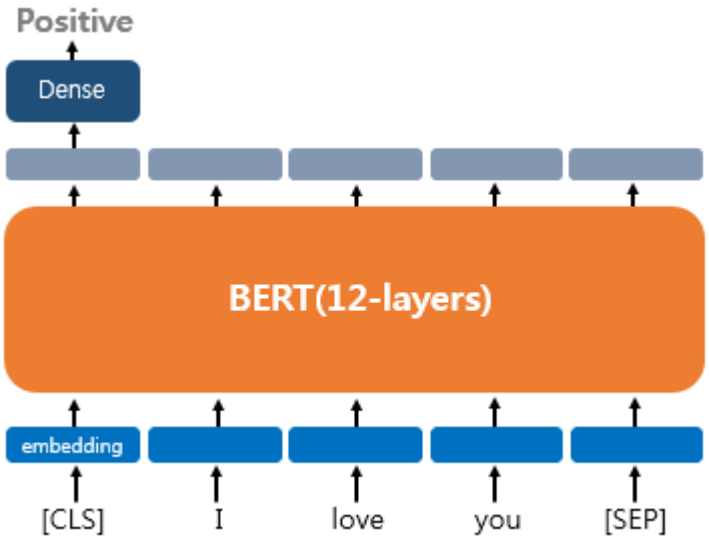
- Batch size = 256 sequences
- Sequnce size = 512 tokens
- #Epoch = 40
- Optimizer : Adam(learning rate = 1e-4, B_1 = 0.9, B_2=0.999, L2 weight decay = 0.01)
- Dropout :0.1 in all layers
- Activation function : gelu
- Loss function : sum of the mean MLM likelihood + mean NSP likelihood

Pre-training에 BERT_BASE는 16개의 TPU로 4일, BERT_LARGE는 64개의 TPU로 4일이 소요

Fine-Tuning BERT

사전학습된 BERT에 우리가 풀고자 하는 task(downstream task)의 데이터를 추가로 학습시켜서 테스트 하는 단계

Single Text Classification



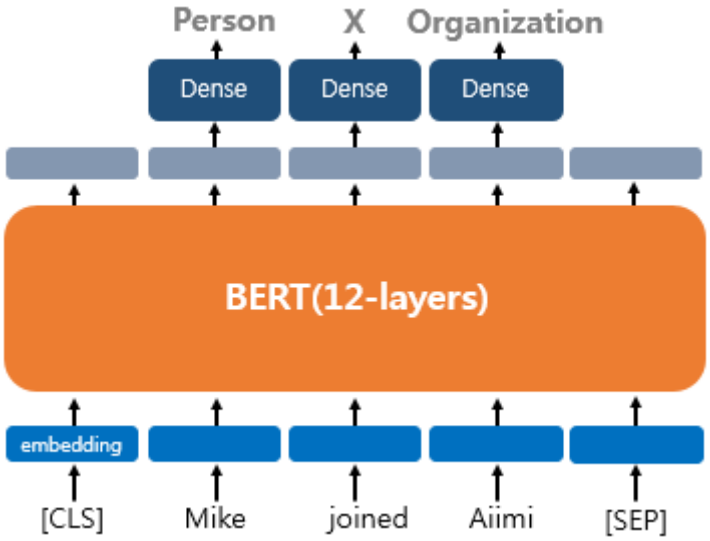
하나의 문서에 대한 텍스트 분류.

감성 분류등과 같이 입력된 문서에 대해서 분류를 하는 유형으로, 시작에 [CLS] 입력

[CLS] 토큰의 위치의 출력층에서 Dense layer 또는 fully-connected layer라고 불리는 층들을 추가하여 분류에 대한 예측

Fine-Tuning BERT

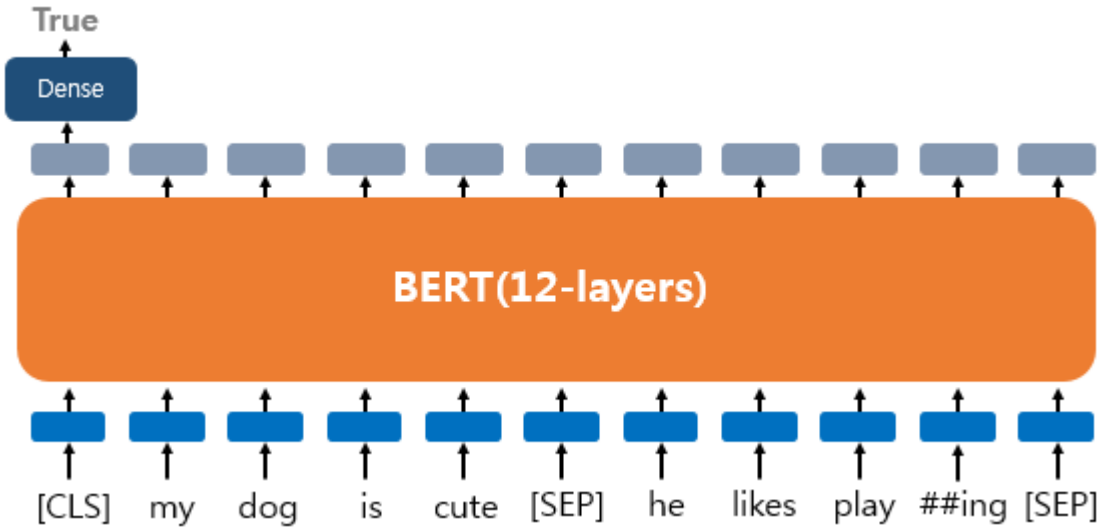
- 하나의 텍스트에 대한 태깅 작업(Tagging)



문장의 각 단어에 품사를 태깅하는 품사 태깅 작업과 개채를 태깅하는 개체명 인식 작업
출력층에서는 입력 텍스트의 각 토큰의 위치에 밀집층을 사용하여 분류에 대한 예측을 하게 된다.

Fine-Tuning BERT

텍스트의 쌍에 대한 분류(Text Pair Classification)



텍스트의 쌍을 입력으로 받는 대표적인 task로 자연어 추론(Natural Language inference)

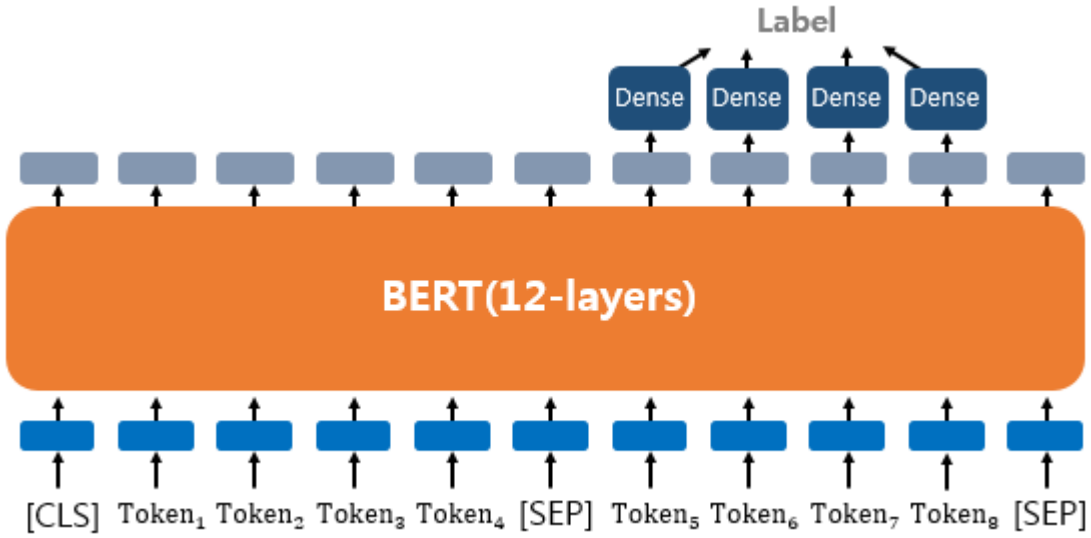
자연어 추론이란 두 문장이 주어졌을 때, 하나의 문장이 다른 문장과 논리적으로 어떤 관계에 있는지를 분류.

유형으로는 모순 관계(contradiction), 함의 관계(entailment), 중립 관계(neutral)

입력 텍스트가 2개 이므로, 텍스트 사이에 [SEP] token을 집어넣고, Sentence 0 임베딩과 Sentence 1 임베딩 두 종류의 Sentence 임베딩을 모두 사용하여 문서를 구분

Fine-Tuning BERT

질의 응답(Question Answering)



QA를 풀기 위해 질문과 본문이라는 두 개의 텍스트의 쌍을 입력.
대표적인 데이터셋으로 SQuAD(Stanford Question Answering Dataset) v1.1
질문과 본문을 입력받으면, 본문의 일부분을 추출해서 질문에 답변.

Fine-Tuning BERT

Detail

Fine-tuning은 Pre-training에 비해 매우 빠른 시간 내에 완료된다. Fine-tuning에서는 각각의 task에 specific하게 input size, output size 등을 조절해야 한다. 또한 token-level task일 경우에는 모든 token들을 사용하고, sentence-level task일 경우에는 CLS token을 사용한다.

Hyperparameters는 대부분은 Pre-training과 동일하게 진행하는 것이 좋지만,
batch size, learning rate ,#epochs는 task-specific하게 결정해야 한다.
하지만 다음의 값들에 대해서는 대체적으로 좋은 성능을 보였다.

- Batch size : 16, 32
- Learning rate : 5e-5, 3e-5, 2e-5
- #epochs : 2, 3, 4

Dataset의 크기가 클 수록 Hyperparameter의 영향이 줄어들었으며, 대부분의 경우 Fine-tuning은 매우 빠른 시간 내에 완료되기 때문에 많은 parameters에 대해 테스트를 진행할 수 있었다.

Evaluating

Evaluating using Matthews Correlation Coefficient

MCC는 이진(binary) 분류의 퀄리티를 측정하기위해 사용되는 방법으로

CoLA의 성능은 MCC로 계산

-1과 1사이의 값을 가지며, 1에 가까울수록 측정하고자 하는 두 관측치가 비슷하다고 볼 수 있다.

- TP=True Positive
- TN=True Negative
- FP=False Positive
- FN=False Negative

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

4개의 혼동 행렬 범주의 균형 비율을 고려하기 때문에 이진 분류 문제를 평가할 때 F1 점수 및 정확도보다 더 많은 정보를 제공

하지만 최종 점수 계산에서 혼동 행렬의 네 가지 클래스 크기를 완전히 고려하지 않기 때문에 완벽히 맞다고는 할 수 없다.

Evaluating

GLUE(The General Language Understanding Evaluation benchmark)

강건하고 범용적인 자연어 이해 시스템의 개발이라는 목적을 가지고 제작된 데이터셋

GLUE 내 9개의 task에 각각 점수를 매게 최종 성능 점수를 계산

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**

SQuAD(The Standford Question Answering Dataset) v1.1

질문과 문단이 주어지고 그 중 substring인 정답을 맞추는 task.

System	Dev EM	Dev F1	Test EM	Test F1
Top Leaderboard Systems (Dec 10th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
Published				
BiDAF+ELMo (Single)	-	85.6	-	85.8
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
$BERT_{BASE}$ (Single)	80.8	88.5	-	-
$BERT_{LARGE}$ (Single)	84.1	90.9	-	-
$BERT_{LARGE}$ (Ensemble)	85.8	91.8	-	-
$BERT_{LARGE}$ (Sgl.+TriviaQA)	**84.2**	**91.1**	**85.1**	**91.8**
$BERT_{LARGE}$ (Ens.+TriviaQA)	**86.2**	**92.2**	**87.4**	**93.2**

Evaluating

Effect of Pre-training Tasks

No NSP : MLM은 사용하지만, 다음 문장 예측(NSP)를 없앤 모델

LTR & No NSP : MLM대신 Left-to-Right(LTR)을 사용하고, NSP도 없앤 모델, 이는 OpenAI GPT 모델과 완전히 동일하지만, 더 많은 트레이닝 데이터를 사용

	Dev Set				
Tasks	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
$\text{BERT}_{\text{BASE}}$	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

- Pre-training task를 하나라도 제거하면 성능이 떨어지는 것을 볼 수 있다.
- No NSP의 경우에는 NLI계열의 task에서 성능이 많이 하락하게 되는데, 이는 NSP task가 문장 간의 논리적인 구조파악에 중요한 역할을 하고 있음을 알 수 있다.
- MLM대신 LTR을 쓰게 되면 성능하락. BiLSTM을 더 붙여도, MLM을 쓸 때보다 성능이 하락하는 것으로 보아, MLM task가 더 Deep Bidirectional한 것임을 알 수 있다.

Summary

- BERT는 transformer의 인코더 스텍만을 사용하여 동시에 sequence의 모든 token에 주의를 기울인다
  1. Pre-training
  - 해결 하고자 하는 문제와 비슷한 문제를 다루는 모델
  - Input = Token Embedding + Segment Embedding + Position Embedding
  - Masked Language Model
  - Next Sentence Prediction
  2. Fine-tuning
  - Pre-training 모델을 기반으로, 해결하고자 하는 문제에 맞춰 학습
- Fine-tuning의 첫 단계는 dataset을 load하고 모델에 필요한 사전 학습된 모듈들을 load
- 모델이 학습되면 성능 측정
- Pretrained model을 fine-tuning 하는 것은 downstream task를 처음부터 훈련 하는 것 보다 기계 자원이 적게 소요
- Fine-tuned models은 다양한 task를 수행할 수 있다

➢ **한계점**

일반 NLP모델에서 잘 작동하지만, Bio, Science, Finance 등 특정 분야의 언어모델에 사용하려면 잘 적용되지 않는다. 사용 단어들이 다르고 언어의 특성이 다르기 때문. 특정 분야의 특성을 반영할 수 있는 언어데이터들을 수집하고, 언어 모델 학습을 추가적으로 진행해 주어야 한다.