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。
  原始notebook里读取数据的包使用的是csv,笔者在此处将其改为pandas。

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

import numpy as np
import pandas as pd
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)
```

- 制作辅助函数
  首先制作一个查看当前机器是否有可用gpu的函数:

```
def get_device():
    return 'cuda' if torch.cuda.is_available() else 'cpu'
```

- 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。
  本数据集可以同时应用于训练，验证和测试模式的数据查询。

```
class CovidDataset(Dataset):
    def __init__(self,path,mode='train',target_only=False):
        self.mode = mode
        data = pd.read_csv(path)  # 读取数据
        data = np.array(data)[:,1:].astype(float)  # 将数据转化为array格式
```

```
        if not target_only:
            feats = list(range(93))  # 特征列的列号列表(因为最后一列是标签)
        else:
            pass
        if mode=='test':  # 进入测试模式
            data = data[:,feats]
            self.data = torch.FloatTensor(data)  # 将数据转化为张量并存储起来
        else:  # 进入训练模式或者验证模式
            target = data[:,-1]  # 拿到target
            data = data[:,feats]  # 拿到features
            if mode == 'train':
                """每十行抽一个数据出来做验证集"""
                indices = [i for i in range(len(data)) if i%10!=0]
            else:
                indices = [i for i in range(len(data)) if i % 10 == 0]
            self.data = torch.FloatTensor(data[indices])
            self.target = torch.FloatTensor(target(indices))
        """对40列以后的数据做Mean-Std标准化"""
        self.data[:, 40:] = (self.data[:, 40:]-self.data[:,
40:].mean(dim=0,keepdim=True))/self.data[:, 40:].std(dim=0,keepdim=True)
        self.dim = self.data.shape[1]  # 返回特征列数
        print("数据集制作完成")

    def __getitem__(self, index):
        if self.mode in ['train','dev']:
            """训练和验证模式下返回特征和标签"""
            return self.data[index],self.target[index]
        else:
            """测试模式下只返回特征"""
            return self.data[index]

    def __len__(self):
        return self.data.shape[0]
```

- 生成一个dataloader

在制作完DataSet后,即可用此DatsSet来生成DataLoader来载入数据,为了增大载入数据的方便性，在此制作一个函数来制作DataLoader,代码如下:

```
def prep_dataloader(path,mode,batch_size,n_jobs=0,target_only=False):
    dataset = CovidDataset(path=path,mode=mode,target_only=target_only)  # 实例化
一个数据集
    dataloader = DataLoader(dataset,batch_size=batch_size,shuffle=
(mode=='train'),drop_last=False,num_workers=n_jobs,
                            pin_memory=True)
    return dataloader
```

2.模型构建

构建模型

```
class NeuralNet(nn.Module):
    def __init__(self,input_dim):
        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 cal_loss(self,pred,target):
        return self.criterion(pred,target)
```

模型构建为一个两层的全连接网络，中间通过ReLu函数作为激活函数，然后使用MSELoss计算损失值。

3.模型训练:

在训练的时候同时要使用验证集进行验证,如果验证结果比较优秀,则对当前模型进行存储,同时更新存储结果,如果结果不好则继续。

```
def train(tr_set,dv_set,model,config,device):
    n_epochs = config['n_epochs']  # epochs最大值
    optimizer = getattr(torch.optim,config['optimizer'])
(model.parameters(),**config['optim_hparas'])  # 获取优化器
    min_mse = 1000
    loss_record = {'train':[],'dev':[]}  # 生成字典嵌套列表来记录损失值
    early_stop_cnt = 0
    epoch = 0
    while epoch < n_epochs:
        model.train()  # 让模型进入训练模式
        for x,y in tr_set:  # 从dataloader里拿到数据
            optimizer.zero_grad()  # 清空优化器中的梯度值
            x, y = x.to(device),y.to(device)  # 将数据存储到指定device中
            pred = model(x)  # 执行forward方法
            mse_loss = model.cal_loss(pred,y)  # 计算本轮训练得出来的损失值
            mse_loss.backward()  # 计算梯度
            optimizer.step()  # 根据计算出来的梯度更新模型参数，更新模型
            loss_record['train'].append(mse_loss.detach().cpu().item())  # 存储损
失值

        """每轮epoch跑完之后,要使用验证集进行验证"""
        dev_mse = dev(dv_set,model,device)
        if dev_mse < min_mse:
            """如果模型验证结果比目前最优结果要好,则应该保留本轮epoch的参数"""
            min_mse = dev_mse  # 更新目前最优模型结果
            print("保存局部最优模型,epoch={:4d},loss={:.4f}".format(epoch+1,
min_mse))
```

```
            torch.save(model.state_dict(),config['save_path'])  # 保存当前模型在指
定路径下
            early_stop_cnt = 0
        else:
            early_stop_cnt += 1
        epoch += 1
        loss_record['dev'].append(dev_mse)  # 存储当前验证集的损失
        if early_stop_cnt > config['early_stop']:
            break
    print("在经过{}epoch后完成训练".format(epoch))
    return min_mse,loss_record
```

4.预测

```
def save_pred(preds, file):
    ''' 保存预测结果'''
    print('Saving results to {}'.format(file))
    with open(file, 'w') as fp:
        writer = csv.writer(fp)
        writer.writerow(['id', 'tested_positive'])
        for i, p in enumerate(preds):
            writer.writerow([i, p])
```