In [1]:

```
import tensorflow as tf

import typing
from typing import Any, Tuple

import tensorflow_text as tf_text

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker


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 [17]:

```
path_to_file = "rus.txt"
```

In [18]:

```
def preprocess_sentence(w):
    w = w.lower().strip()

    # убираю пробелы в начале и в конце
    # где знаки пунктуации окаймляю их пробелами
    w = re.sub(r"([?.!,])", r" \1 ", w)
    w = re.sub(r'[" "]+', " ", w)

    # заменяю все на пробелы кроме (a-z, A-Z, ".", "?", "!", ",", "'")
    w = re.sub(r"[^a-zA-Za-яА-Я?.!,']+", " ", w)

    w = w.strip()

    # добавление маркера начала и конца к предложению
    # чтобы модель знала, когда начинать и прекращать прогнозирование.
    w = '<start> ' + w + ' <end>'
    return w
```

In [19]:

```
a = 'HeLlo, My woRld! ))'
preprocess_sentence(a)
```

Out[19]:

```
'<start> hello , my world ! <end>'
```

In [20]:

```
# 1. Убираю акценты
# 2. Очищаю предложения
# 3. Возвращает пары слов в формате: [ENG, RUS]
def create_dataset(path, num_examples):
    lines = io.open(path, encoding='UTF-8').read().strip().split('\n')

    word_pairs = [[preprocess_sentence(w) for w in l.split('\t')[:2]]  for l in lines[:num_

    return zip(*word_pairs)
```

In [21]:

```
en, ru = create_dataset(path_to_file, None)
print(en[1])
print(ru[1])
```

```
<start> go . <end>
<start> иди . <end>
```

In [22]:

```
lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
      filters='')
lang_tokenizer.fit_on_texts(a)
tensor = lang_tokenizer.texts_to_sequences(a)

tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                       padding='post')

tensor, lang_tokenizer
```

Out[22]:

```
(array([[ 4],
        [ 5],
        [ 1],
        [ 1],
        [ 2],
        [ 6],
        [ 0],
        [ 7],
        [ 8],
        [ 0],
        [ 9],
        [ 2],
        [10],
        [ 1],
        [11],
        [12],
        [ 0],
        [ 3],
        [ 3]]),
 <keras.preprocessing.text.Tokenizer at 0x22912ea9648>)
```

ф-ция преобразует текст в вектор чисел и делает последовательности одинаковой длины

In [23]:

```
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 [24]:

```
def load_dataset(path, num_examples=None):
    # создание очищенных входных и выходных пар
    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 [25]:

```
len(en), len(ru)
```

Out[25]:

```
(444587, 444587)
```

In [26]:

```
num_examples = 100000
target_tensor, input_tensor ,targ_lang, inp_lang  = load_dataset(path_to_file, num_examples

# Вычисляю максимальную длину целевых тензоров
max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]
```

In [27]:

```
# Создание обучающих и проверочных наборов с использованием разделения на 80-20
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = train_test_s

# посмотрим длину
print(len(input_tensor_train), len(target_tensor_train), len(input_tensor_val), len(target_
```

```
80000 80000 20000 20000
```

In [28]:

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

In [29]:

```
print ("Язык ввода; сопоставление индекса со словом")
convert(inp_lang, input_tensor_train[-10])
print ()
print ("Целевой язык; сопоставление индекса со словом")
convert(targ_lang, target_tensor_train[-10])
```

```
Язык ввода; сопоставление индекса со словом
1 ----> <start>
20 ----> don't
422 ----> stand
34 ----> in
22 ----> my
273 ----> way
3 ----> .
2 ----> <end>

Целевой язык; сопоставление индекса со словом
1 ----> <start>
7 ----> не
1239 ----> стой
14 ----> у
16 ----> меня
20446 ----> попер
101 ----> к
2819 ----> дороги
24 ----> !
2 ----> <end>
```

Создаю набор данных tf.data

In [30]:

```
len(inp_lang.word_index)+1
```

Out[30]:

```
7334
```

In [31]:

```
BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
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, target_tensor_train)).shu
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

In [32]:

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

Out[32]:

```
(TensorShape([64, 11]), TensorShape([64, 15]))
```

encoder

In [33]:

```
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))
```

В слое внедрения слов форма образца изменяется с (64, 16) на (64, 11, 256),
то есть каждое слово становится 256-мерным вектором.
В слое GRU с 1024 нейронами форма образца изменяется с (64, 16, 256) на (64, 11, 1024),
форма скрытого вектора в слое GRU равна (64, 1024).

In [34]:

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

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print ('Форма выходного сигнала энкодера: (batch size, sequence length, units) {}'.format(s
print ('Форма скрытого состояния энкодера: (batch size, units) {}'.format(sample_hidden.sha
```

```
Форма выходного сигнала энкодера: (batch size, sequence length, units) (64,
11, 1024)
Форма скрытого состояния энкодера: (batch size, units) (64, 1024)
```

Внимание Богданау

In [39]:

```
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):
        # форма скрытого состояния запроса == (batch_size, hidden size)
        # запрос с формой временной оси == (batch_size, 1, hidden size)
        # размер значения == (batch_size, max_len, hidden size)
        query_with_time_axis = tf.expand_dims(query, 1)

        #  размер score == (batch_size, max_length, 1)
        # мы получаем 1 на последней оси, потому что мы применяем оценку к self.V
        # форма тензора перед применением self.V is (batch_size, max_length, units)
        score = self.V(tf.nn.tanh(
            self.W1(query_with_time_axis) + self.W2(values)))

        # форма  весов внимания == (batch_size, max_length, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # форма context_vector  после сложения == (batch_size, hidden_size)
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

здесь values Фактически, это результат, выводимый кодировщиком после прохождения через нейрон Dense(10)
После слоя его форма изменилась с (64, 11, 1024) на (64, 11, 10).

здесь query Фактически, это выходной вектор скрытого слоя в кодировщике.
Нам нужно изменить его размерность с (64, 1024) на (64, 1, 1024),
чтобы выполнить последующее сложение для вычисления оценки.
Вектор после увеличения размерности будет содержать 10 нейронов Dense
После слоя его форма изменилась с (64, 1, 1024) на (64, 1, 10).

Форма, полученная сложением двух вышеупомянутых выходов, равна (64, 11, 10) после того, как нейрон Dense(1)
Получите после слоя score, Его форма принимает вид (64, 11, 1).

Softmax По умолчанию применяется к последней оси, но здесь мы хотим применить его ко второй оси (т.е. ось = 1), потому что форма оценки (размер пакета, максимальная длина, размер скрытого слоя). Максимальная длина - это длина нашего ввода. Поскольку мы хотим присвоить вес каждому входу, на этой оси следует использовать softmax.
Проходить мимо Softmax После слоя полученная форма веса внимания и score Формы одинаковые, обе (64, 11, 1). Softmax Правила расчета различных осей относятся к:Какова роль оси в tf.nn.softmax (x, axis)?

Вес внимания и values умножаются, чтобы получить вектор контекста, его форма (64, 11, 1024). Этот вектор также является вектором взвешенного кодирования. Вектор контекста суммируется на основе второй оси (причина та же, что и раньше), и получается окончательный вектор контекста, форма которого (64, 1024).

In [40]:

```
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_weigh
```

```
Attention result shape: (batch size, units) (64, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (64, 11, 1)
```

In [41]:

```
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)
    #     self.attention = tf.keras.layers.Attention(use_scale=True, score_mode='concat')

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

        # x форма после прохождения через embedding == (batch_size, 1, embedding_dim)
        x = self.embedding(x)

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

        # передача объединенного вектора в GRU
        output, state = self.gru(x)

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

        # форма выходного  слоя == (batch_size, vocab)
        x = self.fc(output)

        return x, state, attention_weights
```

В декодере сначала использую уровень внимания, чтобы получить вектор контекста (64, 1024) и весовой коэффициент внимания (64, 11, 1).

Передаю ввод через слой внедрения слов, и его форма изменится с (64, 1) на (64, 1, 256).

Добавляю измерение к вектору контекста, чтобы его форма стала (64, 1, 1024), а затем объединяю его с вводом через слой встраивания слов, чтобы получить вектор формы (64, 1, 1280).

Объединенный вектор отправляется в GRU, и получается выходной вектор (64, 1, 1024) и скрытое состояние (64, 1024).

Преобразую выходной вектор в двумерный массив с неизменным последним измерением, то есть форма имеет вид (64 * 1, 1024).

Наконец, этот вектор проходит через полностью связанный слой, содержащий размер нейронов словаря (4935), чтобы получить окончательный результат (64, 4935).

In [42]:

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

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                      sample_hidden, sample_output)

print ('Форма выходного сигнала декодера: (batch_size, vocab size) {}'.format(sample_decode
```

```
Форма выходного сигнала декодера: (batch_size, vocab size) (64, 20544)
```

Определяю оптимизатор и функцию потерь

Вектор, входящий в декодер, каждый раз представляет собой слово во всех текстах в пакете, то есть форма входного вектора каждый раз равна (64, 1).

Когда во входном векторе появляется элемент 0, это означает, что текст,
в котором расположен этот элемент, закончился, и этот текст больше не участвует
в вычислении потерь, поэтому при вычислении потерь используется обработка маски,
чтобы установить потерю законченный текст обнулить

In [43]:

```
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_)
```

Обучение

Градиентный спуск

а. Передаю ввод на кодировщик, и кодировщик вернет Выход энкодера с состоянием скрытого слоя кодировщика.
б. Выход кодера, состояние скрытого слоя кодировщика и вход декодера (т. е.‘’) К декодеру.
в. Декодер возвращает прогноз статус скрытого слоя декодера.
г. Состояние скрытого слоя декодера отправляется обратно в модель, и прогноз используется для вычисления потерь.
д. Использование учителя обязательно(Учитель форсирования) определяет следующий вход декодера.
PS: принуждение учителя - это технология, которая отправляет целевое слово в качестве следующего ввода в декодер.
е. Последний шаг - вычислить градиент и применить его к оптимизатору и обратному распространению.

In [44]:

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

In [26]:

```
@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>']] * 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
```

Тренировочный процесс

In [28]:

```
from tqdm import tqdm
```

In [31]:

```
EPOCHS = 50

for epoch in tqdm(range(EPOCHS)):
    start = time.time()

    enc_hidden = encoder.initialize_hidden_state()
    total_loss = 0

    for (batch, (inp, targ)) in tqdm(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()))
      # saving (checkpoint) the model every 2 epochs
    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))
```

```
  0%|
| 0/50 [00:00<?, ?it/s]
0it [00:00, ?it/s]
1it [00:02,  2.89s/it]

Epoch 1 Batch 0 Loss 1.7854


2it [00:05,  2.50s/it]
3it [00:07,  2.36s/it]
4it [00:09,  2.31s/it]
5it [00:11,  2.25s/it]
6it [00:16,  3.10s/it]
7it [00:19,  3.21s/it]
8it [00:26,  4.15s/it]
9it [00:31,  4.68s/it]
10it [00:34,  4.04s/it]
11it [00:39,  4.49s/it]
12it [00:48,  4.01s/it]
  0%|
| 0/50 [00:48<?, ?it/s]


---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
C:\Temp/ipykernel_4628/3369349150.py in <module>
      8
      9     for (batch, (inp, targ)) in tqdm(enumerate(dataset.take(steps_pe
r_epoch))):
---> 10         batch_loss = train_step(inp, targ, enc_hidden)
     11         total_loss += batch_loss
     12

c:\program files\python37\lib\site-packages\tensorflow\python\util\traceback
```

```
_utils.py in error_handler(*args, **kwargs)
    148     filtered_tb = None
    149     try:
--> 150       return fn(*args, **kwargs)
    151     except Exception as e:
    152       filtered_tb = _process_traceback_frames(e.__traceback__)

c:\program files\python37\lib\site-packages\tensorflow\python\eager\def_func
tion.py in __call__(self, *args, **kwds)
    913
    914       with OptionalXlaContext(self._jit_compile):
--> 915         result = self._call(*args, **kwds)
    916
    917       new_tracing_count = self.experimental_get_tracing_count()

c:\program files\python37\lib\site-packages\tensorflow\python\eager\def_func
tion.py in _call(self, *args, **kwds)
    945       # In this case we have created variables on the first call, so
we run the
    946       # defunned version which is guaranteed to never create variabl
es.
--> 947       return self._stateless_fn(*args, **kwds)  # pylint: disable=no
t-callable
    948     elif self._stateful_fn is not None:
    949       # Release the lock early so that multiple threads can perform
 the call

c:\program files\python37\lib\site-packages\tensorflow\python\eager\functio
n.py in __call__(self, *args, **kwargs)
   2452        filtered_flat_args) = self._maybe_define_function(args, kwarg
s)
   2453     return graph_function._call_flat(
-> 2454         filtered_flat_args, captured_inputs=graph_function.captured_
inputs)  # pylint: disable=protected-access
   2455
   2456   @property

c:\program files\python37\lib\site-packages\tensorflow\python\eager\functio
n.py in _call_flat(self, args, captured_inputs, cancellation_manager)
   1859       # No tape is watching; skip to running the function.
   1860       return self._build_call_outputs(self._inference_function.call(
-> 1861           ctx, args, cancellation_manager=cancellation_manager))
   1862     forward_backward = self._select_forward_and_backward_functions(
   1863         args,

c:\program files\python37\lib\site-packages\tensorflow\python\eager\functio
n.py in call(self, ctx, args, cancellation_manager)
    500               inputs=args,
    501               attrs=attrs,
--> 502               ctx=ctx)
    503         else:
    504           outputs = execute.execute_with_cancellation(

c:\program files\python37\lib\site-packages\tensorflow\python\eager\execute.
py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     53     ctx.ensure_initialized()
     54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op
_name,
---> 55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:
     57     if name is not None:
```

```
KeyboardInterrupt:
```

Прогноз

Функция прогноза

1. Использую preprocess_sentence функция для обработки текста ascii;
2. Преобразую текст в цифровой вектор;
3. Преобразую цифровой вектор в тензор;
4. Инициализирую скрытое состояние кодировщика;
5. Ввожу цифровой тензор и начальное скрытое состояние в кодировщик;
6. Инициализирую вход декодера;
7. Дословно ввожу декодер, чтобы получить предсказанный текст.

In [2]:

```
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)

        # Сохраните веса внимания для последующего рисования
        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

        # Прогнозируемый идентификатор возвращается в модель
        dec_input = tf.expand_dims([predicted_id], 0)

    return result, sentence, attention_plot
```

Функция отображения веса внимания

In [3]:

```
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()
```

Функция перевода

In [5]:

```
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 [45]:

```
# восстановление последней контрольной точки в checkpoint_dir
checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Out[45]:

```
<tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x22926777
b08>
```

In [46]:

```
translate('See you later') # До скорого
```

```
Input: <start> see you later <end>
Predicted translation: увидимся позже . <end>
```

In [47]:

```
translate('Have a nice day')# хорошего дня
```

```
Input: <start> have a nice day <end>
Predicted translation: счастливого дня . <end>
```

In [48]:

```
translate('How are you doing?') # как дела?
```

```
Input: <start> how are you doing ? <end>
Predicted translation: как поживаете ? <end>
```

In [49]:

```
translate('How about you?') # а как у тебя дела?
```

```
Input: <start> how about you ? <end>
Predicted translation: как насч т ? <end>
```

In [50]:

```
translate('Thank you very much') # Спасибо большое
```

```
Input: <start> thank you very much <end>
Predicted translation: спасибо большое спасибо . <end>
```

In [51]:

```
translate('I appreciate it') # я ценю это
```

```
Input: <start> i appreciate it <end>
Predicted translation: я ценю это . <end>
```

In [52]:

```
translate('Always welcome') # Всегда пожалуйста
```

```
Input: <start> always welcome <end>
Predicted translation: все будут лучше . <end>
```

In [53]:

```
translate('Not at all') # Не за что
```

```
Input: <start> not at all <end>
Predicted translation: вс не вс равно . <end>
```

In [54]:

```
translate('Excuse me') # Простите
```

```
Input: <start> excuse me <end>
Predicted translation: простите меня . <end>
```

In [55]:

```
translate('No problem!') # Без проблем!
```

```
Input: <start> no problem ! <end>
Predicted translation: ни у кого не было . <end>
```

In [56]:

```
translate('I’m absolutely sure') # Я совершенно уверен
```

```
Input: <start> i m absolutely sure <end>
Predicted translation: я проспал я . <end>
```

In [57]:

```
translate('Maybe') # Может быть
```

```
Input: <start> maybe <end>
Predicted translation: да те . <end>
```

In [58]:

```
translate('As far as I know') # Насколько я знаю
```

```
Input: <start> as far as i know <end>
Predicted translation: делайте как я знаю как . <end>
```

In [59]:

```
translate('It seems to me') # Мне кажется
```

```
Input: <start> it seems to me <end>
Predicted translation: это кажется мне . <end>
```

In [60]:

```
translate('To be honest') # Честно говоря
```

```
Input: <start> to be honest <end>
Predicted translation: поешьте , будь честен . <end>
```

In [74]:

```
translate('How much is …?') # Сколько стоит …?
```

```
Input: <start> how much is ? <end>
Predicted translation: сколько стоят ? <end>
```

In [75]:

```
translate('How do I go to…?') # Как мне пройти …?
```

```
Input: <start> how do i go to ? <end>
Predicted translation: как мне пойти ? <end>
```

In [70]:

```
translate('Would you like…?') # Не желаете ли вы…?
```

```
Input: <start> would you like ? <end>
Predicted translation: разве бы желаете ? <end>
```

In [71]:

```
translate('Can I offer you…?') # Могу я предложить вам…?
```

```
Input: <start> can i offer you ? <end>
Predicted translation: я могу вам позвонить ? <end>
```

In [72]:

```
translate('I recommend you …') # Я рекомендую вам …
```

```
Input: <start> i recommend you <end>
Predicted translation: я надеюсь тебя . <end>
```

In [73]:

```
# Как странно: я не знал его,Хоть мы – друзья.Расположенья своего Не выдал я.
translate('Strange that I did not know him then,That friend of mine!I did not even show him
```

```
Input: <start> strange that i did not know him then , that friend of mine !
i did not even show him then one friendly sign <end>
Predicted translation: я не сделал е никто не сделал е никто не сделал е ник
то не сделал
```

Вывод

я обучал модель 8 эпох, даже на столь небольшм обучении модель показала вполне приемлимые результаты, конечно есть к чему стремиться дальше