

▼ Installing Dependencies

```
!pip install transformers
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting transformers

Downloading transformers-4.20.1-py3-none-any.whl (4.4 MB)

|██| 4.4 MB 7.0 MB/s

Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.64.0)

Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2022.6.0)

Collecting huggingface-hub<1.0,>=0.1.0

Downloading huggingface_hub-0.8.1-py3-none-any.whl (101 kB)

|██████████████████████████████████████| 101 kB 13.0 MB/s

Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)

Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.21.6)

Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers) (4.12.0)

Collecting pyyaml>=5.1

Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)

|██████████████████████████████████████| 596 kB 73.4 MB/s

Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.7.1)

Collecting tokenizers!=0.11.3,<0.13,>=0.11.1

Downloading tokenizers-0.12.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (6.6 MB)

|██████████████████████████████████████| 6.6 MB 48.8 MB/s

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers) (21.3)

Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from huggingface-hub) (4.1.1)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0) (3.0.9)

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers) (3.6.0)

Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2.10)

Requirement already satisfied: urllib3!=1.25.0,!<1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (1.26.13)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2022.9.24)

Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.0.4)

Installing collected packages: pyyaml, tokenizers, huggingface-hub, transformers

Attempting uninstall: pyyaml

Found existing installation: PyYAML 3.13

Uninstalling PyYAML-3.13:

Successfully uninstalled PyYAML-3.13

Successfully installed huggingface-hub-0.8.1 pyyaml-6.0 tokenizers-0.12.1 transformers-4.20.1

```
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
import torch
import gc
gc.collect()
torch.cuda.empty_cache()

import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import defaultdict
from textwrap import wrap

from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

%matplotlib inline
%config InlineBackend.figure_format='retina'

sns.set(style='whitegrid', palette='muted', font_scale=1.2)

HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02", "#8F00FF"]

sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))

rcParams['figure.figsize'] = 12, 8

RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
```

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

device(type='cuda', index=0)
```

▼ Read the Data

```
df = pd.read_csv("/content/Pelabelan1.csv")
df.head()
```

	category	review content	lemma	rating	Subjectivity	Polarity	TextBlob	Vader Sentiment	vaderSentiment
0	Headsets	This gaming headset ticks all the boxes # look...	['game', 'headset', 'tick', 'box', 'look', 'gr...]	5	0.583333	0.305556	Positive	0.8720	Positive
1	Headsets	Easy setup, rated for 6 hours battery but mine...	['easy', 'setup', 'rat', 'hours', 'battery', '...]	3	0.546289	0.203782	Positive	0.9657	Positive
		I originally bought	I'originally' 'buiv'						

```
df.shape
```

```
(44756, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44756 entries, 0 to 44755
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype
---  -
0   category        44756 non-null  object
1   review content  44756 non-null  object
```

```
2 lemma          44756 non-null object
3 rating          44756 non-null int64
4 Subjectivity    44756 non-null float64
5 Polarity        44756 non-null float64
6 TextBlob        44756 non-null object
7 Vader Sentiment 44756 non-null float64
8 vaderSentiment  44756 non-null object
dtypes: float64(3), int64(1), object(5)
memory usage: 3.1+ MB
```

```
sns.countplot(df.rating)
plt.xlabel('review score');
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: I
FutureWarning
```

We have a balanced dataset with respect to reviews (as we scraped it in a balanced way)

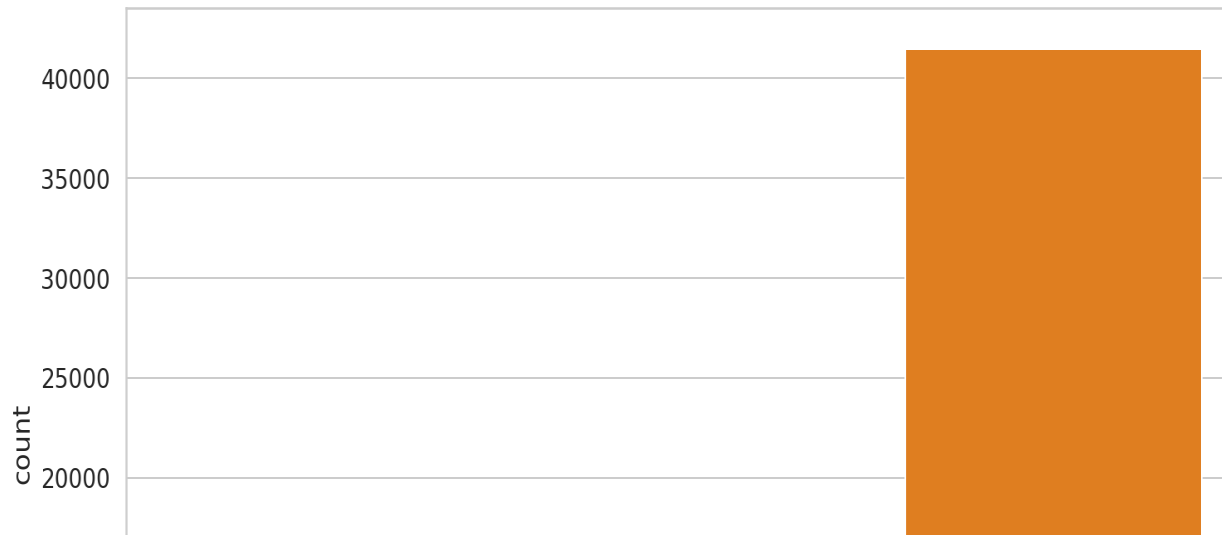
```
def group_sentiment(rating):
    #rating = int(rating)
    if rating <= 2:
        return 0          # Negative sentiment
    elif rating == 3:
        return 1          # Neutral Sentiment
    else:
        return 2          # positive Sentiment

df['TextBlob'] = df.rating.apply(group_sentiment)
```

```
class_names = ['negative', 'neutral', 'positive']
```

```
ax = sns.countplot(df.TextBlob)
plt.xlabel('review sentiment')
ax.set_xticklabels(class_names);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: I
FutureWarning
```



▼ Data Preprocessing

Machine Learning models don't work with raw text. We need to convert text to numbers (of some sort). BERT requires even more attention. Here are the requirements:

- Add special tokens to separate sentences and do classification
- Pass sequences of constant length (introduce padding)
- Create array of 0s (pad token) and 1s (real token) called attention mask

The Transformers library provides a wide variety of Transformer models (including BERT). It also includes prebuilt tokenizers that solves most of the load required for pre-processing

```
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
```

Let's load a pre-trained BertTokenizer:

```
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Downloading: 100%

208k/208k [00:00<00:00, 810kB/s]

Downloading: 100%

29.0/29.0 [00:00<00:00, 857B/s]

Downloading: 100%

570/570 [00:00<00:00, 6.47kB/s]

The requirements of special tokens which indicate separation between sentences is taken care by the tokenizer provided by the huggingface. BERT was trained for question and answering task but we are using it to train with a sequence of sentences as input. We require sequences of equal length which is done by padding tokens of zero meaning without loss of generality to the end of sentences.

We also need to pass attention mask which is basically passing a value of 1 to all the tokens which have meaning and 0 to padding tokens.

- ▶ We'll use this simple text to understand the tokenization process:

```
[ ] ↳ 3 cells hidden
```

- ▶ Special Tokens

[SEP] - marker for ending of a sentence

```
[ ] ↳ 7 cells hidden
```

- ▼ All of the above work can be done using the `encode_plus()` method

```
encoding_test = tokenizer.encode_plus(  
    sample_txt,  
    max_length=32,          # sequence length
```

```

add_special_tokens=True, # Add '[CLS]' and '[SEP]'
return_token_type_ids=False,
pad_to_max_length=True,
return_attention_mask=True,
return_tensors='pt', # Return PyTorch tensors(use tf for tensorflow and keras)
)

```

Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to
 /usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length`
 FutureWarning,

```
encoding_test.keys()
```

```
dict_keys(['input_ids', 'attention_mask'])
```

```

print(' length of the first sequence is : ', len(encoding_test['input_ids'][0]))
print('\n The input id's are : \n', encoding_test['input_ids'][0])
print('\n The attention mask generated is : ', encoding_test['attention_mask'][0])

```

```
length of the first sequence is : 32
```

```
The input id's are :
```

```

tensor([ 101, 1798, 1282, 1115, 146, 1138, 3891, 136, 10271, 1108,
        1103, 1211, 2712, 1105, 146, 4615, 1991, 6918, 1106, 1138,
        3891, 10271, 1120, 1216, 1126, 1346, 1425, 119, 102, 0,
         0, 0])

```

```

The attention mask generated is : tensor([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])

```

We can inverse the tokenization to have a look at the special tokens:

```
tokenizer.convert_ids_to_tokens(encoding_test['input_ids'].flatten())
```

```

['[CLS]',
 'Best',

```



```
'place',  
'that',  
'I',  
'have',  
'visited',  
'?',  
'Iceland',  
'was',  
'the',  
'most',  
'beautiful',  
'and',  
'I',  
'consider',  
'myself',  
'lucky',  
'to',  
'have',  
'visited',  
'Iceland',  
'at',  
'such',  
'an',  
'early',  
'age',  
'.',  
'[SEP]',  
'[PAD]',  
'[PAD]',  
'[PAD]']
```

▼ Choosing Sequence Length

Check if there are any nan in the content column

```
df.loc[df['review content'].isnull()]
```

```

category review content lemma rating Subjectivity Polarity TextBlob Vader Sentiment vaderSentiment
df = df[df['review content'].notna()]
df.head()

```

	category	review content	lemma	rating	Subjectivity	Polarity	TextBlob	Vader Sentiment	vaderSentiment
0	Headsets	This gaming headset ticks all the boxes # look...	['game', 'headset', 'tick', 'box', 'look', 'gr...']	5	0.583333	0.305556	2	0.8720	Positive
1	Headsets	Easy setup, rated for 6 hours battery but mine...	['easy', 'setup', 'rat', 'hours', 'battery', '...']	3	0.546289	0.203782	1	0.9657	Positive
		I originally bought	['originally', 'buy']						

BERT works with fixed-length sequences. We'll use a simple strategy to choose the max length. Let's store the token length of each review:

```

token_lens = []
for text in df['review content']:
    tokens_df = tokenizer.encode(text, max_length=512) # Max possible length for the BERT model.
    token_lens.append(len(tokens_df))

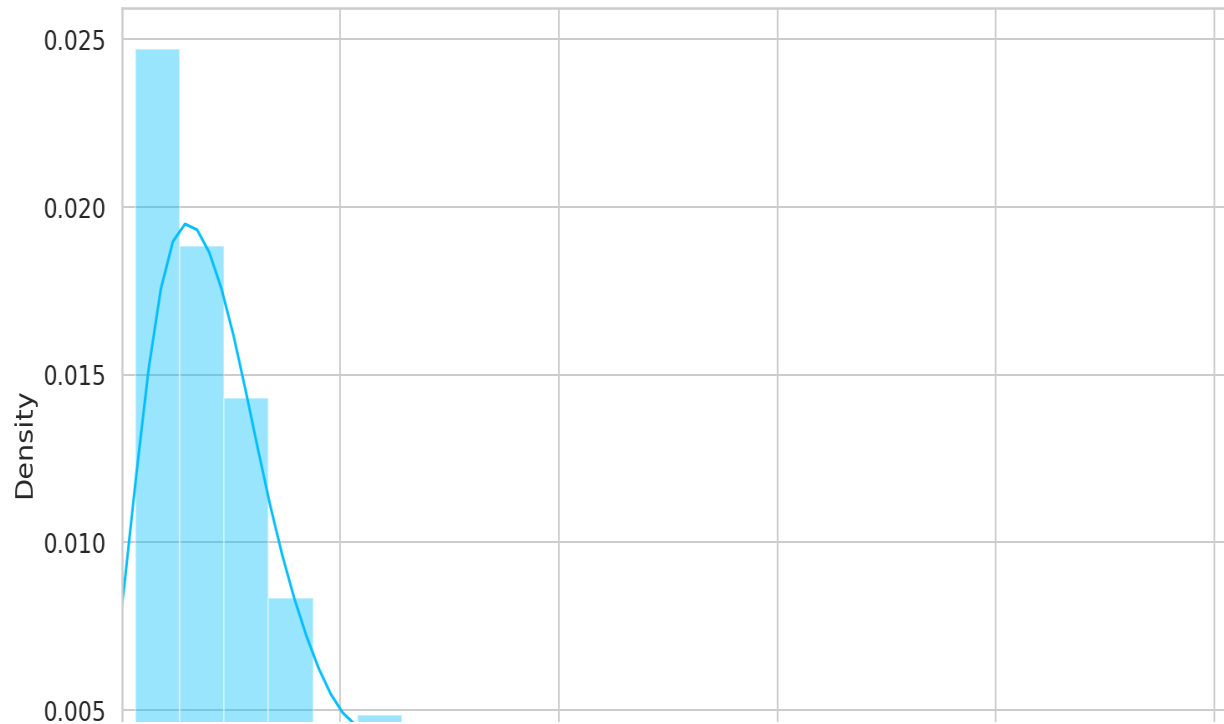
```

```

sns.distplot(token_lens)
plt.xlim([0, 256]);
plt.xlabel('Token count');

```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2619: FutureWarning:
warnings.warn(msg, FutureWarning)
```



Most of the reviews seem to contain less than 128 tokens, but we'll be on the safe side and choose a maximum length of 160.

```
MAX_LEN = 160
```

We have all building blocks required to create a PyTorch dataset. Let's use the same class:

```
class GPReviewDataset(Dataset):

    def __init__(self, reviews, targets, tokenizer, max_len):
        self.reviews = reviews          # Reviews is content column.
        self.targets = targets           # Target is the sentiment column.
        self.tokenizer = tokenizer       # Tokenizer is the BERT_Tokanizer.
        self.max_len = max_len           # max_length of each sequence.
```

```

def __len__(self):
    return len(self.reviews)          # Len of each review.

def __getitem__(self, item):
    review = str(self.reviews[item])  # returns the string of reviews at the index = 'items'
    target = self.targets[item]       # returns the string of targets at the index = 'items'

    encoding = self.tokenizer.encode_plus(
        review,
        add_special_tokens=True,
        max_length=self.max_len,
        return_token_type_ids=False,
        pad_to_max_length=True,
        return_attention_mask=True,
        return_tensors='pt',
    )

    return {
        'review_text': review,
        'input_ids': encoding['input_ids'].flatten(),
        'attention_mask': encoding['attention_mask'].flatten(),
        'targets': torch.tensor(target, dtype=torch.long)          # dictionary containing all the features is returned.
    }

```

The tokenizer is doing most of the heavy lifting for us. We also return the review texts, so it'll be easier to evaluate the predictions from our model. Let's split the data:

▼ Splitting into train and validation sets

```

df_train, df_test = train_test_split(df, test_size=0.2, random_state=RANDOM_SEED)
df_val, df_test = train_test_split(df_test, test_size=0.5, random_state=RANDOM_SEED)

```

```
df_train.shape, df_val.shape, df_test.shape
```

```
((35804, 9), (4476, 9), (4476, 9))
```

- ▼ Create data loaders for to feed as input to our model. The below function does that.

```
def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = GPReviewDataset(
        reviews=df['review content'].values,
        targets=df['TextBlob'].values,
        tokenizer=tokenizer,
        max_len=max_len
    )
    # Dataset would be created which can be used to create and return dataloader.

    return DataLoader(
        ds,
        batch_size=batch_size,
        #num_workers=4
    )
```

```
BATCH_SIZE = 8
```

```
train_data_loader = create_data_loader(df_train, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LEN, BATCH_SIZE)
```

Let's have a look at an example batch from our training data loader:

```
data = next(iter(train_data_loader))
data.keys()
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_len`
FutureWarning,
dict_keys(['review_text', 'input_ids', 'attention_mask', 'targets'])
```

```
print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['targets'].shape)
```

```
torch.Size([8, 160])
torch.Size([8, 160])
torch.Size([8])
```

▼ Sentiment Classification with BERT and Hugging Face

There are a lot of helpers that make using BERT easy with the Transformers library. Depending on the task we might use "BertForSequenceClassification", "BertForQuestionAnswering" or something else.

We'll use the basic BertModel and build our sentiment classifier on top of it. Let's load the model:

```
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
Downloading: 416M/416M [00:07<00:00,
100% 54.4MB/s]
```

```
Some weights of the model checkpoint at bert-base-cased were not used when initiali
- This IS expected if you are initializing BertModel from the checkpoint of a model
```

```
model_test = bert_model(
    input_ids=encoding_test['input_ids'],
    attention_mask=encoding_test['attention_mask']
)
model_test.keys()

dict_keys(['last_hidden_state', 'pooler_output'])
```

The "last_hidden_state" is a sequence of hidden states of the last layer of the model. Obtaining the "pooled_output" is done by applying the BertPooler which basically applies the tanh function to pool all the outputs.

```
last_hidden_state=model_test['last_hidden_state']  
pooled_output=model_test['pooler_output']
```

```
last_hidden_state.shape
```

```
torch.Size([1, 32, 768])
```

We have the hidden state for each of our 32 tokens (the length of our example sequence) and 768 is the number of hidden units in the feedforward-networks. We can verify that by checking the config:

```
bert_model.config.hidden_size
```

```
768
```

We can think of the pooled_output as a summary of the content, according to BERT. Let's look at the shape of the output:

```
pooled_output.shape
```

```
torch.Size([1, 768])
```

We can use all this knowledge to create a sentiment classifier that uses the BERT model:

```
class SentimentClassifier(nn.Module):  
  
    def __init__(self, n_classes):  
        super(SentimentClassifier, self).__init__()  
        self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)  
        self.drop = nn.Dropout(p=0.3)                ## For regularization with dropout probability 0.3.
```

```

self.out = nn.Linear(self.bert.config.hidden_size, n_classes) ## append an Output fully connected layer representing the

def forward(self, input_ids, attention_mask):
    returned = self.bert(
        input_ids=input_ids,
        attention_mask=attention_mask
    )
    pooled_output = returned["pooler_output"]
    output = self.drop(pooled_output)
    return self.out(output)

```

The classifier delegates most of the work to the BertModel. We use a dropout layer for some regularization and a fully-connected layer for our output. We're returning the raw output of the last layer since that is required for the cross-entropy loss function in PyTorch to work.

This should work like any other PyTorch model. Create an instance and move it to the GPU:

```

model = SentimentClassifier(len(class_names))
model = model.to(device)

```

Some weights of the model checkpoint at bert-base-cased were not used when initializing BertModel: ['cls.predictions.decoder']

- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical to

We'll move the example batch of our training data created above using dataloader to the GPU:

```

input_ids = data['input_ids'].to(device)
attention_mask = data['attention_mask'].to(device)

print(input_ids.shape)      # batch size x seq length
print(attention_mask.shape) # batch size x seq length

torch.Size([8, 160])
torch.Size([8, 160])

```


To get the predicted probabilities from our trained model, we'll apply the softmax function to the output obtained from the output layer:

```
F.softmax(model(input_ids, attention_mask), dim=1)

tensor([[0.2277, 0.4702, 0.3021],
        [0.2640, 0.2790, 0.4570],
        [0.2939, 0.2905, 0.4156],
        [0.2085, 0.3749, 0.4166],
        [0.5549, 0.2260, 0.2191],
        [0.2835, 0.3423, 0.3742],
        [0.2556, 0.3621, 0.3823],
        [0.4141, 0.2168, 0.3691]], device='cuda:0', grad_fn=<SoftmaxBackward0>)
```

▼ Training the model

To reproduce the training procedure from the BERT paper, we'll use the AdamW optimizer provided by Hugging Face. It corrects weight decay, so it's similar to the original paper. We'll also use a linear scheduler with no warmup steps:

```
EPOCHS = 10

optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS    # Number of batches * Epochs (Required for the scheduler.)

scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,      # Recommended in the BERT paper.
    num_training_steps=total_steps
)

loss_fn = nn.CrossEntropyLoss().to(device)
```

/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is

FutureWarning,

The BERT authors have some recommendations for fine-tuning:

- Batch size: 16, 32
- Learning rate (Adam): 5e-5, 3e-5, 2e-5
- Number of epochs: 2, 3, 4

Except for the number of epochs recommendation We'll stick with the rest. Increasing the batch size reduces the training time significantly, but gives lower accuracy.

▼ Helper function for training our model for one epoch:

```
def train_epoch(
    model,
    data_loader,
    loss_fn,
    optimizer,
    device,
    scheduler,
    n_examples
):
    model = model.train()    # To make sure that the dropout and normalization is enabled during the training.

    losses = []
    correct_predictions = 0

    for d in data_loader:
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        targets = d["targets"].to(device)

        outputs = model(
            input_ids=input_ids,
```

```

        attention_mask=attention_mask
    )

    max_prob, preds = torch.max(outputs, dim=1)    # Returns 2 tensors, one with max_probability and another with the respective loss
    loss = loss_fn(outputs, targets)

    correct_predictions += torch.sum(preds == targets)
    losses.append(loss.item())

    loss.backward()    # Back_Propagation
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0) # Recommended by the BERT paper to clip the gradients to avoid exploding gradients
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()

    return correct_predictions.double() / n_examples, np.mean(losses)    # Return the mean loss and the ratio of correct predictions

```

Training the model is similar to training a deep neural network, except for two things. The scheduler gets called every time a batch is fed to the model. We're avoiding exploding gradients by clipping the gradients of the model using clip_grad_norm

▼ Helper function to evaluate the model on a given data loader:

```

def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()    # To make sure that the dropout and normalization is disabled during the training.

    losses = []
    correct_predictions = 0

    with torch.no_grad():    # Back propagation is not required. Torch would perform faster.
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)

            outputs = model(
                input_ids=input_ids,

```

```

        attention_mask=attention_mask
    )
    max_prob, preds = torch.max(outputs, dim=1)

    loss = loss_fn(outputs, targets)

    correct_predictions += torch.sum(preds == targets)
    losses.append(loss.item())

return correct_predictions.double() / n_examples, np.mean(losses)

```

Using these two helper functions, we can write our training loop. We'll also store the training history:

```

%%time

history = defaultdict(list)          # Similar to Keras library saves history
best_accuracy = 0

for epoch in range(EPOCHS):

    print(f'Epoch {epoch + 1}/{EPOCHS}')
    print('-' * 10)

    train_acc, train_loss = train_epoch(
        model,
        train_data_loader,
        loss_fn,
        optimizer,
        device,
        scheduler,
        len(df_train)
    )

    print(f'Train loss {train_loss} accuracy {train_acc}')

    val_acc, val_loss = eval_model(

```

```


model,
val_data_loader,
loss_fn,
device,
len(df_val)
)

print(f'Val   loss {val_loss} accuracy {val_acc}')
print()

history['train_acc'].append(train_acc)
history['train_loss'].append(train_loss)
history['val_acc'].append(val_acc)
history['val_loss'].append(val_loss)

if val_acc > best_accuracy:
    torch.save(model.state_dict(), 'best_model_state.bin')
    best_accuracy = val_acc

```

 /usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version. Please use the `padding` argument instead.

FutureWarning,

Epoch 1/10

Train loss 0.23394509967126345 accuracy 0.9410121774103452

Val loss 0.21611709370429577 accuracy 0.9423592493297587

Epoch 2/10

Train loss 0.1686425145880735 accuracy 0.9576025025136856

Val loss 0.28851429480769836 accuracy 0.9407953529937444

Epoch 3/10

Train loss 0.1337419254127866 accuracy 0.9671545078762149

Val loss 0.30782870739097107 accuracy 0.9412421805183199

Epoch 4/10

Train loss 0.10582935492722276 accuracy 0.9760361970729527

Val loss 0.3323856392374507 accuracy 0.9396782841823056

```
Epoch 5/10
-----
Train loss 0.07139312713640018 accuracy 0.9855602725952407
Val   loss 0.35490698625215245 accuracy 0.9423592493297587

Epoch 6/10
-----
Train loss 0.046193289646322744 accuracy 0.9914534688861579
Val   loss 0.4471171006476457 accuracy 0.9367739052725648

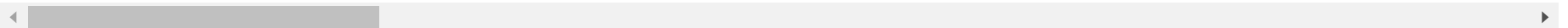
Epoch 7/10
-----
Train loss 0.032726022069391264 accuracy 0.9938833649871522
Val   loss 0.45923905965045314 accuracy 0.943029490616622

Epoch 8/10
-----
Train loss 0.021351659609271035 accuracy 0.9958943134845268
Val   loss 0.5351211025175286 accuracy 0.9392314566577301

Epoch 9/10
-----
Train loss 0.016150615409964476 accuracy 0.997067366774662
Val   loss 0.5383798147514637 accuracy 0.9410187667560321

Epoch 10/10
-----
Train loss 0.012994375374114924 accuracy 0.9975701038990057
Val   loss 0.5386938269140972 accuracy 0.9407953529937444

CPU times: user 2h 6min 59s, sys: 52min 15s, total: 2h 59min 14s
Wall time: 2h 59min 55s
```



Note that we're storing the state of the best model, indicated by the highest validation accuracy.

```
# plt.plot(history['train_acc'], label='train accuracy')
# plt.plot(history['val_acc'], label='validation accuracy')
```

```
# plt.title('Training history')
# plt.ylabel('Accuracy')
# plt.xlabel('Epoch')
# plt.legend()
# plt.ylim([0, 1]);
```

▼ Evaluation

So how good is our model on predicting sentiment? Let's start by calculating the accuracy on the test data:

```
test_acc, _ = eval_model(
    model,
    test_data_loader,
    loss_fn,
    device,
    len(df_test)
)

test_acc.item()
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length`
FutureWarning,
0.9419124218051832
```

The accuracy is about 1% lower on the test set. Our model seems to generalize well.

We'll define a helper function to get the predictions from our model:

```
def get_predictions(model, data_loader):
    model = model.eval()

    review_texts = []
    predictions = []
    prediction_probs = []
```

```

real_values = []

with torch.no_grad():
    for d in data_loader:

        texts = d["review_text"]
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        targets = d["targets"].to(device)

        outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask
        )
        _, preds = torch.max(outputs, dim=1)

        probs = F.softmax(outputs, dim=1)

        review_texts.extend(texts)
        predictions.extend(preds)
        prediction_probs.extend(probs)
        real_values.extend(targets)

predictions = torch.stack(predictions).cpu()
prediction_probs = torch.stack(prediction_probs).cpu()
real_values = torch.stack(real_values).cpu()
return review_texts, predictions, prediction_probs, real_values

```

This is similar to the evaluation function, except that we're storing the text of the reviews and the predicted probabilities (by applying the softmax on the model outputs):

```

y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
    model,
    test_data_loader
)

```



```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version. Please use the `padding` argument instead.
FutureWarning,
```

Let's have a look at the classification report

```
print(classification_report(y_test, y_pred, target_names=class_names))
```

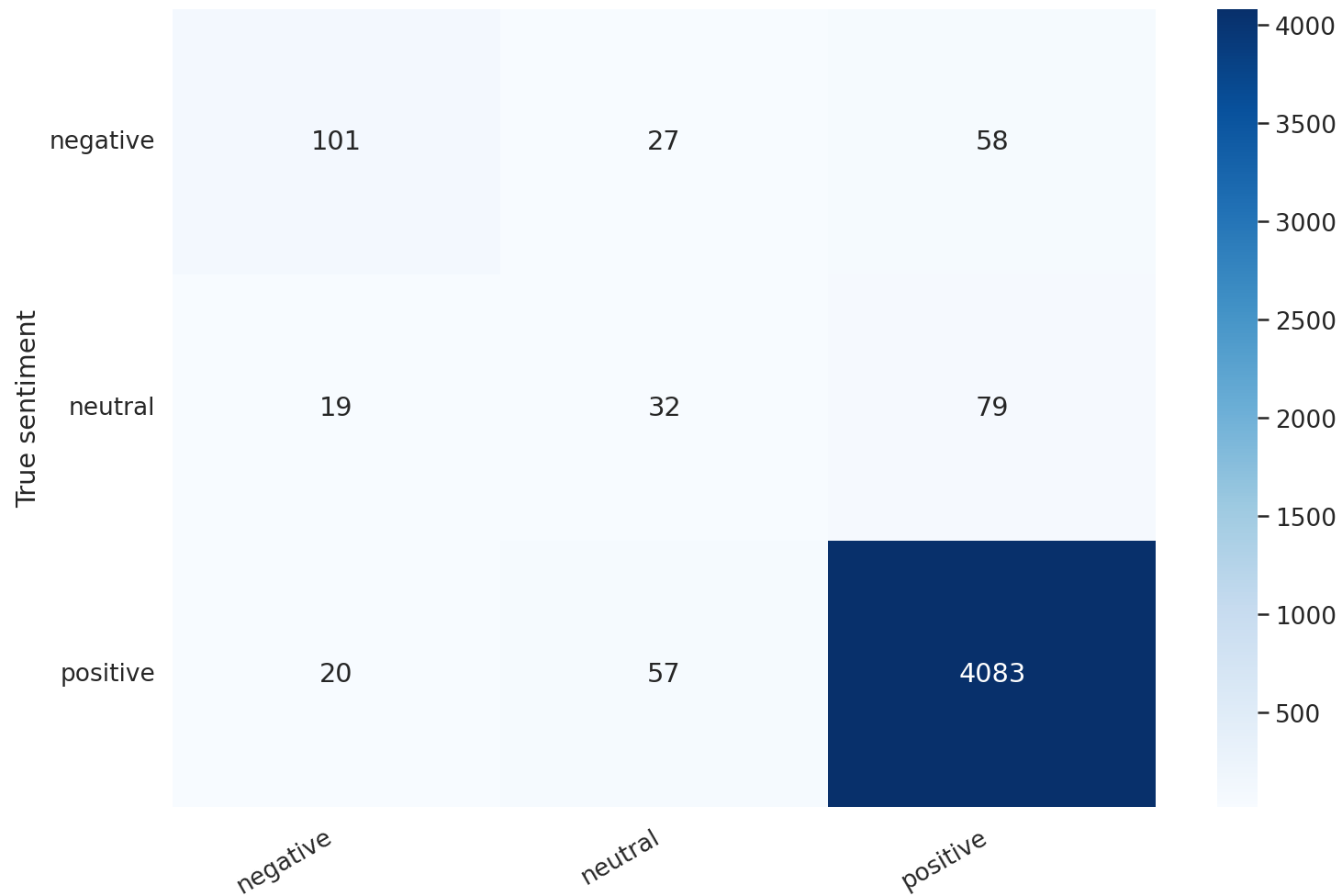
	precision	recall	f1-score	support
negative	0.72	0.54	0.62	186
neutral	0.28	0.25	0.26	130
positive	0.97	0.98	0.97	4160
accuracy			0.94	4476
macro avg	0.65	0.59	0.62	4476
weighted avg	0.94	0.94	0.94	4476

Looks like it is really hard to classify neutral (3 stars) reviews. And I can tell you from experience, looking at many reviews, those are hard to classify.

We'll continue with the confusion matrix:

```
def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0, ha='right')
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30, ha='right')
    plt.ylabel('True sentiment')
    plt.xlabel('Predicted sentiment');

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```



This confirms that our model is having difficulty classifying neutral reviews. It mistakes those for negative and positive at a roughly equal frequency.

That's a good overview of the performance of our model. But let's have a look at an example from our test data:

```
idx = 2

review_text = y_review_texts[idx]
true_sentiment = y_test[idx]
pred_df = pd.DataFrame({
```

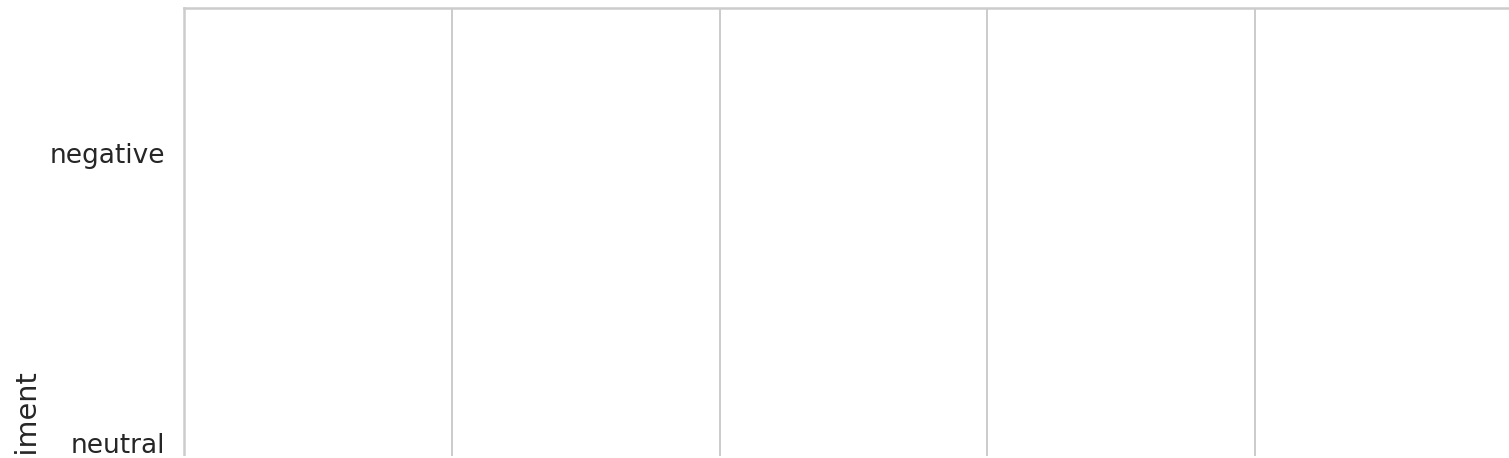
```
'class_names': class_names,  
'values': y_pred_probs[idx]  
)  
  
print("\n".join(wrap(review_text)))  
print()  
print(f'True sentiment: {class_names[true_sentiment]}')
```

I was looking for a camera that would work for taking pictures of my kids, pets, and would also work for macro shots of knitting projects. I did some research, and found that this one had all the features I needed. I also was specifically looking for a Canon camera, as I recently purchased a Canon printer and wanted to stick within the brand. Read full review...

True sentiment: positive

Now we can look at the confidence of each sentiment of our model:

```
sns.barplot(x='values', y='class_names', data=pred_df, orient='h')  
plt.ylabel('sentiment')  
plt.xlabel('probability')  
plt.xlim([0, 1]);
```



► Predicting on Raw Text

Let's use our model to predict the sentiment of some raw text:

[] ↳ 6 cells hidden

