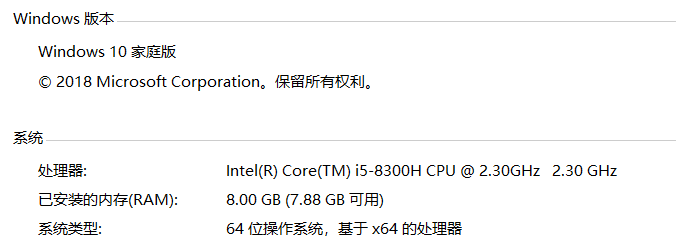
**实验十 神经机器翻译**

1. **实验目的**
2. 掌握循环神经网络及其变种的结构。
3. 了解Encoder-Decoder框架， word2vec技术，以及自然语言处理中的attention机制。
4. 进一步熟悉TensorFlow的使用，学会使用其自带的RNN框架、或LSTM框架，并结合收集到的相关数据，搭建Encoder-Decoder机器翻译模型。
5. **实验要求**

利用Python语言和TensorFlow框架，构造简单的机器翻译模型，以实现英语和汉语的相互转换。

1. **实验运行环境**

Lenovo Legion Y7000p，Intel 8代i5，8G内存，显卡：NVIDIA GeForce GTX 1050 Ti 4G



Python 3.6 + TensorFlow 1.11.0 + CUDA 9.0 + cuDNN 7.6.0 + VSCode

1. **实验步骤与方法**

本次作业我们选择以单向的RNN为例进行网络搭建。

示例模型结构可以分为4个部分。首先是将输入序列进行词嵌入，再将词嵌入后的向量输入到LSTM中编码以聚合输入序列的上下文信息。得到的上下文向量输入到解码器中进行解码，解码器也是由LSTM组成，最终输出翻译结果，如图5所示。下面针对这几层对部分代码片段进行介绍。



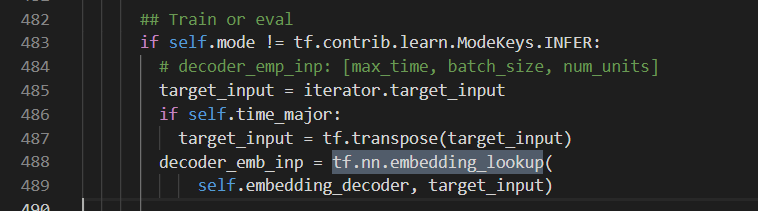
图5 模型结构图

（1）词嵌入

embdding\_ops.embedding\_lookup是tensorflow中的方法，用于查找某个单词id对应的词嵌入向量。这里的encoder\_emb\_inp是词嵌入之后得到的向量，将被传入到encoder层中去。

encoder\_emb\_inp = tf.nn.embedding\_lookup(self.embedding\_encoder, source)

代码见**model.py**



（2）encoder层

Encoder层可以使用单向或双向的RNN或者LSTM。以单向的RNN为例：

cell = self.\_build\_encoder\_cell(

hparams, num\_layers, num\_residual\_layers)

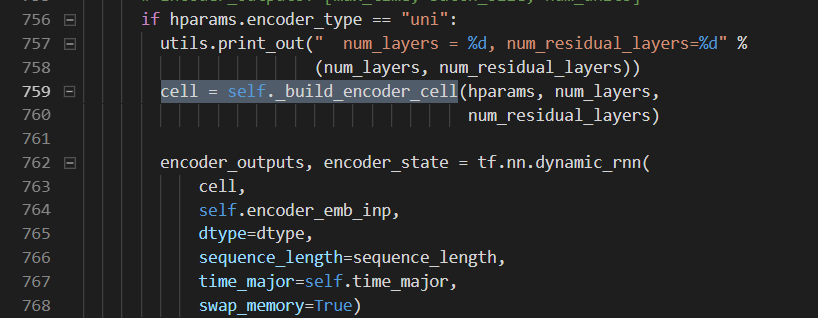
encoder\_outputs, encoder\_state = tf.nn.dynamic\_rnn(

cell, encoder\_emb\_inp, dtype=dtype,

sequence\_length=iterator.source\_sequence\_length,

time\_major=self.time\_major)

代码见**model.py**，以单向RNN为例（uni）:



（3）decoder层

#Bulild RNN cell

decoder\_cell = tf.nn.rnn\_cell.BasicLSTMCell(num\_units)

#Helper

helper = tf.contrib.seq2seq.TrainingHelper(

decoder\_emb\_inp, decoder\_lengths, time\_major=True)

#Decoder

decoder = tf.contrib.seq2seq.BasicDecoder(

decoder\_cell, helper, encoder\_state,

output\_layer=projection\_layer)

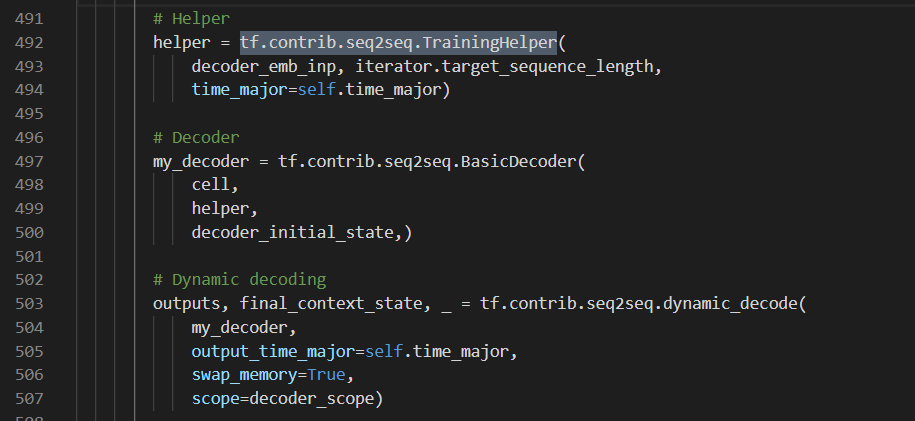
#Dynamic decoding

outputs, \_ = tf.contrib.seq2seq.dynamic\_decode(decoder, ...)

logits = outputs.rnn\_output

这里的核心是tf.contrib.seq2seq.BasicDecoder，它是Tensorflow seq2seq API中的一个类，它接收一个decoder\_cell，一个helper，还有上下文变量encode\_state,投影层变量projection\_layer，最终利用它得到结果。投影层变量的定义如下，它把LSTM中的每一步输出转换为一个vocab\_size维的向量，vocab\_size是目标语言词汇表中所有词的个数。

代码在**model.py**：



projection\_layer = layers\_core.Dense( tgt\_vocab\_size, use\_bias=False)

（4）注意力

除了基本的Encoder-Decoder模型外，还可以在模型中添加注意力机制以提升模型表现。

#attention\_states: [batch\_szie, max\_time, num\_units]

attention\_states = tf.transpose(encoder\_outputs, [1, 0, 2])

#Creat an attention mechansim

attention\_mechanism = tf.contrib.seq2seq.LuongAttention(

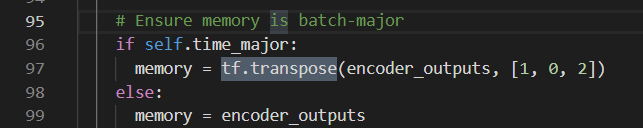
num\_units, attention\_states,

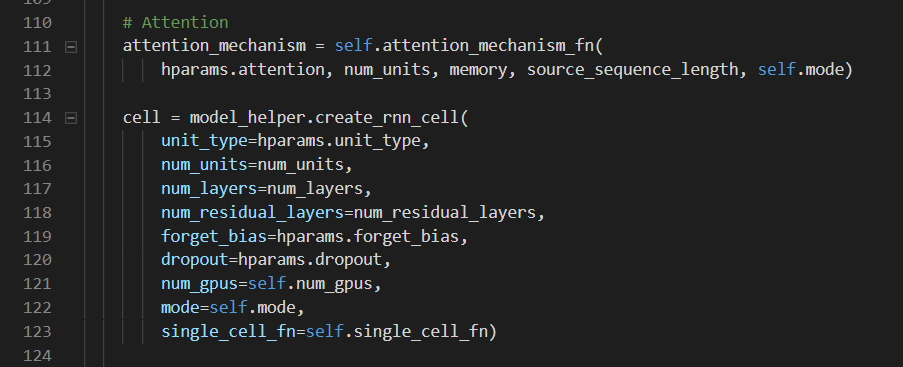
memory\_sequence\_length=source\_sequence\_length)

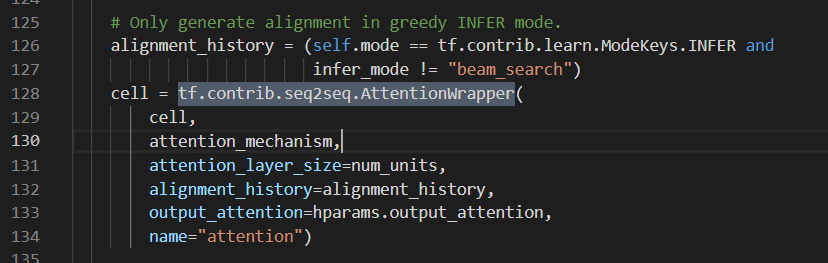
decoder\_cell = tf.contrib.seq2seq.AttentionWrapper(

decoder\_cell, attention\_mechanism,

attention\_layer\_size=num\_units)







1. **代码结构解释**

代码运行的入口是**main.py**，直接在命令行中

python main.py

我们已经根据之前的测试和经验设置了命令行参数，在这个文件中我们也可以读入命令行参数并修改 parser argument，这里我们几乎已经设置好 default 值了，所以应该无需输入命令行参数。

**Main.py**代码会调用**nmt.py**，这个文件也是我们整个模型的入口，也就是接口。

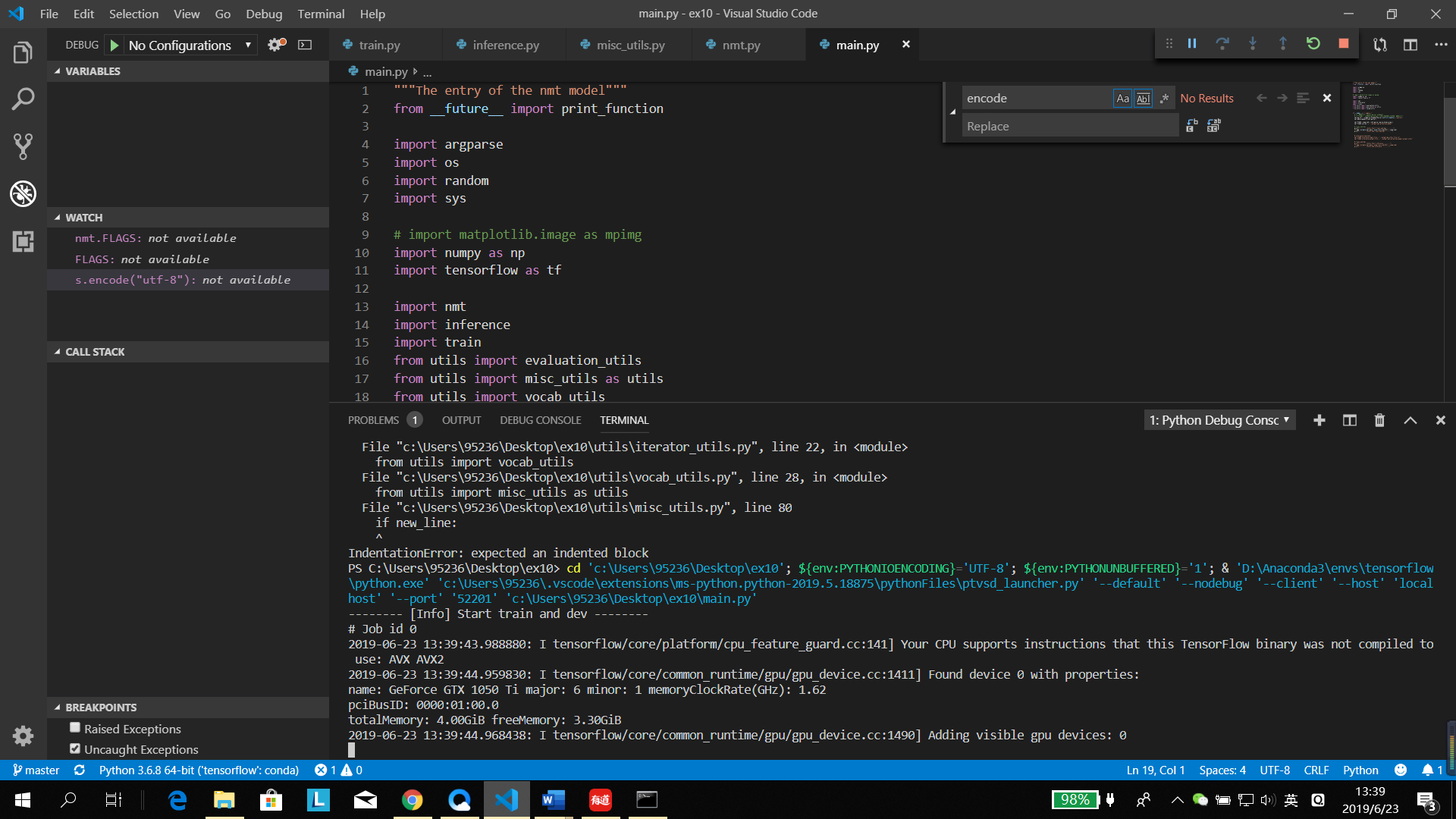
文件**model.py**是我们整个rnn网络结构的模型，是最基础的神经网络nmt模型。

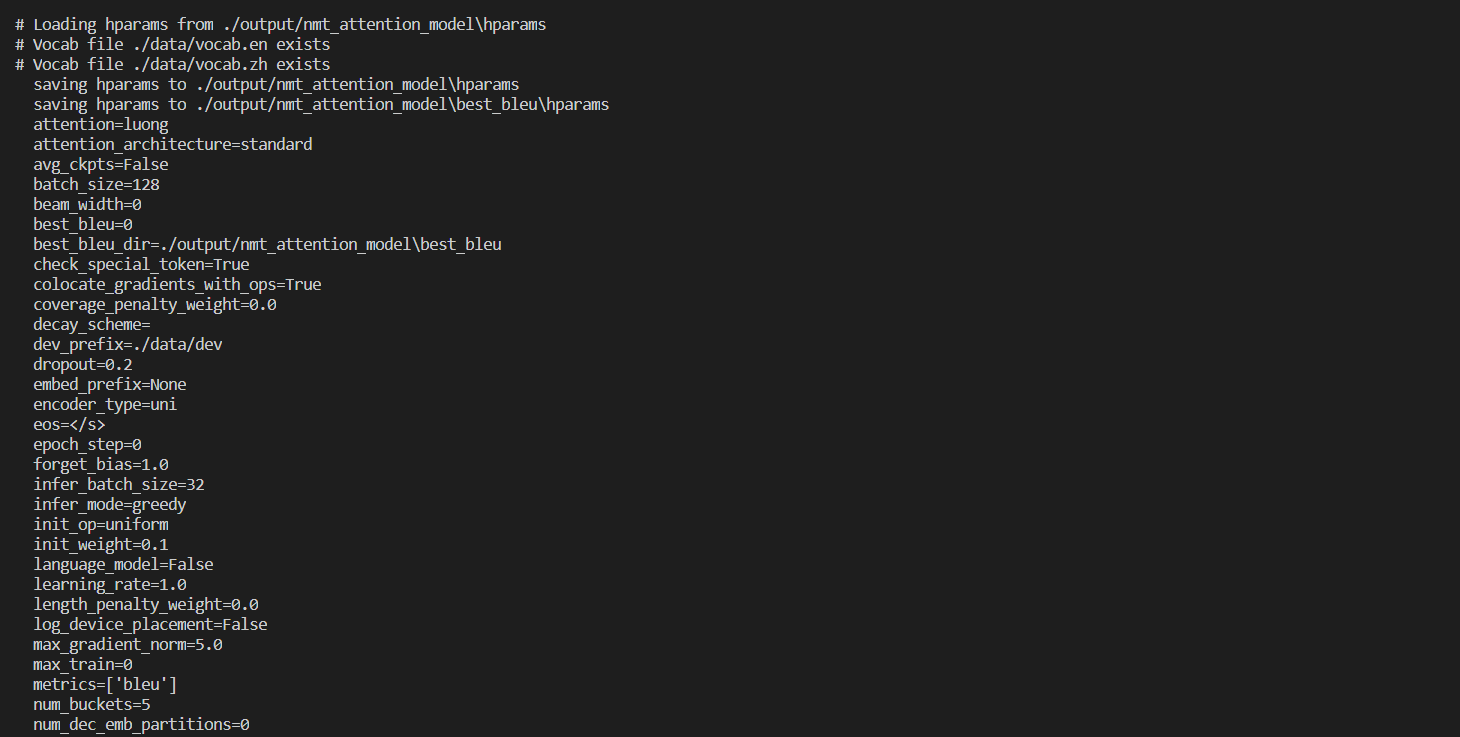
文件**attention\_model.py**继承并进一步实现完善了**model.py**中的rnn模型，给基础模型加上了attention机制。

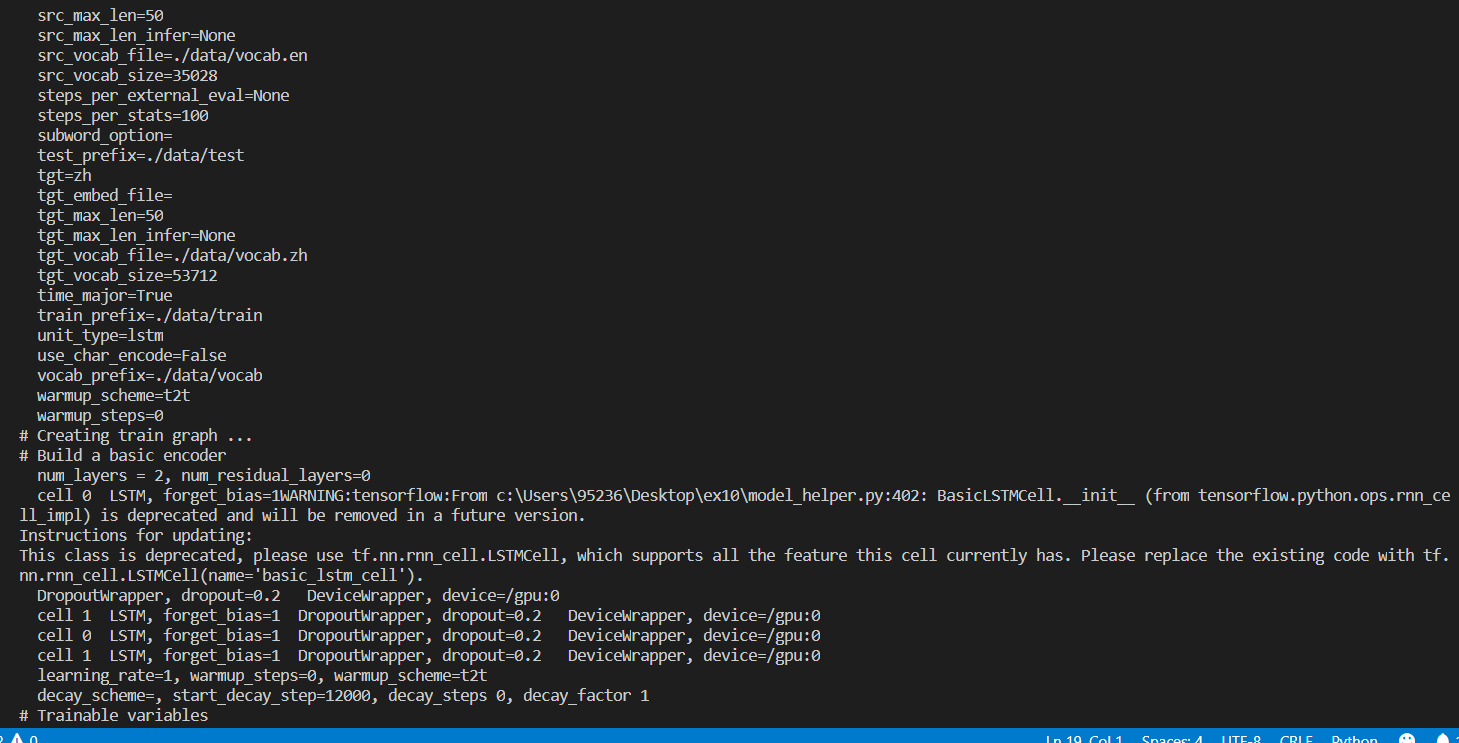
其他文件功能如文件名描述的一样，一些是工具类。

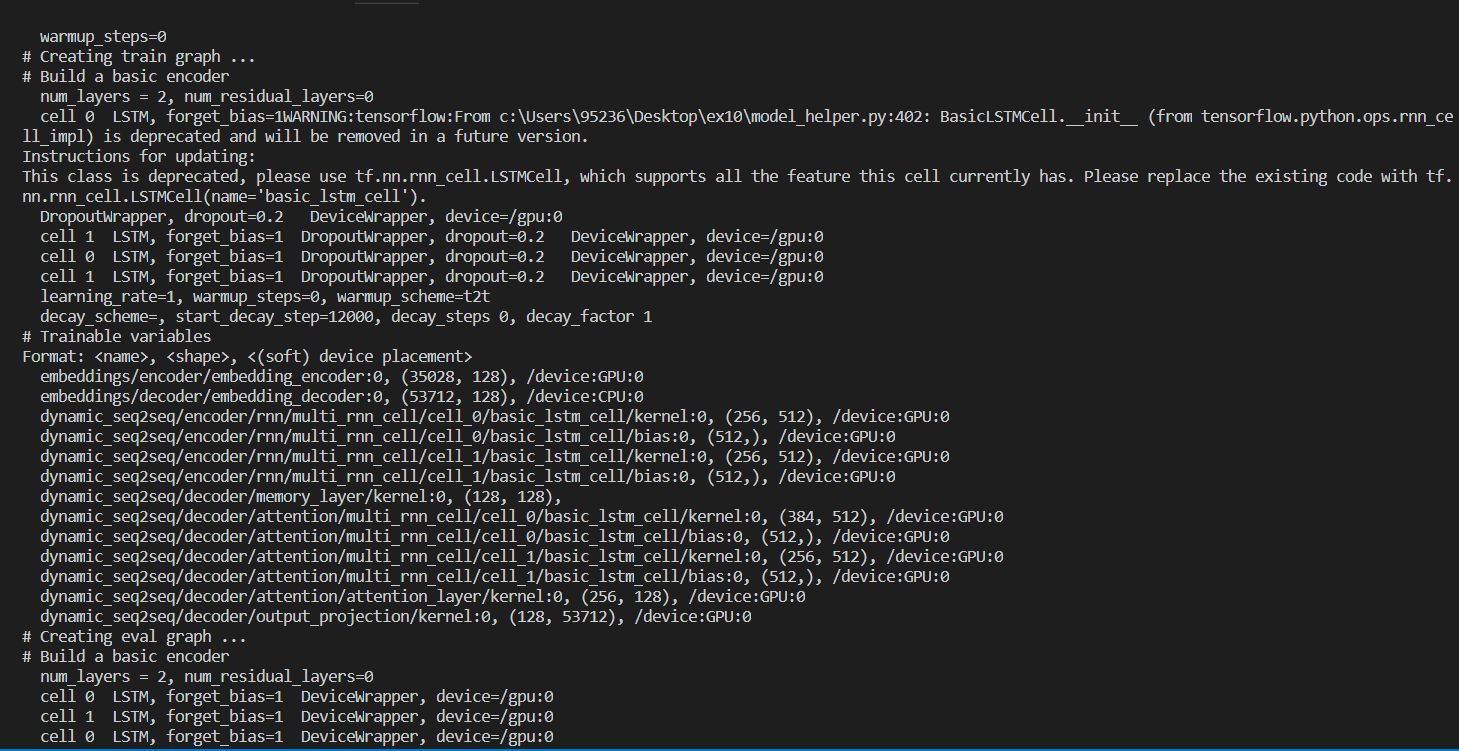
1. **实践与测试**

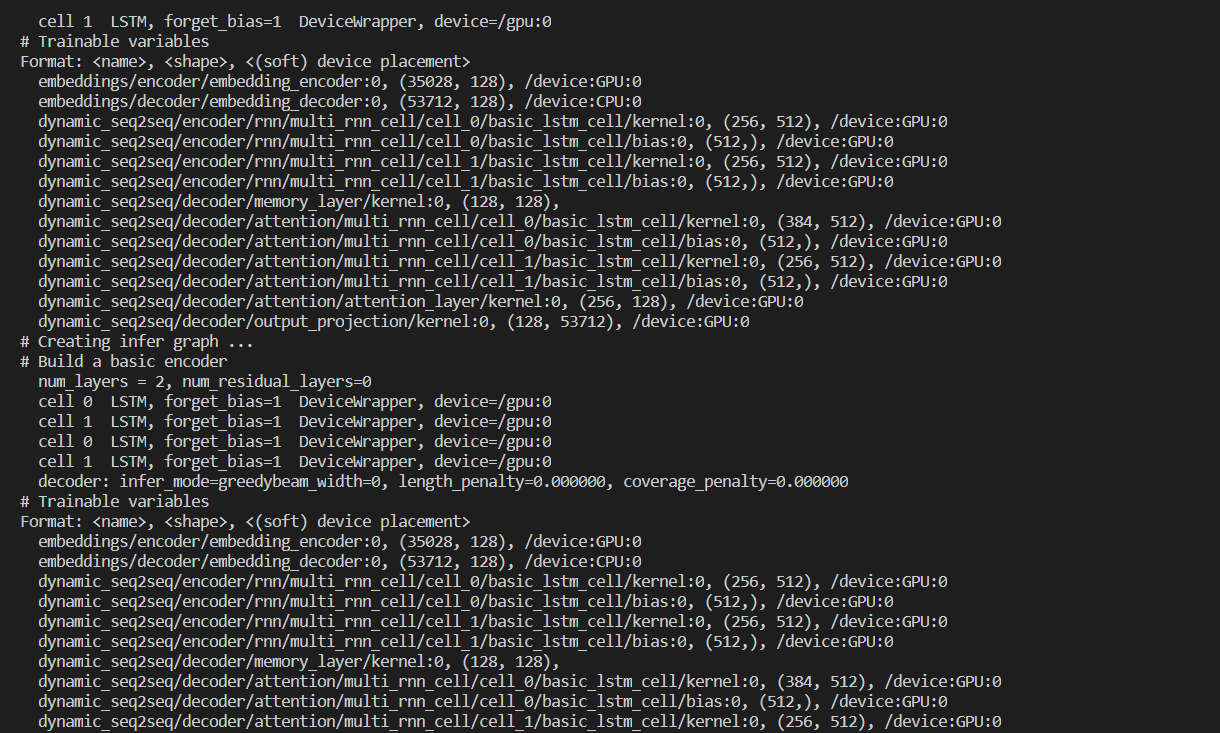
模型训练和验证/测试：因为图片太多只节选部分，验证集和测试集在训练过程中都有反复使用.

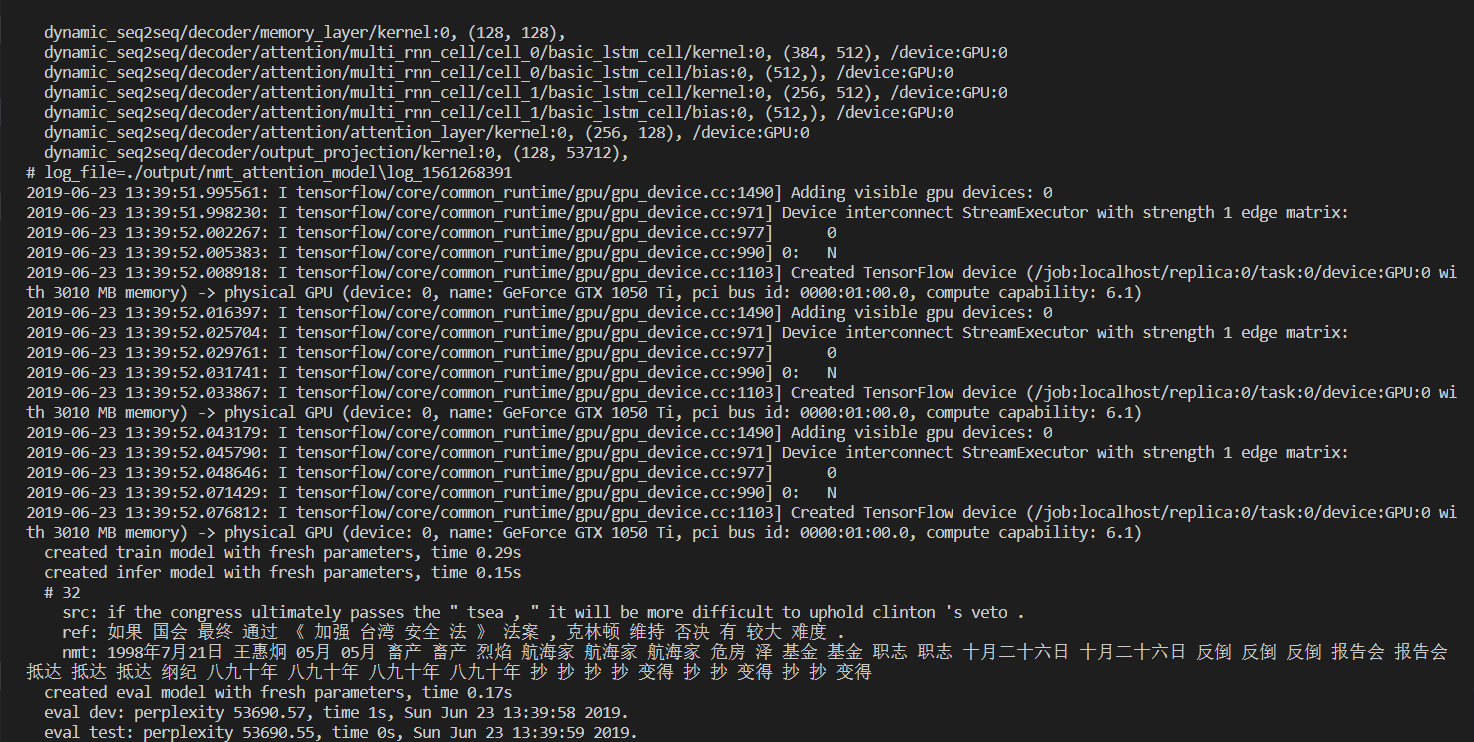


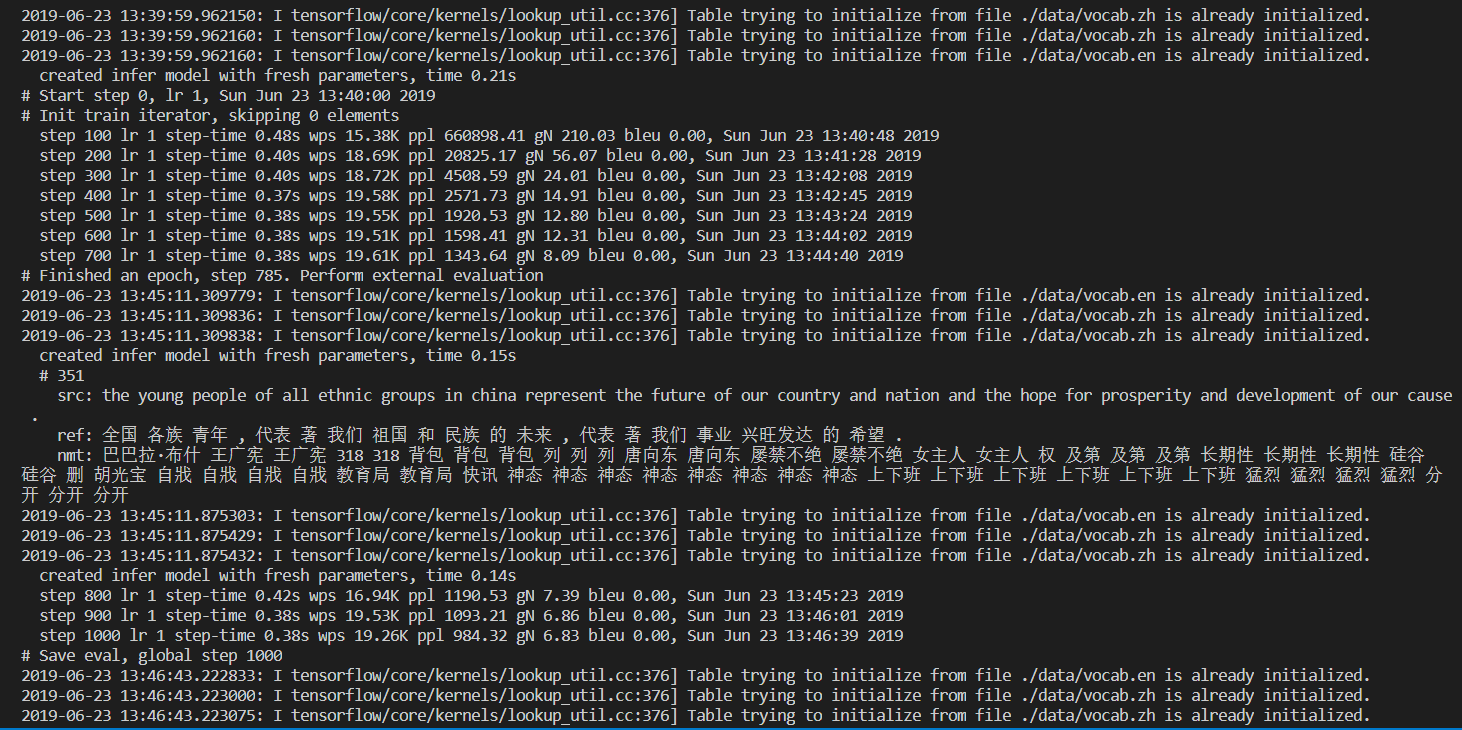


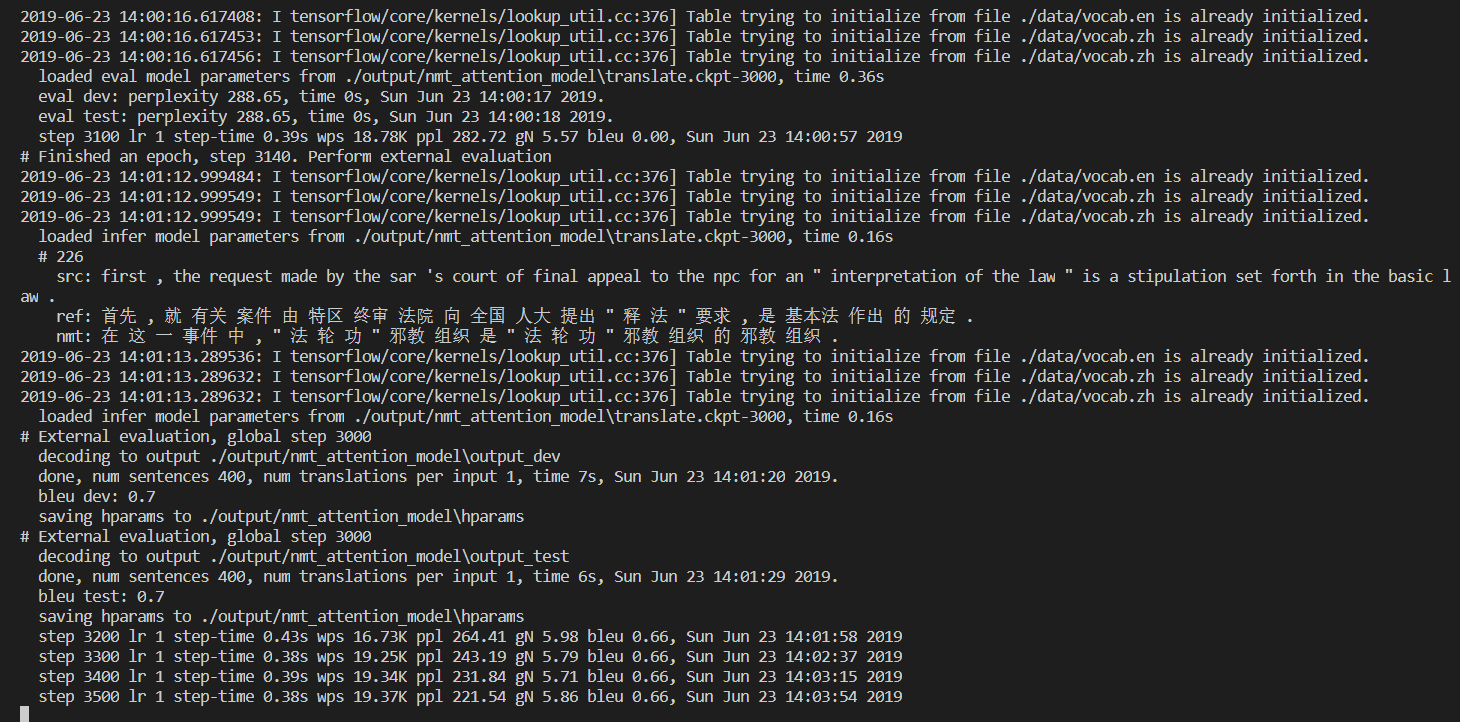


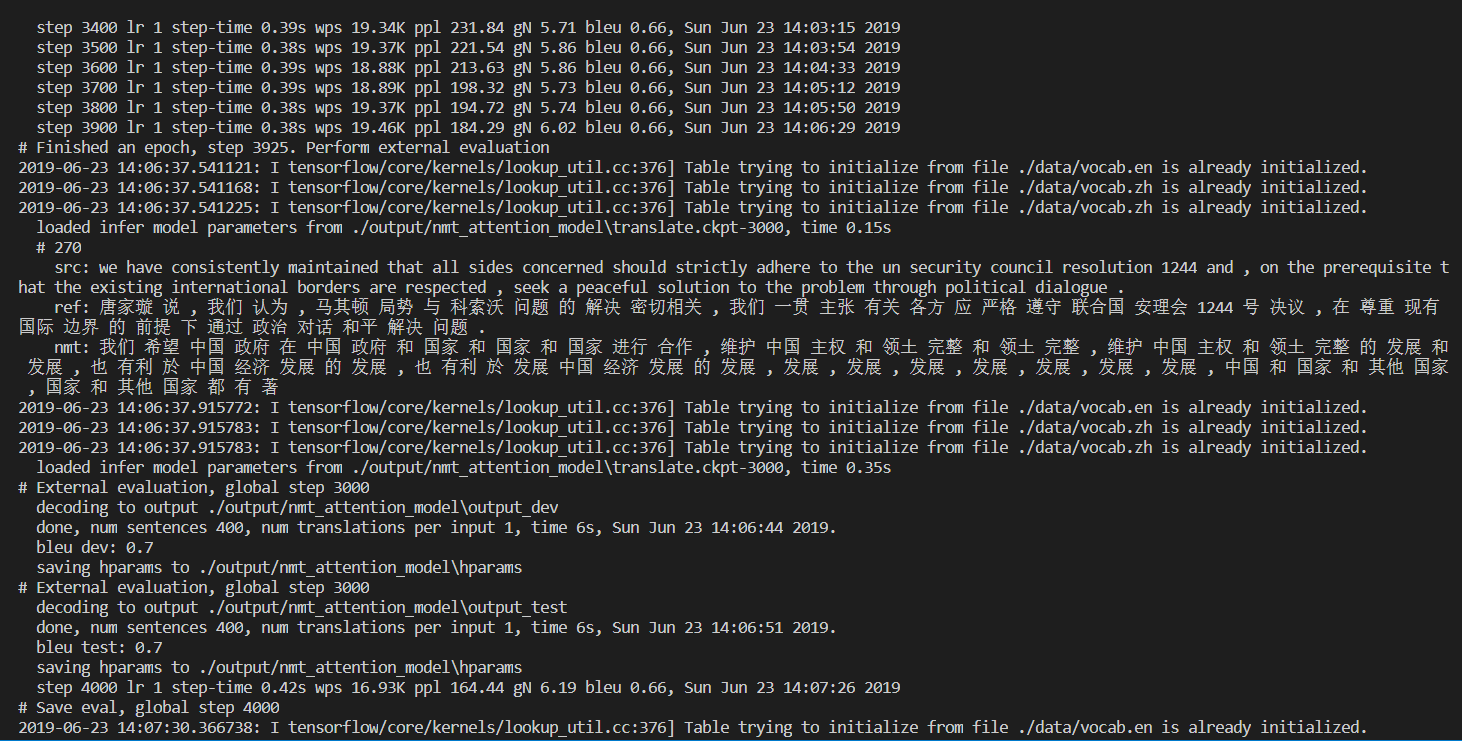


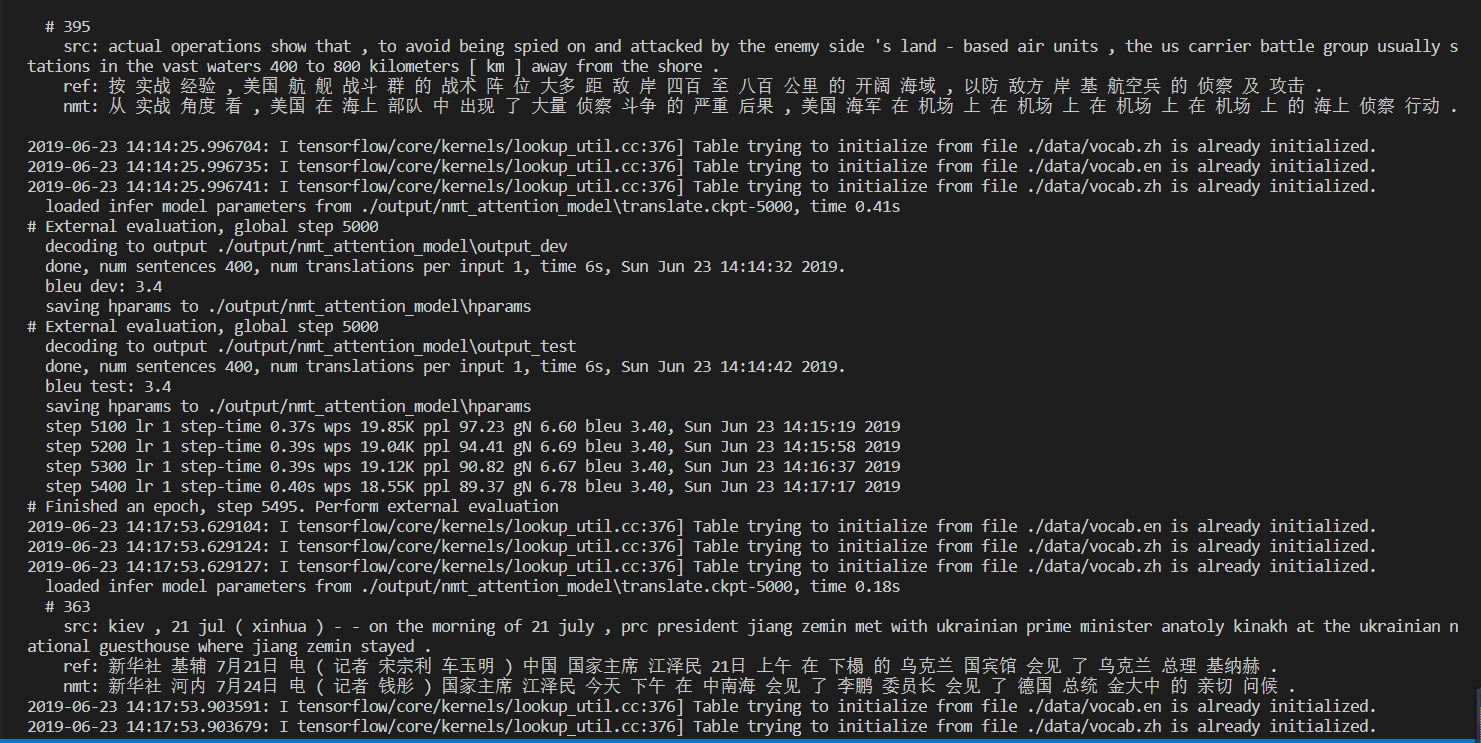


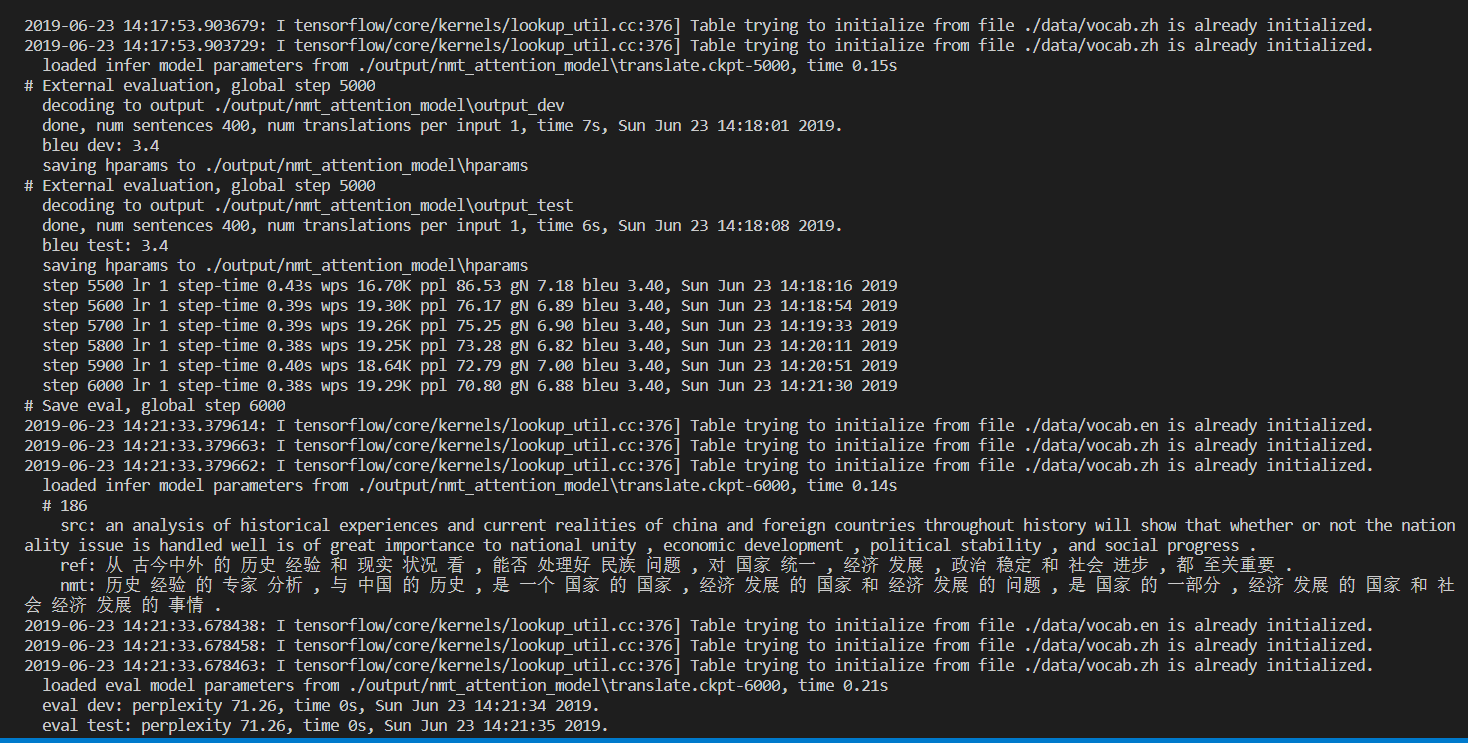


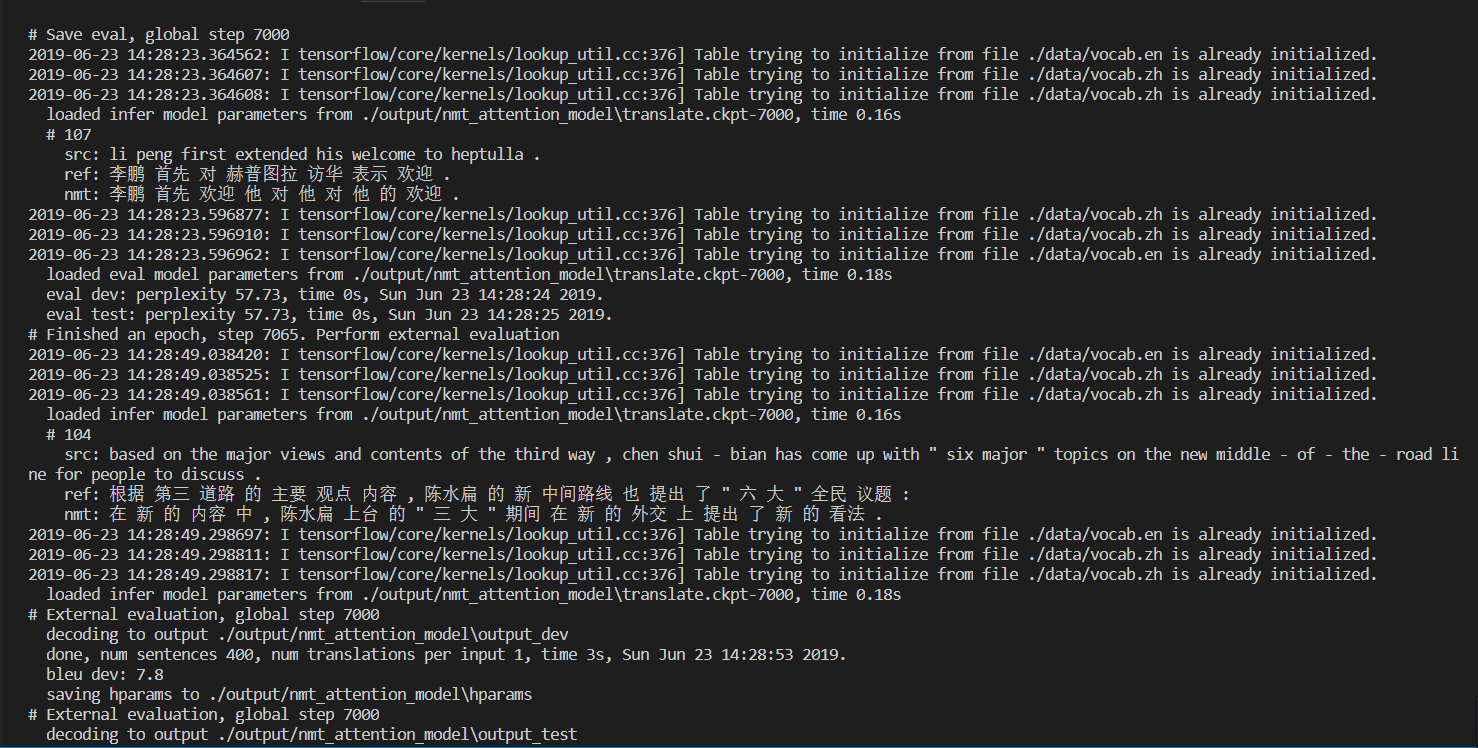


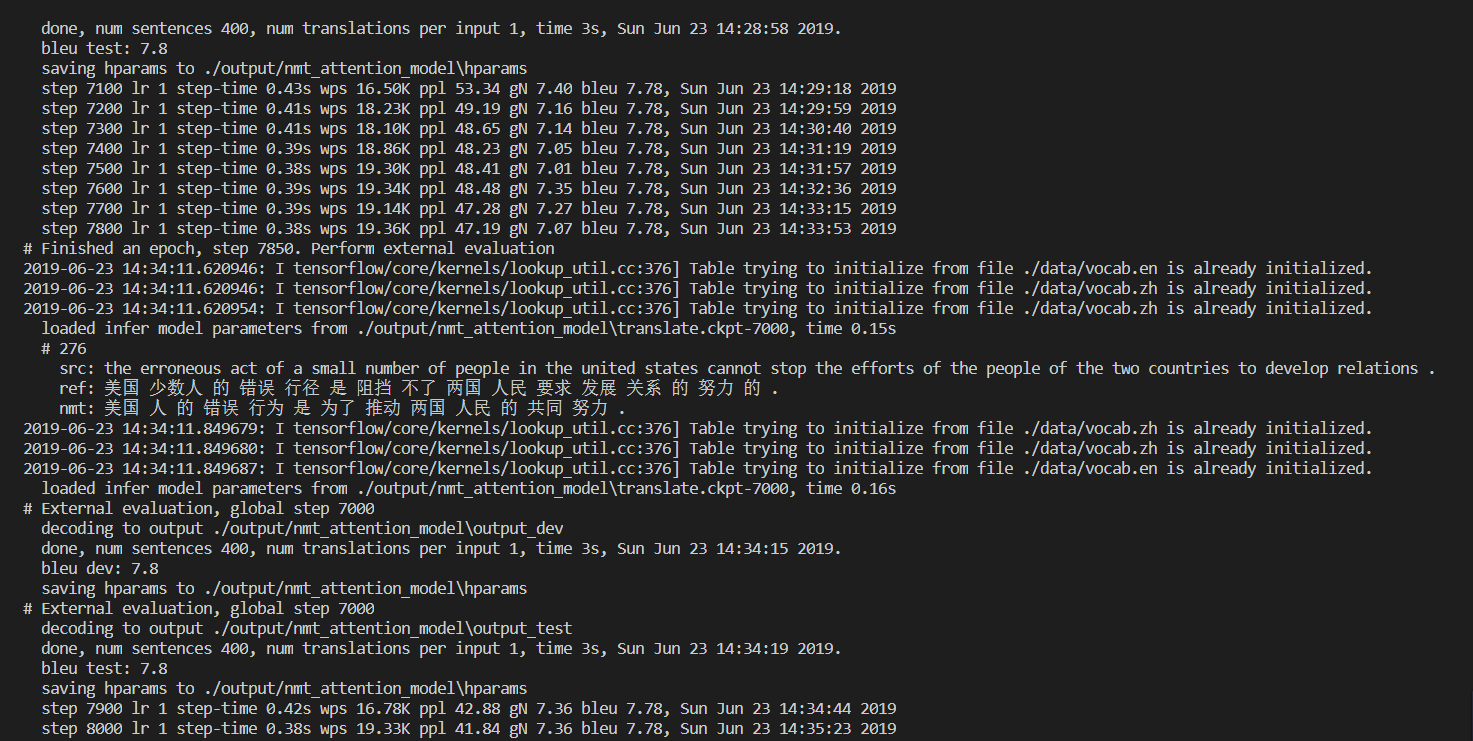


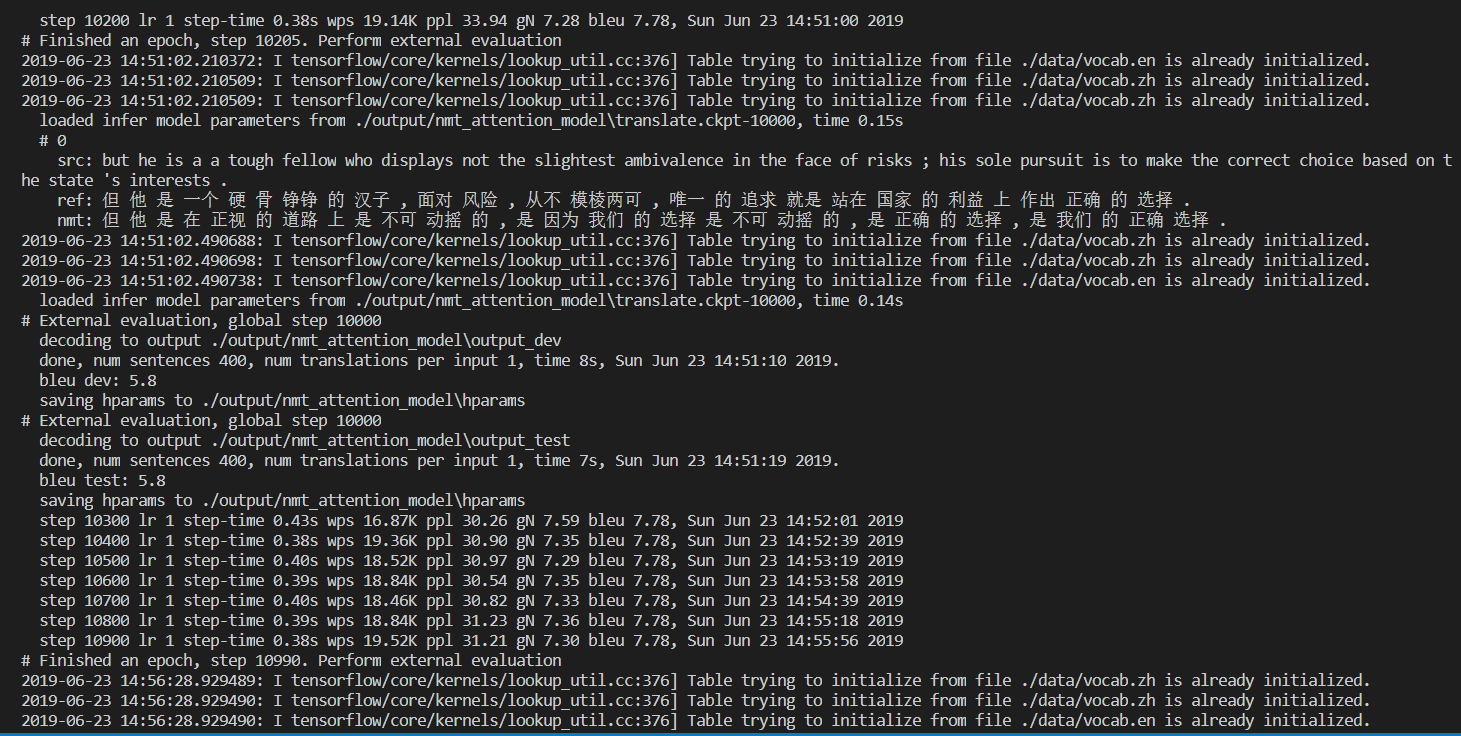


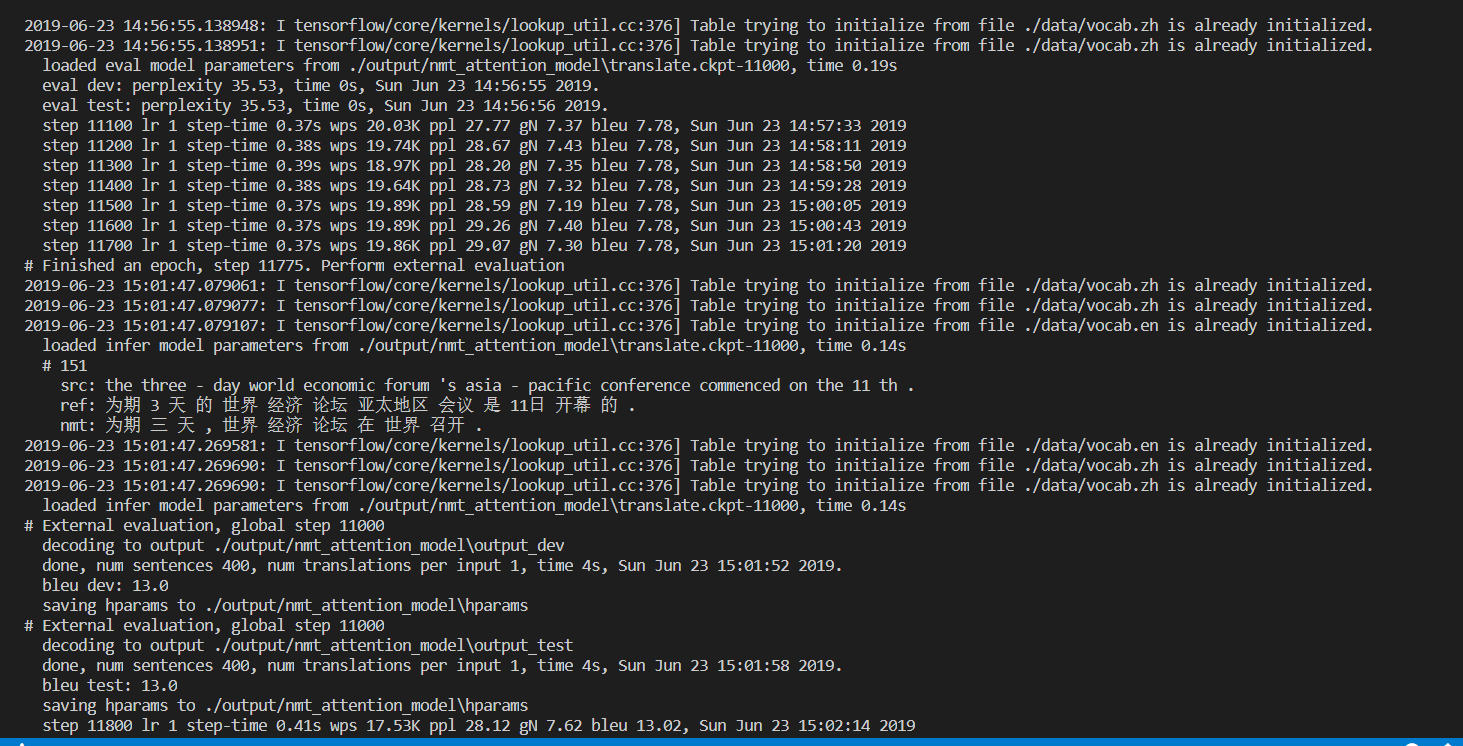


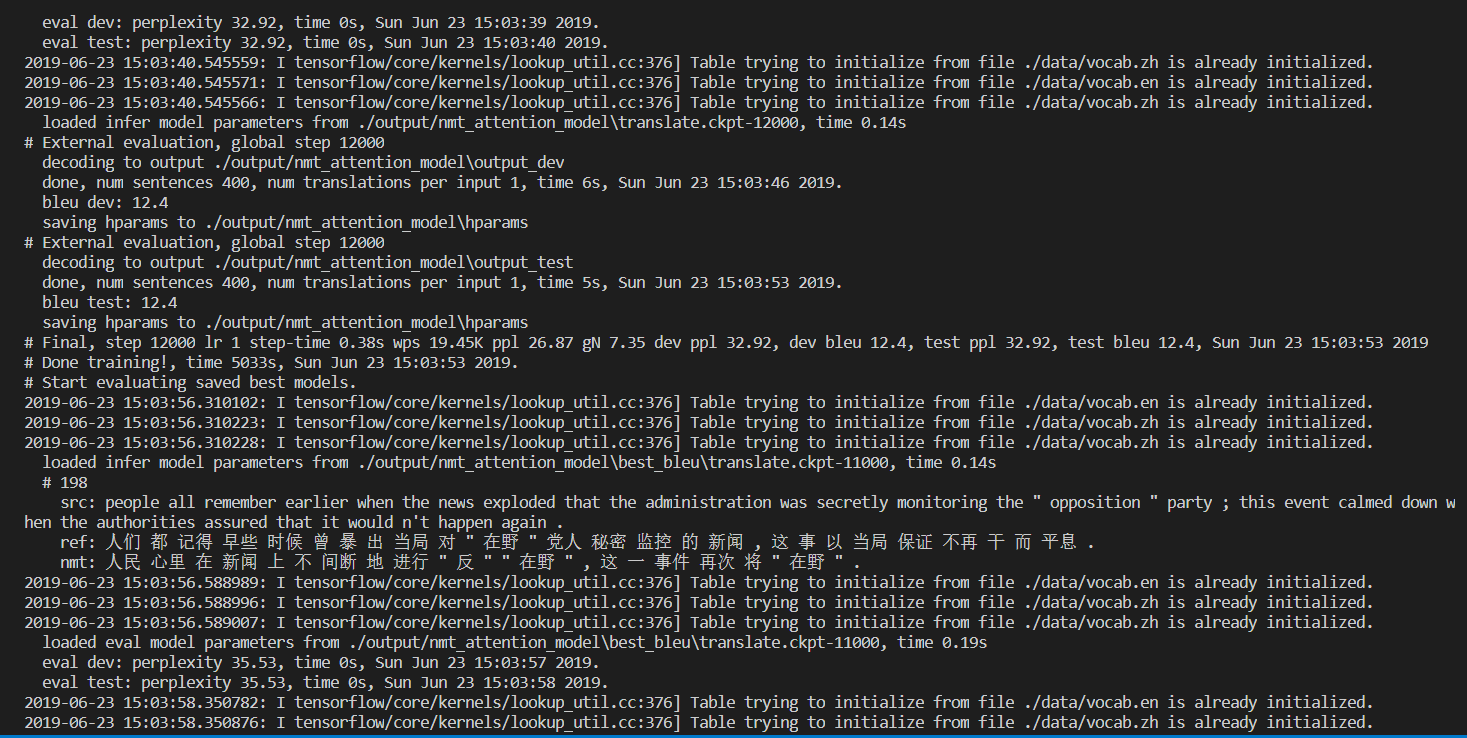












训练结束，保存模型：

