机器学习实验报告 4

姓名：陈海弘

学号：**23354049**

陈海弘

2024.11.21

Contents

1 摘要 **3**

2 神经网络中的前向传播和后向传播 **3**

2.1 实验要求 3

2.2 数据读取 3

2.3 神经网络搭建 3

2.4 训练 5

2.5 训练损失测试损失展示 7

3 总结 **8**

1 摘要

本次实验是关于红酒品质的预测，使用的是一个关于红酒品质的数据集，总共有 1599 个样本，每个样本包含 11 个特征以及 1 个标签，每个标签的取值是连续的。本次实验已经按照 8: 2 的比例划分成了训练数据集'wine_train.csv' 以及测试数据集'wine_test.csv'，且每个数据集都已经做了归一化处理。

引用的库有 pandas、torch、matplotlib、numpy 等，其中最重要的是 torch 库，因为本次实验是关于神经网络的搭建和训练的。本次试验用的 pytorch 版本是 2.5.1。

Tensor 和 ndarray 相似，但是 Tensor 可以利用 GPU 加速。

2 神经网络中的前向传播和后向传播

2.1 实验要求

Red Wine Quality 是一个关于红酒品质的数据集，总共有 1599 个样本，每个样本包含 11 个 (都是连续的) 特征以及 1 个标签，每个标签的取值是连续的。本次实验已经按照 8: 2 的比例划分成了训练数据集'wine_train.csv' 以及测试数据集'wine_test.csv'，且每个数据集都已经做了归一化处理。

2.2 数据读取

读入训练数据集'wine_train.csv' 与测试数据集'wine_test.csv'。

```
train_data = pd.read_csv('wine_train.csv')
test_data = pd.read_csv('wine_test.csv')
```

2.3 神经网络搭建

利用线性层和激活函数搭建一个神经网络，要求输入和输出维度与数据集维度一致，而神经网络深度、隐藏层大小、激活函数种类等超参数自行调整。

输入维度为 11，输出维度为 1，所以输入层到隐藏层、隐藏层到隐藏层、隐藏层到输出层的维度分别为 11、128、32。

神经网络深度是 3 层，所以定义一个神经网络类 WineNet，继承自 nn.Module。在构造函数中定义了三个全连接层，分别是输入层到隐藏层、隐藏层到隐藏层、隐藏层到输出层。

激活函数是 ReLU 函数，即 $ReLU(x) = max(0, x)$。还有其他的激活函数，例如 sigmoid 函数，即 $sigmoid(x) = \frac{1}{1+e^{-x}}$，以及 tanh 函数，即 $tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$。

在 forward 函数中定义了神经网络的前向传播过程，即将输入数据经过全连接层和激活函数处理后输出。

```
    class WineNet(nn.Module):
    def __init__(self):
        super (WineNet, self).__init__()
        self.fc1 = nn.Linear(11,128)
        self.fc2 = nn.Linear(128,32)
        self.fc3 = nn.Linear(32,1)
    def forward(self,x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
net = WineNet()
print(net)
```

```
WineNet(
  (fc1): Linear(in_features=11, out_features=128, bias=True)
  (fc2): Linear(in_features=128, out_features=32, bias=True)
  (fc3): Linear(in_features=32, out_features=1, bias=True)
)
```

Figure 1: 神经网络结构

2.4 训练

用梯度下降法进行模型参数更新，记下每轮迭代中的训练损失和测试损失。

- **数据准备**：将 pandas 数据转换为 PyTorch 张量，特征数据 X 选取除最后一列外的所有列，标签数据 y 选取最后一列，并 reshape 为 $(1, -1)$。
- **训练阶段**：初始化神经网络模型，使用均方误差 (MSE) 作为损失函数，使用随机梯度下降 (SGD) 优化器，学习率为 0.01，迭代次数为 100 次。
- **循环过程**：
  - 将模型设置为训练模式。
  - 进行前向传播，计算训练损失。
  - 反向传播并更新模型参数。
  - 注意清空梯度，避免梯度累积导致性能下降。
- **测试阶段**：将模型设置为测试模式，进行前向传播并计算测试损失。

```
X_train = torch.tensor(train_data.iloc[:, :-1].values, dtype=torch.
    float32) # 训练特征
y_train = torch.tensor(train_data.iloc[:, -1].values, dtype=torch.
    float32).view(-1, 1) # 训练标签
X_test = torch.tensor(test_data.iloc[:, :-1].values, dtype=torch.
    float32) # 测试特征
y_test = torch.tensor(test_data.iloc[:, -1].values, dtype=torch.
    float32).view(-1, 1) # 测试标签

net = WineNet()# 初始化模型
criterion = nn.MSELoss()# 定义损失函数

learning_rate = 0.01
```

```
epochs = 100

train_losses = []
test_losses = []

optimizer = torch.optim.SGD(net.parameters(),lr = learning_rate)

for epoch in range(epochs):
    net.train()# 训练阶段
    output = net(X_train)# 前向传播
    train_loss = criterion(output, y_train) # 计算训练损失

    optimizer.zero_grad()
    train_loss.backward()
    optimizer.step()

    net.zero_grad()# 清空梯度

    # 测试阶段
    net.eval() # 设置为评估模式
    with torch.no_grad(): # 禁用梯度计算
        test_output = net(X_test) # 前向传播
        test_loss = criterion(test_output, y_test) # 计算测试损失

    # 记录每轮的训练损失和测试损失
    train_losses.append(train_loss.item())
    test_losses.append(test_loss.item())

    if epoch % 10 == 0:
        print(f'Epoch[{epoch+1}/{epochs}],TrainLoss:{train_loss.item()
            :.4f},TestLoss:{test_loss.item():.4f}')
```

```
Epoch [1/100], Train Loss: 0.2175, Test Loss: 0.2044
Epoch [11/100], Train Loss: 0.1200, Test Loss: 0.1145
Epoch [21/100], Train Loss: 0.0728, Test Loss: 0.0709
Epoch [31/100], Train Loss: 0.0494, Test Loss: 0.0491
Epoch [41/100], Train Loss: 0.0374, Test Loss: 0.0379
Epoch [51/100], Train Loss: 0.0312, Test Loss: 0.0320
Epoch [61/100], Train Loss: 0.0280, Test Loss: 0.0289
Epoch [71/100], Train Loss: 0.0263, Test Loss: 0.0273
Epoch [81/100], Train Loss: 0.0254, Test Loss: 0.0265
Epoch [91/100], Train Loss: 0.0250, Test Loss: 0.0260
```

Figure 2: 训练损失和测试损失

可以看到，训练损失和测试损失都随着迭代次数的增加而减小，说明模型在逐渐收敛。

除了用 PyTorch 的优化器来更新 net 中的参数外，还可以手动更新参数，用梯度下降法手动实现。

```
    with torch.no_grad(): # 禁用梯度计算
    for f in net.parameters():
        f.data.sub_(f.grad.data * learning_rate) # 手动梯度下降更新参数
```

2.5 训练损失测试损失展示

```
    plt.plot(range(len(train_losses)), train_losses, label='Train Loss')
plt.plot(range(len(test_losses)), test_losses, label='Test Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Testing Loss')
plt.legend()
plt.grid()
plt.show()
```

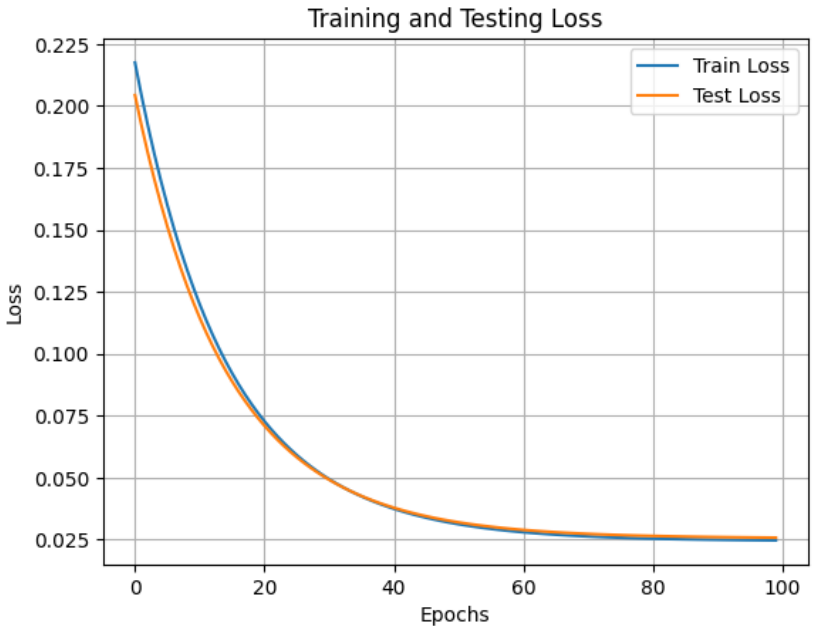

Figure 3: 训练损失和测试损失

该图显示了训练损失和测试损失随迭代次数的变化情况。可以看到，训练损失和测试损失都随着迭代次数的增加而减小，说明模型在逐渐收敛。并且训练损失和测试损失的差距在逐渐减小，说明模型在逐渐泛化。

3 总结

这次试验首先学习了 pytorch 库，还有 tensor 的创建，运算，和 ndarray 的转换。然后学习了神经网络的搭建，前向传播和后向传播。最后学习了如何用 pytorch 训练一个神经网络。

实验主要围绕对红酒品质的预测，用神经网络来预测红酒的品质。我认为我需要注意的地方有，在神经网络的搭建的时候，要注意输入和输出的维度，以及神经网络的深度和隐藏层的大小。在训练的时候，要注意梯度的清空，避免梯度累积导致性能下降。