Assignment1：对给定值曲线进行拟合

作业要求

以给出的一组曲线值作为训练集，另一组值作为测试集，寻找MSE最低的拟合曲线

使用方法

本次作业尝试了多项式拟合（PR）和多层感知机拟合（MLP），在操作过程中发现多项式拟合存在效果差、收敛效率低的情况，故最终采用MLP拟合方法

Pytorch实现

准备

首先进行准备工作，包括库的引用和数据的读取以及处理

```
import time
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from IPython.display import clear_output
from scipy.interpolate import interp1d
# 设置Matplotlib的字体
plt.rcParams['font.family'] = 'Microsoft YaHei'  # 使用SimHei字体（黑体）

# 设置train值
train = pd.read_excel('complex_nonlinear_data.xlsx')
test = pd.read_excel('new_complex_nonlinear_data.xlsx')
# 获取数据
x_train = np.array(train.iloc[:, 0])  # 第一列为x值
y_train = np.array(train.iloc[:, 1])  # 第二列为y值
x_train = torch.tensor(x_train, dtype=torch.float32)
y_train = torch.tensor(y_train, dtype=torch.float32)

x_test = np.array(test.iloc[:, 0])  # 第一列为x值
y_test = np.array(test.iloc[:, 1])  # 第二列为y值
x_test = torch.tensor(x_test, dtype=torch.float32)
y_test = torch.tensor(y_test, dtype=torch.float32)
```

模型建立

再对使用的两个方法的module进行定义，采用`nn.Module`的父类来定义，通过`__init__`和`forward`的方法对两种模型进行定义。模型内部包括线性层以及非线性层，MLP中还包含dropout部分防止过拟合的发生。其中MLP部分和GR部分是单个数据的输入，即一个x值输入。

```
class PR(nn.Module):
    def __init__(self, degree):
```

```
        super(PR, self).__init__()
        self.degree = degree
        self.linear = nn.Linear(self.degree + 1, 1)  # 加1是因为包含常数项
        self.nolinear = nn.Sigmoid()

    def forward(self, x):
        # 转换张量格式，将输入数据转换为多项式
        x = x.reshape(-1, 1)
        x_poly = x ** torch.arange(self.degree + 1, dtype=torch.float32,
device=x.device).unsqueeze(0)
        x_poly = self.linear(x_poly).reshape(-1)
        return x_poly

class MLP(nn.Module):
    def __init__(self):
        super(MLP, self).__init__()
        self.linear1 = nn.Linear(1, 16)
        self.linear2 = nn.Linear(16, 32)
        self.linear3 = nn.Linear(32, 1)
        self.nolinear = nn.Sigmoid()
        self.dropout = nn.Dropout(p=0.1)  # 添加 dropout，丢弃概率为 0.9

    def forward(self, x):
        x = x.reshape(-1, 1)
        x = self.linear1(x)
        x = self.nolinear(x)
        x = self.linear2(x)
        x = self.nolinear(x)
        x = self.linear3(x)
        return x.reshape(-1)
```

模型初始化

定义好需要的工具后，进行模型的初始化准备开始训练，同时建好`losses_train`和`losses_val`两个列表进行存储

```
# 初始化多项式拟合模型
degree = 6  # 多项式的阶数
model1 = PR(degree)

# 初始化多层感知机模型
model2 = MLP()
# 定义损失函数和优化器
loss_fonc = nn.MSELoss()
optimizer = optim.Adam(model2.parameters(), lr=0.1)  # 默认使用MLP模型
losses_train = []
losses_val = []
```

训练模型及验证和测试

进行模型的训练过程，在训练过程中，每个epoch中将给出的训练数据按交叉验证方式分成0.8的训练集和0.2的验证集，并通过给种子赋值为epoch值保证每次分开的数据独立不重复。同时记录在每个epoch下train和val的损失值到`losses_train`和`losses_val`两个列表，并找出val集上MSE最小的epoch作为训练结果。保存这个epoch得到的模型参数，并且得到其在test集上的pred值和MSE损失

```
# 训练模型
num_epochs = 10000
loss_train_best = 1e15
loss_val_best = 1e15
best_train_epoch = 0
best_val_epoch = 0
y_best_val = []
y_best_train = []
y_best_test = []
init_time = time.time()
for epoch in range(num_epochs):
    # 划分训练集和验证集，交叉验证
    x_train_split, x_val_split, y_train_split, y_val_split =
train_test_split(x_train, y_train, test_size=0.2, random_state=epoch)
    optimizer.zero_grad()
    y_train_split_pred = model2(x_train_split)
    loss = loss_fonc(y_train_split_pred, y_train_split)
    losses_train.append(loss.detach().numpy())

    # 选择最优train epoch
    if loss < loss_train_best:
        loss_train_best = loss
        best_train_epoch = epoch + 1
        y_best_train = model2(x_train)  # 输出全部100个数据的pred
    loss.backward()
    optimizer.step()
    epoch_time = time.time()
    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}],USE time total{epoch_time -
init_time:.2f}, MSELoss: {loss.item()}')
    # 测试模型
    with torch.no_grad():
        y_val_pred = model2(x_val_split)
        loss = loss_fonc(y_val_pred, y_val_split)
        losses_val.append(loss.detach().numpy())
        if loss < loss_val_best:
            loss_val_best = loss
            best_val_epoch = epoch + 1
            y_best_val = model2(x_train)
            y_best_test = model2(x_test)  # 选取验证集损失最低作为测试集模型
            loss_test_best = loss_fonc(y_best_test, y_test)
            # 在训练循环中记录val集 MSE 最低时的模型参数
            best_model_state = model2.state_dict()  # 获取模型参数
print(f'BEST Train Epoch [{best_train_epoch}], MSELoss:
{loss_train_best.item()}')
print(f'BEST Val Epoch [{best_val_epoch}], MSELoss: {loss_val_best.item()}')
print(f'BEST Test Epoch [{best_val_epoch}], MSELoss: {loss_test_best.item()}')

# 保存模型参数到文件中
torch.save(best_model_state, 'best_model.pth')
```

拟合结果

MSE结果：0.2476

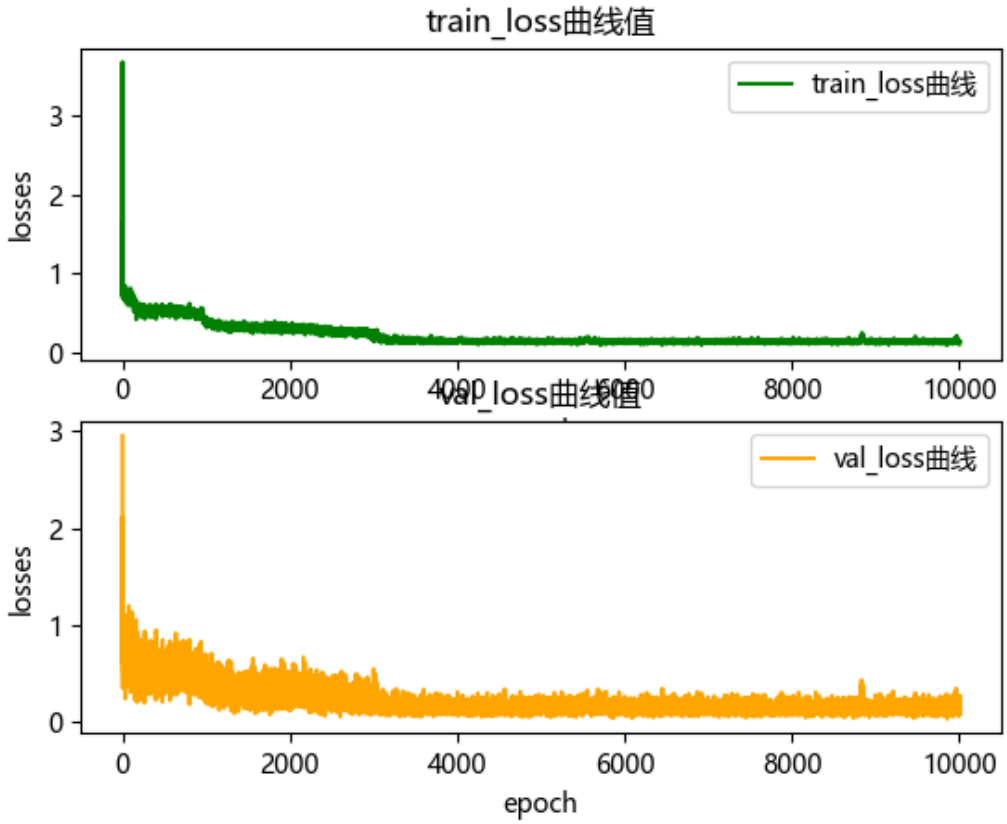
运行上述代码和配套的画图代码（附录可见）得到的结果如下图所示，可见在真正的未知数据上的损失值还是要大于在训练集和验证集上的损失值

```
BEST Train Epoch [9916], MSELoss: 0.10673527419567108
BEST Val Epoch [5117], MSELoss: 0.034190453588962555
BEST Test Epoch [5117], MSELoss: 0.24762500822544098
```

得到的最优模型的测试集MSE损失为 ***0.2476***

Loss曲线

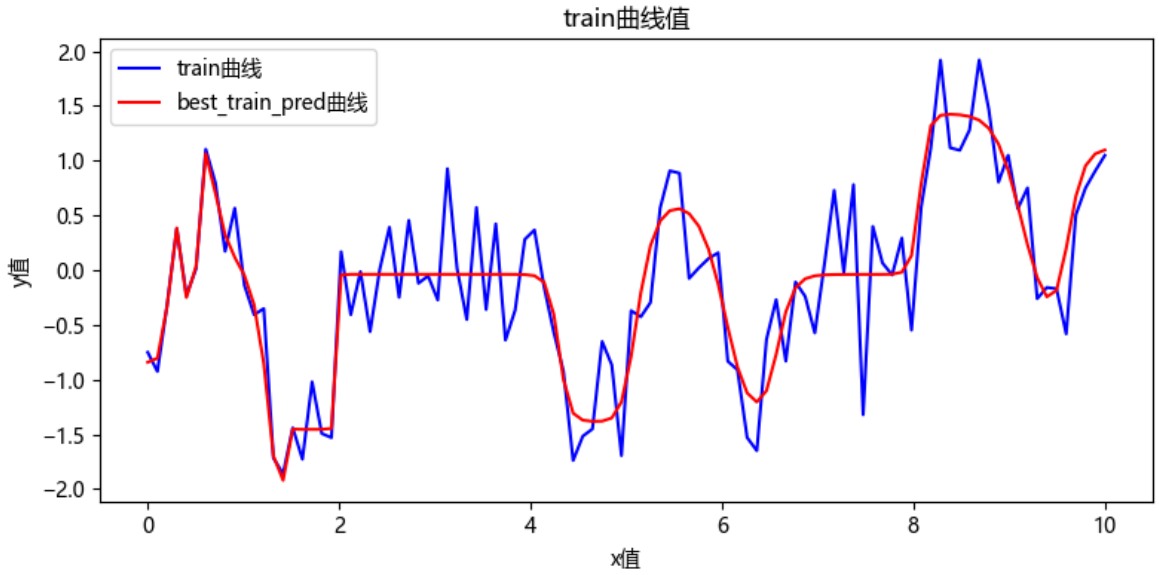
接下来对损失曲线进行绘制，得到训练集和验证集的损失曲线

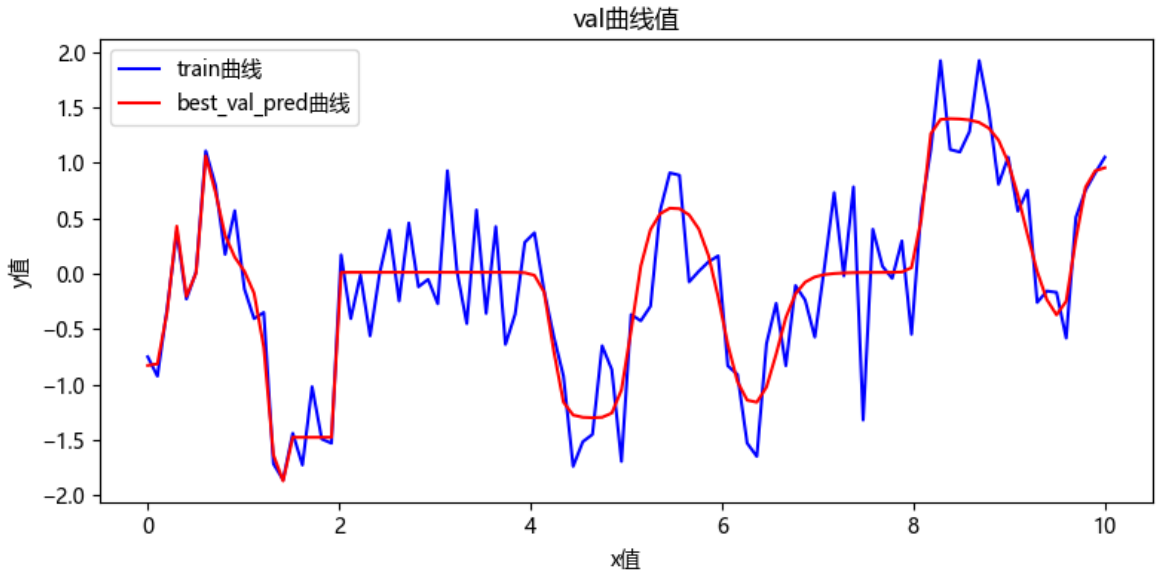


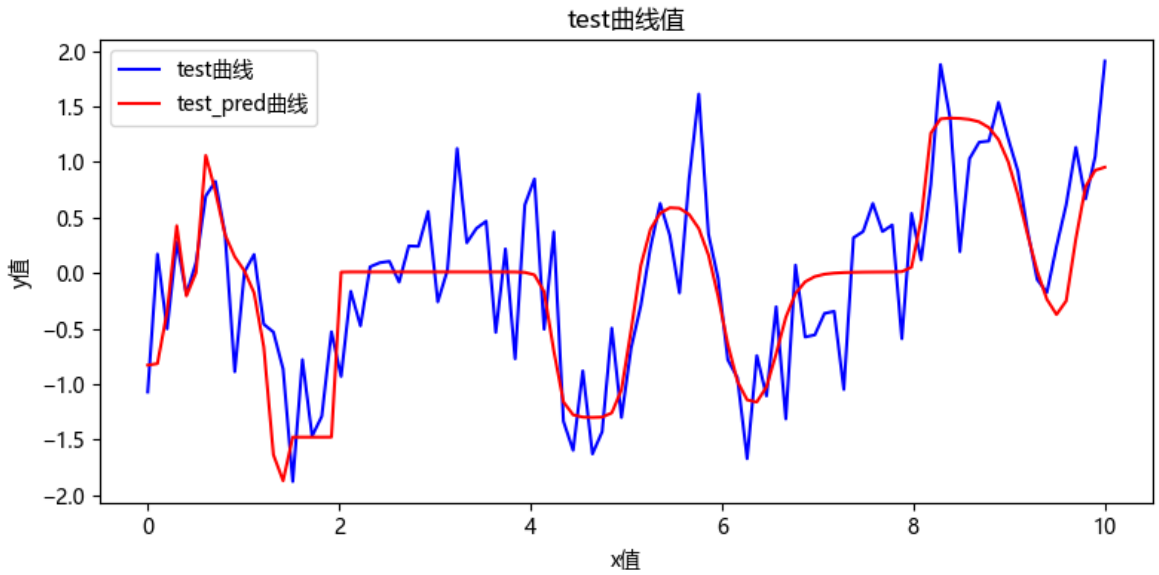
在验证集上的损失相较于训练集存在一些波动，但整体上呈现一个较好的收敛趋势，在3000epoch后MSE不再有太大的优化幅度

拟合曲线

通过对记录下的拟合曲线值，可以更直观的看到拟合的效果。下面三幅图像分别是训练集中最好结果与全部已知数据对比；最好验证结果与全部已知数据对比；最好验证结果与未知数据对比结果。由于训练数据存在较大噪声，拟合的结果智能体现函数的大致趋向以及数据走势，并且将噪声的一小部分拟合到函数中，在[0,1]闭区间内体现较为明显。


train曲线值
train曲线
best_train_pred曲线
y值
x值


val曲线值
train曲线
best_val_pred曲线
y值
x值


test曲线值
test曲线
test_pred曲线
y值
x值

附录

画图代码

```
# 绘制loss曲线
plt.subplot(2, 1, 1)
plt.plot(losses_train, color='green', label='train_loss曲线')
plt.legend()
plt.title('train_loss曲线值')
plt.xlabel('epoch')
plt.ylabel('losses')

plt.subplot(2, 1, 2)
plt.plot(losses_val, color='orange', label='val_loss曲线')
plt.legend()
plt.title('val_loss曲线值')
plt.xlabel('epoch')
plt.ylabel('losses')
plt.show()

# 绘制曲线
x_train = np.array(x_train)
y_train = np.array(y_train)
x_test = np.array(x_test)
y_test = np.array(y_test)
y_best_train = y_best_train.detach().numpy()
y_best_test = np.array(y_best_test)
y_best_val = np.array(y_best_val)

# 设置整个图像的大小
plt.figure(figsize=(8, 12))
plt.subplot(3, 1, 1)  # 三行一列，第一个图
plt.plot(x_train, y_train, color='blue', label='train曲线')
plt.plot(x_train, y_best_train, color='red', label='best_train_pred曲线')
plt.legend()
plt.title('train曲线值')
plt.xlabel('x值')
plt.ylabel('y值')

plt.subplot(3, 1, 2)  # 三行一列，第二个图
plt.plot(x_train, y_train, color='blue', label='train曲线')
plt.plot(x_train, y_best_val, color='red', label='best_val_pred曲线')
plt.legend()
plt.title('val曲线值')
plt.xlabel('x值')
plt.ylabel('y值')

plt.subplot(3, 1, 3)  # 三行一列，第三个图
plt.plot(x_test, y_test, color='blue', label='test曲线')
plt.plot(x_test, y_best_test, color='red', label='test_pred曲线')
plt.legend()
plt.title('test曲线值')
plt.xlabel('x值')
plt.ylabel('y值')

# 显示图形
plt.tight_layout()
plt.show()
```