Watson Studio democratizes machine learning and deep learning to accelerate infusion of AI in your business to drive innovation. Watson Studio provides a suite of tools and a collaborative environment for data scientists, developers and domain experts.

(http://cocl.us/pytorch_link_top)

Softmax Classifier

Table of Contents

In this lab, you will use a single layer Softmax to classify handwritten digits from the MNIST database.

Estimated Time Needed: **25 min**

Preparation

We'll need the following libraries

In [2]:

```
# Import the libraries we need for this lab

# Using the following line code to install the torchvision library
# !conda install -y torchvision

import torch
import torch.nn as nn
import torchvision.transforms as transforms
import torchvision.datasets as dsets
import matplotlib.pylab as plt
import numpy as np
```

Use the following function to plot out the parameters of the Softmax function:

In [3]:

```
# The function to plot parameters

def PlotParameters(model):
    W = model.state_dict()['linear.weight'].data
    w_min = W.min().item()
    w_max = W.max().item()
    fig, axes = plt.subplots(2, 5)
    fig.subplots_adjust(hspace=0.01, wspace=0.1)
    for i, ax in enumerate(axes.flat):
        if i < 10:

            # Set the label for the sub-plot.
            ax.set_xlabel("class: {0}".format(i))

            # Plot the image.
            ax.imshow(W[i, :].view(28, 28), vmin=w_min, vmax=w_max, cmap='seismic')

            ax.set_xticks([])
            ax.set_yticks([])

        # Ensure the plot is shown correctly with multiple plots
        # in a single Notebook cell.
    plt.show()
```

Use the following function to visualize the data:

In [4]:

```
# Plot the data

def show_data(data_sample):
    plt.imshow(data_sample[0].numpy().reshape(28, 28), cmap='gray')
    plt.title('y = ' + str(data_sample[1].item()))
```

Make Some Data

Load the training dataset by setting the parameters `train` to `True` and convert it to a tensor by placing a transform object in the argument `transform`.

In [5]:

```
# Create and print the training dataset

train_dataset = dsets.MNIST(root='./data', train=True, download=True, transform=tra
nsforms.ToTensor())
print("Print the training dataset:\n ", train_dataset)
```

```
Print the training dataset:
  Dataset MNIST
    Number of datapoints: 60000
    Split: train
    Root Location: ./data
    Transforms (if any): ToTensor()
    Target Transforms (if any): None
```

Load the testing dataset by setting the parameters `train` to `False` and convert it to a tensor by placing a transform object in the argument `transform`.

In [6]:

```
# Create and print the validating dataset

validation_dataset = dsets.MNIST(root='./data', train=False, download=True, transfo
rm=transforms.ToTensor())
print("Print the validating dataset:\n ", validation_dataset)
```

```
Print the validating dataset:
  Dataset MNIST
    Number of datapoints: 10000
    Split: test
    Root Location: ./data
    Transforms (if any): ToTensor()
    Target Transforms (if any): None
```

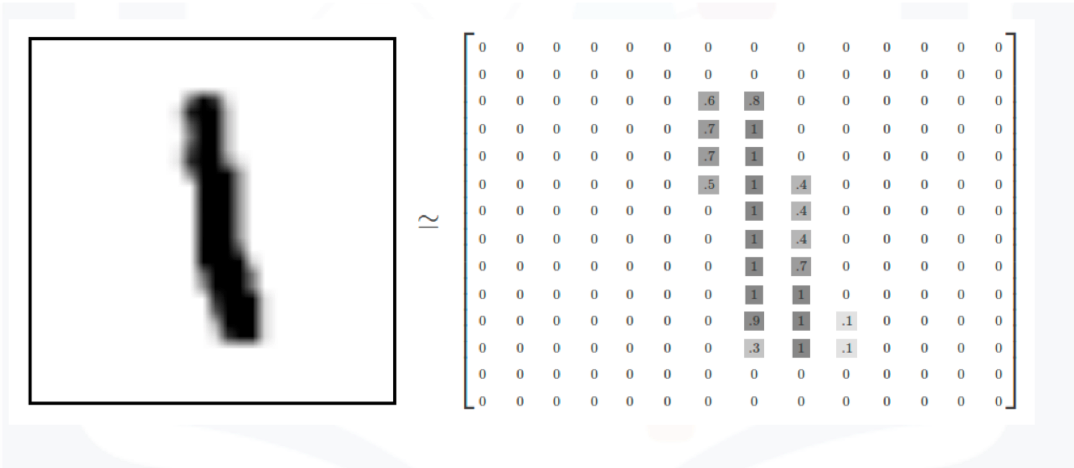
You can see that the data type is long:

In [7]:

```
# Print the type of the element

print("Type of data element: ", train_dataset[0][1].type())
```

```
Type of data element:  torch.LongTensor
```

Each element in the rectangular tensor corresponds to a number that represents a pixel intensity as demonstrated by the following image:



In this image, the values are inverted i.e back represents wight.

Print out the label of the fourth element:

In [8]:

```
# Print the label

print("The label: ", train_dataset[3][1])
```

```
The label:  tensor(1)
```

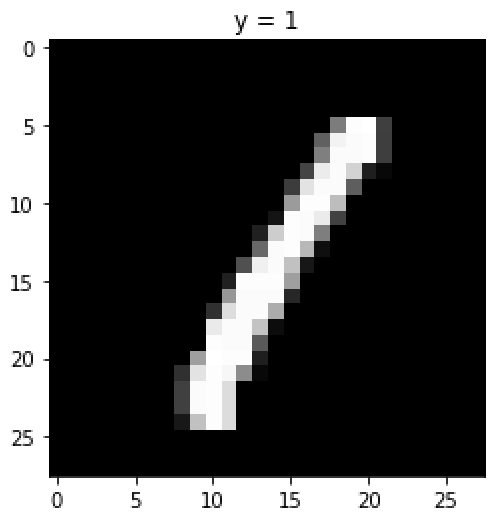
The result shows the number in the image is 1

Plot the fourth sample:

In [9]:

```
# Plot the image

print("The image: ", show_data(train_dataset[3]))
```

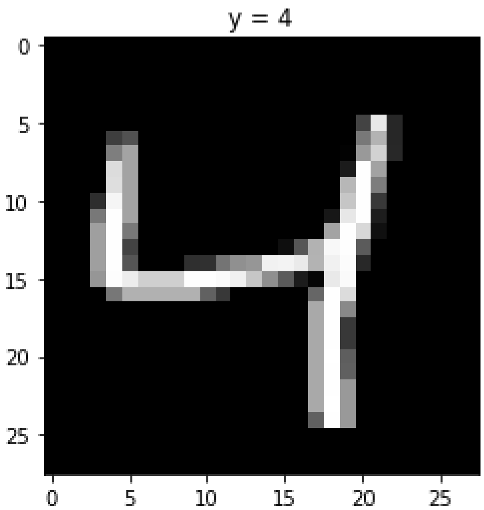
```
The image:  None
```



You see that it is a 1. Now, plot the third sample:

In [10]:

```
# Plot the image

show_data(train_dataset[2])
```



Build a Softmax Classifer

Build a Softmax classifier class:

In [11]:

```python
# Define softmax classifier class

class SoftMax(nn.Module):

    # Constructor
    def __init__(self, input_size, output_size):
        super(SoftMax, self).__init__()
        self.linear = nn.Linear(input_size, output_size)

    # Prediction
    def forward(self, x):
        z = self.linear(x)
        return z
```

The Softmax function requires vector inputs. Note that the vector shape is 28x28.

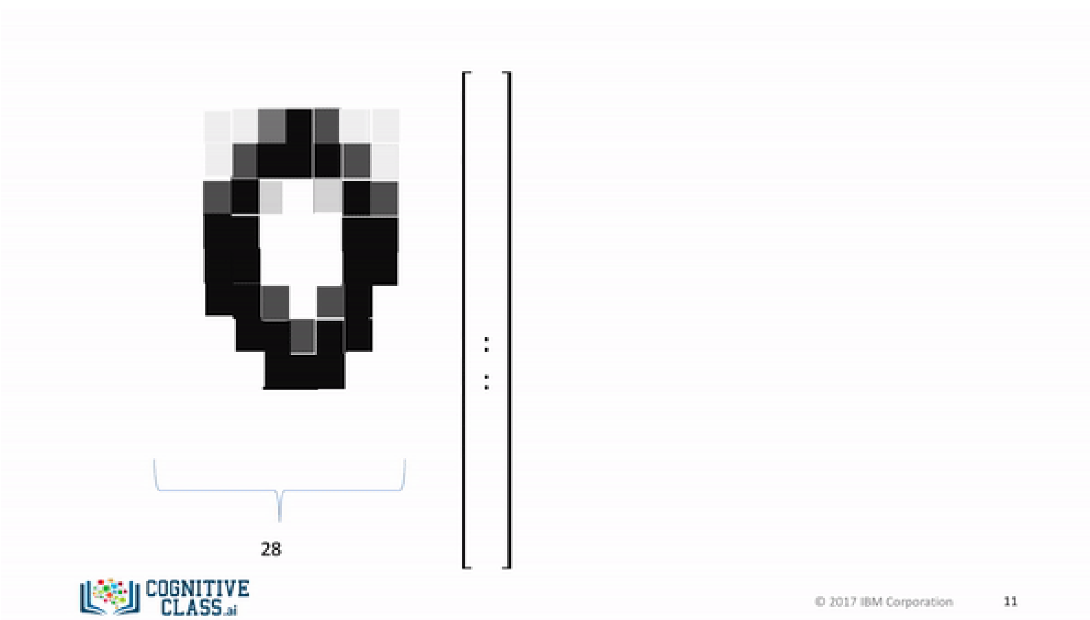
In [12]:

```python
# Print the shape of train dataset

train_dataset[0][0].shape
```

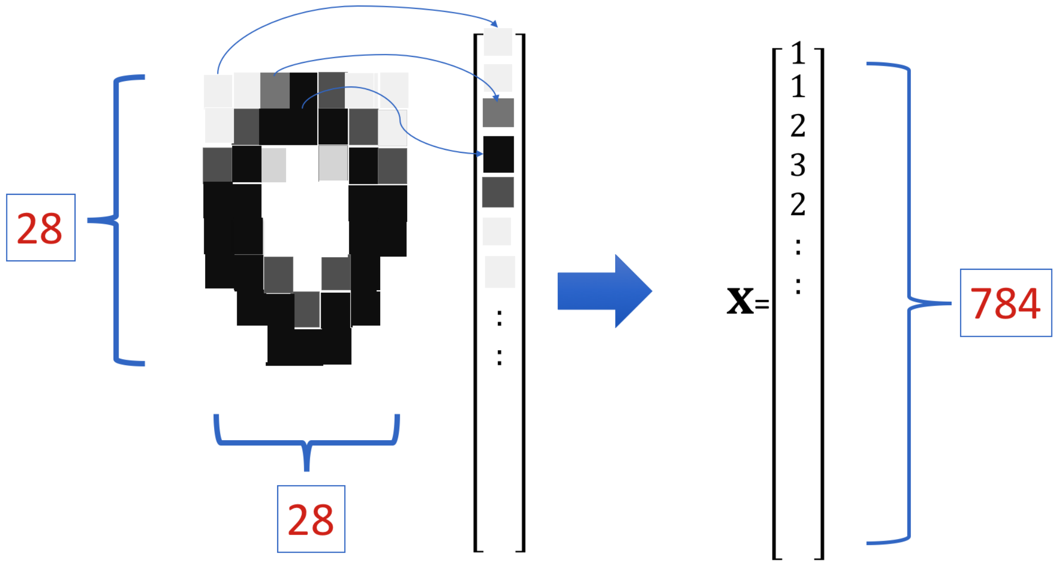
Out[12]:

```
torch.Size([1, 28, 28])
```

Flatten the tensor as shown in this image:



The size of the tensor is now 784.



Set the input size and output size:

In [13]:

```
# Set input size and output size

input_dim = 28 * 28
output_dim = 10
```

Define the Softmax Classifier, Criterion Function, Optimizer, and Train the Model

In [14]:

```
# Create the model

model = SoftMax(input_dim, output_dim)
print("Print the model:\n ", model)
```

```
Print the model:
  SoftMax(
  (linear): Linear(in_features=784, out_features=10, bias=True)
)
```

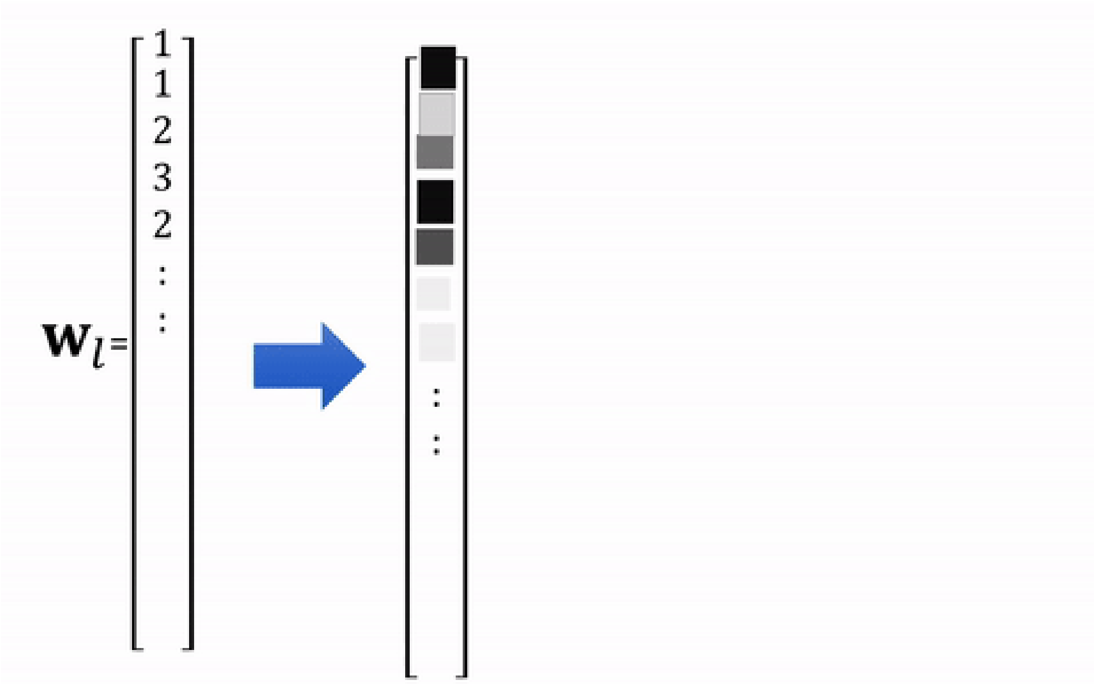
View the size of the model parameters:

In [15]:

```
# Print the parameters

print('W: ',list(model.parameters())[0].size())
print('b: ',list(model.parameters())[1].size())
```

```
W:  torch.Size([10, 784])
b:  torch.Size([10])
```

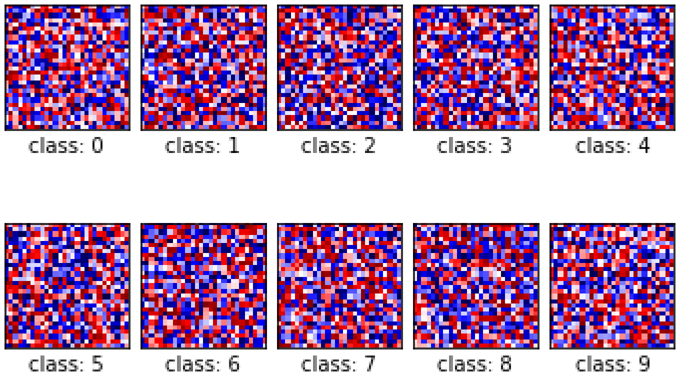
You can cover the model parameters for each class to a rectangular grid:



Plot the model parameters for each class as a square image:

In [16]:

```
# Plot the model parameters for each class

PlotParameters(model)
```



Define the learning rate, optimizer, criterion, data loader:

In [17]:

```
# Define the learning rate, optimizer, criterion and data loader

learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
criterion = nn.CrossEntropyLoss()
train_loader = torch.utils.data.DataLoader(dataset=train_dataset, batch_size=100)
validation_loader = torch.utils.data.DataLoader(dataset=validation_dataset, batch_s
ize=5000)
```

Train the model and determine validation accuracy **(should take a few minutes)**:

In [18]:

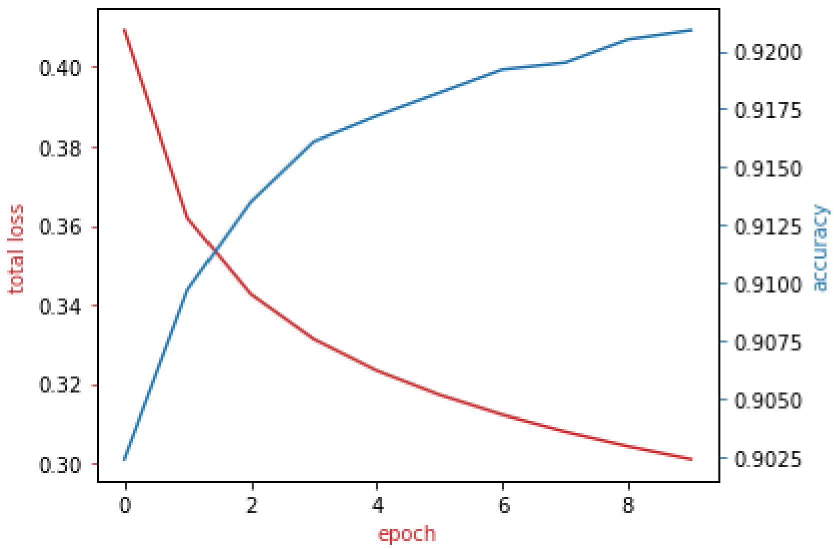
```
# Train the model

n_epochs = 10
loss_list = []
accuracy_list = []
N_test = len(validation_dataset)

def train_model(n_epochs):
    for epoch in range(n_epochs):
        for x, y in train_loader:
            optimizer.zero_grad()
            z = model(x.view(-1, 28 * 28))
            loss = criterion(z, y)
            loss.backward()
            optimizer.step()

        correct = 0
        # perform a prediction on the validationdata
        for x_test, y_test in validation_loader:
            z = model(x_test.view(-1, 28 * 28))
            _, yhat = torch.max(z.data, 1)
            correct += (yhat == y_test).sum().item()
        accuracy = correct / N_test
        loss_list.append(loss.data)
        accuracy_list.append(accuracy)

train_model(n_epochs)
```

Analyze Results

Plot the loss and accuracy on the validation data:

In [19]:

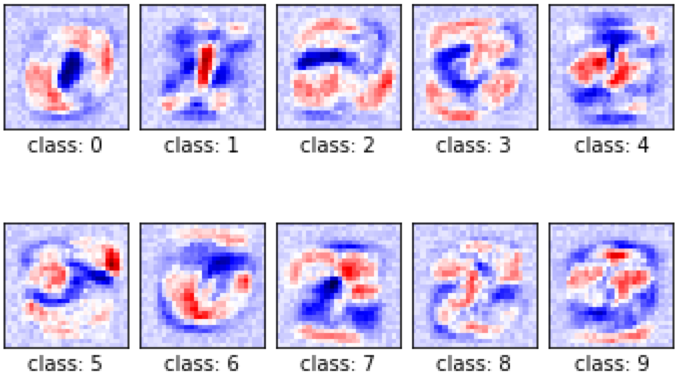
```
# Plot the loss and accuracy

fig, ax1 = plt.subplots()
color = 'tab:red'
ax1.plot(loss_list,color=color)
ax1.set_xlabel('epoch',color=color)
ax1.set_ylabel('total loss',color=color)
ax1.tick_params(axis='y', color=color)

ax2 = ax1.twinx()
color = 'tab:blue'
ax2.set_ylabel('accuracy', color=color)
ax2.plot( accuracy_list, color=color)
ax2.tick_params(axis='y', color=color)
fig.tight_layout()
```



View the results of the parameters for each class after the training. You can see that they look like the corresponding numbers.

In [20]:

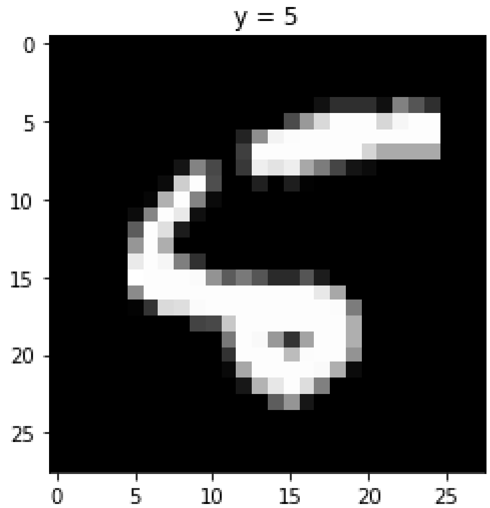
```
# Plot the parameters

PlotParameters(model)
```



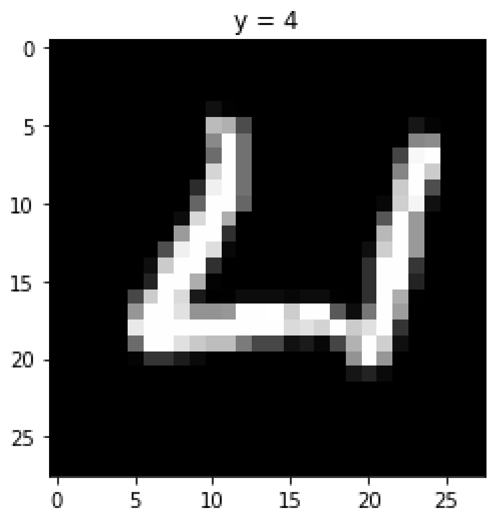
We Plot the first five misclassified samples and the probability of that class.

In [21]:

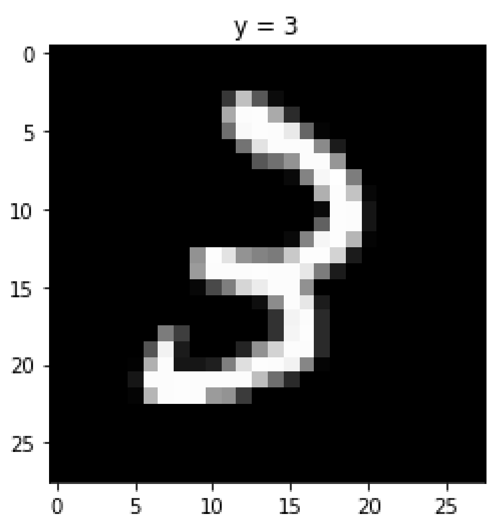
```
# Plot the misclassified samples
Softmax_fn=nn.Softmax(dim=-1)
count = 0
for x, y in validation_dataset:
    z = model(x.reshape(-1, 28 * 28))
    _, yhat = torch.max(z, 1)
    if yhat != y:
        show_data((x, y))
        plt.show()
        print("yhat:", yhat)
        print("probability of class ", torch.max(Softmax_fn(z)).item())
        count += 1
    if count >= 5:
        break
```



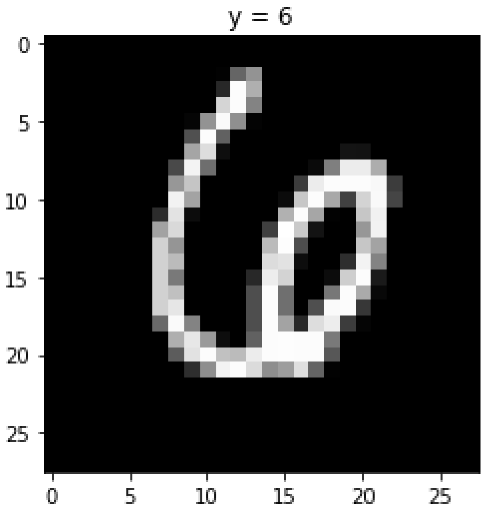

```
yhat: tensor([6])
probability of class  0.989754855632782
```



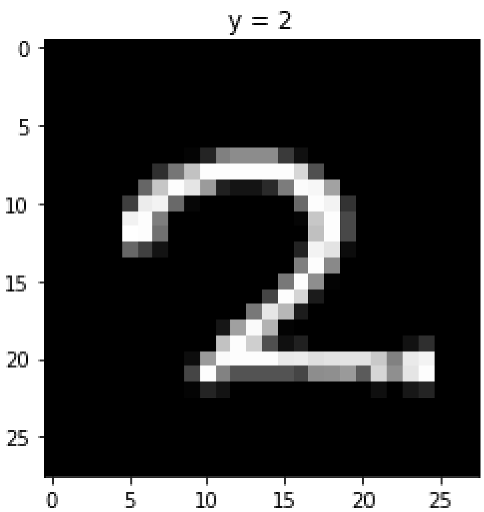

```
yhat: tensor([6])
probability of class  0.4310683012008667
```




```
yhat: tensor([2])
probability of class  0.6805763840675354
```




```
yhat: tensor([7])
probability of class  0.33883747458457947
```

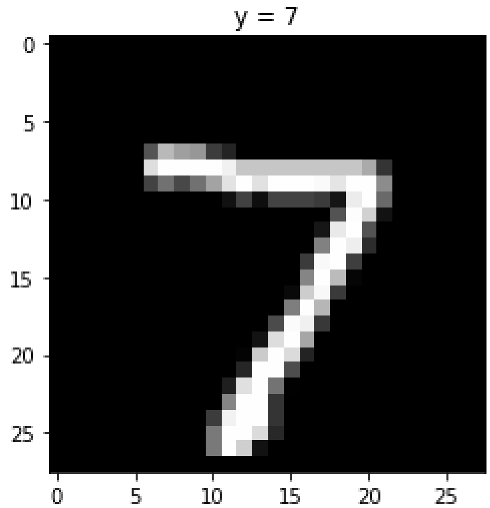



```
yhat: tensor([7])
probability of class  0.5170221328735352
```

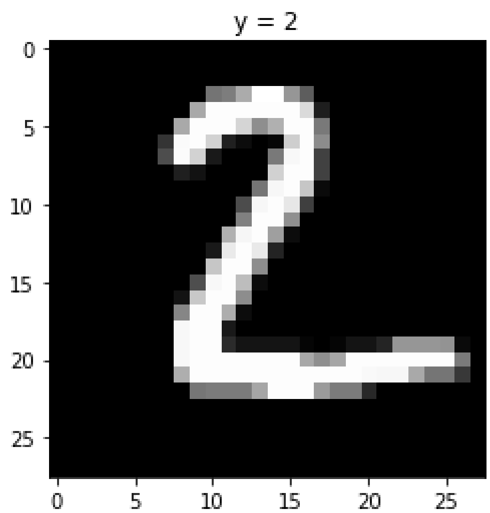
We Plot the first five correctly classified samples and the probability of that class, we see the probability is much larger.

In [22]:

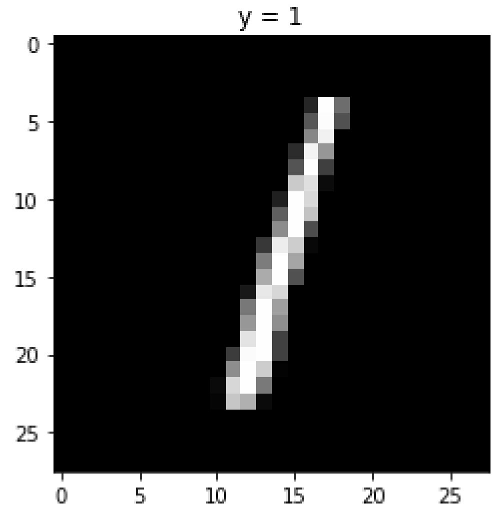
```
# Plot the classified samples
Softmax_fn=nn.Softmax(dim=-1)
count = 0
for x, y in validation_dataset:
    z = model(x.reshape(-1, 28 * 28))
    _, yhat = torch.max(z, 1)
    if yhat == y:
        show_data((x, y))
        plt.show()
        print("yhat:", yhat)
        print("probability of class ", torch.max(Softmax_fn(z)).item())
        count += 1
    if count >= 5:
        break
```



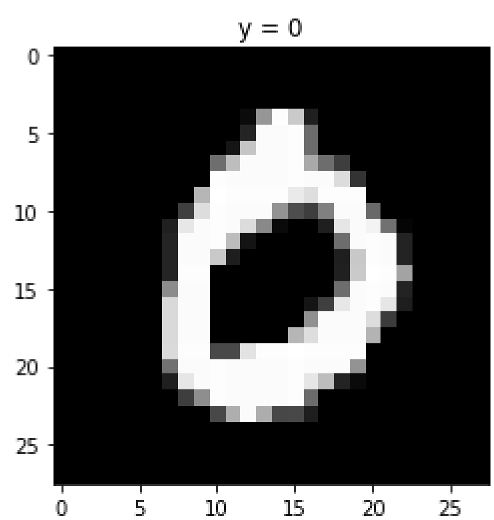

```
yhat: tensor([7])
probability of class  0.9967310428619385
```



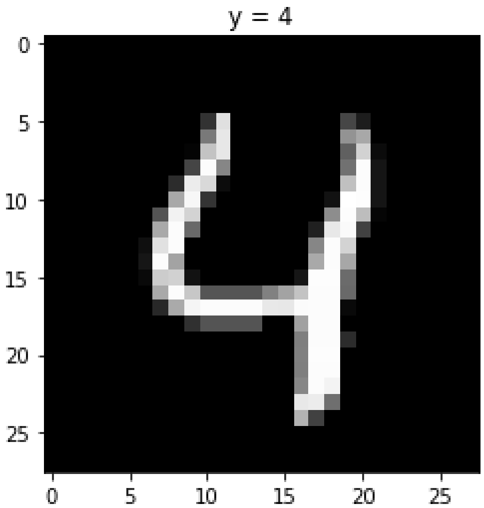

```
yhat: tensor([2])
probability of class  0.9467912912368774
```



yhat: tensor([1])
probability of class 0.9744181632995605



yhat: tensor([0])
probability of class 0.9995574355125427




```
yhat: tensor([4])
probability of class  0.9453811645507812
```


About the Authors:

Joseph Santarcangelo (https://www.linkedin.com/in/joseph-s-50398b136/) has a PhD in Electrical Engineering, his research focused on using machine learning, signal processing, and computer vision to determine how videos impact human cognition. Joseph has been working for IBM since he completed his PhD.

Other contributors: Michelle Carey (https://www.linkedin.com/in/michelleccarey/), Mavis Zhou (www.linkedin.com/in/jiahui-mavis-zhou-a4537814a)