Mo Lab4 Report - Writer Style Recognition

万晨阳 3210105327

1. 实验内容介绍

1.1 实验背景

作家风格是作家在作品中表现出来的独特的审美风貌。

通过分析作品的写作风格来识别作者这一研究有很多应用，比如可以帮助人们鉴定某些存在争议的文学作品的作者、判断文章是否剽窃他人作品等。

作者识别其实就是一个文本分类的过程，文本分类就是在给定的分类体系下，根据文本的内容自动地确定文本所关联的类别。

写作风格学就是通过统计的方法来分析作者的写作风格，作者的写作风格是其在语言文字表达活动中的个人言语特征，是人格在语言活动中的某种体现。

1.2 实验要求

- 建立深度神经网络模型，对一段文本信息进行检测识别出该文本对应的作者。
- 绘制深度神经网络模型图、绘制并分析学习曲线。
- 用准确率等指标对模型进行评估。

1.3 实验环境

可以使用基于 Python 分词库进行文本分词处理，使用 Numpy 库进行相关数值运算，使用 PyTorch 等框架建立深度学习模型等。

1.4 参考资料

- jieba: https://github.com/fxsjy/jieba
- Numpy: https://www.numpy.org/
- Pytorch: https://pytorch.org/docs/stable/index.html
- TorchText: https://torchtext.readthedocs.io/en/latest/

2. 模型建立与训练

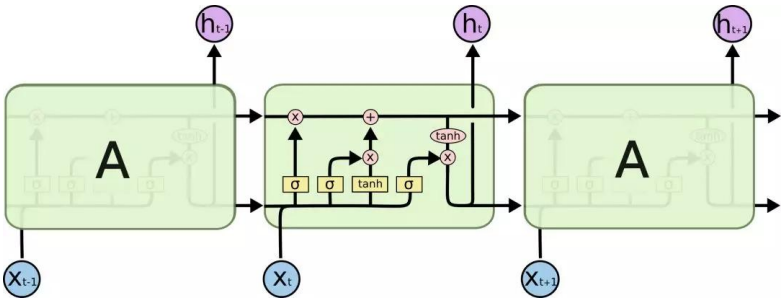
2.1 思路介绍与模型选择

2.1.1 基本思路

对于该任务，我查询了有关的论文资料。在文本分类方面表现较好的有以下几种网络模型： Bi-LSTM with Attention、RCNN、Adversarial LSTM、Transformer、ELMo、BERT等。考虑到网络实现的难度和任务的难度，我选择使用结构相对比较简单的网络完成这一任务：一种思路是基于Bi-LSTM，也即双向长短期记忆网络（含注意力机制）进行实现。

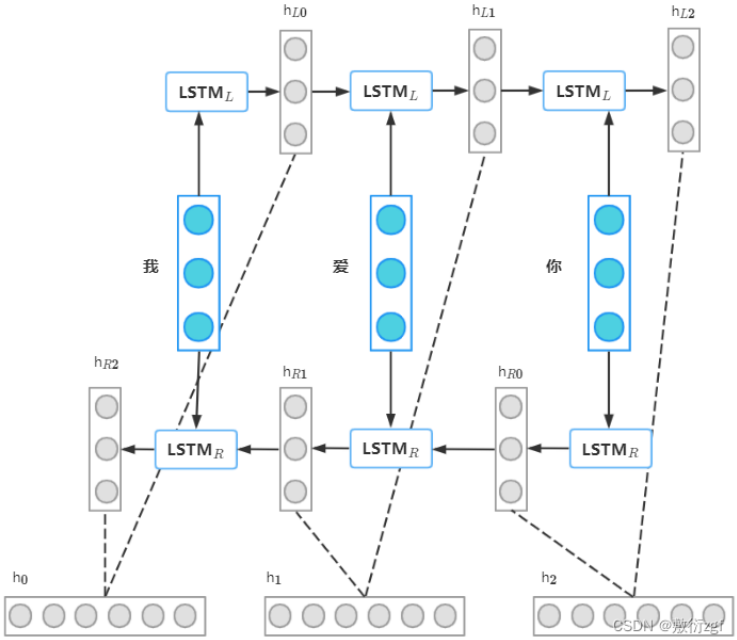
2.1.2 Bi-LSTM with Attention 介绍

LSTM的全称是Long Short-Term Memory，它是RNN（Recurrent Neural Network）的一种。LSTM由于其设计的特点，非常适合用于对时序数据的建模，如文本数据。BiLSTM是Bi-directional Long Short-Term Memory的缩写，是由前向LSTM与后向LSTM组合而成。两者在自然语言处理任务中都常被用来建模上下文信息。如下是一个LSTM的基本结构：

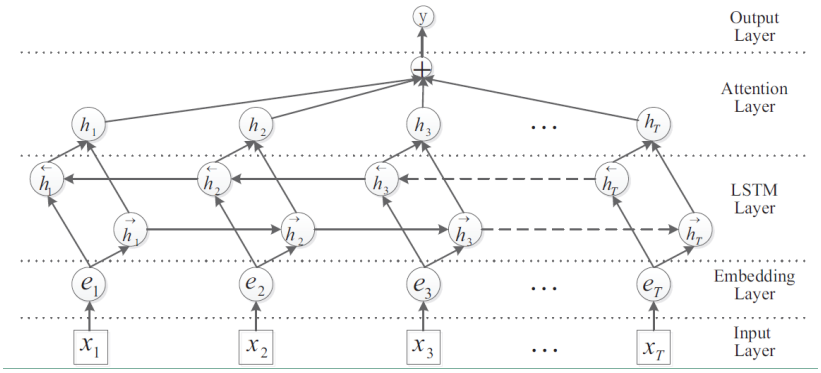


将词的表示组合成句子的表示，可以采用相加的方法，即将所有词的表示进行加和，或者取平均等方法，但是这些方法没有考虑到词语在句子中前后顺序。如句子“我不觉得好”。“不”字是对后面“好”的否定，即该句子的情感极性是贬义。使用LSTM模型可以更好的捕捉到较长距离的依赖关系。因为LSTM通过训练过程可以学到记忆哪些信息和遗忘哪些信息。但是利用LSTM对句子进行建模还存在一个问题：无法编码从后到前的信息。在更细粒度的分类时，如对于强程度的褒义、弱程度的褒义、中性、弱程度的贬义、强程度的贬义的五分类任务需要注意情感词、程度词、否定词之间的交互。**通过BiLSTM可以更好的捕捉双向的语义依赖。**

单层的BiLSTM是由两个LSTM组合而成，一个是正向处理输入序列；另一个反向处理序列，处理完成后将两个LSTM的输出拼接起来。如下图中，前向的LSTM依次输入“我”，“爱”，“你”得到向量结果，而反向的LSTM依次输入“你”，“爱”，“我”得到向量结果。两向量拼接作为BiLSTM的输出结果。



加入注意力机制的好处是能够加权的完成学习任务，也就是能够使其重点关注输入序列中的某些部分。在Bi-LSTM中我们会用最后一个时序的输出向量作为特征向量，然后进行softmax分类。Attention是先计算每个时序的权重，然后将所有时序的向量进行加权和作为特征向量，然后进行softmax分类。在实验中，加上Attention确实对结果有所提升。其模型结构如下图：



2.2 Bi-LSTM 网络的建立与训练

2.2.1 Word Embedding

在训练之前我们要对文本进行处理。首先要做的就是Word Embedding工作，将词汇转化为向量，这样才能够输入网络进行计算。我们使用GloVe完成这部分工作。GloVe的全称叫Global Vectors for Word Representation，它是一个基于全局词频统计（count-based & overall statistics）的词表征（word representation）工具，它可以把一个单词表达成一个由实数组成的向量，这些向量捕捉到了单词之间一些语义特性，比如相似性（similarity）、类比性（analogy）等。我们通过对向量的运算，比如欧几里得距离或者cosine相似度，可以计算出两个单词之间的语义相似性。

2.2.2 网络结构

基于论文内容以及本任务的实际需要，我对网络结构进行一定的调整之后实现了简化的BiLSTM+Attention。代码如下：

```
class BiLSTM_Attention(nn.Module):
    def __init__(self, vocab_size, embedding_dim, hidden_dim, n_layers):
        super(BiLSTM_Attention, self).__init__()
        self.hidden_dim = hidden_dim
        self.n_layers = n_layers
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.rnn = nn.LSTM(embedding_dim, hidden_dim,
                           num_layers=n_layers, bidirectional=True, dropout=0.5)
        self.fc = nn.Linear(hidden_dim * 2, 5)
        self.dropout = nn.Dropout(0.5)

    def attention_net(self, x, query, mask=None):
        d_k = query.size(-1)
        scores = torch.matmul(query, x.transpose(1, 2)) / math.sqrt(d_k)
        p_attn = F.softmax(scores, dim=-1)
        context = torch.matmul(p_attn, x).sum(1)
        return context, p_attn

    def forward(self, x):
        embedding = self.dropout(self.embedding(x))
        output, (final_hidden_state, final_cell_state) = self.rnn(embedding)
        output = output.permute(1, 0, 2)
        query = self.dropout(output)
        attn_output, attention = self.attention_net(output, query)
        logit = self.fc(attn_output)
        return logit
```

下面对于整个流程分别进行介绍：

初始化

我们定义了一个双向LSTM网络，它使用了嵌入层将输入转换为嵌入向量，然后通过LSTM层处理这些嵌入向量，最后通过全连接层输出分类结果。此外，我们还使用了dropout层和注意力机制来提高模型的性能。在`__init__`方法中，我们首先调用了父类的`__init__`方法，然后定义了一些网络需要的参数和层。`hidden_dim`是隐藏层的维度，`n_layers`是网络的层数。`self.embedding`是一个嵌入层，它将输入的词汇索引转换为嵌入向量。`self.rnn`是一个双向LSTM层，它接收嵌入向量作为输入，并输出隐藏状态和单元状态。`self.fc`是一个全连接层，它接收LSTM的输出，并输出最终的分类结果。`self.dropout`是一个dropout层，它在训练过程中随机关闭一部分神经元，以防止过拟合。

注意力机制实现（attention_net）

注意力机制的主要步骤为计算得分，应用softmax函数得到权重，然后进行加权求和。
首先，我们获取查询向量的最后一个维度的大小，存储在`d_k`中。然后，我们使用`torch.matmul`函数计算查询向量和输入数据的转置之间乘积。这个结果再除以`d_k`的平方根，得到的结果存储在`scores`中。这个步骤是为了缩放得分，防止在计算softmax时因为得分过大而导致的数值不稳定。

接下来，我们对`scores`应用softmax函数，得到的结果存储在`p_attn`中。softmax函数会将输入的每个元素缩放到0和1之间，并且所有元素的和为1。这样，`p_attn`就可以看作是输入数据中每个元素的权重。

最后，我们使用`torch.matmul`函数计算`p_attn`和输入数据`x`的矩阵乘法，然后对结果进行求和，得到的结果存储在`context`中。这个步骤是将输入数据的每个元素按照其权重进行加权求和，得到的结果就是注意力机制的输出。

前向传播（forward）

首先，输入数据`x`被传递到嵌入层`self.embedding`，并通过dropout层`self.dropout`进行正则化，结果存储在`embedding`中。这个步骤将输入的词汇索引转换为嵌入向量，并随机关闭一部分神经元以防止过拟合。然后，嵌入向量被传递到LSTM层`self.rnn`，得到输出`output`和最后的隐藏状态`final_hidden_state`以及单元状态`final_cell_state`。这个步骤是通过LSTM层处理嵌入向量，得到每个时间步的输出和最后的隐藏状态。接下来，我们对`output`通过dropout层，得到`query`。然后，我们将`output`和`query`传递到注意力网络`self.attention_net`，得到注意力输出`attn_output`和权重`attention`。这个步骤是通过注意力机制对LSTM的输出进行加权求和，得到每个时间步的注意力输出。最后，我们将注意力输出`attn_output`传递到全连接层`self.fc`，得到最终的分类结果`logit`。

2.2.3 参数设置与调整

网络结构的改进使得我们的模型loss能较快的收敛。所以我们仅仅针对几个模型建立和训练过程中的关键参数进行调整。

1. embedding dimension
表示学习（word embedding/network embedding等等）中，实现表示向量的降维是一个重要目的，所以表示维度应该低于词的数量或节点数量。维度的选取跟具体的数据集有关。一般训练数据越多或者网络越大，需要的维度越高。选取合适的维度，一般只需要采取已往工作的惯例，使得自己的方法和baseline保持一致即可。一般选取50、100或200。我们在此处分别选取50、100、200。我们将loss小于0.01时候的训练轮次数作为评价，得到的结果是embedding dimension为100的时候训练轮次最少，只需要3个epoch就可以达到要求。而对于embedding dimension = 50的时候需要6个epoch。

2. hidden dimension
隐藏层是将输入数据的特征抽象到另一个维度空间，以便进行线性划分和分类的重要组成。隐藏层维度会影响参数量的大小，也即模型的复杂程度。我们这里选取32、64、128、256进行测试。训练时间如下。同时当hidden dimension大于64的时候最终的loss已经不会出现明显的变化，所以我们综合考虑，选择hidden dim = 64。

```
hidden dim = 32, time = 2min 1s
hidden dim = 64, time = 3min 29s
hidden dim = 128, time = 7min 12s
hidden dim = 256, time = 10min 34s
```

3. epochs
训练轮次我们尝试20、30和40。结果见下面可视化展示的部分。可以看到相差不是很大，因为模型收敛的速度比较快，大约在10个epoch内其实就收敛了。不过dropout的存在有效防止了过拟合，所以我依然选择epoch = 30。

4. learning rate
控制收敛速度。分别尝试了1e-1，1e-2，1e-3，1e-4。最终在学习率为1e-3的情况下能达到最小的验证集loss。

最终参数列表如下：

```
    embedding_dim = 100  # 词向量维度
    hidden_dim = 64  # 隐藏层维度
    lr = 1e-3  # 学习率
    epoches = 30  # 训练轮数
```

2.3 可视化分析

2.3.1 网络结构可视化

我们利用torchviz进行网络结构的可视化

```
    out = model(text)
    g = make_dot(out, params=dict(model.named_parameters()),
                 show_attrs=True, show_saved=True)
    g.render('bi_lstm_attention', view=False)
```

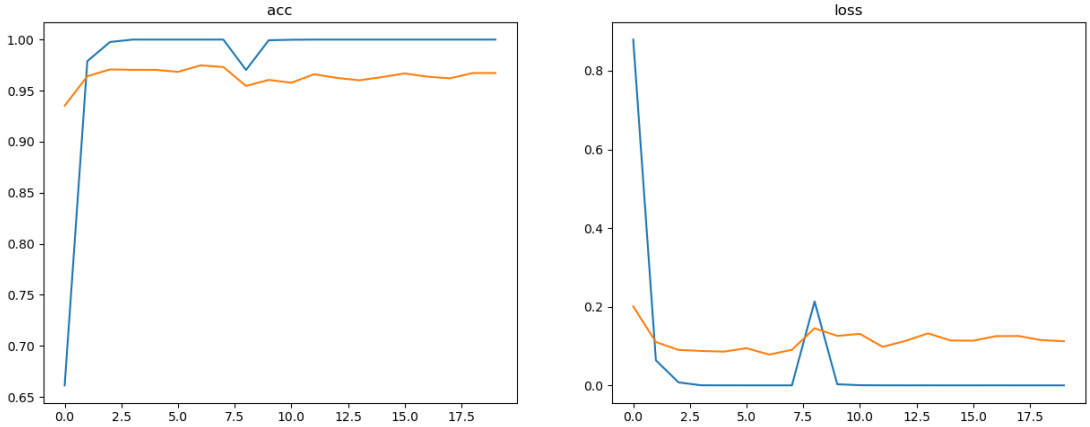
由于可视化之后的结果图片过大，在文档中不便进行展示，所以我上传到了图床。请点击以下链接查看。

网络结构展示
包含各层参数维度的网络结构展示

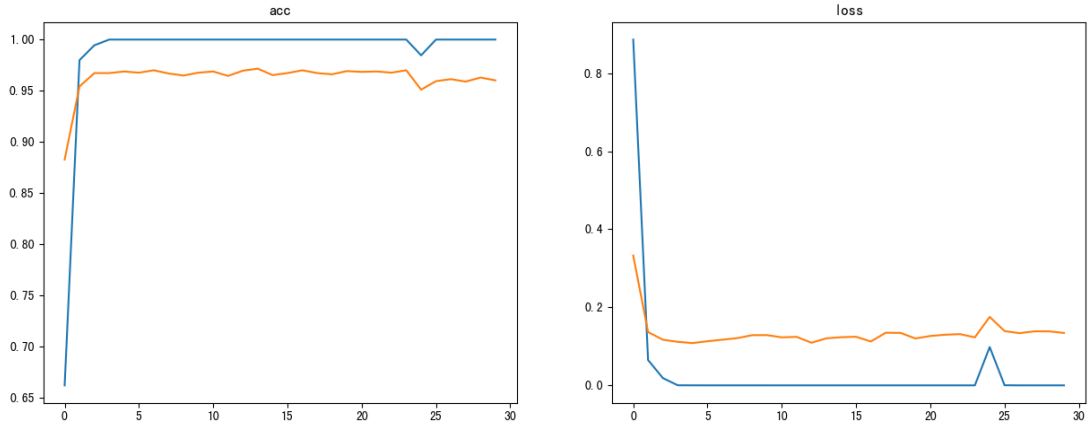
2.3.2 训练过程可视化

在调整训练有关参数的时候，我对于epoch进行了调整，下面是epoch取不同值的时候的训练曲线：

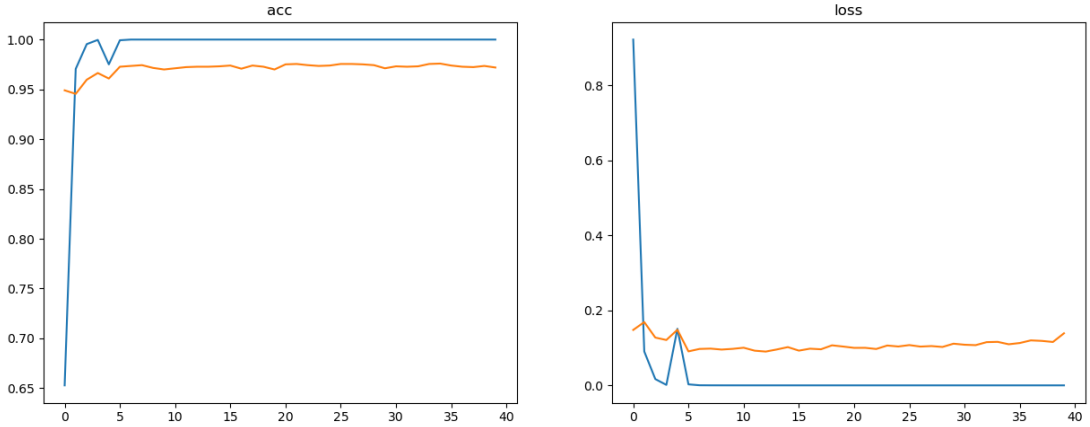
- epoch = 20



- epoch = 30



- epoch = 40



可以看到训练集准确率与验证集准确率均较快收敛。

最终有关的训练数据如下：

```
{"metric": "train_acc", "value": 1.0}
{"metric": "train_loss", "value": 5.210675976155141e-08}
{"metric": "val_acc", "value": 0.9719478467009087}
{"metric": "val_loss", "value": 0.13877506842742665}
```

可以看到，我们的模型在训练集上的准确度达到了100%，loss达到了1e-8级别。而在验证集上，我们的模型也有比较好的表现，准确度达到了97%，loss在0.14左右。通过分析学习曲线与最终的准确率结果，我们可以看到对于训练集上的表现，收敛速度较快并且最终指标较好。但是在验证集上我们的模型仍然存在一定的loss，仍然存在一定的改进空间。我尝试过改变学习率，但是效果并不是很好，我设置在一定iter后loss不发生明显变化则调整学习率，发现此类现象仍然存在（且偶然性较大）。我认为有赖于网络结构的改善。

3. 实验总结与体会

- 这是我第一次进行自然语言处理有关的深度学习任务。在本次实验中，我首次接触到了针对文本分析的库和模型，比如jieba、torchtext、GloVe等。由于所给的代码中没有这一部分，所以我首先学习了从文本到词向量的处理。
- 对于文本数据，我搜集了有关的论文资料，了解到了RNN是处理时序数据，包括语言在内的比较好的方法，所以我选取了其中一种基于LSTM改进的网络进行实现。并且这也是我第一次接触到注意力机制，了解了其中的原理并且做了简化的实现。、
- 实验中遇到的难题是，我基于GPU训练的模型在加载进行推断的时候显示机器不接受，只有CPU可用。然后我通过查询资料了解到保存模型和保存模型参数是不同的，通过修改代码仅将模型的参数进行保存，解决了该问题。
- 我通过本次实验了解了许多绘制神经网络结构图的方法，虽然有的不成功，但是对于科研绘图仍有很大的帮助。