# A neural network for a classification with multiple labels

## import library

In [1]:

```
import numpy as np
import matplotlib.image as img
import matplotlib.pyplot as plt
import matplotlib.colors as colors
from matplotlib import ticker, cm
import os
from tqdm import tqdm
```

## load data

In [2]:

```
directory_data  = './'
filename_data   = 'assignment_04_data.npz'
path_data       = os.path.join(directory_data, filename_data)
data            = np.load(path_data)

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

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

x_train = np.asarray(x_train)
y_train = np.asarray(y_train)

x_test  = np.asarray(x_test)
y_test  = np.asarray(y_test)

vec_x_train = x_train.reshape(x_train.shape[0], x_train.shape[1] * x_train.shape
[2])
vec_x_test  = x_test.reshape(x_test.shape[0], x_test.shape[1] * x_test.shape[2])
```

In [3]:

```
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('size of vector_x_train :', vec_x_train.shape)
print('size of vector_x_test :', vec_x_test.shape)
print('*************************************************')
```

```
*************************************************
size of x_train : (10000, 28, 28)
size of y_train : (10000, 5)
*************************************************
size of x_test : (4500, 28, 28)
size of y_test : (4500, 5)
*************************************************
size of vector_x_train : (10000, 784)
size of vector_x_test : (4500, 784)
*************************************************
```

## index for each class

In [4]:

```
number_class     = y_train.shape[1]
length_data      = vec_x_train.shape[1]

index_train = {}
index_test  = {}

number_index_train   = np.zeros(number_class)
number_index_test    = np.zeros(number_class)

for i in range(number_class):

    index_train[i]   = np.where(y_train[:, i] == 1)
    index_test[i]    = np.where(y_test[:, i] == 1)

    number_index_train[i]    = np.shape(index_train[i])[1]
    number_index_test[i]     = np.shape(index_test[i])[1]
```

In [5]:

```
print('**************************************************')
print('number of classes :', number_class)
print('length of data :', length_data)
print('**************************************************')
print('number of training images for each class :', number_index_train)
print('number of testing images for each class :', number_index_test)
print('**************************************************')
```

```
**************************************************
number of classes : 5
length of data : 784
**************************************************
number of training images for each class : [2000. 2000. 2000. 2000.
2000.]
number of testing images for each class : [900. 900. 900. 900. 900.]
**************************************************
```

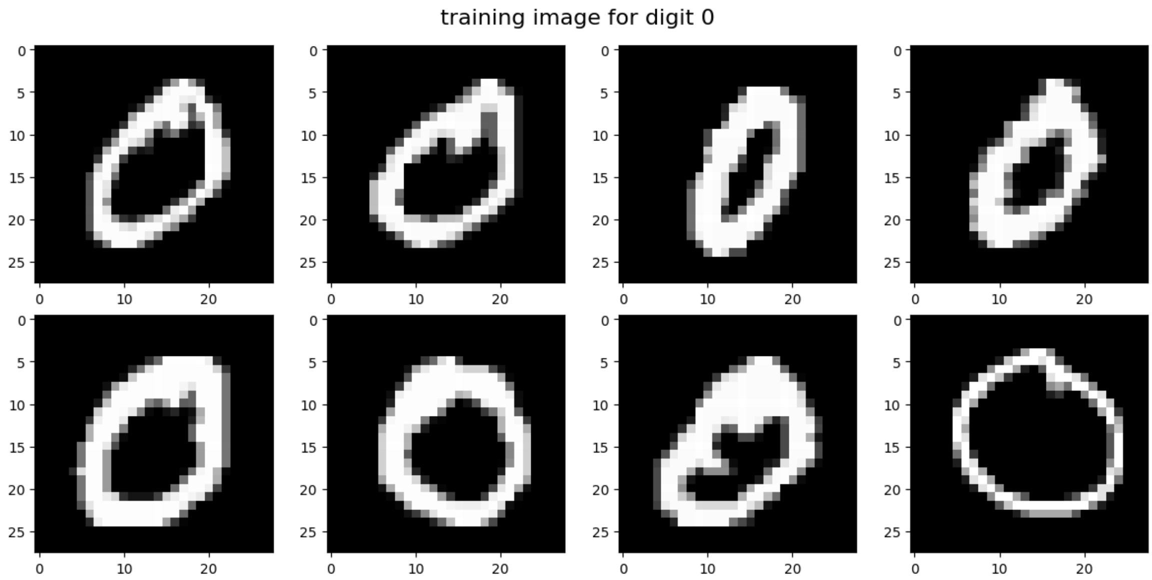
# plot grey image

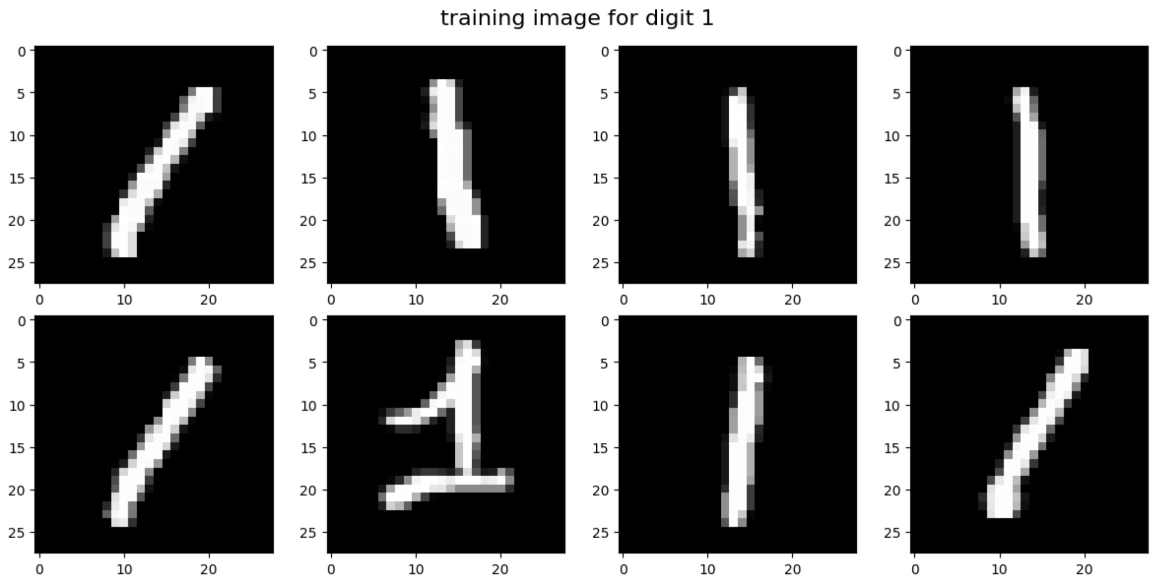
In [6]:

```
def plot_image(title, data):

    nRow = 2
    nCol = 4
    size = 3

    fig, axes = plt.subplots(nRow, nCol, figsize=(size * nCol, size * nRow))
    fig.suptitle(title, fontsize=16)

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

            k = i * nCol + j
            axes[i, j].imshow(data[k], cmap='gray', vmin=0, vmax=1)

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

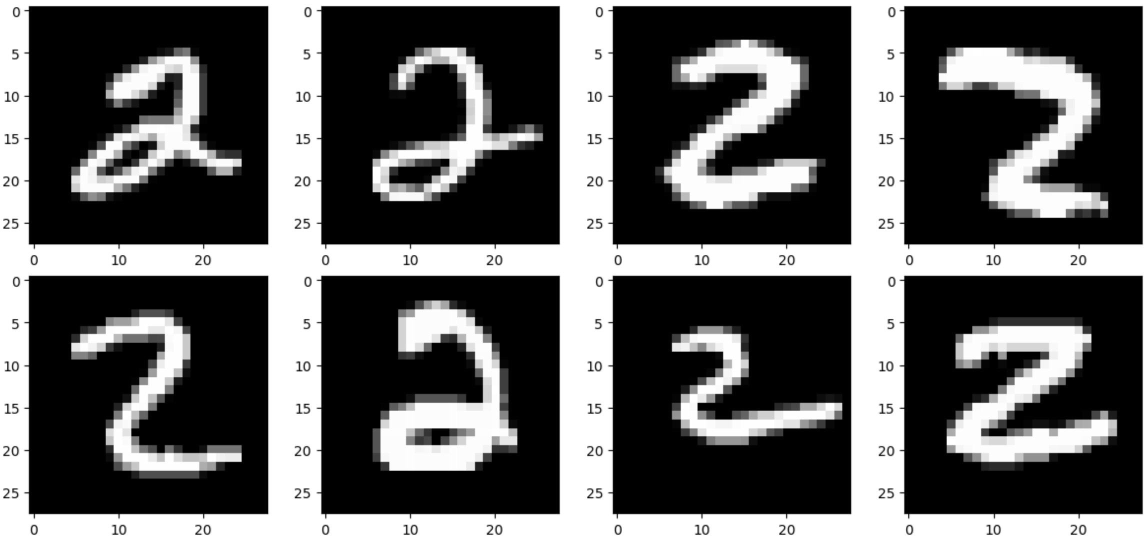
In [7]:

```
for c in range(number_class):

    index_class = c
    title       = 'training image for digit ' + str(index_class)
    plot_image(title, x_train[index_train[index_class][0]])
```

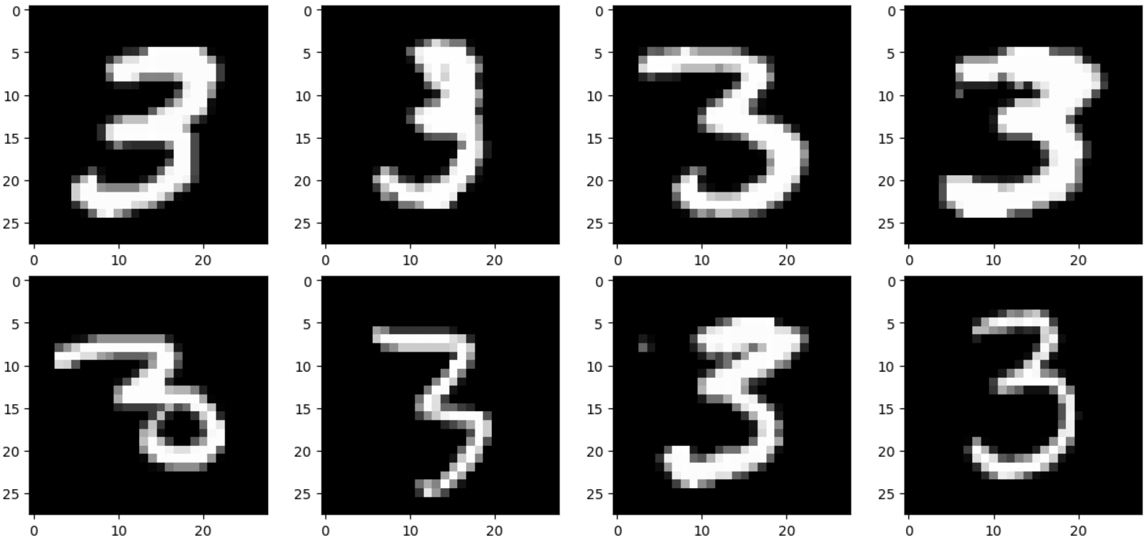

training image for digit 0


training image for digit 1

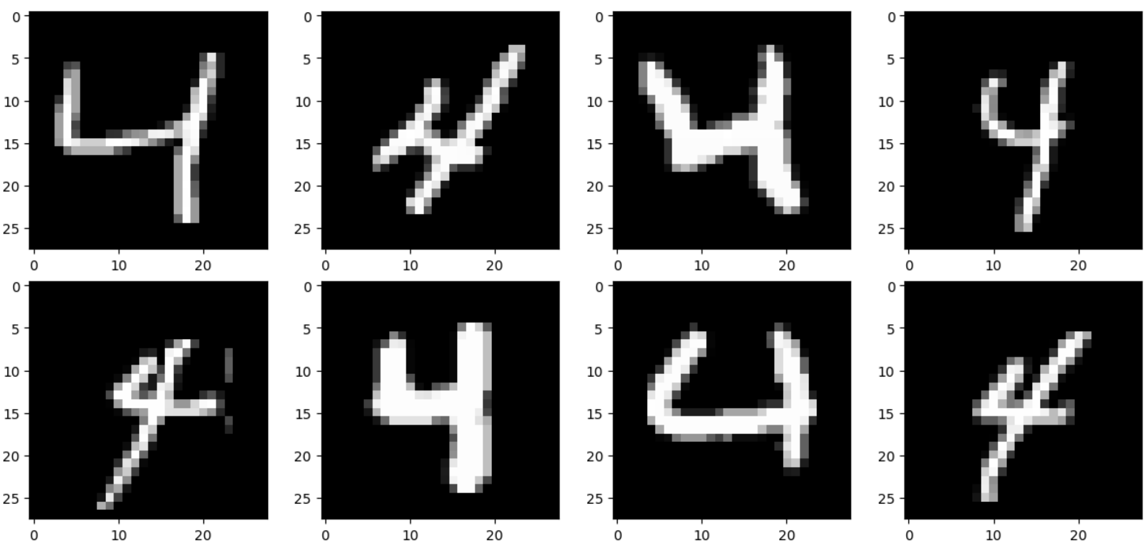
training image for digit 2



training image for digit 3



training image for digit 4



# initialize the neural network

- neural network consists of fullly connected linear layer followed by softmax activation function
- the size of the fully connected layer is input (length of data) and output (number of classes)

# initialize the weights for the fully connected layer

- create one matrix for the weights

In [82]:

```
size_input  = length_data
size_output = number_class

weight = np.ones((size_input, size_output))
```

In [83]:

```
print('size of the weight :', weight.shape)
```

```
size of the weight : (784, 5)
```

# define neural network

## define softmax function

- input : number of data $\times$ number of classes
- output : number of data $\times$ number of classes

In [84]:

```
def activation_softmax(input):

# ==================================================
# fill up the blank
#

    output = np.exp(input) / np.sum(np.exp(input), axis=1)[:,None]

#
# ==================================================

    return output
```

## define the layer

- input : number of data $\times$ length of data
- weight : length of data $\times$ number of classes
- output : number of data $\times$ number of classes

In [85]:

```
def layer_fully_connected(input, weight):

# ==================================================
# fill up the blank
#

    output = np.matmul(input, weight)

#
# ==================================================

    return output
```

## define forward propagation

- input : number of data $\times$ length of data
- weight : length of data $\times$ number of classes
- prediction : number of data $\times$ number of classes

In [86]:

```
def compute_prediction(input, weight):

# ==================================================
# fill up the blank
#

    prediction = activation_softmax(layer_fully_connected(input, weight))

#
# ==================================================

    return prediction
```

## define the loss function

- cross entropy between the ground truth and the prediction
- cross entropy : $-\sum_k y_k \log(h_k)$
    - $y_k$ : $k$-th element in grount truth
    - $h_k$ : $k$-th element in prediction
- prediction : number of data $\times$ number of classes
- label : number of data $\times$ number of classes
- loss : number of data $\times$ 1

In [87]:

```
def compute_loss(prediction, label):

# ==================================================
# fill up the blank
#

    loss = np.sum(-1. * label * np.log(prediction), axis=1)

#
# ==================================================

    return loss
```

## compute the accuracy

- prediction : number of data × number of classes
- label : number of data × number of classes
- accuracy : scalar
- note that iterations over the input data are not allowed inside the function

In [88]:

```
def compute_accuracy(prediction, label):

# ==================================================
# fill up the blank
#

    accuracy = np.sum(np.argmax(prediction, axis=1)==np.argmax(label, axis=1)) / 
label.shape[0]

#
# ==================================================

    return accuracy
```

## compute the gradient with respect to the weights

- note that iterations over the input data are not allowed inside the function
- input : number of data × length of data
- label : number of data × number of classes
- prediction : number of data × number of classes
- gradient : length of data × number of classes

In [89]:

```
def compute_gradient_weight(input, label, prediction):

# ====================================================
# fill up the blank
#

    gradient = np.matmul(input.T, (prediction - label)) / input.shape[0]

#
# ====================================================

    return gradient
```

## gradient descent algorithm

- hyper-parameters

In [90]:

```
number_iteration    = 1000
learning_rate       = 0.05
weight              = weight * 0.001
```

## variables for storing intermediate results

In [91]:

```
accuracy_train  = np.zeros(number_iteration)
accuracy_test   = np.zeros(number_iteration)

loss_train_mean = np.zeros(number_iteration)
loss_train_std  = np.zeros(number_iteration)
loss_test_mean  = np.zeros(number_iteration)
loss_test_std   = np.zeros(number_iteration)

prediction_train_mean = np.zeros((number_class, number_iteration))
prediction_test_mean  = np.zeros((number_class, number_iteration))
```

## run the gradient descent algorithm

In [92]:

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

# ==================================================
# fill up the blank
#

    prediction_train = compute_prediction(vec_x_train, weight)
    loss_train = compute_loss(prediction_train, y_train)
    gradient_train = compute_gradient_weight(vec_x_train, y_train, prediction_tr
ain)

    weight = weight - learning_rate * gradient_train

    prediction_test = compute_prediction(vec_x_test, weight)
    loss_test = compute_loss(prediction_test, y_test)

#
# ==================================================

    accuracy_train[i]   = compute_accuracy(prediction_train, y_train)
    accuracy_test[i]    = compute_accuracy(prediction_test, y_test)

    loss_train_mean[i]  = np.mean(loss_train)
    loss_train_std[i]   = np.std(loss_train)
    loss_test_mean[i]   = np.mean(loss_test)
    loss_test_std[i]    = np.std(loss_test)

    for c in range(number_class):

        prediction_train_mean[c, i]   = np.mean(prediction_train[index_train[c],
c])
        prediction_test_mean[c, i]    = np.mean(prediction_test[index_test[c],c
])
```

```
100%|████████████████████████████████████████| 1000/1000 [00:40<00:0
0, 24.57it/s]
```

## functions for presenting the results

In [68]:

```
def function_result_01():

    title           = 'loss (training)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'loss'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(loss_train_mean)), loss_train_mean, '-', color = color_me
an)
    plt.fill_between(range(len(loss_train_mean)), loss_train_mean - loss_train_s
td, loss_train_mean + loss_train_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [69]:

```
def function_result_02():

    title           = 'loss (testing)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'loss'
    color_mean      = 'red'
    color_std       = 'blue'
    alpha           = 0.3

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

    plt.plot(range(len(loss_test_mean)), loss_test_mean, '-', color = color_mean
)
    plt.fill_between(range(len(loss_test_mean)), loss_test_mean - loss_test_std,
loss_test_mean + loss_test_std, facecolor = color_std, alpha = alpha)

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [70]:

```
def function_result_03():

    title           = 'accuracy (training)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'accuracy'

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

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

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [71]:

```
def function_result_04():

    title           = 'accuracy (testing)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'accuracy'

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

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

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)

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

In [72]:

```
def function_result_05():

    title           = 'prediction (training)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'prediction'

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

    for c in range(number_class):

        plt.plot(prediction_train_mean[c], '-', label=str(c))

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)
    plt.legend()

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

In [73]:

```
def function_result_06():

    title           = 'prediction (testing)'
    label_axis_x    = 'iteration'
    label_axis_y    = 'prediction'

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

    for c in range(number_class):

        plt.plot(prediction_test_mean[c], '-', label=str(c))

    plt.xlabel(label_axis_x)
    plt.ylabel(label_axis_y)
    plt.legend()

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

In [74]:

```
def function_result_07():

    print('final training loss = %6.5f' % (loss_train_mean[-1]))
```

In [75]:

```
def function_result_08():

    print('final testing loss = %6.5f' % (loss_test_mean[-1]))
```

In [76]:

```
def function_result_09():

    print('final training accuracy = %6.5f' % (accuracy_train[-1]))
```

In [77]:

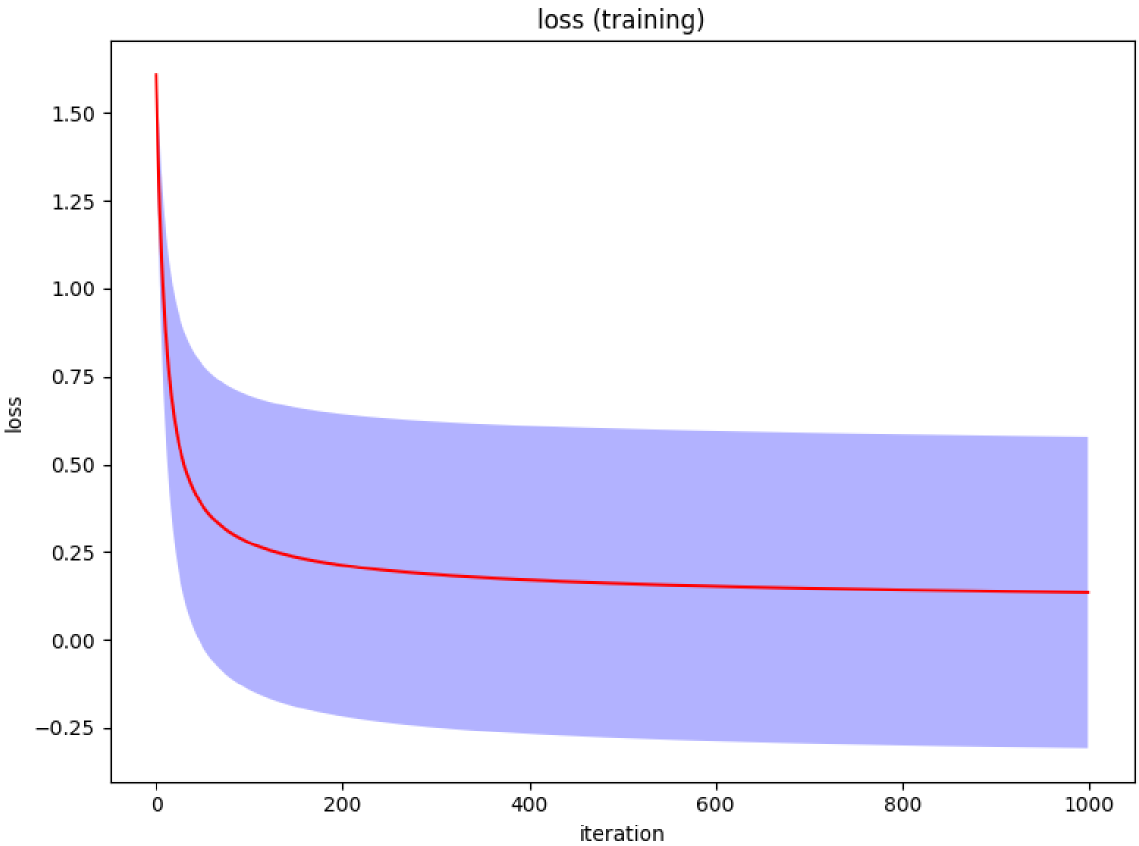
```
def function_result_10():

    print('final testing accuracy = %6.5f' % (accuracy_test[-1]))
```

# results

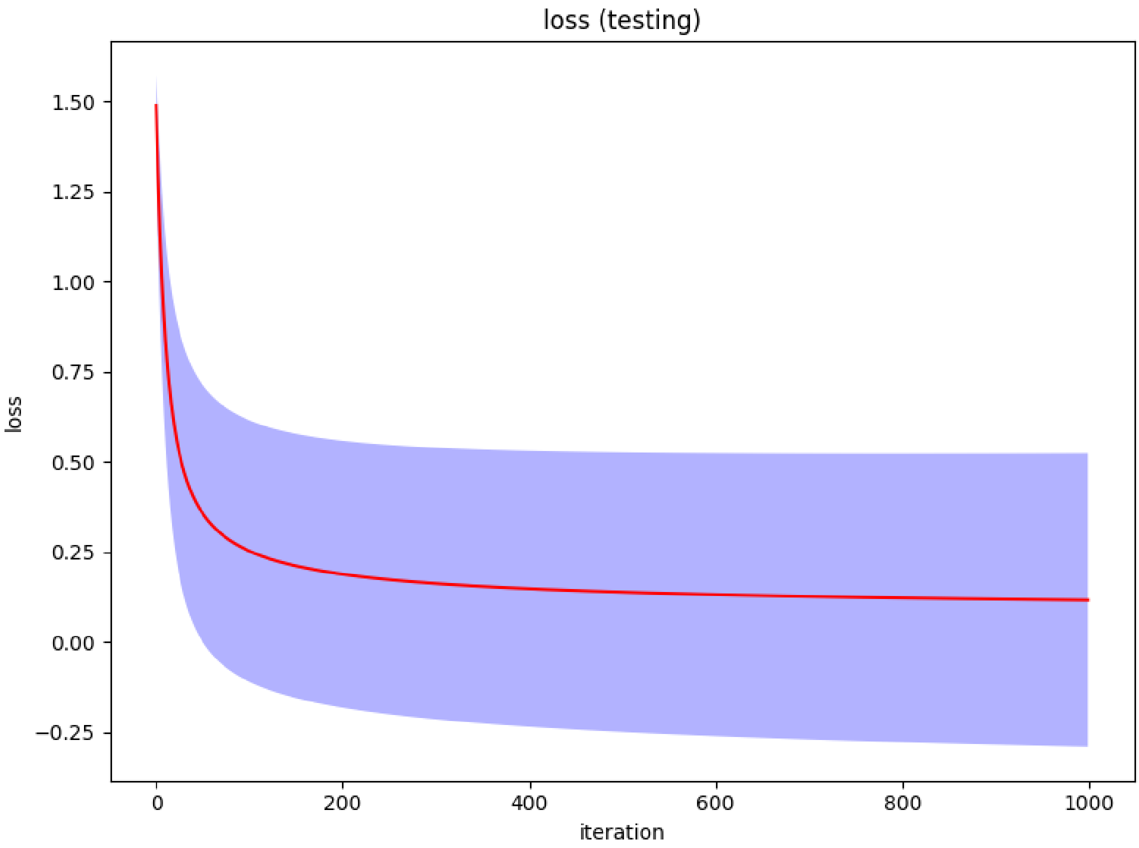
In [93]:

```
number_result = 10

for i in range(number_result):

    title           = '# RESULT # {:02d}'.format(i+1)
    name_function   = 'function_result_{:02d}()'.format(i+1)

    print('')
    print('################################################################################')
    print('#')
    print(title)
    print('#')
    print('################################################################################')
    print('')

    eval(name_function)
```

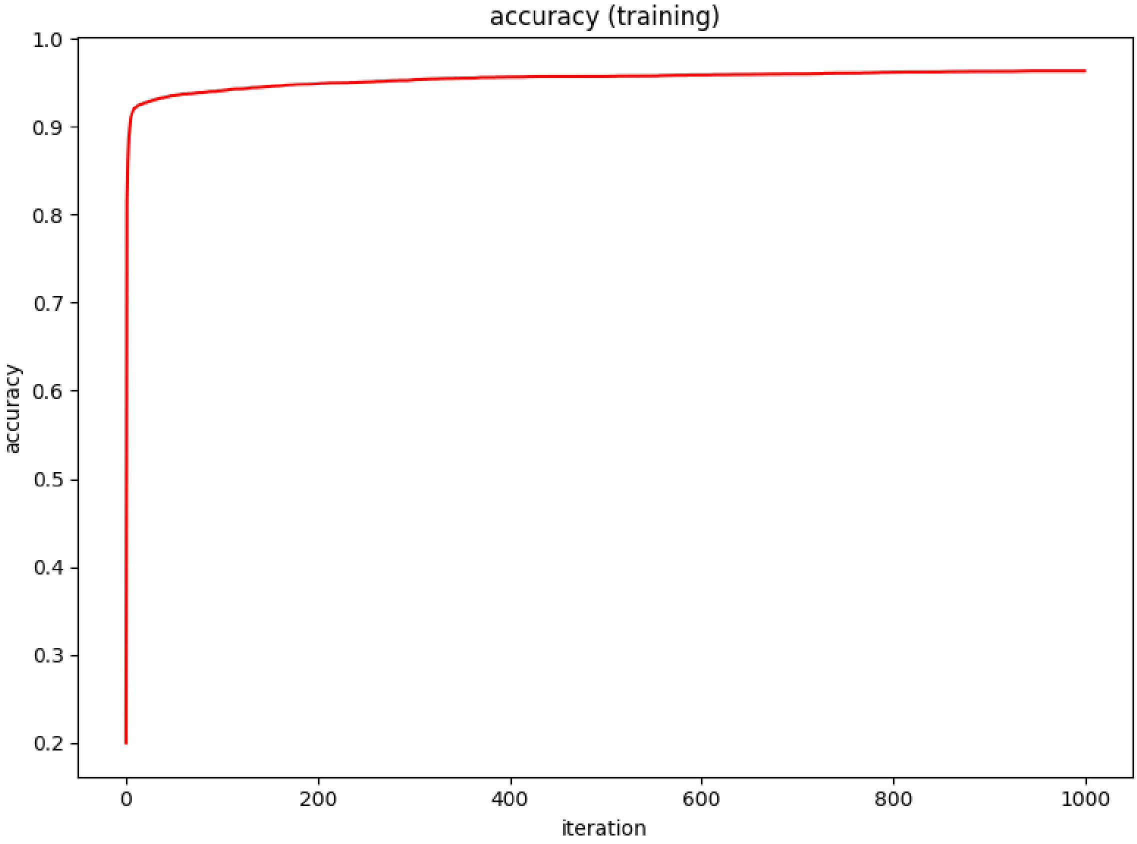
```
############################################################################
#
# RESULT # 01
#
############################################################################
```



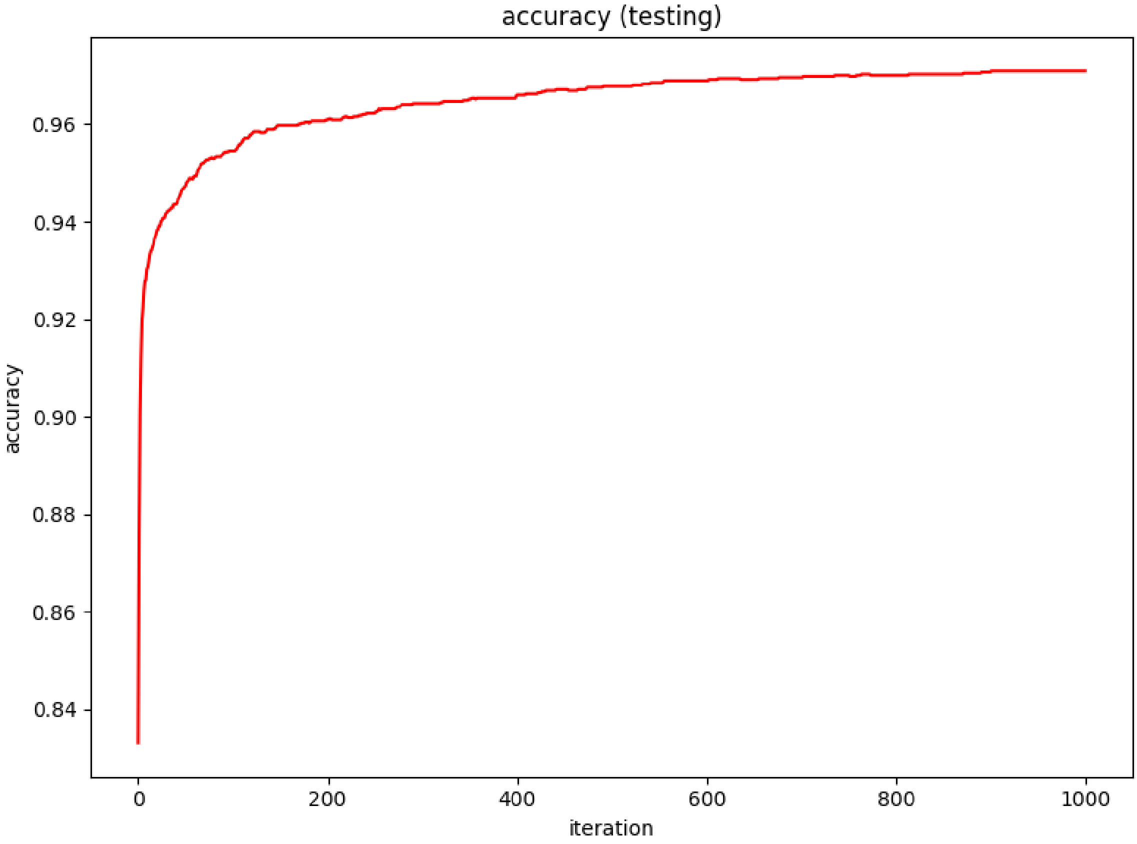

```
############################################################################
#
# RESULT # 02
#
############################################################################
```


loss (testing)
loss
1.50
1.25
1.00
0.75
0.50
0.25
0.00
−0.25
0
200
400
600
800
1000
iteration

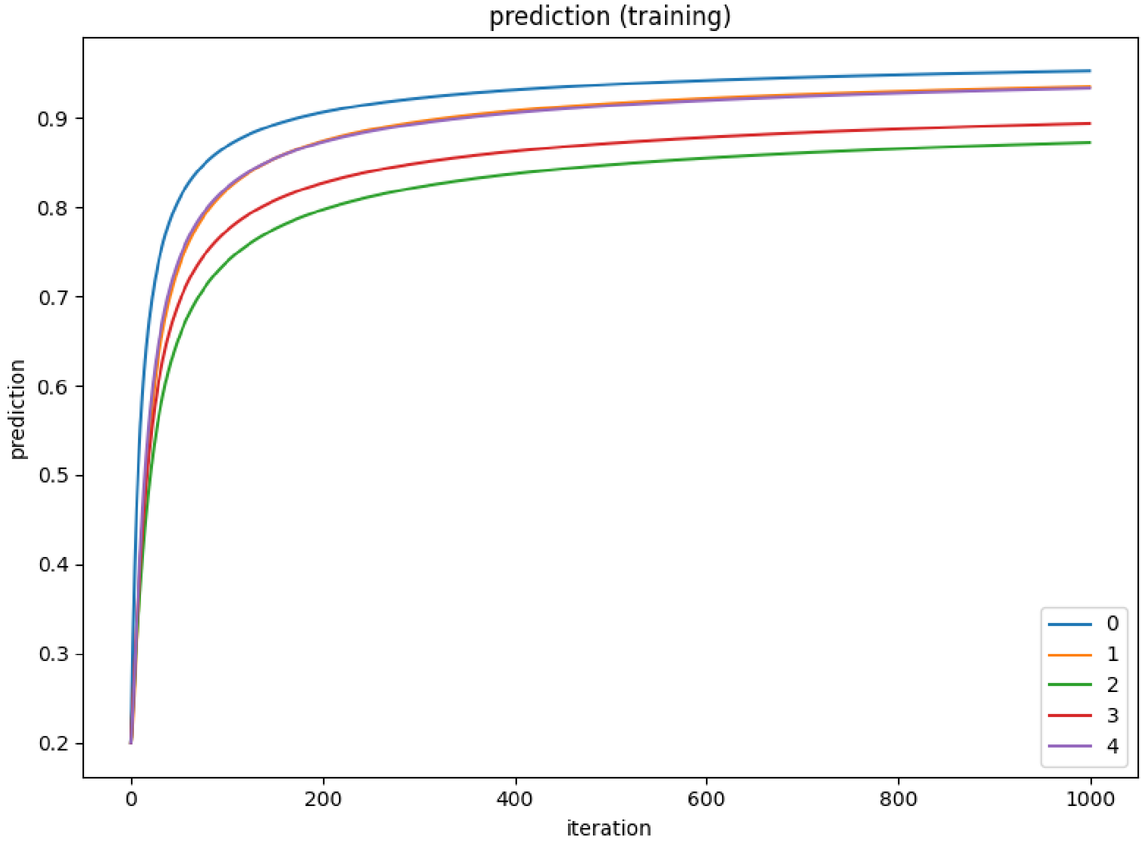
```
########################################################################
#
# RESULT # 03
#
########################################################################
```


accuracy (training)
accuracy
iteration
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
200
400
600
800
1000

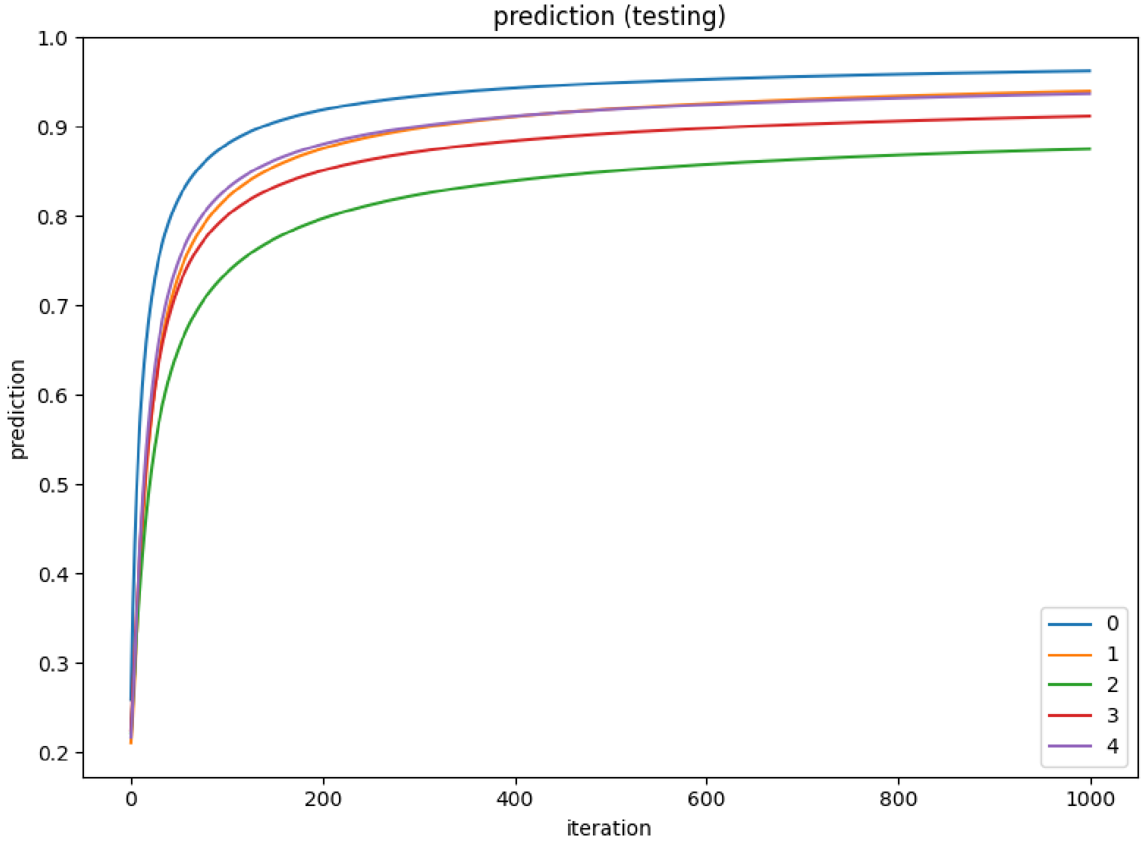
```
######################################################################
############
#
# RESULT # 04
#
######################################################################
############
```


accuracy (testing)
accuracy
0.96
0.94
0.92
0.90
0.88
0.86
0.84
0
200
400
600
800
1000
iteration

```
######################################################################
############
#
# RESULT # 05
#
######################################################################
############
```


prediction (training)
prediction
iteration
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
0
200
400
600
800
1000
0
1
2
3
4

```
########################################################################
############
#
# RESULT # 06
#
########################################################################
############
```


prediction (testing)
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
prediction
0
200
400
600
800
1000
iteration
0
1
2
3
4

```
########################################################################
#
# RESULT # 07
#
########################################################################

final training loss = 0.13488

########################################################################
#
# RESULT # 08
#
########################################################################

final testing loss = 0.11646

########################################################################
#
# RESULT # 09
#
########################################################################

final training accuracy = 0.96310

########################################################################
#
# RESULT # 10
#
########################################################################

final testing accuracy = 0.97089
```

In [ ]: