**《机器学习理论及应用》课程**

**实验报告**

学号：04021503

姓名：江梓贤

2023年 11月20日

**实验一：支持向量机**

**实验要求：**采用支持向量机(SVM)完成手写数字识别数据集MNIST的分类任务。MNIST数据集是机器学习领域中非常经典的一个数据集，由60000个训练样本和10000个测试样本组成，每个样本都是一张28 \* 28像素的灰度手写数字图片，总共包含四个部分：

* Training set images: train-images-idx3-ubyte.gz (9.9 MB, 解压后 47 MB, 包含 60,000 个样本)
* Training set labels: train-labels-idx1-ubyte.gz (29 KB, 解压后 60 KB, 包含 60,000 个标签)
* Test set images: t10k-images-idx3-ubyte.gz (1.6 MB, 解压后 7.8 MB, 包含 10,000 个样本)
* Test set labels: t10k-labels-idx1-ubyte.gz (5KB, 解压后 10 KB, 包含 10,000 个标签)

数据下载：

（1）官方地址：<http://yann.lecun.com/exdb/mnist/> （现需要身份验证，无法下载）

（2）tensorflow直接加载: (train\_images, train\_labels), (test\_images, test\_labels) = tf.keras.datasets.mnist.load\_data()

（3）pytorch采用torchvision.datasets模块下的MNIST类下载数据集，指定下载路径和转换方式：

trainset = torchvision.datasets.MNIST(root='./data', train=True, download = True, transform = torchvision.transforms.ToTensor())

testset = torchvision.datasets.MNIST(root='./data', train=False, download = True, transform= torchvision.transforms.ToTensor())

（4）sklearn加载数据集：

from sklearn.datasets import fetch\_openml

mnist = fetch\_openml("mnist\_784")

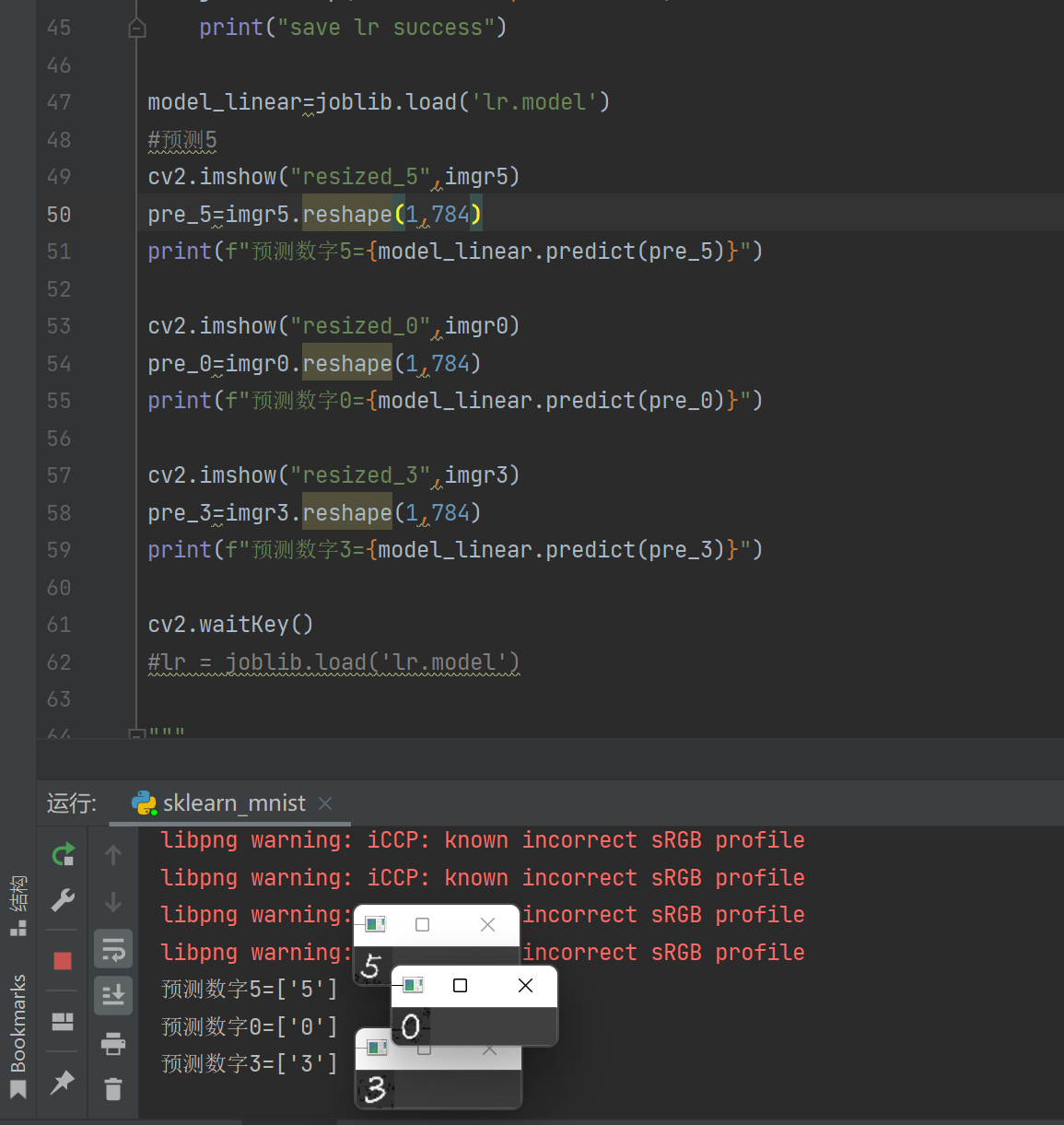
data = mnist['data']

label = mnist['target']

**一、实验内容**

完成SVM对MNIST数据集识别分类。注意给出实验代码和详细过程介绍（包括过程截图）。**并对学号后三位进行识别给出结果**。

（格式：正文12号字体）

from sklearn.datasets import fetch\_openml  
from sklearn import svm  
import joblib  
import numpy as np  
import cv2  
import warnings  
warnings.filterwarnings("ignore")  
#以下为用opencv读取我手写的图片  
img5=cv2.imread("../5.png")  
imgr5=cv2.resize(img5,(28,28),interpolation= cv2.INTER\_LINEAR)  
imgr5=cv2.split(imgr5)  
imgr5=imgr5[0]  
  
img0=cv2.imread("../0.png")  
imgr0=cv2.resize(img0,(28,28),interpolation= cv2.INTER\_LINEAR)  
imgr0=cv2.split(imgr0)  
imgr0=imgr0[0]  
  
img3=cv2.imread("../3.png")  
imgr3=cv2.resize(img3,(28,28),interpolation= cv2.INTER\_LINEAR)  
imgr3=cv2.split(imgr3)  
imgr3=imgr3[0]  
  
load=False  
#lr是一个LogisticRegression模型  
#是否加载训练集  
if(load):  
 mnist = fetch\_openml("mnist\_784")  
 data = mnist['data'][:2500]  
 label = mnist['target'][:2500]  
 test\_data=mnist['data'][:-500]  
 test\_label = mnist['target'][:-500]  
 print(type(data))  
 print(len(data))  
 print(test\_data[:1])  
#线性平面分割  
model\_linear = svm.SVC(kernel='linear', C = 0.001, decision\_function\_shape='ovr')  
#是否训练  
train=False  
if(train and load):  
 # 训练线性内核  
 print("start training lr")  
 model\_linear.fit(data, label) # 训练  
 accuracy = model\_linear.score(test\_data, test\_label)#训练打分  
 print(accuracy)  
 joblib.dump(model\_linear, 'lr.model')#joblib保存训练模型  
 print("save lr success")  
  
model\_linear=joblib.load('lr.model')#joblib读取训练好的模型  
#预测5，0，3 三个数字  
cv2.imshow("resized\_5",imgr5)  
pre\_5=imgr5.reshape(1,784)  
print(f"预测数字5={model\_linear.predict(pre\_5)}")  
  
cv2.imshow("resized\_0",imgr0)  
pre\_0=imgr0.reshape(1,784)  
print(f"预测数字0={model\_linear.predict(pre\_0)}")  
  
cv2.imshow("resized\_3",imgr3)  
pre\_3=imgr3.reshape(1,784)  
print(f"预测数字3={model\_linear.predict(pre\_3)}")  
#可以看到503预测正确  
cv2.waitKey()  
  


可以看到训练后的线性内核模型准确率大概为0.901，对我手写的三位学号的预测准确率为100%

**二、提高训练**

尝试对现有实验做修改和调整，例如正则项、核函数等。比较不同参数带来的性能影响，分析猜测其背后的原因。

#训练sigmoid内核 结果与下两种相似  
model\_poly=svm.SVC(kernel='sigmoid',C=0.001,gamma=0.05,decision\_function\_shape='ovr')  
#训练多项式内核  
print("start training sd")  
model\_poly.fit(data, label) # 训练  
accuracy=model\_poly.score(test\_data,test\_label)  
print(accuracy)  
joblib.dump(model\_poly, 'sd.model')  
print("save sd success")

（格式：正文12号字体）

#通过训练可以看出非线性核带来的效果很差0.09746762589928057  
"""  
model\_poly=svm.SVC(kernel='poly',C=0.001,degree=2,gamma=0.05,decision\_function\_shape='ovr')

#训练多项式内核  
print("start training pl")  
model\_poly.fit(data, label) # 训练  
accuracy=model\_poly.score(test\_data,test\_label)  
print(accuracy)  
joblib.dump(model\_poly, 'pl.model')  
print("save pl success")  
  
model\_rbf=svm.SVC(kernel='rbf',C=0.001,decision\_function\_shape='ovr')  
#训练多项式内核  
print("start training rbf")  
model\_rbf.fit(data, label) # 训练  
accuracy=model\_rbf.score(test\_data,test\_label)  
print(accuracy)  
joblib.dump(model\_rbf, 'rbf.model')  
print("save rbf success")  
"""

**三、实验心得**

实验过程中遇到的什么问题。尝试使用什么方法去解决。通过实验获得了什么感悟与理解。

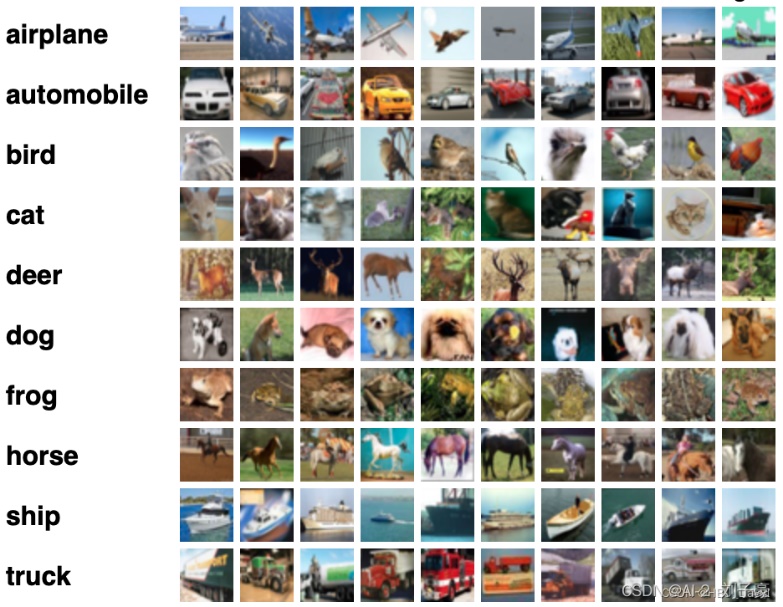
（格式：正文12号字体）

训练集样本数量过大，我通过选取部分样本训练并逐渐扩大训练集的容量直到筛选出训练速度最快且正确率较为理想的训练结果。

训练集采用Pandas data Frame’的数据结构，但之后发现在输入自己的图片时可以通过opencv转化图片为numpy.ndarray 的数据格式同样可以进行预测，所以调用库时需要了解数据接口。

**实验二：卷积神经网络**

**实验要求：**采用卷积神经网络(CNN)完成对CIFAR-10 数据集的分类任务，通过给定的衣服图片数据集训练神经网络模型，通过CNN架构提取图片特征，实现模型对于各类衣服的识别和分类。CIFAR-10数据集包含了10种不同的类别，分别是('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')，共60,000张图像，其中每个类别的图像都是6000张，图像大小均为32×32像素，因此输入为(32, 32, 3)的彩色RGB图片，可以根据内存或显存自由设定和划分训练集、测试集。CIFAR-10数据集的示例如图所示。



数据下载：

（1）链接下载：

http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz

（2）pytorch采用torchvision.datasets模块下的FashionMNIST类下载数据集，指定下载路径和转换方式：

mnist\_train = torchvision.datasets.CIFAR10(root = './FashionMNIST', train = True, download = True, transform=transforms.ToTensor())

mnist\_test = torchvision.datasets.CIFAR10(root = './FashionMNIST', train = False, download = True, transform=transforms.ToTensor())

（3）tensorflow直接加载：

(x, y), (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

（4）腾讯微云

链接：<https://share.weiyun.com/56FKfYz>

密码：nwdmtc

**一、实验内容**

完成CNN对CIFAR-10数据集识别分类。注意给出实验代码和详细过程介绍（包括过程截图）。

（格式：正文12号字体）

import torch  
import torchvision  
import torchvision.transforms as transforms  
import wandb as wandb  
from torch import nn  
from torch.utils import data  
from tqdm import tqdm,trange  
  
  
trainset = torchvision.datasets.CIFAR10(root = './FashionMNIST', train = True,  
 download = True, transform=transforms.ToTensor())  
trainloader = torch.utils.data.DataLoader(trainset, batch\_size=128,  
 shuffle=True, num\_workers=0, pin\_memory=True) # 打乱，包装成batchsize  
  
testset = torchvision.datasets.CIFAR10(root = './FashionMNIST',  
 train = False, download = True, transform=transforms.ToTensor())  
testloader = torch.utils.data.DataLoader(testset, batch\_size=128,  
 shuffle=False, num\_workers=0, pin\_memory=True)  
device = torch.device("cuda:0") # 选择cpu或者GPU  
  
class CNN(nn.Module):  
 def \_\_init\_\_(self):  
 super(CNN, self).\_\_init\_\_()  
 self.conv1 = nn.Sequential(  
 nn.Conv2d(3, 8, 3, 1, 1),  
 nn.ReLU(),  
 nn.MaxPool2d(2)  
 )  
 self.conv2 = nn.Sequential(  
 nn.Conv2d(8, 16, 3, 1, 1),  
 nn.ReLU(),  
 nn.MaxPool2d(2)  
 )  
 self.conv3 = nn.Sequential(  
 nn.Conv2d(16, 32, 3, 1, 1),  
 nn.ReLU(),  
 nn.MaxPool2d(2)  
 )  
 self.conv4 = nn.Sequential(  
 nn.Conv2d(32, 64, 3, 1, 1),  
 nn.ReLU(),  
 nn.MaxPool2d(2)  
 )  
 self.fc1 = nn.Sequential(  
 nn.Linear(64 \* 2 \* 2, 120),  
 nn.ReLU()  
 )  
 self.fc2 = nn.Sequential(  
 nn.Linear(120, 84),  
 nn.ReLU()  
 )  
 self.fc3 = nn.Linear(84, 10)  
 def forward(self, x):  
 x = self.conv1(x)  
 x = self.conv2(x)  
 x = self.conv3(x)  
 x = self.conv4(x)  
 x = x.view(-1, 64 \* 2 \* 2)  
 x = self.fc1(x)  
 x = self.fc2(x)  
 x = self.fc3(x)  
 return x  
  
  
model = CNN()  
model.to(device) # 选择cpu或者gpu  
model.load\_state\_dict(torch.load('CNN\_CIFAR10.pkl'))  
print("load successfully")  
criterion = nn.CrossEntropyLoss() # 交叉熵损失  
optimizer = torch.optim.SGD([{'params': model.parameters()}],  
 lr=0.005, weight\_decay=5e-4, momentum=0.9) # 随机梯度优化策略  
  
  
proc\_bar=tqdm(total=100)  
for epoch in range(80):  
 # 训练  
 model.train()  
 #for i, (img, label) in tqdm(enumerate(trainloader)):  
 # 测试  
 accuracy = 0  
 model.eval()  
 testlen = 0  
  
  
 #for img, label in tqdm(trainloader,dynamic\_ncols=True):  
 for img, label in (trainloader):  
 img, label = img.to(device, non\_blocking=True), label.to(device, non\_blocking=True).long() # 加载数据  
 #print(img.size())  
 output = model(img) # 计算结果  
 loss = criterion(output, label) # 计算损失  
 optimizer.zero\_grad()  
 loss.backward() # 反向传播  
 optimizer.step() # 优化器更新  
  
 #iters = epoch \* len(trainloader) + i  
 #if iters % 10 == 0:  
 # wandb.log({'loss': loss}) # 可视化  
 for i, (img, label) in enumerate(testloader):  
 img, label = img.to(device), label.to(device).long()  
 output = model(img)  
 output = output.max(dim=1)[1] # 预测是哪一类  
 accuracy += (output == label).sum().item() # 准确率计算  
 testlen += len(output)  
 accuracy = accuracy / testlen # 准确率计算  
 proc\_bar.update(100 / 80)  
 proc\_bar.set\_description(f"epoch={epoch:02d},accuracy={accuracy:02f}")  
 # wandb.log({'accuracy': accuracy}) # 可视化  
proc\_bar.close()  
  
torch.save(model.state\_dict(), 'CNN\_CIFAR10.pkl')#保存模型  
print("save successfully")

**二、提高训练**

尝试对现有实验做修改和调整，例如损失函数、激活函数等。比较不同神经网络带来的性能影响，分析猜测其背后的原因。

（格式：正文12号字体）

上述代码中我使用了普通多层cnn网络

之后我尝试使用了cv论文中介绍的其他类型的卷积神经网络，如AlexNet：

class AlexNet(nn.Module):  
 def \_\_init\_\_(self):  
 super(AlexNet, self).\_\_init\_\_()  
 self.conv = nn.Sequential(  
 nn.Conv2d(3, 96, 6, 4), # in\_channels, out\_channels, kernel\_size, stride, padding  
 nn.ReLU(),  
 nn.MaxPool2d(3, 2), # kernel\_size, stride  
 # 减小卷积窗口，使用填充为2来使得输入与输出的高和宽一致，且增大输出通道数  
 nn.Conv2d(96, 256, 5, 1, 2),  
 nn.ReLU(),  
 nn.MaxPool2d(3, 2),  
 # 连续3个卷积层，且使用更小的卷积窗口。除了最后的卷积层外，进一步增大了输出通道数。  
 # 前两个卷积层后不使用池化层来减小输入的高和宽  
 nn.Conv2d(256, 384, 3, 1, 2),  
 nn.ReLU(),  
 nn.Conv2d(384, 384, 3, 1, 2),  
 nn.ReLU(),  
 nn.Conv2d(384, 256, 3, 1, 2),  
 nn.ReLU(),  
 nn.MaxPool2d(3, 2)  
 )  
 # 这里全连接层的输出个数比LeNet中的大数倍。使用丢弃层来缓解过拟合  
 self.fc = nn.Sequential(  
 nn.Linear(256 \* 3 \* 3, 4096),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 nn.Linear(4096, 4096),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 # 输出层。由于这里使用Fashion-MNIST，所以用类别数为10，而非论文中的1000  
 nn.Linear(4096, 10),  
 )  
 def forward(self, img):  
 feature = self.conv(img)  
 output = self.fc(feature.view(img.shape[0], -1))  
 return output

以及较为热门的残差网络ResNet：

class Resblock(nn.Module):  
 def \_\_init\_\_(self,inchannels,downsample):  
 super(Resblock, self).\_\_init\_\_()  
 self.conv1=nn.Sequential(  
 nn.Conv2d(inchannels,32,5,1),  
 nn.BatchNorm2d(32),  
 nn.ReLU(),  
 #nn.MaxPool2d(2,2),  
 nn.Conv2d(32,96,3,1,2),  
 nn.BatchNorm2d(96),  
 nn.ReLU(),  
 #nn.MaxPool2d(2, 2)  
 nn.Conv2d(96, inchannels, 3, 1, 2),  
 )  
 self.downsample = downsample  
 self.relu = nn.ReLU(inplace=True)  
  
 def forward(self,x):  
 residual=x  
 if self.downsample is not None:  
 residual = self.downsample(x)  
 #residual = resize(residual)  
 y=self.conv1(x)  
 y+=residual  
 output=self.relu(y)  
 return output  
  
class ResNet(nn.Module):  
 def \_\_init\_\_(self,inchannels,downsample):  
 super(ResNet, self).\_\_init\_\_()  
 self.rb1=Resblock(inchannels,downsample)  
 self.rb2 = Resblock(inchannels, downsample)  
 self.rb3 = Resblock(inchannels, downsample)  
 self.conv=nn.Sequential(  
 nn.Conv2d(inchannels, 32, 5, 1),  
 nn.BatchNorm2d(32),  
 nn.ReLU(),  
 nn.MaxPool2d(2,2)  
 )  
 self.fc = nn.Sequential(  
 nn.Linear(32 \* 14 \*14, 4096),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 nn.Linear(4096, 4096),  
 nn.ReLU(),  
 nn.Dropout(0.5),  
 # 输出层。由于这里使用Fashion-MNIST，所以用类别数为10，而非论文中的1000  
 nn.Linear(4096, 10),  
 )  
 def forward(self,x):  
 y=self.rb1(x)  
 y=self.rb2(y)  
 y=self.rb3(y)  
 output=self.conv(y)  
 output = self.fc(output.view(output.shape[0], -1))  
 return output

**三、实验心得**

实验过程中遇到的什么问题。尝试使用什么方法去解决。通过实验获得了什么感悟与理解。

（格式：正文12号字体）

实验过程中我发现训练速度极其缓慢，原因是我的pytorch在一年前安装时没有仔细看版本导致安装成了cpu版本，于是我重新下载了适合用于我的cuda11.6的cuda116 pytorch，gpu大大加快了训练速度。

并且我尝试使用tqdm进度条展示我的训练过程

我开始时的cnn深度不够导致训练效果不够理想，之后逐渐加深网络结构，使其能够提取复杂多变的图像特征

并且直接使用训练集训练存在过拟合现象或者数据集太特殊化往往需要数据增强：

data\_transforms = [

# 1. 随机水平翻转

transforms.Compose([transforms.RandomHorizontalFlip(p=1)]),

# 2. 随机旋转

transforms.Compose([transforms.RandomRotation(10)]),

# 3. 随机裁剪

transforms.Compose([transforms.RandomCrop(32, padding=4)]),

# 4. 颜色调整

transforms.Compose([transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1)]),

# 5. 随机灰度化

transforms.Compose([transforms.RandomGrayscale(p=0.1)]),

# 6. 随机调整大小后裁剪

transforms.Compose([transforms.RandomResizedCrop(32, scale=(0.8, 1.0))]),

# 7. 高斯模糊

transforms.Compose([transforms.GaussianBlur(kernel\_size=5)]),

# 8. 随机仿射变换（包括旋转和剪切）

transforms.Compose([transforms.RandomAffine(degrees=15, translate=(0.1, 0.1), scale=(0.8, 1.2), shear=10)]),

# 9. 指定剪切角度的剪切变换

transforms.Compose([transforms.RandomAffine(degrees=0, translate=(0.1, 0.1), scale=(0.8, 1.2), shear=10)]),

# 10. 透视变换

transforms.Compose([transforms.RandomPerspective(distortion\_scale=0.5)]),

]

以及可以分层训练神经网络：

for k,v in model.named\_parameters():  
 if k.find('fc'):  
 v.requires\_grad=False#固定连接层参数  
#两种层最多固定一个  
for name, param in model.named\_parameters():  
 if param.requires\_grad:  
 print(name)

通过分离卷积层和全连接层可以带来小幅度的训练准确度提升

**实验三：循环神经网络**

**实验要求：**采用循环神经网络(RNN)完成对电影评论数据集的情感分类任务。IMDB影评数据集是用于二元情感分类的数据集，在NLP任务中经常被使用。有评级标签的IMDB数据集包含5万篇IMDB影评，负面评论得分<=4，正面评论得分>=7，其中2.5万篇影评用于训练数据集，2.5万篇影评用于用于测试数据集，训练集和测试集电影不相交。此外，还有另外5万篇没有任何评级标签的IMDB影评，用于无监督学习。

结构：

aclImdb

├── test

│ ├── neg

│ └── pos

└── train

├── neg

└── pos

数据下载：

1. 网站下载：

<http://ai.stanford.edu/~amaas/data/sentiment/>

（2）torchtext下载

from torchtext.datasets import IMDB

train\_iter, test\_iter = IMDB()

（3）tensorflow加载 (已经过预处理，评论（单词序列）已经被转换为整数序列，其中每个整数代表字典中的某个单词)

(train\_data, train\_labels), (test\_data, test\_labels) = tf.keras.datasets.imdb.load\_data(num\_words=10000)

（4）百度网盘

链接：https://pan.baidu.com/s/1-2a7QwTzcRonO3pG3DZXbw

密码：yyds

**一、实验内容**

完成RNN对电影评论数据集的情感二分类分类处理。难点在于如何完成文本数据的预处理，需要进行特征提取，一种特征提取方法是词频-逆向文件频率(TF-IDF)，主要思想是如果某一个词或短语在一篇文章中出现的频率高，并且在其他文章中很少出现，则认为此词或短语具有很好的类别区分能力，时适合用来分类。在scikit-learn中提供了TF-IDF来进行文本特征提取的实现，可通过from sklearn.feature\_extraction.text import TfidfVectorizer调用。此外还有词袋模型、词嵌入(Word Embdding)等方法。注意给出实验代码和详细过程介绍（包括过程截图）。

（格式：正文12号字体）

# 获取file\_path路径下的所有TXT文本内容和文件名

使用RNN单字符编码进行训练使用embedding进行词嵌入进行编码  
import os  
import re  
import torch  
from torch import nn  
import jieba  
  
special\_chars="<br /><br />"  
def get\_txts(file\_path):  
 files = os.listdir(file\_path)  
 txt\_list = []  
 pattern = r'[a-zA-Z]'  
 for file in files:  
 with open(os.path.join(file\_path, file), "r", encoding="UTF-8") as f:  
 for l in f:  
 l.strip()  
 l=l.lower()  
 l=l.replace(special\_chars,"")  
 l.split(' ')  
 l = re.sub(r'[^\w\s]', '', l)  
 words=re.split(r'\b',l)  
 for word in words:  
 result = re.search(pattern, word)  
 if result:  
 txt\_list.append(word)  
 return txt\_list, files  
  
pos\_file\_path="F:/RIS\_NOMA\_UAV\_RL\_Pytorch/aclImdb/train/pos"  
poslist,posfile=get\_txts(pos\_file\_path)  
#print(poslist[0:100])  
  
neg\_file\_path="F:/RIS\_NOMA\_UAV\_RL\_Pytorch/aclImdb/train/neg"  
neglist,negfile=get\_txts(neg\_file\_path)  
#print(neglist[0:50])  
  
word\_set=set()  
for word in poslist:  
 word\_set.add(word)  
for word in neglist:  
 word\_set.add(word)  
#print(word\_set)  
  
  
wordlist=list(word\_set)  
n\_words=len(wordlist)+1  
#词嵌入  
def comment\_to\_tensor(comment):  
 tensor=torch.zeros(len(comment),dtype=torch.long)  
 for li,ch in enumerate(comment):  
 try:  
 ind=wordlist.index(ch)  
 except ValueError:  
 ind=n\_words-1  
 tensor[li]=ind  
 return tensor  
embedding=nn.Embedding(n\_words,100)  
  
"""  
one-hot 编码  
def comment\_to\_tensor(comment):  
 tensor=torch.zeros(len(comment),1,n\_words)  
 for li,ch in enumerate(comment):  
 try:  
 ind=wordlist.index(ch)  
 except ValueError:  
 ind =n\_words-1  
 tensor[li][0][ind]=1  
 return tensor  
"""  
#print(comment\_to\_tensor(poslist[0:50]))  
  
#非调库写法  
class RNN(nn.Module):  
 def \_\_init\_\_(self,word\_count,embedding\_size,hidden\_size,output\_size):  
 super(RNN, self).\_\_init\_\_()  
 self.hidden\_size=hidden\_size  
 self.embedding=nn.Embedding(word\_count,embedding\_size)  
 self.i2s=nn.Linear(embedding\_size+hidden\_size,hidden\_size)  
 self.i2o=nn.Linear(embedding\_size+hidden\_size,output\_size)  
 self.softmax=nn.LogSoftmax(dim=0)  
 def forward(self,input\_tensor,hidden):  
 word\_vector=self.embedding(input\_tensor)  
 combined=torch.cat((word\_vector,hidden),1)  
 hidden=self.i2s(combined)  
 output=self.i2o(combined)  
 output=self.softmax(output)  
 return output,hidden  
 def inithidden(self):  
 return torch.zeros(1,self.hidden\_size)  
  
  
def run\_rnn(rnn,input\_tensor):  
 hidden=rnn.inithidden()  
 for i in range(input\_tensor.size()[0]):  
 output,hidden=rnn(input\_tensor[i].unsqueeze(dim=0),hidden)  
 return output  
  
embedding\_size=200  
n\_hidden=128  
n\_categories=2  
rnn=RNN(n\_words,embedding\_size,n\_hidden,n\_categories)  
"""  
#调库写法：  
class RNN(nn.Module):  
 def \_\_init\_\_(self,word\_count,embedding\_size,hidden\_size,output\_size):  
 super(RNN, self).\_\_init\_\_()  
 self.hidden\_size=hidden\_size  
 self.embedding=nn.Embedding(word\_count,embedding\_size)  
 self.rnn=nn.RNN(embedding\_size,hidden\_size,num\_layers=1,bidirectional=False,batch\_first=True)  
 self.cls=nn.Linear(hidden\_size,output\_size)  
 self.softmax=nn.LogSoftmax(dim=0)  
  
 def forward(self,input\_tensor):  
 word\_vector=self.embedding(input\_tensor)  
 output=self.rnn(word\_vector)[0][0][len(input\_tensor)-1]  
 output=self.cls(output)  
 output=self.softmax(output)  
 return output  
  
def run\_rnn(rnn,input\_tensor):  
 output=rnn(input\_tensor.unsqueeze(dim=0))  
 return output  
"""  
  
def get\_traindata(file\_path):  
 files = os.listdir(file\_path)  
 testlist=list()  
 pattern = r'[a-zA-Z]'  
 for file in files:  
 txt\_list = []  
 with open(os.path.join(file\_path, file), "r", encoding="UTF-8") as f:  
 for l in f:  
 l.strip()  
 l=l.lower()  
 l=l.replace(special\_chars,"")  
 l.split(' ')  
 l = re.sub(r'[^\w\s]', '', l)  
 words=re.split(r'\b',l)  
 for word in words:  
 result = re.search(pattern, word)  
 if result:  
 txt\_list.append(word)  
 testlist.append(txt\_list)  
 return testlist  
  
pos\_train\_list=get\_traindata(pos\_file\_path)  
neg\_train\_list=get\_traindata(neg\_file\_path)  
  
print(pos\_train\_list[0])  
print(comment\_to\_tensor(pos\_train\_list[0]))  
print(run\_rnn(rnn,comment\_to\_tensor(pos\_train\_list[0])))  
  
all\_data=list()  
categories=["好评","差评"]  
print("加载数据集")  
for l in pos\_train\_list[0:10]:  
 all\_data.append((comment\_to\_tensor(l),torch.tensor([1],dtype=torch.long)))  
for l in neg\_train\_list[0:10]:  
 all\_data.append((comment\_to\_tensor(l),torch.tensor([0],dtype=torch.long)))  
  
def train(rnn,criterion,input\_tensor,category\_tensor):  
 learning\_rate=0.005  
 rnn.zero\_grad()  
 output=run\_rnn(rnn,input\_tensor)  
 loss=criterion(output,category\_tensor)  
 loss.backward()  
 for p in rnn.parameters():  
 p.data.add\_(p.grad.data,alpha=-learning\_rate)  
 return output,loss.item()  
  
def evaluate(rnn,input\_tensor):  
 with torch.no\_grad():  
 hidden=rnn.inithidden()  
 output=run\_rnn(rnn,input\_tensor)  
 return output  
  
import random  
random.shuffle(all\_data)  
split\_ratio=0.7  
data\_len=len(all\_data)  
test\_data=all\_data[int(data\_len\*split\_ratio)]  
trainif=False  
if trainif:  
 from tqdm import tqdm  
 epoch=1  
 criterion=nn.NLLLoss()  
 loss\_sum=0  
 al\_losses=[]  
 plot\_every=100  
 for e in range(epoch):  
 for ind,(comment\_tensor,label) in enumerate(tqdm(all\_data)):  
 output,loss=train(rnn,criterion,comment\_tensor,label)  
 loss\_sum+=loss  
 if ind % plot\_every==0:  
 al\_losses.append(loss\_sum/plot\_every)  
 loss\_sum=0  
 c=0  
 for i,(title,category) in enumerate(tqdm(test\_data)):  
 output=evaluate(rnn,title)  
 topn,topi=output.topk(1)  
 if topi.item()==category[0].item():  
 c+=1  
 print('accuracy=',c/len(test\_data))  
 torch.save(rnn.state\_dict(), 'zifu\_rnn.pkl')  
 print("savesuccessfully")  
 import matplotlib as plt  
 plt.rigure(figsize=(10,7))  
 plt.ylabel("average loss")  
 plt.plot(al\_losses[1:])

**二、提高训练**

尝试对现有实验做修改和调整，例如修改优化器、损失函数、调整网络结构等，可以对比LSTM等网络的效果。比较不同参数带来的性能影响，分析猜测其背后的原因。

（格式：正文12号字体）

**三、实验心得**

实验过程中遇到的什么问题。尝试使用什么方法去解决。通过实验获得了什么感悟与理解。

（格式：正文12号字体）