Assignment 4 of PRML

Part 1

Ⅰ. 数据预处理

- **数据集**：采用sklearn库中的fetch_20newsgroups数据集，共计20种标签，数据集大小11314、测试集大小7532，相比Assignment2加大了数据量，并提高了分类难度；
- **预处理**：首先将句子中的特殊符号删除，再把大写字母全部改为小写，最后调用split()分为数组。使用fastNLP的Dataset操作如下：

```
# train_set: type fastNLP.Dataset
train_set.apply(lambda x: x['sentence'].translate(str.maketrans("", "",
string.punctuation)).lower(), new_field_name='sentence')

train_set.apply(lambda x: x['sentence'].split(), new_field_name='words')
train_set.apply(lambda x: len(x['words']), new_field_name='seq_len')
```

- **建字典**：设置最低词频为10，可以有效过滤许多含有数字的无意义字符串。使用fastNLP操作如下：

```
vocab = fastNLP.Vocabulary(min_freq=10)
train_set.apply(lambda x: [vocab.add(word) for word in x['words']])
test_set.apply(lambda x: [vocab.add(word) for word in x['words']])
vocab.build_vocab()
```

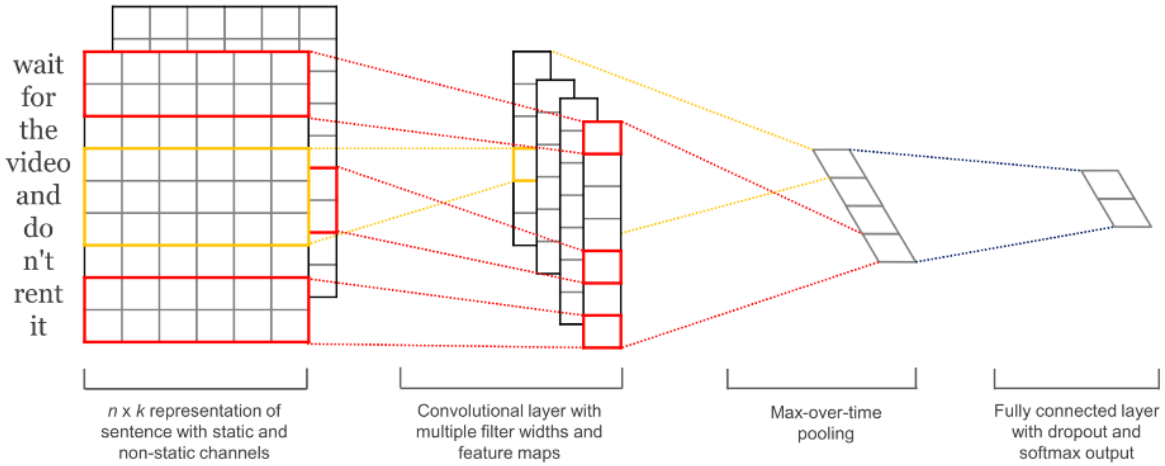
- 对于数据处理fastNLP有着良好的封装库，直接按官方文档调用即可。

Ⅱ. 建立模型

- CNN

- **模型理解**

  第一个模型是使用卷积神经网络（CNN）进行文本分类，类似于图像处理，TextCNN的目标对象是文本数据，矩阵每一行对应一个文本单词，借用Yoon Kim的论文配图展示如下：



TextCNN的框架同样包括四层：**输入、卷积、池化、全连接。**通过word embedding进入输入层得到文章矩阵，接着在卷积层中进行特征提取，并通过池化层进行结果压缩，不断重复卷积和池化的过程，得到特征向量，通过全连接层进行文本分类。

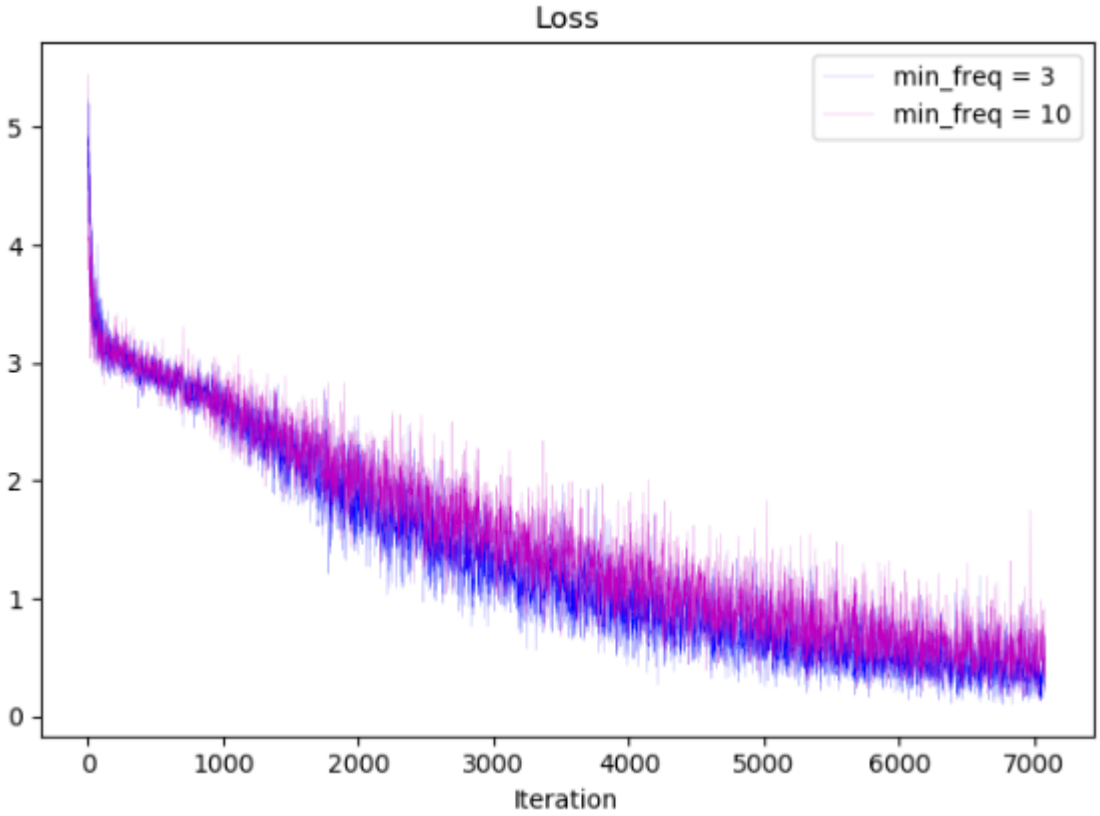
1. **输入层**：采用torch.nn.Embedding，将原文本降维到embedding_dim；
2. **卷积-池化**：采用fastNLP封装好的卷积+池化模块：fastNLP.encoder.ConvMaxpool。该模块采用最大池化方法，使各个卷积核的特征向量对应到超参设定的out_channel大小。超参out_channel定义了不同大小卷积核的数量，而与之相应的另一个超参kernel_size则定义了不同大小的卷积核。在实验中合理选择卷积核的大小和数量，方可获得良好的训练结果；
3. **全连接层**：采用torch.nn.Linear，用于整合来自不同卷积核的特征向量，得到最终的分类结果向量。注意设置dropout防止过拟合化。

- **进行训练**：

  编写train函数。根据fastNLP的官方文档，编写了自己的train函数，并自行添加了torch.cuda的使用、模型的保存以及日志保存的功能。

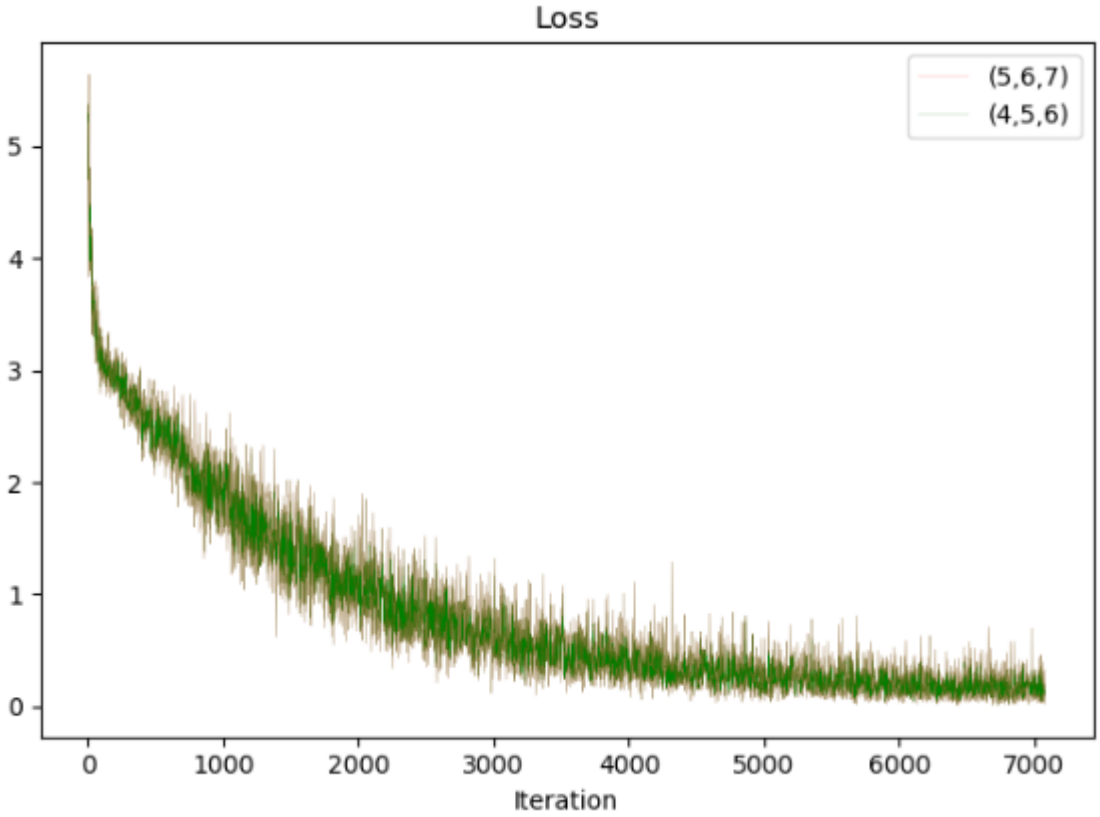
  进行模型训练的一大主要工作还是调参数。对不同参数的调节有如下结果：

  - 建立词典的最小词频 min_freq



二者的Loss下降曲线接近一致，甚至最小词频为3时Loss收敛值更低一些，但是经过测试集的结果，$min\_freq = 3$ 时的准确率为0.557832，而$min_freq = 10$时的准确率为0.625198。因此将最小词频取值放大一些。

- 卷积核大小及数量：$out\_channel.\ kernel\_size$



本次实验固定$out_channel$和$kernel_size$相等，分别取（5，6，7）和（4，5，6）得到上图所示的结果。也可以看到二者收敛速度和最终范围是几乎一样的，但通过准确率的考核，取（5，6，7）的准确率0.629310，取（4，5，6）的准确率0.742801, 因此取（4，5，6）。

- **最终参数设定**:
  - $embedding\_dim = 150$
  - $padding = 2$
  - $out\_channel = kernel\_size = (4, 5, 6)$
  - $batch\_size = 32, epoch = 20$
  - $Loss = CrossEntropyLoss$
- 最终准确率为：0.742801

- RNN

- **模型理解**

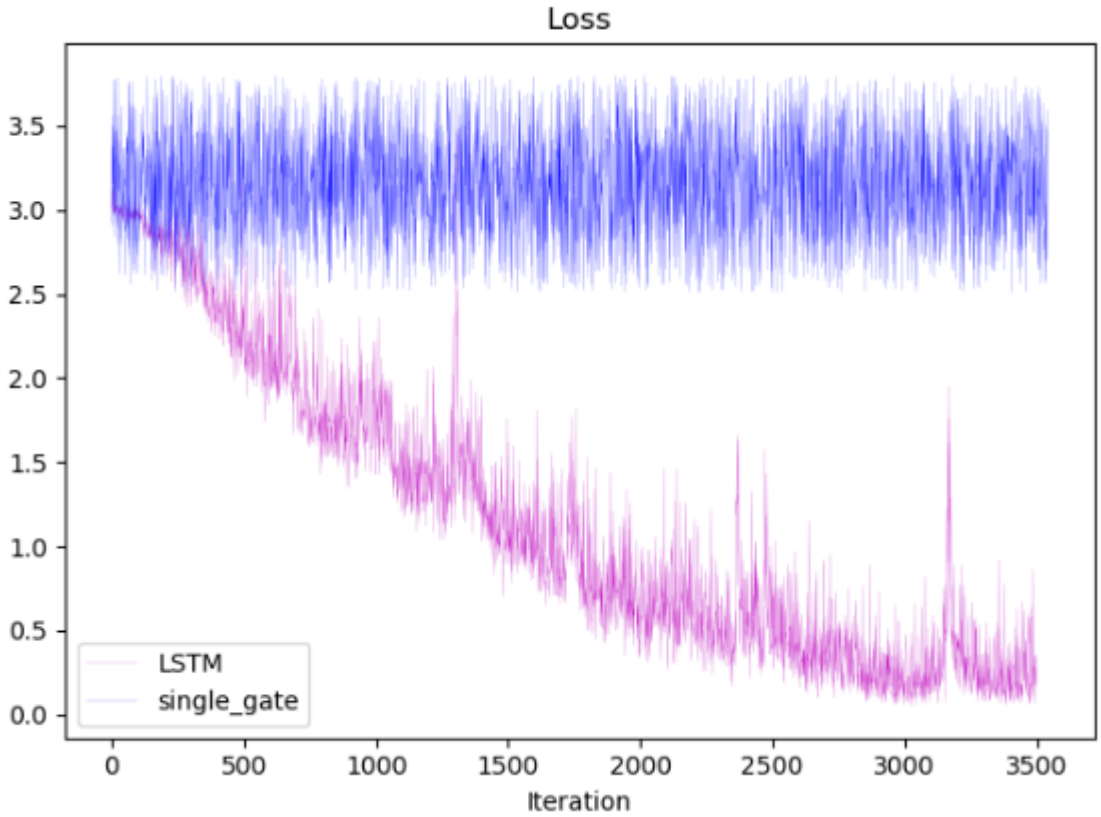
  在完成Assignment3的基础上，理解RNN就容易多了。依然是首先通过word embedding进行降维，之后将每段话的每个词按时序进入RNN层，取全文段输入后得到的最后一个预测结果作为分类依据。

  1. 输入：采用torch.nn.Embedding，将原文本降维到embedding_dim；
  2. 神经元层：一开始尝试使用含单个门的标准RNN，每个神经元仅采用一次激活函数tanh，但是由于单门路RNN的记忆局限性，训练过程的Loss下降缓慢（下文有附图比较）。于是采用RNN中含有细胞状态的长时记忆LSTM，利用fastNLP中封装的encoder.LSTM，经实验测试达到了不错的效果；
  3. 输出：采用torch.nn.Linear，得到最终分类预测。

- **进行训练**：

  编写train函数。依然结合fastNLP的官方文档编写自己的train函数，添加了torch.cuda、模型保存以及日志保存的功能。

  比较一下普通的单门路RNN和长时记忆LSTM的差异：



  可以看出，使用单门路RNN的loss是很难收敛的，这是因为短期记忆对于文段的分类并不有效，不能有效概括其整体文段特征。因此采用LSTM较为合理。

- **最终参数设定**：
  - $embedding\_dim = 130$
  - $hidden\_dim = 130$
  - $batch\_size = 32,\ epoch = 20$
  - $Loss = CrossEntropyLoss$
- 最终准确率：0.796732

Part 2

About FastNLP

- fastNLP的整体使用体验不错，对于数据整合、词典构建等方面封装好的Dataset、Vocabulary等模块使用相当方便；此外，利用封装在torch外的接口编写自己的模型和训练函数，也大大降低了代码编写的复杂性，其框架的复用性也较为良好，对于自然语言的工作者而言比较便捷；
- 存在的问题：
    1. 内置的Trainer需要结合fitlog来记录训练过程和权重矩阵，且win10环境下图线显示有问题，个人认为可以直接将fitlog内置在fastNLP里并降低调用fitlog的粒度，不过由于可以灵活编写自己的训练框架，这个问题不算太大；
    2. debug的过程中，某些提示信息不够详细。例如我在编写自己的单门路RNN模型时，关于BucketSampler中关于序列长度等的注释信息还不足，存在些许阻碍；
    3. 建议fastNLP官方文档多提供一些编写模型和函数的样例供读者参考。