# 题目：卷积神经网络预测面部表情

姓名：赵建权

**学院：河南先进技术研究院**

**年级;2020级**

**专业：计算机科学与技术**

**课程名：2020秋季高级人工智能**

摘 要：

基于PyTorch深度学习模型，实现通过模拟Alexnet卷积神经网络进行面部表情分类预测，对比在进行dropout，L2正则化，未使用任何缓解过拟合手段下的实验结果，实验表明，在部分类别数据量不充分时，实现缓解过拟合的主要策略除了有dropout和正则化两种选择，还可以通过调整batch size进行有效缓解过拟合。

关键词：PyTorch，卷积神经网络，过拟合，dropout，正则化，batch size

Abstract

Based on the PyTorch deep learning model, the results of the experiment under dropout, L2 normalization and no mitigation fit method are realized by simulating the Alexnet convulsive neural network, and the experiment shows that when the amount of data in some categories is insufficient, the main strategy of mitigating overfitt can be effectively alleviated by adjusting patchsize.

**Keywords**: PyTorch, convolutional neural network, overfitting, dropout, normalization, batchsize

目 录

[目 录 4](#_Toc60841621)

[一 实验设计 5](#_Toc60841622)

[（i）分析Alexnet卷积神经网络 5](#_Toc60841623)

[（ii）神经网络设计 7](#_Toc60841624)

[（iii）超参数设置 7](#_Toc60841625)

[二 实验验证 8](#_Toc60841626)

[（i）卷积神经网络训练 8](#_Toc60841627)

[（ii）发现问题 9](#_Toc60841628)

[（iii）解决方法 10](#_Toc60841629)

[三 实验结论： 11](#_Toc60841630)

[参考文献： 12](#_Toc60841631)

[附 录 13](#_Toc60841632)

[（i）代码流程图 13](#_Toc60841633)

[（ii）核心代码及注释 14](#_Toc60841634)

[（1）导入所需包 14](#_Toc60841635)

[（2）数据读入及处理 14](#_Toc60841636)

[（3）设计数据加载器 15](#_Toc60841637)

[（4）实现网络结构 16](#_Toc60841638)

[（6）创建动态处理模型及实例化 18](#_Toc60841639)

[（7）训练过程可视化 20](#_Toc60841640)

一 实验设计

**（i）分析Alexnet卷积神经网络**

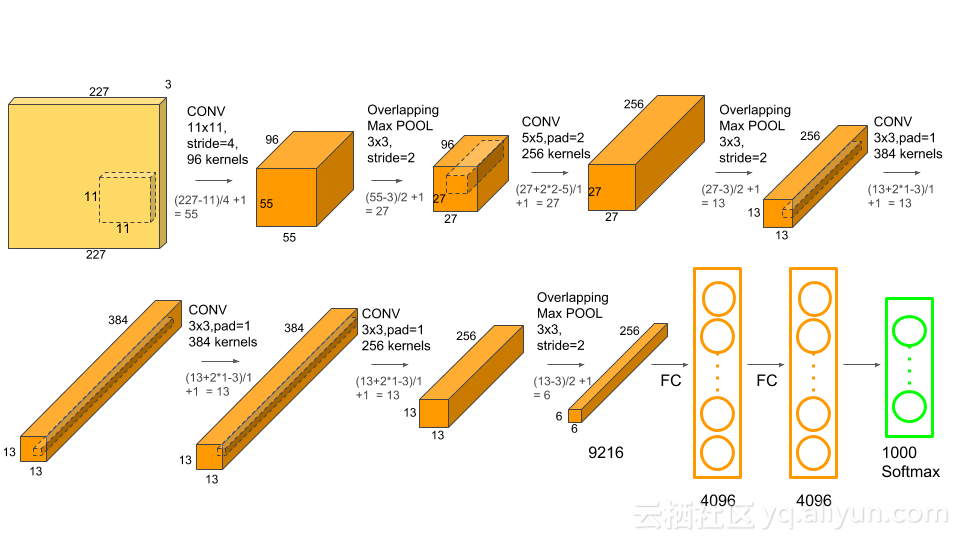


图1.1.1 Alexnet卷积神经网络结构图

网络结构卷积部分（表1.1.1）：

表1.1.1 Alexnet卷积部分分析

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Layer\_name** | **Conv** | **Maxpool** | **Conv** | **Maxpool** | **Conv** | **Conv** | Conv |
| **Input\_channel** | 3 | —— | 96 | —— | 256 | 384 | 384 |
| **out\_channel** | 96 | —— | 256 | —— | 384 | 384 | 256 |
| **Stride** | 4 | 2 | 1 | 2 | 1 | 1 | 1 |
| **Kernel size** | 11x11 | 3x3 | 5x5 | 3x3 | 3x3 | 3x3 | 3x3 |
| **Padding** | —— | —— | 2 | —— | 1 | 1 | 1 |
| **Matrix size** | 56x56x96 | 27x37x96 | 27x27x256 | 13x13x256 | 13x13x384 | 13x13x384 | 13x13x256 |

卷积层实现了将矩阵在二维大小上面的快速减小，并且在channel上面快速的增加，并设定在256。其中，卷积核的大小为从大到小，如表中所示。

**（ii）神经网络设计**

卷积神经网络的卷积层3个，卷积核大小设置为⌈√matrix\_size⌉，全连接层中隐藏层1个网络结构图如图1.2.1所示。

256

FC

2x2x256

4x4x256

5x5x64

11x11x64

42x42x16

Conv3

2x2

256 kernel

Maxpool

3x3

Maxpool

3x3

Maxpool

3x3

Conv2

4x4

64 kernel

Conv1

7x7

16 kernel

48x48

Stride=1

Padding=0

14x14x16

FC

32

7

图1.2.1 卷积神经网络设计结构

网络结构分析：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Layer\_name** | **Conv** | **Maxpool** | **Conv** | **Maxpool** | **Conv** | **Maxpool** |
| **Input\_channel** | 1 | —— | 16 | —— | 64 | —— |
| **out\_channel** | 16 | —— | 64 | —— | 256 | —— |
| **Kernel size** | 7x7 | 3x3 | 4x4 | 3x3 | 2x2 | 3x3 |
| **Matrix size** | 42x42x16 | 14x14x16 | 11x11x64 | 5x5x64 | 4x4x256 | 2x2x256 |

**（iii）超参数设置**

epochs默认设置为100，进行100次迭代训练，batchsize设置为32，以实现更多有效的利用训练集。

二 实验验证

**（i）卷积神经网络训练**

实验表明，当不进行正则化，dropout时，神经网络出现非常严重的过拟合，卷积神经网络在测试集上面的accuracy 最大仅为32.25%，但是在训练集上的accuracy已经达到了99.87%（图2.1.1），epoch设置为30。

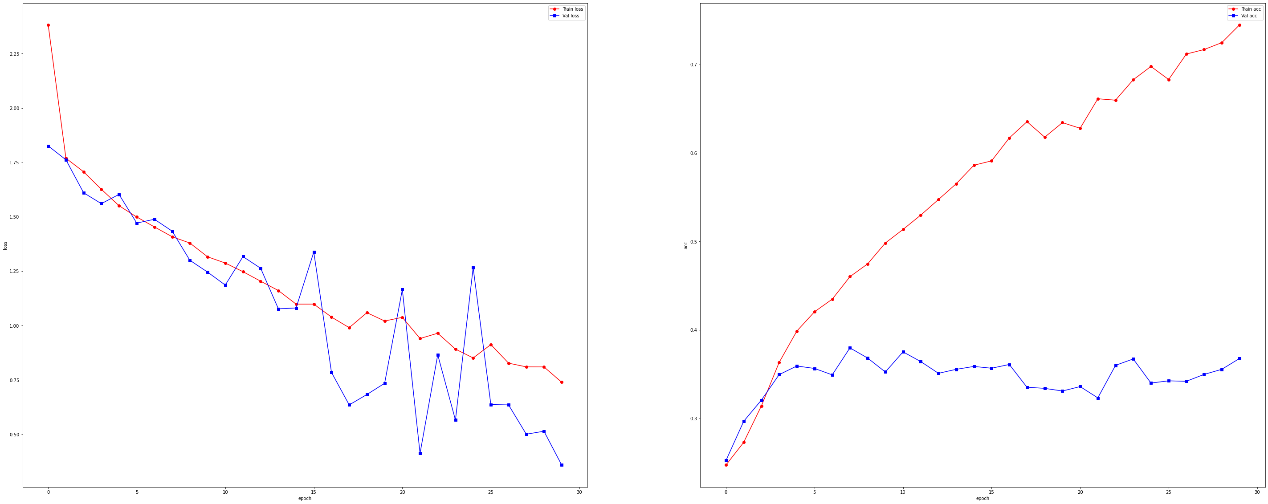
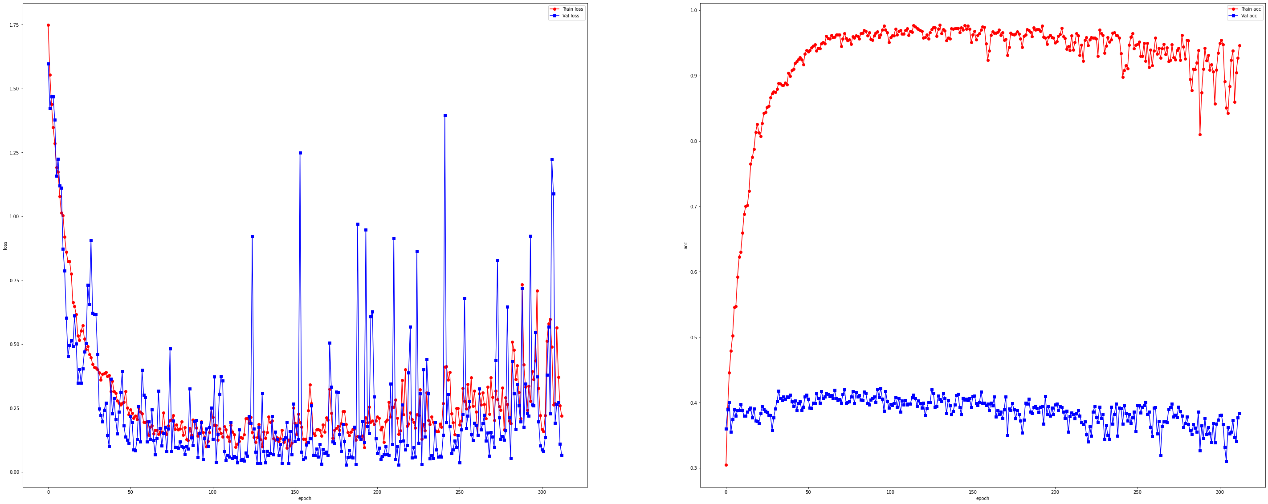


图2.1.1 无正则化，dropout技术应用损失值和精确度

通过调整卷积神经网络卷积层以及全连接部分神经元个数尝试环节过拟合现象，发现除减少训练时间外，并不能缓解过拟合的现象。

同时选择使用dropout和L2正则化的方式进行缓解过拟合现象，实验结果表明，能够将模型的accuracy提升至45.28%，但是在训练集上表现很好，依然可以达到98.61%（图2.1.2）过拟合现象并未得到有效缓解。其中dropout在全连接层的概率分别设置为0.2和0.5，L2正则化系数设置为0.002，epoch设置为313。



**（ii）发现问题**

由于无论怎样设置dropout和正则化的超参数，都无法进一步提升在测试集上面的accuracy，所以考虑是数据出现了问题，对数据进行分析。分析发现，在训练集中，也就是全部数据的前20000个数据中，类别为1的样本量仅有323个，而在测试集的8000个数据中样本类别为1的数据量有72个，训练集中样本类别为1的种类占比虽然大于测试集当中类别为1的占比，但由于训练集中类别为1的样本量较少，比例为0.01565，说明如果每个样本仅使用一次，那么 每批样本量最少要有100个才能保证每批训练样本当中有1个类别为1的样本，但整体上样本量仍然不足，会导致对类别为1的预测不够准确。训练集中各类别数量如表2.2.1。

表2.2.1 训练集中各类别数量

|  |  |
| --- | --- |
| **类别** | **数量** |
| **0** | 2803 |
| **1** | 323 |
| **2** | 2916 |
| **3** | 5008 |
| **4** | 3357 |
| **5** | 2132 |
| **6** | 3460 |

**（iii）解决方法**

为保证每批样本都有类别为1的样本，选择将batchsize调整为1024，以确保尽可能每次训练过程中都能够有7类样本在训练集当中。

实验结果：

batchsize调整之后，在测试集上面的accuracy可以得到再次提升，最高可达53.99%，（图2.3.1），epoch为50。

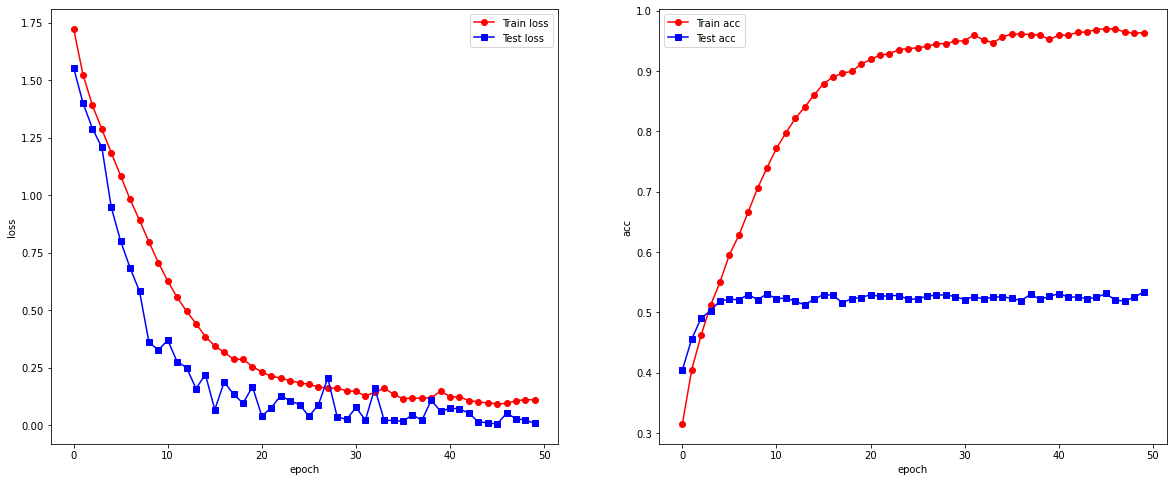


图2.3.1 调整batchsize训练结果

正则化和dropout的影响变小甚至可以不计，因为使用dropout和正则化并没有进一步提升在测试集上面的accuracy（图2.3.2）。

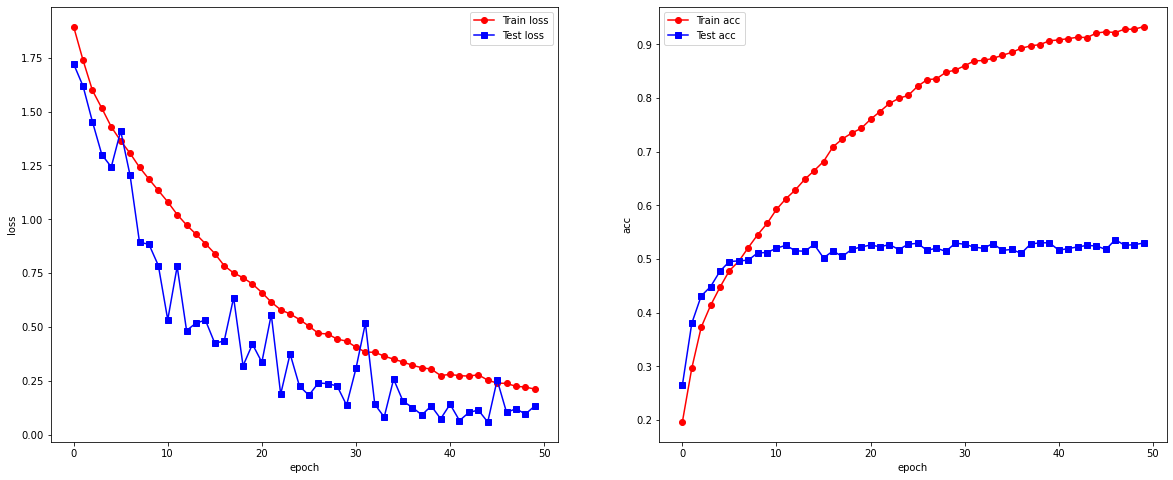


图2.3.2 增加使用dropout和正则化技术

**三 实验结论：**

当实验比较严重的过拟合现象，原因可能不仅仅是网络设计有问题，导致出现过多神经元，从而引起过拟合，还有可能是数据本身的原因导致的过拟合。解决过拟合的方法除了正则化，dropout之外，通过训练数据的优化，也可以缓解过拟合现象。除此之外，还可以使用数据增强技术增加特定种类的数据量，因为实验时个人能力、计算机的计算能力和时间有限，所以数据增强技术的方法未进行实验。

参考文献：

[1]孙玉林，余本国著.PyTorch深度学习入门与实战[M].北京：中国水利水电出版社.2020.

[2]（美）伊恩·古德费洛著；赵申剑，黎彧君，符天凡，李凯译.深度学习[M].北京：人民邮电出版社.2017.

[3]张校捷著.深入浅出PyTorch 从模型到源码[M].北京：电子工业出版社.2020.

附 录

（i）代码流程图

开始

导入所需包

数据读入及处理

创建数据加载器

实现网络结构

选择优化器和损失函数

创建动态处理模型及实例化

训练过程可视化

结束

（ii）核心代码及注释

（1）导入所需包

#Pytorch部分为最主要的部分，使用的模块包括有网络，优化器，损失函数#以及其他工具。

import torch

import torch.nn as nn

from torch.optim import Adam

import torch.utils.data as Data

#导入该部分的目的为在测试集上使用，验证在测试集上的精确度以及损失#值。

from sklearn.metrics import accuracy\_score

#matplotlib.pyplot用来将训练出来的在训练集以及测试集上面的损失值、 #精确度在坐标当中画出来。而PIL中的Image模块用来将任一数据当中按照图#像的模式画出，以检测经过处理的数据是否正确。

import matplotlib.pyplot as plt

from PIL import Image

#numpy以及pandas是用来处理数据的读入以及数据的整理。

import numpy as np

import pandas as pd

import copy

（2）数据读入及处理

#将label和像素数据分离

path = 'project2\_train.csv'

df = pd.read\_csv(path)

#提取label数据

df\_y = df[['label']]

#提取feature（即像素）数据

df\_x = df[['feature']]

#将label写入label.csv

df\_y.to\_csv('label.csv', index=False, header=False)

#将feature数据写入data.csv

df\_x.to\_csv('data.csv', index=False, header=False)

#使用np.loadtxt()可以直接将在.csv文件中同一个单元格内的数据处理成分#开的一个个的数据，而不用在人为去处理

#读取像素数据

x = np.loadtxt('data.csv')

#读取标签数据

y= np.loadtxt('label.csv')

#将数据格式转换为通道数为1，所以每一个图片的维度变为1x48x48

x=x.reshape(28709,1,48,48)

x.shape

#将数据按照要求分为训练集和验证集

X\_train=x[0:20001]

X\_test=x[20001:]

y\_train=y[0:20001]

y\_test=y[20001:]

#转换成PyTorch框架内的数据格式——张量形式

X\_train\_tor=torch.from\_numpy(X\_train.astype(np.float32))

X\_test\_tor=torch.from\_numpy(X\_test.astype(np.float32))

y\_train\_tor=torch.from\_numpy(y\_train.astype(np.int64))

y\_test\_tor=torch.from\_numpy(y\_test.astype(np.int64))

#将训练集的图片特征和标签结合起来成为一个数据整体

train\_data=Data.TensorDataset(X\_train\_tor,y\_train\_tor)

（3）设计数据加载器

#batch\_size 即为前面所述的batch size 所以设定值首先设定为512，为了#能够让数据更加具有随机性，所以此处shuffle=True

train\_loader=Data.DataLoader(

dataset=train\_data

,batch\_size=512

, shuffle=True

)

（4）实现网络结构

#所有超参数的设定即为实验设计中神经网络设计对网络设计的实现

class MyConvNet(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Sequential(

nn.Conv2d(

in\_channels=1

,out\_channels=16

,kernel\_size=7

,stride=1

# ,padding=1

)

# ,nn.ReLU()

,nn.MaxPool2d(

kernel\_size=3

,stride=3

)

)

self.conv2 = nn.Sequential(

nn.Conv2d(

in\_channels=16

,out\_channels=64

,kernel\_size=4

,stride=1

# ,padding=1

)

,nn.MaxPool2d(

kernel\_size=3

,stride=2

)

)

self.conv3 = nn.Sequential(

nn.Conv2d(

in\_channels=64

,out\_channels=256

,kernel\_size=2

,stride=1

# ,padding=1

)

# ,nn.ReLU()

,nn.MaxPool2d(

kernel\_size=2

,stride=2

)

)

self.classifier=nn.Sequential(

nn.Linear(256\*2\*2,256)

,nn.ReLU()

,nn.Dropout(p=0.2)

,nn.Linear(256,32)

,nn.ReLU()

# ,nn.Dropout(p=0.5)

,nn.Linear(32,7)

)

def forward(self, x):

x = self.conv1(x)

x = self.conv2(x)

x = self.conv3(x)

x=x.view(x.size(0),-1)

output=self.classifier(x)

return output

（5）选择优化器和损失函数

#学习率、正则项系数设定为参考文献[1]中提倡的数值。

#损失函数选择分类器常用的交叉熵函数

optimizer=torch.optim.Adam(myconvnet.parameters(), lr=0.0003,weight\_decay=0.01)

criterion = nn.CrossEntropyLoss()

（6）创建动态处理模型及实例化

#传入参数包括有网络结构的实例化对象，训练数据加载器 ，测试集数据，#损失函数，优化器，迭代次数的设定。

def train\_model(

model

, traindataloader

, X\_test

, y\_test

, criterion

, optimizer

, num\_epochs=100

):

#此处的list用来存储每一次迭代后的模型在训练集和测试集上面的损失值#和精确度

batch\_num = len(traindataloader)

train\_loss\_all = []

train\_acc\_all = []

test\_loss\_all = []

test\_acc\_all = []

#训练过程便于观察的单次训练以及结果输出

for epoch in range(num\_epochs):

print('Epoch {} /{} '.format(epoch, num\_epochs))

print('-' \* 20)

train\_loss = 0.0

train\_corrects = 0

train\_num = 0

test\_loss = 0.0

acc = 0.0

for step, (b\_x, b\_y) in enumerate(traindataloader):

if step < batch\_num:

model.train()

output = model(b\_x)

pre\_lab = torch.argmax(output, 1)

#根据损失函数和优化器的方式进行梯度的计算及反向传播进行参数的更新

loss = criterion(output, b\_y)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

#训练的在测试集上面的精确度及损失值的保存

train\_loss += loss.item() \* b\_x.size(0)

train\_corrects += torch.sum(pre\_lab == b\_y.data)

train\_num += b\_x.size(0)

#训练的模型在测试集上面运行，并存储精确度和损失值

model.eval()

output = myconvnet(X\_test)

pre\_lab = torch.argmax(output, 1)

acc = accuracy\_score(y\_test, pre\_lab)

test\_loss = loss.item()

#单次迭代的结果输出

train\_loss\_all.append(train\_loss / train\_num)

train\_acc\_all.append(train\_corrects.double().item() / train\_num)

test\_loss\_all.append(test\_loss)

test\_acc\_all.append(acc)

print('{} Train Loss: {:.4f} Train Acc : {:.4f}'.format(epoch, train\_loss\_all[-1], train\_acc\_all[-1]))

print('{} Test Loss: {:.4f} Test Acc : {:.4f}'.format(epoch, test\_loss\_all[-1], test\_acc\_all[-1]))

#相关数据将作为参数被返回

train\_process = pd.DataFrame(

data={"epoch": range(num\_epochs)

, "train\_loss\_all": train\_loss\_all

, "train\_acc\_all": train\_acc\_all

, "test\_loss\_all": test\_loss\_all

, "test\_acc\_all": test\_acc\_all

}

)

#训练过程模型的实例化

myconvnet,train\_process=train\_model(

myconvnet

,train\_loader

,X\_test\_tor

,y\_test\_tor

,criterion

,optimizer

,num\_epochs=100

)

（7）训练过程可视化

plt.figure(figsize=(20,8))

plt.subplot(1,2,1)

plt.plot(train\_process.epoch,train\_process.train\_loss\_all,"ro-",label="Train loss")

plt.plot(train\_process.epoch,train\_process.test\_loss\_all,"bs-",label="Test loss")

plt.legend()

plt.xlabel("epoch")

plt.ylabel("loss")

plt.subplot(1,2,2)

plt.plot(train\_process.epoch,train\_process.train\_acc\_all,"ro-",label="Train acc")

plt.plot(train\_process.epoch,train\_process.test\_acc\_all,"bs-",label="Test acc")

plt.legend()

plt.xlabel("epoch")

plt.ylabel("acc")

plt.show()