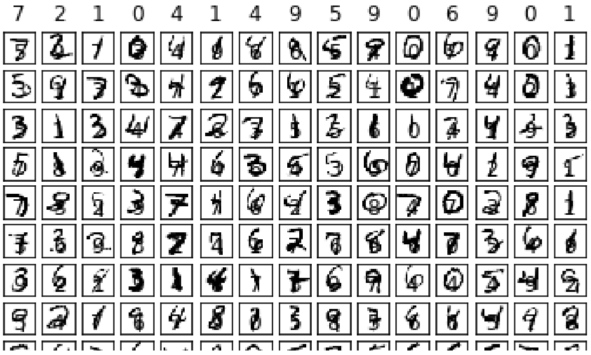
```
cd '/content/drive/My Drive/machine_learning_cl/Multi-label_classify_nn_regularization'
```

```
/content/drive/My Drive/machine_learning_cl/Multi-label_classify_nn_regularization
```

Input Data

```
import matplotlib.pyplot as plt
import numpy as np

file_data   = "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

#
```

```
# plot first 150 images out of 10,000 with their labels
#
f1 = plt.figure(1)

for i in range(150):

    label       = list_label[i]
    im_vector   = list_image[:, i]
    im_matrix   = im_vector.reshape((size_row, size_col))

    plt.subplot(10, 15, i+1)
    plt.title(label)
    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()

#
# plot the average image of all the images for each digit
#
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', interpolation='None')

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

plt.show()
```



Split data to (train images, train labels) and (test images, test labels)

```
numtrainimages = 1000
numtestimages = 9000
train_images  = np.empty((size_row * size_col, numtrainimages), dtype=float)
train_label  = np.empty(numtrainimages, dtype=int)
test_images  = np.empty((size_row * size_col, numtestimages), dtype=float)
test_label  = np.empty(numtestimages, dtype=int)

train_images = list_image[:, :numtrainimages]
train_label = list_label[:numtrainimages]
test_images = list_image[:, numtrainimages:]
test_label  = list_label[numtrainimages:]
```

Neural Network Architecture

Object function

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

Gradient Descent

$$\theta_k^{t+1} := \Theta_k^t(1-\alpha\frac{\lambda}{m}) - \alpha\frac{\partial J(\theta^{(t)})}{\partial\theta_k}\ for\ all\ k$$

$$\frac{\partial\delta}{\partial W_{a,k}} = (\sigma(h_k) - l) * \tilde{z_a}$$

$$a = 1,,,49\ k = 1,,,10$$

$$\frac{\partial\delta}{\partial V_{b,k}} = (\sigma(h) - l) * W_{b,k} * (\sigma(z_k)(1-\sigma(z_k)) * \tilde{y_b}$$

$$b = 1,,,196\ k = 1,,,49$$

$$\frac{\partial\delta}{\partial U_k} = (\sigma(h) - l) * W_1 * (\sigma(z_1)(1-\sigma(z_1)) * V_1 * (\sigma(y_1))(1-\sigma(y_1)) *$$

$$c = 1,,,784\ k = 1,,,196$$

```
# create label for one hot encoding
def label(num):
  label = np.zeros(10, dtype=int)
  label[num] = 1
  return label

def labels():
  labels = np.zeros((10,10), dtype = int)
  for i in range(10):
    labels[i, :] = label(i)
  return labels

l = labels() #10X10
```

```
## NN architecture
numX = 784
numY = 196
numZ = 49
numH = 10
```

```
import pickle

#가장 처음 한 번만 실행
# initializing model parameters

"""
np.random.seed(seed=100) #랜덤값 고정
U = np.random.normal(size=numY*(numX+1)).reshape(numX+1, numY)
V = np.random.normal(size=numZ*(numY+1)).reshape(numY+1, numZ)
W = np.random.normal(size=numH*(numZ+1)).reshape(numZ+1, numH)
past_itr = 0

accuracy_array = np.zeros(past_itr)
cost_array = np.zeros(past_itr)

with open('practice_train_weights.p', 'wb') as file:
    pickle.dump(U, file)
    pickle.dump(V, file)
    pickle.dump(W, file)
    pickle.dump(past_itr, file)
    pickle.dump(accuracy_array, file)
    pickle.dump(cost_array, file)
 """
```

```
"₩nnp.random.seed(seed=100) #랜덤값 고정₩nU = np.random.normal(size=numY*(numX+1)).reshape(numX+1, numY)₩nV =
```

function for using gpu

```
#이미지 1개
from numba import jit
@jit(nopython=True, parallel=True)
def J(H, label): # 10*1 10*1
  cost = (-label)*(np.log(H))
  - (1-label)*(np.log(1-H))
  SUM = np.sum(cost)
  return SUM
```

```
from numba import jit
@jit(nopython=True)
def zeros(length):
    return np.zeros(length, np.float64)  # np.float64 instead of np.float

@jit(nopython=True)
def empty(size):
def empty(size):
    return np.empty(size, np.float64)

@jit(nopython=True, parallel=True)
def sigma(X):
    return 1.0/(1.0+(np.exp(-X)))

@jit(nopython=True, parallel=True)
def sum_square(X):
    return np.square(X)


@jit(nopython=True, parallel=True)
def sum_square(X):
  acc = 0
  for val in X:
    val2 = np.square(val)
    acc += np.sum(val2)
  return acc


@jit(nopython=True)
def model(U, V, W, past_itr, numimages, accuracy_array_t, cost_array_t,image_vec, image_label):
  #gradient descent
  predicted_label = zeros(numimages)
  itr = 120
  lamb = 5

  ##코드를 여러번 돌릴 때마다 accuracy array와 cost array 길이 늘리기
  accuracy_array = zeros(past_itr+itr)
  accuracy_array[:past_itr] = accuracy_array_t[:past_itr]
  cost_array = zeros(past_itr+itr)
  cost_array[:past_itr] = cost_array_t[:past_itr]
  ln_rate = 0.002

  for j in range(itr):#iteration
    j = past_itr + j  #cost와 accuracy array의 index
    errors = 0
    ifzero = 1
    if (j==0):
      ifzero = 0
    if (j%30==0):
      ln_rate = ln_rate * (0.002)
    print("iteration:", j+1)
    for i in range(numimages): #6000개의 데이터  # image n개 #forward propagation
      #print(i)
      X  = empty((size_row * size_col, 1))
      im_vec = np.ascontiguousarray(image_vec[:, i])
      X = im_vec.reshape(((size_row *  size_col), 1) )
      l_indx = image_label[i] #int
      Xt_nob = np.transpose(X)

      #  print("Xt_nob : ", Xt_nob)
      X0 = np.array([1]) #bias
      X_b = np.append(im_vec, X0)
      Xt = X_b.reshape(1,size_row *  size_col+1)
      #  print("Xt : ", Xt)
      Y = np.dot(Xt, U)

      #activation function
      ##sigma = lambda x: 1/(1+(np.exp(-x)))
      ##vsigma = np.vectorize(sigma)

      Y_tilda_nob = sigma(Y)
      #  print("Y_tilda_nob : ", Y_tilda_nob)
```

```
#  print("Y_tilda_nob : ", Y_tilda_nob)
YO = np.array([1])
im_vec_y = np.ascontiguousarray(Y_tilda_nob[0, :])
Y_tilda_b = np.append(im_vec_y, YO)
Y_tilda = Y_tilda_b.reshape(1,numY+1)
#  print("Y_tilda : ", Y_tilda)

Z = np.dot(Y_tilda, V)
Z_tilda_nob = sigma(Z)
#  print("Z_tilda_nob : ", Z_tilda_nob)

ZO = np.array([1])
im_vec_z = np.ascontiguousarray(Z_tilda_nob[0, :])
Z_tilda_b = np.append(im_vec_z, ZO)
Z_tilda = Z_tilda_b.reshape(1,numZ+1)
#  print("Z_tilda : ", Z_tilda)
H = np.dot(Z_tilda, W)
H_tilda = sigma(H)
#  print("H_tilda : ", H_tilda)

onehotencd = l[l_indx]

##error
error = J(H_tilda, onehotencd)
predicted_label[i] = np.argmax(H_tilda)
errors += error
#print(" error : ", error)
#print(" predicted_label[i]", predicted_label[i])
#print(" train_label", train_label[i])

##back propagation

## W - 업데이트 49 X 10 - no bias 50*10 #Ztilda는 49개의 w 업데이트의 과정에서 고정
Z_tilda_nob = Z_tilda_nob.reshape(49,1)
redundant = (H_tilda - onehotencd).reshape(1,10 ) #1x10
if(j== 0):
  W[:49, :] = W[:49, :]- ln_rate * np.dot(Z_tilda_nob, redundant)
W[:49, :] = W[:49, :]*(1-ln_rate*lamb/numimages)- ln_rate * np.dot(Z_tilda_nob, redundant) # 49x10
#print(W.shape)

## V - 업데이트 196X49
z = sigma(Z)*(1-sigma(Z))
Y_tilda_nob = Y_tilda_nob.reshape(196,1)
w = np.ascontiguousarray(W[:49, :])
a = np.dot(redundant, w.reshape(10,49))
b = z
redundant2 = (a*b)
if(j== 0):
  V[:196, :] = (V[:196, :] - ln_rate * np.dot(Y_tilda_nob, redundant2))
V[:196, :] = (V[:196, :]*(1-ln_rate*lamb/numimages) - ln_rate * np.dot(Y_tilda_nob, redundant2))
#print(V.shape)

## U - 업데이트 784X49
y = sigma(Y)*(1-sigma(Y))
Xt_nob = Xt_nob.reshape(784,1)
v =  np.ascontiguousarray(V[:196, :])
c = np.dot(redundant2, v.reshape(49,196))
d = y
redundant3 = (c*d)          #1x196
if(j== 0):
  U[:784, :] = (U[:784, :] - ln_rate * np.dot(Xt_nob,redundant3))
U[:784, :] = (U[:784, :]*(1-ln_rate*lamb/numimages) - ln_rate * np.dot(Xt_nob,redundant3))
#print("U : ", U.shape)
```

```
    cost = errors/numimages + ifzero * (lamb/(2*numimages))*sum_square((im_vec,im_vec_y,im_vec_z))
    ## accuracy
    ## check if the prediction is correct
    count = 0
    for real, hypo in zip(image_label, predicted_label):
        if real == hypo:
          count = count + 1

    accuracy = count/numimages

    print("cost ", cost)
    print("accuracy ", accuracy)
    accuracy_array[j] = accuracy

    cost_array[j] = cost

  past_itr = j + 1
  return U,V,W,past_itr,accuracy_array,cost_array,predicted_label
```

```
import pickle
with open('practice_train_weights.p', 'rb') as file:    # weights.p 파일을 바이너리 읽기 모드(rb)로 열기
    U = pickle.load(file)
    V = pickle.load(file)
    W = pickle.load(file)
    past_itr = pickle.load(file)
    accuracy_array_t = pickle.load(file)
    cost_array_t = pickle.load(file)
```

Learning training data

```
U,V,W,training_itr,accuracy_array,cost_array,predicted_label_train=model(U, V, W, past_itr, numtrainimages, accurac
```

```
with open('practice_train_weights.p', 'wb') as file:    # james.p 파일을 바이너리 쓰기 모드(wb)로 열기
  pickle.dump(U, file)
  pickle.dump(V, file)
  pickle.dump(W, file)
  pickle.dump(past_itr, file)
  pickle.dump(accuracy_array,file)
  pickle.dump(cost_array,file)
```

```
print(training_itr)
```

```
250
```

Learn test data

```
##처음 test data learning 시작할때
"""
import pickle
## train한 weight를 받아옴
with open('practice_train_weights.p', 'rb') as file:    # weights.p 파일을 바이너리 읽기 모드(rb)로 열기
    U = pickle.load(file)
    V = pickle.load(file)
    W = pickle.load(file)
    past_itr = pickle.load(file)
    accuracy_array_t = pickle.load(file)
    cost_array_t = pickle.load(file)
"""
```

```
"""
training_itr = past_itr
accuracy_array = accuracy_array_t
cost_array = cost_array_t
"""
```

```
import pickle
with open('testing_weights.p', 'rb') as file:    # weights.p 파일을 바이너리 읽기 모드(rb)로 열기
    U = pickle.load(file)
    V = pickle.load(file)
    W = pickle.load(file)
    past_itr = pickle.load(file)
    accuracy_array_t = pickle.load(file)
    cost_array_t = pickle.load(file)
```

```
past_itr
```

```
370
```

```
U_t,V_t,W_t,testing_itr,accuracy_array_test,cost_array_test,predicted_label=model(U, V, W, past_itr, numtestimages,
```

```
iteration: 371
cost  0.3069373024529752
accuracy  0.9608888888888889
iteration: 372
cost  0.30777716643859515
accuracy  0.9616666666666667
iteration: 373
```

```
with open('testing_weights.p', 'wb') as file:    # james.p 파일을 바이너리 쓰기 모드(wb)로 열기
    pickle.dump(U, file)
    pickle.dump(V, file)
    pickle.dump(W, file)
    pickle.dump(testing_itr, file)
    pickle.dump(accuracy_array_test,file)
    pickle.dump(cost_array_test,file)
```

```
###training 저장 정보 불러오기
import pickle
with open('practice_train_weights.p', 'rb') as file:    # weights.p 파일을 바이너리 읽기 모드(rb)로 열기
    U = pickle.load(file)
    V = pickle.load(file)
    W = pickle.load(file)
    training_itr = pickle.load(file)
    accuracy_array = pickle.load(file)
    cost_array_t = pickle.load(file)
```

```
import pickle
with open('testing_weights.p', 'rb') as file:    # weights.p 파일을 바이너리 읽기 모드(rb)로 열기
    U = pickle.load(file)
    V = pickle.load(file)
    W = pickle.load(file)
    testing_itr = pickle.load(file)
    accuracy_array_test = pickle.load(file)
    cost_array_test = pickle.load(file)
```

The classification example

```
for x in test_label:
  print(x, end =' ')
```

```
9 0 2 5 1 9 7 8 1 0 4 1 7 9 6 4 2 6 8 1 3 7 5 4 4 1 8 1 3 8 1 2 5 8 0 6 2 1 1 7 1 5 3 4 6 9 5 0 9 2 2 4 8 2 1
```

```
for x in predicted_label[:9000]:
  print(x, end =' ')
```

```
9.0 0.0 2.0 3.0 1.0 9.0 7.0 8.0 1.0 0.0 4.0 1.0 9.0 9.0 5.0 4.0 2.0 6.0 8.0 1.0 3.0 7.0 5.0 4.0 4.0 1.0 8.0 1
```

```
correct_vector = np.zeros((size_row*size_col,10))
wrong_vector  = np.zeros((size_row*size_col,10))

indx = 0 #loop variable
ccount = 0
wcount = 0
c_label = np.zeros(10)
w_label = np.zeros(10)

for real, hypo in zip(test_label, predicted_label[:9000]):
  if ccount < 10 :
    if real == hypo:
      correct_vector[:,ccount] = test_images[:, indx]
      c_label[ccount] = predicted_label[indx]
      ccount += 1
  if wcount < 10:
    if real != hypo:
      wrong_vector[:,wcount] = test_images[:, indx]
      w_label[wcount] = predicted_label[indx]
      wcount += 1
  else: break
  indx = indx+1
```

```
for x in test_label:
  print(x, end = ' ')
```

```
9 0 2 5 1 9 7 8 1 0 4 1 7 9 6 4 2 6 8 1 3 7 5 4 4 1 8 1 3 8 1 2 5 8 0 6 2 1 1 7 1 5 3 4 6 9 5 0 9 2 2 4 8 2 1
```

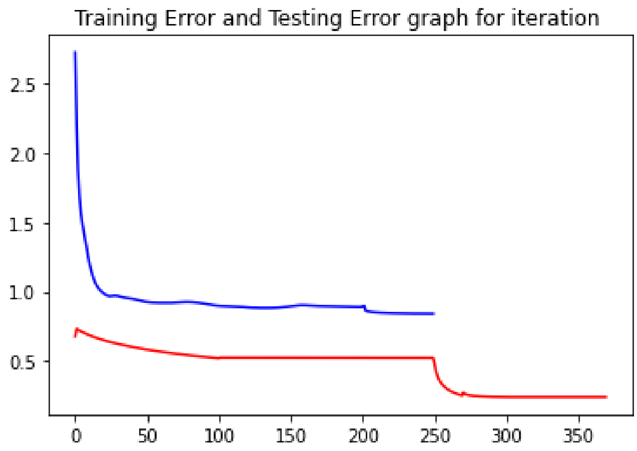
```
for x in predicted_label[:9000]:
  print(int(x), end = ' ')
```

```
9 0 2 3 1 9 7 8 1 0 4 1 9 9 5 4 2 6 8 1 3 7 5 4 4 1 8 1 3 8 1 2 5 8 0 6 2 1 1 9 1 5 3 4 8 9 5 0 9 2 5 4 8 2 1
```

Result

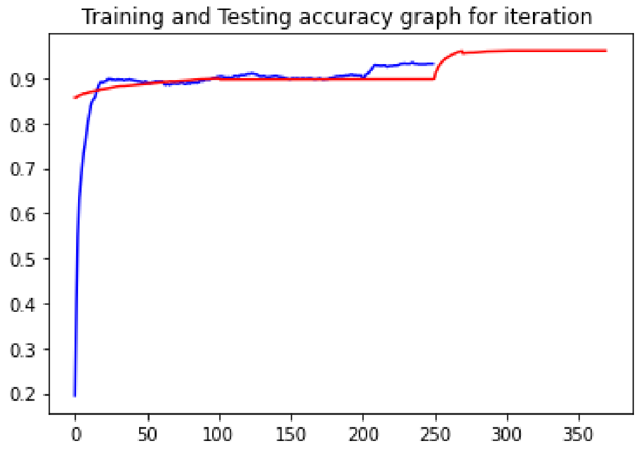
Plot the training error & testing error

```
#J(th0, th1, th2, th3, th4, x, y)
plt.title("Training Error and Testing Error graph for iteration ")
plt.plot(range(training_itr), np.array(cost_array), color = 'blue' )
plt.plot(range(testing_itr), np.array(cost_array_test), color = 'red' )

plt.show()
```



Plot the training & testing accuracy

```
plt.title("Training and Testing accuracy graph for iteration ")
plt.plot(range(training_itr), np.array(accuracy_array), color = 'blue' )
plt.plot(range(testing_itr), np.array(accuracy_array_test), color = 'red' )

plt.show()
```



write down the final training accuracy

```
print("final training accurary : {}%".format(round(accuracy_array[training_itr-1], 3)*100))
```

```
final training accurary : 93.2%
```

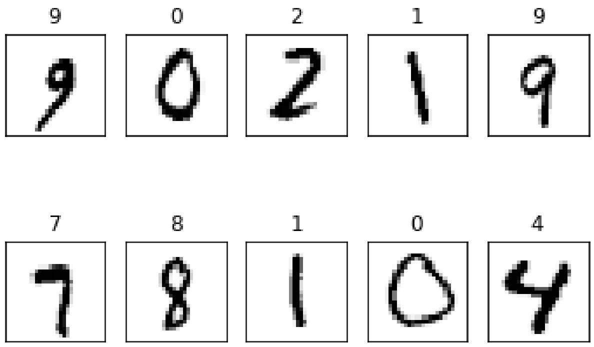
write down the final testing accuracy

```
print("final testing accurary : {}%".format(round(accuracy_array_test[testing_itr-1], 3)*100))
```

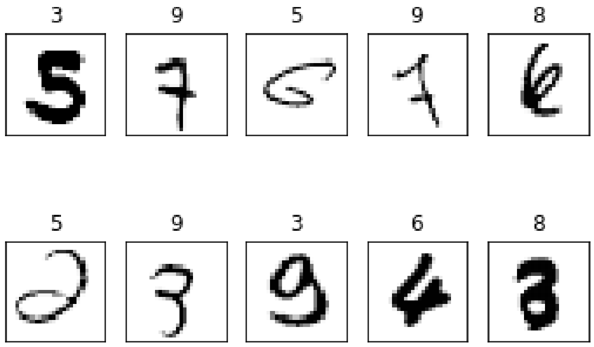
```
final testing accurary : 96.1%
```

correct images

```
f3 = plt.figure(1)
for i in range(10):
  correct_matrix = correct_vector[:, i].reshape(size_row, size_col)
  plt.subplot(2, 5, i+1) # subplot(nrows, ncols, index, **kwargs)
  plt.title(int(c_label[i]))
  plt.imshow(correct_matrix, cmap='Greys', interpolation='None')
  frame   = plt.gca()
  frame.axes.get_xaxis().set_visible(False)
  frame.axes.get_yaxis().set_visible(False)
```



```
f4 = plt.figure(1)
for i in range(10):
  wrong_matrix = wrong_vector[:, i].reshape(size_row, size_col)
  plt.subplot(2, 5, i+1) # subplot(nrows, ncols, index, **kwargs)
  plt.title(int(w_label[i]))
  plt.imshow(wrong_matrix, cmap='Greys', interpolation='None')
  frame   = plt.gca()
  frame.axes.get_xaxis().set_visible(False)
  frame.axes.get_yaxis().set_visible(False)
```


3
9
5
9
8
5
9
3
6
8