函数拟合报告

2253899 蒋怡东

一、函数定义

在本函数拟合实验中，我将拟合函数设置为：

$$f(x) = 5x^4 + 4x^3 + 3x^2 + 2x + 1$$

```
function = lambda x: 5* x**4 + 4* x**3 + 3* x**2 + 2*x + 1
```

图 1

二、数据采集

首先，通过设置随机数种子为 42，由此确保生成的随机数序列是可复现的。然后，使用 NumPy 库的 random.uniform 函数生成 1000 个在-1 到 1 之间的随机浮点数，将其作为输入特征 x_values。接着，使用目标拟合函数计算得到输入特征 x_values 所对应的输出结果 y_values。

随后，将输入特征 x_values 和输出结果 y_values 组合成一个二维数组 data，其中每一行代表一个数据点的输入和输出对。为了确保训练过程中数据的随机性，使用 np.random.shuffle 函数随机打乱组合后的数据集，并将其划分为训练集和测试集。前 800 个数据点被分配为训练集 train_data，剩余的 200 个数据点被分配为测试集 test_data。

```python
np.random.seed(42)

# 生成1000个随机的x值，x在-5到5之间
x_values = np.random.uniform(-1, 1, 1000)
y_values = function(x_values)

# 将x和y组合成数据集
data = np.column_stack((x_values, y_values))

# 打乱数据
np.random.shuffle(data)

# 划分训练集和测试集
train_data = data[:800]
test_data = data[800:]

# 提取训练集和测试集的x和y
x_train, y_train = train_data[:, 0], train_data[:, 1]
x_test, y_test = test_data[:, 0], test_data[:, 1]

# 打印训练集和测试集的形状以验证
print("训练集形状:", x_train.shape, y_train.shape)
print("测试集形状:", x_test.shape, y_test.shape)
```

图 2

三、模型描述

图 3 为 PyTorch 实现的 2 层 Relu 网络，第一层为一个线性变换，将输入维度从 1 扩展到 128 个神经元，随后应用 ReLU 激活函数。第二层也为一个线性变换，将 128 个神经元的输出压缩回单一输出值。

图 4 为 NumPy 实现的 2 层 Relu 网络，其结构与 PyTorch 实现的一致，然而其中的函数均由 NumPy 实现。relu 函数实现了 ReLU 激活函数，它将所有负输入值置为 0，从而引入非线性。loss_compute 函数用于计算模型预测值与真实值之间的均方误差损失，作为评估模型性能的主要指标。gradient_compute 函数实现了反向传播算法，用于计算损失函数关于模型参数的梯度。

```python
class MyModel(nn.Module):
    def __init__(self):
        super(MyModel, self).__init__()
        self.fc1 = nn.Linear(1, 128)
        self.fc2 = nn.Linear(128, 1)

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

图 3

```python
class MyModel:
    def __init__(self, input_size, hidden_size, output_size):
        """
        初始化神经网络模型
        input_size: 输入维度
        hidden_size: 隐藏层维度
        output_size: 输出维度
        """
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.output_size = output_size

        # 初始化权重和偏置
        self.W1 = np.random.randn(input_size, hidden_size) * 0.01
        self.b1 = np.zeros((1, hidden_size))
        self.W2 = np.random.randn(hidden_size, output_size) * 0.01
        self.b2 = np.zeros((1, output_size))

    def relu(self, x):
        """ReLU激活函数"""
        return np.maximum(x, 0)

    def forward(self, x):
        """前向传播"""
        h1 = self.relu(np.matmul(x, self.W1) + self.b1)
        output = np.matmul(h1, self.W2) + self.b2
        return output, h1

    def loss_compute(self, y_pred, y_true):
        """计算均方误差损失"""
        return np.mean((y_pred - y_true)**2)

    def gradient_compute(self, x, y_true, y_pred, h1):
        """计算梯度"""
        # 计算输出层的梯度
        dloss = 2 * (y_pred - y_true) / x.shape[0]  # 均方误差的梯度
        dW2 = np.dot(h1.T, dloss)
        db2 = np.sum(dloss, axis=0, keepdims=True)
        # 计算第一隐藏层的梯度
        dh1 = np.dot(dloss, self.W2.T)  # 从输出层反向传播到h1
        drelu1 = (h1 > 0).astype(float) * dh1  # ReLU梯度
        dW1 = np.dot(x.T, drelu1)
        db1 = np.sum(drelu1, axis=0, keepdims=True)

        return dW1, db1, dW2, db2
```

图 4

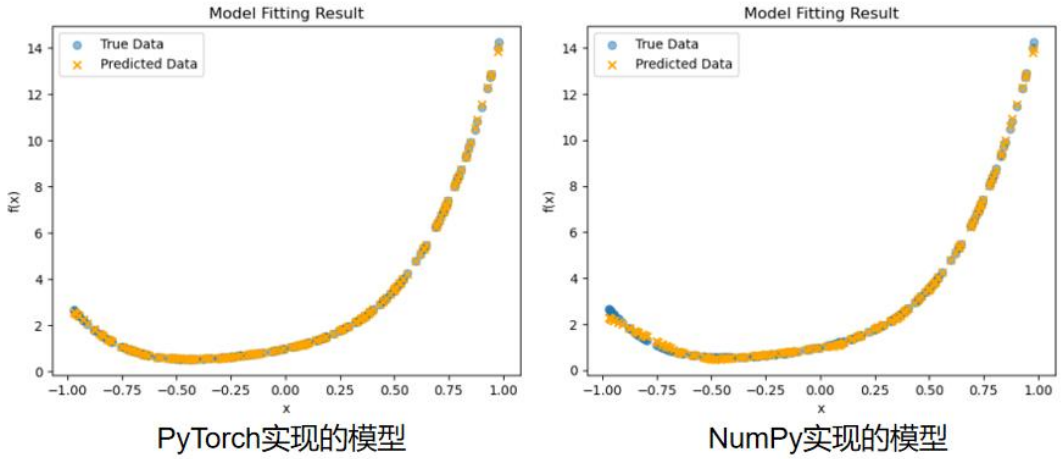
图 5 为 NumPy 实现的模型所用的训练函数。在每个 epoch 中，函数首先进行前向传播，计算模型的预测输出。然后，计算预测输出与真实标签之间的损失。接着，通过反向传播计算损失函数关于模型参数的梯度。最后，根据这些梯度和预定的学习率更新模型的权重和偏置，并输出当前轮的 loss。

```python
def train(self, x_train, y_train, learning_rate=0.01, num_epochs=10000):
    """训练模型"""
    for epoch in range(num_epochs):
        # 前向传播
        y_pred, h1 = self.forward(x_train)

        # 计算损失
        loss = self.compute_loss(y_pred, y_train)

        # 计算梯度
        dW1, db1, dW2, db2 = self.compute_gradients(x_train, y_train, y_pred, h1)

        # 更新权重
        self.W1 -= learning_rate * dW1
        self.b1 -= learning_rate * db1
        self.W2 -= learning_rate * dW2
        self.b2 -= learning_rate * db2

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

图 5

四、拟合效果

下图分别为 PyTorch 实现的和 NumPy 实现的两层 Relu 模型的拟合效果。二者均很好地拟合了目标函数，而在[-1.00, -0.50]区间中，PyTorch 实现的模型的拟合效果更好。



PyTorch实现的模型　　NumPy实现的模型

图 6