Build a seq2seq model for machine translation.

Task: Change LSTM model to Bidirectional LSTM Model， translate English to target language and evaluate using Bleu score.

0. You will do the following:

1. Read and run the code. Please make sure you have installed keras or tensorflow.Running the script on colab will speed up the training process and also prevent package loading issue.
2. Complete the code in Section 1.1, you may fill in your data directory.
3. Directly modify the code in Section 3. Change the current LSTM layer to a Bidirectional LSTM Model.
4. Training your model and translate English to Spanish in Section 4.2. You could try translating other languages.
5. Complete the code in Section 5.

Hint:

To implement `Bi-LSTM` , you will need the following code to build the encoder. Do NOT use Bi-LSTM for the decoder. But there are other codes you need to modify to make it work.

In [1]:
```
# from keras.layers import Bidirectional, Concatenate

# encoder_bilstm = Bidirectional(LSTM(latent_dim, return_state=True,
#                                     dropout=0.5, name='encoder_lstm'))
# _, forward_h, forward_c, backward_h, backward_c = encoder_bilstm(encoder_inp

# state_h = Concatenate()([forward_h, backward_h])
# state_c = Concatenate()([forward_c, backward_c])
```

1. Data preparation (10 points)

1. Download spanish-english data from [http://www.manythings.org/anki/ (http://www.manythings.org/anki/)](http://www.manythings.org/anki/)
2. You may try to use other languages.
3. Unzip the .ZIP file.
4. Put the .TXT file (e.g., "deu.txt") in the directory "./Data/".
5. Fill in your data directory in section 1.1.

1.1. Load and clean text

In [2]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [3]:
```
import re
import string
from unicodedata import normalize
import numpy

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, mode='rt', encoding='utf-8')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text


# split a loaded document into sentences
def to_pairs(doc):
    lines = doc.strip().split('\n')
    pairs = [line.split('\t') for line in  lines]
    return pairs

def clean_data(lines):
    cleaned = list()
    # prepare regex for char filtering
    re_print = re.compile('[^%s]' % re.escape(string.printable))
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for pair in lines:
        clean_pair = list()
        for line in pair:
            # normalize unicode characters
            line = normalize('NFD', line).encode('ascii', 'ignore')
            line = line.decode('UTF-8')
            # tokenize on white space
            line = line.split()
            # convert to lowercase
            line = [word.lower() for word in line]
            # remove punctuation from each token
            line = [word.translate(table) for word in line]
            # remove non-printable chars form each token
            line = [re_print.sub('', w) for w in line]
            # remove tokens with numbers in them
            line = [word for word in line if word.isalpha()]
            # store as string
            clean_pair.append(' '.join(line))
        cleaned.append(clean_pair)
    return numpy.array(cleaned)
```

Fill the following blanks:

In [4]:
```
# e.g., filename = 'Data/deu.txt'
filename = '/content/drive/MyDrive/spa.txt'

# e.g., n_train = 30000
n_train = 30000
```

In [5]:
```
# load dataset
doc = load_doc(filename)

# split into Language1-Language2 pairs
pairs = to_pairs(doc)
print(len(pairs))

rand_indices = numpy.random.permutation(30000)

# clean sentences
clean_pairs = clean_data(pairs)[rand_indices, :]
```

```
139705
```

In [6]:
```
for i in range(3000, 3010):
    print('[' + clean_pairs[i, 0] + '] => [' + clean_pairs[i, 1] + ']')
```

```
[hold it] => [sostenganlo]
[tom began talking] => [tom empezo a hablar]
[are you forgetful] => [eres desmemoriado]
[leave tom] => [dejalo a tomas]
[itll be difficult] => [sera dificil]
[its a small world] => [el mundo es pequeno]
[tom grabbed it] => [tom lo agarro]
[can i keep it] => [puedo quedarmelo]
[what a good shot] => [que buen tiro]
[life aint easy] => [la vida no es facil]
```

In [7]:
```
input_texts = clean_pairs[:, 0]
target_texts = ['\t' + text + '\n' for text in clean_pairs[:, 1]]

print('Length of input_texts:  ' + str(input_texts.shape))
print('Length of target_texts: ' + str(input_texts.shape))
```

```
Length of input_texts:  (30000,)
Length of target_texts: (30000,)
```

In [8]:
```
max_encoder_seq_length = max(len(line) for line in input_texts)
max_decoder_seq_length = max(len(line) for line in target_texts)

print('max length of input  sentences: %d' % (max_encoder_seq_length))
print('max length of target sentences: %d' % (max_decoder_seq_length))
```

```
max length of input  sentences: 20
max length of target sentences: 68
```

Remark: To this end, you have two lists of sentences: input_texts and target_texts

2. Text processing

2.1. Convert texts to sequences

- Input: A list of n sentences (with max length t).
- It is represented by a $n \times t$ matrix after the tokenization and zero-padding.

In [9]:
```
from keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

# encode and pad sequences
def text2sequences(max_len, lines):
    tokenizer = Tokenizer(char_level=True, filters='')
    tokenizer.fit_on_texts(lines)
    seqs = tokenizer.texts_to_sequences(lines)
    seqs_pad = pad_sequences(seqs, maxlen=max_len, padding='post')
    return seqs_pad, tokenizer.word_index


encoder_input_seq, input_token_index = text2sequences(max_encoder_seq_length,
                                                      input_texts)
decoder_input_seq, target_token_index = text2sequences(max_decoder_seq_length,
                                                       target_texts)

print('shape of encoder_input_seq: ' + str(encoder_input_seq.shape))
print('shape of input_token_index: ' + str(len(input_token_index)))
print('shape of decoder_input_seq: ' + str(decoder_input_seq.shape))
print('shape of target_token_index: ' + str(len(target_token_index)))
```

```
shape of encoder_input_seq: (30000, 20)
shape of input_token_index: 27
shape of decoder_input_seq: (30000, 68)
shape of target_token_index: 29
```

In [10]:
```
num_encoder_tokens = len(input_token_index) + 1
num_decoder_tokens = len(target_token_index) + 1

print('num_encoder_tokens: ' + str(num_encoder_tokens))
print('num_decoder_tokens: ' + str(num_decoder_tokens))
```

```
num_encoder_tokens: 28
num_decoder_tokens: 30
```

Remark: To this end, the input language and target language texts are converted to 2 matrices.

- Their number of rows are both n_train.
- Their number of columns are respective max_encoder_seq_length and max_decoder_seq_length.

The followings print a sentence and its representation as a sequence.

In [11]:
```
target_texts[100]
```

Out[11]:
```
'\tamo a mis padres\n'
```

In [12]:
```
decoder_input_seq[100, :]
```

Out[12]:
```
array([ 6,  3, 13,  4,  1,  3,  1, 13, 11,  5,  1, 17,  3, 15, 10,  2,  5,
        7,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
        0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0],
      dtype=int32)
```

2.2. One-hot encode

- Input: A list of n sentences (with max length t).
- It is represented by a $n \times t$ matrix after the tokenization and zero-padding.
- It is represented by a $n \times t \times v$ tensor (t is the number of unique chars) after the one-hot encoding.

In [13]:
```
from tensorflow.keras.utils import to_categorical

# one hot encode target sequence
def onehot_encode(sequences, max_len, vocab_size):
    n = len(sequences)
    data = numpy.zeros((n, max_len, vocab_size))
    for i in range(n):
        data[i, :, :] = to_categorical(sequences[i], num_classes=vocab_size)
    return data

encoder_input_data = onehot_encode(encoder_input_seq, max_encoder_seq_length,
decoder_input_data = onehot_encode(decoder_input_seq, max_decoder_seq_length,

decoder_target_seq = numpy.zeros(decoder_input_seq.shape)
decoder_target_seq[:, 0:-1] = decoder_input_seq[:, 1:]
decoder_target_data = onehot_encode(decoder_target_seq,
                                    max_decoder_seq_length,
                                    num_decoder_tokens)

print(encoder_input_data.shape)
print(decoder_input_data.shape)
```

```
(30000, 20, 28)
(30000, 68, 30)
```

3. Build the networks (for training) (20 points)

- In this section, we have already implemented the LSTM model for you. You can run the code and see what the code is doing.

- You need to change the existing LSTM model to a Bidirectional LSTM model. Just modify the network structrue and do not change the training cell in section 3.4.
- Build encoder, decoder, and connect the two modules to get "model".
- Fit the model on the bilingual data to train the parameters in the encoder and decoder.

3.1. Encoder network

- Input: one-hot encode of the input language
- Return:

  -- output (all the hidden states $h_1, \cdots, h_t$) are always discarded

  -- the final hidden state h_t

  -- the final conveyor belt c_t

In [14]:
```
from tensorflow.keras.layers import Input, LSTM
from tensorflow.keras.models import Model

latent_dim = 256

# inputs of the encoder network
encoder_inputs = Input(shape=(None, num_encoder_tokens),
                       name='encoder_inputs')

# # set the LSTM layer
# encoder_lstm = LSTM(latent_dim, return_state=True,
#                     dropout=0.5, name='encoder_lstm')
# _, state_h, state_c = encoder_lstm(encoder_inputs)

# # build the encoder network model
# encoder_model = Model(inputs=encoder_inputs,
#                       outputs=[state_h, state_c],
#                       name='encoder')

from keras.layers import Bidirectional, Concatenate

encoder_bilstm = Bidirectional(LSTM(latent_dim, return_state=True,
                                    dropout=0.1, name='encoder_lstm'))
_, forward_h, forward_c, backward_h, backward_c = encoder_bilstm(encoder_input

state_h = Concatenate()([forward_h, backward_h])
state_c = Concatenate()([forward_c, backward_c])

# build the encoder network model
encoder_model = Model(inputs=encoder_inputs,
                      outputs=[state_h, state_c],
                      name='encoder')
```

Print a summary and save the encoder network structure to "./encoder.pdf"

In [15]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot, plot_model

SVG(model_to_dot(encoder_model, show_shapes=False).create(prog='dot', format='

plot_model(
    model=encoder_model, show_shapes=False,
    to_file='encoder.pdf'
)

encoder_model.summary()
```

```
Model: "encoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 encoder_inputs (InputLayer)    [(None, None, 28)]   0           []

 bidirectional (Bidirectional)  [(None, 512),        583680      ['encoder_inputs[0][0]']
                                 (None, 256),
                                 (None, 256),
                                 (None, 256),
                                 (None, 256)]

 concatenate (Concatenate)      (None, 512)          0           ['bidirectional[0][1]',
                                                                  'bidirectional[0][3]']

 concatenate_1 (Concatenate)    (None, 512)          0           ['bidirectional[0][2]',
                                                                  'bidirectional[0][4]']

==================================================================================================
Total params: 583,680
Trainable params: 583,680
Non-trainable params: 0
__________________________________________________________________________________________________
```

3.2. Decoder network

- Inputs:

  -- one-hot encode of the target language

  -- The initial hidden state h_t

  -- The initial conveyor belt c_t
- Return:

-- output (all the hidden states) $h_1, \cdots, h_t$

-- the final hidden state h_t (discarded in the training and used in the prediction)

-- the final conveyor belt c_t (discarded in the training and used in the prediction)

In [16]:
```
from keras.layers import Input, LSTM, Dense
from keras.models import Model

# inputs of the decoder network
decoder_input_h = Input(shape=(2*latent_dim,), name='decoder_input_h')
decoder_input_c = Input(shape=(2*latent_dim,), name='decoder_input_c')
decoder_input_x = Input(shape=(None, num_decoder_tokens), name='decoder_input_

# set the LSTM layer
decoder_lstm = LSTM(2*latent_dim, return_sequences=True,
                    return_state=True, dropout=0.1, name='decoder_lstm')
decoder_lstm_outputs, state_h, state_c = decoder_lstm(decoder_input_x,
                                                      initial_state=[decoder_i

# set the dense layer
decoder_dense = Dense(num_decoder_tokens, activation='softmax', name='decoder_
decoder_outputs = decoder_dense(decoder_lstm_outputs)

# build the decoder network model
decoder_model = Model(inputs=[decoder_input_x, decoder_input_h, decoder_input_
                      outputs=[decoder_outputs, state_h, state_c],
                      name='decoder')
```

Print a summary and save the encoder network structure to "./decoder.pdf"

In [17]:
```python
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot, plot_model

SVG(model_to_dot(decoder_model, show_shapes=False).create(prog='dot', format='

plot_model(
    model=decoder_model, show_shapes=False,
    to_file='decoder.pdf'
)

decoder_model.summary()
```

```
Model: "decoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 decoder_input_x (InputLayer)   [(None, None, 30)]   0           []

 decoder_input_h (InputLayer)   [(None, 512)]        0           []

 decoder_input_c (InputLayer)   [(None, 512)]        0           []

 decoder_lstm (LSTM)            [(None, None, 512),  1112064     ['decoder_input_x[0][0]',
                                 (None, 512),                     'decoder_input_h[0][0]',
                                 (None, 512)]                     'decoder_input_c[0][0]']

 decoder_dense (Dense)          (None, None, 30)     15390       ['decoder_lstm[0][0]']

==================================================================================================
Total params: 1,127,454
Trainable params: 1,127,454
Non-trainable params: 0
__________________________________________________________________________________________________
```

3.3. Connect the encoder and decoder

In [18]:
```
# input layers
encoder_input_x = Input(shape=(None, num_encoder_tokens), name='encoder_input_
decoder_input_x = Input(shape=(None, num_decoder_tokens), name='decoder_input_

# connect encoder to decoder
encoder_final_states = encoder_model([encoder_input_x])
decoder_lstm_output, _, _ = decoder_lstm(decoder_input_x, initial_state=encode
decoder_pred = decoder_dense(decoder_lstm_output)

model = Model(inputs=[encoder_input_x, decoder_input_x],
              outputs=decoder_pred,
              name='model_training')
```

In [19]:
```
print(state_h)
print(decoder_input_h)
```

```
KerasTensor(type_spec=TensorSpec(shape=(None, 512), dtype=tf.float32, name=No
ne), name='decoder_lstm/PartitionedCall:2', description="created by layer 'de
coder_lstm'")
KerasTensor(type_spec=TensorSpec(shape=(None, 512), dtype=tf.float32, name='d
ecoder_input_h'), name='decoder_input_h', description="created by layer 'deco
der_input_h'")
```

In [20]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot, plot_model

SVG(model_to_dot(model, show_shapes=False).create(prog='dot', format='svg'))

plot_model(
    model=model, show_shapes=False,
    to_file='model_training.pdf'
)

model.summary()
```

```
Model: "model_training"
_____________________________________________________________________________
_____________________
 Layer (type)                   Output Shape          Param #     Connected to
=============================================================================
=====================
 encoder_input_x (InputLayer)   [(None, None, 28)]    0           []

 decoder_input_x (InputLayer)   [(None, None, 30)]    0           []

 encoder (Functional)           [(None, 512),         583680      ['encoder_in
put_x[0][0]']
                                 (None, 512)]

 decoder_lstm (LSTM)            [(None, None, 512),   1112064     ['decoder_in
put_x[0][0]',
                                 (None, 512),                      'encoder[0]
[0]',
                                 (None, 512)]                      'encoder[0]
[1]']

 decoder_dense (Dense)          (None, None, 30)      15390       ['decoder_ls
tm[1][0]']

=============================================================================
=====================
Total params: 1,711,134
Trainable params: 1,711,134
Non-trainable params: 0
_____________________________________________________________________________
_____________________
```

3.4. Fit the model on the bilingual dataset

- encoder_input_data: one-hot encode of the input language
- decoder_input_data: one-hot encode of the input language
- decoder_target_data: labels (left shift of decoder_input_data)
- tune the hyper-parameters
- stop when the validation loss stop decreasing.

In [21]:
```
print('shape of encoder_input_data' + str(encoder_input_data.shape))
print('shape of decoder_input_data' + str(decoder_input_data.shape))
print('shape of decoder_target_data' + str(decoder_target_data.shape))
```

```
shape of encoder_input_data(30000, 20, 28)
shape of decoder_input_data(30000, 68, 30)
shape of decoder_target_data(30000, 68, 30)
```

In [22]:
```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.save_weights('model_pretrain.h5')
```

In [23]:
```
model.load_weights('model_pretrain.h5')
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder_input_data],  # training data
          decoder_target_data,                       # labels (left shift of t
          batch_size=64, epochs=50, validation_split=0.2)

model.save('seq2seq.h5')
```

```
Epoch 1/50
375/375 [==============================] - 19s 24ms/step - loss: 0.8377 - val
_loss: 0.6686
Epoch 2/50
375/375 [==============================] - 8s 21ms/step - loss: 0.6226 - val_
loss: 0.5755
Epoch 3/50
375/375 [==============================] - 8s 21ms/step - loss: 0.5755 - val_
loss: 0.5391
Epoch 4/50
375/375 [==============================] - 8s 21ms/step - loss: 0.5501 - val_
loss: 0.5185
Epoch 5/50
375/375 [==============================] - 8s 20ms/step - loss: 0.5263 - val_
loss: 0.4877
Epoch 6/50
375/375 [==============================] - 8s 21ms/step - loss: 0.5046 - val_
loss: 0.4668
Epoch 7/50
375/375 [==============================] - 8s 22ms/step - loss: 0.4854 - val_
loss: 0.4481
Epoch 8/50
375/375 [==============================] - 8s 21ms/step - loss: 0.4671 - val_
loss: 0.4321
Epoch 9/50
375/375 [==============================] - 8s 21ms/step - loss: 0.4513 - val_
loss: 0.4143
Epoch 10/50
375/375 [==============================] - 8s 21ms/step - loss: 0.4359 - val_
loss: 0.4064
Epoch 11/50
375/375 [==============================] - 8s 21ms/step - loss: 0.4221 - val_
loss: 0.3882
Epoch 12/50
375/375 [==============================] - 8s 21ms/step - loss: 0.4094 - val_
loss: 0.3748
Epoch 13/50
375/375 [==============================] - 8s 21ms/step - loss: 0.3976 - val_
loss: 0.3656
Epoch 14/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3861 - val_
loss: 0.3563
Epoch 15/50
375/375 [==============================] - 8s 21ms/step - loss: 0.3757 - val_
loss: 0.3472
Epoch 16/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3665 - val_
loss: 0.3420
Epoch 17/50
375/375 [==============================] - 8s 21ms/step - loss: 0.3571 - val_
loss: 0.3320
Epoch 18/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3486 - val_
loss: 0.3256
Epoch 19/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3404 - val_
loss: 0.3196
```

```
Epoch 20/50
375/375 [==============================] - 8s 21ms/step - loss: 0.3325 - val_
loss: 0.3133
Epoch 21/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3258 - val_
loss: 0.3061
Epoch 22/50
375/375 [==============================] - 8s 21ms/step - loss: 0.3184 - val_
loss: 0.3026
Epoch 23/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3109 - val_
loss: 0.2983
Epoch 24/50
375/375 [==============================] - 8s 22ms/step - loss: 0.3052 - val_
loss: 0.2937
Epoch 25/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2992 - val_
loss: 0.2898
Epoch 26/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2932 - val_
loss: 0.2860
Epoch 27/50
375/375 [==============================] - 8s 21ms/step - loss: 0.2872 - val_
loss: 0.2820
Epoch 28/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2821 - val_
loss: 0.2816
Epoch 29/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2773 - val_
loss: 0.2764
Epoch 30/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2727 - val_
loss: 0.2775
Epoch 31/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2676 - val_
loss: 0.2720
Epoch 32/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2628 - val_
loss: 0.2695
Epoch 33/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2583 - val_
loss: 0.2670
Epoch 34/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2539 - val_
loss: 0.2661
Epoch 35/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2496 - val_
loss: 0.2632
Epoch 36/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2458 - val_
loss: 0.2626
Epoch 37/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2410 - val_
loss: 0.2620
Epoch 38/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2383 - val_
loss: 0.2595
```

```
Epoch 39/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2339 - val_
loss: 0.2587
Epoch 40/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2301 - val_
loss: 0.2577
Epoch 41/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2274 - val_
loss: 0.2573
Epoch 42/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2231 - val_
loss: 0.2547
Epoch 43/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2203 - val_
loss: 0.2533
Epoch 44/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2157 - val_
loss: 0.2536
Epoch 45/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2128 - val_
loss: 0.2519
Epoch 46/50
375/375 [==============================] - 9s 23ms/step - loss: 0.2109 - val_
loss: 0.2523
Epoch 47/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2079 - val_
loss: 0.2506
Epoch 48/50
375/375 [==============================] - 8s 22ms/step - loss: 0.2051 - val_
loss: 0.2515
Epoch 49/50
375/375 [==============================] - 8s 23ms/step - loss: 0.2015 - val_
loss: 0.2495
Epoch 50/50
375/375 [==============================] - 8s 22ms/step - loss: 0.1982 - val_
loss: 0.2507
```

4. Make predictions

- In this section, you need to complete section 4.2 to translate English to the target language.

4.1. Translate English to XXX

1. Encoder read a sentence (source language) and output its final states, h_t and c_t.
2. Take the [star] sign "\t" and the final state h_t and c_t as input and run the decoder.
3. Get the new states and predicted probability distribution.
4. sample a char from the predicted probability distribution
5. take the sampled char and the new states as input and repeat the process (stop if reach the [stop] sign "\n").

In [24]:
```
# Reverse-lookup token index to decode sequences back to something readable.
reverse_input_char_index = dict((i, char) for char, i in input_token_index.ite
reverse_target_char_index = dict((i, char) for char, i in target_token_index.i
```

In [25]:
```
def decode_sequence(input_seq):
    states_value = encoder_model.predict(input_seq)

    target_seq = numpy.zeros((1, 1, num_decoder_tokens))
    target_seq[0, 0, target_token_index['\t']] = 1.

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:
        output_tokens, h, c = decoder_model.predict([target_seq] + states_valu

        # this line of code is greedy selection
        # try to use multinomial sampling instead (with temperature)
        sampled_token_index = numpy.argmax(output_tokens[0, -1, :])

        sampled_char = reverse_target_char_index[sampled_token_index]
        decoded_sentence += sampled_char

        if (sampled_char == '\n' or
           len(decoded_sentence) > max_decoder_seq_length):
            stop_condition = True

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

        states_value = [h, c]

    return decoded_sentence
```

In [26]:
```
for seq_index in range(2100, 2120):
    # Take one sequence (part of the training set)
    # for trying out decoding.
    input_seq = encoder_input_data[seq_index: seq_index + 1]
    decoded_sentence = decode_sequence(input_seq)
    print('-')
    print('English:       ', input_texts[seq_index])
    print('Spanish (true): ', target_texts[seq_index][1:-1])
    print('Spanish (pred): ', decoded_sentence[0:-1])
```

```
1/1 [==============================] - 1s 642ms/step
1/1 [==============================] - 0s 359ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
-
English:        be still
Spanish (true):  no te muevas
Spanish (pred):  se pustale
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
```

4.2. Translate an English sentence to the target language (20 points)

1. Tokenization
2. One-hot encode
3. Translate

In [27]:
```
input_sentence = ['I love you']

input_sequence, l = text2sequences(max_encoder_seq_length,input_sentence)

input_x = onehot_encode(input_sequence, max_encoder_seq_length, num_encoder_to

translated_sentence = decode_sequence(input_x)

print('source sentence is: ' + str(input_sentence))
print('translated sentence is: ' + str(translated_sentence))
```

```
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 52ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 32ms/step
source sentence is: ['I love you']
translated sentence is: el condudo por favor
```

5. Evaluate the translation using BLEU score

- We have already translated from English to target language, but how can we evaluate the performance of our model quantitatively?
- In this section, you need to re-train the model we built in secton 3 and then evaluate the bleu score on testing dataset.

Reference:

https://machinelearningmastery.com/calculate-bleu-score-for-text-python/ (https://machinelearningmastery.com/calculate-bleu-score-for-text-python/)

https://en.wikipedia.org/wiki/BLEU (https://en.wikipedia.org/wiki/BLEU)

Hint:

- Randomly partition the dataset to training, validation, and test.
- Evaluate the BLEU score using the test set. Report the average.
- You may use packages to calculate bleu score, e.g., sentence_bleu() from nltk package.

5.1. Partition the dataset to training, validation, and test. Build new token index. (10 points)

1. You may try to load more data/lines from text file.
2. Convert text to sequences and build token index using training data.
3. One-hot encode your training and validation text sequences.

In [28]:
```
filename = '/content/drive/MyDrive/spa.txt'

# e.g., n_train = 30000
n_train = 50000

# load dataset
doc = load_doc(filename)

# split into Language1-Language2 pairs
pairs = to_pairs(doc)

# clean sentences
clean_pairs = clean_data(pairs)[0:n_train, :]
input_texts = clean_pairs[:,0]
target_texts = ['\t' + text + '\n' for text in clean_pairs[:, 1]]
```

In [29]:
```
rand_indices = numpy.random.permutation(50000)
train_indices = rand_indices[0:int(50000*.98)]
test_indices = rand_indices[int(50000*.98):int(50000)]

input_train = input_texts[train_indices]
# input_valid = input_texts[valid_indices]
input_test = input_texts[test_indices]

target_train = numpy.asarray(target_texts)[train_indices]
# target_valid = numpy.asarray(target_texts)[valid_indices]
target_test = numpy.asarray(target_texts)[test_indices]
```

In [30]:
```
encoder_input_seq, input_token_index = text2sequences(max_encoder_seq_length,
                                                       input_train)
decoder_input_seq, target_token_index = text2sequences(max_decoder_seq_length,
                                                        target_train)



print('shape of encoder_input_seq: ' + str(encoder_input_seq.shape))
print('shape of input_token_index: ' + str(len(input_token_index)))
print('shape of decoder_input_seq: ' + str(decoder_input_seq.shape))
print('shape of target_token_index: ' + str(len(target_token_index)))
```

```
shape of encoder_input_seq: (49000, 20)
shape of input_token_index: 27
shape of decoder_input_seq: (49000, 68)
shape of target_token_index: 29
```

In [31]:
```
encoder_input_data = onehot_encode(encoder_input_seq, max_encoder_seq_length,
decoder_input_data = onehot_encode(decoder_input_seq, max_decoder_seq_length,

decoder_target_seq = numpy.zeros(decoder_input_seq.shape)
decoder_target_seq[:, 0:-1] = decoder_input_seq[:, 1:]
decoder_target_data = onehot_encode(decoder_target_seq,
                                    max_decoder_seq_length,
                                    num_decoder_tokens)

print(encoder_input_data.shape)
print(decoder_input_data.shape)
```

```
(49000, 20, 28)
(49000, 68, 30)
```

5.2 Retrain your previous Bidirectional LSTM model with training and validation data and tune the parameters (learning rate, optimizer, etc) based on validation score. (25 points)

1. Use the model structure in section 3 to train a new model with new training and validation datasets.
2. Based on validation BLEU score or loss to tune parameters.

In [32]:
```
model.load_weights("model_pretrain.h5")
model.compile(optimizer='adam', loss='categorical_crossentropy')
model.fit([encoder_input_data, decoder_input_data],  # training data
          decoder_target_data,                       # labels (left shift of t
          batch_size=100, epochs=50)

model.save('seq2seq_split.h5')
```

```
Epoch 1/50
490/490 [==============================] - 19s 27ms/step - loss: 0.8375
Epoch 2/50
490/490 [==============================] - 13s 27ms/step - loss: 0.6219
Epoch 3/50
490/490 [==============================] - 14s 28ms/step - loss: 0.5642
Epoch 4/50
490/490 [==============================] - 14s 28ms/step - loss: 0.5159
Epoch 5/50
490/490 [==============================] - 14s 28ms/step - loss: 0.4742
Epoch 6/50
490/490 [==============================] - 14s 28ms/step - loss: 0.4372
Epoch 7/50
490/490 [==============================] - 14s 28ms/step - loss: 0.4053
Epoch 8/50
490/490 [==============================] - 14s 28ms/step - loss: 0.3781
Epoch 9/50
490/490 [==============================] - 14s 28ms/step - loss: 0.3536
Epoch 10/50
490/490 [==============================] - 14s 28ms/step - loss: 0.3331
Epoch 11/50
490/490 [==============================] - 14s 28ms/step - loss: 0.3153
Epoch 12/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2992
Epoch 13/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2858
Epoch 14/50
490/490 [==============================] - 14s 29ms/step - loss: 0.2730
Epoch 15/50
490/490 [==============================] - 14s 29ms/step - loss: 0.2623
Epoch 16/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2523
Epoch 17/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2434
Epoch 18/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2351
Epoch 19/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2276
Epoch 20/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2208
Epoch 21/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2153
Epoch 22/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2086
Epoch 23/50
490/490 [==============================] - 14s 28ms/step - loss: 0.2036
Epoch 24/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1984
Epoch 25/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1938
Epoch 26/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1898
Epoch 27/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1853
Epoch 28/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1815
Epoch 29/50
```

```
490/490 [==============================] - 14s 29ms/step - loss: 0.1776
Epoch 30/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1748
Epoch 31/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1716
Epoch 32/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1682
Epoch 33/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1650
Epoch 34/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1633
Epoch 35/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1610
Epoch 36/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1574
Epoch 37/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1556
Epoch 38/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1532
Epoch 39/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1512
Epoch 40/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1490
Epoch 41/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1470
Epoch 42/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1451
Epoch 43/50
490/490 [==============================] - 14s 29ms/step - loss: 0.1432
Epoch 44/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1413
Epoch 45/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1400
Epoch 46/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1381
Epoch 47/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1363
Epoch 48/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1356
Epoch 49/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1347
Epoch 50/50
490/490 [==============================] - 14s 28ms/step - loss: 0.1325
```

5.3 Evaluate the BLEU score using the test set. (15 points)

1. Use trained model above to calculate the BLEU score with testing dataset.

In [33]:
```
from nltk.translate.bleu_score import sentence_bleu
from nltk.translate.bleu_score import SmoothingFunction
smoothIt = SmoothingFunction().method2
bleu_list = []
for n in range(len(input_test)):

  test_string = input_test[n]
  target = target_test[n]
  input = [test_string]

  encoder_input_seq,l= text2sequences(max_encoder_seq_length,input)
  input_x = onehot_encode(encoder_input_seq, max_encoder_seq_length, num_encod

  #it was showing some keyErrors because we used a smoothing function

  try:
    translated = decode_sequence(input_x)
  except KeyError as err:
    continue
  score = sentence_bleu(target, translated, smoothing_function = smoothIt)

  print(score)
  bleu_list.append(score)
```

```
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 32ms/step
```

In [34]:
```
print('BLEU score is:',numpy.mean(bleu_list))
```

```
BLEU score is: 0.28469477598521825
```