

```
import pandas as pd
import numpy as np
import string
from string import digits
import matplotlib.pyplot as plt
%matplotlib inline
import re
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense
from tensorflow.keras.models import Model
```

```
lines= pd.read_table('/content/hin.txt', names=['eng', 'hin', ' '])
lines.shape
```

(2779, 3)

```
# Lowercase all characters
lines.eng=lines.eng.apply(lambda x: x.lower())
lines.hin=lines.hin.apply(lambda x: x.lower())

# Remove quotes
lines.eng=lines.eng.apply(lambda x: re.sub('"', '', x))
lines.hin=lines.hin.apply(lambda x: re.sub('"', '', x))

exclude = set(string.punctuation) # Set of all special characters
# Remove all the special characters
lines.eng=lines.eng.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))
lines.hin=lines.hin.apply(lambda x: ''.join(ch for ch in x if ch not in exclude))

# Remove all numbers from text
remove_digits = str.maketrans('', '', digits)
lines.eng=lines.eng.apply(lambda x: x.translate(remove_digits))
lines.hin = lines.hin.apply(lambda x: re.sub("[२३०८१५७९४६]", "", x))

# Remove extra spaces
lines.eng=lines.eng.apply(lambda x: x.strip())
lines.hin=lines.hin.apply(lambda x: x.strip())
lines.eng=lines.eng.apply(lambda x: re.sub(" +", " ", x))
lines.hin=lines.hin.apply(lambda x: re.sub(" +", " ", x))

lines.hin = lines.hin.apply(lambda x : 'START_ '+ x + ' _END')

lines.sample(10)
```

	eng	hin	
2514	i have lots of work to clear up by the weekend	START_ मुझे इस ह े बसत सारा काम कर के खतम ...	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
2523	sometimes she tried talking to him about india	START_ वह कभीकभी उससे भारत की बात करने की कोश...	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
81	how are you	START_ आप कै सी ह _END	CC-BY 2.0 (France) Attribution: tatoeba.org #3...
965	what happened last night	START_ कल रात 5आ _END	CC-BY 2.0 (France) Attribution: tatoeba.org
#2... 2353	during the war we had to do without sugar	START_ जंग के समय म हमे चीनी के िबना ही काम ...	CC-BY 2.0 (France) Attribution: tatoeba.org
#2... 1405	can you teach me how to steal	START_ ा तुम मुझे चोरी करना िसखा सकते हो _END	CC-BY 2.0 (France) Attribution: tatoeba.org
#3...			
301	he began to shout	START_ वह िचवाने लगा। _END	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
1886	i learned french instead of german	START_ मने जमन की बजाय ानसीसी सीखी । _END	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
305	he is walking now	START_ वह अब चल रहा है। _END	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
60	im coming	START_ म आ रहा 5। _END	CC-BY 2.0 (France) Attribution: tatoeba.org #5

```

# Vocabulary of English
all_eng_words=set()
for eng in lines.eng:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

# Vocabulary of Hindi
all_hindi_words=set()
for hin in lines.hin:
    for word in hin.split():
        if word not in all_hindi_words:
            all_hindi_words.add(word)

# Max Length of source sequence
lenght_list=[]
for l in lines.eng:
    lenght_list.append(len(l.split(' ')))
max_length_src = np.max(lenght_list)
max_length_src
print(max_length_src)

# Max Length of target sequence
lenght_list=[]
for l in lines.hin:
    lenght_list.append(len(l.split(' ')))
max_length_tar = np.max(lenght_list)
max_length_tar
print(max_length_tar)

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

num_decoder_tokens += 1 # For zero padding
num_decoder_tokens

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)])

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(input_token_index)
print(target_token_index)
print(reverse_input_char_index)
print(reverse_target_char_index)

22
27
{'a': 1, 'abandoned': 2, 'ability': 3, 'ablaze': 4, 'able': 5, 'about': 6, 'above': 7, 'abroad': 8, 'absence': 9, 'absent': 10, 'absolut
{'START_': 1, '_END': 2, 'a': 3, 'b': 4, 'i': 5, 'अंडे': 6, 'अंकल': 7, 'अंगर': 8, 'अंगूर': 9, 'अं ेज़ी': 10, 'अं ेजी': 11, 'अंडे': 12, 'अंत': 13
{1: 'a', 2: 'abandoned', 3: 'ability', 4: 'ablaze', 5: 'able', 6: 'about', 7: 'above', 8: 'abroad', 9: 'absence', 10: 'absent', 11: 'abs
{1: 'START_', 2: '_END', 3: 'a', 4: 'b', 5: 'i', 6: 'अंडे', 7: 'अंकल', 8: 'अंगर', 9: 'अंगूर', 10: 'अं ेज़ी', 11: 'अं ेजी', 12: 'अंडे', 13: 'अंत'

lines = shuffle(lines)
lines.head(10)

```

```

                                eng                                hin
1346                i gave up smoking for a year    START_ मने एक साल के िलए िसगरेट पीना छोड़ िद... CC-BY 2.0 (France) Attribution: tatoeba.org #3...

# Train - Test Split
X, y = lines.eng, lines.hin
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
X_train.shape, X_test.shape

((2223,), (556,))
2534    when did america become independent of england    START_ अेीका को ई ड से आज़ादी कब िमली _END    CC-BY 2.0 (France) Attribution: tatoeba.org #6...

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='float32')
            decoder_input_data = np.zeros((batch_size, max_length_tar), dtype='float32')
            decoder_target_data = np.zeros((batch_size, max_length_tar, 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 the encoder input data, it maps each word to its corresponding index (according to input_token_index) and stores
                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 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])
            #the yield keyword to perform iteration inside a while True: loop, so each time Keras calls the generator, it gets a batch of dat

generate_batch(X_train, y_train, batch_size = batch_size)

<generator object generate_batch at 0x7c7f250800b0>

latent_dim = 20 #hidden layer dimension

# Encoder
encoder_inputs = Input(shape=(None,)) #specifies that the input will be sequences of varying lengths (indicated by shape=(None,))
enc_emb = Embedding(num_encoder_tokens+1, latent_dim, mask_zero = True)(encoder_inputs) #to convert the input sequences into dense vectors
encoder_lstm = LSTM(latent_dim, return_state=True) #LSTM layer should return the internal states along with the output.
encoder_outputs, state_h, state_c = encoder_lstm(enc_emb)
# We discard `encoder_outputs` and only keep the states.
encoder_states = [state_h, state_c] #the hidden state (state_h) and cell state (state_c) of the LSTM at the last time step are retained as th
#These states will be passed to the decoder part of the model to initialize the decoder's internal state.

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(num_decoder_tokens, latent_dim, mask_zero = True)
dec_emb = dec_emb_layer(decoder_inputs)
# We set up our decoder to return full output sequences,
# and to return internal states as well. We don't use the
# return states in the training model, but we will use them in inference.
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)

# Define the model that will turn
# `encoder_input_data` & `decoder_input_data` into `decoder_target_data`
model = Model([encoder_inputs, decoder_inputs], decoder_outputs)

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])

train_samples = len(X_train)
val_samples = len(X_test)

```

```
batch_size = 128
epochs = 150
```

```
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)
```

```
Epoch 1/150
<ipython-input-54-b22c3ec5e69e>:1: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use
    model.fit_generator(generator = generate_batch(X_train, y_train, batch_size = batch_size),
17/17 [=====] - 20s 568ms/step - loss: 7.9862 - acc: 0.1133 - val_loss: 7.9717 - val_acc: 0.1290
Epoch 2/150
17/17 [=====] - 8s 509ms/step - loss: 7.9530 - acc: 0.1279 - val_loss: 7.9235 - val_acc: 0.1308
Epoch 3/150
17/17 [=====] - 6s 375ms/step - loss: 7.8683 - acc: 0.1284 - val_loss: 7.7806 - val_acc: 0.1286
Epoch 4/150
17/17 [=====] - 8s 494ms/step - loss: 7.6607 - acc: 0.1281 - val_loss: 7.5100 - val_acc: 0.1309
Epoch 5/150
17/17 [=====] - 6s 377ms/step - loss: 7.3611 - acc: 0.1282 - val_loss: 7.1958 - val_acc: 0.1307
Epoch 6/150
17/17 [=====] - 8s 494ms/step - loss: 7.0568 - acc: 0.1279 - val_loss: 6.9069 - val_acc: 0.1290
Epoch 7/150
17/17 [=====] - 6s 378ms/step - loss: 6.7786 - acc: 0.1281 - val_loss: 6.6609 - val_acc: 0.1308
Epoch 8/150
17/17 [=====] - 8s 508ms/step - loss: 6.5484 - acc: 0.1284 - val_loss: 6.4505 - val_acc: 0.1286
Epoch 9/150
17/17 [=====] - 7s 390ms/step - loss: 6.3723 - acc: 0.1279 - val_loss: 6.3041 - val_acc: 0.1309
Epoch 10/150
17/17 [=====] - 8s 497ms/step - loss: 6.2378 - acc: 0.1285 - val_loss: 6.1841 - val_acc: 0.1307
Epoch 11/150
17/17 [=====] - 7s 420ms/step - loss: 6.1412 - acc: 0.1278 - val_loss: 6.1227 - val_acc: 0.1290
Epoch 12/150
17/17 [=====] - 8s 506ms/step - loss: 6.0757 - acc: 0.1283 - val_loss: 6.0685 - val_acc: 0.1308
Epoch 13/150
17/17 [=====] - 6s 379ms/step - loss: 6.0241 - acc: 0.1280 - val_loss: 6.0128 - val_acc: 0.1286
Epoch 14/150
17/17 [=====] - 8s 507ms/step - loss: 5.9884 - acc: 0.1279 - val_loss: 5.9873 - val_acc: 0.1309
Epoch 15/150
17/17 [=====] - 7s 391ms/step - loss: 5.9621 - acc: 0.1284 - val_loss: 5.9608 - val_acc: 0.1307
Epoch 16/150
17/17 [=====] - 8s 476ms/step - loss: 5.9453 - acc: 0.1279 - val_loss: 5.9757 - val_acc: 0.1290
Epoch 17/150
17/17 [=====] - 7s 391ms/step - loss: 5.9311 - acc: 0.1282 - val_loss: 5.9677 - val_acc: 0.1308
Epoch 18/150
17/17 [=====] - 8s 483ms/step - loss: 5.9246 - acc: 0.1287 - val_loss: 5.9440 - val_acc: 0.1286
Epoch 19/150
17/17 [=====] - 7s 390ms/step - loss: 5.9197 - acc: 0.1281 - val_loss: 5.9372 - val_acc: 0.1309
Epoch 20/150
17/17 [=====] - 8s 483ms/step - loss: 5.9043 - acc: 0.1279 - val_loss: 5.9245 - val_acc: 0.1307
Epoch 21/150
17/17 [=====] - 8s 469ms/step - loss: 5.8888 - acc: 0.1284 - val_loss: 5.9423 - val_acc: 0.1290
Epoch 22/150
17/17 [=====] - 7s 410ms/step - loss: 5.8843 - acc: 0.1281 - val_loss: 5.9296 - val_acc: 0.1308
Epoch 23/150
17/17 [=====] - 8s 488ms/step - loss: 5.8645 - acc: 0.1282 - val_loss: 5.8961 - val_acc: 0.1286
Epoch 24/150
17/17 [=====] - 7s 388ms/step - loss: 5.8525 - acc: 0.1279 - val_loss: 5.8826 - val_acc: 0.1309
Epoch 25/150
17/17 [=====] - 8s 504ms/step - loss: 5.8301 - acc: 0.1281 - val_loss: 5.8622 - val_acc: 0.1307
Epoch 26/150
17/17 [=====] - 7s 391ms/step - loss: 5.8134 - acc: 0.1284 - val_loss: 5.8749 - val_acc: 0.1290
Epoch 27/150
17/17 [=====] - 9s 551ms/step - loss: 5.8061 - acc: 0.1279 - val_loss: 5.8630 - val_acc: 0.1308
Epoch 28/150
```

```
generate_batch(X_train, y_train, batch_size = batch_size)
```

```
<generator object generate_batch at 0x7cf250802e0>
```

```
model.save_weights('nmt_weights.h5')
```

```
model.load_weights('nmt_weights.h5')
```

```

# Encode the input sequence to get the "thought vectors"
encoder_model = 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 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)

```

## ▼ Decode sample sequences

```

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

```

## ▼ Evaluation on Train Dataset

```

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

```

```

k+=1

```

```

(input_seq, actual_output), _ = next(train_gen)
print(input_seq)
print(actual_output)
decoded_sentence = decode_sequence(input_seq)
print(decoded_sentence)
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])

[[2041. 1267. 2237. 1016. 813. 1191. 0. 0. 0. 0. 0. 0.
  0. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[1.000e+00 2.567e+03 1.550e+02 2.610e+03 4.800e+02 2.602e+03 4.220e+02
  1.487e+03 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00
  0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00 0.000e+00]]
1/1 [=====] - 1s 1s/step
1/1 [=====] - 2s 2s/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 24ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 25ms/step
1/1 [=====] - 0s 26ms/step
1/1 [=====] - 0s 23ms/step
1/1 [=====] - 0s 44ms/step
1/1 [=====] - 0s 28ms/step
1/1 [=====] - 0s 23ms/step
हम ने ने सारी िलए कम को सकते पर चाहिए। _END
Input English sentence: the men went hunting for lions
Actual Hindi Translation: वे आदमी शेरों का िशकार करने िनकले।
Predicted Hindi Translation: हम ने ने सारी िलए कम को सकते पर चाहिए।

```