机器学习课程　第五次作业

黄昊　　20204205

选择题目5.1　5.2　5.3　5.4　5.5

5.1　使用线性函数作为激活函数，意味着无论增加多少隐藏层，该神经网络都等效于单层神经网络。

5.2　如果某个神经网络的激活函数为图5.2(b)的函数，且只有一层，该层神经元只有一个，且阈值为0，那么该神经网络就等价于对率回归。

5.3　根据链式法则，显然有：

$$
\begin{aligned}
\frac{\partial E_k}{\partial v_{ih}} &= \frac{\partial E_k}{\partial b_h}\frac{\partial b_h}{\partial \alpha_h}\frac{\partial \alpha_h}{\partial v_{ih}} \\
&= \frac{\partial E_k}{\partial b_h} b_h(1-b_h)x_i \\
&= \sum_{i=1}^{l}\frac{\partial E_k}{\partial \hat{y}_i^k}\frac{\partial \hat{y}_i^k}{\partial \beta_i}\frac{\partial \beta_i}{\partial b_h} b_h(1-b_h)x_i \\
&= \sum_{i=1}^{l} g_i w_{hi} b_h(1-b_h)x_i \\
&= -e_h x_i
\end{aligned}
$$

那么有：

$$
\begin{aligned}
\Delta v_{ih} &= -\eta\frac{\partial E_k}{\partial v_{ih}} \\
&= \eta e_h x_i
\end{aligned}
$$

从而成功推导出来。

5.4　首先显然学习率要大于0；因为如果小于0，直接向正梯度方向更新，结果发散；等于0则无法更新。只有大于0，才能得到更新。大于0的情况下，如果学习率取值较小，则收敛较慢，可能会陷入局部最优解；学习率取值较大时，收敛速度可能加快，但也可能直接导致结果发散。

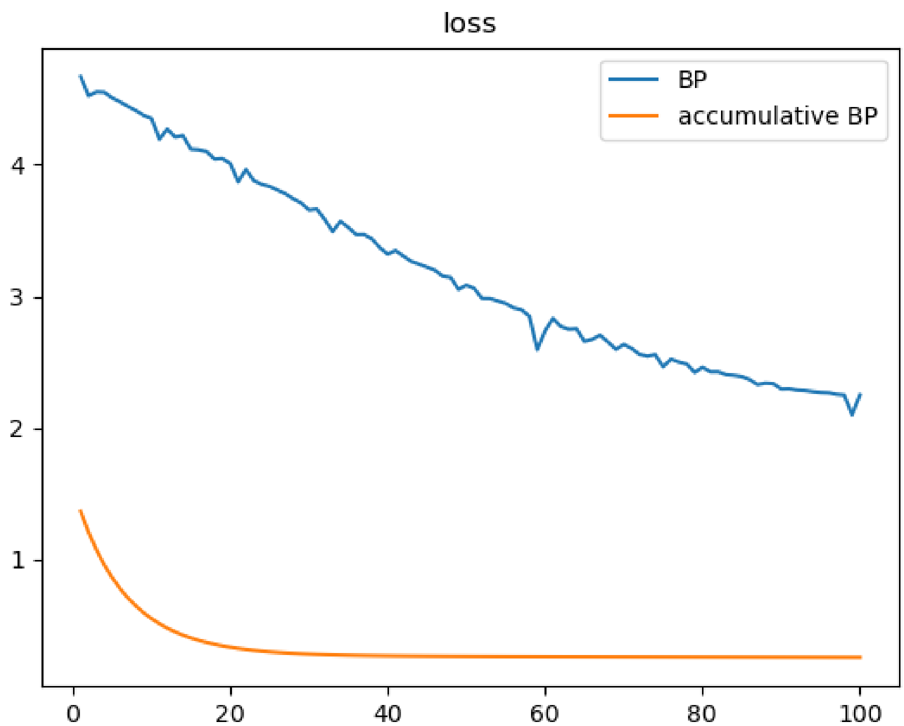
5.5　代码如下：

```
import torch
from torch.utils.data import DataLoader, TensorDataset
from torch import nn, no_grad
from torch import optim
from torch.nn import modules
import matplotlib.pyplot as plt




feature=[["青绿","蜷缩","浊响","清晰","凹陷","硬滑",0.697,0.46],
["乌黑","蜷缩","沉闷","清晰","凹陷","硬滑",0.774,0.376],
["乌黑","蜷缩","浊响","清晰","凹陷","硬滑",0.634,0.264],
["青绿","蜷缩","沉闷","清晰","凹陷","硬滑",0.608,0.318],
["浅白","蜷缩","浊响","清晰","凹陷","硬滑",0.556,0.215],
["青绿","稍蜷","浊响","清晰","稍凹","软粘",0.403,0.237],
["乌黑","稍蜷","浊响","稍糊","稍凹","软粘",0.481,0.149],
["乌黑","稍蜷","浊响","清晰","稍凹","硬滑",0.437,0.211],
["乌黑","稍蜷","沉闷","稍糊","稍凹","硬滑",0.666,0.091],
["青绿","硬挺","清脆","清晰","平坦","软粘",0.243,0.267],
```

```
["浅白","硬挺","清脆","模糊","平坦","硬滑",0.245,0.057],
["浅白","蜷缩","浊响","模糊","平坦","软粘",0.343,0.099],
["青绿","稍蜷","浊响","稍糊","凹陷","硬滑",0.639,0.161],
["浅白","稍蜷","沉闷","稍糊","凹陷","硬滑",0.657,0.198],
["乌黑","稍蜷","浊响","清晰","稍凹","软粘",0.36,0.37],
["浅白","蜷缩","浊响","模糊","平坦","硬滑",0.593,0.042],
["青绿","蜷缩","沉闷","稍糊","稍凹","硬滑",0.719,0.103]]
label=[1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0]
feature_title=["色泽","根蒂","敲声","纹理","脐部","触感","密度","含糖率"]
TEST_NUM␣=␣len(feature)
def autoTagging(feature):
    sample = feature[0]
    lst = []
    for idx, i in enumerate(sample):
        if (type(i) != type("str")):
            continue
        mapping = {}
        cnt = 0
        for ele in feature:
            attr = ele[idx]
            if mapping.get(attr) is None:
                cnt += 1
                mapping[attr] = cnt
        for j in range(len(feature)):
            feature[j][idx] = mapping[feature[j][idx]]
        lst.append(mapping)
    return feature, lst

feature, mapLst = autoTagging(feature)
feature = torch.tensor(feature,dtype=torch.double).reshape((17,8))
label = torch.tensor(label,dtype=torch.double).reshape((17,1))
dataSet = TensorDataset(feature,label)
data_iter_All = DataLoader(
    dataSet,
    batch_size=TEST_NUM,
    shuffle=True,
)

data_iter_1 = DataLoader(
    dataSet,
    batch_size=1,
    shuffle=True,
)

model = nn.Sequential(
    nn.Linear(8, 10),
    nn.Sigmoid(),
    nn.Linear(10, 1)
).double()

loss = modules.MSELoss()
optimizer = optim.SGD(model.parameters(), lr = 0.01)

accPlt_All = []
lossPlt_All = []
```

```
NUM_EPOCH = 100
for epoch in range(1, NUM_EPOCH+1):
    totLoss=0
    for X,Y in data_iter_All:
        model.train()
        output=model(X)
        l=loss(output,Y)
        totLoss+=l
        l.backward()
        optimizer.step()
        optimizer.zero_grad()
    with torch.no_grad():
        model.eval()
        acc=0
        cnt=0
        for x,y in data_iter_1:
            y_hat=model(x)
            if (torch.argmax(y_hat)==torch.argmax(y)):
                acc+=1
            cnt+=1
        accPlt_All.append(acc/TEST_NUM*100)
    lossPlt_All.append(float(totLoss))

accPlt_1 = []
lossPlt_1 = []
NUM_EPOCH = 100
for epoch in range(1, NUM_EPOCH+1):
    totLoss=0
    for X,Y in data_iter_1:
        model.train()
        output=model(X)
        l=loss(output,Y)
        totLoss+=l
        l.backward()
        optimizer.step()
        optimizer.zero_grad()
    with torch.no_grad():
        model.eval()
        acc=0
        cnt=0
        for x,y in data_iter_1:
            y_hat=model(x)
            if (torch.argmax(y_hat)==torch.argmax(y)):
                acc+=1
            cnt+=1
        accPlt_1.append(acc/TEST_NUM*100)
    lossPlt_1.append(float(totLoss))

x = [i for i in range(1,NUM_EPOCH+1)]
plt.plot(x, lossPlt_1)
plt.plot(x, lossPlt_All)
plt.legend(["BP",␣"accumulative BP"])
plt.title("loss")
plt.show()
```

损失曲线如下所示：



从图中可以看出，累积BP算法比标准BP算法收敛更快。