CS 505 Homework 06: Transformers

Problem Two

See the problem one notebook for details on due date, submission, etc.

As with problem one, there is an extensive tutorial section, followed by some tasks at the end you need to complete.

Full Disclosure: This is based on a BERT Fine-Tuning Tutorial with PyTorch, by Chris McCormick and Nick Ryan

⌄ Introduction

Here is the original introduction to this tutorial material:

In this tutorial I'll show you how to use BERT with the huggingface PyTorch library to quickly and efficiently fine-tune a model to get near state of the art performance in sentence classification. More broadly, I describe the practical application of transfer learning in NLP to create high performance models with minimal effort on a range of NLP tasks.

This post is presented in two forms--as a blog post here and as a Colab Notebook here.

The content is identical in both, but:

- The blog post includes a comments section for discussion.
- The Colab Notebook will allow you to run the code and inspect it as you read through.

I've also published a video walkthrough of this post on my YouTube channel! Part 1 and Part 2.

History

2018 was a breakthrough year in NLP. Transfer learning, particularly models like Allen AI's ELMO, OpenAI's Open-GPT, and Google's BERT allowed researchers to smash multiple benchmarks with minimal task-specific fine-tuning and provided the rest of the NLP community with pretrained models that could easily (with less data and less compute time) be fine-tuned and implemented to produce state of the art results. Unfortunately, for many starting out in NLP and even for some experienced practicioners, the theory and practical application of these powerful models is still not well understood.

What is BERT?

BERT (Bidirectional Encoder Representations from Transformers), released in late 2018, is the model we will use in this tutorial to provide readers with a better understanding of and practical guidance for using transfer learning models in NLP. BERT is a method of pretraining language representations that was used to create models that NLP practicioners can then download and use for free. You can either use these models to extract high quality language features from your text data, or you can fine-tune these models on a specific task (classification, entity recognition, question answering, etc.) with your own data to produce state of the art predictions.

This post will explain how you can modify and fine-tune BERT to create a powerful NLP model that quickly gives you state of the art results.

⌄ Advantages of Fine-Tuning

In this tutorial, we will use BERT to train a text classifier. Specifically, we will take the pre-trained BERT model, add an untrained layer of neurons on the end, and train the new model for our classification task. Why do this rather than train a train a specific deep learning model (a CNN, BiLSTM, etc.) that is well suited for the specific NLP task you need?

1. **Quicker Development**

   - First, the pre-trained BERT model weights already encode a lot of information about our language. As a result, it takes much less time to train our fine-tuned model - it is as if we have already trained the bottom layers of our network extensively and only need to gently tune them while using their output as features for our classification task. In fact, the

authors recommend only 2-4 epochs of training for fine-tuning BERT on a specific NLP task (compared to the hundreds of GPU hours needed to train the original BERT model or a LSTM from scratch!).

2. **Less Data**

   - In addition and perhaps just as important, because of the pre-trained weights this method allows us to fine-tune our task on a much smaller dataset than would be required in a model that is built from scratch. A major drawback of NLP models built from scratch is that we often need a prohibitively large dataset in order to train our network to reasonable accuracy, meaning a lot of time and energy had to be put into dataset creation. By fine-tuning BERT, we are now able to get away with training a model to good performance on a much smaller amount of training data.

3. **Better Results**

   - Finally, this simple fine-tuning procedure (typically adding one fully-connected layer on top of BERT and training for a few epochs) was shown to achieve state of the art results with minimal task-specific adjustments for a wide variety of tasks: classification, language inference, semantic similarity, question answering, etc. Rather than implementing custom and sometimes-obscure architetures shown to work well on a specific task, simply fine-tuning BERT is shown to be a better (or at least equal) alternative.

A Shift in NLP

This shift to transfer learning parallels the same shift that took place in computer vision a few years ago. Creating a good deep learning network for computer vision tasks can take millions of parameters and be very expensive to train. Researchers discovered that deep networks learn hierarchical feature representations (simple features like edges at the lowest layers with gradually more complex features at higher layers). Rather than training a new network from scratch each time, the lower layers of a trained network with generalized image features could be copied and transfered for use in another network with a different task. It soon became common practice to download a pre-trained deep network and quickly retrain it for the new task or add additional layers on top - vastly preferable to the expensive process of training a network from scratch. For many, the introduction of deep pre-trained language models in 2018 (ELMO, BERT, ULMFIT, Open-GPT, etc.) signals the same shift to transfer learning in NLP that computer vision saw.

Let's get started!

1. Setup

1.1. Using Colab GPU for Training

Google Colab offers free GPUs and TPUs! Since we'll be training a large neural network it's best to take advantage of this (in this case we'll attach a GPU), otherwise training will take a very long time.

A GPU can be added by going to the menu and selecting:

Edit 🔲 Notebook Settings 🔲 Hardware accelerator 🔲 (GPU)

Then run the following cell to confirm that the GPU is detected.

```
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
```

In order for torch to use the GPU, we need to identify and specify the GPU as the device. Later, in our training loop, we will load data onto the device.

```
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...
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 T4
```

∨ 1.2. Installing the Hugging Face Library

Next, let's install the [transformers](#) package from Hugging Face which will give us a pytorch interface for working with BERT. (This library contains interfaces for other pretrained language models like OpenAI's GPT and GPT-2.) We've selected the pytorch interface because it strikes a nice balance between the high-level APIs (which are easy to use but don't provide insight into how things work) and tensorflow code (which contains lots of details but often sidetracks us into lessons about tensorflow, when the purpose here is BERT!).

At the moment, the Hugging Face library seems to be the most widely accepted and powerful pytorch interface for working with BERT. In addition to supporting a variety of different pre-trained transformer models, the library also includes pre-built modifications of these models suited to your specific task. For example, in this tutorial we will use `BertForSequenceClassification`.

The library also includes task-specific classes for token classification, question answering, next sentence prediciton, etc. Using these pre-built classes simplifies the process of modifying BERT for your purposes.

```
!pip install transformers

    Collecting transformers
      Downloading transformers-4.18.0-py3-none-any.whl (4.0 MB)
          |████████████████████████████████| 4.0 MB 35.6 MB/s
    Collecting pyyaml>=5.1
      Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux20
```

```
|████████████████████████████| 596 kB 62.9 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.6.0
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transforme
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformer
Collecting huggingface-hub<1.0,>=0.1.0
  Downloading huggingface_hub-0.5.1-py3-none-any.whl (77 kB)
|████████████████████████████| 77 kB 3.3 MB/s
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.6
Collecting sacremoses
  Downloading sacremoses-0.0.53.tar.gz (880 kB)
|████████████████████████████| 880 kB 64.1 MB/s
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 54.8 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from hu
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from pack
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata-
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transf
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->t
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformers)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages (from sacremoses->transformer
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (from sacremoses->transforme
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.53-py3-none-any.whl size=895260 sha256=c2686db242785
  Stored in directory: /root/.cache/pip/wheels/87/39/dd/a83eeef36d0bf98e7a4d1933a4ad2d660295a40613079bafc9
Successfully built sacremoses
Installing collected packages: pyyaml, tokenizers, sacremoses, 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-0.5.1 pyyaml-6.0 sacremoses-0.0.53 tokenizers-0.12.1 transformers-4.1
```

The code in this notebook is actually a simplified version of the run_glue.py example script from huggingface.

run_glue.py is a helpful utility which allows you to pick which GLUE benchmark task you want to run on, and which pre-trained model you want to use (you can see the list of possible models here). It also supports using either the CPU, a single GPU, or multiple GPUs. It even supports using 16-bit precision if you want further speed up.

Unfortunately, all of this configurability comes at the cost of *readability*. In this Notebook, we've simplified the code greatly and added plenty of comments to make it clear what's going on.

2. Loading CoLA Dataset

We'll use The Corpus of Linguistic Acceptability (CoLA) dataset for single sentence classification. It's a set of sentences labeled as grammatically correct or incorrect. It was first published in May of 2018, and is one of the tests included in the "GLUE Benchmark" on which models like BERT are competing.

2.1. Download & Extract

We'll use the wget package to download the dataset to the Colab instance's file system.

```
!pip install wget
```

```
Collecting wget
  Downloading wget-3.2.zip (10 kB)
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-py3-none-any.whl size=9675 sha256=6591c13c831f8b1da84f0d2d60aae5
  Stored in directory: /root/.cache/pip/wheels/a1/b6/7c/0e63e34eb06634181c63adacca38b79ff8f35c37e3c13e3c02
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

The dataset is hosted on GitHub in this repo: https://nyu-mll.github.io/CoLA/

```
import wget
import os

print('Downloading dataset...')

# The URL for the dataset zip file.
url = 'https://nyu-mll.github.io/CoLA/cola_public_1.1.zip'

# Download the file (if we haven't already)
if not os.path.exists('./cola_public_1.1.zip'):
    wget.download(url, './cola_public_1.1.zip')

    Downloading dataset...
```

Unzip the dataset to the file system. You can browse the file system of the Colab instance in the sidebar on the left.

```
# Unzip the dataset (if we haven't already)
if not os.path.exists('./cola_public/'):
    !unzip cola_public_1.1.zip

    Archive:  cola_public_1.1.zip
       creating: cola_public/
      inflating: cola_public/README
       creating: cola_public/tokenized/
      inflating: cola_public/tokenized/in_domain_dev.tsv
      inflating: cola_public/tokenized/in_domain_train.tsv
      inflating: cola_public/tokenized/out_of_domain_dev.tsv
       creating: cola_public/raw/
      inflating: cola_public/raw/in_domain_dev.tsv
      inflating: cola_public/raw/in_domain_train.tsv
      inflating: cola_public/raw/out_of_domain_dev.tsv
```

⌄ 2.2. Parse

We can see from the file names that both `tokenized` and `raw` versions of the data are available.

We can't use the pre-tokenized version because, in order to apply the pre-trained BERT, we *must* use the tokenizer provided by the model. This is because (1) the model has a specific, fixed vocabulary and (2) the BERT tokenizer has a particular way of handling out-of-vocabulary words.

We'll use pandas to parse the "in-domain" training set and look at a few of its properties and data points.

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("./cola_public/raw/in_domain_train.tsv", delimiter='\t', header=None, names=['sentence_source',

# Report the number of sentences.
print('Number of training sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

```
Number of training sentences: 8,551
```

	sentence_source	label	label_notes	sentence
6213	c_13	1	NaN	Lucy seems to have been mugged.
3788	ks08	1	NaN	John bought a book for the student.
4863	ks08	0	*	He mended.

The two properties we actually care about are the the `sentence` and its `label`, which is referred to as the "acceptibility judgment" (0=unacceptable, 1=acceptable).

Here are five sentences which are labeled as not grammatically acceptable. Note how much more difficult this task is than something like sentiment analysis!

```
df.loc[df.label == 0].sample(5)[['sentence', 'label']]
```

	sentence	label
4174	One people was dying of thirst.	0
3241	Tony broke herself on the ann.	0
6612	The Ethel was sitting at her desk.	0
5385	She has enough a problem as it is.	0
6129	It seems Sonny to love Cher.	0

Let's extract the sentences and labels of our training set as numpy ndarrays.

```
# Get the lists of sentences and their labels.
sentences = df.sentence.values
labels = df.label.values
```

3. Tokenization & Input Formatting

In this section, we'll transform our dataset into the format that BERT can be trained on.

3.1. BERT Tokenizer

To feed our text to BERT, it must be split into tokens, and then these tokens must be mapped to their index in the tokenizer vocabulary.

The tokenization must be performed by the tokenizer included with BERT--the below cell will download this for us. We'll be using the "uncased" version here.

```
from transformers import BertTokenizer

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

```
    Loading BERT tokenizer...
```

Downloading: 100%	226k/226k [00:00<00:00, 5.87MB/s]
Downloading: 100%	28.0/28.0 [00:00<00:00, 874B/s]
Downloading: 100%	570/570 [00:00<00:00, 16.4kB/s]

Let's apply the tokenizer to one sentence just to see the output.

```
# 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])))

     Original:  Our friends won't buy this analysis, let alone the next one we propose.
    Tokenized:  ['our', 'friends', 'won', "'", 't', 'buy', 'this', 'analysis', ',', 'let', 'alone', 'the', 'next
    Token IDs:  [2256, 2814, 2180, 1005, 1056, 4965, 2023, 4106, 1010, 2292, 2894, 1996, 2279, 2028, 2057, 16599
```

When we actually convert all of our sentences, we'll use the `tokenize.encode` function to handle both steps, rather than calling `tokenize` and `convert_tokens_to_ids` separately.

Before we can do that, though, we need to talk about some of BERT's formatting requirements.

⌄ 3.2. Required Formatting

The above code left out a few required formatting steps that we'll look at here.

Side Note: The input format to BERT seems "over-specified" to me... We are required to give it a number of pieces of information which seem redundant, or like they could easily be inferred from the data without us explicity providing it. But it is what it is, and I suspect it will make more sense once I have a deeper understanding of the BERT internals.

We are required to:

1. Add special tokens to the start and end of each sentence.
2. Pad & truncate all sentences to a single constant length.
3. Explicitly differentiate real tokens from padding tokens with the "attention mask".

⌄ Special Tokens

`[SEP]`

At the end of every sentence, we need to append the special `[SEP]` token.

This token is an artifact of two-sentence tasks, where BERT is given two separate sentences and asked to determine something (e.g., can the answer to the question in sentence A be found in sentence B?).

I am not certain yet why the token is still required when we have only single-sentence input, but it is!

`[CLS]`

For classification tasks, we must prepend the special `[CLS]` token to the beginning of every sentence.

This token has special significance. BERT consists of 12 Transformer layers. Each transformer takes in a list of token embeddings, and produces the same number of embeddings on the output (but with the feature values changed, of course!).
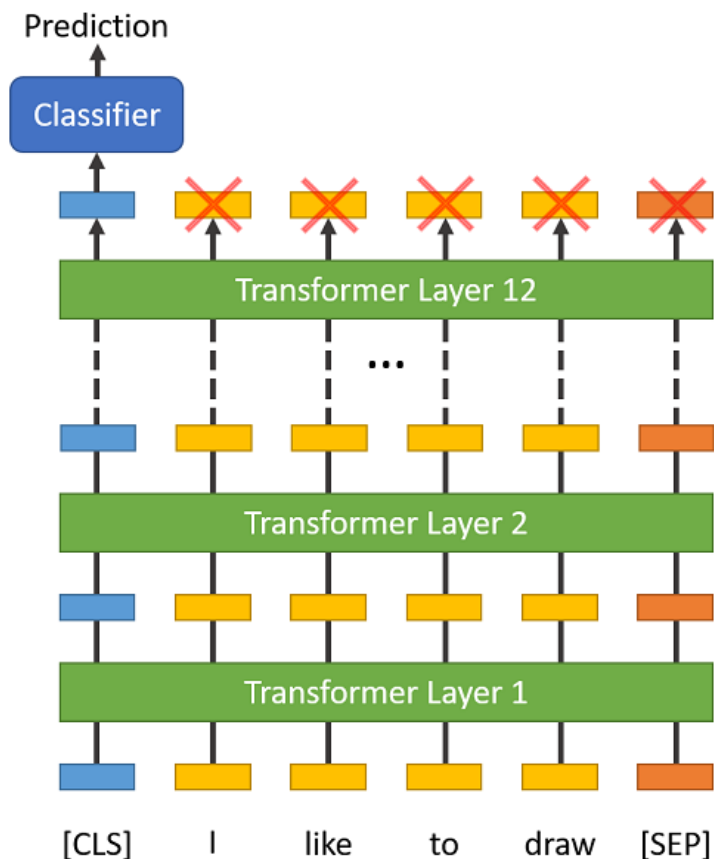
On the output of the final (12th) transformer, *only the first embedding (corresponding to the [CLS] token) is used by the classifier*.

> "The first token of every sequence is always a special classification token ([CLS]). The final hidden state corresponding to this token is used as the aggregate sequence representation for classification tasks." (from the BERT paper)

You might think to try some pooling strategy over the final embeddings, but this isn't necessary. Because BERT is trained to only use this [CLS] token for classification, we know that the model has been motivated to encode everything it needs for the classification step into that single 768-value embedding vector. It's already done the pooling for us!

⌄ Sentence Length & Attention Mask

The sentences in our dataset obviously have varying lengths, so how does BERT handle this?

BERT has two constraints:

1. All sentences must be padded or truncated to a single, fixed length.
2. The maximum sentence length is 512 tokens.

Padding is done with a special [PAD] token, which is at index 0 in the BERT vocabulary. The below illustration demonstrates padding out to a "MAX_LEN" of 8 tokens.

3.3. Tokenize Dataset

The transformers library provides a helpful `encode` function which will handle most of the parsing and data prep steps for us.

Before we are ready to encode our text, though, we need to decide on a **maximum sentence length** for padding / truncating to.

The below cell will perform one tokenization pass of the dataset in order to measure the maximum sentence length.

```
max_len = 0

# For every sentence...
for sent in sentences:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)

    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))

print('Max sentence length: ', max_len)

    Max sentence length:  47
```

Just in case there are some longer test sentences, I'll set the maximum length to 64.

Now we're ready to perform the real tokenization.

The `tokenizer.encode_plus` function combines multiple steps for us:

1. Split the sentence into tokens.
2. Add the special `[CLS]` and `[SEP]` tokens.
3. Map the tokens to their IDs.
4. Pad or truncate all sentences to the same length.
5. Create the attention masks which explicitly differentiate real tokens from `[PAD]` tokens.

The first four features are in `tokenizer.encode`, but I'm using `tokenizer.encode_plus` to get the fifth item (attention masks). Documentation is [here](#).

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

# For every sentence...
for sent in sentences:
    # `encode_plus` 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.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 64,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

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

```
    Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncatio
    /usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2269: FutureWarning: The `pad
      FutureWarning,
    Original:  Our friends won't buy this analysis, let alone the next one we propose.
    Token IDs: tensor([  101,  2256,  2814,  2180,  1005,  1056,  4965,  2023,  4106,  1010,
             2292,  2894,  1996,  2279,  2028,  2057, 16599,  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])
```

3.4. Training & Validation Split

Divide up our training set to use 90% for training and 10% for validation.

```
from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
    7,695 training samples
      856 validation samples
```

We'll also create an iterator for our dataset using the torch DataLoader class. 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.

```
from torch.utils.data import 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 DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

4. Train Our Classification Model

Now that our input data is properly formatted, it's time to fine tune the BERT model.

4.1. BertForSequenceClassification

For this task, we first want to modify the pre-trained BERT model to give outputs for classification, and then we want to continue training the model on our dataset until that the entire model, end-to-end, is well-suited for our task.

Thankfully, the huggingface pytorch implementation includes a set of interfaces designed for a variety of NLP tasks. Though these interfaces are all built on top of a trained BERT model, each has different top layers and output types designed to accomodate their specific NLP task.

Here is the current list of classes provided for fine-tuning:

- BertModel
- BertForPreTraining
- BertForMaskedLM
- BertForNextSentencePrediction
- **BertForSequenceClassification** - The one we'll use.
- BertForTokenClassification
- BertForQuestionAnswering

The documentation for these can be found under here.

We'll be using BertForSequenceClassification. This is the normal BERT model with an added single linear layer on top for classification that we will use as a sentence classifier. As we feed input data, the entire pre-trained BERT model and the additional untrained classification layer is trained on our specific task.

OK, let's load BERT! There are a few different pre-trained BERT models available. "bert-base-uncased" means the version that has only lowercase letters ("uncased") and is the smaller version of the two ("base" vs "large").

The documentation for `from_pretrained` can be found here, with the additional parameters defined here.

```
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()
```

Just for curiosity's sake, we can browse all of the model's parameters by name here.

In the below cell, I've printed out the names and dimensions of the weights for:

1. The embedding layer.
2. The first of the twelve transformers.
3. The output layer.

```
# Get all of the model's parameters as a list of tuples.
params = list(model.named_parameters())

print('The BERT model has {:} different named parameters.\n'.format(len(params)))

print('==== Embedding Layer ====\n')

for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====\n')

for p in params[5:21]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-4:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                   (30522, 768)
bert.embeddings.position_embeddings.weight                 (512, 768)
bert.embeddings.token_type_embeddings.weight                 (2, 768)
bert.embeddings.LayerNorm.weight                               (768,)
bert.embeddings.LayerNorm.bias                                 (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight           (768, 768)
bert.encoder.layer.0.attention.self.query.bias                 (768,)
bert.encoder.layer.0.attention.self.key.weight             (768, 768)
bert.encoder.layer.0.attention.self.key.bias                   (768,)
bert.encoder.layer.0.attention.self.value.weight           (768, 768)
bert.encoder.layer.0.attention.self.value.bias                 (768,)
bert.encoder.layer.0.attention.output.dense.weight         (768, 768)
bert.encoder.layer.0.attention.output.dense.bias               (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight         (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias           (768,)
bert.encoder.layer.0.intermediate.dense.weight            (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                  (3072,)
bert.encoder.layer.0.output.dense.weight                  (768, 3072)
bert.encoder.layer.0.output.dense.bias                         (768,)
bert.encoder.layer.0.output.LayerNorm.weight                   (768,)
bert.encoder.layer.0.output.LayerNorm.bias                     (768,)

==== Output Layer ====

bert.pooler.dense.weight                                   (768, 768)
```

```
bert.pooler.dense.bias                              (768,)
classifier.weight                                   (2, 768)
classifier.bias                                     (2,)
```

4.2. Optimizer & Learning Rate Scheduler

Now that we have our model loaded we need to grab the training hyperparameters from within the stored model.

For the purposes of fine-tuning, the authors recommend choosing from the following values (from Appendix A.3 of the [BERT paper](#)):

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

We chose:

- Batch size: 32 (set when creating our DataLoaders)
- Learning rate: 2e-5
- Epochs: 4 (we'll see that this is probably too many...)

The epsilon parameter `eps = 1e-8` is "a very small number to prevent any division by zero in the implementation" (from [here](#)).

You can find the creation of the AdamW optimizer in `run_glue.py` [here](#).

```python
# 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.
                )
```

```
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:309: FutureWarning: This implementation
  FutureWarning,
```

```python
from transformers import get_linear_schedule_with_warmup

# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 4

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
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_glue.py
                                            num_training_steps = total_steps)
```

4.3. Training Loop

Below is our training loop. There's a lot going on, but fundamentally for each pass in our loop we have a trianing phase and a validation phase.

> *Thank you to [Stas Bekman](#) for contributing the insights and code for using validation loss to detect over-fitting!*

Training:

- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Clear out the gradients calculated in the previous pass.
  - In pytorch the gradients accumulate by default (useful for things like RNNs) unless you explicitly clear them out.
- Forward pass (feed input data through the network)

- Backward pass (backpropagation)
- Tell the network to update parameters with optimizer.step()
- Track variables for monitoring progress

Evalution:

- Unpack our data inputs and labels
- Load data onto the GPU for acceleration
- Forward pass (feed input data through the network)
- Compute loss on our validation data and track variables for monitoring progress

Pytorch hides all of the detailed calculations from us, but we've commented the code to point out which of the above steps are happening on each line.

> *PyTorch also has some [beginner tutorials](#) which you may also find helpful.*

Define a helper function for calculating accuracy.

```
import numpy as np

# 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)
```

Helper function for formatting elapsed times as `hh:mm:ss`

```
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))
```

We're ready to kick off the training!

```python
import random
import numpy as np

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

# 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)

# We'll store a number of quantities such as training and validation loss,
# validation accuracy, and timings.
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# 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_train_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-do-in-pytorch)
    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_dataloader), elapsed))

        # 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-grad-in-pytorch)
```

```python
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
        # In PyTorch, calling `model` will in turn call the model's `forward`
        # function and pass down the arguments. The `forward` function is
        # documented here:
        # https://huggingface.co/transformers/model_doc/bert.html#bertforsequenceclassification
        # The results are returned in a results object, documented here:
        # https://huggingface.co/transformers/main_classes/output.html#transformers.modeling_outputs.SequenceCla
        # Specifically, we'll get the loss (because we provided labels) and the
        # "logits"--the model outputs prior to activation.
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       labels=b_labels,
                       return_dict=True)

        loss = result.loss
        logits = result.logits

        # 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_train_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 all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

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

    # ========================================
    #               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
    total_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:
```

```python
            # 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)

            # Tell pytorch not to bother with constructing the compute graph during
            # the forward pass, since this is only needed for backprop (training).
            with torch.no_grad():

                # Forward pass, calculate logit predictions.
                # token_type_ids is the same as the "segment ids", which
                # differentiates sentence 1 and 2 in 2-sentence tasks.
                result = model(b_input_ids,
                               token_type_ids=None,
                               attention_mask=b_input_mask,
                               labels=b_labels,
                               return_dict=True)

            # Get the loss and "logits" output by the model. The "logits" are the
            # output values prior to applying an activation function like the
            # softmax.
            loss = result.loss
            logits = result.logits

            # Accumulate the validation loss.
            total_eval_loss += loss.item()

            # 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, and
            # accumulate it over all batches.
            total_eval_accuracy += flat_accuracy(logits, label_ids)


    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))

    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )

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

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
======== Epoch 1 / 4 ========
Training...
  Batch    40  of     241.    Elapsed: 0:00:13.
  Batch    80  of     241.    Elapsed: 0:00:26.
  Batch   120  of     241.    Elapsed: 0:00:39.
  Batch   160  of     241.    Elapsed: 0:00:52.
  Batch   200  of     241.    Elapsed: 0:01:06.
  Batch   240  of     241.    Elapsed: 0:01:20.

  Average training loss: 0.51
  Training epcoh took: 0:01:21

Running Validation...
  Accuracy: 0.80
  Validation Loss: 0.44
  Validation took: 0:00:03

======== Epoch 2 / 4 ========
Training...
  Batch    40  of     241.    Elapsed: 0:00:14.
  Batch    80  of     241.    Elapsed: 0:00:27.
  Batch   120  of     241.    Elapsed: 0:00:41.
  Batch   160  of     241.    Elapsed: 0:00:55.
  Batch   200  of     241.    Elapsed: 0:01:08.
  Batch   240  of     241.    Elapsed: 0:01:22.

  Average training loss: 0.33
  Training epcoh took: 0:01:22

Running Validation...
  Accuracy: 0.82
  Validation Loss: 0.47
  Validation took: 0:00:03

======== Epoch 3 / 4 ========
Training...
  Batch    40  of     241.    Elapsed: 0:00:14.
  Batch    80  of     241.    Elapsed: 0:00:27.
  Batch   120  of     241.    Elapsed: 0:00:41.
  Batch   160  of     241.    Elapsed: 0:00:55.
  Batch   200  of     241.    Elapsed: 0:01:09.
  Batch   240  of     241.    Elapsed: 0:01:22.

  Average training loss: 0.21
  Training epcoh took: 0:01:23

Running Validation...
  Accuracy: 0.82
  Validation Loss: 0.51
  Validation took: 0:00:03

======== Epoch 4 / 4 ========
Training...
  Batch    40  of     241.    Elapsed: 0:00:14.
  Batch    80  of     241.    Elapsed: 0:00:28.
  Batch   120  of     241.    Elapsed: 0:00:41.
  Batch   160  of     241.    Elapsed: 0:00:55.
```

Let's view the summary of the training process.

```
import pandas as pd

# Display floats with two decimal places.
pd.set_option('precision', 2)

# Create a DataFrame from our training statistics.
df_stats = pd.DataFrame(data=training_stats)

# Use the 'epoch' as the row index.
df_stats = df_stats.set_index('epoch')

# A hack to force the column headers to wrap.
#df = df.style.set_table_styles([dict(selector="th",props=[('max-width', '70px')])])

# Display the table.
df_stats
```

epoch	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
1	0.51	0.44	0.80	0:01:21	0:00:03
2	0.33	0.47	0.82	0:01:22	0:00:03
3	0.21	0.51	0.82	0:01:23	0:00:03
4	0.15	0.59	0.82	0:01:23	0:00:03

Notice that, while the the training loss is going down with each epoch, the validation loss is increasing! This suggests that we are training our model too long, and it's over-fitting on the training data.

(For reference, we are using 7,695 training samples and 856 validation samples).

Validation Loss is a more precise measure than accuracy, because with accuracy we don't care about the exact output value, but just which side of a threshold it falls on.

If we are predicting the correct answer, but with less confidence, then validation loss will catch this, while accuracy will not.

```python
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(df_stats['Training Loss'], 'b-o', label="Training")
plt.plot(df_stats['Valid. Loss'], 'g-o', label="Validation")

# Label the plot.
plt.title("Training & Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.xticks([1, 2, 3, 4])

plt.show()
```
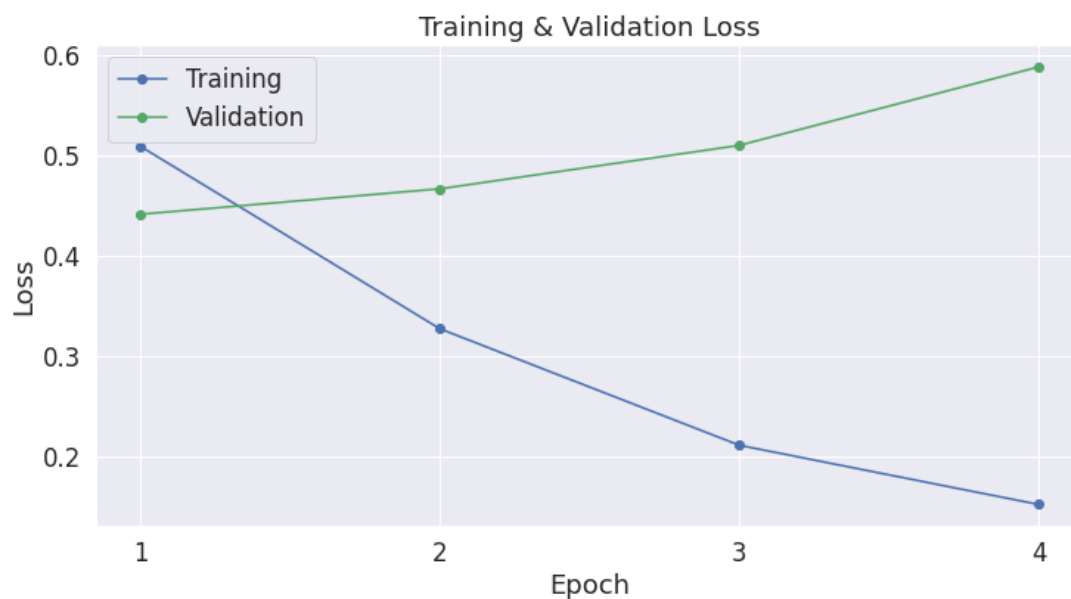

5. Performance On Test Set

5.1. Data Preparation

We'll need to apply all of the same steps that we did for the training data to prepare our test data set.

```python
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("./cola_public/raw/out_of_domain_dev.tsv", delimiter='\t', header=None, names=['sentence_source

# Report the number of sentences.
print('Number of test sentences: {:,}\n'.format(df.shape[0]))

# Create sentence and label lists
sentences = df.sentence.values
labels = df.label.values

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

# For every sentence...
for sent in sentences:
    # `encode_plus` 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.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = 64,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

# Set the batch size.
batch_size = 32

# Create the DataLoader.
prediction_data = TensorDataset(input_ids, attention_masks, labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)
```

```
Number of test sentences: 516

/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2269: FutureWarning: The `pad
  FutureWarning,
```

5.2. Evaluate on Test Set

With the test set prepared, we can apply our fine-tuned model to generate predictions on the test set.

```python
# Prediction on test set

print('Predicting labels for {:,} test sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
for batch in prediction_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 prediction
  with torch.no_grad():
      # Forward pass, calculate logit predictions.
      result = model(b_input_ids,
                    token_type_ids=None,
                    attention_mask=b_input_mask,
                    return_dict=True)

  logits = result.logits

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

  # Store predictions and true labels
  predictions.append(logits)
  true_labels.append(label_ids)

print('    DONE.')

    Predicting labels for 516 test sentences...
        DONE.
```

˅ Conclusion

This post demonstrates that with a pre-trained BERT model you can quickly and effectively create a high quality model with minimal effort and training time using the Pytorch interface, regardless of the specific NLP task you are interested in.

```
1. (10 pts) Fine-tune English BERT with **all** of your examples in rte-train.csv and report the accuracy on rte
> 67.04
2. (5 pts) Report the accuracy of this fine-tuned English BERT on the German subset of XNLI (this is zero-shot a
> 36.14
3. (10 pts) Modify the code to use multilingual (XLM-Roberta) model (xlm-roberta-base), keeping the same hyperpa
> 63.54
4. (5 pts) Report the accuracy of the multilingual model on German subset of XNLI (this is zero-shot accuracy of
> 36.83
```

˅ Do it yourself

Modify the above tutorial to train a Twitter sentiment classification model (you can keep all the hyperparameters as before for this homework, with epochs=4). In reality, fine-tuning the hyperparameters for your data and task (e.g., using validation step to pick the best number of epochs) might improve performance.

1. (10 pts) Fine-tune English BERT with **all** of your examples in rte-train.csv and report the accuracy on rte-val.csv. The tutorial was for a sentence classification task. When this becomes a sentence pair classification task the preprocessing slightly changes.

The tutorial pads special tokens to the beginning and end of a sentence now you have two sentences that you'll pad in a specific way. Look at the original [BERT](#) paper to see how you should do it.

> 57.04

2. (5 pts) Report the accuracy of this fine-tuned English BERT on the German subset of XNLI (this is zero-shot accuracy of the model trained on the English data and test on the German data). The data is provided in xnli-de-val.csv

> 36.14

3. (10 pts) Modify the code to use multilingual (XLM-Roberta) model (xlm-roberta-base), keeping the same hyperparameters as before. Fine-tune the multilingual model with **all** of your examples in rte-train.csv and report the accuracy on rte-val.csv.

> 63.54

4. (5 pts) Report the accuracy of the multilingual model on German subset of XNLI (this is zero-shot accuracy of the model trained on the English data and test on the German data). Does the zero-shot performance using multilingual model improve over the English only BERT model?

> 36.83

```python
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/rte_train.csv")

# Report the number of sentences.
print('Number of training sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

    Number of training sentences: 2,490

	sentence1	sentence2	label	idx
622	A Spanish man who murdered a Welsh couple with...	Juan Carmelo Santana was killed by Brian and T...	1	622
2305	Bledsoe will be released by the Buffalo Bills,...	The Buffalo Bills are releasing Drew Bledsoe, ...	0	2305
2480	On 24 March 1992, the number 3 power generatin...	Nuclear Power Plant of Sosnovyi Bor suffers an...	0	2480
985	Two men have been charged with starting the Za...	Cabrera and Cervantes are accused of having ca...	0	985
208	Scott's execution led to outrage in Ontario, a...	Scott's execution caused the Wolseley Expedition.	0	208
1822	Only 16 women were elected to the 223-member p...	Women are poorly represented in parliament.	0	1822
2385	These early men learned to make fire. They tra...	Humans existed 10,000 years ago.	0	2385
689	This year, however, the contest has taken on a...	Pyongyang is the capital of North Korea.	1	689
1466	Reward offers have led to recovery of stolen art.	Some stolen art is recovered through offered r...	0	1466
1539	Tintin, the Smurfs, Boule et Bill, Astérix, Na...	Spirou was created by Rob-Vel.	1	1539

```python
df.loc[df.label == 0].sample(5)[['sentence1', 'sentence2', 'label']]
```

	sentence1	sentence2	label
2338	The scandal is one of several threatening to u...	Jador Barbalho is a politician.	0
1781	On Aug. 6, 1945, an atomic bomb was exploded o...	In 1945, an atomic bomb was dropped on Hiroshima.	0
1238	Côte d'Ivoire's President Laurent Gbagbo promu...	New elections in Côte d'Ivoire will take place...	0
1745	Digging in a different trench Phoenix's roboti...	A robotic arm has been used for scientific dis...	0
467	In fact, Woolsey had had no first-hand experie...	James Woolsey is the director of the CIA.	0

```python
sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values
```

```
df.head()
```

	sentence1	sentence2	label	idx
0	No Weapons of Mass Destruction Found in Iraq Yet.	Weapons of Mass Destruction Found in Iraq.	1	0
1	A place of sorrow, after Pope John Paul II die...	Pope Benedict XVI is the new leader of the Rom...	0	1
2	Herceptin was already approved to treat the si...	Herceptin can be used to treat breast cancer.	0	2
3	Judie Vivian, chief executive at ProMedica, a ...	The previous name of Ho Chi Minh City was Saigon.	0	3
4	A man is due in court later charged with the m...	Paul Stewart Hutchinson is accused of having s...	1	4

```
from transformers import BertTokenizer

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

```
Loading BERT tokenizer...
```

tokenizer_config.json: 100%	28.0/28.0 [00:00<00:00, 992B/s]
vocab.txt: 100%	232k/232k [00:00<00:00, 3.46MB/s]
tokenizer.json: 100%	466k/466k [00:00<00:00, 9.35MB/s]
config.json: 100%	570/570 [00:00<00:00, 15.0kB/s]

```
# Print the original sentence.
print(' Original: ', sentences1[1])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences1[1]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences1[1])))
```

```
 Original:  A place of sorrow, after Pope John Paul II died, became a place of celebration, as Roman Catholi
Tokenized:  ['a', 'place', 'of', 'sorrow', ',', 'after', 'pope', 'john', 'paul', 'ii', 'died', ',', 'became'
Token IDs:  [1037, 2173, 1997, 14038, 1010, 2044, 4831, 2198, 2703, 2462, 2351, 1010, 2150, 1037, 2173, 1997
```

```
tokenizer(df.sentence1[0], df.sentence2[0],padding='max_length', truncation=True, max_length=128, return_tensors
```

```
{'input_ids': <tf.Tensor: shape=(1, 128), dtype=int32, numpy=
array([[ 101, 2053, 4255, 1997, 3742, 6215, 2179, 1999, 5712, 2664, 1012,
         102, 4255, 1997, 3742, 6215, 2179, 1999, 5712, 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]], dtype=int32)>, 'token_type_ids': <tf.Tensor: shape=(1,
128), dtype=int32, numpy=
array([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 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]],
      dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(1, 128), dtype=int32, numpy=
array([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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]],
      dtype=int32)>}
```

```python
max_len = 0
input_ids = []
attention_masks = []


# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                )

    # Update the maximum sentence pair length.
    max_len = max(max_len, len(encoded_dict['input_ids'][0]))

    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pa
  warnings.warn(
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflwing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```python
# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

print('Max sentence pair length: ', max_len)
```

```
    Max sentence pair length:  64
    <ipython-input-27-9c63b32cdd19>:4: UserWarning: To copy construct from a tensor, it is recommended to use so
      labels = torch.tensor(labels)
```

```python
from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
    2,241 training samples
      249 validation samples
```

```python
from torch.utils.data import 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 = 16

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```
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()
```

```
    Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-un
    You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferenc
    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(
            (0-11): 12 x 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)
                (intermediate_act_fn): GELUActivation()
              )
              (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)
    )
```

```python
# Get all of the model's parameters as a list of tuples.
params = list(model.named_parameters())

print('The BERT model has {:} different named parameters.\n'.format(len(params)))

print('==== Embedding Layer ====\n')

for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====\n')

for p in params[5:21]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-4:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
The BERT model has 201 different named parameters.

==== Embedding Layer ====

bert.embeddings.word_embeddings.weight                    (30522, 768)
bert.embeddings.position_embeddings.weight                  (512, 768)
bert.embeddings.token_type_embeddings.weight                  (2, 768)
bert.embeddings.LayerNorm.weight                                (768,)
bert.embeddings.LayerNorm.bias                                  (768,)

==== First Transformer ====

bert.encoder.layer.0.attention.self.query.weight            (768, 768)
bert.encoder.layer.0.attention.self.query.bias                 (768,)
bert.encoder.layer.0.attention.self.key.weight              (768, 768)
bert.encoder.layer.0.attention.self.key.bias                   (768,)
bert.encoder.layer.0.attention.self.value.weight            (768, 768)
bert.encoder.layer.0.attention.self.value.bias                 (768,)
bert.encoder.layer.0.attention.output.dense.weight          (768, 768)
bert.encoder.layer.0.attention.output.dense.bias               (768,)
bert.encoder.layer.0.attention.output.LayerNorm.weight         (768,)
bert.encoder.layer.0.attention.output.LayerNorm.bias           (768,)
bert.encoder.layer.0.intermediate.dense.weight             (3072, 768)
bert.encoder.layer.0.intermediate.dense.bias                  (3072,)
bert.encoder.layer.0.output.dense.weight                   (768, 3072)
bert.encoder.layer.0.output.dense.bias                         (768,)
bert.encoder.layer.0.output.LayerNorm.weight                   (768,)
bert.encoder.layer.0.output.LayerNorm.bias                     (768,)

==== Output Layer ====

bert.pooler.dense.weight                                    (768, 768)
bert.pooler.dense.bias                                         (768,)
classifier.weight                                             (2, 768)
classifier.bias                                                  (2,)
```

```python
# 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.
                )
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation
  warnings.warn(
```

```python
from transformers import get_linear_schedule_with_warmup


# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 4


# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
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_glue.py
                                            num_training_steps = total_steps)



import numpy as np

# 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))



import random
import numpy as np

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

# 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)

# We'll store a number of quantities such as training and validation loss,
# validation accuracy, and timings.
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# 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()
```

```python
    # Reset the total loss for this epoch.
    total_train_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-do-in-pytorch)
    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_dataloader), elapsed))

        # 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-grad-in-pytorch)
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
        # In PyTorch, calling `model` will in turn call the model's `forward`
        # function and pass down the arguments. The `forward` function is
        # documented here:
        # https://huggingface.co/transformers/model_doc/bert.html#bertforsequenceclassification
        # The results are returned in a results object, documented here:
        # https://huggingface.co/transformers/main_classes/output.html#transformers.modeling_outputs.SequenceClas
        # Specifically, we'll get the loss (because we provided labels) and the
        # "logits"--the model outputs prior to activation.
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       labels=b_labels,
                       return_dict=True)

        loss = result.loss
        logits = result.logits

        # 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_train_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()
```

```python
        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

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


    # ========================================
    #               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
    total_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:

        # 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)

        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():

            # Forward pass, calculate logit predictions.
            # token_type_ids is the same as the "segment ids", which
            # differentiates sentence 1 and 2 in 2-sentence tasks.
            result = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels,
                           return_dict=True)

        # Get the loss and "logits" output by the model. The "logits" are the
        # output values prior to applying an activation function like the
        # softmax.
        loss = result.loss
        logits = result.logits

        # Accumulate the validation loss.
        total_eval_loss += loss.item()

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

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

        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)


    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))

    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )

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

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
    ======== Epoch 1 / 4 ========
    Training...
      Batch    40  of    141.    Elapsed: 0:00:08.
      Batch    80  of    141.    Elapsed: 0:00:16.
      Batch   120  of    141.    Elapsed: 0:00:24.

      Average training loss: 0.69
      Training epcoh took: 0:00:28

    Running Validation...
      Accuracy: 0.55
      Validation Loss: 0.69
      Validation took: 0:00:01

    ======== Epoch 2 / 4 ========
    Training...
      Batch    40  of    141.    Elapsed: 0:00:08.
      Batch    80  of    141.    Elapsed: 0:00:16.
      Batch   120  of    141.    Elapsed: 0:00:24.

      Average training loss: 0.67
      Training epcoh took: 0:00:28

    Running Validation...
      Accuracy: 0.58
      Validation Loss: 0.68
      Validation took: 0:00:01

    ======== Epoch 3 / 4 ========
    Training...
      Batch    40  of    141.    Elapsed: 0:00:08.
      Batch    80  of    141.    Elapsed: 0:00:16.
      Batch   120  of    141.    Elapsed: 0:00:23.

      Average training loss: 0.56
      Training epcoh took: 0:00:27

    Running Validation...
      Accuracy: 0.60
      Validation Loss: 0.73
      Validation took: 0:00:01
```

```
======== Epoch 4 / 4 ========
Training...
  Batch    40  of    141.    Elapsed: 0:00:08.
  Batch    80  of    141.    Elapsed: 0:00:16.
  Batch   120  of    141.    Elapsed: 0:00:24.

  Average training loss: 0.41
  Training epcoh took: 0:00:28

Running Validation...
  Accuracy: 0.61
  Validation Loss: 0.75
  Validation took: 0:00:01

Training complete!
```

```python
import pandas as pd

# Display floats with two decimal places.
pd.set_option('display.precision', 2)

# Create a DataFrame from our training statistics.
df_stats = pd.DataFrame(data=training_stats)

# Use the 'epoch' as the row index.
df_stats = df_stats.set_index('epoch')

# A hack to force the column headers to wrap.
#df = df.style.set_table_styles([dict(selector="th",props=[('max-width', '70px')])])

# Display the table.
df_stats
```

epoch	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
1	0.69	0.69	0.55	0:00:28	0:00:01
2	0.67	0.68	0.58	0:00:28	0:00:01
3	0.56	0.73	0.60	0:00:27	0:00:01
4	0.41	0.75	0.61	0:00:28	0:00:01

```python
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(df_stats['Training Loss'], 'b-o', label="Training")
plt.plot(df_stats['Valid. Loss'], 'g-o', label="Validation")

# Label the plot.
plt.title("Training & Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.xticks([1, 2, 3, 4])

plt.show()
```

```
UsageError: Line magic function `%` not found.
```

```python
# 5. Performance On Test Set
```

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/rte_val.csv")

# Report the number of sentences.
print('Number of testing sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

Number of testing sentences: 277

	sentence1	sentence2	label	idx
135	Mount Olympus towers up from the center of the...	Mount Olympus is in the center of the earth.	0	135
80	The extension of the effective period of marke...	Prospective drugs must have long market life a...	1	80
178	While Bush was in the air, the White House was...	The U.S. military evacuated U.S. citizens.	1	178
86	The job gains mean that President Bush can ce...	More jobs were created during President Bush's...	0	86
125	Pole Robert Kubica became the highlight of the...	Robert Kubica has been appointed by Scuderia F...	1	125
140	The witness then spotted Rodriguez crouching b...	The witness rescued Rodriguez.	1	140
256	Plans are being drawn up to build a £3.3m work...	Charles Darwin took a voyage on a ship named H...	0	256
265	Thursday's assassination of Ms. Bhutto has plu...	Musharraf has ordered rioters to take firm act...	1	265
101	The Kota ("Fort"), or Old City, for example,...	The Kota is the country's business center.	0	101
13	A jury is slated to decide for the first time ...	Jack Kevorkian is the real name of "Dr. Death".	0	13

```python
# sentences1 = df['sentence1'].values
# sentences2 = df['sentence2'].values
# labels = df['label'].values

# max_len = 0
# input_ids = []
# attention_masks = []

# # For every sentence pair...
# for sent1, sent2 in zip(sentences1, sentences2):
#     # `encode_plus` will:
#     #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
#     #   (2) Pad or truncate the sentence pair to `max_length`
#     #   (3) Create attention masks for [PAD] tokens.
#     encoded_dict = tokenizer.encode_plus(
#                         sent1, sent2,
#                         add_special_tokens=True, # Add '[CLS]' and '[SEP]'
#                         max_length=64,           # Pad & truncate all sentence pairs.
#                         pad_to_max_length=True,
#                         return_attention_mask=True,  # Construct attn. masks.
#                         return_tensors='pt',     # Return pytorch tensors.
#                         truncation=True ,
#                    )


#     # Add the encoded sentence pair to the list.
#     input_ids.append(encoded_dict['input_ids'])
#     attention_masks.append(encoded_dict['attention_mask'])



# # Convert the lists into tensors.
# input_ids = torch.cat(input_ids, dim=0)
# attention_masks = torch.cat(attention_masks, dim=0)
# labels = torch.tensor(labels)

# batch_size = 16

# # Create the DataLoader.
# prediction_data = TensorDataset(input_ids, attention_masks, labels)
# prediction_sampler = SequentialSampler(prediction_data)
# prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)




sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values
```

```python
max_len = 0
input_ids = []
attention_masks = []

# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                   )

    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pa
  warnings.warn(
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
      Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```python
# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)


batch_size = 16

# Create the DataLoader.
prediction_data = TensorDataset(input_ids, attention_masks, labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)


# Prediction on test set

print('Predicting labels for {:,} test sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
for batch in prediction_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 prediction
  with torch.no_grad():
      # Forward pass, calculate logit predictions.
      result = model(b_input_ids,
                    token_type_ids=None,
                    attention_mask=b_input_mask,
                    return_dict=True)

  logits = result.logits

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

  # Store predictions and true labels
  predictions.append(logits)
  true_labels.append(label_ids)

print('    DONE.')
```

```
    Predicting labels for 277 test sentences...
        DONE.
```

```python
# Flatten the predictions and true labels
flat_predictions = np.concatenate(predictions, axis=0)
flat_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
flat_predictions = np.argmax(flat_predictions, axis=1).flatten()

# Calculate accuracy
accuracy = np.sum(flat_predictions == flat_true_labels) / len(flat_true_labels)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
    Test Accuracy: 57.04%
```

```python
# German Dataset - Zero Shot


# # Load the dataset into a pandas dataframe.
# df = pd.read_csv("/content/xnli_de_val.csv")

# # Report the number of sentences.
# print('Number of testing sentences: {:,}\n'.format(df.shape[0]))

# # Display 10 random rows from the data.
# df.sample(10)

# sentences1 = df['sentence1'].values
# sentences2 = df['sentence2'].values
# labels = df['label'].values

# max_len = 0
# input_ids = []
# attention_masks = []

# # For every sentence pair...
# for sent1, sent2 in zip(sentences1, sentences2):
#     # `encode_plus` will:
#     #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
#     #   (2) Pad or truncate the sentence pair to `max_length`
#     #   (3) Create attention masks for [PAD] tokens.
#     encoded_dict = tokenizer.encode_plus(
#                         sent1, sent2,
#                         add_special_tokens=True, # Add '[CLS]' and '[SEP]'
#                         max_length=64,           # Pad & truncate all sentence pairs.
#                         pad_to_max_length=True,
#                         return_attention_mask=True,   # Construct attn. masks.
#                         return_tensors='pt',     # Return pytorch tensors.
#                         truncation=True ,
#                    )


#     # Add the encoded sentence pair to the list.
#     input_ids.append(encoded_dict['input_ids'])
#     attention_masks.append(encoded_dict['attention_mask'])

# # Convert the lists into tensors.
# input_ids = torch.cat(input_ids, dim=0)
# attention_masks = torch.cat(attention_masks, dim=0)
# labels = torch.tensor(labels)


# batch_size = 16

# # Create the DataLoader.
# prediction_data = TensorDataset(input_ids, attention_masks, labels)
# prediction_sampler = SequentialSampler(prediction_data)
# prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)


# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/xnli_de_val.csv")

# Report the number of sentences.
print('Number of testing sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

Number of testing sentences: 2,490

	sentence1	sentence2	label	idx
1801	Ihre Investition setzt die hohe Qualität aller...	Größere Investitionen sind für das Museum hilf...	1	1801
1190	Weiter geht es in östlicher Richtung an der un...	Die Komische Oper ist die berühmteste in Deuts...	1	1190
1817	Ich glaube genau das Gegenteil.	Ich glaube du liegst falsch, dass die Antwort,...	1	1817
251	Oh, es war Snake River oh Snake River mit viel...	Snake River ist voller Schlangen.	0	251

```python
sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values
```

| **1411** | Im Gegensatz, die Auswirkung des Volumens is h... | Der Einfluss in Frankreich nimmt zu. | 1 | 1411 |

```python
max_len = 0
input_ids = []
attention_masks = []


# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                    )


    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pa
  warnings.warn(
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
    Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```python
# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)


batch_size = 16

# Create the DataLoader.
prediction_data = TensorDataset(input_ids, attention_masks, labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)


# Prediction on test set

print('Predicting labels for {:,} test sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
for batch in prediction_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 prediction
  with torch.no_grad():
      # Forward pass, calculate logit predictions.
      result = model(b_input_ids,
                     token_type_ids=None,
                     attention_mask=b_input_mask,
                     return_dict=True)

  logits = result.logits

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

  # Store predictions and true labels
  predictions.append(logits)
  true_labels.append(label_ids)

print('    DONE.')
```

```
    Predicting labels for 2,490 test sentences...
        DONE.
```

```python
# Flatten the predictions and true labels
flat_predictions = np.concatenate(predictions, axis=0)
flat_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
flat_predictions = np.argmax(flat_predictions, axis=1).flatten()

# Calculate accuracy
accuracy = np.sum(flat_predictions == flat_true_labels) / len(flat_true_labels)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
Test Accuracy: 36.14%
```

```python
# As Expected, accuracy goes down for Zero - shot learning
```

```python
# XLM-Roberta-Base
```

```python
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/rte_train.csv")

# Report the number of sentences.
print('Number of training sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

```
Number of training sentences: 2,490
```

	sentence1	sentence2	label	idx
271	Reagan was seriously wounded by a bullet fired...	John W. Hinckley Jr. shot Reagan in the chest.	1	271
1451	Speaking outside the hospital moments after ge...	Ciancio was hospitalized for speaking to the b...	1	1451
1125	Geoscience Australia said the 5.3 quake did no...	A tsunami is a series of waves created when a ...	1	1125
2004	Mohamed al-Sumaidaie, Iraq's ambassador to the...	Mohamed al-Sumaidaie is related by blood to th...	1	2004
2015	Hudson's mother, Darnell Donerson, 57, and bro...	Jason Hudson was Jennifer's brother.	1	2015
1011	Ministers of the Association of Southeast Asia...	Ministers of the Association of Southeast Asia...	1	1011
2356	Energy analysts said oil prices could soar as ...	Oil prices surged.	1	2356
451	In more than 40 years as a broadcaster, right ...	Sir Clement Freud was the grandson of Sigmund ...	1	451
1403	John Paul II was the first non-Italian Pope i...	John Paul II is Italian.	1	1403
990	Mr Lopez Obrador, who lost July's presidential...	Mr Lopez Obrador didn't loose the presidential...	1	990

```python
df.loc[df.label == 0].sample(5)[['sentence1', 'sentence2', 'label']]
```

	sentence1	sentence2	label
2234	Wal-Mart has received a lot of negative public...	Wal-Mart faces allegations for underpaying emp...	0
2311	A front page editorial in South Africa's large...	"The Sunday Times" is the name of the largest ...	0
1202	A teenager slashed a woman to death as he atte...	Alan Tanguay killed a woman to steal $6.	0
1890	It was difficult to assess the objectives behi...	Guerrillas increase activity in East Timor.	0
315	Only a few Mag-lev trains have been used comme...	Maglev is commercially used.	0

```python
sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values
```

```python
from transformers import XLMRobertaTokenizer

# Load the BERT tokenizer.
print('Loading XLMRobertaTokenizer...')
tokenizer = XLMRobertaTokenizer.from_pretrained('xlm-roberta-base')
```

```
    Loading XLMRobertaTokenizer...
```

```python
# Print the original sentence.
print(' Original: ', sentences1[1])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences1[1]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences1[1])))
```

```
     Original:  A place of sorrow, after Pope John Paul II died, became a place of celebration, as Roman Catholi
    Tokenized:  ['_A', '_place', '_of', '_sor', 'row', ',', '_after', '_Pop', 'e', '_John', '_Paul', '_II', '_di
    Token IDs:  [62, 3687, 111, 6217, 15555, 4, 7103, 11584, 13, 4939, 10208, 1995, 68, 71, 4, 100512, 10, 3687,
```

```python
tokenizer(df.sentence1[0], df.sentence2[0],padding='max_length', truncation=True, max_length=128, return_tensors
```

```
    {'input_ids': <tf.Tensor: shape=(1, 128), dtype=int32, numpy=
    array([[     0,    438,   1401,   2631,   5245,    111,  74227,    262,
            36716,   1830, 215624,     23,  69496,  63519,      5,      2,
                2,   1401,   2631,   5245,    111,  74227,    262,  36716,
             1830, 215624,     23,  69496,      5,      2,      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,      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,      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,      1,      1,      1,      1,      1,      1,
                1,      1,      1,      1,      1,      1,      1,      1]],
          dtype=int32)>, 'attention_mask': <tf.Tensor: shape=(1, 128), dtype=int32, numpy=
    array([[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, 1, 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]],
          dtype=int32)>}
```

```python
max_len = 0
input_ids = []
attention_masks = []


# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                   )

    # Update the maximum sentence pair length.
    max_len = max(max_len, len(encoded_dict['input_ids'][0]))

    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pa
  warnings.warn(
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
       Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```python
# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

print('Max sentence pair length: ', max_len)
```

```
    Max sentence pair length:  64
```

```python
from torch.utils.data import TensorDataset, random_split

# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.9 * len(dataset))
val_size = len(dataset) - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
    2,241 training samples
      249 validation samples
```

```python
from torch.utils.data import 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 = 16

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )

# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```python
from transformers import XLMRobertaForSequenceClassification, AdamW

# Load XLMRobertaForSequenceClassification, the pretrained XLM-RoBERTa model with a single
# linear classification layer on top.
model = XLMRobertaForSequenceClassification.from_pretrained(
    "xlm-roberta-base", # Use the XLM-RoBERTa base model
    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 attention weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

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

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-ro
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inferenc
XLMRobertaForSequenceClassification(
  (roberta): XLMRobertaModel(
    (embeddings): XLMRobertaEmbeddings(
      (word_embeddings): Embedding(250002, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): XLMRobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x XLMRobertaLayer(
          (attention): XLMRobertaAttention(
            (self): XLMRobertaSelfAttention(
              (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): XLMRobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): XLMRobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): XLMRobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): XLMRobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

```python
# 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()
```

```python
# Get all of the model's parameters as a list of tuples.
params = list(model.named_parameters())

print('The BERT model has {:} different named parameters.\n'.format(len(params)))

print('==== Embedding Layer ====\n')

for p in params[0:5]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== First Transformer ====\n')

for p in params[5:21]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))

print('\n==== Output Layer ====\n')

for p in params[-4:]:
    print("{:<55} {:>12}".format(p[0], str(tuple(p[1].size()))))
```

```
    The BERT model has 201 different named parameters.

    ==== Embedding Layer ====

    roberta.embeddings.word_embeddings.weight                 (250002, 768)
    roberta.embeddings.position_embeddings.weight                (514, 768)
    roberta.embeddings.token_type_embeddings.weight                (1, 768)
    roberta.embeddings.LayerNorm.weight                              (768,)
    roberta.embeddings.LayerNorm.bias                                (768,)

    ==== First Transformer ====

    roberta.encoder.layer.0.attention.self.query.weight         (768, 768)
    roberta.encoder.layer.0.attention.self.query.bias               (768,)
    roberta.encoder.layer.0.attention.self.key.weight           (768, 768)
    roberta.encoder.layer.0.attention.self.key.bias                 (768,)
    roberta.encoder.layer.0.attention.self.value.weight         (768, 768)
    roberta.encoder.layer.0.attention.self.value.bias               (768,)
    roberta.encoder.layer.0.attention.output.dense.weight       (768, 768)
    roberta.encoder.layer.0.attention.output.dense.bias             (768,)
    roberta.encoder.layer.0.attention.output.LayerNorm.weight       (768,)
    roberta.encoder.layer.0.attention.output.LayerNorm.bias         (768,)
    roberta.encoder.layer.0.intermediate.dense.weight          (3072, 768)
    roberta.encoder.layer.0.intermediate.dense.bias                (3072,)
    roberta.encoder.layer.0.output.dense.weight                (768, 3072)
    roberta.encoder.layer.0.output.dense.bias                       (768,)
    roberta.encoder.layer.0.output.LayerNorm.weight                 (768,)
    roberta.encoder.layer.0.output.LayerNorm.bias                   (768,)

    ==== Output Layer ====

    classifier.dense.weight                                     (768, 768)
    classifier.dense.bias                                           (768,)
    classifier.out_proj.weight                                      (2, 768)
    classifier.out_proj.bias                                           (2,)
```

```python
# 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.
                )
```

```
    /usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: FutureWarning: This implementation
      warnings.warn(
```

```python
from transformers import get_linear_schedule_with_warmup

# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 4

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
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_glue.py
                                            num_training_steps = total_steps)
```

```python
import numpy as np

# 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)
```

```python
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))
```

```python
import random
import numpy as np

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

# 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)

# We'll store a number of quantities such as training and validation loss,
# validation accuracy, and timings.
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# 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_train_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-do-in-pytorch)
    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_dataloader), elapsed))

        # 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-grad-in-pytorch)
```

```python
        model.zero_grad()

        # Perform a forward pass (evaluate the model on this training batch).
        # In PyTorch, calling `model` will in turn call the model's `forward`
        # function and pass down the arguments. The `forward` function is
        # documented here:
        # https://huggingface.co/transformers/model_doc/bert.html#bertforsequenceclassification
        # The results are returned in a results object, documented here:
        # https://huggingface.co/transformers/main_classes/output.html#transformers.modeling_outputs.SequenceCla
        # Specifically, we'll get the loss (because we provided labels) and the
        # "logits"--the model outputs prior to activation.
        result = model(b_input_ids,
                       token_type_ids=None,
                       attention_mask=b_input_mask,
                       labels=b_labels,
                       return_dict=True)

        loss = result.loss
        logits = result.logits

        # 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_train_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 all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)

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

    # ========================================
    #               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
    total_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0

    # Evaluate data for one epoch
    for batch in validation_dataloader:
```

```python
        # 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)

        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():

            # Forward pass, calculate logit predictions.
            # token_type_ids is the same as the "segment ids", which
            # differentiates sentence 1 and 2 in 2-sentence tasks.
            result = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels,
                           return_dict=True)

        # Get the loss and "logits" output by the model. The "logits" are the
        # output values prior to applying an activation function like the
        # softmax.
        loss = result.loss
        logits = result.logits

        # Accumulate the validation loss.
        total_eval_loss += loss.item()

        # 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, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)


    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))

    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)

    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)

    print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    print("  Validation took: {:}".format(validation_time))

    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )

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

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
======== Epoch 1 / 4 ========
Training...
  Batch    40  of     141.    Elapsed: 0:00:11.
  Batch    80  of     141.    Elapsed: 0:00:22.
  Batch   120  of     141.    Elapsed: 0:00:33.

  Average training loss: 0.70
  Training epcoh took: 0:00:38

Running Validation...
  Accuracy: 0.48
  Validation Loss: 0.69
  Validation took: 0:00:01

======== Epoch 2 / 4 ========
Training...
  Batch    40  of     141.    Elapsed: 0:00:11.
  Batch    80  of     141.    Elapsed: 0:00:21.
  Batch   120  of     141.    Elapsed: 0:00:32.

  Average training loss: 0.70
  Training epcoh took: 0:00:37

Running Validation...
  Accuracy: 0.53
  Validation Loss: 0.69
  Validation took: 0:00:01

======== Epoch 3 / 4 ========
Training...
  Batch    40  of     141.    Elapsed: 0:00:10.
  Batch    80  of     141.    Elapsed: 0:00:21.
  Batch   120  of     141.    Elapsed: 0:00:31.

  Average training loss: 0.69
  Training epcoh took: 0:00:37

Running Validation...
  Accuracy: 0.57
  Validation Loss: 0.69
  Validation took: 0:00:01

======== Epoch 4 / 4 ========
Training...
  Batch    40  of     141.    Elapsed: 0:00:11.
  Batch    80  of     141.    Elapsed: 0:00:21.
  Batch   120  of     141.    Elapsed: 0:00:32.

  Average training loss: 0.66
  Training epcoh took: 0:00:37

Running Validation...
  Accuracy: 0.59
  Validation Loss: 0.68
  Validation took: 0:00:01

Training complete!
```

```python
# As the accuracy is still increasing, we can keep training the model but as we are asked to keep the same hyper


import pandas as pd

# Display floats with two decimal places.
pd.set_option('display.precision', 2)

# Create a DataFrame from our training statistics.
df_stats = pd.DataFrame(data=training_stats)

# Use the 'epoch' as the row index.
df_stats = df_stats.set_index('epoch')

# A hack to force the column headers to wrap.
#df = df.style.set_table_styles([dict(selector="th",props=[('max-width', '70px')])])

# Display the table.
df_stats
```

epoch	Training Loss	Valid. Loss	Valid. Accur.	Training Time	Validation Time
1	0.70	0.69	0.48	0:00:38	0:00:01
2	0.70	0.69	0.53	0:00:37	0:00:01
3	0.69	0.69	0.57	0:00:37	0:00:01
4	0.66	0.68	0.59	0:00:37	0:00:01

```
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(df_stats['Training Loss'], 'b-o', label="Training")
plt.plot(df_stats['Valid. Loss'], 'g-o', label="Validation")

# Label the plot.
plt.title("Training & Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.xticks([1, 2, 3, 4])

plt.show()
```

```
    UsageError: Line magic function `%` not found.
```

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/rte_val.csv")

# Report the number of sentences.
print('Number of testing sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

```
    Number of testing sentences: 277
```

	sentence1	sentence2	label	idx
30	A smaller proportion of Yugoslavia's Italians ...	Slovenia has 3,000 inhabitants.	1	30
126	Confident that those grading papers would unde...	Haque wants to include English in some exams.	1	126
220	The Daily Telegraph, most prized asset in Lord...	Daily telegraph is sold.	0	220
142	The trial began today for a Pennsylvania state...	Yelenic molested his adopted son.	1	142
255	Phish disbands after a final concert in Vermon...	Rock band Phish holds final concert in Vermont.	0	255
236	Senator Dick Lugar, the ranking Republican mem...	Dick Lugar is a member of the Cuban-American c...	1	236
97	The Supreme Court said today states may bar th...	There is a Supreme Court decision about the re...	0	97
215	Toshiba, with NEC Corp. and Sanyo Electric Co....	Panasonic brand products are produced by Samsu...	1	215
256	Plans are being drawn up to build a £3.3m work...	Charles Darwin took a voyage on a ship named H...	0	256
144	Mrs. Bush's approval ratings have remained ver...	80% approve of Mr. Bush.	1	144

```python
sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values


max_len = 0
input_ids = []
attention_masks = []


# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                )


    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])



# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)

batch_size = 16

# Create the DataLoader.
prediction_data = TensorDataset(input_ids, attention_masks, labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2614: FutureWarning: The `pa
  warnings.warn(
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```python
# Prediction on test set

print('Predicting labels for {:,} test sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
for batch in prediction_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 prediction
  with torch.no_grad():
      # Forward pass, calculate logit predictions.
      result = model(b_input_ids,
                     token_type_ids=None,
                     attention_mask=b_input_mask,
                     return_dict=True)

  logits = result.logits

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

  # Store predictions and true labels
  predictions.append(logits)
  true_labels.append(label_ids)

print('    DONE.')
```

```
    Predicting labels for 277 test sentences...
        DONE.
```

```python
# Flatten the predictions and true labels
flat_predictions = np.concatenate(predictions, axis=0)
flat_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
flat_predictions = np.argmax(flat_predictions, axis=1).flatten()

# Calculate accuracy
accuracy = np.sum(flat_predictions == flat_true_labels) / len(flat_true_labels)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
    Test Accuracy: 63.54%
```

```python
# Load the dataset into a pandas dataframe.
df = pd.read_csv("/content/xnli_de_val.csv")

# Report the number of sentences.
print('Number of testing sentences: {:,}\n'.format(df.shape[0]))

# Display 10 random rows from the data.
df.sample(10)
```

```python
sentences1 = df['sentence1'].values
sentences2 = df['sentence2'].values
labels = df['label'].values
```

```python
max_len = 0
input_ids = []
attention_masks = []

# For every sentence pair...
for sent1, sent2 in zip(sentences1, sentences2):
    # `encode_plus` will:
    #   (1) Tokenize both sentences and add `[CLS]` and `[SEP]` tokens.
    #   (2) Pad or truncate the sentence pair to `max_length`
    #   (3) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        sent1, sent2,
                        add_special_tokens=True, # Add '[CLS]' and '[SEP]'
                        max_length=64,           # Pad & truncate all sentence pairs.
                        pad_to_max_length=True,
                        return_attention_mask=True,   # Construct attn. masks.
                        return_tensors='pt',     # Return pytorch tensors.
                        truncation=True ,
                   )


    # Add the encoded sentence pair to the list.
    input_ids.append(encoded_dict['input_ids'])
    attention_masks.append(encoded_dict['attention_mask'])

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(labels)
```

```python
batch_size = 16

# Create the DataLoader.
prediction_data = TensorDataset(input_ids, attention_masks, labels)
prediction_sampler = SequentialSampler(prediction_data)
prediction_dataloader = DataLoader(prediction_data, sampler=prediction_sampler, batch_size=batch_size)
```

```
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
Be aware, overflowing tokens are not returned for the setting you have chosen, i.e. sequence pairs with the
```

```
# Prediction on test set

print('Predicting labels for {:,} test sentences...'.format(len(input_ids)))

# Put model in evaluation mode
model.eval()

# Tracking variables
predictions , true_labels = [], []

# Predict
# Flatten the predictions and true labels
flat_predictions = np.concatenate(predictions, axis=0)
flat_true_labels = np.concatenate(true_labels, axis=0)

# Convert logits to predicted labels
flat_predictions = np.argmax(flat_predictions, axis=1).flatten()

# Calculate accuracy
accuracy = np.sum(flat_predictions == flat_true_labels) / len(flat_true_labels)
print("Test Accuracy: {:.2f}%".format(accuracy * 100))
```

```
    Test Accuracy: 36.83%
```

HENCE, the zero-shot performance using multilingual model improve over the English only BERT model

Double-click (or enter) to edit

Double-click (or enter) to edit

```
    true_labels.append(label_ids)
```
Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit