Lecture 10-11 编程作业

1.IRIS神经网络

(1) 实验要求

自行设计神经网络实现对这三个目标的识别，实验时每个类别随机选30个样本进行训练，另外20个样本用于测试。希望能通过设计不同的隐含层数、每层的节点数、不同的学习率、不同的激活函数等对实验结果进行讨论。

代码见iris_neural.py

(2) 核心算法

```
class MLP(nn.Module):
    # 声明带有模型参数的层，这里声明了两个全连接层
    def __init__(self, **kwargs):
        super(MLP, self).__init__(**kwargs)
        self.hidden = nn.Linear(4, 32) # 隐藏层
        self.act = nn.ReLU()
        self.output = nn.Linear(32,3)  # 输出层
    # 定义模型的前向计算，即如何根据输入x计算返回所需要的模型输出
    def forward(self, x):
        a = self.act(self.hidden(x))
        return self.output(a)

def train_iris(data,label,net,epoch,alpha):      #训练神经网络
    for params in net.parameters():              #网络参数初始化
        init.normal_(params, mean=0, std=0.01)
        params.requires_grad_(requires_grad=True)
    optimizer = torch.optim.SGD(net.parameters(), lr=alpha) #设置梯度下降
    for t in range(epoch):
        y_hat=softmax(net(data))                 #求出每个样本分到各标签的概率
        l = cross_entropy(y_hat, label).sum()    #计算交叉熵损失函数
        l.backward()                             #求梯度
        optimizer.step()
        for params in net.parameters():
            params.grad.zero_()                  #梯度重置为零
    return net
```

(3) 每层的节点数对实验结果的影响

1层隐含层，模型4-2-3，全连接，激活函数为$ReLU$,学习率$lr=0.001$，采用Adam法.

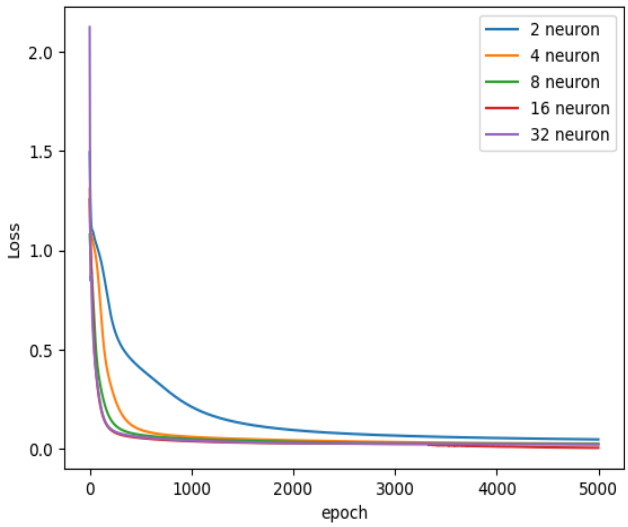
1层隐含层，模型4-4-3，全连接，激活函数为$ReLU$,学习率$lr=0.001$，采用Adam法.

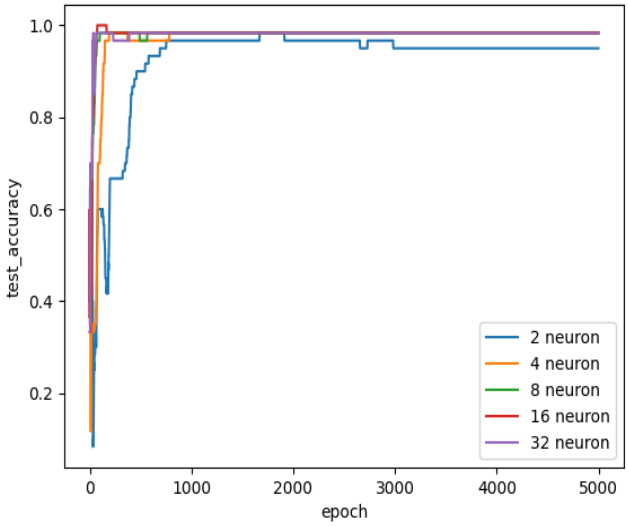
1层隐含层，模型4-8-3，全连接，激活函数为$ReLU$,学习率$lr=0.001$，采用Adam法.

1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率$lr=0.001$，采用Adam法.

1层隐含层，模型4-32-3，全连接，激活函数为$ReLU$,学习率$lr=0.001$，采用Adam法.

不同节点数损失函数和测试集正确率随迭代次数变化曲线





数据分析

训练过程中，单层中神经元个数越多，相同次数时损失率(误差)越低，测试集正确率越高.

(4) 不同的隐含层数对实验结果的影响

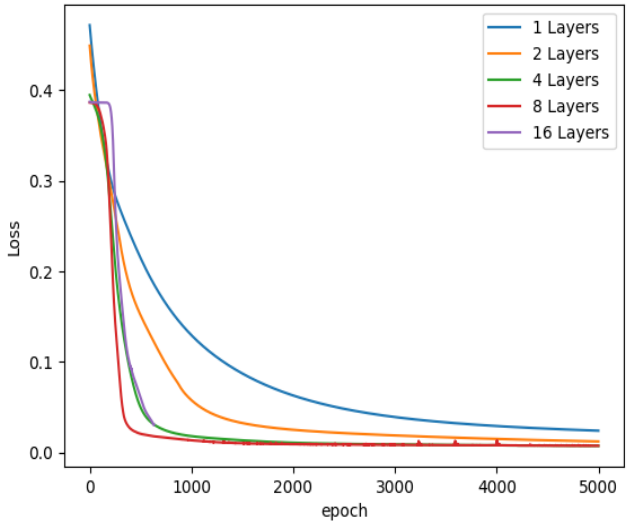
1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr = 0.0003$，采用Adam法.

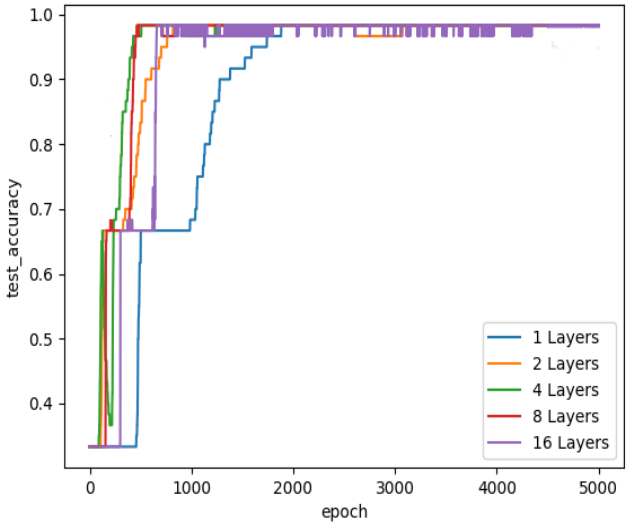
2层隐含层，模型4-16-16-3，全连接，激活函数为$ReLU$,学习率 $lr = 0.0003$，采用Adam法.

4层隐含层，模型4-16-16-16-16-3，全连接，激活函数为$ReLU$,学习率 $lr = 0.0003$，采用Adam法.

8层隐含层，模型4-16-16-16-...(8 layers)-3，全连接，激活函数为$ReLU$,学习率 $lr = 0.0003$，采用Adam法.

不同隐含层数损失函数和测试集正确率随迭代次数变化曲线





数据分析

训练过程中，伴随着深度的加深，神经网络逐渐演化为深度神经网络，相同迭代次数时损失率越低。单次训练时间明显增加，但训练效果逐渐变好。

通过预测结果分析，4层之前产生了部分的欠拟合，而16层由于层数过深，升维过高，导致过拟合现象明显，分类结果不如8层。

(5) 不同的学习率对实验结果的影响

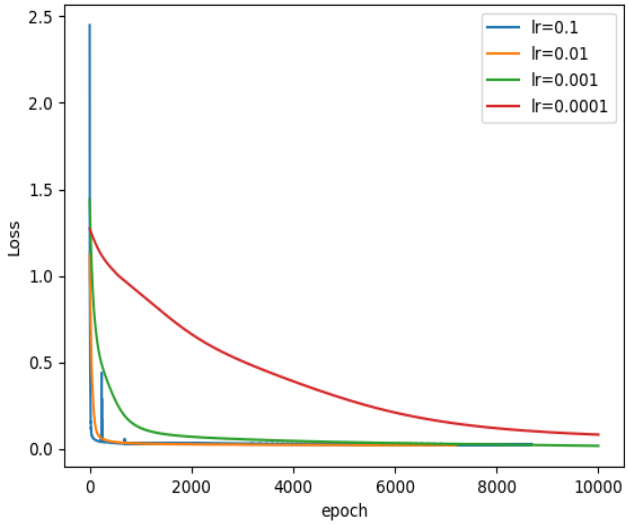
1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr=0.1$，采用Adam法.

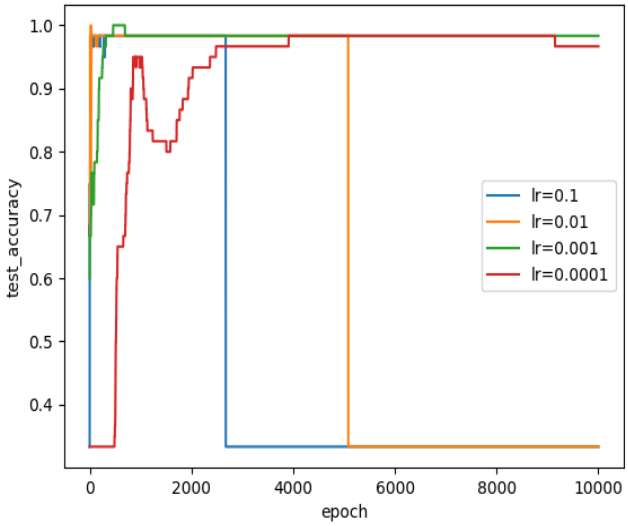
1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr=0.01$，采用Adam法.

1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr=0.001$，采用Adam法.

1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr=0.0001$，采用Adam法.

不同的学习率损失函数和测试集正确率随迭代次数变化曲线





数据分析

训练过程中，学习率为0.1的很难迭代到最小值，因为梯度改变的步长较大，导致其很容易来回横跳。学习率为0.01以下的训练效果较好。但当迭代次数较小时，学习率偏小的很难找到梯度最小的。通过预测结果分析，学习率越小，预测效果基本上会更好一些。但当学习率为0.0001时，损失率下降过慢，容易产生过拟合现象。

(5) 不同的激活函数对实验结果的影响

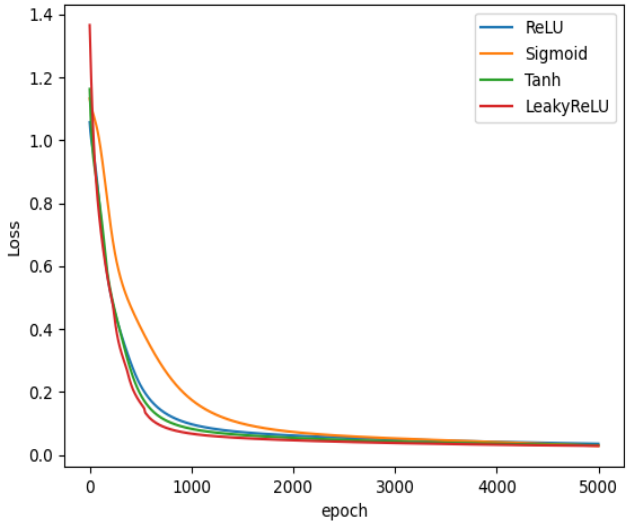
1层隐含层，模型4-16-3，全连接，激活函数为$ReLU$,学习率 $lr = 0.001$，采用Adam法.

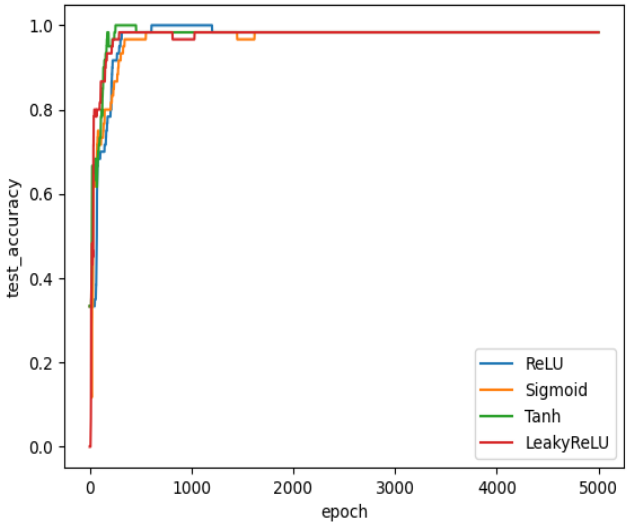
1层隐含层，模型4-16-3，全连接，激活函数为$Sigmoid$,学习率 $lr = 0.001$，采用Adam法.

1层隐含层，模型4-16-3，全连接，激活函数为$Tanh$,学习率 $lr = 0.001$，采用Adam法.

1层隐含层，模型4-16-3，全连接，激活函数为$Leaky-ReLU$,学习率 $lr = 0.001$，采用Adam法.

不同的激活函数损失函数和测试集正确率随迭代次数变化曲线





数据分析

训练过程中，Sigmoid 函数迭代到了局部最小值后不再迭代，其他三者都迭代到了全局最小值。相同迭代次数内, LeakReLU的训练效果更好。

通过预测结果分析，Sigmoid 函数的激活效果明显不如其他三者，而这三者中，ReLU函数的效果最优。

2.LeNet神经网络

代码见 LeNet.py

(1) 核心算法

```
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        self.conv = nn.Sequential(       #卷积层提特征
            nn.Conv2d(1, 6, 5,stride=1,padding=2),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2),
            nn.Conv2d(6, 16, 5,stride=1,padding=0),
            nn.Sigmoid(),
            nn.MaxPool2d(2, 2)
```

```
        )
        self.fc = nn.Sequential(         #全连接层
            nn.Linear(16*5*5, 120),
            nn.Sigmoid(),
            nn.Linear(120, 84),
            nn.Sigmoid(),
            nn.Linear(84, 10)
        )
    def forward(self, img):
        feature = self.conv(img)
        output = self.fc(feature.view(img.shape[0], -1))
        return output

def train_LeNet(train_iter,test_iter,net,epoch,alpha):  #训练LeNet神经网络
    for params in net.parameters():
        init.normal_(params, mean=0, std=0.01)
        params.requires_grad_(requires_grad=True)        #初始化网络参数
    ac=np.zeros(epoch)
    L=np.zeros(epoch)
    optimizer = torch.optim.Adam(net.parameters(), lr=alpha)     #采用adam法梯度下降
    test_ac=np.zeros(epoch)
    for t in range(epoch):
        count=0
        for X, y in train_iter:
            y_hat = softmax(net(X))                          #求出每个样本分到各标签的概率
            L[t]+=cross_entropy(y_hat, y).sum()
            ac[t]+=accuracy(y_hat,y)
            l = cross_entropy(y_hat,y).sum()                 #求出此batch内交叉熵损失函数
            l.backward()                                     #反向传播
            optimizer.step()
            for params in net.parameters():
                params.grad.zero_()                          #梯度重置为0
            count+=1

        L[t]=L[t]/count
        ac[t]=ac[t]/count
        test_ac[t]=test(test_iter,net)
    epoch_plot(epoch,L,ac,test_ac)     #画出迭代函数曲线
    print("mnist_train accuracy:",ac[epoch-1])               #输出训练集分类准确率
    print("mnist_test accuracy:",test_ac[epoch-1])           #输出测试集分类准确率
    return net
```

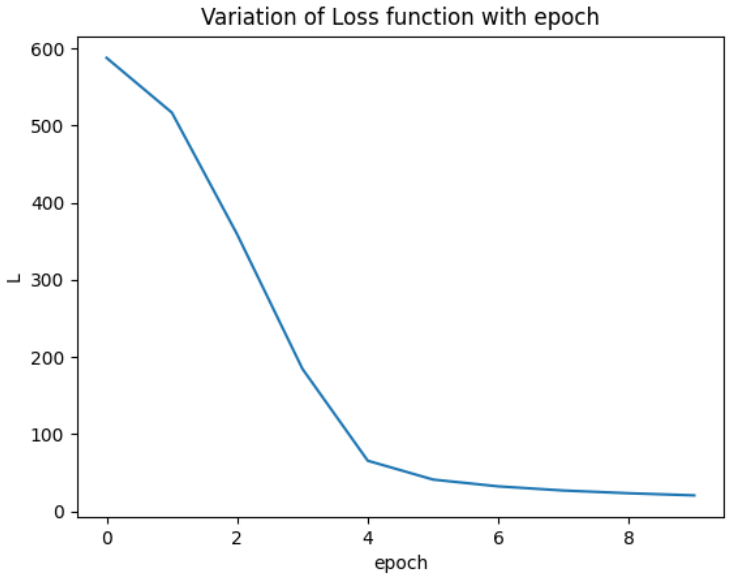
(2) 训练集和测试集上的分类精度

$mnist\_trainaccuracy$: 0.9371453901554676
$mnist\_testaccuracy$: 0.94228515625

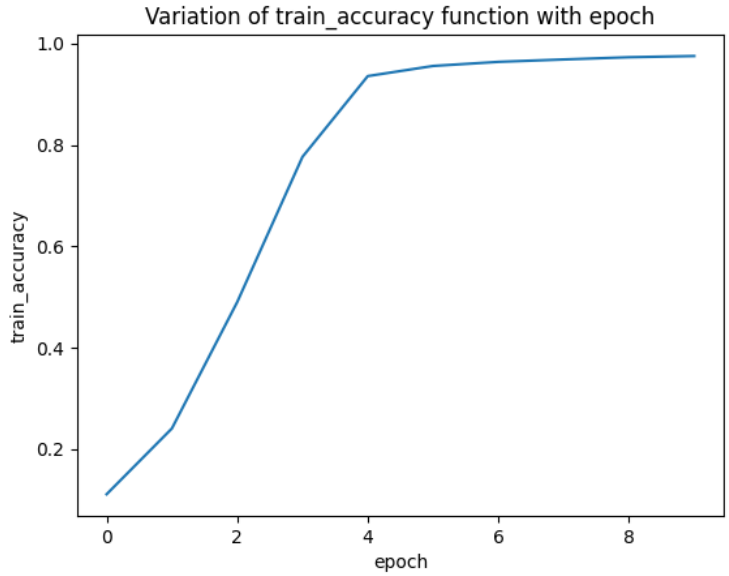
(3) 训练时的损失函数、训练集上的分类精度和测试集上的分类精度随epoch增加的变化曲线

训练时的batch size为256，一共训练10遍epoch

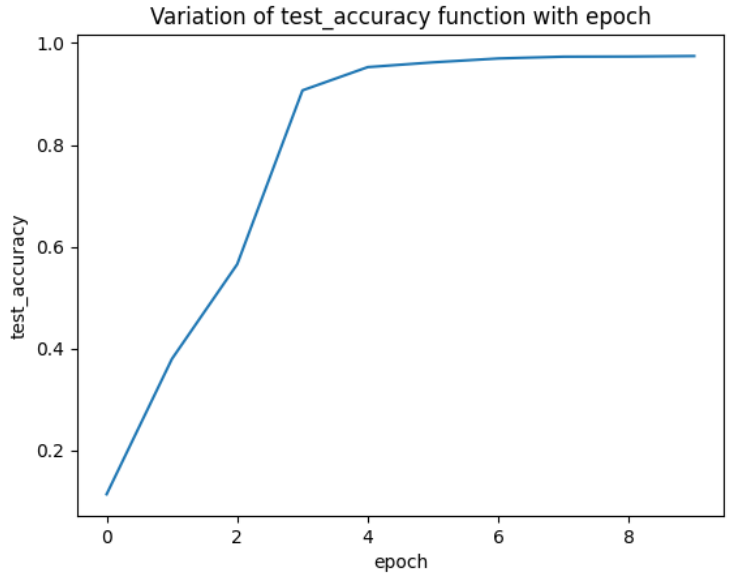
损失函数随迭代次数的变化如图：



训练集分类精度随迭代次数的变化如图：



测试集分类精度随迭代次数的变化如图：



(4) 测试集上随机抽取10个样本，观察分类结果

```
测试集标签:tensor([7., 2., 1., 0., 4., 1., 4., 9., 5., 9.])
预测标签  :tensor([7., 2., 1., 0., 4., 1., 4., 9., 5., 9.])
```