

BERT (Bidirectional Encoder Representations from Transformers)

(<https://arxiv.org/pdf/1810.04805.pdf>)

(<https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f>)

1. Bert Model

- : Language Model의 학습 목표가, 다음단어 예측의 모델 경우처럼, 한쪽 방향으로만의 경향성을 학습하는 것이 문맥 학습(context learning)에 방해가 된다는 관점.
- : 따라서, **방향성 없이(non-directional 또는 bi-directional) 학습 시키는 방법이 유리하다 판단**
- : Transformer의 Decoder 부분은 기본적으로 Left-to-right 방향을 갖는 모델이므로 제외.
그래서 Encoder-only 모델이 자연스러움.
- : **문장 내부의 미묘한 의미와 문장 간의 관계를 모두 포괄적으로 이해할 수 있는 모델이 목표**

● Transformer block 기반의 Encoder-only 모델 (stacked encoders)

- text classification, next sentence prediction, Named-Entity-Recognition (NER), or question-answering 등의 task에 사용

※ [Jim]_{Person} bought 300 shares of [Acme Corp.]_{Organization} in [2006]_{Time}.

● 학습데이터에 대한 bidirectional training 과 WordPiece Tokenizer 사용

- 특정 언어의 'language model'을 학습하는 **Pre-training**과, 전이학습(Transfer learning)을 이용한 특정 task를 학습하는 **Fine-tuning**으로 구성

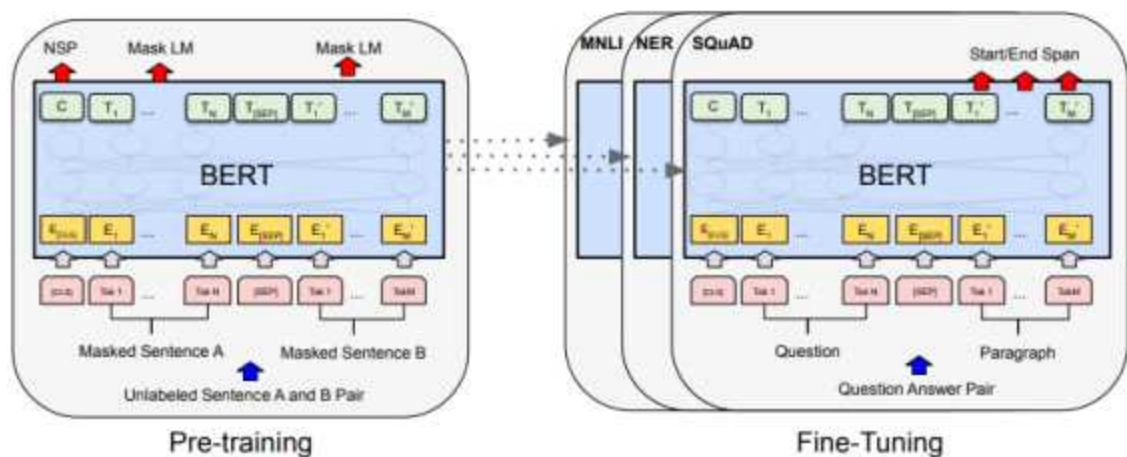
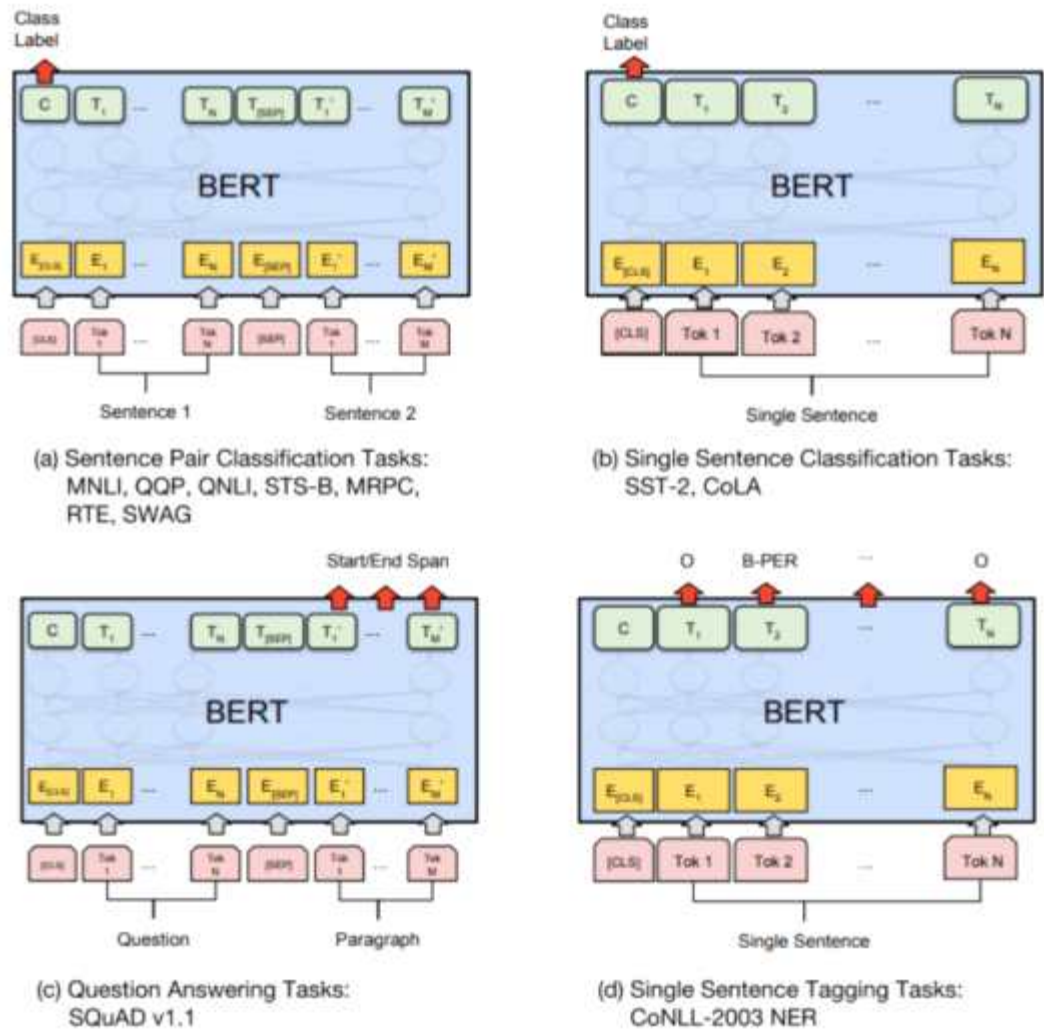


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

● Bert가 가능한 Tasks

(<https://arxiv.org/pdf/1810.04805.pdf>)



● Bert 성능(2018)

(<https://arxiv.org/pdf/1810.04805.pdf>)

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

● BERT의 성능이 좋은 이유

1. It is pre-trained on unlabeled data extracted from **BooksCorpus**, which has 800M words, and from **Wikipedia**, which has 2,500M words.

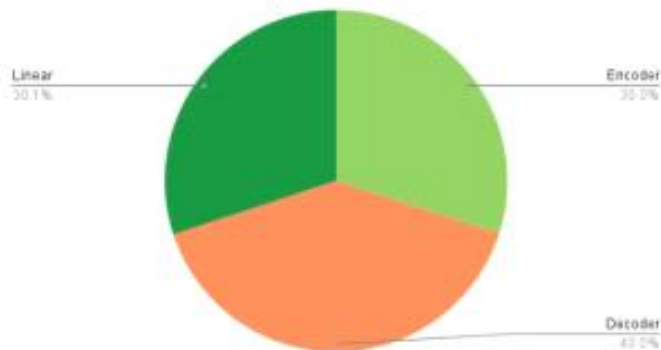
※ BookCorpus is a large collection of free novel books written by unpublished authors, which contains 11,038 books (around 74M sentences and 1G words) of 16 different sub-genres (e.g., Romance, Historical, Adventure, etc.)

2. As the name suggests, it is pre-trained by utilizing the bidirectional nature of the encoder stacks. This means that BERT learns information from a sequence of words not only from left to right, but also from right to left.

● Original Transformer 모델별 파라미터 크기 비교

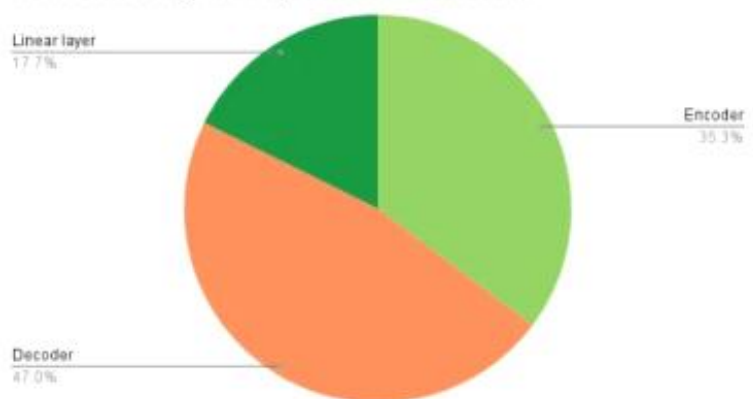
: 모델크기가 커지면 Decoder 부분이 Encoder 부분보다 상대적으로 더 커진다

Transformer base model parameter distribution



Distribution of the base model's parameter

Transformer big model parameter distribution

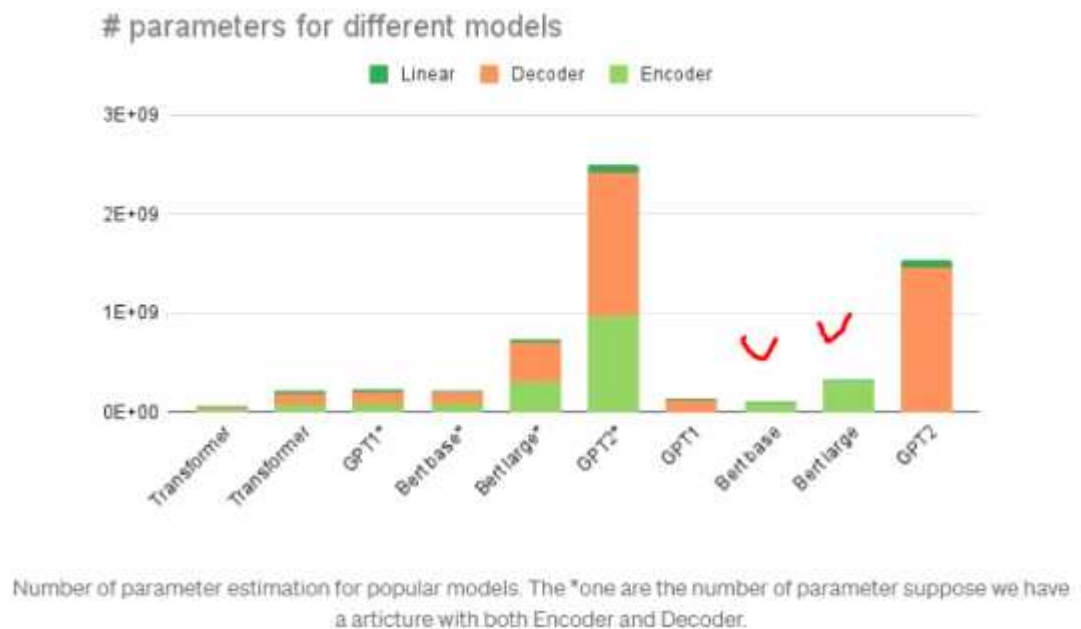


● Bert의 구조

	Transformer	Bert_Base	Bert_Large
Multi Heads #	8	12	16
Embedding Dimension	512	768	1024
Max. Token Size	512	512	512
repeated layers #	6	12	24
Parameters #	62 M(Base) / 214 M(Large)	110 M	340 M

● Transfor 기반 대표적 모델들 간의 파라미터 크기 비교

Estimation of other models based on the calculator



● Special Tokens in Input Representation of BERT

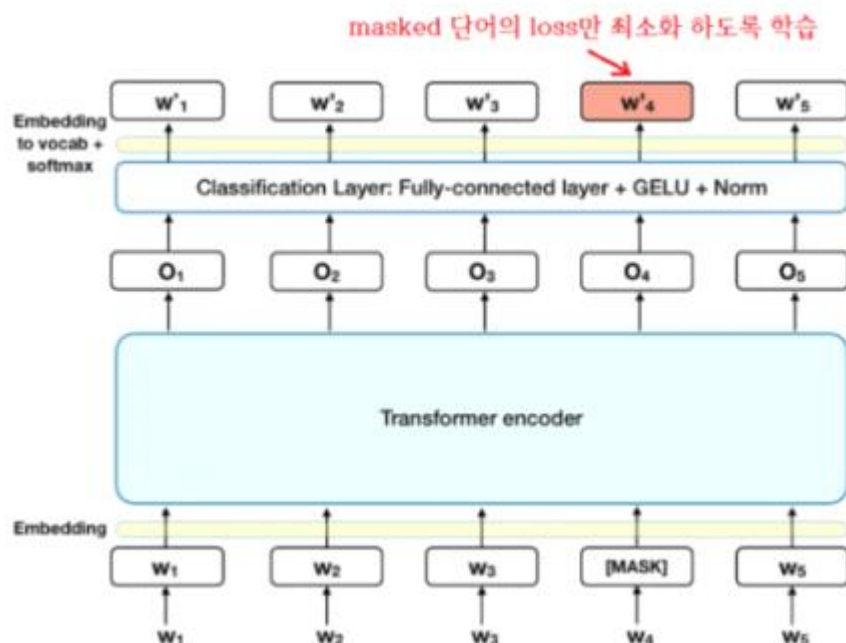
- [CLS] classification 토큰 : 모든 sentence의 첫 번째 토큰
 “The cat is walking. The dog is barking”
 --> “[CLS] the cat is walking [SEP] the dog is barking”
- [SEP] separate 토큰 : 모든 sentence의 마지막 토큰, 두 개의 연속 sentence 사이의 토큰
 ‘다음문장 예측’ 문제나 ‘question-answering’ 문제에 중요한 역할
- [PAD] padding 토큰 : maximum token 길이보다 짧은 문장의 나머지 자리들을 대체하는 토큰
- [MASK] masking 토큰 : 특정 단어를 비우거나 할 때 대신하는 토큰
- [END] End of sentence 토큰

2. BERT의 2가지 기본 학습(Pre-Training) 전략

- Masked Language Model(문장내 문맥 학습) and Next sentence Prediction(문장 간 문맥 학습)을 통한 문맥 이해력 향상
- 학습시에 이 두가지 학습에 대한 loss를 결합하여 손실을 계산하면서 동시에 학습함

(학습 전략 1) Masked LM (MLM)

- 특정(Masked) 단어 기준으로, 문장내 전체 단어들을 고려한 문맥 학습이 가능하도록, [masked] 단어를 맞추도록 학습
 - Before feeding word sequences into BERT, randomly chosen 15% of the words the whole texts are replaced with a [MASK] token, first
 - ※ 모든 문장에 [MASK] 단어를 쓰면 문장의 문맥 학습에 지장을 줄 수 있다는 생각.
 - The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence.
 - 학습을 위해 encoder 상단(출력) 부분에 classification layer를 추가.
 - masked 단어의 loss 만 계산. (masking 하지 않은 단어에 대한 loss는 계산하지 않음)
 - mask 단어로 선택된 15%의 단어들에 대해서, 이 중,
 - 80%는 실제로 [MASK] 라는 특수 기호로 대체,
 - 10%는 임의의 random 단어로 대체,
 - 10%는 원래 문장의 단어를 그대로 살려두는 식으로 입력 벡터를 구성한다.
- ex) I am [MASK] boy (80 %), [MASK]로 대체
She is at [MASK]. I just texted him.
She loved and with chocolate like (10 %), random words
She is at home. I just texted him. (10 %), no mask



(학습 전략 2) Next Sentence Prediction (NSP)

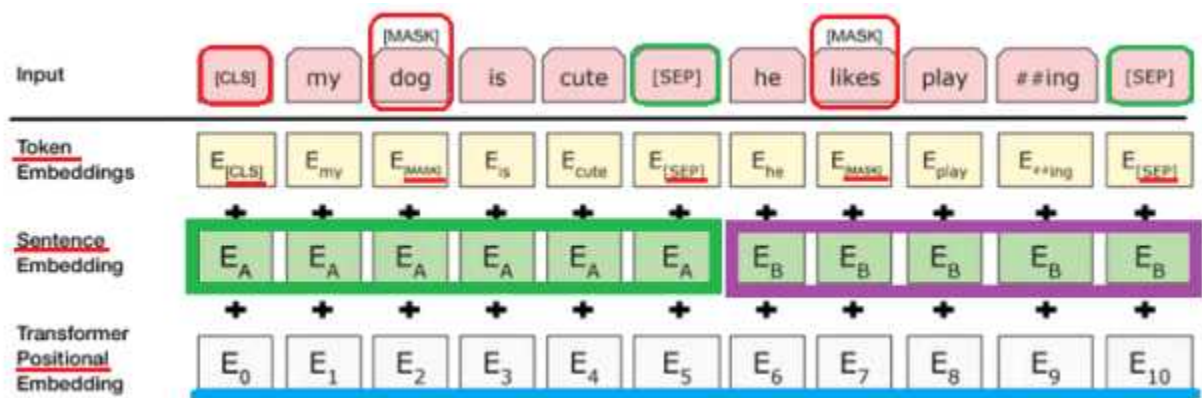
- 실제 본문에서 연속된 2개의 문장 쌍들을 50%, 연속되지 않은 서로 떨어져 있는 random하게 뽑은 2개의 문장을 붙인 문장 쌍들을 나머지 50%로 구성된 학습 데이터로 구성하여, 2개의 문장이 **연속된 문장인지 여부를 맞추도록 학습**

- 모든 문장의 앞부분에 [CLS]토큰을, 끝부분에 [SEP] 토큰을 삽입함
- 2개 문장을 구분하는 Sentence Embedding을 토큰에 더함
- Positional Embedding을 추가
- 출력 부분에 IsNextSequence 확률을 softmax로 계산
- MLM과 NSP를 동시에 학습 시킴

. Token Embedding : [CLS] [SEP] [SEP]

. Sentence Embedding : [a, a, ... a, b, b, ... b]

. Positional Embedding :



ex) I am [MASK] boy

(80 %), one mask

She is at [MASK]. I just texted him.

She loved [MASK] with chocolate [MASK]

(10 %), random masks

She is at home. I just texted him.

(10 %), no mask

3. BERT의 사용(Fine-Tuning)

● Classification (예: Sentiment Analysis)

- 출력 [CLS] 위에 classification layer 추가

● Question Answering

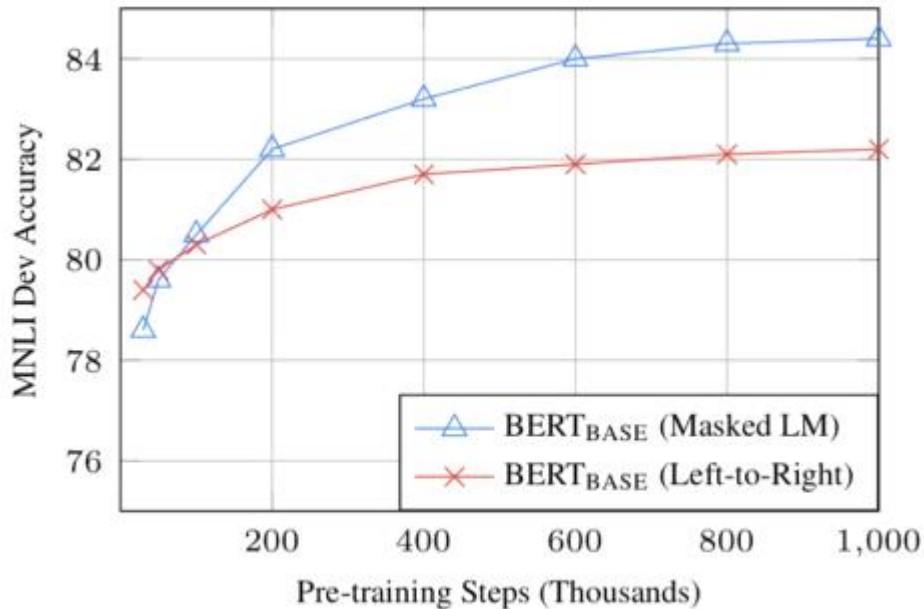
- 질문과 답의 시퀀스를 입력으로 받은 후, 답이되는 문장의 '시작'과 '끝' 위치를 학습

● Named Entity Recognition (NER)

- 출력 벡터의 각 토큰들이 NER label을 맞추도록 학습

4. BERT의 특징

- Model Size가 클수록 성능이 좋다
- 학습 데이터가 많을시에, training epoch가 클수록 성능이 좋다
 - 128,000 words batch size 일 때 1M epochs 만큼 학습 시켰을 때가 0.5M steps 학습 했을때 보다 1% 정확도가 높았음
- BERT는 문장 중 15% 단어만 예측하도록 학습해서, Left-to-Right 방식보다 수렴속도는 느렸지만, 이 미 적은 횟수의 Pre-training 단계부터 정확도는 앞선다



Source: BERT [Devlin et al., 2018]

	Training Compute + Time	Usage Compute
BERT _{BASE}	4 Cloud TPUs, 4 days	1 GPU
BERT _{LARGE}	16 Cloud TPUs, 4 days	1 TPU

- 작은 크기 BERT 모델들도 잘 작동함 (<https://github.com/google-research/bert>)

	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)

2. pytorch를 이용한 BERT IMDB Sentence Classification

2.1 Source Code

(<https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f>)

```
# Huggingface transformers를 이용한 Bert 모델 구현 [문장 분류]
# pip install transformers[torch] datasets torch
import torch, sys
from datasets import load_dataset
from transformers import Trainer, TrainingArguments
from transformers import BertForSequenceClassification
from transformers import BertTokenizer
from transformers import pipeline
# Check for GPU availability
device = 'cuda' if torch.cuda.is_available() else 'cpu'
print("Using device:", device)
#=====
def train_bert(totalEpochs=3):
    # Load dataset
    dataset = load_dataset('imdb', split=['train', 'test'])
    train_dataset, test_dataset = dataset
    # pre-trained tokenizer 설정
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    def tokenize_function(examples):
        return tokenizer(examples['text'], padding="max_length", truncation=True)
    # imdb 데이터 세트의 5%만 사용. 전체 사용시엔 이 부분을 comment 할 것
    """
train_dataset = train_dataset.shuffle(seed=42).select(range(int(0.05 * len(train_dataset))))
test_dataset = test_dataset.shuffle(seed=42).select(range(int(0.05 * len(test_dataset))))
"""

    train_dataset = train_dataset.map(tokenize_function, batched=True)
    test_dataset = test_dataset.map(tokenize_function, batched=True)
    # BERT pre-trained model 설정
    model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
    model.to(device)
    # 학습 파라미터 설정
    training_args = TrainingArguments(
        output_dir='./results',          # output directory
        num_train_epochs=totalEpochs,    # number of training epochs
        per_device_train_batch_size=8,    # batch size for training
        per_device_eval_batch_size=16,    # batch size for evaluation
        warmup_steps=500,                 # number of warmup steps for learning rate scheduler
        weight_decay=0.01,                # strength of weight decay
        logging_dir='./logs',             # directory for storing logs
        logging_steps=10,
        evaluation_strategy="epoch",       # evaluation is done at the end of each epoch
    )
    # trainer 설정
    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=test_dataset
    )
    # 모델 학습 시키기
    trainer.train()
    # 학습결과 저장
    model_path = "bert-finetuned-imdb" # 결과를 저장할 폴더명
    model.save_pretrained(model_path)
    tokenizer.save_pretrained(model_path)
#=====
```

```

# main, train or predict
#=====
showModelFlag =True
if __name__=="__main__":
    print('\n',' '*80)
    print('Usage [Training]: python bert_imdb.py  train  totalEpochNo[예:10]')
    print('Usage [Prediction]: python bert_imdb.py  predict  ')

    if len(sys.argv)>=2 :
        #----- 학습
        if 'train' in sys.argv[1] and int(sys.argv[2])>0:
            totalEpochs =int(sys.argv[2])
            train_bert(totalEpochs)
            # Test prediction using a sentence
        elif 'predict' in sys.argv[1]:
            print('\n*** Model loading.....')
            modelPath = 'bert-finetuned-imdb'
            model =BertForSequenceClassification.from_pretrained(modelPath)
            model =model.to(device)
            tokenizer =BertTokenizer.from_pretrained(modelPath)

            classifier =pipeline('text-classification', model=model,tokenizer=tokenizer, \
                                device=0 if device == 'cuda' else -1)
            print("*** Model loaded successfully from ",modelPath,'\n')

            if showModelFlag :
                print(model)

            while True:
                query_sentence =input("\n*** Type a sentence or 'quit' to stop: ")
                if 'quit' in query_sentence :
                    break
                print("==> query sentence:",query_sentence)
                predicted_output =classifier(query_sentence)
                print("==> Predicted output:",predicted_output,'\n')

```

2.2 실행결과 (Train)

python train 20

```
loss: 0.0003, grad_norm: 0.00917934812605381, learning_rate: 4.0322580645161294e-08, epoch: 19.98
loss: 0.0938, grad_norm: 0.009722145274281502, learning_rate: 3.2258064516129035e-08, epoch: 19.99
loss: 0.0003, grad_norm: 0.007487975526601076, learning_rate: 2.4193548387096776e-08, epoch: 19.99
loss: 0.0003, grad_norm: 0.011291628703474998, learning_rate: 1.6129032258064518e-08, epoch: 19.99
loss: 0.0003, grad_norm: 0.009012524038553238, learning_rate: 8.064516129032259e-09, epoch: 20.0
loss: 0.0003, grad_norm: 0.009824828244745731, learning_rate: 0.0, epoch: 20.0
eval_loss: 0.8251942992210388, eval_runtime: 434.3948, eval_samples_per_second: 57.551, eval_steps_per_second: 3.598, epoch: 20.0
train_runtime: 35622.1456, train_samples_per_second: 14.026, train_steps_per_second: 1.755, train_loss: 0.07741053899220151, epoch: 20.0
0%
```

L (D:) > myprojects > nlp > bert > bert-finetuned-imdb

이름	수정된 날짜	유형	크기
☐ config.json	2024-04-15 오후 9:01	JSON 파일	1KB
☐ model.safetensors	2024-04-15 오후 9:02	SAFETENSORS 파일	427,694KB
☐ special_tokens_map.json	2024-04-15 오후 9:02	JSON 파일	1KB
☐ tokenizer_config.json	2024-04-15 오후 9:02	JSON 파일	2KB
☒ vocab.txt	2024-04-15 오후 9:02	텍스트 문서	256KB

```
config.json - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
{
  "_name_or_path": "bert-base-uncased",
  "architectures": [
    "BertForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "problem_type": "single_label_classification",
  "torch_dtype": "float32",
  "transformers_version": "4.39.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 30522
}
```

```
vocab.txt - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)
artificial
gross
corresponding
convicted
cage
caroline
dialogue
##dor
narrative
stranger
mario
br
christianity
failing
trent
commanding
buddhist
1848
maurice
focusing
yale
bike
altitude
##ering
mouse
revised
##sley
veteran
```

Ln 1, Col 1

Ln 30522, Col 1

100%

Wind

```
tokenizer_config.json - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

{
  "added_tokens_decoder": {
    "0": {
      "content": "[PAD]",
      "lstrip": false,
      "normalized": false,
      "rstrip": false,
      "single_word": false,
      "special": true
    },
    "100": {
      "content": "[UNK]",
      "lstrip": false,
      "normalized": false,
      "rstrip": false,
      "single_word": false,
      "special": true
    },
    "101": {
      "content": "[CLS]",
      "lstrip": false,
      "normalized": false,
      "rstrip": false,
      "single_word": false,
      "special": true
    },
    "102": {
      "content": "[SEP]",
      "lstrip": false,
      "normalized": false,
      "rstrip": false,
      "single_word": false,
      "special": true
    },
    "103": {
      "content": "[MASK]",
      "lstrip": false,
      "normalized": false,
      "rstrip": false,
      "single_word": false,
      "special": true
    }
  },
  "clean_up_tokenization_spaces": true,
  "cls_token": "[CLS]",
  "do_basic_tokenize": true,
  "do_lower_case": true,
  "mask_token": "[MASK]",
  "model_max_length": 512,
  "never_split": null,
  "pad_token": "[PAD]",
  "sep_token": "[SEP]",
  "strip_accents": null,
  "tokenize_chinese_chars": true,
  "tokenizer_class": "BertTokenizer",
  "unk_token": "[UNK]"
}

special_tokens_map.json - Windows 메모장
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

{
  "cls_token": "[CLS]",
  "mask_token": "[MASK]",
  "pad_token": "[PAD]",
  "sep_token": "[SEP]",
  "unk_token": "[UNK]"
}

Ln 1, Col 1 100%
```

```
Ln 1, Col 1 100% Windows (CRLF) UTF-8
```

2.3 실행결과 (Predict)

```
관리자: 명령 프롬프트 - "D:\program\Anaconda3\condatier\conda.bat" activate word2vec - python bert_imdb.py predict

(word2vec) D:\myprojects\nlp\bert>python bert_imdb.py predict
Using device: cuda

=====
Usage [Training]: python bert_imdb.py train totalEpochNo[예:10]
Usage [Prediction]: python bert_imdb.py predict

*** Model loading.....
*** Model loaded successfully from bert-finetuned-imdb

BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
        (intermediate_act_fn): GELUActivation()
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
  )
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)

*** Type a sentence or 'quit' to stop: The movie was fantastic
==> query sentence: The movie was fantastic
==> Predicted output: [{'label': 'LABEL_1', 'score': 0.9998607635498047}]

*** Type a sentence or 'quit' to stop: The movie was very bad
==> query sentence: The movie was very bad
==> Predicted output: [{'label': 'LABEL_0', 'score': 0.999586284160614}]

*** Type a sentence or 'quit' to stop:
```

※ [Questions]

- # 1. huggingface의 transformers package가 제공하는 BERT의 다른 모델에는 어떤 모델들이 있는가?
- # 2. BERT에서 사용하는 Tokenizer에는 어떤 것들이 있나?
- # 3. 한국어나 중국어와 같은 언어 문장을 BERT에서 임베딩 할 경우, 영어와 어떤 차이점이 있나?
- # 4. BERT의 대표적인 다국어 Tokenizer를 이용하여 한국어 문장을 임베딩 하여 보라

※ [실습 문제]

- # 5. 한국어 문장 corpus (주제는 상관없음)를 구하여, BERT 모델을 적용하여 Sentence Classification 을 수행하시오.