**Assignment 4**

**Part I: Text Classifiers**

1. **代码组织**

|  |  |
| --- | --- |
| **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 |
| **w2v.py** | 生成word2vec的embedding的预训练weight，并用pickle导出 |
| **model.py** | 用pytorch定义CNN、RNN、LSTM、RCNN模型 |
| **utils.py** | 定义用于计时Callback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、accuracy，并画出曲线（由于在训练过程中用tee命令把控制台输出保存到log文件中） |

1. **数据处理**

**（代码请看dataset.py）**

* 1. 使用了20news-bydate\_py3.pkz数据，用fetch\_20newsgroups取出全部20类的数据，其中有train和test数据
  2. 把原始train、test数据导入到fastNLP的DataSet中，分别用3个apply来对数据进行以下操作
* 去掉string,punctuation
* 把string.whitespace变成space
* 把数据全部小写化，并以space切词
  1. 把处理好的train数据按4:1划分为train\_data和dev\_dev，而处理好的test数据就作为test\_data
  2. **从train\_data中获得vocabulary，大小为55253**
  3. 按照train\_data获得的vocabulary，分别用apply把trian\_data、dev\_data和test\_data中的word变为index
  4. 用pickle导出最终的trian\_data、dev\_data、test\_data和vocabulary，后缀名为.pkl
  5. **数据集参数**

|  |  |  |  |
| --- | --- | --- | --- |
| 20news-bydate\_py3.pkz | 最终数据 | Size | **Categories** |
| train\_set | train\_data | 9052 (80%)  (vocabulary size: 55253) | **20** |
| dev\_data | 2262 (20%) |
| test\_set | test\_data | 7532 |

1. **word2vec的 embedding预训练weight生成**

**（代码请看w2v.py）**

* 用pickle导入上文生成的trian\_data、dev\_data、test\_data和vocabulary
* 用vocabulary把trian\_data、dev\_data、test\_data中的idx变回word
* 把train\_data中的input数据取出，存到一个二维数组中
* 用gensim定义word2vec模型
* 用上述train\_data的二维数据构建word2vec模型的vocabulary
* 把上述train\_data的二维数据放到word2vec模型中训练
* 从模型中取出每个index对应的embedding，组成weight矩阵。其中index为0的单词是’<pad>’，在word2vec模型中不存在（虽然fastnlp的vocabulary中有‘<pad>’，但是数据中没有’<pad>’），因此把index为0的embedding定为全0
* pickle保存weight矩阵

|  |  |  |  |
| --- | --- | --- | --- |
| **word2vec模型参数** | **window** | **min\_count** | **size** |
| **值** | **1和64** | **1** | **embed\_dim=128** |

1. **模型参数**

**（代码请看config.py）**

* **软编码**：本次实验所涉及的参数全部定义在这个文件里，包括模型参数、文件路径、文件名等。
* **本次实验一共实现了CNN, CNN\_w2v, RNN, LSTM, LSTM\_maxpool, RCNN共6个不一样的模型，**以下是这些模型的参数：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **model** | **CNN** | **CNN\_w2v** | **RNN** | **LSTM** | **LSTM\_maxpool** | **RCNN** |
| **embed\_dim** | 128 | | | | | |
| **kernel\_sizes** | (3, 4, 5) | |  |  |  |  |
| **kernel\_num** | 100 | |  |  |  |  |
| **in\_channels** | 1 | |  |  |  |  |
| **dropout** | 0.5 | | | | | |
| **word2vec** | False | True | False | | | |
| **num\_layers** |  |  | 1 | 2 | 2 | 1和2 |
| **bidirectional** |  |  | true | | | |
| **hidden\_dim** |  |  | 256 | | | |
| **optimizer** | Adam (lr=1e-3, weight\_decay=0) | | | | | |
| **patience** | 100和10 | 20 | 20 | 100和20 | 20 | 20 |
| **max\_epoch** | 128 | | | | | |
| **batch\_size** | 64 | | 8 | | | |
| **print\_every** | 10 | | | | | |
| **validate\_every** | 100 | | | | | |

1. **模型结构（用pytorch实现）**

**（代码请看model.py）**

**一共实现了CNN, CNN\_w2v, RNN, LSTM, LSTM\_maxpool, RCNN共6个不一样的模型。**

* **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CNN\_w2v：和CNN的唯一区别是，在embedding层导入word2vec的embedding预训练weight（初始化），然后再进行训练**
* **模型三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**

* **模型四LSTM：**1个embedding层+2层双向LSTM+dropout+1个全连接层

**input**

**embedding**

**BiLSTM**

**hidden**

**fc**

* **模型五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**

* **模型六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**

1. **计时Callback实现**

**（代码请看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。

1. **Trainer和Tester实现**

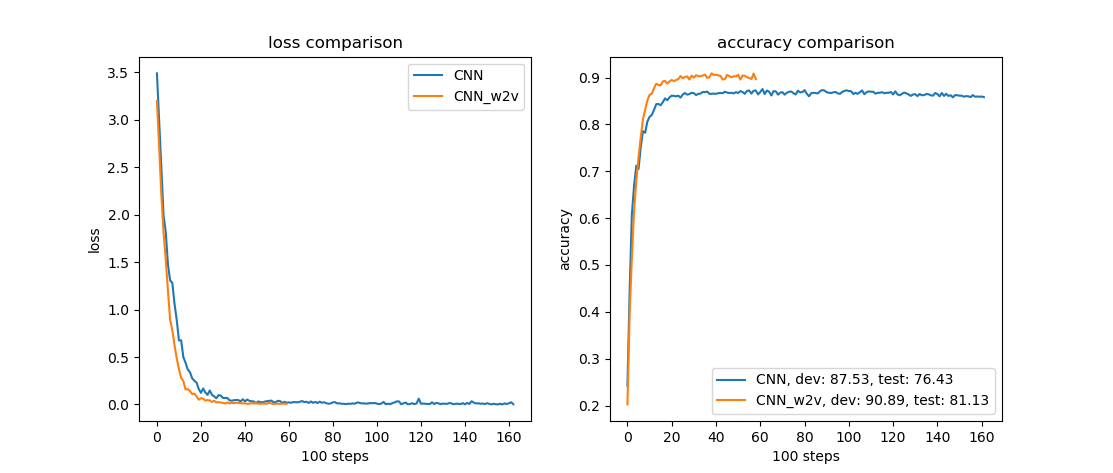
**（代码请看train.py）**

* 用pickle导入trian\_data、dev\_data、test\_data和vocabulary，以及word2vec模型的embedding预训weights
* 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Metric为AccuracyMetric
* 然后定义Trainer，进行训练
* 最后定义Tester，并在test\_data上进行测试

1. **模型结果和对比**

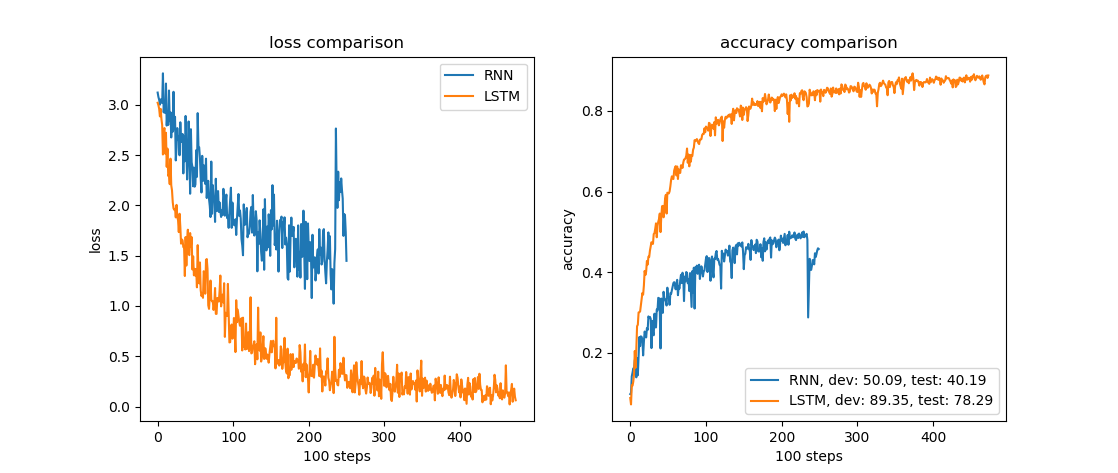
**（以下出现的图有统一的格式：左图为loss对比，右图为dev上的accuracy对比，右图的label显示了dev上的最高accuracy、test上的最终accuracy）**

* **对比一：CNN和CNN\_w2v**



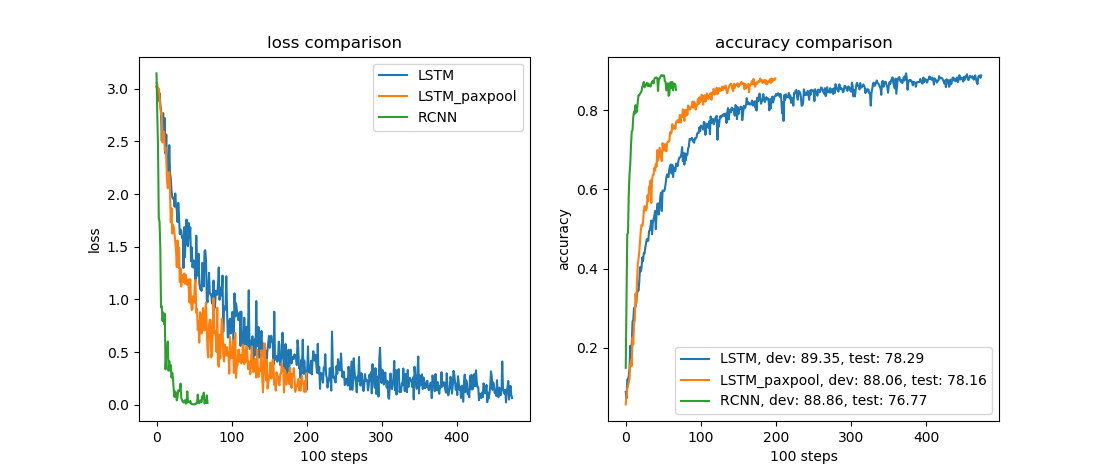
|  |  |  |
| --- | --- | --- |
| 模型 | CNN | CNN\_w2v |
| end\_epoch | 115 | 43 |
| sum time | 3851s | 3049s |
| best dev acc | 87.53 | 90.89 |
| test acc | 76.43 | 81.13 |
| 对比 | * 用了word2vec的embedding的weight初始化后，模型收敛速度更快，而且最终结果提高了~5% | |

* **对比二：RNN和LSTM**



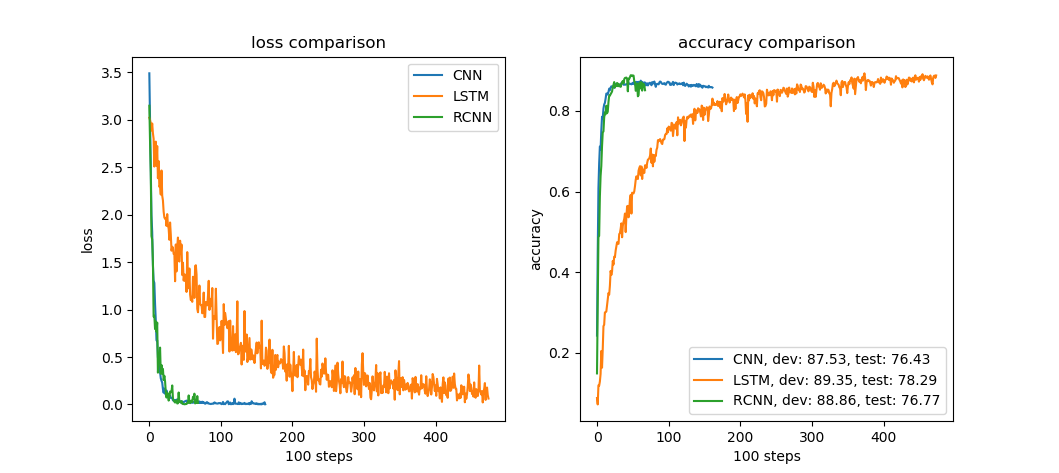
|  |  |  |
| --- | --- | --- |
| 模型 | RNN | LSTM |
| num\_layers | 1 | 2 |
| end\_epoch | 23 | 42 |
| sum time | 6790s | 31128s |
| best dev acc | 50.09 | 89.35 |
| test acc | 40.19 | 78.29 |
| 对比 | * 层数增加了之后，运行时间显著增加 * 应用了LSTM的结构后，accuracy显著提高，因此如今LSTM一般会被作为vanilla RNN | |

* **对比三： LSTM、LSTM\_maxpool和RCNN**



|  |  |  |  |
| --- | --- | --- | --- |
| 模型 | LSTM | LSTM\_maxpool | RCNN |
| end\_epoch | 42 | 18 | 7 |
| sum time | 31128s | 26154s | 2887s |
| best dev acc | 89.35 | 88.06 | 88.86 |
| test acc | 78.29 | 78.16 | 76.77 |
| 对比 | * 应用了max pool结构后，收敛更快了，最终accuracy也差不多 * 而应用了完整的RCNN结构后，收敛还能更快，而且performance也能保持在一个水平 | | |

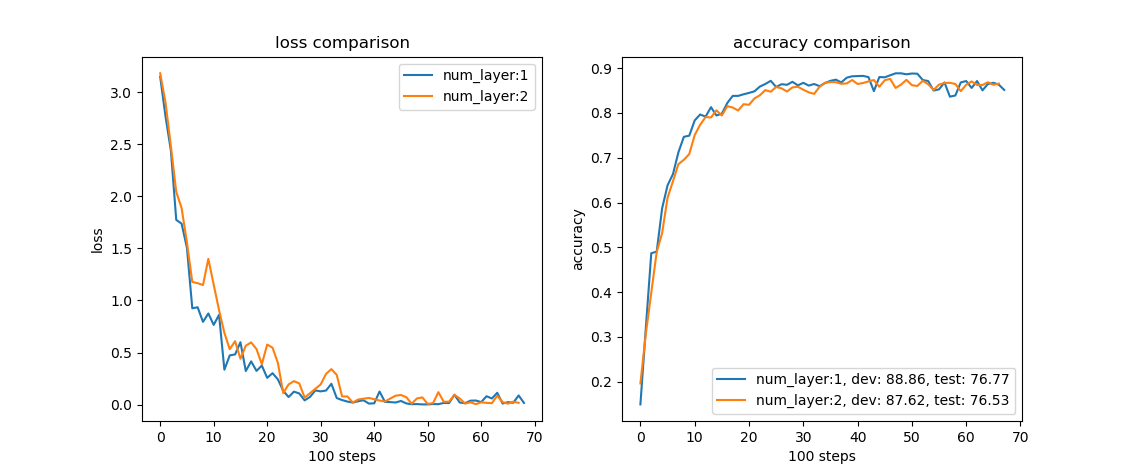
* **对比四： CNN、LSTM和RCNN**



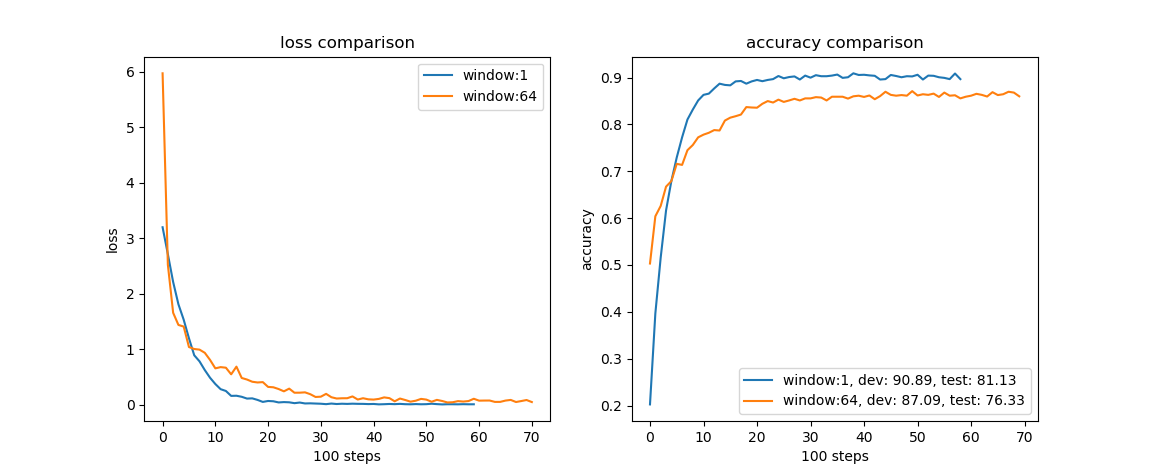
|  |  |  |  |
| --- | --- | --- | --- |
| 模型 | CNN | LSTM | RCNN |
| end\_epoch | 115 | 42 | 7 |
| sum time | 3851s | 31128s | 2887s |
| best dev acc | 87.53 | 89.35 | 88.86 |
| test acc | 76.43 | 78.29 | 76.77 |
| 对比 | * CNN结构的运行时间更快，而具有时序结构的LSTM和RCNN（基于LSTM）运行时间长了很多 * 三者最终的accuracy都差不多 * RCNN既具有了LSTM的特点，也具有了CNN的收敛速度，是两者的综合。 | | |

1. **部分模型参数的简单对比**

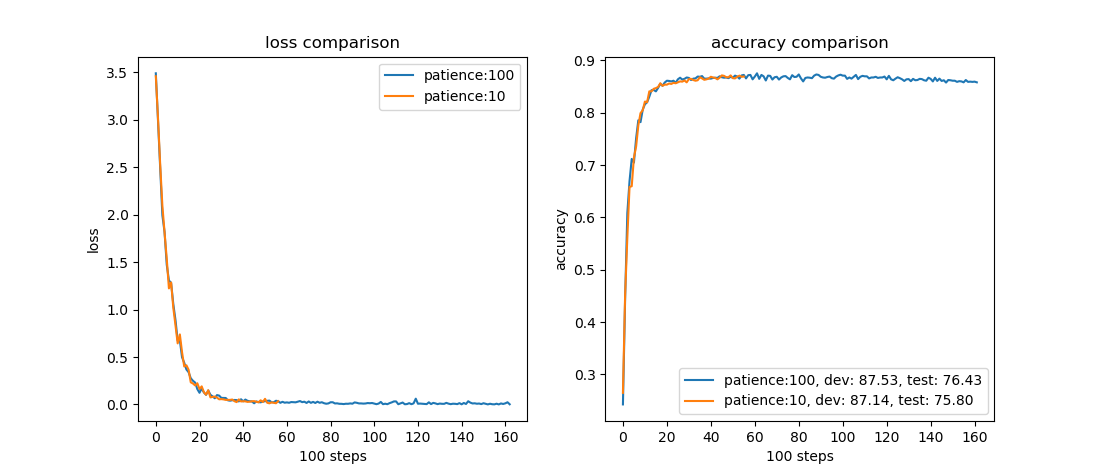
* **对比一： 基于RCNN的num\_layers对比（增加BiLSTM层数并没有显著影响）**



* **对比二： 基于CNN\_w2v的window对比（小的window效果更好）**

****

* **对比三： 基于CNN的patience对比（patience不需要设置的很大，不仅不能他提高accuracy，反而使用了更长的训练时间）**



**Part II: Suggestion**

1. 使用感觉

* 这样的封装感觉还是很好的，写代码可以更快捷、简洁

1. 建议

* Vocabulary在初始化的时候可以添加end\_of\_sentence=’<EOS>’和start\_of\_sentence=’<START>’
* 可以导入dataset直接生成Vocabulary
* Trainer只能传进1个batch\_size，但是train和dev数据其实可以用不同的batch\_size
* 似乎不支持继续训练模型，希望可以支持继续训练模型。
* print\_every输入的是step间隔，可以增加一个输入epoch的间隔
* 文档里有部分的函数或类是没有使用样例的，如果能像pytorch的文档一样有使用样例和输出样例的话，能帮助理解fastNLP的使用

­­­