MLP and BP using torch

iiGray

[1]:
```
import torch
from torch import nn
from torch.utils import data
import pickle
import matplotlib.pyplot as plt
```

[2]:
```
''' 定义数据集'''
class Dataset(data.Dataset):
    def __init__(self,data):
        self.data=data if type(data)==list else pickle.load(open(data,"rb"))
    def __getitem__(self,i):
        return \
    torch.FloatTensor(list(self.data[i][0])),torch.FloatTensor([self.data[i][1]])
    def __len__(self):
        return len(self.data)
```

[3]:
```
''' 定义本问题的网络'''
class Net(nn.Module):
    def __init__(self,in_dim):
        super().__init__()

        self.linear1=nn.Linear(in_dim,5)
        self.tanh1=nn.Tanh()

        self.linear2=nn.Linear(5,3)
        self.tanh2=nn.Tanh()

        self.linear3=nn.Linear(3,1)

    def forward(self,x):
        out=self.linear1(x)
        out=self.tanh1(out)

        out=self.linear2(out)
        out=self.tanh2(out)

        out=self.linear3(out)

        return out
```

[4]:
```
''' 定义绘制函数, 训练函数, 预测函数'''
def draw(data_pth):
```

```
    dots=pickle.load(open(data_pth,"rb"))
    dots0=[[dot[0][0],dot[0][1]] for dot in dots if dot[-1]==0]
    dots0x=[k[0] for k in dots0]
    dots0y=[k[1] for k in dots0]
    dots1=[[dot[0][0],dot[0][1]] for dot in dots if dot[-1]==1]
    dots1x=[k[0] for k in dots1]
    dots1y=[k[1] for k in dots1]
    plt.scatter(dots0x,dots0y,c="g")
    plt.scatter(dots1x,dots1y,c="b")


def train(net,dataloader,epochs,lr,eps=1e-5):

#     optimizer=torch.optim.SGD(net.parameters(),lr=lr)
    optimizer=torch.optim.AdamW(net.parameters(),lr=lr)
    l=nn.BCEWithLogitsLoss()

    loss_lst=[100]
    for _ in range(epochs):
        for x,y in dataloader:

            y_pre=net(x)

            loss=l(y_pre,y)

            loss.backward()

            with torch.no_grad():
                ls=loss.item()

            loss_lst+=[ls]

            optimizer.step()

            optimizer.zero_grad()

        if abs(loss_lst[-1]-loss_lst[-2])<eps:
            break

    print("Update times:",len(loss_lst))
    print("Final_Loss:",loss_lst[-1])

    plt.xlabel("update_times")
    plt.ylabel("loss")
    plt.plot(loss_lst)
    plt.show()

def predict(net,test_dataset,trn_path=False,eps=0.001):
    TP,TN,FP,FN=0,0,0,0
    ALL=len(test_dataset)
    datas=[[[],[]],[[],[]]]
```

```
    for i in range(ALL):
        dot,l=test_dataset[i]
        dot=dot[None,:]
        l=l[None,:]
        out=net(dot)[0][0]
        pre=0 if out <0 else 1

        datas[pre][0]+=[dot[0][0]]
        datas[pre][1]+=[dot[0][1]]

        if l==1:
            if out<0:FP+=1
            else:TP+=1
        else:
            if out<0:TN+=1
            else: FN+=1
    if trn_path:
        draw(trn_path)

    ''' 绘出预测情况，黄色为预测负类，红色为预测正类'''
    plt.scatter(datas[0][0],datas[0][1],c="y")
    plt.scatter(datas[1][0],datas[1][1],c="r")
    plt.show()

    print("Precision:\t",(TP+eps)/(TP+FP+eps))

    print("Recall:\t",(TP+eps)/(TP+FN+eps))

    print("Accuracy:\t",(TP+TN+eps)/(TP+TN+FP+FN+eps))
```

[5]:
```
''' 载入训练集和测试集'''
trn_dataset=Dataset("trn_datas.pkl")
tst_dataset=Dataset("tst_datas.pkl")

''' 设置随机数种子以便复现'''
torch.manual_seed(0)
```

[5]:
```
<torch._C.Generator at 0x2250087f4f0>
```

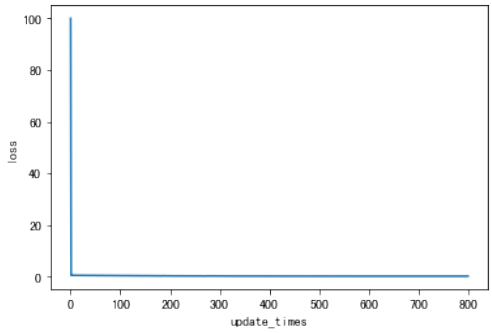
[6]:
```
net=Net(in_dim=2)
dataloader=data.DataLoader(trn_dataset,batch_size=50)
```

[7]:
```
net
```

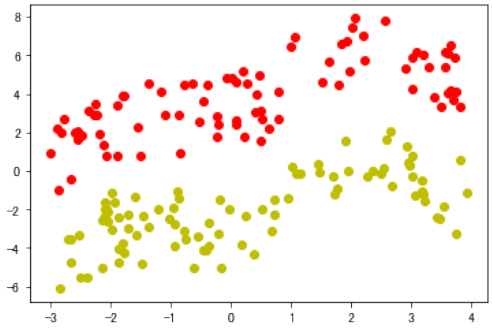
[7]:
```
Net(
  (linear1): Linear(in_features=2, out_features=5, bias=True)
  (tanh1): Tanh()
  (linear2): Linear(in_features=5, out_features=3, bias=True)
  (tanh2): Tanh()
  (linear3): Linear(in_features=3, out_features=1, bias=True)
)
```

[8]:
```
train(net,dataloader,epochs=200,lr=0.001)
```

```
Update times: 801
Final_Loss: 0.28874489665031433
```

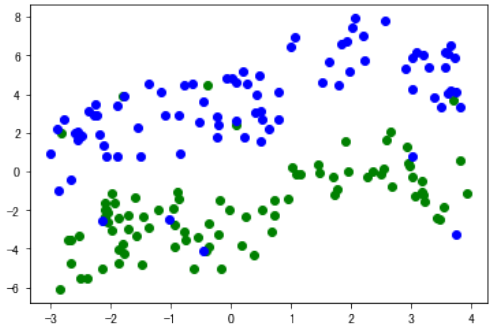


[9]:
```
predict(net,trn_dataset)
```

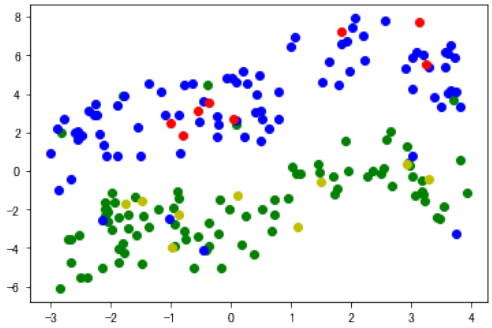


```
Precision:      0.9404768990845347
Recall:  0.9404768990845347
Accuracy:       0.9428574693858892
```

[10]:
```
draw("trn_datas.pkl")
```



[11]:
```
predict(net,tst_dataset,"trn_datas.pkl")
```




```
Precision:      1.0
Recall:  1.0
Accuracy:       1.0
```