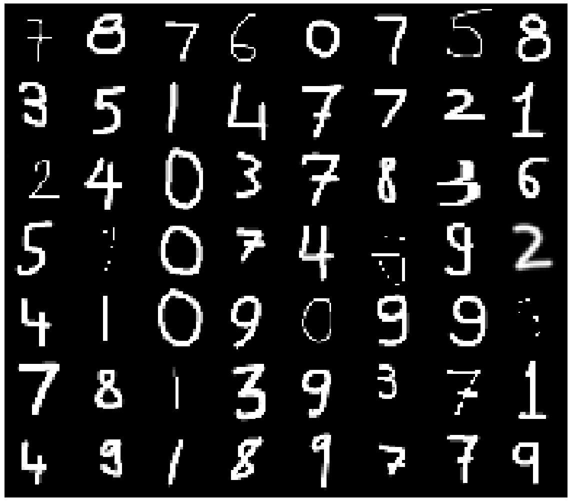
Day 5

Improvement Plans for Neural Network Learning and Final Exercises

Read MYDATA image dataset

Unzip mydata.zip and put the “mydata” folder in the “data” folder in your MATLAB workspace.
Run load_mydata.m to use MYDATA dataset as follows.

```
>> mydata = load_mydata();
        TRAIN_NUM =  1800
        TEST_NUM =  360

>> montage(mydata.train_images(:,:,1,[1:56]))
```



Read_image.m

```
sn = randperm(imgNum);
TRAIN_NUM = 1800
TEST_NUM = imgNum - TRAIN_NUM
mydata.train_images = out_images(:,:,1,sn(1:TRAIN_NUM));
mydata.train_labels = out_labels(sn(1:TRAIN_NUM),1);
mydata.test_images = out_images(:,:,1,sn(TRAIN_NUM + 1:end));
mydata.test_labels = out_labels(sn(TRAIN_NUM+1:end),1);
```

divide into train_data and test data

【Plan1】Learning late decay

If the learning rate λ is gradually decreased, an optimal weights can be efficiently obtained.

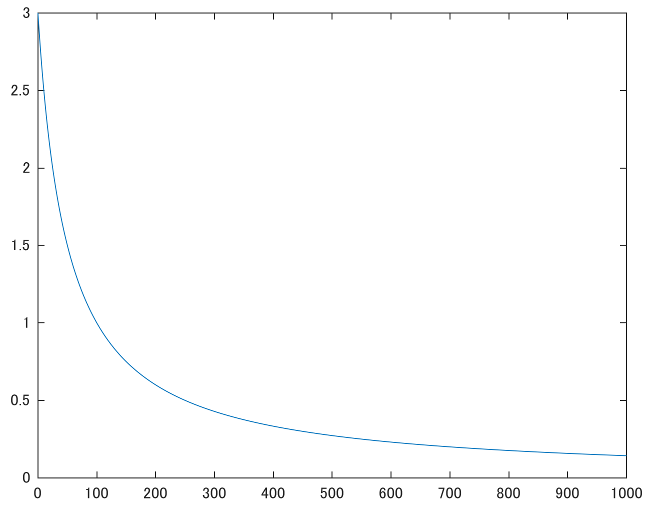
Update function

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

Learinng rate decay

$$\lambda = \Lambda * \frac{1}{1 + \rho * epoch}$$

Λ: Initial learning rate
ρ: Decaying rate



【Plan2】AdaGrad

AdaGrad is an improved method of Learning rate decay.

Adjust the decay rate for each weight and each bias

Update function

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

$$u_{k,j} \leftarrow u_{k,j} + \Delta u \qquad \text{where } \Delta u = -\lambda \frac{\partial L}{\partial u_{k,j}}$$

AdaGrad

$$g_{k,j} \leftarrow g_{k,j} + \left(\frac{\partial L}{\partial u_{k,j}}\right)^2$$

$$\lambda = \Lambda * \frac{1}{\sqrt{g_{k,j}}}$$

The initial value of $g_{k,j}$ is 0

$g_{k,j}$ monotonically increases

Λ: Initial learning rate

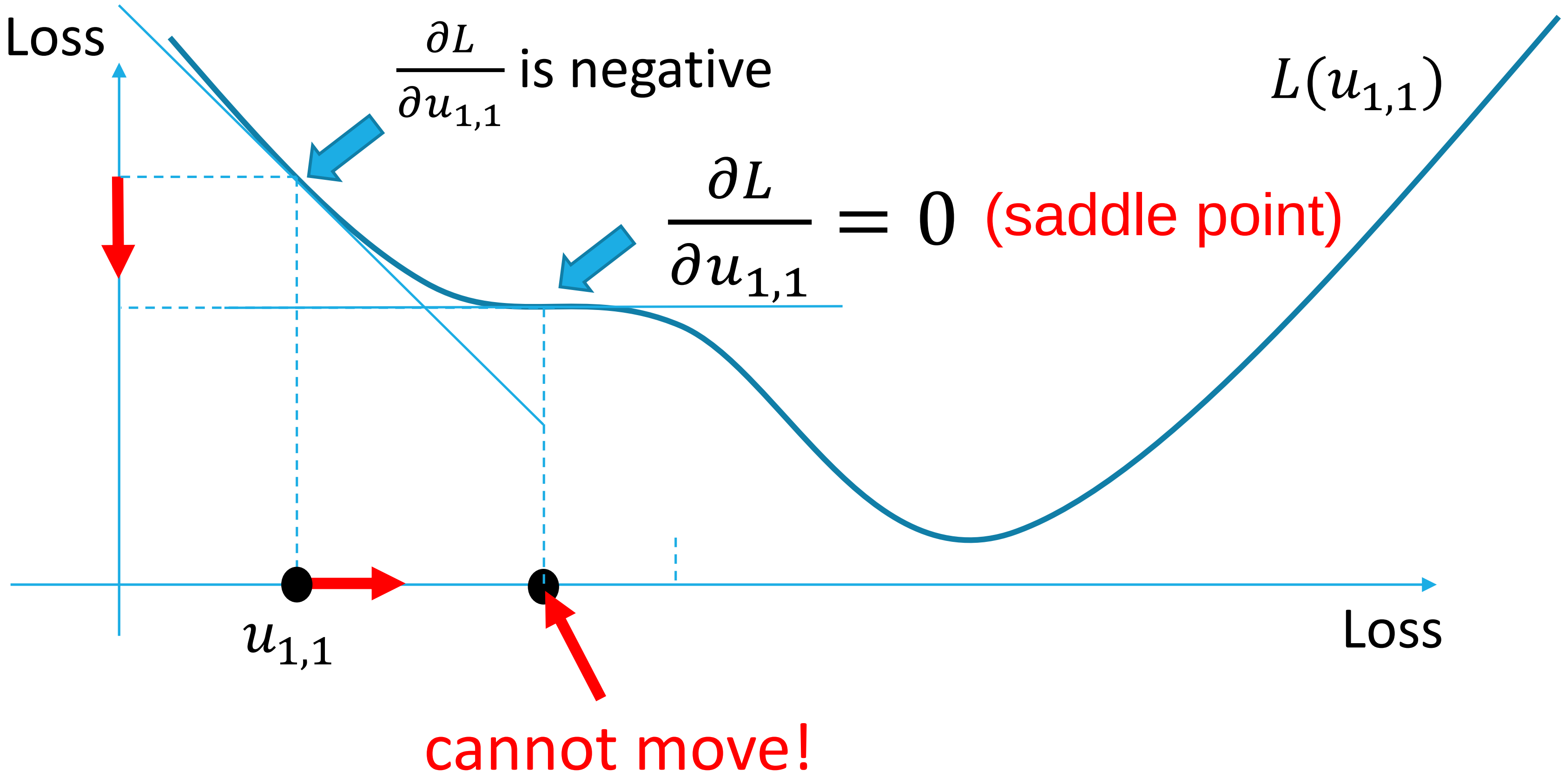
【Plan2】AdaGrad - Implementation on MATLAB

Affine.m using AdaGrad

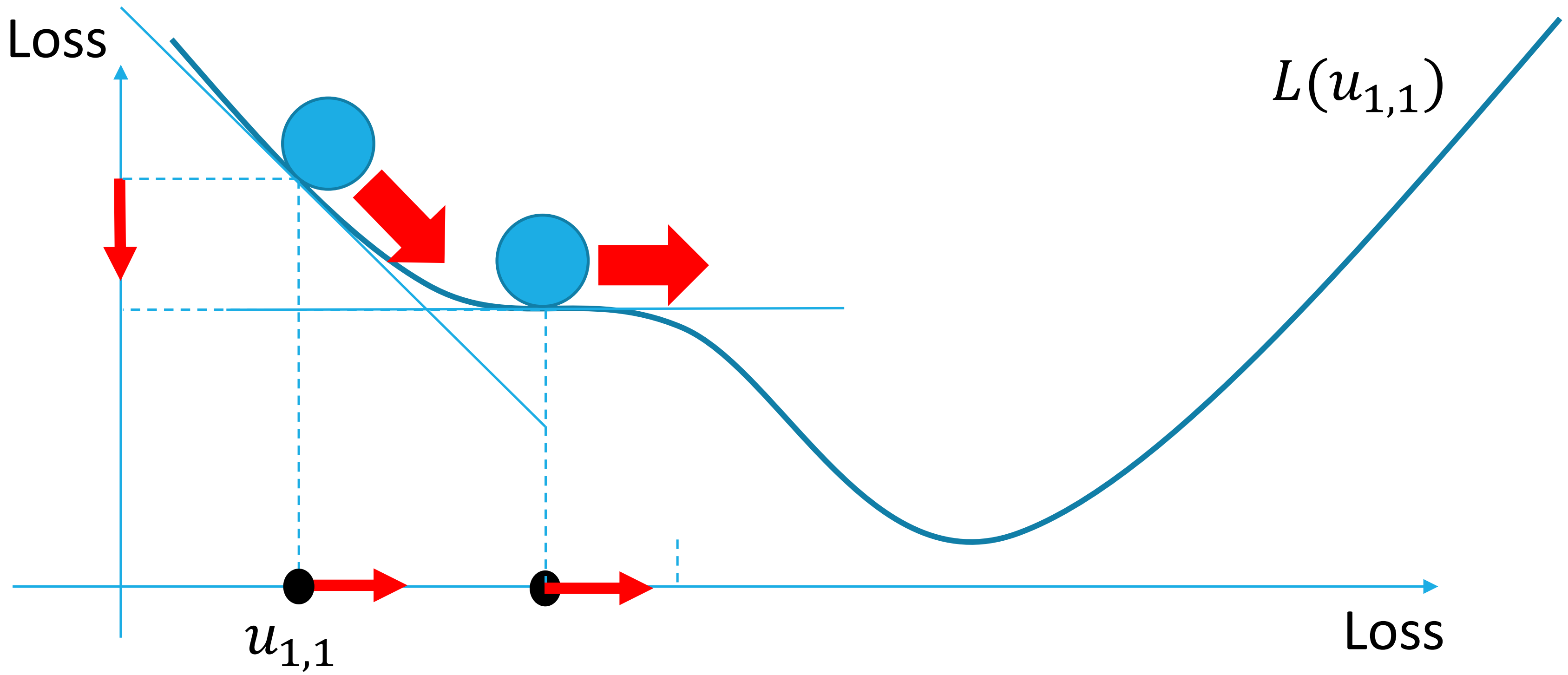
```
classdef Affine < handle
  properties
       :
    gw; % for AdaGrad
    gb; % for AdaGrad
  end

  methods
    function obj = Affine(w,b)
      obj.weights = w;
      obj.bias = b;
      gw = zeros(size(w)) * 0.0001; % for AdaGrad
      gb = zeros(size(b)) * 0.0001; % for AdaGrad
    end

      :

    function update(obj, learning_rate) % AdaGrad
      obj.gw = obj.gw + obj.dw .^ 2;
      obj.gb = obj.gb + obj.db .^ 2;
      lambda_g = learning_rate * obj.gw .^ (-1/2);
      lambda_b = learning_rate * obj.gb .^ (-1/2);
      obj.weights = obj.weights - lambda_g .* obj.dw;
      obj.bias = obj.bias - lambda_b .* obj.db;
    end
  end
end
```

【Plan3】Momentum

Weights cannot move at saddle point


Loss
$\frac{\partial L}{\partial u_{1,1}}$ is negative
$L(u_{1,1})$
$\frac{\partial L}{\partial u_{1,1}} = 0$ (saddle point)
$u_{1,1}$
Loss
cannot move!

【Plan3】Momentum



Weights has its own momentum (i.e., inertia occurs when updating)

【Plan3】Momentum

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

Speed

$$u_{k,j} \leftarrow u_{k,j} + \Delta u \quad \text{where } \Delta u = -\lambda \frac{\partial L}{\partial u_{k,j}}$$

Momentum

$$u_{k,j} \leftarrow u_{k,j} + \Delta u \quad \text{where } \Delta u = \boxed{\eta \Delta u} - \lambda \frac{\partial L}{\partial u_{k,j}}$$

momentum term

η is a momentum parameter

【Plan4】Cross Entropy

Another useful loss function is “Cross Entropy” as follows

$$LOSS = -\frac{1}{N}\sum_n \{t \log y + (1-t)\log(1-y)\}$$

Cross entropy provides high performance to learn neural network and good compatibility with Sigmoid function.

Point 1

Both y and $1-y$ are values between 0 and 1 because y is an output of Sigmoid function. Then, both $\log y$ and $\log(1-y)$ are negative value.
Therefore, LOSS become a positive value.

Point 2

If both t and y are 0, LOSS become 0.
If both t and y are 1, LOSS also become 0.

【Plan4】Cross Entropy

For the loss function, we can ignore the coefficient $\frac{1}{N}$. then,

$$
\begin{aligned}
L(y) &= -(t\log y + (1-t)\log(1-y)) \\
\frac{\partial L}{\partial y} &= -\left(\frac{t}{y} - \frac{1-t}{1-y}\right) \\
&= -\left(\frac{t(1-y) - y(1-t)}{y(1-y)}\right) \\
&= -\left(\frac{t - ty - y + ty}{y(1-y)}\right) \\
&= -\left(\frac{t-y}{y(1-y)}\right) \\
&= \frac{y-t}{y(1-y)}
\end{aligned}
$$

【Plan4】Cross Entropy

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial w_i}$$

The derivatives as follows,

$$\boxed{\frac{\partial L}{\partial y} = \frac{y - t}{y(1-y)}} \qquad \frac{\partial y}{\partial p} = \frac{\partial \sigma(y)}{\partial p} = y(1-y) \qquad \frac{\partial p}{\partial w_i} = x_i$$

$$\boxed{\text{Then,} \quad \frac{\partial L}{\partial w_i} = (y - t)\, x_i}$$

Things you should do and how will I assess your work?

- From the first day to the fifth day, at the beginning of everyday lecture I will give you both theoretical and mathematical principles and some examples using MATLAB.
- Then you have some exercises and answer them.
- On the final day, you will implement a recognition program for hand-written digits and I will give you some assignments about the program. Answer them and send me your answer file in pdf format and the MATLAB script file (*.m file) before the deadline. The deadline is 31 Aug. 2018. My e-mail address is yamashita@tokyo-ct.ac.jp. These assignments are worth 50% of your final grade.
- In addition, on the final day I hand out a piece of paper; that is the final test. Then answer it by the end of the final lecture. The test is worth 50% of your final grade.
- Basically five days attendance is required to get credit.

Assignment 1 (Required)

MYDATA dataset has 2,160 hand-written digit images.

Divide the mydata into 1,800 learning data and 360 test data.

Then, recognize the test data using neural network model learned with learning data and display the loss graph and results matrices.

Summarize the results (the loss graph and results matrices) in a PDF file and submit the PDF file and the MATLAB source code.

Assignment 2 (Required)

Load MYDATA dataset and MNIST dataset into a variable “mydata” and “minist” respectively.

Recognize the MYDATA dataset using neural network model learned with “MNIST dataset” and display the loss graph and results matrices.

Summarize the results (the loss graph and results matrices) in a PDF file and submit the PDF file and the MATLAB source code.

Assignment 3 (Optional)

Recognize the MYDATA or MNIST dataset using NN model with "Learning rate decay" method. Compare the accuracy with and without Learning rate decay method.

Summarize the results matrix, accuracy and misrecognized images in a PDF file and submit the PDF file and the MATLAB source code.

Assignment 4 (Optional)

Recognize the MYDATA or MNIST dataset using NN model with “AdaGrad” method. Compare the accuracy with and without AdaGrad method.

Summarize the results matrix, accuracy and misrecognized images in a PDF file and submit the PDF file and the MATLAB source code.

Assignment 5 (Optional)

Recognize the MYDATA or MNIST dataset using NN model with "Momentum" method. Compare the accuracy with and without Momentum method.

Summarize the results matrix, accuracy and misrecognized images in a PDF file and submit the PDF file and the MATLAB source code.

Assignment 6 (Optional)

Recognize the MYDATA or MNIST dataset using NN model with "cross entropy function as a loss function". Compare the accuracy between MSE function and cross entropy function.

Summarize the results matrix, accuracy and misrecognized images in a PDF file and submit the PDF file and the MATLAB source code.

Review for the final test

Basic Operation in MATLAB (2)
(The colon notation)

```
>> n1 = 1:5
n1 =
     1   2   3   4  5

>> n2 = 1:2:5
n2 =
      1  3  5

>> n3 = 5:-1:1
n3 =
      5  4  3  2  1

>>n4 = 5:-2:1
n4 =
      5  3  1
```

The colon notation is very useful to access blocks of elements.

The notation 1:5 says to start at 1, count up by 1 and stop when the count reaches 5.

The notation 1:2:5 says to start at 1, count up by 2 and stop when the count reaches 5.

The notation 5:-1:1 says to start at 5, decrease by 1 and stop when it reaches 1.

Basic Operation in MATLAB (3) (Vector Manipulations)

```
>> v=[ 1 3 5 7 9 ]
v =
     1   3   5   7  9
```

A row vector, or an array of dimension 1xN, is created by square brackets. The elements are separated by spaces or by commas.

```
>> v(2)
ans =
       3
```

v(2) is the second element of vector v.

Note that MATLAB array indexing starts from 1 not 0.

```
>> v(1:3)
ans =
      1  3  5
```

To access the first three elements of v, we write v(1:3).

Basic Operation in MATLAB (4) (Vector Manipulations)

```
>> v(2:4)
ans =
      3  5  7
```

The notation v(2:4) means we can access the second through the fourth elements.

```
>> v(2:end)
ans =
      3  5  7  9
```

The notation ‘end’ signifies the last element.

```
>> v(1:2:5)
ans =
      1  5  9
```

The index is not restricted to contiguous elements.

```
>> v(end:-2:1)
ans =
      9  5  1
```

The index count starts at the last element, decreases by 2, and stops when it reaches the first element.

Basic Operation in MATLAB (5) (Matrix Manipulations)

```
>> A=[1 2 3 4; 5 6 7 8; 9 1 2 3]
A =
     1   2   3   4
     5   6   7   8
     9   1   2   3
```

A matrix is entered row by row,
And each row is separated by the semicolon(;).
Within each row, elements are separated by a space or a comma(,).

```
>> B=A'
B =
     1   5   9
     2   6   1
     3   7   2
     4   8   3
```

A' is a transpose of matrix A.

Basic Operation in MATLAB (6) (Matrix Manipulations)

```
>> A
A =
      1   2   3   4
      5   6   7   8
      9   1   2   3
```

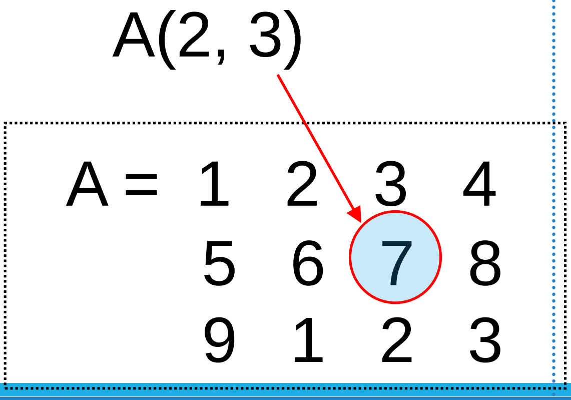
If we want to check the A matrix again, we simply type A.

```
 >> A(2, 3)
ans = 7
```

If we want to extract the element in the second row, third column, we write A(2,3).

```
>> A(2, 3) = 0
A =
      1   2   3   4
      5   6   0   8
      9   1   2   3
```

If we want to set the element to 0, we write A(2, 3)=0.



Basic Operation in MATLAB (7) (Matrix Manipulations)

```
>> A(:, 3)
ans =
      3
      0
      2
```

A(:, 3) is the third column of matrix A.

The colon operator (:) stands for all columns or all rows.

```
>> A(1, :)
ans =
      1   2   3   4
```

A(1, :) represents the first row of matrix A.

```
>> A(1, :) + A(3, :)
ans = 10   3   5   7
```

This means addition of the first and third rows of A

Basic Operation in MATLAB (8)
(Matrix Manipulations)

```
>> A(:, 1) = 0
A =
     0   2   3   4
     0   6   0   8
     0   1   2   3

>> A(end, end) = 0
A =
     0   2   3   4
     0   6   0   8
     0   1   2   0

>> A(end, end-1)
ans =
       2
```

If we want to set the first column to 0s, we write A(:, 1)=0.

Basic Operation in MATLAB (9)
(Matrix Manipulations)

```
>> A(:, 2:4)
ans =
        2   3   4
        6   0   8
        1   2   0
```

A(:,2:3) is a sub-matrix with the last three columns of A.

A(:, 2:4)

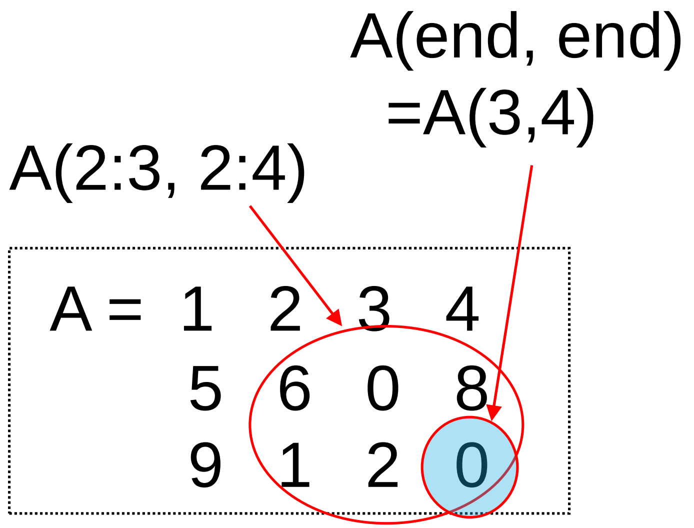
$$A = \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 0 & 8 \\ 9 & 1 & 2 & 0 \end{matrix}$$

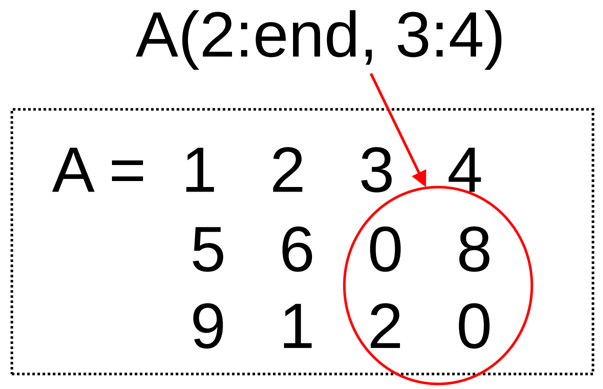
```
>> A(:, 2) = [ ]
ans =
        1   3   4
        5   0   8
        9   2   0
```

A row or a column of a matrix can be deleted by setting it to a null vector, [ ].

Basic Operation in MATLAB (10)
(Matrix Manipulations)

```
>> A(2:3, 2:4)
ans =
      6   0   8
      1   2   0

>> A(2:end, 3:4)
ans =
      0   8
      2   0
```





Basic Operation in MATLAB (11)
(Matrix Manipulations)

```
>> C=[1 3 5; 7 9 2; 4 6 8];
>> C
C =
     1   3   5
     7   9   2
     4   6   8

>> C(end:-1:1, :)
ans =
       4   6   8
       7   9   2
       1   3   5
```

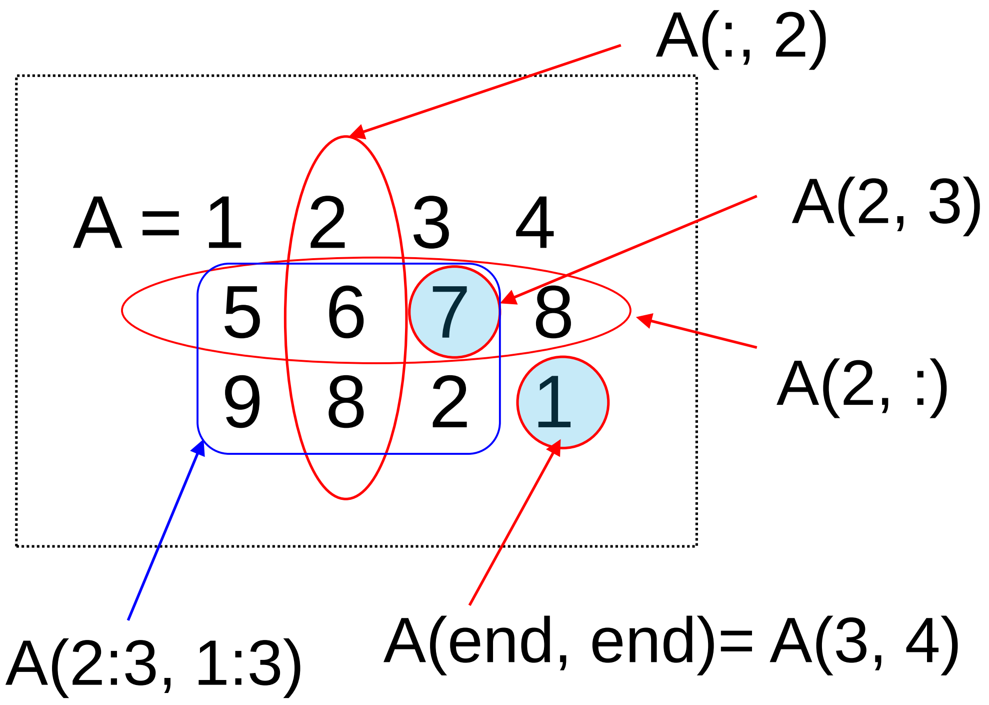
MATLAB does not display output on the screen when an operation ends with the semicolon(;).

If we want to check the C matrix again, we simply type C.

Basic Operation in MATLAB (12)
(**Summary**: Matrix Manipulations – Array Indexing)

- MATLAB supports a number of powerful indexing schemes that simplify array manipulation.
- Basic indexing in two dimensions is illustrated as below.

- A is given as the 4x4 matrix.
- A(2, 3) and A(end, end) represent elements, respectively.
- A(2, :) represents a row vector.
- A(:, 2) represents a column vector.
- A(2:3, 1:3) represents a submatrix.



Basic Operation in MATLAB (13)

You can use a function such as zeros, ones and rand to create a matrix.

```
>> zeros(2, 3)
ans =

   0    0    0
   0    0    0
```

```
>> ones(4, 1)
ans =

   1
   1
   1
   1
```

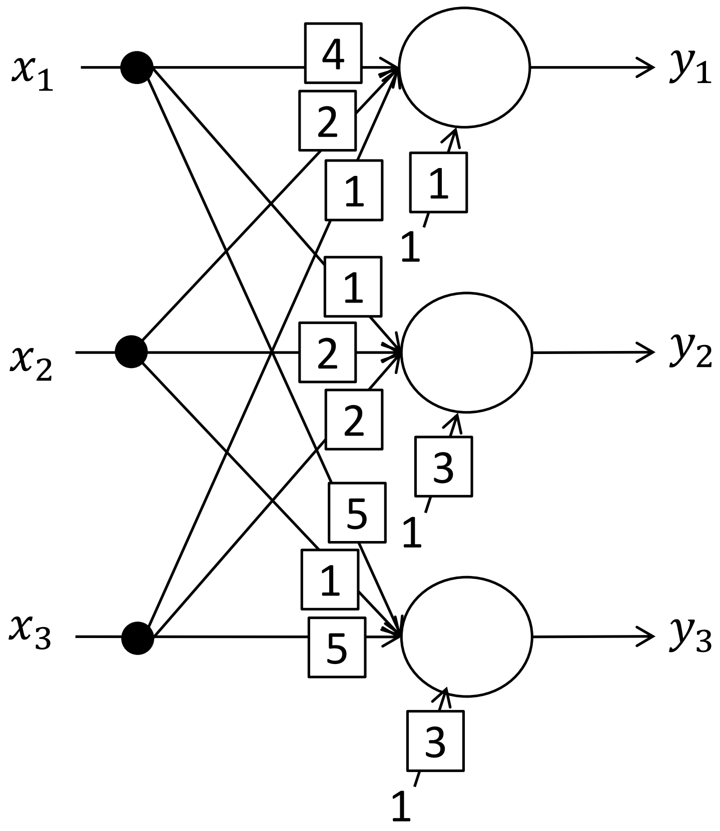
```
>> rand(3, 4)
ans =

   0.56688    0.87818    0.41082    0.62880
   0.11144    0.60390    0.51978    0.82911
   0.48224    0.80686    0.93745    0.39139
```

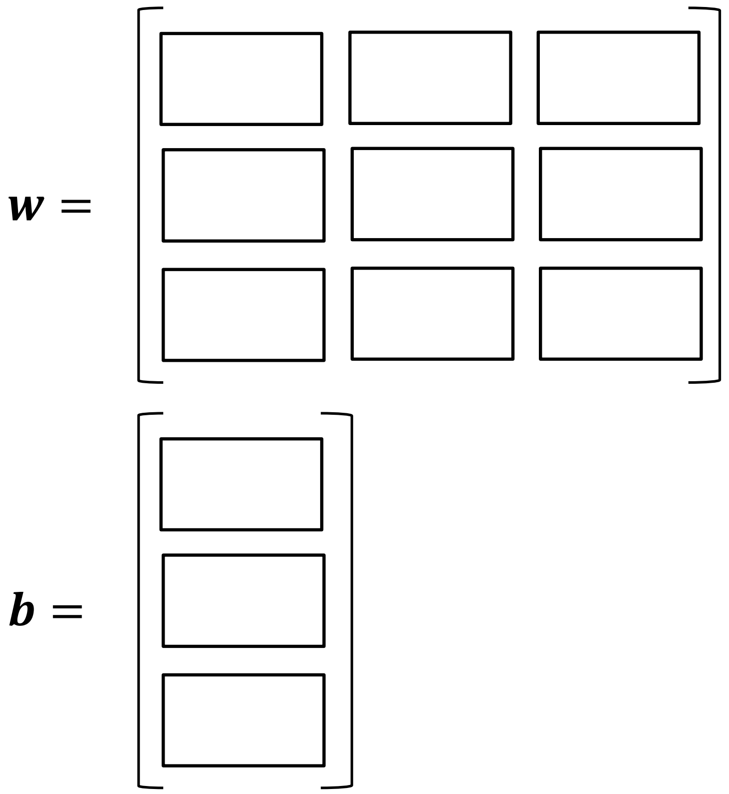
To transpose a matrix, use a single quote (‘).

```
>> c = [1 2 3; 4 5 6; 7 8 9]
c =

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

```
>> c’
ans =

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

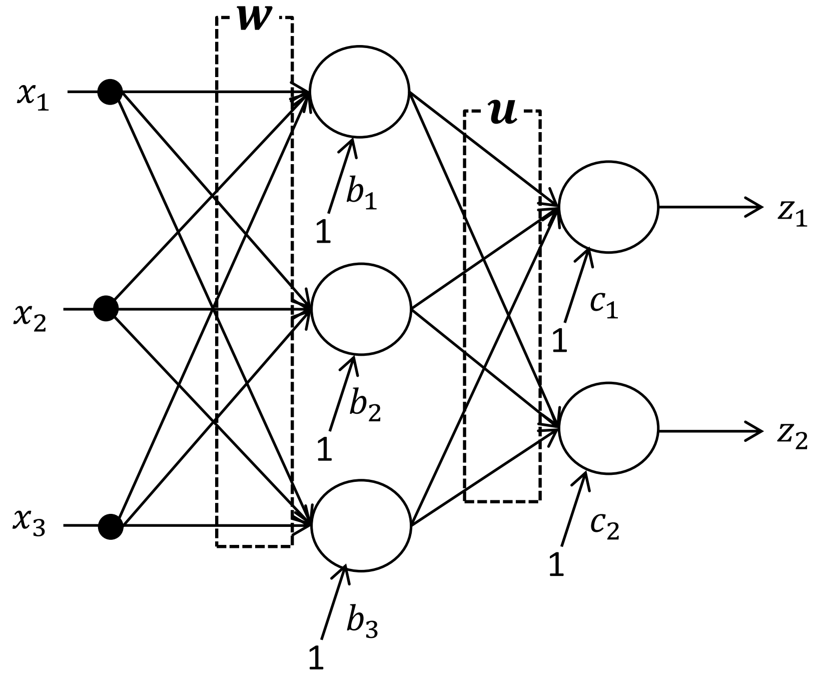
Matrix representation





Feedforward calculation (review for Exercise1.5)

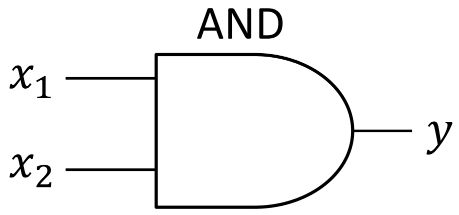
Consider a following neural network which have 4 neurons in the hidden layer and 3 neurons in the output layer. Calculate outputs where the inputs $\boldsymbol{x}$, weights $\boldsymbol{w}, \boldsymbol{u}$ and thresholds $\boldsymbol{h}, \boldsymbol{g}$ are given as follows by hand calculation.



$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} 3 & 2 & 2 \\ 4 & 1 & 5 \\ 0 & 1 & 0 \end{bmatrix} \quad \boldsymbol{h} = \begin{bmatrix} 2 \\ 6 \\ 1 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} 4 & 4 & 3 \\ 3 & 1 & 2 \end{bmatrix} \quad \boldsymbol{g} = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

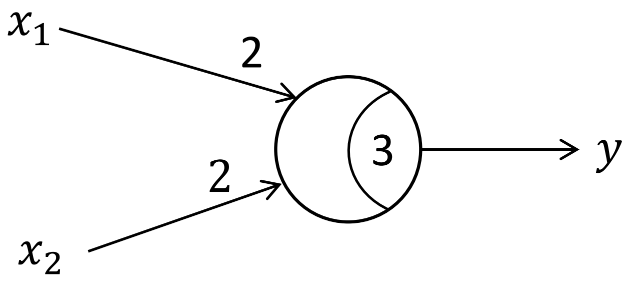
Representation of logic functions using formal neurons (1)

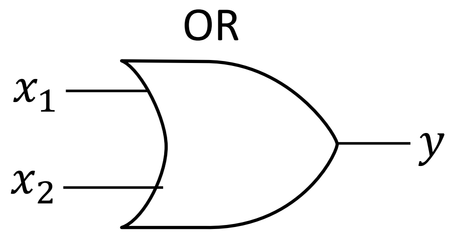
Some of basic logic functions can be constructed with a formal neuron.



Truth table for AND

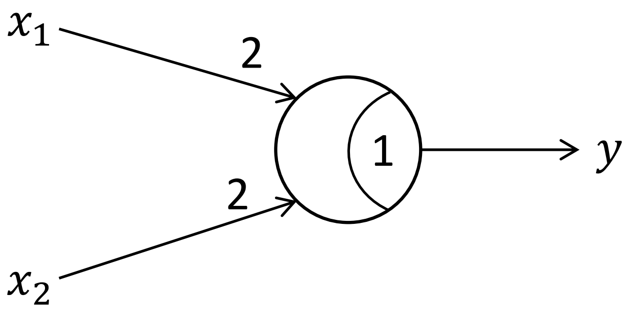
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1





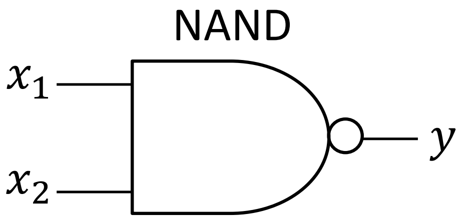
Truth table for OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



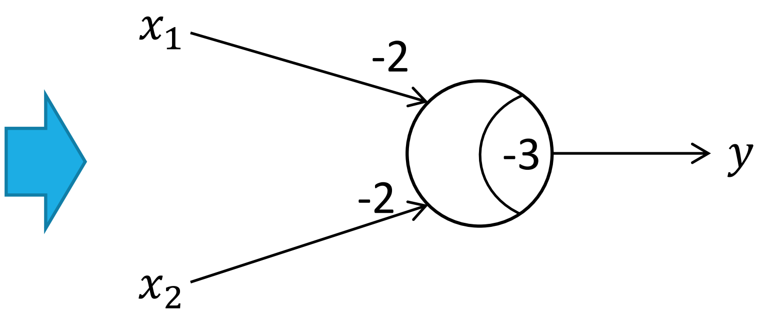
Representation of logic functions using formal neurons (2)

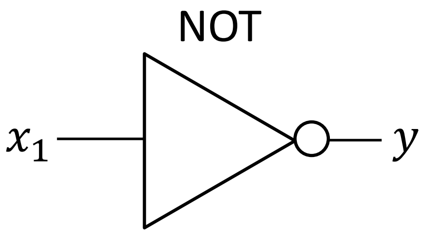
NAND and NOT gate can be constructed by using a negative values as weights and a threshold.



Truth table for NAND

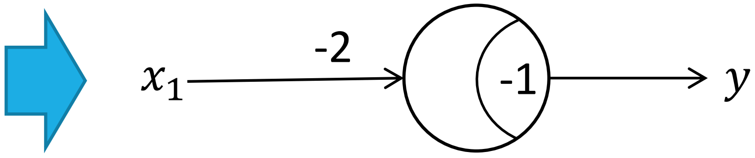
x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0





Truth table for NOT

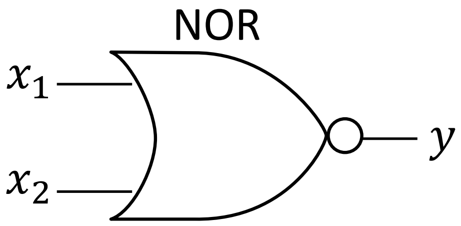
x_1	y
0	1
1	0



※ Any logical circuits can be constructed with only NAND gates.

Exercise2.1

Construct a NOR gate with a formal neuron by setting appropriate weights and a threshold.



Truth table for NOR

x_1	x_2	y
0	0	1
0	1	0
1	0	0
1	1	0