实验4-深度学习实验报告

陈意扬 计96 2019011341

实验目的

1）实现一种/多种深度学习模型并在给定数据集上测试

2）比较不同深度学习模型在相同任务上的性能

3）分析模型超参对模型性能的影响

实验原理

`cnn`：

`cnn`（卷积神经网络）是一种前馈神经网络，它的人工神经元可以响应一部分覆盖范围内的周围单元，对于大型图像处理有出色表现。其特有的

1.卷积层（`convolutional layer`）可以产生一组平行的特征图（`feature map`），它通过在输入图像上滑动不同的卷积核并执行一定的运算而组成。

2.池化层（`pooling layer`）提供一种非线性形式的降采样。有多种不同形式的非线性池化函数，而其中"最大池化（`Max pooling`）"是最为常见的。它是将输入的图像划分为若干个矩形区域，对每个子区域输出最大值。

(参考维基百科)

`LSTM`：

`LSTM`（`Long Short Term Memory networks`）由Hochreiter & Schmidhuber (1997)引入，是一种特殊的RNN网络，该网络设计出来是为了解决长依赖问题。`LSTM`的核心是神经元状态，通过设置忘记门、输入门和输出门对神经元状态进行删除或者添加信息。

（参考链接：https://www.jianshu.com/p/95d5c461924c）

`BERT`：

`BERT`的全称为`Bidirectional Encoder Representation from Transformers`，是一个预训练的语言表征模型。它强调了不再像以往一样采用传统的单向语言模型或者把两个单向语言模型进行浅层拼接的方法进行预训练，而是采用新的`masked language model`（`MLM`），以致能生成深度的双向语言表征。`BERT`论文发表时提及在11个`NLP`任务中获得了新的`state-of-the-art`的结果，令人目瞪口呆。

该模型有以下主要优点：

1）采用`MLM`对双向的`Transformers`进行预训练，以生成深层的双向语言表征。

2）预训练后，只需要添加一个额外的输出层进行`fine-tune`，就可以在各种各样的下游任务中取得`state-of-the-art`的表现。在这过程中并不需要对`BERT`进行任务特定的结构修改。

（参考链接：https://zhuanlan.zhihu.com/p/98855346）

实验步骤

1.文本向量化

`BERT`的文本向量化

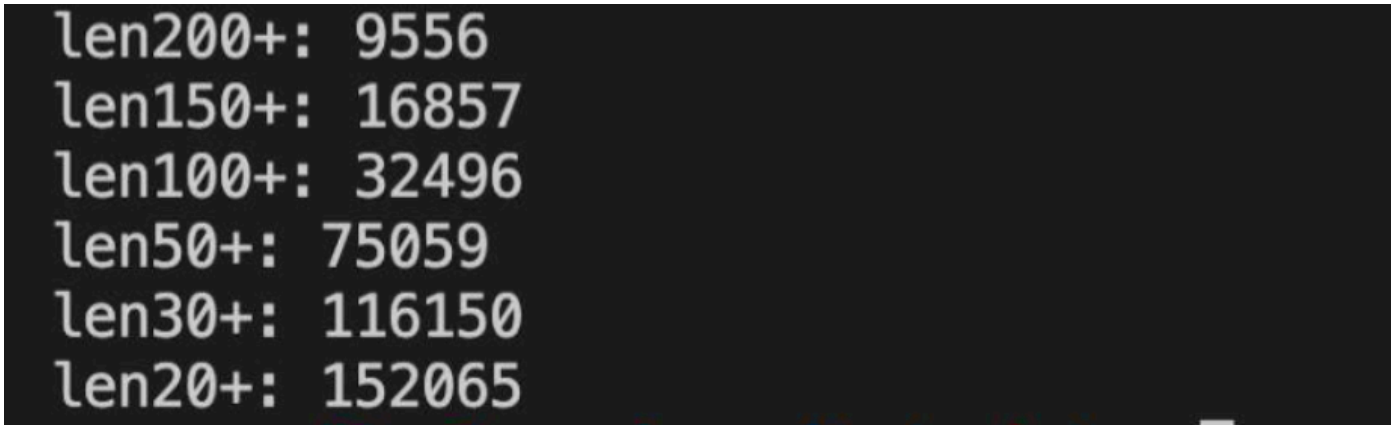
使用的是预训练模型自带的`Tokenizer`，将文本直接输入，`Tokenizer`自动完成向量化。对于每个评论，我将`summary`和`reviewText`按序拼接成长字符串，向量化时，最大长度设置为512（`BERT`默认最大长度）。

```
from transformers import AutoTokenizer

checkpoint = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)
sequence = "I've been waiting for a HuggingFace course my whole life."

model_inputs = tokenizer(sequence, padding="max_length", truncation=True)
print(model_inputs["input_ids"])

>[101, 1045, 1005, 2310, 2042, 3403, 2005, 1037, 17662, 12172, 2607, 2026, 2878, 2166,
1012, ... ,102]
```

`cnn`和`LSTM`的文本向量化（word2vec.py）

我采用了`word embedding`方法，具体是`gensim`库中的`Word2Vec`。

提供的数据集共有22w条对商品的打分以及简评，存储在`exp3-reviews.csv`中，每个评论分为`overall`、`reviewerID`、`asin`、`unixReviewTime`、`summary`和`reviewText`6栏，我提取`overall`部分的打分作为label，并将`summary`和`reviewText`这两栏作为特征进行向量化。对文本进行分词和去停用词后，我将`reviewText`拆分成句子（`summary`作为1个句子），将句子集进行`Word2Vec`训练，将每个词训练成50维的向量（去掉词频低于10的词）。`vocabulary`中包含5w左右个词。把`summary`和`reviewText`合并作为一个文本后进行向量化，首先进行长度统计



长于100词的文本不到15%，长于150词的文本不到8%，所以我选择选择128词（2的整数次幂，加快训练速度）作为标准长度，对文本按词出现的顺序将每个词对应的50维向量排列成128x50的张量（不到128词的文本用0向量填充，超过128词的文本进行截断）。

2.数据集采样和切分

为了节约时间，我采样了数据集的十分之一，并用`sklearn.model_selection.train_test_split`以10比1划分数据集（`train set`20000；`test set` 2000）。为了比较不同模型在相同任务上的性能，要保证每次的训练集和测试集相同，于是我在第一次划分后将结果存储进`npy`文件中。加载数据集后将训练集和测试集分别打包进`train_dataloader`和`test_dataloader`中，这样可以方便地`shuffle`和按`batch`训练。`batch size`设置为32。

3.模型训练

`cnn`（`cnn.py`）

卷积层使用了三个卷积核，之后一个全连接层到输出。每个卷积核由卷积函数、激活函数`ReLU`、最大池化函数和dropout函数组成。由于输入的张量是128个50维向量堆叠而成，每个50维向量代表一个词，所以卷积核不应该在50维中做切分破坏词的粒度，横向维度必须为50。由此三个卷积函数我选择了3x50、4x50和5x50的size。输出通道数（CHANNEL）和丢弃率（DROPOUT）都作为超参在训练过程中调整。

```
        self.core1 = nn.Sequential(
            nn.Conv2d(1, CHANNEL, (3, 50)),
            nn.ReLU(),
            nn.MaxPool2d((128 - 2, 1)),
            nn.Dropout(DROPOUT)
        )
        self.core2 = nn.Sequential(
            nn.Conv2d(1, CHANNEL, (4, 50)),
            nn.ReLU(),
            nn.MaxPool2d((128 - 3, 1)),
            nn.Dropout(DROPOUT)
        )
        self.core3 = nn.Sequential(
            nn.Conv2d(1, CHANNEL, (5, 50)),
            nn.ReLU(),
            nn.MaxPool2d((128 - 4, 1)),
            nn.Dropout(DROPOUT)
        )
        self.fullconnect = nn.Sequential(
            nn.ReLU(),
            nn.Linear(3 * CHANNEL, 5)
        )
```

`LSTM`（`rnn.py`）

使用`torch.nn.LSTM`函数，接上一个全连接层到输出。隐藏层神经元数（H）和LSTM层数（LAYERS）作为超参在训练过程中调整。

```
class RNN(nn.Module):
    def __init__(self):
        super().__init__()
```

```
        self.lstm = nn.LSTM(input_size=50, hidden_size=H,
                            num_layers=LAYERS, batch_first=True, dropout=DROPOUT)
        self.fullconnect = nn.Sequential(
            nn.ReLU(),
            nn.Linear(H * 128, 5)
        )

    def forward(self, inputs):
        states, hidden = self.lstm(inputs)
        outputs = self.fullconnect(
        states.reshape(inputs.shape[0], H * 128))
        return outputs
```

`BERT`（`bert.py`）

使用`transformers`库中的预训练模型`bert-base-uncased`，对于文本分类任务，只用提取文本开头标记`[CLS]`的最后隐藏状态（768维），通过全连接层到输出。由于`BERT`模型参数量实在是太大，手上又没有很好的算力（学校提供的平台用不来），`BERT`模型没有进行调参。

```
class MyClassifier(nn.Module):
    def __init__(self, ):
        super().__init__()

        self.bert = BertModel.from_pretrained(checkpoint)

        self.fullconnect = self.fullconnect = nn.Sequential(
            nn.ReLU(),
            nn.Linear(BERT_H, OUTPUT)
        )

    def forward(self, input_ids, attention_mask):
        outputs = self.bert(input_ids=input_ids,
                            attention_mask=attention_mask)
        bert_state = outputs[0][:, 0, :]
        logits = self.fullconnect(bert_state)
        return logit
```

模型的评价（/results）

由于是多分类，我采用了2个评价指标：

1.准确率`acc`；

2.`MAE`

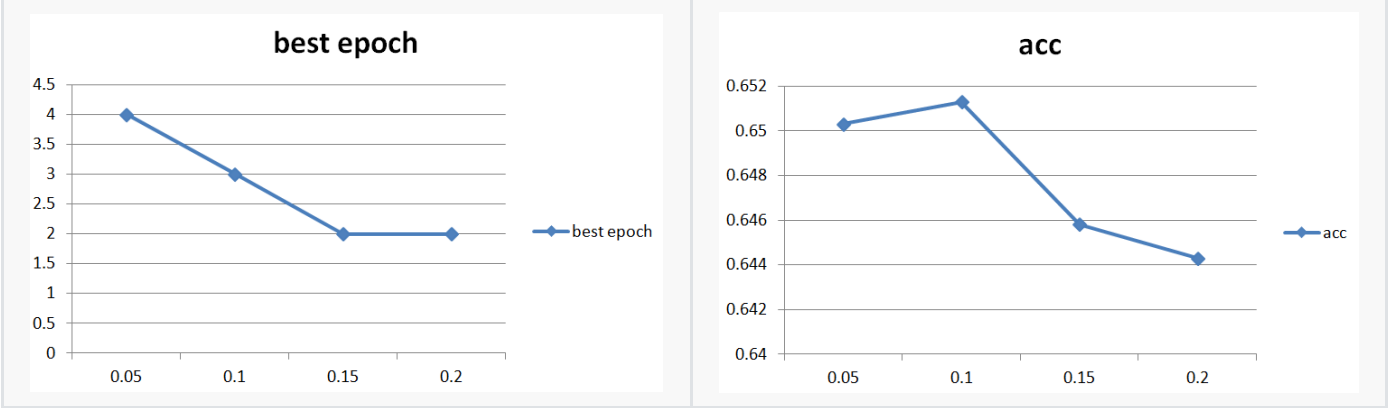
3.`RMSE`

每次训练任务都设置10个epoch，下表记录acc表现最佳的epoch的数据

CNN

CHANNEL；DROPOUT	acc；MAE；RMSE；epoch
100；0.05	65.03%；0.5263；0.9903；4
100；0.10	65.13%；0.5203；0.9858；3 *
100；0.15	64.58%；0.5337；1.0005；2
100；0.20	64.43%；0.5278；0.9841；2
2；0.10	62.40%；0.6190；1.1472；9
20；0.10	64.53%；0.5650；1.0712；3
40；0.10	64.04%；0.5362；0.9966；6
80；0.10	63.79%；0.5809；1.0964；3
100；0.10	65.13%；0.5203；0.9858；3

观察数据可以明显发现随着DROPOUT提升，模型在训练集上表现最好的epoch序号前移，我认为是因为丢弃一些神经元后模型每次需要摄取的信息和调整的参数变少，能够更快收敛；模型性能在DROPOUT=0.10左右在测试集达到最优，这跟传统cnn的最适DROPOUT有一些出入（0.5左右）。我推测原因在于训练集和测试集特征粒度的一致性比较好，用较高耦合度的特征提取可以得到比较好的结果。



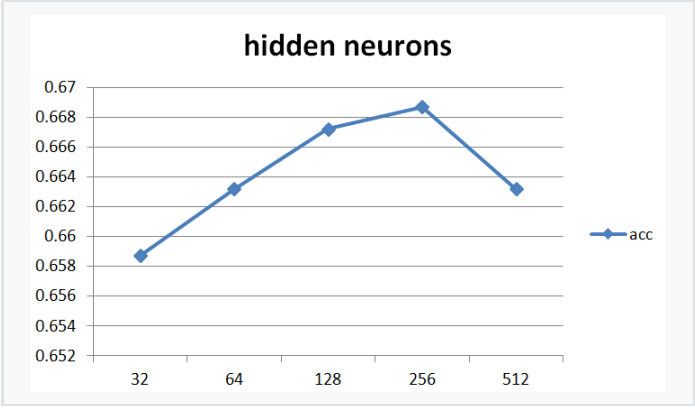
相反，输出通道数对于CNN性能影响不大，除了极小时（CHANNEL=2）模型性能较差，在合适范围（20～100）内模型性能上下波动，没有明显变化趋势。

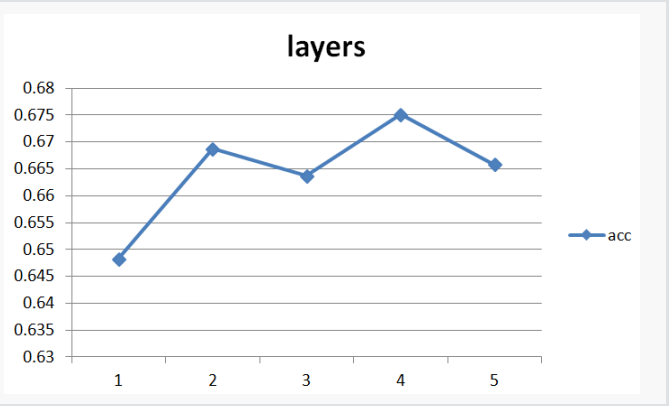
LSTM

H；LAYER	acc；MAE；RMSE；epoch
32；2	65.87%；0.4936；0.9439；5
64；2	66.32%；0.5074；0.9776；2
128；2	66.72%；0.4916；0.9501；3
256；2	66.87%；0.4886；0.9517；2
512；2	66.32%；0.4782；0.9195；2
256；1	64.83%；0.5585；1.0589；2
256；2	66.87%；0.4886；0.9517；2
256；3	66.37%；0.5064；0.9845；3
256；4	67.51%；0.4539；0.8844；6 *
256；5	66.58%；0.5149；1.0073；3

可以看到LSTM收敛的epoch数明显较少，且超参的调整对模型整体的性能影响比较有限。

隐藏层神经元个数和层数对LSTM性能的影响





可以看到随着隐藏层神经元个数增加，模型整体性能上升到峰值66.87%后开始下降。刚开始随着隐藏层神经元个数增加，模型学习能力增强，性能上升；神经元过多后很快就过拟合，性能开始下降。

LSTM层数对模型性能影响没有明显趋势，除了只有一层时，LSTM无法体现循环记忆特性，性能较差，当层数为复数层时性能上下波动。

BERT

acc；MAE；RMSE
59.38%；0.7798；1.3792

令人惊讶的是BERT模型在这个分类任务上表现得很差。后来经过分析，表现得很差可能是数据集规模比较小。该模型是在维基百科词条数据集上预训练得到，和本次实验数据集的文体有差异，并且训练集规模相比预训练集太小，finetune做得不到位，所以性能较差，如果用全部22w条评论finetune会得到更好的结果（算力不允许）。

综合比较

本次任务各个深度学习模型调参后的表现

CNN	LSTM	BERT
65.13%；0.5203；0.9858	67.51%；0.4539；0.8844	59.38%；0.7798；1.3792

训练时间（单个epoch）

Ubuntu 20.04 Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz

CNN	LSTM	BERT
5s	6min	8h

LSTM在性能上最优；考虑效率，CNN在短时间就达到了不错的性能；BERT在耗时和性能上都表现最差。对比实验三的实验数据，较简单分类算法（svm & dt）提升了20%左右。

实验结论

对于本次分类任务

单考虑性能

LSTM>CNN>BERT

考虑效率

CNN>LSTM>BERT

超参方面，dropout rate对CNN性能有明显的影响，隐藏层神经元数对LSTM性能有明显的影响。

BERT表现较差的原因是数据集规模太小。

对比实验三的数据（svm 43% ；decisionTree 56%），所有深度学习模型都表现更好，将准确率提升20%左右。