group-project-MLP-1

July 13, 2019

In [1]:
```
import torch
import pandas as pd
import numpy as np
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, utils
from torch.utils.data.sampler import SubsetRandomSampler

import matplotlib.pyplot as plt

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torchvision

device = torch.device('cuda:0')
```

In [2]:
```
class PUBG_dataset(Dataset):
    def __init__(self, csv_file, transform=None):
        self.frame = pd.read_csv(csv_file)
        self.transform = transform

    def __len__(self):
        return len(self.frame)

    def __getitem__(self, idx):
        # get one line in csv
        player_id = self.frame.iloc [idx, 0]
        player_stats = torch.tensor(self.frame.iloc [idx, [x for x in range(3, 27) if x
        win_place_perc = torch.tensor(self.frame.iloc [idx, 28])
        if self.transform:
            player_stats = self.transform(player_stats)
        sample = {
            "player_id": player_id,
            "player_stats": player_stats,
            "win_place_perc": win_place_perc
        }
        return sample
```

```
In [3]: def get_dataset(csv_file, train_dataset_size_ratio, batch_size):
            dataset = PUBG_dataset(csv_file)
            #  `torch.utils.data.random_split` meets server problem and lead to CRASH
            # see also:
            # - a denied fix PR for this problem: https://github.com/pytorch/pytorch/pull/9237
            #train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, t
            dataset_size = len(dataset)
            indices = list(range(dataset_size))
            split = int(np.floor((1-train_dataset_size_ratio) * dataset_size))
            train_indices, val_indices = indices[split:], indices[:split]

            train_sampler = SubsetRandomSampler(train_indices)
            valid_sampler = SubsetRandomSampler(val_indices)

            train_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, sampler=t
            test_loader = torch.utils.data.DataLoader(dataset, batch_size=batch_size, sampler=va
            print("load dataset: train dataset: {}, test dataset: {}.".format(len(train_loader)*
            return (train_loader, test_loader)

In [4]: class Net(torch.nn.Module):
            def __init__(self, n_feature, n_hidden, n_output):
                super().__init__()
                self.hidden_layer = torch.nn.Linear(n_feature, n_hidden)
                self.predict_layer = torch.nn.Linear(n_hidden, n_output)

            def forward(self, x):
                hidden_result = self.hidden_layer(x)
                relu_result = F.relu(hidden_result)
                predict_result = self.predict_layer(relu_result)
                return predict_result

In [17]: def train(model, train_loader, loss_func, optimizer, device):
             total_loss = 0
             # train the model using minibatch
             for i, data in enumerate(train_loader):
                 stats, prec = data['player_stats'], data['win_place_perc']
                 stats, prec = stats.to(torch.float32).to(device), prec.to(device)

                 # forward
                 outputs = model(stats)
                 loss = loss_func(outputs, prec)

                 # backward and optimize
                 optimizer.zero_grad()
                 loss.backward()
                 optimizer.step()

                 total_loss += loss.item()
```

```
            #if (i + 1) % 10 == 0:
            #    print ("Step [{}/{}] Train Loss: {:.4f}".format(i+1, len(train_loader), lo
        #print ("Train Loss: {:.4f}".format(loss.item()))
        return total_loss / len(train_loader)
```

In [6]:
```
def evaluate(model, val_loader, device):

    model.eval()
    with torch.no_grad():
        loss = 0
        total = 0

        for i, data in enumerate(val_loader):
            stats, prec = data['player_stats'], data['win_place_perc']
            stats, prec = stats.to(torch.float32).to(device), prec.to(device)

            outputs = model(stats)


            loss += (torch.abs(torch.t(outputs) - prec)).sum()
            total += prec.size(0)


        accuracy = loss / total
        #print('Test Loss: {:.4f}'.format(accuracy))
        return accuracy
```

In [7]:
```
def show_curve(ys, title):
    x = np.array(range(len(ys)))
    y = np.array(ys)
    plt.plot(x, y, c='b')
    plt.axis()
    plt.title('{} curve'.format(title))
    plt.xlabel('epoch')
    plt.ylabel('{}'.format(title))
    plt.show()
```

In [19]:
```
def fit(model, num_epochs, optimizer, device):
    loss_func = nn.MSELoss()
    model.to(device)
    if device == torch.device('cuda'):
        model = torch.nn.DataParallel(model)
        cudnn.benchmark = True
    loss_func.to(device)
    losses = []
    accs = []
```

```
            for epoch in range(num_epochs):

                loss = train(model, train_loader, loss_func, optimizer, device)
                losses.append(loss)

                accuracy = evaluate(model, test_loader, device)
                accs.append(accuracy)

                if (epoch+1) % 10 == 0:
                    print("Epoch {}/{}".format(epoch+1, num_epochs))
                    print("Train Loss: {:.4f}".format(loss))
                    print('Test Loss: {:.4f}'.format(accuracy))

            show_curve(losses, "train loss")
            show_curve(accs, "test accuracy")
```

```
In [9]: # load dataset
        csv_file = 'train_small.csv'
        train_dataset_size_ratio = 0.9
        batch_size = 128
        train_loader, test_loader = get_dataset(csv_file, train_dataset_size_ratio, batch_size)
```

```
load dataset: train dataset: 2048, test dataset: 1024.
```

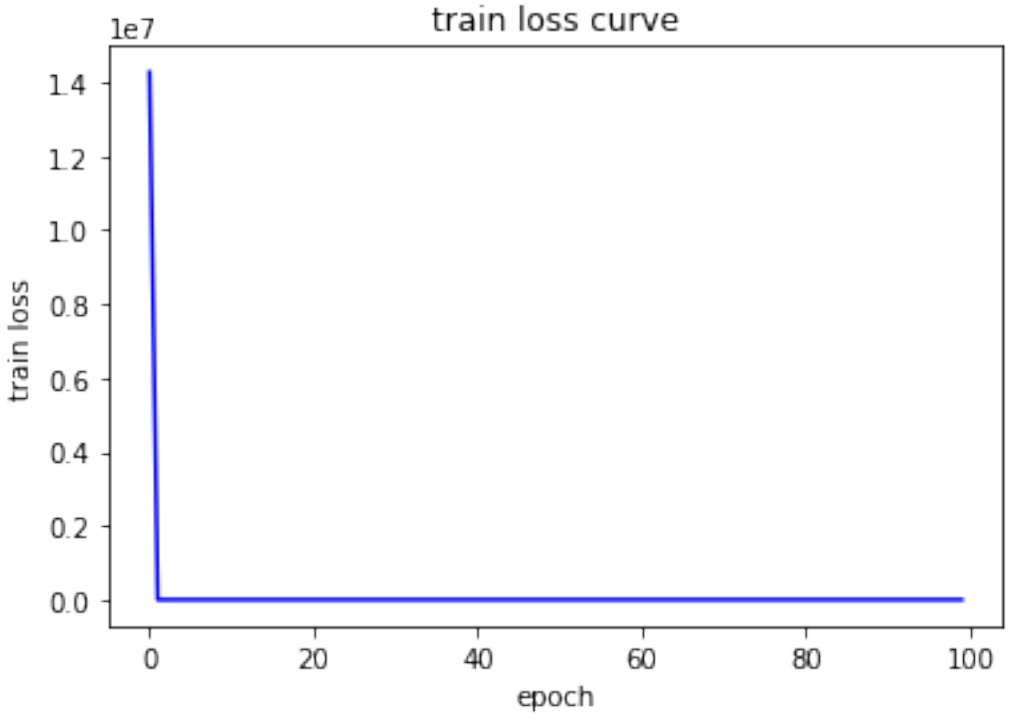
```
In [29]: # training setting
         epoches = 100
         lr = 1e-5
         device = torch.device('cuda:0')

         net = Net(23, 10, 1)
         optimizer = optim.SGD(net.parameters(), lr=lr, momentum=0.9)
         print(net)
```

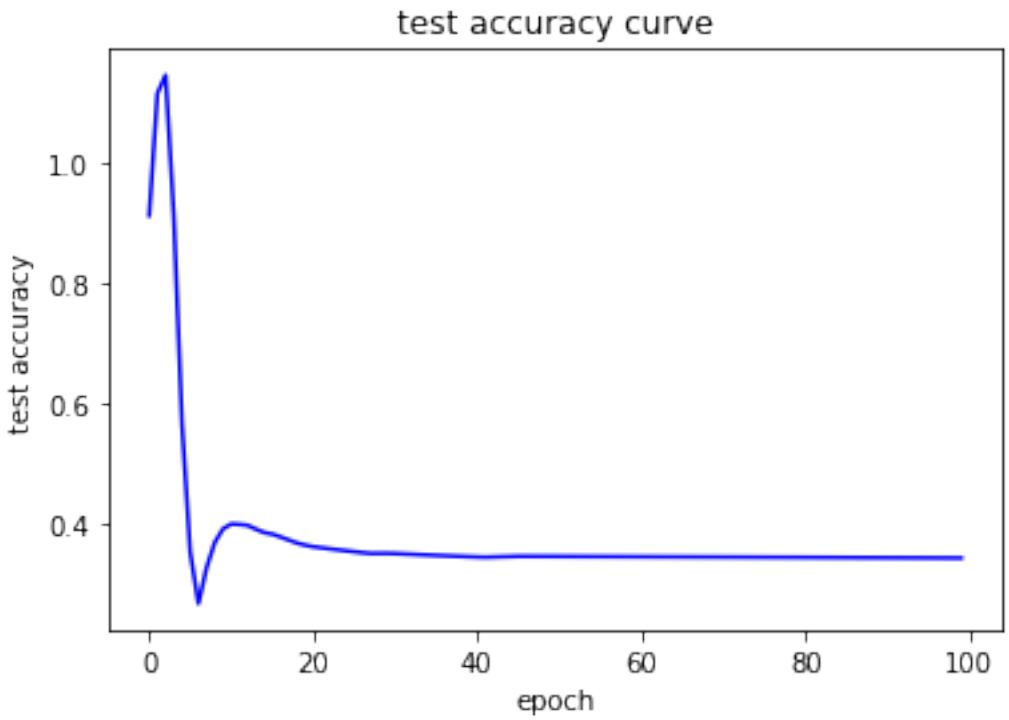
```
Net(
  (hidden_layer): Linear(in_features=23, out_features=10, bias=True)
  (predict_layer): Linear(in_features=10, out_features=1, bias=True)
)
```

```
In [30]: fit(net, epoches, optimizer, device)
```

```
Epoch 10/100
Train Loss: 0.1699
Test Loss: 0.3927
Epoch 20/100
Train Loss: 0.1676
Test Loss: 0.3657
Epoch 30/100
```

```
Train Loss: 0.1541
Test Loss: 0.3521
Epoch 40/100
Train Loss: 0.2058
Test Loss: 0.3465
Epoch 50/100
Train Loss: 0.1835
Test Loss: 0.3470
Epoch 60/100
Train Loss: 0.1371
Test Loss: 0.3465
Epoch 70/100
Train Loss: 0.1628
Test Loss: 0.3460
Epoch 80/100
Train Loss: 0.1825
Test Loss: 0.3455
Epoch 90/100
Train Loss: 0.1565
Test Loss: 0.3449
Epoch 100/100
Train Loss: 0.1411
Test Loss: 0.3444
```





In [60]:
```
class MLP(nn.Module):

    def __init__(self, input_size, hidden_size, out_size):
        super(MLP, self).__init__()
        self.relu = nn.ReLU()
        self.bn = nn.BatchNorm1d(100)
        self.fc1 = nn.Linear(input_size, 28)
        self.fc2 = nn.Linear(28, 28)
        self.fc3 = nn.Linear(28, 1)

    def forward(self, input):
        x = input
        x = self.relu(self.fc1(x))
        x = self.relu(self.fc2(x))
        #x = self.relu(self.fc3(x))
        return self.fc3(x)
```

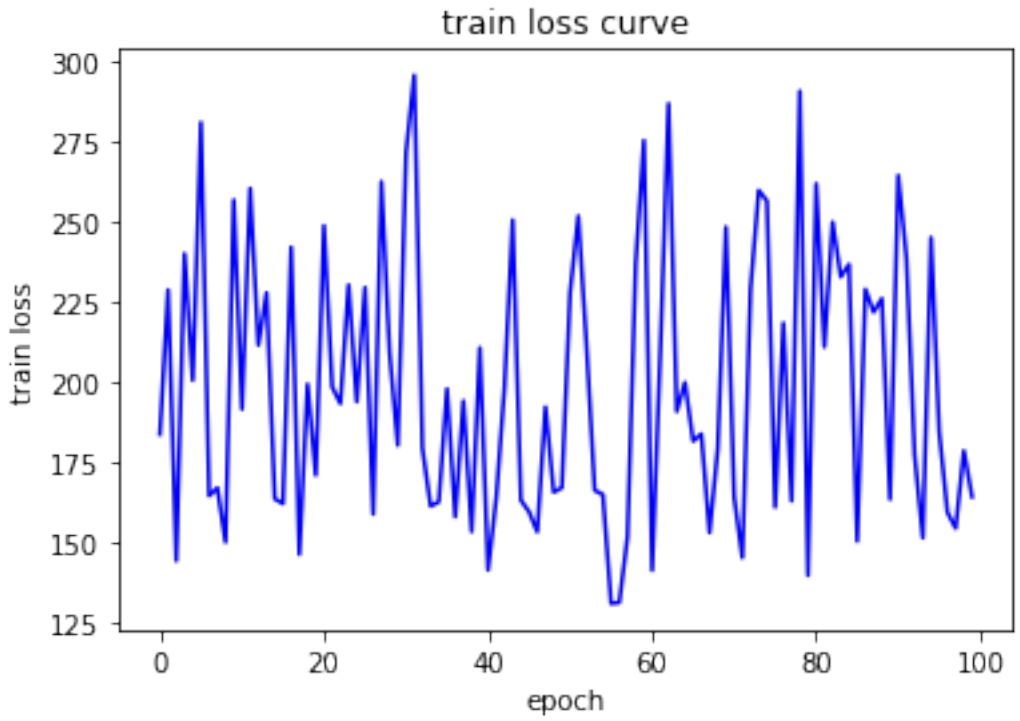
In [61]:
```
# training setting
epoches = 100
lr = 1e-6
batch_size = 1024
device = torch.device('cuda:0')

mlp_net = MLP(23, 23, 1).to(device)
```

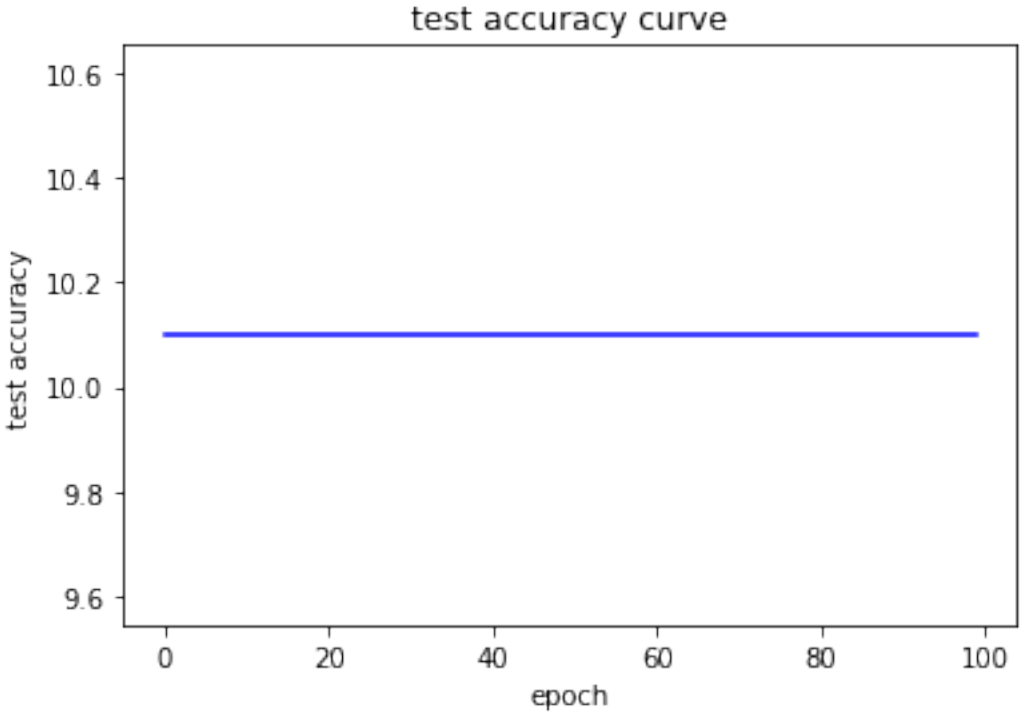
```
        optimizer = optim.SGD(net.parameters(), lr=lr, momentum=0.9)
        print(mlp_net)
```

```
MLP(
  (relu): ReLU()
  (bn): BatchNorm1d(100, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (fc1): Linear(in_features=23, out_features=28, bias=True)
  (fc2): Linear(in_features=28, out_features=28, bias=True)
  (fc3): Linear(in_features=28, out_features=1, bias=True)
)
```

In [62]:
```
fit(mlp_net, epoches, optimizer, device)
```

```
Epoch 10/100
Train Loss: 256.8290
Test Loss: 10.0996
Epoch 20/100
Train Loss: 171.1324
Test Loss: 10.0996
Epoch 30/100
Train Loss: 180.4366
Test Loss: 10.0996
Epoch 40/100
Train Loss: 210.8102
Test Loss: 10.0996
Epoch 50/100
Train Loss: 167.2534
Test Loss: 10.0996
Epoch 60/100
Train Loss: 275.3730
Test Loss: 10.0996
Epoch 70/100
Train Loss: 248.4081
Test Loss: 10.0996
Epoch 80/100
Train Loss: 139.9133
Test Loss: 10.0996
Epoch 90/100
Train Loss: 163.6002
Test Loss: 10.0996
Epoch 100/100
Train Loss: 164.3882
Test Loss: 10.0996
```


train loss curve
train loss
epoch
300
275
250
225
200
175
150
125
0
20
40
60
80
100


test accuracy curve
test accuracy
epoch
10.6
10.4
10.2
10.0
9.8
9.6
0
20
40
60
80
100