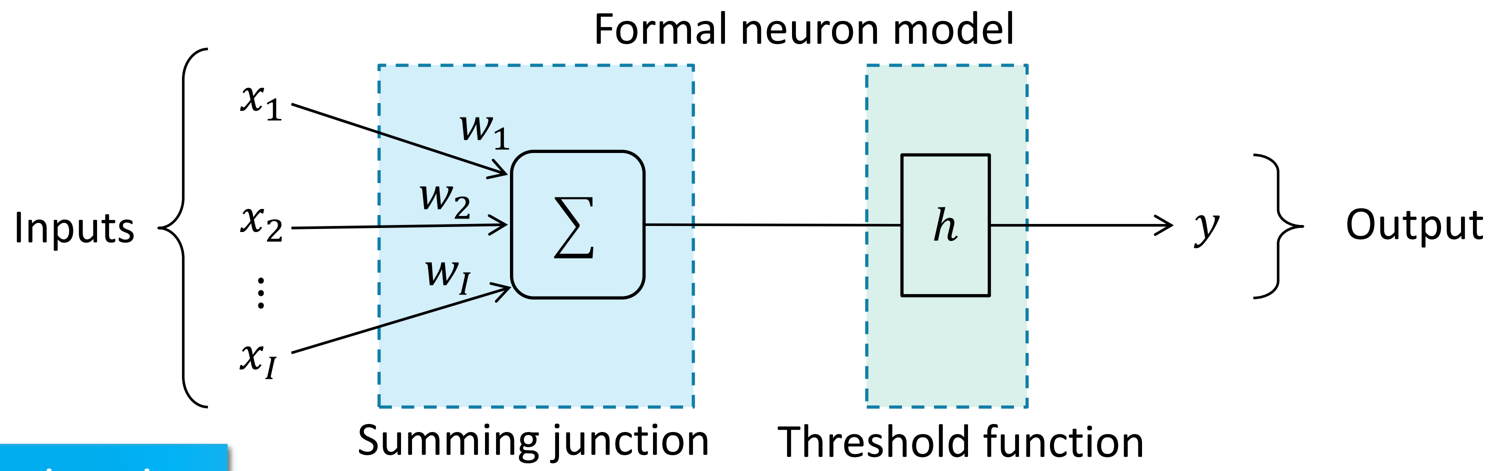
Day2

Making Logic Functions Using Neurons and How to Determine the Weights

【review】Formal Neuron (McCulloch-Pitts Model)



Summing junction

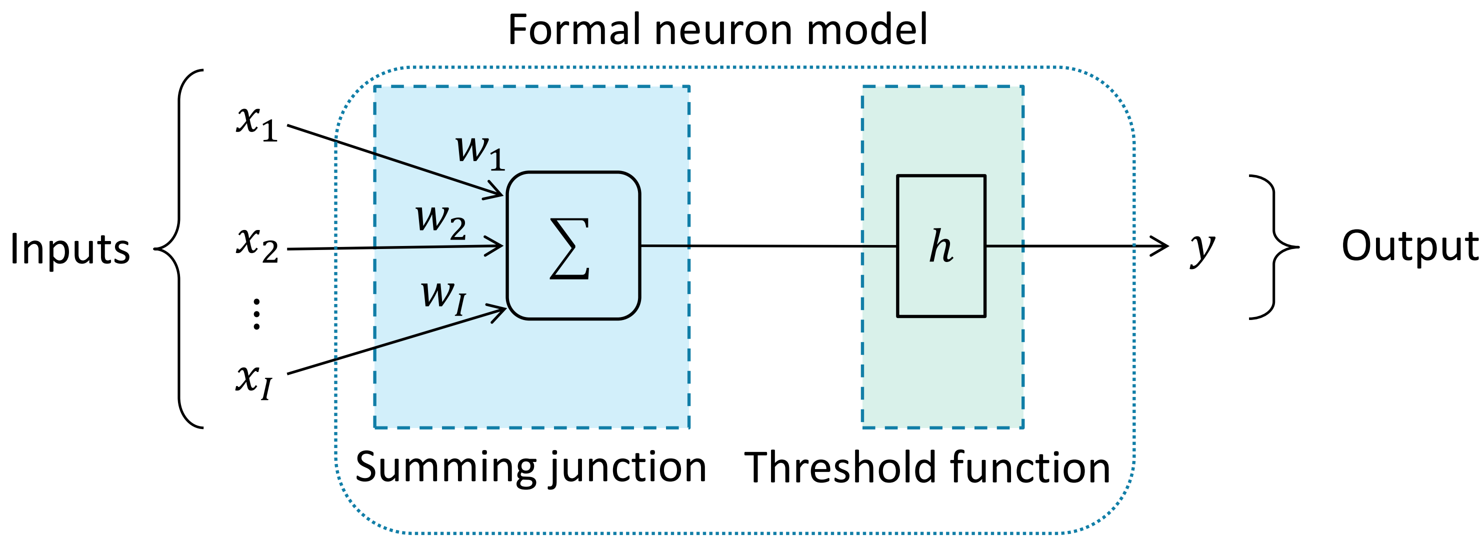
Each inputs $x_1, x_2, \cdots, x_I$ are multiplied by its own weight $w_1, w_2, \cdots, w_I$ respectively. Then a weighted sum value of them (I.e., $\sum_i w_i x_i$) is calculated at summing junction.

Threshold function

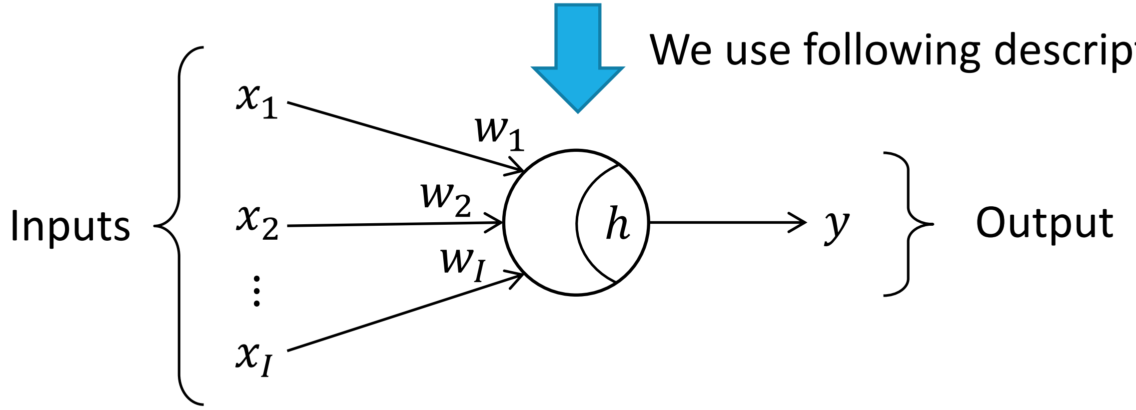
If the weighted sum value is greater than a threshold h, the output y becomes 1. If not, the output y becomes 0. That is,

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

【review】Formal Neuron (McCulloch-Pitts Model)

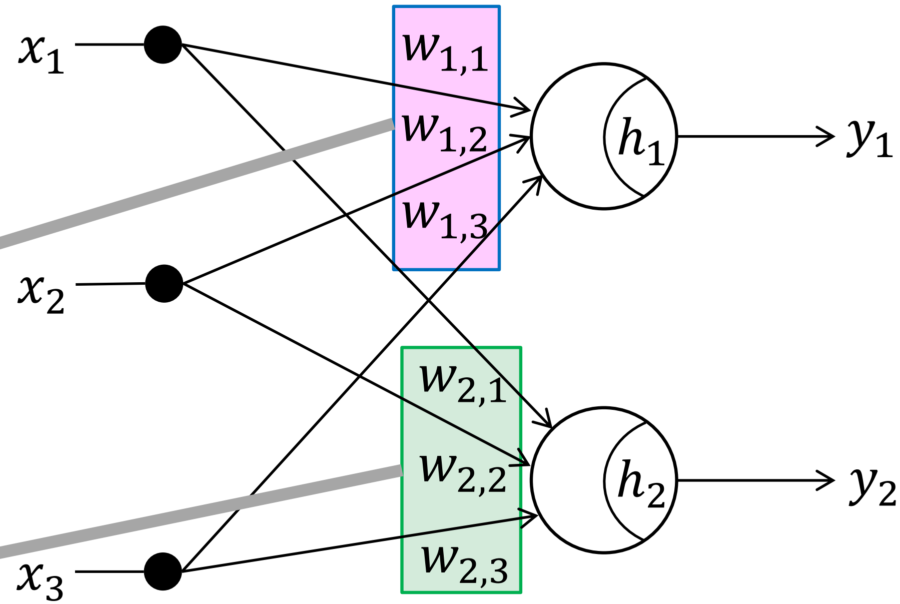


We use following description



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

【review】Feedforward calculation (1)



correspond to
the first row

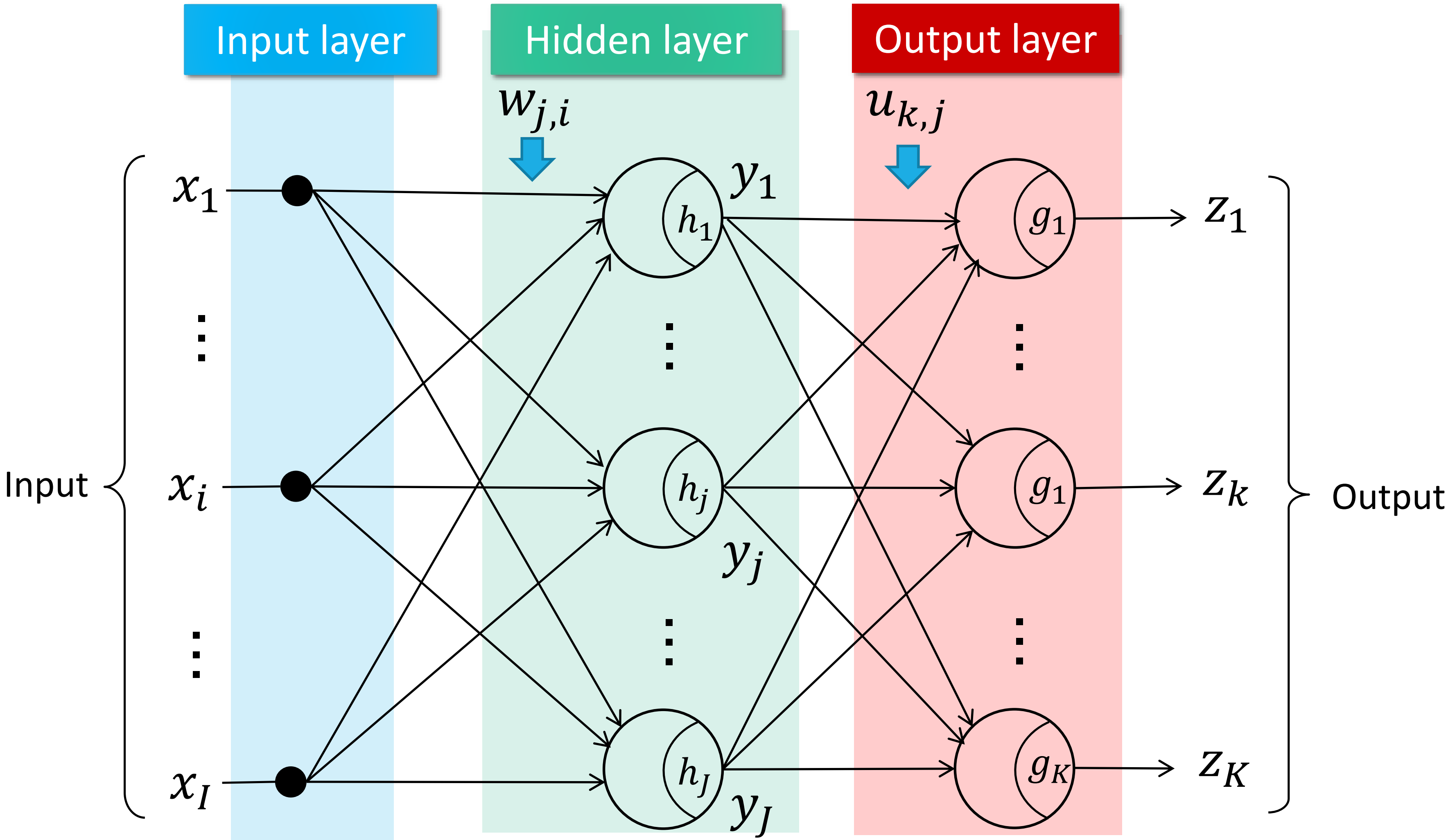
correspond to
the second row

Weight matrix

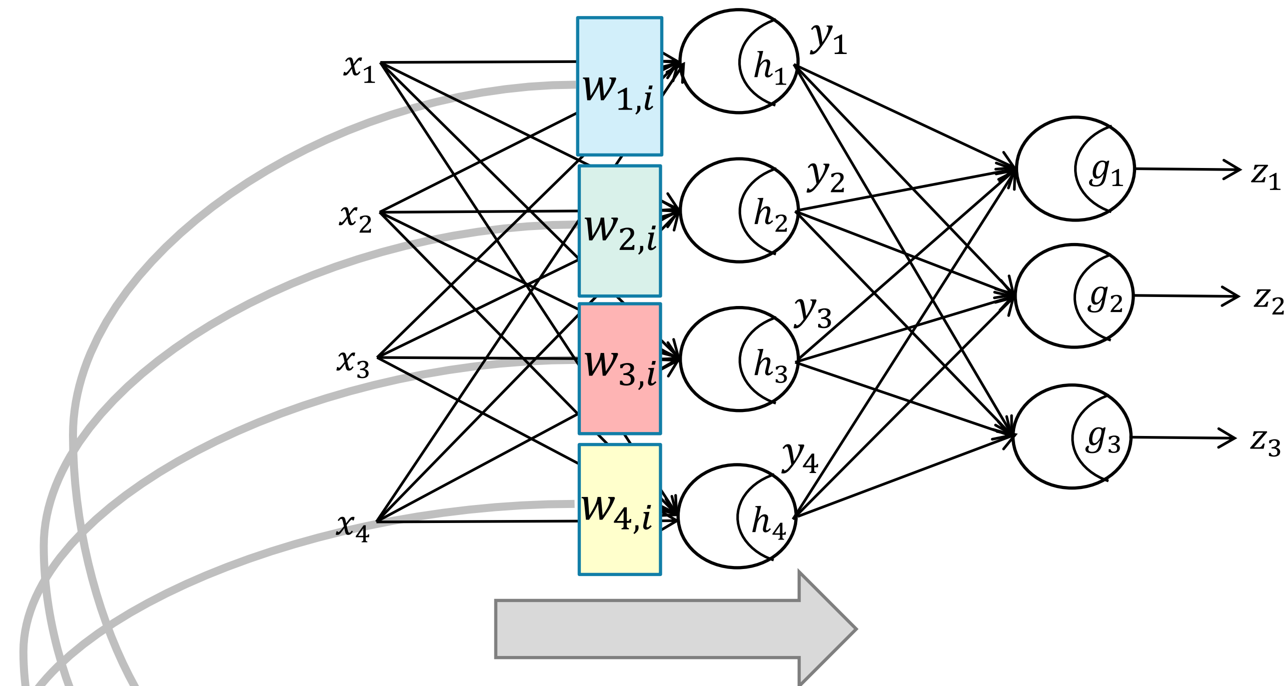
$$\underbrace{\begin{bmatrix} \boxed{w_{1,1}, w_{1,2}, w_{1,3}} \\ \boxed{w_{2,1}, w_{2,2}, w_{2,3}} \end{bmatrix} \begin{bmatrix} x_{1,1} \\ x_{2,1}, \\ x_{3,1} \end{bmatrix}}_{\text{Calculation for Summing junction}} \underbrace{> \begin{bmatrix} h_1 \\ h_2 \end{bmatrix}}_{\text{Calculation for Threshold function}} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

【review】Multiple Layer Neural Network (Perceptron)

Generally, three layer neural network is often used.

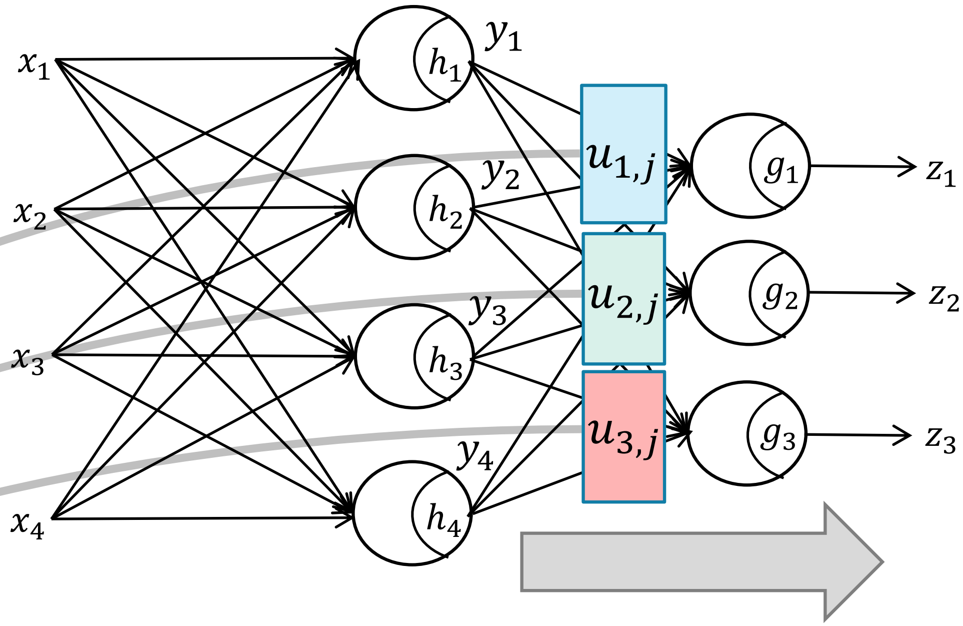


【review】Feedforward calculation (1)



$$\begin{bmatrix} w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} \\ w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} \\ w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} \\ w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} > \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix} \Rightarrow \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

【review】Feedforward calculation (2)

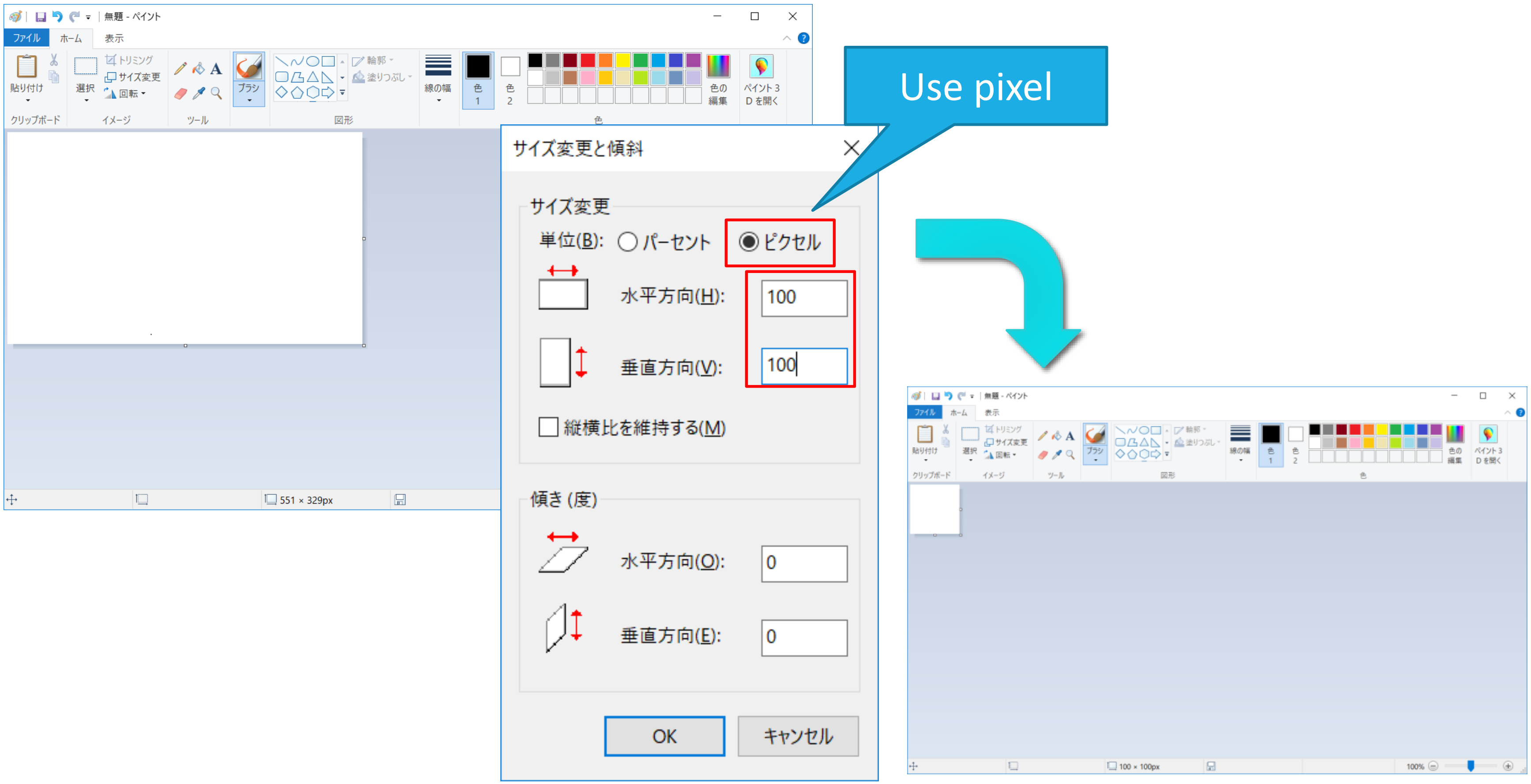


$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ u_{2,1} & u_{2,2} & u_{2,3} & u_{2,4} \\ u_{3,1} & u_{3,2} & u_{3,3} & u_{3,4} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} > \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} \Rightarrow \begin{bmatrix} z_1 \\ z_2 \\ z_3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

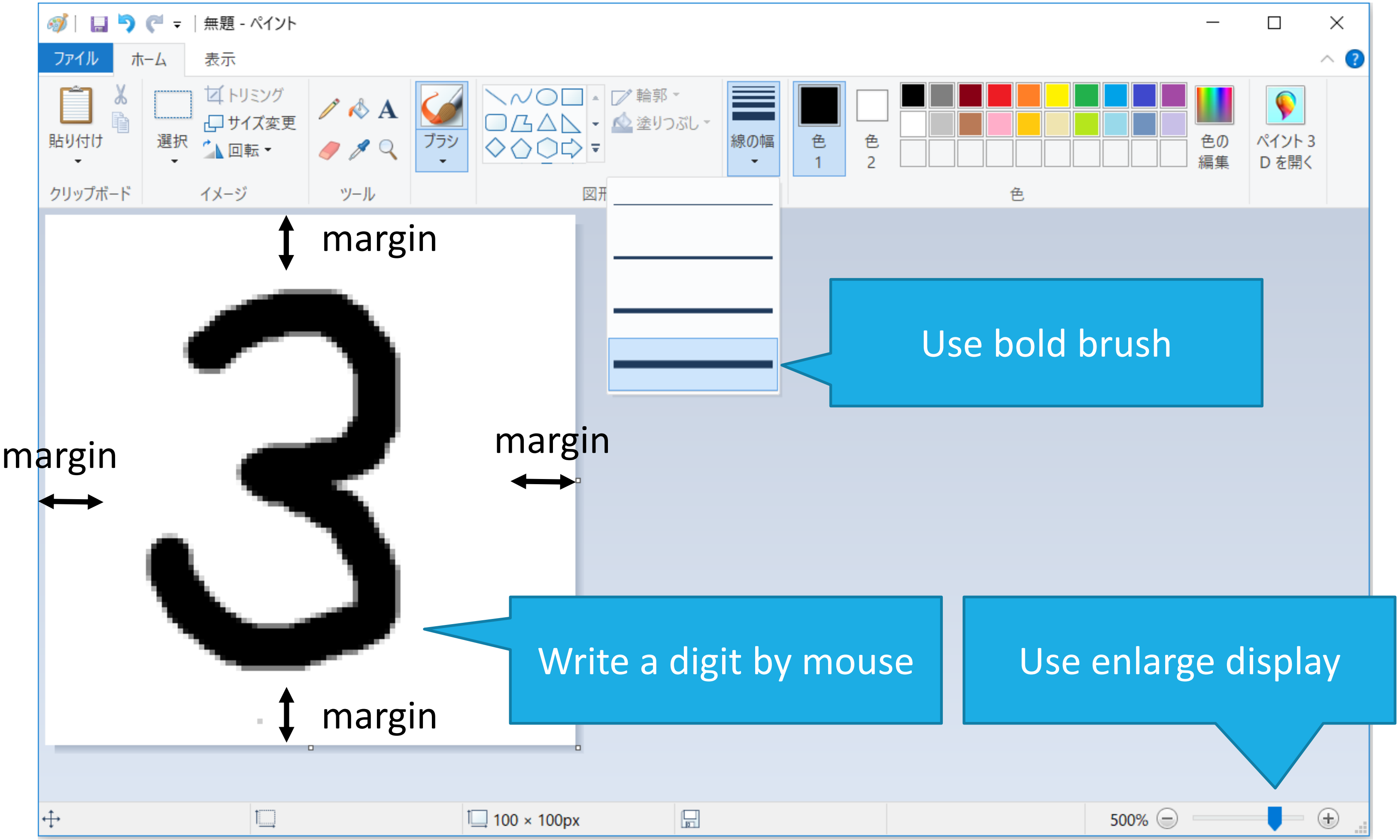
Making Your Own Hand-Written Digit Images

Making your own hand-written digit images

Start “MS Paint” and set the resolution to 100x100px. (To write digits easily, we use 100x100px now. The image size will be reduced to 28x28px using a script when we import them to MATLAB).



Making your hand-written digit images

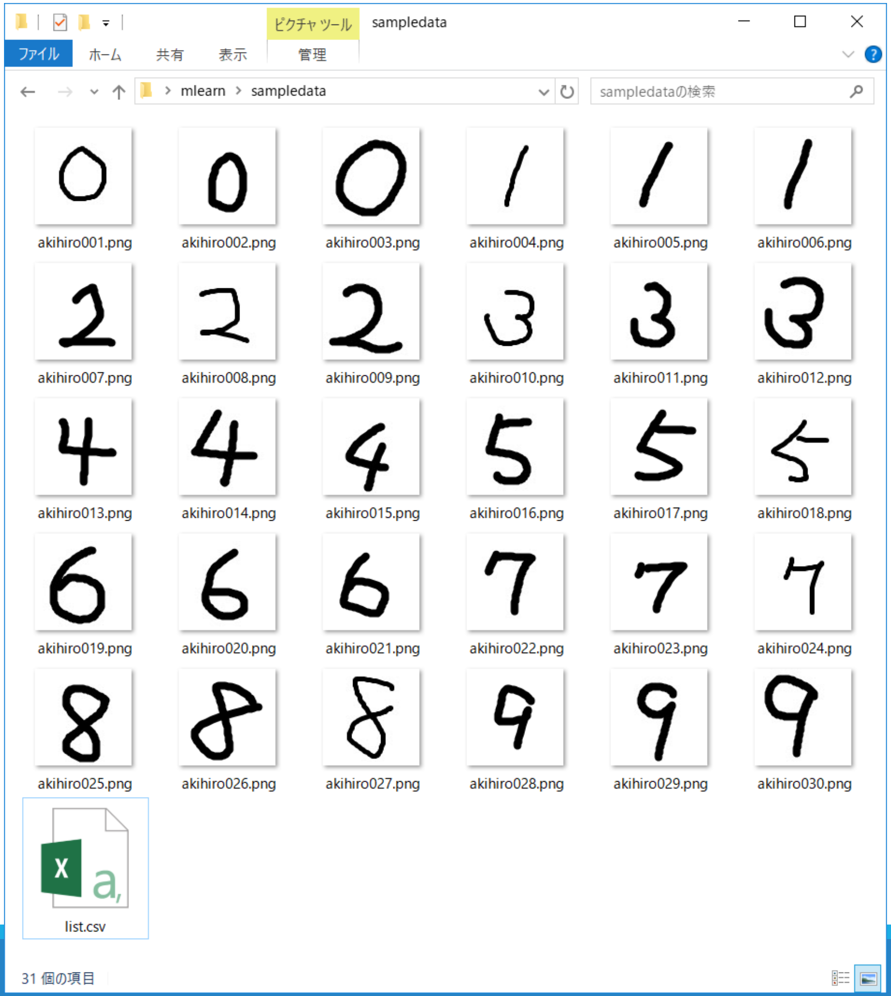


Save as “png file”.

Example data

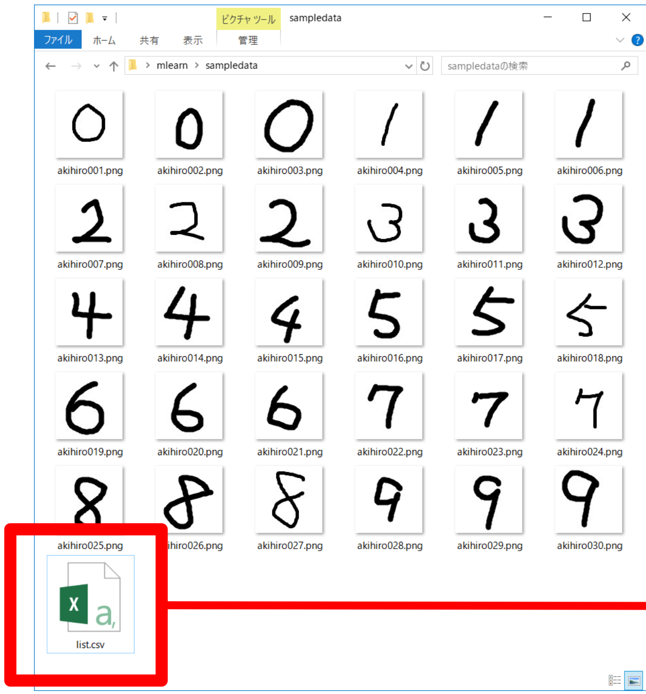
Please make 3 images per each digit.

[first name]001.png - [first name]030.png “akihiro001.png” for example

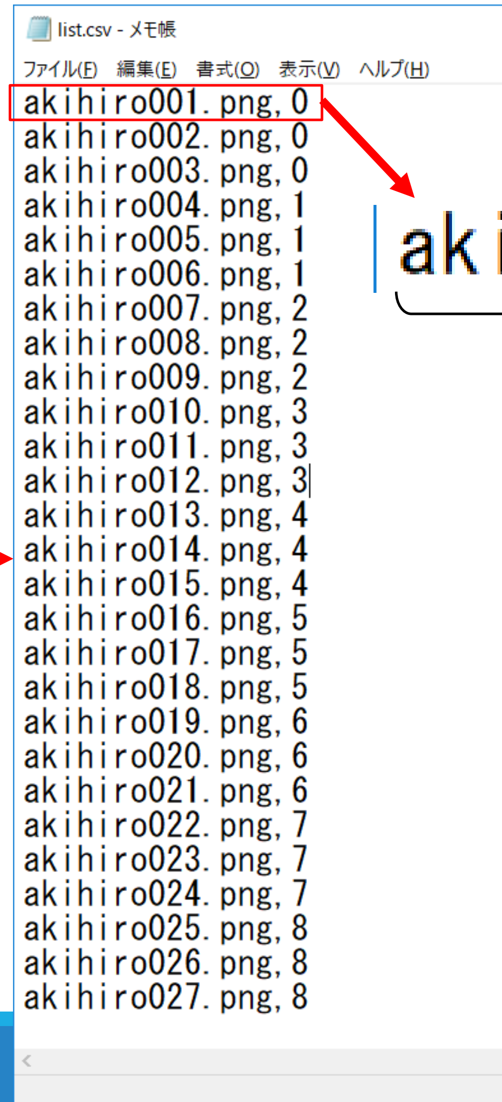


Example data

Please make list file (csv file) too.

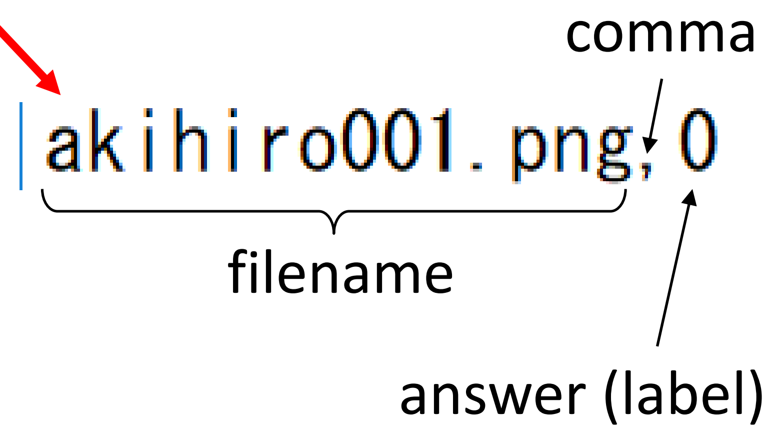


[firstname]_list.csv


list.csv - メモ帳

ファイル(F) 編集(E) 書式(O) 表示(V) ヘルプ(H)

```
akihiro001.png,0
akihiro002.png,0
akihiro003.png,0
akihiro004.png,1
akihiro005.png,1
akihiro006.png,1
akihiro007.png,2
akihiro008.png,2
akihiro009.png,2
akihiro010.png,3
akihiro011.png,3
akihiro012.png,3
akihiro013.png,4
akihiro014.png,4
akihiro015.png,4
akihiro016.png,5
akihiro017.png,5
akihiro018.png,5
akihiro019.png,6
akihiro020.png,6
akihiro021.png,6
akihiro022.png,7
akihiro023.png,7
akihiro024.png,7
akihiro025.png,8
akihiro026.png,8
akihiro027.png,8
```



How to submit them

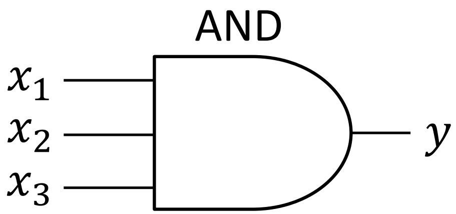
Please make zip file including the image data and csv list file.

Then, submit the zip file to Assignments on "https://oma.metropolia.fi" by 20:00 on Wednesday.

Making Logic Functions Using Neurons

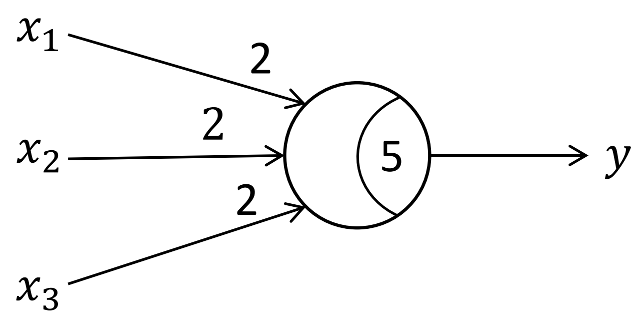
Representation of logic functions using formal neurons (1)

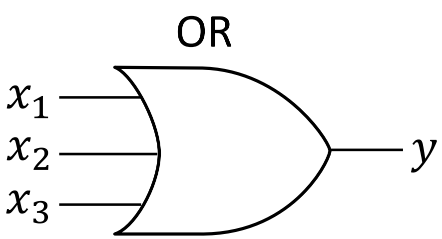
Some of basic logic functions can be constructed with a formal neuron.



Truth table for AND

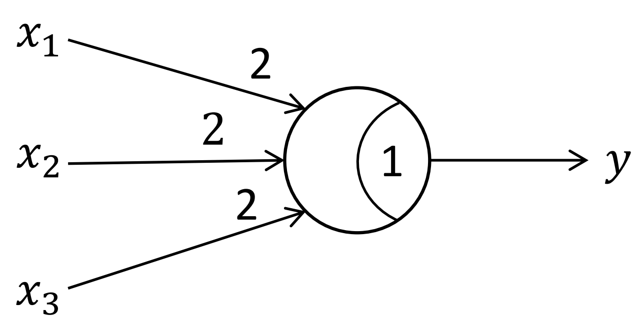
x_1	x_2	x_3	y
0	0	0	**0**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**





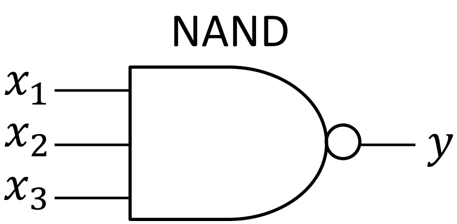
Truth table for OR

x_1	x_2	x_3	y
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**



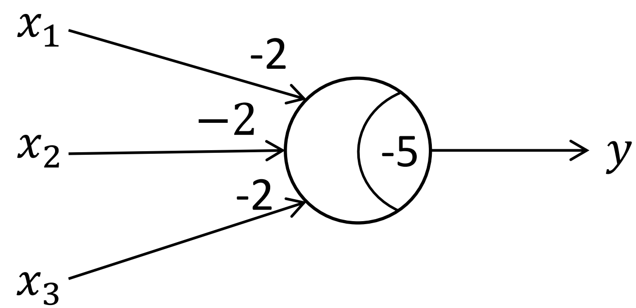
Representation of logic functions using formal neurons (2)

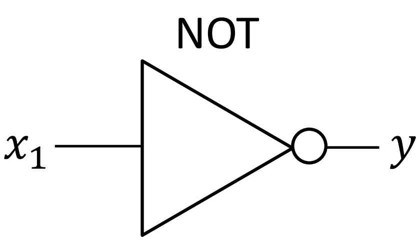
NAND and NOT gate can be constructed by using a negative values as weights and a threshold.



Truth table for NAND

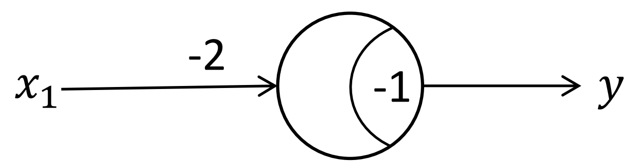
x_1	x_2	x_3	y
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0





Truth table for NOT

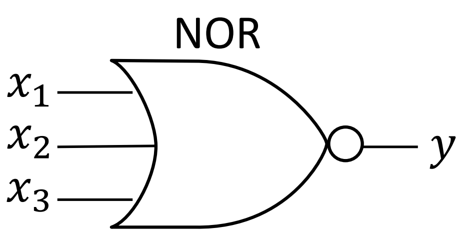
x_1	y
0	1
1	0



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

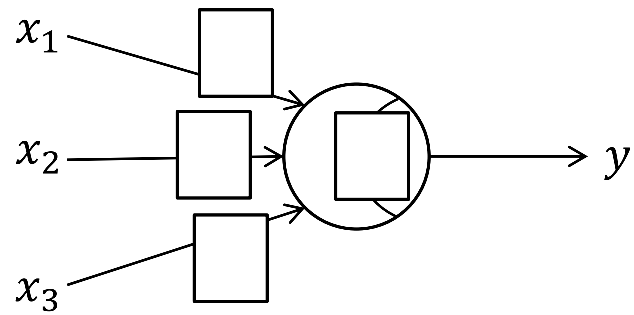
Exercise2.1

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



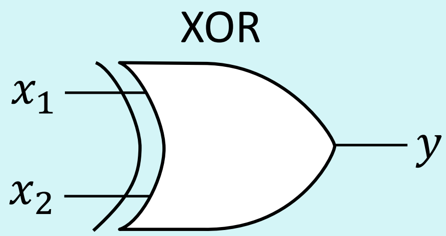
Truth table for NOR

x_1	x_2	x_3	y
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**0**
1	1	0	**0**
1	1	1	**0**



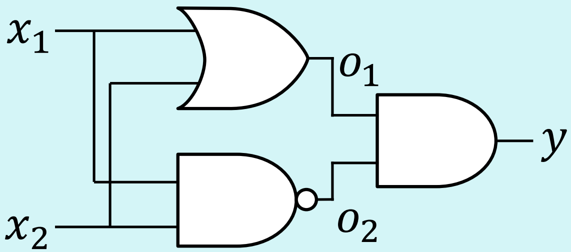
Representation of XOR function

XOR (Exclusive OR) gate is constructed as two layer logic circuit.



Truth table for XOR

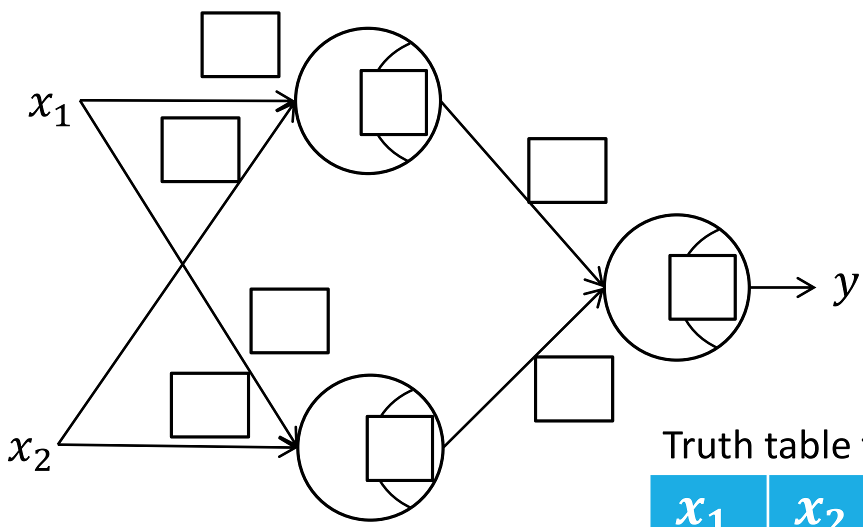
x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**



x_1	x_2	$o_1 (x_1 \text{ OR } x_2)$	$o_2 (x_1 \text{ NAND } x_2)$	$y\ (o_1 \text{AND} o_2)$
0	0	0	1	**0**
0	1	1	1	**1**
1	0	1	1	**1**
1	1	1	0	**0**

Exercise2.2

Construct a XOR gate with three formal neurons by using appropriate weighs and thresholds. Then, please check the answer by implementing a test script with all input pattern (i.e., {(0,0), (0,1), (1,0), (1,1)}) on MATLAB. Display results of mid-term calculation as necessary.



Truth table for XOR

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

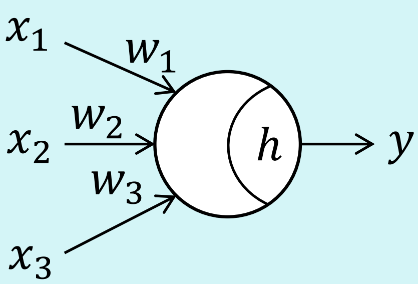
exercise2_2.m

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

w = [*,  *;
     *,  *];
h = [*;
     *];

u = [*,  *];
g = [*];

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

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

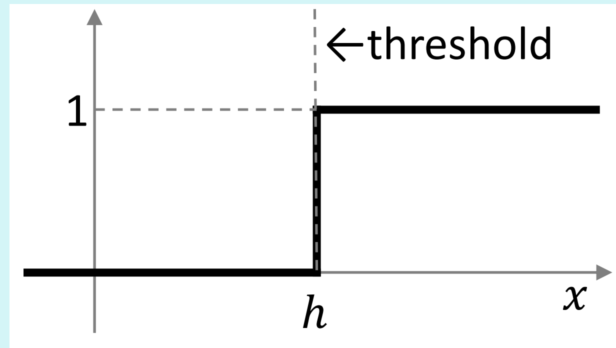
Please implement these values.

Using "a bias term" instead of a threshold

Formal neuron with threshold h

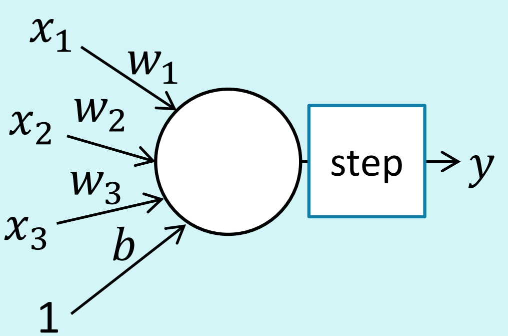


$$y = \begin{cases} 0 & \text{if } \sum_j w_j x_j \le h \\ 1 & \text{if } \sum_j w_j x_j > h \end{cases}$$



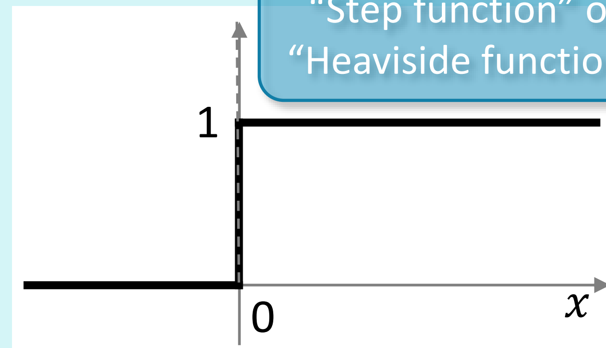
Let $b = -h$

Formal neuron with bias term b



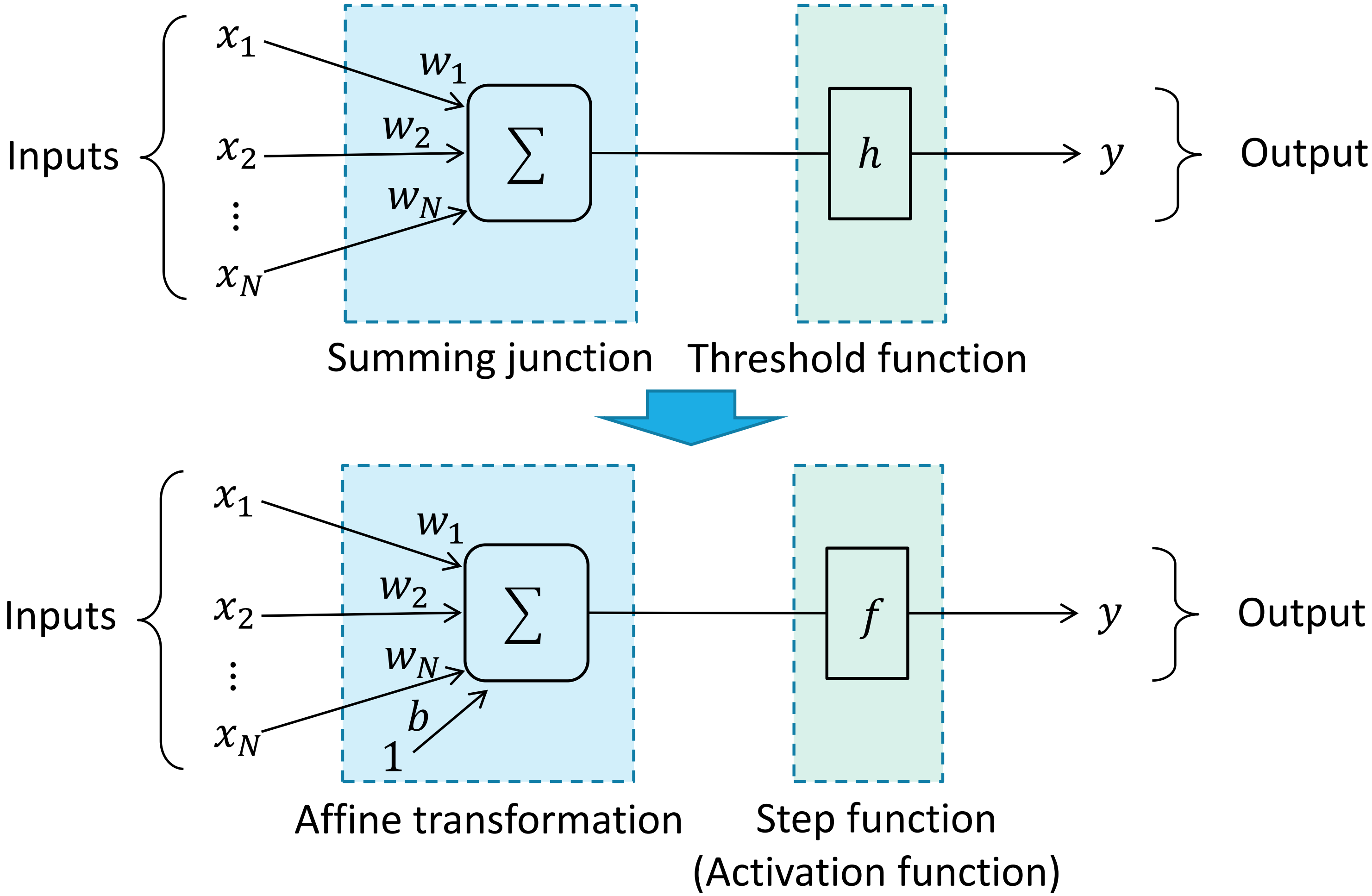
$$y = \begin{cases} 0 & \text{if } \sum_j w_j x_j + b \le 0 \\ 1 & \text{if } \sum_j w_j x_j + b > 0 \end{cases}$$

This is called "Step function" or "Heaviside function"



Either expression is completely same. However, when we use a bias term, we can describe a formal neuron as more general expression, i.e., $y = f(\sum_j w_j x_j + b)$. Note that the bias term is a parameter to control the tendency of the neuron firing.

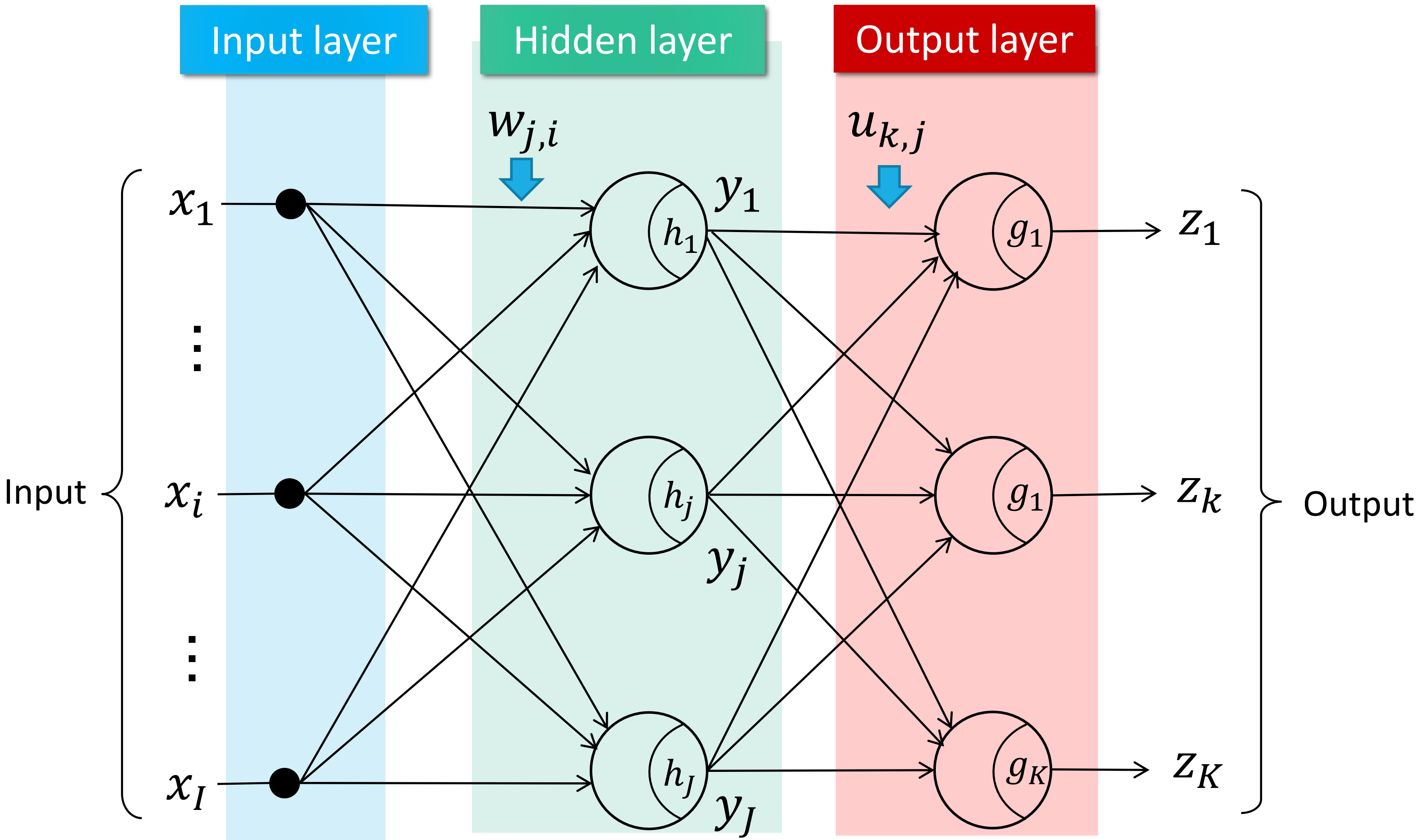
Review of Formal Neuron



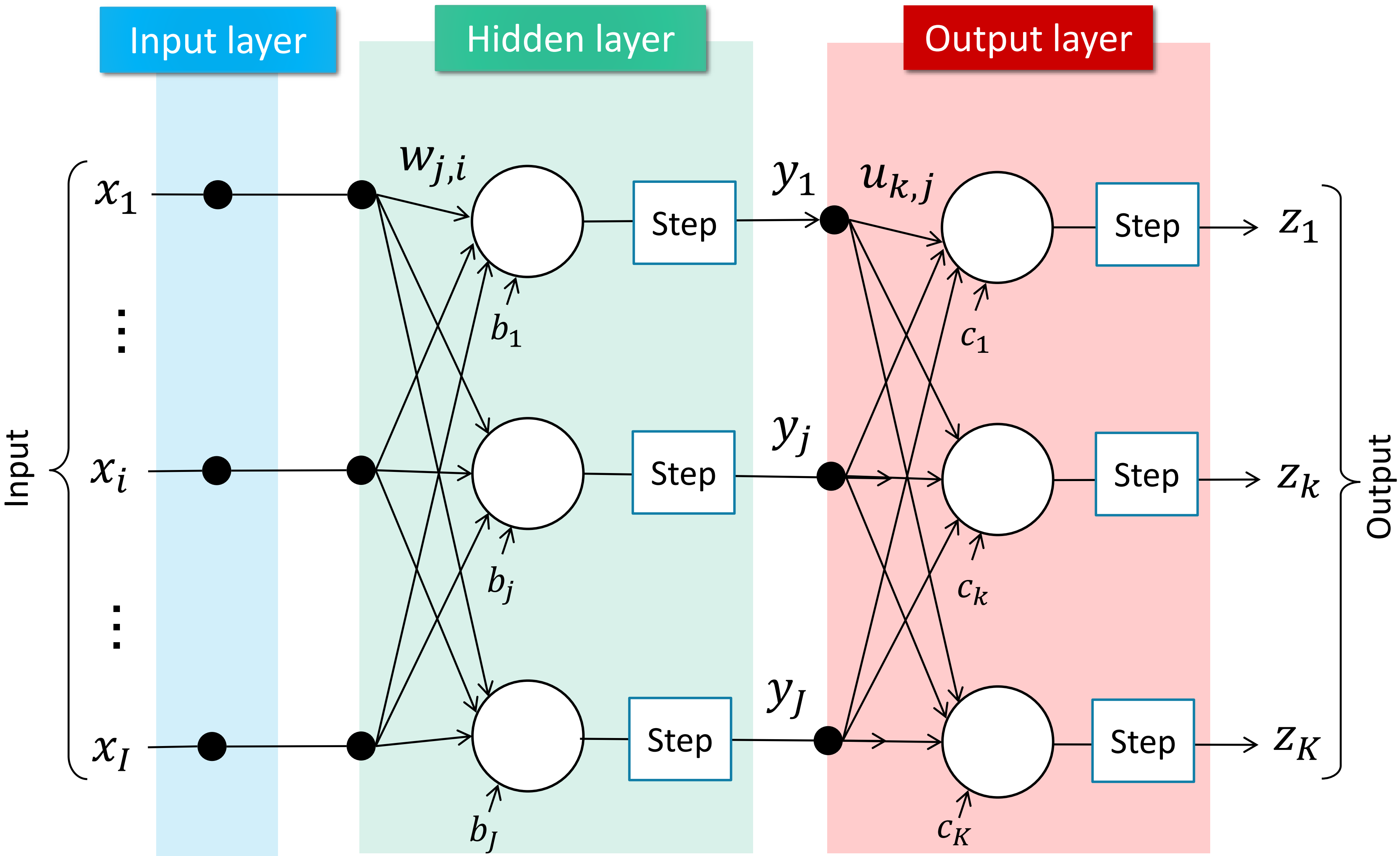
We adopted a step function as $f(x)$ now. However, we can use another function instead of a step function as $f(x)$. Generally, $f(x)$ is called "Activation function" and several typical function is proposed to $f(x)$. One of the most popular and useful function is "Sigmoid function".

【review】Multiple Layer Neural Network (Perceptron)

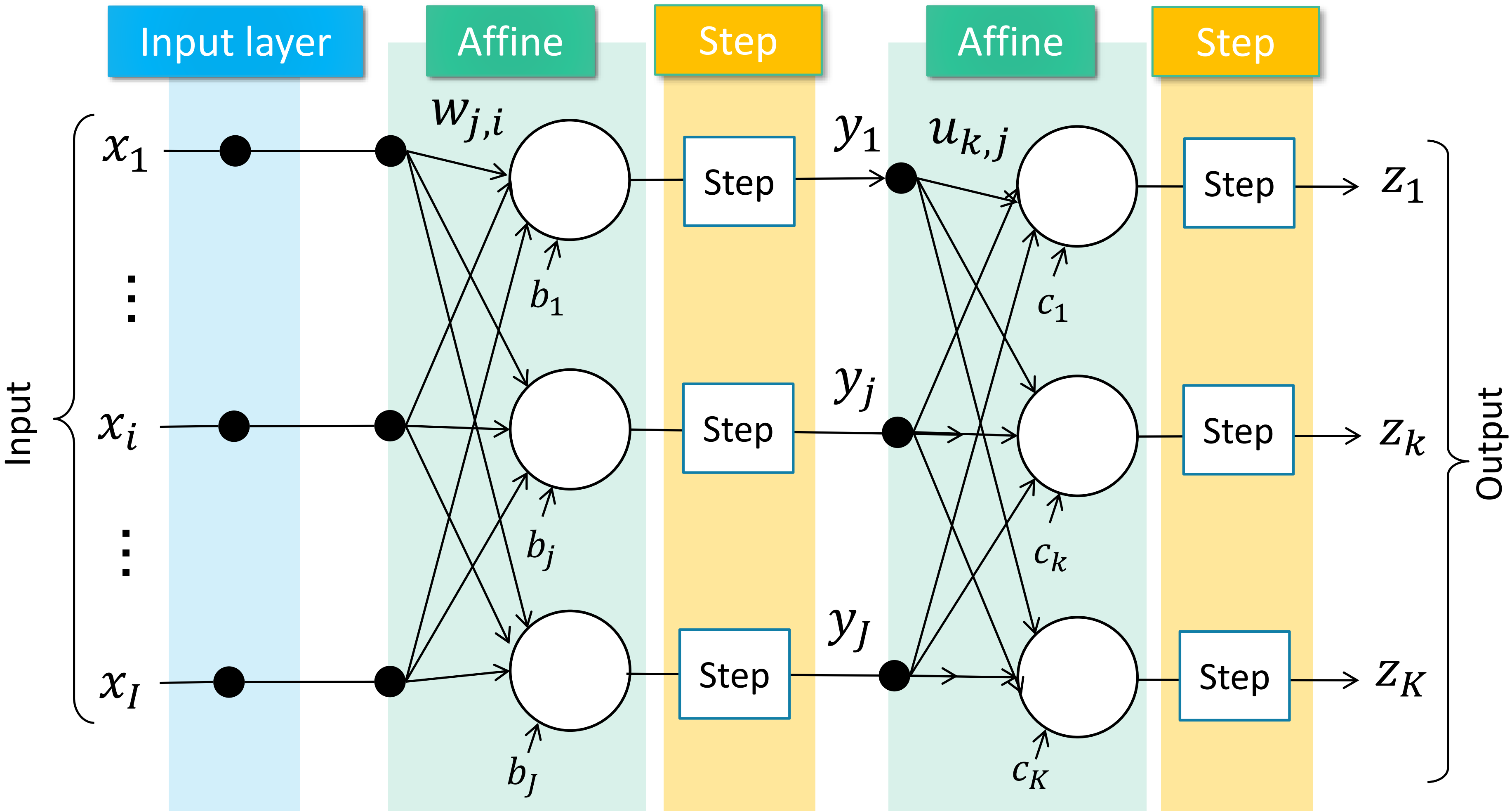
Generally,



Multiple Layer NN using Bias Term and Activation Function

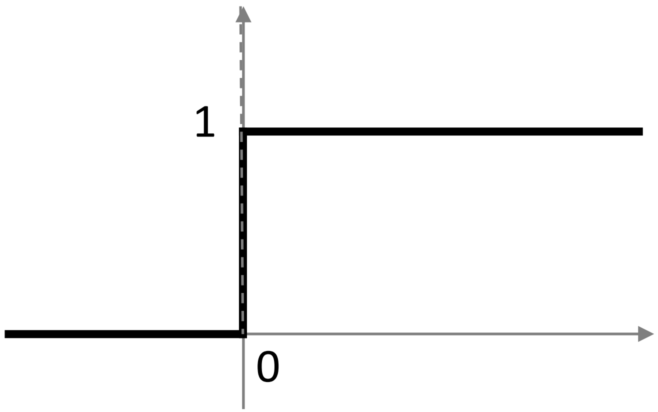


Multiple Layer NN using Bias Term and Activation Function



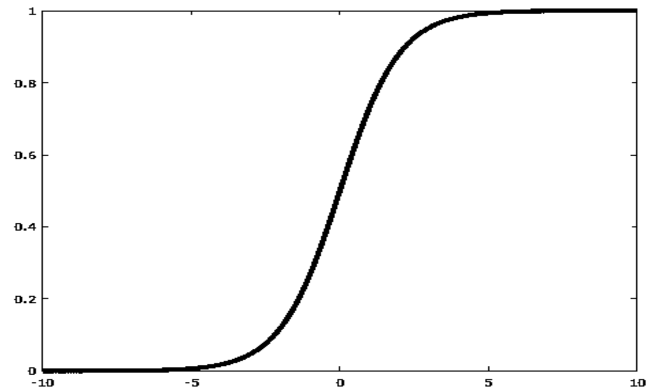
“Affine transformation” or “affine map “ is usually used in geometry.
Affine layer is also called “fully-connected layer”.

Variety of Activation Function



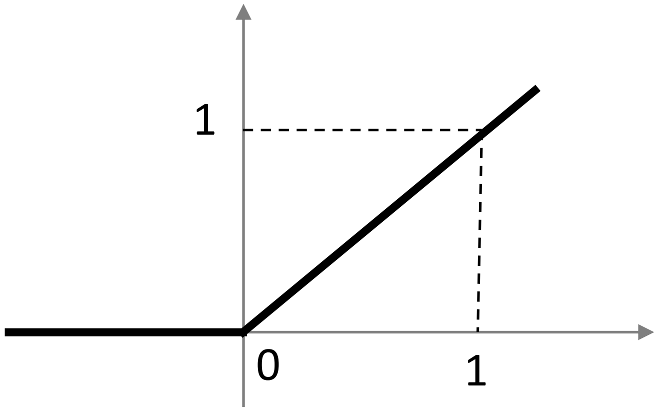
Step function

$$f(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ 1 & \text{if } a > 0 \end{cases}$$



Sigmoid function

$$f(a) = \frac{1}{1 + e^{-a}}$$



ReLU (Rectified Linear Unit)

$$f(a) = \begin{cases} 0 & \text{if } a \leq 0 \\ a & \text{if } a > 0 \end{cases}$$

Rewriting scripts with bias term and activation function

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 = obj.weights * x;
      y = p > obj.threshold;
    end
  end
end
```

Affine.m

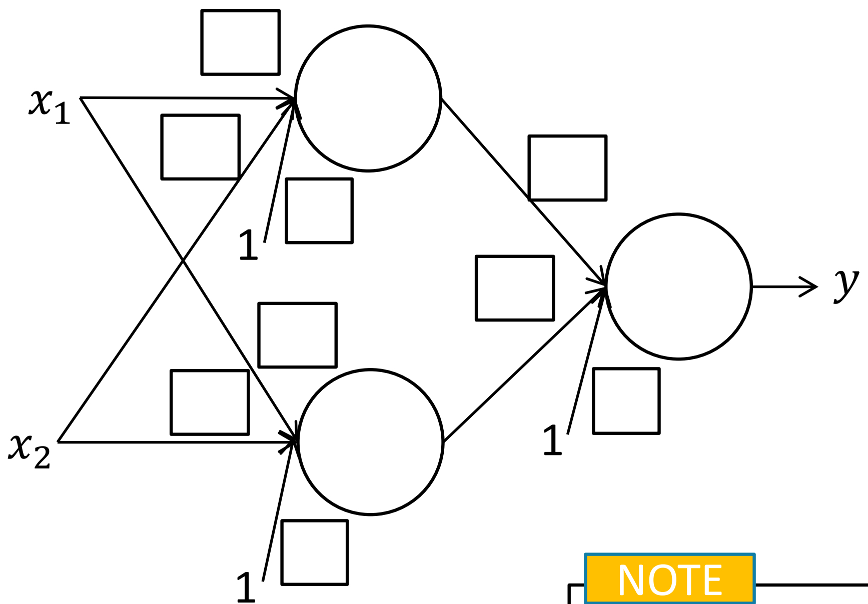
```
classdef Affine < handle
  properties
    weights;
    bias;
  end

  methods
    function obj = Affine(w,b)
      obj.weights = w;
      obj.bias = b;
    end

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

Step.m

```
classdef Step < handle
  methods
    function y = forward(obj, x)
      y = x > 0;
    end
  end
end
```

Exercise2.3

Based on the XOR function created in exercise 2.2, rewrite the neural network with bias term and step function (i.e., using a class “Affine.m” and “Step.m”) and make sure that the output does not change.



NOTE

b = -h

c = -g

Test script

exercise2_3.m

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

w = [*,  *;
     *,  *];
b = [*;
     *];

u = [*,  *];
c = [*];

layer1 = Affine(w, b);
layer2 = Step();
layer3 = Affine(u, c);
layer4 = Step();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
```

Sigmoid Function

Basic operation in MATLAB (14)

Array operation on MATLAB

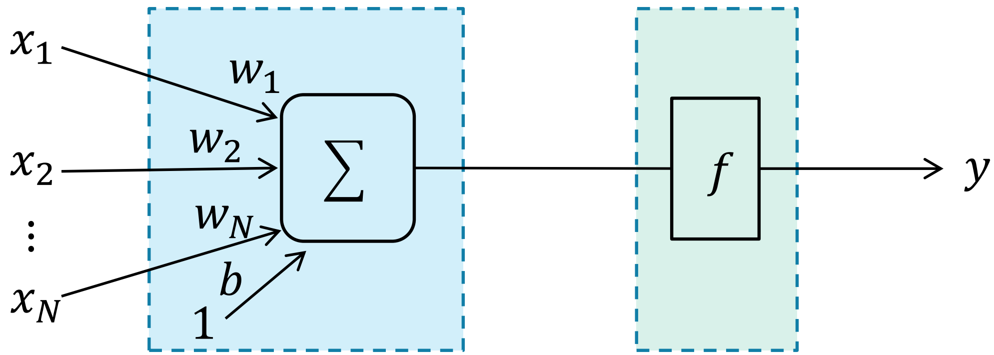
```
a =
    1    2    3
    4    5    6
    7    8    9

b =
    1    1    1
    2    2    2
    3    3    3

>> a.*b
ans =
    1    2    3
    8   10   12
   21   24   27
```

MATLAB arithmetic operators

- `*` Matrix multiplication
- `.*` Array multiplication
- `^` Matrix power
- `.^` Array power
- `/` Matrix right division
- `./` Array right division

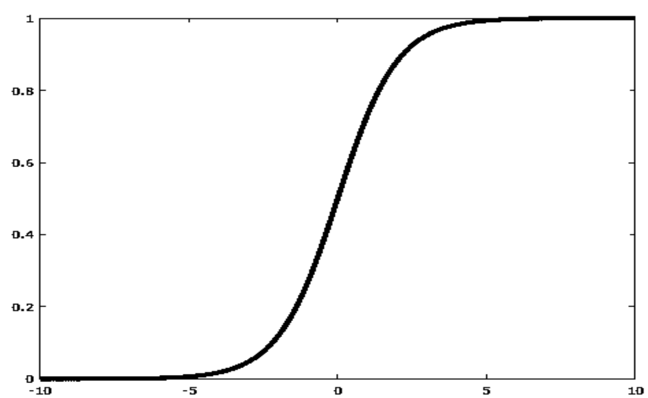
The period character (.) distinguishes array operations from matrix operations.

What's Sigmoid Neuron



Affine transformation

Activation function

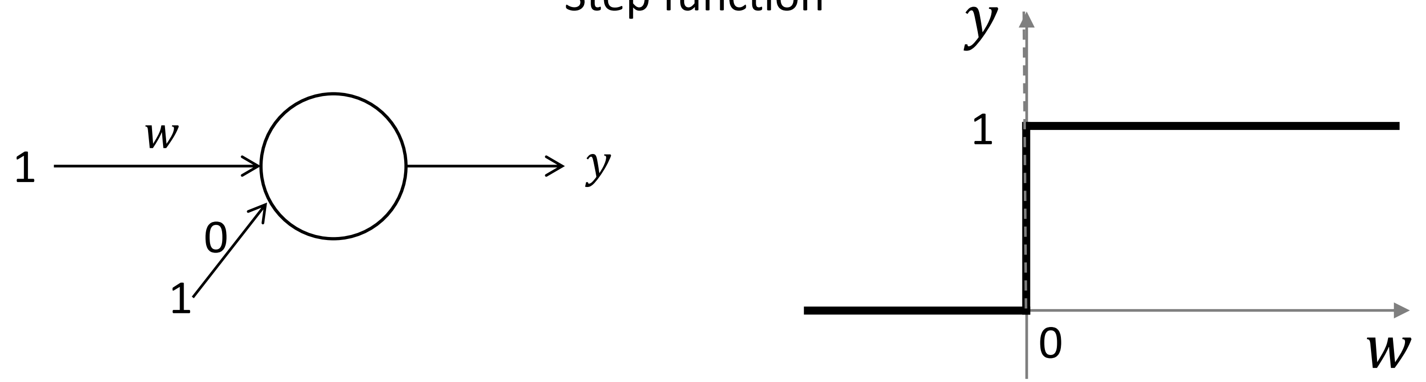


Sigmoid function

$$f(a) = \frac{1}{1 + e^{-a}}$$

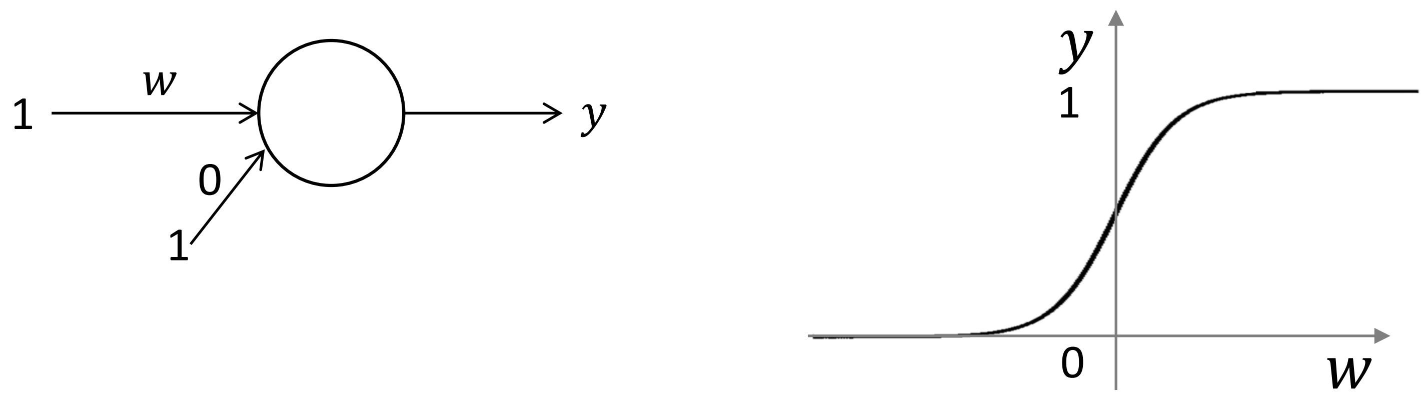
Difference Between Step Function and Sigmoid Function

Step function



The output value is 0 or 1 (two-value output)

Sigmoid function



The output value is a real number between 0 and 1.

Exercise2.4

Implement Sigmoid.m as follows.
Then, execute XOR function with sigmoid function and display the output values.

Sigmoid.m

```
classdef Sigmoid < handle
  methods
    function y = forward(obj, x)
      y = 1 ./ (1 + exp(-x));
    end
  end
end
```

exercise2_4.m

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

w = [*,*;
     *,*];
b = [*;
     *];

u = [*, *];
c = [*];

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = Affine(u,c);
layer4 = Sigmoid();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
```

Please use the values obtained in exercise 2.3.

Please change here from Step() to Sigmoid().

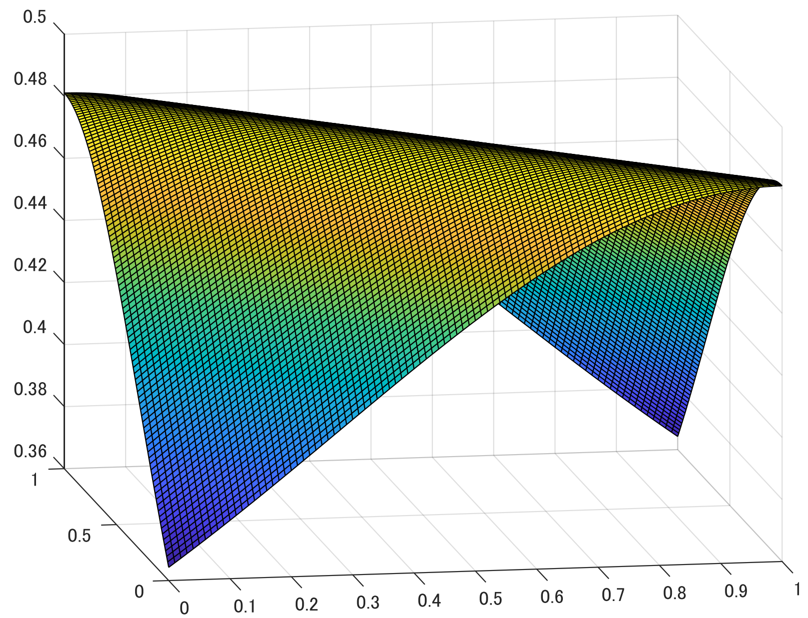
Displaying output graph with sigmoid neuron

example2_1.m

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

w = [2.1,  2;
     -2,  -2];
b = [-1;
     3];

u = [2,  2];
c = [-3];

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = Affine(u,c);
layer4 = Sigmoid();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)

% Display a mesh graph of output
figure(1);
[X,Y] = meshgrid(0.00:0.01:1);
xg = [X(:), Y(:)]';
pg = layer1.forward(xg);
yg = layer2.forward(pg);
qg = layer3.forward(yg);
zg = layer4.forward(qg);
surf(X,Y,reshape(double(zg),[101,101]));
```

Values are for example

Output graph with sigmoid neuron



Please insert these codes to exercise2_4.m
We can see an output graph.

Displaying output graph with formal neuron

example2_2.m

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

w = [2.1,  2;
     -2,  -2];
b = [-1;
     3];

u = [2,  2];
c = [-3];

layer1 = Affine(w,b);
layer2 = Step();
layer3 = Affine(u,c);
layer4 = Step();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)

% Display a mesh graph of output
figure(1);
[X,Y] = meshgrid(0.00:0.01:1);
xg = [X(:), Y(:)]';
pg = layer1.forward(xg);
yg = layer2.forward(pg);
qg = layer3.forward(yg);
zg = layer4.forward(qg);
surf(X,Y,reshape(double(zg),[101,101]));
```

If we use a step function,
what kind of graph is outputted?

?

Exercise 2.5

If we slightly change the value of weights or biases in exercise2_4.m, please check how the output value changes.

exercise2_5.m

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

w = [*, *;
     *, *];
b = [*;
     *];

u = [*,  *];
c = [*];

layer1 = Affine(w, b);
layer2 = Sigmoid();
layer3 = Affine(u, c);
layer4 = Sigmoid();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
```

Change these values slightly.

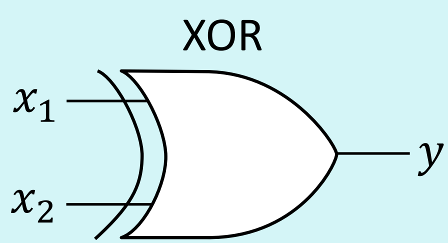
Learning Neural Network

Why we use sigmoid function?

In the exercise 2.2 and 2.3, you could find appropriate value for the parameters (i.e, weights and thresholds).

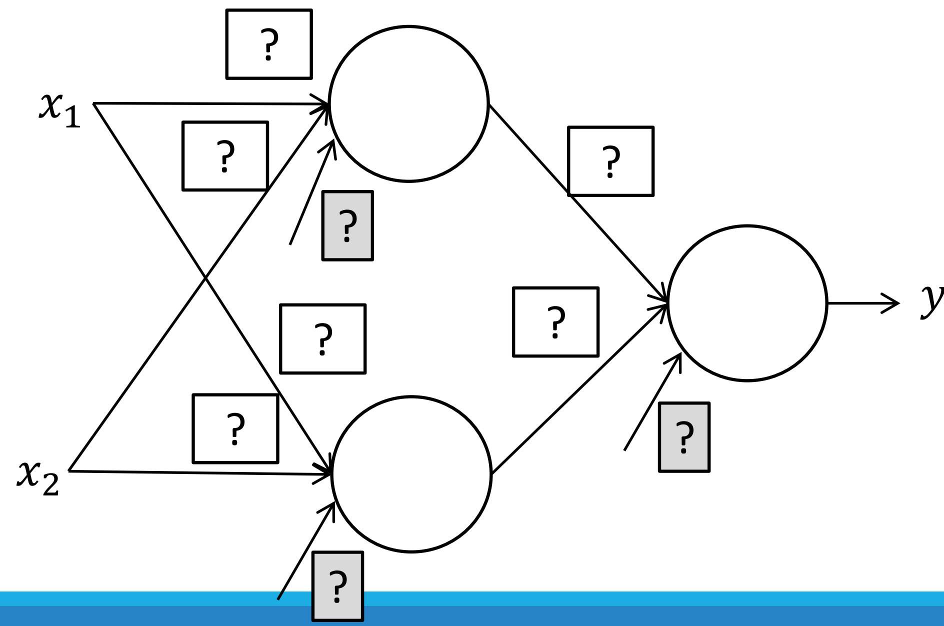
When we want to constract larger neural network in practical, it is hard or impossible to find appropriate value with human power.

We want to find optimal parameters AUTOMATICALY.



Truth table for XOR

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



【review】Difference from traditional programming

Traditional Programming

Data (Inputs) → Computer → Output (Results)
Program (Rules) → Computer

Human input rules between input and output explicitly.
However, rules of real problems are complex in most cases.

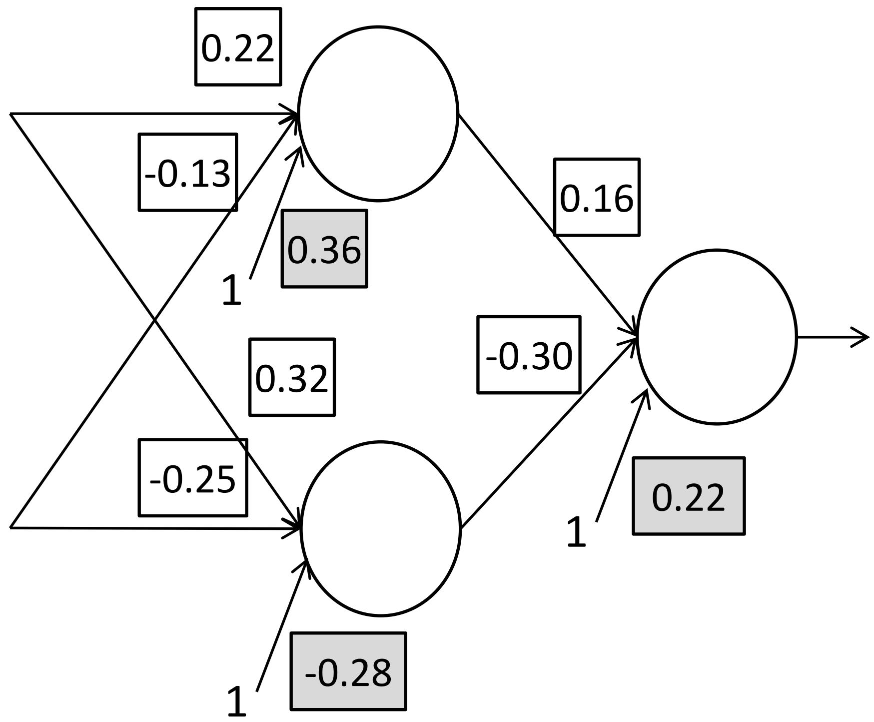
Machine Learning

Data (Inputs) → Computer with Machine Learning Algorithm → Program (Rules)
Output (Results) → Computer with Machine Learning Algorithm

Machine learning provides “automating automation”

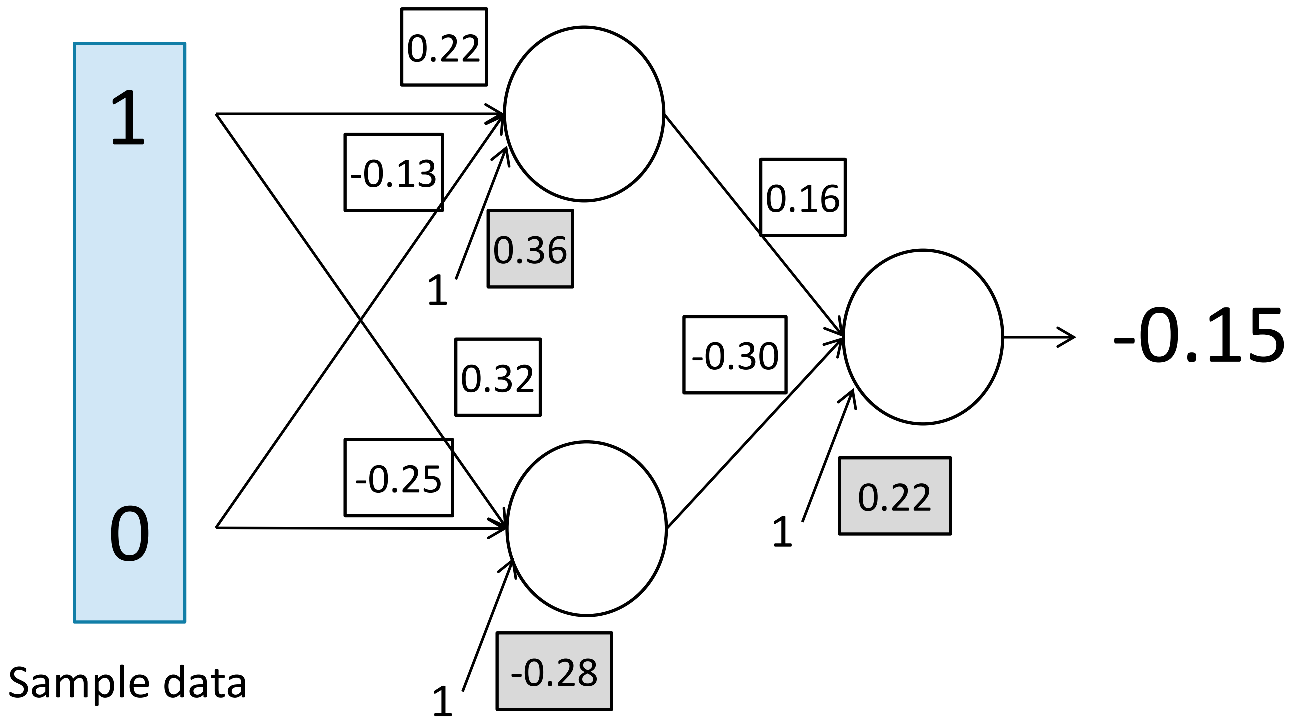
Basic Idea of Learning Neural Network (1)

At first, we set random value to each parameter.



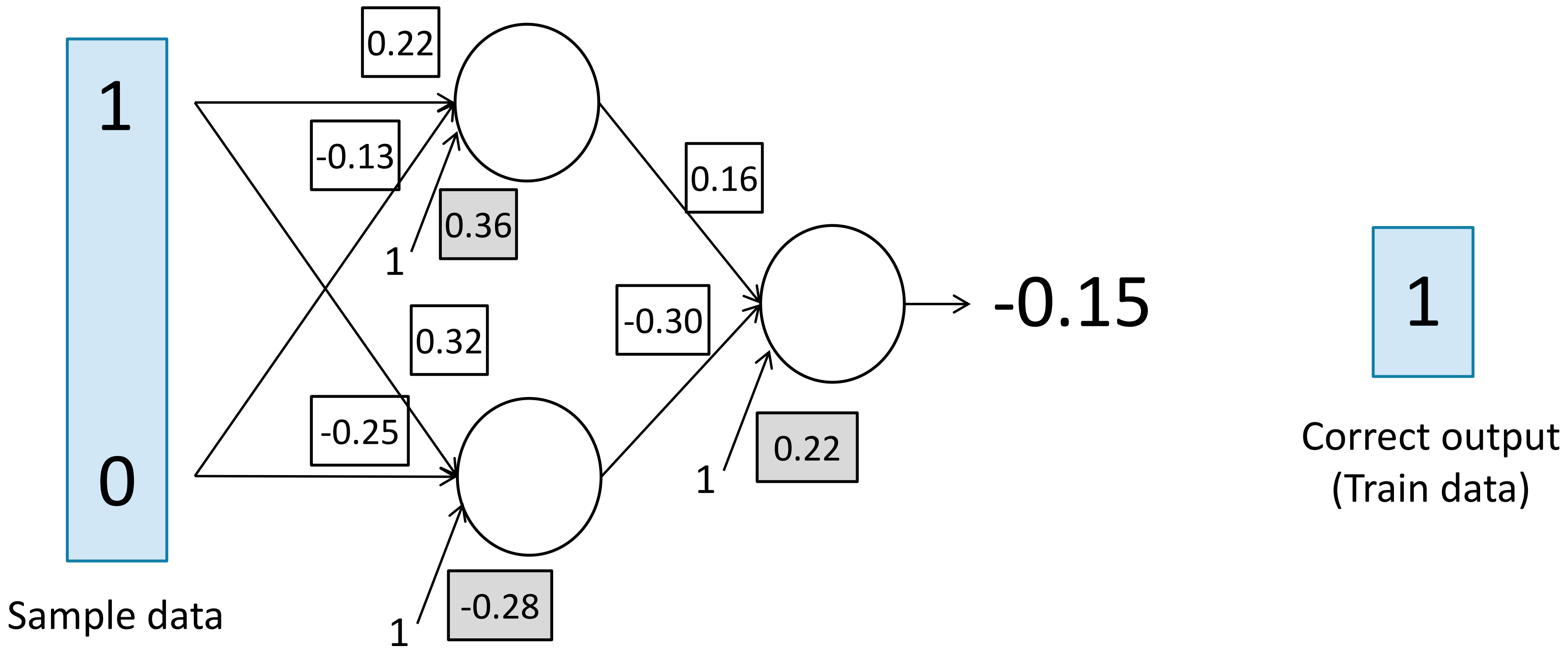
Basic Idea of Learning Neural Network (2)

We input sample data and calculate the output on the neural network with random parameter. Of course, the output value also become random.



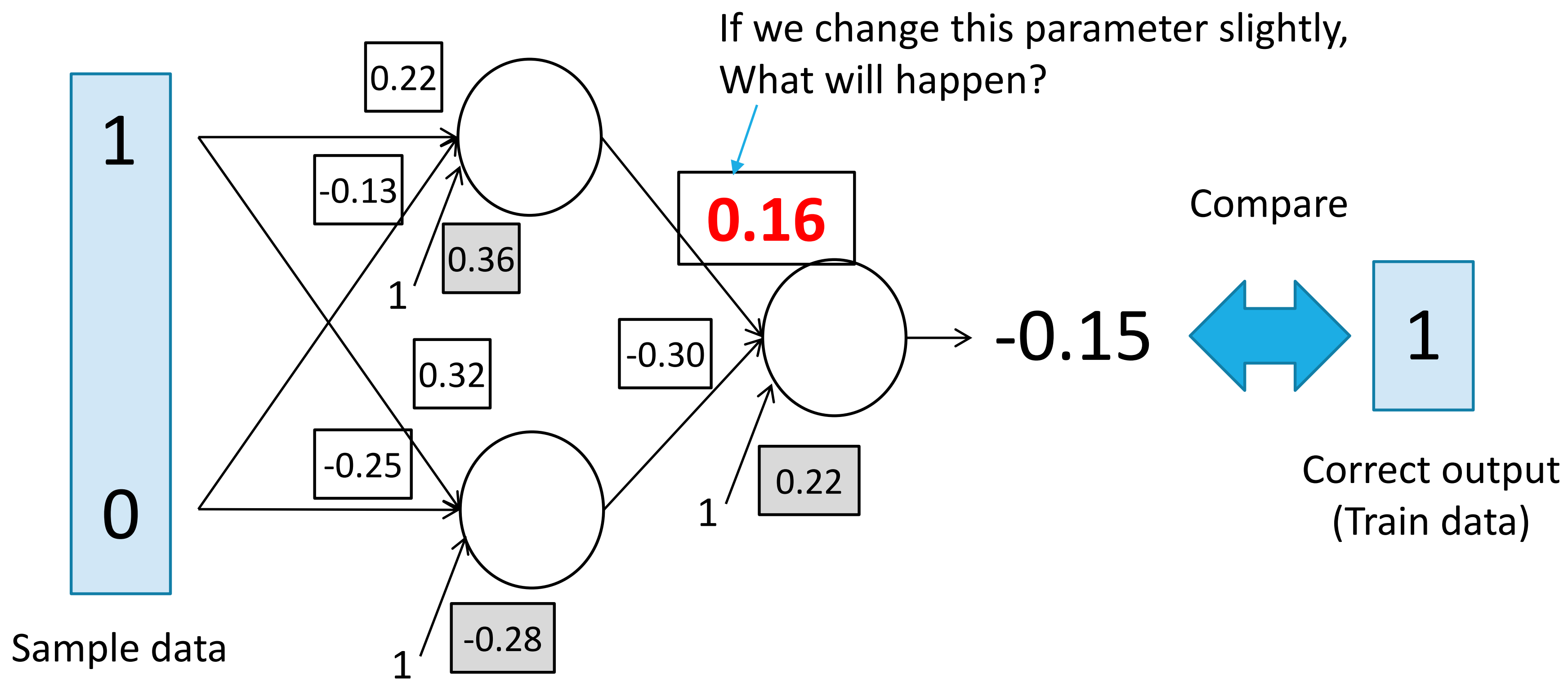
Basic Idea of Learning Neural Network (2)

We input sample data and calculate the output on the neural network with random parameter. Of course, the output value also become random.

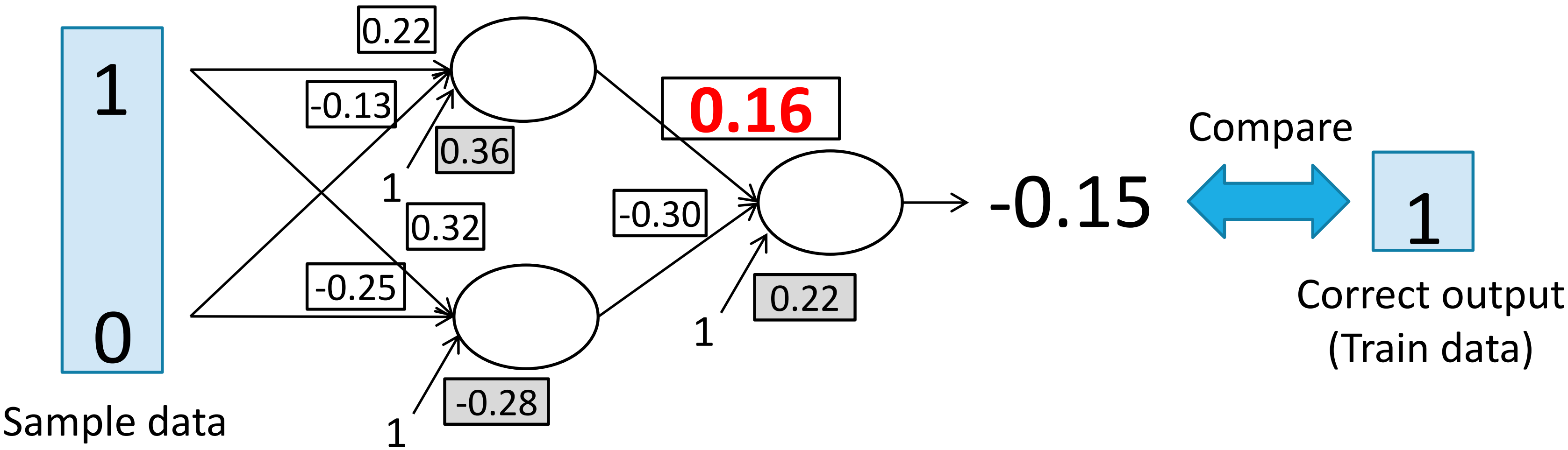


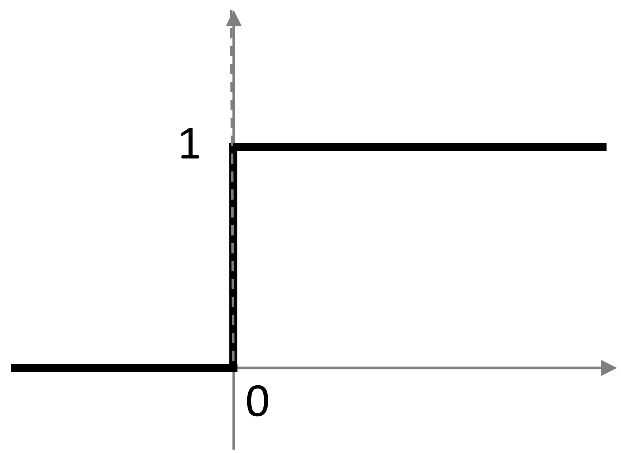
Basic Idea of Learning Neural Network (2)

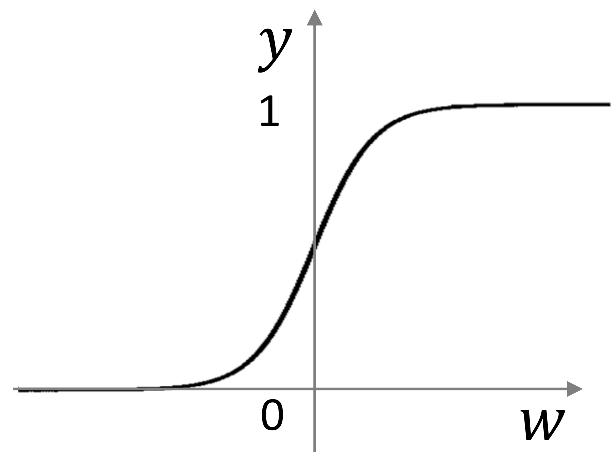
If you change one of the parameters slightly, please think about what the output will change.



Difference between Formal Neuron and Sigmoid Neuron







If we use sigmoid function, we can adjust the parameters gradually.

Exercise2.6

Based on the example1_5.m, create excrcise2_6.m with sigmoid function instead of formal neuron as follows. Then compare the difference between the results of example1_5.m and excrcise2_6.m

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)
```

Change

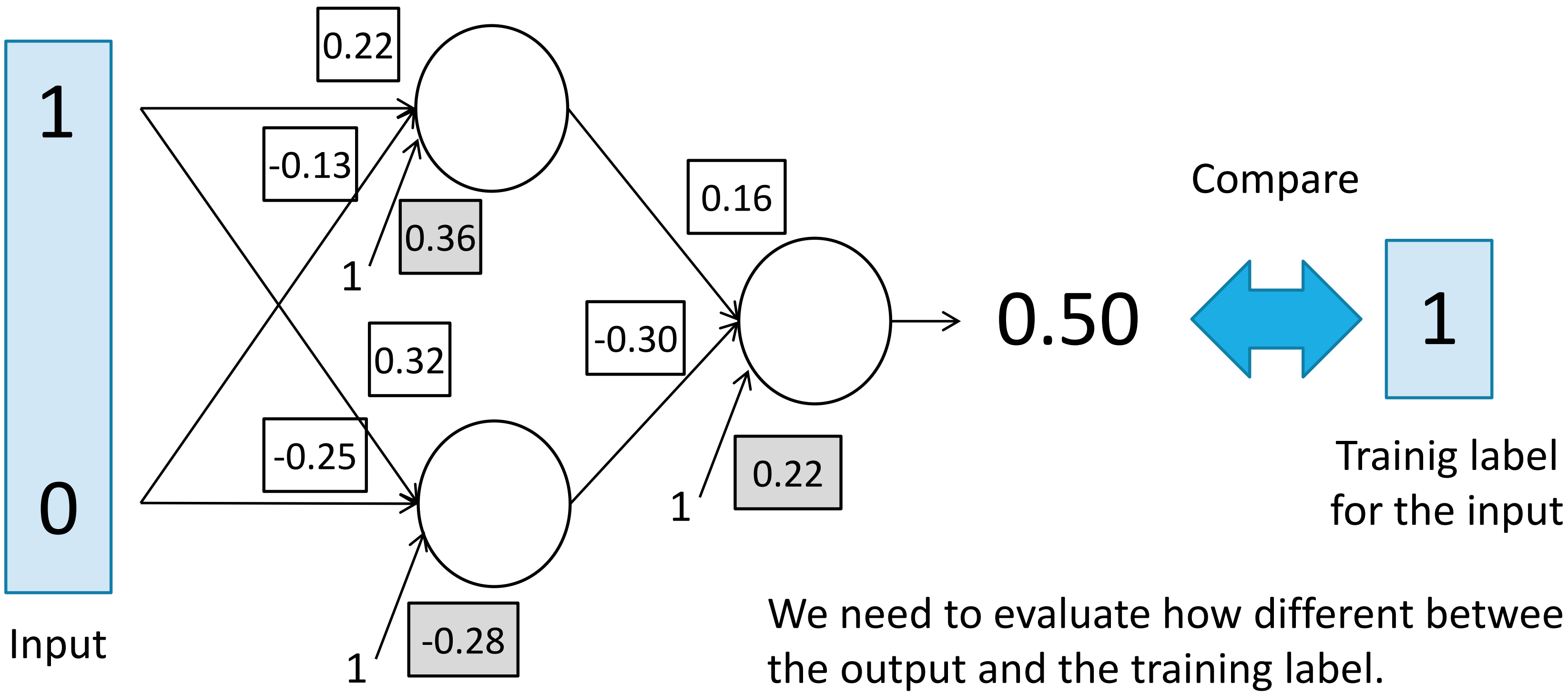
exercise2_6.m

```
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];
b = [-0.5;
     -1.0;
     -0.0];

u = [1.0,  0.5,  0.0;
     0.5,  0.0,  1.0];
c = [-1.0;
     -0.0];

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = Affine(u,c);
layer4 = Sigmoid();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
```

Loss Function

【review】Basic Idea of Learning Neural Network

We input sample data and calculate the output on the neural network with random parameter. Of course, the output value also become random.



We need to evaluate how different between the output and the training label.

Loss Function (or Cost Function)

A loss function (or a cost function) is a function that calculates how different between an output and a training label, i.e. correct value.

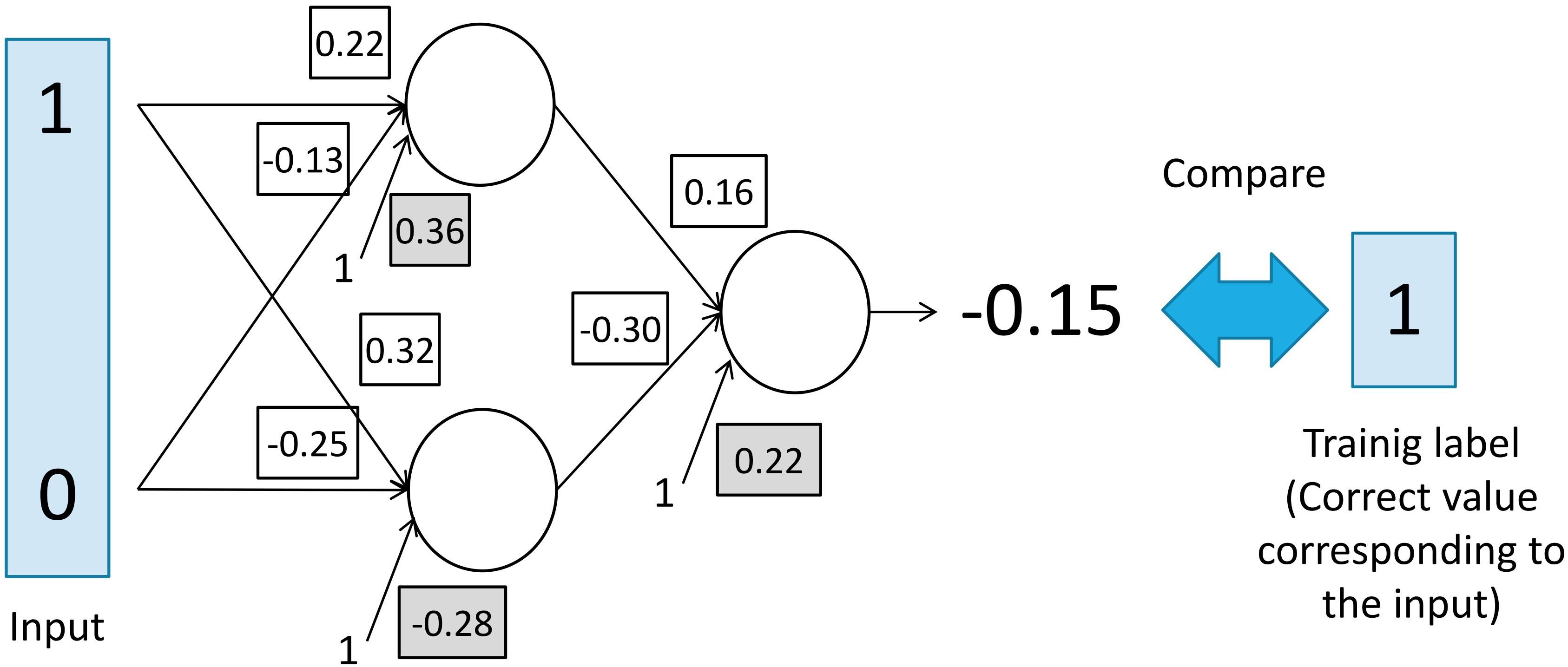
One of the most useful cost function is MSE (Mean Squared Error) as follows.

$$LOSS = L(\boldsymbol{z}) = \frac{1}{2}\sum_{k}(z_k - t_k)^2$$

where k is a dimension of output (i.e., a number of neurons in the output layer) .

MSE is always positive value.

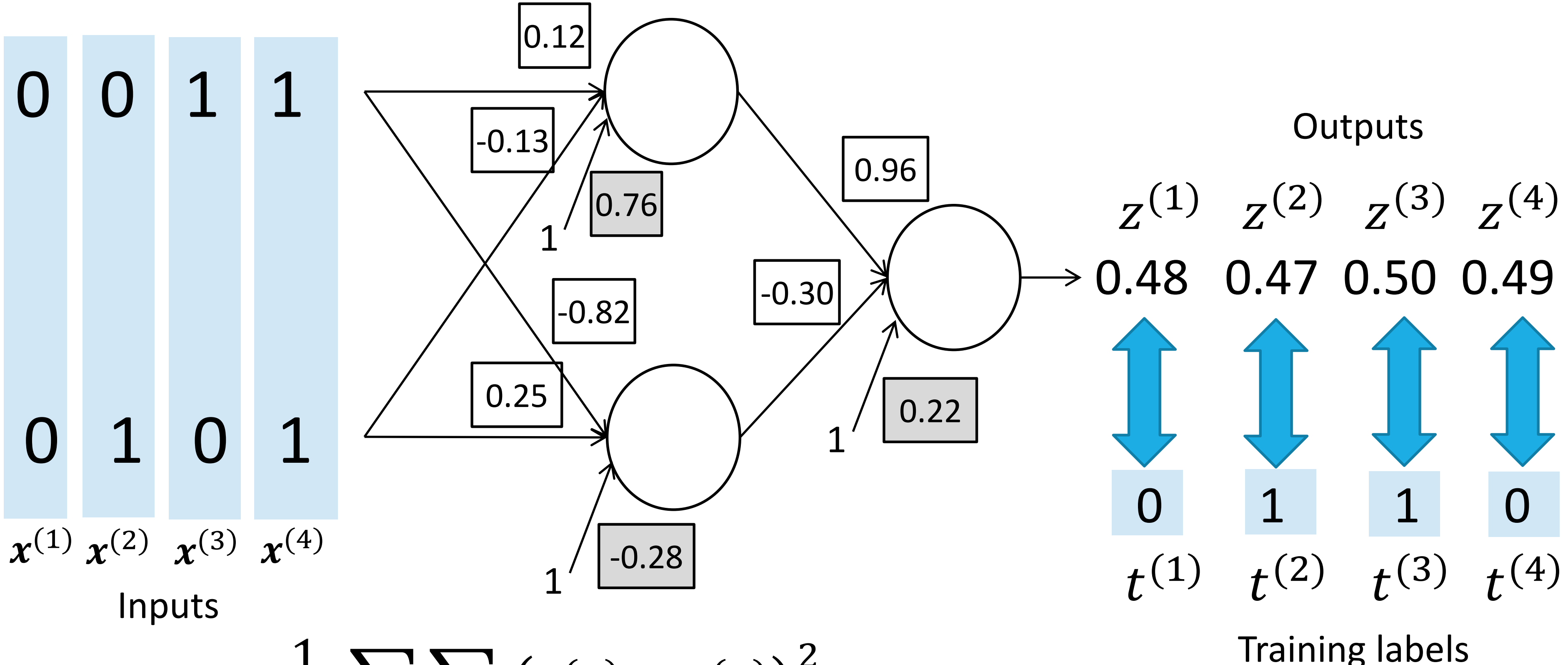
Calculation Example of MSE



For example, we can calculate the MSE as follows.

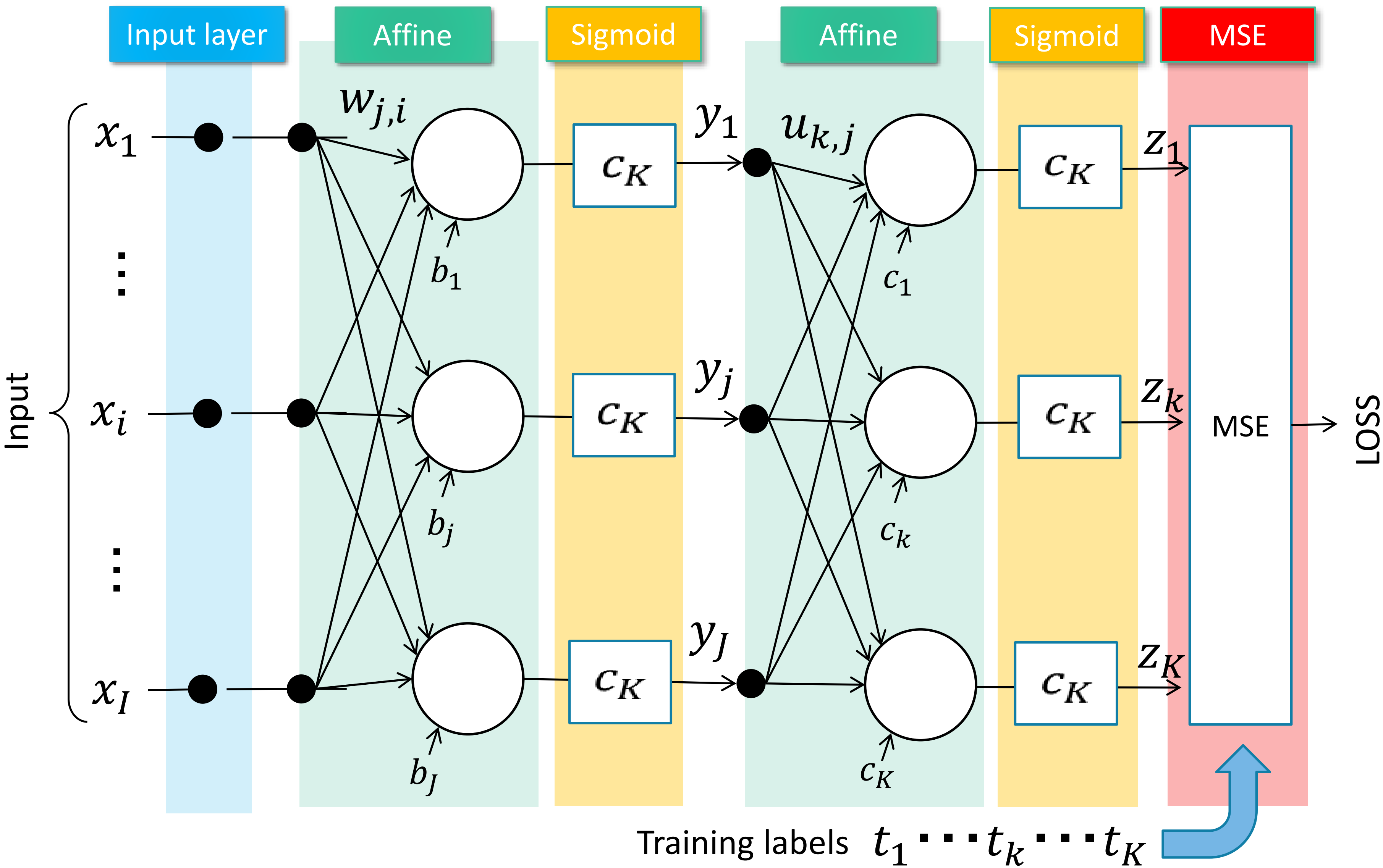
$$LOSS = L(\boldsymbol{z}) = \frac{1}{2}\sum_k (z_k - t_k)^2 = \frac{1}{2}(-0.15 - 1)^2 = 0.66125$$

Calculation Example of MSE



$$
\begin{aligned}
LOSS &= \frac{1}{2N}\sum_n \sum_k \left(z_k^{(n)} - t_k^{(n)}\right)^2 \\
&= \frac{1}{8}\{(0.48-0)^2+(0.47-1)^2+(0.50-1)^2+(0.49-0)^2\} \\
&= 0.125175
\end{aligned}
$$

Multiple Layer NN with LOSS function



Loss function

example2_3.m

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

w = [0.12, -0.13;...
     -0.82, 0.25];
b = [0.76;...
     -0.28];

u = [0.96, -0.30];
c = [0.22];

layer1 = Affine(w, b);
layer2 = Sigmoid();
layer3 = Affine(u, c);
layer4 = Sigmoid();
layer5 = MSE();

p = layer1.forward(x);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
loss = layer5.forward(z, t)
```

MSE.m

```
classdef MSE < handle
  methods
    function loss = forward(obj, z, t)
      [row, col] = size(z);
      loss = sum(sum((z-t).^2)) / (2*col);
    end
  end
end
```

NOTE

row ･･･ dimension of output

col･･･ a number of data

Exercise 2.7

Implement loss function (MSE.m), then calculate the loss value of XOR function implemented in exercise2_4.m.

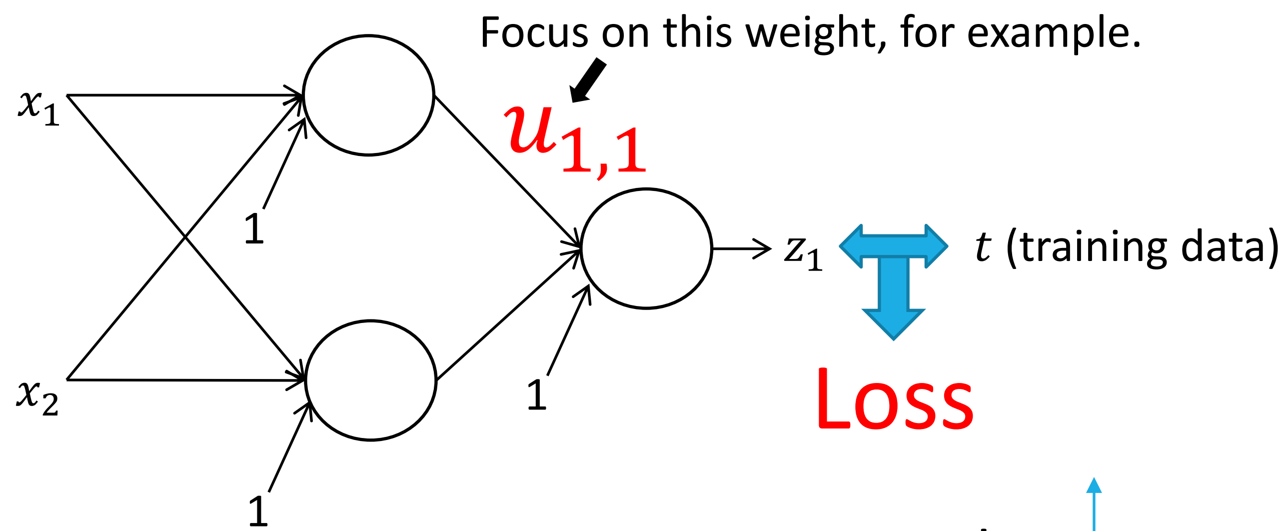
Exercise 2.8

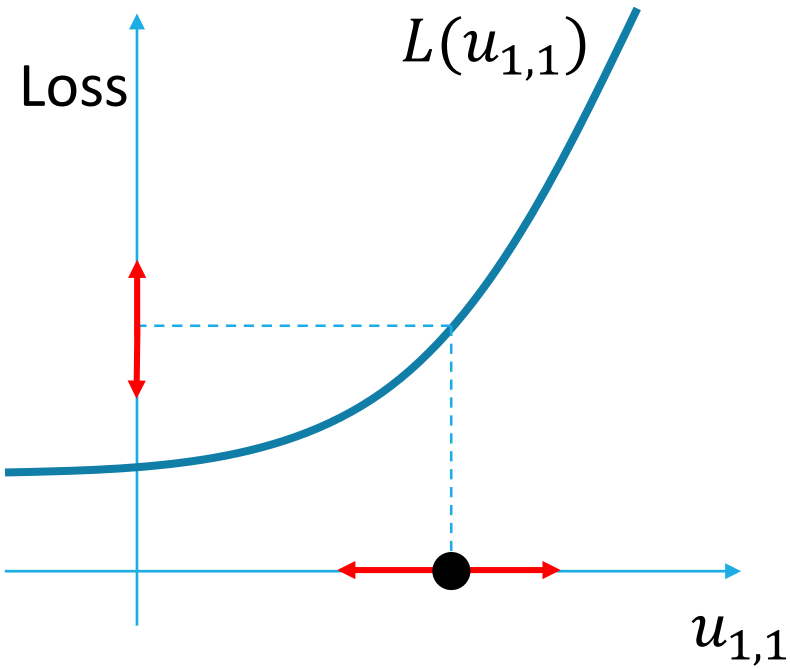
In the XOR function, set the weights and biases as a random number between -1 and 1 as following script. Then calculate and check the LOSS value.

exercise2_8.m

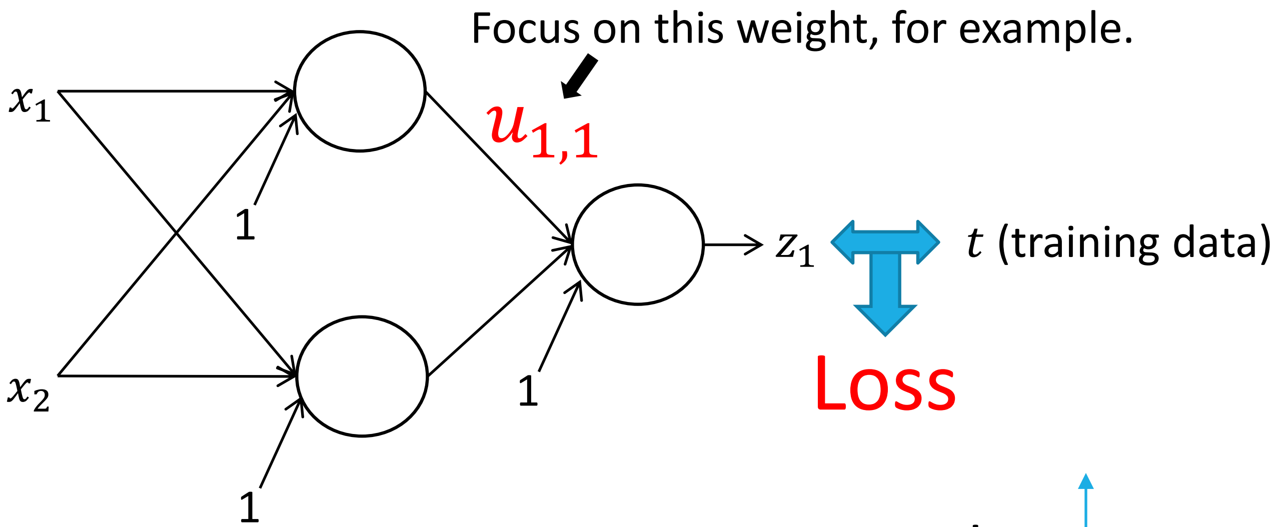
```
clear all

xdata = [0,0,1,1;
         0,1,0,1];
labels = [0,1,1,0];
data_num=4;

IU = 2;     % a number of input neurons
HU = 2;     % a number of hidden neurons
OU = 1;     % a number of output neurons

% initialize weights and biases
% as random numbers between -1.0 and 1.0.
w = 2.0*rand(HU,IU) - 1.0;
b = 2.0*rand(HU,1) - 1.0;
u = 2.0*rand(OU,HU) - 1.0;
c = 2.0*rand(OU,1) - 1.0;

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = Affine(u,c);
layer4 = Sigmoid();
layer5 = MSE();

p = layer1.forward(xdata);
y = layer2.forward(p)
q = layer3.forward(y);
z = layer4.forward(q)
loss = layer5.forward(z,labels)
```

How do we reduce the LOSS (1)



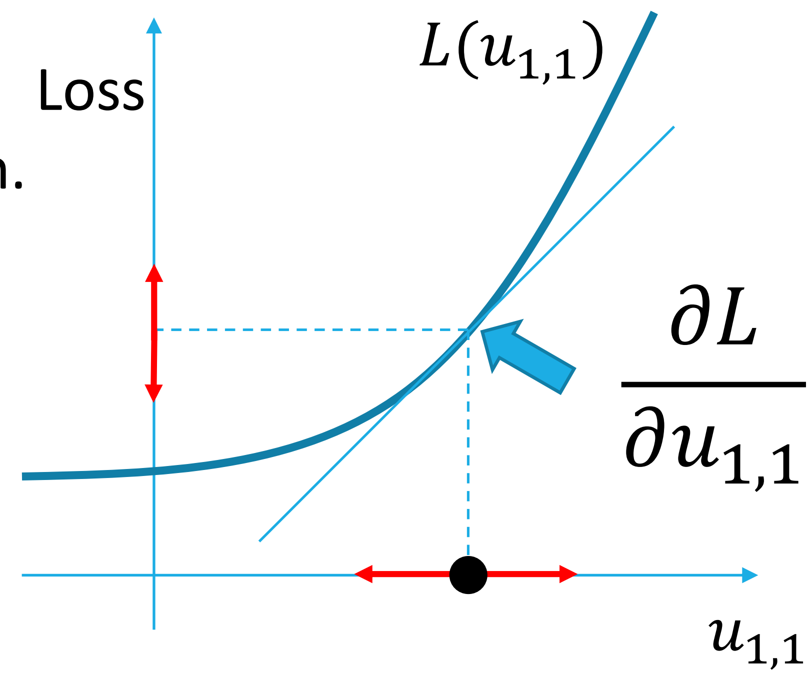


How can we reduce the LOSS (1)



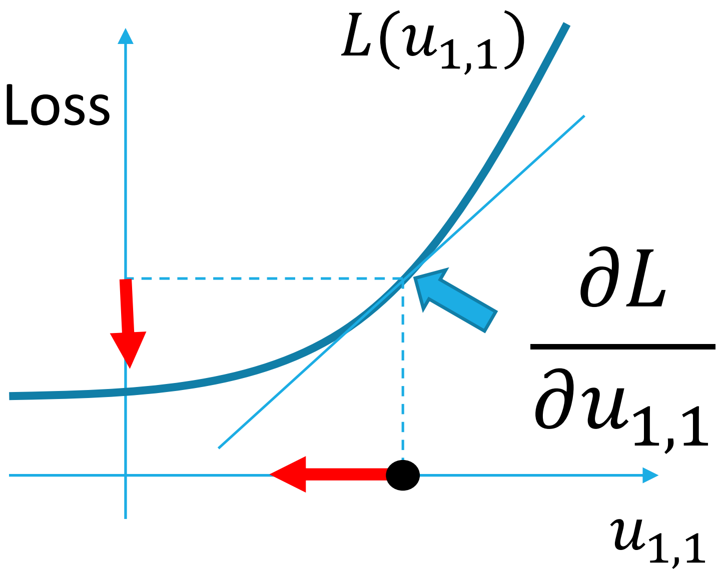
We can calculate slope of Loss function.

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

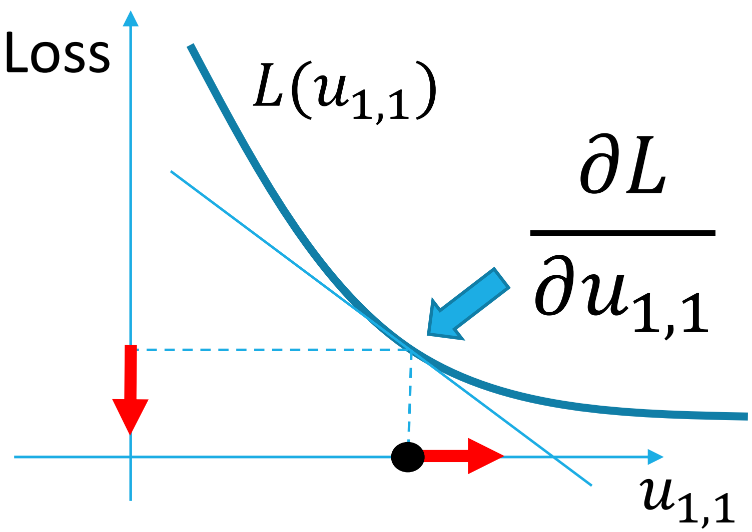


How can we reduce the LOSS (2)

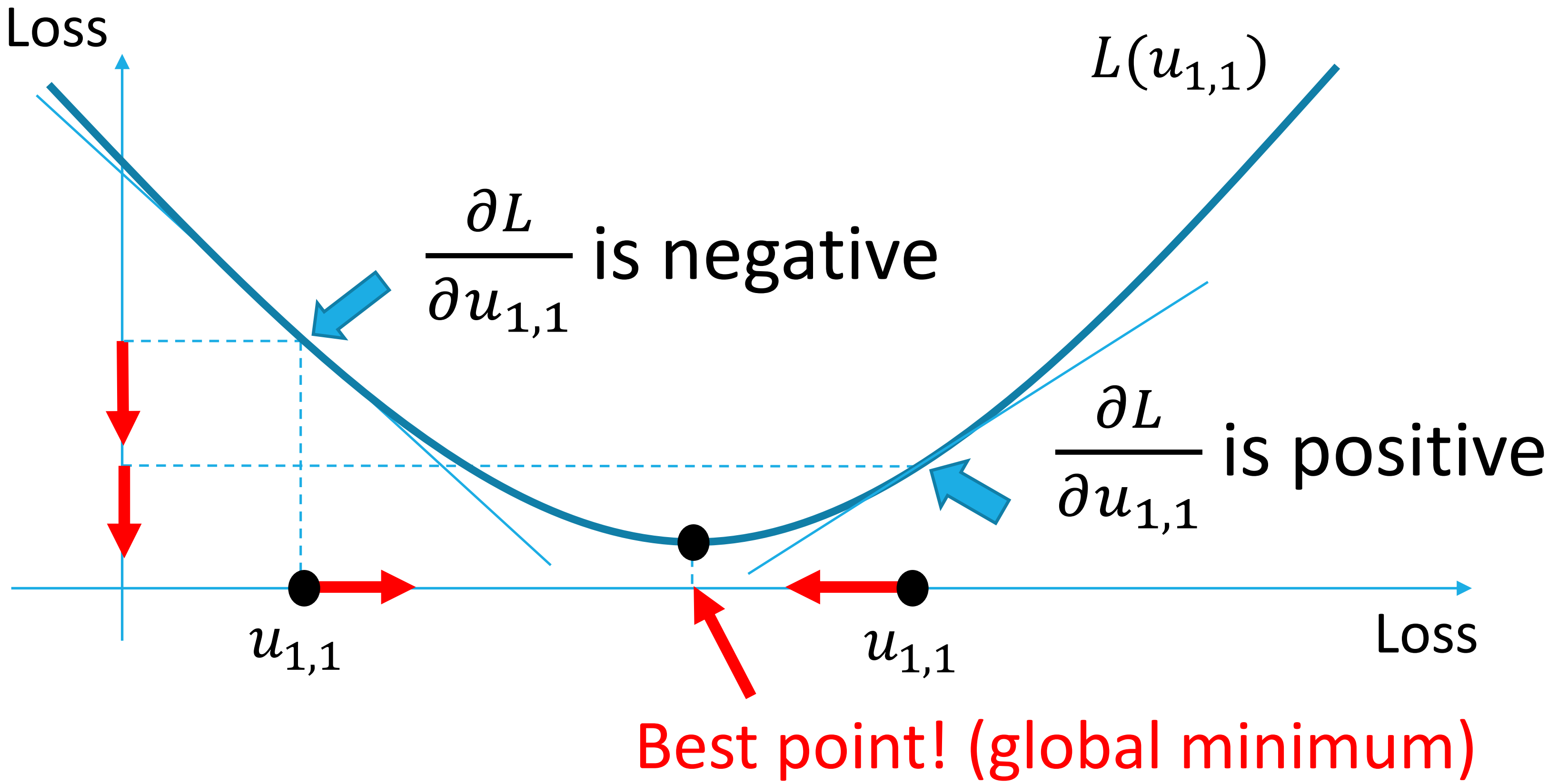
If $\frac{\partial L}{\partial u_{1,1}}$ is a positive value, we should reduce $u_{1,1}$ to reduce LOSS.



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



How can we reduce the LOSS (3)



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

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

How can we reduce the LOSS (3)

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

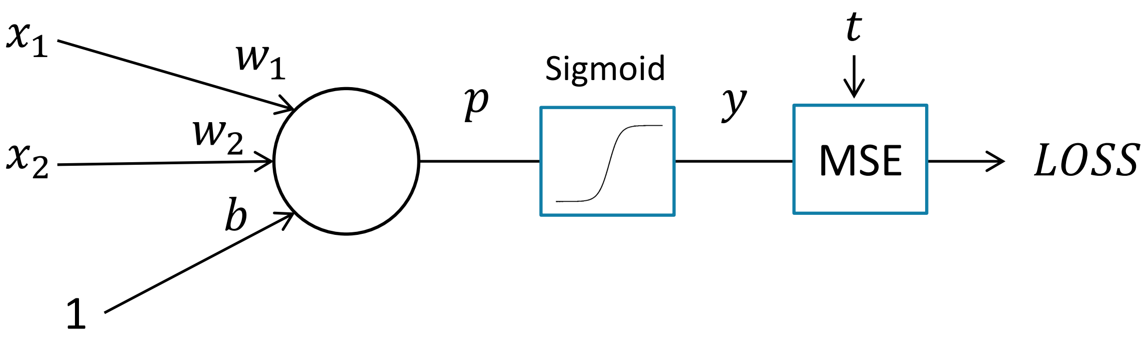
Update function for weights

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

We call λ “learning rate”

Neural Network Learning

Simple example



we want to get $\frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}$ and $\frac{\partial L}{\partial b}$

The Composite Function Rule (Chain Rule)

If y is a function of u and u is a function of x i.e, $y = f(u), u = g(x)$, then

$$\frac{dy}{dx} = \frac{dy}{du} \cdot \frac{du}{dx} \quad \cdots \text{Chain rule}$$

Similarly,

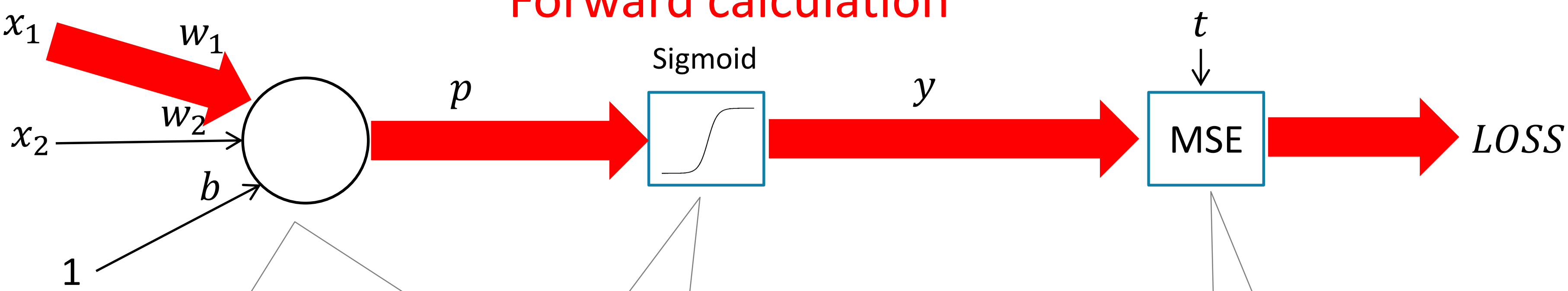
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Furthermore,

$$\boxed{\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial w}}$$

Simple example

Forward calculation



$$p = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + b$$

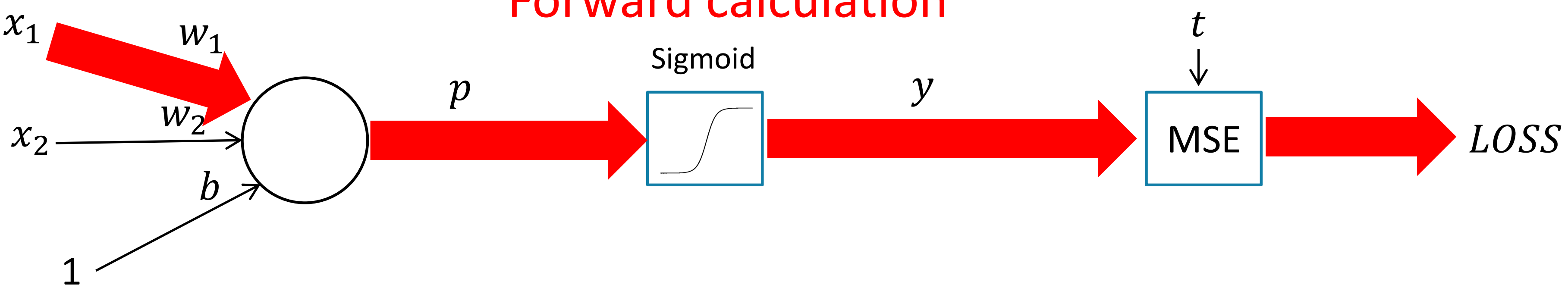
$$y = \sigma(p) = \frac{1}{1 - e^{-p}}$$

$$LOSS = L(\boldsymbol{y}) = \frac{1}{2N} \sum_n (y - t)^2$$

MSE is always positive value.

Simple example

Forward calculation



$$p = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + b$$

$$\frac{\partial p}{\partial w_i} = x_i \qquad \frac{\partial p}{\partial b} = 1$$

$$y = \sigma(p) = \frac{1}{1 - e^{-p}}$$

$$\frac{\partial y}{\partial p} = y\,(1 - y)$$

$$LOSS = L(\boldsymbol{y}) = \frac{1}{2N} \sum_n (y - t)^2$$

$$\frac{\partial L}{\partial y} = y - t$$

【Appendix】Differencial of Sigmoid Function

$$
\begin{aligned}
y = \sigma(p) &= \frac{1}{1+e^{-p}} \\
\sigma'(p) &= \frac{-1}{(1+e^{-p})^2}(1+e^{-p})' \\
&= \frac{e^{-p}}{(1+e^{-p})^2} \\
&= \frac{1}{1+e^{-p}}\frac{e^{-p}}{1+e^{-p}} \\
&= \frac{1}{1+e^{-p}}\left(\frac{1+e^{-p}}{1+e^{-p}} - \frac{1}{1+e^{-p}}\right) \\
&= \frac{1}{1+e^{-p}}\left(1 - \frac{1}{1+e^{-p}}\right) \\
&= \sigma(p)\big(1-\sigma(p)\big)
\end{aligned}
$$

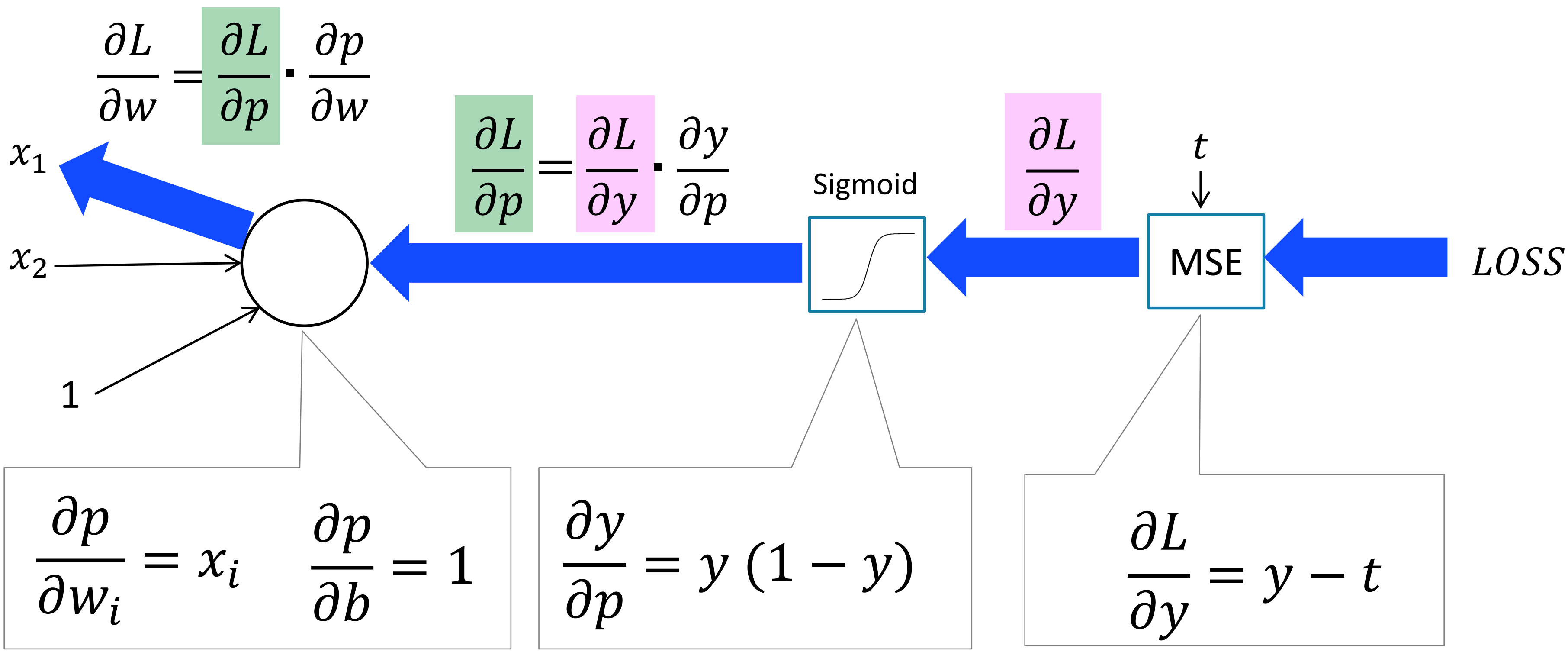
reference

$$
\begin{aligned}
\left(\frac{1}{x}\right)' &= (x^{-1})' \\
&= -\left(x^{-2}\right) \\
&= \frac{-1}{x^2}
\end{aligned}
$$

$$
\begin{aligned}
\left(\frac{1}{f(x)}\right)' &= (f^{-1}(x))' \\
&= -\left(f^{-2}(x)\right)f'(x) \\
&= \frac{-1}{f^2(x)}f'(x)
\end{aligned}
$$

Simple example

Backward calculation



Chain rule

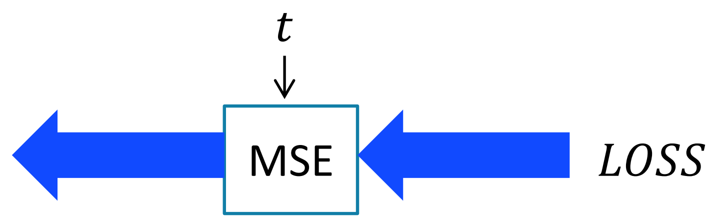
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial w}$$

Implementation for Backward Calculation

MSE.m

```
classdef MSE < handle
  properties
    z;
    t;
  end

  methods
    function loss = forward(obj, z, t)
      obj.z = z;
      obj.t = t;
      [row, col] = size(z);
      loss = sum(sum((z-t).^2)) / (2*col);
    end

    function dL = backward(obj)
      dL = obj.z - obj.t;
    end
  end
end
```

In this class we use variable “z” instead of “y”.

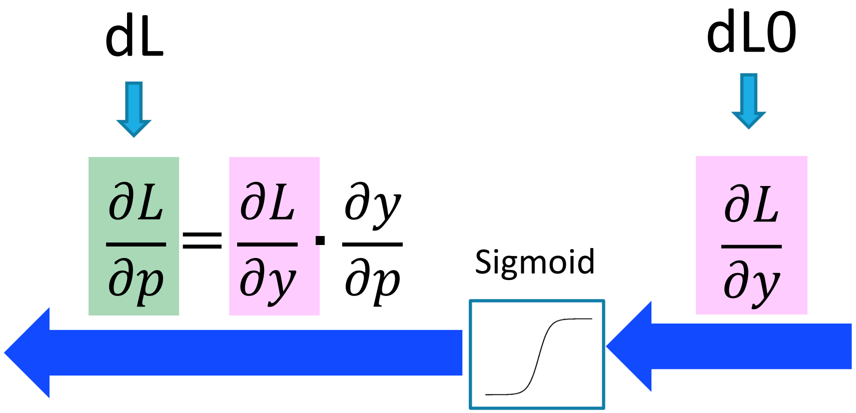


$$\frac{\partial L}{\partial y} = y - t$$

Implementation for Backward Calculation

Sigmoid.m

```
classdef Sigmoid < handle
  properties
    y;
  end

  methods
    function y = forward(obj, x)
      y = 1 ./ (1 + exp(-x));
      obj.y = y;
    end

    function dL = backward(obj, dL0)
      dL = dL0 .* obj.y .* (1.0 - obj.y);
    end
  end
end
```



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

$$\frac{\partial L}{\partial y}$$

$$\frac{\partial y}{\partial p} = y\,(1 - y)$$

Implementation for Backward Calculation

Affine.m

```
classdef Affine < handle
  properties
    weights;
    bias;
    x;
    dw;
    db;
  end

  methods
    function obj = Affine(w,b)
      obj.weights = w;
      obj.bias = b;
    end

    function y = forward(obj, x)
      obj.x = x;
      p = obj.weights * x;
      y = p + obj.bias;
    end

    function dL = backward(obj, dL0)
      dL = obj.weights' * dL0;
      obj.dw = dL0 * obj.x';
      obj.db = sum(dL0, 2);
    end

    function update(obj, learning_rate)
      obj.weights = obj.weights - learning_rate * obj.dw;
      obj.bias = obj.bias - learning_rate * obj.db;
    end
  end
end
```

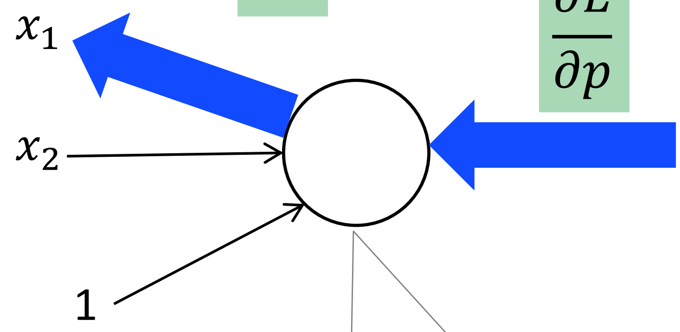
This script is applicable to matrix calculation.
I will explain tomorrow for details!

dL

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p} \cdot \frac{\partial p}{\partial w}$$

dL0

$$\frac{\partial L}{\partial p}$$

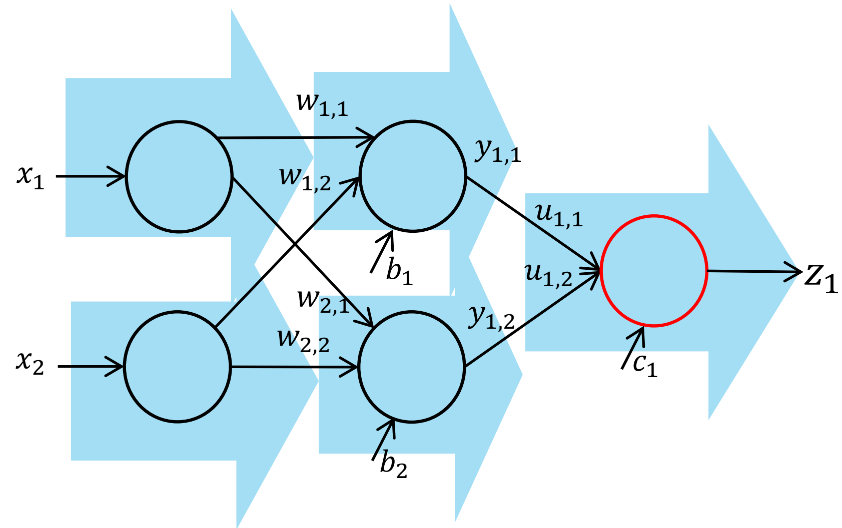


$$\frac{\partial p}{\partial w_i} = x_i \qquad \frac{\partial p}{\partial b} = 1$$

Outline of Learning Neural Network

Feedforward calculation

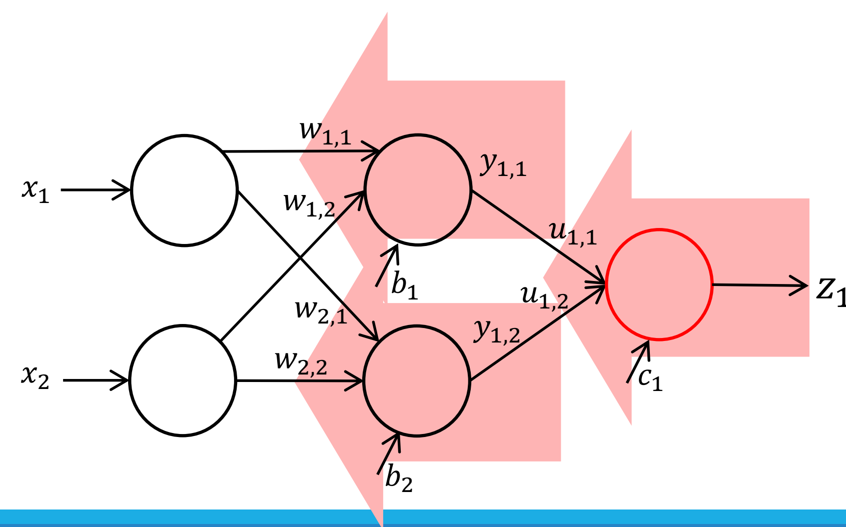
Calculation for outputs from inputs



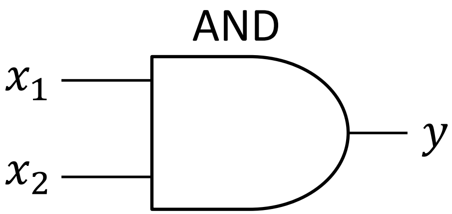
LOSS Calculation

backwards propagation of LOSS

Updating weights and biases to reduce LOSS



Let’s make AND function by learning



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

example2_4.m

```
clear all

xdata = [0,0,1,1;
         0,1,0,1];
labels = [0,0,0,1];
data_num=4;

w = 2.0*rand(1,2) - 1.0;
b = 2.0*rand(1,1) - 1.0;

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = MSE();

% a number of training
EPOCH=1000;
% learning rate
LAMBDA=0.1;

for epoch=1:EPOCH
  p = layer1.forward(xdata);
  y = layer2.forward(p);
  loss(epoch) = layer3.forward(y,labels);

  %calculate gradient
  dy = layer3.backward();
  dp = layer2.backward(dy);
  dx = layer1.backward(dp);

  %learning weights and biases
  layer1.update(LAMBDA);
end

loss

% Display loss change graph
figure(1);
plot(loss)
xlabel('Epoch');
ylabel('LOSS');
```

Exercise2.9

Check the values of output y, layer1.weights and layer1.bias after learning in example2_4.m.

$$w = \begin{bmatrix} \boxed{} & \boxed{} \end{bmatrix} \qquad b = \begin{bmatrix} \boxed{} \end{bmatrix}$$

$$y = \begin{bmatrix} \boxed{} & \boxed{} & \boxed{} & \boxed{} \end{bmatrix}$$

Let's make XOR function by learning

example2_5.m

```
clear all

xdata = [0,0,1,1;
         0,1,0,1];
labels = [0,1,1,0];
data_num=4;

IU = 2;     % a number of input neurons
HU = 2;     % a number of hidden neurons
OU = 1;     % a number of output neurons

% initialize weights and biases
% as random numbers between -1.0 and 1.0.
w = 2.0*rand(HU,IU) - 1.0;
b = 2.0*rand(HU,1) - 1.0;
u = 2.0*rand(OU,HU) - 1.0;
c = 2.0*rand(OU,1) - 1.0;

layer1 = Affine(w,b);
layer2 = Sigmoid();
layer3 = Affine(u,c);
layer4 = Sigmoid();
layer5 = MSE();

EPOCH=1000; % a number of training epochs
LAMBDA=0.1; % learning rate
```

```
for epoch=1:EPOCH
  p = layer1.forward(xdata);
  y = layer2.forward(p);
  q = layer3.forward(y);
  z = layer4.forward(q);
  loss(epoch) = layer5.forward(z,labels);

  %calculate gradient
  dz = layer5.backward();
  dq = layer4.backward(dz);
  dy = layer3.backward(dq);
  dp = layer2.backward(dy);
  dx = layer1.backward(dp);

  %learning weights and biases
  layer1.update(LAMBDA);
  layer3.update(LAMBDA);
end

loss

% Display loss change graph
figure(1);
plot(loss)
xlabel('Epoch');
ylabel('LOSS');
```

Exercise2.10

Check the values of weights and biases after learning in example2_5.m and write down these values to one places of decimals. Then, calculate XOR output by your hand calculation with step function.

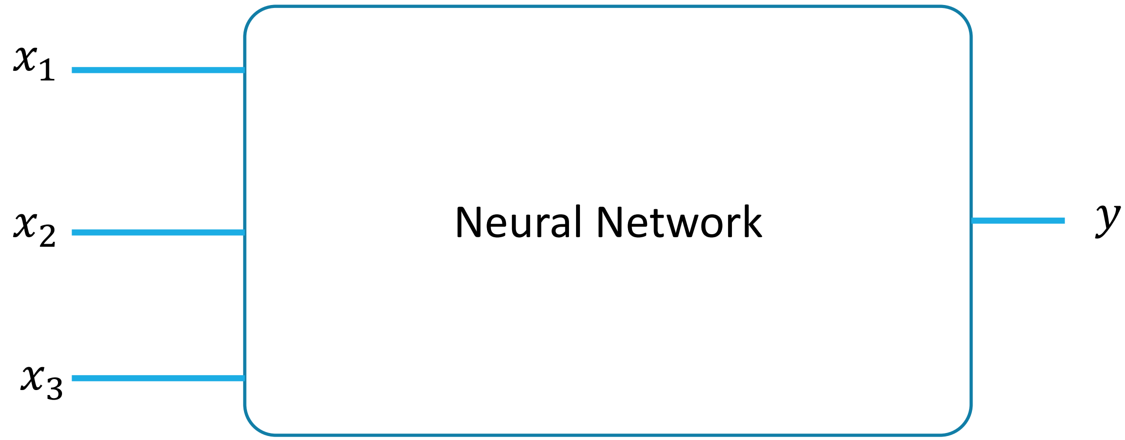
$$w = \begin{bmatrix} \boxed{\qquad} & \boxed{\qquad} \\ \boxed{\qquad} & \boxed{\qquad} \end{bmatrix} \qquad b = \begin{bmatrix} \boxed{\qquad} \\ \boxed{\qquad} \end{bmatrix}$$

$$u = \begin{bmatrix} \boxed{\qquad} & \boxed{\qquad} \end{bmatrix} \qquad \mathrm{c} = \begin{bmatrix} \boxed{\qquad} \end{bmatrix}$$

Exercise2.11

At first, freely define a 3 input 1 output logic function.
Then freely design the neural network and make the logic function by learning.

X1	X2	X3	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	



For example
・Only 1 neuron
・Single layer NN with 3 neuron
・Two layer NN with 3 neuron in in hidden layer and 3 neuron in output layer