1. Student Information

- **Name:** Apoorv Gupta
- **PRN:** 21070126018
- **Batch:** AIML - A1
- **Git Repo:** GitHub Repository (https://github.com/erApoorvGupta/NLP_assignments)

```
In [1]: from google.colab import drive
```

```
In [2]: drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
In [3]: import tensorflow as tf
        from tensorflow.keras.layers import Embedding,LSTM,Dense,RepeatVector,TimeDistributed,Input
        from tensorflow.keras.models import Model
        from tensorflow.keras.losses import sparse_categorical_crossentropy
        import pandas as pd
        import re
        import string
        from string import digits
        import numpy as np
```

```
In [4]: data=pd.read_csv('/content/drive/MyDrive/Colab Notebooks/NLP_LAB/Hindi_English_Truncated_Corpus.csv')
        data['source'].value_counts()
```

```
Out[4]: tides        50000
        ted          39881
        indic2012    37726
        Name: source, dtype: int64
```

```
In [5]: data=data[(data.english_sentence.apply(lambda x: len(str(x))<=30))&
                  (data.hindi_sentence.apply(lambda x: len(str(x))<=30))]
```

In [6]:
```
## changing uppercase to lowercase
data['english_sentence']=data['english_sentence'].apply(lambda x: str(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))

to_exclude=set(string.punctuation) #set of all special character
print("punctuations to exclude::",to_exclude)

#remove all the special characters
data['english_sentence']=data['english_sentence'].apply(lambda x:''.join(ch for ch in x if ch not in to_exclude))

data['hindi_sentence']=data['hindi_sentence'].apply(lambda x:''.join(ch for ch in x if ch not in to_exclude))
```

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

In [7]:
```
from string import digits
#Remove all numbers from text
remove_digits=str.maketrans('','',digits)
data['hindi_sentence']=data['hindi_sentence'].apply(lambda x:x.translate(remove_digits))

data['hindi_sentence']=data['hindi_sentence'].apply(lambda x: x.translate(remove_digits))

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

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

In [8]:
```
data.head()
```

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

In [9]:
```
input_text=[]
target_text=[]
input_characters=set()
target_characters=set()

for eng, hin in data[['english_sentence','hindi_sentence']].itertuples(index=False):
  target='START_'+ hin +'_END' #end sequence
  input_text.append(eng)
  target_text.append(target)

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

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

In [10]:
```
print(len(input_text))
print(len(target_text))
print(len(input_characters))
print(len(target_characters))
```

```
18416
18416
9729
8665
```

In [11]:
```
print("Input Text ->>>>>"+input_text[0] + "->>>>>>> Output Text ->>>>>>>"+target_text[0])
```

```
Input Text ->>>>>category religious text->>>>>>> Output Text ->>>>>>>START_श्रेणीधर्मग्रन्थ_END
```

In [12]:
```
input_char=sorted(list(input_characters))
target_char=sorted(list(target_characters))

num_encoder_tokens=len(input_characters)
num_decoder_tokens=len(target_characters)+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])
```

In [13]:
```
print('Number of samples:',len(input_text))
print('Number of unique input tokens:',num_encoder_tokens)
print('Number of unique tokens output tokens:',num_encoder_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: 9729
Number of unique tokens output tokens: 9729
Max sequence length for inputs: 30
Max sequence length for outputs: 40
```

In [14]:
```
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 [15]:
```
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())
```

In [16]:
```
import pickle
pickle.dump(input_token_index, open('eng_input_token_index.pickle','wb'),protocol=pickle.HIGHEST_PROTOCOL)
pickle.dump(target_token_index, open('hin_target_token_index.pickle','wb'),protocol=pickle.HIGHEST_PROTOCOL)
pickle.dump(reverse_input_char_index, open('eng_reverse_input_char_index.pickle','wb'), protocol=pickle.HIGHEST_PROTOCOL)
pickle.dump(reverse_target_char_index, open('hin_reverse_target_char_index.pickle','wb'), protocol=pickle.HIGHEST_PROTOCOL)
```

In [17]:
```
with open('eng_input_token_index.pickle','rb') as fp:
  input_token_index = pickle.load(fp)
with open('hin_target_token_index.pickle','rb') as fp:
  target_token_index = pickle.load(fp)
with open('eng_reverse_input_char_index.pickle','rb') as fp:
  reverse_input_char_index = pickle.load(fp)
with open('hin_reverse_target_char_index.pickle','rb') as fp:
  reverse_target_char_index = pickle.load(fp)
```

In [18]:
```
from sklearn.model_selection import train_test_split
X, y = data.english_sentence, data.hindi_sentence
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.1,random_state=2)
X_train.shape, X_test.shape
```

Out[18]:
```
((16574,), (1842,))
```

In [19]:
```
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_length),dtype='float32')
      decoder_input_data = np.zeros((batch_size,max_decoder_seq_length),dtype='float32')
      decoder_target_data = np.zeros((batch_size, max_decoder_seq_length,num_decoder_tokens),dtype='float32')
      for i,(input_text, target_text) in enumerate(zip(X[j:j+batch_size],y[j:j+batch_size])):
        for t, word in enumerate(input_text.split()):
          encoder_input_data[i, t] = input_token_index[word] # encoder input seq
          for t, word in enumerate(target_text.split()):
            if t<len(target_text.split())-1:
              decoder_input_data[i, t] = target_token_index[word] # decoder input_seq
            if t>0:
              decoder_target_data[i, t - 1, target_token_index[word]] = 1
              yield([encoder_input_data, decoder_input_data], decoder_target_data)
```

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

In [21]:
```
# Encoder
encoder_inputs = Input(shape=(None,))
enc_emb = Embedding(num_encoder_tokens, latent_dim, mask_zero =True)(encoder_inputs)
encoder_lstm = LSTM(latent_dim, return_state=True)
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
encoder_states = [state_h, state_c]
```

In [22]:
```
# 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_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)
decoder_outputs, _, _ = decoder_lstm(dec_emb,initial_state=encoder_states)
decoder_dense = Dense(num_decoder_tokens, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

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

In [24]:

```
model.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to                  
==================================================================================================
 input_1 (InputLayer)        [(None, None)]               0         []                            
                                                                                                  
 input_2 (InputLayer)        [(None, None)]               0         []                            
                                                                                                  
 embedding (Embedding)       (None, None, 50)             486450    ['input_1[0][0]']             
                                                                                                  
 embedding_1 (Embedding)     (None, None, 50)             433300    ['input_2[0][0]']             
                                                                                                  
 lstm (LSTM)                 [(None, 50),                 20200     ['embedding[0][0]']           
                              (None, 50),                                                         
                              (None, 50)]                                                         
                                                                                                  
 lstm_1 (LSTM)               [(None, None, 50),           20200     ['embedding_1[0][0]',         
                              (None, 50),                            'lstm[0][1]',                
                              (None, 50)]                            'lstm[0][2]']                
                                                                                                  
 dense (Dense)               (None, None, 8666)           441966    ['lstm_1[0][0]']              
                                                                                                  
==================================================================================================
Total params: 1402116 (5.35 MB)
Trainable params: 1402116 (5.35 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

In [25]:

```
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 512
epochs = 45
```

In [26]:

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

```
<ipython-input-26-c06b0b25cab3>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, w
hich supports generators.
  model.fit_generator(
Epoch 1/45
32/32 [==============================] - 60s 1s/step - loss: 8.8221 - acc: 0.3498 - val_loss: 6.5481 - val_acc: 0.0000e+00
Epoch 2/45
32/32 [==============================] - 39s 1s/step - loss: 7.7786 - acc: 0.1832 - val_loss: 9.0925 - val_acc: 0.0000e+00
Epoch 3/45
32/32 [==============================] - 38s 1s/step - loss: 4.8414 - acc: 0.0671 - val_loss: 8.5060 - val_acc: 0.0000e+00
Epoch 4/45
32/32 [==============================] - 39s 1s/step - loss: 4.7249 - acc: 0.0901 - val_loss: 10.6664 - val_acc: 0.0000e+00
Epoch 5/45
32/32 [==============================] - 38s 1s/step - loss: 4.7313 - acc: 0.0802 - val_loss: 10.4426 - val_acc: 0.0000e+00
Epoch 6/45
32/32 [==============================] - 39s 1s/step - loss: 4.3228 - acc: 0.0729 - val_loss: 10.3772 - val_acc: 0.0000e+00
Epoch 7/45
32/32 [==============================] - 38s 1s/step - loss: 4.3608 - acc: 0.0800 - val_loss: 10.6159 - val_acc: 0.1111
Epoch 8/45
32/32 [==============================] - 42s 1s/step - loss: 4.2842 - acc: 0.0865 - val_loss: 10.8135 - val_acc: 0.1111
Epoch 9/45
32/32 [==============================] - 38s 1s/step - loss: 4.3700 - acc: 0.0983 - val_loss: 10.8797 - val_acc: 0.1111
Epoch 10/45
32/32 [==============================] - 39s 1s/step - loss: 4.2796 - acc: 0.0947 - val_loss: 9.6401 - val_acc: 0.0833
Epoch 11/45
32/32 [==============================] - 39s 1s/step - loss: 4.2119 - acc: 0.0837 - val_loss: 9.8750 - val_acc: 0.1429
Epoch 12/45
32/32 [==============================] - 39s 1s/step - loss: 4.0408 - acc: 0.0969 - val_loss: 9.6689 - val_acc: 0.1277
Epoch 13/45
32/32 [==============================] - 38s 1s/step - loss: 4.0175 - acc: 0.0897 - val_loss: 9.9451 - val_acc: 0.0588
Epoch 14/45
32/32 [==============================] - 37s 1s/step - loss: 3.9953 - acc: 0.1090 - val_loss: 9.7832 - val_acc: 0.0536
Epoch 15/45
32/32 [==============================] - 40s 1s/step - loss: 3.8381 - acc: 0.1341 - val_loss: 9.7122 - val_acc: 0.0441
Epoch 16/45
32/32 [==============================] - 39s 1s/step - loss: 3.6766 - acc: 0.1416 - val_loss: 10.5232 - val_acc: 0.0435
Epoch 17/45
32/32 [==============================] - 40s 1s/step - loss: 3.5252 - acc: 0.1546 - val_loss: 10.3000 - val_acc: 0.0435
Epoch 18/45
32/32 [==============================] - 39s 1s/step - loss: 3.3050 - acc: 0.2295 - val_loss: 10.2817 - val_acc: 0.0435
Epoch 19/45
32/32 [==============================] - 38s 1s/step - loss: 3.2064 - acc: 0.2980 - val_loss: 10.0876 - val_acc: 0.0870
Epoch 20/45
32/32 [==============================] - 39s 1s/step - loss: 3.0795 - acc: 0.3702 - val_loss: 10.0842 - val_acc: 0.0870
Epoch 21/45
32/32 [==============================] - 39s 1s/step - loss: 2.7755 - acc: 0.4695 - val_loss: 9.9064 - val_acc: 0.0741
Epoch 22/45
32/32 [==============================] - 40s 1s/step - loss: 2.6306 - acc: 0.5467 - val_loss: 10.1314 - val_acc: 0.0741
Epoch 23/45
32/32 [==============================] - 39s 1s/step - loss: 2.5332 - acc: 0.5807 - val_loss: 10.0346 - val_acc: 0.0370
Epoch 24/45
32/32 [==============================] - 38s 1s/step - loss: 2.2585 - acc: 0.6406 - val_loss: 9.7859 - val_acc: 0.0667
Epoch 25/45
32/32 [==============================] - 39s 1s/step - loss: 1.9497 - acc: 0.7553 - val_loss: 10.0898 - val_acc: 0.0667
Epoch 26/45
32/32 [==============================] - 39s 1s/step - loss: 1.7525 - acc: 0.7982 - val_loss: 10.0110 - val_acc: 0.0947
```

```
Epoch 27/45
32/32 [==============================] - 37s 1s/step - loss: 1.6054 - acc: 0.8151 - val_loss: 10.4166 - val_acc: 0.0857
Epoch 28/45
32/32 [==============================] - 39s 1s/step - loss: 1.5162 - acc: 0.8316 - val_loss: 10.4985 - val_acc: 0.0857
Epoch 29/45
32/32 [==============================] - 37s 1s/step - loss: 1.3193 - acc: 0.8563 - val_loss: 10.9270 - val_acc: 0.0857
Epoch 30/45
32/32 [==============================] - 39s 1s/step - loss: 1.1696 - acc: 0.8918 - val_loss: 11.1906 - val_acc: 0.0571
Epoch 31/45
32/32 [==============================] - 39s 1s/step - loss: 1.1254 - acc: 0.9067 - val_loss: 10.9966 - val_acc: 0.0561
Epoch 32/45
32/32 [==============================] - 38s 1s/step - loss: 1.0864 - acc: 0.9155 - val_loss: 10.6090 - val_acc: 0.0783
Epoch 33/45
32/32 [==============================] - 39s 1s/step - loss: 0.9255 - acc: 0.9311 - val_loss: 10.2736 - val_acc: 0.0756
Epoch 34/45
32/32 [==============================] - 40s 1s/step - loss: 0.9170 - acc: 0.9299 - val_loss: 9.8365 - val_acc: 0.0746
Epoch 35/45
32/32 [==============================] - 37s 1s/step - loss: 0.9112 - acc: 0.9210 - val_loss: 9.6783 - val_acc: 0.0889
Epoch 36/45
32/32 [==============================] - 41s 1s/step - loss: 0.8662 - acc: 0.9192 - val_loss: 9.8693 - val_acc: 0.0667
Epoch 37/45
32/32 [==============================] - 39s 1s/step - loss: 0.7992 - acc: 0.9377 - val_loss: 9.6326 - val_acc: 0.0667
Epoch 38/45
32/32 [==============================] - 40s 1s/step - loss: 0.7830 - acc: 0.9368 - val_loss: 9.6263 - val_acc: 0.0876
Epoch 39/45
32/32 [==============================] - 40s 1s/step - loss: 0.7534 - acc: 0.9400 - val_loss: 9.8826 - val_acc: 0.0816
Epoch 40/45
32/32 [==============================] - 38s 1s/step - loss: 0.6672 - acc: 0.9499 - val_loss: 9.8719 - val_acc: 0.0612
Epoch 41/45
32/32 [==============================] - 40s 1s/step - loss: 0.6442 - acc: 0.9540 - val_loss: 9.9339 - val_acc: 0.0408
Epoch 42/45
32/32 [==============================] - 39s 1s/step - loss: 0.6343 - acc: 0.9485 - val_loss: 9.7375 - val_acc: 0.0592
Epoch 43/45
32/32 [==============================] - 36s 1s/step - loss: 0.5355 - acc: 0.9589 - val_loss: 9.6847 - val_acc: 0.0755
Epoch 44/45
32/32 [==============================] - 39s 1s/step - loss: 0.4621 - acc: 0.9676 - val_loss: 9.7306 - val_acc: 0.1132
Epoch 45/45
32/32 [==============================] - 36s 1s/step - loss: 0.4549 - acc: 0.9721 - val_loss: 9.7260 - val_acc: 0.0755
```

Out[26]:
```
<keras.src.callbacks.History at 0x78b9e2ca6b90>
```

In [30]:
```
model.save_weights('nmt_eng_hin_translation.h5')
```

In [31]:
```
encoder_model = Model(encoder_inputs, encoder_states)
```

In [32]:

```
# 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]
```

In [33]:

```
dec_emb2= dec_emb_layer(decoder_inputs) # Get the embeddings of the decoder sequence
# To predict the next word in the sequence, set the initial states to the states from the previous time step
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2,initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]
decoder_outputs2 = decoder_dense(decoder_outputs2) # A dense softmax layer to generate prob dist. over the target vocabulary
# Final decoder model
decoder_model = Model([decoder_inputs] + decoder_states_inputs,[decoder_outputs2] + decoder_states2)
```

In [34]:

```
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)
    target_seq = np.zeros((1, 1))
    #target_seq[0, 0] = target_token_index['START_']  # Start with the START_ token
    decoded_sentence = ''

    while True:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = reverse_target_char_index[sampled_token_index]

        if sampled_char == '_END' or len(decoded_sentence.split()) > max_decoder_seq_length:
            break

        decoded_sentence += ' ' + sampled_char
        target_seq = np.zeros((1, 1))
        target_seq[0, 0] = sampled_token_index
        states_value = [h, c]

    return decoded_sentence.strip()

# Now you can use the decode_sequence function without running endlessly
val_gen = generate_batch(X_test, y_test, batch_size=1)
k = -1

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)
```