```
import torch

if torch.cuda.is_available():

current_device = torch.device("cuda")

print(' %d GPU available.' %d torch.cuda.device_count())

print('GPU Name:', torch.cuda.get_device_name(0))
else:

print('No GPU’ using the CPU instead.')

current_device = torch.device("cpu")

import requests
import pandas as pd
import io

data_urls=[
'http://saifmohammad.com/WebDocs/EmoInt%20Train%20Data/anger-ratings-0to1.train.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Train%20Data/fear-ratings-0to1.train.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Train%20Data/joy-ratings-0to1.train.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Train%20Data/sadness-ratings-
0to1.train.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Dev%20Data%20With%20Gold/anger-
ratings-0to1.dev.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Dev%20Data%20With%20Gold/fear-
ratings-0to1.dev.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Dev%20Data%20With%20Gold/joy-
ratings-0to1.dev.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Dev%20Data%20With%20Gold/sadness-
ratings-0to1.dev.gold.txt',
]

frams=[]
for data_url in data_urls:

x=requests.get(data_url,allow_redirects=True,headers={"User-Agent": "XY"})

df1=pd.read_csv(io.StringIO(x.text),sep='\t',lineterminator='\n',header=None)


frams.append(df1)

import pandas as pd

column_names= ['id','sentence','emotion','intensity']
tonumber= {'anger':0,'fear':1,'joy':2,'sadness':3}
```

```
df.columns=column_names
print('No. of training data: {:,}\n'.format(df.shape[0]))

df.sample(20)

df['label'] =df.apply (lambda row: tonumber[row.emotion], axis=1)

sentences = df.sentence.values
labels = df.label.values

print(' Original: ', sentences[1])
print('Tokenized: ', tokenizer.tokenize(sentences[1]))
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[1])))

input_ids = []

for sentence in sentences:

encoded_sentence = tokenizer.encode(sentence,add_special_tokens = True)

input_ids.append(encoded_sentence)

print('Original ', sentences[1])
print('Token IDs', input_ids[1])

print('Max sentence length is ', max([len(sentence) for sentence in input_ids]))

from keras.preprocessing.sequence import pad_sequences

maxl= 128

print('padding or truncating all sentences to %d values:' % maxl)

print('padding token: "{:}", id’ {:}'.format(tokenizer.pad_token, tokenizer.pad_token_id))

input_ids = pad_sequences(input_ids, maxlen=mal, dtype="long",
                          value=0, truncating="post", padding="post")


attention_masks = []
for sent in input_ids:

att_mask = [int(token_id > 0) for token_id in sent]

attention_masks.append(att_mask)

from sklearn.model_selection import train_test_split

train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(input_ids,
labels,
```

```
random_state=2018, test_size=0.1)

train_masks, validation_masks, _, _ = train_test_split(attention_masks, labels,

random_state=2018, test_size=0.1)


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)

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

batch_size = 32

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 = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler,
batch_size=batch_size)


from transformers import BertForSequenceClassification
from transformers import  AdamW
from transformers import  BertConfig
from transformers import get_linear_schedule_with_warmup

used_bert_model= bert-base-uncased
model = BertForSequenceClassification.from_pretrained(used_bert_model,num_labels =
4,output_attentions = False,

output_hidden_states = False)

model.cuda()

optimizer = AdamW(model.parameters(),lr = 2e-5,eps = 1e-8)
epochs = 4
total_steps = len(train_dataloader) epochs
scheduler = get_linear_schedule_with_warmup(optimizer,num_warmup_steps =
0,num_training_steps = total_steps)
```

```
import numpy as np

def flat_accuracy(preds, labels):

pred_flat = np.argmax(preds, axis=1).flatten()

labels_flat = labels.flatten()

return np.sum(pred_flat == labels_flat) / len(labels_flat)

import time
import datetime

def format_time(elapsed):

elapsed_rounded = int(round((elapsed)))

return str(datetime.timedelta(seconds=elapsed_rounded))

import random

seed_val = 42

random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)

loss_values = []

for epoch_i in range(0, epochs):


print("")

print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))

print('Training...')


t0 = time.time()


total_loss = 0

model.train()


for step, batch in enumerate(train_dataloader):
```

```
if step % 40 == 0 and not step == 0:

elapsed = format_time(time.time() - t0)
          print('
Batch {:>5,} of {:>5,}.
Elapsed: {:}.'.format(step, len(train_dataloader), elapsed))


b_input_ids = batch[0].to(device)

b_input_mask = batch[1].to(device)

b_labels = batch[2].to(device)

b_labels = batch[3].to(device)


model.zero_grad()


outputs = model(b_input_ids,
token_type_ids=None,attention_mask=b_input_mask,labels=b_labels)

loss = outputs[0]

total_loss += loss.item()

loss.backward()

torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

optimizer.step()

scheduler.step()


avg_train_loss = total_loss / len(train_dataloader)

loss_values.append(avg_train_loss)


print("")

print(" Average training loss: {0:.2f}".format(avg_train_loss))

print(" Training epcoh took: {:}".format(format_time(time.time() - t0)))


print("")
```

```
print("Running Validation...")

t0 = time.time()

model.eval()

eval_loss, eval_accuracy = 0, 0

nb_eval_steps, nb_eval_examples = 0, 0

for batch in validation_dataloader:

batch = tuple(t.to(device) for t in batch)

b_input_ids, b_input_mask, b_labels = batch

with torch.no_grad():

outputs = model(b_input_ids,token_type_ids=None,attention_mask=b_input_mask)

logits = outputs[0]

logits = logits.detach().cpu().numpy()

label_ids = b_labels.to('cpu').numpy()

tmp_eval_accuracy = flat_accuracy(logits, label_ids)

eval_accuracy += tmp_eval_accuracy

nb_eval_steps += 1

print(" Accuracy: {0:.2f}".format(eval_accuracy/nb_eval_steps))

print(" Validation took: {:}".format(format_time(time.time() - t0)))

print("")
print("Training complete!")

import matplotlib.pyplot as plt
% matplotlib inline

import seaborn as sns
```

```
# Use plot styling from seaborn.
sns.set(style='darkgrid')

# Increase the plot size and font size.
sns.set(font_scale=1.5)
plt.rcParams["figure.figsize"] = (12,6)

# Plot the learning curve.
plt.plot(loss_values, 'b-o')

# Label the plot.
plt.title("Training loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")

plt.show()

import pandas as pd

data_urls=[
'http://saifmohammad.com/WebDocs/EmoInt%20Test%20Gold%20Data/anger-ratings-
0to1.test.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Test%20Gold%20Data/fear-ratings-
0to1.test.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Test%20Gold%20Data/joy-ratings-
0to1.test.gold.txt',
'http://saifmohammad.com/WebDocs/EmoInt%20Test%20Gold%20Data/sadness-ratings-
0to1.test.gold.txt'
]

frams=[]
for data_url in data_urls:

x=requests.get(data_url,allow_redirects=True,headers={"User-Agent": "XY"})

df1=pd.read_csv(io.StringIO(x.text),sep='\t',lineterminator='\n',header=None)

frams.append(df1)
    df=pd.concat(frams)

column_names= ['id','sentence','emotion','intensity']
tonumber= {'anger':0,'fear':1,'joy':2,'sadness':3}
df.columns=column_names
df['label'] =df.apply (lambda row: tonumber[row.emotion], axis=1)

sentences = df.sentence.values
labels = df.label.values

input_ids = []
```

```
for sent in sentences:

encoded_sent = tokenizer.encode(sent,add_special_tokens = True,)

input_ids.append(encoded_sent)
input_ids = pad_sequences(input_ids, maxlen=maxl,dtype="long", truncating="post",
padding="post")

attention_masks = []

for seq in input_ids:

seq_mask = [float(i>0) for i in seq]

attention_masks.append(seq_mask)

prediction_inputs = torch.tensor(input_ids)
prediction_masks = torch.tensor(attention_masks)
prediction_labels = torch.tensor(labels)

batch_size = 32

prediction_data = TensorDataset(prediction_inputs, prediction_masks, prediction_labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler,
batch_size=batch_size)

model.eval()
predictions , true_labels = [], []

for batch in prediction_dataloader:

batch = tuple(t.to(device) for t in batch)

b_input_ids, b_input_mask, b_labels = batch


with torch.no_grad():

outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask)


logits = outputs[0]

logits = logits.detach().cpu().numpy()

label_ids = b_labels.to('cpu').numpy()

predictions.append(logits)
```

```
true_labels.append(label_ids)
```