

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **PyTorch基本操作实验** |
| **姓 名：** |  |
| **学 号：** |  |
| **日 期：** |  |

## 一、实验内容

实验内容包含要进行什么实验，实验的目的是什么，实验用到的算法及其原理的简单介绍。

#### PyTorch基本操作实验

1. 使用 Tensor 初始化一个 1×3的矩阵M和一个 2×1的矩阵 N，对两矩阵进行减法操作（要求实现三种不同的形式），给出结果并分析三种方式的不同（如果出现报错，分析报错的原因），同时需要指出在计算过程中发生了什么。

2. ① 利用Tensor 创建两个大小分别 3×2 和 4×2 的随机数矩阵 P和Q ，要求服从均值为0，标准差0.01为的正态分布；② 对第二步得到的矩阵Q进行形状变换得到Q的转置QT；③ 对上述得到的矩阵P 和矩阵 QT求矩阵相乘。

3. 给定公式 y3 = y1 + y2 = x2 + x3，且x = 1。利用学习所得到的Tensor的相关知识，求y3对x的梯度，即。要求在计算过程中，在计算x3 时中断梯度的追踪，观察结果并进行原因分析。

提示, 可使用 with torch.no\_grad()， 举例:

with torch.no\_grad():

y = x \* 5

实验目的：理解Tensor的概念，可以熟练的掌握的PyTorch基本操作，广播机制，掌握如何利用PyTorch深度学习框架求解梯度，进行梯度的追踪。

#### 1.2 Logistic 回归实验

1. 要求动手从0实现 logistic 回归（只借助Tensor和Numpy相关的库）在人工构造的数据集上进行训练和测试，并从loss以及训练集上的准确率等多个角度对结果进行分析（可借助nn.BCELoss或nn.BCEWithLogitsLoss作为损失函数，从零实现二元交叉熵为选作）

2. 利用 torch.nn 实现 logistic 回归在人工构造的数据集上进行训练和测试，并对结果进行分析， 并从loss以及训练集上的准确率等多个角度对结果进行分析。

实验目的：从零实现和利用torch.nn 实现 logistic 回归，更好地理解线性回归的过程，掌握PyTorch框架。

#### 1.3 Softmax 回归实验

1. 要求动手从0实现 softmax 回归（只借助Tensor和Numpy相关的库）在Fashion-MNIST数据集上进行训练和测试，并从loss、训练集以及测试集上的准确率等多个角度对结果进行分析（要求从零实现交叉熵损失函数）

2. 利用torch.nn实现 softmax 回归在Fashion-MNIST数据集上进行训练和测试，并从loss，训练集以及测试集上的准确率等多个角度对结果进行分析。

实验目的：从零实现和利用torch.nn 实现 Softmax 回归，更好地理解Softmax回归的过程，掌握PyTorch框架。

## 二、实验环境及实验数据集

实验环境：Python3.7+Anaconda+PyTorch+Jupyter Notebook

涉及的数据集：人工构造的数据集；Fashion-MNIST数据集

## 实验过程

实验过程包括在编写代码时一些需要注意的事项，可附代码片段进行说明；还应该包括在训练模型时进行的一些步骤、参数设置等内容。

#### 3.1 PyTorch基本操作实验

1.矩阵减法操作

当形状不同的Tensor按元素发生减法运算时，计算过程中可能会触发广播机制，适当复制元素使得这两个Tensor形状相同后再按元素运算。主要代码如下：

1. import torch
2. M=torch.rand(1,3)
3. N=torch.rand(2,1)
4. print(M-N)
5. print(torch.sub(M,N))
6. M.sub\_(N)
7. print(M)
8. 转置及乘法操作

Tensor运算：t（转置）、mm（矩阵乘法）。主要代码如下：

1. import torch
2. P=torch.normal(0,0.01,(3,2))
3. print(P)
4. Q=torch.normal(0,0.01,(4,2))
5. print(Q)
6. Q.t\_()
7. print(Q)
8. print(torch.mm(P,Q))
9. 求梯度操作

创建一个Tensor并设置requires\_grad=True，并进行其他运算操作。在求梯度时，不允许张量对张量求导，只允许标量对张量求导，求导结果是和自变量同形的张量。即在y3.backward()时，如果y3是标量，则不需要为backward()传入任何参数；否则，需要传入一个与y3同形的Tensor将其转换成标量，再进行标量对张量的求导。主要代码如下：

1. import torch
2. x=torch.tensor(1.,requires\_grad=True)
3. y1=x\*x
4. with torch.no\_grad():
5. y2=x\*x\*x
6. y3=y1+y2
7. out=y3.mean()
8. out.backward()
9. print(x.grad)

#### 3.2 Logistic 回归实验

logistic回归是一种分类方法，用于两分类问题。其基本思想为：a. 寻找合适的假设函数，即分类函数，用以预测输入数据的判断结果；b. 构造代价函数，即[损失函数](https://so.csdn.net/so/search?q=%E6%8D%9F%E5%A4%B1%E5%87%BD%E6%95%B0&spm=1001.2101.3001.7020" \t "https://blog.csdn.net/weixin_44645198/article/details/_blank)，用以表示预测的输出结果与训练数据的实际类别之间的偏差；c. 最小化代价函数，从而获取最优的模型参数。

1.从零实现logistic回归实验

1. 生成和读取数据
2. *#生成数据*
3. n\_data=torch.ones(50,2)
4. x1=torch.normal(2\*n\_data,1)
5. y1=torch.zeros(50)
6. x2=torch.normal(-2\*n\_data,1)
7. y2=torch.ones(50)
8. features=torch.cat((x1,x2),0).type(torch.FloatTensor)
9. labels=torch.cat((y1,y2),0).type(torch.FloatTensor)
10. testfeatures=torch.cat((x1,x2),0).type(torch.FloatTensor)
11. testlabels=torch.cat((y1,y2),0).type(torch.FloatTensor)
12. *#读取数据*
13. def data\_iter(batch\_size,features,labels):
14. num\_examples=len(features)
15. indices=list(range(num\_examples))
16. np.random.shuffle(indices)
17. for i in range(0,num\_examples,batch\_size):
18. j=torch.LongTensor(indices[i:min(i+batch\_size,num\_examples)])
19. yield features.index\_select(0,j),labels.index\_select(0,j)
20. 构建模型
21. w=torch.tensor(np.random.normal(0,0.01,(num\_inputs,1)),dtype=torch.float32)
22. b=torch.zeros(1,dtype=torch.float32)
23. w.requires\_grad\_(requires\_grad=True)
24. b.requires\_grad\_(requires\_grad=True)
25. def linreg(X,w,b):
26. return torch.mm(X,w)+b
27. 损失函数和优化函数
28. def squared\_loss(y\_hat,y):
29. return(y\_hat-y.view(y\_hat.size()))\*\*2/2
30. def sgd(params,lr,batch\_size):
31. for param in params:
32. param.data-=lr\*param.grad/batch\_size
33. 模型训练
34. lr = 0.03
35. num\_epochs = 20
36. net = linreg
37. loss = squared\_loss
38. batch\_size = 10
39. train\_acc= []
40. train\_loss =[]
41. for epoch in range(num\_epochs): *# 训练模型一共需要num\_epochs个迭代周期*
42. train\_l\_sum, train\_acc\_sum,n = 0.0,0.0,0
43. *#在每一个迭代周期中，会使用训练数据集中所有样本一次*
44. for X, y in data\_iter(batch\_size, features, labels): *# x和y分别是小批量样本的特征和标签*
45. y\_hat = net(X, w, b)
46. l = loss(y\_hat, y).sum() *# l是有关小批量X和y的损失*
47. l.backward() *# 小批量的损失对模型参数求梯度*
48. sgd([w, b], lr, batch\_size) *# 使用小批量随机梯度下降迭代模型参数*
49. w.grad.data.zero\_() *# 梯度清零*
50. b.grad.data.zero\_() *# 梯度清零*
51. *#计算每个epoch的loss*
52. train\_l\_sum += l.item()
53. *#计算训练样本的准确率*
54. y\_hat = torch.squeeze(torch.where(y\_hat>0.5,torch.tensor(1.0),torch.tensor(0.0)))
55. train\_acc\_sum += (y\_hat==y).sum().item()
56. *#每一个epoch的所有样本数*
57. n+= y.shape[0]
58. *#train\_l = loss(net(trainfeatures, w, b), trainlabels)*
59. train\_acc.append(train\_acc\_sum/n)
60. train\_loss.append(train\_l\_sum/n)
61. print('epoch %d, loss %.4f, train accuracy %.3f'
62. % (epoch + 1, train\_loss[epoch], train\_acc[epoch]))
63. torch.nn实现logistic回归实验
64. 生成和读取数据
65. *# 2、读取数据，采用PyTorch提供的data库读取数据。*
66. dataset = Data.TensorDataset(features, labels)
67. lr = 0.03
68. num\_epochs = 20
69. batch\_size = 10
70. *# 把dataset放入DataLoader*
71. data\_iter = Data.DataLoader(
72. dataset=dataset,  *# torch TensorDataset format*
73. batch\_size=batch\_size,
74. shuffle=True,  *# 是否打乱数据*
75. num\_workers=0,  *# 多线程来读数据，在Win下需要设置为0*
76. )
77. test\_iter = Data.DataLoader(
78. dataset=dataset,
79. batch\_size=batch\_size,
80. shuffle=True,
81. num\_workers=0,
82. )
83. 构建模型
84. *# 构建模型，最常见的是继承nn.Module然后构建自己的网络*
85. class LogisticNet(torch.nn.Module):
86. def \_\_init\_\_(self, n\_feature):
87. super(LogisticNet, self).\_\_init\_\_()
88. self.linear = torch.nn.Linear(n\_feature, 1)
89. *# 向前传播*
90. def forward(self, x):
91. y\_hat = 1/(1+torch.exp(self.linear(x)))
92. return y\_hat
93. net = LogisticNet(num\_inputs)
94. 模型参数初始化
95. *# 模型参数初始化*
96. init.normal\_(net.linear.weight, mean=0, std=0.01)
97. init.constant\_(net.linear.bias, val=0)  *# 也可以直接修改bias的data:net[0].bias.data.fill\_(0)*
98. 损失函数和优化
99. *# 损失函数和优化*
100. loss = torch.nn.BCELoss()
101. optimizer = optim.SGD(net.parameters(), lr=0.03)
102. 模型训练
103. *# 模型训练*
104. for epoch in range(num\_epochs + 1):
105. for x, y in data\_iter:
106. y\_hat = net(x)
107. l = loss(y\_hat, y)
108. optimizer.zero\_grad()  *# 梯度清零，等价于net.zero\_grad()*
109. l.backward()
110. optimizer.step()  *# 更新所有参数*
111. *# 训练集上的正确率*
112. allTrain = 0
113. rightTrain = 0
114. for train\_x, train\_y in test\_iter:
115. allTrain += len(train\_y)
116. *# train\_x = train\_x.view(train\_x.size()[0], train\_x.size()[1], -1)*
117. train\_out = net(train\_x)
118. mask = train\_out.ge(0.5).float()
119. correct = (mask.view(-1, 1) == train\_y.view(-1, 1)).sum()
120. rightTrain += correct.float().sum()
121. print('epoch %d, loss %f, train accuracy:%f' % (epoch, l.item(),rightTrain/allTrain))

#### 3.3 Softmax 回归实验

1.从零实现softmax回归实验

1）导入Fashion-MNIST数据集

1. *# 1、加载Fashion-MNIST数据集（采用已划分好的训练集和测试集）*
2. mnist\_train=torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST',train=True,download=True,transform=transforms.ToTensor())
3. mnist\_test=torchvision.datasets.FashionMNIST(root='~/Datasets/FashionMNIST',train=False,download=True,transform=transforms.ToTensor())
4. *# 2、通过Dataloader读取小批量数据样本*
5. batch\_size=256
6. train\_iter = torch.utils.data.DataLoader(mnist\_train, batch\_size=batch\_size, shuffle=True, num\_workers=0)
7. test\_iter = torch.utils.data.DataLoader(mnist\_test, batch\_size=batch\_size, shuffle=False, num\_workers=0)
8. 构建模型，并进行参数初始化。损失函数采用交叉熵损失函数，并定义优化器优化。
9. *# 3.1参数初始化*
10. W = torch.normal(0, 0.1, (num\_inputs, num\_outputs), dtype=torch.float32)  *# 784\*10*
11. b = torch.normal(0, 0.01, (1, num\_outputs), dtype=torch.float32)  *# 偏差参数1\*10*
12. *# 模型的参数梯度*
13. W.requires\_grad\_(requires\_grad=True)
14. b.requires\_grad\_(requires\_grad=True)
15. *# 3.2 softmax回归模型*
16. def softmax(X):  *# softmax计算*
17. X\_exp = X.exp()  *# 对每个元素做指数运算*
18. partition = X\_exp.sum(dim=1, keepdim=True)  *# 求列和，即对同行元素求和 n\*1*
19. return X\_exp / partition  *# broadcast*
20. def net(X):
21. return softmax(torch.mm(X.view((-1, num\_inputs)), W) + b)
22. *#3.3 交叉熵损失函数*
23. def loss(y\_hat, y):
24. return - torch.log(y\_hat.gather(1, y.view(-1, 1)))
25. *# 3.4 优化器*
26. def sgd(params, lr, batch\_size):
27. for param in params:
28. param.data -= lr \* param.grad / batch\_size  *# 注意这里更改param时用的param.data*
29. 计算准确率
30. *# 4、计算分类准确率*
31. def evaluate\_accurcy(data\_iter, net):
32. right\_count, all\_num = 0.0, 0
33. for x, y in data\_iter:
34. right\_count += (net(x).argmax(dim=1) == y).float().sum().item()
35. all\_num += y.shape[0]
36. return right\_count / all\_num
37. 模型训练和测试
38. lr = 0.1
39. num\_epochs = 5
40. for epoch in range(num\_epochs):
41. train\_right\_sum, train\_all\_sum, train\_loss\_sum = 0.0, 0, 0.0
42. for X, y in tqdm(train\_iter):  *# tqdm显示训练进度条*
43. y\_hat = net(X)
44. l = loss(y\_hat, y).sum()  *# 计算loss*
45. l.backward()  *# 求梯度*
46. sgd([W, b], lr, batch\_size)  *# 参数更新*
47. W.grad.data.zero\_()
48. b.grad.data.zero\_()  *# 梯度清零*
49. train\_loss\_sum += l.item()  *# 损失*
50. train\_right\_sum += (y\_hat.argmax(dim=1) == y).sum().item()  *# 训练集准确率*
51. train\_all\_sum += y.shape[0]
52. test\_acc = evaluate\_accurcy(test\_iter, net)  *# 测试集准确率*
53. print('epoch %d, loss %.4f' % (epoch+1, train\_loss\_sum/train\_all\_sum))
54. print('train accuracy %.3f, test accuracy %.3f' % (train\_right\_sum/train\_all\_sum, test\_acc))

2.torch.nn实现softmax回归实验

1）导入Fashion-MNIST数据集

2）构建模型

1. num\_inputs = 784
2. num\_outputs = 10  *# 共10类*
3. class softmaxnet(torch.nn.Module):
4. def \_\_init\_\_(self, n\_features, n\_labels):
5. super(softmaxnet, self).\_\_init\_\_()
6. self.linear = torch.nn.Linear(n\_features, n\_labels)
7. def forward(self, x):
8. x\_ = x.view((-1, num\_inputs))
9. y\_ = self.linear(x\_)
10. return y\_
11. net = softmaxnet(num\_inputs, num\_outputs)
12. 损失函数及优化器
13. loss = torch.nn.CrossEntropyLoss()
14. optimizer = optim.SGD(net.parameters(), lr=lr)
15. 准确度计算
16. def get\_test\_info(data\_iter, net):
17. right\_count, all\_count = 0.0, 0
18. for x, y in data\_iter:
19. y\_ = net(x)
20. l = loss(y\_, y)
21. right\_count += (y\_.argmax(dim=1)==y).sum().item()
22. all\_count += y.shape[0]
23. return right\_count/all\_count, l.item()
24. 模型训练
25. for epoch in range(num\_epoch):
26. train\_r\_num, train\_all\_num = 0.0, 0
27. for X, y in tqdm(train\_iter):
28. y\_ = net(X)
29. l = loss(y\_, y)
30. l.backward()
31. optimizer.step()
32. optimizer.zero\_grad()
33. train\_r\_num += (y\_.argmax(dim=1) == y).sum().item()
34. train\_all\_num += y.shape[0]
35. test\_acc, test\_ave\_loss = get\_test\_info(test\_iter, net)
36. print('epoch %d, train loss %.4f, train acc %.3f' % (epoch+1, l.item(), train\_r\_num/train\_all\_num))
37. print('test loss %.4f, test acc %.3f' % (test\_ave\_loss, test\_acc))

## 四、实验结果

实验结果包括程序运行结果以及对结果的分析，尽量用图表展示实验结果，并且通过结果进行相关的分析。

#### 4.1 PyTorch基本操作实验

1.矩阵减法程序运行结果如图1所示。

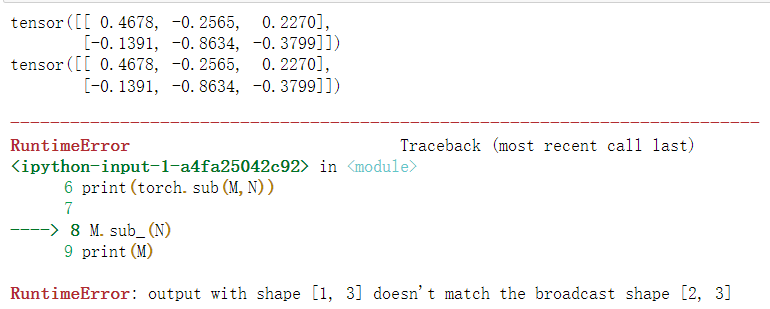


图1 矩阵减法程序运行结果

过程分析：形状不同的Tensor按元素发生减法运算时，可能触发了广播机制。由于 M 和N分别是1行3列和2行1列的矩阵，如果要计算 M-N，那么M中第一行的3个元素被广播（复制） 到了第二行，而 N 中第一列的2个元素被广播（复制）到了第二列和第三列。如此，就可以对2个2行3列的矩阵按元素相减。但是，sub\_()属于in\_place操作，in\_place操作不允许tensor使用广播机制那样来改变张量形状维度大小。

2.矩阵转置及相乘程序运行结果如图2所示。

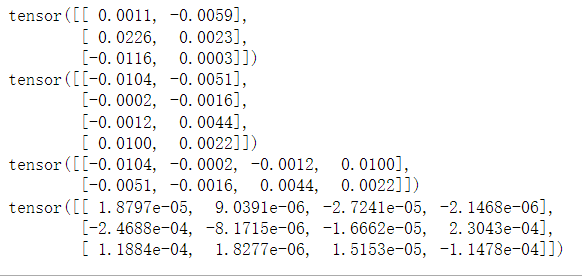


图2 矩阵转置及相乘程序运算结果

3.求梯度程序运算结果如图3所示。



图3 求梯度程序运算结果

过程分析：在进行求梯度运算时不允许张量对张量求导，只允许标量对张量求导，求导结果是和自变量同形的张量。由于y3不是一个标量，所以在调用backward()时需要传入一个和y3 同形的权重向量进行加权求和来得到一个标量，再对x进行求导。

#### 4.2 Logistic 回归实验

1.从零实现logistic回归程序运行结果如图4所示。

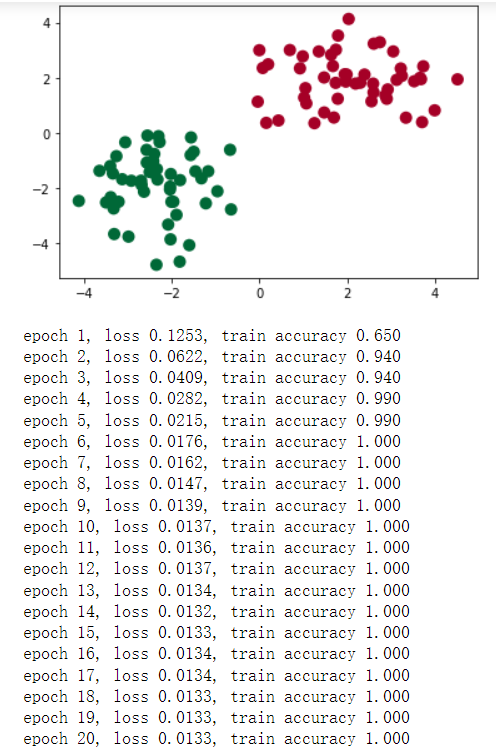


图4 从零实现logistic回归程序运行结果

结果分析：从零实现logistic回归实验中损失函数loss值成下降趋势逐步趋于0，训练集准确率几乎趋于1。

1. torch.nn实现logistic回归程序运行结果如图5所示。

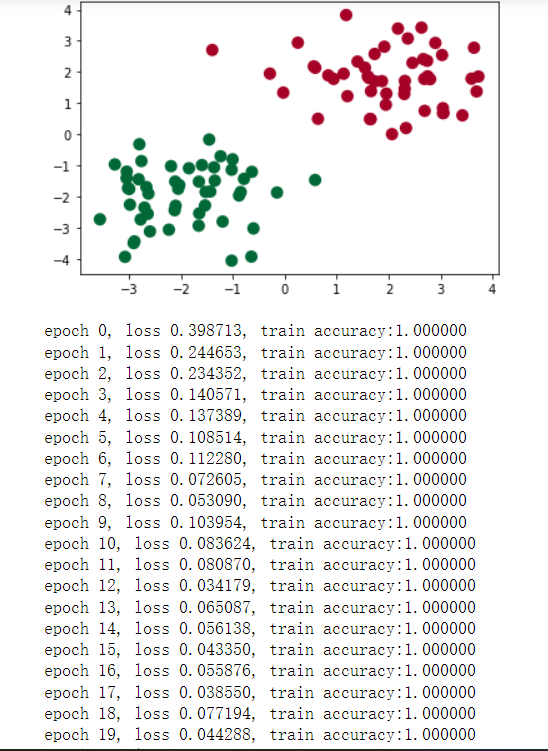


图5 torch.nn实现logistic回归程序运行结果

结果分析：torch.nn实现logistic回归实验中损失函数loss值成下降趋势逐步趋于0，训练集准确率几乎趋于1。

#### 4.3 Softmax 回归实验

1.从零实现Softmax回归程序运行结果如图6所示。

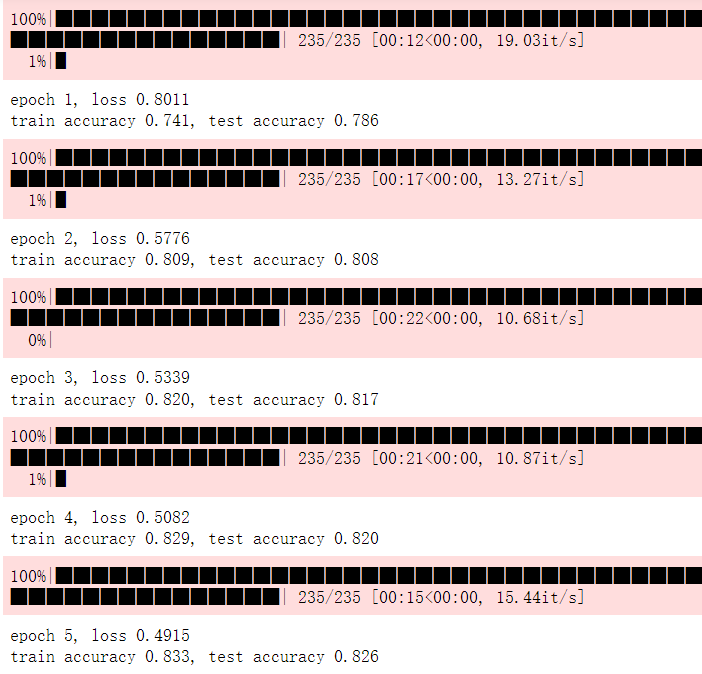


图6 从零实现softmax回归程序运行结果

结果分析：从零实现Softmax回归实验中训练集和测试集的损失函数loss值都呈下降趋势逐步趋于0，训练集准确率几乎趋于1，测试集准确率也几乎趋于1。

2.torch.nn实现Softmax回归程序运行结果如图7所示。

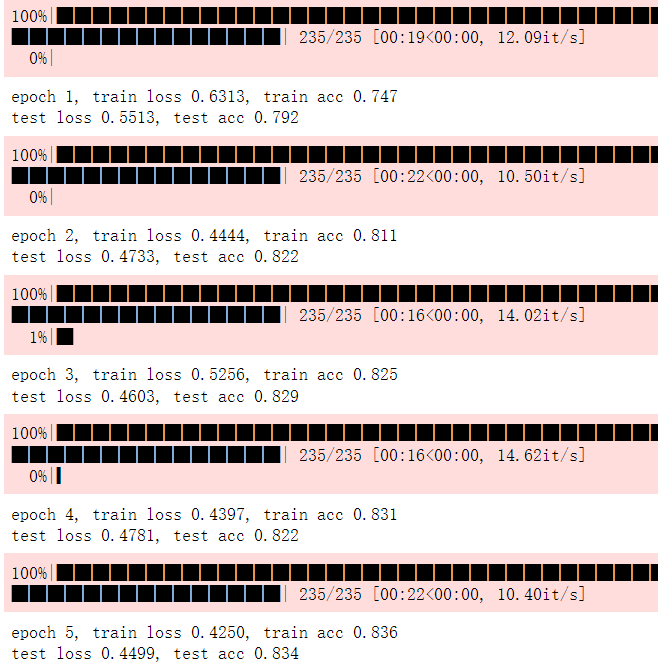


图7 torch.nn实现softmax回归程序运行结果

结果分析：torch.nn实现Softmax回归实验中训练集和测试集损失函数loss值都呈下降趋势逐步趋于0，训练集准确率几乎趋于1，测试集准确率也几乎趋于1。

## 五、实验心得体会

实验是个将课堂知识与动手实践融会贯通的部分，在实验的过程中我将logistic回归和softmax回归的整个逻辑有了更清楚的认识，在做回归实验问题时我也碰上了一些问题，最后在多次查看PPT和查找相关资源的帮助了解决了问题，实现了回归过程的实现。我的不足是代码编写的还是不太熟练，接下来会加强代码程序的练习。