Information

Writer : Junhyeok Woo
Std. ID : 2016-11317
Assignment#8 : Multi - label classification using neural networks
Deadline : May 26, 2020

Library

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

Data

```
[2] file_data = '/content/drive/My Drive/Spring2020/Machine_Learning/Lab/Machine&Deep Learning Lab/Machine&Deep Learning/Assignment#8/train.csv'
    header_file = open(file_data, 'r')
    header = header_file.readline()
    header_file.close()

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

    num_image = 12000(6%)
    nmax      = 2    # output the size number of samples

    #
    # 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)

    #
    # Compute 2D distance between two vectors a and b
    #
    def distance(a, b):
        #
        d = (a - p) ** 2
        d = np.sum(d)
        d = np.sqrt(d)

        return(d)

    #
    # make a matrix each column of which represents an image 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:])
        np_vector  = normalize(im_vector)

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

        count += 1
```

Split data as train one and test one

- Train (5-4000)
- Test (6001 - 10000)

```
[3] train_n = 4000
    test_n = 4000
    train_image = list_image[:, :4000]
    test_image = list_image[:, 6000:]
    train_label = list_label[:4000]
    test_label = list_label[6000:]
```

Neural Network Architecture

- input layer : 784 (= 28*28)
- first hidden layer : 256 (= 16*16)
- second hidden layer : 49 (= 7*7)
- output layer : 10

```
[4] nodes = 5
    sizes = 1
    node_num = [784, 256, 49, 10]

    network = {}
    for idx in range(1, len(node_num)):
        globals.update(locals().update({'idx': idx, 'node_num': node_num[idx], 'node_num1': node_num[idx-1]}))
```

Sigmoid Function (= activation function)

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

```
[5] def sigmoid(x):
        return 1.0 / (1. + np.exp(-x))

    def sigmoid_p(x):
        return (1.0 - x)*x
```

Objective Function

$$J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[-\tilde{y}_k^{(i)}\log(\hat{y}_k^{(i)}) - (1 - \tilde{y}_k^{(i)})\log(1 - \hat{y}_k^{(i)})]$$
$$\Theta = \{\Theta_1, \Theta_2\}$$

```
[6] def cos_dist(x):
        num_image = np.zeros((12,num))
        for i, v in enumerate(x):
            for j, w in enumerate(x):
                num_image[i,j] = distance(v, w)
        return num_image

    def J_func(h, label, size):
        loss_list = (label*np.log(h) - (1 - label)*np.log(1-h))
        loss = np.sum(loss_list)
        return(-1/size*loss)
```

Gradient Descent (Back-propagation)

```
[7] def propagate(w, b, data):
        #
        h_, *p = np.zeros(nodes), data, data[:0]
        for i, w in enumerate(w):
            p = np.dot(w.T(p[-1], h_)
            h = sigmoid(p)
            # Saving data
            h_, *p = np.zeros(nodes[i+1]), p, []
            p = np.dot(w.T(p[-1], h_))
            h = sigmoid(p)
            # Saving data
            h_, *p = np.zeros(nodes[i+2]), p, []
            p = np.dot(w.T(p[-1], h_))
            h = sigmoid(p)
            # Saving data
            h_, *p = np.zeros(nodes[i+3]), p, []
            p = np.dot(w.T(p[-1], h_))
            h = sigmoid(p)
            # Saving data
            return [h, p_, h_, h_, h_]

    def backprop(nodes, node_1, sizes):
        alpha = 0.1
        alpha_sum = 0
        w = node[0]
        w = node[1]
        w = node[2]
        w = node[3]
        w = node[4]

        for i
        # Initial parameters
        node = propagate(w, b, data)

        # Train loss
        J_tr = J_func(node[0], train_y0_label, train_size)

        # Train accuracy
        y_tr = np.zeros(train_size)
        for i, h in enumerate(node[1,4]):
            h_argmax.append(np.argmax(h))
        accuracy_tr = np.mean(np.asarray(h_argmax, train_label))

        # Test
        # Initial propagation
        node_t = propagate(w, b, test_data, test_image)

        # Test loss
        J_te = J_func(node[1], test_y0_label, test_sz)

        # Test accuracy
        y_te = np.zeros(test_sz)
        for i, h in enumerate(node_t[1,4]):
            h_argmax.append(np.argmax(h))
        accuracy_te = np.mean(np.asarray(h_argmax, test_label))

        # Back propagation
        node_t = backprop(w, node, train_y0_label, train_size)

        # Update
        w[1] = w[1]-alpha*(node_t[1], train_y0_label, train_sz)

        # Train accuracy
        y_tr = np.zeros(train_size)
        for i, h in enumerate(node_t[1,4]):
            h_argmax.append(np.argmax(h))
        accuracy_tr = np.mean(np.asarray(h_argmax, train
```