实验一

SA23011253 任永文

实验要求

使用 pytorch 或者 tensorflow 手写一个前馈神经网络，用于近似函数：

$$y = \sin(x), \quad x \in [0, 2\pi)$$

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。

实验步骤

1. **网络框架**：要求选择 pytorch 或 tensorflow 其中之一，依据官方网站的指引安装包。若你需要使用 GPU，可能还需安装 CUDA 驱动。本次实验仅利用 CPU 也可以完成，但仍强烈推荐大家安装 GPU 版本，以满足后续实验需求。
2. **数据生成**：本次实验的数据集仅需使用程序自动生成，即在 $[0, 2\pi)$ 范围内随机 sample 样本作为 x 值，并计算 y 值。要求生成三个**互不相交**的数据集分别作为**训练集、验证集、测试集**。训练只能在训练集上完成，实验调参只能在验证集上完成。
3. **模型搭建**：采用 pytorch 或 tensorflow 所封装的 module 编写模型，例如 torch.nn.Linear(), torch.nn.Relu() 等，无需手动完成底层 forward、backward 过程。
4. **模型训练**：将生成的训练集输入搭建好的模型进行前向的 loss 计算和反向的梯度传播，从而训练模型，同时也建议使用网络框架封装的 optimizer 完成参数更新过程。训练过程中记录模型在训练集和验证集上的损失，并绘图可视化。
5. **调参分析**：将训练好的模型在验证集上进行测试，以 **Mean Square Error(MSE)** 作为网络性能指标。然后，对网络深度、学习率、网络宽度、激活函数等模型超参数进行调整，再重新训练、测试，并分析对模型性能的影响。
6. **测试性能**：选择你认为最合适的（例如，在验证集上表现最好的）一组超参数，重新训练模型，并在测试集上测试（注意，这理应是你的实验中**唯一**一次在测试集上的测试），并记录测试的结果（MSE）。

实验提交

本次实验截止日期为 **11 月 1 日 23:59:59**，提交到邮箱 ustcdl2023@163.com ，具体要求如下：

1. 全部文件打包在一个压缩包内，压缩包命名为 学号- 姓名 - exp1.zip
2. 代码仅包含 .py 文件，请勿包含实验中间结果（例如中间保存的数据集等），如果有多个文件，放在 src/ 文件夹内。
3. 代码中提供一个可以直接运行的并输出结果的 **main.py**，结果包括训练集损失、验证集损失随 epoch 改变的曲线（保存下来）和测试集的 MSE。

4. 代码中提供一个描述所有需依赖包的 requirements.txt，手动列入代码中用到的所有非标准库及版本或者使用 `pip freeze > requirements.txt` 命令生成。
5. 实验报告要求 pdf 格式，要求包含姓名、学号。内容包括简要的**实验过程**和**关键代码**展示，对超参数的**实验分析**，最优超参数下的训练集、验证集**损失曲线**以及测试集上的**实验结果。**

网络深度、学习率、网络宽度、激活函数

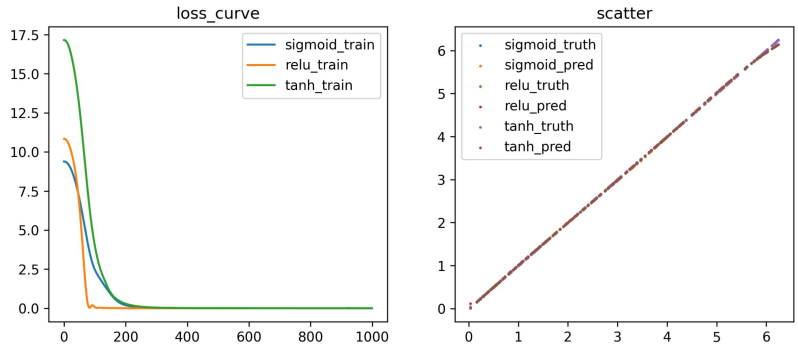
实验设计

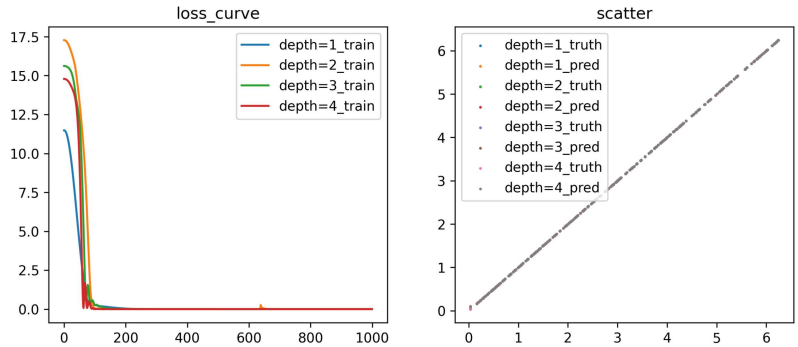
实验整体设置为控制变量法，基准参数为width=5,depth=1,lr=0.1,activation=relu

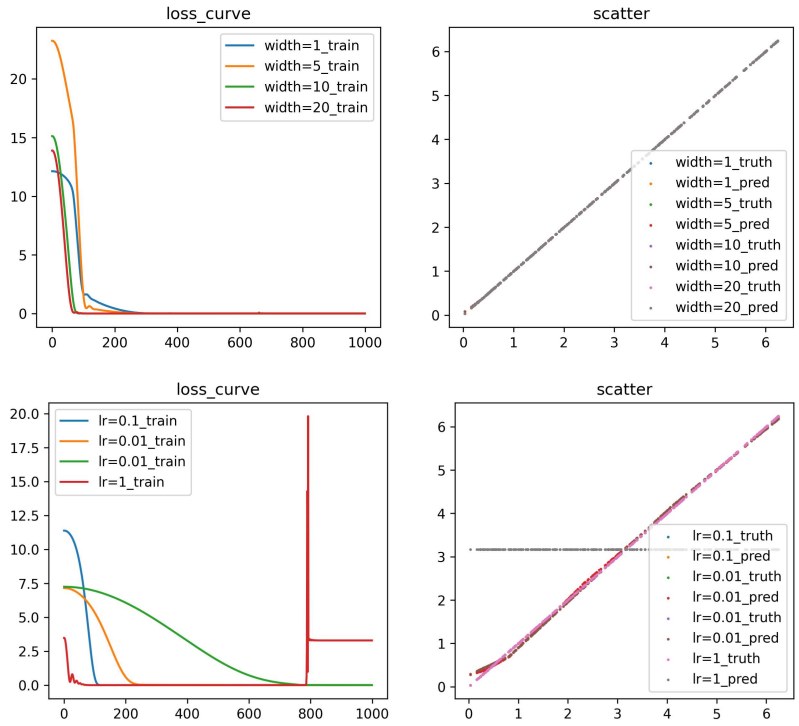
- 激活函数：设计激活函数分别为relu,sigmoid,tanh对比
- 网络深度：设计网络深度分别为1,2,3,4对比
- 网络宽度：设计网络宽度分别为1,5,10,20对比
- 学习率：设计学习率分别为1,0.1,0.01,0.001对比

实验结论

- 网络深度：一般情况下深度越大网络性能越好，但是如果宽度较小增加深度对模型的改进有限
- 学习率：学习率较低时学习速度较慢，但是学习率过高时可能找不到最优值
- 网络宽度：一般情况下宽度越大网络参数越多性能越好
- 激活函数：经过对比relu函数表现效果更好
- 最终获得的最优参数结果为：width=5,depth=1,lr=0.1,activation=relu







In [ ]:
```
import torch
import numpy as np
import torch.nn as nn
import torch.optim as optim
import matplotlib.pyplot as plt
from torch.optim.lr_scheduler import LambdaLR
```

1. 数据生成

In [ ]:
```
x = np.random.uniform(0, 2*np.pi, size=(3000,1))
y = np.sin(x)

x_train = torch.from_numpy(x[0 : 2400]).float()
x_val = torch.from_numpy(x[2400 : 2700]).float()
x_test = torch.from_numpy(x[2700 : 3000]).float()

y_train = torch.from_numpy(x[0 : 2400]).float()
y_val = torch.from_numpy(x[2400 : 2700]).float()
y_test = torch.from_numpy(x[2700 : 3000]).float()
```

2. 模型搭建

In [ ]:
```
class Net(nn.Module):
    def __init__(self, activation = torch.sigmoid, layers = [1, 20, 1]):
        super(Net, self).__init__()
        self.activation = activation
        self.num_layers = len(layers)-1
        self.fctions = nn.ModuleList()
        for i in range(self.num_layers):
            self.fctions.append(nn.Linear(layers[i], layers[i+1]))

    def forward(self, x):
        for i in range(self.num_layers-1):
```

```
            x = self.activation(self.fctions[i](x))
        x = self.fctions[-1](x)
        return x
```

3. 模型训练

In [ ]:

```
# Define a lambda function that returns the learning rate multiplier based on th
def lr_lambda(current_step):
    warmup_steps = 1000
    if current_step < warmup_steps:
        return float(current_step/warmup_steps)
    else:
        return 1.0

def Training(traindata , valdata, layers, activation, lr):
    # create the network, optimizer, and loss function
    net = Net(activation = activation, layers = layers)
    optimizer = optim.Adam(net.parameters(), lr=lr, betas=(0.9, 0.999), eps=1e-0
    loss_func = nn.MSELoss()
    # Create a learning rate scheduler that uses the lambda function
    scheduler = LambdaLR(optimizer, lr_lambda)
    # train the network
    x_train, y_train = traindata
    x_val, y_val = valdata
    loss_train = []
    loss_val = []
    for i in range(1000):
        optimizer.zero_grad()
        y_pred = net(x_train)
        loss_t = loss_func(y_pred, y_train)
        loss_t.backward()
        optimizer.step()
        scheduler.step()
        loss_train.append(loss_t.item())
        y_vpred = net(x_val)
        loss_v = loss_func(y_vpred, y_val)
        loss_val.append(loss_v.item())

    loss_curve = [loss_train, loss_val]
    return net, loss_curve
```

4. 性能测试

In [ ]:

```
def Test(activation=torch.sigmoid,depth=1,width=1,lr=0.1):
    layers = [1]
    for i in range(depth):
        layers.append(width)
    layers.append(1)
    net, loss_curve = Training((x_train, y_train), (x_val, y_val), layers, activ
    t = len(loss_curve[0])
    y_tpred = net(x_test)
    mse = nn.MSELoss()(y_tpred, y_test)
    print(f"训练集mse: {loss_curve[0][-1]}\t验证集mse: {loss_curve[1][-1]}\t测试
    return range(t), loss_curve, x_test, y_test, y_tpred
```

5. 调参分析

In [ ]:

```
def plotter(title,p):
    fig, axs = plt.subplots(1, 2, figsize=(10, 4), dpi=300)
    axs[0].set_title('loss_curve')
    axs[1].set_title('scatter')
    legend0 = []
    legend1 = []
    for i in range(len(title)):
        axs[0].plot(p[i][0], p[i][1][0])
        legend0.extend([title[i]+'_train'])
        axs[1].scatter(p[i][2], p[i][3], s=1)
        axs[1].scatter(p[i][2], p[i][4].detach().numpy(), s=1)
        legend1.extend([title[i]+'_truth',title[i]+'_pred'])
    axs[0].legend(legend0)
    axs[1].legend(legend1)
    plt.show()

# 损失函数对比
p11 = Test(activation=torch.sigmoid,depth=1,width=5,lr=0.1)
p12 = Test(activation=torch.relu,depth=1,width=5,lr=0.1)
p13 = Test(activation=torch.tanh,depth=1,width=5,lr=0.1)
plotter(['sigmoid','relu','tanh'],[p11,p12,p13])

# 深度对比
p21 = Test(activation=torch.relu,depth=1,width=5,lr=0.1)
p22 = Test(activation=torch.relu,depth=2,width=5,lr=0.1)
p23 = Test(activation=torch.relu,depth=3,width=5,lr=0.1)
p24 = Test(activation=torch.relu,depth=4,width=5,lr=0.1)
plotter(['depth=1','depth=2','depth=3','depth=4'],[p21,p22,p23,p24])

# 宽度对比
p31 = Test(activation=torch.relu,depth=1,width=1,lr=0.1)
p32 = Test(activation=torch.relu,depth=1,width=5,lr=0.1)
p33 = Test(activation=torch.relu,depth=1,width=10,lr=0.1)
p34 = Test(activation=torch.relu,depth=1,width=20,lr=0.1)
plotter(['width=1','width=5','width=10','width=20'],[p31,p32,p33,p34])

# 学习率对比
p41 = Test(activation=torch.relu,depth=1,width=5,lr=0.1)
p42 = Test(activation=torch.relu,depth=1,width=5,lr=0.01)
p43 = Test(activation=torch.relu,depth=1,width=5,lr=0.001)
p44 = Test(activation=torch.relu,depth=1,width=5,lr=1)
plotter(['lr=0.1','lr=0.01','lr=0.01','lr=1'],[p41,p42,p43,p44])
```