numpy版本报告

一、函数定义

1. 目标函数

目标函数 `target_function` 用于生成数据，其数学表达式为 $y=3x^3+x^2+2x+1$。代码实现如下：

```
def target_function(x):
    return  3*x**3+ x**2 + 2*x + 1
```

该函数接收输入 x，并返回对应的 y 值，作为模型要学习拟合的真实数据。

2. 激活函数及其导数

- **ReLU 函数**：`relu` 函数是神经网络中常用的激活函数，其作用是引入非线性。数学表达式为 (ReLU(x)=\max(0,x))。代码实现如下：

```
def relu(self, x):
    return np.maximum(0, x)
```

- **ReLU 导数函数**：`relu_derivative` 函数用于反向传播时计算梯度。当 (x>0) 时，导数为 1；当 (x\leq0) 时，导数为 0。代码实现如下：

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

3. 绘图函数

`plot_results` 函数用于绘制真实函数和模型预测结果的对比图，便于直观观察模型的拟合效果。代码实现如下：

```
def plot_results(x_test, y_test, y_pred, title):
    plt.plot(x_test, y_test, label='True Function', color='green')
    plt.plot(x_test, y_pred, label='Model Prediction', color='red',
linestyle='dashed')
    plt.legend()
    plt.title(title)
    plt.show()
```

二、数据采集

1. 训练数据

使用 `np.random.uniform(-1, 1, 100)` 从区间 ([-1, 1]) 内均匀随机采样 100 个点作为输入特征 (x{train})，*将其输入到目标函数* `target_function` *中得到对应的输出值* (y{train})，从而生成 100 个训练样本 ((x{train}, y{train}))。

2. 测试数据

使用 `np.linspace(-1, 1, 100)` 在区间 ([-1, 1]) 内均匀生成 100 个点作为输入特征 (x{test})，*同样输入到目标函数得到对应的输出值* (y{test})。这些样本用于评估模型在未见过数据上的泛化能力。

3. 数据形状调整

为了满足模型输入要求，将 `x_train`、`y_train` 和 `x_test` 调整为二维数组形状，代码如下：

```
x_train_np = x_train.reshape(-1, 1)
y_train_np = y_train.reshape(-1, 1)
x_test_np = x_test.reshape(-1, 1)
```

三、模型描述

1. 模型结构

`NumPyReLUNet` 是一个两层的 ReLU 神经网络，包含输入层、隐藏层和输出层。输入层接收 1 个特征，隐藏层有 30 个神经元，输出层输出 1 个预测值。

2. 初始化参数

在 `__init__` 方法中，随机初始化权重矩阵 `W1` 和 `W2`，并将偏置向量 `b1` 和 `b2` 初始化为零。学习率 `lr` 初始化为 0.05。代码如下：

```
def __init__(self, input_size=1, hidden_size=30, output_size=1, lr=0.05):
    self.lr = lr
    self.W1 = np.random.randn(input_size, hidden_size) * 0.1
    self.b1 = np.zeros((1, hidden_size))
    self.W2 = np.random.randn(hidden_size, output_size) * 0.1
    self.b2 = np.zeros((1, output_size))
```

3. 前向传播

`forward` 方法实现了前向传播过程，输入数据依次经过线性变换、ReLU 激活函数和另一个线性变换得到预测值。代码如下：

```
def forward(self, x):
    self.z1 = x @ self.W1 + self.b1
    self.a1 = self.relu(self.z1)
    self.z2 = self.a1 @ self.W2 + self.b2
    return self.z2
```

4. 反向传播

`backward` 方法实现了反向传播过程，根据预测值和真实值的误差计算梯度，并更新模型参数。代码如下：

```
def backward(self, x, y, y_pred):
    loss_grad = 2 * (y_pred - y) / y.shape[0]
    dz2 = loss_grad
    dW2 = self.a1.T @ dz2
    db2 = np.sum(dz2, axis=0, keepdims=True)
    dz1 = (dz2 @ self.W2.T) * self.relu_derivative(self.z1)
    dW1 = x.T @ dz1
    db1 = np.sum(dz1, axis=0, keepdims=True)

    self.W1 -= self.lr * dW1
    self.b1 -= self.lr * db1
    self.W2 -= self.lr * dW2
    self.b2 -= self.lr * db2
```

5. 训练过程

`train` 方法通过多次迭代进行训练，每次迭代先进行前向传播得到预测值，再进行反向传播更新参数。每 100 个 epoch 打印一次损失值。代码如下：

```
def train(self, x, y, epochs=5000):
    for epoch in range(epochs):
        y_pred = self.forward(x)
        self.backward(x, y, y_pred)
        if epoch % 100 == 0:
            loss = np.mean((y_pred - y) ** 2)
            print(f"Epoch {epoch}, Loss: {loss:.4f}")
```

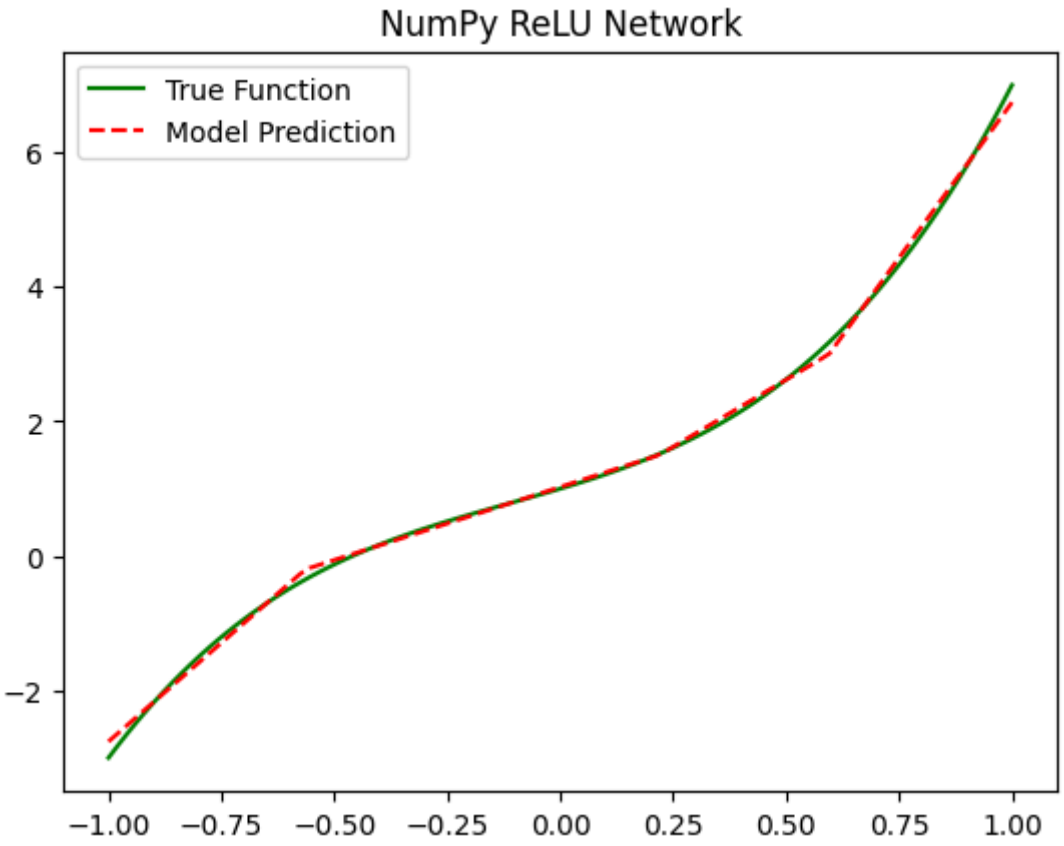
四、拟合效果

1.真实值与预测值对比

通过 `plot_results` 函数绘制真实函数和模型预测结果的对比图，可以直观地观察到模型的拟合效果。如果预测曲线与真实曲线较为接近，说明模型能够较好地拟合目标函数。

```
def plot_results(x_test, y_test, y_pred, title):
    plt.plot(x_test, y_test, label='True Function', color='green')
    plt.plot(x_test, y_pred, label='Model Prediction', color='red',
linestyle='dashed')
    plt.legend()
    plt.title(title)
    plt.show()
```

2.可视化结果


NumPy ReLU Network
True Function
Model Prediction
6
4
2
0
−2
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00