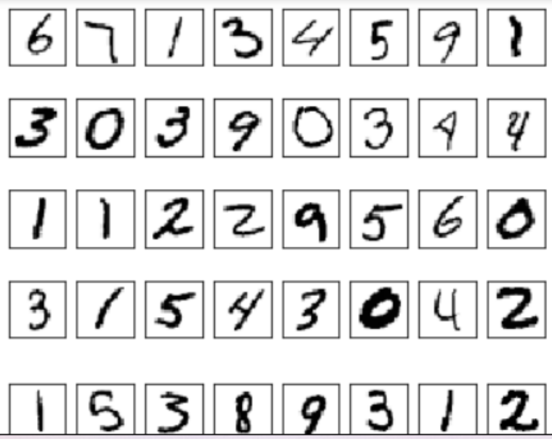
In [1]:
```python
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision.datasets import MNIST
from torchvision import transforms
from torch import nn
import math
import os
os.environ["KMP_DUPLICATE_LIB_OK"]="TRUE"
```

In [2]:
```python
# batch_size 超参，根据硬件配置相应大小
batch_size = 32 #批处理大小
trans_img = transforms.Compose([
 transforms.ToTensor(),
 transforms.Normalize((0.1307,), (0.3081,))
 ])
```

In [3]:
```python
# MNIST 数据集每张图片是灰度图片，大小为 28x28
trainset = MNIST('data', train=True, download=True, transform=trans_img)
testset = MNIST('data', train=False, download=True, transform=trans_img)
train_loader = DataLoader(trainset, batch_size=batch_size,
 shuffle=True, num_workers=1)
test_loader = DataLoader(testset, batch_size=batch_size,
 shuffle=True, num_workers=1)# 运行环境
```

In [4]:
```python
import matplotlib.pyplot as plt
plt.ion()
cnt = 0
for (img_batch, label) in train_loader:
    cnt += 1
    if cnt > 10:
        break
    fig, ax = plt.subplots(
        nrows=4,
        ncols=8,
        sharex=True,
        sharey=True, )
    ax = ax.flatten()
    for i in range(32):
        img = img_batch[i].numpy().reshape(28, 28)
        ax[i].imshow(img, cmap='Greys', interpolation='nearest')
    ax[0].set_xticks([])
    ax[0].set_yticks([])
    plt.show()
    plt.close()
plt.ioff()
```



In [5]:
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()

        self.features = nn.Sequential(
            nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True), #inplace = True ,
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(p = 0.5),
            nn.Linear(64 * 7 * 7, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(p = 0.5),
            nn.Linear(512, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(p = 0.5),
            nn.Linear(512, 10),
        )

        for m in self.features.children():
            if isinstance(m, nn.Conv2d):
                n = m.kernel_size[0] * m.kernel_size[1] * m.out_channels
                m.weight.data.normal_(0, math.sqrt(2. / n))
            elif isinstance(m, nn.BatchNorm2d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
        for m in self.classifier.children():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform(m.weight)
            elif isinstance(m, nn.BatchNorm1d):
                m.weight.data.fill_(1)
                m.bias.data.zero_()
    def forward(self, x):
        x = self.features(x)
        x = x.view(x.size(0), -1) #这句话是说将最后一次卷积的输出拉伸为一行
        x = self.classifier(x)
        return x
```

In [6]:

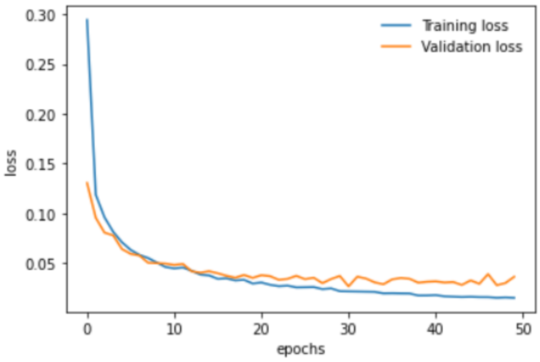
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
from torch import optim #优化器
from torch.autograd import Variable
import warnings
warnings.filterwarnings("ignore",category=UserWarning)
model = Net().to(device) #GUP
learning_rate = 0.001
criterion = nn.CrossEntropyLoss(size_average=False)
optimizer = optim.SGD(model.parameters(), lr = learning_rate)
# 总的训练轮数
epochs = 50
train_losses=[]
test_losses=[]
for epoch in range(epochs):
    running_loss, running_acc = 0., 0.
    for (img, label) in train_loader:
        img = Variable(img).to(device)
        label = Variable(label).to(device)

        optimizer.zero_grad() #梯度归零;

        output = model(img) ##前向传播
        loss = criterion(output, label) #计算 loss

        loss.backward() #反向传播, 计算当前梯度;
        optimizer.step() #反向传播, 计算当前梯度;
        running_loss += loss.item() #item () 取出张量具体位置的元素元素值
        _, predict = torch.max(output, 1)
 #_,predicted = torch.max(outputs.data,dim): 返回最大值所在索引,  dim=1 时, 按行返回最大值所在索引
        correct_num = (predict == label).sum()
        running_acc += correct_num.item()
```

```python
    running_loss /= len(trainset)
    running_acc /= len(trainset)

    with torch.no_grad():#不求梯度
        test_loss, test_acc = 0., 0.
        for images, labels in test_loader:
            images = Variable(images).to(device)
            labels = Variable(labels).to(device)
            output = model(images)
            loss = criterion(output, labels)
            test_loss += loss.item()
            _, predict = torch.max(output, 1)
            correct_num = (predict == labels).sum()
            test_acc += correct_num.item()

    test_loss /=len(testset)
    test_acc /=len(testset)

    train_losses.append(running_loss)
    test_losses.append(test_loss)
    print("Epoch: {}/{}.. ".format(epoch+1, epochs),
          "Training Loss: {:.3f}.. ".format(train_losses[-1]),
          "Training Accuracy: {:.3f} %".format(100*running_acc),
          "Test Loss: {:.3f}.. ".format(test_losses[-1]),
          "Test Accuracy: {:.3f} %".format(100*test_acc))
# 保存模型
torch.save(model, 'conv.pth.tar')
```

```
Epoch: 1/50..  Training Loss: 0.294..  Training Accuracy: 90.587 % Test Loss: 0.131..  Test Accuracy: 95.920 %
Epoch: 2/50..  Training Loss: 0.119..  Training Accuracy: 96.377 % Test Loss: 0.096..  Test Accuracy: 96.920 %
Epoch: 3/50..  Training Loss: 0.096..  Training Accuracy: 97.008 % Test Loss: 0.081..  Test Accuracy: 97.530 %
Epoch: 4/50..  Training Loss: 0.081..  Training Accuracy: 97.527 % Test Loss: 0.078..  Test Accuracy: 97.540 %
Epoch: 5/50..  Training Loss: 0.071..  Training Accuracy: 97.815 % Test Loss: 0.064..  Test Accuracy: 97.950 %
Epoch: 6/50..  Training Loss: 0.063..  Training Accuracy: 98.033 % Test Loss: 0.059..  Test Accuracy: 98.080 %
Epoch: 7/50..  Training Loss: 0.058..  Training Accuracy: 98.212 % Test Loss: 0.058..  Test Accuracy: 98.200 %
Epoch: 8/50..  Training Loss: 0.055..  Training Accuracy: 98.318 % Test Loss: 0.050..  Test Accuracy: 98.440 %
Epoch: 9/50..  Training Loss: 0.051..  Training Accuracy: 98.450 % Test Loss: 0.050..  Test Accuracy: 98.280 %
Epoch: 10/50..  Training Loss: 0.046..  Training Accuracy: 98.548 % Test Loss: 0.049..  Test Accuracy: 98.460 %
Epoch: 11/50..  Training Loss: 0.045..  Training Accuracy: 98.655 % Test Loss: 0.048..  Test Accuracy: 98.440 %
Epoch: 12/50..  Training Loss: 0.046..  Training Accuracy: 98.582 % Test Loss: 0.049..  Test Accuracy: 98.590 %
Epoch: 13/50..  Training Loss: 0.042..  Training Accuracy: 98.687 % Test Loss: 0.042..  Test Accuracy: 98.710 %
Epoch: 14/50..  Training Loss: 0.038..  Training Accuracy: 98.802 % Test Loss: 0.040..  Test Accuracy: 98.790 %
Epoch: 15/50..  Training Loss: 0.038..  Training Accuracy: 98.823 % Test Loss: 0.042..  Test Accuracy: 98.750 %
Epoch: 16/50..  Training Loss: 0.034..  Training Accuracy: 98.923 % Test Loss: 0.040..  Test Accuracy: 98.860 %
Epoch: 17/50..  Training Loss: 0.035..  Training Accuracy: 98.962 % Test Loss: 0.037..  Test Accuracy: 98.770 %
Epoch: 18/50..  Training Loss: 0.033..  Training Accuracy: 99.000 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 19/50..  Training Loss: 0.033..  Training Accuracy: 98.953 % Test Loss: 0.038..  Test Accuracy: 98.840 %
Epoch: 20/50..  Training Loss: 0.029..  Training Accuracy: 99.068 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 21/50..  Training Loss: 0.031..  Training Accuracy: 99.077 % Test Loss: 0.038..  Test Accuracy: 98.800 %
Epoch: 22/50..  Training Loss: 0.028..  Training Accuracy: 99.103 % Test Loss: 0.037..  Test Accuracy: 98.900 %
Epoch: 23/50..  Training Loss: 0.027..  Training Accuracy: 99.168 % Test Loss: 0.033..  Test Accuracy: 98.980 %
Epoch: 24/50..  Training Loss: 0.027..  Training Accuracy: 99.152 % Test Loss: 0.034..  Test Accuracy: 98.970 %
Epoch: 25/50..  Training Loss: 0.026..  Training Accuracy: 99.187 % Test Loss: 0.037..  Test Accuracy: 98.850 %
Epoch: 26/50..  Training Loss: 0.026..  Training Accuracy: 99.210 % Test Loss: 0.034..  Test Accuracy: 99.010 %
Epoch: 27/50..  Training Loss: 0.026..  Training Accuracy: 99.157 % Test Loss: 0.035..  Test Accuracy: 98.870 %
Epoch: 28/50..  Training Loss: 0.024..  Training Accuracy: 99.250 % Test Loss: 0.030..  Test Accuracy: 99.040 %
Epoch: 29/50..  Training Loss: 0.025..  Training Accuracy: 99.218 % Test Loss: 0.034..  Test Accuracy: 98.960 %
Epoch: 30/50..  Training Loss: 0.022..  Training Accuracy: 99.315 % Test Loss: 0.037..  Test Accuracy: 98.840 %
Epoch: 31/50..  Training Loss: 0.022..  Training Accuracy: 99.253 % Test Loss: 0.027..  Test Accuracy: 99.120 %
```

In [7]:
```python
plt.plot(train_losses, label='Training loss')
plt.plot(test_losses, label='Validation loss')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.legend(frameon=False)
```

Out[7]: <matplotlib.legend.Legend at 0x23ae6452c70>



In [9]:
```python
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = torch.load('conv.pth.tar')
print('testing cnn model')
testloss, testacc = 0., 0.
for (img, label) in test_loader:
    img = Variable(img).to(device)
    label = Variable(label).to(device)
    out = model(img)
    loss = criterion(out, label)
    testloss += loss.item()
    _, predict = torch.max(out, 1)
    correct_num = (predict == label).sum()
    testacc += correct_num.item()
testloss /= len(testset)
testacc /= len(testset)
print('cnn model, Test: Loss: %.5f, Acc: %.2f' %
      (testloss, 100 * testacc))
```

```
testing cnn model
cnn model, Test: Loss: 0.03346, Acc: 99.09
```