XLNET_TWEETS

```
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')
```

```
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_c

    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")
```

1.2. Installing the Hugging Face Library

```
!pip install transformers
```

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

2. Loading Dataset

```
# from google.colab import files
# uploaded = files.upload()
```

```
import pandas as pd

# Load the dataset into a pandas dataframe.
```

```
df = pd.read_csv("Final_data.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)
```

```
df.drop(columns=['Unnamed: 0'],axis=1,inplace=True)
```

```
sentences  = []
for sentence in df['Tweets']:
  sentence = sentence+"[SEP] [CLS]"
  sentences.append(sentence)
```

```
sentences[0]
```

```
labels=df['Analysis'].values
```

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

IMPORTING DEPENDENCIES

```
import transformers
from transformers import XLNetTokenizer, XLNetModel, AdamW, get_
import torch
```

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

from torch import nn, optim
from keras.preprocessing.sequence import pad_sequences
from torch.utils.data import TensorDataset,RandomSampler,Sequent
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
```

3. Tokenization & Input Formatting

XLNET Tokenizer

```
from transformers import XLNetTokenizer

# Load the XLNet tokenizer.
```

```
print('Loading XLNet tokenizer...')
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased',
                                            do_lower_case=True)
```

```
# Tokenize all of the sentences and map the tokens to thier word
tokenized_text = [tokenizer.tokenize(sent) for sent in sentences
ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_tex
```

maximum length of our sentences

```
print('Max sentence length: ', max([len(sen) for sen in ids]))
MAX_LEN=155
```

```
# We'll borrow the `pad_sequences` utility function to do this.
from keras.preprocessing.sequence import pad_sequences
MAX_LEN = 170
```

```
print('\nPadding/truncating all sentences to %d values...' % MAX

print('\nPadding token: "{:}", ID: {:}'.format(tokenizer.pad_tok

input_ids2 = pad_sequences(ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="p

print('\nDone.')
```

```
# Use train_test_split to split our data into train and validati
# training
from sklearn.model_selection import train_test_split

# Use 90% for training and 10% for validation.
xtrain,xtest,ytrain,ytest= train_test_split(input_ids2, labels,
                                                     random_state
                                                    test_size=0.1
```

```
# Convert all inputs and labels into torch tensors, the required
# for our model.
Xtrain = torch.tensor(xtrain)
Ytrain = torch.tensor(ytrain)
Xtest = torch.tensor(xtest)
Ytest = torch.tensor(ytest)
```

```
from torch.utils.data import TensorDataset, DataLoader, RandomSa

# The DataLoader needs to know our batch size for training, so w
```

```
# here.
# For fine-tuning XLNET on a specific task, the authors recommen
# 48.

batch_size = 4

# Create the DataLoader for our training set.
train_data = TensorDataset(Xtrain,Ytrain)
loader = DataLoader(train_data,batch_size=batch_size)

# Create the DataLoader for our test set.
test_data = TensorDataset(Xtest,Ytest)
test_loader = DataLoader(test_data,batch_size=batch_size)
```

```
from transformers import XLNetForSequenceClassification, AdamW,

# Load BertForSequenceClassification, the pretrained BERT model
# linear classification layer on top.
model = XLNetForSequenceClassification.from_pretrained(
    "xlnet-base-cased", # Use the 12-layer BERT model, with an u
    num_labels = 2, # The number of output labels--2 for binary
                    # You can increase this for multi-class task

)

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

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

```
# Note: AdamW is a class from the huggingface library (as oppose
# I believe the 'W' stands for 'Weight Decay fix"
optimizer = AdamW(model.parameters(),
```

```
                    lr = 2e-5, # args.learning_rate - default is 5
                    eps = 1e-8 # args.adam_epsilon  - default is 1
                  )
```

Training the model

```
import torch.nn as nn
criterion = nn.CrossEntropyLoss()
```

```
import numpy as np
def flat_accuracy(preds,labels):  # A function to predict Accura
  correct=0
  for i in range(0,len(labels)):
    if(preds[i]==labels[i]):
      correct+=1
  return (correct/len(labels))*100
```

```
no_train=0
epochs = 1
for epoch in range(epochs):
```

```
  print("TRAINING EPOCH ",epoch)
  model.train()
  loss1 = []
  steps = 0
  train_loss = []
  l = []
  for inputs,labels1 in loader :
    inputs.to(device)
    labels1.to(device)
    optimizer.zero_grad()
    outputs = model(inputs.to(device))
    loss = criterion(outputs[0],labels1.to(device)).to(device)
    # logits = outputs[1]
    #ll=outp(loss)
    [train_loss.append(p.item()) for p in torch.argmax(outputs[0
    [l.append(z.item()) for z in labels1]# real labels
    loss.backward()
    optimizer.step()
    loss1.append(loss.item())
    no_train += inputs.size(0)
    steps += 1
  print("Current Loss is : {} Step is : {} number of Example : {
```

```
Requirement already satisfied: transformers in /usr/local/l
Requirement already satisfied: dataclasses; python_version
Requirement already satisfied: requests in /usr/local/lib/p
```

XLNET_TWEETS

```
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')
```

```
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_c

    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")
```

1.2. Installing the Hugging Face Library

```
!pip install transformers
```

2. Loading Dataset

```
# from google.colab import files
# uploaded = files.upload()
```

```
import pandas as pd

# Load the dataset into a pandas dataframe.
df = pd.read_csv("Final_data.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)
```

```
df.drop(columns=['Unnamed: 0'],axis=1,inplace=True)
```

```
sentences  = []
for sentence in df['Tweets']:
  sentence = sentence+"[SEP] [CLS]"
```

```
    sentences.append(sentence)
```

```
sentences[0]
```

```
labels=df['Analysis'].values
```

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

IMPORTING DEPENDENCIES

```
import transformers
from transformers import XLNetTokenizer, XLNetModel, AdamW, get_
import torch
```

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

from torch import nn, optim
from keras.preprocessing.sequence import pad_sequences
from torch.utils.data import TensorDataset,RandomSampler,Sequent
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
```

3. Tokenization & Input Formatting

XLNET Tokenizer

```
from transformers import XLNetTokenizer

# Load the XLNet tokenizer.
```

```
print('Loading XLNet tokenizer...')
tokenizer = XLNetTokenizer.from_pretrained('xlnet-base-cased',
                                            do_lower_case=True)
```

```
# Tokenize all of the sentences and map the tokens to thier word
tokenized_text = [tokenizer.tokenize(sent) for sent in sentences
ids = [tokenizer.convert_tokens_to_ids(x) for x in tokenized_tex
```

maximum length of our sentences

```
print('Max sentence length: ', max([len(sen) for sen in ids]))
MAX_LEN=155
```

```
# We'll borrow the `pad_sequences` utility function to do this.
from keras.preprocessing.sequence import pad_sequences
MAX_LEN = 170
```

```
print('\nPadding/truncating all sentences to %d values...' % MAX

print('\nPadding token: "{:}", ID: {:}'.format(tokenizer.pad_tok

input_ids2 = pad_sequences(ids, maxlen=MAX_LEN, dtype="long",
                          value=0, truncating="post", padding="p

print('\nDone.')
```

```
# Use train_test_split to split our data into train and validati
# training
from sklearn.model_selection import train_test_split

# Use 90% for training and 10% for validation.
xtrain,xtest,ytrain,ytest= train_test_split(input_ids2, labels,
                                                    random_state
                                                   test_size=0.1
```

```
# Convert all inputs and labels into torch tensors, the required
# for our model.
Xtrain = torch.tensor(xtrain)
Ytrain = torch.tensor(ytrain)
Xtest = torch.tensor(xtest)
Ytest = torch.tensor(ytest)
```

```
from torch.utils.data import TensorDataset, DataLoader, RandomSa

# The DataLoader needs to know our batch size for training, so w
```

```
# here.
# For fine-tuning XLNET on a specific task, the authors recommen
# 48.

batch_size = 4

# Create the DataLoader for our training set.
train_data = TensorDataset(Xtrain,Ytrain)
loader = DataLoader(train_data,batch_size=batch_size)

# Create the DataLoader for our test set.
test_data = TensorDataset(Xtest,Ytest)
test_loader = DataLoader(test_data,batch_size=batch_size)
```

```
from transformers import XLNetForSequenceClassification, AdamW,

# Load BertForSequenceClassification, the pretrained BERT model
# linear classification layer on top.
model = XLNetForSequenceClassification.from_pretrained(
    "xlnet-base-cased", # Use the 12-layer BERT model, with an u
    num_labels = 2, # The number of output labels--2 for binary
                    # You can increase this for multi-class task

)

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

```
# Note: AdamW is a class from the huggingface library (as oppose
# I believe the 'W' stands for 'Weight Decay fix"
optimizer = AdamW(model.parameters(),
```

```
                  lr = 2e-5, # args.learning_rate - default is 5
                  eps = 1e-8 # args.adam_epsilon  - default is 1
                )
```

Training the model

```
import torch.nn as nn
criterion = nn.CrossEntropyLoss()
```

```
import numpy as np
def flat_accuracy(preds,labels):  # A function to predict Accura
  correct=0
  for i in range(0,len(labels)):
    if(preds[i]==labels[i]):
      correct+=1
  return (correct/len(labels))*100
```

```
no_train=0
epochs = 1
for epoch in range(epochs):
```

```
print("TRAINING EPOCH ",epoch)
model.train()
loss1 = []
steps = 0
train_loss = []
l = []
for inputs,labels1 in loader :
  inputs.to(device)
  labels1.to(device)
  optimizer.zero_grad()
  outputs = model(inputs.to(device))
  loss = criterion(outputs[0],labels1.to(device)).to(device)
  # logits = outputs[1]
  #ll=outp(loss)
  [train_loss.append(p.item()) for p in torch.argmax(outputs[0
  [l.append(z.item()) for z in labels1]# real labels
  loss.backward()
  optimizer.step()
  loss1.append(loss.item())
  no_train += inputs.size(0)
  steps += 1
print("Current Loss is : {} Step is : {} number of Example : {
```