## Challenging\_Project实验报告

##### ——基于LSTM的虚假新闻检测系统的Pytorch实现

1. **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）

1. **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]

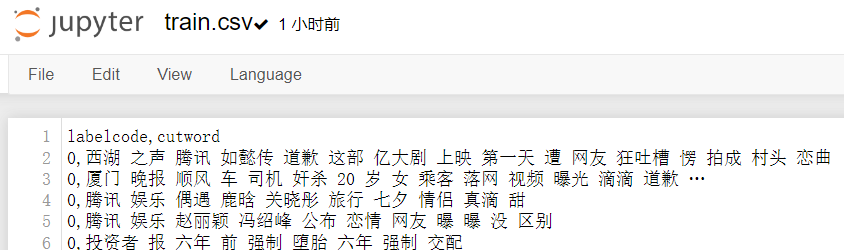
本项目只使用其中的[OfficialAccountName,Title,label]

1. **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
4. 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）
5. 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
6. 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
6. 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

1. **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 = []

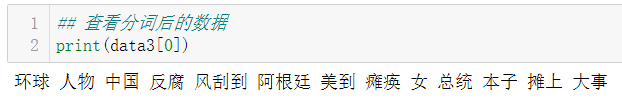
for j in seg\_list:

if j not in stopwords:

tmplist.append(j)

result = ' '.join(tmplist)

data3.append(result)



1. 文本向量化
2. 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)

1. 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

]

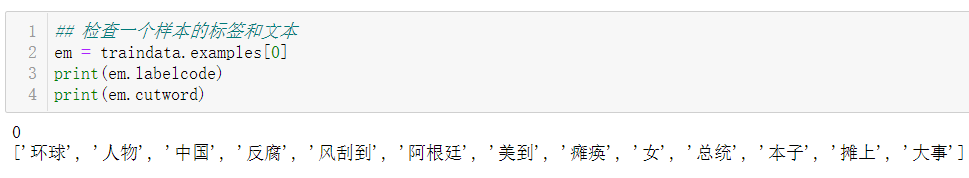
traindata,testdata = data.TabularDataset.splits(

path="data-derived", format="csv",

train="train\_.csv", fields=text\_data\_fields,

test = "test.csv", skip\_header=True

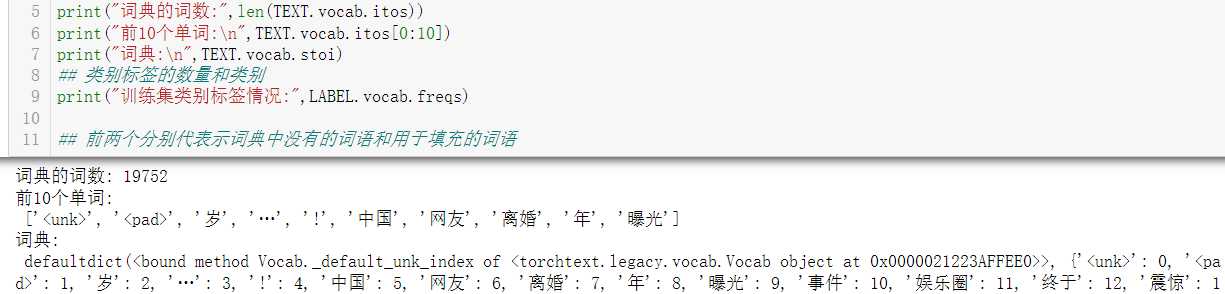
)



1. build\_vocab创建词典

TEXT.build\_vocab(traindata,vectors = None)

LABEL.build\_vocab(traindata)

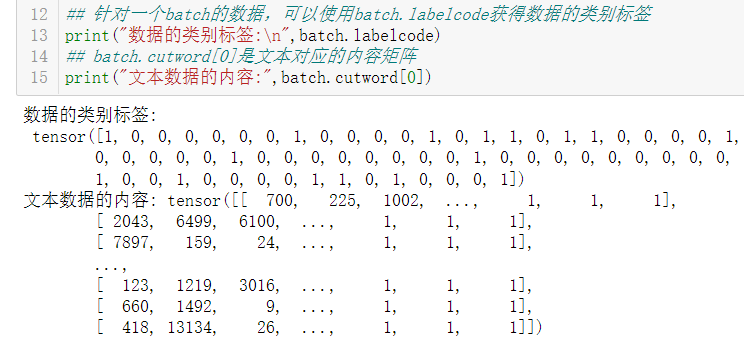


1. BucketIterator打包数据同时返回数据的一个迭代器

BATCH\_SIZE = 64

train\_iter = data.BucketIterator(traindata,batch\_size = BATCH\_SIZE)

test\_iter = data.BucketIterator(testdata,batch\_size = BATCH\_SIZE)



1. 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

1. 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

1. 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()

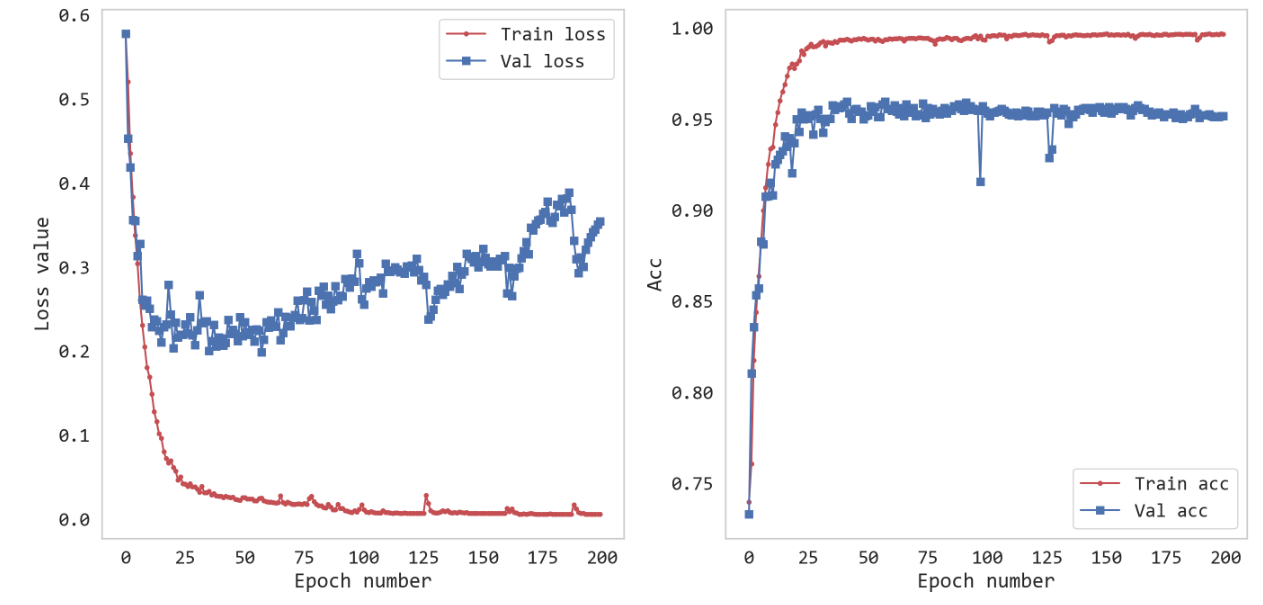
1. **实验参数的选择**
2. 训练过程展示（train.news.csv划分成的训练集和验证集的比例为4/1）

经过多次对参数的不断调整，最终我的网络选择以下参数：

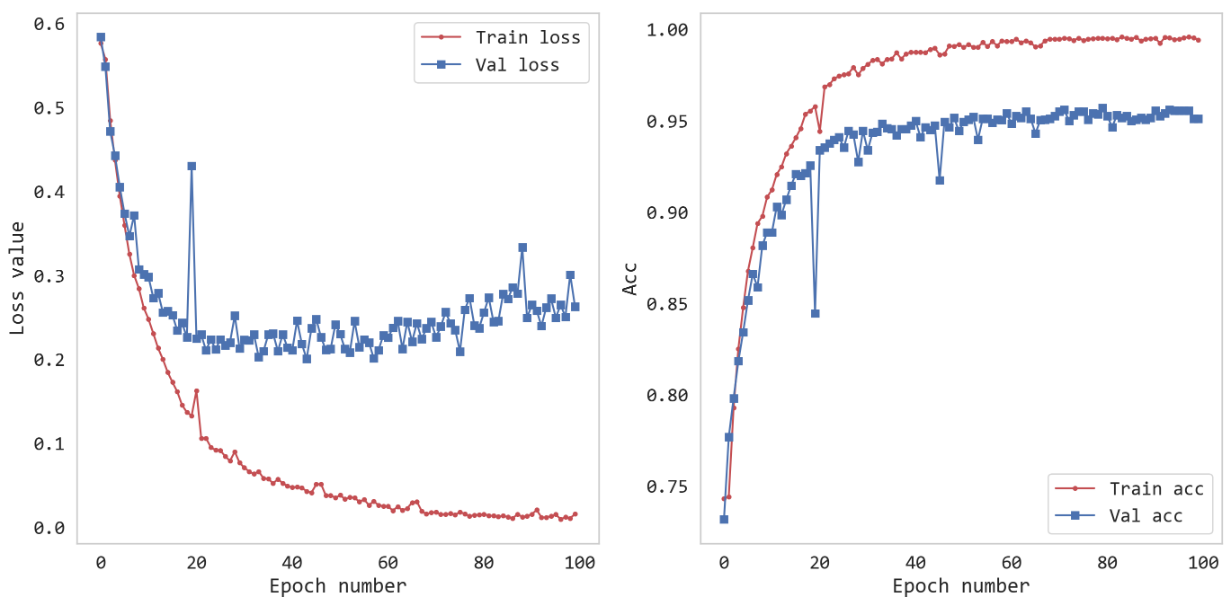
双隐藏层(dropout=0.2),epoch= 80,lr=0.0003,使用交叉熵损失函数,使用Adam优化器,L2正则化参数weight\_decay=0.0003,词向量长度为20

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：



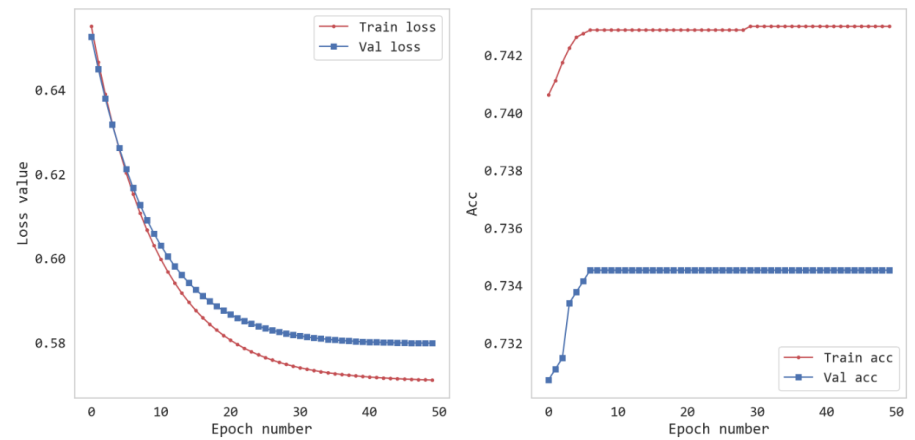
当epoch=100时，



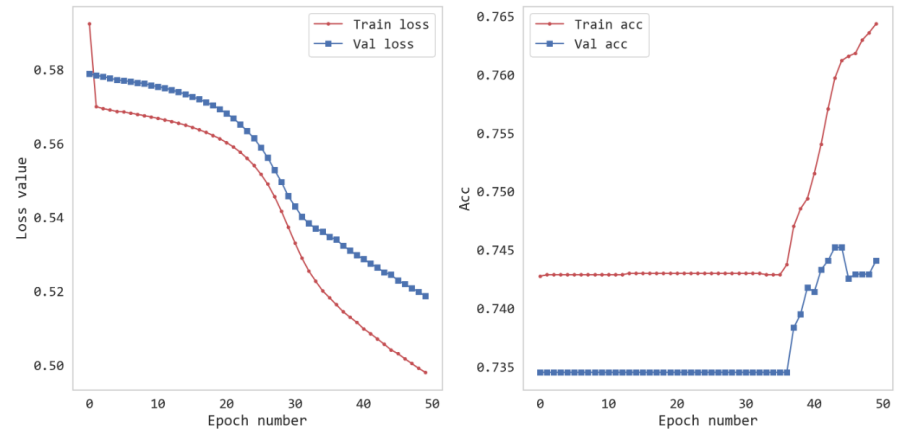
1. 四个优化函数对比曲线

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

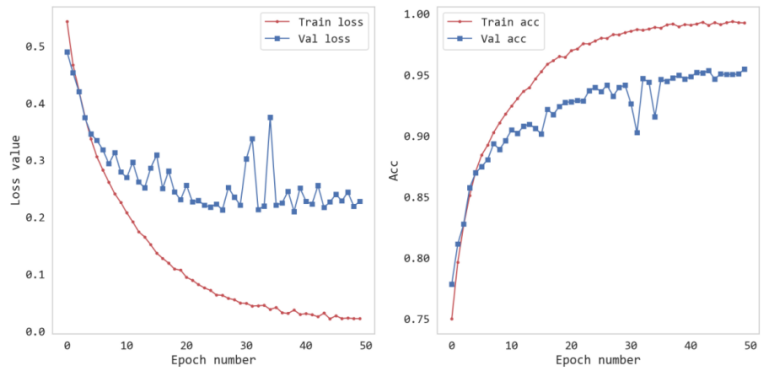
1. SGD



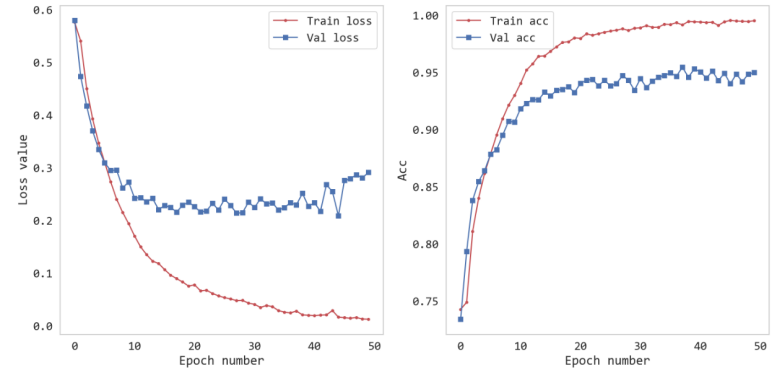
1. Adagrad



1. Rmsprop



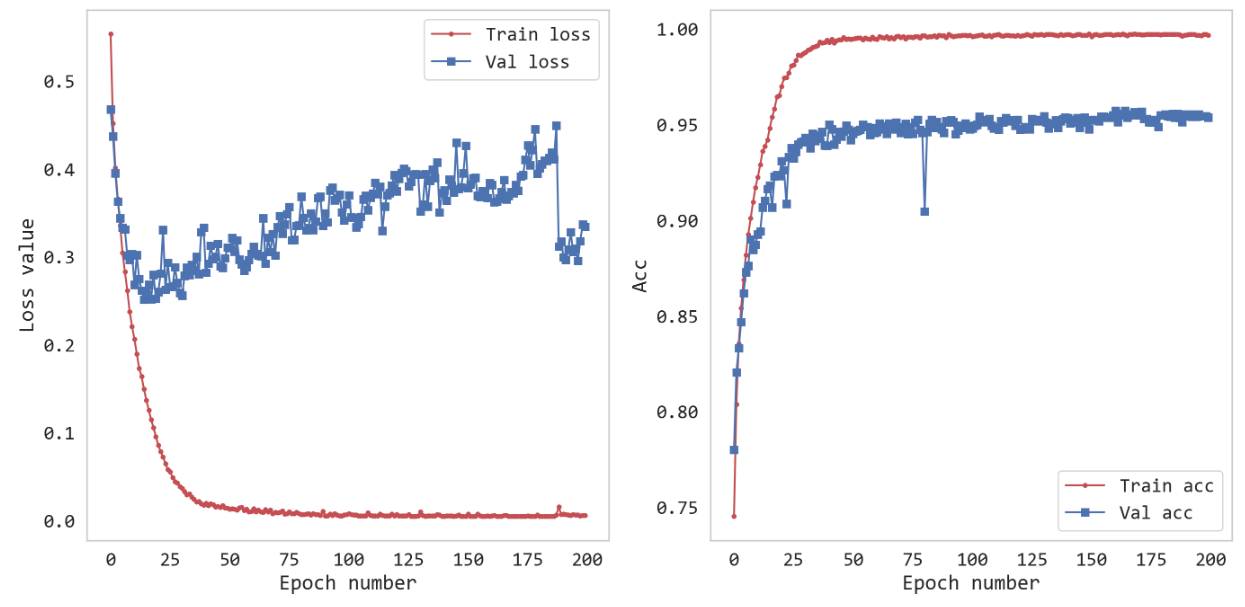
1. Adam



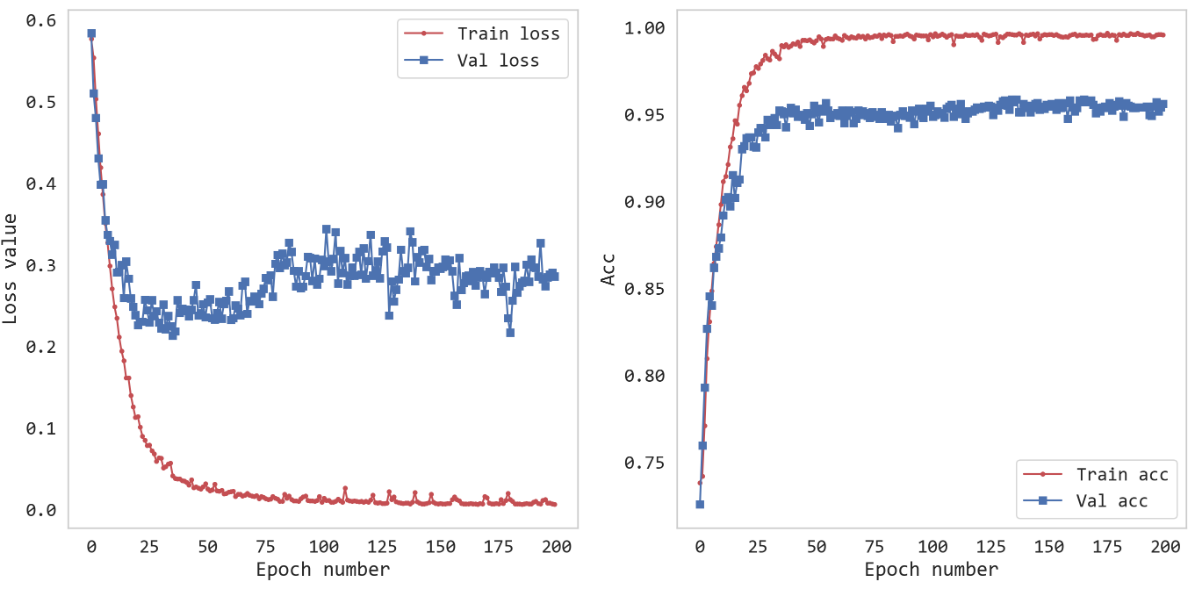
进一步对比RMSprop和Adam:

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



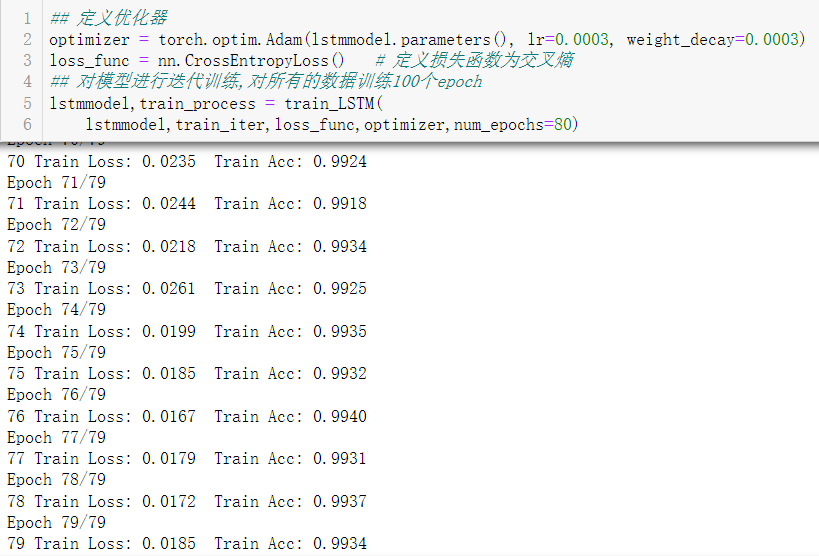
1. Adam

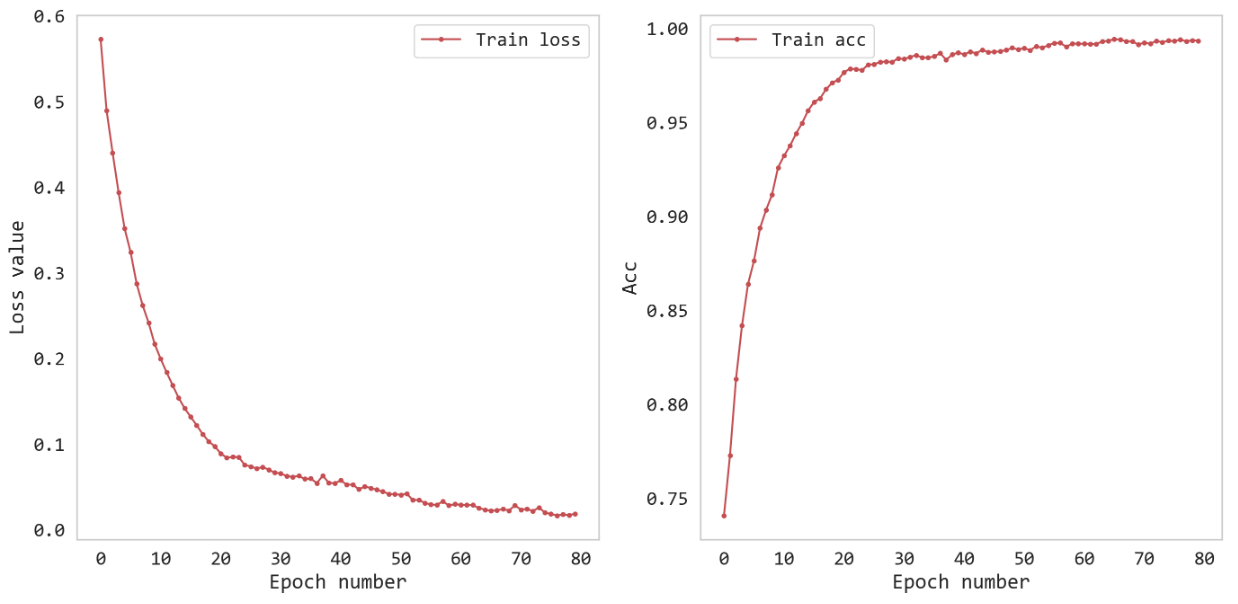


两者的曲线接近，而且在测试集上的各项指标也十分接近，综合考虑，最终还是选择Adam优化函数。

1. **实验结果及分析评价**
2. 实验结果
3. 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*





1. 各项评价指标

Accuracy: 0.9027709298885711

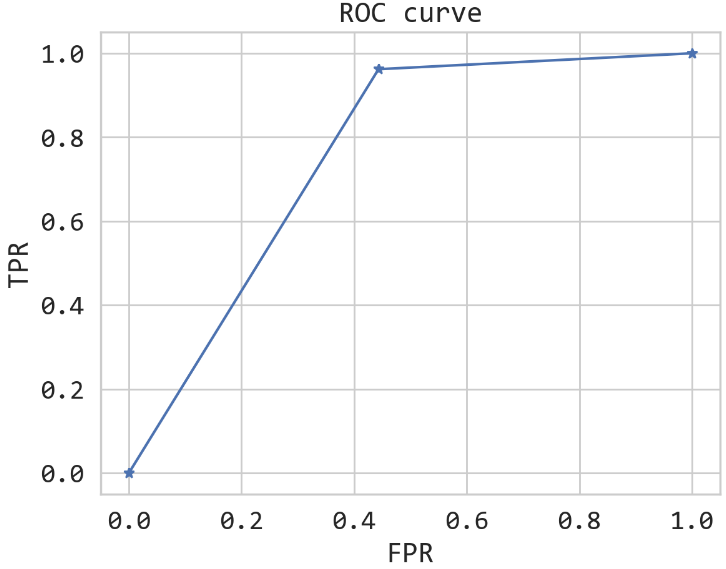
Precision: 0.926894403026594

Recall: 0.9620048504446241

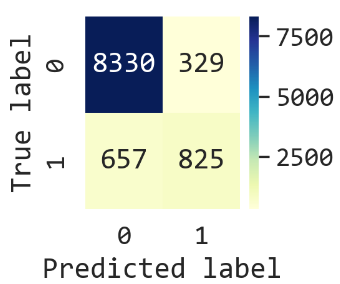
F1: 0.9441233140655106

AUC: 0.759342506193972

ROC曲线图：



混淆矩阵的热力图表示：



1. 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。