In [1]:

```
from __future__ import absolute_import, division, print_function, unicode_literals

import tensorflow as tf

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
from sklearn.model_selection import train_test_split

import unicodedata
import re
import numpy as np
import os
import io
import time
```

In [2]:

```
path_to_zip = tf.keras.utils.get_file(
    'spa-eng.zip',
    origin='http://storage.googleapis.com/download.tensorflow.org/data/spa-eng.zip',
    extract=True
)

path_to_file = os.path.dirname(path_to_zip) + "/spa-eng/spa.txt"
```

```
Downloading data from http://storage.googleapis.com/download.tensorflow.org/data/s
pa-eng.zip
2646016/2638744 [==============================] - 1s 0us/step
```

In [3]:

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

def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())
    w = re.sub(r"([?.!,¿])", r"W1", w)
    w = re.sub(r'[" "]+', " ", w)
    w = re.sub(r"[^a-zA-Z?.!,¿]+", " ", w)
    w = w.rstrip().strip()
    w = '<start> ' + w + ' <end>'
    return w
```

In [4]:

```
en_sentence = u"May I borrow this book?"
sp_sentence = u"¿Puedo tomar prestado este libro?"

print(preprocess_sentence(en_sentence))
print(preprocess_sentence(sp_sentence))
```

```
<start> may i borrow this book? <end>
<start> ¿puedo tomar prestado este libro? <end>
```

In [6]:

```
def create_dataset(path, num_examples):
    lines = io.open(path, encoding='UTF-8').read().strip().split('Wn')
    word_pairs = [[preprocess_sentence(w) for w in l.split('Wt')] W
                  for l in lines[:num_examples]]
    return zip(*word_pairs)
```

In [8]:

```
en, sp = create_dataset(path_to_file, None)

print(en[-1])
print()
print(sp[-1])
```

```
<start> if you want to sound like a native speaker, you must be willing to practic
e saying the same sentence over and over in the same way that banjo players practi
ce the same phrase over and over until they can play it correctly and at the desir
ed tempo. <end>

<start> si quieres sonar como un hablante nativo, debes estar dispuesto a practica
r diciendo la misma frase una y otra vez de la misma manera en que un musico de ba
njo practica el mismo fraseo una y otra vez hasta que lo puedan tocar correctament
e y en el tiempo esperado. <end>
```

In [9]:

```
def max_length(tensor):
    return max(len(t) for t in tensor)
```

In [10]:

```
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(filters='')
    lang_tokenizer.fit_on_texts(lang)
    tensor = lang_tokenizer.texts_to_sequences(lang)
    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                           padding='post')
    return tensor, lang_tokenizer
```

In [11]:

```
def load_dataset(path, num_examples=None):
    targ_lang, inp_lang = create_dataset(path, num_examples)
    input_tensor, inp_lang_tokenizer = tokenize(inp_lang)
    target_tensor, targ_lang_tokenizer = tokenize(targ_lang)
    return input_tensor, target_tensor, inp_lang_tokenizer, targ_lang_tokenizer
```

In [12]:

```
num_examples = 30000
input_tensor, target_tensor, inp_lang, targ_lang = load_dataset(
    path_to_file, num_examples)
max_length_targ, max_length_inp = max_length(target_tensor), max_length(input_tensor)
```

In [13]:

```
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = W
train_test_split(input_tensor, target_tensor, test_size=0.2)

print(len(input_tensor_train), len(target_tensor_train),
      len(input_tensor_val), len(target_tensor_val))
```

```
24000 24000 6000 6000
```

In [14]:

```
def convert(lang, tensor):
    for t in tensor:
        if t != 0:
            print("%d ----> %s" % (t, lang.index_word[t]))
```

In [15]:

```
print ("Input Language; index to word mapping")
convert(inp_lang, input_tensor_train[0])
print()
print("Target Language; index to word mapping")
convert(targ_lang, target_tensor_train[0])
```

```
Input Language; index to word mapping
1 ----> <start>
7 ----> a
4 ----> tom
92 ----> nunca
25 ----> le
568 ----> gusto
42 ----> mary.
2 ----> <end>

Target Language; index to word mapping
1 ----> <start>
4 ----> tom
100 ----> never
368 ----> liked
66 ----> mary.
2 ----> <end>
```

In [16]:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index) + 1
vocab_tar_size = len(targ_lang.word_index) + 1
dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, W
                                              target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

In [17]:

```
example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape
```

Out[17]:

```
(TensorShape([64, 14]), TensorShape([64, 9]))
```

In [25]:

```
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units, return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state=hidden)
        return output, state

    def initialize_hidden_state(self):
        return tf.zeros((self.batch_sz, self.enc_units))
```

In [26]:

```
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)

print('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape
))
print ('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.shape))
```

```
Encoder output shape: (batch size, sequence length, units) (64, 14, 1024)
Encoder Hidden state shape: (batch size, units) (64, 1024)
```

In [27]:

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
        hidden_with_time_axis = tf.expand_dims(query, 1)
        score = self.V(tf.nn.tanh(self.W1(values) + W
                                  self.W2(hidden_with_time_axis)))
        attention_weights = tf.nn.softmax(score, axis=1)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)
        return context_vector, attention_weights
```

In [28]:

```
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output)

print("Attention result shape: (batch size, units) {}".format(attention_result.shape))
print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(attention_weights.sh
ape))
```

```
Attention result shape: (batch size, units) (64, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (64, 14, 1)
```

In [30]:

```
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units, return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        context_vector, attention_weights = self.attention(hidden, enc_output)
        x = self.embedding(x)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)
        output, state = self.gru(x)
        output = tf.reshape(output, (-1, output.shape[2]))
        x = self.fc(output)
        return x, state, attention_weights
```

In [31]:

```
decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
sample_decoder_output, _, _ = decoder(tf.random.uniform((64, 1)), sample_hidden,
                                      sample_output)

print('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_output.shape))
```

```
Decoder output shape: (batch_size, vocab size) (64, 7578)
```

In [32]:

```
optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True,
                                                             reduction='none')

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    return tf.reduce_mean(loss_)
```

In [33]:

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(optimizer=optimizer,
                                 encoder=encoder,
                                 decoder=decoder)
```

In [34]:

```
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0
    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)
        dec_hidden = enc_hidden
        dec_input = tf.expand_dims([targ_lang.word_index['<start>']] * W
                                   BATCH_SIZE, 1)
        for t in range(1, targ.shape[1]):
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output)
            loss += loss_function(targ[:, t], predictions)
            dec_input = tf.expand_dims(targ[:, t], 1)
        batch_loss = (loss / int(targ.shape[1]))
        variables = encoder.trainable_variables + decoder.trainable_variables
        gradients = tape.gradient(loss, variables)
        optimizer.apply_gradients(zip(gradients, variables))
        return batch_loss
```

In [35]:

```
EPOCHS = 10

for epoch in range(EPOCHS):
    start = time.time()
    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in enumerate(dataset.take(steps_per_epoch)):
        batch_loss = train_step(inp, targ, enc_hidden)
        total_loss += batch_loss

        if batch % 100 == 0:
            print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1, batch,
                                                         batch_loss.numpy()))
    if (epoch + 1) % 2 == 0:
        checkpoint.save(file_prefix = checkpoint_prefix)

    print('Epoch {} Loss {:.4f}'.format(epoch + 1, total_loss / steps_per_epoch))
    print('Time taken for 1 epoch {} sec₩n'.format(time.time() - start))
```

```
Epoch 1 Batch 0 Loss 4.8698
Epoch 1 Batch 100 Loss 2.7608
Epoch 1 Batch 200 Loss 2.3120
Epoch 1 Batch 300 Loss 2.0999
Epoch 1 Loss 0.0391
Time taken for 1 epoch 45.418179512023926 sec

Epoch 2 Batch 0 Loss 1.9763
Epoch 2 Batch 100 Loss 1.9518
Epoch 2 Batch 200 Loss 1.8668
Epoch 2 Batch 300 Loss 1.7426
Epoch 2 Loss 0.0280
Time taken for 1 epoch 34.18775749206543 sec

Epoch 3 Batch 0 Loss 1.5558
Epoch 3 Batch 100 Loss 1.3890
Epoch 3 Batch 200 Loss 1.3416
Epoch 3 Batch 300 Loss 1.2968
Epoch 3 Loss 0.0206
Time taken for 1 epoch 32.60940766334534 sec

Epoch 4 Batch 0 Loss 0.9675
Epoch 4 Batch 100 Loss 0.8828
Epoch 4 Batch 200 Loss 1.0555
Epoch 4 Batch 300 Loss 0.8913
Epoch 4 Loss 0.0144
Time taken for 1 epoch 34.58552169799805 sec

Epoch 5 Batch 0 Loss 0.6234
Epoch 5 Batch 100 Loss 0.6572
Epoch 5 Batch 200 Loss 0.5679
Epoch 5 Batch 300 Loss 0.6786
Epoch 5 Loss 0.0096
Time taken for 1 epoch 32.49030900001526 sec

Epoch 6 Batch 0 Loss 0.3515
Epoch 6 Batch 100 Loss 0.3323
Epoch 6 Batch 200 Loss 0.4153
Epoch 6 Batch 300 Loss 0.3530
Epoch 6 Loss 0.0062
Time taken for 1 epoch 34.159064292907715 sec

Epoch 7 Batch 0 Loss 0.2276
Epoch 7 Batch 100 Loss 0.2529
Epoch 7 Batch 200 Loss 0.2212
Epoch 7 Batch 300 Loss 0.2795
Epoch 7 Loss 0.0040
Time taken for 1 epoch 32.48130941390991 sec

Epoch 8 Batch 0 Loss 0.1325
Epoch 8 Batch 100 Loss 0.1971
Epoch 8 Batch 200 Loss 0.1272
Epoch 8 Batch 300 Loss 0.1643
Epoch 8 Loss 0.0026
Time taken for 1 epoch 39.56803822517395 sec

Epoch 9 Batch 0 Loss 0.0922
Epoch 9 Batch 100 Loss 0.1107
Epoch 9 Batch 200 Loss 0.0910
Epoch 9 Batch 300 Loss 0.1146
Epoch 9 Loss 0.0019
```

```
Time taken for 1 epoch 32.66144371032715 sec

Epoch 10 Batch 0 Loss 0.0908
Epoch 10 Batch 100 Loss 0.0661
Epoch 10 Batch 200 Loss 0.0715
Epoch 10 Batch 300 Loss 0.0952
Epoch 10 Loss 0.0015
Time taken for 1 epoch 34.75432300567627 sec
```

In [36]:

```
def evaluate(sentence):
    attention_plot = np.zeros((max_length_targ, max_length_inp))
    sentence = preprocess_sentence(sentence)
    inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
    inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
                                                           maxlen=max_length_inp,
                                                           padding='post')
    inputs = tf.convert_to_tensor(inputs)
    result = ''
    hidden = [tf.zeros((1, units))]
    enc_out, enc_hidden = encoder(inputs, hidden)
    dec_hidden = enc_hidden
    dec_input = tf.expand_dims([targ_lang.word_index['<start>']], 0)

    for t in range(max_length_targ):
        predictions, dec_hidden, attention_weights = decoder(dec_input, dec_hidden,
                                                             enc_out)
        attention_weights = tf.reshape(attention_weights, (-1, ))
        attention_plot[t] = attention_weights.numpy()
        predicted_id = tf.argmax(predictions[0]).numpy()
        result += targ_lang.index_word[predicted_id] + ' '

        if targ_lang.index_word[predicted_id] == '<end>':
            return result, sentence, attention_plot
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence, attention_plot
```

In [37]:

```
def plot_attention(attention, sentence, predicted_sentence):
    fig = plt.figure(figsize=(10,10))
    ax = fig.add_subplot(1, 1, 1)
    ax.matshow(attention, cmap='viridis')

    fontdict = {'fontsize': 14}

    ax.set_xticklabels([''] + sentence, fontdict=fontdict, rotation=90)
    ax.set_yticklabels([''] + predicted_sentence, fontdict=fontdict)

    ax.xaxis.set_major_locator(ticker.MultipleLocator(1))
    ax.yaxis.set_major_locator(ticker.MultipleLocator(1))

    plt.show()
```

In [38]:

```
def translate(sentence):
    result, sentence, attention_plot = evaluate(sentence)

    print('Input: %s' % (sentence))
    print('Predicted translation: {}'.format(result))

    attention_plot = attention_plot[:len(result.split(' ')), :len(sentence.split(' '))]
    plot_attention(attention_plot, sentence.split(' '), result.split(' '))
```

In [39]:

```
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

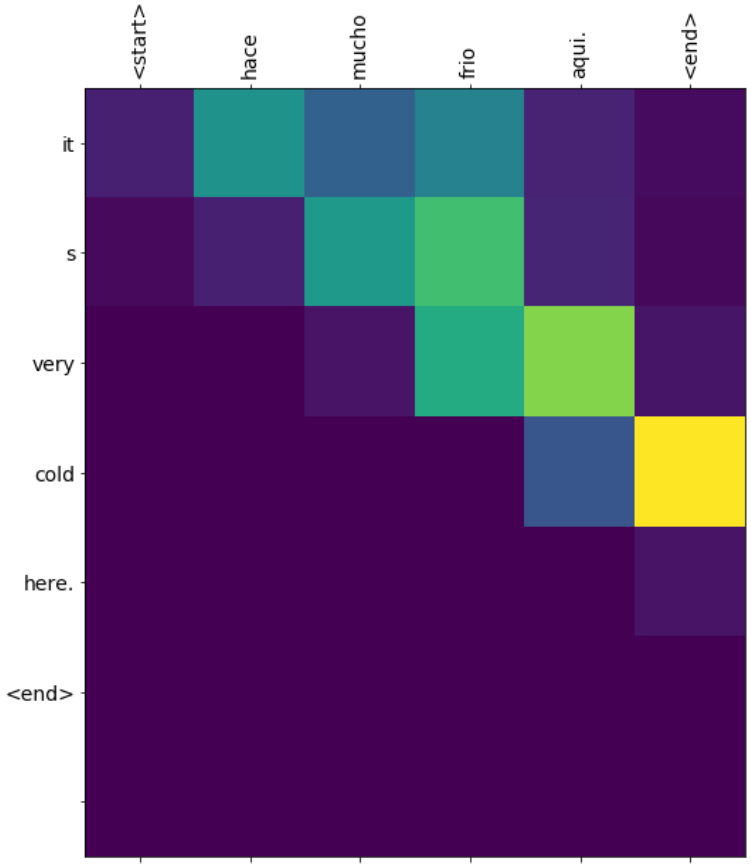
Out[39]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x1c6a9fa8e80>
```

In [40]:

```
translate(u'hace mucho frio aqui.')
```

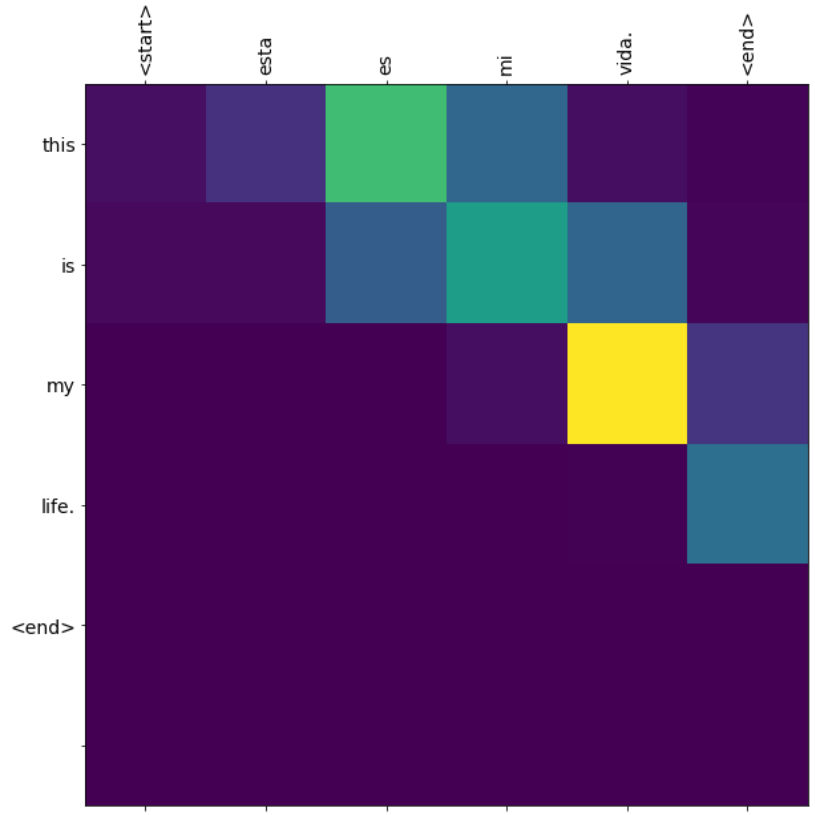
```
Input: <start> hace mucho frio aqui. <end>
Predicted translation: it s very cold here. <end>
```



In [41]:

```
translate(u'esta es mi vida.')
```

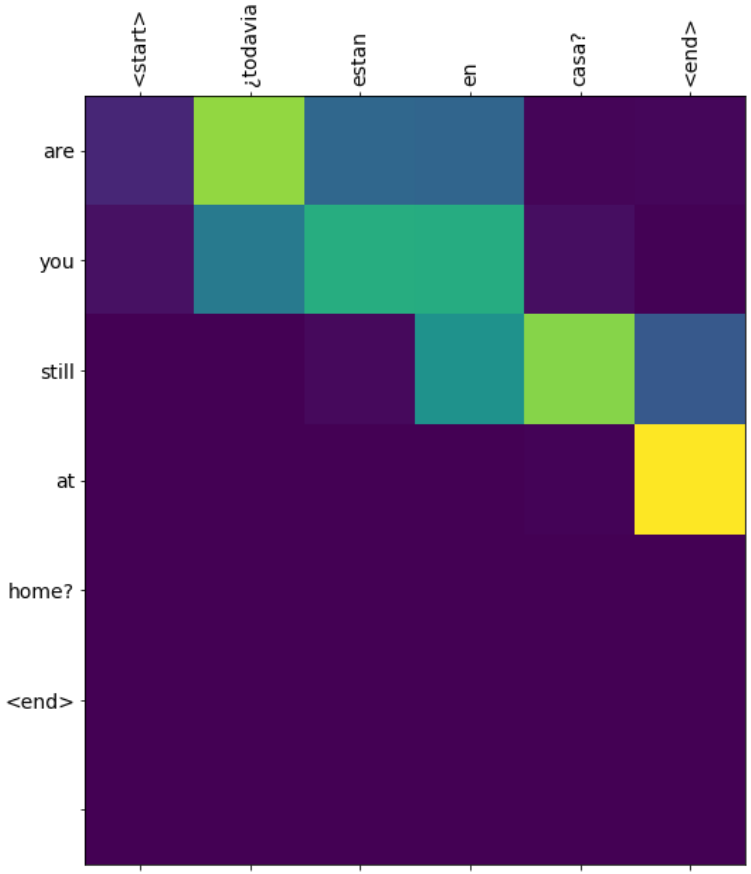
```
Input: <start> esta es mi vida. <end>
Predicted translation: this is my life. <end>
```



In [42]:

```
translate(u'¿todavia estan en casa?')
```

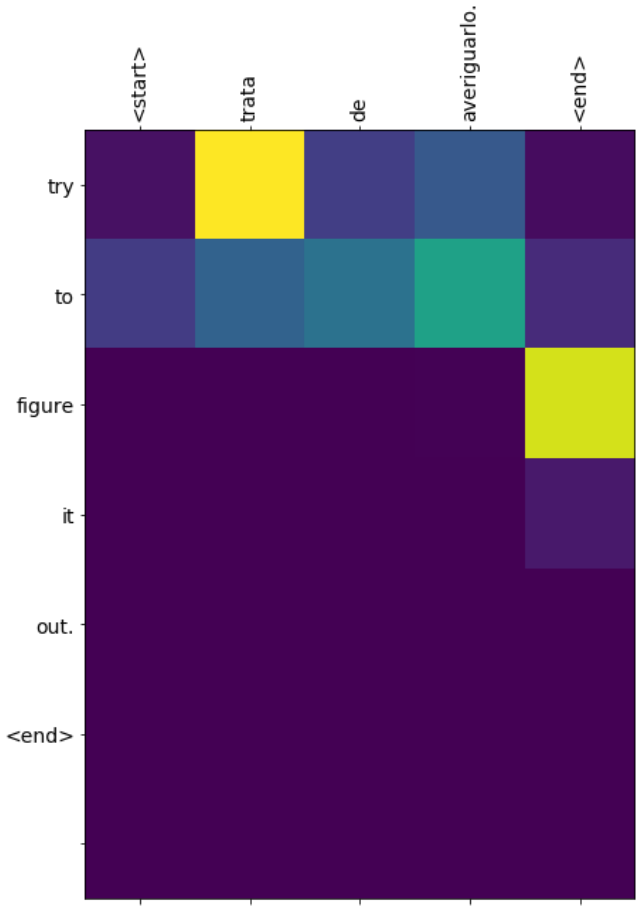
```
Input: <start> ¿todavia estan en casa? <end>
Predicted translation: are you still at home? <end>
```



In [43]:

```
translate(u'trata de averiguarlo.')
```

```
Input: <start> trata de averiguarlo. <end>
Predicted translation: try to figure it out. <end>
```



In [ ]:

```

```