In [1]:

```
# 1 加载必要的库
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
```

In [2]:

```
# 2 定义超参数
BATCH_SIZE = 16 # 每批处理的数据
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu") # 是否用GPU
EPOCHS = 10 # 训练数据集的轮次
```

In [3]:

```
# 3 构建pipeline，对图像做处理
pipeline = transforms.Compose([
    transforms.ToTensor(), # 将图片转换成tensor
    transforms.Normalize((0.1307,), (0.3081,)) # 正则化：降低模型复杂度
])
```

In [4]:

```
# 4 下载、加载数据
from torch.utils.data import DataLoader

# 下载数据集
train_set = datasets.MNIST("data", train=True, download=True, transform=pipeline

test_set = datasets.MNIST("data", train=False, download=True, transform=pipeline

# 加载数据
train_loader = DataLoader(train_set, batch_size=BATCH_SIZE, shuffle=True)

test_loader = DataLoader(test_set, batch_size=BATCH_SIZE, shuffle=True)
```

In [5]:

```
## 插入代码，显示MNIST中的图片
with open("./data/MNIST/raw/train-images-idx3-ubyte", "rb") as f:
    file = f.read()
```

In [6]:

```
image1 = [int(str(item).encode('ascii'), 16) for item in file[16 : 16+784]]
```

In [7]:

```
import cv2
import numpy as np

image1_np = np.array(image1, dtype=np.uint8).reshape(28, 28, 1)

print(image1_np.shape)
```

```
(28, 28, 1)
```

In [8]:

```
cv2.imwrite("digit.jpg", image1_np)
```

Out[8]:

```
True
```

In [9]:

```
# 5 构建网络模型
class Digit(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 10, 5) # 1：灰度图片的通道， 10：输出通道，5：kerne
        self.conv2 = nn.Conv2d(10, 20, 3) # 10：输入通道，20：输出通道，3：Kernel
        self.fc1 = nn.Linear(20*10*10, 500) # 20*10*10：输入通道，500：输出通道
        self.fc2 = nn.Linear(500, 10) # 500：输入通道，10：输出通道

    def forward(self, x):
        input_size = x.size(0) # batch_size
        x = self.conv1(x) # 输入：batch*1*28*28 ，输出：batch*10*24*24 ( 28 - 5 +
        x = F.relu(x) # 保持shpae不变，输出：batch*10*24*24
        x = F.max_pool2d(x, 2, 2) # 输入：batch*10*24*24 输出：batch*10*12*12

        x = self.conv2(x) # 输入：batch*10*12*12 输出：batch*20*10*10  ( 12 - 3 +
        x = F.relu(x)

        x = x.view(input_size, -1) # 拉平， -1 自动计算维度，20*10*10= 2000

        x = self.fc1(x) # 输入：batch*2000 输出：batch*500
        x = F.relu(x) # 保持shpae不变

        x = self.fc2(x) # 输入：batch*500 输出：batch*10

        output = F.log_softmax(x, dim=1) # 计算分类后，每个数字的概率值

        return output
```

In [10]:

```
# 6 定义优化器
model = Digit().to(DEVICE)

optimizer = optim.Adam(model.parameters())
```

In [11]:

```
# 7 定义训练方法
def train_model(model, device, train_loader, optimizer, epoch):
    # 模型训练
    model.train()
    for batch_index, (data , target) in enumerate(train_loader):
        # 部署到DEVICE上去
        data, target = data.to(device), target.to(device)
        # 梯度初始化为0
        optimizer.zero_grad()
        # 训练后的结果
        output = model(data)
        # 计算损失
        loss = F.cross_entropy(output, target)
        # 反向传播
        loss.backward()
        # 参数优化
        optimizer.step()
        if batch_index % 3000 == 0:
            print("Train Epoch : {} \t Loss : {:.6f}".format(epoch, loss.item())
```

In [12]:

```
# 8 定义测试方法
def test_model(model, device, test_loader):
    # 模型验证
    model.eval()
    # 正确率
    correct = 0.0
    # 测试损失
    test_loss = 0.0
    with torch.no_grad(): # 不会计算梯度，也不会进行反向传播
        for data, target in test_loader:
            # 部署到device上
            data, target = data.to(device), target.to(device)
            # 测试数据
            output = model(data)
            # 计算测试损失
            test_loss += F.cross_entropy(output, target).item()
            # 找到概率值最大的下标
            pred = output.max(1, keepdim=True)[1] # 值，索引
            # pred = torch.max(ouput, dim=1)
            # pred = output.argmax(dim=1)
            # 累计正确的值
            correct += pred.eq(target.view_as(pred)).sum().item()
        test_loss /= len(test_loader.dataset)
        print("Test — Average loss : {:.4f}, Accuracy : {:.3f}\n".format(
            test_loss, 100.0 * correct / len(test_loader.dataset)))
```

In [13]:

```
# 9 调用 方法7 / 8
for epoch in range(1, EPOCHS + 1):
    train_model(model, DEVICE, train_loader, optimizer, epoch)
    test_model(model, DEVICE, test_loader)
```

```
Train Epoch : 1          Loss : 2.283712
Train Epoch : 1          Loss : 0.001297
Test —— Average loss : 0.0035, Accuracy : 98.280

Train Epoch : 2          Loss : 0.160084
Train Epoch : 2          Loss : 0.000344
Test —— Average loss : 0.0025, Accuracy : 98.750

Train Epoch : 3          Loss : 0.022007
Train Epoch : 3          Loss : 0.035978
Test —— Average loss : 0.0019, Accuracy : 99.000

Train Epoch : 4          Loss : 0.001649
Train Epoch : 4          Loss : 0.002896
Test —— Average loss : 0.0022, Accuracy : 98.940

Train Epoch : 5          Loss : 0.000423
Train Epoch : 5          Loss : 0.000009
Test —— Average loss : 0.0022, Accuracy : 99.080

Train Epoch : 6          Loss : 0.000153
Train Epoch : 6          Loss : 0.002014
Test —— Average loss : 0.0024, Accuracy : 99.000

Train Epoch : 7          Loss : 0.000057
Train Epoch : 7          Loss : 0.000008
Test —— Average loss : 0.0025, Accuracy : 99.010

Train Epoch : 8          Loss : 0.000140
Train Epoch : 8          Loss : 0.000002
Test —— Average loss : 0.0033, Accuracy : 98.880

Train Epoch : 9          Loss : 0.002539
Train Epoch : 9          Loss : 0.000686
Test —— Average loss : 0.0046, Accuracy : 98.600

Train Epoch : 10         Loss : 0.008171
Train Epoch : 10         Loss : 0.155714
Test —— Average loss : 0.0038, Accuracy : 98.850
```

In [ ]: