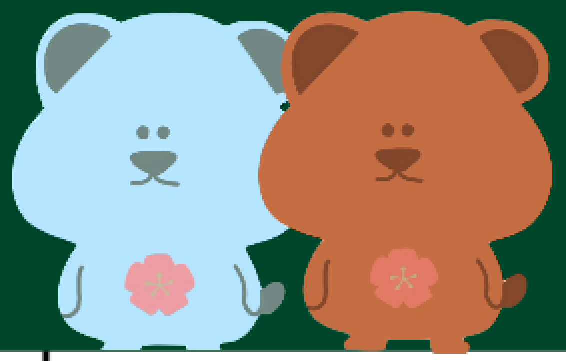
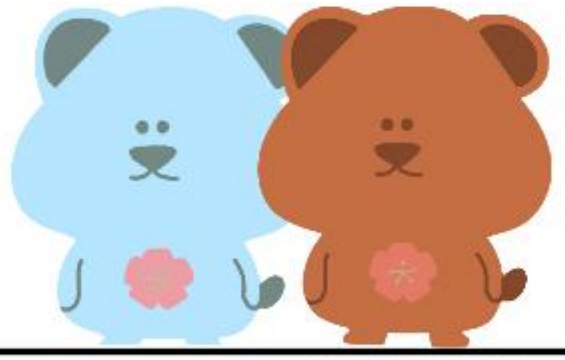


Backpropagation and Neural Networks

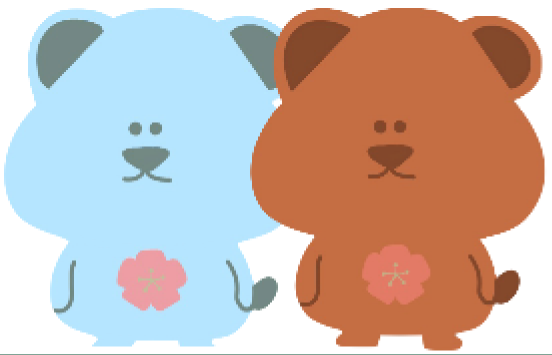
Week4_발표자: 구미진, 하수민

EWHA
EURON

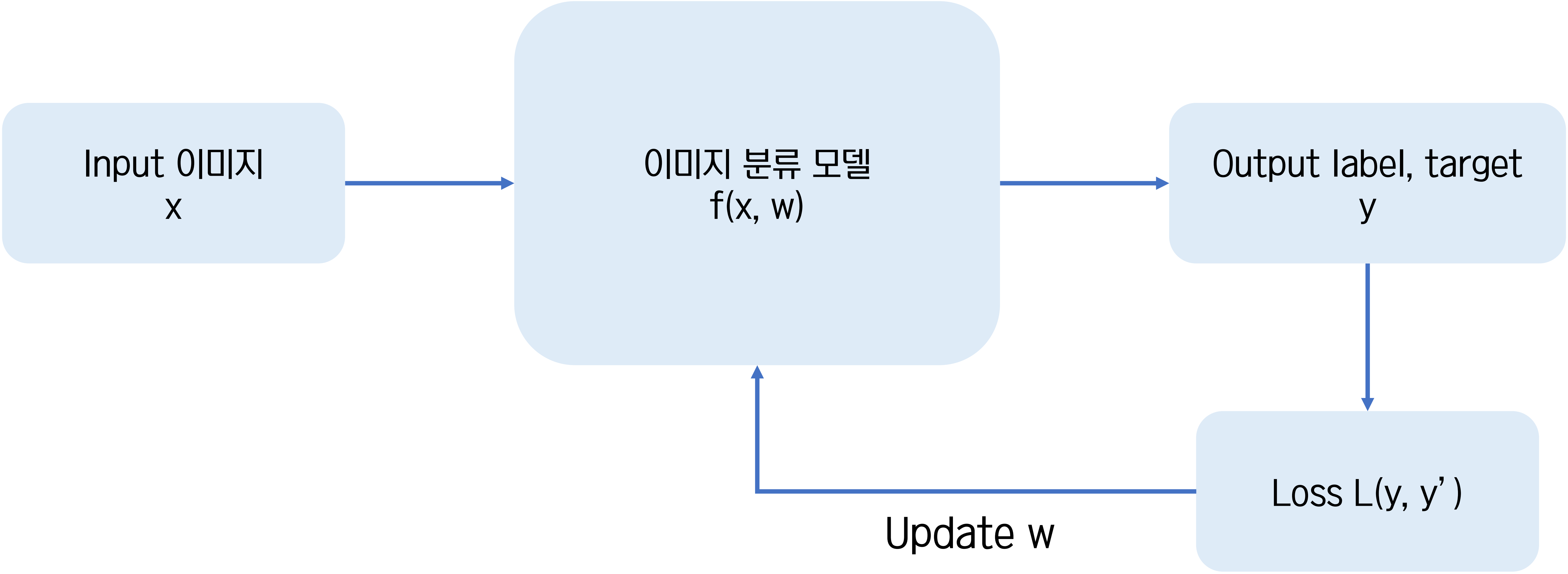
목차



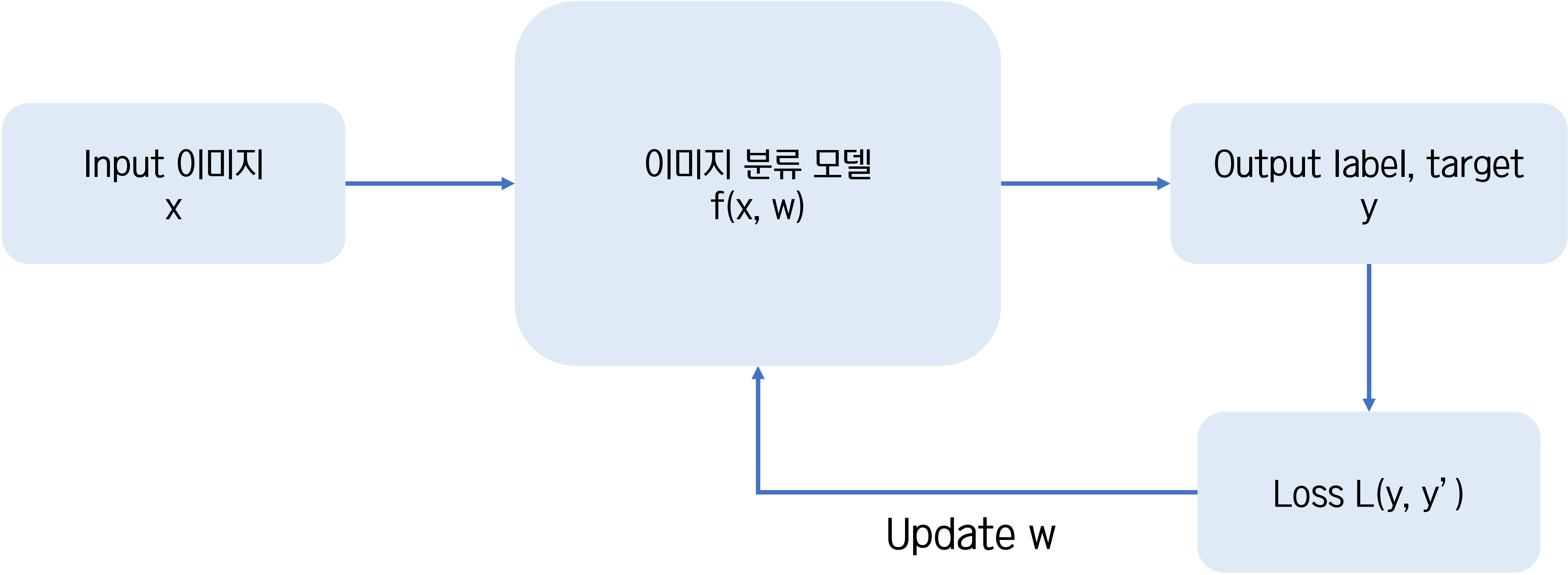
Review



01 Review


Input 이미지
x
이미지 분류 모델
f(x, w)
Output label, target
y
Loss L(y, y’)
Update w

01 Review



Optimization 최적의 w,
즉 loss를 최소로 만드는 w를 찾아가는 과정

Loss function 분류기의 성능을 **정량화**하는 방법

01 Review

Gradient descent

- 최적화 알고리즘 : loss를 **최소**로 만드는 w를 찾는 알고리즘
  -> loss function의 기울기(gradient)를 구하자!

- Numerical gradient (수치적 방법)
  - 하나하나 계산하는 방법으로 계산 속도가 너무 느림

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

- Analytical gradient (해석학적 방법)
  - 미분을 이용하는 방법으로 정확하고 계산 속도가 빠름

$$s = f(x; W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

want $\boxed{\nabla_W L}$

01 Review

Gradient descent

- 최적화 알고리즘 : loss를 **최소**로 만드는 w를 찾는 알고리즘
  -> loss function의 기울기(gradient)를 구하자!

- Numerical gradient (수치적 방법)
  - 하나하나 계산하는 방법으로 계산 속도가 너무 느림

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

- Analytical gradient (해석학적 방법)
  - 미분을 이용하는 방법으로 정확하고 계산 속도가 빠름

$$s = f(x;W) = Wx$$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

$$L = \frac{1}{N} \sum_{i=1}^{N} L_i + \sum_k W_k^2$$

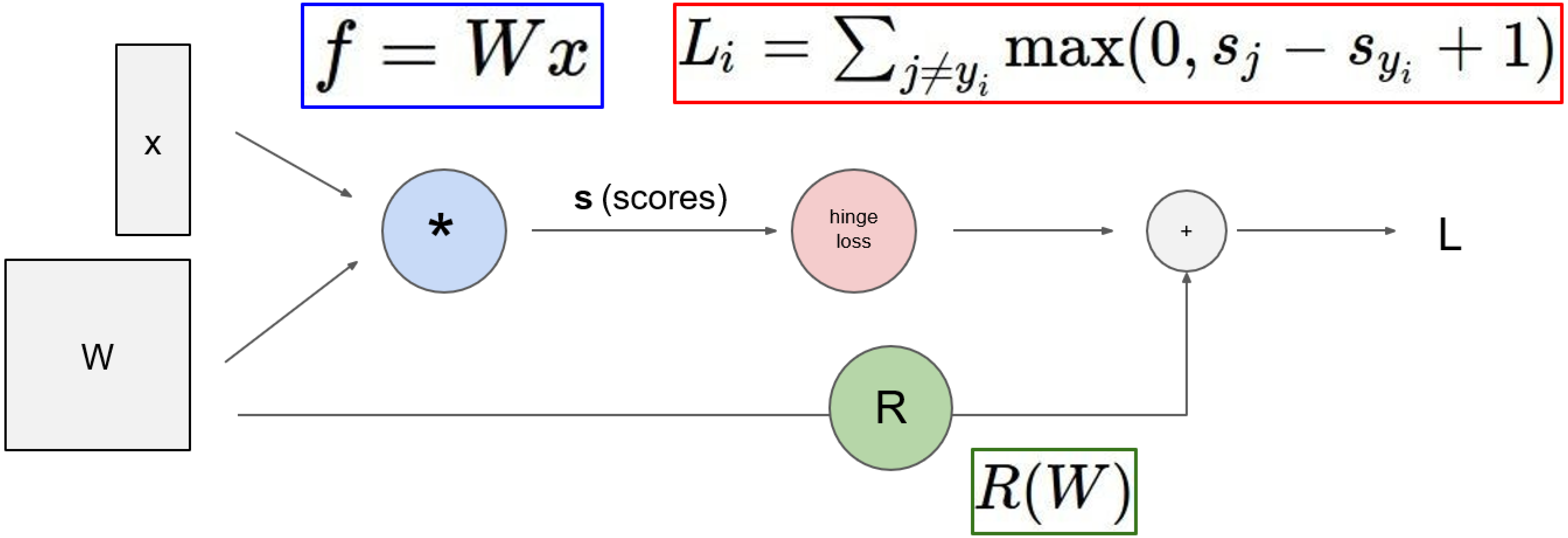
want $\boxed{\nabla_W L}$

gradient는 곧 공간에 대한 기울기!
-> 함수가 스칼라가 아닌 **벡터**를 인자로 가질 땐,
편미분(partial derivatives)을 사용

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right)$$

Computational graph

02 Computational graph



Back propagation

03 Back propagation

Forward pass 입력에 대한 출력 값을 계산

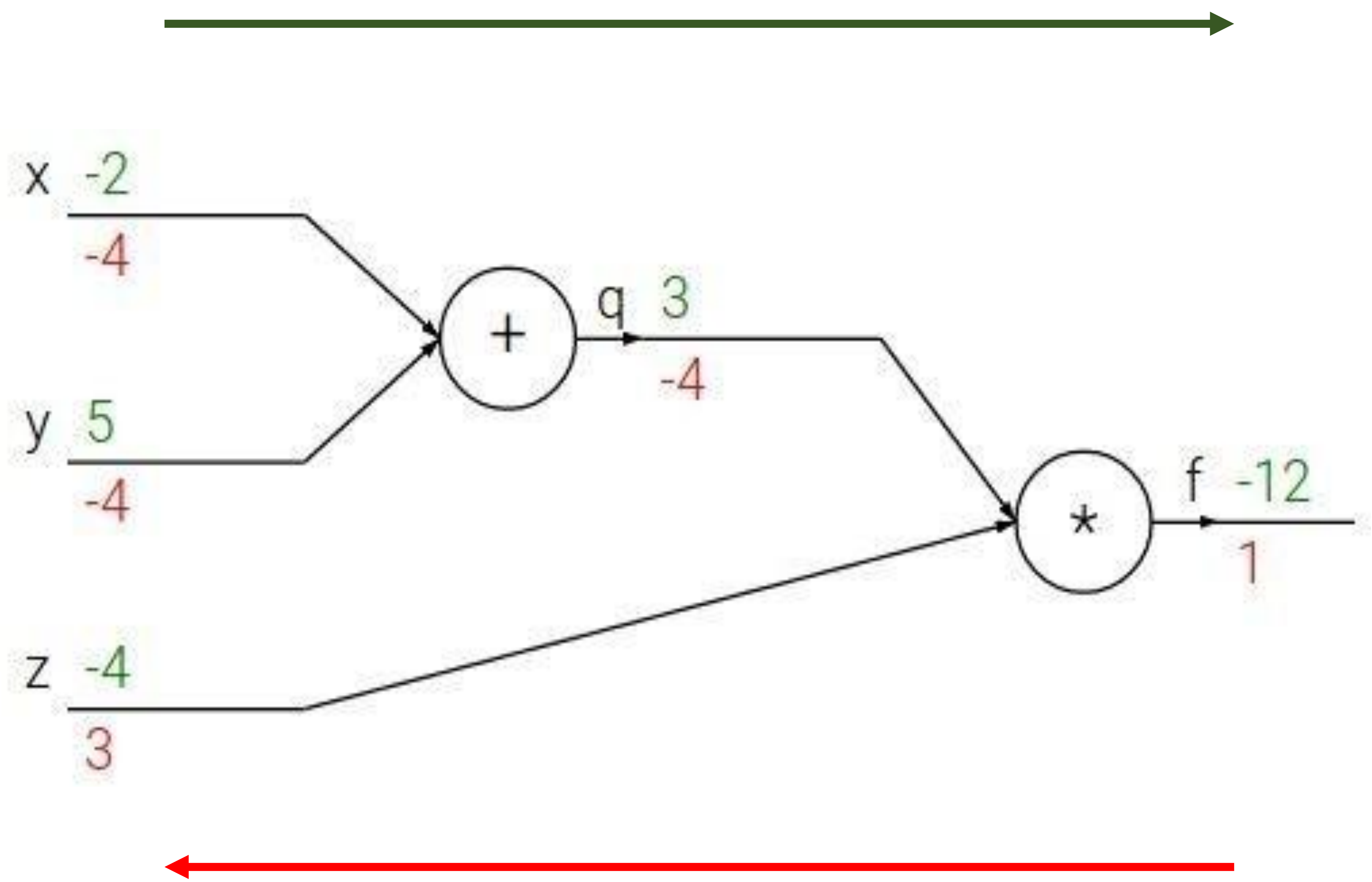
Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

그래디언트 ∇f는 편미분 벡터

$$\nabla f = [\ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}\ ]$$

"x에 대한 편미분"
"x에 대한 그래디언트"



Backward pass chain rule을 적용하여 편미분 값을 계산

03 Back propagation

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

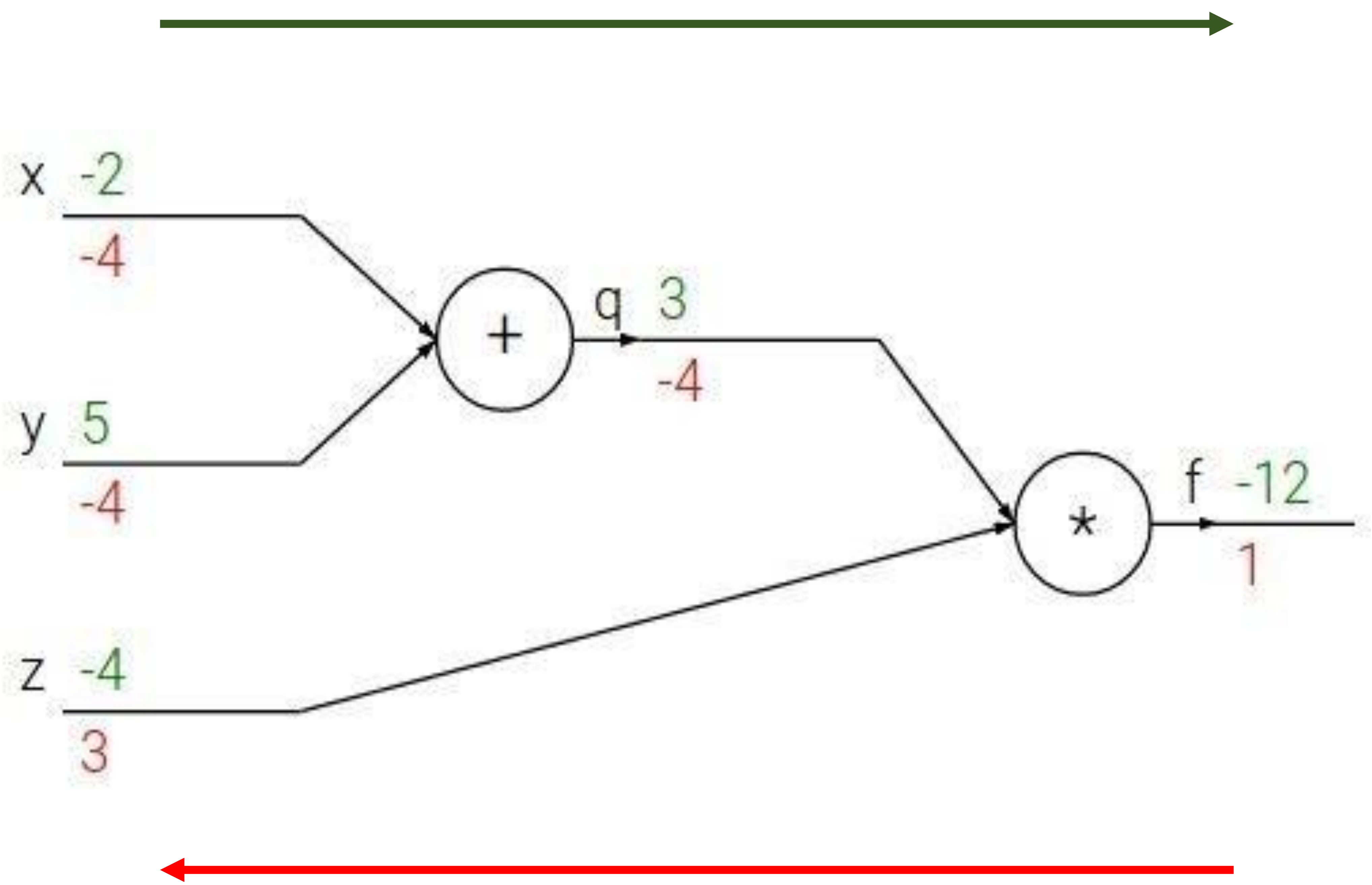
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

<u>Forward pass</u> 입력에 대한 출력 값을 계산



<u>Backward pass</u> chain rule을 적용하여 편미분 값을 계산

03 Back propagation

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

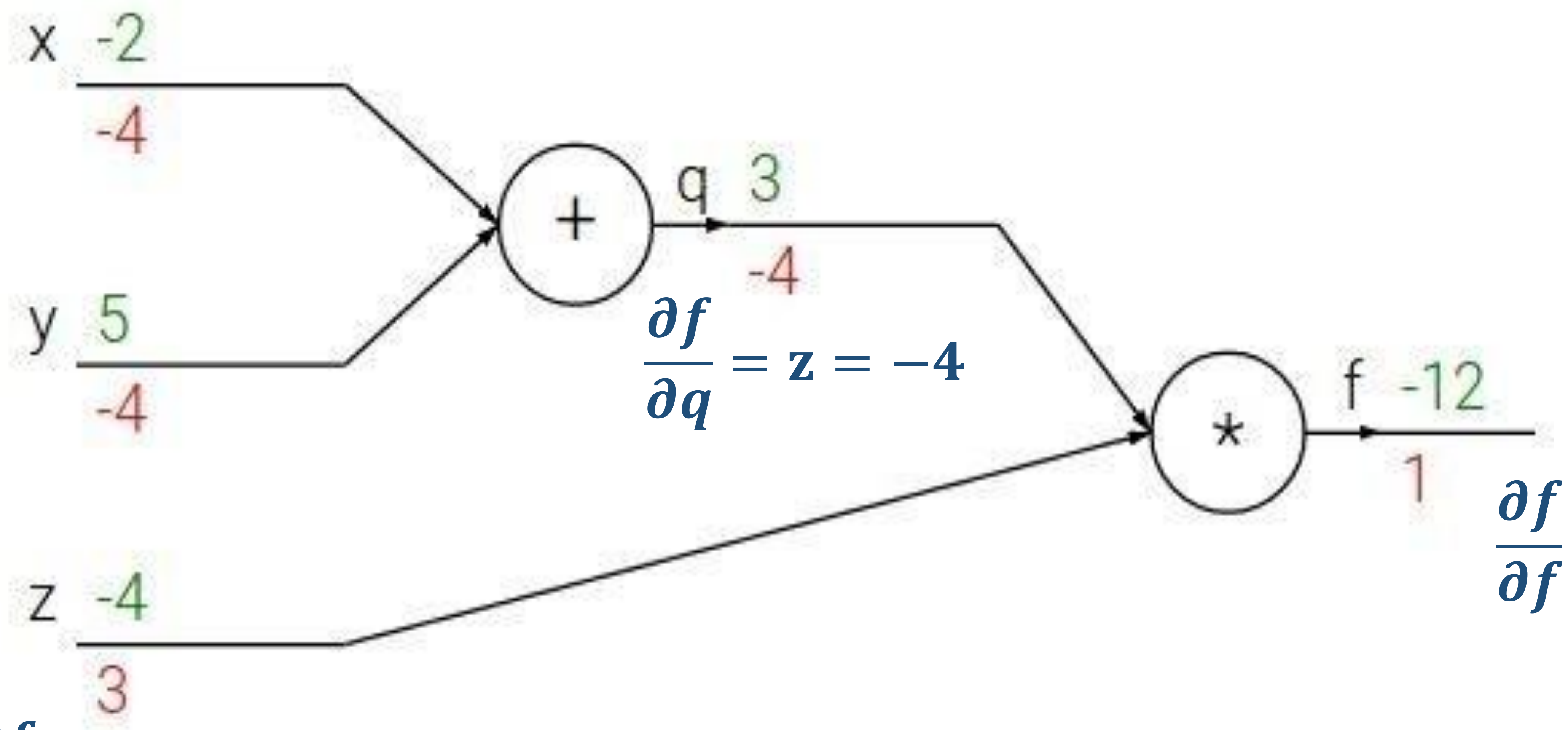
e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} * \frac{\partial q}{\partial x} = -4$$



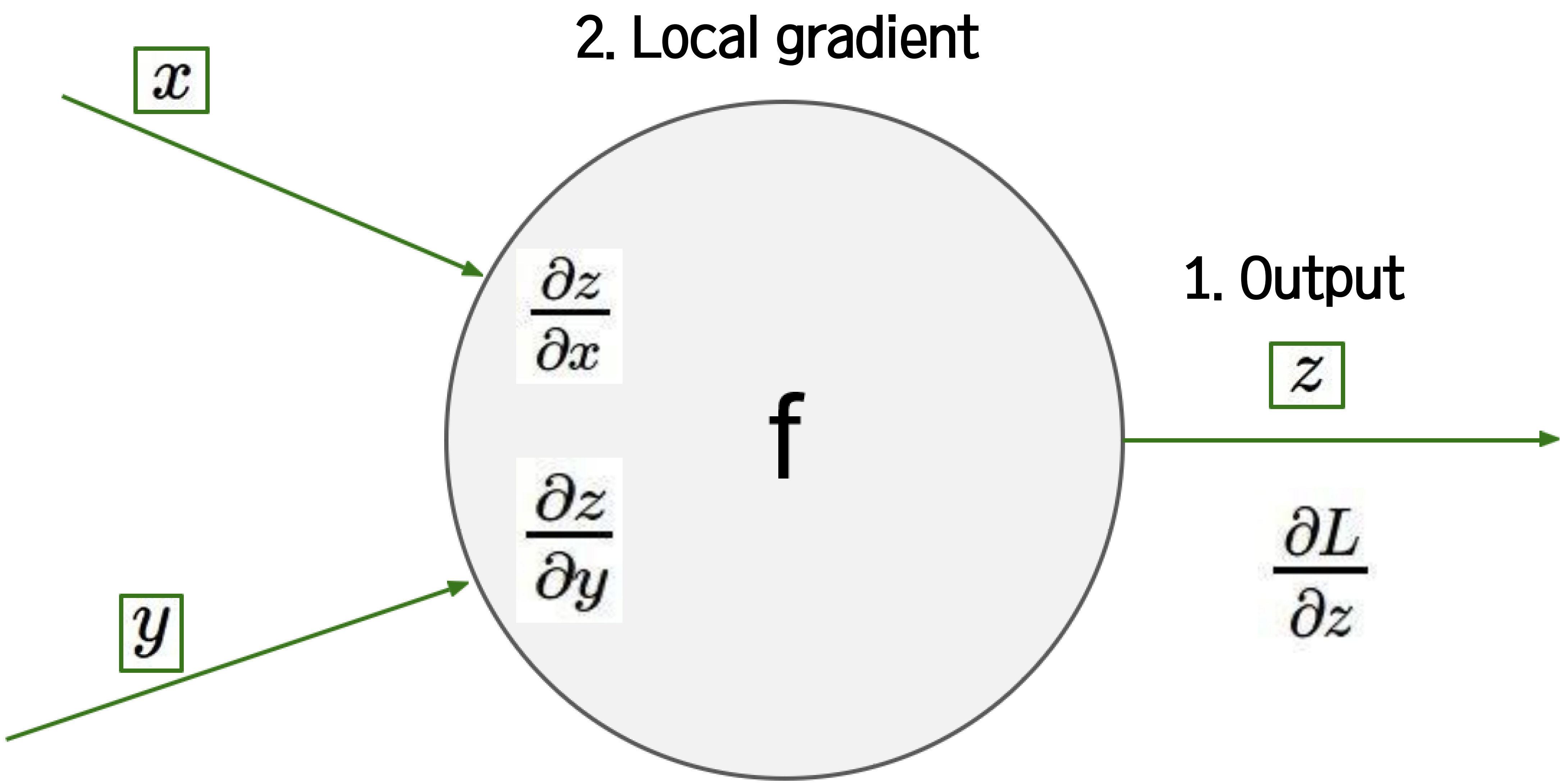
$$\frac{\partial f}{\partial q} = \mathbf{z} = -4$$

$$\frac{\partial f}{\partial f} = 1$$

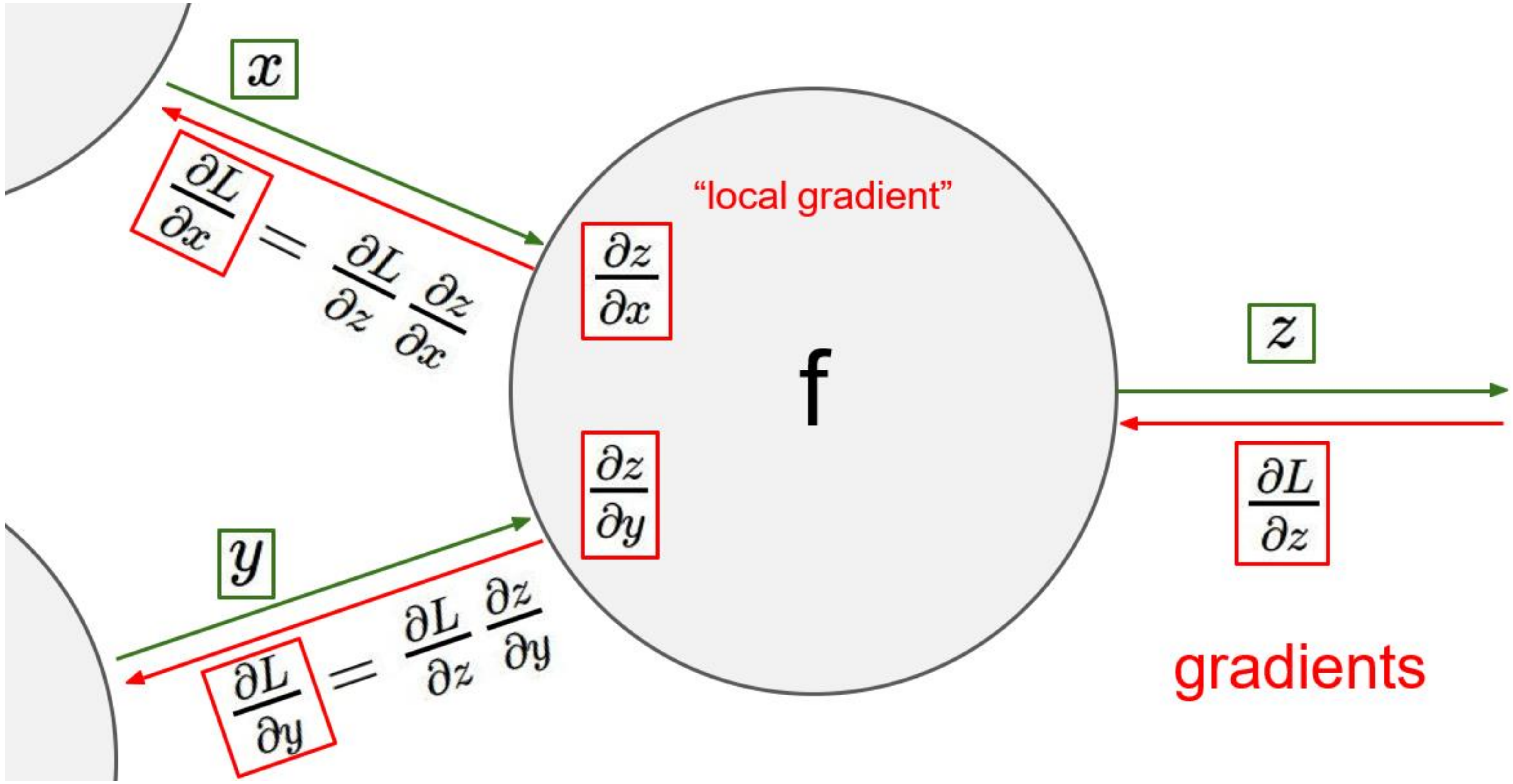
$$\frac{\partial f}{\partial z} = \mathbf{q} = 3$$

Backward pass chain rule을 적용하여 편미분 값을 계산

03 Back propagation

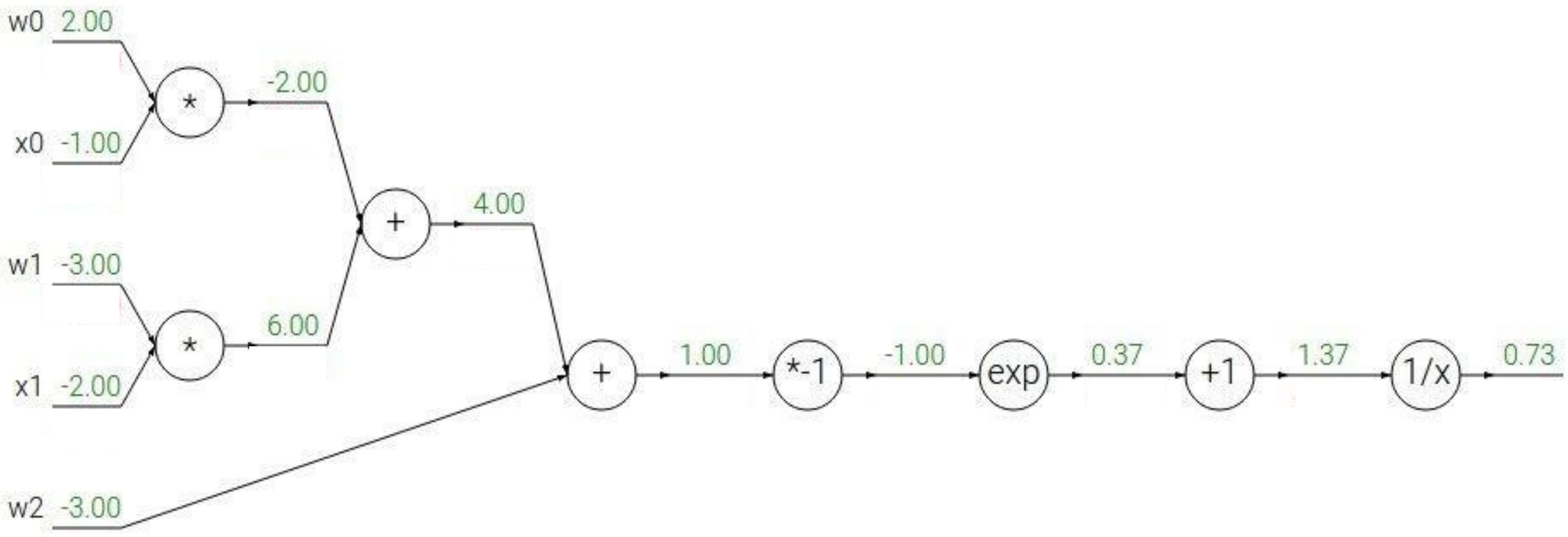


03 Back propagation



03 Back propagation

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$



03 Back propagation

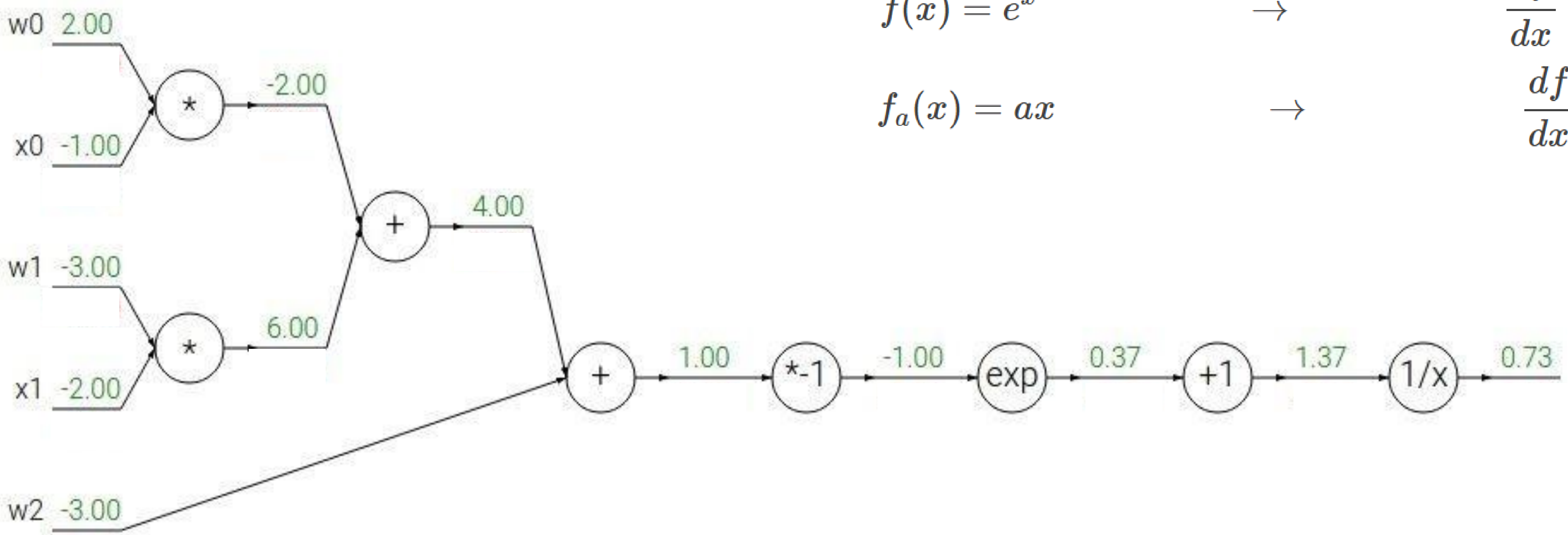
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$



EWHA EURON

03 Back propagation

$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = \frac{1}{x} \quad \rightarrow \quad \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \quad \rightarrow \quad \frac{df}{dx} = 1$$

$$f(x) = e^x \quad \rightarrow \quad \frac{df}{dx} = e^x$$

$$f_a(x) = ax \quad \rightarrow \quad \frac{df}{dx} = a$$

Modularity

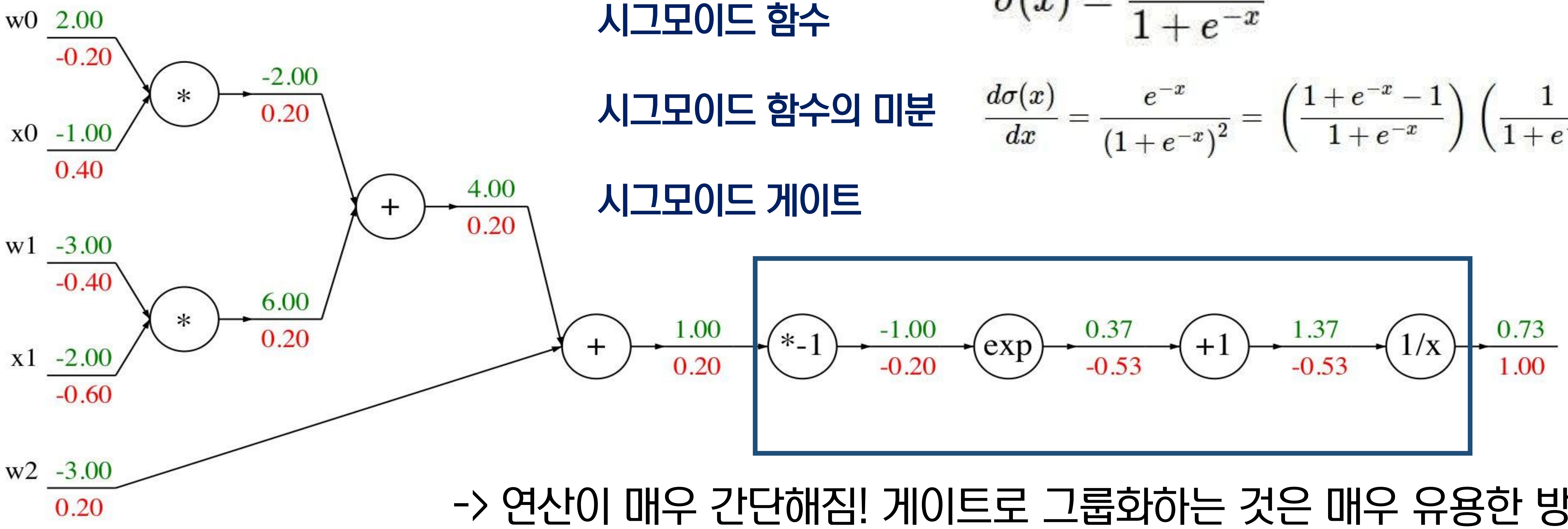
시그모이드 함수

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

시그모이드 함수의 미분

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\sigma(x)$$

시그모이드 게이트



-> 연산이 매우 간단해짐! 게이트로 그룹화하는 것은 매우 유용한 방법임

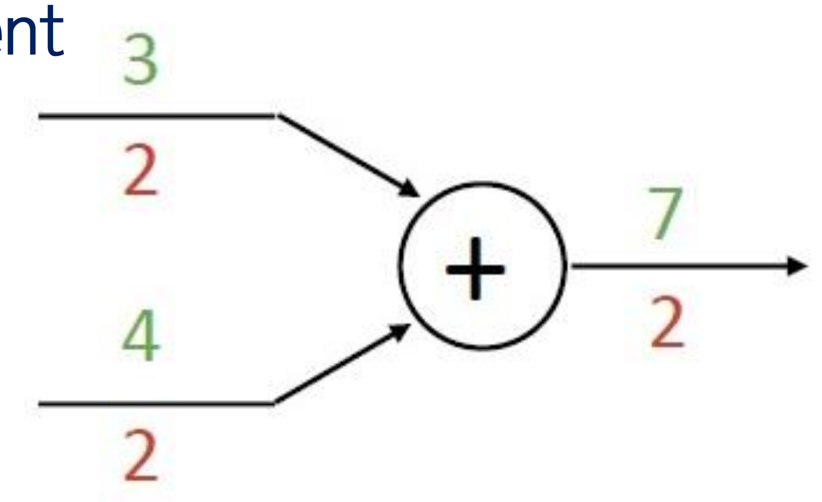
03 Back propagation

Local gradient = 1
Gradient = upstream gradient

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

$$\frac{\partial f}{\partial x} = 1 \qquad \frac{\partial f}{\partial y} = 1$$

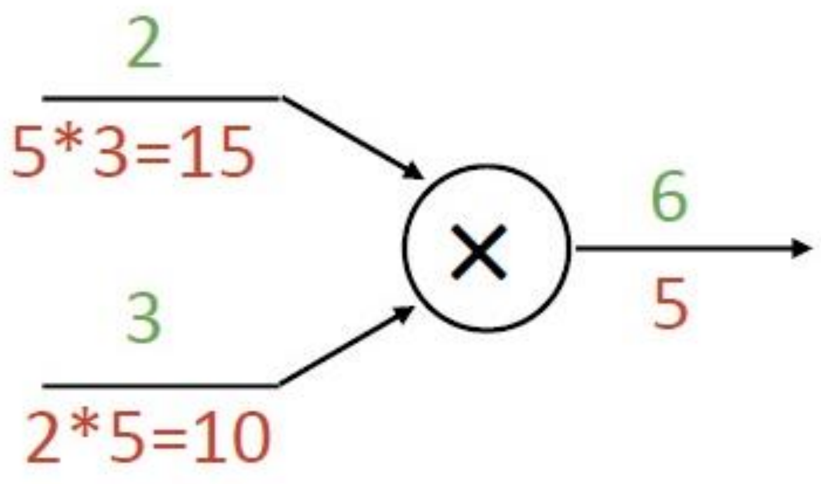
add gate: gradient distributor



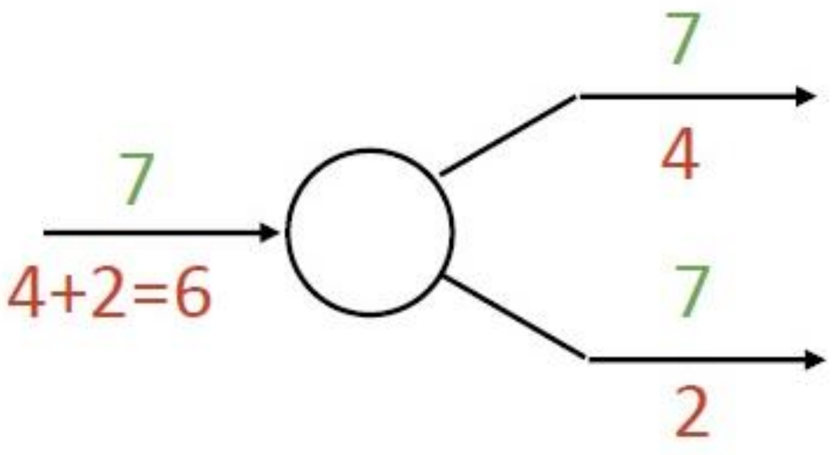
$$f(x,y) = xy$$

$$\frac{\partial f}{\partial x} = y \qquad \frac{\partial f}{\partial y} = x$$

mul gate: "swap multiplier"

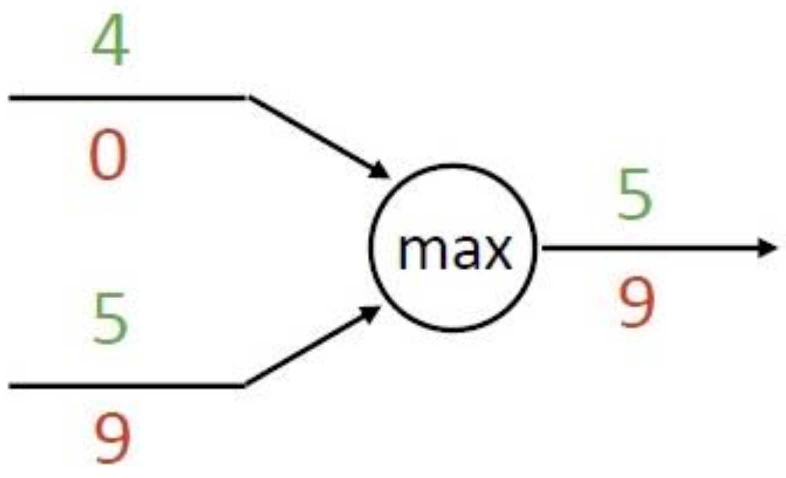


copy gate: gradient adder



local gradient = upstream에서 온 **gradient1**
+ upstream에서 온 **gradient2**

max gate: gradient router



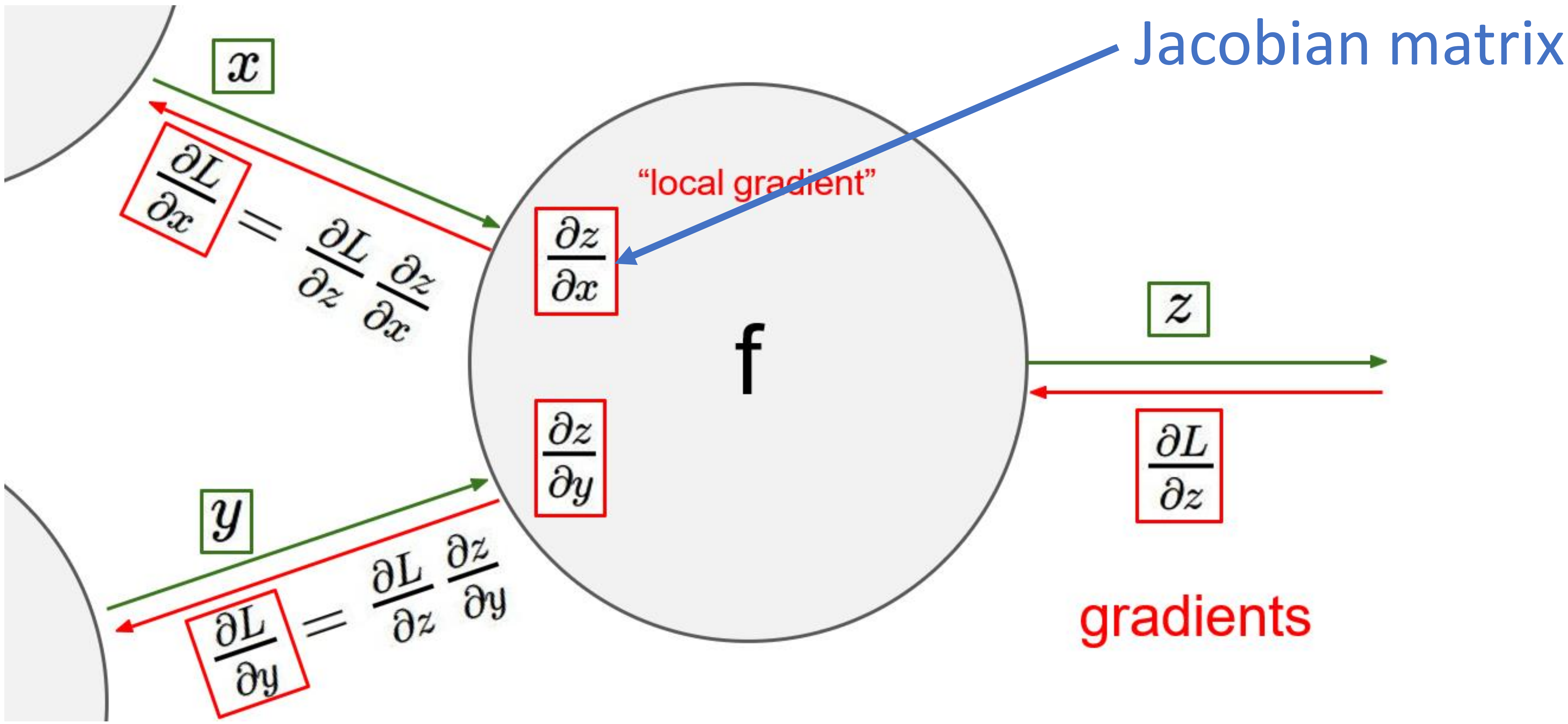
$$f(x,y) = \max(x,y)$$

$$\frac{\partial f}{\partial x} = 1(x >= y) \qquad \frac{\partial f}{\partial y} = 1(y >= x)$$

03 Back propagation

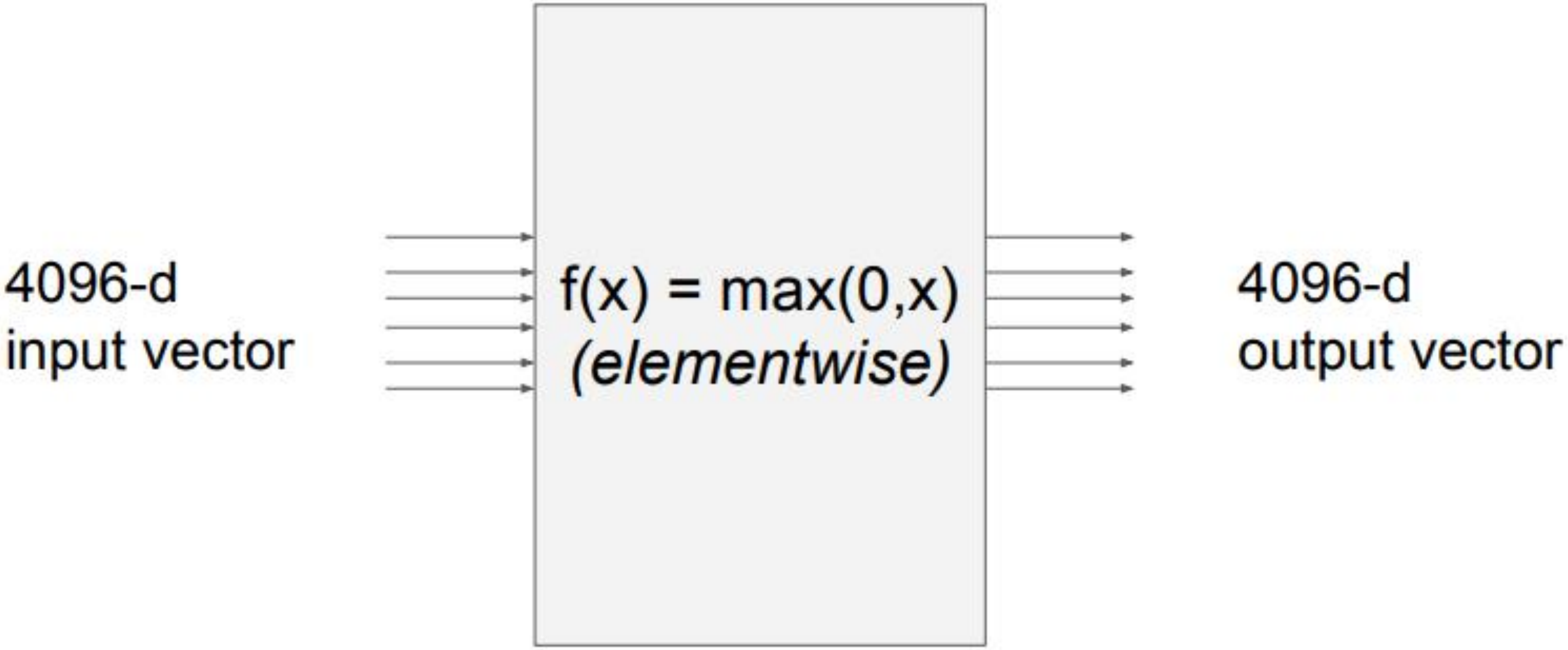
```
w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the sigmoid gradient
derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the circuit
```

03 Back propagation



03 Back propagation

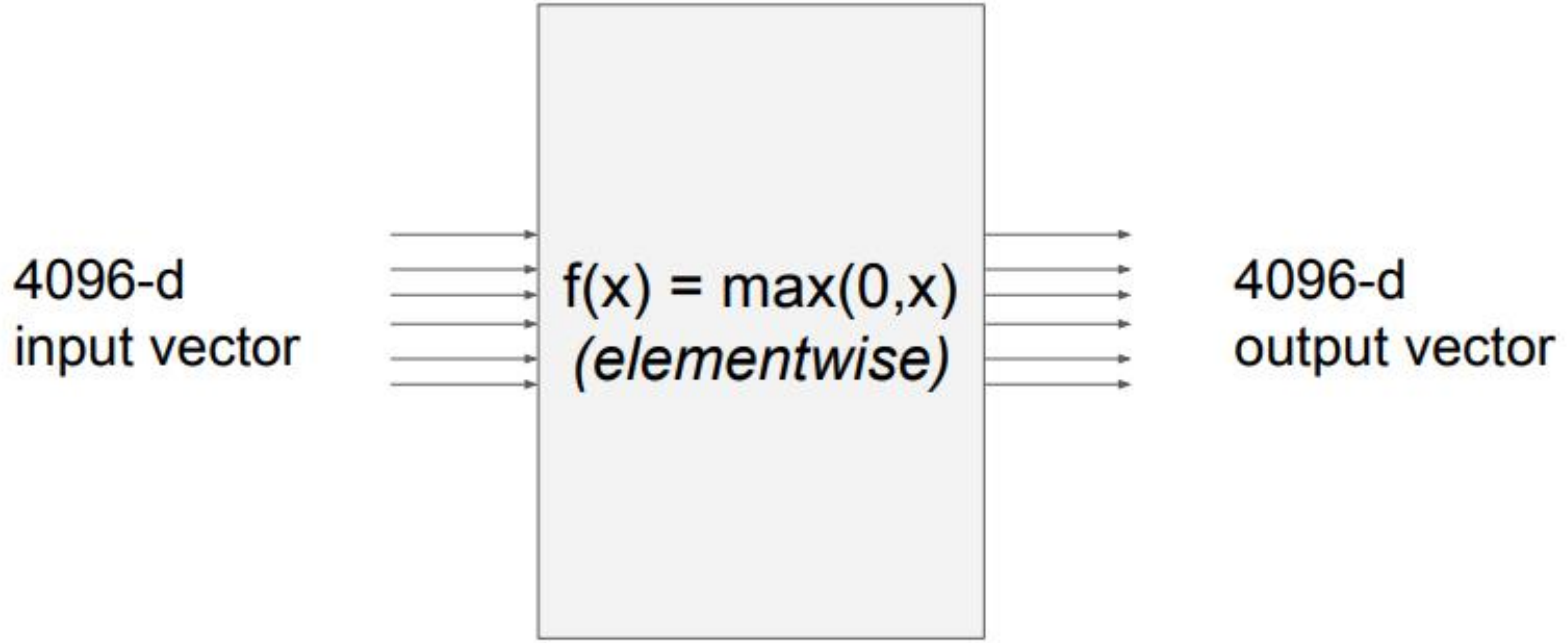
Q. What is the size of the Jacobian matrix?



A. [4096 x 4096]

03 Back propagation

Q. What does Jacobian matrix look like?



A. 대각행렬

03 Back propagation

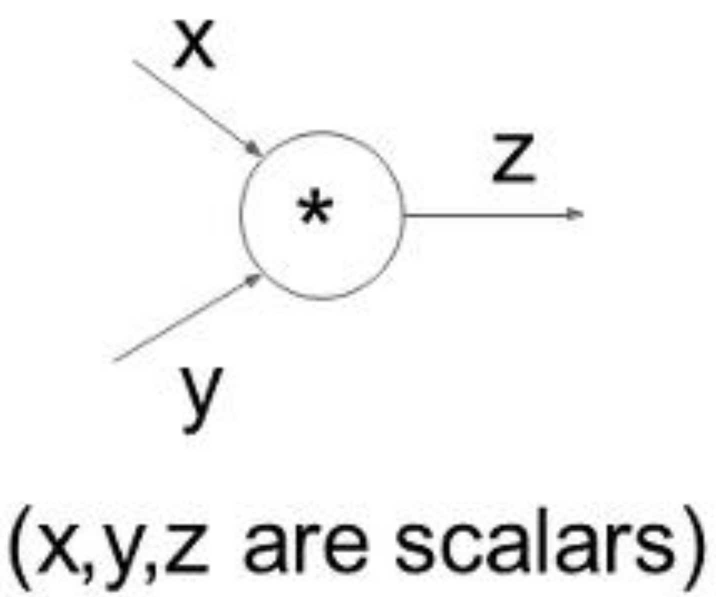
Jacobian matrix 예제

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$\in \mathbb{R}^n \in \mathbb{R}^{n \times n}$

03 Back propagation

forward() / backward() API



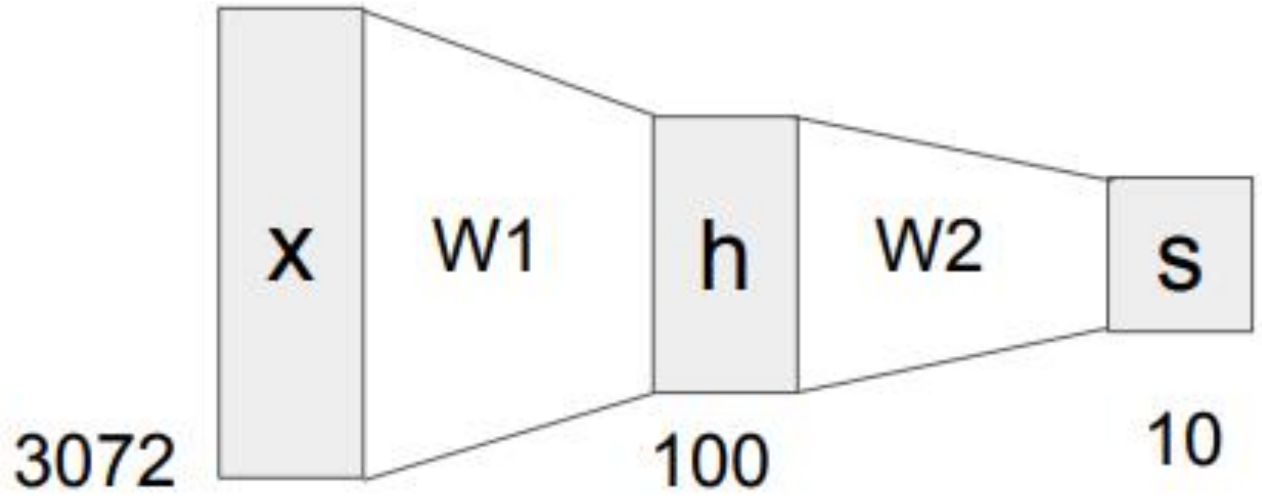

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

Forward: 인자 값인 x와 y를 곱해서 z값을 도출한 후 z 리턴

Backward: loss 값 L을 z로 미분한 값을 인자로 가져오고, L을 x와 y로 미분한 값을 리턴

Neural Networks

04 Neural Network

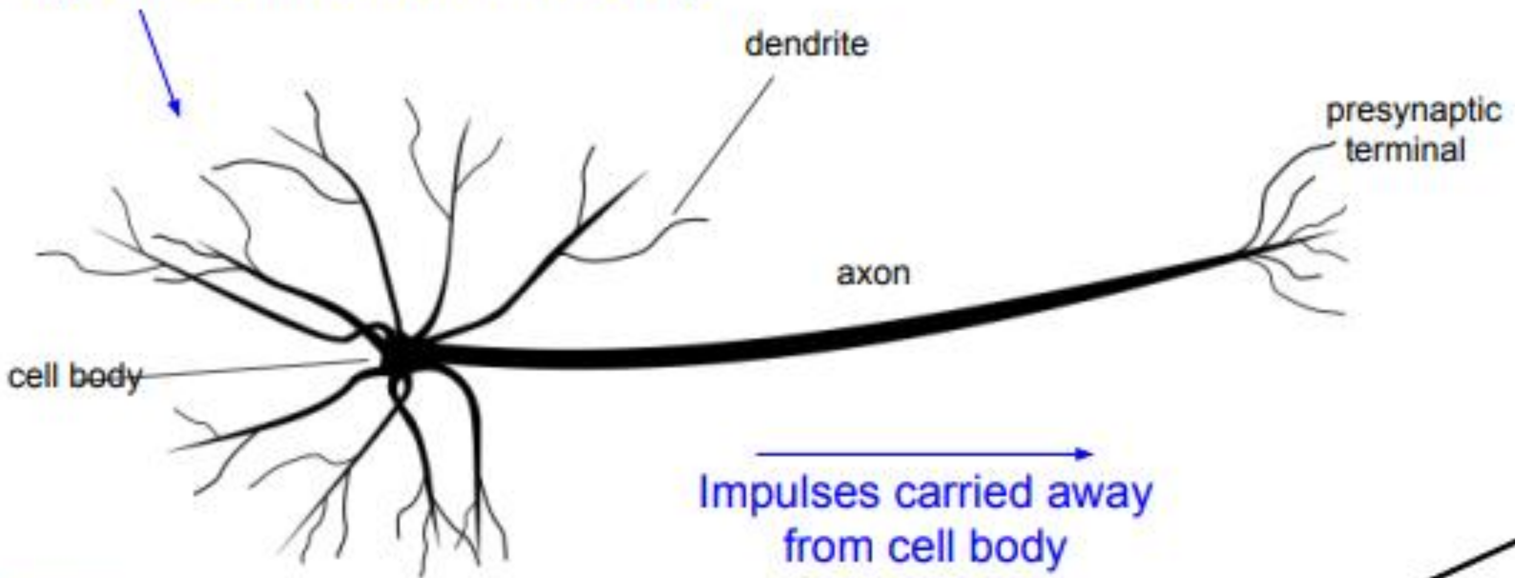


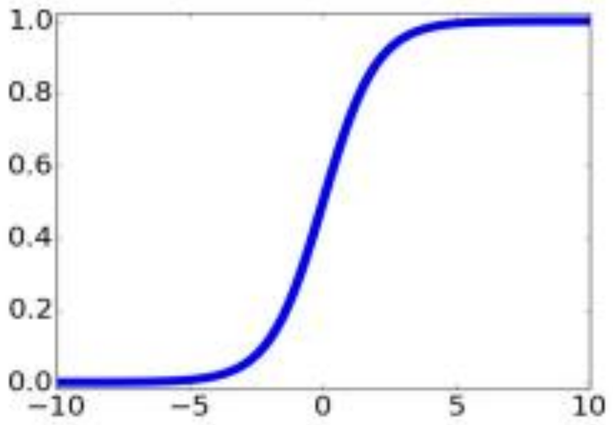
(**Before**) Linear score function: $f = Wx$

(**Now**) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
or 3-layer Neural Network

$$f = W_3 \max(0, W_2 \max(0, W_1 x))$$

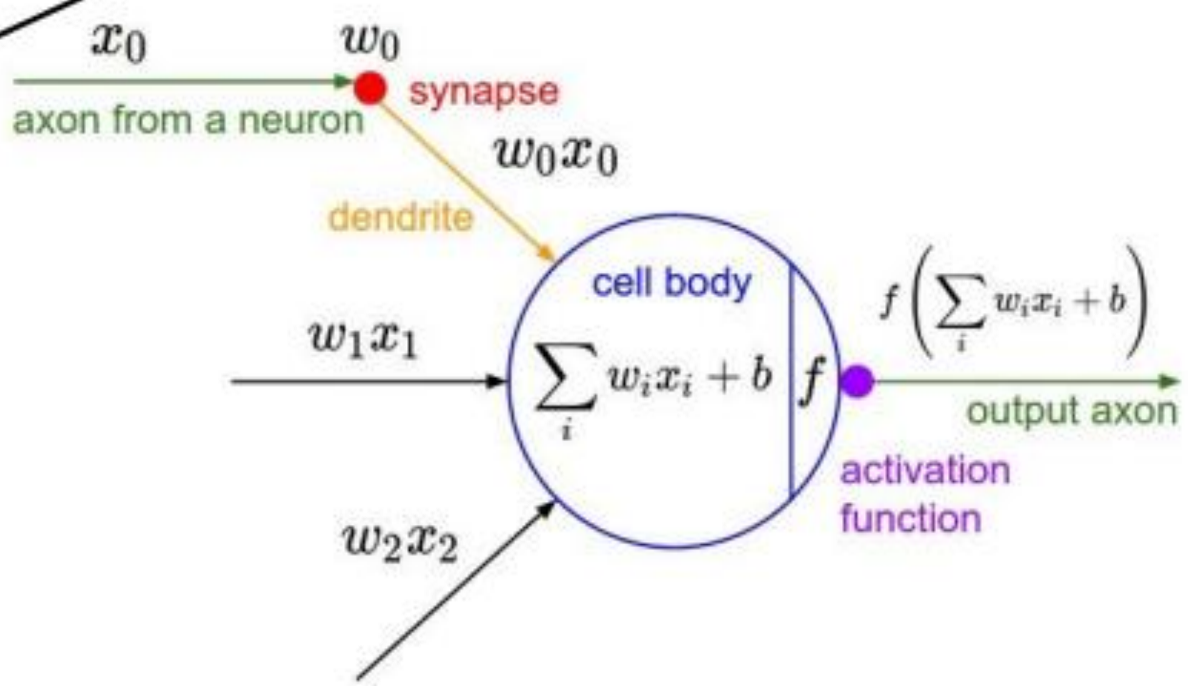
04 Neural Network





sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



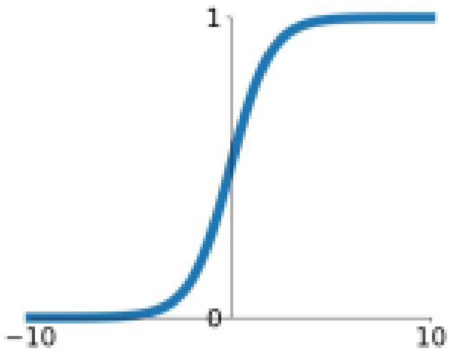

```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation func
    return firing_rate
```

04 Neural Network

Activation function

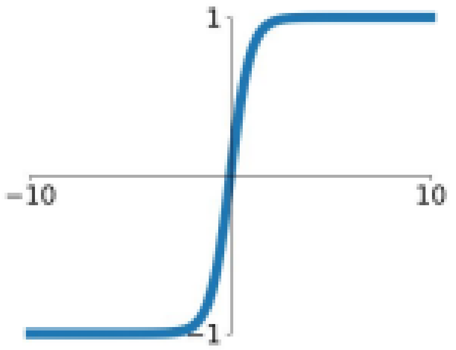
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



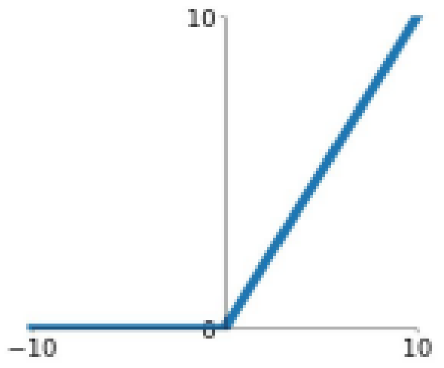
tanh

$\tanh(x)$



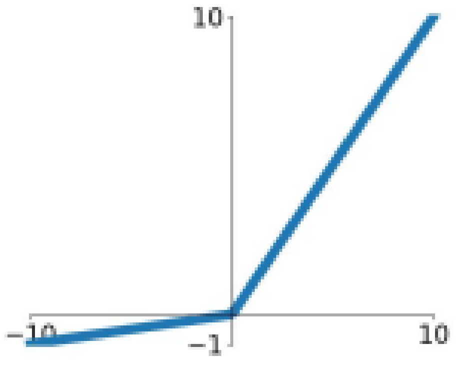
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

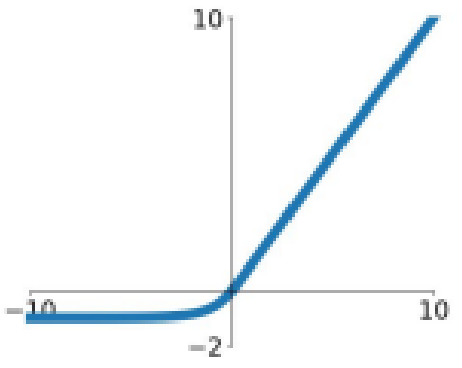


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

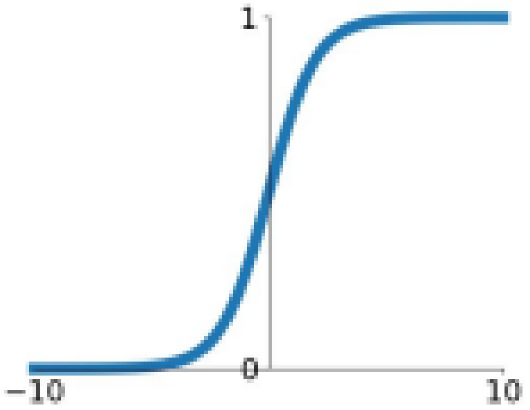
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



04 Neural Network

Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



- Output 값을 0 ~ 1로

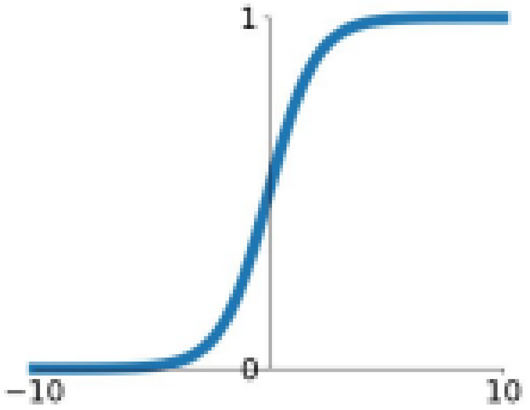
- 대부분의 경우에서 Sigmoid 함수는 좋지 않기 때문에 잘 사용하지 않음
  ➡ binary classification 경우 예외로 Sigmoid 함수를 사용

- 그래프를 보면 input값이 어느정도 크거나 작으면 기울기가 아주 작아짐
  ➡ “Vanishing gradient” 문제 발생

04 Neural Network

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



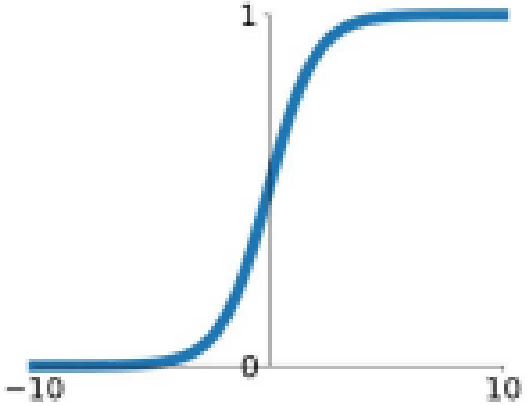
“Vanishing gradient”

: Sigmoid로 여러 layer를 쌓았을 때, 입력층 쪽으로 갈수록 대부분의 노드에서 기울기가 0이 되어 결국 gradient가 사라짐.

04 Neural Network

Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



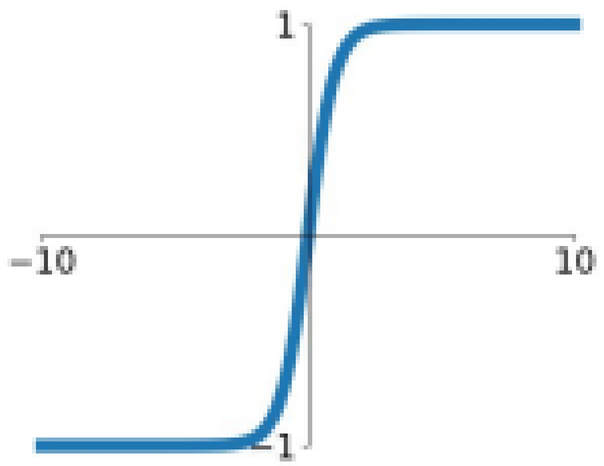
👍**장점**: binary classification을 **사용할 때**

👎**단점**: Vanishing gradient

04 Neural Network

tanh

$$\tanh(x)$$



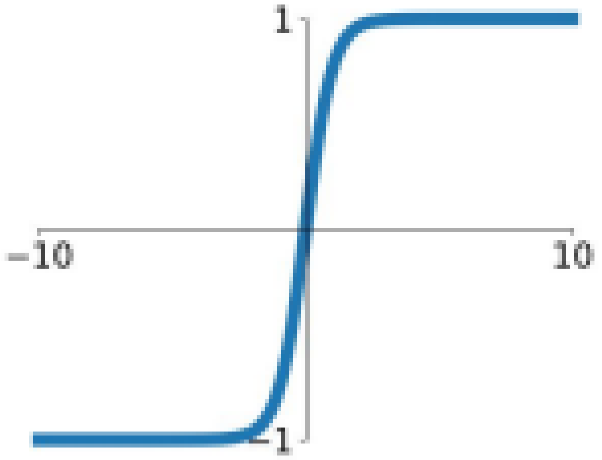
- Sigmoid 함수와 유사
    - 공통점: Vanishing gradient
    - 차이점: output 값이 -1 ~ 1

- 대부분의 경우에서 Sigmoid 함수보다 성능이 좋음

04 Neural Network

tanh

$$\tanh(x)$$



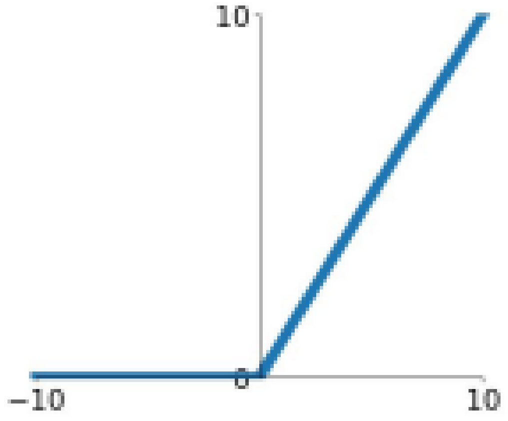
👍**장점**: Sigmoid**보다 대부분의 경우에서 학습이 더 잘 됨**

👎**단점**: Sigmoid**와 마찬가지로** Vanishing gradient

04 Neural Network

ReLU

$\max(0, x)$



- 일반적으로 ReLU의 성능이 가장 좋아서 많이 사용

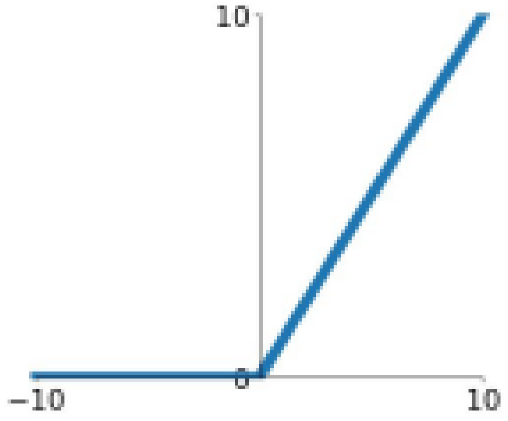
- 대부분의 input 값에 대해 기울기가 0이 아니기 때문에 학습이 빨리 됨
  ➡ 학습을 느리게 하는 원인이 gradient가 0이 되는 것
  ➡ hidden layer에서 대부분 노드의 z값이 0보다 크기 때문에 기울기가 0이 되는 경우가 적음

- X가 0보다 작을 경우, 기울기가 0이기 때문에 학습 과정에서 뉴런이 죽는 경우가 생겨 값이 더 이상 업데이트 되지 않음

04 Neural Network

ReLU

$\max(0, x)$



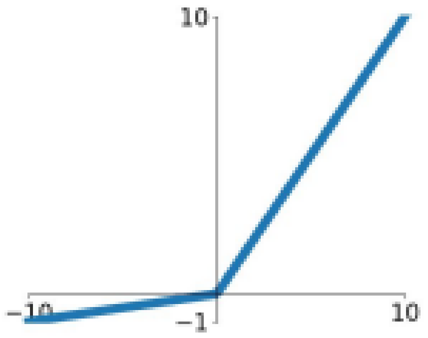
👍장점: 대부분의 경우에서 기울기가 0이 되는 것을 방지해주기 때문에 학습이 빠르게 잘 됨

👎단점: z가 음수일 때 기울기가 0

04 Neural Network

Leaky ReLU

$\max(0.1x, x)$



- ReLU의 단점을 해결하기 위해 등장한 함수

  ➡ x가 0이하인 구간에 죽는 뉴런이 생겨 학습을 할 수 없었던 문제

- ReLU와의 차이점: max(0,z)가 아니라 max(0.01z, z)

  ➡ Input 값인 z가 음수일 경우 기울기가 0이 아닌 0.01

- 많이 쓰이진 않지만 ReLU보다 학습이 더 잘 됨

👍장점: ReLU보다 학습이 더 잘 됨

👎단점: 음수에서 선형성이 생겨 복잡한 분류에선 사용 불가

04 Neural Network

Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

- ReLU와 Leaky ReLU를 일반화한 함수
  ➡ ReLU가 가진 장점을 모두 가짐

- 각 노드별로 학습을 시켜야 할 파라미터 weight, bias를 추가
  ➡ 연산량이 두 배 이상 증가

- 성능 대비 인기 없음

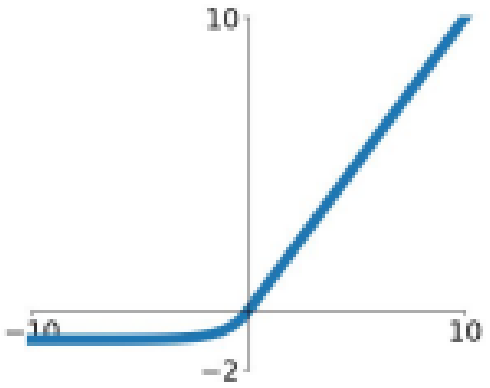
👍장점: ReLU의 장점

👎단점: 연산량 증가

04 Neural Network

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



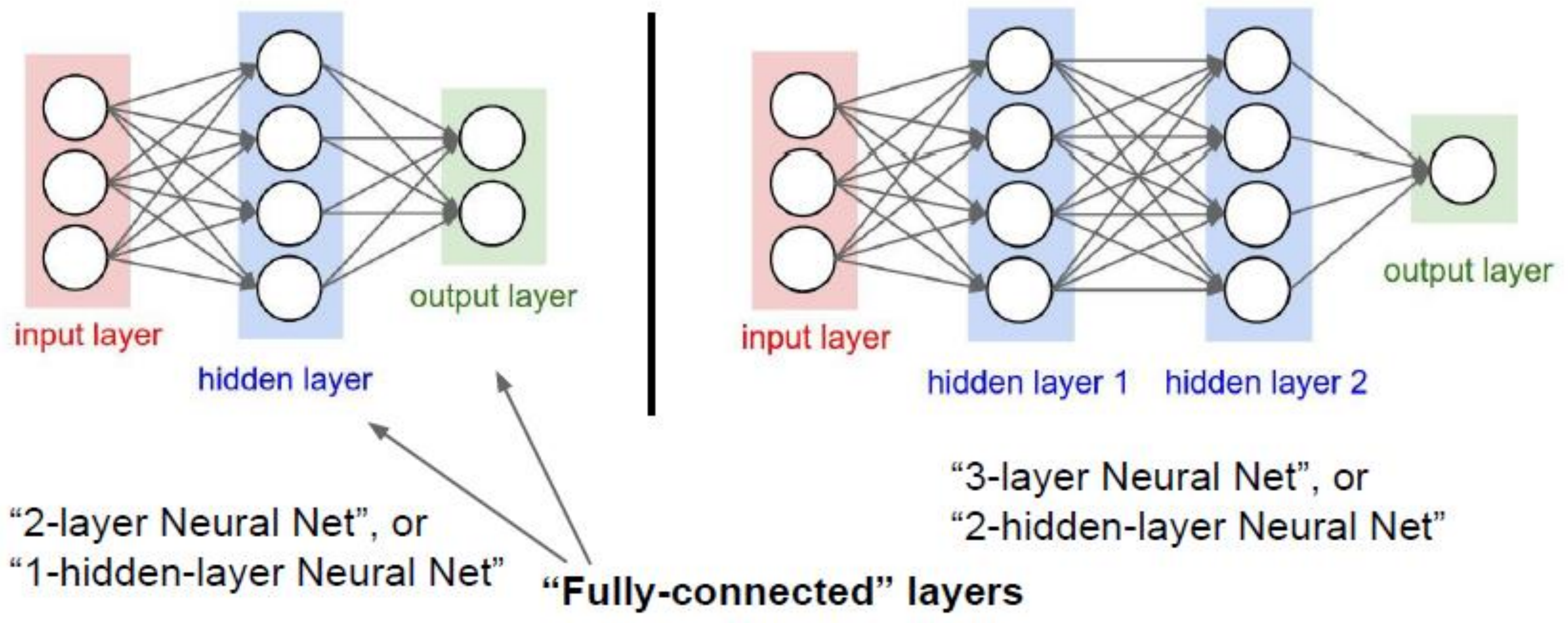
- ReLU가 가진 장점을 가짐

- ReLU와의 차이점: 하이퍼파라미터 α는 x가 음수일 때 수렴하는 값을 정의 (보통 1)
  ➡ Leaky ReLU처럼 죽은 뉴런을 만들지 않는다는 장점
  ➡ α가 1일 때, x=0에서 매끄럽게 변하기 때문에 gradient decent에서 수렴 속도가 빠름

👍장점: ReLU + Leaky ReLU 장점
👎단점: exp()에 대한 미분값 계산이 필요해 연산 비용

04 Neural Network

Fully-Connected Layer



Q. Layer가 3개인데 왜 2-layer?
A. Weight를 갖고 있는 것만 layer라고 부름. Input layer의 경우 weight를 가지지 않기 때문에 제외

04 Neural Network

Fully-Connected Layer vs. 1x1 Convolution Layer

- Weight 수가 같을 때 FC 출력 뉴런 1개와 1x1 Conv. 출력 평면 1개 대응

📌차이점:

- 1x1 Conv.를 사용하면 평면 위의 픽셀이 그대로 남아있게 되어 위치정보가 남음

- FC는 출력 뉴런 1개가 input tensor에 있는 모든 뉴런에 대해 의존도 따짐

  ➡ channel 축이든 spatial 축이든 global하게 dependent
- 1x1 Conv.는 channel 축으로 모든 픽셀들을 고려하지만, spatial 축으로는 고려하지 않음

THANK YOU