

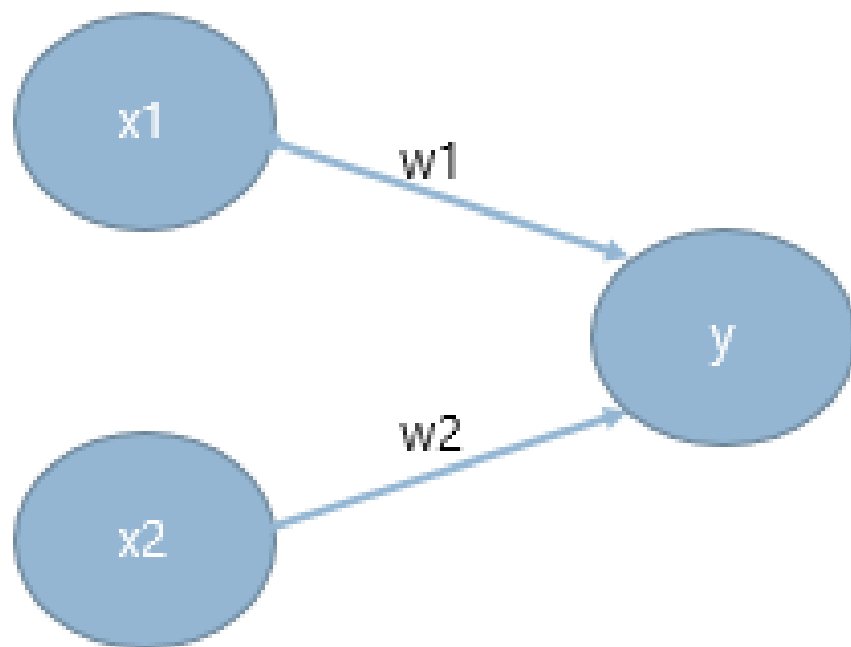
1

딥러닝 입문

딥러닝에 대해 이해하고 이를 실습해 본다.

퍼셉트론

입력으로 2개 신호를 받는 퍼셉트론의 예.
 x_1, x_2 는 입력신호, y 는 출력신호, w_1, w_2 는 가중치



퍼셉트론 동작원리









$$0 \quad (w_1 * x_1 + w_2 * x_2 \leq \theta)$$

$$y = \begin{cases} 0 & (w_1 * x_1 + w_2 * x_2 \leq \theta) \\ 1 & (w_1 * x_1 + w_2 * x_2 > \theta) \end{cases}$$

θ (theta) : 임계값

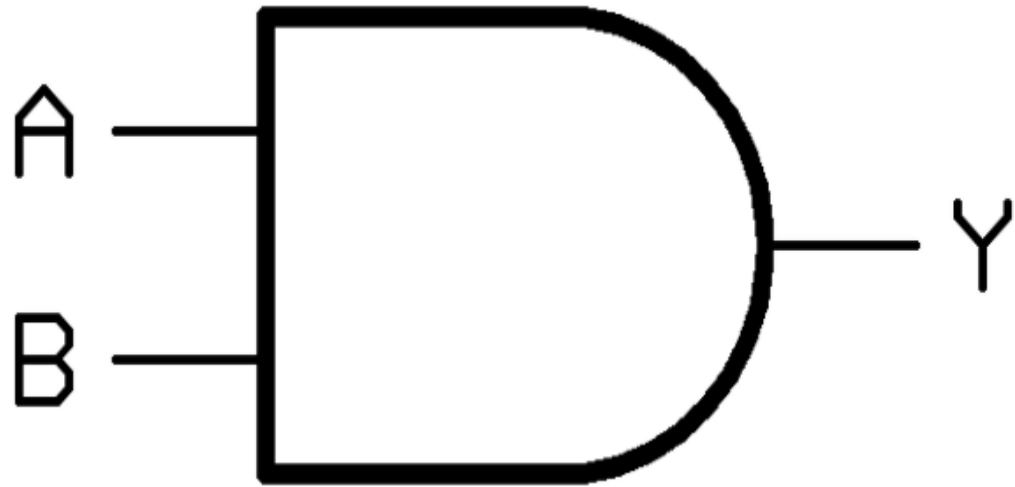
논리 회로

045. 논리 게이트

게이트	기호	의미	진리표	논리식															
AND		입력신호가 모두 1일 때 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	$Y = A \cdot B$ $Y = AB$
A	B	Y																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR		입력신호 중 1개만 1이어도 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$Y = A + B$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NOT		입력된 정보를 반대로 변환하여 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	$Y = A'$ $Y = \overline{A}$									
A	Y																		
0	1																		
1	0																		
BUFFER		입력된 정보를 그대로 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	Y	0	0	1	1	$Y = A$									
A	Y																		
0	0																		
1	1																		
NAND		NOT + AND, 즉 AND의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	$Y = \overline{A \cdot B}$ $Y = \overline{AB}$
A	B	Y																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR		NOT + OR, 즉 OR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	$Y = \overline{A + B}$
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
XOR		입력신호가 모두 같으면 0, 한 개라도 틀리면 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$Y = A \oplus B$ $Y = AB + \overline{A}\overline{B}$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
XNOR		NOT + XOR, 즉 XOR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1	$Y = A \odot B$ $Y = \overline{A \oplus B}$ $Y = AB + \overline{A}\overline{B}$
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

논리 회로 - AND

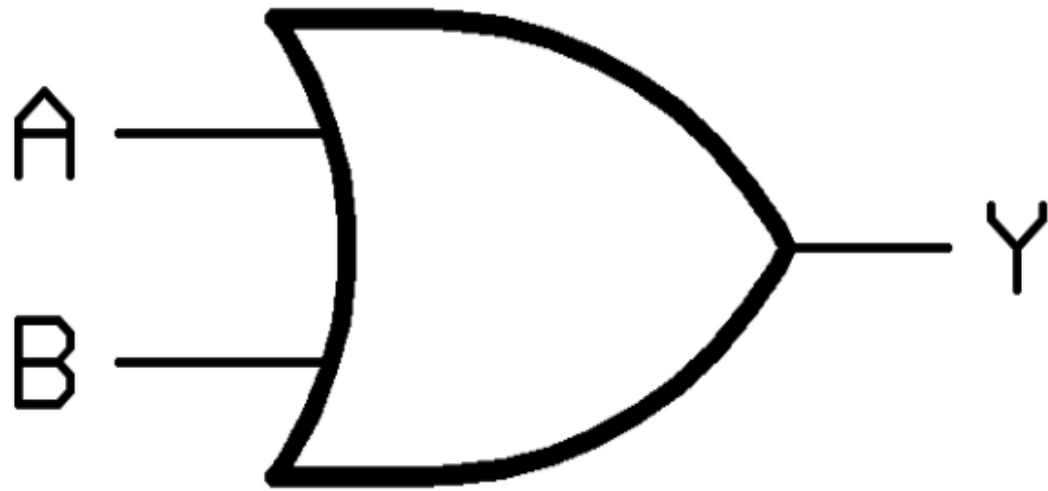
AND 게이트는 입력이 둘이고, 출력이 하나이다.
두 입력이 모두 1일 때만 1을 출력, 그 외에는 0을 출력



A	B	Y
0	0	0
1	0	0
0	1	0
1	1	1

논리 회로 - OR

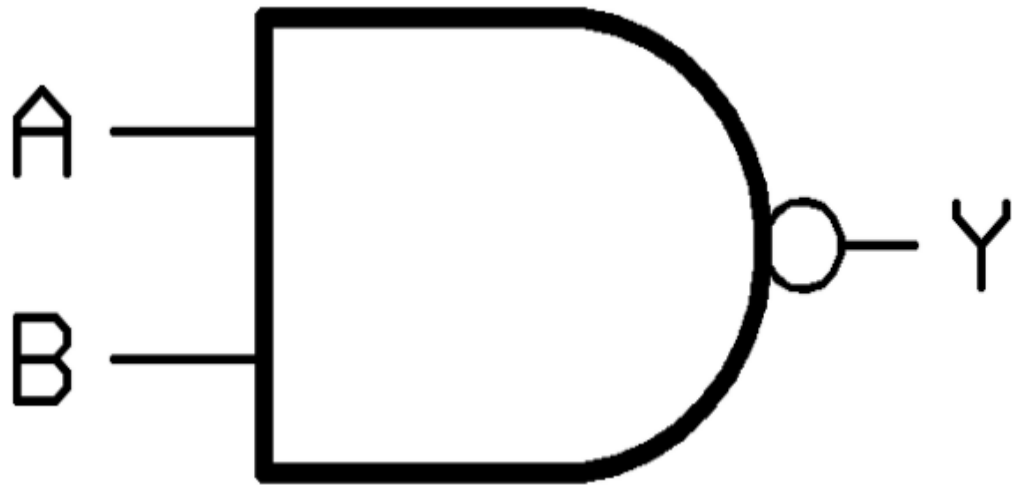
OR 게이트는 입력이 둘이고, 출력이 하나이다.
입력 신호 중 하나 이상이 1이면 출력이 1이 되는 논리 회로.



A	B	Y
0	0	0
1	0	1
0	1	1
1	1	1

논리 회로 - NAND

AND 게이트는 입력이 둘이고, 출력이 하나이다.
두 입력이 모두 1일 때만 0을 출력, 그 외에는 1을 출력



A	B	Y
0	0	1
1	0	1
0	1	1
1	1	0

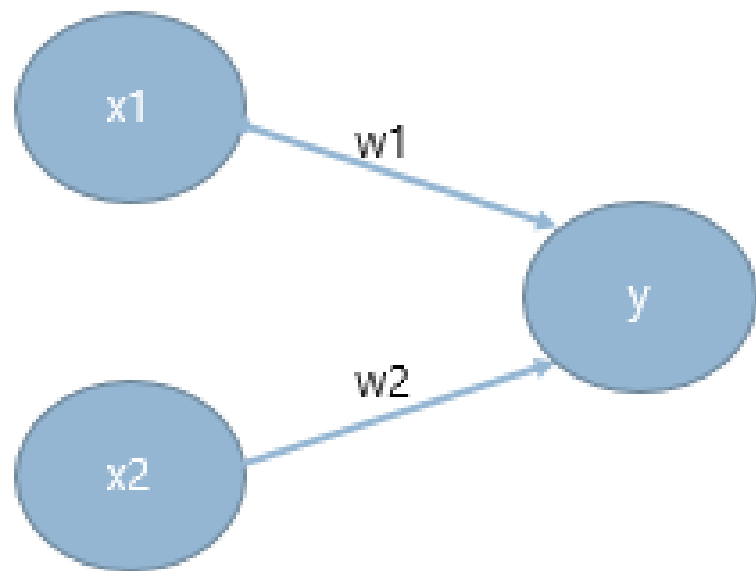
퍼셉트론 구현 – Simple

각각의 입력(x_1 , x_2)에 따른 AND 게이트의 결과를 확인할 수 있다.

```
In [7]: def AND(x1, x2):  
        w1, w2, theta=0.5, 0.5, 0.7  
        tmp=x1*w1 + x2*x2  
        if tmp <= theta:  
            return 0  
        elif tmp > theta:  
            return 1  
  
        print(AND(0,0))  
        print(AND(1,0))  
        print(AND(0,1))  
        print(AND(1,1))
```

```
0  
0  
1  
1
```


퍼셉트론 동작원리



$$y = \begin{cases} 0 & (w1*x1 + w2*x2 - \theta \leq 0) \\ 1 & (w1*x1 + w2*x2 - \theta > 0) \end{cases}$$

참고

θ (theta) : 임계값

퍼셉트론 구현 – 가중치와 편향 도입

x는 입력, w는 가중치, b는 편향

```
In [11]: import numpy as np
x = np.array([0,1])      # 입력
w = np.array([0.5, 0.5])  # 가중치
b = -0.7                 # 편향
w * x
```

```
Out[11]: array([ 0. ,  0.5])
```

```
In [12]: np.sum(w*x)
```

```
Out[12]: 0.5
```

```
In [13]: np.sum(w*x) + b
```

```
Out[13]: -0.19999999999999996
```

퍼셉트론 구현 - AND 게이트

0, 1 입력에 대한 AND게이트 구현결과 0이 나온다.

```
def AND(x1, x2):  
    x = np.array([x1, x2])      # 입력  
    w = np.array([0.5, 0.5])    # 가중치  
    b = -0.7                    # 편향  
    rvalue = np.sum(w*x) + b    # 연산값  
    if rvalue <= 0:  
        return 0  
    else:  
        return 1
```

AND(0,1)

0

퍼셉트론 구현 – NAND 게이트

AND와는 가중치 (w와 b)만 다르다.

```
def NAND(x1, x2):  
    x = np.array([x1, x2])          # 입력  
    w = np.array([-0.5, -0.5])      # 가중치  
    b = 0.7                          # 편향  
    rvalue = np.sum(w*x) + b        # 연산값  
    if rvalue <= 0:  
        return 0  
    else:  
        return 1
```

NAND(1,1)

0

퍼셉트론 구현 – OR 게이트

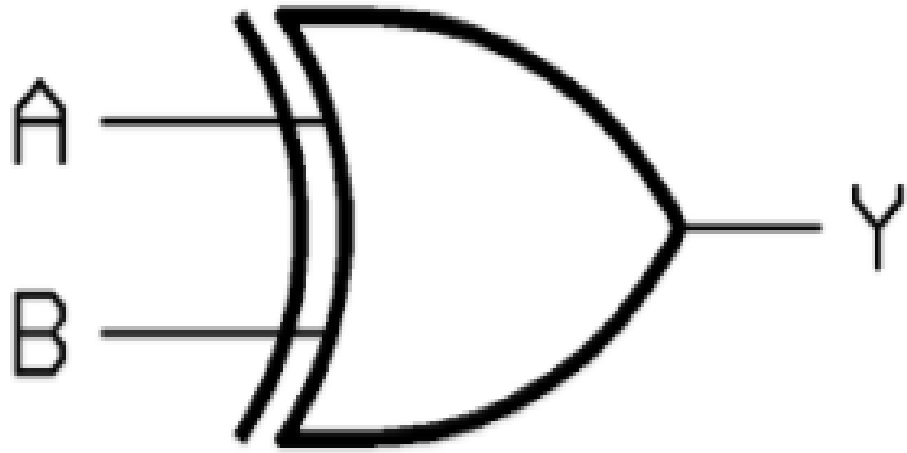
입력이 둘 중 하나라도 1이 나오면 출력이 1이 나온다.

```
def OR(x1, x2):  
    x = np.array([x1, x2])    # 입력  
    w = np.array([0.5, 0.5])  # 가중치  
    b = -0.2                  # 편향  
    rvalue = np.sum(w*x) + b  # 연산값  
    if rvalue <= 0:  
        return 0  
    else:  
        return 1  
  
print(OR(0,1))  
print(OR(0,0))
```

1
0

퍼셉트론 구현 - XOR 게이트

배타적 논리합이라는 논리 회로. x1과 x2중 한쪽이 1일 때만 1을 출력



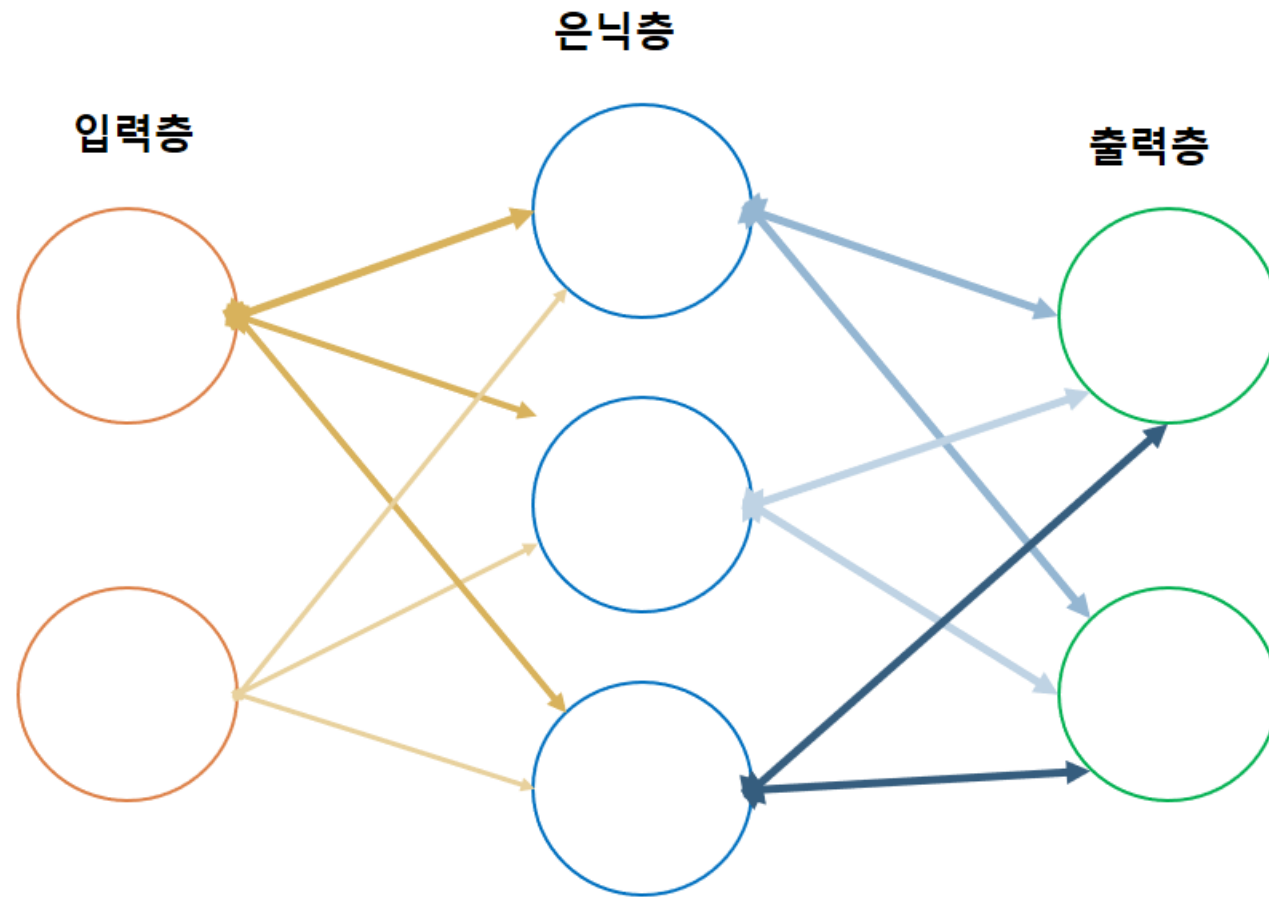
XOR truth table

Input		Output
A	B	
0	0	0
0	1	1
1	0	1
1	1	0

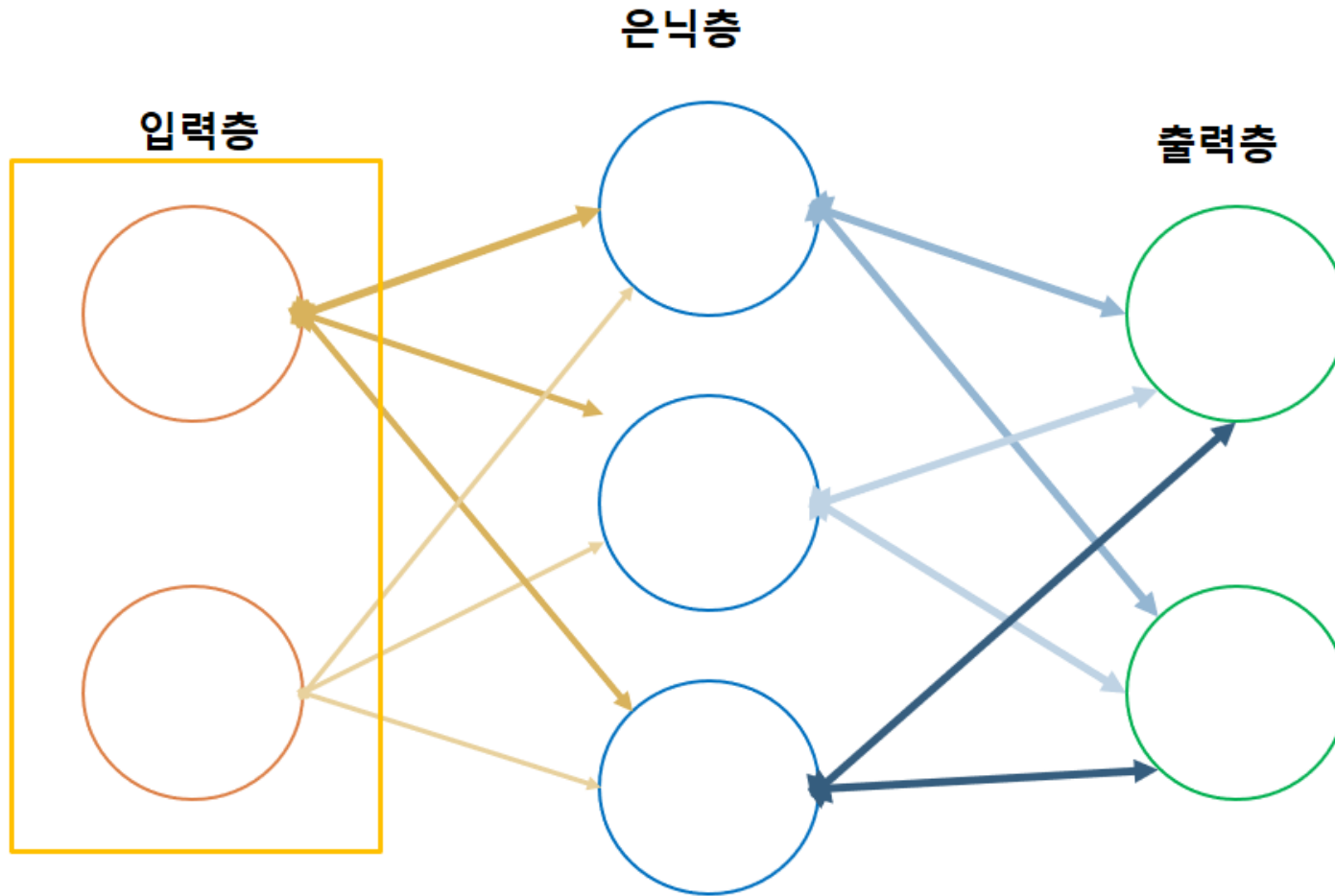
- 0, false
- 1, true

CH 3 신경망

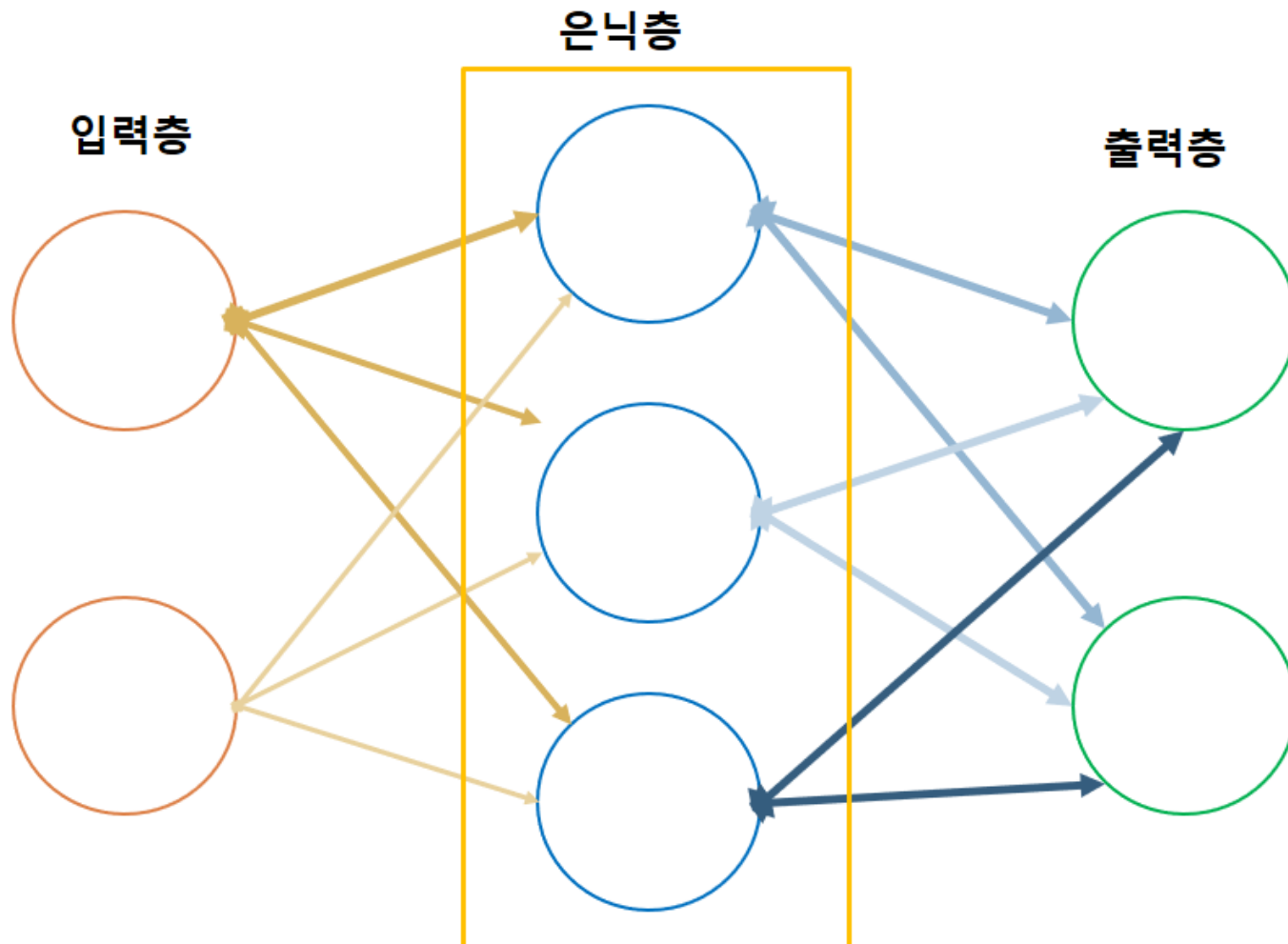
신경망이란?



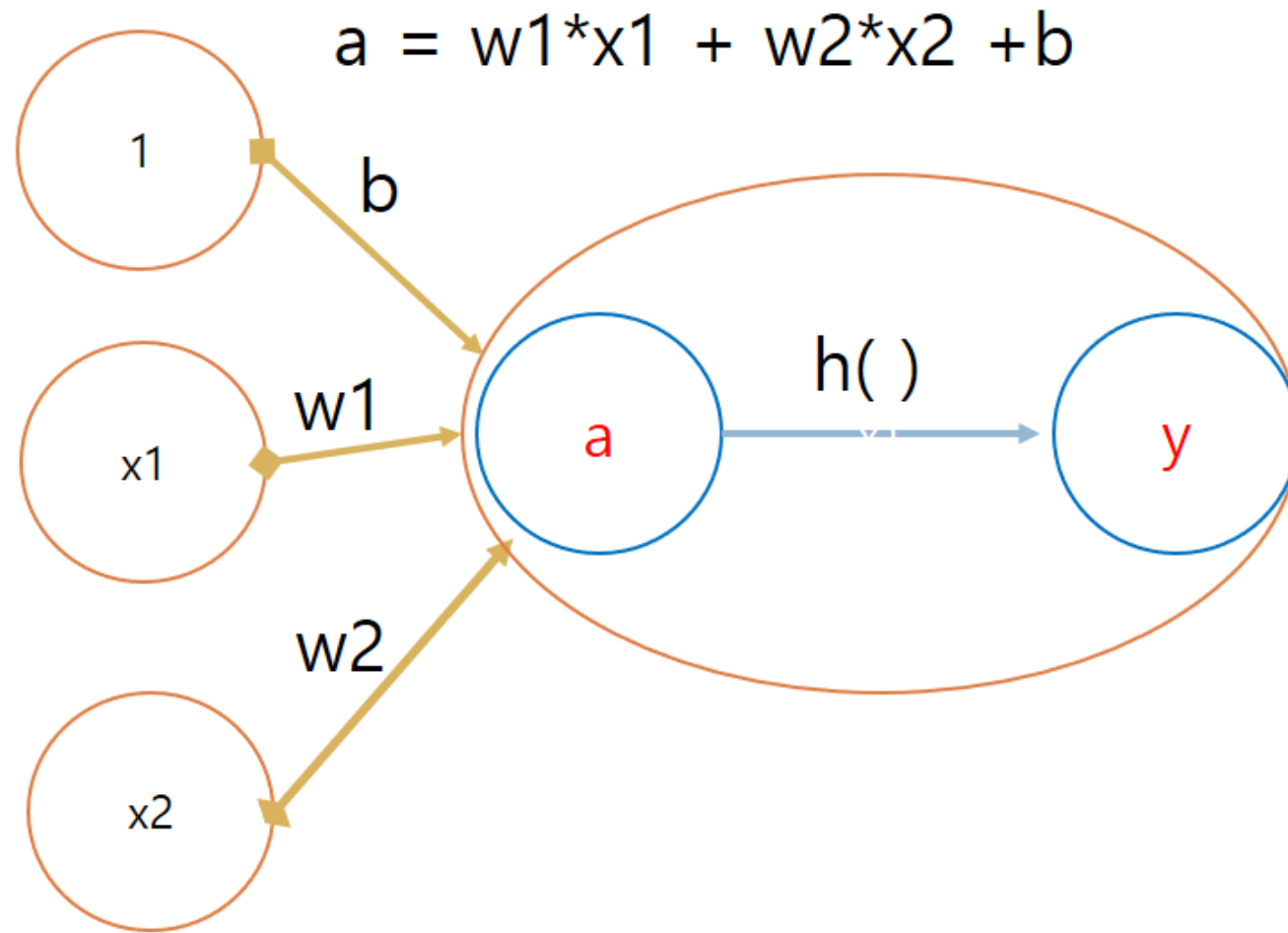
신경망 - 입력층



신경망 - 은닉층



활성화 함수의 등장



$h()$: 활성화 함수

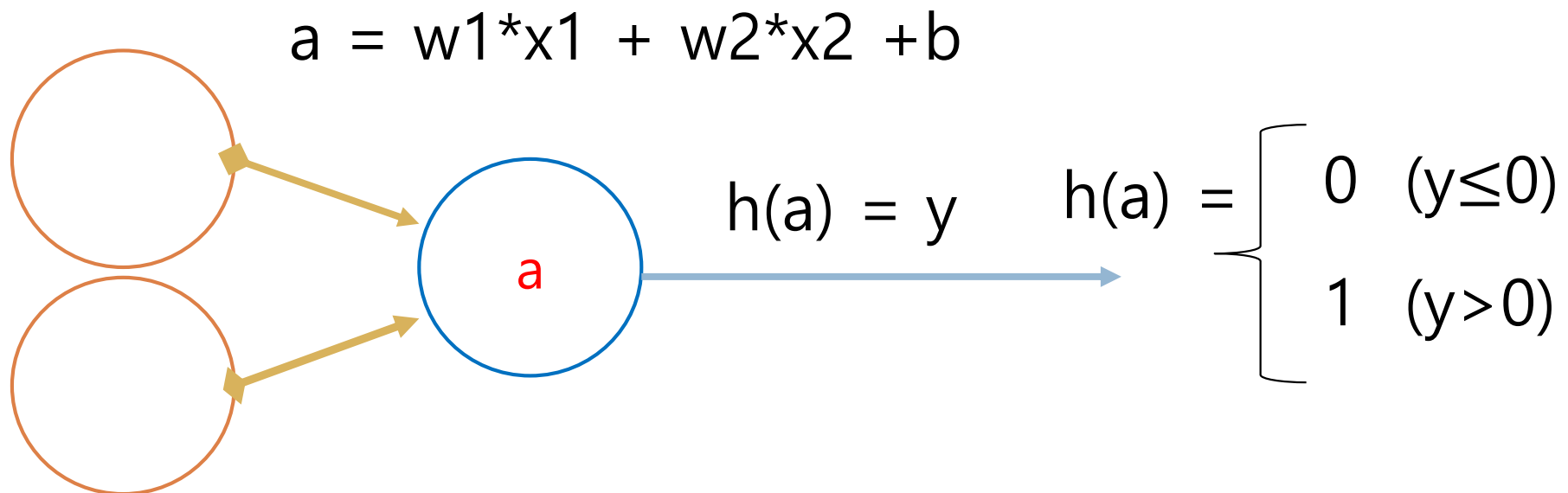
활성화 함수

$$h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

신경망이란?

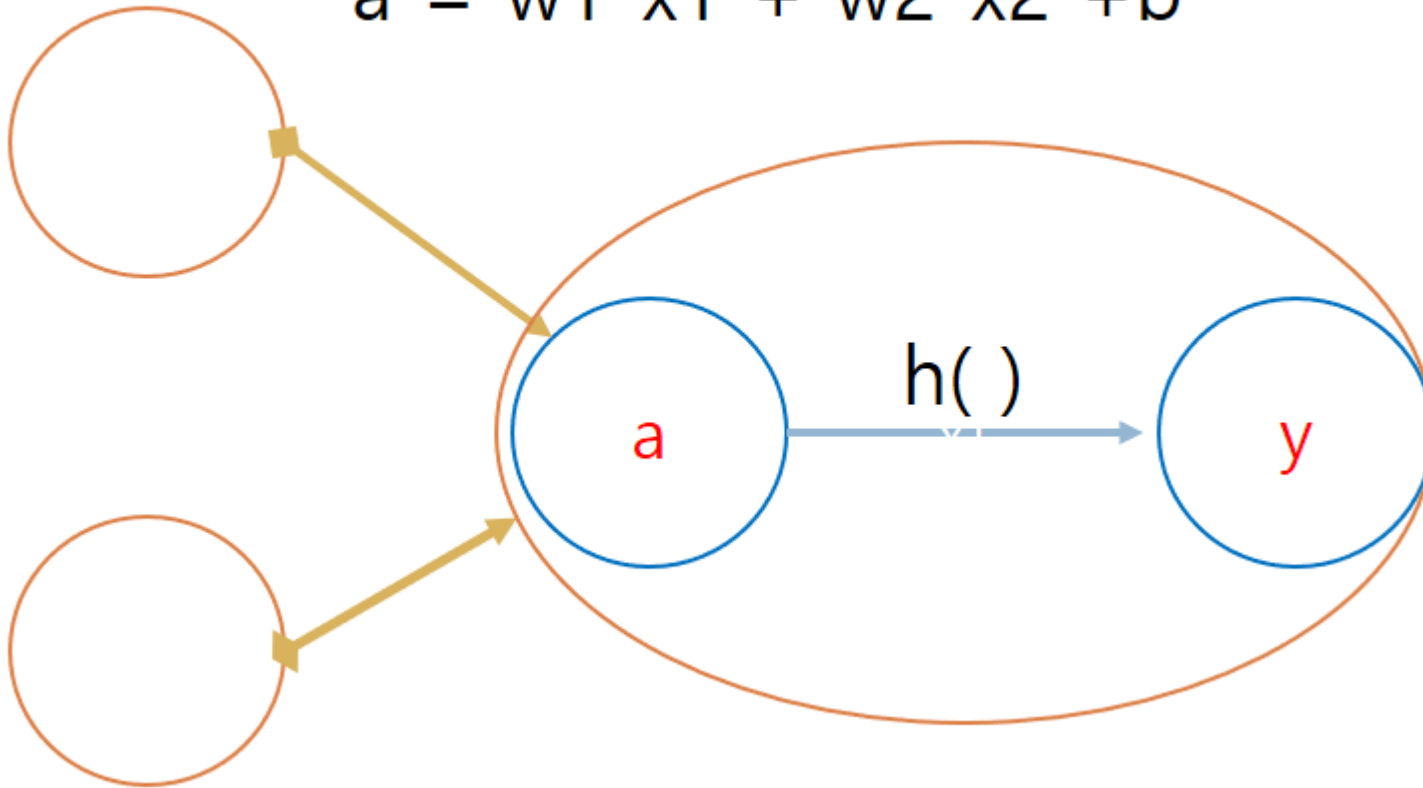
퍼셉트론을 수식으로 표현

$$y = \begin{cases} 0 & (w_1 * x_1 + w_2 * x_2 + b \leq 0) \\ 1 & (w_1 * x_1 + w_2 * x_2 + b > 0) \end{cases}$$



신경망이란?

$$a = w1*x1 + w2*x2 + b$$



퍼셉트론 & 신경망

단순 퍼셉트론은 단층 네트워크에서 계단 함수를 활성화 함수로 사용한 모델을 가르킨다.

다중 퍼셉트론은 신경망(여러 층으로 구성 시그모이드 함수 등의 활성화 함수 사용)을 가르킨다.

활성화 함수- 시그모이드 함수

$$h(x) = \frac{1}{1+\exp(-x)}$$

$\exp(-x)$ 는 e^{-x} 를 뜻한다.

e 는 자연상수로 2.7182의 값을 갖는 실수이다.

$h(1.0) = 0.731$, $h(2.0)=0.880$

활성화 함수- 계단함수

```
def step_function(x):  
    if x > 0:  
        return 1  
    else:  
        return 0  
  
print(step_function(1))  
print(step_function(-5))
```

1
0

활성화 함수- 계단함수

```
In [2]: import numpy as np  
x = np.array([-1.0, 1.0, 2.0])  
x
```

x에 numpy 배열을 넣기

```
Out[2]: array([-1.,  1.,  2.])
```

```
In [4]: y=x>0  
y
```

x의 값을 x>0 연산 후,

```
Out[4]: array([False,  True,  True], dtype=bool)
```

y의 값을 출력

```
In [6]: y=y.astype(np.int)  
y
```

bool의 값에서 int형으로 변경하기

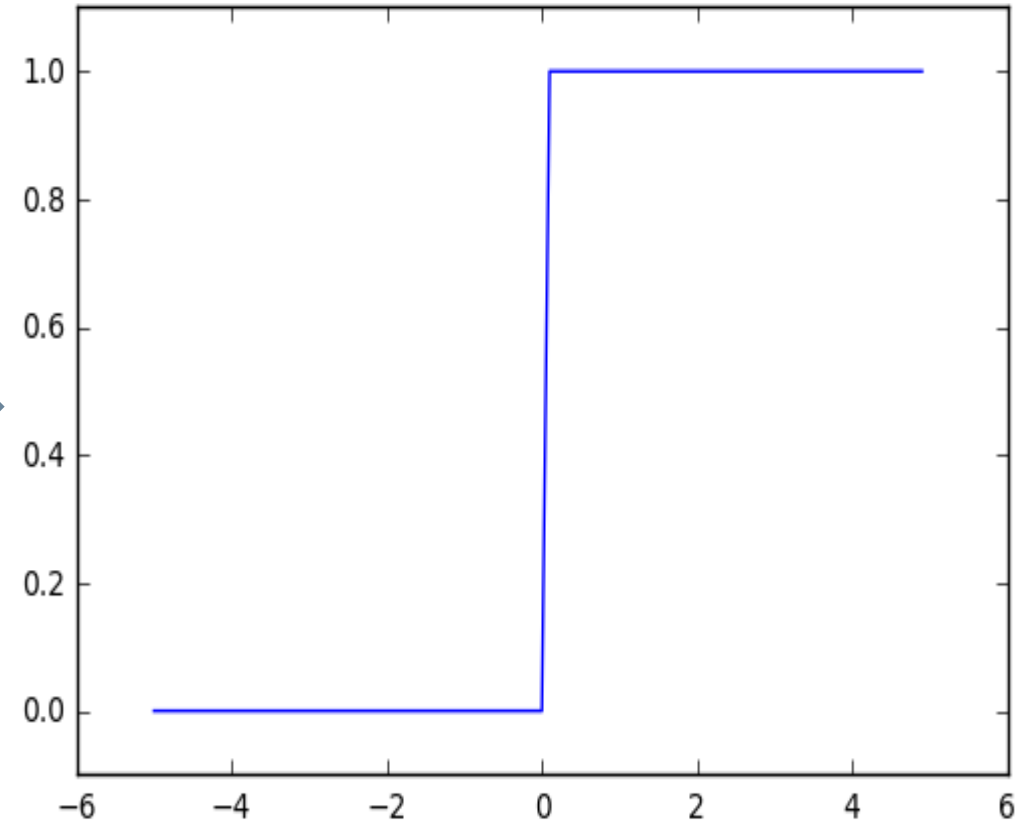
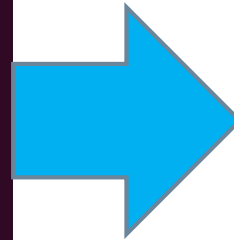
```
Out[6]: array([0, 1, 1])
```

활성화 함수- 계단함수

```
import numpy as np
import matplotlib.pyplot as plt

def step_function(x):
    return np.array(x > 0, dtype=np.int)

x = np.arange(-5.0, 5.0, 0.1)
y = step_function(x)
plt.plot(x,y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
plt.show()
```



활성화 함수- 시그모이드 함수

시그모이드 함수 구현

```
import numpy as np
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

x = np.array([-1.0, 1.0, 2.0])
sigmoid(x)
```

출력 결과

```
array([ 0.26894142,  0.73105858,  0.88079708])
```

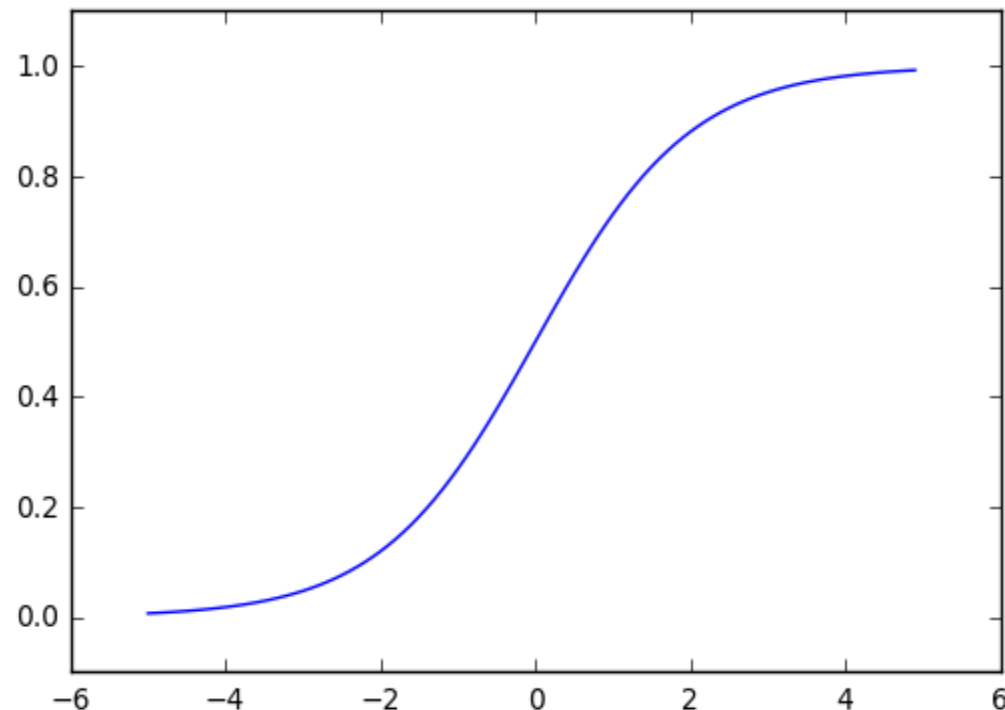
활성화 함수- 시그모이드 함수

시그모이드 함수

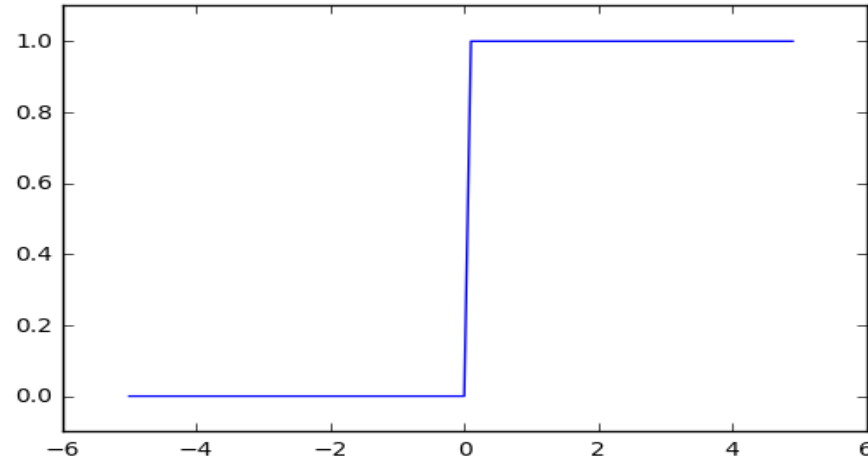
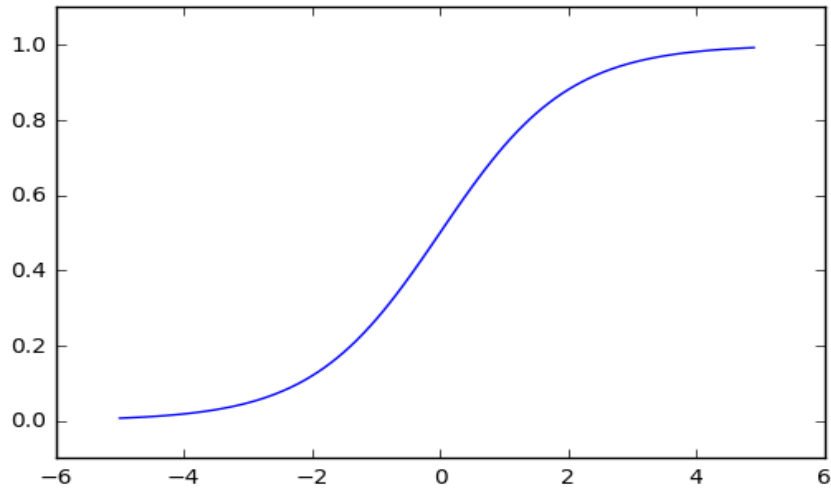
```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(x):
    return 1/ (1 + np.exp(-x))

x=np.arange(-5.0, 5.0, 0.1)
y=sigmoid(x)
plt.plot(x,y)
plt.ylim(-0.1, 1.1)
plt.show()
```

출력 결과

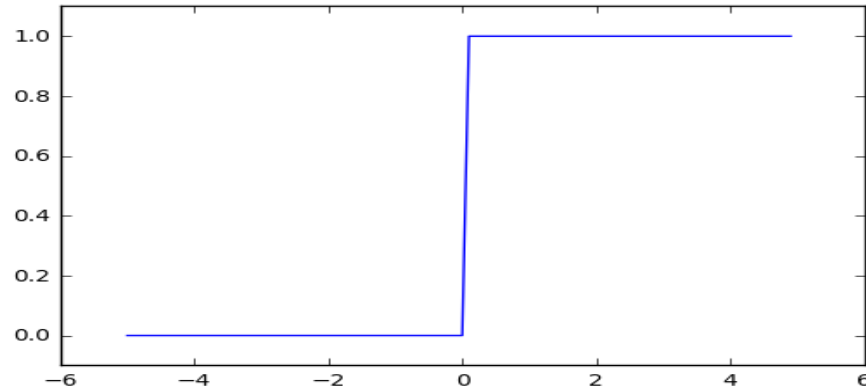
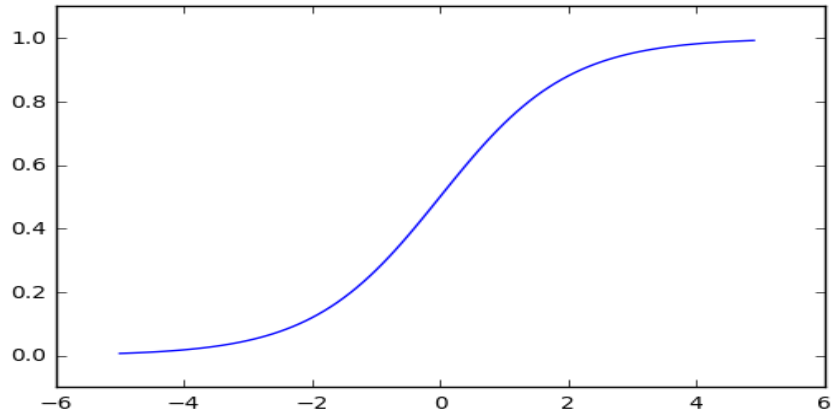


시그모이드 함수 vs 계단함수



- (1) 시그모이드 함수는 부드러운 곡선, 입력에 따라 출력이 연속적으로 변화
- (2) 계단 함수가 0과 1중의 하나의 값을 돌려준다면 시그모이드 함수는 (0.731, ..., 0.880,...)을 돌려준다는 것도 다르다.

비선형 함수



(1) 신경망에서는 활성화 함수로 비선형 함수를 사용해야 한다.
선형 함수를 사용한다면 신경망의 층을 깊게 하는 의미가 없음.

(예) $h(x) = c(x)$ 를 활성화 함수를 적용

$$y(x) = h(h(h(x)))$$

$$y(x) = c * c * c * x, y(x) = ax \text{와 같다.}$$

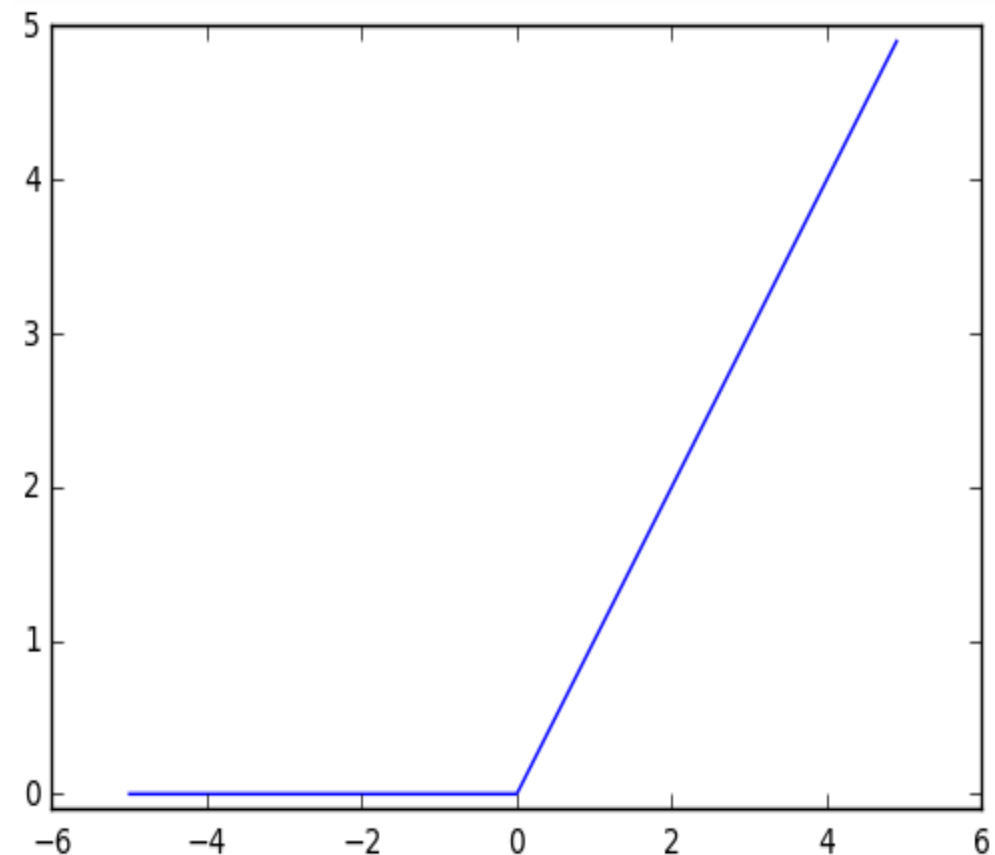
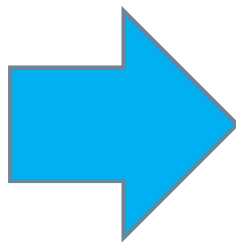
활성화 함수- ReLU 함수

최근에 많이 사용되는 활성화 함수

```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
    return np.maximum(0,x)

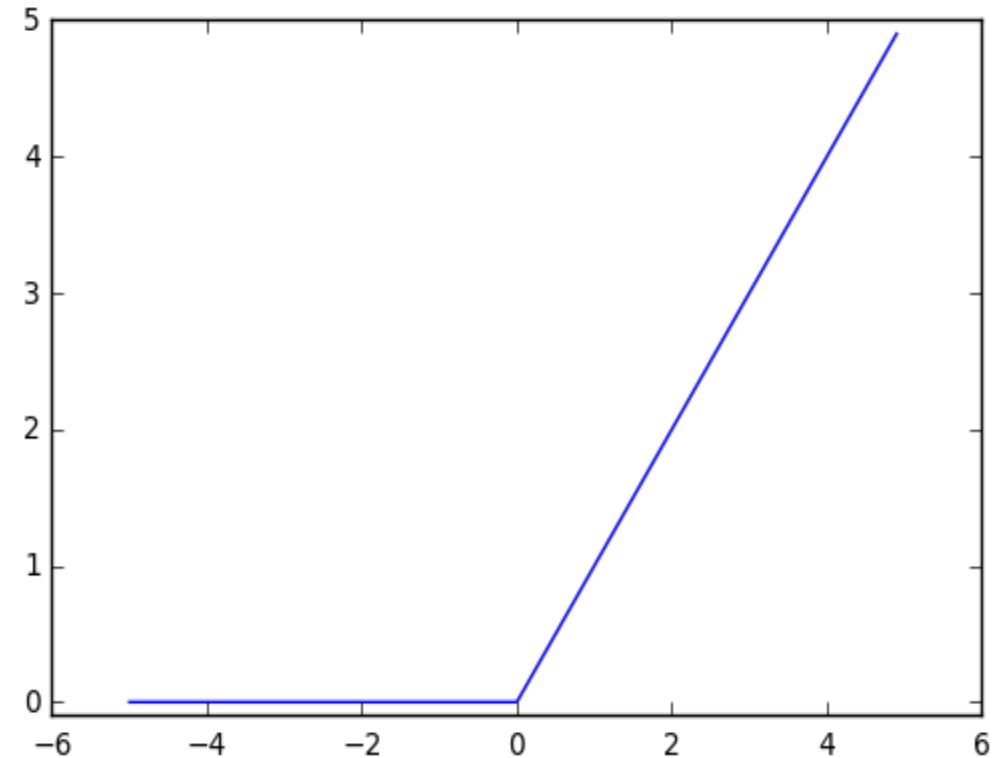
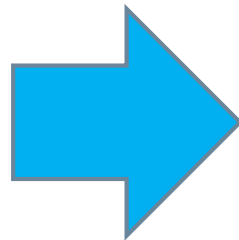
x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x,y)
plt.ylim(-0.1, 5)    # y축의 범위 지정
plt.show()
```



활성화 함수- ReLU 함수

최근에 많이 사용되는 활성화 함수

$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$



다차원 배열의 연산

```
import numpy as np
A = np.array([1,2,3,4])
print(A)
```

[1 2 3 4]

```
np.ndim(A)
```

배열의 차원수 확인

1

```
A.shape
```

배열의 형상

(4,)

```
A.shape[0]
```

4

```
import numpy as np
B = np.array([[1,2],[3,4],[5,6]])
print(B)
```

[[1 2]
 [3 4]
 [5 6]]

```
np.ndim(B)
```

배열의 차원수 확인

2

```
B.shape
```

배열의 형상

(3, 2)

행렬의 내적(곱)

```
A = np.array([[1,2],[3,4]])  
A.shape
```

```
(2, 2)
```

```
B = np.array([[5,6],[7,8]])  
B.shape
```

```
(2, 2)
```

```
np.dot(A,B)
```

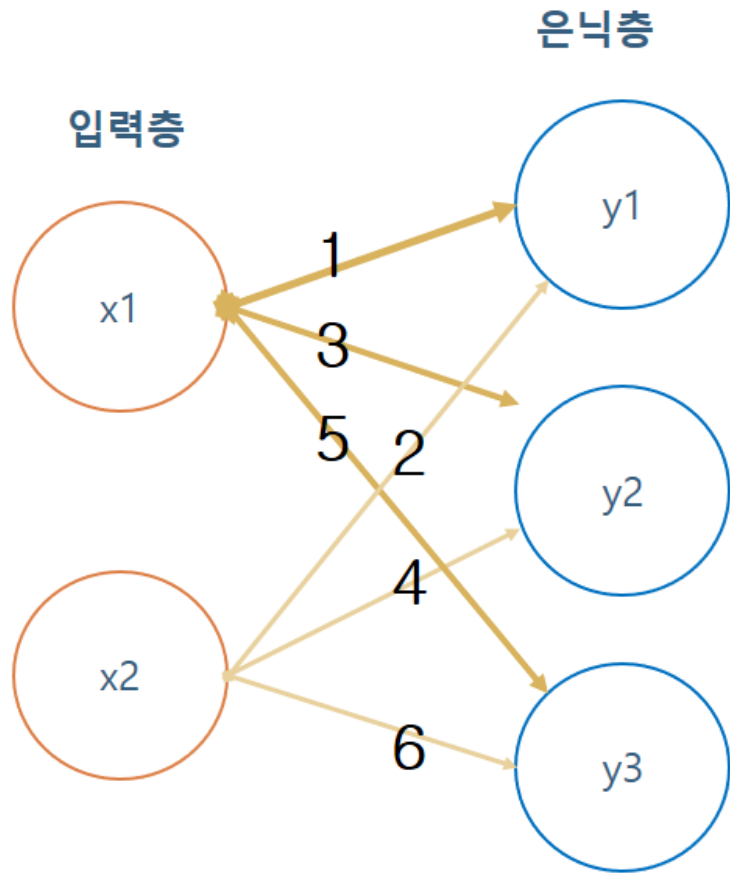
```
array([[19, 22],  
       [43, 50]])
```

행렬의 내적(곱) 주의

행렬의 곱에서는 대응하는 차원의 원소 수를 일치 시켜야 함.

$$\begin{array}{ccc} A & B & = C \\ 3 \times \boxed{2} & \boxed{2} \times 4 & 3 \times 4 \end{array}$$

신경망의 내적



$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$



$$\begin{matrix} X & W & = & Y \\ 2 & 2 \times 3 & & 3 \end{matrix}$$

신경망의 내적

```
import numpy as np
X = np.array([1,2])
X.shape
```

(2,)

```
W = np.array([[1,3,5], [2,4,6]])
print(W)
```

```
[[1 3 5]
 [2 4 6]]
```

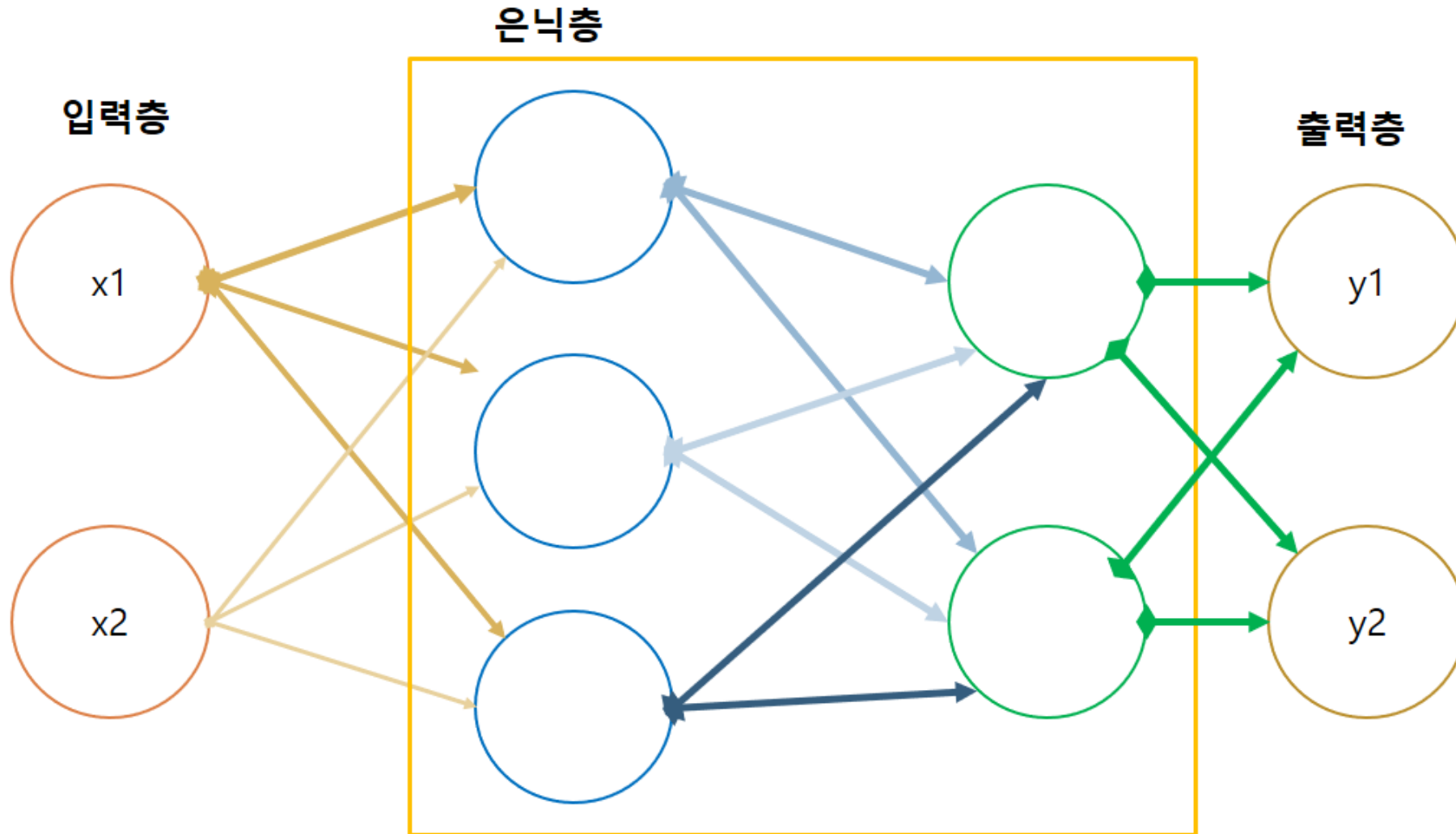
```
W.shape
```

(2, 3)

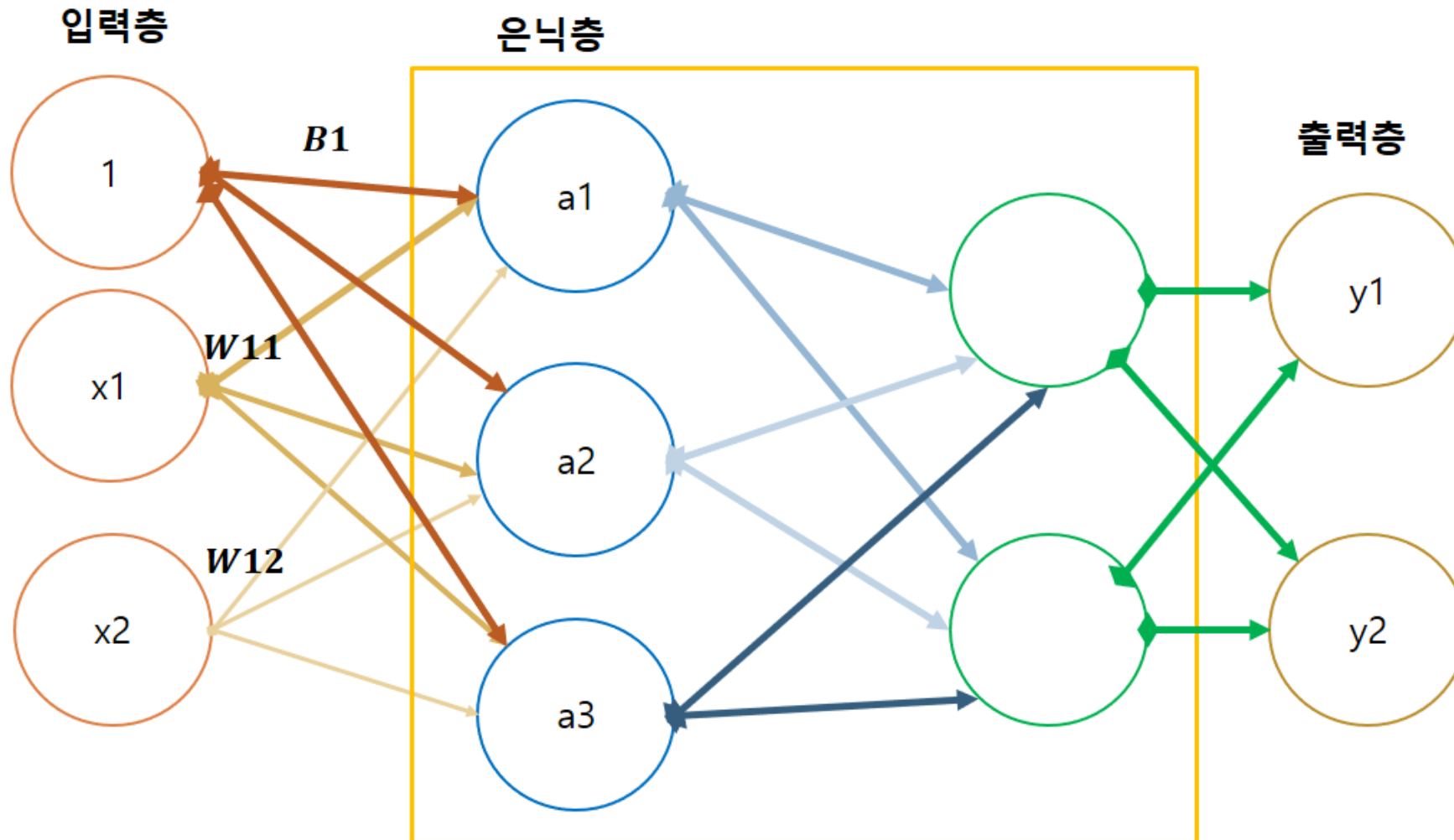
```
Y=np.dot(X,W)
print(Y)
```

```
[ 5 11 17]
```

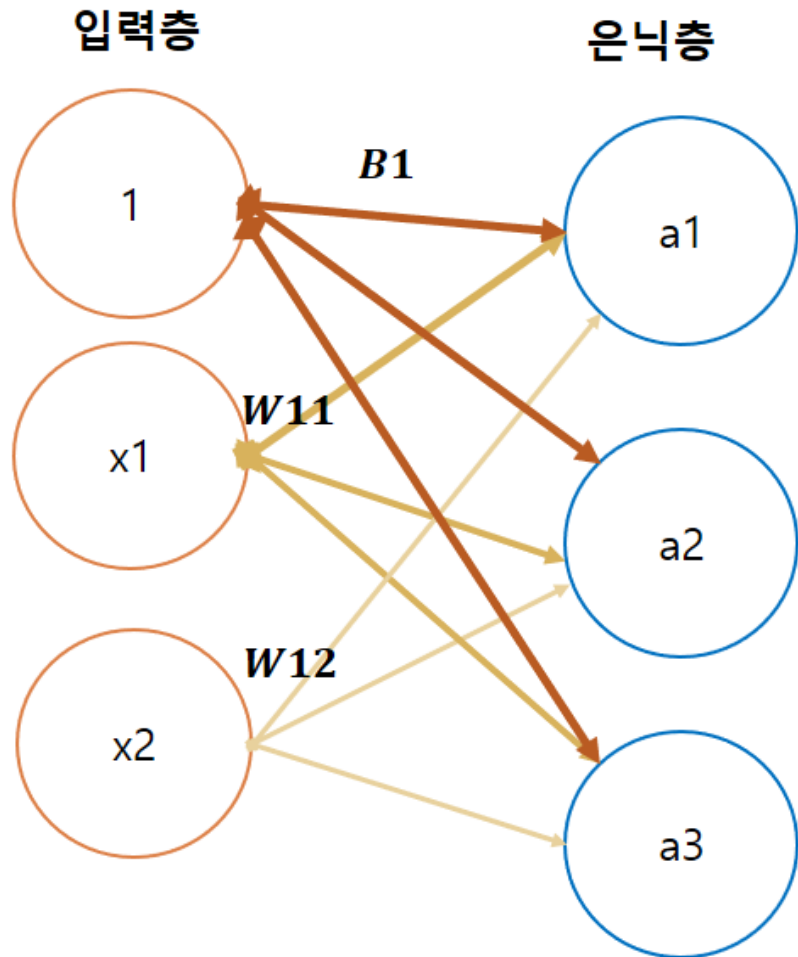
3층 신경망 구현하기



3층 신경망 구현하기 – 입력층에서 1층 신호전달



3층 신경망 구현하기 - 입력층에서 1층 신호전달



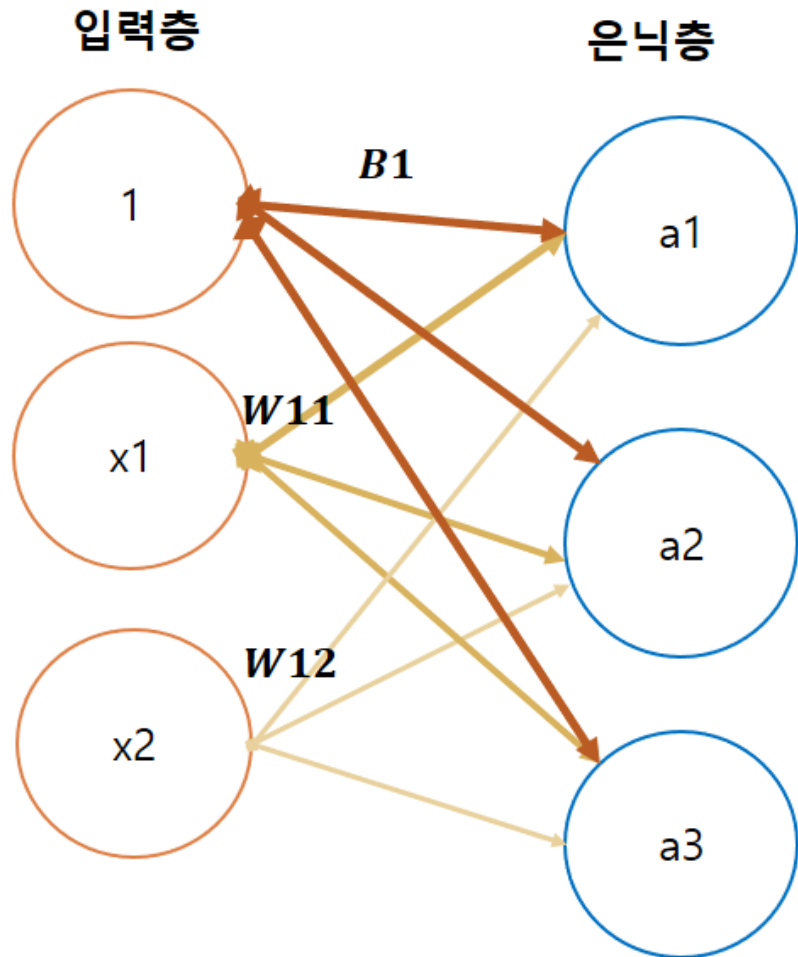
$$a1 = w11*x1 + w12*x2 + b1$$

$$a2 = w21*x1 + w22*x2 + b2$$

$$a3 = w31*x1 + w32*x2 + b3$$

$$\mathbf{A} = \mathbf{XW} + \mathbf{B}$$

3층 신경망 구현하기 - 입력층에서 1층 신호전달



$$\mathbf{A} = \mathbf{XW} + \mathbf{B}$$

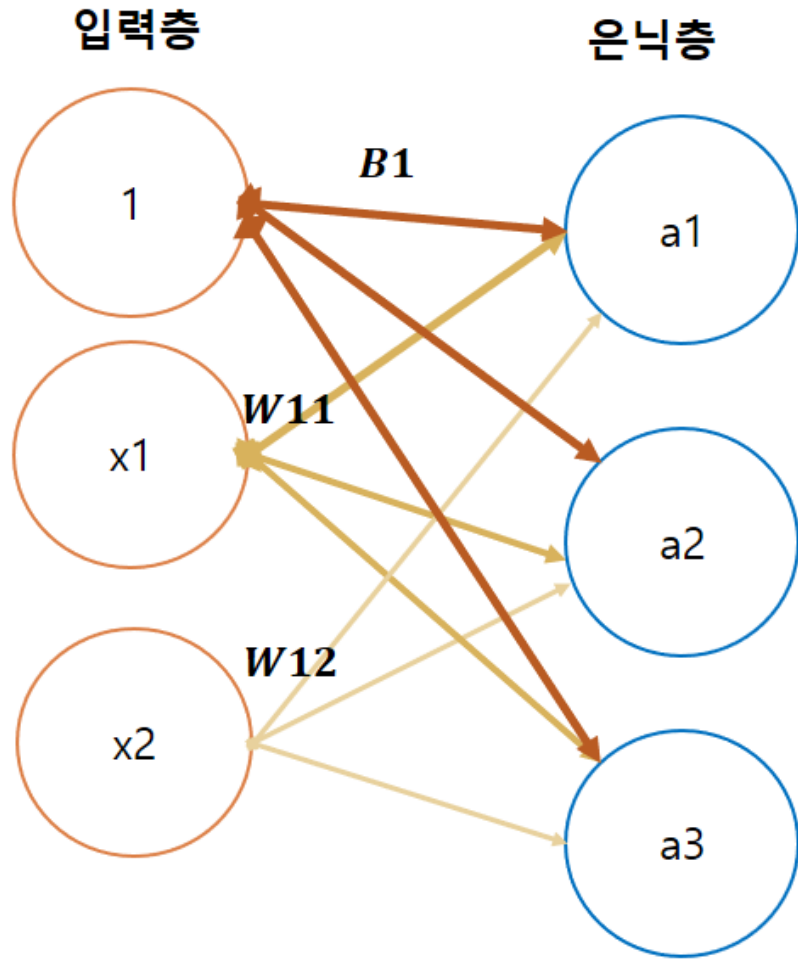
$$\mathbf{A} = \begin{bmatrix} a1 & a2 & a3 \end{bmatrix}$$

$$\mathbf{X} = \begin{bmatrix} x1 & x2 \end{bmatrix}$$

$$\mathbf{B} = \begin{bmatrix} b1 & b2 & b3 \end{bmatrix}$$

$$\mathbf{W} = \begin{bmatrix} w11 & w21 & w31 \\ w12 & w22 & w32 \end{bmatrix}$$

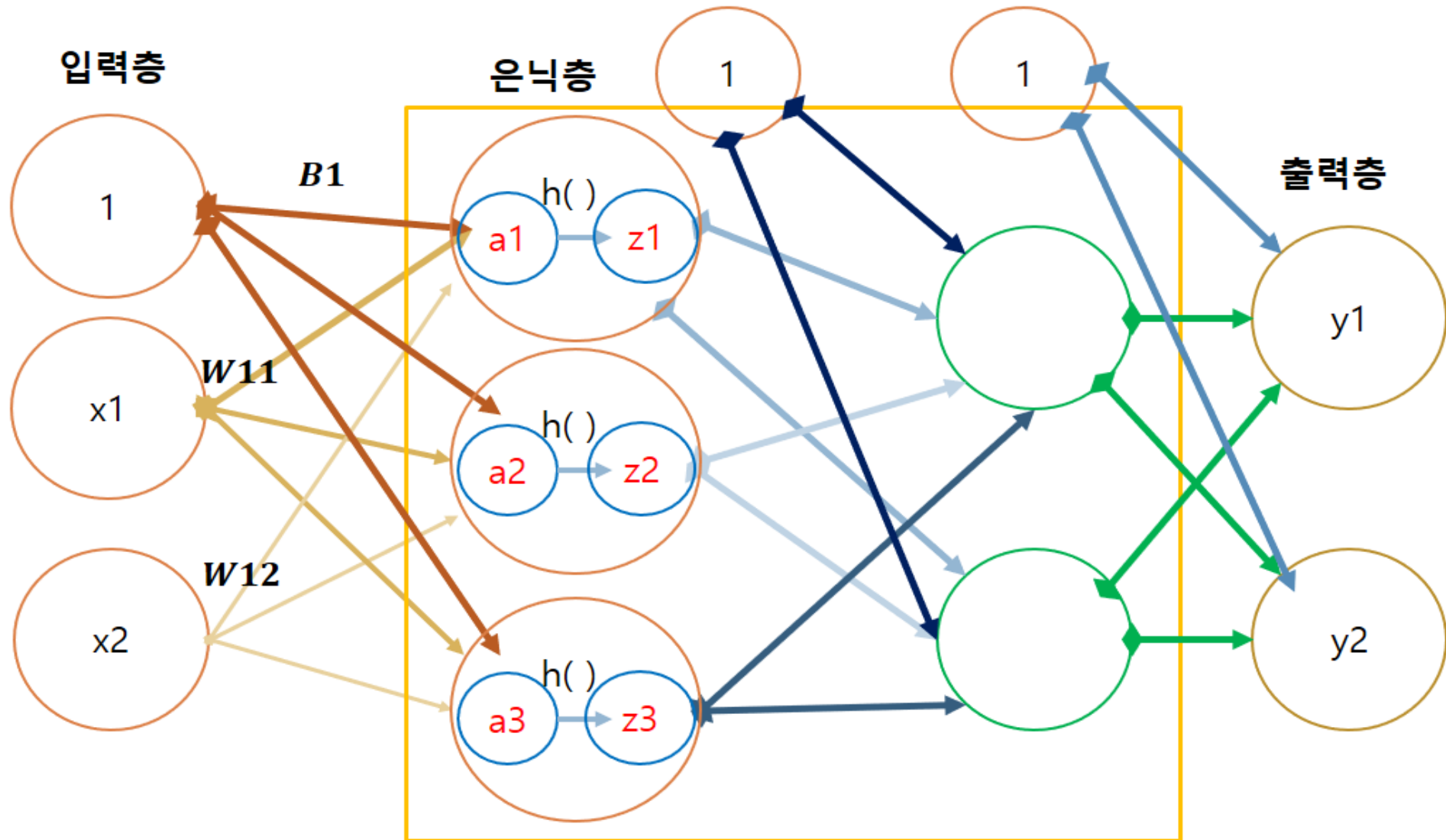
3층 신경망 구현하기 - 입력층에서 1층 신호전달



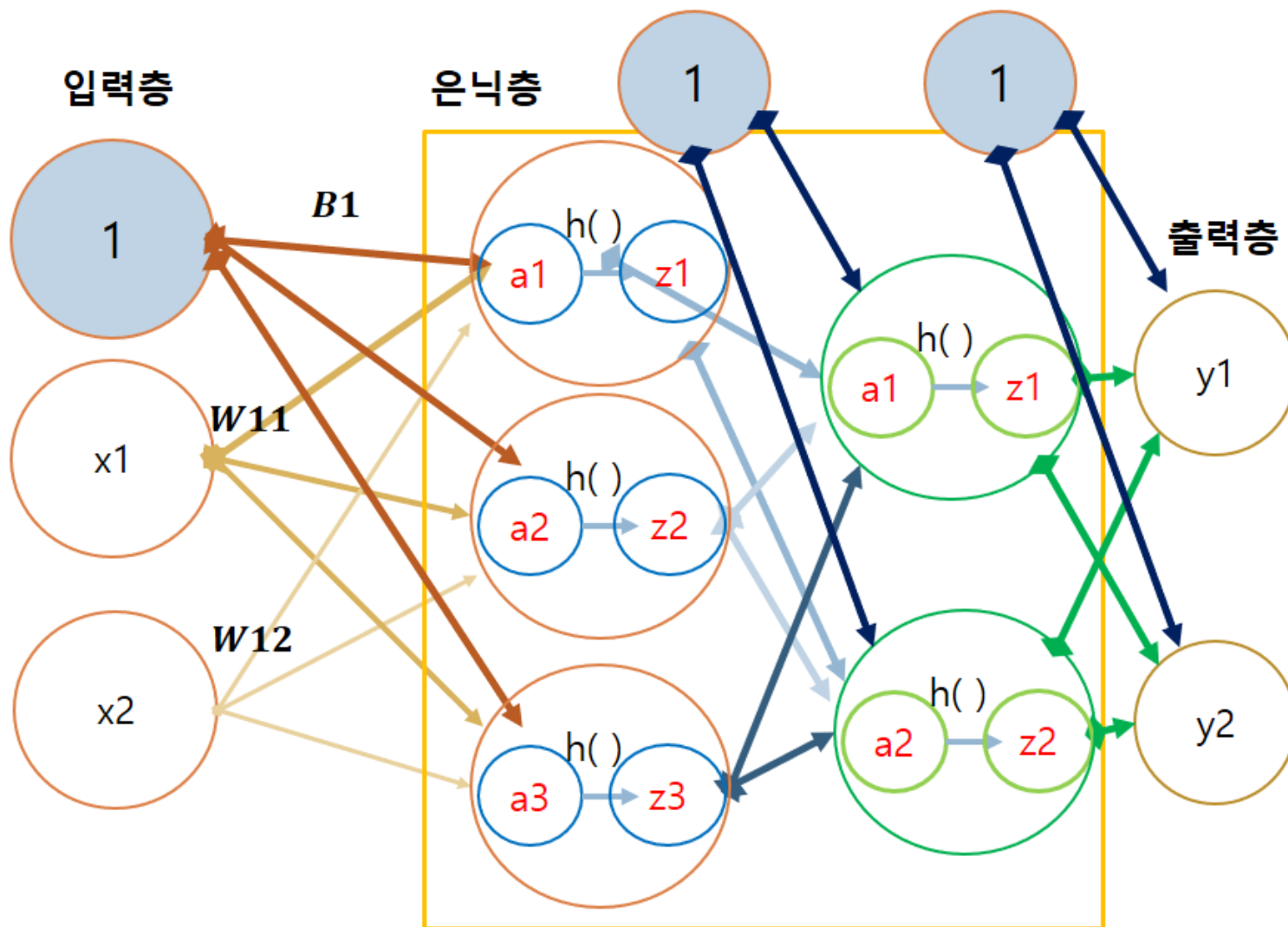
```
X=np.array([1.0, 0.5])
W1=np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
B1=np.array([0.1, 0.2, 0.3])
print(W1.shape)
print(X.shape)
print(B1.shape)
A1 = np.dot(X, W1) + B1
print("A1 : ", A1)
```

```
(2, 3)
(2,)
(3,)
A1 : [ 0.3  0.7  1.1]
```

3층 신경망 구현하기 - 1층에서 2층 신호전달



3층 신경망 구현하기 - 2층에서 3층 신호전달



```
import numpy as np
import matplotlib.pyplot as plt
def sigmoid(x):
    return 1 / (1 + np.exp(-x))

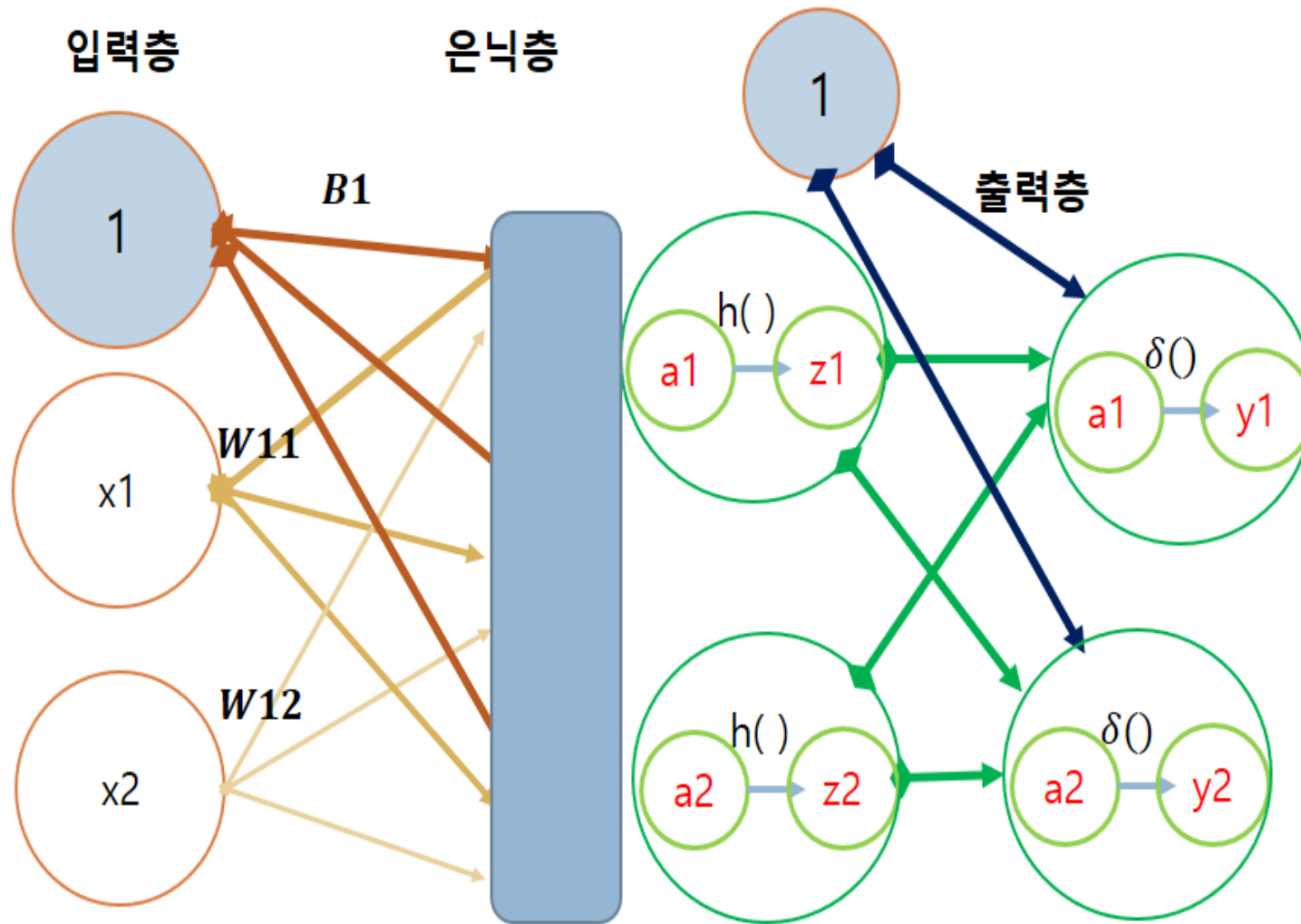
A1 = np.dot(X, W1) + B1
Z1 = sigmoid(A1)
W2=np.array([[0.1,0.4], [0.2, 0.5], [0.3, 0.6]])
B2=np.array([0.1, 0.2])

print(Z1.shape) # (3,)
print(W2.shape) # (3,2)
print(B2.shape) # (2,)

A2 = np.dot(Z1, W2) + B2
Z2 = sigmoid(A2)
print("A2 => ", A2)
print("Z2 => ", Z2)

(3,)
(3, 2)
(2,)
A2 => [ 0.51615984  1.21402696]
Z2 => [ 0.62624937  0.7710107 ]
```

3층 신경망 구현하기 - 2층에서 3층 신호전달



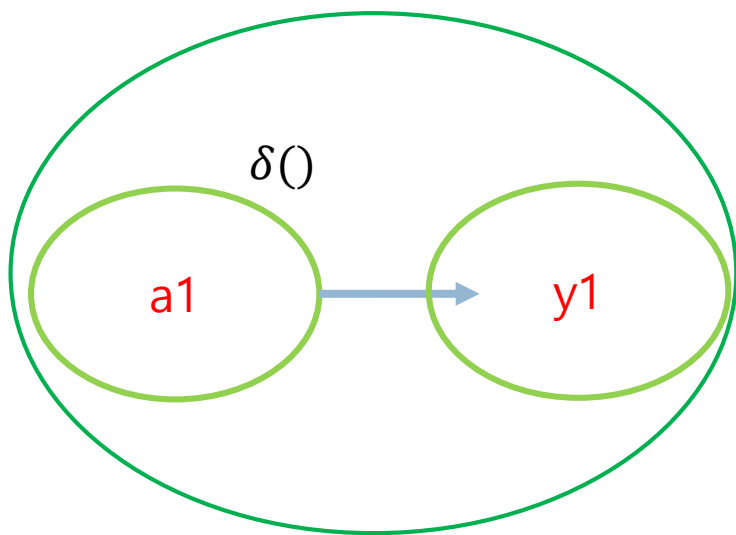
```
import numpy as np
import matplotlib.pyplot as plt
def identity_function(x):
    return x

W3 = np.array([[0.1, 0.3], [0.2, 0.4]])
B3 = np.array([0.1, 0.2])

A3 = np.dot(Z2, W3) + B3
Y = identity_function(A3)
print("W3 =>", Z2)
print("Z2 =>", Z2)
print("A3(Z2 * W3) =>", A3)
print("Y =>", Y)
```

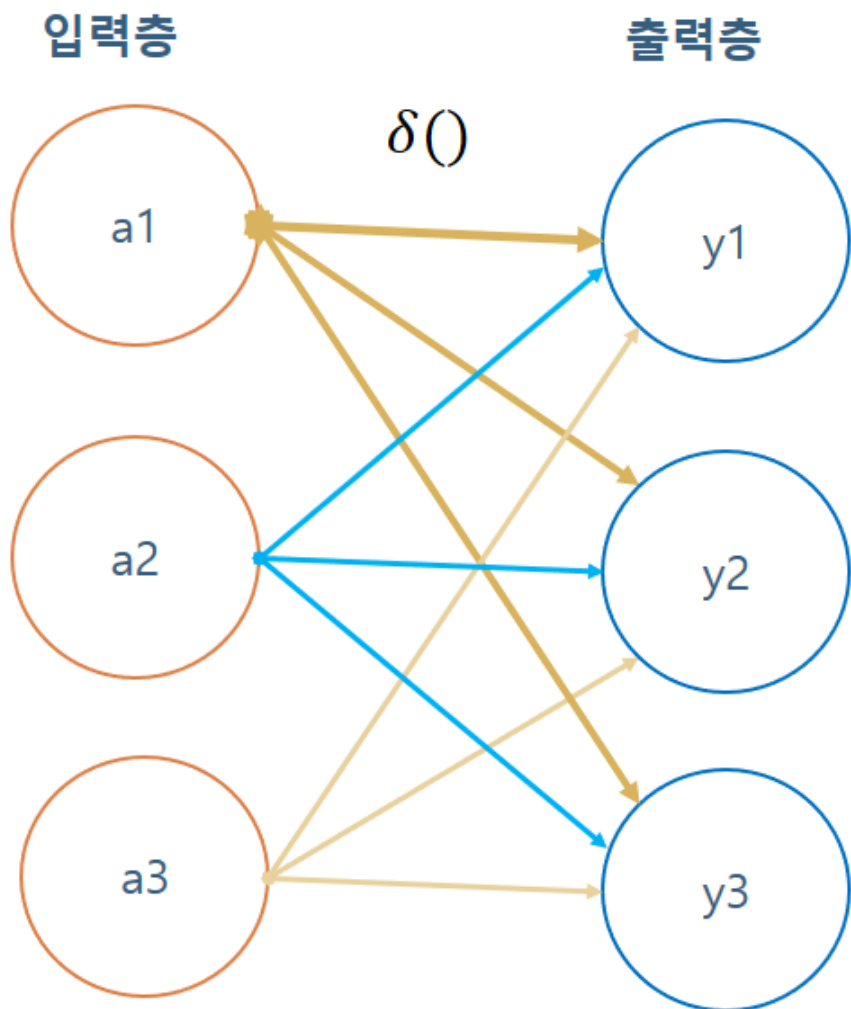
W3 => [0.62624937 0.7710107]
Z2 => [0.62624937 0.7710107]
A3(Z2 * W3) => [0.31682708 0.69627909]
Y => [0.31682708 0.69627909]

3층 신경망 구현하기 - 항등함수



신경망은 분류와 회귀 모두에 이용이 가능하다.

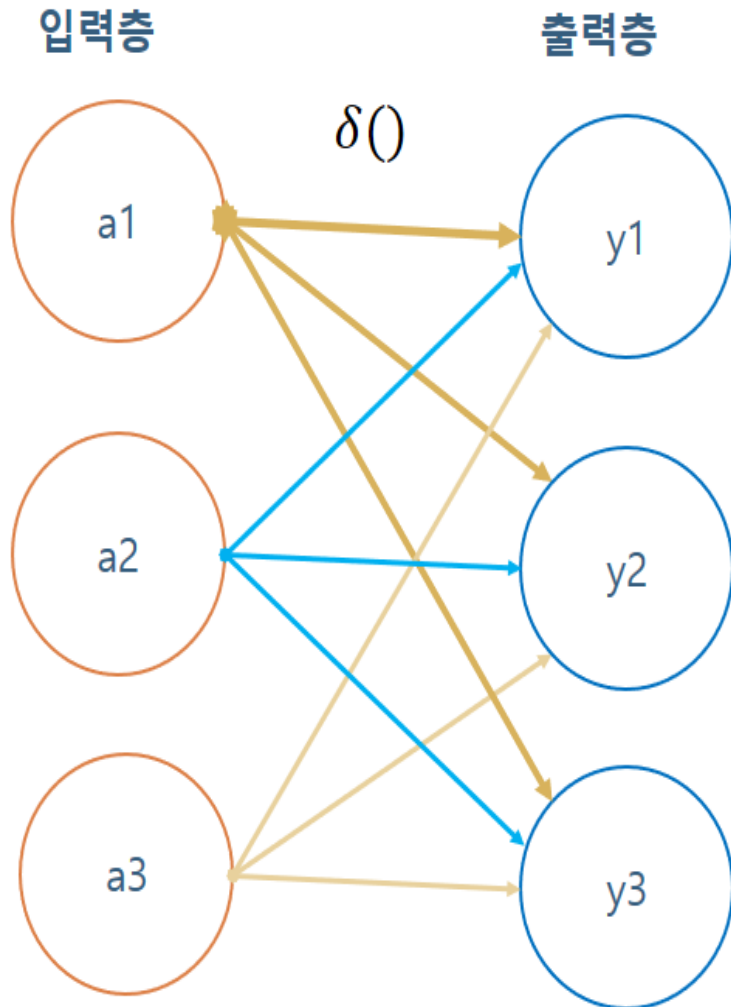
3층 신경망 구현하기 - 소프트맥스 함수



n : 출력층의 뉴런수
 y_k : k번째 출력
 a_k : 입력 신호

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$

3층 신경망 구현하기 - 소프트맥스 함수



n : 출력층의 뉴런수
 y_k : k 번째 출력
 a_k : 입력 신호

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$

```
a = np.array([0.3, 2.9, 4.0])
exp_a = np.exp(a)
print(exp_a)

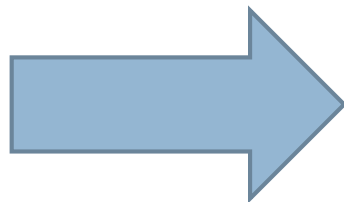
sum_exp_a = np.sum(exp_a) # 지수 함수의 합
print(sum_exp_a)

y = exp_a / sum_exp_a
print(y)
```

```
[ 1.34985881 18.17414537 54.59815003]
74.1221542102
[ 0.01821127 0.24519181 0.73659691]
```

소프트맥스 함수

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$



$$y_k = \frac{\exp(a_k + C')}{\sum_{i=1}^n \exp(a_i + C')}$$

소프트맥스 함수는 지수함수를 사용한다.

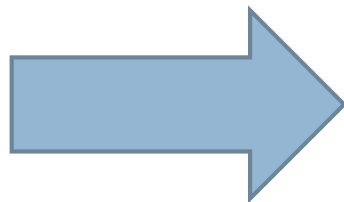
지수는 매우 큰 값을 갖는다. 이때 컴퓨터는 큰 값에 대해 오버플로 문제 발생

따라서 큰 값끼리 나눗셈을 하게 되면 결과 수치가 불안정해진다.

$$y_k = \frac{C \exp(a_k)}{C \sum_{i=1}^n \exp(a_i)} = \frac{\exp(a_k + \log C)}{\sum_{i=1}^n \exp(a_i + \log C)} = \frac{\exp(a_k + C')}{\sum_{i=1}^n \exp(a_i + C')}$$

소프트맥스 함수

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$



$$y_k = \frac{\exp(a_k + C')}{\sum_{i=1}^n \exp(a_i + C')}$$

```
a = np.array([1010, 1000, 990])
exp_a = np.exp(a)
print(exp_a)

sum_exp_a = np.sum(exp_a) # 지수 함수의 합
print(sum_exp_a)

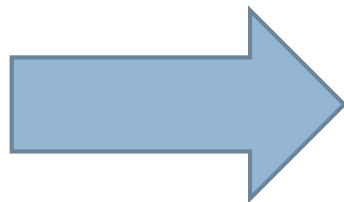
y = exp_a / sum_exp_a
print(y)
```

```
[ inf  inf  inf]
inf
[ nan  nan  nan]
```

```
C:\Users\front\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: RuntimeWarning: overflow encountered in exp
  from ipykernel import kernelapp as app
C:\Users\front\Anaconda3\lib\site-packages\ipykernel\__main__.py:8: RuntimeWarning: invalid value encountered in true_divide
```

소프트맥스 함수

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^n \exp(a_i)}$$



$$y_k = \frac{\exp(a_k + C')}{\sum_{i=1}^n \exp(a_i + C')}$$

```
a = np.array([1010, 1000, 990])
exp_a = np.exp(a)
print(exp_a)

sum_exp_a = np.sum(exp_a) # 지수 함수의 합
print(sum_exp_a)

y = exp_a / sum_exp_a
print(y)

c=np.max(a)
a-c
np.exp(a-c) / np.sum(np.exp(a-c))
```

```
[ inf  inf  inf]
inf
[ nan  nan  nan]
```

```
C:\Users\front\Anaconda3\lib\site-packages\ipykernel\__main__.py:2: RuntimeWarning: overflow encountered in exp
  from ipykernel import kernelapp as app
```

```
C:\Users\front\Anaconda3\lib\site-packages\ipykernel\__main__.py:8: RuntimeWarning: invalid value encountered in true_divide
```

```
array([ 9.99954600e-01,  4.53978686e-05,  2.06106005e-09])
```

소프트맥스 함수

```
def softmax(a):  
    c = np.max(a)  
    exp_a = np.exp(a-c) # 오버플로 대책  
    sum_exp_a = np.sum(exp_a)  
    y = exp_a / sum_exp_a  
  
    return y
```

```
a = np.array([0.3, 2.9, 4.0])  
y = softmax(a)  
print(y)  
np.sum(y)|
```

```
[ 0.01821127  0.24519181  0.73659691]
```

```
1.0
```