In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries i
# It is defined by the kaggle/python Docker image: https://github.com/kag
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) wil

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) th
# You can also write temporary files to /kaggle/temp/, but they won't be
```

```
/kaggle/input/machine-translate/Hindi_English_Truncated_Corpus.csv
```

Content

1. Introduction

1.1. Author's Details

Name : Nisarg Patel

PRN : 21070126060

Batch : AIML - A3

Git-Repo : GitRepo

1.2. Importing the Libraries

In [1]:
```
import tensorflow as tf
from tensorflow.keras.layers import Embedding , LSTM , Dense, RepeatVecto
from tensorflow.keras.models import Model
from tensorflow.keras.losses import sparse_categorical_crossentropy
import re
import string
from string import digits
import numpy as np
```

1.3. Importing the Dataset

In [3]:
```
df = pd.read_csv('/kaggle/input/machine-translate/Hindi_English_Truncated
df['source'].value_counts()
```

Out[3]:
```
source
tides        50000
ted          39881
indic2012    37726
Name: count, dtype: int64
```

In [4]:
```
df = df[(df.english_sentence.apply(lambda x: len(str(x))<=30)) &
            (df.hindi_sentence.apply(lambda x: len(str(x))<=30))]
```

2. Data Preprocessing

2.1. Changing the Uppercase to Lowercase

In [5]:
```
# Changing uppercase to lowercase
df['english_sentence'] = df['english_sentence'].apply(lambda x:str(x).low
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: x.lower())

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

2.2. Removing the Punctuations

In [6]:
```
to_exclude = set(string.punctuation) # Set of all special characters
print("Punctuations to be excluded : ",to_exclude)

# Remove all the special characters
df['english_sentence'] = df['english_sentence'].apply(lambda x: ''.join(c
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: ''.join(ch fo
```

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

2.3. Removing the Digits

In [7]:
```
# Remove all the digits
from string import digits
remove_digits = str.maketrans('', '', digits)
df['english_sentence'] = df['english_sentence'].apply(lambda x: x.transla
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: x.translate(r
df
```

Out[7]:

	source	english_sentence	hindi_sentence
11	indic2012	category religious text	श्रेणीधर्मग्रन्थ
23	ted	this changed slowly	धीरे धीरे ये सब बदला
26	ted	were being produced	उत्पन्न नहीं कि जाती थी
33	indic2012	maine	मेन
35	ted	can you imagine saying that	क्या आप ये कल्पना कर सकते है
...	...	...	...
127580	indic2012	google author	गूगल अर्थ
127581	tides	the die was cast	सांचा ढल गया
127582	indic2012	devotional period	भक्ति काल १३७५१७००
127590	indic2012	travelling	यात्राएं
127598	ted	thank you	धन्यवाद

18416 rows × 3 columns

In [8]:
```
# Remove all the digits
from string import digits
remove_digits = str.maketrans('', '', digits)
df['english_sentence'] = df['english_sentence'].apply(lambda x: x.transla
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: x.translate(r
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: re.sub("[२३०८
df
```

Out[8]:

	source	english_sentence	hindi_sentence
11	indic2012	category religious text	श्रेणीधर्मग्रन्थ
23	ted	this changed slowly	धीरे धीरे ये सब बदला
26	ted	were being produced	उत्पन्न नहीं कि जाती थी
33	indic2012	maine	मेन
35	ted	can you imagine saying that	क्या आप ये कल्पना कर सकते है
...	...	...	...
127580	indic2012	google author	गूगल अर्थ
127581	tides	the die was cast	सांचा ढल गया
127582	indic2012	devotional period	भक्ति काल
127590	indic2012	travelling	यात्राएं
127598	ted	thank you	धन्यवाद

18416 rows × 3 columns

2.4. Removing the Extra Spaces

In [9]:
```
# Remove extra spaces
df['english_sentence'] = df['english_sentence'].apply(lambda x: x.strip()
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: x.strip())
df['english_sentence'] = df['english_sentence'].apply(lambda x: re.sub("
df['hindi_sentence'] = df['hindi_sentence'].apply(lambda x: re.sub(" +",
df
```

Out[9]:

	source	english_sentence	hindi_sentence
11	indic2012	category religious text	श्रेणीधर्मग्रन्थ
23	ted	this changed slowly	धीरे धीरे ये सब बदला
26	ted	were being produced	उत्पन्न नहीं कि जाती थी
33	indic2012	maine	मेन
35	ted	can you imagine saying that	क्या आप ये कल्पना कर सकते है
...	...	...	...
127580	indic2012	google author	गूगल अर्थ
127581	tides	the die was cast	सांचा ढल गया
127582	indic2012	devotional period	भक्ति काल
127590	indic2012	travelling	यात्राएं
127598	ted	thank you	धन्यवाद

18416 rows × 3 columns

2.5. Adding the Start and End Sequence

In [10]:
```
# Create 2 list to send the data
input_text = []
target_text = []

# Creating the characters
input_characters = set()
target_characters = set()

for eng, hin in df[['english_sentence','hindi_sentence']].itertuples(inde
    target = 'START_ ' + hin + ' _END' # End Sequence
    input_text.append(eng)
    target_text.append(target)

    for char in eng.split():
        if char not in input_characters:
            input_characters.add(char)

    for hin_char in hin.split():
        if hin_char not in target_characters:
            target_characters.add(hin_char)
```

2.6. Printing the Data

In [11]:
```
# Making the input_char
input_char = sorted(list(input_characters))
target_char = sorted(list(target_characters))
```

```
# Encoder defininig
num_encoder_tokens = len(input_char)
num_decoder_tokens = len(target_char) + 1

max_encoder_seq_length = max([len(txt) for txt in input_text])
max_decoder_seq_length = max([len(txt) for txt in target_text])
print(num_encoder_tokens)
print(num_decoder_tokens)
print(max_encoder_seq_length)
print(max_decoder_seq_length)
```

```
9232
8666
30
42
```

In [12]:
```
# Printing the data
print("Number of samples:", len(input_text))
print("Number of unique input tokens:", num_encoder_tokens)
print("Number of unique output tokens:", num_decoder_tokens)
print("Max sequence length for inputs:", max_encoder_seq_length)
print("Max sequence length for outputs:", max_decoder_seq_length)
```

```
Number of samples: 18416
Number of unique input tokens: 9232
Number of unique output tokens: 8666
Max sequence length for inputs: 30
Max sequence length for outputs: 42
```

3. Model Building

3.1. Creating the Dictionary

In [13]:
```
# Creating the dictionary
input_token_index = dict([(word,i+1)for i, word in enumerate(input_char)]
target_token_index = dict([(word,i+1)for i, word in enumerate(target_char
```

In [14]:
```
reverse_input_char_index = dict((i, word) for word, i in input_token_inde
reverse_target_char_index = dict((i, word) for word, i in target_token_in
```

3.2. Creating the Batch

In [15]:
```
# Splitting the data
from sklearn.model_selection import train_test_split
X,y = df.english_sentence , df.hindi_sentence
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1,
```

In [16]:
```
# Printinng the shape
print(X_train.shape)
print(X_test.shape)
```

```
(16574,)
(1842,)
```

In [17]:
```
# Defining the batch
def generate_batch(X,y,batch_size):
    while True:
        for j in range(0, len(X), batch_size):
            encoder_input_data = np.zeros((batch_size, max_encoder_seq_le
            decoder_input_data = np.zeros((batch_size, max_decoder_seq_le
            decoder_target_data = np.zeros((batch_size, max_decoder_seq_l
            for i, (input_text, target_text) in enumerate(zip(X[j:j+batch
                for t, word in enumerate(input_text.split()):
                    encoder_input_data[i, t] = input_token_index[word] #
                for t, word in enumerate(target_text.split()):
                    if t<len(target_text.split())-1:
                        decoder_input_data[i, t] = target_token_index[wor
                    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[
            yield([encoder_input_data, decoder_input_data], decoder_targe
```

3.3. Making the Encoder

In [20]:
```
latent_dim = 50
```

In [21]:
```
# Making the Encoder
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(num_encoder_tokens, latent_dim, mask_zero=True)(encod

# Adding two more LSTM layers
encoder_lstm1 = LSTM(latent_dim, return_sequences=True, return_state=True
encoder_outputs1, state_h1, state_c1 = encoder_lstm1(enc_emb)
encoder_lstm2 = LSTM(latent_dim, return_sequences=True, return_state=True
encoder_outputs2, state_h2, state_c2 = encoder_lstm2(encoder_outputs1)
encoder_lstm3 = LSTM(latent_dim, return_sequences=True, return_state=True
encoder_outputs3, state_h3, state_c3 = encoder_lstm3(encoder_outputs2)

encoder_states = [state_h3, state_c3]
```

3.4. Making the Decoder

In [22]:
```
# Making 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)

# Adding two more LSTM layers
decoder_lstm1 = LSTM(latent_dim, return_sequences=True, return_state=True
decoder_outputs1, _, _ = decoder_lstm1(dec_emb, initial_state=encoder_sta
decoder_lstm2 = LSTM(latent_dim, return_sequences=True, return_state=True
decoder_outputs2, _, _ = decoder_lstm2(decoder_outputs1)
decoder_lstm3 = LSTM(latent_dim, return_sequences=True, return_state=True
decoder_outputs3, _, _ = decoder_lstm3(decoder_outputs2)
```

```
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs3)
```

3.5. Making the Model

In [23]:
```
# Making the model
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model.compile(optimizer='adam', loss='categorical_crossentropy',metrics=[
```

In [24]:
```
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected
to
==================================================================
 input_3 (InputLayer)        [(None, None)]            0         []

 embedding (Embedding)       (None, None, 50)          461600    ['input_3
[0][0]']

 lstm (LSTM)                 [(None, None, 50),        20200     ['embeddi
ng[0][0]']
                              (None, 50),
                              (None, 50)]

 input_4 (InputLayer)        [(None, None)]            0         []

 lstm_1 (LSTM)               [(None, None, 50),        20200     ['lstm[0]
[0]']
                              (None, 50),
                              (None, 50)]

 embedding_1 (Embedding)     (None, None, 50)          433300    ['input_4
[0][0]']

 lstm_2 (LSTM)               [(None, None, 50),        20200     ['lstm_1
[0][0]']
                              (None, 50),
                              (None, 50)]

 lstm_3 (LSTM)               [(None, None, 50),        20200     ['embeddi
ng_1[0][0]',
                              (None, 50),                        'lstm_2
[0][1]',
                              (None, 50)]                        'lstm_2
[0][2]']

 lstm_4 (LSTM)               [(None, None, 50),        20200     ['lstm_3
[0][0]']
                              (None, 50),
                              (None, 50)]

 lstm_5 (LSTM)               [(None, None, 50),        20200     ['lstm_4
[0][0]']
                              (None, 50),
                              (None, 50)]

 dense (Dense)               (None, None, 8666)        441966    ['lstm_5
[0][0]']

==================================================================
Total params: 1,458,066
Trainable params: 1,458,066
Non-trainable params: 0
_________________________________________________________________
```

4. Model Training

4.1. Training the Model

In [25]:
```
# Training the model
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 512
epochs = 100
```

4.2. Fitting the Model

In [26]:
```
# Fitting the model

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

```
/tmp/ipykernel_28/1164885454.py:3: UserWarning: `Model.fit_generator` is d
eprecated and will be removed in a future version. Please use `Model.fit`,
which supports generators.
  model.fit_generator(generator = generate_batch(X_train, y_train, batch_s
ize = batch_size),
```

```
Epoch 1/100
32/32 [==============================] - 100s 2s/step - loss: 9.0152 - ac
c: 0.0424 - val_loss: 8.7644 - val_acc: 0.0434
Epoch 2/100
32/32 [==============================] - 56s 2s/step - loss: 7.8140 - acc:
0.0449 - val_loss: 7.2922 - val_acc: 0.0434
Epoch 3/100
32/32 [==============================] - 56s 2s/step - loss: 7.0481 - acc:
0.0449 - val_loss: 7.2119 - val_acc: 0.0434
Epoch 4/100
32/32 [==============================] - 54s 2s/step - loss: 6.9833 - acc:
0.0449 - val_loss: 7.2363 - val_acc: 0.0434
Epoch 5/100
32/32 [==============================] - 56s 2s/step - loss: 6.9641 - acc:
0.0449 - val_loss: 7.2586 - val_acc: 0.0434
Epoch 6/100
32/32 [==============================] - 56s 2s/step - loss: 6.9525 - acc:
0.0447 - val_loss: 7.2800 - val_acc: 0.0434
Epoch 7/100
32/32 [==============================] - 55s 2s/step - loss: 6.9420 - acc:
0.0447 - val_loss: 7.2986 - val_acc: 0.0434
Epoch 8/100
32/32 [==============================] - 56s 2s/step - loss: 6.9329 - acc:
0.0448 - val_loss: 7.3048 - val_acc: 0.0434
Epoch 9/100
32/32 [==============================] - 57s 2s/step - loss: 6.9278 - acc:
0.0447 - val_loss: 7.3036 - val_acc: 0.0434
Epoch 10/100
32/32 [==============================] - 56s 2s/step - loss: 6.9180 - acc:
0.0450 - val_loss: 7.3042 - val_acc: 0.0434
Epoch 11/100
32/32 [==============================] - 56s 2s/step - loss: 6.9071 - acc:
0.0447 - val_loss: 7.3094 - val_acc: 0.0434
Epoch 12/100
32/32 [==============================] - 55s 2s/step - loss: 6.9037 - acc:
0.0447 - val_loss: 7.3141 - val_acc: 0.0434
Epoch 13/100
32/32 [==============================] - 55s 2s/step - loss: 6.8988 - acc:
0.0450 - val_loss: 7.3211 - val_acc: 0.0434
Epoch 14/100
32/32 [==============================] - 55s 2s/step - loss: 6.8910 - acc:
0.0451 - val_loss: 7.3294 - val_acc: 0.0434
Epoch 15/100
32/32 [==============================] - 56s 2s/step - loss: 6.8855 - acc:
0.0451 - val_loss: 7.3380 - val_acc: 0.0434
Epoch 16/100
32/32 [==============================] - 55s 2s/step - loss: 6.8806 - acc:
0.0451 - val_loss: 7.3455 - val_acc: 0.0434
Epoch 17/100
32/32 [==============================] - 55s 2s/step - loss: 6.8726 - acc:
0.0451 - val_loss: 7.3511 - val_acc: 0.0434
Epoch 18/100
32/32 [==============================] - 55s 2s/step - loss: 6.8695 - acc:
0.0450 - val_loss: 7.3594 - val_acc: 0.0434
Epoch 19/100
32/32 [==============================] - 55s 2s/step - loss: 6.8651 - acc:
0.0453 - val_loss: 7.3666 - val_acc: 0.0440
Epoch 20/100
32/32 [==============================] - 55s 2s/step - loss: 6.8551 - acc:
0.0450 - val_loss: 7.3622 - val_acc: 0.0437
```

```
Epoch 21/100
32/32 [==============================] - 55s 2s/step - loss: 6.8425 - acc:
0.0450 - val_loss: 7.3415 - val_acc: 0.0462
Epoch 22/100
32/32 [==============================] - 56s 2s/step - loss: 6.8197 - acc:
0.0537 - val_loss: 7.3188 - val_acc: 0.0542
Epoch 23/100
32/32 [==============================] - 55s 2s/step - loss: 6.7958 - acc:
0.0544 - val_loss: 7.3050 - val_acc: 0.0545
Epoch 24/100
32/32 [==============================] - 56s 2s/step - loss: 6.7755 - acc:
0.0546 - val_loss: 7.2922 - val_acc: 0.0567
Epoch 25/100
32/32 [==============================] - 56s 2s/step - loss: 6.7570 - acc:
0.0556 - val_loss: 7.2811 - val_acc: 0.0573
Epoch 26/100
32/32 [==============================] - 56s 2s/step - loss: 6.7376 - acc:
0.0561 - val_loss: 7.2786 - val_acc: 0.0573
Epoch 27/100
32/32 [==============================] - 55s 2s/step - loss: 6.7193 - acc:
0.0561 - val_loss: 7.2768 - val_acc: 0.0573
Epoch 28/100
32/32 [==============================] - 56s 2s/step - loss: 6.7034 - acc:
0.0561 - val_loss: 7.2758 - val_acc: 0.0573
Epoch 29/100
32/32 [==============================] - 55s 2s/step - loss: 6.6886 - acc:
0.0560 - val_loss: 7.2691 - val_acc: 0.0570
Epoch 30/100
32/32 [==============================] - 55s 2s/step - loss: 6.6685 - acc:
0.0564 - val_loss: 7.2625 - val_acc: 0.0567
Epoch 31/100
32/32 [==============================] - 56s 2s/step - loss: 6.6372 - acc:
0.0583 - val_loss: 7.2527 - val_acc: 0.0584
Epoch 32/100
32/32 [==============================] - 56s 2s/step - loss: 6.6087 - acc:
0.0607 - val_loss: 7.2395 - val_acc: 0.0630
Epoch 33/100
32/32 [==============================] - 54s 2s/step - loss: 6.5778 - acc:
0.0630 - val_loss: 7.2465 - val_acc: 0.0587
Epoch 34/100
32/32 [==============================] - 55s 2s/step - loss: 6.5571 - acc:
0.0655 - val_loss: 7.2297 - val_acc: 0.0604
Epoch 35/100
32/32 [==============================] - 54s 2s/step - loss: 6.5195 - acc:
0.0678 - val_loss: 7.2174 - val_acc: 0.0641
Epoch 36/100
32/32 [==============================] - 55s 2s/step - loss: 6.4849 - acc:
0.0717 - val_loss: 7.2116 - val_acc: 0.0644
Epoch 37/100
32/32 [==============================] - 56s 2s/step - loss: 6.4450 - acc:
0.0722 - val_loss: 7.2001 - val_acc: 0.0627
Epoch 38/100
32/32 [==============================] - 56s 2s/step - loss: 6.4165 - acc:
0.0739 - val_loss: 7.1778 - val_acc: 0.0666
Epoch 39/100
32/32 [==============================] - 55s 2s/step - loss: 6.3837 - acc:
0.0744 - val_loss: 7.1777 - val_acc: 0.0675
Epoch 40/100
32/32 [==============================] - 55s 2s/step - loss: 6.3466 - acc:
0.0751 - val_loss: 7.1509 - val_acc: 0.0681
```

```
Epoch 41/100
32/32 [==============================] - 56s 2s/step - loss: 6.3144 - acc:
0.0756 - val_loss: 7.1397 - val_acc: 0.0678
Epoch 42/100
32/32 [==============================] - 55s 2s/step - loss: 6.2998 - acc:
0.0761 - val_loss: 7.1388 - val_acc: 0.0678
Epoch 43/100
32/32 [==============================] - 56s 2s/step - loss: 6.2789 - acc:
0.0766 - val_loss: 7.1089 - val_acc: 0.0658
Epoch 44/100
32/32 [==============================] - 55s 2s/step - loss: 6.2399 - acc:
0.0763 - val_loss: 7.0831 - val_acc: 0.0689
Epoch 45/100
32/32 [==============================] - 56s 2s/step - loss: 6.2096 - acc:
0.0767 - val_loss: 7.0674 - val_acc: 0.0686
Epoch 46/100
32/32 [==============================] - 56s 2s/step - loss: 6.1799 - acc:
0.0774 - val_loss: 7.0584 - val_acc: 0.0683
Epoch 47/100
32/32 [==============================] - 56s 2s/step - loss: 6.1468 - acc:
0.0789 - val_loss: 7.0381 - val_acc: 0.0701
Epoch 48/100
32/32 [==============================] - 55s 2s/step - loss: 6.1194 - acc:
0.0838 - val_loss: 7.0101 - val_acc: 0.0743
Epoch 49/100
32/32 [==============================] - 56s 2s/step - loss: 6.0820 - acc:
0.0878 - val_loss: 6.9929 - val_acc: 0.0769
Epoch 50/100
32/32 [==============================] - 56s 2s/step - loss: 6.0400 - acc:
0.0907 - val_loss: 6.9971 - val_acc: 0.0811
Epoch 51/100
32/32 [==============================] - 56s 2s/step - loss: 6.0180 - acc:
0.0933 - val_loss: 6.9581 - val_acc: 0.0831
Epoch 52/100
32/32 [==============================] - 55s 2s/step - loss: 5.9953 - acc:
0.0954 - val_loss: 6.9627 - val_acc: 0.0786
Epoch 53/100
32/32 [==============================] - 56s 2s/step - loss: 5.9706 - acc:
0.0965 - val_loss: 6.9253 - val_acc: 0.0834
Epoch 54/100
32/32 [==============================] - 55s 2s/step - loss: 5.9502 - acc:
0.0988 - val_loss: 6.9419 - val_acc: 0.0822
Epoch 55/100
32/32 [==============================] - 56s 2s/step - loss: 5.9125 - acc:
0.1011 - val_loss: 6.8935 - val_acc: 0.0876
Epoch 56/100
32/32 [==============================] - 55s 2s/step - loss: 5.8788 - acc:
0.1044 - val_loss: 6.8923 - val_acc: 0.0851
Epoch 57/100
32/32 [==============================] - 55s 2s/step - loss: 5.8490 - acc:
0.1057 - val_loss: 6.8803 - val_acc: 0.0874
Epoch 58/100
32/32 [==============================] - 55s 2s/step - loss: 5.8191 - acc:
0.1068 - val_loss: 6.8741 - val_acc: 0.0862
Epoch 59/100
32/32 [==============================] - 56s 2s/step - loss: 5.7930 - acc:
0.1087 - val_loss: 6.8543 - val_acc: 0.0876
Epoch 60/100
32/32 [==============================] - 55s 2s/step - loss: 5.7599 - acc:
0.1094 - val_loss: 6.8458 - val_acc: 0.0893
```

```
Epoch 61/100
32/32 [==============================] - 55s 2s/step - loss: 5.7399 - acc:
0.1104 - val_loss: 6.8369 - val_acc: 0.0888
Epoch 62/100
32/32 [==============================] - 55s 2s/step - loss: 5.7190 - acc:
0.1103 - val_loss: 6.8532 - val_acc: 0.0868
Epoch 63/100
32/32 [==============================] - 56s 2s/step - loss: 5.6916 - acc:
0.1109 - val_loss: 6.8175 - val_acc: 0.0916
Epoch 64/100
32/32 [==============================] - 57s 2s/step - loss: 5.6550 - acc:
0.1131 - val_loss: 6.8081 - val_acc: 0.0919
Epoch 65/100
32/32 [==============================] - 56s 2s/step - loss: 5.6200 - acc:
0.1143 - val_loss: 6.8145 - val_acc: 0.0905
Epoch 66/100
32/32 [==============================] - 56s 2s/step - loss: 5.5773 - acc:
0.1174 - val_loss: 6.7716 - val_acc: 0.0933
Epoch 67/100
32/32 [==============================] - 54s 2s/step - loss: 5.5509 - acc:
0.1198 - val_loss: 6.7669 - val_acc: 0.0927
Epoch 68/100
32/32 [==============================] - 56s 2s/step - loss: 5.5091 - acc:
0.1221 - val_loss: 6.7721 - val_acc: 0.0916
Epoch 69/100
32/32 [==============================] - 56s 2s/step - loss: 5.4724 - acc:
0.1252 - val_loss: 6.7871 - val_acc: 0.0910
Epoch 70/100
32/32 [==============================] - 55s 2s/step - loss: 5.4468 - acc:
0.1265 - val_loss: 6.7489 - val_acc: 0.0947
Epoch 71/100
32/32 [==============================] - 55s 2s/step - loss: 5.4231 - acc:
0.1291 - val_loss: 6.7438 - val_acc: 0.0995
Epoch 72/100
32/32 [==============================] - 55s 2s/step - loss: 5.4027 - acc:
0.1313 - val_loss: 6.7719 - val_acc: 0.0987
Epoch 73/100
32/32 [==============================] - 55s 2s/step - loss: 5.3693 - acc:
0.1326 - val_loss: 6.7333 - val_acc: 0.1024
Epoch 74/100
32/32 [==============================] - 55s 2s/step - loss: 5.3285 - acc:
0.1371 - val_loss: 6.7402 - val_acc: 0.1010
Epoch 75/100
32/32 [==============================] - 55s 2s/step - loss: 5.2992 - acc:
0.1380 - val_loss: 6.7428 - val_acc: 0.1061
Epoch 76/100
32/32 [==============================] - 55s 2s/step - loss: 5.2662 - acc:
0.1410 - val_loss: 6.7483 - val_acc: 0.1052
Epoch 77/100
32/32 [==============================] - 55s 2s/step - loss: 5.2352 - acc:
0.1451 - val_loss: 6.7415 - val_acc: 0.1044
Epoch 78/100
32/32 [==============================] - 55s 2s/step - loss: 5.2080 - acc:
0.1459 - val_loss: 6.7210 - val_acc: 0.1044
Epoch 79/100
32/32 [==============================] - 55s 2s/step - loss: 5.1923 - acc:
0.1485 - val_loss: 6.7274 - val_acc: 0.1061
Epoch 80/100
32/32 [==============================] - 55s 2s/step - loss: 5.1753 - acc:
0.1486 - val_loss: 6.7242 - val_acc: 0.1064
```

```
Epoch 81/100
32/32 [==============================] - 55s 2s/step - loss: 5.1422 - acc:
0.1508 - val_loss: 6.6946 - val_acc: 0.1092
Epoch 82/100
32/32 [==============================] - 55s 2s/step - loss: 5.1078 - acc:
0.1548 - val_loss: 6.7009 - val_acc: 0.1064
Epoch 83/100
32/32 [==============================] - 56s 2s/step - loss: 5.0833 - acc:
0.1564 - val_loss: 6.7186 - val_acc: 0.1081
Epoch 84/100
32/32 [==============================] - 55s 2s/step - loss: 5.0670 - acc:
0.1564 - val_loss: 6.7435 - val_acc: 0.1078
Epoch 85/100
32/32 [==============================] - 54s 2s/step - loss: 5.0457 - acc:
0.1579 - val_loss: 6.7563 - val_acc: 0.1069
Epoch 86/100
32/32 [==============================] - 56s 2s/step - loss: 5.0270 - acc:
0.1585 - val_loss: 6.7261 - val_acc: 0.1081
Epoch 87/100
32/32 [==============================] - 55s 2s/step - loss: 5.0133 - acc:
0.1604 - val_loss: 6.7425 - val_acc: 0.1072
Epoch 88/100
32/32 [==============================] - 55s 2s/step - loss: 4.9836 - acc:
0.1603 - val_loss: 6.7886 - val_acc: 0.1072
Epoch 89/100
32/32 [==============================] - 55s 2s/step - loss: 4.9647 - acc:
0.1626 - val_loss: 6.7694 - val_acc: 0.1095
Epoch 90/100
32/32 [==============================] - 55s 2s/step - loss: 4.9527 - acc:
0.1636 - val_loss: 6.7138 - val_acc: 0.1123
Epoch 91/100
32/32 [==============================] - 55s 2s/step - loss: 4.9276 - acc:
0.1663 - val_loss: 6.7197 - val_acc: 0.1140
Epoch 92/100
32/32 [==============================] - 55s 2s/step - loss: 4.8895 - acc:
0.1691 - val_loss: 6.7739 - val_acc: 0.1157
Epoch 93/100
32/32 [==============================] - 56s 2s/step - loss: 4.8598 - acc:
0.1713 - val_loss: 6.7551 - val_acc: 0.1134
Epoch 94/100
32/32 [==============================] - 56s 2s/step - loss: 4.8466 - acc:
0.1726 - val_loss: 6.7238 - val_acc: 0.1140
Epoch 95/100
32/32 [==============================] - 55s 2s/step - loss: 4.8257 - acc:
0.1743 - val_loss: 6.7564 - val_acc: 0.1140
Epoch 96/100
32/32 [==============================] - 56s 2s/step - loss: 4.7945 - acc:
0.1764 - val_loss: 6.7820 - val_acc: 0.1134
Epoch 97/100
32/32 [==============================] - 56s 2s/step - loss: 4.7722 - acc:
0.1784 - val_loss: 6.7437 - val_acc: 0.1143
Epoch 98/100
32/32 [==============================] - 56s 2s/step - loss: 4.7556 - acc:
0.1794 - val_loss: 6.7698 - val_acc: 0.1134
Epoch 99/100
32/32 [==============================] - 55s 2s/step - loss: 4.7306 - acc:
0.1820 - val_loss: 6.7746 - val_acc: 0.1177
Epoch 100/100
32/32 [==============================] - 55s 2s/step - loss: 4.7165 - acc:
0.1840 - val_loss: 6.7439 - val_acc: 0.1149
```

Out[26]:
```
<keras.callbacks.History at 0x7e619c33b9d0>
```

5. Model Evaluation

5.1. Printing the Validation Score

In [27]:
```
# Printing the validation score
model.evaluate_generator(generator = generate_batch(X_test, y_test, batch
```

```
/tmp/ipykernel_28/2723858562.py:2: UserWarning: `Model.evaluate_generator`
is deprecated and will be removed in a future version. Please use `Model.e
valuate`, which supports generators.
  model.evaluate_generator(generator = generate_batch(X_test, y_test, batc
h_size = batch_size), steps = val_samples//batch_size)
```

Out[27]:
```
[6.74385404586792, 0.11486103385686874]
```

5.2. Saving the Weights

In [28]:
```
# Saving the weights
model.save_weights('nmt_eng_hin_translation.h5')
```

6. Model Prediction

6.1. Making the Inference Model

In [29]:
```
# Inference

encoder_model = Model(encoder_inputs,encoder_states)
```

6.2. Making the Decoder Model

In [30]:
```
# Decoder Setup
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]
```

In [32]:
```
dec_emb2 = dec_emb_layer(decoder_inputs) # Get the embeddings of the deco
decoder_outputs2, state_h2, state_c2 = decoder_lstm3(dec_emb2, initial_st
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) # A dense softmax laye

# Final decoder model
decoder_model = Model([decoder_inputs] + decoder_states_inputs,[decoder_o
```

In [33]:
```
def decode_sequence(input_seq):
    # Encode the input as state vectors.
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1,1))
    # Populate the first character of target sequence with the start char
    #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
        # 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) > 25):
            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
```

6.3. Printing the Predicted Output

In [34]:
```
val_gen = generate_batch(X_test, y_test, batch_size = 1)
k=-1
```

In [35]:
```
k+=2
(input_seq, actual_output), _ = next(val_gen)
decoded_sentence = decode_sequence(input_seq)
print('Input English sentence:', X_test[k:k+1].values[0])
print('Actual Hindi Translation:', y_test[k:k+1].values[0])
print('Predicted Hindi Translation:', decoded_sentence[:])
```

```
1/1 [==============================] - 4s 4s/step
1/1 [==============================] - 1s 1s/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 22ms/step
Input English sentence: class west bengal
Actual Hindi Translation: श्रेणीपश्चिम बंगाल
Predicted Hindi Translation:  झंडा झंडा झंडा झंडा झंडा झंडा
```

In [ ]: