Train a model to predict COVID Cases with PyTorch

August 11, 2021

[1]:
```
import torch
import torch.optim as optim
import torch.nn as nn
import torch.utils.data.dataloader as dataloader
import torch.nn.functional as F
import pandas as pd
```

[2]:
```
url1 = 'https://raw.githubusercontent.com/tvelichkovt/PyTorch/master/covid.test.
 ↪csv'
url2 = 'https://raw.githubusercontent.com/tvelichkovt/PyTorch/master/covid.
 ↪train.csv'

tt_path = pd.read_csv(url1)
tr_path = pd.read_csv(url2)


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

[3]:
```
# 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)
```

[4]:
```
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()
```

[5]:
```
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)
```

[6]:
```
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
```

[7]:
```
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 L1/L2 regularization here
        return self.criterion(pred, target)
```

[8]:
```
# Training

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
```

[9]:
```
# Validation

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
```

[10]:
```
# Testing

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
```

[11]:
```
# Setup Hyper-parameters

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
}
```

[12]:
```
# Load data and model

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)
```

[13]:
```
model = NeuralNet(tr_set.dataset.dim).to(device)  # Construct model and move to␣
↪device
```

[14]:
```
# Start Training

model_loss, model_loss_record = train(tr_set, dv_set, model, config, device)
```

```
Saving model (epoch =    1, loss = 78.8524)
Saving model (epoch =    2, loss = 37.6170)
Saving model (epoch =    3, loss = 26.1203)
```

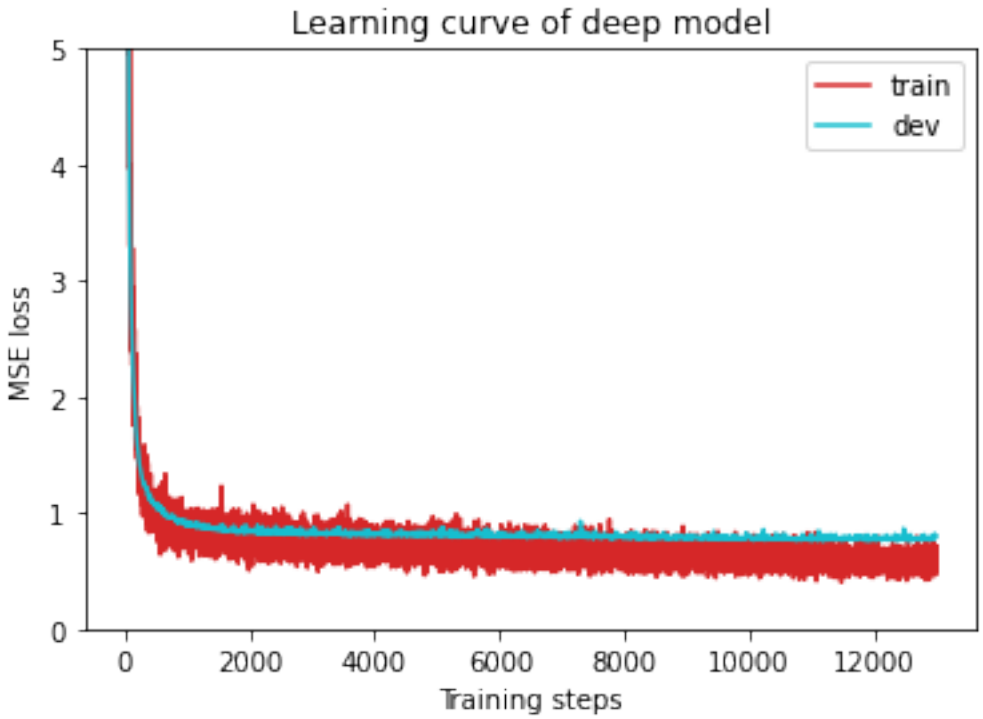
```
Saving model (epoch =    4, loss = 16.1862)
Saving model (epoch =    5, loss = 9.7153)
Saving model (epoch =    6, loss = 6.3701)
Saving model (epoch =    7, loss = 5.1802)
Saving model (epoch =    8, loss = 4.4255)
Saving model (epoch =    9, loss = 3.8009)
Saving model (epoch =   10, loss = 3.3691)
Saving model (epoch =   11, loss = 3.0943)
Saving model (epoch =   12, loss = 2.8176)
Saving model (epoch =   13, loss = 2.6274)
Saving model (epoch =   14, loss = 2.4542)
Saving model (epoch =   15, loss = 2.3012)
Saving model (epoch =   16, loss = 2.1766)
Saving model (epoch =   17, loss = 2.0641)
Saving model (epoch =   18, loss = 1.9399)
Saving model (epoch =   19, loss = 1.8978)
Saving model (epoch =   20, loss = 1.7950)
Saving model (epoch =   21, loss = 1.7164)
Saving model (epoch =   22, loss = 1.6455)
Saving model (epoch =   23, loss = 1.5912)
Saving model (epoch =   24, loss = 1.5599)
Saving model (epoch =   25, loss = 1.5197)
Saving model (epoch =   26, loss = 1.4698)
Saving model (epoch =   27, loss = 1.4189)
Saving model (epoch =   28, loss = 1.3992)
Saving model (epoch =   29, loss = 1.3696)
Saving model (epoch =   30, loss = 1.3442)
Saving model (epoch =   31, loss = 1.3231)
Saving model (epoch =   32, loss = 1.2834)
Saving model (epoch =   33, loss = 1.2804)
Saving model (epoch =   34, loss = 1.2471)
Saving model (epoch =   36, loss = 1.2414)
Saving model (epoch =   37, loss = 1.2138)
Saving model (epoch =   38, loss = 1.2083)
Saving model (epoch =   41, loss = 1.1591)
Saving model (epoch =   42, loss = 1.1484)
Saving model (epoch =   44, loss = 1.1209)
Saving model (epoch =   47, loss = 1.1122)
Saving model (epoch =   48, loss = 1.0937)
Saving model (epoch =   50, loss = 1.0842)
Saving model (epoch =   53, loss = 1.0655)
Saving model (epoch =   54, loss = 1.0613)
Saving model (epoch =   57, loss = 1.0524)
Saving model (epoch =   58, loss = 1.0394)
Saving model (epoch =   60, loss = 1.0267)
Saving model (epoch =   63, loss = 1.0247)
Saving model (epoch =   66, loss = 1.0100)
Saving model (epoch =   70, loss = 0.9829)
```

```
Saving model (epoch =   72, loss = 0.9814)
Saving model (epoch =   73, loss = 0.9742)
Saving model (epoch =   75, loss = 0.9670)
Saving model (epoch =   78, loss = 0.9643)
Saving model (epoch =   79, loss = 0.9597)
Saving model (epoch =   85, loss = 0.9550)
Saving model (epoch =   86, loss = 0.9533)
Saving model (epoch =   90, loss = 0.9466)
Saving model (epoch =   92, loss = 0.9434)
Saving model (epoch =   93, loss = 0.9230)
Saving model (epoch =   95, loss = 0.9127)
Saving model (epoch =  104, loss = 0.9114)
Saving model (epoch =  107, loss = 0.8992)
Saving model (epoch =  110, loss = 0.8938)
Saving model (epoch =  116, loss = 0.8885)
Saving model (epoch =  124, loss = 0.8869)
Saving model (epoch =  128, loss = 0.8724)
Saving model (epoch =  139, loss = 0.8674)
Saving model (epoch =  146, loss = 0.8652)
Saving model (epoch =  156, loss = 0.8644)
Saving model (epoch =  159, loss = 0.8528)
Saving model (epoch =  167, loss = 0.8497)
Saving model (epoch =  173, loss = 0.8490)
Saving model (epoch =  176, loss = 0.8460)
Saving model (epoch =  178, loss = 0.8410)
Saving model (epoch =  182, loss = 0.8372)
Saving model (epoch =  199, loss = 0.8297)
Saving model (epoch =  212, loss = 0.8276)
Saving model (epoch =  235, loss = 0.8252)
Saving model (epoch =  238, loss = 0.8235)
Saving model (epoch =  251, loss = 0.8211)
Saving model (epoch =  253, loss = 0.8200)
Saving model (epoch =  258, loss = 0.8173)
Saving model (epoch =  284, loss = 0.8135)
Saving model (epoch =  308, loss = 0.8131)
Saving model (epoch =  312, loss = 0.8077)
Saving model (epoch =  324, loss = 0.8039)
Saving model (epoch =  400, loss = 0.8037)
Saving model (epoch =  404, loss = 0.8006)
Saving model (epoch =  466, loss = 0.7995)
Saving model (epoch =  492, loss = 0.7994)
Saving model (epoch =  525, loss = 0.7986)
Saving model (epoch =  561, loss = 0.7942)
Saving model (epoch =  584, loss = 0.7900)
Saving model (epoch =  665, loss = 0.7899)
Saving model (epoch =  667, loss = 0.7886)
Saving model (epoch =  717, loss = 0.7817)
Saving model (epoch =  776, loss = 0.7805)
```

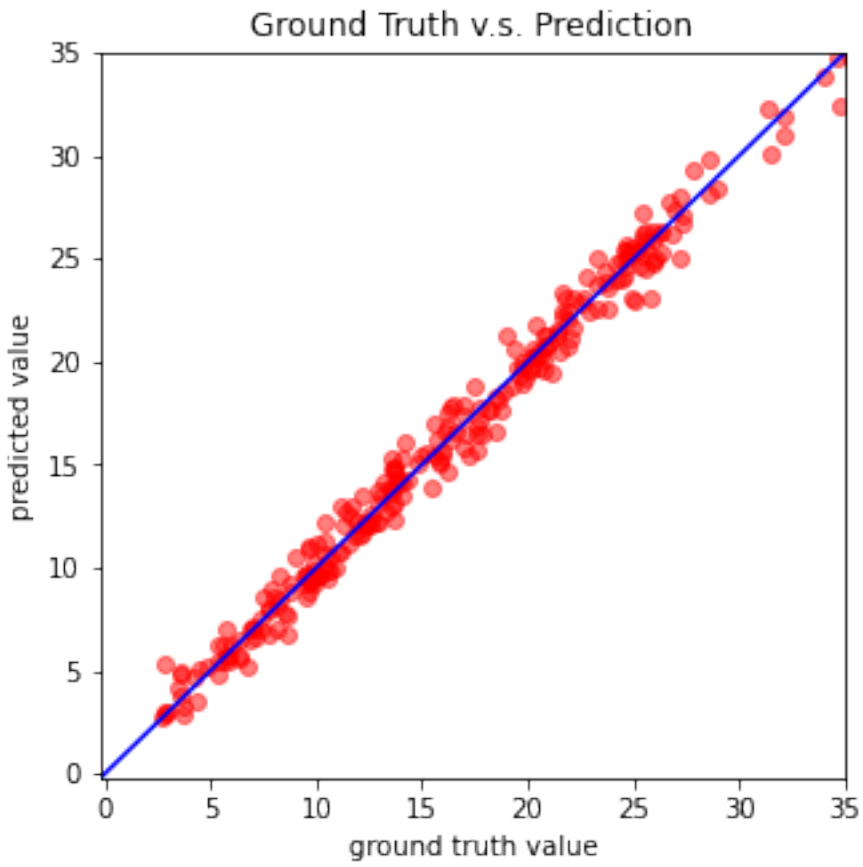
```
Saving model (epoch =  835, loss = 0.7803)
Saving model (epoch =  866, loss = 0.7768)
Saving model (epoch =  919, loss = 0.7764)
Saving model (epoch =  933, loss = 0.7740)
Saving model (epoch =  965, loss = 0.7697)
Saving model (epoch = 1027, loss = 0.7662)
Saving model (epoch = 1119, loss = 0.7640)
Saving model (epoch = 1196, loss = 0.7608)
Saving model (epoch = 1234, loss = 0.7596)
Saving model (epoch = 1243, loss = 0.7563)
Finished training after 1444 epochs
```

[15]:
```
# Plot

plot_learning_curve(model_loss_record, title='deep model')
```



[16]:
```
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
```



[17]:
```
# Testing

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
```