BERT (No Augmentation)

Real News and Fake News (~60k total)

Class: Label

Real: 1

Fake: 0

```
import tensorflow as tf

# Get the GPU device name.
device_name = tf.test.gpu_device_name()

# The device name should look like the following:
if device_name == '/device:GPU:0':
    print('Found GPU at: {}'.format(device_name))
else:
    raise SystemError('GPU device not found')
```

```
Found GPU at: /device:GPU:0
```

```
import torch

# If there's a GPU available...
if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")

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

    print('We will use the GPU:', torch.cuda.get_device_name(0))

# If not...
```

```
# if not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
We will use the GPU: Tesla P100-PCIE-16GB
```

```
!pip install transformers
```

```
Collecting transformers
  Downloading https://files.pythonhosted.org/packages/a3/78/92cedda05552398352ed9784908b834ee32a0bd071a9b32de
     |████████████████████████████████| 573kB 2.8MB/s
Requirement already satisfied: dataclasses; python_version < "3.7" in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from transformers) (2.21.0
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.6/dist-packages (from transformers
Requirement already satisfied: filelock in /usr/local/lib/python3.6/dist-packages (from transformers) (3.0.12
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from transformers) (1.18.2)
Requirement already satisfied: boto3 in /usr/local/lib/python3.6/dist-packages (from transformers) (1.12.40)
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/74/f4/2d5214cbf13d06e7cb2c20d84115ca25b53ea76fa1f0ade0e
     |████████████████████████████████| 1.0MB 44.0MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.6/dist-packages (from transformers) (4.38
Collecting sacremoses
  Downloading https://files.pythonhosted.org/packages/99/50/93509f906a40bffd7d175f97fd75ea328ad9bd91f48f59c4b
     |████████████████████████████████| 890kB 42.5MB/s
Collecting tokenizers==0.5.2
  Downloading https://files.pythonhosted.org/packages/d1/3f/73c881ea4723e43c1e9acf317cf407fab3a278daab3a69c98
     |████████████████████████████████| 3.7MB 32.5MB/s
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->trans
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->t
Requirement already satisfied: botocore<1.16.0,>=1.15.40 in /usr/local/lib/python3.6/dist-packages (from boto
Requirement already satisfied: s3transfer<0.4.0,>=0.3.0 in /usr/local/lib/python3.6/dist-packages (from boto3
Requirement already satisfied: jmespath<1.0.0,>=0.7.1 in /usr/local/lib/python3.6/dist-packages (from boto3->
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers)
Requirement already satisfied: click in /usr/local/lib/python3.6/dist-packages (from sacremoses->transformers
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from sacremoses->transforme
Requirement already satisfied: docutils<0.16,>=0.10 in /usr/local/lib/python3.6/dist-packages (from botocore
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in /usr/local/lib/python3.6/dist-packages (from bo
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.41-cp36-none-any.whl size=893334 sha256=03622d07f650
  Stored in directory: /root/.cache/pip/wheels/22/5a/d4/b020a81249de7dc63758a34222feaa668dbe8ebfe9170cc9b1
Successfully built sacremoses
Installing collected packages: sentencepiece, sacremoses, tokenizers, transformers
Successfully installed sacremoses-0.0.41 sentencepiece-0.1.85 tokenizers-0.5.2 transformers-2.8.0
```

No Augmentation

```
import pandas as pd
import numpy as np

import sklearn
from sklearn.model_selection import train_test_split

file = "combined_{}.csv"
dfs = []
for i in range(3):
    fp = file.format(i+1)
    read = pd.read_csv(fp)
    read = read[['label', 'clean_text']]
    dfs.append(read)

dfs[2] = dfs[2][:-13000]

data = pd.concat(dfs)
data.tail()
data.reset_index(inplace=True, drop=True)
print('All Data:', data.shape)

data.dropna(inplace=True)
train_data, test_data = train_test_split(data, test_size=0.2)

print('\nTrain Data:', train_data.shape)
print(train_data[train_data.label == 1].shape[0], "Real")
print(train_data[train_data.label == 0].shape[0], "Fake")

print('\nTest Data:', test_data.shape)
print(test_data[test_data.label == 1].shape[0], "Real")
print(test_data[test_data.label == 0].shape[0], "Fake")

sentences = train_data.clean_text.values
labels = train_data.label.values

train_data.head(10)
```

```
All Data: (59818, 2)

Train Data: (47567, 2)
25119 Real
22448 Fake

Test Data: (11892, 2)
6330 Real
5562 Fake
```

	label	clean_text
15323	0	shares prince abdullah alsaud saudi arabia...
23487	0	best mix hardhitting real news cuttingedge al...
30416	0	miss russia afpeast news miss russia alisa man...
43407	0	rick santorum says rick perry requested earma...
33772	0	inside bill clinton inc hacked memo shows inte...
45096	0	obamas justicedesignate sotomayor threw new fi...
7991	1	paris afp marine le pens aversion european uni...
17254	0	via truthandaction sponsored links location to...
43768	0	says cathy jordan arrested dragged home swat t...
3396	0	leave reply diane canfield biggest step evolut...

```
from transformers import BertTokenizer

# Load the BERT tokenizer.
print('Loading BERT tokenizer...')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)

# Print the original sentence.
print(' Original: ', sentences[0])

# Print the sentence split into tokens.
```

```
print('Tokenized: ', tokenizer.tokenize(sentences[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences[0])))
```

```
Loading BERT tokenizer...
Downloading: 100%                    232k/232k [00:00<00:00, 2.62MB/s]

 Original:  shares     prince abdullah alsaud saudi arabias ambassador united states confronted reporter inte
Tokenized:  ['shares', 'prince', 'abdullah', 'als', '##aud', 'saudi', 'arabia', '##s', 'ambassador', 'united
Token IDs:  [6661, 3159, 14093, 25520, 19513, 8174, 9264, 2015, 6059, 2142, 2163, 12892, 6398, 19115, 3613, 2
```

```
# Tokenize all of the sentences and map the tokens to thier word IDs.
input_ids = []

# For every sentence...
for sent in sentences:
    # `encode` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    encoded_sent = tokenizer.encode(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 512  # Truncate all sentences.
                        #return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_sent)

# Print sentence 0, now as a list of IDs.
print('Original: ', sentences[0])
print('Token IDs:', input_ids[0])
```

```
Original:  shares     prince abdullah alsaud saudi arabias ambassador united states confronted reporter inte
Token IDs: [101, 6661, 3159, 14093, 25520, 19513, 8174, 9264, 2015, 6059, 2142, 2163, 12892, 6398, 19115, 36
```

```
import statistics

print('Avg sentence length: ', statistics.mean([len(sen) for sen in input_ids]))
```

```
Avg sentence length:  255.94416297012634
```

```
print('Max sentence length: ', max([len(sen) for sen in input_ids]))
```

```
Max sentence length:  512
```

```
import keras
# We'll borrow the `pad_sequences` utility function to do this.
from keras.preprocessing.sequence import pad_sequences

# Set the maximum sequence length.
# I've chosen 64 somewhat arbitrarily. It's slightly larger than the
# maximum training sentence length of 47...
MAX_LEN = 128

print('\nPadding/truncating all sentences to %d values...' % MAX_LEN)

print('\nPadding token: "{:}", ID: {:}'.format(tokenizer.pad_token, tokenizer.pad_token_id))

# Pad our input tokens with value 0.
# "post" indicates that we want to pad and truncate at the end of the sequence,
# as opposed to the beginning.
input_ids = pad_sequences(input_ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="post")

print('\nDone.')
```

```
Padding/truncating all sentences to 128 values...

Padding token: "[PAD]", ID: 0
Using TensorFlow backend.

Done.
```

```
# Create attention masks
attention_masks = []

# For each sentence...
for sent in input_ids:

    # Create the attention mask.
    #   - If a token ID is 0, then it's padding, set the mask to 0.
    #   - If a token ID is > 0, then it's a real token, set the mask to 1.
    att_mask = [int(token_id > 0) for token_id in sent]

    # Store the attention mask for this sentence.
    attention_masks.append(att_mask)
```

```
# Use 90% for training and 10% for validation.
train_inputs, validation_inputs, train_labels, validation_labels = train_test_split(input_ic
                                                            random_state=2018, test_size=0.:
# Do the same for the masks.
train_masks, validation_masks, _, _ = train_test_split(attention_masks, labels,
                                             random_state=2018, test_size=0.1)
```

```
# Convert all inputs and labels into torch tensors, the required datatype
# for our model.
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

# The DataLoader needs to know our batch size for training, so we specify it
# here.
# For fine-tuning BERT on a specific task, the authors recommend a batch size of
# 16 or 32.

batch_size = 32

# Create the DataLoader for our training set.
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)

# Create the DataLoader for our validation set.
validation_data = TensorDataset(validation_inputs, validation_masks, validation_labels)
validation_sampler = SequentialSampler(validation_data)
validation_dataloader = DataLoader(validation_data, sampler=validation_sampler, batch_size=b


from transformers import BertForSequenceClassification, AdamW, BertConfig

# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 2, # The number of output labels--2 for binary classification.
                    # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()
```

Downloading: 100% 361/361 [00:21<00:00, 16.4B/s]

Downloading: 100% 440M/440M [00:08<00:00, 49.9MB/s]

```
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (1): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
```

```
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (2): BertLayer(
    (attention): BertAttention(
      (self): BertSelfAttention(
        (query): Linear(in_features=768, out_features=768, bias=True)
        (key): Linear(in_features=768, out_features=768, bias=True)
        (value): Linear(in_features=768, out_features=768, bias=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (output): BertSelfOutput(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (intermediate): BertIntermediate(
      (dense): Linear(in_features=768, out_features=3072, bias=True)
    )
    (output): BertOutput(
      (dense): Linear(in_features=3072, out_features=768, bias=True)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
  )
  (3): BertLayer(
```

```
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (4): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
```

```
      ,
    )
    (5): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
    )
    (6): BertLayer(
      (attention): BertAttention(
        (self): BertSelfAttention(
          (query): Linear(in_features=768, out_features=768, bias=True)
          (key): Linear(in_features=768, out_features=768, bias=True)
          (value): Linear(in_features=768, out_features=768, bias=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
        (output): BertSelfOutput(
          (dense): Linear(in_features=768, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (intermediate): BertIntermediate(
        (dense): Linear(in_features=768, out_features=3072, bias=True)
      )
      (output): BertOutput(
        (dense): Linear(in_features=3072, out_features=768, bias=True)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
```

```
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (7): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
        (output): BertOutput(
          (dense): Linear(in_features=3072, out_features=768, bias=True)
          (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
      (8): BertLayer(
        (attention): BertAttention(
          (self): BertSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
          (dense): Linear(in_features=768, out_features=3072, bias=True)
        )
```

```
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (9): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (10): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
```

```
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

```
# Note: AdamW is a class from the huggingface library (as opposed to pytorch)
```

```
# Note: AdamW is a class from the huggingface library (as opposed to pytorch)
# I believe the 'W' stands for 'Weight Decay fix"
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate - default is 5e-5, our notebook had 2e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                )



from transformers import get_linear_schedule_with_warmup

# Number of training epochs (authors recommend between 2 and 4)
epochs = 5

# Total number of training steps is number of batches * number of epochs.
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default value in run_glu
                                            num_training_steps = total_steps)


# Function to calculate the accuracy of our predictions vs labels
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):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # Round to the nearest second.
    elapsed_rounded = int(round((elapsed)))

    # Format as hh:mm:ss
```

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


    torch.cuda.empty_cache()


    import random

    # This training code is based on the `run_glue.py` script here:
    # https://github.com/huggingface/transformers/blob/5bfcd0485ece086ebcbed2d008813037968a9e58/


    # Set the seed value all over the place to make this reproducible.
    seed_val = 42

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

    # Store the average loss after each epoch so we can plot them.
    loss_values = []

    # For each epoch...
    for epoch_i in range(0, epochs):

        # ========================================
        #               Training
        # ========================================

        # Perform one full pass over the training set.

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

        # Measure how long the training epoch takes.
        t0 = time.time()

        # Reset the total loss for this epoch.
```

```
    total_loss = 0

    # Put the model into training mode. Don't be mislead--the call to
    # `train` just changes the *mode*, it doesn't *perform* the training.
    # `dropout` and `batchnorm` layers behave differently during training
    # vs. test (source: https://stackoverflow.com/questions/51433378/what-does-model-train-c
    model.train()

    # For each batch of training data...
    for step, batch in enumerate(train_dataloader):

        # Progress update every 40 batches.
        if step % 40 == 0 and not step == 0:
            # Calculate elapsed time in minutes.
            elapsed = format_time(time.time() - t0)

            # Report progress.
            print('  Batch {:>5,}  of  {:>5,}.    Elapsed: {:}.'.format(step, len(train_data

        # Unpack this training batch from our dataloader.
        #
        # As we unpack the batch, we'll also copy each tensor to the GPU using the
        # `to` method.
        #
        # `batch` contains three pytorch tensors:
        #   [0]: input ids
        #   [1]: attention masks
        #   [2]: labels
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        # Always clear any previously calculated gradients before performing a
        # backward pass. PyTorch doesn't do this automatically because
        # accumulating the gradients is "convenient while training RNNs".
        # (source: https://stackoverflow.com/questions/48001598/why-do-we-need-to-call-zero-
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
```

```
        # This will return the loss (rather than the model output) because we
        # have provided the `labels`.
        # The documentation for this `model` function is here:
        # https://huggingface.co/transformers/v2.2.0/model_doc/bert.html#transformers.BertFo
        outputs = model(b_input_ids,
                    token_type_ids=None,
                    attention_mask=b_input_mask,
                    labels=b_labels)

        # The call to `model` always returns a tuple, so we need to pull the
        # loss value out of the tuple.
        loss = outputs[0]

        # Accumulate the training loss over all of the batches so that we can
        # calculate the average loss at the end. `loss` is a Tensor containing a
        # single value; the `.item()` function just returns the Python value
        # from the tensor.
        total_loss += loss.item()

        # Perform a backward pass to calculate the gradients.
        loss.backward()

        # Clip the norm of the gradients to 1.0.
        # This is to help prevent the "exploding gradients" problem.
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        # Update parameters and take a step using the computed gradient.
        # The optimizer dictates the "update rule"--how the parameters are
        # modified based on their gradients, the learning rate, etc.
        optimizer.step()

        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over the training data.
    avg_train_loss = total_loss / len(train_dataloader)

    # Store the loss value for plotting the learning curve.
    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)))

    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set.

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

    t0 = time.time()

    # Put the model in evaluation mode--the dropout layers behave differently
    # during evaluation.
    model.eval()

    # Tracking variables
    eval_loss, eval_accuracy = 0, 0
    nb_eval_steps, nb_eval_examples = 0, 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:

        # 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

        # Telling the model not to compute or store gradients, saving memory and
        # speeding up validation
        with torch.no_grad():

            # Forward pass, calculate logit predictions.
            # This will return the logits rather than the loss because we have
            # not provided labels.
```

```
            # not provided labels.
            # token_type_ids is the same as the "segment ids", which
            # differentiates sentence 1 and 2 in 2-sentence tasks.
            # The documentation for this `model` function is here:
            # https://huggingface.co/transformers/v2.2.0/model_doc/bert.html#transformers.Be
            outputs = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask)

        # Get the "logits" output by the model. The "logits" are the output
        # values prior to applying an activation function like the softmax.
        logits = outputs[0]

        # Move logits and labels to CPU
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()

        # Calculate the accuracy for this batch of test sentences.
        tmp_eval_accuracy = flat_accuracy(logits, label_ids)

        # Accumulate the total accuracy.
        eval_accuracy += tmp_eval_accuracy

        # Track the number of batches
        nb_eval_steps += 1

    # Report the final accuracy for this validation run.
    print("  Accuracy: {0:.3f}".format(eval_accuracy/nb_eval_steps))
    print("  Validation took: {:}".format(format_time(time.time() - t0)))

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

```
======== Epoch 1 / 5 ========
Training...
  Batch    40  of  1,338.    Elapsed: 0:00:16.
  Batch    80  of  1,338.    Elapsed: 0:00:33.
  Batch   120  of  1,338.    Elapsed: 0:00:49.
  Batch   160  of  1,338.    Elapsed: 0:01:05.
  Batch   200  of  1,338.    Elapsed: 0:01:21.
  Batch   240  of  1,338.    Elapsed: 0:01:37.
  Batch   280  of  1,338.    Elapsed: 0:01:53.
  Batch   320  of  1,338.    Elapsed: 0:02:09.
  Batch   360  of  1,338.    Elapsed: 0:02:25.
  Batch   400  of  1,338.    Elapsed: 0:02:41.
  Batch   440  of  1,338.    Elapsed: 0:02:57.
  Batch   480  of  1,338.    Elapsed: 0:03:14.
  Batch   520  of  1,338.    Elapsed: 0:03:30.
  Batch   560  of  1,338.    Elapsed: 0:03:46.
  Batch   600  of  1,338.    Elapsed: 0:04:02.
  Batch   640  of  1,338.    Elapsed: 0:04:18.
  Batch   680  of  1,338.    Elapsed: 0:04:34.
  Batch   720  of  1,338.    Elapsed: 0:04:50.
  Batch   760  of  1,338.    Elapsed: 0:05:06.
  Batch   800  of  1,338.    Elapsed: 0:05:22.
  Batch   840  of  1,338.    Elapsed: 0:05:38.
  Batch   880  of  1,338.    Elapsed: 0:05:54.
  Batch   920  of  1,338.    Elapsed: 0:06:10.
  Batch   960  of  1,338.    Elapsed: 0:06:27.
  Batch 1,000  of  1,338.    Elapsed: 0:06:43.
  Batch 1,040  of  1,338.    Elapsed: 0:06:59.
  Batch 1,080  of  1,338.    Elapsed: 0:07:15.
  Batch 1,120  of  1,338.    Elapsed: 0:07:31.
  Batch 1,160  of  1,338.    Elapsed: 0:07:47.
  Batch 1,200  of  1,338.    Elapsed: 0:08:03.
  Batch 1,240  of  1,338.    Elapsed: 0:08:19.
  Batch 1,280  of  1,338.    Elapsed: 0:08:35.
  Batch 1,320  of  1,338.    Elapsed: 0:08:52.

  Average training loss: 0.31
  Training epcoh took: 0:08:59

Running Validation...
  Accuracy: 0.883
  Validation took: 0:00:18
```

```
======== Epoch 2 / 5 ========
Training...
  Batch    40  of  1,338.    Elapsed: 0:00:16.
  Batch    80  of  1,338.    Elapsed: 0:00:32.
  Batch   120  of  1,338.    Elapsed: 0:00:48.
  Batch   160  of  1,338.    Elapsed: 0:01:04.
  Batch   200  of  1,338.    Elapsed: 0:01:20.
  Batch   240  of  1,338.    Elapsed: 0:01:36.
  Batch   280  of  1,338.    Elapsed: 0:01:52.
  Batch   320  of  1,338.    Elapsed: 0:02:08.
  Batch   360  of  1,338.    Elapsed: 0:02:25.
  Batch   400  of  1,338.    Elapsed: 0:02:41.
  Batch   440  of  1,338.    Elapsed: 0:02:57.
  Batch   480  of  1,338.    Elapsed: 0:03:13.
  Batch   520  of  1,338.    Elapsed: 0:03:29.
  Batch   560  of  1,338.    Elapsed: 0:03:45.
  Batch   600  of  1,338.    Elapsed: 0:04:01.
  Batch   640  of  1,338.    Elapsed: 0:04:17.
  Batch   680  of  1,338.    Elapsed: 0:04:34.
  Batch   720  of  1,338.    Elapsed: 0:04:50.
  Batch   760  of  1,338.    Elapsed: 0:05:06.
  Batch   800  of  1,338.    Elapsed: 0:05:22.
  Batch   840  of  1,338.    Elapsed: 0:05:38.
  Batch   880  of  1,338.    Elapsed: 0:05:54.
  Batch   920  of  1,338.    Elapsed: 0:06:10.
  Batch   960  of  1,338.    Elapsed: 0:06:26.
  Batch 1,000  of  1,338.    Elapsed: 0:06:43.
  Batch 1,040  of  1,338.    Elapsed: 0:06:59.
  Batch 1,080  of  1,338.    Elapsed: 0:07:15.
  Batch 1,120  of  1,338.    Elapsed: 0:07:31.
  Batch 1,160  of  1,338.    Elapsed: 0:07:47.
  Batch 1,200  of  1,338.    Elapsed: 0:08:03.
  Batch 1,240  of  1,338.    Elapsed: 0:08:19.
  Batch 1,280  of  1,338.    Elapsed: 0:08:35.
  Batch 1,320  of  1,338.    Elapsed: 0:08:52.

  Average training loss: 0.19
  Training epcoh took: 0:08:59

Running Validation...
  Accuracy: 0.890
  Validation took: 0:00:18
```