1.数据准备

- 指定文件路径

```
train_path = 'Data/covid.train.csv'
test_path = 'Data/covid.test.csv'
```

- 导入本次实验需要的一些包，同时设定随机数种子，固定卷积算法，关闭自动搜索算法，确保每次本模型返回的结果都是相同的。

```
import torch
import torch.nn as nn
from torch.utils.data import DataLoader,Dataset

import numpy as np
import csv
import os
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
myseed = 42069  # 创作随机数种子
torch.backends.cudnn.deterministic = True  # 每次返回的卷积算法将是确定的，即默认算法
torch.backends.cudnn.benchmark = False  # 设定不用自行探索卷积算法
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)
```

- 制作自定义数据集
  本数据集一共95列，第一列为id，删去。第2-94列为特征，后1列为标签。

```
class CovidDataset(Dataset):
    def __init__(self,path) -> None:
        self.features = list(range(1,94))
        data = pd.read_csv(path)
        data = np.array(data)
        self.target = data[:,-1]
        self.data = data[:,self.features]

    def __getitem__(self, index):
        return self.data[index],self.target[index]

    def __len__(self):
        return self.data.shape[0]
covid_dataset = CovidDataset(train_path)
covid_dataloader = DataLoader(covid_dataset,64,shuffle=True,drop_last=False)
```

2.模型构建以及训练

构建模型

```
class LinearNet(nn.Module):
    def __init__(self,input_dim):
        super(LinearNet,self).__init__()
        self.linear1 = nn.Linear(input_dim,64)
        self.relu1 = nn.ReLU()
        self.linear2 = nn.Linear(64,1)
        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self,x):
        x = self.linear1(x)
        x = self.relu1(x)
        x = self.linear2(x)
        return x

    def getloss(self, x, y):  # 实现LogicNet类的损失值接口
        y_pred = self.forward(x)
        loss = self.criterion(y_pred, y)  # 计算损失值的交叉熵
        return loss
```

训练:

```
from numpy import average


model = LinearNet(93)
optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
for i in range(200):
    losses = list()
    for x,y in covid_dataloader:
        optimizer.zero_grad()
        loss = model.getloss(x,y)
        losses.append(loss.item())
        loss.backward()
        optimizer.step()
    print(average(losses))
```

3.预测

```
test = pd.read_csv(test_path)
test = np.array(test)[:, list(range(1, 94))]
test = torch.tensor(test)
model.forward(test)
```