Python 大数据与人工智能课程设计报告

姓　　名：	高小博
学　　院：	计算机科学与技术学院
专　　业：	计算机科学与技术
班　　级：	2103
学　　号：	U202115410
指导教师：	周正勇，邹逸雄

分数	
教师签名	

2023 年 1 月 15 日

目录

1 实验名

Sentiment Analysis on Movie Reviews

1.1 实验目的

正文统一采用小四号宋体/Times New Roman 和 1.25 倍行距。

随着互联网技术以及社交网络技术的不断发展，越来越多的人更加喜欢在网络上发表自己对于电影电视剧等多媒体艺术的评论看法，通过这些在线的影评，这也为电影投资人带来了更加便利的方式来获取观影用户对于电影的反馈，从而能够做出更加可靠合理的电影投资决策，同时这也使得电影爱好者们在观影前能够对各电影有一个比较初步的了解，协助选择观影。

当然要达到这些目的，对网络上大量的影评进行合理的数据处理是必要的，这必须借助计算机的手段来完成，在这个领域中，情感分析是一个非常重要的课题。

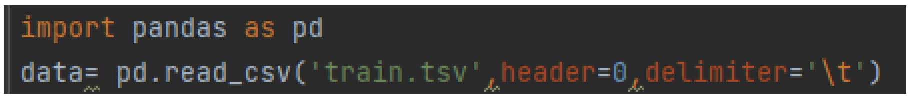
情感分析或者也叫观点挖掘[1]是一种研究人们对于某些事物例如产品、服务、个人、组织等等的观点和看法的基于计算机的研究[2]，主要的是将文本分类为两类积极的（正面的）和消极的（负面的），自从在 2002 年被提出[3]，情感分析已经成了为自然语言处理（NLP）领域一个非常活跃的研究方向

1.2 实验内容

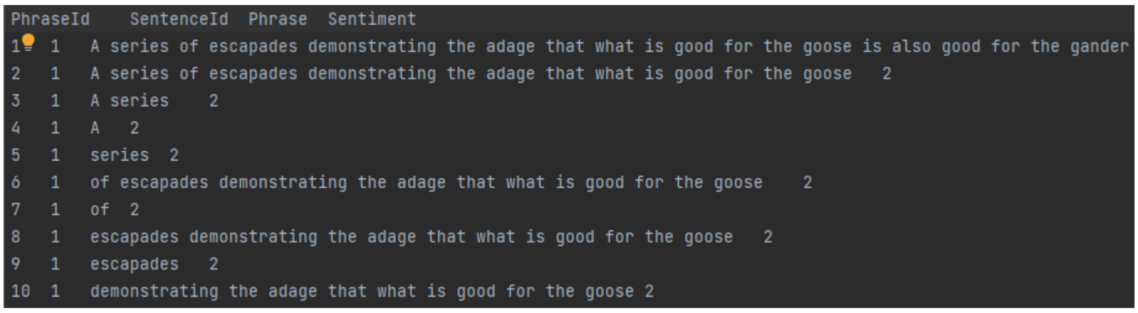
该任务是对影评的数据做分类，积极还是消极的评论。通过基于词频统计作为特征的分类模型对 Kaggle 上的电影评论进行情感分析。Rotten Tomatoes 电影评论数据集是用于情感分析的电影评论语料库，最初由 Pang 和 Lee [1]收集。在他们关于情感树库的工作中，Socher 等人。[2]使用亚马逊的 Mechanical Turk 为语料库中的所有解析短语创建细粒度标签。本次竞赛提供了一个机会，可以对您在 Rotten Tomatoes 数据集上的情绪分析想法进行基准测试。你被要求按五个等级标记短语：消极，有点消极，中立，有点积极，积极。句子否定，讽刺，简洁，语言模糊以及许多其他障碍使得这项任务非常具有挑战性。

1.3 实验设计

利用 pandas 库对 train.tsv 数据集进行解析可打开该训练数据集：

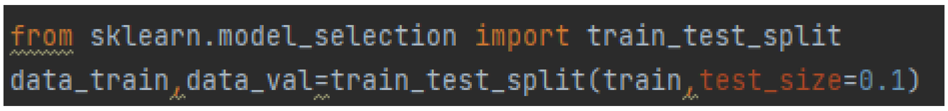


```
import pandas as pd
data= pd.read_csv('train.tsv',header=0,delimiter='\t')
```



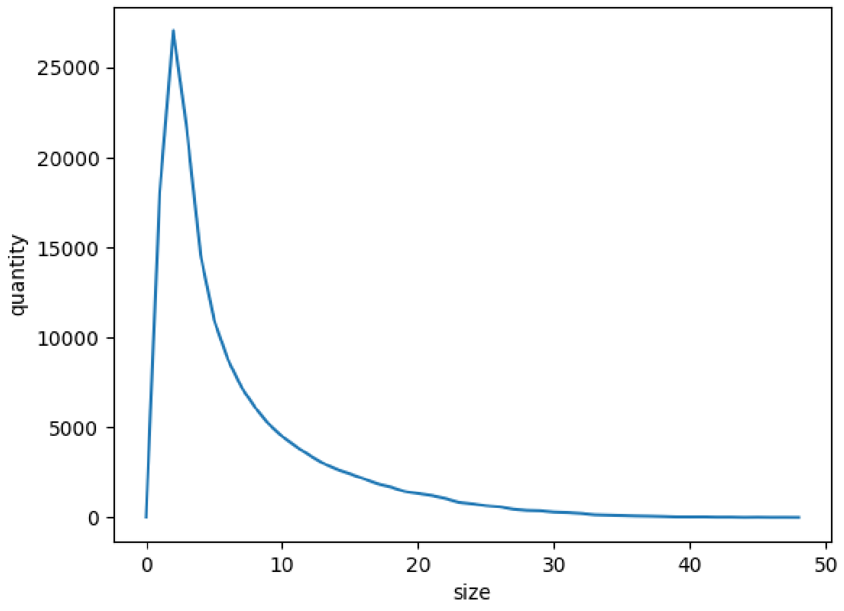
```
PhraseId    SentenceId  Phrase  Sentiment
1   1   A series of escapades demonstrating the adage that what is good for the goose is also good for the gander
2   1   A series of escapades demonstrating the adage that what is good for the goose   2
3   1   A series    2
4   1   A   2
5   1   series  2
6   1   of escapades demonstrating the adage that what is good for the goose    2
7   1   of  2
8   1   escapades demonstrating the adage that what is good for the goose   2
9   1   escapades   2
10  1   demonstrating the adage that what is good for the goose 2
```

利用 sklearn 中的 train_test_split 函数对训练数据（train.tsv）中按 9：1 比例随机划分训练集（data_train）与验证集（data_val）：

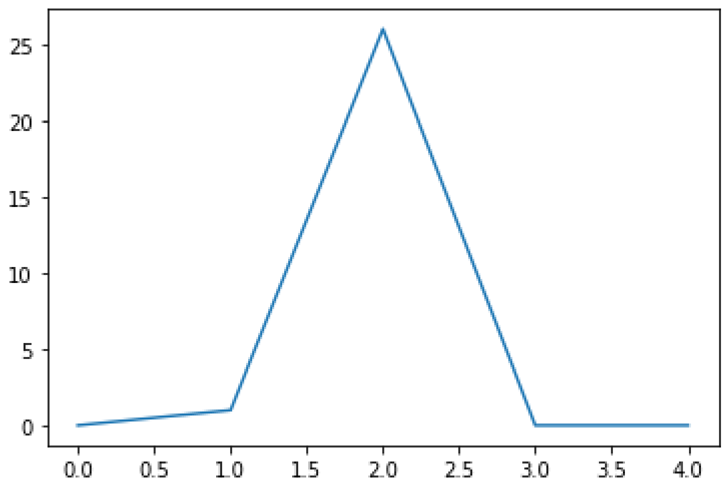


```
from sklearn.model_selection import train_test_split
data_train,data_val=train_test_split(train,test_size=0.1)
```

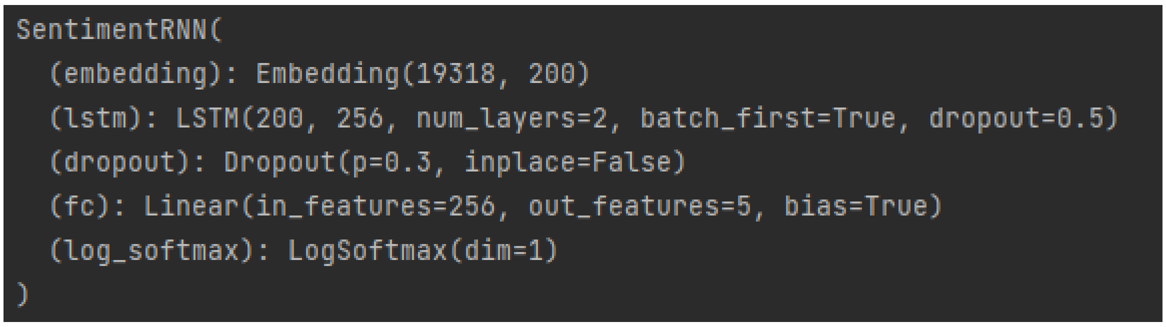
对数据集分词并统计，可以发现大部分词汇集中在 0-10 区间，可视化展示为：



统计训练数据集标签中出现最多的分数为 2，可视化展示效果为：

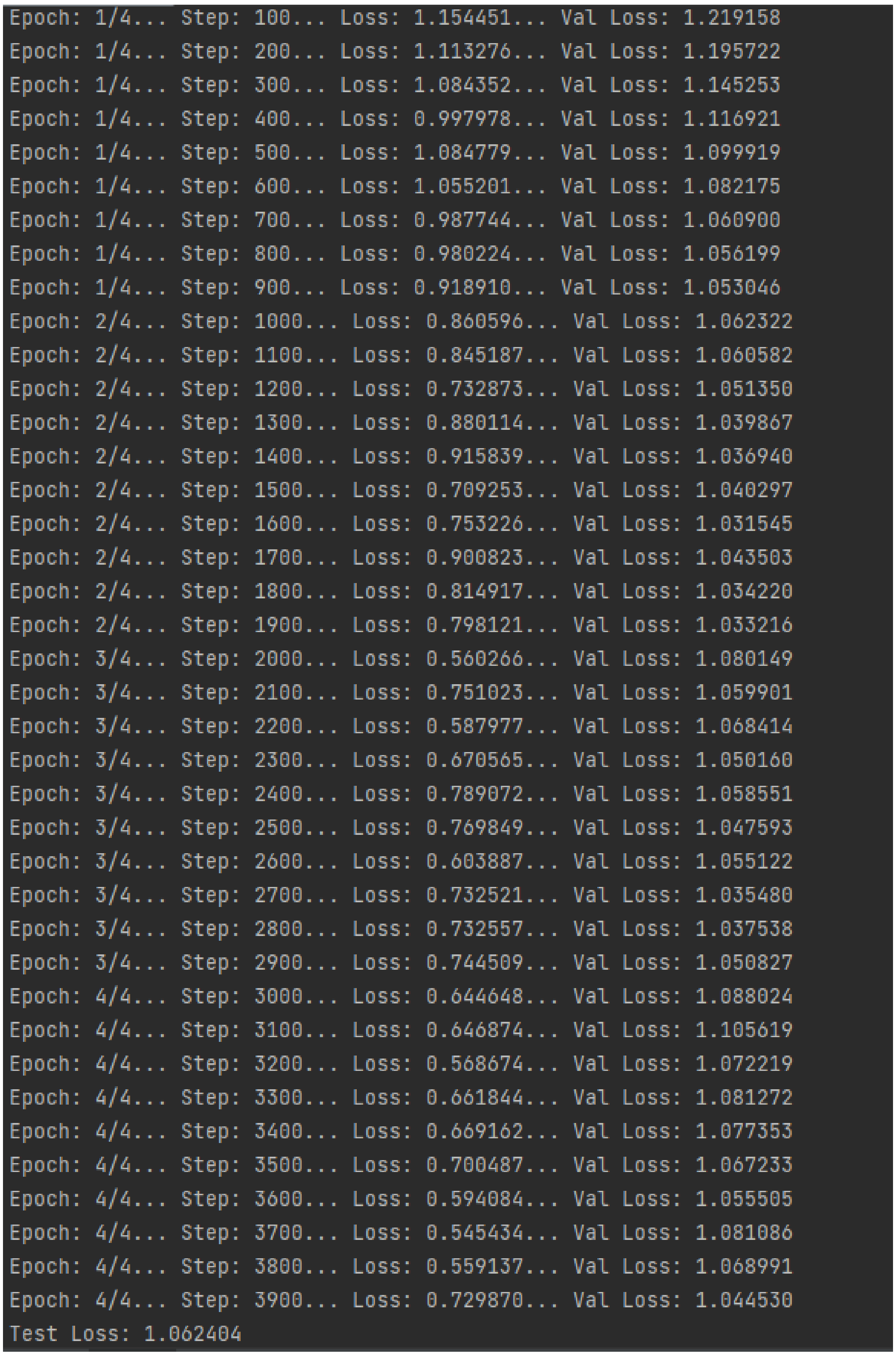


网络模型选择 SentimentRNN 模型，网络结构为：



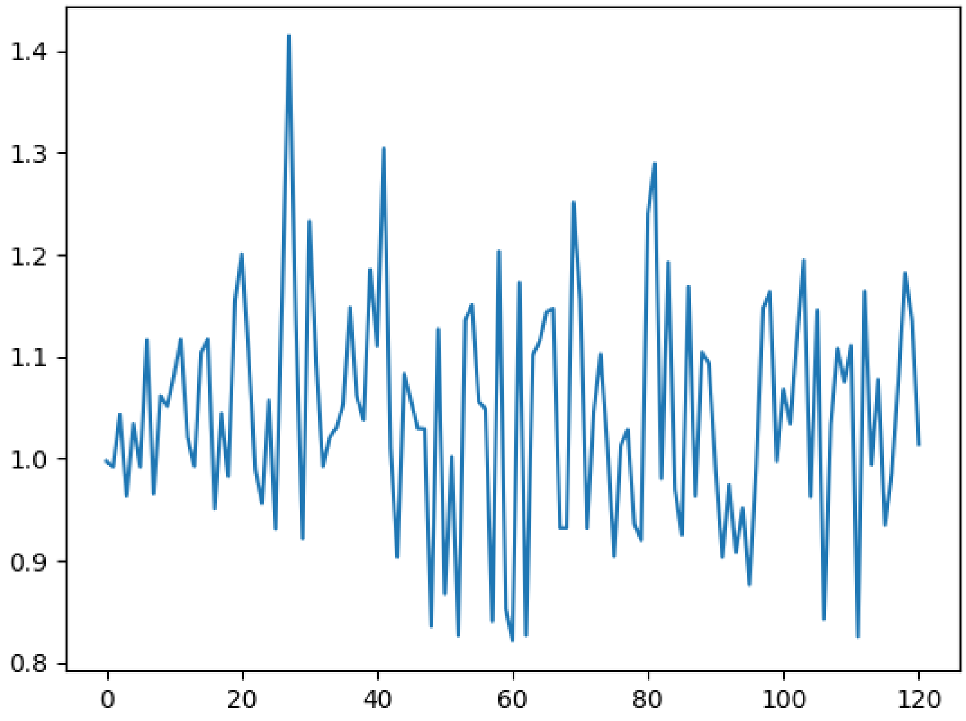
```
SentimentRNN(
  (embedding): Embedding(19318, 200)
  (lstm): LSTM(200, 256, num_layers=2, batch_first=True, dropout=0.5)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc): Linear(in_features=256, out_features=5, bias=True)
  (log_softmax): LogSoftmax(dim=1)
)
```

对训练数据集进行训练：训练 4 轮；batch_size 为 128；损失函数选择 NLLLoss();优化器选择 Adam；学习率选择：0.001；训练过程经过 GPU 计算，训练过程如下：

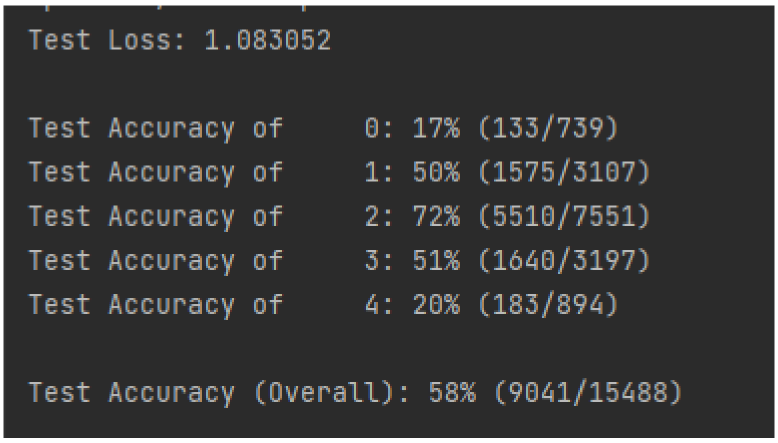


```
Epoch: 1/4... Step: 100... Loss: 1.154451... Val Loss: 1.219158
Epoch: 1/4... Step: 200... Loss: 1.113276... Val Loss: 1.195722
Epoch: 1/4... Step: 300... Loss: 1.084352... Val Loss: 1.145253
Epoch: 1/4... Step: 400... Loss: 0.997978... Val Loss: 1.116921
Epoch: 1/4... Step: 500... Loss: 1.084779... Val Loss: 1.099919
Epoch: 1/4... Step: 600... Loss: 1.055201... Val Loss: 1.082175
Epoch: 1/4... Step: 700... Loss: 0.987744... Val Loss: 1.060900
Epoch: 1/4... Step: 800... Loss: 0.980224... Val Loss: 1.056199
Epoch: 1/4... Step: 900... Loss: 0.918910... Val Loss: 1.053046
Epoch: 2/4... Step: 1000... Loss: 0.860596... Val Loss: 1.062322
Epoch: 2/4... Step: 1100... Loss: 0.845187... Val Loss: 1.060582
Epoch: 2/4... Step: 1200... Loss: 0.732873... Val Loss: 1.051350
Epoch: 2/4... Step: 1300... Loss: 0.880114... Val Loss: 1.039867
Epoch: 2/4... Step: 1400... Loss: 0.915839... Val Loss: 1.036940
Epoch: 2/4... Step: 1500... Loss: 0.709253... Val Loss: 1.040297
Epoch: 2/4... Step: 1600... Loss: 0.753226... Val Loss: 1.031545
Epoch: 2/4... Step: 1700... Loss: 0.900823... Val Loss: 1.043503
Epoch: 2/4... Step: 1800... Loss: 0.814917... Val Loss: 1.034220
Epoch: 2/4... Step: 1900... Loss: 0.798121... Val Loss: 1.033216
Epoch: 3/4... Step: 2000... Loss: 0.560266... Val Loss: 1.080149
Epoch: 3/4... Step: 2100... Loss: 0.751023... Val Loss: 1.059901
Epoch: 3/4... Step: 2200... Loss: 0.587977... Val Loss: 1.068414
Epoch: 3/4... Step: 2300... Loss: 0.670565... Val Loss: 1.050160
Epoch: 3/4... Step: 2400... Loss: 0.789072... Val Loss: 1.058551
Epoch: 3/4... Step: 2500... Loss: 0.769849... Val Loss: 1.047593
Epoch: 3/4... Step: 2600... Loss: 0.603887... Val Loss: 1.055122
Epoch: 3/4... Step: 2700... Loss: 0.732521... Val Loss: 1.035480
Epoch: 3/4... Step: 2800... Loss: 0.732557... Val Loss: 1.037538
Epoch: 3/4... Step: 2900... Loss: 0.744509... Val Loss: 1.050827
Epoch: 4/4... Step: 3000... Loss: 0.644648... Val Loss: 1.088024
Epoch: 4/4... Step: 3100... Loss: 0.646874... Val Loss: 1.105619
Epoch: 4/4... Step: 3200... Loss: 0.568674... Val Loss: 1.072219
Epoch: 4/4... Step: 3300... Loss: 0.661844... Val Loss: 1.081272
Epoch: 4/4... Step: 3400... Loss: 0.669162... Val Loss: 1.077353
Epoch: 4/4... Step: 3500... Loss: 0.700487... Val Loss: 1.067233
Epoch: 4/4... Step: 3600... Loss: 0.594084... Val Loss: 1.055505
Epoch: 4/4... Step: 3700... Loss: 0.545434... Val Loss: 1.081086
Epoch: 4/4... Step: 3800... Loss: 0.559137... Val Loss: 1.068991
Epoch: 4/4... Step: 3900... Loss: 0.729870... Val Loss: 1.044530
Test Loss: 1.062404
```

其训练损失可视化效果为：



测试平均准确率：58%：

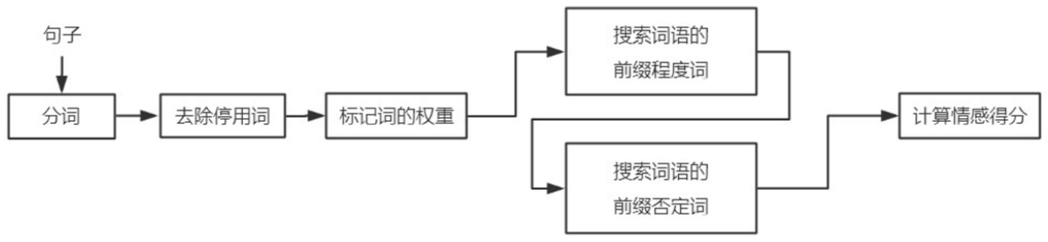


```
Test Loss: 1.083052

Test Accuracy of     0: 17% (133/739)
Test Accuracy of     1: 50% (1575/3107)
Test Accuracy of     2: 72% (5510/7551)
Test Accuracy of     3: 51% (1640/3197)
Test Accuracy of     4: 20% (183/894)

Test Accuracy (Overall): 58% (9041/15488)
```

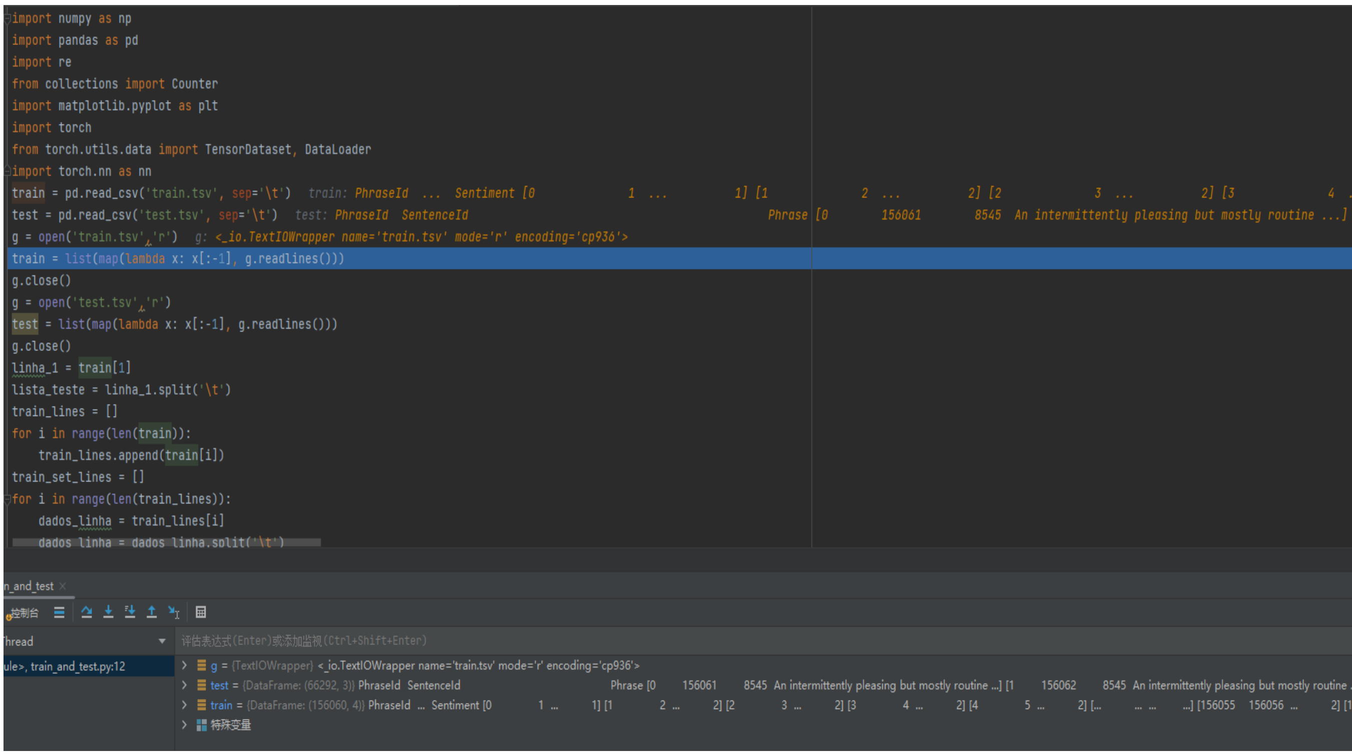
1.3.1开发环境

Python3.7；torch=1.13； 显卡 RTX3050 ； CUDA=11.6;程序依赖包：numpy ； pandas ;re; collections; matplotlib

1.3.2 实验设计



1.4 实验调试


```
import numpy as np
import pandas as pd
import re
from collections import Counter
import matplotlib.pyplot as plt
import torch
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
train = pd.read_csv('train.tsv', sep='\t')  train: PhraseId  ...  Sentiment [0          1  ...          1] [1          2  ...          2] [2          3  ...          2] [3          4
test = pd.read_csv('test.tsv', sep='\t')  test: PhraseId  SentenceId                                          Phrase [0       156061       8545  An intermittently pleasing but mostly routine ...]
g = open('train.tsv','r')  g: <_io.TextIOWrapper name='train.tsv' mode='r' encoding='cp936'>
train = list(map(lambda x: x[:-1], g.readlines()))
g.close()
g = open('test.tsv','r')
test = list(map(lambda x: x[:-1], g.readlines()))
g.close()
linha_1 = train[1]
lista_teste = linha_1.split('\t')
train_lines = []
for i in range(len(train)):
    train_lines.append(train[i])
train_set_lines = []
for i in range(len(train_lines)):
    dados_linha = train_lines[i]
    dados_linha = dados_linha.split('\t')
```

```
n_and_test
控制台
Thread
ule>, train_and_test.py:12
评估表达式(Enter)或添加监视(Ctrl+Shift+Enter)
> g = {TextIOWrapper} <_io.TextIOWrapper name='train.tsv' mode='r' encoding='cp936'>
> test = {DataFrame: (66292, 3)} PhraseId  SentenceId          Phrase [0     156061     8545  An intermittently pleasing but mostly routine ...] [1     156062     8545  An intermittently pleasing but mostly routine
> train = {DataFrame: (156060, 4)} PhraseId  ...  Sentiment [0      1 ...     1] [1      2 ...     2] [2      3 ...     2] [3      4 ...     2] [4      5 ...     2] [...     ... ...     ...] [156055   156056 ...     2] [1
> 特殊变量
```

1.4.1实验步骤

1， 准备数据集
2， 数据预处理
3， 将数据集分割为训练集、测试集、验证集
4， 搭建网络模型
5， 确定损失函数、优化器
6， 训练

7， 测试

1.4.2 实验调试及心得

根据错误提示信息及业务功能逻辑的推理等相关信息找到有可能会导致错误发生的位置。日志信息的输出位置一是控制台输出，这时最及时的一种方式，在程序运行时可以很容易就看到日志输出信息。二是输出日志信息到日志文件中，这种方式根据项目要求配置日志相关信息。

当程序运行到断点，这时程序在一个暂停状态，我们需要进行程序单步向下执行，观察每一步中的输入参数数据是否正确或调用方法的返回值数据是否正确，当程序运行到一个方法时我们可以继续单步向下执行不管方法内部的逻辑只关心返回值，也可以使用程序调试方法中的进入方法内部继续追踪程序（进入方法内部后继续单步执行或跳出方法内部），直到程序执行发生错误，判断是否为程序错误发生的实际位置。

另一种情况当程序运行起来后，业务处理发生错误，并没有进入我们的断点或打印输出语句，这时就是我们推测的程序发生的位置不对，需要根据错误信息和业务处理流程逻辑重新推测错误发生位置，重新再次进行第 2 步。

附录 实验代码

```
import numpy as np
import pandas as pd
import re
from collections import Counter
import matplotlib.pyplot as plt
import torch
from torch.utils.data import TensorDataset, DataLoader
import torch.nn as nn
train = pd.read_csv('train.tsv', sep='\t')
test = pd.read_csv('test.tsv', sep='\t')
g = open('train.tsv','r')
train = list(map(lambda x: x[:-1], g.readlines()))
g.close()
g = open('test.tsv','r')
test = list(map(lambda x: x[:-1], g.readlines()))
g.close()
linha_1 = train[1]
lista_teste = linha_1.split('\t')
train_lines = []
for i in range(len(train)):
    train_lines.append(train[i])
train_set_lines = []
for i in range(len(train_lines)):
    dados_linha = train_lines[i]
    dados_linha = dados_linha.split('\t')
    train_set_lines.append(dados_linha)
test_lines = []
for i in range(len(test)):
    test_lines.append(test[i])
test_set_lines = []
for i in range(len(test_lines)):
    dados_linha = test_lines[i]
    dados_linha = dados_linha.split('\t')
    test_set_lines.append(dados_linha)
for i in range(1, len(train_set_lines)):
    train_set_lines[i][3] = int(train_set_lines[i][3])

for i in range(len(train_set_lines)):
    train_set_lines[i][2] = train_set_lines[i][2].lower()
    string = train_set_lines[i][2]
    train_set_lines[i][2] = re.sub('[^a-z0-9\s]', '', string)
for i in range(len(test_set_lines)):
    test_set_lines[i][2] = test_set_lines[i][2].lower()
    string = test_set_lines[i][2]
    test_set_lines[i][2] = re.sub('[^a-z0-9\s]', '', string)

all_text = []
for i in range(1,len(train_set_lines)):
    all_text.append(train_set_lines[i][2])
for i in range(1,len(test_set_lines)):
    all_text.append(test_set_lines[i][2])
all_text = ' '.join(all_text)
words = all_text.split()
counts = Counter(words)
vocabulary = sorted(counts, key=counts.get, reverse=True)
vocabulary_to_int = {}
int_to_vocabulary = {}
for i, word in enumerate(vocabulary, 1):
    vocabulary_to_int[word] = i
    int_to_vocabulary[i] = word

train_set_int = []
for review in train_set_lines:
    if review[0] == 'PhraseId':
        continue
    train_set_int.append([vocabulary_to_int[word] for word in
review[2].split()])
```

```
encoded_labels = np.array([train_set_lines[i][3] for i in
range(1,len(train_set_lines))])

test_set_int = []
for review in test_set_lines:
    if review[0] == 'PhraseId':
        continue
    test_set_int.append([vocabulary_to_int[word] for word in
review[2].split()])
review_49 = train_set_int[49] # review em formato token
review_49_words = ' '.join([int_to_vocabulary[i] for i in review_49]) #
refazendo a conversão

tamanho_reviews = Counter([len(x) for x in train_set_int]) # Dict: {objeto:
contagem}
tamanho_reviews_test = Counter([len(x) for x in test_set_int])
tamanho = sorted([key for key in tamanho_reviews.keys()])
numero = [tamanho_reviews[lenght] for lenght in tamanho]
plt.plot(tamanho, numero)
plt.xlabel('size')
plt.ylabel('quantity')
plt.show()

zero_idx_train = [i for i, review in enumerate(train_set_int) if
len(review) == 0]
encoded_labels_null = [encoded_labels[i] for i in zero_idx_train]
count_label_null = Counter(encoded_labels_null)
x_data = [0,1,2,3,4]
y_data = [0,1,26,0,0]
plt.plot(x_data, y_data)
plt.show()

non_zero_idx_train = [i for i, review in enumerate(train_set_int) if
len(review) != 0]
non_zero_idx_test = [i for i, review in enumerate(test_set_int) if
len(review) != 0]
train_set_int = [train_set_int[i] for i in non_zero_idx_train]
encoded_labels = np.array([encoded_labels[i] for i in
non_zero_idx_train])
test_set_int = [test_set_int[i] for i in non_zero_idx_test]
def padding_features(review_ints, seq_length):
    features = np.zeros((len(review_ints), seq_length), dtype=int)
    for i, row in enumerate(review_ints):
        features[i, -len(row):] = np.array(row[:seq_length])
    return features
seq_length = 52
features_train = padding_features(train_set_int, seq_length=seq_length)
features_test = padding_features(test_set_int, seq_length=seq_length)

split_frac = 0.8
split_idx = int(len(features_train)*split_frac)
train_x, remaining_x = features_train[:split_idx],
features_train[split_idx:]
train_y, remaining_y = encoded_labels[:split_idx],
encoded_labels[split_idx:]
test_idx = int(len(remaining_x)*0.5)
val_x, test_x = remaining_x[:test_idx], remaining_x[test_idx:]
val_y, test_y = remaining_y[:test_idx], remaining_y[test_idx:]

train_data = TensorDataset(torch.from_numpy(train_x),
torch.from_numpy(train_y))
valid_data = TensorDataset(torch.from_numpy(val_x),
torch.from_numpy(val_y))
test_data = TensorDataset(torch.from_numpy(test_x),
torch.from_numpy(test_y))

batch_size = 128
train_loader = DataLoader(train_data, shuffle=True,
batch_size=batch_size, drop_last=True)
valid_loader = DataLoader(valid_data, shuffle=True,
batch_size=batch_size, drop_last=True)
test_loader = DataLoader(test_data, shuffle=True, batch_size=batch_size,
drop_last=True)

dataiter = iter(train_loader)
sample_x, sample_y = dataiter.__next__()
train_on_gpu=torch.cuda.is_available()

if(train_on_gpu):
    print('Cheating on GPU.')
else:
    print('GPU not available, training on the CPU.')
class SentimentRNN(nn.Module):
    """
    Modelo de uma rede recorrente usada para realizar analise de
sentimentos.
    """

    def __init__(self, vocab_size, output_size, embedding_dim,
hidden_dim, n_layers, drop_prob=0.5):
        # Aqui vamos inicializar a rede e os seus parâmetros

        super(SentimentRNN, self).__init__()

        self.output_size = output_size
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim

        # Camadas de embedding e da LSTM
        # Na LSTM, deve-se colocar batch_first pois os tensores
        # de input e output tem a forma: (batch, seq, feature)
        self.embedding = nn.Embedding(vocab_size, embedding_dim)
        self.lstm = nn.LSTM(embedding_dim, hidden_dim, n_layers,
                            dropout=drop_prob, batch_first=True)

        # Dropout
        self.dropout = nn.Dropout(p=0.3)

        # Camada linear, dropout e softmax
        self.fc = nn.Linear(hidden_dim, output_size)
        self.log_softmax = nn.LogSoftmax(dim=1)

    def forward(self, x, hidden):
        # Essa função é para realizar o forward pass do modelo no input
e nos hidden states

        batch_size = x.size(0)

        # Embeddings e lstm_out
        embeds = self.embedding(x)
        lstm_out, hidden = self.lstm(embeds, hidden)

        # Juntando os outputs da LSTM
        lstm_out = lstm_out.contiguous().view(-1, self.hidden_dim)

        # Dropout e a camada totalmente conectada
        out = self.dropout(lstm_out)
        out = self.fc(out)
        # Função LogSoftmax
        soft_out = self.log_softmax(out)

        # Reshape para o batchfirst
        soft_out = soft_out.view(batch_size, -1, self.output_size)
        soft_out = soft_out[:, -1, :]  # pega as labels(o output) da ultima
posiçao te todas as linhas

        # Retorna o último output da softmax e do hidden state
        return soft_out, hidden

    def init_hidden(self, batch_size):
        # Essa função inicializa o hidden state

        # Cria dois novos tensores de tamanho n_layers x batch_size x
hidden_dim,
        # inicializados com zeros, pra o hidden state e a cell state da
LSTM
        weight = next(self.parameters()).detach()

        if (train_on_gpu):
            hidden = (weight.new(self.n_layers, batch_size,
self.hidden_dim).zero_().cuda(),
                      weight.new(self.n_layers, batch_size,
self.hidden_dim).zero_().cuda())
        else:
            hidden = (weight.new(self.n_layers, batch_size,
self.hidden_dim).zero_(),
                      weight.new(self.n_layers, batch_size,
self.hidden_dim).zero_())

        return hidden

vocab_size = len(vocabulary_to_int) + 1 # +1 pelo padding de 0's
output_size = 5
embedding_dim = 200
hidden_dim = 256
n_layers = 2
net = SentimentRNN(vocab_size, output_size, embedding_dim, hidden_dim,
n_layers)

print(net)
lr = 0.001
criterion = nn.NLLLoss()
optimizer = torch.optim.Adam(net.parameters(), lr=lr)
epochs = 4
counter = 0
print_every = 100
```

```
clip = 5
if(train_on_gpu):
    net.cuda()
net.train()
for e in range(epochs):
    h = net.init_hidden(batch_size)
    losses = []
    for inputs, labels in train_loader:
        if ((inputs.shape[0], inputs.shape[1]) != (batch_size,
seq_length)):
            continue
        counter += 1
        if(train_on_gpu):
           inputs, labels = inputs.cuda(), labels.cuda()
        h = tuple([each.detach() for each in h])
        net.zero_grad()
        output, h = net(inputs, h)
        loss = criterion(output.squeeze(), labels.long())
        losses.append(loss.item())
        loss.backward()
        nn.utils.clip_grad_norm_(net.parameters(), clip)
        optimizer.step()
        if counter % print_every == 0:
            val_h = net.init_hidden(batch_size)
            val_losses = []
            net.eval()
            for inputs, labels in valid_loader:
                val_h = tuple([each.detach() for each in val_h])
                if(train_on_gpu):
                   inputs, labels = inputs.cuda(), labels.cuda()
                output, val_h = net(inputs, val_h)
                val_loss = criterion(output.squeeze(), labels.long())
                val_losses.append(val_loss.item())
            net.train()
            print("Epoch: {}/{}...".format(e + 1, epochs),
                  "Step: {}...".format(counter),
                  "Loss: {:.6f}...".format(loss.item()),
                  "Val Loss: {:.6f}".format(np.mean(val_losses)))
plt.plot(val_losses)
plt.show()
torch.save(net.state_dict(), 'sentiment_net.pt')
net.load_state_dict(torch.load('sentiment_net.pt'))

test_loss = 0.0
```

```python
class_correct = list(0. for i in range(5))
class_total = list(0. for i in range(5))
h = net.init_hidden(batch_size)
net.eval()
for inputs, labels in test_loader:
    h = tuple([each.data for each in h])
    if(train_on_gpu):
        inputs, labels = inputs.cuda(), labels.cuda()
    output, h = net(inputs, h)
    loss = criterion(output, labels.long())
    test_loss += loss.item() * inputs.size(0)
    _, pred = torch.max(output, 1)
    correct = np.squeeze(pred.eq(labels.data.view_as(pred)))
    for i in range(len(labels)):
        label = labels.data[i]
        class_correct[label] += correct[i].item()
        class_total[label] += 1
test_loss = test_loss / len(test_loader.sampler)
print('Test Loss: {:.6f}\n'.format(test_loss))
for i in range(5):
    if class_total[i] > 0:
        print('Test Accuracy of %5s: %2d%% (%2d/%2d)' % (
            str(i), 100 * class_correct[i] / class_total[i],
            np.sum(class_correct[i]), np.sum(class_total[i])))
    else:
        print('Test Accuracy of %5s: N/A (no training examples)' %
(class_total[i]))
print('\nTest Accuracy (Overall): %2d%% (%2d/%2d)' % (100. *
np.sum(class_correct) / np.sum(class_total), np.sum(class_correct),
np.sum(class_total)))
```