作业 1: 基于DNN的新冠病例预测 (回归)

1 数据声明

In [1]:
```
tr_path = 'covid.train.csv'  # 训练数据
```

2 使用包声明

In [2]:
```
# PyTorch
import torch
import torch.nn as nn   #神经网络
from torch.utils.data import Dataset, DataLoader    #数据集相关

# For data preprocess
import numpy as np     #矩阵运算，用于计算梯度、更新神经网络等。
import csv             #数据表格的读写
import os              #系统驱动函数

# 绘图相关
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

myseed = 42069  # 随机种子
torch.backends.cudnn.deterministic = True   #采用确定的算法
torch.backends.cudnn.benchmark = False      #采用默认算法
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():               #如果使用显卡，则为显卡设置随机种子
    torch.cuda.manual_seed_all(myseed)
```

3 函数定义

In [3]:
```
def get_device():
    ''' 获取设备信息 (优先选择GPU) '''
    return 'cuda' if torch.cuda.is_available() else 'cpu'

def plot_learning_curve(loss_record, title=''):
    ''' 绘制性能曲线 (误差曲线) '''
    total_steps = len(loss_record['train'])   #训练数据集上的训练步数
    x_1 = range(total_steps)
    x_2 = x_1[::len(loss_record['train']) // len(loss_record['dev'])]   #每个Epoch记录一次dev loss
                                                                        #一个Batch，即为1个Training
    figure(figsize=(6, 4))
    plt.plot(x_1, loss_record['train'], c='tab:red', label='train')  #训练数据集
    plt.plot(x_2, loss_record['dev'], c='tab:cyan', label='dev')     #验证数据集
    plt.ylim(0.0, 5.)
    plt.xlabel('Training steps')
    plt.ylabel('MSE loss')
    plt.title('Learning curve of {}'.format(title))
    plt.legend()
    plt.show()


def plot_pred(dv_set, model, device, lim=35., preds=None, targets=None):
    ''' 绘制神经网络的准确度 '''
    if preds is None or targets is None:
        model.eval()   #开始模型的测试（训练结束后）
        preds, targets = [], []
        for x, y in dv_set:         #验证数据集
            x, y = x.to(device), y.to(device)    #指定计算设备
            with torch.no_grad():
                pred = model(x)
                preds.append(pred.detach().cpu())
                targets.append(y.detach().cpu())
        preds = torch.cat(preds, dim=0).numpy()
        targets = torch.cat(targets, dim=0).numpy()

    figure(figsize=(5, 5))
    plt.scatter(targets, preds, c='r', alpha=0.5)
    plt.plot([-0.2, lim], [-0.2, lim], c='b')
    plt.xlim(-0.2, lim)
    plt.ylim(-0.2, lim)
    plt.xlabel('ground truth value')
    plt.ylabel('predicted value')
    plt.title('Ground Truth v.s. Prediction')
    plt.show()
```

4 数据处理

三种类型的数据:

- train : 训练数据集
- dev : 验证数据集
- test : 测试训练集 (无y的值)

4.1 Dataset (可改进)

类 COVID19Dataset 的功能为:

- 读取 .csv 文件

- 提取数据属性（特征）
- 将 covid.train.csv 分为训练数据集、验证数据集
- 归一化数据属性

完成 TODO 部分，可以获得较高分数

In [4]:

```
class COVID19Dataset(Dataset):
    '''处理COVID19数据集的dataset类'''
    def __init__(self,
                 path,
                 mode='train',
                 target_only=False):
        self.mode = mode

        # 读取数据至numpy数组
        with open(path, 'r') as fp:
            data = list(csv.reader(fp))
            data = np.array(data[1:])[:, 1:].astype(float)

        if not target_only:     #读取x的值，即前93列
            feats = list(range(93))
        else:                   #读取y的值
            # TODO1: 完成下一行代码，读取40个州、前2天的test_positive数据。即读取第57、75列。
            # feats =
            pass

        if mode == 'test':
            # 测试数据集
            # data: 675 x 93 (40 states + day 1 (18) + day 2 (18) + day 3 (17))
            data = data[:, feats]
            self.data = torch.FloatTensor(data)
        else:
            # 训练数据集 (训练/验证)
            # data: 2025 x 94 (40 states + day 1 (18) + day 2 (18) + day 3 (18))
            target = data[:, -1]
            data = data[:, feats]

            # 将训练数据分为训练、验证数据集，即train & dev
            if mode == 'train': #训练
                indices = [i for i in range(len(data)) if i % 10 != 0]
            elif mode == 'dev': #验证
                indices = [i for i in range(len(data)) if i % 10 == 0]

            # 将数据转换为PyTorch tensors
            self.data = torch.FloatTensor(data[indices])
            self.target = torch.FloatTensor(target[indices])

        # 归一化数据 (可以尝试注释掉，看看会发生什么)
        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]

        print('Finished reading the {} set of COVID19 Dataset ({} samples found, each dim = {})'
              .format(mode, len(self.data), self.dim))

    def __getitem__(self, index):
        # 每次返回一个训练数据样本
        if self.mode in ['train', 'dev']:
            # 训练
            return self.data[index], self.target[index]
        else:
            # 测试（没有target）
            return self.data[index]

    def __len__(self):
        # 返回数据集的大小
        return len(self.data)
```

4.2 DataLoader

DataLoader 从 Dataset 中提取数据，并组装为Batch.

In [5]:
```
def prep_dataloader(path, mode, batch_size, n_jobs=0, target_only=False):
    ''' 构建一个dataset,然后以Batch形式装载进dataloader. '''
    dataset = COVID19Dataset(path, mode=mode, target_only=target_only)  # 构建dataset
    dataloader = DataLoader(
        dataset, batch_size,
        shuffle=(mode == 'train'), drop_last=False,
        num_workers=n_jobs, pin_memory=True)                            # 构建dataloader
    return dataloader
```

5 深度神经网络DNN (可改进)

NeuralNet 由 nn.Module 派生。 DNN包括2个全连接层，采用ReLU激活函数。 其中函数 cal_loss 用来计算损失（误差）.

In [6]:
```
class NeuralNet(nn.Module):
    ''' 简单的全连接神经网络 '''
    def __init__(self, input_dim):
        super(NeuralNet, self).__init__()

        # 可以在此处构建自己的神经网络
        # TODO2: 尝试修改网络结构，以提高性能
        self.net = nn.Sequential(
            nn.Linear(input_dim, 4),  # 4个神经元
            nn.Sigmoid(),
            nn.Linear(4, 1)
        )

        # 损失函数：采用均方误差
        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self, x):
        ''' 给定输入数据维度 (batch_size X input_dim)，计算网络的输出 '''
        return self.net(x).squeeze(1)

    def cal_loss(self, pred, target):
        '''计算损失函数'''
        # TODO3: 此处可以采用L2正则化，不妨试试。补充相关代码
        #
        return self.criterion(pred, target)
```

6 训练/验证

In [7]:
```python
def train(tr_set, dv_set, model, config, device):
    ''' 训练DNN '''

    n_epochs = config['n_epochs']  # 最大epochs

    # 设置优化器
    optimizer = getattr(torch.optim, config['optimizer'])(
        model.parameters(), **config['optim_hparas'])

    min_mse = 1000.
    loss_record = {'train': [], 'dev': []}      # 记录训练、验证误差
    early_stop_cnt = 0
    epoch = 0
    while epoch < n_epochs:
        model.train()                           # 设置模型为“训练”状态
        for x, y in tr_set:                      # 每一个数据样本
            optimizer.zero_grad()               # 梯度归零
            x, y = x.to(device), y.to(device)   # 数据装载至设备 (cpu/cuda)
            pred = model(x)                     # 前向计算，得到输出
            mse_loss = model.cal_loss(pred, y)  # 计算误差
            mse_loss.backward()                 # 计算梯度 (BP算法)
            optimizer.step()                    # 利用优化器，更新模型参数
            loss_record['train'].append(mse_loss.detach().cpu().item()) #记录误差

        # 每个epoch结束，采用验证数据集测试.
        dev_mse = dev(dv_set, model, device)
        if dev_mse < min_mse:
            # 若模型改进，即误差减小，则存储模型参数
            min_mse = dev_mse
            print('Saving model (epoch = {:4d}, loss = {:.4f})'
                .format(epoch + 1, min_mse))
            torch.save(model.state_dict(), config['save_path'])  # 保存模型至“save_path”
            early_stop_cnt = 0
        else:
            early_stop_cnt += 1   #模型没有改进，则激活early stop机制

        epoch += 1
        loss_record['dev'].append(dev_mse)   #记录验证数据集的误差
        if early_stop_cnt > config['early_stop']:
            # 若经过 "config['early_stop']" epochs，模型不再改进，则停止训练
            break

    print('Finished training after {} epochs'.format(epoch))
    return min_mse, loss_record
```

In [8]:
```python
# 验证
def dev(dv_set, model, device):
    model.eval()                                # 将模型调整为测试状态
    total_loss = 0
    for x, y in dv_set:                          # 每一组验证数据
        x, y = x.to(device), y.to(device)       # 装载数据至 (cpu/cuda)
        with torch.no_grad():                   # 禁用梯度
            pred = model(x)                     # 前向计算，得到网络的输出（预测）
            mse_loss = model.cal_loss(pred, y)  # 计算损失（mse）
        total_loss += mse_loss.detach().cpu().item() * len(x)  # 损失累加
    total_loss = total_loss / len(dv_set.dataset)              # 求平均值

    return total_loss
```

7 设置超参数 (可改进)

config 包含 超参数 以及模型参数的 存储路径 。

In [9]:
```
device = get_device()                 # 获取当前可用的设备 ('cpu' 或 'cuda')
os.makedirs('models', exist_ok=True)  # 模型参数存储文件夹 ./models/
target_only = False                   # TODO: Using 40 states & 2 tested_positive features

# TODO4: 尝试调整这些超参数，以改善性能
config = {
    'n_epochs': 30,                # 最大epochs数目
    'batch_size': 10,               # mini-batch的大小
    'optimizer': 'SGD',             # 优化方法 (optimizer in torch.optim)
    'optim_hparas': {               # 优化器的超参数 (与选择的算法有关)
        'lr': 0.1,                 # SGD的学习率
        'momentum': 0.3             # SGD的momentum
    },
    'early_stop': 200,              # early stopping epochs (连续若干个epoch下，模型仍未改进，则结
    'save_path': 'models/model.pth'  # 模型存储路径
}
```

8 装载数据与模型

In [10]:
```
tr_set = prep_dataloader(tr_path, 'train', config['batch_size'], target_only=target_only)  #训练数
dv_set = prep_dataloader(tr_path, 'dev', config['batch_size'], target_only=target_only)    #验证数
```

```
Finished reading the train set of COVID19 Dataset (1822 samples found, each dim = 93)
Finished reading the dev set of COVID19 Dataset (203 samples found, each dim = 93)
```

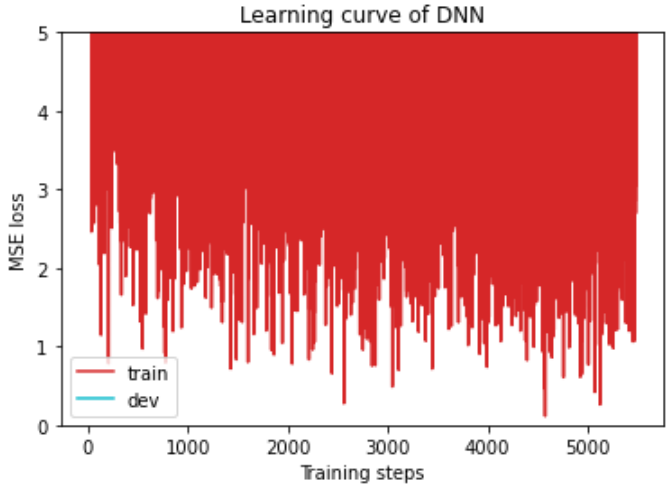
In [11]:
```
model = NeuralNet(tr_set.dataset.dim).to(device)  # 构建模型并装载至（cpu/cuda）
```

9 开始训练

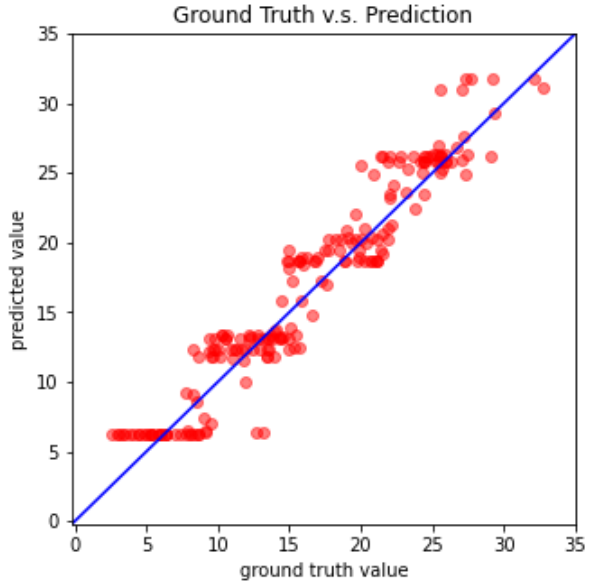
In [12]:
```
model_loss, model_loss_record = train(tr_set, dv_set, model, config, device)
```

```
Saving model (epoch =    1, loss = 10.7528)
Saving model (epoch =    2, loss = 7.7793)
Saving model (epoch =    3, loss = 7.2512)
Saving model (epoch =    6, loss = 6.7850)
Saving model (epoch =   10, loss = 5.7512)
Saving model (epoch =   16, loss = 5.7002)
Saving model (epoch =   17, loss = 5.5935)
Saving model (epoch =   19, loss = 5.3123)
Saving model (epoch =   28, loss = 5.1934)
Finished training after 30 epochs
```

In [13]:
```
plot_learning_curve(model_loss_record, title='DNN')  #绘图
```



In [14]:
```
del model
model = NeuralNet(tr_set.dataset.dim).to(device)
ckpt = torch.load(config['save_path'], map_location='cpu')  # 加载最佳的模型
model.load_state_dict(ckpt)
plot_pred(dv_set, model, device)  # 绘图：验证集上的预测情况
```



10 测试 (无视这一部分，仅用作评分)

将模型在测试数据集上的预测结果存储于 pred.csv .

In [15]:
```
def test(tt_set, model, device):
    model.eval()                                # 将模型调整为测试状态
    preds = []
    for x in tt_set:                            # 每一组“测试”数据（没有target，即y）
        x = x.to(device)                        # 装载数据至 (cpu/cuda)
        with torch.no_grad():                   # 禁用梯度
            pred = model(x)                     # 前向计算，得到网络的输出（预测）
            preds.append(pred.detach().cpu())   # 记录计算结果
    preds = torch.cat(preds, dim=0).numpy()     # 转换为 numpy array
    return preds

def save_pred(preds, file):
    ''' 将预测结果存储于指定文件 '''
    print('存储预测数据至{}'.format(file))
    with open(file, 'w', newline = '') as fp:
        writer = csv.writer(fp)
        writer.writerow(['id', 'tested_positive'])
        for i, p in enumerate(preds):
            writer.writerow([i, p])

TestMode = True
if TestMode == True:
    tt_path = 'covid.test.csv'    # 测试数据路径
    tt_set = prep_dataloader(tt_path, 'test', config['batch_size'], target_only=target_only)   #测
    preds = test(tt_set, model, device)  # 预测COVID-19
    save_pred(preds, 'pred.csv')         # 将数据存储至pred.csv
else:
    pass
```

```
Finished reading the test set of COVID19 Dataset (675 samples found, each dim = 93)
存储预测数据至pred.csv
```

11 提示

11.1 及格分

- 运行样本程序
- TODO1：属性选择: 40个州数据、2天的 `tested_positive` 数据 (`TODO1`)

11.2 进一步改善性能

- TODO1：进一步尝试选择不同的数据属性（Feature）来预测
- TODO2：DNN结构 (设置层数、维度或神经元个数、激活函数)
- TODO3：L2正则化
- TODO4：优化训练 (mini-batch的大小、优化算法的选择、学习率的设置等)
- 样本代码中有一些错误，能否发现?

12 致谢

原始代码作者：Heng-Jui Chang @ NTUEE。本人在此基础上进行了调整和修改。

In [ ]:

In [ ]: