

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：** |  |
| **学 号：** |  |
| **日 期：** |  |

## 一、实验内容

#### 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

分析实验结果并绘制训练集和测试集的loss曲线

#### 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

分析实验结果并绘制训练集和测试集的loss曲线

#### 3 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

#### 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

使用不同的隐藏层层数和隐藏单元个数，进行对比实验并分析实验结果

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

探究不同丢弃率对实验结果的影响（可用loss曲线进行展示）

#### 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

探究惩罚项的权重对实验结果的影响（可用loss曲线进行展示）

#### 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

要求除了最终结果外还需以表格的形式展示每折的实验结果

## 实验设计

（略）

## 三、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：回归、二分类、多分类任务数据集；MNIST数据集

## 四、实验过程

#### 4.1 手动实现前馈神经网络解决上述回归、二分类、多分类任务

1.回归任务

1）自定义数据集

1. num\_inputs = 500
2. num\_examples = 10000
3. true\_w = torch.ones(500,1)\*0.0056
4. true\_b = 0.028
5. *#随机生成的数据样本*
6. features = torch.tensor(np.random.normal(0, 1, (num\_examples, num\_inputs)), dtype=torch.float)*#行\*列=10000\*500*
7. labels = torch.mm(features,true\_w) + true\_b
8. labels += torch.tensor(np.random.normal(0, 0.01, size=labels.size()), dtype=torch.float) *#扰动项*
9. *#训练集和测试集上的样本&标签数----真实的特征和样本*
10. trainfeatures = features[:7000]
11. trainlabels = labels[:7000]
12. testfeatures = features[7000:]
13. testlabels = labels[7000:]
14. 构造数据迭代器
15. batch\_size = 50 *# 设置小批量大小*
16. def load\_array(data\_arrays, batch\_size, is\_train=True):  *#自定义函数*
17. *#"""构造一个PyTorch数据迭代器。"""*
18. dataset = Data.TensorDataset(\*data\_arrays)*#features 和 labels作为list传入，得到PyTorch的一个数据集*
19. return Data.DataLoader(dataset, batch\_size, shuffle=is\_train,num\_workers=0)*#返回的是实例化后的DataLoader*
20. train\_iter = load\_array([trainfeatures,trainlabels],batch\_size)
21. test\_iter = load\_array([testfeatures,testlabels],batch\_size)
22. 初始化参数
23. *#定义超参数*
24. num\_inputs=500
25. num\_hiddens = 256
26. num\_outputs = 1
27. *#定义参数*
28. W1 = torch.tensor(np.random.normal(0, 0.01, (num\_inputs,num\_hiddens)), dtype=torch.float32)
29. b1 = torch.zeros(1, dtype=torch.float32)
30. W2 = torch.tensor(np.random.normal(0, 0.01, (num\_hiddens,num\_outputs)), dtype=torch.float32)
31. b2 = torch.zeros(1, dtype=torch.float32)
32. params = [W1,b1,W2,b2]
33. for param in params:
34. param.requires\_grad\_(requires\_grad = True)*#设置为true，追踪并记录所有在计算图上的操作（正向积累）*
35. 定义模型、激活函数、损失函数、优化算法
36. def net(X):
37. X = X.view((-1,num\_inputs)) *#将数据进行展平，对于空间结构的数据生效*
38. H = relu(torch.matmul(X,W1)+b1)
39. return torch.matmul(H,W2)+b2
40. def relu(x):
41. x = torch.max(input=x,other=torch.tensor(0.0))
42. return x
43. loss = torch.nn.MSELoss()
44. def SGD (params,lr,batch\_size):
45. for param in params:
46. param.data -= lr \* param.grad/batch\_size
47. 定义训练函数
48. def train (net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params = None,lr=None,optimizer=None):
49. train\_loss=[]
50. test\_loss=[]
51. for epoch in range(num\_epochs):*#外循环控制循环轮次*
52. *#step1在训练集上，进行小批量梯度下降更新参数*
53. for X,y in train\_iter:*#内循环控制训练批次*
54. y\_hat = net(X)
55. l = loss(y\_hat,y)*#l.size = torch.Size([]),即说明loss为表示\*标量\*的tensor`*
56. *#梯度清零*
57. if optimizer is not None:
58. optimizer.zero\_grad()
59. elif params is not None and params[0].grad is not None:
60. for param in params:
61. param.grad.data.zero\_()
62. l.backward()
63. if optimizer is None:
64. SGD(params,lr,batch\_size)
65. else:
66. optimizer.step()
67. *#step2 每经过一个轮次的训练， 记录训练集和测试集上的loss*
68. train\_labels = trainlabels.view(-1,1)
69. test\_labels = testlabels.view(-1,1)
70. train\_loss.append((loss(net(trainfeatures),train\_labels)).item())*#注意取平均值*
71. test\_loss.append((loss(net(testfeatures),test\_labels)).item())
72. print("epoch %d,train\_loss %.6f,test\_loss %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch]))
73. return train\_loss, test\_loss

2.二分类任务

1）自定义数据集

1. num\_inputs = 200
2. *#1类*
3. x1 = torch.normal(2,1,(10000,num\_inputs))
4. y1 = torch.ones(10000,1)*#标签1*
5. x1\_train = x1[:7000]
6. x1\_test = x1[7000:]
7. *#0类*
8. x2 = torch.normal(-2,1,(10000,num\_inputs))
9. y2 = torch.zeros(10000,1)*#标签1*
10. x2\_train = x1[:7000]
11. x2\_test = x1[7000:]
12. *#合并数据----按行合并，即dim=0，且分训练集和测试集*
13. *##合并训练集数据（包括特征和标签）*
14. trainfeatures=torch.cat((x1\_train,x2\_train),0).type(torch.FloatTensor)
15. trainlabels = torch.cat((y1[:7000],y2[:7000]),0).type(torch.FloatTensor)
16. *##合并测试集数据*
17. testfeatures=torch.cat((x1\_test,x2\_test),0).type(torch.FloatTensor)
18. testlabels=torch.cat((y1[7000:],y2[7000:]),0).type(torch.FloatTensor)
19. print(trainfeatures.shape,trainlabels.shape,testfeatures.shape,testlabels.shape)

2）构建数据迭代器

3）定义模型、激活函数、损失函数、优化器

1. def relu(x):
2. x=torch.max(x,torch.tensor(0.0))
3. return x
4. def net (X):
5. X=X.view((-1,num\_inputs))
6. H=relu(torch.mm(X,W1)+b1)
7. return torch.mm(H,W2)+b2
8. *# 定义二分类交叉熵损失函数*
9. loss = torch.nn.BCEWithLogitsLoss()
10. def SGD (params,lr,batch\_size):
11. for param in params:
12. param.data -= lr \* param.grad/batch\_size

4）定义训练函数

1. def train (net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params = None,lr=None,optimizer=None):
2. train\_loss=[]
3. test\_loss=[]
4. for epoch in range(num\_epochs):*#外循环控制循环轮次*
5. train\_l\_sum,test\_l\_sum,n=0.0,0.0,0*#记录训练集上的损失*
6. *#step1在训练集上，进行小批量梯度下降更新参数*
7. for X,y in train\_iter:*#内循环控制训练批次*
8. y\_hat = net(X)
9. *#保证y与y\_hat维度一致，否则将会发生广播*
10. l = loss(y\_hat,y.view(-1,1))*#这里计算出的loss是已经求过平均的，l.size = torch.Size([]),即说明loss为表示\*标量\*的tensor`*
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()
18. if optimizer is None:
19. SGD(params,lr,batch\_size)
20. else:
21. optimizer.step()
22. *#计算每个epoch的loss*
23. *#train\_l\_sum += l.item()*
24. *#n+=y.shape[0]*
25. *#step2 每经过一个轮次的训练， 记录训练集和测试集上的loss*
26. *#注意要取平均值，loss已经默认求了平均值，因此我们不用再老费苦心，直接apply在测试集和训练集上。*
27. test\_l\_sum = loss(net(testfeatures),testlabels).item()
28. train\_l\_sum = loss(net(trainfeatures),trainlabels).item()
29. train\_loss.append(train\_l\_sum)
30. test\_loss.append(test\_l\_sum)
31. print("epoch %d , train\_loss %.6f , test\_loss %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch]))
32. return train\_loss, test\_loss
33. 多分类任务
34. 下载MNIST数据集
35. train\_dataset = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=True,  download=True, transform=transforms.ToTensor())
36. test\_dataset = torchvision.datasets.MNIST(root='./Datasets/MNIST', train=False,  download=True, transform=transforms.ToTensor())
37. 构建数据迭代器
38. 定义模型、激活函数、损失函数、优化器
39. def net (X):
40. *#因为我们忽略了空间结构，所以我们使用reshape将每个二维图像转换为一个长度为num\_inputs的向量。*
41. X = X.reshape((-1,num\_inputs))*#X.shape为torch.Size([32, 1, 28, 28]) 展平为32\*728*
42. H = relu(torch.mm(X,W1)+b1)
43. return (torch.mm(H,W2)+b2)*#所得结果shape为32\*10,代表32个样本,分别在10个输出层神经元上的输出。*
44. def relu(x):
45. x = torch.max(x,torch.tensor(0.0))
46. return x
47. loss = torch.nn.CrossEntropyLoss()
48. def SGD(paras,lr,batch\_size):
49. for param in params:
50. param.data -= lr \* param.grad/batch\_size
51. 定义评价函数
52. def evaluate\_accuracy\_loss(net, data\_iter):
53. acc\_sum=0.0
54. loss\_sum=0.0
55. n=0
56. global flag
57. for X,y in data\_iter:
58. y\_hat = net(X)
59. *#if flag==0:print (y\_hat)#测试一下y\_hat是否已经softmax激活*
60. *#flag = 1*
61. acc\_sum += (y\_hat.argmax(dim=1)==y).sum().item()
62. l = loss(y\_hat,y)
63. loss\_sum += l.item()\*y.shape[0]#由于loss(y\_hat,y)默认为求平均，因此\*y.shape[0]意味着求和。
64. n+=y.shape[0]
65. return acc\_sum/n,loss\_sum/n
66. 定义训练函数
67. def train (net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params = None,lr=None,optimizer=None):
68. train\_loss=[]
69. test\_loss=[]
70. for epoch in range(num\_epochs):*#外循环控制循环轮次---跑完一轮，也就把数据走了一遍*
71. train\_l\_sum=0.0*#记录训练集上的损失*
72. train\_acc\_num=0.0*#记录训练集上的准确数*
73. n =0.0
74. *#step1在训练集上，进行小批量梯度下降更新参数*
75. for X,y in train\_iter:*#内循环控制训练批次*
76. y\_hat = net(X)
77. *#保证y与y\_hat维度一致，否则将会发生广播*
78. l = loss(y\_hat,y)*#这里计算出的loss是已经求和过的，l.size = torch.Size([]),即说明loss为表示\*标量\*的tensor`*
79. *#梯度清零*
80. if optimizer is not None:
81. optimizer.zero\_grad()
82. elif params is not None and params[0].grad is not None:
83. for param in params:
84. param.grad.data.zero\_()
85. l.backward()
86. if optimizer is None:
87. SGD(params,lr,batch\_size)
88. else:
89. optimizer.step()
90. *#每一个迭代周期中得到的训练集上的loss累积进来*
91. train\_l\_sum += l.item()\*y.shape[0]
92. *#计算训练样本的准确率---将每个迭代周期中预测正确的样本数累积进来*
93. train\_acc\_num += (y\_hat.argmax(dim=1)==y).sum().item()*#转为int类型*
94. n += y.shape[0]
95. *#step2 每经过一个轮次的训练， 记录训练集和测试集上的loss*
96. *#注意要取平均值，loss默认求了sum*
97. train\_loss.append(train\_l\_sum/n)*#训练集loss*
98. test\_acc,test\_l = evaluate\_accuracy\_loss(net,test\_iter)
99. test\_loss.append(test\_l)
100. print("epoch %d,train\_loss %.6f,test\_loss %.6f,train\_acc %.6f,test\_acc %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch],train\_acc\_num/n,test\_acc))
101. return train\_loss, test\_loss

#### 4.2 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

1.回归任务

1）自定义数据集

2）构造数据迭代器

3）模型构建

1. *#实现FlattenLayer层：将数据展平*
2. class FlattenLayer (nn.Module):
3. def \_init\_ (self):
4. super(FlattenLayer,self).\_init\_()
5. def forward (self,x):
6. return x.view(x.shape[0],-1)
8. *#模型定义和参数初始化*
9. *# num\_inputs = 500*
10. num\_hiddens = 256
11. num\_outputs = 1
12. net = nn.Sequential(
13. FlattenLayer(),*#输入层*
14. nn.Linear(num\_inputs,num\_hiddens),*#隐藏层*
15. nn.ReLU(),*#隐藏层激活函数Relu*
16. nn.Linear(num\_hiddens,num\_outputs),*#输出层*
17. )

4）参数初始化

1. for param in net.parameters():
2. init.normal\_(param,mean=0,std=0.01)
3. net.parameters()

5）损失函数和优化器

1. loss = torch.nn.MSELoss()
2. optimizer = torch.optim.SGD(net.parameters(),lr)

6）定义训练函数

1. def train1 (net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params = None,lr=None,optimizer=None):
2. train\_loss=[]
3. test\_loss=[]
4. for epoch in range(num\_epochs):*#外循环控制循环轮次*
5. *#step1在训练集上，进行小批量梯度下降更新参数*
6. for X,y in train\_iter:*#内循环控制训练批次*
7. y\_hat = net(X)
8. l = loss(y\_hat,y)*#l.size = torch.Size([]),即说明loss为表示\*标量\*的tensor`*
9. *#梯度清零*
10. if optimizer is not None:
11. optimizer.zero\_grad()
12. elif params is not None and params[0].grad is not None:
13. for param in params:
14. param.grad.data.zero\_()
15. l.backward()
16. if optimizer is None:
17. SGD(params,lr,batch\_size)
18. else:
19. optimizer.step()
20. *#step2 每经过一个轮次的训练， 记录训练集和测试集上的loss*
21. train\_labels = trainlabels.view(-1,1)
22. test\_labels = testlabels.view(-1,1)
23. train\_loss.append((loss(net(trainfeatures),train\_labels)).item())*#loss本身就默认了取平均值！*
24. test\_loss.append((loss(net(testfeatures),test\_labels)).item())
25. print("epoch %d,train\_loss %.6f,test\_loss %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch]))
26. return train\_loss, test\_loss

2. 二分类任务

1. 自定义数据集
2. 构建数据迭代器
3. 模型构建
4. 参数初始化
5. 损失函数和优化器
6. *# 定义二分类交叉熵损失函数*
7. loss = torch.nn.BCEWithLogitsLoss()
8. *# 定义sgd优化器*
9. optimizer = torch.optim.SGD(net.parameters(),lr)
10. 定义训练函数
11. *#定义模型训练函数*
12. def train2(net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params=None,lr=None,optimizer=None):
13. train\_ls = []
14. test\_ls = []
15. for epoch in range(num\_epochs): *# 训练模型一共需要num\_epochs个迭代周期*
16. train\_l\_sum, train\_acc\_num,n = 0.0,0.0,0
17. *# 在每一个迭代周期中，会使用训练数据集中所有样本一次*
18. for X, y in train\_iter: *# x和y分别是小批量样本的特征和标签*
19. y\_hat = net(X)
20. l = loss(y\_hat, y.view(-1,1)) *# l是有关小批量X和y的损失*
21. *#梯度清零*
22. if optimizer is not None:
23. optimizer.zero\_grad()
24. elif params is not None and params[0].grad is not None:
25. for param in params:
26. param.grad.data.zero\_()
27. l.backward() *# 小批量的损失对模型参数求梯度*
28. if optimizer is None:
29. SGD(params,lr)
30. else:
31. optimizer.step()
32. *#计算每个epoch的loss*
33. train\_l\_sum += l.item()\*y.shape[0]
34. *#每一个epoch的所有样本数*
35. n+= y.shape[0]
36. train\_labels = trainlabels.view(-1,1)
37. test\_labels = testlabels.view(-1,1)
38. train\_ls.append(train\_l\_sum/n)
39. test\_ls.append(loss(net(testfeatures),test\_labels).item())
40. print("epoch %d,train\_loss %.6f,test\_loss %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch]))
41. return train\_ls,test\_ls
42. 多分类任务
43. 下载MNIST数据集
44. 定义数据迭代器
45. 模型构建
46. 参数初始化
47. 损失函数和优化器
48. loss = torch.nn.CrossEntropyLoss()
49. optimizer = optim.SGD(net.parameters(),lr)
50. 定义训练函数
51. def train3 (net,train\_iter,test\_iter,loss,num\_epochs,batch\_size,params = None,lr=None,optimizer=None):
52. train\_loss=[]
53. test\_loss=[]
54. for epoch in range(num\_epochs):*#外循环控制循环轮次---跑完一轮，也就把数据走了一遍*
55. train\_l\_sum=0.0*#记录训练集上的损失*
56. train\_acc\_num=0.0*#记录训练集上的准确数*
57. n =0.0
58. *#step1在训练集上，进行小批量梯度下降更新参数*
59. for X,y in train\_iter:*#内循环控制训练批次*
60. y\_hat = net(X)
61. *#保证y与y\_hat维度一致，否则将会发生广播*
62. l = loss(y\_hat,y)*#这里计算出的loss是已经求和过的，l.size = torch.Size([]),即说明loss为表示\*标量\*的tensor`*
63. *#梯度清零*
64. if optimizer is not None:
65. optimizer.zero\_grad()
66. elif params is not None and params[0].grad is not None:
67. for param in params:
68. param.grad.data.zero\_()
69. l.backward()
70. if optimizer is None:
71. SGD(params,lr,batch\_size)
72. else:
73. optimizer.step()
74. *#每一个迭代周期中得到的训练集上的loss累积进来*
75. train\_l\_sum += l.item()\*y.shape[0]
76. *#计算训练样本的准确率---将每个迭代周期中预测正确的样本数累积进来*
77. train\_acc\_num += (y\_hat.argmax(dim=1)==y).sum().item()*#转为int类型*
78. n += y.shape[0]
79. *#step2 每经过一个轮次的训练， 记录训练集和测试集上的loss*
80. *#注意要取平均值，loss默认求了sum*
81. train\_loss.append(train\_l\_sum/n)*#训练集loss*
82. test\_acc,test\_l = evaluate\_accuracy\_loss(net,test\_iter)
83. test\_loss.append(test\_l)
84. print("epoch %d,train\_loss %.6f,test\_loss %.6f,train\_acc %.6f,test\_acc %.6f"%(epoch+1,train\_loss[epoch],test\_loss[epoch],train\_acc\_num/n,test\_acc))
85. return train\_loss, test\_loss

#### 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

1. net = nn.Sequential(
2. FlattenLayer(),
3. nn.Linear(num\_inputs,num\_hiddens),
4. nn.ReLU(),
5. *#nn.Sigmoid(),*
6. *#nn.Tanh(),*
7. nn.Linear(num\_hiddens,num\_outputs)
8. )

#### 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

1）隐藏层层数即定义了几个num\_hiddens，

1. num\_inputs, num\_outputs, num\_hiddens1, num\_hiddens2 = 784, 10, 256, 256 *#隐藏层个数为2*

2）隐藏单元个数即num\_hiddens值，

1. num\_inputs=784 *#28\*28*
2. num\_hiddens=256 *#num\_hiddens的值即隐层单元个数，还可为16,32,64...512,1024*
3. num\_outputs=10

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

1.手动实现dropout

1）定义dropout函数

1. def dropout(X, drop\_prob):
2. X = X.float()
3. assert 0 <= drop\_prob <= 1
4. keep\_prob = 1 - drop\_prob
5. if keep\_prob == 0:
6. return torch.zeros\_like(X)
7. mask = (torch.rand(X.shape) < keep\_prob).float()
8. return mask \* X / keep\_prob

2）定义模型

1. def net(X, is\_training=True):
2. X = X.view(-1, num\_inputs)
3. H1 = (torch.matmul(X, W1.t()) + b1).relu()
4. if is\_training:
5. H1 = dropout(H1, drop\_prob1)
6. return (torch.matmul(H1,W2.t()) + b2).relu()

2.torch.nn实现dropout

1. 定义模型
2. class LinearNet(nn.Module):
3. def \_\_init\_\_(self,num\_inputs, num\_outputs, num\_hiddens1, num\_hiddens2, drop\_prob1,drop\_prob2):
4. super(LinearNet,self).\_\_init\_\_()
5. self.linear1 = nn.Linear(num\_inputs,num\_hiddens1)
6. self.relu = nn.ReLU()
7. self.drop1 = nn.Dropout(drop\_prob1)
8. self.linear2 = nn.Linear(num\_hiddens1,num\_hiddens2)
9. self.drop2 = nn.Dropout(drop\_prob2)
10. self.linear3 = nn.Linear(num\_hiddens2,num\_outputs)
11. self.flatten  = nn.Flatten()
12. def forward(self,x):
13. x = self.flatten(x)
14. x = self.linear1(x)
15. x = self.relu(x)
16. x = self.drop1(x)
17. x = self.linear2(x)
18. x = self.relu(x)
19. x = self.drop2(x)
20. x = self.linear3(x)
21. y = self.relu(x)
22. return y

**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**

1.手动实现L2正则化

1）定义L2范数惩罚项

1. def l2\_penalty(W1, W2):  # 定义L2范数惩罚项
2. return (W1 \*\* 2).sum() / 2 + (W2 \*\* 2).sum() / 2
3. 定义训练函数
4. def train\_ch3(net, train\_iter, test\_iter, loss, num\_epochs, batch\_size, params , lr, lambd):
5. train\_ls, test\_ls, x\_epoch = [], [], []
6. for epoch in range(num\_epochs):
7. train\_1\_sum = 0.0
8. train\_1\_test\_sum = 0.0
9. train\_acc\_sum = 0.0
10. n = 0
11. n\_test = 0
12. for X, y in train\_iter:
13. y\_hat = net(X)
14. l = loss(y\_hat, y) + lambd \* l2\_penalty(W1, W2)    #添加惩罚项
15. l = l.sum()
16. if params[0].grad is not None:
17. for param in params:
18. param.grad.data.zero\_()
19. l.backward()
20. sgd(params, lr, batch\_size)
21. train\_1\_sum += l.item()
22. train\_acc\_sum += (y\_hat.argmax(dim =1) == y).sum().item()
23. n += y.shape[0]
24. train\_ls.append(train\_1\_sum / n)
25. x\_epoch.append(epoch + 1)
26. test\_acc = evaluate\_accuracy(test\_iter, net)
27. print('epoch %d, loss %.4f, train acc %.3f, test acc %.3f' % (epoch + 1, train\_1\_sum / n, train\_acc\_sum / n, test\_acc))

2.torch.nn实现L2正则化

利用torch.optim的weight\_decay参数实现L2范数正则化

1. optimizer = torch.optim.SGD(net.parameters(), lr=0.01, weight\_decay=0.01)

**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**

1. **回归10折交叉验证**

1）获取k折交叉验证某一折的训练集和验证集

1. def get\_k\_fold\_data(k, i, X, y):
2. *# 返回第i折交叉验证时所需要的训练和验证数据，分开放，X\_train为训练数据，X\_valid为验证数据*
3. assert k > 1
4. fold\_size = X.shape[0] // k
5. X\_train, y\_train = None, None
6. for j in range(k):
7. idx = slice(j \* fold\_size, (j + 1) \* fold\_size)  *#slice(start,end,step)切片函数*
8. X\_part, y\_part = X[idx, :], y[idx]
9. if j == i:
10. X\_valid, y\_valid = X\_part, y\_part
11. elif X\_train is None:
12. X\_train, y\_train = X\_part, y\_part
13. else:
14. X\_train = torch.cat((X\_train, X\_part), dim=0)
15. y\_train = torch.cat((y\_train, y\_part), dim=0)
16. return X\_train, y\_train, X\_valid, y\_valid

2）依次对每一折数据进行训练和测试，并计算k折平均值

1. def k\_fold(k, X\_train, y\_train, num\_epochs, batch\_size):
2. for i in range(k):
3. data = get\_k\_fold\_data(k, i, X\_train, y\_train) *# 获取k折交叉验证的训练和验证数据*
4. train\_ls, valid\_ls = train(\*data)
5. train\_l\_sum, valid\_l\_sum, train\_acc\_sum, valid\_acc\_sum = 0.0, 0.0, 0.0, 0.0
6. for ii in train\_ls:
7. train\_l\_sum += ii
8. for jj in valid\_ls:
9. valid\_l\_sum += jj
10. print('fold %d, train loss %.6f, valid loss %.6f' % (i + 1, train\_l\_sum / num\_epochs, valid\_l\_sum / num\_epochs))
11. 二分类10折交叉验证
12. def get\_k\_fold\_data(k, i, X, y):
13. *# 返回第i折交叉验证时所需要的训练和验证数据，分开放，X\_train为训练数据，X\_valid为验证数据*
14. assert k > 1
15. fold\_size = X.shape[0] // k  # 双斜杠表示除完后再向下取整
16. X\_train, y\_train = None, None
17. for j in range(k):
18. idx = slice(j \* fold\_size, (j + 1) \* fold\_size) #slice(start,end,step)切片函数
19. X\_part, y\_part = X[idx, :], y[idx]
20. if j == i:
21. X\_valid, y\_valid = X\_part, y\_part
22. elif X\_train is None:
23. X\_train, y\_train = X\_part, y\_part
24. else:
25. X\_train = torch.cat((X\_train, X\_part), dim=0)
26. y\_train = torch.cat((y\_train, y\_part), dim=0)
27. return X\_train, y\_train, X\_valid, y\_valid
28. def k\_fold(k, X\_train, y\_train, num\_epochs, batch\_size):
29. for i in range(k):
30. data = get\_k\_fold\_data(k, i, X\_train, y\_train) # 获取k折交叉验证的训练和验证数据
31. train\_ls, valid\_ls, train\_acc, valid\_acc = train(batch\_size, num\_epochs, lr, \*data)
32. train\_l\_sum, valid\_l\_sum, train\_acc\_sum, valid\_acc\_sum = 0.0, 0.0, 0.0, 0.0
33. for ii in train\_ls:
34. train\_l\_sum += ii
35. for jj in valid\_ls:
36. valid\_l\_sum += jj
37. for ii in train\_acc:
38. train\_acc\_sum +=ii
39. for jj in valid\_acc:
40. valid\_acc\_sum +=jj
41. print('fold %d, train loss %.4f, valid loss %.4f, train acc %.4f, test acc %.4f' % (i + 1, train\_l\_sum / num\_epochs, valid\_l\_sum / num\_epochs, train\_acc\_sum / num\_epochs, valid\_acc\_sum / num\_epochs))
42. 多分类10折交叉验证
43. def get\_k\_fold\_data(k, i, X, y):
44. *# 返回第i折交叉验证时所需要的训练和验证数据，分开放，X\_train为训练数据，X\_valid为验证数据*
45. assert k > 1
46. fold\_size = X.shape[0] // k
47. X\_train, y\_train = None, None
48. for j in range(k):
49. idx = slice(j \* fold\_size, (j + 1) \* fold\_size) #slice(start,end,step)切片函数
50. X\_part, y\_part = X[idx, :], y[idx]
51. if j == i:
52. X\_valid, y\_valid = X\_part, y\_part
53. elif X\_train is None:
54. X\_train, y\_train = X\_part, y\_part
55. else:
56. X\_train = torch.cat((X\_train, X\_part), dim=0)
57. y\_train = torch.cat((y\_train, y\_part), dim=0)
58. return X\_train, y\_train, X\_valid, y\_valid
59. def k\_fold(k, X\_train, y\_train, num\_epochs, batch\_size):
60. for i in range(k):
61. data = get\_k\_fold\_data(k, i, X\_train, y\_train) # 获取k折交叉验证的训练和验证数据
62. train\_ls, valid\_ls, train\_acc, valid\_acc = train(\*data, num\_epochs, batch\_size)
63. train\_l\_sum, valid\_l\_sum, train\_acc\_sum, valid\_acc\_sum = 0.0, 0.0, 0.0, 0.0
64. for ii in train\_ls:
65. train\_l\_sum += ii
66. for jj in valid\_ls:
67. valid\_l\_sum += jj
68. for ii in train\_acc:
69. train\_acc\_sum += ii
70. for jj in valid\_acc:
71. valid\_acc\_sum += jj
72. print('fold %d, train loss %.4f, valid loss %.4f, train acc %.4f, test acc %.4f' % (i + 1, train\_l\_sum / num\_epochs, valid\_l\_sum / num\_epochs, train\_acc\_sum / num\_epochs, valid\_acc\_sum / num\_epochs))

## 五、实验结果

#### 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

1.手动实现回归任务结果图如5.1所示。

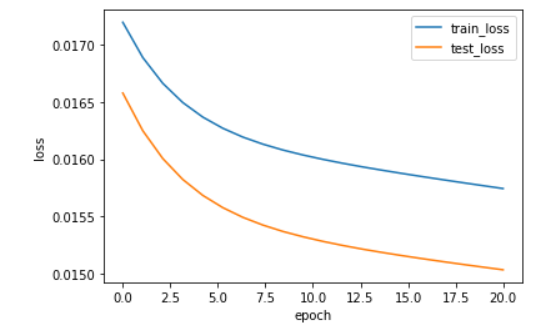


图5.1 手动实现回归任务结果图

2.手动实现二分类任务结果图如图5.2所示。

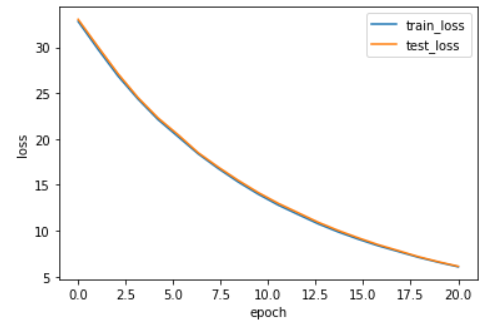


图5.2 手动实现二分类任务结果图

3.手动实现多分类任务结果图如图5.3所示。

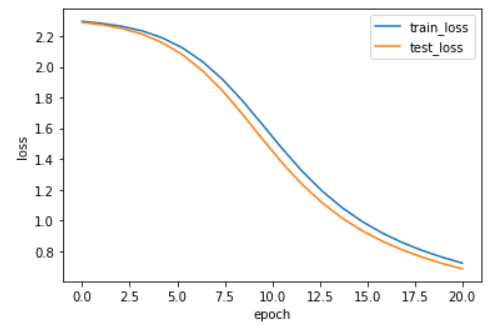


图5.3 手动实现多分类任务结果图

#### 5.2 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

1.torch.nn实现回归任务结果图如图5.4所示。

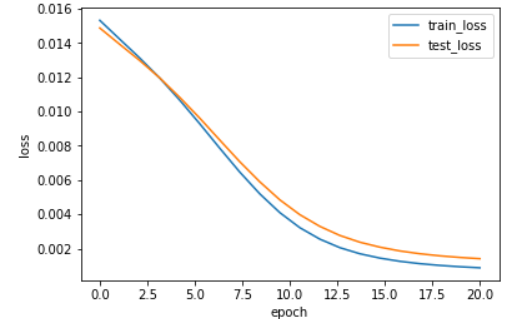


图5.4 torch. nn实现回归任务结果图

2.torch.nn实现二分类任务结果图如图5.5所示。

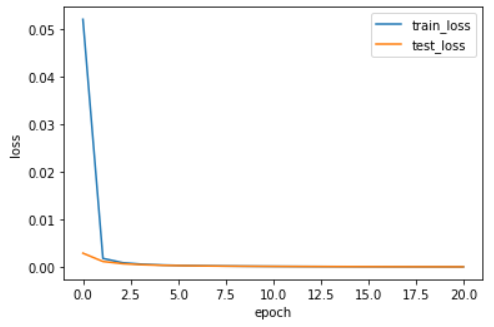


图5.5 torch.nn实现二分类任务结果图

3.torch.nn实现多分类任务结果图如图5.6所示。

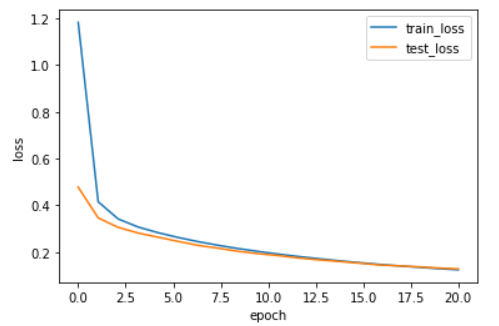


图5.6 torch.nn实现多分类任务结果图

#### 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

1.采用ReLu激活函数结果图如上图5.6所示。

2.采用Sigmoid激活函数结果图如图5.7所示。

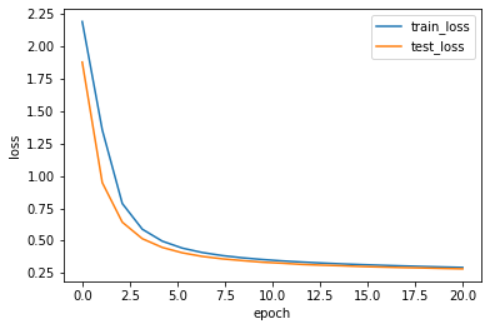


图5.7 Sigmoid激活函数结果图

3.采用Tanh激活函数结果图如图5.8所示。

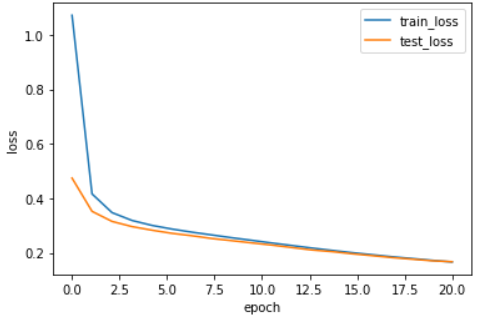


图5.8 Tanh激活函数结果图

#### 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

1.隐藏层个数为1，隐藏单元个数为256结果图如上图5.6所示。

2.隐藏层个数为1，隐藏单元个数为64结果图如图5.9所示。

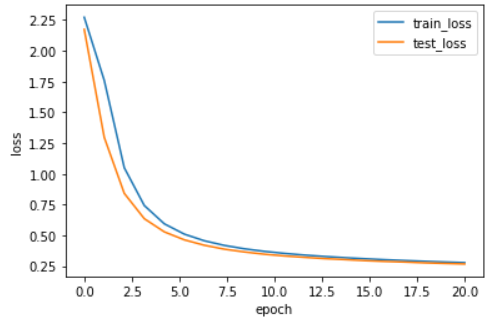


图5.9 隐藏层个数为1，隐藏单元个数为64结果图

3.隐藏层个数为2，隐藏单元个数为256结果图如图5.10所示。

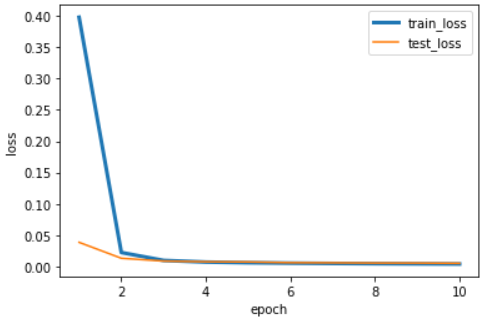


图5.10 隐藏层个数为2，隐藏单元个数为256结果图

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

1.手动实现dropout结果图如图5.11所示。

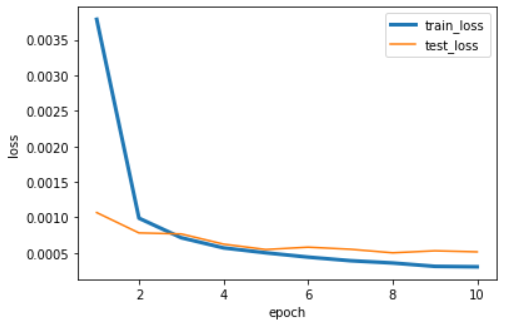


图5.11 手动实现dropout结果图

2.torch.nn实现dropout结果图如图5.12所示。

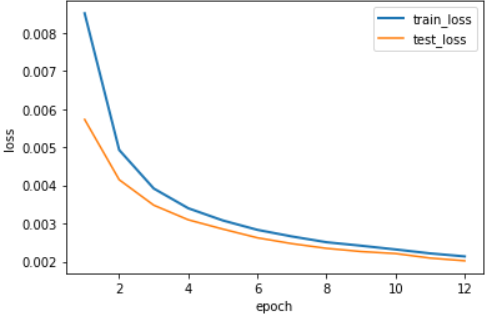


图5.12 torch.nn实现dropout结果图

#### 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

1.手动实现L2正则化结果图如图5.13所示。

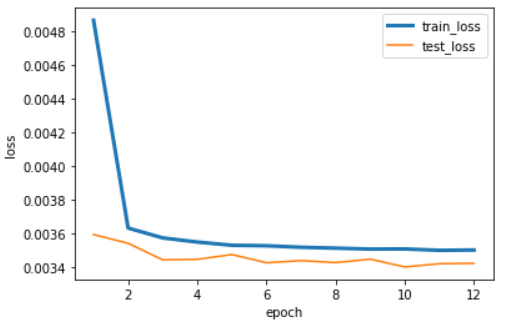


图5.13 手动实现L2正则化结果图

2.torch.nn实现L2正则化结果图如图5.14所示。

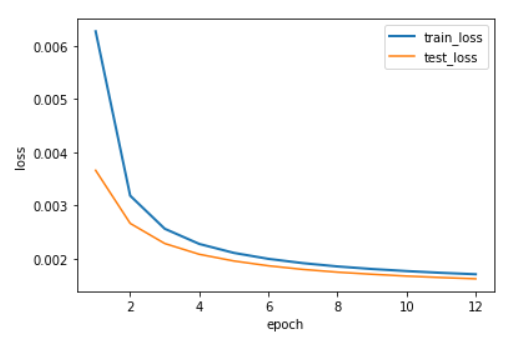


图5.14 torch.nn实现L2正则化结果图

#### 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

1.回归10折交叉验证评估实验结果表如表5.1所示。

表5.1 回归10折交叉验证评估实验结果表

|  |  |  |
| --- | --- | --- |
| fold | train loss | valid loss |
| 1 | 0.002749 | 0.000194 |
| 2 | 0.003578 | 0.000232 |
| 3 | 0.003604 | 0.000236 |
| 4 | 0.002041 | 0.000147 |
| 5 | 0.002485 | 0.000180 |
| 6 | 0.002277 | 0.000169 |
| 7 | 0.004013 | 0.000252 |
| 8 | 0.002703 | 0.000194 |
| 9 | 0.002221 | 0.000156 |
| 10 | 0.002759 | 0.000204 |

2.二分类10折交叉验证评估实验结果表如表5.2所示。

表5.2 二分类10折交叉验证评估实验结果表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| fold | train loss | valid loss | train acc | test acc |
| 1 | 0.0106 | 0.0010 | 0.9993 | 1.0000 |
| 2 | 0.0106 | 0.0010 | 0.9992 | 1.0000 |
| 3 | 0.0106 | 0.0010 | 0.9996 | 1.0000 |
| 4 | 0.0106 | 0.0010 | 0.9994 | 1.0000 |
| 5 | 0.0109 | 0.0010 | 0.9990 | 1.0000 |
| 6 | 0.0107 | 0.0010 | 0.9990 | 1.0000 |
| 7 | 0.0107 | 0.0010 | 0.9983 | 1.0000 |
| 8 | 0.0112 | 0.0010 | 0.9981 | 1.0000 |
| 9 | 0.0105 | 0.0010 | 0.9997 | 1.0000 |
| 10 | 0.0107 | 0.0010 | 0.9989 | 1.0000 |

3.多分类10折交叉验证评估实验结果表如表5.3所示。

表5.3 多分类10折交叉验证评估实验结果表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| fold | train loss | valid loss | train acc | test acc |
| 1 | 0.0007 | 0.0007 | 0.9598 | 0.9612 |
| 2 | 0.0007 | 0.0007 | 0.9602 | 0.9617 |
| 3 | 0.0007 | 0.0008 | 0.9593 | 0.9585 |
| 4 | 0.0007 | 0.0007 | 0.9596 | 0.9602 |
| 5 | 0.0007 | 0.0008 | 0.9596 | 0.9596 |
| 6 | 0.0007 | 0.0008 | 0.9594 | 0.9602 |
| 7 | 0.0007 | 0.0008 | 0.9605 | 0.9613 |
| 8 | 0.0007 | 0.0008 | 0.9613 | 0.9616 |
| 9 | 0.0007 | 0.0008 | 0.9605 | 0.9609 |
| 10 | 0.0007 | 0.0006 | 0.9592 | 0.9608 |

## 六、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，在这次的实验过程中我对前馈神经网络的逻辑有了更清楚的认识，虽然在这次实验问题上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，实现了前馈神经网络回归、二分类、多分类任务的实现，对于深度学习实验代码的编写也有了很大的提升。

## 七、参考文献

## 八、附录

（略）