函数拟合

问题描述

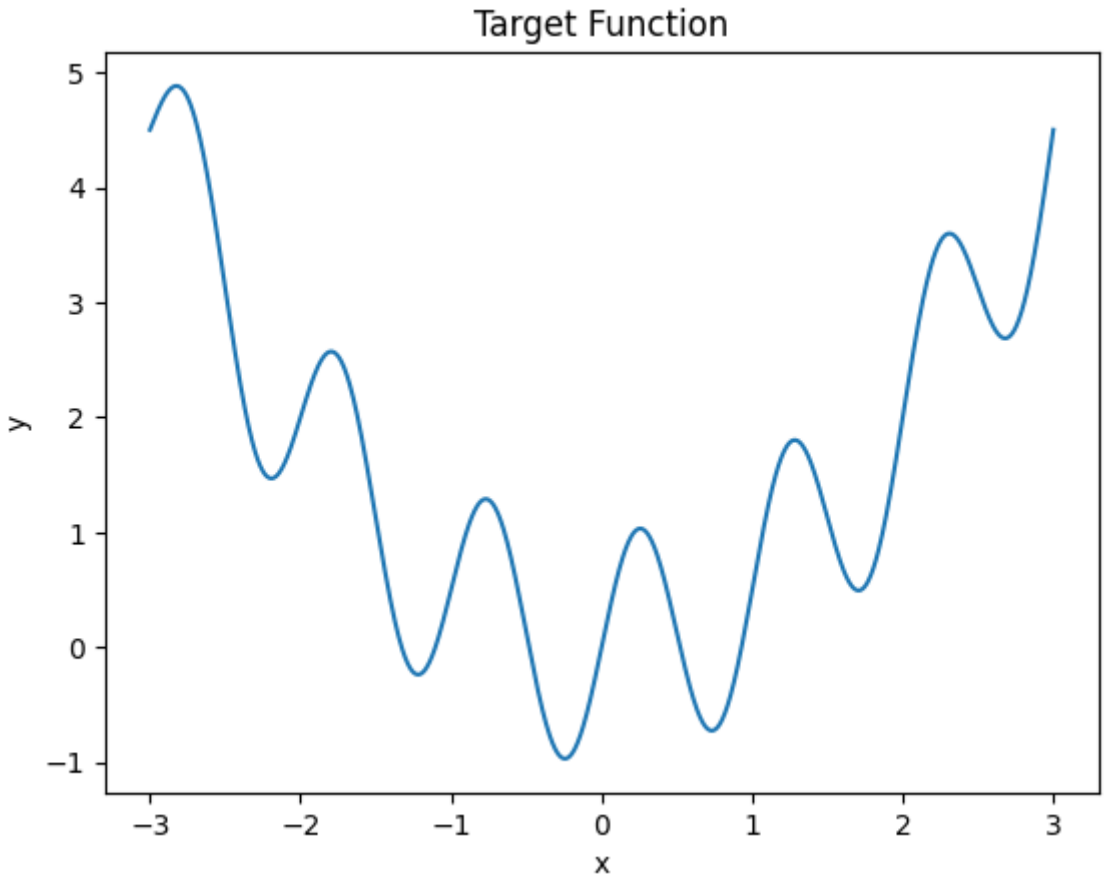
理论和实验证明，一个两层的`ReLU`网络可以模拟任何函数。请自行定义一个函数, 并使用基于`ReLU`的神经网络来拟合此函数。

实验设计

- 模拟的目标函数为 $sin(2\pi x) + 0.5x^2$
- 采用`pytorch`深度学习框架实现
- 搭建一个两层的`ReLU`网络
- 计划在某个区间上均匀的进行数据采用，采取`<x,y>`数据对，对其进行随机打乱和划分

函数定义

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



数据采集

```
def generate_data(num_samples=1000, test_size=0.2):
    np.random.seed(0)
    # 在[-3, 3]区间生成num_samples个随机数
    x = np.linspace(-3, 3, num_samples)
```

```
    np.random.shuffle(x)
    # 计算目标函数值
    y = target_function(x)

    # 分割训练集和测试集
    split_index = int(num_samples * (1 - test_size))
    train_x, test_x = x[:split_index], x[split_index:]
    train_y, test_y = y[:split_index], y[split_index:]

    return train_x, train_y, test_x, test_y
```

模型描述

模型采用双层全连接网络，网络后的激活函数为`ReLU`函数，最后经过一个全连接网络得到结果，

也就是$f(x) = fc_3(relu(fc_2(relu(fc_1(x)))))$

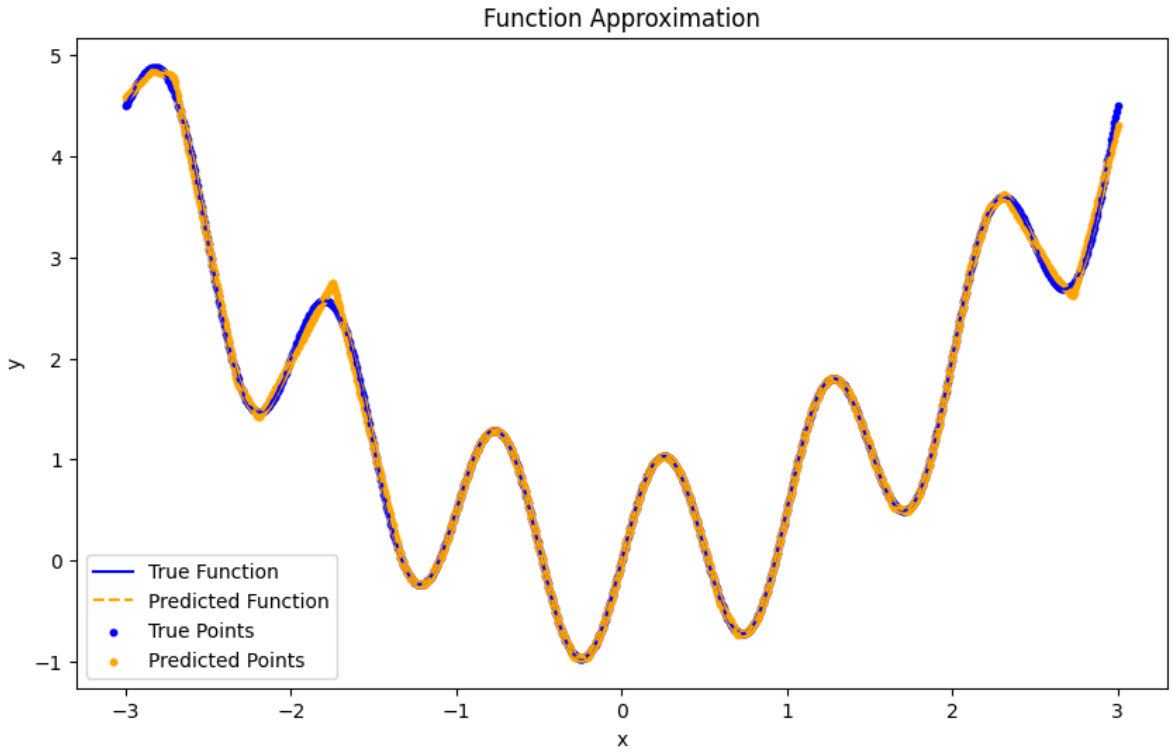
```
class TwoLayerReLUModel(nn.Module):
    def __init__(self, input_dim=1, hidden_dim=100, output_dim=1):
        super(TwoLayerReLUModel, self).__init__()
        # 定义网络结构，双层全连接网络
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.fc3 = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        # 定义前向传播过程，每个层使用ReLU激活函数
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

拟合效果

最后几个epoch的训练结果为:

```
Epoch 8600, Loss: 0.0018
Epoch 8700, Loss: 0.0018
Epoch 8800, Loss: 0.0024
Epoch 8900, Loss: 0.0017
Epoch 9000, Loss: 0.0017
Epoch 9100, Loss: 0.0017
Epoch 9200, Loss: 0.0032
Epoch 9300, Loss: 0.0017
Epoch 9400, Loss: 0.0017
Epoch 9500, Loss: 0.0018
Epoch 9600, Loss: 0.0016
Epoch 9700, Loss: 0.0016
Epoch 9800, Loss: 0.0016
Epoch 9900, Loss: 0.0018
```

与真实值的对比图为，


Function Approximation
True Function
Predicted Function
True Points
Predicted Points
x
y