In this project, we use the data provided by http://www.manythings.org/anki/ (http://www.manythings.org/anki/) , train a Bi-LSTM Seq2Seq model to translate the English into Target Language.

Enviroment Set up

In [1]:
```
# only use for colab
import tensorflow as tf
tf.test.gpu_device_name()
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU'
)))

# read file from cloud file
import os
from google.colab import drive
drive.mount('/content/drive')

path = '/content/drive/MyDrive/Colab Notebooks/CS583/spa-eng'

os.chdir(path)
os.listdir(path)
```

```
Num GPUs Available:  1
Drive already mounted at /content/drive; to attempt to forcibly remoun
t, call drive.mount("/content/drive", force_remount=True).
```

Out[1]:
```
['spa.txt',
 '_about.txt',
 'encoder.pdf',
 'decoder.pdf',
 'model_training.pdf',
 'seq2seq.h5',
 'seq2seq1.h5']
```

Bi-LSTM Seq2Seq model, translate English to other languages

Data preparation

In this dataset we have english to 80 target languages, and in this demo we only use spainish as an example. And the data looks like English + TAB + The Other Language + TAB + Attribution.

This work isn't easy.	この仕事は簡単じゃない。	CC-BY 2.0 (France) Attribution: tatoeba.org #3737550 (CK) & #7977622 (Ninja)
Those are sunflowers.	それはひまわりです。	CC-BY 2.0 (France) Attribution: tatoeba.org #441940 (CK) & #205407 (arnab)
Tom bought a new car.	トムは新車を買った。	CC-BY 2.0 (France) Attribution: tatoeba.org #1026984 (CK) & #2733633 (tommy_san)
This watch is broken.	この時計は壊れている。	CC-BY 2.0 (France) Attribution: tatoeba.org #58929 (CK) & #221604 (bunbuku)

The attribution gets imported into Anki as a tag, by default This attribution contains the domain name of the source material, the sentences' ID numbers, and the sentence owners' usernames. You can basically ignore the attribution field if you are using this material for personal use and not redistributing these files. However, it's needed here to comply with the CC-BY license. Let's take Spanish as an example, the pre-view shown in the figure below.

	Go.	Ve.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986655 (cueyayotl)
0	Go.	Vete.	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
1	Go.	Vaya.	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
2	Go.	Váyase.	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
3	Hi.	Hola.	CC-BY 2.0 (France) Attribution: tatoeba.org #5...
4	Run!	¡Corre!	CC-BY 2.0 (France) Attribution: tatoeba.org #9...

The description of the data is shown in the figure below.

	Go.	Ve.	CC-BY 2.0 (France) Attribution: tatoeba.org #2877272 (CM) & #4986655 (cueyayotl)
count	138436	138436	138436
unique	117884	130468	138436
top	You can put it there.	¡Órale!	CC-BY 2.0 (France) Attribution: tatoeba.org #2...
freq	68	10	1

There are 138436 data contained in the Spanish dataset. We can split it into 3 parts (train, validation, and test). And the first stage of the model output is shown in the figure below. The parameters need to tune in the further work.

Load and clean text

In [2]:
```
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)
```

In [3]:
```
filename = 'spa.txt'
n_train = 12000
# 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, :]
```

In [4]:
```
for i in range(3000, 3010):
    print('[' + clean_pairs[i, 0] + '] => [' + clean_pairs[i, 1] + ']')
```

```
[youre here] => [estas aqui]
[youre here] => [estais aqui]
[youre late] => [estas retrasado]
[youre lost] => [estas perdido]
[youre mean] => [eres mala]
[youre mean] => [eres mezquino]
[youre mine] => [tu eres mio]
[youre nice] => [eres simpatico]
[youre nuts] => [estas loco]
[youre nuts] => [estas chiflado]
```

In [5]:
```
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:  (12000,)
Length of target_texts: (12000,)
```

In [6]:
```
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: 16
max length of target sentences: 41
```

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

Text processing

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 [7]:
```
from keras.preprocessing.text import Tokenizer
from 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_le
ngth,
                                                      input_texts)
decoder_input_seq, target_token_index = text2sequences(max_decoder_seq_l
ength,
                                                       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: (12000, 16)
shape of input_token_index: 27
shape of decoder_input_seq: (12000, 41)
shape of target_token_index: 29
```

In [8]:
```
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 [9]:
```
target_texts[100]
```

Out[9]:
```
'\tno puede ser\n'
```

In [10]:
```
decoder_input_seq[100, :]
```

Out[10]:
```
array([ 6,  9,  3,  1, 17, 14,  2, 15,  2,  1,  5,  2, 10,  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], dtype=int32)
```

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 [11]:
```
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_s
ize)
    return data

encoder_input_data = onehot_encode(encoder_input_seq, max_encoder_seq_le
ngth, num_encoder_tokens)
decoder_input_data = onehot_encode(decoder_input_seq, max_decoder_seq_le
ngth, num_decoder_tokens)

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)
```

```
(12000, 16, 28)
(12000, 41, 30)
```

Build the networks (for training)

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 [12]:
```
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')
```

In [13]:

```
from tensorflow.keras.layers import Input, LSTM
from tensorflow.keras.models import Model
from keras.layers import Bidirectional, Concatenate, LSTM

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)

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
_inputs)

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 [14]:
```python
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', fo
rmat='svg'))

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

encoder_model.summary()
```

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

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

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

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

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

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 [15]:

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

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

# set the LSTM layer
decoder_lstm = LSTM(latent_dim * 2, return_sequences=True,
                    return_state=True, dropout=0.5, name='decoder_lstm')
decoder_lstm_outputs, state_h, state_c = decoder_lstm(decoder_input_x,
                                                      initial_state=[dec
oder_input_h, decoder_input_c])

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

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

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

In [16]:
```
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', fo
rmat='svg'))

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

decoder_model.summary()
```

```
Model: "decoder"
______________________________________________________________________
___________________________
 Layer (type)                   Output Shape          Param #     Connec
ted 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     ['deco
der_input_x[0][0]',
                                 (None, 512),                      'deco
der_input_h[0][0]',
                                 (None, 512)]                      'deco
der_input_c[0][0]']

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

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

Connect the encoder and decoder

In [17]:

```
# input layers
encoder_input_x = Input(shape=(None, num_encoder_tokens), name='encoder_
input_x')
decoder_input_x = Input(shape=(None, num_decoder_tokens), name='decoder_
input_x')

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

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

In [18]:
```
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='sv
g'))

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

model.summary()
```

```
Model: "model_training"
__________________________________________________________________________
___________________________
 Layer (type)                   Output Shape          Param #     Connec
ted to
==========================================================================
===========================
 encoder_input_x (InputLayer)   [(None, None, 28)]    0           []

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

 encoder (Functional)           [(None, 512),         583680      ['enco
der_input_x[0][0]']
                                 (None, 512)]

 decoder_lstm (LSTM)            [(None, None, 512),   1112064     ['deco
der_input_x[0][0]',
                                 (None, 512),                      'enco
der[0][0]',
                                 (None, 512)]                      'enco
der[0][1]']

 decoder_dense (Dense)          (None, None, 30)      15390       ['deco
der_lstm[1][0]']

==========================================================================
===========================
Total params: 1,711,134
Trainable params: 1,711,134
Non-trainable params: 0
__________________________________________________________________________
___________________________
```

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 [19]:
```
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(12000, 16, 28)
shape of decoder_input_data(12000, 41, 30)
shape of decoder_target_data(12000, 41, 30)
```

In [20]:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy')

model.fit([encoder_input_data, decoder_input_data],  # training data
          decoder_target_data,                       # labels (left shif
t of the target sequences)
          batch_size=64, epochs=66, validation_split=0.2)

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

```
Epoch 1/66
150/150 [==============================] - 13s 45ms/step - loss: 1.2321
- val_loss: 1.1166
Epoch 2/66
150/150 [==============================] - 3s 23ms/step - loss: 0.9307
- val_loss: 0.9469
Epoch 3/66
150/150 [==============================] - 3s 19ms/step - loss: 0.8710
- val_loss: 0.8756
Epoch 4/66
150/150 [==============================] - 3s 19ms/step - loss: 0.8375
- val_loss: 0.8348
Epoch 5/66
150/150 [==============================] - 3s 19ms/step - loss: 0.8104
- val_loss: 0.8045
Epoch 6/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7871
- val_loss: 0.7846
Epoch 7/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7678
- val_loss: 0.7541
Epoch 8/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7504
- val_loss: 0.7386
Epoch 9/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7346
- val_loss: 0.7193
Epoch 10/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7212
- val_loss: 0.7087
Epoch 11/66
150/150 [==============================] - 3s 19ms/step - loss: 0.7104
- val_loss: 0.6941
Epoch 12/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6986
- val_loss: 0.6828
Epoch 13/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6855
- val_loss: 0.6717
Epoch 14/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6755
- val_loss: 0.6655
Epoch 15/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6648
- val_loss: 0.6559
Epoch 16/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6545
- val_loss: 0.6526
Epoch 17/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6472
- val_loss: 0.6458
Epoch 18/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6371
- val_loss: 0.6465
Epoch 19/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6283
- val_loss: 0.6305
```

```
Epoch 20/66
150/150 [==============================] - 3s 22ms/step - loss: 0.6208
- val_loss: 0.6275
Epoch 21/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6144
- val_loss: 0.6250
Epoch 22/66
150/150 [==============================] - 3s 19ms/step - loss: 0.6050
- val_loss: 0.6171
Epoch 23/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5998
- val_loss: 0.6194
Epoch 24/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5948
- val_loss: 0.6121
Epoch 25/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5889
- val_loss: 0.6171
Epoch 26/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5817
- val_loss: 0.6077
Epoch 27/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5725
- val_loss: 0.6048
Epoch 28/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5653
- val_loss: 0.6030
Epoch 29/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5635
- val_loss: 0.5999
Epoch 30/66
150/150 [==============================] - 3s 21ms/step - loss: 0.5576
- val_loss: 0.5961
Epoch 31/66
150/150 [==============================] - 3s 20ms/step - loss: 0.5503
- val_loss: 0.5961
Epoch 32/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5492
- val_loss: 0.5926
Epoch 33/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5411
- val_loss: 0.5897
Epoch 34/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5385
- val_loss: 0.5886
Epoch 35/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5334
- val_loss: 0.5975
Epoch 36/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5277
- val_loss: 0.5888
Epoch 37/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5221
- val_loss: 0.5907
Epoch 38/66
150/150 [==============================] - 3s 18ms/step - loss: 0.5201
- val_loss: 0.5866
```

```
Epoch 39/66
150/150 [==============================] - 3s 18ms/step - loss: 0.5127
- val_loss: 0.5893
Epoch 40/66
150/150 [==============================] - 3s 19ms/step - loss: 0.5114
- val_loss: 0.5961
Epoch 41/66
150/150 [==============================] - 3s 18ms/step - loss: 0.5072
- val_loss: 0.5885
Epoch 42/66
150/150 [==============================] - 3s 18ms/step - loss: 0.5040
- val_loss: 0.5878
Epoch 43/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4972
- val_loss: 0.5792
Epoch 44/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4939
- val_loss: 0.5860
Epoch 45/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4905
- val_loss: 0.5916
Epoch 46/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4883
- val_loss: 0.5863
Epoch 47/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4806
- val_loss: 0.5868
Epoch 48/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4795
- val_loss: 0.5869
Epoch 49/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4742
- val_loss: 0.5833
Epoch 50/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4695
- val_loss: 0.5872
Epoch 51/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4704
- val_loss: 0.5875
Epoch 52/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4678
- val_loss: 0.5817
Epoch 53/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4587
- val_loss: 0.5881
Epoch 54/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4560
- val_loss: 0.5949
Epoch 55/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4592
- val_loss: 0.5903
Epoch 56/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4510
- val_loss: 0.5826
Epoch 57/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4511
- val_loss: 0.5947
```

```
Epoch 58/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4486
- val_loss: 0.5922
Epoch 59/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4472
- val_loss: 0.5915
Epoch 60/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4421
- val_loss: 0.5876
Epoch 61/66
150/150 [==============================] - 3s 19ms/step - loss: 0.4387
- val_loss: 0.5937
Epoch 62/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4383
- val_loss: 0.5912
Epoch 63/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4319
- val_loss: 0.5917
Epoch 64/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4314
- val_loss: 0.5930
Epoch 65/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4295
- val_loss: 0.5897
Epoch 66/66
150/150 [==============================] - 3s 18ms/step - loss: 0.4260
- val_loss: 0.5946
```

Make predictions

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 [21]:
```
# Reverse-lookup token index to decode sequences back to something reada
ble.
reverse_input_char_index = dict((i, char) for char, i in input_token_ind
ex.items())
reverse_target_char_index = dict((i, char) for char, i in target_token_i
ndex.items())
```

In [22]:

```
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] + state
s_value)

        # 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 [23]:

```
input_sentences = clean_data(pairs)[n_train:n_train + 20][:, 0]
target_sentences = clean_data(pairs)[n_train:n_train + 20][:, 1]
for i, input_sentence in enumerate(input_sentences):
    input_sentence_length = len(input_sentence)
    input_sequence, input_token_index = text2sequences(input_sentence_le
ngth, [input_sentence])
    input_x = onehot_encode([input_sequence], input_sentence_length, num
_encoder_tokens)
    translated_sentence = decode_sequence(input_x)
    print('source sentence is: ' + input_sentence)
    print('target sentence is:' + target_sentences[i])
    print('translated sentence is: ' + translated_sentence)
```

source sentence is: check your inbox
target sentence is:comprueba tu buzon
translated sentence is: tom se acubre

source sentence is: check your order
target sentence is:verifique su orden
translated sentence is: come un poco

source sentence is: cherries are red
target sentence is:las cerezas son rojas
translated sentence is: el combio es corte

source sentence is: choose carefully
target sentence is:elige sabiamente
translated sentence is: el vio a mi mina

source sentence is: choose carefully
target sentence is:elige con cuidado
translated sentence is: el vio a mi mina

source sentence is: choose carefully
target sentence is:elige con prudencia
translated sentence is: el vio a mi mina

source sentence is: clean the mirror
target sentence is:limpia el espejo
translated sentence is: ella confia en ti

source sentence is: clean your hands
target sentence is:lavate las manos
translated sentence is: acaso es mucho

source sentence is: close the blinds
target sentence is:cierra las persianas
translated sentence is: me despide comido

source sentence is: close the drawer
target sentence is:cierra el cajon
translated sentence is: esas me adianas

source sentence is: close the window
target sentence is:cierra la ventana
translated sentence is: come un poco

source sentence is: close the window
target sentence is:cerra la ventana
translated sentence is: come un poco

source sentence is: close your books
target sentence is:cierre sus libros
translated sentence is: el se pio en la caraza

source sentence is: close your books
target sentence is:cierren sus libros
translated sentence is: el se pio en la caraza

source sentence is: close your mouth

```
target sentence is:cierra la boca
translated sentence is: el por alexiro

source sentence is: close your mouth
target sentence is:cerra la boca
translated sentence is: el por alexiro

source sentence is: come and help me
target sentence is:ven a ayudarme
translated sentence is: come un poco de para

source sentence is: come and help me
target sentence is:ven a echarme una mano
translated sentence is: come un poco de para

source sentence is: come and help me
target sentence is:ven y ayudame
translated sentence is: come un poco de para

source sentence is: come and help us
target sentence is:venga a ayudarnos por favor
translated sentence is: ellas estan bien
```

Because of computing power, this model only uses about 20% of the data. And the parameters need to tune in the further work. The model predictions are not very good. This model may be used in the final project.

Translate an English sentence to the target language

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

In [24]:

```
input_sentence = 'I love you'
input_sentence_length = len(input_sentence)

input_sequence, input_token_index = text2sequences(input_sentence_length
, [input_sentence])

input_x = onehot_encode([input_sequence], input_sentence_length, num_enc
oder_tokens)

translated_sentence = decode_sequence(input_x)

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

```
source sentence is: I love you
translated sentence is: deja de misar
```

Evaluate the BLEU score using the test set.

In [25]:
```
import random
from nltk.translate.bleu_score import sentence_bleu, corpus_bleu
```

In [36]:
```
references = []
candidates = []
input_sentences = clean_data(pairs)[n_train + 125:n_train + 477][:, 0]
target_sentences = clean_data(pairs)[n_train + 125:n_train + 477][:, 1]
score = 0
for i, input_sentence in enumerate(input_sentences):
    input_sentence_length = len(input_sentence)
    input_sequence, input_token_index = text2sequences(input_sentence_le
ngth, [input_sentence])
    input_x = onehot_encode([input_sequence], input_sentence_length, num
_encoder_tokens)
    translated_sentence = decode_sequence(input_x)
    candidates = translated_sentence.split()
    references = [target_sentences[i].split()]
    score_temp = sentence_bleu(references, candidates)
    score += score_temp
print("Bleu Score:", score/352)
```

```
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:49
0: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:49
0: UserWarning:
Corpus/Sentence contains 0 counts of 3-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)

Bleu Score: 0.14822081469853618
```