Transformer & BERT 실습 (BERT)

최 석 재

lingua@naver.com

BERT 실습

구글 드라이브와 연결

- 런타임 > 런타임 유형 변경 > 하드웨어 가속기 > GPU 로 설정

- from google.colab import drive
- drive.mount('/content/gdrive')

Hugging Face 트랜스포머 설치

- !pip install transformers
- !pip install sacremoses

라이브러리 import

- import os
- import re
- import numpy as np
- 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

- import pandas as pd
- import matplotlib.pyplot as plt

사용자 정의 시각화 함수

- def plot_graphs(history, string):
    plt.plot(history.history[string])
    plt.plot(history.history['val_'+string], '')
    plt.xlabel("Epochs")
    plt.ylabel(string)
    plt.legend([string, 'val_'+string])
    plt.show()

랜덤 시드 고정

- tf.random.set_seed(1111)
- np.random.seed(1111)

하이퍼파라미터 지정

- CLASS_NUMBER = 2
- BATCH_SIZE = 32
- NUM_EPOCHS = 2
- VALID_SPLIT = 0.2
- MAX_LEN = 40
- BERT_CKPT = '/content/gdrive/My Drive/pytest/data/KOR/naver_movie/bert_ckpt/'
- DATA_IN_PATH = '/content/gdrive/My Drive/pytest/data/KOR/naver_movie/data_in/'
- DATA_OUT_PATH = "/content/gdrive/My Drive/pytest/data/KOR/naver_movie/data_out/"

※ BERT_CKPT 와 DATA_OUT_PATH 의 내용은 작업이 끝나면 삭제하여 공간을 확보한다

읽기와 쓰기 함수 정의 및 데이터 로딩

- def listToString(listdata):
  result = 'id₩tdocument₩tlabel₩n'
  for data_each in listdata:
  if data_each:
  result += data_each[0]+"₩t"+data_each[1]+"₩t"+data_each[2]+"₩n"
  return result;

- def read_data(filename, encoding='cp949', start=0): # 읽기 함수 정의
  with open(filename, 'r', encoding=encoding) as f:
  data = [line.split('₩t') for line in f.read().splitlines()]
  data = data[start:]
  return data

- def write_data(data, filename, encoding='cp949'): # 쓰기 함수 정의
  with open(filename, 'w', encoding=encoding) as f:
  f.write(data)

- data_ratings = read_data(os.path.join(DATA_IN_PATH, "ratings_utf8_small.txt") , encoding='utf-8', start=1) # 자료 읽기

훈련데이터와 테스트데이터 분리

- from sklearn.model_selection import train_test_split
- ratings_train, ratings_test = train_test_split(data_ratings)

- ratings_train = listToString(ratings_train)
- ratings_test = listToString(ratings_test)

- write_data(ratings_train, os.path.join(DATA_IN_PATH, "ratings_train.txt"), encoding='utf-8')
- write_data(ratings_test, os.path.join(DATA_IN_PATH, "ratings_test.txt"), encoding='utf-8')

BERT 토크나이저 다운로드

- tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased", cache_dir=os.path.join(BERT_CKPT, "tokenizer"), do_lower_case=False)

BERT 토크나이저 연습 1

- test_sentence = "안녕하세요, 반갑습니다."

- encode = tokenizer.encode(test_sentence)
- token_print = [tokenizer.decode(token) for token in encode]

- print(encode)
- print(token_print)

```
[101, 9521, 118741, 35506, 24982, 48549, 117, 9321, 118610, 119081, 48345, 119, 102]
['[ C L S ]', '안', '# # 녕', '# # 하', '# # 세', '# # 요', ',', '반', '# # 갑', '# # 습', '# # ㄴ 다', '.', '[ S E P ]']
```

BERT 토크나이저 연습 2

- kor_encode = tokenizer.encode("안녕하세요, 반갑습니다")
- eng_encode = tokenizer.encode("Hello world")
- kor_decode = tokenizer.decode(kor_encode)
- eng_decode = tokenizer.decode(eng_encode)

- print(kor_encode)
- print(eng_encode)
- print(kor_decode)
- print(eng_decode)

```
[101, 9521, 118741, 35506, 24982, 48549, 117, 9321, 118610, 119081, 48345, 102]
[101, 31178, 11356, 102]
[CLS] 안녕하세요, 반갑습니다 [SEP]
[CLS] Hello world [SEP]
```

데이터 확인

- DATA_TRAIN_PATH = os.path.join(DATA_IN_PATH, "ratings_train.txt")
- DATA_TEST_PATH = os.path.join(DATA_IN_PATH, "ratings_test.txt")

- train_data = pd.read_csv(DATA_TRAIN_PATH, header = 0, delimiter = '₩t', quoting = 3)
- train_data = train_data.dropna()
- train_data.head()

	id	document	label
0	7743368	박흥식영화 사랑해 말순씨 ***보면 장애우를 배려하지않는 저질 영화장애우가 성희롱이...	0
1	10067386	안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화.	1
2	4254270	성형부작용 같은 주인공 얼굴 때문에 집중이 안됨..;;;	0
3	530171	맥티어난의 최고 졸작품	0
4	9462634	어린 외계인역 하신분 귀엽네요..ㅎ	1

스페셜 토큰 확인

- print(tokenizer.all_special_tokens, "₩n", tokenizer.all_special_ids)

- kor_encode = tokenizer.encode("안녕하세요, 반갑습니다. ")
- eng_encode = tokenizer.encode("Hello world")

- kor_decode = tokenizer.decode(kor_encode)
- eng_decode = tokenizer.decode(eng_encode)

- print(kor_encode)
- print(eng_encode)
- print(kor_decode)
- print(eng_decode)

```
['[UNK]', '[SEP]', '[PAD]', '[CLS]', '[MASK]']
 [100, 102, 0, 101, 103]
[101, 9521, 118741, 35506, 24982, 48549, 117, 9321, 118610, 119081, 48345, 119, 102]
[101, 31178, 11356, 102]
[CLS] 안녕하세요, 반갑습니다. [SEP]
[CLS] Hello world [SEP]
```

사용자 정의 BERT 토크나이저 함수

```
def bert_tokenizer(sent, MAX_LEN):
    encoded_dict = tokenizer.encode_plus(
        text = sent,
        add_special_tokens = True,      # [CLS]와 [SEP] 추가
        max_length = MAX_LEN,
        pad_to_max_length = True,
        return_attention_mask = True
    )
```

- input_id = encoded_dict['input_ids'] # 각 토큰을 인덱스로 변환
- attention_mask = encoded_dict['attention_mask'] # 어텐션 마스크 생성
- token_type_id = encoded_dict['token_type_ids'] # 문장이 1개일 경우 0, 2개일 경우 0과 1로 구분하여 생성

- return input_id, attention_mask, token_type_id

훈련데이터 변환

- input_ids = [] ; attention_masks = [] ; token_type_ids = [] ; train_data_labels = []
- for train_sent, 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_sent, 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)
```

- train_movie_input_ids = np.array(input_ids, dtype=int)
- train_movie_attention_masks = np.array(attention_masks, dtype=int)
- train_movie_type_ids = np.array(token_type_ids, dtype=int)
- train_movie_inputs = (train_movie_input_ids, train_movie_attention_masks, train_movie_type_ids)

- train_data_labels = np.asarray(train_data_labels, dtype=np.int32)

변환된 문장 확인

- input_id = train_movie_input_ids[1]
- attention_mask = train_movie_attention_masks[1]
- token_type_id = train_movie_type_ids[1]

- print(input_id)
- print(attention_mask)
- print(token_type_id)
- print(tokenizer.decode(input_id))

```
[   101   9521  21789   9651 119168  11102   9326  35506 118762  10530
   9138  13767   9757  48210  89851  18589  42428    119    102      0
      0      0      0      0      0      0      0      0      0      0
      0      0      0      0      0      0      0      0      0      0]
[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 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0]
[CLS] 안개 자욱한 밤하늘에 떠 있는 초승달 같은 영화. [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

BERT 사전학습모델을 이용한 분류기 정의 – init()

```
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, name="classifier")
```

BERT 사전학습모델을 이용한 분류기 정의 – call()

- def call(self, inputs, attention_mask=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
```

BERT 모델 다운로드 및 분류기 실행

- cls_model = TFBertClassifier(model_name='bert-base-multilingual-cased’, dir_path=os.path.join(BERT_CKPT, "model"), num_class=CLASS_NUMBER)

모델 컴파일

- 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])

최적 모델 경로 생성

- model_name = "tf2_bert_naver_movie"
- earlystop_callback = EarlyStopping(monitor='val_accuracy', min_delta=0.0001,patience=2)

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

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

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

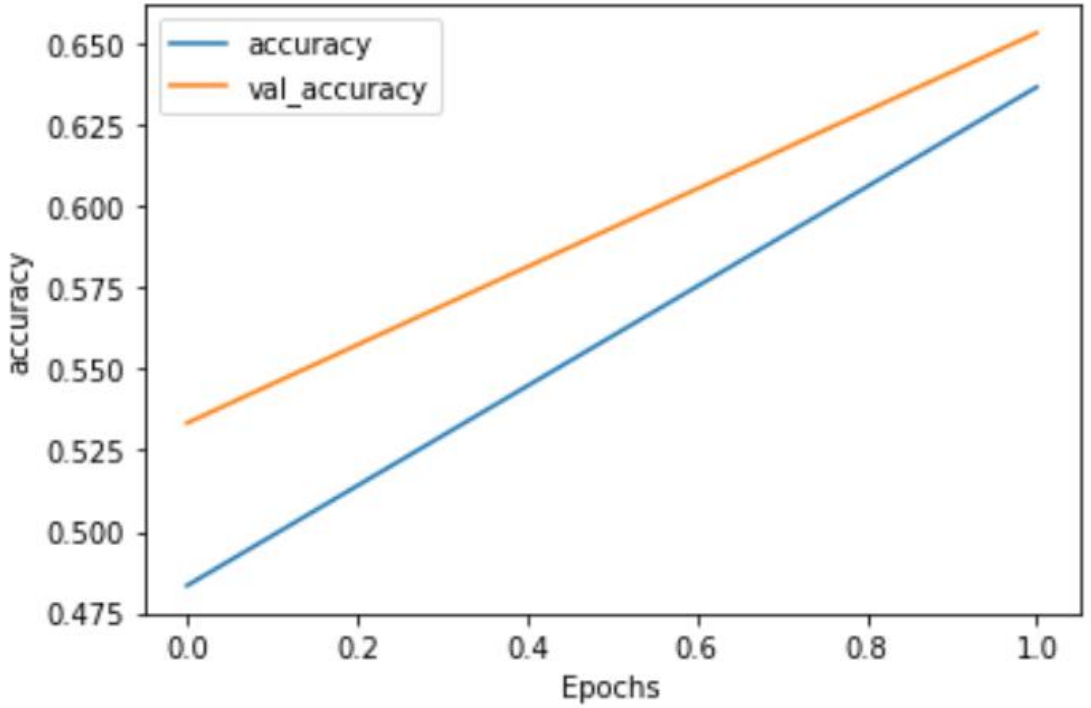
최적 모델 학습

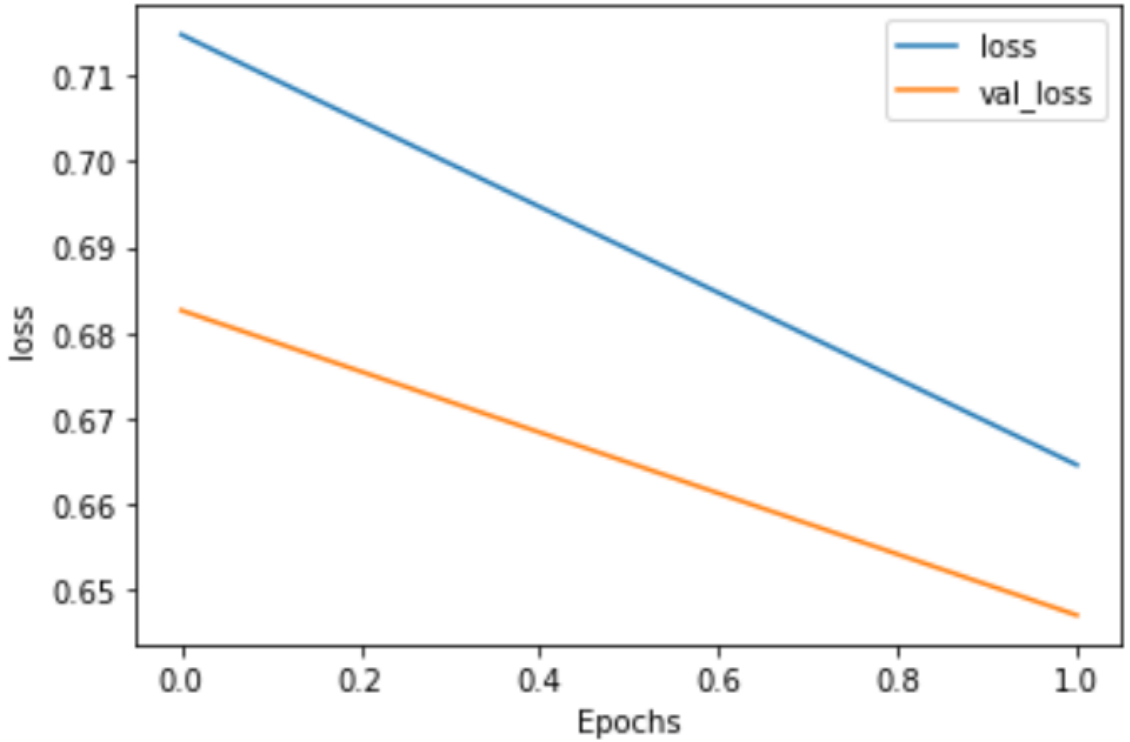
- history = cls_model.fit(train_movie_inputs, train_data_labels, epochs=NUM_EPOCHS, batch_size=BATCH_SIZE, validation_split = VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
- print(history.history)

```
Epoch 1/2
10/10 [==============================] - ETA: 0s - loss: 0.7148 - accuracy: 0.4833
Epoch 1: val_accuracy improved from -inf to 0.53333, saving model to /content/gdrive/My Drive/pytest/data/KOR/naver_movie/data_
10/10 [==============================] - 81s 1s/step - loss: 0.7148 - accuracy: 0.4833 - val_loss: 0.6826 - val_accuracy: 0.533
Epoch 2/2
10/10 [==============================] - ETA: 0s - loss: 0.6645 - accuracy: 0.6367
Epoch 2: val_accuracy improved from 0.53333 to 0.65333, saving model to /content/gdrive/My Drive/pytest/data/KOR/naver_movie/da
10/10 [==============================] - 7s 687ms/step - loss: 0.6645 - accuracy: 0.6367 - val_loss: 0.6470 - val_accuracy: 0.6
{'loss': [0.7147801518440247, 0.6645481586456299], 'accuracy': [0.4833333194255829, 0.6366666555404663], 'val_loss': [0.682584
```

Plotting Accuracy & Loss

- plot_graphs(history, 'accuracy')
- plot_graphs(history, 'loss')





테스트 데이터 확인

- test_data = pd.read_csv(DATA_TEST_PATH, header = 0, delimiter = '₩t', quoting = 3)
- test_data = test_data.dropna()
- test_data.head()

	id	document	label
0	9904263	그 반지의 제왕을 만든 피터 잭슨 ㅋㅋ 이런 영화을 찍었다는거 부터가 웃겼다. ㅋㅋ...	1
1	307000	삐질삐질;;	0
2	8620641	내가 없어질 내인생. 제목을 참 잘 지었다.	1
3	7553915	굿	0
4	9339372	중3인데 일단 2까지봣다. 뭐 말할것도없다장국영 자살이 안타깝다	1

테스트 데이터 변환

- input_ids = [] ; attention_masks = [] ; token_type_ids = [] ; test_data_labels = []

- for test_sent, test_label in tqdm(zip(test_data["document"], test_data["label"])):
  try:
    input_id, attention_mask, token_type_id = bert_tokenizer(test_sent, 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)
- test_movie_input_ids = np.array(input_ids, dtype=int)
- test_movie_attention_masks = np.array(attention_masks, dtype=int)
- test_movie_type_ids = np.array(token_type_ids, dtype=int)
- test_movie_inputs = (test_movie_input_ids, test_movie_attention_masks, test_movie_type_ids)

- test_data_labels = np.asarray(test_data_labels, dtype=np.int32)

테스트 데이터 평가

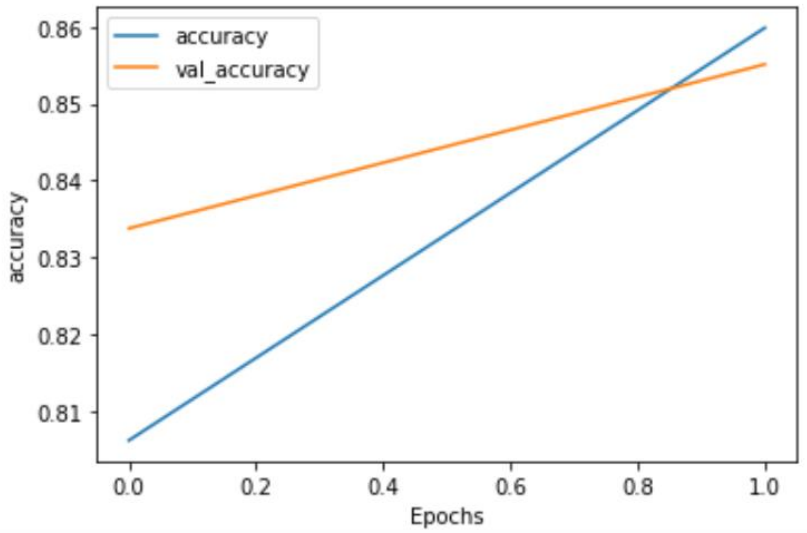
- results = cls_model.evaluate(test_movie_inputs, test_data_labels, batch_size=BATCH_SIZE)
- print("test loss, test acc: ", results)

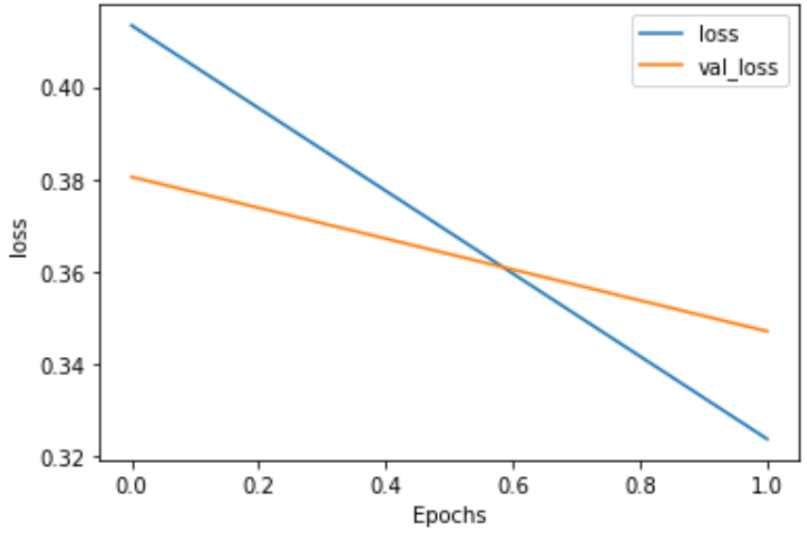
```
4/4 [==============================] - 0s 111ms/step - loss: 0.6762 - accuracy: 0.5760
test loss, test acc:  [0.6761707067489624, 0.5759999752044678]
```

전체 data를 사용한 경우의 성능 (epoch 2)

epoch 1 epoch 2

- {'loss': [0.4132840633392334, 0.3237639665603637],
  'accuracy': [0.8062252402305603, 0.8598774671554565],
  'val_loss': [0.3805010914802551, 0.3470880091190338],
  'val_accuracy': [0.8337944746017456, 0.8551285266876221]}



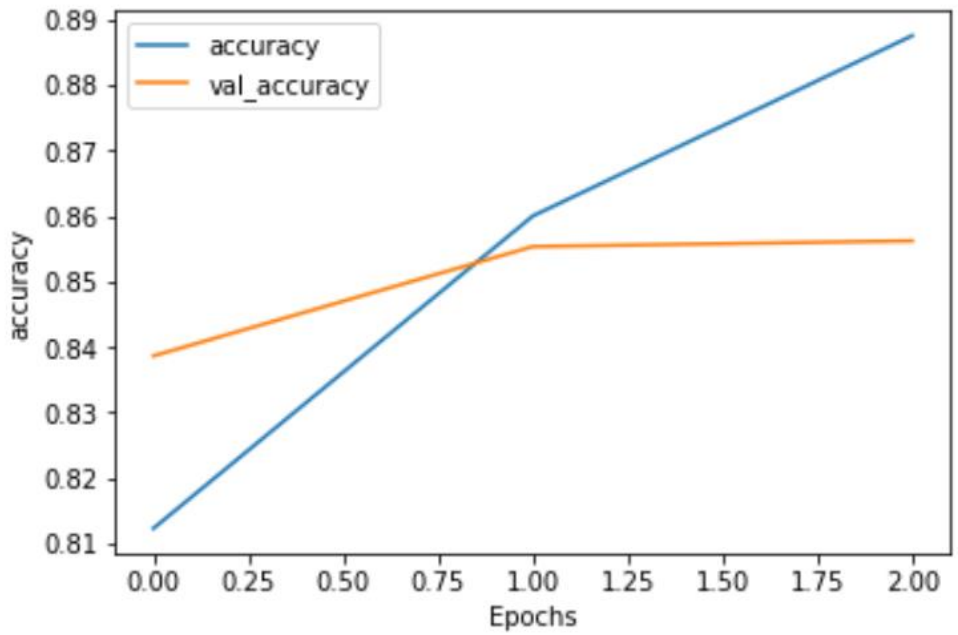


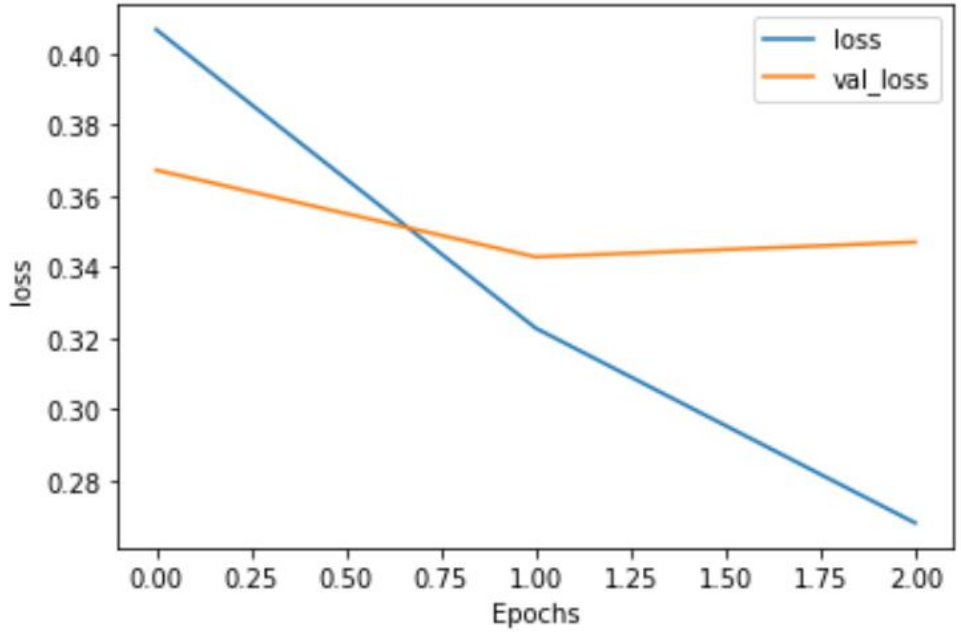
test loss, test acc: [0.3450827002525329 6, 0.8530341386795044]

전체 data를 사용한 경우의 성능 (epoch 3)

epoch 1 epoch 2 epoch 3

- {'loss': [0.4067926406860351 6, 0.3227856457233429, 0.2679786086082458 5],
  'accuracy': [0.8123171925544739, 0.8600274920463562, 0.8875619769096375],
  'val_loss': [0.3672649562358856, 0.3428157567977905, 0.3470034599304199],
  'val_accuracy': [0.8386613130569458, 0.855361819267273, 0.8562285304069519]}





test loss, test acc: [0.3358493447303772, 0.8593543767929077]

검증데이터 성능 비교

	검증데이터 최고수치
ANN	0.832
Pre-embedding	0.762
Train data Embedding	0.823
SimpleRNN	0.821
LSTM	0.839
GRU	0.841
GRU 2층	0.837
BERT	0.859

용량 확보

- pytest > data > KOR > naver_movie 아래에 있는
- bert_ckpt 와 data_out 폴더를 삭제하여 용량을 확보한다
- 이 두 폴더는 다음 수행 시 자동 생성되며,
- 다음 수행 시 관련 파일은 덮어쓰기가 되므로 삭제하지 않아도 무방하다