Machine Translation

Develop a model that can translate a Tamil input sentence to English.

- Analyze the impact of various embedding techniques in the translation task.
- Use sequence model to translate the Tamil input sentence and analyze the performance of various techniques.
- Translate the input sentences using Transformer model and understand its functioning.
- Translate the following Tamil Phrase and evaluate your model based on the expected output.
  - Input 1: நான் மிகவும் சந்த ாஷமாக இருக்கிதேன்
    - Expected Output: Im so happy
  - Input 2: அது அவசியமில்லை
    - Expected Output: It wasnt necessary
  - Input 3: யவுசசய்து அல மீண்டும் சசய்யவும்
    - Expected Output: Please do that again
  - Input 4: அது ஒரு நல்ை தயாசலை
    - Expected Output: That is a good idea
  - Input 5: அவர்கள் ஒன்ோக தவலை சசய்ய ஒப்புக்சகாண்டைர்
    - Expected Output: They agreed to work together

Import Libraries

In [ ]:
```
import keras
import string
import numpy as np
import pandas as pd
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential, Model
from keras.layers import LSTM, Embedding, RepeatVector, Dense, Input
```

Preprocessing the Text

In [ ]:
```
def read_sentences(path):
    with open(path, 'r', encoding='utf-8') as f:
        lines = f.read().split('\n')
    return lines
```

In [ ]:
```
english_path = "/kaggle/input/tamiltoenglish/data.en"
tamil_path = "/kaggle/input/tamiltoenglish/data.ta"

english_sentences = []
tamil_sentences = []

for i in range(1, 7):
    english_sentences.extend(read_sentences(english_path + str(i)))
    tamil_sentences.extend(read_sentences(tamil_path + str(i)))
```

In [ ]:
```
path = "/kaggle/input/tamiltoenglish/tam.txt"

with open(path, 'r', encoding='utf-8') as f:
    lines = f.read().split('\n')
```

In [ ]:
```
for line in lines[:-1]:
    english_sent, tamil_sent, _ = line.split('\t')

    tamil_sentences.append(tamil_sent)
    english_sentences.append(english_sent)
```

In [ ]:
```
english_sentences = english_sentences[:10000]
tamil_sentences = tamil_sentences[:10000]
```

In [ ]:
```
english_sentences = ['START_ ' + sent.lower().translate(str.maketrans('', '', string.punctuation)) + ' _END' 
tamil_sentences = [sent.translate(str.maketrans('', '', string.punctuation)) for sent in tamil_sentences]
```

In [ ]:
```
lines = pd.DataFrame({'tamil': tamil_sentences, 'english': english_sentences})
```

In [ ]:
```
lines['length_eng_sentence']=lines['english'].apply(lambda x:len(x.split(" ")))
lines['length_tam_sentence']=lines['tamil'].apply(lambda x:len(x.split(" ")))
```

In [ ]:
```
lines=lines[lines['length_eng_sentence']<=50]
lines=lines[lines['length_tam_sentence']<=50]
```

In [ ]:
```
all_eng_words=set()
for eng in english_sentences:
    for word in eng.split():
        if word not in all_eng_words:
            all_eng_words.add(word)

all_tamil_words=set()
for tam in tamil_sentences:
    for word in tam.split():
        if word not in all_tamil_words:
            all_tamil_words.add(word)
```

In [ ]:
```
max_length_src=max(lines['length_tam_sentence'])
max_length_tar=max(lines['length_eng_sentence'])
```

In [ ]:
```
input_words = sorted(list(all_tamil_words))
target_words = sorted(list(all_eng_words))
num_encoder_tokens = len(all_tamil_words)
num_decoder_tokens = len(all_eng_words)
num_encoder_tokens, num_decoder_tokens
```

Out[ ]:
```
(46735, 18682)
```

In [ ]:
```
num_decoder_tokens += 1 #for zero padding
```

In [ ]:
```
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)])
```

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

X_train, X_test, y_train, y_test = train_test_split(lines['tamil'], lines['english'], test_size=0.2, random_s
```

In [ ]:
```
X_train.to_pickle('X_train.pkl')
X_test.to_pickle('X_test.pkl')
```

In [ ]:
```
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 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)
```

Sequence Model

In [ ]:
```
latent_dim=300
```

In [ ]:
```
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 [ ]:
```
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)

model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
```

In [ ]:
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
```

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

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, None)	0	-
input_layer_1 (InputLayer)	(None, None)	0	-
embedding (Embedding)	(None, None, 300)	14,020,500	input_layer[0][0]
not_equal (NotEqual)	(None, None)	0	input_layer[0][0]
embedding_1 (Embedding)	(None, None, 300)	5,604,900	input_layer_1[0]…
lstm (LSTM)	[(None, 300), (None, 300), (None, 300)]	721,200	embedding[0][0], not_equal[0][0]
lstm_1 (LSTM)	[(None, None, 300), (None, 300), (None, 300)]	721,200	embedding_1[0][0… lstm[0][1], lstm[0][2]
dense (Dense)	(None, None, 18683)	5,623,583	lstm_1[0][0]

Total params: 26,691,383 (101.82 MB)

Trainable params: 26,691,383 (101.82 MB)

Non-trainable params: 0 (0.00 B)

In [ ]:
```
train_samples = len(X_train)
val_samples = len(X_test)
batch_size = 128
epochs = 100
```

In [ ]:
```
# model.fit(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)
```

In [ ]:
```
# model.save_weights('nmt_weights.h5')
```

In [ ]:
```
model.load_weights('/kaggle/input/tamiltoenglish/nmt_weights.h5')
```

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

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)

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)

decoder_model = Model(
    [decoder_inputs] + decoder_states_inputs,
    [decoder_outputs2] + decoder_states2)
```

In [ ]:
```
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_']
```

```
    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        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]
        decoded_sentence += ' '+sampled_char


        if (sampled_char == '_END' or
           len(decoded_sentence) > 50):
            stop_condition = True

        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        states_value = [h, c]

    return decoded_sentence
```

In [ ]:
```
train_gen = generate_batch(X_train, y_train, batch_size = 1)
k=-1
```

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 2s 2s/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 380ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
Input Tamil sentence: உலகம் முழுவதிலும்  போர் என்பது அதிகரித்த உண்மையான துன்பம் மற்றும் கஷ்டங்க
ளையுமே உருவாக்கும்
Actual English Translation:  throughout the world  the war will mean increased hardship and real suffering
Predicted English Translation:  the government is a political party of the wor
```

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

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 18ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step
Input Tamil sentence: தேவரீர் சமுத்திரத்தின் பெருமையை ஆளுகிறவர் அதின் அலைகள் எழும்பும்போது அவைக
ளை அடங்கப்பண்ணுகிறீர்
Actual English Translation:  you rule the raging of the sea when the waves thereof arise  you still them
Predicted English Translation:  and i have not a man to the lord that he shall
```

Transformer Model

In [ ]:
```
from transformers import pipeline

pipe = pipeline("translation", model="facebook/nllb-200-distilled-600M")
```

In [ ]:
```
test_sentences = ["நான் மிகவும் சந்தோஷமாக இருக்கிறேன்","அது அவசியமில்லை","தயவுசெய்து அதை மீண்டு
for i in test_sentences:
```

```
    print(i)
    print(pipe(i, src_lang='tamil', tgt_lang='english')[0]['translation_text'])
```

நான் மிகவும் சந்தோஷமாக இருக்கிறேன்

அது அவசியமில்லை
It's not necessary
தயவுசெய்து அதை மீண்டும் செய்யவும்
Please repeat it.
அது ஒரு நல்ல யோசனை

அவர்கள் ஒன்றாக வேலை செய்ய ஒப்புக்கொண்டனர்
They agreed to work together.