1.1 Building model and training model

In [ ]:

```
import torch
from torchvision import transforms, datasets
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from matplotlib import pyplot as plt

#import dataset
transform = transforms.ToTensor()
train_data = datasets.MNIST(root='data', train=True, download=True, transform=transform)
test_data = datasets.MNIST(root='data', train=False, download=True, transform=transform)
print("number of image in train_data:{} | no. of image in test_data: {}\n".format(len(train_data), len(test_data)))

#mini batch(each batch contain 60 images)
batch_size = 60 #each batch contain 60 images
trainset = torch.utils.data.DataLoader(train_data, batch_size=batch_size, shuffle=True)
testset = torch.utils.data.DataLoader(test_data, batch_size=batch_size, shuffle=True)

#define modle class
class MLP_Net (nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(28*28, 520)
        self.l2 = nn.Linear(520, 320)
        self.l3 = nn.Linear(320, 240)
        self.l4 = nn.Linear(240, 120)
        self.l5 = nn.Linear(120, 10)

    def forward(self, x):
        x = x.view(-1, 784) #flatten the data from(n,1,28,28) -> (n, 784)
        x = F.relu(self.l1(x))
        x = F.relu(self.l2(x))
        x = F.relu(self.l3(x))
        x = F.relu(self.l4(x))
        x = self.l5(x)
        return(x)

#make an instance of the model
model = MLP_Net()

# define loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.5)

def training_loop(n_epoch):
    for epoch in range(n_epoch):
        for batch_idx, (data, target) in enumerate(trainset):
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            if batch_idx % 100 == 0:
              print('Train Epoch: {} | Batch_idx: {} | Batch Status: {}/{} (
{:.0f}%) | Loss: {:.6f}'.format(
                    epoch, batch_idx,  batch_idx * len(data), len(train_data),
                    100. * batch_idx*len(data) / len(train_data), loss.item
()))
```

1.2 Model training

In [ ]:

```
#Model training
n_epoch =3
training_loop(n_epoch)
```

1.3 Prediction base on trained model

In [ ]:

```
i=1003
image = test_data[i][0]

#do prediction:
p = model(image.view(-1, 28*28))
print("\nPredict result is :\n", p)
print ("\nPredict value is {0}, Actual value is {1} : ".format(torch.argmax(p)
.data.numpy(), test_data[i][1] ))
plt.imshow(image.numpy()[0], cmap='gray')
```