深度学习 实验报告

计82 郑凯文 2018011314

实验简述

实验任务为使用深度学习进行Amazon商品评论数据集上的Rating预测。Rating分为1~5共5个等级，因此可以作为一个多分类问题。与此同时，由于我们使用RMSE作为评价指标，另一种选择是作为回归问题对待，给出Rating的实数预测值。

模型输入为用户的评论文本，因此是一个典型的nlp问题。在词嵌入方法、模型结构、优化目标上，体现着不同的权衡和选择。

Part 1: 简单网络

在这一部分中，我实现了若干类型的简易网络，使用预训练词向量进行Word Embedding并训练。

数据集处理

预训练词向量

使用GloVe 6B作为预训练词向量，其语料库为Wikipedia 2014 + Gigaword 5。为加速训练，选择的嵌入长度为50。

数据预处理

将每条数据的summary和reviewText拼接在一起，替换其中的标点符号，并用空格分隔为单词序列。

处理缺失词

对于单词序列，可能出现未在库中出现的词。对此，随机一个库中的词向量代替。

截取与填充

由于是序列数据，文章的长度并不是固定的，而这给 Pytorch 中 Dataloader 的使用和 mini-batch 的训练方式带来诸多不便。于是设定一个 padding 值，长度小于 padding 的文章 被 0 填充至 padding 长度，多余的部分则被截断。

在本节中，padding长度被设置为64，这可以保证比较重要的summary被保留，并保留一部分reviewText。

网络结构

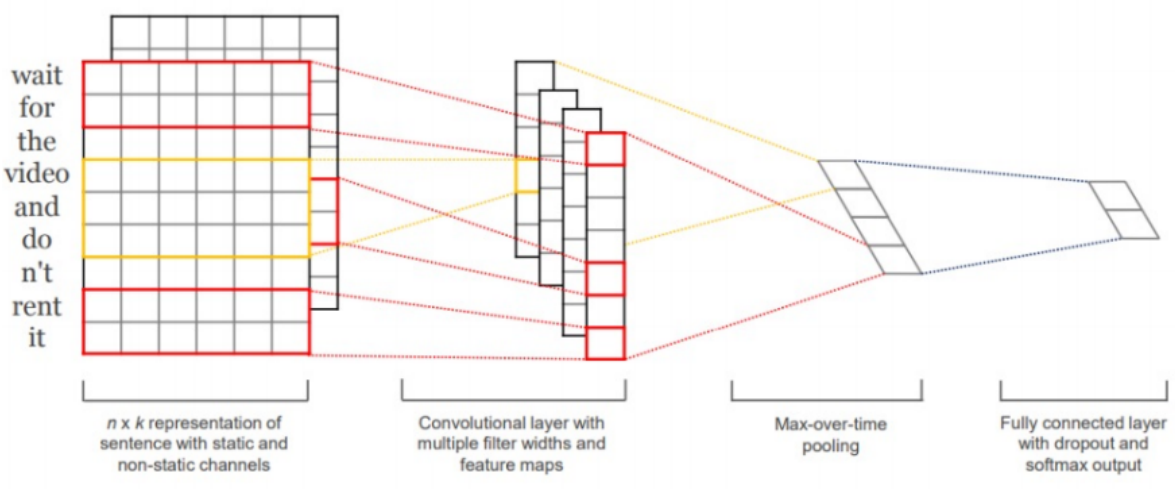
我实现了简易的MLP、CNN、RNN、双向LSTM四种网络模型进行训练和对比。

MLP

- Flatten
- FC(in_channels=50*padding, out_channels=512)
- Sigmoid
- FC(in_channels=512, out_channels=8)
- Softmax

CNN

我采取的CNN结构仿照了TextCNN进行设计



- conv1=Conv2d(in_channels=1, out_channels=100, kernel=(3,50))
- conv2=Conv2d(in_channels=1, out_channels=100, kernel=(4,50))
- conv3=Conv2d(in_channels=1, out_channels=100, kernel=(5,50))
- x1=MaxPool1d(conv1(x))
- x2=MaxPool1d(conv2(x))
- x3=MaxPool1d(conv3(x))
- Concate(Flatten(x1), Flatten(x2), Flatten(x3))
- FC(in_channels=300, out_channels=8)
- ReLU
- Softmax

RNN

利用 Pytorch 自带的 RNN 单元，接上全连接层

- RNN(hidden_size=64, num_layers=2)
- FC(in_channels=64*padding, out_channels=8)
- Softmax

双向LSTM

利用 Pytorch 自带的双向 LSTM 单元，接上全连接层

- BiLSTM(hidden_size=64, num_layers=2, dropout=dropout_rate)
- FC(in_channels=64*padding*2, out_channels=8)
- Softmax

损失函数

在这一节中，作为多分类任务看待。得到网络经过Softmax的输出后，使用交叉熵损失。

正则化

正则化是防止模型过拟合的方法，本实验中主要通过优化器的 weight decay 来控制。weight decay 对应于 L2 正则化，经尝试设置为0.05。

处理类别不平衡：类别权重

在之前的参数下尝试训练，我发现四种模型均出现了loss不下降、准确率维持在58.8%的情况。这个现象令我很疑惑，经过探究发现，这个准确率差不多是全部预测5的准确率。于是我恍然大悟，由于训练集类别不均衡，模型陷入了局部极小无法跳出。

我对5种类别进行统计，发现在训练集比例上，大约是0.46% : 0.46% : 10% : 22% : 58.7%。经过尝试，我在交叉熵损失上给类别赋予[0.3, 0.3, 0.15, 0.15, 0.1]的权重。

这种权重设置下，四种模型都可以跳脱出局部极小，随着训练的进行，准确率从58.8%开始有微小的提升。我也试过更极端的权重，如[0.4, 0.4, 0.08, 0.08, 0.04]，但此时四种模型准确率都在40%~50%波动，反而有负面效果。这是由于：模型的参数量较小，对rating较低的样本拟合不佳，而rating为5的样本权重过小，反而丧失了占据绝大多数的rating为5的样本的准确性，得不偿失。

数据集划分

固定随机种子，取训练集：验证集=9：1。在验证集上测试准确率、RMSE等指标的效果。

优化器

优化器的选择上，SGD、Momentum 和 Nesterov 是基本和略作修改的梯度下降法。Adadelta、RMSprop、Adam、Adamax 等则进行了各种策略的学习率自适应调整，整体效果相差不大。本实验使用了 Adam 优化器，学习率为 Pytorch 默认的推荐值 1e-3。

实验结果与分析

取训练100个epoch验证集上准确率最高的模型，各指标如下表

模型	Accuracy	MAE	RMSE
MLP	59.0%	0.6672	1.2063
CNN	61.3%	0.6419	1.1971
RNN	64.6%	0.5075	0.9800
双向LSTM	65.3%	0.4890	0.9552

从结果来看，TextCNN 模型的效果优于 MLP，其参数量也比MLP小了一个数量级，在更少的参数下达到更好的效果，这是由于其权值共享的结构使得网络具有更少的参数量，更易学习，且更容易抓住和提取局部特征。

与意料相符的是，RNN系列模型远远优于MLP、CNN，且双向LSTM略优于RNN。这说明对于文本分类问题，序列模型可以更好地处理信息的前后传递和联系。相对于RNN，LSTM有着记忆和遗忘机制和更多的参数，因此可以达到更好的效果。

与此结果相对地，之前在人工智能导论课上，我也尝试过CNN和RNN进行情感分析，但之前的时候CNN效果是远好于RNN的。这之间的区别在于，之前可能是一些情感特色浓厚的词决定了整体的类别；而在此次的数据集上，上下文信息对Rating的影响更重要，如有一些句子：if ..., it would get 5 starts，这种虚拟语态不能仅仅根据出现了'5 starts'就认定Rating为5。

Part 2: BERT

在上个部分中，我实现了几种简单的网络并进行了比较。但总体来说，准确率只比之前使用传统机器学习方法如SVM提升了一点，不能让人满意。接下来提高效果还有一些方法：

- 构建更复杂、更深的网络结构，增加参数量，提高网络的拟合能力

- 使用更前沿的模型，如加入注意力机制，使用Transformer结构等

综合以上两点，我决定尝试*BERT* (Bidirectional Encoder Representations from Transformers)。BERT的优势在于，提供了不同大小的预训练模型和预训练分词器，我们只需要在其基础上进行fine-tune。这样，可以免去调节网络结构、进行长时间从头训练的时间。

Version1: 多分类

在数据集的划分上，仍采用训练集：验证集=9:1的比例。对于每个样本，将summary和reviewText拼接在一起，截取一定长度，并使用BERT的预训练分词器进行处理。

在模型结构上，我只是在BERT预训练模型后接上了一个输出长度为5的全连接层，对应到5个分类，同样使用交叉熵作为损失函数。

我对不同的截断长度和模型大小进行了对照实验。我尝试的主要是两种大小的预训练模型：

- `bert-base-uncased`: 12-layer, 768-hidden, 12-heads, 110M parameters
- `bert-large-uncased`: 24-layer, 1024-hidden, 16-heads, 340M parameters

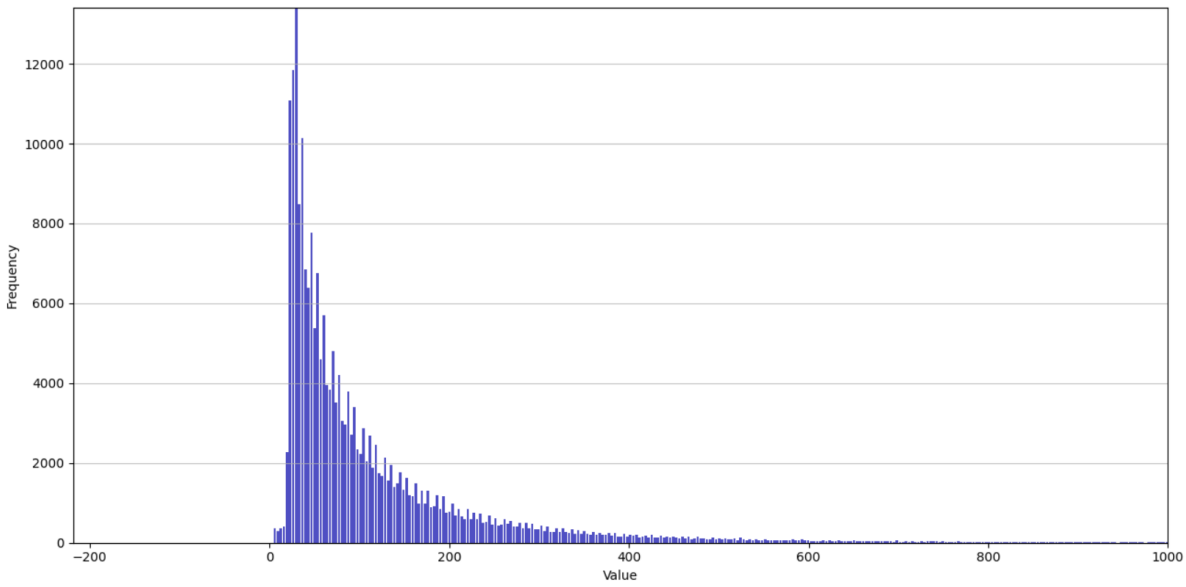
由于进行的是fine-tune，模型训练2~3个epoch便可达到最好效果，继续训练会发生过拟合。当然，由于模型较大，且有多达22W条数据，训练过程十分缓慢。

探究1： 句子截断长度

我选取了64和128两种截断长度，使用bert-base-uncased预训练模型，在验证集上的结果如下：

截断长度	Accuracy	MAE	RMSE
64	71.8%	0.3420	0.7114
128	73.2%	0.3215	0.6844

截断长度的增加可以保留更多的信息，从64%增加到128%可以带来准确率的明显提升。通过统计数据集中的词序列长度，可以画出如下的频数直方图：



当截断长度为128时，80%的样本可以保留80%以上的信息；而截断长度为64时，只有56%的样本可以保留80%以上的信息。当然，截断长度增加会增加显存占用、减缓训练速度，经过折衷，之后的实验中均设置截断长度为128。

探究2： 模型大小

在base和large两种大小的模型下，验证集结果如下：

模型	Accuracy	MAE	RMSE
bert-base-uncased	73.2%	0.3215	0.6844
bert-large-uncased	73.8%	0.3003	0.6319

使用更大的模型可以增加训练效果。虽然准确率提升不大（只提升0.6%），但若以RMSE为指标，其具有显著的降低，在Kaggle提交的large模型的预测文件更是达到了0.61562的RMSE值。

Version2: 回归

即使使用较大的BERT模型和较长的截断长度，多分类任务的RMSE也很难降至0.6以下。于是我转念一想，既然预测的Rating可以是连续的实数，不如将其作为回归任务。相比于多分类任务，进行的改变如下：

- 模型最后的全连接层输出长度改为1，即预测的实数值
- 使用MSE作为损失函数

可以发现，我们评价的指标是RMSE，而训练时恰好使用MSE作为损失，可谓是针对性非常强。相比于使用交叉熵时以多分类准确率为优化目标，回归任务对于RMSE指标的优化更直接，也可以预想到能达到更好的效果。

使用bert-large-uncased模型，截断长度128，回归任务最终在验证集上的结果为：

```
MAE: 0.3643229427487505 RMSE: 0.5604358416687345
```

在Kaggle上，提交结果为为0.55306。

这个优化目标的转变可谓是一阵见血的。但从另一方面，是否回归比多分类更好呢？只能说这取决于评价指标了，比如回归任务失去了准确率的概念，且可以看到MAE是不如之前的。只能说在RMSE指标下，回归任务的预测确实更“准确”，但若用MAE来评价和真实Rating之间的“距离”，那么结论恰恰相反。

Version3: 回归+集成

进一步地，我尝试了使用集成方法优化RMSE。我集成的方法是：先前训练集：验证集=9：1，那么我干脆做一个10折，取不同的1份作为验证集。这弥补了选取一部分作为验证集时，无法在训练时使用的问题。对不同的模型进行集成时，对于回归问题，直接将多份预测结果取平均。

由于训练十分缓慢，我仅仅集成了很少的模型：

集成个数	Kaggle RMSE
1	0.55306
2	0.54184
3	0.53813
4	0.53755

随着集成个数的增加，测试集的RMSE可以看到明显的减小，这也说明了集成方法的有效性：4个模型的RMSE均为0.55~0.57，但由于它们均划分了一部分样本作为验证集，在某些测试样本上一定出现了严重的“失误”；而4个模型的预测取平均，便能以长补短，修正失误。

最终结果与分析

最终，我采用4个BERT回归模型进行集成，在Kaggle上的结果为0.53755。

#	Team Name	Notebook	Team Members	Score	Entries	Last
1	nonstopfor			0.48775	22	2d
2	lemonramen3			0.49930	28	5h
3	xwwwwwwww			0.50903	8	5d
4	Xiangzhe Kong			0.52392	8	2d
5	JiayunWu			0.53063	3	3d
6	Wendi Zheng			0.53193	8	12h
7	Kaiwen Zheng123			0.53755	18	2m

Your Best Entry

Your submission scored 0.53755, which is an improvement of your previous score of 0.53813. Great job! Tweet this!

截止目前，rank 7/72，大约排在10%的位置。

从以上探究中，我有如下感想：

- 在nlp相关任务中，预训练词向量、预训练模型等十分关键。若单纯地设计更复杂的循环神经网络、手动加入注意力机制等，即使能进一步提升准确率，也需要付出巨大的训练时间、设计时间、测试时间。然而使用BERT的预训练模型，即使它复杂度很高、训练很慢，但我只需要fine-tune两三个epoch就有极好的效果了
- 需要根据评价指标设计优化目标，如将多分类问题转化为回归问题以尽量降低RMSE，合理的方法可以达到事半功倍的效果
- 集成方法永不过时，尤其在Kaggle这种刷榜项目中，多训练几天，集成一下，往往是稳定提高表现的万能方法