

목차 2

- 2. 퍼셉트론
- 3. 신경망
- 4. 신경망 학습
- 5. 오차역전파법
- 6. 학습 관련 기술들
- 7. 합성곱 신경망(CNN)

2. 퍼셉트론

2.1 퍼셉트론이란?

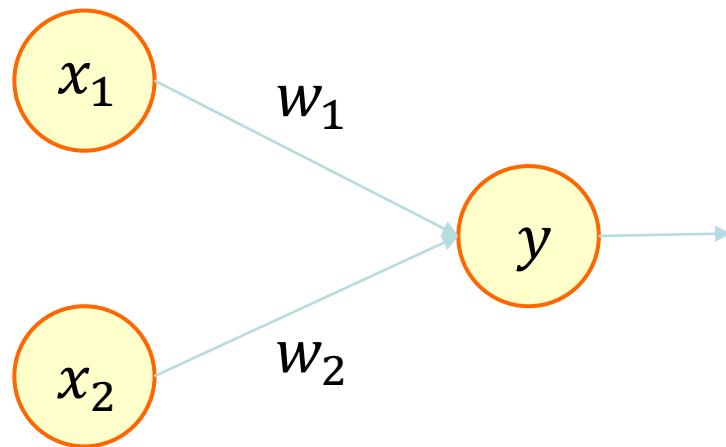
2.2 단순한 논리 회로

2.3 퍼셉트론 구현하기

2.4 퍼셉트론의 한계

2.5 다층 퍼셉트론

2.6 NAND에서 컴퓨터 까지



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

2. 퍼셉트론

2.1 퍼셉트론이란?

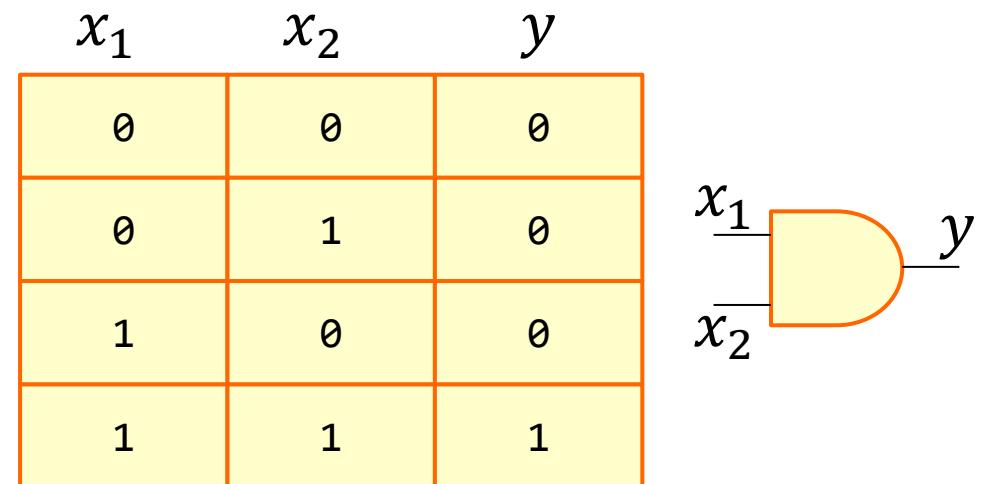
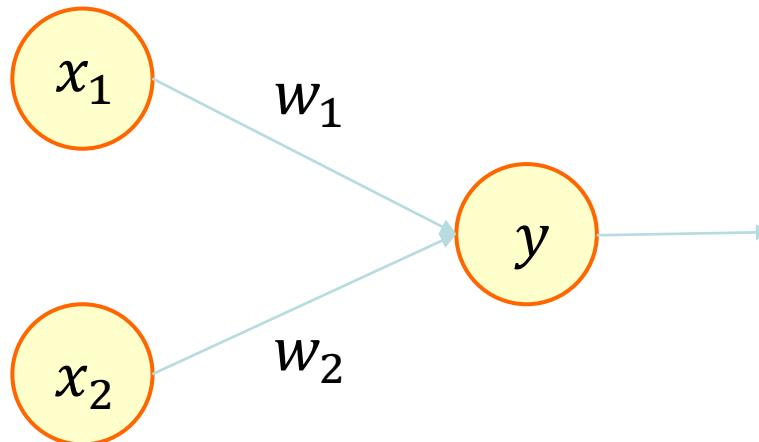
2.2 단순한 논리 회로

2.3 퍼셉트론 구현하기

2.4 퍼셉트론의 한계

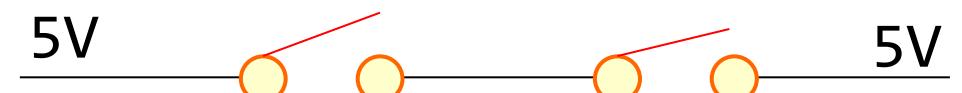
2.5 다층 퍼셉트론

2.6 NAND에서 컴퓨터 까지



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

$$(w_1, w_2, \theta) \Rightarrow (0.5, 0.5, 0.7), (0.5, 0.5, 0.8)$$

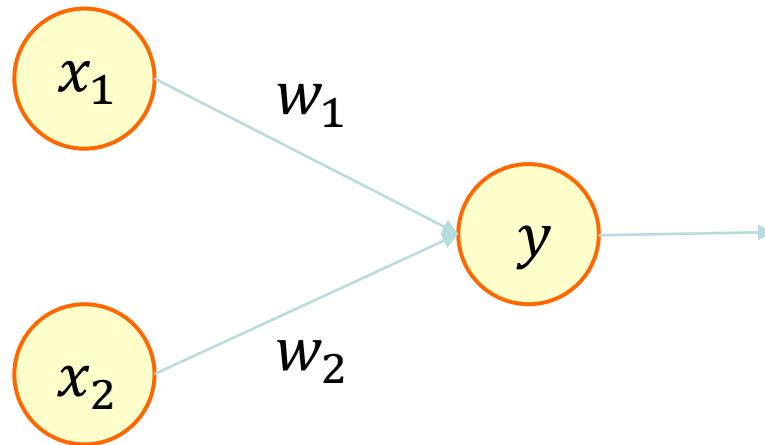


$$(0.5 * 0 + 0.5 * 0 \leq 0.8) 0$$

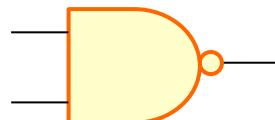
$$(0.5 * 0 + 0.5 * 1 \leq 0.8) 0$$

$$(0.5 * 1 + 0.5 * 0 \leq 0.8) 0$$

$$(0.5 * 1 + 0.5 * 1 > 0.8) 1$$

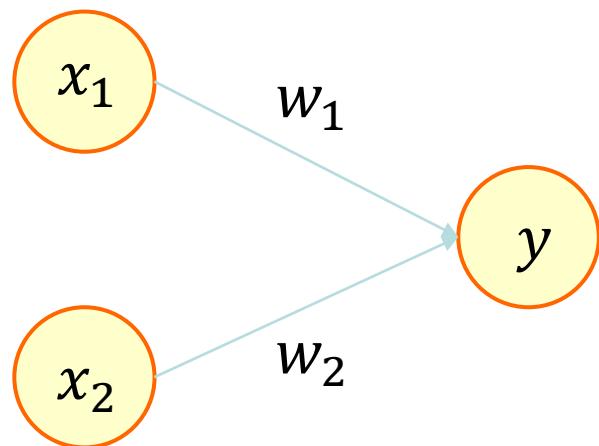


x_1	x_2	y
0	0	1
0	1	1
1	0	1
1	1	0



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

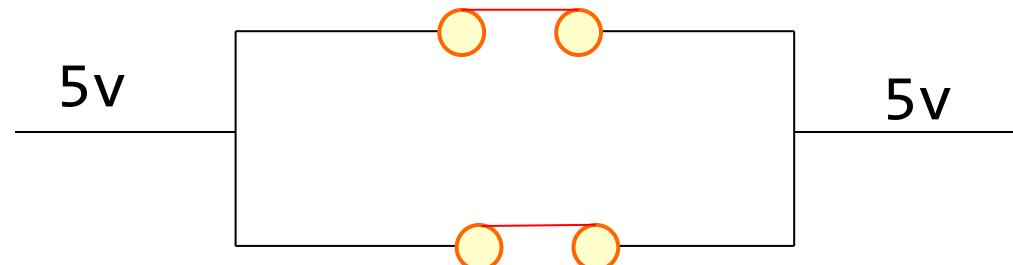
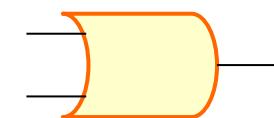
$$(w_1, w_2, \theta) \Rightarrow (-0.5, -0.5, -0.7), (-0.5, -0.5, -0.8)$$



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

$$(w_1, w_2, \theta) \Rightarrow (1.0, 1.0, 0.7)$$

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



$$(1.0 * 0 + 1.0 * 0 \leq 0.7) \ 0$$

$$(1.0 * 0 + 1.0 * 1 > 0.7) \ 1$$

$$(1.0 * 1 + 1.0 * 0 > 0.7) \ 1$$

$$(1.0 * 1 + 1.0 * 1 > 0.7) \ 1$$

$$\begin{array}{r}
 & 0 \\
 + & 0 \\
 \hline
 0 & 0
 \end{array}
 \quad
 \begin{array}{r}
 & 0 \\
 + & 1 \\
 \hline
 0 & 1
 \end{array}$$

carry carry

$$\begin{array}{r}
 1 \\
 + 0 \\
 \hline
 0 & 1
 \end{array}
 \quad
 \begin{array}{r}
 & 1 \\
 + & 1 \\
 \hline
 1 & 0
 \end{array}$$

carry carry

x_1	x_2	carry	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

and xor

2. 퍼셉트론

2.1 퍼셉트론이란?

2.2 단순한 논리 회로

2.3 퍼셉트론 구현하기

2.4 퍼셉트론의 한계

2.5 다층 퍼셉트론

2.6 NAND에서 컴퓨터 까지

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

```
print(AND(0, 0)) # 0 을 출력
print(AND(0, 1)) # 0 을 출력
print(AND(1, 0)) # 0 을 출력
print(AND(1, 1)) # 1 을 출력
```

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

$$(w_1, w_2, \theta) \Rightarrow (0.5, 0.5, 0.7)$$

가중치와 편향을 도입하자.

2.3 퍼셉트론 구현하기

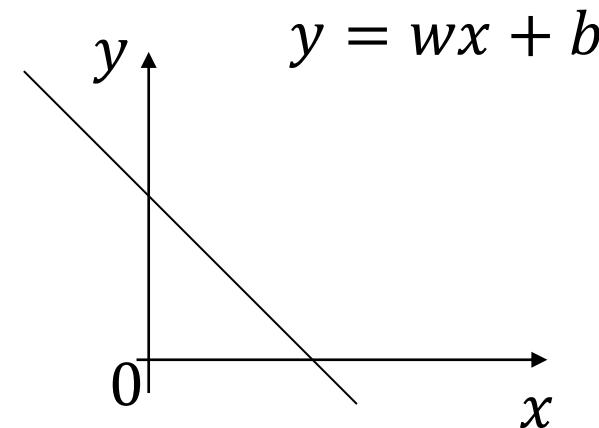
```
def AND(x1, x2):  
    w1, w2, b = 0.5, 0.5, -0.7  
    tmp = x1*w1 + x2*w2 + b  
    if tmp <= 0:  
        return 0  
    elif tmp > 0:  
        return 1
```

AND(0, 0) # 0 을 출력
AND(0, 1) # 0 을 출력
AND(1, 0) # 0 을 출력
AND(1, 1) # 1 을 출력

$$b = -\theta$$

$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

$$(w_1, w_2, b) \Rightarrow (0.5, 0.5, -0.7)$$



```

import numpy as np

def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (0, 1), (1, 0), (1, 1)]:
        y = AND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))

```

$$x \begin{matrix} 0 \\ 0 \end{matrix}$$

$$w \begin{matrix} 0.5 \\ 0.5 \end{matrix}$$

$$x^*w \begin{matrix} 0 \\ 0 \end{matrix}$$

`np.sum(x*w)+b`

`np.dot(x,w)+b`

```
import numpy as np

def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (0, 1), (1, 0), (1, 1)]:
        y = NAND(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

```
import numpy as np

def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1

if __name__ == '__main__':
    for xs in [(0, 0), (0, 1), (1, 0), (1, 1)]:
        y = OR(xs[0], xs[1])
        print(str(xs) + " -> " + str(y))
```

2. 퍼셉트론

2.1 퍼셉트론이란?

2.2 단순한 논리 회로

2.3 퍼셉트론 구현하기

2.4 퍼셉트론의 한계

2.5 다층 퍼셉트론

2.6 NAND에서 컴퓨터 까지

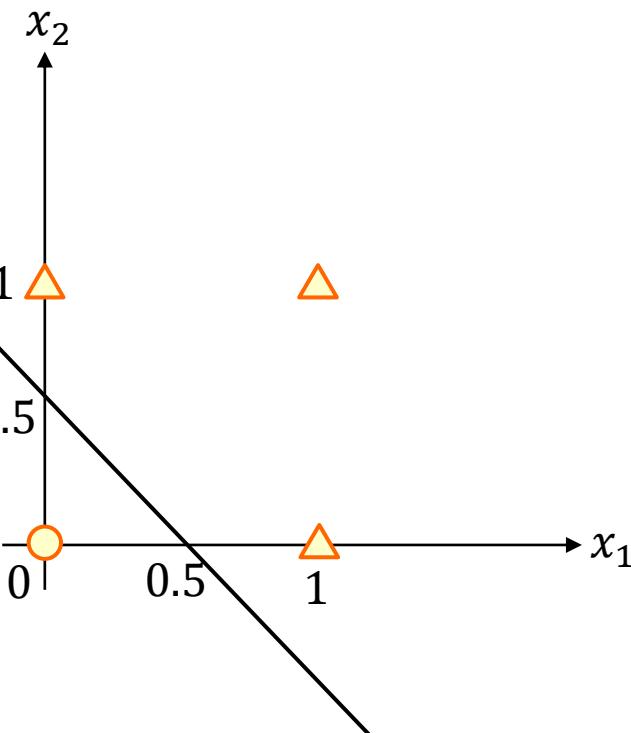
도전! XOR 게이트

2.4 퍼셉트론의 한계

or gate 결정 경계

$$y = \begin{cases} 0 & (-0.5 + x_1 + x_2 \leq 0) \\ 1 & (-0.5 + x_1 + x_2 > 0) \end{cases}$$

$$(w_1, w_2, b) \Rightarrow (1, 1, -0.5)$$



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

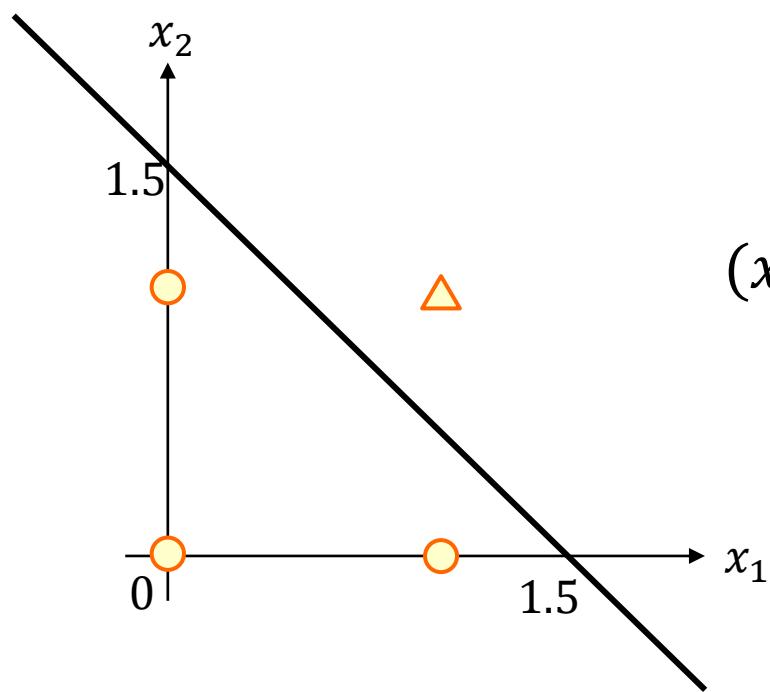
$$x_2 = 0.5 - x_1$$

$$(x_2 = 0.5 + -x_1)$$

and gate 결정 경계

$$y = \begin{cases} 0 & (-1.5 + x_1 + x_2 \leq 0) \\ 1 & (-1.5 + x_1 + x_2 > 0) \end{cases}$$

$$(w_1, w_2, b) \Rightarrow (1, 1, -1.5)$$



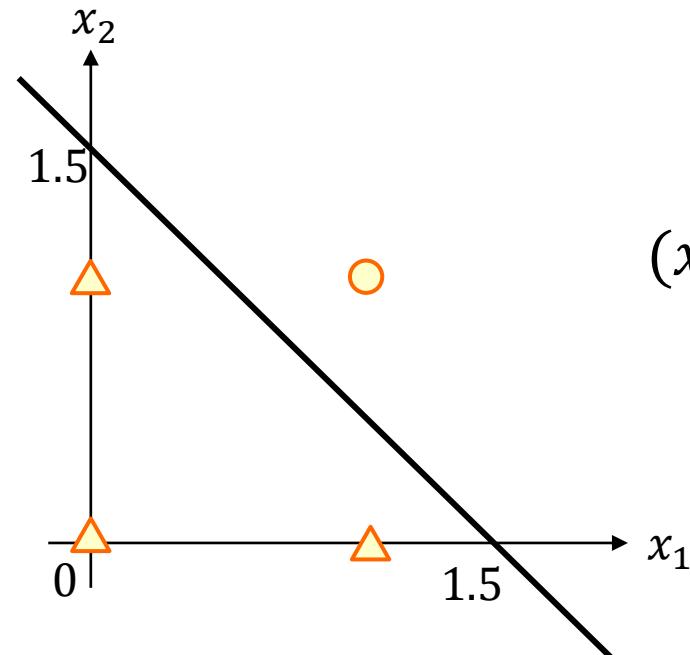
$$(x_2 = 1.5 + -x_1)$$

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

nand gate 결정 경계

$$y = \begin{cases} 0 & (1.5 - x_1 - x_2 \leq 0) \\ 1 & (1.5 - x_1 - x_2 > 0) \end{cases}$$

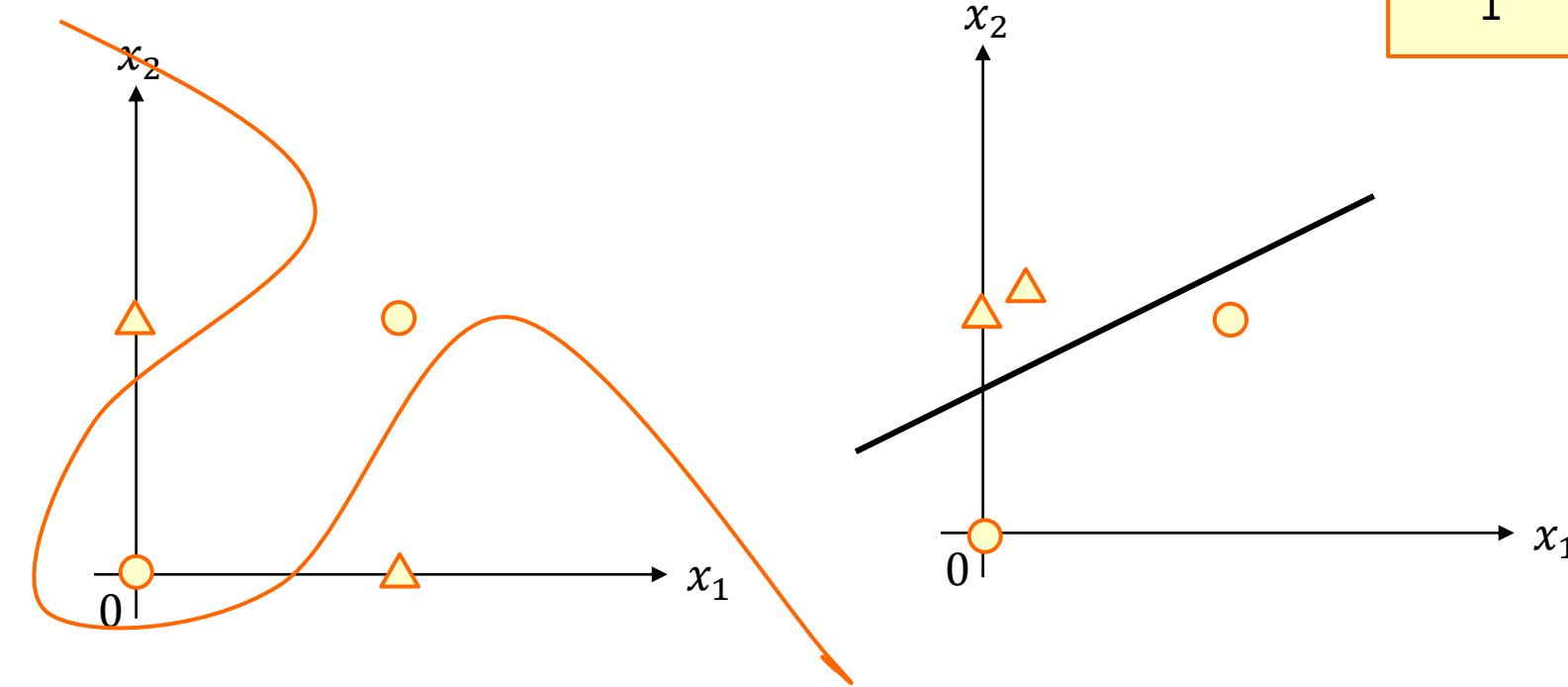
$$(w_1, w_2, b) \Rightarrow (-1, -1, 1.5)$$



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

xor gate 결정 경계

: 직선 방정식으로는 해결할 수 없다.



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

2. 퍼셉트론

2.1 퍼셉트론이란?

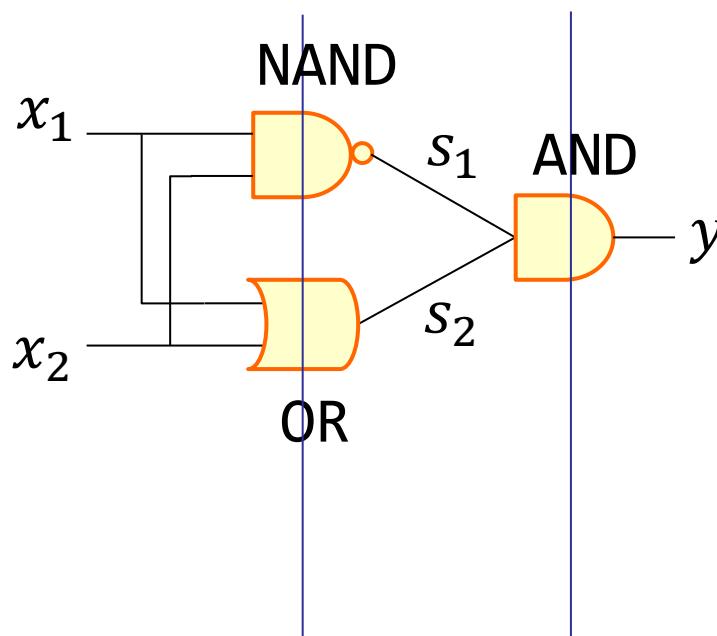
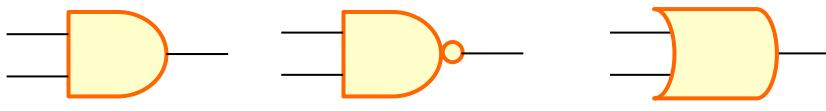
2.2 단순한 논리 회로

2.3 퍼셉트론 구현하기

2.4 퍼셉트론의 한계

2.5 다층 퍼셉트론

2.6 NAND에서 컴퓨터 까지

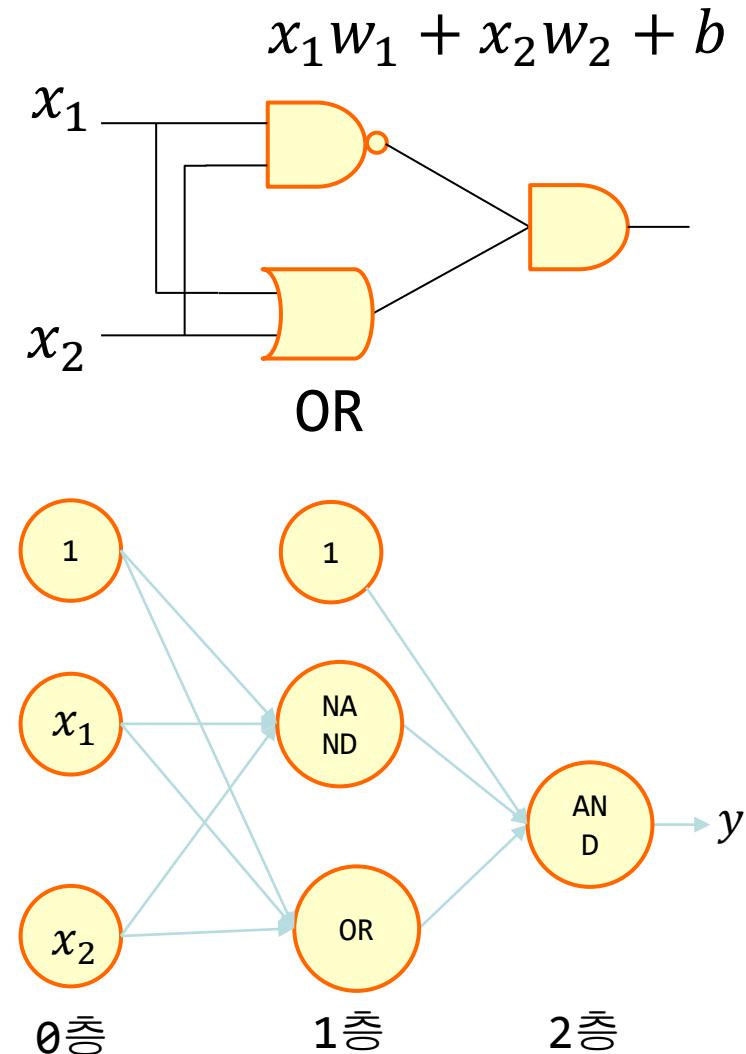


x_1	x_2	s_1	s_2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

XOR 게이트 구현하기

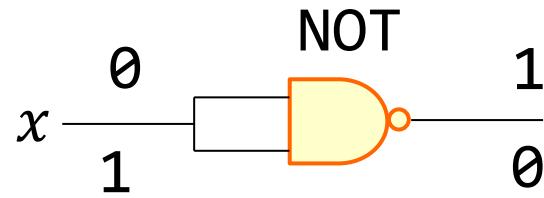
2.5 다층 퍼셉트론

```
from and_gate import AND  
from or_gate import OR  
from nand_gate import NAND  
  
def XOR(x1, x2):  
    s1 = NAND(x1, x2)  
    s2 = OR(x1, x2)  
    y = AND(s1, s2)  
    return y  
  
if __name__ == '__main__':  
    for xs in [(0, 0), (0, 1), (1, 0), (1, 1)]:  
        y = XOR(xs[0], xs[1])  
        print(str(xs) + " -> " + str(y))
```



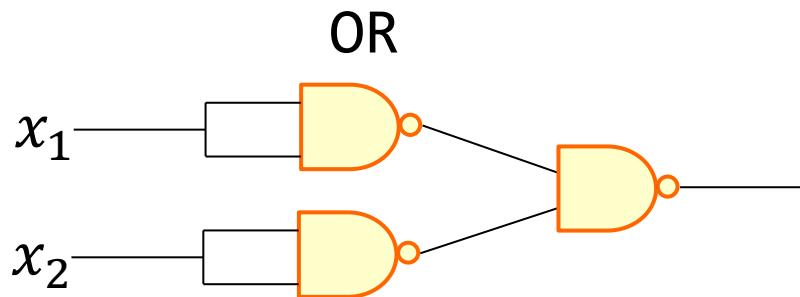
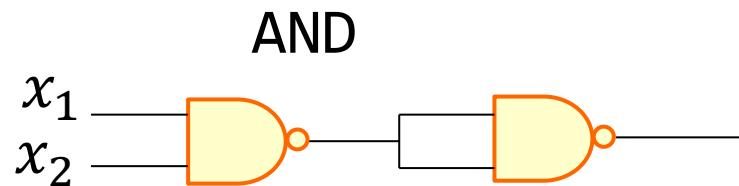
2. 퍼셉트론

- 2.1 퍼셉트론이란?
 - 2.2 단순한 논리 회로
 - 2.3 퍼셉트론 구현하기
 - 2.4 퍼셉트론의 한계
 - 2.5 다층 퍼셉트론
- 2.6 NAND에서 컴퓨터 까지**
-



$$(x_1 x_2)' \Rightarrow (x'_1 + x'_2)$$

$$\begin{array}{r}
 & & 1 & 1 & 1 & 1 \\
 & & 1 & 1 & 0 & 1 \\
 & & 1 & 0 & 1 & 1 \\
 0 & + & 0 & 0 & 0 & 0
 \end{array}$$



$$\begin{array}{r}
 & 1 & 1 & 1 & 0 \\
 & 1 & 0 & 0 & 1 \\
 0 & 1 & 0 & 1 \\
 0 & 0 & 0 & 1
 \end{array}$$

3. 신경망

3.1 퍼셉트론에서 신경망으로

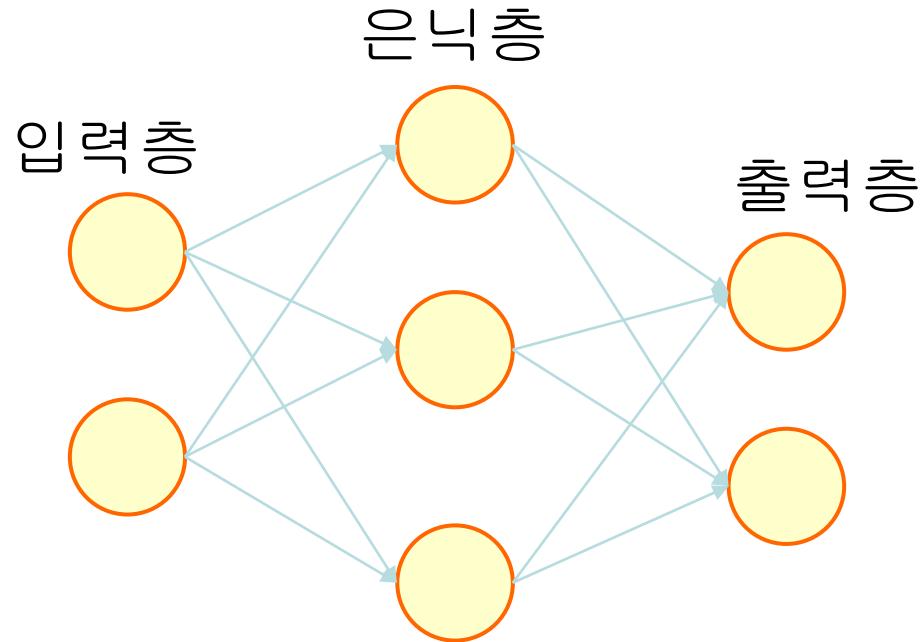
3.2 활성화 함수

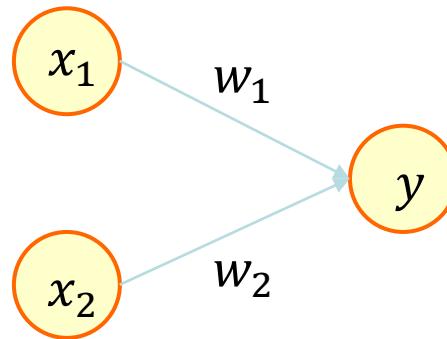
3.3 다차원 배열의 계산

3.4 3층 신경망 구현하기

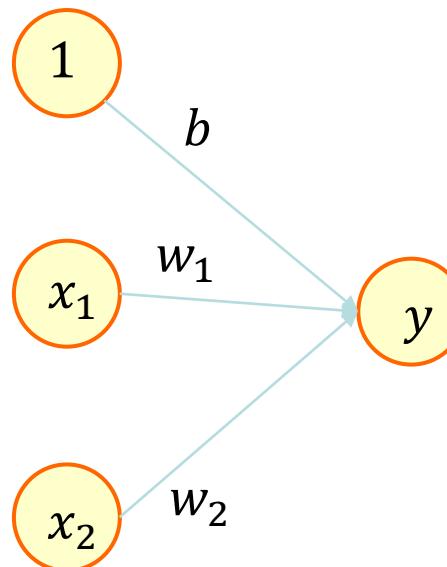
3.5 출력층 설계하기

3.6 손글씨 숫자 인식





$$y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

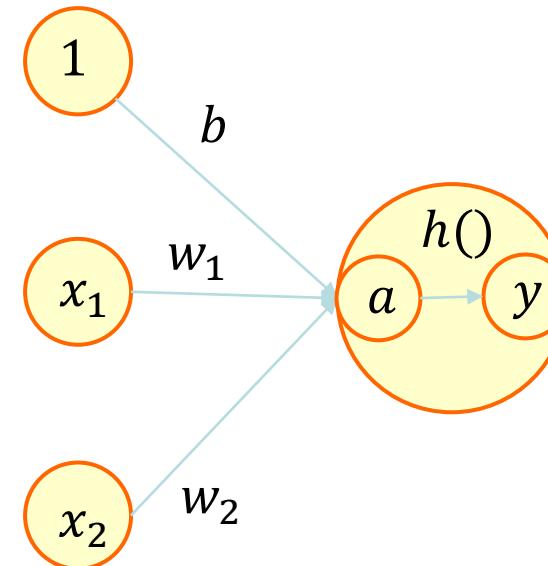


$$y = h(b + w_1x_1 + w_2x_2)$$

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

$$a = b + w_1x_1 + w_2x_2$$

$$y = h(a)$$



3. 신경망

3.1 퍼셉트론에서 신경망으로

3.2 활성화 함수

3.3 다차원 배열의 계산

3.4 3층 신경망 구현하기

3.5 출력층 설계하기

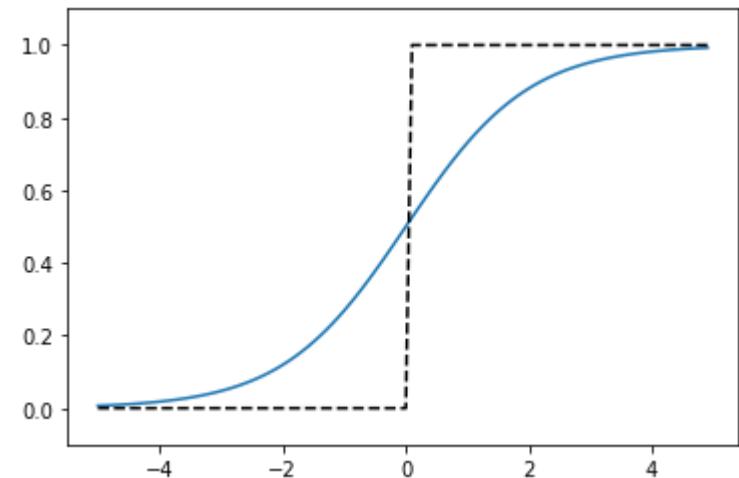
3.6 손글씨 숫자 인식

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

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

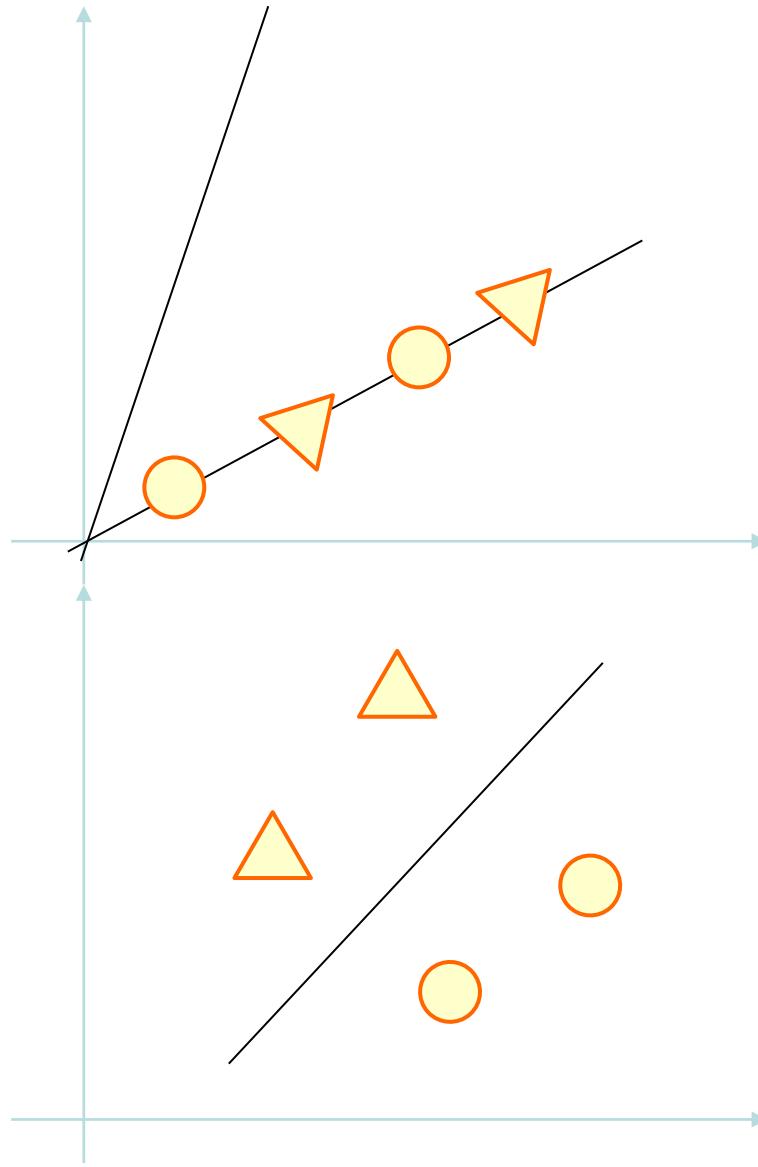
```
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))
```

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



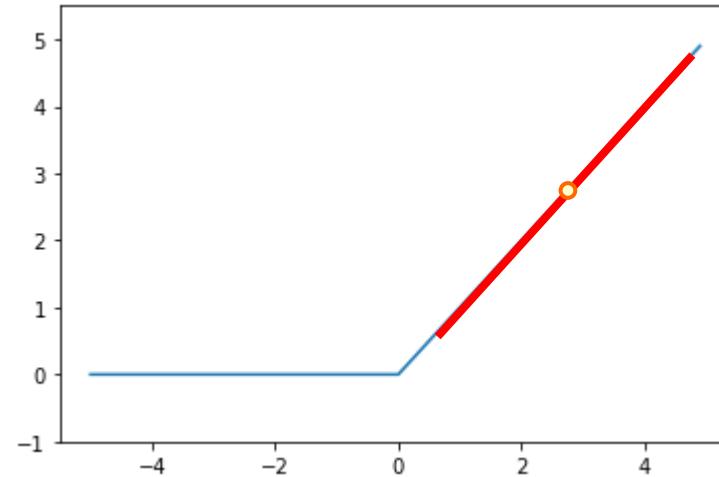
$$h(x) = cx$$

$$h(h(h(x))) = c * c * c * x$$



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

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



3. 신경망

3.1 퍼셉트론에서 신경망으로

3.2 활성화 함수

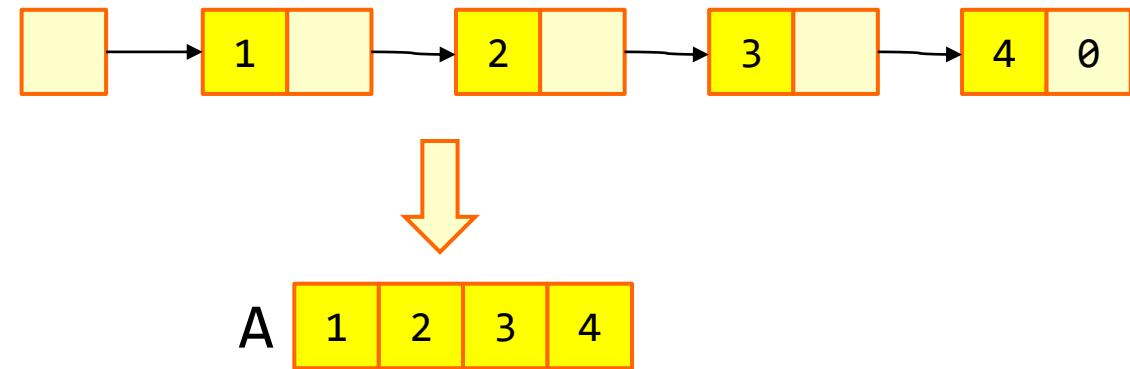
3.3 다차원 배열의 계산

3.4 3층 신경망 구현하기

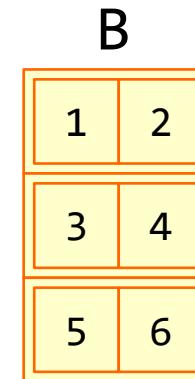
3.5 출력층 설계하기

3.6 손글씨 숫자 인식

```
l1 = [1,2,3,4]
A = np.array([1, 2, 3, 4])
print(A)      # [1 2 3 4]    l1
np.ndim(A)    # 1
A.shape       # (4,)
A.shape[0]    # 4
len(A)        # 4
```



```
B = np.array([[1,2], [3,4], [5,6]])
print(B)      # [[1 2] [3 4] [5 6]]
np.ndim(B)    # 2
B.shape       # (3, 2)
B.shape[0]    # 3
len(B)        # 3
```



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

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

```
np.dot(A,B) # (2,2)(2,2) => (2,2)
```

$$\begin{matrix} & 1 \times 5 + 2 \times 7 \\ \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} & \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \\ & 3 \times 5 + 4 \times 7 \end{matrix}$$

The diagram illustrates the calculation of the dot product of two 2x2 matrices, A and B. The matrices are:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}, \quad B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$$

The resulting matrix is:

$$AB = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix}$$

The calculation is shown as:

$$(1 \times 5 + 2 \times 7) \quad (3 \times 5 + 4 \times 7)$$

Elements 1, 2, 5, 6, 7, 8 are highlighted in red boxes. Arrows point from these highlighted elements to the terms in the calculation formulas.

```
A = np.array([[1,2,3], [4,5,6]])  
B = np.array([[1,2], [3,4], [5,6]])  
  
np.dot(A,B)
```

$$(2, 3) \cdot (3, 2) \Rightarrow (2, 2)$$



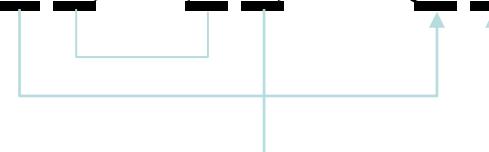
$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline 4 & 5 & 6 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 22 & 28 \\ \hline 49 & 64 \\ \hline \end{array}$$

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

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

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

$$(2, 3) \cdot (2, 2) \Rightarrow (2, 2)$$



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

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

$$(2,) \cdot (2,) \Rightarrow () \Rightarrow \text{값}$$

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 3 & 4 \\ \hline \end{array} = 11$$

$$1*3 + 2*4 = 11$$

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

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

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

$$\begin{matrix} 1 \times 7 + 2 \times 8 \\ \left(\begin{matrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{matrix} \right) \left(\begin{matrix} 7 & 8 \end{matrix} \right) = (23 \quad 53 \quad 83) \end{matrix}$$

$$(3, 2) \cdot (2,) \Rightarrow (3,)$$

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

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

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

$$(3,2) \cdot (2,1) \Rightarrow (3,1)$$

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline 5 & 6 \\ \hline \end{array} \quad \cdot \quad \begin{array}{|c|} \hline 7 \\ \hline 8 \\ \hline \end{array} \quad = \quad \begin{array}{|c|} \hline 23 \\ \hline 53 \\ \hline 83 \\ \hline \end{array}$$

```
A = np.array([[1,2], [3,4], [5,6]])
B = np.array([7,8,9])
```

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

$$7 \times 1 + 8 \times 3 + 9 \times 5$$

$$7 \times 2 + 8 \times 4 + 9 \times 6$$

~~$(3, 2) \cdot (3,) \Rightarrow$~~

$\underline{(3,)} \cdot \underline{(3, 2)} \Rightarrow \underline{(2,)}$

$$(7 \quad 8 \quad 9) \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} = (76 \quad 100)$$

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

```
B = np.array([[4,5], [6,7]])
```

$$(2,2) \cdot (2,2) \Rightarrow (2,2)$$

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

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

$$(2,2) \cdot (2,2) \Rightarrow (2,2)$$

1	2
3	4

•

4	5
6	7

=

16	19
36	43

4	5
6	7

•

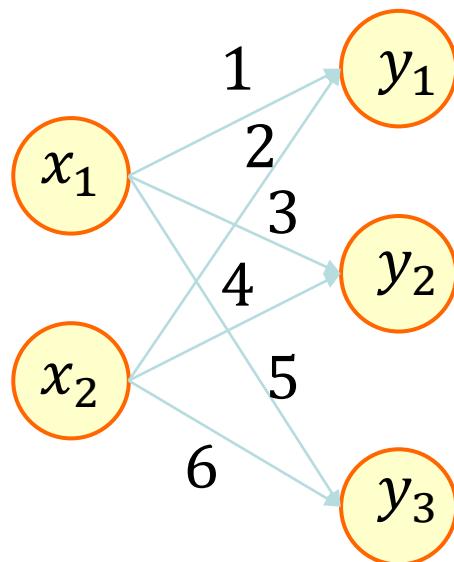
1	2
3	4

=

19	28
27	40

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

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

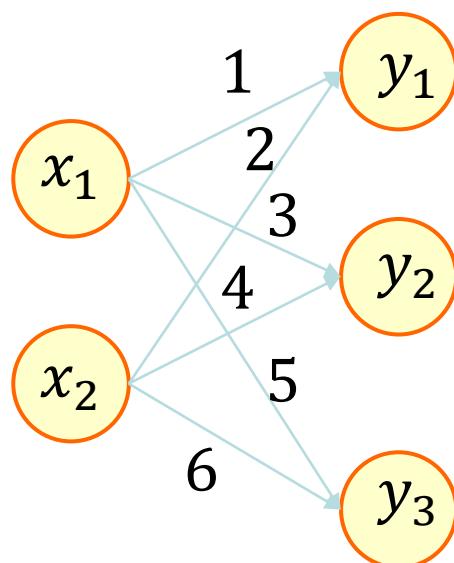


$$\begin{matrix} X & W & Y \\ (\underline{\underline{2}},) \cdot (\underline{\underline{2,3}}) \Rightarrow (\underline{\underline{3}},) \end{matrix}$$

$$(1 \quad 2) \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = (5 \quad 11 \quad 17)$$

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

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



$$\begin{matrix} X & W & Y \\ \underline{(1, 2)} & \cdot \underline{(2, 3)} & \Rightarrow \underline{(1, 3)} \end{matrix}$$

$$(1 \quad 2) \begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix} = [[5 \ 9 \ 17]]$$

3. 신경망

3.1 퍼셉트론에서 신경망으로

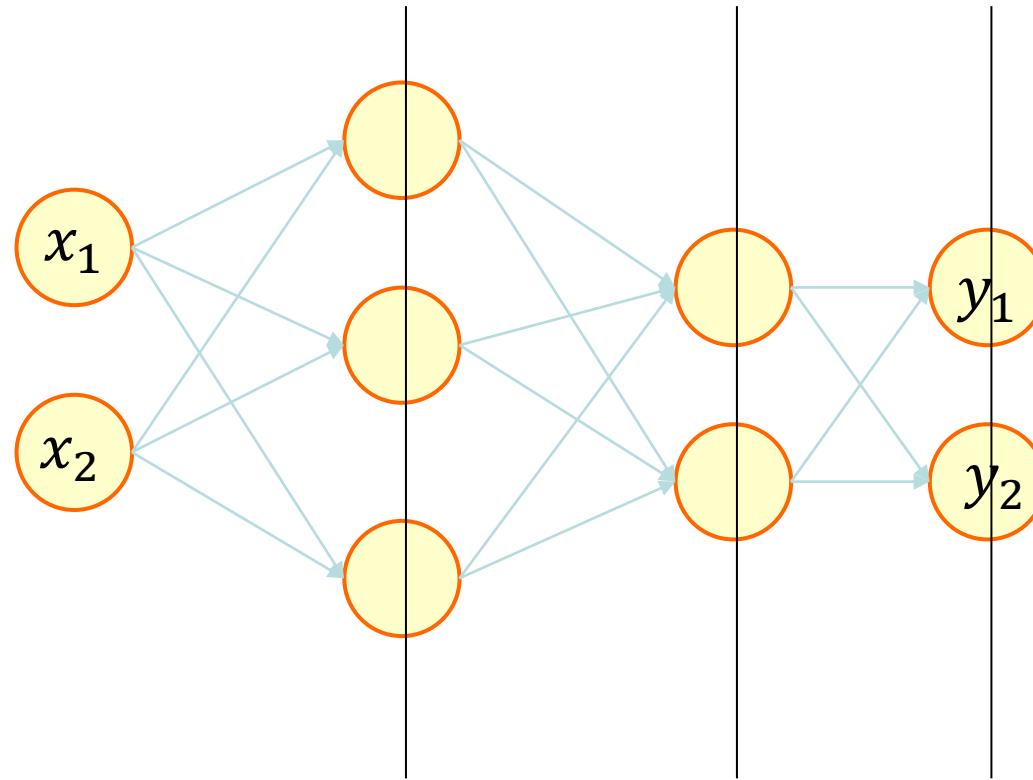
3.2 활성화 함수

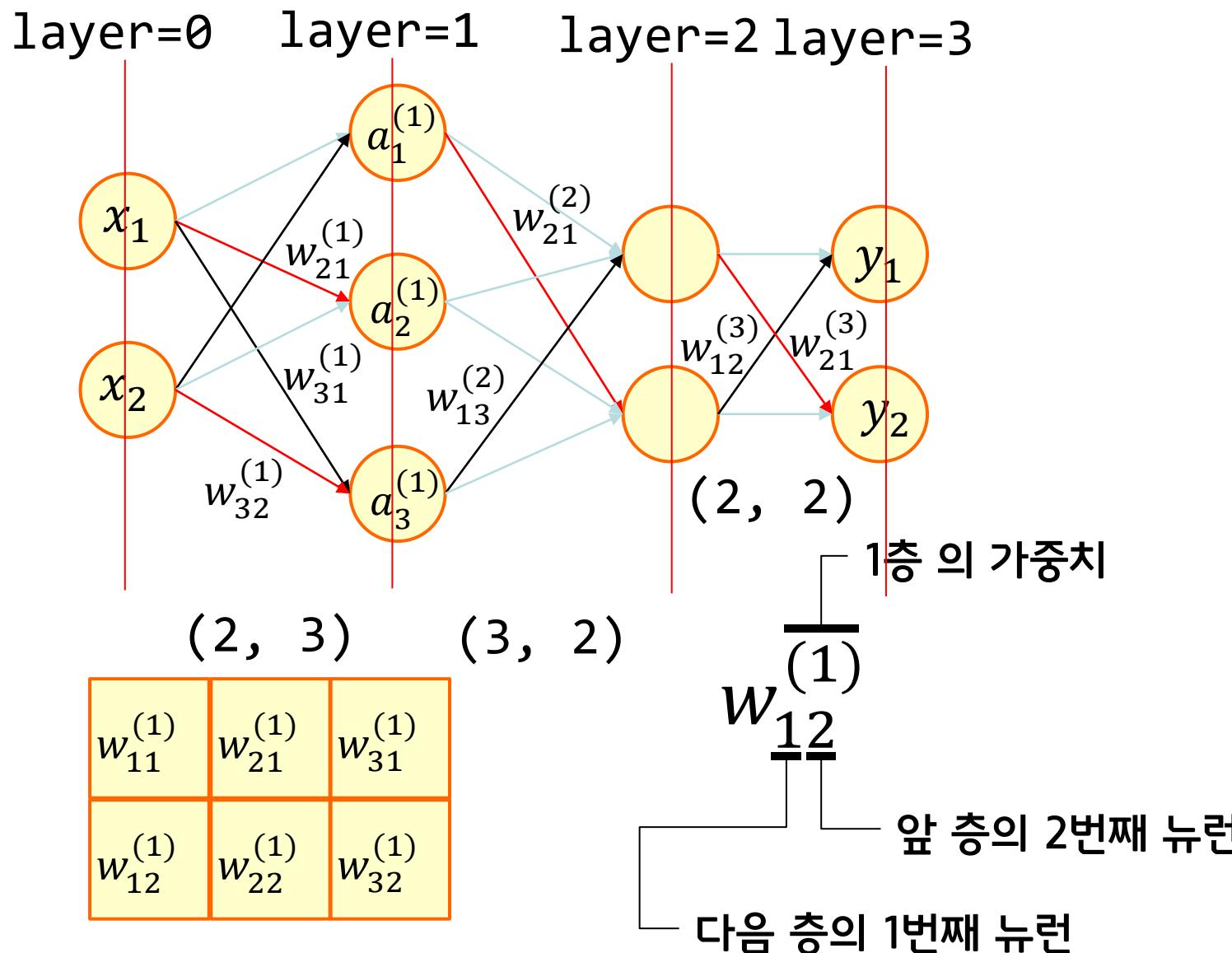
3.3 다차원 배열의 계산

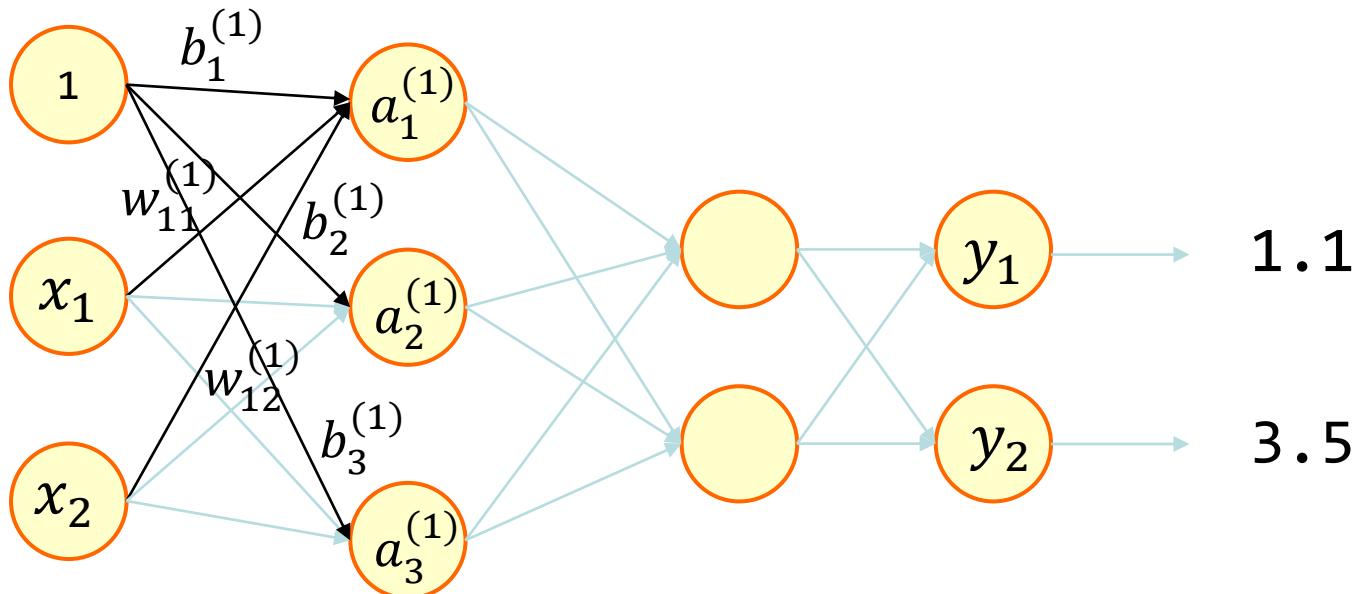
3.4 3층 신경망 구현하기

3.5 출력층 설계하기

3.6 손글씨 숫자 인식



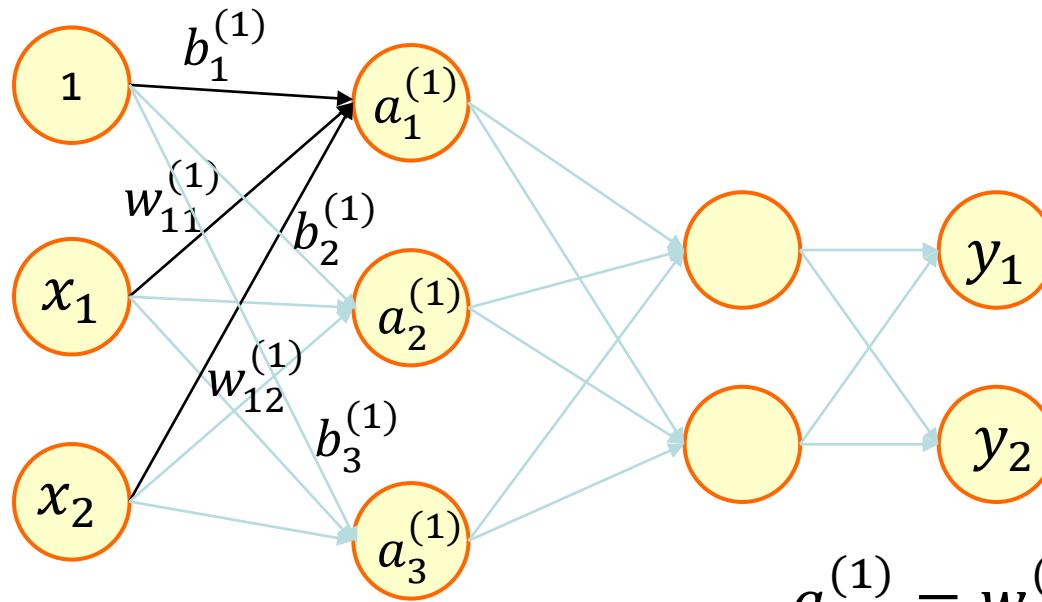




$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$a_2^{(1)} = w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)}$$

$$a_3^{(1)} = w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + b_3^{(1)}$$



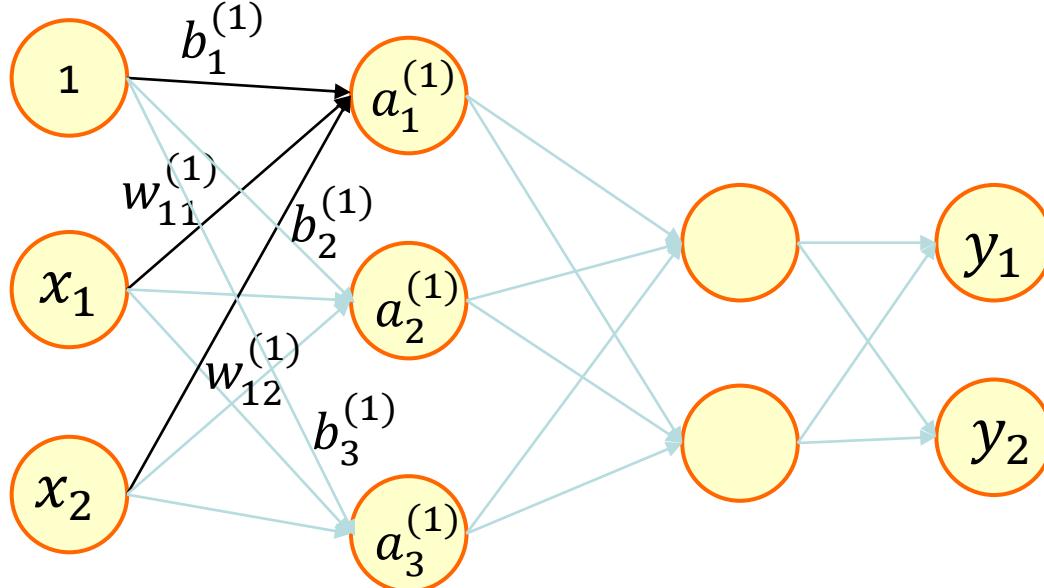
$$a_1^{(1)} = w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + b_1^{(1)}$$

$$A^{(1)} = XW^{(1)} + B^{(1)}$$

$$A^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)}), \quad X = (x_1, x_2), \quad B^{(1)} = (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

$$W^{(1)} = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

$w_{11}^{(1)}$	$w_{21}^{(1)}$	$w_{31}^{(1)}$
$w_{12}^{(1)}$	$w_{22}^{(1)}$	$w_{32}^{(1)}$



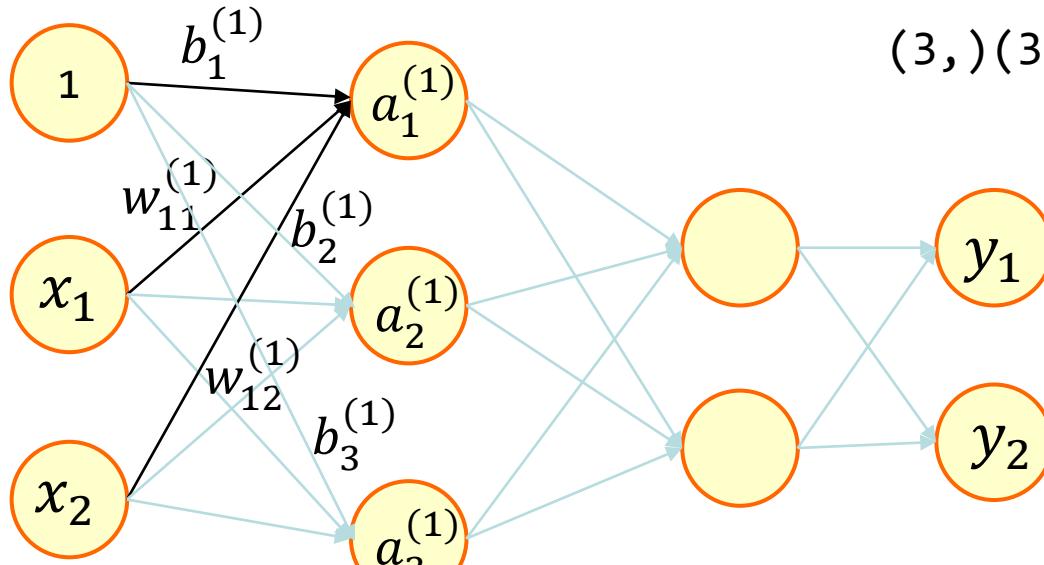
$$A^{(1)} = XW^{(1)} + B^{(1)}$$

$$A^{(1)} = (a_1^{(1)}, a_2^{(1)}, a_3^{(1)}), X = (x_1, x_2), B^{(1)} = (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

$$(2,) \cdot (2, 3) + (3,) \Rightarrow (3,)$$

$$(x_1, x_2) \cdot \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix} + (b_1^{(1)}, b_2^{(1)}, b_3^{(1)})$$

$$\begin{aligned} a_1^{(1)} &= x_1 w_{11}^{(1)} + x_2 w_{12}^{(1)} + b_1^{(1)} \\ a_2^{(1)} &= x_1 w_{21}^{(1)} + x_2 w_{22}^{(1)} + b_2^{(1)} \\ a_3^{(1)} &= x_1 w_{31}^{(1)} + x_2 w_{32}^{(1)} + b_3^{(1)} \end{aligned}$$

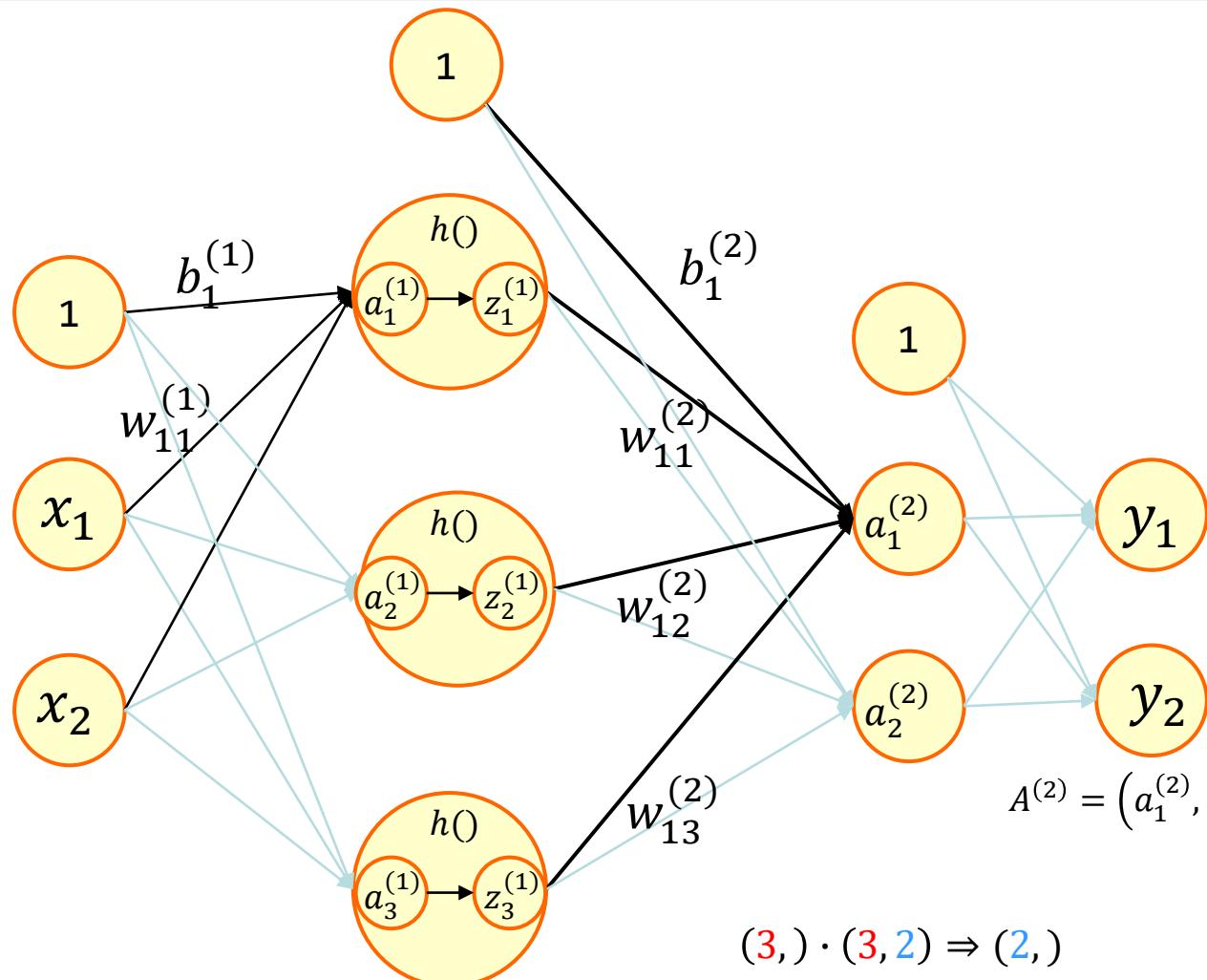
 $(3,)(3,) \Rightarrow ()$

$$\begin{array}{c} X \\ \cdot \\ W1 \\ = \\ \begin{bmatrix} 1.0 & 0.5 \end{bmatrix} \cdot \begin{bmatrix} 0.1 & 0.3 & 0.5 \\ 0.2 & 0.4 & 0.6 \end{bmatrix} \\ \\ B1 \\ = \\ \begin{bmatrix} 0.2 & 0.5 & 0.8 \end{bmatrix} + \begin{bmatrix} 0.1 & 0.2 & 0.3 \end{bmatrix} \\ \\ \begin{bmatrix} 0.3 & 0.7 & 1.1 \end{bmatrix} \end{array}$$

 $(2,)(2,3)+(3,) \Rightarrow (3,)$

```
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(X.shape)
print(W1.shape)
print(B1.shape)
```

```
A1 = np.dot(X, W1) + B1
print(A1.shape)
print(A1)
```



$$A^{(2)} = Z^{(1)}W^{(2)} + B^{(2)}$$

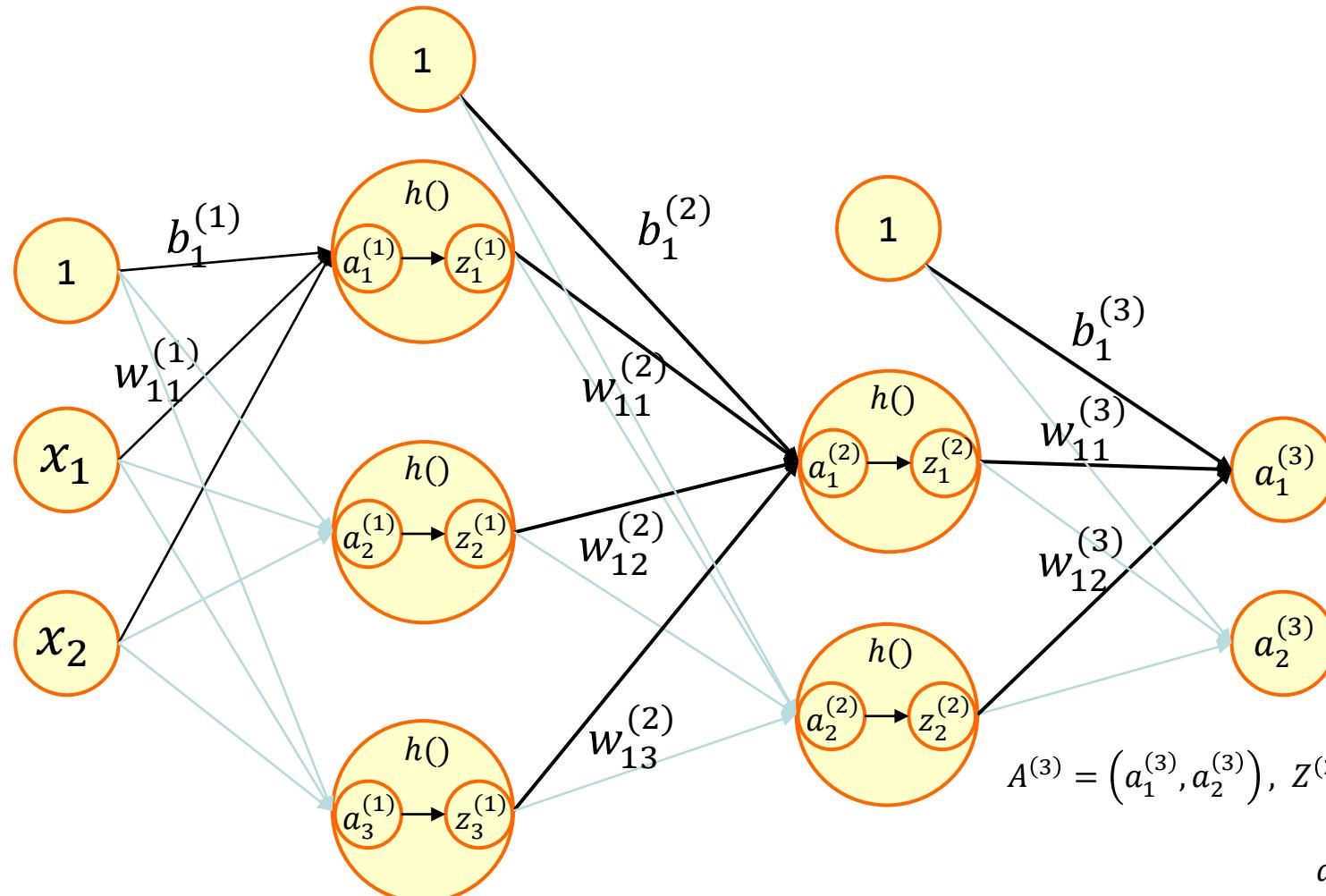
$$A^{(2)} = (a_1^{(2)}, a_2^{(2)}), \quad Z^{(1)} = (z_1^{(1)}, z_2^{(1)}, z_3^{(1)}), \quad B^{(2)} = (b_1^{(2)}, b_2^{(2)})$$

$$a_1^{(2)} = z_1^{(1)}w_{11}^{(2)} + z_2^{(1)}w_{12}^{(2)} + z_3^{(1)}w_{13}^{(2)} + b_1^{(2)}$$

$$a_2^{(2)} = z_1^{(1)}w_{21}^{(2)} + z_2^{(1)}w_{22}^{(2)} + z_3^{(1)}w_{23}^{(2)} + b_2^{(2)}$$

$$(3,) \cdot (3, 2) \Rightarrow (2,)$$

$$\left(z_1^{(1)}, z_2^{(1)}, z_3^{(1)} \right) \cdot \begin{pmatrix} w_{11}^{(2)} & w_{21}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} \\ w_{13}^{(2)} & w_{23}^{(2)} \end{pmatrix} + (b_1^{(2)}, b_2^{(2)})$$



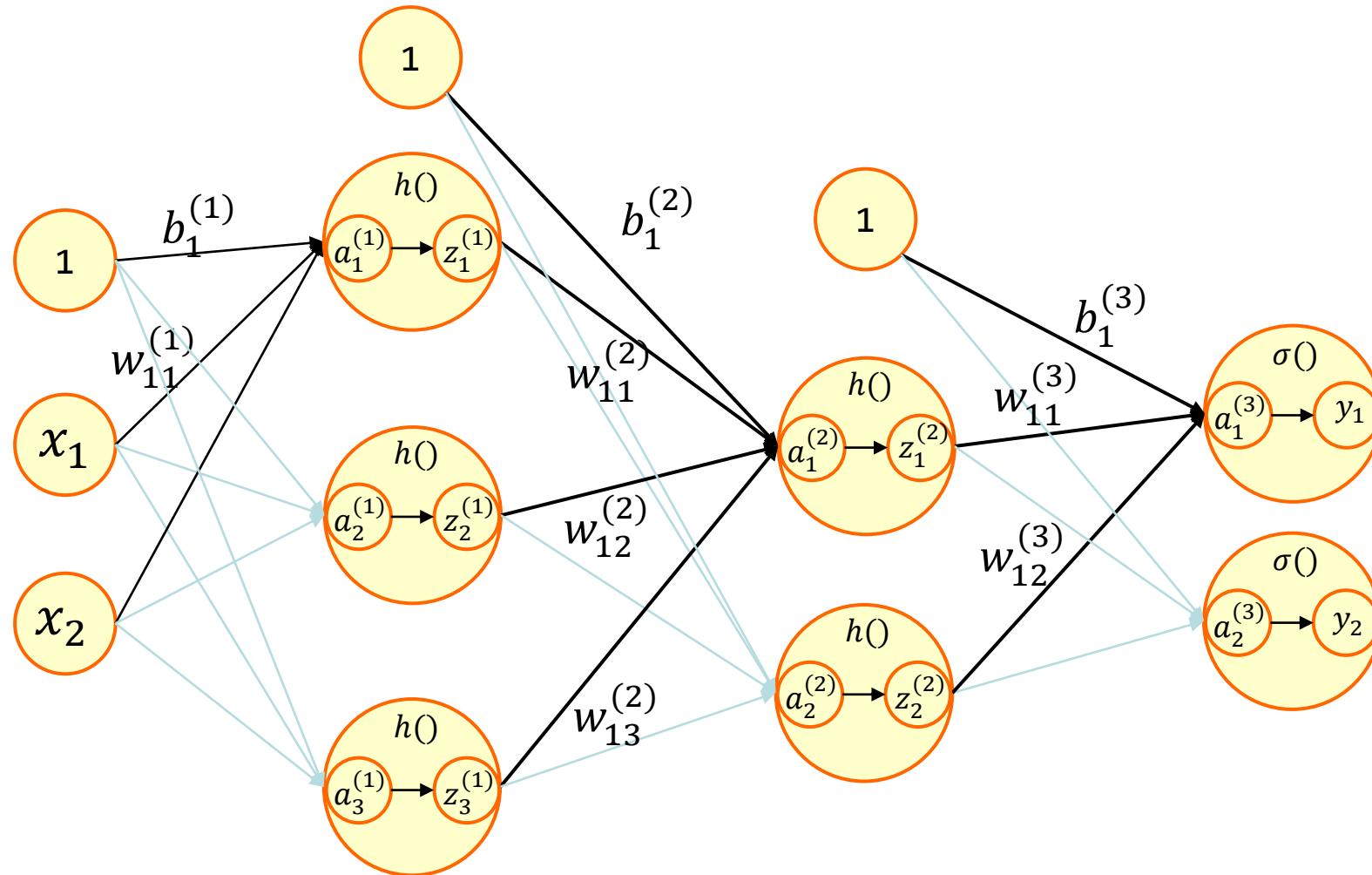
$$(2,) \cdot (2, 2) \Rightarrow (2,)$$

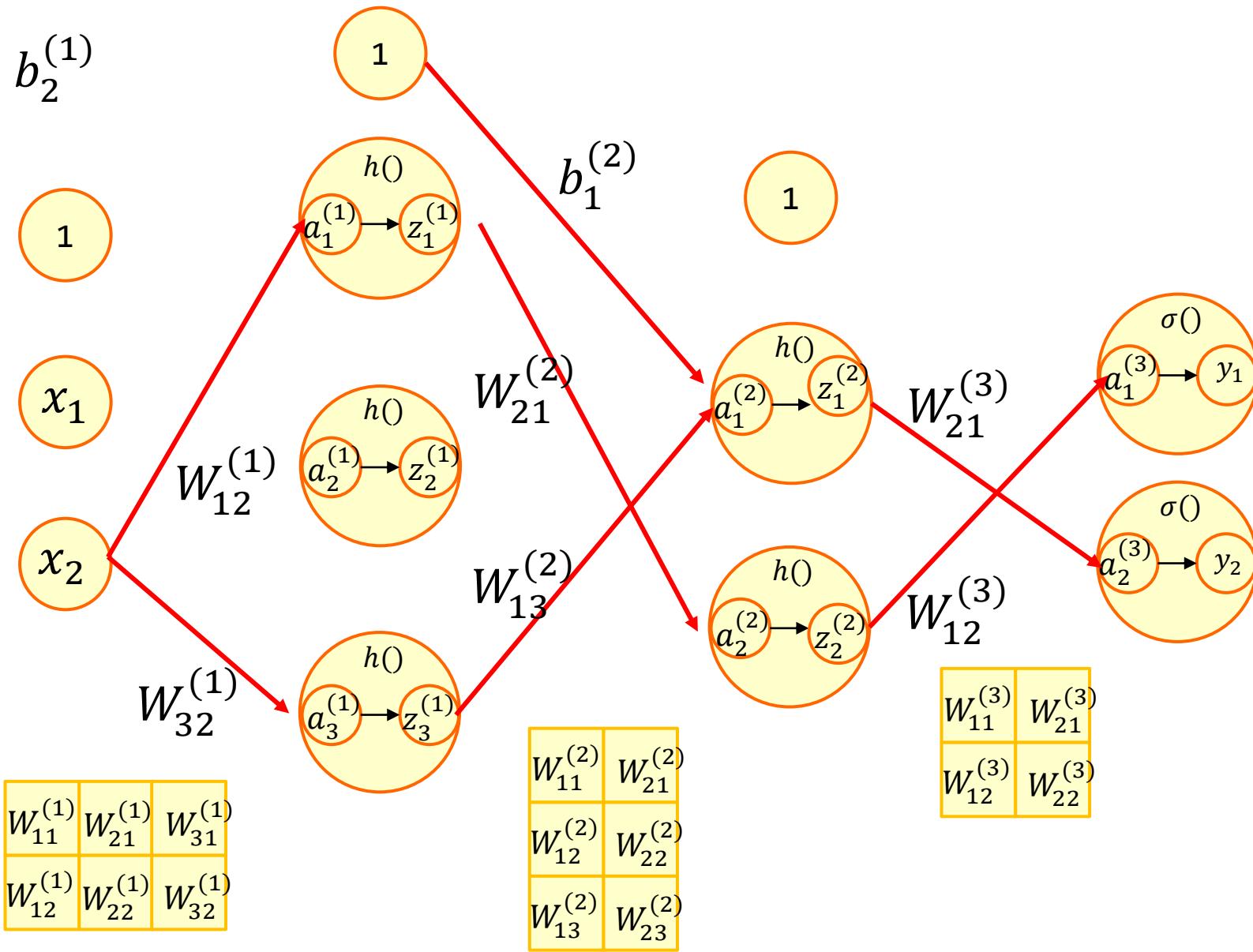
$$\begin{pmatrix} z_1^{(2)}, z_2^{(2)} \end{pmatrix} \cdot \begin{pmatrix} w_{11}^{(3)} & w_{21}^{(3)} \\ w_{12}^{(3)} & w_{22}^{(3)} \end{pmatrix} + (b_1^{(3)}, b_2^{(3)})$$

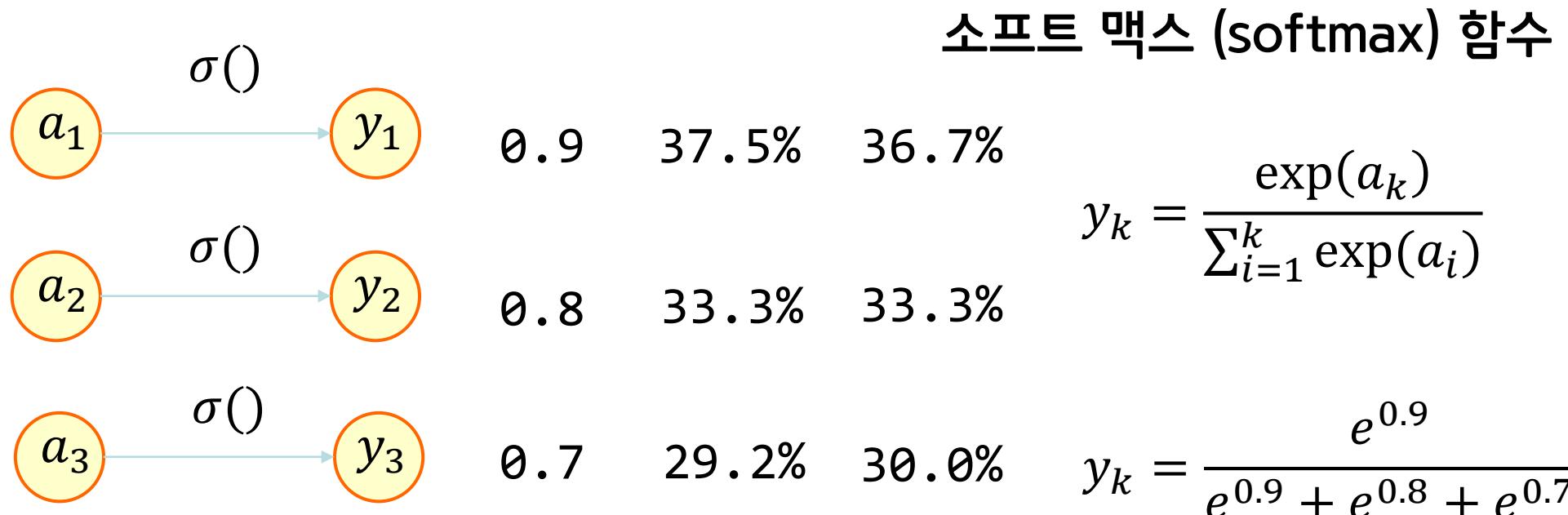
$$A^{(3)} = \begin{pmatrix} a_1^{(3)}, a_2^{(3)} \end{pmatrix}, Z^{(2)} = \begin{pmatrix} z_1^{(2)}, z_2^{(2)} \end{pmatrix}, B^{(3)} = (b_1^{(3)}, b_2^{(3)})$$

$$a_1^{(3)} = z_1^{(2)}w_{11}^{(3)} + z_2^{(2)}w_{21}^{(3)} + b_1^{(3)}$$

$$a_2^{(3)} = z_1^{(2)}w_{21}^{(3)} + z_2^{(2)}w_{22}^{(3)} + b_2^{(3)}$$







$$y_1 = \frac{0.9}{0.9 + 0.8 + 0.7}$$

$$y_1 = \frac{2.46}{2.46 + 2.23 + 2.01}$$

$$y_1 = \frac{2.46}{6.7}$$

$$y_2 = \frac{2.23}{6.7}$$

$$y_2 = \frac{2.01}{6.7}$$

a_1	$\sigma()$	y_1	2.2	100%	73%
a_2	$\sigma()$	y_2	1.1	50%	24%
a_3	$\sigma()$	y_3	-1.1	-50%	3%

소프트 맥스 (softmax) 함수

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

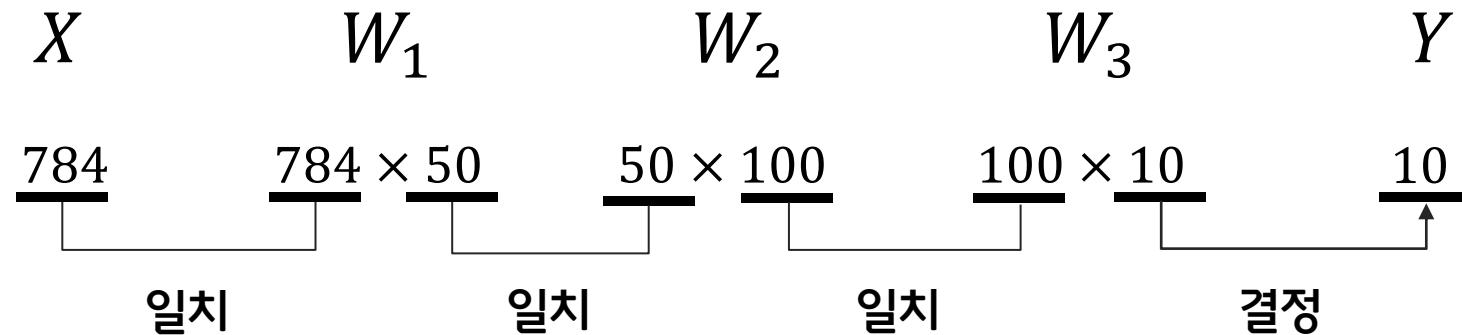
$$y_k = \frac{e^{2.2}}{e^{2.2} + e^{1.1} + e^{-1.1}}$$

$$y_1 = \frac{9.0}{12.3} \quad y_2 = \frac{3.0}{12.3} \quad y_3 = \frac{0.3}{12.3}$$

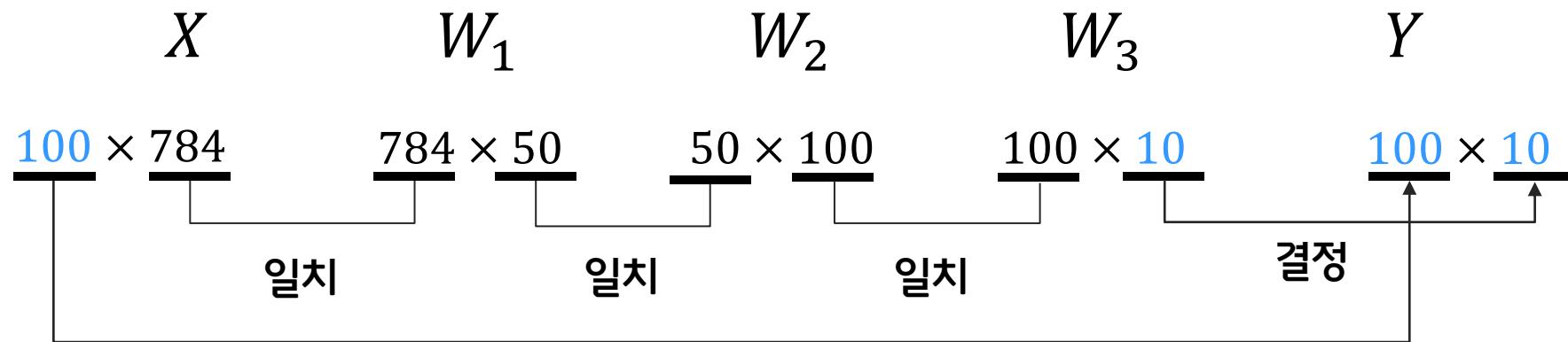
소프트 맥스 (softmax) 함수

$$\begin{aligned}y_k &= \frac{\exp(a_k)}{\sum_{i=1}^k \exp(a_i)} = \frac{C \exp(a_k)}{C \sum_{i=1}^k \exp(a_i)} \\&= \frac{\exp(a_k + \log C)}{\sum_{i=1}^k \exp(a_i + \log C)} \\&= \frac{\exp(a_k + C')}{\sum_{i=1}^k \exp(a_i + C')}\end{aligned}$$

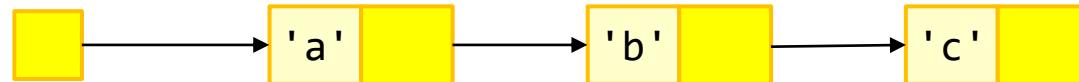
신경망 각 층의 배열의 형상



배치 처리를 위한 각 층의 배열의 형상

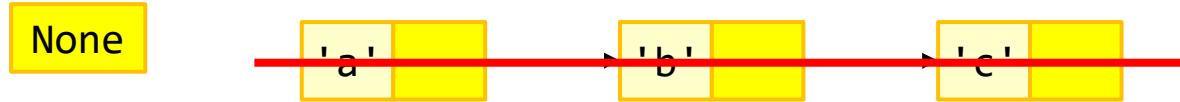


`my_list` `ref_count=1`



`['a', 'b', 'c']`

`my_list` `ref_count=0`



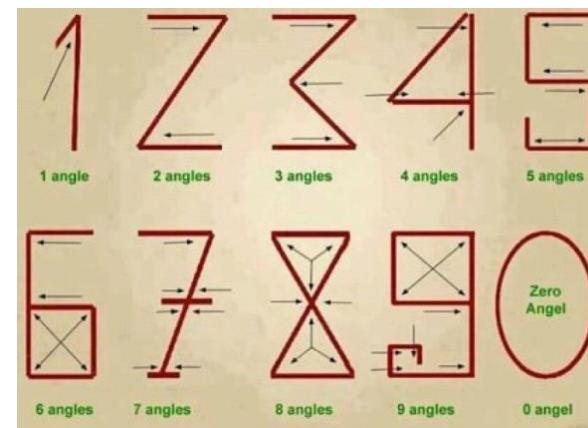
4. 신경망 학습

4.1 데이터에서 학습한다.

4.2 손실함수



1 Z Z 4



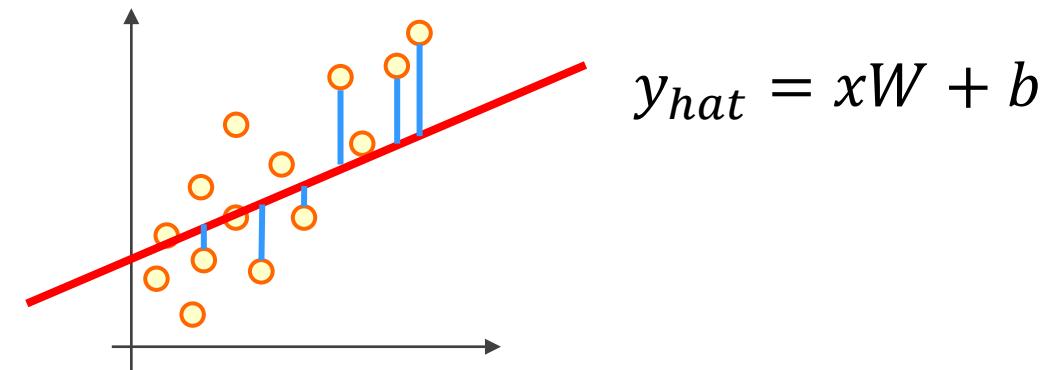
4. 신경망 학습

4.1 데이터에서 학습한다.

4.2 손실함수

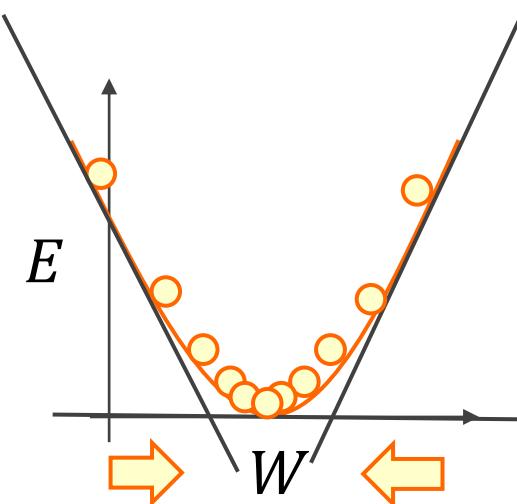
$$E = \frac{1}{2m} \sum_k^m (y_k - t_k)^2$$

MSE



$$E = \frac{1}{2} \sum_k^m (y_k - t_k)^2$$

SSE



GD

$$w = w - \frac{\partial E}{\partial w}$$

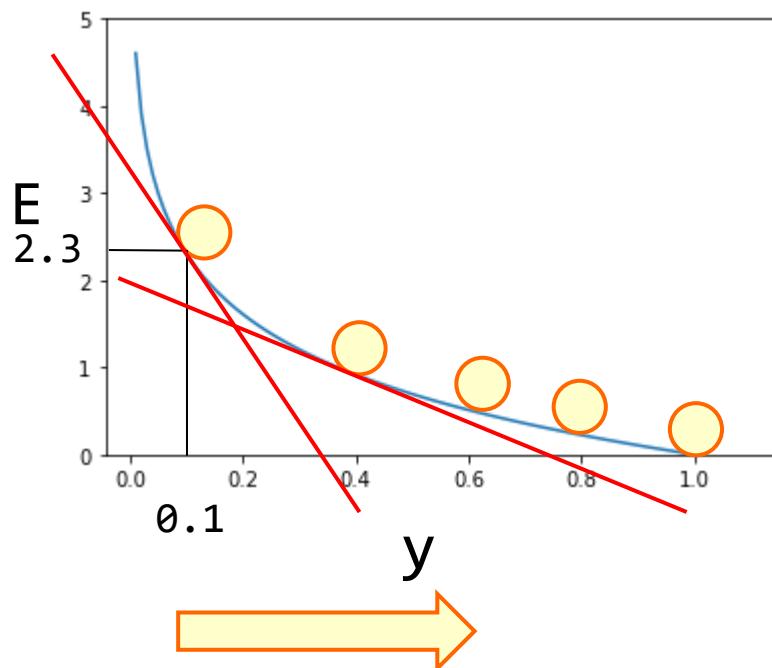
크로스 엔트로피(cross entropy error)

4.2 손실함수

$$E = - \sum_{k=1}^3 t_k \log(y_k)$$

$$E = -(t_1 \log(y_1) + t_2 \log(y_2) + t_3 \log(y_3))$$

$$E = -\log(y_1)$$



t	1	0	0
y	0.1	0.7	0.2
y	0.4	0.4	0.2
y	0.6	0.3	0.1
y	0.8	0.1	0.1
y	1.0	0.0	0.0

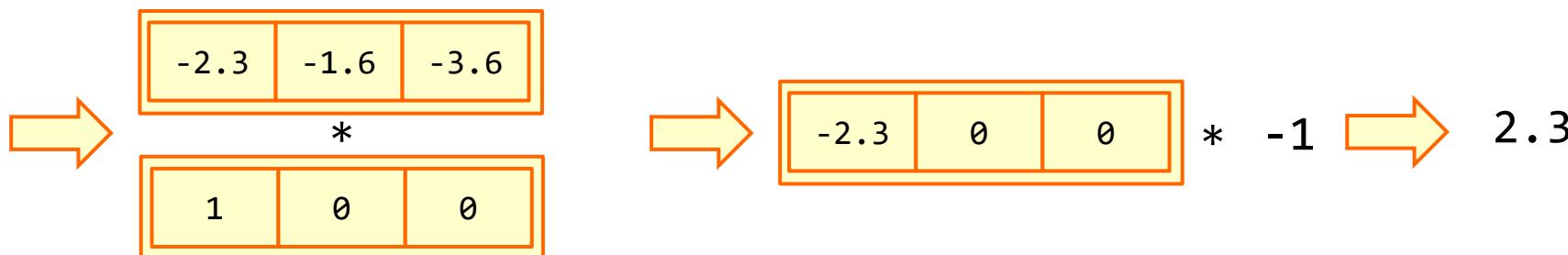
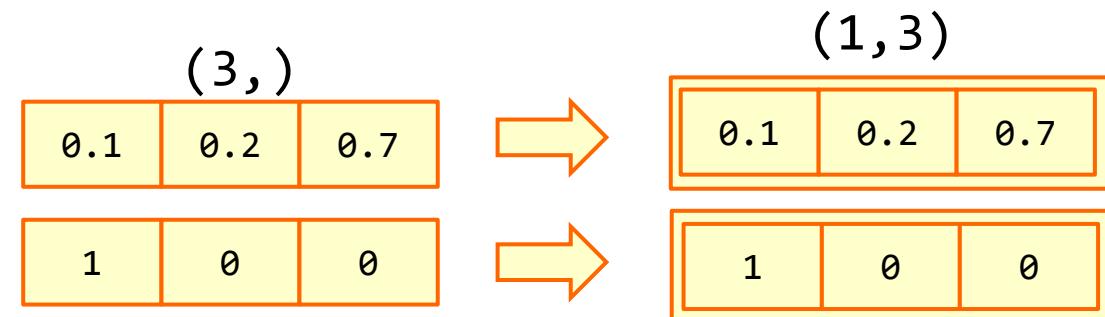
y	0.1	0.7	0.2
t	1	0	0
y-t	-0.9	0.7	0.2

$$w = w - \frac{\partial E}{\partial w}$$

```
def cross_entropy_error(y, t):
    delta = 1e-7
    if y.ndim == 1 :
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)
    batch_size = y.shape[0]
    return -np.sum( t*np.log(y + delta) ) / batch_size
```

$y = [0.1, 0.2, 0.7]$

$t = [1, 0, 0]$



크로스 엔트로피(cross entropy error)

4.2 손실함수

```
def cross_entropy_error(y, t):
    delta = 1e-7
    if y.ndim == 1 :
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)
    batch_size = y.shape[0]
    return -np.sum( t*np.log(y + delta) ) / batch_size
```

(2, 3)

0.1	0.2	0.7
0.2	0.3	0.5

(2, 3)

1	0	0
0	1	0



-2.3	-1.6	-3.6
-1.6	-1.2	-0.69

*

1	0	0
0	1	0



-2.3	0	0
0	-1.2	0

* -1



2.3	0	0
0	1.2	0

$$3.5/2 \Rightarrow 1.75$$

4. 신경망 학습

4.1 데이터에서 학습한다.

4.2 손실함수

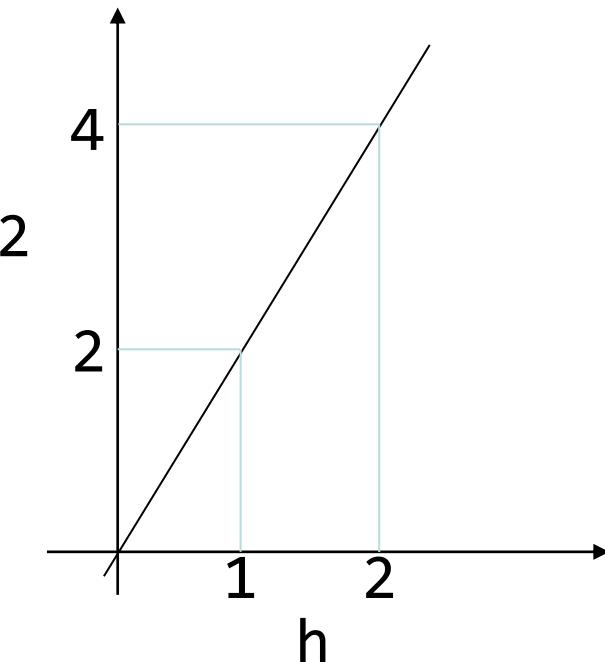
4.3 수치 미분

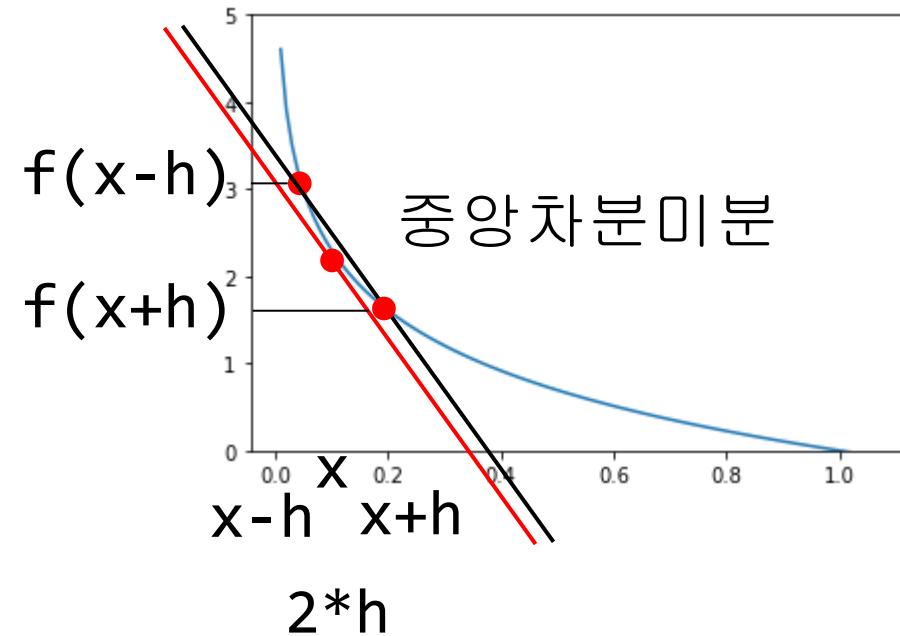
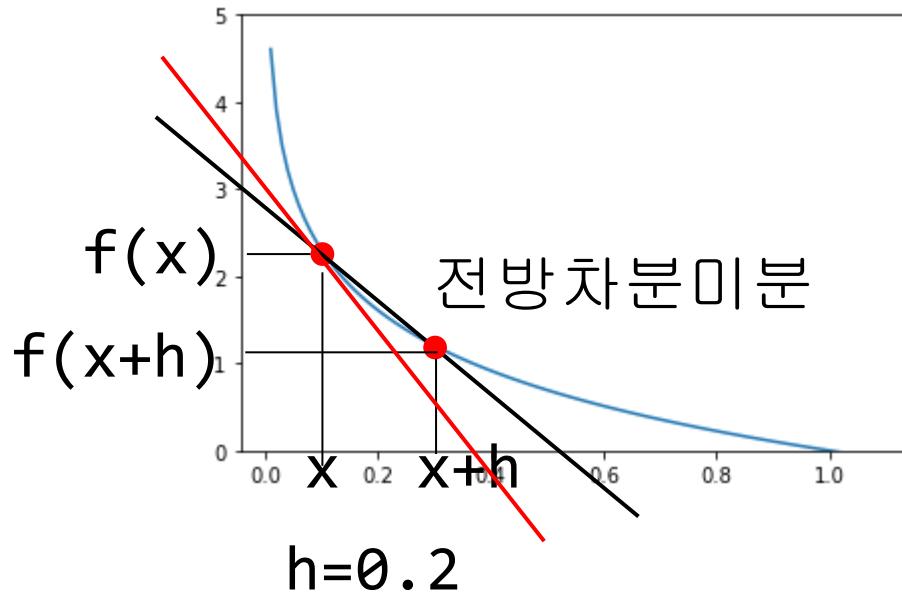
$$\frac{dy}{dx} = \frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$

$$y = 2x$$

$$\frac{dy}{dx} = 2$$

$$\frac{4 - 2}{1} = 2$$





```
def numerical_diff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h) - f(x)) / h
```

```
def numerical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)
```

$$x = 5, 10$$

$$y = 0.01x^2 + 0.1x$$

$$\frac{dy}{dx} = 0.02x + 0.1$$

```
def numerical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)

def function(x):
    return 0.01*x**2 + 0.1*x

numerical_diff(function, 5)
numerical_diff(function, 10)
```

$$x_0 = 3, x_1 = 4$$

$$f(x_0, x_1) = x_0^2 + x_1^2$$

$$\frac{\partial y}{\partial x_0} = 2x_0^1$$

```
def numerical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)

def function_tmp1(x0):
    return x0**2 + 4**2

numerical_diff(function_tmp1, 3)
```

$$x_0 = 3, x_1 = 4$$

$$f(x_0, x_1) = x_0^2 + x_1^2$$

$$\frac{\partial y}{\partial x_1} = 2x_1^1$$

```
def numerical_diff(f, x):
    h = 1e-4
    return (f(x+h) - f(x-h)) / (2*h)

def function_tmp2(x1):
    return 3**2 + x1**2

numerical_diff(function_tmp2, 4)
```

$$x_0 = x_0 - \eta \frac{\partial f(x)}{\partial x_0}$$

$$x_1 = x_1 - \eta \frac{\partial f(x)}{\partial x_1}$$

$$W = \begin{pmatrix} w_{11}^{(1)} & w_{21}^{(1)} & w_{31}^{(1)} \\ w_{12}^{(1)} & w_{22}^{(1)} & w_{32}^{(1)} \end{pmatrix}$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}^{(1)}} & \frac{\partial L}{\partial w_{21}^{(1)}} & \frac{\partial L}{\partial w_{31}^{(1)}} \\ \frac{\partial L}{\partial w_{12}^{(1)}} & \frac{\partial L}{\partial w_{22}^{(1)}} & \frac{\partial L}{\partial w_{32}^{(1)}} \end{pmatrix}$$

5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

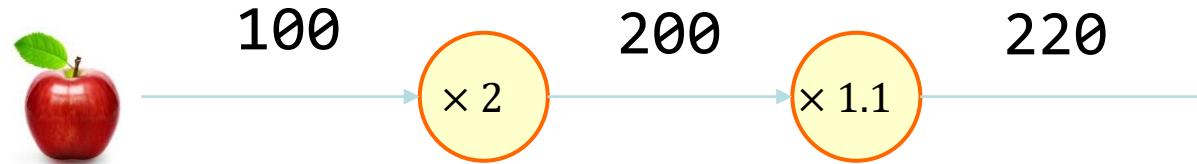
5.3 역전파

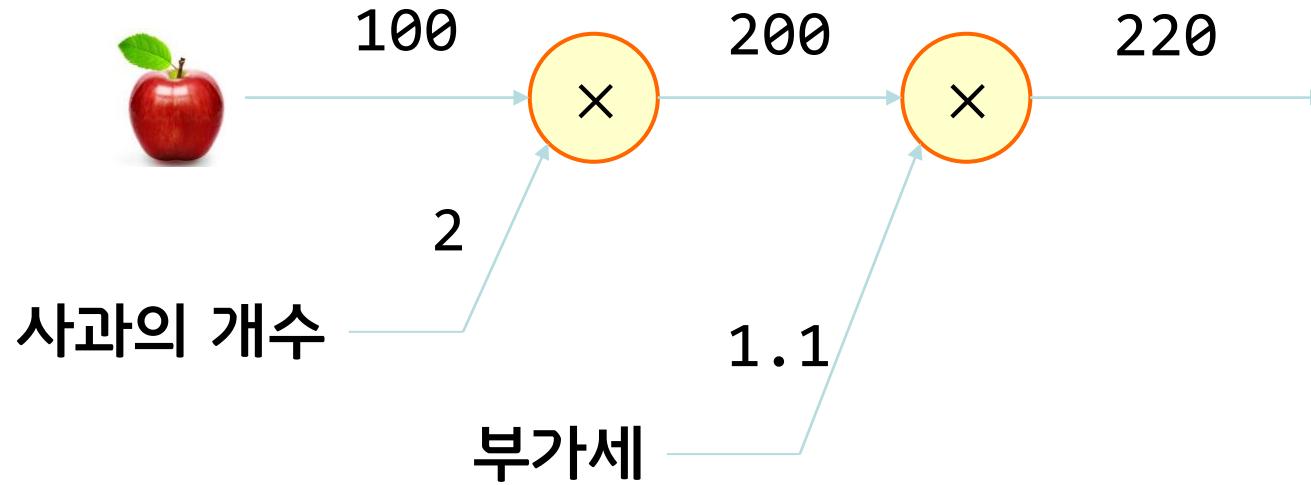
5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기





사과의 개수



2

100

×

$$(100*2 + 150*3)*1.1 = 715$$

귤의 개수



150

×

부가세

3

200

450

1.1

+

650

×

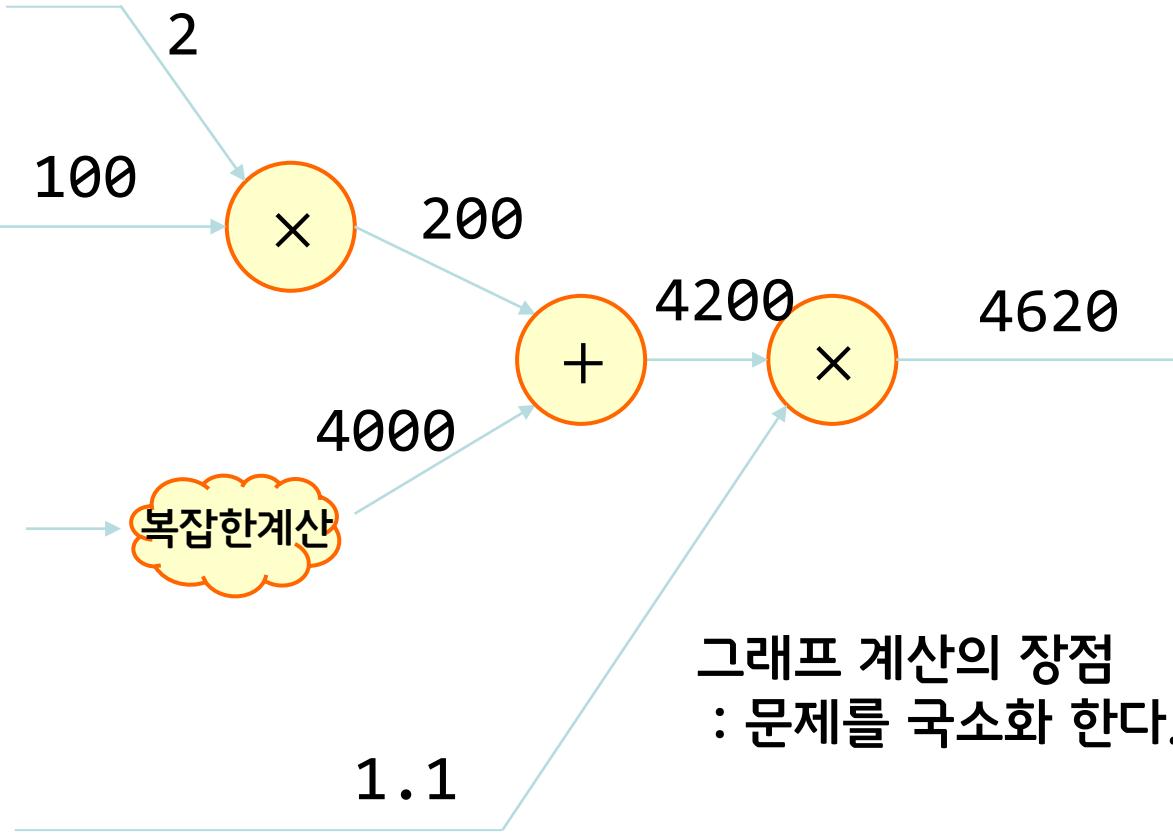
715

1. 계산 그래프를 구성한다.
2. 그래프에서 계산을 왼쪽에서 오른쪽으로 진행한다.

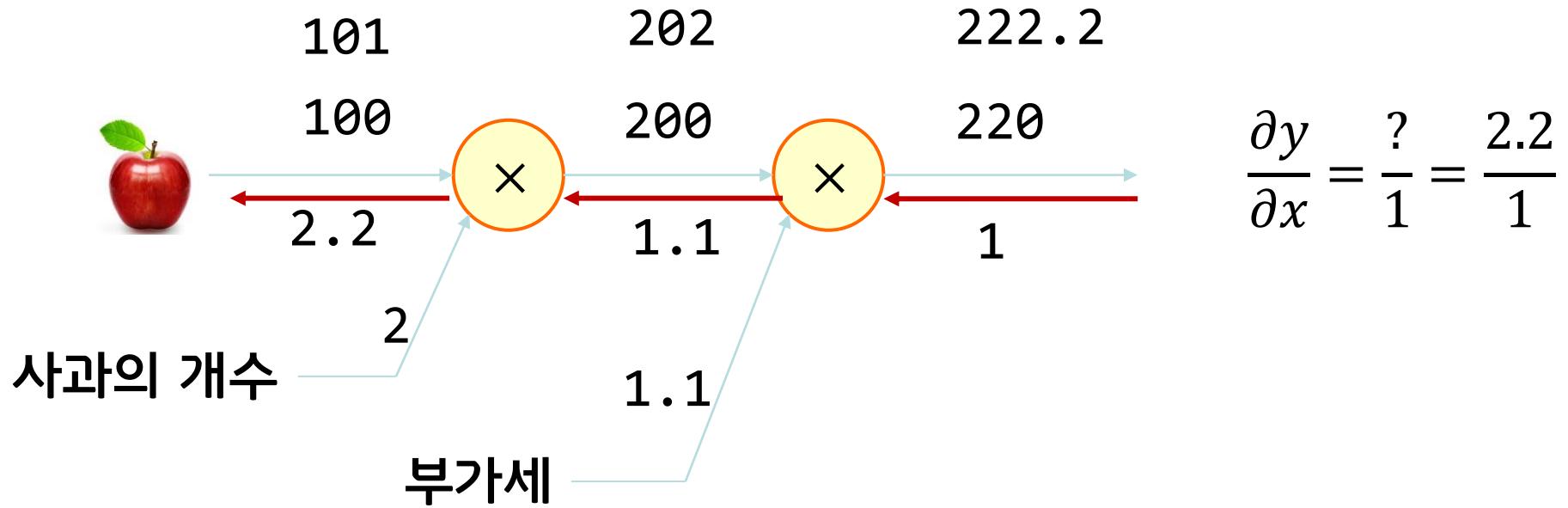
사과의 개수



부가세



그래프 계산의 장점
: 문제를 국소화 한다.



사과 가격이 1원 오를때 전체 가격의 증분 값이
사과에 전달되는 미분 값이다.

5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

5.3 역전파

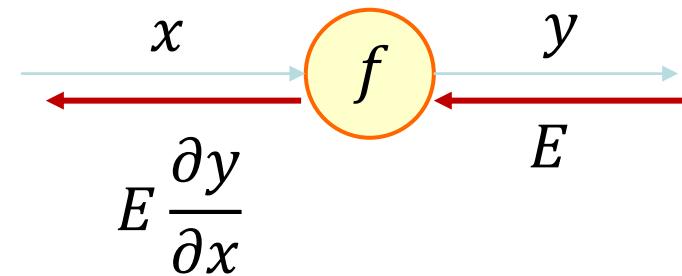
5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기

$$y = f(x)$$



$$z = (x + y)^2$$

$$z = t^2$$

$$\frac{\partial}{\partial x} (x + y)^2$$

$$t = (x + y)$$

$$\frac{\partial}{\partial x} (x^2 + 2xy + y^2)$$

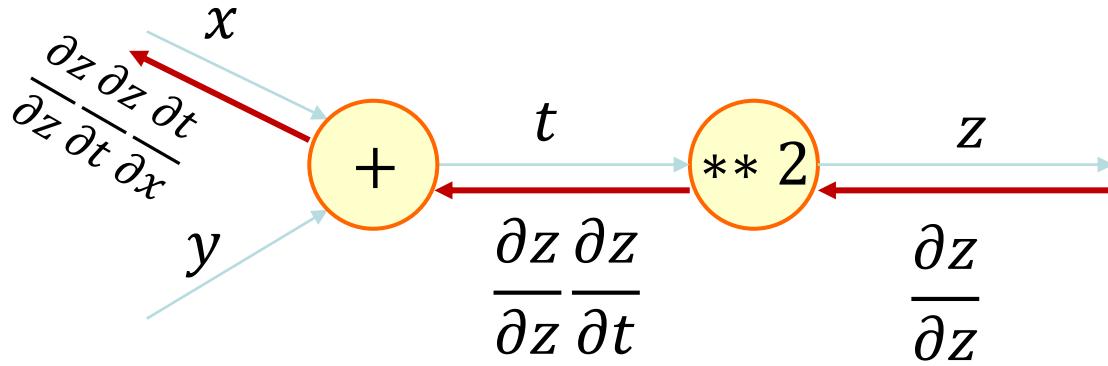
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = 2(x + y)$$

$$(2x + 2y)$$

$$\frac{\partial z}{\partial t} = 2t$$

$$2(x + y)$$

$$\frac{\partial t}{\partial x} = 1$$



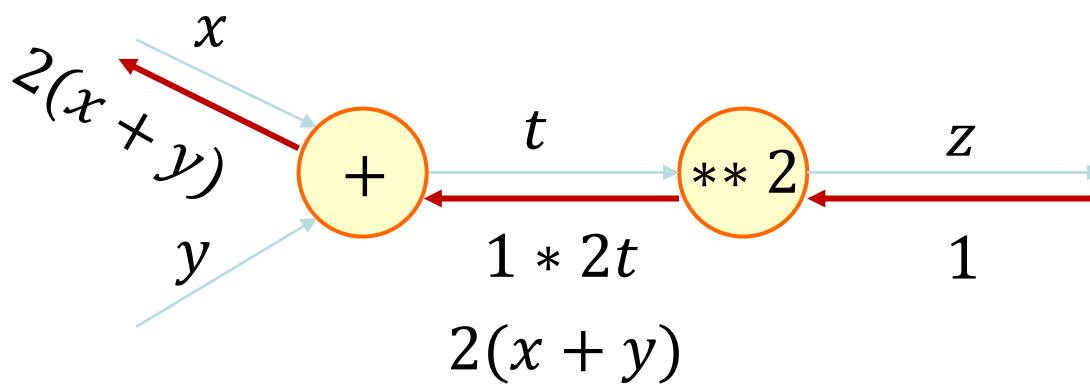
$$z = t^2$$

$$t = (x + y)$$

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} \frac{\partial t}{\partial x} = 2(x + y)$$

$$\frac{\partial z}{\partial t} = 2t$$

$$\frac{\partial t}{\partial x} = 1$$



5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

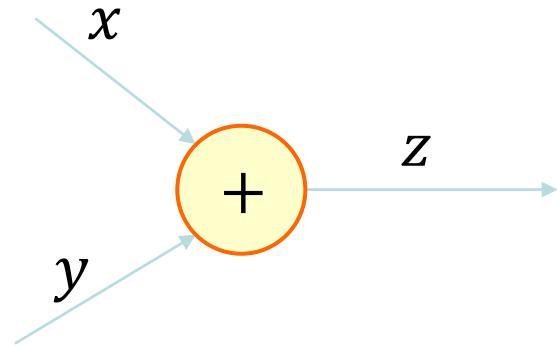
5.3 역전파

5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

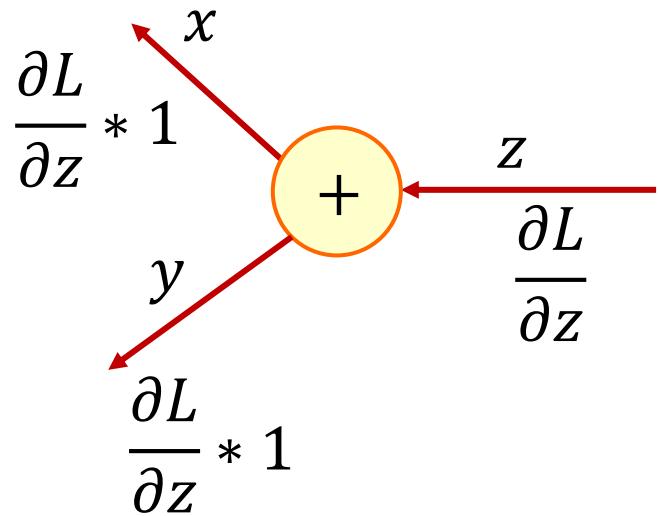
5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기

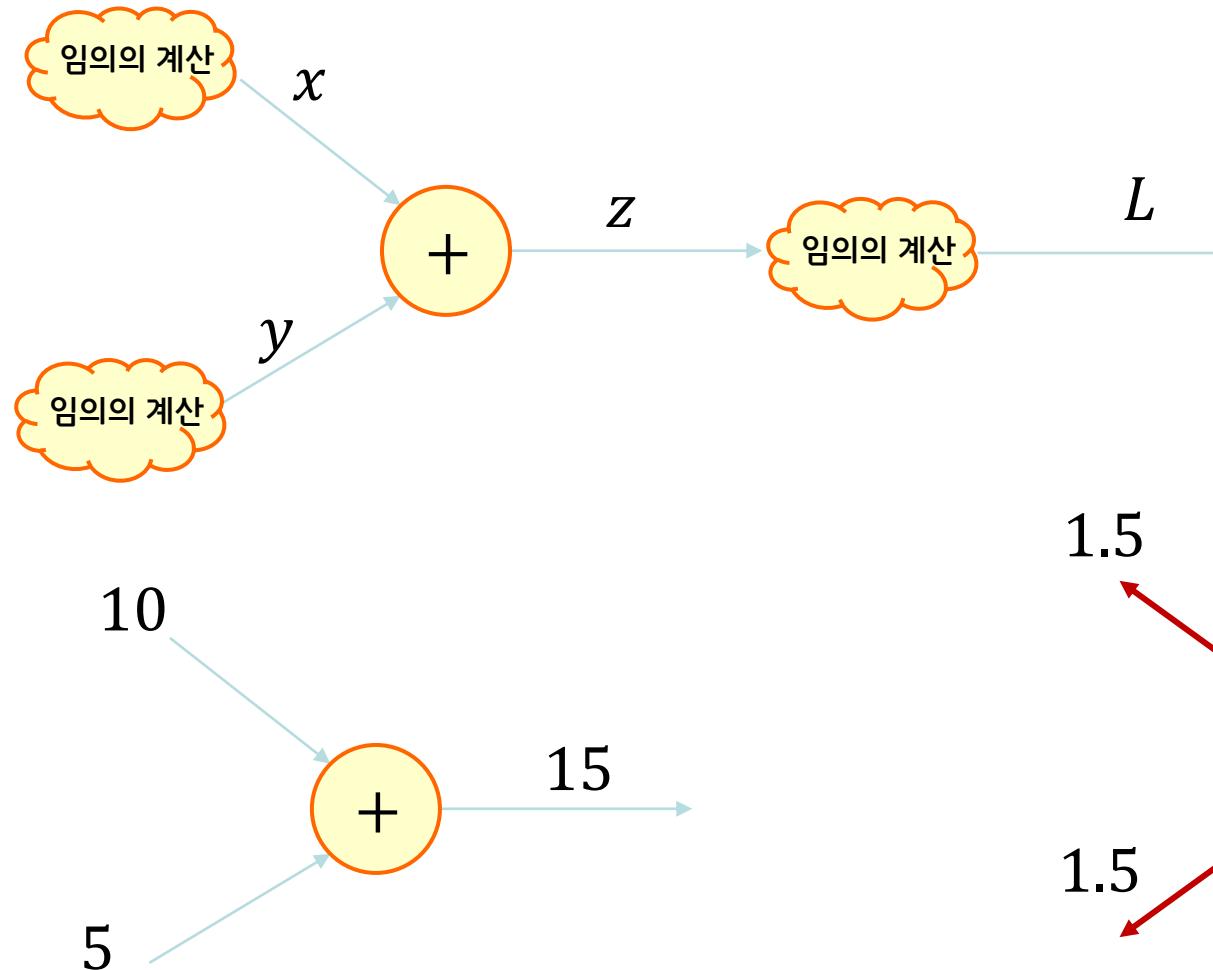


$$z = x + y$$

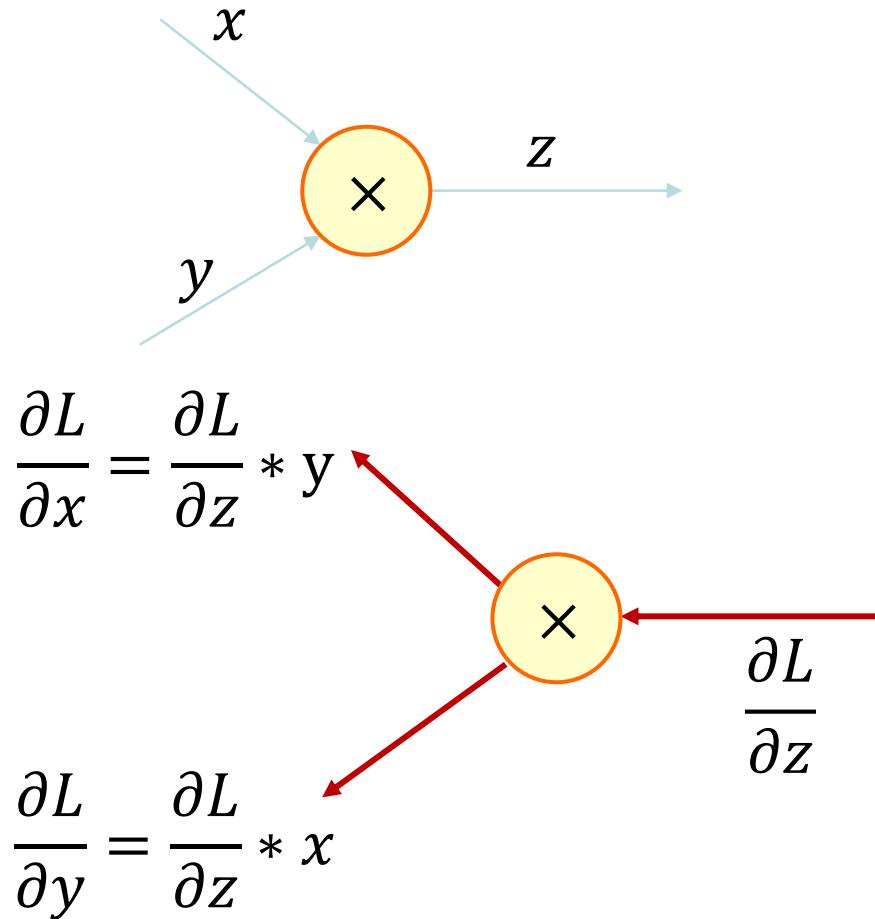
$$\frac{\partial z}{\partial x} = 1$$



$$\frac{\partial z}{\partial y} = 1$$



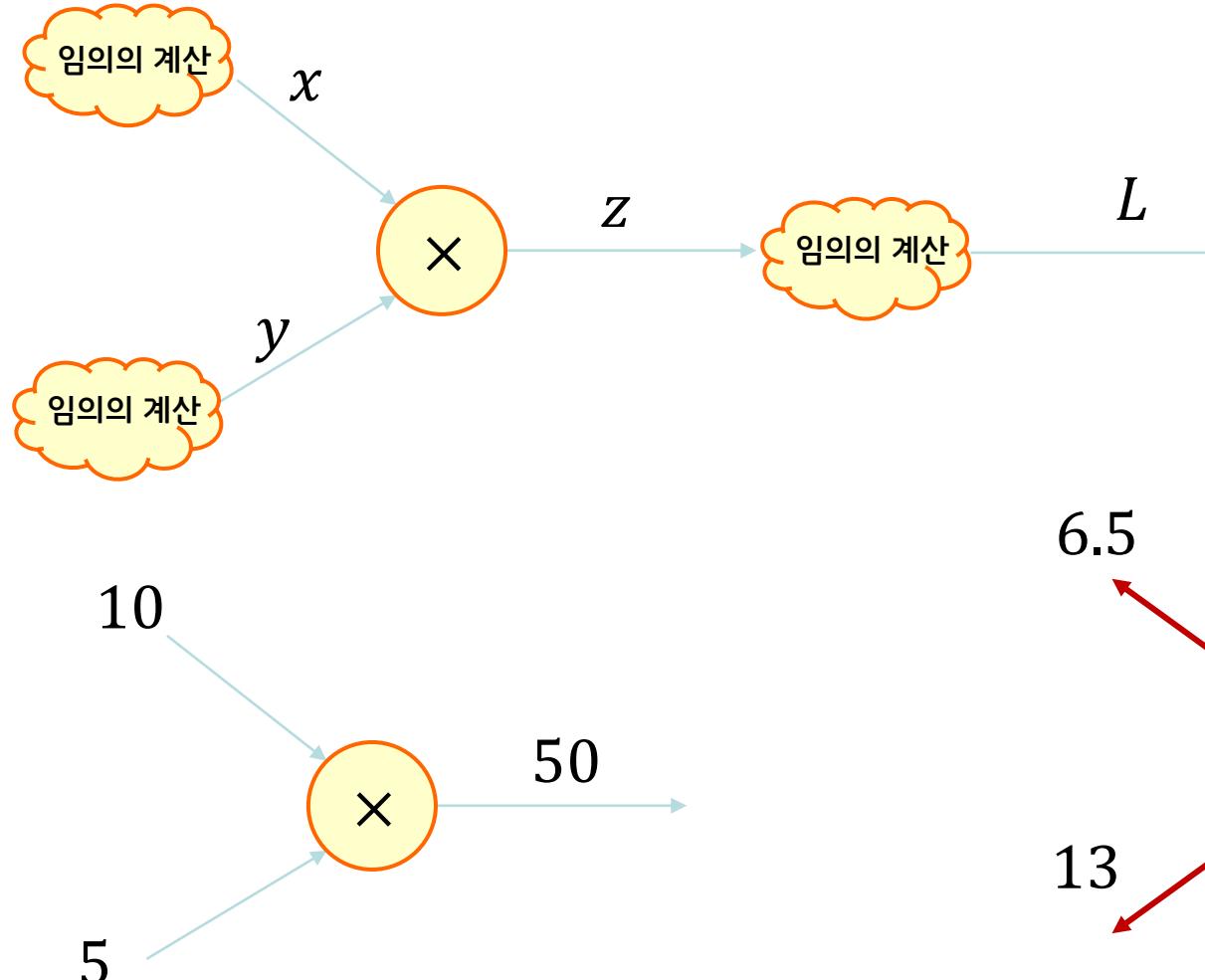
덧셈 노드의 미분값은 리피트이다.



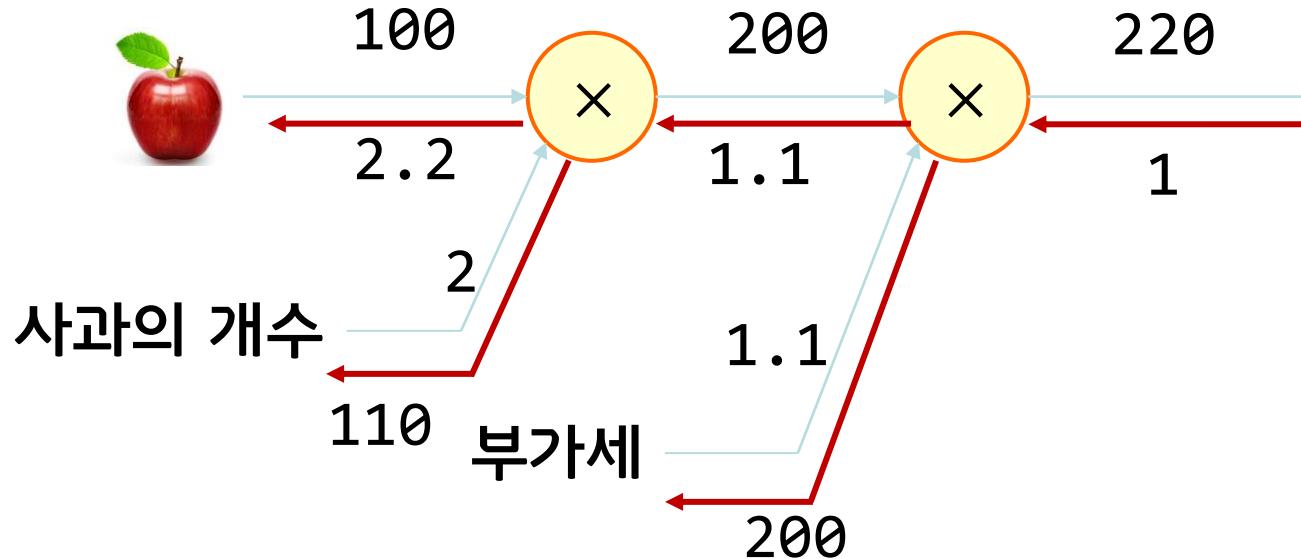
$$z = x \times y$$

$$\frac{\partial z}{\partial x} = y$$

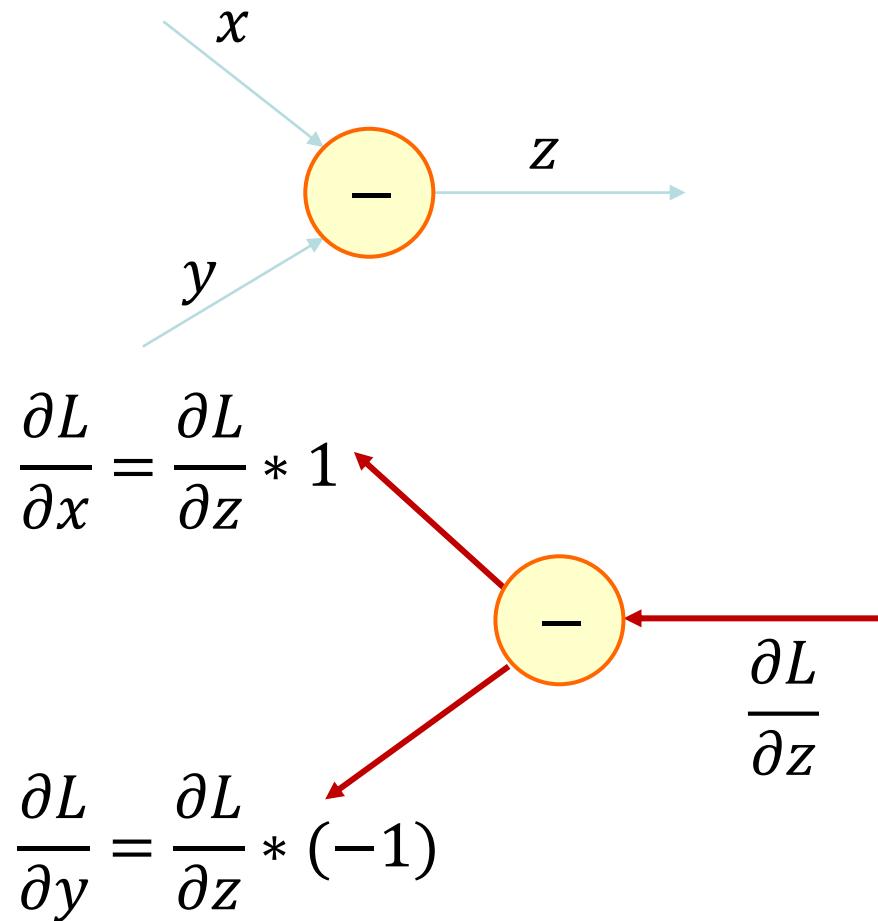
$$\frac{\partial z}{\partial y} = x$$



곱셈 노드의 뒤에서 온 미분값에
정방향 연산시 반대쪽 값을 곱한다.



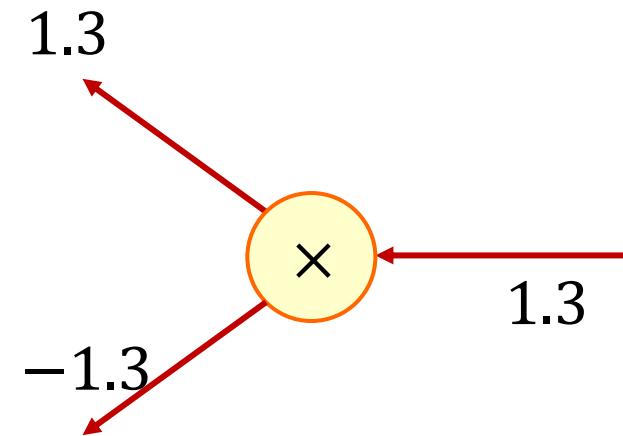
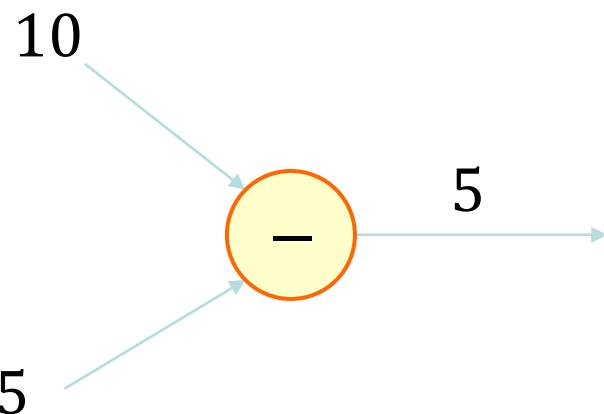
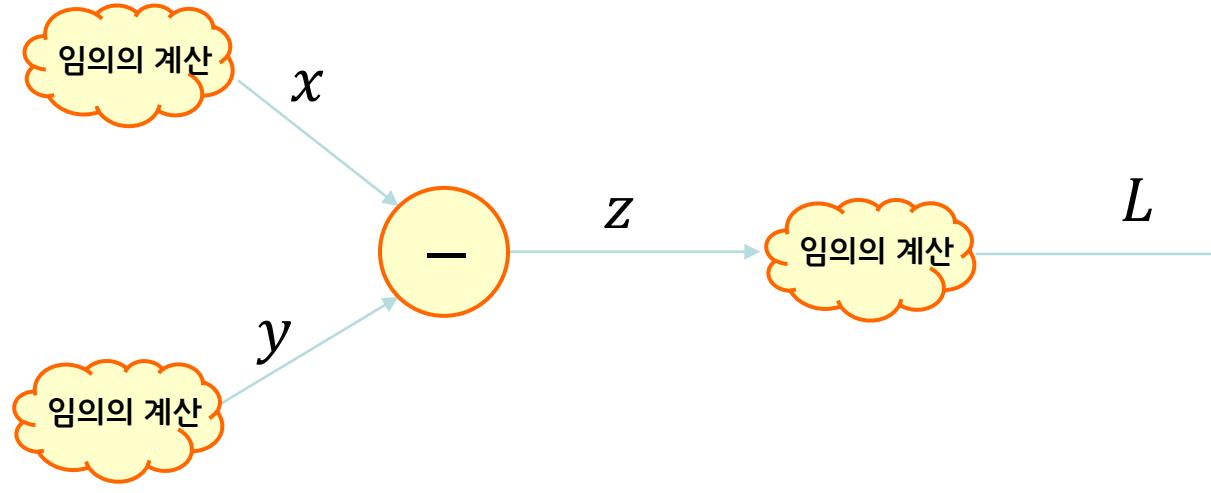
곱셈 노드의 뒤에서 온 미분값에
정방향 연산시 반대쪽 값을 곱한다.



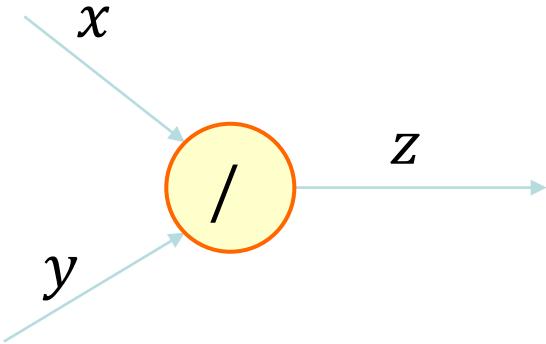
$$z = x - y$$

$$\frac{\partial z}{\partial x} = 1$$

$$\frac{\partial z}{\partial y} = -1$$



x 쪽에는 미분값을 그대로 전달하고
 y 쪽에는 미분값에 -1 을 곱하여 전달한다.



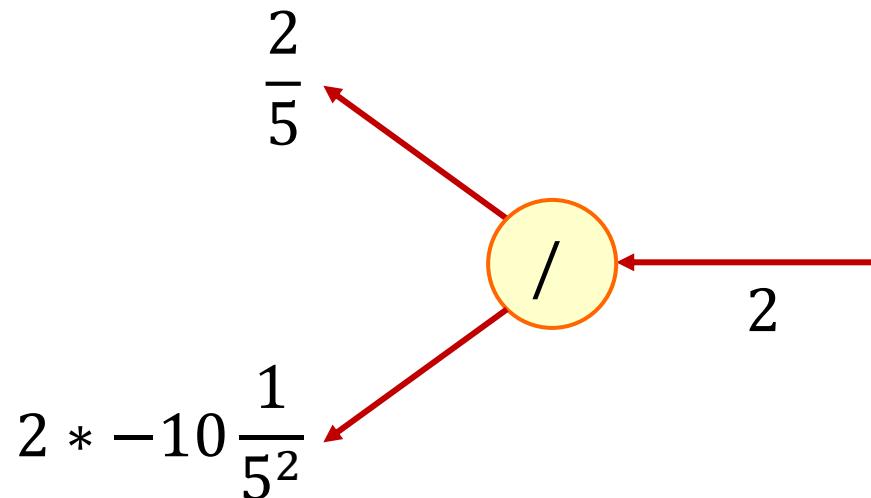
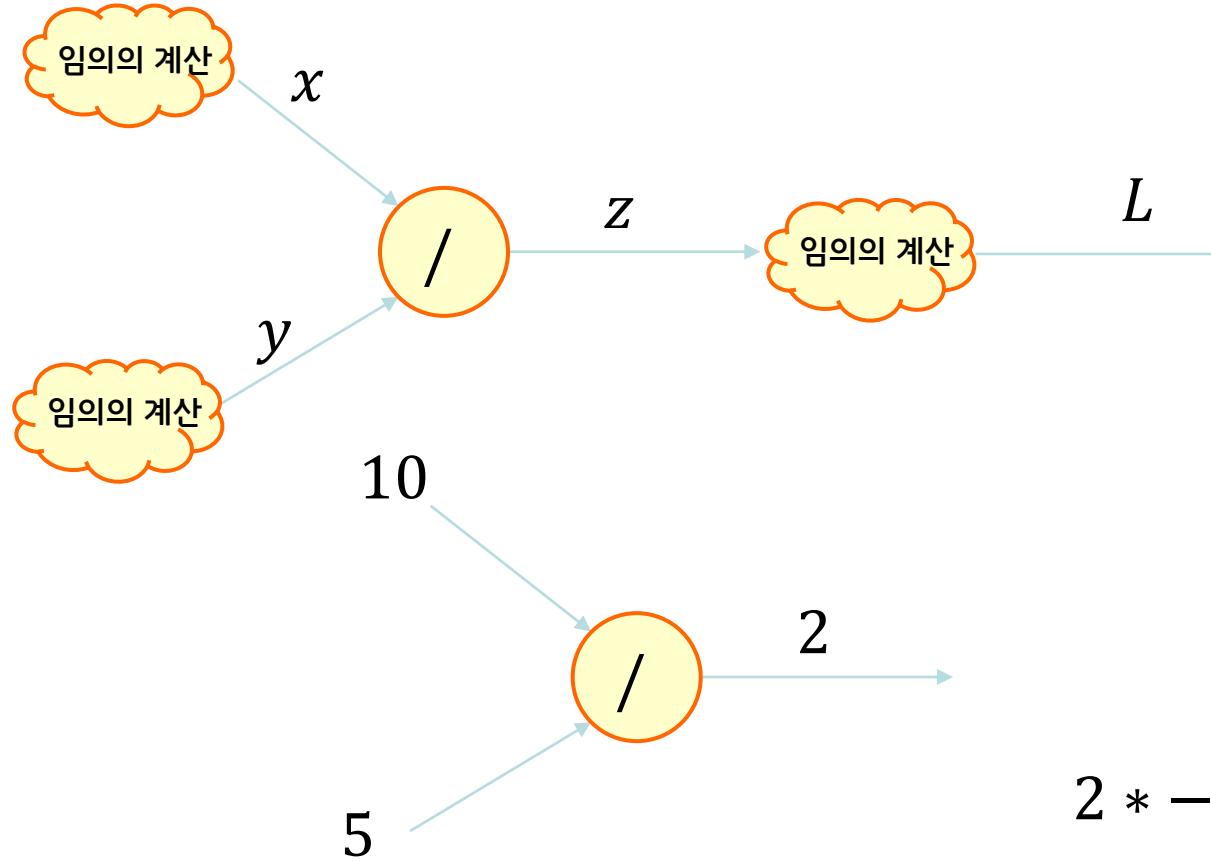
$$z = x * \frac{1}{y} \quad z = x * y^{-1}$$

$$\frac{\partial z}{\partial x} = \frac{1}{y}$$

$$\frac{\partial z}{\partial y} = -x \frac{1}{y^2}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z} * \frac{1}{y}$$

$$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z} * -x \frac{1}{y^2}$$



x 쪽에는 미분값 $\frac{1}{y}$ 를 곱하고

y 쪽에는 미분값에 $-x * \frac{1}{y^2}$ 을 곱하여 전달한다.

사과의 개수



귤의 개수



부가세



5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

5.3 역전파

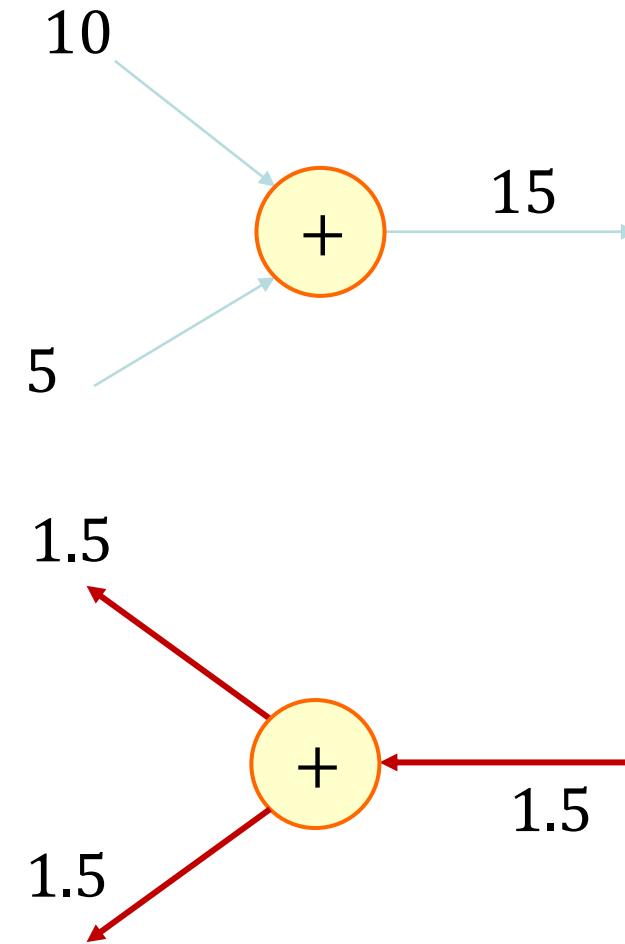
5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

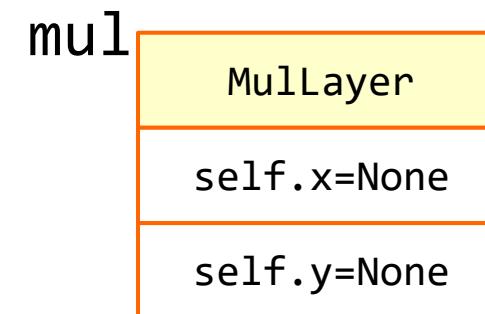
5.7 오차역전파법 구현하기

```
class AddLayer:  
    def __init__(self):  
        pass  
  
    def forward(self, x, y):  
        out = x + y  
  
        return out  
  
    def backward(self, dout):  
        dx = dout * 1  
        dy = dout * 1  
  
        return dx, dy
```

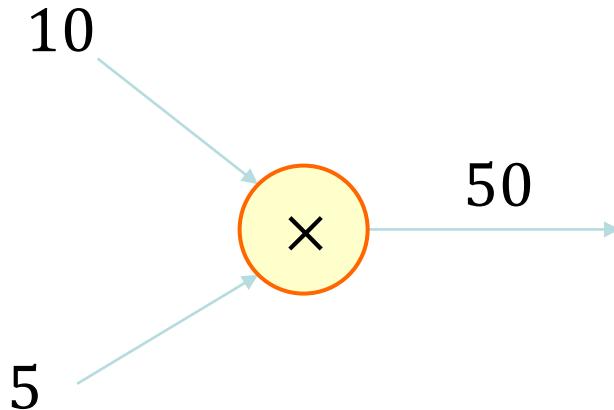
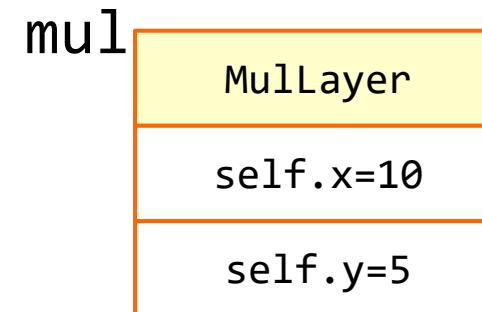


```
class MulLayer:  
    def __init__(self):  
        self.x = None  
        self.y = None  
  
    def forward(self, x, y):  
        self.x = x  
        self.y = y  
        out = x * y  
  
        return out  
  
    def backward(self, dout):  
        dx = dout * self.y # x와 y를 바꾼다.  
        dy = dout * self.x  
  
        return dx, dy
```

```
mul = MulLayer()
```



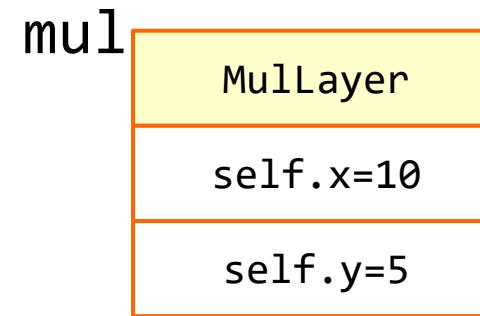
```
mul = MulLayer()  
  
out = mul.forward(10, 5)  
print(out)
```



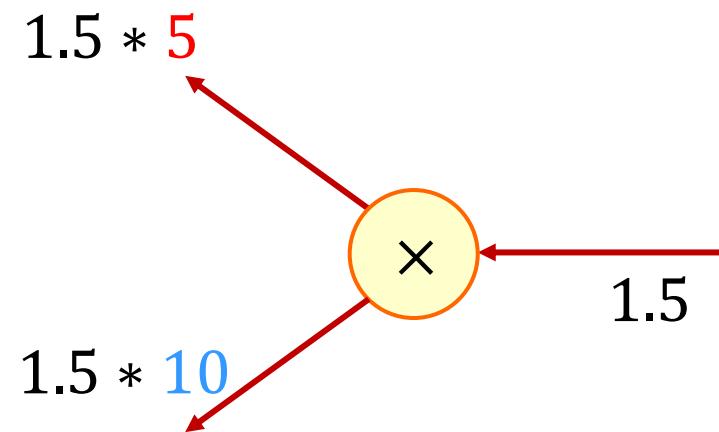
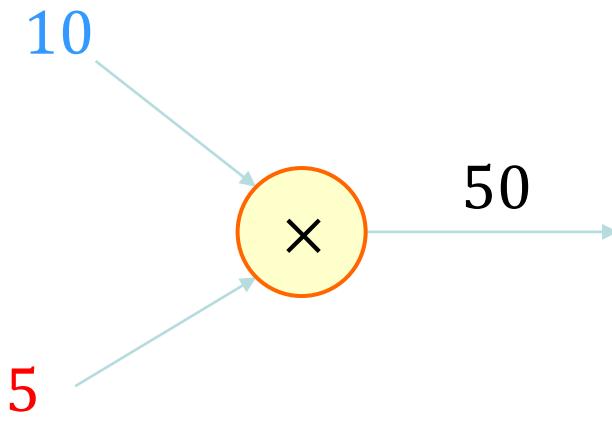
곱연산의 정방향 연산에서는 forward의 인자로 전달된 x,y를 객체안에 저장하는게 중요하다.
이유는 미분시 필요하기 때문이다.

```
mul = MulLayer()

dx, dy = mul.backward(1.5)
print("dx=", dx)
print("dy=", dy)
```



곱연산의 역방향 연산에서는 forward의 인자로 전달된 x, y 를 이용하여, dx 에는 뒤에서 온 미분값인 $dout$ 과 정방향 연산시 반대편 입력인 `self.y`를 곱하여 대입하고 dy 에는 뒤에서 온 미분값인 $dout$ 과 정방향 연산시 반대편 입력인 `self.x`를 곱하여 대입한다.



```

from layer_naive import *

apple = 100
apple_num = 2
tax = 1.1

mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

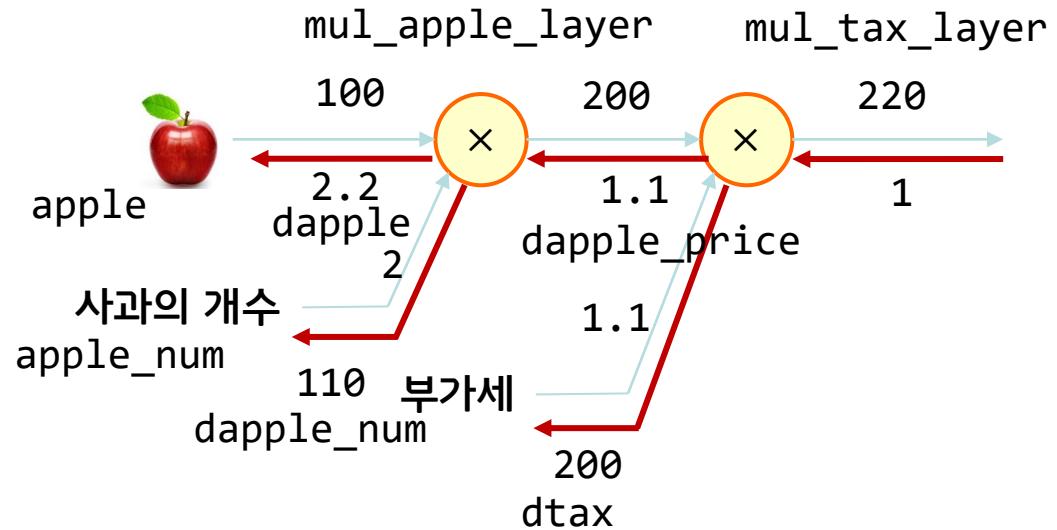
```

```

# forward
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

# backward
dprice = 1
dapple_price, dtax= mul_tax_layer.backward(dprice)
dapple, dapple_num = mul_apple_layer.backward(dapple_price)

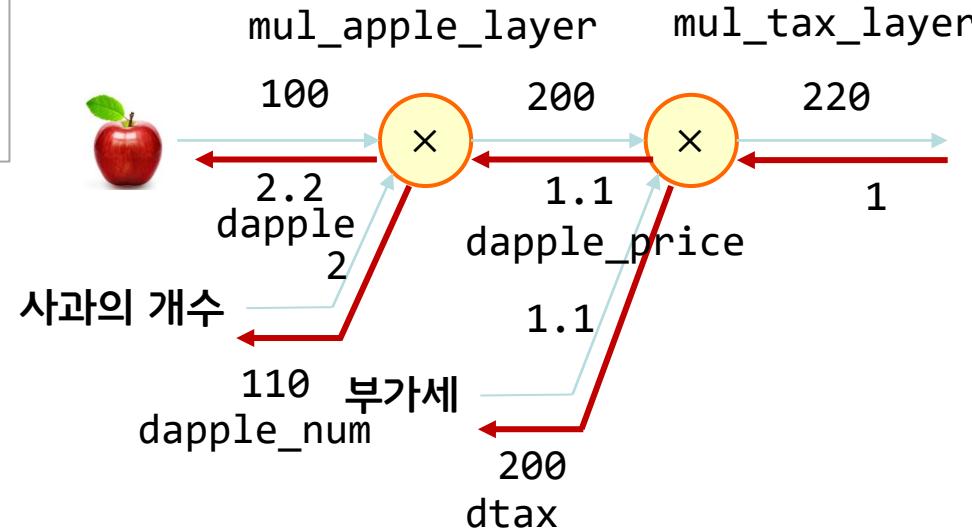
```



사과 노드의 구현

5.4 단순한 계층 구현하기

```
print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dTax:", dtax)
```

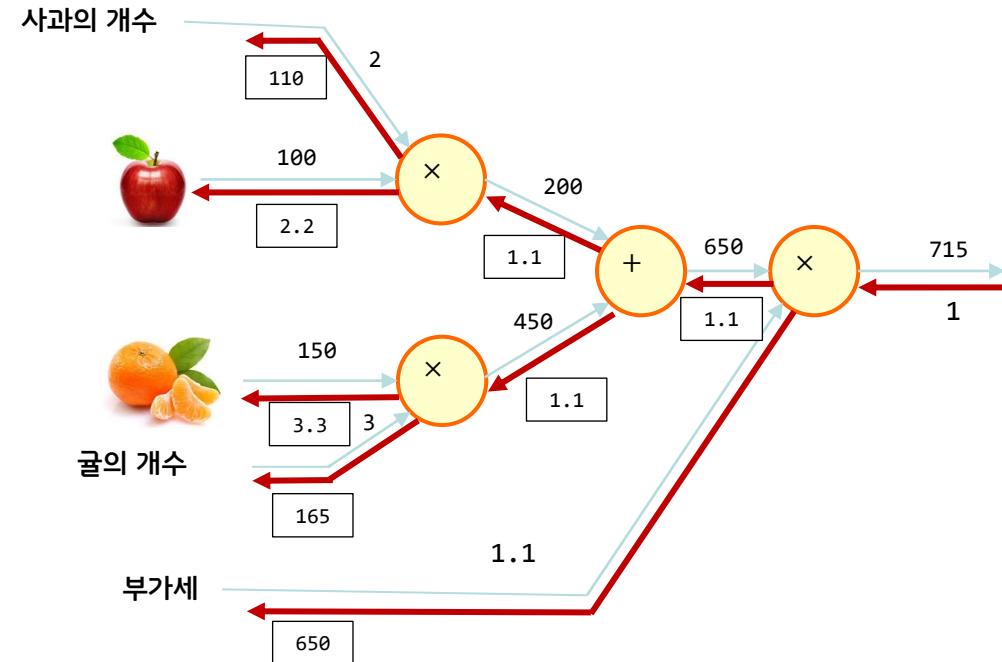


```
from layer_naive import *
```

```
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1
```

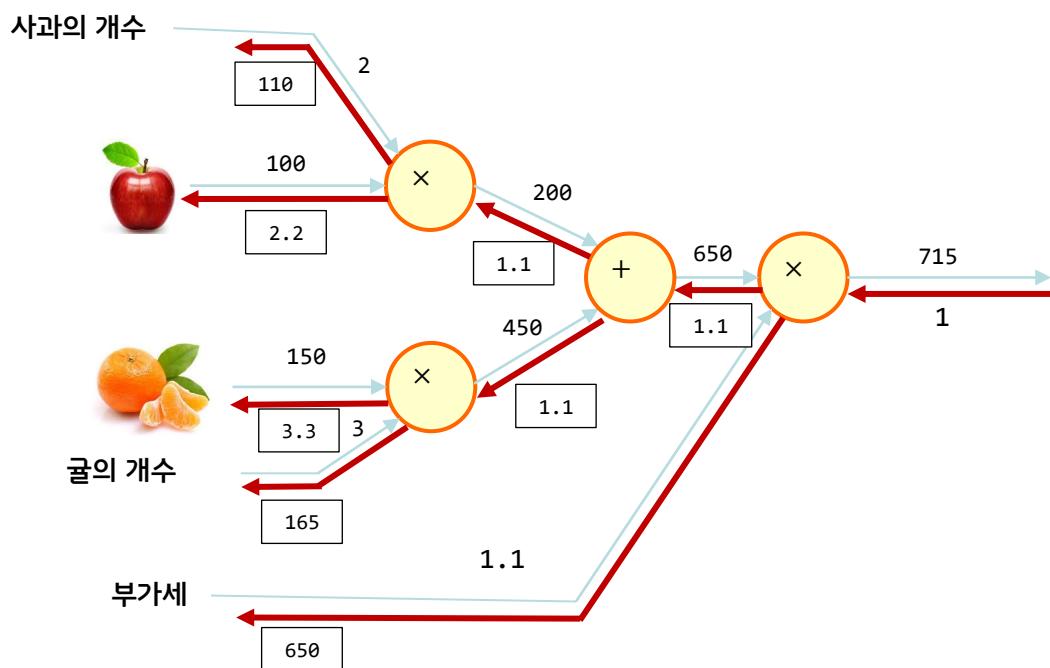
```
# layer
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()
```

```
# forward
apple_price = mul_apple_layer.forward(apple, apple_num) # (1)
orange_price = mul_orange_layer.forward(orange, orange_num) # (2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price) # (3)
price = mul_tax_layer.forward(all_price, tax) # (4)
```



```
# backward
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice) # (4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price) # (3)
dorange, dorange_num = mul_orange_layer.backward(dorange_price) # (2)
dapple, dapple_num = mul_apple_layer.backward(dapple_price) # (1)

print("price:", int(price))
print("dApple:", dapple)
print("dApple_num:", int(dapple_num))
print("dOrange:", dorange)
print("dOrange_num:", int(dorange_num))
print("dTax:", dtax)
```



5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

5.3 역전파

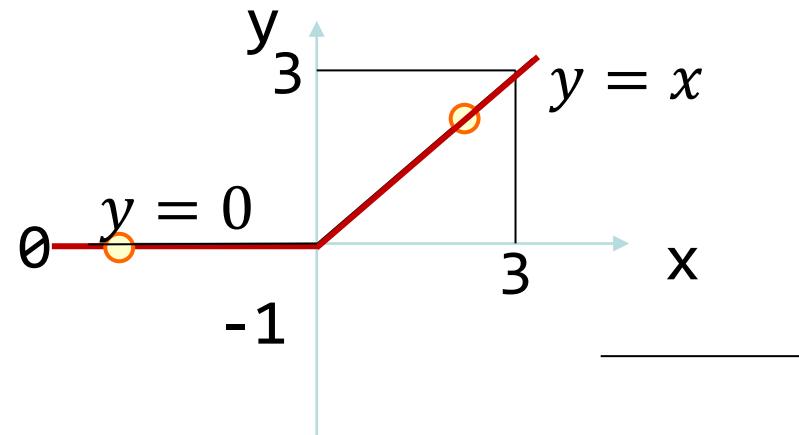
5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기

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

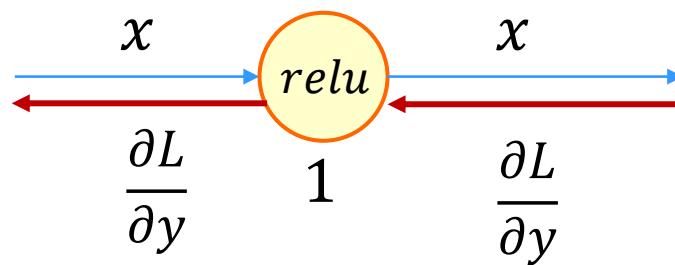


$$\frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

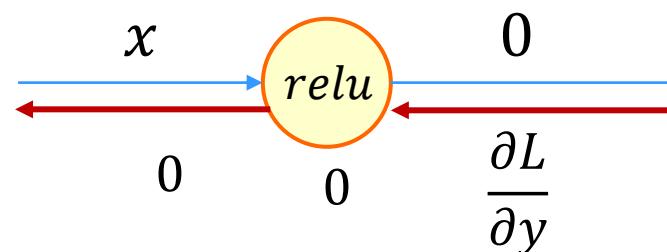
$$\begin{bmatrix} 1, -2, 3, -4, 5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1, 0, 3, 0, 5 \end{bmatrix}$$

$$\begin{bmatrix} -1, 0, -1, 0, -1 \end{bmatrix} \leq \begin{bmatrix} -1, 1, -1, 1, -1 \end{bmatrix}$$

$$x > 0$$



$$x \leq 0$$



```

class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x<=0)
        print(self.mask)
        out = x.copy()
        out[self.mask] = 0
        return out

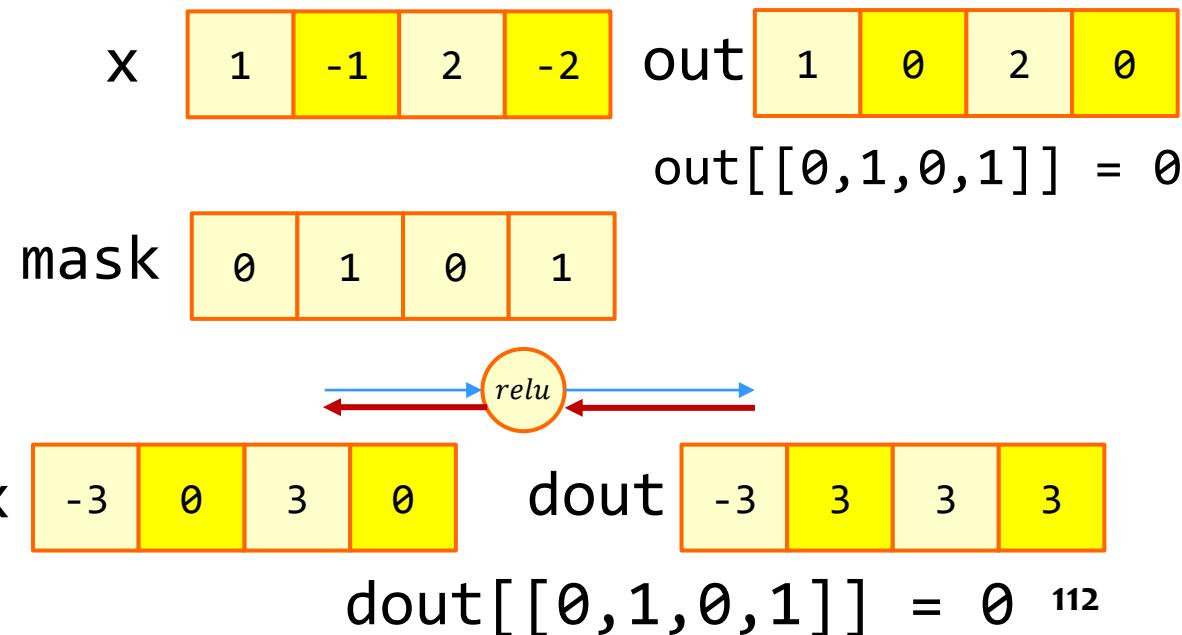
    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout;
        return dx

```

```

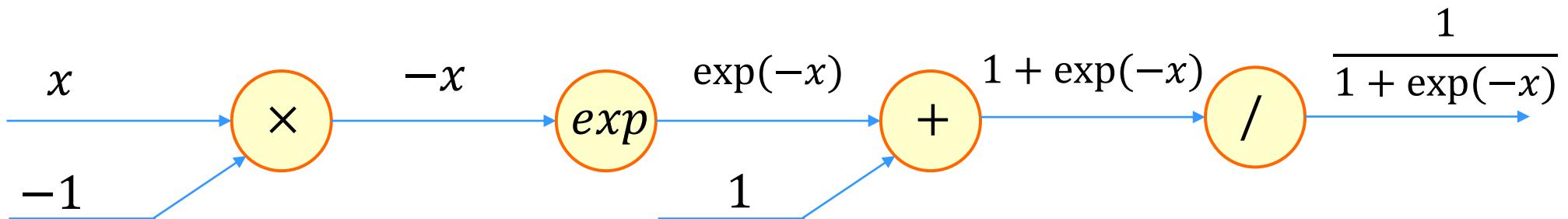
import numpy as np
r1 = Relu()
x = np.array([1,-1,2,-2])
y = r1.forward(x)
print(y)
dx = r1.backward(np.array([3,3,3,3]))
print(dx)

```

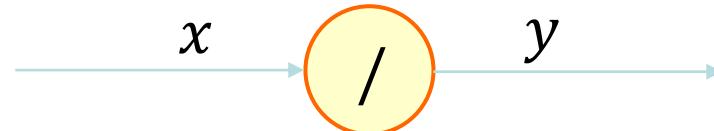


$$y = \frac{1}{1 + \exp(-x)}$$

$$y * (1 - y)$$



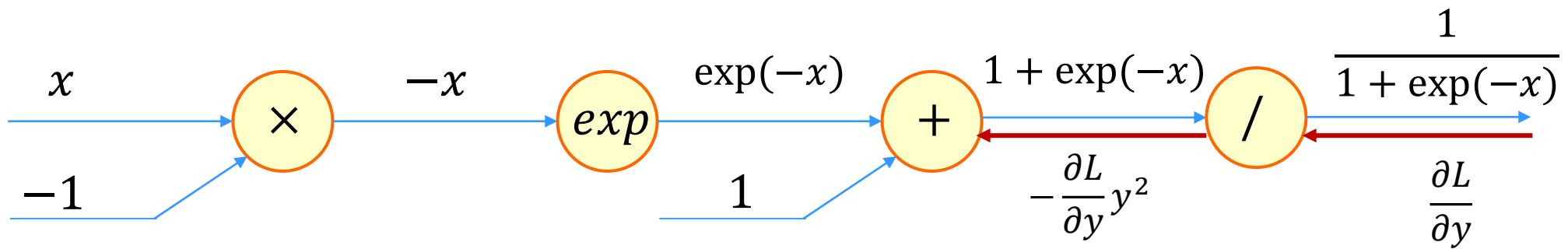
1 단계

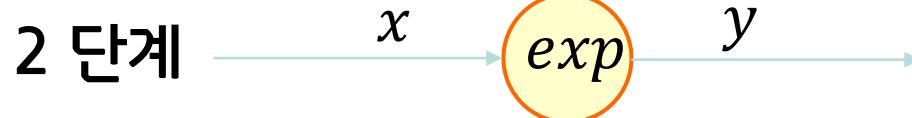


$$y = \frac{1}{1 + \exp(-x)}$$

$$y = \frac{1}{x}$$

$$\frac{\partial y}{\partial x} = -\frac{1}{x^2} = -y^2$$



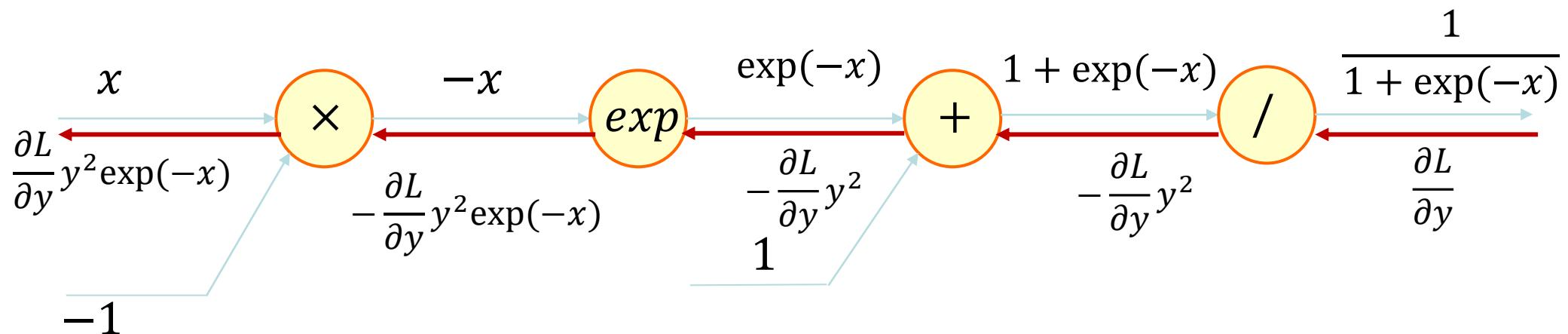


$$y = \frac{1}{1 + \exp(-x)}$$

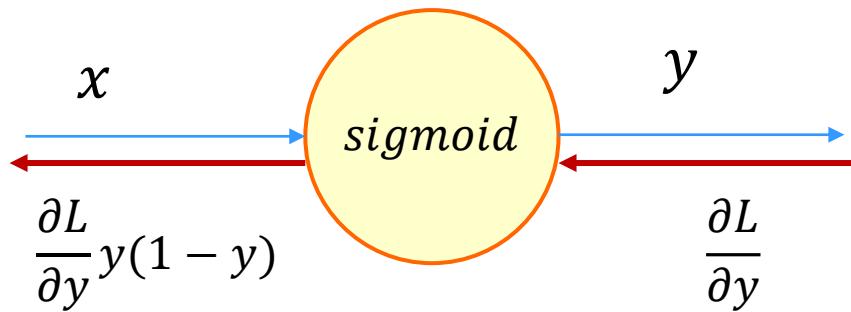
$$y = \exp(x)$$

$$\frac{\partial y}{\partial x} = \exp(x)$$

$$y^*(1-y)$$



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



$$y^2 \exp(-x)$$

$$= \frac{1}{(1 + \exp(-x))^2} \exp(-x)$$

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

$$= y(1 - y)$$

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

Sigmoid 계층

5.5 활성화 함수 계층 구현하기

```
class Sigmoid:  
    def __init__(self):  
        self.out = None  
  
    def forward(self, x):  
        out = 1/(1+np.exp(-x))  
        self.out = out  
        return out  
  
    def backward(self, dout):  
        dx = dout*self.out*(1.0-self.out);  
        return dx
```

```
s1 = Sigmoid()  
x = np.array([-2,-1,1,2])  
y = s1.forward(x)  
print(y)  
dx = s1.backward(np.array([30,30,30,30]))  
print(dx)
```

5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

5.3 역전파

5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기

```
X = np.random.rand(2)
W = np.random.rand(2,3)
B = np.random.rand(3)
print(X.shape)
print(W.shape)
print(B.shape)

Y = np.dot(X, W) + B
Y
```

$$X \cdot W + B = Y$$
$$(2,) \quad (2,3) \quad (3,) \quad (3,)$$

Affine 계층

5.6 Affine/Softmax 계층 구현하기

```
X = np.random.rand(5,2)
W = np.random.rand(2,3)
B = np.random.rand(3)
print(X.shape)
print(W.shape)
print(B.shape)

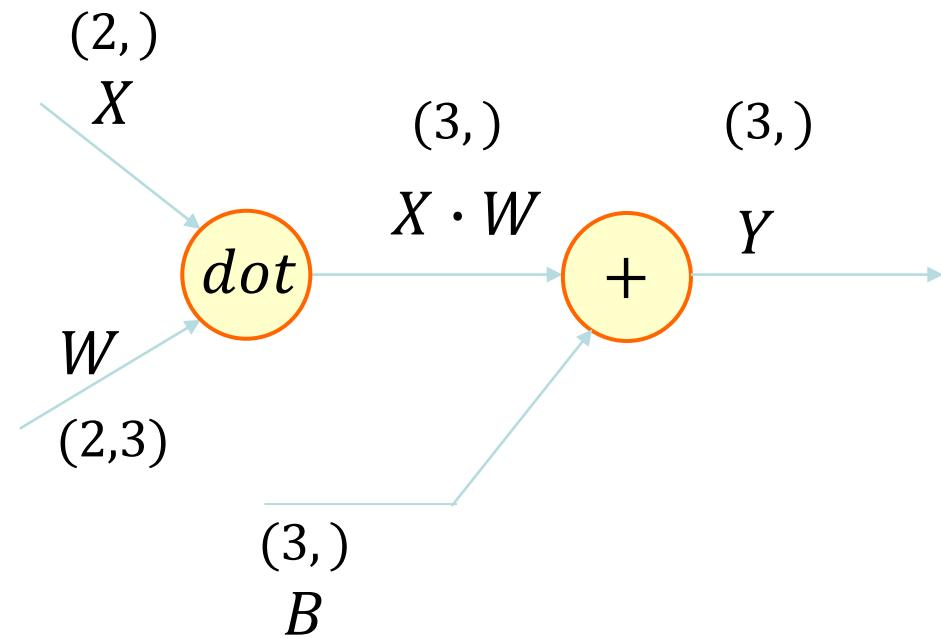
Y = np.dot(X, W) + B
Y
```

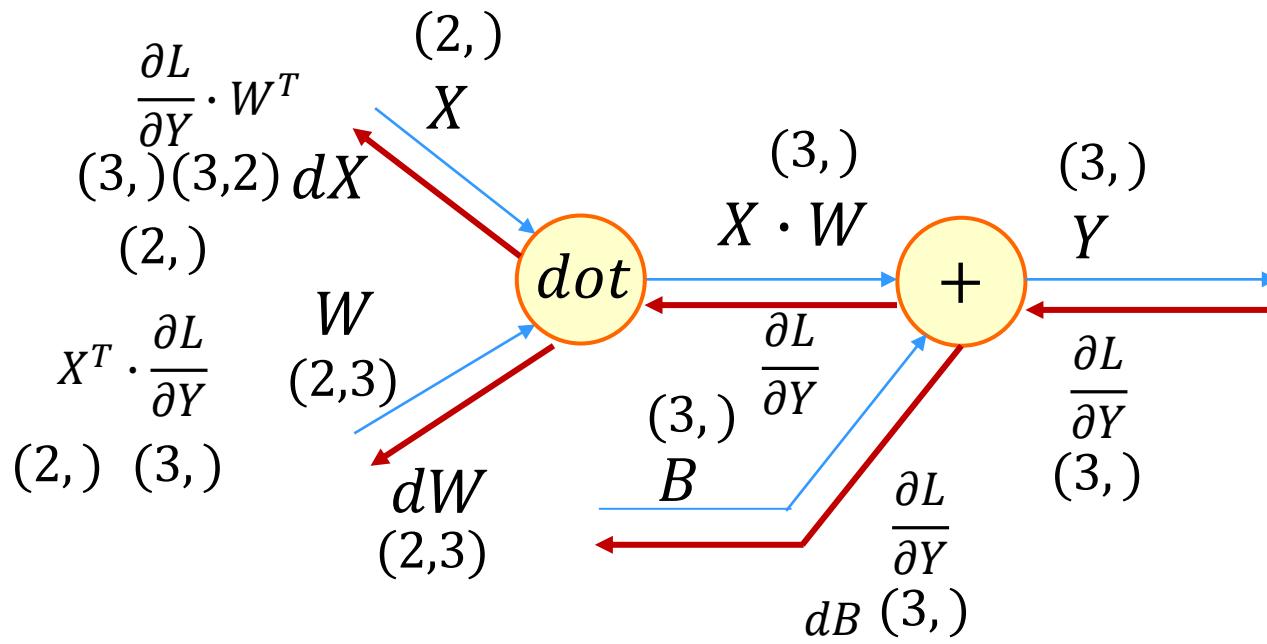
$$X \cdot W + B = Y$$

$(5, 2)$ $(2, 3)$ $(3,)$ $(5, 3)$

$(5, 3) + (3,)$

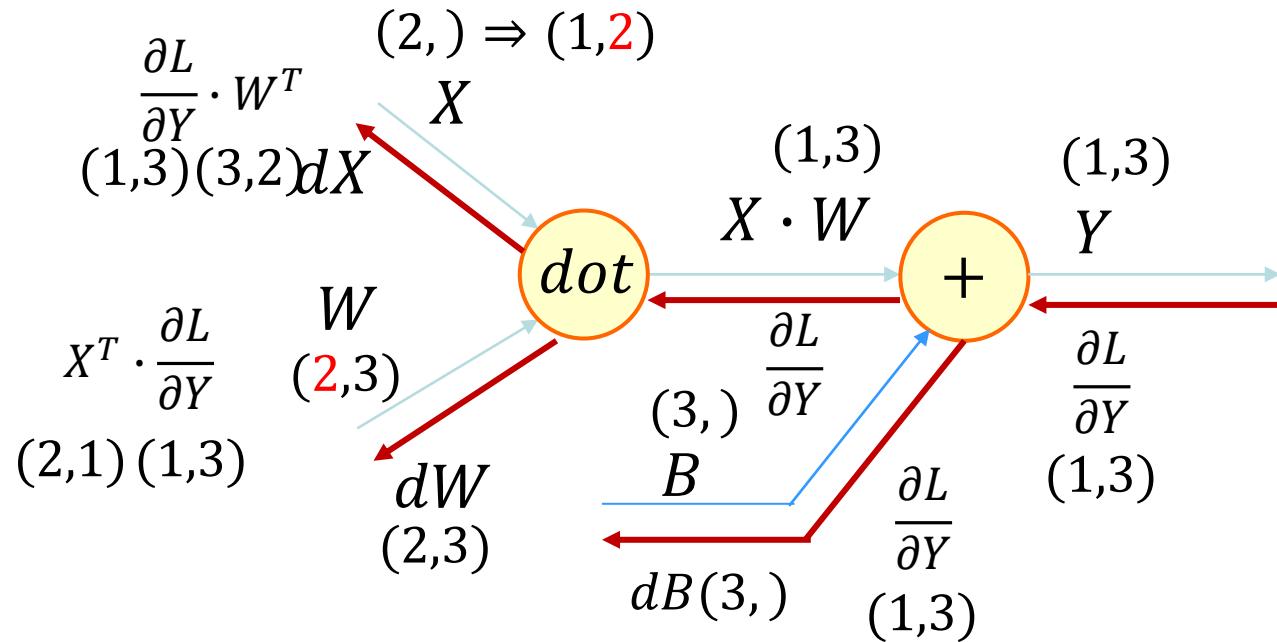
$(5, 3) + (5, 3)$





$$\begin{aligned}
 & X \cdot W \\
 \frac{\partial L}{\partial X} &= \frac{\partial L}{\partial Y} \cdot W^T \\
 & \quad (3,) \quad (3,2) \quad = (2,) \\
 & X \cdot W \\
 \frac{\partial L}{\partial W} &= X^T \cdot \frac{\partial L}{\partial Y} \\
 & \quad (2,) \quad (3,)
 \end{aligned}$$

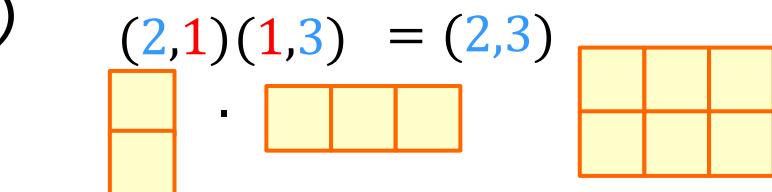
행렬곱의 미분은 편미분으로
사라진 변수 자리에 뒤에서 온
미분값을 쓰고 남은 행렬은 전치하여
행렬곱을 해준다.



$$\begin{aligned} & X \cdot W \\ \frac{\partial L}{\partial X} &= \frac{\partial L}{\partial Y} \cdot W^T \\ (1,3)(3,2) &= (1,2) \\ \frac{\partial L}{\partial W} &= X^T \cdot \frac{\partial L}{\partial Y} \\ (2,1)(1,3) &= (2,3) \end{aligned}$$

`db=np.sum(dout, axis=0)`

행렬곱의 미분은 편미분으로
사라진 변수 자리에 뒤에서 온
미분값을 쓰고 남은 행렬은 전치하여
행렬곱을 해준다.



$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial Y}$$

python의 broadcast 연산

```
a = np.array([1,2,3])
b = np.array([4,5,6])
c = a+b
print(c)
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 4 & 5 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 5 & 7 & 9 \\ \hline \end{array}$$

python의 broadcast 연산

```
a = np.array([1,2,3])
b = np.array([4,5])
c = a+b
print(c)
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 4 & 5 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 5 & 7 & \\ \hline \end{array}$$

python의 broadcast 연산 (Repeat 연산)

```
a = np.array([1,2,3])
b = 4
c = a+b
print(c)
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 4 & 4 & 4 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 5 & 6 & 7 \\ \hline \end{array}$$

python의 broadcast 연산 (Repeat 연산)

```
a = np.array([[1,2],[3,4]])  
b = 4  
c = a+b  
print(c)
```

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 4 & 4 \\ \hline 4 & 4 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 5 & 6 \\ \hline 7 & 8 \\ \hline \end{array}$$

(2,2) (2,2)

python의 broadcast 연산 (Repeat 연산)

```
a = np.array([[1,2],[3,4]])  
b = np.array([1,2])  
c = a+b  
print(c)
```

$$\begin{array}{|c|c|} \hline 1 & 2 \\ \hline 3 & 4 \\ \hline \end{array} + \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 2 & 4 \\ \hline 4 & 6 \\ \hline \end{array}$$

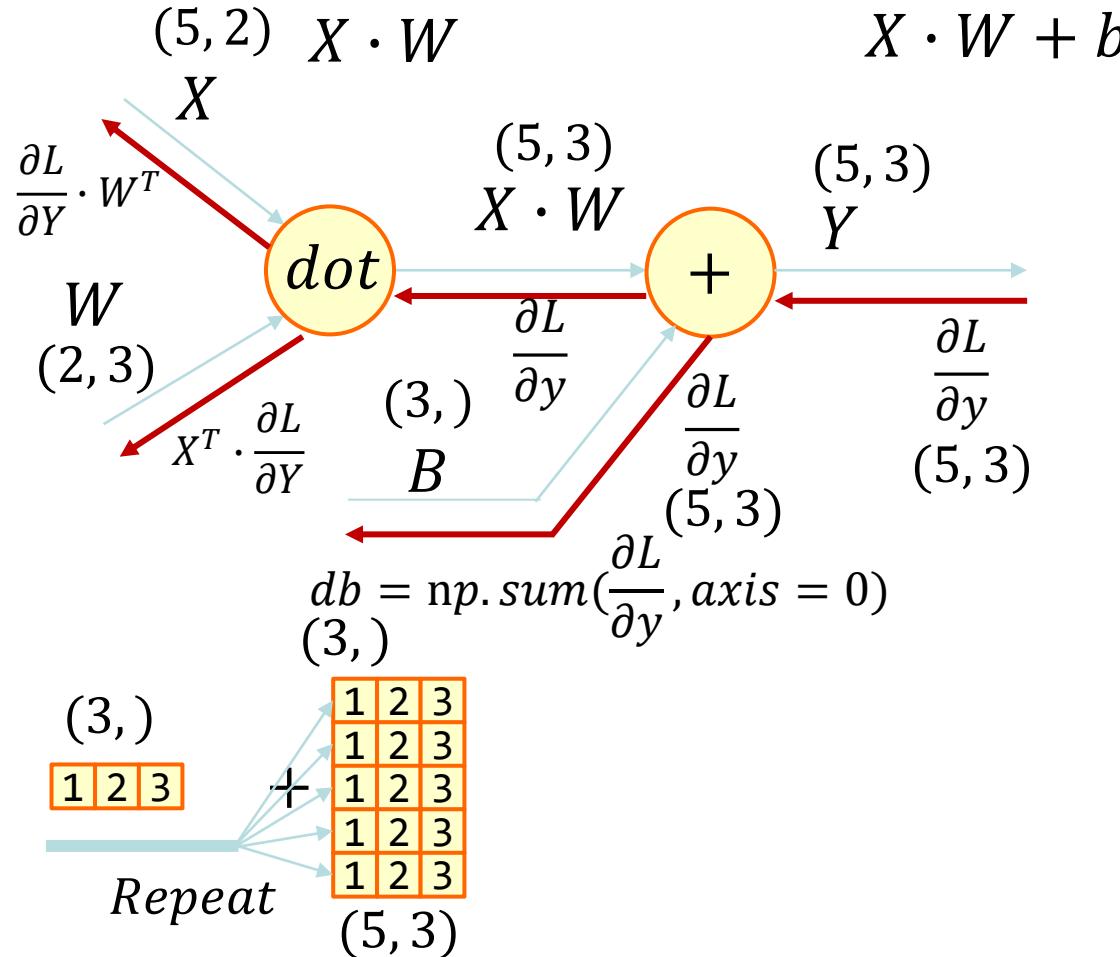
(2,2) (2,2)

python의 broadcast 연산 (Repeat 연산)

```
a = np.array([[1,2,3]])  
b = np.array([2,2,2])  
c = a+b  
print(c)
```

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 2 & 2 & 2 \\ \hline \end{array} = \\ (1,3) \qquad (3,) \\$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 3 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 2 & 2 & 2 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 3 & 4 & 5 \\ \hline \end{array} \\ (1,3) \qquad (1,3) \\$$



리피트 노드의 미분값은 합이다.

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

$$(5, 3) \quad (3, 2) = (5, 2)$$

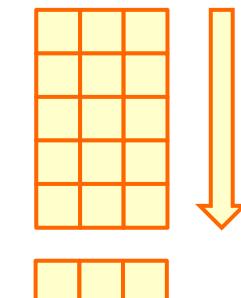
$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

$$(2, 5) \quad (5, 3) = (2, 3)$$

$\frac{\partial L}{\partial B} = \frac{\partial L}{\partial Y}$ 의 첫번째 축(0축)의 합

(5,3)

(3,)



```

class Affine:
    def __init__(self, w, b):
        self.w = w
        self.b = b
        self.X = None
        self.dw = None
        self.db = None

    def forward(self, x):
        self.X = x
        out = np.dot(x, self.w) + self.b
        return out

    def backward(self, dout):
        dx = np.dot(dout, self.w.T)
        self.dw = np.dot(self.X.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx

```

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

$$np.sum(\frac{\partial L}{\partial y}, axis=0)$$

```

class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.X = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.X = x
        out = np.dot(x, self.W) + self.b
        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.X.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx

```

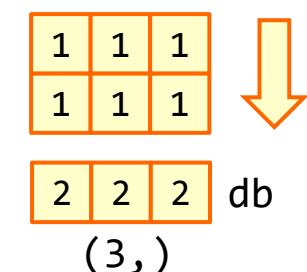
```

W = np.array([[1,1,1],[2,2,2]])
b = np.array([1,1,1])
a1 = Affine(W, b)
X = np.array([[1,1],[2,2]])
Y = a1.forward(X)
print(Y)
dout=[[1,1,1],[1,1,1]]
dx = a1.backward(dout)
print(dx)

```

$$\begin{array}{c}
 \text{self.X} \quad \text{self.W} \quad \text{self.b} \quad \text{Y} \\
 \begin{array}{|c|c|} \hline 1 & 1 \\ \hline 2 & 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 2 & 2 & 2 \\ \hline \end{array} + \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 4 & 4 & 4 \\ \hline 7 & 7 & 7 \\ \hline \end{array} \\
 (2,2) \quad (2,3) \quad (3,) \quad (2,3)
 \end{array}$$

$$\begin{array}{c}
 \text{dout} \quad \text{self.W.T} \quad \text{dx} \\
 \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \cdot \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 1 & 2 \\ \hline 1 & 2 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 3 & 6 \\ \hline 3 & 6 \\ \hline \end{array} \\
 (2,3) \quad (3,2) \quad (2,2) \\
 \text{self.X.T} \quad \text{dout} \quad \text{dw} \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline 1 & 2 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 3 & 3 & 3 \\ \hline 3 & 3 & 3 \\ \hline \end{array} \\
 (2,2) \quad (2,3) \quad (2,3)
 \end{array}$$



배치용 Affine 계층 구현

5.6 Affine/Softmax 계층 구현하기

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.X = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.X = x
        out = np.dot(x, self.W) + self.b
        return out

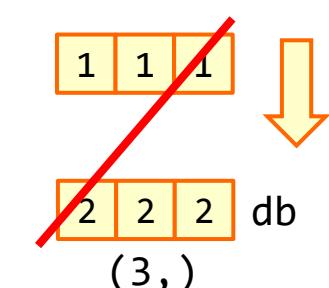
    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.X.T, dout)
        self.db = np.sum(dout, axis=0)
        return dx
```

```
w = np.array([[1,1,1],[2,2,2]])
b = np.array([1,1,1])
a1 = Affine(w, b)
X = np.array([1,1])
Y = a1.forward(X)
print(Y)
dout=[1,1,1]
dx = a1.backward(dout)
print(dx)
```

$$\begin{matrix} \text{self.X} \\ \begin{matrix} 1 & 1 \\ (2,) \end{matrix} \end{matrix} \cdot \begin{matrix} \text{self.W} \\ \begin{matrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ (2,3) \end{matrix} \end{matrix} + \begin{matrix} \text{self.b} \\ \begin{matrix} 1 & 1 & 1 \\ (3,) \end{matrix} \end{matrix} = \begin{matrix} Y \\ \begin{matrix} 4 & 4 & 4 \\ (3,) \end{matrix} \end{matrix}$$

$$\begin{matrix} \text{dout} \\ \begin{matrix} 1 & 1 & 1 \\ (3,) \end{matrix} \end{matrix} \cdot \begin{matrix} \text{self.W.T} \\ \begin{matrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \\ (3,2) \end{matrix} \end{matrix} = \begin{matrix} \text{dx} \\ \begin{matrix} 3 & 6 \\ (2,) \end{matrix} \end{matrix}$$

$$\begin{matrix} \text{self.X.T} \\ \begin{matrix} 1 & 1 \\ (2,) \end{matrix} \end{matrix} \cdot \begin{matrix} \text{dout} \\ \begin{matrix} 1 & 1 & 1 \\ (3,) \end{matrix} \end{matrix} = \begin{matrix} \text{dW} \\ \begin{matrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ (2,3) \end{matrix} \end{matrix}$$



배치용 Affine 계층 구현

5.6 Affine/Softmax 계층 구현하기

```

class Affine:
    def __init__(self, W, b):
        ...
        ...

    def forward(self, x):
        self.original_x_shape = x.shape # (2,)
        if x.ndim == 1 :
            x = x.reshape(-1, x.shape[0])
        self.x = x

        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx = dx.reshape(*self.original_x_shape)
        return dx

```

```

W = np.array([[1,1,1],[2,2,2]])
b = np.array([1,1,1])
a1 = Affine(W, b)
X = np.array([1,1])
Y = a1.forward(X)
print(Y)
dout=[[1,1,1]]
dx = a1.backward(dout)
print(dx)

```

Forward pass:

$$\text{self.X} \cdot \text{self.W} + \text{self.b} = Y$$

$\begin{matrix} 1 & 1 \\ 2 & 2 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \\ 2 & 2 & 2 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \end{matrix}$	$\begin{matrix} 4 & 4 & 4 \end{matrix}$
(1, 2)	(2, 3)	(3,)	(1, 3)

Backward pass:

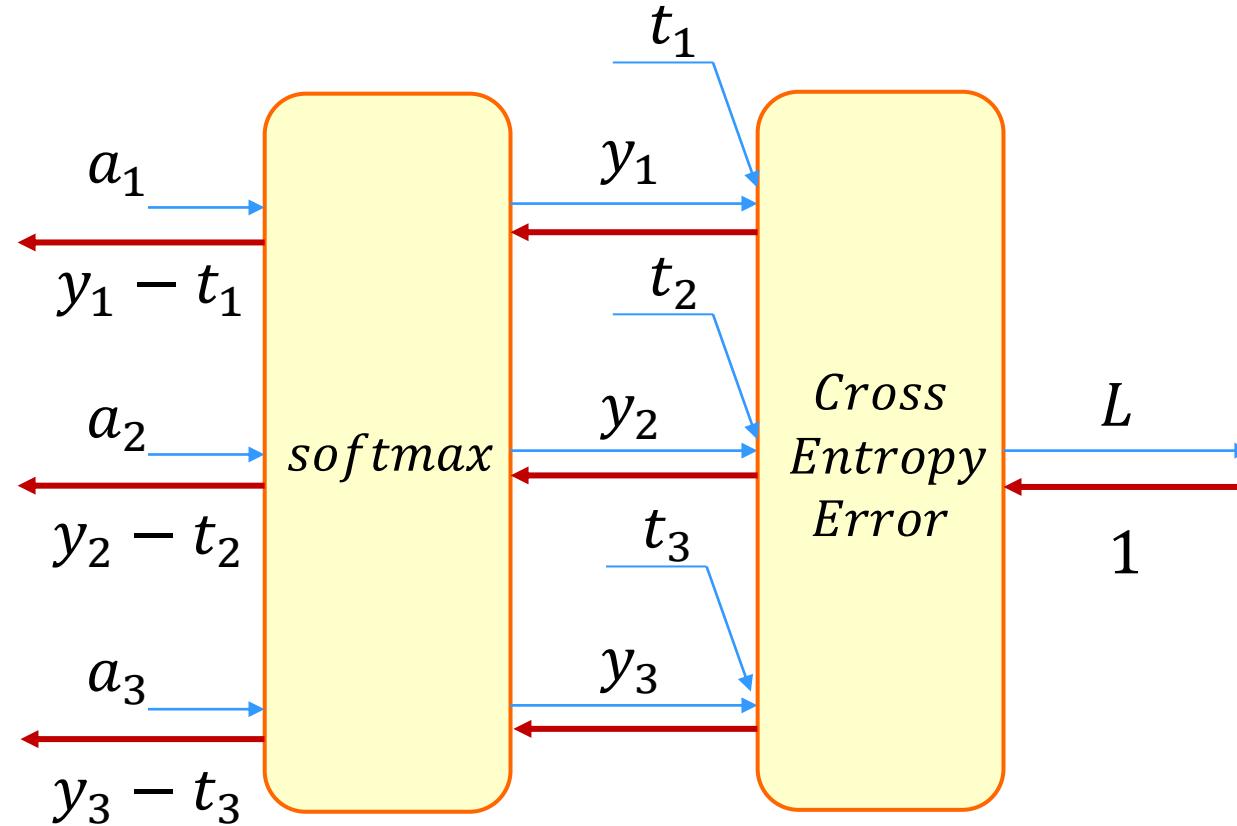
$$\text{dout} \cdot \text{self.W.T} = \text{dx}$$

$\begin{matrix} 1 & 1 & 1 \end{matrix}$	$\begin{matrix} 1 & 2 \\ 1 & 2 \\ 1 & 2 \end{matrix}$	$\begin{matrix} 3 & 6 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \end{matrix}$
(1, 3)	(3, 2)	(1, 2)	(1, 3)

$$\text{self.X.T} \cdot \text{dout} = \text{dw}$$

$\begin{matrix} 1 \\ 1 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$
(2, 1)	(1, 3)	(2, 3)

\downarrow db



```
def softmax(x):
    if x.ndim == 2:
        x = x.T
        x = x - np.max(x, axis=0)
        y = np.exp(x) / np.sum(np.exp(x), axis=0)
        return y.T

    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))

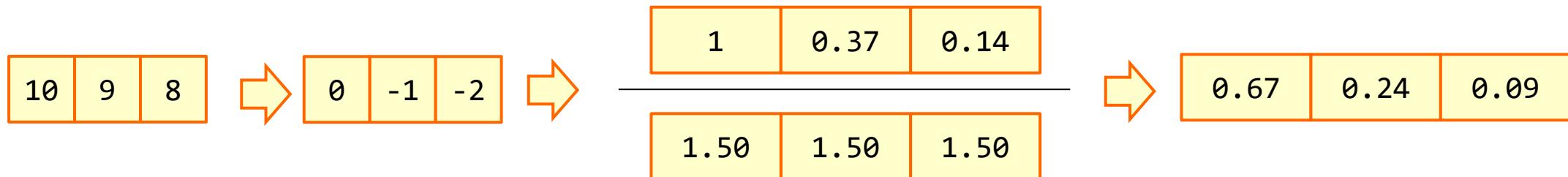
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

```
def softmax(x):
    if x.ndim == 2:
        x = x - np.max(x, axis=1).reshape(-1,1)
        y = np.exp(x) / np.sum(np.exp(x), axis=1).reshape(-1,1)
        return y

    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))
```



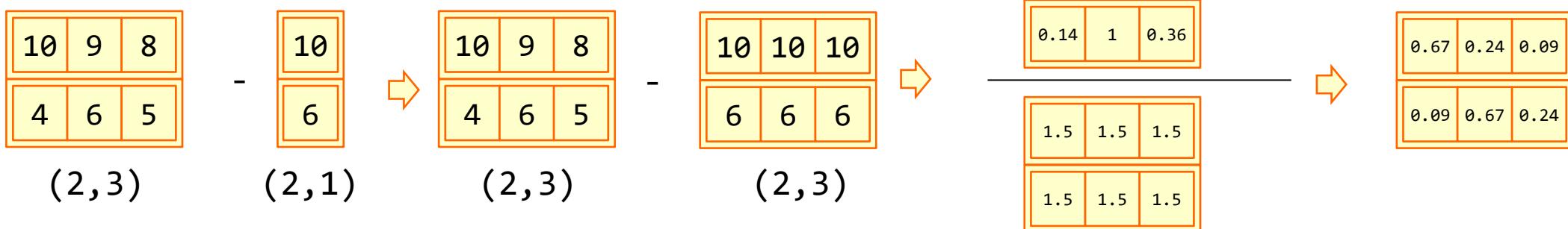
Softmax with Loss 계층 구현

5.6 Affine/Softmax 계층 구현하기

```
def softmax(x):
    if x.ndim == 2:
        x = x - np.max(x, axis=1).reshape(-1,1)
        y = np.exp(x) / np.sum(np.exp(x), axis=1).reshape(-1,1)
        return y

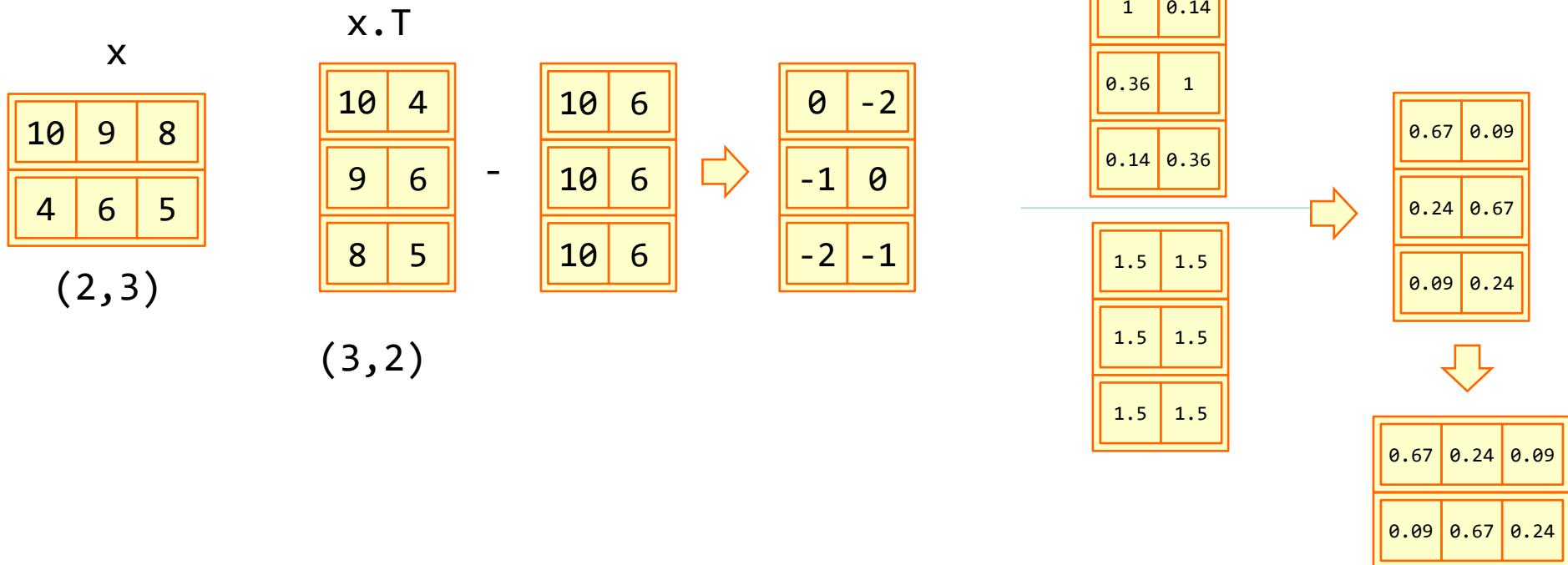
    x = x - np.max(x) # 오버플로 대책
    return np.exp(x) / np.sum(np.exp(x))
```

```
x = np.array([[10,9,8],
              [ 4,6,5]])
```



```
def softmax(x):
    if x.ndim == 2:
        x = x.T
    x = x - np.max(x, axis=0)
    y = np.exp(x) / np.sum(np.exp(x), axis=0)
    return y.T
...

```



```

def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size) # (3,) => (1,3)
        y = y.reshape(1, y.size) # (3,) => (1,3)

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size

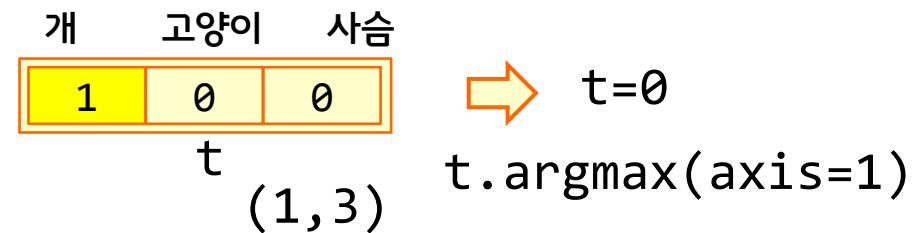
```

$$-\sum_{i=1}^3 t_i * \log(y_i) = -(t_1 * \log(y_1) + t_2 * \log(y_2) + t_3 * \log(y_3))$$

1.6

0.2	0.7	0.1
-----	-----	-----

y
(1, 3)



```

def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    # 훈련 데이터가 원-핫 벡터라면 정답 레이블의 인덱스로 반환
    if t.size == y.size:
        t = t.argmax(axis=1)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size

```

$$\begin{aligned}
 & y[[0,1],[0,1]] \Rightarrow [0.2, 0.1] \Rightarrow [-1.6, -2.3] \\
 & -\sum_{i=1}^3 t_i * \log(y_i) \\
 & -(t_1 * \log(y_1) + t_2 * \log(y_2) + t_3 * \log(y_3)) \\
 & \frac{-1.6 - 2.3}{2} = -1.95 = 1.95
 \end{aligned}$$

0.2	0.1	0.7
0.3	0.1	0.6

y (2, 3)

개 고양이 사슴

1	0	0
0	1	0

t (2, 3)



t (2,)

```
class SoftmaxWithLoss:  
    def __init__(self):  
        self.loss = None  
        self.y = None  
        self.t = None  
  
    def forward(self, x, t):  
        self.t = t  
        self.y = softmax(x)  
        print(self.y)  
        self.loss = cross_entropy_error(self.y, self.t)  
        return self.loss  
  
    def backward(self, dout=1):  
        batch_size = self.t.shape[0]  
        dx = (self.y - self.t) / batch_size  
        return dx
```

개 고양이 사슴						
-						
y						
t						

```

class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None
        self.t = None

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        print(self.y)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size
        return dx

```

$$\begin{array}{r}
 \begin{array}{|c|c|c|} \hline
 -0.8 & 0.1 & 0.7 \\ \hline
 \end{array} \\
 \hline
 \text{batch_size}
 \end{array}$$

$$\begin{array}{r}
 \begin{array}{|c|c|c|} \hline
 -0.8 & 0.1 & 0.7 \\ \hline
 \end{array} \\
 \hline
 \mathbf{dx}
 \end{array}$$

$$w = w - rate * dw$$

Softmax with Loss 계층 구현

5.6 Affine/Softmax 계층 구현하기

```

class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None
        self.t = None

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        print(self.y)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size
        return dx

```

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline
 0.2 & 0.1 & 0.7 \\ \hline
 0.3 & 0.1 & 0.6 \\ \hline
 \end{array} - \begin{array}{|c|c|c|} \hline
 1 & 0 & 0 \\ \hline
 0 & 1 & 0 \\ \hline
 \end{array} \\
 \text{y} \qquad \qquad \qquad \text{t}
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline
 -0.8 & 0.1 & 0.7 \\ \hline
 0.3 & -0.9 & 0.6 \\ \hline
 \end{array} \\
 \hline
 \end{array}$$

batch_size

$$\begin{array}{c}
 \begin{array}{|c|c|c|} \hline
 -0.4 & 0.05 & 0.35 \\ \hline
 0.15 & -0.45 & 0.3 \\ \hline
 \end{array} \\
 \hline
 \end{array}$$

dx

$w = w - rate * dw$

5. 오차역전파

5.1 계산그래프

5.2 연쇄법칙

5.3 역전파

5.4 단순한 계층 구현하기

5.5 활성화 함수 계층 구현하기

5.6 Affine/Softmax 계층 구현하기

5.7 오차역전파법 구현하기

1단계 - 미니배치

2단계 - 기울기 산출

3단계 - 매개변수 갱신

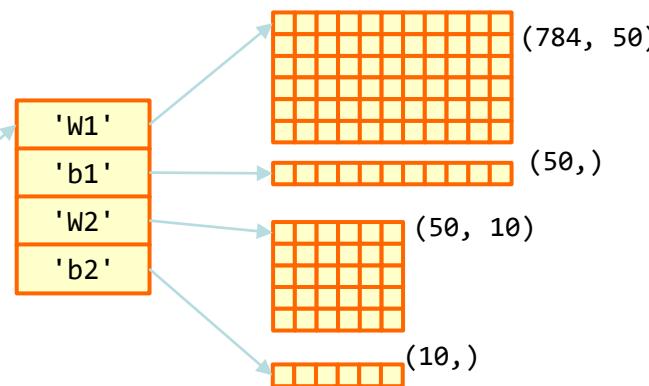
4단계 - 반복

1~3단계를 반복한다.

```
TwoLayerNet.__init__()
```

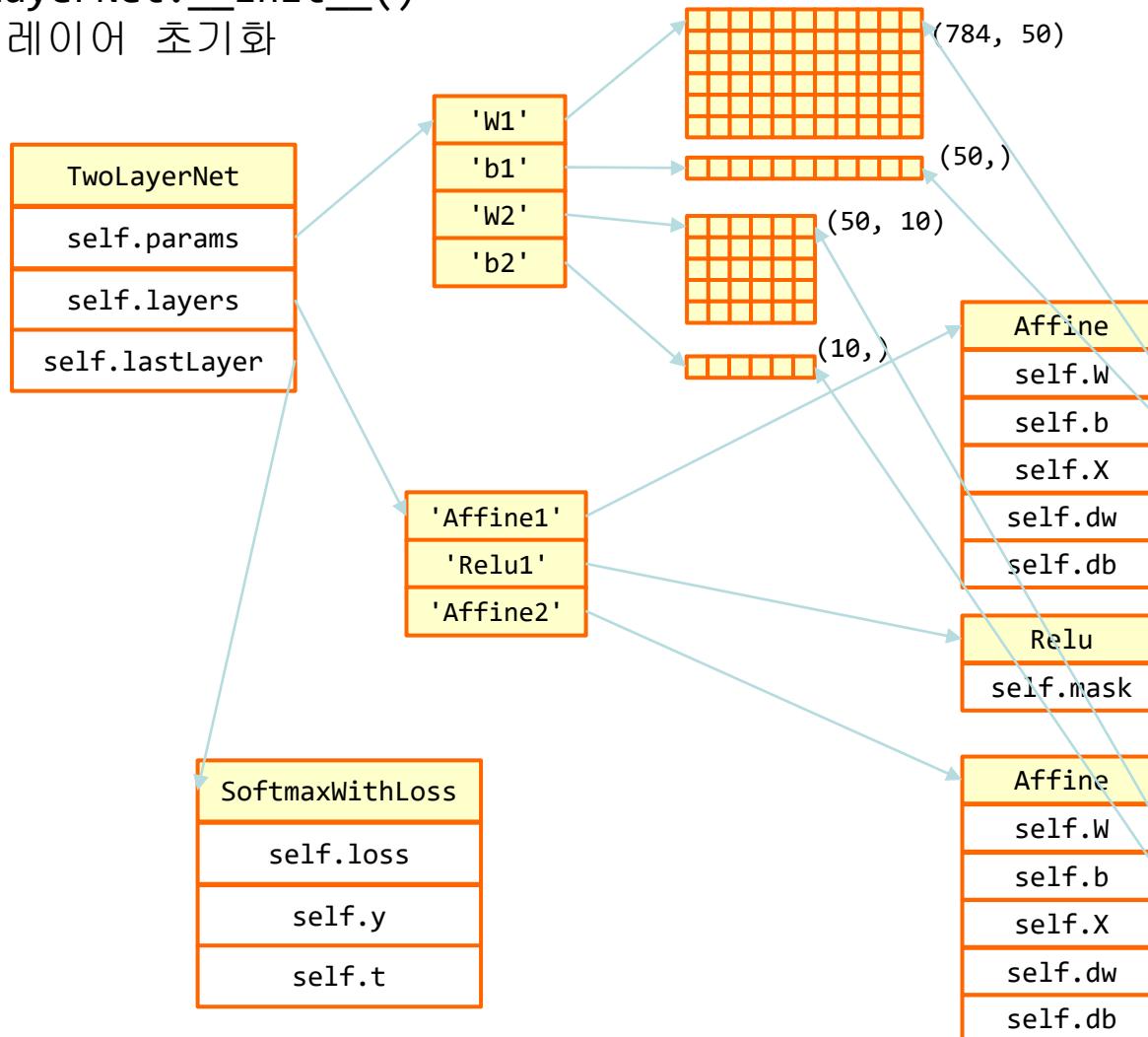
```
# 가중치 초기화
```

TwoLayerNet
self.params
self.layers
self.lastLayer

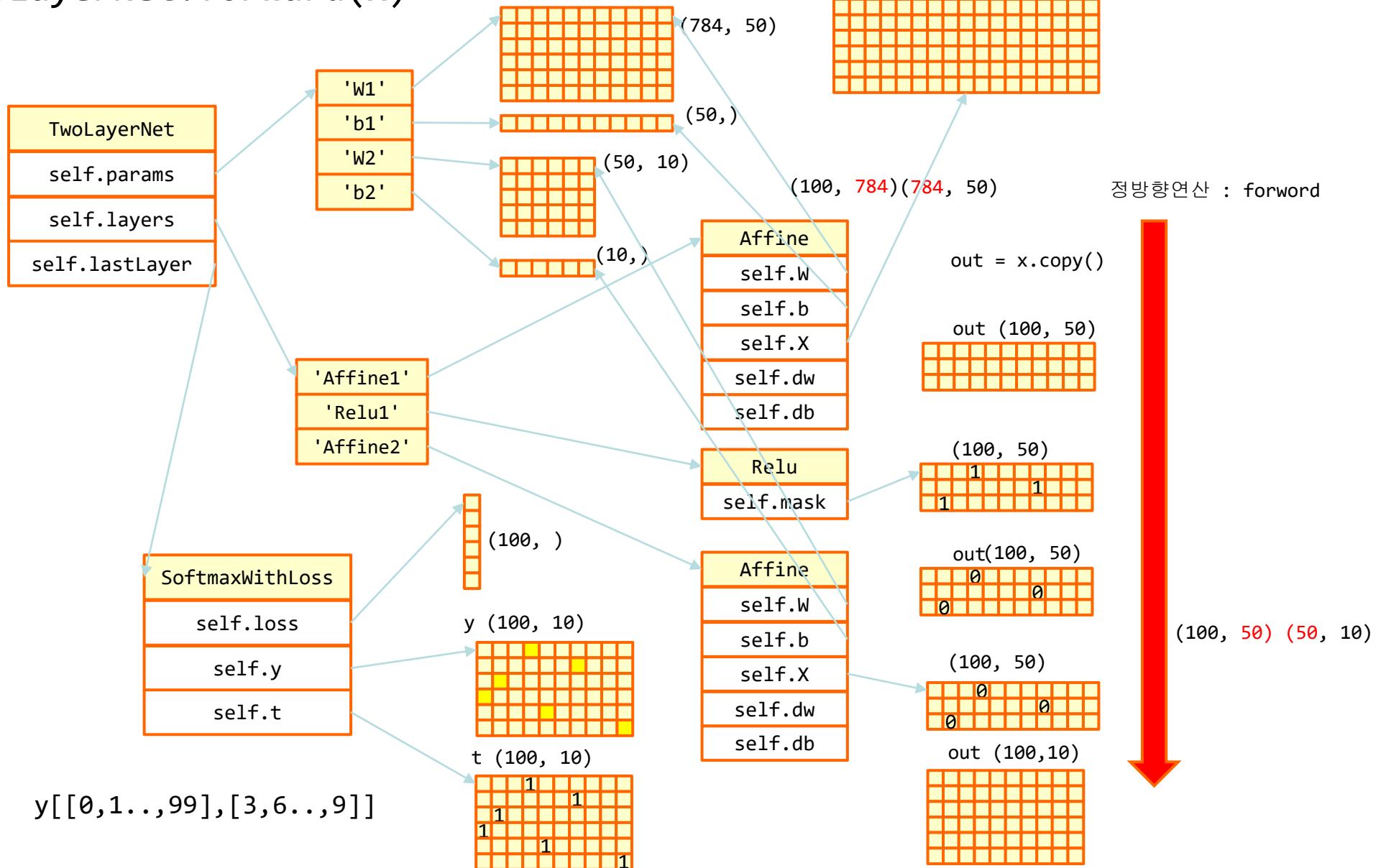


TwoLayerNet.__init__()

레이어 초기화



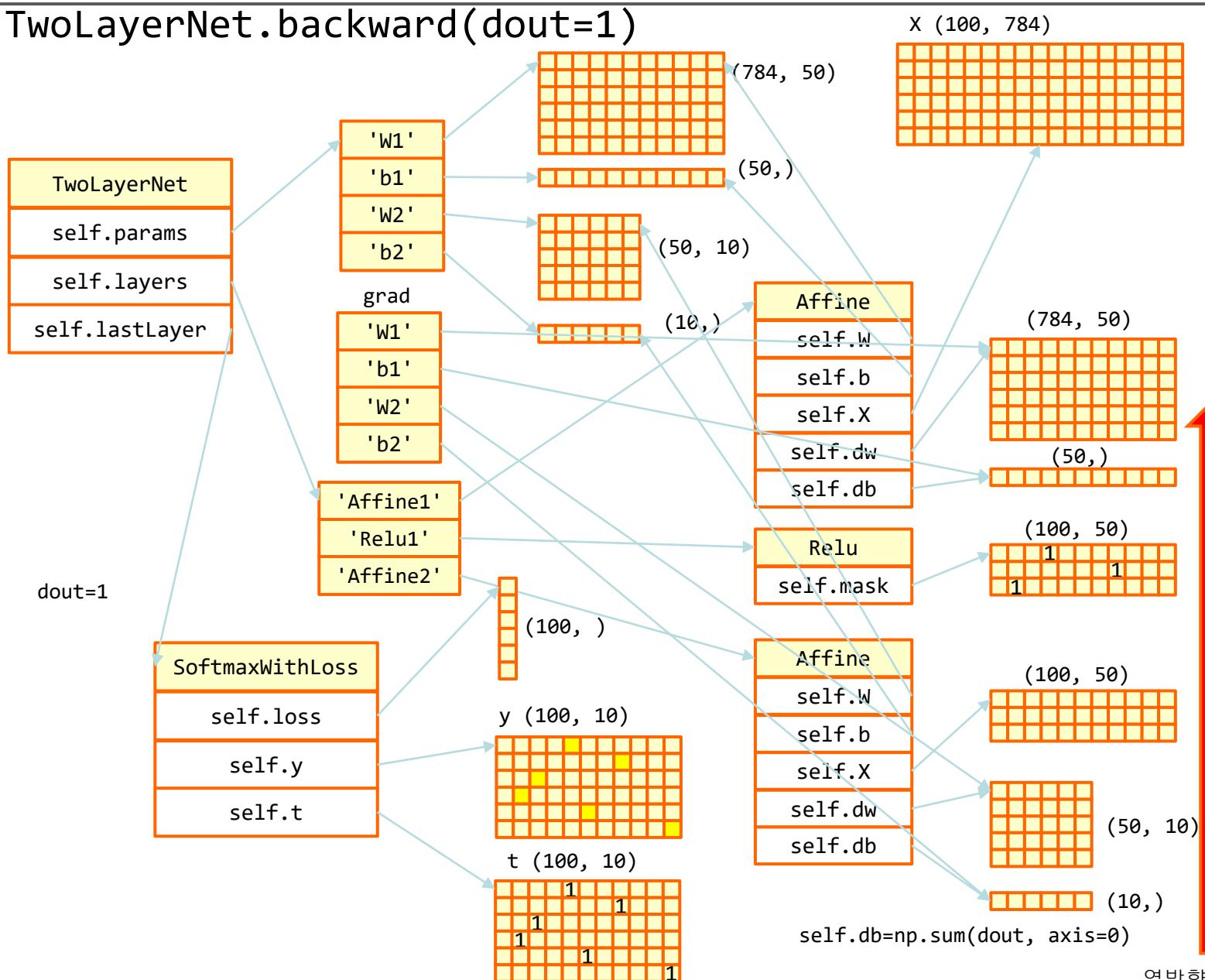
TwoLayerNet.forward(x)



신경망 학습 전체 그림

5.7 오차역전파법 구현하기

TwoLayerNet.backward(dout=1)



역방향연산 : backward

`self.db=np.sum(dout, axis=0)`

`self.dx=np.dot(dout, self.W.T)`
 $(100, 784) \quad (100, 50) \quad (50, 784)$

`self.dw=np.dot(self.X.T, dout)`
 $(784, 50) \quad (784, 100) \quad (100, 50)$

$(100, 50) \quad \text{dout}$
 $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

$\text{dout}[\text{self.mask}] = 0$
 $(100, 50) \quad \text{dout}$
 $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

$\text{dout} = \text{np.dot}(\text{dout}, \text{self.W.T})$
 $(100, 50) \quad (100, 10) \quad (10, 50)$

`self.dw=np.dot(self.X.T, dout)`
 $(50, 10) \quad (50, 100) \quad (100, 10)$

dout
 $(y-t)/100 \quad (100, 10)$

6. 학습 관련 기술들

6.1 매개변수 갱신

6.2 가중치의 초기값

6.3 배치 정규화

6.4 바른 학습을 위해

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

에타(ETA) : H η

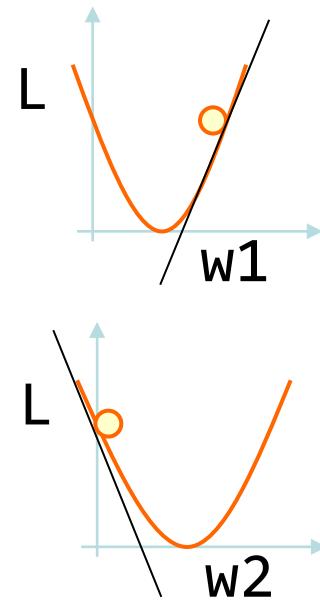
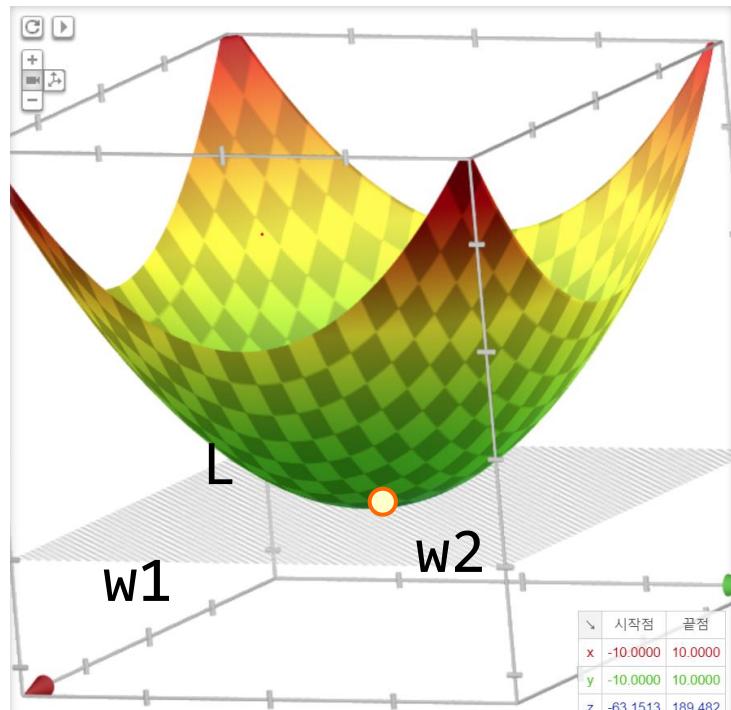
```
class SGD:
```

```
    """확률적 경사 하강법 (Stochastic Gradient Descent) """
```

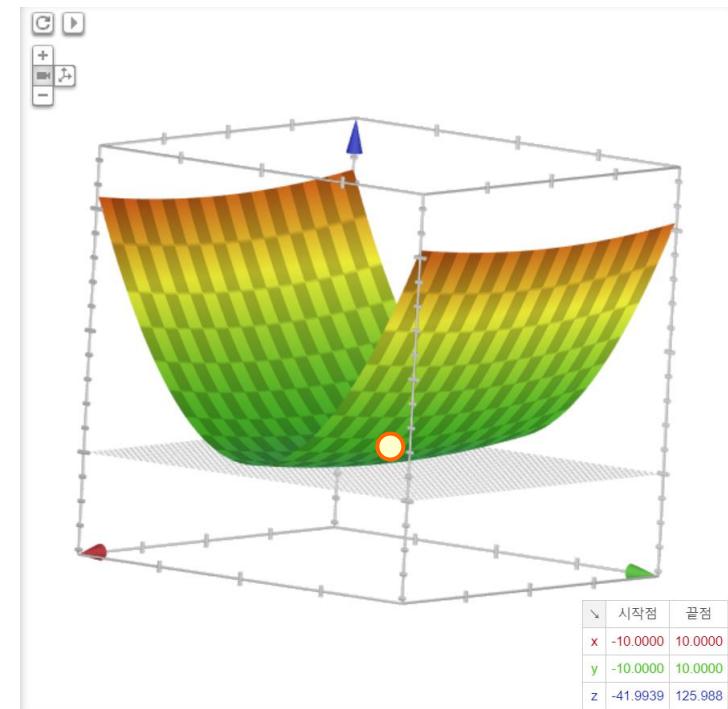
```
    def __init__(self, lr=0.01):  
        self.lr = lr
```

```
    def update(self, params, grads):  
        for key in params.keys():  
            params[key] -= self.lr * grads[key]
```

$$f(x, y) = x^2 + y^2$$

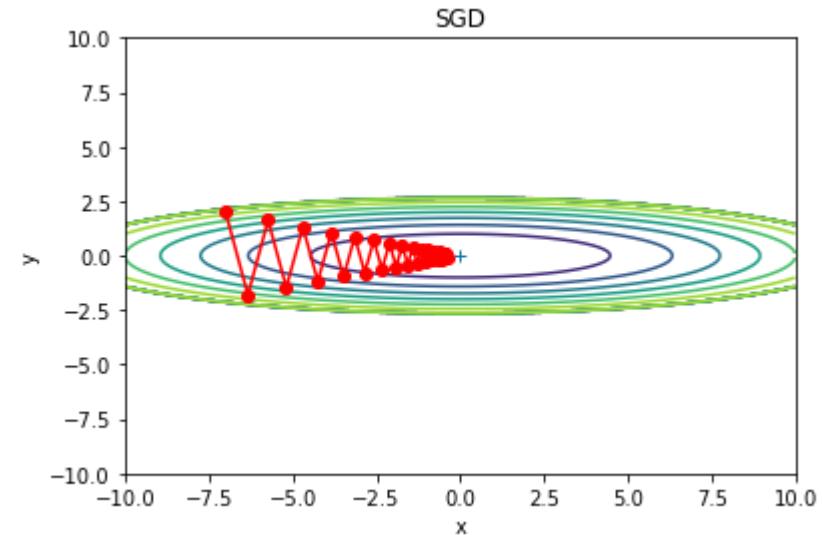
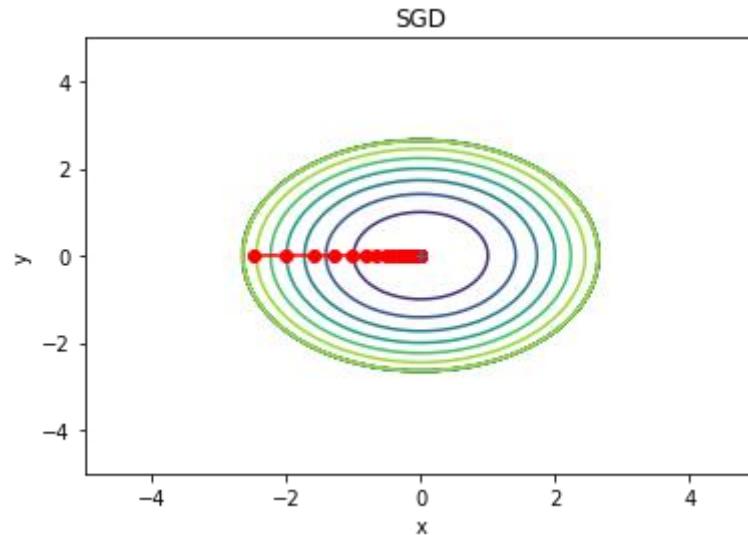


$$f(x, y) = 1/20x^2 + y^2$$



$$f(x, y) = x^2 + y^2$$

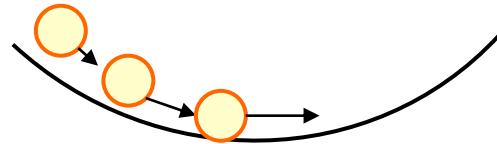
$$f(x, y) = 1/20x^2 + y^2$$

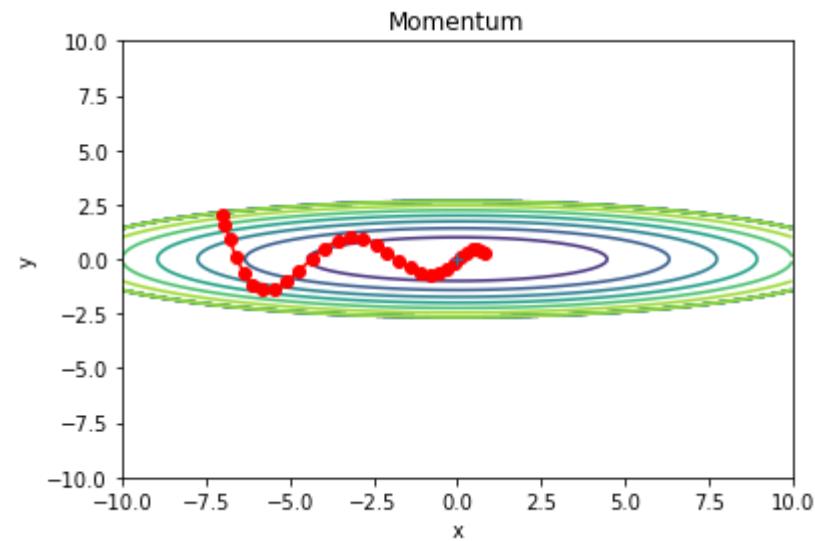
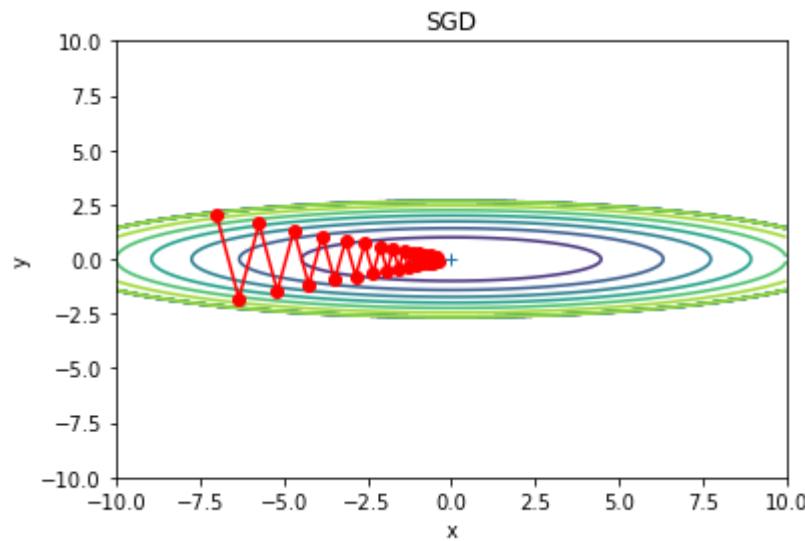


```
class Momentum:  
  
    """모멘텀 SGD"""  
  
    def __init__(self, lr=0.01, momentum=0.9):  
        self.lr = lr  
        self.momentum = momentum  
        self.v = None  
  
    def update(self, params, grads):  
        if self.v is None:  
            self.v = {}  
            for key, val in params.items():  
                self.v[key] = np.zeros_like(val)  
  
        for key in params.keys():  
            self.v[key] = self.momentum*self.v[key] - self.lr*grads[key] # 점화식  
            params[key] += self.v[key]  
  
-self.lr*grads[key]  
-self.lr*grads[key]*0.9 - self.lr*grads[key]  
(-self.lr*grads[key]*0.9 - self.lr*grads[key])*0.9 - self.lr*grads[key]
```

$$\nu \leftarrow \alpha \nu - \eta \frac{\partial L}{\partial W}$$

$$w \leftarrow w + \nu$$





```
class AdaGrad:

    """AdaGrad"""

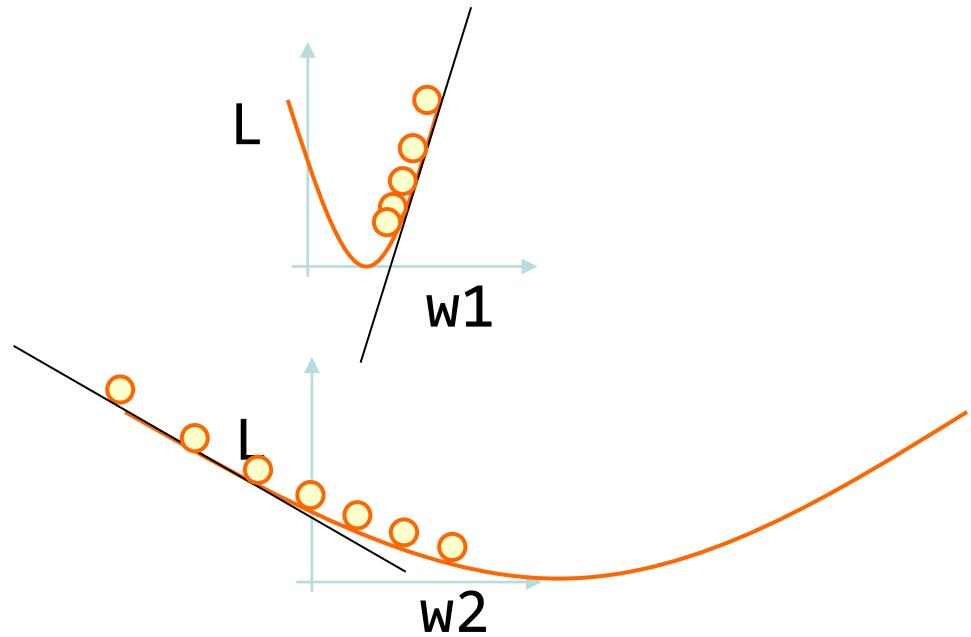
    def __init__(self, lr=0.01):
        self.lr = lr
        self.h = None

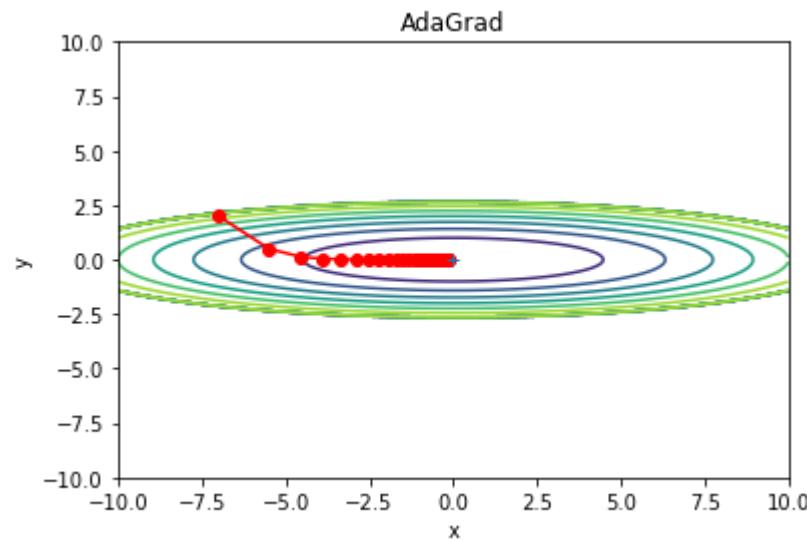
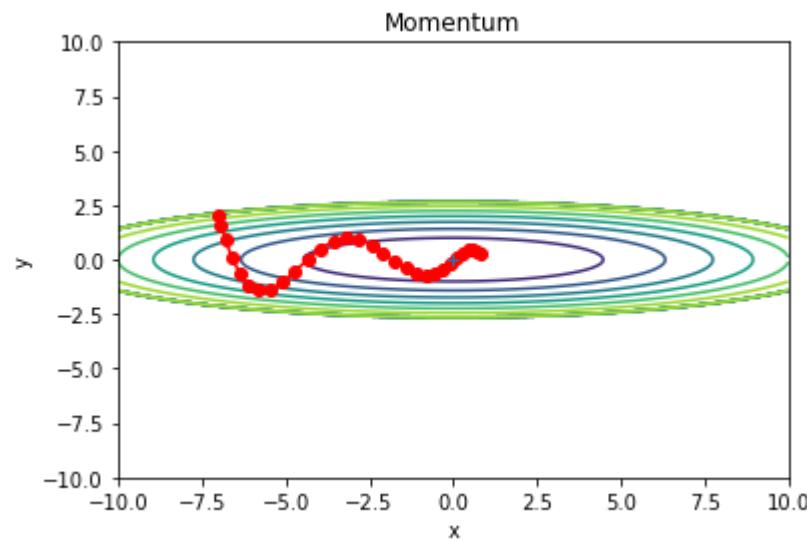
    def update(self, params, grads):
        if self.h is None:
            self.h = {}
        for key, val in params.items():
            self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] = self.h[key] + grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```

$$h \leftarrow h + \frac{\partial L}{\partial W} * \frac{\partial L}{\partial W}$$

$$w \leftarrow w - \eta * \frac{1}{\sqrt{h}} * \frac{\partial L}{\partial W}$$





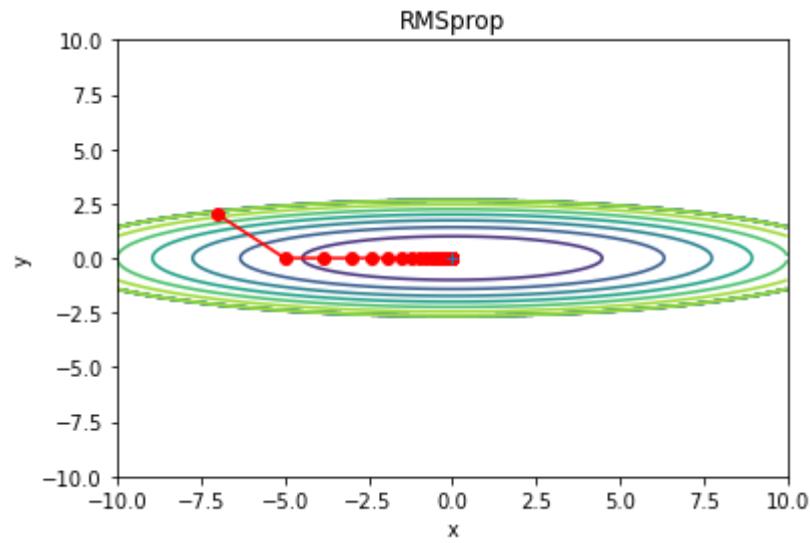
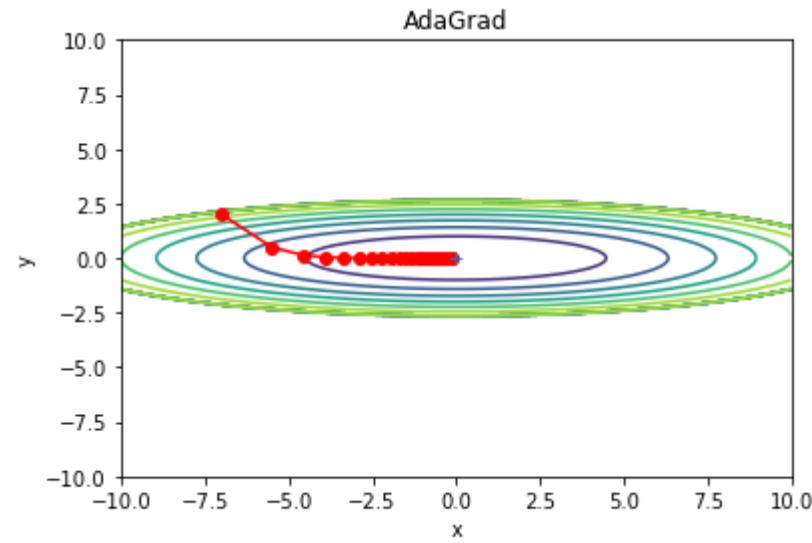
```
class RMSprop:

    """RMSprop"""

    def __init__(self, lr=0.01, decay_rate = 0.99):
        self.lr = lr
        self.decay_rate = decay_rate
        self.h = None

    def update(self, params, grads):
        if self.h is None:
            self.h = {}
            for key, val in params.items():
                self.h[key] = np.zeros_like(val)

        for key in params.keys():
            self.h[key] *= self.decay_rate
            self.h[key] += (1 - self.decay_rate) * grads[key] * grads[key]
            params[key] -= self.lr * grads[key] / (np.sqrt(self.h[key]) + 1e-7)
```



```
class Adam:

    """Adam (http://arxiv.org/abs/1412.6980v8)"""

    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.iter = 0
        self.m = None
        self.v = None

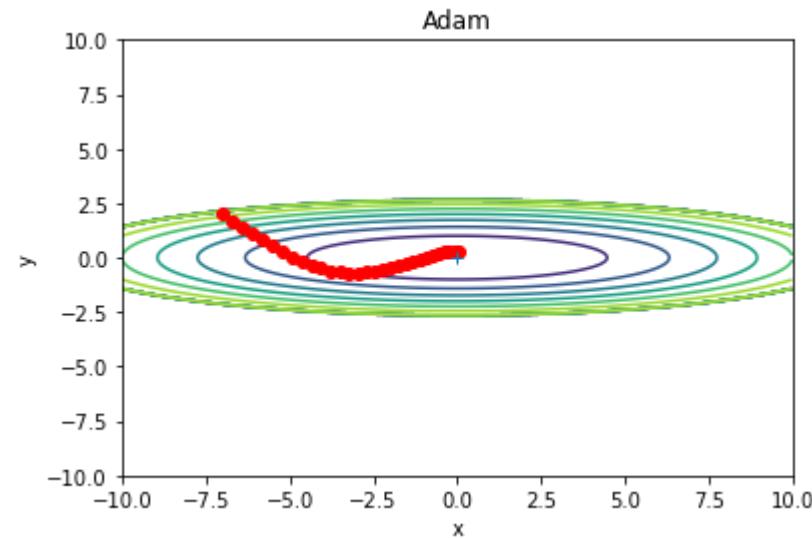
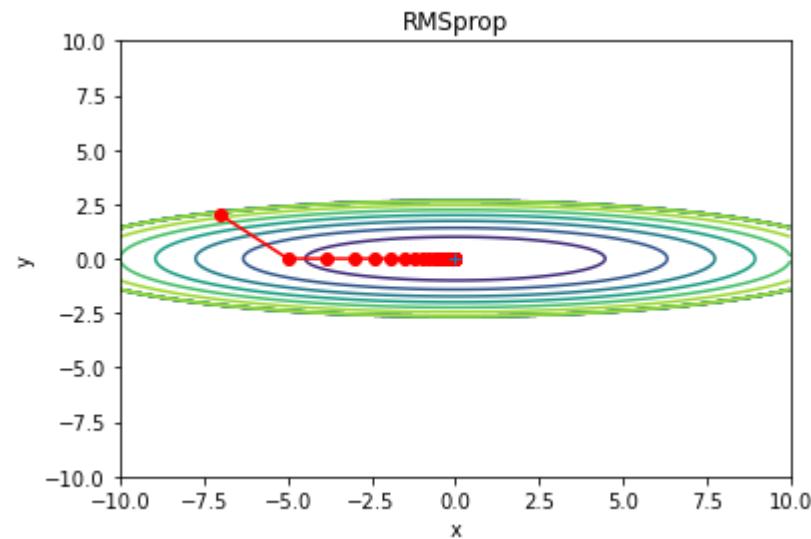
    def update(self, params, grads):
        if self.m is None:
            self.m, self.v = {}, {}
            for key, val in params.items():
                self.m[key] = np.zeros_like(val)
                self.v[key] = np.zeros_like(val)

        self.iter += 1
        lr_t = self.lr * np.sqrt(1.0 - self.beta2**self.iter) / (1.0 - self.beta1**self.iter)

        for key in params.keys():
            #self.m[key] = self.beta1*self.m[key] + (1-self.beta1)*grads[key]
            #self.v[key] = self.beta2*self.v[key] + (1-self.beta2)*(grads[key]**2)
            self.m[key] += (1 - self.beta1) * (grads[key] - self.m[key])
            self.v[key] += (1 - self.beta2) * (grads[key]**2 - self.v[key])

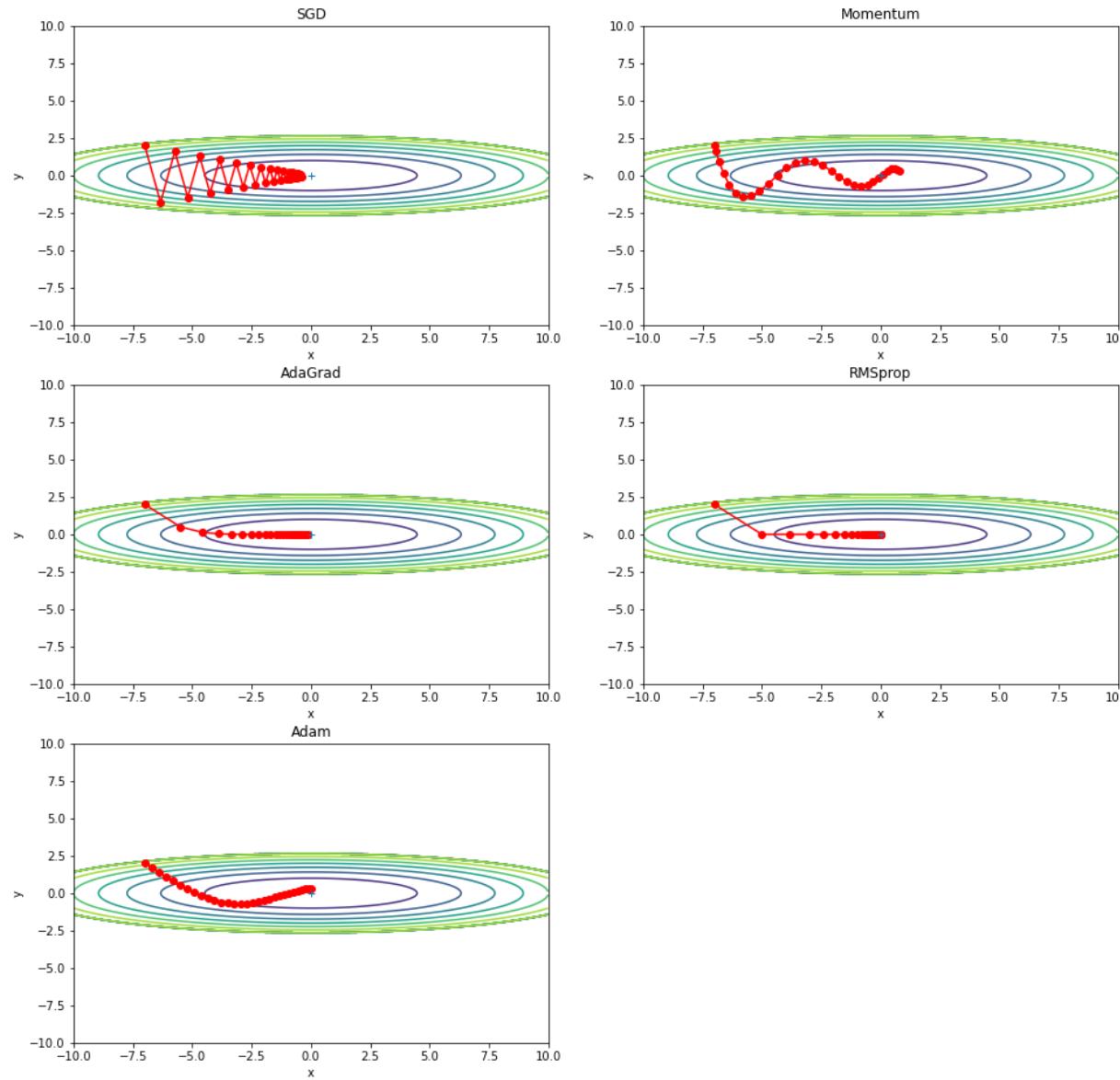
            params[key] -= lr_t * self.m[key] / (np.sqrt(self.v[key]) + 1e-7)

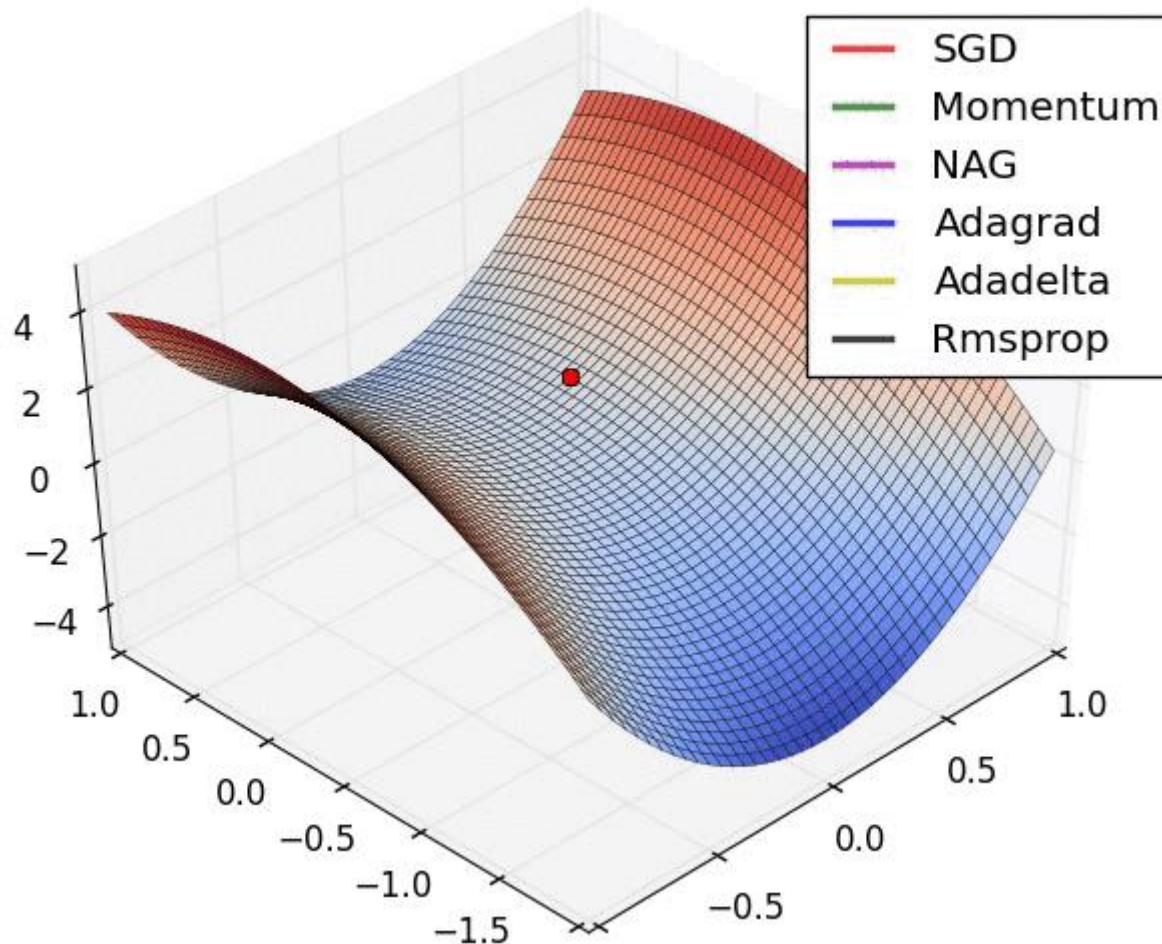
            #unbias_m += (1 - self.beta1) * (grads[key] - self.m[key]) # correct bias
            #unbias_b += (1 - self.beta2) * (grads[key]*grads[key] - self.v[key]) # correct bias
            #params[key] += self.lr * unbias_m / (np.sqrt(unbias_b) + 1e-7)
```

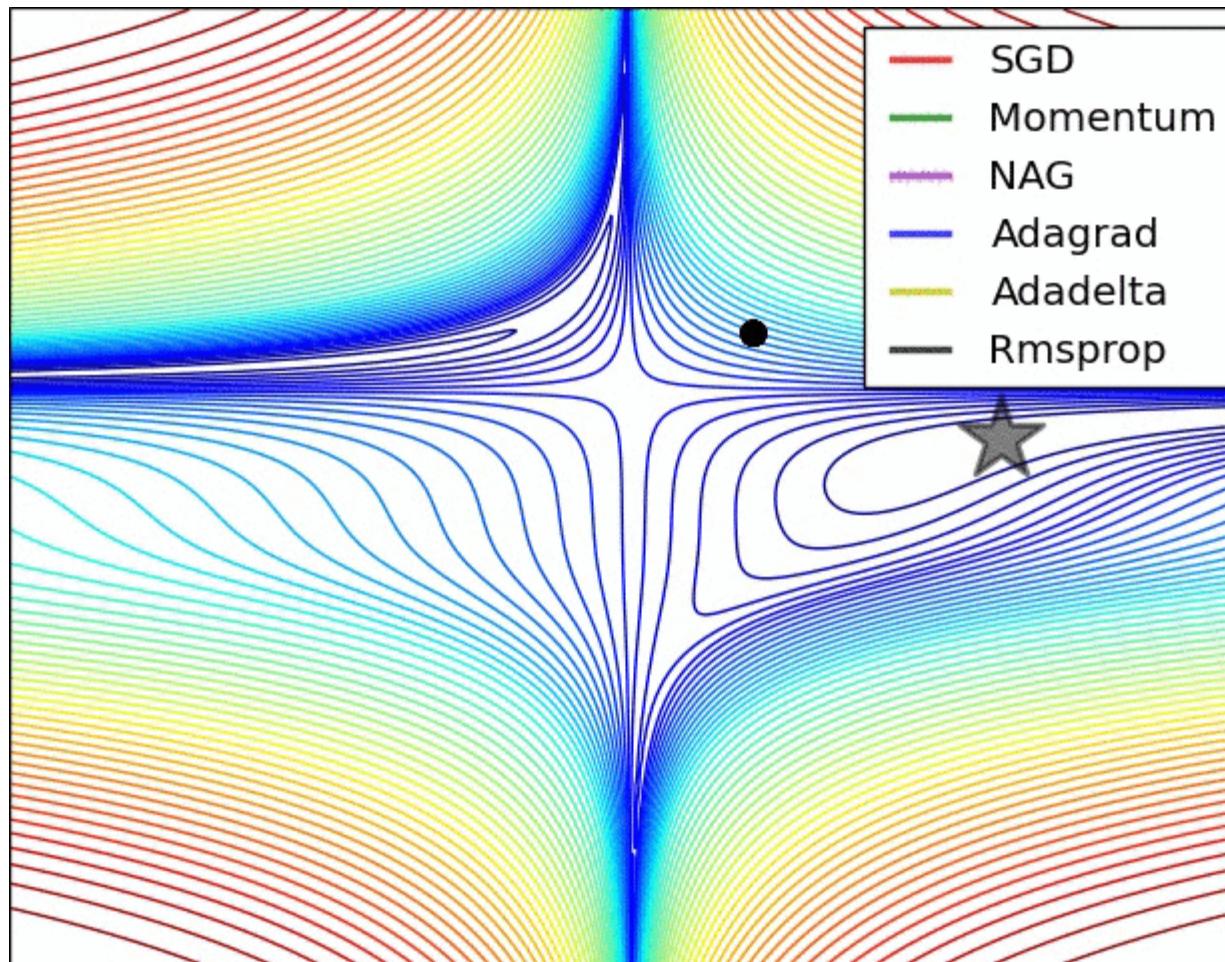


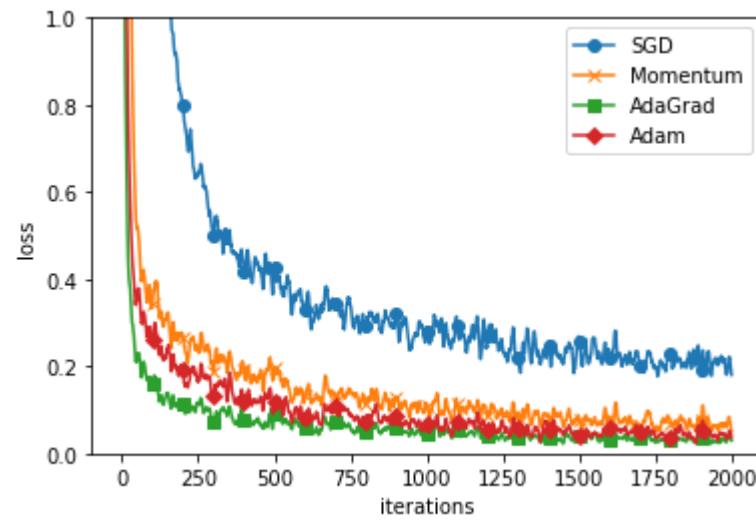
각 최적화기 비교

6.1 매개변수 갱신









6. 학습 관련 기술들

6.1 매개변수 갱신

6.2 가중치의 초기값

6.3 배치 정규화

6.4 바른 학습을 위해

초깃값을 0으로 하면? 절대로 안된다.

$W \leftarrow 0$

w = np.zeros(100, 10)

$XW + b$

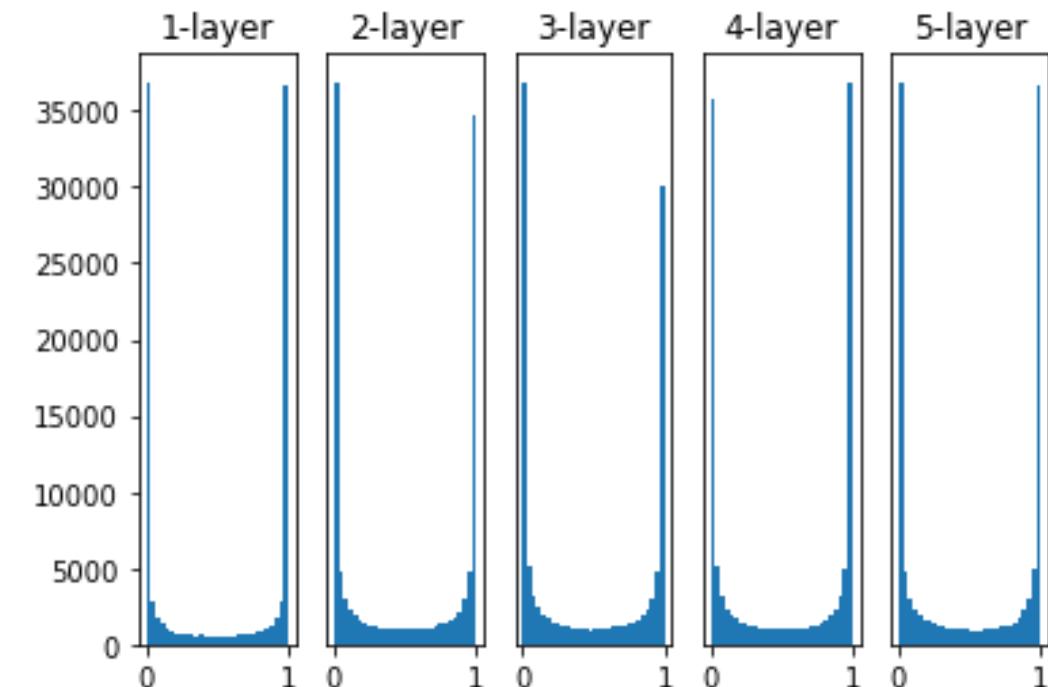
w = np.ones(100, 10)

순전파시 입력층의 가중치가 0이기 때문에 두 번째 층의 뉴런에 모두 같은 값이 전달된다.

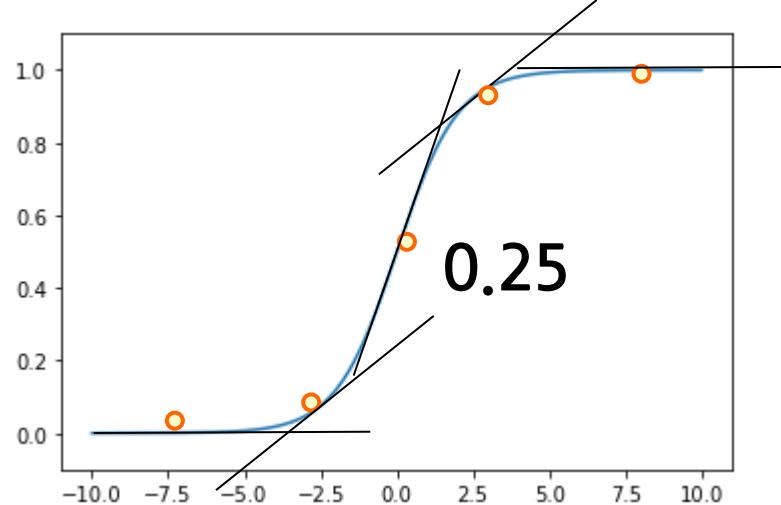
역전파 시에도 기울기가 동일한 값이 전달된다.

초깃값을 랜덤하게 해야한다. (평균:0, 표준편차:1 np.random.randn함수를 사용하자.)

```
w = np.random.randn(node_num, node_num) * 1
a = np.dot(x,w)
z = sigmoid(a)
```

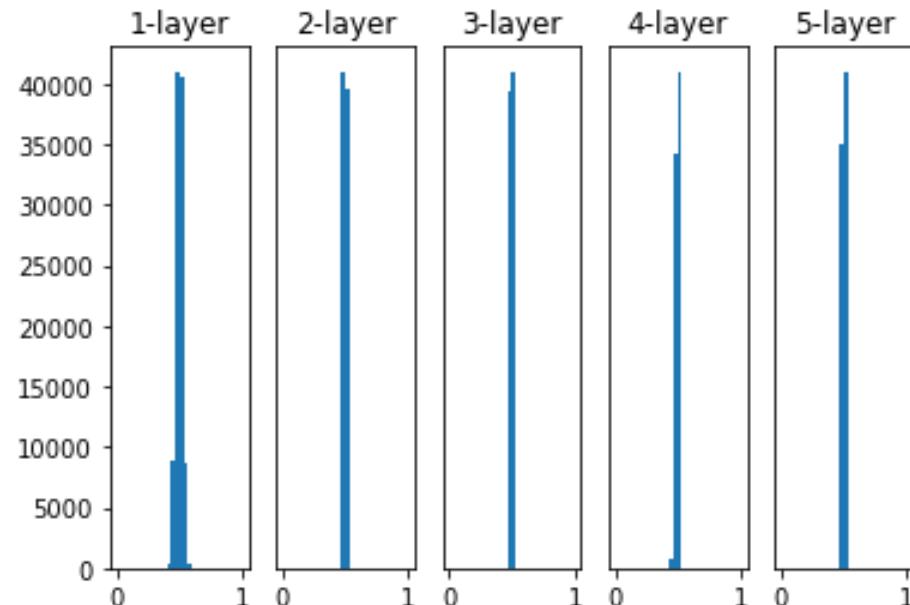


$$w = w - dw$$



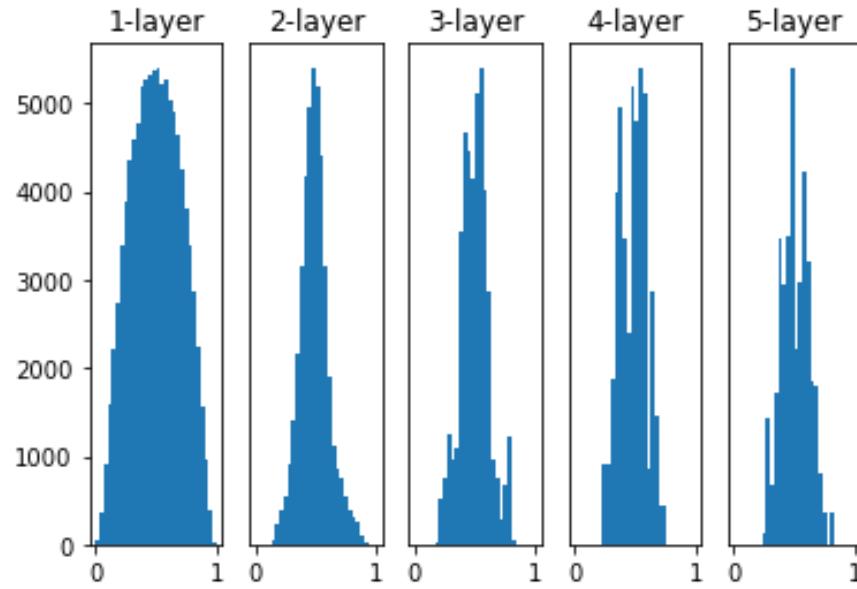
활성화 함수가
sigmoid인 경우는
0에 가까울수록
1에 가까울수록
기울기는 0에 가까워
지므로
학습 되지 않는다.

```
w = np.random.randn(node_num, node_num) * 0.01
```



활성화 함수가
기울기 소실 문제를
해소하려면 w 의 범위를
제한 하면 되는데
0.01로 제한 했을 경우
가운데로 활성화 값이
모이는 문제가 발생한다.

```
w = np.random.randn(node_num, node_num) / np.sqrt(node_num)
```

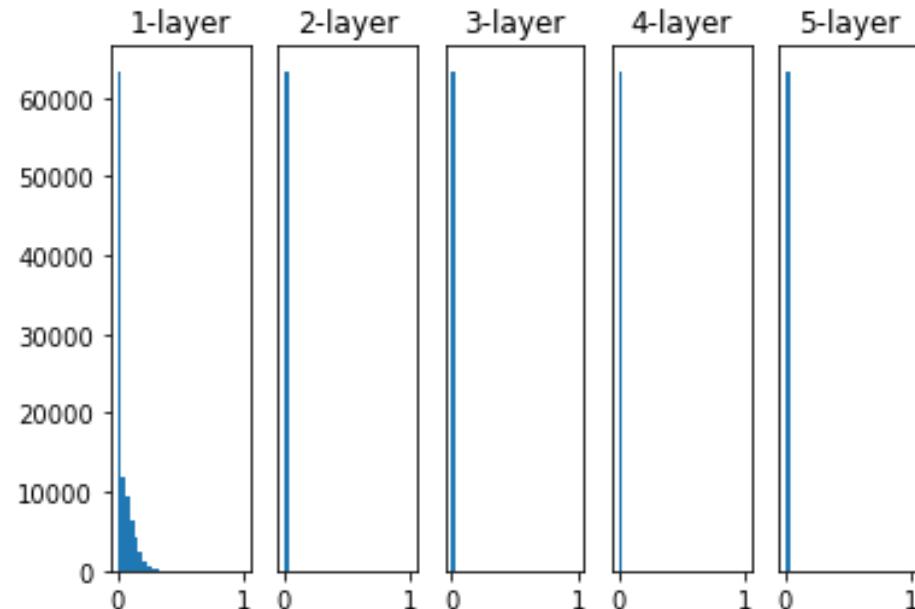


Xavier Glorot : 사비에르 글로로트

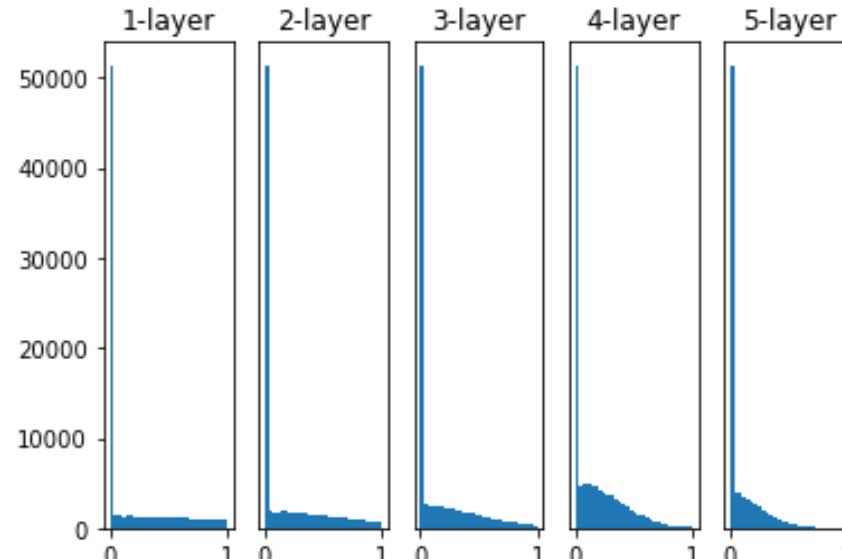


표준편차가 $\sqrt{\frac{1}{n}}$ 의
정규분포 초기화

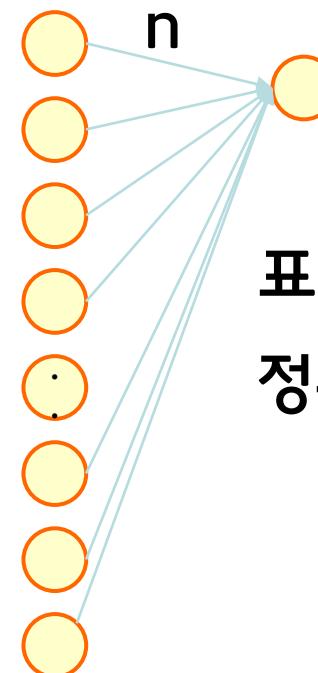
```
w = np.random.randn(node_num, node_num) * 0.01
```



```
w = np.random.randn(node_num, node_num) * np.sqrt(1.0 / node_num)
```

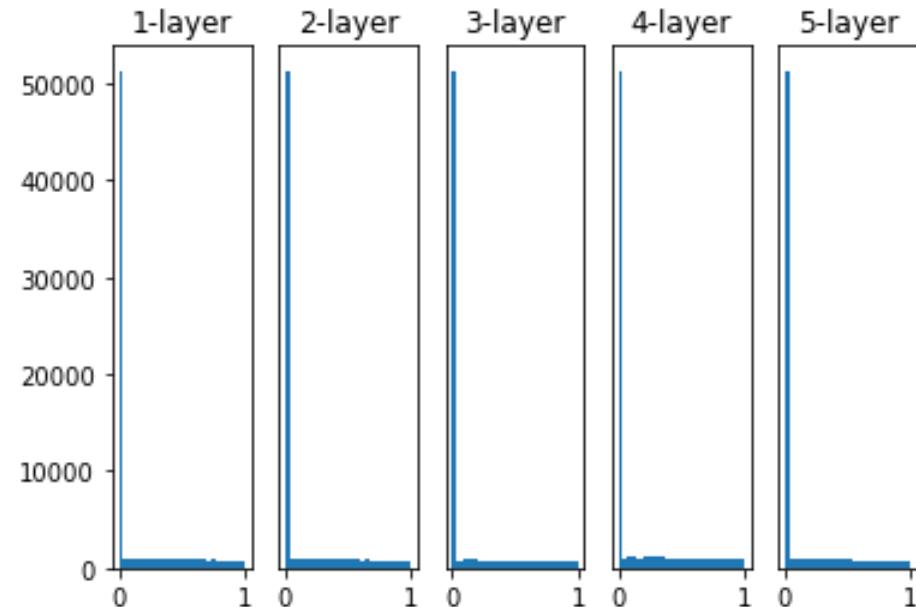


Xavier Glorot
: 사비에르 글로로트

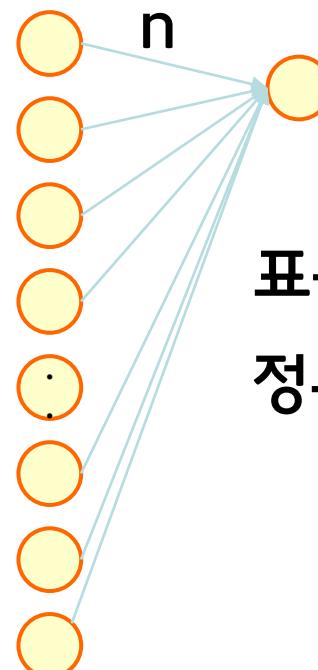


표준편차가 $\sqrt{\frac{1}{n}}$ 의
정규분포 초기화

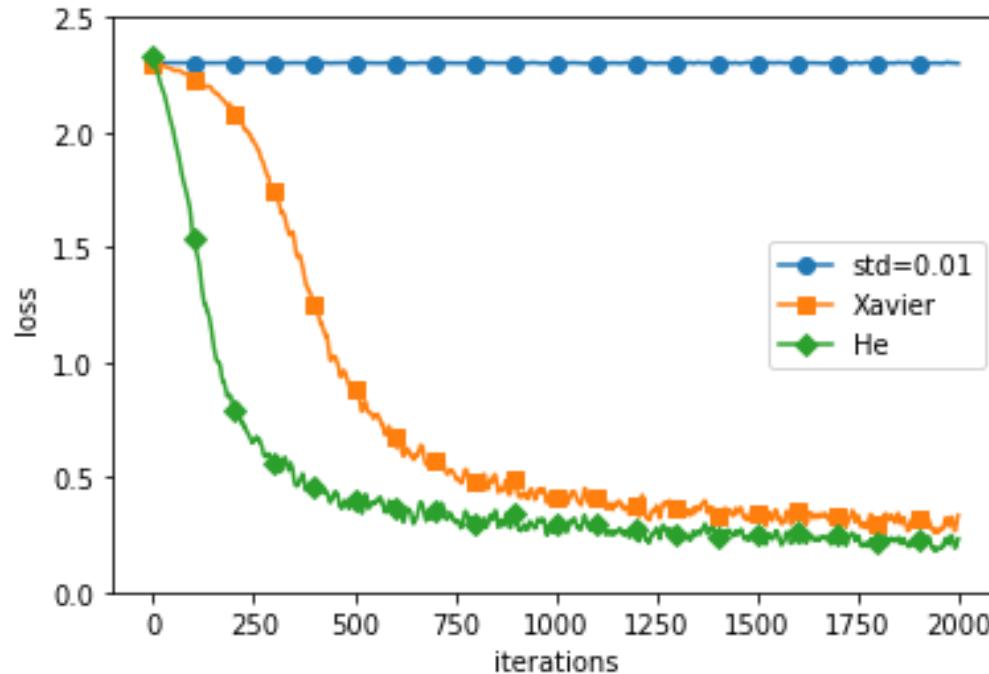
```
w = np.random.randn(node_num, node_num) * np.sqrt(2.0 / node_num)
```



He 초기화
: 카이밍 히(Kaiming He)



표준편차가 $\sqrt{\frac{2}{n}}$ 의
정규분포 초기화



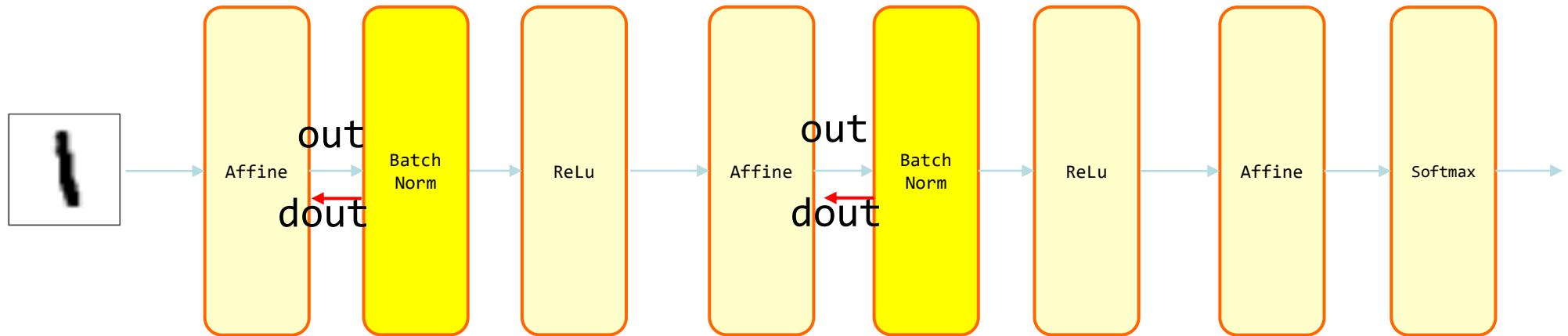
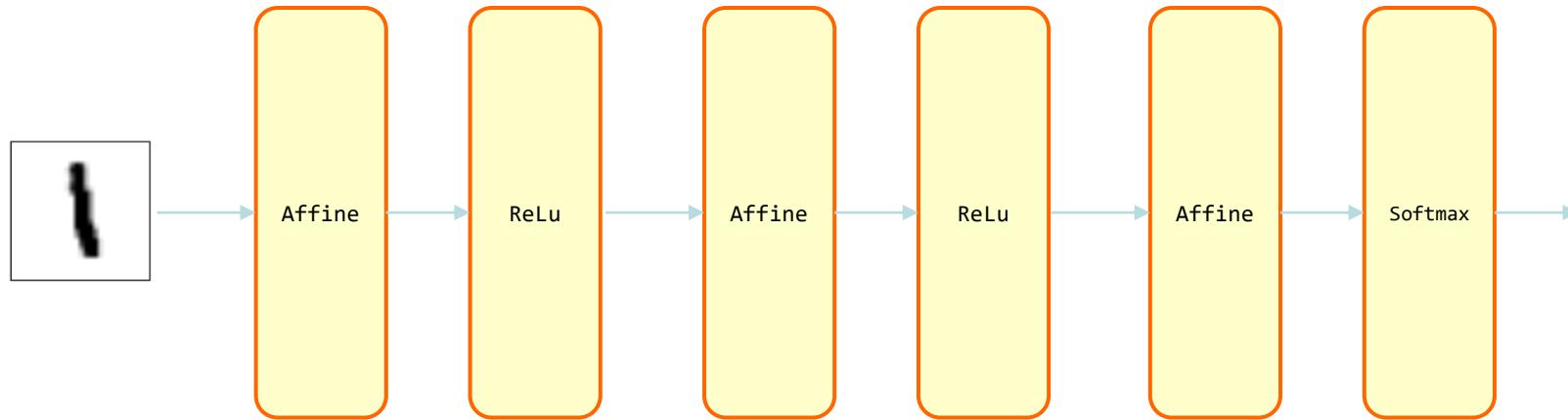
6. 학습 관련 기술들

6.1 매개변수 갱신

6.2 가중치의 초기값

6.3 배치 정규화

6.4 바른 학습을 위해



$$\mu_B \rightarrow \frac{1}{m} \sum_{i=1}^m x_i$$

```
mu = x.mean(axis=0)
```

$$\sigma_B^2 \rightarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

```
xc = x - mu
var = np.mean(xc**2, axis=0)
```

$$\hat{x}_i \rightarrow \frac{(x_i - \mu_B)}{\sqrt{\sigma_B^2 + \varepsilon}}$$

```
std = np.sqrt(var + 10e-7)
xn = xc / std
```

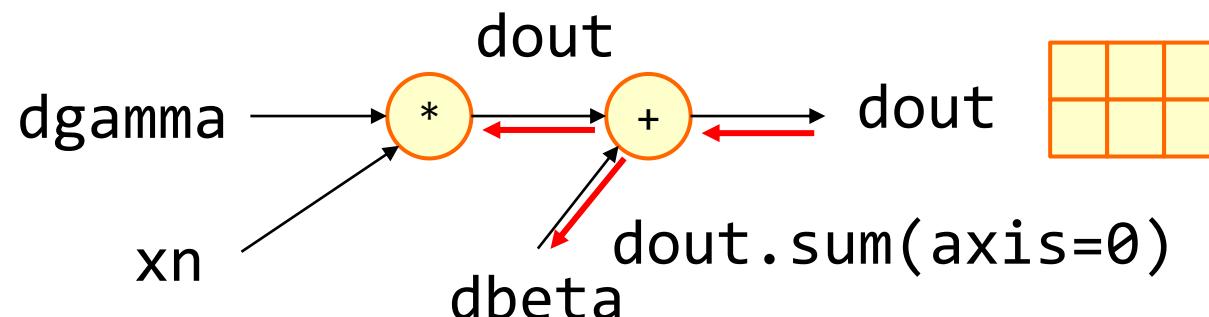
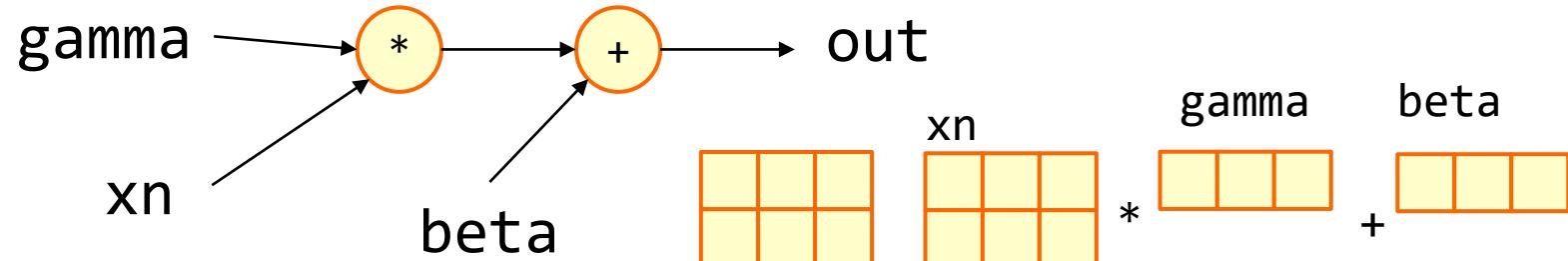
$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

```
out = self.gamma * xn + self.beta
```

```
out = self.gamma * xn + self.beta
```

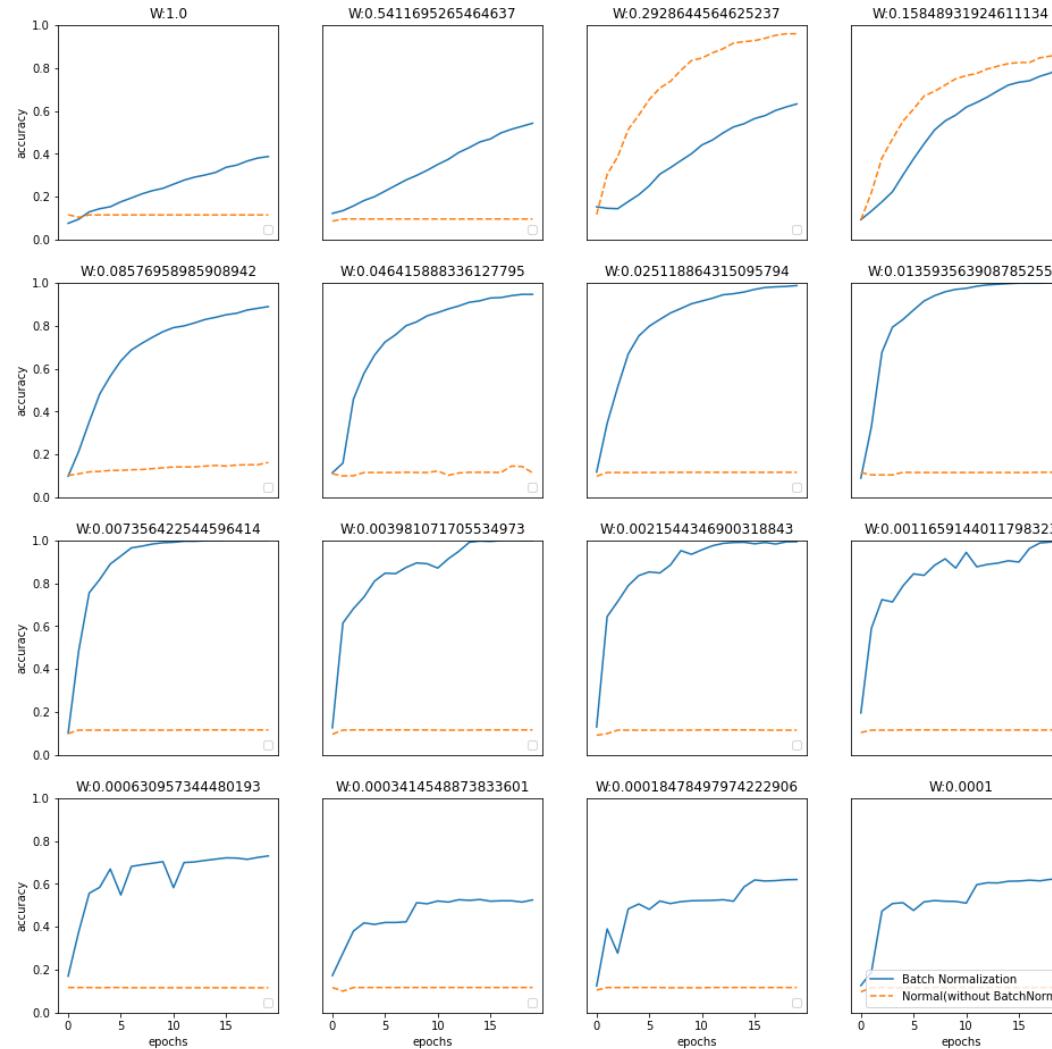
$$y_i \leftarrow \gamma \hat{x}_i + \beta$$

```
dgamma = np.sum(self.xn * dout, axis=0)
```



$\theta \sim -4$

4/15



6. 학습 관련 기술들

6.1 매개변수 갱신

6.2 가중치의 초기값

6.3 배치 정규화

6.4 바른 학습을 위해

$$L = \text{crossentropy}() + \frac{1}{2} \lambda \sum_{i=1}^k W_i^2$$

norm

L2 규제는 손실을 계산할 때 누적된다.

$$\frac{1}{2} \lambda (w_1^2 + w_2^2 + w_3^2)$$

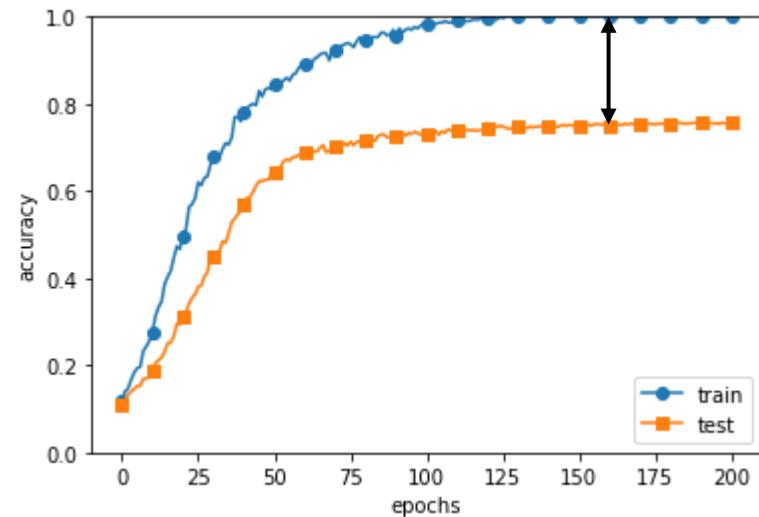
```
weight_decay += 0.5 * self.weight_decay_lambda * np.sum(W ** 2)
```

$$\lambda(W_1) = \frac{dL}{dw_1} \frac{1}{2} \lambda (w_1^2 + w_2^2 + w_3^2)$$

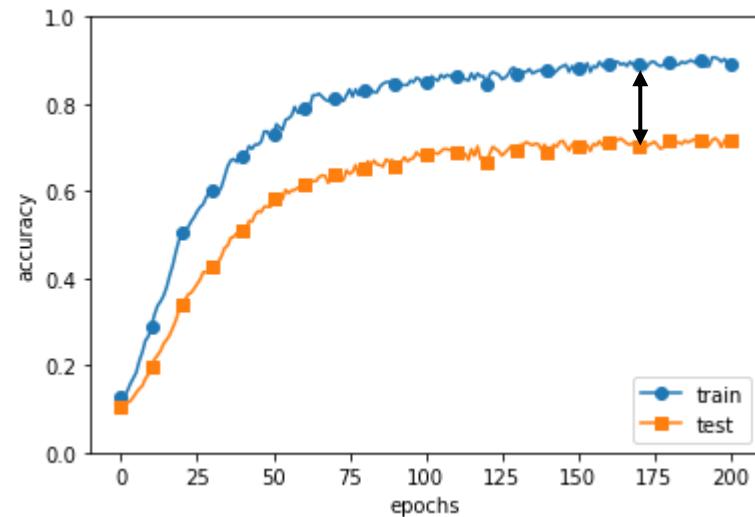
$$W_1 = W_1 - \eta \left(\frac{dL}{dw_1} + \lambda(W_1) \right)$$

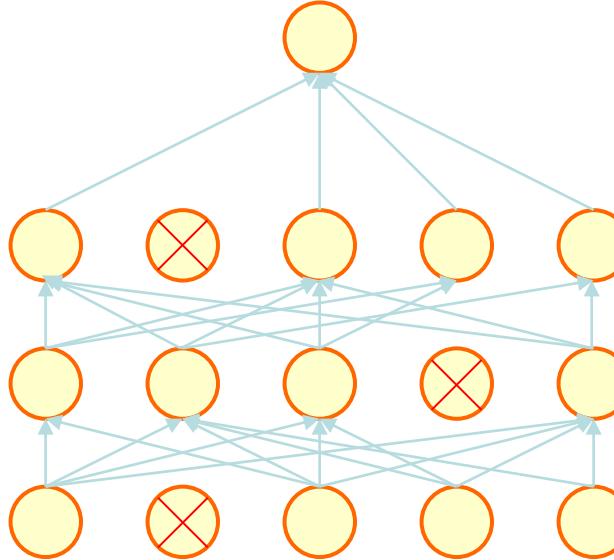
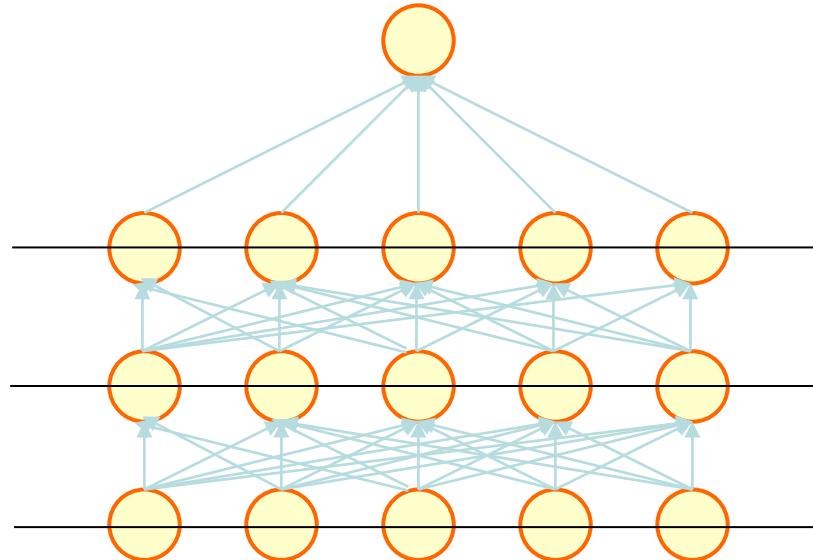
```
grads['W' + str(idx)] = self.layers['Affine' + str(idx)].dw
+ self.weight_decay_lambda * self.layers['Affine' + str(idx)].W
```

과대적합 => over fit

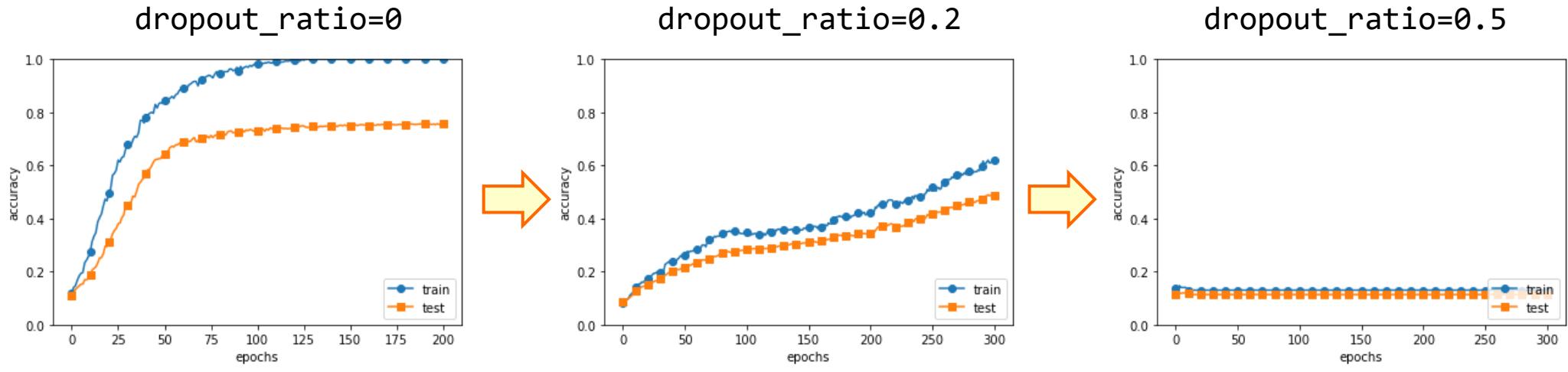


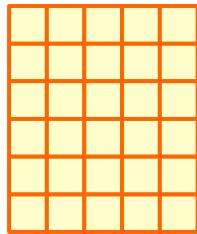
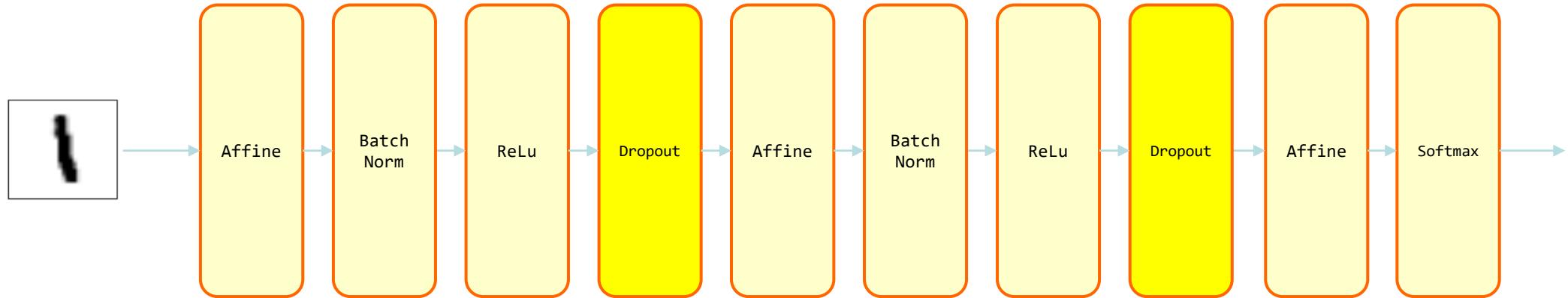
규제 적용 후 모습



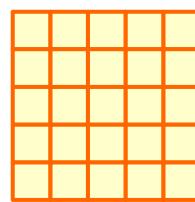


20%

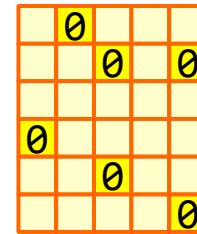




(6, 5)



(5, 5)



(6, 5)

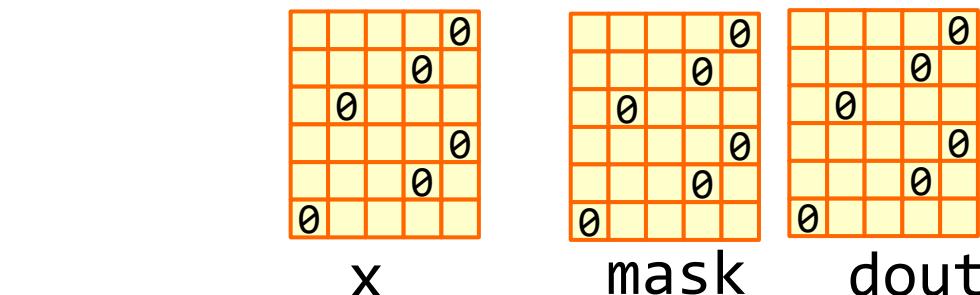
```

class Dropout:
    def __init__(self, dropout_ratio=0.5):
        self.dropout_ratio = dropout_ratio
        self.mask = None

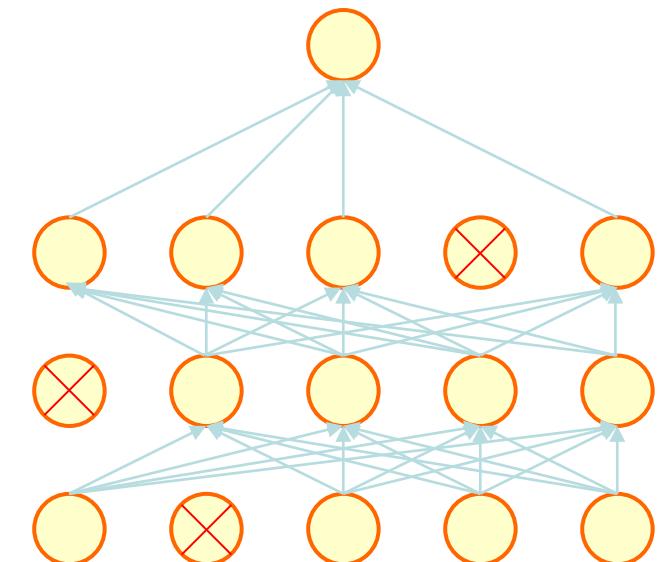
    def forward(self, x, train_flg=True):
        if train_flg:
            self.mask = np.random.rand(*x.shape) > self.dropout_ratio
            return x * self.mask
        else:
            return x * (1.0 - self.dropout_ratio)

    def backward(self, dout):
        return dout * self.mask

```



`dropout_ratio = 0.2`



7. 합성곱 신경망

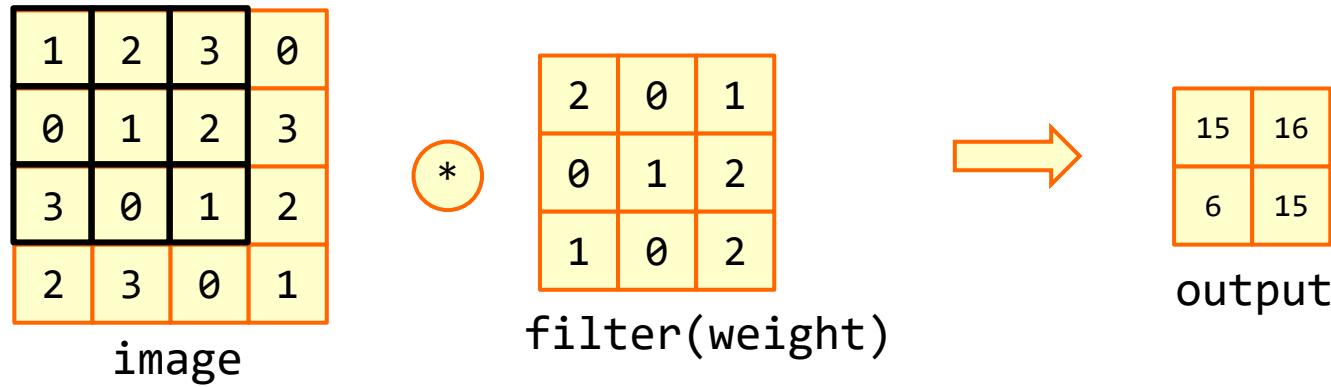
7.1 합성곱 연산

7.2 풀링 계층

7.3 합성곱/풀링 구현

7.4 CNN 구현

7.5 합성곱 신경망 시각화



$$1 * 2 + 2 * 0 + 3 * 1 + 0 * 0 + 1 * 1 + 2 * 2 + 3 * 1 + 0 * 0 + 1 * 2 = 15$$

내적

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1

*

2	0	1
0	1	2
1	0	2

+

3



18	19
9	18

입력 데이터

필터

weight

편향

bias

출력 데이터

$$N - F + 1 = 2$$

$$4 - 3 + 1 = 2$$

$$N + 2P - F + 1 = 4$$

0	0	0	0	0	0
0	1	2	3	0	0
0	0	1	2	3	0
0	3	0	1	2	0
0	2	3	0	1	0
0	0	0	0	0	0

입력 데이터

*

2	0	1
0	1	2
1	0	2

필터



7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

출력 데이터

stride=1

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	0
0	1



1	2	3
0	1	2
3	0	1

stride=2

1	2	3	0
0	1	2	3
3	0	1	2
2	3	0	1



2	0
0	1



1	2
0	1

입력 데이터

필터

출력 데이터

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

OH : 출력 높이

H : 입력 높이

P : 패딩

FH : 필터 높이

S : Stride



$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

OH : 출력 높이
 H : 입력 높이
 P : 패딩
 FH : 필터 높이
 S : Stride

입력:(4,4), 패딩:1, Stride:1, 필터:(3,3)

$$OH = \frac{4 + 2 * 1 - 3}{1} + 1$$

$$OW = \frac{4 + 2 * 1 - 3}{1} + 1$$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

OH : 출력 높이
 H : 입력 높이
 P : 패딩
 FH : 필터 높이
 S : Stride

입력:(7,7), 패딩:0, Stride:2, 필터:(3,3)

$$OH = \frac{7 + 2 * 0 - 3}{2} + 1$$

$$OW = \frac{7 + 2 * 0 - 3}{2} + 1$$

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

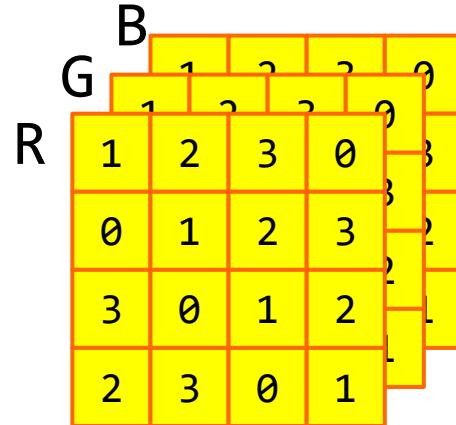
OH : 출력 높이
 H : 입력 높이
 P : 패딩
 FH : 필터 높이
 S : Stride

입력:(28,31), 패딩:2, Stride:3, 필터:(5,5)

$$OH = 10 = \frac{28 + 2 * 2 - 5}{3} + 1$$

$$OW = 11 = \frac{31 + 2 * 2 - 5}{3} + 1$$

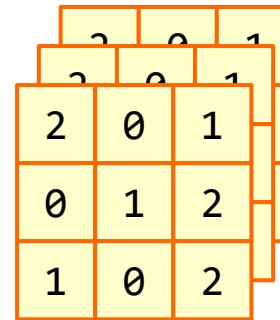
channel = (R, G, B)



(3, 4, 4)

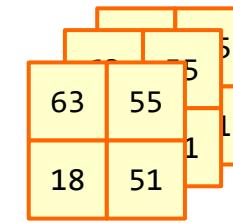
입력 데이터

*



(3, 3, 3)

필터



(1, 2, 2)

출력 데이터

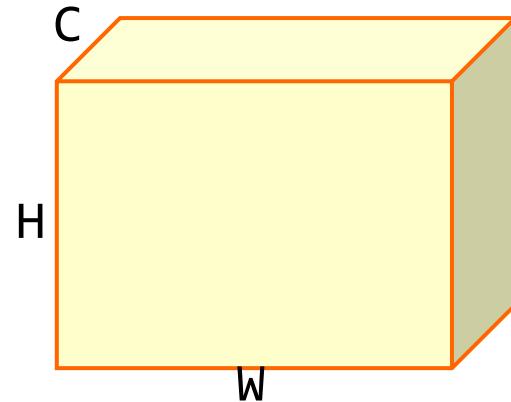
height

(3, 448, 640)

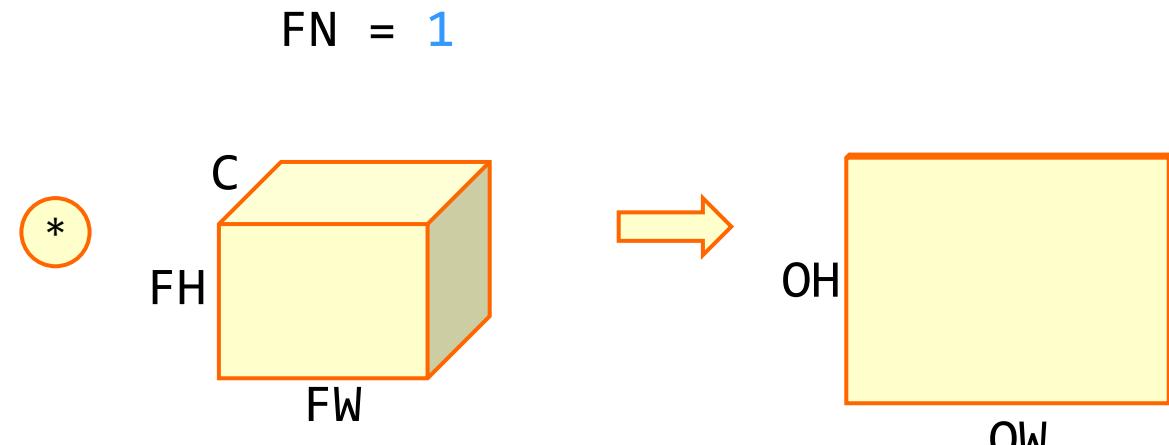
(C, H, W)

width





입력 데이터

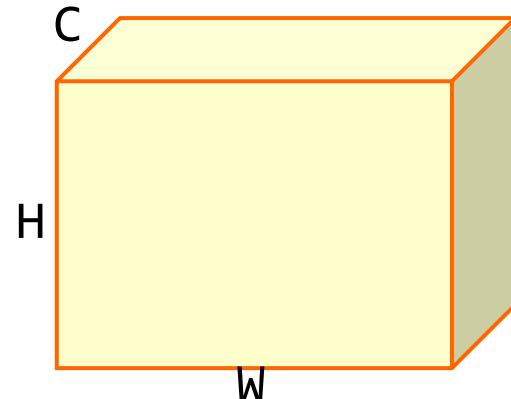
 (C, H, W) 

필터

 (C, FH, FW)

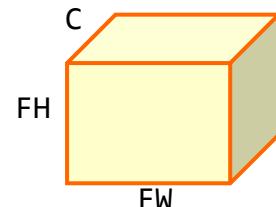
출력 데이터

 $(1, OH, OW)$

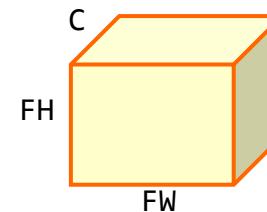


입력 데이터
(C, H, W)

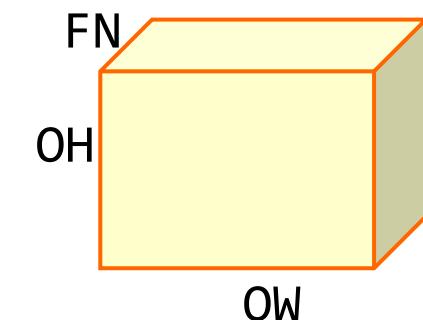
*



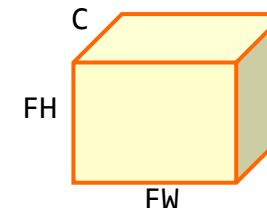
FN : 필터의 수



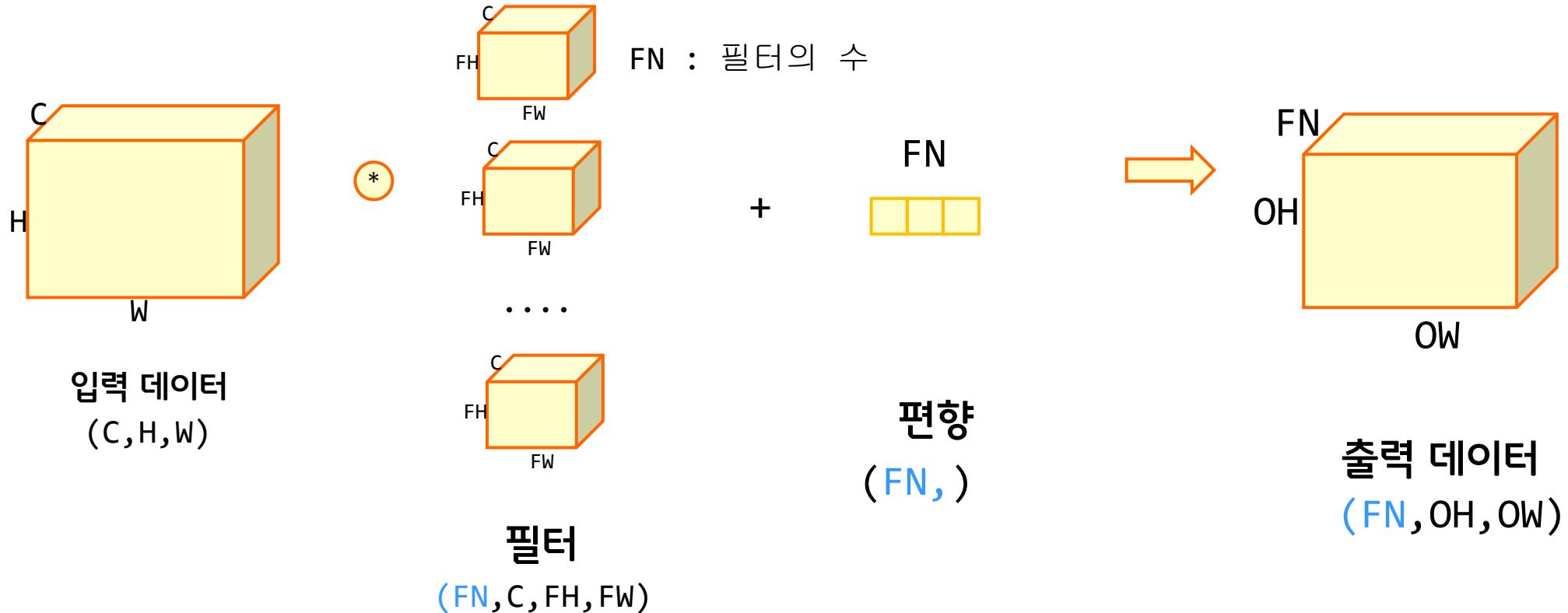
....

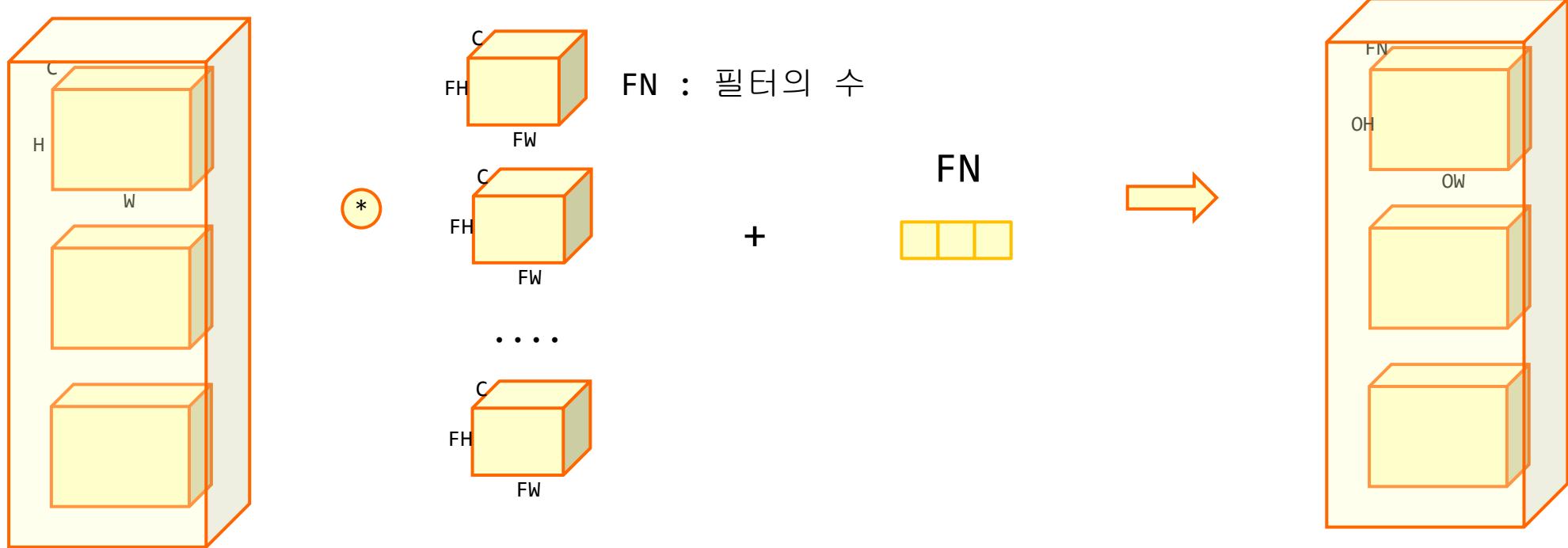


출력 데이터
(FN, OH, OW)



(FN, C, FH, FW)





N개의 입력 데이터
(**N**, C, H, W)

필터
(FN, C, FH, FW)

편향
($FN, ,$)

출력 데이터
(**N**, FN , OH, OW)

7. 합성곱 신경망

7.1 합성곱 연산

7.2 풀링 계층

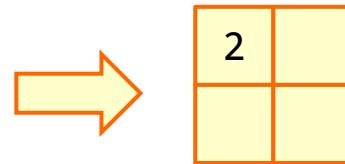
7.3 합성곱/풀링 구현

7.4 CNN 구현

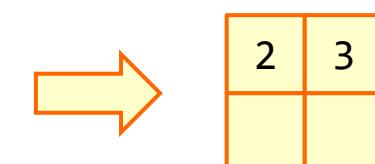
7.5 합성곱 신경망 시각화

필터의 크기가 2이면 stride도 2이다.

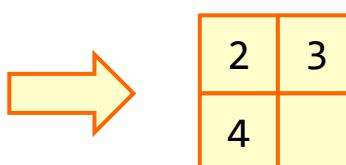
1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



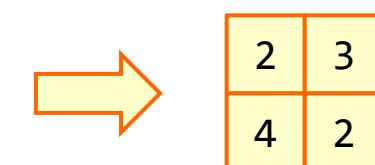
1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



1	2	1	0
0	1	2	3
3	0	1	2
2	4	0	1



학습해야 할 매개 변수가 없다.

그림 수와 채널 수가 변하지 않는다.

1	2	3	2	3	0
0	1	2	3	2	3
3	0	1	2	1	2
2	3	0	1	-	-
2	3	0	1	-	-

입력 데이터



2	3
3	0
2	3

출력 데이터

입력의 변화에 영향을 적게 받는다.

1	2	0	7	1	0
0	9	2	3	2	3
3	0	1	2	1	2
2	4	0	1	0	1
6	0	1	2	1	2
2	3	0	1	8	1

9	7
6	8

1	2	0	2	1	0
0	2	9	3	7	3
3	0	1	2	1	2
2	4	0	1	0	1
0	6	1	2	1	8
2	3	0	1	2	1

9	7
6	8

7. 합성곱 신경망

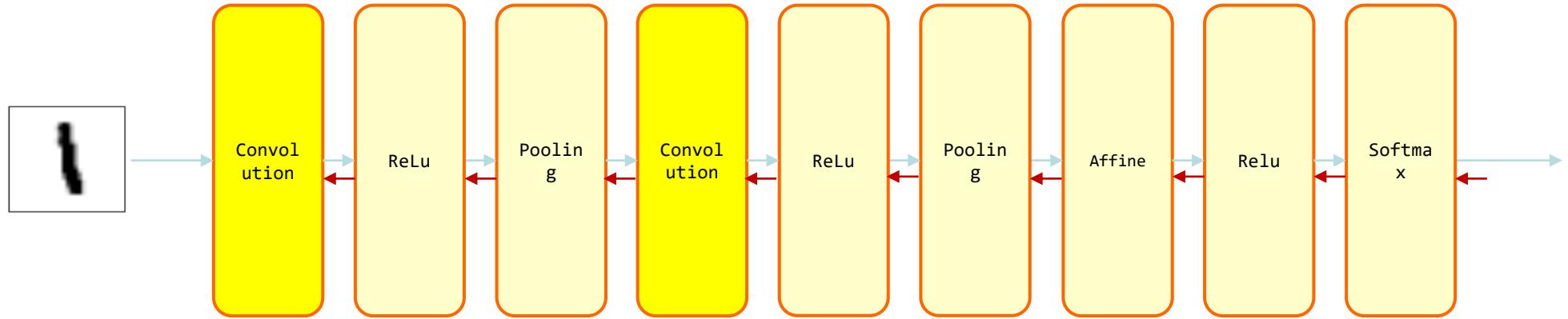
7.1 합성곱 연산

7.2 풀링 계층

7.3 합성곱/풀링 구현

7.4 CNN 구현

7.5 합성곱 신경망 시각화



```

import sys, os
sys.path.append(os.pardir)
import numpy as np
from common.layers import Convolution

image = np.arange(16).reshape(1,1,4,4)
print(image.shape)
print(image)

W = np.ones((1,1,2,2))
print(W)
b = np.full((1,), 3)
print(b.shape)
print(b)
conv = Convolution(W, b)
out = conv.forward(image)
print(out.shape)
print(out)

```

image

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$*$

W

1	1
1	1

b

3

(1, 1, 2, 2)

+

(1,)

=

(1, 1, 3, 3)

[[[[13. 17. 21.]
[29. 33. 37.]
[45. 49. 53.]]]]

Convolution.forward(x)

```

FN, C, FH, FW = self.W.shape # (1, 1, 2, 2)
N, C, H, W = x.shape       # (1, 1, 4, 4)
out_h = 1 + int((H + 2*self.pad - FH) / self.stride)
out_w = 1 + int((W + 2*self.pad - FW) / self.stride)
    
```

$$OH = \frac{H + 2P - FH}{S} + 1$$

필터

(FN, C, FH, FW)

width

N개의 입력 데이터

(N, C, H, W) height

(1, 3, 448, 640)

(N, C, H, W)



(447, 639)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(1, 1, 4, 4)

1	1
1	1

(1, 1, 2, 2)

3

$$[[[[13, 17, 21], [29, 33, 37], [45, 49, 53]]]]$$

13	17	21
29	33	37
45	49	53

(1,)

13	17	21
29	33	37
45	49	53

(1, 1, 3, 3)

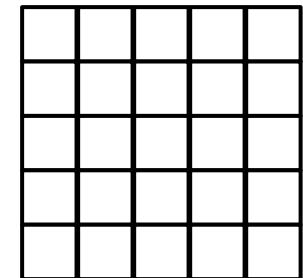
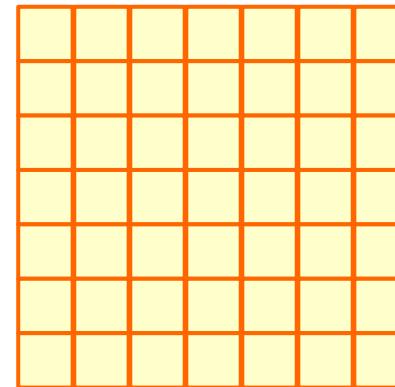
```
def forward(self, x):
    ...
    col = im2col(x, FH, FW, self.stride, self.pad)
    ...
```

im2col 테스트

```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from common.util import im2col

x1 = np.random.rand(1, 1, 7, 7)
col1 = im2col(x1, 5, 5, stride=1, pad=0 )
print(col1.shape) # (9, 25) ( N*OH*OW, C*FH*FW)

x2 = np.random.rand(10, 3, 7, 7)
col2 = im2col(x2, 5, 5, stride=1, pad=0 )
print(col2.shape) # (90, 75) ( N*OH*OW, C*FH*FW)
```



$$OH = H - FH + 1$$

$$OW = W - FW + 1$$

```
def im2col(input_data, filter_h, filter_w, stride=1, pad=0):
    N, C, H, W = input_data.shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1

    img = np.pad(input_data, [(0,0), (0,0), (pad, pad), (pad, pad)], 'constant')
    col = np.zeros((N, C, filter_h, filter_w, out_h, out_w)) # (1,1,2,2,3,3)

    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]

    col = col.transpose(0, 4, 5, 1, 2, 3).reshape(N*out_h*out_w, -1)
    return col
```

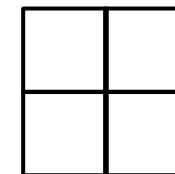
```
image = np.arange(16).reshape(1,1,4,4)
```

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

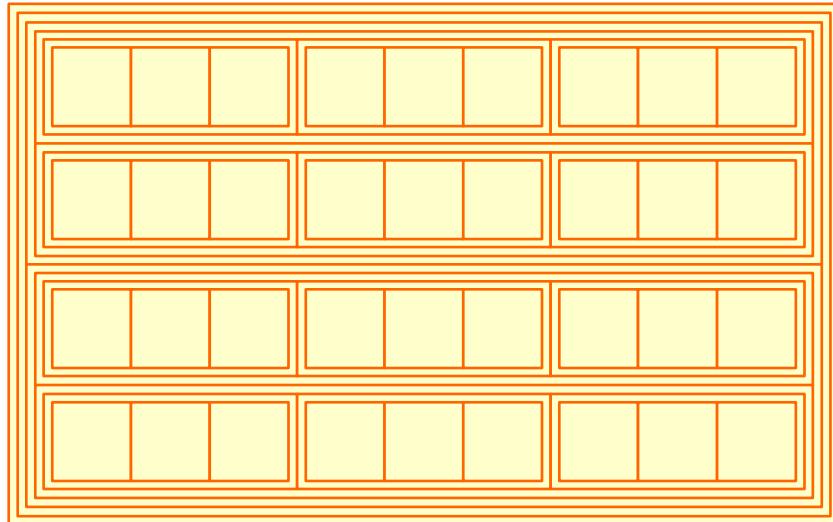
```
filter = np.ones(1,1,2,2)
```

1	1
1	1



```
col = np.zeros((N=1, C=1, filter_h=2, filter_w=2, out_h=3, out_w=3))
```

col



```

for y in range(filter_h):
    y_max = y + stride*out_h
    for x in range(filter_w):
        x_max = x + stride*out_w
        col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
    
```

```

col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]
    
```

width



height
(3, 448, 640)
(C, H, W)

(1, 3, 2, 2) (1, 3, 447, 639)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(1, 1, 2, 2, 3, 3)

(1, 1, 3, 3)

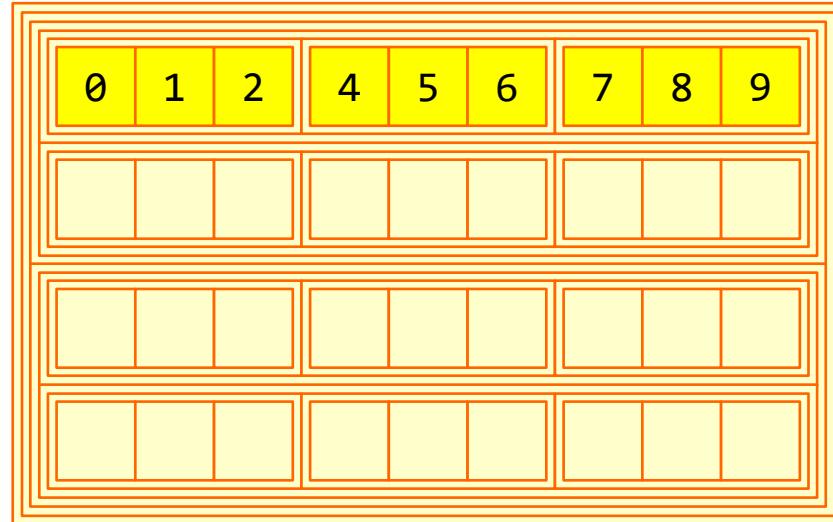
```

col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]
    
```

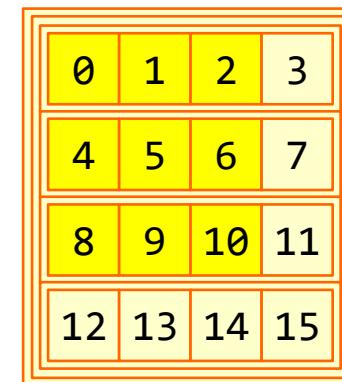
(1, 1, 4, 4)

(1, 1, 3, 3)

col



img



(1, 1, 2, 2, 3, 3)

(1, 1, 3, 3)

```

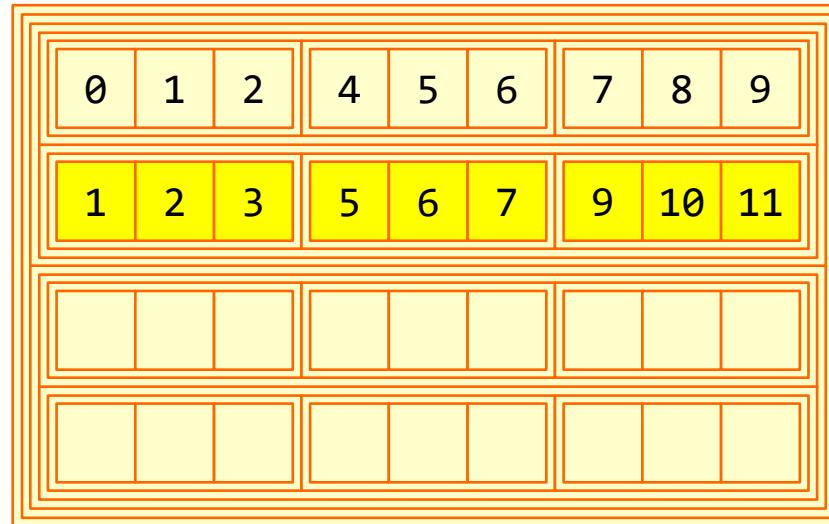
col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

```

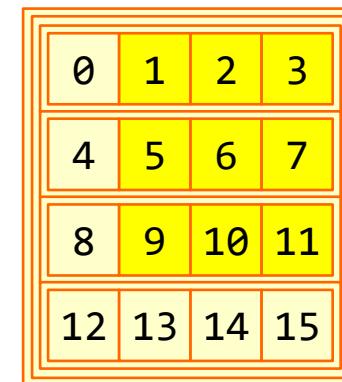
(1, 1, 4, 4)

(1, 1, 3, 3)

col



img



(1, 1, 2, 2, 3, 3)

(1, 1, 3, 3)

```

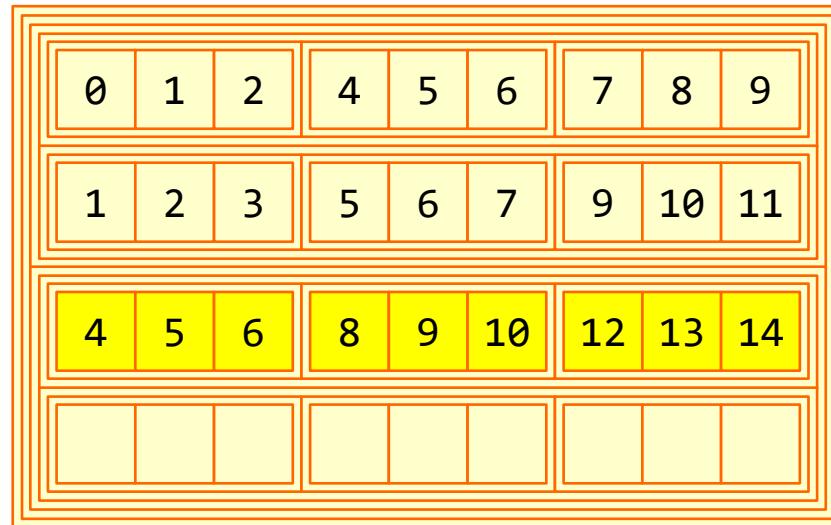
col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

```

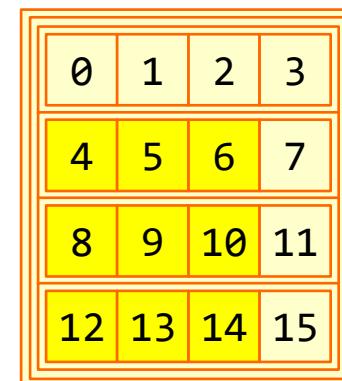
(1, 1, 4, 4)

(1, 1, 3, 3)

col



img



(1, 1, 2, 2, 3, 3)

(1, 1, 3, 3)

```

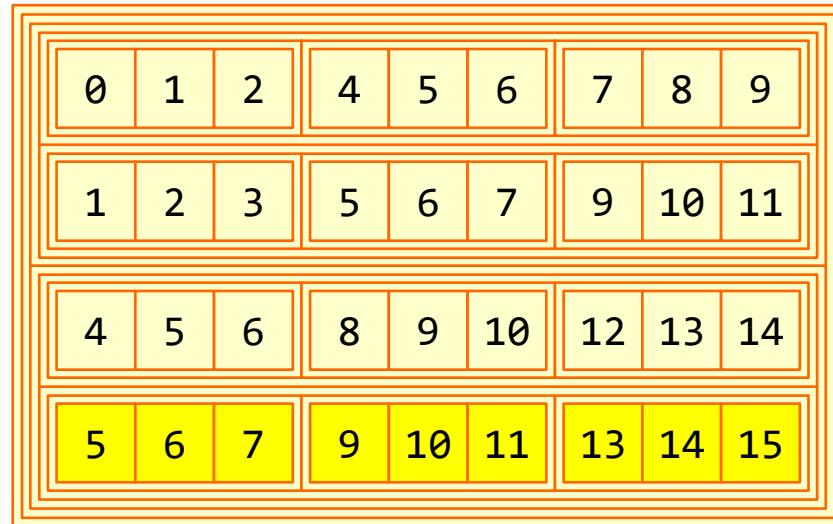
col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

```

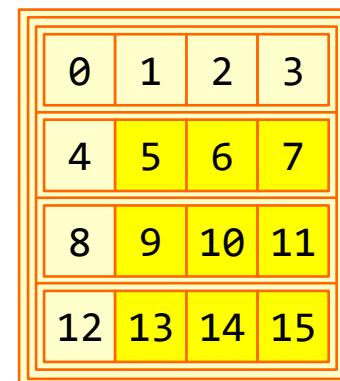
(1, 1, 4, 4)

(1, 1, 3, 3)

col

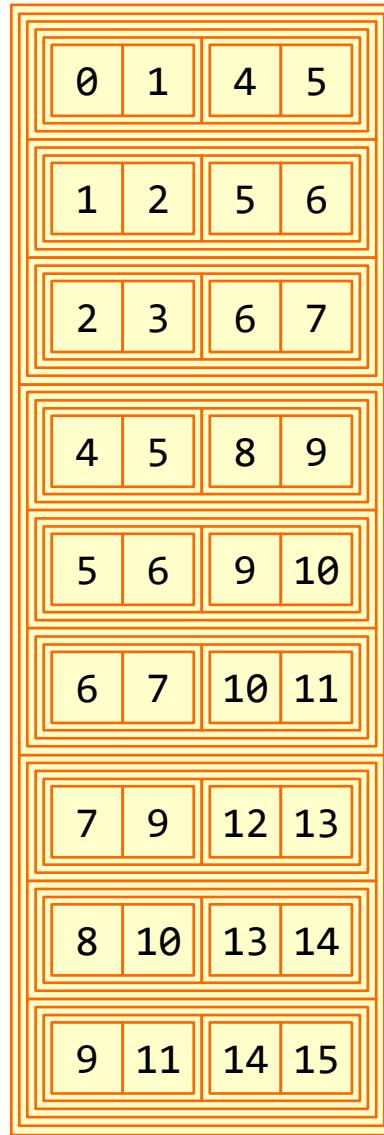


img



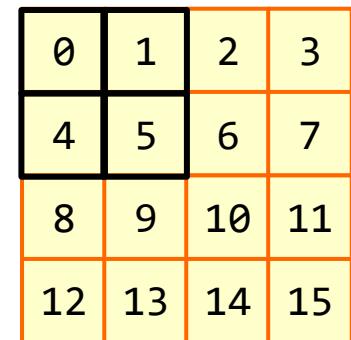
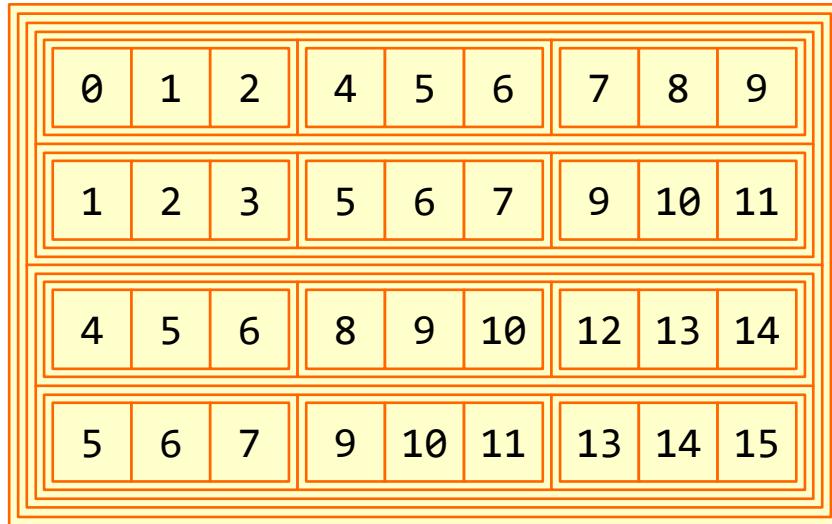
col

`col.col.transpose(0, 4, 5, 1, 2, 3)`

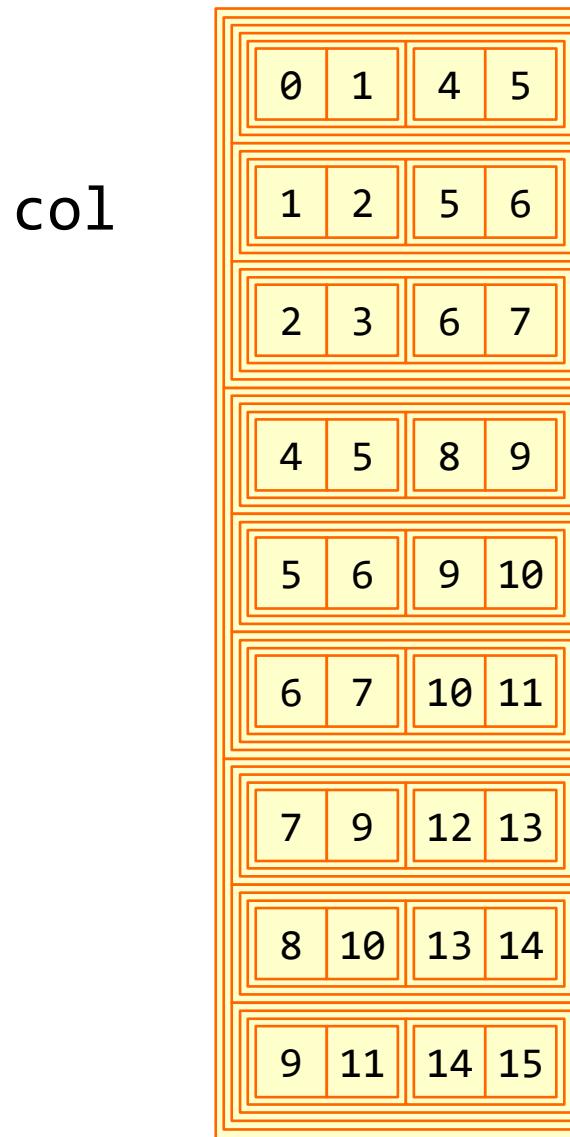


(1, 1, 2, 2, 3, 3)
 (1, 3, 3, 1, 2, 2)
 (N*OH*OW, C*FH*FW)

col



`col.reshape(N*out_h*out_w, -1) (1*3*3, 1*2*2) => (9,4)`



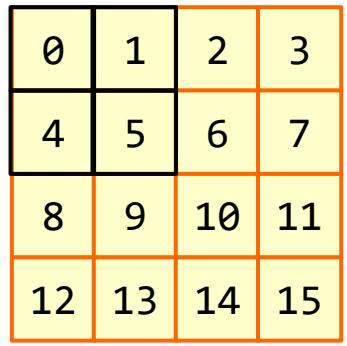
col

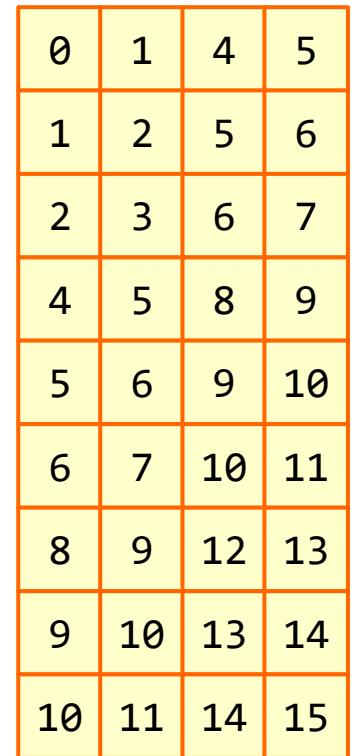
0	1	4	5
1	2	5	6
2	3	6	7
4	5	8	9
5	6	9	10
6	7	10	11
8	9	12	13
9	10	13	14
10	11	14	15

```
[[ 0.  1.  4.  5.]
 [ 1.  2.  5.  6.]
 [ 2.  3.  6.  7.]
 [ 4.  5.  8.  9.]
 [ 5.  6.  9.  10.]
 [ 6.  7.  10. 11.]
 [ 8.  9.  12. 13.]
 [ 9.  10. 13. 14.]
 [10. 11. 14. 15.]]
```

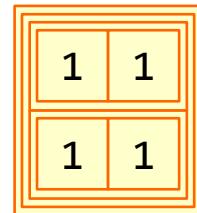
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

```
col_W = self.W.reshape(FN, -1).T
out = np.dot(col, col_W) + self.b
```

image

 $(1, 1, 4, 4)$

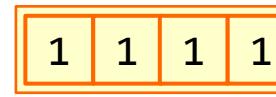
col(9, 4)


$(1, 1, 2, 2)$



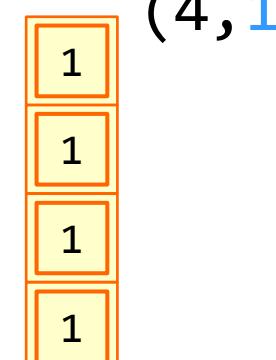
\downarrow

$(1, 4)$

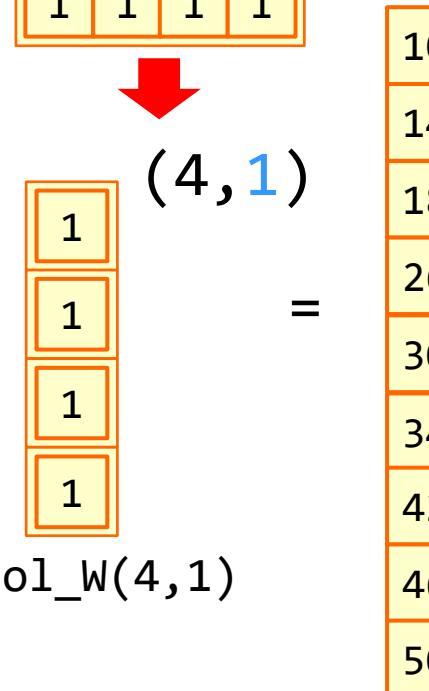


\downarrow

$(4, 1)$



\cdot
 $\text{col}_W(4, 1)$



$(9, 1)$

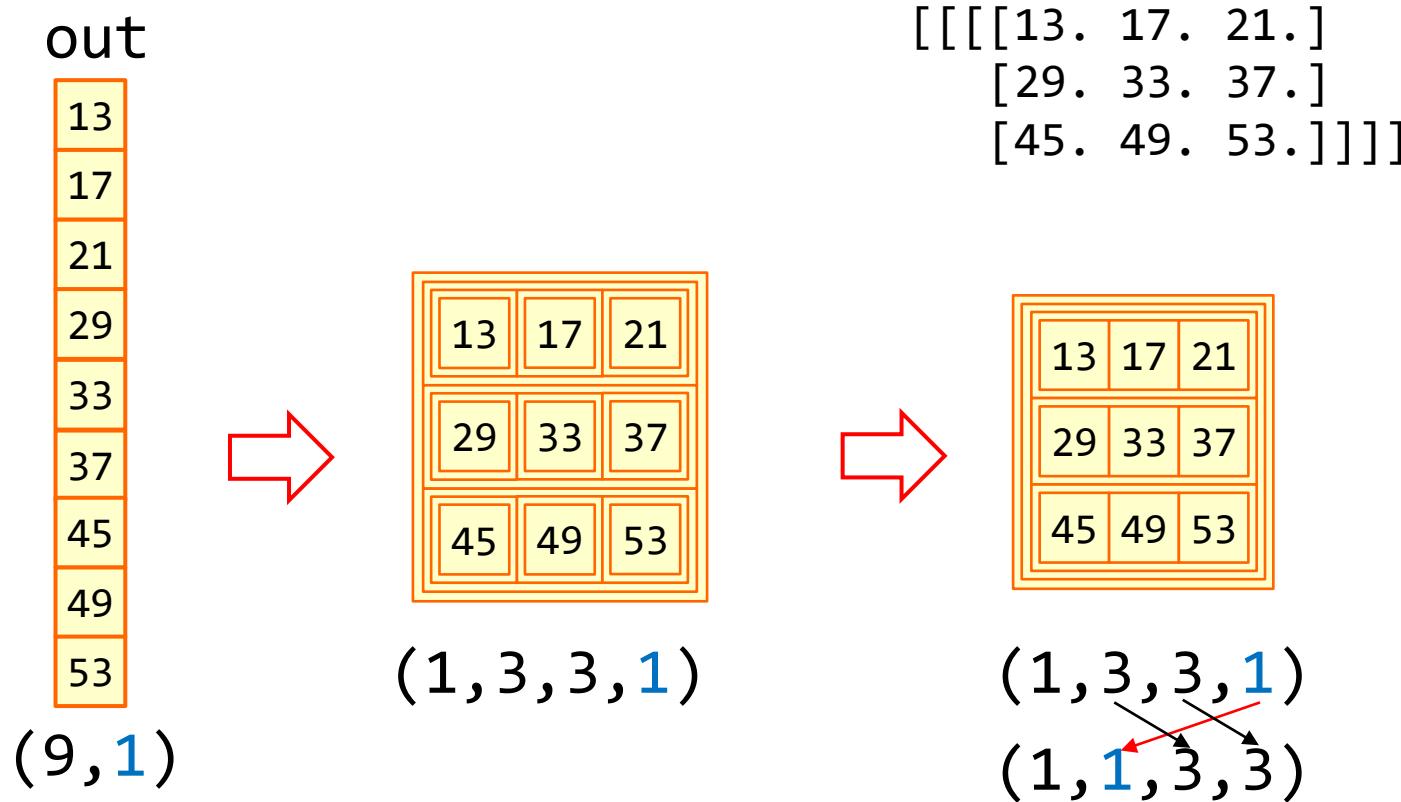
$(9, 4)(4, 1) \Rightarrow (9, 1)$

$+ \begin{matrix} 3 \end{matrix} = \begin{matrix} 13 \\ 17 \\ 21 \\ 29 \\ 33 \\ 37 \\ 45 \\ 49 \\ 53 \end{matrix}$

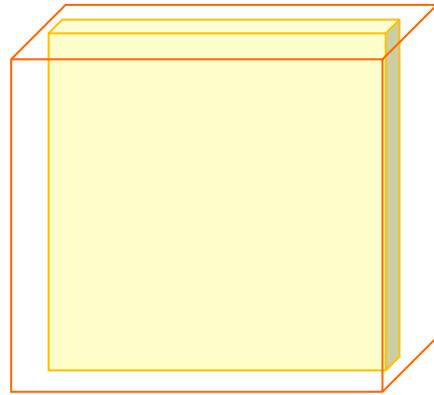
$b(1,)$

$\text{out}(9, 1)$

```
out = out.reshape(N, out_h, out_w, -1)
out = out.transpose(0, 3, 1, 2)
```



im2col

 $(1, 1, 4, 4)$ 

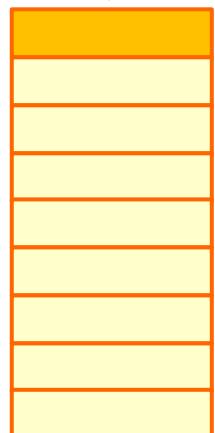
입력 데이터

 $(1 * 3 * 3, 1 * 2 * 2)$
 $(N * OH * OW, C * FH * FW)$

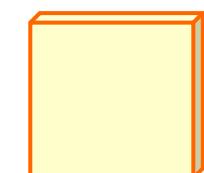

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

 $(1, 1, 2, 2)$
 (FN, C, FH, FW)

↓ im2col



•

 $(1 * 2 * 2, 1)$
 $(C * FH * FW, FN)$
 $(9, 1)$  $(1, 3, 3, 1)$  $(1, 1, 3, 3)$

img (1, 2, 4, 4)

col (1, 2, 2, 2, 3, 3)

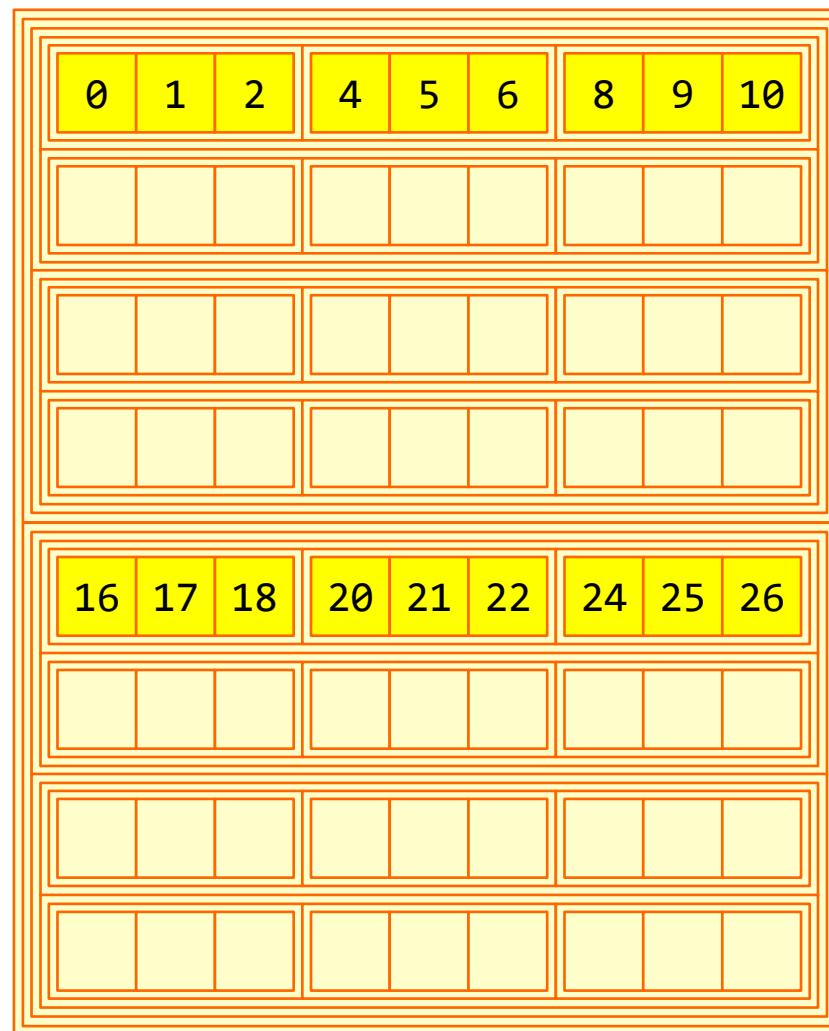
```

col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

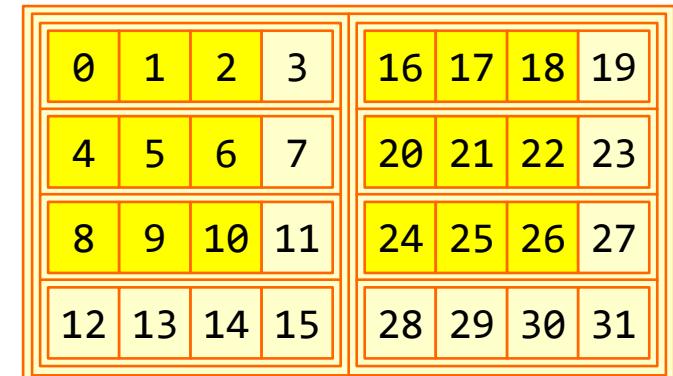
```

(1, 2, 4, 4)

(1, 2, 3, 3)



img



합성곱 구현

7.3 합성곱/풀링 구현

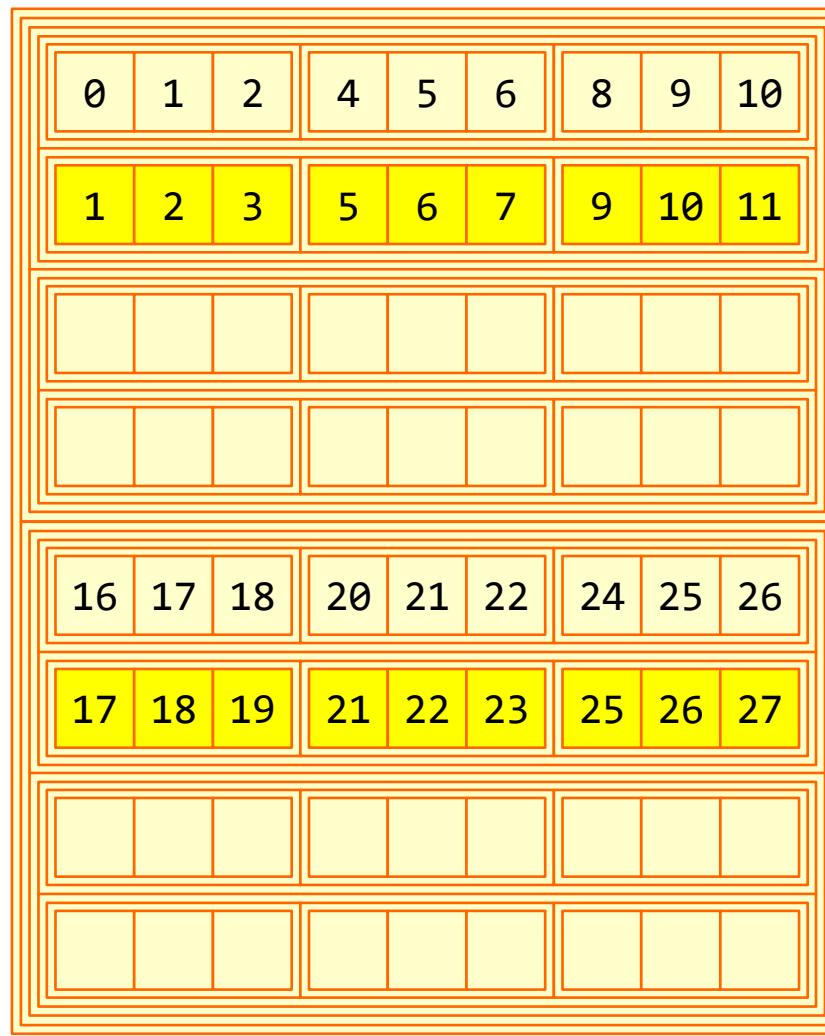
img (1, 2, 4, 4)

col (1, 2, 2, 2, 3, 3)

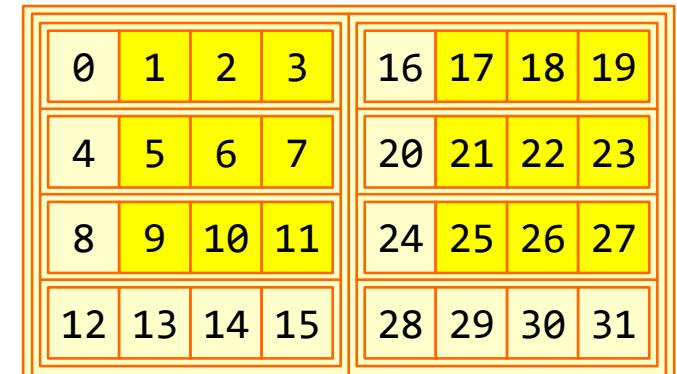
```
col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]  
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]  
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]  
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]
```

(1, 2, 4, 4)

(1, 2, 3, 3)



img



img (1, 2, 4, 4)

col (1, 2, 2, 2, 3, 3)

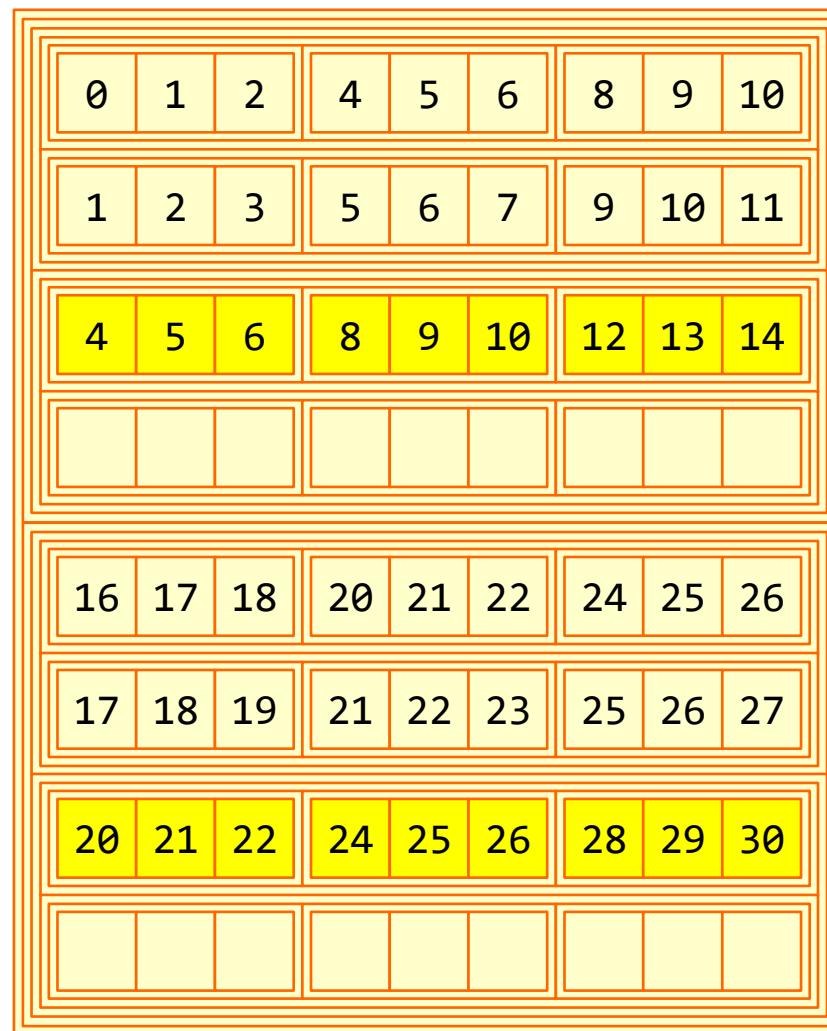
```

col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

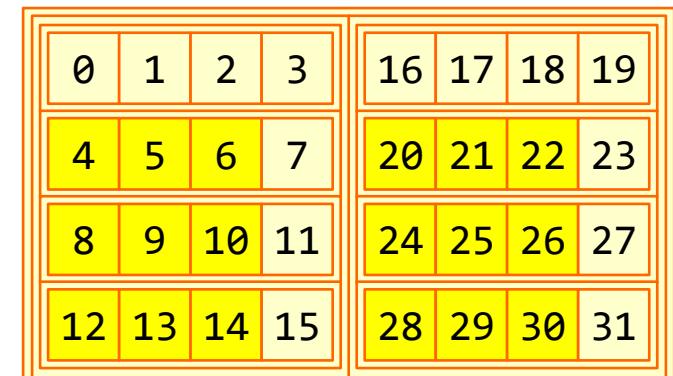
```

(1, 2, 4, 4)

(1, 2, 3, 3)



img



img (1, 2, 4, 4)

col (1, 2, 2, 2, 3, 3)

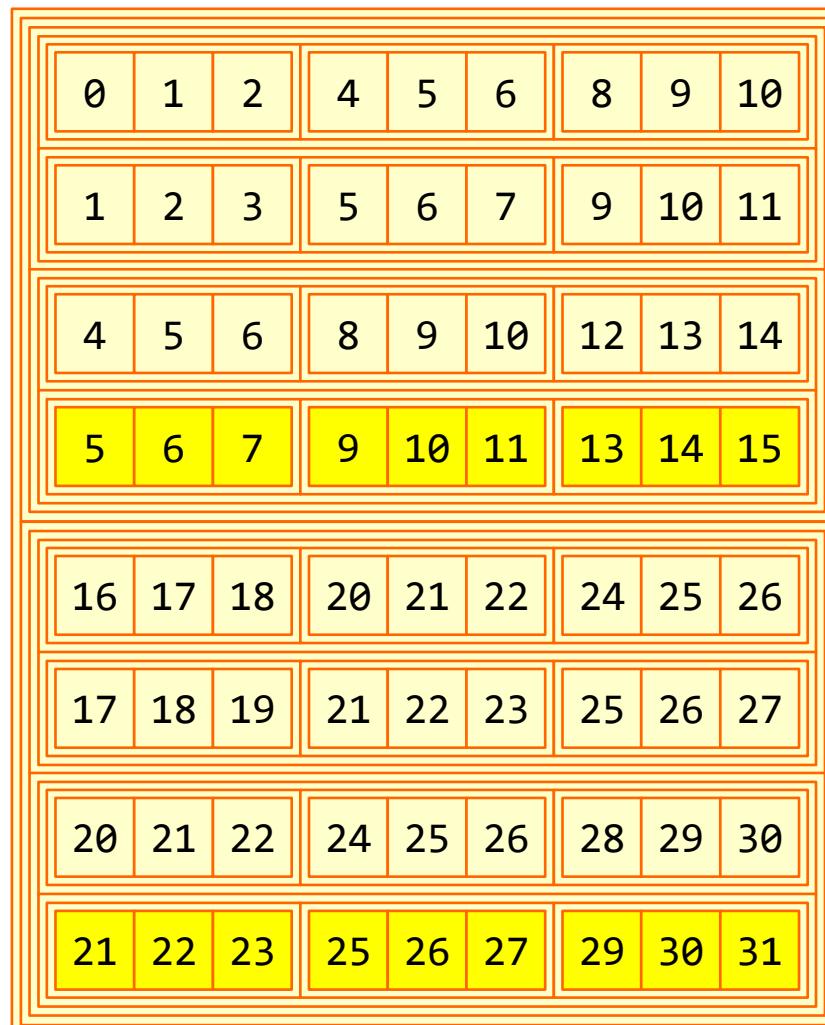
```

col[:, :, 0, 0, :, :] = img[:, :, 0:3:1, 0:3:1]
col[:, :, 0, 1, :, :] = img[:, :, 0:3:1, 1:4:1]
col[:, :, 1, 0, :, :] = img[:, :, 1:4:1, 0:3:1]
col[:, :, 1, 1, :, :] = img[:, :, 1:4:1, 1:4:1]

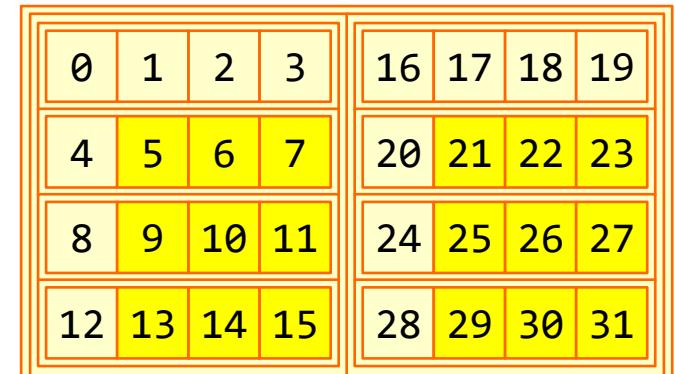
```

(1, 2, 4, 4)

(1, 2, 3, 3)



img



`col.transpose(0, 4, 5, 1, 2, 3)`

`col`

0	1	2	4	5	6	8	9	10
1	2	3	5	6	7	9	10	11
4	5	6	8	9	10	12	13	14
5	6	7	9	10	11	13	14	15
16	17	18	20	21	22	24	25	26
17	18	19	21	22	23	25	26	27
20	21	22	24	25	26	28	29	30
21	22	23	25	26	27	29	30	31

(1, 2, 2, 2, 3, 3)
~~(1, 3, 3, 2, 2, 2)~~



0	1	4	5	16	17	20	21
1	2	5	6	17	18	21	22
2	3	6	7	18	19	22	24
4	5	8	9	20	21	24	25
5	6	9	10	21	22	25	26
6	7	10	11	22	23	26	27
8	9	12	13	24	25	28	29
9	10	13	14	25	26	29	30
10	11	14	15	26	27	30	31

`col.reshape(N*OH*OW, C*FH*FW) => (1,3,3,2,2,2) => (9, 8)`

col

0	1	4	5	16	17	20	21
1	2	5	6	17	18	21	22
2	3	6	7	18	19	22	24
4	5	8	9	20	21	24	25
5	6	9	10	21	22	25	26
6	7	10	11	22	23	26	27
8	9	12	13	24	25	28	29
9	10	13	14	25	26	29	30
10	11	14	15	26	27	30	31



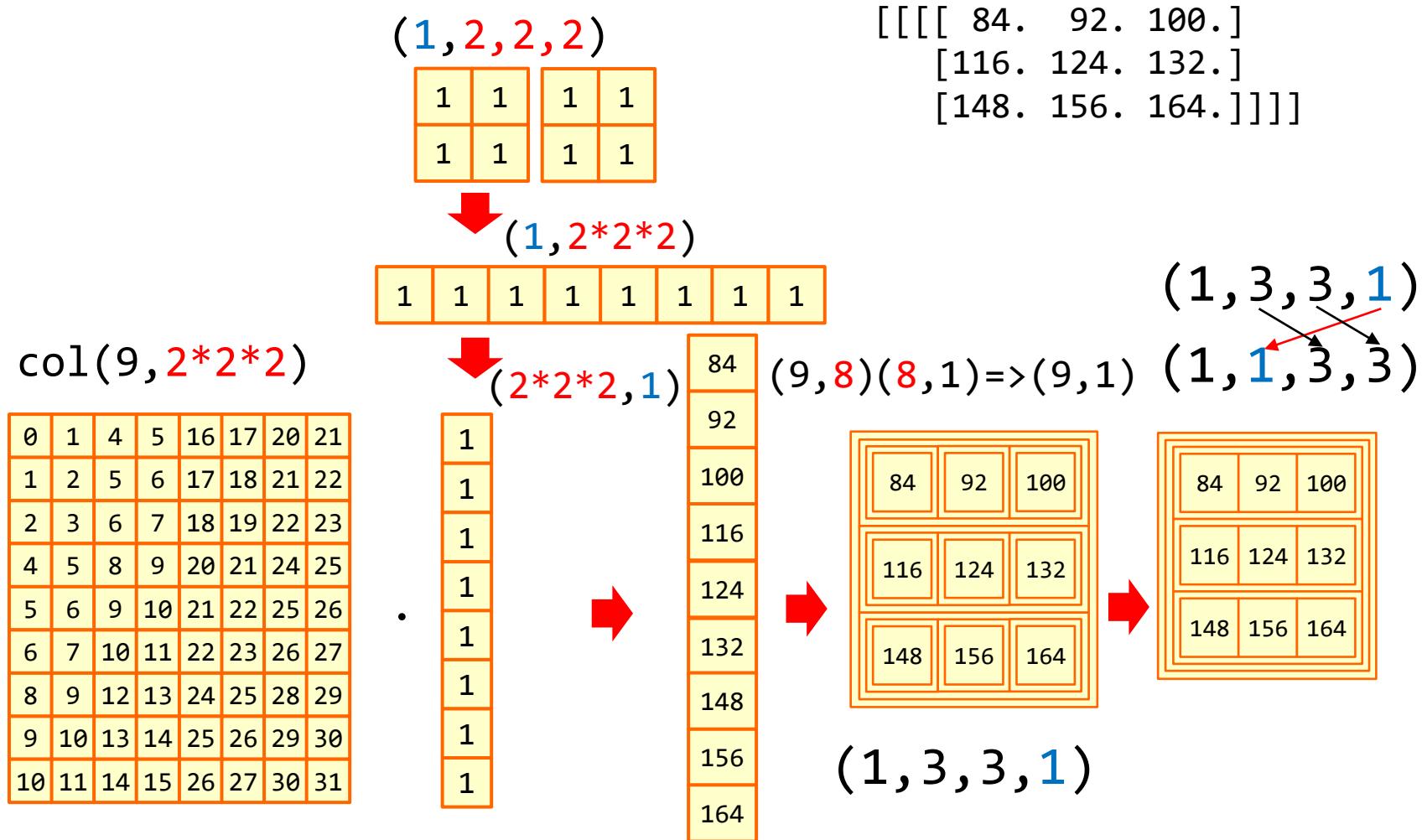
col

0	1	4	5	16	17	20	21
1	2	5	6	17	18	21	22
2	3	6	7	18	19	22	23
4	5	8	9	20	21	24	25
5	6	9	10	21	22	25	26
6	7	10	11	22	23	26	27
8	9	12	13	24	25	28	29
9	10	13	14	25	26	29	30
10	11	14	15	26	27	30	31

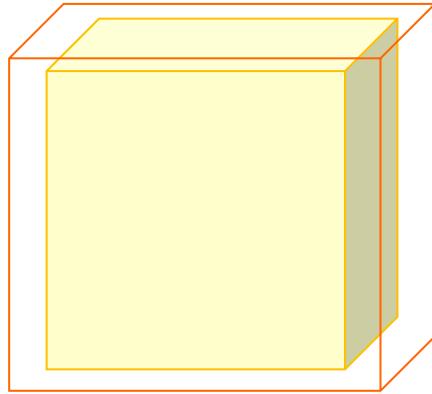
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

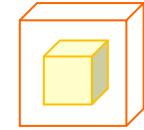
```
col_W = self.W.reshape(FN, -1).T
out = np.dot(col, col_W) + self.b
```



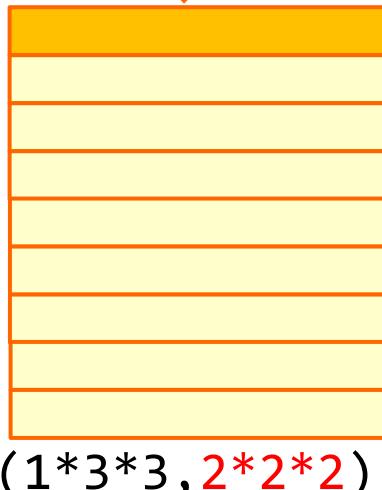
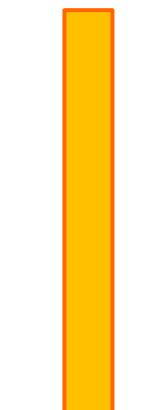
im2col

 $(1, 2, 4, 4)$ 

입력 데이터

 $(1, 2, 2, 2)$

↓
im2col

 $(2 * 2 * 2, 1)$ $(9, 8)(8, 1)$ $(2 * 2 * 2, 1)$ $(9, 1)$  $(1, 1, 3, 3)$

```

col_W = self.w.reshape(FN, -1).T
out = np.dot(col, col_W) + self.b
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)

```

 $(2, 1, 2, 2)$

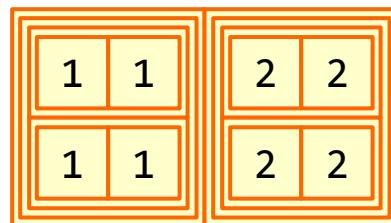
1	1
1	1
2	2
2	2

 $(2, 4)$

1	1	1	1
2	2	2	2

 $(4, 2)$

1	2
1	2
1	2
1	2



```
col_w = self.w.reshape(FN, -1).T  
out = np.dot(col, col_w) + self.b  
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

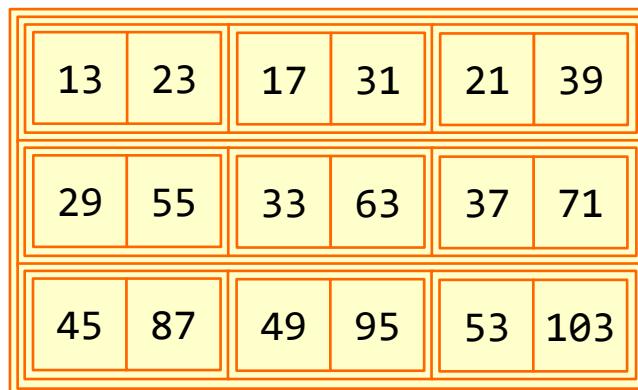
$$\begin{array}{c}
 \text{col} \\
 \begin{array}{|c|c|c|c|} \hline 0 & 1 & 4 & 5 \\ \hline 1 & 2 & 5 & 6 \\ \hline 2 & 3 & 6 & 7 \\ \hline 4 & 5 & 8 & 9 \\ \hline 5 & 6 & 9 & 10 \\ \hline 6 & 7 & 10 & 11 \\ \hline 8 & 9 & 12 & 13 \\ \hline 9 & 10 & 13 & 14 \\ \hline 10 & 11 & 14 & 15 \\ \hline \end{array}
 \end{array}
 \cdot
 \begin{array}{c}
 \text{col_w} \\
 \begin{array}{|c|c|} \hline 1 & 2 \\ \hline \end{array}
 \end{array}
 =
 \begin{array}{|c|c|} \hline 10 & 20 \\ \hline 14 & 28 \\ \hline 18 & 36 \\ \hline 26 & 52 \\ \hline 30 & 60 \\ \hline 34 & 68 \\ \hline 42 & 84 \\ \hline 46 & 92 \\ \hline 50 & 100 \\ \hline \end{array}
 +
 \begin{array}{|c|c|} \hline 3 & 3 \\ \hline \end{array}
 =
 \begin{array}{|c|c|} \hline 13 & 23 \\ \hline 17 & 31 \\ \hline 21 & 39 \\ \hline 29 & 55 \\ \hline 33 & 63 \\ \hline 37 & 71 \\ \hline 45 & 87 \\ \hline 49 & 95 \\ \hline 53 & 103 \\ \hline \end{array}$$

```
col_W = self.w.reshape(FN, -1).T
out = np.dot(col, col_W) + self.b
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

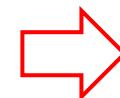
out

13	23
17	31
21	39
29	55
33	63
37	71
45	87
49	95
53	103

(9, 2)



(1, 3, 3, 2)



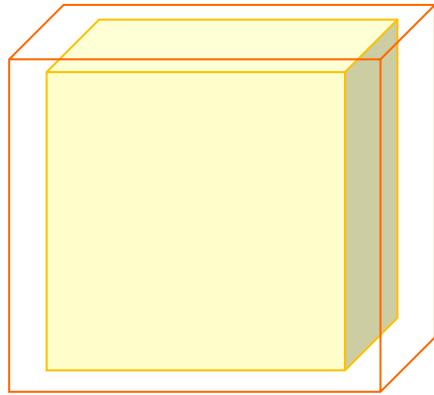
(1, 3, 3, 2)

(1, 2, 3, 3)

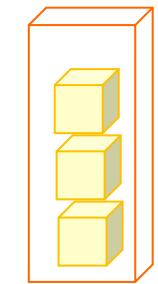
13	17	21
29	33	37
45	49	53
23	31	39
55	63	71
87	95	103

im2col

$(1, 1, 4, 4)$



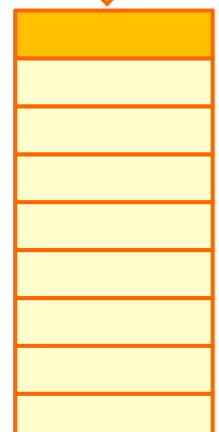
입력 데이터



$(2, 1, 2, 2)$



im2col

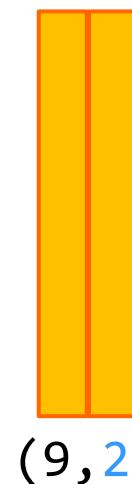


$(1*3*3, 1*2*2)$

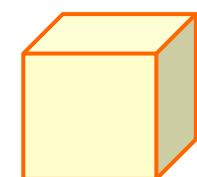
•



$(1*2*2, 2)$



$(9, 2)$

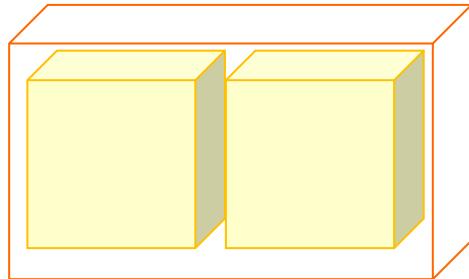


$(1, 2, 3, 3)$

$(1, 3, 3, 2)$

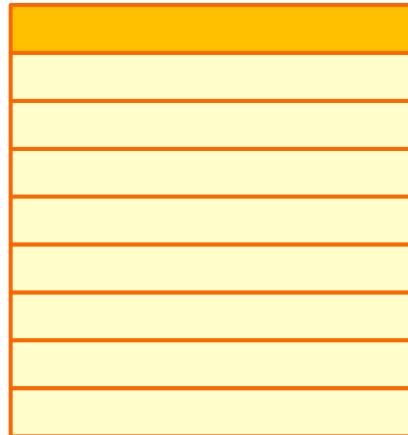
im2col

(1, 2, 4, 4)



입력 데이터

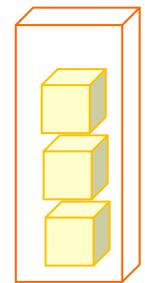
↓
im2col



(9, 8) (8, 3)

(N*OH*OW, C*FH*FW)

(1*3*3, 2*2*2)

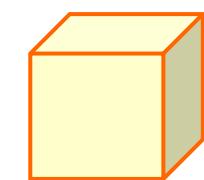


(3, 2, 2, 2)

(9, 3) => (1, 3, 3, 3) => (1, 3, 3, 3)



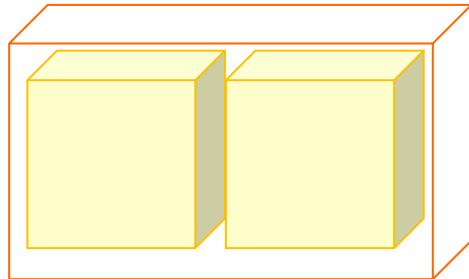
(9, 3)



(1, 3, 3, 3)

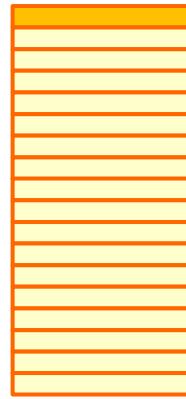
im2col

(2, 2, 4, 4)



입력 데이터

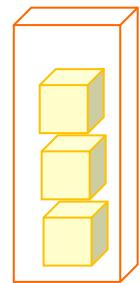
↓
im2col



(18, 8)(8, 3) (2*2*2, 3)

(N*OH*OW, C*FH*FW)

(2*3*3, 2*2*2)

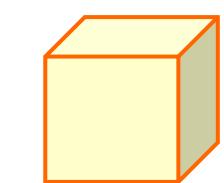


(3, 2, 2, 2)

(18, 3) => (2, 3, 3, 3) => (2, 3, 3, 3)



(18, 3)

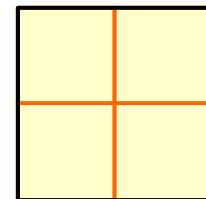


(2, 3, 3, 3)

$(1, 1, 4, 4)$
img

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

$(1, 1, 2, 2)$
filter



$(1, 1, 3, 3)$
out

width



height

$(1, 3, 182, 277)$
(N, C, H, W)

$(1, 3, 2, 2)$

$(1, 1, 181, 276)$

```
col = np.zeros((1, 1, 2, 2, 3, 3))
for y in range(filter_h):
    y_max = y + stride*out_h
    for x in range(filter_w):
        x_max = x + stride*out_w
        col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
```

```
conv = Convolution(w, b, pad=1)
```

(16, 4)

img

(1,1,4,4)

(16, 1)

col

 $OH = (H+2*P-FH)/S + 1$

1	1	1
1	1	1
1	1	1

(9, 1)

filter
(1,1,3,3)

17
27
33
25
36
57
66
48
60
93
102
72
49
75
81
57

out
(1,1,4,4)

17	27	33	25

0	0	0	0	0	0	0
0	1	2	3	4	0	0
0	5	6	7	8	0	0
0	9	10	11	12	0	0
0	13	14	15	16	0	0
0	0	0	0	0	0	0

col_w
(9, 1)

1
1
1
1
1
1
1
1
1

•

=

+

3

=

```
def forward(self, x):
    col = im2col(x, FH, FW, self.stride, self.pad)
    col_W = self.W.reshape(FN, -1).T

    out = np.dot(col, col_W) + self.b
    out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

```
def backward(self, dout):
    FN, C, FH, FW = self.W.shape
    dout = dout.transpose(0, 2, 3, 1).reshape(-1, FN)

    self.db = np.sum(dout, axis=0)
    self.dW = np.dot(self.col.T, dout)
    self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)

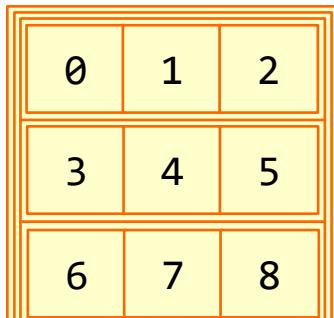
    dcol = np.dot(dout, self.col_W.T)
    dx = col2im(dcol, self.x.shape, FH, FW, self.stride, self.pad)

    return dx
```

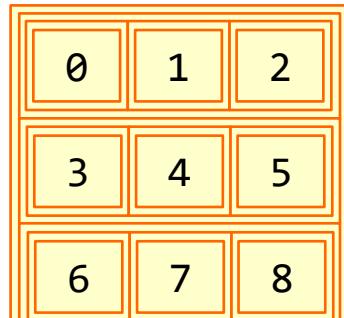
```
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

```
dout = dout.transpose(0,2,3,1).reshape(-1, FN)
```

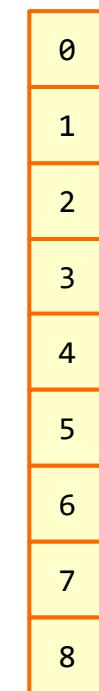
$dout(1,1,3,3)$



$(1, 1, 3, 3)$



$(1, 3, 3, 1)$



$dout$

$(1, 1, 4, 4) \quad (1, 1, 2, 2) \quad (1, 1, 3, 3)$

$(1, 3, 3, 1) \rightarrow (1, 1, 3, 3)$

$(1, 3, 3, 1) \leftarrow (1, 1, 3, 3)$

```
self.db = np.sum(dout, axis=0)
```

dout
(9,1)



3
self.b
(1,)

36
self.db
(1,)

$$\begin{array}{c}
 \text{col} \\
 (2, 3) \\
 \boxed{\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}}
 \end{array}
 \cdot
 \begin{array}{c}
 \text{w} \\
 (3, 4) \\
 \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline \end{array}}
 \end{array}
 =
 \begin{array}{c}
 \text{out} \\
 (2, 4) \\
 \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}}
 \end{array}$$

$$\begin{array}{c}
 \text{col.T} \\
 (3, 2) \\
 \boxed{\begin{array}{|c|c|}\hline & \\ \hline & \\ \hline & \\ \hline \end{array}}
 \end{array}
 \cdot
 \begin{array}{c}
 \text{dout} \\
 (2, 4) \\
 \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}}
 \end{array}
 =
 \begin{array}{c}
 \text{dw} \\
 (3, 4) \\
 \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline \end{array}}
 \end{array}$$

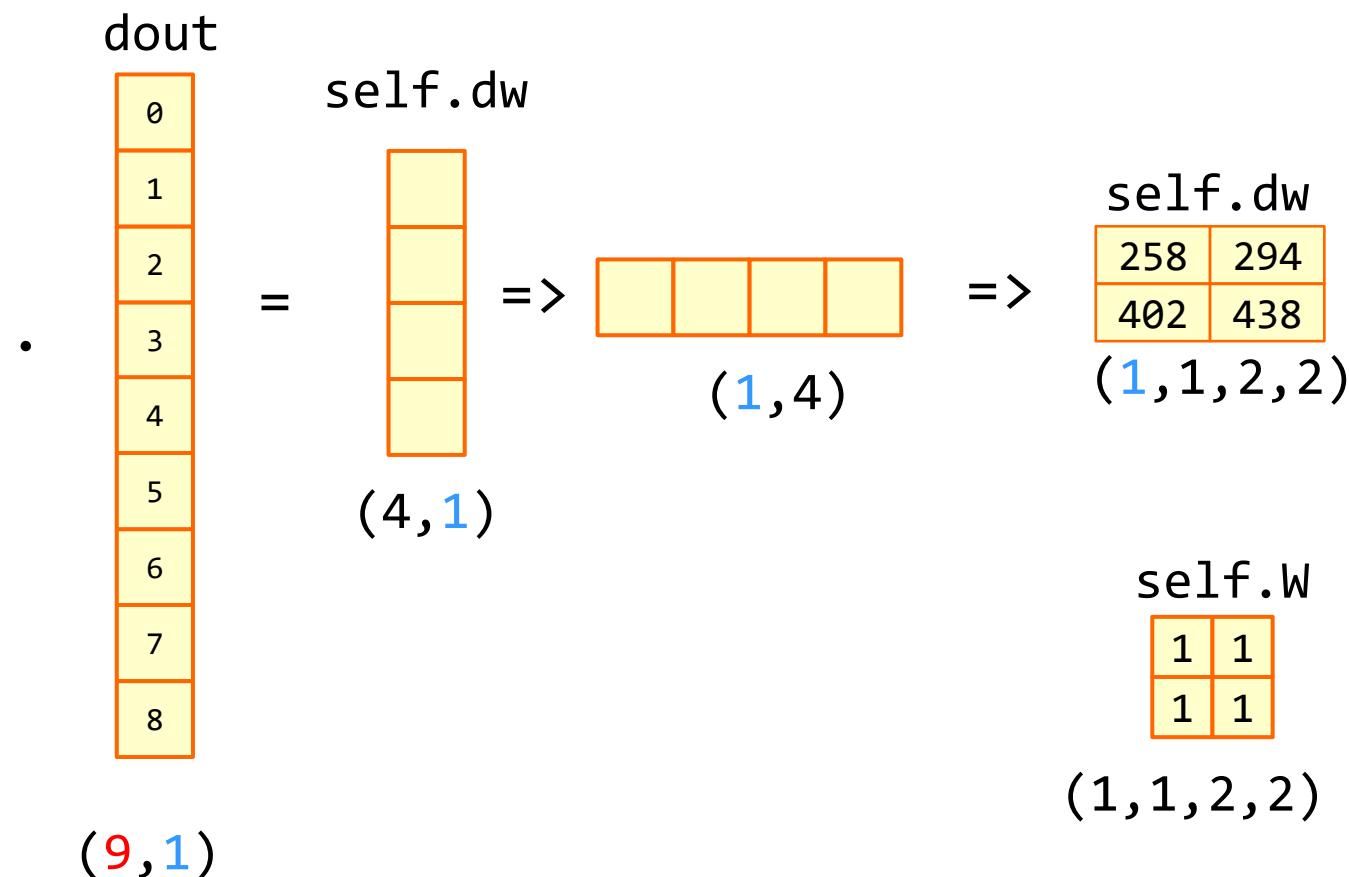
$$\begin{array}{c}
 \text{dout} \\
 (2, 4) \\
 \boxed{\begin{array}{|c|c|c|c|}\hline & & & \\ \hline & & & \\ \hline & & & \\ \hline \end{array}}
 \end{array}
 \cdot
 \begin{array}{c}
 \text{w.T} \\
 (4, 3) \\
 \boxed{\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}}
 \end{array}
 =
 \begin{array}{c}
 \text{dcol} \\
 (2, 3) \\
 \boxed{\begin{array}{|c|c|c|}\hline & & \\ \hline & & \\ \hline & & \\ \hline \end{array}}
 \end{array}$$

```
self.dW = np.dot(self.col.T, dout)
self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
```

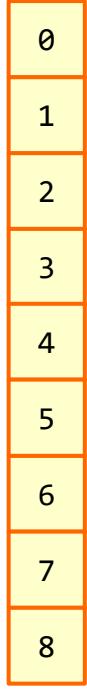
$(1*3*3, 1*2*2) \Rightarrow (9, 4)$

self.col.T								
0	1	2	4	5	6	8	9	10
1	2	3	5	6	7	9	10	11
4	5	6	8	9	10	12	13	14
5	6	7	9	10	11	13	14	15

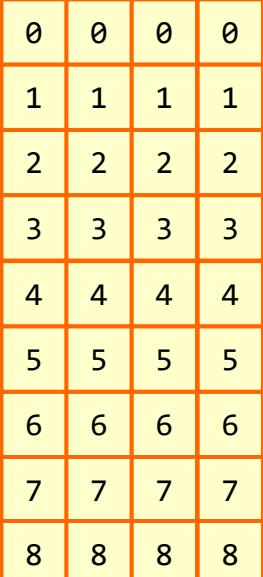
$(4, 9)$

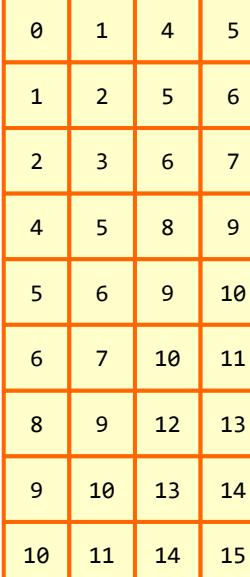


```
dcol = np.dot(dout, self.col_w.T)
```

dout

 $(9, 1)$

\cdot 
 $=$
 self.col_w.T
 $(1, 4)$

dcol
 $(9, 4)$


col
 $(9, 4)$


```
dx = col2im(dcol, self.x.shape=(1,1,4,4),
             FH=2, FW=2, self.stride=1, self.pad=0)

def col2im(col, input_shape, filter_h, filter_w, stride=1, pad=0):
    N, C, H, W = input_shape
    out_h = (H + 2*pad - filter_h)//stride + 1
    out_w = (W + 2*pad - filter_w)//stride + 1
    col = col.reshape(N, out_h, out_w, C, filter_h, filter_w).transpose(0, 3, 4, 5, 1, 2)

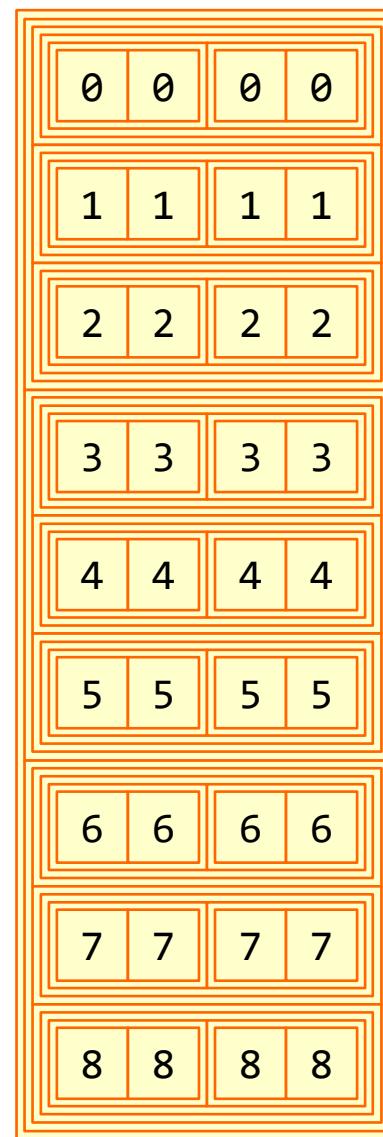
    img = np.zeros((N, C, H + 2*pad + stride - 1, W + 2*pad + stride - 1))
    for y in range(filter_h):
        y_max = y + stride*out_h
        for x in range(filter_w):
            x_max = x + stride*out_w
            img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]

    return img[:, :, pad:H + pad, pad:W + pad]
```

```
dcol = dcol.reshape(N, out_h, out_w, C, filter_h, filter_w)
```

dcol
(9,4)

0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3
4	4	4	4
5	5	5	5
6	6	6	6
7	7	7	7
8	8	8	8

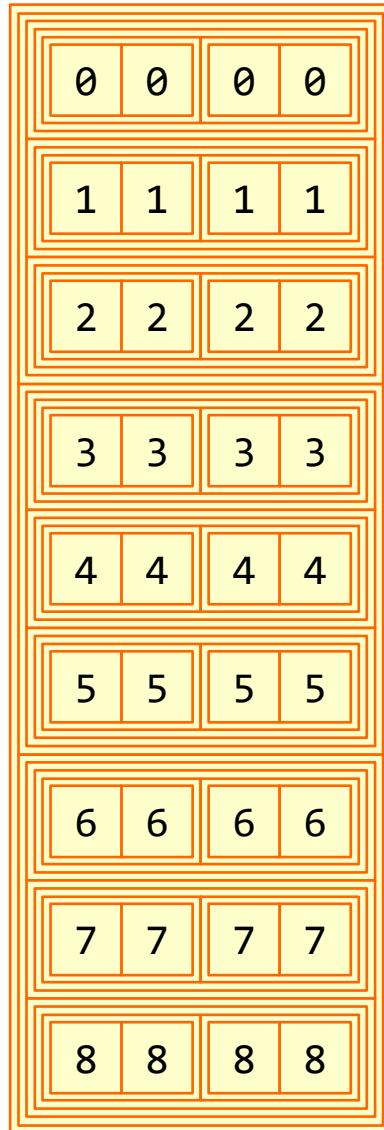


dcol
(1,3,3,1,2,2)

```
dcol = dcol.transpose(0, 3, 4, 5, 1, 2)
```

dcol

(1,3,3,1,2,2)

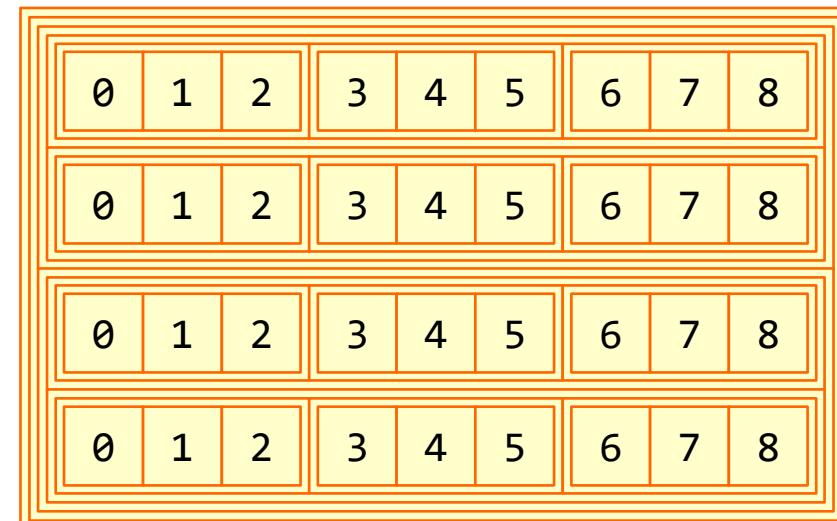


(1,3,3,1,2,2)

(1,1,2,2,3,3)

dcol

(1,1,2,2,3,3)



im2col

```
for y in range(filter_h):
    y_max = y + stride*out_h
    for x in range(filter_w):
        x_max = x + stride*out_w
        col[:, :, y, x, :, :] = img[:, :, y:y_max:stride, x:x_max:stride]
```

col2im

```
for y in range(filter_h):
    y_max = y + stride*out_h
    for x in range(filter_w):
        x_max = x + stride*out_w
        img[:, :, y:y_max:stride, x:x_max:stride] += col[:, :, y, x, :, :]
```

(1,1,4,4)

`img[:, :, 0:3:1, 0:3:1] += dcol[:, :, 0, 0, :, :]`

(1,1,3,3)

(1,1,2,2,3,3)

(1,1,3,3)

img

0	1	2	
3	4	5	
6	7	8	

dcol

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8



(1,1,4,4)

`img[:, :, 0:3:1, 1:4:1] += dcol[:, :, 0, 1, :, :]`

(1,1,3,3)

(1,1,2,2,3,3)

(1,1,3,3)

img

0	1	3	2
3	7	9	5
6	13	15	8

dcol

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8



(1,1,4,4)

img[:, :, 1:4:1, 0:3:1] += dcol[:, :, 1, 0, :, :]

(1,1,3,3)

(1,1,**2**,**2**,3,3)

(1,1,3,3)

img

0	1	3	2
3	8	11	5
9	17	20	8
6	7	8	

dcol



0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8

(1,1,4,4)

`img[:, :, 1:4:1, 1:4:1] += dcol[:, :, 1, 1, :, :]`

(1,1,3,3)

(1,1,**2**,**2**,3,3)

(1,1,3,3)

img

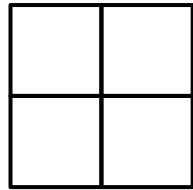
0	1	3	2
3	8	12	7
9	20	24	13
6	13	15	8

```
[[[[ 0.  1.  3.  2. ]
   [ 3.  8. 12.  7. ]
   [ 9. 20. 24. 13. ]
   [ 6. 13. 15.  8.]]]]
```

dcol

0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8
0	1	2	3	4	5	6	7	8





img

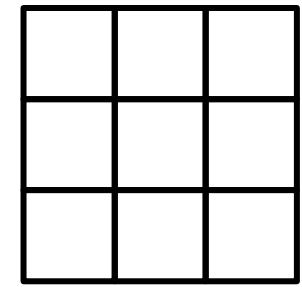
(1,1,4,4)

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

col

0	1	4	5
1	2	5	6
2	3	6	7
4	5	8	9
5	6	9	10
6	7	10	11
8	9	12	13
9	10	13	14
10	11	14	15

합성곱 연산은 리피트 연산이다.

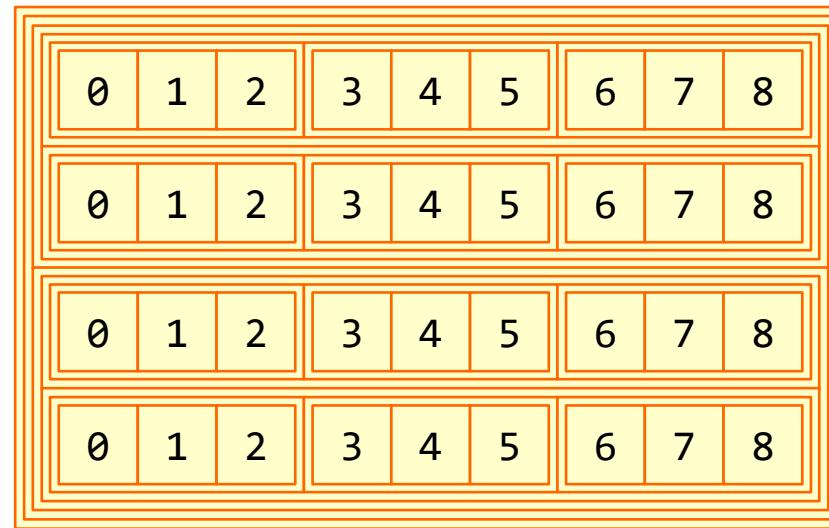


dx
(1, 1, 4, 4)

0	1	3	2
3	8	12	7
9	20	24	13
6	13	15	8



dcol

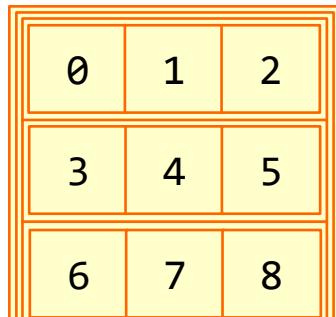


합성곱 연산의 미분은 합연산이다.

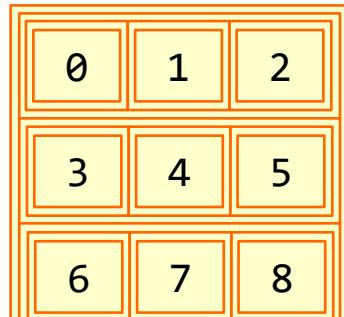
```
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

```
dout = dout.transpose(0,2,3,1).reshape(-1, FN)
```

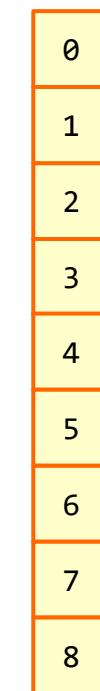
$dout(1,1,3,3)$



$(1, 1, 3, 3)$



$(1, 3, 3, 1)$



$(9, 1)$
dout

$(1, 1, 4, 4) \quad (1, 1, 2, 2) \quad (1, 1, 3, 3)$

$(1, 3, 3, 1) \rightarrow (1, 1, 3, 3)$

$(1, 3, 3, 1) \leftarrow (1, 1, 3, 3)$

```
self.dW = np.dot(self.col.T, dout)
self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
```

$(1*3*3, 2*2*2) \Rightarrow (9, 8)$

0	1	2	4	5	6	8	9	10
1	2	3	5	6	7	9	10	11
4	5	6	8	9	10	12	13	14
5	6	7	9	10	11	13	14	15
16	17	18	20	21	22	24	25	26
17	18	19	21	22	23	25	26	27
20	21	22	24	25	26	28	29	30
21	22	23	25	26	27	29	30	31

$(8, 9)$

dout

0
1
2
3
4
5
6
7
8

$(9, 1)$

self.dw

$(8, 1)$

=

=>

$(1, 8)$

self.dw

$(1, 2, 2, 2)$

self.W

1	1	1	1
1	1	1	1

$(1, 2, 2, 2)$

```
dcol = np.dot(dout, self.col_w.T)
```

dout

0
1
2
3
4
5
6
7
8

$$\cdot \begin{matrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{matrix} =$$

self.col_w.T
(1, 8)

(9, 1)

dcol
(9, 8)

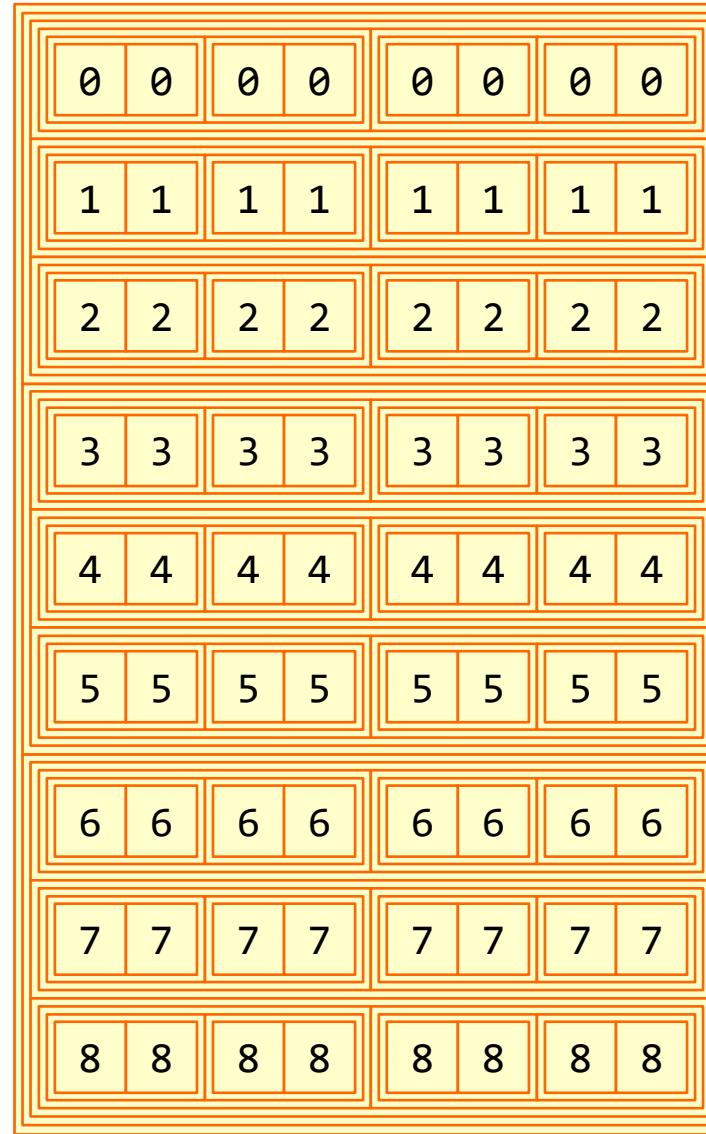
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8

col
(9, 8)

0	1	4	5	16	17	20	21
1	2	5	6	17	18	21	22
2	3	6	7	18	19	22	23
4	5	8	9	20	21	24	25
5	6	9	10	21	22	25	26
6	7	10	11	22	23	26	27
8	9	12	13	24	25	28	29
9	10	13	14	25	26	29	30
10	11	14	15	26	27	30	31

$(1, 3, 3, \textcolor{red}{2}, 2, 2)$ dcol
(9,8)

0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8



$(1, 3, 3, \cancel{2}, \cancel{2}, \cancel{2}) \Rightarrow$
 $(1, \cancel{2}, 2, 2, 3, 3)$

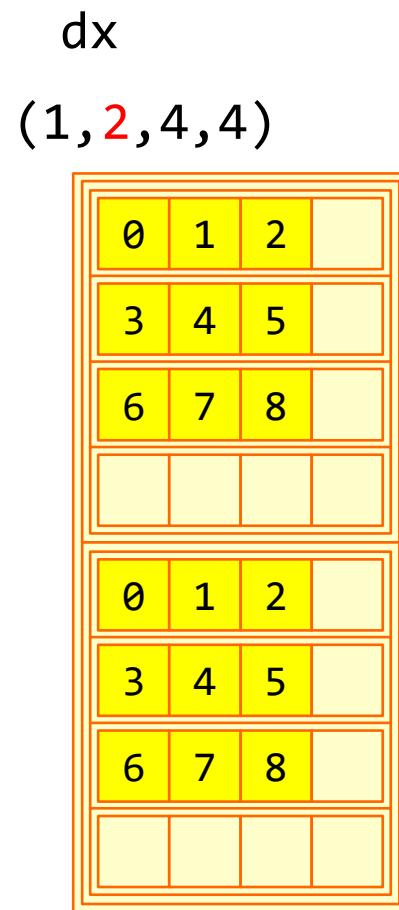
0 0	0 0	0 0	0 0
1 1	1 1	1 1	1 1
2 2	2 2	2 2	2 2
3 3	3 3	3 3	3 3
4 4	4 4	4 4	4 4
5 5	5 5	5 5	5 5
6 6	6 6	6 6	6 6
7 7	7 7	7 7	7 7
8 8	8 8	8 8	8 8



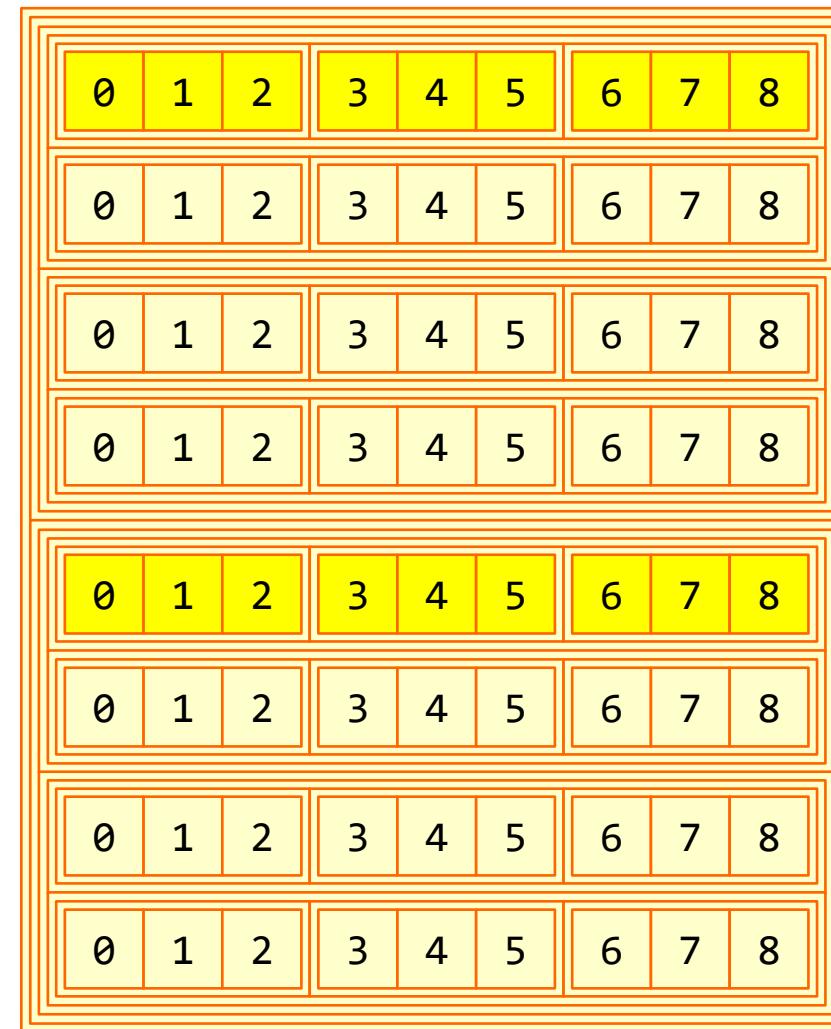
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8
0 1 2	3 4 5	6 7 8

$\text{col}(1, \color{red}{2}, 2, 2, 3, 3)$

col



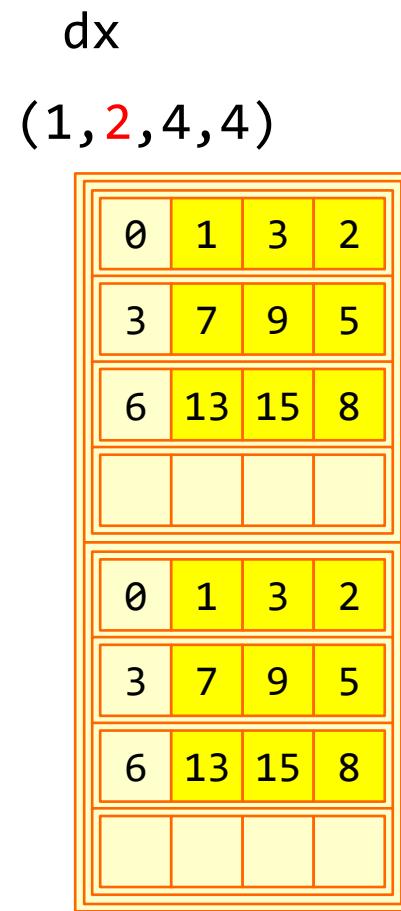
=



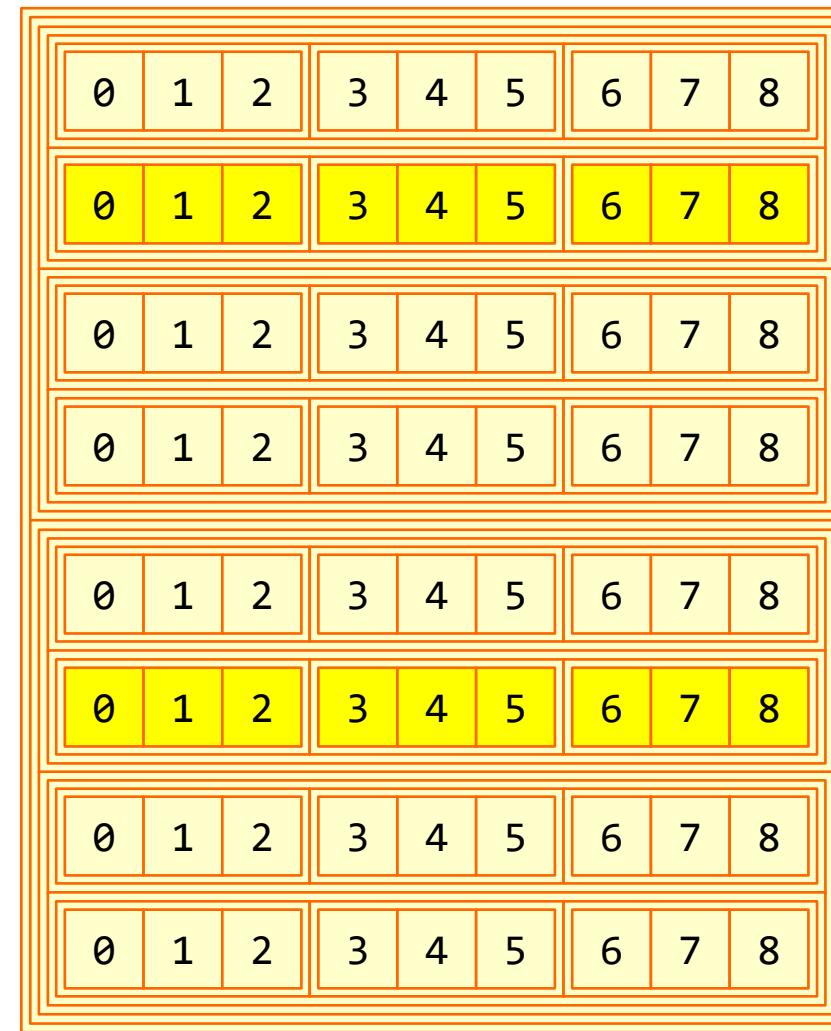
`img[:, :, 0:3:1, 0:3:1] += col[:, :, 0, 0, :, :]`

$\text{col}(1, \color{red}{2}, 2, 2, 3, 3)$

col



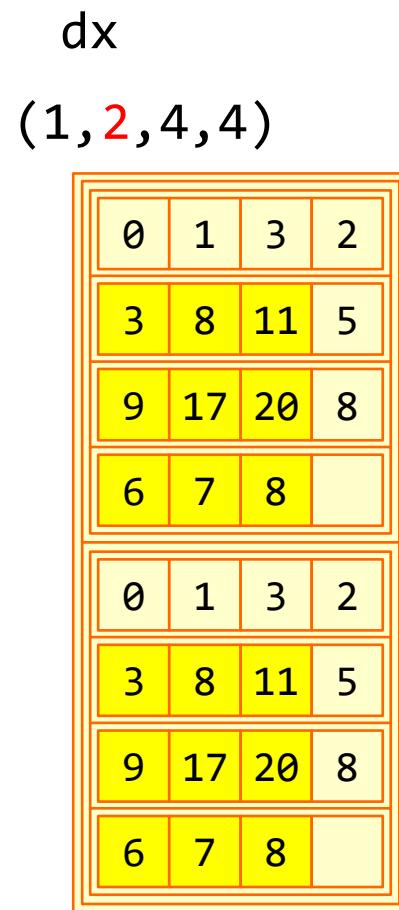
=



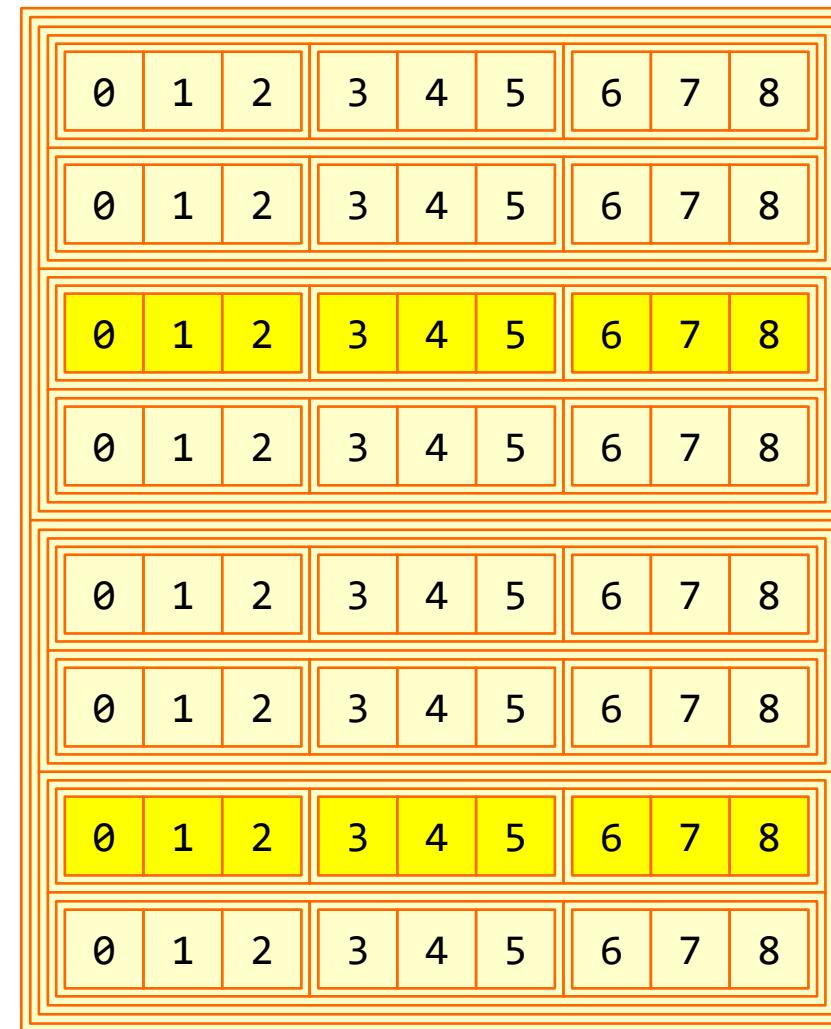
$\text{img}[:, :, 0:3:1, 1:4:1] \color{red}{=} \text{col}[:, :, 0, 1, :, :]$

$\text{col}(1, \color{red}{2}, 2, 2, 3, 3)$

col



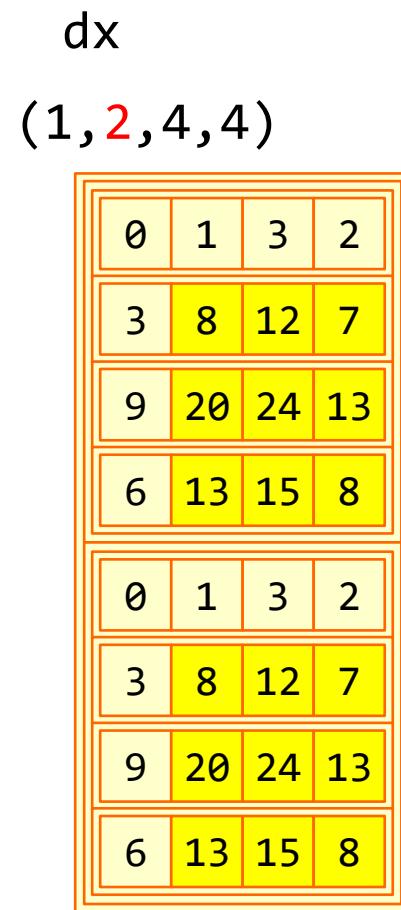
=



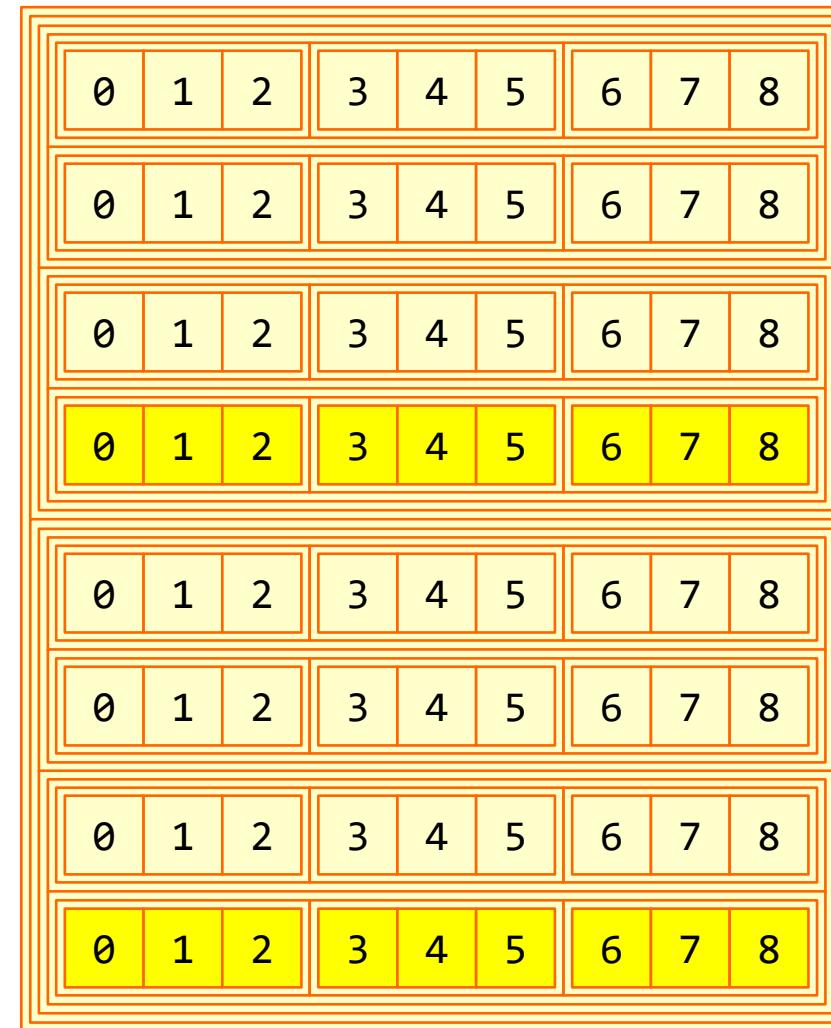
`img[:, :, 1:4:1, 0:3:1] += col[:, :, 1, 0, :, :]`

$\text{col}(1, \color{red}{2}, 2, 2, 3, 3)$

col



=



$\text{img}[:, :, 1:4:1, 1:4:1] \color{red}{=} \text{col}[:, :, 1, 1, :, :]$

col

0	1	4	5	16	17	20	21
1	2	5	6	17	18	21	22
2	3	6	7	18	19	22	23
4	5	8	9	20	21	24	25
5	6	9	10	21	22	25	26
6	7	10	11	22	23	26	27
8	9	12	13	24	25	28	29
9	10	13	14	25	26	29	30
10	11	14	15	26	27	30	31

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31

```
out = out.reshape(N, out_h, out_w, -1).transpose(0, 3, 1, 2)
```

```
dout = dout.transpose(0,2,3,1).reshape(-1, FN)
```

$dout(1, 1, 3, 3)$

0	1	2
3	4	5
6	7	8
9	10	11
12	13	14
15	16	17

$(1, 2, 3, 3)$

0	9	1	10	2	11
3	12	4	13	5	14
6	15	7	16	8	17

$(1, 3, 3, 2)$

$(1, 2, 3, 3)$

$\cancel{(1, 3, 3, 2)}$

0	9
1	10
2	11
3	12
4	13
5	14
6	15
7	16
8	17

$(9, 2)$

$dout$

```
self.dW = np.dot(self.col.T, dout)
self.dW = self.dW.transpose(1, 0).reshape(FN, C, FH, FW)
```

$(1*3*3, 2*2*2) \Rightarrow (9, 8)$

self.col.T								
0	1	2	4	5	6	8	9	10
1	2	3	5	6	7	9	10	11
4	5	6	8	9	10	12	13	14
5	6	7	9	10	11	13	14	15

$(4, 9)$

$$\begin{array}{c} \text{dout} \\ \cdot \\ \text{self.col.T} \end{array} = \begin{array}{c} \text{self.dw} \\ \Rightarrow \\ (4, 2) \end{array}$$

0	9
1	10
2	11
3	12
4	13
5	14
6	15
7	16
8	17

$(9, 2)$

$\begin{bmatrix} \begin{bmatrix} 258. & 294. \\ 402. & 438. \end{bmatrix} \end{bmatrix}$

$\begin{bmatrix} \begin{bmatrix} 663. & 780. \\ 1131. & 1248. \end{bmatrix} \end{bmatrix}$

$\begin{array}{c} \text{self.dw} \\ \Rightarrow \\ \begin{bmatrix} \begin{bmatrix} & & \\ & & \end{bmatrix} \end{bmatrix} \\ (2, 1, 2, 2) \end{array}$

$\begin{array}{c} \text{self.w} \\ \begin{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{bmatrix} \\ (2, 1, 2, 2) \end{array}$

```
def forward(self, x):
    N, C, H, W = x.shape
    out_h = int(1 + (H - self.pool_h) / self.stride)
    out_w = int(1 + (W - self.pool_w) / self.stride)

    col = im2col(x, self.pool_h, self.pool_w, self.stride, self.pad)
    col = col.reshape(-1, self.pool_h*self.pool_w)

    arg_max = np.argmax(col, axis=1)
    out = np.max(col, axis=1)
    out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

    self.x = x
    self.arg_max = arg_max

    return out
```

```
def forward(self, x):
    N, C, H, W = x.shape
    out_h = int(1 + (H - self.pool_h) / self.stride)
    out_w = int(1 + (W - self.pool_w) / self.stride)
```

$$OH = \frac{H - PH}{S} + 1$$

$$OW = \frac{W - PW}{S} + 1$$

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10



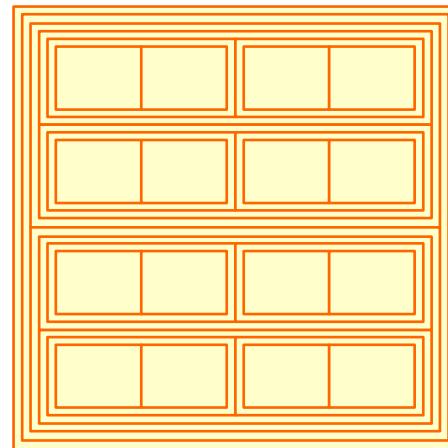
11	15
8	17

```
[[[[[ 7 11 13 15]
      [ 3 4 2 3]
      [ 1 2 17 9]
      [ 1 8 3 10]]]]]
```

```
[[[[[11. 15.]
      [ 8. 17.]]]]]
```

```
col = np.zeros((N, C, filter_h, filter_w, out_h, out_w))
```

col (1,1,2,2,2,2)

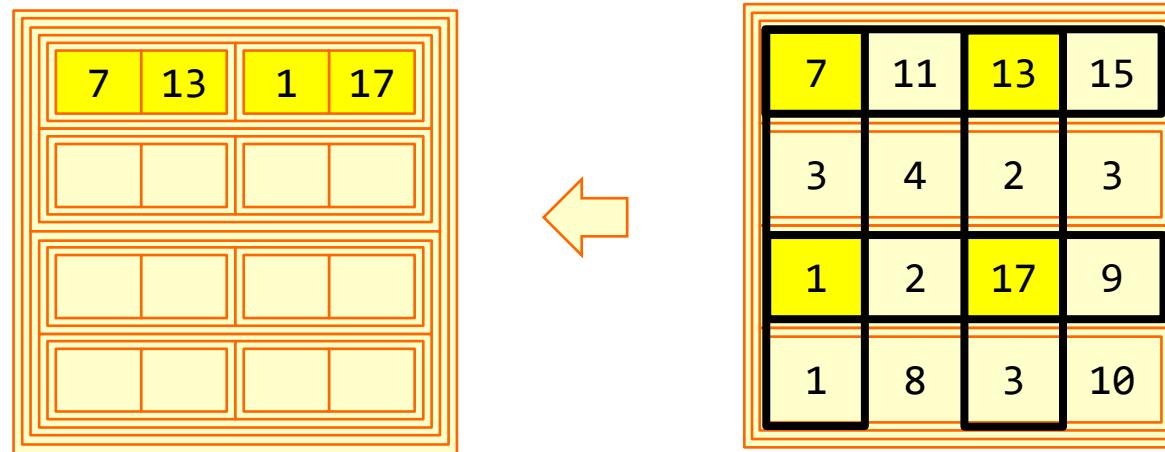


img (1,1,4,4)

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10

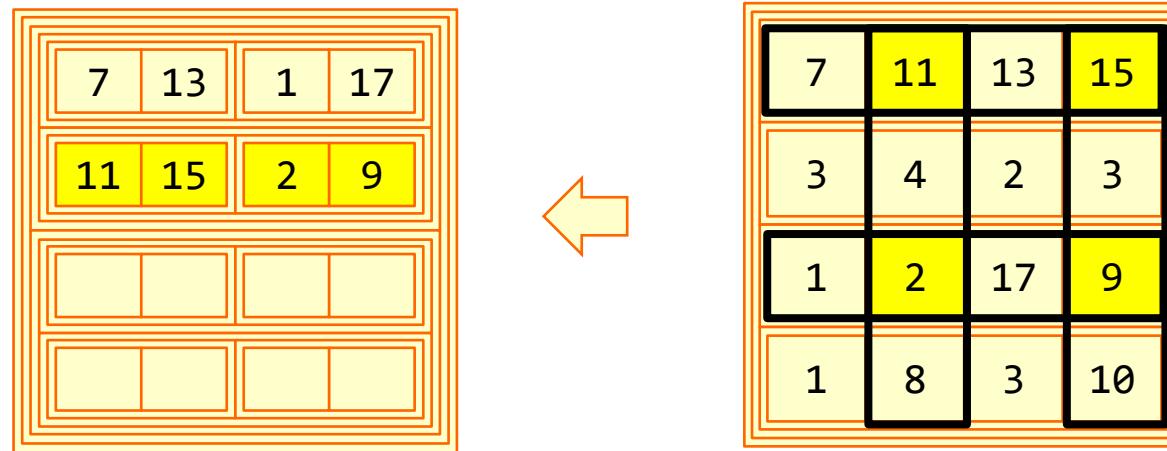
```
col[:, :, 0, 0, :, :] = img[:, :, 0:4:2, 0:4:2]
```

col (1,1,2,2,2,2) img (1,1,4,4)



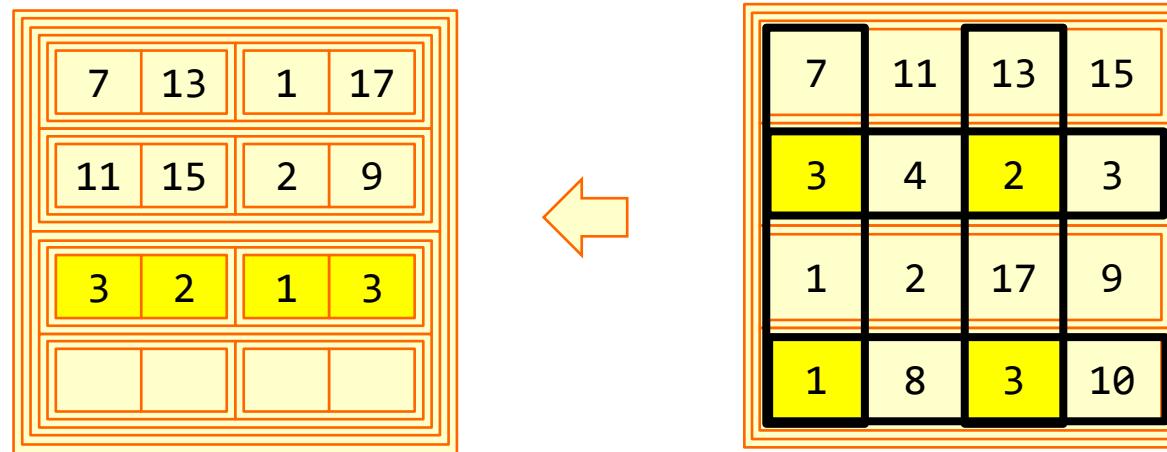
```
col[:, :, 0, 1, :, :] = img[:, :, 0:4:2, 1:5:2]
```

col (1,1,2,2,2,2) img (1,1,4,4)



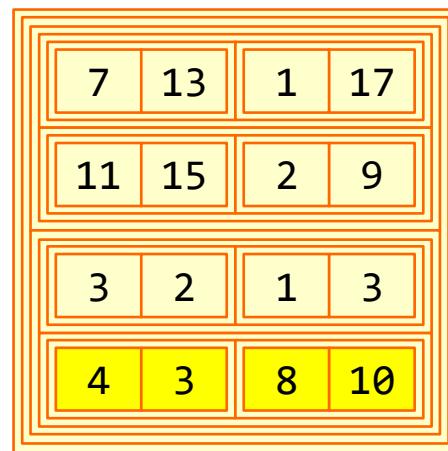
```
col[:, :, 1, 0, :, :] = img[:, :, 1:5:2, 0:4:2]
```

col (1,1,2,2,2,2) img (1,1,4,4)

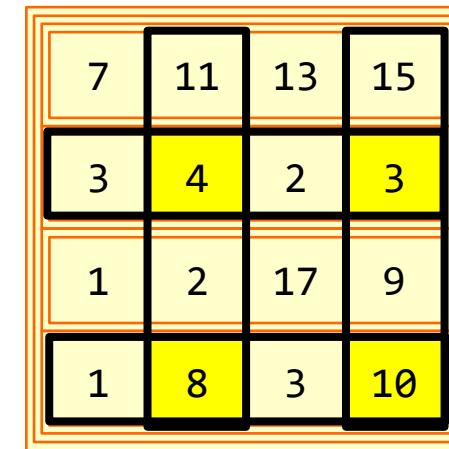


```
col[:, :, 1, 1, :, :] = img[:, :, 1:5:2, 1:5:2]
```

col (1,1,2,2,2,2)



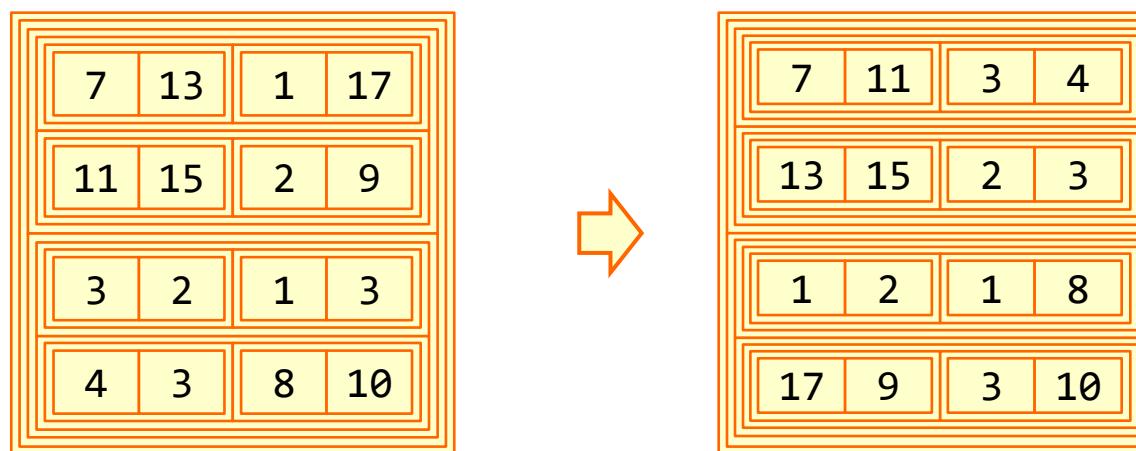
img (1,1,4,4)



```
col = col.transpose(0, 4, 5, 1, 2, 3)
```

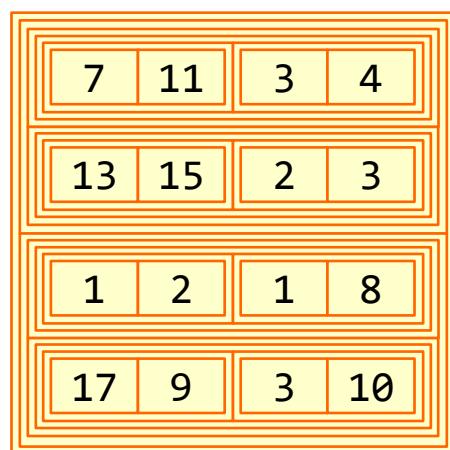
(1, 1, 2, 2, 2, 2)
(1, 2, 2, 1, 2, 2)

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10



```
col = col.reshape(N*out_h*out_w, -1)  
(N*out_h*out_w, C*pool_h*pool_w)  
(1*2*2, 1*2*2)
```

col (1,2,2,1,2,2)



col (4,4)

7	11	3	4
13	15	2	3
1	2	1	8
17	9	3	10

```
[[ 7. 11. 3. 4.]  
[13. 15. 2. 3.]  
[ 1. 2. 1. 8.]  
[17. 9. 3. 10.]]
```

```
col = col.reshape(-1, self.pool_h*self.pool_w)
```

$(N*out_h*out_w, C*pool_h*pool_w)$ $(1, 2, 2, 2, 2, 2)$

$(N*C*out_h*out_w, pool_h*pool_w)$ $(1, 2, 2, 2, 2, 2)$
 $(8, 4)$

$(1, 2, 2, 1, 2, 2)$

col $(4, 4)$

7	11	3	4
13	15	2	3
1	2	1	8
17	9	3	10



col $(4, 4)$

7	11	3	4
13	15	2	3
1	2	1	8
17	9	3	10

```

arg_max = np.argmax(col, axis=1)
out = np.max(col, axis=1)
out = out.reshape(N, out_h, out_w, C).transpose(0, 3, 1, 2)

```

	0	1	2	3
0	7	11	3	4
1	13	15	2	3
2	1	2	1	8
3	17	9	3	10

col

self.arg_max (4,)

1	1	3	0
---	---	---	---

out (4,)

11	15	8	17
----	----	---	----

out

11	15
8	17

(1, 1, 2, 2)

self.x

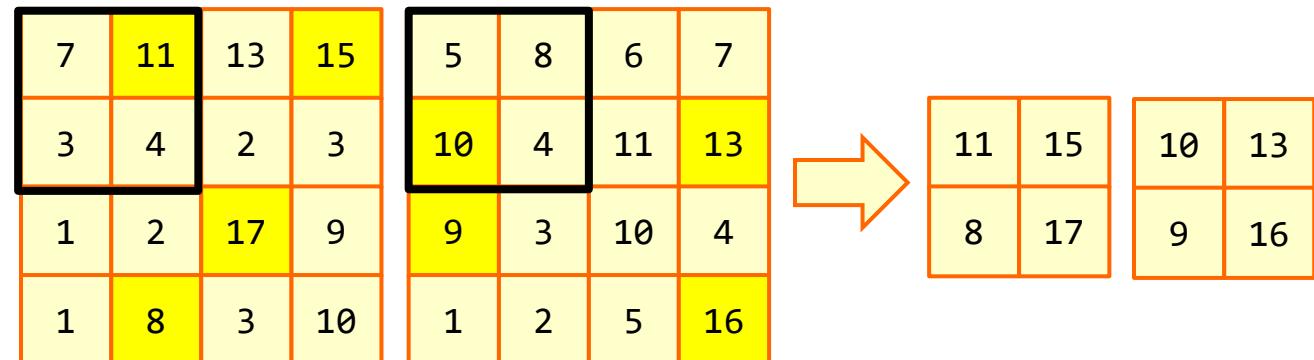
7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10

(1, 1, 4, 4)

```
def forward(self, x):
    N, C, H, W = x.shape
    out_h = int(1 + (H - self.pool_h) / self.stride)
    out_w = int(1 + (W - self.pool_w) / self.stride)
```

$$OH = \frac{H - PH}{S} + 1$$

$$OW = \frac{W - PW}{S} + 1$$

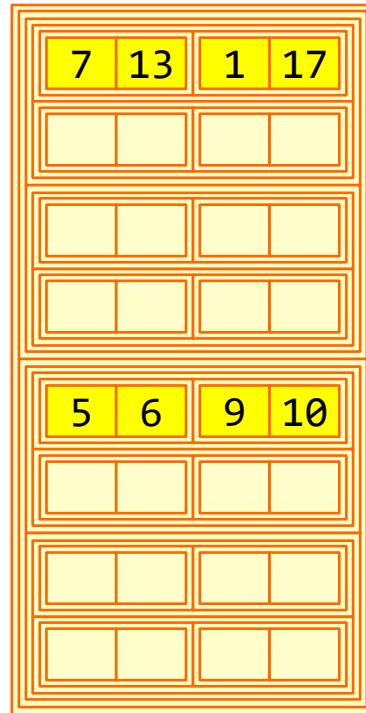


```
col[:, :, 0, 0, :, :] = img[:, :, 0:4:2, 0:4:2]
```

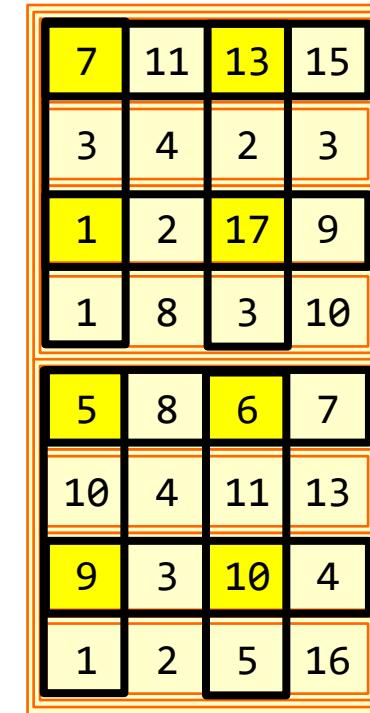
(1, 2, 2, 2)

(1, 2, 4, 4) => (1, 2, 2, 2)

col (1, 2, 2, 2, 2, 2)



img (1, 2, 4, 4)



```
col[:, :, 0, 1, :, :] = img[:, :, 0:4:2, 1:5:2]
```

(1, 2, 2, 2)

(1, 2, 4, 4) => (1, 2, 2, 2)

col (1, 2, 2, 2, 2, 2)

7	13	1	17
11	15	2	9
5	6	9	10
8	7	3	4

img (1, 2, 4, 4)

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10
5	8	6	7
10	4	11	13
9	3	10	4
1	2	5	16



```
col[:, :, 1, 0, :, :] = img[:, :, 1:5:2, 0:4:2]
```

(1, 2, 2, 2)

(1, 2, 4, 4) => (1, 2, 2, 2)

col (1, 2, 2, 2, 2, 2)

7	13	1	17
11	15	2	9
3	2	1	3
5	6	9	10
8	7	3	4
10	11	1	5

img (1, 2, 4, 4)

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10
5	8	6	7
10	4	11	13
9	3	10	4
1	2	5	16



```
col[:, :, 1, 1, :, :] = img[:, :, 1:5:2, 1:5:2]
```

(1, 2, 2, 2)

(1, 2, 4, 4) => (1, 2, 2, 2)

col (1, 2, 2, 2, 2, 2)

7	13	1	17
11	15	2	9
3	2	1	3
4	3	8	10
5	6	9	10
8	7	3	4
10	11	1	5
4	13	2	16

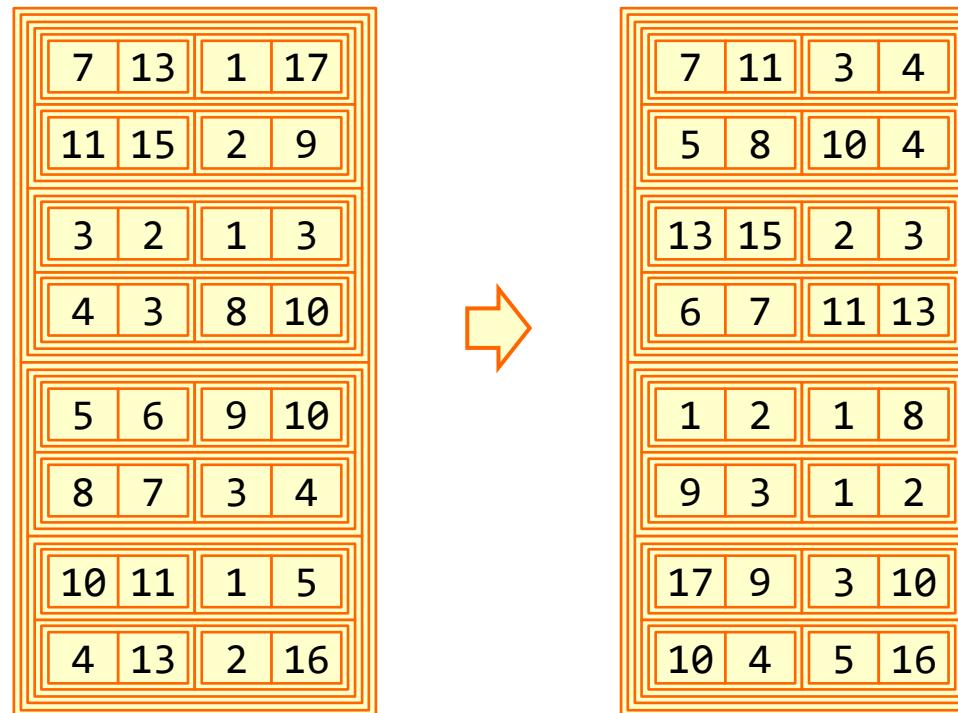
img (1, 2, 4, 4)

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10
5	8	6	7
10	4	11	13
9	3	10	4
1	2	5	16



```
col = col.transpose(0, 4, 5, 1, 2, 3)
```

(1, 2, 2, 2, 2, 2)
(1, 2, 2, 2, 2, 2)



```
col = col.reshape(N*out_h*out_w, -1)
(N*out_h*out_w, C*pool_h*pool_w)
(1*2*2, 2*2*2)
```

(1, 2, 2, 2, 2, 2)

7	11	3	4
5	8	10	4
13	15	2	3
6	7	11	13
1	2	1	8
9	3	1	2
17	9	3	10
10	4	5	16



col (8,4)

7	11	3	4	5	8	10	4
13	15	2	3	6	7	11	13
1	2	1	8	9	3	1	2
17	9	3	10	10	4	5	16

```
[[ 7. 11. 3. 4. 5. 8. 10. 4.]
 [13. 15. 2. 3. 6. 7. 11. 13.]
 [ 1. 2. 1. 8. 9. 3. 1. 2.]
 [17. 9. 3. 10. 10. 4. 5. 16.]]
```

```
col = col.reshape(-1, self.pool_h*self.pool_w)
```

$(N*out_h*out_w, C*pool_h*pool_w)$

$(N*C*out_h*out_w, pool_h*pool_w)$

col (4,8)

7	11	3	4	5	8	10	4
13	15	2	3	6	7	11	13
1	2	1	8	9	3	1	2
17	9	3	10	10	4	5	16



col (8,4)

7	11	3	4
5	8	10	4
13	15	2	3
6	7	11	13
1	2	1	8
9	3	1	2
17	9	3	10
10	4	5	16

```
arg_max = np.argmax(col, axis=1)  
out = np.max(col, axis=1)
```

	0	1	2	3									
0	7	11	3	4	self.argmax (8,)								
1	5	8	10	4	1	2	1	3	3	0	0	0	3
2	13	15	2	3	out (8,)								
3	6	7	11	13	11	10	15	13	8	9	17	16	
4	1	2	1	8									
5	9	3	1	2									
6	17	9	3	10									
7	10	4	5	16									

```
out = out.reshape(N, out_h, out_w, C)
out = out.transpose(0, 3, 1, 2)
```

$(1, 2, 2, 2)$
 ~~$(1, 2, 2, 2)$~~

out

 $(1, 2, 2, 2)$

0	1	2	3	
0	7	11	3	4
1	5	8	10	4
2	13	15	2	3
3	6	7	11	13
4	1	2	1	8
5	9	3	1	2
6	17	9	3	10
7	10	4	5	16

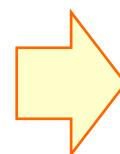
self.arg_max (8,)

1	2	1	3	3	0	0	3
---	---	---	---	---	---	---	---

out (8,)

11	10	15	13	8	9	17	16
----	----	----	----	---	---	----	----

$(1, 2, 2, 2)$
 ~~$(1, 2, 2, 2)$~~



11	10
15	13
8	9
17	16



11	15
8	17
10	13
9	16

7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10

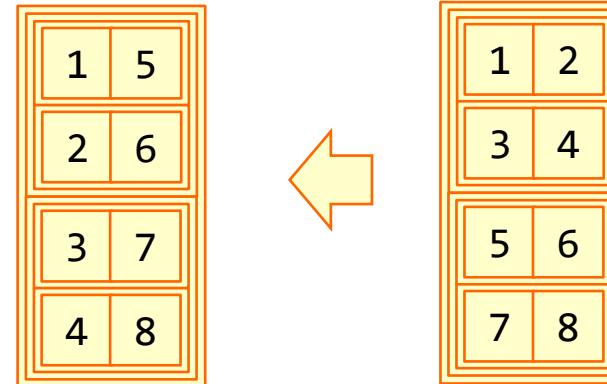
5	8	6	7
10	4	11	13
9	3	10	4
1	2	5	16

```
dout = dout.transpose(0, 2, 3, 1)
```

dout (1, 2, 2, 2)

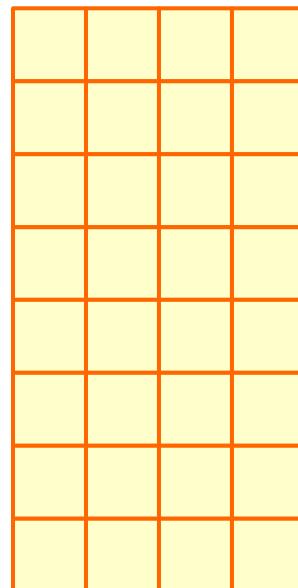
dout (1, 2, 2, 2)

(1, ~~2~~, 2, 2)
(1, 2, ~~2~~, ~~2~~)



```
pool_size = self.pool_h * self.pool_w  
dmax = np.zeros((dout.size, pool_size))
```

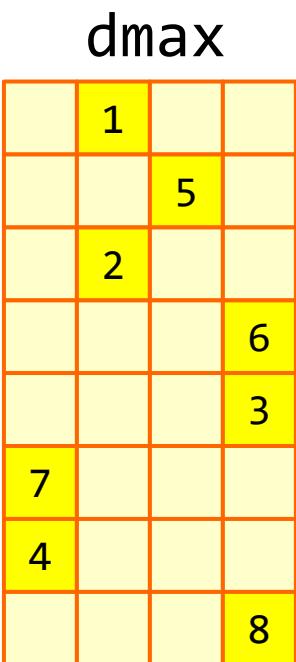
dmax (8,4)



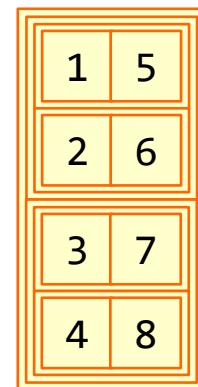
```
dmax[np.arange(self.arg_max.size), self.arg_max.flatten()] = dout.flatten()
```

```
dmax[[0,1,2,3,4,5,6,7], [1,2,1,3,3,0,0,3]] = [0,4,1,5,2,6,3,7]
```

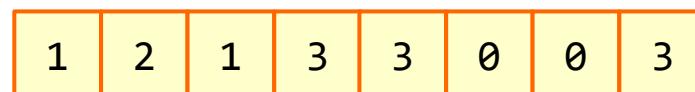
```
[[0.  1.  0.  0.]  
 [0.  0.  5.  0.]  
 [0.  2.  0.  0.]  
 [0.  0.  0.  6.]  
 [0.  0.  0.  3.]  
 [7.  0.  0.  0.]  
 [4.  0.  0.  0.]  
 [0.  0.  0.  8.]]
```



dout



self.arg_max (8,)



```
dmax = dmax.reshape(dout.shape + (pool_size,))  
        (1,2,2,2) + (4,)  
        (1,2,2,2,4)
```

dmax (8,4)

	1		
		5	
	2		
			6
			3
7			
4			
			8



dmax

0	1	0	0
0	0	5	0
0	2	0	0
0	0	0	6
0	0	0	3
7	0	0	0
4	0	0	0
0	0	0	8

```
dcol = dmax.reshape(dmax.shape[0] * dmax.shape[1] * dmax.shape[2], -1)
```

dmax (1,2,2,2,4)

0	1	0	0
0	0	5	0
0	2	0	0
0	0	0	6
0	0	0	3
7	0	0	0
4	0	0	0
0	0	0	8



dcol (4,8)

0	1	0	0	0	0	5	0
0	2	0	0	0	0	0	6
0	0	0	3	7	0	0	0
4	0	0	0	0	0	0	8

```
[[0. 1. 0. 0. 0. 0. 5. 0.]  
 [0. 2. 0. 0. 0. 0. 0. 6.]  
 [0. 0. 0. 3. 7. 0. 0. 0.]  
 [4. 0. 0. 0. 0. 0. 0. 8.]]
```

```
dx = col2im(dcol, self.x.shape, self.pool_h, self.pool_w, self.stride, self.pad)
(1,2,4,4)
```

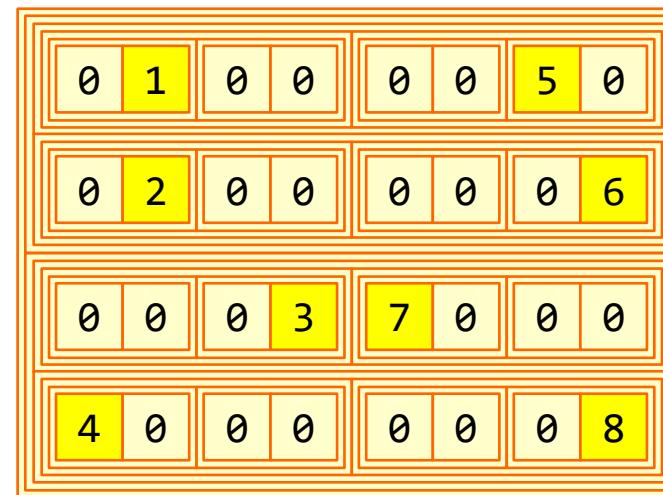
```
dcol = dcol.reshape(1, 2, 2, 2, 2, 2)
```

dcol (4,8)

0	1	0	0	0	0	5	0
0	2	0	0	0	0	0	6
0	0	0	3	7	0	0	0
4	0	0	0	0	0	0	8



dcol (1,2,2,2,2,2)



```
dcol = dcol.transpose(0, 3, 4, 5, 1, 2)
```

dcol $(1, \cancel{2}, \cancel{2}, 2, 2, 2)$

0	1	0	0	0	0	5	0
0	2	0	0	0	0	0	6
0	0	0	3	7	0	0	0
4	0	0	0	0	0	0	8



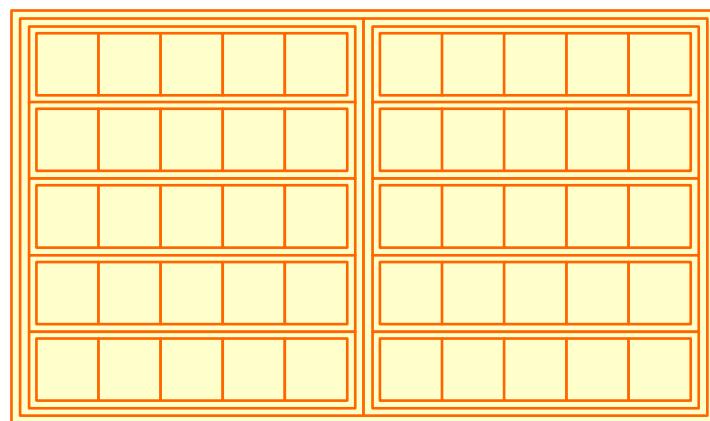
dcol $(1, 2, 2, 2, \cancel{2}, \cancel{2})$

0	0	0	4	1	2	0	0
0	0	0	0	0	0	3	0
0	0	7	0	0	0	0	0
5	0	0	0	0	6	0	8

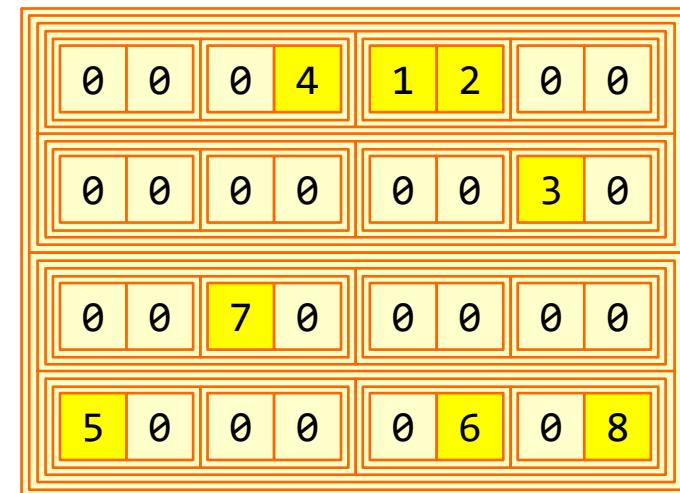
$(1, \cancel{2}, \cancel{2}, 2, 2, 2)$
 $(1, 2, 2, 2, \cancel{2}, \cancel{2})$

```
img = np.zeros((N, C, H + 2*pad + stride - 1, W + 2*pad + stride - 1))  
        (1,2,5,5)
```

img (1,2,5,5)

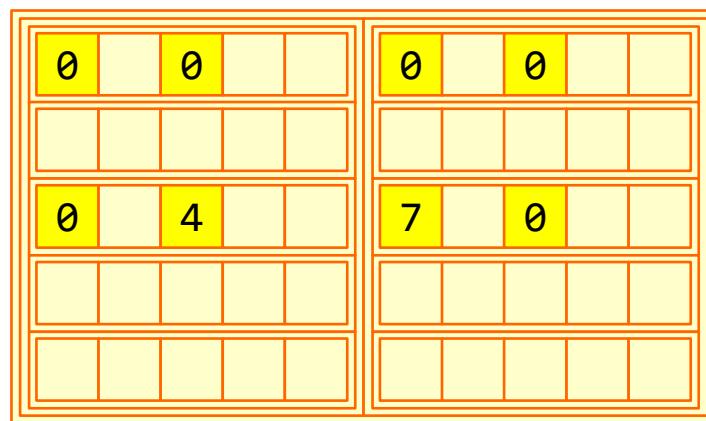


dcol (1,2,2,2,2,2)

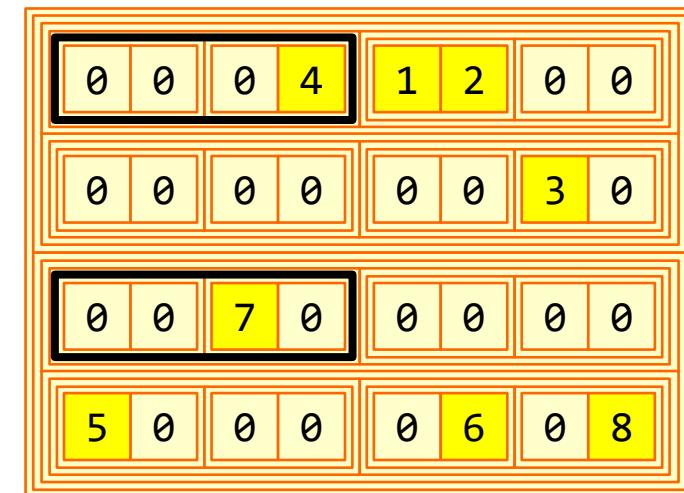


```
img[:, :, 0:4:2, 0:4:2] += col[:, :, 0, 0, :, :]
```

img (1, 2, 5, 5)



dcol (1, 2, 2, 2, 2, 2)



```
img[:, :, 0:4:2, 1:5:2] += col[:, :, 0, 1, :, :]
```

img (1,2,5,5)

0	1	0	1	
0	0	4	0	
7	0	0	0	



dcol (1,2,2,2,2,2)

0	0	0	4	1	2	0	0
0	0	0	0	0	0	3	0
0	0	7	0	0	0	0	0
5	0	0	0	0	6	0	8

```
img[:, :, 1:5:2, 0:4:2] += col[:, :, 1, 0, :, :]
```

img (1,2,5,5)

0	1	0	1	
0	0	0	0	
5		0		
0	0	4	0	
0	0			



dcol (1,2,2,2,2,2)

0	0	0	4	1	2	0	0
0	0	0	0	0	0	3	0
0	0	7	0	0	0	0	0
5	0	0	0	0	0	6	0

```
img[:, :, 1:5:2, 1:5:2] += col[:, :, 1, 1, :, :]
```

img (1,2,5,5)

0	1	0	2	
0	0	0	0	
0	0	4	0	
0	3	0	0	

0	0	0	0
5	0	0	6
7	0	0	0
0	0	0	8



dcol (1,2,2,2,2,2)

0	0	0	4	1	2	0	0
0	0	0	0	0	0	3	0
0	0	7	0	0	0	0	0
5	0	0	0	0	6	0	8

```
return img[:, :, pad:H + pad, pad:W + pad]
```

```
[[[[[0. 1. 0. 2.]  
[0. 0. 0. 0.]  
[0. 0. 4. 0.]  
[0. 3. 0. 0.]]]  
[[0. 0. 0. 0.]  
[5. 0. 0. 6.]  
[7. 0. 0. 0.]  
[0. 0. 0. 8.]]]]
```

dx (1,2,4,4)

0	1	0	2
0	0	0	0
5	0	0	6
7	0	0	0
0	0	0	8

dout (1,2,2,2)

1	2
3	4
5	6
7	8



7	11	13	15
3	4	2	3
1	2	17	9
1	8	3	10

5	8	6	7
10	4	11	13
9	3	10	4
1	2	5	16