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

1.1 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。

1.2 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"}]}

1.3 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

2. 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

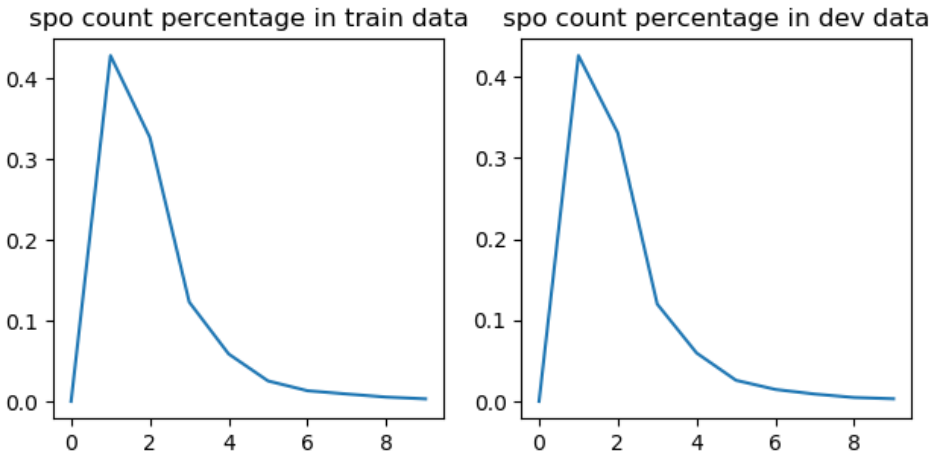
visualize.py	从 log 文件中提取出 loss、f1 score、precision、recall 的值，并画出曲线

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

2.4 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：

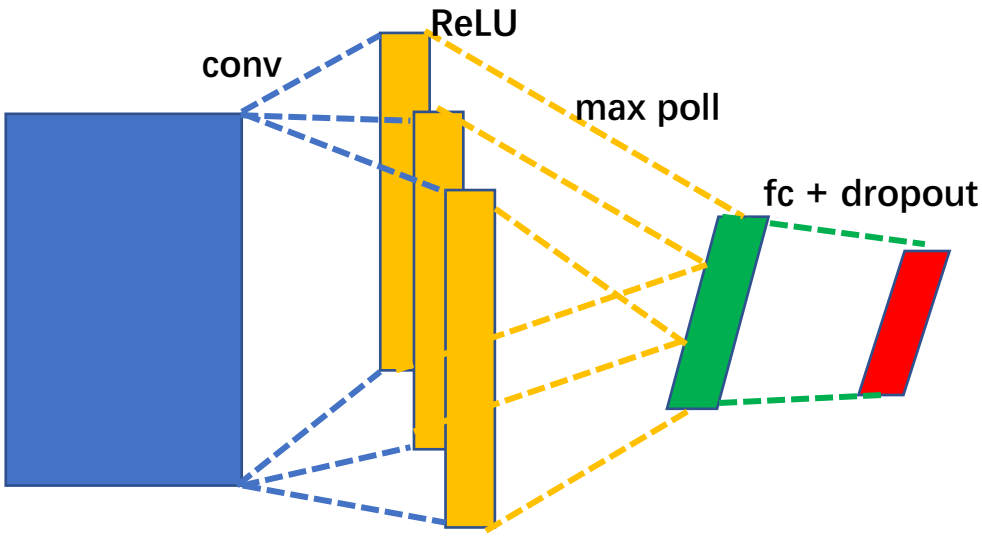
model	CNN	RNN	LSTM	LSTM_maxpool	RCNN
embed_dim	128				
kernel_sizes	(3, 4, 5)				
kernel_num	128				
in_channels	1				
dropout	0.5				
num_layers		1			
hidden_dim		256			
class_num	50				
sequence_length	320				
optimizer	Adam(lr=1e-3, weight_decay=0)				
patience	20				
batch_size	64				

2.5 模型结构

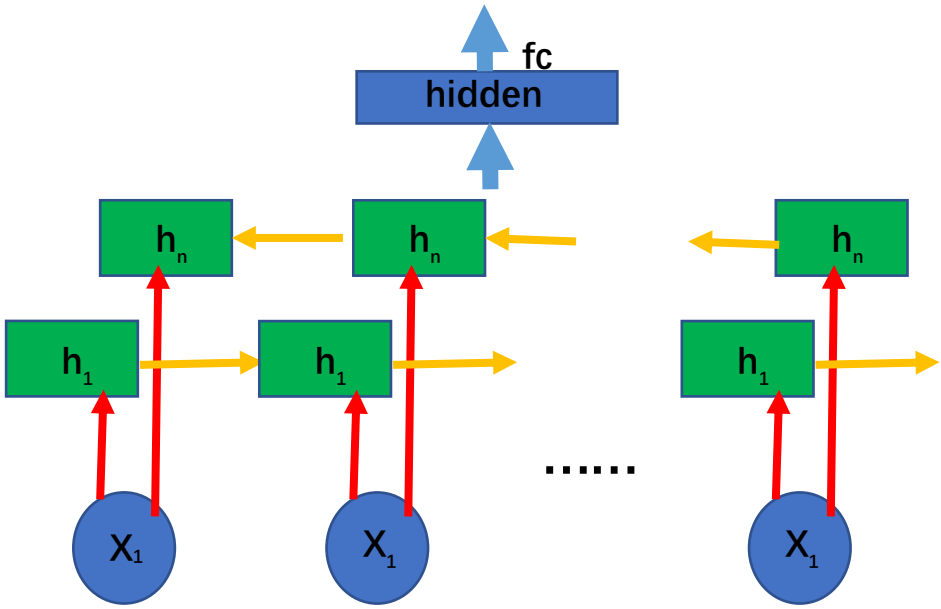
（见 p_classification/cnn/文件夹中的 model.py）

一共实现了 CNN, RNN, LSTM, LSTM_maxpool, RCNN 共 5 个模型。

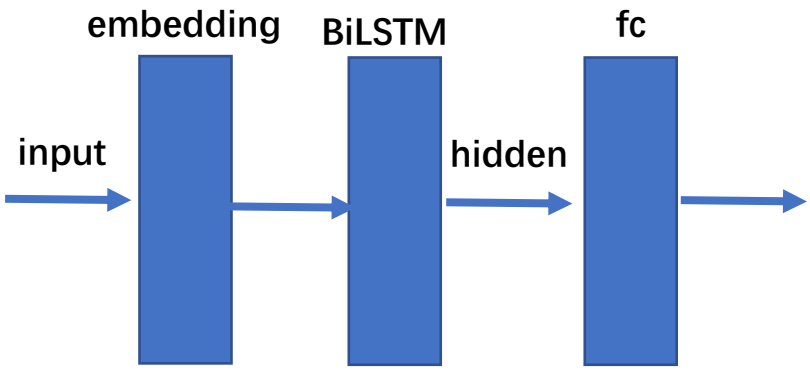
- **模型一 CNN：**input 先过一个 embedding 层，再过一个卷积层。卷积核大小为(3, 4, 5)，有 100 个。过了卷积层后用 ReLU 激活，然后 max pool。再把 3 个卷积核的结果连起来，然后 dropout，最后过全连接层，然后输出.



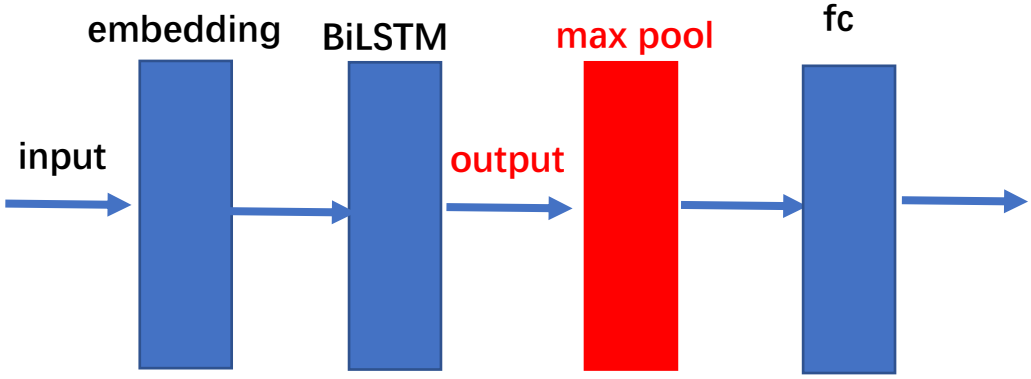
- **模型二 RNN：**1 个 embedding 层+1 层双向 RNN+1 个全连接层



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

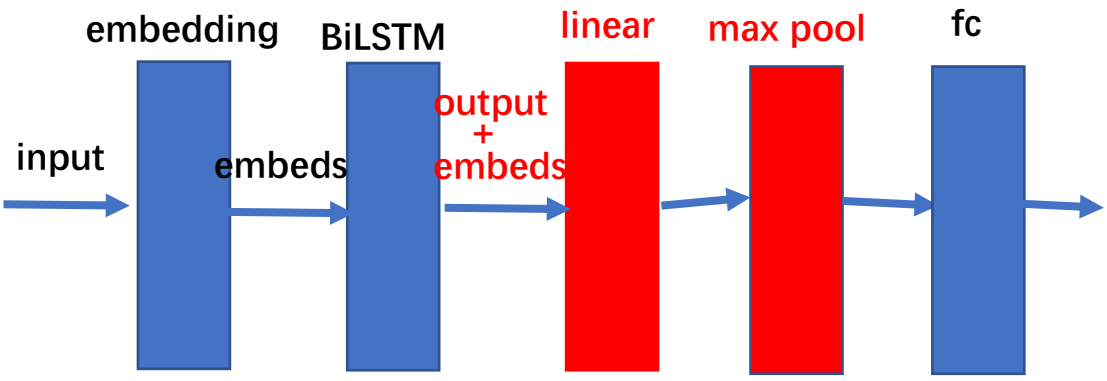


- **模型四 LSTM_maxpool：**1 个 embedding 层+2 层双向 LSTM+max pool+dropout+1 个全连接层（这个模型介于 LSTM 和 RCNN 之间，只是比 LSTM 多了一层 max pool）



- **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



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。

2.7 Trainer 和 Tester 实现

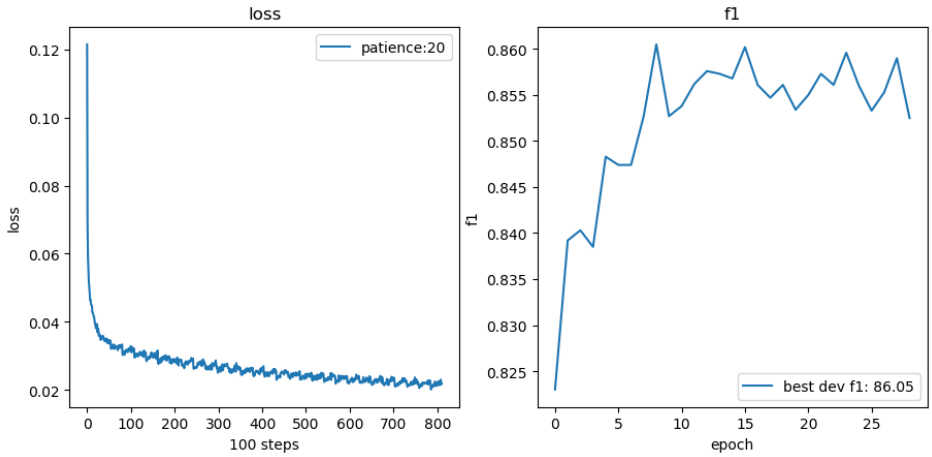
（代码请看 train.py）

- 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

2.8 模型结果

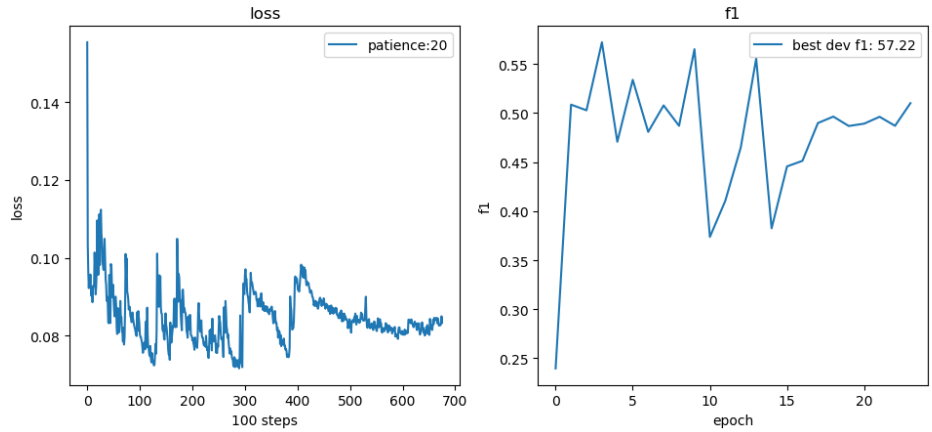
- 模型一 CNN：

运行时间	best epoch	best f1
6590s	9	86.05



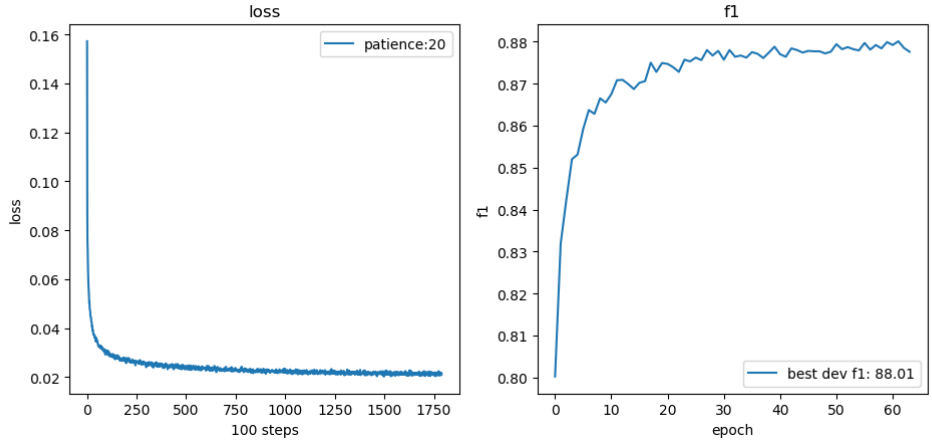
- 模型二 RNN：

运行时间	best epoch	best f1
9273	4	57.22



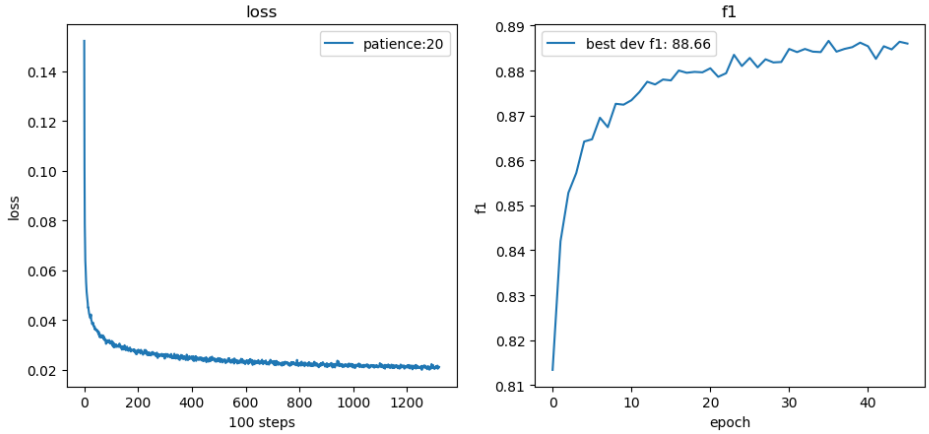
- 模型三 LSTM：

运行时间	best epoch	best f1
23680s	47	88.01



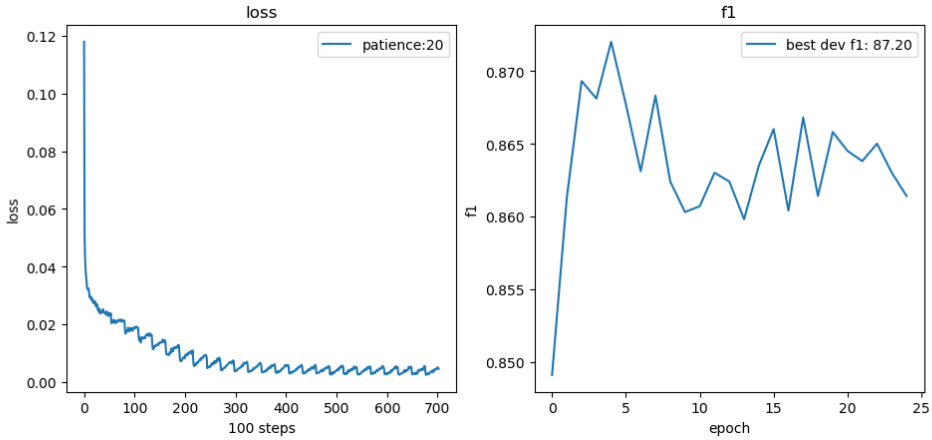
- 模型四 LSTM_maxpool

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



2.9 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

3. 预训分类模型

（见 p_classification/bert/文件夹）

3.1 模型实现

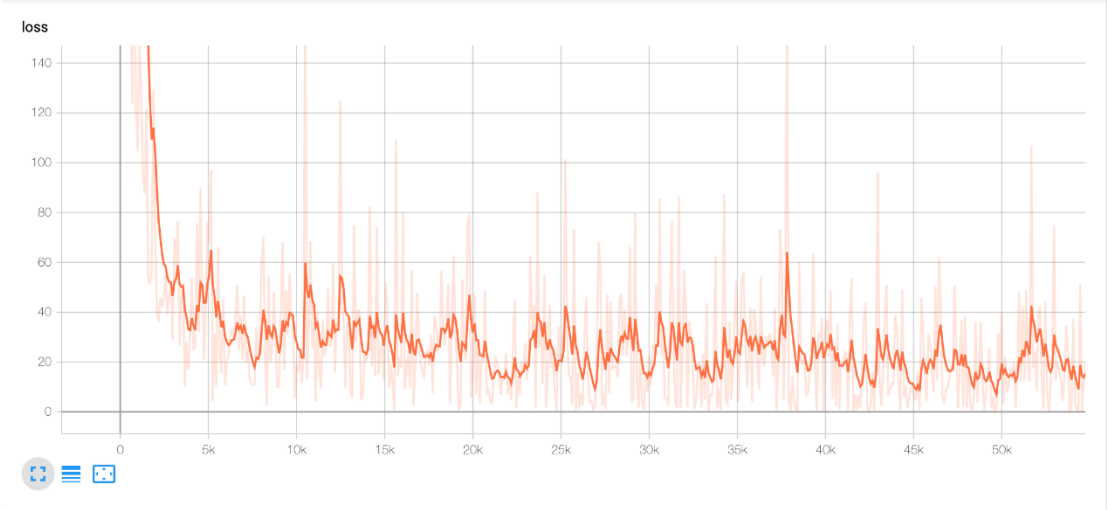
- 使用了 **bert** 来进行多分类，在 bert 的中文预训练模型参数下 finetune。
- 基于 google-reserce 的 bert repository 来实现，直接在上面改代码
- 主要基于 run_classifier.py 修改，原文件是用于单分类的任务，通过把 softmax loss 改成

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

4. 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

5. 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

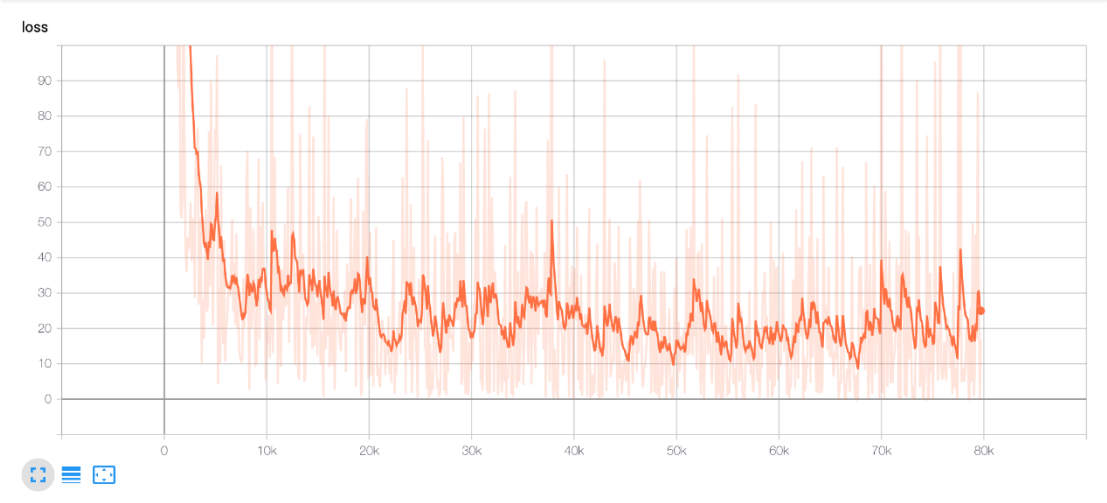
- bert 模型输入 text
- 从 bert 模型 get pooled output，获得 predicate，从 bert 模型 get sequence output，获得序列标注**（标注采用的是百度 baseline 的 BIO， 另外 label 为 SUB 和 OBJ）**
- 计算 predicate 的 loss 和序列标注的 loss，相加作为最终的 loss
- 最终模型输出的是 predicate 和这个 predicate 对应的序列标注
- 而分类模型已经输出了 text 对应的 predicate，接着就从中选出真正的 predicate 和对应的序列标注

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）

```
F1_score: f1=tensor(0.8134, dtype=torch.float64), recall=tensor(0.7714, dtype=torch.float64), precision=tensor(0.8603, dtype=torch.float64)
```

6. 优化标注模型

（见 so_labeling/bilstm-crf/文件夹）

6.1 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

6.2 模型结构

（见 so_labeling/bilstm-crf/文件夹中的 model.py 文件）

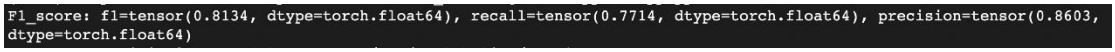
- 结构为 Embedding + LayerNorm + BiLSTM(2 layers) + FC + LayerNorm + DropOut + FC + CRF
- 其中 embedding 由 4 部分组成，分别是按字分割的 text 的 embedding(词典大小为 8163)、按词分割的 text 的 embedding（词典大小为 379713）、text 的词性列表的 embedding（词典大小为 26），以及 predicate（词典大小为 53，还有一个是'无'）的 embedding
- 以上的词典大小均加上了'<unk>'和'<pad>'

6.3 模型结果

- 由于时间原因，这个模型还没来的及训练完。

7. 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



8. 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。