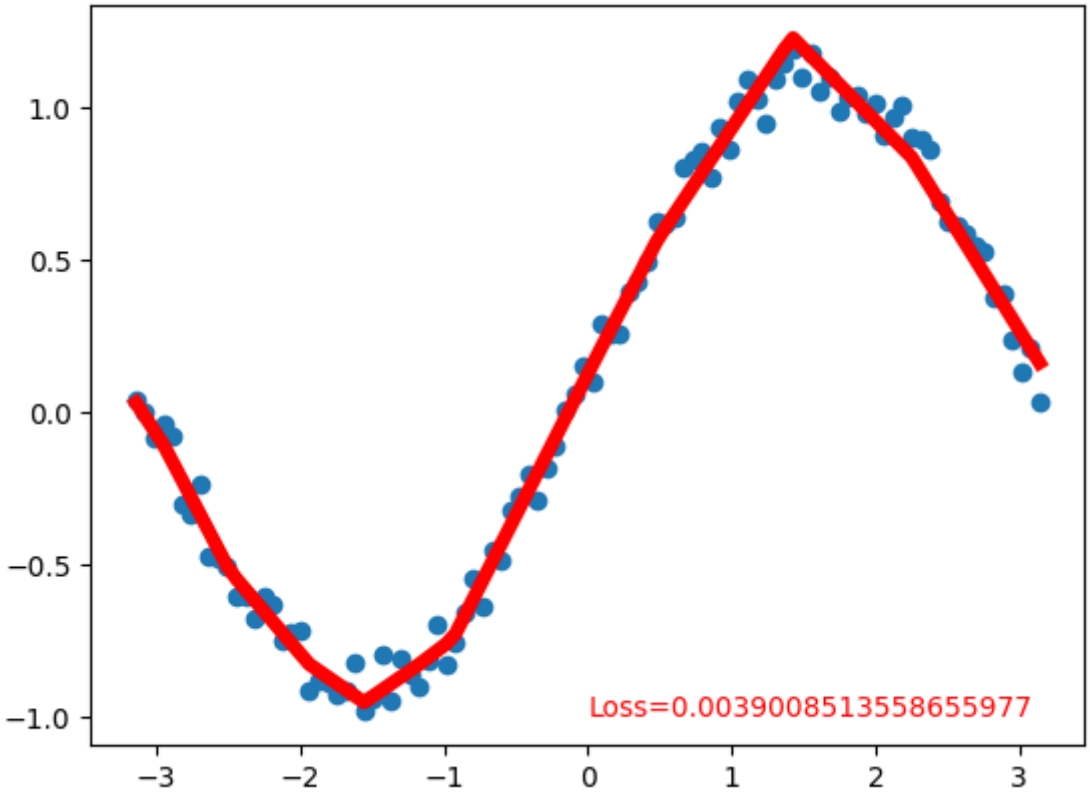
二、数据回归实验



姓名：游霄童

学号：21009200158

created：2024/03/28

实验目的

本实验旨在利用神经网络完成数据回归任务，并探究不同网络结构和参数对回归效果的影响。

实验数据

实验采用具有一定随机性的散点分布数据，由以下公式生成：

```
x = torch.unsqueeze(torch.linspace(- np.pi, np.pi, 100), dim=1)
y = torch.sin(x) + 0.5 * torch.rand(x.size())
```

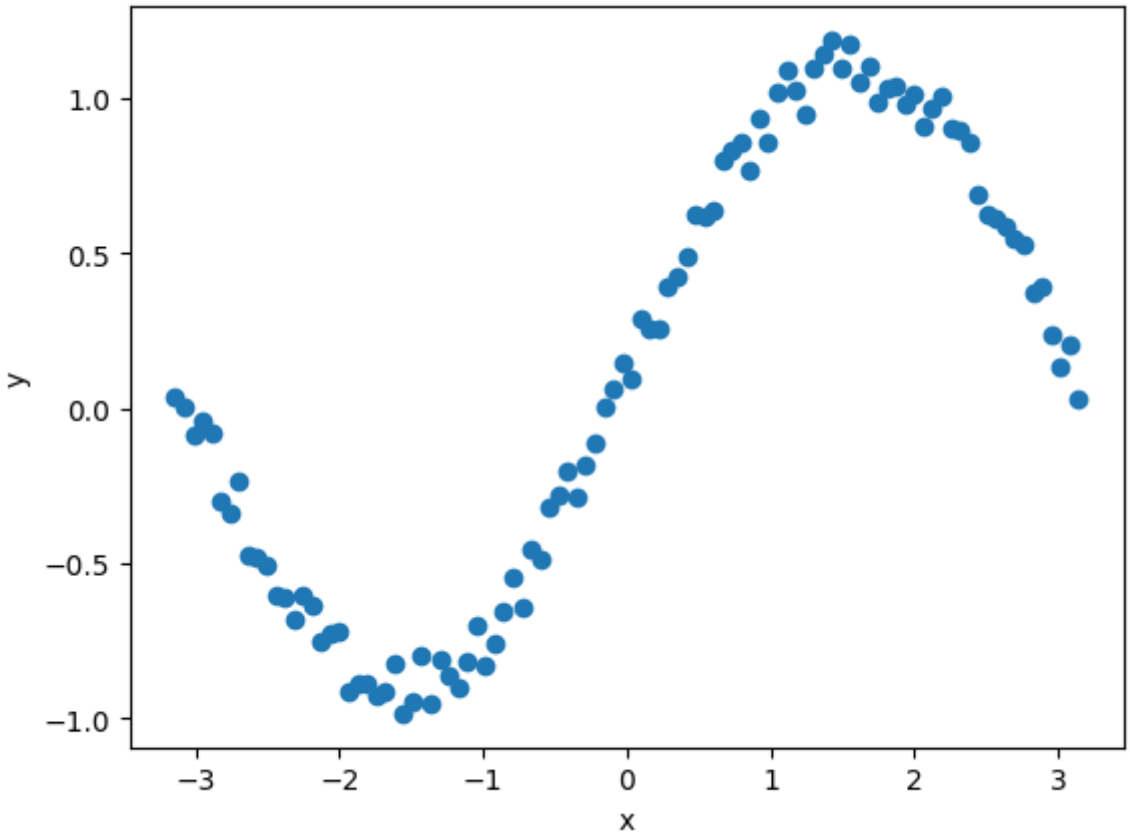
其中，x 为等差数列，y 在 sin(x) 函数周围添加了随机噪声。

实验环境

- 硬件平台：Nvidia Jetson TX1 嵌入式系统
- 软件平台：Python 3.x，PyTorch 1.x

实验步骤

1. 导入必要的模块:
    - numpy：用于数据处理
    - torch：用于构建神经网络
    - matplotlib：用于可视化结果
2. 构造样本集:



    - 生成 x 和 y 数据，并将 x 转换为二维数组以便输入网络。

```
#自行构建散点集
x = torch.unsqueeze(torch.linspace(-np.pi,np.pi,100),dim=1)
y = torch.sin(x) + 0.2*torch.rand(x.size())
plt.plot(x.numpy(),y.numpy(),'o')
plt.ylabel('y')
plt.xlabel('x')
plt.show()
```

3. 构建网络:
    - 定义一个神经网络类 Net，包含网络结构和前向传播过程。
    - 实验中，我们以隐藏节点数为 10 的单隐层网络为例，并使用 ReLU 激活函数。

- ```
  #用类的方式构建网络
  class Net(nn.Module):
      def __init__(self):
          super(Net,self).__init__()
          self.predict=nn.Sequential(
              nn.Linear(1,10),
              nn.ReLU(),
              nn.Linear(10,1)
          )
      def forward(self,x):
          predict=self.predict(x)
          return predict
  ```

4. 训练网络:
   - 实例化网络对象 net。
   - 设置优化器为 SGD，学习率为 0.05。
   - 设置损失函数为 MSE。
   - 使用循环进行训练，并在每次迭代中:
     - 将 x 输入网络得到预测值 out。
     - 计算 out 与真实值 y 之间的损失。
     - 清除梯度。
     - 反向传播误差。
     - 更新网络参数。

```
#实例化网络，定义优化器和损失函数
net=Net()
optimizer=torch.optim.Adam(net.parameters(),lr=0.01)
loss_func=nn.MSELoss()
```

5. 动态显示:
   - 使用 plt.ion() 打开交互模式。
   - 每隔一定次数训练，绘制当前训练结果，包括样本散点图和拟合曲线。
   - 使用 plt.ioff() 关闭交互模式，并使用 plt.show() 显示最终结果。

```
# 训练模型
# 打开交互模式
plt.ion()

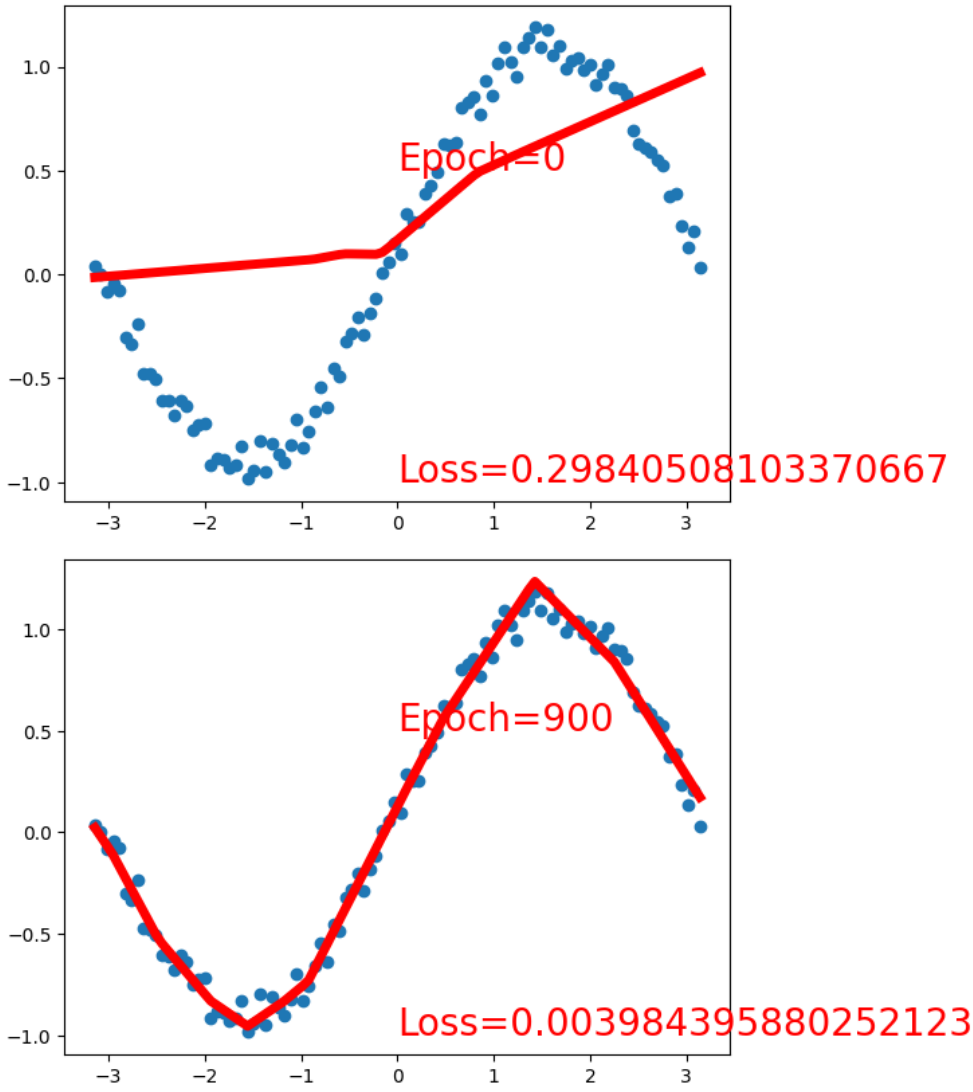
for epoch in range(1000):
    predict = net(x)
    loss = loss_func(predict, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```
```

```
    #每隔100次迭代绘制一次图像
    if epoch % 100 == 0:
        # 清除原有图像
        plt.cla()
        # 绘制散点图
        plt.scatter(x.detach().numpy(), y.detach().numpy())
        # 绘制预测曲线
        plt.plot(x.detach().numpy(), predict.detach().numpy(), 'r-',
lw=5)
        # 设置坐标轴，写清楚代数和损失函数值
        plt.text(0, -1, f'Loss={loss.item()}', fontdict={'size': 20,
'color': 'red'})
        plt.text(0, 0.5, f'Epoch={epoch}', fontdict={'size': 20,
'color': 'red'})
        plt.pause(0.1)

# 关闭交互模式
plt.ioff()
plt.show()
```

6. **测试网络**:
    - 使用测试集数据评估网络的预测效果。
    - 计算并输出测试集上的损失值和预测精度等指标。

实验结果



- 经过训练，网络能够较好地拟合目标函数，并能较准确地预测测试集数据。
- 调整网络结构和参数，例如增加隐层数、节点数，或改变激活函数，可以进一步提升网络的预测性能。

```
#保存模型
torch.save(net,'net_2.pkl')
#%%
#加载模型
net=torch.load('net_2.pkl')
#%%
#给出一个新的x值
x=torch.unsqueeze(torch.linspace(-np.pi,np.pi,100),dim=1)
#%%
#测试模型
predict=net(x)
plt.plot(x.numpy(),y.numpy(),'o')
```

```
plt.plot(x.numpy(),predict.data.numpy(),'r-',lw=5)
#损失值画在图像上
plt.text(0,-1,f'Loss={loss.item()}',fontdict={'size':10,'color':'red'})
plt.show()
```

结论

本实验表明，神经网络可以有效地用于数据回归问题。通过合理设计网络结构和参数，可以获得较高的预测精度。

进一步研究

- 尝试更复杂的网络结构，例如多层网络、卷积神经网络等。
- 尝试不同的优化算法和学习率策略。
- 将神经网络应用于其他回归问题，例如时间序列预测、图像回归等。

代码

```
# 导入必要的模块
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt

# 构造样本集
x = torch.unsqueeze(torch.linspace(- np.pi, np.pi, 100), dim=1)
y = torch.sin(x) + 0.5 * torch.rand(x.size())

# 定义网络结构
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.predict = nn.Sequential(
            nn.Linear(1, 10),
            nn.ReLU(),
            nn.Linear(10, 1)
        )

    def forward(self, x):
        prediction = self.predict(x)
        return prediction

# 训练网络
net = Net()
optimizer = torch.optim.SGD(net.parameters(), lr=0.05)
loss_func = nn.MSELoss()

for epoch in range(1000):
```

```
    out = net(x)
    loss = loss_func(out, y)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    # 动态显示训练结果
    if epoch % 100 == 0:
        plt.cla()
        plt.scatter(x.data.numpy(), y.data.numpy())
        plt.plot(x.data.numpy(), out.data.numpy(), 'r-', lw=5)
        plt.text(0.5, 0, 'Loss=%.4f' % loss.data.numpy(), fontdict=
{'size': 20, 'color': 'red'})
        plt.pause(0.1)

plt.ioff()
plt.show()
```