```
# original code from https://github.com/ensembles4612/medical_intent_detector_using_BERT

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 on

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
```

```
!pip install transformers
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting transformers
  Downloading transformers-4.25.1-py3-none-any.whl (5.8 MB)
     |████████████████████████████████| 5.8 MB 24.0 MB/s
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (21.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (1.21.6)
Collecting huggingface-hub<1.0,>=0.10.0
  Downloading huggingface_hub-0.11.1-py3-none-any.whl (182 kB)
     |████████████████████████████████| 182 kB 76.7 MB/s
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (7.6 MB)
     |████████████████████████████████| 7.6 MB 65.9 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.8.0)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (6.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers) (2.23.0)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.0,>=0.10.0->
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>=20.0->transformers) (
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2022.9.24)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->transfo
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.11.1 tokenizers-0.13.2 transformers-4.25.1
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
data = pd.read_csv('gdrive/My Drive/bert/dog symptom feed - v2.csv')
data1 = data[['phrase','prompt']]
data1.sample(5)
```

	phrase	prompt
5187	When My dog eat too much sugar my body gets we...	Body feels weak
3338	My son nicked his neck with an old razor and t...	Infected wound
3401	My head hurts and My dog lose the sensation in...	Head ache
2527	My hair is falling out after My dog take a sho...	Hair falling out
605	My dog have a cut on my foot that became infec...	Infected wound

```
df=data1.copy()
df.isna().sum()
```

```
phrase    0
prompt    0
dtype: int64
```

```
df['prompt'].value_counts()
```

```
Acne                  328
Shoulder pain         320
Joint pain            318
Infected wound        306
Knee pain             305
Cough                 293
Feeling dizzy         283
Muscle pain           282
Heart hurts           273
Ear ache              270
Hair falling out      264
Head ache             263
Feeling cold          263
Skin issue            262
Stomach ache          261
Back pain             259
Neck pain             251
Internal pain         248
Blurry vision         246
Body feels weak       241
Hard to breath        233
Emotional pain        231
Injury from sports    230
Foot ache             223
Open wound            208
Name: prompt, dtype: int64
```

```
print('Total number of intents: %d'%(len(df['prompt'].value_counts().index)))
```

```
Total number of intents: 25
```

```
from sklearn.model_selection import train_test_split

X, sentence_test, y, intent_test = train_test_split(df.phrase, df.prompt, stratify = df.prompt,test_size=0.2, random_state=4612)
sentence_train, sentence_val, intent_train, intent_val = train_test_split(X, y, stratify = y,test_size=0.125, random_state=4612)
```

```
print(f"#examples in training set:{ sentence_train.shape[0]}\n#examples in validation set:{ sentence_val.shape[0]}\n#examples in test set:{ s
```

```
#examples in training set:4662
#examples in validation set:666
#examples in test set:1333
```

```
# Defining some key variables that will be used later on in the training
TRAIN_BATCH_SIZE = 32
VALID_BATCH_SIZE = 64
EPSILON = 1e-08
EPOCHS = 4
LEARNING_RATE = 2e-5
SEED = 1215
from transformers import BertTokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
Downloading: 100%                                         232k/232k [00:00<00:00, 843kB/s]

Downloading: 100%                                         28.0/28.0 [00:00<00:00, 352B/s]

Downloading: 100%                                         570/570 [00:00<00:00, 6.32kB/s]
```

```
max_len = 0
input = []
length=[]
# For every sentence...
for sent in sentence_train:

    # Tokenize the text and add special tokens--`[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)
    input.append(input_ids)
    length.append(len(input_ids))
    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))
    mean_len = sum(length)/len(length)
#39 tokens is the maximum number of tokens in a sentence (transcription). Also, a sentence has 14 tokens on average.
print('Max sentence length:%d \nMean sentence length:%d' % (max_len,mean_len))
```

```
     Max sentence length:40
     Mean sentence length:15
```

```
# create a function to tokenize sentences.
def tokenize(sentence):
  batch = tokenizer(list(sentence),
                  #is_pretokenized = False,
                  #Pad or truncate all sentences to the same length. Create the attention masks which explicitly differentiate real tokens fr
                  padding = True,
                  truncation = True,
                  return_tensors="pt")
  return batch
```

```
tok_train = tokenize(sentence_train)
tok_val = tokenize(sentence_val)
tok_test = tokenize(sentence_test)
```

```
from sklearn.preprocessing import LabelEncoder
# encode "intent" to 25 number labels
LE = LabelEncoder()
label_train = torch.tensor((LE.fit_transform(intent_train)))
label_val = torch.tensor((LE.fit_transform(intent_val)))
label_test = torch.tensor((LE.fit_transform(intent_test)))
```

```
from torch.utils.data import TensorDataset

train_dataset = TensorDataset(tok_train['input_ids'], tok_train['attention_mask'],label_train)
validation_dataset = TensorDataset(tok_val['input_ids'], tok_val['attention_mask'],label_val)
test_dataset = TensorDataset(tok_test['input_ids'], tok_test['attention_mask'],label_test)
```

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

# 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 = TRAIN_BATCH_SIZE # Trains with this batch size.
        )

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

test_dataloader = DataLoader(
            validation_dataset,
            sampler = SequentialSampler(validation_dataset),
            batch_size = VALID_BATCH_SIZE
```

```
from transformers import BertForSequenceClassification, AdamW, BertConfig

## use pretained base(relatively small) BERT mdoel for sequence classification
#CUDA_LAUNCH_BLOCKING=1
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels = 25)
model.cuda() # make pytorch run this model on GPU.

## use AdamW optimizer
optimizer = AdamW(model.parameters(),
                  lr = LEARNING_RATE,
                  eps = EPSILON) #very small number to prevent any division by zero

from transformers import get_linear_schedule_with_warmup

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

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

```
Downloading: 100%                                        440M/440M [00:05<00:00, 75.6MB/s]

Some weights of the model checkpoint at bert-base-uncased were not used when initializi
- This IS expected if you are initializing BertForSequenceClassification from the checkp
- This IS NOT expected if you are initializing BertForSequenceClassification from the ch
Some weights of BertForSequenceClassification were not initialized from the model checkp
You should probably TRAIN this model on a down-stream task to be able to use it for pred
/usr/local/lib/python3.8/dist-packages/transformers/optimization.py:306: FutureWarning:
  warnings.warn(
```

```
# Function to calcuate the accuracy of the model

def calcuate_accu(big_idx, targets):
    n_correct = (big_idx==targets).sum().item()
    return n_correct


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))


from torch.utils.tensorboard import SummaryWriter

# default `log_dir` is "runs" - we'll be more specific here
writer = SummaryWriter('runs/Tensorboard')


# Start the training process:
import random
import torch

random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
torch.cuda.manual_seed_all(SEED)
def train(epochs):
  total_t0 = time.time() # Measure the total training time for the whole run.
  tr_loss = 0
  n_correct = 0
  nb_tr_steps = 0
  nb_tr_examples = 0

  # For each epoch...
  for epoch in range(0, epochs):
      print('======== Epoch {:} / {:} ========'.format(epoch + 1, epochs))
      print('Training...')
      t0 = time.time()     # Measure how long the training epoch takes.
      total_tr_loss = 0
      total_n_correct = 0
      total_nb_tr_examples = 0
      model.train()    # Put the model into training mode

      # For each batch of training data...
      for step, batch in enumerate(train_dataloader, 0):
          # 'batch' contains three pytorch tensors:[0]: input ids, [1]: attention masks, [2]: labels
          input_ids = batch[0].to(device, dtype = torch.long)
          input_mask = batch[1].to(device, dtype = torch.long)
          labels = batch[2].to(device, dtype = torch.long)

          model.zero_grad()       #clear any previously calculated gradients

          outputs = model(input_ids, token_type_ids=None, attention_mask=input_mask)
          loss_function = torch.nn.CrossEntropyLoss()
          loss = loss_function(outputs[0], labels) #`loss` is a Tensor containing a single value
          tr_loss += loss.item() #.item()` function just returns the Python value from the tensor
          total_tr_loss += loss.item()
          big_val, big_idx = torch.max(outputs[0], dim=1)
          n_correct += calcuate_accu(big_idx, labels)
          total_n_correct += calcuate_accu(big_idx, labels)
          nb_tr_steps += 1
          nb_tr_examples+=labels.size(0)
          total_nb_tr_examples+=labels.size(0)

          if step % 20==19:
              loss_step = tr_loss/nb_tr_steps
              accu_step = n_correct/nb_tr_examples # #correct examples/all examples
              print(f"Training Loss per 20 steps(batches): {loss_step}")
              print(f"Training Accuracy per 20 steps(batches): {accu_step}")
              elapsed = format_time(time.time() - t0)    # Calculate elapsed time in minutes.
              # Report progress.
              print('Batch {} of {}.  Elapsed: {:}.'.format(step+1, len(train_dataloader), elapsed))
              #writer.add_scalar('training loss', loss_step, (epoch +1)*len(trainloader) )
              tr_loss = 0;n_correct = 0;nb_tr_steps = 0;nb_tr_examples = 0

          loss.backward() # Perform a backward pass to calculate the gradients.
          torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0) # Clip the norm of the gradients to 1.0. This is to help prevent the "explo
          optimizer.step()
          scheduler.step() # Update the learning rate.

    # Calculate the average loss over all of the batches.
      train_loss_per_epoch = total_tr_loss / len(train_dataloader)
      train_accuracy_per_epoch=total_n_correct/total_nb_tr_examples
      # Measure how long this epoch took.
      training_time = format_time(time.time() - t0)

      print("")
      print("training loss per epoch: {0:.2f}".format(train_loss_per_epoch))
      print("training accuracy per epoch: {0:.2f}".format(train_accuracy_per_epoch))
      print("Training 1 epcoh took: {:}".format(training_time))

train(epochs = EPOCHS)
```

```
    Training Accuracy per 20 steps(batches): 0.984375
    Batch 120 of 146.  Elapsed: 0:00:26.
    Training Loss per 20 steps(batches): 0.3349503383040428
    Training Accuracy per 20 steps(batches): 0.9921875
    Batch 140 of 146.  Elapsed: 0:00:30.

    training loss per epoch: 0.45
    training accuracy per epoch: 0.98
    Training 1 epcoh took: 0:00:32
    ======== Epoch 4 / 4 ========
    Training...
    Training Loss per 20 steps(batches): 0.3103522383249723
    Training Accuracy per 20 steps(batches): 0.9927007299270073
    Batch 20 of 146.  Elapsed: 0:00:04.
    Training Loss per 20 steps(batches): 0.274543996155262
    Training Accuracy per 20 steps(batches): 0.99375
    Batch 40 of 146.  Elapsed: 0:00:09.
    Training Loss per 20 steps(batches): 0.2482662871479988
    Training Accuracy per 20 steps(batches): 0.996875
    Batch 60 of 146.  Elapsed: 0:00:13.
    Training Loss per 20 steps(batches): 0.23863801509141921
    Training Accuracy per 20 steps(batches): 0.9921875
    Batch 80 of 146.  Elapsed: 0:00:18.
    Training Loss per 20 steps(batches): 0.24163946211338044
    Training Accuracy per 20 steps(batches): 0.990625
    Batch 100 of 146.  Elapsed: 0:00:22.
    Training Loss per 20 steps(batches): 0.2324072815477848
    Training Accuracy per 20 steps(batches): 0.996875
    Batch 120 of 146.  Elapsed: 0:00:26.
    Training Loss per 20 steps(batches): 0.22953578904271127
    Training Accuracy per 20 steps(batches): 0.9984375
    Batch 140 of 146.  Elapsed: 0:00:31.

    training loss per epoch: 0.25
    training accuracy per epoch: 0.99
    Training 1 epcoh took: 0:00:32
```

```
# test the model on the validation set
def valid(model, validation_loader):
  model.eval()
  val_loss = 0
  nb_val_examples = 0
  n_correct = 0
  with torch.no_grad():
    for _, data in enumerate(validation_loader, 0):
      ids = data[0].to(device, dtype = torch.long)
      mask = data[1].to(device, dtype = torch.long)
      targets = data[2].to(device, dtype = torch.long)
      outputs = model(ids, mask)
      loss_function = torch.nn.CrossEntropyLoss()
      loss = loss_function(outputs[0], targets)
      val_loss += loss.item()
      big_val, big_idx = torch.max(outputs[0], dim=1)
      n_correct += calcuate_accu(big_idx, targets)
      nb_val_examples+=targets.size(0)

  val_ave_loss = val_loss/len(validation_loader)
  val_accu = (n_correct*100)/nb_val_examples
  print("Loss on validation/test data: %0.2f" % val_ave_loss)
  print("Accuracy on validation/test data: %0.2f%%" % val_accu)

  return
```

```
valid(model, validation_dataloader)
```

```
    Loss on validation/test data: 0.18
    Accuracy on validation/test data: 99.40%
```

```
valid(model, test_dataloader)
```

```
    Loss on validation/test data: 0.18
    Accuracy on validation/test data: 99.40%
```

```
import os

# Saving best-practices: if you use defaults names for the model, you can reload it using from_pretrained()

output_dir = './Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/'
```

```
# Create output directory if needed
if not os.path.exists(output_dir):
  os.makedirs(output_dir)

print("Saving model to %s" % output_dir)

# Save a trained model, configuration and tokenizer using `save_pretrained()`.
# They can then be reloaded using `from_pretrained()`
model_to_save = model.module if hasattr(model, 'module') else model  # Take care of distributed/parallel training
model_to_save.save_pretrained(output_dir)
tokenizer.save_pretrained(output_dir)
```

```
Saving model to ./Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/
('./Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/tokenizer_config.json',
 './Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/special_tokens_map.json',
 './Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/vocab.txt',
 './Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/added_tokens.json')
```

```
df_label = pd.DataFrame(tuple(zip(range(25),LE.classes_)), columns=['id','intent'])
df_label.to_pickle('./Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/df_label.pkl')
```

```
# Copy the model files to a directory in Google Drive.
!cp -r ./Documents/intent_detection_healthcare_bert/saved_bert_model_and_tokenizer/ "gdrive/My Drive/bert/"
```

```
#### load the model and build the detector for deployment
!pip install transformers
import pandas as pd
from transformers import BertTokenizer, BertForSequenceClassification

input_dir = 'gdrive/My Drive/bert/saved_bert_model_and_tokenizer/'

loaded_model = BertForSequenceClassification.from_pretrained(input_dir)
loaded_model.eval()
loaded_tokenizer = BertTokenizer.from_pretrained(input_dir)
loaded_df_label = pd.read_pickle('gdrive/My Drive/bert/saved_bert_model_and_tokenizer/df_label.pkl')
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-packages (4.25.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.8.0)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (6.0)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.13.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers) (2.23.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (21.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (1.21.6)
Requirement already satisfied: huggingface-hub<1.0,>=0.10.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.11.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.0,>=0.10.0->
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>=20.0->transformers) (
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2022.9.24)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->transfo
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (3.0.4)
```

```
# test the model on an unseen example

def medical_symptom_detector(intent):

  pt_batch = loaded_tokenizer(
  intent,
  padding=True,
  truncation=True,
  return_tensors="pt")

  pt_outputs = loaded_model(**pt_batch)
  __, id = torch.max(pt_outputs[0], dim=1)
  prediction = loaded_df_label.iloc[[id.item()]]['intent'].item()
  print('You may have a medical condition: %s.'%(prediction))
  return


input = 'My dog is inactive and does not eat. He is also breathing bad.'
medical_symptom_detector(input)
```

```
You may have a medical condition: Hard to breath.
```

Colab paid products - Cancel contracts here

✓ 0s completed at 11:55 AM