NAME - AMITESH PATRA BATCH - AIML A1 PRN - 21070126011 github - https://github.com/amitesh30/NLP

Import Librarires

In [20]:

```
import pandas as pd
import tensorflow as tf
import keras
from keras import layers
import numpy as np
from sklearn.model_selection import train_test_split
from keras.layers import LSTM, Dense,Input, Embedding
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import seaborn as sns
import re
import string
```

In [22]:

```
data = pd.read_csv('Hindi_English_Truncated_Corpus.csv',encoding='utf-8')
data.head()
```

Out[22]:

	source	english_sentence	hindi_sentence
0	ted	politicians do not have permission to do what ...	राजनीतिज्ञों के पास जो कार्य करना चाहिए, वह कर...
1	ted	I'd like to tell you about one such child,	मई आपको ऐसे ही एक बच्चे के बारे में बताना चाहू...
2	indic2012	This percentage is even greater than the perce...	यह प्रतिशत भारत में हिन्दुओं प्रतिशत से अधिक है।
3	ted	what we really mean is that they're bad at not...	हम ये नहीं कहना चाहते कि वो ध्यान नहीं दे पाते
4	indic2012	.The ending portion of these Vedas is called U...	इन्हीं वेदों का अंतिम भाग उपनिषद कहलाता है।

In [23]:

```
# counting sources
data['source'].value_counts()
```

Out[23]:

```
source
tides        50000
ted          39881
indic2012    37726
Name: count, dtype: int64
```

In [24]:

```
# selcting data with source ted
data = data[data.source == 'tides']
data.shape
```

Out[24]:

```
(50000, 3)
```

In [5]:

```python
# checking null values
data.isna().sum()
```

Out[5]:

```
source              0
english_sentence    0
hindi_sentence      0
dtype: int64
```

In [6]:

```python
# checking duplicated data
isDuplicated = data.duplicated().any()
if isDuplicated:
    total_duplicates = data.duplicated().sum()
    print("Total duplicate rows are: ",total_duplicates)
    data.drop_duplicates(inplace = True)
```

```
Total duplicate rows are:  1078
```

Text preprocessing

In [8]:

```python
## changing uppercase to lowercase
data['english_sentence'] = data['english_sentence'].apply(lambda x: x.lower())
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: x.lower())

# Remove quotes
data['english_sentence']=data['english_sentence'].apply(lambda x: re.sub("'", '', x
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: re.sub("'", '', x))
```

In [9]:

```python
to_exclude = set(string.punctuation) # Set of all special characters
print("punctuations to exclude:: ",to_exclude)
# Remove all the special characters
data['english_sentence']=data['english_sentence'].apply(lambda x: ''.join(ch for ch
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: ''.join(ch for ch in
```

```
punctuations to exclude::  {'>', '&', '~', '`', '!', "'", ';', '^', '/', ')', '#',
'+', '-', '\\', '|', '{', '_', ',', '.', ']', '?', '%', '}', '@', '<', '$', ':',
'=', '"', '(', '*', '['}
```

In [10]:

```python
from string import digits
# Remove all numbers from text
remove_digits = str.maketrans('', '', digits)

data['english_sentence']=data['english_sentence'].apply(lambda x: x.translate(remov
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: x.translate(remove_di

data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: re.sub("[२३०८१५७९४६]

# Remove extra spaces
data['english_sentence']=data['english_sentence'].apply(lambda x: x.strip())
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: x.strip())
data['english_sentence']=data['english_sentence'].apply(lambda x: re.sub(" +", " ",
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: re.sub(" +", " ", x))
```

In [11]:

```
## adding start and end token to the target sentence
data['hindi_sentence'] = data['hindi_sentence'].apply(lambda x: "START_ " + x + " _
```

In [12]:

```
## counting length of english and hindi sentence
data['english_length'] = data['english_sentence'].apply(lambda x: len(x.split(' '))
data['hindi_length'] = data['hindi_sentence'].apply(lambda x: len(x.split(' ')))

data.head()
```

Out[12]:

	source	english_sentence	hindi_sentence	english_length	hindi_length
123152	ted	is not about belief but about behavior	START_ वो विश्वास के बारे में नहीं वरन लेकिन व...	7	14
31468	ted	than the story were going to tell about it later	START_ उससे जो हम बाद में बताने वाले हैं _END	10	10
102720	ted	and if they need a pair of glasses they are av...	START_ यदि किसी को चश्मे की ज़रूरत है तो उसे श...	17	19
40273	ted	the feedback here is immediate	START_ यहाँ तुरंत नतीजा मिलता है _END	5	7
17027	ted	a rather astonishing demonstration of the abil...	START_ यह दिमाग की एक अद्भुत क्षमता का प्रदर्श...	10	11

In [13]:

```
print("Maximum length of English Sentence: ", max(data['english_length']))
print("Maximum length of Hindi Sentence: ",max(data['hindi_length']))
```

```
Maximum length of English Sentence:  21
Maximum length of Hindi Sentence:  32
```

In [14]:

```
### Get English and Hindi Vocabulary
all_eng_words=set()
for eng in data['english_sentence']:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_hindi_words=set()
for hin in data['hindi_sentence']:
    for word in hin.split():
        if word not in all_hindi_words:
            all_hindi_words.add(word)


print("toral english words: ",len(all_eng_words))
print('total hind words: ',len(all_hindi_words))
```

```
toral english words:  12483
total hind words:  15529
```

In [15]:

```
## using only sentence with length less than 20
mask1 = data['english_length'] < 21
mask2 = data['hindi_length'] < 21
data = data[mask1 & mask2]
data.shape
```

```
Out[15]:
(19830, 5)
```

In [16]:

```
print("maximum length of Hindi Sentence ",max(data['hindi_length']))
print("maximum length of English Sentence ",max(data['english_length']))
```

```
maximum length of Hindi Sentence  20
maximum length of English Sentence  20
```

In [17]:

```
input_words = sorted(list(all_eng_words))
target_words = sorted(list(all_hindi_words))
num_encoder_tokens = len(all_eng_words)
num_decoder_tokens = len(all_hindi_words)

num_encoder_tokens, num_decoder_tokens
```

```
Out[17]:
(12483, 15529)
```

In [18]:

```
num_decoder_tokens += 1 #for zero padding
```

In [19]:

```
input_token_index = dict([(word, i+1) for i, word in enumerate(input_words)])
target_token_index = dict([(word, i+1) for i, word in enumerate(target_words)])
print("Token for accelerating is: ",input_token_index['accelerating'])
```

```
Token for accelerating is:  50
```

In [20]:

```
reverse_input_char_index = dict((i, word) for word, i in input_token_index.items())
reverse_target_char_index = dict((i, word) for word, i in target_token_index.items(
print("Character for toker 50 is: ",reverse_input_char_index[50])
```

```
Character for toker 50 is:  accelerating
```

In [21]:

```
# splitting data
X_, y_ = data['english_sentence'], data['hindi_sentence']
X_train, X_test, y_train, y_test = train_test_split(X_, y_, test_size = 0.2,random_
print("Total number of training data: ",X_train.shape[0])
print("Toral number of testing data: ",X_test.shape[0])
```

```
Total number of training data:  15864
Toral number of testing data:  3966
```

In [22]:

```
latent_dim = 300

# Encoder
encoder_inputs = Input(shape=(None,))
enc_emb =  Embedding(num_encoder_tokens, latent_dim, mask_zero=True)(encoder_inputs

encoder_lstm1 = LSTM(latent_dim, return_sequences=True, return_state=True)
```

```python
encoder_outputs1, state_h1, state_c1 = encoder_lstm1(enc_emb)

encoder_lstm2 = LSTM(latent_dim, return_state=True)
encoder_outputs2, state_h2, state_c2 = encoder_lstm2(encoder_outputs1)

# Concatenate the states from both LSTM layers
encoder_states = [state_h2, state_c2]

# Set up the decoder
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm1 = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs1, _, _ = decoder_lstm1(dec_emb, initial_state=encoder_states)

decoder_lstm2 = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs2, _, _ = decoder_lstm2(decoder_outputs1, initial_state=encoder_stat

decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs2)

# Define the model
model = tf.keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.summary()
```

```
Model: "model"
__________________________________________________________________________________
_______________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================
===============
 input_1 (InputLayer)           [(None, None)]       0           []

 embedding (Embedding)          (None, None, 300)    3744900     ['input_1[0][0]']

 input_2 (InputLayer)           [(None, None)]       0           []

 lstm (LSTM)                    [(None, None, 300),  721200      ['embedding[0]
[0]']

                                 (None, 300),
                                 (None, 300)]

 embedding_1 (Embedding)        (None, None, 300)    4659000     ['input_2[0][0]']

 lstm_1 (LSTM)                  [(None, 300),        721200      ['lstm[0][0]']
                                 (None, 300),
                                 (None, 300)]

 lstm_2 (LSTM)                  [(None, None, 300),  721200      ['embedding_1[0]
[0]',

                                 (None, 300),                     'lstm_1[0][1]',
                                 (None, 300)]                     'lstm_1[0][2]']

 lstm_3 (LSTM)                  [(None, None, 300),  721200      ['lstm_2[0][0]',
                                 (None, 300),                     'lstm_1[0][1]',
                                 (None, 300)]                     'lstm_1[0][2]']

 dense (Dense)                  (None, None, 15530)  4674530     ['lstm_3[0][0]']
```

```
================================================================================
===============
Total params: 15,963,230
Trainable params: 15,963,230
Non-trainable params: 0
________________________________________________________________________________
_______________
```

In [23]:

```python
from tensorflow.keras.optimizers import Adam
```

In [24]:

```python
model.compile(optimizer=Adam(learning_rate=0.001, epsilon=1e-7), loss='categorical_
```

In [25]:

```python
max_length_src = 20
max_length_tar = 20

def generate_batch(X = X_train, y = y_train, batch_size = 128):
    ''' Generate a batch of data '''
    while True:
        for j in range(0, len(X), batch_size):
            encoder_input_data = np.zeros((batch_size, max_length_src),dtype='float
            decoder_input_data = np.zeros((batch_size, max_length_tar),dtype='float
            decoder_target_data = np.zeros((batch_size, max_length_tar, num_decoder
            for i, (input_text, target_text) in enumerate(zip(X[j:j+batch_size], y[
                for t, word in enumerate(input_text.split()):
                    encoder_input_data[i, t] = input_token_index[word] # encoder in
                for t, word in enumerate(target_text.split()):
                    if t<len(target_text.split())-1:
                        decoder_input_data[i, t] = target_token_index[word] # decod
                    if t>0:
                        # decoder target sequence (one hot encoded)
                        # does not include the START_ token
                        # Offset by one timestep
                        decoder_target_data[i, t - 1, target_token_index[word]] = 1
            yield([encoder_input_data, decoder_input_data], decoder_target_data)
```

In [26]:

```python
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128
epochs = 100
```

In [27]:

```python
model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = batch
                    steps_per_epoch = train_samples//batch_size,
                    epochs=epochs,
                    validation_data = generate_batch(X_test, y_test, batch_size = b
                    validation_steps = val_samples//batch_size)
```

```
/tmp/ipykernel_29/2108290721.py:1: UserWarning: `Model.fit_generator` is deprecated
and will be removed in a future version. Please use `Model.fit`, which supports gen
erators.
  model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = bat
ch_size),
Epoch 1/100
123/123 [==============================] - 92s 574ms/step - loss: 6.8924 - accurac
y: 0.1200 - val_loss: 6.5331 - val_accuracy: 0.1275
Epoch 2/100
```

```
123/123 [==============================] - 44s 357ms/step - loss: 6.3666 - accurac
y: 0.1313 - val_loss: 6.5401 - val_accuracy: 0.1290
Epoch 3/100
123/123 [==============================] - 46s 374ms/step - loss: 6.3009 - accurac
y: 0.1327 - val_loss: 6.4936 - val_accuracy: 0.1309
Epoch 4/100
123/123 [==============================] - 44s 355ms/step - loss: 6.1550 - accurac
y: 0.1368 - val_loss: 6.2743 - val_accuracy: 0.1427
Epoch 5/100
123/123 [==============================] - 43s 354ms/step - loss: 5.8717 - accurac
y: 0.1496 - val_loss: 6.1069 - val_accuracy: 0.1517
Epoch 6/100
123/123 [==============================] - 46s 372ms/step - loss: 5.6580 - accurac
y: 0.1677 - val_loss: 6.0047 - val_accuracy: 0.1694
Epoch 7/100
123/123 [==============================] - 45s 369ms/step - loss: 5.4802 - accurac
y: 0.1863 - val_loss: 5.9380 - val_accuracy: 0.1785
Epoch 8/100
123/123 [==============================] - 44s 355ms/step - loss: 5.3337 - accurac
y: 0.1986 - val_loss: 5.8928 - val_accuracy: 0.1843
Epoch 9/100
123/123 [==============================] - 44s 354ms/step - loss: 5.1881 - accurac
y: 0.2079 - val_loss: 5.8481 - val_accuracy: 0.1873
Epoch 10/100
123/123 [==============================] - 46s 373ms/step - loss: 5.0424 - accurac
y: 0.2192 - val_loss: 5.7961 - val_accuracy: 0.1963
Epoch 11/100
123/123 [==============================] - 45s 371ms/step - loss: 4.9117 - accurac
y: 0.2309 - val_loss: 5.7986 - val_accuracy: 0.2016
Epoch 12/100
123/123 [==============================] - 46s 372ms/step - loss: 4.7979 - accurac
y: 0.2387 - val_loss: 5.7890 - val_accuracy: 0.2066
Epoch 13/100
123/123 [==============================] - 44s 356ms/step - loss: 4.6916 - accurac
y: 0.2458 - val_loss: 5.7809 - val_accuracy: 0.2085
Epoch 14/100
123/123 [==============================] - 45s 369ms/step - loss: 4.5883 - accurac
y: 0.2535 - val_loss: 5.8057 - val_accuracy: 0.2112
Epoch 15/100
123/123 [==============================] - 45s 369ms/step - loss: 4.4878 - accurac
y: 0.2602 - val_loss: 5.8832 - val_accuracy: 0.2128
Epoch 16/100
123/123 [==============================] - 44s 356ms/step - loss: 4.3989 - accurac
y: 0.2662 - val_loss: 5.8277 - val_accuracy: 0.2144
Epoch 17/100
123/123 [==============================] - 43s 353ms/step - loss: 4.3060 - accurac
y: 0.2734 - val_loss: 5.8441 - val_accuracy: 0.2168
Epoch 18/100
123/123 [==============================] - 45s 370ms/step - loss: 4.2078 - accurac
y: 0.2798 - val_loss: 5.8462 - val_accuracy: 0.2158
Epoch 19/100
123/123 [==============================] - 45s 363ms/step - loss: 4.1081 - accurac
y: 0.2869 - val_loss: 5.8557 - val_accuracy: 0.2163
Epoch 20/100
123/123 [==============================] - 45s 370ms/step - loss: 4.0147 - accurac
y: 0.2940 - val_loss: 5.8614 - val_accuracy: 0.2197
Epoch 21/100
123/123 [==============================] - 46s 372ms/step - loss: 3.9259 - accurac
y: 0.3010 - val_loss: 5.8877 - val_accuracy: 0.2168
Epoch 22/100
```

```
123/123 [==============================] - 43s 352ms/step - loss: 3.8511 - accurac
y: 0.3078 - val_loss: 5.9200 - val_accuracy: 0.2171
Epoch 23/100
123/123 [==============================] - 44s 356ms/step - loss: 3.7658 - accurac
y: 0.3163 - val_loss: 5.9297 - val_accuracy: 0.2208
Epoch 24/100
123/123 [==============================] - 46s 373ms/step - loss: 3.6771 - accurac
y: 0.3264 - val_loss: 5.9471 - val_accuracy: 0.2210
Epoch 25/100
123/123 [==============================] - 44s 355ms/step - loss: 3.5855 - accurac
y: 0.3390 - val_loss: 6.0035 - val_accuracy: 0.2202
Epoch 26/100
123/123 [==============================] - 44s 358ms/step - loss: 3.5015 - accurac
y: 0.3503 - val_loss: 5.9884 - val_accuracy: 0.2203
Epoch 27/100
123/123 [==============================] - 46s 373ms/step - loss: 3.4228 - accurac
y: 0.3605 - val_loss: 6.0223 - val_accuracy: 0.2215
Epoch 28/100
123/123 [==============================] - 45s 366ms/step - loss: 3.3431 - accurac
y: 0.3717 - val_loss: 6.0555 - val_accuracy: 0.2234
Epoch 29/100
123/123 [==============================] - 45s 368ms/step - loss: 3.2588 - accurac
y: 0.3823 - val_loss: 6.0770 - val_accuracy: 0.2210
Epoch 30/100
123/123 [==============================] - 47s 379ms/step - loss: 3.1874 - accurac
y: 0.3933 - val_loss: 6.1227 - val_accuracy: 0.2215
Epoch 31/100
123/123 [==============================] - 46s 375ms/step - loss: 3.1309 - accurac
y: 0.4024 - val_loss: 6.1377 - val_accuracy: 0.2237
Epoch 32/100
123/123 [==============================] - 44s 360ms/step - loss: 3.0428 - accurac
y: 0.4140 - val_loss: 6.1514 - val_accuracy: 0.2227
Epoch 33/100
123/123 [==============================] - 47s 378ms/step - loss: 2.9591 - accurac
y: 0.4269 - val_loss: 6.1770 - val_accuracy: 0.2198
Epoch 34/100
123/123 [==============================] - 46s 378ms/step - loss: 2.8757 - accurac
y: 0.4407 - val_loss: 6.2190 - val_accuracy: 0.2190
Epoch 35/100
123/123 [==============================] - 46s 377ms/step - loss: 2.7989 - accurac
y: 0.4525 - val_loss: 6.2775 - val_accuracy: 0.2181
Epoch 36/100
123/123 [==============================] - 44s 356ms/step - loss: 2.7314 - accurac
y: 0.4618 - val_loss: 6.3076 - val_accuracy: 0.2173
Epoch 37/100
123/123 [==============================] - 46s 377ms/step - loss: 2.6635 - accurac
y: 0.4740 - val_loss: 6.3462 - val_accuracy: 0.2169
Epoch 38/100
123/123 [==============================] - 46s 373ms/step - loss: 2.6024 - accurac
y: 0.4833 - val_loss: 6.3913 - val_accuracy: 0.2232
Epoch 39/100
123/123 [==============================] - 47s 384ms/step - loss: 2.5419 - accurac
y: 0.4931 - val_loss: 6.4301 - val_accuracy: 0.2199
Epoch 40/100
123/123 [==============================] - 45s 370ms/step - loss: 2.4741 - accurac
y: 0.5050 - val_loss: 6.4473 - val_accuracy: 0.2143
Epoch 41/100
123/123 [==============================] - 47s 379ms/step - loss: 2.4112 - accurac
y: 0.5164 - val_loss: 6.5238 - val_accuracy: 0.2183
Epoch 42/100
```

```
123/123 [==============================] - 47s 379ms/step - loss: 2.3490 - accurac
y: 0.5279 - val_loss: 6.6147 - val_accuracy: 0.2215
Epoch 43/100
123/123 [==============================] - 46s 378ms/step - loss: 2.2916 - accurac
y: 0.5369 - val_loss: 6.6324 - val_accuracy: 0.2164
Epoch 44/100
123/123 [==============================] - 46s 378ms/step - loss: 2.2312 - accurac
y: 0.5477 - val_loss: 6.6189 - val_accuracy: 0.2107
Epoch 45/100
123/123 [==============================] - 46s 379ms/step - loss: 2.1750 - accurac
y: 0.5577 - val_loss: 6.6465 - val_accuracy: 0.2131
Epoch 46/100
123/123 [==============================] - 46s 376ms/step - loss: 2.1173 - accurac
y: 0.5684 - val_loss: 6.6696 - val_accuracy: 0.2128
Epoch 47/100
123/123 [==============================] - 44s 361ms/step - loss: 2.0577 - accurac
y: 0.5799 - val_loss: 6.7097 - val_accuracy: 0.2159
Epoch 48/100
123/123 [==============================] - 44s 359ms/step - loss: 1.9956 - accurac
y: 0.5909 - val_loss: 6.7772 - val_accuracy: 0.2160
Epoch 49/100
123/123 [==============================] - 45s 364ms/step - loss: 1.9392 - accurac
y: 0.6017 - val_loss: 6.7784 - val_accuracy: 0.2135
Epoch 50/100
123/123 [==============================] - 44s 357ms/step - loss: 1.8830 - accurac
y: 0.6127 - val_loss: 6.8402 - val_accuracy: 0.2097
Epoch 51/100
123/123 [==============================] - 44s 356ms/step - loss: 1.8280 - accurac
y: 0.6233 - val_loss: 6.9179 - val_accuracy: 0.2105
Epoch 52/100
123/123 [==============================] - 44s 359ms/step - loss: 1.7784 - accurac
y: 0.6331 - val_loss: 6.9240 - val_accuracy: 0.2049
Epoch 53/100
123/123 [==============================] - 46s 375ms/step - loss: 1.7263 - accurac
y: 0.6427 - val_loss: 6.9797 - val_accuracy: 0.2044
Epoch 54/100
123/123 [==============================] - 46s 371ms/step - loss: 1.6739 - accurac
y: 0.6534 - val_loss: 7.0709 - val_accuracy: 0.2079
Epoch 55/100
123/123 [==============================] - 46s 372ms/step - loss: 1.6218 - accurac
y: 0.6637 - val_loss: 7.1302 - val_accuracy: 0.2035
Epoch 56/100
123/123 [==============================] - 44s 356ms/step - loss: 1.5655 - accurac
y: 0.6759 - val_loss: 7.2110 - val_accuracy: 0.2048
Epoch 57/100
123/123 [==============================] - 46s 371ms/step - loss: 1.5147 - accurac
y: 0.6866 - val_loss: 7.2086 - val_accuracy: 0.2075
Epoch 58/100
123/123 [==============================] - 46s 371ms/step - loss: 1.4640 - accurac
y: 0.6966 - val_loss: 7.2181 - val_accuracy: 0.2025
Epoch 59/100
123/123 [==============================] - 46s 371ms/step - loss: 1.4154 - accurac
y: 0.7070 - val_loss: 7.2832 - val_accuracy: 0.2078
Epoch 60/100
123/123 [==============================] - 45s 370ms/step - loss: 1.2678 - accurac
y: 0.7401 - val_loss: 7.4047 - val_accuracy: 0.2006
Epoch 63/100
123/123 [==============================] - 47s 379ms/step - loss: 1.2250 - accurac
y: 0.7490 - val_loss: 7.5097 - val_accuracy: 0.2052
Epoch 64/100
```

```
123/123 [==============================] - 46s 379ms/step - loss: 1.1811 - accurac
y: 0.7590 - val_loss: 7.5975 - val_accuracy: 0.2076
Epoch 65/100
123/123 [==============================] - 46s 379ms/step - loss: 1.1402 - accurac
y: 0.7678 - val_loss: 7.5935 - val_accuracy: 0.2043
Epoch 66/100
123/123 [==============================] - 44s 360ms/step - loss: 1.0996 - accurac
y: 0.7761 - val_loss: 7.6354 - val_accuracy: 0.1961
Epoch 67/100
123/123 [==============================] - 44s 360ms/step - loss: 1.0571 - accurac
y: 0.7853 - val_loss: 7.7630 - val_accuracy: 0.2055
Epoch 68/100
123/123 [==============================] - 44s 356ms/step - loss: 1.0147 - accurac
y: 0.7957 - val_loss: 7.7433 - val_accuracy: 0.1982
Epoch 69/100
123/123 [==============================] - 46s 374ms/step - loss: 0.9692 - accurac
y: 0.8053 - val_loss: 7.7830 - val_accuracy: 0.1959
Epoch 70/100
123/123 [==============================] - 44s 356ms/step - loss: 0.9271 - accurac
y: 0.8157 - val_loss: 7.9347 - val_accuracy: 0.2040
Epoch 71/100
123/123 [==============================] - 46s 374ms/step - loss: 0.8833 - accurac
y: 0.8262 - val_loss: 7.9950 - val_accuracy: 0.2020
Epoch 72/100
123/123 [==============================] - 46s 377ms/step - loss: 0.8459 - accurac
y: 0.8347 - val_loss: 7.9892 - val_accuracy: 0.1964
Epoch 73/100
123/123 [==============================] - 46s 374ms/step - loss: 0.8079 - accurac
y: 0.8430 - val_loss: 8.0303 - val_accuracy: 0.1970
Epoch 74/100
123/123 [==============================] - 46s 377ms/step - loss: 0.7715 - accurac
y: 0.8518 - val_loss: 8.0814 - val_accuracy: 0.1972
Epoch 75/100
123/123 [==============================] - 46s 375ms/step - loss: 0.7359 - accurac
y: 0.8613 - val_loss: 8.1370 - val_accuracy: 0.1958
Epoch 76/100
123/123 [==============================] - 46s 373ms/step - loss: 0.7017 - accurac
y: 0.8696 - val_loss: 8.3113 - val_accuracy: 0.2031
Epoch 77/100
123/123 [==============================] - 46s 372ms/step - loss: 0.6767 - accurac
y: 0.8751 - val_loss: 8.4213 - val_accuracy: 0.2034
Epoch 78/100
123/123 [==============================] - 46s 372ms/step - loss: 0.6415 - accurac
y: 0.8821 - val_loss: 8.3667 - val_accuracy: 0.1964
Epoch 79/100
123/123 [==============================] - 46s 373ms/step - loss: 0.6117 - accurac
y: 0.8897 - val_loss: 8.3487 - val_accuracy: 0.1914
Epoch 80/100
123/123 [==============================] - 43s 354ms/step - loss: 0.5813 - accurac
y: 0.8954 - val_loss: 8.3680 - val_accuracy: 0.1862
Epoch 81/100
123/123 [==============================] - 44s 357ms/step - loss: 0.5573 - accurac
y: 0.9007 - val_loss: 8.4394 - val_accuracy: 0.1902
Epoch 82/100
123/123 [==============================] - 43s 352ms/step - loss: 0.5340 - accurac
y: 0.9055 - val_loss: 8.5362 - val_accuracy: 0.1950
Epoch 83/100
123/123 [==============================] - 46s 373ms/step - loss: 0.5051 - accurac
y: 0.9122 - val_loss: 8.5421 - val_accuracy: 0.1875
Epoch 84/100
```

```
123/123 [==============================] - 43s 352ms/step - loss: 0.4741 - accurac
y: 0.9198 - val_loss: 8.6279 - val_accuracy: 0.1854
Epoch 85/100
123/123 [==============================] - 46s 372ms/step - loss: 0.4453 - accurac
y: 0.9276 - val_loss: 8.7753 - val_accuracy: 0.1942
Epoch 86/100
123/123 [==============================] - 46s 372ms/step - loss: 0.4163 - accurac
y: 0.9345 - val_loss: 8.8584 - val_accuracy: 0.1965
Epoch 87/100
123/123 [==============================] - 46s 373ms/step - loss: 0.3884 - accurac
y: 0.9413 - val_loss: 8.7799 - val_accuracy: 0.1903
Epoch 88/100
123/123 [==============================] - 46s 373ms/step - loss: 0.3672 - accurac
y: 0.9455 - val_loss: 8.8244 - val_accuracy: 0.1865
Epoch 89/100
123/123 [==============================] - 44s 355ms/step - loss: 0.3476 - accurac
y: 0.9493 - val_loss: 8.8847 - val_accuracy: 0.1856
Epoch 90/100
123/123 [==============================] - 45s 371ms/step - loss: 0.3308 - accurac
y: 0.9530 - val_loss: 8.9784 - val_accuracy: 0.1873
Epoch 91/100
123/123 [==============================] - 46s 371ms/step - loss: 0.3167 - accurac
y: 0.9554 - val_loss: 9.0788 - val_accuracy: 0.1939
Epoch 92/100
123/123 [==============================] - 46s 373ms/step - loss: 0.3018 - accurac
y: 0.9584 - val_loss: 9.1110 - val_accuracy: 0.1959
Epoch 93/100
123/123 [==============================] - 44s 357ms/step - loss: 0.2830 - accurac
y: 0.9623 - val_loss: 9.1672 - val_accuracy: 0.1935
Epoch 94/100
123/123 [==============================] - 43s 354ms/step - loss: 0.2583 - accurac
y: 0.9680 - val_loss: 9.2720 - val_accuracy: 0.1955
Epoch 95/100
123/123 [==============================] - 43s 353ms/step - loss: 0.2451 - accurac
y: 0.9712 - val_loss: 9.2456 - val_accuracy: 0.1937
Epoch 96/100
123/123 [==============================] - 44s 358ms/step - loss: 0.2103 - accurac
y: 0.9788 - val_loss: 9.2723 - val_accuracy: 0.1890
Epoch 97/100
123/123 [==============================] - 46s 372ms/step - loss: 0.1956 - accurac
y: 0.9813 - val_loss: 9.3594 - val_accuracy: 0.1900
Epoch 98/100
123/123 [==============================] - 46s 371ms/step - loss: 0.1748 - accurac
y: 0.9856 - val_loss: 9.4831 - val_accuracy: 0.1915
Epoch 99/100
123/123 [==============================] - 44s 355ms/step - loss: 0.1592 - accurac
y: 0.9879 - val_loss: 9.5400 - val_accuracy: 0.1927
Epoch 100/100
123/123 [==============================] - 43s 353ms/step - loss: 0.1466 - accurac
y: 0.9899 - val_loss: 9.5896 - val_accuracy: 0.1926
```

Out[27]:

```
<keras.callbacks.History at 0x7c50aff47d00>
```

In [37]:

```python
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128
epochs = 10
```

In [38]:

```
model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = batch
                    steps_per_epoch = train_samples//batch_size,
                    epochs=epochs,
                    validation_data = generate_batch(X_test, y_test, batch_size = b
                    validation_steps = val_samples//batch_size)
```

```
/tmp/ipykernel_29/2108290721.py:1: UserWarning: `Model.fit_generator` is deprecated
and will be removed in a future version. Please use `Model.fit`, which supports gen
erators.
  model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = bat
ch_size),
Epoch 1/10
123/123 [==============================] - 45s 368ms/step - loss: 0.1093 - accurac
y: 0.9948 - val_loss: 9.6977 - val_accuracy: 0.1914
Epoch 2/10
123/123 [==============================] - 43s 351ms/step - loss: 0.1016 - accurac
y: 0.9957 - val_loss: 9.7523 - val_accuracy: 0.1915
Epoch 3/10
123/123 [==============================] - 43s 353ms/step - loss: 0.0930 - accurac
y: 0.9966 - val_loss: 9.7925 - val_accuracy: 0.1910
Epoch 4/10
123/123 [==============================] - 43s 348ms/step - loss: 0.0870 - accurac
y: 0.9969 - val_loss: 9.8405 - val_accuracy: 0.1895
Epoch 5/10
123/123 [==============================] - 45s 369ms/step - loss: 0.0811 - accurac
y: 0.9974 - val_loss: 9.9214 - val_accuracy: 0.1904
Epoch 6/10
123/123 [==============================] - 43s 350ms/step - loss: 0.0769 - accurac
y: 0.9975 - val_loss: 9.9866 - val_accuracy: 0.1926
Epoch 7/10
123/123 [==============================] - 43s 349ms/step - loss: 0.0734 - accurac
y: 0.9977 - val_loss: 9.9998 - val_accuracy: 0.1917
Epoch 8/10
123/123 [==============================] - 45s 368ms/step - loss: 0.0688 - accurac
y: 0.9980 - val_loss: 10.0777 - val_accuracy: 0.1912
Epoch 9/10
123/123 [==============================] - 46s 371ms/step - loss: 0.0658 - accurac
y: 0.9979 - val_loss: 10.1585 - val_accuracy: 0.1915
Epoch 10/10
123/123 [==============================] - 43s 351ms/step - loss: 0.0619 - accurac
y: 0.9982 - val_loss: 10.1820 - val_accuracy: 0.1912
```

Out[38]:

```
<keras.callbacks.History at 0x7c4c147ce920>
```

In [39]:

```
# Encode the input sequence to get the "thought vectors"
encoder_model = tf.keras.Model(encoder_inputs, encoder_states)

# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_states_inputs = [decoder_state_input_h, decoder_state_input_c]

dec_emb2= dec_emb_layer(decoder_inputs) # Get the embeddings of the decoder sequenc

# To predict the next word in the sequence, set the initial states to the states fr
decoder_outputs2, state_h2, state_c2 = decoder_lstm2(dec_emb2, initial_state=decode
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) # A dense softmax layer to gener
```

```
# Final decoder model
decoder_model = tf.keras.Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)

def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)
    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start character.
    target_seq[0, 0] = target_token_index['START_']

    # Sampling loop for a batch of sequences
    # (to simplify, here we assume a batch of size 1).
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += ' '+sampled_char

        # Exit condition: either hit max length
        # or find stop character.
        if (sampled_char == '_END' or
           len(decoded_sentence) > 50):
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update states
        states_value = [h, c]

    return decoded_sentence
```

In [40]:

```
train_gen = generate_batch(X_train, y_train, batch_size = 1)
k = -1
```

In [41]:

```
k+=1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
Input English sentence: which occurs in nature
Actual Hindi Translation:  जो प्रकृति में होता है
Predicted Hindi Translation:  जो जो जिसका हरी का किसी बिजली
```

In [42]:

```
k = k + 1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
Input English sentence: are actually going to matter in the long run
Actual Hindi Translation:  लंबे समय में हमारे लिए मायने रखते है
Predicted Hindi Translation:  उसमें वो बच्चों बच्चों जहाँ में जहाँ में में में
```

In [43]:

```
k = k + 1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
Input English sentence: i mean really though
Actual Hindi Translation:  मेरा मतलब सच में कठिन
Predicted Hindi Translation:  मेरा कहेता में गरीबी अलग या की
```

In [44]:

```
k = k + 1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
```

```
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
Input English sentence: and youre aware of how many people are around you
Actual Hindi Translation:  और आपको अंदाज़ा है कि कितने लोग आपके आसपास है
Predicted Hindi Translation:  और आप आपको आप आपको लेकिन आपको आपको लेकिन आपको हमे
```

In [45]:

```python
k = k + 1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
Input English sentence: and so i get angry and i get pissed
Actual Hindi Translation:  क्यों के ये बात पे मुझे बहोत ही अधिक ग़ुस्सा आता है
Predicted Hindi Translation:  आज इस के मेरी हूँ पिछले मेरी मेरी मेरी मेरी मेरी
```

In [46]:

```python
k = k + 1
(input_seq, actual_output), _ = next(train_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_train[k:k+1].values[0])
print('Actual Hindi Translation:', y_train[k:k+1].values[0][6:-4])
print('Predicted Hindi Translation:', decoded_sentence[:-4])
```

```
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
Input English sentence: who is very interested in success i really want to be succe
ssful
```

```
Actual Hindi Translation:  जो सफलता में बहुत रूचि लेते है मैं वास्तव में सफल बनना चाहता हूँ
Predicted Hindi Translation:  जो मेरे मेरे मैं मैं मैंने मैंने मैंने मैंने म
```

In [ ]:

```

```

In [ ]:

```

```