ReLU 网络拟合函数作业报告

2253372 康嘉玮

一、题目要求

理论和实践证明，一个两层的 ReLU 网络可以模拟任何函数[1]。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。

二、函数定义与网络构建

2.1 函数定义

我们选择函数 y=sin(x)作为模拟对象。

2.2 数据采集

随机选择 1000 个在[-5, 5)区间上均匀分布的数据作为自变量的值，以它们的正弦值作为函数值。以此我们得到一个训练集。再随机选择 100 个在[-5, 5)区间上均匀分布的数据作为自变量的值，得到一个大小为训练集 1/10 的测试集。由于自变量是在[-5, 5]上随机取值的，故能在较大程度上保证训练集与测试集并不相同。

2.3 模型描述

我们选择一个两层的神经网络，隐藏层大小为 64。下面代码变量名中的 h 表示隐藏层。

```
        self.Wh = np.random.randn(1, 64)
        self.bh = np.random.randn(64)
        self.W = np.random.randn(64, 1)
        self.b = np.random.randn(1)
```

隐藏层的激活函数选择 ReLU。下面的 myRelu 是自行编写的 ReLU 函数。

```
        self.zh = np.dot(x, self.Wh) + self.bh
        self.hidden = myRelu(self.zh) # h = relu(zh)
        self.z = np.dot(self.hidden, self.W) + self.b
```

损失函数定义为真实值与预测值的差值的平方和的平均数，也就是均方误差。

反向传播算法如下。

```
        dl_dz2 = 2 * (self.z - y) / x.shape[0]

        dl_dw = np.dot(self.hidden.T, dl_dz2)
        dl_db = np.sum(dl_dz2)

        dl_dh = np.dot(dl_dz2, self.W.T)
        dl_dz1 = dl_dh * myRelu_der(self.zh)

        dl_dwh = np.dot(x.T, dl_dz1)
        dl_dbh = np.sum(dl_dz1)

        self.W -= learning_rate * dl_dw
        self.b -= learning_rate * dl_db
        self.Wh -= learning_rate * dl_dwh
        self.bh -= learning_rate * dl_dbh
```

训练完成（10000 epochs）之后，用测试集进行测试，将测试集对应的点显示在一张图上，并与正弦函数的图象进行对比。此处用到了 matplotlib 库。

三、拟合效果

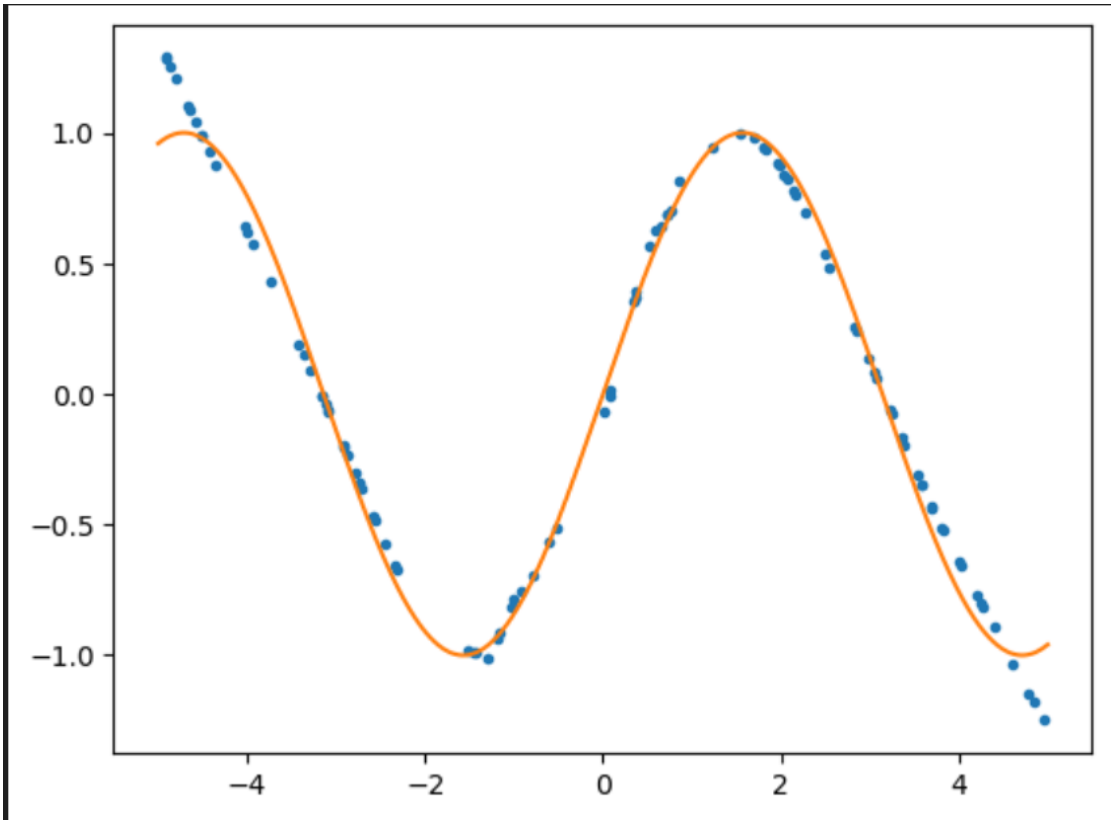
运行一次代码，观察拟合效果与最终的图象。下面是部分运行结果图。

```
test_dataset['x'] = np.random.rand(batch_size // 10, 1) * 10 - 5 # 产生[-5, 5)的随机数
test_dataset['y_est'] = model.test(test_dataset['x'])

plt.plot(test_dataset['x'], test_dataset['y_est'], '.') #画预测值的散点图

x = np.linspace(-5, 5, 10000)
y = np.sin(x)
plt.plot(x, y) # 画正弦函数的图象

plt.show()
```

✓ 8.1s

```
epoch100的训练损失:  0.14176799898957437
epoch200的训练损失:  0.08731867860983338
epoch300的训练损失:  0.07373239306821656
epoch400的训练损失:  0.0653237903399959
epoch500的训练损失:  0.05918276793669987
epoch600的训练损失:  0.052362847639332216
epoch700的训练损失:  0.04491496841718667
epoch800的训练损失:  0.03917610708956461
epoch900的训练损失:  0.03517141278770374
epoch1000的训练损失:  0.03240340901674445
epoch1100的训练损失:  0.030290394037977023
epoch1200的训练损失:  0.028235613904423616
epoch1300的训练损失:  0.02633801847158558
epoch1400的训练损失:  0.024930691338035223
epoch1500的训练损失:  0.023829448148630635
epoch1600的训练损失:  0.022913087820811476
epoch1700的训练损失:  0.022113716470457314
epoch1800的训练损失:  0.021369160407199644
epoch1900的训练损失:  0.020666849372503654
epoch2000的训练损失:  0.020010840641349056
epoch2100的训练损失:  0.019384925356283684
epoch2200的训练损失:  0.01879647585605678
epoch2300的训练损失:  0.018236005217074188
epoch2400的训练损失:  0.017703187355354855
epoch2500的训练损失:  0.01718534022583785
```

```
epoch8000的训练损失:  0.007477369332393201
epoch8100的训练损失:  0.007432442990524056
epoch8200的训练损失:  0.007389295892141273
epoch8300的训练损失:  0.007348366542403639
epoch8400的训练损失:  0.0073093527981138206
epoch8500的训练损失:  0.007272657287448885
epoch8600的训练损失:  0.00723748214045928
epoch8700的训练损失:  0.007204299090697294
epoch8800的训练损失:  0.007172744608996229
epoch8900的训练损失:  0.007142400854210641
epoch9000的训练损失:  0.007113454961332583
epoch9100的训练损失:  0.007085568871406656
epoch9200的训练损失:  0.007058406226645797
epoch9300的训练损失:  0.007032157642393782
epoch9400的训练损失:  0.007006748591817828
epoch9500的训练损失:  0.0069821318389134655
epoch9600的训练损失:  0.006958474450957053
epoch9700的训练损失:  0.006935755058117383
epoch9800的训练损失:  0.00691388033035955
epoch9900的训练损失:  0.006892712487591465
epoch10000的训练损失:  0.006872291135589239
```



由散点图可以得知，拟合的效果较好，大多数散点都大致在正弦曲线上。

四、收获

通过本次作业，我对 ReLU 和前馈神经网络的理解加深了许多。希望在日后的学习中，我可以学到深度学习和神经网络方面更多的知识。

参考文献：

[1] G. Cybenko. 1989. Approximation by superpositions of a sigmoidal function.

[2] K. Hornik, M. Stinchcombe, and H. White. 1989. Multilayer feedforward networks are universal approximators.
[3] Moshe Leshno, et al. 1993. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function
[4] Vinod Nair and Geoffrey E. Hinton. 2010. Rectified linear units improve restricted boltzmann machines.
[5] Xavier Glorot, Antoine Bordes, Yoshua Bengio. 2011. Deep Sparse Rectifier Neural Networks. PMLR 15:315-323.