homework 2-2

余苏皓　2152199

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。

理论证明：

证明：

1. 任意连续函数都可以由分段线性函数逼近

对于一个闭区间上的连续函数，可以构造一系列多项式，使得这些多项式在该闭区间上的一致逼近误差可以任意小。

具体来说，证明过程如下：

1. 对于一个闭区间$[a, b]$上的连续函数 $f(x)$，可以构造一个伯恩斯坦多项式序列 $B_n(x)$，使得 $B_n(x)$ 在 $[a, b]$ 上一致收敛于 $f(x)$。
2. 伯恩斯坦多项式是分段线性函数的线性组合。因此，对于任意给定的 $\epsilon>0$，都存在一个自然数 N，使得对于所有 $n≥N$，都有：

$$|f(x) - B_n(x)| < \epsilon$$

它表明，分段线性函数可以用来逼近任意连续函数，并且逼近精度可以任意高。

2. 两层ReLU网络可以表示任意分段线性函数

两层ReLU网络可以表示如下形式：

$$f(x) = w_2\sigma(w_1x + b_1) + b_2$$

其中，$w1$ 和 $w2$ 是权重矩阵，$b1$ 和 $b2$ 是偏置向量，$\sigma(x)$ 是ReLU激活函数。

ReLU激活函数定义如下：

$$\sigma(x) = max(0, x)$$

对于任意分段线性函数，都可以将其表示为一系列ReLU函数的组合。例如，对于分段线性函数：

$$f(x) = \begin{cases} 0 & x < c \\ x & c \le x < d \\ d - c & x \ge d \end{cases}$$

可以将其表示为以下ReLU函数的组合：

$$f(x) = \sigma(x - c) - \sigma(x - d)$$

因此,可以用两层ReLU来拟合任意的分段函数

证毕

实验证明：

一、函数定义

可更改需要拟合的函数，本实验测试下面这个多项式函数：

$$2x^2 - 3x + 1$$

代码如下：

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

二、数据采集

1. **设置随机种子:** 为了保证每次运行代码的结果一致，代码设置了随机种子为 `0`。
2. **生成数据:**
    - 使用 `np.random.uniform` 函数生成 `100` 个随机数，范围为 `[-1, 1]`，并将其赋值给 `X` 变量。
    - 使用 `fit_function` 函数计算 `X` 对应的真实值，并加入随机噪声，模拟真实数据中的误差。将结果赋值给 `Y` 变量。

```
np.random.seed(0)
num_points = 100
X_train = np.random.uniform(-1, 1, num_points)
Y_train = fit_function(X_train) + np.random.normal(0, 0.1, num_points)

X_test = np.random.uniform(-1, 1, num_points)
Y_test = fit_function(X_test) + np.random.normal(0, 0.1, num_points)
```

三、模型描述

定义网络结构和初始化参数

1. **设置超参数:**
    - `input_size`: 输入数据的维度，在本例中为 `1`，表示输入数据是一个单变量。
    - `hidden_size`: 隐藏层的节点数，在本例中为 `100`，表示隐藏层有 `100` 个神经元。
    - `output_size`: 输出数据的维度，在本例中为 `1`，表示输出数据是一个单变量。
2. **初始化权重和偏置:**
    - 使用 `np.random.randn` 函数生成随机权重矩阵 `W1` 和 `W2`，分别用于连接输入层和隐藏层，以及隐藏层和输出层。
    - 使用 `np.zeros` 函数生成零偏置向量 `b1` 和 `b2`，分别用于隐藏层和输出层。

```
input_size = 1
hidden_size = 100
output_size = 1

np.random.seed(0)
W1 = np.random.randn(hidden_size, input_size)
b1 = np.zeros((hidden_size, 1))
W2 = np.random.randn(output_size, hidden_size)
b2 = np.zeros((output_size, 1))
```

定义ReLU函数

ReLU函数的公式:

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

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

训练网络

1. **定义超参数:**
    - `num_epochs`: 训练迭代次数。
    - `learning_rate`: 学习率，控制参数更新的幅度。
2. **训练循环:**
    - 遍历所有训练迭代周期 (epoch) ：
        - 前向传播
        - 计算损失
            - 计算均方误差损失：`loss = np.mean((A2 - Y_train.reshape(1, -1))**2)`

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

        - 使用梯度下降法更新权重和偏置
        - 每 50 个迭代周期打印一次损失值

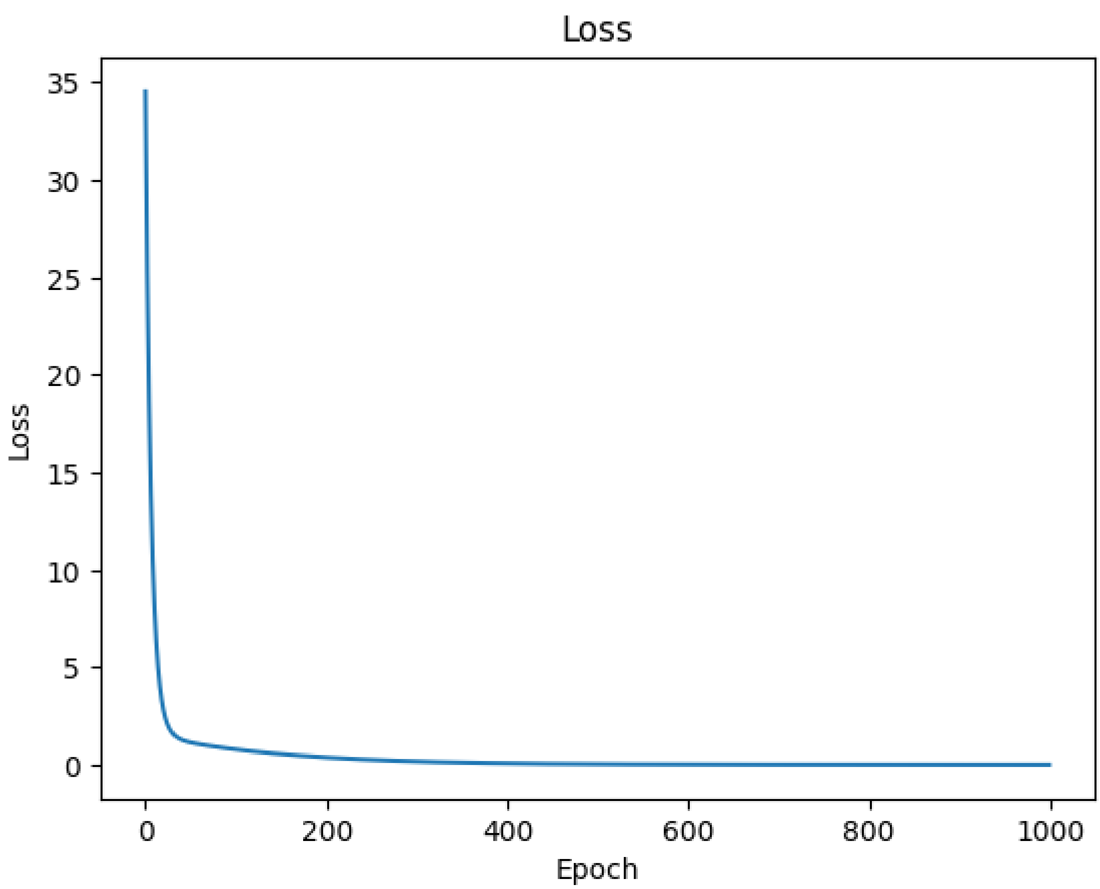
```
for epoch in range(num_epochs):
    # 前向传播
    Z1 = np.dot(W1, X_train.reshape(1, -1)) + b1
    A1 = relu(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = Z2

    # 计算损失
    loss = np.mean((A2 - Y_train.reshape(1, -1))**2)

    # 反向传播
    dA2 = 2 * (A2 - Y_train.reshape(1, -1))
```

```python
    dZ2 = dA2
    dW2 = np.dot(dZ2, A1.T)
    db2 = np.sum(dZ2, axis=1, keepdims=True)
    dA1 = np.dot(W2.T, dZ2)
    dZ1 = dA1 * (Z1 > 0)
    dW1 = np.dot(dZ1, X_train.reshape(1, -1).T)
    db1 = np.sum(dZ1, axis=1, keepdims=True)

    # 更新参数
    W1 -= learning_rate * dW1
    b1 -= learning_rate * db1
    W2 -= learning_rate * dW2
    b2 -= learning_rate * db2

    # 打印损失
    if epoch % 50 == 0:
        print(f"Epoch {epoch}, Loss: {loss}")
```

四、拟合效果

1. **训练集loss**

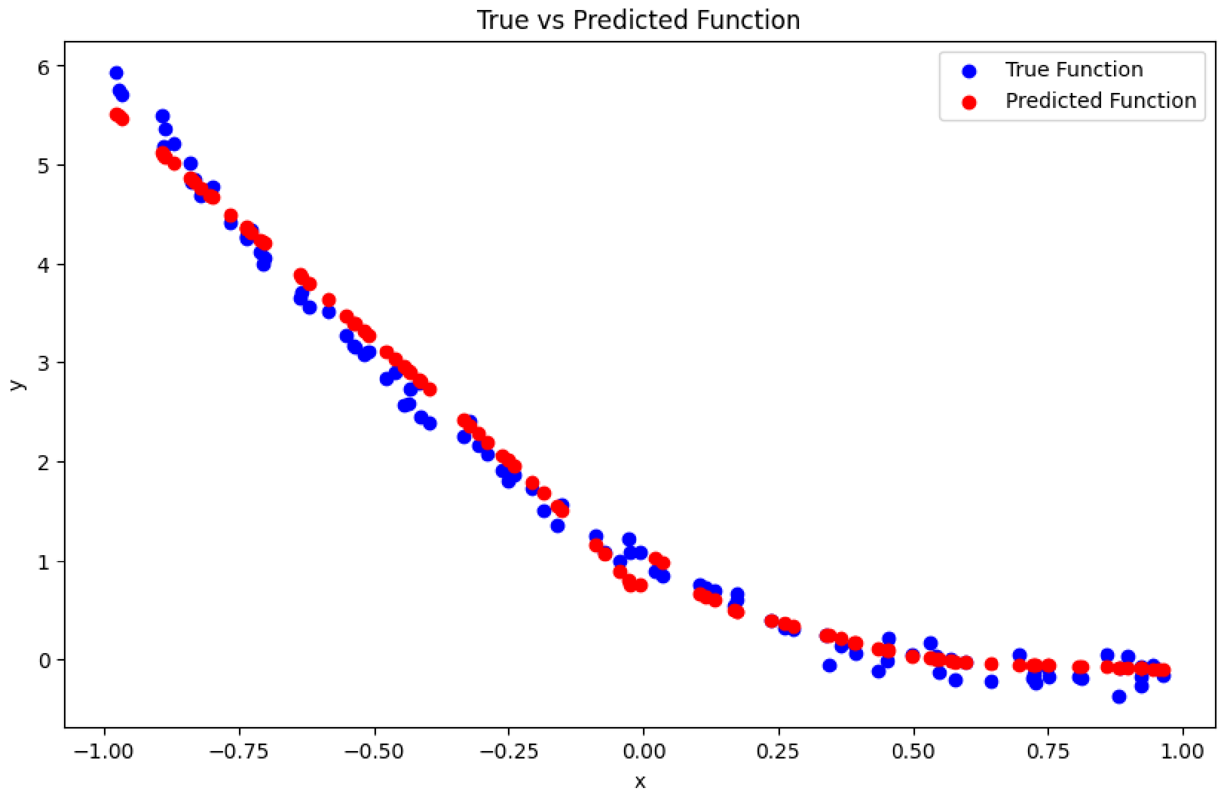
   最后一轮训练的Loss: 0.0218994134982404 73



2. **测试集拟合效果**

Test Loss: 0.030224253171291456

下图可视化结果可以看出效果非常好



True vs Predicted Function
True Function
Predicted Function
6
5
4
3
2
1
0
y
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
x