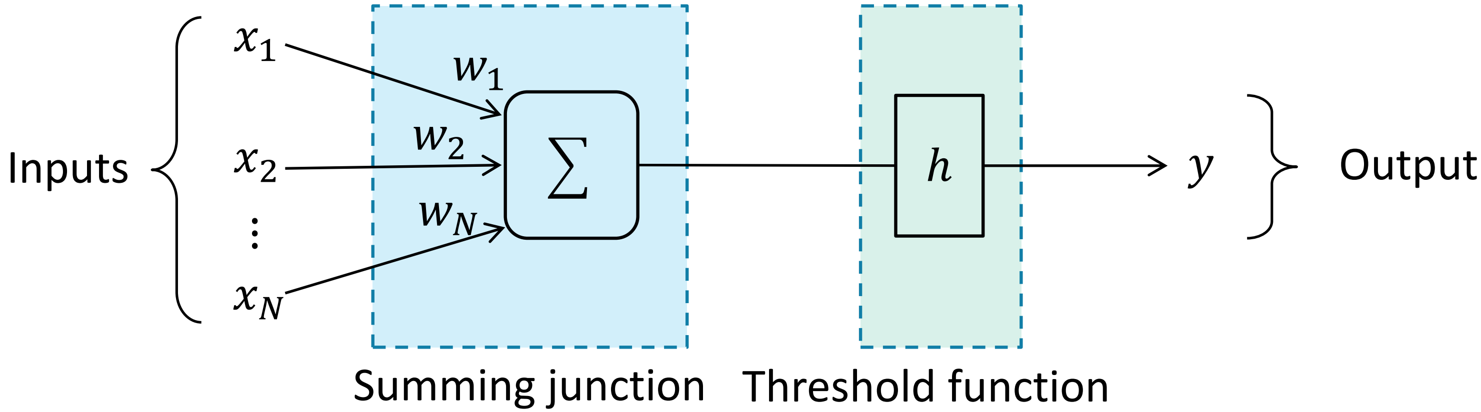
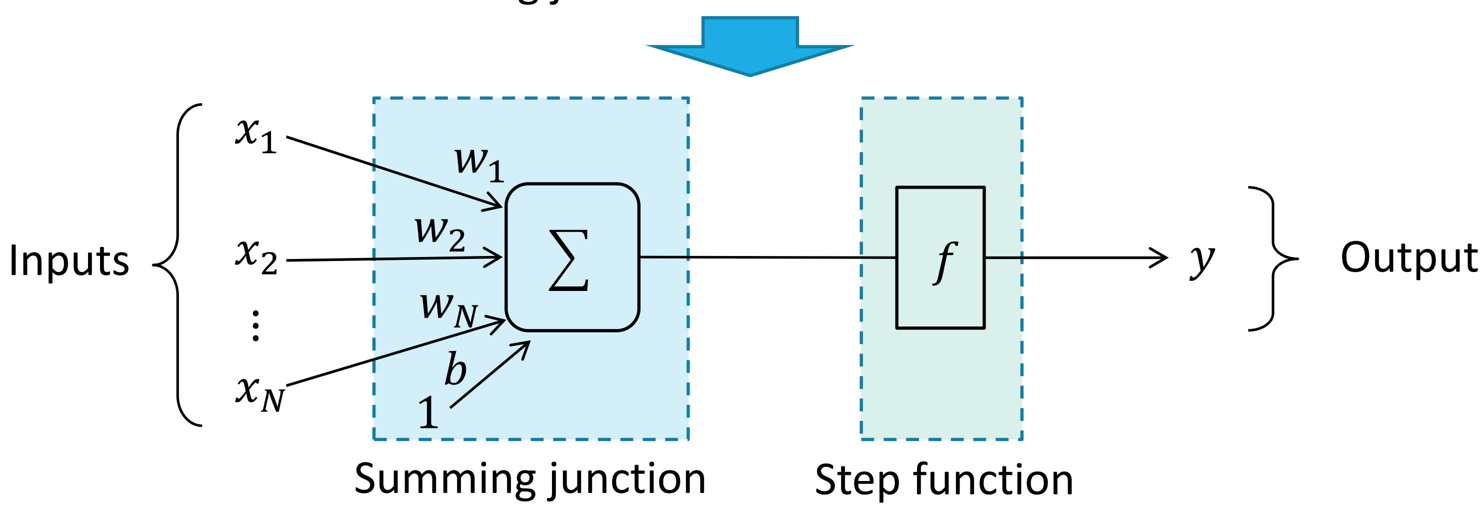
Day3

Learning Neural Network
- Backpropagation -

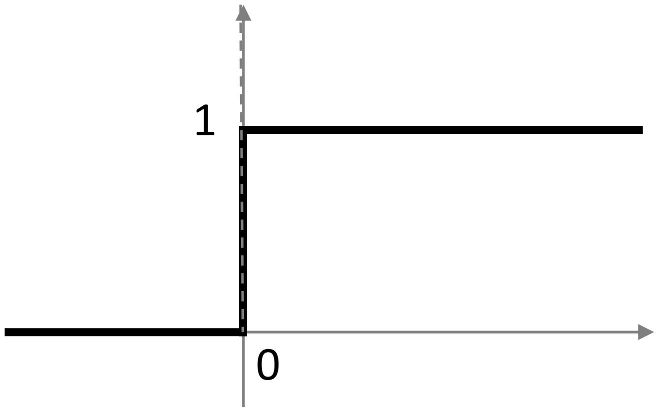
【Review】Introduction of Activation function





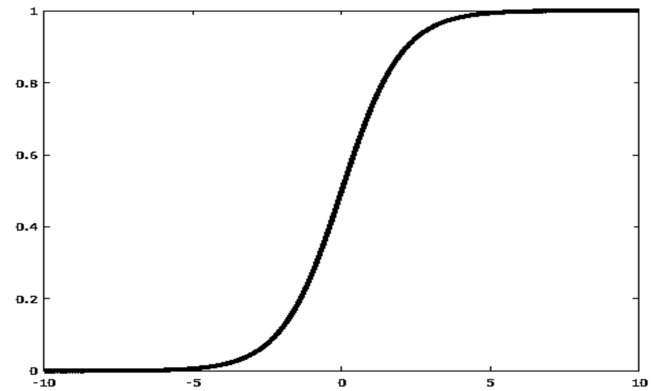
So far, we adopted a step function as $f(x)$. However, we can use another function instead of a step function as $f(x)$. Generally, $f(x)$ is called "Activation function" and several typical function is proposed to $f(x)$. One of the most popular and useful function is "Sigmoid function".

【Review】Variety of Activation Function



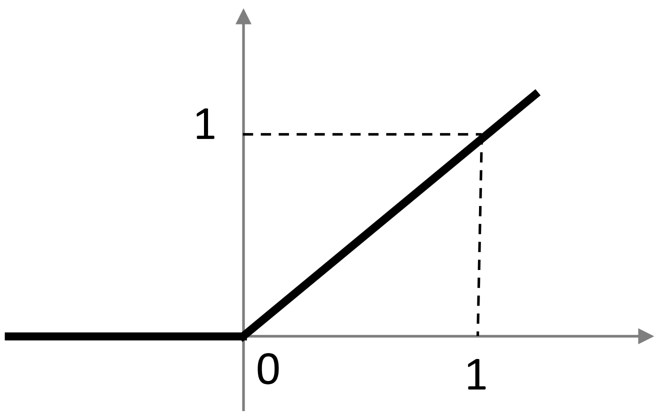
Step function

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



Sigmoid function

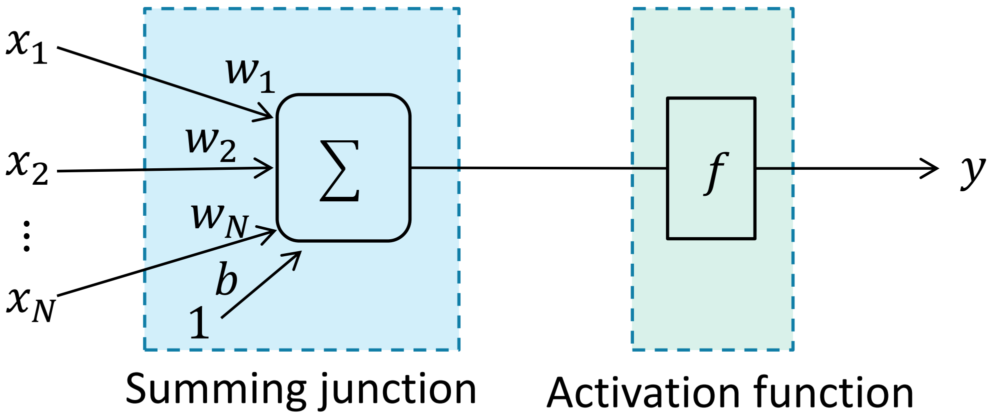
$$f(a) = \frac{1}{1 + e^{-a}}$$

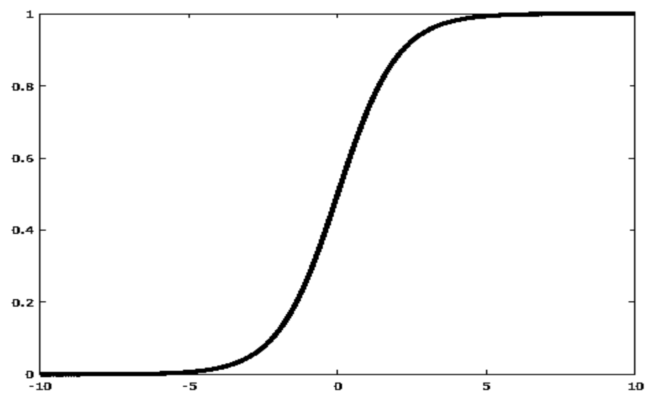


ReLU (Rectified Linear Unit)

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

【Review】What's Sigmoid Neuron



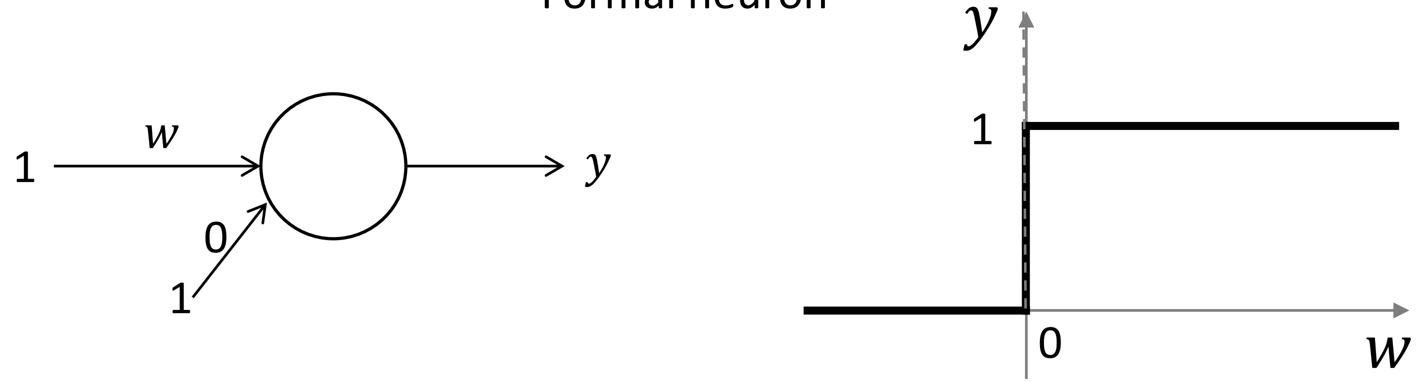


Sigmoid function

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

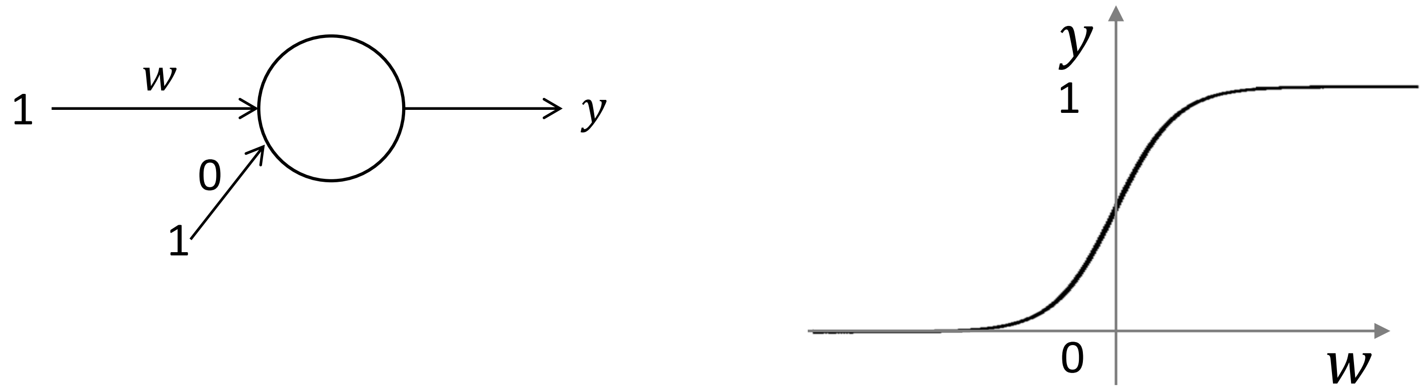
【Review】Difference between Formal Neuron and Sigmoid Neuron

Formal neuron



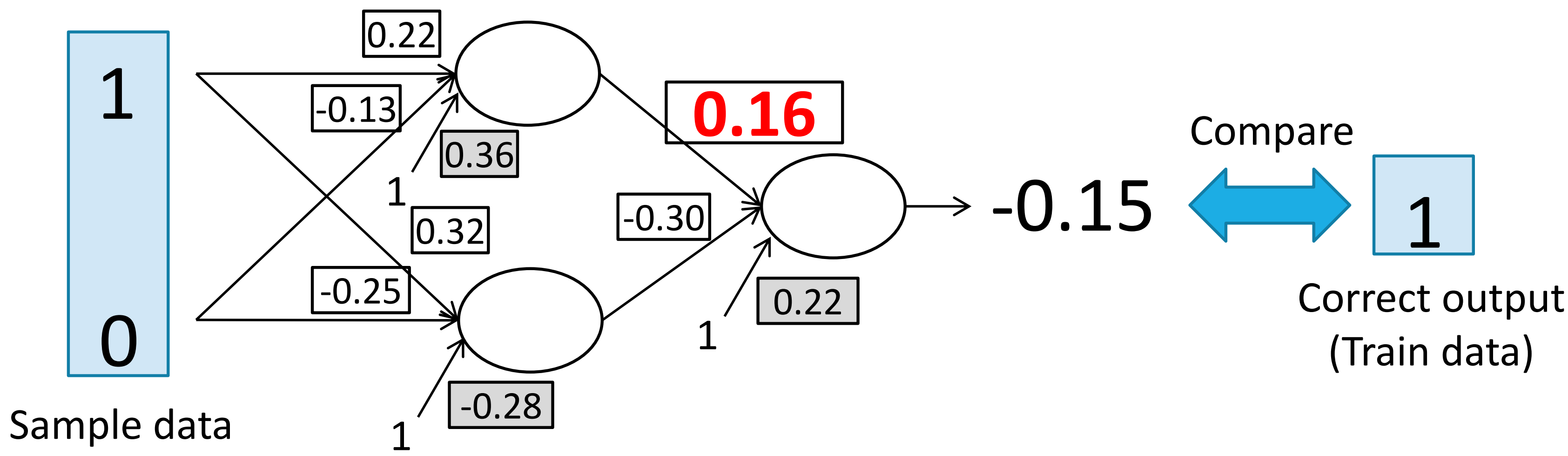
The output value is 0 or 1 (two-value output)

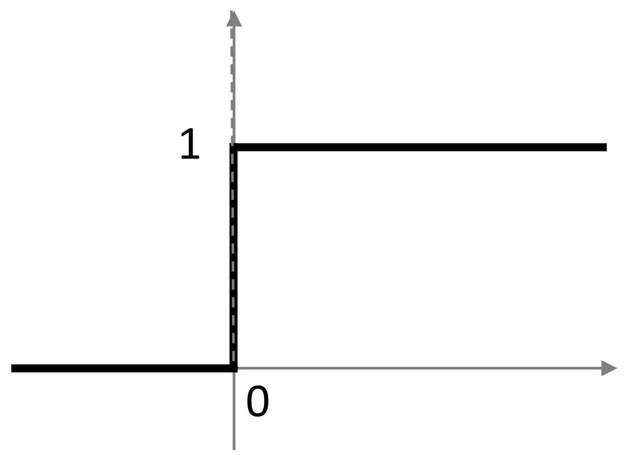
Sigmoid neuron

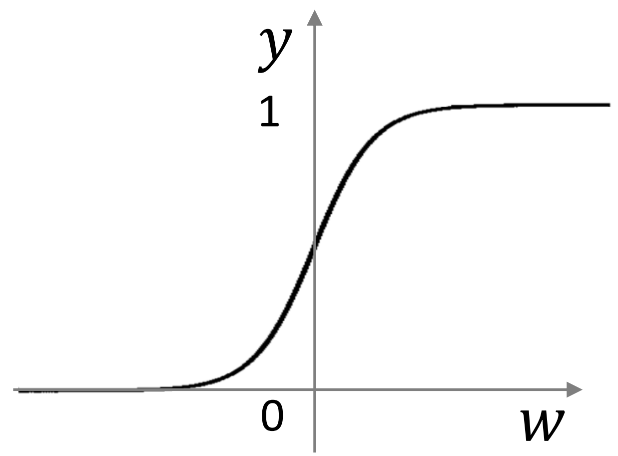


The output value is a real number between 0 and 1.

【Review】Difference between Formal Neuron and Sigmoid Neuron







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

【Review】Introduction of 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 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) .

【Appendix】Another function for Loss function

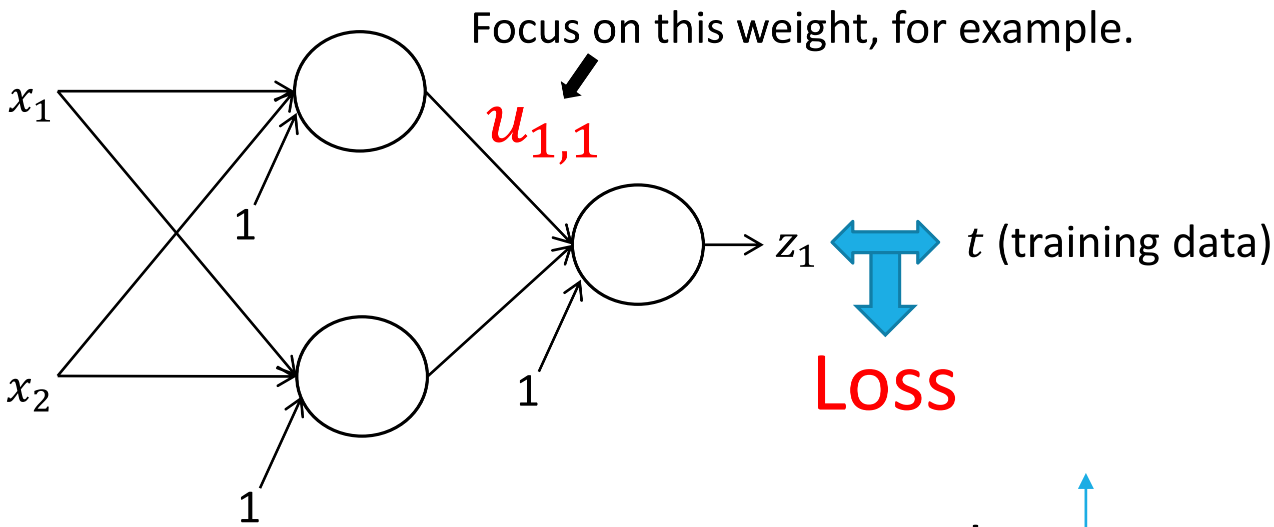
Another useful cost function is “Cross Entropy” as follows

$$LOSS = -\frac{1}{N}\sum_{k}\{t_k \log z_k + (1 - t_k)\log(1 - z_k)\}$$

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

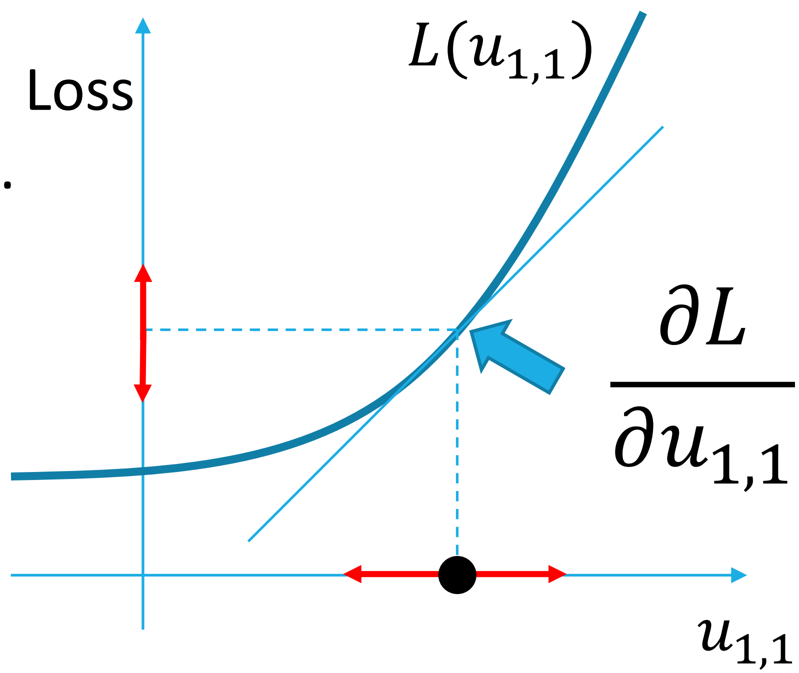
However, intuitively hard to understand

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

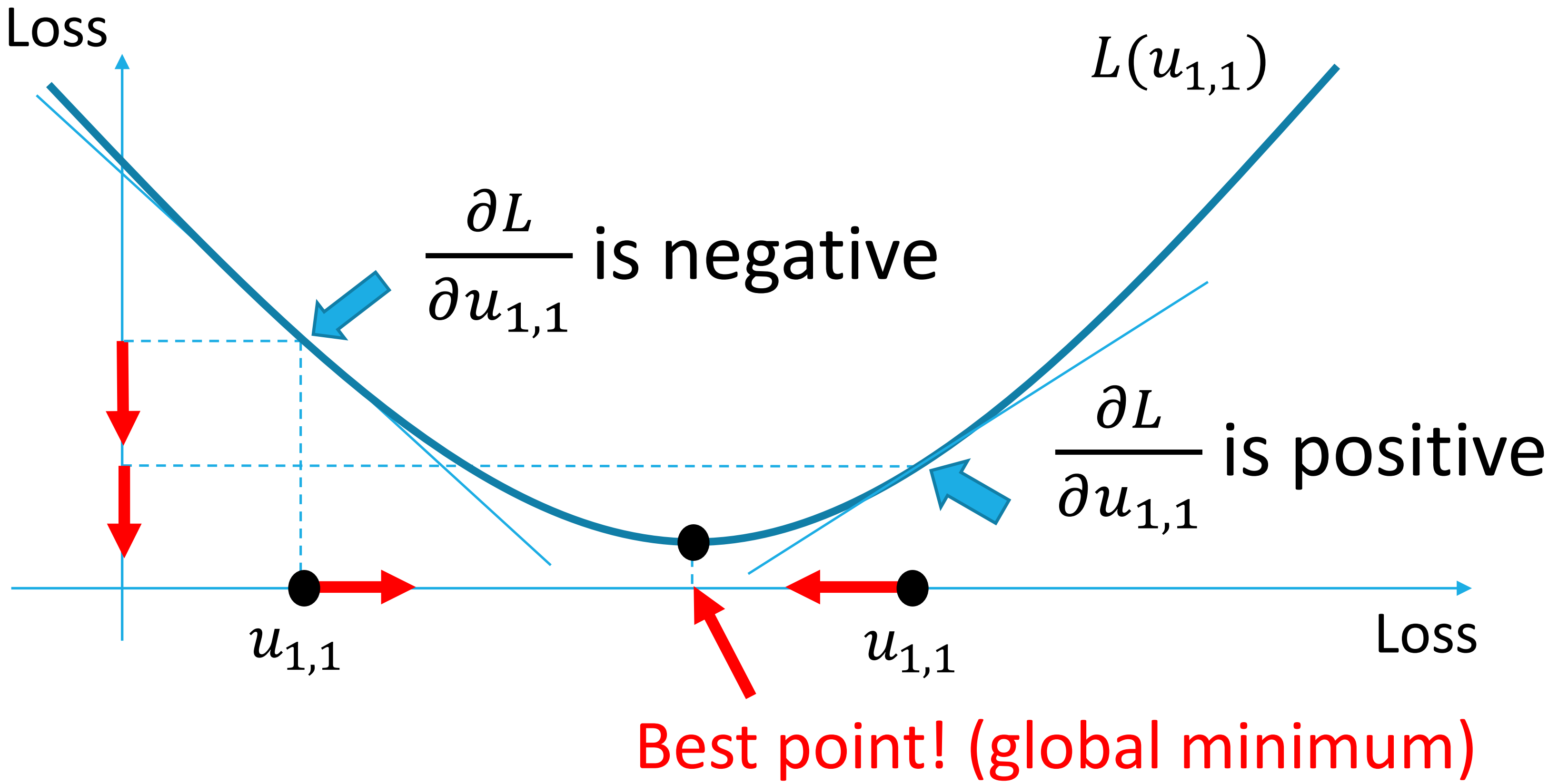


We can calculate slope of Loss function.

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



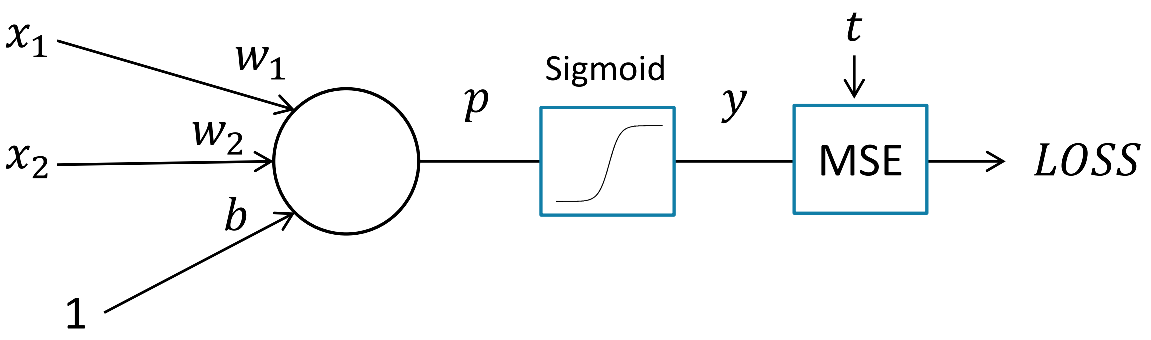
【Review】How can we reduce the LOSS (2)



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

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

【Review】Simple example



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

【Review】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,

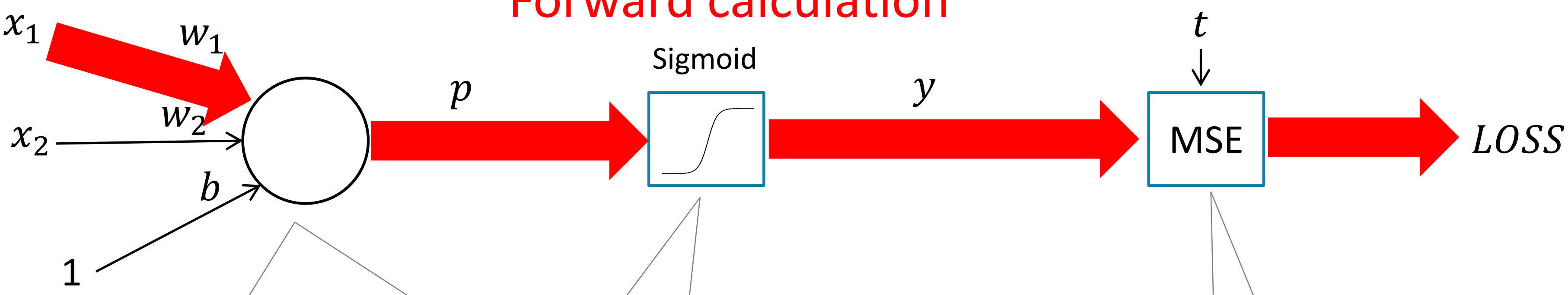
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Furthermore,

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

【Review】Simple example

Forward calculation



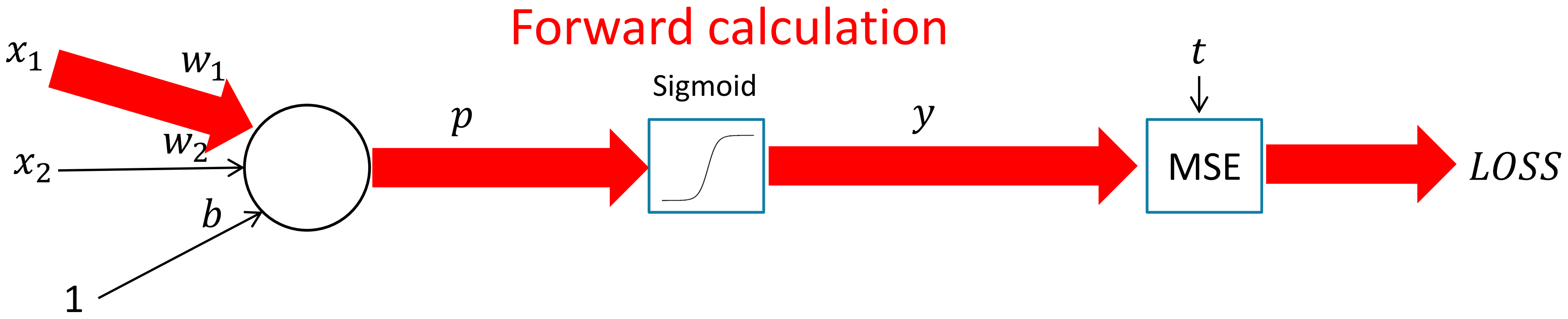
$$p = \sum_i w_i x_i + b = w_1 x_1 + w_2 x_2 + b$$

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

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

MSE is always positive value.

【Review】Simple example



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

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

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

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

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

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

【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

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

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

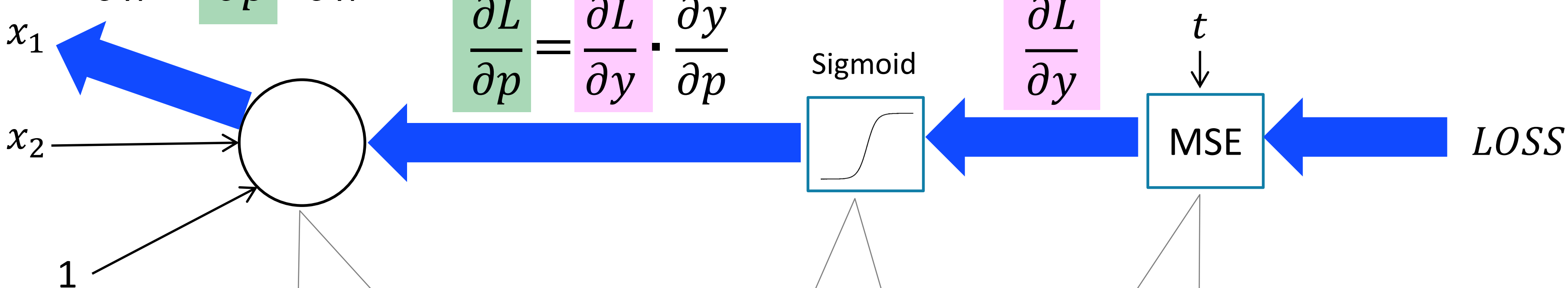
【Review】Simple example

Backward calculation

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

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

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



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

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

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

Chain rule

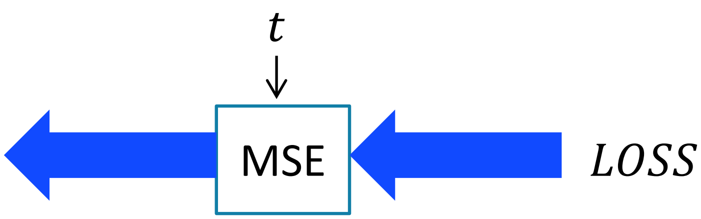
$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial p} \cdot \frac{\partial p}{\partial w}$$

【Review】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”.



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

【Review】Implementation for Backward Calculation

Sigmoid.m

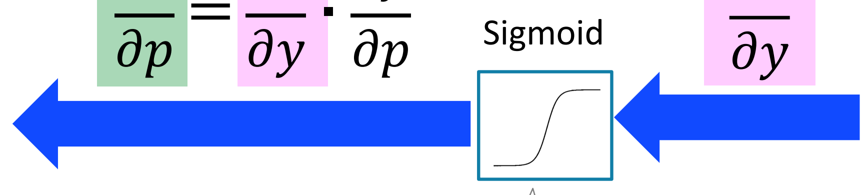
```
classdef Sigmoid < handle
  properties
    y;
  end

  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
```

dL

dL0

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

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



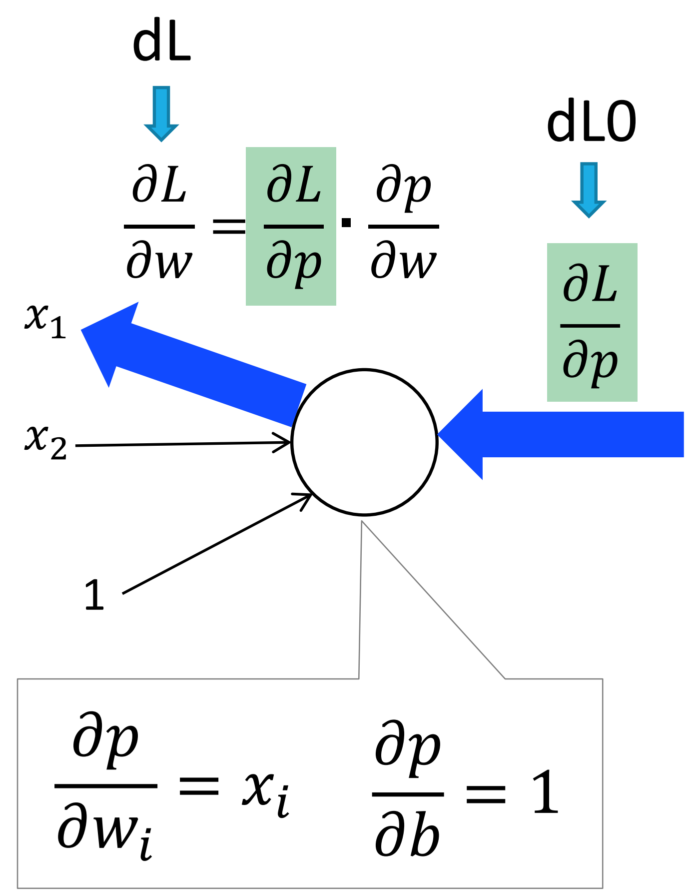
$$\frac{\partial y}{\partial p} = y\,(1 - y)$$

【Review】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!



【review】Outline of Learning Neural Network

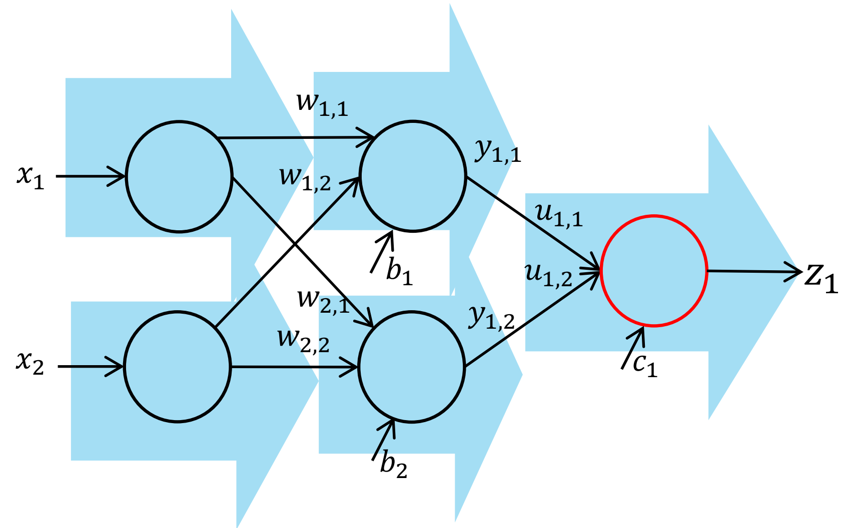
Feedforward calculation

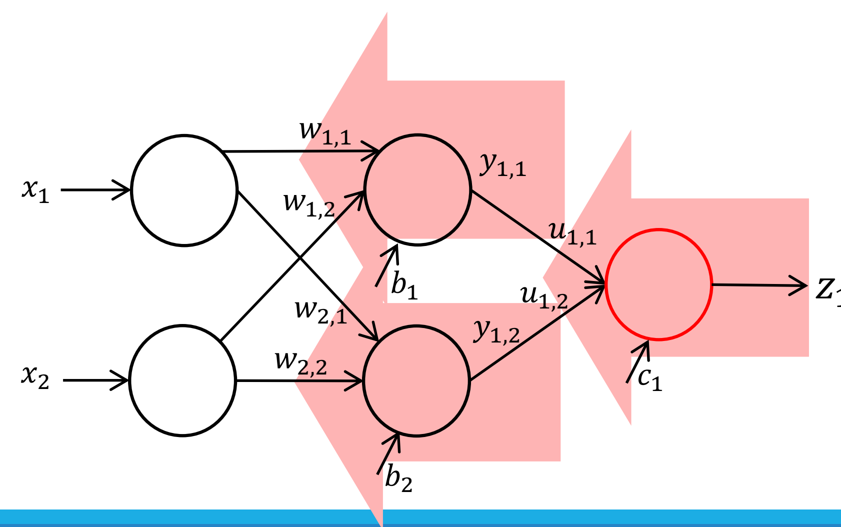
Calculation for outputs from inputs

LOSS Calculation

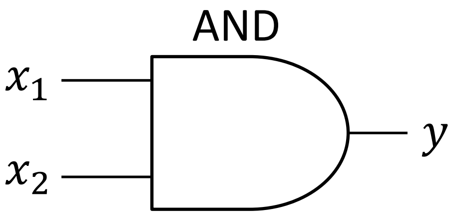
backwards propagation of LOSS

Updating weights and biases to reduce LOSS





【Review】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');
```

【Review】Exercise2.9

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

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

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

【Review】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');
```

【Review】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.

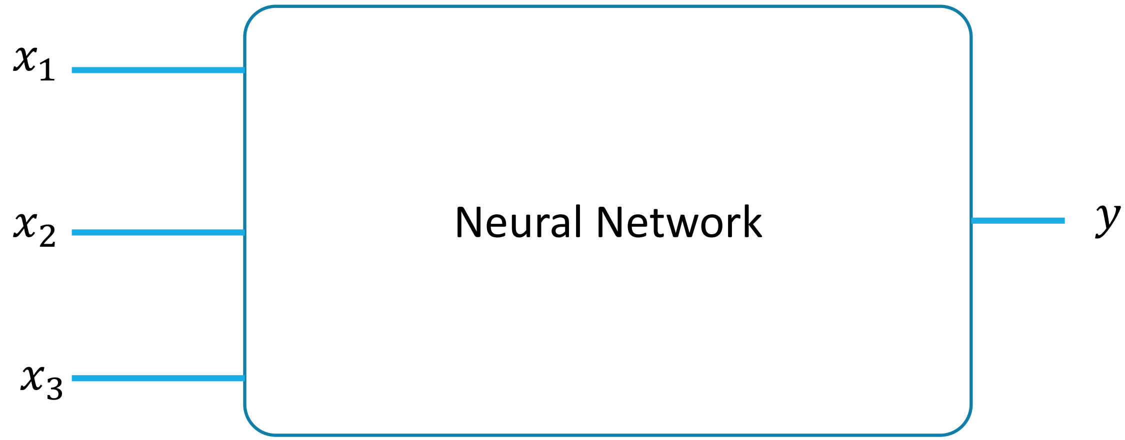
$$w = \begin{bmatrix} \boxed{} & \boxed{} \\ \boxed{} & \boxed{} \end{bmatrix} \qquad b = \begin{bmatrix} \boxed{} \\ \boxed{} \end{bmatrix}$$

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

【Review】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.

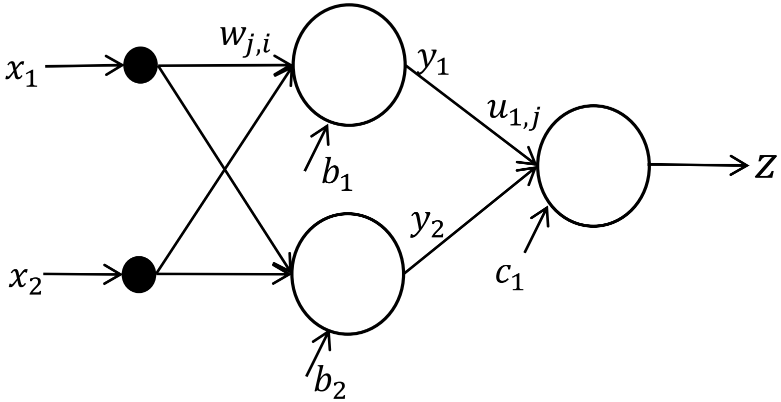
X1	X2	X3	Y
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	



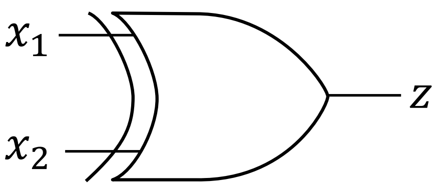
For example
・Only 1 neuron
・Single layer NN with 3 neuron
・Two layer NN with 3 neuron in in hidden layer and 3 neuron in output layer

Exercise 3.1

Construct a neural network with 2 input neurons, 2 hidden neurons and 1 output neuron as follows. Then learning the neural network for 2-input XOR function. Truth table for 2-input XOR is shown below. After learning, please check loss value, obtained weights and biases and check feed forward calculation with step function by yourself.



2-input XOR

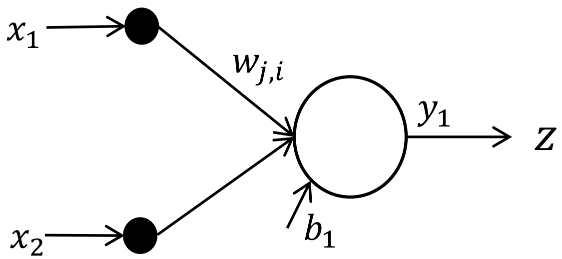


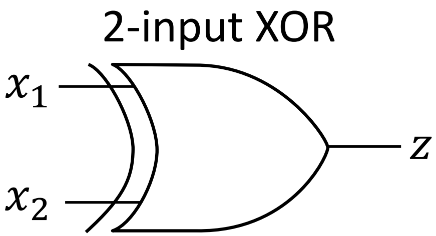
Truth table for 2-input XOR

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

Exercise 3.2

Construct a neural network with 2 input neurons and only 1 output neuron as follows. Then learning the neural network for 2-input XOR function. Truth table for 2-input XOR is shown below. After learning, please check obtained weights and biases and check feed forward calculation with step function by yourself.



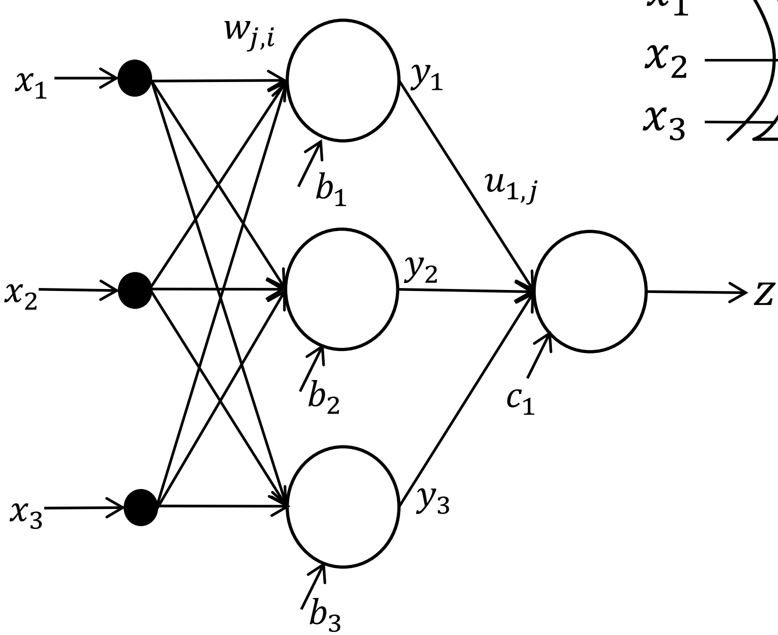


Truth table for 2-input XOR

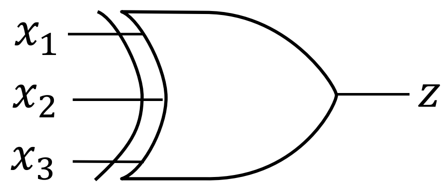
x_1	x_2	y
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

Exercise 3.3

Construct a neural network with 3 input neuron, 3 hidden neuron and 1 output neuron as follows. Then learning the neural network for 3-input XOR function. Truth table for 3-input XOR is shown below. After learning, please check obtained weights and biases and confirm feed forward calculation with step function by yourself.



3-input XOR

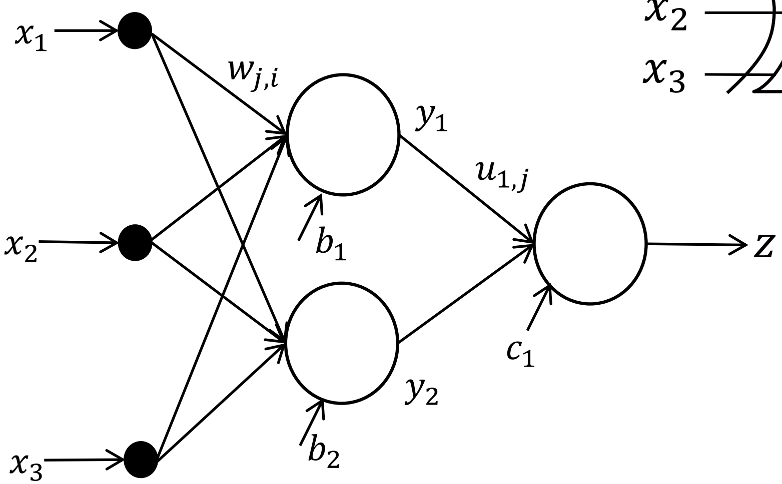


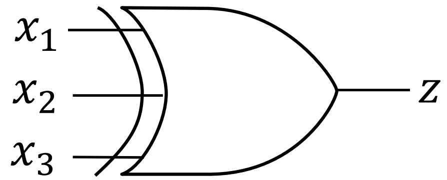
Truth table for 3-input XOR

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

Exercise 3.4

Construct a neural network with 3 input neuron, 2 hidden neuron and 1 output neuron as follows. Then learning the neural network for 3-input XOR function. Truth table for 3-input XOR is shown below. After learning, please check obtained weights and biases and confirm feed forward calculation with step function by yourself.



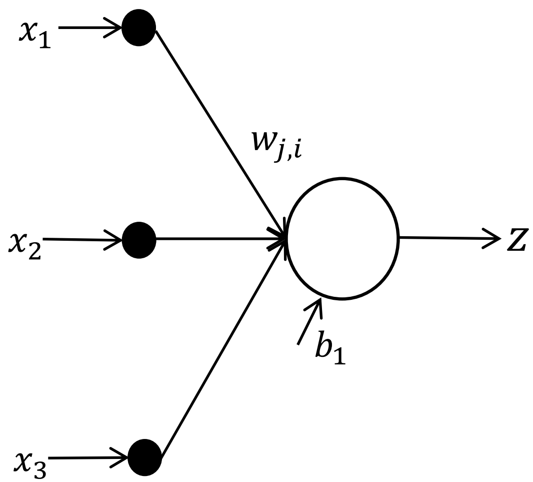


Truth table for 3-input XOR

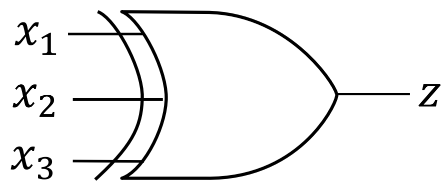
x_1	x_2	x_3	y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Exercise 3.5

Construct a neural network **with 3 input neurons and only 1 output neuron** as follows. Then learning the neural network for 3-input XOR function. Truth table for 3-input XOR is shown below. After learning, please check obtained weights and biases and confirm feed forward calculation with step function by yourself.



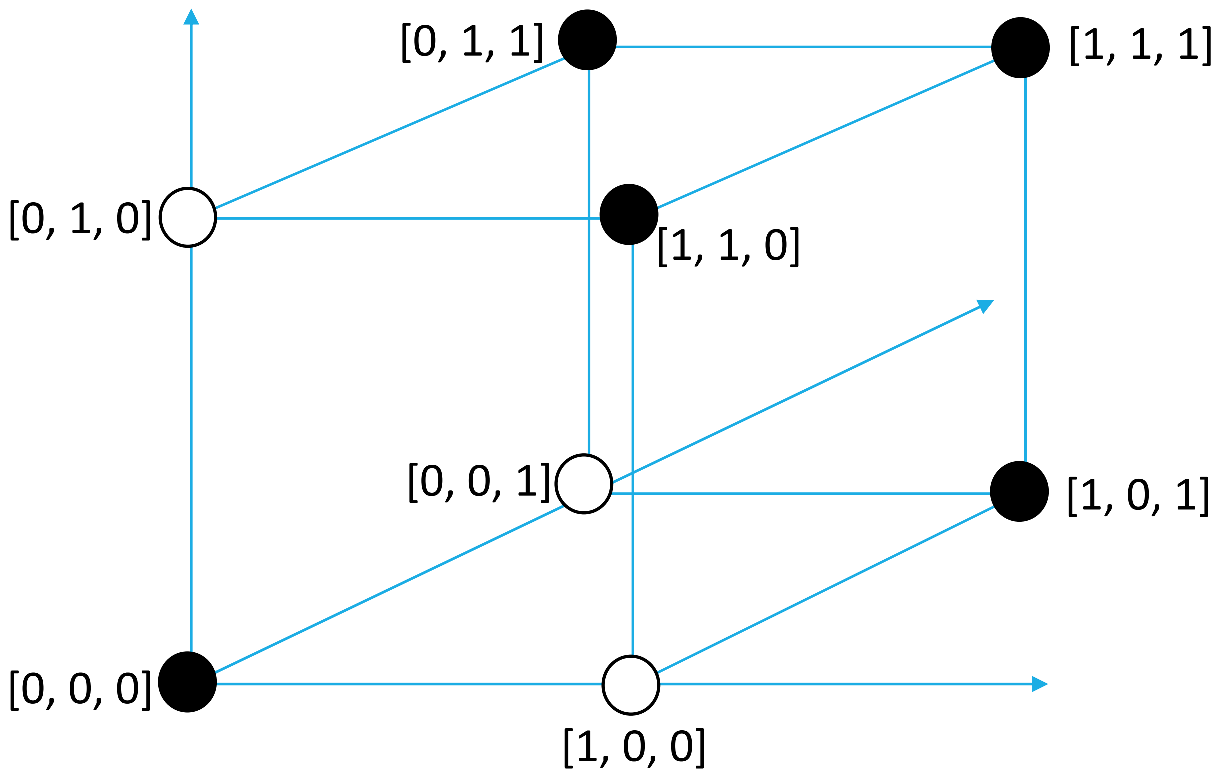
3-input XOR



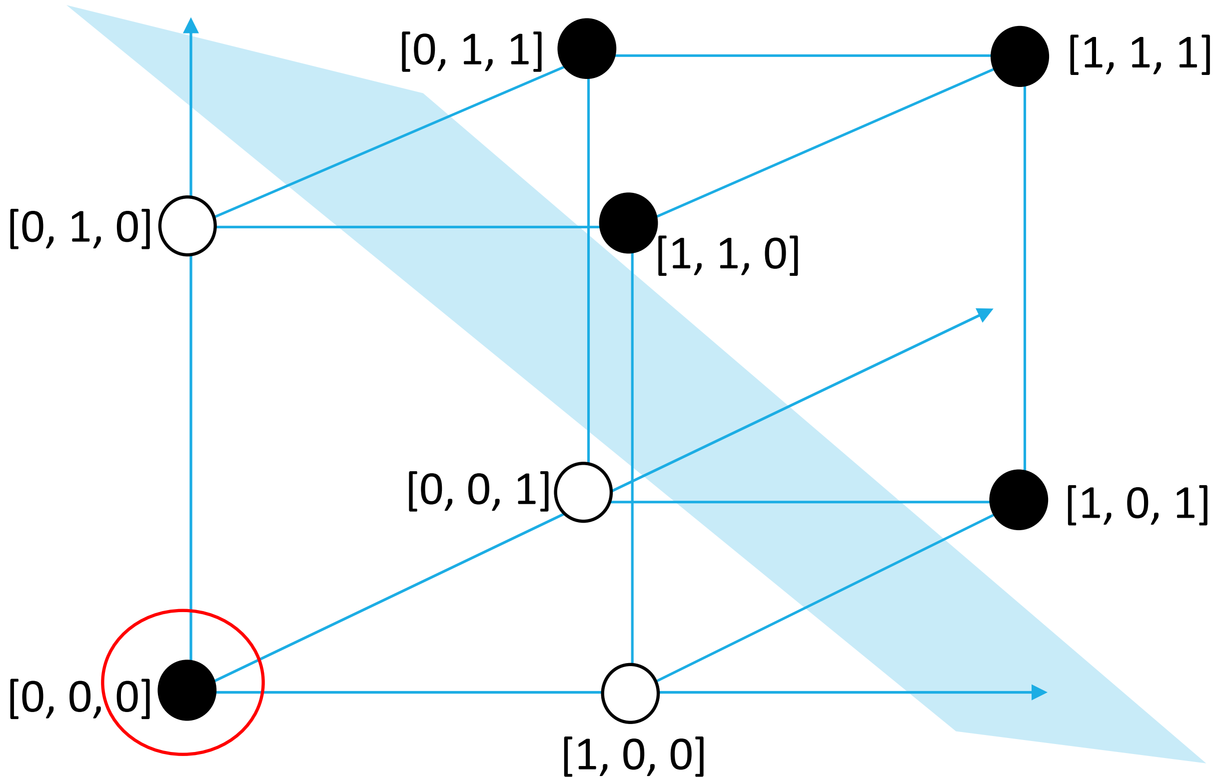
Truth table for 3-input XOR

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

3-input XOR



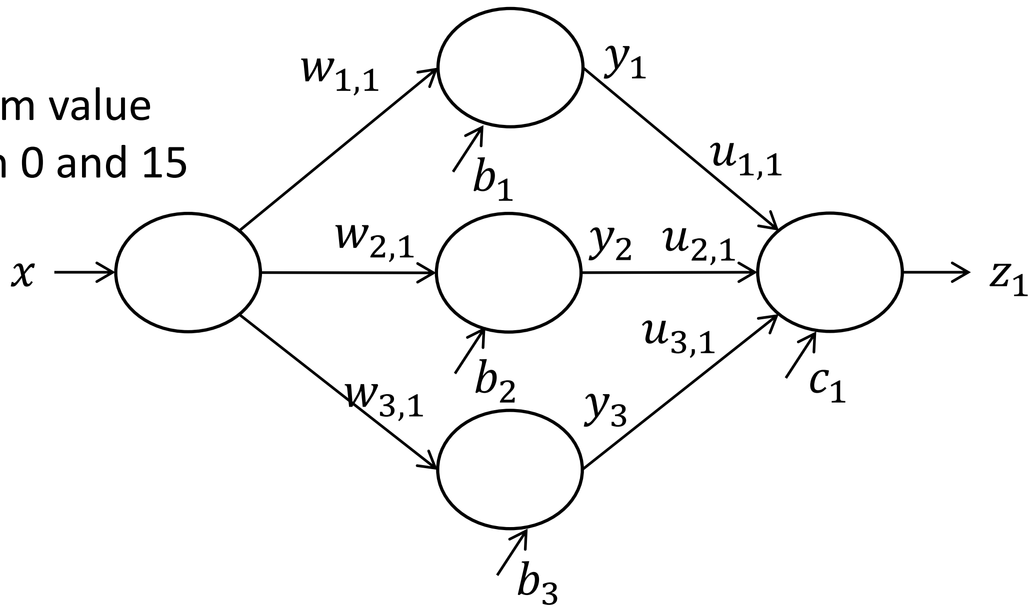
3-input XOR

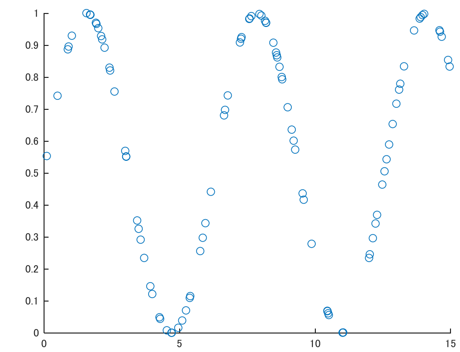


NN Learning for Mathematical Functions

Learning sin function using neural network

Random value
between 0 and 15





Training data from sin finction

Generating sin data and display the result graph

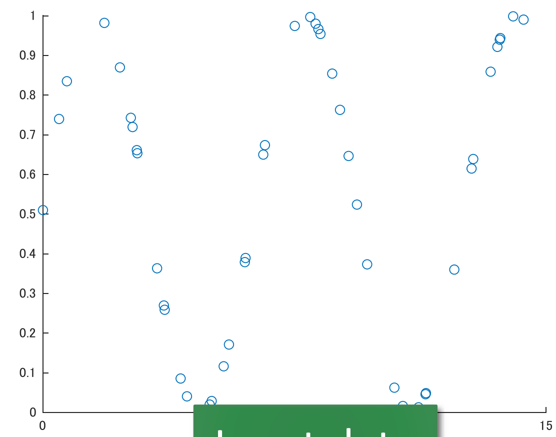
Example3_1.m

```
data_num=50;
xdata = 15*rand(1,data_num);
labels = (sin(xdata)+1)/2; % between 0 and 1

figure(1);
scatter(xdata, labels);
```

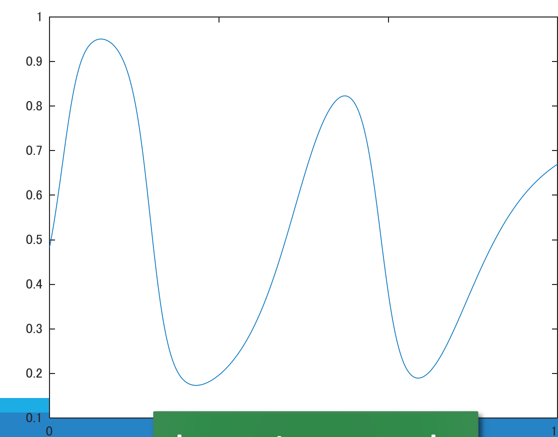
The labels are normalized to a value from 0 to 1 because output value (output of sigmoid function) is between 0 and 1.

Neural network construction
and learning code.
(Please substitute various values for hidden neuron size, epoch number, learning rate and try.)

```
% Display loss change graph
figure(2);
plot(loss)
axis([0 EPOCH 0 max(loss)])
xlabel('Epoch');
ylabel('LOSS');

% Display output graph
xt = [0:0.01:15];
pt = layer1.forward(xt);
yt = layer2.forward(pt);
qt = layer3.forward(yt);
zt = layer4.forward(qt);

figure(3);
plot(xt,zt)
```



Input data



learning result

Exercise 3.6

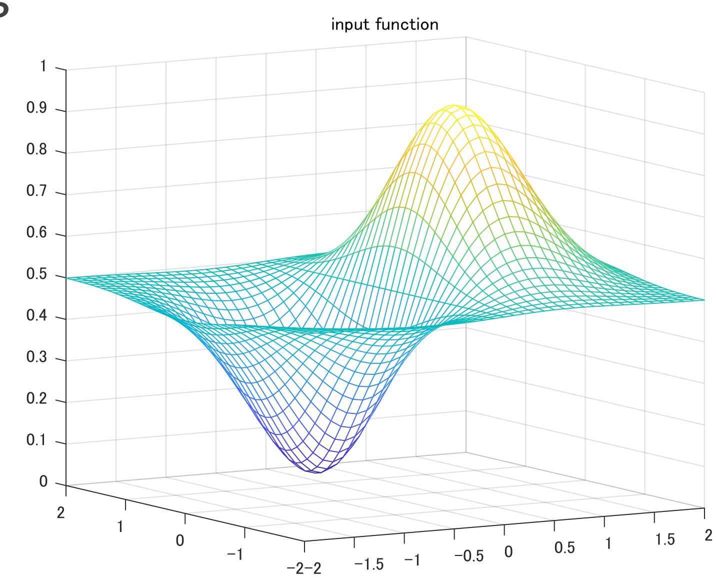
Construct neural network with 1 input, 3-30 hidden, 1 output neurons. Then, learning neural network for sin function normalized to a value from 0 to 1.

Change a number of hidden neuron, learning rate, a number of epoch and a number of xdata (input data) and consider about the output results.

Exercise 3.7

Construct neural network with 2 input (x_1, x_2), 3-30 hidden, 1 output neurons. Then, learning neural network for following function.

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



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

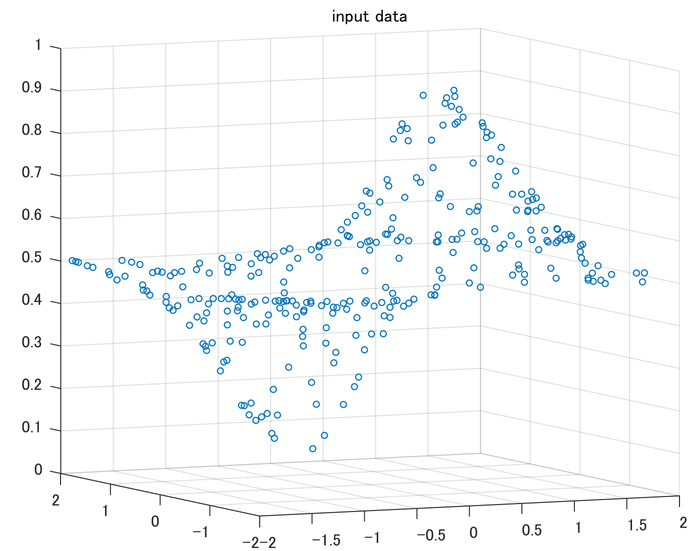
Tips for exercise3_7.m

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

Tips for exercise3_7.m

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

```
% Display output graph
[X1 X2] = meshgrid(-2:0.1:2);
pt = layer1.forward([X1(:)';X2(:)']);
yt = layer2.forward(pt);
qt = layer3.forward(yt);
zt = layer4.forward(qt);
figure(3);
zsize = sqrt(size(zt));
mesh(X1,X2,reshape(zt,[zsize(2),zsize(2)]));
title('learning results')
```



Learning result

?