

# 基于seq2seq with attention model 的英中翻译

---

## 基于seq2seq with attention model 的英中翻译

- 1.项目背景
- 2.解决思路
  - 2.1 问题分析
  - 2.2 seq2seq with attention详解
    - 2.2.1 我们为什么需要seq2seq
    - 2.2.2 什么是seq2seq
    - 2.2.3 带有注意力机制的seq2seq
    - 2.2.4 seq2seq with attention网络全貌
- 3.项目实施
  - 3.1 数据处理
    - 3.1.1数据集准备
    - 3.1.2 提取文本
    - 3.1.3 分词
    - 3.1.4 提取中英文
    - 3.1.5文本序列化
    - 3.1.6加载清理好的数据集
    - 3.1.7划分训练集与验证集
    - 3.1.8设置训练结构
    - 3.1.9处理dataset
  - 3.2网络结构
    - 3.2.1 Encoder层
    - 3.2.2 Attention层
    - 3.2.3 Decoder层
  - 3.3 优化器与损失函数
  - 3.4训练与评价
  - 3.5 训练结果测试

## 1.项目背景

---

语言是人类沟通的重要手段，有时候在阅读外语或者听外语时，可能会存在一些因语言不通而造成的理解问题。

在观看外语视频的时候，某些视频因为缺少字幕，而造成了理解上的困难。为此，我希望写一个视频的语音识别+机器翻译的程序来实现再看外文视频时能够自动加中文字幕的功能。

而机器翻译就是其中的一环，我的机器翻译学习从这里开始。

就目前而言，最新的主流机器翻译方案就是transform和bert. 模型的发展是由基本的seq2seq、seq2seq with attention、transform、bert 这四个发展阶段，他们的架构越来越复杂，参数数量也越来越多，但是这并不代表任何情况下后面的复杂的模型都比前面的简单的模型好，因为这是需要数据做为支撑，如果本身数据量很小，而去选择参数非常多的模型，也很难发挥其模型的实力。同时，后面的模型也是由前面的一步改进而来，而非一步到位的，学习、理解前面的简单的模型，对于学习理解后面的模型也有很大的帮助。

## 2.解决思路

### 2.1 问题分析

机器翻译就是从文本到文本的一种典型的模型，我们接受一段文本，通过一个模型，产生输出一段文本，这个模型理论上来说可以是任意的，但是普通的RNN网络具有不能处理输入输出长度不相等的问题，而另一种RNN的变种:seq2seq model就可以用来解决这个问题。

这里我们采用seq2seq with attention的模型。

### 2.2 seq2seq with attention详解

#### 2.2.1 我们为什么需要seq2seq

下图是传统的RNN结构

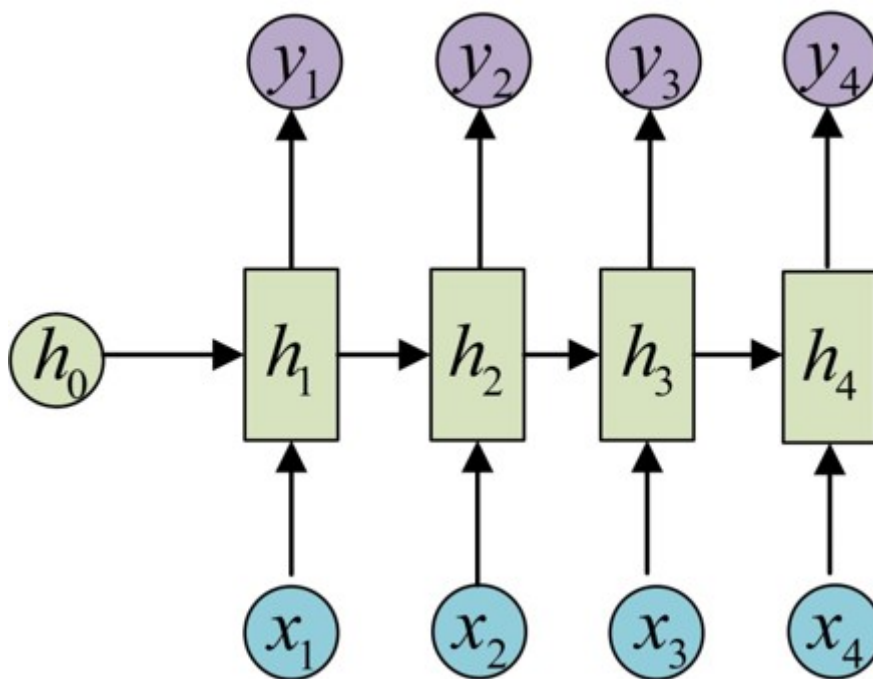


图1.传统RNN

其中

$$h_i = f(Ux_i + Wh_{i-1} + b)$$

$$y_i = \text{softmax}(Vh_i + c)$$

其参数U,W,V都是共享的

我们可以看到， $x_i$ 与 $y_i$ 是一一对应的，也就是说，传统的RNN的输入与输出必须是等长的,即N to N。

但这在一些条件下并不符合要求，比如在接下来的机器翻译中，源语言与目标语言的句子长度往往不相等，那么我们就需要一种输入与输出长度可以不相等的网络结构。

Seq2seq就是为解决问题而诞生的。

## 2.2.2 什么是seq2seq

简单来看，seq2seq的大体框架就是下面这样

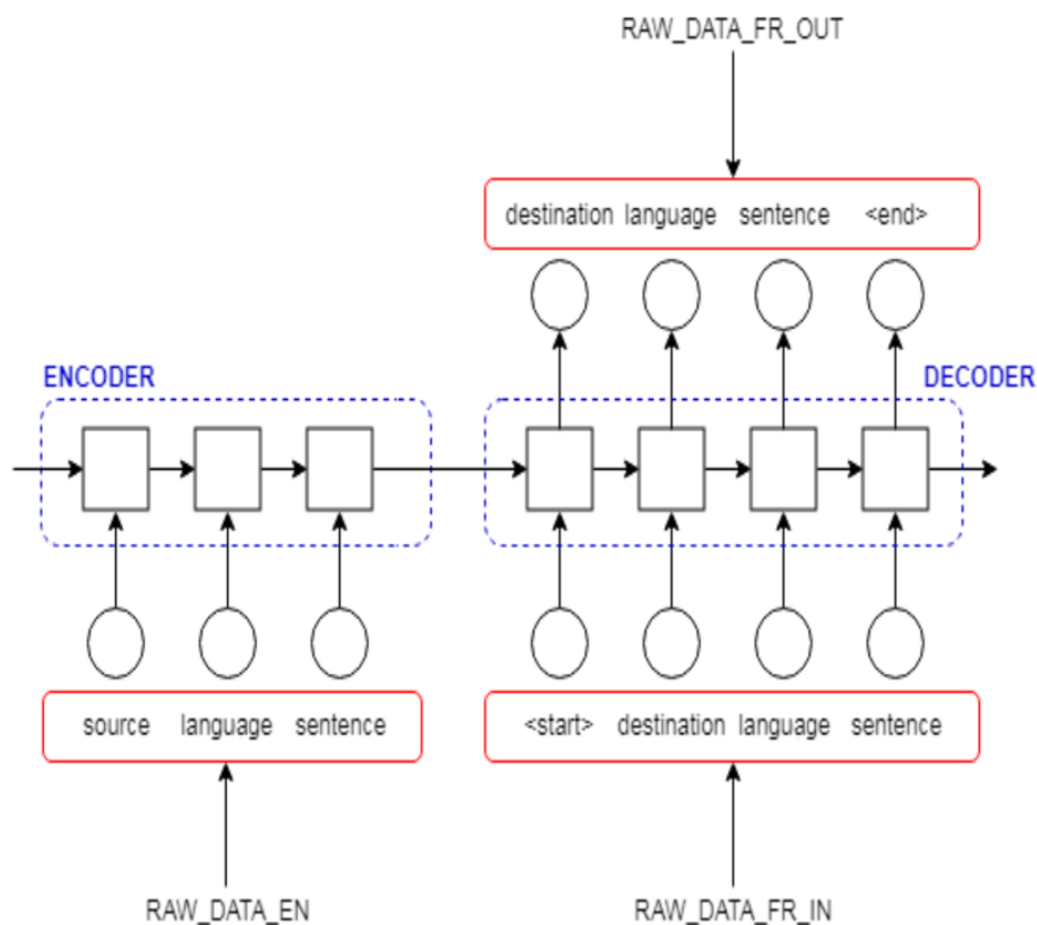


图2.seq2seq大体框架

下图展示了基本的seq2seq网络结构；

**图解**

### Seq2Seq Attention

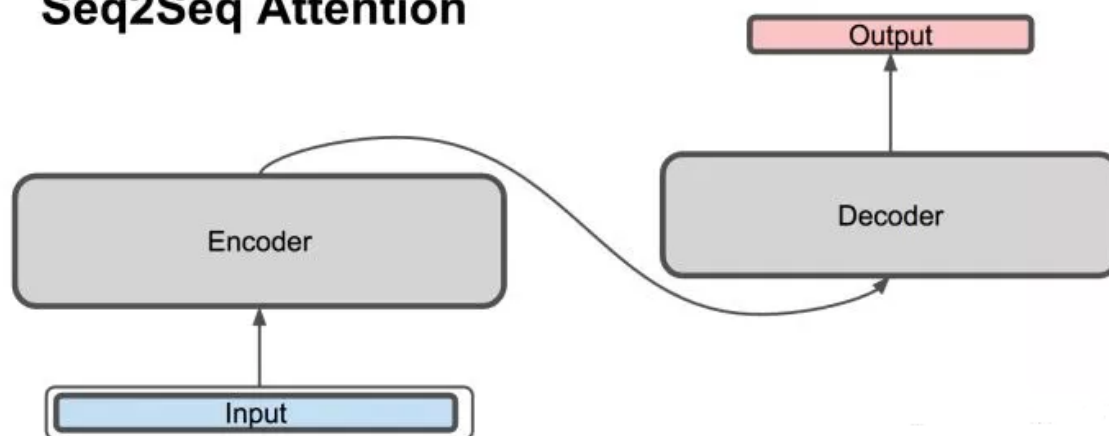


图3.seq2seq基本结构

实际上，seq2seq是RNN的一种变种网络，它是由两个RNN(左边的encoder和右边的decoder都是一个RNN)拼接而成。通过encode和decode网络来实现不定长序列之间的映射。实际实现中，RNN一般选用LSTM或者是GRU网络来一定程度上减缓长期记忆遗失的问题，但是这还不够，这也是后面attention机制被应用在seq2seq网络的原因。

虽然两个RNN各自的输入输出仍是相同。我们可以先通过encode网络生成input的语义向量，作为h0输入到decode中。

Decode网络的输入的长度即为整个网络的输出，与网络的输入的长度无关，这样就可以有一个长度的input到另一个长度的output的映射关系。就实现了N to M的映射。

我们可以看到，encode的隐藏层的输出都被丢掉，只有最后一层的输出被传给了decode，随着反复的向前传播、反向传播，input对output的影响会越来越小。这也是seq2seq网络的一个问题。

事实上，正因为Seq2seq是两个RNN的拼接，因此，基本的seq2seq也存在着基本RNN存在的问题----长期记忆的遗失（类似于梯度爆炸或消失）。虽然在RNN中也有LSTM或GRU的解决思路，但是效果还不够好，于是，有人提出了带有attention机制的seq2seq网络解决这一问题。

### 2.2.3 带有注意力机制的seq2seq

下图展示了带有注意力机制的seq2seq网络结构



图4.摘自 Effective Approaches to Attention-based Neural Machine Translation

该图摘自2015年提出了Luong Attention的论文**Effective Approaches to Attention-based Neural Machine Translation**，这是作者用于说明该网络的一个示例。

但是我们直接看这个图，恐怕还是有点难看懂。因此，在看这个图之前，我需要说明一些预备知识。比较重要的内容有两个：

1.The alignment vector(即图中的 attention weights)

$$\begin{aligned} a_t(s) &= \text{align}(h_t, \bar{h}_s) \\ &= \frac{\exp(\text{score}(h_t, \bar{h}_s))}{\sum_{s'} \exp(\text{score}(h_t, \bar{h}_{s'}))} \end{aligned}$$

## Equation 1: Equation for alignment vector

图5.The Alignment Vector

The alignment vector是一个与输入源序列相同长度的的向量，它是在decoder的每个时间步里计算而来。

这个向量的每个值就是对应的源输入序列的score(或者说是可能性)。

那么alignment是如何产生的？

我们看下面两个计算公式

对于 $a_t(s)$ 的计算，我相信如果比较敏感的话，就能一下子看出来，这实际上就是对score求softmax，也很好理解，就是将得分(权重)最高的选出来。

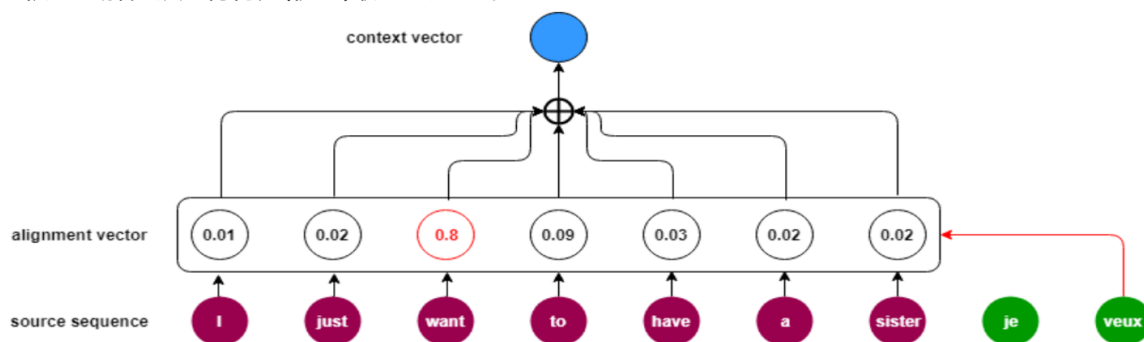


图6.The Alignment Vector计算方式

下图是Luong给出的三种计算score的方式。实际上除了Luong的attention之外，还有另一种attention: Bahdanau attention，不过他们大体上是很相似的，其中一处差别就是，Bahdanau attention 提倡只使用concat计算score而不使用其他的计算方式。

$$\text{score}(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & \text{dot} \\ h_t^\top W_a \bar{h}_s & \text{general} \\ v_a^\top \tanh(W_a [h_t; \bar{h}_s]) & \text{concat} \end{cases}$$

Equation 2: Score functions

图7.score计算方式

## 2.The context vector(上下文向量)

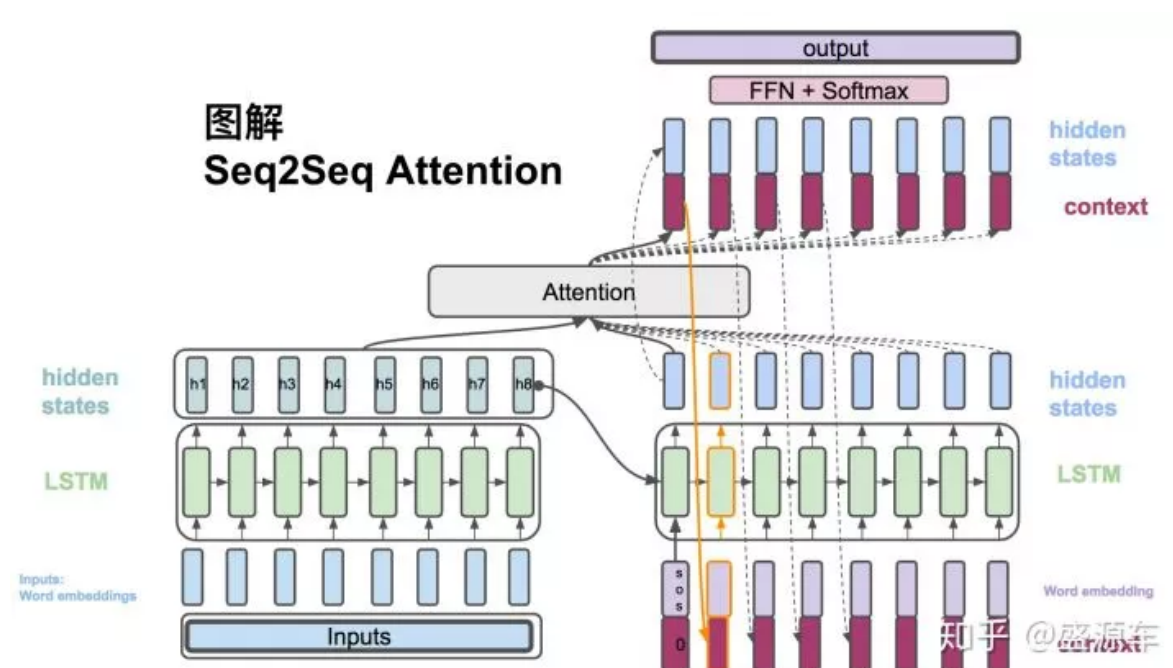


图8.The context vector

The context vector 是我们用来计算最后的decoder的输出，具体怎样计算，我会在后面给出说明。

它是encoder网络输出结果的加权平均。

其具体计算的方式即为 encode的输出与alignment vector做点积运算，即可得到context vector

$$c_i = \sum_{j=1}^{T_x} a_{ij} h_j$$

以上两个部分就是attention机制的核心，使用attention，简单来说，就是为了得到这个context vector。

当然如果没有attention，也可以认为有context vector，但是那个context vector只是基于ecoder的最后一个hidden state而与其他无关，同时这个context vector都是相同的，这代表所有的输入对于输出的影响是相同的，我们可以说这是没有注意到某些关键的元素的，效果并不好。attention则是通过加权平均，给予每个输入不同的权重，让机器从数据中学习，来获取某些关键的点，最后获得一个相对比较好的语义抽象--context vector。

## 2.2.4 seq2seq with attention网络全貌

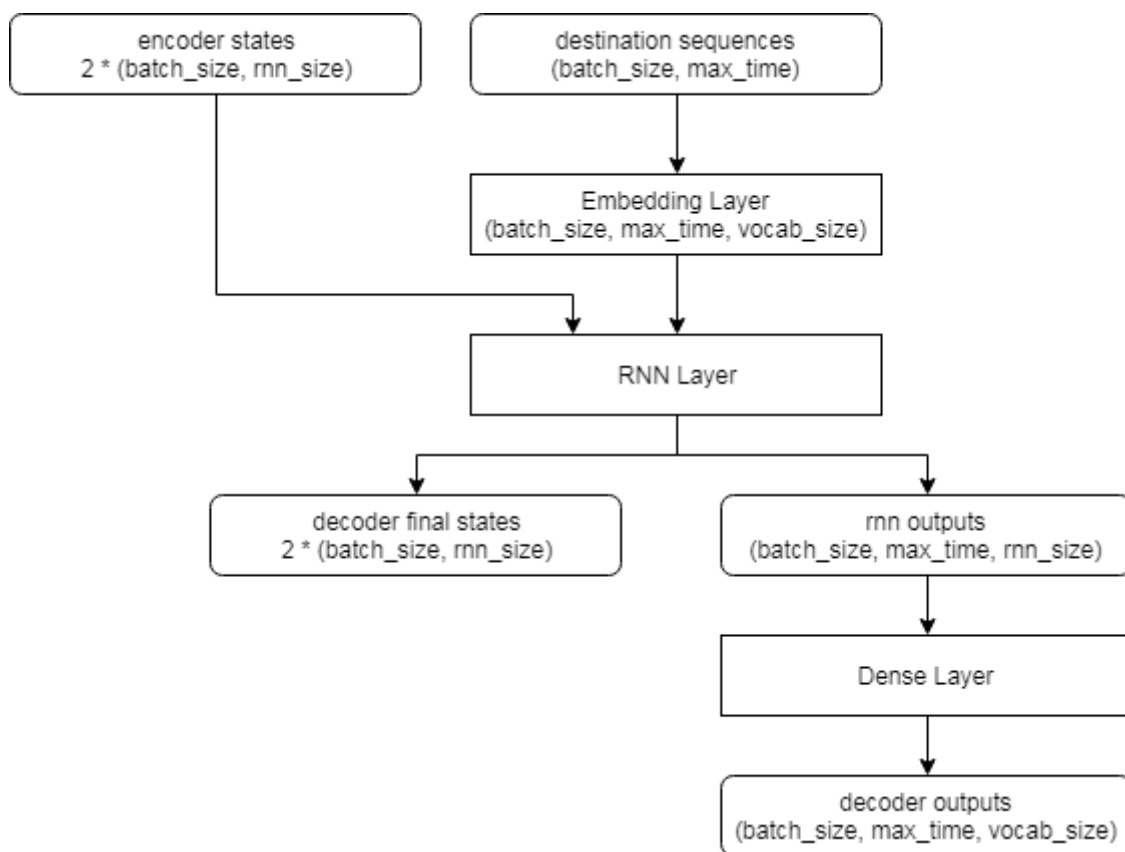


图9.seq2seq\_attention 整体架构

左侧为Encoder，右侧为Decoder，中间为Attention。

Encoder一般是由一层embedding层和多层RNN(一般选用LSTM或GRU)层组成，Decoder层也同样如此。

一个时间步的流程如下：

1. 从左边Encoder开始，输入转换为word embedding, 进入LSTM。LSTM会在每一个时间点上输出hidden states。如图中的h1,h2,...,h8。
2. 接下来进入右侧Decoder，输入为中文的句子，以及从encoder最后一个hidden state: h8。LSTM的输出是一个hidden state (cell state不需要使用)。

3. 随后，Decoder的hidden state与Encoder所有的hidden states作为输入，放入Attention模块开始计算一个context vector，就是在2.2.3节我们提到的计算方法。

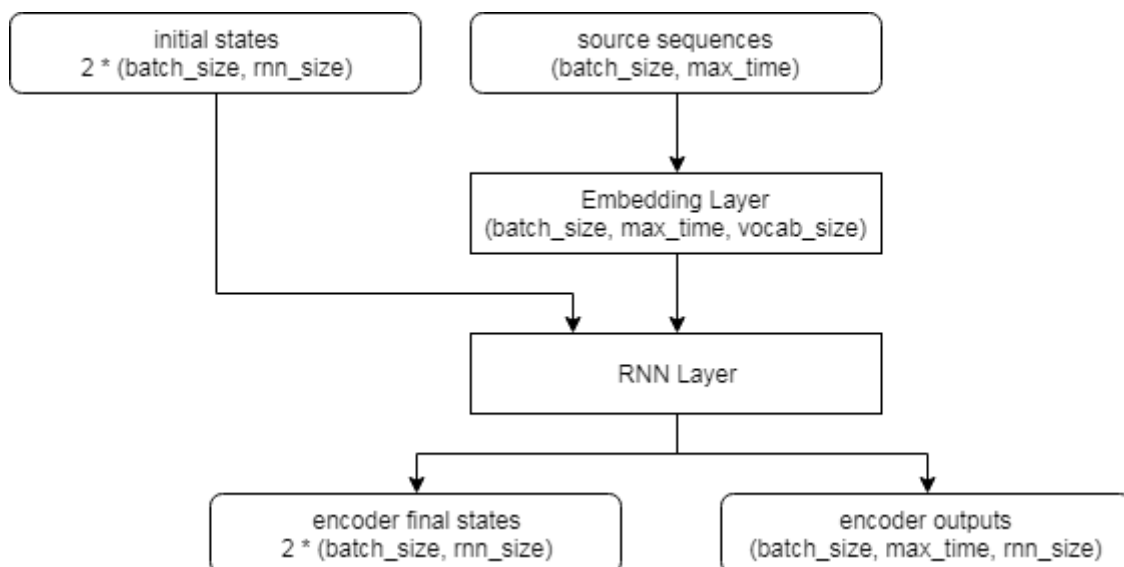


图10.seq2seq.attention 第2个时间步

我们来到第2个时间步:之前的context vector可以作为输入和目标的单词串起来作为RNN（即LSTM）的输入。之后又回到一个hidden state。以此循环。也就是说，decoder每走过一时间步，就会在context vector产生一个值，并会与目标串连接作为下一个时间步decoder的输入，直到走到最后的时间步。

走完了所有的时间步后，context也同步地更新完成。这时候， $\hat{s}_t = \tanh(W_c[c_t; s_t])$ ，即将context vector和decoder的hidden states连接；通过通过 $p(y_t | y < t, x) = \text{softmax}(W_s \hat{s}_t)$ 来计算最后的输出概率

## 3.项目实现

### 3.1 数据处理

#### 3.1.1数据集准备

[点击这里下载数据集](#)

这里我们选用的是'<http://www.manythings.org/anki/>'上的中英翻译的数据集，这个数据集是比较小的，只有20000条的数据，文件不到3MB，但是我们这次主要还是为了学习，所以并不是特别看重这个效果，要知道，谷歌训练自己的谷歌翻译的数据集都是TB级别的。

#### 3.1.2 提取文本

```
1 def loadFile(filename=path_to_file):
2     with open(filename, 'r', encoding='utf8') as f:
3         raw_data = f.readlines()
4     return raw_data
```

我们来看一下数据的格式是怎样的:

```
1 print(raw_data[:10])
```



输出结果:

```
1 ['Hi.\t嗨.\n', 'Hi.\t你好.\n', 'Run.\t你用跑的.\n', 'wait!\t等等!\n',  
  'Hello!\t你好.\n', 'I try.\t让我来.\n', 'I won!\t我赢了.\n', 'Oh no!\t不会吧.\n',  
  'Cheers!\t乾杯!\n', 'He ran.\t他跑了.\n']
```

### 3.1.3 分词

我们要对英文和中文进行分词，如果是英文，不需要特别处理，直接按照空格分割即可；对于中文，我们通过分词工具jieba来实现。那么如何判断一个句子是英文还是中文？我们可以通过Unicode编码来判断，中文的Unicode编码是在4e00到9fff之间的，如果句子中出现了这个编码区间内的字符，即认为该句子是中文，并采用中文的处理方式。

```
1 def check_contain_chinese(w):  
2     flag = True  
3     flagtop=False  
4     for check_str in w:  
5         for ch in check_str:  
6             if u'\u4e00' >= ch or ch >= u'\u9fff':  
7                 flag = False  
8                 flagtop=False  
9                 break  
10            if not flagtop :  
11                break  
12        return flag  
13  
14 def preprocess_chinese(w):  
15     line=jieba.cut(w)  
16     w=[x for x in line]  
17     item=' '.join(w)  
18     return item  
19  
20 def preprocess_sentence(w):  
21     w = re.sub(r"([?.!,;])", r" \1 ", w)#在单词与跟在其后的标点符号之间插入一个空格  
22     w = re.sub(r'[" "]+' , " ", w)#多个空格合并为一个空格  
23     if check_contain_chinese(w):  
24         w=preprocess_chinese(w)  
25     else:  
26         w = w.lower().strip()#因为大小写是不影响含义的，我们不能说大写的ME与小写的me  
27         #意思不同，                                #因此全部转化为小写  
28         w = '<start> ' + w + ' <end>'
```

### 3.1.4 提取中英文

```
1 def create_dataset(path, num_examples):  
2     lines = io.open(path, encoding='UTF-8').read().strip().split('\n')  
3  
4     word_pairs = [[preprocess_sentence(w) for w in l.split('\t')] for l in  
5                     lines[:num_examples]]  
6  
7     return zip(*word_pairs)
```

我们看一下现在English和Chinese的最后一个结果是怎样的:



```

1 en,ch = create_dataset(path_to_file, None)
2 print(en[-1])
3 print(ch[-1])

```

输出:

```

1 <start> if a person has not had a chance to acquire his target language by
  the time he's an adult , he's unlikely to be able to reach native speaker
  level in that language . <end>
2 <start> 如果 一個 人 在 成人 前 沒 有 機會 習 得 目標 語言 , 他 對 該 語言 的 認識 達
  到 母語者 程度 的 機會 是 相當 小 的 。 <end>

```

### 3.1.5文本序列化

我们通过TensorFlow.keras中的Tokenizer类来帮助我们完成这一工作, 这里 `fit_on_texts` 用来对输入的文本产生一个字典, 这个字典是一个文字到整数的映射; `texts_to_sequences` 将输入的文本按照刚刚生成的字典将文本的文字转化为一个个数字, 生成整数的tensor; `pad_sequences` 这里的 `padding='post'` 是让刚刚产生的所有tensor扩展为相同的长度(即原本最长的tensor的长度)

```

1 def tokenize(lang):
2     lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
3         filters='')
4     lang_tokenizer.fit_on_texts(lang)
5
6     tensor = lang_tokenizer.texts_to_sequences(lang)
7
8     tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
9                                                             padding='post')
10
11     return tensor, lang_tokenizer

```

### 3.1.6加载清理好的数据集

```

1 def load_dataset(path, num_examples=None):
2     # 创建清理过的输入输出对
3     inp_lang, targ_lang = create_dataset(path, num_examples)
4
5     input_tensor, inp_lang_tokenizer = tokenize(inp_lang)
6     target_tensor, targ_lang_tokenizer = tokenize(targ_lang)
7
8     return input_tensor, target_tensor, inp_lang_tokenizer,
9         targ_lang_tokenizer

```

```

1 # 尝试实验不同大小的数据集
2 num_examples = 20133
3 input_tensor, target_tensor, inp_lang, targ_lang = load_dataset(path_to_file,
4 num_examples)
5
6 # 计算目标张量的最大长度 (max_length)
7 max_length_targ, max_length_inp = max_length(target_tensor),
8 max_length(input_tensor)

```

### 3.1.7划分训练集与验证集

```
1 # 采用 80 - 20 的比例切分训练集和验证集
2 input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val
  = train_test_split(input_tensor, target_tensor, test_size=0.2)
```

### 3.1.8设置训练结构

这里可以修改每个buffer的大小、每个batch的大小、embedding层的维度，和输入的unit数量

```
1 BUFFER_SIZE = len(input_tensor_train)
2 BATCH_SIZE = 64
3 steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
4 embedding_dim = 256
5 units = 1024
6 vocab_inp_size = len(inp_lang.word_index)+1
7 vocab_tar_size = len(targ_lang.word_index)+1
```

### 3.1.9处理dataset

我们通过 `tf.data.Dataset` 创建TensorFlow的数据集格式，并打乱、按batch切分数据集

```
1 dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train,
  target_tensor_train)).shuffle(BUFFER_SIZE)
2 dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)
```

## 3.2网络结构

### 3.2.1 Encoder层

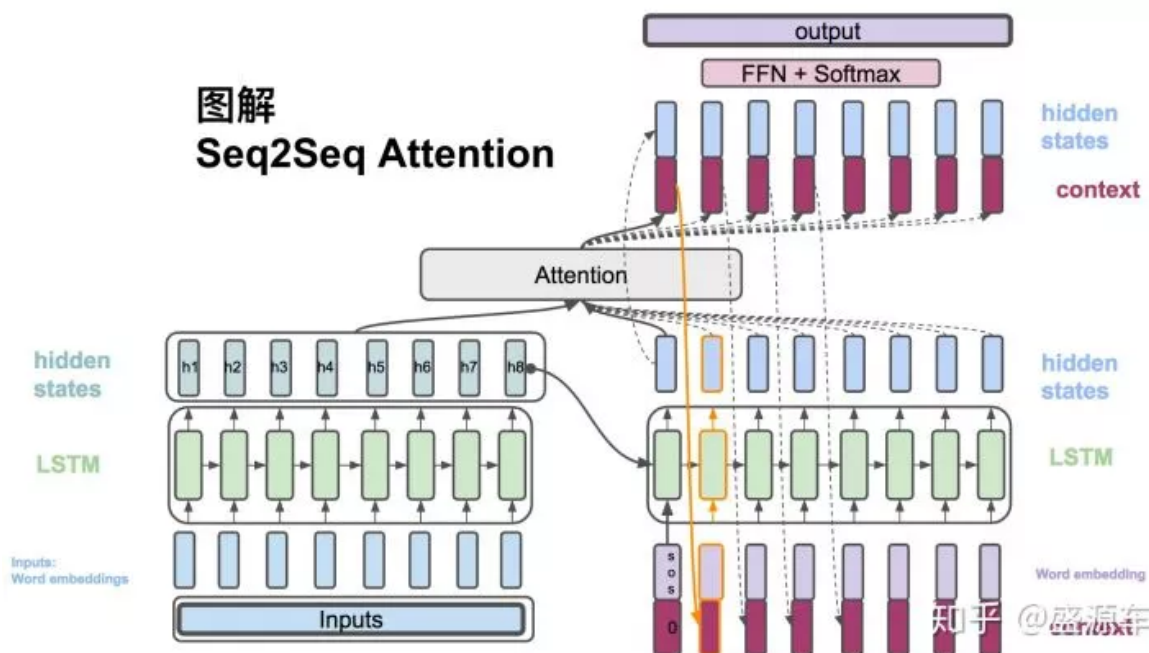


图11.encoder网络结构

Encoder由一层embedding层以及一层或多层RNN组成，这在前面已经有过说明。这里，为了减少运算量，我们的RNN选用一层GRU。

```
1 class Encoder(tf.keras.Model):
```

```

2     def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
3         super(Encoder, self).__init__()
4         self.batch_sz = batch_sz
5         self.enc_units = enc_units
6         self.embedding = tf.keras.layers.Embedding(vocab_size,
embedding_dim)
7         self.gru = tf.keras.layers.GRU(self.enc_units,
8                                         return_sequences=True,
9                                         return_state=True,
10
recurrent_initializer='glorot_uniform')
11
12     def call(self, x, hidden):
13         x = self.embedding(x)
14         output, state = self.gru(x, initial_state = hidden)
15         return output, state
16
17     def initialize_hidden_state(self):
18         return tf.zeros((self.batch_sz, self.enc_units))

```

### 3.2.2 Attention层

```

1 class BahdanauAttention(tf.keras.layers.Layer):
2     def __init__(self, units):
3         super(BahdanauAttention, self).__init__()
4         self.w1 = tf.keras.layers.Dense(units)
5         self.w2 = tf.keras.layers.Dense(units)
6         self.v = tf.keras.layers.Dense(1)
7
8     def call(self, query, values):
9         # 隐藏层的形状 == (批大小, 隐藏层大小)
10        # hidden_with_time_axis 的形状 == (批大小, 1, 隐藏层大小)
11        # 这样做是为了执行加法以计算分数
12        hidden_with_time_axis = tf.expand_dims(query, 1)
13
14        # 分数的形状 == (批大小, 最大长度, 1)
15        # 我们在最后一个轴上得到 1, 因为我们把分数应用于 self.v
16        # 在应用 self.v 之前, 张量的形状是 (批大小, 最大长度, 单位)
17        score = self.v(tf.nn.tanh(
18            self.w1(values) + self.w2(hidden_with_time_axis)))
19
20        # 注意力权重 (attention_weights) 的形状 == (批大小, 最大长度, 1)
21        attention_weights = tf.nn.softmax(score, axis=1)
22
23        # 上下文向量 (context_vector) 求和之后的形状 == (批大小, 隐藏层大小)
24        context_vector = attention_weights * values
25        context_vector = tf.reduce_sum(context_vector, axis=1)
26
27        return context_vector, attention_weights

```

我们创建attention层,设置输入单元为10

```

1 attention_layer = BahdanauAttention(10)
2 attention_result, attention_weights = attention_layer(sample_hidden,
sample_output)

```

### 3.2.3 Decoder层

```
: 1 translate(u" we like it")  
  
<generator object evaluate.<locals>.<genexpr> at 0x0000022102D390C8>  
Input: <start> we like it <end>  
Predicted translation: 我們 喜歡 它 。 <end>
```

图12.Decoder网络结构

一个Decoder层由一层embedding层、多层RNN(LSTM/GRU)、一个全连接层(dense)、attention层、一个softmax层组成。

```
1  
2 class Decoder(tf.keras.Model):  
3     def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):  
4         super(Decoder, self).__init__()  
5         self.batch_sz = batch_sz  
6         self.dec_units = dec_units  
7         self.embedding = tf.keras.layers.Embedding(vocab_size,  
embedding_dim)  
8         self.gru = tf.keras.layers.GRU(self.dec_units,  
9                                         return_sequences=True,  
10                                        return_state=True,  
11  
recurrent_initializer='glorot_uniform')  
12         self.fc = tf.keras.layers.Dense(vocab_size)  
13  
14         # 用于注意力  
15         self.attention = BahdanauAttention(self.dec_units)  
16  
17     def call(self, x, hidden, enc_output):  
18         # 编码器输出 (enc_output) 的形状 == (批大小, 最大长度, 隐藏层大小)  
19         context_vector, attention_weights = self.attention(hidden,  
enc_output)  
20  
21         # x 在通过嵌入层后的形状 == (批大小, 1, 嵌入维度)  
22         x = self.embedding(x)  
23  
24         # x 在拼接 (concatenation) 后的形状 == (批大小, 1, 嵌入维度 + 隐藏层大  
小)  
25         x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)  
26  
27         # 将合并后的向量传送到 GRU  
28         output, state = self.gru(x)  
29  
30         # 输出的形状 == (批大小 * 1, 隐藏层大小)  
31         output = tf.reshape(output, (-1, output.shape[2]))  
32  
33         # 输出的形状 == (批大小, vocab)  
34         x = self.fc(output)  
35  
36         return x, state, attention_weights
```

像前面一样，我们创建decoder层

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

至此，我们的网络结构的三个大组件，就像搭乐高积木那样，已经搭建完毕。

### 3.3 优化器与损失函数

这里我们直接通过tensorflow提供的接口，使用Adam优化器和均方差的损失函数

```
1 | optimizer = tf.keras.optimizers.Adam()
2 | loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
3 |     from_logits=True, reduction='none')
4 |
5 | def loss_function(real, pred):
6 |     mask = tf.math.logical_not(tf.math.equal(real, 0))
7 |     loss_ = loss_object(real, pred)
8 |
9 |     mask = tf.cast(mask, dtype=loss_.dtype)
10 |    loss_ *= mask
11 |
12 |    return tf.reduce_mean(loss_)
```

### 3.4 训练与评价

定义训练函数:基本就是将输入传入encoder和decoder，计算损失，梯度下降

在训练的过程中，我们使用了教师强制，你可能会问了，什么是教师强制？

“教师强制”的概念是使用实际目标输出作为每个下一个输入，而不是使用解码器的猜测作为下一个输入。使用教师强制会导致其收敛更快。用一种比较直观的方式解释的话，那就是，最开始的时候，网络还什么都不会，这时候第一个时间步训练时，它的输出可能都是错的，那么我们拿一个错的输出作为下一个输入，显然也难以得到正确的结果。那么我们通过使用“教师强制”，将正确答案作为输入而不是错的答案，就可以帮助这个网络更快地学到这些潜在的东西。但并不是任何时候都要用“教师强制”，这里一篇文章表明，当使用受过训练的网络时，[可能会显示不稳定](#)。

概括一下，就是对于完全没有训练过的网络，使用“教师强制”会帮助我们加快训练速度，但是如果这个网络已经训练地比较多了，那么我们就该不再使用“教师强制”。

```
1 |
2 | @tf.function
3 | def train_step(inp, targ, enc_hidden):
4 |     loss = 0
5 |
6 |     with tf.GradientTape() as tape:
7 |         enc_output, enc_hidden = encoder(inp, enc_hidden)
8 |
9 |         dec_hidden = enc_hidden
10 |
11 |         dec_input = tf.expand_dims([targ_lang.word_index['<start>']], *
12 |                                     BATCH_SIZE, 1)
13 |
14 |         # 教师强制 - 将目标词作为下一个输入
15 |         for t in range(1, targ.shape[1]):
16 |             # 将编码器输出 (enc_output) 传送至解码器
```

```

16         predictions, dec_hidden, _ = decoder(dec_input, dec_hidden,
enc_output)
17
18         loss += loss_function(targ[:, t], predictions)
19
20         # 使用教师强制
21         dec_input = tf.expand_dims(targ[:, t], 1)
22
23     batch_loss = (loss / int(targ.shape[1]))
24
25     variables = encoder.trainable_variables + decoder.trainable_variables
26
27     gradients = tape.gradient(loss, variables)
28
29     optimizer.apply_gradients(zip(gradients, variables))
30
31     return batch_loss
32
33

```

我们训练10个epoch，每2个epoch保存一次模型，这样在下次训练时无需从头开始，接着上一次训练时保存的epoch接着训练即可。

```

1
2  EPOCHS = 10
3
4  for epoch in range(EPOCHS):
5      start = time.time()
6
7      enc_hidden = encoder.initialize_hidden_state()
8      total_loss = 0
9
10     for (batch, (inp, targ)) in
tqdm(enumerate(dataset.take(steps_per_epoch))):
11         batch_loss = train_step(inp, targ, enc_hidden)
12         total_loss += batch_loss
13
14         if batch % 100 == 0:
15             print('Epoch {} Batch {} Loss {:.4f}'.format(epoch + 1,
16                                                             batch,
17                                                             batch_loss.numpy()))
18
19     # 每 2 个周期 (epoch)，保存 (检查点) 一次模型
20     if (epoch + 1) % 2 == 0:
21         checkpoint.save(file_prefix = checkpoint_prefix)
22
23     print('Epoch {} Loss {:.4f}'.format(epoch + 1,
24                                         total_loss / steps_per_epoch))
25     print('Time taken for 1 epoch {} sec\n'.format(time.time() - start))
26

```

评价函数基本上与训练是一致的，只是这里不需要使用教师强制

```

1  def evaluate(sentence):
2      attention_plot = np.zeros((max_length_targ, max_length_inp))
3

```

```

4     sentence = preprocess_sentence(sentence)
5     print(i for i in sentence.split(" "))
6     inputs = [inp_lang.word_index[i]
7               for i in sentence.split(' ')]
8     inputs = tf.keras.preprocessing.sequence.pad_sequences([inputs],
9
10    maxlen=max_length_inp,
11
12
13    padding='post')
14
15    inputs = tf.convert_to_tensor(inputs)
16
17    result = ''
18
19    hidden = [tf.zeros((1, units))]
20    enc_out, enc_hidden = encoder(inputs, hidden)
21
22    dec_hidden = enc_hidden
23    dec_input = tf.expand_dims([targ_lang.word_index['<start>']], 0)
24
25    for t in range(max_length_targ):
26        predictions, dec_hidden, attention_weights = decoder(dec_input,
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2660
2661
2662
2663
2664

```



```
: 1 translate(u" i need you")  
  
<generator object evaluate.<locals>.<genexpr> at 0x000002214C87E7C8>  
Input: <start> i need you <end>  
Predicted translation: 我 需要 你 。 <end>
```

图13.正确的测试-1

```
1 translate(u" be quiet")  
  
<generator object evaluate.<locals>.<genexpr> at 0x000002210092B548>  
Input: <start> be quiet <end>  
Predicted translation: 保持 安静 ！ <end>
```

图14.正确的测试-2

```
1 translate(u" i will go there")  
  
<generator object evaluate.<locals>.<genexpr> at 0x000002210092B548>  
Input: <start> i will go there <end>  
Predicted translation: 我 明天 要 去 那裡 。 <end>
```

图15.正确的测试-3

```
1 translate(u"we")  
  
<generator object evaluate.<locals>.<genexpr> at 0x000002214F157A48>  
Input: <start> we <end>  
Predicted translation: 我們 很 合作 。 <end>
```

图16.正确的测试-4

```
1 translate(u" we believe we will make a better future")  
  
<generator object evaluate.<locals>.<genexpr> at 0x0000022153821A48>  
Input: <start> we believe we will make a better future <end>  
Predicted translation: 我們 不想 再 跟 我們 不高興 。 <end>
```

图17.错误的测试-1

```
1 translate(u"please be quiet")  
  
<generator object evaluate.<locals>.<genexpr> at 0x000002210092B548>  
Input: <start> please be quiet <end>  
Predicted translation: 请 敲门 。 <end>
```

图18.错误的测试-2

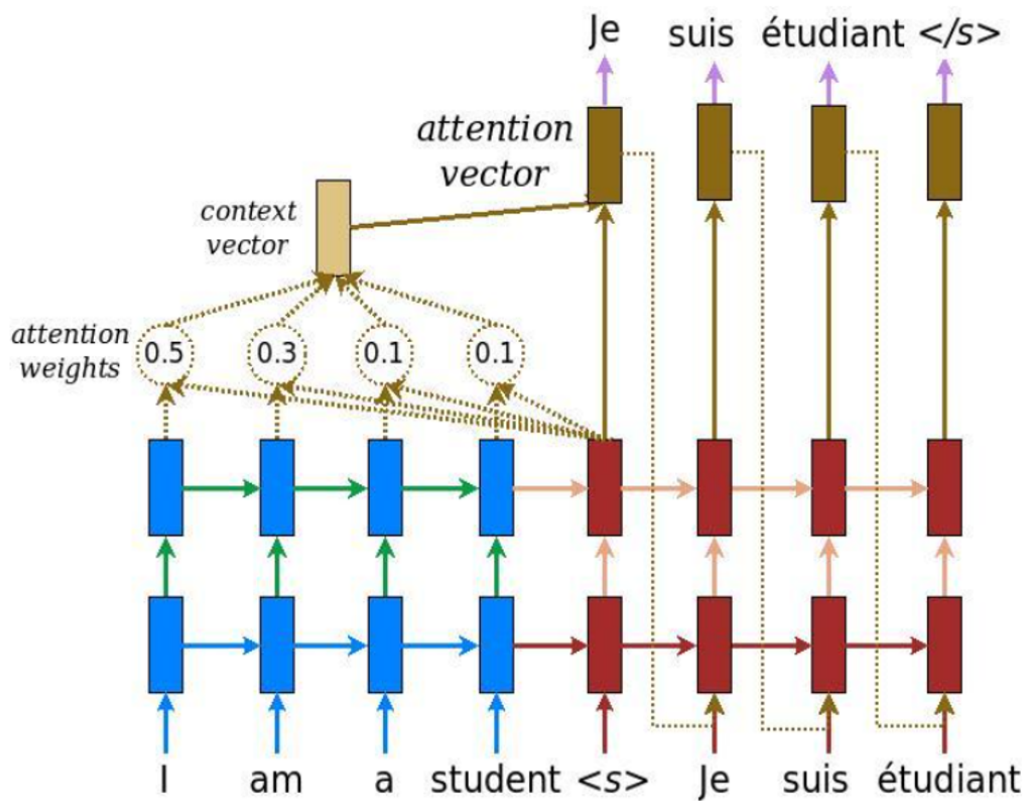


图19.错误的测试-3

我们可以看到，对于一些比较简单的、不是太长也不是只有单个词的时候，还是有一定的翻译效果，这说明其也确实学到了一些东西；但是显然，这个还不够准确，对于单个的词和比较长的句子的翻译效果很差。不过，我认为这也与我使用的数据集很小有关。

以上，就是本次 seq2seq with attention 英中翻译的全部内容。