Assignment 4 Report

代码说明：

config.py	参数设置
data.py	调用 fetch_20newsgroups 生成 vocabulary、train_dataset、validate_dataset、test_dataset
model.py	用 pytorch 定义的 RNN 和 CNN
train.py	训练模型并保存
test.py	加载指定的模型并测试准确率

一、Part 1

1. Dataset 构建

1.1 数据集获取

调用 sklearn.datasets 的 fetch_20newsgroups 函数即可。**（本次实验使用了所有数据，共 20 类）**

1.2 数据预处理

首先对于每篇文章，利用正则表达式，进行标点忽略、换行符空格替换、大小写转换、切割为单词等一系列操作，核心代码如下所示：

```
dataset_.apply(lambda x: re.sub('[%s]' % re.escape(string.punctuation), '', x['raw_sentence']), new_field_name = 'sentence') #忽略标点
dataset_.apply(lambda x: re.sub('[%s]' % re.escape(string.whitespace), ' ', x['sentence']), new_field_name = 'sentence') #将空格、换行符等空白替换为空格
dataset_.apply(lambda x: x['sentence'].lower(), new_field_name = 'sentence') #转换为小写
dataset_.apply_field(lambda x: x.split(), field_name='sentence', new_field_name='input')
```

接着需要做的是将获取到的原始训练集切割为训练集(train_data)和验证集(dev_data)（这里使用 8：2），并由训练集构建 fastNLP 的 vocabulary。之后利用 vocabulary 将数据集中的单词转换为索引。

最后设置数据集的 input 和 target 以便 fastNLP 识别。

具体细节见 data.py。

2. 模型搭建

1.1 CNN

仿照 fastNLP 的 cnn_text_classifcation 搭建，使用了三个不同的卷积核，但将卷积核的数量固定为 64。由于 embedding 后的数据的维度为(batch,seq_len,embedding_size)，不符合卷积层的要求，因此需使用 unsqueeze 增加维度。在经过卷积层、激活函数、池化层后，维度变为（batch，kernel_num）。将三个输出连接到一起后，最后使用 dropout 加上全连接层得到最终结果。

具体细节见 model.py 中的 CNN 类，其中含有维度变化的详细注释。

1.2 RNN

RNN 模型我主要使用的是**双向 LSTM**，在经过 LSTM 后，需要思考的是应该将什么数据送给全连接层。最初，我设想直接使用 LSTM 最后的 hidden，但实验发现无法收敛。最后我一共使用了四种方式：

- LSTM with attention
- LSTM 输出在 seq_len 维度上取平均（batch,seq_len,2*hidden）

->(batch,2*hidden)

- LSTM 输出在 seq_len 维度上取最小值
- LSTM 输出在 seq_len 维度上取最大值

具体细节见 model.py 中的 RNN 类，其中含有维度变化的详细注释。

3. 使用 fastNLP 进行训练

事实上，我已经在 assigment3 中使用过 fastNLP，这里再重复一遍使用 fastnlp 中的 Trainer 需要注意的点：

- 数据需要定义“input”、“target”等 field，以便 fastnlp 识别使用的参数
- 模型的输出需要是一个字典，以便 fastnlp 进行参数识别
- 如果需要在训练时进行其它操作，可定义一个继承 callback 的类，并重载相关函数。这里使用了 EarlyStopCallback。
- 如果需要对模型进行评估，需提供验证集（dev_data）
- 为了计算相关指标，需提供一个继承 MetricBase 的类。例如本实验需使用 AccuracyMetric。
- 提供 metric_key（来自 metric 类的输出）以决定最好的 model

在熟悉 fastnlp 后，只需设置好相关参数，即可进行训练，同时将训练好的模型保存到设置的路径。细节见 train.py。

4. 训练及测试结果

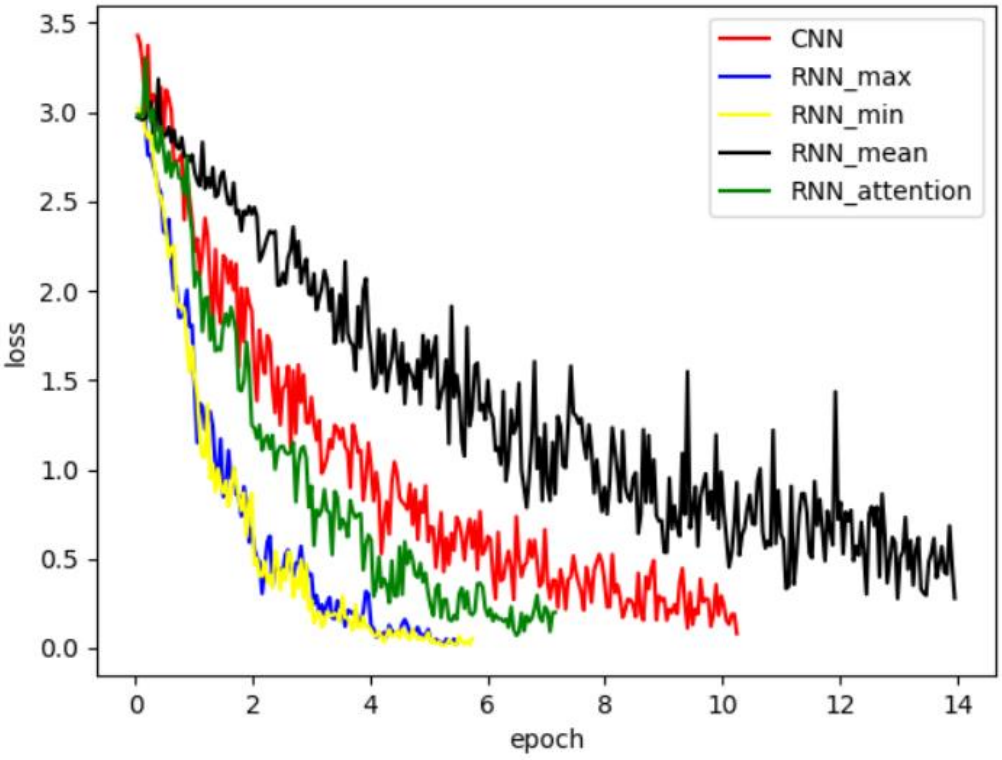
- **参数设置**

learning_rate	class_num	batch_size	input_size	hidden_size	optimizer	patience
1e-3	20	16	128	256	Adam	10

- **准确率**

model	CNN	RNN_attention	RNN_mean	RNN_min	RNN_max
Accuracy on dev_data	0.86	0.78	0.74	0.88	0.88
Accuracy on test_data	0.74	0.65	0.64	0.77	0.79

- **loss 曲线**



从以上结果可以看到，本次试验中 RNN_max 和 RNN_min 表现最好，收敛速度快，并且准确率最高。CNN 表现也还不错，虽然收敛速度不是特别快，但准确率还比较高。而 RNN_atention 和 RNN_mean 表现效果较差，但相比较与 RNN_mean，RNN_attention 准确率有所提升，收敛速度也非常快。

二、part 2

关于 fastNLP 的一些想法和建议：

1. **在设置 batch_size 时，希望可以对 train_data 和 dev_data 分别设置。这是因为我**有一次在 assigment3 中调试 PerplexityMetric 时，希望 dev_data 的 batch_size 可以是 1。
2. **在用 Trainer 中，希望可以提供一个能够恢复训练状态的功能。**这是因为在训练时我偶尔会因为特殊原因意外中断训练，这时我只能从头开始训练。这要求在保存模型时不能只简单保存模型参数，还可能需要保存其它一些必要的参数（比如当前的 wait 值）。（好像有点困难）
3. **在保存模型时，可以让用户自定义保存模型的名字，并选择保存模型的个数（比如 top 3）。**这是因为我在训练 RNN 时，由于有多种 RNN，而保存的模型的名字只有时间信息，导致我有时很难区分它们，只能手动改变名字。而保存多个模型（top k）可以让用户测试模型是否过拟合，以及其它作用。
4. **希望 Trainer 可以将训练得到的 loss 返回，以便用户能够自行决定如何操纵 loss（比如绘制 loss 曲线）。**这是因为我尝试使用 fitlog，但遇到了一些错误，最终只能将打印的 loss 保存，然后读取文件进行字符串处理来绘制 loss 曲线。