Loading all the Libraries

In [ ]:

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

```
Mounted at /content/drive
```

In [ ]:

```
!pip install transformers==3.5.1
```

```
Collecting transformers==3.5.1
  Downloading
https://files.pythonhosted.org/packages/3a/83/e74092e7f24a08d751aa59b37a9fc572b2e4af3918cb66f7766c3
1b4/transformers-3.5.1-py3-none-any.whl (1.3MB)
     |████████████████████████████████| 1.3MB 7.4MB/s
Requirement already satisfied: dataclasses; python_version < "3.7" in
/usr/local/lib/python3.6/dist-packages (from transformers==3.5.1) (0.8)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (2019.12.20)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (4.41.1)
Requirement already satisfied: protobuf in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (3.12.4)
Collecting tokenizers==0.9.3
  Downloading
https://files.pythonhosted.org/packages/4c/34/b39eb9994bc3c999270b69c9eea40ecc6f0e97991dba28282b9fc
4ee/tokenizers-0.9.3-cp36-cp36m-manylinux1_x86_64.whl (2.9MB)
     |████████████████████████████████| 2.9MB 24.5MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (20.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (1.18.5)
Requirement already satisfied: filelock in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (3.0.12)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from
transformers==3.5.1) (2.23.0)
Collecting sentencepiece==0.1.91
  Downloading
https://files.pythonhosted.org/packages/d4/a4/d0a884c4300004a78cca907a6ff9a5e9fe4f090f5d95ab341c53c
c58/sentencepiece-0.1.91-cp36-cp36m-manylinux1_x86_64.whl (1.1MB)
     |████████████████████████████████| 1.1MB 42.3MB/s
Collecting sacremoses
  Downloading
https://files.pythonhosted.org/packages/7d/34/09d19aff26edcc8eb2a01bed8e98f13a1537005d31e95233fd482
d10/sacremoses-0.0.43.tar.gz (883kB)
     |████████████████████████████████| 890kB 45.7MB/s
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-packages (from
protobuf->transformers==3.5.1) (50.3.2)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.6/dist-packages (from protobuf->
transformers==3.5.1) (1.15.0)
Requirement already satisfied: pyparsing>=2.0.2 in /usr/local/lib/python3.6/dist-packages (from
packaging->transformers==3.5.1) (2.4.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from
requests->transformers==3.5.1) (2020.11.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.6/dist-packages (from
requests->transformers==3.5.1) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests->transformers==3.5.1) (1.24.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from
requests->transformers==3.5.1) (3.0.4)
Requirement already satisfied: click in /usr/local/lib/python3.6/dist-packages (from sacremoses-
>transformers==3.5.1) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.6/dist-packages (from sacremoses->
transformers==3.5.1) (0.17.0)
Building wheels for collected packages: sacremoses
  Building wheel for sacremoses (setup.py) ... done
  Created wheel for sacremoses: filename=sacremoses-0.0.43-cp36-none-any.whl size=893257
```

```
sha256=fe738906e00c5c2ace1e047302f8ac56519bbb0ec6381086b50e3026b4e80f9c
  Stored in directory:
/root/.cache/pip/wheels/29/3c/fd/7ce5c3f0666dab31a50123635e6fb5e19ceb42ce38d4e58f45
Successfully built sacremoses
Installing collected packages: tokenizers, sentencepiece, sacremoses, transformers
Successfully installed sacremoses-0.0.43 sentencepiece-0.1.91 tokenizers-0.9.3 transformers-3.5.1
```

In [ ]:

```
import pandas as pd
import numpy as np
import re
import math
import random
import tqdm
import pickle
import random
import time
from datetime import datetime
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

#Torch
import torch
import torch.nn as nn
from torch.utils import data
from torch.utils.data import DataLoader
from torchsummary import summary
from torch.utils.data import random_split
from torch.utils.data import TensorDataset
from torch.utils.data import RandomSampler
from torch.utils.data import SequentialSampler
from torch.optim.lr_scheduler import CosineAnnealingLR
from torch.optim.lr_scheduler import ReduceLROnPlateau

#Transformers
from transformers import get_linear_schedule_with_warmup
from transformers import BertForSequenceClassification, AdamW, BertConfig
from transformers import BertTokenizer

#Data
from sklearn.datasets import fetch_20newsgroups
```

Task 1: Classification using BERT or BERT variations

Preparing the Data

In [ ]:

```
#Train and test dataset
train = fetch_20newsgroups(subset='train', remove=('headers', 'footers', 'quotes'))
test = fetch_20newsgroups(subset='test', remove=('headers', 'footers', 'quotes'))
```

```
Downloading 20news dataset. This may take a few minutes.
Downloading dataset from https://ndownloader.figshare.com/files/5975967 (14 MB)
```

In [ ]:

```
params = dict(
  seed_val = 1234,
  batch_size = 16,
  path = r'/content/drive/MyDrive')
```

In [ ]:

```
#Train
df_train = pd.DataFrame()
df_train['data'] = train.data
df_train['labels'] = train.target
```

```
#Test
df_test = pd.DataFrame()
df_test['data'] = test.data
df_test['labels'] = test.target
#df_train.shape, df_test.shape
```

In [ ]:

```
#Train
sentences_train = df_train.data.values
labels_train = df_train.labels.values
#Test
sentences_test = df_test.data.values
labels_test = df_test.labels.values
```

In [ ]:

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

In [ ]:

```
# Print the original sentence.
print(' Original: ', sentences_train[0])
# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(sentences_train[0]))
# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(sentences_train[0])))
```

```
 Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Tokenized:  ['i', 'was', 'wondering', 'if', 'anyone', 'out', 'there', 'could', 'en', '##light', '#
#en', 'me', 'on', 'this', 'car', 'i', 'saw', 'the', 'other', 'day', '.', 'it', 'was', 'a', '2', '-
', 'door', 'sports', 'car', ',', 'looked', 'to', 'be', 'from', 'the', 'late', '60s', '/', 'early',
'70s', '.', 'it', 'was', 'called', 'a', 'brick', '##lin', '.', 'the', 'doors', 'were', 'really', '
small', '.', 'in', 'addition', ',', 'the', 'front', 'bumper', 'was', 'separate', 'from', 'the', 'r
est', 'of', 'the', 'body', '.', 'this', 'is', 'all', 'i', 'know', '.', 'if', 'anyone', 'can',
'tell', '##me', 'a', 'model', 'name', ',', 'engine', 'spec', '##s', ',', 'years', 'of',
'production', ',', 'where', 'this', 'car', 'is', 'made', ',', 'history', ',', 'or', 'whatever', 'i
nfo', 'you', 'have', 'on', 'this', 'funky', 'looking', 'car', ',', 'please', 'e', '-', 'mail', '.'
]
Token IDs:  [1045, 2001, 6603, 2065, 3087, 2041, 2045, 2071, 4372, 7138, 2368, 2033, 2006, 2023, 24
82, 1045, 2387, 1996, 2060, 2154, 1012, 2009, 2001, 1037, 1016, 1011, 2341, 2998, 2482, 1010, 2246,
2000, 2022, 2013, 1996, 2397, 20341, 1013, 2220, 17549, 1012, 2009, 2001, 2170, 1037, 5318, 4115, 1
012, 1996, 4303, 2020, 2428, 2235, 1012, 1999, 2804, 1010, 1996, 2392, 21519, 2001, 3584, 2013, 199
6, 2717, 1997, 1996, 2303, 1012, 2023, 2003, 2035, 1045, 2113, 1012, 2065, 3087, 2064, 2425, 4168,
1037, 2944, 2171, 1010, 3194, 28699, 2015, 1010, 2086, 1997, 2537, 1010, 2073, 2023, 2482, 2003, 20
81, 1010, 2381, 1010, 2030, 3649, 18558, 2017, 2031, 2006, 2023, 24151, 2559, 2482, 1010, 3531, 104
1, 1011, 5653, 1012]
```

All the necessary Functions

In [ ]:

```
#Tokenizer and preprcessing
def tokenizer_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      encoded_dict = tokenizer.encode_plus(
                          sent,
                          add_special_tokens = True,
                          max_length = max_len,
                          pad_to_max_length = True,
                          return_attention_mask = True,
```

```
                        return_tensors = 'pt',
                   )

      # 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)

  return input_ids, attention_masks
```

In [ ]:

```
#data loaders function

def loaders(sentences_train, labels_train, sentences_test, labels_test, max_len, batch_size):
  input_ids_train, attention_masks_train, input_ids_test, attention_masks_test, labels_train, label
s_test = sentences_t(sentences_train, labels_train, sentences_test, labels_test, max_len)

  train_dataset = TensorDataset(input_ids_train, attention_masks_train, labels_train)
  test_dataset = TensorDataset(input_ids_test, attention_masks_test, labels_test)

  train_dataloader = DataLoader(
              train_dataset,
              sampler = RandomSampler(train_dataset),
              batch_size = batch_size
          )

  validation_dataloader = DataLoader(
              test_dataset,
              sampler = SequentialSampler(test_dataset),
              batch_size = batch_size
          )
  return train_dataloader, validation_dataloader
```

In [ ]:

```
#Sentences to Tokens conversion
def sentences_t(sentences_train, labels_train, sentences_test, labels_test, max_len):
  input_ids_train, attention_masks_train = tokenizer_s(sentences_train, max_len)
  labels_train = torch.tensor(labels_train)

  input_ids_test, attention_masks_test = tokenizer_s(sentences_test, max_len)
  labels_test = torch.tensor(labels_test)

  # Print sentence 0, now as a list of IDs.
  print('Original: ', sentences_train[0])
  print('Token IDs:', input_ids_train[0].shape)

  return input_ids_train, attention_masks_train, input_ids_test, attention_masks_test, labels_train
, labels_test
```

In [ ]:

```
#Train loop
def train_loop(model, train_dataloader, validation_dataloader, optimizer, scheduler, epochs, seed,
device):

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

  train_losses= np.zeros(epochs)
  valid_losses= np.zeros(epochs)

  for epoch in range(epochs):

    t0= datetime.now()
    train_loss=[]
```

```
    train_loss=[]

    model.train()
    for batch in train_dataloader:

      b_input_ids = batch[0].to(device)
      b_input_mask = batch[1].to(device)
      b_labels = batch[2].to(device)
      # forward pass

      loss, logits = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels)

      # set gradients to zero
      optimizer.zero_grad()

      # backward pass
      loss.backward()
      torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
      optimizer.step()
      scheduler.step()
      train_loss.append(loss.item())

    train_loss=np.mean(train_loss)

    valid_loss=[]
    model.eval()
    with torch.no_grad():
      for batch in validation_dataloader:

        # forward pass
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)

        loss, logits = model(b_input_ids,
                             token_type_ids=None,
                             attention_mask=b_input_mask,
                             labels=b_labels)

        valid_loss.append(loss.item())

      valid_loss=np.mean(valid_loss)

    # save Losses
    train_losses[epoch]= train_loss
    valid_losses[epoch]= valid_loss
    dt= datetime.now()-t0
    print(f'Epoch {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}    Valid Loss: {valid_loss:.4f},
Duration: {dt}')

  return train_losses, valid_losses
```

In [ ]:

```
# Accuracy- write a function to get accuracy
# use this function to get accuracy and print accuracy
def get_accuracy(data_iter, model, device):
  model.eval()
  with torch.no_grad():
    correct =0
    total =0

    for batch in data_iter:

      b_input_ids = batch[0].to(device)
      b_input_mask = batch[1].to(device)
      b_labels = batch[2].to(device)
    # forward pass

      loss, logits = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels)
```

```
      _,indices = torch.max(logits,dim=1)
      correct+= (b_labels==indices).sum().item()
      total += b_labels.shape[0]

    acc= correct/total

    return acc
```

In [ ]:

```
# Write a function to get predictions
def get_predictions(data_iter, model):
  model.eval()
  with torch.no_grad():
    predictions= np.array([])
    y_test= np.array([])

    for batch in data_iter:

      b_input_ids = batch[0].to(device)
      b_input_mask = batch[1].to(device)
      b_labels = batch[2].to(device)
    # forward pass

      loss, logits = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels)

      _,indices = torch.max(logits,dim=1)
      predictions=np.concatenate((predictions,indices.cpu().numpy()))
      y_test = np.concatenate((y_test,b_labels.cpu().numpy()))

  return y_test, predictions
```

In [ ]:

```
def model_parameters(model):
  epochs = 2
  no_decay = ['bias', 'LayerNorm.weight']
  optimizer_grouped_parameters = [
          {'params': [p for n, p in model.named_parameters()
            if not any(nd in n for nd in no_decay)],
          'weight_decay': 0.5},

          {'params': [p for n, p in model.named_parameters()
          if any(nd in n for nd in no_decay)],
          'weight_decay': 0.0}
          ]

  optimizer = AdamW(optimizer_grouped_parameters,
                    lr = 5e-5,
                    eps = 1e-8
                   )

  scheduler = get_linear_schedule_with_warmup(optimizer,
                                              num_warmup_steps = 0,
                                              num_training_steps = len(train_dataloader) * epochs)

  return epochs, optimizer, scheduler
```

In [ ]:

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

EXPERIMENT 1

BERT Model with 512 tokens

In [ ]:

In [ ]:

```
params['max_len'] = 0
for i,sent in enumerate(sentences_train):
    input_ids = tokenizer.encode(sent, add_special_tokens=True)
    params['max_len'] = max(params['max_len'], len(input_ids))
print('Max sentence length: ', params['max_len'])
params['max_len'] = 512
```

```
Token indices sequence length is longer than the specified maximum sequence length for this model
(604 > 512). Running this sequence through the model will result in indexing errors
```

```
Max sentence length:  52886
```

In [ ]:

```
#Train Data Loader
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use
`truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' tru
ncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select
this strategy more precisely by providing a specific strategy to `truncation`.
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([512])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
#Parameters

for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'],  params['scheduler'] = model_parameters(model)
```

In [ ]:

```
model.to(device);
```

In [ ]:

```
#Train Loss and Validation loss
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8773    Valid Loss: 2.7825, Duration: 0:12:14.064195
Epoch 2/2, Train Loss: 2.7270    Valid Loss: 2.6938, Duration: 0:12:19.591873
```

In [ ]:

```
#Getting accuracies
train_acc = get_accuracy(train_dataloader, model,device)
valid_acc = get_accuracy(validation_dataloader, model,device)
print("Train acc: ",train_acc,"Valid acc: ", valid_acc)
```

```
Train acc:  0.25799893936715573 Valid acc:  0.2457514604354753
```

BERT Model with 128 tokens

In [ ]:

```
params['max_len'] = 128
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([128])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
```

```
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
#Training
model.to(device);
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8839    Valid Loss: 2.7787, Duration: 0:02:43.894216
Epoch 2/2, Train Loss: 2.7245    Valid Loss: 2.6867, Duration: 0:02:43.378686
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model,device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print(train_acc, valid_acc)
```

```
0.2647162807141594 0.2624800849707913
```

BERT Model with 140 tokens

In [ ]:

```
params['max_len'] = 140
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([140])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'],  params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8841    Valid Loss: 2.7832, Duration: 0:03:09.593033
Epoch 2/2, Train Loss: 2.7236    Valid Loss: 2.6880, Duration: 0:03:08.959613
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print(train_acc, valid_acc)
```

```
0.2672794767544635 0.2627456186935741
```

BERT Model 4 by truncating from the end

In [ ]:

```
def tokenizer_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      sent = sent.split(" ")
      sent.reverse()
      sent = " ".join(sent)
      encoded_dict = tokenizer.encode_plus(
                          sent,
                          add_special_tokens = True,
                          max_length = max_len,
                          pad_to_max_length = True,
                          return_attention_mask = True,
                          return_tensors = 'pt',
                    )

      # 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)

  return input_ids, attention_masks
```

In [ ]:

```
params['max_len'] = 140
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([140])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
```

```
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'],  params['optimizer'],  params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8720    Valid Loss: 2.7455, Duration: 0:03:09.559264
Epoch 2/2, Train Loss: 2.7120    Valid Loss: 2.6613, Duration: 0:03:08.964845
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train acc:", train_acc, "Valid acc:", valid_acc)
```

```
Train acc: 0.27346650167933534 Valid acc: 0.2705788635156665
```

For Sequence length, 128

In [ ]:

```
params['max_len'] = 128
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
```

```
Token IDs: torch.Size([128])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
#Train valid loss
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8730    Valid Loss: 2.7592, Duration: 0:02:41.486860
Epoch 2/2, Train Loss: 2.7193    Valid Loss: 2.6702, Duration: 0:02:42.259626
```

In [ ]:

```
#Train valid accuracy
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model,device)
print("Train acc:", train_acc,"Valid acc:", valid_acc)
```

```
Train acc: 0.27196393848329503 Valid acc: 0.2586298459904408
```

For Sequence length, 512

In [ ]:

```
params['max_len'] = 512
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([512])
```

In [ ]:

```
#Model
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
#Train valid loss
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8569    Valid Loss: 2.7333, Duration: 0:12:16.321910
Epoch 2/2, Train Loss: 2.6893    Valid Loss: 2.6386, Duration: 0:12:17.209193
```

In [ ]:

```
#train valid accuracy
train_acc = get_accuracy(train_dataloader, model, device)
```

```
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train acc:", train_acc, "Valid acc:",valid_acc)
```

```
Train acc: 0.2864592540215662 Valid acc: 0.2831917153478492
```

BERT Model - Truncating from the mid

In [ ]:

```
def tokenizer_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      sent = sent.split(" ")
      sent.reverse()
      sent = " ".join(sent)
      encoded_dict = tokenizer.encode_plus(
                          sent,
                          add_special_tokens = True,
                          max_length = max_len,
                          pad_to_max_length = True,
                          return_attention_mask = True,
                          return_tensors = 'pt',
                     )

      # 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)

  return input_ids, attention_masks
```

For Sequence length, 140

In [ ]:

```
params['max_len'] = 140
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([140])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
```

```
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'],  params['scheduler'] = model_parameters(model)
```

In [ ]:

```
#Train valid loss
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.8720    Valid Loss: 2.7455, Duration: 0:03:02.680551
Epoch 2/2, Train Loss: 2.7120    Valid Loss: 2.6613, Duration: 0:03:09.193130
```

In [ ]:

```
#Train valid accuracy
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model,device)
print("Train accuracy:",train_acc, "Valid accuracy:",valid_acc)
```

```
Train accuracy: 0.27346650167933534 Valid accuracy: 0.2705788635156665
```

Chunking

In [ ]:

```
def chunking(sentences, labels, max_length):
  final_sentences = []
  final_labels = []
  order = []
  for target_order, (sent, label) in enumerate(zip(sentences, labels)):
    sent = sent.split(" ")
    length = len(sent)
    parts = math.ceil(length/max_length)
    for i in range(1, parts + 1):
      chunk = " ".join(sent[max_length*(i-1) : max_length*i])
      final_sentences.append(chunk)
      final_labels.append(label)
      order.append(target_order)
  # return final_sentences, final_labels, order
```

```
  # return final_sentences, final_labels, order
  return np.array(final_sentences), np.array(final_labels), np.array(order)
```

In [ ]:

```
params['max_length'] = 140
sentences_train_chunked, labels_train_chunked, order_train = chunking(sentences_train,
labels_train, params['max_length'])
sentences_test_chunked, labels_test_chunked, order_test = chunking(sentences_test, labels_test, par
ams['max_length'])
```

In [ ]:

```
#Creating the model
def tokenize_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      sent = sent.split(" ")
      sent = preprocessing(sent, max_len)
      encoded_dict = tokenizer.encode_plus(
                          sent,                      # Sentence to encode.
                          add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                          max_length = max_len,           # 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)

  return input_ids, attention_masks
```

In [ ]:

```
def processing(sent, max_len):
  return sent
```

For sequence length, 140

In [ ]:

```
params['max_len'] = 140
train_dataloader, validation_dataloader = loaders(sentences_train_chunked, labels_train_chunked,
sentences_test_chunked,
                                                  labels_test_chunked, params['max_len'], para
['batch_size'])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use
`truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' tru
ncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select
this strategy more precisely by providing a specific strategy to `truncation`.
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
```

```
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([140])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
    param.requires_grad = False
model.classifier.weight.requires_grad = True
model.classifier.bias.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'],  params['scheduler'] = model_parameters(model)
```

In [ ]:

```
 train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['op
timizer'], params['scheduler'], params['epochs'], 1234, device)
```

```
Epoch 1/2, Train Loss: 2.9801    Valid Loss: 2.9615, Duration: 0:05:35.546815
Epoch 2/2, Train Loss: 2.9582    Valid Loss: 2.9563, Duration: 0:05:46.537441
```

In [ ]:

```
def get_chunk_accuracy(model, data_iter, order, device):
  predictions = []
  correct_labels = []

  model.eval()
  with torch.no_grad():

    for batch in data_iter:

      b_input_ids = batch[0].to(device)
      b_input_mask = batch[1].to(device)
      b_labels = batch[2].to(device)
    # forward pass
```

```
      loss, logits = model(b_input_ids,
                           token_type_ids=None,
                           attention_mask=b_input_mask,
                           labels=b_labels)

      _,indices = torch.max(logits,dim=1)

      indices = indices.cpu().numpy()
      predictions += list(indices)
      b_labels = b_labels.cpu().numpy()
      correct_labels += list(b_labels)

  return predictions, correct_labels, order
```

In [ ]:

```
predictions, correct_labels, order = get_chunk_accuracy(model, validation_dataloader, order_test, d
evice)
```

In [ ]:

```
state = pd.DataFrame({'order':order, 'preds':predictions,'correct':correct_labels})
state_2 = state.groupby('order').agg(lambda x:x.value_counts().index[0])
state_2[state_2['preds']==state_2['correct']].shape[0]/state['order'].nunique()
```

Out[ ]:

```
0.06120552310143388
```

In [ ]:

```
del sentences_train_chunked, labels_train_chunked, order_train, sentences_test_chunked,
labels_test_chunked, order_test
```

In [ ]:

```
import gc
gc.collect()
```

Out[ ]:

```
222
```

In experiment 1, after reducing the sequence length from 512 we observe 1% improvement in the accuracy of the model whereas 'truncating the model from end' & 'truncating the model from mid' gives us the same accuracy as the one observed after reducing the sequence length. In case of chunking we observe significant reduction of 20% in comparsion to other model.

TASK 2 - Fine tune Bert

In [ ]:

```
def tokenize_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      sent = sent.split(" ")
      sent.reverse()
      sent = " ".join(sent)
      encoded_dict = tokenizer.encode_plus(
                           sent,
                           add_special_tokens = True,
                           max_length = max_len,
                           pad_to_max_length = True,
                           return_attention_mask = True,
                           return_tensors = 'pt',
                    )

      # 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)

    return input_ids, attention_masks
```

Sequence Length, 512

In [ ]:

```
params['max_len'] = 512
params['batch_size'] = 8
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([512])
```

In [ ]:

```
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
BertForSequenceClassification: ['cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.seq_relationship.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.LayerNorm.bias']
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a
model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint o
f a model that you expect to be exactly identical (initializing a BertForSequenceClassification mo
del from a BertForSequenceClassification model).
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at be
rt-base-uncased and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
torch.cuda.empty_cache()
```

```
print(torch.cuda.memory_summary(device=None, abbreviated=False))
```

```
|===========================================================================|
|                  PyTorch CUDA memory summary, device ID 0                 |
|---------------------------------------------------------------------------|
|            CUDA OOMs: 0            |        cudaMalloc retries: 0         |
|===========================================================================|
|        Metric         | Cur Usage  | Peak Usage | Tot Alloc  | Tot Freed  |
|---------------------------------------------------------------------------|
| Allocated memory      |     858 MB |    7938 MB |  107110 GB |  107109 GB |
|       from large pool |     857 MB |    7936 MB |  107088 GB |  107087 GB |
|       from small pool |       1 MB |      21 MB |      22 GB |      22 GB |
|---------------------------------------------------------------------------|
| Active memory         |     858 MB |    7938 MB |  107110 GB |  107109 GB |
|       from large pool |     857 MB |    7936 MB |  107088 GB |  107087 GB |
|       from small pool |       1 MB |      21 MB |      22 GB |      22 GB |
|---------------------------------------------------------------------------|
| GPU reserved memory   |     924 MB |    8234 MB |    8664 MB |    7740 MB |
|       from large pool |     922 MB |    8206 MB |    8636 MB |    7714 MB |
|       from small pool |       2 MB |      28 MB |      28 MB |      26 MB |
|---------------------------------------------------------------------------|
| Non-releasable memory |   67086 KB |  261595 KB |   27587 GB |   27587 GB |
|       from large pool |   66130 KB |  259666 KB |   27564 GB |   27564 GB |
|       from small pool |     956 KB |    7075 KB |      22 GB |      22 GB |
|---------------------------------------------------------------------------|
| Allocations           |     405    |    1066    |    8230 K  |    8229 K  |
|       from large pool |     150    |     496    |    5096 K  |    5096 K  |
|       from small pool |     255    |     714    |    3133 K  |    3133 K  |
|---------------------------------------------------------------------------|
| Active allocs         |     405    |    1066    |    8230 K  |    8229 K  |
|       from large pool |     150    |     496    |    5096 K  |    5096 K  |
|       from small pool |     255    |     714    |    3133 K  |    3133 K  |
|---------------------------------------------------------------------------|
| GPU reserved segments |      51    |     246    |     264    |     213    |
|       from large pool |      50    |     232    |     250    |     200    |
|       from small pool |       1    |      14    |      14    |      13    |
|---------------------------------------------------------------------------|
| Non-releasable allocs |      44    |      59    |    3482 K  |    3482 K  |
|       from large pool |      25    |      33    |    2465 K  |    2465 K  |
|       from small pool |      19    |      31    |    1016 K  |    1016 K  |
|===========================================================================|
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'],1234, device)
```

```
Epoch 1/2, Train Loss: 1.0224    Valid Loss: 1.1226, Duration: 0:23:04.048650
Epoch 2/2, Train Loss: 1.0085    Valid Loss: 1.1113, Duration: 0:23:30.232013
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train accuracy:", train_acc, "Valid Accuracy:" ,valid_acc)
```

```
Train accuracy: 0.782440232714563428 Valid Accuracy: 0.6912464132147456348
```

After fine-tuning the model we observe significant improvement in both training and validation accuracy.

TASK 3 - Different BERT variations from Huggingface library

ALBERT

In [ ]:

```
from transformers import AlbertTokenizer, AlbertForSequenceClassification
```

In [ ]:

```
tokenizer = AlbertTokenizer.from_pretrained('albert-base-v2')
```

In [ ]:

```
def tokenize_s(sentences, max_len):
  input_ids = []
  attention_masks = []
  for sent in sentences:
      sent = sent.split(" ")
      sent.reverse()
      sent = " ".join(sent)
      encoded_dict = tokenizer.encode_plus(
                          sent,                      # Sentence to encode.
                          add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                          max_length = max_len,           # 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)

  return input_ids, attention_masks
```

Sequence Length, 512

In [ ]:

```
params['max_len'] = 512
params['batch_size'] = 8
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use
`truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' tru
ncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select
this strategy more precisely by providing a specific strategy to `truncation`.
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([512])
```

In [ ]:

```
model = AlbertForSequenceClassification.from_pretrained(
    "albert-base-v2",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at albert-base-v2 were not used when initializing
AlbertForSequenceClassification: ['predictions.bias', 'predictions.LayerNorm.weight',
'predictions.LayerNorm.bias', 'predictions.dense.weight', 'predictions.dense.bias',
'predictions.decoder.weight', 'predictions.decoder.bias']
- This IS expected if you are initializing AlbertForSequenceClassification from the checkpoint of
a model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing AlbertForSequenceClassification from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSequenceClassification m
odel from a BertForSequenceClassification model).
Some weights of AlbertForSequenceClassification were not initialized from the model checkpoint at
albert-base-v2 and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

Without Fine-tunning

In [ ]:

```
model = AlbertForSequenceClassification.from_pretrained(
    "albert-base-v2",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at albert-base-v2 were not used when initializing
AlbertForSequenceClassification: ['predictions.bias', 'predictions.LayerNorm.weight',
'predictions.LayerNorm.bias', 'predictions.dense.weight', 'predictions.dense.bias',
'predictions.decoder.weight', 'predictions.decoder.bias']
- This IS expected if you are initializing AlbertForSequenceClassification from the checkpoint of
a model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing AlbertForSequenceClassification from the checkpoint
of a model that you expect to be exactly identical (initializing a BertForSequenceClassification m
odel from a BertForSequenceClassification model).
Some weights of AlbertForSequenceClassification were not initialized from the model checkpoint at
albert-base-v2 and are newly initialized: ['classifier.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
  param.requires_grad = False
for param in model.classifier.parameters():
  param.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'],1234, device)
```

```
Epoch 1/2, Train Loss: 2.9722    Valid Loss: 2.9379, Duration: 0:11:57.462271
Epoch 2/2, Train Loss: 2.9219    Valid Loss: 2.9135, Duration: 0:11:58.317175
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train accuracy:", train_acc, "Valid Accuracy:" ,valid_acc)
```

```
Train accuracy: 0.1251546756231218 Valid Accuracy: 0.12705788635156665
```

With Fine-Tunning

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'],1234, device)
```

```
Epoch 1/2, Train Loss: 1.6314    Valid Loss: 1.2753, Duration: 0:23:53.068745
Epoch 2/2, Train Loss: 0.9262    Valid Loss: 1.1068, Duration: 0:23:54.198061
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train accuracy:", train_acc, "Valid Accuracy:" ,valid_acc)
```

```
Train accuracy: 0.7959165635495846 Valid Accuracy: 0.6733935209771641
```

In case of "ALBERTA" we observe similar results as in case of "bert-base-uncased" for fine-tuned model but in case of without fine-tuned model we observe 12% higher accuracy for "bert-base-uncased" model.

ROBERTA

In [ ]:

```
from transformers import RobertaTokenizer, RobertaForSequenceClassification
```

In [ ]:

```
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
```

Sequence Length, 512

In [ ]:

```
params['max_len'] = 512
params['batch_size'] = 8
```

In [ ]:

```
train_dataloader, validation_dataloader = loaders(sentences_train, labels_train, sentences_test, la
bels_test, params['max_len'], params['batch_size'])
```

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use
`truncation=True` to explicitly truncate examples to max length. Defaulting to 'longest_first' tru
ncation strategy. If you encode pairs of sequences (GLUE-style) with the tokenizer you can select
this strategy more precisely by providing a specific strategy to `truncation`.
```

```
/usr/local/lib/python3.6/dist-packages/transformers/tokenization_utils_base.py:2022:
FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future vers
ion, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use
`padding='max_length'` to pad to a max length. In this case, you can give a specific length with `
max_length` (e.g. `max_length=45`) or leave max_length to None to pad to the maximal input size of
the model (e.g. 512 for Bert).
  FutureWarning,
```

```
Original:  I was wondering if anyone out there could enlighten me on this car I saw
the other day. It was a 2-door sports car, looked to be from the late 60s/
early 70s. It was called a Bricklin. The doors were really small. In addition,
the front bumper was separate from the rest of the body. This is
all I know. If anyone can tellme a model name, engine specs, years
of production, where this car is made, history, or whatever info you
have on this funky looking car, please e-mail.
Token IDs: torch.Size([512])
```

In [ ]:

```
model = RobertaForSequenceClassification.from_pretrained(
    "roberta-base",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at roberta-base were not used when initializing
RobertaForSequenceClassification: ['lm_head.bias', 'lm_head.dense.weight', 'lm_head.dense.bias', '
lm_head.layer_norm.weight', 'lm_head.layer_norm.bias', 'lm_head.decoder.weight',
'roberta.pooler.dense.weight', 'roberta.pooler.dense.bias']
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of
a model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at
roberta-base and are newly initialized: ['classifier.dense.weight', 'classifier.dense.bias', 'clas
sifier.out_proj.weight', 'classifier.out_proj.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

With Fine Tunning

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'],1234, device)
```

```
Epoch 1/2, Train Loss: 1.2981    Valid Loss: 1.1378, Duration: 0:23:06.112439
Epoch 2/2, Train Loss: 0.7336    Valid Loss: 1.0544, Duration: 0:23:06.559929
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train accuracy:", train_acc, "Valid Accuracy:" ,valid_acc)
```

```
Train accuracy: 0.851864946084497 Valid Accuracy: 0.7130908125331917
```

Without Fine Tunning

In [ ]:

```
model = RobertaForSequenceClassification.from_pretrained(
    "roberta-base",
    num_labels = 20,
    output_attentions = False,
    output_hidden_states = False,
)
model.cuda();
```

```
Some weights of the model checkpoint at roberta-base were not used when initializing
RobertaForSequenceClassification: ['lm_head.bias', 'lm_head.dense.weight', 'lm_head.dense.bias', '
lm_head.layer_norm.weight', 'lm_head.layer_norm.bias', 'lm_head.decoder.weight',
'roberta.pooler.dense.weight', 'roberta.pooler.dense.bias']
- This IS expected if you are initializing RobertaForSequenceClassification from the checkpoint of
a model trained on another task or with another architecture (e.g. initializing a
BertForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing RobertaForSequenceClassification from the
checkpoint of a model that you expect to be exactly identical (initializing a
BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at
roberta-base and are newly initialized: ['classifier.dense.weight', 'classifier.dense.bias', 'clas
sifier.out_proj.weight', 'classifier.out_proj.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions an
d inference.
```

In [ ]:

```
for param in model.parameters():
  param.requires_grad = False
for param in model.classifier.parameters():
  param.requires_grad = True
```

In [ ]:

```
params['epochs'], params['optimizer'], params['scheduler'] = model_parameters(model)
```

In [ ]:

```
train_losses, valid_losses = train_loop(model, train_dataloader, validation_dataloader, params['opt
imizer'], params['scheduler'], params['epochs'],1234, device)
```

```
Epoch 1/2, Train Loss: 2.9565    Valid Loss: 2.9206, Duration: 0:10:40.283758
Epoch 2/2, Train Loss: 2.8939    Valid Loss: 2.8862, Duration: 0:10:40.264726
```

In [ ]:

```
train_acc = get_accuracy(train_dataloader, model, device)
valid_acc = get_accuracy(validation_dataloader, model, device)
print("Train accuracy:", train_acc, "Valid Accuracy:" ,valid_acc)
```

```
Train accuracy: 0.35310235106947147 Valid Accuracy: 0.3341741901221455
```

Conclusion:- After implementing all the experiments we observe that "ROBERTA" gives us the best possible validation accuracy for both fine-tuned model and without fine-tuned model of 33.4% & 71.3% respectively.

----------THE END----------