## 1 结果说明

|  |  |  |  |
| --- | --- | --- | --- |
|  | Attention | | Transformer |
| Greedy解码 | Beamsearch解码 |
| my\_bleu\_avg | 0.42 | 0.39 | 0.55 |
| my\_bleu\_max | 0.72 | 1.0 | 1.0 |
| my\_bleu\_min | 0.08 | 0.0 | 0.26 |
| nltk\_bleu\_avg | 0.18 | 0.15 | 0.10 |
| nltk\_bleu\_max | 0.46 | 1.0 | 0.37 |
| nltk\_bleu\_min | 0.03 | 0.0 | 0 |

my\_bleu\_XX是使用自己写的bleu函数对测试集1300个句子计算的bleu值的平均值、最大值和最小值，其中w1, w2, w3, w4 = 0.25, 0.25, 0.25, 0.25；nltk\_bleu\_XX是使用nltk的sentence\_bleu函数计算的bleu值的平均值、最大值和最小值，其中w1, w2, w3, w4 = 1，0，0，0。由上表可知，Transformer具有更好的效果。

注：在自己写的bleu函数里，如果计算过程会有p2或p3或p4=0导致无法取log的情况， 为了运算此时令wi=0；如果在自己写的函数中也令w1, w2, w3, w4 = 1，0，0，0， 结果与nltk接口计算结果相同。

## 2 Attention模块具体说明

1. **代码说明**

./Homework5-Machine Translation/Attention/

│ bleu.py // bleu评测(包括自己写的bleu函数和调用nltk的bleu函数)

│ data\_process.py //数据处理，包括 wordid建立，数据读取、生成batches等

│ main.py //主函数入口

│ model.py // RNN+Attention模型

│ README-运行命令.txt //一键运行命令说明

│ test.py // 随意测的代码，可忽略

│ word2vector\_cn\_embedding120.pkl // 中文词向量的pickle文件

│ word2vector\_en\_embedding120.pkl // 英文词向量的pickle文件

├─data/… // 目录内容略,存储训练数据、测试数据

│

├─model/… //目录内容略，存储训练好的模型，测试时可直接加载

│

└─\_\_pycache\_\_

关于具体的函数和操作，均在代码中进行了相应注释标注。

1. **思路大致描述**

（1）前期准备部分：主要是建立word\_id,数据的读取等工作,以及一些必要函数的准备，具体见data\_process.py。在data\_process.py中，还包括了RNN+Attention中所使用的生成baches的函数、进行padding的函数等。

（2） 核心过程：基本就是model.py的完成过程，具体包括：

① 初始化必要的基本变量，具体见\_\_init\_\_部分；

②定义模型架构的搭建和运算关系，具体如build\_graph()函数中所示，主要对编码器和解码器的定义进行解释：

* Encoder: 首先通过tf.nn.embedding\_lookup查询词向量，然后定义BasicRNNCell, 通过tf.nn.dynamic\_rnn运行得到最后的outputs和states。注意这里time\_major=False, outputs的维度是[batch\_size, max\_time, output\_size].
* Decoder: 首先定义具有Attention的decoder\_cell,具体见\_build\_decoder\_cell()函数。Attention机制的加入通过tf提供的接口tf.contrib.seq2seq.LuongAttention和AttentionWrapper完成，且以Encoder最终的encoder\_states作为decoder\_cell的初始化（）。同时注意，如果解码要使用beamsearch, 则需要根据beam\_width进行bile\_batch操作。

然后，如果是训练模式，则给英文输入句首加上标志<GO>然后进行词向量查询，接下来主要通过TrainingHelper、BasicDecoder和dynamic\_decoder三个函数完成功能；如果是测试模式，则通过GreedyEMbeddingHelper 或BeamSearchDecoder函数进行解码。

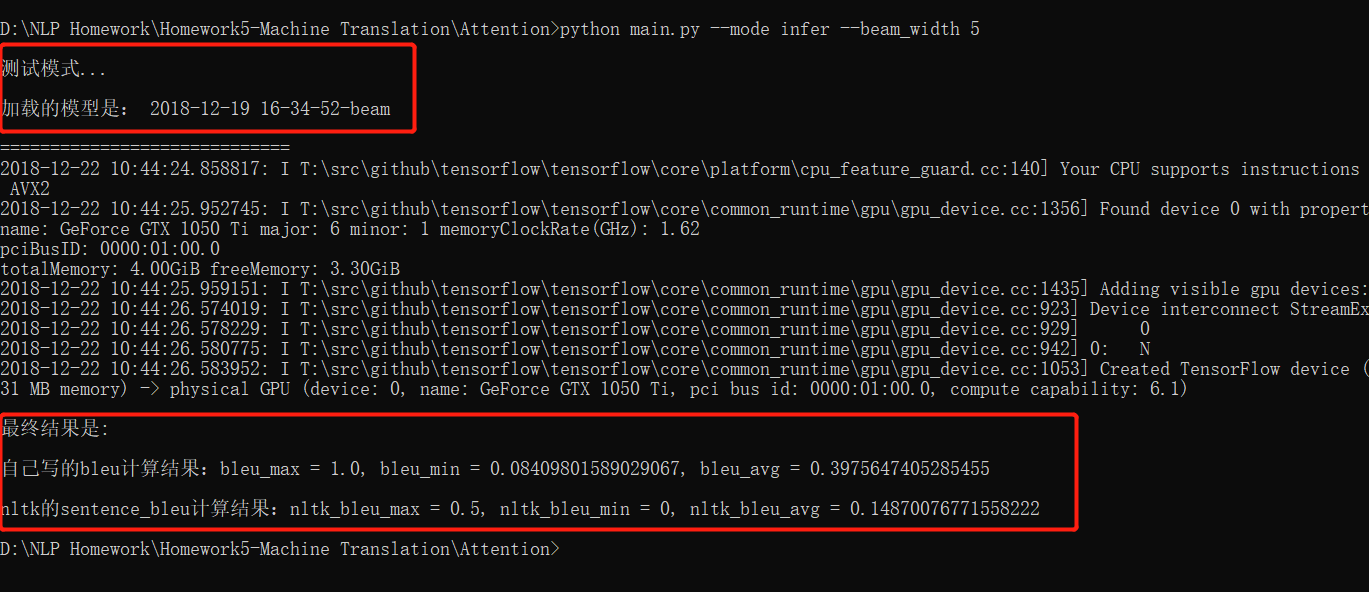
（3）测试

最后运行时，根据训练或翻译测试的需求，选择train()函数或test()运行即可。

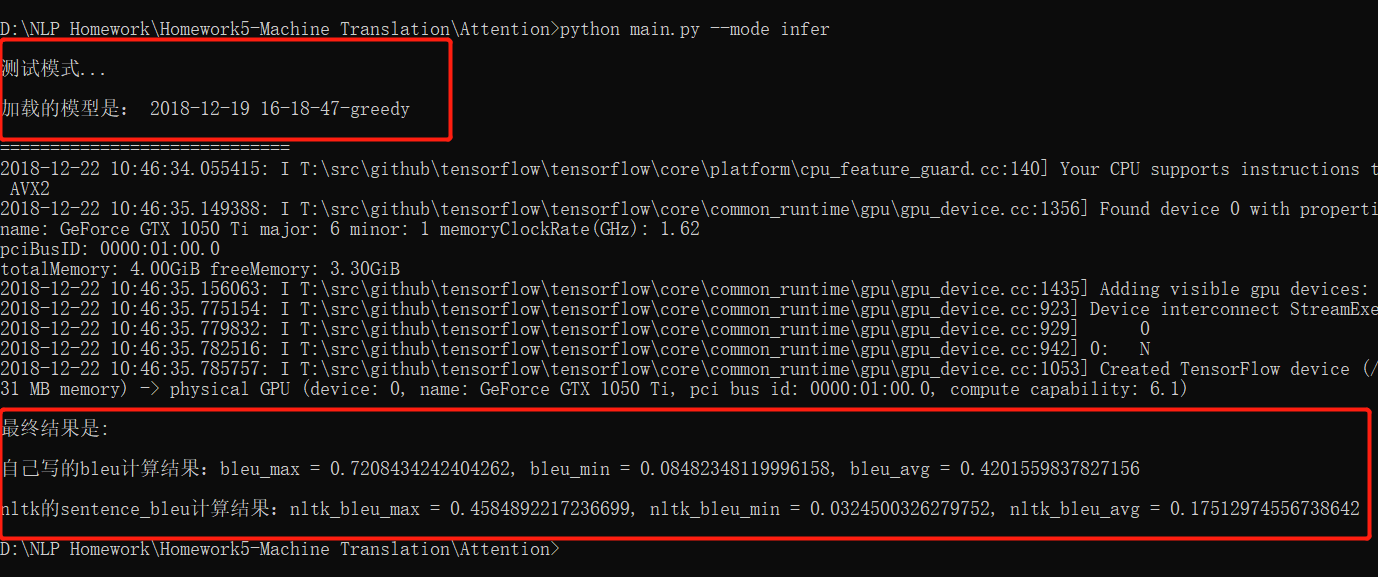
1. **实验结果**

实验结果可参见第一页总的结果说明，这里是实际的截图

BeamSearch模式下：



Greedy



1. **代码运行说明**

(1）进入 ./Attention/ 目录下: cd Attention

（2）以Greedy模式运行： python main.py --mode infer

（3）以BeamSearch模式运行： python main.py --mode infer --beam\_width 5

## 3 Transformer模块说明

1. **代码说明**

./Homework5-Machine Translation/Attention/

│ assist\_func.py // model调用相关函数, eg.multiattention, feedforward等

│ bleu.py // bleu评测(包括自己写的bleu函数和调用nltk的bleu函数)

│ data\_process.py //数据处理，包括 wordid建立，数据读取、生成batches等

│ main.py //主函数入口

│ model.py // Transformer模型

│ README-运行命令.txt //一键运行命令说明

│ test.py // 随意测的代码，可忽略

│ word2vector\_cn\_embedding120.pkl // 中文预料词向量

│ word2vector\_en\_embedding120.pkl // 英文预料词向量

├─data/… // 目录内容略,存储训练数据、测试数据

│

├─model/… //目录内容略，存储训练好的模型，测试时可直接加载

│

└─\_\_pycache\_\_

关于具体的函数和操作，均在代码中进行了相应注释标注。

1. **思路大致描述**

（1）前期准备部分：主要是建立word\_id,构建词向量等工作,以及一些必要函数（batch生成等）的准备，具体见data\_process.py。

（2） 核心过程：基本就是model.py的完成过程，核心是Encoder和Decoder的构建：

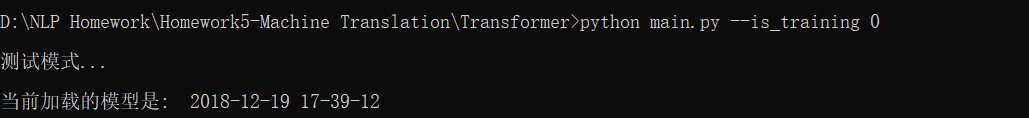
①Encoder: 在进行词向量查询和位置编码处理后，进行六个层的堆叠，每个层主要包括MulTiHead-Attention和一个前馈神经网络。

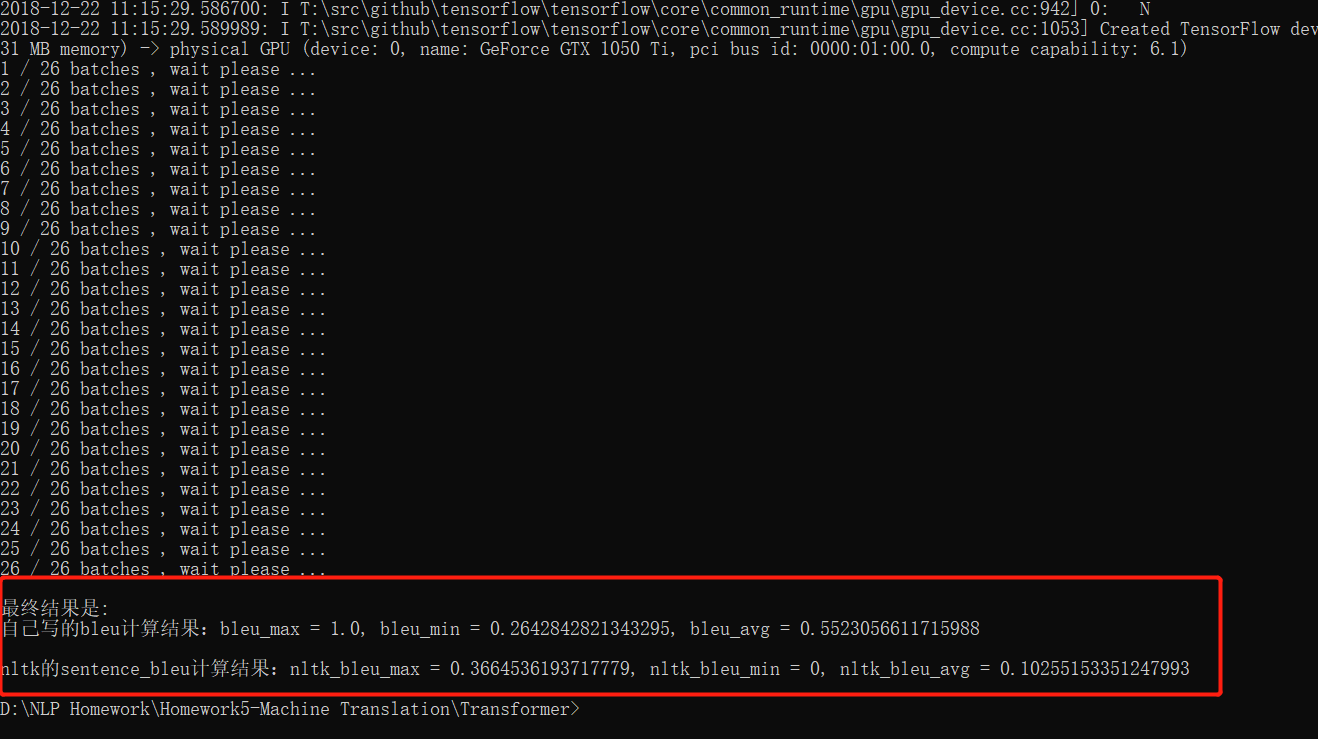
②Decoder：在进行词向量查询和位置编码处理后，进行六个层的堆叠，每个层的第一个MultiHead-Attention需要进行屏蔽(mask)，第二个MultiHead-Attention则不需要mask，然后再加一个前馈神经网络。

具体见代码中的详细标注。

**3 实验结果**

实验结果可参见第一页总的结果说明，这里是实际的截图





Transformer运行比Attention慢，因此打印出了具体的batch信息

1. **代码运行说明**

(1) 进入 ./Transformer/ 目录下： cd Transformer

(2) python main.py --is\_training 0 (运行会比Attention慢，运行过程会打印出batch信息，请稍安勿躁)