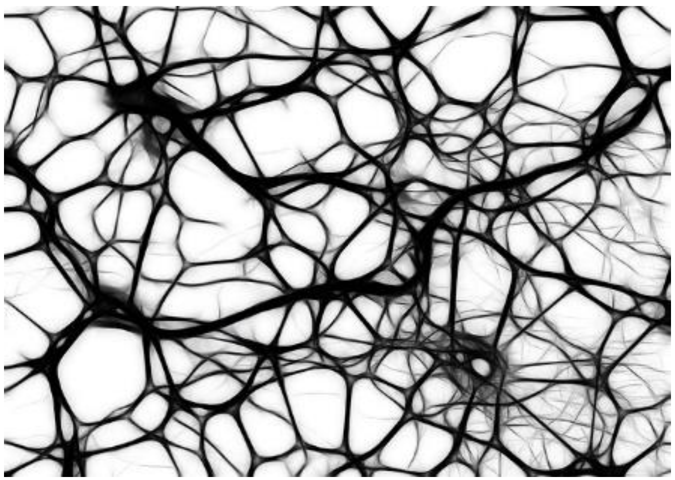
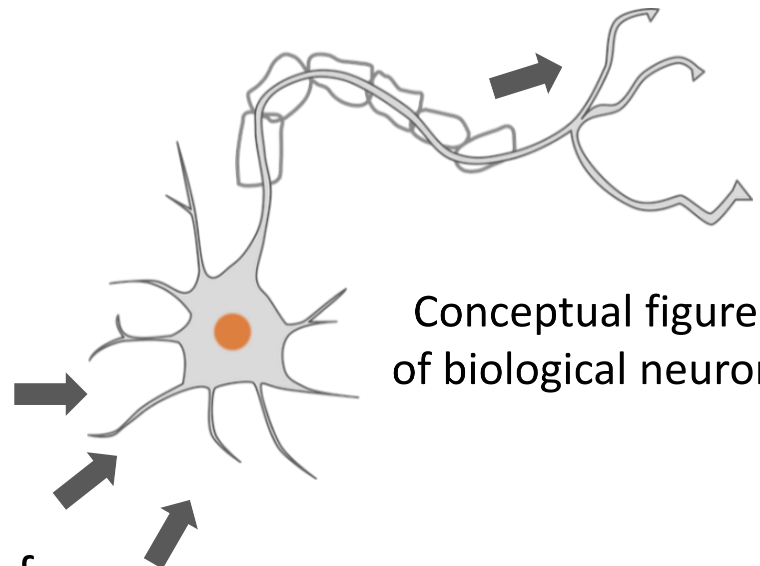
Perceptron

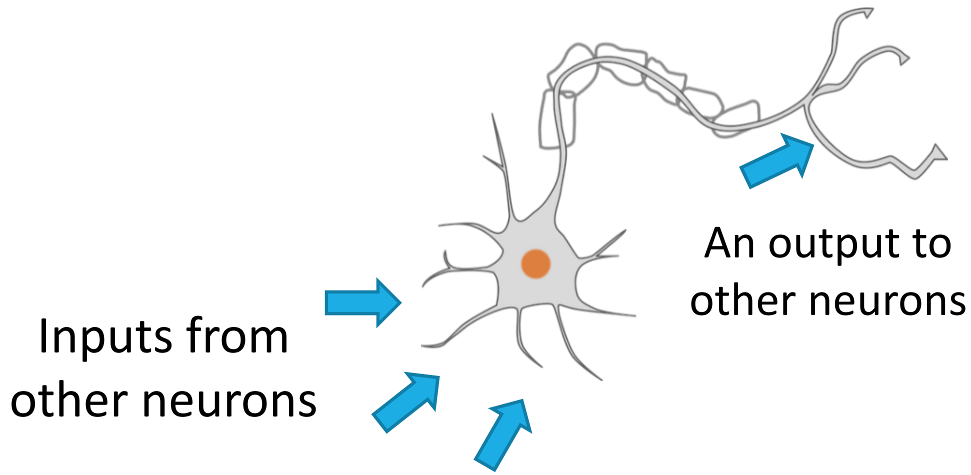
Formal Neuron (McCulloch-Pitts Model)

Formal neuron is a simple model proposed by *Warren McCulloch* and *Walter Pitts* in 1943. It is expressed in one mathematical function derived from the simplification of biological neurons.

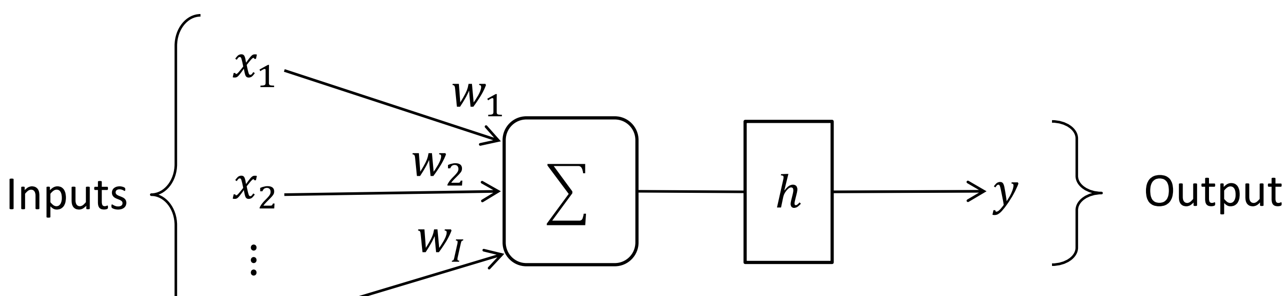




Formal Neuron (McCulloch-Pitts Model)

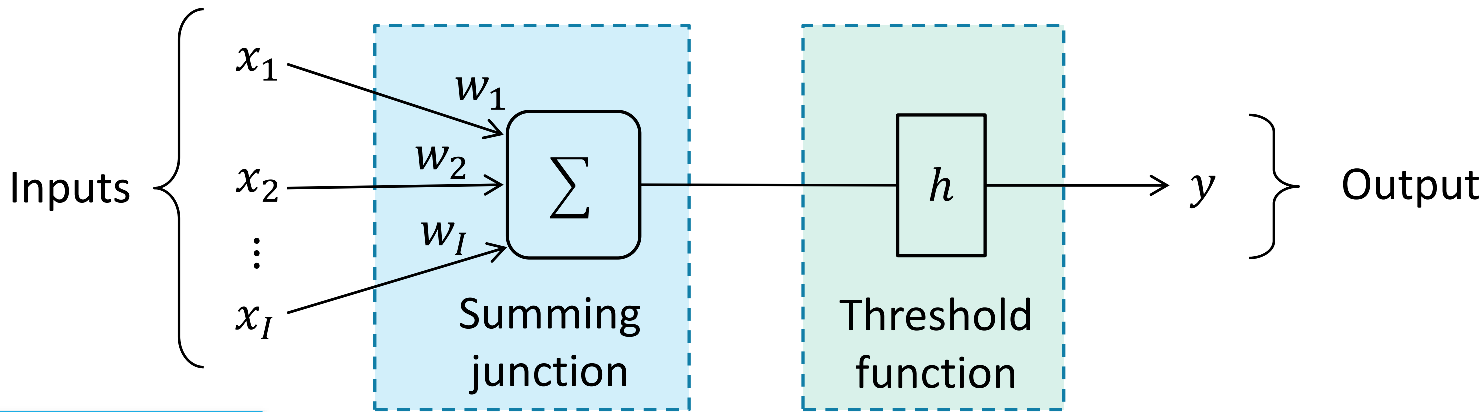


Each formal neuron obtains several binary values $x_1, x_2, \cdots, x_I$ as inputs and emit one binary value y as an output. A neuron has several ***weights***, corresponding to each input.



Formal neuron model

Formal Neuron (McCulloch-Pitts Model)



Summing junction

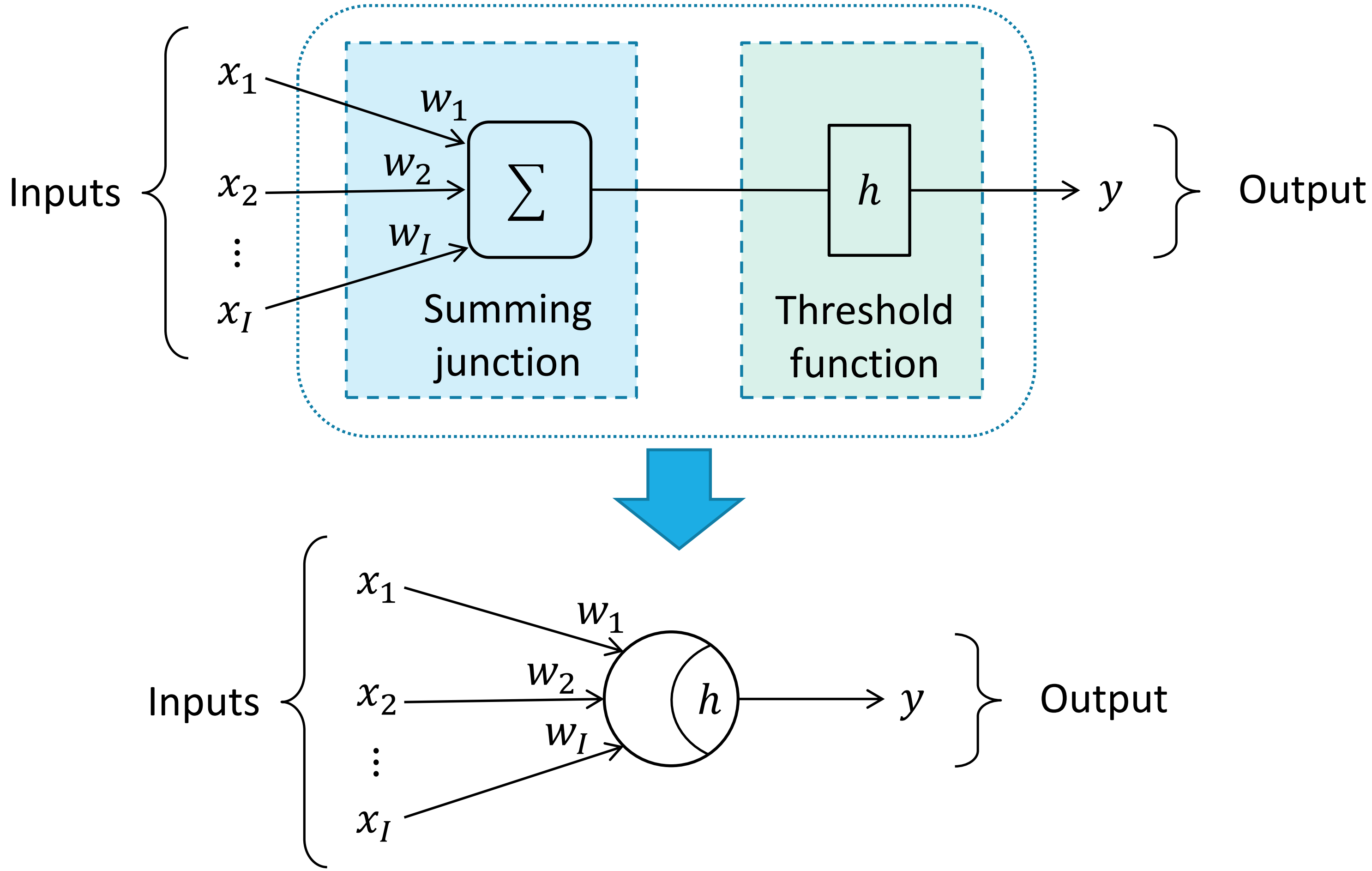
The inputs $x_1, x_2, \cdots, x_I$ are multiplied by the corresponding weights $w_1, w_2, \cdots, w_I$ respectively. Then the weighted sum value (i.e., $\sum_i x_i w_i$) is calculated.

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,

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

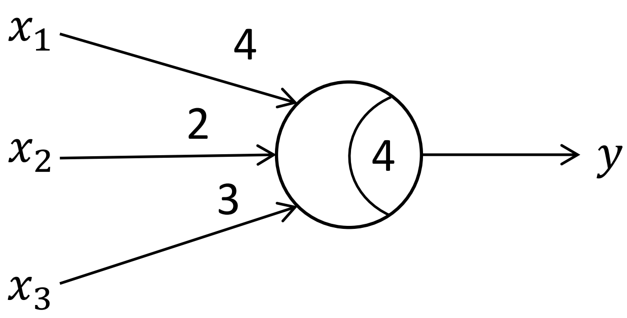
Formal Neuron (McCulloch-Pitts Model)



When the input signal exceeds a threshold, it is sometimes described as ***firing***.

【Example 1.1】

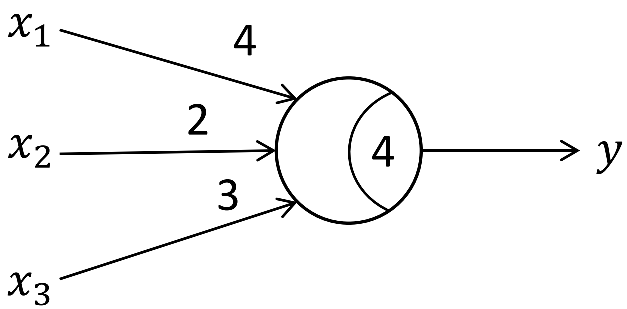
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



x_1	x_2	x_3	$\sum_i x_i w_i$	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		

【Example 1.1】

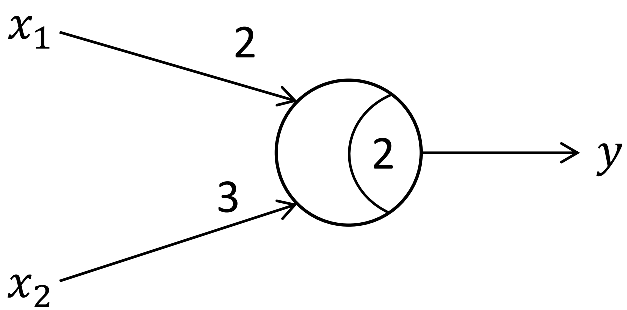
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



x_1	x_2	x_3	$\sum_i x_i w_i$	y
0	0	0	0	0
0	0	1	3	0
0	1	0	2	0
0	1	1	5	1
1	0	0	4	0
1	0	1	7	1
1	1	0	6	1
1	1	1	9	1

【Exercise 1.1】

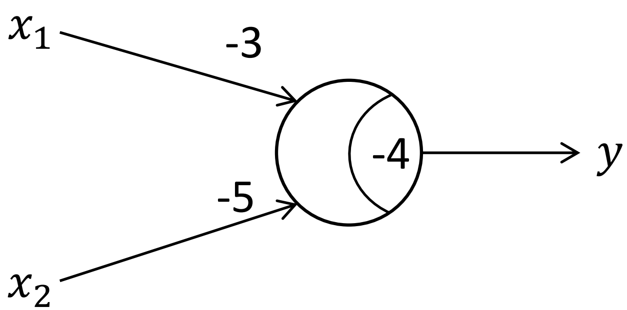
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



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

【Exercise 1.2】

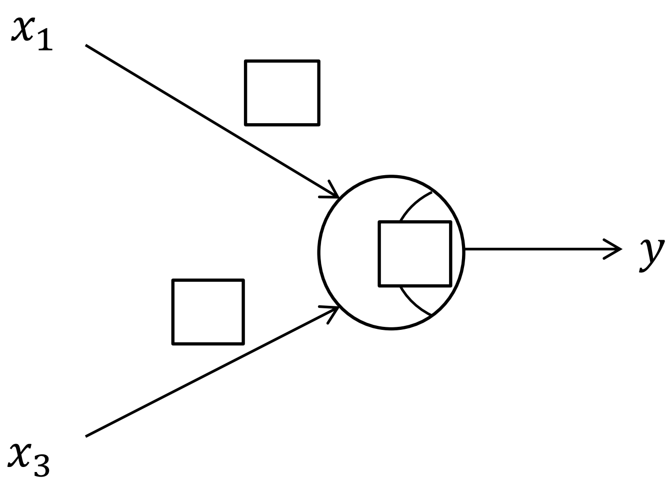
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



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

【Exercise 1.3】

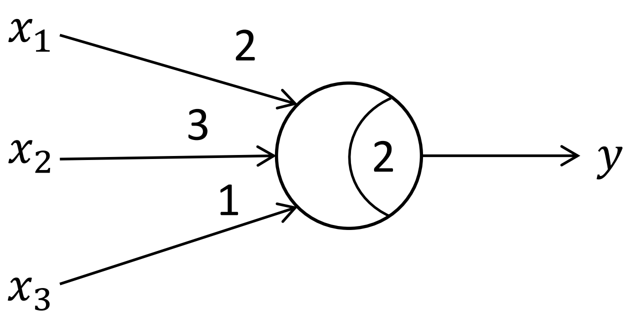
For the following neuron, determine the weights and the threshold so that the outputs y for each input are as shown in the table below.



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

【Exercise 1.4】

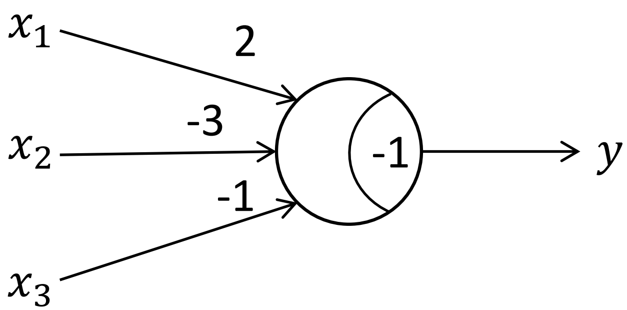
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



x_1	x_2	x_3	$\sum_i x_i w_i$	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 1.5】

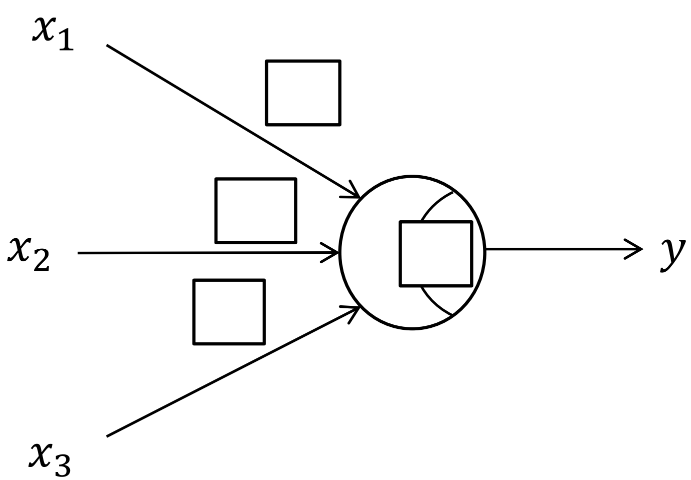
For the following formal neuron, given the values of inputs, weights and a threshold as shown in the table below, find the weighted sum values and outputs.



x_1	x_2	x_3	$\sum_i x_i w_i$	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 1.6】

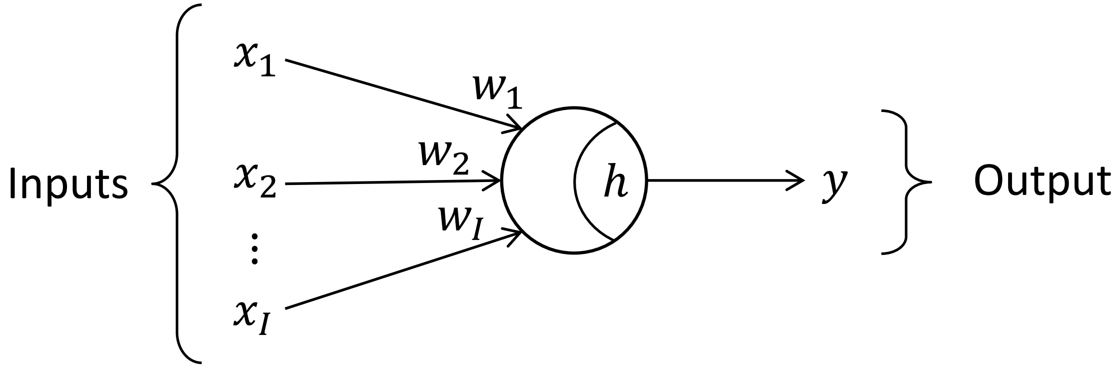
For the following neuron, determine the weights and the threshold so that the outputs y for each input are as shown in the table below.



Can you find these values?

x_1	x_2	x_3	$\sum_j x_j w_j$	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 dot-product



Here, we define a vector $\boldsymbol{x}$ whose elements are inputs x_i.

$$\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_I]$$

Similary, we also define a vector $\boldsymbol{w}$ whose elements are weight w_i.

$$\boldsymbol{w} = [w_1 \quad w_2 \quad \cdots \quad w_I]$$

Then, the weighted sum value is calculated as **dot-product** of $\boldsymbol{x}$ and $\boldsymbol{w}^t$, where $\boldsymbol{w}^t$ is the transpose of $\boldsymbol{w}$.

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

【Example 1.2】Implementing dot-product

It is easy to implement in Python or MATLAB.

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

Python

```
import numpy as np

x = np.array([1, 0, 1])
w = np.array([2, 3, -1])
np.dot(x, w.T)

Out:
1
```

```
import numpy as np

x = np.array([1, 0, 1])
w = np.array([[2], [3], [-1]])
np.dot(x, w)

Out:
Array([1])
```

MATLAB

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

ans =
     1
```

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

ans =
     1
```

【Example1.3】Implementing threshold

If a is greater than b, the result in Python or MATLAB operation "a > b" is 1 (True), otherwise the result is 0 (False).

Python

```
a = 3
b = 2
print(a > b)

True
```

```
a = 2
b = 3
print(a > b)

False
```

MATLAB

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

ans = 1
```

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

ans = 0
```

#【Exercise 1.7】

Get the following dot product in Python or MATLAB.

Note that the definition of row vectors and the definition of column vectors are different in the program.

Also check that you get an error when you try to calculate whose dimensions do not match.

(A) $\boldsymbol{x}\boldsymbol{w}$
(B) $\boldsymbol{x}\boldsymbol{w}^t$
(C) $\boldsymbol{x}^t\boldsymbol{w}$
(D) $\boldsymbol{x}^t\boldsymbol{w}^t$

$$\text{, where } \boldsymbol{x} = [3 \quad 5 \quad 1 \quad 6], \boldsymbol{w} = [2 \quad 7 \quad 1 \quad 2].$$

【Exercise 1.8】

Perform the following comparison operations in Python or MATLAB.
Check the results of vector-to-scalar and vector-to-vector comparisons.

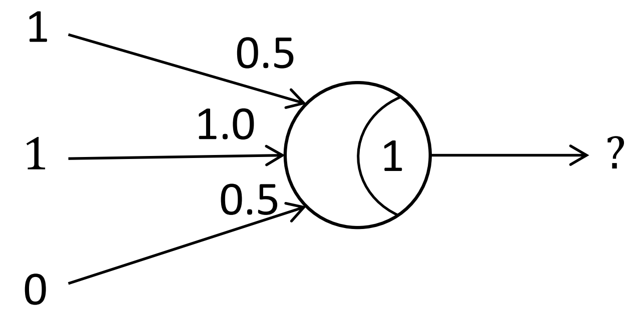
(A) $a > b$
(B) $b > a$
(C) $\boldsymbol{x} > a$
(D) $\boldsymbol{x} > \boldsymbol{w}$

, where $a = 5,\quad b = 3,\quad \boldsymbol{x} = [3 \quad 5 \quad 1 \quad 6],\ \boldsymbol{w} = [2 \quad 7 \quad 1 \quad 2].$

【Example 1.4】 Implementing a formal neuron model

Python

```
import numpy as np

x = np.array([1, 1, 0])
w = np.array([0.5, 1.0, 0.5])
h = 1

def formal_neuron(x, w, h):

    p = np.dot(x, w)
    y = p > h
    #cast from  boolean to integer
    return y.astype(np.int)

y = formal_neuron(x, w.T, h)
print(y)
```



The weighted sum is calculated as

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

Note $\boldsymbol{x}$ is a row vector.
$\boldsymbol{w}^{\boldsymbol{t}}$ is a column vector.
p is a scalar.

【Example 1.4】Implementing a formal neuron model

MATLAB

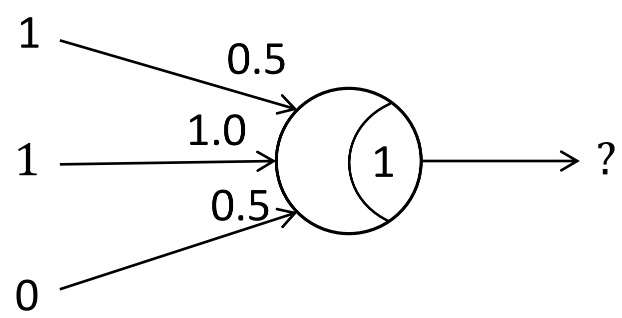
example1_4.m

```
x = [1, 1, 0];
w = [0.5, 1.0, 0.5];
h = 1;
y = formal_neuron(x,w',h)
```

Create formal_neuron.m
as a function file as follows.

formal_neuron.m

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



The weighted sum is calculated as

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

Note

$\boldsymbol{x}$ is a row vector.
$\boldsymbol{w}^t$ is a column vector.
p is a scalar.

【Example 1.4】Results

Results in Python

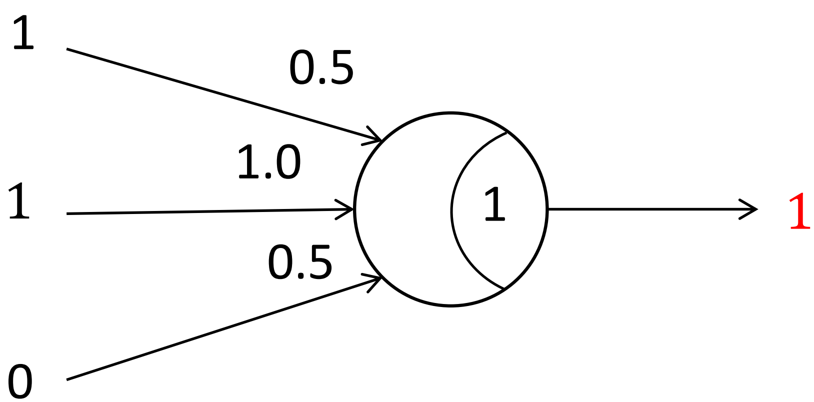
```
Out:
1
```

Results in MATLAB

```
>> example1_4
y = 1
```

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

> 1 (Threshold)



【Exercise1.9】

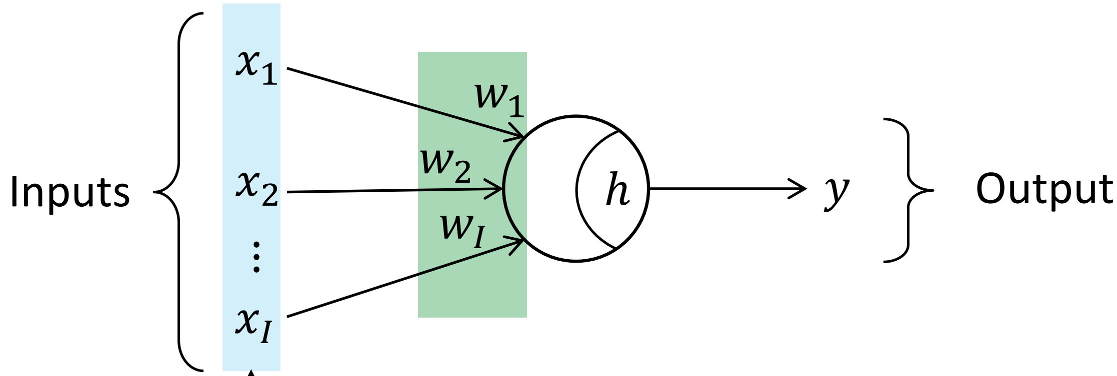
Implement example1.4 in Pythons or MATLAB.

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

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

$$\boldsymbol{w} = [2, -1, 3]$$
$$h = 1$$

Dot-product for multiple input data



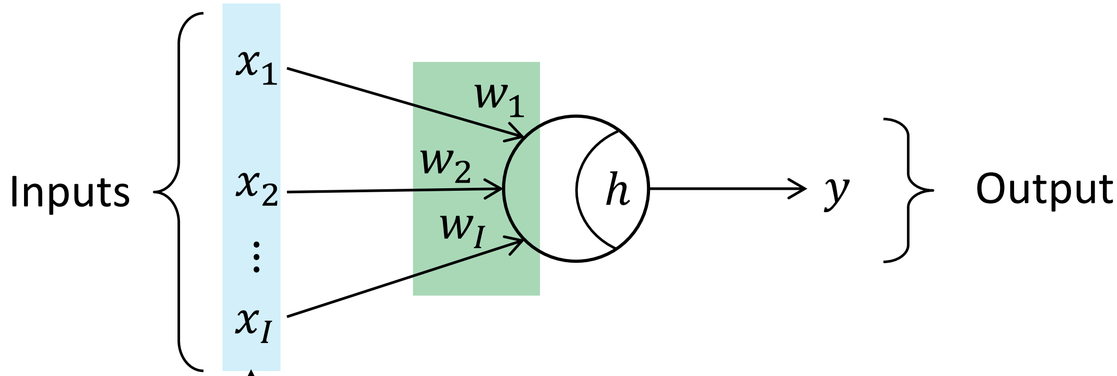
$$\boldsymbol{x} = [x_1 \quad x_2 \quad \cdots \quad x_I] \qquad \boldsymbol{w}^t = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_I \end{bmatrix}$$

$$p = \boldsymbol{x}\boldsymbol{w}^t$$

$$y = \begin{cases} 0 & \text{if } p \leq h \\ 1 & \text{if } p > h \end{cases}$$

A single input vector represents a single piece of sample data.

Dot-product for multiple input data



$$\boldsymbol{x_1} = [x_{1,1} \quad x_{1,2} \quad \cdots \quad x_{1,I}]$$

$$\boldsymbol{x_2} = [x_{2,1} \quad x_{2,2} \quad \cdots \quad x_{2,I}]$$

$$\boldsymbol{x_3} = [x_{3,1} \quad x_{3,2} \quad \cdots \quad x_{3,I}]$$

$$\vdots$$

$$\boldsymbol{x_N} = [x_{N,1} \quad x_{N,2} \quad \cdots \quad x_{N,I}]$$

$$\boldsymbol{w^t} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_I \end{bmatrix}$$

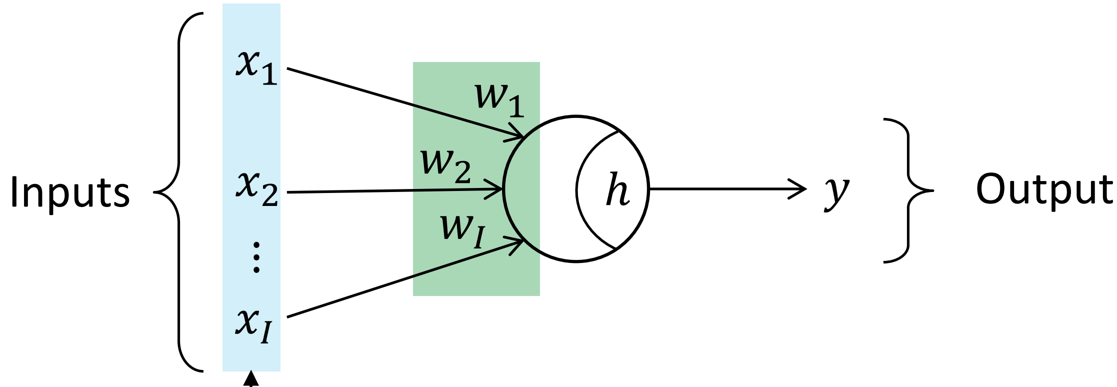
$$p_1 = \boldsymbol{x_1 w^t}$$

$$p_2 = \boldsymbol{x_2 w^t}$$

$$p_3 = \boldsymbol{x_3 w^t}$$

$$p_N = \boldsymbol{x_N w^t}$$

Dot-product for multiple input data



$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,I} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,I} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,I} & x_{N,2} & \cdots & x_{N,I} \end{bmatrix} \quad \boldsymbol{w}^t = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_I \end{bmatrix} \quad \boldsymbol{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_1\boldsymbol{w}^t \\ \boldsymbol{x}_2\boldsymbol{w}^t \\ \vdots \\ \boldsymbol{x}_N\boldsymbol{w}^t \end{bmatrix} = \boldsymbol{X}\boldsymbol{w}^t$$

By representing multiple inputs as an input matrix, the entire output can be computed in a single calculation.

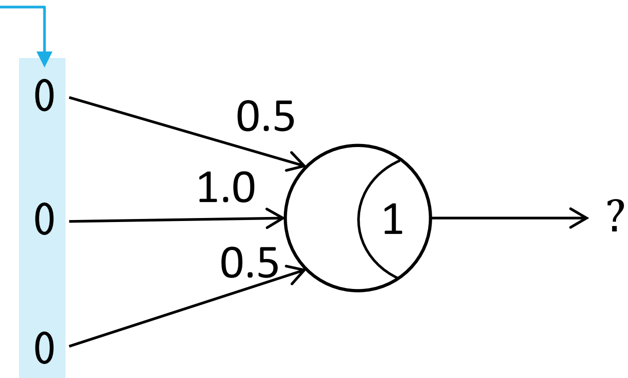
【Example1.5】Implementing a formal neuron model 2

Multiple input data can be represented as **an input matrix.**

Python

```
import numpy as np

x = np.array([[0, 0, 0],
              [0, 0, 1],
              [0, 1, 0],
              [0, 1, 1],
              [1, 0, 0],
              [1, 0, 1],
              [1, 0, 1],
              [1, 1, 0],
              [1, 1, 1]])
w = np.array([0.5,  1.0,  0.5])
h = 1

def formal_neuron(x,  w,  h):

    p = np.dot(x,  w)
    y = p > h
    #cast from boolean to integer
    return y.astype(np.int)

y = formal_neuron(x,  w.T,  h)
print(y)
```

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



$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_8 \end{bmatrix} = \underbrace{\overbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ \vdots & \vdots & \vdots \\ x_{8,I} & x_{8,2} & x_{8,3} \end{bmatrix}}^{I=3 \text{ (Input dimension)}}}_{\text{Input Matrix}} * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

N=8 (the number of samples)

Note

$\boldsymbol{X}$ is a 8*3 ($N \times I$) matrix.

$\boldsymbol{w^t}$ is a column vector.

$\boldsymbol{p}$ is a column vector.

【Example1.5】Implementing a formal neuron model 2

Multiple input data can be represented as **an input matrix.**

MATLAB

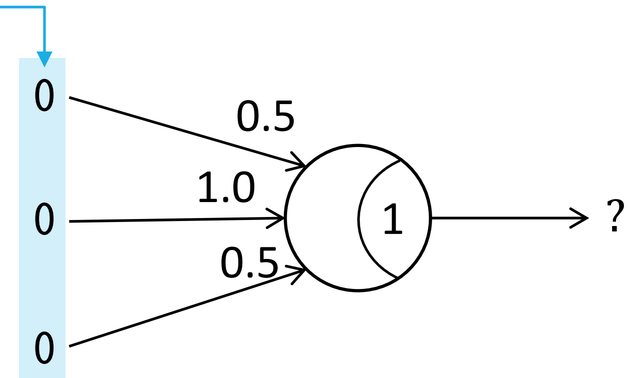
example1_5.m

```
x = [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 = [0.5, 1.0, 0.5];
h = 1;
y = formal_neuron(x,w',h)
```

formal_neuron.m

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

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



I=3 (Input dimension)

$$\begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_8 \end{bmatrix} = N\text{=8 (the number of samples)} \left\{ \overbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ \vdots & \vdots & \vdots \\ x_{8,I} & x_{8,2} & x_{8,3} \end{bmatrix}}^{I=3} \right. * \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

Input Matrix

Note

$\boldsymbol{X}$ is a 8*3 ($N \times I$) matrix.
$\boldsymbol{w}^t$ is a column vector.
$\boldsymbol{p}$ is a column vector.

【Example1.5】Results

Results in Python

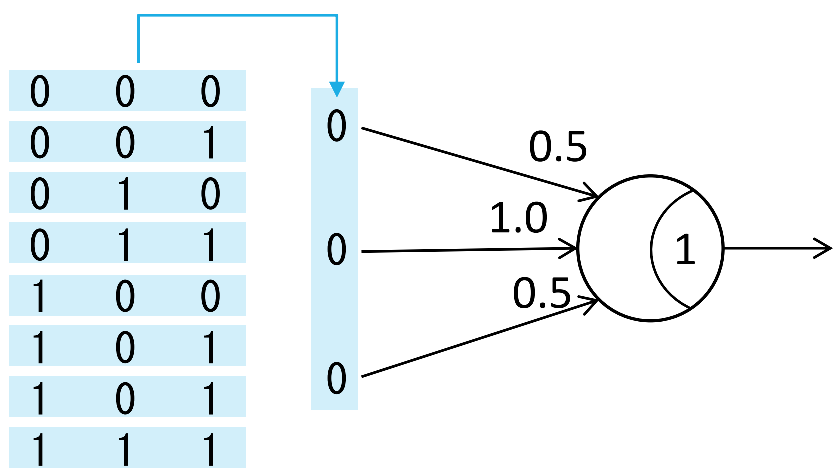
```
Out:
[0 0 1 0 0 1 1]
```

Results in MATLAB

```
>> example1_5
y =

   0
   0
   0
   1
   0
   0
   1
   1
```

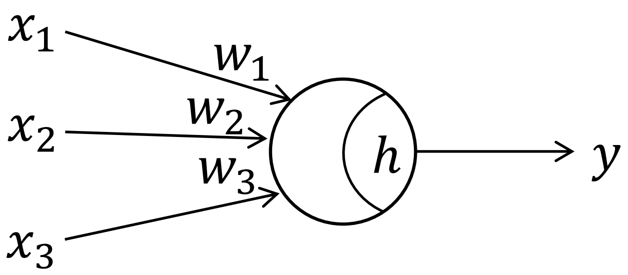
$$\begin{bmatrix}0&0&0\\0&0&1\\0&1&0\\0&1&1\\1&0&0\\1&0&1\\1&1&0\\1&1&1\end{bmatrix}\cdot\begin{bmatrix}0.5\\1.0\\0.5\end{bmatrix}=\begin{bmatrix}0\\0.5\\1.0\\1.5\\0.5\\1.0\\1.5\\2.0\end{bmatrix}$$

$$\begin{bmatrix}0\\0.5\\1.0\\1.5\\0.5\\1.0\\1.5\\2.0\end{bmatrix}>1=\begin{bmatrix}0\\0\\0\\1\\0\\0\\1\\1\end{bmatrix}$$



【Exercise1.10】

Consider the single 3-input neuron (the input dimension is 3) as follows.



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

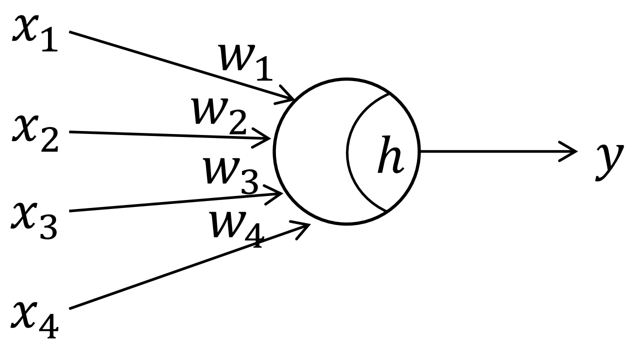
$$\boldsymbol{X} = \overbrace{\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}}^{\text{Input dimension is 3.}} \left.\vphantom{\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}}\right\} \text{There are 3 samples}$$

$$\boldsymbol{w} = [0.6 \quad -1.5 \quad 1.0]$$

$$h = 1.0$$

【Exercise1.11】

Consider the single 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 Python or MATLAB script.

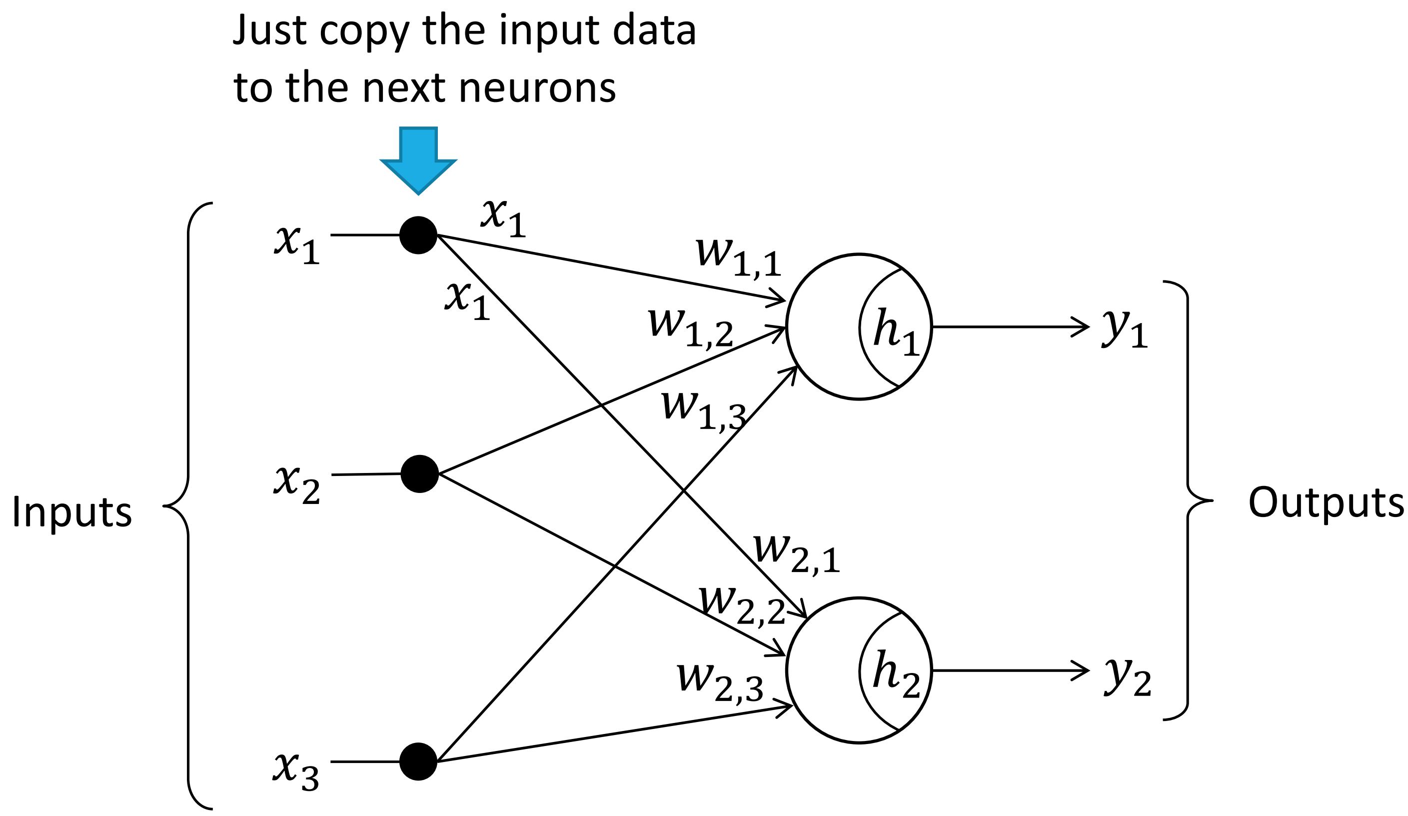
$$\boldsymbol{X} = \overbrace{\begin{bmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}}^{\text{Input dimension is 4.}} \left.\vphantom{\begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}}\right\} \text{There are 3 samples}$$

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

$$h = 1.5$$

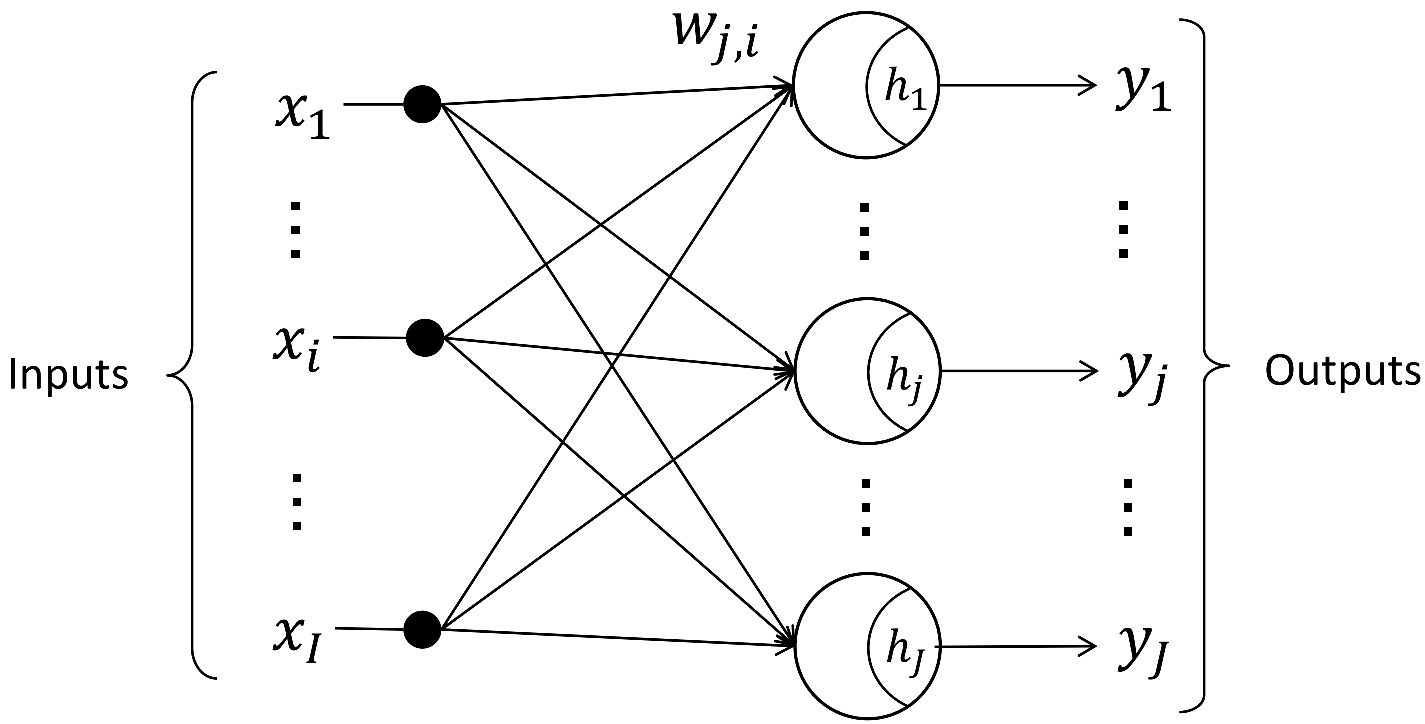
Using multiple neurons

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

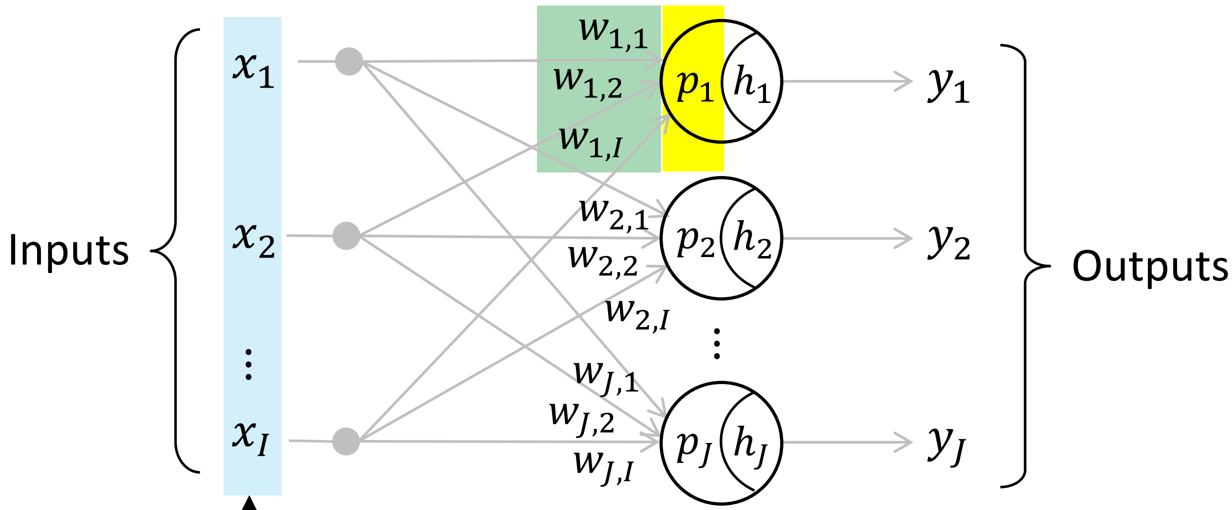


Using multiple neurons

Generally,



Dot-product for multiple neurons



$$\boldsymbol{x_1} = [x_{1,1} \quad x_{1,2} \quad \cdots \quad x_{1,I}]$$

$$\boldsymbol{x_2} = [x_{2,1} \quad x_{2,2} \quad \cdots \quad x_{2,I}]$$

$$\boldsymbol{x_3} = [x_{3,1} \quad x_{3,2} \quad \cdots \quad x_{3,I}]$$

$$\vdots$$

$$\boldsymbol{x_N} = [x_{N,1} \quad x_{N,2} \quad \cdots \quad x_{N,I}]$$

$$\boldsymbol{w_1^t} = \begin{bmatrix} w_{1,1} \\ w_{1,2} \\ \vdots \\ w_{1,I} \end{bmatrix}$$

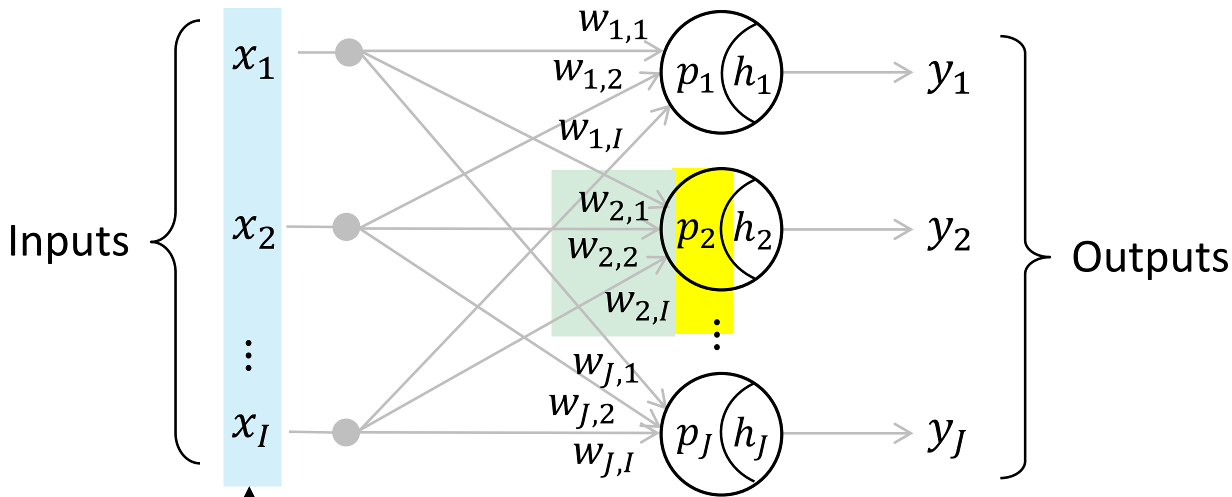
$$p_{1,1} = \boldsymbol{x_1 w^t}$$

$$p_{2,1} = \boldsymbol{x_2 w^t}$$

$$p_{3,1} = \boldsymbol{x_3 w^t}$$

$$p_{N,1} = \boldsymbol{x_N w^t}$$

Dot-product for multiple neurons



$$\boldsymbol{x_1} = [x_{1,1} \quad x_{1,2} \quad \cdots \quad x_{1,I}]$$

$$\boldsymbol{x_2} = [x_{2,1} \quad x_{2,2} \quad \cdots \quad x_{2,I}]$$

$$\boldsymbol{x_3} = [x_{3,1} \quad x_{3,2} \quad \cdots \quad x_{3,I}]$$

$$\vdots$$

$$\boldsymbol{x_N} = [x_{N,1} \quad x_{N,2} \quad \cdots \quad x_{N,I}]$$

$$\boldsymbol{w_2^t} = \begin{bmatrix} w_{2,1} \\ w_{2,2} \\ \vdots \\ w_{2,I} \end{bmatrix}$$

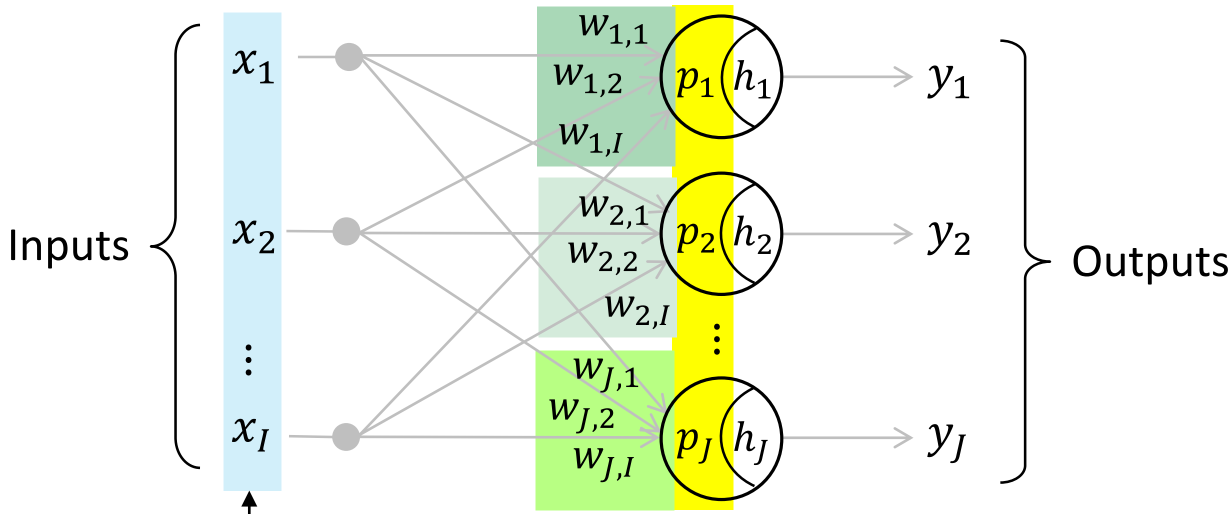
$$p_{1,2} = \boldsymbol{x_1 w^t}$$

$$p_{2,2} = \boldsymbol{x_2 w^t}$$

$$p_{3,2} = \boldsymbol{x_3 w^t}$$

$$p_{N,2} = \boldsymbol{x_N w^t}$$

Dot-product for multiple neurons

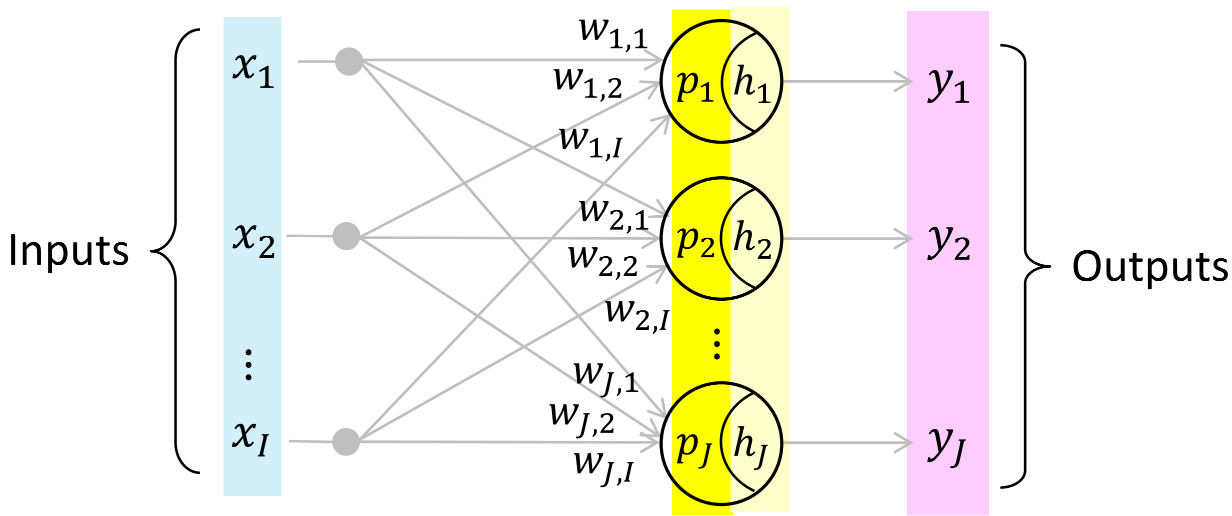


$$\boldsymbol{X} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,I} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,I} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,I} \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} w_{1,1} & w_{2,1} & \cdots & w_{J,1} \\ w_{1,2} & w_{2,2} & \cdots & w_{J,2} \\ \vdots & \vdots & \ddots & \vdots \\ w_{1,I} & w_{2,I} & \cdots & w_{J,I} \end{bmatrix} \quad \boldsymbol{P} = \boldsymbol{X}\boldsymbol{W} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,J} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N,I} & p_{2,I} & \cdots & p_{N,J} \end{bmatrix}$$

$$\boldsymbol{X} = N \times I \qquad \boldsymbol{W} = I \times J \qquad \boldsymbol{P} = N \times J$$

The weight vectors of each neuron are arranged vertically and represented as a matrix.

Threshold calculation for multiple neurons



$$\boldsymbol{P} = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,J} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ p_{N,I} & p_{2,I} & \cdots & p_{N,J} \end{bmatrix} \begin{matrix} \leftarrow > \rightarrow \\ \leftarrow > \rightarrow \\ \\ \leftarrow > \rightarrow \end{matrix} \quad \boldsymbol{h} = [h_1 \quad h_2 \quad \cdots \quad h_J] \quad \Rightarrow \quad \boldsymbol{Y} = \begin{bmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,J} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ y_{N,I} & y_{2,I} & \cdots & y_{N,J} \end{bmatrix}$$

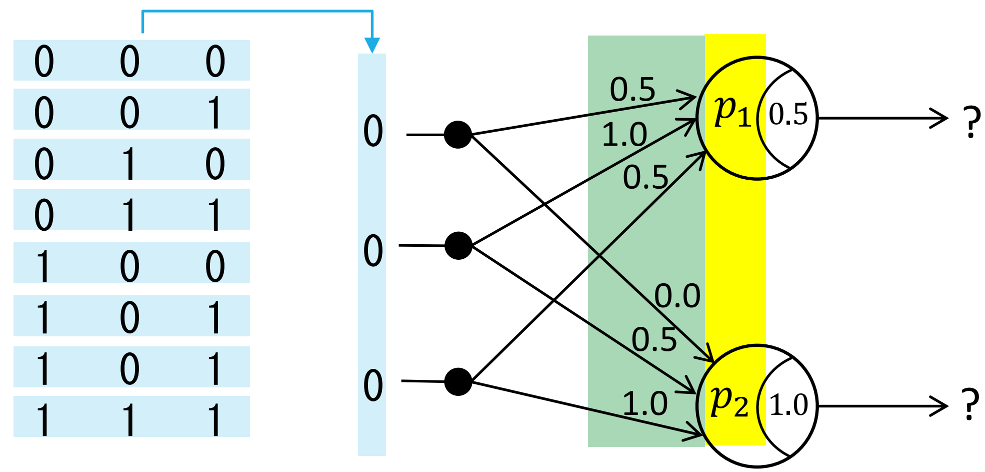
$$\boldsymbol{P} = N \times J \qquad \boldsymbol{Y} = N \times J$$

【Example1.6】Implementing a formal neuron model 3

The weights of multiple neurons can be represented by **a weight matrix**

Python

```
import numpy as np

x=np.array([[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 = np.array([[0.5, 0.0],
              [1.0, 0.5],
              [0.5, 1.0]])
h = np.array([0.5, 1.0])

def formal_neuron(x, w, h):
  p = np.dot(x, w)
  y = p > h
  #convert boolean to integer
  return y.astype(np.int)

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



I=3 (Input dimension)

N=8 (the number of samples)

$$\begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{2,2} & p_{2,2} \\ \vdots & \vdots \\ p_{8,1} & p_{8,2} \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ \vdots & \vdots & \vdots \\ x_{8,1} & x_{8,2} & x_{8,3} \end{bmatrix} * \begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \\ w_{1,3} & w_{2,3} \end{bmatrix}$$

Input Matrix

Note

$\boldsymbol{X}$ is a 8*3 ($N \times I$) matrix.
$\boldsymbol{W}$ is a 3*2 ($I \times J$) matrix.
$\boldsymbol{P}$ is a 8*2 ($N \times J$) matrix.

【Example1.6】Implementing a formal neuron model 3

The weights of multiple neurons can be represented by **a weight matrix**

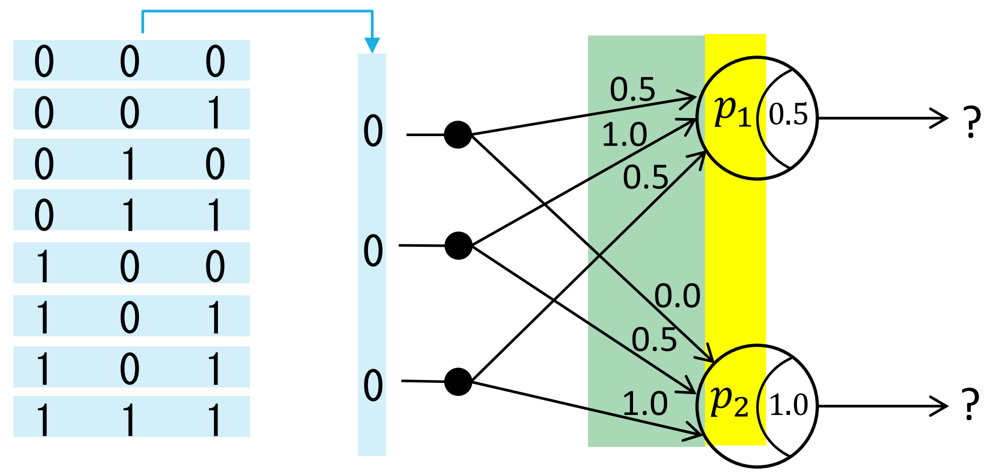
MATLAB

example1_6.m

```
x = [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 = [0.5, 0.0;
     1.0, 0.5;
     0.5, 1.0];
h = [0.5, 1.0];

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

formal_neuron.m

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



$$
\begin{bmatrix} p_{1,1} & p_{1,2} \\ p_{2,2} & p_{2,2} \\ \vdots & \vdots \\ p_{8,1} & p_{8,2} \end{bmatrix}
=
\underbrace{\overbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & x_{1,3} \\ x_{2,1} & x_{2,2} & x_{2,3} \\ \vdots & \vdots & \vdots \\ x_{8,1} & x_{8,2} & x_{8,3} \end{bmatrix}}^{I=3 \text{ (Input dimension)}}}_{\text{Input Matrix}}
*
\begin{bmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \\ w_{1,3} & w_{2,3} \end{bmatrix}
$$

N=8 (the number of samples)

Note

$\boldsymbol{X}$ is a 8*3 ($N \times I$) matrix.

$\boldsymbol{W}$ is a 3*2 ($I \times J$) matrix.

$\boldsymbol{P}$ is a 8*2 ($N \times J$) matrix.

【Example1.6】Results

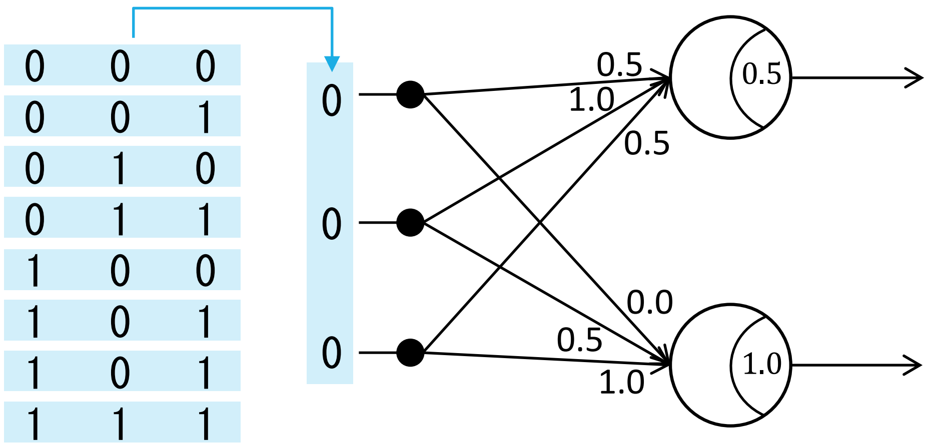
Results in Python

```
OUT:
[[0 0]
[0 0]
[1 0]
[1 1]
[0 0]
[1 0]
[1 0]
[1 1]]
```

Results in MATLAB

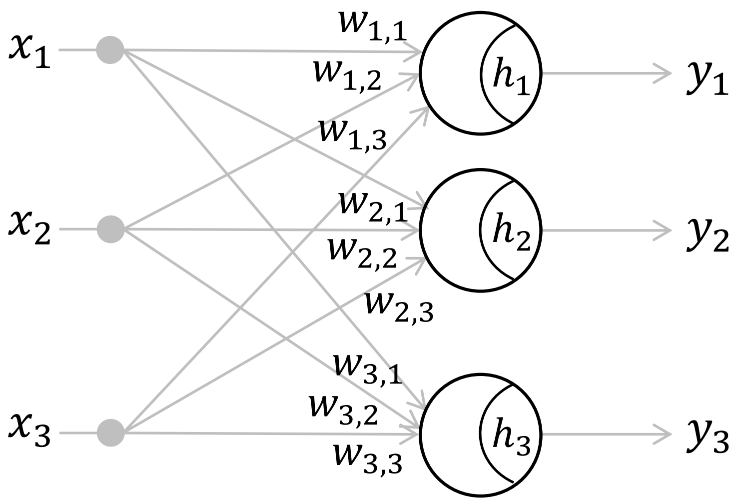
```
>> example1_6
y =

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

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 0.5 & 0.0 \\ 1.0 & 0.5 \\ 0.5 & 1.0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 0.5 & 1.0 \\ 1.0 & 0.5 \\ 1.5 & 1.5 \\ 0.5 & 0.0 \\ 1.0 & 1.0 \\ 1.5 & 0.5 \\ 2.0 & 1.5 \end{bmatrix} \quad \begin{bmatrix} 0 & 0 \\ 0.5 & 1.0 \\ 1.0 & 0.5 \\ 1.5 & 1.5 \\ 0.5 & 0.0 \\ 1.0 & 1.0 \\ 1.5 & 0.5 \\ 2.0 & 1.5 \end{bmatrix} > [0.5 \quad 1.0] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 1 & 1 \\ 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$



【Exercise1.12】

In the three 3-input neurons as follows,

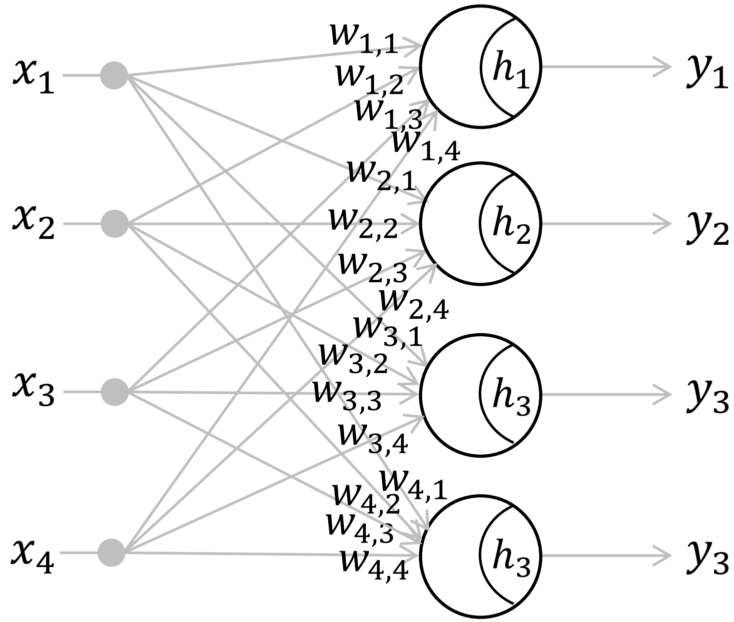


calculate outputs by hand calculation where the inputs, weights and a threshold are given as follows. Then check the answer using Python or MATLAB script.

$$\boldsymbol{X} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} 2 & -1 & 3 \\ 0 & 4 & 1 \\ -4 & 2 & -2 \end{bmatrix} \quad \boldsymbol{h} = \begin{bmatrix} 2 & 0 & 1 \end{bmatrix}$$

【Exercise1.13】

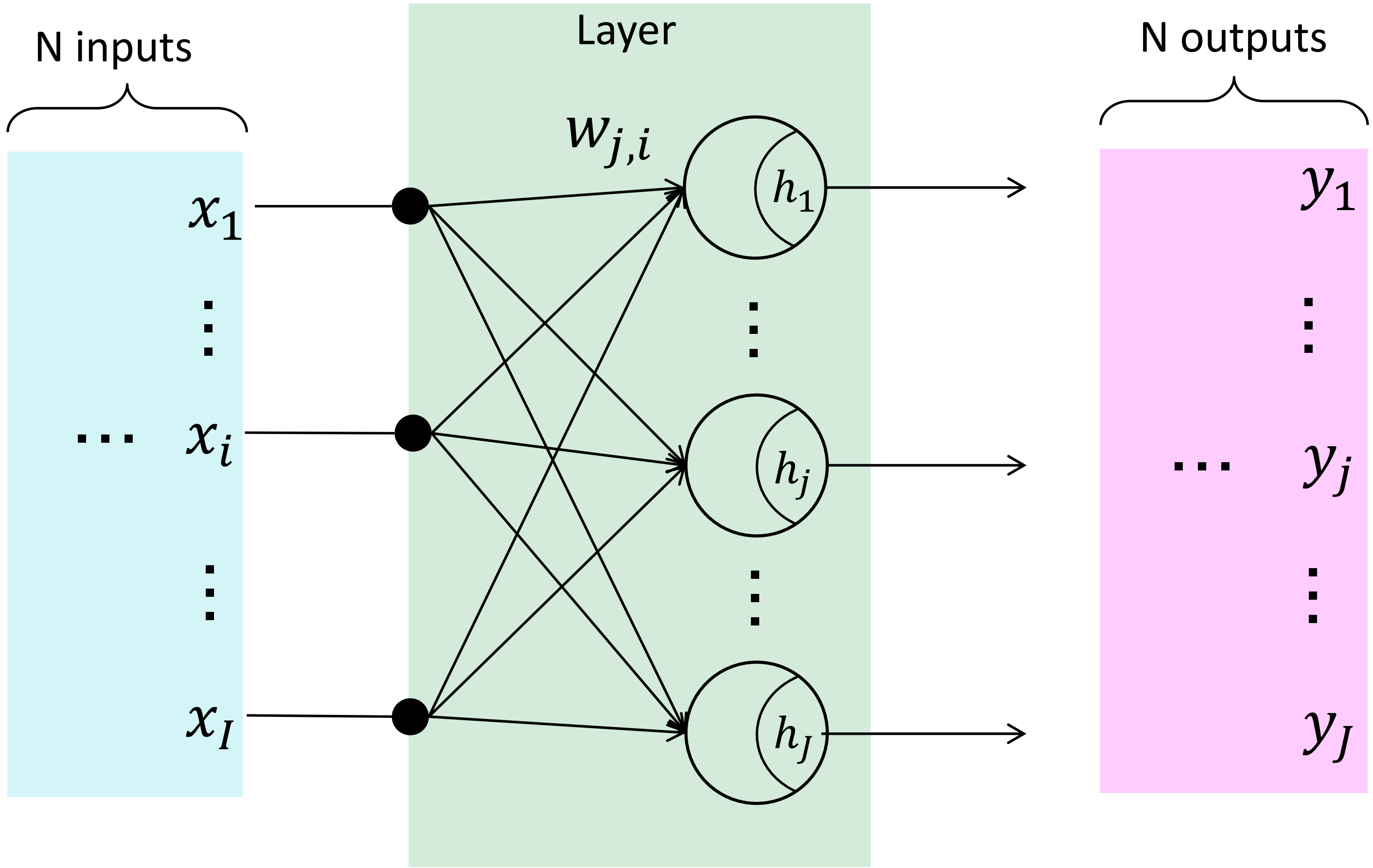
In the four 3-input neurons as follows,



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

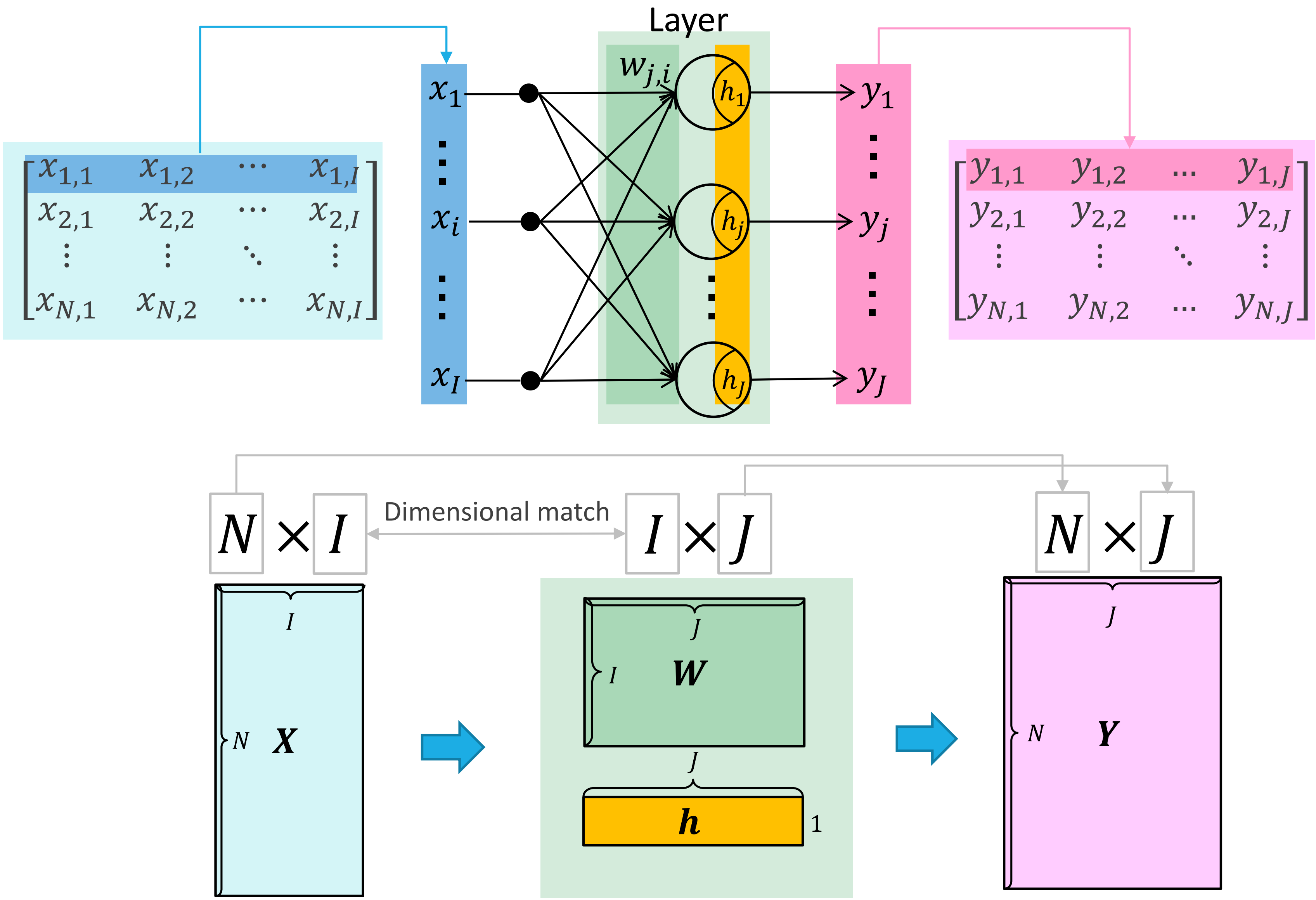
$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} -2 & 0 & 2 & 2 \\ -1 & 2 & 1 & -1 \\ 4 & -3 & 1 & 0 \\ 1 & 1 & 0 & -3 \end{bmatrix} \qquad \boldsymbol{h} = \begin{bmatrix} 1 & 0 & 0 & 2 \end{bmatrix}$$

Layers in Neural Network Model



Layers are made up of a number of interconnected neurons.

Layers in Neural Network Model



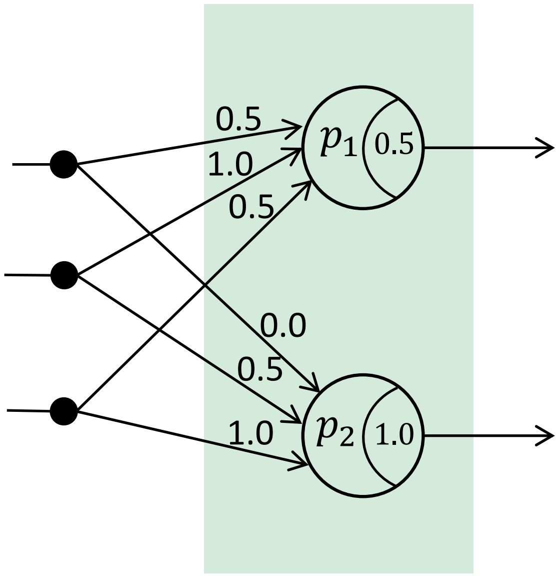
【Example1.7】Implementation using "Class"

Python

```python
import numpy as np

class FormalNeuronLayer:
  def __init__(self, w, h):
    self.w = w
    self.h = h

  def forward(self, x):
    p = np.dot(x, self.w)
    y = p > self.h
    return y.astype(np.int)

x = np.array([[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 = np.array([[0.5, 0.0],
              [1.0, 0.5],
              [0.5, 1.0]])
h = np.array([0.5, 1.0])

formalNeuron = FormalNeuronLayer(w,h)
y = formalNeuron.forward(x)
print(y)
```

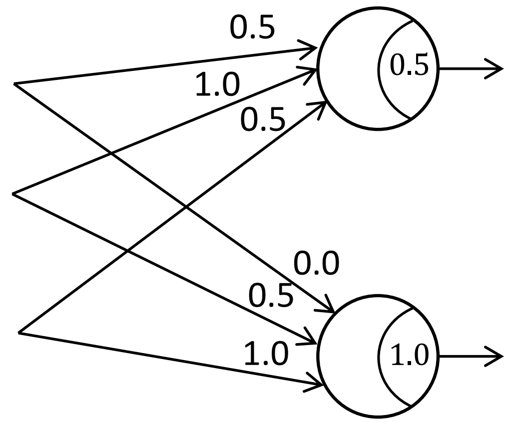
Constructor

Calculate from input to output



【Example1.7】Implementation using "Class"

MATLAB



FormalNeuronLayer.m

```
classdef FormalNeuronLayer < handle
  properties
    weights;
    threshold;
  end

  methods

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

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

  end
end
```

Constructor

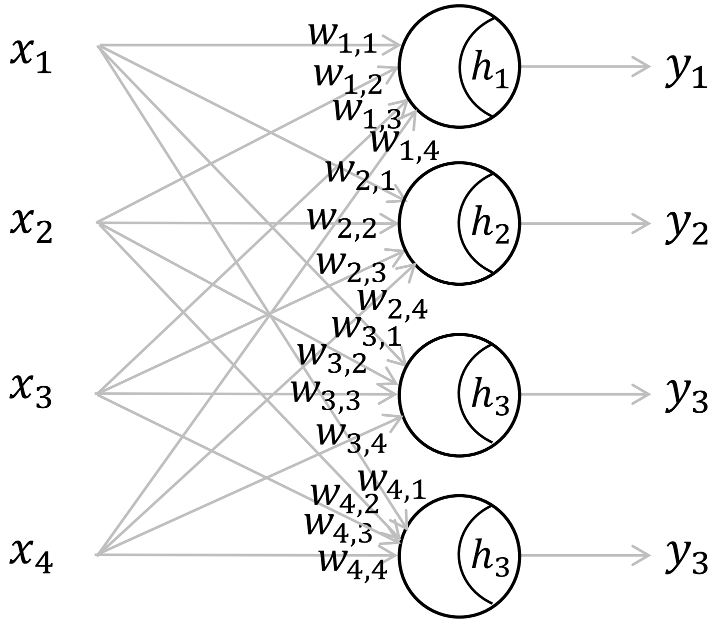
Calculate from input to output

example1_7.m

```
x = [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 = [0.5, 0.0;
     1.0, 0.5;
     0.5, 1.0];
h = [0.5, 1.0];

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

【Exercise1.14】

In the four 3-input neurons as follows,



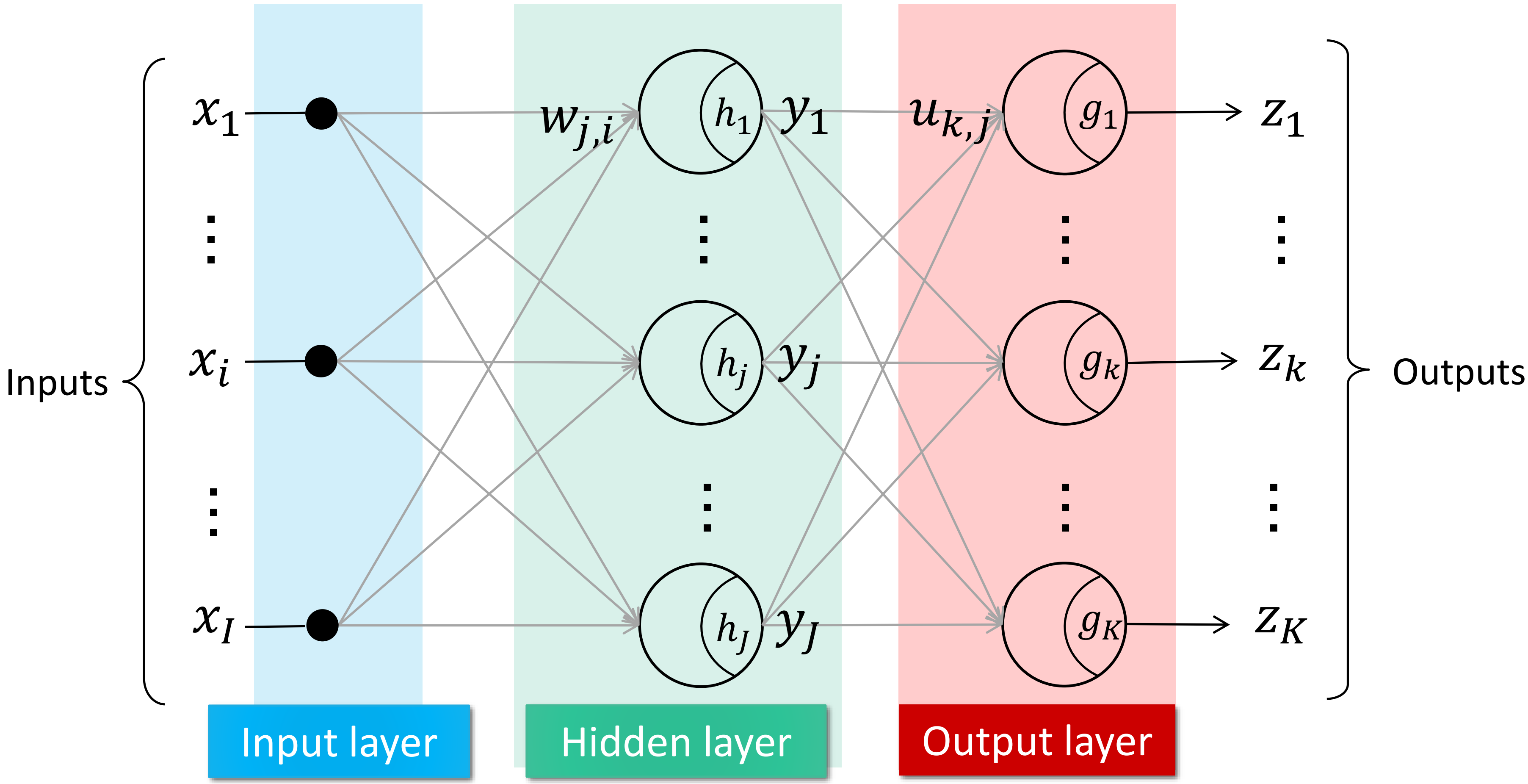
calculate outputs **using *FormalNeuronLayer* class** in Python or MATLAB where the inputs, weights and threshold are the same as in Exercise 1.13, as follows.

Compare the output with Exercise 1.13.

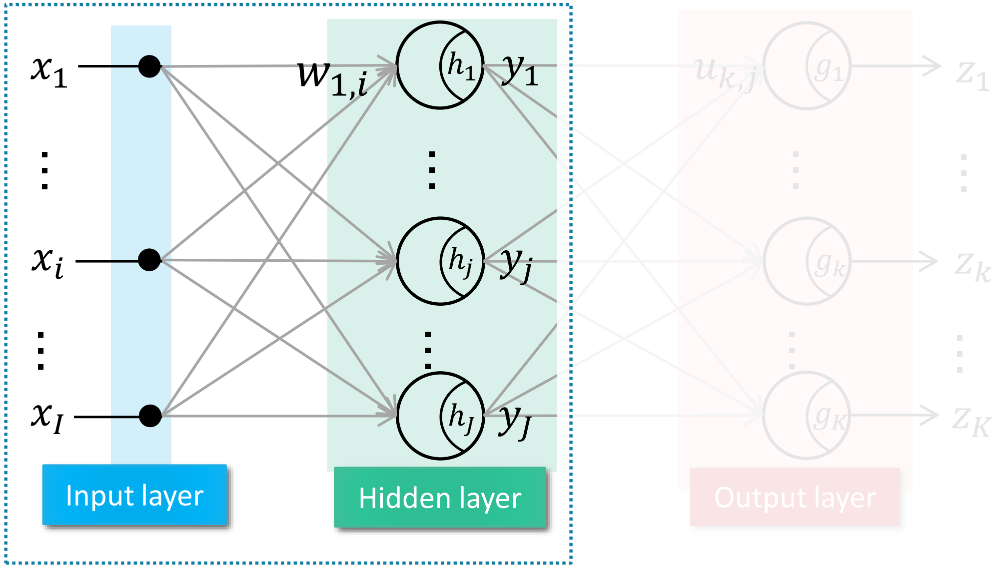
$$\boldsymbol{X} = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix} \qquad \boldsymbol{W} = \begin{bmatrix} -2 & 0 & 2 & 2 \\ -1 & 2 & 1 & -1 \\ 4 & -3 & 1 & 0 \\ 1 & 1 & 0 & -3 \end{bmatrix} \qquad \boldsymbol{h} = \begin{bmatrix} 1 & 0 & 0 & 2 \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)

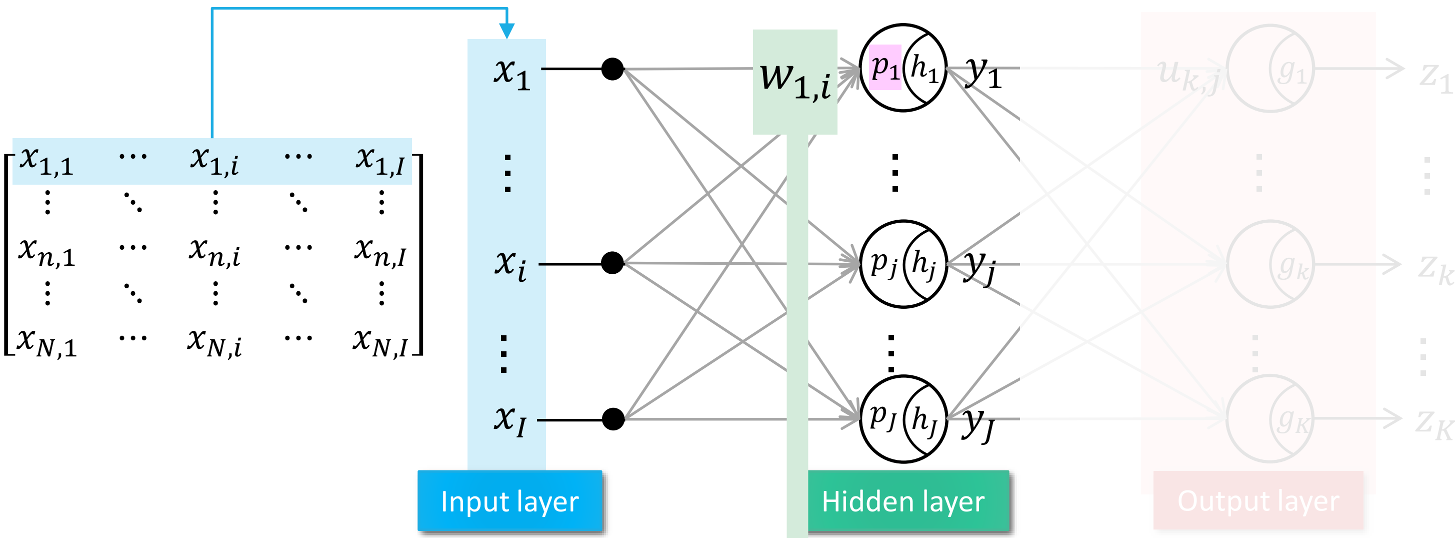
We can construct a three layers neural network as follows. The first layer simply copies the inputs. The outputs of the second layer are the inputs of the third layer.



Multiple Layer Neural Network (Perceptron)



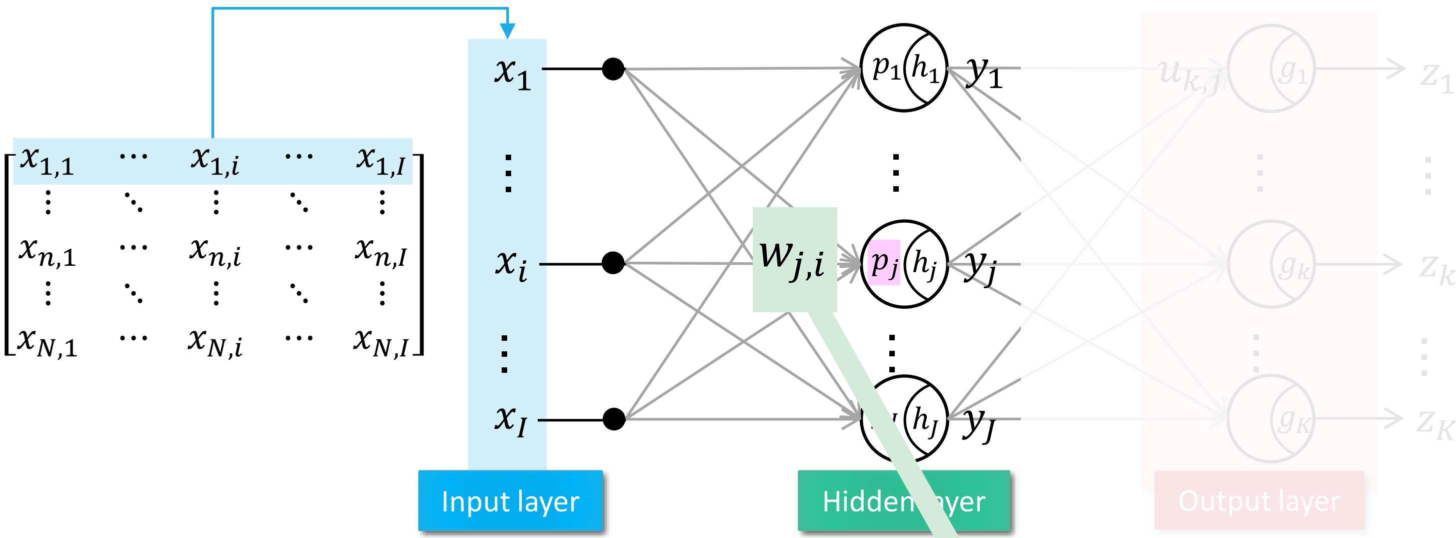
Multiple Layer Neural Network (Perceptron)



dot product (weighted sum)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,i} & \cdots & x_{1,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,i} & \cdots & x_{n,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,i} & \cdots & x_{N,I} \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & \cdots & w_{j,1} & \cdots & w_{J,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,i} & \cdots & w_{j,i} & \cdots & w_{J,i} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,I} & \cdots & w_{j,I} & \cdots & w_{J,I} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,j} & \cdots & p_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,j} & \cdots & p_{N,J} \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)

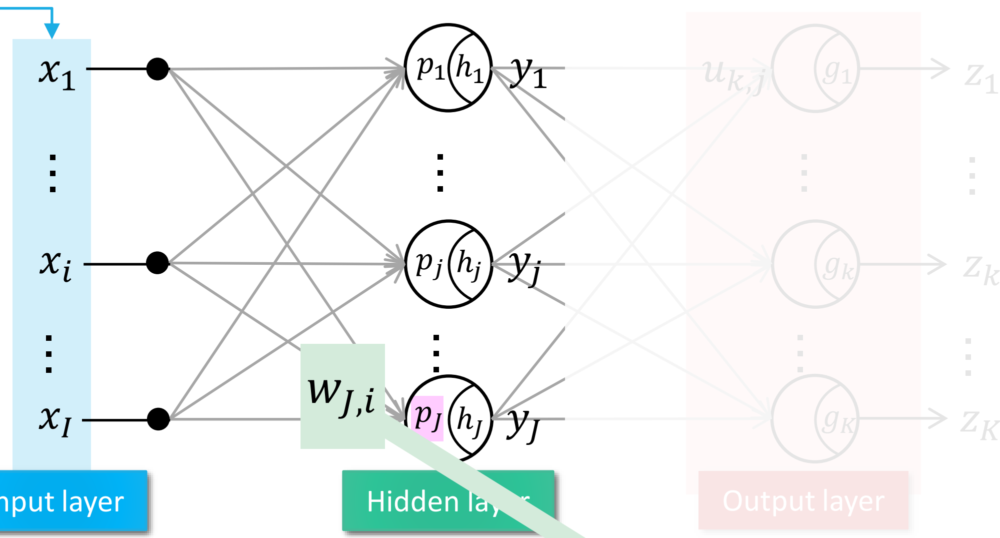


dot product (weighted sum)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,i} & \cdots & x_{1,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,i} & \cdots & x_{n,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,i} & \cdots & x_{N,I} \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & \cdots & w_{j,1} & \cdots & w_{J,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,i} & \cdots & w_{j,i} & \cdots & w_{J,i} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,I} & \cdots & w_{j,I} & \cdots & w_{J,I} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,j} & \cdots & p_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,j} & \cdots & p_{N,J} \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)

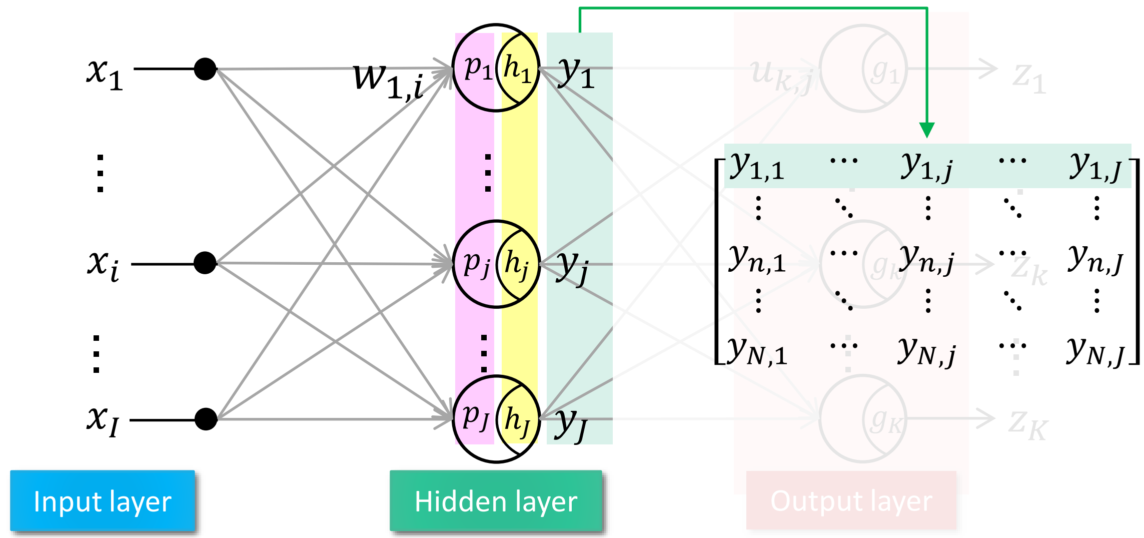
$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,i} & \cdots & x_{1,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,i} & \cdots & x_{n,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,i} & \cdots & x_{N,I} \end{bmatrix}$$



dot product (weighted sum)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,i} & \cdots & x_{1,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,i} & \cdots & x_{n,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,i} & \cdots & x_{N,I} \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & \cdots & w_{j,1} & \cdots & w_{J,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,i} & \cdots & w_{j,i} & \cdots & w_{J,i} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,I} & \cdots & w_{j,I} & \cdots & w_{J,I} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,j} & \cdots & p_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,j} & \cdots & p_{N,J} \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)



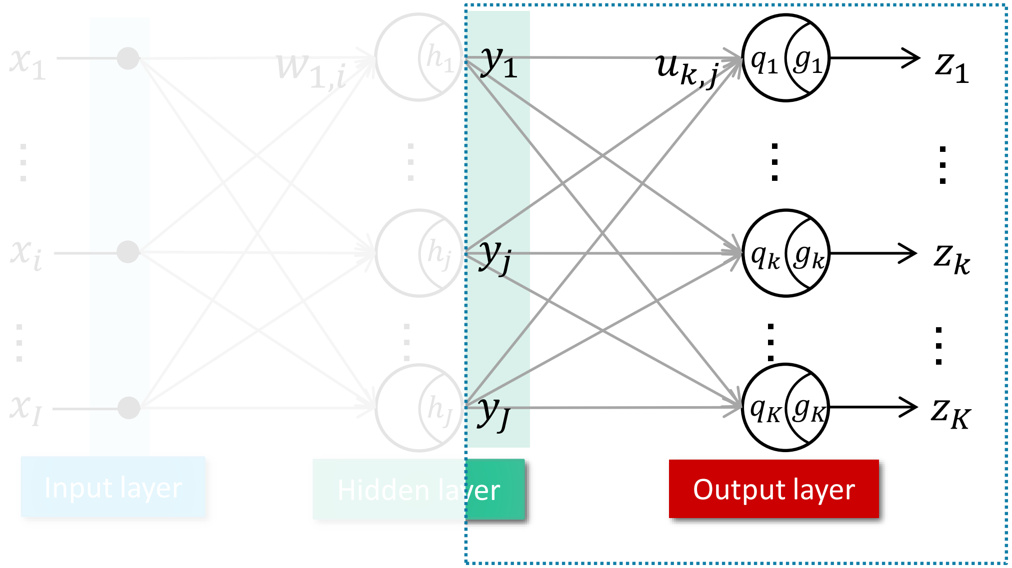
dot product (weighted sum)

$$\begin{bmatrix} x_{1,1} & \cdots & x_{1,i} & \cdots & x_{1,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,i} & \cdots & x_{n,I} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ x_{N,1} & \cdots & x_{N,i} & \cdots & x_{N,I} \end{bmatrix} \cdot \begin{bmatrix} w_{1,1} & \cdots & w_{j,1} & \cdots & w_{J,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,i} & \cdots & w_{j,i} & \cdots & w_{J,i} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{1,I} & \cdots & w_{j,I} & \cdots & w_{J,I} \end{bmatrix} = \begin{bmatrix} p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,j} & \cdots & p_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,j} & \cdots & p_{N,J} \end{bmatrix}$$

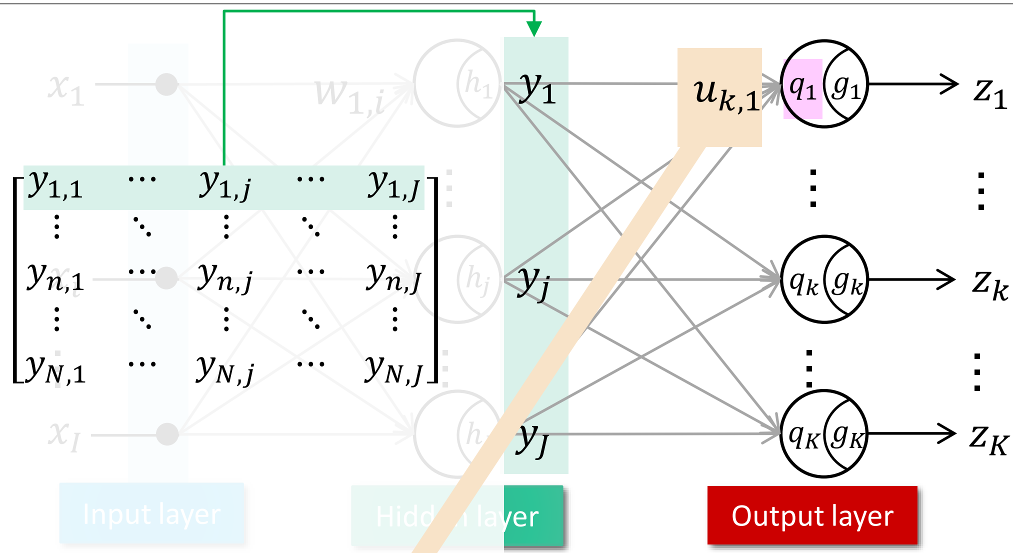
threshold

$$\begin{bmatrix} p_{1,1} & \cdots & p_{1,j} & \cdots & p_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{n,1} & \cdots & p_{n,j} & \cdots & p_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_{N,1} & \cdots & p_{N,j} & \cdots & p_{N,J} \end{bmatrix} > \begin{bmatrix} h_1 & \cdots & h_j & \cdots & h_J \end{bmatrix} = \begin{bmatrix} y_{1,1} & \cdots & y_{1,j} & \cdots & y_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{n,1} & \cdots & y_{n,j} & \cdots & y_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,j} & \cdots & y_{N,J} \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)



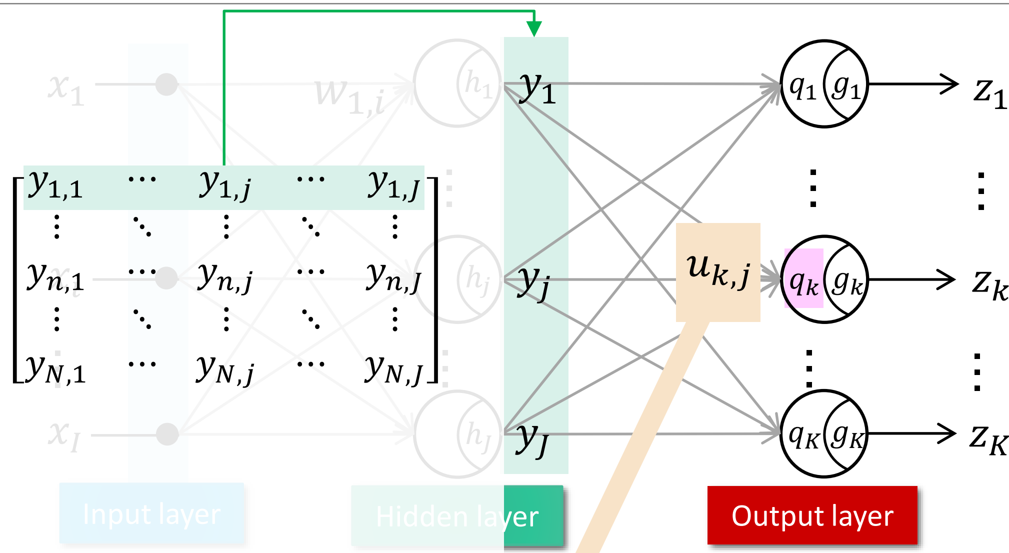
Multiple Layer Neural Network (Perceptron)



dot product (weighted sum)

$$\begin{bmatrix} y_{1,1} & \cdots & y_{1,j} & \cdots & y_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{n,1} & \cdots & y_{n,j} & \cdots & y_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,j} & \cdots & j_{N,J} \end{bmatrix} \cdot \begin{bmatrix} u_{1,1} & \cdots & u_{k,1} & \cdots & u_{K,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,j} & \cdots & u_{k,j} & \cdots & u_{K,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,J} & \cdots & u_{k,J} & \cdots & u_{K,J} \end{bmatrix} = \begin{bmatrix} q_{1,1} & \cdots & q_{1,k} & \cdots & q_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,k} & \cdots & q_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{N,1} & \cdots & q_{N,k} & \cdots & q_{N,K} \end{bmatrix}$$

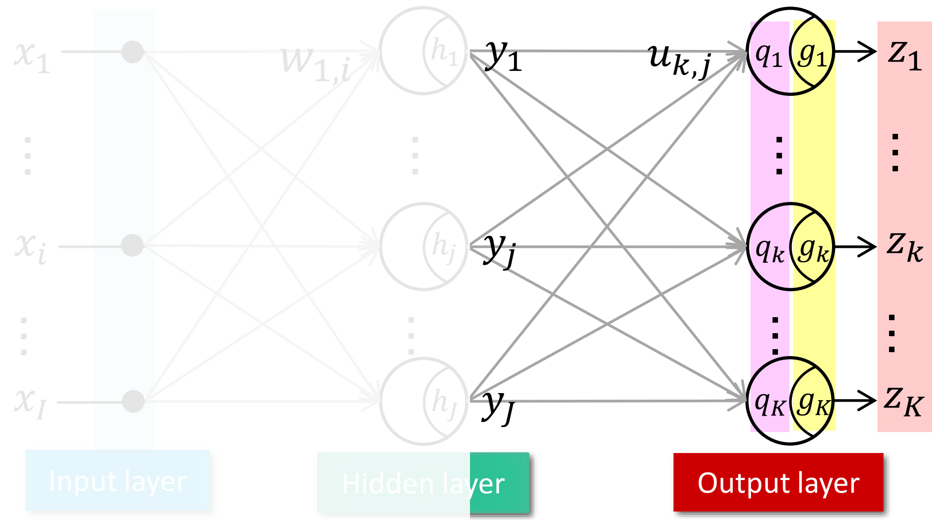
Multiple Layer Neural Network (Perceptron)



dot product (weighted sum)

$$\begin{bmatrix} y_{1,1} & \cdots & y_{1,j} & \cdots & y_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{n,1} & \cdots & y_{n,j} & \cdots & y_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,j} & \cdots & j_{N,J} \end{bmatrix} \cdot \begin{bmatrix} u_{1,1} & \cdots & u_{k,1} & \cdots & u_{K,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,j} & \cdots & u_{k,j} & \cdots & u_{K,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,J} & \cdots & u_{k,J} & \cdots & u_{K,J} \end{bmatrix} = \begin{bmatrix} q_{1,1} & \cdots & q_{1,k} & \cdots & q_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,k} & \cdots & q_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{N,1} & \cdots & q_{N,k} & \cdots & q_{N,K} \end{bmatrix}$$

Multiple Layer Neural Network (Perceptron)



$$\begin{bmatrix} z_{1,1} & \cdots & z_{1,k} & \cdots & z_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{n,1} & \cdots & z_{n,k} & \cdots & z_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{N,k} & \cdots & z_{N,K} \end{bmatrix}$$

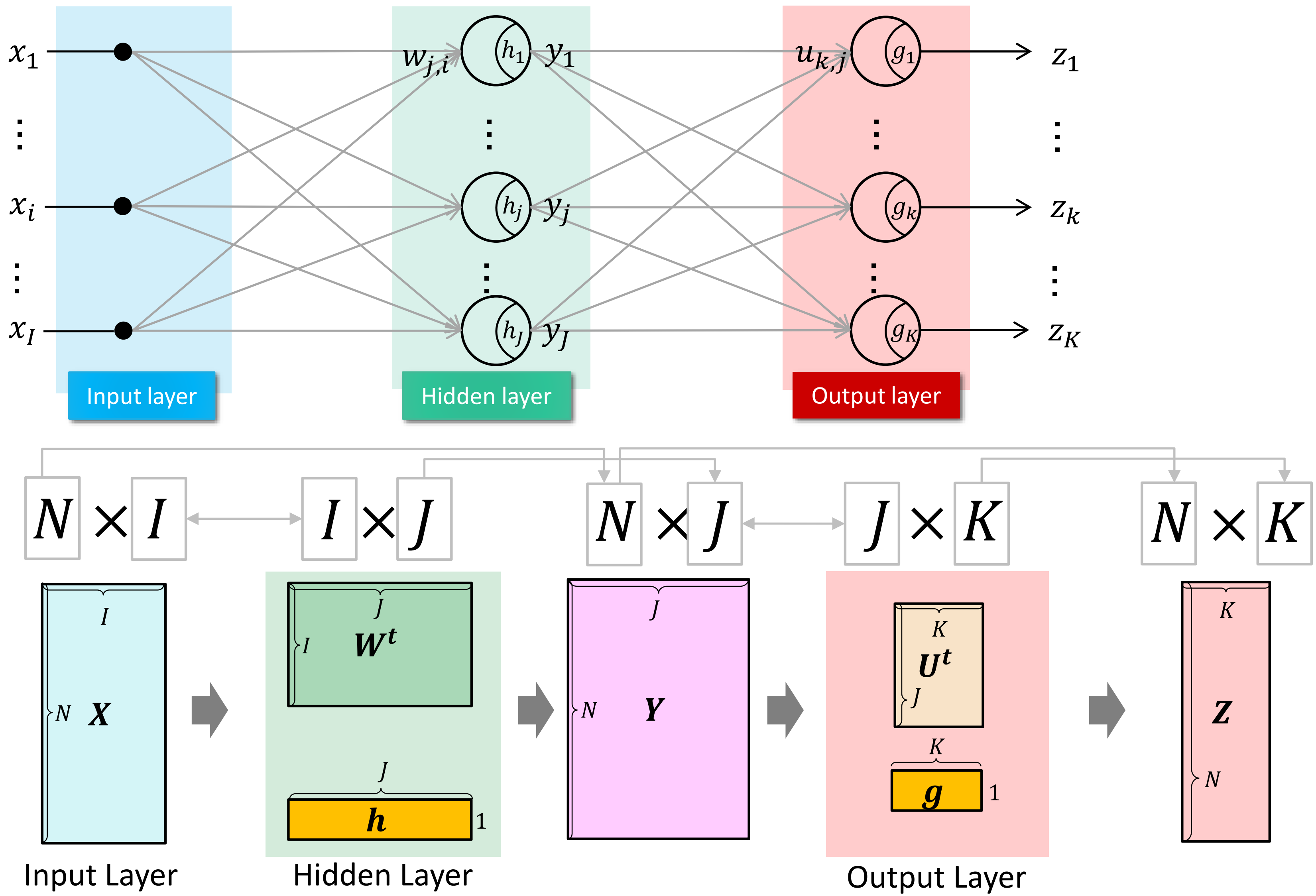
dot product (weighted sum)

$$\begin{bmatrix} y_{1,1} & \cdots & y_{1,j} & \cdots & y_{1,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{n,1} & \cdots & y_{n,j} & \cdots & y_{n,J} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ y_{N,1} & \cdots & y_{N,j} & \cdots & j_{N,J} \end{bmatrix} \cdot \begin{bmatrix} u_{1,1} & \cdots & u_{k,1} & \cdots & u_{K,1} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,j} & \cdots & u_{k,j} & \cdots & u_{K,j} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ u_{1,J} & \cdots & u_{k,J} & \cdots & u_{K,J} \end{bmatrix} = \begin{bmatrix} q_{1,1} & \cdots & q_{1,k} & \cdots & q_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,k} & \cdots & q_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{N,1} & \cdots & q_{N,k} & \cdots & q_{N,K} \end{bmatrix}$$

threshold

$$\begin{bmatrix} q_{1,1} & \cdots & q_{1,k} & \cdots & q_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{n,1} & \cdots & q_{n,k} & \cdots & q_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ q_{N,1} & \cdots & q_{N,k} & \cdots & q_{N,K} \end{bmatrix} > [g_1 \ \cdots \ g_k \ \cdots \ g_K] = \begin{bmatrix} z_{1,1} & \cdots & z_{1,k} & \cdots & z_{1,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{n,1} & \cdots & z_{n,k} & \cdots & z_{n,K} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ z_{N,1} & \cdots & z_{N,k} & \cdots & z_{N,K} \end{bmatrix}$$

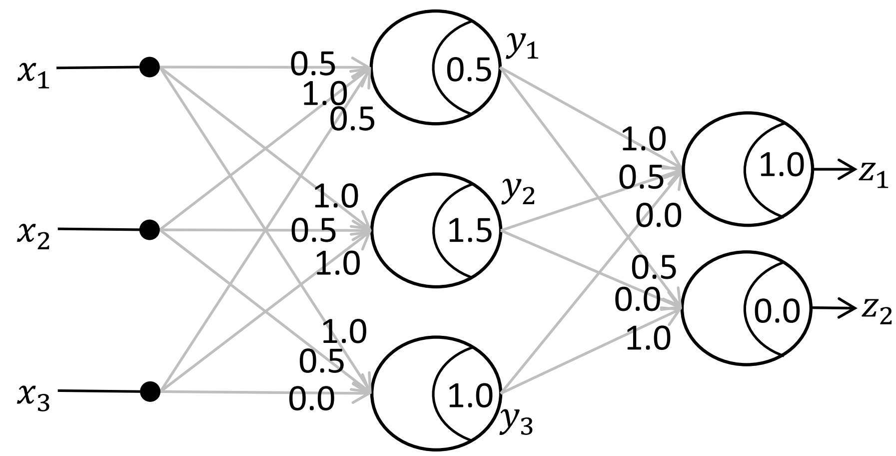
Multiple Layer Neural Network (Perceptron)



【Example1.8】Let's implement "Perceptron"

Python

```python
import numpy as np

class FormalNeuronLayer:
  def __init__(self, w, h):
    self.w = w
    self.h = h

  def forward(self, x):
    p = np.dot(x, self.w)
    y = p > self.h
    return y.astype(np.int)

x = np.array([[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 = np.array([[0.5, 1.0, 1.0],
              [1.0, 0.5, 0.5],
              [0.5, 1.0, 0.0]])
h = np.array([0.5, 1.5, 1.0])
u = np.array([[1.0, 0.5],
              [0.5, 0.0],
              [0.0, 1.0]])
g = np.array([1.0, 0.0])

formalNeuron1 = FormalNeuronLayer(w,h)
formalNeuron2 = FormalNeuronLayer(u,g)
y = formalNeuron1.forward(x)
print(y)
z = formalNeuron2.forward(y)
print(z)
```



Execution and results

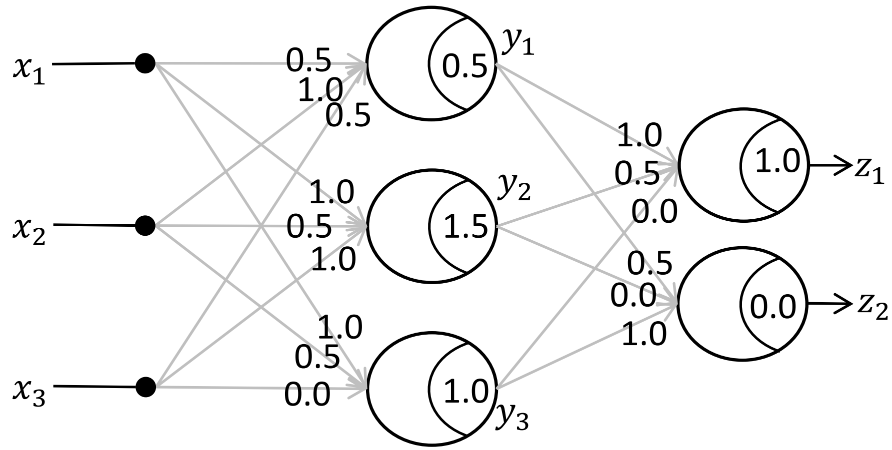
```
[[0 0 0]
 [0 0 0]
 [1 0 0]
 [1 0 0]
 [0 0 0]
 [1 1 0]
 [1 0 1]
 [1 1 1]]
[[0 0]
 [0 0]
 [0 1]
 [0 1]
 [0 0]
 [1 1]
 [0 1]
 [1 1]]
```

【Example1.8】Let’s implement “Perceptron”

MATLAB

example1_8.m

```
x = [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 = [0.5, 1.0, 1.0;
     1.0, 0.5, 0.5;
     0.5, 1.0, 0.0];
h = [0.5, 1.5, 1.0];
u = [1.0, 0.5;
     0.5, 0.0;
     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)
```

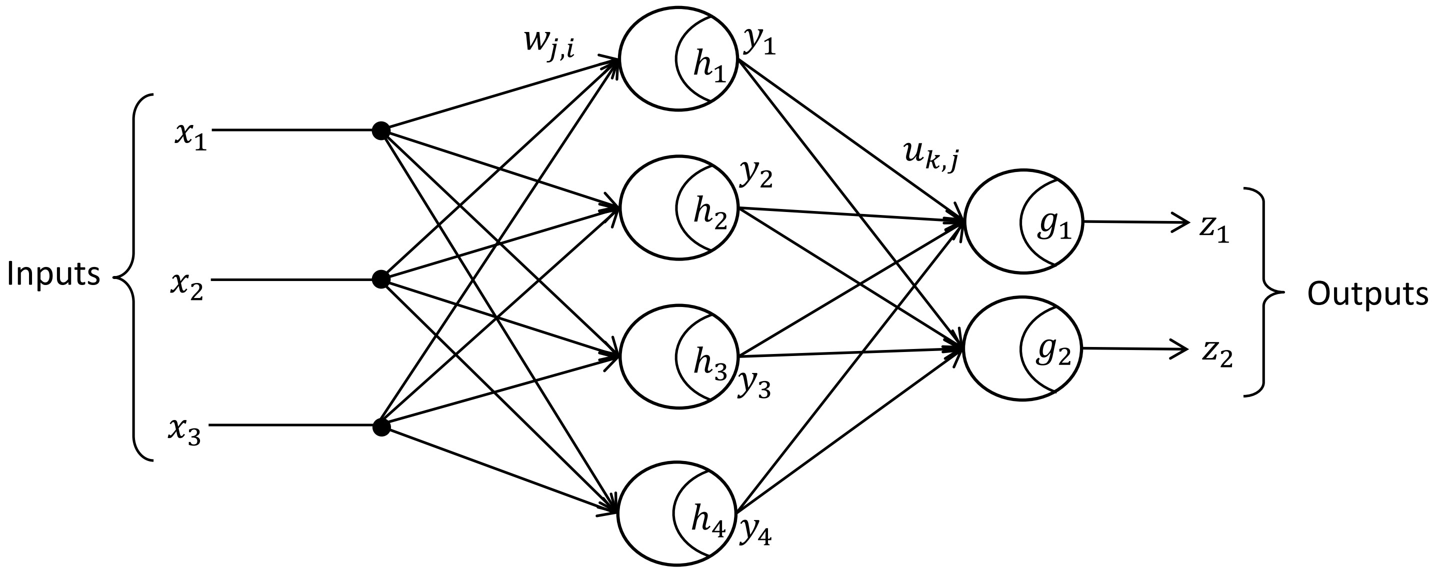


Execution and results

```
>> example1_8
y =

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

z =

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

【Exercise1.15】

In the following neural network, calculate outputs both by hand-calculation and by using Python or MATLAB scripts where the inputs $\boldsymbol{X}$, weights $\boldsymbol{W}, \boldsymbol{U}$ and thresholds $\boldsymbol{h}, \boldsymbol{g}$ are given as followsn.

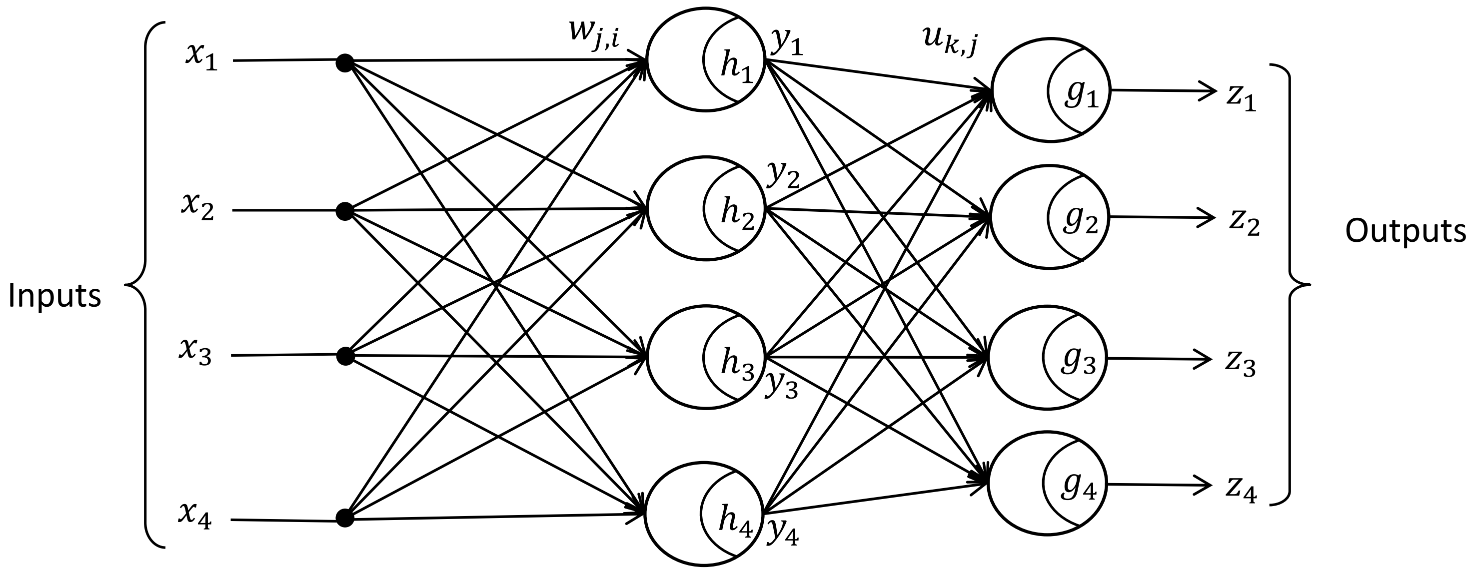


$$\boldsymbol{x} = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{w} = \begin{bmatrix} -2 & 1 & 0 & 1 \\ 0 & 1 & -2 & -1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \quad \boldsymbol{u} = \begin{bmatrix} 4 & 1 \\ 2 & 3 \\ 0 & 1 \\ 3 & 1 \end{bmatrix}$$

$$\boldsymbol{h} = \begin{bmatrix} 2 & -1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{g} = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

【Exercise1.16】

In the following neural network, calculate outputs both by hand-calculation and by using Python or MATLAB scripts where the inputs $\boldsymbol{X}$, weights $\boldsymbol{W}, \boldsymbol{U}$ and thresholds $\boldsymbol{h}, \boldsymbol{g}$ are given as followsn.



$$\boldsymbol{X} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} 3 & 2 & 2 & 0 \\ 4 & 1 & 5 & 2 \\ 1 & 0 & 1 & 4 \\ 0 & 1 & 0 & 1 \end{bmatrix} \quad \boldsymbol{U} = \begin{bmatrix} 4 & 1 & 4 & 3 \\ 2 & 3 & 0 & 1 \\ 0 & 1 & 2 & 4 \\ 3 & 1 & 1 & 2 \end{bmatrix}$$

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