PRML - lab4 - text classification

Overview

本次实验基于pytorch与fastNLP， 实现了对20 newsgroups数据集的多种经典文本分类模型：RNN，CNN，RCNN，word2vec+CNN，也让我对fastNLP有了进一步的了解。

Requirement 1: text classification

在文本分类模型的构建过程中，我把主要精力放在不同模型的尝试与对fastNLP的探索上，没有过多在调参、数据清洗等其它能够提高预测准确度的方法上下功夫。

dataset

在数据的选取上，我采用了20newsgroups text dataset全数据集。其中，共有11314条训练集，7532条测试集；数据集中所有文本的平均长度为1902，最小长度为115，最大长度为160616，中位数为1175。了解数据集的基本信息有助于我们在设置文本长度取padding操作上取更为合理的数值。

preprocess

预处理过程基本按照fastNLP的详细教程修改，其主要步骤如下：

1. 数据读入：从sklearn获取数据后转存dataset
2. 数据集处理：进行小写化、分词等一系列必须的操作
3. 构建词表：按照数据集内容构建词表，注意最小词频需根据数据集规模修改，在这里较大数据集的情况下可采取较大数值
4. 数据集分割：按照8：2把训练集划分为实际训练集与测试集

值得一提的是，在整个框架中，fastNLP中的属性标定非常重要，我们需要利用set_input, set_target等函数在模型训练前标定数据集中的输入属性及目标属性，而在训练时将输出属性标定为pred。这样的设定是为了快速训练trainer封装的便捷性，但牺牲了一定的灵活程度；好在Const类的存在下，我们可以弥补一些自由。

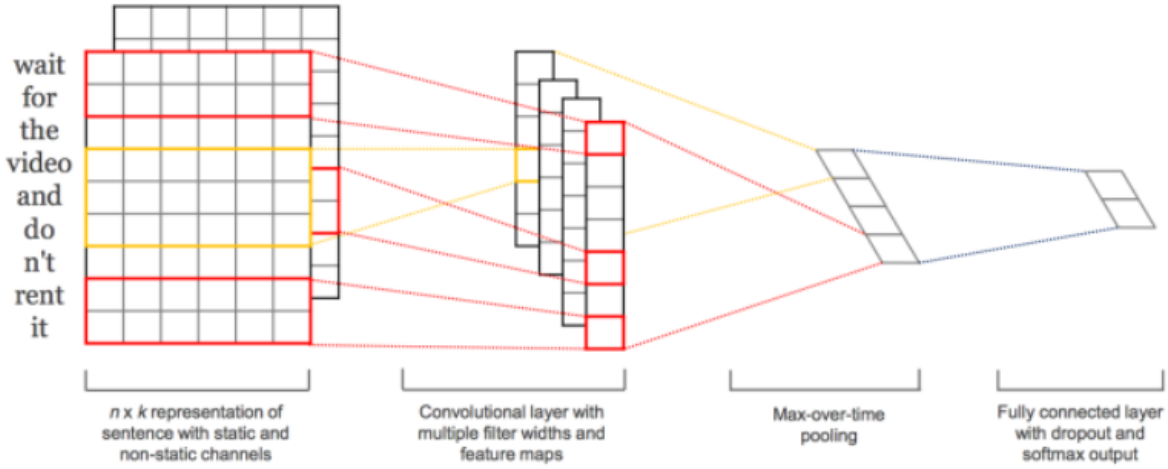
model

CNN

在fastNLP中，CNNText作为经典的模型已封装至fastNLP. models中，我们可以方便地调用它；当然，其原理也十分清晰，可以容易地实现。

- 原理

  下图展示了CNN文本分类的具体机制。对一个sentence, 我们可以构建seq_len * embedding_size的输入矩阵。与CV中CNN不同的是，这里采用一维卷积处理数据，卷积核的第一维固定为embedding_size，我们只需要输入卷积核第二维的大小即可。为了获取不同宽度的视野，我们往往采取不同大小的kernel size。这一步也导致了卷积结果长度的不一致，所幸我们第二层max pooling（即取卷积结果的最大值并拼接）正好滤去了此不一致性带来的影响。最后一步是常规的FC层，不提。CNN的核心在于捕捉到了局部相关性，得到类似n-gram的信息，从而达到文本分类的效果。



- 实现

  fastNLP源码中对CNNText的封装是较为灵活的：我们可以为不同kernel size分配不同数量，例如，定义2个kernel size为2、3个kernel size为3、4个kernel size为4的卷积层；另外，我们也可以自由定义padding大小以及dropout大小。同样，也有少部分内容是hard coding，如取max pool时，规定只取top1，而不是topk。我的实现也主要沿用了源码的思想。

- 结果

  在paper中提到，采用（2，3，4，5）作为kernel size能达到比较好的效果。在**embedding_dim=128, dropout = 0.1, kernel_nums = (2,3,4,5), kernel_size = (2,3,4,5), padding = 2**时，模型训练结果如下：

```
Evaluation at Epoch 1/10. Step:566/5660. AccuracyMetric: acc=0.418214

Evaluation at Epoch 2/10. Step:1132/5660. AccuracyMetric: acc=0.587975

Evaluation at Epoch 3/10. Step:1698/5660. AccuracyMetric: acc=0.669761

Evaluation at Epoch 4/10. Step:2264/5660. AccuracyMetric: acc=0.689213

Evaluation at Epoch 5/10. Step:2830/5660. AccuracyMetric: acc=0.707781

Evaluation at Epoch 6/10. Step:3396/5660. AccuracyMetric: acc=0.701149

Evaluation at Epoch 7/10. Step:3962/5660. AccuracyMetric: acc=0.700265

Evaluation at Epoch 8/10. Step:4528/5660. AccuracyMetric: acc=0.717065

Evaluation at Epoch 9/10. Step:5094/5660. AccuracyMetric: acc=0.699823

Evaluation at Epoch 10/10. Step:5660/5660. AccuracyMetric: acc=0.713086
```

  最终正确率约为0.7，在未调参的情况下尚可接受。

  在增加**kernel_nums = （10，10，10，10）**时，模型最高准确率达到0.79：

```
Evaluation at Epoch 7/10. Step:3962/5660. AccuracyMetric: acc=0.789125

Evaluation at Epoch 8/10. Step:4528/5660. AccuracyMetric: acc=0.773652

Evaluation at Epoch 9/10. Step:5094/5660. AccuracyMetric: acc=0.751989

Evaluation at Epoch 10/10. Step:5660/5660. AccuracyMetric: acc=0.77542
```

可以看出，kernel_num对模型的影响还是相当大的。

RNN

RNN文本分类问题在fastNLP详细说明中给出了lstm版本。其具体思想比CNN更容易理解，RNN本身就是为处理关联信息而生的，在lab3唐诗生成中已有相关介绍，在此不做赘述。同样地，由于lstm效率的问题，在此lab中我选取了GRU作为lstm的替代品，取得了不错的效果。

在**embedding_dim=128，dropout=0.1, hidden_dim=64, num_layers=2**的参数下，模型结果如下：

```
training epochs started 2019-05-28-22-29-47
Evaluation at Epoch 1/10. Step:283/2830. AccuracyMetric: acc=0.570292

Evaluation at Epoch 2/10. Step:566/2830. AccuracyMetric: acc=0.772767

Evaluation at Epoch 3/10. Step:849/2830. AccuracyMetric: acc=0.813882

Evaluation at Epoch 4/10. Step:1132/2830. AccuracyMetric: acc=0.811229

Evaluation at Epoch 5/10. Step:1415/2830. AccuracyMetric: acc=0.81786

Evaluation at Epoch 6/10. Step:1698/2830. AccuracyMetric: acc=0.827144

Evaluation at Epoch 7/10. Step:1981/2830. AccuracyMetric: acc=0.830239

Evaluation at Epoch 8/10. Step:2264/2830. AccuracyMetric: acc=0.821839

Evaluation at Epoch 9/10. Step:2547/2830. AccuracyMetric: acc=0.82405

Evaluation at Epoch 10/10. Step:2830/2830. AccuracyMetric: acc=0.821397


In Epoch:7/Step:1981, got best dev performance:AccuracyMetric: acc=0.830239
```

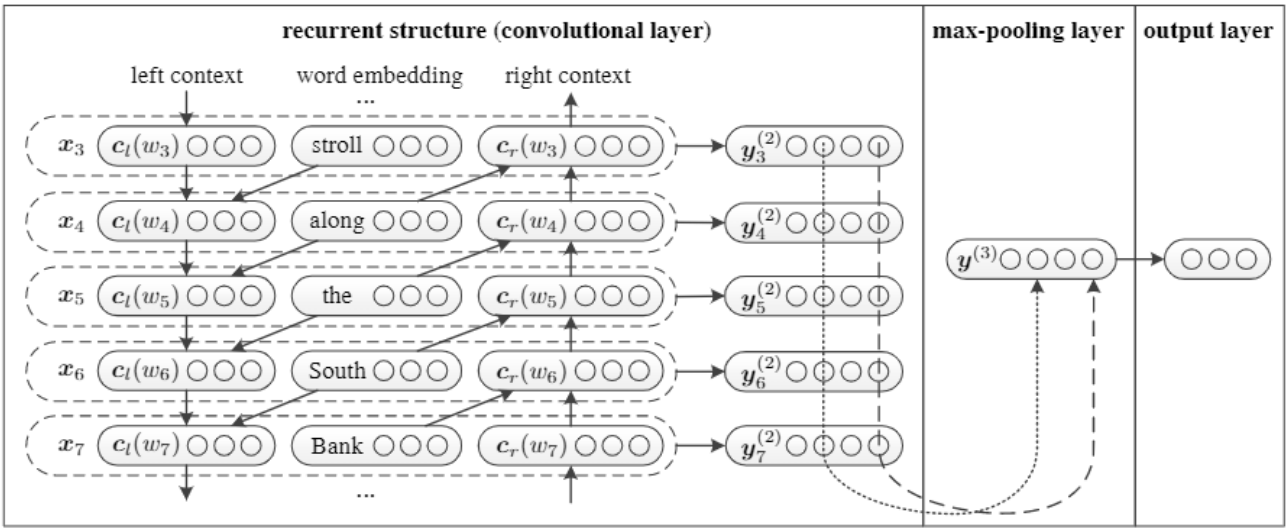
相较基本的CNN，模型表现相当不错。

在基于RNN文本分类任务上，我们学院提出的Recurrent Neural Network for Text Classification with Multi-Task Learning也很有意思，之后会尝试复现。

RCNN

Recurrent Convolutional Neural Networks for Text Classification(RCNN)是中科院15年的paper，嵌套了双向RNN与CNN（其实算是max pooling），也取得了不错的成果。这篇文章的提出也很符合"各取所长"的计算机哲学，和当下大热，融合了transformer+ elmo/双向gpt的bert有异曲同工之妙。

RCNN的具体模型如下所示：



简单来说，RCNN可以分为如下几个步骤：

1. word embedding，得到e(w)
2. 将词向量输入到双向RNN中，得到left content(Cl)和 right content(Cr)
3. 拼接结果并输入到tanh激活函数

   $x_i = [c_l(w_i); e(w_i); c_r(w_i)]$

   $y_i = tanh(Wx_i + b)$
4. 对3中的y进行用一维的max pooling池化
5. 输出层

总体看来，模型还是较为清晰的，在**dropout = 0.1, kernel_nums = (2,3,4,5), kernel_size = (2,3,4,5), padding = 2， hidden_size=64, num_layers=1**基本实现的结果如下图所示：

```
Evaluation at Epoch 10/20. Step:5660/11320. AccuracyMetric: acc=0.720601

Evaluation at Epoch 11/20. Step:6226/11320. AccuracyMetric: acc=0.708223

Evaluation at Epoch 12/20. Step:6792/11320. AccuracyMetric: acc=0.725022

Evaluation at Epoch 13/20. Step:7358/11320. AccuracyMetric: acc=0.723696

Evaluation at Epoch 14/20. Step:7924/11320. AccuracyMetric: acc=0.717507

Evaluation at Epoch 15/20. Step:8490/11320. AccuracyMetric: acc=0.709549

Evaluation at Epoch 16/20. Step:9056/11320. AccuracyMetric: acc=0.725022

Evaluation at Epoch 17/20. Step:9622/11320. AccuracyMetric: acc=0.732538

Evaluation at Epoch 18/20. Step:10188/11320. AccuracyMetric: acc=0.722812

Evaluation at Epoch 19/20. Step:10754/11320. AccuracyMetric: acc=0.736516

Evaluation at Epoch 20/20. Step:11320/11320. AccuracyMetric: acc=0.732095
```

模型结果较CNN稍好，但不如RNN，当然这和隐藏层数只有1层相关。

为了取得更好的效果，我参照2017年知乎看山杯冠军ensemble模型之一做了简单修改。修改后的版本相较基本模型做了如下改动（只选取了改动部分）：

```
#CNN layer
self.conv = nn.Sequential(
    nn.Conv1d(
        in_channels=hidden_size*2 + self.embed.embedding_dim,
        out_channels=content_dim,
        kernel_size=kernel_sizes
    ),
    nn.BatchNorm1d(content_dim),
    nn.ReLU(inplace=True),
    nn.Conv1d(
        in_channels=content_dim,
        out_channels=content_dim,
        kernel_size=kernel_sizes
    ),
    nn.BatchNorm1d(content_dim),
    nn.ReLU(inplace=True)
)

#fc
self.fc = nn.Sequential(
    nn.Linear(content_dim, linear_hidden_dim),
    nn.BatchNorm1d(linear_hidden_dim),
    nn.ReLU(inplace=True),
    nn.Linear(linear_hidden_dim,num_classes)
)
```

可以看出，主要区别在于2层更大的卷积、2层linear与relu激活函数。relu能一定程度上使网络稀疏，减少了参数的相互依存关系，缓解了过拟合问题的产生，这一点和dropout的作用有点类似。

采用改进后的rcnn后，10个epoch内模型也达到0.8左右的准确率：

```
Evaluation at Epoch 1/10. Step:566/5660. AccuracyMetric: acc=0.283378

Evaluation at Epoch 2/10. Step:1132/5660. AccuracyMetric: acc=0.634836

Evaluation at Epoch 3/10. Step:1698/5660. AccuracyMetric: acc=0.740495

Evaluation at Epoch 4/10. Step:2264/5660. AccuracyMetric: acc=0.784704

Evaluation at Epoch 5/10. Step:2830/5660. AccuracyMetric: acc=0.797082

Evaluation at Epoch 6/10. Step:3396/5660. AccuracyMetric: acc=0.794872

Evaluation at Epoch 7/10. Step:3962/5660. AccuracyMetric: acc=0.814766

Evaluation at Epoch 8/10. Step:4528/5660. AccuracyMetric: acc=0.802387

Evaluation at Epoch 9/10. Step:5094/5660. AccuracyMetric: acc=0.814766

Evaluation at Epoch 10/10. Step:5660/5660. AccuracyMetric: acc=0.792661
```

可以设想，若加深RNN层数，模型能取得更好的效果。

word2vec+CNN

word2vec除原始论文外，Rong Xin的word2vec Parameter Learning Explained解释的很好。在这里，我尝试用word2vec进行embedding操作，测试是否对模型结果提升有点帮助。

gensim包提供了word2vec的支持。我们首先将数据集输入word2vec训练，将模型训练的结果作为embedding传入卷积层，其它操作与普通基于CNN的方法类似。但出乎意料的是，在其它参数不变的前提下，模型训练准确率只达到0.68，甚至不如baseline。这是个值得探索的问题，找机会一探原因。

Requirement 2: thoughts about fastNLP

fastNLP凝聚了许多学长与老师的心血，也毫无疑问在很多方面做的很出色，在这里从正负两方面谈谈使用感想。

一些体验较好的部分：

- Vocabulary的封装非常方便实用：这我最喜欢的模块，功能虽然简单但是十分清晰，胜在没有冗余繁杂的部分，在接下来的机器学习过程中应该会一直沿用
- Dataset设计合理：在基本的功能外有基于ratio的划分、直接对csv的读取等，关于input、target的设定也和整个体系融合
- trainer有满满的细节：如提供input与target的shape、提供已进行的时间与待进行的时间等，都十分实用。jupyter notebook中暂停kernel时进度条由蓝变红也很细节
- 开放自由度：callback提供了很好的自由度，是一个很有潜力的模块，期待有关callback的更详细功能介绍

一些改进的建议：

- Dataset建议设置如pandas之类的自动省略部分输出的功能，在使用jupyter notebook查看Dataset某一field输出时总是一不小心就死机了
- trainer进度条后文字过密，容易直接忽视其中文字，其实loss, 训练时间等输出都是挺重要的
- 源码部分注释与内容不符，容易对读者造成困扰