Neural Machine Translation using Transformers

```
# nightly version of tensorflow was used as they have more features available
!pip install -q tf-nightly > /dev/null 2>&1
!pip install -q tensorflow_text_nightly > /dev/null 2>&1
!pip install -q tensorflow_datasets

import os
import pathlib
import re
from nltk.translate.bleu_score import corpus_bleu
import numpy as np
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow_text as text
import tensorflow as tf
# from nightly
from tensorflow_text.tools.wordpiece_vocab import bert_vocab_from_dataset as bert_vocab


# to download the data-set, do not re-run if previously run
# dataset is already in lowercase and have space before and after punctuation.
tfds.disable_progress_bar()
builder = tfds.builder('ted_hrlr_translate/pt_to_en', data_dir=os.getcwd())
builder.download_and_prepare()


# loads the dataset
sets, info = tfds.load('ted_hrlr_translate/pt_to_en', data_dir=os.getcwd(),with_info=True, as_supervised=True, download=False)
train, val, test = sets['train'], sets['validation'], sets['test']  # 51785, 1193, 1803 examples in respective sets
for por_examples, eng_examples in test.batch(3).take(1):
  for pt in por_examples.numpy():
    print("Sample Portuguese: ", pt.decode('utf-8'))
  for en in eng_examples.numpy():
    print("Sample English: ", en.decode('utf-8'))

# Bert pre-trained language representation is used to feed vectors to the transformer model. It enhances the language understanding,
# by retaining context unlike word2vec or GloVe representation where each word only has one vector without context.
# Bert gives bidirectional context using encoder based on other words in the sequence giving more exploitation potential to the model.

# Bert wordpiece vocabulary of tensorflow is used, it uses Bert's splitting algorithm to split the text into words
# before generating the subword vocabulary on the pre-trained bert by Tensorflow team. Example vocabulary for hello: he###, hell##, #ello, etc.
```

```
reserved_tokens = ["[START]","[END]","[PAD]","[UNK]",]
bert_vocab_args = dict(
    vocab_size = 5000,
    bert_tokenizer_params=dict(lower_case=True),
    reserved_tokens = reserved_tokens,
    learn_params = {}
)

eng_train = train.map(lambda pt, en:en)
por_train = train.map(lambda pt, en:pt)

# creating the vocabulary file
eng_vocab = bert_vocab.bert_vocab_from_dataset(eng_train.batch(1000).prefetch(2), **bert_vocab_args)
with open(os.path.join(os.getcwd(),'eng_vocab.txt'), 'w') as f:
  for tok in eng_vocab:
    print(tok, file=f)
por_vocab = bert_vocab.bert_vocab_from_dataset(por_train.batch(1000).prefetch(2), **bert_vocab_args)
with open(os.path.join(os.getcwd(),'por_vocab.txt'), 'w') as f:
  for tok in por_vocab:
    print(tok,file=f)

addons = ["[START]","[END]","[PAD]"]

def add_reserved_tokens(vector):
  count = vector.bounding_shape()[0]
  starts = tf.fill([count,1], tf.argmax(tf.constant(addons) == "[START]"))
  ends = tf.fill([count,1], tf.argmax(tf.constant(addons) == "[END]"))
  return tf.concat([starts, vector, ends], axis=1)


def post_process(words):
  remove_addons = "|".join([re.escape(token) for token in addons])
  result = tf.ragged.boolean_mask(words, ~tf.strings.regex_full_match(words, remove_addons))
  return tf.strings.reduce_join(result, separator=' ', axis=-1)


# The bert wordpiece vocabulary is used by BertTokenizer to convert text string to wordpiece tokenization.
class custom_Bert(tf.Module):
  def __init__(self, vocab_path):
    self.bert = text.BertTokenizer(vocab_path, lower_case=True)
    self.vocab_path = vocab_path
    self.vocab = tf.Variable(pathlib.Path(vocab_path).read_text().splitlines())

  # vectorize the given string to token ids, preprocess data, add start and end tokens
  def tokenize(self, strings):
    tokenized_string = self.bert.tokenize(strings).merge_dims(-2,-1)
```

```
    tokenized_string = self.bert.tokenize(strings).merge_dims(-2,-1)
    return add_reserved_tokens(tokenized_string)

  # recreates the sentence using encoded tokens
  def detokenize(self, tokens):
    return post_process(self.bert.detokenize(tokens))

  # find the word from vocabulary using ids
  def ids_to_word(self, ids):
    return tf.gather(self.vocab, ids)

  def vocab_size(self):
    return tf.shape(self.vocab)[0]


bert_tokenizer = tf.Module()
bert_tokenizer.eng = custom_Bert(os.path.join(os.getcwd(),'eng_vocab.txt'))
bert_tokenizer.por = custom_Bert(os.path.join(os.getcwd(),'por_vocab.txt'))

# positional encoding layer using sin and cosines on alternate positions to get positional context.
def positional_encodings(position, embed_d):
  frequencies = np.arange(position).reshape(position,1)/(np.power(10000,(2*(np.arange(embed_d).reshape(1,embed_d)//2)/embed_d)))
  frequencies[:, 0::2] = np.sin(frequencies[:,0::2])
  frequencies[:, 1::2] = np.cos(frequencies[:,1::2])
  pos_enc = tf.cast(frequencies[np.newaxis, ...],tf.float32)
  return pos_enc


def feed_forward_network(embed_d, inner_d):
  ffn = tf.keras.Sequential(
      [
       tf.keras.layers.Dense(inner_d, activation='relu'),
       tf.keras.layers.Dense(embed_d)
      ]
  )
  return ffn

# attention to mask other values and only show seq which is being attended upon
def dot_product_attention(query, key, value, decode_mask):
  q_dot_k = tf.matmul(query,key,transpose_b=True)
  d = tf.cast(tf.shape(key)[-1], tf.float32)
  scaled_product = q_dot_k/tf.math.sqrt(d)
  if decode_mask is not None:
    scaled_product += (decode_mask * -1e9)
  attention_weights = tf.nn.softmax(scaled_product, axis=-1)
  scaled_attention = tf.matmul(attention_weights, value)
```

```
  return attention_weights, scaled_attention

# used to split heads and calculate attention which allows the model to jointly attend
# to the information from different representational dimensions.
class MultiHeadedAttention(tf.keras.layers.Layer):
  def __init__(self, embed_d, n_heads):
    super(MultiHeadedAttention, self).__init__()
    self.embed_d = embed_d
    self.n_heads = n_heads
    self.wq = tf.keras.layers.Dense(embed_d)
    self.wk = tf.keras.layers.Dense(embed_d)
    self.wv = tf.keras.layers.Dense(embed_d)
    self.depth = embed_d//self.n_heads
    self.dense = tf.keras.layers.Dense(embed_d)

  def spliting_heads(self, qkv, batch_size):
    qkv = tf.reshape(qkv, (batch_size, -1, self.n_heads, self.depth))
    return tf.transpose(qkv, perm=[0, 2, 1, 3])

  def call(self, v, k, q, mask):
    batch_size = tf.shape(q)[0]
    q = self.spliting_heads(self.wq(q), batch_size)
    k = self.spliting_heads(self.wk(k), batch_size)
    v = self.spliting_heads(self.wv(v), batch_size)

    attention_weights, scaled_attention = dot_product_attention(q, k, v, mask)
    scaled_attention = tf.transpose(scaled_attention, perm=[0,2,1,3])
    concated_attention = tf.reshape(scaled_attention, (batch_size, -1, self.embed_d))
    output = self.dense(concated_attention)
    return output, attention_weights

# Encoder architecture allows the self-attention where all of the keys, values and queries are same.
# hence attenting to all positions in the previous step of layer.
class EncoderLayer(tf.keras.layers.Layer):
  def __init__(self, embed_d, n_heads, inner_d, drop_rate):
    super(EncoderLayer, self).__init__()
    self.multi_heads = MultiHeadedAttention(embed_d, n_heads)
    self.dropout_layer1 = tf.keras.layers.Dropout(drop_rate)
    self.layer_norm1 = tf.keras.layers.LayerNormalization(axis=-1, epsilon=1e-6)
    self.ffn = feed_forward_network(embed_d, inner_d)
    self.dropout_layer2 = tf.keras.layers.Dropout(drop_rate)
    self.layer_norm2 = tf.keras.layers.LayerNormalization(axis=-1, epsilon=1e-6)

  def call(self, v, mask, training):
    encoder_self_attention, encoder_attention_weights = self.multi_heads(v,v,v, mask)
```

```
    encoder_self_attention = self.dropout_layer1(encoder_self_attention, training=training)
    normalized1 = self.layer_norm1(v+encoder_self_attention)
    linear_transform = self.ffn(normalized1)
    linear_transform = self.dropout_layer2(linear_transform, training=training)
    normalized2 = self.layer_norm2(normalized1+linear_transform)
    return normalized2

# this module includes embedding layers for input, positional encodings and output layer to feed into decoder
class Encoder(tf.keras.layers.Layer):
  def __init__(self, n_enc_layers, embed_d, n_heads, inner_d, vocab_size, max_pos, drop_rate):
    super(Encoder, self).__init__()
    self.embed_d = embed_d
    self.embedding_layer = tf.keras.layers.Embedding(vocab_size, embed_d)
    self.pos_encoding = positional_encodings(max_pos, self.embed_d)
    self.n_enc_layers = n_enc_layers
    self.enc_layers = [EncoderLayer(embed_d, n_heads, inner_d, drop_rate) for layer in range(n_enc_layers)]
    self.dropout_layer = tf.keras.layers.Dropout(drop_rate)

  def call(self, v, mask, training):
    input_seq_len = tf.shape(v)[1]
    v = self.embedding_layer(v)
    v *= tf.math.sqrt(tf.cast(self.embed_d, tf.float32))
    v += self.pos_encoding[:, :input_seq_len, :]
    v = self.dropout_layer(v, training=training)
    for layer in range(self.n_enc_layers):
      v = self.enc_layers[layer](v, mask, training)
    return v


class DecoderLayer(tf.keras.layers.Layer):
  def __init__(self, embed_d, n_heads, inner_d, drop_rate):
    super(DecoderLayer, self).__init__()
    self.masked_multi_heads = MultiHeadedAttention(embed_d, n_heads)
    self.dropout_layer1 = tf.keras.layers.Dropout(drop_rate)
    self.layer_norm1 = tf.keras.layers.LayerNormalization(axis=-1, epsilon=1e-6)
    self.multi_heads = MultiHeadedAttention(embed_d, n_heads)
    self.dropout_layer2 = tf.keras.layers.Dropout(drop_rate)
    self.layer_norm2 = tf.keras.layers.LayerNormalization(axis=-1, epsilon=1e-6)
    self.ffn = feed_forward_network(embed_d, inner_d)
    self.dropout_layer3 = tf.keras.layers.Dropout(drop_rate)
    self.layer_norm3 = tf.keras.layers.LayerNormalization(axis=-1, epsilon=1e-6)

  def call(self, v, look_ahead_mask, padding, training, encoder_output):
    decoder_self_attention, decoder_self_attn_wts = self.masked_multi_heads(v, v, v, look_ahead_mask)
    decoder_self_attention = self.dropout_layer1(decoder_self_attention, training=training)
    normalized1 = self.layer_norm1(decoder_self_attention + v)
```

```
    normalized1 = self.layer_norm1(decoder_self_attention + v)
    dec_enc_attention, dec_enc_attn_wts = self.multi_heads(encoder_output, encoder_output, normalized1, padding)
    dec_enc_attention = self.dropout_layer2(dec_enc_attention, training=training)
    normalized2 = self.layer_norm2(dec_enc_attention + normalized1)
    linear_transform = self.ffn(normalized2)
    linear_transform = self.dropout_layer3(linear_transform, training=training)
    normalized3 = self.layer_norm3(linear_transform + normalized2)
    return normalized3, decoder_self_attn_wts, dec_enc_attn_wts


class Decoder(tf.keras.layers.Layer):
  def __init__(self, n_dec_layers, embed_d, n_heads, inner_d, vocab_size, max_pos, drop_rate):
    super(Decoder, self).__init__()
    self.embed_d = embed_d
    self.embedding_layer = tf.keras.layers.Embedding(vocab_size, embed_d)
    self.pos_encoding = positional_encodings(max_pos, self.embed_d)
    self.n_dec_layers = n_dec_layers
    self.dec_layers = [DecoderLayer(embed_d, n_heads, inner_d, drop_rate) for layer in range(n_dec_layers)]
    self.dropout_layer = tf.keras.layers.Dropout(drop_rate)

  def call(self, v, look_ahead_mask, padding, training, encoder_output):
    target_seq_len = tf.shape(v)[1]
    v = self.embedding_layer(v)
    v *= tf.math.sqrt(tf.cast(self.embed_d, tf.float32))
    v += self.pos_encoding[:, :target_seq_len, :]
    v = self.dropout_layer(v, training=training)
    for layer in range(self.n_dec_layers):
      v, decoder_self_attn_wts, dec_enc_attn_wts = self.dec_layers[layer](v, look_ahead_mask, padding, training, encoder_output)
    return v, decoder_self_attn_wts, dec_enc_attn_wts


class NMT_Transformer(tf.keras.Model):
  def __init__(self, n_layers, embed_d, n_heads, inner_d, inp_vocab_size, tar_vocab_size, max_pos, drop_rate):
    super(NMT_Transformer, self).__init__()
    self.encoder = Encoder(n_layers, embed_d, n_heads, inner_d, inp_vocab_size, max_pos, drop_rate)
    self.decoder = Decoder(n_layers, embed_d, n_heads, inner_d, tar_vocab_size, max_pos, drop_rate)
    self.linear = tf.keras.layers.Dense(tar_vocab_size)

  def call(self, enc_dec_mask, look_ahead_mask, input, target, training):
    encoder_output = self.encoder(input, enc_dec_mask, training)
    decoder_output, decoder_self_attn_wts, dec_enc_attn_wts = self.decoder(target, look_ahead_mask, enc_dec_mask, training, encoder_output)
    linear_output = self.linear(decoder_output)
    return linear_output, decoder_self_attn_wts, dec_enc_attn_wts


class AdaptiveLR(tf.keras.optimizers.schedules.LearningRateSchedule):
```

```
  def __init__(self, embed_d, warmup_steps):
    super(AdaptiveLR, self).__init__()
    self.embed_d = tf.cast(embed_d, tf.float32)
    self.warmup_steps = warmup_steps

  def __call__(self, schedule):
    variable_lr = tf.math.rsqrt(self.embed_d)*tf.math.minimum(tf.math.rsqrt(schedule), schedule*(self.warmup_steps**-1.5))
    return variable_lr


train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.Mean(name='train_accuracy')


def loss_func(predicted, true):
  sparse_cat_loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')
  loss = sparse_cat_loss(true, predicted)
  pad_mask = tf.cast(tf.math.logical_not(tf.math.equal(true,0)), dtype=loss.dtype)
  return tf.reduce_sum(loss*pad_mask)/tf.reduce_sum(pad_mask)


def acc_func(predicted, true):
  acc = tf.equal(true, tf.argmax(predicted, axis=2))
  pad_mask = tf.math.logical_not(tf.math.equal(true,0))
  acc = tf.math.logical_and(pad_mask, acc)
  return tf.reduce_sum(tf.cast(acc, dtype=tf.float32))/tf.reduce_sum(tf.cast(pad_mask, dtype=tf.float32))


def masks(input, target):
  # padding masks will create a mask of 1's wherever there are 0's
  enc_dec_values = tf.cast(tf.math.equal(input, 0), tf.float32)
  enc_dec_mask = enc_dec_values[:, tf.newaxis, tf.newaxis, :]
  # look ahead mask is used to hide the future tokens from an index in decoder as that index needs to be predicted
  dec_look_ahead_mask = 1-tf.linalg.band_part(tf.ones((tf.shape(target)[1],tf.shape(target)[1])), -1, 0)
  dec_pad_values = tf.cast(tf.math.equal(target, 0), tf.float32)
  dec_target_mask = dec_pad_values[:, tf.newaxis, tf.newaxis, :]
  dec_masked = tf.maximum(dec_look_ahead_mask, dec_target_mask)
  return enc_dec_mask, dec_masked


def bert_mapping(pt, en):
  pt = bert_tokenizer.por.tokenize(pt).to_tensor()
  en = bert_tokenizer.eng.tokenize(en).to_tensor()
  return pt, en

train_batch = train.cache().shuffle(10000).batch(64).map(bert_mapping, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)
```

```
train_batch = train.cache().shuffle(10000).batch(64).map(bert_mapping, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)
val_batch = val.cache().shuffle(10000).batch(64).map(bert_mapping, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)
test_batch = test.cache().shuffle(10000).batch(64).map(bert_mapping, num_parallel_calls=tf.data.AUTOTUNE).prefetch(tf.data.AUTOTUNE)

nmt = NMT_Transformer(n_layers=2, embed_d=64, n_heads=2, inner_d=256, inp_vocab_size=bert_tokenizer.eng.vocab_size(), tar_vocab_size=bert_tokenizer.por.voca
lr = AdaptiveLR(embed_d=64, warmup_steps=1000)
optimizer = tf.keras.optimizers.Adam(lr, beta_1=0.9, beta_2=0.98, epsilon=1e-9)

train_signature = [tf.TensorSpec(shape=(None,None), dtype=tf.int64), tf.TensorSpec(shape=(None,None), dtype=tf.int64)]

@tf.function(input_signature=train_signature)
def training(input, target):
  # target input leaves last word in seq to predict
  target_input = target[:,:-1]
  # target input leaves first word in seq
  target_real = target[:,1:]
  enc_dec_mask, dec_masked = masks(input, target_input)

  with tf.GradientTape() as tape:
    prediction, decoder_self_attn_wts, dec_enc_attn_wts = nmt(enc_dec_mask, dec_masked, input, target_input, training=True)
    loss = loss_func(prediction, target_real)

  grad = tape.gradient(loss, nmt.trainable_variables)
  optimizer.apply_gradients(zip(grad, nmt.trainable_variables))
  train_loss(loss)
  train_accuracy(acc_func(prediction,target_real))
```

```
Sample Portuguese:  depois , podem fazer-se e testar-se previsões .
Sample Portuguese:  forçou a parar múltiplos laboratórios que ofereciam testes brca .
Sample Portuguese:  as formigas são um exemplo clássico ; as operárias trabalham para as rainhas e vice-versa .
Sample English:  then , predictions can be made and tested .
Sample English:  it had forced multiple labs that were offering brca testing to stop .
Sample English:  ants are a classic example ; workers work for queens and queens work for workers .
```

```
epochs = 2
for epoch in range(epochs):
  train_loss.reset_states()
  train_accuracy.reset_states()
  for (batch, (por, eng)) in enumerate(train_batch):
    training(eng,por)
    if batch % 100 ==0:
      print(f'Epoch no. {epoch+1} Batch {batch} Loss {train_loss.result():.3f} Accuracy {train_accuracy.result():.3f}')
  print(f'Epoch no. {epoch + 1} Loss {train_loss.result():.3f} Accuracy {train_accuracy.result():.3f}')
nmt.save_weights(os.path.join(os.getcwd(), 'weights'))

def translate(sentence, max_seq_len=40):
```

```
  tokenized_input = bert_tokenizer.eng.tokenize(tf.convert_to_tensor([sentence])).to_tensor()
  start_token, end_token = bert_tokenizer.por.tokenize([''])[0]
  target = tf.expand_dims(tf.convert_to_tensor([start_token]),0)
  for token in range(max_seq_len):
    enc_dec_mask, dec_masked = masks(tokenized_input, target)
    prediction, decoder_self_attn_wts, dec_enc_attn_wts = nmt(enc_dec_mask, dec_masked, tokenized_input, target, training=False)
    prediction_id = tf.argmax(prediction[:, -1:, :], axis=-1)
    target = tf.concat([target, prediction_id], axis=-1)
    if prediction_id == end_token:
      break

  translated_text = bert_tokenizer.por.detokenize(target)[0]
  translated_tokens = bert_tokenizer.por.ids_to_word(target)[0]
  return translated_text, translated_tokens, decoder_self_attn_wts, dec_enc_attn_wts
```

```
    Epoch no. 1 Batch 0 Loss 8.506 Accuracy 0.000
    Epoch no. 1 Batch 100 Loss 8.093 Accuracy 0.028
    Epoch no. 1 Batch 200 Loss 7.349 Accuracy 0.044
    Epoch no. 1 Batch 300 Loss 6.865 Accuracy 0.075
    Epoch no. 1 Batch 400 Loss 6.507 Accuracy 0.100
    Epoch no. 1 Batch 500 Loss 6.227 Accuracy 0.120
    Epoch no. 1 Batch 600 Loss 6.000 Accuracy 0.137
    Epoch no. 1 Batch 700 Loss 5.807 Accuracy 0.152
    Epoch no. 1 Batch 800 Loss 5.639 Accuracy 0.165
    Epoch no. 1 Loss 5.626 Accuracy 0.166
    Epoch no. 2 Batch 0 Loss 4.378 Accuracy 0.267
    Epoch no. 2 Batch 100 Loss 4.334 Accuracy 0.269
    Epoch no. 2 Batch 200 Loss 4.276 Accuracy 0.275
    Epoch no. 2 Batch 300 Loss 4.237 Accuracy 0.278
    Epoch no. 2 Batch 400 Loss 4.190 Accuracy 0.283
    Epoch no. 2 Batch 500 Loss 4.141 Accuracy 0.288
    Epoch no. 2 Batch 600 Loss 4.095 Accuracy 0.293
    Epoch no. 2 Batch 700 Loss 4.047 Accuracy 0.298
    Epoch no. 2 Batch 800 Loss 4.003 Accuracy 0.303
    Epoch no. 2 Loss 3.999 Accuracy 0.304
```

```
def test_bleu():
  test_input = []
  test_predicted = []
  test_truth = []
  bleu_scores = []
  for por_examples, eng_examples in test.batch(30).take(10):
    for pt in por_examples.numpy():
      test_truth.append(pt.decode('utf-8'))
    for en in eng_examples.numpy():
      test_input.append(en.decode('utf-8'))
      translated_text, ... = translate(en.decode('utf-8'))
```

```
      translated_text, _, _, _ = translate(en.decode("utf-8"))
      test_predicted.append(translated_text.numpy().decode("utf-8"))

  bleu1 = corpus_bleu(test_truth, test_predicted, weights=(1.0, 0, 0, 0))
  print('BLEU-1 results: {:.3f}'.format(bleu1))
  bleu2 = corpus_bleu(test_truth, test_predicted, weights=(0.5, 0.5, 0, 0))
  print('BLEU-2 results: {:.3f}'.format(bleu2))
  bleu3 = corpus_bleu(test_truth, test_predicted, weights=(0.3, 0.3, 0.3, 0))
  print('BLEU-3 results: {:.3f}'.format(bleu3))
  bleu4 = corpus_bleu(test_truth, test_predicted, weights=(0.25, 0.25, 0.25, 0.25))
  print('BLEU-4 results: {:.3f}'.format(bleu4))
  bleu_scores = [bleu1, bleu2, bleu3, bleu4]
  return test_input, test_predicted, test_truth, bleu_scores

print("Bleu scores computed on 300 examples from test set and sample results: ")
test_input, test_predicted, test_truth, bleu_scores = test_bleu()
for ind in range(6):
  print("Sample Input: ", test_input[ind])
  print("Sample Predicted: ", test_predicted[ind])
  print("Sample Truth: ", test_truth[ind])
```

```
Bleu scores computed on 300 examples from test set and sample results:
/usr/local/lib/python3.7/dist-packages/nltk/translate/bleu_score.py:490: UserWarning:
Corpus/Sentence contains 0 counts of 2-gram overlaps.
BLEU scores might be undesirable; use SmoothingFunction().
  warnings.warn(_msg)
BLEU-1 results: 0.143
BLEU-2 results: 0.378
BLEU-3 results: 0.557
BLEU-4 results: 0.615
Sample Input:  then , predictions can be made and tested .
Sample Predicted:  ` ` ` ` ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' , exproventaveis . '
Sample Truth:  depois , podem fazer-se e testar-se previsões .
Sample Input:  it had forced multiple labs that were offering brca testing to stop .
Sample Predicted:  ` ` ` ` ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' '
Sample Truth:  forçou a parar múltiplos laboratórios que ofereciam testes brca .
Sample Input:  ants are a classic example ; workers work for queens and queens work for workers .
Sample Predicted:  ` ` ` ` ' ' ' ' um exemplo , os dias , os dias expeitos para os trabalhadores . ' '
Sample Truth:  as formigas são um exemplo clássico ; as operárias trabalham para as rainhas e vice-versa .
Sample Input:  one of every hundred children born worldwide has some kind of heart disease .
Sample Predicted:  ` ` ` ` ' ' ' ' de todos os miudos do mundo . ' ' '
Sample Truth:  uma em cada cem crianças no mundo nascem com uma doença cardíaca .
Sample Input:  at this point in her life , she 's suffering with full-blown aids and had pneumonia .
Sample Predicted:  ` ` ` ` ' ' ' o seu ponto de ` ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' ' . '
Sample Truth:  neste momento da sua vida , ela está a sofrer de sida no seu expoente máximo e tinha pneumonia .
Sample Input:  where are economic networks ?
Sample Predicted:  ` ` ` ` ' ' ' onde sao as redes sociais ? ' ' '
Sample Truth:  onde estão as redes económicas ?
```

```
sample_text = "where are we going after lunch, do you have any plans or we should make on our way?"
translated_text, _, _, _ = translate(sample_text)
print(translated_text.numpy().decode("utf-8"))

# truth: as formigas são um exemplo clássico ; as operárias trabalham para as rainhas e vice-versa .
sample_text3 = "ants are a classic example ; workers work for queens and queens work for workers."
translated_text, _, _, _ = translate(sample_text3)
print(translated_text.numpy().decode("utf-8"))
```

```
    ` ` ` ` ' ' ' onde nos vamos ter descordar ou os nossos estudos ? ' ' '
    ` ` ` ` ' ' ' ' um exemplo , os dias , os dias expeitos para os trabalhadores . ' '
```

✓ 3s completed at 22:28