LeNet-5 pytorch 实现

先安装pytorch： conda install pytorch torchvision torchaudio cpuonly -c pytorch

遇到下载静止时，考虑添加国内源： conda config --add channels https://mirrors.ustc.edu.cn/anaconda/pkgs/free/ (https://mirrors.ustc.edu.cn/anaconda/pkgs/free/)

In [1]:

```
import numpy as np
import torch
from torchvision.datasets import mnist
from torch.nn import CrossEntropyLoss
from torch.optim import SGD
from torch.utils.data import DataLoader
from torchvision.transforms import ToTensor

#每次同时计算256张图片
batch_size = 256
#导数训练集 有60000张图片和标签
train_dataset = mnist.MNIST(root='./train', train=True, transform=ToTensor())
#导入测试集 有10000张图片和标签
test_dataset = mnist.MNIST(root='./test', train=False, transform=ToTensor())
#按256每次分配数据
train_loader = DataLoader(train_dataset, batch_size=batch_size)
test_loader = DataLoader(test_dataset, batch_size=batch_size)
#总共重复训练10次，这里改成100精度会更高
epoch = 10

print(train_dataset)
print(test_dataset)
```

```
Dataset MNIST
    Number of datapoints: 60000
    Root location: ./train
    Split: Train
    StandardTransform
Transform: ToTensor()
Dataset MNIST
    Number of datapoints: 10000
    Root location: ./test
    Split: Test
    StandardTransform
Transform: ToTensor()
```

In [2]:

```
# build network
from torch.nn import Module
from torch import nn


class Model(Module):
    def __init__(self):
        super(Model, self).__init__()
        #定义卷积层，原图28*28 padding后28+2*2=32
        #卷积输入32*32，输出6@28*28
        self.conv1 = nn.Conv2d(1, 6, 5, stride=1, padding=2)
        #定义激活函数
        self.relu1 = nn.ReLU()
        #定义最大池化
        #输入6@28*28，输出6@14*14
        self.pool1 = nn.MaxPool2d(2)
        #输入6@14*14，输出16@10*10
        self.conv2 = nn.Conv2d(6, 16, 5, stride=1,padding=0)
        self.relu2 = nn.ReLU()
        #输入16@10*10，输出16@5*5
        self.pool2 = nn.MaxPool2d(2)
        #定义全连接层
        self.fc1 = nn.Linear(400, 120)
        self.relu3 = nn.ReLU()
        self.fc2 = nn.Linear(120, 84)
        self.relu4 = nn.ReLU()
        self.fc3 = nn.Linear(84, 10)
        self.relu5 = nn.ReLU()

    def forward(self, x):
        y = self.conv1(x)
        y = self.relu1(y)
        y = self.pool1(y)
        y = self.conv2(y)
        y = self.relu2(y)
        y = self.pool2(y)
        #调整成一维 16*5*5=400
        y = y.view(y.shape[0], -1)
        y = self.fc1(y)
        y = self.relu3(y)
        y = self.fc2(y)
        y = self.relu4(y)
        y = self.fc3(y)
        y = self.relu5(y)
        return y
model = Model()
sgd = SGD(model.parameters(), lr=1e-1)
cost = CrossEntropyLoss()
print(model)
```

```
Model(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (relu1): ReLU()
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (relu2): ReLU()
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=400, out_features=120, bias=True)
  (relu3): ReLU()
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (relu4): ReLU()
  (fc3): Linear(in_features=84, out_features=10, bias=True)
  (relu5): ReLU()
)
```

In [3]:

```
for _epoch in range(epoch):
    # train
    for idx, (train_x, train_label) in enumerate(train_loader):
        label_np = np.zeros((train_label.shape[0], 10)) #载入数据和标签
        sgd.zero_grad() #梯度清零
        predict_y = model(train_x.float()) #计算预测值
        loss = cost(predict_y, train_label.long()) #计算损失函数
        if idx % 10 == 0:
            print('idx: {}, loss: {}'.format(idx, loss.sum().item()))
        loss.backward() #误差反向传播
        sgd.step()

    correct = 0
    _sum = 0

    # test
    for idx, (test_x, test_label) in enumerate(test_loader):
        predict_y = model(test_x.float()).detach()#将预测图片输入模型并获得预测值
        predict_ys = np.argmax(predict_y, axis=-1)#用于返回一个numpy数组中最大值的索引值
        label_np = test_label.numpy() #取出真值
        truth = predict_ys == test_label #计算有几个正确的
        correct += np.sum(truth.numpy(), axis=-1)# 计算测试集总数
        _sum += truth.shape[0]

    print('accuracy: {:.2f}'.format(correct / _sum))
    torch.save(model, 'models/mnist_{:.2f}.pkl'.format(correct / _sum))
```

```
idx: 60, loss: 0.04866766929626465
idx: 70, loss: 0.03454180061817169
idx: 80, loss: 0.032987453043460846
idx: 90, loss: 0.01965368166565895
idx: 100, loss: 0.05873149633407593
idx: 110, loss: 0.029835600405931473

idx: 120, loss: 0.0416700653731823
idx: 130, loss: 0.03798697516322136
idx: 140, loss: 0.03571157157421112
idx: 150, loss: 0.04377454146742821
idx: 160, loss: 0.02403966337442398
idx: 170, loss: 0.06582824885845184
idx: 180, loss: 0.0722731202840805
idx: 190, loss: 0.01521842461079359
idx: 200, loss: 0.07573795318603516
idx: 210, loss: 0.037947118282318115
idx: 220, loss: 0.034353744238615036
idx: 230, loss: 0.0009964826749637723
accuracy: 0.98
```

```
In [ ]:
```