In [1]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchvision import datasets, transforms
import torchvision
from torch.autograd import Variable
import torch.utils.data as Data
```

In [2]:

```
train_set = torchvision.datasets.MNIST('./mnist', train=True,transform=torchvision.transforms.ToTen
train_loader = Data.DataLoader(dataset=train_set, batch_size=4, shuffle=True)

test_set = torchvision.datasets.MNIST('./mnist', train=False, transform=torchvision.transforms.ToTe
test_loader = Data.DataLoader(dataset=test_set, batch_size=4, shuffle=True)
```

In [3]:

```
class LeNet(nn.Module):
  def __init__(self):
    super(LeNet, self).__init__()
    self.conv1 = nn.Sequential(nn.Conv2d(1, 6, 3, 1, 2), nn.ReLU(),
                 nn.MaxPool2d(2, 2))

    self.conv2 = nn.Sequential(nn.Conv2d(6, 16, 5), nn.ReLU(),
                 nn.MaxPool2d(2, 2))

    self.fc1 = nn.Sequential(nn.Linear(16 * 5 * 5, 120),
                nn.BatchNorm1d(120), nn.ReLU())

    self.fc2 = nn.Sequential(
      nn.Linear(120, 84),
      nn.BatchNorm1d(84),
      nn.ReLU(),
      nn.Linear(84, 10))
        # 最后的结果一定要变为 10，因为数字的选项是 0 ~ 9

  def forward(self, x):
    x = self.conv1(x)
    x = self.conv2(x)
    x = x.view(x.size()[0], -1)
    x = self.fc1(x)
    x = self.fc2(x)
    return x
```

In [4]:

```
device = torch.device('cpu')
batch_size = 64
LR = 0.001

net = LeNet().to(device)
# 损失函数使用交叉熵
criterion = nn.CrossEntropyLoss()
# 优化函数使用 Adam 自适应优化算法
optimizer = optim.Adam(
  net.parameters(),
  lr=LR,
)

epoch = 1
if __name__ == '__main__':
  for epoch in range(epoch):
    sum_loss = 0.0
    for i, data in enumerate(train_loader):
      inputs, labels = data
      inputs, labels = Variable(inputs), Variable(labels)
      optimizer.zero_grad() #将梯度归零
      outputs = net(inputs) #将数据传入网络进行前向运算
      loss = criterion(outputs, labels) #得到损失函数
      loss.backward() #反向传播
      optimizer.step() #通过梯度做一步参数更新

      sum_loss += loss.item()
      if i % 100 == 99:
        print('[%d,%d] loss:%.03f' %
            (epoch + 1, i + 1, sum_loss / 100))
        sum_loss = 0.0
```

```
[1,100] loss:1.799
[1,200] loss:1.303
[1,300] loss:1.093
[1,400] loss:0.941
[1,500] loss:0.843
[1,600] loss:0.730
[1,700] loss:0.649
[1,800] loss:0.601
[1,900] loss:0.695
[1,1000] loss:0.622
[1,1100] loss:0.637
[1,1200] loss:0.652
[1,1300] loss:0.614
[1,1400] loss:0.602
[1,1500] loss:0.568
[1,1600] loss:0.470
[1,1700] loss:0.538
[1,1800] loss:0.492
[1,1900] loss:0.504
```

In [5]:

```
net.eval() #将模型变换为测试模式
correct = 0
total = 0
for data_test in test_loader:
    images, labels = data_test
    images, labels = Variable(images), Variable(labels)
    output_test = net(images)
    _, predicted = torch.max(output_test, 1)
    total += labels.size(0)
    correct += (predicted == labels).sum()
print("correct1: ", correct)
print("Test acc: {0}".format(correct.item() /len(test_set)))
```

```
correct1:  tensor(9680)
Test acc: 0.968
```