

**Python大数据与人工智能实践**

**课程设计报告**

|  |  |
| --- | --- |
| 姓 名： | 高书恒 |
| 学 院： | 计算机科学与技术学院 |
| 专 业： | 计算机科学与技术 |
| 班 级： | 计卓2101班 |
| 学 号： | U202115300 |
| 指导教师： | 邹逸雄 |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2023 年 1 月 13 日

目 录

[课程设计：烂番茄电影评论打分 1](#_Toc7955)

**[1 实验目的 1](#_Toc3685)**

**[2 实验内容 1](#_Toc29817)**

**[3 实验设计 2](#_Toc595)**

[3.1 开发环境 2](#_Toc17400)

[3.2 实验设计 2](#_Toc31516)

3.2.1问题分析...................................................................................................................2

3.2.1数据预处理...............................................................................................................3

3.2.1模型构建...................................................................................................................4

**[4 实验调试 7](#_Toc26721)**

[4.1 实验步骤 7](#_Toc5039)

4.1.1总体流程...................................................................................................................7

4.1.2模型主要优势...........................................................................................................7

4.1.3模型主要缺点...........................................................................................................5

4.1.4尝试的优化策略.......................................................................................................8

[4.2 实验调试及心得 1](#_Toc30431)0

4.2.1实验结果分析.........................................................................................................10

4.2.2主要出现的问题.....................................................................................................11

4.2.3心得体会.................................................................................................................11

**[附录 实验代码 1](#_Toc8883)2**

# 课程设计：烂番茄电影评论打分

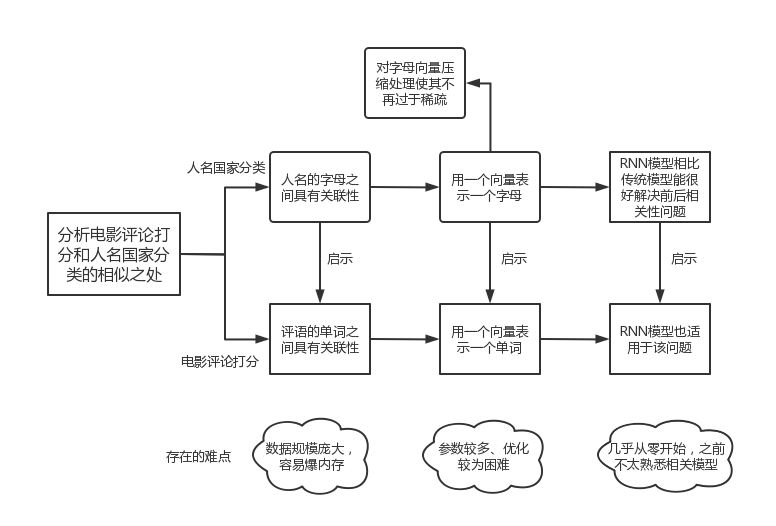
## 1实验目的

1. 理解常见机器学习模型RNN、LSTM、GRU的特点和内部原理。
2. 运用机器学习完成电影评论打分模型的训练。
3. 探索各种方法不断优化模型，提高准确率。

## 2实验内容

在烂番茄电影的评论区有大量观众的评语和对应的评分，它们之间很显然存在某种对应关系。如果评语大多是褒义、赞美、惊奇相关描述的语句，则评分一般会较高，反之亦然。因此，运用相关机器学习模型可以在知道某用户评语后预测该用户的评分。

对比所学的人名国家分类问题，电影评论打分问题具有诸多相似之处，我们从中获得启发（具体如图2.1所示），并克服相应的难点，最终训练出一个较好的模型。



**图2.1 电影评论打分和人名国家分类的相似之处**

本组使用了RNN循环神经网络模型。在尝试了原始RNN模型、LSTM模型、不同优化器如Adamax和AdamW、添加停用词表等多种优化策略后，最终选择了**GRU模型和Adam优化器**。随后进行了数据清洗、语句读入、词典构建、稀疏词向量转化、RNN模型训练、不断调整隐藏层大小和层数以及学习率等参数验证随机划分的测试集等步骤，最终模型的平均准确率达**67.46%**。

## 3实验设计

### 3.1 开发环境

|  |  |
| --- | --- |
| 代码编写软件 | Visual Studio Code |
| OS | Windows |
| Python版本 | 3.9.12 |
| Pytorch Compute Platform | CUDA 11.6 |

### 3.2 实验设计

**（1） 问题分析**

1. 单词组成语句的基本单位

对于语句的情感色彩分析，首先我们明确，对于该语句的分析是以一个单词为基本单位而不是以字母为基本单位的，因为若跨过单词层面而直接以字母为基本单位，字母组合作为输入比单词组合作为输入更为复杂，且需要更多的数据样本，更多的训练过程以提炼出特定字母组合组成的具有特定含义的单词。综上，我们采用单词为基本单位。

1. 5分类问题

不同于简单二分类（正面，负面），该数据标签共有5个情感等级，而其中等级3可以表示中性评价，且通过观察数据我们发现，许多子句是完整语句的拆分，由没有情感色彩或实际评价意义的词语组成，这里也可以视为等级3。进一步，为了提高训练效率，我们将没有参考意义的标点符号等停用符号不加入字典。

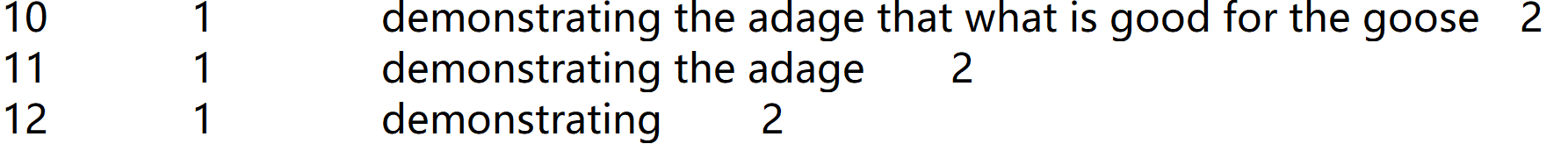
1. 单词间含义的相关性分析

在许多经典的机器学习模型中，如支持向量机，普通的人工神经网络，输入所产生的结果与各个维度进行输入的先后顺序是没有关系的。但对于自然语言处理，语音识别等问题，由于单词与单词之间的含义具有相关性，如否定，修饰，倒装等，我们必须把一个句子看成一个整体，且无法以传统的数据分析模型处理数据，否则会破坏单词顺序为主的潜在输入信息。

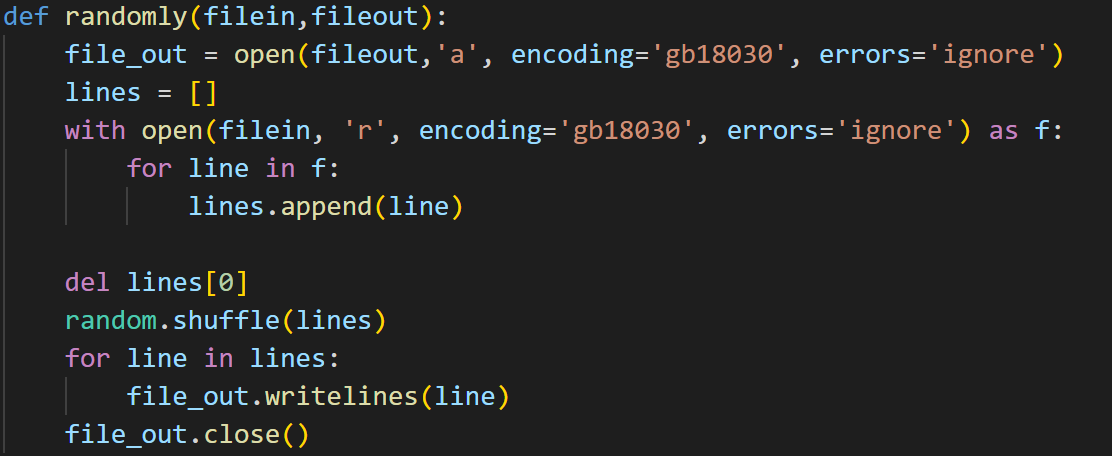
**（2）数据预处理**

1. 随机打乱

通过观察数据我们发现，在初始训练集中，出自同一语句的不同子句被划分在连续的序列区间，因此在相同条件下，完全打乱的数据集相较于没有被随机打乱的数据集拥有更多的训练不同语句的机会，即后者存在数据分布不均匀的问题。



对此我们定义randomly函数，依据序列将数据文件随机打乱。经过验证，数据集被随机打乱后训练集测试准确率（67%）比未打乱测试准确率（50%）平均高15%



**图3.1 相关代码**

1. 以特定的字典集组成标号与单词的映射关系

在处理输入数据时，对于某一单词，我们可以用一个具体数字来唯一标识它，使得每一个语句都简化成一串数字列表。这样以一一映射的方式代替单词，用特定数字表达特定单词的含义，既没有对单词的含义造成偏差，保持了输入数据的完整性，同时简化了输入数据，使得我们能更方便地建立模型，处理数据。例如对于如下语句

an absolute delight for all audiences

若我们有字典dic,且

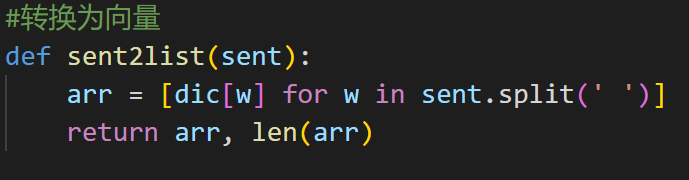
dic[0]=”delight” dic[3]=”an”

dic[1]=”absolute” dic[5]=”audiences”

dic[2]=”for” dic[6]=”all’

则我们根据字典标识对该语句进行如下的简化处理：

Sent\_list=[3,1,0,2,6,5]



**图3.2 转换为向量的代码**

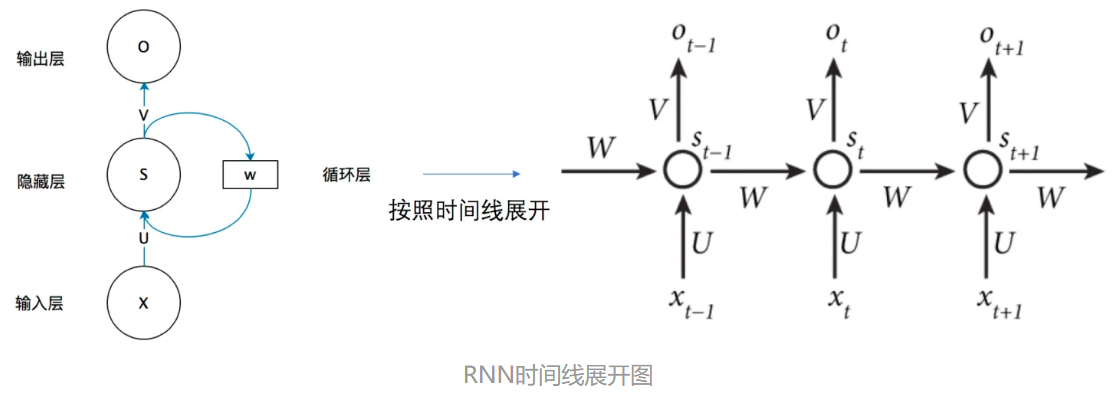
1. 划分训练集与测试集

在将数据集随机打乱后，我们将15万个数据样本按照3:1划分为训练集与测试集，在用训练集数据训练出神经网络参数后，用该模型对测试集数据进行预测，并统计预测准确率。

**（3）模型构建**

根据上述分析结论，且考虑到该问题属于输入关联的多分类问题，我们决定采用循环神经网络来构建具体模型。如下图所示：Xi代表输入中的每一个元素，U代表输入权重，Si代表隐藏层输出，W代表隐藏层输出权重，Oi表示该输入对应元素的输出。 即对于具有序列关系输入中的每一个元素Xi，通过与权重矩阵U相乘，与经过处理的上一隐藏层的输出Si-1相加，在当前隐藏层St非线性变换作用下，输出结果Oi。即有

文本, 信件

描述已自动生成

**图3.3 RNN时间线展开**

利用后向传播与梯度下降方法，我们可以训练得到网络模型参数，但此类方法存在梯度消失或梯度爆炸问题，即在训练时计算和反向传播，梯度倾向于在某一时刻递减或者递增，经过一段时间之后，梯度就会收敛到零，或者发散到无穷。也即随着输入序列变长，近距离输入元素成为梯度的主要决定因素，导致当前参数与远距离输入难以建立关联。

为了解决以上问题，我们采用GRU作为RNN隐藏层结点，GRU内部主要由重置控制门控r与控制更新门控z构成。在获得上一个传输下来的状态Si−1和当前节点的输入Xi后，我们获得两个门控的输出表达式：

r=sigmoid(W\*[Xi:St-1])

z=sigmoid(W\*[Xi:St-1])

其中[Xi:St-1]代表二者的增广向量，用sigmoid模拟门控信号的01状态。得到门控信号后，通过重置门信号r重置S’=tanh（W\*[Xi:St-1’]）.这里的St’主要包含当前输入的数据Xi。利用先前的更新门控z，我们对St进行更新

St=（1-z）\*St-1+z\*S’

该表达式中，（1-z）\*St-1在已经构建的隐藏层信息中进行选择性提取，放弃不重要的信息，即遗忘门。而z\*S’表示对当前输入元素进行选择性提取，即记忆门。其中1-z与z分别表示遗忘与记忆的多少。如图所示：

图示

描述已自动生成

**图3.4 GRU内部结构**

文本

描述已自动生成

**图3.5 关键代码**

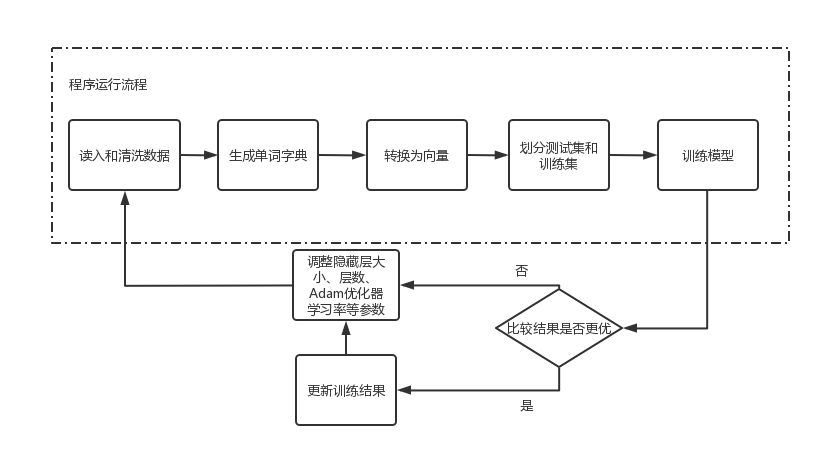
我们在自定义的RNN分类器中直接利用torch模块中neural network方法torch.nn.GRU。

## 4 实验调试

### 4.1 实验步骤

**1.总体流程**

实验总体流程如下图所示：



**图4.1 总体流程图**

**2.该模型的主要优势：**

RNN循环神经网络的原理理解和编写相对较为简单，且具有记忆功能，适合解决情感分析类问题。对于本题，对于对文本序列的非线性特征的学习较为恰当。

**3.该模型的主要缺点：**

RNN模型中下一时刻的输出仅受上一时刻影响，可能在一句话中前后相隔较远的单词存在联系而未被考虑到，从而不够准确。此外，训练时需要的内存较大，参数寻优较为困难。使用改进版的LSTM可以相对改善较远单词的联系问题，但LSTM计算复杂且会极大地降低运算速度，在普通电脑上运行受到一定的限制。

**4.尝试的优化策略：**

在此流程的基础上，我们继续尝试了其它的优化选择，例如：

1. **利用网上开源的glove.6B词向量数据集**

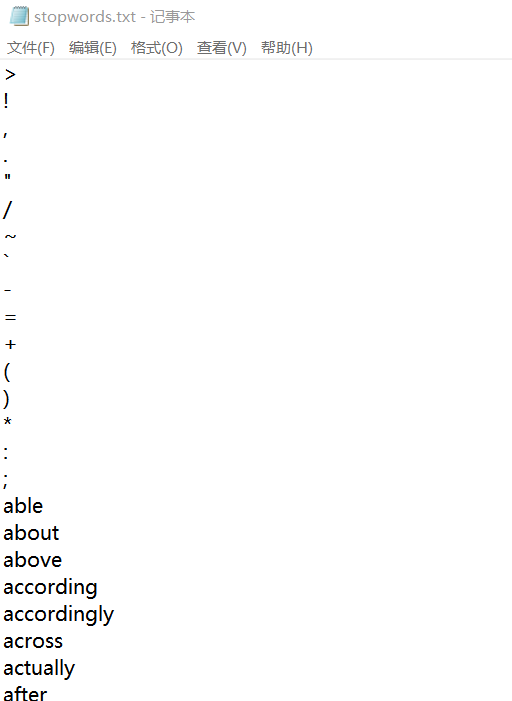
**优化结果：**经过测试，我们发现glove.6B的数据集过大，特有词与初始读入的数据词表不够契合，且该泛化的语言情感数据集应用于电影评论场景所带来的优化很低。

1. **使用不同的优化器如AdamW、SGD、Adamax等**

**优化结果：**其它优化器如Adadelta、LBFGS、SGD等均不适合该模型，循环训练后准确率没有丝毫改善，而AdamW优化器准确率略低于Adam，Adamax优化器峰值准确率略低于Adam，收敛后与Adam相当。

1. **使用停用词表屏蔽“and”一类的连接词**

**优化结果：**使用停用词表屏蔽连词后准确率反而有所下降，最佳参数下平均为64.16%。分析其原因可能是因为数据集本身短语较多，长篇的语句很少，若其中出现看似中性的词如“？”、“！”、“did”、“but”等往往评分有较为极端的趋势，因此不适用于该数据集。



**图4.2 部分停用词示例**

1. **分别使用GRU模型和LSTM模型进行预先测试**

**优化结果：**为了确定选择的模型，我们考虑了GRU和LSTM两种模型对原先的RNN模型进行尝试优化。最初我们未调整参数时，使用RNN的预测准确率约为52%。而后使用GRU模型时，它的准确率提高至62%。

我们还尝试了LSTM模型以测试该模型对于情感分类的准确度。我们将原代码中的RNN分类器从使用GRU模型转换为LSTM的代码，并对原数据进行处理。经过测试，其最优拟合时准确率为56.1%，没有使用GRU时的正确率高。

其原因可能是该数据集大多数语句长度较短，而短语较多。LSTM模型具有两种不同的输入门和一个遗忘门，使得其对于长句效率更高，在处理短句时未必有GRU模型有效。另外，LSTM模型在其模型加载过程中所需要使用的空间比GRU大，超出了我们计算设备的显存上限，而只能用CPU运行，使得其计算速度较慢。

基于这两种不利因素的考虑，我们最终选择了GRU模型进行进一步训练，并尝试找得其最佳参数。我们**初步训练**的结果如下表。

|  |  |  |  |
| --- | --- | --- | --- |
| 不同模型 | 训练结果（最优拟合时的训练损失值和正确率） | 测试正确率 | 训练时间 |
| RNN模型（3层，随机初始化） | Loss\_Max = 0.00713219  Accuracy\_Max = 61.93%  Loss\_Final = 0.00031276  Accuracy\_Final = 58.05% | 61.93% | 923s(GPU) |
| LSTM模型（3层，随机初始化） | Loss = 0.01869482  Accuracy = 56.10%  Loss\_Final = 0.00400197  Accuracy\_Final = 52.92% | 56.10% | 5644s(CPU) |
| GRU模型（3层，随机初始化） | Loss = 0.00108742  Accuracy = 65.82%  Loss\_Final = 0.00018147  Accuracy\_Final = 61.99% | 65.82% | 540s(GPU) |

**表4.1 初步训练结果**

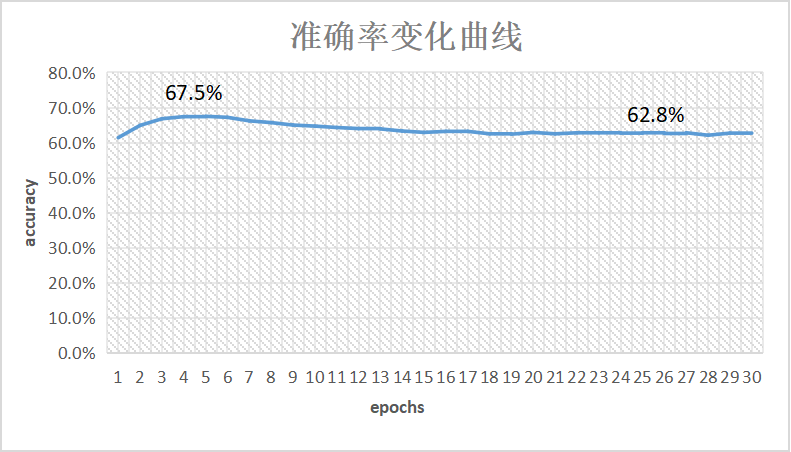
### 4.2 实验调试及心得

**1.调试结果分析**

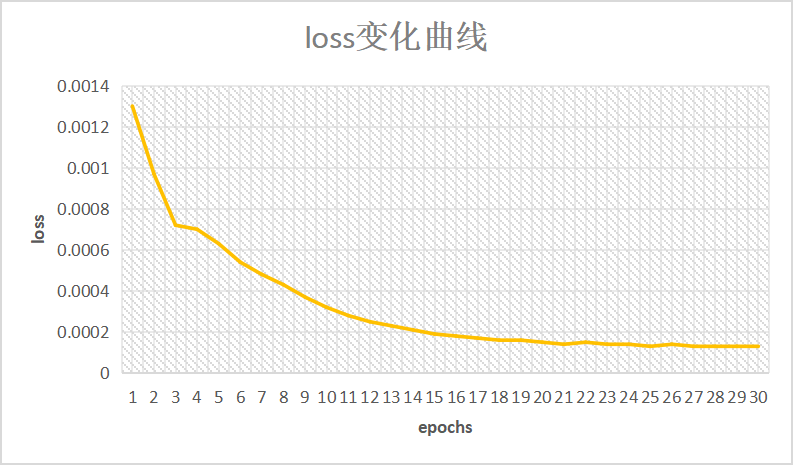
我们通过逐步调试和反复试验，使用不断迭代参数的方式尝试获得GRU模型的最佳参数。经过漫长的调试，我们获得的最优参数如下表所示。

|  |  |  |
| --- | --- | --- |
| HIDDEN\_SIZE（大小） | N\_LAYER（层数） | Learning Rate（学习率） |
| 70 | 3 | 0.004 |

准确率及损失函数随迭代次数关系如下图所示：



**图4.3 准确率曲线**



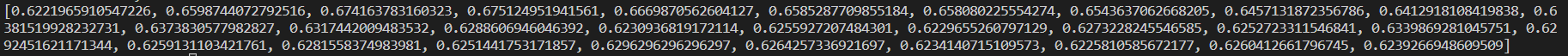
**图4.4 loss曲线**

通过上述准确率曲线和损失值曲线，我们可以看出随着训练的进行，其训练得出的准确率先上升后下降。这是符合规律的，模型随着训练先逐步与数据集拟合，则准确率上升；随后发生了过拟合现象，使得其准确率开始有下降现象。而loss曲线逐渐收敛至一个较低的稳定值，表示真实值和预测值相差较小。

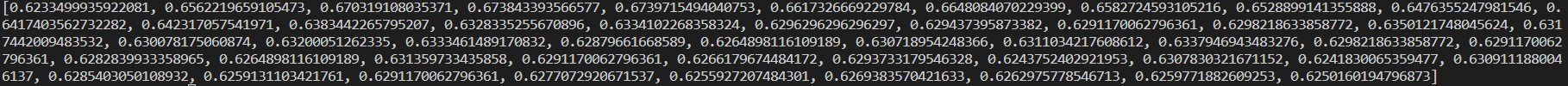
我们将相应的运行截图附录如下。

屏幕截图 2023-01-10 203534

**图4.4 第1次运行准确率**



**图4.5 第2次运行准确率**



**图4.6 第3次运行准确率**

**2.主要出现的问题：**

（1）最开始仍对pytorch的使用熟悉度不高，对机器学习模型较为陌生

解决方法：我们仔细分析了教学PPT，自学相关知识的使用，并找相关案例练习。

（2）使用传统稀疏矩阵表示词向量导致内存不够

解决方法：我们使用embedding的压缩技巧，构建高效的词向量。

1. 参数过多，调到最优有些困难

解决方法：我们采用迭代调试的方式，探索出大致区间，并在其附近不断寻找，最后逐渐将正确率提高至67%以上。

**3.心得体会**

我们经过对原题进行整体的分析，再选择模型、调试参数，最终获得了较好的结果。经过这次大作业的练习，我们增进了对于RNN模型的认识和使用，也激发了我们对机器学习这一庞杂学科的兴趣。在调参时的优化探索需要很强的直觉和耐心，在选择优化模型时也要注意应用范围以及和要处理数据的锲合度，这样可以省去不少麻烦，少走很多弯路。本次作业的实质是NLP情感分析问题，该领域正在蓬勃发展，是当下AI热点之一，未来将会有很多新的理论和模型提出。总而言之，此次大作业让我们受益匪浅。

## 附录 实验代码

**源代码如下：**

import torch

import random

from torch.utils.data import Dataset, DataLoader

from torch.nn.utils.rnn import pad\_sequence, pack\_padded\_sequence, pack\_sequence, pad\_packed\_sequence

import gzip

import csv

import time

import pandas as pd

import os

*# Parameters*

HIDDEN\_SIZE = 70

BATCH\_SIZE = 100

N\_LAYER = 3

N\_EPOCHS = 4

N\_WORDS = 40000

N\_LABELS = 5

USE\_GPU = True

N\_TRAIN = 156060

PART = 140454*#训练集占比0.9,测试集占比0.1*

dic={}

N\_TEST = 66292

cnt=1

**class** SentDataset(Dataset):

**def** \_\_init\_\_(self,part,file, is\_train\_set=True):

        f=open(file,'r')

        self.sents=[]

        self.labels = []

        reader = csv.reader(f,delimiter='\t')*#读取数据*

        rows = list(reader)

        del rows[0]

        if is\_train\_set:*#读入训练集*

            for i in range(part):

                self.sents.append(rows[i][2])

                self.labels.append(int(rows[i][3]))

            self.len=len(self.sents)

            self.label\_num = len(self.labels)

        else:*#读入测试集*

            for i in range(part,N\_TRAIN):

                self.sents.append(rows[i][2])

                self.labels.append(int(rows[i][3]))

            self.len=len(self.sents)

            self.label\_num = len(self.labels)

**def** \_\_getitem\_\_(self, index):

        return self.sents[index], self.labels[index]

**def** \_\_len\_\_(self):

        return self.len

**class** SentTestset(Dataset):

**def** \_\_init\_\_(self,file):

        f=open(file,'r')

        self.sents=[]

        self.len=0

        reader = csv.reader(f,delimiter='\t')*#读取数据*

        rows = list(reader)

        del rows[0]

        for i in range(N\_TEST):

            self.sents.append(rows[i][2])

        self.len=len(self.sents)

**def** \_\_getitem\_\_(self, index):

        return self.sents[index]

**def** \_\_len\_\_(self):

        return len(self.sents)

*#RNN分类器*

**class** RNNClassifier(torch.nn.Module):

**def** \_\_init\_\_(self, input\_size, hidden\_size, output\_size, n\_layers=1, bidirectional=True):

        super(RNNClassifier, self).\_\_init\_\_()

        self.hidden\_size = hidden\_size

        self.n\_layers = n\_layers

        self.n\_directions = 2 if bidirectional else 1

        self.embedding = torch.nn.Embedding(input\_size, hidden\_size)

        self.gru = torch.nn.GRU(hidden\_size, hidden\_size, n\_layers,

                                bidirectional=bidirectional)

        self.fc = torch.nn.Linear(hidden\_size \* self.n\_directions, output\_size)

**def** \_init\_hidden(self, batch\_size):

        hidden = torch.zeros(self.n\_layers \* self.n\_directions,

                            batch\_size, self.hidden\_size)

        return create\_tensor(hidden)

**def** forward(self, input, seq\_lengths):

        input = input.t()

        batch\_size = input.size(1)

        hidden = self.\_init\_hidden(batch\_size)

        embedding = self.embedding(input)

        output, hidden = self.gru(embedding, hidden)

        if self.n\_directions == 2:

            hidden\_cat = torch.cat([hidden[-1], hidden[-2]], dim=1)

        else:

            hidden\_cat = hidden[-1]

        fc\_output = self.fc(hidden\_cat)

        return fc\_output

**def** tensors\_train(sents, labels):

    labels=torch.Tensor(labels)

    sequences\_and\_lengths = [sent2list(sent) for sent in sents]

    sent\_sequences = [sl[0] for sl in sequences\_and\_lengths]

    seq\_lengths = torch.LongTensor([sl[1] for sl in sequences\_and\_lengths])

labels = labels.long()

    seq\_tensor = torch.zeros(len(sent\_sequences), seq\_lengths.max()).long()

    for idx, (seq, seq\_len) in enumerate(zip(sent\_sequences, seq\_lengths), 0):

        seq\_tensor[idx, :seq\_len] = torch.LongTensor(seq)

    return create\_tensor(seq\_tensor), \

           create\_tensor(seq\_lengths),\

           create\_tensor(labels)

**def** tensors\_test(sents):

    sequences\_and\_lengths = [sent2list(sent) for sent in sents]

    sent\_sequences = [sl[0] for sl in sequences\_and\_lengths]

seq\_lengths = torch.LongTensor([sl[1] for sl in sequences\_and\_lengths])

    seq\_tensor = torch.zeros(len(sent\_sequences), seq\_lengths.max()).long()

    for idx, (seq, seq\_len) in enumerate(zip(sent\_sequences, seq\_lengths), 0):

        seq\_tensor[idx, :seq\_len] = torch.LongTensor(seq)

    return create\_tensor(seq\_tensor), \

           create\_tensor(seq\_lengths)

**def** sent2list(sent):*#将句子转换成由单词序列组成的列表*

    arr=[]

    for w in sent.split(' '):

        w=w.lower()

        if w in dic.keys():

            arr.append(dic[w])

*#arr = [dic[w] for w in sent.split(' ')]*

    return arr, len(arr)

*# def sent2list(sent):*

*#     arr = [dic[w.lower()] for w in sent.split(' ')]*

*#     return arr, len(arr)*

*#创建tensor*

**def** create\_tensor(tensor):

    if USE\_GPU:

        device = torch.device("cuda:0")

        tensor = tensor.to(device)

return tensor

*#训练模型*

**def** trainModel():

    total\_loss = 0

    for i, (sents, labels) in enumerate(trainloader, 1):

        inputs, seq\_lengths, target = tensors\_train(sents, labels)

        output = classifier(inputs, seq\_lengths)

        loss = criterion(output, target)

        optimizer.zero\_grad()

        loss.backward()

        optimizer.step()

        total\_loss += loss.item()

        if i % 10 == 0:

            print(**f**'[{i \* len(inputs)}/{len(trainset)}] ', end='')

            print(**f**'loss={total\_loss / (i \* len(inputs))}')

    return total\_loss

*#测试准确率*

**def** testModel():

    correct = 0

    total = len(testset)

    print("evaluating trained model ...")

    with torch.no\_grad():

        for i, (sents, labels) in enumerate(testloader, 1):

            inputs, seq\_lens, target = tensors\_train(sents, labels)

            output = classifier(inputs, seq\_lens)

            pred = output.max(dim=1, keepdim=True)[1]

            correct += pred.eq(target.view\_as(pred)).sum().item()

        percent = '%.2f' % (100 \* correct / total)

        print(**f**'Test set: Accuracy {correct}/{total} {percent}%')

return correct / total

**def** test():

*#total = len(verifyset)*

    f1 = open('C:\\Users\\2901233546\\Desktop\\1.3\\python\_work\\data\\submission.csv', 'w')

    cnt=156061

    with torch.no\_grad():

        for i, (sents) in enumerate(verifyloader, 1):

            inputs,seq\_lens=tensors\_test(sents)

            output = classifier(inputs, seq\_lens)

            pred = output.max(dim=1, keepdim=True)[1]

            lis1=pred.tolist()

            for e in lis1:

*# e.insert(0,str(cnt))*

                f1.writelines(str(cnt)+','+str(e[0])+os.linesep)

*# f1.write(',')*

*# f1.write(str(e[0])+os.linesep)*

                cnt=cnt+1

*# print(output,type(output),len(output))*

f1.close()

*#随机打乱划分训练集和测试集*

**def** randomly(filein,fileout):

    file\_out = open(fileout,'a', encoding='gb18030', errors='ignore')

    lines = []

    with open(filein, 'r', encoding='gb18030', errors='ignore') as f:

        for line in f:

            lines.append(line)

    del lines[0]

    random.shuffle(lines)

    for line in lines:

        file\_out.writelines(line)

file\_out.close()

*#将出现过的所有单词存入字典*

**def** prepare(filename,num):

**global** cnt

f=open(filename,'r')

    for i in range(num+1):

        line=f.readline()

        if i == 0 :

            continue

        for word in line.split('\t')[2].split(' '):

            word=word.lower()

            if word not in dic.keys():

                cnt=cnt+1

                dic[word]=cnt

if \_\_name\_\_ == '\_\_main\_\_':

    filein = 'C:\\Users\\2901233546\\Desktop\\1.3\\python\_work\\data\\train.tsv'

    fileout = 'C:\\Users\\2901233546\\Desktop\\1.3\\python\_work\\data\\train1.tsv'

fileverify='C:\\Users\\2901233546\\Desktop\\1.3\\python\_work\\data\\test.tsv'

    prepare(filein,N\_TRAIN)

    prepare(fileverify,N\_TEST)

    print("Training for %d epochs..." % N\_EPOCHS)

    classifier = RNNClassifier(N\_WORDS, HIDDEN\_SIZE, N\_LABELS, N\_LAYER)

    if USE\_GPU:

        device = torch.device("cuda:0")

        classifier.to(device)

    criterion = torch.nn.CrossEntropyLoss()

*# optimizer = torch.optim.Adam(classifier.parameters(), lr=0.0035)#此处调整学习率*

    optimizer = torch.optim.Adam(classifier.parameters(), lr=0.004)*#此处调整学习率*

*#110,0.005   67.01*

*#90,0.003    67.38*

*#90,0.004    67.46*

*#90,0.001    67.04*

*#80,0.003    67.12*

*#70,0.0035   67.49*

*#70,0.003    67.39*

*#70,0.004    67.34*

*#70,0.002    66.81*

*#90,0.004  Adamax  66.56*

*#90,0.001    66.59*

    acc\_list = []*#记录准确率*

*# randomly(filein,fileout)#打乱数据*

    verifyset = SentTestset(fileverify)*#读入验证集*

verifyloader = DataLoader(verifyset, batch\_size=BATCH\_SIZE, shuffle=False)

    trainset = SentDataset(PART,fileout,is\_train\_set=True)*#读入训练集*

    trainloader = DataLoader(trainset, batch\_size=BATCH\_SIZE, shuffle=True)

    testset = SentDataset(PART,fileout,is\_train\_set=False)*#读入测试集*

testloader = DataLoader(testset, batch\_size=BATCH\_SIZE, shuffle=False)

    for epoch in range(1, N\_EPOCHS + 1):

        print('epoch: %d'%epoch)

        trainModel()*#训练模型*

        acc = testModel()*#测试*

        acc\_list.append(acc)*#记录准确率*

    test()

*#print(acc\_list)*

**实验中我们使用的LSTM所改动的相关函数代码如下：**

*#LSTM*

**class** RNNClassifier(torch.nn.Module):

**def** \_\_init\_\_(self, input\_size, hidden\_size, output\_size, num\_layers=1, bidirectional=True):

        super(RNNClassifier,self).\_\_init\_\_()

        self.embedding=torch.nn.Embedding(input\_size,N\_WORDS)

        self.encoder=torch.nn.LSTM(input\_size=input\_size,

                            hidden\_size=hidden\_size,

                            num\_layers=num\_layers,

                            bidirectional=True)

        self.decoder=torch.nn.Linear(4\*hidden\_size,output\_size)

**def** \_init\_hidden(self, batch\_size):

        hidden = torch.zeros(self.n\_layers \* self.n\_directions,

                            batch\_size, self.hidden\_size)

        return create\_tensor(hidden)

**def** forward(self, input, seq\_lengths):

*# input  shape  : B x  S -> S x  B*

*# inputs的形状是(批量大小, seq\_len)，因为LSTM需要将序列长度(seq\_len)作为第一维，所以将输入转置后*

*# 再提取词特征，输出形状为(seq\_len, 批量大小, 词向量维度)*

        input = input.t()

        embeddings=self.embedding(input)

*# outputs形状是(seq\_len, 批量大小, 2 \* 隐藏单元个数)*

        outputs,\_=self.encoder(embeddings)

*# 连结初始时间步和最终时间步的隐藏状态作为全连接层输入。它的形状为*

*# (批量大小, 4 \* 隐藏单元个数)。*

        encoding = torch.cat((outputs[0], outputs[-1]), -1)

        outs = self.decoder(encoding)

        return outs

**实验中我们使用的停用词表所改动的相关函数代码如下：**

**def** prepare(filename):*#搜集所有单词，构建字典*

    f=open('D:\Personal\pythonClass\\bighomework\stopwords.txt',"r",encoding='utf-8')

    for i in range(1,568):

        a=f.readline()

        a=a[0:-1]

        stop\_list.append(a)

    f=open(filename,'r')

    cnt=-1

    for i in range(N\_TRAIN+1):

        line=f.readline()

        if i == 0 :

            continue

        for word in line.split('\t')[2].split(' '):

            word=word.lower()

            if word not in dic.keys() and word not in stop\_list:

                cnt=cnt+1

                dic[word]=cnt

**def** sent2list(sent):*#将句子转换成由单词序列组成的列表*

    arr=[]

    for w in sent:

        if w in dic.keys():

            arr.append(dic[w])

*#arr = [dic[w] for w in sent.split(' ')]*

    return arr, len(arr)