BERT 네이버 영화 리뷰 분류 (1)

BERT 는 ELMo나 GPT - 1과 마찬가지로 문맥을 반영한 임베딩을 사용하고 있습니다.

transformers, tensorflow 라이브러리를 사용함. 라이브러리 설치할 때 버전을 잘 맞춰주어야 합니다.

```
!pip install transformers==2.11.0
!pip install tensorflow==2.2.0
```

필요한 라이브러리 불러오기

```
import os
import re
import json
import copy
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('seaborn-white')

from tqdm import tqdm
import tensorflow as tf
from transformers import *
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

batch_size, num_epochs, valid_split, max_len을 먼저 지정해 줍니다.

```
tf.random.set_seed(111)
np.random.seed(111)

BATCH_SIZE = 32
NUM_EPOCHS = 3
VALID_SPLIT = 0.2
MAX_LEN = 39
```

데이터 다운로드

```
import urllib.request

train_file = urllib.request.urlopen("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_train.txt")
test_file = urllib.request.urlopen("https://raw.githubusercontent.com/e9t/nsmc/master/ratings_test.txt")

train_data = pd.read_table(train_file)
test_data = pd.read_table(test_file)

train_data = train_data.dropna() #불필요한 값 제거
test_data = test_data.dropna()
```

Train 데이터와 test 데이터를 보면 id, documend, label로 이루어져 있음을 알 수 있습니다.

```
train_data.head()
```

	id	document	label
0	9976970	아 더빙.. 진짜 짜증나네요 목소리	0
1	3819312	흠...포스터보고 초딩영화줄....오버연기조차 가볍지 않구나	1
2	10265843	너무재밋었다그래서보는것을추천한다	0
3	9045019	교도소 이야기구먼 ..솔직히 재미는 없다..평점 조정	0
4	6483659	사이몬페그의 익살스런 연기가 돋보였던 영화!스파이더맨에서 늙어보이기만 했던 커스틴 ...	1

```
test_data.head()
```

	id	document	label
0	6270596	굳 ㅋ	1
1	9274899	GDNTOPCLASSINTHECLUB	0
2	8544678	뭐야 이 평점들은.... 나쁘진 않지만 10점 짜리는 더더욱 아니잖아	0
3	6825595	지루하지는 않은데 완전 막장임... 돈주고 보기에는....	0
4	6723715	3D만 아니었어도 별 다섯 개 줬을텐데.. 왜 3D로 나와서 제 심기를 불편하게 하죠??	0

BertTokenizer

```
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased', cache_dir = 'bert_ckpt', do_lower_case=False)
```

‘bert-base-multilingual-cased’인 BERT 다국어 기본 모델을 사용해 tokenizer를 진행함. 버트 모델은 학습한 데이터에 따라 여러 형태로 공개가 되어 있지만, 대부분은 영어를 위한 모델. 한글 데이터를 활용한 버트를 활용할 것이기 때문에 다국어 지원을 위한 버트 모델을 활용함. 모델 관련해 더 자세하게 궁금하신 분들은 https://huggingface.co/bert-base-multilingual-cased 참조해주세요.

```
def bert_tokenizer(sentence, MAX_LEN):

  encoded_dict = tokenizer.encode_plus(
      text = sentence,
      add_special_tokens = True,
      max_length = MAX_LEN,
      pad_to_max_length = True,
      return_attention_mask = True
  )

  input_id = encoded_dict['input_ids']
  attention_mask = encoded_dict['attention_mask']
  token_type_id = encoded_dict['token_type_ids']

  return input_id, attention_mask, token_type_id
```

encode_plus 기능은 bert에 필요한 입력 형태로 변환할 뿐만 아니라 문장을 최대 길이에 맞게 패딩하고 결과값을 딕셔너리로 출력해줍니다.

encode_plus 의 순서

1. 문장 토크나이징
2. add_special_tokens를 True로 지정하면 토큰의 시작점에 ‘[CLS]’ 토큰, 토

큰의 마지막에 ‘[SEP]’토큰을 붙여줍니다.

1. 각 토큰을 인덱스로 변환해줍니다.
2. max_length에 MAX_LEN(최대 길이)에 따라 문장의 길이를 맞추는 직업을 진행함.
3. pad_to_max_length 기능을 통해 MAX_LEN의 길이에 미치지 못하는 문장에 패딩을 적용함.
4. return_attention_mask 기능을 통해 어텐션 마스크를 생성합니다.

토큰 타입은 문장이 1개일 경우 0으로, 문장이 2개일 경우 0과 1로 구분해서 생성합니다.

6-1. 문장이 바뀔 때 마다 0에서 1로 바뀐 후 다시 1에서 0으로 바뀐다

6-2. ex [0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0] <= 3문장이라는 것을 알 수 있다.

문장을 bert에 활용하기 위해서는 활용하는 특정 분야에 맞게 다양한 입력값으로 치환해야 함. 여기서 bert는 일반적으로 3가지 입력값을 넣어줍니다.

1. input_ids: 문장을 토크나이즈해서 인덱스 값으로 변환함. 일반적으로 버트에서는 단어를 subword 단위로 변환시키는 워드 피스 토크나이저를 사용함.

BERT가 사용한 토크나이저는 WordPiece 토크나이저

서브워드 토크나이저는 ‘기본적으로 자주 등장하는 단어는 그대로 단어 집합에 추가,

자주 등장하지 않는 단어의 경우에는 더 작은 단위의 서브워드로 분리되어 서브워드들이 단어 집합에 추가된다.’라는 의미

1. attention_mask: 패딩된 부분에 대해 학습에 영향을 받지 않기 위해 처리해 주는 입력값, 버트 토크나이저에서는 1은 어텐션에 영향을 받는 토큰, 0은 영향을 받지 않는 토큰임.
2. token_type_id: 두 개의 시퀀스 입력으로 활용할 때 0과 1로 문장의 토큰 값을 분리함.

[CLS]는 문장의 시작, [SEP]는 문장이 분리되는 부분을 의미하는 토큰임.

```
input_ids = []
attention_masks = []
token_type_ids = []
train_data_labels = []

for train_sentence, train_label in tqdm(zip(train_data['document'], train_data['label']), total=len(train_data)):

  try:
    input_id, attention_mask, token_type_id = bert_tokenizer(train_sentence, MAX_LEN)

    input_ids.append(input_id)
    attention_masks.append(attention_mask)
    token_type_ids.append(token_type_id)
    train_data_labels.append(train_label)
  except Exception as e:
    print(e)
    pass

#학습할 데이터
train_movie_input_ids = np.array(input_ids, dtype=int)
train_movie_attention_masks = np.array(attention_masks, dtype=int)
train_movie_token_type_ids = np.array(token_type_ids, dtype=int)
train_movie_inputs = (train_movie_input_ids, train_movie_attention_masks, train_movie_token_type_ids)
train_data_labels = np.asarray(train_data_labels, dtype=np.int32)

print("Sentences: {}₩nLabels: {}".format(len(train_movie_input_ids), len(train_data_labels)))
```

```
100%|██████████| 149995/149995 [01:02<00:00, 2398.44it/s]
Sentences: 149995
Labels: 149995
```

버트 토크나이저에서는 문장의 시작이나 끝, 모델별 목적에 맞게 스페셜 토큰에 활용함.

서브워드 토크나이저는 '기본적으로 자주 등장하는 단어는 그대로 단어 집합에 추가, 자주 등장하지 않는 단어의 경우에는 더 작은 단위의 서브워드로 분리되어 서브워드들이 단어 집합에 추가된다.'라는 의미

```
idx = 5

input_id = train_movie_input_ids[idx]
attention_mask = train_movie_attention_masks[idx]
token_type_id = train_movie_token_type_ids[idx]

print(input_id)
print(attention_mask)
print(token_type_id)
print(tokenizer.decode(input_id)) #실제적으로 들어간 값 출력됨됨
```

```
[   101   9247   8867  32158  23811    100    124  24982  17655   9757
  55511    122  23321  10954  24017  12030    129 106249  24974  30858
  18227    119    100    119    119    119   9353  30134  21789  12092
   9519 118671 119169    119    102      0      0      0      0]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0]
[CLS] 막 걸음마 [UNK] 3세부터 초등학교 1학년생인 8살용영화. [UNK]... 별반개도 아까움. [SEP] [PAD] [PAD] [PAD] [PAD]
```

차례대로 input_id, attention_mask, token_type_id, tokenizer.decode(input_id) 값을 확인할 수 있습니다.

두번째 줄은 attention_mask입니다. 1은 어텐션에 영향을 받는 토큰, 0은 영향을 받지 않는 토큰입니다. 뒤에 0인 4개의 값은 padding되어 추가 된 값입니다.

token_type_id는 0으로만 이루어져 있습니다. 1문장으로만 되어있는 것을 알 수 있습니다.

input_id를 decode한 값을 볼 수 있습니다.

BERT의 스페셜 토큰

스페셜 토큰	역할
[UNK]	모르는 단어에 대한 토큰
[MASK]	마스크 토큰, 사전 학습에서 활용
[PAD]	최대 길이를 맞추는 용도
[SEP]	문장의 끝을 알림
[CLS]	문장의 시작을 알림

```
class TFBertClassifier(tf.keras.Model):
  def __init__(self, model_name, dir_path, num_class):
    super(TFBertClassifier, self).__init__()

    self.bert = TFBertModel.from_pretrained(model_name, cache_dir=dir_path)
    self.dropout = tf.keras.layers.Dropout(self.bert.config.hidden_dropout_prob)
    self.classifier = tf.keras.layers.Dense(num_class,
                                            kernel_initializer = tf.keras.initializers.TruncatedNormal(self.bert.config.initializer_range),
                                            name='classifier')

  def call(self, inputs, attention_masks=None, token_type_ids=None, training=False):
    outputs = self.bert(inputs, attention_mask=attention_mask, token_type_ids=token_type_ids)
    pooled_output = outputs[1]
    pooled_output = self.dropout(pooled_output, training=training)
    logits = self.classifier(pooled_output)

    return logits

cls_model = TFBertClassifier(model_name='bert-base-multilingual-cased',
                             dir_path='bert_ckpt',
                             num_class=2)
```

```
Downloading: 100% 625/625 [00:00<00:00, 13.1kB/s]
Downloading: 100% 1.08G/1.08G [00:36<00:00, 24.4MB/s]
```

bert 기본 모델을 사용함.

모델 학습

```
optimizer = tf.keras.optimizers.Adam(3e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')
cls_model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

```
model_name = "tf2_bert_naver_movie"

es_callback = EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=2)

checkpoint_path = os.path.join('./', model_name, 'weights.h5')
checkpoint_dir = os.path.dirname(checkpoint_path)

if os.path.exists(checkpoint_dir):
  print("{} Directory already exists₩n".format(checkpoint_dir))
else:
  os.makedirs(checkpoint_dir, exist_ok=True)
  print("{} Directory create complete₩n".format(checkpoint_dir))

cp_callback = ModelCheckpoint(checkpoint_path, monitor='val_accuracy',
                              verbose=1, save_best_only=True, save_weights_only=True)

history = cls_model.fit(train_movie_inputs, train_data_labels,
                        epochs=NUM_EPOCHS, batch_size=BATCH_SIZE, validation_split=VALID_SPLIT,
                        callbacks=[es_callback, cp_callback])
```

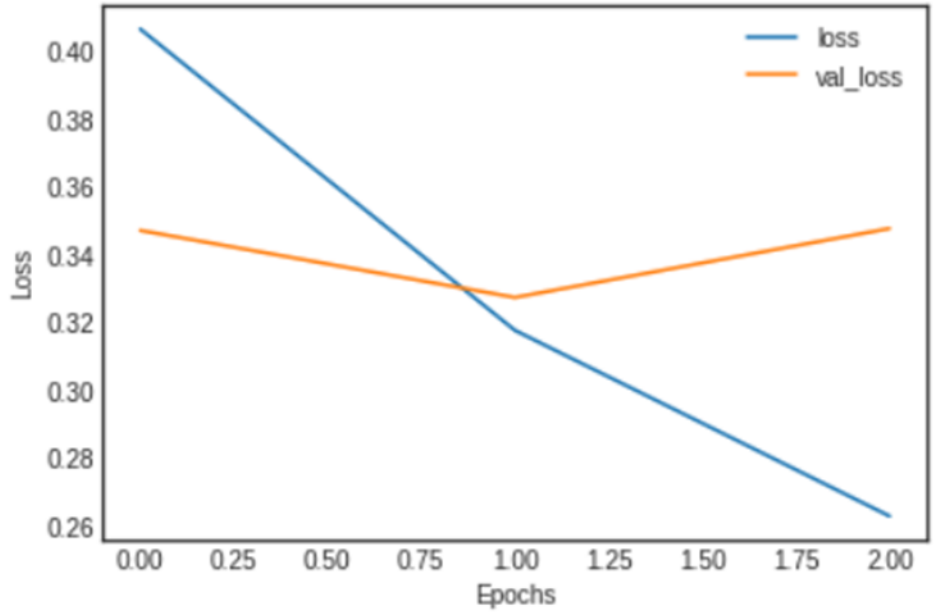
```
./tf2_bert_naver_movie Directory create complete

Epoch 1/3
3750/3750 [==============================] - ETA: 0s - loss: 0.4065 - accuracy: 0.8119
Epoch 1: val_accuracy improved from -inf to 0.84599, saving model to ./tf2_bert_naver_movie/weights.h5
3750/3750 [==============================] - 1300s 331ms/step - loss: 0.4065 - accuracy: 0.8119 - val_loss: 0.3472 - val_accuracy: 0.8460
Epoch 2/3
3750/3750 [==============================] - ETA: 0s - loss: 0.3178 - accuracy: 0.8618
Epoch 2: val_accuracy improved from 0.84599 to 0.85796, saving model to ./tf2_bert_naver_movie/weights.h5
3750/3750 [==============================] - 1237s 330ms/step - loss: 0.3178 - accuracy: 0.8618 - val_loss: 0.3275 - val_accuracy: 0.8580
Epoch 3/3
3750/3750 [==============================] - ETA: 0s - loss: 0.2631 - accuracy: 0.8902
Epoch 3: val_accuracy did not improve from 0.85796
3750/3750 [==============================] - 1174s 313ms/step - loss: 0.2631 - accuracy: 0.8902 - val_loss: 0.3478 - val_accuracy: 0.8567
```

sparse_categorical_crossentropy의 경우, 정수 형태로 반환함.

categorical_crossentropy의 경우, 원-핫 벡터 형태로 반환

정확도는 약 85퍼센트임.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'], '')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['loss', 'val_loss'])
plt.show()
```



모델 결과를 시각화해서 나타냄.

모델 평가

```
#test 데이터를 모델에 넣어줄 준비
input_ids = []
attention_masks = []
token_type_ids = []
test_data_labels = []

for test_sentence, test_label in tqdm(zip(test_data['document'], test_data['label'])):

  try:
    input_id, attention_mask, token_type_id = bert_tokenizer(test_sentence, MAX_LEN)

    input_ids.append(input_id)
    attention_masks.append(attention_mask)
    token_type_ids.append(token_type_id)
    test_data_labels.append(test_label)
  except Exception as e:
    print(e)
    pass

#학습할 데이터
test_movie_input_ids = np.array(input_ids, dtype=int)
test_movie_attention_masks = np.array(attention_masks, dtype=int)
test_movie_token_type_ids = np.array(token_type_ids, dtype=int)
test_movie_inputs = (test_movie_input_ids, test_movie_attention_masks, test_movie_token_type_ids)
test_data_labels = np.asarray(test_data_labels, dtype=np.int32)
```

```
49997it [00:23, 2095.14it/s]
```

```
cls_model.evaluate(test_movie_inputs, test_data_labels, batch_size=1024)
```

```
49/49 [==============================] - 122s 2s/step - loss: 0.3524 - accuracy: 0.8538
[0.3523990213871002, 0.8537912368774414]
```

test 데이터 모델링 결과 정확도는 0.85정도 나왔음.