两层的ReLU网络拟合任意函数

理论证明

对于任何一个连续函数f(x)，我们可以证明它可以通过它的无穷小部分的斜率之和来近似。因为ReLU 可以用来表示任意线，因此它们可以组成这些斜率，最终近似于函数 f(x)。

由于函数的参数是可学习的，因此神经网络会调整这些参数（权重和偏置），并近似函数。

代码实现

本次作业尝试使用两种方式完成两层的ReLU网络搭建，分别是使用Pytorch框架和numpy库。

函数定义

本次作业尝试两种函数，分别是y=sin(x)和y=x^2。定义代码如下：

```
x = np.linspace(-2*np.pi, 2*np.pi, 1000).reshape(-1, 1)
y = np.sin(x)
# y = x*
```

```
# generate training data
torch.manual_seed(0)
x = torch.linspace(-2*np.pi, 2*np.pi, 1000).view(-1, 1)
y = torch.sin(x)
```

数据采集

在numpy实现中，使用了numpy随机数进行数组切片，代码如下：

```
# shuffle the index
indices = np.random.permutation(x.shape[0])
x = x[indices]
y = y[indices]

# split
split_ratio = 0.8
split_idx = int(len(x) * split_ratio)
x_train, x_test = x[:split_idx], x[split_idx:]
y_train, y_test = y[:split_idx], y[split_idx:]
```

在pytorch实现中，使用了Dataloader类和torch.utils.data.random_split()方法制作DataLoader，代码如下：

```
# generate training data
torch.manual_seed(0)
x = torch.linspace(-2*np.pi, 2*np.pi, 1000).view(-1, 1)
y = torch.sin(x)

# shuffle the index
indices = np.random.permutation(len(x))
x = x[indices]
y = y[indices]

# convert to tensor
x_tensor = torch.tensor(x, dtype=torch.float32).view(-1, 1)
y_tensor = torch.tensor(y, dtype=torch.float32).view(-1, 1)

# generate dataset and dataloader
dataset = TensorDataset(x_tensor, y_tensor)
train_size = int(0.8 * len(dataset))
test_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size,
test_size])
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

模型描述

如下代码所述，在numpy和Pytorch的实现中，都分别定义了两层全连接层以及一层ReLU激活层，在numpy实现中手动完成了反向传播计算梯度以及梯度下降。

```python
# numpy implementation
import numpy as np
def relu(x):
    return np.maximum(0, x)
class Model():
    def __init__(self, input_size=1, hidden_size=512, output_size=1) -> None:
        self.W1 = np.random.randn(input_size, hidden_size)
        self.b1 = np.zeros(hidden_size)
        self.W2 = np.random.randn(hidden_size, output_size)
        self.b2 = np.zeros(output_size)
    def forward(self, x):
        self.input = x
        self.h1 = np.dot(x, self.W1)+self.b1
        self.r1 = relu(self.h1)
        self.o1 = np.dot(self.r1, self.W2)+self.b2
        return self.o1
    def backward(self, grad_y, lr=1e-5):
        # layer 2
        grad_W2 = np.dot(self.r1.T, grad_y)
        grad_b2 = np.sum(grad_y, axis=0)
        grad_h1 = np.dot(grad_y, self.W2.T) # for relu
        # relu
        grad_h1[self.h1<=0] = 0
        # layer 1
        grad_W1 = np.dot(self.input.T, grad_h1)
        grad_b1 = np.sum(grad_h1, axis=0)
        # gradient descent
        self.W1 -= lr * grad_W1
        self.b1 -= lr * grad_b1
        self.W2 -= lr * grad_W2
        self.b2 -= lr * grad_b2
```

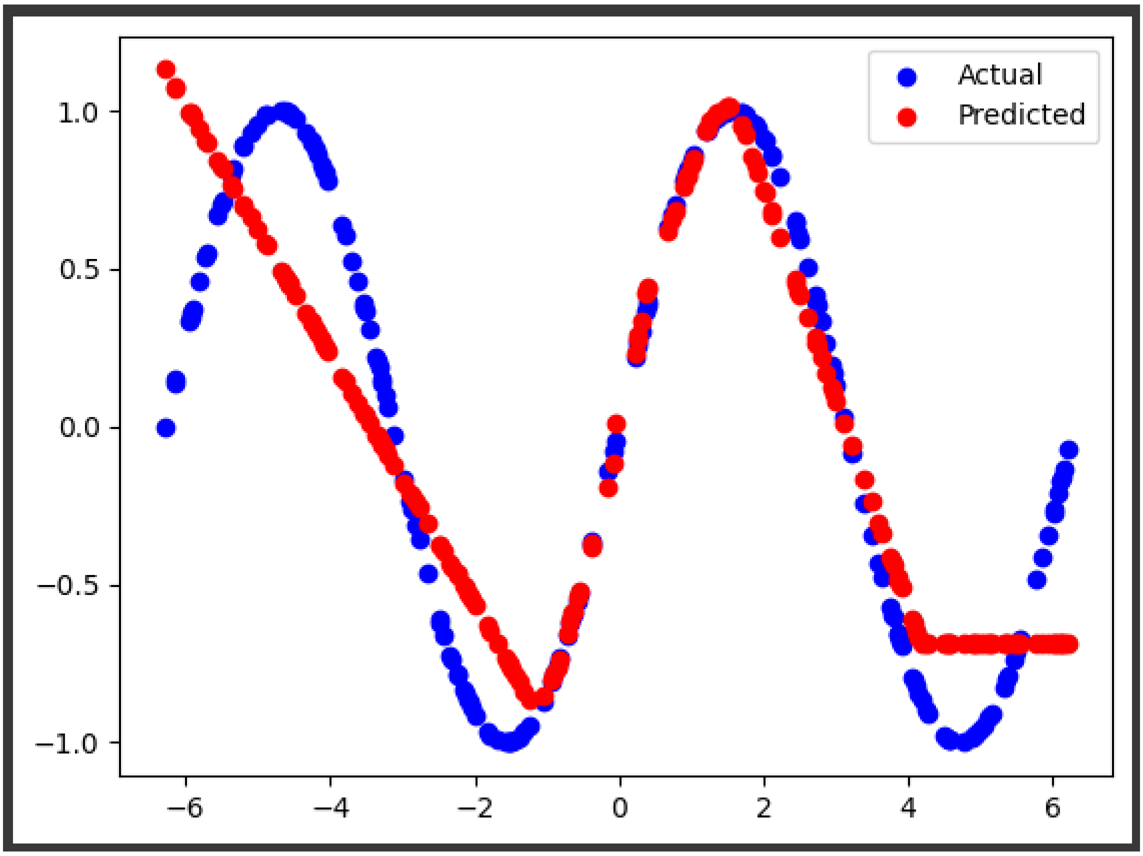
```python
# pytorch implementation
import torch.nn as nn
# define network
class TwoReLUNet(nn.Module):
    def __init__(self, input_size, hidden_size,  output_size):
        super(TwoReLUNet, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size, output_size)
    # only forward needed
    def forward(self, x):
        x = self.fc1(x)
        x = self.relu1(x)
        x = self.fc2(x)
        return x
```

拟合效果

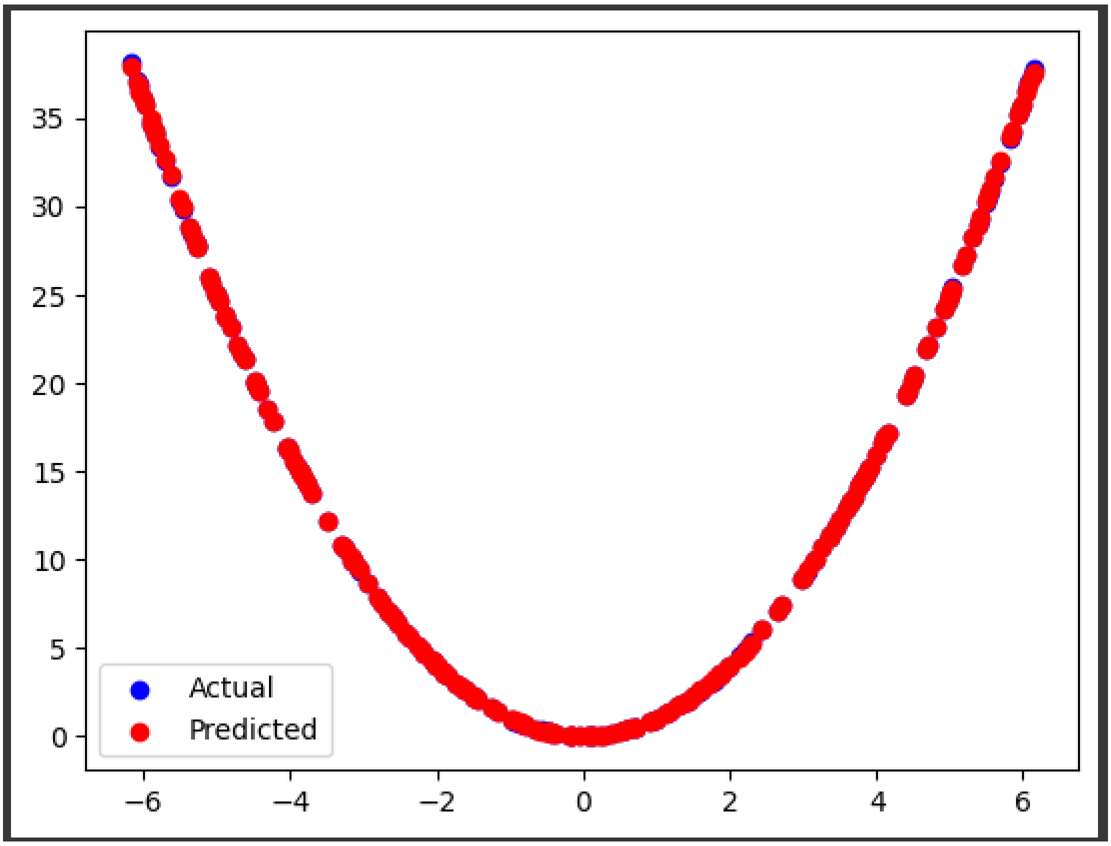
在调整学习率和迭代次数后，看到对于两个函数的拟合效果分别如下：

numpy

y=sin(x)

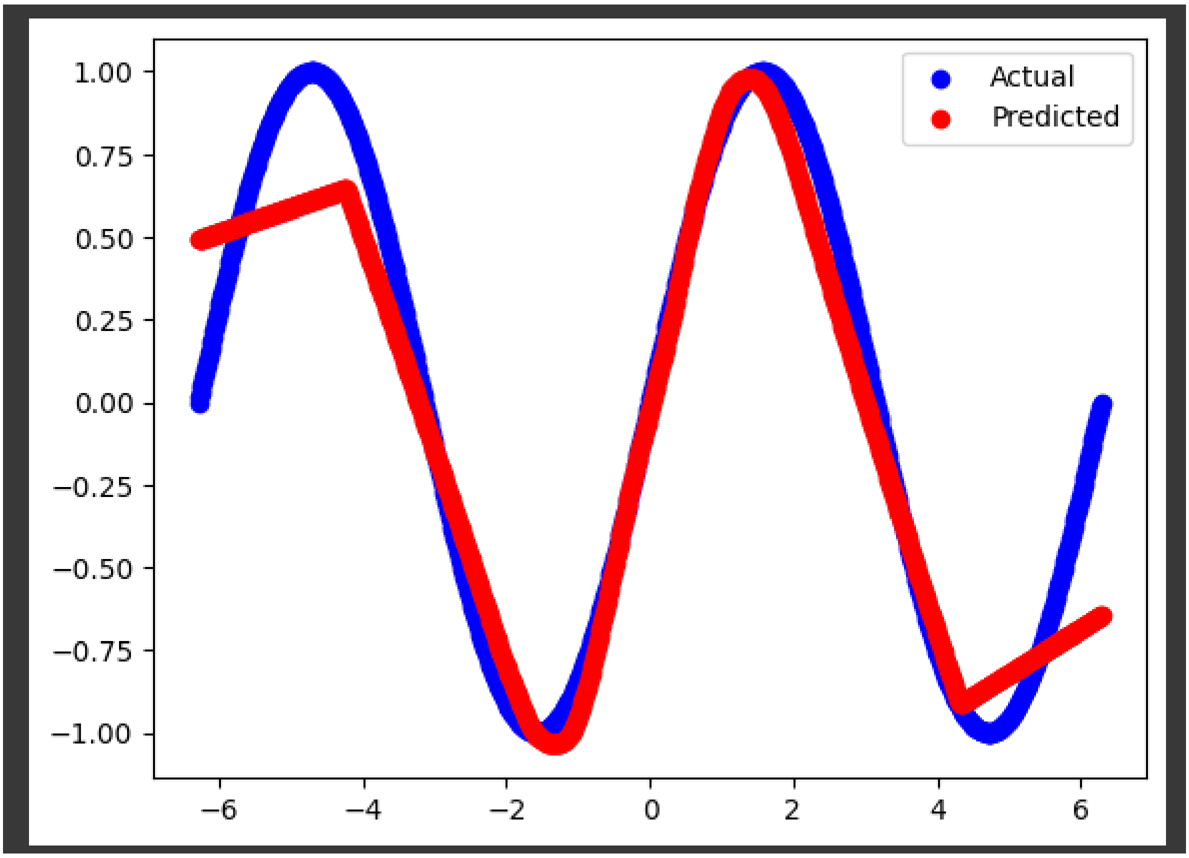


y=x^2



pytorch

y=sin(x)



y=x^2