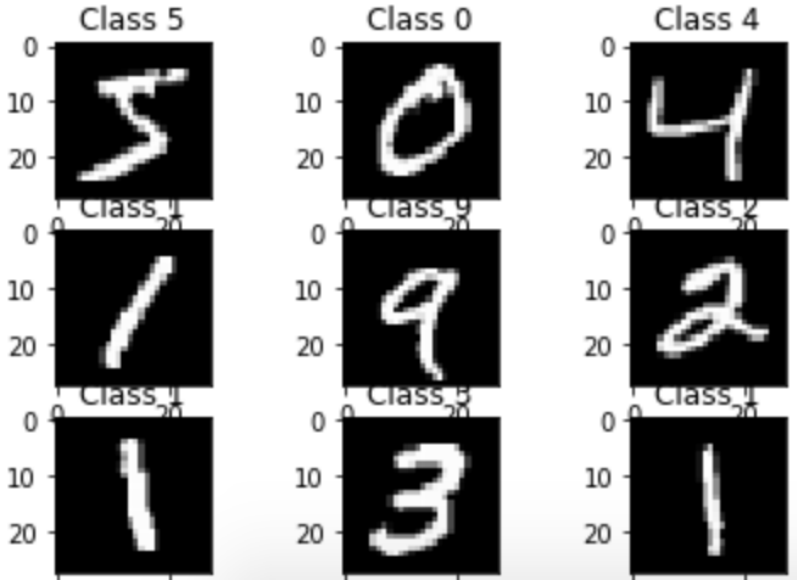
Softmax exercise

In this exercise, you are asked to implement six functionalities involved in the binary classification of MNIST dataset (1 or not 1) and answer the inline question: why do we expect our loss to be close to $-\log(0.1)$?

1. Exploration and visualization of MNIST dataset:

Print the shape of the training data and testing data
Plot the previous 9 training data and title their class
Count the number of data for each class in training data (np.unique)
You would get something similar with the following graph:

```
X_train original shape (60000, 28, 28)
y_train original shape (60000,)
X_test original shape (10000, 28, 28)
y_test original shape (10000,)
class   count
0       5923
1       6742
2       5958
3       6131
4       5842
5       5421
6       5918
7       6265
8       5851
9       5949
```



2. A navie softmax loss:

Compute the softmax loss and its gradient using explicit loops. Don’t forget regularization.

The idea of softmax is to define a new type of output layer for our neural networks. It begins in the same way as with a sigmoid layer, by forming the weight inputs $z^L_j = \sum_k w^L_{jk} a^{L-1}_k + b^L_j$. However, we don’t apply the sigmoid function to get the output. Instead, in a softmax layer we apply the so-called *softmax function* to the z^L_j . According to this function, the activation a^L_l of the j th output neuron is

$$a_j^L = \frac{e^{z_j^L}}{\sum_k e^{z_k^L}}$$

Where in the denominator we sum over all the output neurons.
Your code should be something like this:

```
N = X.shape[0]
D = X.shape[1]
C = W.shape[1]
data_loss = 0.0

prob = np.zeros((N, C))
for i in range(N):
  score = np.zeros((C))
  for j in range(C):
    score[j] = np.sum(X[i] * W[:, j])
pass

dscore = np.array(prob)
for i in range(N):
  dscore[i, y[i]] -= 1
pass
```

3. A vectorized softmax loss

Compute the softmax loss and its gradient using no explicit loops.
Your code should be something like this:

```
N = X.shape[0]
D = X.shape[1]
C = W.shape[1]
data_loss = 0.0

scores = np.dot(X, W)
exp_scores = np.exp(scores)
```

```
  probs = exp_scores / np.sum(exp_scores, axis = 1,
keepdims = True) # keepdims: broadcast
  correct_logprobs = - np.log(probs[range(N), y]) #
range(N)
  data_loss += np.sum(correct_logprobs) / N

  reg_loss = 0.5 * reg * np.sum(W * W)
  loss = data_loss + reg_loss

  dscores = probs
  dscores[range(N), y] -= 1
  dscores /= N
  dW = np.dot(X.T, dscores)
  dW[:, :-1] += reg * W[:, :-1]
```

4. The training function of softmax classifier

Sample batch_size elements from the training data and their corresponding labels to use in this round of gradient descent.
Update the weights using the gradient and the learning rate.
Your code should be something like this:

```
      idx = np.random.choice(range(num_train),
batch_size, replace = False)
      pass

       self.W += - learning_rate * grad
```

5. The predicting function of softmax classifier

Predict the labels given a number of flattened images.
Your code should be something like this:

```
    scores = np.dot(X, self.W)
    pass # np.argmax
```

6. Hyperparameters tuning

Use validation set to set the learning rate and regularization strength. Save the best trained softmax classifier.

Your code should be something like this:

```
for lr in learning_rates:
    for rs in regularization_strengths:
        clf = Softmax()
        pass
```