Challenging_Project 实验报告

——基于 LSTM 的虚假新闻检测系统的 Pytorch 实现

一，问题描述

给定一个信息的标题（必须）、出处、相关链接以及相关评论，尝试判别信息真伪。

输入：信息来源、标题、相关超链接、评论

输出：真伪标签（0：消息为真，1：消息为假）

二，数据集说明

数据来自 https://github.com/yaqingwang/WeFEND-AAAI20，于 2020-07-26 该数据被收集者公开(来自微信公众号的新闻)，本项目选择其中被标记真假的数据，包括训练集 train.news.csv 和测试机 test.news.csv。每条数据包含六个内容，分别为

[OfficialAccountName,Title,News Url,Image Url,Report_Content,label]

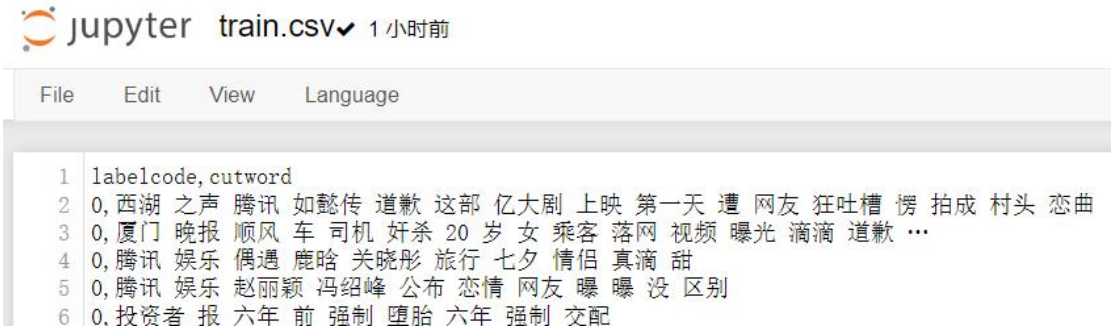
本项目只使用其中的[OfficialAccountName,Title,label]

三，方法介绍

本项目主要分为五个步骤：

1. 数据预处理

这一部分主要使用了基于 jieba 的分词方法，对训练集和测试集数据进行处理，最终得到的文本形式如图所示：



步骤为：

(1) 打开训练集文件进行读取，将同一个数据的 OfficialAccountName 和 Title 合并成为一个字符串

(2) 使用 jieba 进行分词，其中停用词来自

https://github.com/goto456/stopwords

本项目将四个停用词表整合为到了一个文件中 stopwords/stopwords.txt

1. 划分 train.news.csv 为训练集和验证集(划分比例为 4:1)，这一部分主要是为了根据神经网络模型在 val 上的表现，来对参数进行修正(在实际使用中，不进行划分，使用 train.news.csv 的全部数据)
2. 将得到的数据写入新的 csv 表格，并去除空行
3. 正式实验时，不划分划分训练集和验证集，将 train.news.csv 用作训练集，

test.news.csv 用作测试集，写入 csv 表格并去除空行

2. 文本向量化

这一部分，主要使用了 torchtext 来对预处理好的文本进行向量化

import torch.legacy.data as data

（1）使用 data.Field 定义文本(TEXT)和标签(LABEL)的切分方式

（2）使用 data.TabularDataset.splits 划分数据集

（3）使用 data.build_vocab 来构建词典，同时实现向量化

（4）使用 data.BucketIterator 以 BATCH_SIZE 为单位打包方便后续送入神经网络，同时返回一个只有数据集的迭代器（BucketIterator 会打乱数据的顺序，便于神经网络的训练）

3. 构造 LSTM 神经网络

这一部分，主要是构建 LSTMNet 神经网络

from torch import nn

（1）使用 nn.Embedding 进行词向量的处理，将高维的词向量映射到低维向量上(20 维)

（2）使用 nn.LSTM 定义循环神经网络的第一个隐藏层，输入为 20 个神经元，输出为 128 个神经元

（3）使用 nn.LSTM 定义循环神经网络的第二个隐藏层，输入为 128 个神经元，输出为 64 个神经元

（4）使用 nn.Linear 定义循环神经网络的全连接层，输入为 64 个神经元，输出为 2 个神经元

（5）定义前向传播函数

4. 在训练集上训练

定义了 train_LSTM 函数用于神经网络的训练

通过 model.train 设置网络为训练模式

(1) 正向传播算出 hypothesis

(2) 计算 loss，清空梯度

(3) 反向传播计算当前梯度

(4) 根据梯度值更新当前网络参数

(5) 重复上述过程共 epoch 次

5. 在测试集上测试并评价

通过 model.eval 设置网络为评估模式

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,roc_curve,auc,confusion_matrix

计算出 Accuracy，Precision，Recall，F1，AUC 等值，画出 ROC 曲线，计算混淆矩阵同时使用热力图进行可视化，根据数据评估模型

四，关键代码细节、运行截图

详细过程解释，请参见.ipynb 文件中的注释，下面仅进行粗略说明

1. 数据预处理

下列代码实现了分词操作，同时去除了停用词，分词后的数据存储在 data3 中

```
data3 = []
stwf = open(r'stopwords\stopwords.txt', 'r', encoding="utf-8")
```

```
stopwords = [line.strip() for line in stwf.readlines()]
for i in range(rows):
    seg_list = list(jieba.cut(data2[i], cut_all=False))
    tmplist = []
    for j in seg_list:
        if j not in stopwords:
            tmplist.append(j)
    result = ' '.join(tmplist)
    data3.append(result)
```

```
## 查看分词后的数据
print(data3[0])
```

```
环球 人物 中国 反腐 风刮到 阿根廷 美到 瘫痪 女 总统 本子 摊上 大事
```

2. 文本向量化

(1) Field 定义文本切分方式

```
mytokenize = lambda x: x.split()
TEXT = data.Field(sequential=True, tokenize=mytokenize,
                  include_lengths=True, use_vocab=True,
                  batch_first=True, fix_length=20)
LABEL = data.Field(sequential=False, use_vocab=False,
                   pad_token=None, unk_token=None)
```

(2) TabularDataset.splits 划分数据集

```
text_data_fields = [
    ("labelcode", LABEL), # 对标签的操作
    ("cutword", TEXT) # 对文本的操作
]
traindata,testdata = data.TabularDataset.splits(
    path="data-derived", format="csv",
    train="train_.csv", fields=text_data_fields,
    test = "test.csv", skip_header=True
)
```

```
## 检查一个样本的标签和文本
em = traindata.examples[0]
print(em.labelcode)
print(em.cutword)
```

```
0
['环球', '人物', '中国', '反腐', '风刮到', '阿根廷', '美到', '瘫痪', '女', '总统', '本子', '摊上', '大事']
```

(3) build_vocab 创建词典

```
TEXT.build_vocab(traindata,vectors = None)
LABEL.build_vocab(traindata)
```

```
print("词典的词数:",len(TEXT.vocab.itos))
print("前10个单词:\n",TEXT.vocab.itos[0:10])
print("词典:\n",TEXT.vocab.stoi)
## 类别标签的数量和类别
print("训练集类别标签情况:",LABEL.vocab.freqs)

## 前两个分别代表词典中没有的词语和用于填充的词语
```

```
词典的词数: 19752
前10个单词:
 ['<unk>', '<pad>', '岁', '…', '!', '中国', '网友', '离婚', '年', '曝光']
词典:
 defaultdict(<bound method Vocab._default_unk_index of <torchtext.legacy.vocab.Vocab object at 0x0000021223AFFEE0>>, {'<unk>': 0, '<pad>': 1, '岁': 2, '…': 3, '!': 4, '中国': 5, '网友': 6, '离婚': 7, '年': 8, '曝光': 9, '事件': 10, '娱乐圈': 11, '终于': 12, '震惊': 1
```

(4) BucketIterator 打包数据同时返回数据的一个迭代器

```
BATCH_SIZE = 64
train_iter = data.BucketIterator(traindata,batch_size = BATCH_SIZE)
test_iter = data.BucketIterator(testdata,batch_size = BATCH_SIZE)
```

```
## 针对一个batch的数据，可以使用batch.labelcode获得数据的类别标签
print("数据的类别标签:\n",batch.labelcode)
## batch.cutword[0]是文本对应的内容矩阵
print("文本数据的内容:",batch.cutword[0])
```

```
数据的类别标签:
 tensor([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        1, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1])
文本数据的内容: tensor([[  700,   225,  1002,  ...,     1,     1,     1],
        [ 2043,  6499,  6100,  ...,     1,     1,     1],
        [ 7897,   159,    24,  ...,     1,     1,     1],
        ...,
        [  123,  1219,  3016,  ...,     1,     1,     1],
        [  660,  1492,     9,  ...,     1,     1,     1],
        [  418, 13134,    26,  ...,     1,     1,     1]])
```

3. 构造 LSTM 神经网络

以下代码删除了部分注释，详情请在.ipynb 中查看

```
class LSTMNet(nn.Module):

    def __init__(self, vocab_size):
        """
        vocab_size:词典长度
        """
        super(LSTMNet, self).__init__()
    ## 对文本进行词向量处理，每个词使用 20 维的向量表示
        self.embedding = nn.Embedding(vocab_size, 20)
    ## 1 层 128 个神经元的 LSTM 层
        self.lstm1 = nn.LSTM(20, 128, 1,batch_first=True)
    ## 一层 64 个神经元的 LSTM 层
        self.lstm = nn.LSTM(128, 64, 2,batch_first=True)
    ## 全连接层(Full Connection Layer)的输入的神经元个数为 64，输出神经元个数为 2
        self.fc1 = nn.Linear(64, 2)

    def forward(self, x):
```

```
        embeds = self.embedding(x)
        r1_out, (h1_n, c1_n) = self.lstm1(embeds, None)
        r2_out, (h2_n, c1_n) = self.lstm2(r1_out, None)
        hypothesis = self.fc1(r2_out[:, -1, :])
        return hypothesis
```

4. 在训练集上训练

以下代码定义了训练过程函数 train_LSTM

```
def train_LSTM(model,traindataloader,cost_function,
               optimizer,num_epochs=25,):
    """
    model:网络模型;
    traindataloader:训练数据集
    valdataloader:验证数据集
    cost_function：损失函数
    optimizer：优化方法;
    num_epochs:训练的轮数
    """
    train_loss_all = []
    train_acc_all = []
    val_loss_all = []
    val_acc_all = []
    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        # 每个 epoch 有两个阶段,训练阶段和验证阶段
        train_loss = 0.0
        train_corrects = 0
        train_num = 0
        val_loss = 0.0
        val_corrects = 0
        val_num = 0
        model.train() ## 设置模型为训练模式
        for step,batch in enumerate(traindataloader):
            textdata,target = batch.cutword[0],batch.labelcode.view(-1)
            hypothesis = model(textdata)# 正向传播算出 H 值
            pre_lab = torch.argmax(hypothesis,1) # 预测的标签
            loss = cost_function(hypothesis, target) # 计算损失函数值
            optimizer.zero_grad()# 清空梯度值
            loss.backward()# 反向传播，计算当前梯度
            optimizer.step()# 根据梯度更新网络参数
            train_loss += loss.item() * len(target)
            train_corrects += torch.sum(pre_lab == target.data)
            train_num += len(target)
        ## 计算一个 epoch 在训练集上的损失和精度
```

```
            train_loss_all.append(train_loss / train_num)
            train_acc_all.append(train_corrects.double().item()/train_num)
            print('{} Train Loss: {:.4f}  Train Acc: {:.4f}'.format(
                epoch, train_loss_all[-1], train_acc_all[-1]))
                    ## 计算一个 epoch 的训练后在验证集上的损失和精度
        train_process = pd.DataFrame(
            data={"epoch":range(num_epochs),
                  "train_loss_all":train_loss_all,
                  "train_acc_all":train_acc_all})
        return model,train_process
```

5. 在测试集上测试并评价

对测试集进行预测并计算各项指标的代码如下：

```
lstmmodel.eval() ## 设置模型为训练模式评估模式
test_y_all = torch.LongTensor()
pre_lab_all = torch.LongTensor()
for step,batch in enumerate(test_iter):
    textdata,target = batch.cutword[0],batch.labelcode.view(-1)
    hypothesis = lstmmodel(textdata)
    pre_lab = torch.argmax(hypothesis,1)
    test_y_all = torch.cat((test_y_all,target)) ##测试集的标签
    pre_lab_all = torch.cat((pre_lab_all,pre_lab))##测试集的预测标签

## 将真 0 假 1 转变为真 1 假 0，以便于调用函数
y_test = []
y_pre = []
for i in test_y_all.detach().numpy():
    if i == 1:
        y_test.append(0)
    else:
        y_test.append(1)
for i in pre_lab_all.detach().numpy():
    if i == 1:
        y_pre.append(0)
    else:
        y_pre.append(1)


# Accuracy
# Precision
# Recall
# F1
acc = accuracy_score(y_test,y_pre)
pre = precision_score(y_test,y_pre)
```

```
    rec = recall_score(y_test,y_pre)
    f1  = f1_score(y_test,y_pre)
    print("Accuracy:",acc)
    print("Precision:",pre)
    print("Recall:",rec)
    print("F1:",f1)



    # ROC 曲线
    # AUC
    ## 绘制 ROC 曲线并且计算 AUV 值
    fpr, tpr, thresholds = roc_curve(y_test,y_pre)
    roc_auc = auc(fpr,tpr)
    print("AUC:",roc_auc)
    plt.plot(fpr,tpr,'*-')
    plt.ylabel('TPR')
    plt.xlabel('FPR')
    plt.title('ROC curve')

    ## 计算混淆矩阵并可视化
    conf_mat                                                                    =
confusion_matrix(test_y_all.detach().numpy(),pre_lab_all.detach().numpy())
    plt.figure(figsize = (2,2))
    heatmap = sns.heatmap(conf_mat, annot=True, fmt="d",cmap="YlGnBu")
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()
```

五，实验参数的选择

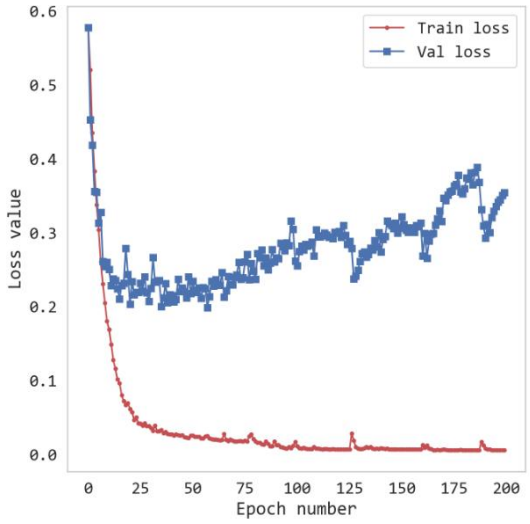
1. 训练过程展示（train.news.csv 划分成的训练集和验证集的比例为 4/1）

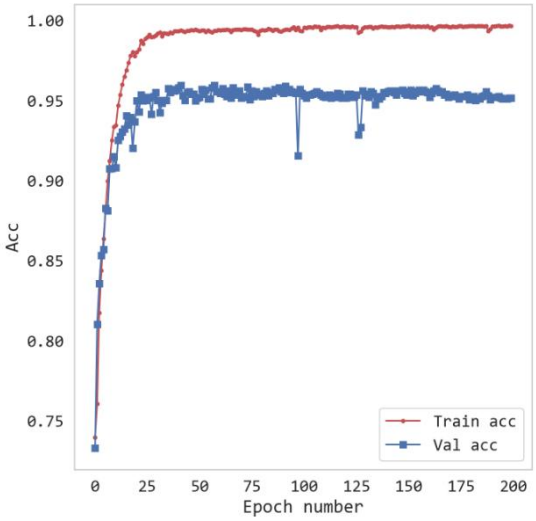
   经过多次对参数的不断调整，最终我的网络选择以下参数：

   双隐藏层(dropout=0.2),epoch= 80,lr=0.0003,使用交叉熵损失函数,使用 Adam 优化器,L2 正则化参数 weight_decay=0.0003,词向量长度为 20

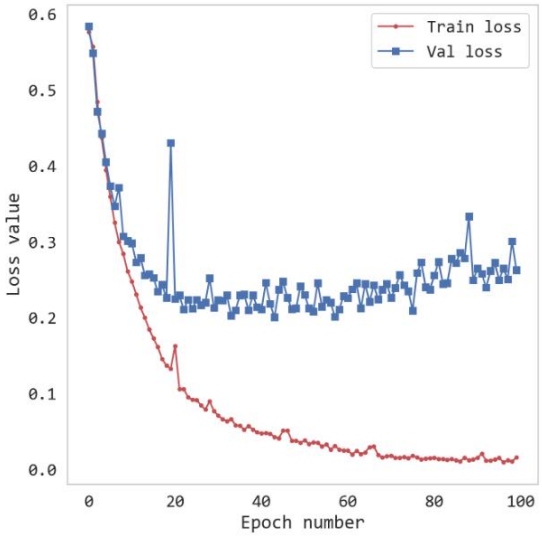
训练过程可视化曲线(红色为 train.csv 的 loss 和 acc 值,蓝色为 val.csv 的 loss 和 acc 值)

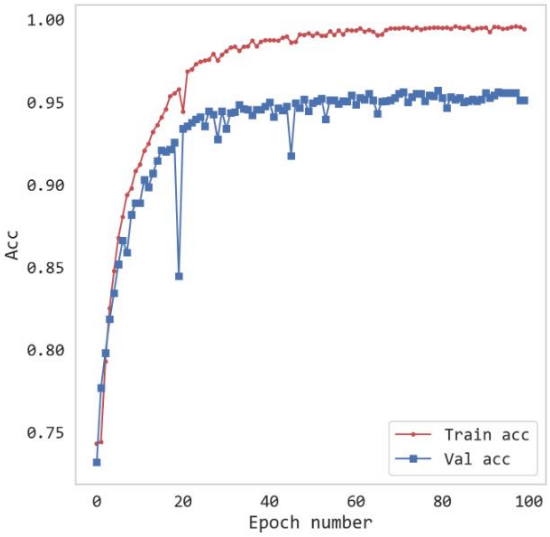
当 epoch=200 时:





当 epoch=100 时,

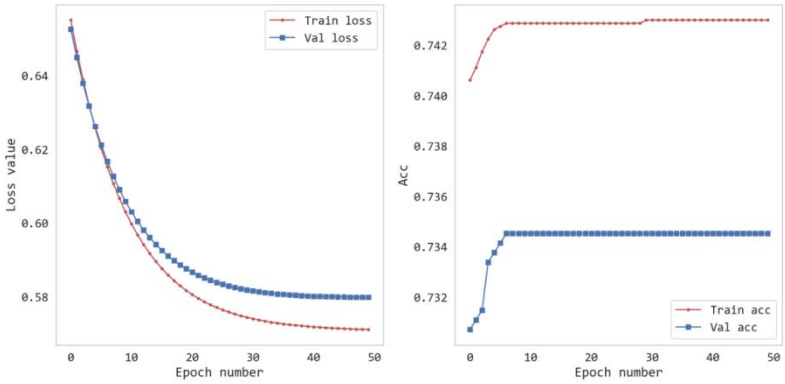




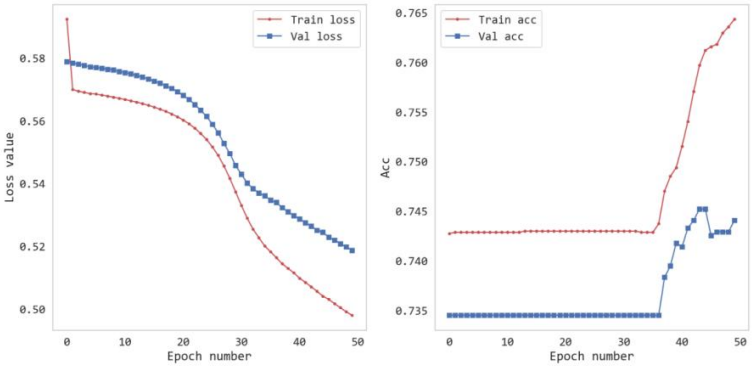
2. 四个优化函数对比曲线

损失函数选择交叉熵,优化函数不使用正则化,
lr=0.0003,epochs=50,train/val = 3/1

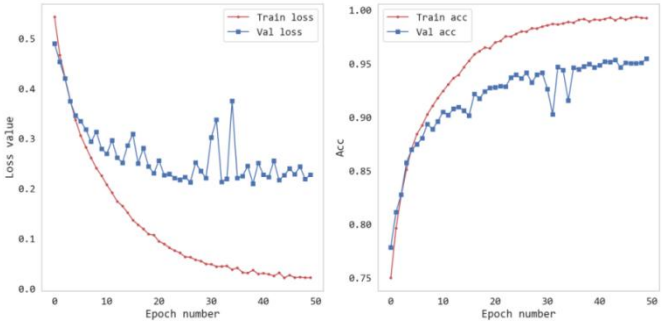
(1) SGD



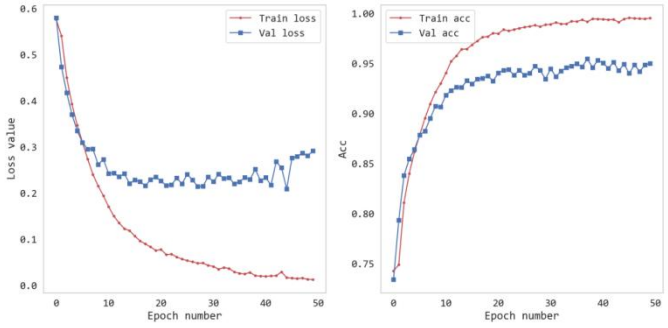
（2）Adagrad



（3）Rmsprop

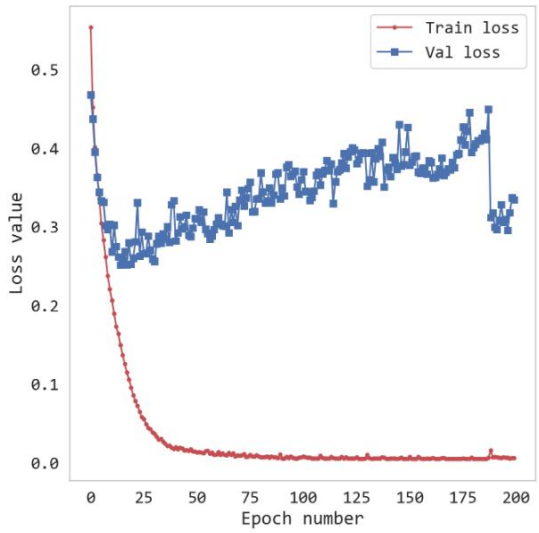


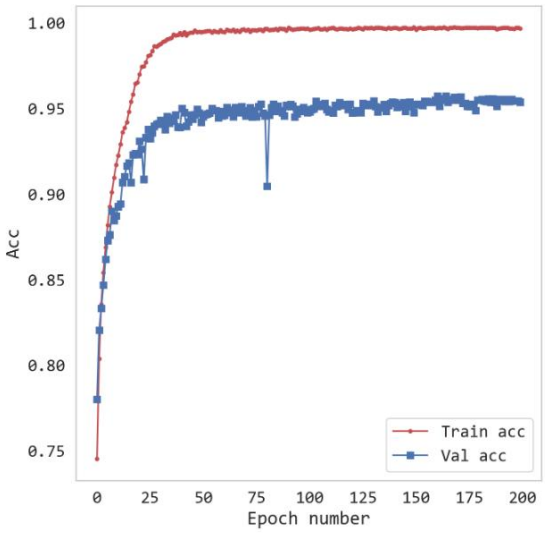
（4）Adam



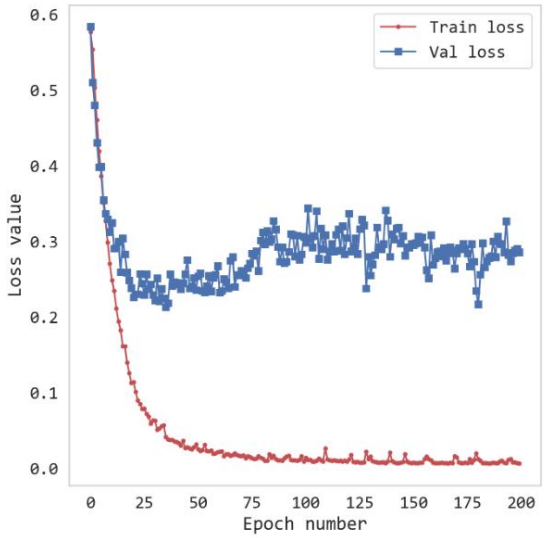
进一步对比 RMSprop 和 Adam:
损失函数选择交叉熵，不进行正则化，lr=0.0003,epochs=200,train/val = 4/1

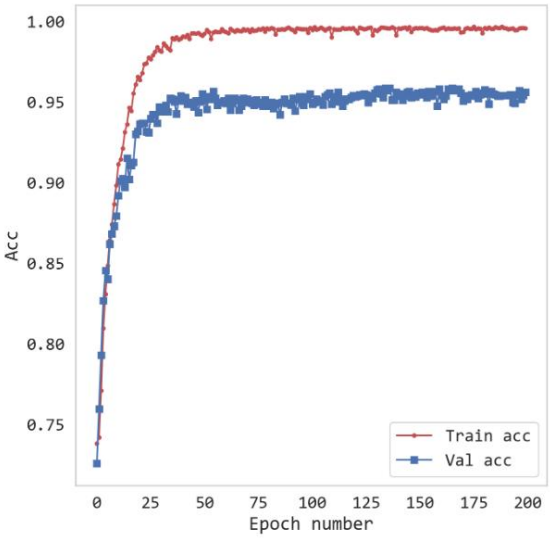
（1）Rmsprop





（2）Adam





两者的曲线接近，而且在测试集上的各项指标也十分接近，综合考虑，最终还是选择 Adam 优化函数。

六，实验结果及分析评价

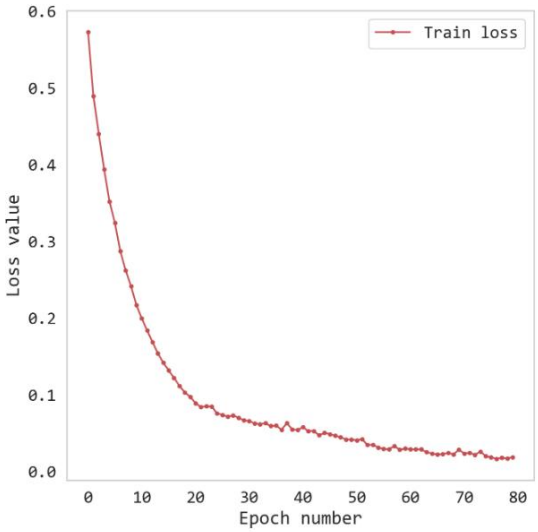
1. 实验结果

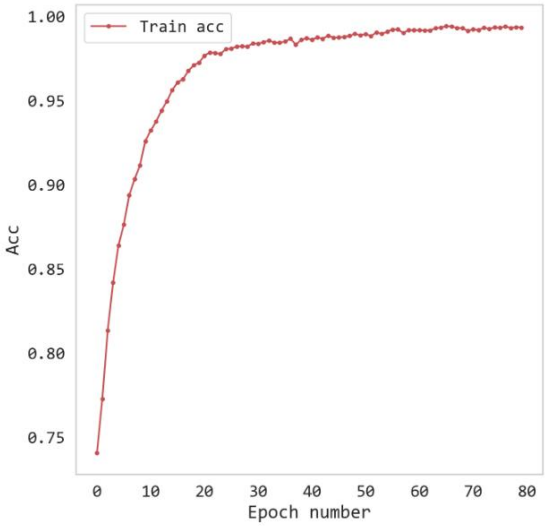
（1）训练过程中的 loss 和 acc 值及曲线图

双隐藏层(dropout=0.2),epoch= 80,lr=0.0003,使用交叉熵损失函数,使用 Adam 优化器,L2 正则化参数 weight_decay=0.0003,词向量长度为 20

```
## 定义优化器
optimizer = torch.optim.Adam(lstmmodel.parameters(), lr=0.0003, weight_decay=0.0003)
loss_func = nn.CrossEntropyLoss()    # 定义损失函数为交叉熵
## 对模型进行迭代训练,对所有的数据训练100个epoch
lstmmodel,train_process = train_LSTM(
    lstmmodel,train_iter,loss_func,optimizer,num_epochs=80)
```

```
70 Train Loss: 0.0235  Train Acc: 0.9924
Epoch 71/79
71 Train Loss: 0.0244  Train Acc: 0.9918
Epoch 72/79
72 Train Loss: 0.0218  Train Acc: 0.9934
Epoch 73/79
73 Train Loss: 0.0261  Train Acc: 0.9925
Epoch 74/79
74 Train Loss: 0.0199  Train Acc: 0.9935
Epoch 75/79
75 Train Loss: 0.0185  Train Acc: 0.9932
Epoch 76/79
76 Train Loss: 0.0167  Train Acc: 0.9940
Epoch 77/79
77 Train Loss: 0.0179  Train Acc: 0.9931
Epoch 78/79
78 Train Loss: 0.0172  Train Acc: 0.9937
Epoch 79/79
79 Train Loss: 0.0185  Train Acc: 0.9934
```

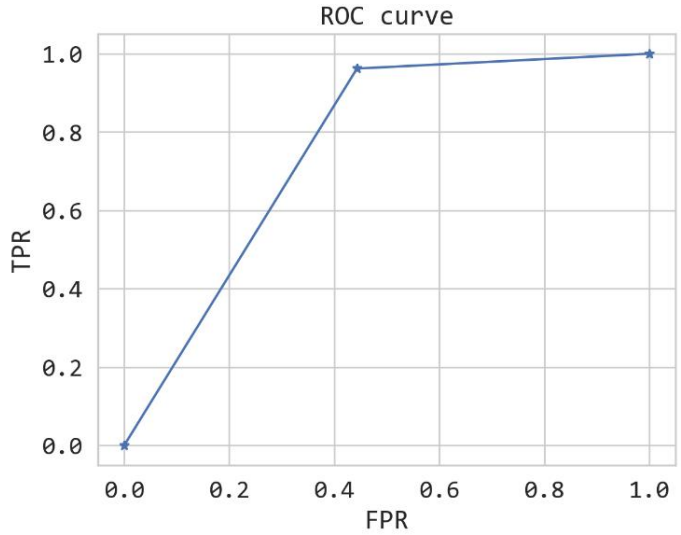




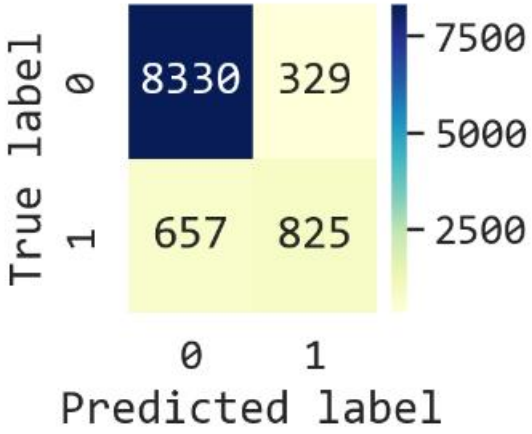
（2）各项评价指标

Accuracy: 0.9027709298885711
Precision: 0.926894403026594
Recall: 0.9620048504446241
F1: 0.9441233140655106
AUC: 0.759342506193972

ROC 曲线图：



混淆矩阵的热力图表示：



2. 模型评价

（1）该模型的 Accuracy,Precision,Recall,F1,AUC 值均较高，说明模型较为可靠；

（2）ROC 曲线偏向左上方，可知分类器的分类性能较好，可以很好的对新闻进行分类；

（3）由 Precision=0.926894403026594 略小于 Recall= 0.9620048504446241，可知，该分类器相对而言更容易把假新闻识别为真新闻。

（4）模型也相对的存在一些缺点，比如存在一定的过拟合，我通过 L2 正则化方式及增加隐藏层并增设 dropout 值,才稍微缓解了模型的过拟合，未来可以选择更多方式，来对模型进行改进。