In [4]:

```
import torch
import torchvision
from torchvision import transforms, datasets
```

In [5]:

```
train = datasets.MNIST("", train=True, download=True,
                transform = transforms.Compose([transforms.ToTensor()]))

test = datasets.MNIST("", train=False, download=True,
                transform = transforms.Compose([transforms.ToTensor()]))
```

In [6]:

```
train
```

Out[6]:

```
Dataset MNIST
    Number of datapoints: 60000
    Root location:
    Split: Train
    StandardTransform
Transform: Compose(
               ToTensor()
           )
```

In [7]:

```
trainset = torch.utils.data.DataLoader(train, batch_size=10, shuffle=True)

testset = torch.utils.data.DataLoader(test, batch_size=10, shuffle=True)
```

In [8]:

```
for data in trainset:
    print(data)
    break
```

```
[tensor([[[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]],


         [[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]],


         [[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]],


         ...,


         [[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]],


         [[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]],


         [[[0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           ...,
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.],
           [0., 0., 0.,  ..., 0., 0., 0.]]]]), tensor([1, 1, 2, 6, 3, 2,
7, 2, 4, 7])]
```

In [9]:

```
x,y = data[0][0], data[1][0]

print(x,y)
```

```
tensor([[[0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.4000, 0.8314, 0.0000, 0.0000, 0.0000, 0.4745, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.094
1,
          0.9569, 0.8275, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.568
6,
          0.9961, 0.5490, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.1176, 0.925
5,
          0.9294, 0.0627, 0.0000, 0.0000, 0.0000, 0.0118, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
```

```
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.5098, 0.996
   1,
               0.6941, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.6941, 0.996
   1,
               0.4275, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.3922, 0.9961, 0.964
   7,
               0.1333, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.6980, 0.9961, 0.627
   5,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.1765, 0.9098, 0.9961, 0.302
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2471, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.8118, 0.9961, 0.8902, 0.082
   4,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.3373, 0.9686, 0.9961, 0.4824, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.8431, 0.9961, 0.9020, 0.0392, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
               0.0000, 0.0000, 0.0000, 0.0000],
              [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.3843, 0.9961, 0.9961, 0.3294, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0431, 0.8353, 0.9961, 0.7647, 0.0627, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.3490, 0.9961, 0.9373, 0.1216, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.7569, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0353, 0.8392, 1.0000, 0.7333, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.2157, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.2353, 0.9961, 0.9961, 0.1216, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.7059, 0.9961, 0.7804, 0.0157, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.9843, 0.9490, 0.1804, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.8392, 0.7608, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000],
            [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
   0,
             0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
```

```
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000],
         [0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.000
0,
          0.0000, 0.0000, 0.0000, 0.0000]]]) tensor(1)
```

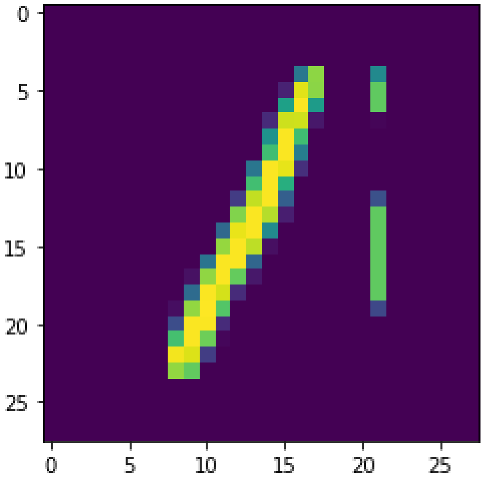
In [10]:

```
import matplotlib.pyplot as plt

plt.imshow(x.view(28,28))
```

Out[10]:

```
<matplotlib.image.AxesImage at 0x7f8a41e36490>
```



In [11]:

```
total = 0
counter = {0:0,1:0,2:0,3:0,4:0,5:0,6:0,7:0,8:0,9:0}

for data in trainset:
    xs, ys = data
    for y in ys:
        counter[int(y)] += 1
        total += 1
print(list(map(lambda x: x/total, counter.values())))
# data is roughly balanced
```

```
[0.09871666666666666, 0.11236666666666667, 0.0993, 0.10218333333333333,
0.09736666666666667, 0.09035, 0.09863333333333334, 0.10441666666666667,
0.09751666666666667, 0.09915]
```

In [12]:

```
import torch.nn as nn
import torch.nn.functional as F
```

In [13]:

```
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.fc1 = nn.Linear(28*28, 64)
        self.fc2 = nn.Linear(64, 64)
        self.fc3 = nn.Linear(64, 64)
        self.fc4 = nn.Linear(64, 10)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = F.relu(self.fc3(x))
        x = self.fc4(x)
        return F.log_softmax(x, dim=1)

net = Net()
print(net)
```

```
Net(
  (fc1): Linear(in_features=784, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=64, bias=True)
  (fc4): Linear(in_features=64, out_features=10, bias=True)
)
```

In [14]:

```
X = torch.rand((28,28))
```

In [15]:

```
output = net(X.view(1,28*28))
output
```

Out[15]:

```
tensor([[-2.4057, -2.3311, -2.2866, -2.2854, -2.3012, -2.3534, -2.2225,
-2.2857,
         -2.2391, -2.3282]], grad_fn=<LogSoftmaxBackward>)
```

In [16]:

```
import torch.optim as optim

optimizer = optim.Adam(net.parameters(), lr=0.001)

EPOCHS = 3

for epoch in range(EPOCHS):
    for data in trainset:
        # data is a batch of featuresets and lables
        X, y = data
        #
        net.zero_grad()
        output = net(X.view(-1, 28*28))
        loss = F.nll_loss(output, y)
        loss.backward()
        optimizer.step()
    print(loss)
```

```
tensor(0.3145, grad_fn=<NllLossBackward>)
tensor(0.0077, grad_fn=<NllLossBackward>)
tensor(0.0746, grad_fn=<NllLossBackward>)
```

In [20]:

```
correct = 0
total = 0

with torch.no_grad():
    for data in testset:
        X,y = data
        output = net(X.view(-1,28*28))
        for idx, i in enumerate(output):
            if torch.argmax(i) == y[idx]:
                correct += 1
            total += 1
print("Accuracy: ", round(correct/total, 3))
```

```
Accuracy:  0.967
```