利用 RNN 语言模型实现文本分类

姓名：陈泽豪　学号：SA22001009

2022 年 12 月 11 日

摘要

本报告首先介绍一下有关本次实验的大体实现框架，以及对 RNN 模型以及 LSTM 模型的介绍，然后给出简要的实验过程以及对超参数的实验分析，模型在测试集上的测试结果等。

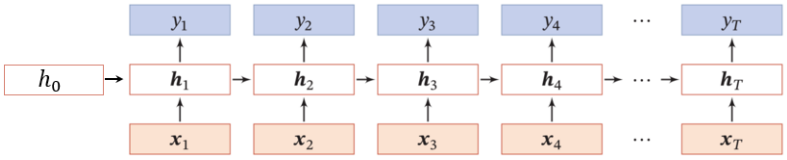
一、循环神经网络介绍

1.1 对循环神经网络的简要介绍

在上一次的实验中，我们进行了前馈神经网络的测试，并在课程中也介绍了仅仅考虑输入数据周边几个数据情况的卷积神经网络，与这两种神经网络有所不同，在循环神经网络中需要考虑到时间层这一维度的概念，将下一个时间层的输出与下一个时间层的输入以及这个时间层的输出进行关联，使得整个网络后续的输出均会考虑前一次输出的结果，由此将整个网络进行关联。循环神经网络通过使用带自反馈的神经元，能够处理任意长度的时序数据，如果以 h_t 表示 t 时刻的输出状态，以 x_t 表示 t 时刻的输入向量，以 f_W 表示参数变换矩阵为 W 的矩阵，则最终得到的计算形式如下所示：

$$\begin{aligned} h_t &= f_W(h_{t-1}, x_t) \\ &= f_W(f_W(h_{t-2}, x_{t-1}), x_t) \end{aligned} \tag{1}$$

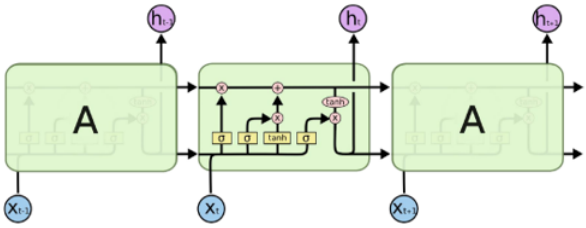
具体的循环神经网络流程可以表示为如下的形式：



同时根据循环神经网络的通用近似定理我们理论上可以将任意一个非线性系统表达为一个完全连接的循环神经网络。

1.2 对 LSTM 的简要讨论

之所以会产生长短期记忆神经网络，是因为在普通的循环神经网络中，在做梯度反向传播的过程中很容易出现梯度爆炸或者梯度消失的问题，导致学习只能只有短周期的依赖关系，即会产生长程依赖问题，因此 LSTM 增加了门控机制，使得新增的记忆单元可以在某个时刻捕捉到关键信息，并有能力保存一定的时间间隔。



在本身网络的基础上我们又可以向上堆砌整个网络的深度，网络是否为双向循环神经网络等参数，使得整个网络本身更加的复杂，以表示复杂的情形。

二、实验配置

本次实验通过 Anaconda 搭建出了虚拟环境，整个代码在 python 3.9.15，cuda 11.3.1，numpy 1.23.4，pytorch 1.12.1，torchtext 0.13.1 环境下进行实验，GPU 使用的是 NVIDIA Geforce RTX 3060。

三、对数据集的处理

下面简要介绍一下本次实验如何对数据集进行处理，这里主要的处理方式就是利用 torchtext 库内的函数去对整个数据集进行处理，主要代码参考了网络上的公开代码。

首先下载了 IMDB 公开数据集后，按照整个数据集的定义方式，其在 test 与 train 文件中分别存储了测试集与训练集数据，同时在这两个文件中又有 pos 与 neg 两个文件夹，这两个文件夹中存储着正面情绪的评论与负面情绪的评论。因此第一步写了一个函数 *read_acimbd* 去读取所有的相关句子，将句子用 torchtext 中的函数 *get_tokenizer* 进行划分，并以 0 表示这个句子为负面情绪，以 1 表示这个句子为正面情绪。具体代码展示如下：

```
def read_acimbd(is_train, path='./aclImdb_v1/aclImdb'):
review_list = []
label_list = []

# 定义一个标记器去分割英语单词
tokenizer = get_tokenizer('basic_english')

# 定义路径
if is_train:
valid_file = os.path.join(path, 'train')
else:
valid_file = os.path.join(path, 'test')

valid_file_pos = os.path.join(valid_file, 'pos')
valid_file_neg = os.path.join(valid_file, 'neg')

# 遍历所有的positive文件，用tokenizer进行分割
for filename in os.listdir(valid_file_pos):
with open(os.path.join(valid_file_pos, filename), 'r', encoding='utf-8') as file_content:
review_list.append(tokenizer(file_content.read()))
label_list.append(1)

# 遍历所有的negative文件
```

```
for filename in os.listdir(valid_file_neg):
with open(os.path.join(valid_file_neg, filename), 'r', encoding='utf-8') as file_content:
review_list.append(tokenizer(file_content.read()))
label_list.append(0)

return review_list, label_list
```

第二步利用拆分后的词语建立词汇与某个 $index$ 之间的对应关系，在经过了函数 $VocabTransform$ 后将所有的句子中的词语用它在词汇表里对应的 $index$ 代替，同时利用函数 Truncate，ToTensor ，PadTransform 将所有句子的长度做了统一处理，并将多填充的部分用词汇 $'<pad>'$ 的 $index$ 代替。将处理完之后的句子 $index$ 向量与句子本身的情感二分类 label 用函数 TensorDataset 进行打包，至此就完成了整个数据集的初步处理，具体代码展示如下：

```
# 将review_list与label_list经过vocab的index转化后用TensorDataset打包
def build_dataset(review_list, label_list, _vocab, max_len=512):
# 建立一个词表转化,vocab里存储词汇与它的唯一标签，利用VocabTransform将词汇转化为对应的数字
# 利用Truncate将所有的句子的最长长度限制在了max_len
# 利用ToTensor将所有的句子按照此时最长的句子进行填充为张量，填充的内容为'<pad>'对应的数字
# 利用PadTransform将ToTensor里所有的句子长度均填充为max_len
seq_to_tensor = T.Sequential(
T.VocabTransform(vocab=_vocab),
T.Truncate(max_seq_len=max_len),
T.ToTensor(padding_value=_vocab['<pad>']),
T.PadTransform(max_length=max_len, pad_value=_vocab['<pad>'])
)
dataset = TensorDataset(seq_to_tensor(review_list), torch.tensor(label_list))
return dataset
```

四、实验过程

在本次实验中我编写了两个网络，一个使用的就是普通的 RNN 网络，一个使用的是 LSTM 网络，因为在编写完 RNN 网络进行测试的时候效果并不尽人意，因此最终还是打算采用 LSTM 对结果进行测试。对于这两个网络，我都设置了参数 embedding 层的维数，隐藏层参数的维数，隐藏层的个数，是否采用双向 RNN 网络，是否使用预训练词向量，是否使用 Xavier 初始参数化的选项。为了之后讨论超参数的方便，这里我们就给出使用普通 RNN 以及使用 LSTM 的不同的结果，在抉择出最后使用的模型为 LSTM 之后便不再讨论普通 RNN 网络的超参数设置以及结果讨论。

下面的 RNN，LSTM 实验结果统一采用的参数为 $learning\_rate = 0.001$，$epoch\_num = 20$，$embed\_size = 300$，$hidden\_size = 512$，$num\_layers = 2$ 并采用词向量，使用 Xavier 初始化的结果。

4.1 普通 RNN 网络的结果

下面给出普通 RNN 网络在单向与双向的情况下得出来的 loss 随训练次数的变化情况，以及训练完之后的模型在测试集上测试后的准确率结果：

首先对于单向的网络，其训练 20 次后在测试集上的结果只能达到 50%，每个 batch 上的平均误差随训练次数增加的变化如下图 1 所示

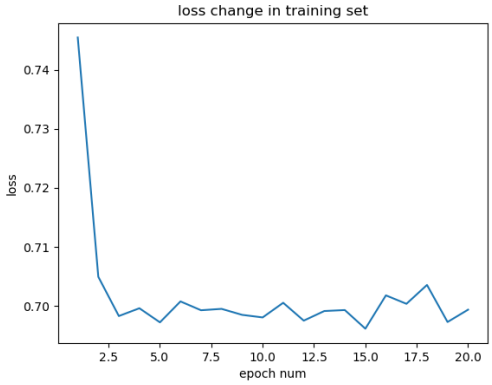


图 1: 单向 RNN 网络的 loss 随训练次数变化曲线

对于双向的 Rnn 网络，其训练 20 次后在测试集上的结果为 54.96%，变化结果如下图 2 所示：

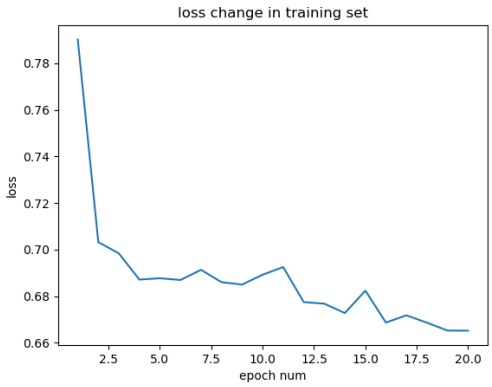


图 2: 双向 RNN 网络的 loss 随训练次数变化曲线

虽然这种情况下的双向无法降到符合我们预期的结果，但是如果不使用预训练词向量，同时也不使用 Xavier 初始化，其在每个训练周期内的误差比起前面还是有明显下降的，在测试集上结果为 62.85%，loss 下降的结果为下图 3：

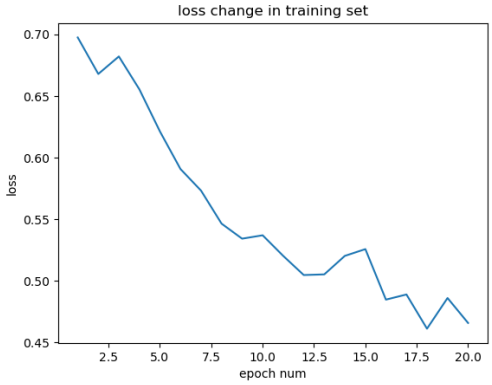
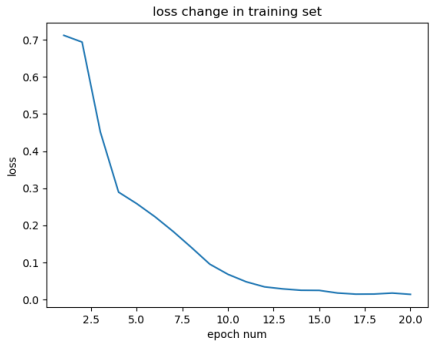


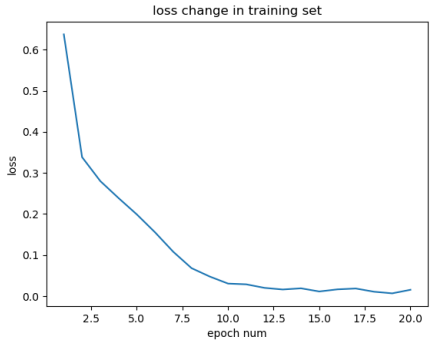
图 3: 不同条件下双向 RNN 网络的 loss 随训练次数变化曲线

4.2 LSTM 网络下的结果

对于单向的 LSTM 网络，我们设置 *batch_size* 为 128，否则会超过 cuda 可以容纳的内存，其在测试集上的准确率为 89.53%，其每个 batch 上的平均误差随训练次数增加的变化如下图所示：



对于双向的 LSTM 网络，由于 GPU 容量的限制，只能将 *batch_size* 设置为 64，其得到的测试集上的测试结果准确率为 89.34%，且平均误差随训练次数增加的变化如下图所示：



因此综合上面的考虑，可以得出结论就是普通的 RNN 网络确实在训练了足够多次数的情况下误差也可以收敛，但是它的收敛速度很慢，需要足够多的参数支持它网络的收敛，而 LSTM 则可以在较短的训练次数内达到较好的效果，因此下面我们选择采用单向的 LSTM 网络讨论它的超参数如何选取更加合适。

五、超参数测试

5.1 观察 Xavier 初始化与 Glove 词向量的使用带来的影响

在进行其他参数的测试前，我们首先验证 Xavier 初始化以及使用 Glove 词向量为结果带来的变化，训练次数依然为 20 次，$batch\_size$ 为 128，$learning\_rate = 0.001$，$epoch\_num = 20$，$embed\_size = 300$，$hidden\_size = 512$，$num\_layers = 2$，首先测试不使用 Glove 词向量，不使用 Xavier 初始化的结果，其在测试集上的测试结果准确率为 86.08%，其训练过程中 loss 变化图像如下：

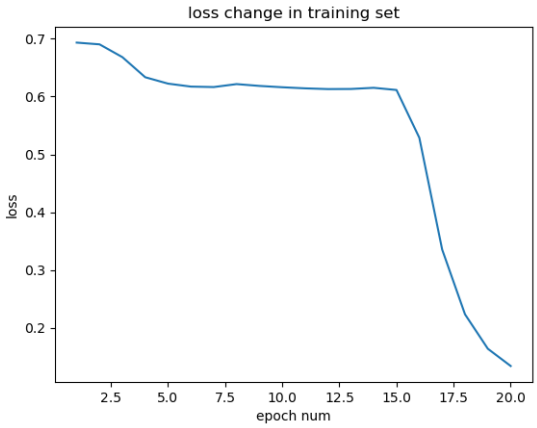


图 4: 不使用 Xavier，不使用 Glove

再给出使用了 Xavier 参数化但是不使用 Glove 词向量的结果，其在测试集上的准确率为 88.43%：

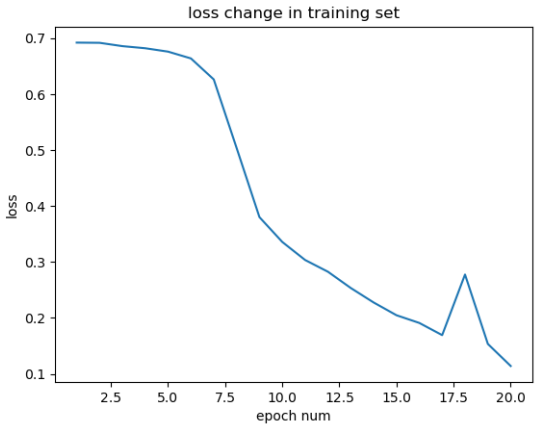


图 5: 使用 Xavier，不使用 Glove

根据这两种情况以及上面已经展示过的同时使用 Xavier 初始化与 Glove 词向量的情况可以发现当我们不使用 Xavier 初始化时，在经过多个 epoch 的训练之后也是可以将 loss 降下来的，但是如果使用了合适的初始化可以使我们的收敛更快的出现，而当我们使用了词向量之后，正确率确实有了小幅度的上升。

5.2 考虑学习率带来的变化

上面的实验过程中给出的学习率是 0.001，下面我们再考虑学习率为 0.01，0.0001 时的学习率如何，当学习率改为 0.01 时，整个模型误差无法收敛至 0，正确率只有 50%，误差曲线如下：

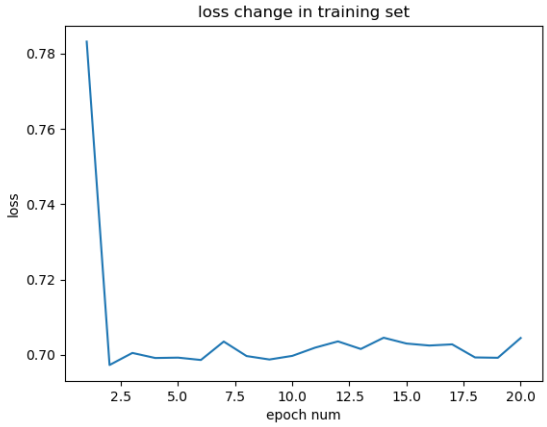


图 6: 学习率为 0.01 时

同理当学习率为 0.0001 时也可以有比较好的表现，在测试集上的准确率为 85.58%，变化曲线如下所示：

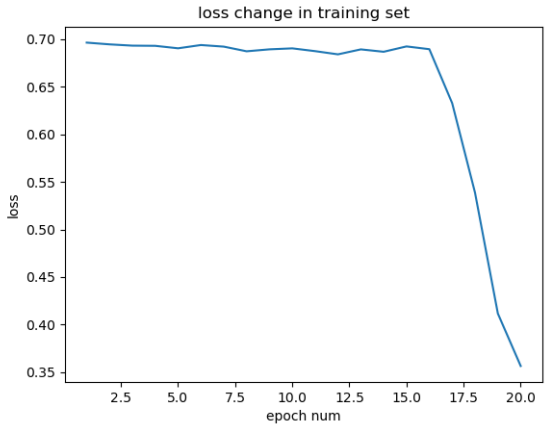


图 7: 学习率为 0.0001 时

但是和学习率为 0.001 相比，这种情况的收敛需要经历较长时间的训练才会出现下降的情况，因此这里我们还是选择学习率为 0.001 的情况。

5.3 考虑隐藏层的矩阵维数带来的变化

我们知道对于一个复杂的网络，如果它的变元较多，那么它可以表达的情况越复杂，因此这里不论是增加隐藏层的个数还是增加隐藏层的维数应该都会影响最终网络的表达，在上面我们用的是维数为 512 的隐藏层，下面我们简单展示一下维数为 256 的情形，它在测试集上的正确率为 88.51%：

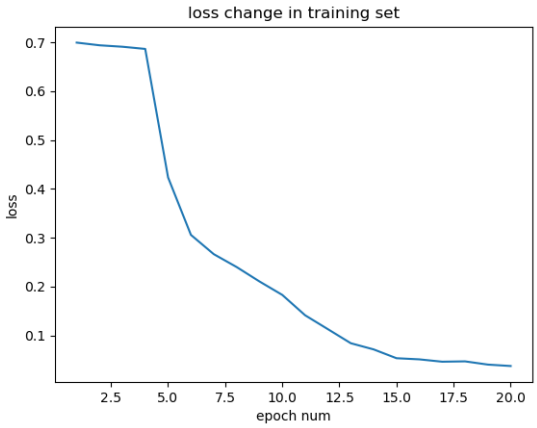


图 8: hidden size 为 256 时

下面再给出维数为 128 时的情形，其在测试集上的正确率为 88.38%，训练曲线如下：

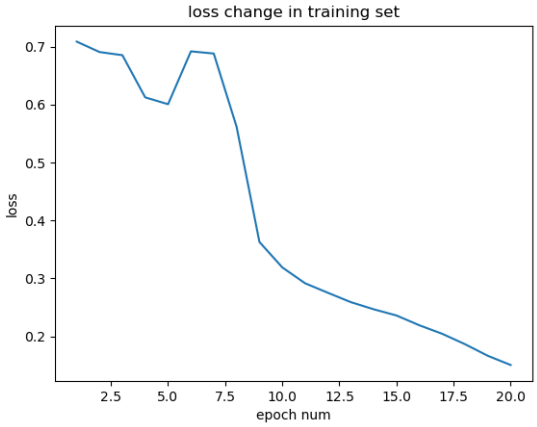


图 9: hidden size 为 128 时

可以发现随着隐藏层的维数的增加，确实是可以提高在测试集上的正确率的，但是提升幅度不是很大。下面我们就选择模型去做最终的训练。

5.4 最终测试结果以及对应的参数选择

最终我们选择的参数为 $learning\_rate = 0.001$,$epoch\_num = 30$,$embed\_size = 300$,$hidden\_size = 512$，$num\_layers = 2$，同时使用了 Xavier 初始化以及 Glove 词向量，最终得到的结果如图 10 所示，它在测试集上的测试结果精度为 89.40%，可以看到在 20 轮训练之后带给整个模型并没有太大的提升，如果需要进一步提高正确率可能需要继续改进模型的复杂度，利用别的深度学习模型进行进一步的优化。

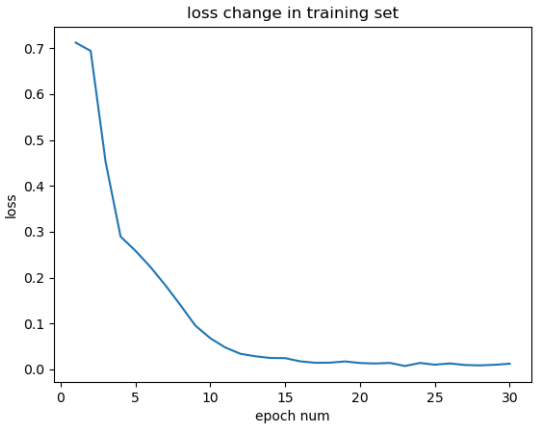


图 10: 最终的测试结果

六、实验结论

通过本次实验了解了如何利用 RNN 网络搭建出处理语言模块的情感分类代码，不仅了解了文本处理的方式，还了解了 RNN 网络的作用。