**基于schema约束的SPO信息抽取任务**

**项目报告**

16307130194 陈中钰

1. **项目背景和概要**
   1. **项目背景**

* 信息抽取是从自然语言文本中抽取实体、属性、关系及事件类信息的文本处理技术；
* 信息抽取使得机器可以理解文本内容，是信息检索、智能问答等应用的重要基础；
* 信息抽取包括实体识别、关系分类等任务。
  1. **项目概要**
* 给定50种schema约束，其中有：

{"object\_type": "地点", "predicate": "祖籍", "subject\_type": "人物"}

* 输入句子text，句子分词和对应词性postag，如：

postag: [{"word": "南迦帕尔巴特峰", "pos": "ns"}, {"word": "，", "pos": "w"}, {"word": "8125米", "pos": "m"}]

text: "南迦帕尔巴特峰，8125米"

* 输出句子中符合给定schema的关系，以及该关系predicate的主体subject和客体object，包括其类型，如：

spo\_list: [{"predicate": "海拔", "object\_type": "Number", "subject\_type": "地点", "object": "8125米", "subject": "南迦帕尔巴特峰"}]}

* 1. **项目实现基本框架**

项目实现分以下三部分：

* 分类问题：输入text和postag，输出符合schema约束的关系predicate；
* 标注问题：输入text、postag和关系predicate，标注text中的subject和object；
* 输出spo：给定text和关系predicate，组合对应的subject和object，形成spo三元组，并补上对应的subject\_type和object\_type

1. **普通分类模型**

**2.1 代码组织（这部分的模型在p\_classification/cnn/文件夹中）**

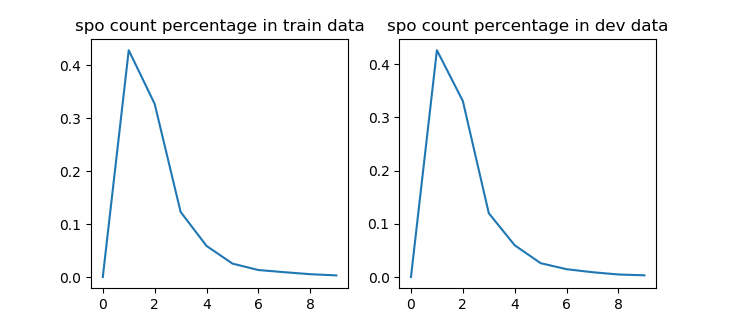
|  |  |
| --- | --- |
| **文件名** | **功能** |
| **config.py** | 在class中定义模型参数 |
| **dataset.py** | 读取、处理数据，利用fastnlp生成train\_data、dev\_data、test\_data和vocabulary，并用pickle导出 |
| **model.py** | 用pytorch定义CNN、RNN、LSTM、LSTM\_maxpool、RCNN模型 |
| **utils.py** | 定义用于计时的Callback类、BCEWithLogitsLoss的loss类、计算f1 score和metric类 |
| **train.py** | 定义trainer，用于训练、测试模型 |
| **visualize.py** | 从log文件中提取出loss、f1 score、precision、recall的值，并画出曲线 |

**2.2 基本数据统计**

* 数据量

|  |  |  |  |
| --- | --- | --- | --- |
| **all\_50\_schema/类别数量** | train\_data,json | dev\_data.json | test1\_data\_postag |
| **50** | 173108 | 21639 | 9949 |

* 50个关系中，有两个关系的predicate是相同的，因此仅用predicate是不能区分不同的spo，可以把subject\_type、object\_type和predicate连接起来，这样才能区分不用的spo
* 每个text所拥有的spo数量的百分比，unique的spo的数量百分比也呈现一样的分布。因此这个分类问题是多分类问题。



* 最大text长度均小于等于300，因此选取sequence length为320.

|  |  |  |  |
| --- | --- | --- | --- |
| train\_data,json | dev\_data.json | test1\_data\_postag | **sequence length** |
| 300 | 299 | 300 | **320** |

**2.3 数据处理**

**（见p\_classification/cnn/文件夹中的dataset.py文件）**

* 读取数据，用fastnlp的dataset类型实例化train、dev、test 数据
* 把各个dataset中的text，并按字分开
* 从train data中获取vocabulary，大小为8182
* 用上述vocabulary把train、dev、test data的字都变为index
* 在比sequence length短的text前面补’<pad>’对应的index，补到长度为sequence length
* 提取spo\_list中的关系，变成长度为50的向量，每位的值对应一个关系是否存在，0为不存在，1为存在
* 设置text向量为input，设置关系向量为target
* 用pickle导出train、dev、test data和vocabulary
  1. **模型参数**

**（见p\_classification/cnn/文件夹中的config.py）**

* **软编码**：模型所涉及的参数全部定义在这个文件的Config类里，包括模型参数、文件读写路径、文件名等。
* **基本分类模型一共实现了CNN, RNN, LSTM, LSTM\_maxpool, RCNN共5个模型，**以下是这些模型的参数：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| model | CNN | RNN | LSTM | LSTM\_maxpool | RCNN |
| embed\_dim | 128 |  |  |  |  |
| kernel\_sizes | (3, 4, 5) |  |  |  |  |
| kernel\_num | 128 |  |  |  |  |
| in\_channels | 1 |  |  |  |  |
| dropout | 0.5 |  |  |  |  |
| num\_layers |  | 1 | | | |
| hidden\_dim |  | 256 | | | |
| class\_num | 50 | | | | |
| sequence\_length | 320 | | | | |
| optimizer | Adam(lr=1e-3, weight\_decay=0) | | | | |
| patience | 20 | | | | |
| batch\_size | 64 | | | | |

* 1. **模型结构**

**（见p\_classification/cnn/文件夹中的model.py）**

**一共实现了CNN, RNN, LSTM, LSTM\_maxpool, RCNN共5个模型。**

* **模型一CNN：**input先过一个embedding层，再过一个卷积层。卷积核大小为(3, 4, 5)，有100个。过了卷积层后用ReLU激活，然后max pool。再把3个卷积核的结果连起来，然后dropout，最后过全连接层，然后输出.

**conv**

**ReLU**

**max poll**

**fc + dropout**

* **模型二RNN：**1个embedding层+1层双向RNN+1个全连接层

**X1**

**……**

**h1**

**hn**

**X1**

**h1**

**hn**

**X1**

**h1**

**hn**

**hidden**

**fc**

* **模型三LSTM：**1个embedding层+2层双向LSTM+dropout+1个全连接层

**input**

**embedding**

**BiLSTM**

**hidden**

**fc**

* **模型四LSTM\_maxpool：**1个embedding层+2层双向LSTM+max pool+dropout+1个全连接层（这个模型介于LSTM和RCNN之间，只是比LSTM多了一层max pool）

**input**

**embedding**

**BiLSTM**

**output**

**fc**

**max pool**

* **模型五RCNN：**1个embedding层+1层双向LSTM+1个线性层+max pool+1个全连接层

**input**

**embedding**

**BiLSTM**

**output**

**+**

**embeds**

**fc**

**max pool**

**embeds**

**linear**

**2.6 其他实现**

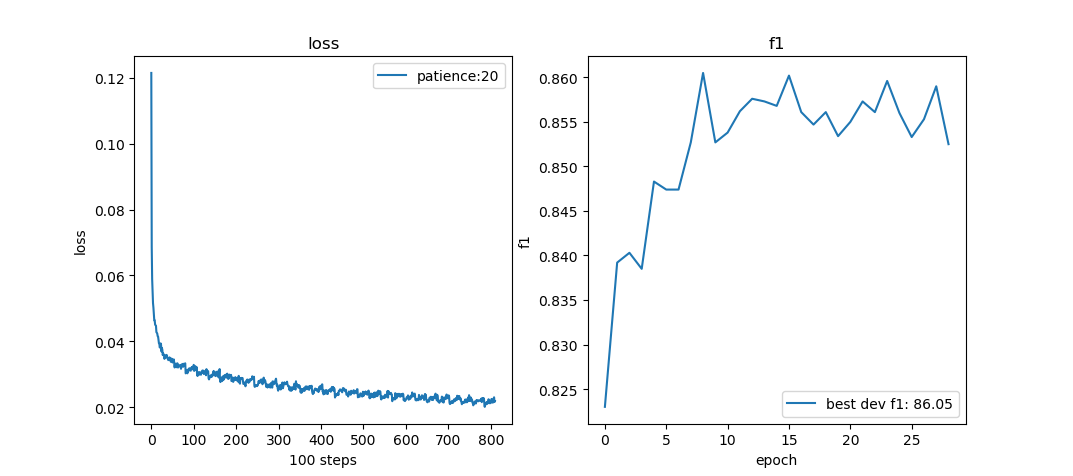
**（见p\_classification/cnn/文件夹中的utils.py）**

* 基于fastNLP的Callback，实现了on\_epoch\_end的计时功能，可以用于比较不同模型的运行时间。
* 基于fastnlp的BCEWithLogitsLoss实现，封装了pytorch的F的binary\_cross\_entropy\_with\_logits，适用于多分类模型的loss计算
* 基于fastnlp的f1 scrore的metric实现，封装了batched的f1 score的向量运算。
  1. **Trainer和Tester实现**

**（代码请看train.py）**

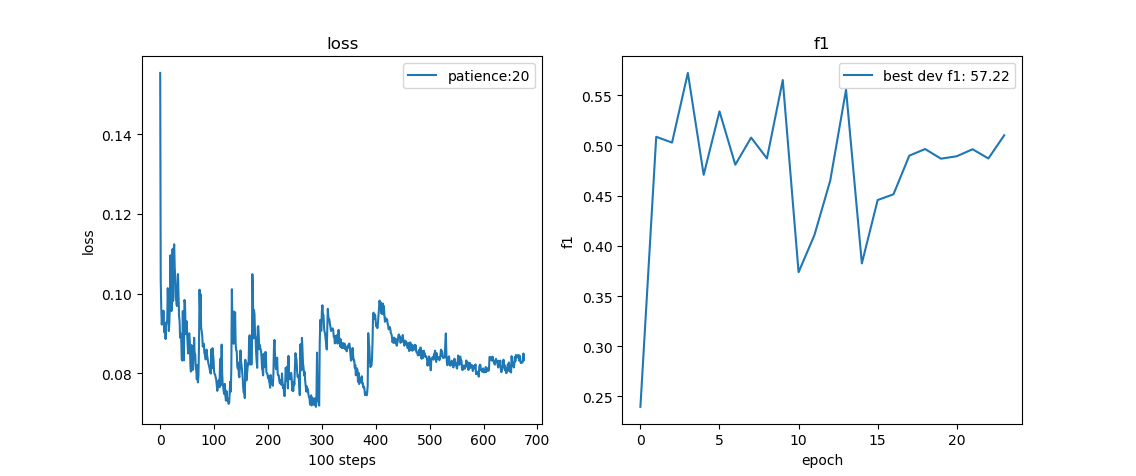
* 用pickle导入trian\_data、dev\_data、test\_data和vocabulary
* 根据config中的task\_name参数来定义对应的模型，并导入对应模型所需的参数
* 定义Adam的optimizer
* 定义计时Callback、EarlyStop的Callback、
* 定义BCEWithLogitsLoss的loss、f1 scrore的metric
* 然后定义Trainer，进行训练
* 最后定义Tester，由于test data并没有给出spo\_list，因此是在dev data上进行测试
  1. **模型结果**
* 模型一CNN：

|  |  |  |
| --- | --- | --- |
| 运行时间 | best epoch | best f1 |
| 6590s | 9 | 86.05 |



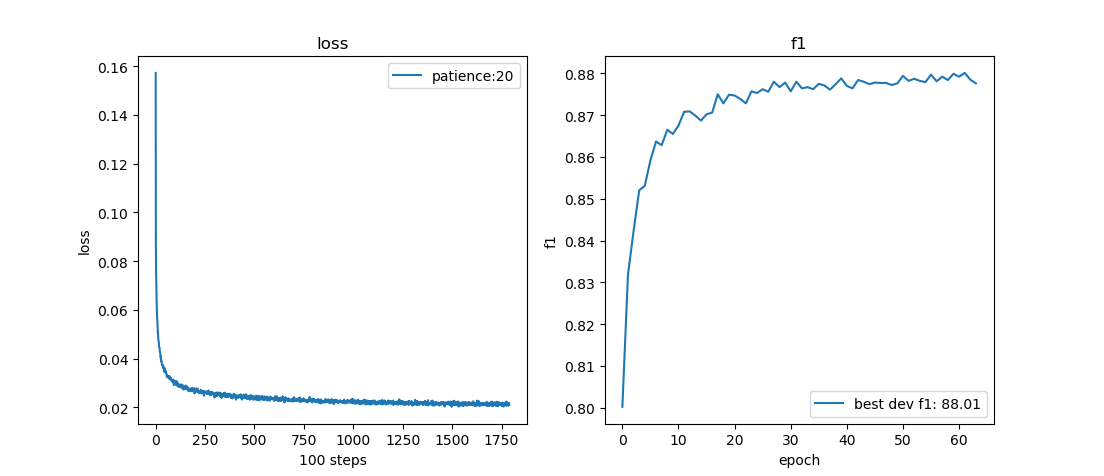
* 模型二RNN：

|  |  |  |
| --- | --- | --- |
| 运行时间 | best epoch | best f1 |
| 9273 | 4 | 57.22 |



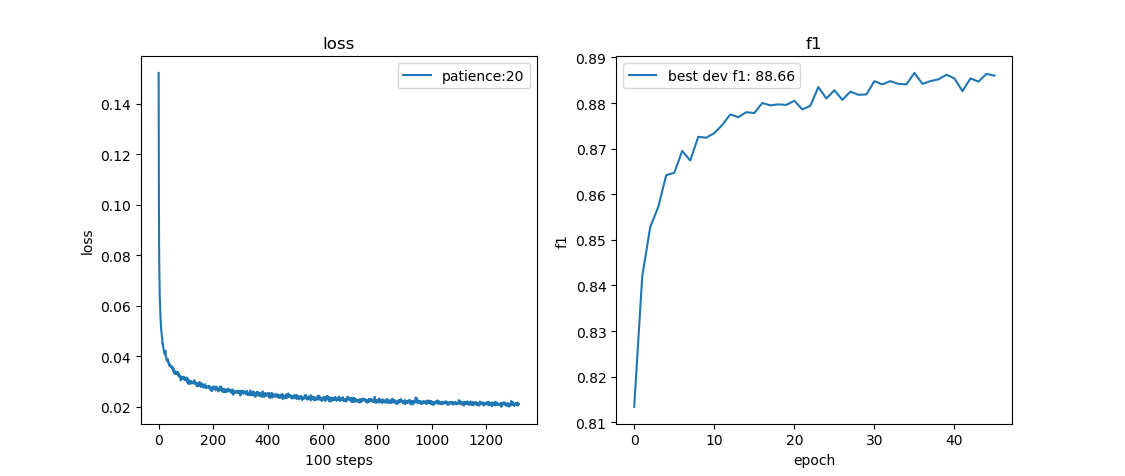
* 模型三LSTM：

|  |  |  |
| --- | --- | --- |
| 运行时间 | best epoch | best f1 |
| 23680s | 47 | 88.01 |



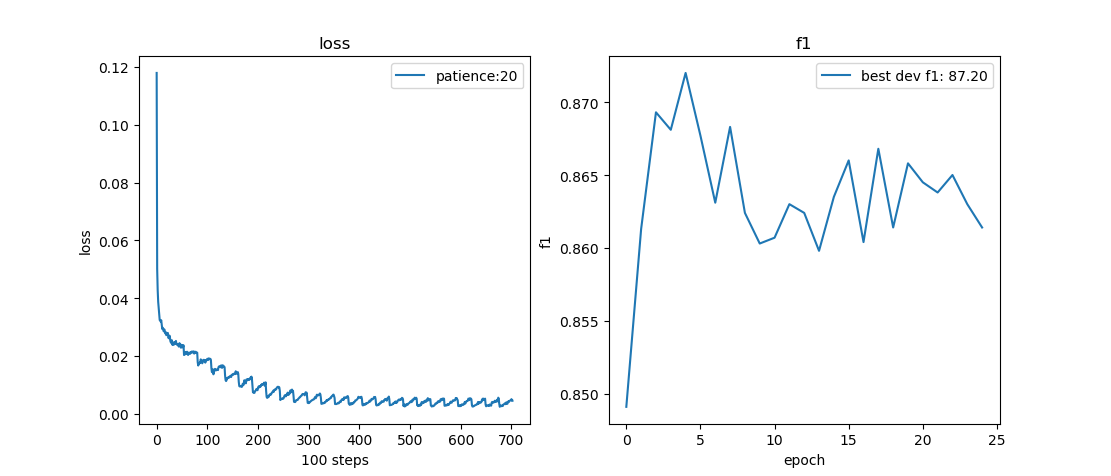
* 模型四LSTM\_maxpool

|  |  |  |
| --- | --- | --- |
| 运行时间 | best epoch | best f1 |
| 22570s | 41 | 88.66 |



* 模型五RCNN

|  |  |  |
| --- | --- | --- |
| 运行时间 | best epoch | best f1 |
| 14657s | 4 | 87.20 |



* 1. **模型比较**
* CNN相对于基于RNN/LSTM的其他模型，运行时间短很多，而效果还是很不错的
* 基于LSTM的模型比RNN效果有显著提高，因此在实际训练的时候基本上LSTM是最基本的RNN模型，而不会使用RNN、
* 尽管基于LSTM的模型比CNN运行时间更长，但是效果能有提高
* LSTM加上max pooling的结构，能缩短训练时间，并提高分类结果

1. **预训分类模型**

**（见p\_classification/bert/文件夹）**

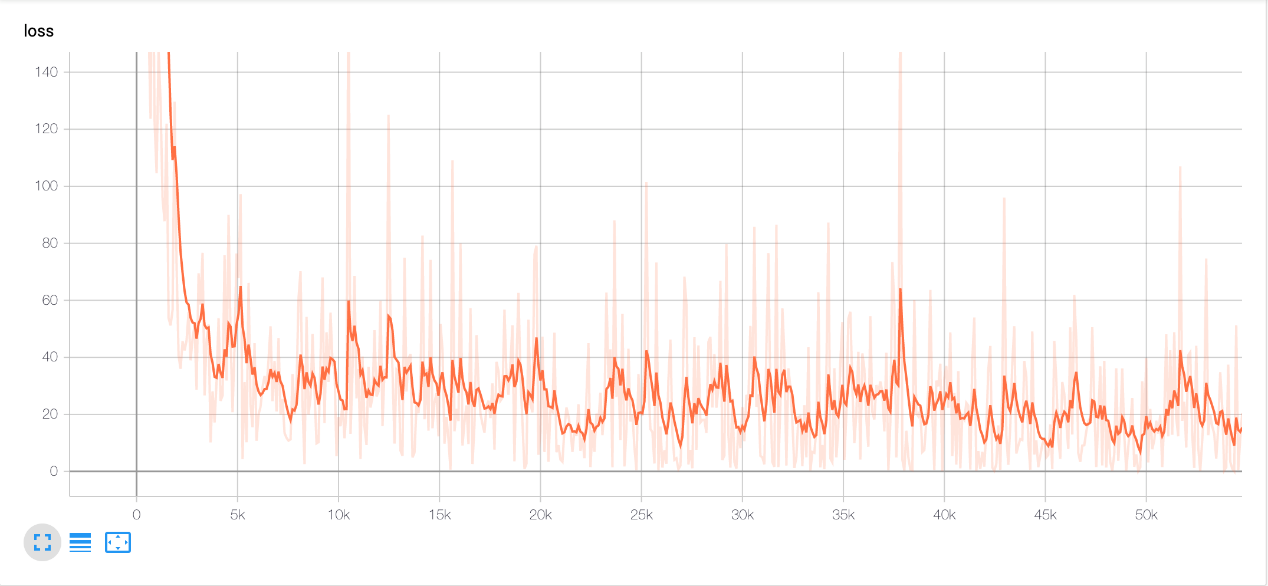
**3.1 模型实现**

* 使用了**bert**来进行多分类，在bert的中文预训练模型参数下finetune。
* 基于google-reserce的bert repository来实现，直接在上面改代码
* 主要基于run\_classifier.py修改，原文件是用于单分类的任务，通过把softmax loss改成sigmoid loss，以及一些其他小的修改，可以把单分类修改为多分类

**3.2 模型结果**

|  |  |  |  |
| --- | --- | --- | --- |
| 运行时间 | epoch | ckpt | best f1 |
| ~12h | 2 | 230810 | 89.48 |

* loss曲线：



**3.3 比较**

* 通过导入bert的中文预训练模型参数，再对数据集进行finetune，相比上文的普通分类方法，f1有所提高，但是训练时间显著增多

1. **分类模型ensemble**

**4.1 ensemble分析**

* 上述5个基本模型和1个bert预训练模型，除了RNN的f1只有57.22，其他的f1都在86~89，效果都很不错，因此剔除了RNN，用剩下的5个模型进行ensemble
* ensemble采用最普通的投票机制，对于1个text中的1个类，如果>=3个模型的结果为1，则ensemble结果为1，否则结果为0

**4.2 ensemble结果**

* 最终ensemble的在dev data上的结果为92.23，相对于5个模型的f1值的最大值提高了约为2

1. **预训标注模型**

**（见so\_labeling/bert/文件夹）**

**5.1 基本数据统计**

* 数据量（1条text+1个对应的关系，对应1种text的序列标注，因此1条+1个对应的关系就是一条数据）

|  |  |  |
| --- | --- | --- |
| train data | dev data | vocabulary |
| 303407 | 37987 | 8163 |

* train和dev data中的subject和object都是由text分割好的词连接而成，在训练模型的时候，可以考虑使用这个特性。
* 对应属性为空的数据条数，注意处理空的数据（忽略掉）。

|  |  |  |  |
| --- | --- | --- | --- |
|  | postag | text | spo\_list |
| train\_data.json | 50 | 0 | 13 |
| dev\_data.json | 10 | 0 | 0 |
| test1\_data\_postag.json | 4 | 0 |  |

**5.2 模型实现**

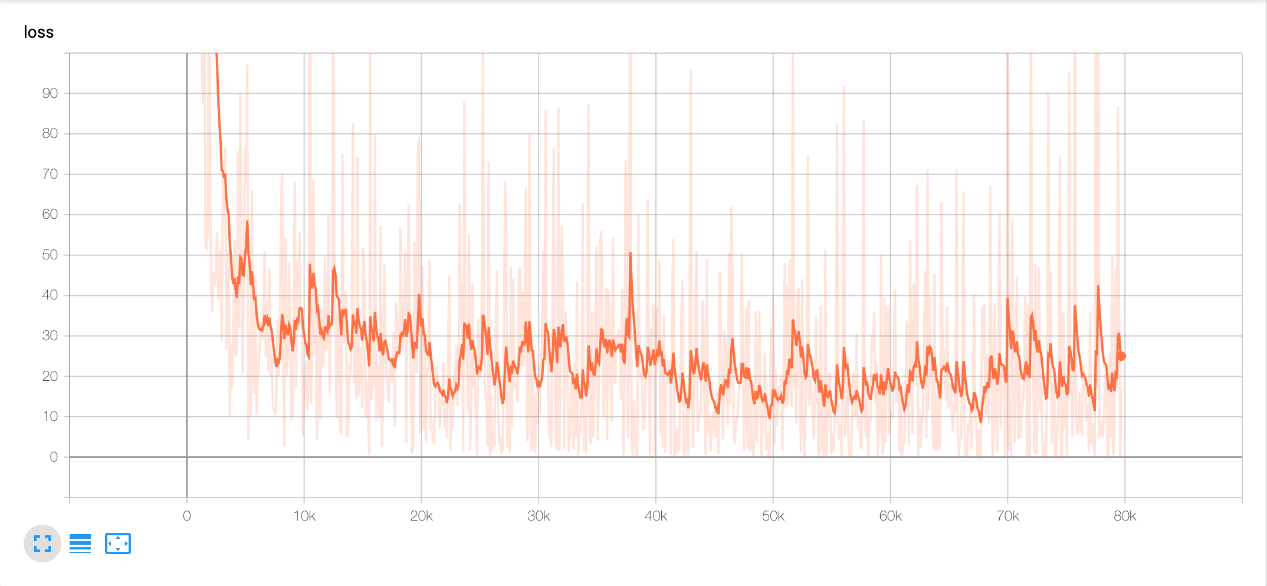
* bert模型输入text
* 从bert模型get pooled output，获得predicate，从bert模型get sequence output，获得序列标注**（标注采用的是百度baseline的BIO， 另外label为SUB和OBJ）**
* 计算predicate的loss和序列标注的loss，相加作为最终的loss
* 最终模型输出的是predicate和这个predicate对应的序列标注
* 而分类模型已经输出了text对应的predicate，接着就从中选出真正的predicate和对应的序列标注

**5.3 模型输出处理**

* 获得了text对应的predicate对应的序列标注后，可以从句子中找出subject和object，并利用subject和object只能是由分词拼成的特点去修正提取的subject和object
* 对同一个text的同一个predicate的subject列表和object列表进行全链接来形成spo，并且补上对应的type
* 输出结果

**5.4 模型结果**

* 训练时间：~18h
* loss曲线：



* 模型f1结果：f1为0.8134（这个结果是基于分类模型的分类ensembel结果，所以是任务的最终结果）

C:\Users\ECHOES\Desktop\result.png

1. **优化标注模型**

**（见so\_labeling/bilstm-crf/文件夹）**

* 1. **序列标注**

**（见so\_labeling/bilstm-crf/文件夹中的tagging.py文件）**

* 采用了BIESO的方式进行标注（有研究表明BIESO的方式比BIO更好），而label仍然是SUB和OBJ，一共9种标注
  1. **模型结构**

**（见so\_labeling/bilstm-crf/文件夹中的model.py文件）**

* 结构为Embedding + LayerNorm + BiLSTM(2 layers) + FC + LayerNorm + DropOut + FC + CRF
* 其中embedding由4部分组成，分别是按字分割的text的embedding(词典大小为8163)、按词分割的text的embedding（词典大小为379713）、text的词性列表的embedding（词典大小为26），以及predicate（词典大小为53，还有一个是’无’）的embedding
* 以上的词典大小均加上了’<unk>’和‘<pad>’
  1. **模型结果**
* 由于时间原因，这个模型还没来的及训练完。

1. **最终结果**

* 由于上述优化标注模型还来得及训练完，所以标注模型就只有上文的1个，因此上文中的结果就是最终的结果：f1为0.8134

C:\Users\ECHOES\Desktop\result.png

1. **思考**

* 把模型分成两步做，思路更加的明晰，但是由于后面模型的结果是基于前面模型的结果，因此整个模型的最终结果是近似于两部分的乘积，在两部分都约90的情况下，最终结果也就只有80左右，因此如果能构造一个不用分两步处理的模型就最好了。
* 这次实验不仅使用了pytorch+fastnlp的框架，还尝试去修改基于tensorflow的bert模型，对深度学习的框架有了更深的理解。