Sentimental Analysis Using Bert

- 목차 -

1. 개요

사람과 감정적인 대화를 할 수 있는 챗봇을 만드는 과정에서 사람의 감성을 파악하는 감정분석 (Sentimental Analysis)은 필수적인 요소이다. Multi Modal에서 Text 도메인의 감정 분석 성능이 좋지 않아 다양한 NLP분야에서 좋은 성능을 기록하는 Bert를 사용하여 Text 도메인의 감정 분석 프로그램을 제작하게 되었다.

2. 개발 환경

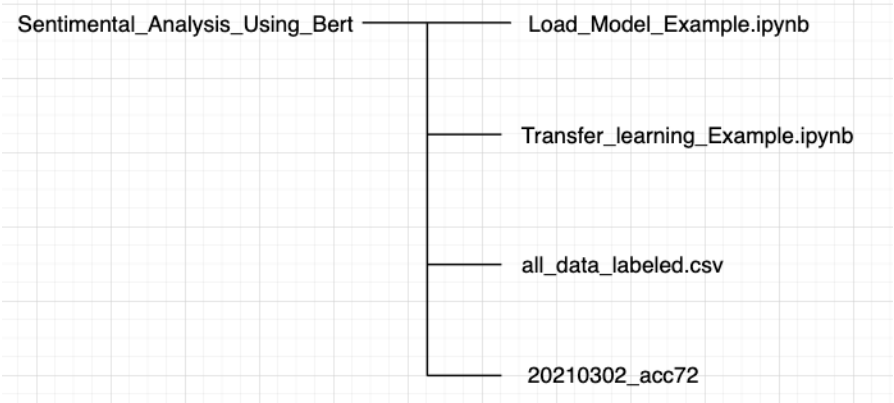
- Ubuntu 18.04 LTS
- Python 3.7.3

필요한 python library는 다음과 같다.

1. Transformers : Bert, Bart, GPT와 같은 다양한 pre-trained 모델을 불러와 사용할 수 있다.
2. Tensorflow == 2.4.1
3. torch == 1.7.1
4. pandas
5. matplotlib
6. scikit-learn
7. jupyter

가상환경이 없다면, 가상환경을 만들고 필요한 라이브러리를 설치해주어야 한다.

3. 파일 구조


Sentimental_Analysis_Using_Bert
Load_Model_Example.ipynb
Transfer_learning_Example.ipynb
all_data_labeled.csv
20210302_acc72

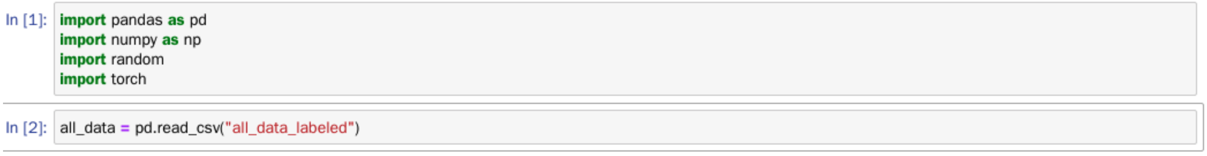
- Load_Model_Example.ipynb : 학습된 모델을 불러오고, 사용하는 예제이다.
- Transfer_learning_Example.ipynb : pre-trained 된 Bert를 불러와 주어진 task에 맞게 transfer learning 시키는 과정이다.
- All_data_labeled.csv : 학습에 필요한 emocap data를 정리해놓은 파일이다.
- 20210302_acc72 : 72% 정확도 성능을 보인 모델이다.

4. Transfer Learning & Save Model

Pytorch를 사용하는 Transfer Learning

1. 데이터 준비

Transfer Learning에는 기존 multi modal에서 사용했던 emocap data를 사용한다. Emocap data중 text 도메인 데이터는 all_data_labeled.csv 파일로 저장되어 있다.

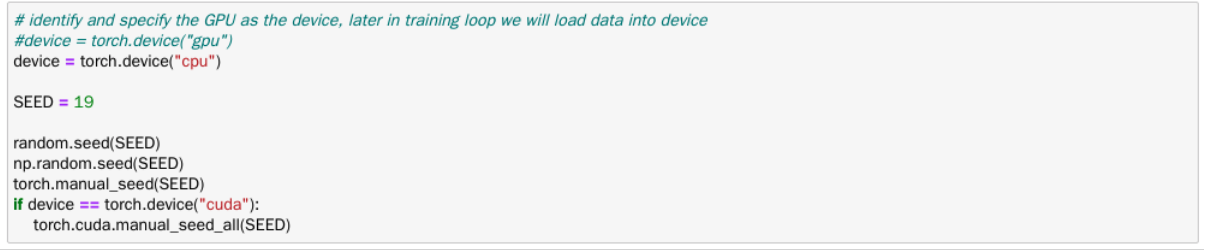


```
In [1]: import pandas as pd
        import numpy as np
        import random
        import torch
```

```
In [2]: all_data = pd.read_csv("all_data_labeled")
```

2. Gpu세팅

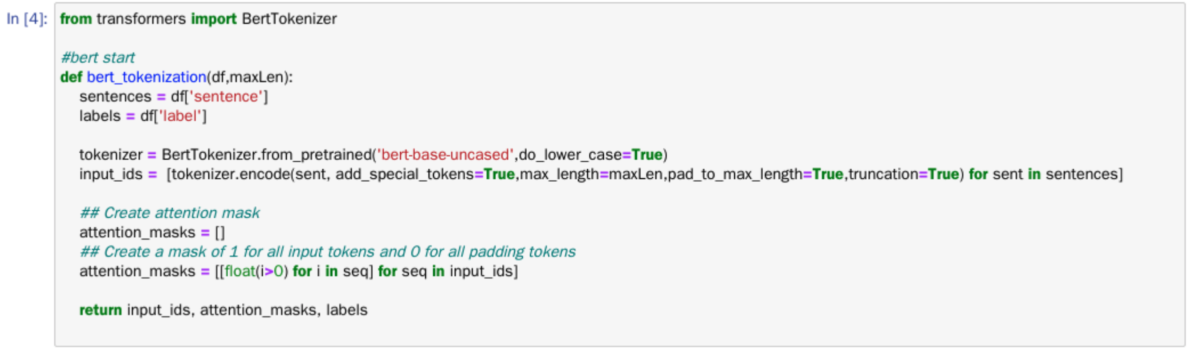
현재 컴퓨터에서는 gpu세팅이 안되어 있어 cpu를 사용하였다.



```
# identify and specify the GPU as the device, later in training loop we will load data into device
#device = torch.device("gpu")
device = torch.device("cpu")

SEED = 19

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if device == torch.device("cuda"):
    torch.cuda.manual_seed_all(SEED)
```

3. Tokenizer 정의

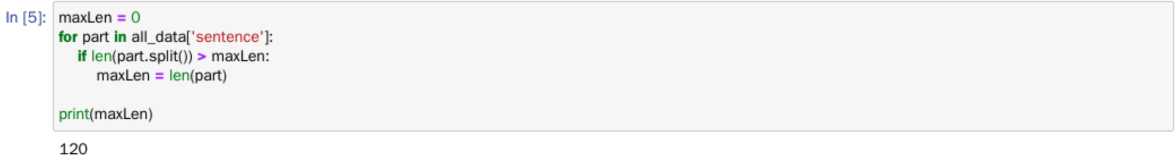
BertTokenizer를 불러와 사용한다.



In [4]:

```
from transformers import BertTokenizer

#bert start
def bert_tokenization(df,maxLen):
    sentences = df['sentence']
    labels = df['label']

    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',do_lower_case=True)
    input_ids =  [tokenizer.encode(sent, add_special_tokens=True,max_length=maxLen,pad_to_max_length=True,truncation=True) for sent in sentences]

    ## Create attention mask
    attention_masks = []
    ## Create a mask of 1 for all input tokens and 0 for all padding tokens
    attention_masks = [[float(i>0) for i in seq] for seq in input_ids]

    return input_ids, attention_masks, labels
```

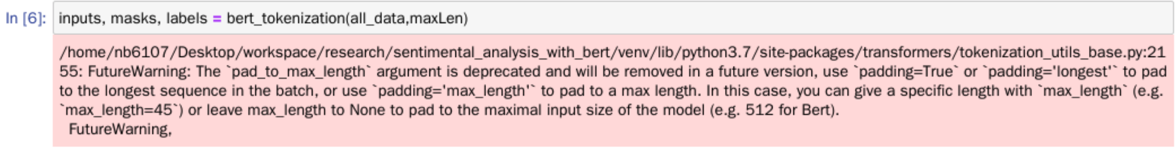
4. 데이터를 전처리한다.



In [5]:

```
maxLen = 0
for part in all_data['sentence']:
    if len(part.split()) > maxLen:
        maxLen = len(part)

print(maxLen)
```

```
120
```



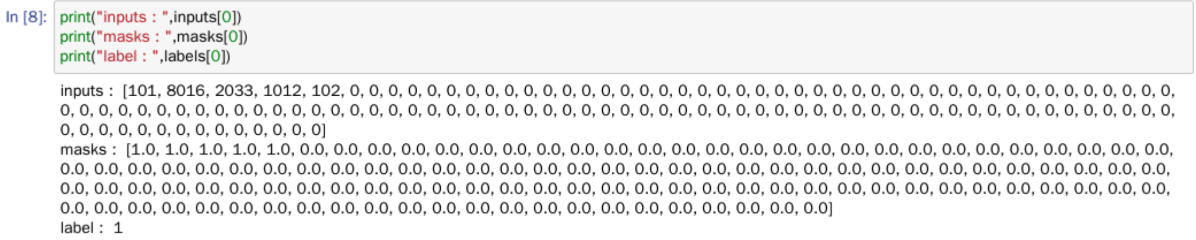
In [6]:

```
inputs, masks, labels = bert_tokenization(all_data,maxLen)
```

```
/home/nb6107/Desktop/workspace/research/sentimental_analysis_with_bert/venv/lib/python3.7/site-packages/transformers/tokenization_utils_base.py:21
55: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version, use `padding=True` or `padding='longest'` to pad
to the longest sequence in the batch, or use `padding='max_length'` to pad to a max length. In this case, you can give a specific length with `max_length` (e.g.
`max_length=45`) or leave max_length to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  FutureWarning,
```

5. 데이터 확인

전처리가 잘 된것을 확인할 수 있다.

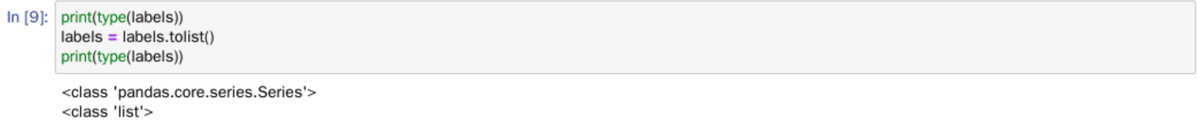


In [8]:

```
print("inputs : ",inputs[0])
print("masks : ",masks[0])
print("label : ",labels[0])
```

```
inputs :  [101, 8016, 2033, 1012, 102, 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, 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]
masks :  [1.0, 1.0, 1.0, 1.0, 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, 0.0, 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, 0.0, 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, 0.0, 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]
label :  1
```

6. Labels의 type을 Series -> list로 바꿔준다.

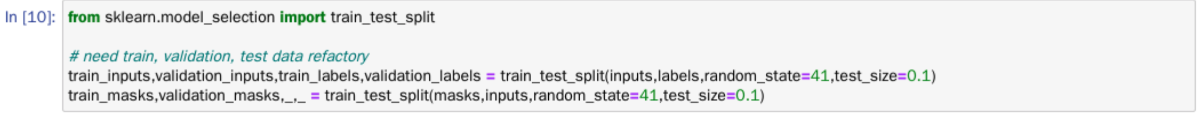


In [9]:

```
print(type(labels))
labels = labels.tolist()
print(type(labels))
```

```
<class 'pandas.core.series.Series'>
<class 'list'>
```

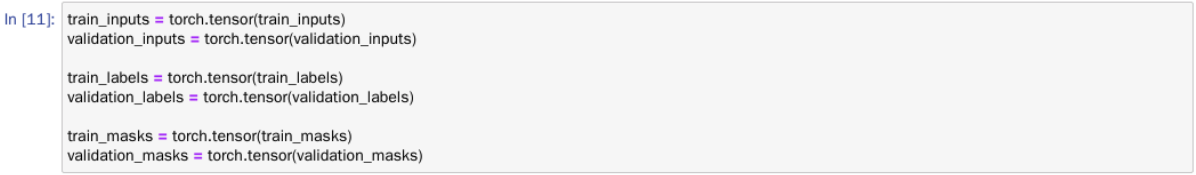
7. Train data, validation data를 분리한다.



In [10]:

```
from sklearn.model_selection import train_test_split

# need train, validation, test data refactory
train_inputs,validation_inputs,train_labels,validation_labels = train_test_split(inputs,labels,random_state=41,test_size=0.1)
train_masks,validation_masks,_,_ = train_test_split(masks,inputs,random_state=41,test_size=0.1)
```

8. Data type을 tensor로 변환해준다.

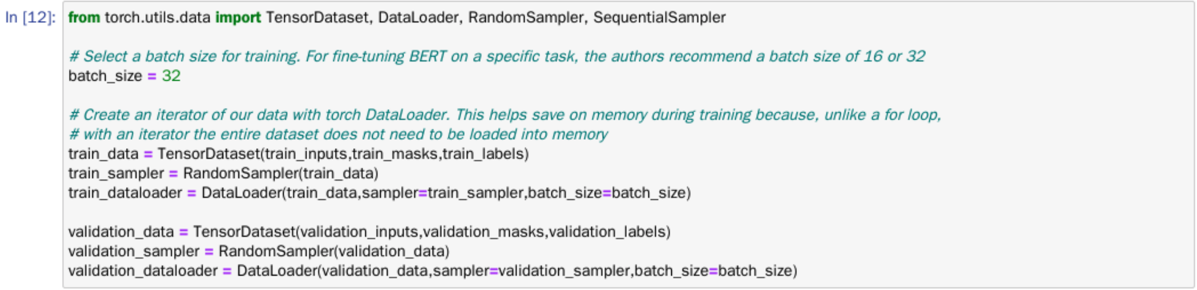


In [11]:

```
train_inputs = torch.tensor(train_inputs)
validation_inputs = torch.tensor(validation_inputs)

train_labels = torch.tensor(train_labels)
validation_labels = torch.tensor(validation_labels)

train_masks = torch.tensor(train_masks)
validation_masks = torch.tensor(validation_masks)
```

9. Torch에서 사용하는 dataloader를 만든다.

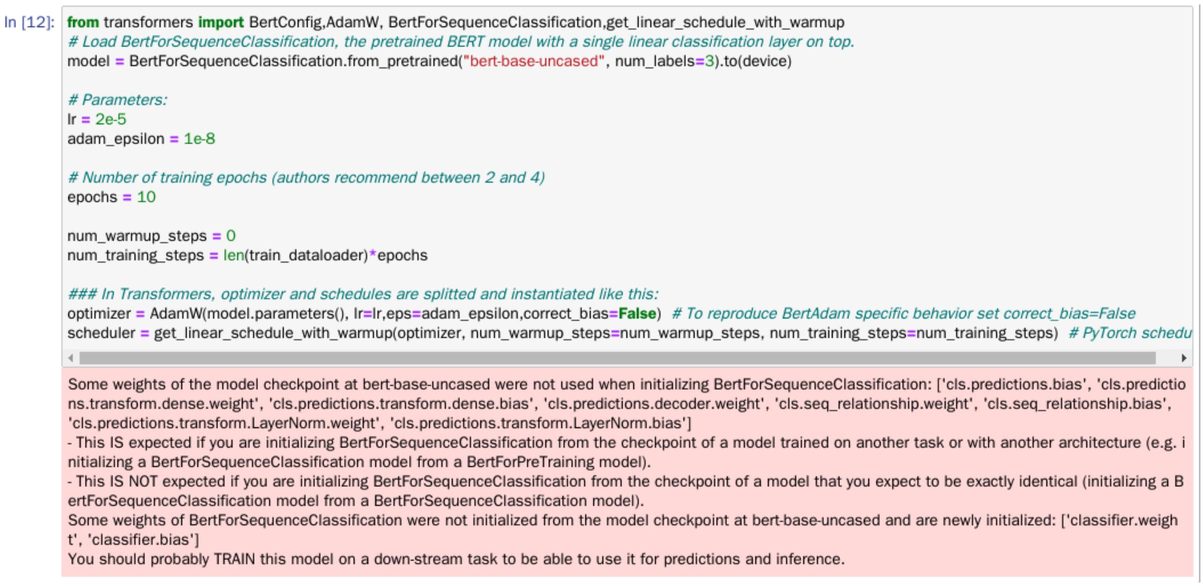
batch_size는 32로 하였다. 자료를 찾은 결과 batch_size는 16 또는 32가 적절하다고 한다.



In [12]:

```
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

# Select a batch size for training. For fine-tuning BERT on a specific task, the authors recommend a batch size of 16 or 32
batch_size = 32

# Create an iterator of our data with torch DataLoader. This helps save on memory during training because, unlike a for loop,
# with an iterator the entire dataset does not need to be loaded into memory
train_data = TensorDataset(train_inputs,train_masks,train_labels)
train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data,sampler=train_sampler,batch_size=batch_size)

validation_data = TensorDataset(validation_inputs,validation_masks,validation_labels)
validation_sampler = RandomSampler(validation_data)
validation_dataloader = DataLoader(validation_data,sampler=validation_sampler,batch_size=batch_size)
```

10. Pre-trained 된 Bert를 불러온 뒤 optimizer, scheduler, learning_rate, epochs등을 정해준다. 현재는 10 epochs로 되어 있지만 적절하게 조절하여 학습할 수 있다.

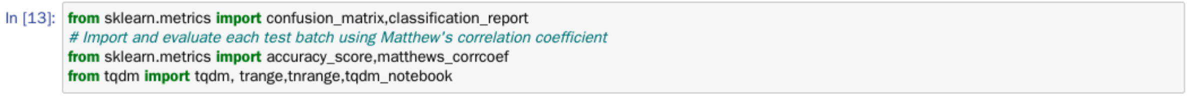


In [12]:

```
from transformers import BertConfig,AdamW, BertForSequenceClassification,get_linear_schedule_with_warmup
# Load BertForSequenceClassification, the pretrained BERT model with a single linear classification layer on top.
model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=3).to(device)

# Parameters:
lr = 2e-5
adam_epsilon = 1e-8

# Number of training epochs (authors recommend between 2 and 4)
epochs = 10

num_warmup_steps = 0
num_training_steps = len(train_dataloader)*epochs

### In Transformers, optimizer and schedules are splitted and instantiated like this:
optimizer = AdamW(model.parameters(), lr=lr,eps=adam_epsilon,correct_bias=False)  # To reproduce BertAdam specific behavior set correct_bias=False
scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=num_warmup_steps, num_training_steps=num_training_steps)  # PyTorch schedu
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictio
ns.transform.dense.weight', 'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with another architecture (e.g. i
nitializing a BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exactly identical (initializing a B
ertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.weigh
t', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

11. 학습에 사용되는 여러 라이브러리 import

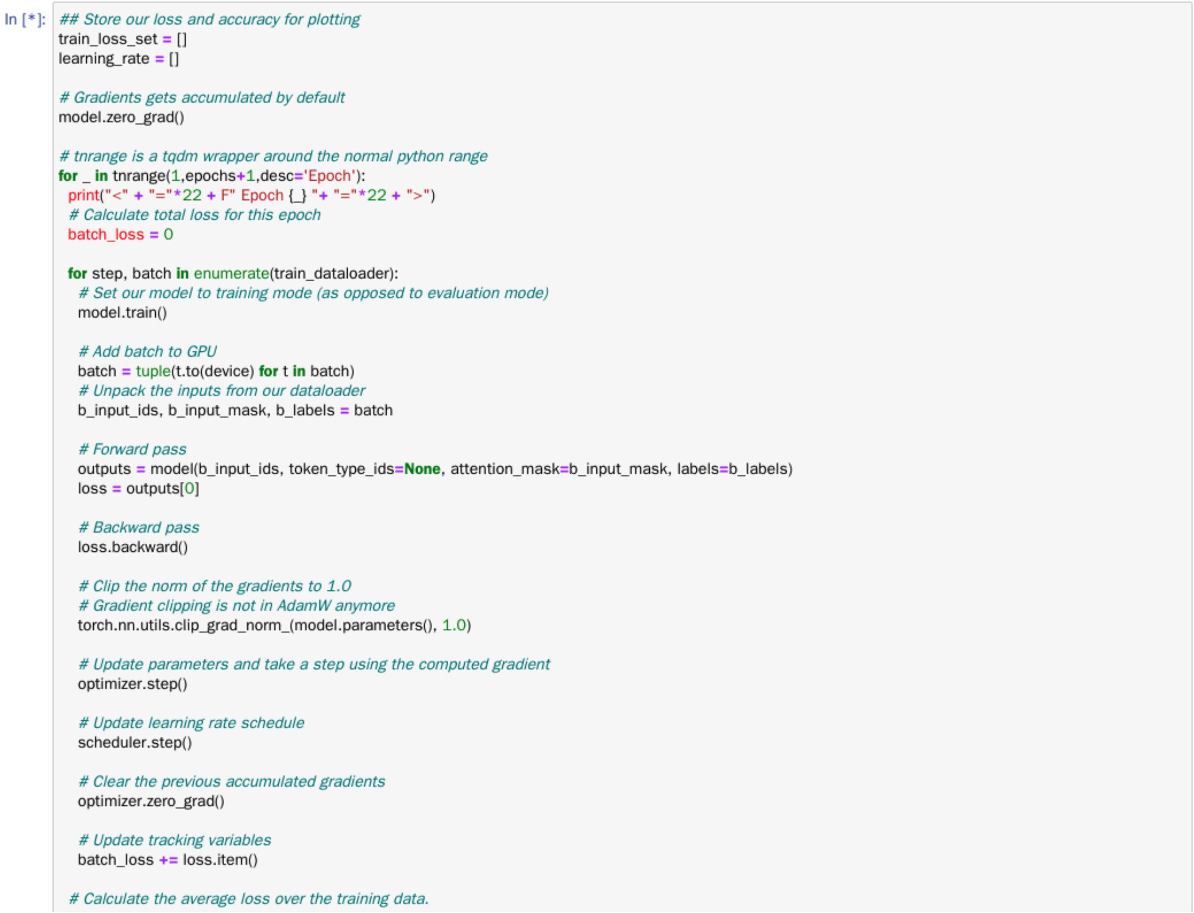


In [13]:

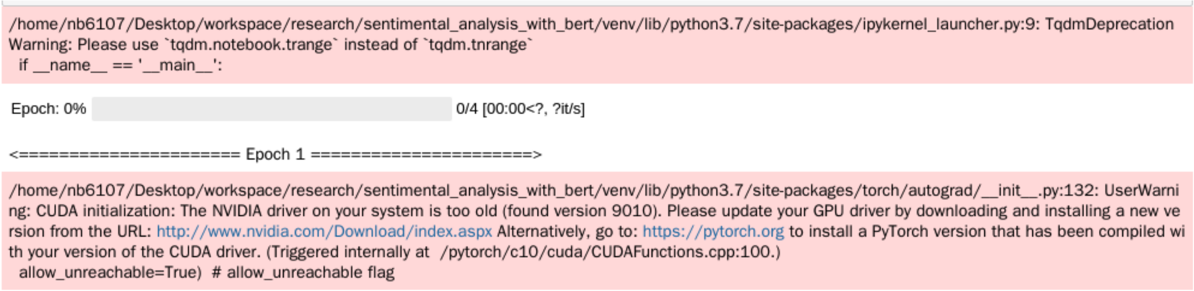
```
from sklearn.metrics import confusion_matrix,classification_report
# Import and evaluate each test batch using Matthew's correlation coefficient
from sklearn.metrics import accuracy_score,matthews_corrcoef
from tqdm import tqdm, trange,tnrange,tqdm_notebook
```

12. 학습.

코드가 너무 길어 일부만 첨부한다.

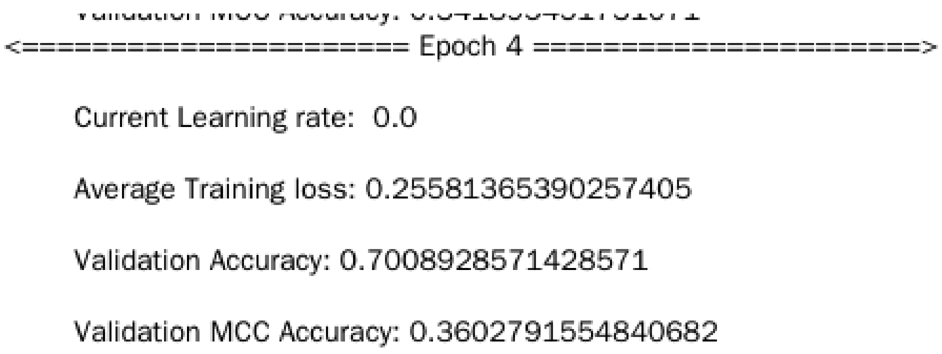


```
In [*]:
## Store our loss and accuracy for plotting
train_loss_set = []
learning_rate = []

# Gradients gets accumulated by default
model.zero_grad()

# tnrange is a tqdm wrapper around the normal python range
for _ in tnrange(1,epochs+1,desc='Epoch'):
  print("<" + "="*22 + F" Epoch {_} "+ "="*22 + ">")
  # Calculate total loss for this epoch
  batch_loss = 0

  for step, batch in enumerate(train_dataloader):
    # Set our model to training mode (as opposed to evaluation mode)
    model.train()

    # Add batch to GPU
    batch = tuple(t.to(device) for t in batch)
    # Unpack the inputs from our dataloader
    b_input_ids, b_input_mask, b_labels = batch

    # Forward pass
    outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask, labels=b_labels)
    loss = outputs[0]

    # Backward pass
    loss.backward()

    # Clip the norm of the gradients to 1.0
    # Gradient clipping is not in AdamW anymore
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

    # Update parameters and take a step using the computed gradient
    optimizer.step()

    # Update learning rate schedule
    scheduler.step()

    # Clear the previous accumulated gradients
    optimizer.zero_grad()

    # Update tracking variables
    batch_loss += loss.item()

  # Calculate the average loss over the training data.
```



```
/home/nb6107/Desktop/workspace/research/sentimental_analysis_with_bert/venv/lib/python3.7/site-packages/ipykernel_launcher.py:9: TqdmDeprecation
Warning: Please use `tqdm.notebook.trange` instead of `tqdm.tnrange`
  if __name__ == '__main__':

Epoch: 0%                                    0/4 [00:00<?, ?it/s]

<====================== Epoch 1 ======================>

/home/nb6107/Desktop/workspace/research/sentimental_analysis_with_bert/venv/lib/python3.7/site-packages/torch/autograd/__init__.py:132: UserWarni
ng: CUDA initialization: The NVIDIA driver on your system is too old (found version 9010). Please update your GPU driver by downloading and installing a new ve
rsion from the URL: http://www.nvidia.com/Download/index.aspx Alternatively, go to: https://pytorch.org to install a PyTorch version that has been compiled wi
th your version of the CUDA driver. (Triggered internally at  /pytorch/c10/cuda/CUDAFunctions.cpp:100.)
  allow_unreachable=True)  # allow_unreachable flag
```

학습이 잘 되고 있음을 알 수 있다. 시간이 약 1시간 이상 걸린다.

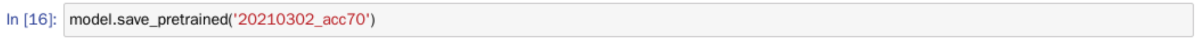


```
    Validation MCC Accuracy: [illegible]
<======================= Epoch 4 =======================>

    Current Learning rate:  0.0

    Average Training loss: 0.255813653902574O5

    Validation Accuracy: 0.7008928571428571

    Validation MCC Accuracy: 0.3602791554840682
```

약 70%의 Validation Accuracy의 성능을 보인다.
추가로 학습할 경우 가장 성능이 좋은 모델은 약 72%의 성능까지 학습이 되었고 72%의 정확도를 보이는 모델을 사용하기로 하였다.

13. 학습된 모델 저장하기

Model.save_pretrained(모델이름) 을 통해 학습된 모델을 저장할 수 있다.
학습한 날짜와 성능을 한눈에 알 수 있도록 모델이름을 설정하였다.



```
In [16]: model.save_pretrained('20210302_acc70')
```

□ 🗀 20210302_acc70 1분 전

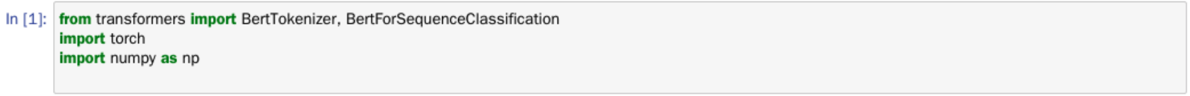
정상적으로 저장이 되었다.

5. Load Model & How to Use

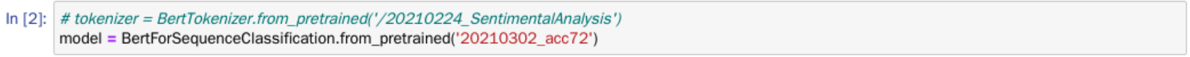
학습한 모델을 불러 사용할 수 있다.

1) 필요한 library를 import 하고, 학습된 모델을 불러온다.

.from_pretrained(모델이름) 을 사용하여 미리 학습한 모델을 불러올 수 있다.

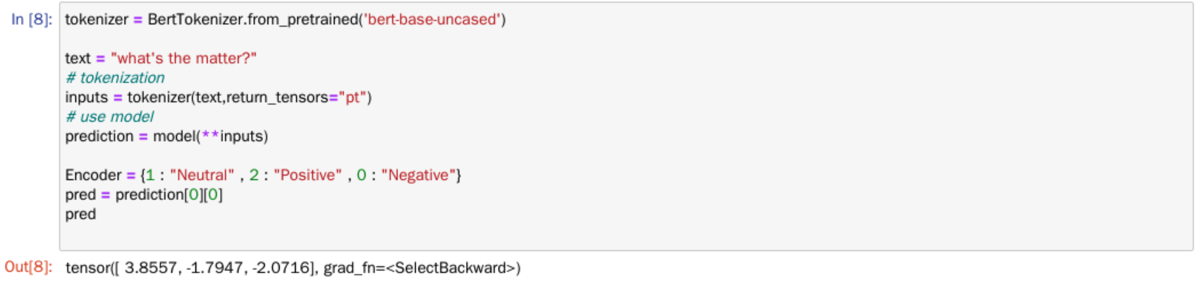


```
In [1]: from transformers import BertTokenizer, BertForSequenceClassification
        import torch
        import numpy as np
```



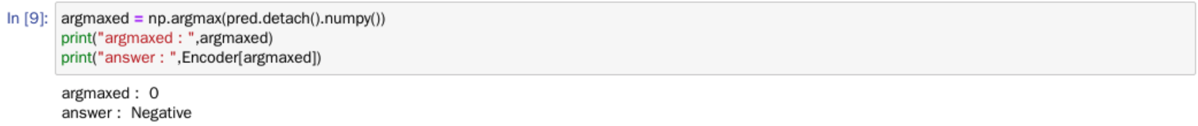
```
In [2]: # tokenizer = BertTokenizer.from_pretrained('/20210224_SentimentalAnalysis')
        model = BertForSequenceClassification.from_pretrained('20210302_acc72')
```

2) Tokenizer를 불러오고, 임의의 문장을 tokenization한 뒤 model에 넣어준다.



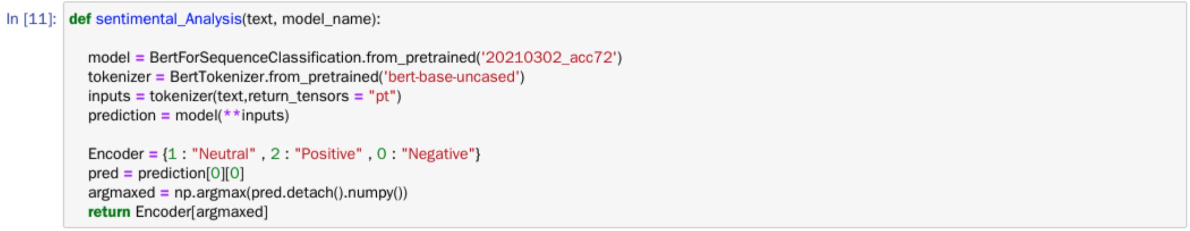
```
In [8]: tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

        text = "what's the matter?"
        # tokenization
        inputs = tokenizer(text,return_tensors="pt")
        # use model
        prediction = model(**inputs)

        Encoder = {1 : "Neutral" , 2 : "Positive" , 0 : "Negative"}
        pred = prediction[0][0]
        pred

Out[8]: tensor([ 3.8557, -1.7947, -2.0716], grad_fn=<SelectBackward>)
```

3) Argmax를 한 뒤 Decoding을 하면 text에 대한 감정값이 나오는 것을 볼 수 있다.

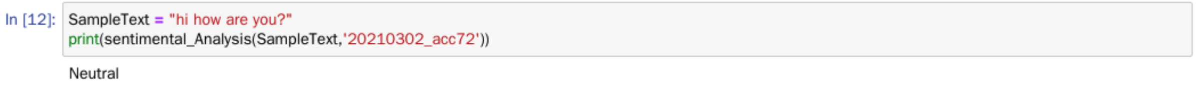


```
In [9]: argmaxed = np.argmax(pred.detach().numpy())
        print("argmaxed : ",argmaxed)
        print("answer : ",Encoder[argmaxed])

        argmaxed :  0
        answer :  Negative
```

함수로 만들면 다음과 같다.



```
In [11]: def sentimental_Analysis(text, model_name):

             model = BertForSequenceClassification.from_pretrained('20210302_acc72')
             tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
             inputs = tokenizer(text,return_tensors = "pt")
             prediction = model(**inputs)

             Encoder = {1 : "Neutral" , 2 : "Positive" , 0 : "Negative"}
             pred = prediction[0][0]
             argmaxed = np.argmax(pred.detach().numpy())
             return Encoder[argmaxed]
```



```
In [12]: SampleText = "hi how are you?"
         print(sentimental_Analysis(SampleText,'20210302_acc72'))

         Neutral
```

정상적으로 동작한다.

궁굼한 점이 있으시면 madogisa12@naver.com 으로 연락주시면 친절하게 답변드리겠습니다!

- 박정무

-참고문헌-

https://www.analyticsvidhya.com/blog/2020/07/transfer-learning-for-nlp-fine-tuning-bert-for-text-classification/

https://www.kaggle.com/praveengovi/classify-emotions-in-text-with-bert

https://medium.com/atheros/text-classification-with-transformers-in-tensorflow-2-bert-2f4f16eff5ad

https://gist.github.com/papapabi/124c6ac406e6bbd1f28df732e953ac6d

https://stackoverflow.com/questions/59978959/how-to-use-hugging-face-transformers-library-in-tensorflow-for-text-classificati