Day 1

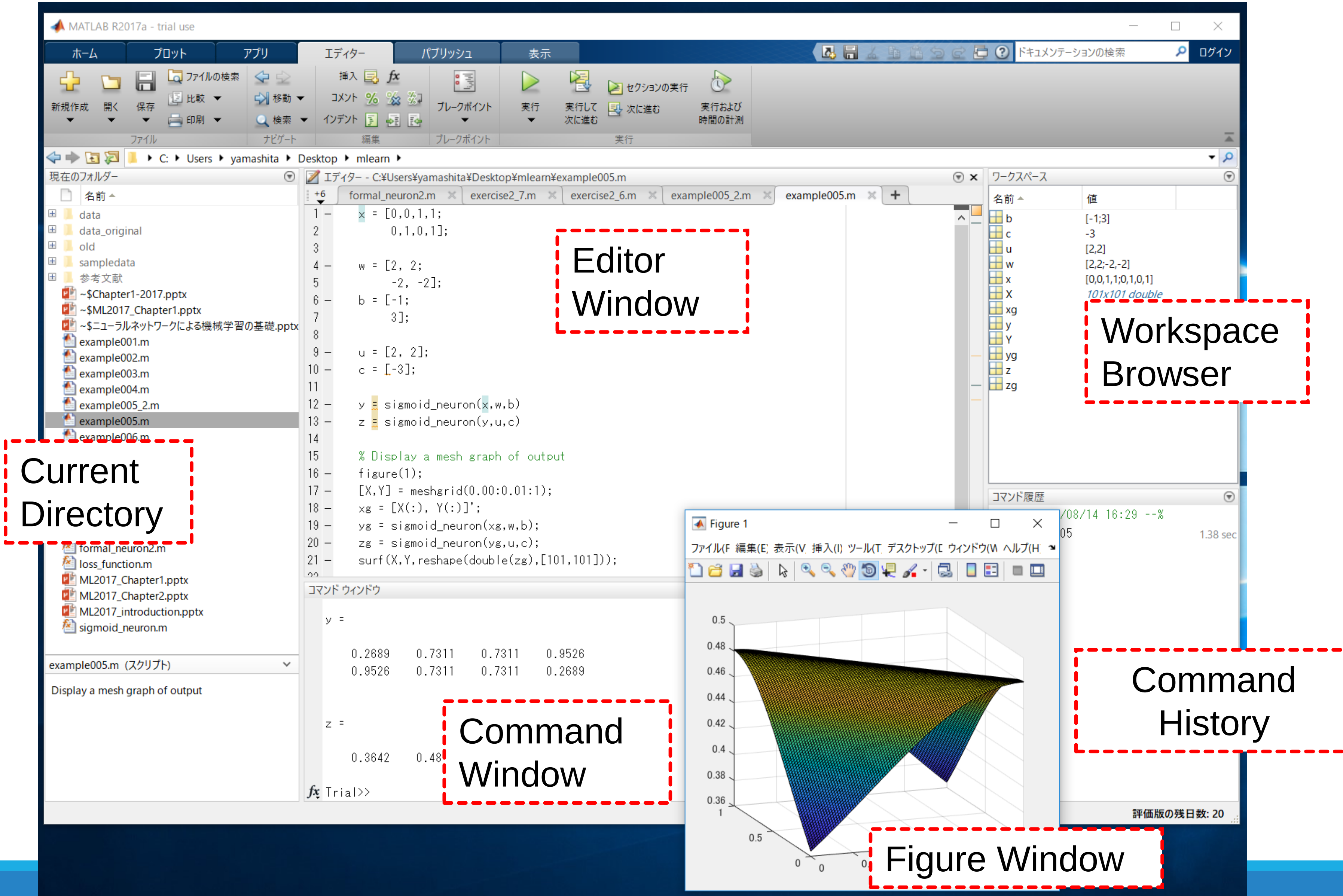
Basic Operations in MATLAB for Constructing Neural Network

MATLAB Programming

- MATLAB (MATrix LABoratory) stands for matrix (i.e., two-dimensional array) laboratory.
- MATLAB is a high-performance language for technical computing, and an array-oriented language.
- Calculations when constructing or using a neural network are almost represented by matrix operations.

We will use MATLAB in this course to implement neural network model and cultivate a better understanding with several exercises.

The MATLAB desktop and its principal components



We can use GNU Octave instead of MATLAB for FREE

MATLAB environment is very powerful and useful but EXPENSIVE.

GNU Octave https://www.gnu.org/software/octave/

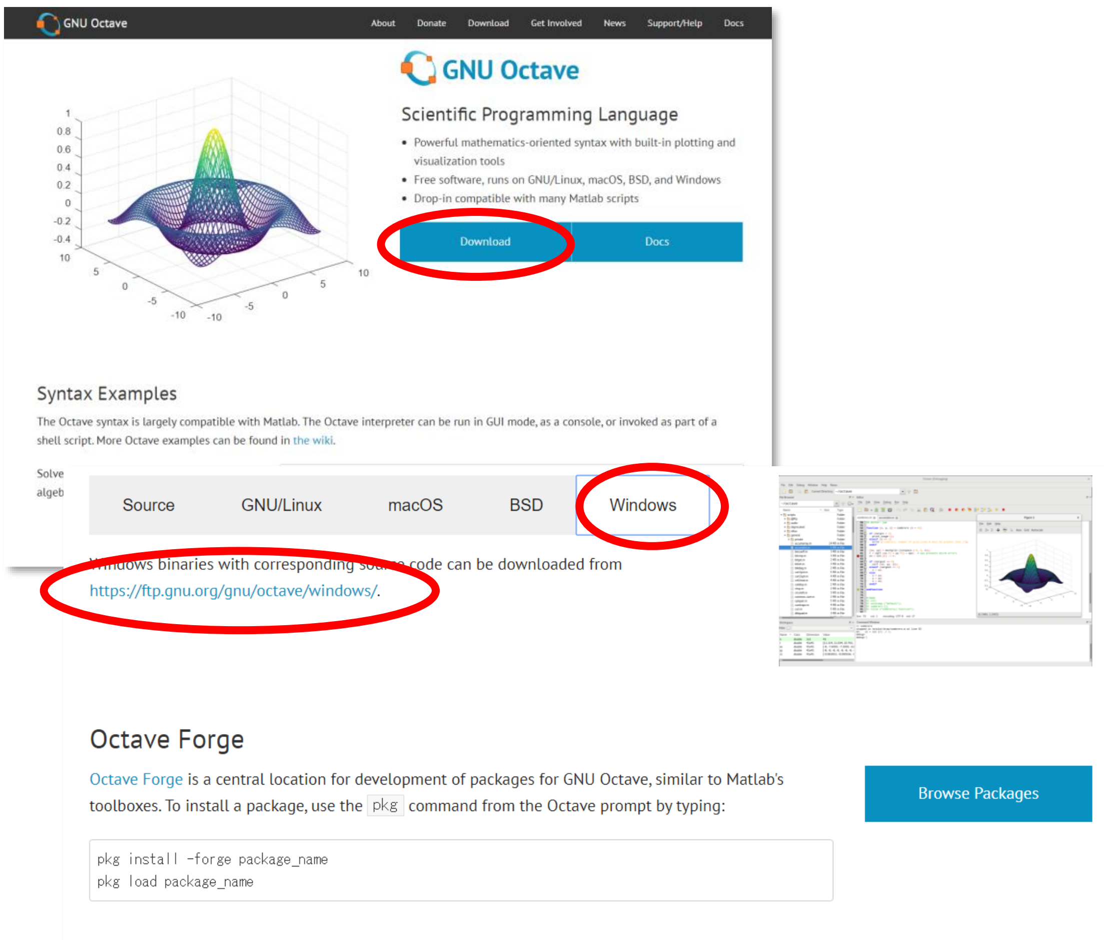
GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab. It may also be used as a batch-oriented language. Octave has extensive tools for solving common numerical linear algebra problems, finding the roots of nonlinear equations, integrating ordinary functions, manipulating polynomials, and integrating ordinary differential and differential-algebraic equations. It is easily extensible and customizable via user-defined functions written in Octave's own language, or using dynamically loaded modules written in C++, C, Fortran, or other languages. [from official website]

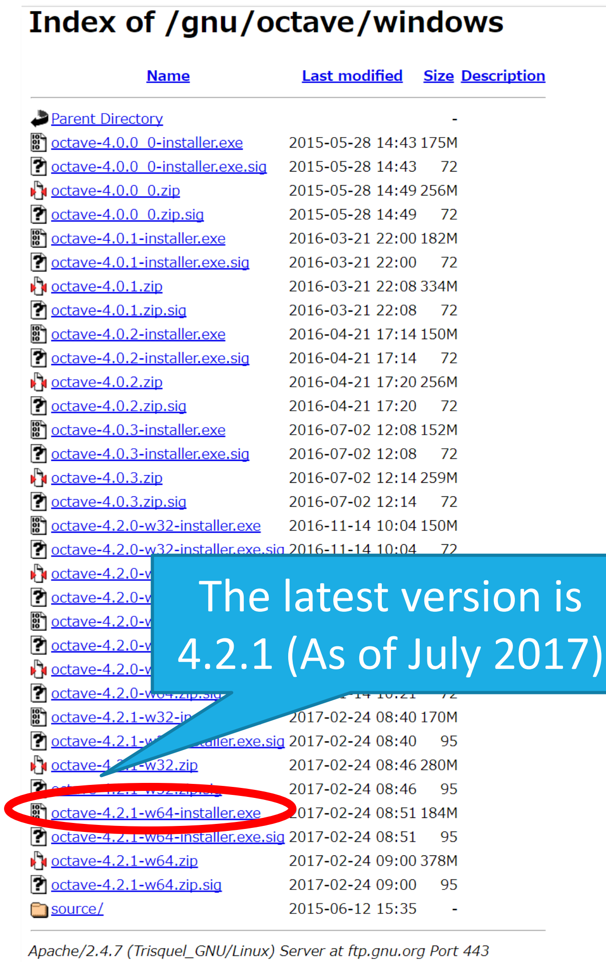
You can use octave to do exercises on your own computer.

【Appendix】How to install Octave

Download from https://www.gnu.org/software/octave/

Following installation procedure is for Windows, however we can install on MAC or Linux.





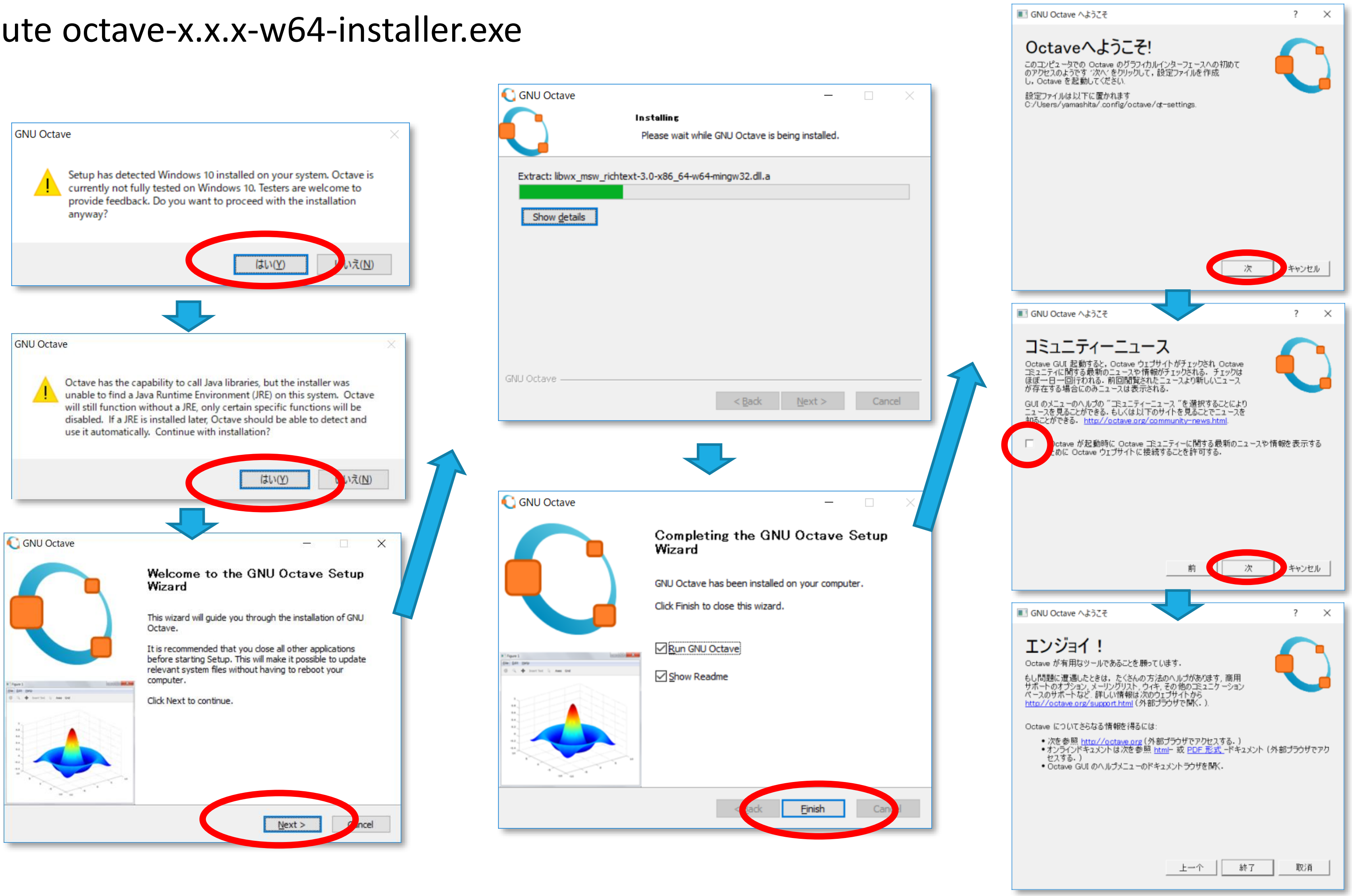
Index of /gnu/octave/windows

Name	Last modified	Size	Description
Parent Directory		-	
octave-4.0.0_0-installer.exe	2015-05-28 14:43	175M	
octave-4.0.0_0-installer.exe.sig	2015-05-28 14:43	72	
octave-4.0.0_0.zip	2015-05-28 14:49	256M	
octave-4.0.0_0.zip.sig	2015-05-28 14:49	72	
octave-4.0.1-installer.exe	2016-03-21 22:00	182M	
octave-4.0.1-installer.exe.sig	2016-03-21 22:00	72	
octave-4.0.1.zip	2016-03-21 22:08	334M	
octave-4.0.1.zip.sig	2016-03-21 22:08	72	
octave-4.0.2-installer.exe	2016-04-21 17:14	150M	
octave-4.0.2-installer.exe.sig	2016-04-21 17:14	72	
octave-4.0.2.zip	2016-04-21 17:20	256M	
octave-4.0.2.zip.sig	2016-04-21 17:20	72	
octave-4.0.3-installer.exe	2016-07-02 12:08	152M	
octave-4.0.3-installer.exe.sig	2016-07-02 12:08	72	
octave-4.0.3.zip	2016-07-02 12:14	259M	
octave-4.0.3.zip.sig	2016-07-02 12:14	72	
octave-4.2.0-w32-installer.exe	2016-11-14 10:04	150M	
octave-4.2.0-w32-installer.exe.sig	2016-11-14 10:04	72	
octave-4.2.0-v[illegible]			
octave-4.2.0-v[illegible]			
octave-4.2.0-v[illegible]			
octave-4.2.0-v[illegible]			
octave-4.2.0-v[illegible]			
octave-4.2.0-w64.zip.sig	[illegible] 14:21	72	
octave-4.2.1-w32-in[illegible]	2017-02-24 08:40	170M	
octave-4.2.1-w[illegible]aller.exe.sig	2017-02-24 08:40	95	
octave-4.2.1-w32.zip	2017-02-24 08:46	280M	
[illegible]	2017-02-24 08:46	95	
octave-4.2.1-w64-installer.exe	2017-02-24 08:51	184M	
octave-4.2.1-w64-installer.exe.sig	2017-02-24 08:51	95	
octave-4.2.1-w64.zip	2017-02-24 09:00	378M	
octave-4.2.1-w64.zip.sig	2017-02-24 09:00	95	
source/	2015-06-12 15:35	-	

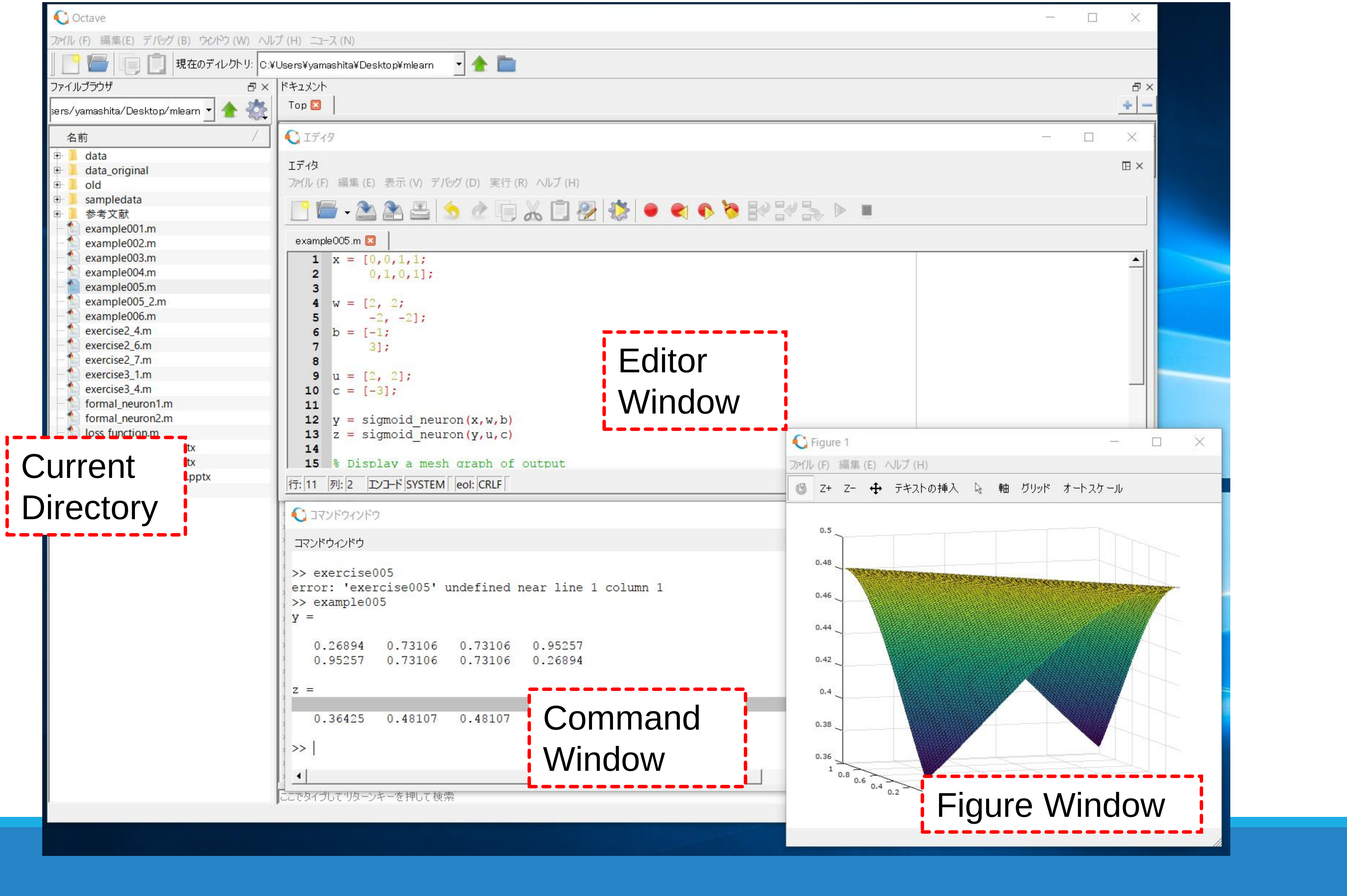
Apache/2.4.7 (Trisquel_GNU/Linux) Server at ftp.gnu.org Port 443

【Appendix】Installation procedure of Octave

Execute octave-x.x.x-w64-installer.exe



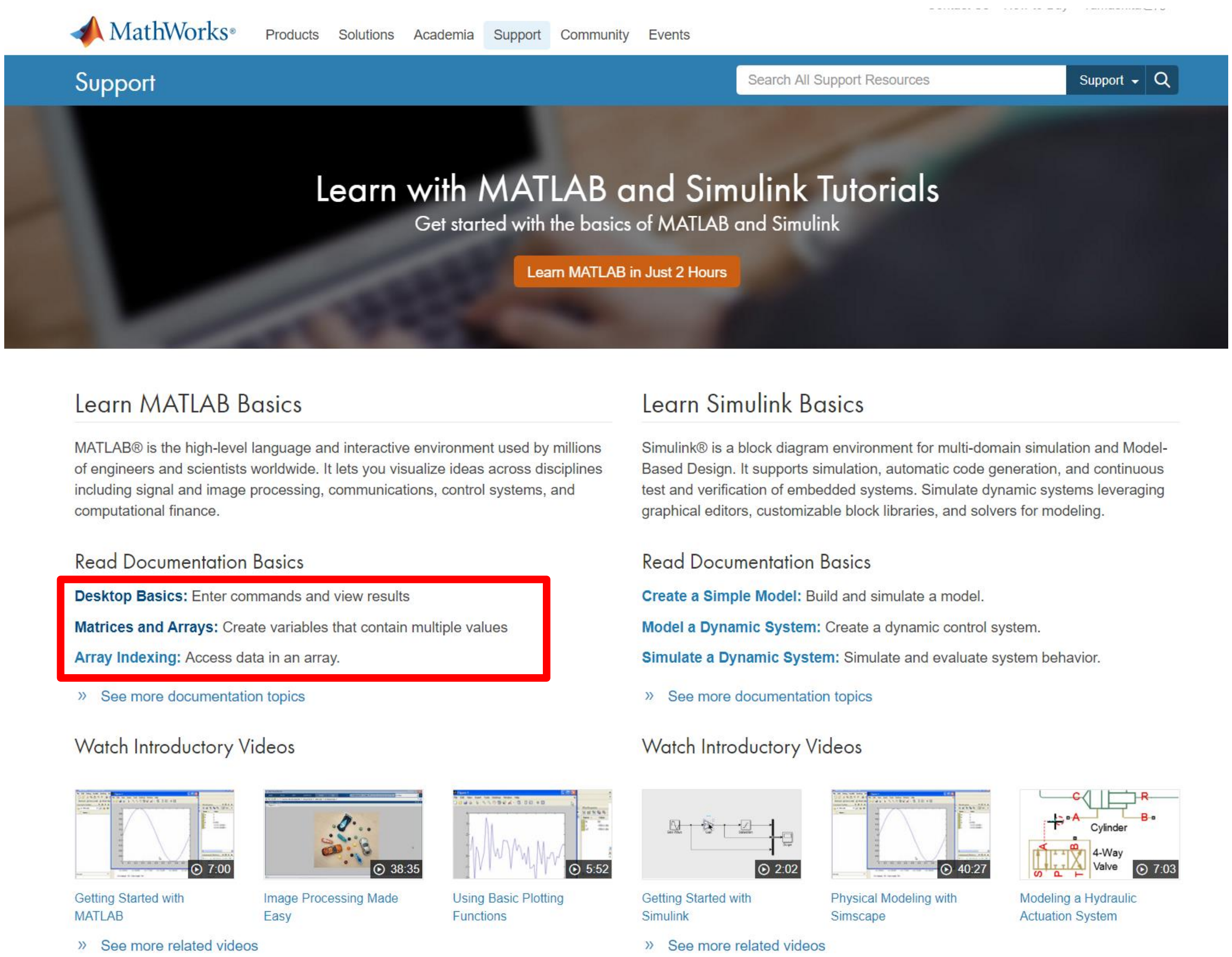
【Appendix】Octave desktop



Basic Operations in MATLAB

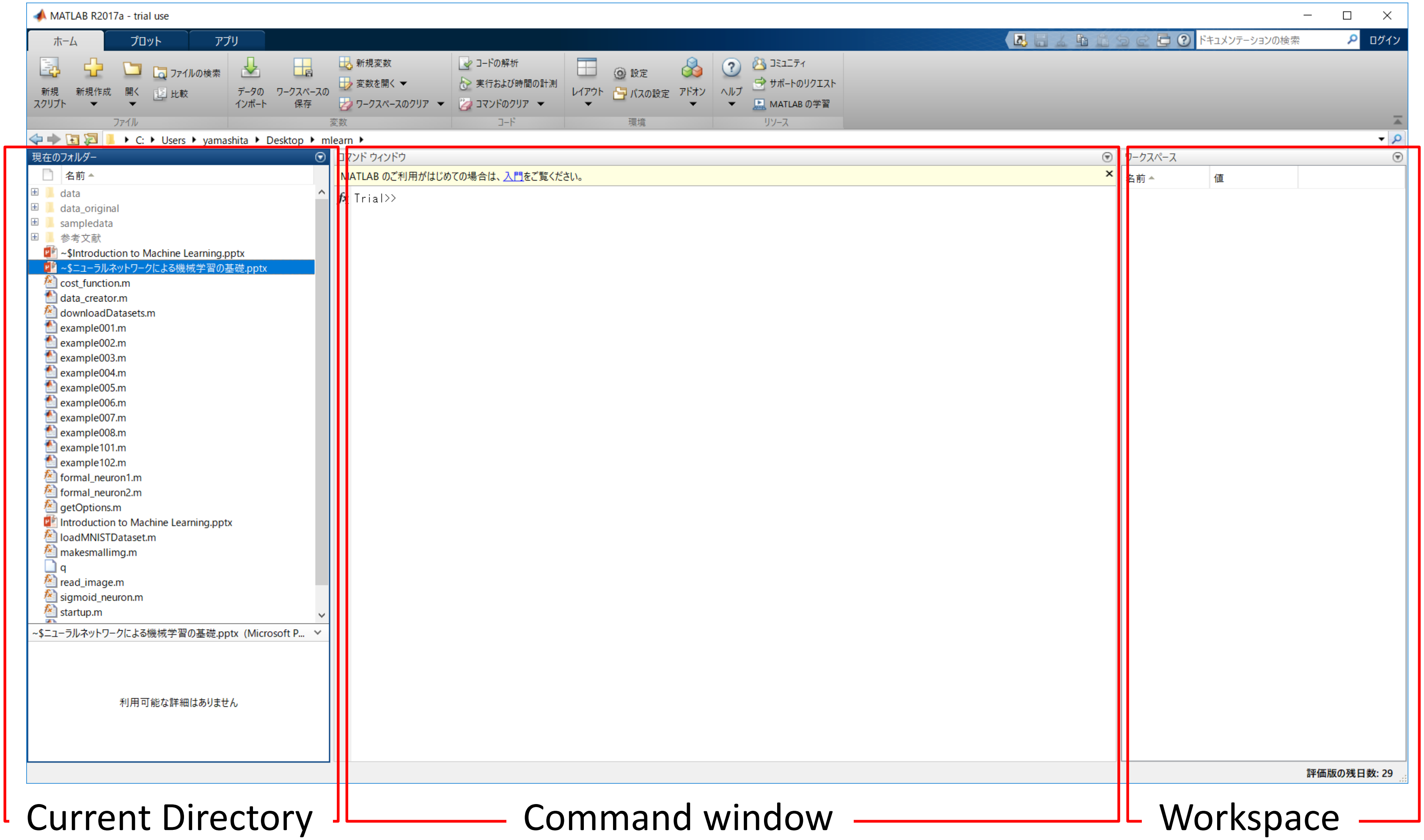
Tutorials for MATLAB in Mathworks Website

https://www.mathworks.com/support/learn-with-matlab-tutorials.html



Start MATLAB

When you start MATLAB, the desktop in its default layout.



Basic Operation in MATLAB (1)

```
>>
```

Prompt symbol designates the beginning of command line.

```
>> 2+3*4
ans =
      14
```

You can use arithmetic operators +, -, *, and / .

If you do not specify an output variable, MATLAB uses a default variable ans.

```
>> x = 2+3*4
>> x =
      14
```

You may assign a value to a variable.

```
>> 2*x
ans =
      28
```

This variable name can always be used to refer to the results of the previous computations.

```
>> y = 2*3 ;
```

The semicolon at the end is for suppressing output. If a semicolon is included, MATLAB doesn't display the results of the operation.

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.

A(2, 3)

$$A = \begin{matrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 1 & 2 & 3 \end{matrix}$$

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
```

A(end, end)
=A(3,4)

A(2:3, 2:4)

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

A(2:end, 3:4)

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

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
```

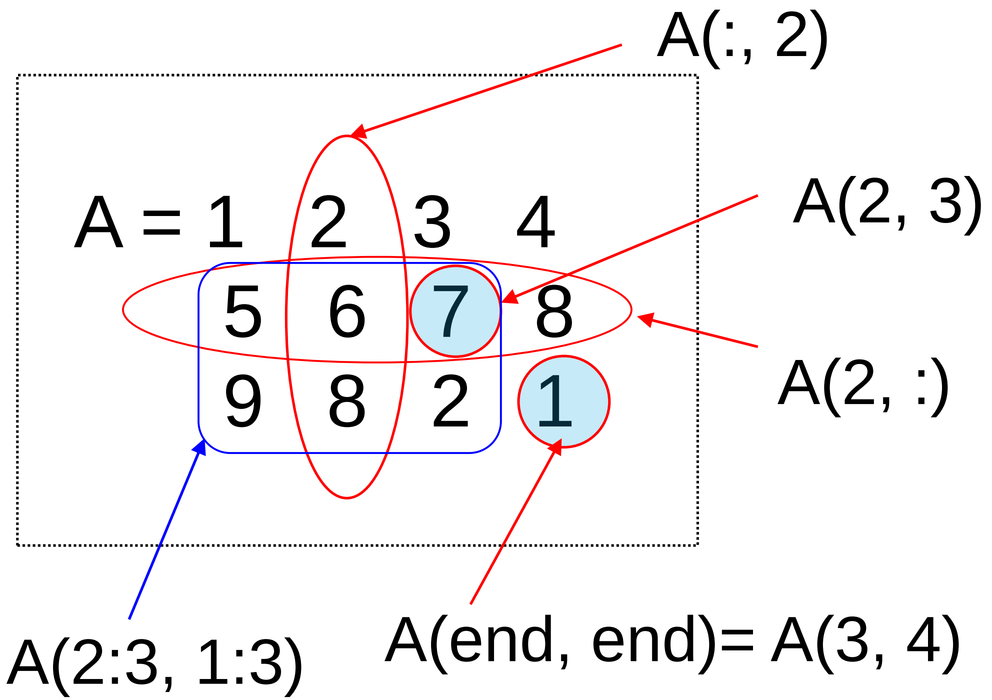
MATLAB does not display output on the screen when an operation ends with the semicolon(;).

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 (‘).

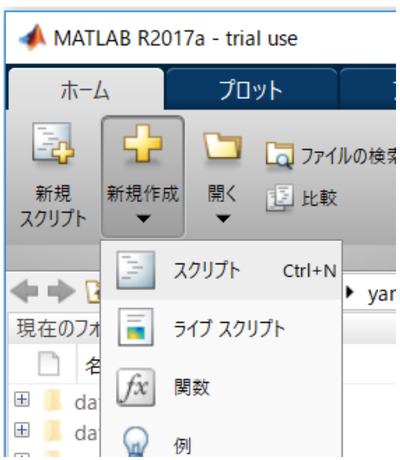
```
>> c = [1 2 3; 4 5 6; 7 8 9]
c =

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

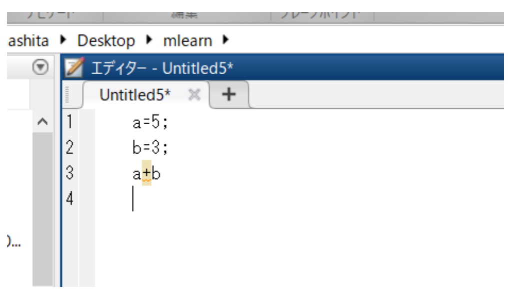
```
>> c'
ans =

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

MATLAB script and script file

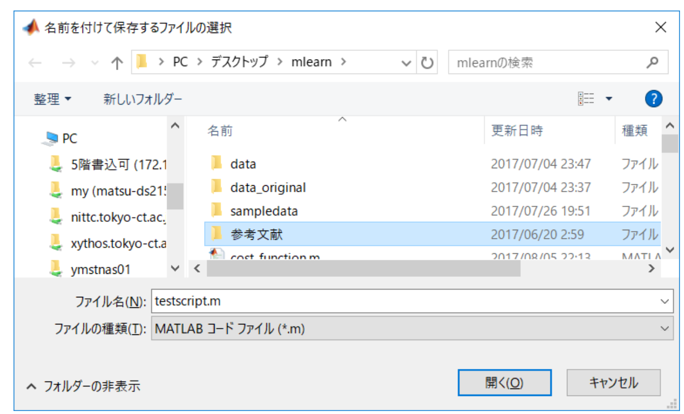
MATLAB program is called *script*. A script is a file with a ".m" extension that contains multiple sequential lines of MATLAB commands and function calls. You can run a script by typing its name at the command window.

If you want to create a new script file, press "New" button and select "Script". Then, describe a script (i.e., commands and functions) at an editor window. Finally, save as *.m file in your workfoldar.





Describe a script (commands)



Save as *.m file

Execute a script

We can execute a script by enter the script filename.

testscript.m

```
a=5;
b=3;
a+b
```

Command window

```
>> testscript

ans =
     8
```

The script filename

Definition of a original function

We can define an original function on MATLAB as follows.

testfunc.m

```
function x=testfunc(a, b)
  x = a+b;
end
```

Command window

```
>> x =testfunc(3,4)

ans =
     7
```

The function name

【NOTICE】
You have to use a same name for the function name and the filename.

Operations for vectors and matrices(1)

You can add scalar to vector or add two vectors

```
>> a = [1,2,3,4,5];
>> b = 2;
>> a+b

ans =
    3    4    5    6    7
```

Add scalar to vector

```
>> a = [1,2,3,4,5];
>> b = [1,3,5,7,9];
>> a+b

ans =
    2    5    8   11   14
```

Add two vectors

Similarly, you can add scalar to array (matrix) or add two arrays (matrices)

```
>> a = [1,2,3;4,5,6;7,8,9];
>> b = 3;
>> a+b

ans =

    4    5    6
    7    8    9
   10   11   12
```

Add scalar to array

```
>> a = [1,2,3;4,5,6;7,8,9];
>> b = [1,3,5;7,9,2;4,6,8];
>> a+b

ans =

    2    5    8
   11   14    8
   11   14   17
```

Add two arrays

Operations for vectors and matrices(2)

You can calculate a product of scalar and vector.

```
>> a = [1,2,3,4,5];
>> b = 2;
>> a*b

ans =
    2    4    6    8    10
```

A product of scalar and vector

You can calculate a inner product of two vectors.

```
>> a = [1,2,3,4,5];
>> b = [1,3,5,7,9];
>> a*b'

ans =
   95
```

Inner product $a * b'$ means a sum of products of all elements of vector a and b (i.e., $\sum_j a_j\, b_j$).
In this case,
1*1 + 2*3 + 3*5 + 4*7 + 5*9 = 95

Note that the dimensions of two vectors a and b need to coincide.

Operations for vectors and matrices(2)

You can calculate a product of scalar and matrix.

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

ans =
    2     4     6
    8    10    12
   14    16    18
```

A product of scalar and matrix

You can calculate a product of two matrices.

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

ans =
   27    39    33
   63    93    78
   99   147   123
```

Each element $x_{i,j}$ of the result is calculated as an inner product of row i of A and column j of B.

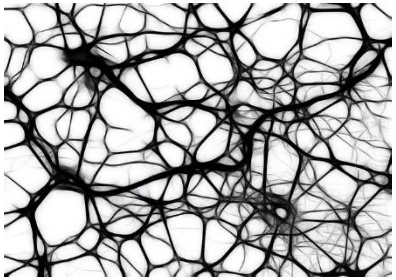
1*1+2*7+3*4=27

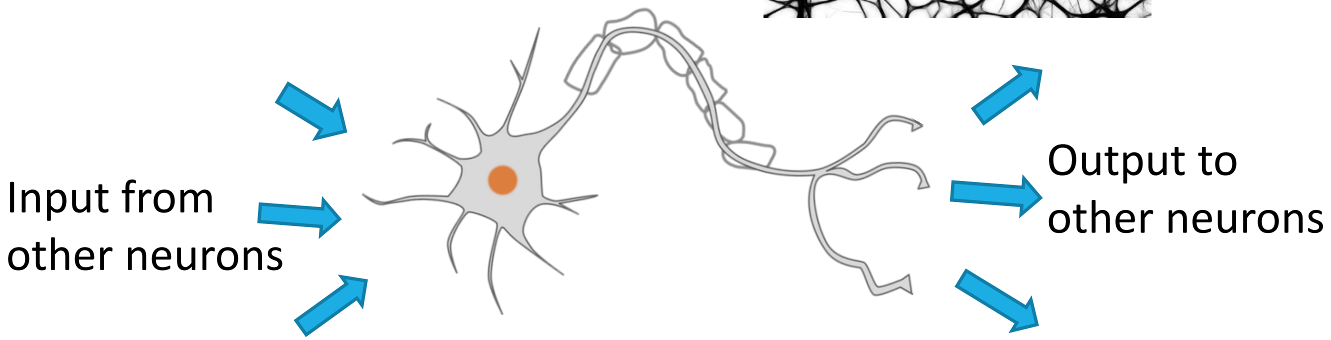
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \cdot \begin{pmatrix} 1 & 3 & 5 \\ 7 & 9 & 2 \\ 4 & 6 & 8 \end{pmatrix} = \begin{pmatrix} \boxed{27} & 39 & 33 \\ 63 & 93 & 78 \\ 99 & 147 & 123 \end{pmatrix}$$

Perceptron

Formal Neuron (McCulloch-Pitts Model)

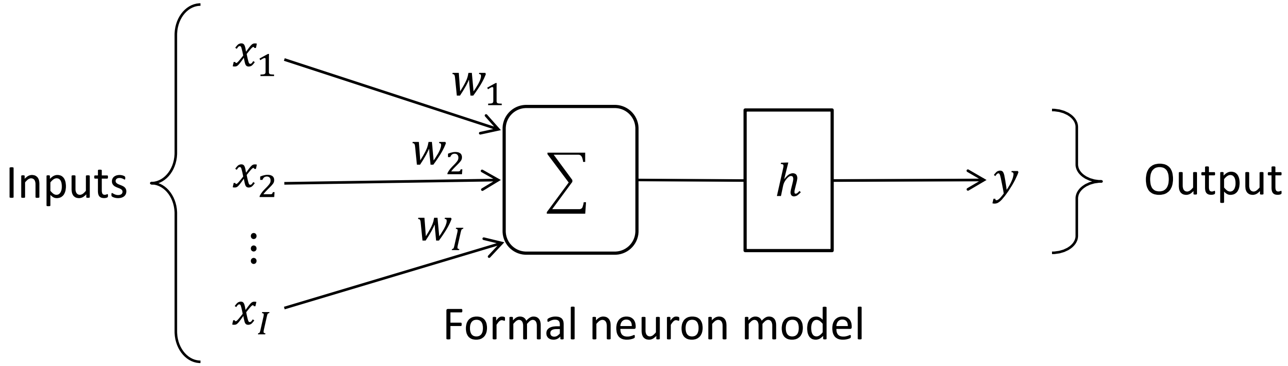
Formal neuron model was proposed by Warren McCulloch and Walter Pitts in 1943.
A formal neuron is a simplified mathematical function obtained from a simplification of a biological neuron.





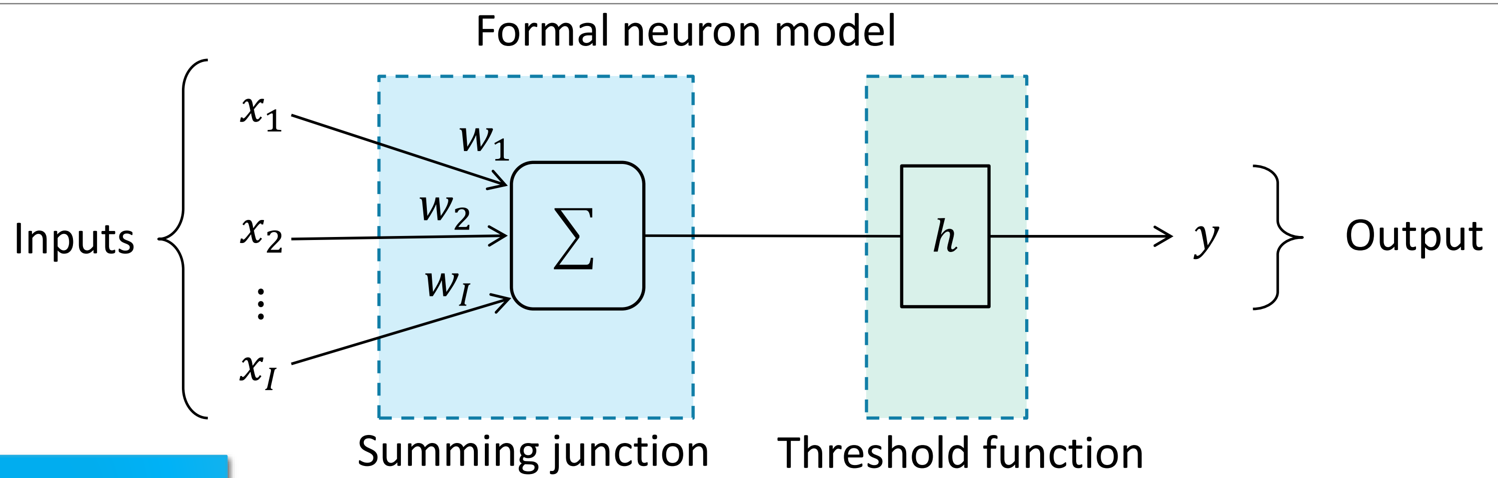
Conceptual figure of biological neuron

A formal neuron obtains several binary values $x_1, x_2, \cdots, x_I$ as input and send out one binary value y as output.



Formal neuron model

Formal Neuron (McCulloch-Pitts Model)



Summing junction

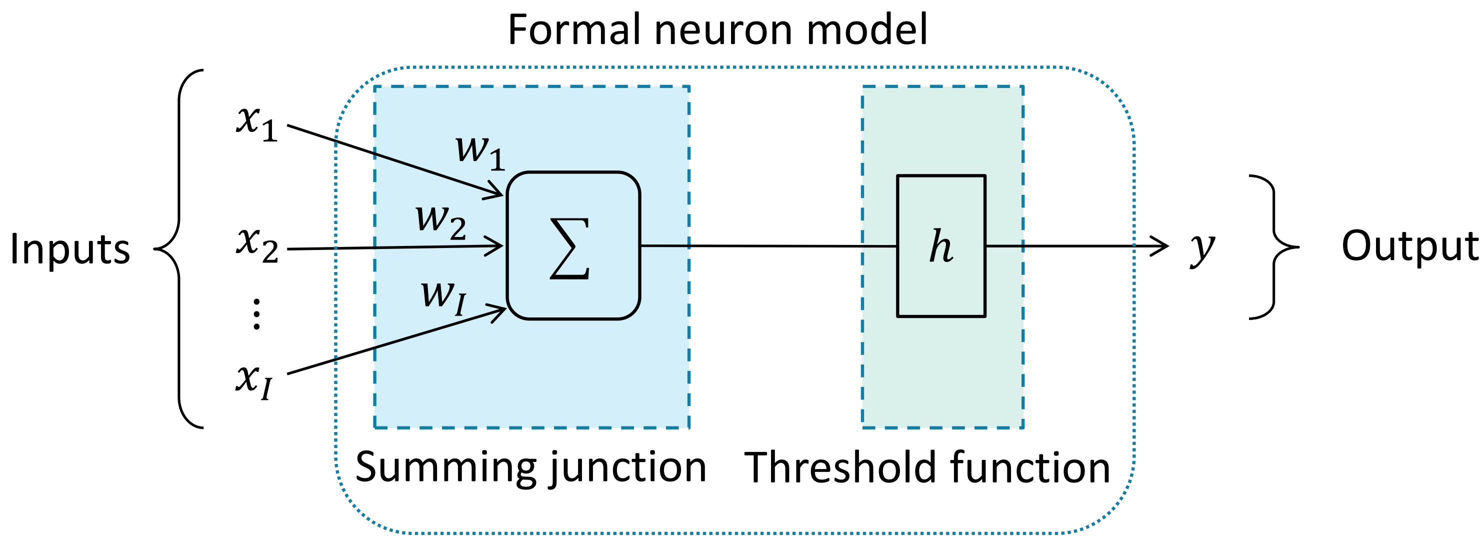
Each inputs $x_1, x_2, \cdots, x_I$ are multiplied by its own weight $w_1, w_2, \cdots, w_I$ respectively. Then a weighted sum value of them (I.e., $\sum_i w_i x_i$) is calculated at summing junction.

Threshold function

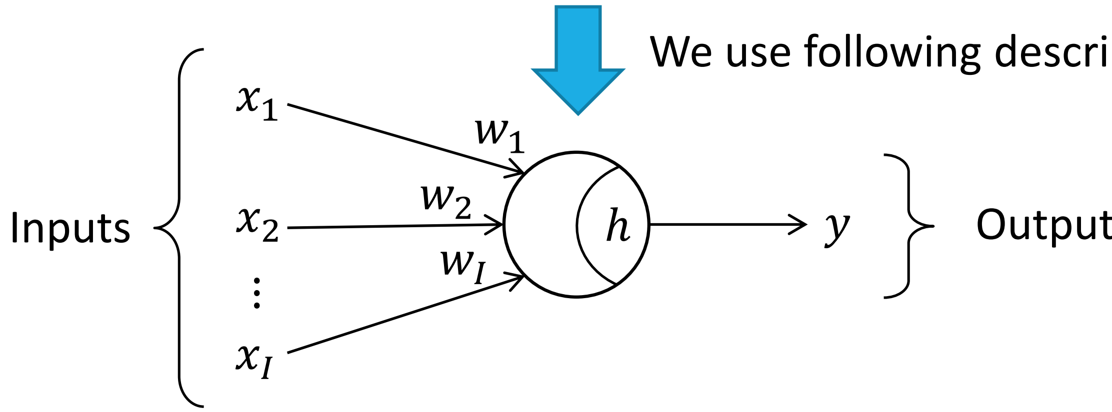
If the weighted sum value is greater than a threshold h, the output y becomes 1, if not the output y becomes 0. That is,

$$y = \begin{cases} 0 & \text{if } \sum_i w_i x_i \leq h \\ 1 & \text{if } \sum_j w_i x_i > h \end{cases}$$

Formal Neuron (McCulloch-Pitts Model)



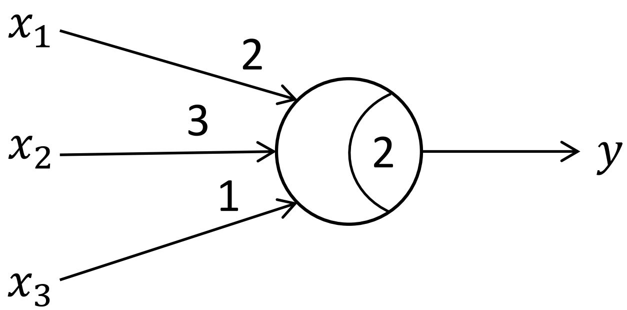
We use following description



In other words, the neuron will fire when the weighted sum value of inputs is greater than a threshold h.

Exercise 1.1

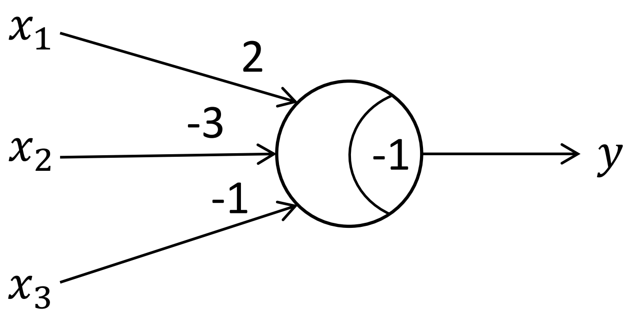
For the following neuron, calculate the weighted sum values $\sum_i w_i x_i$ and outputs y for each input in the table.



x_1	x_2	x_3	$\sum_i w_i x_i$	y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Exercise 1.2

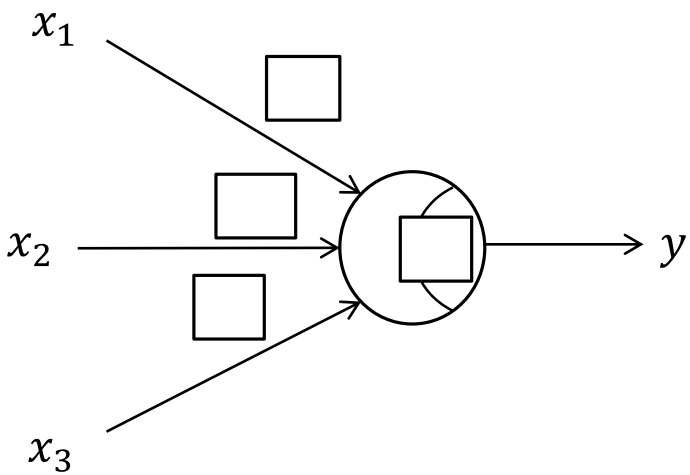
For the following neuron, calculate the weighted sum values $\sum_i w_i x_i$ and outputs y for each input in the table.



x_1	x_2	x_3	$\sum_i w_i x_i$	y
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Exercise 1.3

For the following neuron, determine weights and thresholds so that the outputs y for each of inputs are as shown in the table.



Can you find these values?

x_1	x_2	x_3	$\sum_j w_j x_j$	y
0	0	0		0
0	0	1		1
0	1	0		0
0	1	1		0
1	0	0		1
1	0	1		1
1	1	0		1
1	1	1		1

Weighted sum operation and inner-product

When input x_i and weight w_i are defined as input vector $\boldsymbol{x}$ and weight vector $\boldsymbol{w}$ respectively, weighted sum operation is calculated as inner product of w and x^t.

$$\sum_i w_i x_i = [w_1 \quad w_2 \quad \cdots \quad w_I] \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_I \end{bmatrix} = \boldsymbol{w} \cdot \boldsymbol{x}^t$$

It is easy to calculate on MATLAB.

For example,

```
>> x=[1 0 1];
>> w=[2 3 -1];
>> w*x'

ans =
    1
```

```
>> x=[1; 0; 1];
>> w=[2 3 -1];
>> w*x

ans =
    1
```

Threshold calculation

If a is greater than b, the result of MATLAB operation “a > b” is 1 (True), otherwise the result is 0 (False).

```
>> a=3;
>> b=2;
>> a>b

ans =
  logical
   1
```

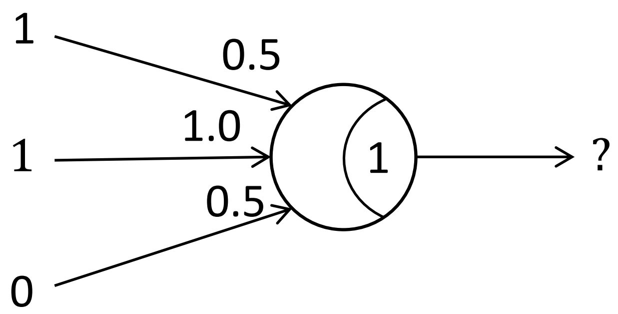
```
>> a=2;
>> b=3;
>> a>b

ans =
  logical
   0
```

Let’s implement formal neuron function on MATLAB (1)

Test script

example1_1.m

```
x = [1;
     1;
     0];
w = [0.5, 1.0, 0.5];
h = 1;

y = formal_neuron(x,w,h)
```



A function for formal neuron calculation

Create formal_neuron.m as a function file as follows.

formal_neuron.m

```
function y = formal_neuron(x,w,h)
  p = w*x;
  y = p>h;
end
```

Input x and w are vectors in this function. The weighted sum is calculated as

$$p = [w_1, w_2, w_3] * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Note w is a row vector.
x is a column vector.
p is a scalar.

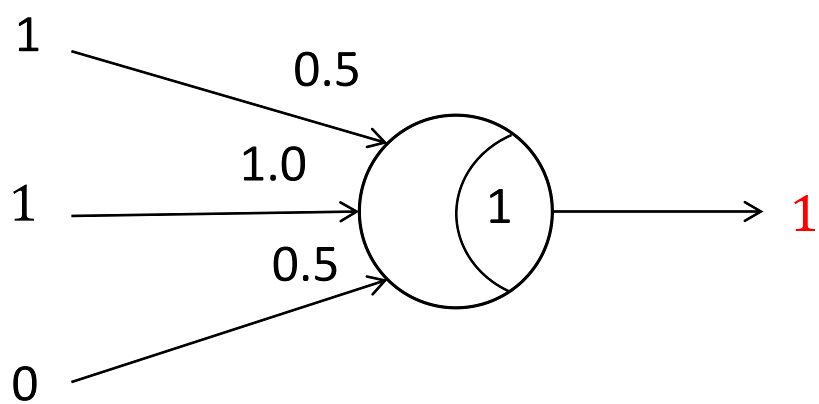
Results

Execution and results

```
>> example1_1

y =
  logical
  1
```

$$[0.5\ 1.0\ 0.5]\cdot\begin{bmatrix}1\\1\\0\end{bmatrix} = 1.5 \Rightarrow 1$$

> 1 (Threshold)



Exercise1.4

Implement example1_1.m and formal_neuron.m on MATLAB.

Check the outputs where inputs, weights and a threshold are given as follows by both hand calculation and MATLAB scripts execution.

x_1	x_2	x_3
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

$$w = [2, -1, 3]$$

$$h = 1$$

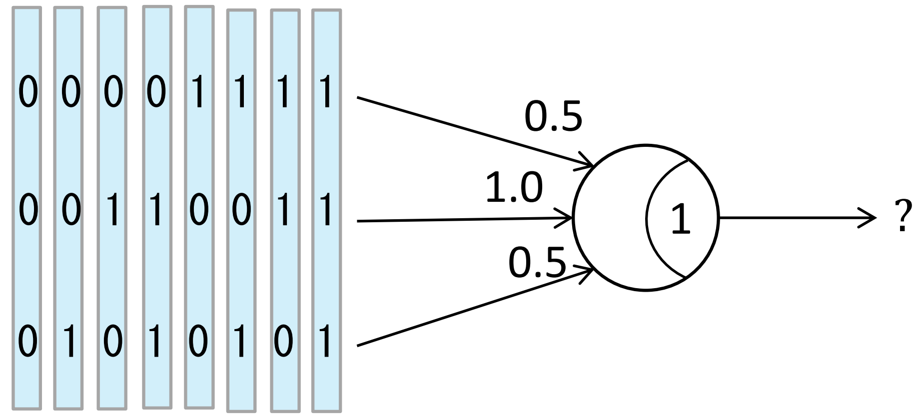
Let's implement formal neuron function on MATLAB (2)

If we use multiple sample data, we prepare it in an input matrix.

Test script

example1_2.m

```
x = [0, 0, 0, 0, 1, 1, 1, 1;
     0, 0, 1, 1, 0, 0, 1, 1;
     0, 1, 0, 1, 0, 1, 0, 1];

w = [0.5,  1.0,  0.5];
h = 1;

y = formal_neuron(x, w, h)
```



Formal neuron function

We can use a same function!

formal_neuron.m

```
function y = formal_neuron(x, w, h)
  p = w*x;
  y = p>h;
end
```

$$[p_1,\cdots,p_N] = [w_1,w_2,w_3] * \overbrace{\begin{bmatrix} x_{1,1}, x_{1,2}, \cdots, x_{1,N} \\ x_{2,1}, x_{2,2}, \cdots, x_{2,N} \\ x_{3,1}, x_{3,2}, \cdots, x_{3,N} \end{bmatrix}}^{N\ (\text{number of sample data})} \left.\vphantom{\begin{bmatrix} a \\ b \\ c \end{bmatrix}}\right\} 3$$

Input Matrix

Note

w is a row vector.
x is a 3*N matrix.
p is a row vector.

Results

Execution and results

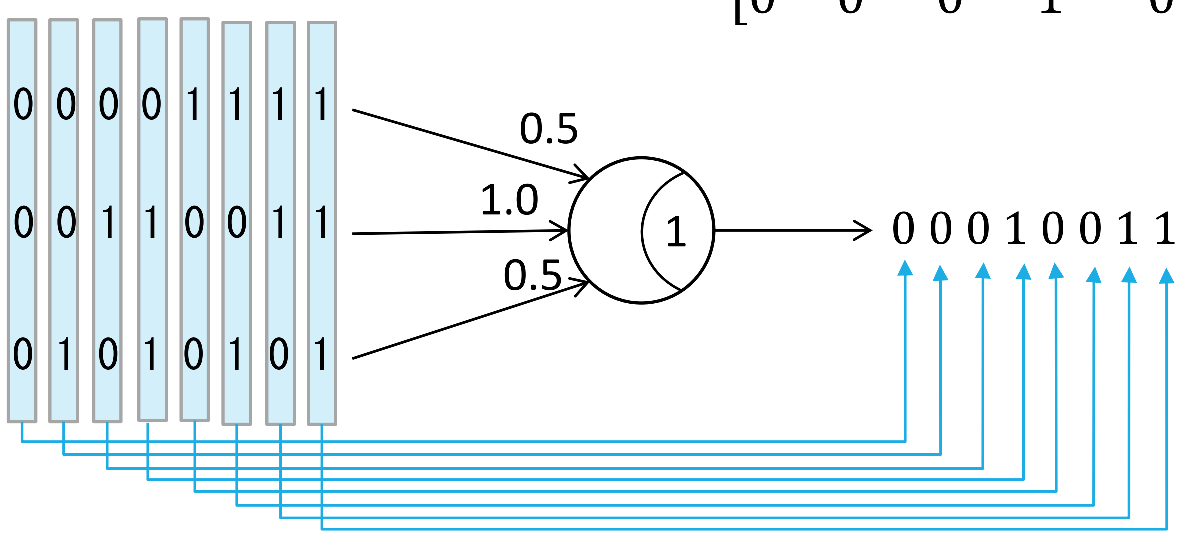
```
>> example1_2
y =

   0   0   0   1   0   0   1   1
```

$$[0.5\ 1.0\ 0.5]\cdot\begin{bmatrix}0 & 0 & 0 & 0 & 1 & 1 & 1 & 1\\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1\\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1\end{bmatrix}$$

$$= [0\quad 0.5\quad 1.0\quad 1.5\quad 0.5\quad 1.0\quad 1.5\quad 2.0]$$

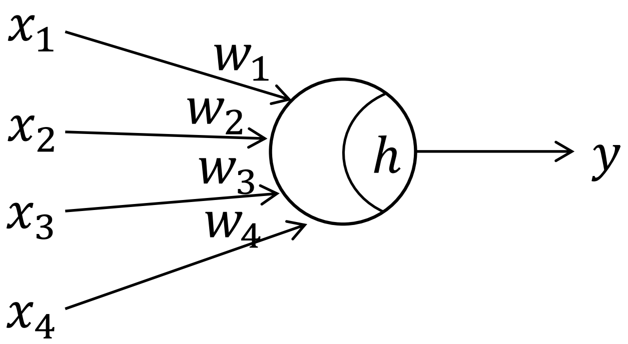
> 1 (Threshold)

$$[0\quad 0\quad 0\quad 1\quad 0\quad 0\quad 1\quad 1]$$



Exercise1.5

Consider a 4 input neuron (the input dimension is 4) as follows.



Calculate outputs by hand calculation where inputs, weights and a threshold are given as follows. Then check the answer using MATLAB.

There are 3 samples.

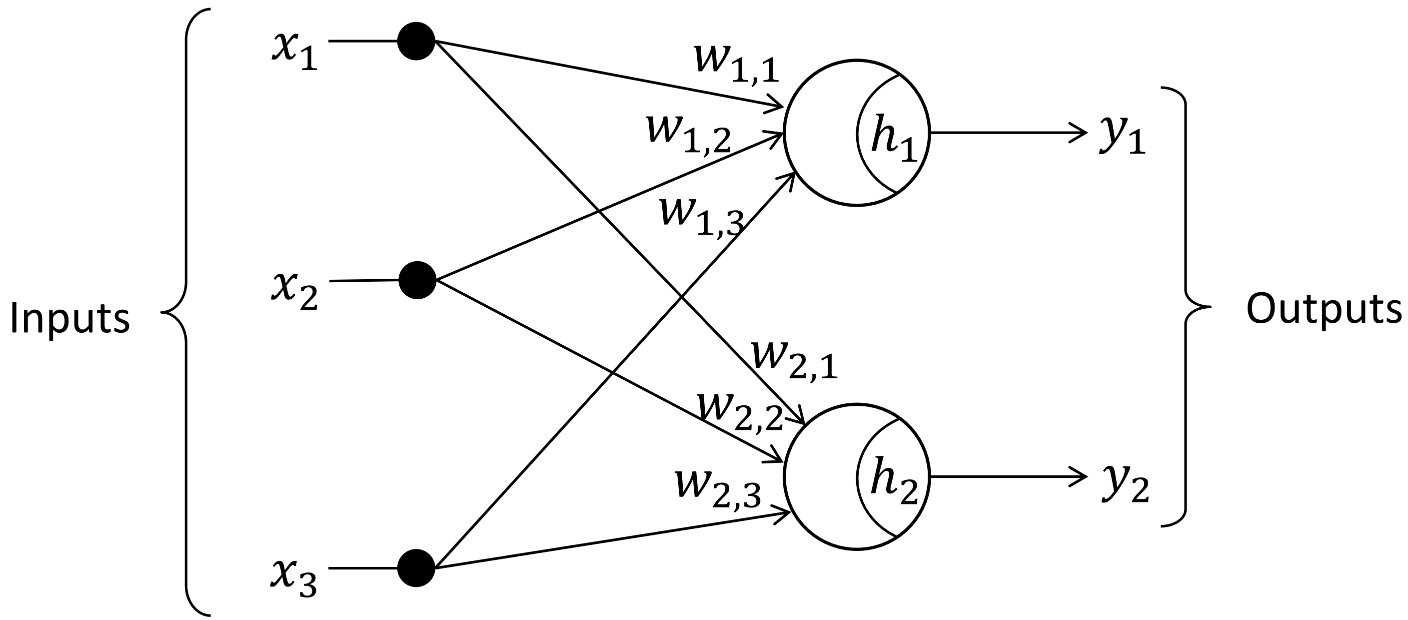
$$\boldsymbol{x} = \left.\overbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{bmatrix}}\right\}$$ Input dimension is 4.

$$\boldsymbol{w} = [0.5 \;\; 1 \;\; 1.5 \;\; 0]$$

$$h = 1.5$$

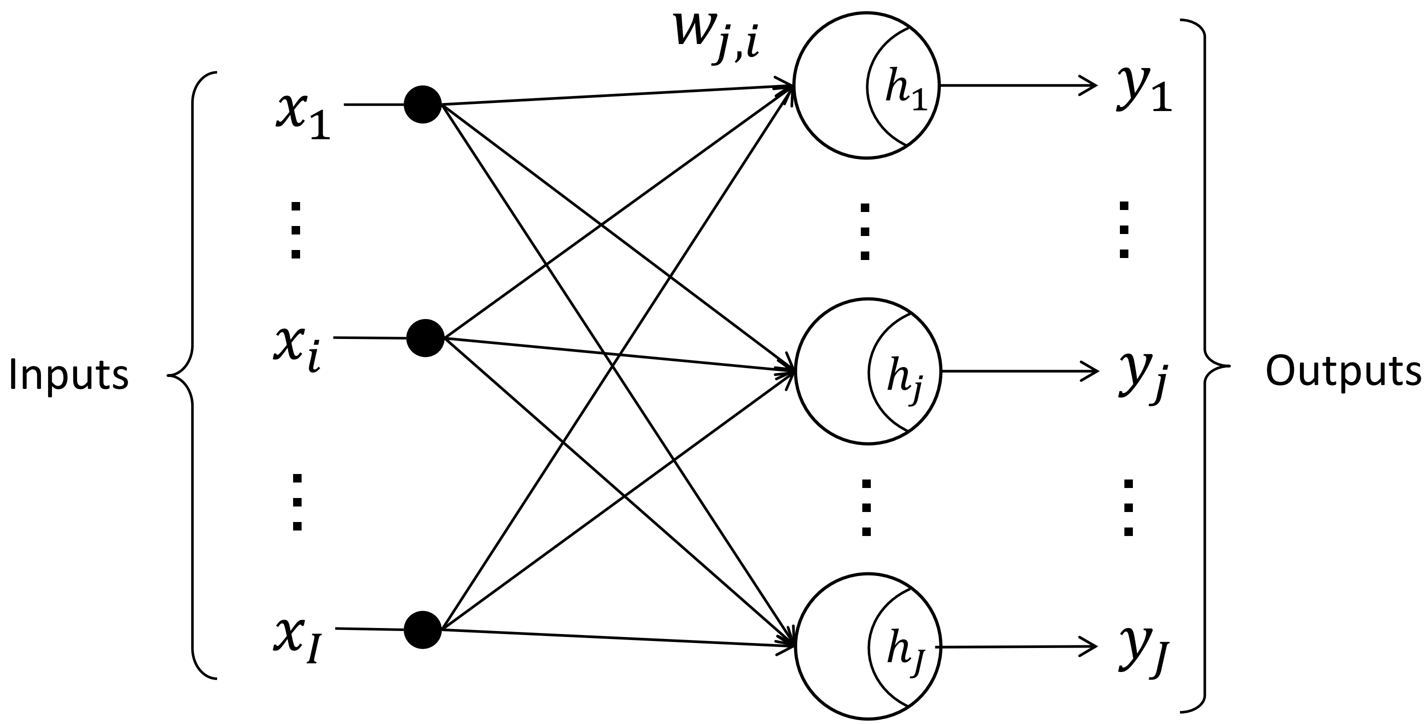
Using multiple neurons

We can construct a neural network with two or more neurons as follows,



Using multiple neurons

Generally,



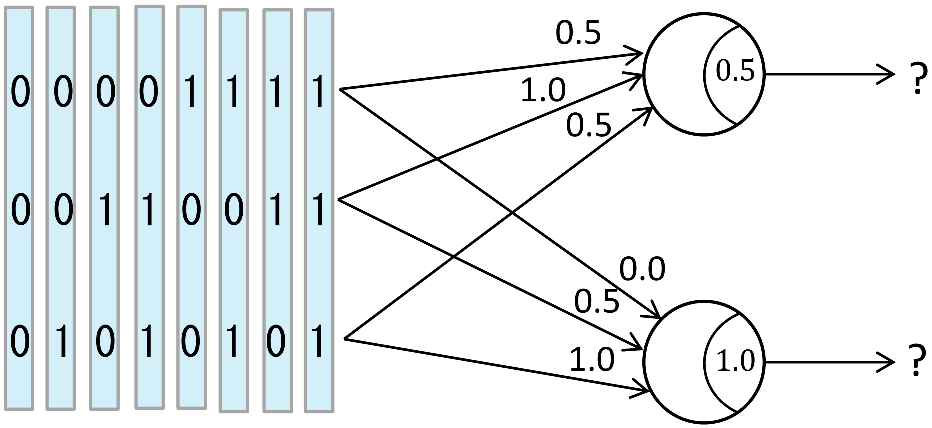
Let’s implement formal neuron function on MATLAB (3)

If there are multiple neuron, the outputs are calculated as follows.

Test script

example1_3.m

```
x = [0, 0, 0, 0, 1, 1, 1, 1;...
     0, 0, 1, 1, 0, 0, 1, 1;...
     0, 1, 0, 1, 0, 1, 0, 1];
w = [0.5,  1.0,  0.5;...
     0.0,  0.5,  1.0];
h = [0.5;...
     1.0];

y = formal_neuron(x, w, h)
```



Formal neuron function

We can use a same function again this time!

formal_neuron.m

```
function y = formal_neuron(x, w, h)
  p = w*x;
  y = p>h;
end
```

$$\begin{bmatrix} p_{1,1},\cdots,p_{1,N} \\ p_{2,1},\cdots,p_{2,N} \end{bmatrix} = \begin{bmatrix} w_{1,1},w_{1,2},w_{1,3} \\ w_{2,1},w_{2,2},w_{2,3} \end{bmatrix} * \left.\overbrace{\begin{bmatrix} x_{1,1},x_{1,2},\cdots,x_{1,N} \\ x_{2,1},x_{2,2},\cdots,x_{2,N} \\ x_{3,1},x_{3,2},\cdots,x_{3,N} \end{bmatrix}}^{N\text{ (number of sample data)}}\right\}3$$

Input Matrix

Note

w is a 2*3 matrix.
x is a 3*N matrix.
p is a 2*N matrix.

Results

Execution and results

```
>> example1_3
y =

  0   0   1   1   0   1   1   1
  0   0   0   1   0   0   0   1
```

$$[0.5\ 1.0\ 0.5]\cdot\begin{bmatrix}0&0&0&0&1&1&1&1\\0&0&1&1&0&0&1&1\\0&1&0&1&0&1&0&1\end{bmatrix}$$

$$=[0\quad 0.5\quad 1.0\quad 1.5\quad 0.5\quad 1.0\quad 1.5\quad 2.0]$$

>0.5 (Threshold)

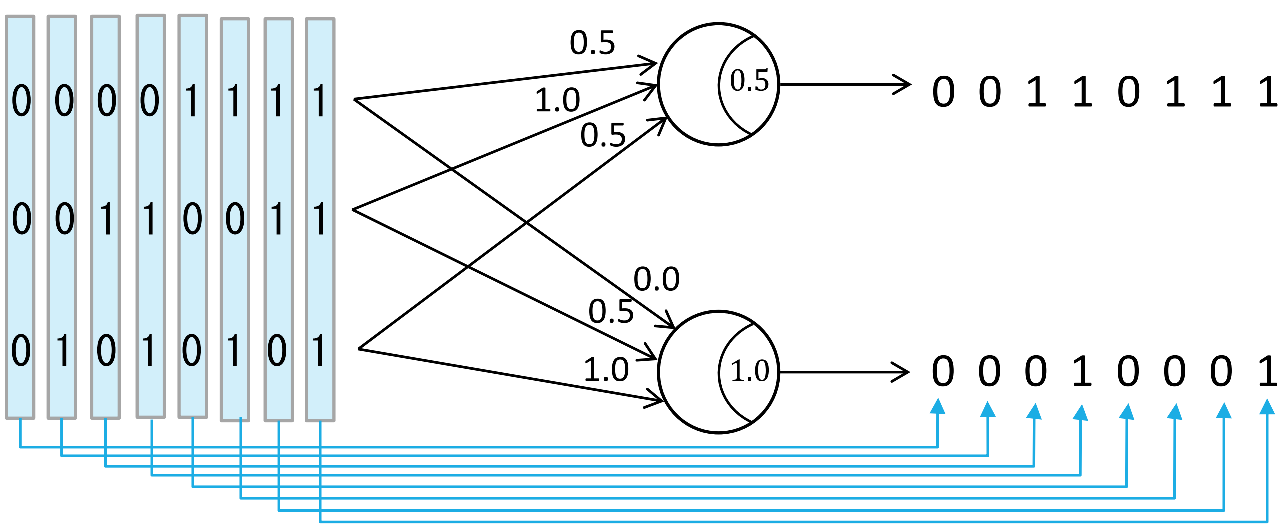
$$[0\quad 0\quad 1\quad 1\quad 0\quad 1\quad 1\quad 1]$$

$$[0.0\ 0.5\ 1.0]\cdot\begin{bmatrix}0&0&0&0&1&1&1&1\\0&0&1&1&0&0&1&1\\0&1&0&1&0&1&0&1\end{bmatrix}$$

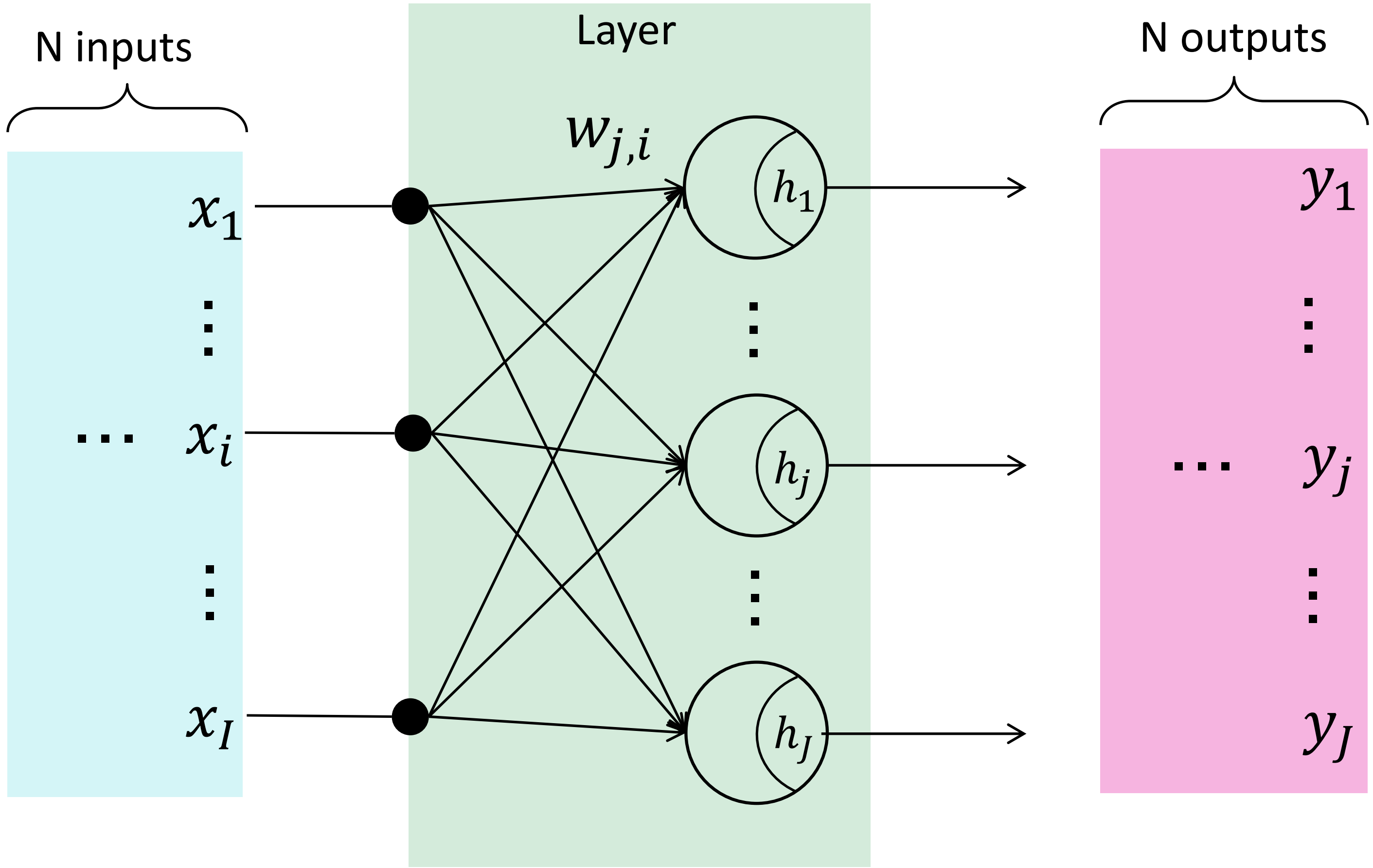
$$=[0\quad 1.0\quad 0.5\quad 1.5\quad 0.0\quad 1.0\quad 0.5\quad 1.5]$$

>1.0 (Threshold)

$$[0\quad 0\quad 0\quad 1\quad 0\quad 0\quad 0\quad 1]$$

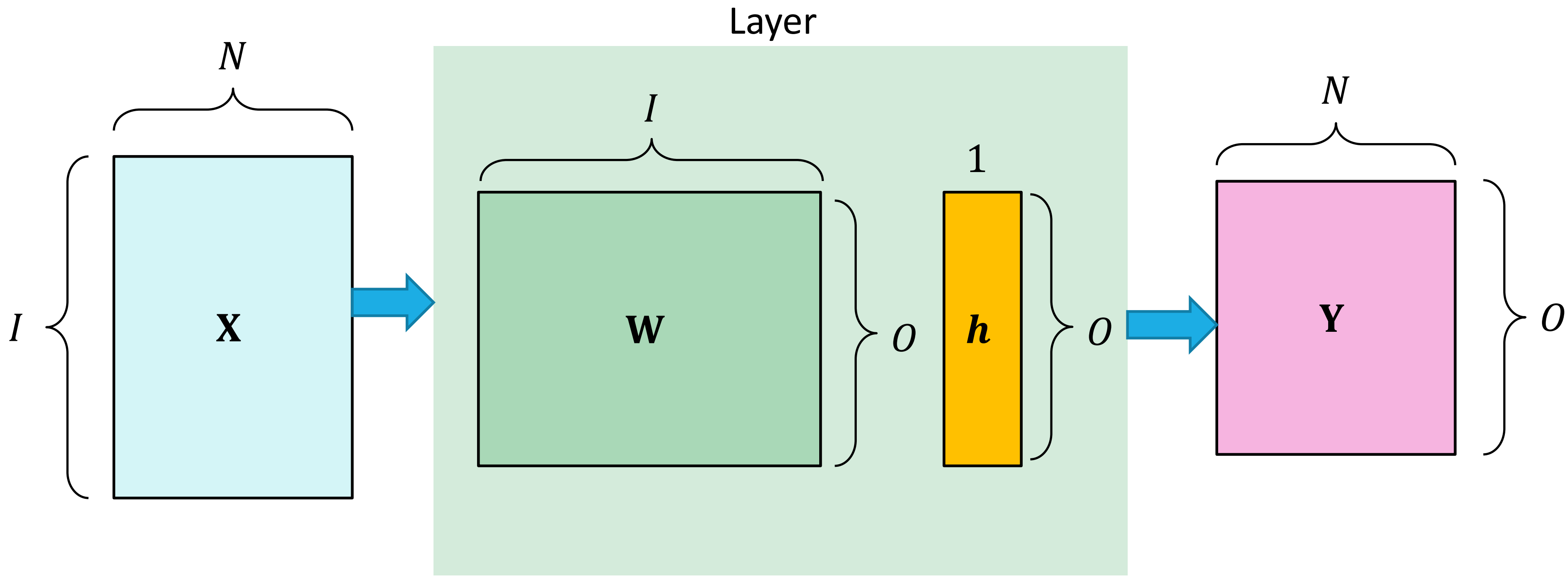


Layers in Neural Network Model



Layers are made up of a number of interconnected neurons.

Layers in Neural Network Model



Implementation using FormalNeuronClass

example1_4.m

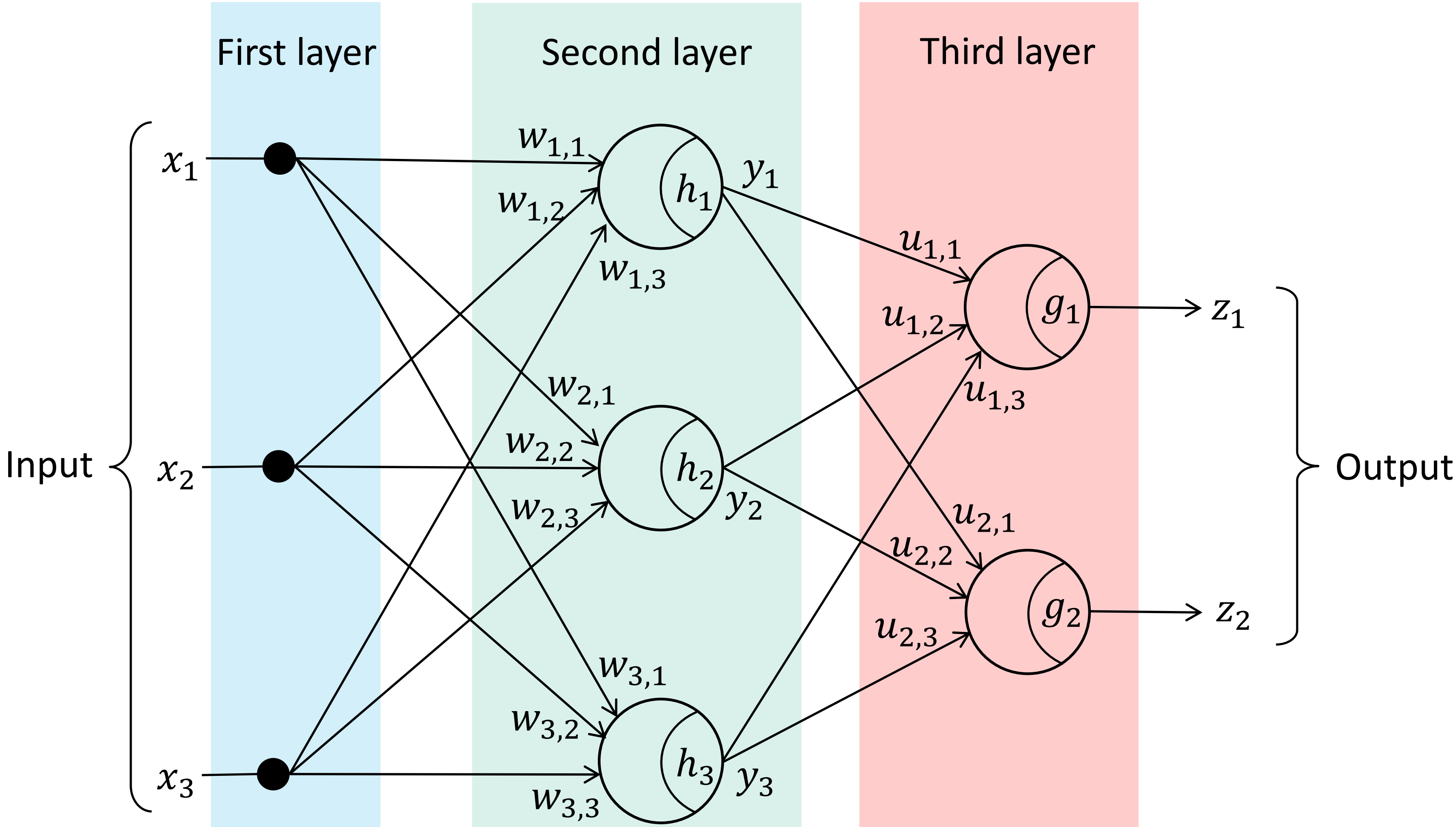
```
x = [0,0,0,0,1,1,1,1;
     0,0,1,1,0,0,1,1;
     0,1,0,1,0,1,0,1];

w = [0.5,  1.0,  0.5;
     0.0,  0.5,  1.0];
h = [0.5;...
     1.0];

layer1 = FormalNeuronLayer(w,h);
y = layer1.forward(x)
```

FormalNeuronLayer.m

```
classdef FormalNeuronLayer
  properties
    weights;
    threshold;
  end

  methods
    function obj = FormalNeuronLayer(w,h)
      obj.weights = w;
      obj.threshold = h;
    end

    function y = forward(obj, x)
      p = obj.weights * x;
      y = p > obj.threshold;
    end
  end
end
```

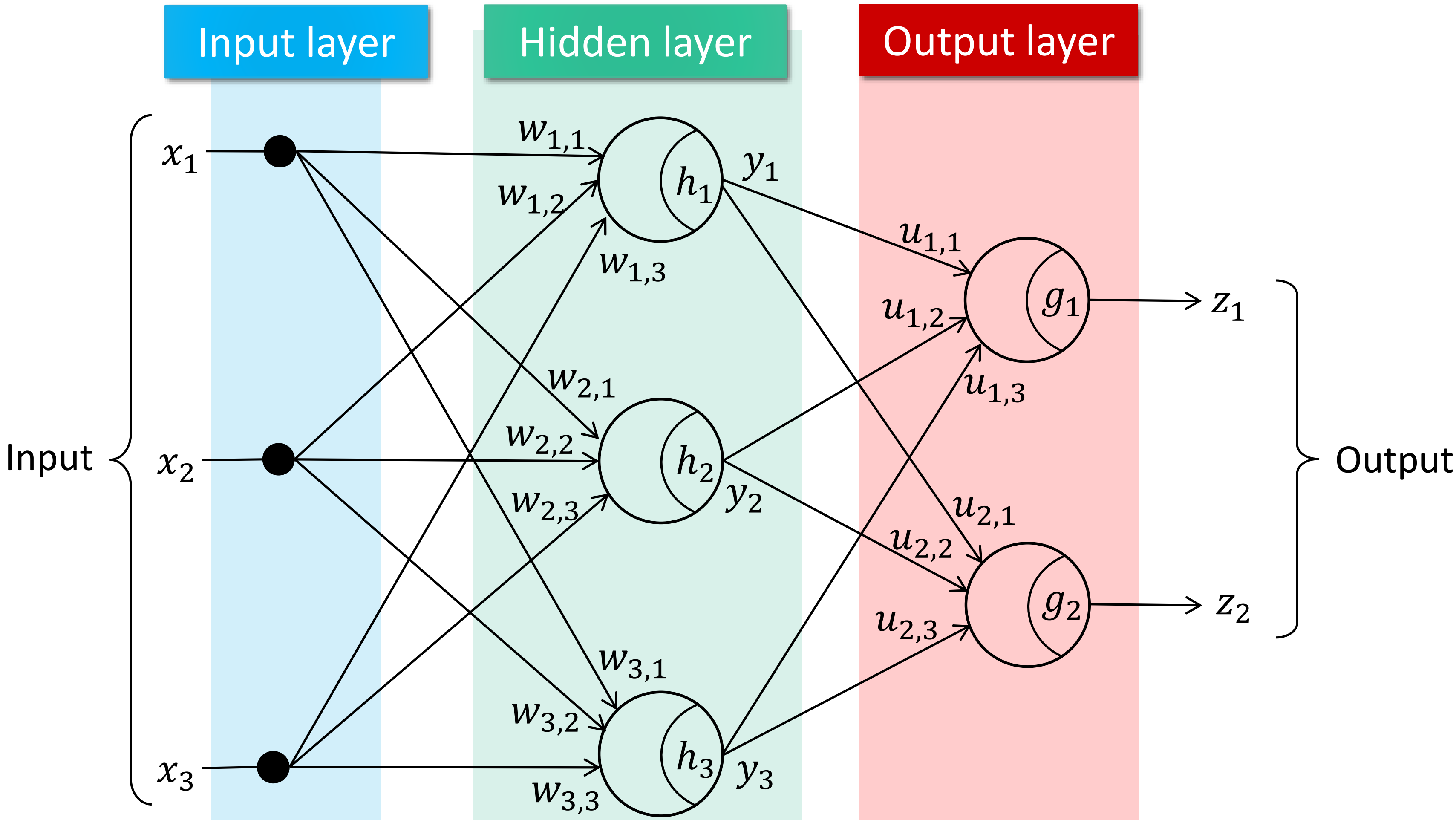
Multiple Layer Neural Network (Perceptron)

We can construct a three layer neural network as follows. The first layer simply copies the inputs. The outputs of the second layer are the inputs of the third layer.



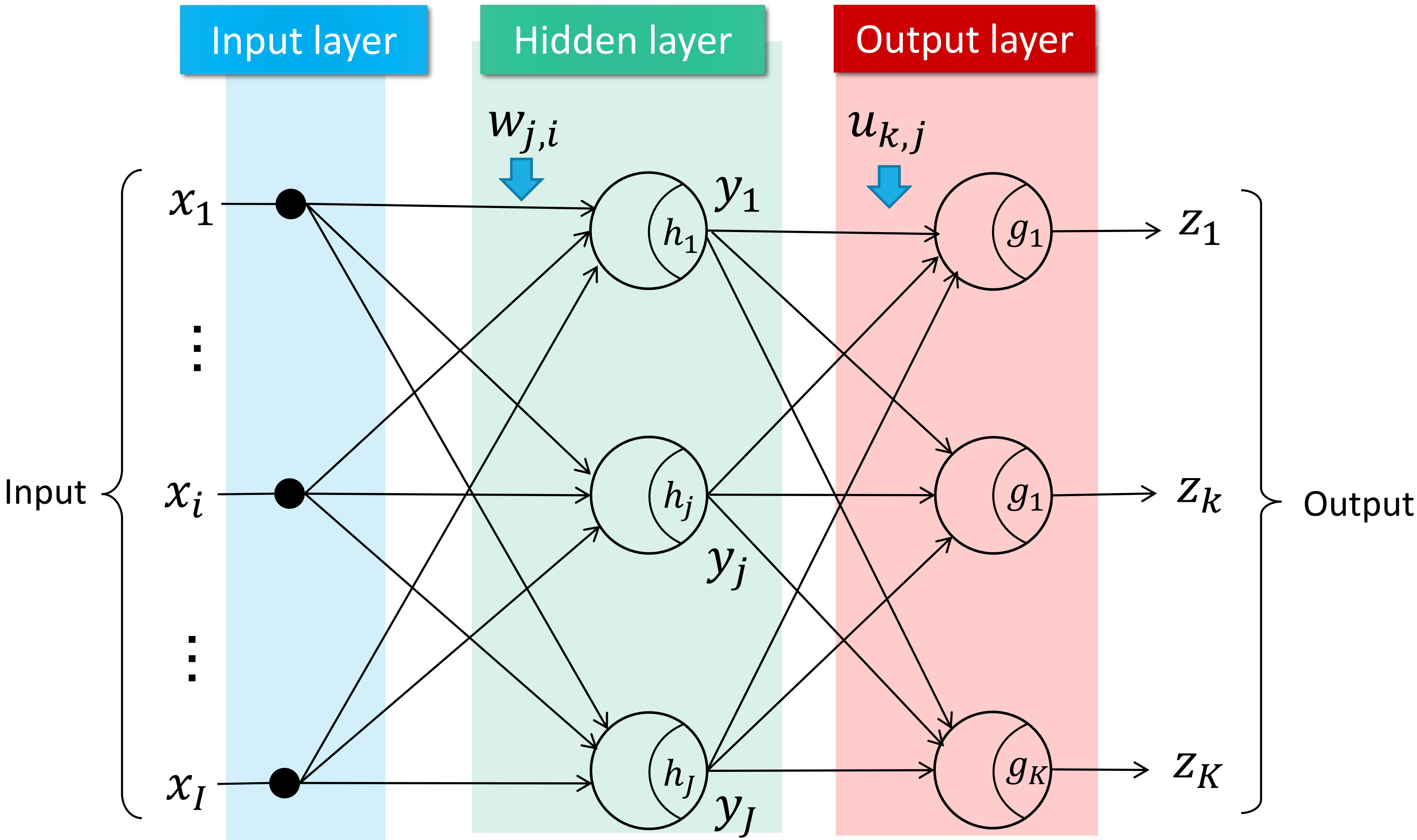
Multiple Layer Neural Network (Perceptron)

We refer to each layer as "input layer", "hidden layer" and "output layer" respectively.

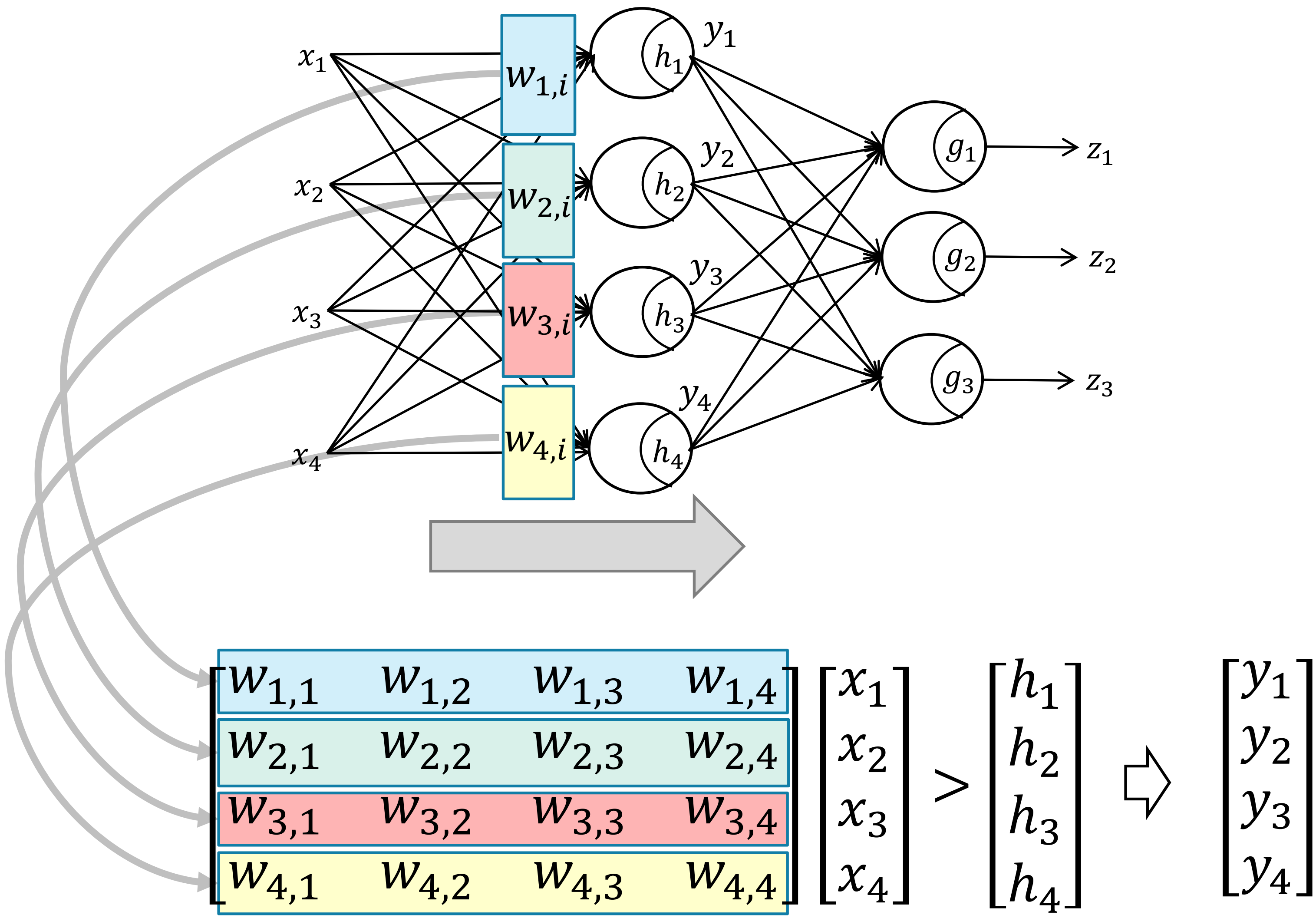


Multiple Layer Neural Network (Perceptron)

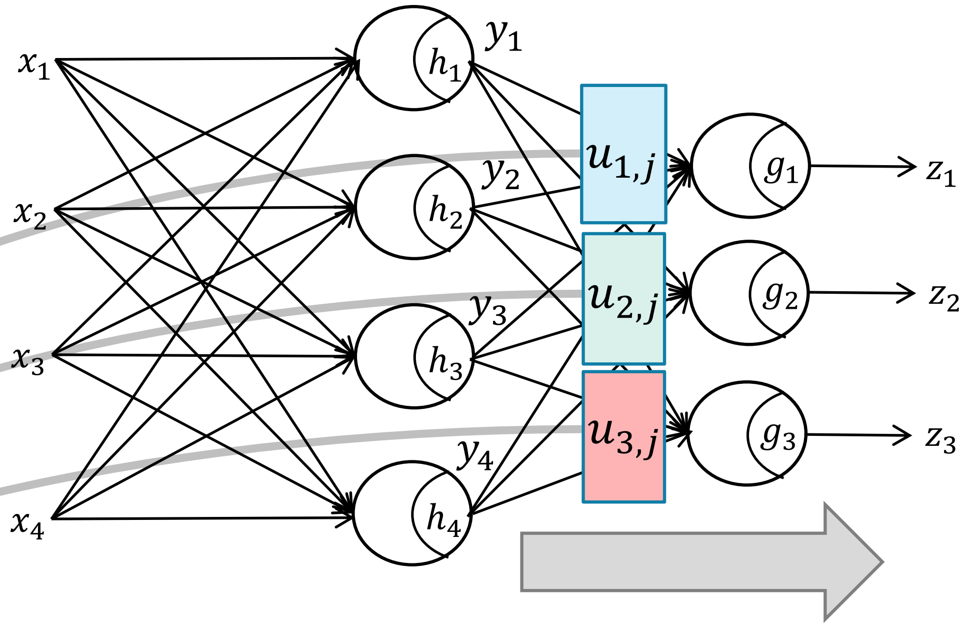
Generally,



Feedforward calculation (1)



Feedforward calculation (2)



$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} \\ u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} > \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \Rightarrow \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}$$

Let’s implement “Perceptron” on MATLAB

Test script

example1_5.m

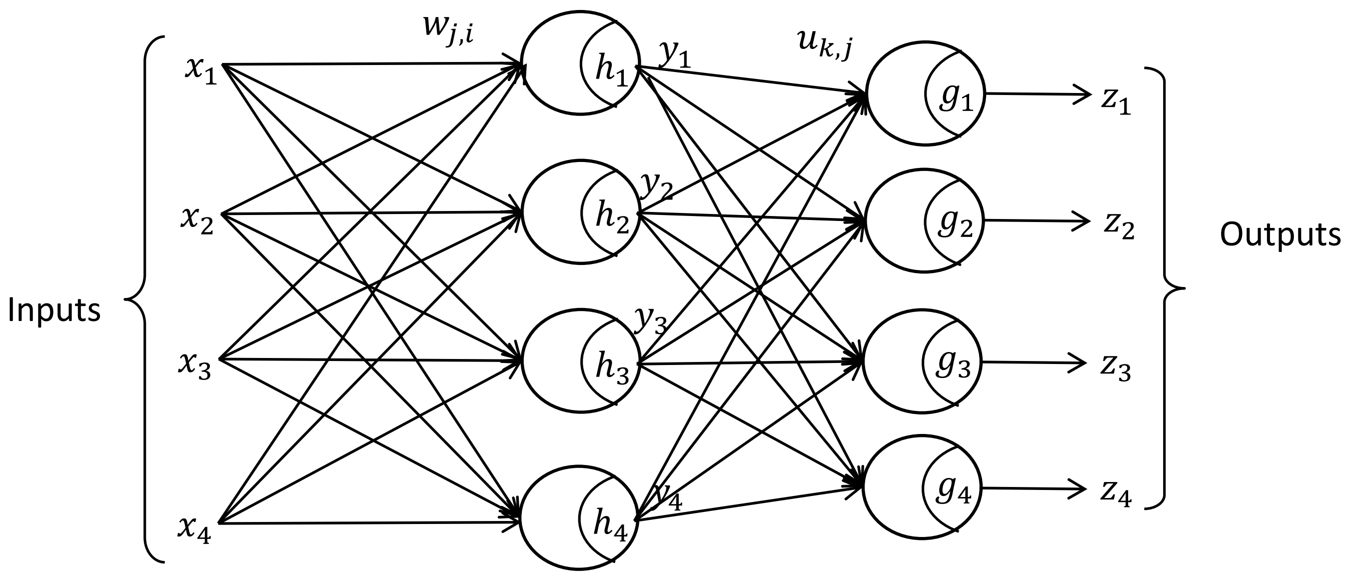
```
x = [0,0,0,0,1,1,1,1;
     0,0,1,1,0,0,1,1;
     0,1,0,1,0,1,0,1];

w = [0.5,  1.0,  0.5;
     0.0,  0.5,  1.0;
     1.0,  0.5,  0.0];
h = [0.5;
     1.0;
     0.0];

u = [1.0,  0.5,  0.0;
     0.5,  0.0,  1.0];
g = [1.0;
     0.0];

layer1 = FormalNeuronLayer(w,h);
layer2 = FormalNeuronLayer(u,g);

y = layer1.forward(x)
z = layer2.forward(y)
```

Execution and results

```
>> example1_5
y =

   0   0   1   1   0   1   1   1
   0   0   0   1   0   0   0   1
   0   0   1   1   1   1   1   1

z =

   0   0   0   1   0   0   0   1
   0   0   1   1   1   1   1   1
```

Exercise1.6

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.



There are 3 samples.

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