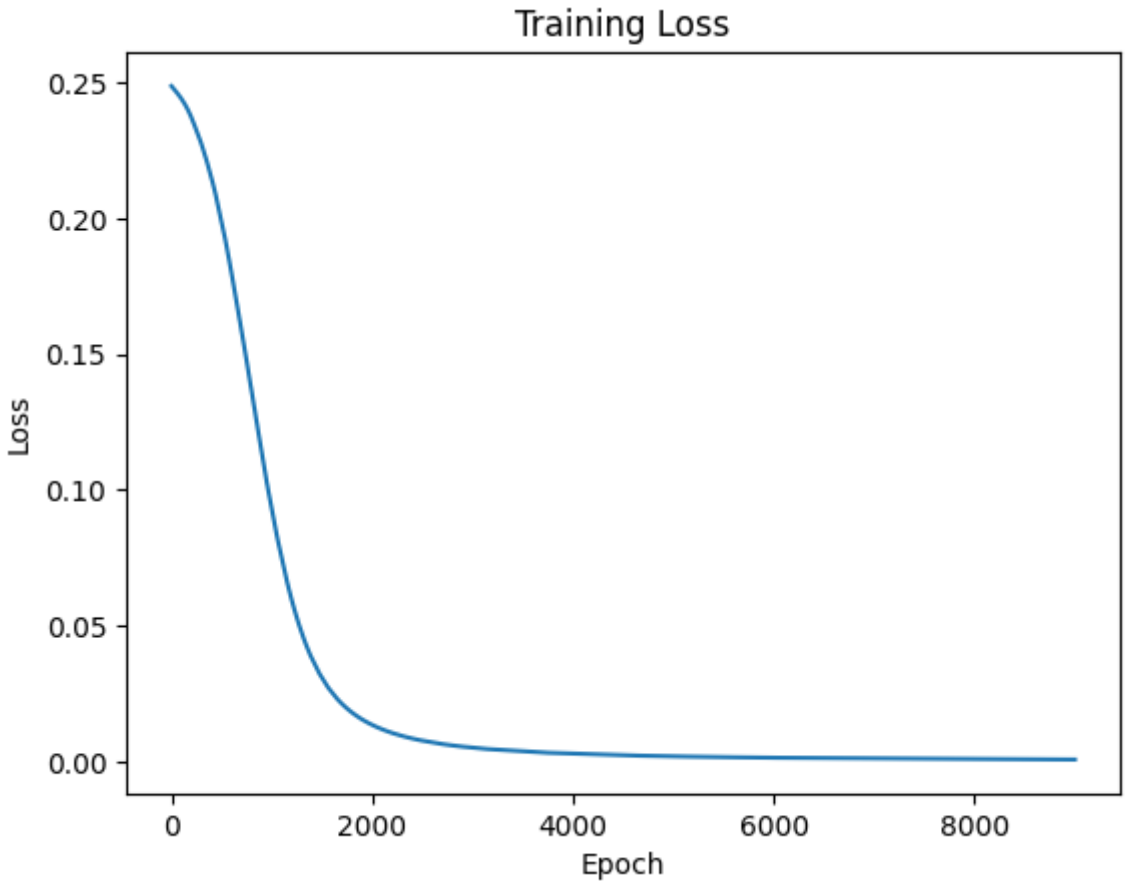
一、异或运算



姓名：游霄童

学号：21009200158

created：2024/03/28

实验目的

设计一个神经网络实现异或运算。

实验数据

异或运算数据。

X	Y
[0,0]	0
[0,1]	1

X	Y
[1,0]	1
[1,1]	0

```
#构建数据集
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,1,1,0]
#x,y转换为tensor
x_tensor=torch.tensor(x,dtype=torch.float32).cuda()
y_tensor=torch.tensor(y,dtype=torch.float32).cuda()
```

实验环境

- NVIDIA GeForce RTX 3060 Laptop GPU

实现描述

- 采用含有一个隐含层的神经网络，隐含层节点的个数为 20。
- 网络隐含层的激活函数采用 ReLU 函数，输出层激活函数采用 Sigmoid 函数。
- 实现过程分为训练部分和测试部分。
    - 训练部分完成模型的训练。
    - 测试部分编写程序加载训练好的模型，给出相应的输入并获得输出，观察结果是否是所期望的结果。

网络模型

- 使用 PyTorch 框架构建网络。
- 输入层有 2 个节点，隐含层有 20 个节点，输出层有 1 个节点。
- 使用 ReLU 激活函数和 Sigmoid 激活函数。
- 优化器采用 SGD，损失函数采用均方差损失函数。

```
#设计一个神经网络的异或实验
net=nn.Sequential(
    nn.Linear(2,20),
    nn.ReLU(),
    nn.Linear(20,1),
    nn.Sigmoid()
).cuda()
#%%
optimizer=optim.SGD(net.parameters(),lr=0.05)
loss_func=nn.MSELoss()
```

训练程序

- 创建输入和期望输出，转换成 Tensor 类型变量。
- 计划训练 5000 次，使用 GPU 加速。
- 设置优化器和损失函数。
- 进行网络训练，每 1000 次训练输出当前迭代次数及损失函数值。

```
from IPython.display import clear_output
losses = []  # 用于存储损失值

for epoch in range(10000):
    out=net(x_tensor)
    loss=loss_func(out,y_tensor.view(-1,1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    losses.append(loss.item())  # 添加当前损失值到列表
    if epoch % 1000 == 0:
        clear_output(wait=True)  # 清除之前的图像
        plt.plot(losses)  # 绘制损失值曲线
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title('Training Loss')
        plt.show()
```

测试程序

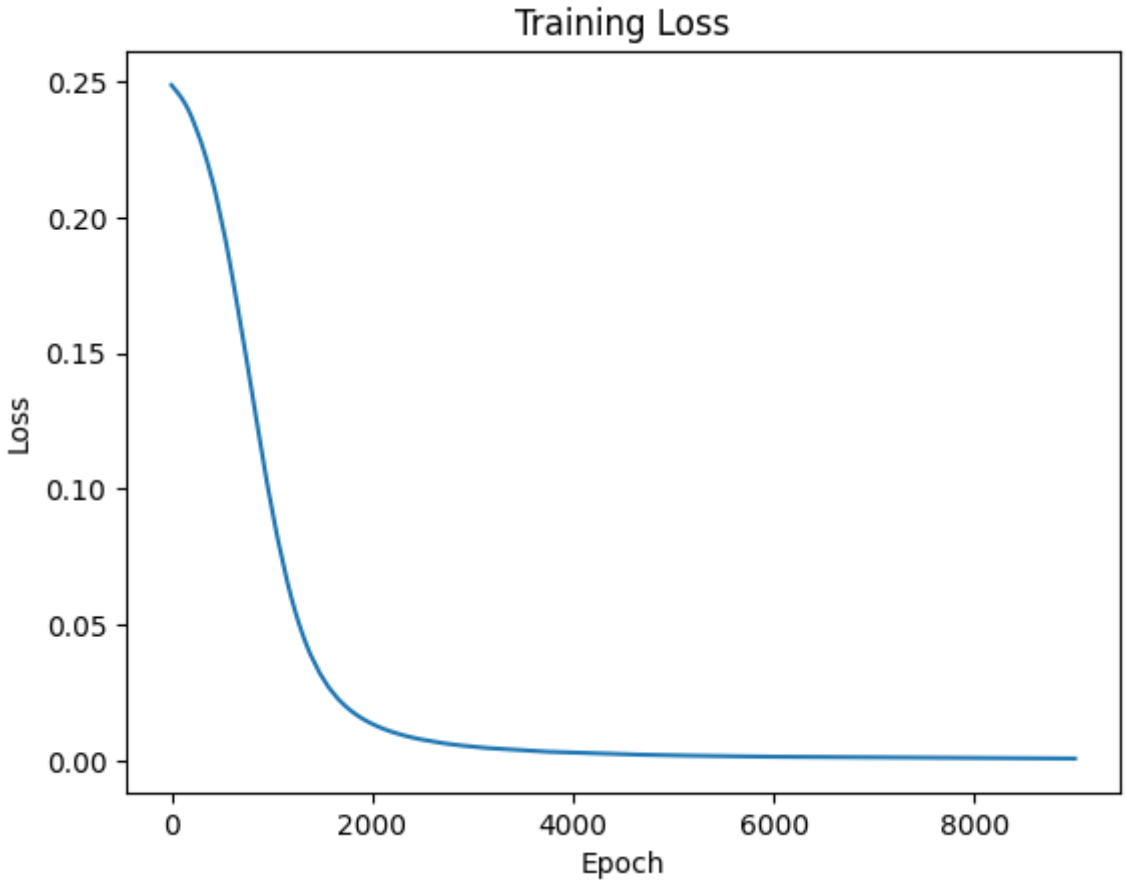
- 设置输入，转换成 Tensor 变量并传入 GPU。
- 加载训练好的网络模型进行测试。
- 根据输出结果判断异或运算是否成功。

```
out=net(x_tensor).cpu()
print(f'out:{out.detach().numpy()}')
#%%
#保存训练好的网络
torch.save(net.state_dict(),'./net_1.pth')
#%%
#测试训练好的网络
net2=nn.Sequential(
    nn.Linear(2,20),
    nn.ReLU(),
    nn.Linear(20,1),
    nn.Sigmoid()
).cuda()
net2.load_state_dict(torch.load('./net_1.pth'))
x_tensor=torch.tensor([0,0],dtype=torch.float32).cuda()
out=net2(x_tensor).cpu()
print(f'out:{out.detach().numpy()}')
```

```
#%%
out
#%%
#调整输出，若out大于0.5为1，小于0.5为0
out[out>0.5]=1
out[out<=0.5]=0
print(f'out:{out.detach().numpy()}')
```

结果

损失如下图所示：



测试结果如下所示：

```
out:[[0.02495466]
 [0.9728757 ]
 [0.97043836]
 [0.02624389]]
```

```
out:[[0.]
 [1.]
 [1.]
 [0.]]
```

实验心得

作为一名大三智能科学与技术的本科生，在学习过程中，我深切感受到了人工智能领域的魅力和挑战。通过这次设计并实现异或运算的神经网络实验，我进一步加深了对神经网络结构和训练过程的理解，同时也锻炼了我的编程能力和问题解决能力。

在实验中，我学习了如何使用 PyTorch 框架构建神经网络，包括定义网络结构、选择合适的激活函数、设置优化器和损失函数等。通过实际编写训练程序和测试程序，我更加熟悉了神经网络的训练流程，包括数据准备、前向传播、损失计算、反向传播和参数更新等步骤。这个过程让我体会到了理论与实践相结合的重要性。

此外，这次实验也让我认识到了实验环境的重要性。使用NVIDIA GeForce RTX 3060 Laptop GPU嵌入式系统作为实验平台，不仅提供了强大的计算能力，还使我对嵌入式系统在智能科学与技术领域的应用有了更深入的了解。

总的来说，这次实验不仅增强了我的实践能力，也激发了我对智能科学与技术领域的热情。我期待在未来的学习和研究中，能够继续探索更多人工智能的知识和技术，为我的专业领域做出贡献。

附录

全部代码如下：

```
#%%
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from keras.models import Sequential
import matplotlib.pyplot as plt
#%%
#构建数据集
x=[[0,0],[0,1],[1,0],[1,1]]
y=[0,1,1,0]
#x,y转换为tensor
x_tensor=torch.tensor(x,dtype=torch.float32).cuda()
y_tensor=torch.tensor(y,dtype=torch.float32).cuda()
#%%
#设计一个神经网络的异或实验
net=nn.Sequential(
    nn.Linear(2,20),
    nn.ReLU(),
    nn.Linear(20,1),
    nn.Sigmoid()
).cuda()
#%%
optimizer=optim.SGD(net.parameters(),lr=0.05)
```

```
loss_func=nn.MSELoss()
#%%
from IPython.display import clear_output
losses = []  # 用于存储损失值

for epoch in range(10000):
    out=net(x_tensor)
    loss=loss_func(out,y_tensor.view(-1,1))
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()

    losses.append(loss.item())  # 添加当前损失值到列表
    if epoch % 1000 == 0:
        clear_output(wait=True)  # 清除之前的图像
        plt.plot(losses)  # 绘制损失值曲线
        plt.xlabel('Epoch')
        plt.ylabel('Loss')
        plt.title('Training Loss')
        plt.show()
#%%
out=net(x_tensor).cpu()
print(f'out:{out.detach().numpy()}')
#%%
#保存训练好的网络
torch.save(net.state_dict(),'./net_1.pth')
#%%
#测试训练好的网络
net2=nn.Sequential(
    nn.Linear(2,20),
    nn.ReLU(),
    nn.Linear(20,1),
    nn.Sigmoid()
).cuda()
net2.load_state_dict(torch.load('./net_1.pth'))
x_tensor=torch.tensor(x,dtype=torch.float32).cuda()
out=net2(x_tensor).cpu()
print(f'out:{out.detach().numpy()}')
#%%
out
#%%
#调整输出，若out大于0.5为1，小于0.5为0
out[out>0.5]=1
out[out<=0.5]=0
print(f'out:{out.detach().numpy()}')
```