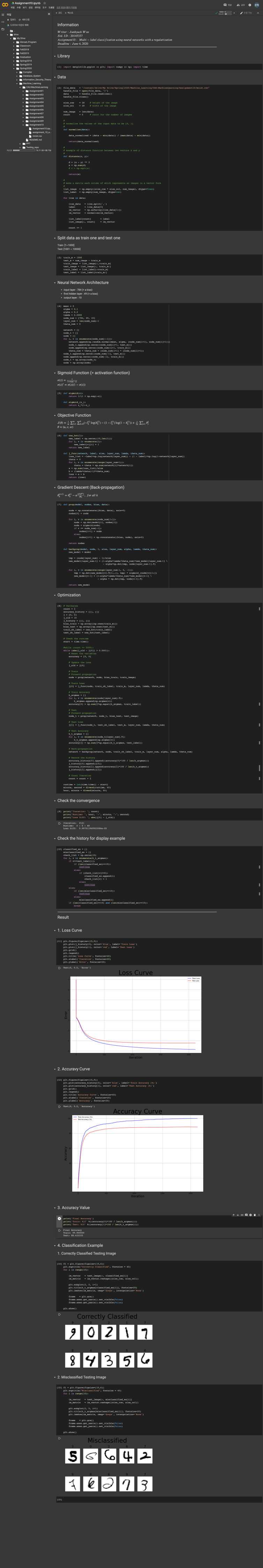

Information

Writer : Junhyuck Woo
Std. ID : 20145337
Assignment10 : Multi − label classification using neural networks with a regularization
Deadline : June 4, 2020

Library

```
[1] import matplotlib.pyplot as plt; import numpy as np; import time
```

Data

```
[2] file_data   = "/content/drive/My Drive/Spring_2020/Machine_Learning/CAU-MachineLearning/Assignment10/mnist.csv"
    handle_file = open(file_data, "r")
    data        = handle_file.readlines()
    handle_file.close()

    size_row    = 28    # height of the image
    size_col    = 28    # width of the image

    num_image   = len(data)
    count       = 0     # count for the number of images

    #
    # normalize the values of the input data to be [0, 1]
    #
    def normalize(data):

        data_normalized = (data - min(data)) / (max(data) - min(data))

        return(data_normalized)

    #
    # example of distance function between two vectors x and y
    #
    def distance(x, y):

        d = (x - y) ** 2
        s = np.sum(d)
        # r = np.sqrt(s)

        return(s)

    #
    # make a matrix each column of which represents an images in a vector form
    #
    list_image  = np.empty((size_row * size_col, num_image), dtype=float)
    list_label  = np.empty(num_image, dtype=int)

    for line in data:

        line_data   = line.split(',')
        label       = line_data[0]
        im_vector   = np.asfarray(line_data[1:])
        im_vector   = normalize(im_vector)

        list_label[count]       = label
        list_image[:, count]    = im_vector

        count += 1
```

Split data as train one and test one

Train: [1~1000]
Test: [1001 ~ 10000]

```
[3] train_m = 1000
    test_m = num_image - train_m
    train_image = list_image[:,:train_m]
    test_image = list_image[:, train_m:]
    train_label = list_label[:train_m]
    test_label = list_label[train_m:]
```

Neural Network Architecture

- input layer : 784 (+ a bias)
- first hidden layer : 49 (+ a bias)
- output layer : 10

```
[4] mean = 0
    sigma = 0.1
    alpha = 0.2
    lamda = 0.0005
    node_num = [784, 49, 10]
    layer_num = len(node_num)-1
    theta_num = 0

    network = []
    node_t = []
    node = []
    for i, v in enumerate(node_num[:-1]):
        network.append(np.random.normal(mean, sigma, (node_num[i+1], node_num[i]+1)))
        node_t.append(np.zeros((node_num[i]+1, test_m)))
        node.append(np.zeros((node_num[i]+1, train_m)))
        theta_num = theta_num + (node_num[i+1] * (node_num[i]+1))
    node_t.append(np.zeros((node_num[-1], test_m)))
    node.append(np.zeros((node_num[-1], train_m)))
    node_t = np.array(node_t)
    node = np.array(node)
```

Sigmoid Function (= activation function)

$\sigma(z) = \frac{1}{1+\exp(-z)}$
$\sigma(z)' = \sigma(z)(1 - \sigma(z))$

```
[5] def sigmoid(z):
        return 1/(1 + np.exp(-z))

    def sigmoid_(z_):
        return z_*(1-z_)
```

Objective Function

$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=0}^{9}(-l_k^{(i)}\log(h_k^{(i)}) - (1-l_k^{(i)})\log(1-h_k^{(i)})) + \frac{\lambda}{2n}\sum_{j=1}^{n}\theta_j^2$
$\theta = (u, v, w)$

```
[6] def one_hot(l):
        new_label = np.zeros((10,len(l)))
        for i, v in enumerate(l):
            new_label[v][i] = 1
        return new_label

    def j_func(network, label, size, layer_num, lamda, theta_num):
        loss_list = -label*np.log(network[layer_num]) - (1 - label)*np.log(1-network[layer_num])
        theta = 0
        for i, v in enumerate(range(layer_num+1)):
            theta = theta + np.sum(network[i]*network[i])
        a = np.sum(loss_list)/size
        b = (lamda*theta)/(2*theta_num)
        loss = a + b
        return (loss)
```

Gradient Descent (Back-propagation)

$\theta_k^{(t+1)} = \theta_k^{(t)} - \alpha\frac{\partial J(\theta^{(t)})}{\partial \theta_k}$, *for all k*

```
[7] def prop(model, nodes, bias, data):

        node = np.concatenate((bias, data), axis=0)
        nodes[0] = node

        for i, v in enumerate(node_num[1:]):
            node = np.dot(model[i], nodes[i])
            node = sigmoid(node)
            if v == node_num[-1]:
                nodes[i+1] = node
            else:
                nodes[i+1] = np.concatenate((bias, node), axis=0)

        return nodes

    def backprop(model, node, l, size, layer_num, alpha, lamda, theta_num):
        new_model = model

        tmp = (node[layer_num] - l)/size
        new_model[layer_num-1] = (1-alpha*lamda/theta_num)*new_model[layer_num-1] \
                                 - alpha*np.dot(tmp, node[layer_num-1].T)

        for i, v in enumerate(range(layer_num-1, 0, -1)):
            tmp = np.dot(new_model[v].T[1:,:], tmp) * sigmoid_(node[v][1:])
            new_model[v-1] = (1-alpha*lamda/theta_num)*new_model[v-1] \
                             - alpha * np.dot(tmp, node[v-1].T)

        return new_model
```

Optimization

```
[8] # Varibales
    count = 1
    accuracy_history = [[], []]
    j = [0, 0]
    j_old = 10
    j_history = [[], []]
    bias_train = np.array([np.ones(train_m)])
    bias_test = np.array([np.ones(test_m)])
    train_oh_label = one_hot(train_label)
    test_oh_label = one_hot(test_label)

    # Check the runtime
    start = time.time()

    #while (count <= 5000):
    while (abs(j_old - j[0]) > 0.0001):
        # Reset the variables
        accuracy = [0, 0]

        # Update the Loss
        j_old = j[0]

        # Train
        # Forward propagation
        node = prop(network, node, bias_train, train_image)

        # Train Loss
        j[0] = j_func(node, train_oh_label, train_m, layer_num, lamda, theta_num)

        # Train Accuracy
        h_argmax = []
        for i, v in enumerate(node[layer_num].T):
            h_argmax.append(np.argmax(v))
        accuracy[0] = np.sum(1*np.equal(h_argmax, train_label))

        # Test
        # Forward propagation
        node_t = prop(network, node_t, bias_test, test_image)

        # Test Loss
        j[1] = j_func(node_t, test_oh_label, test_m, layer_num, lamda, theta_num)

        # Test Accuracy
        h_t_argmax = []
        for i, v in enumerate(node_t[layer_num].T):
            h_t_argmax.append(np.argmax(v))
        accuracy[1] = np.sum(1*np.equal(h_t_argmax, test_label))

        # Back-propagation
        network = backprop(network, node, train_oh_label, train_m, layer_num, alpha, lamda, theta_num)

        # Record the history
        accuracy_history[0].append((accuracy[0]*100 / len(h_argmax)))
        j_history[0].append(j[0])
        accuracy_history[1].append(accuracy[1]*100 / len(h_t_argmax))
        j_history[1].append(j[1])

        # Count Iteration
        count = count + 1

    runtime = int(time.time() - start)
    minute, second = divmod(runtime, 60)
    hour, minute = divmod(minute, 60)
```

Check the convergence

```
[9] print("Iteration: ", count)
    print("Runtime: ", hour, ":", minute, ":", second)
    print("Loss Diff: ", abs(j[0] - j_old))
```

```
Iteration:  2121
Runtime:  0 : 5 : 48
Loss Diff:  9.997011949922086e-05
```

Check the history for display example

```
[10] classified_ex = []
     misclassified_ex = []
     check_list = np.zeros(10)
     for i, v in enumerate(h_t_argmax):
         if v==test_label[i]:
             if (len(classified_ex)==10):
                 continue
             else:
                 if (check_list[v]==0):
                     classified_ex.append(i)
                     check_list[v] = 1
                 else:
                     continue
         else:
             if (len(misclassified_ex)==10):
                 continue
             else:
                 misclassified_ex.append(i)
         if (len(classified_ex)==10) and (len(misclassified_ex)==10):
             break
```

Result

1. Loss Curve

```
[11] plt.figure(figsize=(15,9))
     plt.plot(j_history[0], color='blue', label='Train Loss')
     plt.plot(j_history[1], color='red', label='Test Loss')
     plt.grid()
     plt.legend()
     plt.title('Loss Curve', fontsize=40)
     plt.xlabel('Iteration', fontsize=20)
     plt.ylabel('Error', fontsize=20)
```

```
Text(0, 0.5, 'Error')
```

2. Accuravy Curve

```
[12] plt.figure(figsize=(15,9))
     plt.plot(accuracy_history[0], color='blue', label='Train Accuracy (%)')
     plt.plot(accuracy_history[1], color='red', label='Test Accuracy (%)')
     plt.grid()
     plt.legend()
     plt.title('Accuracy Curve', fontsize=40)
     plt.xlabel('Iteration', fontsize=20)
     plt.ylabel('Accuracy', fontsize=20)
```

```
Text(0, 0.5, 'Accuracy')
```

3. Accuracy Value

```
print('Final Accuracy')
print('Train: %lf' %((accuracy[0]*100 / len(h_argmax))))
print('Test: %lf' %((accuracy[1]*100 / len(h_t_argmax))))
```

```
Final Accuracy
Train: 99.900000
Test: 88.433333
```

4. Classification Example

1. Correctly Classified Testing Image

```
[14] f1 = plt.figure(figsize=(10,6))
     plt.suptitle("Correctly Classified", fontsize = 40)
     for i in range(10):

         im_vector   = test_image[:, classified_ex[i]]
         im_matrix   = im_vector.reshape((size_row, size_col))

         plt.subplot(2, 5, i+1)
         plt.title(h_t_argmax[classified_ex[i]], fontsize=20)
         plt.imshow(im_matrix, cmap='Greys', interpolation='None')

         frame   = plt.gca()
         frame.axes.get_xaxis().set_visible(False)
         frame.axes.get_yaxis().set_visible(False)

     plt.show()
```

2. Misclassified Testing Image

```
[15] f1 = plt.figure(figsize=(10,6))
     plt.suptitle("Misclassified", fontsize = 40)
     for i in range(10):

         im_vector   = test_image[:, misclassified_ex[i]]
         im_matrix   = im_vector.reshape((size_row, size_col))

         plt.subplot(2, 5, i+1)
         plt.title(h_t_argmax[misclassified_ex[i]], fontsize=20)
         plt.imshow(im_matrix, cmap='Greys', interpolation='None')

         frame   = plt.gca()
         frame.axes.get_xaxis().set_visible(False)
         frame.axes.get_yaxis().set_visible(False)

     plt.show()
```

```
[15]
```