lab1：前馈神经网络拟合复合函数

姓名：李晖茜

学号：SA22218131

一、实验要求

使用 `pytorch` 或者 `tensorflow` 手写一个前馈神经网络，用于近似以下函数

$$y = sinx + e^{-x}, x \in [0, 4\pi),$$

并研究网络深度、学习率、网络宽度、激活函数对模型性能的影响。

二、实验环境

torch	1.12.1
numpy	1.21.6
matplotlib	3.5.3

三、实验过程

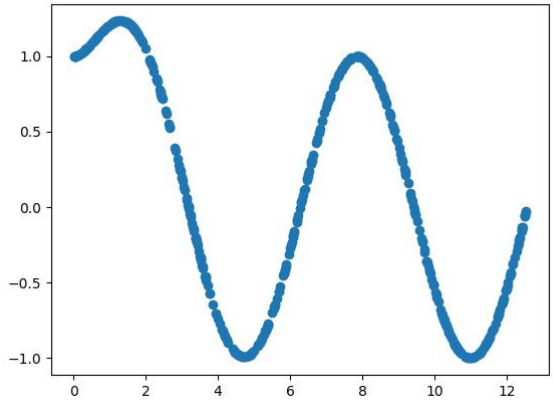
3.1 数据生成

使用`numpy`生成区间[0,4π)上的均匀数据样本作为训练、验证和测试数据，生成总样本`size=5000`个，然后随机打乱，随机种子设置为`torch.manual_seed(0)`，以确保每次生成的数据集是一样的。按照8:1:1的比例划分为训练集、测试集、验证集，最后返回`x_train, y_train, x_val, y_val, x_test, y_test`。代码如下：

```
def data(size):
    # 生成数据, size个等距, dim=1 1维的数据转换成2维
    x = torch.linspace(0,4*np.pi,size)
    x = torch.unsqueeze(x,dim=1)
    y = np.sin(x) + np.exp(-x)
    # 生成x,y; 按8:1:1划分
    x_train, x_val, x_test = torch.utils.data.random_split(x, [int(0.8*size),
int(0.1*size), int(0.1*size)],torch.manual_seed(0))
    y_train, y_val, y_test = torch.utils.data.random_split(y, [int(0.8*size),
int(0.1*size), int(0.1*size)],torch.manual_seed(0))
    x_train, x_val, x_test = x[x_train.indices], x[x_val.indices],
x[x_test.indices]
    y_train, y_val, y_test = y[y_train.indices], y[y_val.indices],
y[y_test.indices]

    # 将tensor置入Variable中
    x_train,y_train,x_val,y_val,x_test,y_test =
(Variable(x_train),Variable(y_train),
```

```
        Variable(x_val),Variable(y_val),Variable(x_test),Variable(y_test))

    # 数据可视化
    plt.scatter(x_test.data,y_test.data)
    # 或者采用如下的方式也可以输出x,y
    # plt.scatter(x.data.numpy(),y.data.numpy())
    plt.savefig("/home/lihuiqian/hw/lab1/fig/data.jpg")

    return x_train,y_train,x_val,y_val,x_test,y_test
```

其中数据可视化部分代码，在坐标上打印出生成的测试数据，如下：



3.2 网络搭建

搭建简单的前馈神经网络，由简单的输入层、隐藏层、输出层组成；输入层和输出层的神经元个数固定为1，隐藏层大小和隐藏层深度通过参数传递。激活函数可指定为`relu`，`tanh`，`sigmoid`，`leakyrelu`。搭建网络函数如下：

```
# 搭建网络FNN，n个全连接层组成的隐藏层
class Net(nn.Module): # 继承nn.Module
    def __init__(self,n_input,n_hidden,n_output,num_layers,activation):
        super(Net,self).__init__() # 获得Net类的父类的构造方法
        # 定义每层结构形式，num_layers个隐藏层
        activation = activation.lower()
        act_map = {
            'relu': nn.ReLU,
            'tanh': nn.Tanh,
            'sigmoid': nn.Sigmoid,
            'leakyrelu': nn.LeakyReLU,
        }
        self.fc_layers = nn.ModuleList()
        self.activations = nn.ModuleList()
        self.fc_layers.append(nn.Linear(n_input,n_hidden)) # 第一个隐藏层
        for i in range(num_layers-1):
            self.fc_layers.append(nn.Linear(n_hidden,n_hidden))
            if i < num_layers - 1:
                self.activations.append(act_map[activation]())
        self.fc_layers.append(nn.Linear(n_hidden,n_output)) # 预测层
    # 将各层的神经元搭建成完整的神经网络的前向通路
    def forward(self,input):
        x = input
        for i, layer in enumerate(self.fc_layers):
            x = layer(x)
```

```
            if i < len(self.activations):
                x = self.activations[i](x) # 对隐藏层激活
        return x
```

3.3 网络训练

定义训练函数`train(hidde_size,num_layers,activation,epoch,batch_size,lr,x,y,x_val,y_val)`，其中可传递参数的含义分别是隐藏层大小，隐藏层层数，激活函数，迭代轮次，批大小，学习率，训练集x，训练集y，验证集x，验证集y。通过`DataLoader`加载数据，使用SGD优化器，损失函数使用`MSELoss`，具体函数如下：

```
def train(hidde_size,num_layers,activation,epoch,batch_size,lr,x,y,x_val,y_val):
    # 定义神经网络，打印输出net的结构（隐藏层hidde_size个节点）
    net = Net(1,hidde_size,1,num_layers,activation)
    print(net)

    # 加载数据
    # batch_size = 32
    dataset = TensorDataset(x,y)
    dataloader = DataLoader(dataset, batch_size, shuffle=True)
    valset = TensorDataset(x_val,y_val)
    validation_data = DataLoader(valset, 1, shuffle=True)

    # 优化器和损失函数
    optimizer = torch.optim.SGD(net.parameters(),lr)
    loss_func = torch.nn.MSELoss()

    # 网络训练过程。随机梯度下降，设置学习率为0.1，迭代epoch次
    # epoch = 1000
    mean_losses,val_losses = [],[]
    for t in range(epoch):
        train_loss = 0  # 统计loss
        for x, y in dataloader:
            prediction = net(x) # 数据x传给net，输出预测值
            loss = loss_func(prediction,y) # 计算误差，注意参数顺序
            optimizer.zero_grad()  # 清空上一步的更新参数值
            loss.backward()        # 误差反向传播
            optimizer.step()       # 计算得到的更新值赋给net.parameters()
            train_loss += loss.item() # 统计loss

        with torch.no_grad(): # 计算验证集loss
            val_loss = 0    # 统计loss
            for val_x, val_y in validation_data:
                out = net(val_x)
                loss = loss_func(out, val_y)
                val_loss += loss.item()
            val_loss = round(val_loss/len(validation_data),6)

        mean_loss = round(train_loss/len(dataloader),6)
        print('epoch:', t, ', loss:', mean_loss,', val_loss:', val_loss)
        mean_losses.append(mean_loss)
        val_losses.append(val_loss)
    plt.cla()
    plt.plot(range(epoch), mean_losses, 'r-', lw=2)
    plt.plot(range(epoch), val_losses, 'b-', lw=2)
```

```
    plt.xlabel('epoches')
    plt.ylabel('Train loss (red), Val loss (blue)')
    plt.savefig("/home/lihuiqian/hw/lab1/fig/train.jpg")
    return net
```

3.4 调参分析

训练过程中，通过绘制训练集和验证集的loss曲线，调整超参数`epoch`，`batch_size`等，具体调整过程见实验结果4.1比较相关超参数。

3.5 测试性能

编写测试函数，输出预测值和真实值之间的`MSELoss`，同时，绘制真实值和预测值，用来观察网络拟合复合函数的效果。函数如下：

```
def test(net,x,y):
    dataset = TensorDataset(x,y)
    dataloader = DataLoader(dataset, batch_size=len(dataset), shuffle=False)
    loss_func = torch.nn.MSELoss()
    for x, y in dataloader:
        prediction = net(x) # 数据x传给net，输出预测值
        loss = loss_func(prediction,y) # 计算误差，注意参数顺序
        print(loss.item())
        # 可视化训练过程
        plt.cla()
        plt.scatter(x.data.numpy(), y.data.numpy())
        plt.scatter(x.data.numpy(), prediction.data.numpy(), c='r')
    plt.suptitle('test: Loss = %.4f'% loss.data, fontsize = 20)
    plt.ioff()
    plt.xlabel('x')
    plt.ylabel('y or pred')
    plt.savefig("/home/lihuiqian/hw/lab1/fig/test.jpg")
    return 0
```

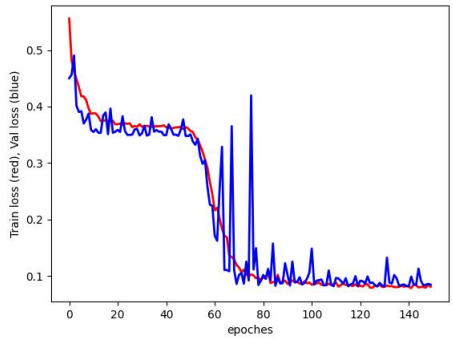
四、实验结果

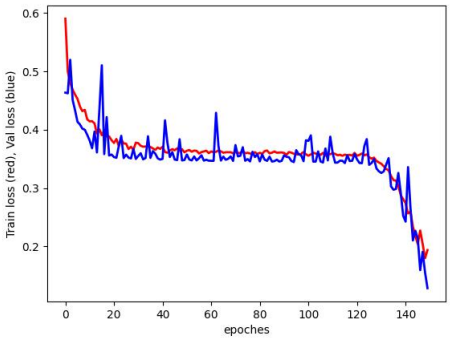
初始训练参数如下：

```
hidde_size = 20      # 隐藏层节点数(大小，宽度)
num_layers = 2       # 隐藏层数量(深度)
activation = 'relu' # 激活函数
epoch = 100          # 训练轮次
batch_size = 16      # 批大小
lr = 0.01            # 学习率
net =
train(num_depth,num_layers,activation,epoch,batch_size,lr,x_train,y_train,x_val,
y_val)
```

4.1 比较相关超参数

首先，通过绘制训练过程中训练集和验证集的loss曲线，调整参数`epoch`和`batch_size`，使网络训练达到一个相对较好的程度，之后的训练保持此参数。





左图为`epoch=150，batch_size=16`，右图为`epoch=150，batch_size=32`。图中红色曲线为训练集loss，蓝色曲线为验证集loss。观察得到在`batch_szie=16`时，`epoch`达到100~150时loss曲线趋于稳定；`batch_szie=32`时loss值下降更慢，但是更稳定，可能需要更多epoch，考虑后以下实验均选择`epoch=100，batch_size=16`。

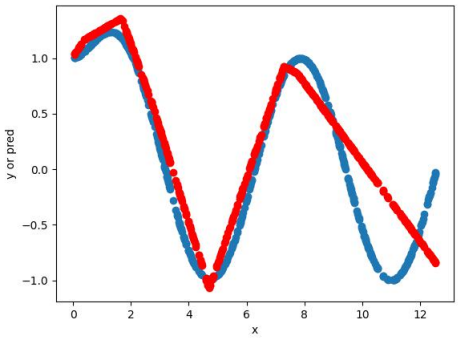
4.2 比较网络深度

固定其他参数，改变`num_layers`，分别设置为如下参数，独立的训练并测试得到最终的MSELoss。其他参数设置为`hidde_size = 20`，`activation = 'relu'`，`lr = 1e-2`。

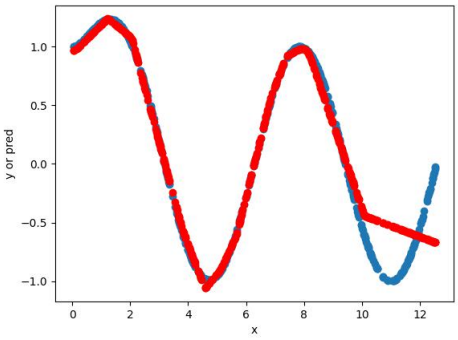
```
num_layers = 2      # 隐藏层数量(深度)
num_layers = 3      # 隐藏层数量(深度)
num_layers = 4      # 隐藏层数量(深度)
num_layers = 5      # 隐藏层数量(深度)
```

验证集上的拟合效果和loss值如下图：其中红色的点为预测值，蓝色为真实值。**注意：以下结果均为验证集上的测试结果。**

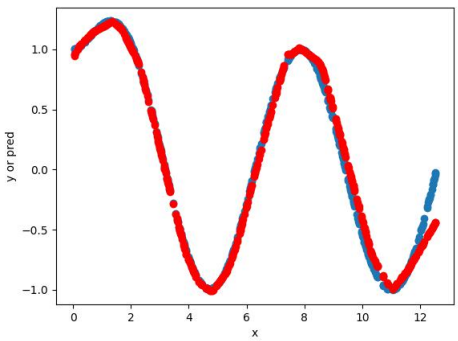
test: num_layers = 2, Loss = 0.0934



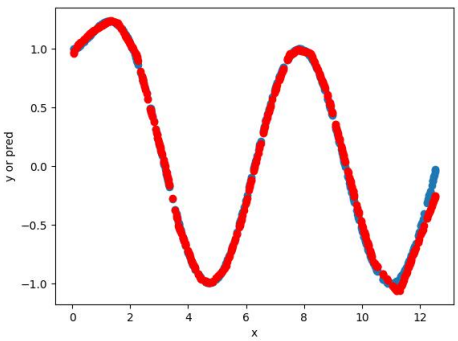
test: num_layers = 3, Loss = 0.0261



test: num_layers = 4, Loss = 0.0071



test: num_layers = 5, Loss = 0.0019



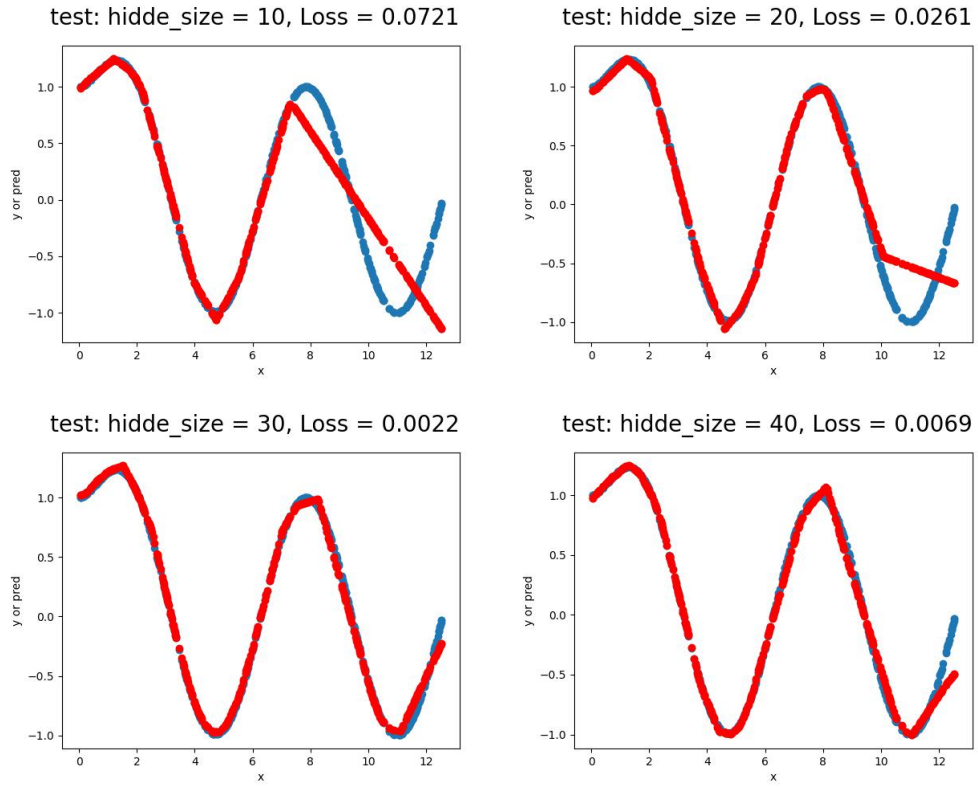
可以注意到当深度较大时，拟合效果会更好，在`num_layers=5`时，loss值最小。

4.3 比较网络宽度

固定其他参数，改变`hidde_size`，分别设置为如下参数，独立的训练并测试得到最终的MSELoss。其他参数设置为`num_layers = 3`，`activation = 'relu'`，`lr = 1e-2`。

```
hidde_size = 10      # 隐藏层节点数(大小，宽度)
hidde_size = 20      # 隐藏层节点数(大小，宽度)
hidde_size = 30      # 隐藏层节点数(大小，宽度)
hidde_size = 40      # 隐藏层节点数(大小，宽度)
```

验证集上的拟合效果和loss值如下图：其中红色的点为预测值，蓝色为真实值。



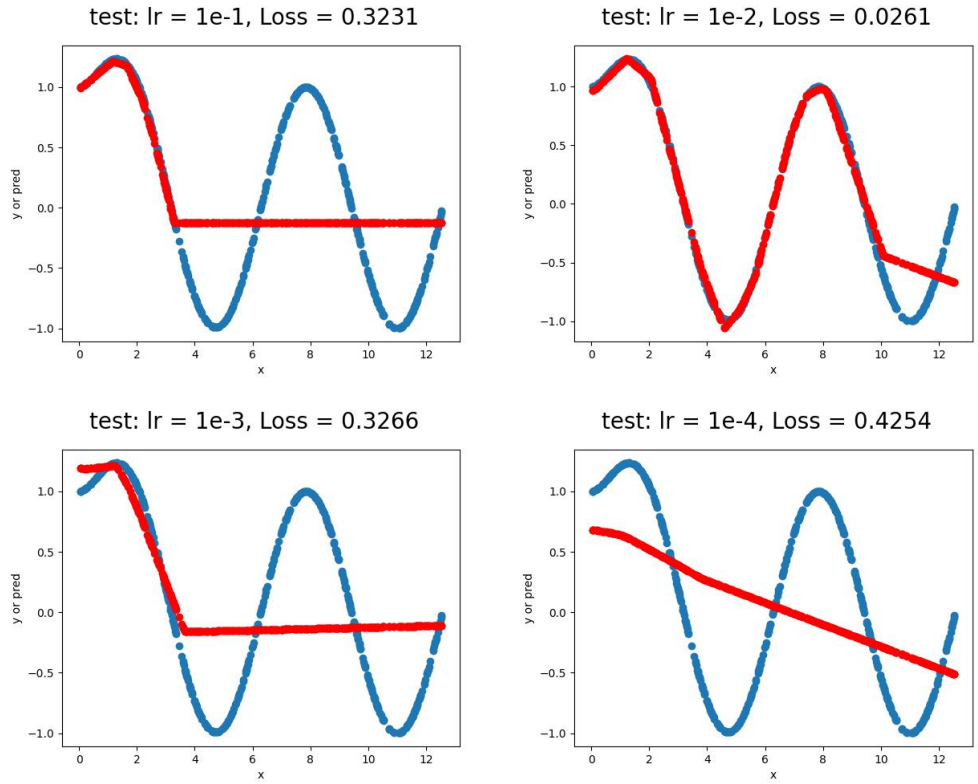
随着网络宽度的增加，模型的拟合效果趋于变好，但当网络过宽时，模型的性能也会下降，在`hidde_size=30`时，loss值最小。

4.4 比较学习率

固定其他参数，改变`lr`，分别设置为如下参数，独立的训练并测试得到最终的MSELoss。其他参数设置为`num_layers = 3`，`hidde_size = 20`，`activation = 'relu'`。

```
lr = 1e-1            # 学习率
lr = 1e-2            # 学习率
lr = 1e-3            # 学习率
lr = 1e-4            # 学习率
```

验证集上的拟合效果和loss值如下图：其中红色的点为预测值，蓝色为真实值。



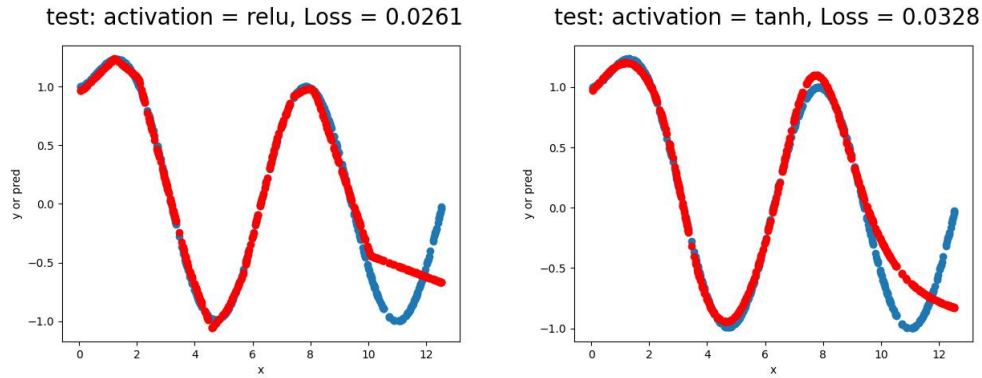
随着学习率慢慢变大，模型性能先变好后变差。学习率过大时，参数更新步长较大，导致目标函数波动较大，使得收敛速度变慢；而学习率太小时，参数更新步长较小，收敛速度较慢。

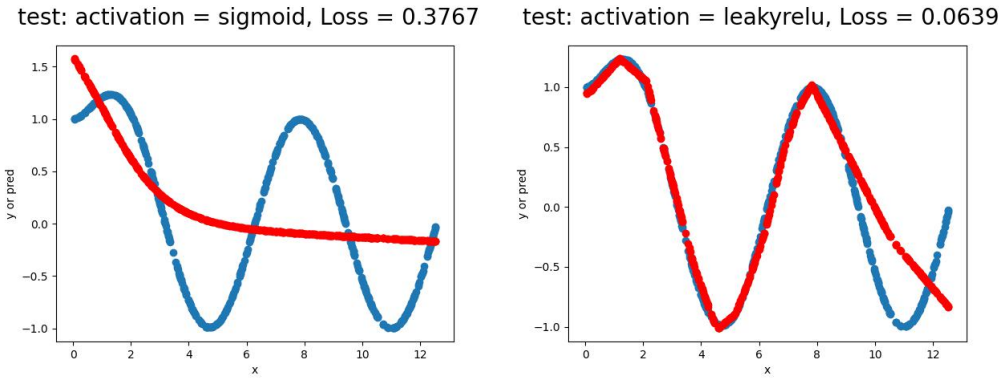
4.5 比较激活函数

固定其他参数，改变`activation`，分别设置为如下参数，独立的训练并测试得到最终的MSELoss。其他参数设置为`num_layers = 3`，`hidde_size = 20`，`lr = 1e-2`。

```
activation = 'relu' # 激活函数
activation = 'tanh' # 激活函数
activation = 'sigmoid' # 激活函数
activation = 'leakyrelu' # 激活函数
```

验证集上的拟合效果和loss值如下图：其中红色的点为预测值，蓝色为真实值。

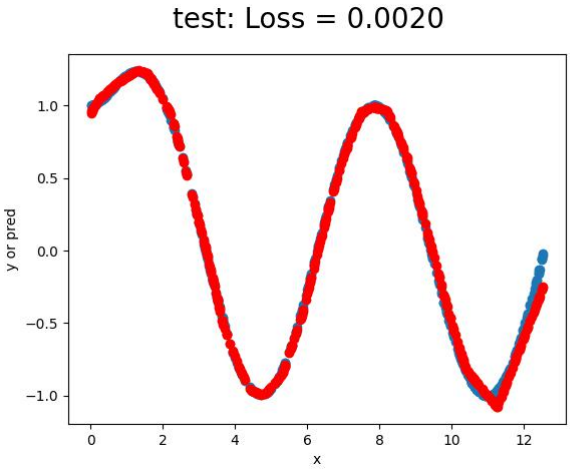




观察可知，sigmoid的拟合效果最差；relu系的激活函数比较适合此任务，而leakyrelu又比relu的效果差一些；tanh比起其他拟合的更加平滑。

五、实验总结

本实验中，基于pytorch初步实现了前馈神经网络拟合复合函数，其中参数设置为`num_layers = 5`，`hidde_size = 20`，`activation = 'relu'`，`lr = 1e-2`，`epoch=100`，`batch_size=16`时，验证集上函数拟合效果最好。使用该参数在**测试集**上进行测试（唯一一次在测试集上的测试），loss为`0.001974601997062564`，拟合效果如下图：



通过不同参数的比较，学习到了网络的深度、宽度、学习率以及激活函数对神经网络性能的影响，每个参数都不宜过大或者过小，需要调整到合适的值。对于网络的深度和宽度，越宽的网络收敛速度会更快，但是会引发梯度消失的问题；对于学习率，过大过小都会造成收敛变慢；而激活函数，relu及其变种对网络的正面效果更佳。