```
[1] from google.colab import drive
    drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
cd gdrive/My\ Drive
```

```
/content/gdrive/My Drive
```

```
[3] cd NMT-Attention/
```

```
/content/gdrive/My Drive/NMT-Attention
```

```
#load data
###data
datapath = "data/train-en-vi/"

#read data
with open(datapath+"train.en", 'r', encoding='utf-8') as f:
    linesEN = f.read().split('\n')

with open(datapath+"train.vi", 'r', encoding='utf-8') as f:
    linesVI = f.read().split('\n')

trainEN = [] #data clean
trainVI = [] #data clean

for line in linesEN:
    temp = preprocess_sentence(line)
    trainEN.append(temp)

for line in linesVI:
    temp = preprocess_sentence(line)
    trainVI.append(temp)

#examples
trainEN = trainEN[0:30000]
trainVI = trainVI[0:30000]
```

```
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

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

def preprocess_sentence(w):
    w = unicode_to_ascii(w.lower().strip())
    w = re.sub(r"([?.!,¿])", r" \1 ", w)
    w = re.sub(r'[" "]+', " ", w)
    w = re.sub(r"[^a-zA-Z?.!,¿]+", " ", w)
    w = w.strip()
    w = '<start> ' + w + ' <end>'
    return w
```

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

    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
```

```
[6] def load_dataset(trainEN, trainVI, num_examples=None):
        # creating cleaned input, output pairs
        inp_lang = trainEN
        targ_lang = trainVI

        input_tensor, inp_lang_tokenizer = tokenize(in
        target_tensor, targ_lang_tokenizer = tokenize(targ_lang)

        return input_tensor, target_tensor, inp_lang_tokenizer, targ_lang_tokenizer

    # Try experimenting with the size of that dataset
    num_examples = 30000
    input_tensor, target_tensor, inp_lang, targ_lang = load_dataset(trainEN, trainVI, num_examples)

    # Calculate max_length of the target tensors
    max_length_targ, max_length_inp = max_length(target_tensor), max_length(input_tensor)

    # Creating training and validation sets using an 80-20 split
    input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = train_test_split(input_tensor, target_tensor, test_size=0.2)

    # Show length
    print(len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_tensor_val))

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

    #create dataset
    BUFFER_SIZE = len(input_tensor_train)
    BATCH_SIZE = 64
    steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
    embedding_dim = 256
    units = 128
    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, target_tensor_train)).shuffle(BUFFER_SIZE)
    dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

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

Tokenizer: targ_lang

keras_preprocessing.text.Tokenizer instance

```
24000 24000 6000 6000
(TensorShape([64, 137]), TensorShape([64, 202]))
```

[9]
```
#Encoder
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))


encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)
```

```
#Attention
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):
        # query hidden state shape == (batch_size, hidden size)
        # query_with_time_axis shape == (batch_size, 1, hidden size)
        # values shape == (batch_size, max_len, hidden size)
        # we are doing this to broadcast addition along the time axis to calculate the score
        query_with_time_axis = tf.expand_dims(query, 1)

        # score shape == (batch_size, max_length, 1)
        # we get 1 at the last axis because we are applying score to self.V
        # the shape of the tensor before applying self.V is (batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # attention_weights shape == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights

attention_layer = BahdanauAttention(10)
```

```
#Decoder
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)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)


    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # output shape == (batch_size * 1, hidden_size)
        output = tf.reshape(output, (-1, output.shape[2]))

        # output shape == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights

decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH_SIZE)
```

```
[18] translate('how are you?')
```

```
Input: <start> how are you ? <end>
Predicted translation: ban co the nao ? <end>
```

```
[19] translate('My name is Tri?')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-19-89d3039347dd> in <module>()
----> 1 translate('My name is Tri?')

                                  2 frames
<ipython-input-14-88c350b1987b> in <listcomp>(.0)
      4   sentence = preprocess_sentence(sentence)
      5
----> 6   inputs = [inp_lang.word_index[i] for i in sentence.split(' ')]
      7   inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
      8                                                          maxlen=max_length_inp,

KeyError: 'tri'

SEARCH STACK OVERFLOW
```