assignment_03

September 24, 2021

1 Logistic regression for binary classification

1.1 import libraries

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import os
from tqdm import tqdm
```

1.2 load data

[ ]:
```
directory_data  = '/Users/lyuwan-u/Desktop/2021-2/
 ↪assignment-machine-learning-project/assignment03'
filename_data   = 'assignment_03_data.npz'
data            = np.load(os.path.join(directory_data, filename_data))

x_train = data['x_train']
y_train = data['y_train']

x_test  = data['x_test']
y_test  = data['y_test']

print('************************************************')
print('size of x_train :', x_train.shape)
print('size of y_train :', y_train.shape)
print('************************************************')
print('size of x_test :', x_test.shape)
print('size of y_test :', y_test.shape)
print('************************************************')
print('number of training image :', x_train.shape[0])
print('height of training image :', x_train.shape[1])
print('width of training image :', x_train.shape[2])
print('************************************************')
print('number of testing image :', x_test.shape[0])
print('height of testing image :', x_test.shape[1])
print('width of testing image :', x_test.shape[2])
print('************************************************')
```

```
************************************************
size of x_train : (10000, 28, 28)
size of y_train : (10000,)
************************************************
size of x_test : (1800, 28, 28)
size of y_test : (1800,)
************************************************
number of training image : 10000
height of training image : 28
width of training image : 28
************************************************
number of testing image : 1800
height of testing image : 28
width of testing image : 28
************************************************
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.3 vectorize image data

[ ]:
```
vector_x_train  = x_train.reshape(x_train.shape[0], x_train.shape[1] * x_train.
 ↪shape[2])
vector_x_test   = x_test.reshape(x_test.shape[0], x_test.shape[1] * x_test.
 ↪shape[2])

print(vector_x_train.shape)
print(vector_x_test.shape)
```

```
(10000, 784)
(1800, 784)
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.4 index for each class

[ ]:
```
index_train_0 = [i for i in range(len(y_train)) if y_train[i]==0]
index_train_1 = [i for i in range(len(y_train)) if y_train[i]==1]

index_test_0 =  [i for i in range(len(y_test)) if y_test[i]==0]
index_test_1 =  [i for i in range(len(y_test)) if y_test[i]==1]
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

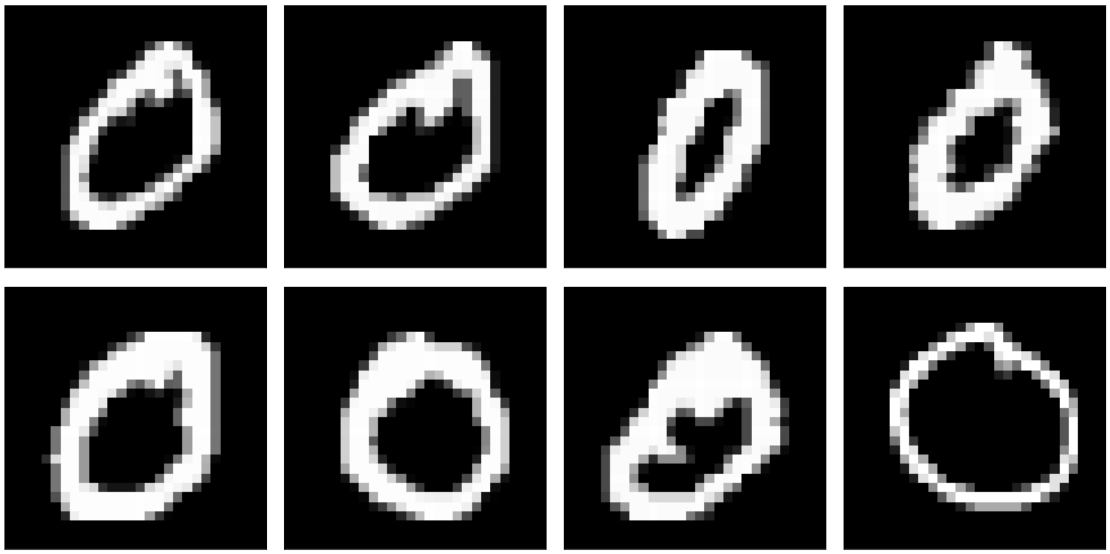
1.5 plot data

[ ]:
```
def plot_data_grid(data, index_data, nRow, nCol):

    fig, axes = plt.subplots(nRow, nCol, constrained_layout=True, figsize=(nCol␣
 ↪* 3, nRow * 3))
```

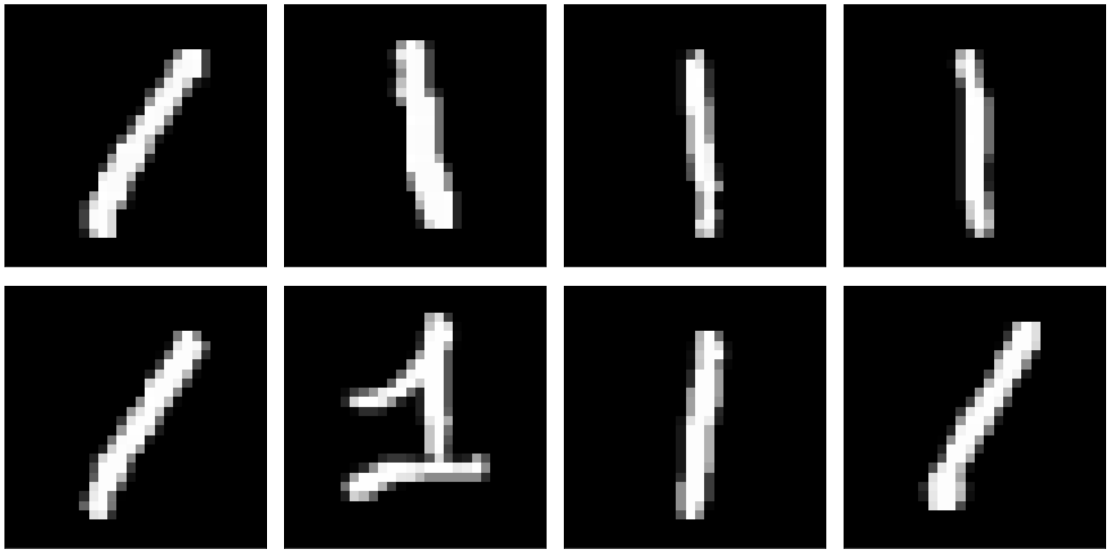
```
    for i in range(nRow):
        for j in range(nCol):

            k       = i * nCol + j
            index   = index_data[k]

            axes[i, j].imshow(data[index], cmap='gray', vmin=0, vmax=1)
            axes[i, j].xaxis.set_visible(False)
            axes[i, j].yaxis.set_visible(False)

    plt.show()
```

[ ]:
```
nRow    = 2
nCol    = 4
nPlot   = nRow * nCol
```

[ ]:
```
index_data_0 = np.array(range(nPlot))
plot_data_grid(x_train, index_data_0, nRow, nCol)
```



[ ]:
```
index_data_1 = index_data_0 + 5000
plot_data_grid(x_train, index_data_1, nRow, nCol)
```



```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.6 linear layer

[ ]:
```python
def layer_linear(input, weight):

    output = np.dot(input,weight.T)

    return output
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.7 activation function : Sigmoid

[ ]:
```python
def activation_sigmoid(input):

    output = 1 / (1+np.exp(-input))
    return output
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.8 compute prediction by the forward propagation of the neural network

[ ]:
```python
def compute_prediction(input, weight):

    output      = layer_linear(input,weight)
    prediction  = activation_sigmoid(output)

    return prediction
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.9 compute loss function

[ ]:
```
def compute_loss(prediction, label):

    loss            = np.sum(-1 * (label * np.log(prediction) + (1-label) * np.
 ↪log(1-prediction)))
    loss_average    = loss / len(label)

    return loss_average
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.10 compute gradient

[ ]:
```
def compute_gradient(input, prediction, label):

    residual    =  (prediction-label).dot(input)
    gradient    =  residual / len(label)

    return gradient
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.11 compute accuracy

[ ]:
```
def compute_accuracy(prediction, label):

    threshold   = 0.5
    temp        = [0 if i < threshold else 1 for i in prediction]
    bCorrect    = [True if temp[i]==label[i] else False for i in␣
 ↪range(len(label))]
    accuracy    = np.sum(bCorrect) / len(label)

    return accuracy
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.12 initialize weight

[ ]:
```
length_weight   = vector_x_train.shape[1]
weight          = np.ones(length_weight)
weight          = weight * 0.001

print('number of weights: ', length_weight)
```

```
number of weights:  784
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.13 hyper-parameters

[ ]:
```
number_iteration    = 1000
learning_rate       = 0.1
```

1.14 variables for optimization information

[ ]:
```
loss_train_iteration        = np.zeros(number_iteration)
loss_test_iteration         = np.zeros(number_iteration)

accuracy_train_iteration    = np.zeros(number_iteration)
accuracy_test_iteration     = np.zeros(number_iteration)

pred_0_train_mean_iteration = np.zeros(number_iteration)
pred_0_train_std_iteration  = np.zeros(number_iteration)
pred_1_train_mean_iteration = np.zeros(number_iteration)
pred_1_train_std_iteration  = np.zeros(number_iteration)

pred_0_test_mean_iteration = np.zeros(number_iteration)
pred_0_test_std_iteration  = np.zeros(number_iteration)
pred_1_test_mean_iteration = np.zeros(number_iteration)
pred_1_test_std_iteration  = np.zeros(number_iteration)
```

1.15 gradient descent iterations

[ ]:
```
for i in tqdm(range(number_iteration)):

    prediction_train    = compute_prediction(vector_x_train,weight)
    prediction_test     = compute_prediction(vector_x_test,weight)

    gradient_train      =␣
 ↪compute_gradient(vector_x_train,prediction_train,y_train)
    weight              = weight - learning_rate * gradient_train

    prediction_train    = compute_prediction(vector_x_train,weight)
    prediction_test     = compute_prediction(vector_x_test,weight)

    loss_train          = compute_loss(prediction_train,y_train)
    loss_test           = compute_loss(prediction_test,y_test)

    accuracy_train      = compute_accuracy(prediction_train,y_train)
    accuracy_test       = compute_accuracy(prediction_test,y_test)

    pred_train_0    = prediction_train[index_train_0]
```

```
    pred_train_1    = prediction_train[index_train_1]

    pred_test_0     = prediction_test[index_test_0]
    pred_test_1     = prediction_test[index_test_1]

    pred_0_train_mean_iteration[i]  = np.mean(pred_train_0)
    pred_0_train_std_iteration[i]   = np.std(pred_train_0)
    pred_1_train_mean_iteration[i]  = np.mean(pred_train_1)
    pred_1_train_std_iteration[i]   = np.std(pred_train_1)

    pred_0_test_mean_iteration[i]   = np.mean(pred_train_0)
    pred_0_test_std_iteration[i]    = np.std(pred_train_0)
    pred_1_test_mean_iteration[i]   = np.mean(pred_train_1)
    pred_1_test_std_iteration[i]    = np.std(pred_train_1)

    loss_train_iteration[i]         = loss_train
    loss_test_iteration[i]          = loss_test

    accuracy_train_iteration[i]     = accuracy_train
    accuracy_test_iteration[i]      = accuracy_test
```

```
100%|      | 1000/1000 [00:59<00:00, 16.94it/s]
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.16 plot curve

[ ]:
```
def plot_curve(data, x_label, y_label, title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    plt.plot(range(len(data)), data, '-', color='red')

    plt.xlabel(x_label)
    plt.ylabel(y_label)

    plt.tight_layout()
    plt.show()
```

[ ]:
```
def plot_curve2(data1, label_data1, data2, label_data2, x_label, y_label,␣
 ↪title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    plt.plot(range(len(data1)), data1, '-', color = 'blue', label = label_data1)
```

```
    plt.plot(range(len(data2)), data2, '-', color = 'red', label = label_data2)

    plt.xlabel(x_label)
    plt.ylabel(y_label)

    plt.tight_layout()
    plt.show()
```

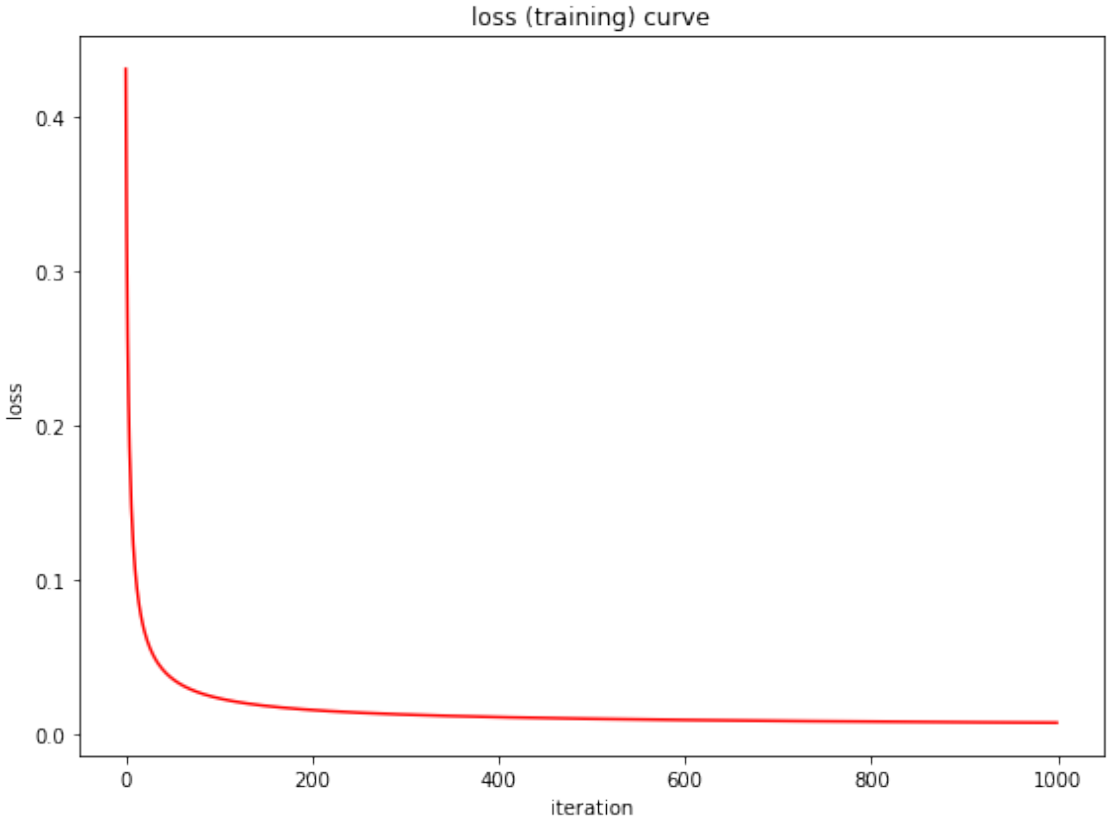
[ ]:
```
def plot_curve_error(data_mean, data_std, x_label, y_label, title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    alpha = 0.3

    plt.plot(range(len(data_mean)), data_mean, '-', color = 'red')
    plt.fill_between(range(len(data_mean)), data_mean - data_std, data_mean +␣
 ↪data_std, facecolor = 'blue', alpha = alpha)

    plt.xlabel(x_label)
    plt.ylabel(y_label)

    plt.tight_layout()
    plt.show()
```

[ ]:
```
def plot_curve_error2(data1_mean, data1_std, data1_label, data2_mean,␣
 ↪data2_std, data2_label, x_label, y_label, title):

    plt.figure(figsize=(8, 6))
    plt.title(title)

    alpha = 0.3

    plt.plot(range(len(data1_mean)), data1_mean, '-', color = 'blue', label =␣
 ↪data1_label)
    plt.fill_between(range(len(data1_mean)), data1_mean - data1_std, data1_mean␣
 ↪+ data1_std, facecolor = 'blue', alpha = alpha)

    plt.plot(range(len(data2_mean)), data2_mean, '-', color = 'red', label =␣
 ↪data2_label)
    plt.fill_between(range(len(data2_mean)), data2_mean - data2_std, data2_mean␣
 ↪+ data2_std, facecolor = 'red', alpha = alpha)

    plt.xlabel(x_label)
    plt.ylabel(y_label)
```

```
    plt.tight_layout()
    plt.show()
```

1.17 loss (training) curve

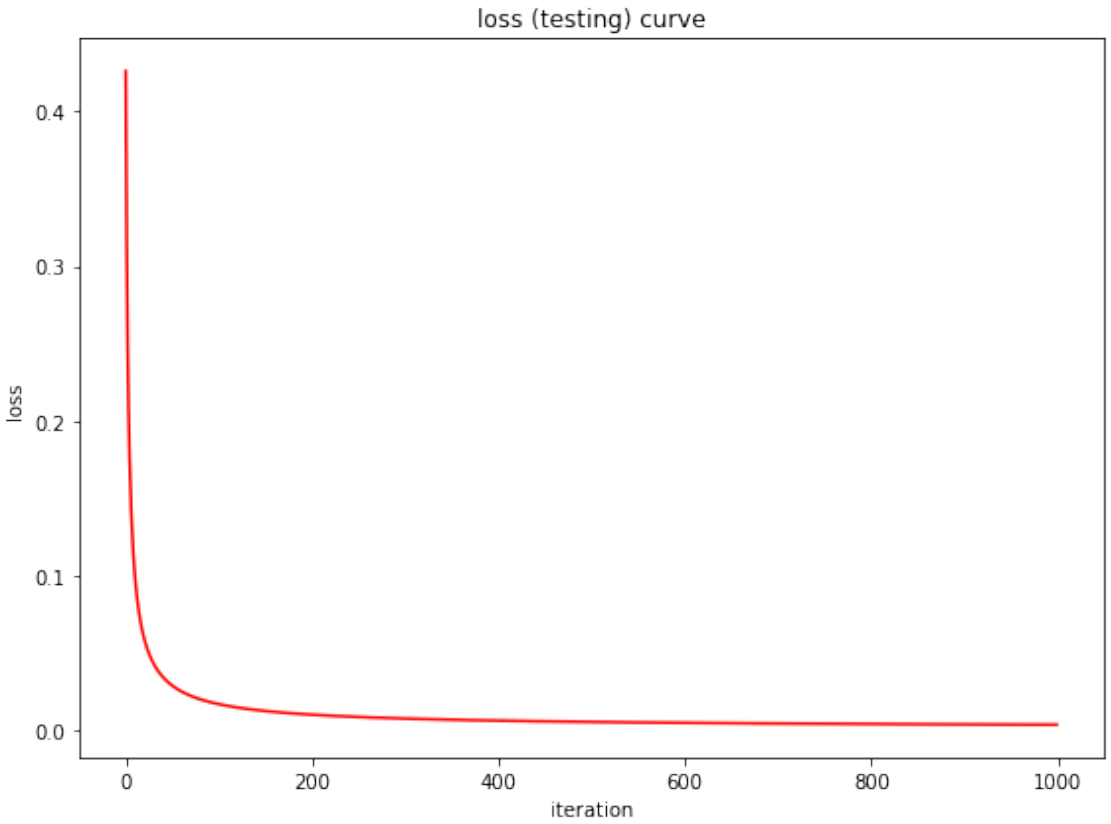
[ ]:
```
plot_curve(loss_train_iteration, 'iteration', 'loss', 'loss (training) curve')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.18 loss (testing) curve

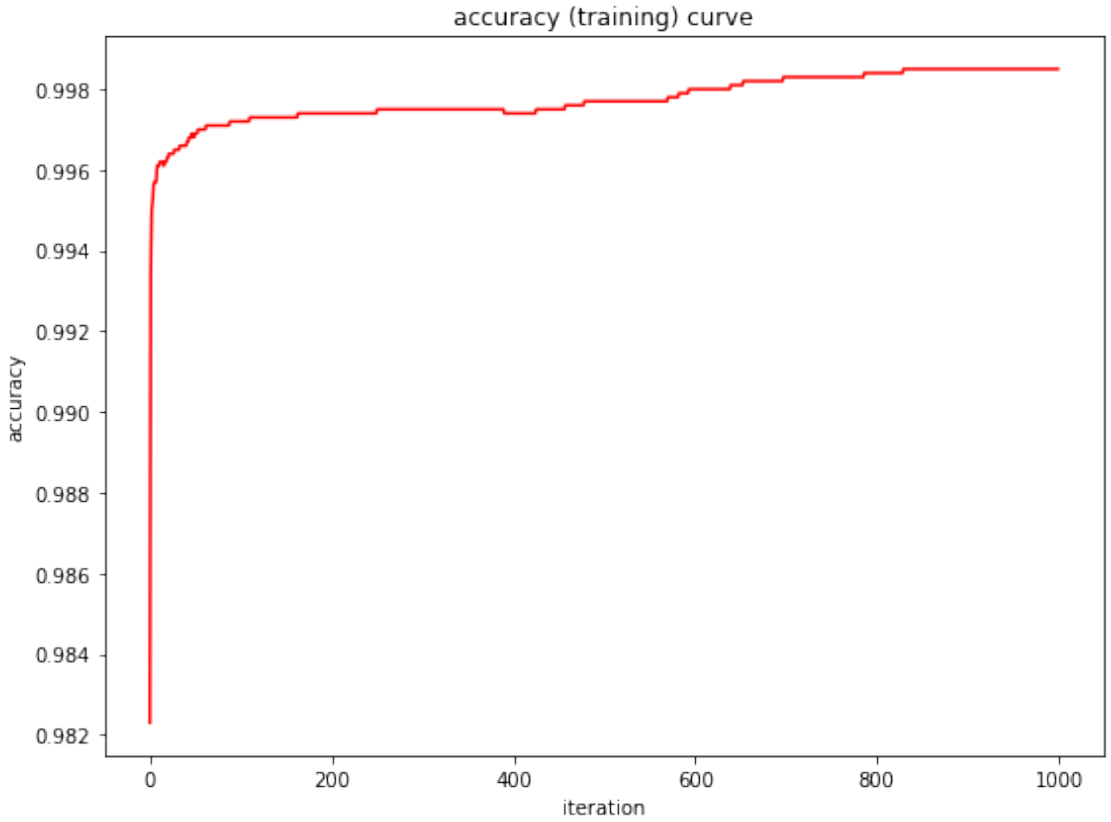
[ ]:
```
plot_curve(loss_test_iteration, 'iteration', 'loss', 'loss (testing) curve')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.19 accuracy (training) curve

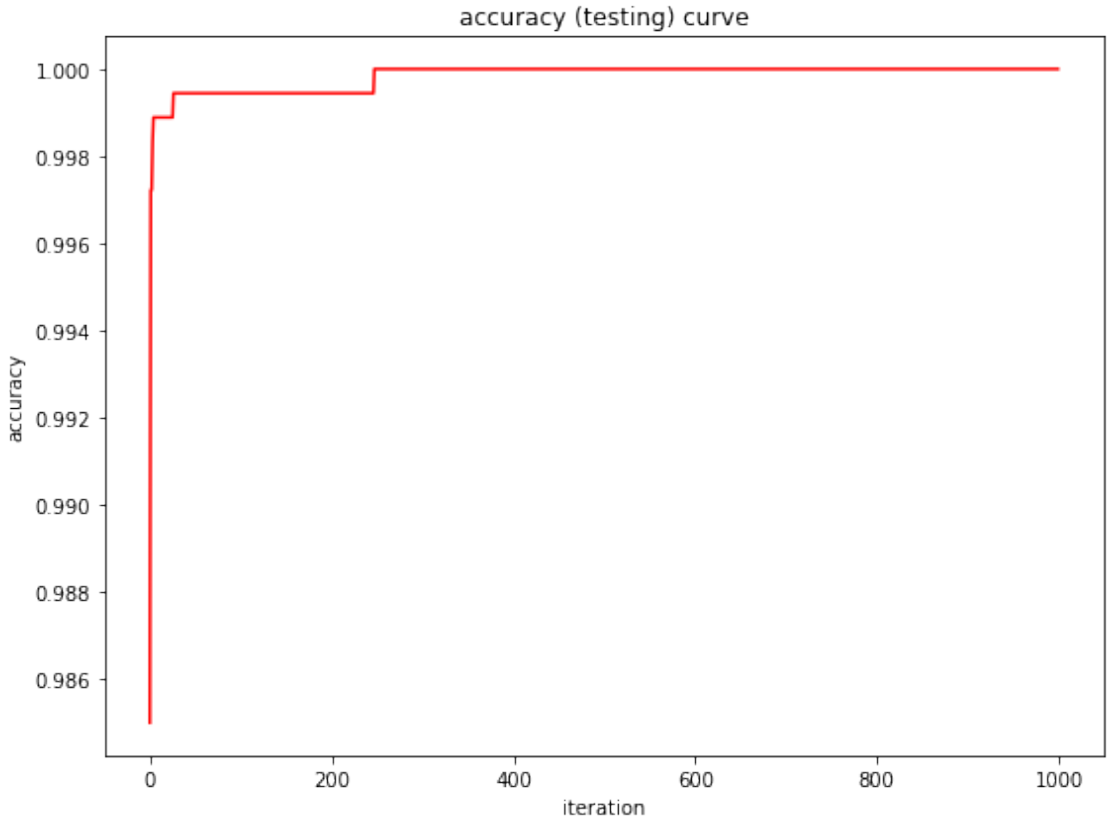
[ ]:
```
plot_curve(accuracy_train_iteration, 'iteration', 'accuracy', 'accuracy␣
↪(training) curve')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.20 accuracy (testing) curve

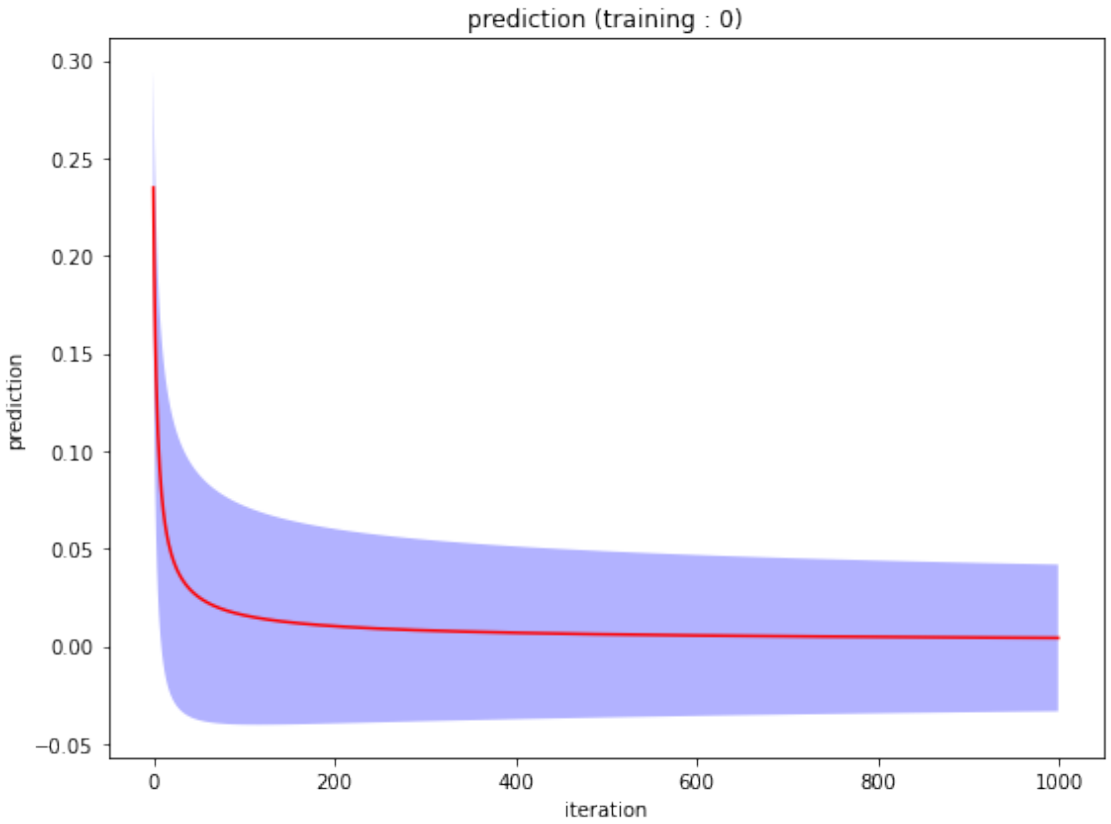
[ ]:
```
plot_curve(accuracy_test_iteration, 'iteration', 'accuracy', 'accuracy␣
↪(testing) curve')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

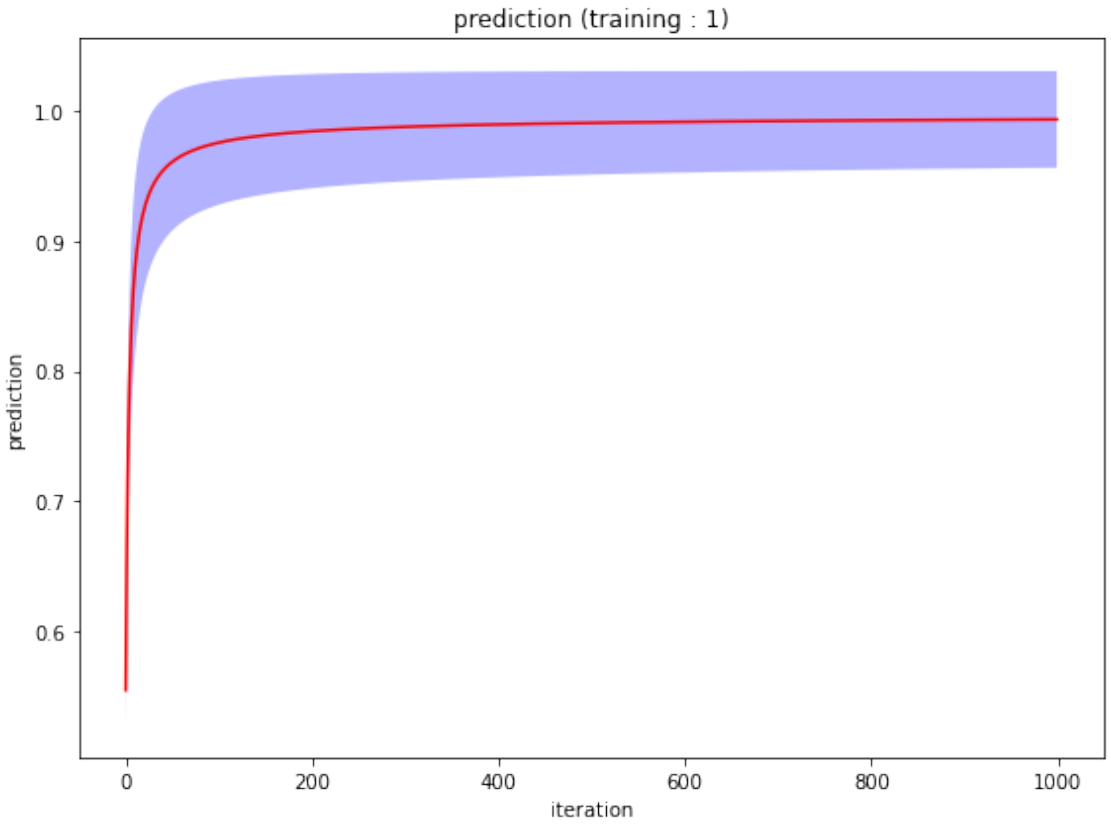
1.21 plot prediction values

[ ]:
```
plot_curve_error(pred_0_train_mean_iteration, pred_0_train_std_iteration,␣
↪'iteration', 'prediction', 'prediction (training : 0)')
```



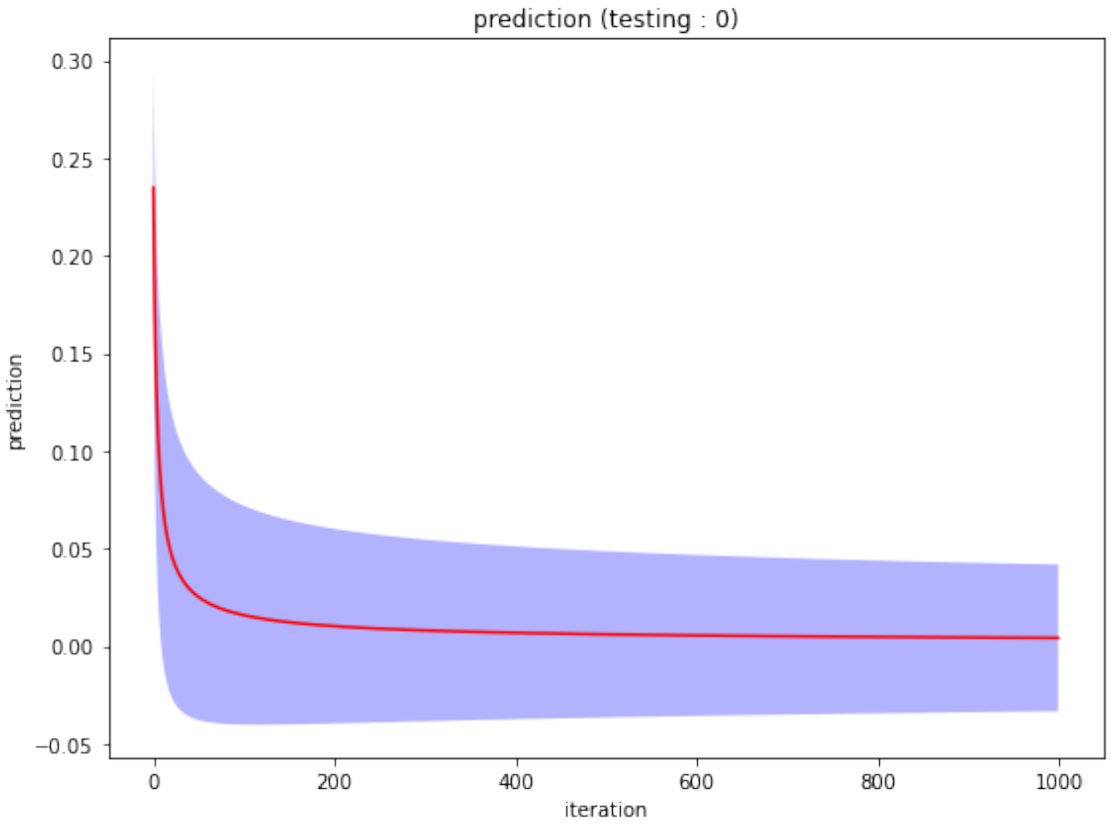

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

[ ]:
```
plot_curve_error(pred_1_train_mean_iteration, pred_1_train_std_iteration,␣
↪'iteration', 'prediction', 'prediction (training : 1)')
```



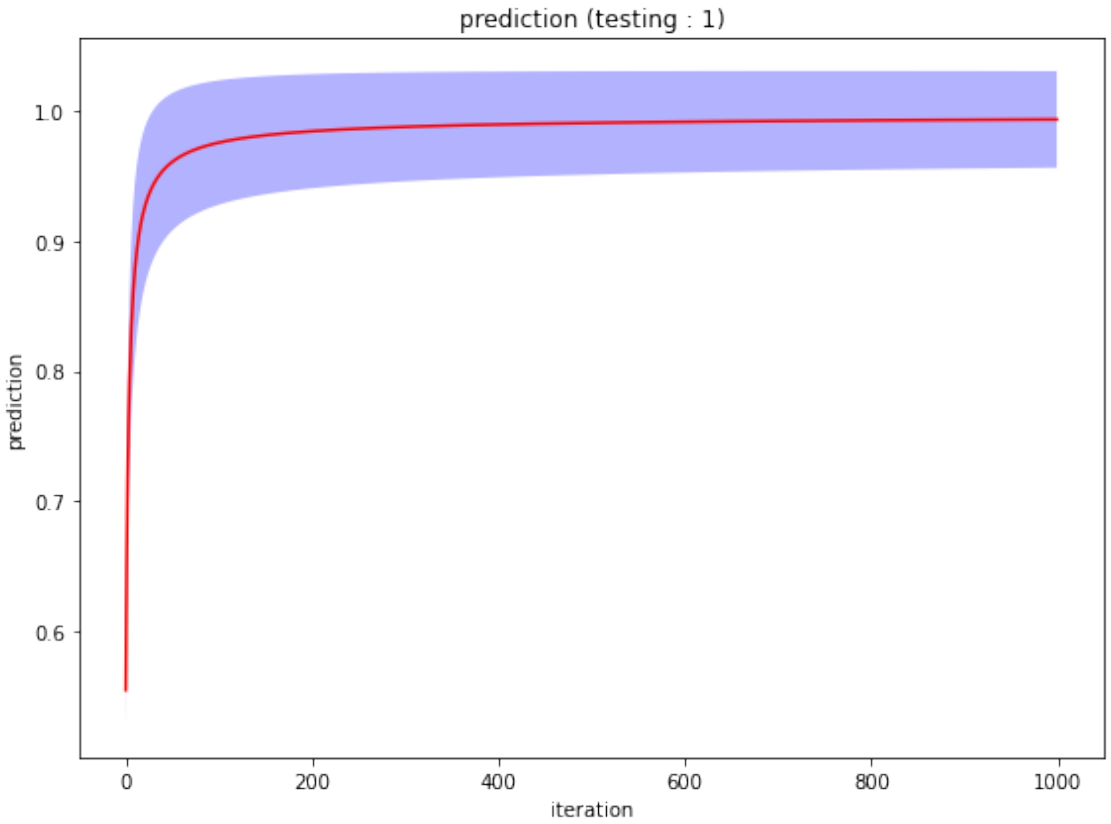

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

[ ]:
```
plot_curve_error(pred_0_test_mean_iteration, pred_0_test_std_iteration,␣
↪'iteration', 'prediction', 'prediction (testing : 0)')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

[ ]:
```
plot_curve_error(pred_1_test_mean_iteration, pred_1_test_std_iteration,␣
↪'iteration', 'prediction', 'prediction (testing : 1)')
```




```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.22 print values

[ ]:
```
def print_curve(data, index):

    for i in range(len(index)):

        idx = index[i]
        val = data[idx]

        print('index = %4d, value = %12.10f' % (idx, val))
```

1.23 given iterations at which the values are presented

[ ]:
```
index = np.array([0, 100, 200, 300, 400, 500, 600, 700, 800, 900])
```

1.24 training loss

[ ]:
```
print_curve(loss_train_iteration, index)
```

```
index =    0, value = 0.4307224938
index =  100, value = 0.0228587163
index =  200, value = 0.0152848437
index =  300, value = 0.0123252409
index =  400, value = 0.0106733249
index =  500, value = 0.0095914393
index =  600, value = 0.0088148843
index =  700, value = 0.0082233149
index =  800, value = 0.0077534390
index =  900, value = 0.0073684871
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.25 testing loss

[ ]:
```
print_curve(loss_test_iteration, index)
```

```
index =    0, value = 0.4263057540
index =  100, value = 0.0167458571
index =  200, value = 0.0099479648
index =  300, value = 0.0074345387
index =  400, value = 0.0060961283
index =  500, value = 0.0052563282
index =  600, value = 0.0046772451
index =  700, value = 0.0042526445
index =  800, value = 0.0039275776
index =  900, value = 0.0036706180
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.26 training accuracy

[ ]:
```
print_curve(accuracy_train_iteration, index)
```

```
index =    0, value = 0.9823000000
index =  100, value = 0.9972000000
index =  200, value = 0.9974000000
index =  300, value = 0.9975000000
index =  400, value = 0.9974000000
index =  500, value = 0.9977000000
index =  600, value = 0.9980000000
index =  700, value = 0.9983000000
index =  800, value = 0.9984000000
index =  900, value = 0.9985000000
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.27 testing accuracy

[ ]:
```
print_curve(accuracy_test_iteration, index)
```

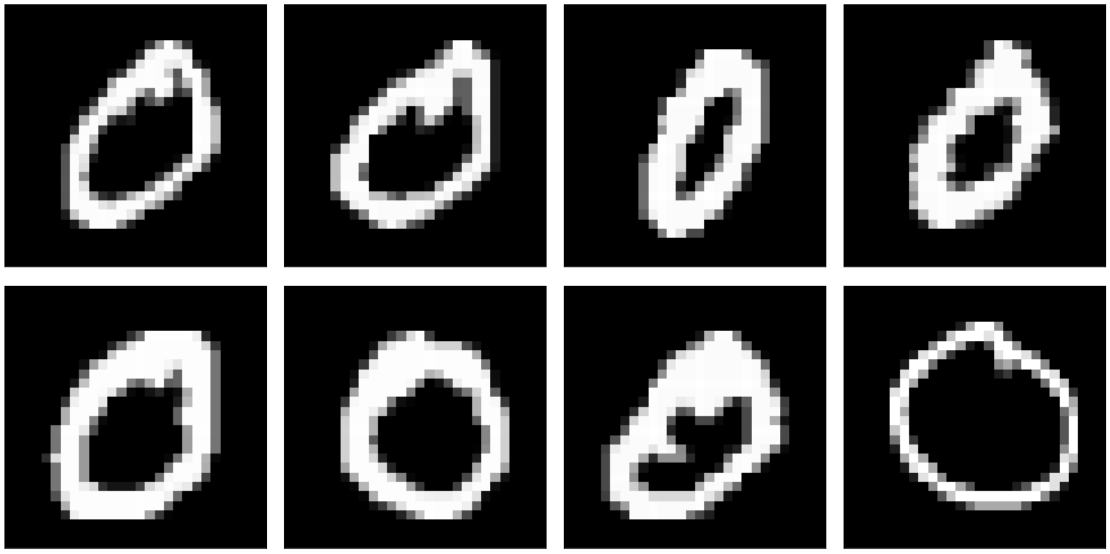
```
index =    0, value = 0.9850000000
index =  100, value = 0.9994444444
index =  200, value = 0.9994444444
index =  300, value = 1.0000000000
index =  400, value = 1.0000000000
index =  500, value = 1.0000000000
index =  600, value = 1.0000000000
index =  700, value = 1.0000000000
index =  800, value = 1.0000000000
index =  900, value = 1.0000000000
```

```
$ git commit -m "YOUR MESSAGE" assignment_03.ipynb
```

1.28 functions for presenting the results

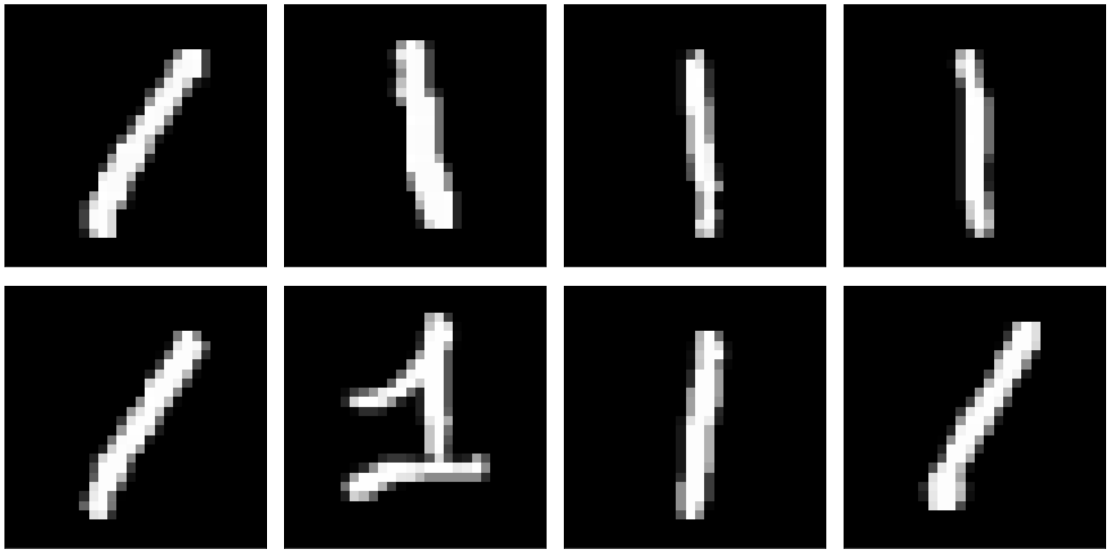
[ ]:
```
def function_results_01(data,nRow,nCol):
    nPlot      = nRow * nCol
    index_data = np.array(range(nPlot))
    fig, axes  = plt.
 ↪subplots(nRow,nCol,constrained_layout=True,figsize=(nCol*3,nRow*3))

    for i in range(nRow):
        for j in range(nCol):

            k     = i * nCol + j
            index = index_data[k]

            axes[i,j].imshow(data[index],cmap='gray',vmin=0,vmax=1)
            axes[i,j].xaxis.set_visible(False)
            axes[i,j].yaxis.set_visible(False)
    plt.show()
```

[ ]:
```
function_results_01(x_train,2,4)
```



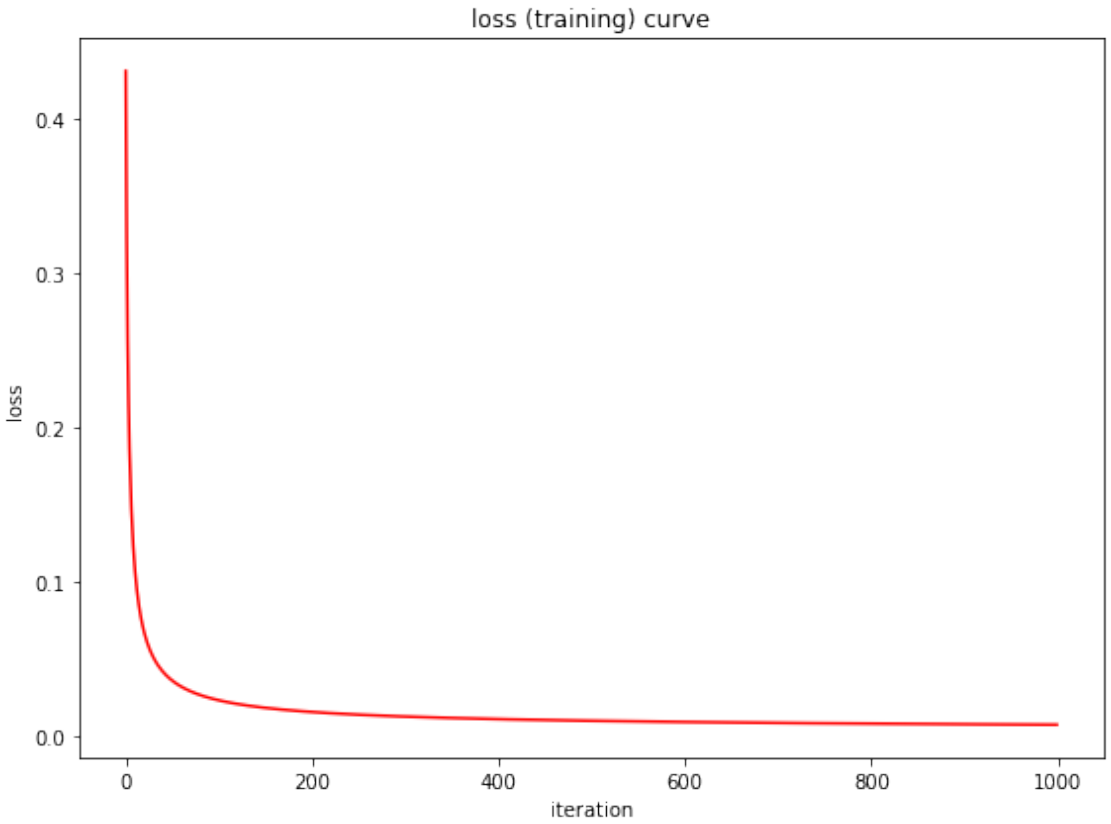
[ ]:
```
def function_results_02(data,nRow,nCol):
    nPlot      = nRow * nCol
    index_data = np.array(range(nPlot))
    index_data = index_data + 5000
    fig, axes  = plt.
↪subplots(nRow,nCol,constrained_layout=True,figsize=(nCol*3,nRow*3))

    for i in range(nRow):
        for j in range(nCol):

            k     = i * nCol + j
            index = index_data[k]

            axes[i,j].imshow(data[index],cmap='gray',vmin=0,vmax=1)
            axes[i,j].xaxis.set_visible(False)
            axes[i,j].yaxis.set_visible(False)
    plt.show()
```

[ ]:
```
function_results_02(x_train,nRow=2,nCol=4)
```



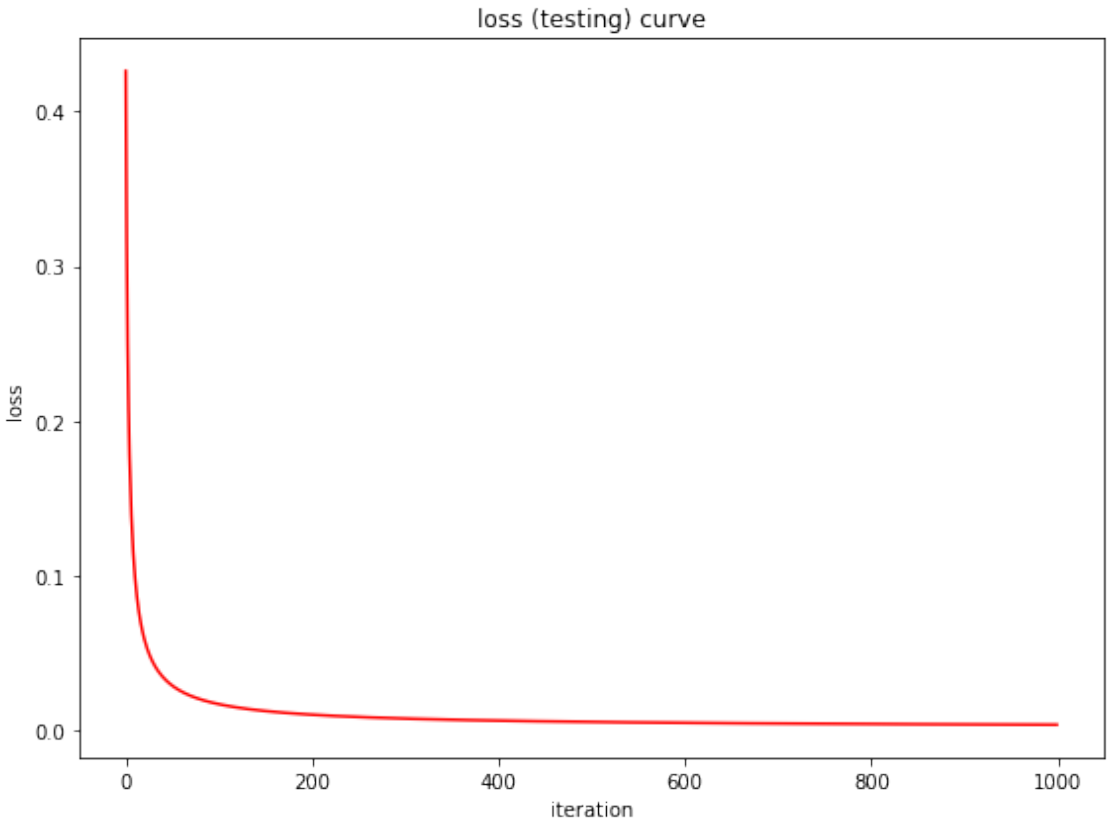
[ ]:
```
def function_results_03():
    plot_curve(loss_train_iteration, 'iteration', 'loss', 'loss (training)␣
↪curve')
```

[ ]:
```
function_results_03()
```



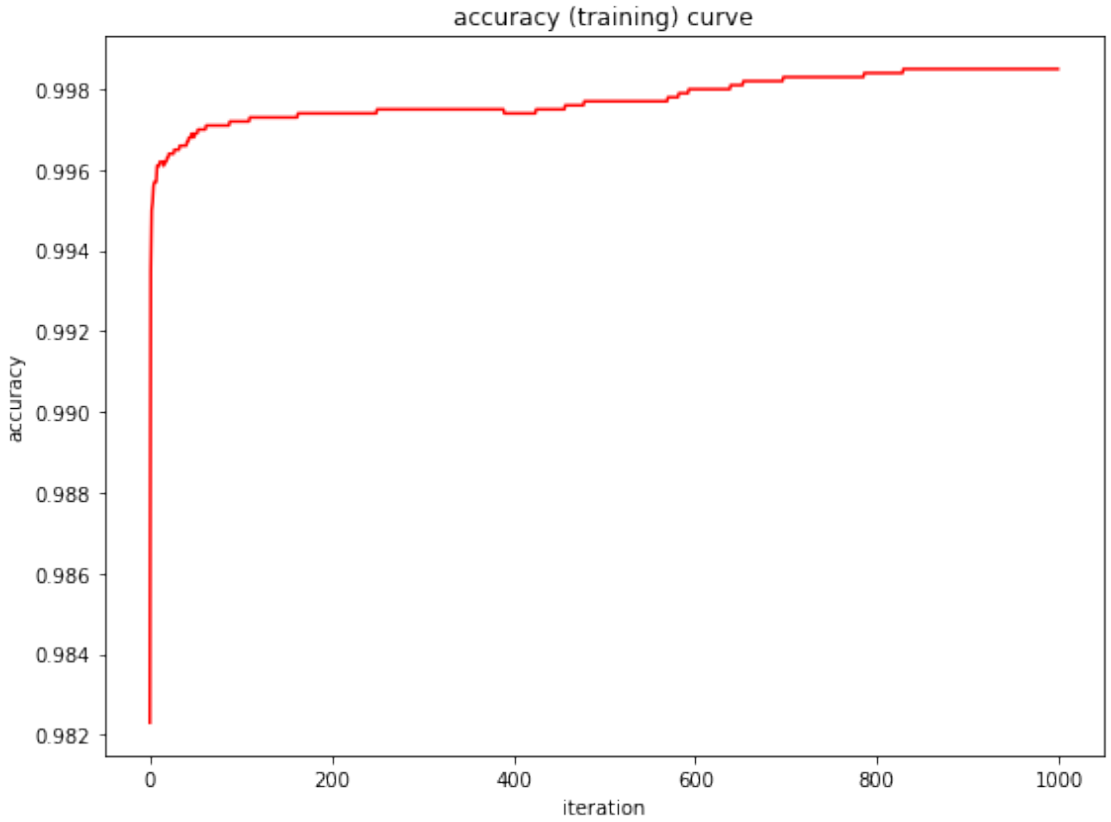
[ ]:
```
def function_results_04():
    plot_curve(loss_test_iteration, 'iteration', 'loss', 'loss (testing) curve')
```

[ ]:
```
function_results_04()
```



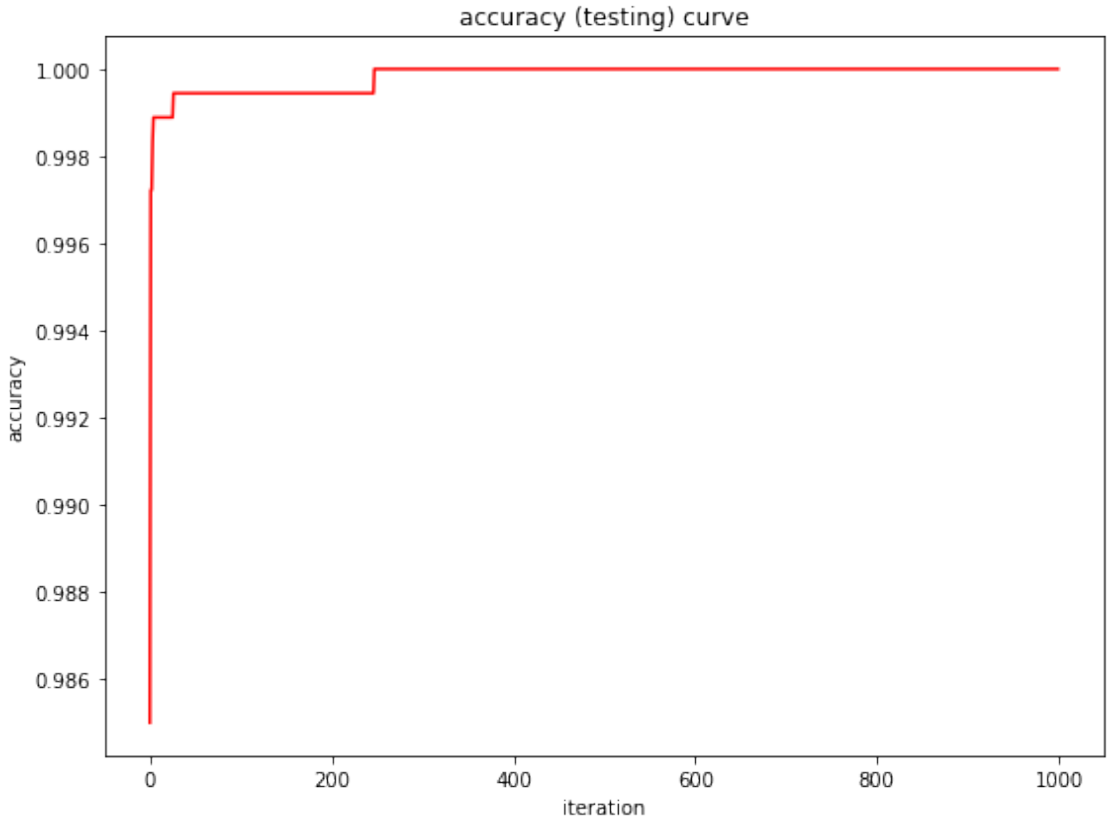
[ ]:
```
def function_results_05():
    plot_curve(accuracy_train_iteration, 'iteration', 'accuracy', 'accuracy␣
↪(training) curve')
```

[ ]:
```
function_results_05()
```



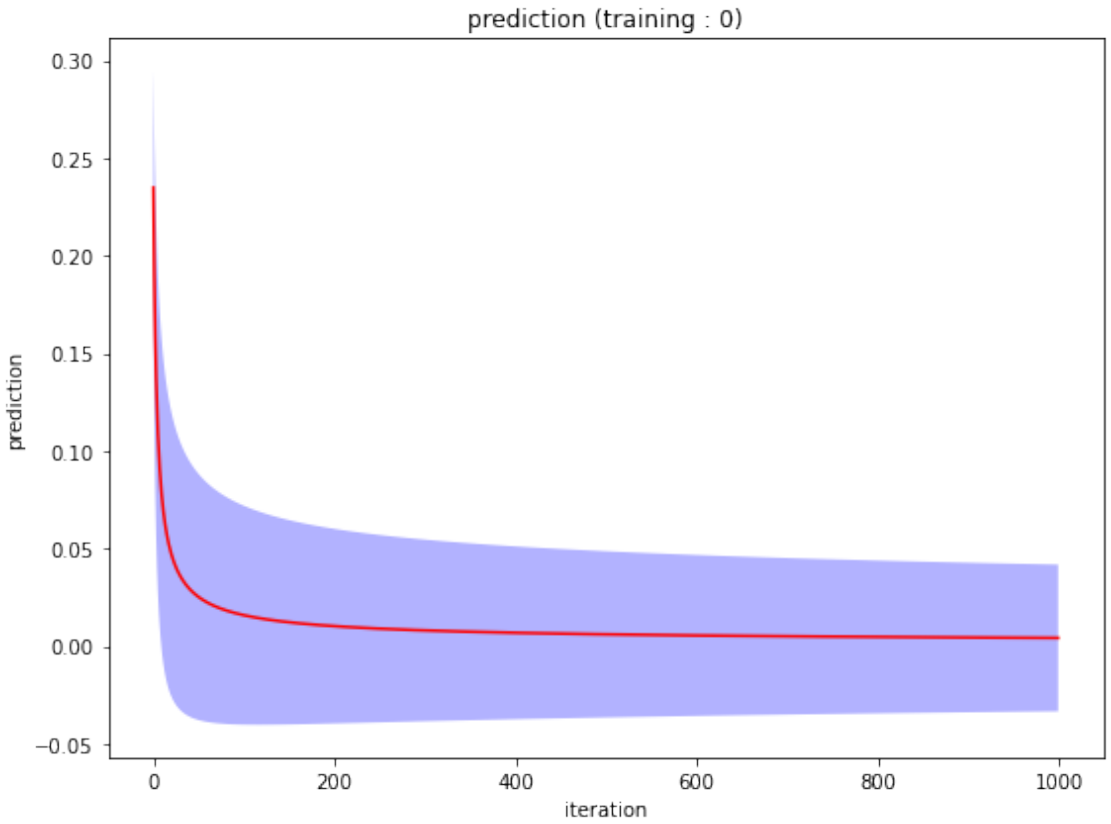
[ ]:
```
def function_results_06():
    plot_curve(accuracy_test_iteration, 'iteration', 'accuracy', 'accuracy␣
↪(testing) curve')
```

[ ]:
```
function_results_06()
```



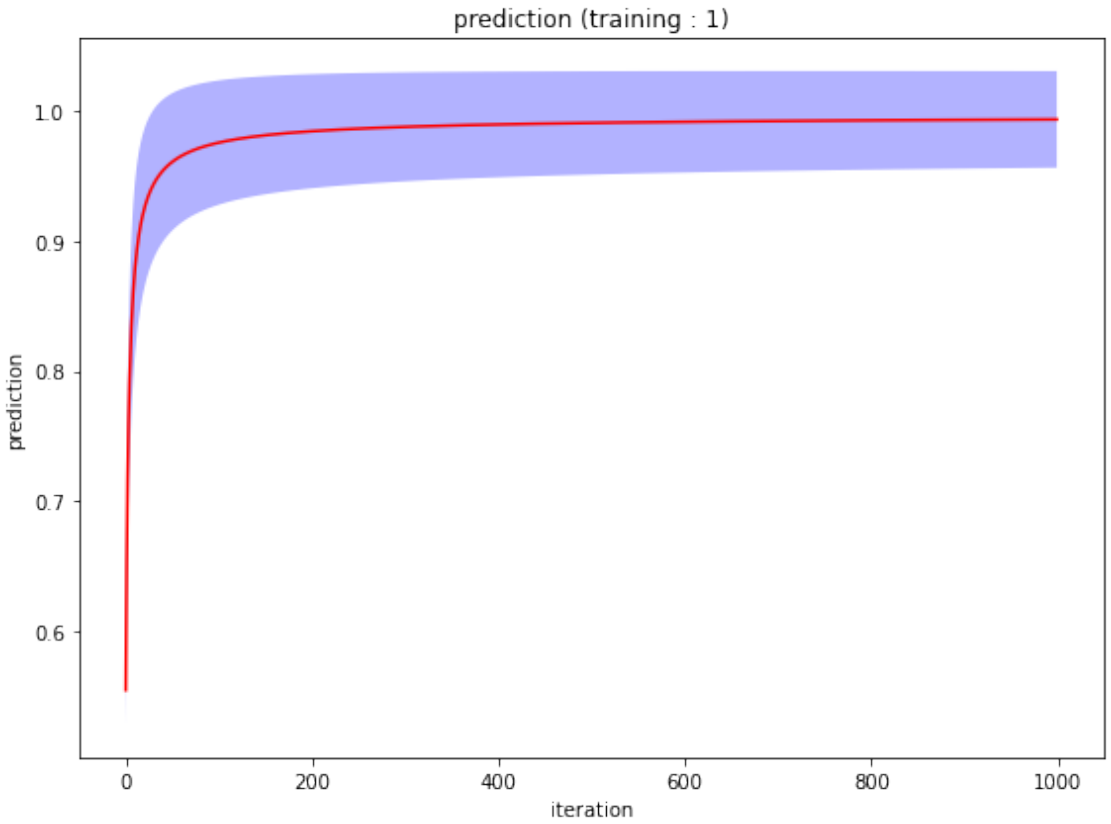
[ ]:
```
def function_results_07():
    plot_curve_error(pred_0_train_mean_iteration, pred_0_train_std_iteration,␣
↪'iteration', 'prediction', 'prediction (training : 0)')
```

[ ]:
```
function_results_07()
```



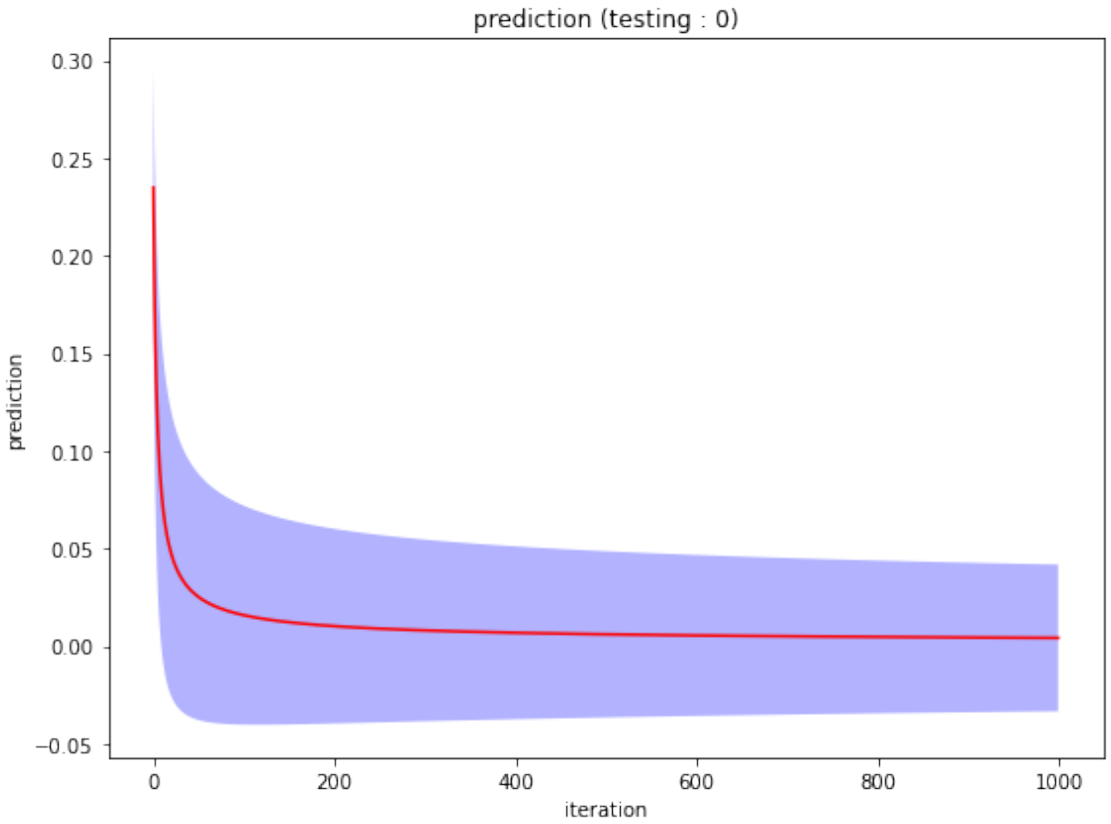
[ ]:
```
def function_results_08():
    plot_curve_error(pred_1_train_mean_iteration, pred_1_train_std_iteration,␣
↪'iteration', 'prediction', 'prediction (training : 1)')
```

[ ]:
```
function_results_08()
```



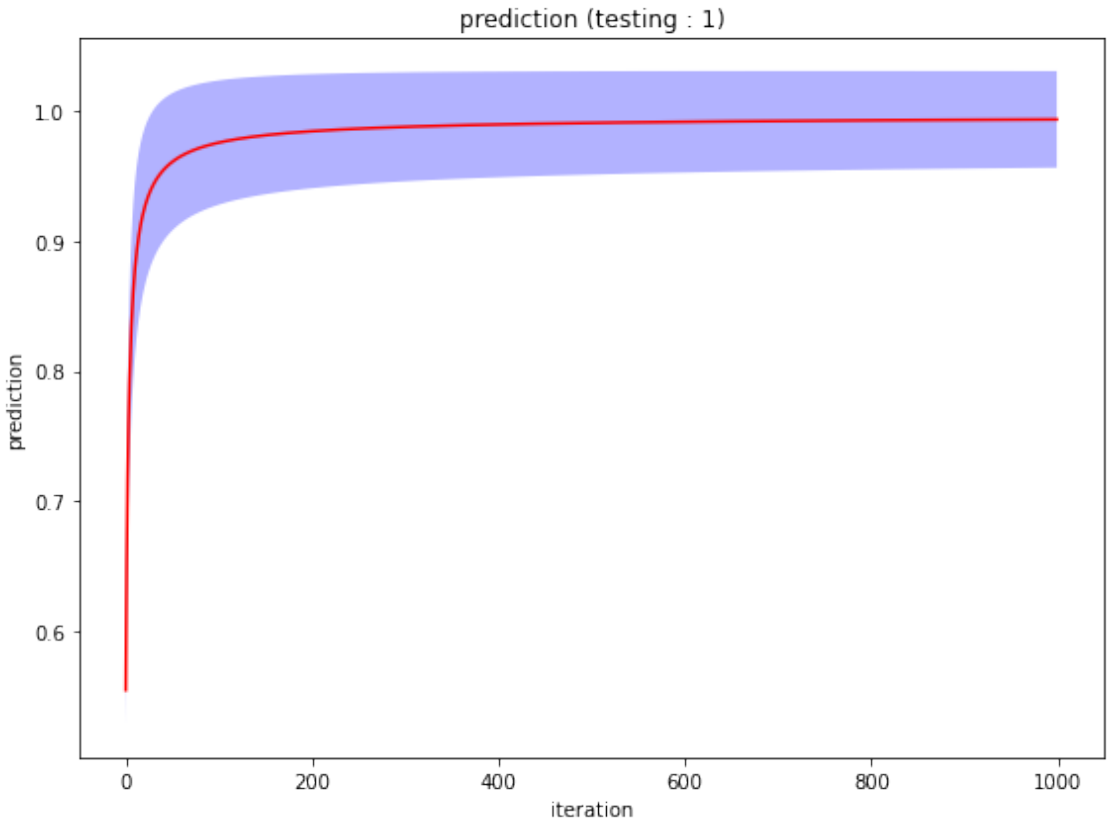
[ ]:
```
def function_results_09():
    plot_curve_error(pred_0_test_mean_iteration, pred_0_test_std_iteration,␣
↪'iteration', 'prediction', 'prediction (testing : 0)')
```

[ ]:
```
function_results_09()
```



[ ]:
```
def function_results_10():
    plot_curve_error(pred_1_test_mean_iteration, pred_1_test_std_iteration,␣
↪'iteration', 'prediction', 'prediction (testing : 1)')
```

[ ]:
```
function_results_10()
```



[ ]:
```
def function_results_11(data):

    index = np.array([0,100,200,300,400,500,600,700,800,900])

    for idx in index:
        val = data[idx]
        print('index = %4d, value = %12.10f' % (idx, val))
```

[ ]:
```
function_results_11(loss_train_iteration)
```

```
index =    0, value = 0.4307224938
index =  100, value = 0.0228587163
index =  200, value = 0.0152848437
index =  300, value = 0.0123252409
index =  400, value = 0.0106733249
index =  500, value = 0.0095914393
index =  600, value = 0.0088148843
index =  700, value = 0.0082233149
index =  800, value = 0.0077534390
index =  900, value = 0.0073684871
```

[ ]:
```
def function_results_12(data):

    index = np.array([0,100,200,300,400,500,600,700,800,900])

    for idx in index:
        val = data[idx]
        print('index = %4d, value = %12.10f' % (idx, val))
```

[ ]:
```
function_results_12(loss_test_iteration)
```

```
index =    0, value = 0.4263057540
index =  100, value = 0.0167458571
index =  200, value = 0.0099479648
index =  300, value = 0.0074345387
index =  400, value = 0.0060961283
index =  500, value = 0.0052563282
index =  600, value = 0.0046772451
index =  700, value = 0.0042526445
index =  800, value = 0.0039275776
index =  900, value = 0.0036706180
```

[ ]:
```
def function_results_13(data):

    index = np.array([0,100,200,300,400,500,600,700,800,900])

    for idx in index:
        val = data[idx]
        print('index = %4d, value = %12.10f' % (idx, val))
```

[ ]:
```
function_results_13(accuracy_train_iteration)
```

```
index =    0, value = 0.9823000000
index =  100, value = 0.9972000000
index =  200, value = 0.9974000000
index =  300, value = 0.9975000000
index =  400, value = 0.9974000000
index =  500, value = 0.9977000000
index =  600, value = 0.9980000000
index =  700, value = 0.9983000000
index =  800, value = 0.9984000000
index =  900, value = 0.9985000000
```

[ ]:
```
def function_results_14(data):

    index = np.array([0,100,200,300,400,500,600,700,800,900])

    for idx in index:
```

```
        val = data[idx]
        print('index = %4d, value = %12.10f' % (idx, val))
```

[ ]:
```
function_results_14(accuracy_test_iteration)
```

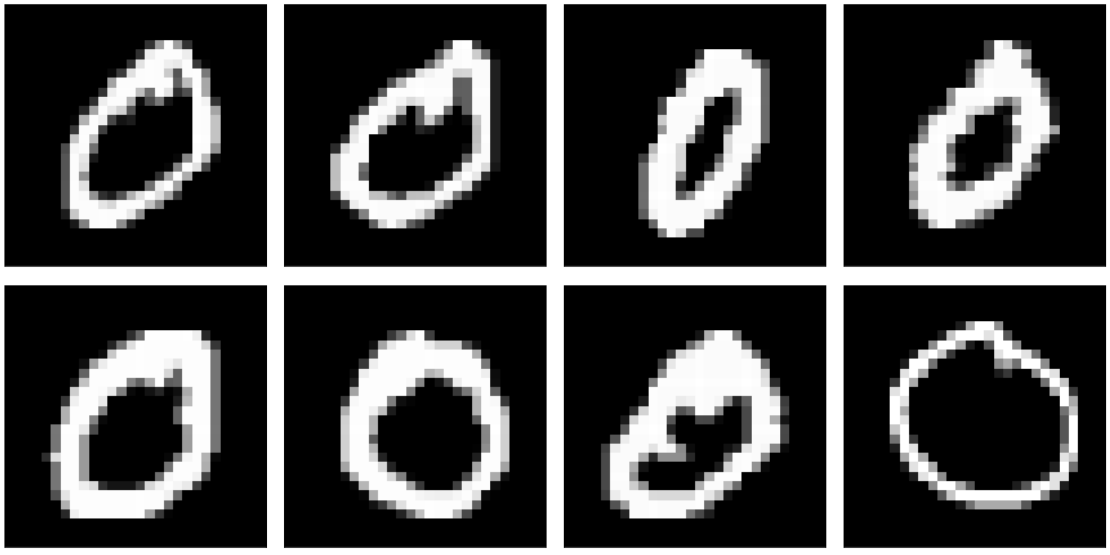
```
index =    0, value = 0.9850000000
index =  100, value = 0.9994444444
index =  200, value = 0.9994444444
index =  300, value = 1.0000000000
index =  400, value = 1.0000000000
index =  500, value = 1.0000000000
index =  600, value = 1.0000000000
index =  700, value = 1.0000000000
index =  800, value = 1.0000000000
index =  900, value = 1.0000000000
```

---

2 RESULTS

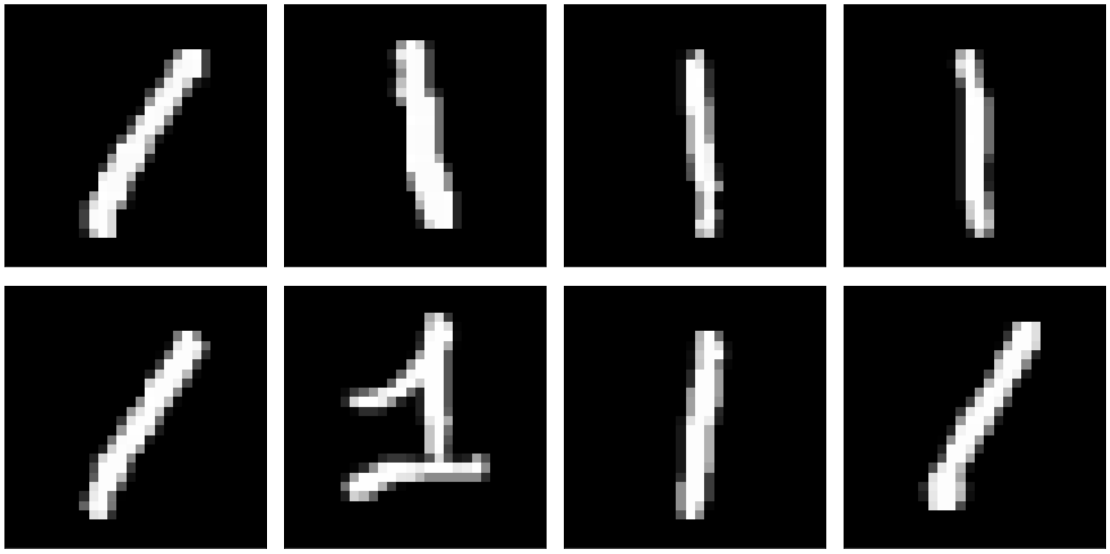
2.1 # 01. plot examples of the input training images for '0'

[ ]:
```
function_results_01(x_train,2,4)
```



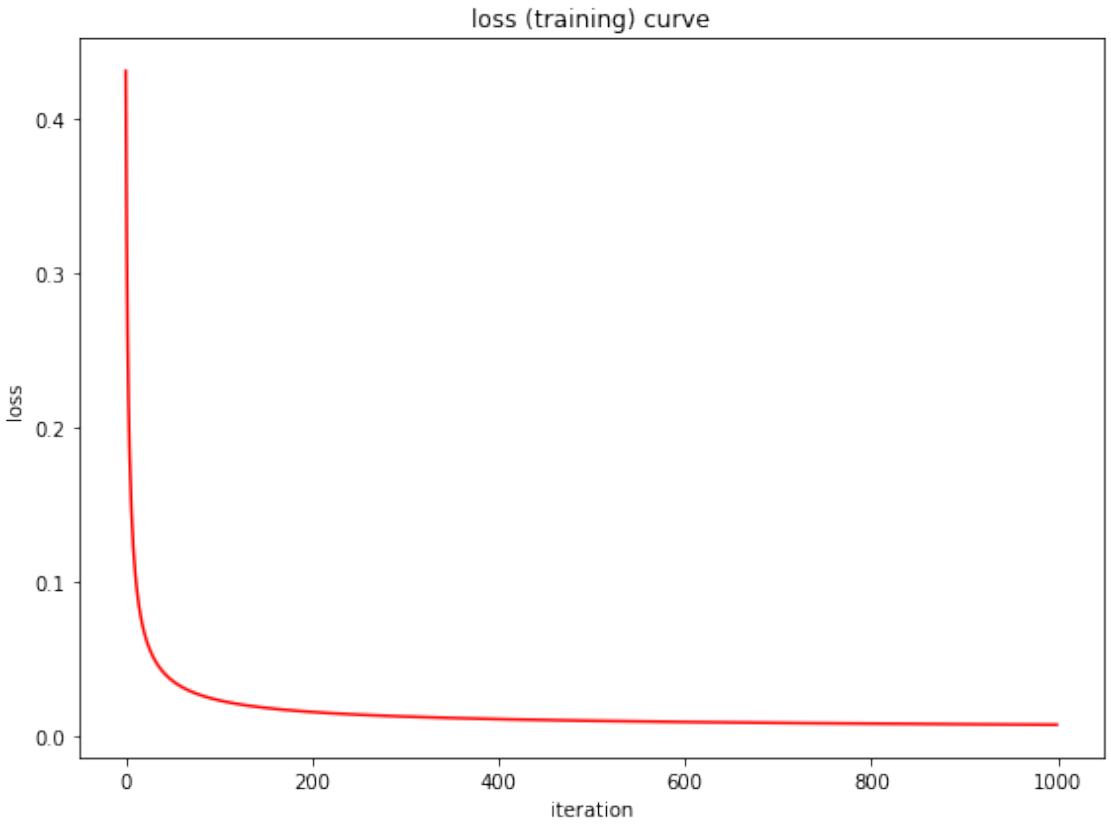
2.2 # 02. plot examples of the input training images for '1'

[ ]:
```
function_results_02(x_train,2,4)
```



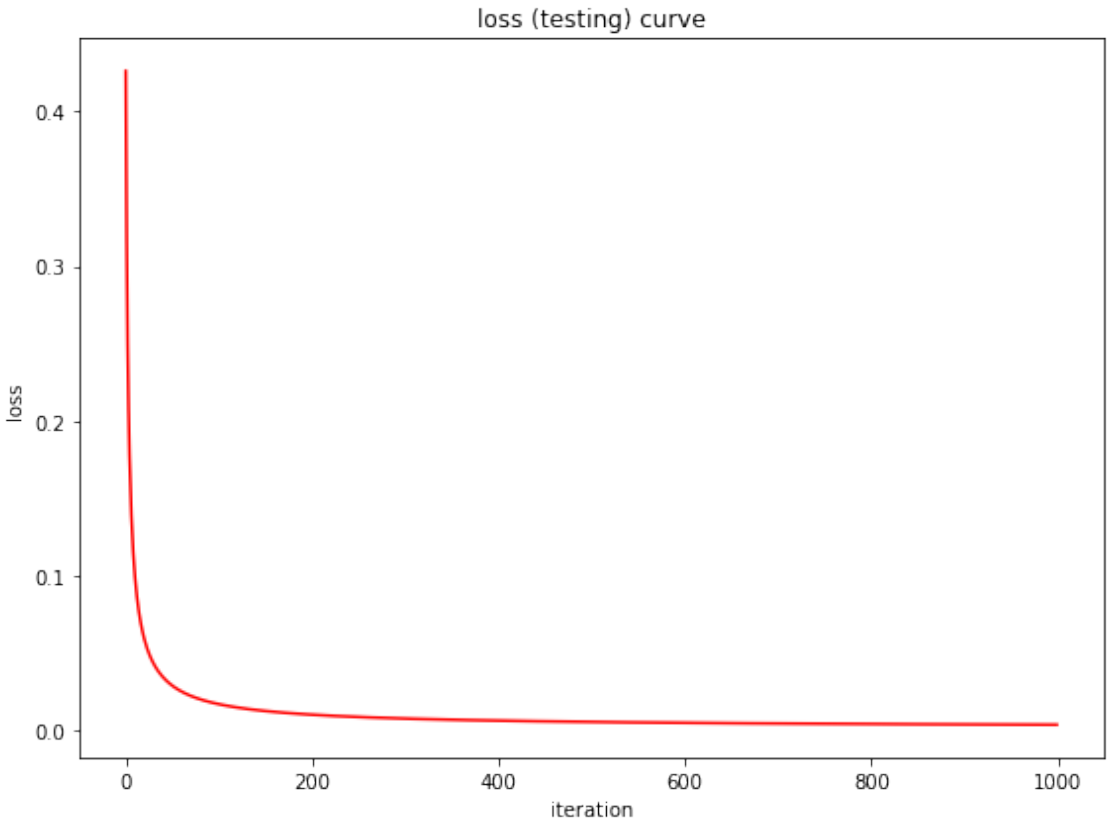
2.3 # 03. plot the training loss curve (x-axis: iteration, y-axis: loss)

[ ]:
```
function_results_03()
```



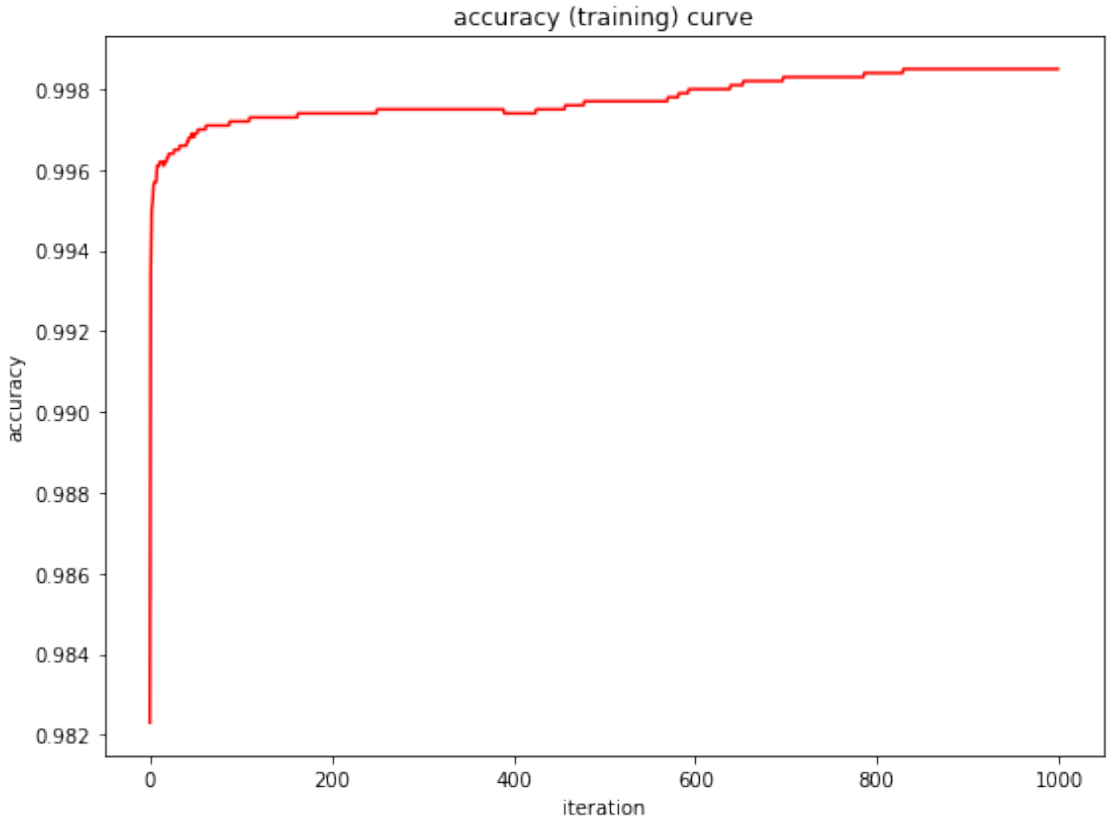
2.4 # 04. plot the testing loss curve (x-axis: iteration, y-axis: loss)

[ ]:
```
function_results_04()
```



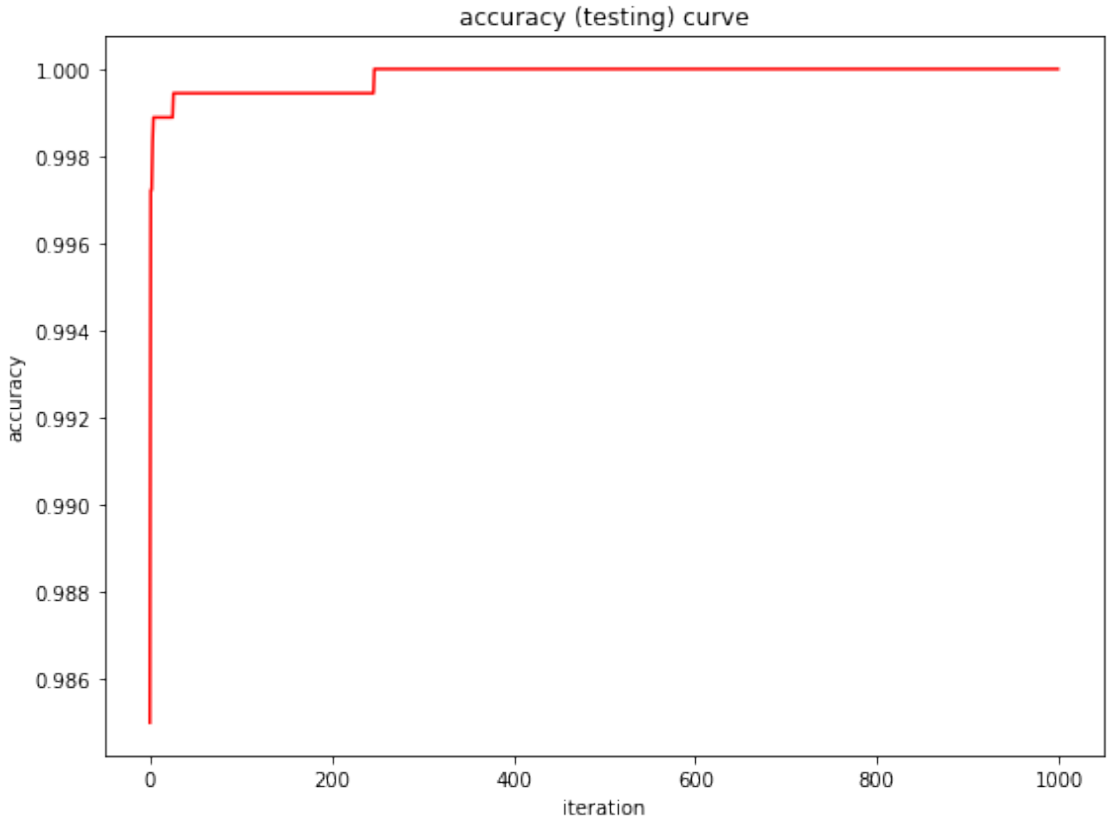
2.5 # 05. plot the training accuracy curve (x-axis: iteration, y-axis: accuracy)

```
[ ]: function_results_05()
```



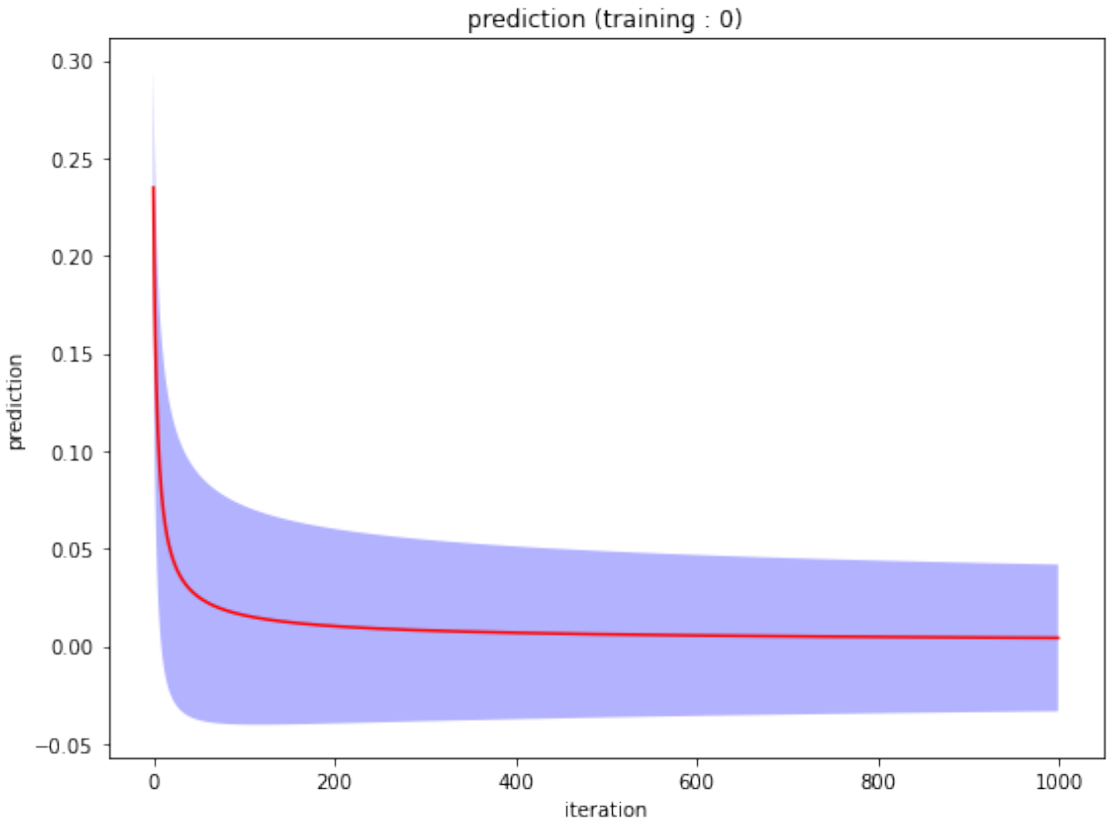
2.6 # 06. plot the testing accuracy curve (x-axis: iteration, y-axis: accuracy)

[ ]:
```
function_results_06()
```



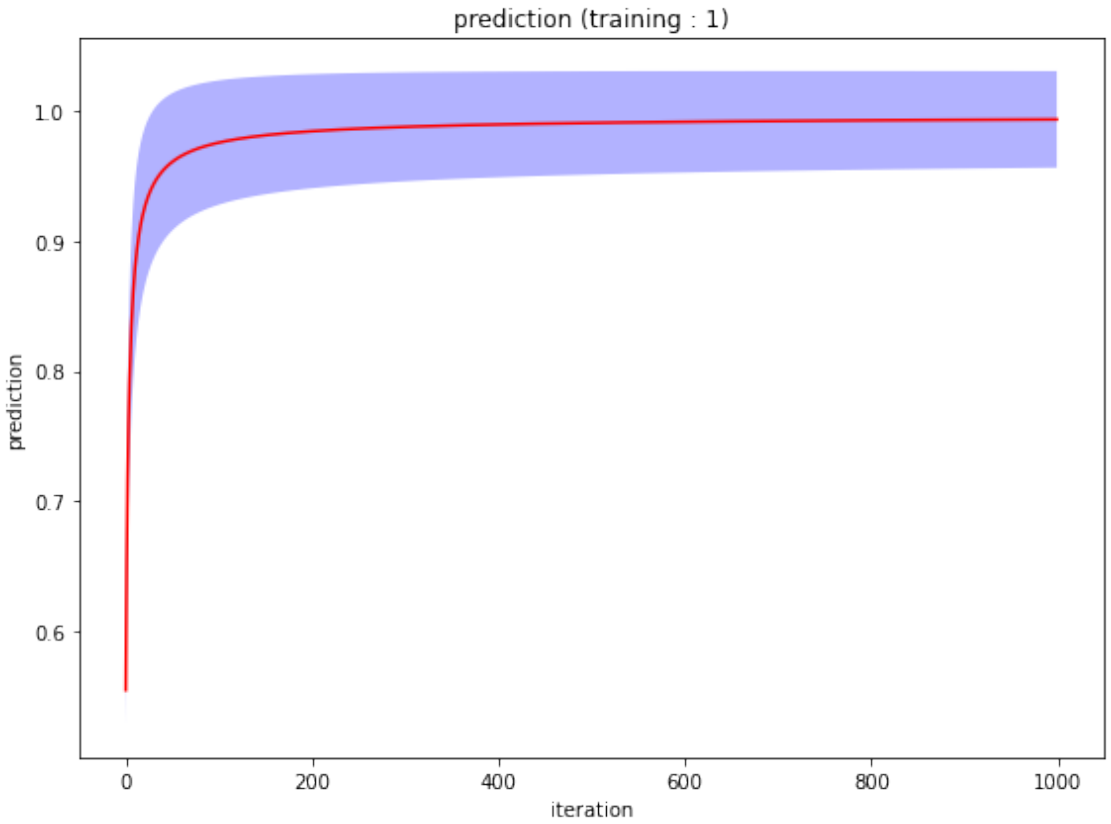
2.7 # 07. plot the training prediction curve (mean and std) for image 0 (x-axis: iteration, y-axis: prediction)

[ ]:
```
function_results_07()
```



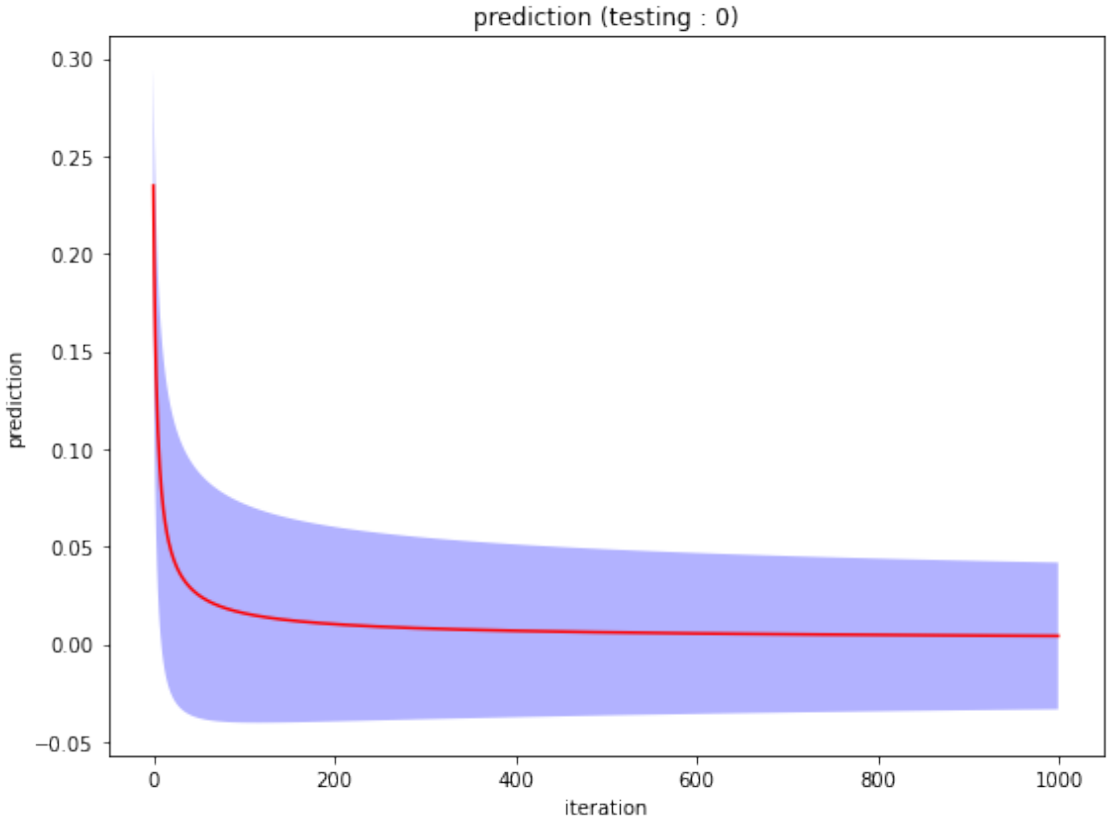
2.8 # 08. plot the training prediction curve (mean and std) for image 1 (x-axis: iteration, y-axis: prediction)

[ ]:
```
function_results_08()
```



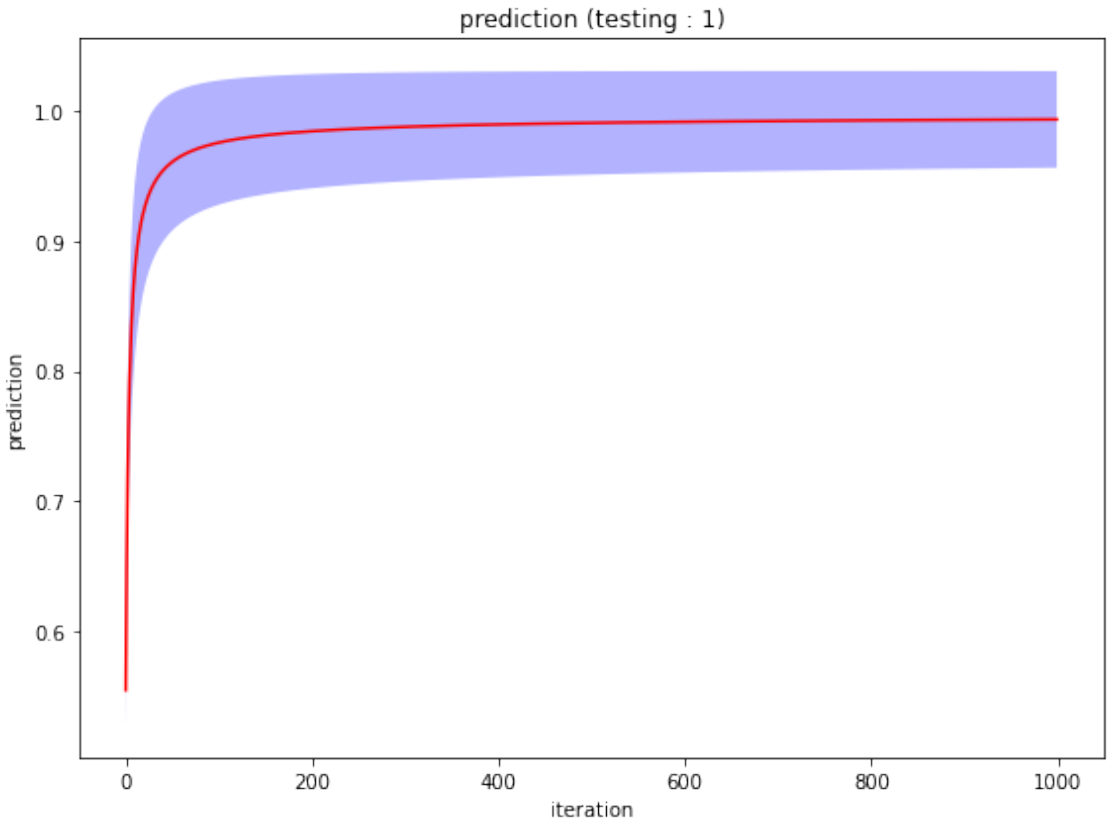
2.9 # 09. plot the testing prediction curve (mean and std) for image 0 (x-axis: iteration, y-axis: prediction)

[ ]:
```
function_results_09()
```



2.10 # 10. plot the testing prediction curve (mean and std) for image 1 (x-axis: iteration, y-axis: prediction)

[ ]:
```
function_results_10()
```



2.11 # 11. print the training loss at iterations 0, 100, 200, 300, 400, 500, 600, 700, 800, 900

[ ]:
```
function_results_11(loss_train_iteration)
```

```
index =    0, value = 0.4307224938
index =  100, value = 0.0228587163
index =  200, value = 0.0152848437
index =  300, value = 0.0123252409
index =  400, value = 0.0106733249
index =  500, value = 0.0095914393
index =  600, value = 0.0088148843
index =  700, value = 0.0082233149
index =  800, value = 0.0077534390
index =  900, value = 0.0073684871
```

2.12 # 12. print the testing loss at iterations 0, 100, 200, 300, 400, 500, 600, 700, 800, 900

[ ]:
```
function_results_12(loss_test_iteration)
```

```
index =    0, value = 0.4263057540
index =  100, value = 0.0167458571
index =  200, value = 0.0099479648
index =  300, value = 0.0074345387
index =  400, value = 0.0060961283
index =  500, value = 0.0052563282
index =  600, value = 0.0046772451
index =  700, value = 0.0042526445
index =  800, value = 0.0039275776
index =  900, value = 0.0036706180
```

2.13 # 13. print the training accuracy at iterations 0, 100, 200, 300, 400, 500, 600, 700, 800, 900

[ ]:
```
function_results_13(accuracy_train_iteration)
```

```
index =    0, value = 0.9823000000
index =  100, value = 0.9972000000
index =  200, value = 0.9974000000
index =  300, value = 0.9975000000
index =  400, value = 0.9974000000
index =  500, value = 0.9977000000
index =  600, value = 0.9980000000
index =  700, value = 0.9983000000
index =  800, value = 0.9984000000
index =  900, value = 0.9985000000
```

2.14 # 14. print the testing accuracy at iterations 0, 100, 200, 300, 400, 500, 600, 700, 800, 900

[ ]:
```
function_results_14(accuracy_test_iteration)
```

```
index =    0, value = 0.9850000000
index =  100, value = 0.9994444444
index =  200, value = 0.9994444444
index =  300, value = 1.0000000000
index =  400, value = 1.0000000000
index =  500, value = 1.0000000000
index =  600, value = 1.0000000000
index =  700, value = 1.0000000000
index =  800, value = 1.0000000000
index =  900, value = 1.0000000000
```