lab4实验报告

数据预处理

本次实验中采用了全部的 20 Newsgroups数据集，共20分类，根据assignment2的预处理方式，首先去除所有的标点符号，根据空格分词，预处理的过程主要利用了fastNLP中Dataset的apply方法，再利用Vocabulary建立词典，去除所有出现次数小于10次的单词。调用Vocabulary的index_dataset方法得到train_set, test_set，最后以9:1的比例将train_set划分为train_data和dev_data。

算法实现

Trainer

fastNLP中的trainer自带了对模型的训练功能，使用时分别传入模型、损失函数、优化器、训练周期数、训练数据、验证数据、测量指标等参数。这次实验我分别采用了CNN和RNN模型，损失函数为交叉熵，优化器为Adam，测量指标为分类准确率。另外为了提高训练的效率，我还调用了EarlyStopCallback函数，并设epoch为10。

CNN

实现的算法依据论文Convolutional Neural Networks for Sentence Classification，算法思路为利用三个不同大小的卷积核（3、4、5）分别对输入的文本序列做一维卷积提取特征，卷积核大小不同蕴涵了多尺度的思想。提取特征后再接一个池化层，这里的池化层有多种选择，可以为平均、最小、最大等，根据经验主义和实验结果的比较，最终选取最大池化层。最后再接一个dropout层和全连接层，dropout层在每次训练中以一定概率将神经元置0，有利于防止过拟合，提高模型的泛化性能。

实验的超参设置如下：

learning rate	weight decay	batch size	vocab size	embedding dim
1e-3	1e-4	16	17249	128

在CNN模型中，卷积核数量的不同对性能的影响较大，更多的卷积核有助于提取不同方面的特征，提高分类准确率，对比实验结果如下：

kernel num	epoch	validation accuracy	test accuracy
(5, 5, 5)	50	81.7%	67.1%
(10, 10, 10)	48	88.5%	75.2%
(50, 50, 50)	50	90.7%	78.5%

随着卷积核数量的增长，性能有了明显的提升。

RNN

对于文本序列的分类问题，我们自然可以考虑用序列模型加以解决。本次实验中的RNN采用了双向lstm模型，用lstm输出的hidden state作为文本特征，接一个dropout层后再接一个全连接层，双向的RNN更有助于提取上下文信息。

实验的超参设置如下：

learning rate	weight decay	batch size	vocab size	embedding dim	hidden dim
1e-3	1e-4	8	17249	128	128

实验中探究了lstm层数对性能的影响：

num of layers	epoch	validation accuracy	test accuracy
1	48	82.5%	65.2%
2	50	84.5%	67.5%
3	38	86.7%	67.9%

随着层数的加深，性能有了部分提升，但总体性能还是弱于CNN，并且训练所花的时间也长于CNN。

关于fastNLP

使用体验及建议

这次实验主要调用了fastNLP的Tester, Trainer, Dataset, Instance,Vocabulary以及fastNLP.modules中的encoder。明显地感觉到从数据的预处理，到模型网络的搭建，再到训练过程的编写都方便了很多，经过fastNLP的封装后，很多之前要重复写的代码都可以通过调库解决。另外文档也非常良心，指南中给出的例子以及API的说明可以让人很快上手，使用体验非常好，不过还有几个小建议：

1、Trainer中建议加一个选项可以返回tensor或者numpy形式的loss值，这样可以为研究loss曲线提供另外一种直观的方式。

2、使用Dataset中的Instance报错时可以考虑不输出具体的instance的全部内容（或者只输出开头的几行），这样在终端使用时可以更直观地看到报错信息。

3、文档中的callback部分希望多加一些示例，另外希望提供一下trainer中多卡训练的例子。

对别的nlp框架的调研

除了上述的使用建议以外，我还调研了另外两个nlp的深度学习框架，分别是基于tensorflow的lingvo和同样基于pytorch的pytext，并总结了一些它们相对于fastnlp的优势，在后续的开发中可以考虑添加进来。

lingvo

1、首先lingvo主要面向sequence模型，于是针对句子的输入处理部分专门开发了input generator模块，对于batch中不同长度的句子可以自动填充，另外支持从多个文件中读取同一个数据集，并且允许使用者自己编写预处理函数，有更大的自由度。

2、lingvo提供synchronous模式，支持分布式训练，可以在分布式文件系统中读写模型存档。

3、在lingvo中可以register多任务模型，可以选择不同模块间是否共享参数，另外专门开发了base_model . DistillationTask支持知识蒸馏。

4、相比于training阶段，inference阶段有不同的计算特性，lingvo在inference阶段给用户更多的自由度，可以几行代码内完成一个基础的inference graph，也可以编写更为复杂的inference graph。

pytext

1、pytext支持集成训练，集成对于提升模型的性能有很多的帮助，尤其是在打比赛的时候，支持集成可以减少编写代码的工作量。

2、在PyTorch 1.0的C10d backend基础上加入了分布式训练。

3、模型和任务的搭建可以通过一些Extensible components更方便地完成。

另外pytext和lingvo的共同优势是有很多的预训练好的state-of-the-art的模型，在后续的开发中fastnlp可以考虑加入更多的论文的复现模型。