Homework 1: COVID-19 Cases Prediction (Regression)

Author: Heng-Jui Chang

Slides: [https://github.com/ga642381/ML2021-Spring/blob/main/HW01/HW01.pdf (https://github.com/ga642381/ML2021-Spring/blob/main/HW01/HW01.pdf)](https://github.com/ga642381/ML2021-Spring/blob/main/HW01/HW01.pdf)
Video: TBA

Objectives:

- Solve a regression problem with deep neural networks (DNN).
- Understand basic DNN training tips.
- Get familiar with PyTorch.

If any questions, please contact the TAs via TA hours, NTU COOL, or email.

Download Data

If the Google drive links are dead, you can download data from [kaggle (https://www.kaggle.com/c/ml2021spring-hw1/data)](https://www.kaggle.com/c/ml2021spring-hw1/data), and upload data manually to the workspace.

In [ ]:
```
tr_path = 'covid.train.csv'  # path to training data
tt_path = 'covid.test.csv'   # path to testing data

!gdown --id '19CCyCgJrUxtvgZF53vnctJi0J23T5mqF' --output covid.train.csv
!gdown --id '1CE240jLm2npU-tdz81-oVKEF3T2yfT10' --output covid.test.csv
```

```
'gdown' �����ǵ����ⲿ���� Xҳ����ǿ�����е� e ij���
���������ļ���
'gdown' �����ǵ����ⲿ���� Xҳ����ǿ�����е� e ij���
���������ļ���
```

Import Some Packages

In [ ]:
```
# PyTorch
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

# For data preprocess
import numpy as np
import csv
import os

# For plotting
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

myseed = 42069  # set a random seed for reproducibility
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
np.random.seed(myseed)
torch.manual_seed(myseed)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(myseed)
```

Some Utilities

You do not need to modify this part.

In [ ]:
```
def get_device():
    ''' Get device (if GPU is available, use GPU) '''
    return 'cuda' if torch.cuda.is_available() else 'cpu'

def plot_learning_curve(loss_record, title=''):
    ''' Plot learning curve of your DNN (train & dev loss) '''
    total_steps = len(loss_record['train'])
    x_1 = range(total_steps)
    x_2 = x_1[::len(loss_record['train']) // len(loss_record['dev'])]
    figure(figsize=(6, 4))
    plt.plot(x_1, loss_record['train'], c='tab:red', label='train')
    plt.plot(x_2, loss_record['dev'], c='tab:cyan', label='dev')
    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):
    ''' Plot prediction of your DNN '''
    if preds is None or targets is None:
        model.eval()
        preds, targets = [], []
        for x, y in dv_set:
            x, y = x.to(device), y.to(device)
            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()
```

Preprocess

We have three kinds of datasets:

- `train` : for training
- `dev` : for validation
- `test` : for testing (w/o target value)

Dataset

The `COVID19Dataset` below does:

- read `.csv` files
- extract features
- split `covid.train.csv` into train/dev sets
- normalize features

Finishing `TODO` below might make you pass medium baseline.

In [ ]:
```
class COVID19Dataset(Dataset):
    ''' Dataset for loading and preprocessing the COVID19 dataset '''
    def __init__(self,
                 path,
                 mode='train',
                 target_only=False):
        self.mode = mode

        # Read data into numpy arrays
        with open(path, 'r') as fp:
            data = list(csv.reader(fp))
            data = np.array(data[1:])[:, 1:].astype(float)

        if not target_only:
            feats = list(range(93))
        else:
            # TODO: Using 40 states & 2 tested_positive features (indices = 57 & 75)
            pass

        if mode == 'test':
            # Testing data
            # data: 893 x 93 (40 states + day 1 (18) + day 2 (18) + day 3 (17))
            data = data[:, feats]
            self.data = torch.FloatTensor(data)
        else:
            # Training data (train/dev sets)
            # data: 2700 x 94 (40 states + day 1 (18) + day 2 (18) + day 3 (18))
            target = data[:, -1]
            data = data[:, feats]

            # Splitting training data into train & dev sets
            if mode == 'train':
                indices = [i for i in range(len(data)) if i % 10 != 0]
            elif mode == 'dev':
                indices = [i for i in range(len(data)) if i % 10 == 0]

            # Convert data into PyTorch tensors
            self.data = torch.FloatTensor(data[indices])
            self.target = torch.FloatTensor(target[indices])

        # Normalize features (you may remove this part to see what will happen)
```

```
        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):
        # Returns one sample at a time
        if self.mode in ['train', 'dev']:
            # For training
            return self.data[index], self.target[index]
        else:
            # For testing (no target)
            return self.data[index]

    def __len__(self):
        # Returns the size of the dataset
        return len(self.data)
```

DataLoader

A `DataLoader` loads data from a given `Dataset` into batches.

In [ ]:

```
def prep_dataloader(path, mode, batch_size, n_jobs=0, target_only=False):
    ''' Generates a dataset, then is put into a dataloader. '''
    dataset = COVID19Dataset(path, mode=mode, target_only=target_only)  # Construct dataset
    dataloader = DataLoader(
        dataset, batch_size,
        shuffle=(mode == 'train'), drop_last=False,
        num_workers=n_jobs, pin_memory=True)                            # Construct dataloader
    return dataloader
```

Deep Neural Network

`NeuralNet` is an `nn.Module` designed for regression. The DNN consists of 2 fully-connected layers with ReLU activation. This module also included a function `cal_loss` for calculating loss.

In [ ]:
```
class NeuralNet(nn.Module):
    ''' A simple fully-connected deep neural network '''
    def __init__(self, input_dim):
        super(NeuralNet, self).__init__()

        # Define your neural network here
        # TODO: How to modify this model to achieve better performance?
        self.net = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

        # Mean squared error loss
        self.criterion = nn.MSELoss(reduction='mean')

    def forward(self, x):
        ''' Given input of size (batch_size x input_dim), compute output of the network '''
        return self.net(x).squeeze(1)

    def cal_loss(self, pred, target):
        ''' Calculate loss '''
        # TODO: you may implement L2 regularization here
        return self.criterion(pred, target)
```

Train/Dev/Test

Training

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

    n_epochs = config['n_epochs']  # Maximum number of epochs

    # Setup optimizer
    optimizer = getattr(torch.optim, config['optimizer'])(
        model.parameters(), **config['optim_hparas'])

    min_mse = 1000.
    loss_record = {'train': [], 'dev': []}      # for recording training loss
    early_stop_cnt = 0
    epoch = 0
    while epoch < n_epochs:
        model.train()                           # set model to training mode
        for x, y in tr_set:                     # iterate through the dataloader
            optimizer.zero_grad()               # set gradient to zero
            x, y = x.to(device), y.to(device)   # move data to device (cpu/cuda)
            pred = model(x)                     # forward pass (compute output)
            mse_loss = model.cal_loss(pred, y)  # compute loss
            mse_loss.backward()                 # compute gradient (backpropagation)
            optimizer.step()                    # update model with optimizer
            loss_record['train'].append(mse_loss.detach().cpu().item())

        # After each epoch, test your model on the validation (development) set.
        dev_mse = dev(dv_set, model, device)
        if dev_mse < min_mse:
            # Save model if your model improved
            min_mse = dev_mse
            print('Saving model (epoch = {:4d}, loss = {:.4f})'
                .format(epoch + 1, min_mse))
            torch.save(model.state_dict(), config['save_path'])  # Save model to specified path
            early_stop_cnt = 0
        else:
            early_stop_cnt += 1

        epoch += 1
        loss_record['dev'].append(dev_mse)
        if early_stop_cnt > config['early_stop']:
            # Stop training if your model stops improving for "config['early_stop']" epochs.
            break
```

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

Validation

In [ ]:
```
def dev(dv_set, model, device):
    model.eval()                                # set model to evalutation mode
    total_loss = 0
    for x, y in dv_set:                         # iterate through the dataloader
        x, y = x.to(device), y.to(device)       # move data to device (cpu/cuda)
        with torch.no_grad():                   # disable gradient calculation
            pred = model(x)                     # forward pass (compute output)
            mse_loss = model.cal_loss(pred, y)  # compute loss
        total_loss += mse_loss.detach().cpu().item() * len(x)  # accumulate loss
    total_loss = total_loss / len(dv_set.dataset)              # compute averaged loss

    return total_loss
```

Testing

In [ ]:
```
def test(tt_set, model, device):
    model.eval()                                # set model to evalutation mode
    preds = []
    for x in tt_set:                            # iterate through the dataloader
        x = x.to(device)                        # move data to device (cpu/cuda)
        with torch.no_grad():                   # disable gradient calculation
            pred = model(x)                     # forward pass (compute output)
            preds.append(pred.detach().cpu())   # collect prediction
    preds = torch.cat(preds, dim=0).numpy()     # concatenate all predictions and convert to a numpy array
    return preds
```

Setup Hyper-parameters

`config` contains hyper-parameters for training and the path to save your model.

In [ ]:
```
device = get_device()                 # get the current available device ('cpu' or 'cuda')
os.makedirs('models', exist_ok=True)  # The trained model will be saved to ./models/
target_only = False                   # TODO: Using 40 states & 2 tested_positive features

# TODO: How to tune these hyper-parameters to improve your model's performance?
config = {
    'n_epochs': 3000,                # maximum number of epochs
    'batch_size': 270,               # mini-batch size for dataloader
    'optimizer': 'SGD',              # optimization algorithm (optimizer in torch.optim)
    'optim_hparas': {                # hyper-parameters for the optimizer (depends on which optimizer you are using)
        'lr': 0.001,                 # learning rate of SGD
        'momentum': 0.9              # momentum for SGD
    },
    'early_stop': 200,               # early stopping epochs (the number epochs since your model's last improvement)
    'save_path': 'models/model.pth'  # your model will be saved here
}
```

Load data and model

In [ ]:
```
tr_set = prep_dataloader(tr_path, 'train', config['batch_size'], target_only=target_only)
dv_set = prep_dataloader(tr_path, 'dev', config['batch_size'], target_only=target_only)
tt_set = prep_dataloader(tt_path, 'test', config['batch_size'], target_only=target_only)
```

```
Finished reading the train set of COVID19 Dataset (2430 samples found, each dim = 93)
Finished reading the dev set of COVID19 Dataset (270 samples found, each dim = 93)
Finished reading the test set of COVID19 Dataset (893 samples found, each dim = 93)
```

In [ ]:
```
model = NeuralNet(tr_set.dataset.dim).to(device)  # Construct model and move to device
```

Start Training!

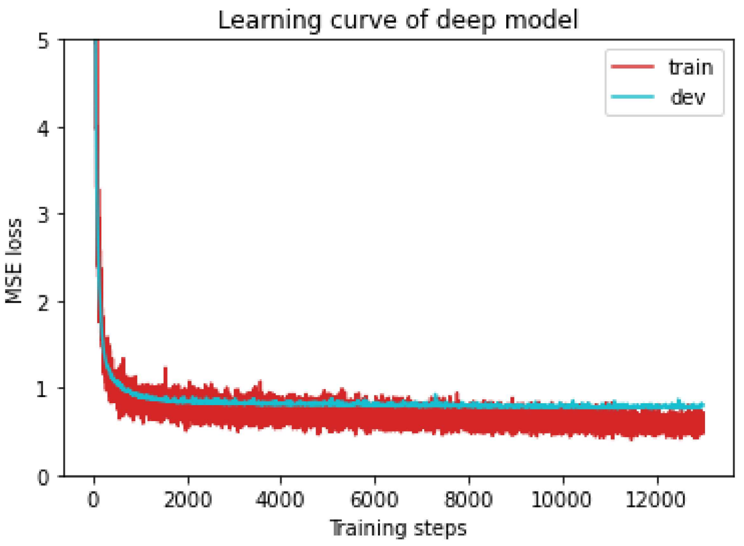
In [ ]:
```
model_loss, model_loss_record = train(tr_set, dv_set, model, config, device)
```

```
Saving model (epoch =    1,  loss = 78.8435)
Saving model (epoch =    2,  loss = 37.6133)
Saving model (epoch =    3,  loss = 26.1188)
Saving model (epoch =    4,  loss = 16.1867)
Saving model (epoch =    5,  loss = 9.7153)
Saving model (epoch =    6,  loss = 6.3715)
Saving model (epoch =    7,  loss = 5.1784)
Saving model (epoch =    8,  loss = 4.4246)
Saving model (epoch =    9,  loss = 3.8007)
Saving model (epoch =   10,  loss = 3.3693)
Saving model (epoch =   11,  loss = 3.0944)
Saving model (epoch =   12,  loss = 2.8165)
Saving model (epoch =   13,  loss = 2.6271)
Saving model (epoch =   14,  loss = 2.4546)
Saving model (epoch =   15,  loss = 2.3012)
Saving model (epoch =   16,  loss = 2.1764)
Saving model (epoch =   17,  loss = 2.0634)
Saving model (epoch =   18,  loss = 1.9386)
Saving model (epoch =   19,  loss = 1.8965)
Saving model (epoch =   20,  loss = 1.7939)
Saving model (epoch =   21,  loss = 1.7151)
Saving model (epoch =   22,  loss = 1.6442)
Saving model (epoch =   23,  loss = 1.5895)
Saving model (epoch =   24,  loss = 1.5597)
Saving model (epoch =   25,  loss = 1.5191)
Saving model (epoch =   26,  loss = 1.4690)
Saving model (epoch =   27,  loss = 1.4181)
Saving model (epoch =   28,  loss = 1.3983)
Saving model (epoch =   29,  loss = 1.3682)
Saving model (epoch =   30,  loss = 1.3425)
Saving model (epoch =   31,  loss = 1.3212)
Saving model (epoch =   32,  loss = 1.2817)
Saving model (epoch =   33,  loss = 1.2775)
Saving model (epoch =   34,  loss = 1.2440)
Saving model (epoch =   36,  loss = 1.2370)
Saving model (epoch =   37,  loss = 1.2101)
Saving model (epoch =   38,  loss = 1.2046)
Saving model (epoch =   41,  loss = 1.1564)
Saving model (epoch =   42,  loss = 1.1477)
Saving model (epoch =   44,  loss = 1.1195)
Saving model (epoch =   47,  loss = 1.1117)
```

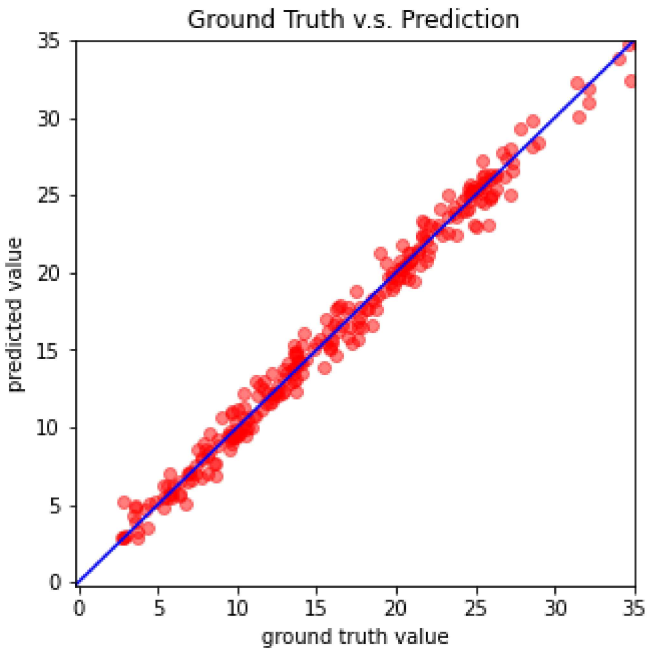
```
Saving model (epoch =   48, loss = 1.0929)
Saving model (epoch =   50, loss = 1.0831)
Saving model (epoch =   53, loss = 1.0659)
Saving model (epoch =   54, loss = 1.0613)
Saving model (epoch =   57, loss = 1.0530)
Saving model (epoch =   58, loss = 1.0394)
Saving model (epoch =   60, loss = 1.0265)
Saving model (epoch =   63, loss = 1.0248)
Saving model (epoch =   66, loss = 1.0094)
Saving model (epoch =   70, loss = 0.9836)
Saving model (epoch =   72, loss = 0.9822)
Saving model (epoch =   73, loss = 0.9747)
Saving model (epoch =   75, loss = 0.9672)
Saving model (epoch =   78, loss = 0.9649)
Saving model (epoch =   79, loss = 0.9602)
Saving model (epoch =   85, loss = 0.9562)
Saving model (epoch =   86, loss = 0.9537)
Saving model (epoch =   90, loss = 0.9474)
Saving model (epoch =   92, loss = 0.9438)
Saving model (epoch =   93, loss = 0.9233)
Saving model (epoch =   95, loss = 0.9125)
Saving model (epoch =  104, loss = 0.9114)
Saving model (epoch =  107, loss = 0.8988)
Saving model (epoch =  110, loss = 0.8929)
Saving model (epoch =  116, loss = 0.8880)
Saving model (epoch =  124, loss = 0.8865)
Saving model (epoch =  128, loss = 0.8715)
Saving model (epoch =  139, loss = 0.8666)
Saving model (epoch =  146, loss = 0.8640)
Saving model (epoch =  159, loss = 0.8522)
Saving model (epoch =  167, loss = 0.8490)
Saving model (epoch =  173, loss = 0.8486)
Saving model (epoch =  176, loss = 0.8460)
Saving model (epoch =  178, loss = 0.8412)
Saving model (epoch =  182, loss = 0.8374)
Saving model (epoch =  199, loss = 0.8300)
Saving model (epoch =  202, loss = 0.8299)
Saving model (epoch =  212, loss = 0.8274)
Saving model (epoch =  235, loss = 0.8250)
Saving model (epoch =  238, loss = 0.8237)
Saving model (epoch =  251, loss = 0.8214)
Saving model (epoch =  253, loss = 0.8198)
```

```
Saving model (epoch =  258, loss = 0.8177)
Saving model (epoch =  284, loss = 0.8136)
Saving model (epoch =  312, loss = 0.8078)
Saving model (epoch =  324, loss = 0.8050)
Saving model (epoch =  359, loss = 0.8041)
Saving model (epoch =  396, loss = 0.8039)
Saving model (epoch =  400, loss = 0.8031)
Saving model (epoch =  404, loss = 0.8003)
Saving model (epoch =  466, loss = 0.7989)
Saving model (epoch =  492, loss = 0.7986)
Saving model (epoch =  525, loss = 0.7984)
Saving model (epoch =  561, loss = 0.7938)
Saving model (epoch =  584, loss = 0.7897)
Saving model (epoch =  667, loss = 0.7889)
Saving model (epoch =  717, loss = 0.7812)
Saving model (epoch =  776, loss = 0.7806)
Saving model (epoch =  835, loss = 0.7794)
Saving model (epoch =  866, loss = 0.7762)
Saving model (epoch =  933, loss = 0.7740)
Saving model (epoch =  965, loss = 0.7699)
Saving model (epoch = 1027, loss = 0.7673)
Saving model (epoch = 1140, loss = 0.7673)
Saving model (epoch = 1196, loss = 0.7632)
Saving model (epoch = 1243, loss = 0.7590)
Finished training after 1444 epochs
```

In [ ]:
```
plot_learning_curve(model_loss_record, title='deep model')
```



In [ ]:
```
del model
model = NeuralNet(tr_set.dataset.dim).to(device)
ckpt = torch.load(config['save_path'], map_location='cpu')  # Load your best model
model.load_state_dict(ckpt)
plot_pred(dv_set, model, device)  # Show prediction on the validation set
```



Testing

The predictions of your model on testing set will be stored at `pred.csv` .

In [ ]:
```
def save_pred(preds, file):
    ''' Save predictions to specified file '''
    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])

preds = test(tt_set, model, device)  # predict COVID-19 cases with your model
save_pred(preds, 'pred.csv')         # save prediction file to pred.csv
```

```
Saving results to pred.csv
```

Hints

Simple Baseline

- Run sample code

Medium Baseline

- Feature selection: 40 states + 2 `tested_positive` (`TODO` in dataset)

Strong Baseline

- Feature selection (what other features are useful?)
- DNN architecture (layers? dimension? activation function?)
- Training (mini-batch? optimizer? learning rate?)
- L2 regularization
- There are some mistakes in the sample code, can you find them?

Reference

This code is completely written by Heng-Jui Chang @ NTUEE.
Copying or reusing this code is required to specify the original author.

E.g.
Source: Heng-Jui Chang @ NTUEE (https://github.com/ga642381/ML2021-Spring/blob/main/HW01/HW01.ipynb (https://github.com/ga642381/ML2021-Spring/blob/main/HW01/HW01.ipynb))