```
import torch
import torch.nn as nn
import torch.optim as optim


X = torch.tensor([[0, 0],
                  [0, 1],
                  [1, 0],
                  [1, 1]], dtype=torch.float32)

y = torch.tensor([[0],
                  [1],
                  [1],
                  [0]], dtype=torch.float32)

class XORNet(nn.Module):
    def __init__(self):
        super(XORNet, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.sigmoid(self.fc1(x))
        x = self.sigmoid(self.fc2(x))
        return x

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
net = XORNet().to(device)

criterion = nn.BCELoss()
optimizer = optim.SGD(net.parameters(), lr=0.1)

num_epochs = 10000
X, y = X.to(device), y.to(device)

for epoch in range(num_epochs):
    optimizer.zero_grad()
    outputs = net(X)
    loss = criterion(outputs, y)
    loss.backward()
    optimizer.step()

    if (epoch+1) % 1000 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss:
{loss.item():.4f}')

with torch.no_grad():
    predictions = net(X)
```

```
    print("\nPredictions after training:")
    print(predictions)
```

```
Epoch [1000/10000], Loss: 0.6933
Epoch [2000/10000], Loss: 0.6929
Epoch [3000/10000], Loss: 0.6917
Epoch [4000/10000], Loss: 0.6780
Epoch [5000/10000], Loss: 0.5840
Epoch [6000/10000], Loss: 0.4358
Epoch [7000/10000], Loss: 0.1809
Epoch [8000/10000], Loss: 0.0790
Epoch [9000/10000], Loss: 0.0470
Epoch [10000/10000], Loss: 0.0328

Predictions after training:
tensor([[0.0236],
        [0.9710],
        [0.9710],
        [0.0471]], device='cuda:0')
```