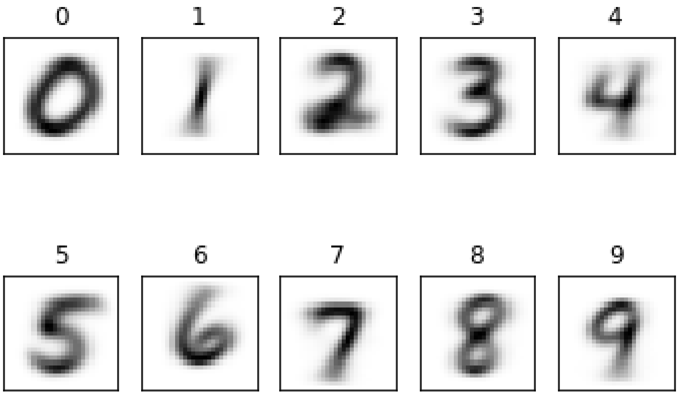
```
!curl https://colab.chainer.org/install | sh -
```

```
import matplotlib.pyplot as plt
import numpy as np
import cupy as cp
from termcolor import colored
file_data   = "drive/My Drive/Classroom/Machine Learning (2) 2020-1\
/class-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

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 = cp.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)
idx_label = [[] for i in range(10)]
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
    idx_label[int(label)].append(count)
    count += 1

f2 = plt.figure(2)
```

```

im_average  = np.zeros((size_row * size_col, 10), dtype=float)
im_count    = np.zeros(10, dtype=int)

for i in range(num_image):

    im_average[:, list_label[i]] += list_image[:, i]
    im_count[list_label[i]] += 1

for i in range(10):

    im_average[:, i] /= im_count[i]

    plt.subplot(2, 5, i+1)
    plt.title(i)
    plt.imshow(im_average[:,i].reshape((size_row, size_col)), cmap='Greys', inte

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

plt.show()
list_image =cp.array(list(list_image))
```



```
'''
Config
'''

learning_rate = 0.02
batch_size=100

one_hot=cp.zeros((10, num_image))
for i in range(num_image) :
    one_hot[list_label[i]][i]=1

train_image = list_image[:, :1000]
train_label = one_hot[:, :1000]

test_image = list_image[:, 1000:]
test_label = one_hot[:, 1000:]

num_train = train_image.shape[1]
num_test = test_image.shape[1]
num_test = test_image.shape[1]


layer1=cp.empty((512, num_train), dtype=float)
layer2=cp.empty((512, num_train), dtype=float)
layer3=cp.empty((512, num_train), dtype=float)
layer4=cp.empty((10, num_train), dtype=float)
layers=[train_image, layer1, layer2, layer3, layer4]


std_w0 = cp.sqrt(2/1296)
weight0=cp.random.normal(0., std_w0, (512, size_row * size_col))
std_w1 = cp.sqrt(2/1024)
weight1=cp.random.normal(0., std_w1, (512, 512))
std_w2 = cp.sqrt(2/1024)
weight2=cp.random.normal(0., std_w2, (512, 512))
std_w3 = cp.sqrt(2/522)
weight3=cp.random.normal(0., std_w2, (10, 512))
weights=[weight0, weight1, weight2, weight3]

# m_t=[np.zeros((196, 784)), np.zeros((49, 196)), np.zeros((10, 49))]
# v_t=[np.zeros((196, 784)), np.zeros((49, 196)), np.zeros((10, 49))]

m_t=[cp.zeros((512, 784)), cp.zeros((512, 512)), cp.zeros((512, 512)), cp.zeros(
v_t=[cp.zeros((512, 784)), cp.zeros((512, 512)), cp.zeros((512, 512)), cp.zeros(

bias0, bias1, bias2, bias3 = 0.1, 0.1, 0.1, 0.1
biases=[bias0, bias1, bias2, bias3]

train_loss=[]
train_accuracy=[]

test_loss=[]
test_accuracy=[]

print(num_train)
print(num_test)
```

```
1000
9000
```

```
do=[0, 0, 0, 1]
p_do=[0, 0, 0, 1]
def sigmoid(x) :
    return 1/(1+cp.exp(-x))

def fw_propagation() :
    #a_list=[]
    #input_data = np.vstack(((np.full((1, list_image.shape[1]), bias0)), list_im
    for i in range(len(weights)-1) :
        #print('a'*50)
        do[i] = cp.random.binomial(1, 0.8, size=(layers[i+1][:,0].shape[0], 1))
        #print('-'*50)
        p_do[i] = float(cp.count_nonzero(do[i])*1000 / (layers[i+1].shape[0] * l
        #print('*'*50)
    # do[0]=np.random.binomial(1, 0.7, size=layers[1].shape)
    # do[1]=np.random.binomial(1, 0.7, size=layers[2].shape)
    # do[2]=np.random.binomial(1, 0.7, size=layers[3].shape)
    for i in range(len(weights)) :
        layers[i+1]=sigmoid(weights[i] @ layers[i]) * do[i]
        #layers[i+1]=sigmoid(weights[i] @ layers[i])

def predict() :
    for i in range(len(weights)) :
        if i==0 :
            output = sigmoid(weights[i] @ test_image) * p_do[i]
        elif i==len(weights)-1 :
            output = sigmoid(weights[i] @ output)
        else :
            output = sigmoid(weights[i] @ output) * p_do[i]
    return output
```

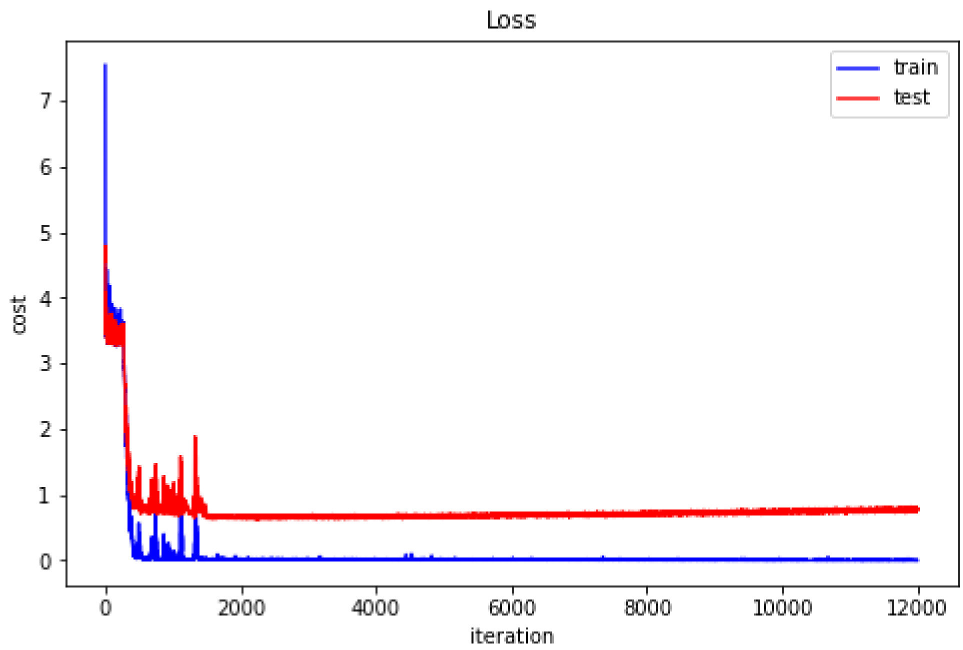
```

'''
Back propagation
'''
lbd = 0.25
beta1, beta2 = 0.9, 0.999
eps=1e-8
t=0
def bw_propagation() :
    global t, m_t, v_t, eps, beta1, beta2
    dev_h=layers[-1] - train_label
    t+=1
    for i in range(len(weights)-1, -1, -1) :
        dev_w=dev_h @ layers[i].T
        if i==0 :
            pass
        else :
            dev_h=weights[i].T @ dev_h * layers[i] * (1-layers[i])

        weights[i] -= learning_rate * dev_w / num_train + learning_rate * lbd *w

def loss(output, label) :
    return cp.mean(cp.sum(-label*cp.log(output) - (1-label)*cp.log(1-output), ax
def accuracy(output, label):
  return (output == label.argmax(axis = 0)).mean()*100
```

```
import time
start = time.time()

for i in range(12000) :
    # input_data, a0, a1, a2 = fw_propagation()
    if i%500==0 :
        print("time :", time.time() - start)
    fw_propagation()
    bw_propagation()
    p = predict()
    train_loss.append(loss(layers[-1], train_label))
    test_loss.append(loss(p, test_label))
    train_accuracy.append(accuracy(layers[-1].argmax(axis=0), train_label))
    test_accuracy.append(accuracy(p.argmax(axis=0), test_label))
    if float(test_accuracy[-1]) > 91 :
        learning_rate=1e-5
print("time :", time.time() - start)
```

1. Plot the loss curve

- train loss는 파란색, test loss는 빨간색으로 plot했습니다.

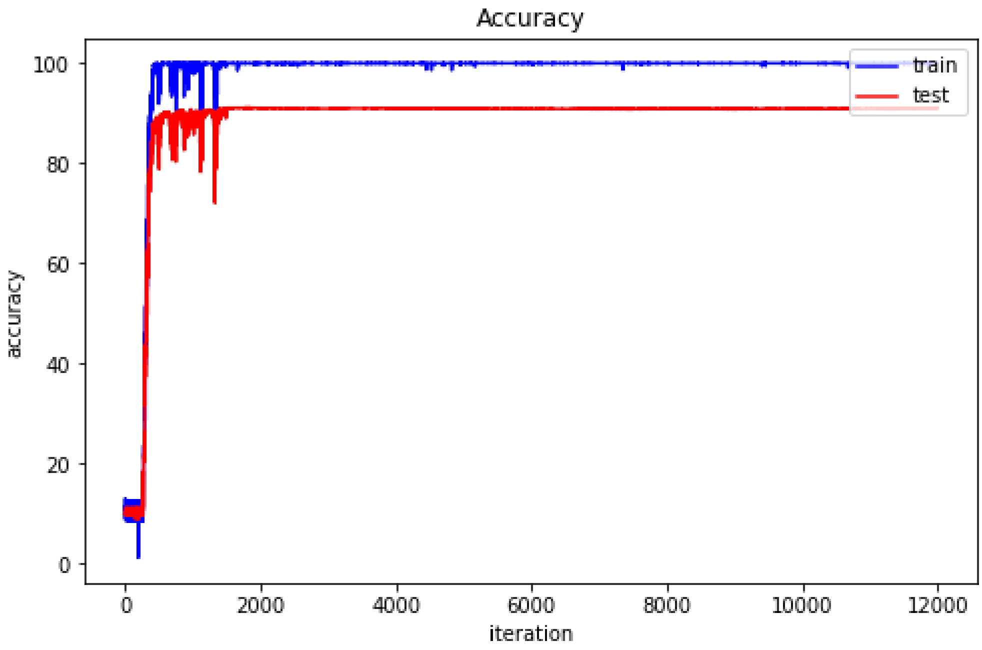
```
'''
Visualize Loss
'''
plt.figure(figsize=(8, 5))
plt.plot(train_loss, color='blue', label='train')
plt.plot(test_loss, color='red', label='test')
plt.xlabel("iteration")
plt.ylabel("cost")
plt.legend(loc='upper right')
plt.title("Loss")
plt.show()
```



2. Plot the accuracy curve

- train accuracy는 파란색, test accuracy는 빨간색으로 plot했습니다.

```
'''
Visualize accuracy
'''
plt.figure(figsize=(8, 5))
plt.plot(train_accuracy, color='blue', label='train')
plt.plot(test_accuracy, color='red', label='test')
plt.xlabel("iteration")
plt.ylabel("accuracy")
plt.legend(loc='upper right')
plt.title("Accuracy")
plt.show()
```



3. Plot the accuracy value

- train accuracy는 파란색, test accuracy는 빨간색으로 print했습니다.

```
print("<Final accuracy>")
print('-'*50)
train_final='train accuracy : '+str(train_accuracy[-1]) + '%'
test_final='test accuracy : '+str(test_accuracy[-1]) + '%'
print(colored(train_final, 'blue'))
print(colored(test_final, 'red'))
print('-'*50)
```

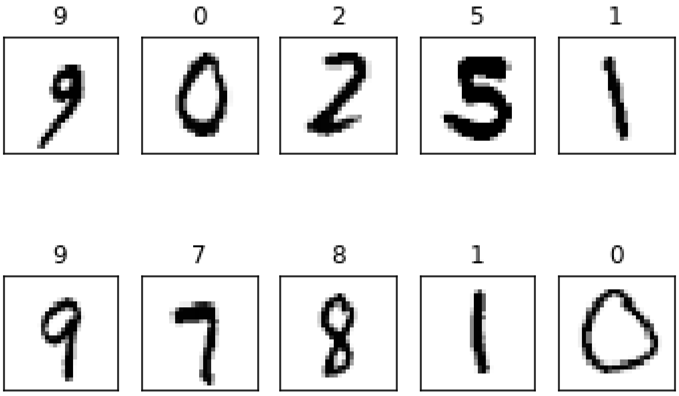
```
<Final accuracy>
--------------------------------------------------
train accuracy : 100.0%
test accuracy : 91.12222222222222%
--------------------------------------------------
```

4. Plot the classification example

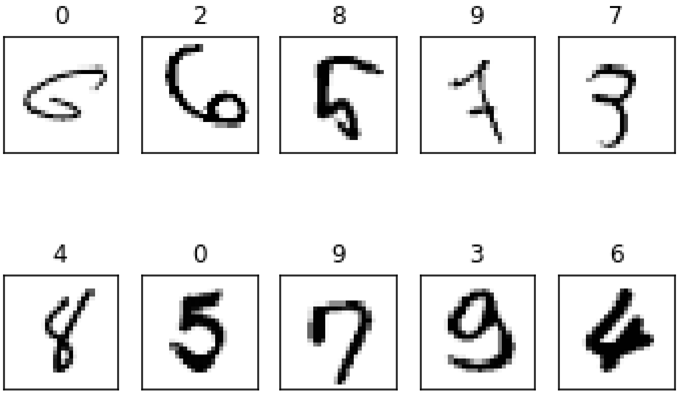
첫번째는 test image에서 답을 맞춘 경우, 두번째는 틀린 경우를 plot했습니다

```
p=cp.asnumpy(p)
test_image=cp.asnumpy(test_image)
#plt.figure(figsize=(15,3))
pred = p.argmax(axis=0)
label = test_label.argmax(axis=0)

count=1
print("<Correct>")
print("-"*35)
for i in range(len(label)) :
    if pred[i] == label[i] :
        plt.subplot(2, 5, count)
        plt.title(pred[i])
        plt.imshow(test_image[:, i].reshape(28, 28), cmap='Greys', interpolation
        frame = plt.gca()
        frame.axes.get_xaxis().set_visible(False)
        frame.axes.get_yaxis().set_visible(False)
        count+=1
    if count >10 :
        break
plt.show()

count=1
print("<Wrong>")
print("-"*35)
for i in range(len(label)) :
    if pred[i] != label[i] :
        plt.subplot(2, 5, count)
        plt.title(pred[i])
        plt.imshow(test_image[:, i].reshape(28, 28), cmap='Greys', interpolation
        frame = plt.gca()
        frame.axes.get_xaxis().set_visible(False)
        frame.axes.get_yaxis().set_visible(False)
        count+=1
    if count >10 :
        break
plt.show()
```

```
<Correct>
-----------------------------------
```


9
0
2
5
1
9
7
8
1
0

```
<Wrong>
-----------------------------------
```


0
2
8
9
7
4
0
9
3
6