中文文本分类作业

魏少杭
学号：20373594
学院：人工智能研究院

2022 年 11 月 28 日

目录

1 简介

在课上，我们学习了对于文本分类问题的几类深度学习模型，第一个是 fast_Text，它具有结构简单的特点，分类效果较好、运算速度极快。第二个是 TextCNN，TextCNN 使用经过词向量表达的文本，因此输入的是一维数据，首先用一维卷积，设计不同的 filter_size 的 filter 获得不同宽度的视野，然后 Pooling 池化，保留 k 个最大的信息；其主要问题是固定了 filter_size 的视野，对于像作业中这样的文本长度分布极不均匀的问题，难以提取合适的上下文信息作为特征提取的参考。第三个是 TextRNN，作为递归神经网络，其能够更好地表达上下文信息，在文本分类任务中，双向 LSTM 可以捕获变长、双向的 n-gram 信息。在 TextRNN 基础上，加入 Attention 机制可以直观地通过给出每一个词对结果的贡献。此

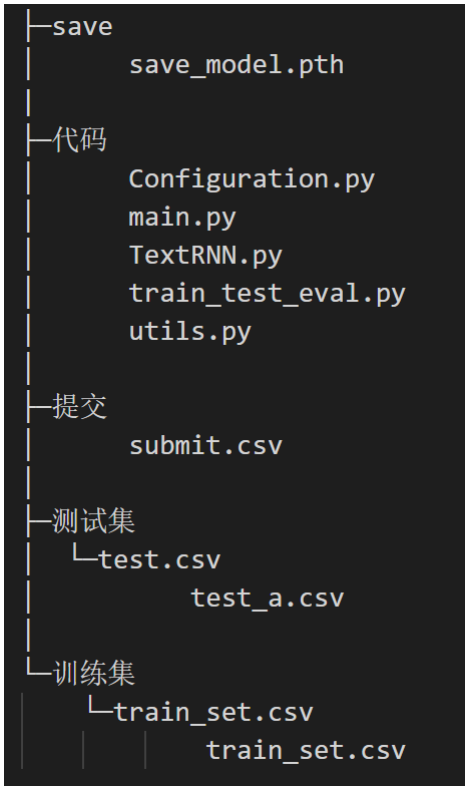

图 1: 此次作业文件关系

外，还有 TextRCNN，用前后向 RNN 获得每一个词的前向后向上下文表示，然后将当前词向量和上下文 concat 起来，最后使用 TextCNN 相同卷积层、池化层。

在这里，我使用 TextRNN（即 Bi-directional RNN）的原因有两点：第一，由于初次接触深度学习中对自然语言等序列文本进行处理的模型，有必要以基础的重要模型循环神经网络作为入手，实践一遍；第二，由于个人能力和时间问题，没有精力深入学习实践其他如 Attention、Transformer 以及 Bert 预训练下的模型，同时经检索网络资料发现 TextRNN 的分类能力已经很强，所以先选择了 TextRNN 作为本次作业的模型。

我将分为 1. 数据处理、包装数据集；2. 主函数的实现；3.TextRNN 的实现；4. 超参数调试与理解作为本次作业的说明，最后是自己的收获和反思。

图 1是本次作业的文件依赖关系。

2 实现过程

2.1 数据处理、包装数据集

在 utils.py 中，定义了本次使用算法之前的数据处理和其他工具函数，包括了读取 csv 文件（read_csv）、从 csv 文件中获取 labels（get_labels）、获取网站给的预处理过后的 text 列表（get_corpus）、获取词典的大小（get_vocab_size）、创建数据集（build_dataset）、创建

迭代器 (build_iterator)、获取运行时间 (get_time_dif) 的函数，还定义了 DatasetIterator 类，用于按照 batch_size 来切分数据集、将数据集转化为可训练的 tensor，并且包装成一个迭代器用于取用。

构建数据时，首先对 df 进行解析，分别取 label 列和 text 列作为结果，然后构建 dataset。由于本次的数据集每一行代表一个文本（及其分类结果），文本本身按照 char 级别划分，text 长度各不同，而进行 embedding 之前需要使所有的样本的 text 化为一样的长度，所以我们预先给定 pad_size，然后“多删少补”，并在此过程中确定 sequence 的长度。返回一个总的 dataset，每一个样本为 ((text, label), seq_len)

对所有样本的 text 进行统一长度处理

```
def load_dataset(corpus_lst, label_lst, vocab_size, pad_size):
    contents = []
    for i in range(len(corpus_lst)):
        content = corpus_lst[i]
        seq_len = len(content)
        if seq_len < pad_size:
            content.extend([vocab_size+1]*(pad_size-seq_len))
        else:
            content = content[:pad_size]
            seq_len = pad_size
        label = label_lst[i]
        contents.append((content, int(label), seq_len))
        # contents.append((content, int(label)))

    return contents
```

再对 dataset 构造 Iterator。其中主要是构造一个 Class DatasetIterator，其包含了对迭代器返回的 batch 属性的初始化，包括 batches 数量、大小以及将样本进行训练时的 device。

核心是构造迭代取值的函数 ___next___，每次取的时候，先切片，再对索引加 1，最后返回 batches。

最后返回一个被切分好了的数据集（每个样本数据都是可训练的 Tensor 与 seq_len 构成的 tuple、一个 label 值）。

2.2 主函数的实现

首先，定义环境 config，这是一个在 Config 模块中定义的类，其中存了绝大多数的文件路径、超参数。

其次，开始读取数据、构造数据集。

然后，定义模型，并将它放到 device 上训练。初始化所有的 params。如果存在之前训练好的模型，直接加载模型参数继续训练。

最后，利用训练数据集进行训练。所有 epoch 结束后，利用测试数据集运行模型，并输出预测的新闻分类标签到 csv 文件中。

2.3 TextRNN 实现

模型本身的实现如下：

Modle

```
class Model(nn.Module):
    def __init__(self, config) -> None:
        super(Model, self).__init__()
        seq_len, embed_dim)
        self.embedding = nn.Embedding(config.vocab_size+2, config.embed_dim,
             padding_idx=config.vocab_size+1)
        self.lstm = nn.LSTM(config.embed_dim, config.hidden_size,
                            config.num_layers, bidirectional=True,
                            batch_first=True, dropout=config.dropout)
        self.fc = nn.Linear(config.hidden_size * 2, config.num_classes)

    def forward(self, x):
        x, _ = x        # 去掉最后一维
        out = self.embedding(x)
        out, _ = self.lstm(out)
        out = self.fc(out[:, -1, :])    # 取最后的隐状态
        return out
```

embedding 层的输入 shape 为 (batch_size, seq_len, vocab_size) 而输出为 (batch_size, seq_len, embed_dim)。

embedding 后，将输入两层双向 LSTM，并加入 dropout=0.5。在这个双向 RNN 层中，输出为 (batch_size, seq_len, hidden_size)，相当于每一个样本输出为 seq_len 长的隐藏状态。每一个隐藏状态，又有 hidden_size*2 的大小。这里使用 dropout 的理解是：对于一个深度学习模型，可能会出现在训练数据上过拟合现象。具体地对于 RNN 而言，其参数量巨大，为了减少过拟合，我们可以用 dropout 来随机丢弃一些参数。考虑 RNN 的模型，对于每一个隐形状态 h_t，其是由上一个隐形状态 h_(t-1) 和当前值 x_t 分别乘上权重 U、V 产生的。因为 LSTM 本身的设计目的就是需要随着时间的变化进行长序列的记忆存储，双向 RNN 更是考虑到了前后文状态共同作用于当前词的情况，所以在隐状态过长的延续过程中，隐状态之间的作用权重不应该被 dropout，否则共享的权值 U 可能会趋于 0，消失掉；而每一个当前状态对于新隐状态产生的贡献（即对应的权重 V）则是可以修正的，我们可以对它 dropout。

最后一层就是全连接网络，我们取 LSTM 输出的所有时间序列的隐状态中最后一个（即当前隐状态）作为输入，然后输出 14 维分类结果对应的值。

由于是分类问题，我们使用 CrossEntropy 作为 Loss 函数；优化器为 Adam 优化器（因为比较稳定）。

2.4 超参数调试与理解

这次模型的所有超参数、文件路径全部在 Configuration.py 中集中实现。

文件路径包含了 train、test 数据集的路径、checkpoint 的保存路径，对 test 预测的输出 csv 路径、用于训练的 device 等。

超参数有：数据集构建过程中的 batch_size、vocab_size（这个不能自己调，而是根据题目预处理后的结果进行计数）、pad_size（即所有样本 text 长度的合理值）；模型参数：dropout 值（0.5 是从网上查到建议值）、embed_dim 即 embedding 的时候对每一个样本 text 映射后的值的个数、hidden_size 为每一个隐藏状态的维度、LSTM 层数 num_layers=2、以及分类结果维度数（即新闻类别数）num_classes=14；训练过程的参数：num_epochs、lr。

所有超参数中，比较重要的有 pad_size。在调试过程中，我遇到的问题有 pad_size>150 时就会使得自己的电脑 RAM 或者 Colab 的 RAM 无法承载而报错，CUDA 也不能加载。因此我选择 2 的倍数 128。

lr 不能太大，否则对于双层 RNN 这个模型容易梯度爆炸。此外，为了避免梯度爆炸，还可以实行剪枝操作。

Config

```
    class Config():
    """
    说明:
        直接在Config里面写超参数、文件名
    """
    def __init__(self) -> None:
        self.train_set_path = "./Practice3_News Text Classification/训练集/
            train_set.csv/train_set.csv"
        self.test_set_path = "./Practice3_News Text Classification/测试集/
            test.csv/test_a.csv"
        save_dir = os.path.join('.', 'save')
        if not os.path.exists(save_dir):
            os.makedirs(save_dir)
        self.save_path = os.path.join(save_dir, 'save_model.pth')
        output_dir = os.path.join('.', 'output')
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)
        self.output_csv_path = os.path.join(output_dir, 'submit.csv')

        self.device = torch.device('cuda' if torch.cuda.is_available() else
            'cpu')
        # self.device = torch.device('cpu')


        self.vocab_size = 0 # 在utils中产生
        self.pad_size = 128  # log_2(vocab_size = 7000)  TODO:
```

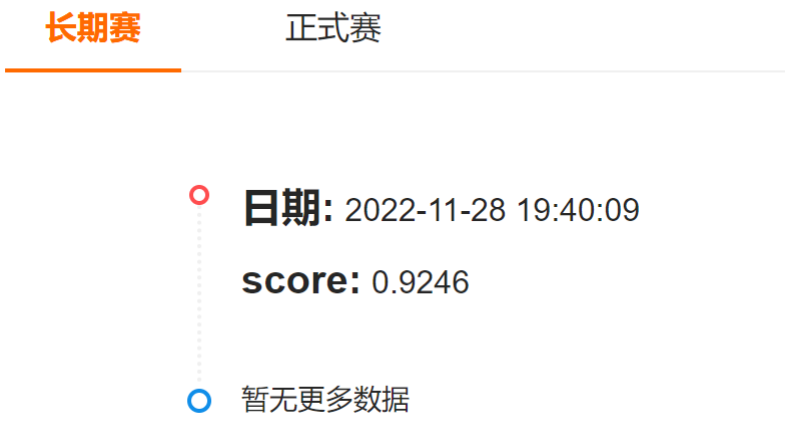

图 2: 结果截图

```
        self.batch_size = 128   # TODO:
        self.num_epochs = 1    # TODO:
        self.lr = 1e-3
        self.dropout = 0.5      # 随机失活
        # self.max_improve = 1000     # 超过1000次之后若不在更新则提前结束训
            练
        self.embed_dim = 300    # 词向量的维度
        self.hidden_size = 128  # lstm隐藏状态的维度
        self.num_layers = 2     # lstm共2层
        self.num_classes = 14
```

3 结果

将测试集中的数据进行分批输入模型,输出结果,提交天池平台测试结果为:F1score=0.9246,如图 2

4 收获和反思

4.1 NLP 中序列数据集构建

在构建数据集的时候，不同样本经常会出现较明显的长度差异，需要预先统计整个训练集、数据集的样本中的长度分布情况，尽可能按照样本 text 长度平均值或中位数作为 pad_size 来训练，然而，本题出现的问题就是由于训练样本较大，对 RAM 提出了很高的要求。由于我是在 Colab 免费版和本地笔记本尝试训练的，所以效果没有太好。后来才注意到阿里云本身提供了训练平台，以后可以尝试这类平台。

在构建迭代器的时候，需要注意所有样本按 batch_size 划分后可能会有残余部分，需要专门设置 batch_n 等进行标记，以便将所有样本都能用于生成迭代器。

4.2 TextRNN 理解

对于 TextRNN，使用双向 LSTM 来表示对上下文信息的同时提取，展现了良好的上下文理解能力。但是本次遗憾的是没有仔细研究 Attention 机制，这种机制将有助于更加突出地提取必要特征，将在以后的相关实践机会中好好学习。