****

**《自然语言处理》**

**实验报告**

**（期末作业）**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **学院名称** | **：** | 数据科学与计算机学院 | | | | | |
| **专业（班级）** | **：** | 17计算机教学3班 | | | | | |
| **学生姓名** | **：** | 何家栋 | | | | | |
| **学号** | **：** | 17308054 | | | | | |
| **时间** | **：** | 2019 | 年 | 12 | 月 | 21 | 日 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  | **成绩** | **:** |  |
| **项目** | **：** | **基于深度学习的中英机器翻译** | | | |

### 一、 实验目的

(1) 通过编写深度神经网络，了解对应的深度学习框架；

(2) 通过编写深度神经网络翻译模型，了解机器翻译对应的神经网络架构；

(3) 通过编写带注意力机制的神经网络语言翻译模型，了解注意力机制在神经网络中的应用方式；

### 实验要求

1. **平台**

Pytorch，Tensorflow或者keras

1. **模型要求**

* 两个lstm分别作为Encoder和Decoder
* 实现基于注意力机制的机器翻译
* 使用分词工具
* 改变teacher forcing ratio，观察效果
* Beam Search策略

1. **评估指标**

BLEU值

1. **词向量**

随机初始化或者自选与训练词向量

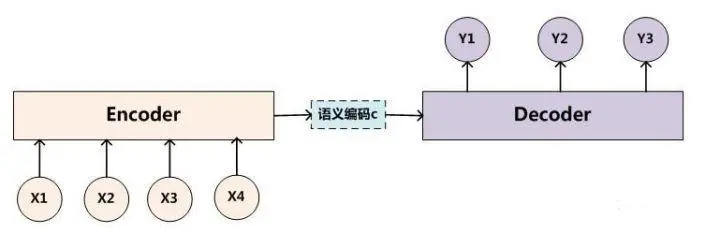
1. **设备**

CPU/GPU

### 实验原理

1. **基础的Seq2seq模型：**

实际上就是一种通用的编码器解码器模型，通过encoder将输入的序列变成一种隐藏状态，或者对隐藏状态进行进一步的变换，来作为输入句子的语义信息，而decoder通过隐藏层信息以及之前的句子的状态来获得下一步的句子翻译结果。整个的模型可以看作是：



其中主要是两个阶段，一个是编码阶段，一个是解码阶段。

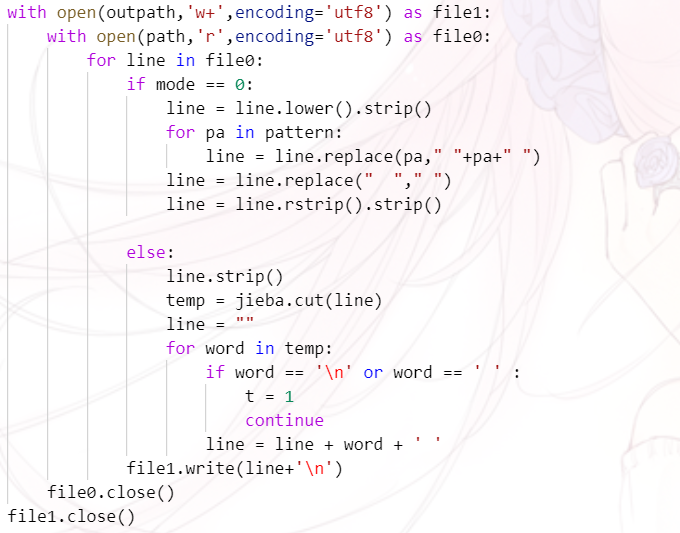
* 1. 编码阶段：主要是输出语义信息，每一个时刻的隐藏层状态又是与输入的句子以及上一个隐藏状态有关的，实际上就是一个lstm层的事情。
  2. 解码阶段：我们需要基于编码器的语义信息以及上一次的生成的输出序列来得到下次的预测输出序列，即

1. **注意力机制**

通过encoder的输出以及上一个句子状态来得到注意力的权重，而语义是在encoder的输出的基础上再和注意力权重点乘得到的，再由这个新的语义去生成下一步的预测。

### 实验过程

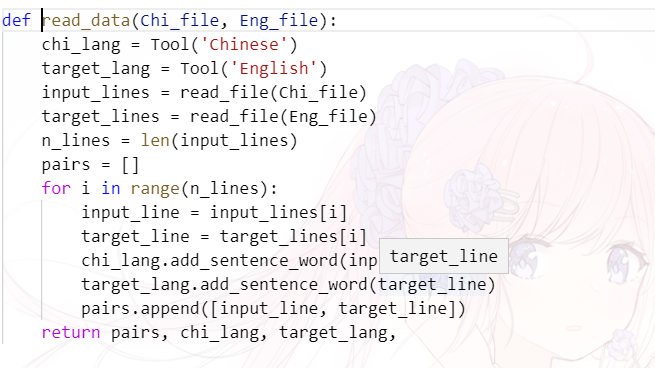
* 1. **预处理**
* 去除句子中的特殊符号，并且对句子进行分词处理



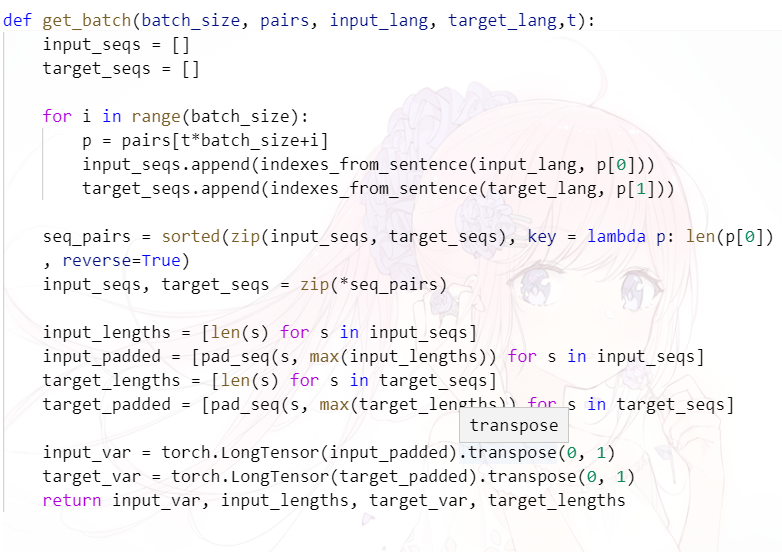
* 进行训练之前，需要对句子进行一个数据化以及填充处理，建立字典等等，我这里建立了一个工具类，保存对应数据集的字典，可以对句子进行分割，把数字数组转换成单词句子



* 在读取数据时，会先建立对应语言的工具，并且返回对应目标以及原的句子对：



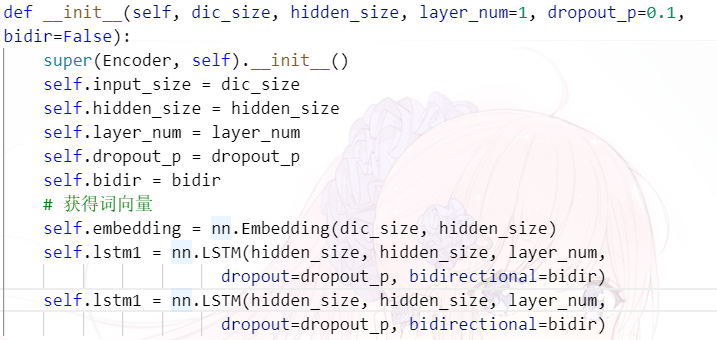
* 需要注意的是我这里将句子转换为对应的数组是在创建输入的batch时再进行填充的。先是转换为对应的数组，再转换成为数组的时候就在结尾加上EOS标志，然后进行padding，将该batch统一的长度变成最长的那个，中间填上PAD。



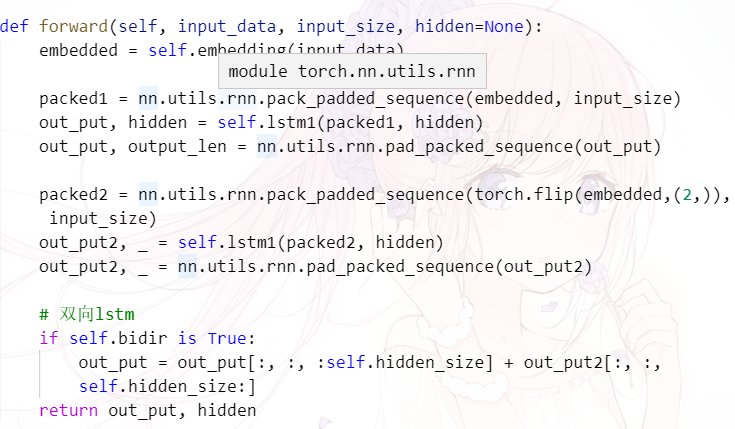
* 1. **编写Encoder**

我这里的神经网络比较浅，第一层时embedding层，用于将数组转换成为词向量，然后是两层lstm实现双向lstm。

* Encoder初始化过程



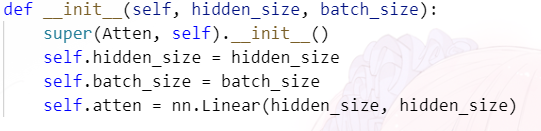
* Encoder迭代过程，首先输入到Embedding层，然后对输出的词向量进行pack，再输入到两个lstm层中，由于是双向lstm最终输出的时候，把两个方向的输出进行求和。



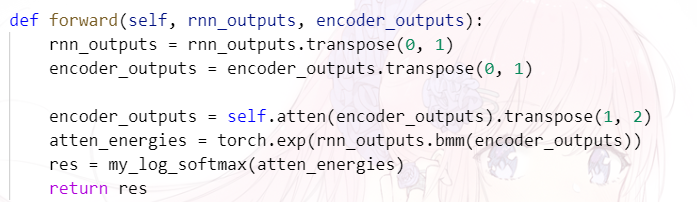
* 1. **编写带注意力机制的decoder**

这里的注意力机制实际上也是类似一个神经网络，包含一个线性层将Encoder的输出进行变换，这并不改变输入的形状，起到一个类似全连接层的作用，根据给出的注意力机制的原理，我们将其输出与Decoder层的隐藏层进行点乘，取soft\_max，这样得到每一个位置的注意力权重。然后把得到的注意力权重作为一个新的隐藏层状态与输入的Encoder的输出进行点乘及线性变换，然后与decoder的输出进行结合得到最终的输出。

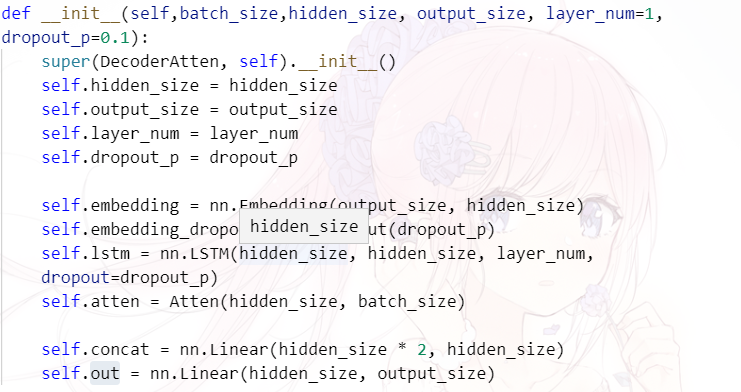
* 建立Attention机制：原理以及建立过程已经在上面描述了，主要只是建立一个线性层：



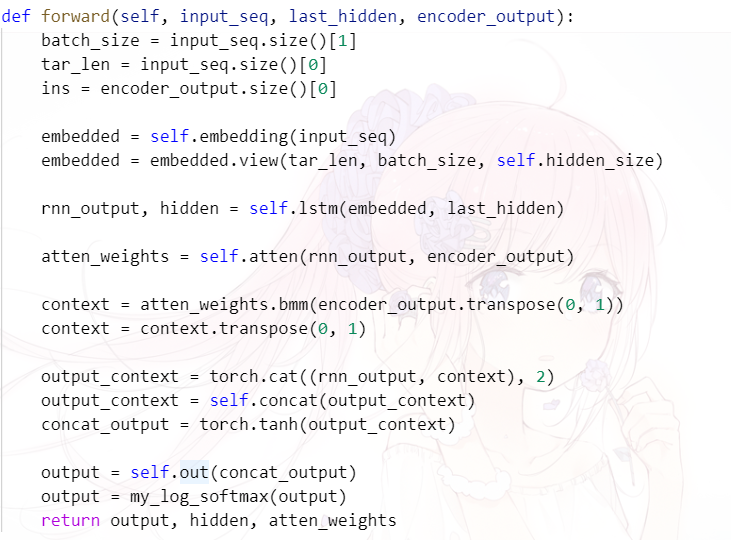
* 注意力机制的迭代：其实就是输入到神经网络，然后点乘，再取softmax，最后输出注意力权重：



* 然后是构建Decoder：首先初始化的时候，建立一个embedding层，带有dropout防止过拟合，然后接着的是lstm层以及atten层，然后是负责把两个隐藏层的输出进行联合，得到一个新的隐藏层状态，最后再经过一个全连接层后，进行softmax得到最后的输出。



* 接着就是Decoder的迭代过程：实际上就是把输入的数据放入到Decoder的神经网络中，不过需要注意的是，由于这里会进行几次点乘，每一次我们都需要注意其中的两个矩阵的size是否匹配，要进行向量对齐。



* 1. **训练过程**

我这里分了两个函数来写，一个是训练的外层函数，主要用于初始化Encoder、Decoder以及各种参数、优化器，还要进行数据的读取，获得每一次需要训练的batch的数据，以及epoch循环。内层的训练函数主要是，通过句子长度次数的迭代来获得整个句子的最后翻译结果，Decoder初始的句子状态，给的是全1，表示SOS，句子开始。

* 外层训练函数：
  + 初始化各种参数以及Decoder、Encoder还有优化器



* + 循环部分：每次训练batch数个句子，总共epoch轮，并且保存模型以及记录loss的情况

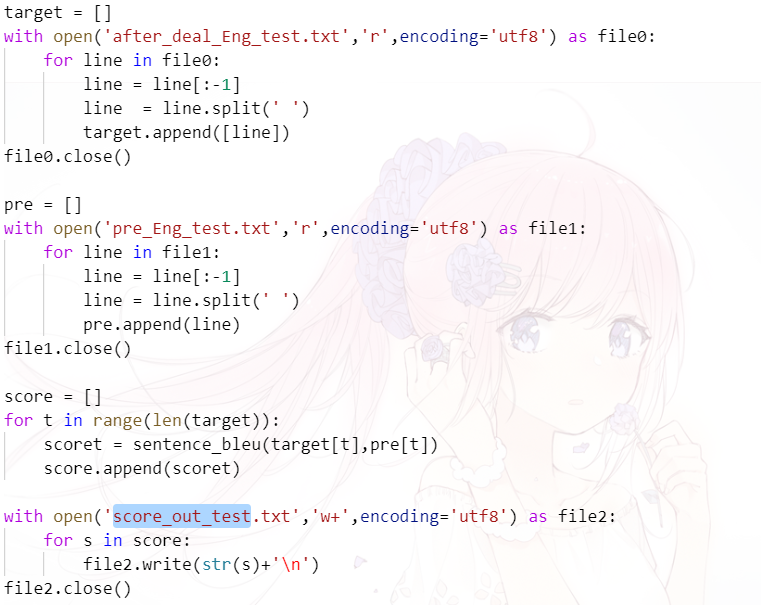


* 1. **预测部份**

实际上与训练部分并没有特别大的区别，只是多了模型的加载以及将得到的预测数组进行转置后转换成为对应的句子。

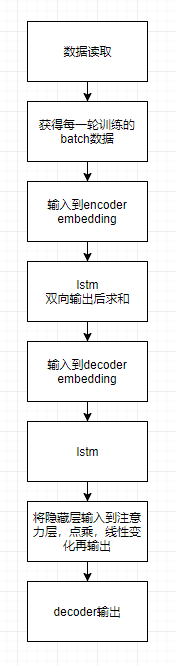
* 1. **计算得分部分**

这里主要就是运用nltk包进行bleu的分析，并没有什么特别的原理部分以及关键的东西。



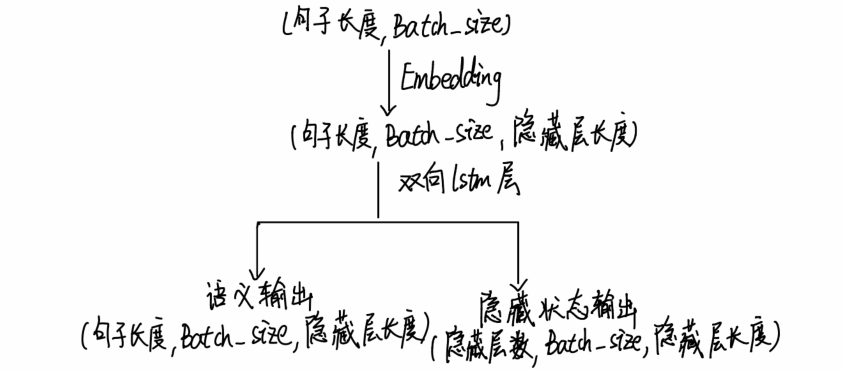
**五、实验过程与结果**

**整个运行的流程：**

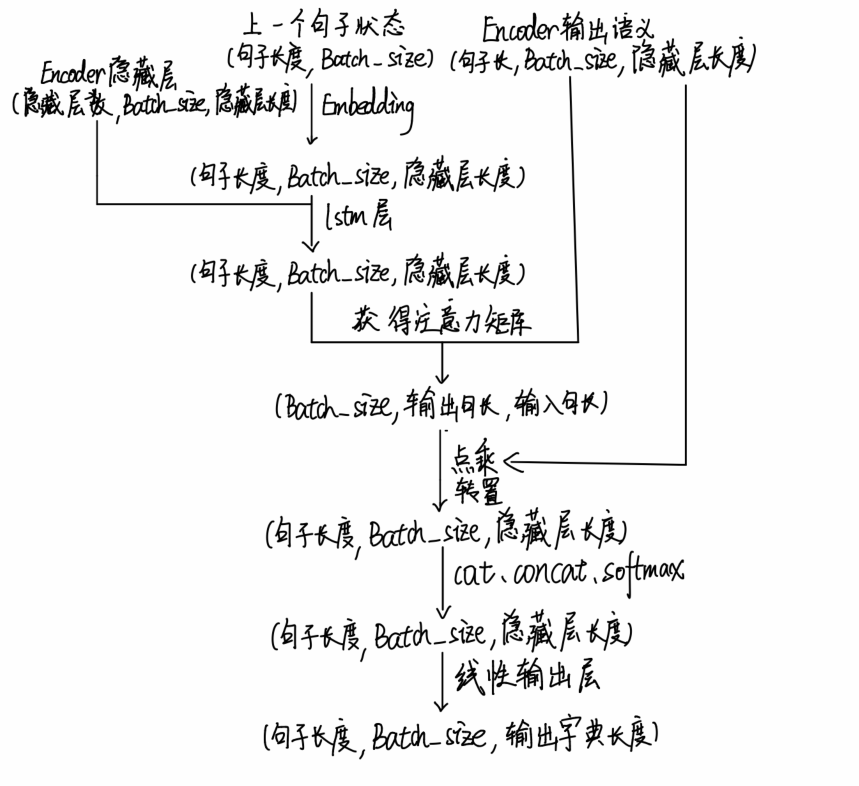


**输出的维度：**

**Encoder中的维度变化：**



**Decoder中的维度变化：**

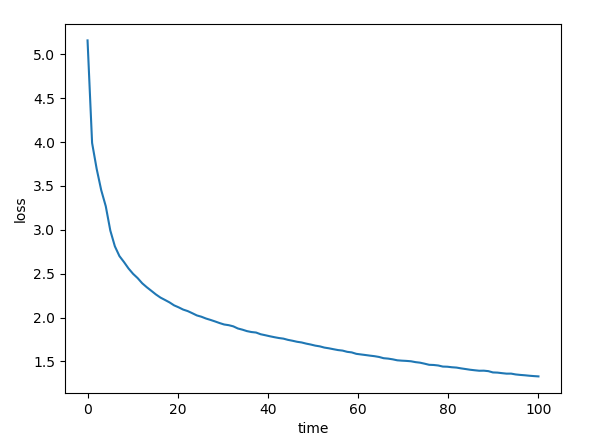


**实验结果：（这里主要是对比两个不同的teacher ratio下的情况）**

1. **Teacher ratio为0.7时**

**测试的平均bleu值为1.907327579676822e-158，可以看出来预测的情况并不是很理想。**

**迭代收敛情况：**

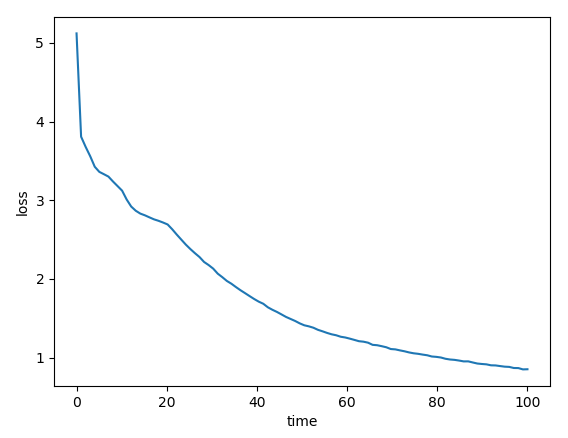


**我们可以看见在迭代的前20次中，loss下降速度很快，但是到了80次左右，loss渐渐趋于稳定，收敛的速度变慢了很多。在这个程度上，估计在110次迭代左右就完全收敛了。而且整个的学习曲线比较平滑。**

1. **Teacher ratio为0.3时**

**测试集上的bleu值为1.907327579676822e-158，和ratio为0.7的时候没有区别，但是也说明，学习效果不好，翻译出来的句子效果不好。**

**迭代收敛情况：**



**我们可以看见刚开始的时候loss的变换上下浮动的情况比较明显，但是最后的收敛点也是差不多的。**

**输出的预测样例对比：**

1. **Teacher ratio为0.7:**

**预测的结果：**

**预测的标准答案：**

1. **Teacher ratio为0.3:**

**预测的结果：**

**预测的标准答案：**

**六、实验环境要求**

**Python3.6.6 pytorch numpy matplotlib**