全连接神经网络之函数拟合及使用 NUMPY 实现

2152180 文溪

1. 函数定义

此处对函数进行定义，直接使用函数返回，便于定义所有函数：

```
def target_function(x):
    return np.sin(x) # 拟合的函数
```

这里以 sinx 为例，这是一个周期性函数，常见于自然界和工程领域，具有代表性和挑战性。

2. 数据采集

我生成了两个数据集：一个用于训练，另一个用于测试。训练数据集包含了从 x = -20 到 x = 20 范围内的 10000 个均匀分布的点。测试数据集包含了 x = -10 到 x = 10 范围内的 400 个均匀分布的点。对于每个数据点 x_i，我们使用目标函数 f(x_i) = sin(x_i)计算其对应的 y_i 值。

```
# 生成数据点
x_train = np.linspace(-20, 20, 10000).reshape(-1, 1)  # x 的范围从-10 到
20
y_train = target_function(x_train)  # 计算目标函数值
# 使用模型进行预测
x_test = np.linspace(-10, 10, 400).reshape(-1, 1)
y_pred = model.predict(x_test)
```

3. 模型描述

3.1 模型架构

该模型是一个基于 TensorFlow 的深度神经网络，由以下主要层次结构组成：

输入层：接受单一特征 x 的输入，这个特征是一个实数，代表我们希望拟合的函数 sin(x)的输入值。

隐藏层：第一隐藏层和第二隐藏层都配置有 64 个神经元。使用 ReLU (Rectified Linear Unit) 激活函数。ReLU 函数定义为 f(x)=max(0,x)，是目前深度学习中最普遍使用的激活函数之一，主要因为它在非负区域内保持梯度恒定，这有助于解决梯度消失问题，同时计算上也更为高效。

输出层：单一神经元，不带激活函数，直接输出网络的预测值。这个值旨在逼近目标函数

3.2 模型编译

优化器：模型使用 Adam 优化器。Adam 是一种自适应学习率优化算法，它结合了 RMSProp 和 Momentum 算法的特点，不仅关注过去梯度的指数衰减平均，还保留了梯度的方向，能够调整每个参数的学习率，使得模型训练更为高效和稳定。

损失函数：使用均方误差(MSE)作为损失函数，MSE 测量的是预测值与实际值之间差的平方的平均值，是回归任务中常用的损失函数。

4. 拟合效果

通过可视化工具（如 matplotlib），我们可以将模型的预测结果与目标函数 sin(x)的实际值进行对比，从而直观地评估模型的拟合效果。

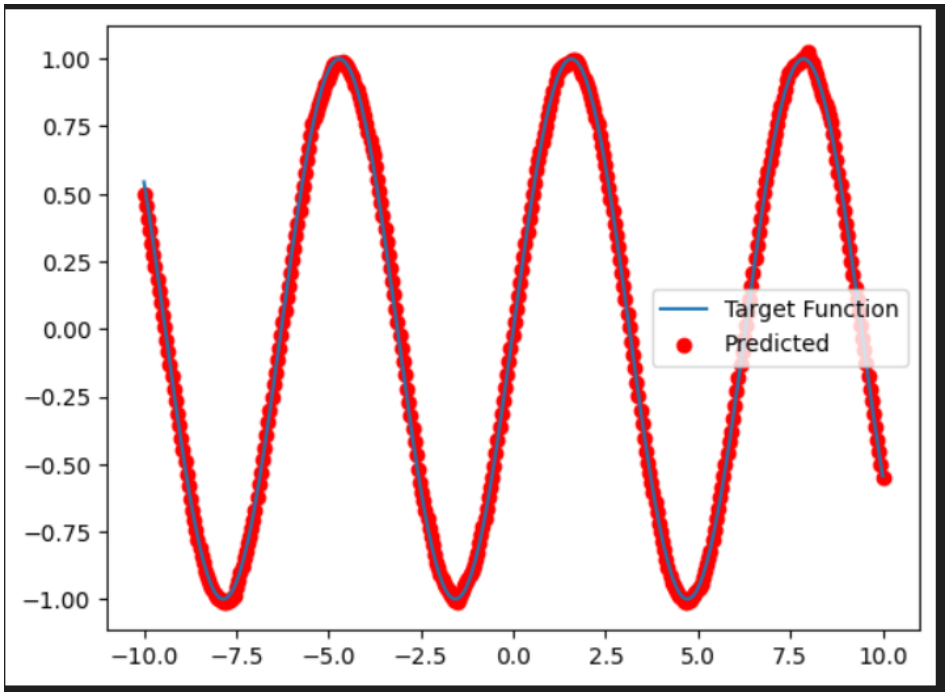

图 1 拟合 sinx 函数的图像

我们继续拟合另外的多项式函数观察效果：

比如说 x*x + 2*x + 5:

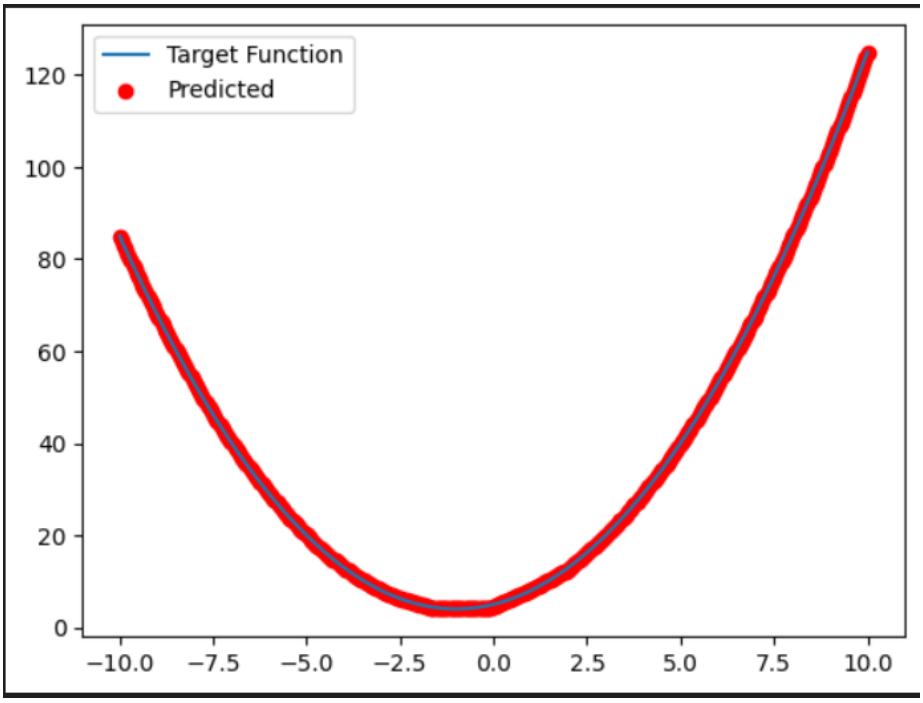

图 2 拟合 x*x + 2*x + 5 的图像

最后试一下线性函数 3*x + 4 的拟合效果：

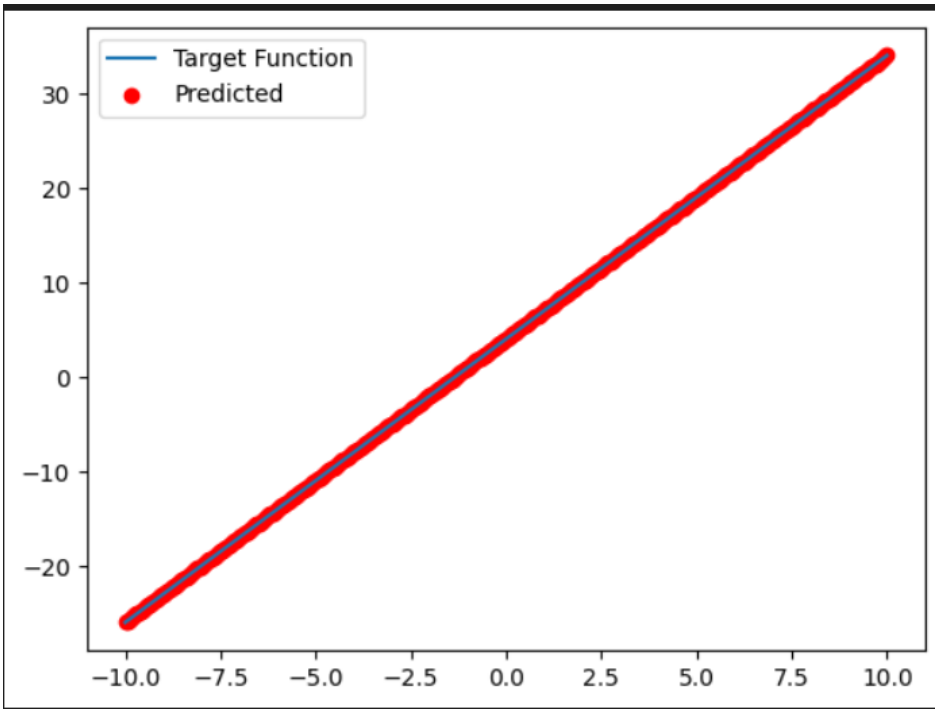

图 3 拟合 3*x + 4 的图像

加分项完成 -- 直接用 NumPy 实现

此处函数定义与数据采集与上面类似，直接进入模型描述。

1. 模型描述

1.1 初始化网络参数

输入层大小 (input_size) 设为 1，因为模型每次只接受单一的输入值 x。

隐藏层大小 (hidden_size) 设为 64，这是隐藏层神经元的数量，足以捕捉复杂的数据模式。

输出层大小 (output_size) 设为 1，因为模型的输出是一个实数，表示 x 的目标函数值。

权重(W1, W2)和偏置(b1, b2)通过随机初始化，并在训练过程中进行调整。

```
# 初始化网络参数
input_size = 1
hidden_size = 64
output_size = 1
np.random.seed(42) # 确保结果可复现
W1 = np.random.randn(input_size, hidden_size)
b1 = np.zeros((1, hidden_size))
W2 = np.random.randn(hidden_size, output_size)
b2 = np.zeros((1, output_size))
```

1.2 ReLU 激活函数及其导数

ReLU 激活函数 用于增加网络的非线性能力，定义为 ReLU(z)=max(0,z)。ReLU 导数用于反向传播过程，指示哪些神经元是激活的(z>0)。

```
# ReLU 激活函数及其导数
def relu(z):
    return np.maximum(0, z)

def relu_deriv(z):
    return z > 0
```

1.3 损失函数

使用均方误差（MSE）作为损失函数，计算模型预测和真实目标值之间的平均平方差，用于在训练过程中评估模型性能。

```
# 损失函数
def mse_loss(y_true, y_pred):
    return ((y_true - y_pred) ** 2).mean()
```

1.4 前向传播

通过计算输入数据与权重的点积，加上偏置，然后应用 ReLU 激活函数，来完成从输入层到隐藏层再到输出层的数据流动。

```
# 前向传播
def forward_pass(x):
    z1 = x @ W1 + b1
    a1 = relu(z1)
    z2 = a1 @ W2 + b2
    return z2, z1, a1
```

1.5 反向传播和训练

反向传播 根据损失函数相对于网络参数的梯度来更新权重和偏置，使用梯度下降算法。在每次迭代中，计算损失函数相对于每个参数的导数（梯度），然后按照这些梯度的方向调整参数值，以减小损失。学习率 (learning_rate) 控制着参数更新的步长。

```
# 反向传播和训练
learning_rate = 0.001
epochs = 10000
for epoch in range(epochs):
    # 前向传播
    y_pred, z1, a1 = forward_pass(x_train)

    # 计算损失
    loss = mse_loss(y_train, y_pred)

    # 反向传播
    d_loss_y_pred = 2.0 * (y_pred - y_train) / y_train.size
    d_loss_W2 = a1.T @ d_loss_y_pred
    d_loss_b2 = d_loss_y_pred.sum(axis=0)
    d_loss_a1 = d_loss_y_pred @ W2.T
    d_loss_z1 = d_loss_a1 * relu_deriv(z1)
    d_loss_W1 = x_train.T @ d_loss_z1
    d_loss_b1 = d_loss_z1.sum(axis=0)

    # 参数更新
    W1 -= learning_rate * d_loss_W1
    b1 -= learning_rate * d_loss_b1
    W2 -= learning_rate * d_loss_W2
    b2 -= learning_rate * d_loss_b2
```

2. 拟合效果

和使用深度学习框架一样，我们同样使用相同的函数来拟合，比较一下效果：

通过可视化工具（如 matplotlib），我们可以将模型的预测结果与目标函数 sin(x)的实际值进行对比，从而直观地评估模型的拟合效果。

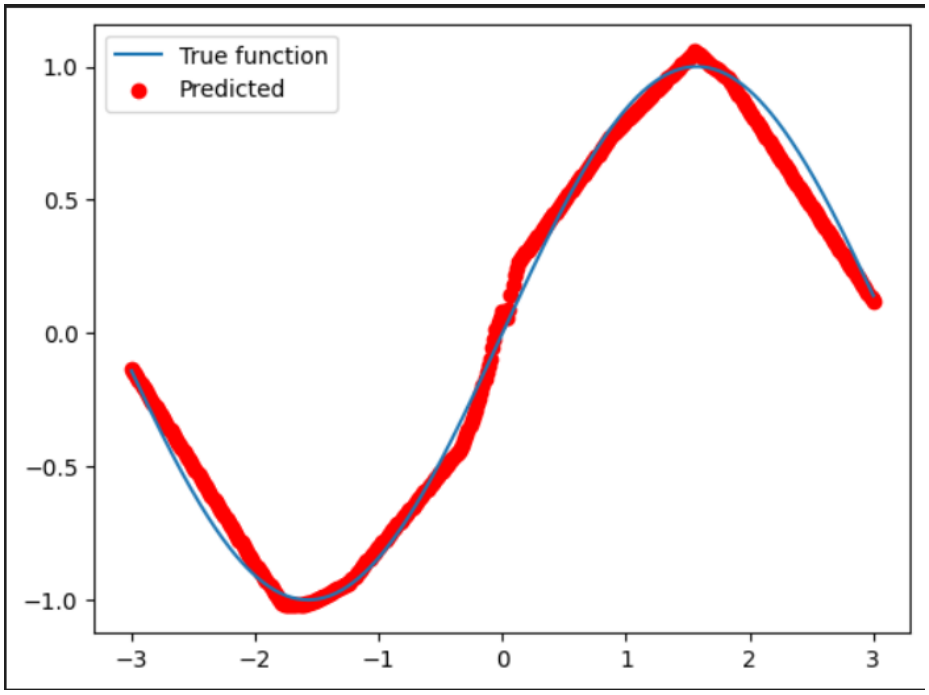

图 4 手写拟合 sinx 函数的图像

我们继续拟合另外的多项式函数观察效果：

比如说 x*x + 2*x + 5:

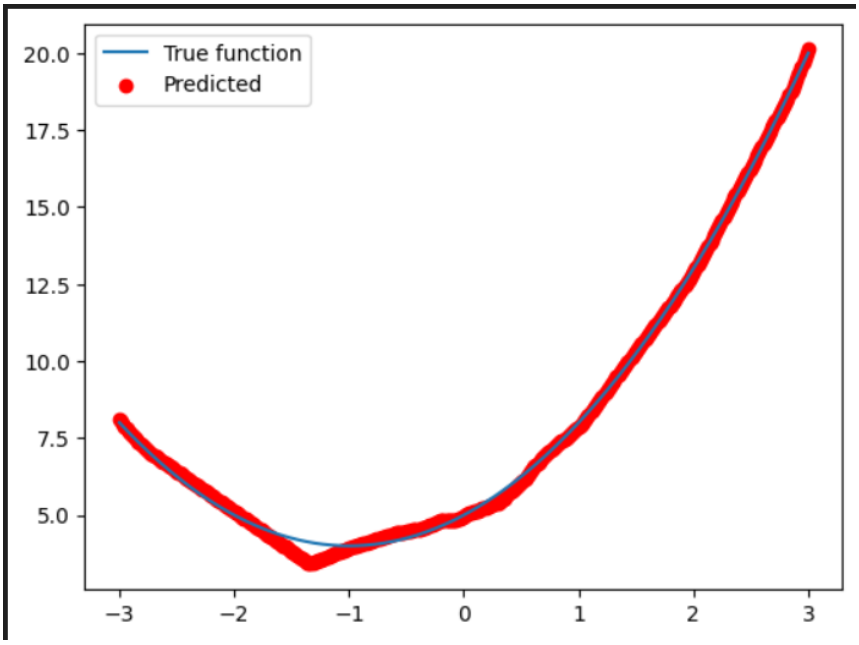

图 5 手写拟合 x*x + 2*x + 5 的图像

最后试一下线性函数 3*x + 4 的拟合效果：

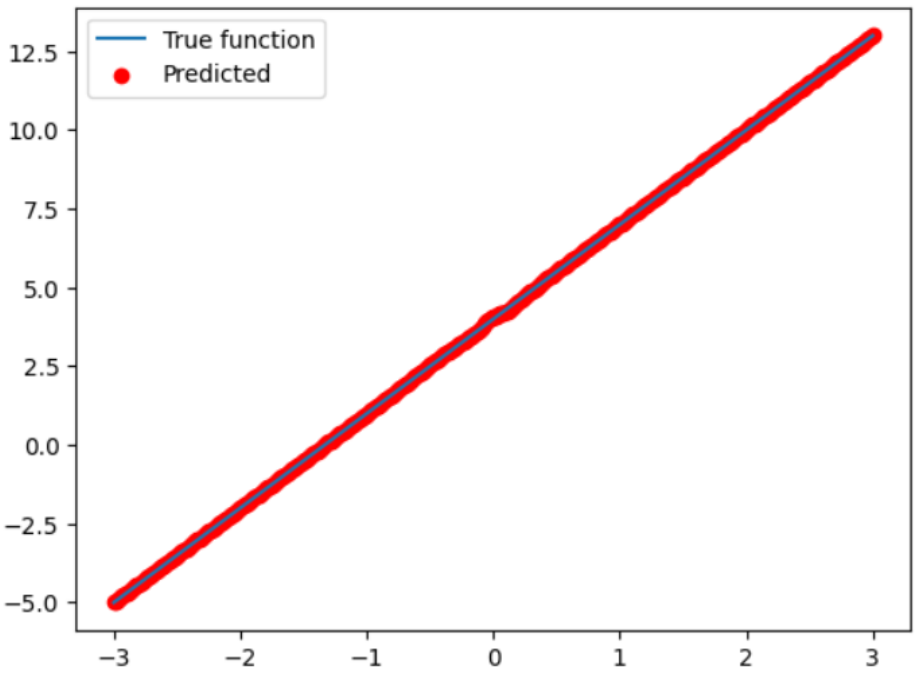

图 6 手写拟合 3*x + 4 的图像

综合比较可以发现总体来上说拟合效果还可以，当时比不上直接使用框架来对函数进行拟合，不过某种程度上确实称得上是可以拟合任何函数！加分项完成，代码也在项目中给出，以 ipynb 形式给出。