```
import collections
import numpy as np

from keras.layers import Input, Dense, Bidirectional, LSTM,Embedding
from keras.models import Model,Sequential
from keras.optimizers import Adam
from keras.losses import sparse_categorical_crossentropy
from keras.layers.embeddings import Embedding
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.layers import GRU, Input, Dense, TimeDistributed, Activation, RepeatVector, Bidire


# loading dataset
with open('small_vocab_en.txt','r') as f:
  eng_sentences=f.read().split('\n')
with open('small_vocab_fr.txt','r') as f:
  fr_sentences=f.read().split('\n')


#tokenizing i.e giving some word ids to each word in sentences.
def tokenize(x,encode_start_end=False):
  if encode_start_end:
        sentences = ["startofsentence " + s + "endofsentence" for s in sentences]
  token=Tokenizer(char_level=False)
  token.fit_on_texts(x)
  tokenized_sen=token.texts_to_sequences(x)
  return tokenized_sen,token


#padding
def pad(x,length=None):
  if length is None:
        length = max([len(sentence) for sentence in x])
  padded_x = pad_sequences(x, maxlen = length, padding = 'post', truncating = 'post')
  return padded_x


#preprocessing
def preprocess(x, y):
    preprocess_x, x_tk = tokenize(x)
    preprocess_y, y_tk = tokenize(y)
    preprocess_x = pad(preprocess_x)
    preprocess_y = pad(preprocess_y)
    #we need to reshape here because sparse_categorical_crossentropy works on 3 dim but y is
    preprocess_y = preprocess_y.reshape(*preprocess_y.shape, 1)
    return preprocess_x, preprocess_y, x_tk, y_tk



proc_eng_sentences,proc_fre_sentences,eng_tokenizer,fre_tokenizer=preprocess(eng_sentences,fr
```

```
#to check preprocessed sentences
max_english_sequence_length=proc_eng_sentences.shape[1]
max_french_sequence_length=proc_fre_sentences.shape[1]
eng_vocab_size=len(eng_tokenizer.word_index)+1
fre_vocab_size=len(fre_tokenizer.word_index)+1
print('Data Preprocessed')
print("Max English sentence length:", max_english_sequence_length)
print("Max French sentence length:", max_french_sequence_length)
print("English vocabulary size:", eng_vocab_size)
print("French vocabulary size:", fre_vocab_size)
```

```
Data Preprocessed
Max English sentence length: 15
Max French sentence length: 21
English vocabulary size: 200
French vocabulary size: 345
```

```
# training
encoder_input = Input(shape = (None, ),
                      name = "Encoder_Input")

embedding_dim = 200
embedded_input = Embedding(input_dim = eng_vocab_size,
                           output_dim = embedding_dim,
                           name = "Embedding_Layer")(encoder_input)

encoder_lstm = LSTM(units = 256,
                    activation = "relu",
                    return_sequences = False,
                    return_state = True,
                    name = "Encoder_LSTM")

_, last_h_encoder, last_c_encoder = encoder_lstm(embedded_input)

decoder_input = Input(shape = (None, 1),
                      name = "Deocder_Input")

decoder_lstm = LSTM(units = 256,
                    activation = "relu",
                    return_sequences = True,
                    return_state = True,
                    name = "Decoder_LSTM")
all_h_decoder, _, _ = decoder_lstm(decoder_input,
                                   initial_state = [last_h_encoder, last_c_encoder])

final_dense = Dense(fre_vocab_size,
                    activation = 'softmax',
                    name = "Final_Dense_Layer")

logits = final_dense(all_h_decoder)
```

```
logits = final_dense(all_h_decoder)
```

```
seq2seq_model = Model(inputs = [encoder_input, decoder_input],
                      outputs = logits)


seq2seq_model.compile(loss = sparse_categorical_crossentropy,
                      optimizer = Adam(lr = 0.002),
                      metrics = ['accuracy'])


decoder_fre_input = proc_fre_sentences.reshape((-1, max_french_sequence_length, 1))[:, :-1, :
decoder_fre_target = proc_fre_sentences.reshape((-1, max_french_sequence_length, 1))[:, 1:, :

seq2seq_model.fit([proc_eng_sentences, decoder_fre_input],
                  decoder_fre_target,
                  epochs = 5,
                  batch_size = 10)
```

```
Epoch 1/5
13787/13787 [==============================] - 1243s 90ms/step - loss: 0.4541 - accuracy
Epoch 2/5
13787/13787 [==============================] - 1235s 90ms/step - loss: 0.0462 - accuracy
Epoch 3/5
13787/13787 [==============================] - 1223s 89ms/step - loss: 0.0386 - accuracy
Epoch 4/5
13787/13787 [==============================] - 1248s 90ms/step - loss: 0.0312 - accuracy
Epoch 5/5
 2277/13787 [===>..........................] - ETA: 16:55 - loss: 0.0262 - accuracy: 0.9
```

```
accuracy: 0.8699
Epoch 2/5
13787/13787 [==============================] - 1235s 90ms/step - loss: 0.0462 - accuracy: 0.9
Epoch 3/5
13787/13787 [==============================] - 1223s 89ms/step - loss: 0.0386 - accuracy: 0.9
Epoch 4/5
13787/13787 [==============================] - 1248s 90ms/step - loss: 0.0312 - accuracy: 0.9
Buffered data was truncated after reaching the output size limit.
```