# 自然语言处理实践作业2

马煜1120210526

代码文件结构和使用方法：

Exp\_Train\_bert&ernie是预训练的主函数入口

Exp\_Train\_bert2trans是蒸馏的主函数入口

Exp\_Train是Bi-LSTM和transformer的主函数入口，需要改变其中的part超参数

1. **分别采用如下模型/方法实现汉语短文本分类：**

**Bi-LSTM**

**Transformer**

**预训练的Transformer**

1. **实验要求重述和代码解读**

三部分实验均使用一致的超参数如下所示

embedding\_dim = 300     *# 每个词向量的维度*

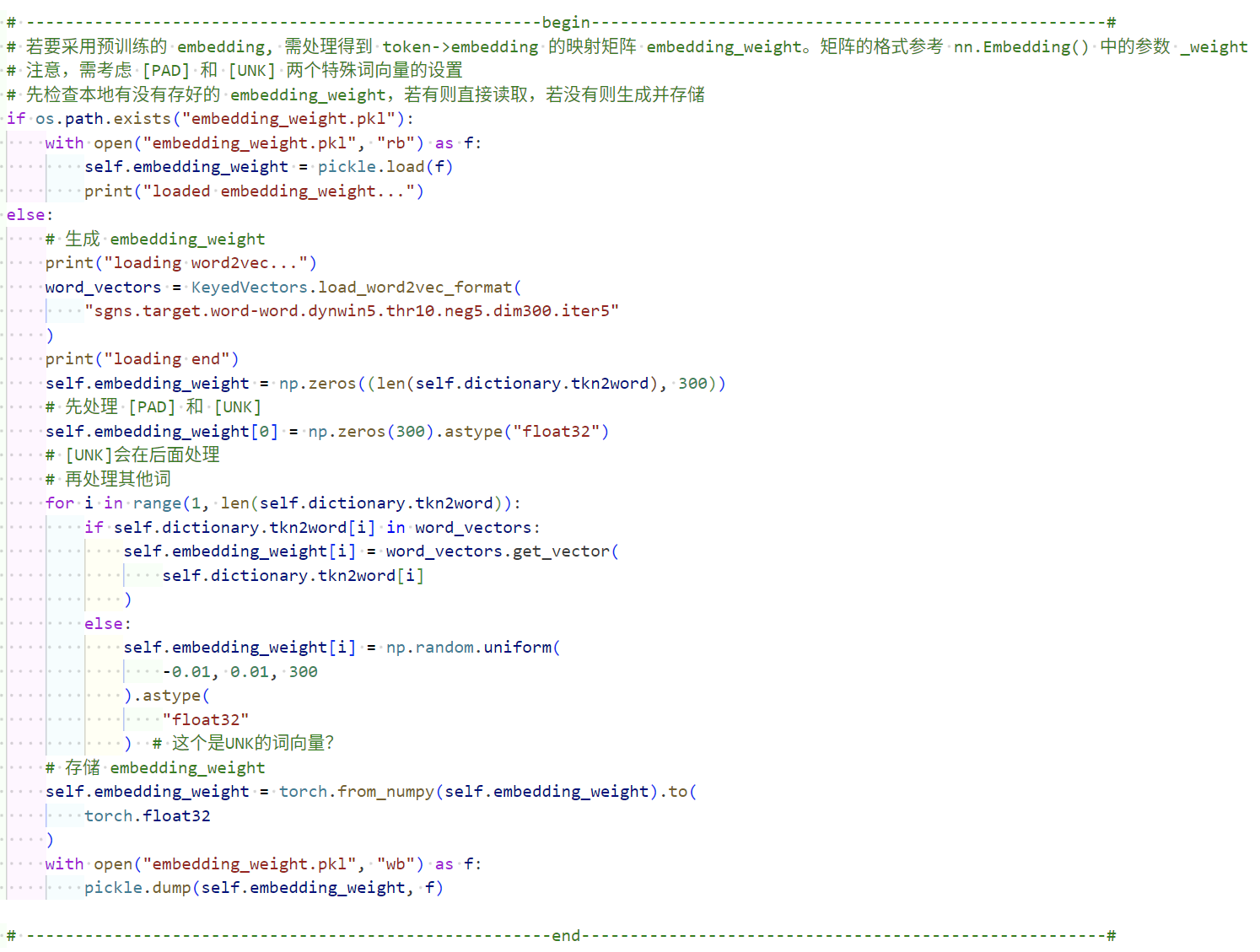
    max\_token\_per\_sent = 50 *# 每个句子预设的最大 token 数*

    batch\_size = 64

    num\_epochs = 5

    lr = 1e-3

1. **Bi-Lstm部分**
2. **预训练词向量的加载**



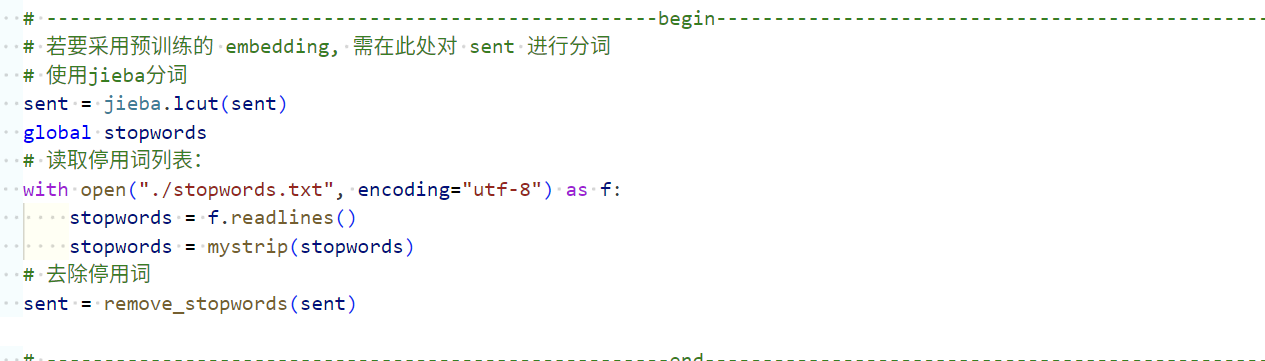
仿照第一次实践作业中加载词向量的方法，注意的点是词向量训练时的维度是300。

self.embedding\_weight[0] = np.zeros(300).astype("float32")

这一行代码是在处理[PAD]

1. **分词部分**

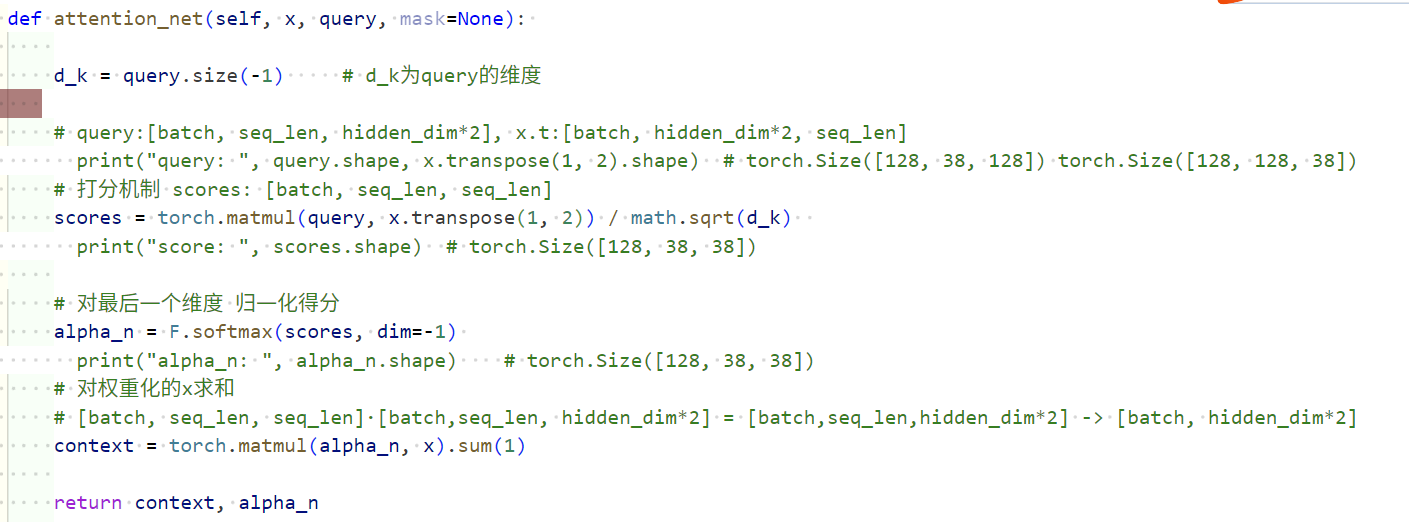
仿照第一次实践作业

均调用第一次实践分词部分的函数，因为使用预训练embedding所以必须分词，将整个句子变为词token组成的句子

1. **隐藏层的同处理方法，共7种**

本部分代码如下：

首先是对不同处理方法对应的线性分类器（全连接网络）

 **以上是对隐藏层输入进行自注意力机制的网络层**

**以下是forward部分：**

****

****

**下面进行解释：**

**第一种方法：同层隐藏输出计算平均池化，层间直接拼接，最后用线性分类器分类**

**第二种方法：不池化，直接将每一层最后的一个隐藏结点进行拼接，最终拼接为[(batch\_size,nlayers\*2)]，只保留每一层正向和反向最后一个时间步的状态**

**第三种方法：将每一层的正向和反向的隐藏层进行拼接，最终拼接为[(batch\_size,nlayers\*2\*hidden\_size)]，保留所有层数的状态**

**第四种方法：直接将输出线性相加（最基本的LSTM）**

**第五种方法：将每一层的正向和反向的隐藏层进行最大池化pooling**

**第六种方法：将每一层的正向和反向的隐藏层进行平均pooling**

**第七种方法：将每一层输出结果放入自注意力机制中（原理同attention is all you need）**

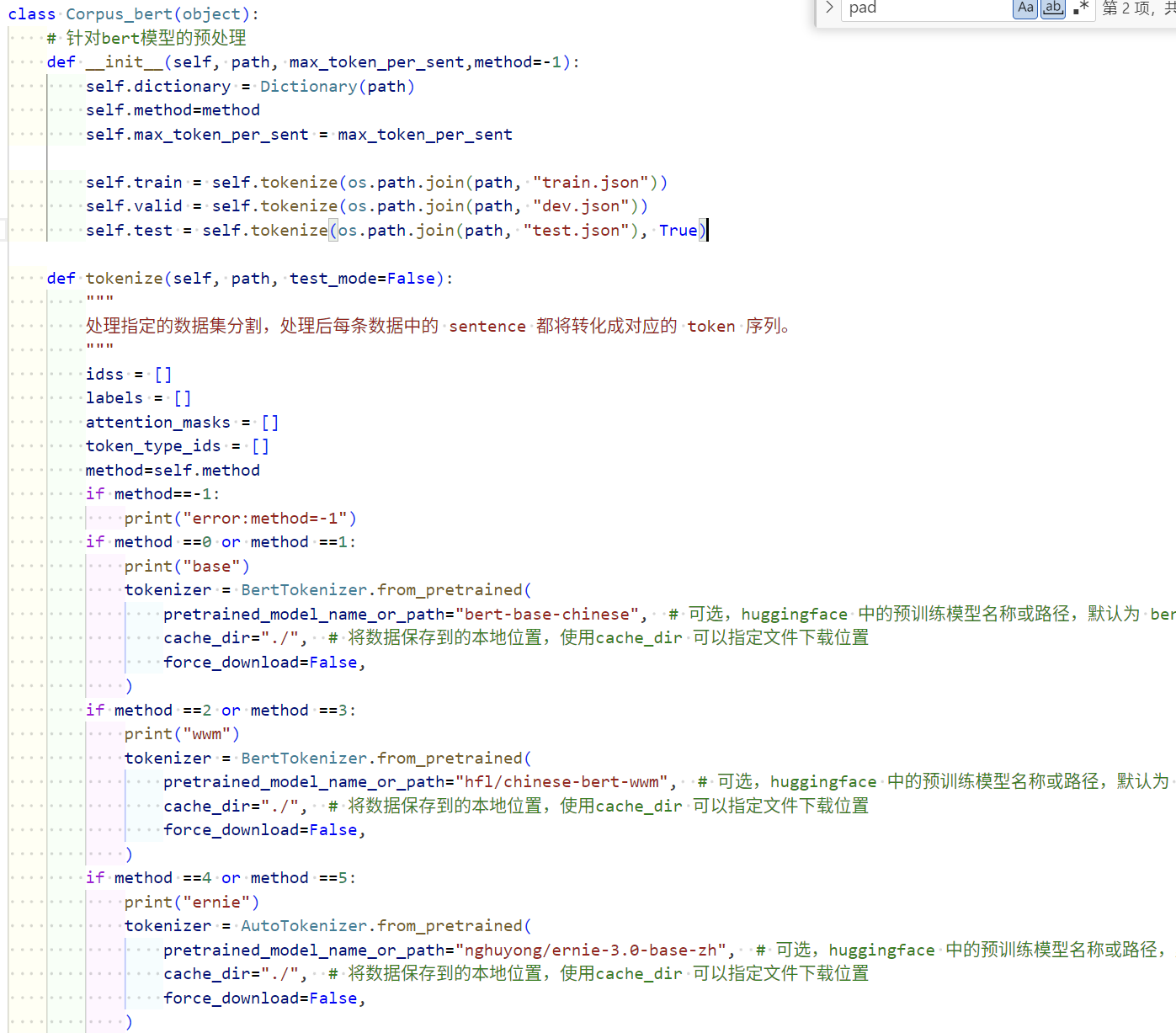
1. **Transformer部分**
2. **预训练词向量加载和分词同bi-lstm**
3. **隐藏层处理方法，共三种**

**本部分代码：**

**可以看到，在nhead=15, nlayers=2情况下，在第一种方法是将输出对所有token求和，第二种方法是将输出对所有token求平均，第三种方法是只使用最后一个token的输出**

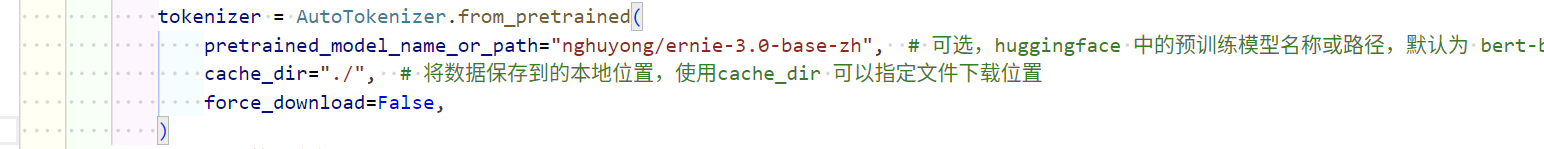
1. **预训练transformer部分**
2. **Token化**

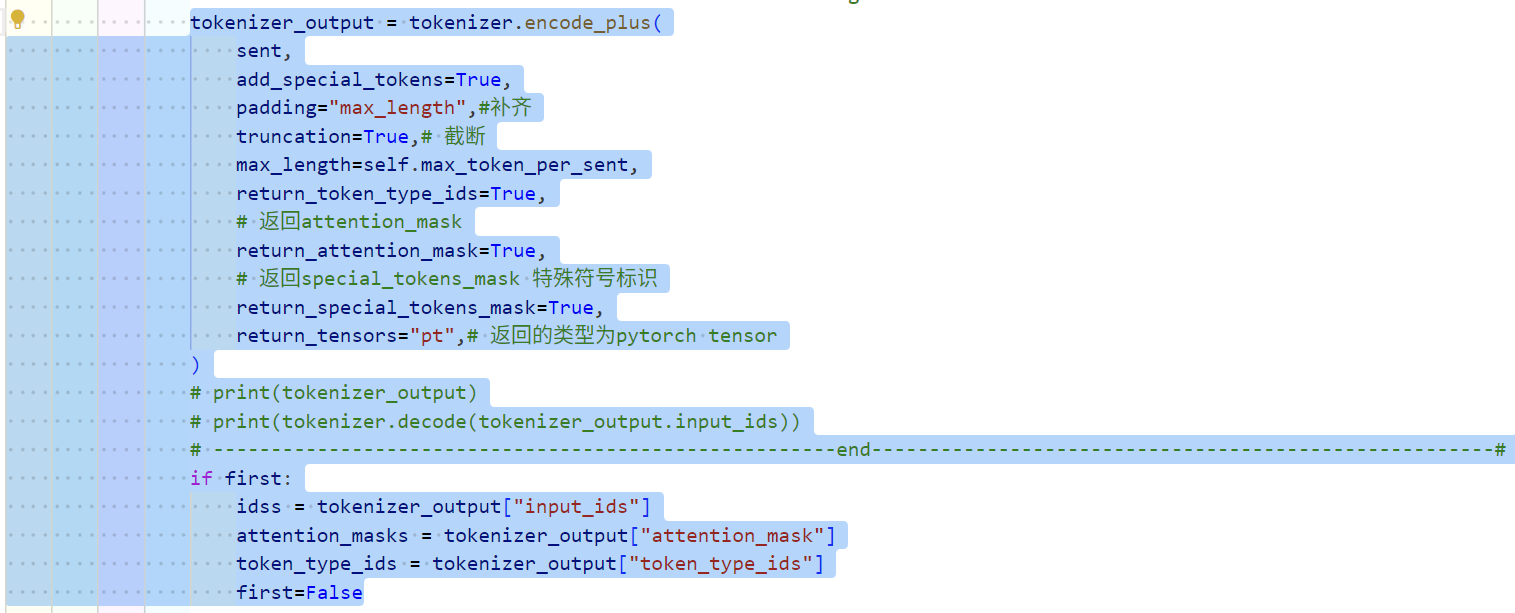
**在bi-lstm和transformer Token化的部分调用改为如下类：**

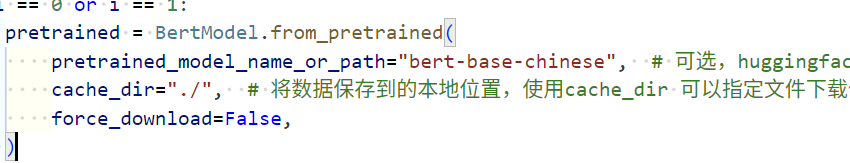


可以看到调用token化工具的语句是：



而得到token结果的语句是

1. **调用模型本体**

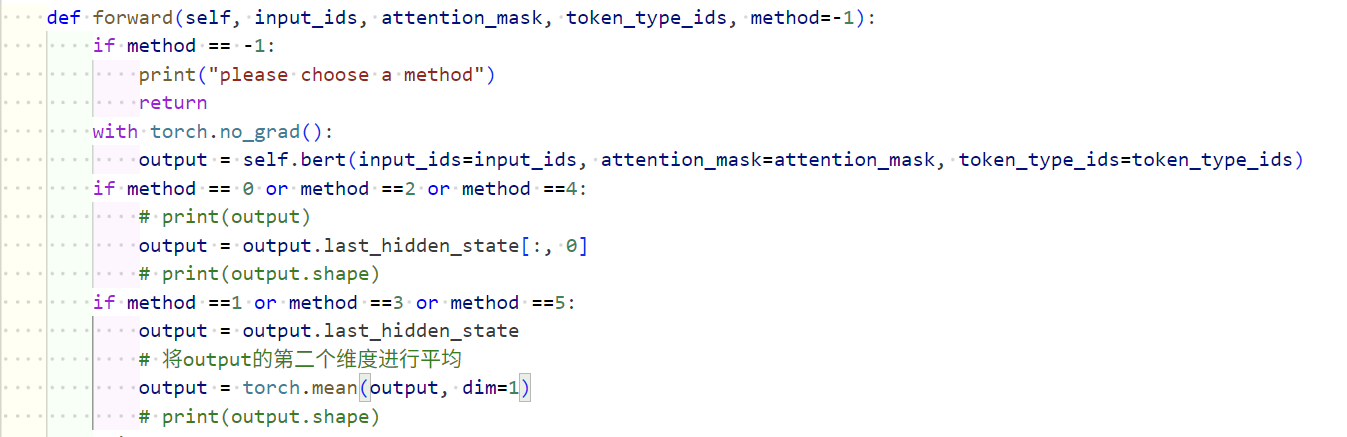
****

**注意要和token化工具对应上**

**训练时输入也变了：**

****

1. **对输出的不同处理方法：**

****

比较简单，就两种方法，一种是取[cls]输出，一种是取所有然后进行平均

1. **预训练transformer蒸馏**

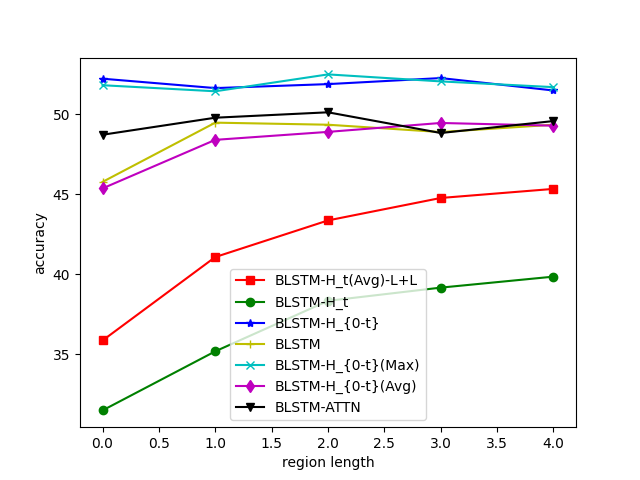
**加载老师网络和学生网络：**

**学生网络未进行初始化**

**训练过程如下:** ****

其中Dataloader加载每个batch的时候为了保证学生和老师的输入对应一致，把随机打乱FALSE了

1. **实验过程和结果比较**
2. **Bi-Lstm部分**

****

结果比较：

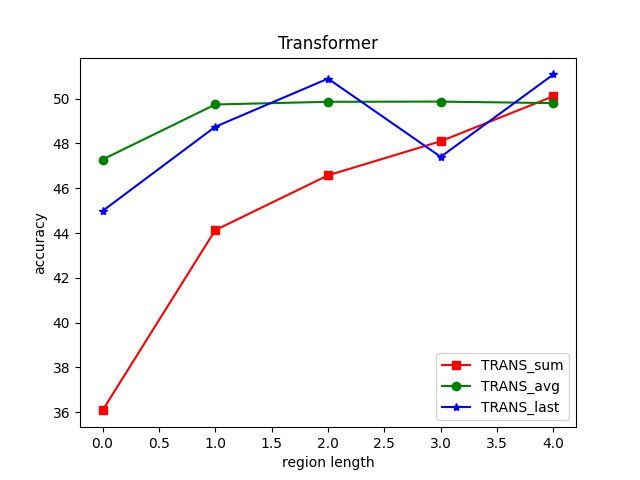
**第一个方法记为方法1，以此类推共7个方法**

方法1和方法2均和其他方法差距过大，可见对同层正反隐藏层进行平均池化以及取每层正反时间步最后一个结点并不能获得很多的句子信息

方法3和方法5，方法6做对比，可以看出，对每个隐藏层所有时间步输出的处理方法中，直接链接和进行最大池化效果较好，而且可以说非常好。而平均池化就欠点火候。

方法4和方法7相比，可以看出来添加注意力及之后的网络效果比直接使用输出的网络效果要稍好，体现了注意力机制的优越性

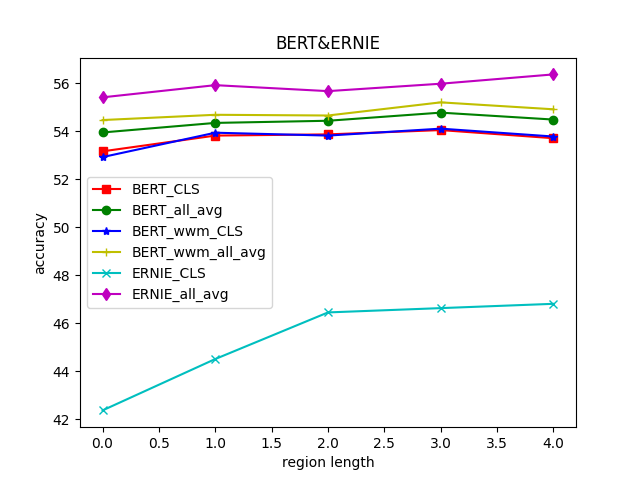
1. **Transformer部分**

****

对比可以看到，对每个token的求和不如求平均效果好，而取最后一个token的结果也能体现一定的句内信息（和transformer原理相契合）

至于transformer效果不如bi-lstm，有两种解释，一种是transformer参数大了，我电脑不太能训的动，于是多头设置了很多，实际上削减了transformer的能力。二是没太多时间调参数。

1. **预训练transformer部分**

在这部分，一开始选用了google的中文bert，随后想到了bert的好朋友ernie，最后根据ernie对词mask的特性找到了“chinese-bert-wwm”，也就是和ernie mask 策略一致的bert。对这三种预训练模型做了进一步的比较研究

可以看到ernie非常的两极分化，cls方法效果很差，感觉没学到多少句子的语义信息，而评价所有输出的方法效果极佳。整体来看三种cls方法成为了三种最差的方法，可见求平均更能反映句子整体信息。

此外，wwm方法的bert也比初始bert效果好，证明了ernie的mask策略的优越性。

1. **蒸馏**

这部分代码写出来了，但效果非常堪忧，我用了效果最好的ernie\_all\_avg作为教师模型，但训练巨慢，效果也不及学生模型原本的效果，还有待进一步调整。

Transformer\_sum第20个epoch的结果才到了51%将将到了transformer本身的性能水平

epoch: 19, train loss: 0.1221, train accuracy: 58.07%, valid accuracy: 51.04%

可能是因为用的蒸馏方法就是最简单的对结果进行软损失的求解，没有用教师模型初始化学生模型