```
import numpy as np
import re
import shutil
import tensorflow as tf
from tensorflow.keras.layers import Embedding, GRU, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import pandas as pd
import os
import unicodedata
import urllib3
import zipfile
```

```
tf.__version__
```

```
'2.7.0'
```

```
num_samples = 33000
```

```
http = urllib3.PoolManager()
url ='http://www.manythings.org/anki/fra-eng.zip'
filename = 'fra-eng.zip'
path = os.getcwd()
zipfilename = os.path.join(path, filename)
with http.request('GET', url, preload_content=False) as r, open(zipfilename, 'wb') as out_file:
    shutil.copyfileobj(r, out_file)

with zipfile.ZipFile(zipfilename, 'r') as zip_ref:
    zip_ref.extractall(path)
```

```
def unicode_to_ascii(s):
  return ''.join(c for c in unicodedata.normalize('NFD', s)
      if unicodedata.category(c) != 'Mn')
```

```
def preprocess_sentence(sent):
    # 위에서 구현한 함수를 내부적으로 호출
    sent = unicode_to_ascii(sent.lower())

    # 단어와 구두점 사이에 공백을 만듭니다.
    # Ex) "he is a boy." => "he is a boy ."
    sent = re.sub(r"([?.!,¿])", r" W1", sent)

    # (a-z, A-Z, ".", "?", "!", ",") 이들을 제외하고는 전부 공백으로 변환합니다.
    sent = re.sub(r"[^a-zA-Z!.?]+", r" ", sent)

    sent = re.sub(r"Ws+", " ", sent)
    return sent
```

```
def load_preprocessed_data():
```

```
    encoder_input, decoder_input, decoder_target = [], [], []

    with open("fra.txt", "r") as lines:
        for i, line in enumerate(lines):

            # source 데이터와 target 데이터 분리
            src_line, tar_line, _ = line.strip().split('₩t')

            # source 데이터 전처리
            src_line = [w for w in preprocess_sentence(src_line).split()]

            # target 데이터 전처리
            tar_line = preprocess_sentence(tar_line)
            tar_line_in = [w for w in ("<sos> " + tar_line).split()]
            tar_line_out = [w for w in (tar_line + " <eos>").split()]

            encoder_input.append(src_line)
            decoder_input.append(tar_line_in)
            decoder_target.append(tar_line_out)

            if i == num_samples - 1:
                break

    return encoder_input, decoder_input, decoder_target
```

```
# 인코딩 테스트
en_sent = u"Have you had dinner?"
fr_sent = u"Avez-vous déjà diné?"
print(preprocess_sentence(en_sent))
print(preprocess_sentence(fr_sent).encode('utf-8'))
```

```
have you had dinner ?
b'avez vous deja dine ?'
```

```
sents_en_in, sents_fra_in, sents_fra_out = load_preprocessed_data()
```

```
print(sents_en_in[:5])
print(sents_fra_in[:5])
print(sents_fra_out[:5])
```

```
[['go', '.'], ['go', '.'], ['go', '.'], ['hi', '.'], ['hi', '.']]
[['<sos>', 'va', '!'], ['<sos>', 'marche', '.'], ['<sos>', 'bouge', '!'], ['<sos>', 'salut',
[['va', '!', '<eos>'], ['marche', '.', '<eos>'], ['bouge', '!', '<eos>'], ['salut', '!', '<ec
```

```
tokenizer_en = Tokenizer(filters="", lower=False)
tokenizer_en.fit_on_texts(sents_en_in)
encoder_input = tokenizer_en.texts_to_sequences(sents_en_in)
encoder_input = pad_sequences(encoder_input, padding="post")

tokenizer_fra = Tokenizer(filters="", lower=False)
tokenizer_fra.fit_on_texts(sents_fra_in)
tokenizer_fra.fit_on_texts(sents_fra_out)
```

```
decoder_input = tokenizer_fra.texts_to_sequences(sents_fra_in)
decoder_input = pad_sequences(decoder_input, padding="post")

decoder_target = tokenizer_fra.texts_to_sequences(sents_fra_out)
decoder_target = pad_sequences(decoder_target, padding="post")
```

```
print(encoder_input.shape)
print(decoder_input.shape)
print(decoder_target.shape)
```

```
(33000, 8)
(33000, 16)
(33000, 16)
```

```
vocab_size_en = len(tokenizer_en.word_index) + 1
vocab_size_fra = len(tokenizer_fra.word_index) + 1
print("영어 단어 집합의 크기 (en): {:d}, 프랑스어 단어 집합의 크기 (spa): {:d}".format(vocab_size_e
```

```
영어 단어 집합의 크기 (en): 4606, 프랑스어 단어 집합의 크기 (spa): 8107
```

```
src_to_index = tokenizer_en.word_index
index_to_src = tokenizer_en.index_word

tar_to_index = tokenizer_fra.word_index
index_to_tar = tokenizer_fra.index_word
```

```
max_src_len = encoder_input.shape[1]
max_tar_len = decoder_input.shape[1]
```

```
src_vocab_size = len(tokenizer_en.word_index) + 1
tar_vocab_size = len(tokenizer_fra.word_index) + 1
```

```
print(max_src_len)
print(max_tar_len)
```

```
8
16
```

```
indices = np.arange(encoder_input.shape[0])
np.random.shuffle(indices)
print(indices)
```

```
[17039 27366  3073 ... 19706 28995 26957]
```

```
encoder_input = encoder_input[indices]
decoder_input = decoder_input[indices]
decoder_target = decoder_target[indices]
```

```
encoder_input[30997]
```

```
array([ 37,   3, 217,   6,   4,   0,   0,   0], dtype=int32)
```

```
decoder_input[30997]
```

```
array([  2,  23,  62,   9, 232,   6,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0], dtype=int32)
```

```
decoder_target[30997]
```

```
array([ 23,  62,   9, 232,   6,   3,   0,   0,   0,   0,   0,   0,   0,
         0,   0,   0], dtype=int32)
```

```
n_of_val = int(33000*0.1)
print(n_of_val)
```

```
3300
```

```
encoder_input_train = encoder_input[:-n_of_val]
decoder_input_train = decoder_input[:-n_of_val]
decoder_target_train = decoder_target[:-n_of_val]

encoder_input_test = encoder_input[-n_of_val:]
decoder_input_test = decoder_input[-n_of_val:]
decoder_target_test = decoder_target[-n_of_val:]
```

```
print(encoder_input_train.shape)
print(decoder_input_train.shape)
print(decoder_target_train.shape)
print(encoder_input_test.shape)
print(decoder_input_test.shape)
print(decoder_target_test.shape)
```

```
(29700, 8)
(29700, 16)
(29700, 16)
(3300, 8)
(3300, 16)
(3300, 16)
```

```
latent_dim = 50
```

```
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Masking
from tensorflow.keras.models import Model
```

```
# 인코더
encoder_inputs = Input(shape=(None,))
enc_emb =  Embedding(src_vocab_size, latent_dim)(encoder_inputs) # 임베딩 층
enc_masking = Masking(mask_value=0.0)(enc_emb) # 패딩 0은 연산에서 제외
encoder_lstm = LSTM(latent_dim, return_state=True) # 상태값 리턴을 위해 return_state는 True
```

```
encoder_outputs, state_h, state_c = encoder_lstm(enc_masking) # 은닉 상태와 셀 상태를 리턴
encoder_states = [state_h, state_c] # 인코더의 은닉 상태와 셀 상태를 저장

# 디코더
decoder_inputs = Input(shape=(None,))
dec_emb_layer = Embedding(tar_vocab_size, latent_dim) # 임베딩 층
dec_emb = dec_emb_layer(decoder_inputs) # 패딩 0은 연산에서 제외
dec_masking = Masking(mask_value=0.0)(dec_emb)

# 상태값 리턴을 위해 return_state는 True, 모든 시점에 대해서 단어를 예측하기 위해 return_sequences는
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True)

# 인코더의 은닉 상태를 초기 은닉 상태(initial_state)로 사용
decoder_outputs, _, _ = decoder_lstm(dec_masking,
                                     initial_state=encoder_states)

# 모든 시점의 결과에 대해서 소프트맥스 함수를 사용한 출력층을 통해 단어 예측
decoder_dense = Dense(tar_vocab_size, activation='softmax')
decoder_outputs = decoder_dense(decoder_outputs)
```

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

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

```
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)     230300      ['input_1[0][0]']

 embedding_1 (Embedding)        (None, None, 50)     405350      ['input_2[0][0]']

 masking (Masking)              (None, None, 50)     0           ['embedding[0][0]']

 masking_1 (Masking)            (None, None, 50)     0           ['embedding_1[0][0]']

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

 lstm_1 (LSTM)                  [(None, None, 50),   20200       ['masking_1[0][0]',
                                 (None, 50),                      'lstm[0][1]',
                                 (None, 50)]                      'lstm[0][2]']

 dense (Dense)                  (None, None, 8107)   413457      ['lstm_1[0][0]']

==================================================================================================
Total params: 1,089,507
Trainable params: 1,089,507
```

```
Non-trainable params: 0
_________________________________________________________________
```

```
model.fit(x=[encoder_input_train, decoder_input_train], y=decoder_target_train, W
          validation_data = ([encoder_input_test, decoder_input_test], decoder_target_test),
          batch_size=128, epochs=50)
```

```
Epoch 1/50
233/233 [==============================] - 25s 55ms/step - loss: 3.1008 - acc: 0.6251 - val_l
Epoch 2/50
233/233 [==============================] - 9s 40ms/step - loss: 1.6461 - acc: 0.7433 - val_lo
Epoch 3/50
233/233 [==============================] - 9s 40ms/step - loss: 1.4753 - acc: 0.7605 - val_lo
Epoch 4/50
233/233 [==============================] - 9s 40ms/step - loss: 1.3843 - acc: 0.7739 - val_lo
Epoch 5/50
233/233 [==============================] - 9s 40ms/step - loss: 1.3048 - acc: 0.7898 - val_lo
Epoch 6/50
233/233 [==============================] - 9s 40ms/step - loss: 1.2412 - acc: 0.7991 - val_lo
Epoch 7/50
233/233 [==============================] - 9s 40ms/step - loss: 1.1907 - acc: 0.8066 - val_lo
Epoch 8/50
233/233 [==============================] - 9s 40ms/step - loss: 1.1498 - acc: 0.8127 - val_lo
Epoch 9/50
233/233 [==============================] - 9s 40ms/step - loss: 1.1143 - acc: 0.8189 - val_lo
Epoch 10/50
233/233 [==============================] - 9s 39ms/step - loss: 1.0827 - acc: 0.8241 - val_lo
Epoch 11/50
233/233 [==============================] - 9s 40ms/step - loss: 1.0552 - acc: 0.8275 - val_lo
Epoch 12/50
233/233 [==============================] - 9s 39ms/step - loss: 1.0300 - acc: 0.8306 - val_lo
Epoch 13/50
233/233 [==============================] - 9s 40ms/step - loss: 1.0065 - acc: 0.8337 - val_lo
Epoch 14/50
233/233 [==============================] - 9s 40ms/step - loss: 0.9845 - acc: 0.8366 - val_lo
Epoch 15/50
233/233 [==============================] - 9s 40ms/step - loss: 0.9643 - acc: 0.8393 - val_lo
Epoch 16/50
233/233 [==============================] - 9s 40ms/step - loss: 0.9456 - acc: 0.8420 - val_lo
Epoch 17/50
233/233 [==============================] - 9s 39ms/step - loss: 0.9283 - acc: 0.8442 - val_lo
Epoch 18/50
233/233 [==============================] - 9s 39ms/step - loss: 0.9120 - acc: 0.8464 - val_lo
Epoch 19/50
233/233 [==============================] - 9s 39ms/step - loss: 0.8962 - acc: 0.8483 - val_lo
Epoch 20/50
233/233 [==============================] - 9s 40ms/step - loss: 0.8802 - acc: 0.8503 - val_lo
Epoch 21/50
233/233 [==============================] - 9s 41ms/step - loss: 0.8647 - acc: 0.8519 - val_lo
Epoch 22/50
233/233 [==============================] - 9s 40ms/step - loss: 0.8520 - acc: 0.8539 - val_lo
Epoch 23/50
233/233 [==============================] - 9s 40ms/step - loss: 0.8403 - acc: 0.8554 - val_lo
Epoch 24/50
233/233 [==============================] - 10s 41ms/step - loss: 0.8300 - acc: 0.8571 - val_l
Epoch 25/50
233/233 [==============================] - 10s 41ms/step - loss: 0.8199 - acc: 0.8589 - val_l
Epoch 26/50
233/233 [==============================] - 9s 40ms/step - loss: 0.8106 - acc: 0.8601 - val_lo
```

```
Epoch 27/50
233/233 [==============================] - 9s 41ms/step - loss: 0.8018 - acc: 0.8614 - val_lo
Epoch 28/50
233/233 [==============================] - 9s 40ms/step - loss: 0.7935 - acc: 0.8630 - val_lo
Epoch 29/50
233/233 [==============================] - 9s 40ms/step - loss: 0.7858 - acc: 0.8640 - val_lo
```

```
# 인코더
encoder_model = Model(encoder_inputs, encoder_states)
```

```
# 디코더 설계 시작
# 이전 시점의 상태를 보관할 텐서
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]

# 훈련 때 사용했던 임베딩 층을 재사용
dec_emb2 = dec_emb_layer(decoder_inputs)

# 다음 단어 예측을 위해 이전 시점의 상태를 현 시점의 초기 상태로 사용
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=decoder_states_inputs)
decoder_states2 = [state_h2, state_c2]

# 모든 시점에 대해서 단어 예측
decoder_outputs2 = decoder_dense(decoder_outputs2)
```

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

```
def decode_sequence(input_seq):
    # 입력으로부터 인코더의 상태를 얻음
    states_value = encoder_model.predict(input_seq)

    # <SOS>에 해당하는 정수 생성
    target_seq = np.zeros((1,1))
    target_seq[0, 0] = tar_to_index['<sos>']

    stop_condition = False
    decoded_sentence = ''

    # stop_condition이 True가 될 때까지 루프 반복
    # 구현의 간소화를 위해서 이 함수는 배치 크기를 1로 가정합니다.
    while not stop_condition:
        # 이점 시점의 상태 states_value를 현 시점의 초기 상태로 사용
        output_tokens, h, c = decoder_model.predict([target_seq] + states_value)

        # 예측 결과를 단어로 변환
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_char = index_to_tar[sampled_token_index]

         # 현재 시점의 예측 단어를 예측 문장에 추가
```

```
        decoded_sentence += ' '+sampled_char

        # <eos>에 도달하거나 정해진 길이를 넘으면 중단.
        if (sampled_char == '<eos>' or
           len(decoded_sentence) > 50):
            stop_condition = True

        # 현재 시점의 예측 결과를 다음 시점의 입력으로 사용하기 위해 저장
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # 현재 시점의 상태를 다음 시점의 상태로 사용하기 위해 저장
        states_value = [h, c]

    return decoded_sentence
```

```
# 원문의 정수 시퀀스를 텍스트 시퀀스로 변환
def seq2src(input_seq):
    sentence = ''
    for i in input_seq:
        if(i!=0):
            sentence = sentence + index_to_src[i]+' '
    return sentence

# 번역문의 정수 시퀀스를 텍스트 시퀀스로 변환
def seq2tar(input_seq):
    sentence =''
    for i in input_seq:
        if((i!=0 and i!=tar_to_index['<sos>']) and i!=tar_to_index['<eos>']):
            sentence = sentence + index_to_tar[i] + ' '
    return sentence
```

```
#과제 부분

for seq_index in [68, 168, 3068, 1868, 2168, 468, 668, 868, 1068, 1568]:
  input_seq = encoder_input_train[seq_index: seq_index + 1]
  decoded_sentence = decode_sequence(input_seq)

  print("원문 : ",seq2src(encoder_input_train[seq_index]))
  print("번역문 :",seq2tar(decoder_input_train[seq_index]))
  print("예측문 :",decoded_sentence[1:-5])
  print("₩n")
```

```
원문 :  i ran downstairs .
번역문 : je me suis precipite en bas .
예측문 : je me suis a l air en train de nous .


원문 :  get to bed .
번역문 : va au lit !
예측문 : au lit !


원문 :  tom fell silent .
```

```
번역문 : tom s est tu .
예측문 : tom se sont l exterieur .


원문 :  they have come .
번역문 : ils sont venus .
예측문 : ils sont en train de nous .


원문 :  what a funny man !
번역문 : quel homme curieux !
예측문 : quel homme n est pas bon ?


원문 :  turn right .
번역문 : prenez a droite .
예측문 : a l interieur .


원문 :  i barely knew tom .
번역문 : je connaissais a peine tom .
예측문 : je vis a trouve tom .


원문 :  i may die tomorrow .
번역문 : je mourrai peut etre demain .
예측문 : je sais se yeux demain .


원문 :  i ve had enough .
번역문 : j en ai assez gobe .
예측문 : j ai assez en train de rire .


원문 :  i cry a lot .
번역문 : je pleure beaucoup .
예측문 : je deteste la tete .
```

```
#과제 부분

for seq_index in [68, 168, 3068, 1868, 2168, 468, 668, 868, 1068, 1568]:
  input_seq = encoder_input_test[seq_index: seq_index + 1]
  decoded_sentence = decode_sequence(input_seq)

  print("원문 : ",seq2src(encoder_input_test[seq_index]))
  print("번역문 :",seq2tar(decoder_input_test[seq_index]))
  print("예측문 :",decoded_sentence[1:-5])
  print("₩n")
```

```
원문 :  it isn t optional .
번역문 : cela n a rien de facultatif .
예측문 : ce n est pas de temps .


원문 :  i misunderstood .
번역문 : j ai compris de travers .
```

```
예측문 : j ai ete apprecie .


원문 :  tom s funny .
번역문 : tom est drole .
예측문 : tom est marrant .


원문 :  tom remembers you .
번역문 : tom se souvient de vous .
예측문 : tom vous a fait .


원문 :  she loves shopping .
번역문 : elle adore faire les courses .
예측문 : elle adore faire des courses .


원문 :  attack !
번역문 : a l attaque !
예측문 : c est pas du lit !


원문 :  are you a maniac ?
번역문 : etes vous maniaque ?
예측문 : es tu grand ?


원문 :  that s a nice coat .
번역문 : c est un beau manteau .
예측문 : c est une enfant .


원문 :  i tried it .
번역문 : je l ai essaye .
예측문 : j ai essaye de le maison .


원문 :  here s my ticket .
번역문 : voici mon ticket .
예측문 : mon pere est .
```

```
#훈련이 끝난 시스템을 재사용할 수 있도록 하는 방법
#모델을 디스크에 저장하는데 사용할 수 있는 두 형식은 Tensorflow SavedModel 형식과 이전 Keras H5 형식
#이 중에서도 Tensorflow SavedModel 형식을 권장한다.
#현재 이 모델의 경우 keras 모델이다.
#keras 모델을 재사용하기 위해서는 저장을 한다.
#저장을 하는 방법은 다음과 같다.

from keras.models import load_model
model.save('my_model.h5')

#이를 불러오려면 다음과 같이 한다.
model2 = load_model('my_model.h5')
```

```
#위에서 정의했던 model과 같음을 알 수 있다.
model2.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)          230300      ['input_1[0][0]']

embedding_1 (Embedding)      (None, None, 50)          405350      ['input_2[0][0]']

masking (Masking)            (None, None, 50)          0           ['embedding[0][0]']

masking_1 (Masking)          (None, None, 50)          0           ['embedding_1[0][0]']

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

lstm_1 (LSTM)                [(None, None, 50),        20200       ['masking_1[0][0]',
                              (None, 50),                         'lstm[0][1]',
                              (None, 50)]                         'lstm[0][2]']

dense (Dense)                (None, None, 8107)        413457      ['lstm_1[0][0]']

=================================================================================================
Total params: 1,089,507
Trainable params: 1,089,507
Non-trainable params: 0
_________________________________________________________________________________________________
```