Pytorch Experience

Dataset

```
import torch
from torch.utils.data import Dataset

class MyDataset(Dataset):
    def __init__(self):
        pass
    def __getitem__(self, item):
        pass
    def __len__(self):
        pass
```

数据集设计需要考虑：

源数据存储　网络输入　数据处理　Batch

- `__getitem__`
  - 输出必须包括**标签**和**数据**，必要的话输出**字典**， pytorch会自动给字典中的元素加batch
  - 必须考虑 **batch**， 如果 $B > 1$ 则要将处理**数据的矩阵大小**
  - **数据处理（采样，数据增强）**等可以在此阶段进行

Module (network)

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class MyNet(nn.Module):
    def __init__(self):
        pass
    def forward(self, x):
        passs
```

- `数据输入矩阵大小`

  各个层都和数据矩阵大小有关，所以得考虑好输入的大小是多少，数据集必须对应地设计

Training

重要组成部分：dataset　net　criterion　optimizer　(scheduler)

```
# necessary import
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.utils.data # for data splitting and Dataloader
import torch.optim as optim # for optimizer

# template of training
def train():

    # dataset
    batch = 64

    dataset = MyDataset()
    train_len = int(len(dataset) * 0.8)
    train_data, valid_data = torch.utils.data.random_split(dataset, [train_len, len(dataset) - train_len])
    train_dataloader = torch.utils.data.DataLoader(train_data, batch)
    valid_dataloader = torch.utils.data.DataLoader(valid_data, batch)

    # net
    device = "cuda:%d" % 0
    net = MyNet()
    net.to(device)

    # loss function
    criterion = nn.CrossEntropyLoss()

    # optim
    optimizer = optim.Adam(net.parameters(), lr=0.01)

    # train
    epochs = 100
    for epoch in range(epochs):
        for i, data in enumerate(train_dataloader):
            # get data & label
            source, label = data[0].to(device), data[1].to(device)

            # go through net
            pred= net(source)

            optimizer.zero_grad()
            loss = criterion(pred, label.long())
            loss.backward()
            optimizer.step()
```

💡 损失函数用 `criterion` 和 `F` 作用一样，只不过前者封装了一下