In [1]:

```
from fastai.vision.all import *
```

In [2]:

```
path = untar_data(URLs.MNIST_SAMPLE)
```

In [3]:

```
Path.BASE_PATH = path
```

In [4]:

```
path.ls()
```

Out[4]:

```
(#3) [Path('train'),Path('valid'),Path('labels.csv')]
```

In [5]:

```
threes = (path/'train'/'3').ls().sorted()
sevens = (path/'train'/'7').ls().sorted()
```

In [6]:

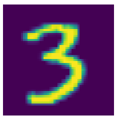
```
seven_tensors = [tensor(Image.open(o)) for o in sevens]
three_tensors = [tensor(Image.open(o)) for o in threes]
len(three_tensors),len(seven_tensors)
```

Out[6]:

```
(6131, 6265)
```

In [7]:

```
show_image(three_tensors[1]);
```



In [8]:

```
stacked_sevens = torch.stack(seven_tensors).float()/255
stacked_threes = torch.stack(three_tensors).float()/255
stacked_threes.shape
```

Out[8]:

```
torch.Size([6131, 28, 28])
```

In [9]:

```
train_x = torch.cat([stacked_threes, stacked_sevens]).view(-1, 28*28)
train_y = tensor([1]*len(threes) + [0]*len(sevens)).unsqueeze(1)
train_x.shape,train_y.shape
```

Out[9]:

```
(torch.Size([12396, 784]), torch.Size([12396, 1]))
```

In [10]:

```
dset = list(zip(train_x,train_y))
x,y = dset[0]
x.shape,y
```

Out[10]:

```
(torch.Size([784]), tensor([1]))
```

In [11]:

```
valid_3_tens = torch.stack([tensor(Image.open(o)) for o in (path/'valid'/'3')
valid_3_tens = valid_3_tens.float()/255
valid_7_tens = torch.stack([tensor(Image.open(o)) for o in (path/'valid'/'7')
valid_7_tens = valid_7_tens.float()/255
valid_3_tens.shape,valid_7_tens.shape
```

Out[11]:

```
(torch.Size([1010, 28, 28]), torch.Size([1028, 28, 28]))
```

In [12]:

```
valid_x = torch.cat([valid_3_tens, valid_7_tens]).view(-1, 28*28)
valid_y = tensor([1]*len(valid_3_tens) + [0]*len(valid_7_tens)).unsqueeze(1)
valid_dset = list(zip(valid_x,valid_y))
```

In [13]:

```
dl = DataLoader(dset, batch_size=256)
xb,yb = first(dl)
xb.shape,yb.shape
```

Out[13]:

```
(torch.Size([256, 784]), torch.Size([256, 1]))
```

In [14]:

```
valid_dl = DataLoader(valid_dset, batch_size=256)
```

In [15]:

```
data_loaders = DataLoaders(dl, valid_dl)
```

In [16]:

```
def train_epoch(model):
    for xb,yb in dl:
        calc_grad(xb, yb, model)
        opt.step()
        opt.zero_grad()
```

In [17]:

```
def train_model(model, epochs):
    for i in range(epochs):
        train_epoch(model)
        print(validate_epoch(model), end=' ')
```

In [18]:

```
def init_params(size, std=1.0):
    return (torch.randn(size)*std).requires_grad_()
```

In [19]:

```
def mnist_loss(predictions, targets):
    predictions = predictions.sigmoid()
    return torch.where(targets==1, 1-predictions, predictions).mean()
```

In [20]:

```
def calc_grad(xb, yb, model):
    preds = model(xb)
    loss = mnist_loss(preds, yb)
    loss.backward()
```

In [21]:

```
def validate_epoch(model):
    accs = [batch_accuracy(model(xb), yb) for xb,yb in valid_dl]
    return round(torch.stack(accs).mean().item(), 4)
```

In [22]:

```
def batch_accuracy(xb, yb):
    preds = xb.sigmoid()
    correct = (preds>0.5) == yb
    return correct.float().mean()
```

In [23]:

```
class BasicOptim:
    def __init__(self, params, lr):
        self.params,self.lr = list(params),lr

    def step(self, *args, **kwargs):
        for p in self.params: p.data -= p.grad.data * self.lr

    def zero_grad(self, *args, **kwargs):
        for p in self.params: p.grad = None
```

In [24]:

```
class LinearModel:
    """A simple linear model."""
    def __init__(self, in_features, out_features):
        self.weights = init_params((in_features, out_features))  # torch.Size
        self.bias = init_params(out_features)  # torch.Size([out_features])

    def parameters(self):
        return (self.weights, self.bias)

    def __call__(self, xb):
        return (xb @ self.weights) + self.bias
```

In [25]:

```
class SimpleNet:
    """A simple multi layer neural network."""
    def __init__(self, in_features, out_features):
        self.layer1 = LinearModel(in_features, 30)
        self.layer2 = lambda xb: xb.max(tensor(0.0))
        self.layer3 = LinearModel(30, out_features)

    def parameters(self):
        w1, b1 = self.layer1.parameters()
        w2, b2 = self.layer3.parameters()
        return (w1, b1, w2, b2)

    def __call__(self, xb):
        res = self.layer1(xb)
        res = self.layer2(res)
        res = self.layer3(res)
        return res
```

In [26]:

```
# learner = SimpleLearner(data_loaders, LinearModel(28*28, 1))
# learner.train_model(20, learning_rate=1.0)
```

In [ ]:

In [27]:

```
model = LinearModel(28*28, 1)
```

In [28]:

```
opt = BasicOptim(model.parameters(), lr=1.0)
```

In [29]:

```
train_model(model=model, epochs=20)
```

```
0.7025 0.8549 0.9154 0.9413 0.9525 0.9574 0.9594 0.9618 0.9647 0.9667 0.9691
0.9696 0.9701 0.9701 0.9711 0.972 0.9716 0.9725 0.973 0.973
```

In [ ]:

```
```

In [30]:

```
model = SimpleNet(28*28, 1)
```

In [31]:

```
opt = BasicOptim(model.parameters(), lr=1.0)
```

In [32]:

```
train_model(model=model, epochs=20)
```

```
0.5703 0.8032 0.8984 0.9306 0.9438 0.9521 0.9526 0.9599 0.9604 0.9653 0.9687
0.9717 0.9726 0.9721 0.9731 0.9736 0.9746 0.9741 0.9746 0.9746
```

In [ ]:

```
```