Day 4

Recognizing hand-written digits using Neural Network

【review】Outline of Learning Neural Network

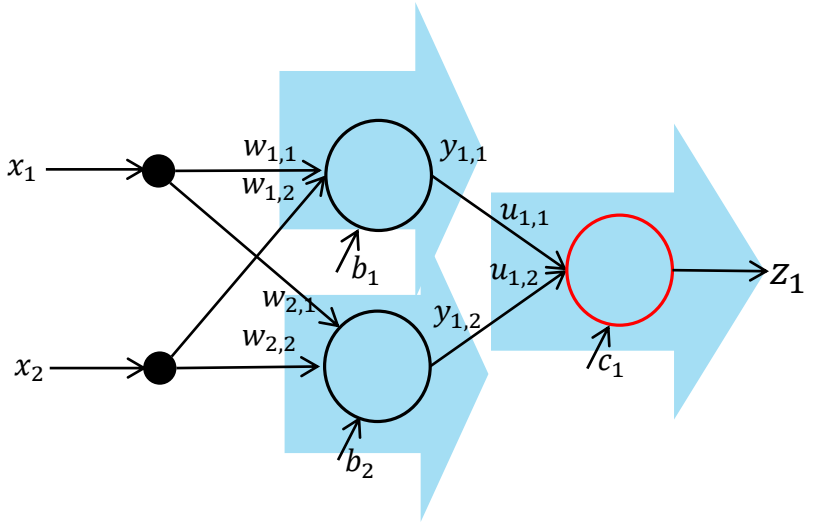
Feedforward calculation

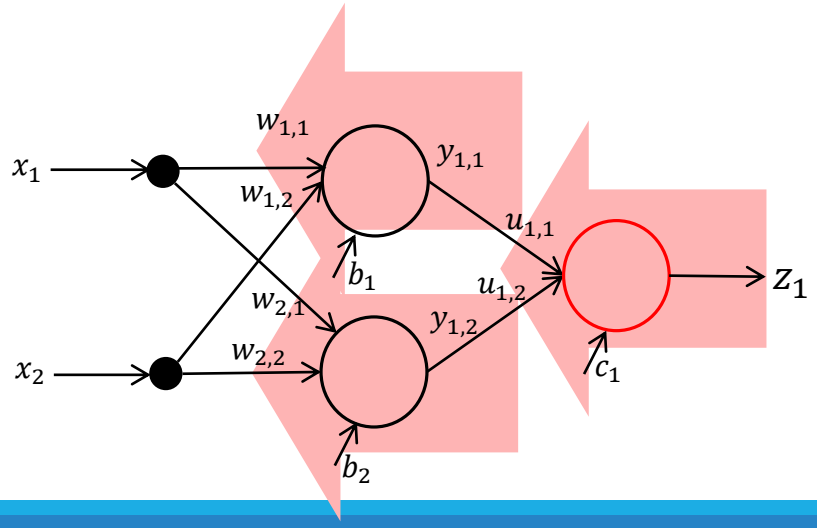
Calculation for outputs from inputs

LOSS Calculation

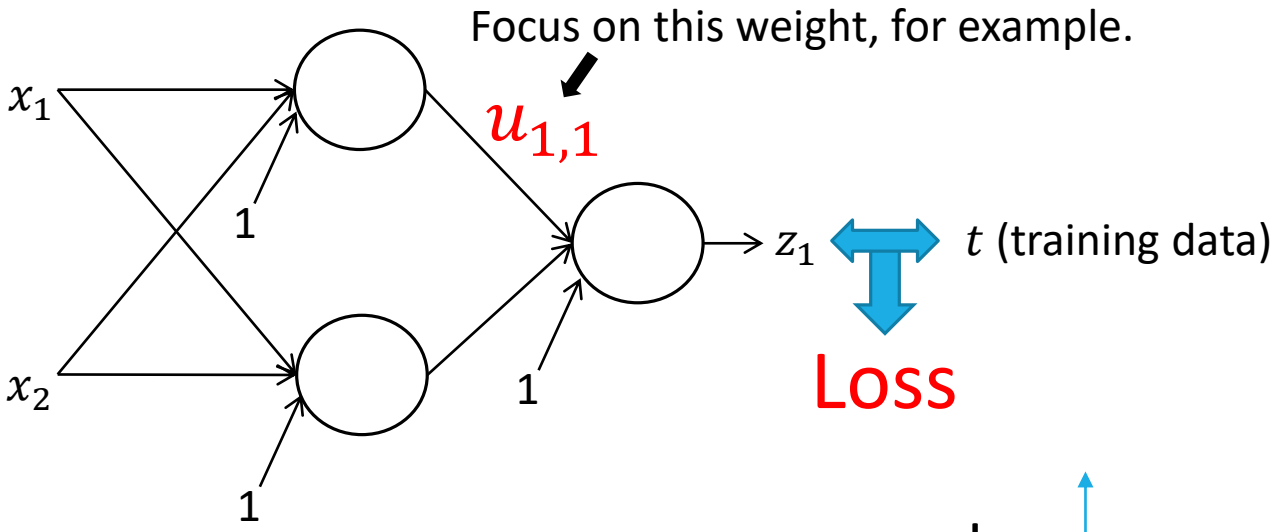
backwards propagation of LOSS

Updating weights and biases to reduce LOSS



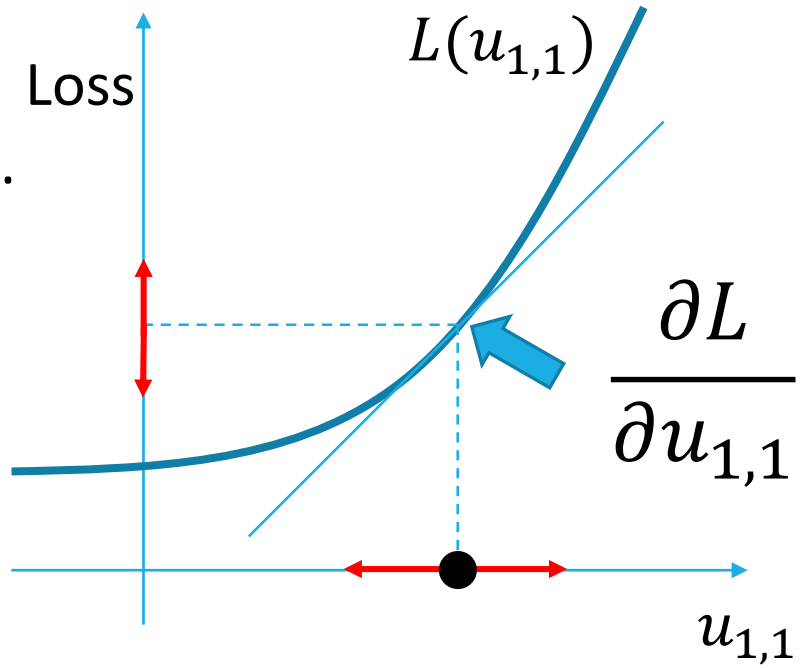


【Review】How can we reduce the LOSS (1)

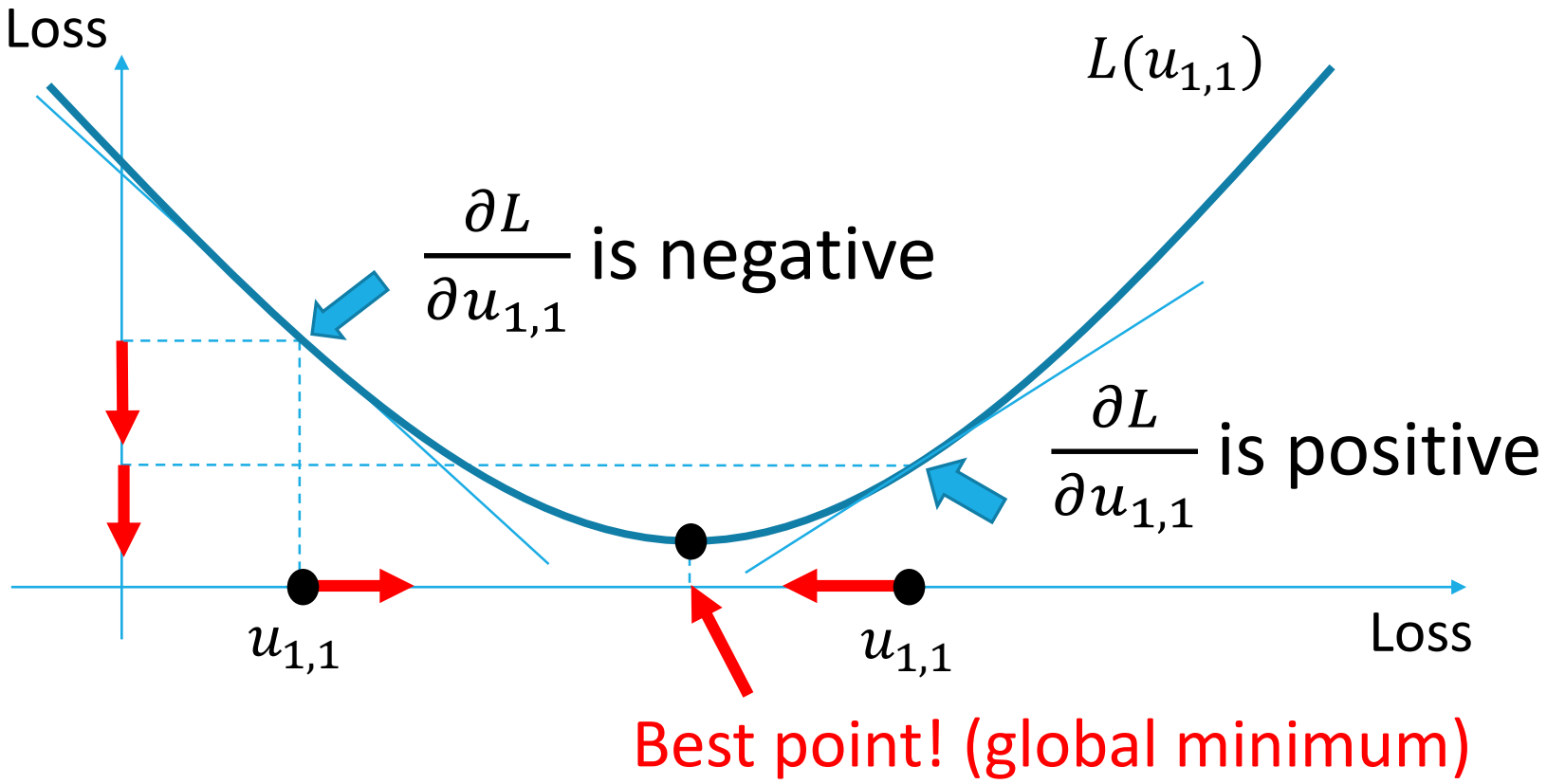


We can calculate slope of Loss function.

The slope is calculated by partial derivative of Loss function with respect to $u_{1,1}$ i.e., $\frac{\partial L}{\partial u_{1,1}}$.



【Review】How can we reduce the LOSS (2)



If $\frac{\partial L}{\partial u_{1,1}}$ is a positive value, we should reduce $u_{1,1}$ to reduce LOSS.

If $\frac{\partial L}{\partial u_{1,1}}$ is a negative value, we should increase $u_{1,1}$ to reduce LOSS.

【Review】Partial Derivative of Loss Function (2)

We can describe $$\frac{\partial L}{\partial u_{k,j}} = \frac{\partial L}{\partial z_k} \cdot \frac{\partial z_k}{\partial q_k} \cdot \frac{\partial q_k}{\partial u_{k,j}}$$

Summing junction

$$q_k = f(u_{k,j}) = \sum_k u_{k,j} y_j + c_k$$

q_k is a function of $u_{k,j}$

Sigmoid function (Activation function)

$$z_k = \sigma(q_k) = \frac{1}{1 + e^{-q_k}}$$

z is a function of q

MSE (Loss Function)

$$L = \frac{1}{2} \sum_k (z_k - t_k)^2$$

L is a function of z_k

【Review】How can we reduce the LOSS (3)

Therefore, we can decline LOSS where $u_{1,1}$ is updated as follows.

Update function for weights

$$u_{1,1} \leftarrow u_{1,1} - \lambda \frac{\partial L}{\partial u_{1,1}}$$

We call λ learning rate

As a calculation result, we can get $\frac{\partial L}{\partial u_{k,j}}$ as follows.

$$\frac{\partial L}{\partial u_{k,j}} = (z_k - t_k)(z_k(1 - z_k))y_j$$

$$= \delta^{out} y_j \quad \text{where, } \delta^{out} = (z_k - t_k)(z_k(1 - z_k))$$

【Review】Partial Differential of Loss Function (8)

Reviewing the calculation process of output z_t

Output Layer

$$L = \frac{1}{2}\sum_k (z_k - t_k)^2$$

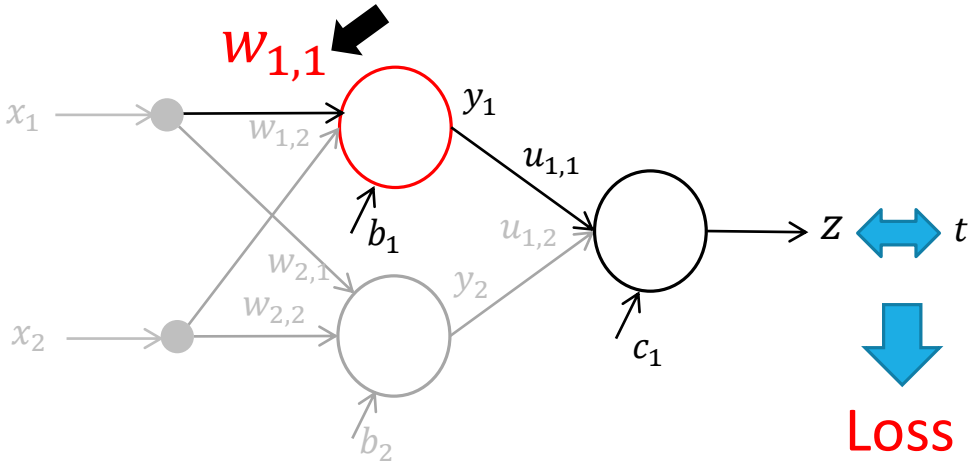
$$z_k = \sigma(q_k) = \frac{1}{1 + e^{-q_k}}$$

$$q_k = \sum_k u_{k,j} y_j + c_k$$

$$y_j = \sigma(p_j) = \frac{1}{1 + e^{-p_j}}$$

$$p_j = \sum_j w_{j,i} x_i + b_j$$

Input Layer



$$\frac{\partial L}{\partial w_{j,i}} = \sum_k \frac{\partial L}{\partial z_k}\frac{\partial z_k}{\partial q_k}\frac{\partial q_k}{\partial y_j}\frac{\partial y_j}{\partial p_j}\frac{\partial p_j}{\partial w_{j,i}}$$

【Review】Update functions for weights and biases

Each update functions for weights and biases is obtained as follows.

$$u_{k,j} \leftarrow u_{k,j} - \lambda \delta_k^{out} y_j$$

$$c_k \leftarrow c_k - \lambda \delta_k^{out}$$

$$w_{j,i} \leftarrow w_{j,i} - \lambda \delta_j^{hidden} x_i$$

$$b_j \leftarrow b_j - \lambda \delta_j^{hidden}$$

where,

$$\delta_k^{out} = (z_k - t_k)(z_k(1 - z_k))$$

$$\delta_j^{hidden} = \sum_k \left(\delta_k^{out} u_{k,j}\right)\left(y_j(1 - y_j)\right)$$

Implement as MATLAB code

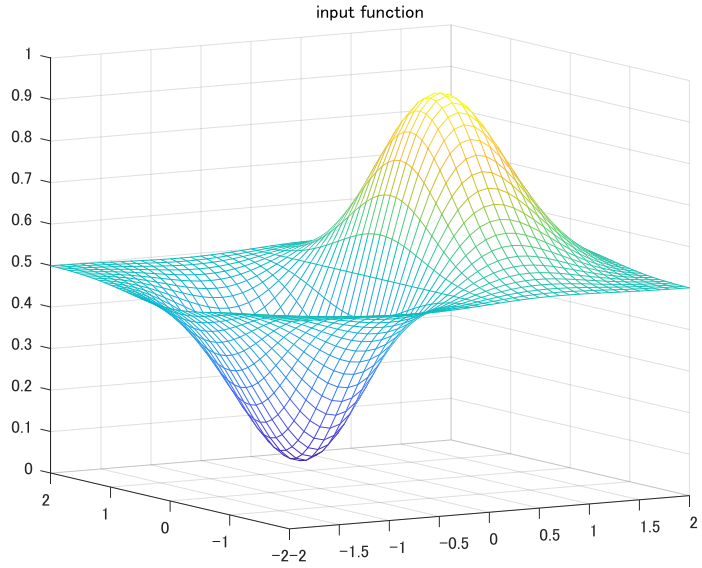
```
delta_out = (z-t).*(z.*(1-z));
delta_hidden = (delta_out'*u)'.*(y.*(1-y));

u = u - LAMBDA * delta_out * y';
c = c - LAMBDA * delta_out;
w = w - LAMBDA * delta_hidden * x';
b = b - LAMBDA * delta_hidden;
```

【Review】Exercise 3.3

Construct neural network with 2 input (x_1, x_2), 3 hidden, 1 output neurons. Then, learning neural network for following function.

$$z = x_1 e^{x_1^2 - x_2^2} + \frac{1}{2}$$

Change a number of hidden neuron, learning rate and a number of epoch and consider about the output results.

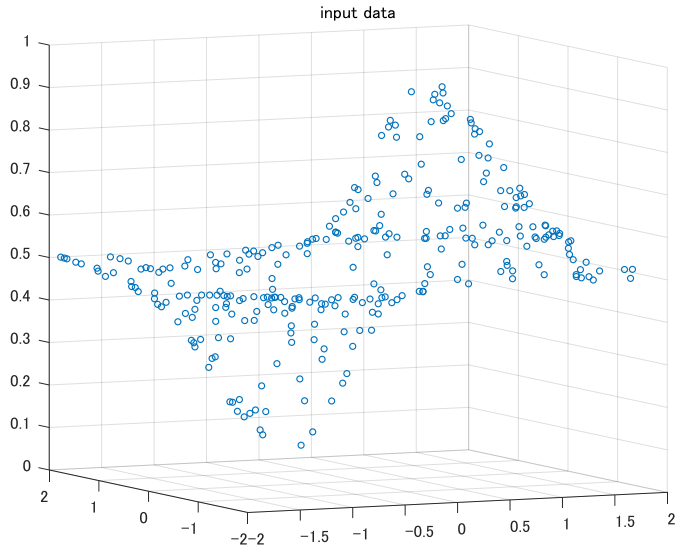


Tips for exercise3_3.m

```
%data generation
data_num=300;
xdata = 4*rand(2,data_num)-2;
labels = xdata(1,:).*exp(-xdata(1,:).^2 - xdata(2,:).^2) + 0.5;
```

【Review】Tips for exercise3_3.m

```
%display input data
figure(1);
scatter3(xdata(1,:), xdata(2,:), labels, 10);
title('input data');
```




```
% Display output graph
[X1 X2] = meshgrid(-2:0.1:2);
y = sigmoid_neuron([X1(:)';X2(:)'],w,b);
z = sigmoid_neuron(y,u,c);
figure(4);
zsize = sqrt(size(z));
mesh(X1,X2,reshape(z,[zsize(2),zsize(2)]));
title('learning results')
```

Learning result

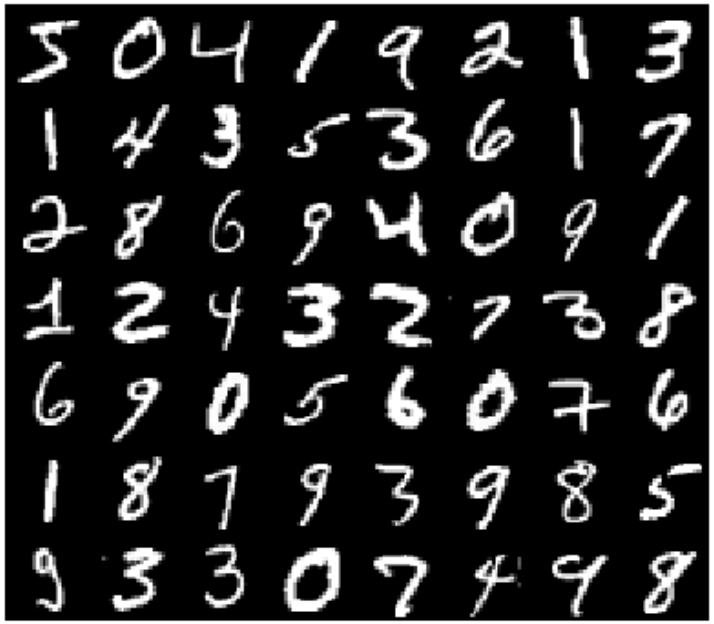
?

Recognizing Hand-Written Digits Using Neural Network

MNIST Dataset

http://yann.lecun.com/exdb/mnist/

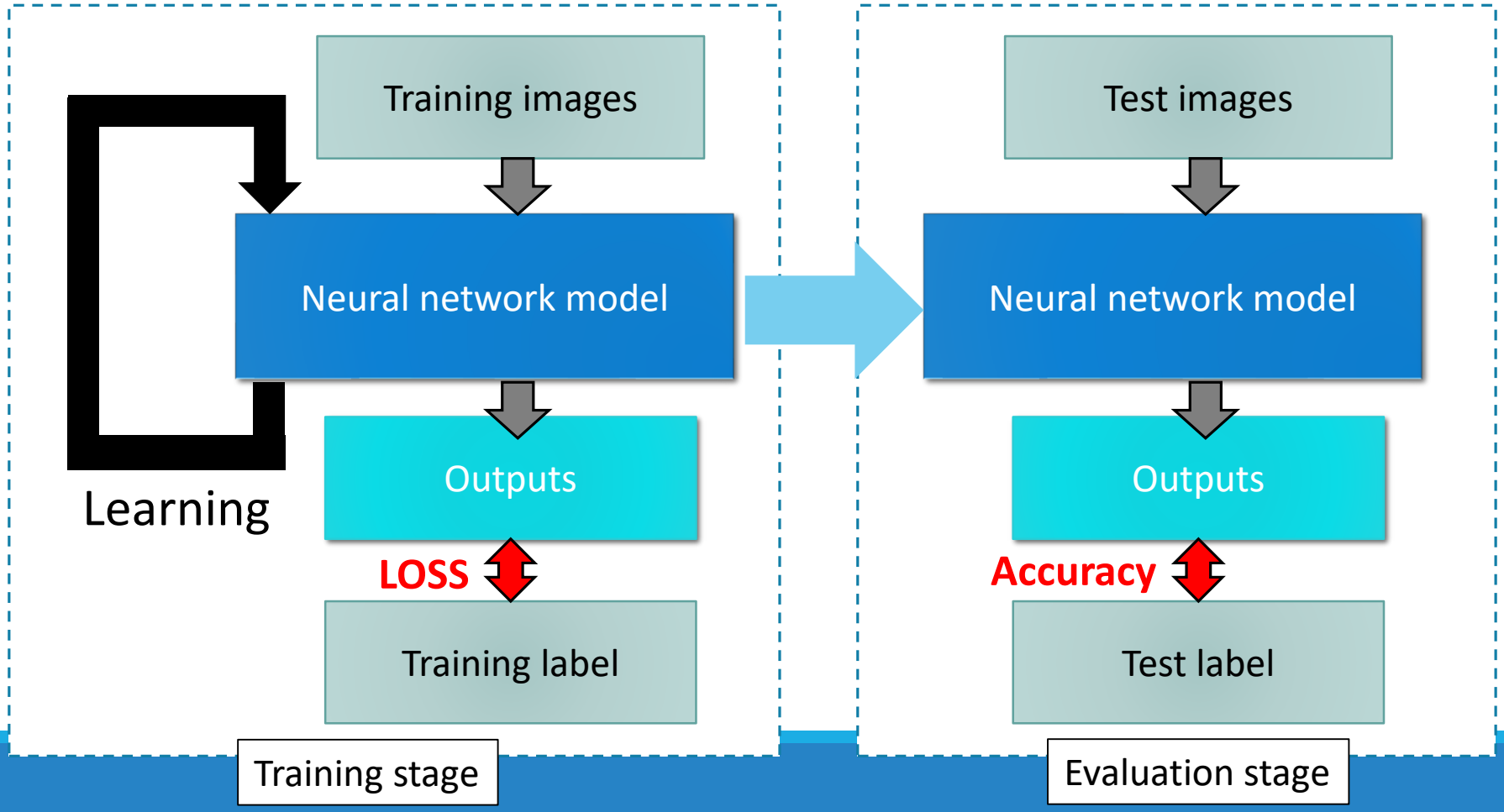
- The MNIST dataset of handwritten digits has a training set of 60,000 examples and a test set of 10,000 examples.
- MNIST dataset is a subset of a larger set available from NIST.
- Each image is a normalized grayscale 28x28 pixel image.
- Each pixel is represented by an integer between 0 (black) and 255(white).



Training Dataset and Test Dataset

We use training dataset to learn neural network parameters, i,.e., minimize LOSS with the training dataset.

After finish learning, we use test dataset to calculate the accuracy of obtained neural network model with optimized weights and biases.



Download MNIST Dataset

Put “downloadDataset.m” in your MATLAB work folder and run the script.

If it succeeds, four files as follows will be created in “data/mnist/ “ folder.

Filename	Description
t10k-images-idx3-ubyte	Test images
t10k-labels-idx1-ubyte	Test labels
train-images-idx3-ubyte	Training images
train-labels-idx1-ubyte	Training labels

These files are binary files for MATLAB, so we cannot read them as text files.

Load MNIST Dataset

Put load_dataset.m in your MATLAB work folder and run the script as follows.

```
>> mnist = load_dataset();
Loading data/mnist/train-labels-idx1-ubyte
Loading data/mnist/train-images-idx3-ubyte
Loading data/mnist/t10k-labels-idx1-ubyte
Loading data/mnist/t10k-images-idx3-ubyte
```

Then, the dataset is loaded to the variable “mnist”. You can see a sample data as follows.

```
>> mnist.train_images(:,:,1,1)
```

“mnist” is a 28 x 28 x 1 x 60000 (4 dimensional array) variable

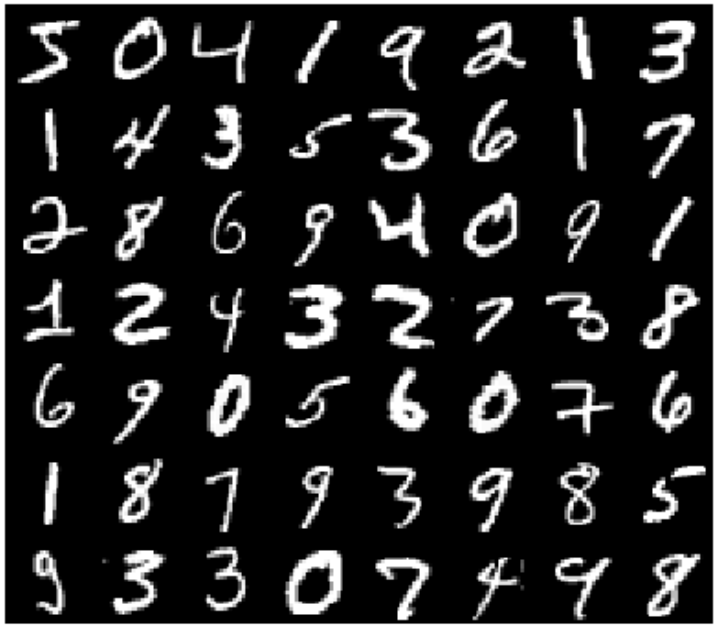
```
ans =
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    3   18   18   18  126  136  175   26  166  255  247  127    0    0    0    0
    0    0    0    0    0    0    0    0   30   36   94  154  170  253  253  253  253  253  225  172  253  242  195   64    0    0    0    0
    0    0    0    0    0    0    0   49  238  253  253  253  253  253  253  253  253  251   93   82   82   56   39    0    0    0    0    0
    0    0    0    0    0    0    0   18  219  253  253  253  253  253  198  182  247  241    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0   80  156  107  253  253  205   11    0   43  154    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0   14    1  154  253   90    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0  139  253  190    2    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0   11  190  253   70    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0   35  241  225  160  108    1    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0   81  240  253  253  119   25    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0   45  186  253  253  150   27    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0   16   93  252  253  187    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0  249  253  249   64    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0   46  130  183  253  253  207    2    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0   39  148  229  253  253  253  250  182    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0   24  114  221  253  253  253  253  201   78    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0   23   66  213  253  253  253  253  198   81    2    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0   18  171  219  253  253  253  253  195   80    9    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0   55  172  226  253  253  253  253  244  133   11    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0  136  253  253  253  212  135  132   16    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
```

Display sample data

Training images

```
>> montage(mnist.train_images(:, :, :, 1:56));
```

You can see following figure on your screen.



Training labels

```
>> mnist.train_labels(1:56)

ans =
    5
    0
    4
    1
    9
    2
    1
    3
    1
    4
    3
    5
    3
    6
    :
```

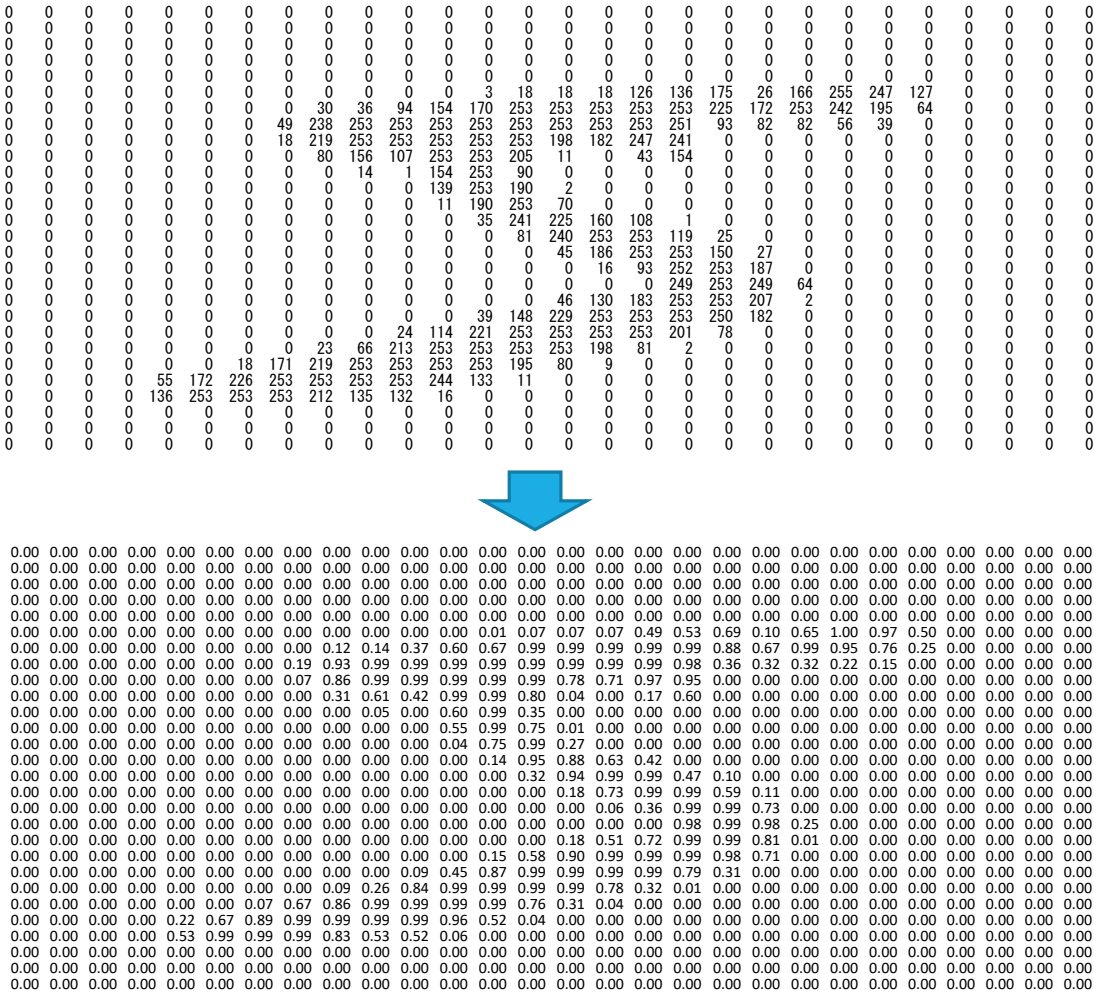
Normalization

Generally, normalization process is required and very important in neural network so that all the inputs are at a comparable range.

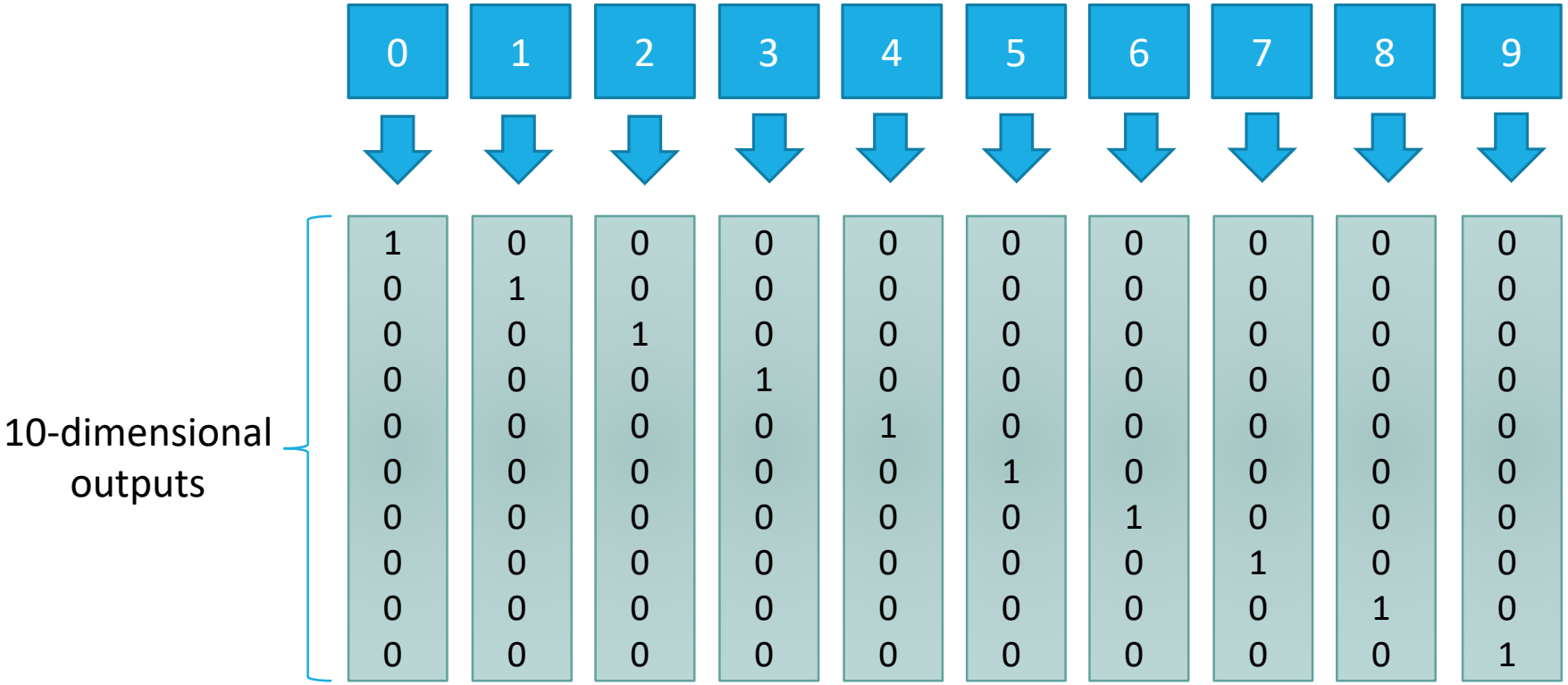
In this case, we will normalize the sample data from 0-255 range (integer value) to 0-1 range (double value). It is most fundamental normalization.

We can use mat2gray() to the conversion as follows.

```
train_images =
mat2gray(mnist.train_images);
```

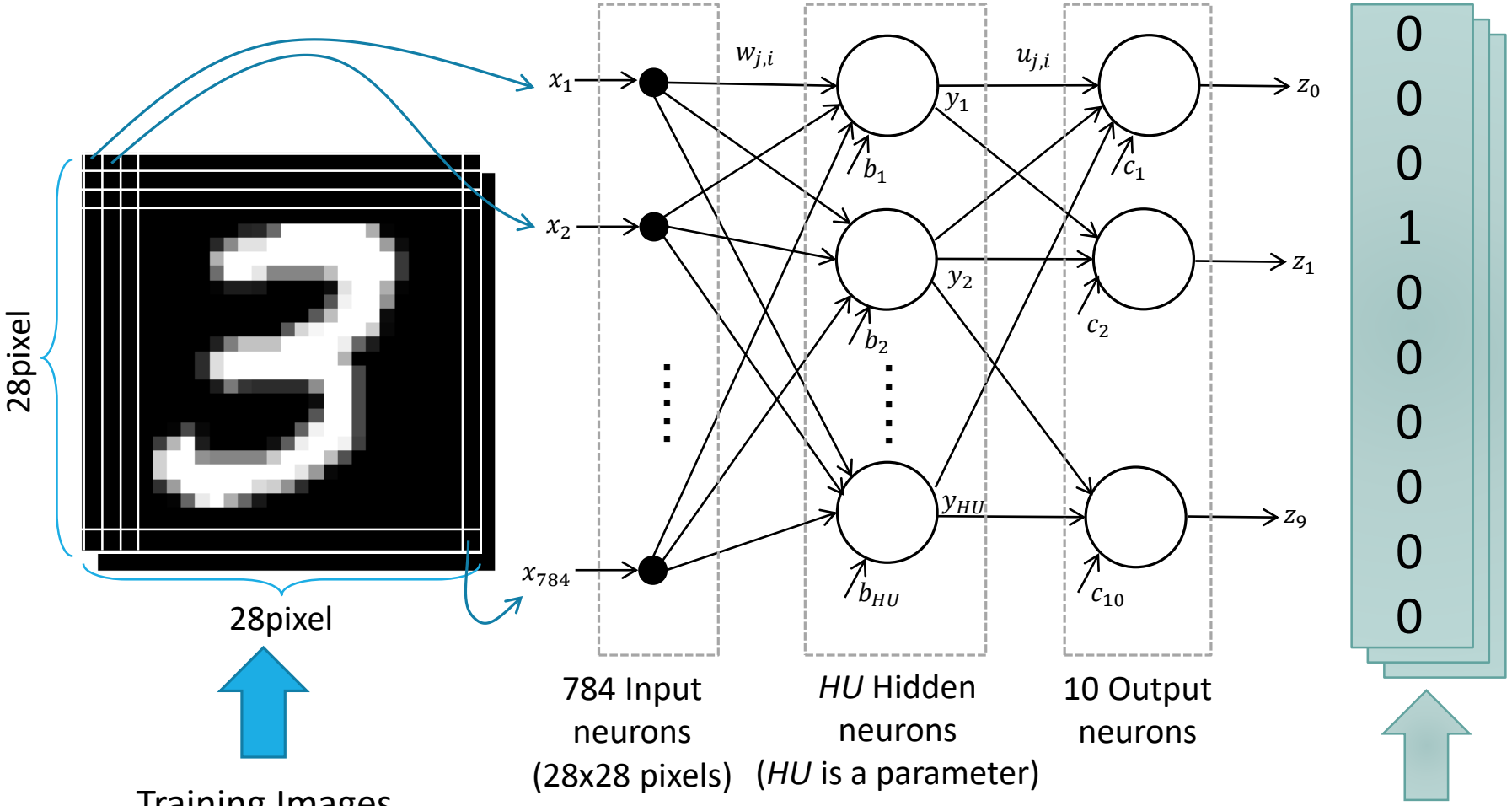


Make label vector



We will use label vector instead of lavel value because in order to equalize distances between each characters.

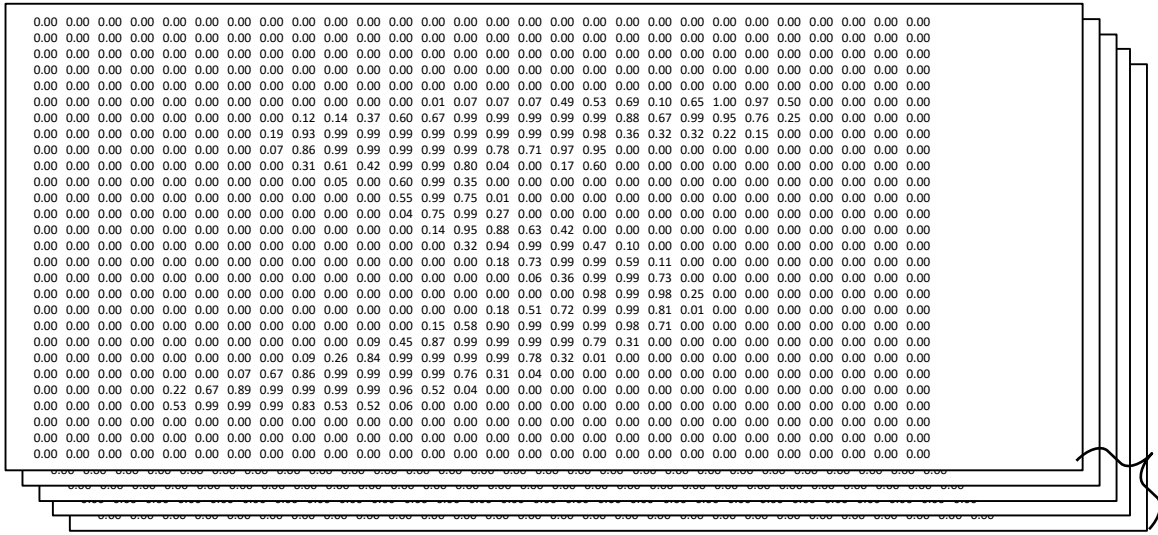
Neural Network Structure



Data loading

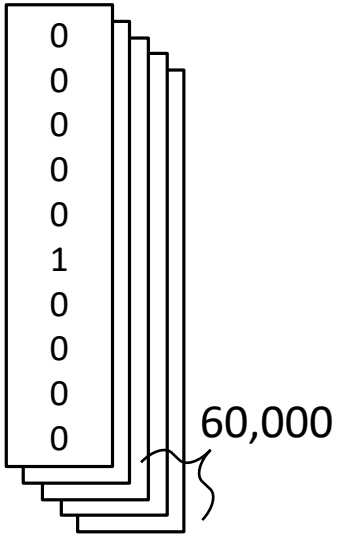
Training data

train_img_mats (28x28x1x60000)



60,000

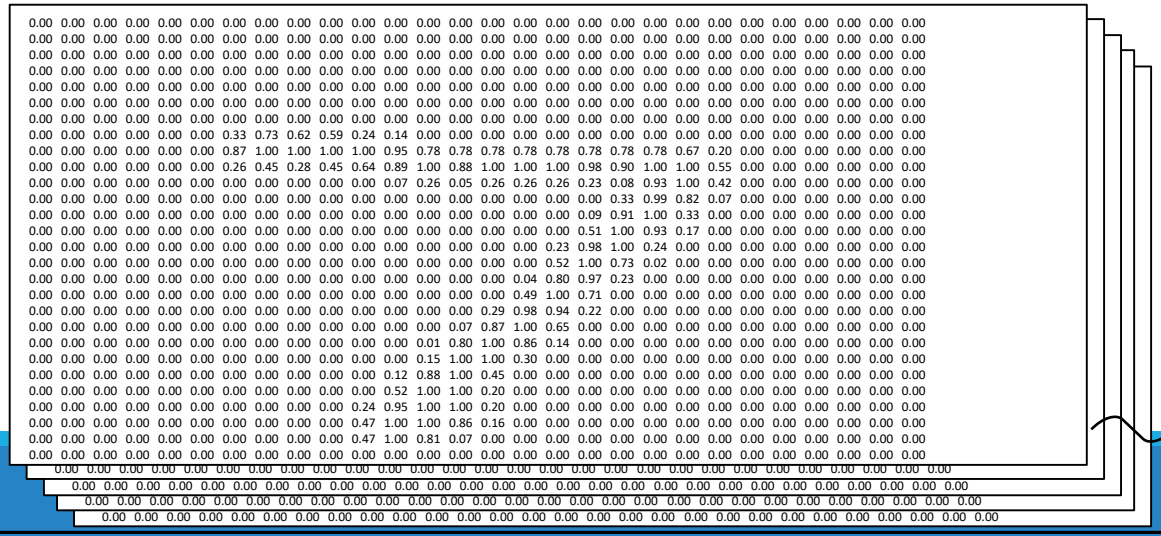
train_label_vecs (10x60000)



60,000

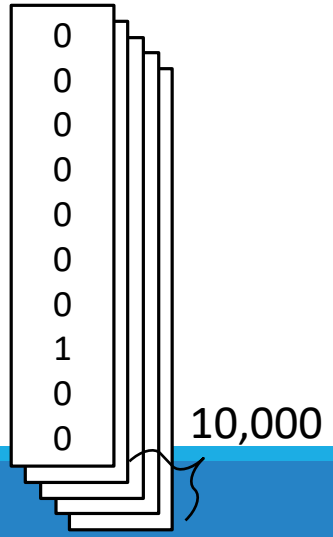
Test data

test_img_mats (28x28x1x10000)



1,000

test_label_vecs (10x10000)



10,000

MATLAB code

Preparing the data

```
%use MNIST traininig datasets
train_num = 1000; % Number of Training samples (MNIST has 60,000 Training samples)
train_images = mat2gray(mnist.train_images); % normalize images
train_labels = mnist.train_labels;
train_label_vecs = zeros(10,train_num);
for n=1:train_num
  %create training label vectors
  train_label_vecs(train_labels(n)+1, n) = 1;
end

%use MNIST test datasets
test_num = 500;   % Number of Test samples (MNIST has 10,000 Test samples)
test_images = mat2gray(mnist.test_images); % normalize images
test_labels = mnist.test_labels;
test_label_vecs = zeros(10,test_num);
for n=1:test_num
  %create test label vectors
  test_label_vecs(test_labels(n)+1, n) = 1;
end
```

Neural network model and learning

example008.m

```
IU = 784; % a number of input neurons (28*28)
HU = 30;  % a number of hidden neurons
OU = 10;  % a number of output neurons

w = 2.0*rand(HU,IU) - 1.0;
b = 2.0*rand(HU,1) - 1.0;
u = 2.0*rand(OU,HU) - 1.0;
c = 2.0*rand(OU,1) - 1.0;

EPOCH=50;  % a number of training epochs
LAMBDA=0.1; % learning rate
```

```
for epoch=1:EPOCH
  for n=1:train_num
    img = train_images(:,:,1,n);
    x = img(:);
    y = sigmoid_neuron(x,w,b);
    z = sigmoid_neuron(y,u,c);
    t = train_label_vecs(1:10,n);
    o(1:10,n)=z;

    delta_out = (z-t).*(z.*(1-z));
    delta_hidden = (delta_out'*u)'.*(y.*(1-y));

    u=u-LAMBDA * delta_out * y';
    c=c-LAMBDA * delta_out;
    w=w-LAMBDA * delta_hidden * x';
    b=b-LAMBDA * delta_hidden;
  end

  % calculate current LOSS
  loss(epoch) = loss_function(o, train_label_vecs);
end

%%%%%%%%%% Display the results %%%%%%%%%%%%%%%
% Display cost change graph
figure(1);
plot([1:epoch], loss)
title('Cost on the training data');
xlabel('Epoch');
ylabel('Cost');
```

example101.m

Neural network model and learning

example008.m

```
IU = 784; % a number of input neurons (28*28)
HU = 30;  % a number of hidden neurons
OU = 10;  % a number of output neurons

w = 2.0*rand(HU,IU) - 1.0;
b = 2.0*rand(HU,1) - 1.0;
u = 2.0*rand(OU,HU) - 1.0;
c = 2.0*rand(OU,1) - 1.0;

EPOCH=50;  % a number of training epochs
LAMBDA=0.1; % learning rate
```

```
for epoch=1:EPOCH
  for n=1:train_num
    img = train_images(:,:,1,n);
    x = img(:);
    y = sigmoid_neuron(x,w,b);
    z = sigmoid_neuron(y,u,c);
    t = train_label_vecs(1:10,n);
    o(1:10,n)=z;

    delta_out = (z-t).*(z.*(1-z));
    delta_hidden = (delta_out'*u)'.*(y.*(1-y));

    u=u-LAMBDA * delta_out * y';
    c=c-LAMBDA * delta_out;
    w=w-LAMBDA * delta_hidden * x';
    b=b-LAMBDA * delta_hidden;
  end

  % calculate current LOSS
  loss(epoch) = loss_function(o, train_label_vecs);
end

%%%%%%%%%% Display the results %%%%%%%%%%%%%%%
% Display cost change graph
figure(1);
plot([1:epoch], loss)
title('Cost on the training data');
xlabel('Epoch');
ylabel('Cost');
```

example101.m

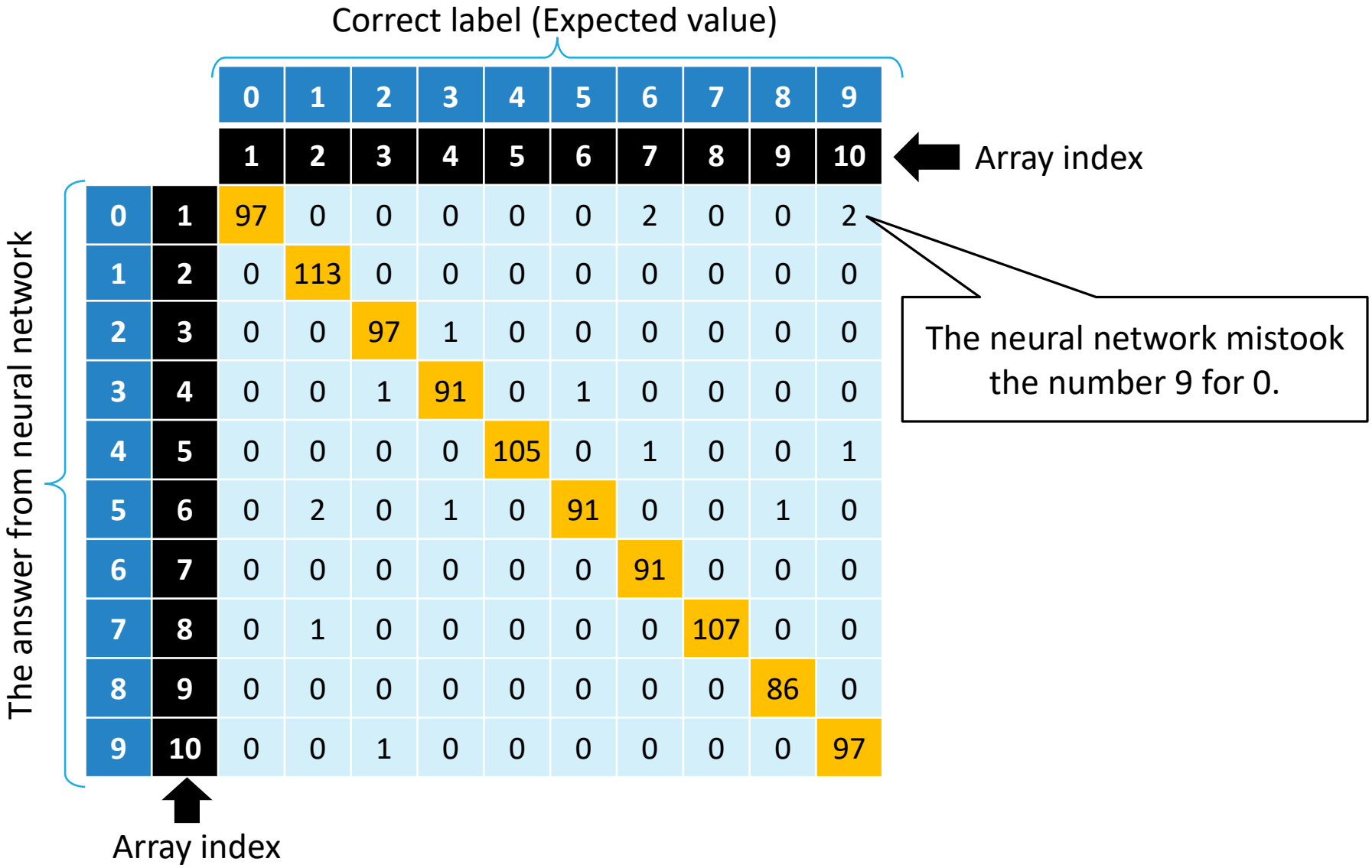
Displaying results

```
% Calculate train accuracy
train_res_mat = zeros(10, 10);
for n=1:train_num
  img = train_images(:,:,1,n);
  x = img(:);
  y = sigmoid_neuron(x,w,b);
  z = sigmoid_neuron(y,u,c);
  [value, index] = max(z);
  train_res_mat(index, train_labels(n)+1) = ...
    train_res_mat(index, train_labels(n)+1) + 1;
end
train_res_mat
train_accuracy =
trace(train_res_mat)/sum(sum(train_res_mat))
```

Training accuracy

```
train_res_mat =

    97     0     0     0     0     0     2     0     0     2
     0   113     0     0     0     0     0     0     0     0
     0     0    97     1     0     0     0     0     0     0
     0     0     1    91     0     1     0     0     0     0
     0     0     0     0   105     0     1     0     0     1
     0     2     0     1     0    91     0     0     1     0
     0     0     0     0     0     0    91     0     0     0
     0     1     0     0     0     0     0   117     0     0
     0     0     0     0     0     0     0     0    86     0
     0     0     1     0     0     0     0     0     0    97

train_accuracy =  0.98500
```

Result matrics



Displaying results

```
% Calculate train accuracy
train_res_mat = zeros(10, 10);
for n=1:train_num
  img = train_images(:,:,1,n);
  x = img(:);
  y = sigmoid_neuron(x,w,b);
  z = sigmoid_neuron(y,u,c);
  [value, index] = max(z);
  train_res_mat(index, train_labels(n)+1) = ...
    train_res_mat(index, train_labels(n)+1) + 1;
end
train_res_mat
train_accuracy = trace(train_res_mat)/sum(sum(train_res_mat))
```

Same as learning part

```
z =
   0.1
   0.2
   0.9
   0.0
   0.3
   0.1

>> [value, index] = max(z)

value =
   0.9
index =
    3
```

Sum of diagonal elements

Sum of all elements

train_labels(n)+1

		0	1	2	3	4	5	6	7	8	9
		1	2	3	4	5	6	7	8	9	10
0	1	97	0	0	0	0	0	2	0	0	2
1	2	0	113	0	0	0	0	0	0	0	0
2	3	0	0	97	1	0	0	0	0	0	0
3	4	0	0	1	91	0	1	0	0	0	0
4	5	0	0	0	0	105	0	1	0	0	1
5	6	0	2	0	1	0	91	0	0	1	0
6	7	0	0	0	0	0	0	91	0	0	0
7	8	0	1	0						0	0
8	9	0	0	0						86	0
9	10	0	0	1	0	0	0	0	0	0	97

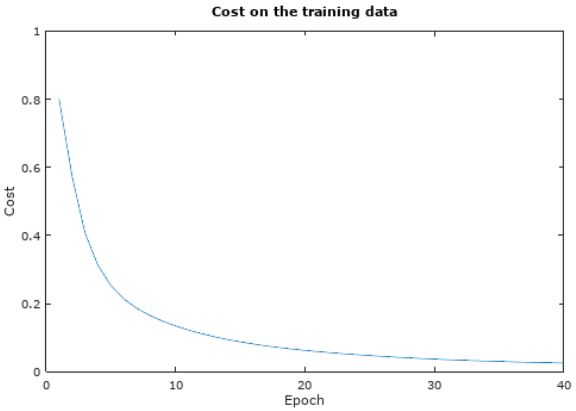
index

Increment!

Displaying results

```
%%%%%%%%%% Display the results %%%%%%%%%%%%%%
% Display cost change graph
figure(1);
plot([1:epoch], loss)
title('Cost on the training data');
xlabel('Epoch');
ylabel('Cost');

% Calculate train accuracy
train_res_mat = zeros(10, 10);
for n=1:train_num
  img = train_images(:,:,1,n);
  x = img(:);
  y = sigmoid_neuron(x,w,b);
  z = sigmoid_neuron(y,u,c);
  [value, index] = max(z);
  train_res_mat(index, train_labels(n)+1) =
train_res_mat(index, train_labels(n)+1) + 1;
end
train_res_mat
train_accuracy = trace(train_res_mat)/sum(sum(train_res_mat))

% Calculate test accuracy
test_res_mat = zeros(10, 10);
for n=1:test_num
  img = test_images(:,:,1,n);
  x = img(:);
  y = sigmoid_neuron(x,w,b);
  z = sigmoid_neuron(y,u,c);
  [value, index] = max(z);
  test_res_mat(index, test_labels(n)+1) = test_res_mat(index,
test_labels(n)+1) + 1;
end
test_res_mat
test_accuracy = trace(test_res_mat)/sum(sum(test_res_mat))
```

LOSS changing



Training accuracy

```
train_res_mat =

    97     0     0     0     0     0     2     0     0     2
     0   113     0     0     0     0     0     0     0     0
     0     0    97     1     0     0     0     0     0     0
     0     0     1    91     0     1     0     0     0     0
     0     0     0     0   105     0     1     0     0     1
     0     2     0     1     0    91     0     0     1     0
     0     0     0     0     0     0    91     0     0     0
     0     1     0     0     0     0     0   117     0     0
     0     0     0     0     0     0     0     0    86     0
     0     0     1     0     0     0     0     0     0    97

train_accuracy =  0.98500
```

Testing accuracy

```
test_res_vec =

    41     0     0     0     0     1     2     0     1     0
     0    67     0     0     0     0     1     0     0     0
     0     0    49     3     0     2     4     0     1     0
     0     0     0    32     0     1     0     2     0     1
     0     0     0     0    49     0     2     1     1     0
     0     0     0     8     0    42     3     0     3     2
     0     0     0     0     2     1    31     0     1     0
     1     0     4     2     0     3     0    44     1     5
     0     0     1     0     0     0     0     0    29     1
     0     0     1     0     4     0     0     2     3    45

test_accuracy =  0.85800
```

Exercise 4.1

Execute example008.m and obtain

(1) cost changing graph

(2) training accuracy and the result matrics

(3) testing accuracy and the result matrics

Exercise 4.2

Try to increase the performance of the neural network by setting the parameter to various values.