Sequence to sequence(Enoder-Decoder) Model implementation for Machine TransLation.

In [1]:

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

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

In [5]:

```
! unzip "/content/drive/MyDrive/1. My_folder/Seq_model/ita-eng.zip"
```

```
Archive:  /content/drive/MyDrive/1. My_folder/Seq_model/ita-eng.zip
  inflating: ita.txt
  inflating: _about.txt
```

Loading and Preprocessing the data From original data file.

In [2]:

```
import pandas as pd
import numpy as np
import re
```

In [8]:

```
with open("/content/ita.txt", 'r', encoding="utf8") as f:
    eng=[]
    ita=[]
    for i in f.readlines():
        eng.append(i.split("\t")[0])
        ita.append(i.split("\t")[1])
data = pd.DataFrame(data=list(zip(eng, ita)), columns=['english','italian'])
print("Shape of data is :",data.shape)
```

```
Shape of data is : (343813, 2)
```

In [9]:
```
data.head()
```

Out[9]:

	english	italian
0	Hi.	Ciao!
1	Run!	Corri!
2	Run!	Corra!
3	Run!	Correte!
4	Who?	Chi?

In [10]:

```
def decontractions(phrase):
    """decontracted takes text and convert contractions into natural form.
     ref: https://stackoverflow.com/questions/19790188/expanding-english-language-contractions-in-python/4709149
    # specific
    phrase = re.sub(r"won\'t", "will not", phrase)
    phrase = re.sub(r"can\'t", "can not", phrase)
    phrase = re.sub(r"won\’t", "will not", phrase)
    phrase = re.sub(r"can\’t", "can not", phrase)

    # general
    phrase = re.sub(r"n\'t", " not", phrase)
    phrase = re.sub(r"\'re", " are", phrase)
    phrase = re.sub(r"\'s", " is", phrase)
    phrase = re.sub(r"\'d", " would", phrase)
    phrase = re.sub(r"\'ll", " will", phrase)
    phrase = re.sub(r"\'t", " not", phrase)
    phrase = re.sub(r"\'ve", " have", phrase)
    phrase = re.sub(r"\'m", " am", phrase)

    phrase = re.sub(r"n\’t", " not", phrase)
    phrase = re.sub(r"\’re", " are", phrase)
    phrase = re.sub(r"\’s", " is", phrase)
    phrase = re.sub(r"\’d", " would", phrase)
    phrase = re.sub(r"\’ll", " will", phrase)
    phrase = re.sub(r"\’t", " not", phrase)
    phrase = re.sub(r"\’ve", " have", phrase)
    phrase = re.sub(r"\’m", " am", phrase)

    return phrase

def preprocess(text):
    # convert all the text into lower letters
    # use this function to remove the contractions: https://gist.github.com/anandborad/d410a49a493b56dace4f814ab
    # remove all the spacial characters: except space ' '
    text = text.lower()
    text = decontractions(text)
    text = re.sub('[^A-Za-z0-9 ]+', '', text)
    return text

def preprocess_ita(text):
    # convert all the text into lower letters
    # remove the words betweent brakets ()
    # remove these characters: {'$', ')', '?', '"', '’', '.',  '°', '!', ';', '/', "'", '€', '%', ':', ',', '('}
    # replace these spl characters with space: '\u200b', '\xa0', '-', '/'
    # we have found these characters after observing the data points, feel free to explore more and see if you c
    # you are free to do more proprocessing
    # note that the model will learn better with better preprocessed data

    text = text.lower()
    text = decontractions(text)
    text = re.sub('[$)\?"’.°!;\'€%:,(/]', '', text)
    text = re.sub('\u200b', ' ', text)
    text = re.sub('\xa0', ' ', text)
    text = re.sub('-', ' ', text)
    return text


data['english'] = data['english'].apply(preprocess)
data['italian'] = data['italian'].apply(preprocess_ita)
data.head()
```

Out[10]:

	english	italian
0	hi	ciao
1	run	corri
2	run	corra
3	run	correte
4	who	chi

In [11]:

```
data['italian_len'] = data['italian'].str.split().apply(len)
data = data[data['italian_len'] < 20]

data['english_len'] = data['english'].str.split().apply(len)
data = data[data['english_len'] < 20]

data['english_inp'] = '<start> ' + data['english'].astype(str)
data['english_out'] = data['english'].astype(str) + ' <end>'

data = data.drop(['english','italian_len','english_len'], axis=1)

data.head()
```

Out[11]:

	italian	english_inp	english_out
0	ciao	<start> hi	hi <end>
1	corri	<start> run	run <end>
2	corra	<start> run	run <end>
3	correte	<start> run	run <end>
4	chi	<start> who	who <end>

In [12]:
```
data.sample(10)
```

Out[12]:

	italian	english_inp	english_out
340214	se qualcuno dovesse telefonare dite che torner...	<start> if anyone should phone say i will be b...	if anyone should phone say i will be back at o...
65245	tom ha superato i trentanni	<start> tom is past thirty	tom is past thirty <end>
85450	lavete già venduta	<start> have you sold it yet	have you sold it yet <end>
138778	tom è un pianista di talento	<start> tom is a gifted pianist	tom is a gifted pianist <end>
314698	lui gioca a golf due o tre volte al mese	<start> he plays golf two or three times a month	he plays golf two or three times a month <end>
137891	questa casa è abbandonata	<start> this house is abandoned	this house is abandoned <end>
28597	ora sono preoccupata	<start> now i am worried	now i am worried <end>
5613	sono assetato	<start> i am thirsty	i am thirsty <end>
249775	io ho fatto tre fuoricampo lanno scorso	<start> i hit three home runs last year	i hit three home runs last year <end>
158056	perché andrei a boston	<start> why would i go to boston	why would i go to boston <end>

In [13]:
```
# Saving data so we don't need to perform preprocessing again
data.to_csv("preprocessed_data.csv",index= False)
```

Loading Preproces data.

In [43]:
```
import pandas as pd
data = pd.read_csv("preprocessed_data.csv")
print("Shape of data is :",data.shape)
```

```
Shape of data is : (343388, 3)
```

In [45]:
```
data.head(3)
```

Out[45]:

	italian	english_inp	english_out
0	ciao	<start> hi	hi <end>
1	corri	<start> run	run <end>
2	corra	<start> run	run <end>

Getting Train and Test Data.

In [46]:

```
from sklearn.model_selection import train_test_split
train, validation = train_test_split(data, test_size=0.2)
print("Shape of train data is :",train.shape)
print("Shape of validation data is :",validation.shape)
```

```
Shape of train data is : (274710, 3)
Shape of validation data is : (68678, 3)
```

In [47]:

```
# for one sentence we will be adding <end> token so that the tokanizer learns the word <end>
# with this we can use only one tokenizer for both encoder output and decoder output
train.iloc[0]['english_inp']= str(train.iloc[0]['english_inp'])+' <end>'
train.iloc[0]['english_out']= str(train.iloc[0]['english_out'])+' <end>'
```

In [50]:

```
print("Train Data Head :")
print("-"*100)
train.head(3)
```

```
Train Data Head :
----------------------------------------------------------------------------------------------------
```

Out[50]:

	italian	english_inp	english_out
231290	non ci saremmo mai dovuti arrendere	<start> we should never have given up <end>	we should never have given up <end> <end>
113657	vengono forniti servizi per linfanzia	<start> child care is provided	child care is provided <end>
39725	non sono miei	<start> they are not mine	they are not mine <end>

In [19]:

```
print("validation Data Head :")
print("-"*100)
validation.head(3)
```

```
validation Data Head :
----------------------------------------------------------------------------------------------------
```

Out[19]:

	italian	english_inp	english_out
711	lho mangiata	<start> i ate it	i ate it <end>
1193	vieni presto	<start> come soon	come soon <end>
784	io sono restata	<start> i stayed	i stayed <end>

Creating Tokenizer on the train data and learning vocabulary.

In [51]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [52]:

```
tknizer_ita = Tokenizer()
tknizer_ita.fit_on_texts(train['italian'].values)
encoder_seq = tknizer_ita.texts_to_sequences(train['italian'].values)
max_len_ita = 20
padded_italian = pad_sequences(encoder_seq, maxlen=max_len_ita, dtype='int32', padding='post')
```

In [53]:

```
# For validation data
encoder_seq = tknizer_ita.texts_to_sequences(validation['italian'].values)
val_padded_italian = pad_sequences(encoder_seq, maxlen=max_len_ita, dtype='int32', padding='post')
```

In [54]:

```
tknizer_eng = Tokenizer(filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n')
tknizer_eng.fit_on_texts(train['english_inp'].values)
decoder_inp_seq = tknizer_eng.texts_to_sequences(train['english_inp'].values)
max_len_eng = 20
padded_input_english = pad_sequences(decoder_inp_seq, maxlen=max_len_eng, dtype='int32', padding='post')
```

In [55]:
```
# For validation data
seq = tknizer_eng.texts_to_sequences(validation['english_inp'].values)
val_padded_input_english = pad_sequences(seq, maxlen=max_len_eng, dtype='int32', padding='post')
```

In [56]:
```
# For Decoder_output
decoder_out_seq = tknizer_eng.texts_to_sequences(train['english_out'].values)
padded_output_english = pad_sequences(decoder_out_seq, maxlen=max_len_eng, dtype='int32', padding='post')
```

In [57]:
```
# For validation data
seq = tknizer_eng.texts_to_sequences(validation['english_out'].values)
val_padded_output_english = pad_sequences(seq, maxlen=max_len_eng, dtype='int32', padding='post')
```

In [59]:
```
vocab_size_ita=len(tknizer_ita.word_index.keys())+1
print("Vocab size of Italian Sentences is :",vocab_size_ita)
vocab_size_eng=len(tknizer_eng.word_index.keys())+1
print("-"*100)
print("Vocab size of English Sentences is :",vocab_size_eng)
```

```
Vocab size of Italian Sentences is : 26219
----------------------------------------------------------------------------------------------------
Vocab size of English Sentences is : 12852
```

In [60]:
```
start_word_index = tknizer_eng.word_index['<start>']
print("Index of Start token in english is :",start_word_index)
print("-"*100)
end_word_index = tknizer_eng.word_index['<end>']
print("Index of end token in english is :",end_word_index)
```

```
Index of Start token in english is : 1
----------------------------------------------------------------------------------------------------
Index of end token in english is : 10106
```

In [61]:
```
eng_index_to_word={}
for key,value in tknizer_eng.word_index.items():
    eng_index_to_word[value]=key
```

In [62]:
```
padded_italian.shape
```

Out[62]:
```
(274710, 20)
```

In [63]:
```
padded_input_english.shape
```

Out[63]:
```
(274710, 20)
```

In [64]:
```
padded_output_english.shape
```

Out[64]:
```
(274710, 20)
```

Implementing custom encoder Layer

In [65]:
```
import pandas as pd
import re
import tensorflow as tf
from tensorflow.keras.layers import Embedding, LSTM, Dense
from tensorflow.keras.models import Model
import numpy as np
```

In [66]:
```
class Encoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,encoder_final_state_h,encoder_final
    '''

    def __init__(self,inp_vocab_size,embedding_size,lstm_size,input_length):
        super().__init__()
        self.lstm_output = 0
        self.lstm_state_h=0
        self.lstm_state_c=0
        self.lstm_size = lstm_size
        #Initialize Embedding layer
        self.embedding = Embedding(input_dim = inp_vocab_size, output_dim = embedding_size,
                                   input_length = input_length,
                                   mask_zero=True, name="embedding_layer_encoder")
        #Intialize Encoder LSTM layer
        self.lstm = LSTM(lstm_size, return_state=True, return_sequences=True, name="Encoder_LSTM")


    def call(self,input_sequence,states):

        '''
          This function takes a sequence input and the initial states of the encoder.
          Pass the input_sequence input to the Embedding layer, Pass the embedding layer ouput to encoder_lstm
          returns -- encoder_output, last time step's hidden and cell state
        '''

        input_embedd = self.embedding(input_sequence)

        self.lstm_output, self.lstm_state_h,self.lstm_state_c = self.lstm(input_embedd)

        return self.lstm_output, self.lstm_state_h,self.lstm_state_c

    def initialize_states(self,batch_size):

        return (tf.zeros([batch_size, self.lstm_size]),
              tf.zeros([batch_size, self.lstm_size]))
```

Implementing custom Decoder Layer

In [68]:

```
class Decoder(tf.keras.Model):
    '''
    Encoder model -- That takes a input sequence and returns output sequence
    '''

    def __init__(self,out_vocab_size,embedding_size,lstm_size,input_length):
        super().__init__()
        self.vocab_size = out_vocab_size
        self.embedding_dim = embedding_size
        self.lstm_size = lstm_size
        self.input_length = input_length

        #Initialize Embedding layer
        self.embedding = Embedding(input_dim=self.vocab_size, output_dim=self.embedding_dim,
                                   input_length=self.input_length,
                                   mask_zero=True, name="embedding_layer_decoder")

        #Intialize Decoder LSTM layer
        self.lstm = LSTM(self.lstm_size, return_sequences=True, return_state=True, name="Encoder_LSTM")


    def call(self,input_sequence,initial_states):

        target_embedd                           = self.embedding(input_sequence)
        lstm_output, decoder_h,decoder_c        = self.lstm(target_embedd, initial_state = initial_states)

        return lstm_output,decoder_h,decoder_c
```

Implementing custom Encoder-Decoder Model.

In [71]:
```
class Encoder_decoder(tf.keras.Model):

    def __init__(self,encoder_inputs_length,decoder_inputs_length,output_vocab_size,vocab_size_ita,vocab_size_er
        super().__init__()
        self.vocab_size_ita = vocab_size_ita
        self.encoder_inputs_length = encoder_inputs_length
        self.vocab_size_eng = vocab_size_eng
        self.decoder_inputs_length = decoder_inputs_length
        self.output_vocab_size = output_vocab_size
        #Create encoder object
        self.encoder = Encoder(inp_vocab_size=self.vocab_size_ita, embedding_size=100 , lstm_size = 256 ,
                               input_length=self.encoder_inputs_length)

        #Create decoder object
        self.decoder = Decoder(out_vocab_size=self.vocab_size_eng , embedding_size=100, lstm_size = 256 ,
                               input_length=self.decoder_inputs_length)

        #Intialize Dense layer(out_vocab_size) with activation='softmax'
        self.dense   = Dense(self.output_vocab_size, activation='softmax')

    def call(self,data):

        input,output = data[0], data[1]
        encoder_output, encoder_h, encoder_c = self.encoder(input,0)
        states = [encoder_h, encoder_c]
        decoder_output ,decoder_h,decoder_c  = self.decoder(output, states)
        output                               = self.dense(decoder_output)
        return output
```

In [72]:
```
#Create an object of encoder_decoder Model class,
model  = Encoder_decoder(encoder_inputs_length=20,decoder_inputs_length=20,
                         output_vocab_size=vocab_size_eng,
                         vocab_size_ita = vocab_size_ita,vocab_size_eng= vocab_size_eng)
```

In [73]:
```
optimizer = tf.keras.optimizers.Adam()
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy')
```

In [74]:
```
model.fit([padded_italian, padded_input_english], padded_output_english ,
          epochs = 10,
          validation_data = ([val_padded_italian,val_padded_input_english],val_padded_output_english),
          verbose = True,
          batch_size = 16 )
```

```
Epoch 1/10
17170/17170 [==============================] - 715s 41ms/step - loss: 1.1624 - val_loss: 0.4817
Epoch 2/10
17170/17170 [==============================] - 704s 41ms/step - loss: 0.3874 - val_loss: 0.3040
Epoch 3/10
17170/17170 [==============================] - 694s 40ms/step - loss: 0.2249 - val_loss: 0.2447
Epoch 4/10
17170/17170 [==============================] - 687s 40ms/step - loss: 0.1573 - val_loss: 0.2205
Epoch 5/10
17170/17170 [==============================] - 687s 40ms/step - loss: 0.1198 - val_loss: 0.2051
Epoch 6/10
17170/17170 [==============================] - 693s 40ms/step - loss: 0.0975 - val_loss: 0.1979
Epoch 7/10
17170/17170 [==============================] - 708s 41ms/step - loss: 0.0814 - val_loss: 0.1947
Epoch 8/10
17170/17170 [==============================] - 707s 41ms/step - loss: 0.0706 - val_loss: 0.1926
Epoch 9/10
17170/17170 [==============================] - 704s 41ms/step - loss: 0.0622 - val_loss: 0.1932
Epoch 10/10
17170/17170 [==============================] - 705s 41ms/step - loss: 0.0564 - val_loss: 0.1934
```

Out[74]:
```
<tensorflow.python.keras.callbacks.History at 0x7fc0738dc9e8>
```

Thanks For Coming.!! :)