experimentalrun7

April 19, 2021

[ ]:
```
#step1 import all the required libraries
#install this version of transformers and pytorch
!pip install transformers==2.8.0
!pip install torch==1.4.0
from transformers import T5Tokenizer, T5ForConditionalGeneration
import tensorflow_datasets as tfds
import torch
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
import nltk,spacy,re,string,random,time
import matplotlib.pyplot as plt
from gensim.parsing.preprocessing import STOPWORDS
from spacy.lang.en.stop_words import STOP_WORDS
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from collections import Counter
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import␣
↪Input,LSTM,Embedding,Dense,Concatenate,TimeDistributed,Bidirectional
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping,ReduceLROnPlateau
from attention import AttentionLayer
from keras.initializers import Constant
from keras.optimizers import Adam
from keras import backend as K
from rouge import rouge_n,rouge_l_sentence_level,rouge
from bleau import compute_bleu
#disable eager execution
#tf.compat.v1.disable_eager_execution()
#stopwords removal list
nltk.download('stopwords')
#punkt for tokenization
nltk.download('punkt')
```

```python
#for tokenaizations
nltk.download('wordnet')
#combine all the stopwords and create one single list of stopwords
s1=stopwords.words('english')
s2=list(STOP_WORDS)
s3=list(STOPWORDS)
#final list of stopwords
stop_words = s1+s2+s3
#use cuda if available
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

#step2
#contraction are used to replace words with their longer meaningfull counter␣
 ↪parts
contraction = {
"ain't": "am not / are not / is not / has not / have not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is / how does",
"I'd": "I had / I would",
"I'd've": "I would have",
"I'll": "I shall / I will",
"I'll've": "I shall have / I will have",
"I'm": "I am",
"I've": "I have",
"isn't": "is not",
"it'd": "it had / it would",
```

```
"it'd've": "it would have",
"it'll": "it shall / it will",
"it'll've": "it shall have / it will have",
"it's": "it has / it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she had / she would",
"she'd've": "she would have",
"she'll": "she shall / she will",
"she'll've": "she shall have / she will have",
"she's": "she has / she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as / so is",
"that'd": "that would / that had",
"that'd've": "that would have",
"that's": "that has / that is",
"there'd": "there had / there would",
"there'd've": "there would have",
"there's": "there has / there is",
"they'd": "they had / they would",
"they'd've": "they would have",
"they'll": "they shall / they will",
"they'll've": "they shall have / they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we had / we would",
"we'd've": "we would have",
```

```
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what shall / what will",
"what'll've": "what shall have / what will have",
"what're": "what are",
"what's": "what has / what is",
"what've": "what have",
"when's": "when has / when is",
"when've": "when have",
"where'd": "where did",
"where's": "where has / where is",
"where've": "where have",
"who'll": "who shall / who will",
"who'll've": "who shall have / who will have",
"who's": "who has / who is",
"who've": "who have",
"why's": "why has / why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you had / you would",
"you'd've": "you would have",
"you'll": "you shall / you will",
"you'll've": "you shall have / you will have",
"you're": "you are",
"you've": "you have",
"rec'd": "received"
}
#rec'd this is my addition to the list of contractions

#step3
#process_text function is used to remove unwanted characters, stopwords, and␣
↪format the text to create fewer nulls word embeddings
def process_text(text,contractions,remove_stopwords = True):
  #convert words to lower case
```

```
  text = text.lower()

  #replace contractions with their longer forms
  if True:
    text = text.split()
    new_text = []
    for word in text:
      if word in contractions:
        new_text.append(contractions[word])
      else:
        new_text.append(word)
    text = " ".join(new_text)

  #format words and remove unwanted characters
  text = re.sub(r'https?:\/\/.*[\r\n]*', '', text, flags=re.MULTILINE) #remove␣
↪https string
  text = re.sub(r'\<a href', ' ', text) #remove hyperlink
  text = re.sub(r'&amp;', '', text) #remove & in text
  text = re.sub(r'[_"\-;%()|+&=*%.,!?:#$@\[\]/]', ' ', text) #remove unwanted␣
↪characters like puntuation and others
  text = re.sub(r'<br />', ' ', text) #remove new line spaces
  text = re.sub(r'\'', ' ', text) #remove slashes
  text = " ".join(text.split()) #remove trailing spaces
  #string.printable returns all sets of punctuation, digits, ascii_letters and␣
↪whitespace.
  printable = set(string.printable)
  #filter to remove punctuations,digits, ascii_letters and whitespaces
  text = "".join(list(filter(lambda x: x in printable, text)))
  #remove stop words is true then remove stopwords also
  if remove_stopwords:
    text = text.split()
    text = [w for w in text if not w in stop_words]
    text = " ".join(text)

  return text

#step4
#get_data function gets the data from gz file into a dataframe and process the␣
↪columns
#stops are not removed for summary they are only removed from text this is done␣
↪to get more human like summaries
#after processing it returns a dataframe
def get_data(contractions):
  st=time.time()
  #load the data into a dataframe
```

```
  df = pd.read_json('/content/drive/MyDrive/
↪reviews_Clothing_Shoes_and_Jewelry_5.json.gz', lines=True,␣
↪compression='gzip')
  #drop unwanted columns
  df.drop(columns=['reviewerID', 'asin', 'reviewerName',␣
↪'helpful','overall','unixReviewTime', 'reviewTime'],inplace=True)
  print("length of the data",len(df))
  #apply preprocess function on the columns of the dataframe
  df['reviewText'] = df['reviewText'].apply(lambda x:␣
↪process_text(x,contractions,remove_stopwords = True))
  df['summary'] = df[ 'summary'].apply(lambda x:␣
↪process_text(x,contractions,remove_stopwords = False))
  #write preprocesssed data to csv file
  df.to_csv("/content/drive/MyDrive/product_reviews.csv",index=False)
  print("total time to generate data and write in csv file ",time.time()-st)


#step5
#get_embeddings function is used to gett te word embeddings
#i am using conceptual number batch word embeddings
def get_embeddings():
  #get word embeddings
  embeddings_index = {}
  with open('/content/drive/MyDrive/numberbatch-en-19.08.txt',␣
↪encoding='utf-8') as f:
    for line in f:
      values = line.split(' ')
      word = values[0]
      embedding = np.asarray(values[1:], dtype='float32')
      embeddings_index[word] = embedding

  print('Word embeddings:', len(embeddings_index))
  return embeddings_index

#step6
#this function is used to build vocabulary
def get_vocab(embeddings_index,word_counts,threshold):
  #get the number of missing words
  missing_words={k:v for k,v in word_counts.items() if v >= threshold if k not␣
↪in embeddings_index.keys()}
  missing_ratio = round(len(missing_words)/len(word_counts),4)*100
  print("Number of words missing from word_embeddings:", len(missing_words))
  print("Percent of words that are missing from our vocabulary: {}%".
↪format(missing_ratio))

  #mapping vocab to index
```

```python
  lr=iter([item for item in range(0,len(word_counts))])
  vocab_to_int={k:next(lr) for k,v in word_counts.items() if v >= threshold or k in embeddings_index.keys()}

  #mapping index to vocab
  lr=iter([item for item in range(0,len(word_counts))])
  int_to_vocab={next(lr):k for k,v in word_counts.items() if v >= threshold or k in embeddings_index.keys()}

  # Special tokens that will be added to our vocab
  codes = ["<UNK>","<PAD>","<EOS>","<GO>"]

  # Add codes to vocab
  for code in codes:
      vocab_to_int[code] = len(vocab_to_int)
      int_to_vocab[len(int_to_vocab)] = code

  #print usage of words in our model and their percent
  usage_ratio = round(len(vocab_to_int) / len(word_counts),4)*100
  print("Total number of unique words:", len(word_counts))
  print("Number of words we will use:", len(vocab_to_int))
  print("Percent of words we will use: {}%".format(usage_ratio))
  print("length vocab_to_int",len(vocab_to_int))
  print("length int_to_vocab",len(int_to_vocab))

  return vocab_to_int,int_to_vocab

#step7
#function to map words with its word embeddings
#if embeddings not found for the word then map it with a random number in range(-1.0,1.0)
def word_embedding_index(vocab_to_int,embeddings_index):
  #using 300 for embedding dimensions to match CN's vectors.
  embedding_dim = 300
  nb_words = len(vocab_to_int)

  # Create matrix with default values of zero
  word_embedding_matrix = np.zeros((nb_words, embedding_dim), dtype=np.float32)
  for word, i in vocab_to_int.items():
    if word in embeddings_index:
      word_embedding_matrix[i] = embeddings_index[word]
    else:
      # If word not in CN, create a random embedding for it
      new_embedding = np.array(np.random.uniform(-1.0, 1.0, embedding_dim))
      #embeddings_index[word] = new_embedding
      word_embedding_matrix[i] = new_embedding
```

```python
  # Check if value matches len(vocab_to_int)
  print("length of word embedding matrix",len(word_embedding_matrix))
  return word_embedding_matrix

#step8
#append unk and eos tokens
#if eos is equal to true then append go and eos token at begining and end of␣
↪the summary
#add unknown token for word not found in vocabulary
def convert_to_ints(text,vocab_to_int,eos=False):
  ints = []
  for word in text.split():
    if word in vocab_to_int:
      ints.append(vocab_to_int[word])
    else:
      ints.append(vocab_to_int["<UNK>"])
  if eos:
    ints.insert(0,vocab_to_int["<GO>"])
    ints.insert(len(ints),vocab_to_int["<EOS>"])
  return ints

#step9
#count unknown tokens
def count_unk(text):
  unk=0
  eos=0
  #print(text)
  for value in text:
    if 41413 in value:
      unk+=1
  return unk

#step10
def counts(val):
  c=0
  for i in val:
    try:
      if i==41413:
        c+=1
    except:
      pass
  return c

#step11
#remove rows from data frame that dosent staisfy the condition this is done so␣
↪model is trained with proper data
#redundancey is less and input text is accurate
```

```
def get_refined_output(df,max_rl,max_sl):
  unk_rl=1 #unknown token review limit
  unk_sl=0 #unknown token summary limit
  min_rl=2 #minimum review length
  #get the total length of reviewText this is used for sorting
  df["total_length"]=df['reviewText'].apply(lambda x: len(x))
  #get reviewText whose length is greater then minimum review length
  df=df[df['reviewText'].apply(lambda x: len(x)>=min_rl)]
  #get reviewText whose length is less than maximum review length
  df=df[df['reviewText'].apply(lambda x: len(x)<=max_rl)]
  #filter out the unknwon tokens based on unknown token reviewText limit
  df=df[df['reviewText'].apply(lambda x: counts(x)<=unk_rl)]
  #get summary whose length is less than maximum summary length
  df=df[df['summary'].apply(lambda x: len(x)<=max_sl)]
  #filter out the unkown tokens based on unkown token summary limit
  df=df[df['summary'].apply(lambda x: counts(x)<=unk_sl)]
  #sort the values in ascending order
  df.sort_values(by=["total_length"],ascending=True,inplace=True)
  #drop unwanted columns
  df.drop(columns=["total_length","word"],inplace=True)
  #reset index
  df.reset_index(drop=True,inplace=True)
  return df

#step12
#function to plot the length of training, validation and testing
def plot_tr_tval_tt_len(xtr,xval,xtt):
  names = ['Training','Validation','Testing']
  values = [len(xtr),len(xval),len(xtt)]
  plt.figure(figsize=(10,5))
  plt.subplot(131)
  plt.
 ↪bar(names,values,color=['darkorange','coral','coral'],edgecolor='darkblue')
  plt.suptitle('Categorical Plotting')
  plt.show()

#step13
#function to plot loss and accuracy curves on training and validation set
def plotgraph(history):
  plt.figure(figsize=[8,6])
  plt.plot(history.history['loss'],'firebrick',linewidth=3.0)
  plt.plot(history.history['accuracy'],'turquoise',linewidth=3.0)
  plt.plot(history.history['val_loss'],'midnightblue',linewidth=3.0)
  plt.legend(['Training loss','Training Accuracy','Validation␣
 ↪loss'],fontsize=18)
  plt.xlabel('Epochs',fontsize=16)
  plt.ylabel('Loss and Accuracy',fontsize=16)
```

```
    plt.title('Loss Curves and Accuracy Curves for text␣
 ↪summarization',fontsize=16)

#step14
#this function is used to get the preprocessed csv file for our text summarizer
def Get_the_data():
    #lower the string in contractions and convert it into dict
    contractions = dict((k.lower(), v.lower()) for k, v in contraction.items())
    #till this step all data is processed and we get our csv file of cleaned texts
    get_data(contractions)

    #free memory
    del contractions

#step15 is used to call function Get_the_data which get the preprocessed data␣
 ↪and writes it into a csv file
#Get_the_data()

#step16
#this function combines all the above ouput generated by the above function in␣
 ↪a proper squence of steps
def combining_all_steps():

    st=time.time()
    #get the final cleaned data
    df=pd.read_csv('/content/drive/MyDrive/product_reviews.csv')[:180000]
    print("The length of dataset is ",len(df))
    #combine reviewText and summary so common vocabulary can be created by␣
 ↪finding frequent words
    df["word"]=df[['reviewText','summary']].apply(lambda x : '{} {}'.
 ↪format(x[0],x[1]), axis=1)
    #get frequency of words
    word_counts=pd.Series(np.concatenate([x.split() for x in df.word])).
 ↪value_counts()
    word_counts=word_counts.to_dict()
    #print(type(word_counts))
    print("vocab length",len(word_counts))
    #set the threshold
    threshold = 20
    max_rl=80 #maximum review length
    max_sl=10 #maximum summary length
    #get the embeddings matrix
    embeddings_index= get_embeddings()
    #get vocab to index and index to vocab mapping of words
    vocab_to_int,int_to_vocab=get_vocab(embeddings_index,word_counts,threshold)
    #get word embedding for the words in vocab
```

```
word_embedding_matrix=word_embedding_index(vocab_to_int,embeddings_index)
#convert words to integers based on their index positions
df['reviewText'] = df['reviewText'].apply(lambda x: convert_to_ints(str(x),vocab_to_int,eos=False))
df['summary'] = df[ 'summary'].apply(lambda x: convert_to_ints(str(x),vocab_to_int,eos=True))
print("after word to index for reviewText",df["reviewText"][0])
print("after word to index for summary",df["summary"][0])
rvunk=count_unk(df["reviewText"])
smunk=count_unk(df["summary"])
print("total number of unk token are",rvunk+smunk)
#apply the filters and get the final preprocessed data
df=get_refined_output(df,max_rl,max_sl)
print("length of dataset that will be used",len(df))
#split data into 75% train, 15% validation and 15% test datasets
x_tr,x_val,y_tr,y_val=train_test_split(df['reviewText'],df['summary'],test_size=0.3,random_state=1,shuffle=True)
x_tt,x_val,y_tt,y_val=train_test_split(x_val,y_val,test_size=0.5,random_state=1,shuffle=True)
print("length of split datasets train {}, test {} and validation {}".format(len(x_tr),len(x_tt),len(x_val)))
print("Vocabulary Size: {}".format(len(vocab_to_int)))
print("voc_to_int_",vocab_to_int['<UNK>'],vocab_to_int['<PAD>'],vocab_to_int['<EOS>'])
#reset index
x_tr=x_tr.reset_index()
y_tr=y_tr.reset_index()
x_tt=x_tt.reset_index()
y_tt=y_tt.reset_index()
x_val=x_val.reset_index()
y_val=y_val.reset_index()
#find max lenght just to verfiy the output of get refined function
#max([len(sentence) for sentence in y_tt["summary"]])
#pad the reviewText and summary to the specified max length
xtr=pad_sequences(x_tr["reviewText"], padding='post',maxlen=max_rl, value=vocab_to_int["<PAD>"])
ytr=pad_sequences(y_tr["summary"], padding='post',maxlen=max_sl, value=vocab_to_int["<PAD>"])
xtt=pad_sequences(x_tt["reviewText"], padding='post',maxlen=max_rl, value=vocab_to_int["<PAD>"])
ytt=pad_sequences(y_tt["summary"], padding='post',maxlen=max_sl, value=vocab_to_int["<PAD>"])
xval=pad_sequences(x_val["reviewText"], padding='post',maxlen=max_rl, value=vocab_to_int["<PAD>"])
```

```
  yval=pad_sequences(y_val["summary"], padding='post',maxlen=max_sl,␣
↪value=vocab_to_int["<PAD>"])
  #find the number of unique tokens in the list
  #flat_list_rt = [item for sublist in df["reviewText"] for item in sublist]
  #flat_list_s = [item for sublist in df["summary"] for item in sublist]
  #rt=len(np.unique(flat_list_rt))
  #st=len(np.unique(flat_list_s))
  #print("number of unique tokens reviewText {} and summary {}".format(rt,st))
  #plot the length of training, validation and testing
  plot_tr_tval_tt_len(xtr,xval,xtt)
  print("total time to complete all the above steps and get final data ",time.
↪time()-st)
  #free memory delete values stored in variables which are not required further
  del df,word_counts,embeddings_index,x_tr,x_val,y_tr,y_val,x_tt,y_tt

  return␣
↪xtr,ytr,xtt,ytt,xval,yval,vocab_to_int,int_to_vocab,word_embedding_matrix,max_rl,max_sl

#step17
#function to get summary given a sequence
def seq_to_summary(seq,vocab_to_int,int_to_vocab):
  newstring=''
  for i in seq:
    if ((i!=0 and i!=vocab_to_int['<GO>']) and i!=vocab_to_int['<EOS>']):
      newstring=newstring+int_to_vocab[i]+' '
  return newstring

#step18
#function to get text given a sequence
def seq_to_text(seq,int_to_vocab):
  newstring=''
  for i in seq:
    if (i!=0):
      newstring=newstring+int_to_vocab[i]+' '
  return newstring

#step19
#this function get the data for the pretrained model t5small
def combining_all_steps_t5():
  #get the final cleaned data
  df=pd.read_csv('/content/drive/MyDrive/product_reviews.csv')[:180000]
  print("The length of dataset is ",len(df))

  #set the threshold
  threshold = 20
  max_rl=80 #maximum review length
  max_sl=10 #maximum summary length
```

```
    #get reviewText whose length is less than maximum review length
    df['reviewText']=df['reviewText'].str.slice(0,max_rl)

    #get summary whose length is less than maximum summary length
    df['summary']=df['summary'].str.slice(0,max_rl)

    #split data into 75% train, 15% validation and 15% test datasets
␣
↪x_tr,x_val,y_tr,y_val=train_test_split(df['reviewText'],df['summary'],test_size=0.
↪3,random_state=1,shuffle=True)
    x_tt,x_val,y_tt,y_val=train_test_split(x_val,y_val,test_size=0.
↪5,random_state=1,shuffle=True)

    #reset index
    x_tr=x_tr.reset_index()
    y_tr=y_tr.reset_index()
    x_tt=x_tt.reset_index()
    y_tt=y_tt.reset_index()
    x_val=x_val.reset_index()
    y_val=y_val.reset_index()
    print("train {}, val {}, test {}".format(len(x_tr),len(x_val),len(x_tt)))
    return x_tr,y_tr,x_tt,y_tt,x_val,y_val
```

```
Collecting transformers==2.8.0
  Downloading https://files.pythonhosted.org/packages/a3/78/92cedda0555239
8352ed9784908b834ee32a0bd071a9b32de287327370b7/transformers-2.8.0-py3-none-
any.whl (563kB)
     |                    | 573kB 12.5MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-
packages (from transformers==2.8.0) (3.0.12)
Collecting tokenizers==0.5.2
  Downloading https://files.pythonhosted.org/packages/d6/e3/5e49e9a83fb605
aaa34a1c1173e607302fecae529428c28696fb18f1c2c9/tokenizers-0.5.2-cp37-cp37m-manyl
inux1_x86_64.whl (5.6MB)
     |                    | 5.6MB 31.1MB/s
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.7/dist-packages (from transformers==2.8.0) (4.41.1)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.7/dist-packages (from transformers==2.8.0) (2019.12.20)
Collecting boto3
  Downloading https://files.pythonhosted.org/packages/40/60/78919d8b178668
aac44b5d5f4fbe660880179ada1e9000cf3ee3bfcb6421/boto3-1.17.50.tar.gz (99kB)
     |                    | 102kB 10.6MB/s
Collecting sacremoses
  Downloading https://files.pythonhosted.org/packages/08/cd/342e584ee544d0
44fb573ae697404ce22ede086c9e87ce5960772084cad0/sacremoses-0.0.44.tar.gz (862kB)
```

```
     |                 | 870kB 45.6MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (from transformers==2.8.0) (1.19.5)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from transformers==2.8.0) (2.23.0)
Collecting sentencepiece
  Downloading https://files.pythonhosted.org/packages/f5/99/e0808cb947ba10
f575839c43e8fafc9cc44e4a7a2c8f79c60db48220a577/sentencepiece-0.1.95-cp37-cp37m-m
anylinux2014_x86_64.whl (1.2MB)
     |                 | 1.2MB 57.5MB/s
Collecting botocore<1.21.0,>=1.20.50
  Downloading https://files.pythonhosted.org/packages/f7/ae/e7e003597f9542
83f90f21891bda64bab0fc1738951aeb09a7c798ef0a60/botocore-1.20.50-py2.py3-none-
any.whl (7.4MB)
     |                 | 7.4MB 42.2MB/s
Collecting jmespath<1.0.0,>=0.7.1
  Downloading https://files.pythonhosted.org/packages/07/cb/5f001272b6faeb23c1c9
e0acc04d48eaaf5c862c17709d20e3469c6e0139/jmespath-0.10.0-py2.py3-none-any.whl
Collecting s3transfer<0.4.0,>=0.3.0
  Downloading https://files.pythonhosted.org/packages/98/14/0b4be62b65c52d
6d1c442f24e02d2a9889a73d3c352002e14c70f84a679f/s3transfer-0.3.6-py2.py3-none-
any.whl (73kB)
     |                 | 81kB 10.4MB/s
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-
packages (from sacremoses->transformers==2.8.0) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers==2.8.0) (7.1.2)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers==2.8.0) (1.0.1)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==2.8.0)
(3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->transformers==2.8.0) (2.10)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==2.8.0)
(2020.12.5)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers==2.8.0)
(1.24.3)
Requirement already satisfied: python-dateutil<3.0.0,>=2.1 in
/usr/local/lib/python3.7/dist-packages (from
botocore<1.21.0,>=1.20.50->boto3->transformers==2.8.0) (2.8.1)
Building wheels for collected packages: boto3, sacremoses
  Building wheel for boto3 (setup.py) … done
  Created wheel for boto3: filename=boto3-1.17.50-py2.py3-none-any.whl
size=128779
sha256=35ef9514c3dbfc73a7bcb7154f506887375439e8cf9e6e355afe64b9916aac25
```

```
  Stored in directory: /root/.cache/pip/wheels/28/e5/43/ef6fc36c3008477a35f9324c
0e490c7aa20f7b51993a388267
  Building wheel for sacremoses (setup.py) … done
  Created wheel for sacremoses: filename=sacremoses-0.0.44-cp37-none-any.whl
size=886084
sha256=109a8ea2b9b2f8256ec87ff5faddeb10e6f1cff6e4e51038107ad90586f57e59
  Stored in directory: /root/.cache/pip/wheels/3e/fb/c0/13ab4d63d537658f44836674
4654323077c4d90069b6512f3c
Successfully built boto3 sacremoses
ERROR: botocore 1.20.50 has requirement urllib3<1.27,>=1.25.4, but you'll
have urllib3 1.24.3 which is incompatible.
Installing collected packages: tokenizers, jmespath, botocore, s3transfer,
boto3, sacremoses, sentencepiece, transformers
Successfully installed boto3-1.17.50 botocore-1.20.50 jmespath-0.10.0
s3transfer-0.3.6 sacremoses-0.0.44 sentencepiece-0.1.95 tokenizers-0.5.2
transformers-2.8.0
Collecting torch==1.4.0
  Downloading https://files.pythonhosted.org/packages/1a/3b/fa92ece1e58a6a
48ec598bab327f39d69808133e5b2fb33002ca754e381e/torch-1.4.0-cp37-cp37m-manylinux1
_x86_64.whl (753.4MB)
     |                  | 753.4MB 22kB/s
ERROR: torchvision 0.9.1+cu101 has requirement torch==1.8.1, but you'll
have torch 1.4.0 which is incompatible.
ERROR: torchtext 0.9.1 has requirement torch==1.8.1, but you'll have torch
1.4.0 which is incompatible.
Installing collected packages: torch
  Found existing installation: torch 1.8.1+cu101
    Uninstalling torch-1.8.1+cu101:
      Successfully uninstalled torch-1.8.1+cu101
Successfully installed torch-1.4.0
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Unzipping corpora/wordnet.zip.
```

[ ]:
```
#summary using T5small pretrained model
```

[ ]:
```
#step26
#function is used to return the loss
def step(inputs_ids, attention_mask, y, pad_token_id, model):
  y_ids = y[:, :-1].contiguous()
  lm_labels = y[:, 1:].clone()
  lm_labels[y[:, 1:] == pad_token_id] = -100
```

```
  output = model(inputs_ids, attention_mask=attention_mask,␣
↪decoder_input_ids=y_ids, lm_labels=lm_labels)
  # loss
  return output[0]
```

[ ]:
```
#step25
#this function is used to train the pretrained t5small model
def t5train(train_loader,val_loader,pad_token_id,model,EPOCHS,log_interval):
  #initialize empty list for train_loss and val_loss
  train_loss = []
  val_loss = []
  #optimizer
  optimizer = torch.optim.Adam(model.parameters(),lr=1e-4, weight_decay=1e-4/25)
  #iterate for number of epochs
  for epoch in range(EPOCHS):
    model.train()
    #start time
    start_time = time.time()
    #for data in train_loader train the model
    for i, (inputs_ids, attention_mask, y) in enumerate(train_loader):
      inputs_ids = inputs_ids.to(device)
      attention_mask = attention_mask.to(device)
      y = y.to(device)

      optimizer.zero_grad()
      loss = step(inputs_ids, attention_mask, y, pad_token_id, model)
      train_loss.append(loss.item())
      loss.backward()
      torch.nn.utils.clip_grad_norm_(model.parameters(), 0.5)
      optimizer.step()

      if (i + 1) % log_interval == 0:
        with torch.no_grad():
          x, x_mask, y = next(iter(val_loader))
          x = x.to(device)
          x_mask = x_mask.to(device)
          y = y.to(device)

          v_loss = step(x, x_mask, y, pad_token_id, model)
          v_loss = v_loss.item()

          elapsed = time.time() - start_time
          print('| epoch {:3d} | [{:5d}/{:5d}] | '
                'ms/batch {:5.2f} | '
                'loss {:5.2f} | val loss {:5.2f}'.format(
                  epoch, i, len(train_loader),
                  elapsed * 1000 / log_interval,
```

```
                loss.item(), v_loss))
        start_time = time.time()
        val_loss.append(v_loss)

  return model
```

[ ]:

```
#step26
#function to test the model it writes original and predicted summary in txt file
def testT5(model,tokenizer,test_loader):
  #intialize the empty lists
  predictions = []
  real_og=[]
  pred_op=[]
  c=0
  b=1000
  #for data in test loader
  for i, (input_ids, attention_mask, y) in enumerate(test_loader):
    input_ids = input_ids.to(device)
    attention_mask = attention_mask.to(device)
    y = y.to(device)
    #generate summaries
    #store real and predicted summary in a list and write in txt file
    summaries = model.generate(input_ids=input_ids,␣
↪attention_mask=attention_mask,max_length=10)
    pred = [tokenizer.decode(g, skip_special_tokens=True,␣
↪clean_up_tokenization_spaces=False) for g in summaries]
    real = [tokenizer.decode(g, skip_special_tokens=True,␣
↪clean_up_tokenization_spaces=False) for g in y]
    for pred_sent, real_sent in zip(pred, real):
      if c>b:
        print("Original: {}".format(real_sent))
        print("Predicted: {}".format(pred_sent))
        print("\n")
        b+=b
      real_og.append(real_sent)
      pred_op.append(pred_sent)
      predictions.append(str("pred sentence: " + pred_sent + "\t\t real␣
↪sentence: " + real_sent+"\n"))
      c+=1
  file1 = open("/content/drive/MyDrive/TFIVE.txt","w")
  file1.writelines(predictions)
  file1.close()
  #calculate scores
  bleau=compute_bleu(real_og,pred_op, max_order=4,smooth=False)
  rougen=rouge_n(pred_op, real_og, n=2)
  ro=rouge(pred_op, real_og)
```

```
print("bleu, precisions, bp, ratio, translation_length,␣
↪reference_length",bleau)
print("rouge2",rougen)
print("rouge",ro)
```

[ ]:
```
#step27
#fucntion to get the data and call all the functions in a squence
def tf5token():
  class MyDataset(torch.utils.data.Dataset):
    def __init__(self, articles, highlights):
      self.x = articles
      self.y = highlights

    def __getitem__(self,index):
      x = tokenizer.encode_plus(model.config.prefix + str(self.x[index]),␣
↪max_length=80, return_tensors="pt", pad_to_max_length=True)
      y = tokenizer.encode(str(self.y[index]), max_length=10,␣
↪return_tensors="pt", pad_to_max_length=True)
      return x['input_ids'].view(-1), x['attention_mask'].view(-1), y.view(-1)

    def __len__(self):
      return len(self.x)

  #get the data
  x_tr,y_tr,x_tt,y_tt,x_val,y_val=combining_all_steps_t5()
  BATCH_SIZE = 128
  SHUFFEL_SIZE = 1024
  EPOCHS = 100
  log_interval = 200
  #get the pretrained model t5-small
  tokenizer = T5Tokenizer.from_pretrained('t5-small')
  model = T5ForConditionalGeneration.from_pretrained('t5-small').to(device)

  task_specific_params = model.config.task_specific_params
  if task_specific_params is not None:
    model.config.update(task_specific_params.get("summarization", {}))

  #create train,test and validation datasets
  train_ds = MyDataset(x_tr["reviewText"],y_tr["summary"])
  val_ds = MyDataset(x_val["reviewText"],y_val["summary"])
  test_ds = MyDataset(x_tt["reviewText"],y_tt["summary"])

  train_loader = torch.utils.data.DataLoader(train_ds, batch_size=BATCH_SIZE)
  val_loader = torch.utils.data.DataLoader(val_ds, batch_size=BATCH_SIZE)
  test_loader = torch.utils.data.DataLoader(test_ds, batch_size=BATCH_SIZE)

  x, x_mask, y = next(iter(val_loader))
```

```python
print(x.shape, x_mask.shape, y.shape)
pad_token_id = tokenizer.pad_token_id

#call the train function
model=t5train(train_loader,val_loader,pad_token_id,model,EPOCHS,log_interval)
#call the test function
testT5(model,tokenizer,test_loader)
```

[7]:
```python
tf5token()
```

```
The length of dataset is  180000
train 126000, val 27000, test 27000
```

```
HBox(children=(FloatProgress(value=0.0, description='Downloading', max=791656.0,␣
↪style=ProgressStyle(descripti…
```

```
HBox(children=(FloatProgress(value=0.0, description='Downloading', max=1197.0,␣
↪style=ProgressStyle(description…
```

```
HBox(children=(FloatProgress(value=0.0, description='Downloading', max=242065649.
↪0, style=ProgressStyle(descri…
```

```
torch.Size([128, 80]) torch.Size([128, 80]) torch.Size([128, 10])
| epoch   0 | [  199/  985] | ms/batch 247.84 | loss  4.21 | val loss  4.24
| epoch   0 | [  399/  985] | ms/batch 246.78 | loss  4.18 | val loss  4.08
| epoch   0 | [  599/  985] | ms/batch 247.75 | loss  4.29 | val loss  4.05
| epoch   0 | [  799/  985] | ms/batch 249.15 | loss  3.83 | val loss  3.98
| epoch   1 | [  199/  985] | ms/batch 250.21 | loss  3.73 | val loss  3.90
| epoch   1 | [  399/  985] | ms/batch 248.36 | loss  3.94 | val loss  3.82
| epoch   1 | [  599/  985] | ms/batch 248.31 | loss  3.94 | val loss  3.79
| epoch   1 | [  799/  985] | ms/batch 247.38 | loss  3.71 | val loss  3.68
| epoch   2 | [  199/  985] | ms/batch 248.53 | loss  3.58 | val loss  3.74
| epoch   2 | [  399/  985] | ms/batch 246.75 | loss  3.72 | val loss  3.73
| epoch   2 | [  599/  985] | ms/batch 246.90 | loss  3.85 | val loss  3.71
| epoch   2 | [  799/  985] | ms/batch 246.27 | loss  3.48 | val loss  3.68
| epoch   3 | [  199/  985] | ms/batch 246.67 | loss  3.44 | val loss  3.66
| epoch   3 | [  399/  985] | ms/batch 246.75 | loss  3.65 | val loss  3.59
| epoch   3 | [  599/  985] | ms/batch 247.11 | loss  3.73 | val loss  3.58
| epoch   3 | [  799/  985] | ms/batch 246.96 | loss  3.43 | val loss  3.56
| epoch   4 | [  199/  985] | ms/batch 246.22 | loss  3.37 | val loss  3.49
| epoch   4 | [  399/  985] | ms/batch 246.85 | loss  3.62 | val loss  3.52
| epoch   4 | [  599/  985] | ms/batch 246.64 | loss  3.64 | val loss  3.56
| epoch   4 | [  799/  985] | ms/batch 248.27 | loss  3.30 | val loss  3.52
| epoch   5 | [  199/  985] | ms/batch 247.52 | loss  3.33 | val loss  3.53
| epoch   5 | [  399/  985] | ms/batch 245.98 | loss  3.53 | val loss  3.50
```

```
| epoch   5 | [  599/  985] | ms/batch 246.68 | loss  3.60 | val loss  3.52
| epoch   5 | [  799/  985] | ms/batch 246.76 | loss  3.27 | val loss  3.46
| epoch   6 | [  199/  985] | ms/batch 246.01 | loss  3.30 | val loss  3.54
| epoch   6 | [  399/  985] | ms/batch 247.91 | loss  3.55 | val loss  3.48
| epoch   6 | [  599/  985] | ms/batch 248.00 | loss  3.55 | val loss  3.48
| epoch   6 | [  799/  985] | ms/batch 247.10 | loss  3.20 | val loss  3.48
| epoch   7 | [  199/  985] | ms/batch 247.56 | loss  3.19 | val loss  3.54
| epoch   7 | [  399/  985] | ms/batch 247.74 | loss  3.42 | val loss  3.44
| epoch   7 | [  599/  985] | ms/batch 246.98 | loss  3.56 | val loss  3.53
| epoch   7 | [  799/  985] | ms/batch 246.34 | loss  3.21 | val loss  3.51
| epoch   8 | [  199/  985] | ms/batch 246.64 | loss  3.20 | val loss  3.52
| epoch   8 | [  399/  985] | ms/batch 247.07 | loss  3.41 | val loss  3.41
| epoch   8 | [  599/  985] | ms/batch 249.35 | loss  3.43 | val loss  3.48
| epoch   8 | [  799/  985] | ms/batch 247.46 | loss  3.10 | val loss  3.40
| epoch   9 | [  199/  985] | ms/batch 246.53 | loss  3.15 | val loss  3.50
| epoch   9 | [  399/  985] | ms/batch 246.46 | loss  3.42 | val loss  3.42
| epoch   9 | [  599/  985] | ms/batch 246.44 | loss  3.42 | val loss  3.44
| epoch   9 | [  799/  985] | ms/batch 247.05 | loss  3.14 | val loss  3.42
| epoch  10 | [  199/  985] | ms/batch 246.55 | loss  3.08 | val loss  3.48
| epoch  10 | [  399/  985] | ms/batch 246.23 | loss  3.40 | val loss  3.43
| epoch  10 | [  599/  985] | ms/batch 246.93 | loss  3.39 | val loss  3.44
| epoch  10 | [  799/  985] | ms/batch 250.06 | loss  3.07 | val loss  3.46
| epoch  11 | [  199/  985] | ms/batch 246.89 | loss  3.07 | val loss  3.37
| epoch  11 | [  399/  985] | ms/batch 247.07 | loss  3.36 | val loss  3.43
| epoch  11 | [  599/  985] | ms/batch 246.72 | loss  3.30 | val loss  3.41
| epoch  11 | [  799/  985] | ms/batch 246.75 | loss  2.97 | val loss  3.43
| epoch  12 | [  199/  985] | ms/batch 249.47 | loss  2.96 | val loss  3.42
| epoch  12 | [  399/  985] | ms/batch 250.09 | loss  3.33 | val loss  3.41
| epoch  12 | [  599/  985] | ms/batch 247.76 | loss  3.35 | val loss  3.38
| epoch  12 | [  799/  985] | ms/batch 248.59 | loss  2.96 | val loss  3.42
| epoch  13 | [  199/  985] | ms/batch 247.21 | loss  2.93 | val loss  3.45
| epoch  13 | [  399/  985] | ms/batch 247.22 | loss  3.21 | val loss  3.41
| epoch  13 | [  599/  985] | ms/batch 247.58 | loss  3.24 | val loss  3.36
| epoch  13 | [  799/  985] | ms/batch 247.10 | loss  2.96 | val loss  3.47
| epoch  14 | [  199/  985] | ms/batch 246.55 | loss  2.97 | val loss  3.42
| epoch  14 | [  399/  985] | ms/batch 246.52 | loss  3.12 | val loss  3.43
| epoch  14 | [  599/  985] | ms/batch 246.47 | loss  3.23 | val loss  3.43
| epoch  14 | [  799/  985] | ms/batch 247.15 | loss  2.91 | val loss  3.47
| epoch  15 | [  199/  985] | ms/batch 246.55 | loss  2.89 | val loss  3.42
| epoch  15 | [  399/  985] | ms/batch 246.96 | loss  3.23 | val loss  3.40
| epoch  15 | [  599/  985] | ms/batch 247.12 | loss  3.22 | val loss  3.42
| epoch  15 | [  799/  985] | ms/batch 247.06 | loss  2.89 | val loss  3.47
| epoch  16 | [  199/  985] | ms/batch 248.09 | loss  2.81 | val loss  3.39
| epoch  16 | [  399/  985] | ms/batch 247.55 | loss  3.20 | val loss  3.40
| epoch  16 | [  599/  985] | ms/batch 246.98 | loss  3.24 | val loss  3.43
| epoch  16 | [  799/  985] | ms/batch 246.00 | loss  2.82 | val loss  3.43
| epoch  17 | [  199/  985] | ms/batch 249.92 | loss  2.85 | val loss  3.38
| epoch  17 | [  399/  985] | ms/batch 248.41 | loss  3.16 | val loss  3.37
```

```
| epoch  17 | [  599/  985] | ms/batch 247.59 | loss  3.15 | val loss  3.41
| epoch  17 | [  799/  985] | ms/batch 248.04 | loss  2.83 | val loss  3.38
| epoch  18 | [  199/  985] | ms/batch 247.95 | loss  2.76 | val loss  3.36
| epoch  18 | [  399/  985] | ms/batch 248.01 | loss  3.02 | val loss  3.42
| epoch  18 | [  599/  985] | ms/batch 250.86 | loss  3.16 | val loss  3.50
| epoch  18 | [  799/  985] | ms/batch 247.63 | loss  2.82 | val loss  3.42
| epoch  19 | [  199/  985] | ms/batch 248.58 | loss  2.71 | val loss  3.32
| epoch  19 | [  399/  985] | ms/batch 246.93 | loss  3.03 | val loss  3.42
| epoch  19 | [  599/  985] | ms/batch 244.98 | loss  3.07 | val loss  3.38
| epoch  19 | [  799/  985] | ms/batch 245.06 | loss  2.77 | val loss  3.37
| epoch  20 | [  199/  985] | ms/batch 244.68 | loss  2.75 | val loss  3.46
| epoch  20 | [  399/  985] | ms/batch 245.16 | loss  3.16 | val loss  3.38
| epoch  20 | [  599/  985] | ms/batch 245.61 | loss  3.05 | val loss  3.42
| epoch  20 | [  799/  985] | ms/batch 245.97 | loss  2.75 | val loss  3.37
| epoch  21 | [  199/  985] | ms/batch 245.72 | loss  2.72 | val loss  3.43
| epoch  21 | [  399/  985] | ms/batch 244.66 | loss  2.99 | val loss  3.35
| epoch  21 | [  599/  985] | ms/batch 245.70 | loss  3.08 | val loss  3.36
| epoch  21 | [  799/  985] | ms/batch 245.04 | loss  2.71 | val loss  3.41
| epoch  22 | [  199/  985] | ms/batch 247.96 | loss  2.73 | val loss  3.40
| epoch  22 | [  399/  985] | ms/batch 245.96 | loss  3.01 | val loss  3.50
| epoch  22 | [  599/  985] | ms/batch 244.56 | loss  3.01 | val loss  3.34
| epoch  22 | [  799/  985] | ms/batch 245.31 | loss  2.76 | val loss  3.44
| epoch  23 | [  199/  985] | ms/batch 246.48 | loss  2.74 | val loss  3.38
| epoch  23 | [  399/  985] | ms/batch 248.31 | loss  3.01 | val loss  3.35
| epoch  23 | [  599/  985] | ms/batch 246.65 | loss  2.97 | val loss  3.36
| epoch  23 | [  799/  985] | ms/batch 245.00 | loss  2.71 | val loss  3.41
| epoch  24 | [  199/  985] | ms/batch 244.73 | loss  2.74 | val loss  3.34
| epoch  24 | [  399/  985] | ms/batch 244.71 | loss  2.89 | val loss  3.43
| epoch  24 | [  599/  985] | ms/batch 244.66 | loss  3.02 | val loss  3.40
| epoch  24 | [  799/  985] | ms/batch 244.84 | loss  2.63 | val loss  3.40
| epoch  25 | [  199/  985] | ms/batch 244.53 | loss  2.58 | val loss  3.37
| epoch  25 | [  399/  985] | ms/batch 244.98 | loss  2.97 | val loss  3.39
| epoch  25 | [  599/  985] | ms/batch 244.53 | loss  2.90 | val loss  3.43
| epoch  25 | [  799/  985] | ms/batch 244.08 | loss  2.59 | val loss  3.44
| epoch  26 | [  199/  985] | ms/batch 244.65 | loss  2.66 | val loss  3.44
| epoch  26 | [  399/  985] | ms/batch 244.99 | loss  2.89 | val loss  3.39
| epoch  26 | [  599/  985] | ms/batch 243.99 | loss  2.88 | val loss  3.38
| epoch  26 | [  799/  985] | ms/batch 244.38 | loss  2.63 | val loss  3.38
| epoch  27 | [  199/  985] | ms/batch 243.84 | loss  2.57 | val loss  3.45
| epoch  27 | [  399/  985] | ms/batch 244.27 | loss  2.83 | val loss  3.39
| epoch  27 | [  599/  985] | ms/batch 245.72 | loss  2.88 | val loss  3.46
| epoch  27 | [  799/  985] | ms/batch 245.07 | loss  2.64 | val loss  3.43
| epoch  28 | [  199/  985] | ms/batch 245.13 | loss  2.45 | val loss  3.51
| epoch  28 | [  399/  985] | ms/batch 246.72 | loss  2.83 | val loss  3.40
| epoch  28 | [  599/  985] | ms/batch 244.73 | loss  2.83 | val loss  3.43
| epoch  28 | [  799/  985] | ms/batch 245.03 | loss  2.53 | val loss  3.42
| epoch  29 | [  199/  985] | ms/batch 244.72 | loss  2.48 | val loss  3.53
| epoch  29 | [  399/  985] | ms/batch 244.29 | loss  2.81 | val loss  3.43
```

```
| epoch  29 | [  599/  985] | ms/batch 244.13 | loss  2.81 | val loss  3.44
| epoch  29 | [  799/  985] | ms/batch 243.95 | loss  2.57 | val loss  3.45
| epoch  30 | [  199/  985] | ms/batch 244.11 | loss  2.42 | val loss  3.49
| epoch  30 | [  399/  985] | ms/batch 244.00 | loss  2.82 | val loss  3.43
| epoch  30 | [  599/  985] | ms/batch 243.66 | loss  2.86 | val loss  3.36
| epoch  30 | [  799/  985] | ms/batch 243.93 | loss  2.57 | val loss  3.46
| epoch  31 | [  199/  985] | ms/batch 244.21 | loss  2.44 | val loss  3.48
| epoch  31 | [  399/  985] | ms/batch 243.66 | loss  2.75 | val loss  3.41
| epoch  31 | [  599/  985] | ms/batch 244.11 | loss  2.79 | val loss  3.46
| epoch  31 | [  799/  985] | ms/batch 243.55 | loss  2.51 | val loss  3.44
| epoch  32 | [  199/  985] | ms/batch 243.45 | loss  2.42 | val loss  3.46
| epoch  32 | [  399/  985] | ms/batch 243.81 | loss  2.82 | val loss  3.40
| epoch  32 | [  599/  985] | ms/batch 243.70 | loss  2.76 | val loss  3.38
| epoch  32 | [  799/  985] | ms/batch 243.84 | loss  2.61 | val loss  3.45
| epoch  33 | [  199/  985] | ms/batch 244.19 | loss  2.45 | val loss  3.43
| epoch  33 | [  399/  985] | ms/batch 243.33 | loss  2.70 | val loss  3.42
| epoch  33 | [  599/  985] | ms/batch 242.42 | loss  2.80 | val loss  3.50
| epoch  33 | [  799/  985] | ms/batch 243.04 | loss  2.55 | val loss  3.43
| epoch  34 | [  199/  985] | ms/batch 242.68 | loss  2.40 | val loss  3.46
| epoch  34 | [  399/  985] | ms/batch 242.48 | loss  2.76 | val loss  3.49
| epoch  34 | [  599/  985] | ms/batch 242.49 | loss  2.74 | val loss  3.49
| epoch  34 | [  799/  985] | ms/batch 242.75 | loss  2.54 | val loss  3.58
| epoch  35 | [  199/  985] | ms/batch 243.34 | loss  2.38 | val loss  3.50
| epoch  35 | [  399/  985] | ms/batch 242.89 | loss  2.65 | val loss  3.43
| epoch  35 | [  599/  985] | ms/batch 242.51 | loss  2.75 | val loss  3.59
| epoch  35 | [  799/  985] | ms/batch 242.82 | loss  2.46 | val loss  3.43
| epoch  36 | [  199/  985] | ms/batch 242.95 | loss  2.35 | val loss  3.50
| epoch  36 | [  399/  985] | ms/batch 242.72 | loss  2.67 | val loss  3.45
| epoch  36 | [  599/  985] | ms/batch 243.42 | loss  2.62 | val loss  3.41
| epoch  36 | [  799/  985] | ms/batch 242.50 | loss  2.47 | val loss  3.44
| epoch  37 | [  199/  985] | ms/batch 242.84 | loss  2.27 | val loss  3.51
| epoch  37 | [  399/  985] | ms/batch 242.76 | loss  2.67 | val loss  3.42
| epoch  37 | [  599/  985] | ms/batch 243.18 | loss  2.67 | val loss  3.49
| epoch  37 | [  799/  985] | ms/batch 242.47 | loss  2.45 | val loss  3.41
| epoch  38 | [  199/  985] | ms/batch 242.70 | loss  2.30 | val loss  3.65
| epoch  38 | [  399/  985] | ms/batch 243.44 | loss  2.64 | val loss  3.45
| epoch  38 | [  599/  985] | ms/batch 243.17 | loss  2.65 | val loss  3.52
| epoch  38 | [  799/  985] | ms/batch 242.87 | loss  2.42 | val loss  3.48
| epoch  39 | [  199/  985] | ms/batch 243.42 | loss  2.33 | val loss  3.53
| epoch  39 | [  399/  985] | ms/batch 243.29 | loss  2.59 | val loss  3.41
| epoch  39 | [  599/  985] | ms/batch 243.09 | loss  2.66 | val loss  3.49
| epoch  39 | [  799/  985] | ms/batch 243.36 | loss  2.38 | val loss  3.60
| epoch  40 | [  199/  985] | ms/batch 243.58 | loss  2.28 | val loss  3.48
| epoch  40 | [  399/  985] | ms/batch 243.50 | loss  2.56 | val loss  3.39
| epoch  40 | [  599/  985] | ms/batch 244.04 | loss  2.59 | val loss  3.55
| epoch  40 | [  799/  985] | ms/batch 243.07 | loss  2.36 | val loss  3.62
| epoch  41 | [  199/  985] | ms/batch 243.33 | loss  2.24 | val loss  3.64
| epoch  41 | [  399/  985] | ms/batch 243.78 | loss  2.53 | val loss  3.53
```

```
| epoch  41 | [  599/  985] | ms/batch 243.65 | loss  2.67 | val loss  3.58
| epoch  41 | [  799/  985] | ms/batch 243.30 | loss  2.40 | val loss  3.57
| epoch  42 | [  199/  985] | ms/batch 243.13 | loss  2.25 | val loss  3.59
| epoch  42 | [  399/  985] | ms/batch 243.60 | loss  2.53 | val loss  3.54
| epoch  42 | [  599/  985] | ms/batch 243.30 | loss  2.58 | val loss  3.48
| epoch  42 | [  799/  985] | ms/batch 244.52 | loss  2.34 | val loss  3.54
| epoch  43 | [  199/  985] | ms/batch 244.44 | loss  2.21 | val loss  3.53
| epoch  43 | [  399/  985] | ms/batch 246.95 | loss  2.49 | val loss  3.49
| epoch  43 | [  599/  985] | ms/batch 245.97 | loss  2.53 | val loss  3.62
| epoch  43 | [  799/  985] | ms/batch 245.25 | loss  2.20 | val loss  3.58
| epoch  44 | [  199/  985] | ms/batch 244.72 | loss  2.24 | val loss  3.64
| epoch  44 | [  399/  985] | ms/batch 244.24 | loss  2.46 | val loss  3.46
| epoch  44 | [  599/  985] | ms/batch 245.49 | loss  2.57 | val loss  3.52
| epoch  44 | [  799/  985] | ms/batch 247.49 | loss  2.35 | val loss  3.57
| epoch  45 | [  199/  985] | ms/batch 247.50 | loss  2.17 | val loss  3.58
| epoch  45 | [  399/  985] | ms/batch 245.56 | loss  2.48 | val loss  3.42
| epoch  45 | [  599/  985] | ms/batch 244.95 | loss  2.52 | val loss  3.61
| epoch  45 | [  799/  985] | ms/batch 245.74 | loss  2.30 | val loss  3.56
| epoch  46 | [  199/  985] | ms/batch 246.02 | loss  2.23 | val loss  3.55
| epoch  46 | [  399/  985] | ms/batch 245.64 | loss  2.48 | val loss  3.54
| epoch  46 | [  599/  985] | ms/batch 246.00 | loss  2.45 | val loss  3.62
| epoch  46 | [  799/  985] | ms/batch 246.55 | loss  2.26 | val loss  3.57
| epoch  47 | [  199/  985] | ms/batch 247.29 | loss  2.14 | val loss  3.56
| epoch  47 | [  399/  985] | ms/batch 246.71 | loss  2.38 | val loss  3.56
| epoch  47 | [  599/  985] | ms/batch 247.28 | loss  2.39 | val loss  3.58
| epoch  47 | [  799/  985] | ms/batch 247.74 | loss  2.25 | val loss  3.59
| epoch  48 | [  199/  985] | ms/batch 247.69 | loss  2.15 | val loss  3.67
| epoch  48 | [  399/  985] | ms/batch 247.74 | loss  2.38 | val loss  3.64
| epoch  48 | [  599/  985] | ms/batch 246.77 | loss  2.42 | val loss  3.56
| epoch  48 | [  799/  985] | ms/batch 247.55 | loss  2.28 | val loss  3.67
| epoch  49 | [  199/  985] | ms/batch 247.64 | loss  2.16 | val loss  3.66
| epoch  49 | [  399/  985] | ms/batch 248.40 | loss  2.37 | val loss  3.61
| epoch  49 | [  599/  985] | ms/batch 248.78 | loss  2.42 | val loss  3.52
| epoch  49 | [  799/  985] | ms/batch 248.59 | loss  2.20 | val loss  3.72
| epoch  50 | [  199/  985] | ms/batch 248.50 | loss  2.12 | val loss  3.64
| epoch  50 | [  399/  985] | ms/batch 248.11 | loss  2.32 | val loss  3.66
| epoch  50 | [  599/  985] | ms/batch 247.54 | loss  2.36 | val loss  3.60
| epoch  50 | [  799/  985] | ms/batch 247.41 | loss  2.16 | val loss  3.62
| epoch  51 | [  199/  985] | ms/batch 245.80 | loss  2.13 | val loss  3.73
| epoch  51 | [  399/  985] | ms/batch 245.59 | loss  2.29 | val loss  3.55
| epoch  51 | [  599/  985] | ms/batch 246.00 | loss  2.34 | val loss  3.74
| epoch  51 | [  799/  985] | ms/batch 245.39 | loss  2.12 | val loss  3.67
| epoch  52 | [  199/  985] | ms/batch 246.09 | loss  2.04 | val loss  3.59
| epoch  52 | [  399/  985] | ms/batch 245.69 | loss  2.32 | val loss  3.68
| epoch  52 | [  599/  985] | ms/batch 246.01 | loss  2.36 | val loss  3.60
| epoch  52 | [  799/  985] | ms/batch 245.69 | loss  2.20 | val loss  3.66
| epoch  53 | [  199/  985] | ms/batch 245.50 | loss  2.06 | val loss  3.64
| epoch  53 | [  399/  985] | ms/batch 245.83 | loss  2.27 | val loss  3.74
```

```
| epoch  53 | [   599/   985] | ms/batch 245.18 | loss   2.31 | val loss  3.71
| epoch  53 | [   799/   985] | ms/batch 245.62 | loss   2.14 | val loss  3.63
| epoch  54 | [   199/   985] | ms/batch 245.82 | loss   1.98 | val loss  3.69
| epoch  54 | [   399/   985] | ms/batch 244.68 | loss   2.30 | val loss  3.55
| epoch  54 | [   599/   985] | ms/batch 244.99 | loss   2.28 | val loss  3.72
| epoch  54 | [   799/   985] | ms/batch 245.13 | loss   2.14 | val loss  3.60
| epoch  55 | [   199/   985] | ms/batch 245.12 | loss   2.02 | val loss  3.66
| epoch  55 | [   399/   985] | ms/batch 245.05 | loss   2.28 | val loss  3.66
| epoch  55 | [   599/   985] | ms/batch 244.59 | loss   2.27 | val loss  3.64
| epoch  55 | [   799/   985] | ms/batch 245.44 | loss   2.07 | val loss  3.69
| epoch  56 | [   199/   985] | ms/batch 245.81 | loss   1.96 | val loss  3.78
| epoch  56 | [   399/   985] | ms/batch 246.24 | loss   2.16 | val loss  3.64
| epoch  56 | [   599/   985] | ms/batch 245.46 | loss   2.27 | val loss  3.63
| epoch  56 | [   799/   985] | ms/batch 245.88 | loss   2.08 | val loss  3.75
| epoch  57 | [   199/   985] | ms/batch 245.05 | loss   1.85 | val loss  3.83
| epoch  57 | [   399/   985] | ms/batch 245.47 | loss   2.22 | val loss  3.71
| epoch  57 | [   599/   985] | ms/batch 248.29 | loss   2.27 | val loss  3.67
| epoch  57 | [   799/   985] | ms/batch 248.71 | loss   2.00 | val loss  3.81
| epoch  58 | [   199/   985] | ms/batch 246.17 | loss   1.96 | val loss  3.81
| epoch  58 | [   399/   985] | ms/batch 246.07 | loss   2.26 | val loss  3.74
| epoch  58 | [   599/   985] | ms/batch 246.82 | loss   2.18 | val loss  3.72
| epoch  58 | [   799/   985] | ms/batch 246.00 | loss   1.98 | val loss  3.78
| epoch  59 | [   199/   985] | ms/batch 246.16 | loss   1.93 | val loss  3.79
| epoch  59 | [   399/   985] | ms/batch 245.59 | loss   2.20 | val loss  3.63
| epoch  59 | [   599/   985] | ms/batch 245.78 | loss   2.20 | val loss  3.76
| epoch  59 | [   799/   985] | ms/batch 246.80 | loss   2.05 | val loss  3.89
| epoch  60 | [   199/   985] | ms/batch 247.06 | loss   1.90 | val loss  3.78
| epoch  60 | [   399/   985] | ms/batch 247.55 | loss   2.11 | val loss  3.67
| epoch  60 | [   599/   985] | ms/batch 246.77 | loss   2.17 | val loss  3.64
| epoch  60 | [   799/   985] | ms/batch 247.16 | loss   1.96 | val loss  3.77
| epoch  61 | [   199/   985] | ms/batch 247.92 | loss   1.91 | val loss  3.75
| epoch  61 | [   399/   985] | ms/batch 247.83 | loss   2.12 | val loss  3.76
| epoch  61 | [   599/   985] | ms/batch 247.94 | loss   2.17 | val loss  3.78
| epoch  61 | [   799/   985] | ms/batch 247.50 | loss   1.96 | val loss  3.87
| epoch  62 | [   199/   985] | ms/batch 246.81 | loss   1.88 | val loss  3.68
| epoch  62 | [   399/   985] | ms/batch 247.00 | loss   2.09 | val loss  3.77
| epoch  62 | [   599/   985] | ms/batch 246.80 | loss   2.15 | val loss  3.83
| epoch  62 | [   799/   985] | ms/batch 247.36 | loss   1.89 | val loss  3.87
| epoch  63 | [   199/   985] | ms/batch 246.88 | loss   1.88 | val loss  3.81
| epoch  63 | [   399/   985] | ms/batch 247.51 | loss   2.07 | val loss  3.87
| epoch  63 | [   599/   985] | ms/batch 246.72 | loss   2.18 | val loss  3.92
| epoch  63 | [   799/   985] | ms/batch 247.59 | loss   1.99 | val loss  3.81
| epoch  64 | [   199/   985] | ms/batch 247.64 | loss   1.88 | val loss  3.82
| epoch  64 | [   399/   985] | ms/batch 248.46 | loss   2.14 | val loss  3.86
| epoch  64 | [   599/   985] | ms/batch 248.16 | loss   2.13 | val loss  3.85
| epoch  64 | [   799/   985] | ms/batch 248.45 | loss   1.91 | val loss  3.80
| epoch  65 | [   199/   985] | ms/batch 247.42 | loss   1.79 | val loss  3.92
| epoch  65 | [   399/   985] | ms/batch 247.82 | loss   2.06 | val loss  3.85
```

```
| epoch  65 | [  599/  985] | ms/batch 247.24 | loss  2.13 | val loss  3.90
| epoch  65 | [  799/  985] | ms/batch 248.80 | loss  1.93 | val loss  3.92
| epoch  66 | [  199/  985] | ms/batch 248.24 | loss  1.80 | val loss  3.87
| epoch  66 | [  399/  985] | ms/batch 248.89 | loss  2.01 | val loss  3.92
| epoch  66 | [  599/  985] | ms/batch 248.64 | loss  2.06 | val loss  3.99
| epoch  66 | [  799/  985] | ms/batch 249.86 | loss  2.00 | val loss  3.73
| epoch  67 | [  199/  985] | ms/batch 248.36 | loss  1.77 | val loss  3.94
| epoch  67 | [  399/  985] | ms/batch 248.62 | loss  2.06 | val loss  3.94
| epoch  67 | [  599/  985] | ms/batch 249.21 | loss  2.05 | val loss  3.93
| epoch  67 | [  799/  985] | ms/batch 248.85 | loss  1.94 | val loss  3.80
| epoch  68 | [  199/  985] | ms/batch 247.82 | loss  1.77 | val loss  3.93
| epoch  68 | [  399/  985] | ms/batch 247.94 | loss  2.00 | val loss  3.92
| epoch  68 | [  599/  985] | ms/batch 248.12 | loss  2.03 | val loss  3.99
| epoch  68 | [  799/  985] | ms/batch 248.98 | loss  1.93 | val loss  3.98
| epoch  69 | [  199/  985] | ms/batch 248.87 | loss  1.78 | val loss  3.89
| epoch  69 | [  399/  985] | ms/batch 248.49 | loss  1.99 | val loss  3.90
| epoch  69 | [  599/  985] | ms/batch 249.55 | loss  2.00 | val loss  3.88
| epoch  69 | [  799/  985] | ms/batch 247.91 | loss  1.81 | val loss  3.97
| epoch  70 | [  199/  985] | ms/batch 246.70 | loss  1.76 | val loss  3.97
| epoch  70 | [  399/  985] | ms/batch 247.57 | loss  2.05 | val loss  3.95
| epoch  70 | [  599/  985] | ms/batch 247.41 | loss  2.00 | val loss  3.88
| epoch  70 | [  799/  985] | ms/batch 247.07 | loss  1.90 | val loss  4.17
| epoch  71 | [  199/  985] | ms/batch 248.25 | loss  1.68 | val loss  3.99
| epoch  71 | [  399/  985] | ms/batch 249.27 | loss  1.90 | val loss  3.99
| epoch  71 | [  599/  985] | ms/batch 249.17 | loss  1.92 | val loss  3.97
| epoch  71 | [  799/  985] | ms/batch 248.58 | loss  1.81 | val loss  3.93
| epoch  72 | [  199/  985] | ms/batch 248.46 | loss  1.58 | val loss  4.09
| epoch  72 | [  399/  985] | ms/batch 248.14 | loss  1.87 | val loss  3.97
| epoch  72 | [  599/  985] | ms/batch 248.69 | loss  1.94 | val loss  4.01
| epoch  72 | [  799/  985] | ms/batch 248.90 | loss  1.76 | val loss  4.02
| epoch  73 | [  199/  985] | ms/batch 248.34 | loss  1.73 | val loss  3.97
| epoch  73 | [  399/  985] | ms/batch 247.80 | loss  1.91 | val loss  4.01
| epoch  73 | [  599/  985] | ms/batch 248.58 | loss  2.00 | val loss  4.07
| epoch  73 | [  799/  985] | ms/batch 248.09 | loss  1.78 | val loss  3.95
| epoch  74 | [  199/  985] | ms/batch 248.35 | loss  1.66 | val loss  3.98
| epoch  74 | [  399/  985] | ms/batch 248.99 | loss  1.95 | val loss  3.92
| epoch  74 | [  599/  985] | ms/batch 247.78 | loss  1.95 | val loss  3.93
| epoch  74 | [  799/  985] | ms/batch 247.67 | loss  1.76 | val loss  4.08
| epoch  75 | [  199/  985] | ms/batch 249.44 | loss  1.63 | val loss  4.13
| epoch  75 | [  399/  985] | ms/batch 248.41 | loss  1.89 | val loss  4.03
| epoch  75 | [  599/  985] | ms/batch 248.06 | loss  1.89 | val loss  4.09
| epoch  75 | [  799/  985] | ms/batch 248.12 | loss  1.76 | val loss  4.06
| epoch  76 | [  199/  985] | ms/batch 248.43 | loss  1.60 | val loss  4.14
| epoch  76 | [  399/  985] | ms/batch 248.18 | loss  1.87 | val loss  4.19
| epoch  76 | [  599/  985] | ms/batch 248.54 | loss  1.91 | val loss  4.03
| epoch  76 | [  799/  985] | ms/batch 248.09 | loss  1.71 | val loss  4.14
| epoch  77 | [  199/  985] | ms/batch 247.36 | loss  1.60 | val loss  4.10
| epoch  77 | [  399/  985] | ms/batch 248.82 | loss  1.88 | val loss  4.08
```

```
| epoch  77 | [  599/  985] | ms/batch 247.96 | loss  1.84 | val loss  4.25
| epoch  77 | [  799/  985] | ms/batch 248.56 | loss  1.76 | val loss  4.24
| epoch  78 | [  199/  985] | ms/batch 252.09 | loss  1.55 | val loss  4.22
| epoch  78 | [  399/  985] | ms/batch 248.91 | loss  1.74 | val loss  4.11
| epoch  78 | [  599/  985] | ms/batch 248.31 | loss  1.89 | val loss  4.00
| epoch  78 | [  799/  985] | ms/batch 248.18 | loss  1.83 | val loss  4.19
| epoch  79 | [  199/  985] | ms/batch 251.45 | loss  1.54 | val loss  4.13
| epoch  79 | [  399/  985] | ms/batch 250.27 | loss  1.76 | val loss  4.15
| epoch  79 | [  599/  985] | ms/batch 250.61 | loss  1.85 | val loss  4.09
| epoch  79 | [  799/  985] | ms/batch 251.67 | loss  1.71 | val loss  4.03
| epoch  80 | [  199/  985] | ms/batch 250.29 | loss  1.54 | val loss  4.07
| epoch  80 | [  399/  985] | ms/batch 250.83 | loss  1.77 | val loss  4.19
| epoch  80 | [  599/  985] | ms/batch 251.29 | loss  1.71 | val loss  4.09
| epoch  80 | [  799/  985] | ms/batch 248.91 | loss  1.67 | val loss  4.13
| epoch  81 | [  199/  985] | ms/batch 249.60 | loss  1.55 | val loss  4.16
| epoch  81 | [  399/  985] | ms/batch 248.91 | loss  1.77 | val loss  4.21
| epoch  81 | [  599/  985] | ms/batch 249.08 | loss  1.83 | val loss  4.15
| epoch  81 | [  799/  985] | ms/batch 248.25 | loss  1.65 | val loss  4.09
| epoch  82 | [  199/  985] | ms/batch 248.98 | loss  1.48 | val loss  4.19
| epoch  82 | [  399/  985] | ms/batch 248.19 | loss  1.84 | val loss  4.11
| epoch  82 | [  599/  985] | ms/batch 248.06 | loss  1.78 | val loss  4.19
| epoch  82 | [  799/  985] | ms/batch 248.38 | loss  1.58 | val loss  4.28
| epoch  83 | [  199/  985] | ms/batch 247.39 | loss  1.51 | val loss  4.27
| epoch  83 | [  399/  985] | ms/batch 246.71 | loss  1.79 | val loss  4.24
| epoch  83 | [  599/  985] | ms/batch 247.87 | loss  1.72 | val loss  4.33
| epoch  83 | [  799/  985] | ms/batch 246.67 | loss  1.66 | val loss  4.17
| epoch  84 | [  199/  985] | ms/batch 246.96 | loss  1.49 | val loss  4.28
| epoch  84 | [  399/  985] | ms/batch 247.36 | loss  1.67 | val loss  4.20
| epoch  84 | [  599/  985] | ms/batch 246.73 | loss  1.77 | val loss  4.11
| epoch  84 | [  799/  985] | ms/batch 246.88 | loss  1.61 | val loss  4.29
| epoch  85 | [  199/  985] | ms/batch 247.70 | loss  1.44 | val loss  4.14
| epoch  85 | [  399/  985] | ms/batch 247.37 | loss  1.68 | val loss  4.26
| epoch  85 | [  599/  985] | ms/batch 248.25 | loss  1.64 | val loss  4.19
| epoch  85 | [  799/  985] | ms/batch 249.01 | loss  1.56 | val loss  4.23
| epoch  86 | [  199/  985] | ms/batch 248.91 | loss  1.51 | val loss  4.32
| epoch  86 | [  399/  985] | ms/batch 248.31 | loss  1.65 | val loss  4.31
| epoch  86 | [  599/  985] | ms/batch 249.76 | loss  1.69 | val loss  4.34
| epoch  86 | [  799/  985] | ms/batch 248.97 | loss  1.57 | val loss  4.29
| epoch  87 | [  199/  985] | ms/batch 248.57 | loss  1.51 | val loss  4.26
| epoch  87 | [  399/  985] | ms/batch 249.17 | loss  1.62 | val loss  4.33
| epoch  87 | [  599/  985] | ms/batch 246.95 | loss  1.65 | val loss  4.35
| epoch  87 | [  799/  985] | ms/batch 246.70 | loss  1.54 | val loss  4.23
| epoch  88 | [  199/  985] | ms/batch 247.84 | loss  1.46 | val loss  4.40
| epoch  88 | [  399/  985] | ms/batch 248.33 | loss  1.70 | val loss  4.26
| epoch  88 | [  599/  985] | ms/batch 248.31 | loss  1.63 | val loss  4.32
| epoch  88 | [  799/  985] | ms/batch 248.33 | loss  1.64 | val loss  4.36
| epoch  89 | [  199/  985] | ms/batch 248.66 | loss  1.43 | val loss  4.40
| epoch  89 | [  399/  985] | ms/batch 248.12 | loss  1.55 | val loss  4.28
```

```
| epoch  89 | [  599/  985] | ms/batch 245.73 | loss  1.70 | val loss  4.26
| epoch  89 | [  799/  985] | ms/batch 246.46 | loss  1.53 | val loss  4.32
| epoch  90 | [  199/  985] | ms/batch 246.01 | loss  1.34 | val loss  4.34
| epoch  90 | [  399/  985] | ms/batch 243.69 | loss  1.60 | val loss  4.31
| epoch  90 | [  599/  985] | ms/batch 244.97 | loss  1.64 | val loss  4.34
| epoch  90 | [  799/  985] | ms/batch 242.83 | loss  1.48 | val loss  4.42
| epoch  91 | [  199/  985] | ms/batch 242.54 | loss  1.43 | val loss  4.22
| epoch  91 | [  399/  985] | ms/batch 243.09 | loss  1.65 | val loss  4.22
| epoch  91 | [  599/  985] | ms/batch 243.47 | loss  1.74 | val loss  4.17
| epoch  91 | [  799/  985] | ms/batch 243.63 | loss  1.50 | val loss  4.34
| epoch  92 | [  199/  985] | ms/batch 244.74 | loss  1.41 | val loss  4.51
| epoch  92 | [  399/  985] | ms/batch 244.68 | loss  1.49 | val loss  4.34
| epoch  92 | [  599/  985] | ms/batch 244.42 | loss  1.64 | val loss  4.52
| epoch  92 | [  799/  985] | ms/batch 244.45 | loss  1.57 | val loss  4.40
| epoch  93 | [  199/  985] | ms/batch 243.07 | loss  1.31 | val loss  4.26
| epoch  93 | [  399/  985] | ms/batch 242.90 | loss  1.59 | val loss  4.29
| epoch  93 | [  599/  985] | ms/batch 243.91 | loss  1.59 | val loss  4.49
| epoch  93 | [  799/  985] | ms/batch 242.87 | loss  1.50 | val loss  4.36
| epoch  94 | [  199/  985] | ms/batch 243.11 | loss  1.31 | val loss  4.56
| epoch  94 | [  399/  985] | ms/batch 244.01 | loss  1.44 | val loss  4.48
| epoch  94 | [  599/  985] | ms/batch 243.27 | loss  1.56 | val loss  4.37
| epoch  94 | [  799/  985] | ms/batch 243.22 | loss  1.46 | val loss  4.44
| epoch  95 | [  199/  985] | ms/batch 244.01 | loss  1.30 | val loss  4.34
| epoch  95 | [  399/  985] | ms/batch 243.47 | loss  1.62 | val loss  4.23
| epoch  95 | [  599/  985] | ms/batch 243.21 | loss  1.65 | val loss  4.54
| epoch  95 | [  799/  985] | ms/batch 242.98 | loss  1.46 | val loss  4.53
| epoch  96 | [  199/  985] | ms/batch 243.32 | loss  1.30 | val loss  4.55
| epoch  96 | [  399/  985] | ms/batch 242.68 | loss  1.46 | val loss  4.43
| epoch  96 | [  599/  985] | ms/batch 243.85 | loss  1.43 | val loss  4.34
| epoch  96 | [  799/  985] | ms/batch 245.39 | loss  1.46 | val loss  4.55
| epoch  97 | [  199/  985] | ms/batch 244.68 | loss  1.24 | val loss  4.49
| epoch  97 | [  399/  985] | ms/batch 244.74 | loss  1.46 | val loss  4.36
| epoch  97 | [  599/  985] | ms/batch 245.59 | loss  1.63 | val loss  4.46
| epoch  97 | [  799/  985] | ms/batch 245.00 | loss  1.38 | val loss  4.61
| epoch  98 | [  199/  985] | ms/batch 245.03 | loss  1.32 | val loss  4.57
| epoch  98 | [  399/  985] | ms/batch 245.50 | loss  1.63 | val loss  4.53
| epoch  98 | [  599/  985] | ms/batch 242.46 | loss  1.58 | val loss  4.28
| epoch  98 | [  799/  985] | ms/batch 243.23 | loss  1.38 | val loss  4.54
| epoch  99 | [  199/  985] | ms/batch 245.77 | loss  1.34 | val loss  4.62
| epoch  99 | [  399/  985] | ms/batch 243.09 | loss  1.53 | val loss  4.49
| epoch  99 | [  599/  985] | ms/batch 242.85 | loss  1.58 | val loss  4.81
| epoch  99 | [  799/  985] | ms/batch 244.12 | loss  1.39 | val loss  4.49
Original: love this shirt
Predicted: s cotton t shirt rugged long


Original: daughter loves it
Predicted: capezio women s socks are special
```

```
Original: i bought for my daughter
Predicted: linguri linguri linguri is true to


Original: very nice sleeveless shirts
Predicted: s sexy and quality good


Original: such a cool belt
Predicted: a little bit cheap looking but it has


bleu, precisions, bp, ratio, translation_length, reference_length (0.0,
[0.22711127204147336, 0.0, 0.0, 0.0], 1.0, 33.85662962962963, 914129, 27000)
rouge2 (0.1780322698616376, 0.172686230248307, 0.18371990140934083)
rouge {'rouge_1/f_score': 0.05149547597277218, 'rouge_1/r_score':
0.07128905055849499, 'rouge_1/p_score': 0.045626734273956485, 'rouge_2/f_score':
0.005826158524531902, 'rouge_2/r_score': 0.00920049970605526, 'rouge_2/p_score':
0.00494594356261023, 'rouge_l/f_score': 0.0401753149587903, 'rouge_l/r_score':
0.06868816872427982, 'rouge_l/p_score': 0.03976701940035273}
```

[ ]:
```python
#get the final cleaned data
df=pd.read_csv('/content/drive/MyDrive/product_reviews.csv')[:117799]
print("The length of dataset is ",len(df))

#set the threshold
threshold = 20
max_rl=80 #maximum review length
max_sl=10 #maximum summary length

#get reviewText whose length is less than maximum review length
df['reviewText']=df['reviewText'].str.slice(0,max_rl)

#get summary whose length is less than maximum summary length
df['summary']=df['summary'].str.slice(0,max_rl)
```

[ ]:
```python
'''
f = open("/content/drive/MyDrive/TFIVE.txt", "r")
text=f.readlines()
text=pd.DataFrame(text,columns=["value"])
text=text["value"].str.split("\t",expand=True)
text.columns=["predicted","value","original"]
text.drop(columns=["value"],inplace=True)
text["predicted"]=text["predicted"].str.split(":").str[1]
text["original"]=text["original"].str.split(":").str[1]
```

```
text["original"]=text["original"].replace('\n','', regex=True)
'''
```

[ ]:
```
df[df["summary"]=='best birthday gift ever']
```

[ ]:
```
df["reviewText"][21619]
```

[ ]:
```
df["reviewText"][86599]
```

[ ]:
```
df["original"][0]
```

[ ]:
```

```