# 机器翻译

## 1任务

用神经网络方法实现 中-->英 翻译系统。

### 1.1模型要求：

（1）RNN + Attention 模型

（2）Transformer 模型

任选其中一种模型。

### 1.2数据及平台：

( 1）数据集：/data，6834个中英平行语对。/cn.txt和/en.txt的每一行一一对应

( 2）模型实现可用平台Tensorflow 或 Pytorch

### 1.3提交内容：

( 1）整个工程项目：数据处理代码，模型核心代码，训练得到的模型文件

( 2）项目说明文件：模型参数设置，模型测试详细过程

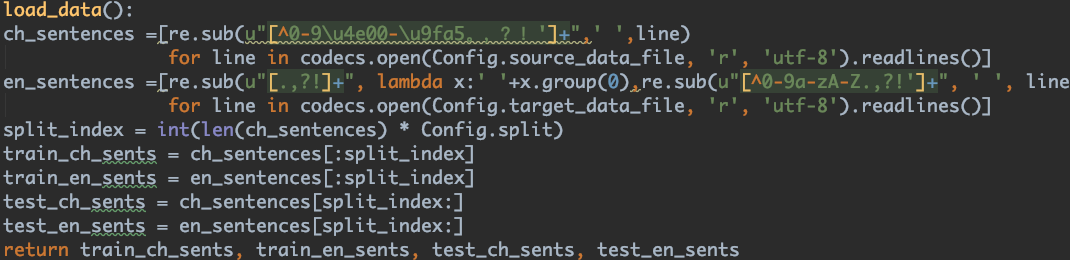
**Note**：本次作业实现了以上提到的两种模型，并对模型加以比较。

## 2 Transformer

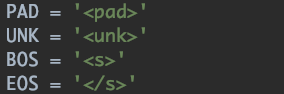
### 2.1数据处理

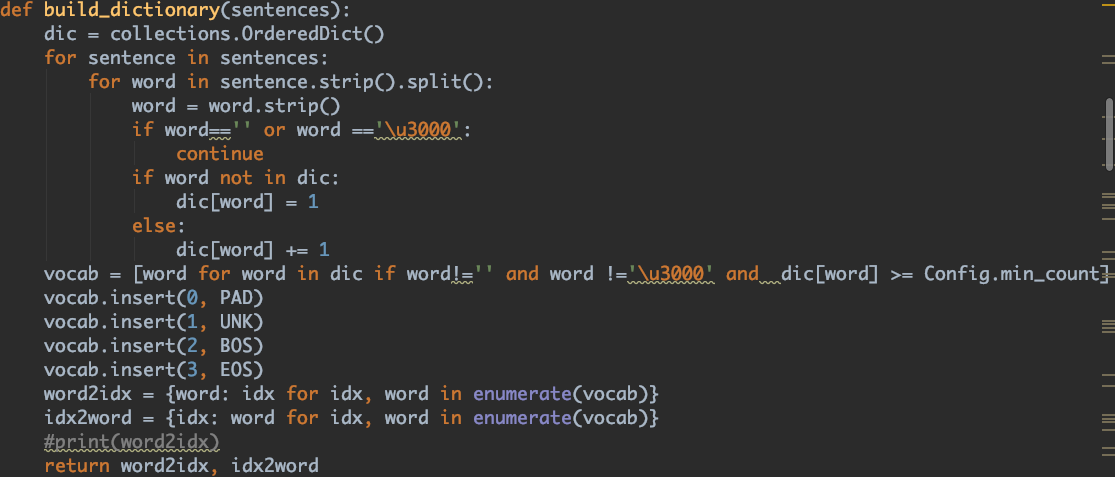
数据处理部分主要完成以下几个部分：数据读取、构造字典、构造词向量、构造batch等。

数据读取部分：读取文件，去除一些罕见的符号（只保留[，。？！,.?!]以及数字），然后按照设置的数据集划分比例划分出训练集和测试集。

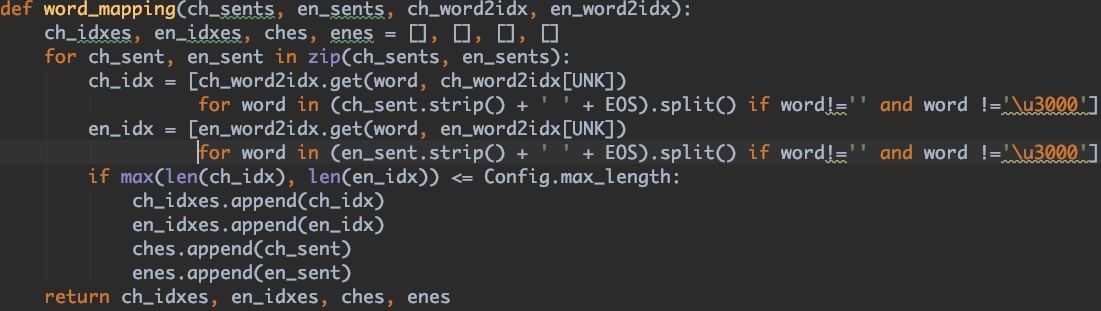


字典构造：只对词频数>阈值的词 设置字典。然后在词典中插入以下四个符号。

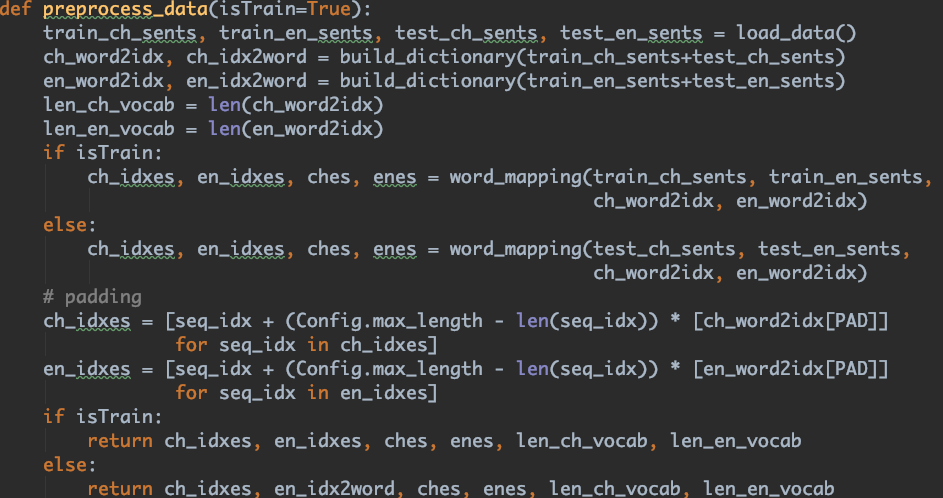




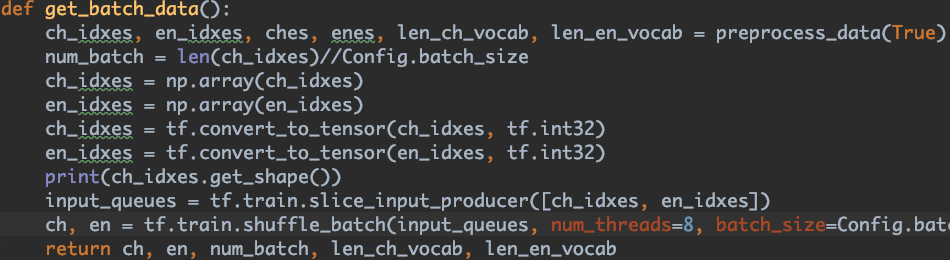
词向量构造：对每个词做映射，获得向量表示，此处只选择那些句子长度小于设定的长度阈值的数据集作为最后的训练集或测试集。



获取数据：这部分是与模型直接交互，用于获取最后处理好的训练集源语句、词向量表示、字典大小等。



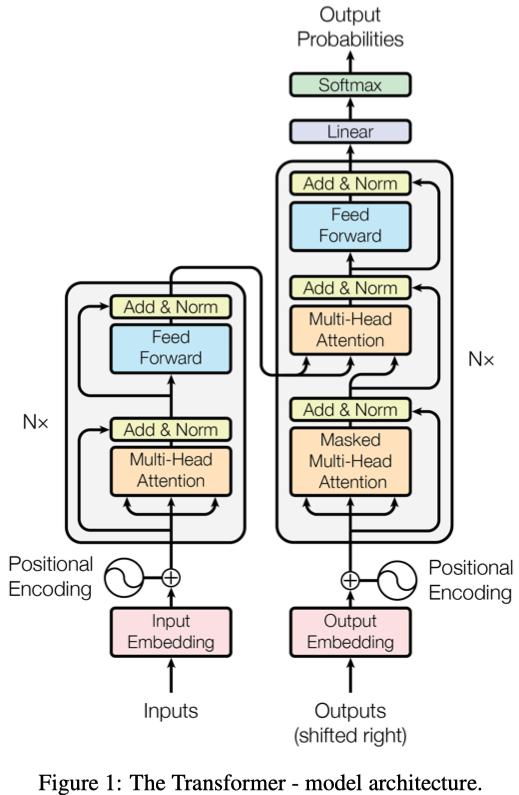
获取batch：使用TensorFlow提供的工具来获取一个batch queue。

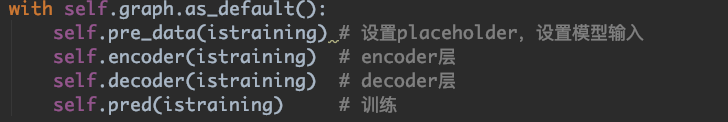


### 2.2模型核心部分

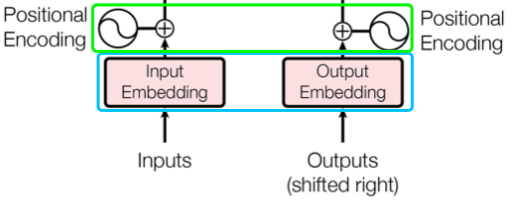
#### 2.2.1总体架构

该模型架构如下所示，代码中采用模块化设计。

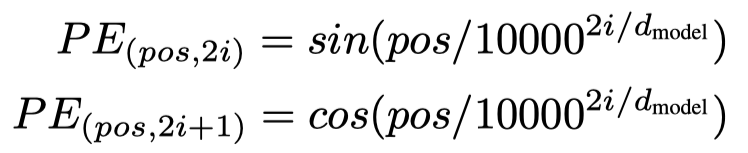


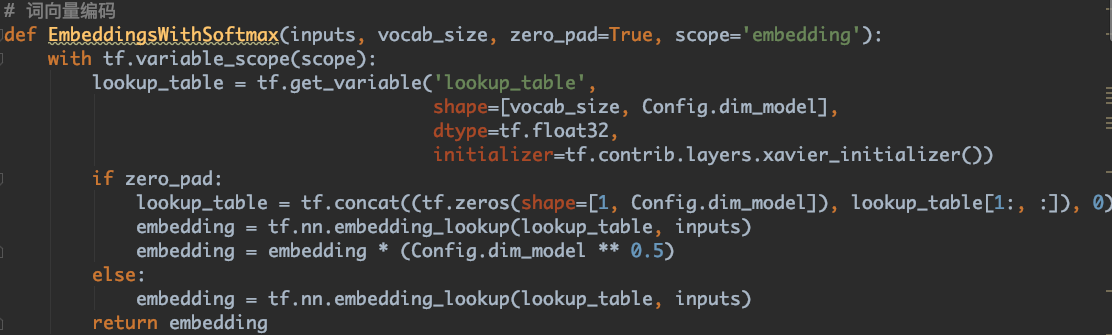


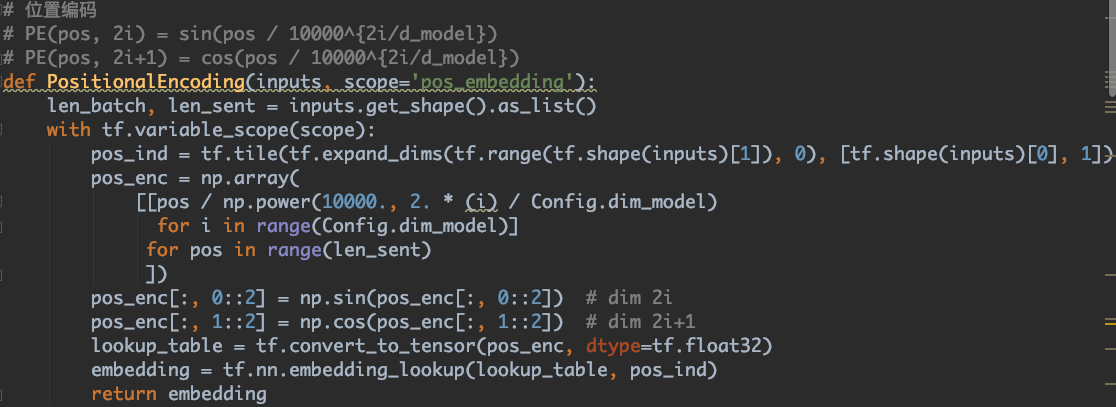
#### 2.2.1词向量输入层：



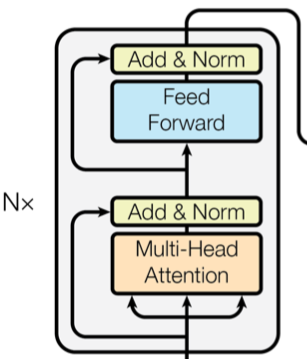
主要处理encoder和decoder的输入词向量、位置词向量。位置词向量可以采用固定的计算方法，如下所示；也可以直接采用随机生成词向量，一同参与训练。



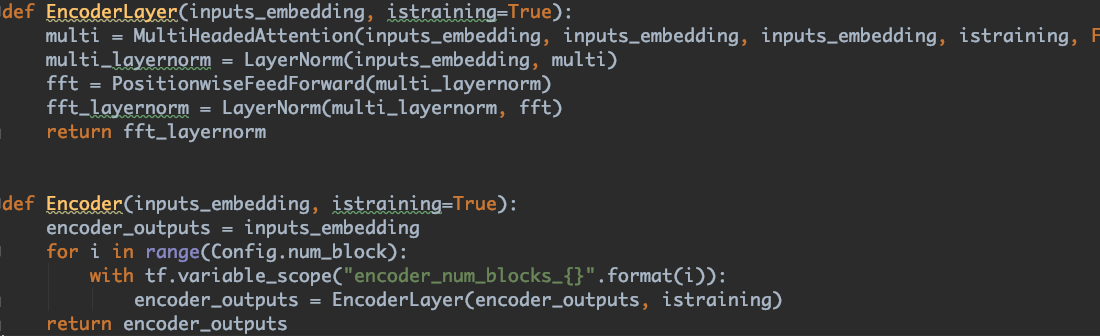




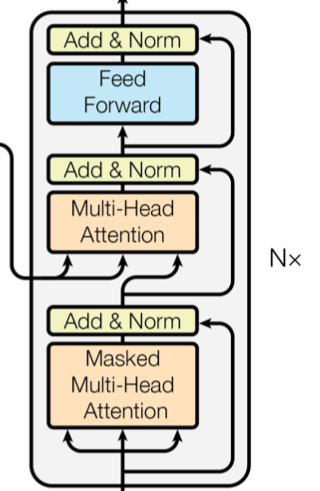
#### 2.2.2 encoder层



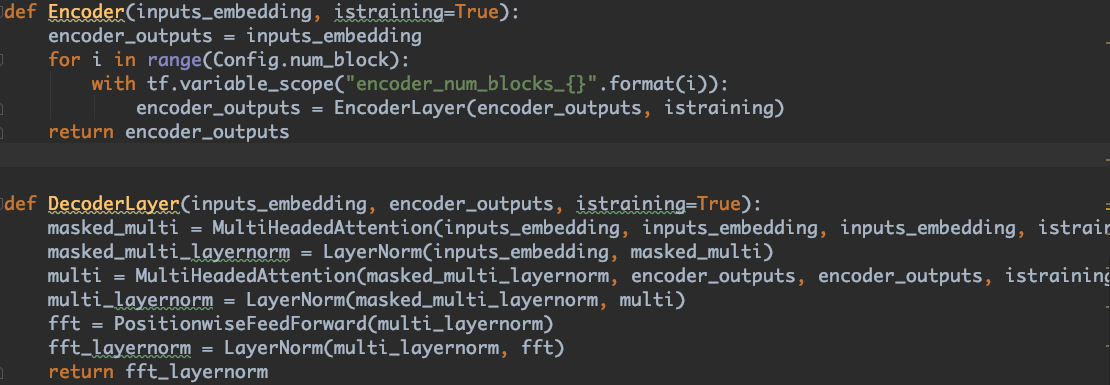
原作者采用6个完全相同设计的单个encoder block形成的stack来组成一个完整的encoder，单层encoder都由两个子层组成：multi-head self-attention机制和positionwise 全连接前馈网络组成。两个子层最后都运用了residual connection并做了layer normalization。



#### 2.2.3 decoder层

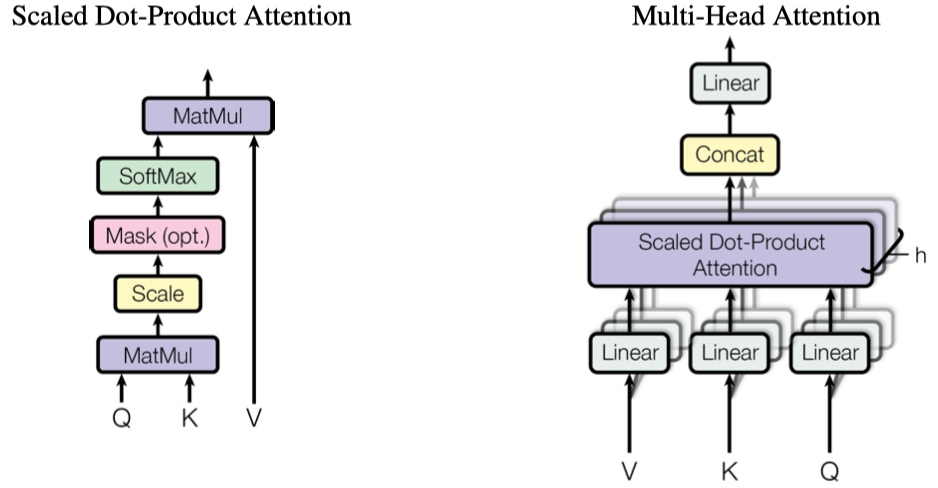


decoder子层和encoder子层设计基本相似，只是多了一个masked multi-head self-attention。这主要是为了保证当前预测所依赖的知识只是它已经预测出的知识，对句子后面的知识目前是不清楚的。此外，此处的输入来自两部分：自身的输出值以及encoder层的输出值。

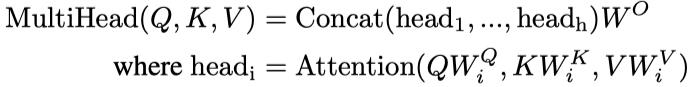


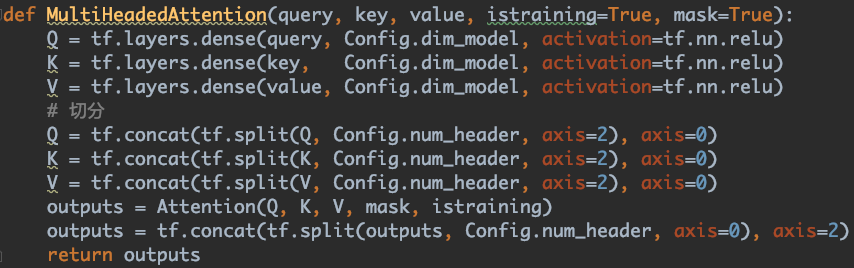
#### 2.2.4 Multi-Head Attention子层

这部分是精髓所在，主要架构如下所示：

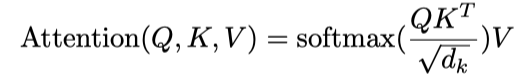


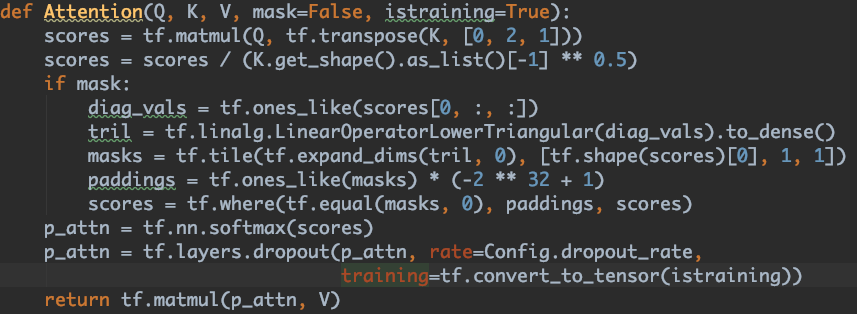
采用Multi-Head Attention，分成多个header分别计算，最后训练处多个具有不同倚重的特征，能有更好的性能表现。最后将多个header进行concat。





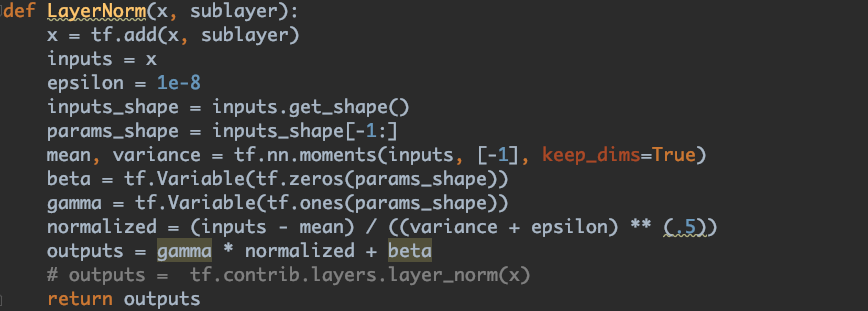
每个小的attention采用Scaled Dot-Product Attention设计。计算方式如下：





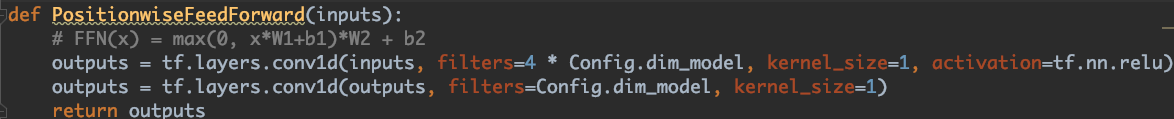
#### 2.2.5 layer normalization层

此处使用残差网络，然后使用layer normalization。



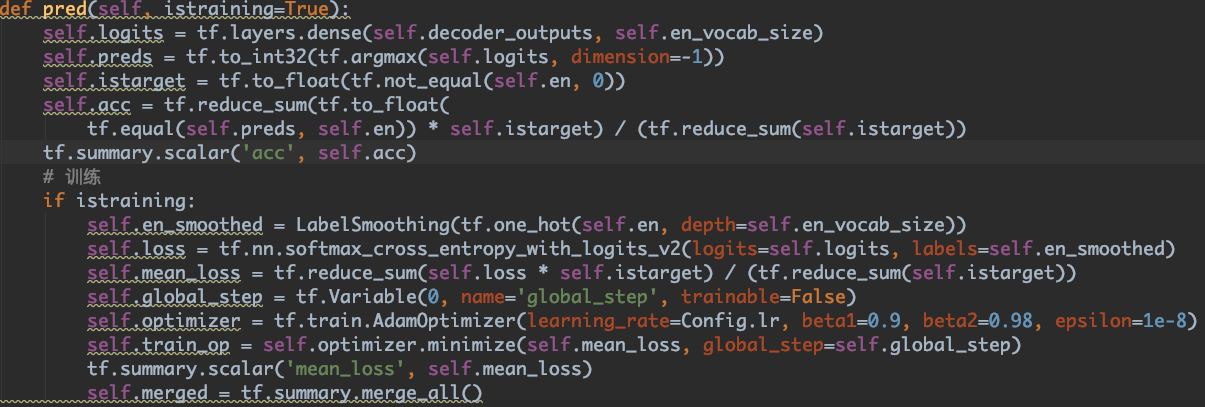
#### 2.2.6 Position-wise前馈网络



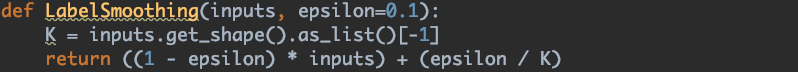


#### 2.2.7 训练

使用Adam优化器，具体参数也使用原论文中的设置。



训练时使用Label Smoothing，参数为0.1。



### 2.3模型参数设置

模型参数设置如下：



### 2.4模型测试过程

模型测试分为两部分：

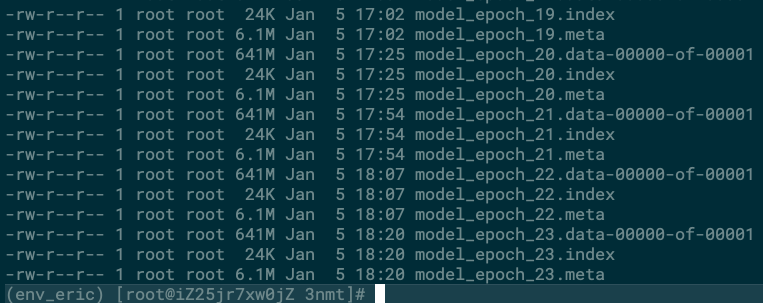
第一步：训练模型

python3 main.py train

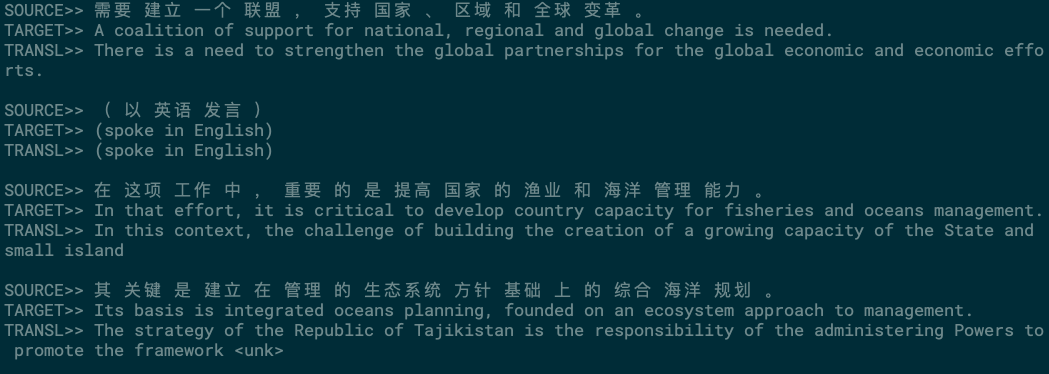
第二步：测试模型

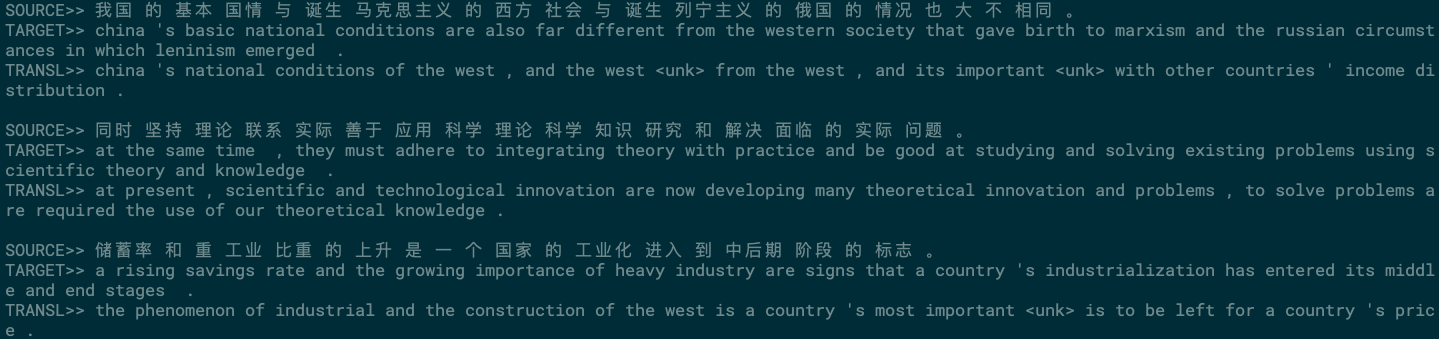
python3 main.py test

下图为模型训练过程保存的模型：



下图为测试输出结果文件内容：





本次作业使用nltk工具包对翻译结果做评测，在测试集上最终的结果为：



可以说结果相当不理想，但在其它大语料短语句上测试时bleu值就比较理想。

### 2.4问题与解决

问题1：最初由于对模型的理解不足，decoder的输入最初没有搞清楚，在编码的时候就出现了错误，即使使用大语料去训练仍然无法得到理想的结果，最后经过反复排查，才得知问题所在。

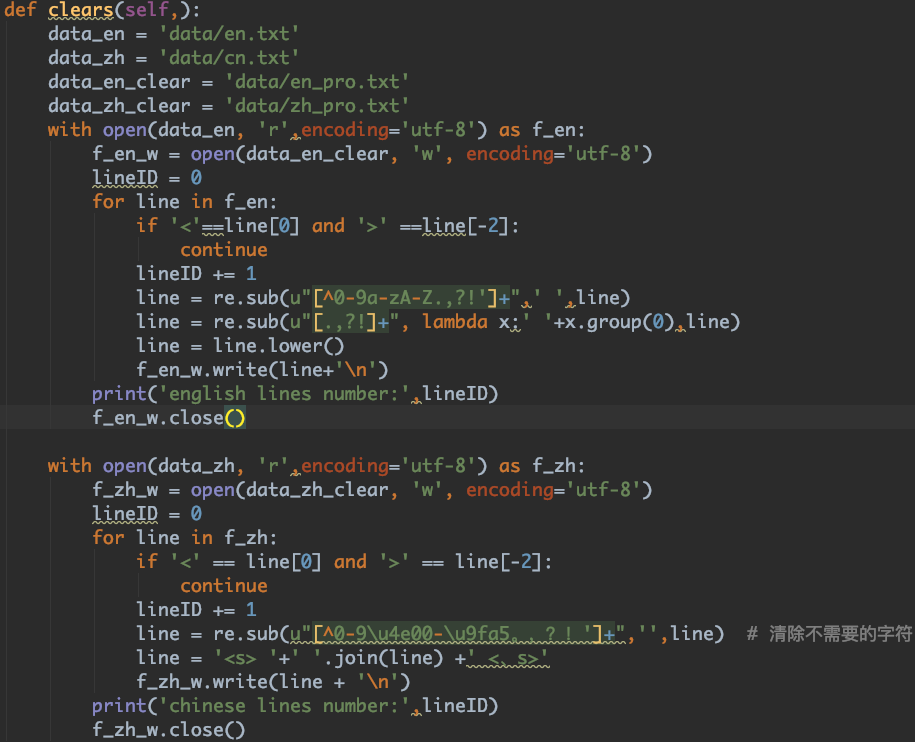
问题2：当设置句子最大长度比较大时，由于语料比较小，而且句子长度基本都处于20-32之间，训练出来模型进行测试，测试结果很不理想，一直怀疑自己是否模型复现有误；最后设置一个比较小的句子长度阈值，在大语料中训练，结果还是比较理想的。从上面两张图也可以看出来，较长的语句翻译结果并不理想。

问题3：由于代码中的字典构造是在程序运行时自己构造的，所以一旦改变了训练集，其字典也就发生了变化，这样训练好的模型就无法使用。解决方法是可以提前将字典保存好，在其它语料上仍然使用该字典，这样就可以测试以训练好的模型。

## 3 Seq2seq+Attention模型

### 3.1数据处理

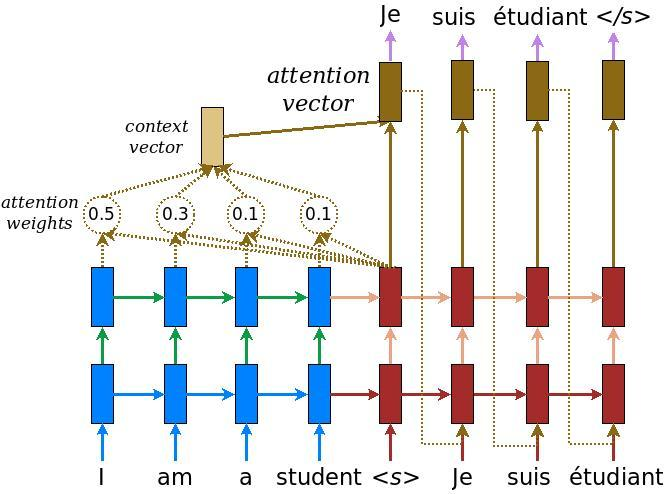
数据处理部分和transformer实现有一些不同，此处需要对中文语料中的词分割成一个个字，然后存储在一个处理后的文件中，并在每段话前后加<s></s>标志。此外，吸取实现transformer中的教训，本次是将字典存储在文件中，这样即使中英文语料不同，也能使用该字典。



### 3.2模型核心部分

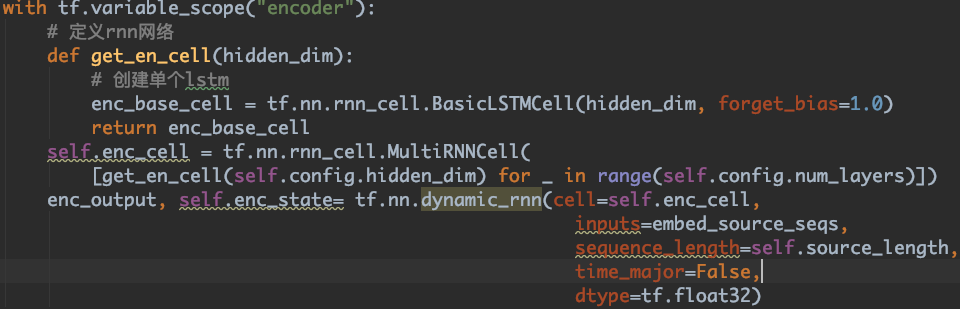
#### 3.2.1整体框架

本次实现是在Seq2seq模型上加入Attention机制，总体框架如下图所示，encoder和decoder都使用两层的LSTM实现。



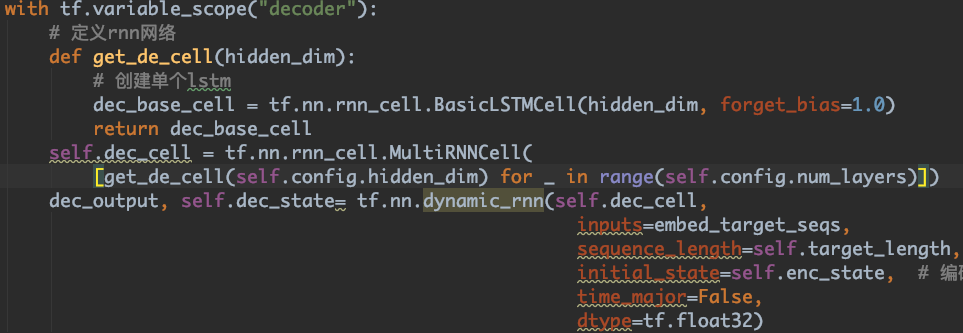
#### 3.2.2 encoder层

使用两层LSTM实现。



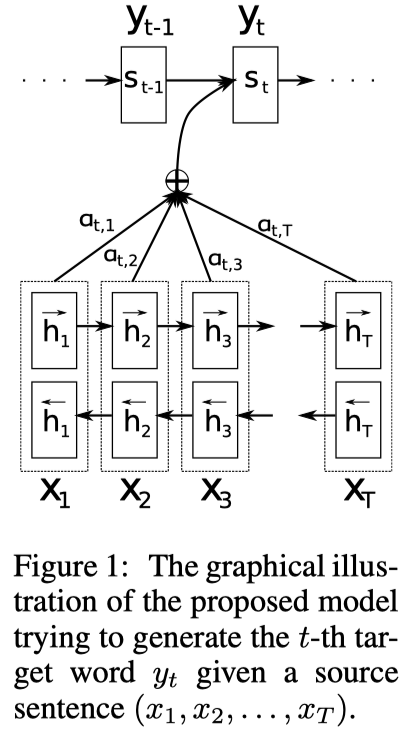
#### 3.2.3 decoder层

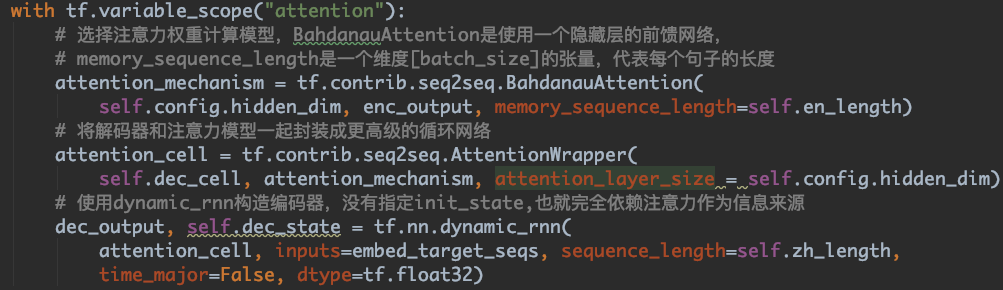
使用两层的LSTM实现。



#### 3.2.4 Attention机制

本次实现使用的Attention机制是BahdanauAttention，它是Bahdanau在论文NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE中提出的，整体Attention结构如下图：





### 3.3模型参数设置

模型参数设置如下：



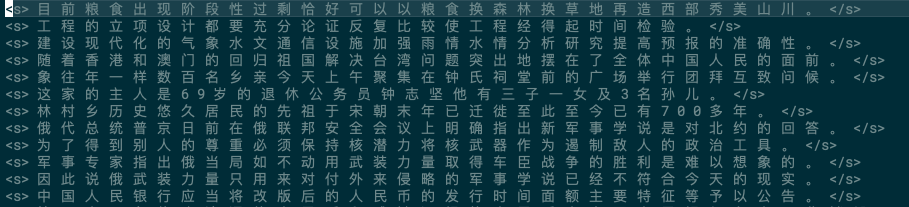
### 3.4模型测试过程

主要分为三部分

1. 数据预处理

python3 data\_util.py

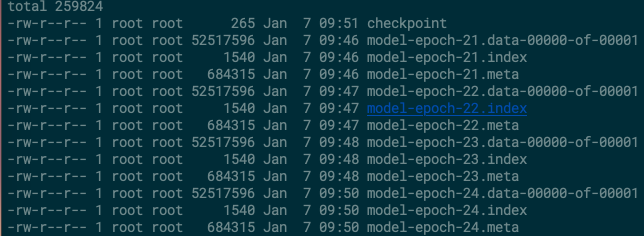
处理后的语料如下：



1. 训练模型

python3 train.py

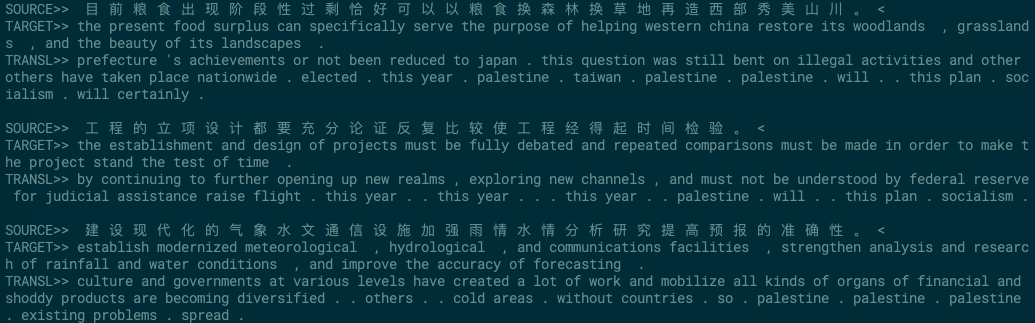
训练的模型如下



1. 测试模型

python3 test.py

结果如下所示，可发现训练结果不是很理想



## 4模型比较

### 4.1基于Attention Mechanism + LSTM的Seq2Seq模型

**优点：**

自适应地计算一个权值矩阵W，权重矩阵W长度与X的的词数目一致，每个权重衡量输入序列X中每个词对输入序列Y的重要程度，不需要考虑输入序列X与输出序列Y中，词与词之间的距离关系。

**缺点：**

1）attention mechanism通常是和RNN结合使用，但RNN**依赖**t-1的历史信息来计算t时刻的信息，因此**不能并行实现**，**计算效率比较低**，特别是训练样本量非常大的时候。

2）顺序计算的过程中信息会丢失，尽管LSTM等门机制的结构一定程度上缓解了**长期依赖**的问题，但是对于特别长期的依赖现象,LSTM依旧无能为力。

### 4.2 Transformer模型

**优点：**

1）Transformer完全摒弃了递归结构，依赖注意力机制，挖掘输入和输出之间的关系，这样做最大的好处是能够**并行计算**了；

2）要计算一个序列长度为n的信息要经过的路径长度。cnn需要增加卷积层数来扩大视野，rnn需要从1到n逐个进行计算，而self-attention只需要一步矩阵计算就可以，Transformer 在结构上优越，任何两个 token 可以无视距离直接互通。所以也可以看出，self-attention可以比rnn更好地解决长时依赖问题。当然如果计算量太大，比如序列长度n>序列维度d这种情况，也可以用窗口限制self-attention的计算数量。

3）self-attention模型更可解释，attention结果的分布表明了该模型学习到了一些语法和语义信息

**缺点：**

1）实践上：有些rnn轻易可以解决的问题transformer没做到，比如复制string，尤其是碰到比训练时的sequence更长的时。

2）理论上：transformers非computationally universal（图灵完备），这种非RNN式的模型是非图灵完备的，无法单独完成NLP中推理、决策等计算问题（包括使用transformer的bert模型等等）。