ReLU 神经网络拟合报告

一、函数定义

1.目标函数

```
def target_function(x):
    return x*x+0.5*x+5
```

功能：定义一个二次函数 y=x2+0.5x+5，作为拟合的目标函数。

2.ReLU 激活函数及其导数

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

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

relu(x)：实现 ReLU 激活函数，输出非负值。

relu_derivative(x)：计算 ReLU 的导数，输入大于 0 时导数为 1，否则为 0。

3.前向传播

```
def forward_pass(X, params):
    W1, b1, W2, b2 = params
    Z1 = np.dot(X, W1) + b1
    A1 = relu(Z1)
    Z2 = np.dot(A1, W2) + b2
    A2 = Z2
    return Z1, A1, Z2, A2
```

输入数据通过隐藏层（ReLU 激活）和输出层（无激活函数），返回各层计算结果。

4.损失函数

```
def mse_loss(y_true, y_pred):
    return np.mean((y_true - y_pred) ** 2)
```

计算均方误差（MSE），用于衡量预测值与真实值的差异。

5.反向传播与参数更新

backward_pass：通过链式法则计算各参数的梯度。

update_params：根据梯度下降法更新权重和偏置。

二、数据采集

数据采集部分使用 numpy 库生成了目标函数的数据。具体步骤如下：

1. 数据生成

```
np.random.seed(42)
X = np.linspace(-10, 10, 500).reshape(-1, 1)
y = target_function(X)
```

使用 np.linspace(-10, 10, 500)生成了 500 个在区间[−10,10]内的均匀分布的自变量

x 值。
2. 数据集划分

```
split_idx = int(0.8 * len(X))
X_train, X_test = X[:split_idx], X[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]
```

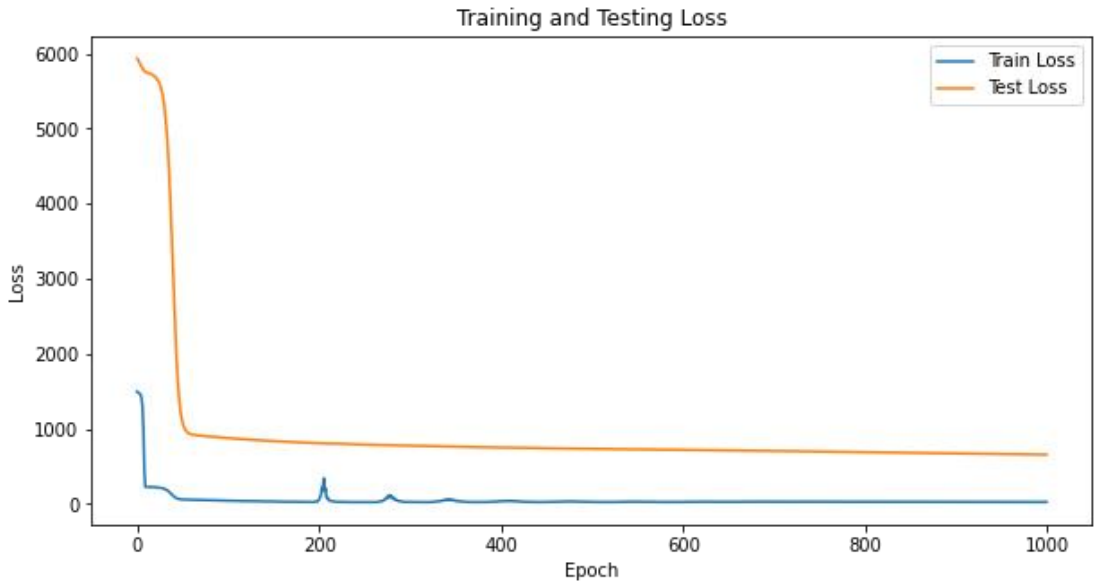
将这些 x 值通过目标函数 target_function(x)计算得到对应的因变量 y 值。
将数据划分为训练集和测试集，其中 80%的数据作为训练集，20%的数据作为测试集。训练集用于训练神经网络，测试集用于评估模型的泛化能力。

三、模型描述
1. 网络结构
输入层：1 个神经元（单变量输入）。
隐藏层：20 个神经元，使用 ReLU 激活函数。
输出层：1 个神经元，线性输出（无激活函数）。
2. 参数初始化

```
W1 = np.random.randn(input_dim, hidden_units) * 0.01
b1 = np.zeros((1, hidden_units))
```

3. 训练配置
训练轮次：1000 次。
学习率：0.005。
优化方法：梯度下降法，通过反向传播更新参数。

四、拟合效果
1. 损失曲线：
训练结束后，绘制了训练集和测试集的损失曲线。从图中可以看出，随着训练周期的增加，训练集和测试集的损失值均逐渐减小，表明模型在训练过程中逐渐学习到了数据的规律。
训练集的损失值在训练过程中持续下降，最终趋于稳定。测试集的损失值也表现出类似的下降趋势，但可能会略高于训练集的损失值，这是由于模型在测试集上存在一定的泛化误差。



2. 拟合结果图：
绘制了目标函数的真实数据点（训练数据和测试数据）以及神经网络拟合出的函数曲线。
从图中可以看出，神经网络拟合出的函数曲线能够较好地逼近目标函数的真实形状。在训练数据范围内，拟合曲线与真实数据点基本重合，说明模型对训练数据的拟合效果较好。
对于测试数据，拟合曲线也能较好地预测其对应的输出值，表明模型具有一定的泛化能力。