```
# 导入必要的库
import pandas as pd # 数据处理和分析库
import numpy as np # 科学计算库
from sklearn import preprocessing # 数据预处理库
import torch # PyTorch 深度学习框架
from torch import optim # 优化器模块
import torch.nn.functional as F # 神经网络函数模块
from net import housing_NN # 自定义的神经网络模型
import matplotlib.pyplot as plt # 可视化库

# 加载和预处理数据
def load_data():
    feature=pd.read_excel("BostonHousingData.xlsx", sheet_name="Sheet1", header=0) # 从 Excel 文件中读取数据

    #取出标签，同时在读入的数据中删除标签
    label=feature["MEDV"]
    label=np.array(label)

    feature=feature.drop("MEDV",axis=1)
    data=np.array(feature)

    #对输入数据做归一化
    data=preprocessing.StandardScaler().fit_transform(data)


    #print(data)

    #3:1 划分测试集和训练集
```

```
    train_data=[]
    train_label=[]
    test_data=[]
    test_label=[]
    for i in range(len(data)):
        if(i%3==0):
            test_data.append(data[i])
            test_label.append(label[i])
        else:
            train_data.append(data[i])
            train_label.append(label[i])

    #转为 tensor 向量
    train_data=torch.tensor(train_data,dtype=float,requires_grad=True).to(torch.float32)
    train_label=torch.tensor(train_label,dtype=float,requires_grad=True).to(torch.float32)
    test_data=torch.tensor(test_data,dtype=float,requires_grad=True).to(torch.float32)
    test_label=torch.tensor(test_label,dtype=float,requires_grad=True).to(torch.float32)

    return train_data, train_label, test_data, test_label

#训练
def train(epoches,model,loss_func,opt,batch_size,data,label):
    print("start training...")
    for epoch in range(epoches): # 迭代次数
        for start in range(0,len(data),batch_size):
            if start+batch_size<=len(data):
                end=start+batch_size
            else:
                end=len(data)
            x=data[start:end] # 当前批次的输入数据
```

```
            y=label[start:end] # 当前批次的标签
            model.train() # 设置模型为训练模式
            pre=model(x) # 前向传播
            loss=loss_func(pre,y) # 计算差值

            # 反向传播和优化
            opt.zero_grad() # 清空梯度
            loss.backward() # 反向传播计算梯度
            opt.step() # 更新模型参数
        if epoch%500==0: # 每 500 个 epoch 打印一次 loss
            print(f"epoch:{epoch},loss:{loss}")
            # 如果 loss 小于预设值，保存模型并停止训练
        if loss<0.1:
            print(f"epoch:{epoch},loss:{loss}")
            print("已达预设")
            torch.save(model.state_dict(), "best_model.pth")
            print("model saved")
            break
    # 保存最终模型
    torch.save(model.state_dict(), "last_model.pth")
    print("model saved")

# 测试模型
def test(my_model, test_data, test_label, loss_func):
    pre=[] # 预测结果
    act=[] # 实际结果
    mean_mse = 0 # 平均均方误差
    print("start testing...")
```

```
    # 加载最优模型
    my_model.load_state_dict(torch.load("best_model.pth"))
    my_model = my_model.eval()

    # 遍历测试数据
    for i in range(len(test_data)):
        p=my_model(test_data[i])
        pre.append(p.item())
        act.append(test_label[i].item())
         # 计算 loss
        loss = loss_func(p, test_label[i])
        mean_mse += loss.item()
    mean_mse /= len(test_data)
    print("mean_mse = " + str(mean_mse))
    plt.figure(1)
    plt.plot(pre,color="r")
    plt.plot(act,color="b")
    plt.savefig("pred.png")


# 主函数
def main(mode):

    my_epoches=100000 # 最大训练迭代次数
    my_model=housing_NN() # 初始化模型
    my_loss_func = F.mse_loss # 使用均方误差作为损失函数
    my_opt=optim.Adam(my_model.parameters(),lr=0.001)
    my_batch_size=64
```

```
    train_data, train_label, test_data, test_label = load_data()

    if mode == "train":
        train(my_epoches,my_model,my_loss_func,my_opt,my_batch_size,train_data,train_label)

    if mode == "test":
        test(my_model, test_data, test_label, my_loss_func)

if __name__ == "__main__":
    #main("train") # 训练模式
    main("test") # 测试模式
```