Sequence to sequence implementation

There will be some functions that start with the word "grader" ex: grader_check_encoder(), grader_check_attention(), grader_onestepdecoder() etc, you should not change those function definition.

Every Grader function has to return True.

Note 1: There are many blogs on the attention mechanisum which might be misleading you, so do read the references completly and after that only please check the internet. The best things is to read the research papers and try to implement it on your own.

Note 2: To complete this assignment, the reference that are mentioned will be enough.

Note 3: If you are starting this assignment, you might have completed minimum of 20 assignment. If you are still not able to implement this algorithm you might have rushed in the previous assignments with out learning much and didn't spend your time productively.

Task -1: Simple Encoder and Decoder

Implement simple Encoder-Decoder model

In [1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

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

1. Download the **Italian** to **English** translation dataset from here (http://www.manythings.org/anki/ita-eng.zip)
2. You will find **ita.txt** file in that ZIP, you can read that data using python and preprocess that data this way only:

```
Encoder input: "<start> vado a scuola <end>"
Decoder input: "<start> i am going school"
Decoder output: "i am going school <end>"
```

3. You have to implement a simple Encoder and Decoder architecture
4. Use BLEU score as metric to evaluate your model. You can use any loss function you need.
5. You have to use Tensorboard to plot the Graph, Scores and histograms of gradients.
6. a. Check the reference notebook
   b. Resource 2 (https://medium.com/analytics-vidhya/understand-sequence-to-sequence-models-in-a-more-intuitive-way-1d517d8795bb)

Load the data

In [2]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
import re
```

In [3]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/ass-28/ita.txt', delimit
df
```

Out[3]:

	english	italian
0	Hi.	Ciao!
1	Run!	Corri!
2	Run!	Corra!
3	Run!	Correte!
4	Who?	Chi?
...	...	...
345239	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
345240	If you want to sound like a native speaker, yo...	Se vuoi sembrare un madrelingua, devi essere d...
345241	If someone who doesn't know your background sa...	Se qualcuno che non conosce il tuo background ...
345242	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...
345243	Doubtless there exists in this world precisely...	Senza dubbio esiste in questo mondo proprio la...

345244 rows × 2 columns

Preprocess data

In [4]:
```
def decontracted(phrase):
    '''This function returns the decontracted words for English Language'''
    # specific
    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)
    return phrase

def preprocessor(text):
    '''This function returns preprocessed data for English Language'''
    text = text.lower()
    text = decontracted(text)
    text = re.sub("[^A-Za-z0-9 ]+",'',text)
    return text

def preprocessor_ita(text):
    '''This function returns preprocessed data for Italian Language'''
    text = text.lower()
    text = decontracted(text)
    text = re.sub("[^A-Za-z0-9 ]+",'',text)
    return text
```

In [5]:
```
# PREPROCESSED DATA INTO A DATAFRAME
df["english"] = df.english.apply(preprocessor)
df["italian"] = df.italian.apply(preprocessor_ita)
df
```

Out[5]:

	english	italian
0	hi	ciao
1	run	corri
2	run	corra
3	run	correte
4	who	chi
...	...	...
345239	if you want to sound like a native speaker you...	se vuoi sembrare un madrelingua devi essere di...
345240	if you want to sound like a native speaker you...	se vuoi sembrare un madrelingua devi essere di...
345241	if someone who does not know your background s...	se qualcuno che non conosce il tuo background ...
345242	doubtless there exists in this world precisely...	senza dubbio esiste in questo mondo proprio la...
345243	doubtless there exists in this world precisely...	senza dubbio esiste in questo mondo proprio la...

345244 rows × 2 columns

In [6]:
```
#REMOVING SENTENCES WITH MAXIMUM LENGTH GREATER THAN 20
df["eng_len"] = df.english.apply(lambda x: len(x.split()))
df = df[df.eng_len<=20]
df["ita_len"] = df.italian.apply(lambda x: len(x.split()))
df  = df[df.ita_len<=20]

# ADDING <start> TO THE BEGINING OF ENGLISH SENTENCES
df["english_inp"]= "<start> "+ df.english
# ADDING <end> TO THE END IN ENGLISH SENTENCES
df["english_out"]= df.english+ " <end>"
df.drop(["english","eng_len","ita_len"],axis=1,inplace=True)
df
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: SettingWithCopy
Warning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/sta
ble/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pyd
ata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-c
opy)
  This is separate from the ipykernel package so we can avoid doing imports unt
il
```

Out[6]:

	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>
...	...	...	...
344915	charles moore cre il forth nel tentativo di au...	<start> charles moore created forth in an atte...	charles moore created forth in an attempt to i...
344998	se la tua azienda opera principalmente con lam...	<start> if your company primarily does busines...	if your company primarily does business with a...
344999	se la sua azienda opera principalmente con lam...	<start> if your company primarily does busines...	if your company primarily does business with a...
345000	se la vostra azienda opera principalmente con ...	<start> if your company primarily does busines...	if your company primarily does business with a...
345001	lintelligenza fondata nella capacit di ricono...	<start> intelligence is found in the capacity ...	intelligence is found in the capacity to recog...

344860 rows × 3 columns

Train_Test split

In [7]:
```
from sklearn.model_selection import train_test_split
```

In [8]:
```
train , validation = train_test_split(df,test_size = 0.2)
```

In [9]:
```
# ADDING <end> TO THE END OF FIRST ENGLISH SENTENCE IN "english_inp"
train.iloc[0]["english_inp"]  = train.iloc[0]["english_inp"] + " <end>"
```

In [9]:
```

```

Tokenize

In [10]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
```

In [11]:
```
# TOKENIZING ENGLISH SENTENCES
tk_eng = Tokenizer(filters = '!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n')
tk_eng.fit_on_texts(train.english_inp.values)
```

In [12]:
```
# TOKENIZING INALIAN SENTENCES
tk_ita = Tokenizer()
tk_ita.fit_on_texts(train.italian)
```

In [12]:
```

```

Implement custom encoder decoder

Encoder

In [13]:
```
from tensorflow.keras import layers
```

In [14]:
```
class Encoder(tf.keras.layers.Layer):
    '''
    Encoder model -- That takes a input sequence and returns encoder-outputs,encc
    '''
    def __init__(self , vocab_size , embedding_dim , enc_units , input_len):
        super().__init__()

        # STATING ALL THE VARIABLES
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.input_len = input_len
        self.enc_units = enc_units
        self.enc_output = 0
        self.state_h = 0
        self.state_c=0
        # INITALIZING EMBEDDING LAYER
        self.embedding = layers.Embedding(input_dim= self.vocab_size,
                                          output_dim = self.embedding_dim,
                                          mask_zero = True,
                                           input_length = self.input_len
                                          )
        # INTIALIZING LSTM LAYER
        self.lstm = layers.LSTM(units= self.enc_units,return_state = True,return_

    def call(self,input,state):
        '''
          This function takes a sequence input and the initial states of the encc
          Pass the input_sequence input to the Embedding layer, Pass the embeddir
          returns -- encoder_output, last time step's hidden and cell state
        '''
        # CONVERTING INPUT TO EMBEDDED VECTORS
        emb = self.embedding(input)
        # PASSING THROUGH LSTM LAYER
        self.enc_output , self.state_h , self.state_c = self.lstm(emb,initial_sta

        return self.enc_output , self.state_h , self.state_c


    def initialize_states(self,batch_size):
        '''
      Given a batch size it will return intial hidden state and intial cell state
      If batch size is 32- Hidden state is zeros of size [32,lstm_units], cell st
      '''
        # INITIALIZING THE VALUES OF H AND C STATES FOR LSTM
        initial_h = tf.zeros(shape=(batch_size,self.enc_units))
        initial_c = tf.zeros(shape=(batch_size,self.enc_units))
        return [initial_h , initial_c]
```

Grader function - 1

In [15]:
```
def grader_check_encoder():
    '''
        vocab-size: Unique words of the input language,
        embedding_size: output embedding dimension for each word after embedding
        lstm_size: Number of lstm units,
        input_length: Length of the input sentence,
        batch_size
    '''
    vocab_size=10
    embedding_size=20
    lstm_size=32
    input_length=10
    batch_size=16
    #Intialzing encoder
    encoder=Encoder(vocab_size,embedding_size,lstm_size,input_length)
    input_sequence=tf.random.uniform(shape=[batch_size,input_length],maxval=vocab
    #Intializing encoder initial states
    initial_state=encoder.initialize_states(batch_size)

    encoder_output,state_h,state_c=encoder(input_sequence,initial_state)

    assert(encoder_output.shape==(batch_size,input_length,lstm_size) and state_h.
    return True
print(grader_check_encoder())
```

```
True
```

Decoder

In [16]:

```
class Decoder(tf.keras.layers.Layer):
    def __init__(self,vocab_size , embedding_dim, dec_unit,input_len ):
        super().__init__()
        # INITALIZING ALL THE VARIABLES
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim
        self.input_len = input_len
        self.dec_unit =dec_unit

    def build(self,input_shape):

        # INITALIZING EMBEDDING AND LSTM LAYER
        self.embedding = layers.Embedding(input_dim = self.vocab_size,
                                          output_dim = self.embedding_dim,
                                         mask_zero=True,
                                         input_length = self.input_len)
        self.lstm = layers.LSTM(units = self.dec_unit,
                                return_sequences=True,
                                return_state=True)

    def call(self,input, state):
        # FORMING THE EMBEDDED VECTORS
        emb = self.embedding(input)

        # LSTM OUTPUT
        dec_out,state_h,state_c = self.lstm(emb,initial_state = state)

        return dec_out,state_h,state_c
```

Grader function - 2

In [17]:
```
def grader_decoder():
    '''
        out_vocab_size: Unique words of the target language,
        embedding_size: output embedding dimension for each word after embedding
        dec_units: Number of lstm units in decoder,
        input_length: Length of the input sentence,
        batch_size


    '''
    out_vocab_size=13
    embedding_dim=12
    input_length=10
    dec_units=16
    batch_size=32

    target_sentences=tf.random.uniform(shape=(batch_size,input_length),maxval=10,
    encoder_output=tf.random.uniform(shape=[batch_size,input_length,dec_units])
    state_h=tf.random.uniform(shape=[batch_size,dec_units])
    state_c=tf.random.uniform(shape=[batch_size,dec_units])
    states=[state_h,state_c]
    decoder=Decoder(out_vocab_size, embedding_dim, dec_units,input_length )
    output,_,_=decoder(target_sentences, states)
    assert(output.shape==(batch_size,input_length,dec_units))
    return True
print(grader_decoder())
```

```
True
```

ENCODER_DECODER

In [18]:
```
class Encoder_decoder(tf.keras.Model):
    '''
    ARGUMENTS:
    enc_vocab_size,
    enc_emb_dim,
    enc_units,
    enc_input_length,
    dec_vocab_size,
    dec_emb_dim,
    dec_units,
    dec_input_length,
    batch_size'''
    def __init__(self,enc_vocab_size,enc_emb_dim,enc_units,enc_input_length,dec_v
        super().__init__()
        # INITIALIZING ALL REQUIRED VARIABLES
        # BATCH SIZE
        self.batch_size = batch_size
        # ENCODER MODEL
        self.encoder = Encoder(vocab_size= enc_vocab_size , embedding_dim= enc_em
                               enc_units=enc_units ,input_len=enc_input_length)
        # DECODER MODEL
        self.decoder = Decoder(vocab_size = dec_vocab_size  , embedding_dim = dec
                               dec_unit=dec_units ,input_len=dec_input_length)
        # DENSE LAYER
        self.dense = layers.Dense(dec_vocab_size,activation = "softmax")

    def call(self,data):
        '''
        A. Pass the input sequence to Encoder layer -- Return encoder_output,encc
        B. Pass the target sequence to Decoder layer with intial states as encode
        C. Pass the decoder_outputs into Dense layer

        Return decoder_outputs
        '''
        # GETTING THE INPUT FOR ENCODER AND DECODER
        input,output  = data[0],data[1]

        # INITIAL STATES FOR ENCODER METHOD
        initial_states = self.encoder.initialize_states(self.batch_size)
        # PASSING THE INPUT AND INTIAL STATES TO ENCODER
        enc_output,state_h,state_c = self.encoder(input,initial_states)

        enc_states = [state_h,state_c]
        # PASSING DECODER INPUT AND ENDOER OUTPUT STATES TO DECODER
        dec_output,_,_ = self.decoder(output,enc_states)
        # PASSING DECODER OUTPUT TO DENSE LATER
        dense_output = self.dense(dec_output)

        # RETURNING DENSE OUTPUT
        return dense_output
```

Data Generator

In [19]:
```
from  tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [20]:
```
class Dataset :
    def __init__(self, data,tk_eng,tk_ita,max_len):
        self.encoder_inp = data["italian"].values
        self.decoder_inp = data["english_inp"].values
        self.decoder_out = data["english_out"].values
        self.tk_eng = tk_eng
        self.tk_ita = tk_ita
        self.max_len = max_len

    def __getitem__(self,i):
        # ITALIAN TO INTEGER SEQUENCES
        self.encoder_seq = self.tk_ita.texts_to_sequences([self.encoder_inp[i]])
        # ENGLISH TO INTEGER SEQUENCES
        self.decoder_inp_seq = self.tk_eng.texts_to_sequences([self.decoder_inp[i
        # ENGLISH TO INTEGER SEQUENCES
        self.decoder_out_seq = self.tk_eng.texts_to_sequences([self.decoder_out[i

        # PADDING THE ENCODER INPUT SEQUENCES
        self.encoder_seq = pad_sequences(self.encoder_seq,maxlen = self.max_len,p
        # PADDING THE DECODER INPUT SEQUENCES
        self.decoder_inp_seq = pad_sequences(self.decoder_inp_seq,maxlen = self.m
        # PADDING DECODER OUTPUT SEQUENCES
        self.decoder_out_seq = pad_sequences(self.decoder_out_seq,maxlen = self.m
        return self.encoder_seq ,  self.decoder_inp_seq,  self.decoder_out_seq

    def __len__(self):
        # RETURN THE LEN OF INPUT ENDODER
        return len(self.encoder_inp)
```

In [21]:
```
class Dataloader(tf.keras.utils.Sequence):
    def __init__(self,batch_size,dataset):
        # INTIALIZING THE REQUIRED VARIABLES
        self.dataset = dataset
        self.batch_size = batch_size
        self.totl_points = self.dataset.encoder_inp.shape[0]

    def __getitem__(self,i):
        # STATING THE START AND STOP VATIABLE CONTAINGING INDEX VALUES FOR EACH E
        start = i * self.batch_size
        stop = (i+1)*self.batch_size

        # PLACEHOLDERS FOR BATCHED DATA
        batch_ita =[]
        batch_eng_input = []
        batch_eng_out =[]

        for j in range(start,stop): # FOR EACH VALUE IN START TO STOP

            a,b,c = self.dataset[j] # DATASET RETURNS ITALIAN , ENGLIGH_INPUT, EN
            batch_ita.append(a[0]) # APPENDING ITALIAN TO batch_ita
            batch_eng_input.append(b[0]) # APPENGIND ENGLISH INPUT TO batch_eng_i
            batch_eng_out.append(c[0]) # APPENDING ENGLISH OUTPUT TO batch_eng_ou

        # Conveting list to array
        batch_ita = (np.array(batch_ita))
        batch_eng_input = np.array(batch_eng_input)
        batch_eng_out = np.array(batch_eng_out)

        return [batch_ita , batch_eng_input],batch_eng_out

    def __len__(self):
        # Returning the number of batches
        return int(self.totl_points/self.batch_size)
```

In [22]:
```
# FORMING OBJECTS OF DATASET AND DATALOADER FOR TRAIN DATASET
train_dataset = Dataset(train,tk_eng,tk_ita,20)
train_dataloader = Dataloader( batch_size = 1024 , dataset=train_dataset)
```

In [23]:
```
# FORMING OBJECTS OF DATASET AND DATALOADER FOR TEST DATASET
val_dataset = Dataset(validation,tk_eng,tk_ita,20)
val_dataloader = Dataloader(batch_size=1024,dataset=val_dataset)
```

In [24]:
```
# TRAINING THE MODEL FOR 50 EPOCHS
model1 = Encoder_decoder(enc_vocab_size=len(tk_ita.word_index)+1,
                         enc_emb_dim = 50,
                         enc_units=256,enc_input_length=20,
                         dec_vocab_size =len(tk_eng.word_index)+1,
                         dec_emb_dim =100,
                         dec_units=256,
                         dec_input_length = 20,
                         batch_size=1024)

train_steps = train_dataloader.__len__()
val_steps  = val_dataloader.__len__()

model1.compile(optimizer="adam",loss='sparse_categorical_crossentropy')
model1.fit(train_dataloader,steps_per_epoch=train_steps,epochs=50,validation_data
```

```
Epoch 1/50
269/269 [==============================] - 90s 317ms/step - loss: 1.8412 - val_
loss: 1.6170
Epoch 2/50
269/269 [==============================] - 84s 311ms/step - loss: 1.4966 - val_
loss: 1.3763
Epoch 3/50
269/269 [==============================] - 84s 313ms/step - loss: 1.2919 - val_
loss: 1.2158
Epoch 4/50
269/269 [==============================] - 84s 312ms/step - loss: 1.1505 - val_
loss: 1.0903
Epoch 5/50
269/269 [==============================] - 84s 312ms/step - loss: 1.0268 - val_
loss: 0.9782
Epoch 6/50
269/269 [==============================] - 84s 314ms/step - loss: 0.9178 - val_
loss: 0.8847
Epoch 7/50
269/269 [==============================] - 84s 312ms/step - loss: 0.8277 - val_
loss: 0.8086
Epoch 8/50
269/269 [==============================] - 85s 314ms/step - loss: 0.7519 - val_
loss: 0.7421
Epoch 9/50
269/269 [==============================] - 85s 315ms/step - loss: 0.6839 - val_
loss: 0.6859
Epoch 10/50
269/269 [==============================] - 85s 316ms/step - loss: 0.6226 - val_
loss: 0.6316
Epoch 11/50
269/269 [==============================] - 85s 316ms/step - loss: 0.5659 - val_
loss: 0.5839
Epoch 12/50
269/269 [==============================] - 85s 316ms/step - loss: 0.5148 - val_
loss: 0.5436
Epoch 13/50
269/269 [==============================] - 85s 315ms/step - loss: 0.4676 - val_
loss: 0.5023
Epoch 14/50
```

```
269/269 [==============================] - 85s 315ms/step - loss: 0.4255 - val_
loss: 0.4684
Epoch 15/50
269/269 [==============================] - 85s 316ms/step - loss: 0.3879 - val_
loss: 0.4381
Epoch 16/50
269/269 [==============================] - 85s 315ms/step - loss: 0.3538 - val_
loss: 0.4122
Epoch 17/50
269/269 [==============================] - 85s 315ms/step - loss: 0.3241 - val_
loss: 0.3885
Epoch 18/50
269/269 [==============================] - 85s 315ms/step - loss: 0.2980 - val_
loss: 0.3699
Epoch 19/50
269/269 [==============================] - 85s 315ms/step - loss: 0.2746 - val_
loss: 0.3509
Epoch 20/50
269/269 [==============================] - 85s 315ms/step - loss: 0.2542 - val_
loss: 0.3362
Epoch 21/50
269/269 [==============================] - 85s 316ms/step - loss: 0.2361 - val_
loss: 0.3239
Epoch 22/50
269/269 [==============================] - 85s 314ms/step - loss: 0.2197 - val_
loss: 0.3131
Epoch 23/50
269/269 [==============================] - 85s 316ms/step - loss: 0.2052 - val_
loss: 0.3019
Epoch 24/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1924 - val_
loss: 0.2933
Epoch 25/50
269/269 [==============================] - 85s 315ms/step - loss: 0.1809 - val_
loss: 0.2861
Epoch 26/50
269/269 [==============================] - 85s 315ms/step - loss: 0.1704 - val_
loss: 0.2792
Epoch 27/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1606 - val_
loss: 0.2732
Epoch 28/50
269/269 [==============================] - 85s 315ms/step - loss: 0.1518 - val_
loss: 0.2678
Epoch 29/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1436 - val_
loss: 0.2634
Epoch 30/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1362 - val_
loss: 0.2585
Epoch 31/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1296 - val_
loss: 0.2553
Epoch 32/50
269/269 [==============================] - 85s 317ms/step - loss: 0.1233 - val_
loss: 0.2520
Epoch 33/50
```

```
269/269 [==============================] - 85s 317ms/step - loss: 0.1181 - val_
loss: 0.2488
Epoch 34/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1119 - val_
loss: 0.2468
Epoch 35/50
269/269 [==============================] - 85s 317ms/step - loss: 0.1071 - val_
loss: 0.2443
Epoch 36/50
269/269 [==============================] - 85s 316ms/step - loss: 0.1023 - val_
loss: 0.2414
Epoch 37/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0977 - val_
loss: 0.2390
Epoch 38/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0934 - val_
loss: 0.2386
Epoch 39/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0895 - val_
loss: 0.2372
Epoch 40/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0860 - val_
loss: 0.2362
Epoch 41/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0825 - val_
loss: 0.2358
Epoch 42/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0791 - val_
loss: 0.2347
Epoch 43/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0763 - val_
loss: 0.2334
Epoch 44/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0730 - val_
loss: 0.2336
Epoch 45/50
269/269 [==============================] - 85s 317ms/step - loss: 0.0701 - val_
loss: 0.2328
Epoch 46/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0674 - val_
loss: 0.2325
Epoch 47/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0654 - val_
loss: 0.2319
Epoch 48/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0627 - val_
loss: 0.2332
Epoch 49/50
269/269 [==============================] - 85s 316ms/step - loss: 0.0605 - val_
loss: 0.2320
Epoch 50/50
269/269 [==============================] - 86s 319ms/step - loss: 0.0582 - val_
loss: 0.2328
```

Out[24]: <tensorflow.python.keras.callbacks.History at 0x7f148889a2d0>

In [25]:
```
def predict(ita_text,model):
    '''This function inputs the datapoint and return the predicted translated out
    # forming integer sequences
    seq = tk_ita.texts_to_sequences([ita_text])
    # padding the sequences
    seq = pad_sequences(seq,maxlen = 20 , padding="post")
    # initializing the states for states of lstms
    state = model.layers[0].initialize_states(1)
    # generating the output from encoder
    enc_output,state_h,state_c= model.layers[0](seq,state)

    # placeholder for predicted output
    pred = []

    input_state = [state_h,state_c]
    # initailizing the vector for inputing to decoder
    current_vec = tf.ones((1,1))

    for i in range(20): # for each word in the input
        # passing each word through decoder layer
        dec_output,dec_state_h,dec_state_c = model.layers[1](current_vec , input_
        # passing decoder output through dense  layer
        dense = model.layers[2](dec_output)
        # taking argmax and getting the word index and updating the current vecto
        current_vec = np.argmax(dense ,axis = -1)
        # updating the decoder states
        input_state = [dec_state_h,dec_state_c]
        # getting the actual word from the vocab
        pred.append(tk_eng.index_word[current_vec[0][0]])

        # if the actual word is <end> break the loop
        if tk_eng.index_word[current_vec[0][0]]=="<end>":
            break

    return " ".join(pred)
```

In [26]:
```
import nltk.translate.bleu_score as bleu
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
```

BLUE SCORE

In [27]:
```
# GETTING THE AVG BELU SCORE AFTER PREDICTING 1000 RANDOM SENTENCES
BLEU = []

test_data = validation.loc[np.random.choice(validation.index,size = 1000)][["ital
for ind,i in tqdm(test_data.iterrows(),position=0):
    pred = predict(i.italian,model1)
    act = i.english_out
    b =bleu.sentence_bleu(act,pred)
    BLEU.append(b)

np.mean(BLEU)
```

```
1000it [00:42, 23.62it/s]
```

Out[27]:
```
0.8380497994495004
```

In [28]:
```
print("BLEU SCORE",np.mean(BLEU))
```

```
BLEU SCORE 0.8380497994495004
```

PRDICTIONS

In [31]:
```
random = np.random.randint(0,2000,1)[0]
print("Predicted==>",predict( validation.italian.values[random] , model1))
print("Actual==>",validation.english_out.values[random])
```

```
Predicted==> i have got to get to book before this month <end>
Actual==> i have to finish reading that book by tomorrow <end>
```

In [32]:
```
random = np.random.randint(0,2000,1)[0]
print("Predicted==>",predict( validation.italian.values[random] , model1))
print("Actual==>",validation.english_out.values[random])
```

```
Predicted==> i am learning english to boston by road <end>
Actual==> i am learning english with the idea of going to america <end>
```

In [33]:
```
random = np.random.randint(0,2000,1)[0]
print("Predicted==>",predict( validation.italian.values[random] , model1))
print("Actual==>",validation.english_out.values[random])
```

```
Predicted==> tom is anxious to leave <end>
Actual==> tom is anxious to leave <end>
```

In [ ]:
```

```