```
pip install transformers tqdm boto3 requests regex -q
```

```
import torch
from transformers import BertTokenizer
from IPython.display import clear_output

# BERT-BASE model, all words are lowercase
PRETRAINED_MODEL_NAME = "bert-base-uncased"

# get the tokenizer of this model
tokenizer = BertTokenizer.from_pretrained(PRETRAINED_MODEL_NAME)

clear_output()
print("PyTorch version: ", torch.__version__)
```

```
PyTorch version:  1.5.0+cu101
```

```
# connect to google drive to get the data
from google.colab import drive
drive.mount("/content/drive")

import pandas as pd
train = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data/train.csv', index_col=0)
train.to_csv("train.tsv", sep = "\t", index = False)
train.head(3)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

textID	text	selected_text	sentiment
cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
088c60f138	my boss is bullying me...	bullying me	negative

```
# delete the empty data
empty_title = ((train['text'].isnull()) | (train['selected_text'].isnull()))
df_train = train[~empty_title]
```

```
# see all the sentiment percent in the data
df_train.sentiment.value_counts() / len(df_train)
```

```
neutral     0.404549
positive    0.312300
negative    0.283151
Name: sentiment, dtype: float64
```

```
df_test = pd.read_csv('/content/drive/My Drive/Colab Notebooks/data/test.csv')
df_test = df_test.loc[:, ["text", "sentiment"]]
df_test.columns = ["text", "label"]
df_test.to_csv("test.tsv", sep="\t", index=False)

print("number of test data：", len(df_test))
df_test.head()
```

```
number of test data：  3534
```

	text	label
0	Last session of the day http://twitpic.com/67ezh	neutral
1	Shanghai is also really exciting (precisely -...	positive
2	Recession hit Veronique Branquinho, she has to...	negative
3	happy bday!	positive
4	http://twitpic.com/4w75p - I like it!!	positive

```
from torch.utils.data import Dataset

class Dataset(Dataset):
    # read tsv file and initial variables
    def __init__(self, mode, tokenizer):
        assert mode in ["Mytrain", "Mytest"]
        self.mode = mode
        # iterator = True while trainging the Big data
        self.df = pd.read_csv("/content/drive/My Drive/Colab Notebooks/data/" + mode + ".tsv", sep = "\t").fillna("")
        self.len = len(self.df)
        self.label_map = {'negative': 0, 'neutral': 1, 'positive': 2}
        # use BERT tokenizer
        self.tokenizer = tokenizer

    def __getitem__(self, idx):
        text, label = self.df.iloc[idx, :].values
        label_id = self.label_map[label]
        label_tensor = torch.tensor(label_id)

        # create BERT tokens [CLS] of sentence and add the separation sign [SEP]
        word_pieces = ["[CLS]"]
        tokens_a = self.tokenizer.tokenize(text)
        word_pieces += tokens_a + ["[SEP]"]
        len_a = len(word_pieces)

        # convert tokens to ids
        ids = self.tokenizer.convert_tokens_to_ids(word_pieces)
        tokens_tensor = torch.tensor(ids)

        return (tokens_tensor, label_tensor)
```

```
    def __len__(self):
        return self.len


# initial a Dataset to read training data
trainset = Dataset("Mytrain", tokenizer=tokenizer)
```

```
sample_idx = 0

# compare to the original data
text, label = trainset.df.iloc[sample_idx].values

# get the id tensors from Dataset
tokens_tensor, label_tensor = trainset[sample_idx]

# convert tokens_tensor to original words
tokens = tokenizer.convert_ids_to_tokens(tokens_tensor.tolist())
combined_text = " ".join(tokens)

print(f"""[original sentence]
sentence：{text}
label   ：{label}
--------------------
[tensors from Dataset]
tokens_tensor   ：{tokens_tensor}
label_tensor    ：{label_tensor}
--------------------
[convert tokens_tensors to original words]
{combined_text}""")
```

```
[original sentence]
sentence：  I`d have responded, if I were going
label   ：neutral
--------------------
[tensors from Dataset]
tokens_tensor   ：tensor([ 101, 1045, 1036, 1040, 2031, 5838, 1010, 2065, 1045, 2020, 2183,  102])
label_tensor    ：1
--------------------
[convert tokens_tensors to original words]
[CLS] i ` d have responded , if i were going [SEP]
```

```
from torch.utils.data import DataLoader
from torch.nn.utils.rnn import pad_sequence

# input of this function (samples) is a list, it contain 2 tensors return from Dataset
# tokens_tensors : (batch_size, max_seq_len_in_batch) & label_ids : (batch_size)
def create_mini_batch(samples):
    tokens_tensors = [s[0] for s in samples]

    if samples[0][1] is not None:
        label_ids = torch.stack([s[1] for s in samples])
    else:
        label_ids = None

    # zero pad to the same size
    tokens_tensors = pad_sequence(tokens_tensors, batch_first=True)

    # attention masks: set the value to 1 if tokens_tensors is not zero padding
    # so BERT can just pay attention to those non-zero tokens
    masks_tensors = torch.zeros(tokens_tensors.shape, dtype=torch.long)
    masks_tensors = masks_tensors.masked_fill(tokens_tensors != 0, 1)

    # return tokens_tensors, segments_tensors, masks_tensors, label_ids
    return tokens_tensors, masks_tensors, label_ids


# initial a 32 batch size DataLoader
# use "collate_fn" to combine list of samples to a mini-batch
BATCH_SIZE = 32
trainloader = DataLoader(trainset, batch_size = BATCH_SIZE, collate_fn = create_mini_batch)
```

```
data = next(iter(trainloader))

tokens_tensors, masks_tensors, label_ids = data
```

```
print(f"""
tokens_tensors.shape = {tokens_tensors.shape}
{tokens_tensors}
------------------------
masks_tensors.shape  = {masks_tensors.shape}
{masks_tensors}
------------------------
label_ids.shape      = {label_ids.shape}
{label_ids}
""")
```

```
tokens_tensors.shape = torch.Size([32, 51])
tensor([[  101,  1045,  1036,  ...,     0,     0,     0],
        [  101, 17111,  2080,  ...,     0,     0,     0],
        [  101,  2026,  5795,  ...,     0,     0,     0],
        ...,
        [  101,  2253,  2000,  ...,     0,     0,     0],
        [  101,  1045,  1036,  ...,     0,     0,     0],
        [  101,  1045,  3246,  ...,     0,     0,     0]])
------------------------
masks_tensors.shape  = torch.Size([32, 51])
tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
```

```
       [1, 1, 1,  ..., 0, 0, 0],
       [1, 1, 1,  ..., 0, 0, 0],
       [1, 1, 1,  ..., 0, 0, 0]])
-----------------------
label_ids.shape      = torch.Size([32])
tensor([1, 0, 0, 0, 0, 1, 2, 1, 1, 2, 1, 2, 0, 0, 1, 0, 0, 0, 0, 1, 1, 2, 1, 1,
       1, 2, 0, 0, 2, 0, 2, 2])
```

```
from transformers import BertForSequenceClassification

PRETRAINED_MODEL_NAME = "bert-base-uncased"
NUM_LABELS = 3

model = BertForSequenceClassification.from_pretrained(PRETRAINED_MODEL_NAME, num_labels = NUM_LABELS)

clear_output()

# show modules in this model
print("""
name            module
----------------------""")
for name, module in model.named_children():
    if name == "bert":
        for n, _ in module.named_children():
            print(f"{name}:{n}")
    else:
        print("{:15} {}".format(name, module))
```

```
name            module
----------------------
bert:embeddings
bert:encoder
bert:pooler
dropout         Dropout(p=0.1, inplace=False)
classifier      Linear(in_features=768, out_features=3, bias=True)
```

```
def get_predictions(model, dataloader):
    predictions = None
    correct = 0
    total = 0

    with torch.no_grad():
        for data in dataloader:
            # put all tensors on GPU
            if next(model.parameters()).is_cuda:
                data = [t.to("cuda:0") for t in data if t is not None]

            # put the two tensors and their parameter names in the model
            tokens_tensors, masks_tensors = data[:2]
            outputs = model(input_ids = tokens_tensors, attention_mask = masks_tensors)

            logits = outputs[0]
            _, pred = torch.max(logits.data, 1)

            # calculate the accuracy of our classification
            labels = data[2]
            total += labels.size(0)
            correct += (pred == labels).sum().item()

            # record the current batch
            if predictions is None:
                predictions = pred
            else:
                predictions = torch.cat((predictions, pred))
    acc = correct / total
    return predictions, acc

device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
```

```
def get_learnable_params(module):
    return [p for p in module.parameters() if p.requires_grad]

model_params = get_learnable_params(model)
clf_params = get_learnable_params(model.classifier)
```

```
# start training
model.train()

# use Adam Optim to update all the parameters
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-5)

EPOCHS = 10
for epoch in range(EPOCHS):

    running_loss = 0.0
    for data in trainloader:

        tokens_tensors, masks_tensors, labels = [t.to(device) for t in data]

        # zero the parameters gradient
        optimizer.zero_grad()

        # forward pass
        outputs = model(input_ids = tokens_tensors, attention_mask = masks_tensors, labels = labels)

        loss = outputs[0]
        # backward
        loss.backward()
```

```
        optimizer.step()


        # record the current batch loss
        running_loss += loss.item()

    # calculate the accuracy
    predictions, acc = get_predictions(model, trainloader)

    print('[epoch %d] loss: %.3f, acc: %.3f' % (epoch + 1, running_loss, acc))
```

```
[epoch 1] loss: 532.257, acc: 0.814
[epoch 2] loss: 389.787, acc: 0.847
[epoch 3] loss: 316.021, acc: 0.889
[epoch 4] loss: 240.471, acc: 0.922
[epoch 5] loss: 173.932, acc: 0.929
[epoch 6] loss: 127.552, acc: 0.951
[epoch 7] loss: 99.833, acc: 0.962
[epoch 8] loss: 79.566, acc: 0.960
[epoch 9] loss: 65.384, acc: 0.974
[epoch 10] loss: 54.432, acc: 0.982
```

```
testset = Dataset("Mytest", tokenizer = tokenizer)
testloader = DataLoader(testset, batch_size = 512, collate_fn = create_mini_batch)

# test the accuracy of test data
predictions, acc = get_predictions(model, testloader)
print('acc: ' , acc)
```

```
acc:  0.7866440294284097
```