

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **前馈神经网络实验** |
| **姓 名：** | **束天成** |
| **学 号：** | **22121501** |
| **日 期：** | **2022/7/22** |

## 一、实验内容

本次实验为前馈神经网络实验，本次实验主要包含使用不同的方法分别实现前馈神经网络的回归，二分类以及多分类，编程的语言为Python，其目的是了解前馈神经网络的基本操作任务，为后续深度学习的研究打下基础。

#### 1.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

前馈神经网络基本操作实验包含了数据库的构建、数据库的处理和训练集和测试集的loss分析部分，分别绘制不同的loss曲线，从而判断模型的好坏。训练集和测试集下的损失曲线基本上接近时，可以判断训练模型具有一定的泛化能力。

#### 1.2 利用torch.nn实现前馈神经网络解决上述回归、二分类、多分类任务

利用torch.nn实现前馈神经网络的原理和手动实现前馈神经网络的原理较为类似，有所不同的是该实验是通过自己构造torch.nn函数实现相关功能。由于该实验使用了torch.nn功能，因此和手动实现相比更具有模块化的特点。

#### 1.3 Softmax 回归实验

Softmax回归其实是 logistic 回归的一般形式，logistic 回归用于二分类，而 softmax回归用于多分类，其在数据处理中的适用性和通用性更广。本次实验主要需要使用人工构造的数据集并使用两种方法实现Softmax的回归，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。由于Softmax用于多分类问题，因此对于复杂问题的通用性更强。

## 二、实验设计

本次实验主要内容都为指定内容，均指定了实验需要的数据矩阵维度，矩阵构造方式，运算函数，使用的方法以及需要分析的内容，不包含过多需要自主设计的模型和方法等，因此不再赘述。

## 三、实验环境及实验数据集

Torch版本：1.5.1

操作系统：Windows10

硬件：

CPU：i5 9300h GPU：GTX1650

数据集：torch.nn，Fashion-MNIST数据集，nn.BCEWithLogitsLoss函数等

## 四、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。

注：为了代码片段尽量的美观、统一，建议附代码片段时只附加关键的片段，不要全部粘贴，并尽量使用下面提供的网站进行代码高亮等格式转换后再粘贴。

<http://www.planetb.ca/syntax-highlight-word>

#### 4.1 PyTorch基本操作实验

**4.1.1手动实现前馈神经网络回归任务**

手动实现前馈神经网络解决上述回归，分析实验结果并绘制训练集和测试集的loss曲线。

1. import torch
2. import torch.nn as nn
3. import numpy as np
4. import torchvision
5. import torchvision as transforms
6. import torch.utils.data as Data
7. import random
8. from IPython import display
9. from matplotlib import pyplot as plt
10. *#导入相关的数据包等*
11. *#自定义数据包*
12. num\_inputs = 500
13. num\_examples = 10000
14. true\_w = torch.ones(500,1)\*0.0056
15. true\_b = 0.028
16. a\_features = torch.tensor(np.random.normal(0, 1, (num\_examples, num\_inputs)), dtype=torch.float)
17. b\_labels = torch.mm(a\_features,true\_w) + true\_b
18. b\_labels += torch.tensor(np.random.normal(0, 0.01, size=b\_labels.size()), dtype=torch.float)
19. *#手动生成回归任务相关的训练集，数据集，服从回归实验的要求*
20. *#服从特定要求数据集*
21. trainfeatures =a\_features[:7000]
22. trainlabels = b\_labels[:7000]
23. print(trainfeatures.shape)
24. *#定义训练集*
25. testfeatures =a\_features[7000:]
26. testlabels = b\_labels[7000:]
27. print(testfeatures.shape)
28. *#定义测试集*
29. *#读取数据*
30. batch\_size = 50
31. dataset = Data.TensorDataset(trainfeatures, trainlabels)
32. *# 对训练的数据集进行整合*
33. *# 把 dataset 放入 DataLoader*
34. train\_iter = Data.DataLoader(
35. dataset=dataset, # torch TensorDataset format
36. batch\_size=batch\_size, # mini batch size
37. shuffle=True, # 打乱数据 ，训练过程一般需要打乱
38. num\_workers=0, # 多线程来读数据， 注意在Windows下需要设置为0
39. )
40. dataset = Data.TensorDataset(testfeatures, testlabels)
41. *# 对测试的数据集进行整合*
42. *# 把 dataset 放入 DataLoader*
43. test\_iter = Data.DataLoader(
44. dataset=dataset, # torch TensorDataset format
45. batch\_size=batch\_size, # mini batch size
46. shuffle=True, # 打乱数据 ，训练过程一般需要打乱
47. num\_workers=0, # 多线程来读数据， 注意在Windows下需要设置为0
48. )
49. num\_hiddens,num\_outputs = 256,1#设置隐藏层，输出层
50. W1 = torch.tensor(np.random.normal(0, 0.01, (num\_hiddens,num\_inputs)), dtype=torch.float32)
51. b1 = torch.zeros(1, dtype=torch.float32)
52. W2 = torch.tensor(np.random.normal(0, 0.01, (num\_outputs,num\_hiddens)), dtype=torch.float32)
53. b2 = torch.zeros(1, dtype=torch.float32)
54. params =[W1,b1,W2,b2]
55. for param in params:
56. param.requires\_grad\_(requires\_grad=True)
57. *#对权重和偏移大小进行整合，设置梯度值*
58. def relu(x):
59. x = torch.max(input=x,other=torch.tensor(0.0))
60. return x
61. def net(X):
62. X = X.view((-1,num\_inputs))
63. H = relu(torch.matmul(X,W1.t())+b1)
64. return torch.matmul(H,W2.t())+b2
65. def squared\_loss(y\_hat,y):
66. return (y\_hat-y.view(y\_hat.size()))\*\*2/2
67. *#定义最小化均方误差*
68. def SGD(paras,lr,batch\_size):
69. for param in params:
70. param.data -= lr \* param.grad/batch\_size
71. *#定义随机梯度下降法*
72. def evaluate\_accuracy(data\_iter,net):
73. acc\_sum,n = 0.0,0
74. for X,y in data\_iter:
75. result=model.forward(X)
76. acc\_sum+=(result.argmax(dim=1)==y).float().sum().item()
77. test\_1\_sum+=loss\_func(result,y).item()
78. n+=y.shape[0]
79. c+=1
80. acc\_sum += (net(X)==y).float().sum().item()
81. n+=y.shape[0]
82. return acc\_sum/n,test\_1\_sum/c
83. *#定义训练精度*
85. loss = torch.nn.MSELoss()
86. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
87. train\_ls = []
88. test\_ls = []
89. for epoch in range(num\_epochs):
90. train\_l\_sum, train\_acc\_num,n = 0.0,0.0,0
91. for X, y in train\_iter: # x和y分别是小批量样本的特征和标签
92. y\_hat = net(X)
93. l = loss(y\_hat, y.view(-1,1)) # l是有关小批量X和y的损失
94. *#梯度清零*
95. if optimizer is not None:
96. optimizer.zero\_grad()
97. elif params is not None and params[0].grad is not None:
98. for param in params:
99. param.grad.data.zero\_()
100. l.backward() # 小批量的损失对模型参数求梯度
101. if optimizer is None:
102. SGD(params,lr,batch\_size)
103. else:
104. optimizer.step()
105. train\_labels = trainlabels.view(-1,1)
106. test\_labels = testlabels.view(-1,1)
107. train\_ls.append(loss(net(trainfeatures),train\_labels).item())
108. test\_ls.append(loss(net(testfeatures),test\_labels).item())
109. print('epoch %d, train\_loss %.6f,test\_loss %f'%(epoch+1, train\_ls[epoch], test\_ls[epoch]))
110. return train\_ls,test\_ls
111. lr = 0.05
112. num\_epochs = 200
113. train\_loss,test\_loss = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params,lr)
114. *#开始训练*
115. x = np.linspace(0,len(train\_loss),len(train\_loss))
116. plt.plot(x,train\_loss,label="train\_loss",linewidth=1.5)
117. plt.plot(x,test\_loss,label="test\_loss",linewidth=1.5)
118. plt.xlabel("epoch")
119. plt.ylabel("loss")
120. plt.legend()
121. plt.show()

**4.1.2手动实现前馈神经网络二分类任务**

利用torch.nn实现前馈神经网络解决上述回归、二分类、多分类任务，分析实验结果并绘制训练集和测试集的loss曲线。

由于该实验大部分代码和4.1.1类似，因此这里只展示不一样的部分。

**数据集的构造部分：**

1. *#自定义数据包*
2. num\_inputs = 200
3. num\_examples = 10000
4. a1 = torch.normal(0.1,1,(10000, num\_inputs))
5. b1 = torch.ones(10000,1)
6. a1\_train = a1[:7000]
7. a1\_test = a1[7000:] *#定义数据集A及其对应的标签*
8. a2 = torch.normal(-0.1,1,(10000, num\_inputs))
9. b2 = torch.zeros(10000,1)
10. a2\_train = a2[:7000]
11. a2\_test = a2[7000:] *#定义数据集B及其对应的标签*
12. trainfeatures = torch.cat((a1\_train,a2\_train), 0).type(torch.FloatTensor)*#合并训练集A*
13. trainlabels = torch.cat((b1[:7000], b2[:7000]), 0).type(torch.FloatTensor)*#标签A*
14. testfeatures = torch.cat((a1\_test,a2\_test), 0).type(torch.FloatTensor)  *#合并训练集B*
15. testlabels = torch.cat((b1[7000:], b2[7000:]), 0).type(torch.FloatTensor)*#标签B*

**训练模型和训练精度的构造部分：**

1. *#定义训练精度*
2. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
3. train\_ls = []
4. test\_ls = []
5. for epoch in range(num\_epochs):
6. train\_l\_sum, train\_acc\_num,n = 0.0,0.0,0
7. for X, y in train\_iter: *# x和y分别是小批量样本的特征和标签*
8. y\_hat = net(X)
9. l = loss(y\_hat, y.view(-1,1)) *# l是有关小批量X和y的损失*
10. *#梯度清零*
11. if optimizer is not None:
12. optimizer.zero\_grad()
13. elif params is not None and params[0].grad is not None:
14. for param in params:
15. param.grad.data.zero\_()
16. l.backward() *# 小批量的损失对模型参数求梯度*
17. if optimizer is None:
18. SGD(params,lr,batch\_size)
19. else:
20. optimizer.step()
21. train\_l\_sum += l.item()\*y.shape[0]
22. n+= y.shape[0]
23. train\_acc\_num += (y\_hat.argmax(dim=1)==y).sum().item()
24. test\_acc = evaluate\_accuracy(test\_iter,net)
25. test\_labels = testlabels.view(-1,1)
26. train\_ls.append(train\_l\_sum/n)
27. test\_ls.append(loss(net(testfeatures),test\_labels).item())
28. print('epoch %d, train\_loss %.6f,test\_loss %.6f,train\_acc %.3f'%(epoch+1, train\_ls[epoch],test\_ls[epoch],train\_acc\_num/n))
29. return train\_ls,test\_ls
30. lr = 0.01
31. num\_epochs = 200
32. train\_loss,test\_loss = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params,lr)
33. *#开始训练*
34. x = np.linspace(0,len(train\_loss),len(train\_loss))
35. plt.plot(x,train\_loss,label="train\_loss",linewidth=1.5)
36. plt.plot(x,test\_loss,label="test\_loss",linewidth=1.5)
37. plt.xlabel("epoch")
38. plt.ylabel("loss")
39. plt.legend()
40. plt.show()

**4.1.3手动实现前馈神经网络多分类任务**

手动实现前馈神经网络解决多分类任务，分析实验结果并绘制训练集和测试集的loss曲线，由于该实验大部分代码和4.1.1类似，因此这里只展示不一样的部分。

**数据集的构造部分：**

1. *#下载MNIST手写数据集*
2. mnist\_train = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=True,
3. download=True, transform=transforms.ToTensor())
4. mnist\_test = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=False,
5. download=True, transform=transforms.ToTensor())
6. *#读取数据*
7. batch\_size = 32
8. train\_iter = torch.utils.data.DataLoader(mnist\_train, batch\_size=batch\_size, shuffle=True,num\_workers=0)
9. test\_iter = torch.utils.data.DataLoader(mnist\_test, batch\_size=batch\_size, shuffle=False,num\_workers=0)
10. *#初始化参数*
11. num\_inputs,num\_hiddens,num\_outputs = 784,256,10

**定义loss函数和训练精度：**

1. *#定义测试集loss和精准度acc*
2. def evaluate\_loss(data\_iter,net):
3. acc\_sum,loss\_sum,n = 0.0,0.0,0
4. for X,y in data\_iter:
5. y\_hat = net(X)
6. acc\_sum += (y\_hat.argmax(dim=1)==y).sum().item()
7. l = loss(y\_hat,y) # l是有关小批量X和y的损失
8. loss\_sum += l.sum().item()\*y.shape[0]
9. n+=y.shape[0]
10. return acc\_sum/n,loss\_sum/n

**定义训练模型函数：**

1. *#定义模型训练函数*
2. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
3. train\_ls = []
4. test\_ls = []
5. for epoch in range(num\_epochs): *# 训练模型一共需要num\_epochs个迭代周期*
6. train\_l\_sum, train\_acc\_num,n = 0.0,0.0,0
7. *# 在每一个迭代周期中，会使用训练数据集中所有样本一次*
8. for X, y in train\_iter: *# x和y分别是小批量样本的特征和标签*
9. y\_hat = net(X)
10. l = loss(y\_hat, y).sum() *# l是有关小批量X和y的损失*
11. *#梯度清零*
12. if optimizer is not None:
13. optimizer.zero\_grad()
14. elif params is not None and params[0].grad is not None:
15. for param in params:
16. param.grad.data.zero\_()
17. l.backward() *# 小批量的损失对模型参数求梯度*
18. if optimizer is None:
19. SGD(params,lr)
20. else:
21. optimizer.step()
22. *#计算每个epoch的loss*
23. train\_l\_sum += l.item()\*y.shape[0]
24. *#计算训练样本的准确率*
25. train\_acc\_num += (y\_hat.argmax(dim=1)==y).sum().item()
26. *#每一个epoch的所有样本数*
27. n+= y.shape[0]
28. train\_ls.append(train\_l\_sum/n)
29. test\_acc,test\_l = evaluate\_loss(test\_iter,net)
30. test\_ls.append(test\_l)
31. print('epoch %d, train\_loss %.6f,test\_loss %f,train\_acc %.6f,test\_acc %.6f'%(epoch+1, train\_ls[epoch],test\_ls[epoch],train\_acc\_num/n,test\_acc))
32. return train\_ls,test\_ls

#### 4.2 利用torch.nn实现前馈神经网络解决回归问题

**4.2.1 利用torch.nn实现前馈神经网络解决上述回归、二分类、多分类任务**

利用torch.nn实现前馈神经网络解决回归问题，分析实验结果并绘制训练集和测试集的loss曲线

1. import torch
2. from torch import nn
3. import numpy as np
4. import matplotlib.pyplot as plt
5. from torch.utils.data import DataLoader,TensorDataset
6. from sklearn.model\_selection import train\_test\_split
7. from collections import OrderedDict
8. from torch.nn import init
9. *#导入必要的数据包等其他相关内容*
10. num\_input ,num\_example = 500,10000
11. true\_w = torch.ones(1,num\_input)\*0.0056
12. true\_b = 0.028
13. a = torch.tensor(np.random.normal(0,0.001,size  = (num\_example,num\_input)),dtype = torch.float32)
14. b = torch.mm(x\_data,true\_w.t()) +true\_b
15. b += torch.normal(0,0.001,b.shape)
16. *#定义总的数据集*
17. a1\_train = a[:7000]
18. a1\_test  = a[7000:]
19. b1\_train = b[:7000]
20. b1\_test  = b[7000:]
21. batch\_size = 50
22. *#定义相关训练集和其他内容*
23. train\_dataset = TensorDataset(a1\_train,b1\_train)
24. train\_iter = DataLoader(
25. dataset = train\_dataset,
26. batch\_size = batch\_size,## mini batch size
27. shuffle = True,#打乱相关数据
28. num\_workers = 0,# 多线程来读数据,在Windows下需要设置为0
29. )
30. *#定义相关测试集和其他内容*
31. test\_dataset = TensorDataset(a1\_test,b1\_test)
32. test\_iter = DataLoader(
33. dataset = test\_dataset,
34. batch\_size = batch\_size,
35. shuffle = True,
36. num\_workers = 0,
37. )
38. model= nn.Sequential(OrderedDict([
39. ('linear1',nn.Linear(num\_input,256)),
40. ('linear2',nn.Linear(256,128)),
41. ('linear3',nn.Linear(128,1)),
42. ])
43. )
44. for param in model.parameters():
45. init.normal\_(param,mean = 0 ,std = 0.001)
46. *#定义部分参数的初始值*
47. lr = 0.001
48. loss = nn.MSELoss()
49. *#定义损失函数*
50. optimizer = torch.optim.SGD(model.parameters(),lr)
51. *#定义优化算法*
52. def train(model,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr):
53. train\_ls,test\_ls = [],[]
54. for epoch in range(num\_epochs):
55. train\_ls\_sum ,test\_ls\_sum = 0,0
56. for x,y in train\_iter:
57. y\_pred = model(x)
58. l = loss(y\_pred,y)
59. optimizer.zero\_grad()#梯度先清零，和手动不一样
60. l.backward()
61. optimizer.step()
62. train\_ls\_sum += l.item()
63. for x ,y in test\_iter:#训练集函数
64. y\_pred = model(x)
65. l = loss(y\_pred,y)
66. test\_ls\_sum +=l.item()
67. train\_ls.append(train\_ls\_sum)
68. test\_ls.append(test\_ls\_sum)
69. print('epoch %d,train\_loss %.6f,test\_loss %f'%(epoch+1, train\_ls[epoch],test\_ls[epoch]))
70. return train\_ls,test\_ls
71. *#定义测试函数*
72. num\_epochs = 200
73. train\_loss ,test\_loss = train(model,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr)
74. *#设置迭代次数等相关参数*
75. x = np.linspace(0,len(train\_loss),len(train\_loss))
76. plt.plot(x,train\_loss,label="train\_loss",linewidth=1.5)
77. plt.plot(x,test\_loss,label="test\_loss",linewidth=1.5)
78. plt.xlabel("epoch")
79. plt.ylabel("loss")
80. plt.legend()
81. plt.show()

**4.2.2 利用torch.nn实现前馈神经网络解决二分类任务**

利用torch.nn实现前馈神经网络解决二分类任务，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，分析实验结果并绘制训练集和测试集的loss曲线。由于该实验大部分代码和4.2.1类似，因此这里只展示不一样的部分。

**定义输入部分函数：**

1. import torch
2. import torch.nn as nn
3. from torch.utils.data import TensorDataset,DataLoader
4. from torch.nn import init
5. import torch.optim as optim
6. from sklearn.model\_selection import train\_test\_split
7. import numpy as np
8. import matplotlib.pyplot as plt
9. *#导入需要的数据包*
10. num\_inputs,num\_example = 200,10000
11. a1 = torch.normal(2,1,(num\_example,num\_inputs))
12. b1 = torch.ones((num\_example,1))
13. a2 = torch.normal(-2,1,(num\_example,num\_inputs))
14. b2 = torch.zeros((num\_example,1))
15. a = torch.cat((a1,a2),dim=0)
16. b = torch.cat((b1,b2),dim = 0)
17. *#合并相应的数据集及其标签*
18. a1\_train = a[:7000]
19. a1\_test  = a[7000:]
20. b1\_train = b[:7000]
21. b1\_test  = b[7000:]
22. *#分别对应训练集和测试集*
23. *#定义总的数据集*

**定义模型部分：**

1. num\_input,num\_hidden,num\_output = 200,256,1*#定义模型参数和模型结构等相关内容*
2. class net(nn.Module):
3. def \_\_init\_\_(self,num\_input,num\_hidden,num\_output):
4. super(net,self).\_\_init\_\_()*#定义模型的输入和输出内容等*
5. self.linear1 = nn.Linear(num\_input,num\_hidden,bias =False)
6. self.linear2 = nn.Linear(num\_hidden,num\_output,bias=False)
7. def forward(self,input):*#定义模型的前向传播过程*
8. out = self.linear1(input)
9. out = self.linear2(out)
10. return out
12. model = net(num\_input,num\_hidden,num\_output)
13. print(model)

**定义参数训练部分：**

1. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size):
2. train\_ls,test\_ls,train\_acc,test\_acc = [],[],[],[]
3. for epoch in range(num\_epochs):
4. train\_ls\_sum,train\_acc\_sum,n = 0,0,0
5. for x,y in train\_iter:#训练集的训练函数
6. y\_pred = model(x)
7. l = loss(y\_pred,y)
8. optimizer.zero\_grad()
9. l.backward()
10. optimizer.step()
11. train\_ls\_sum +=l.item()
12. train\_acc\_sum += (((y\_pred>0.5)==y)+0.0).sum().item()
13. n += y\_pred.shape[0]
14. train\_ls.append(train\_ls\_sum)
15. train\_acc.append(train\_acc\_sum/n)
17. test\_ls\_sum,test\_acc\_sum,n = 0,0,0
18. for x,y in test\_iter:#测试集的训练函数
19. y\_pred = model(x)
20. l = loss(y\_pred,y)
21. test\_ls\_sum +=l.item()
22. test\_acc\_sum += (((y\_pred>0.5)==y)+0.0).sum().item()
23. n += y\_pred.shape[0]
24. test\_ls.append(test\_ls\_sum)
25. test\_acc.append(test\_acc\_sum/n)
26. print('epoch %d, train\_loss %.6f,test\_loss %f, train\_acc %.6f,test\_acc %f'
27. %(epoch+1, train\_ls[epoch],test\_ls[epoch], train\_acc[epoch],test\_acc[epoch]))
28. return train\_ls,test\_ls,train\_acc,test\_acc

**4.2.3 利用torch.nn实现前馈神经网络解决二分类任务**

利用torch.nn实现前馈神经网络解决多分类任务，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，分析实验结果并绘制训练集和测试集的loss曲线。由于该实验大部分代码和4.2.1类似，因此这里只展示不一样的部分。

**导入与下载相关数据**

1. import torch
2. import numpy as np
3. from torch import nn
4. from torchvision.datasets import MNIST
5. import torchvision.transforms  as transforms
6. import matplotlib.pyplot as plt
7. from sklearn.model\_selection import train\_test\_split
8. from torch.utils.data import DataLoader
9. from torch import nn
10. *#导入进行多回归任务所需要的包*
11. mnist\_train = MNIST(
12. root='./datasets/MNIST',
13. train = True,
14. download =True,
15. transform=transforms.ToTensor())
16. *#导入训练集所需要的测试数据*
17. mnist\_test = MNIST(
18. root='./datasets/MNIST',
19. train = False,
20. download =True,
21. transform=transforms.ToTensor())
22. *#导入测试集所需要的测试数据*

**输入层、输出层已经函数模型等**

1. *#定义输入层，输出层等相关内容*
2. num\_input,num\_hidden1,num\_hidden2,num\_output = 28\*28,512,256,10
3. class MM(nn.Module):*#定义模型训练函数*
4. def \_\_init\_\_(self,num\_input,num\_hidden1,num\_hidden2,num\_output):
5. super(MM,self).\_\_init\_\_()
6. self.linear1 = nn.Linear(num\_input,num\_hidden1)
7. self.linear2 = nn.Linear(num\_hidden1,num\_hidden2)
8. self.linear3 = nn.Linear(num\_hidden2,num\_output)*#定义几个神经网络层的关系*
9. def forward(self,input):
10. input = input.view(-1,784)
11. out = self.linear1(input)
12. out = self.linear2(out)
13. out = self.linear3(out)*#定义传递关系*
14. return out

**训练函数模型与测试函数模型**

1. def train(net,train\_iter,test\_iter,loss,num\_epochs):
2. train\_ls,test\_ls,train\_acc,test\_acc = [],[],[],[]
3. for epoch in range(num\_epochs):
4. train\_ls\_sum,train\_acc\_sum,n = 0,0,0
5. for x,y in train\_iter:#定义训练函数模型
6. y\_pred = net(x)
7. l = loss(y\_pred,y)
8. optimizer.zero\_grad()
9. l.backward()
10. optimizer.step()
11. train\_ls\_sum +=l.item()
12. train\_acc\_sum += (y\_pred.argmax(dim = 1)==y).sum().item()
13. n += y\_pred.shape[0]
14. train\_ls.append(train\_ls\_sum)
15. train\_acc.append(train\_acc\_sum/n)
17. test\_ls\_sum,test\_acc\_sum,n = 0,0,0#定义测试函数模型
18. for x,y in test\_iter:
19. y\_pred = net(x)
20. l = loss(y\_pred,y)
21. test\_ls\_sum +=l.item()
22. test\_acc\_sum += (y\_pred.argmax(dim = 1)==y).sum().item()
23. n += y\_pred.shape[0]
24. test\_ls.append(test\_ls\_sum)
25. test\_acc.append(test\_acc\_sum/n)
26. print('epoch %d, train\_loss %.6f,test\_loss %f, train\_acc %.6f,test\_acc %f'
27. %(epoch+1, train\_ls[epoch],test\_ls[epoch], train\_acc[epoch],test\_acc[epoch]))
28. return train\_ls,test\_ls,train\_acc,test\_acc

#### 4.3 在多分类实验的基础上使用至少三种不同的激活函数

**4.3.1 在多分类实验的基础上使用至少三种不同的激活函数，对比使用不同激活函数的实验结果**

**导入相关材料**

1. import torch
2. import numpy as np
3. import random
4. from IPython import display
5. from matplotlib import pyplot as plt
6. from torchvision.datasets import MNIST
7. from torch import nn
8. import torch.utils.data as Data
9. import torch.optim as optim
10. from torch.nn import init
11. import torchvision
12. from torch.utils.data import DataLoader
13. import torchvision.transforms as transforms
14. *#导入相关的数据包等相关材料*

**定义不同的激活函数和模型前向传播过程**

1. class FlattenLayer(torch.nn.Module):
2. def \_\_init\_\_(self):
3. super(FlattenLayer, self).\_\_init\_\_()
4. def forward(self, x):
5. return x.view(x.shape[0],-1)
6. *#定义模型的前向传播过程*
7. *#模型定义和参数初始化*
8. num\_inputs,num\_hiddens,num\_outputs = 784,256,10
9. def use\_ReLU():
10. net = nn.Sequential(
11. FlattenLayer(),
12. nn.Linear(num\_inputs,num\_hiddens),
13. nn.ReLU(),
14. nn.Linear(num\_hiddens,num\_outputs)
15. )
16. return net
17. *#实现ReLU激活函数*
18. def use\_ELU():
19. net = nn.Sequential(
20. FlattenLayer(),
21. nn.Linear(num\_inputs,num\_hiddens),
22. nn.ELU(),
23. nn.Linear(num\_hiddens,num\_outputs)
24. )
25. return net
26. *#实现ELU激活函数*
27. def use\_Sigmoid():
28. net = nn.Sequential(
29. FlattenLayer(),
30. nn.Linear(num\_inputs,num\_hiddens),
31. nn.Sigmoid(),
32. nn.Linear(num\_hiddens,num\_outputs)
33. )
34. return net
35. *#实现Sigmoid激活函数*
36. def init\_params(net):
37. for params in net.parameters():
38. init.normal\_(params,mean=0,std=0.01)
39. return  torch.optim.SGD(net.parameters(),lr)
40. *#对训练参数进行初始化*

**定义训练函数等相关内容**

1. *#定义模型训练函数*
2. def train(model,train\_loader,test\_loader,loss\_fn,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
3. train\_ls = []
4. test\_ls = []
5. for epoch in range(num\_epochs): *# 训练模型一共需要num\_epochs个迭代周期*
6. train\_loss\_sum, train\_acc\_num,total\_examples = 0.0,0.0,0
7. for x, y in train\_loader: *# x和y分别是小批量样本的特征和标签*
8. y\_pred = model(x)
9. loss = loss\_fn(y\_pred, y)  *#计算损失*
10. optimizer.zero\_grad() *# 梯度清零*
11. loss.backward()  *# 反向传播*
12. optimizer.step() *#梯度更新*
13. total\_examples += y.shape[0]
14. train\_loss\_sum += loss.item()
15. train\_acc\_num += (y\_pred.argmax(dim=1)==y).sum().item()
16. train\_ls.append(train\_loss\_sum)
17. test\_acc,test\_loss = evaluate\_testset(test\_loader,model)
18. test\_ls.append(test\_loss)
19. print('epoch %d, train\_loss %.6f,test\_loss %f,train\_acc %.6f,test\_acc %.6f'%(epoch+1, train\_ls[epoch],test\_ls[epoch],train\_acc\_num/total\_examples,test\_acc))
20. return train\_ls,test\_ls

#### 4.4 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

**4.4.1使用不同的隐藏层层数，进行对比实验并分析实验结果**

1. import torch
2. import numpy as np
3. from torch import nn
4. from torchvision.datasets import MNIST
5. import torchvision.transforms  as transforms
6. import matplotlib.pyplot as plt
7. from sklearn.model\_selection import train\_test\_split
8. from torch.utils.data import DataLoader
9. from torch import nn
10. *#导入进行多回归任务所需要的包*
11. mnist\_train = MNIST(
12. root='./datasets/MNIST',
13. train = True,
14. download =True,
15. transform=transforms.ToTensor())
16. *#导入训练集所需要的测试数据*
17. mnist\_test = MNIST(
18. root='./datasets/MNIST',
19. train = False,
20. download =True,
21. transform=transforms.ToTensor())
22. *#导入测试集所需要的测试数据*
23. *#读取数据*
24. batch\_size =64
25. *# 把训练数据放入 DataLoader*
26. train\_iter = DataLoader(
27. dataset = mnist\_train,
28. batch\_size = batch\_size,
29. shuffle = True,
30. )
31. *# 把测试数据放入 DataLoader*
32. test\_iter = DataLoader(
33. dataset = mnist\_test,
34. batch\_size = batch\_size,
35. shuffle = True,
36. )
37. next(iter(train\_iter))[0].shape
38. next(iter(test\_iter))[0].shape
39. *#训练次数和学习率*
40. num\_epochs ,lr = 50, 0.01
41. num\_inputs, num\_outputs = 28\*28, 10
42. class MM1(nn.Module):
43. def \_\_init\_\_(self,num\_inputs=784, num\_outputs=10, num\_hiddens=100):
44. super(MM1,self).\_\_init\_\_()
45. self.linear1 = nn.Linear(num\_inputs,num\_hiddens)
46. self.relu = nn.ReLU()
47. self.linear2 = nn.Linear(num\_hiddens,num\_outputs)
49. def forward(self,x):
50. x = self.linear1(x)
51. x = self.relu(x)
52. x = self.linear2(x)
53. y = self.relu(x)
54. return y
55. class MM2(nn.Module):
56. def \_\_init\_\_(self,num\_inputs=784, num\_outputs=10, num\_hiddens1=100, num\_hiddens2=100):
57. super(MM2,self).\_\_init\_\_()
58. self.linear1 = nn.Linear(num\_inputs,num\_hiddens1)
59. self.relu = nn.ReLU()
60. self.linear2 = nn.Linear(num\_hiddens1,num\_hiddens2)
61. self.linear3 = nn.Linear(num\_hiddens2,num\_outputs)
63. def forward(self,x):
64. x = self.linear1(x)
65. x = self.relu(x)
66. x = self.linear2(x)
67. x = self.relu(x)
68. x = self.linear3(x)
69. y = self.relu(x)
70. return y
72. class MM3(nn.Module):
73. def \_\_init\_\_(self,num\_inputs=784, num\_outputs=10, num\_hiddens1=100, num\_hiddens2=100, num\_hiddens3=100):
74. super(MM3,self).\_\_init\_\_()
75. self.linear1 = nn.Linear(num\_inputs,num\_hiddens1)
76. self.relu = nn.ReLU()
77. self.linear2 = nn.Linear(num\_hiddens1,num\_hiddens2)
78. self.linear3 = nn.Linear(num\_hiddens2,num\_hiddens3)
79. self.linear4 = nn.Linear(num\_hiddens3,num\_outputs)
81. def forward(self,x):
82. x = self.linear1(x)
83. x = self.relu(x)
84. x = self.linear2(x)
85. x = self.relu(x)
86. x = self.linear3(x)
87. x = self.relu(x)
88. x = self.linear4(x)
89. y = self.relu(x)
90. return y
92. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
93. train\_ls, test\_ls = [], []
94. for epoch in range(num\_epochs):
95. ls, count = 0, 0
96. for X,y in train\_iter:
97. X = X.reshape(-1,num\_inputs)  *#[32, 28, 28]  ->  [32, 784]*
98. l=loss(net(X),y)
99. optimizer.zero\_grad()
100. l.backward()
101. optimizer.step()
102. ls += l.item()\*y.shape[0]
103. train\_ls.append(ls)
104. ls, count = 0, 0
105. for X,y in test\_iter:
106. X = X.reshape(-1,num\_inputs)
107. l=loss(net(X),y)
108. ls += l.item()\*y.shape[0]
109. test\_ls.append(ls)
110. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
111. return train\_ls,test\_ls
112. total\_net = [MM1,MM2,MM3]
113. Train\_loss, Test\_loss = [], []
114. *#定义损失函数*
115. loss = nn.CrossEntropyLoss()
116. for cur\_net in total\_net:
117. net = cur\_net()
118. for param in net.parameters():
119. nn.init.normal\_(param,mean=0, std= 0.01)
120. optimizer = torch.optim.SGD(net.parameters(),lr = 0.001)
122. train\_ls, test\_ls = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,net.parameters,lr,optimizer)
123. Train\_loss.append(train\_ls)
124. Test\_loss.append(test\_ls)
126. x = np.linspace(0,len(train\_ls),len(train\_ls))
127. plt.figure(figsize=(10,8))
128. for i in range(0,3):
129. plt.plot(x,Train\_loss[i],label= f'with {i+1} hiddens layers:',linewidth=1.5)
130. plt.xlabel('epoch')
131. plt.ylabel('loss')
132. plt.legend()
133. plt.title('train loss')
134. plt.show()

**4.4.2使用不同的隐藏单元数，进行对比实验并分析实验结果**

1. import torch
2. import numpy as np
3. from torch import nn
4. from torchvision.datasets import MNIST
5. import torchvision.transforms  as transforms
6. import matplotlib.pyplot as plt
7. from sklearn.model\_selection import train\_test\_split
8. from torch.utils.data import DataLoader
9. from torch import nn
10. *#导入进行多回归任务所需要的包*
11. mnist\_train = MNIST(
12. root='./datasets/MNIST',
13. train = True,
14. download =True,
15. transform=transforms.ToTensor())
16. *#导入训练集所需要的测试数据*
17. mnist\_test = MNIST(
18. root='./datasets/MNIST',
19. train = False,
20. download =True,
21. transform=transforms.ToTensor())
22. *#导入测试集所需要的测试数据*
23. *#读取数据*
24. batch\_size =64
25. *# 把训练数据放入 DataLoader*
26. train\_iter = DataLoader(
27. dataset = mnist\_train,
28. batch\_size = batch\_size,
29. shuffle = True,
30. )
31. *# 把测试数据放入 DataLoader*
32. test\_iter = DataLoader(
33. dataset = mnist\_test,
34. batch\_size = batch\_size,
35. shuffle = True,
36. )
37. #训练次数和学习率，迭代次数等相关内容
38. num\_epochs ,lr = 50, 0.01
39. class MM1(nn.Module):
40. def \_\_init\_\_(self,num\_inputs=784, num\_outputs=10, num\_hiddens=100):
41. super(MM1,self).\_\_init\_\_()
42. self.linear1 = nn.Linear(num\_inputs,num\_hiddens)
43. self.relu = nn.ReLU()
44. self.linear2 = nn.Linear(num\_hiddens,num\_outputs)
45. def forward(self,x):
46. x = self.linear1(x)
47. x = self.relu(x)
48. x = self.linear2(x)
49. y = self.relu(x)
50. return y
51. *#定义单层神经网络*
52. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
53. train\_ls, test\_ls = [], []
54. for epoch in range(num\_epochs):
55. ls, count = 0, 0
56. for X,y in train\_iter:
57. X = X.reshape(-1,num\_inputs)
58. l=loss(net(X),y)
59. optimizer.zero\_grad()
60. l.backward()
61. optimizer.step()
62. ls += l.item()
63. count += y.shape[0]
64. train\_ls.append(ls)
65. ls, count = 0, 0
66. for X,y in test\_iter:
67. X = X.reshape(-1,num\_inputs)
68. l=loss(net(X),y)
69. ls += l.item()
70. count += y.shape[0]
71. test\_ls.append(ls)
72. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
73. return train\_ls,test\_ls
74. hiddens = [50,100,150]
75. #定义输入层神经元个数和输出层神经元个数
76. num\_inputs, num\_outputs = 784, 10
77. #定义损失函数
78. loss = nn.CrossEntropyLoss()
79. Train\_loss, Test\_loss = [], []
80. for cur\_hiddens in hiddens:
81. net = MM1(num\_inputs, num\_outputs, cur\_hiddens)
82. optimizer = torch.optim.SGD(net.parameters(),lr = 0.001)
83. for param in net.parameters():
84. nn.init.normal\_(param,mean=0, std= 0.01)
85. train\_ls, test\_ls = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,net.parameters,lr,optimizer)
86. Train\_loss.append(train\_ls)
87. Test\_loss.append(test\_ls)
89. x = np.linspace(0,len(train\_ls),len(train\_ls))
90. plt.figure(figsize=(10,8))
91. for i in range(0,len(hiddens)):
92. plt.plot(x,Train\_loss[i],label= f'Neuronss:{ hiddens[i]}',linewidth=1.5)
93. plt.xlabel('epoch')
94. plt.ylabel('loss')
95. plt.legend()
96. plt.title('Train loss vs different hiddens')
97. plt.show()

#### 4.5 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

**4.5.1在多分类任务实验中手动实现**

1. import torch
2. import numpy as np
3. import torch.nn as nn
4. import random
5. from IPython import display
6. from matplotlib import pyplot as plt
7. import torchvision
8. import torchvision.transforms as transforms
9. *#导入相关的数据包等*
10. *#下载MNIST手写数据集*
11. mnist\_train = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=True,
12. download=True, transform=transforms.ToTensor())#下载相关内容
13. mnist\_test = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=False,
14. download=True, transform=transforms.ToTensor())
15. *#读取数据*
16. batch\_size = 256
17. train\_iter = torch.utils.data.DataLoader(mnist\_train, batch\_size=batch\_size, shuffle=True,
18. num\_workers=0)
19. test\_iter = torch.utils.data.DataLoader(mnist\_test, batch\_size=batch\_size, shuffle=False,
20. num\_workers=0)
21. *#把相应的参数放到指定位置*
22. *#初始化参数*
23. num\_inputs,num\_hiddens,num\_outputs = 784,256,10
24. *#设置输入、隐藏层和输出层*
25. def Datanum():
26. A1 = torch.tensor(np.random.normal(0, 0.01, (num\_hiddens,num\_inputs)), dtype=torch.float32)
27. B1 = torch.zeros(1, dtype=torch.float32)
28. A2 = torch.tensor(np.random.normal(0, 0.01, (num\_outputs,num\_hiddens)), dtype=torch.float32)
29. B2 = torch.zeros(1, dtype=torch.float32)
30. params =[A1,B1,A2,B2]
31. for param in params:
32. param.requires\_grad\_(requires\_grad=True)
33. return A1,B1,A2,B2
34. num\_epochs=30
35. lr = 0.001
36. *#定义手动实现的规则*
37. def dropout(X, drop\_prob):
38. X = X.float()
39. assert 0 <= drop\_prob <= 1
40. keep\_prob = 1 - drop\_prob
41. if keep\_prob == 0:
42. return torch.zeros\_like(X)
43. mask = (torch.rand(X.shape) < keep\_prob).float()
44. return mask \* X / keep\_prob
45. *#定义drop损失大小*
46. def net(X, is\_training=True):
47. X = X.view(-1, num\_inputs)
48. H1 = (torch.matmul(X, A1.t()) + B1).relu()
49. if is\_training:
50. H1 = dropout(H1, drop\_prob1)
51. return (torch.matmul(H1,A2.t()) + B2).relu()
52. #定义相关模型
53. #定义模型训练函数
54. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr=None,optimizer=None):
55. train\_ls, test\_ls = [], []
56. for epoch in range(num\_epochs):
57. ls, count = 0, 0
58. for X,y in train\_iter:
59. l=loss(net(X),y)
60. optimizer.zero\_grad()
61. l.backward()
62. optimizer.step()
63. ls += l.item()
64. count += y.shape[0]
65. train\_ls.append(ls)
66. ls, count = 0, 0
67. for X,y in test\_iter:
68. l=loss(net(X,is\_training=True),y)
69. ls += l.item()
70. count += y.shape[0]
71. test\_ls.append(ls)
72. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
73. return train\_ls,test\_ls
74. drop\_probs = np.arange(0,0.5,0.1)
75. Train\_ls, Test\_ls = [], []
76. for drop\_prob in drop\_probs:
77. drop\_prob1 = drop\_prob
78. A1,B1,A2,B2 = Datanum()
79. loss = nn.CrossEntropyLoss()
80. optimizer = torch.optim.SGD([A1,B1,A2,B2],lr = 0.001)
81. train\_ls, test\_ls =  train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr,optimizer)
82. Train\_ls.append(train\_ls)
83. Test\_ls.append(test\_ls)
84. x = np.linspace(0,len(train\_ls),len(train\_ls))
85. plt.figure(figsize=(10,8))
86. for i in range(0,len(drop\_probs)):
87. plt.plot(x,Train\_ls[i],label= 'drop\_prob=%.1f'%(drop\_probs[i]),linewidth=1.5)
88. plt.xlabel('epoch')
89. plt.ylabel('loss')
90. *# plt.legend()*
91. plt.legend(loc=2, bbox\_to\_anchor=(1.05,1.0),borderaxespad = 0.)
92. plt.title('train loss with dropout')
93. plt.show()

**4.5.1在多分类任务实验使用torch.nn实现**

1. import torch
2. import numpy as np
3. import torch.nn as nn
4. from torch import nn
5. from torchvision.datasets import MNIST
6. import torchvision.transforms  as transforms
7. import matplotlib.pyplot as plt
8. from sklearn.model\_selection import train\_test\_split
9. from torch.utils.data import DataLoader
10. from torch import nn
11. *#导入进行多回归任务所需要的包*
12. mnist\_train = MNIST(
13. root='./datasets/MNIST',
14. train = True,
15. download =True,
16. transform=transforms.ToTensor())
17. *#导入训练集所需要的测试数据*
18. mnist\_test = MNIST(
19. root='./datasets/MNIST',
20. train = False,
21. download =True,
22. transform=transforms.ToTensor())
23. *#导入测试集所需要的测试数据*
24. batch\_size = 32
25. *# 把训练数据放入 DataLoader*
26. train\_iter = DataLoader(
27. dataset = mnist\_train,
28. batch\_size = batch\_size,
29. shuffle = True,
30. )
31. *# 把测试数据放入 DataLoader*
32. test\_iter = DataLoader(
33. dataset = mnist\_test,
34. batch\_size = batch\_size,
35. shuffle = True,
36. )
37. class MM1(nn.Module):
38. def \_\_init\_\_(self,num\_inputs, num\_outputs, num\_hiddens1, num\_hiddens2,drop\_prob1,drop\_prob2):
39. super(MM1,self).\_\_init\_\_()
40. self.linear1 = nn.Linear(num\_inputs,num\_hiddens1)
41. self.relu = nn.ReLU()
42. self.drop1 = nn.Dropout(drop\_prob1)
43. self.linear2 = nn.Linear(num\_hiddens1,num\_hiddens2)
44. self.drop2 = nn.Dropout(drop\_prob2)
45. self.linear3 = nn.Linear(num\_hiddens2,num\_outputs)
46. self.flatten  = nn.Flatten()
48. def forward(self,x):
49. x = self.flatten(x)
50. x = self.linear1(x)
51. x = self.relu(x)
52. x = self.drop1(x)
53. x = self.linear2(x)
54. x = self.relu(x)
55. x = self.drop2(x)
56. x = self.linear3(x)
57. y = self.relu(x)
58. return y
60. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
61. train\_ls, test\_ls = [], []
62. for epoch in range(num\_epochs):
63. ls, count = 0, 0
64. for X,y in train\_iter:
65. l=loss(net(X),y)
66. optimizer.zero\_grad()
67. l.backward()
68. optimizer.step()
69. ls += l.item()
70. count += y.shape[0]
71. train\_ls.append(ls)
72. ls, count = 0, 0
73. for X,y in test\_iter:
74. l=loss(net(X),y)
75. ls += l.item()
76. count += y.shape[0]
77. test\_ls.append(ls)
78. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
79. return train\_ls,test\_ls
80. num\_inputs,num\_hiddens1,num\_hiddens2,num\_outputs =784, 256,256,10
81. num\_epochs=20
82. lr = 0.001
83. drop\_probs = np.arange(0,0.5,0.1)
84. Train\_ls, Test\_ls = [], []
85. for drop\_prob in drop\_probs:
86. net = MM1(num\_inputs, num\_outputs, num\_hiddens1, num\_hiddens2, drop\_prob,drop\_prob)
87. for param in net.parameters():
88. nn.init.normal\_(param,mean=0, std= 0.01)
89. loss = nn.CrossEntropyLoss()
90. optimizer = torch.optim.SGD(net.parameters(),lr)
91. train\_ls, test\_ls = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,net.parameters,lr,optimizer)
92. Train\_ls.append(train\_ls)
93. Test\_ls.append(test\_ls)
95. x = np.linspace(0,len(train\_ls),len(train\_ls))
96. plt.figure(figsize=(10,8))
97. for i in range(0,len(drop\_probs)):
98. plt.plot(x,Train\_ls[i],label= 'drop\_prob=%.1f'%(drop\_probs[i]),linewidth=1.5)
99. plt.xlabel('epoch')
100. plt.ylabel('loss')
101. plt.legend(loc=2, bbox\_to\_anchor=(1.05,1.0),borderaxespad = 0.)
102. plt.title('train loss with dropout')
103. plt.show()
104. input = torch.randn(2, 5, 5)
105. m = nn.Sequential(
106. nn.Flatten()
107. )
108. output = m(input)
109. output.size()

#### 4.6 在多分类任务实验中分别手动实现和用torch.nn实现L2正则化

**4.6.1在多分类任务实验中手动实现**

1. import torch
2. import numpy as np
3. from torch import nn
4. from torchvision.datasets import MNIST
5. import torchvision.transforms  as transforms
6. import matplotlib.pyplot as plt
7. from sklearn.model\_selection import train\_test\_split
8. from torch.utils.data import DataLoader
9. from torch import nn
10. #导入进行多回归任务所需要的包
11. mnist\_train = MNIST(
12. root='./datasets/MNIST',
13. train = True,
14. download =True,
15. transform=transforms.ToTensor())
16. *#导入训练集所需要的测试数据*
17. mnist\_test = MNIST(
18. root='./datasets/MNIST',
19. train = False,
20. download =True,
21. transform=transforms.ToTensor())
22. *#导入测试集所需要的测试数据*
23. *#读取数据*
24. batch\_size =128
25. *# 把训练数据放入 DataLoader*
26. train\_iter = DataLoader(
27. dataset = mnist\_train,
28. batch\_size = batch\_size,
29. shuffle = True,
30. )
31. *# 把测试数据放入 DataLoader*
32. test\_iter = DataLoader(
33. dataset = mnist\_test,
34. batch\_size = batch\_size,
35. shuffle = True,
36. )
37. #初始化参数
38. num\_inputs,num\_hiddens,num\_outputs = 784,256,10
39. def init\_param():
40. A1 = torch.tensor(np.random.normal(0, 0.01, (num\_hiddens,num\_inputs)), dtype=torch.float32)
41. B1 = torch.zeros(1, dtype=torch.float32)
42. A2 = torch.tensor(np.random.normal(0, 0.01, (num\_outputs,num\_hiddens)), dtype=torch.float32)
43. B2 = torch.zeros(1, dtype=torch.float32)
44. params =[A1,B1,A2,B2]
45. for param in params:
46. param.requires\_grad\_(requires\_grad=True)
47. return A1,B1,A2,B2
48. def relu(x):
49. x = torch.max(input=x,other=torch.tensor(0.0))
50. return x
51. #定义模型
52. def net(X):
53. X = X.view((-1,num\_inputs))
54. H = relu(torch.matmul(X,A1.t())+B1)
55. return torch.matmul(H,A2.t())+B2
56. #定义交叉熵损失函数
57. loss = torch.nn.CrossEntropyLoss()
58. #定义随机梯度下降法
59. def SGD(paras,lr):
60. for param in params:
61. param.data -= lr \* param.grad
63. #测试集loss
64. def evaluate\_loss(data\_iter,net):
65. acc\_sum,loss\_sum,n = 0.0,0.0,0
66. for X,y in data\_iter:
67. y\_hat = net(X)
68. acc\_sum += (y\_hat.argmax(dim=1)==y).sum().item()
69. l = loss(y\_hat,y) # l是有关小批量X和y的损失
70. loss\_sum += l.sum().item()\*y.shape[0]
71. n+=y.shape[0]
72. return acc\_sum/n,loss\_sum/n
73. def l2\_penalty(w):
74. return (w\*\*2).sum()/2
75. *#定义L2范数惩罚项*
76. *#定义模型训练函数*
77. def train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr=None,optimizer=None,sxlambda=0):
78. train\_ls, test\_ls = [], []
79. for epoch in range(num\_epochs):
80. ls, count = 0, 0
81. for X,y in train\_iter :
82. X = X.reshape(-1,num\_inputs)
83. l=loss(net(X),y)+ sxlambda\*l2\_penalty(A1) + sxlambda\*l2\_penalty(A2)#训练集中加入惩罚项
84. optimizer.zero\_grad()
85. l.backward()
86. optimizer.step()
87. ls += l.item()
88. count += y.shape[0]
89. train\_ls.append(ls)
90. ls, count = 0, 0
91. for X,y in test\_iter:
92. X = X.reshape(-1,num\_inputs)
93. l=loss(net(X),y) +  sxlambda\*l2\_penalty(A1) + sxlambda\*l2\_penalty(A2)#测试集中加入惩罚项
94. ls += l.item()
95. count += y.shape[0]
96. test\_ls.append(ls)
97. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
98. return train\_ls,test\_ls
99. lr = 0.01
100. num\_epochs = 50
101. Lamda = [0,0.1,0.2,0.3]#定义所有的惩罚因子数目=
102. Train\_ls, Test\_ls = [], []
103. for lamda in Lamda:
104. print("current lambda is %f"%lamda)
105. A1,B1,A2,B2 = init\_param()
106. loss = torch.nn.CrossEntropyLoss()
107. optimizer = torch.optim.SGD([A1,B1,A2,B2],lr = 0.001)
108. train\_ls, test\_ls = train(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,lr,optimizer,lamda)
109. Train\_ls.append(train\_ls)#训练集所有的惩罚因子
110. Test\_ls.append(test\_ls)#测试集所有的惩罚因子
111. x = np.linspace(0,len(Train\_ls[1]),len(Train\_ls[1]))
112. plt.figure(figsize=(10,8))
113. for i in range(0,len(Lamda)):
114. plt.plot(x,Train\_ls[i],label= f'L2\_Regularization:{Lamda [i]}',linewidth=1.5)
115. plt.xlabel('different epoch')
116. plt.ylabel('loss')
117. plt.legend(loc=2, bbox\_to\_anchor=(1.1,1.0),borderaxespad = 0.)
118. plt.title('train loss')
119. plt.show()

**4.6.2在多分类任务实验中使用torch.nn实现**

1. import torch
2. import numpy as np
3. from torch import nn
4. from torchvision.datasets import MNIST
5. import torchvision.transforms  as transforms
6. import matplotlib.pyplot as plt
7. from sklearn.model\_selection import train\_test\_split
8. from torch.utils.data import DataLoader
9. from torch import nn
10. *#导入进行多回归任务所需要的包*
11. mnist\_train = MNIST(
12. root='./datasets/MNIST',
13. train = True,
14. download =True,
15. transform=transforms.ToTensor())
16. *#导入训练集所需要的测试数据*
17. mnist\_test = MNIST(
18. root='./datasets/MNIST',
19. train = False,
20. download =True,
21. transform=transforms.ToTensor())
22. *#导入测试集所需要的测试数据*
23. *#读取数据*
24. batch\_size =64
25. *# 把训练数据放入 DataLoader*
26. train\_iter = DataLoader(
27. dataset = mnist\_train,
28. batch\_size = batch\_size,
29. shuffle = True,
30. )
31. *# 把测试数据放入 DataLoader*
32. test\_iter = DataLoader(
33. dataset = mnist\_test,
34. batch\_size = batch\_size,
35. shuffle = True,
36. )
37. num\_input,num\_hidden ,num\_output = 784,256,10
38. def init\_w\_b():
39. A1 = torch.tensor(np.random.normal(0, 0.01, (num\_inputs, num\_hiddens)), dtype=torch.float)
40. B1 = torch.zeros(num\_hiddens, dtype = torch.float)
41. A2 = torch.tensor(np.random.normal(0, 0.01, (num\_hiddens, num\_outputs)), dtype=torch.float)
42. B2 = torch.zeros(num\_outputs,dtype=torch.float)
43. params = [A1,B1,A2,B2]
44. for param in params:
45. param.requires\_grad\_(requires\_grad=True)
46. return A1,B1,A2,B2#定义惩罚权重因子
47. class MM1(nn.Module):
48. def \_\_init\_\_(self,num\_inputs, num\_outputs, num\_hiddens):
49. super(MM1,self).\_\_init\_\_()
50. self.linear1 = nn.Linear(num\_inputs,num\_hiddens)
51. self.relu = nn.ReLU()
52. self.linear2 = nn.Linear(num\_hiddens,num\_outputs)
53. self.flatten  = nn.Flatten()
55. def forward(self,x):
56. x = self.flatten(x)
57. x = self.linear1(x)
58. x = self.relu(x)
59. x = self.linear2(x)
60. y = self.relu(x)
61. return y#定义神经网络层数等相关内容
63. def train\_torch(lamda):#定义训练模型
64. num\_epochs = 50
65. train\_ls, test\_ls = [], []
66. for epoch in range(num\_epochs):
67. ls, count = 0, 0
68. for X,y in train\_iter:
69. l=loss(net(X),y)
70. optimizer\_w.zero\_grad()
71. optimizer\_b.zero\_grad()
72. l.backward()
73. optimizer\_w.step()
74. optimizer\_b.step()
75. ls += l.item()
76. count += y.shape[0]
77. train\_ls.append(ls)
78. ls, count = 0, 0
79. for X,y in test\_iter:
80. l=loss(net(X),y)
81. ls += l.item()
82. count += y.shape[0]
83. test\_ls.append(ls)
85. print('epoch: %d, train loss: %f, test loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
86. return train\_ls,test\_ls
87. Lamda = [0,1,2,3,4]
88. torch\_Train\_ls, torch\_Test\_ls = [], []
89. for lamda in Lamda:
90. A1,B1,A2,B2 = init\_w\_b()
91. loss = nn.CrossEntropyLoss()
92. net = MM1(num\_inputs, num\_outputs, num\_hiddens)
93. optimizer\_w = torch.optim.SGD([A1,A2],lr = 0.001,weight\_decay=lamda)
94. optimizer\_b = torch.optim.SGD([B1,B2],lr = 0.001)
95. train\_ls, test\_ls = train\_torch(lamda)
96. torch\_Train\_ls.append(train\_ls)
97. torch\_Test\_ls.append(test\_ls)
99. x = np.linspace(0,len(torch\_Train\_ls[1]),len(torch\_Train\_ls[1]))
100. plt.figure(figsize=(10,8))
101. for i in range(0,len(Lamda)):
102. plt.plot(x,torch\_Train\_ls[i],label= f'L2\_Regularization:{Lamda [i]}',linewidth=1.5)
103. plt.xlabel('epoch')
104. plt.ylabel('loss')
105. plt.legend(loc=2, bbox\_to\_anchor=(1.1,1.0),borderaxespad = 0.)
106. plt.title('train loss')
107. plt.show()

#### 4.7 对回归、二分类、多分类任务分别选择上述实验中效果最好的模型，采用10折交叉验证评估实验结果

**4.7.1对回归任务实现**

1. import torch
2. import numpy as np
3. import torch.nn as nn
4. import random
5. from IPython import display
6. from matplotlib import pyplot as plt
7. import torchvision
8. import torchvision.transforms as transforms
9. from pandas import \*
10. %matplotlib inline
11. *#导入必要的数据包等相关内容*
12. A = torch.rand(10000, 500)
13. B = (0.0028 \* A).sum(dim=1) + 0.028
14. B += torch.tensor(np.random.normal(0,0.01,size=B.size()), dtype=torch.float)
15. *#定义数据的读取*
16. def get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size):
17. train\_dataset = torch.utils.data.TensorDataset(X\_train,y\_train)
18. test\_dataset = torch.utils.data.TensorDataset(X\_valid,y\_valid)
19. train\_iter = torch.utils.data.DataLoader(train\_dataset,batch\_size=batch\_size,shuffle=True)
20. test\_iter = torch.utils.data.DataLoader(test\_dataset,batch\_size=batch\_size,shuffle=False)
21. return train\_iter, test\_iter
22. class MM(nn.Module):*#定义模型训练函数*
23. def \_\_init\_\_(self,num\_input,num\_hidden,num\_output):
24. super(MM,self).\_\_init\_\_()
25. self.linear1 = nn.Linear(num\_input,num\_hidden)
26. self.relu = nn.ReLU()
27. self.linear2 = nn.Linear(num\_hidden,num\_output)*#定义几个神经网络层的关系*
28. def forward(self,x):
29. x = self.linear1(x)
30. x = self.relu(x)
31. x = self.linear2(x)
32. y = self.relu(x)*#定义传递关系*
33. return y
35. *#模型训练*
36. def train(train\_iter,test\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss):
37. optimizer = torch.optim.SGD(net.parameters(),lr=0.001)
38. train\_ls, test\_ls = [], []
39. for epoch in range(num\_epochs):
40. ls, count = 0, 0
41. if if\_reshape ==False:
42. for X,y in train\_iter:
43. l=loss(net(X),y.view(-1,1))
44. optimizer.zero\_grad()
45. l.backward()
46. optimizer.step()
47. ls += l.item()
48. count += y.shape[0]
49. train\_ls.append(ls/count)
50. ls, count = 0, 0
51. for X,y in test\_iter:
52. l=loss(net(X),y.view(-1,1))
53. ls += l.item()
54. count += y.shape[0]
55. else:
56. for X,y in train\_iter:
57. X = X.reshape(-1,num\_inputs)
58. l=loss(net(X),y).sum()
59. optimizer.zero\_grad()
60. l.backward()
61. optimizer.step()
62. ls += l.item()
63. count += y.shape[0]
64. train\_ls.append(ls/count)
65. ls, count = 0, 0
66. for X,y in test\_iter:
67. X = X.reshape(-1,num\_inputs)
68. l=loss(net(X),y).sum()
69. ls += l.item()
70. count += y.shape[0]
71. test\_ls.append(ls/count)
72. print('epoch: %d, train loss: %f, valid loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
73. return train\_ls,test\_ls
74. *#获取k折交叉验证某一折的训练集和验证集*
75. def get\_kfold\_data(k, i, X, y):
76. fold\_size = X.shape[0]//k
77. val\_start = i \* fold\_size
78. if i  != k - 1:
79. val\_end = (i + 1) \* fold\_size
80. X\_valid, y\_valid = X[val\_start:val\_end],y[val\_start:val\_end]
81. X\_train = torch.cat((X[0:val\_start],X[val\_end:]),dim=0)
82. y\_train = torch.cat((y[0:val\_start],y[val\_end:]),dim=0)
83. else:
84. X\_valid,y\_valid = X[val\_start:], y[val\_start:]
85. X\_train = X[0:val\_start]
86. y\_train = y[0:val\_start]
88. return X\_train, y\_train, X\_valid, y\_valid
89. *#循环K次，取平均值，内容来源于PPT*
90. def k\_fold(k, X\_train, y\_train,if\_reshape,num\_epochs,num\_inputs,net,loss):
91. my\_k\_train\_ls, my\_k\_valid\_ls = [], []
92. train\_loss\_sum, valid\_loss\_sum = 0, 0
93. for i in range(k):
94. print('第', i+1, '折验证结果')
95. X\_train, y\_train, X\_valid, y\_valid = get\_kfold\_data(k, i, X\_train, y\_train)
96. train\_iter, valid\_iter = get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size=100)
97. train\_loss, val\_loss = train(train\_iter,valid\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss)
99. my\_k\_train\_ls.append(train\_loss)
100. my\_k\_valid\_ls.append(val\_loss)
101. train\_loss\_sum += train\_loss[-1]
102. valid\_loss\_sum += val\_loss[-1]
104. print("最终平均k折交叉验证结果")
106. print(f'average train loss: {train\_loss\_sum/k}')
107. print(f'average valid loss: {valid\_loss\_sum/k}')
109. return my\_k\_train\_ls, my\_k\_valid\_ls
110. k=10
111. mynum\_epochs= 20
112. num\_inputs,num\_outputs,num\_hiddens = 500,1,256
113. mynet = MM(num\_inputs, num\_outputs, num\_hiddens)
114. MSEloss= nn.MSELoss()
115. *# record\_k\_train\_ls, record\_k\_valid\_ls = k\_fold(k, X\_train, y\_train,if\_reshape=False,num\_epochs=20,num\_inputs,net ,loss)*
116. my\_k\_train\_ls, my\_k\_valid\_ls = k\_fold(k, A, B,if\_reshape=False,num\_epochs=mynum\_epochs,num\_inputs = 500,net =mynet,loss= MSEloss)
117. *# 绘图*
118. train\_loss, valid\_loss = [], []
119. for i in range(len(my\_k\_train\_ls)):
120. train\_loss.append(my\_k\_train\_ls[i][-1])
121. valid\_loss.append(my\_k\_valid\_ls[i][-1])
123. x = np.linspace(0,len(my\_k\_train\_ls),len(my\_k\_train\_ls))
124. plt.plot(x,train\_loss,'o-',label='train\_loss',linewidth=1.5)
125. plt.plot(x,valid\_loss,'o-',label='valid\_loss',linewidth=1.5)
126. plt.xlabel('K value')
127. plt.ylabel('loss')
128. plt.legend()
129. plt.show()
130. *# 绘制表格*
131. from pylab import mpl
132. mpl.rcParams['font.sans-serif'] = ['SimHei'] *# 指定默认字体*
133. mpl.rcParams['axes.unicode\_minus'] = False *# 解决保存图像是负号'-'显示为方块的问题*
134. randn = np.random.randn
135. idx = []
136. for i in range(1,21):
137. idx.append(f'epoch {i}')
138. data\_train, data\_valid = np.zeros((10,20)),np.zeros((10,20))
139. for i in range(10):
140. for j in range(20):
141. data\_train[i,j], data\_valid[i,j] = my\_k\_train\_ls[i][j], my\_k\_valid\_ls[i][j]
143. df = DataFrame(data\_train.T, index=idx, columns=['第1折', '第2折', '第3折', '第4折', '第5折',
144. '第6折', '第7折', '第8折', '第9折', '第10折'])
145. vals = np.around(df.values,7)
146. fig = plt.figure(figsize=(8,3))
147. ax = fig.add\_subplot(111, frameon=False, xticks=[], yticks=[])
148. the\_table=plt.table(cellText=vals, rowLabels=df.index, colLabels=df.columns,
149. colWidths = [0.1]\*vals.shape[1], loc='center',cellLoc='center')
150. the\_table.set\_fontsize(20)
151. the\_table.scale(2.5,2.58)

**4.7.1对二分类任务实现**

1. import torch
2. import numpy as np
3. import torch.nn as nn
4. import random
5. from IPython import display
6. from matplotlib import pyplot as plt
7. import torchvision
8. import torchvision.transforms as transforms
9. from pandas import \*
10. %matplotlib inline
11. *#导入必要的数据包等相关内容*
12. num\_inputs = 200
13. num\_examples = 10000
14. a1 = torch.normal(0.1,1,(10000, num\_inputs))
15. b1 = torch.ones(10000,1)
16. a2 = torch.normal(-0.1,1,(10000, num\_inputs))
17. b2 = torch.zeros(10000,1)
18. class\_2\_features = torch.cat((a1,a2),dim=0)
19. class\_2\_labels = torch.cat((b1,b2))
20. index = [i for i in range(len(class\_2\_labels))]
21. np.random.shuffle(index)
22. X\_train = class\_2\_features[index,:]
23. y\_train = class\_2\_labels[index]
24. #定义数据的读取
25. def get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size):
26. train\_dataset = torch.utils.data.TensorDataset(X\_train,y\_train)
27. test\_dataset = torch.utils.data.TensorDataset(X\_valid,y\_valid)
28. train\_iter = torch.utils.data.DataLoader(train\_dataset,batch\_size=batch\_size,shuffle=True)
29. test\_iter = torch.utils.data.DataLoader(test\_dataset,batch\_size=batch\_size,shuffle=False)
30. return train\_iter, test\_iter
31. class MM(nn.Module):#定义模型训练函数
32. def \_\_init\_\_(self,n\_feature):
33. super(MM,self).\_\_init\_\_()
34. self.linear1 = nn.Linear(n\_feature,100)
35. self.relu = nn.ReLU()
36. self.linear2 = nn.Linear(100,1)
37. self.Sigmoid = nn.Sigmoid()
39. def forward(self,x):
40. x = self.linear1(x)
41. x = self.relu(x)
42. x = self.linear2(x)
43. y = self.Sigmoid(x)
44. return y
46. #模型训练
47. *#模型训练*
48. def train(train\_iter,test\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss):
49. optimizer = torch.optim.SGD(net.parameters(),lr=0.001)
50. train\_ls, test\_ls = [], []
51. for epoch in range(num\_epochs):
52. ls, count = 0, 0
53. if if\_reshape ==False:
54. for X,y in train\_iter:
55. l=loss(net(X),y.view(-1,1))
56. optimizer.zero\_grad()
57. l.backward()
58. optimizer.step()
59. ls += l.item()
60. count += y.shape[0]
61. train\_ls.append(ls/count)
62. ls, count = 0, 0
63. for X,y in test\_iter:
64. l=loss(net(X),y.view(-1,1))
65. ls += l.item()
66. count += y.shape[0]
67. else:
68. for X,y in train\_iter:
69. X = X.reshape(-1,num\_inputs)
70. l=loss(net(X),y).sum()
71. optimizer.zero\_grad()
72. l.backward()
73. optimizer.step()
74. ls += l.item()
75. count += y.shape[0]
76. train\_ls.append(ls/count)
77. ls, count = 0, 0
78. for X,y in test\_iter:
79. X = X.reshape(-1,num\_inputs)
80. l=loss(net(X),y).sum()
81. ls += l.item()
82. count += y.shape[0]
83. test\_ls.append(ls/count)
84. print('epoch: %d, train loss: %f, valid loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
85. return train\_ls,test\_ls
86. #获取k折交叉验证某一折的训练集和验证集
87. def get\_kfold\_data(k, i, X, y):
88. fold\_size = X.shape[0]//k
89. val\_start = i \* fold\_size
90. if i  != k - 1:
91. val\_end = (i + 1) \* fold\_size
92. X\_valid, y\_valid = X[val\_start:val\_end],y[val\_start:val\_end]
93. X\_train = torch.cat((X[0:val\_start],X[val\_end:]),dim=0)
94. y\_train = torch.cat((y[0:val\_start],y[val\_end:]),dim=0)
95. else:
96. X\_valid,y\_valid = X[val\_start:], y[val\_start:]
97. X\_train = X[0:val\_start]
98. y\_train = y[0:val\_start]
100. return X\_train, y\_train, X\_valid, y\_valid
101. *#循环K次，取平均值，内容来源于PPT*
102. def k\_fold(k, X\_train, y\_train,if\_reshape,num\_epochs,num\_inputs,net,loss):
103. my\_k\_train\_ls, my\_k\_valid\_ls = [], []
104. train\_loss\_sum, valid\_loss\_sum = 0, 0
105. for i in range(k):
106. print('第', i+1, '折验证结果')
107. X\_train, y\_train, X\_valid, y\_valid = get\_kfold\_data(k, i, X\_train, y\_train)
108. train\_iter, valid\_iter = get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size=100)
109. train\_loss, val\_loss = train(train\_iter,valid\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss)
111. my\_k\_train\_ls.append(train\_loss)
112. my\_k\_valid\_ls.append(val\_loss)
113. train\_loss\_sum += train\_loss[-1]
114. valid\_loss\_sum += val\_loss[-1]
116. print("最终平均k折交叉验证结果")
118. print(f'average train loss: {train\_loss\_sum/k}')
119. print(f'average valid loss: {valid\_loss\_sum/k}')
121. return my\_k\_train\_ls, my\_k\_valid\_ls
122. k=10
123. num\_epochs= 20
124. num\_inputs = 200
125. net = MM(num\_inputs)
126. loss= nn.MSELoss()
127. if\_reshape=False
128. net = MM(200)
129. k\_train\_ls, k\_valid\_ls = k\_fold(k,X\_train, y\_train,if\_reshape,num\_epochs,num\_inputs ,net,loss)
130. # 绘图
131. train\_loss, valid\_loss = [], []
132. for i in range(len(k\_train\_ls)):
133. train\_loss.append(k\_train\_ls[i][-1])
134. valid\_loss.append(k\_valid\_ls[i][-1])
136. x = np.linspace(0,len(k\_train\_ls),len(k\_train\_ls))
137. plt.plot(x,train\_loss,'o-',label='train\_loss',linewidth=1.5)
138. plt.plot(x,valid\_loss,'o-',label='valid\_loss',linewidth=1.5)
139. plt.xlabel('K value')
140. plt.ylabel('loss')
141. plt.legend()
142. plt.show()
143. # 绘制表格
144. from pylab import mpl
145. mpl.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
146. mpl.rcParams['axes.unicode\_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
147. randn = np.random.randn
148. idx = []
149. *#10折交叉验证，20轮*
150. for i in range(1,21):
151. idx.append(f'epoch {i}')
152. *#10折交叉验证，20轮*
153. data\_train, data\_valid = np.zeros((10,20)),np.zeros((10,20))
154. for i in range(10):
155. for j in range(20):
156. data\_train[i,j], data\_valid[i,j] = k\_train\_ls[i][j], k\_valid\_ls[i][j]
158. df = DataFrame(data\_train.T, index=idx, columns=['第1折', '第2折', '第3折', '第4折', '第5折','第6折', '第7折', '第8折', '第9折', '第10折'])
160. vals = np.around(df.values,7)
161. fig = plt.figure(figsize=(8,3))
162. ax = fig.add\_subplot(111, frameon=False, xticks=[], yticks=[])
163. the\_table=plt.table(cellText=vals, rowLabels=df.index, colLabels=df.columns,
164. colWidths = [0.1]\*vals.shape[1], loc='center',cellLoc='center')
165. the\_table.set\_fontsize(20)
166. the\_table.scale(2.5,2.58)

**4.7.3对多回归任务实现**

1. import torch
2. import numpy as np
3. import torch.nn as nn
4. import random
5. from IPython import display
6. from matplotlib import pyplot as plt
7. from torchvision.datasets import MNIST
8. import torchvision
9. from torch.utils.data import DataLoader
10. import torchvision.transforms as transforms
11. from pandas import \*
12. %matplotlib inline
13. *#导入必要的数据包等相关内容*
14. mnist\_train = MNIST(
15. root='./datasets/MNIST',
16. train = True,
17. download =True,
18. transform=transforms.ToTensor())
19. *#导入训练集所需要的测试数据*
20. mnist\_test = MNIST(
21. root='./datasets/MNIST',
22. train = False,
23. download =True,
24. transform=transforms.ToTensor())
25. *#导入测试集所需要的测试数据*
26. *#读取数据*
27. batch\_size =64
28. *# 把训练数据放入 DataLoader*
29. train\_iter = DataLoader(
30. dataset = mnist\_train,
31. batch\_size = batch\_size,
32. shuffle = True,
33. )
34. *# 把测试数据放入 DataLoader*
35. test\_iter = DataLoader(
36. dataset = mnist\_test,
37. batch\_size = batch\_size,
38. shuffle = True,
39. )
40. mnist\_features = torch.cat((train\_iter.dataset.data,test\_iter.dataset.data),dim=0)
41. mnist\_labels = torch.cat((train\_iter.dataset.train\_labels,test\_iter.dataset.test\_labels))
42. mnist\_features = mnist\_features.float()
43. mnist\_labels = mnist\_labels.long()
44. #定义数据的读取
45. def get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size):
46. train\_dataset = torch.utils.data.TensorDataset(X\_train,y\_train)
47. test\_dataset = torch.utils.data.TensorDataset(X\_valid,y\_valid)
48. train\_iter = torch.utils.data.DataLoader(train\_dataset,batch\_size=batch\_size,shuffle=True)
49. test\_iter = torch.utils.data.DataLoader(test\_dataset,batch\_size=batch\_size,shuffle=False)
50. return train\_iter, test\_iter
51. class MM(nn.Module):
52. def \_\_init\_\_(self,num\_inputs, num\_outputs, num\_hiddens):
53. super(MM,self).\_\_init\_\_()
54. self.linear1 = nn.Linear(num\_inputs,num\_hiddens)
55. self.relu = nn.ReLU()
56. self.linear2 = nn.Linear(num\_hiddens,num\_outputs)
58. def forward(self,x):
59. x = self.linear1(x)
60. x = self.relu(x)
61. x = self.linear2(x)
62. y = self.relu(x)
63. return y
65. #模型训练
66. *#模型训练*
67. def train(train\_iter,test\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss):
68. optimizer = torch.optim.SGD(net.parameters(),lr=0.001)
69. train\_ls, test\_ls = [], []
70. for epoch in range(num\_epochs):
71. ls, count = 0, 0
72. if if\_reshape ==False:
73. for X,y in train\_iter:
74. l=loss(net(X),y.view(-1,1))
75. optimizer.zero\_grad()
76. l.backward()
77. optimizer.step()
78. ls += l.item()
79. count += y.shape[0]
80. train\_ls.append(ls/count)
81. ls, count = 0, 0
82. for X,y in test\_iter:
83. l=loss(net(X),y.view(-1,1))
84. ls += l.item()
85. count += y.shape[0]
86. else:
87. for X,y in train\_iter:
88. X = X.reshape(-1,num\_inputs)
89. l=loss(net(X),y).sum()
90. optimizer.zero\_grad()
91. l.backward()
92. optimizer.step()
93. ls += l.item()
94. count += y.shape[0]
95. train\_ls.append(ls/count)
96. ls, count = 0, 0
97. for X,y in test\_iter:
98. X = X.reshape(-1,num\_inputs)
99. l=loss(net(X),y).sum()
100. ls += l.item()
101. count += y.shape[0]
102. test\_ls.append(ls/count)
103. print('epoch: %d, train loss: %f, valid loss: %f'%(epoch+1,train\_ls[-1],test\_ls[-1]))
104. return train\_ls,test\_ls
105. #获取k折交叉验证某一折的训练集和验证集
106. def get\_kfold\_data(k, i, X, y):
107. fold\_size = X.shape[0]//k
108. val\_start = i \* fold\_size
109. if i  != k - 1:
110. val\_end = (i + 1) \* fold\_size
111. X\_valid, y\_valid = X[val\_start:val\_end],y[val\_start:val\_end]
112. X\_train = torch.cat((X[0:val\_start],X[val\_end:]),dim=0)
113. y\_train = torch.cat((y[0:val\_start],y[val\_end:]),dim=0)
114. else:
115. X\_valid,y\_valid = X[val\_start:], y[val\_start:]
116. X\_train = X[0:val\_start]
117. y\_train = y[0:val\_start]
119. return X\_train, y\_train, X\_valid, y\_valid
120. *#循环K次，取平均值，内容来源于PPT*
121. def k\_fold(k, X\_train, y\_train,if\_reshape,num\_epochs,num\_inputs,net,loss):
122. my\_k\_train\_ls, my\_k\_valid\_ls = [], []
123. train\_loss\_sum, valid\_loss\_sum = 0, 0
124. for i in range(k):
125. print('第', i+1, '折验证结果')
126. X\_train, y\_train, X\_valid, y\_valid = get\_kfold\_data(k, i, X\_train, y\_train)
127. train\_iter, valid\_iter = get\_data\_iter(X\_train, y\_train, X\_valid, y\_valid,batch\_size=100)
128. train\_loss, val\_loss = train(train\_iter,valid\_iter,if\_reshape,num\_epochs,num\_inputs,net,loss)
130. my\_k\_train\_ls.append(train\_loss)
131. my\_k\_valid\_ls.append(val\_loss)
132. train\_loss\_sum += train\_loss[-1]
133. valid\_loss\_sum += val\_loss[-1]
135. print("最终平均k折交叉验证结果")
137. print(f'average train loss: {train\_loss\_sum/k}')
138. print(f'average valid loss: {valid\_loss\_sum/k}')
140. return my\_k\_train\_ls, my\_k\_valid\_ls
141. k=10
142. mynum\_epochs= 20
143. mynet=MM(784, 10, 100)
144. my\_k\_train\_ls, my\_k\_valid\_ls = k\_fold(k, mnist\_features, mnist\_labels,if\_reshape=True, num\_epochs=mynum\_epochs, num\_inputs = 784, net =mynet ,loss=nn.CrossEntropyLoss())
145. # 绘图
146. train\_loss, valid\_loss = [], []
147. for i in range(len(my\_k\_train\_ls)):
148. train\_loss.append(my\_k\_train\_ls[i][-1])
149. valid\_loss.append(my\_k\_valid\_ls[i][-1])
151. x = np.linspace(0,len(my\_k\_train\_ls),len(my\_k\_train\_ls))
152. plt.plot(x,train\_loss,'o-',label='train\_loss',linewidth=1.5)
153. plt.plot(x,valid\_loss,'o-',label='valid\_loss',linewidth=1.5)
154. plt.xlabel('K value')
155. plt.ylabel('loss')
156. plt.legend()
157. plt.show()
158. # 绘制表格
159. from pylab import mpl
160. mpl.rcParams['font.sans-serif'] = ['SimHei'] # 指定默认字体
161. mpl.rcParams['axes.unicode\_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
162. randn = np.random.randn
163. idx = []
164. for i in range(1,21):
165. idx.append(f'epoch {i}')
166. data\_train, data\_valid = np.zeros((10,20)),np.zeros((10,20))
167. for i in range(10):
168. for j in range(20):
169. data\_train[i,j], data\_valid[i,j] = my\_k\_train\_ls[i][j], my\_k\_valid\_ls[i][j]

172. df = DataFrame(data\_train.T, index=idx, columns=['第1折', '第2折', '第3折', '第4折', '第5折',
173. '第6折', '第7折', '第8折', '第9折', '第10折'])
174. vals = np.around(df.values,7)
175. fig = plt.figure(figsize=(8,3))
176. ax = fig.add\_subplot(111, frameon=False, xticks=[], yticks=[])
177. the\_table=plt.table(cellText=vals, rowLabels=df.index, colLabels=df.columns,
178. colWidths = [0.1]\*vals.shape[1], loc='center',cellLoc='center')
179. the\_table.set\_fontsize(20)
180. the\_table.scale(2.5,2.58)

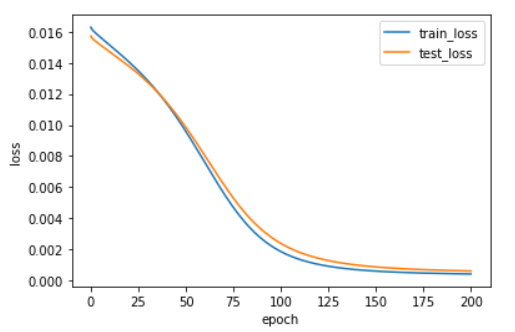
## 五、实验结果

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。

#### 5.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

**5.1.1 实验4.1.1运行结果**

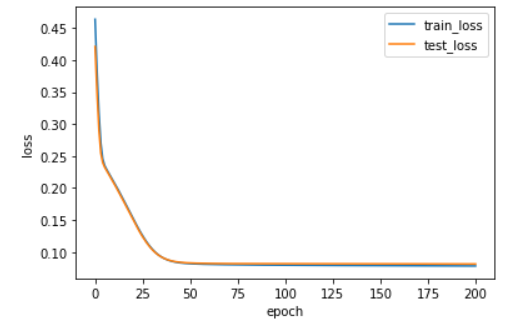
1. epoch 1, train\_loss 0.016285,test\_loss 0.015722
2. epoch 2, train\_loss 0.016093,test\_loss 0.015543
3. epoch 3, train\_loss 0.015970,test\_loss 0.015434
4. epoch 4, train\_loss 0.015859,test\_loss 0.015337
5. epoch 5, train\_loss 0.015751,test\_loss 0.015244
6. epoch 6, train\_loss 0.015645,test\_loss 0.015152
7. epoch 7, train\_loss 0.015540,test\_loss 0.015061
8. epoch 8, train\_loss 0.015436,test\_loss 0.014971
9. epoch 9, train\_loss 0.015332,test\_loss 0.014881
10. epoch 10, train\_loss 0.015228,test\_loss 0.014792
11. *######迭代将近200轮以后######*
12. epoch 190, train\_loss 0.000429,test\_loss 0.000631
13. epoch 191, train\_loss 0.000427,test\_loss 0.000628
14. epoch 192, train\_loss 0.000425,test\_loss 0.000624
15. epoch 193, train\_loss 0.000423,test\_loss 0.000621
16. epoch 194, train\_loss 0.000420,test\_loss 0.000619
17. epoch 195, train\_loss 0.000418,test\_loss 0.000616
18. epoch 196, train\_loss 0.000417,test\_loss 0.000613
19. epoch 197, train\_loss 0.000415,test\_loss 0.000610
20. epoch 198, train\_loss 0.000413,test\_loss 0.000608
21. epoch 199, train\_loss 0.000411,test\_loss 0.000605
22. epoch 200, train\_loss 0.000409,test\_loss 0.000603



分析：在将近200轮左右的迭代以后，训练集和测试集的结果如上所示。根据关系可知，测试损失略大于训练损失，但是由于二者相近，可以近似认为训练结果较为标准，绘制两条曲线，二者损失图像较为接近，可以认为结果正确。

**5.1.2 实验4.1.2运行结果**

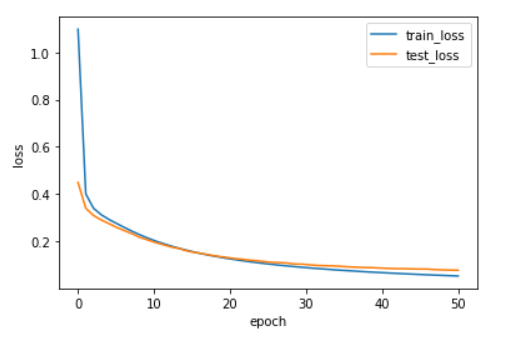
1. epoch 1, train\_loss 0.463795,test\_loss 0.421186,train\_acc 0.082
2. epoch 2, train\_loss 0.384372,test\_loss 0.349733,train\_acc 0.096
3. epoch 3, train\_loss 0.318894,test\_loss 0.291195,train\_acc 0.079
4. epoch 4, train\_loss 0.270582,test\_loss 0.254767,train\_acc 0.064
5. epoch 5, train\_loss 0.245700,test\_loss 0.239614,train\_acc 0.107
6. epoch 6, train\_loss 0.235451,test\_loss 0.232605,train\_acc 0.111
7. epoch 7, train\_loss 0.229530,test\_loss 0.227288,train\_acc 0.100
8. epoch 8, train\_loss 0.224344,test\_loss 0.222185,train\_acc 0.071
9. epoch 9, train\_loss 0.219218,test\_loss 0.217060,train\_acc 0.093
10. epoch 10, train\_loss 0.214042,test\_loss 0.211861,train\_acc 0.096
11. *#####经过多轮迭代以后#####*
12. epoch 191, train\_loss 0.078350,test\_loss 0.081405,train\_acc 0.100
13. epoch 192, train\_loss 0.078337,test\_loss 0.081399,train\_acc 0.100
14. epoch 193, train\_loss 0.078326,test\_loss 0.081401,train\_acc 0.082
15. epoch 194, train\_loss 0.078316,test\_loss 0.081392,train\_acc 0.100
16. epoch 195, train\_loss 0.078304,test\_loss 0.081381,train\_acc 0.086
17. epoch 196, train\_loss 0.078296,test\_loss 0.081385,train\_acc 0.086
18. epoch 197, train\_loss 0.078282,test\_loss 0.081384,train\_acc 0.068
19. epoch 198, train\_loss 0.078274,test\_loss 0.081379,train\_acc 0.082
20. epoch 199, train\_loss 0.078259,test\_loss 0.081381,train\_acc 0.082
21. epoch 200, train\_loss 0.078250,test\_loss 0.081365,train\_acc 0.086



分析：在将近200轮左右的迭代以后，训练集和测试集的结果如上所示。根据关系可知，可以近似认为训练结果较为标准，绘制两条曲线，二者损失图像较为接近。值得注意的是，设置不同的初始数据对训练又不一样的结果。若设置矩阵的a1 = torch.normal(2,1,(10000, num\_inputs))，a2 = torch.normal(2,1,(10000, num\_inputs))，则会出现训练损失垂直下降的现象。

**5.1.3 实验4.1.3运行结果**

1. epoch 1, train\_loss 1.097721,test\_loss 0.449021,train\_acc 0.746783,test\_acc 0.883600
2. epoch 2, train\_loss 0.400539,test\_loss 0.339998,train\_acc 0.888117,test\_acc 0.903600
3. epoch 3, train\_loss 0.339836,test\_loss 0.309307,train\_acc 0.902317,test\_acc 0.912800
4. epoch 4, train\_loss 0.312171,test\_loss 0.290130,train\_acc 0.910450,test\_acc 0.917400
5. epoch 5, train\_loss 0.292318,test\_loss 0.274840,train\_acc 0.916417,test\_acc 0.920800
6. epoch 6, train\_loss 0.275069,test\_loss 0.258577,train\_acc 0.921467,test\_acc 0.926500
7. epoch 7, train\_loss 0.258231,test\_loss 0.244900,train\_acc 0.926183,test\_acc 0.930400
8. epoch 8, train\_loss 0.242022,test\_loss 0.231718,train\_acc 0.931300,test\_acc 0.933900
9. epoch 9, train\_loss 0.227204,test\_loss 0.216078,train\_acc 0.935767,test\_acc 0.938300
10. epoch 10, train\_loss 0.213537,test\_loss 0.205624,train\_acc 0.939467,test\_acc 0.941700
11. *#####经过多轮迭代以后#####*
12. epoch 41, train\_loss 0.065915,test\_loss 0.085121,train\_acc 0.982383,test\_acc 0.973700
13. epoch 42, train\_loss 0.064393,test\_loss 0.084237,train\_acc 0.982600,test\_acc 0.974500
14. epoch 43, train\_loss 0.062616,test\_loss 0.084095,train\_acc 0.983150,test\_acc 0.974400
15. epoch 44, train\_loss 0.061166,test\_loss 0.083032,train\_acc 0.983500,test\_acc 0.974600
16. epoch 45, train\_loss 0.059710,test\_loss 0.082438,train\_acc 0.984200,test\_acc 0.974500
17. epoch 46, train\_loss 0.058338,test\_loss 0.082352,train\_acc 0.984417,test\_acc 0.975700
18. epoch 47, train\_loss 0.057032,test\_loss 0.080239,train\_acc 0.984967,test\_acc 0.975200
19. epoch 48, train\_loss 0.055698,test\_loss 0.079012,train\_acc 0.985400,test\_acc 0.975800
20. epoch 49, train\_loss 0.054394,test\_loss 0.078121,train\_acc 0.985733,test\_acc 0.976100
21. epoch 50, train\_loss 0.053194,test\_loss 0.077571,train\_acc 0.986333,test\_acc 0.976500

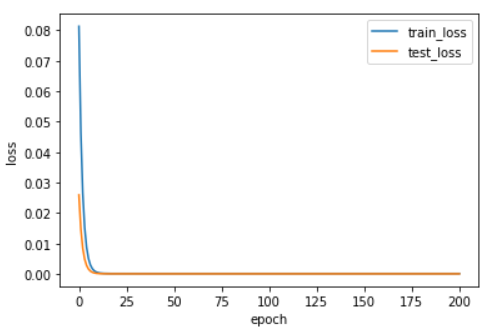


**分析：**在将近200轮左右的迭代以后，训练集和测试集的结果如上所示。根据关系可知，可以近似认为训练结果较为标准，绘制两条曲线，二者损失图像较为接近。可以近似认为训练模型具有一定的精度和通用性。

#### 5.2 利用torch.nn实现前馈神经网络解决回归等相关问题

**5.2.1 实验4.2.1运行结果**

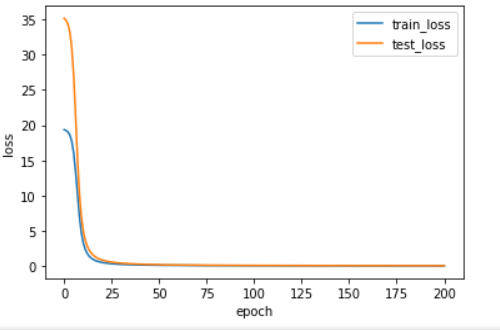
1. epoch 1,train\_loss 0.081307,test\_loss 0.025945
2. epoch 2,train\_loss 0.046471,test\_loss 0.014835
3. epoch 3,train\_loss 0.026587,test\_loss 0.008493
4. epoch 4,train\_loss 0.015237,test\_loss 0.004874
5. epoch 5,train\_loss 0.008758,test\_loss 0.002807
6. epoch 6,train\_loss 0.005060,test\_loss 0.001628
7. epoch 7,train\_loss 0.002950,test\_loss 0.000955
8. epoch 8,train\_loss 0.001745,test\_loss 0.000571
9. epoch 9,train\_loss 0.001057,test\_loss 0.000351
10. epoch 10,train\_loss 0.000664,test\_loss 0.000226
11. *######进行了接近200轮迭代以后######*
12. epoch 191,train\_loss 0.000142,test\_loss 0.000060
13. epoch 192,train\_loss 0.000142,test\_loss 0.000060
14. epoch 193,train\_loss 0.000142,test\_loss 0.000060
15. epoch 194,train\_loss 0.000142,test\_loss 0.000060
16. epoch 195,train\_loss 0.000142,test\_loss 0.000060
17. epoch 196,train\_loss 0.000142,test\_loss 0.000060
18. epoch 197,train\_loss 0.000142,test\_loss 0.000060
19. epoch 198,train\_loss 0.000142,test\_loss 0.000060
20. epoch 199,train\_loss 0.000142,test\_loss 0.000060
21. epoch 200,train\_loss 0.000142,test\_loss 0.000060



**分析：**根据结果分析可得，随着迭代次数的增加，函数的损失值不断减少，训练精度逐渐上升，模型的准确性随着迭代次数的增加不断提高。值得注意的是，使用torch.nn方法在结果上更容易收敛，从而使损失曲线提前进入平缓阶段。

**5.2.2 实验4.2.2运行结果**

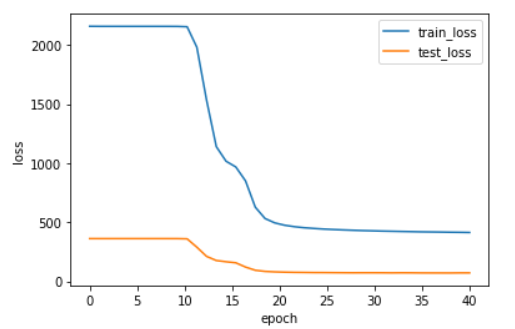
1. epoch 1, train\_loss 19.362644,test\_loss 35.176718, train\_acc 0.000000,test\_acc 0.769231
2. epoch 2, train\_loss 19.247452,test\_loss 34.899738, train\_acc 0.000000,test\_acc 0.769231
3. epoch 3, train\_loss 19.031980,test\_loss 34.344482, train\_acc 0.000000,test\_acc 0.769231
4. epoch 4, train\_loss 18.590755,test\_loss 33.203409, train\_acc 0.000000,test\_acc 0.769231
5. epoch 5, train\_loss 17.702024,test\_loss 30.964588, train\_acc 0.000000,test\_acc 0.769231
6. epoch 6, train\_loss 16.048558,test\_loss 27.037336, train\_acc 0.000000,test\_acc 0.769231
7. epoch 7, train\_loss 13.414571,test\_loss 21.399958, train\_acc 0.429286,test\_acc 1.000000
8. epoch 8, train\_loss 10.118850,test\_loss 15.310209, train\_acc 1.000000,test\_acc 1.000000
9. epoch 9, train\_loss 7.030992,test\_loss 10.381610, train\_acc 1.000000,test\_acc 1.000000
10. epoch 10, train\_loss 4.762599,test\_loss 7.084534, train\_acc 1.000000,test\_acc 1.000000
11. *######经过200轮迭代后######*
12. epoch 191, train\_loss 0.015724,test\_loss 0.028402, train\_acc 1.000000,test\_acc 1.000000
13. epoch 192, train\_loss 0.015622,test\_loss 0.028213, train\_acc 1.000000,test\_acc 1.000000
14. epoch 193, train\_loss 0.015519,test\_loss 0.028033, train\_acc 1.000000,test\_acc 1.000000
15. epoch 194, train\_loss 0.015402,test\_loss 0.027855, train\_acc 1.000000,test\_acc 1.000000
16. epoch 195, train\_loss 0.015302,test\_loss 0.027678, train\_acc 1.000000,test\_acc 1.000000
17. epoch 196, train\_loss 0.015209,test\_loss 0.027497, train\_acc 1.000000,test\_acc 1.000000
18. epoch 197, train\_loss 0.015131,test\_loss 0.027324, train\_acc 1.000000,test\_acc 1.000000
19. epoch 198, train\_loss 0.015045,test\_loss 0.027153, train\_acc 1.000000,test\_acc 1.000000
20. epoch 199, train\_loss 0.014928,test\_loss 0.026981, train\_acc 1.000000,test\_acc 1.000000
21. epoch 200, train\_loss 0.014829,test\_loss 0.026816, train\_acc 1.000000,test\_acc 1.000000



**分析：**根据结果分析可得，随着迭代次数的增加，函数的损失值不断减少。值得注意的是，尽管训练精度已经到达了1，但是其训练的损失函数和测试的损失函数依然处于不断下降的状态。

**5.2.3 实验4.2.3运行结果**

1. epoch 1, train\_loss 2159.163836,test\_loss 361.294559, train\_acc 0.111933,test\_acc 0.113500
2. epoch 2, train\_loss 2158.635346,test\_loss 361.247352, train\_acc 0.112367,test\_acc 0.113500
3. epoch 3, train\_loss 2158.544258,test\_loss 361.249302, train\_acc 0.112367,test\_acc 0.113500
4. epoch 4, train\_loss 2158.533356,test\_loss 361.261121, train\_acc 0.112367,test\_acc 0.113500
5. epoch 5, train\_loss 2158.522419,test\_loss 361.271713, train\_acc 0.112367,test\_acc 0.113500
6. epoch 6, train\_loss 2158.493233,test\_loss 361.273228, train\_acc 0.112367,test\_acc 0.113500
7. epoch 7, train\_loss 2158.494904,test\_loss 361.245870, train\_acc 0.112367,test\_acc 0.113500
8. epoch 8, train\_loss 2158.441734,test\_loss 361.255812, train\_acc 0.112367,test\_acc 0.113500
9. epoch 9, train\_loss 2158.346801,test\_loss 361.194889, train\_acc 0.112367,test\_acc 0.113500
10. epoch 10, train\_loss 2158.004590,test\_loss 361.098798, train\_acc 0.112367,test\_acc 0.113500
11. *#####经过将近40轮的迭代后#####*
12. epoch 31, train\_loss 425.009542,test\_loss 71.909434, train\_acc 0.870933,test\_acc 0.869800
13. epoch 32, train\_loss 423.398281,test\_loss 71.134676, train\_acc 0.872783,test\_acc 0.872700
14. epoch 33, train\_loss 421.690776,test\_loss 71.764338, train\_acc 0.872617,test\_acc 0.872200
15. epoch 34, train\_loss 420.199684,test\_loss 71.539792, train\_acc 0.874317,test\_acc 0.871600
16. epoch 35, train\_loss 418.625546,test\_loss 70.634065, train\_acc 0.873333,test\_acc 0.874600
17. epoch 36, train\_loss 417.643464,test\_loss 70.448624, train\_acc 0.874000,test\_acc 0.872200
18. epoch 37, train\_loss 416.675161,test\_loss 70.317870, train\_acc 0.874100,test\_acc 0.873100
19. epoch 38, train\_loss 415.260405,test\_loss 70.174687, train\_acc 0.875100,test\_acc 0.872600
20. epoch 39, train\_loss 414.434988,test\_loss 70.965286, train\_acc 0.875183,test\_acc 0.871500
21. epoch 40, train\_loss 413.145515,test\_loss 70.945907, train\_acc 0.875950,test\_acc 0.875700

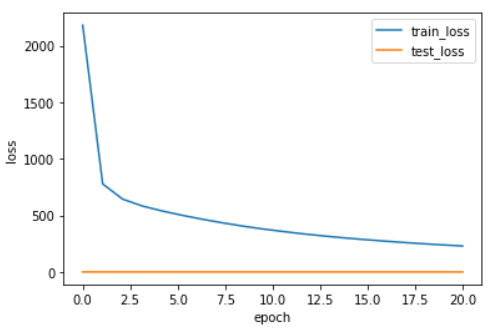


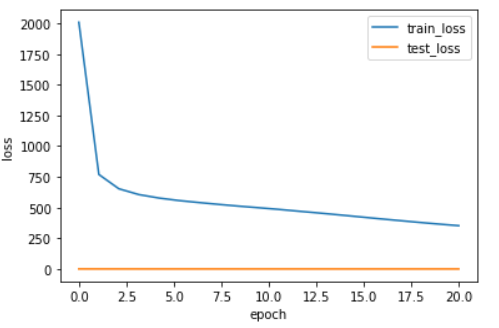
**分析：**根据结果分析可得，随着迭代次数的增加，函数的损失值不断减少，尽管训练精度已经有向1逼近的趋势，但是其训练的损失函数和测试的损失函数还是处于一个比较高的数值，但是其主要趋势依然处于不断下降的状态。由此可知还需要进行更多次的迭代才能进一步提高训练精度和测试精度。

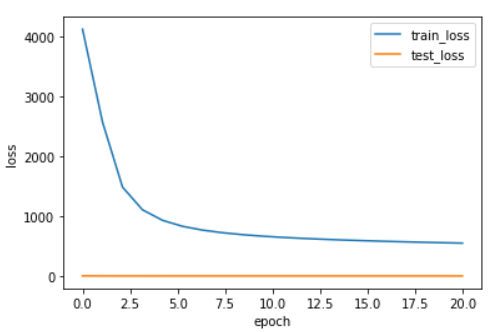
#### 5.3 在多分类实验的基础上使用至少三种不同的激活函数

**5.3实验4.3运行结果**

1. epoch 15, train\_loss 287.851969,test\_loss 0.150231,train\_acc 0.957500,test\_acc 0.957300
2. epoch 16, train\_loss 274.425651,test\_loss 0.144066,train\_acc 0.958467,test\_acc 0.958600
3. epoch 17, train\_loss 261.581794,test\_loss 0.138388,train\_acc 0.960717,test\_acc 0.959800
4. epoch 18, train\_loss 250.021016,test\_loss 0.134018,train\_acc 0.962783,test\_acc 0.961000
5. epoch 19, train\_loss 239.531268,test\_loss 0.128801,train\_acc 0.964250,test\_acc 0.962700
6. epoch 20, train\_loss 229.977510,test\_loss 0.124969,train\_acc 0.966350,test\_acc 0.963700
7. *#####ReLU训练结果#####*
8. epoch 15, train\_loss 425.267893,test\_loss 0.222478,train\_acc 0.936533,test\_acc 0.936300
9. epoch 16, train\_loss 409.630476,test\_loss 0.213813,train\_acc 0.939100,test\_acc 0.939900
10. epoch 17, train\_loss 394.718483,test\_loss 0.207564,train\_acc 0.941217,test\_acc 0.941100
11. epoch 18, train\_loss 379.809425,test\_loss 0.199233,train\_acc 0.943733,test\_acc 0.942300
12. epoch 19, train\_loss 365.807334,test\_loss 0.193883,train\_acc 0.945383,test\_acc 0.945200
13. epoch 20, train\_loss 352.666589,test\_loss 0.185778,train\_acc 0.947467,test\_acc 0.945300
14. *#####ELU训练结果#####*
15. epoch 16, train\_loss 580.820770,test\_loss 0.296475,train\_acc 0.911033,test\_acc 0.915900
16. epoch 17, train\_loss 572.056011,test\_loss 0.292323,train\_acc 0.912317,test\_acc 0.916000
17. epoch 18, train\_loss 563.627448,test\_loss 0.289734,train\_acc 0.913617,test\_acc 0.916000
18. epoch 19, train\_loss 556.127413,test\_loss 0.285581,train\_acc 0.914783,test\_acc 0.917900
19. epoch 20, train\_loss 548.996190,test\_loss 0.281626,train\_acc 0.915633,test\_acc 0.920200
20. *#####Sigmoid训练结果#####*





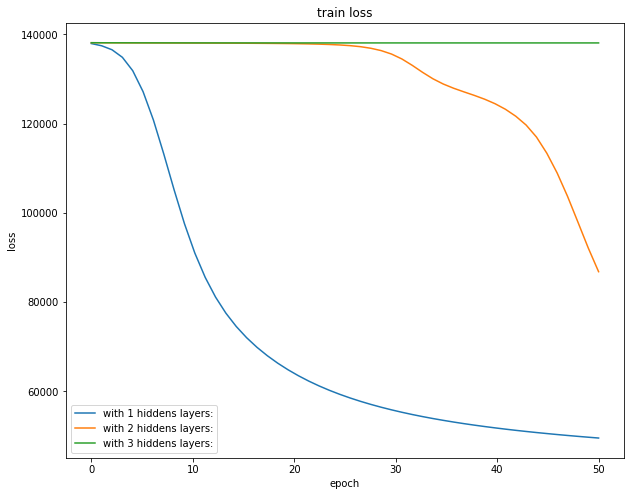


**分析：**根据结果分析可得，随着迭代次数的增加，函数的损失值不断减少。其中ReLU的准确度最高，对应的训练损失最小；ELU训练模型的准确度相比于ReLU模型准确度略低，训练损失也略大；Sigmoid模型的准确度是三个模型中最低的，对应的训练损失也是最大的。

#### 5.4 对多分类任务中的模型评估隐藏层层数和隐藏单元个数对实验结果的影响

**5.4.1实验4.4.1运行结果**

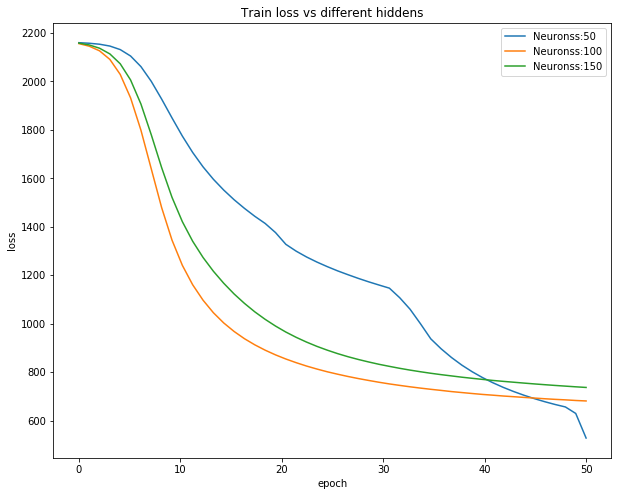
1. epoch: 41, train loss: 51529.647943, test loss: 8459.027495
2. epoch: 42, train loss: 51262.679983, test loss: 8417.045765
3. epoch: 43, train loss: 51009.397888, test loss: 8375.546829
4. epoch: 44, train loss: 50768.532118, test loss: 8337.240803
5. epoch: 45, train loss: 50541.058537, test loss: 8302.553730
6. epoch: 46, train loss: 50319.529032, test loss: 8267.696076
7. epoch: 47, train loss: 50112.346863, test loss: 8235.430816
8. epoch: 48, train loss: 49911.724876, test loss: 8201.537838
9. epoch: 49, train loss: 49721.004353, test loss: 8173.010512
10. epoch: 50, train loss: 49536.304882, test loss: 8142.532957
11. *######使用1个隐藏层######*
12. epoch: 41, train loss: 123233.264297, test loss: 20376.795818
13. epoch: 42, train loss: 121672.497574, test loss: 20068.105915
14. epoch: 43, train loss: 119694.514610, test loss: 19677.682045
15. epoch: 44, train loss: 117004.145313, test loss: 19117.366650
16. epoch: 45, train loss: 113378.062946, test loss: 18420.307346
17. epoch: 46, train loss: 108937.314579, test loss: 17581.485107
18. epoch: 47, train loss: 103716.882614, test loss: 16627.814423
19. epoch: 48, train loss: 97960.869476, test loss: 15622.155939
20. epoch: 49, train loss: 92150.156578, test loss: 14665.078419
21. epoch: 50, train loss: 86823.574966, test loss: 13831.545929
22. *######使用2个隐藏层######*
23. epoch: 41, train loss: 138101.151108, test loss: 23015.874977
24. epoch: 42, train loss: 138101.122101, test loss: 23015.873520
25. epoch: 43, train loss: 138101.096703, test loss: 23015.869911
26. epoch: 44, train loss: 138101.084610, test loss: 23015.867989
27. epoch: 45, train loss: 138101.060265, test loss: 23015.866840
28. epoch: 46, train loss: 138101.042747, test loss: 23015.867458
29. epoch: 47, train loss: 138101.022079, test loss: 23015.865574
30. epoch: 48, train loss: 138101.010254, test loss: 23015.865395
31. epoch: 49, train loss: 138100.993416, test loss: 23015.864651
32. epoch: 50, train loss: 138100.975616, test loss: 23015.861755
33. *######使用3个隐藏层######*



**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏层数量越多，训练模型初始损耗大小下降的越慢，但是迭代到一定的次数后，训练损失下降的趋势越来越大，精度也随之提高，因此可以定性分析二者之间的关系。

**5.4.2实验4.4.2运行结果**

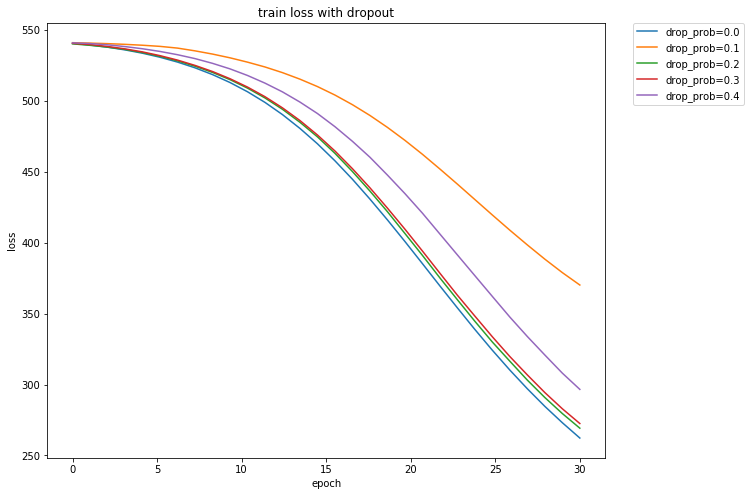
1. epoch: 41, train loss: 755.943523, test loss: 120.543442
2. epoch: 42, train loss: 736.913339, test loss: 117.814164
3. epoch: 43, train loss: 719.841092, test loss: 115.060570
4. epoch: 44, train loss: 704.442430, test loss: 112.611863
5. epoch: 45, train loss: 690.518861, test loss: 110.757044
6. epoch: 46, train loss: 677.866682, test loss: 108.386804
7. epoch: 47, train loss: 666.397066, test loss: 106.479315
8. epoch: 48, train loss: 656.168533, test loss: 104.734380
9. epoch: 49, train loss: 629.347928, test loss: 90.408983
10. epoch: 50, train loss: 527.892548, test loss: 79.910999
11. *######使用50个单元######*
12. epoch: 41, train loss: 704.682214, test loss: 115.323043
13. epoch: 42, train loss: 701.261644, test loss: 114.558937
14. epoch: 43, train loss: 698.456377, test loss: 114.065125
15. epoch: 44, train loss: 695.575150, test loss: 114.074451
16. epoch: 45, train loss: 692.838290, test loss: 113.466566
17. epoch: 46, train loss: 690.003199, test loss: 112.890211
18. epoch: 47, train loss: 687.509210, test loss: 112.696430
19. epoch: 48, train loss: 685.416612, test loss: 112.156060
20. epoch: 49, train loss: 682.848344, test loss: 111.746084
21. epoch: 50, train loss: 680.870272, test loss: 111.119949
22. *######使用100个单元######*
23. epoch: 41, train loss: 765.547214, test loss: 125.951322
24. epoch: 42, train loss: 761.744125, test loss: 125.617559
25. epoch: 43, train loss: 758.206106, test loss: 125.210213
26. epoch: 44, train loss: 754.619243, test loss: 124.233955
27. epoch: 45, train loss: 751.199071, test loss: 123.898542
28. epoch: 46, train loss: 748.010607, test loss: 123.107898
29. epoch: 47, train loss: 744.950927, test loss: 122.974599
30. epoch: 48, train loss: 742.082470, test loss: 122.168594
31. epoch: 49, train loss: 739.249240, test loss: 121.966318
32. epoch: 50, train loss: 736.614097, test loss: 121.881072
33. *######使用150个单元######*



**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏相同时，隐藏单元个数越少，训练模型初始损耗大小下降的越慢，但是迭代到一定的次数后，训练损失下降的趋势越来越大，精度也随之提高。然而训练过程也不是隐藏单元越多，训练损失下降的越快，还需要根据实际情况具体分析。

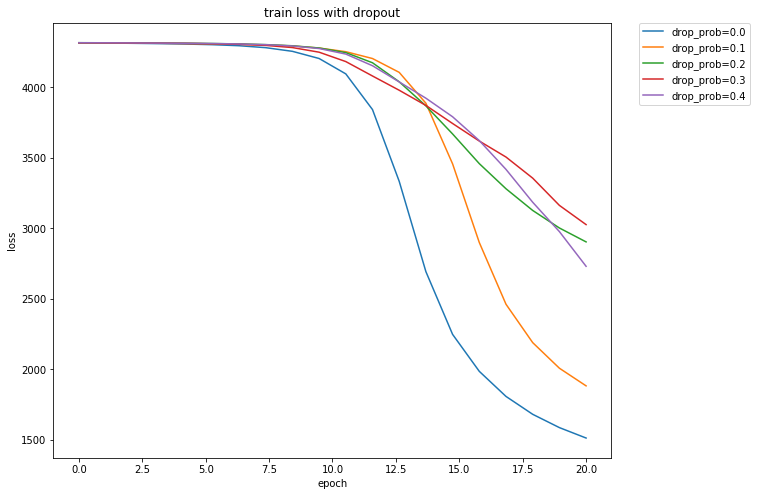
#### 5.5 在多分类任务实验中分别手动实现和用torch.nn实现dropout

**5.5.1实验4.5.1运行结果**



**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏相同时，当隐藏单元的丢弃率为0时，模型的训练损失下降得最快，模型也更容易收敛；也并不是损失率越小损失函数下降得越快，当损失率为0.1时，反而损失函数下降得慢，因此对应特定的训练模型，合理的损失率才可以提高模型的准确度。

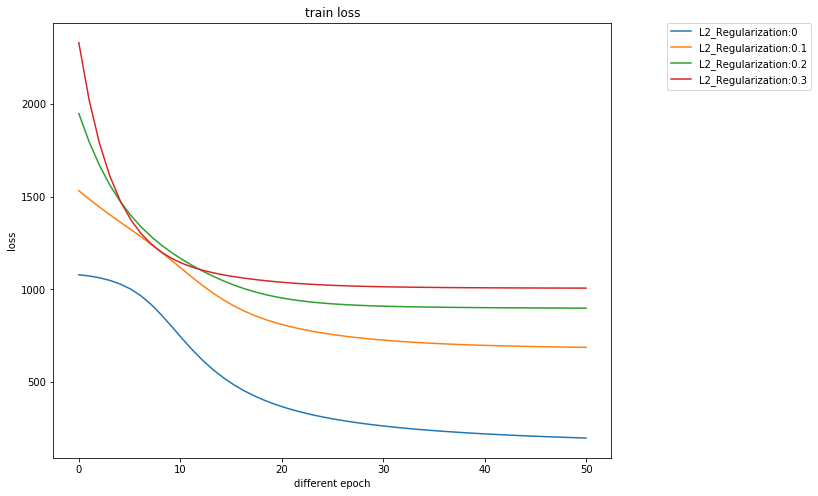
**5.5.2实验4.5.2运行结果**



**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏相同时，当隐藏单元的丢弃率为0时，模型的训练损失下降得最快，模型也更容易收敛；也并不是损失率越小损失函数下降得越快，当损失率为0.3时，反而损失函数下降得慢，因此对应特定的训练模型，合理的损失率才可以提高模型的准确度。

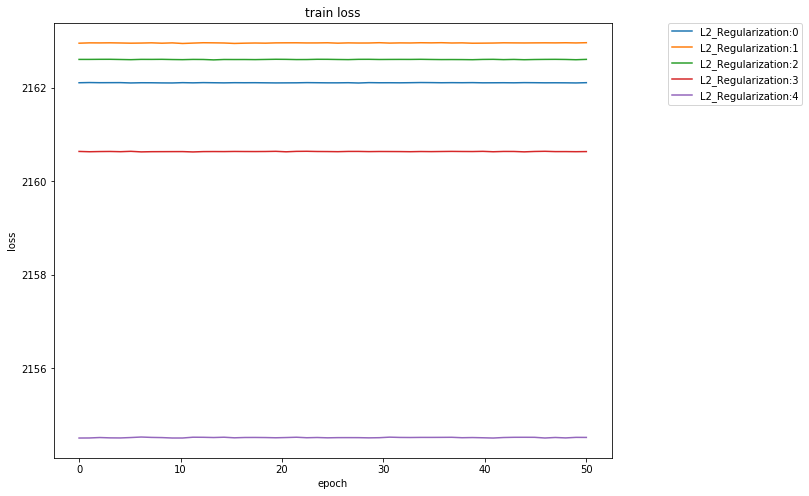
#### 5.6 在多分类任务实验中分别手动实现和用torch.nn实现L2正则化

**5.6.1 实验4.6.1运行结果**



**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏相同时，惩罚项的大小决定了模型的收敛程度。当惩罚项的数值越小，模型越容易收敛，但是由于收敛速度过快，当迭代足够的数量时反而容易提前进入过拟合，相反，过大的惩罚因子也会导致模型难以收敛，因此需要选择合适的惩罚项大小。

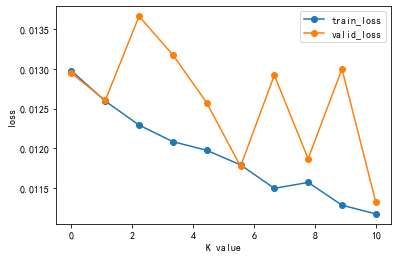
**5.6.2 实验4.6.2运行结果**



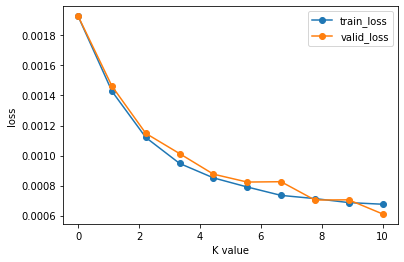
**分析：**根据结果分析可得，在迭代次数相同的情况下，隐藏相同时，惩罚项的大小决定了模型的收敛程度。可能是模型有问题，这导致了模型初始时的Loss较大，但是也可以看出惩罚项越大，函数的损失相对越小。

#### 5.7 对回归、二分类、多分类任务分别选择上述实验中效果最好的模型，采用10折交叉验证评估实验结果

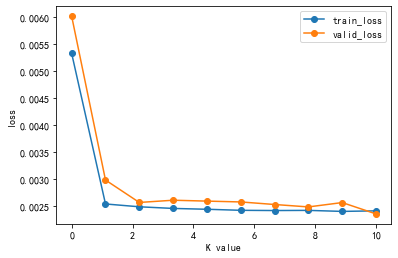
**5.7.1 实验4.7.1运行结果**



**5.7.2 实验4.7.2运行结果**



**5.7.3 实验4.7.3运行结果**



综合5.7.1-5.7.3三个实验的内容，我们可以发现一个大致的规律，采用10折交叉验证评估实验结果，随着折数越大，训练函数的损失越小，最终走向收敛。但是在这个过程中也不难发现，在进行回归实验的过程中，出现了模型损失函数不佳的状况，这很有可能是由于训练函数训练过好从而出现了过拟合的问题，在二分类任务和多分类任务中的表现相对更好，对于回归任务还需要进一步重新修改模型，从而进一步优化训练模型。

## 六、实验心得体会

这次实验时PyTorch的实战操作部分，这里面包含了应用神经网络解决实际问题，包含了回归任务，二分类任务和多回归任务。我分别使用手动模型和torch.nn方法分别校验了训练模型的好坏，也得到了相应的结果。和第一次实验相比，这次实验的难度有了明显的提升，仅仅参考PPT里面的内容几乎无法完成相关的任务，但是我依然坚持自己努力学习，多参考CSDN和博客园上的东西，不抄袭作业，也不向任何学长师兄索要结果，在这个过程中我学会了好多东西，也收获了许多靠得不到的东西。在这个过程中我也发现了自己的编程基础还是相对薄弱的，好多内容还需要在博客园和CSDN的指导下才能进一步完成，在接下来我还会进一步努力，取长补短，争取做得更好。

## 七、参考文献

[1] https://blog.csdn.net/qq\_37534947/article/details/108179408

[2] 动⼿学深度学习-PyTorch

[3] 博客园相关帖子和教学内容

## 八、附录

需要补充说明的内容，如无可略。

**实验报告填写要求**

1. 正文要求小四号宋体，行间距1.5倍；

2. 英文要求小四号Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等。