实验报告

——使用基于 `ReLU` 的神经网络拟合函数

函数定义

本次实验需要拟合的函数为：

$$y = 3x^3 + 4x^2 + 5x + 6$$

从公式中可以看出输入 $\mathbf{x}$ 一共 3 个维度，输出的 y 是一个标量。并且输入输出均是连续值。

数据采集

数据采集分为两步：

- 生成数据：

  设置总数据集的大小为 1000，训练集与测试集的比例是 $7:3$。

  - 输入 $\mathbf{x}$：从标准正态分布 $\mathcal{N}(0,1)$ 中采样输入数据，由于特征维度是 3，因此采样得到的输入矩阵形状是 1000×3
  - 标签 y：根据函数定义 $y = 3x^3 + 4x^2 + 5x + 6$，代入 $\mathbf{x}$ 计算得到标签 y，标签矩阵的形状是 1000×1

  具体代码如下：

```
def synthetic_data(num_examples):
    '''生成数据'''
    x = torch.normal(0,1,(num_examples,3)) # (样本数量，特征数量)
    y = 3 * torch.pow(x[:,0],3) + 4 * torch.pow(x[:,1],2) + 5 * x[:,2] + 6
    return x,y.reshape(-1,1)
```

- 数据标准化：

  对输入 $\mathbf{x}$ 和标签 y 做归一化处理，保证其的均值为 0，标准差为 1，这样做有助于加速收敛。

  具体代码如下：

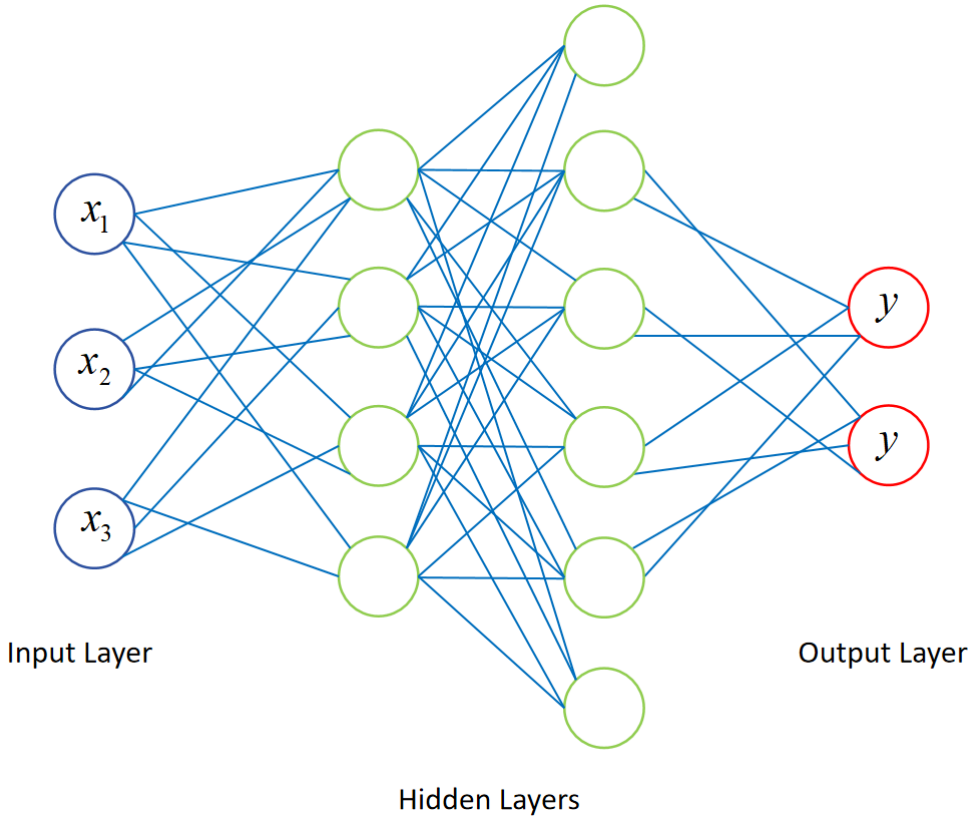
```
def scale_data(x,y):
    '''数据标准化'''
    x_mean = torch.mean(x)
    x_std = torch.std(x)
    y_mean = torch.mean(y)
    y_std = torch.std(y)
    x_scale = (x - x_mean) / x_std
    y_scale = (y - y_mean) / y_std
    return x_scale,y_scale
```

模型描述

- 模型架构：

  模型一共有 3 层，2 个隐藏层的激活函数均使用 $ReLU$ 函数，第一层隐藏层的神经元数量为 4，第二层隐藏层的神经元数量为 6，输入的特征维度是 3，输出是一个标量。

- 损失函数：平方和损失函数 `MSE`
- 优化器：`Adam` 优化器，学习率是 0.1 ，迭代次数为 200 。



训练过程的具体代码如下：

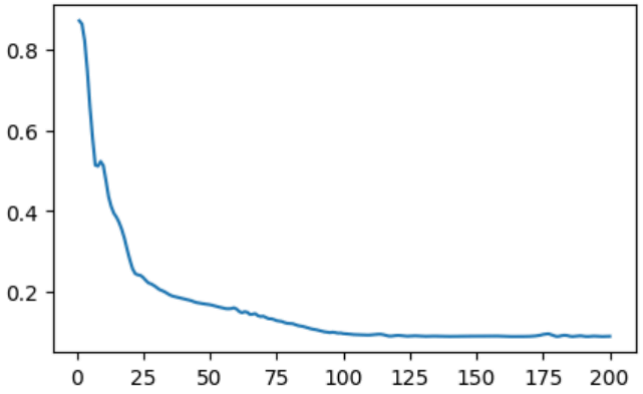
```
class ReLU_2(nn.Module):
    '''模型定义'''
    def __init__(self,num_inputs,num_hidden1,num_hidden2,num_outputs):
        super(ReLU_2,self).__init__()
        self.W1 = nn.Parameter(torch.normal(0,0.1,size=
(num_inputs,num_hidden1),requires_grad=True))
        self.b1 = nn.Parameter(torch.zeros((1,num_hidden1),requires_grad=True))

        self.W2 = nn.Parameter(torch.normal(0,0.1,size=
(num_hidden1,num_hidden2),requires_grad=True))
        self.b2 = nn.Parameter(torch.zeros((1,num_hidden2),requires_grad=True))

        self.W3 = nn.Parameter(torch.normal(0,0.1,size=
(num_hidden2,num_outputs),requires_grad=True))
        self.b3 = nn.Parameter(torch.zeros((1,num_outputs),requires_grad=True))

    def forward(self,x):
        h1 = F.relu(torch.matmul(x,self.W1)+self.b1)
        h2 = F.relu(torch.matmul(h1,self.W2)+self.b2)
        outputs = torch.matmul(h2,self.W3)+self.b3
        return outputs

# 参数设置
num_epochs,lr = 200,0.1
relu_2 = ReLU_2(num_inputs=3,num_hidden1=4,num_hidden2=6,num_outputs=1)
optimizer = torch.optim.Adam(relu_2.parameters(),lr=lr)

# 训练
```

```
def train(net,x,y,num_epochs,lr,optimizer):
    for epoch in range(num_epochs):
        y_hat = net(x)
        loss = MSE_loss(y_hat,y)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        print(f"epoch: {epoch+1}\tloss: {loss}")
    return y_hat

y_hat = train(relu_2,x_train,y_train,num_epochs,lr,optimizer)
```

拟合效果

- 损失值：

训练集上的损失值在迭代过程中的变化情况如下图所示：



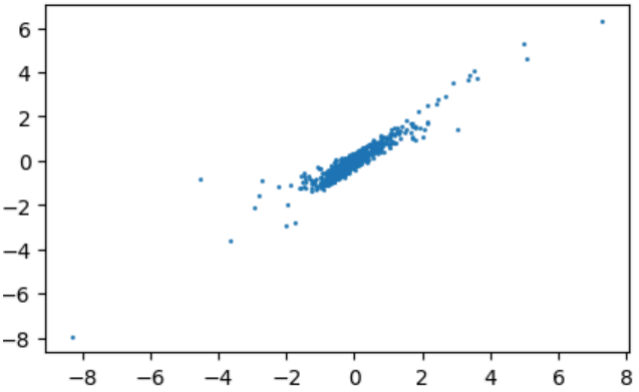
可以看到训练过程较为稳定，损失值随着训练的进行不断下降，最终损失值低于 0.1 。

测试集上的误差为 0.163 ，损失同样较低。

- 一致性：

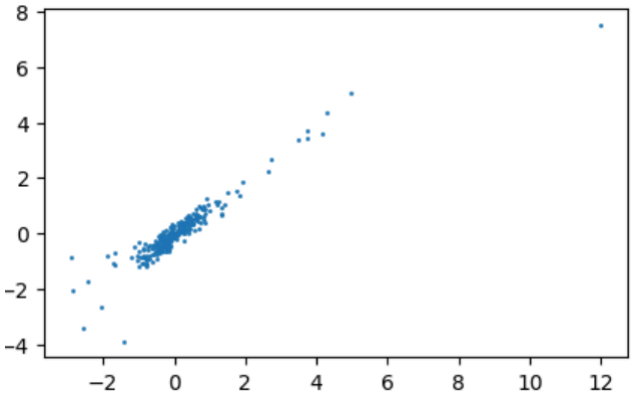
在训练集上，观察预测值和真实值之间的一致性程度，并绘制 $y_{train}-\hat{y}$ 散点图，可以看到 $y_{train}-\hat{y}$ 接近于 $y=x$ ，说明它们之间的一致性程度较高，拟合效果良好。

```
plt.figure(figsize=(5,3))
plt.scatter(y_train.detach().numpy(),y_hat.detach().numpy(),1)
```



在测试集上，使用同样的方法，观察到拟合效果同样良好，$y_{test}-\hat{y}$ 接近于 $y=x$ 。

```
plt.figure(figsize=(5,3))
plt.scatter(y_test.detach().numpy(),y_hat.detach().numpy(),1)
```



总结

通过实验证明，以多项式函数 $y = 3x^3 + 4x^2 + 5x + 6$ 为例，最终取得的拟合结果良好， **两层函数能够拟合任何函数** 在这个例子中成立。