基于 NumPy 的 ReLU 神经网络函数拟合报告

2252086 孙靖贻

一、函数定义

本实验旨在使用 两层 ReLU 神经网络 来拟合一个自定义函数。目标函数定义如下：

$$f(x) = sin(2\pi x) + 0.5x$$

其中：

- $sin(2\pi x)$是一个周期性函数，增加了非线性成分；
- $0.5x$ 是一个线性项，增加了一定的趋势；
- 目标是让神经网络通过学习数据，逼近该函数。

二、数据采集

在区间[-1, 1]上采样数据：

- 训练集：100 个点，并加入随机噪声$N(0, 0.1^2)$以增强泛化能力；
- 测试集：50 个点，不添加噪声，用于评估拟合效果。

数据采样方式：

```
num_train, num_test = 100, 50
x_train = np.linspace(1, 1, num_train).reshape(1, 1)
y_train = target_function(x_train) + 0.1    np.random.randn(num_train, 1)

x_test = np.linspace(1, 1, num_test).reshape(1, 1)
y_test = target_function(x_test)
```

其中 target_function(x)是目标函数。

三、模型描述

本实验采用两层 ReLU 神经网络，结构如下：

①输入层：1 个神经元（输入x）。
②隐藏层 1：50 个神经元，ReLU 激活。
③隐藏层 2：30 个神经元，ReLU 激活。
④输出层：1 个神经元，线性输出。

数学表达：

$$z_1 = W_1 x + b_1$$
$$a_1 = ReLU(z_1)$$
$$z_2 = W_2 a_1 + b_2$$
$$a_2 = ReLU(z_2)$$
$$z_3 = W_3 a_2 + b_3$$
$$\hat{y} = z_3$$

其中：

W_1，W_2，W_3是权重矩阵。

b_1，b_2，b_3是偏置项。

ReLU 定义为：

$$ReLU(x) = max(0, x)$$

权重初始化采用 Xavier 初始化，优化方法采用梯度下降，损失函数使用均方误差(MSE)：

$$Loss = \frac{1}{N}\sum(\hat{y} - y)^2$$

训练 5000 轮，每 500 轮打印一次损失。

四、拟合效果

1. 训练损失

训练过程中，损失不断下降，表明网络在学习目标函数：

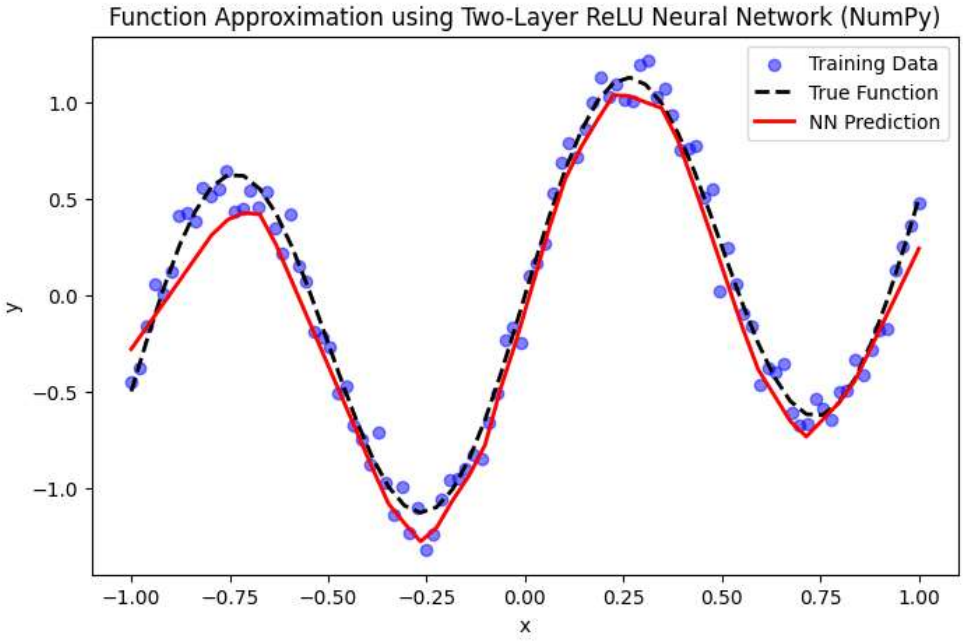
Epoch 0, Loss: 0.576609
Epoch 500, Loss: 0.131081
Epoch 1000, Loss: 0.107649
Epoch 1500, Loss: 0.084627
Epoch 2000, Loss: 0.066338
Epoch 2500, Loss: 0.053041
Epoch 3000, Loss: 0.029449
Epoch 3500, Loss: 0.030105
Epoch 4000, Loss: 0.024282
Epoch 4500, Loss: 0.020131

可以看出，虽然损失下降，但依然有一定误差，说明网络可能需要更深层结构或者更长训练时间。

2. 可视化结果

下图展示了神经网络的拟合情况：

- 蓝色点：训练数据。
- 黑色虚线：真实目标函数 $f(x)$。
- 红色线：神经网络预测的 y 值。



训练后，红色曲线逐渐向目标函数逼近，但拟合仍然存在一定误差，尤其是在极值点附近。

该实验证明了 ReLU 网络的拟合能力，也展示了 NumPy 实现神经网络的可行性。

附件：代码

```
import numpy as np
import matplotlib.pyplot as plt

# 设置随机种子
np.random.seed(42)

# 目标函数
def target_function(x):
    return np.sin(2 * np.pi * x) + 0.5 * x

# 生成数据
num_train, num_test = 100, 50
x_train = np.linspace(-1, 1, num_train).reshape(-1, 1)
y_train = target_function(x_train) + 0.1 * np.random.randn(num_train,
1)
x_test = np.linspace(-1, 1, num_test).reshape(-1, 1)
y_test = target_function(x_test)

# 两层隐藏层结构
input_size = 1
hidden_size1 = 50  # 第一层隐藏层
hidden_size2 = 30  # 第二层隐藏层
output_size = 1

# Xavier 初始化
W1 = np.random.randn(input_size, hidden_size1) * np.sqrt(2 /
input_size)
b1 = np.zeros((1, hidden_size1))

W2 = np.random.randn(hidden_size1, hidden_size2) * np.sqrt(2 /
hidden_size1)
b2 = np.zeros((1, hidden_size2))

W3 = np.random.randn(hidden_size2, output_size) * np.sqrt(2 /
hidden_size2)
b3 = np.zeros((1, output_size))

# ReLU 及其梯度
def relu(x):
```

```
    return np.maximum(0, x)

def relu_derivative(x):
    return (x > 0).astype(float)

# 训练参数
learning_rate = 0.05
epochs = 5000

# 训练
for epoch in range(epochs):
    # 前向传播
    z1 = x_train @ W1 + b1
    a1 = relu(z1)  # 第一层 ReLU

    z2 = a1 @ W2 + b2
    a2 = relu(z2)  # 第二层 ReLU

    z3 = a2 @ W3 + b3  # 线性输出层
    y_pred = z3

    # 计算损失
    loss = np.mean((y_pred - y_train) ** 2)

    # 反向传播
    grad_y_pred = 2 * (y_pred - y_train) / num_train  # dL/dy_pred

    grad_W3 = a2.T @ grad_y_pred
    grad_b3 = np.sum(grad_y_pred, axis=0, keepdims=True)

    grad_a2 = grad_y_pred @ W3.T
    grad_z2 = grad_a2 * relu_derivative(z2)

    grad_W2 = a1.T @ grad_z2
    grad_b2 = np.sum(grad_z2, axis=0, keepdims=True)

    grad_a1 = grad_z2 @ W2.T
    grad_z1 = grad_a1 * relu_derivative(z1)

    grad_W1 = x_train.T @ grad_z1
    grad_b1 = np.sum(grad_z1, axis=0, keepdims=True)

    # 参数更新
    W1 -= learning_rate * grad_W1
```

```
    b1 -= learning_rate * grad_b1
    W2 -= learning_rate * grad_W2
    b2 -= learning_rate * grad_b2
    W3 -= learning_rate * grad_W3
    b3 -= learning_rate * grad_b3

    # 打印损失
    if epoch % 500 == 0:
        print(f"Epoch {epoch}, Loss: {loss:.6f}")

# 测试
z1_test = x_test @ W1 + b1
a1_test = relu(z1_test)

z2_test = a1_test @ W2 + b2
a2_test = relu(z2_test)

z3_test = a2_test @ W3 + b3
y_test_pred = z3_test

# 可视化
plt.figure(figsize=(8, 5))
plt.scatter(x_train, y_train, label="Training Data", color="blue",
alpha=0.5)
plt.plot(x_test, y_test, label="True Function", color="black",
linestyle="dashed", linewidth=2)
plt.plot(x_test, y_test_pred, label="NN Prediction", color="red",
linewidth=2)
plt.legend()
plt.title("Function Approximation using Two-Layer ReLU Neural Network
(NumPy)")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```