Introduction to Artificial Intelligence

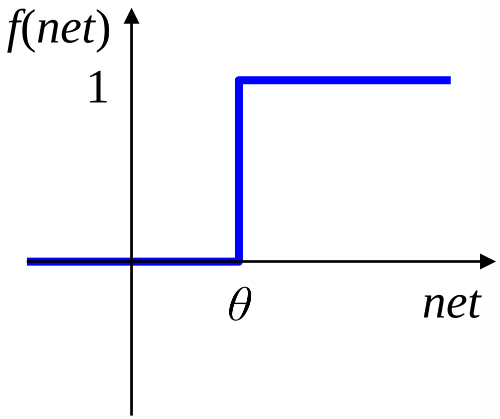
2019

Ha-Jin Yu, Dept. of Computer Science, University of Seoul
서울시립대학교 컴퓨터과학부 유하진

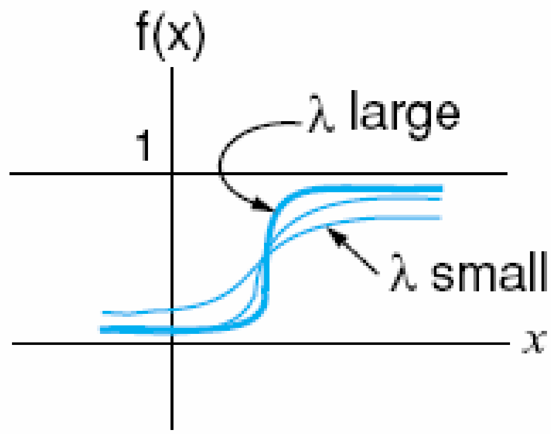
HJYU@UOS.AC.KR

서울시립대학교
UNIVERSITY OF SEOUL

Activation (Thresholding) functions.



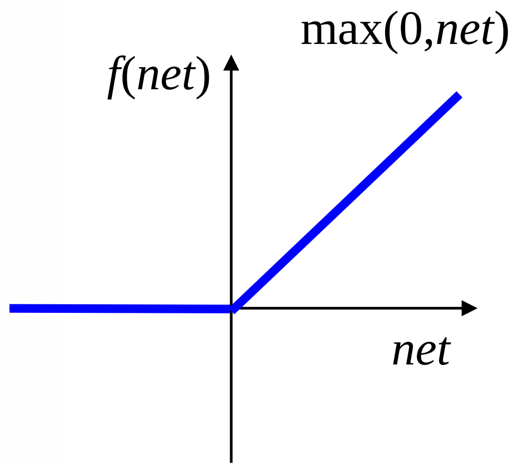
Hard limiting



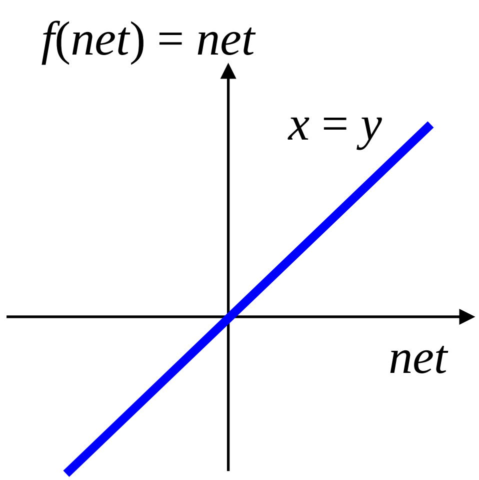
Sigmoid

$$\sigma(net) = \frac{1}{1 + e^{-\lambda \cdot net}}$$

$$net = \sum_{i=0}^{n} x_i w_i$$



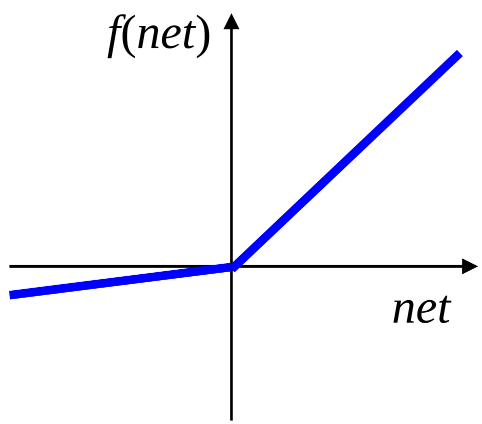
Rectified linear unit (ReLU)



Linear

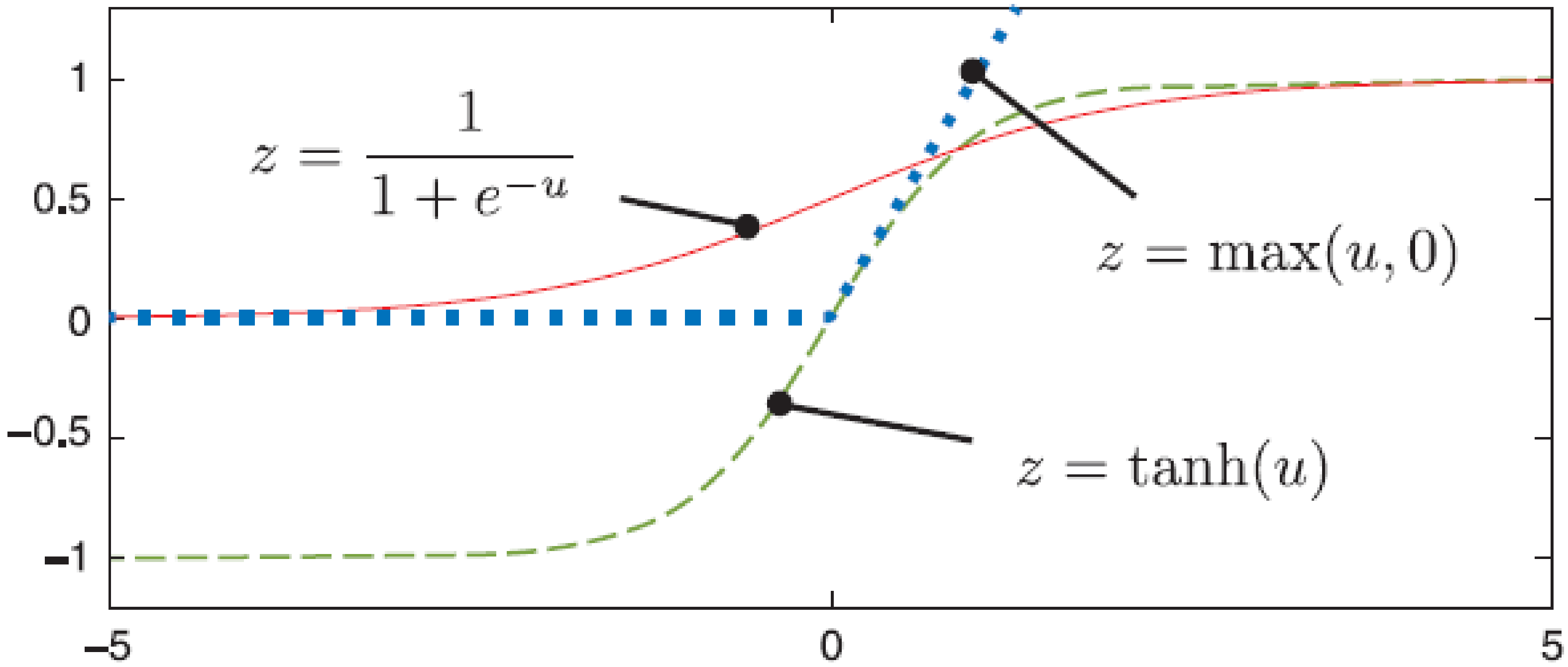
Maxout

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



Leaky Rectified linear unit

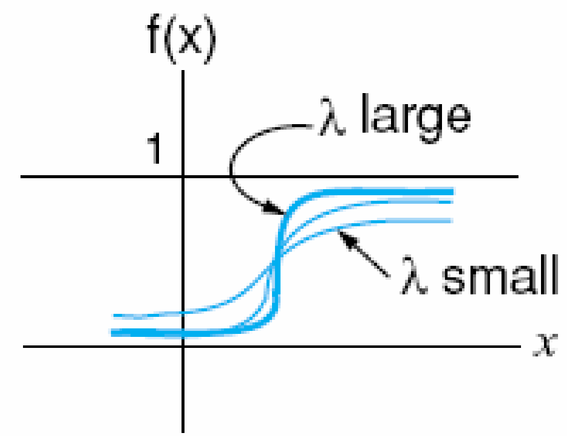
Activation (Thresholding) functions



딥 러닝 제대로 시작하기, 제이펍, 오카타니 타카유키, 심효섭 역, 2016

Sigmoid function

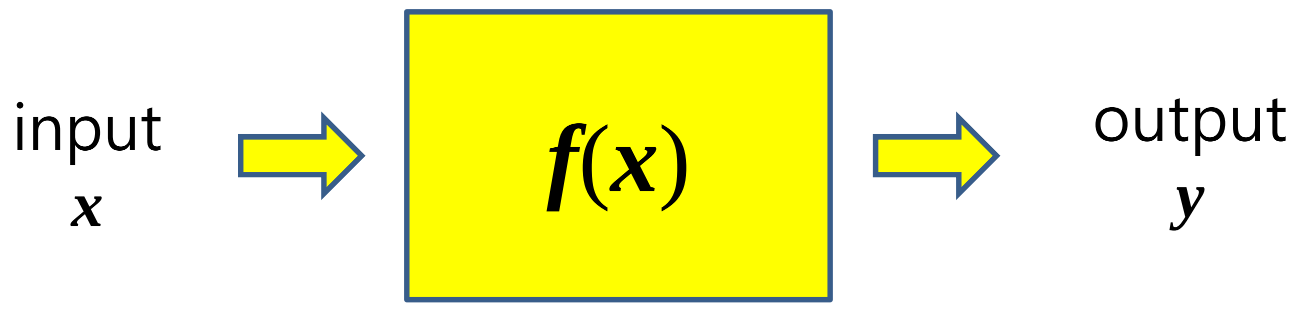
- *Logistic* function
- λ : "squashing parameter" used to fine-tune the sigmoidal curve.
  - 함수의 기울기 조절
  - 큰값 → linear threshold function over {0,1};
  - 1 에 근접 → 선형.
- 연속 함수 이면서 thresholding 특성



Learning

학습 (learning)

- 같은 구조이지만 weights 값에 따라서 output 이 달라진다.



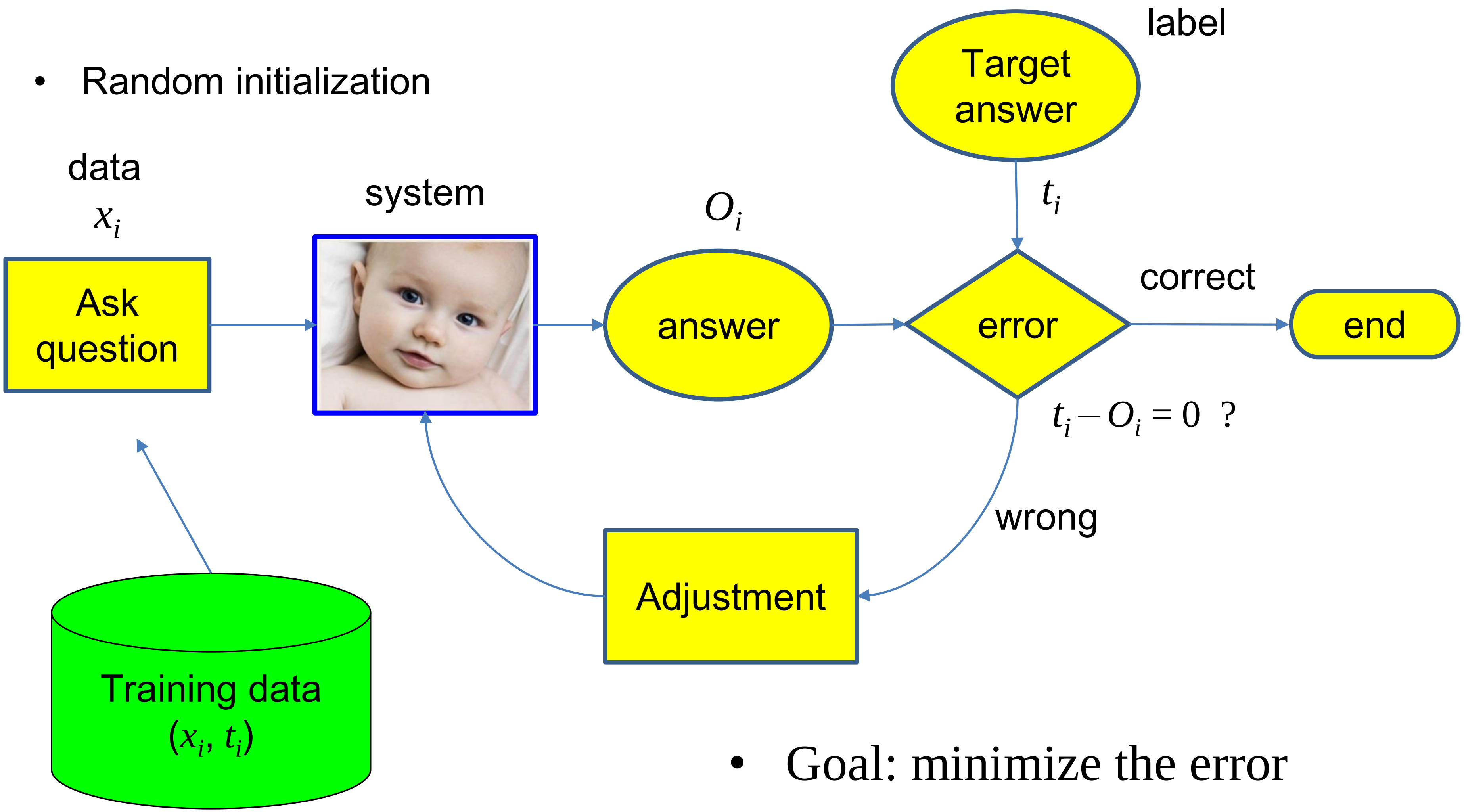
- Goal: Input 에 맞는 output을 구하는 함수 $f(x)$를 찾는 것.
- 함수 $f(x)$를 찾는 것은?

  == Input 에 맞는 output을 구하는 weights를 찾는다는 것.

  → 학습 (learning)
- Weights의 개수는?

Principle of learning

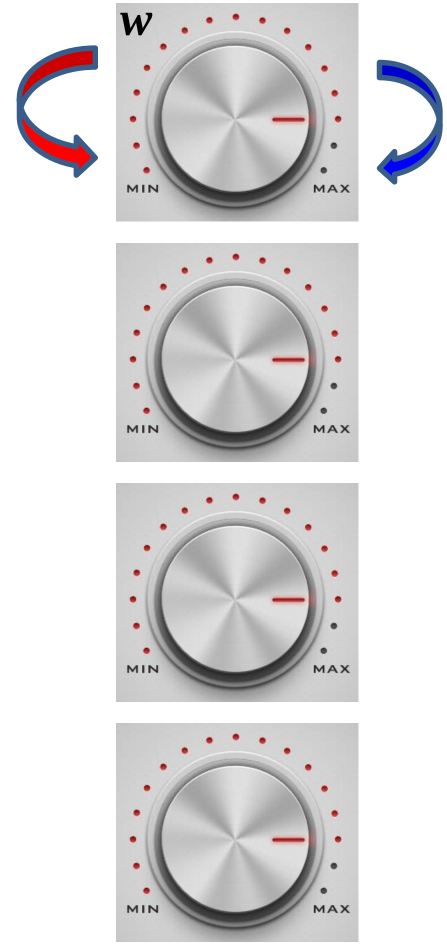
- Random initialization



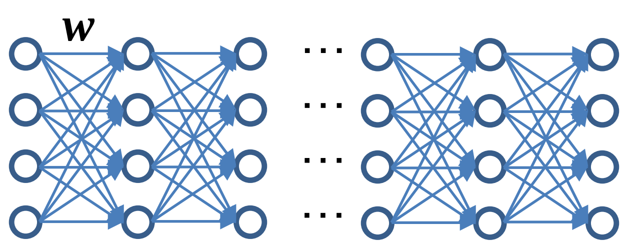
- Goal: minimize the error

Weights Adjustment

Just try.



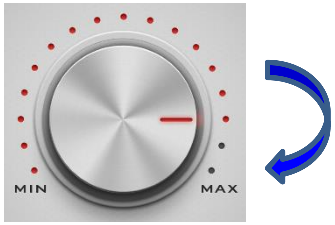
input
$\boldsymbol{x}$



output
O

Weights Adjustment

- You can calculate the direction to turn …



Goal

- Minimize the errors

- Errors
  - 틀린 개수
  - 정답(target)과 system의 output $f(\boldsymbol{x})$ 와의 차이

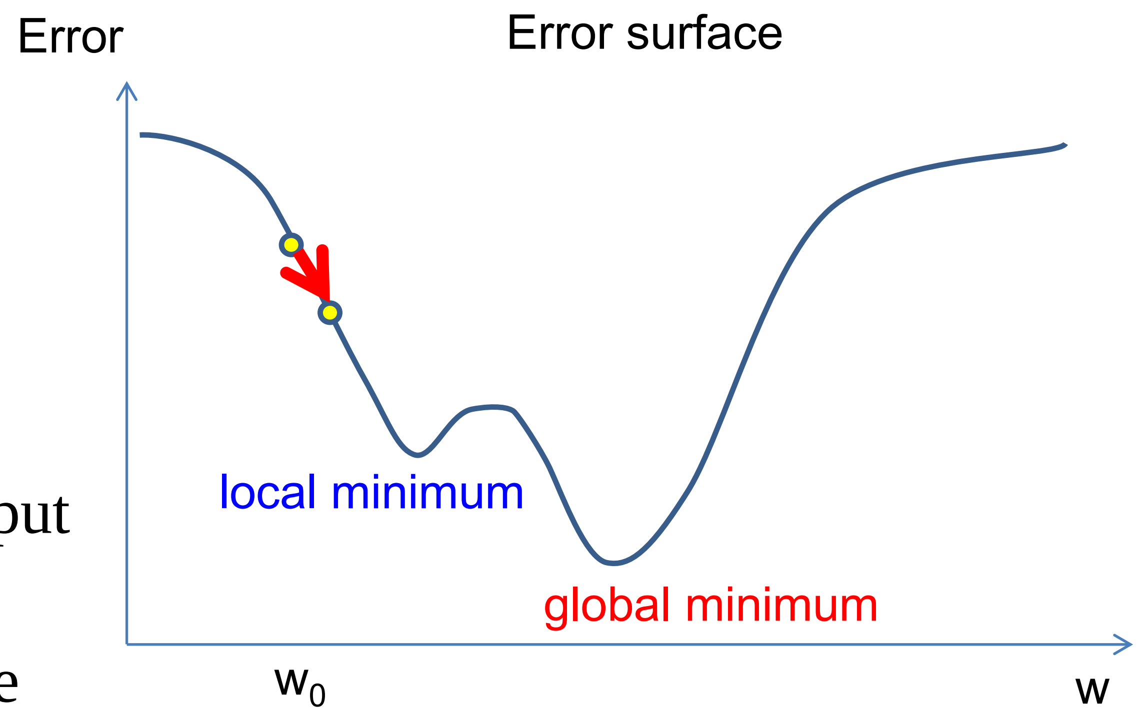
Mean squared error

$$Error = \frac{1}{2}\sum_i (t_i - f(x_i))^2$$

target output

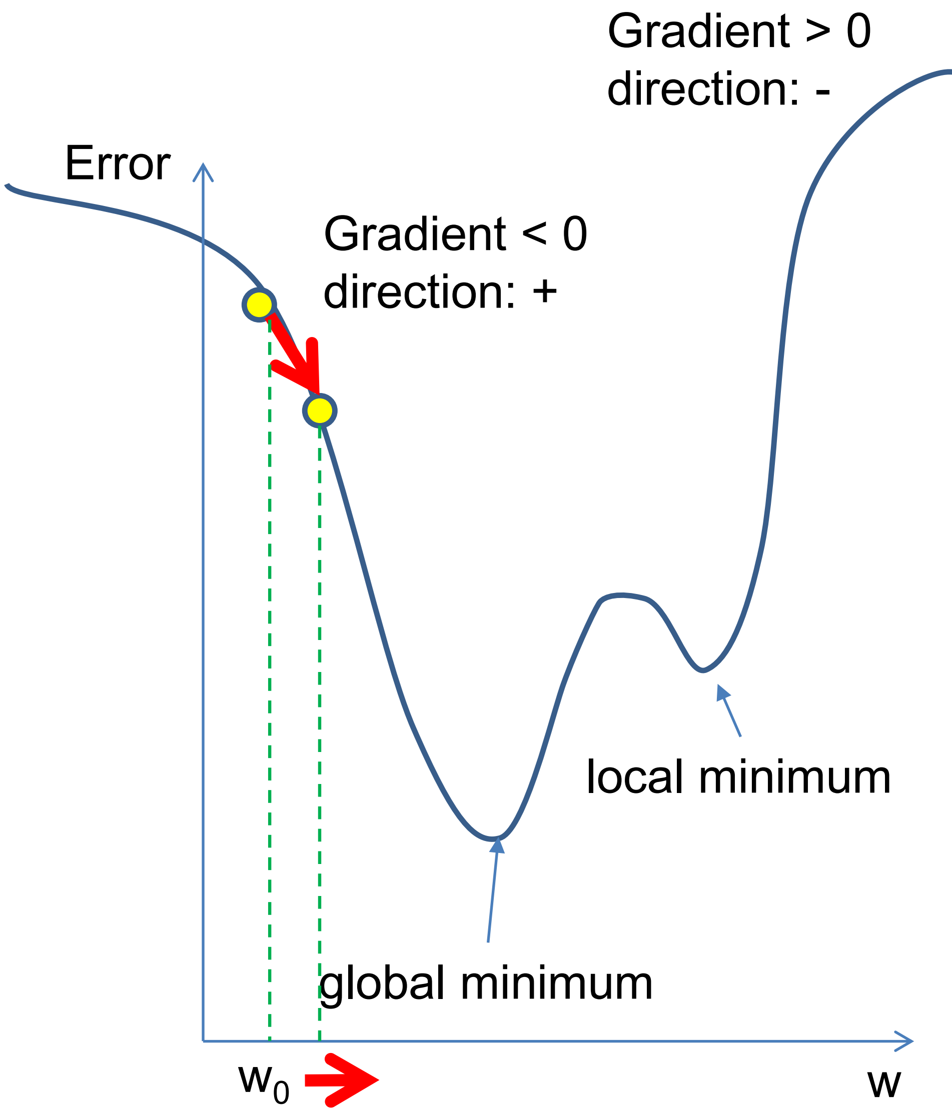
- t_i = the desired value
- $f(x_i) = O_i$ the actual output of the node i.
- Training: minimizing the errors

→ Objective functions

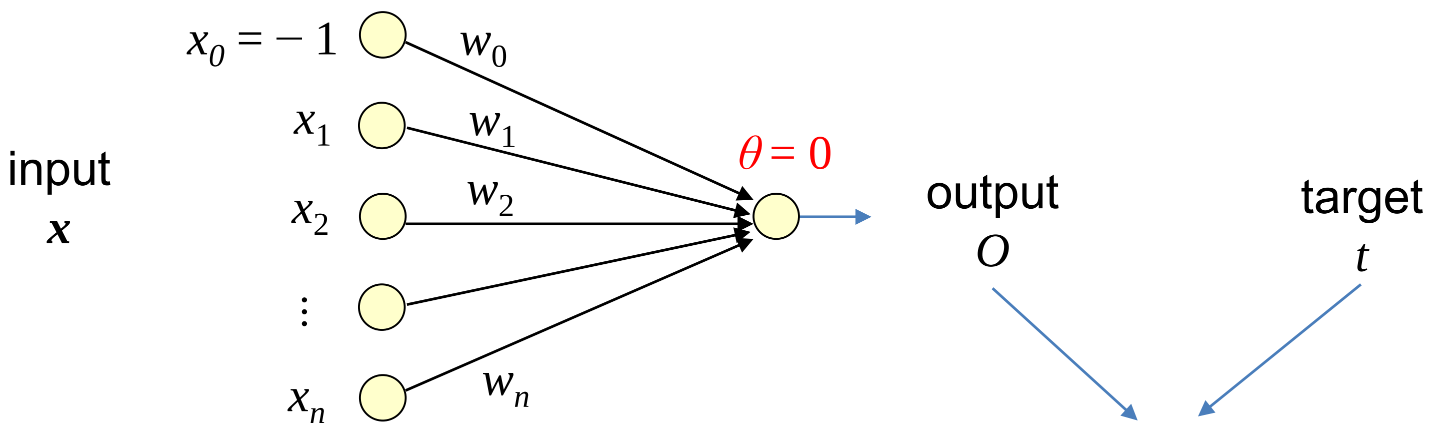


Gradient descent learning

- Training starts with **random weights**
- During training, the network should adapt its **weights** so that the overall error is reduced.
- The weights should be adjusted in the direction of steepest descent.
- Use the **derivative** of the error surface with respect to a weight



Gradient descent learning



$$Error(\boldsymbol{w}) = \frac{1}{2}(t_i - O_i)^2$$

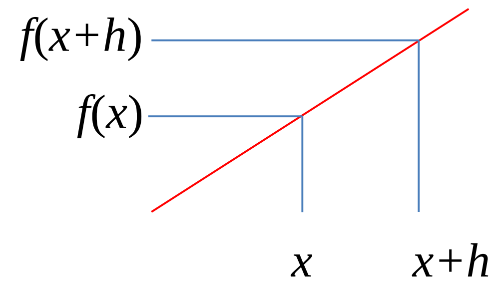
$$O_i = \sigma\left(\sum_{i=0}^{n} x_i w_i\right)$$

- Calculate $\dfrac{\partial Error}{\partial w_k}$

Calculating Gradients

- 미분 복습 …

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



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

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

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

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

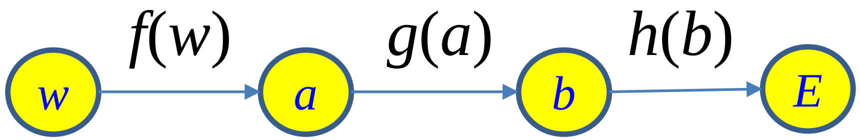
$f(g(x))\,' = ?$

$h(\,g(\,f(x)\,))\,' = ?$

$f_1(\,f_2(\,f_3(\,f_4(x))))\,' = ?$

$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2)}} \quad \rightarrow \quad ?$$

Calculating Gradients



computational graph

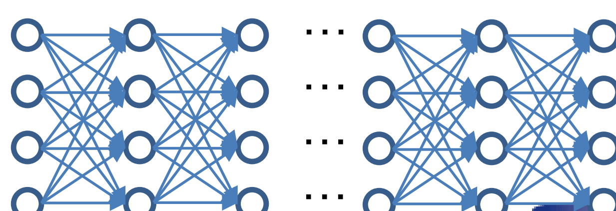
$$\frac{\partial E}{\partial w} = ?$$

$$a = f(w)$$
$$b = g(a)$$
$$E = h(b)$$

$$\Rightarrow \quad E = h(\, g(\, f(w)\,)\,)$$

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial b} \cdot \frac{\partial b}{\partial w} = \frac{\partial E}{\partial b} \cdot \frac{\partial b}{\partial a} \cdot \frac{\partial a}{\partial w}$$

chain rule



서울시립대학교
UNIVERSITY OF SEOUL

Back propagation

- http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Backpropagation: a simple example

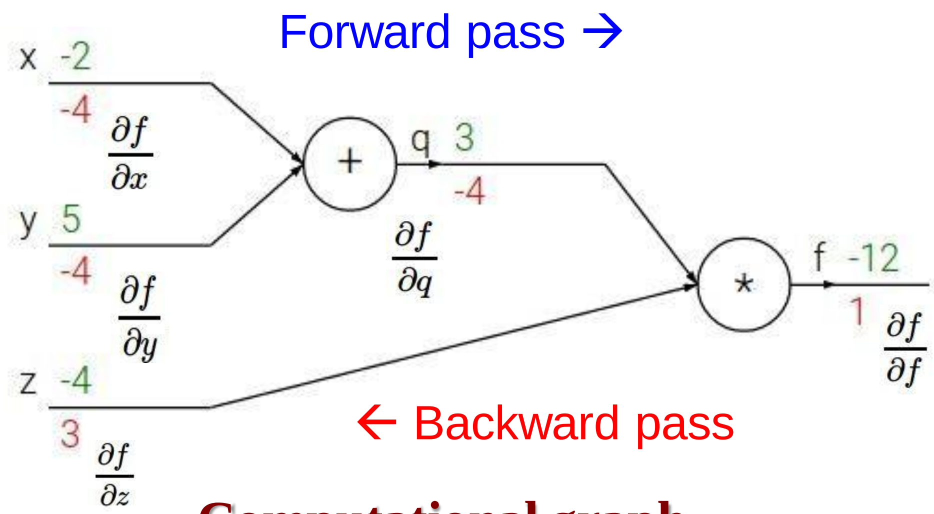
$$f(x, y, z) = (x + y)z$$

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

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

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

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



Computational graph

Chain rule:

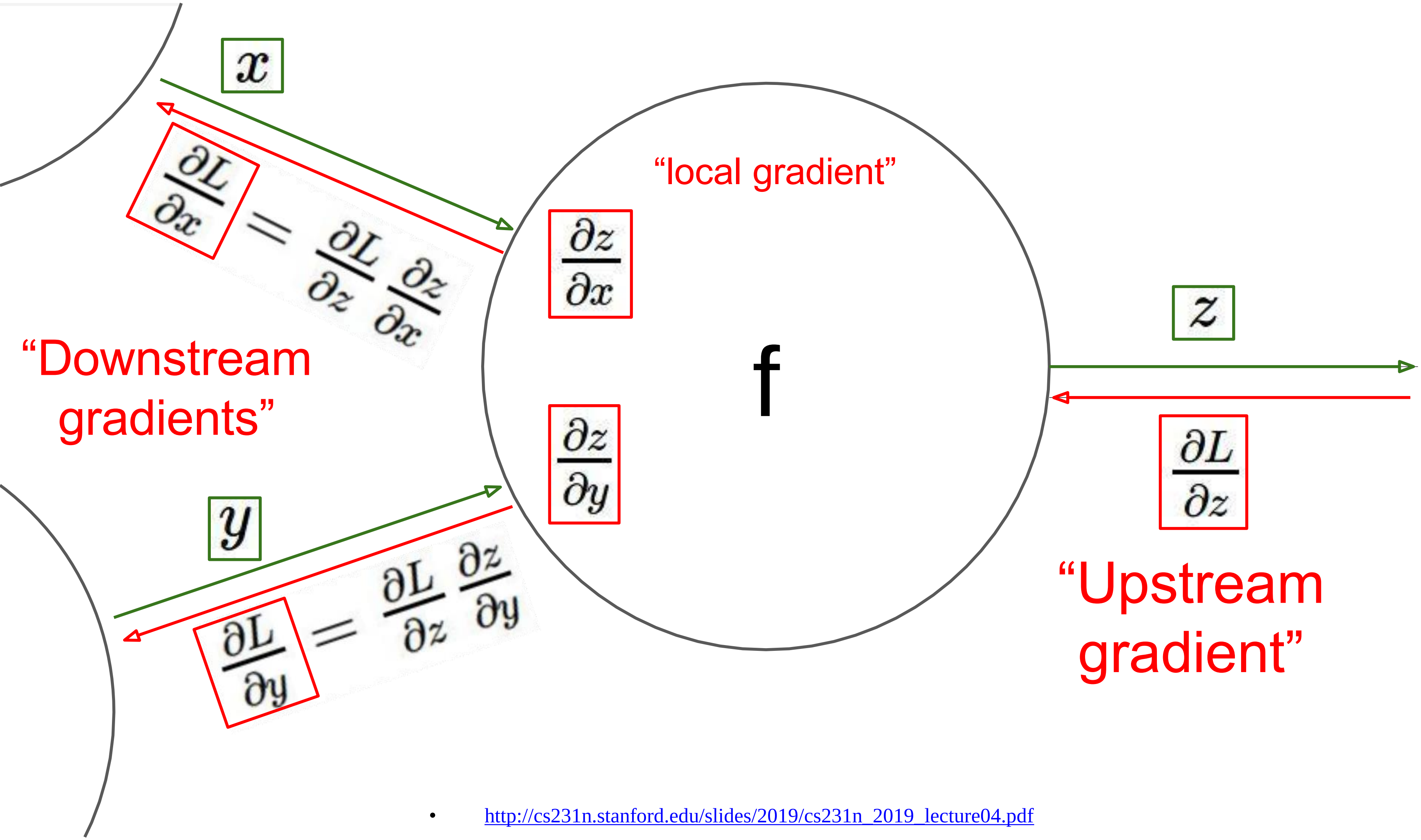
$$\boxed{\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}}$$

Upstream gradient

Local gradient

- http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

서울시립대학교
UNIVERSITY OF SEOUL

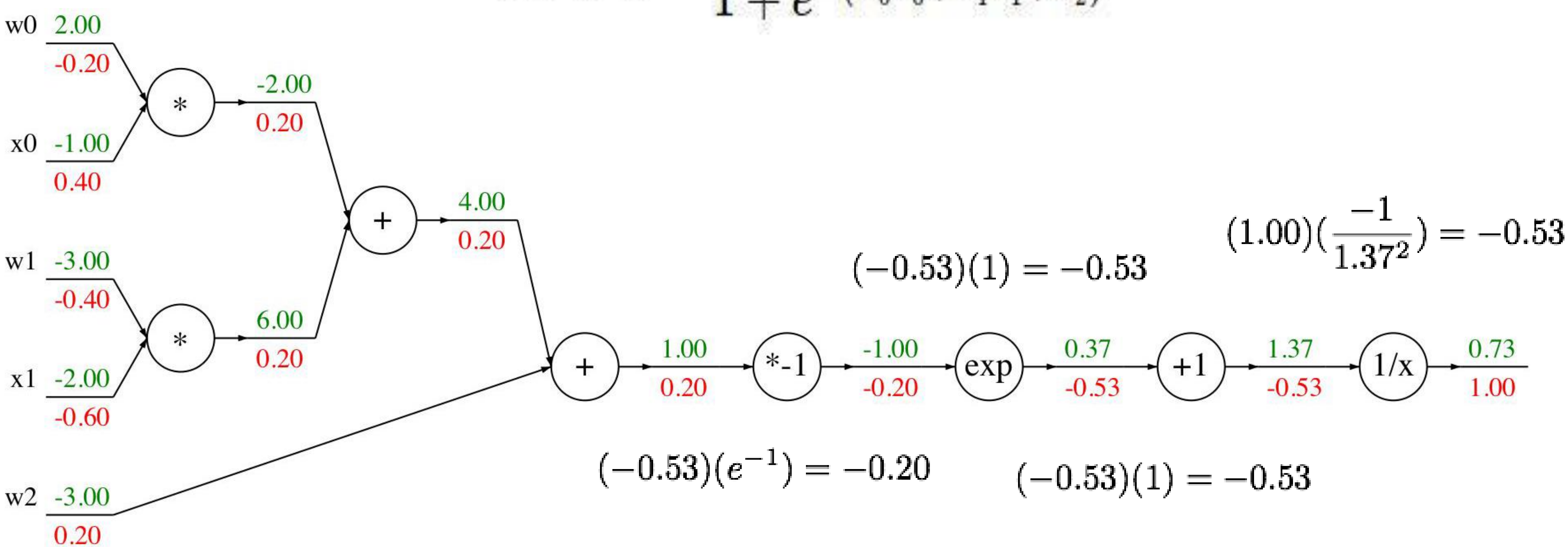


- http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

Perceptron example

x0: [0.2] x [2] = 0.4
w0: [0.2] x [-1] = -0.2

$$f(w,x) = \frac{1}{1+e^{-(w_0 x_0 + w_1 x_1 + w_2)}}$$



[0.2] x [1] = 0.2

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

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

- http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture04.pdf

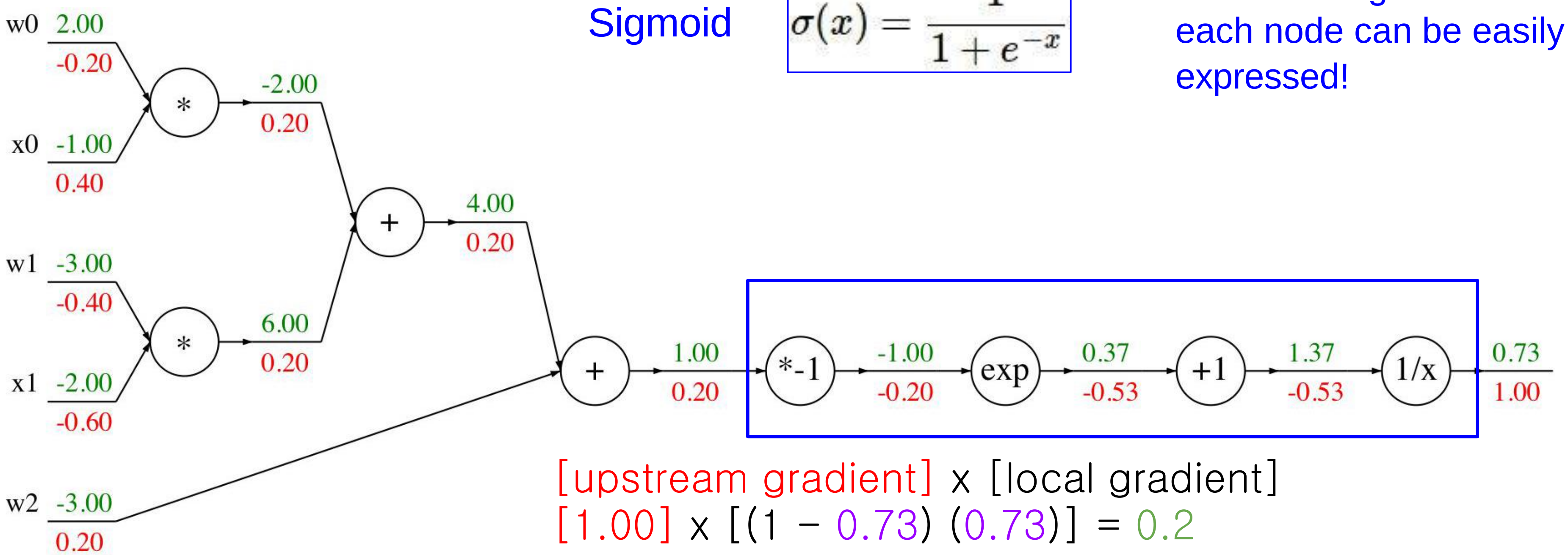
서울시립대학교 UNIVERSITY OF SEOUL

Sigmoid function

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

Computational graph representation may not be unique. Choose one where local gradients at each node can be easily expressed!

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



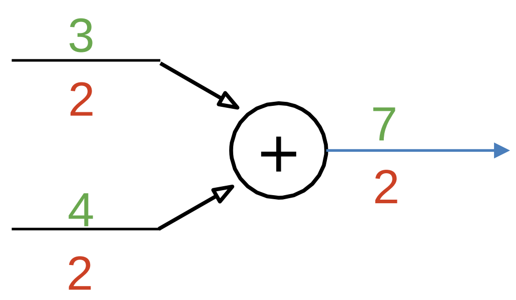
[upstream gradient] x [local gradient]
[1.00] x [(1 – 0.73) (0.73)] = 0.2

Sigmoid local gradient:

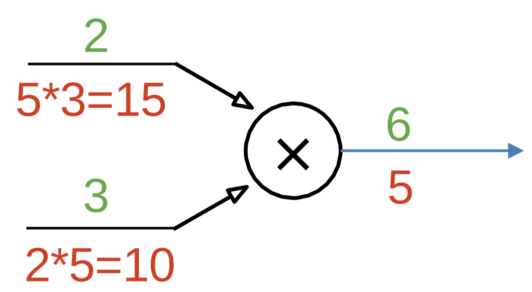
$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

Patterns in gradient flow

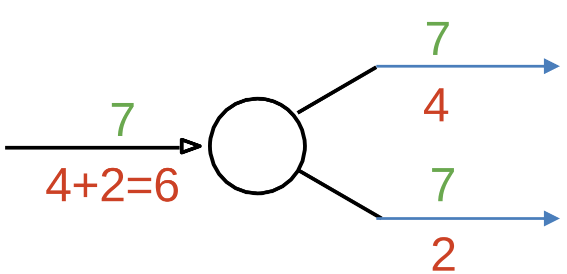
add gate: gradient distributor



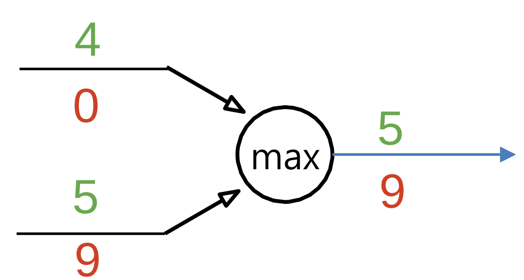
mul gate: “swap multiplier”



copy gate: gradient adder

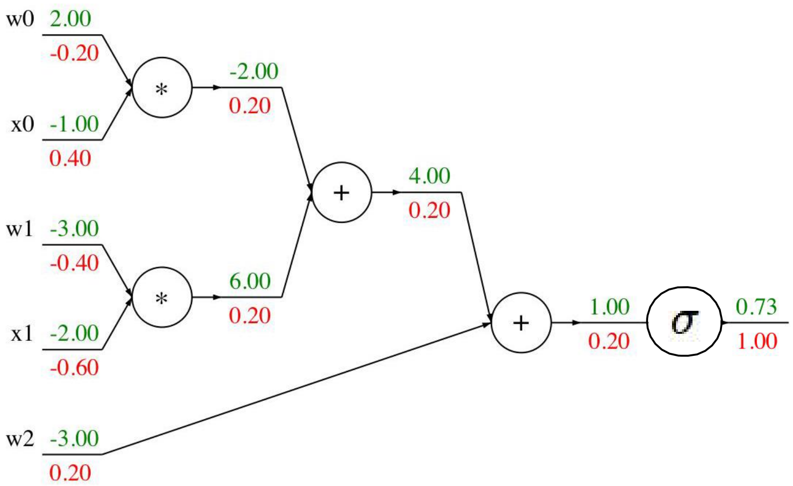


max gate: gradient router



서울시립대학교
UNIVERSITY OF SEOUL

Backprop Implementation: "Flat" code



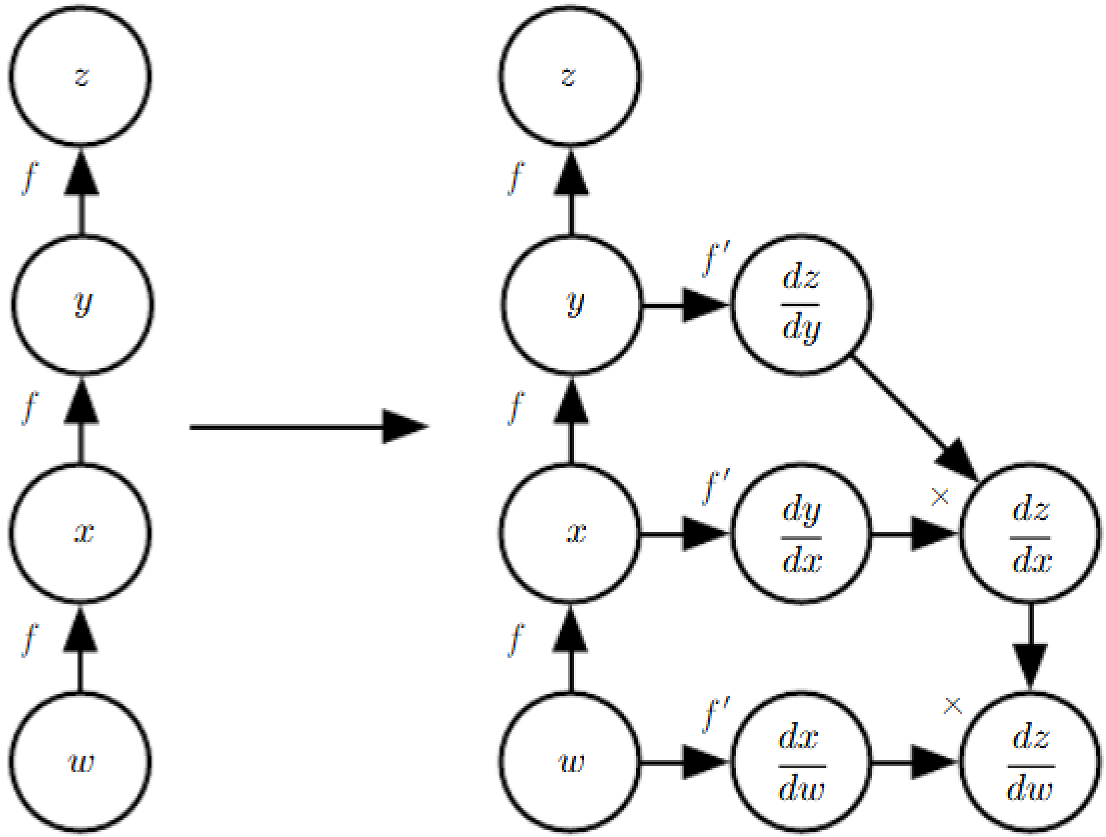
Forward pass:
Compute output

```
def f(w0, x0, w1, x1, w2):
    s0 = w0 * x0
    s1 = w1 * x1
    s2 = s0 + s1
    s3 = s2 + w2
    L = sigmoid(s3)
```

Backward pass:
Compute grads

```
    grad_L = 1.0
    grad_s3 = grad_L * (1 - L) * L
    grad_w2 = grad_s3
    grad_s2 = grad_s3
    grad_s0 = grad_s2
    grad_s1 = grad_s2
    grad_w1 = grad_s1 * x1
    grad_x1 = grad_s1 * w1
    grad_w0 = grad_s0 * x0
    grad_x0 = grad_s0 * w0
```

A computational graph with a symbolic description of the derivative



Ian Goodfellow, Yoshua Bengio and Aaron Courville, *Deep Learning*, MIT Press,2016

Calculating Gradients

$$E = \frac{1}{2}(t_i - O_i)^2$$

partial derivative →

$$\frac{\partial E}{\partial w_k} = \frac{\partial E}{\partial O_i}\frac{\partial O_i}{\partial w_k} = \frac{\partial E}{\partial O_i}\frac{\partial O_i}{\partial net_i}\frac{\partial net_i}{\partial w_k} = -(t_i - O_i)f'(net_i)x_k$$

chain rule

$$\frac{\partial E}{\partial O_i} = \frac{\partial \frac{1}{2}(t_i - O_i)^2}{\partial O_i} = -(t_i - O_i)$$

$$\frac{\partial O_i}{\partial net_i} = f'(net_i)$$

$$\frac{\partial net_i}{\partial w_k} = x_k$$

$$O_i = f(net_i)$$

$$net = w_0 x_0 + w_1 x_1 + \ldots + w_n x_n$$

Minimization of the error

- Weight change 는 gradient component 의 반대 방향 (–)

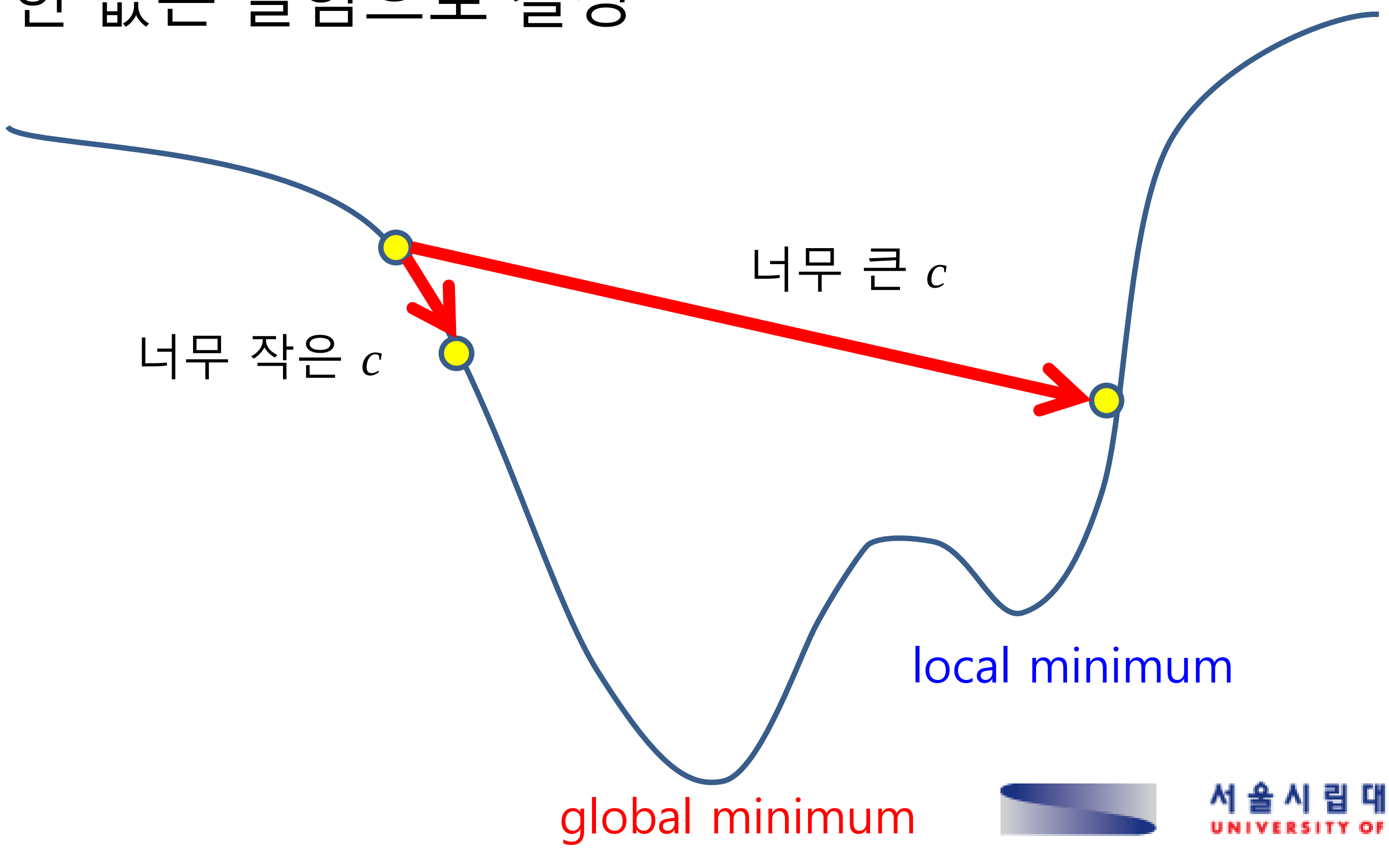
$$\Delta w_k = -c\frac{\partial E}{\partial w_k} = c(t_i - O_i)f'(net_i)x_k$$

$$\frac{\partial E}{\partial w_k} = -(t_i - O_i)f'(net_i)x_k$$

c = Learning rate, a constant

Learning rate, c

- 학습 속도에 큰 영향
- 큰 값 → optimal value 로 빨리 이동
  → 너무 크게 이동하면 진동
- 작은 값 → 너무 느려짐
- 적당한 값은 실험으로 결정



Perceptron training algorithm

Initialize weights with small random values

repeat

for each training data (x_i , t_i)

calculate output

$$net_j = \sum_{i=0}^{n} x_i w_i \,, \qquad O_i = f(net_i)$$

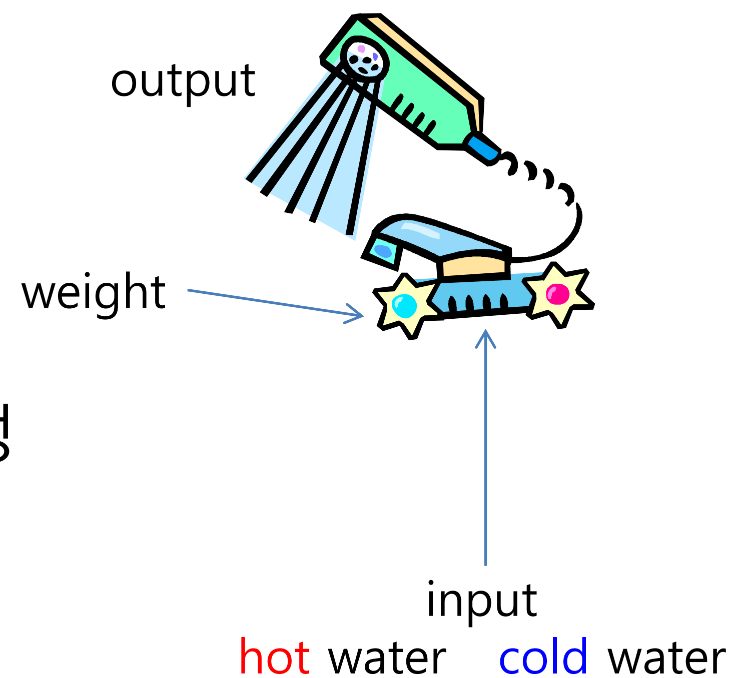
$$\Delta w_k = \mathrm{c}(t_i - O_i) f'\left(net_j\right) x_k$$

adjust weight $w_k \leftarrow w_k + \Delta w_k$

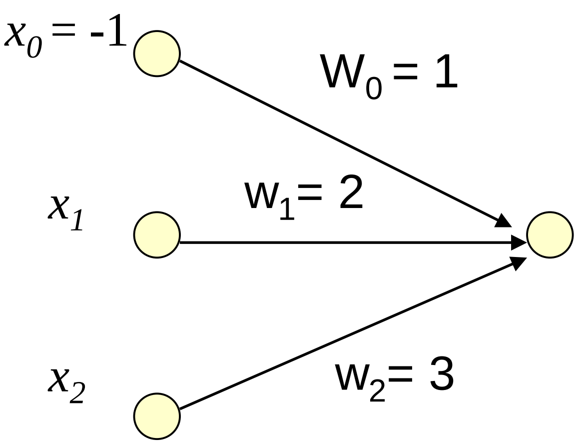
until satisfied

A learning rule – weights 수정

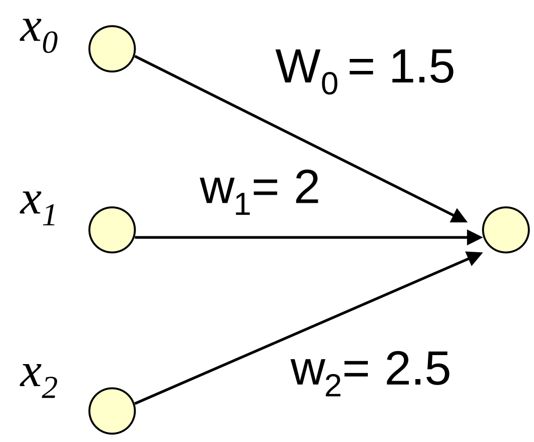
- Learning : 특정 input x 에 대하여 desired output y 를 얻도록 weights를 수정하는 것.
- Initial value : small random values
- Learning : error → adapt the weights
- x : input
- Activation (i.e. Output) = $f(x)$
- A target output (desired output) = t
- Error: $\delta = t - f(x)$, $f'(net)=1$ 이라고 가정
- The delta rule : $\Delta w = c\delta x$
- c = learning rate, a real number
- The new weight: $w_{new} = w + \Delta w$



Learning example (AND gate)



x1=0, x2=1
c = 0.5
$\delta = t - f(x) = 0 - 1 = -1$
$\Delta w0 = c\delta x = 0.5 \times -1 \times -1 = .5$
w0 = 1+0.5 = 1.5
$\Delta w2 = c\delta x = 0.5 \times -1 \times 1 = -.5$
w2 = 3-0.5 = 2.5

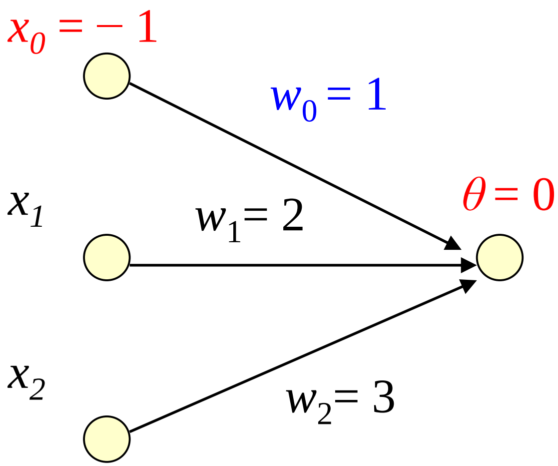


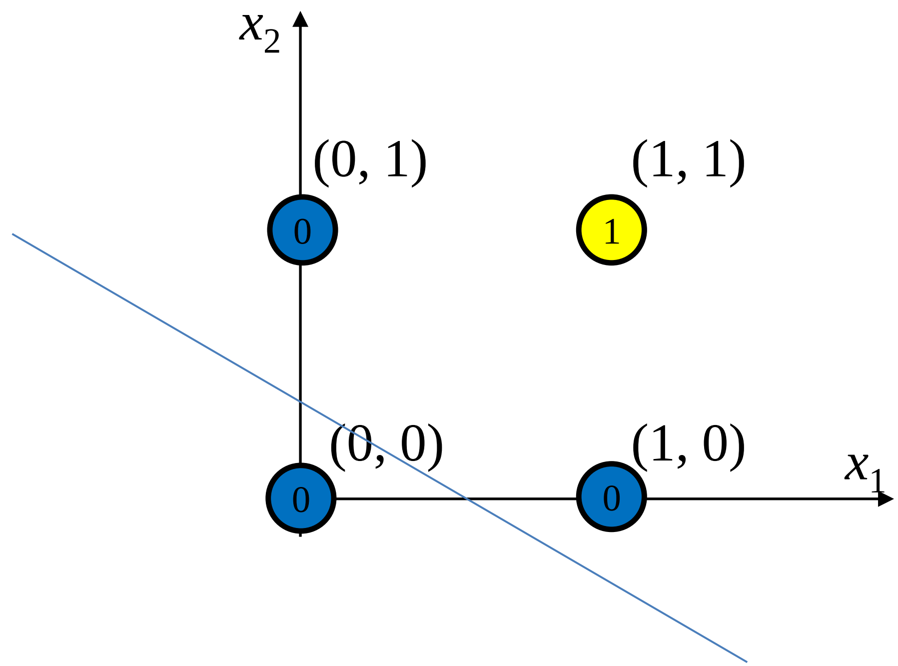
f(x)=1 if x>=0, 0 otherwise

input		initial		Iter = 1		Iter = 2	
x_1	x_2	in	out	in	out	in	out
0	0						
0	1						
1	0						
1	1						

Initial Values

- AND gate



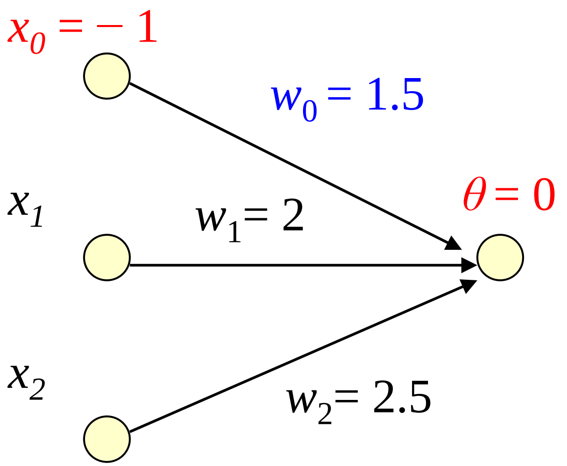


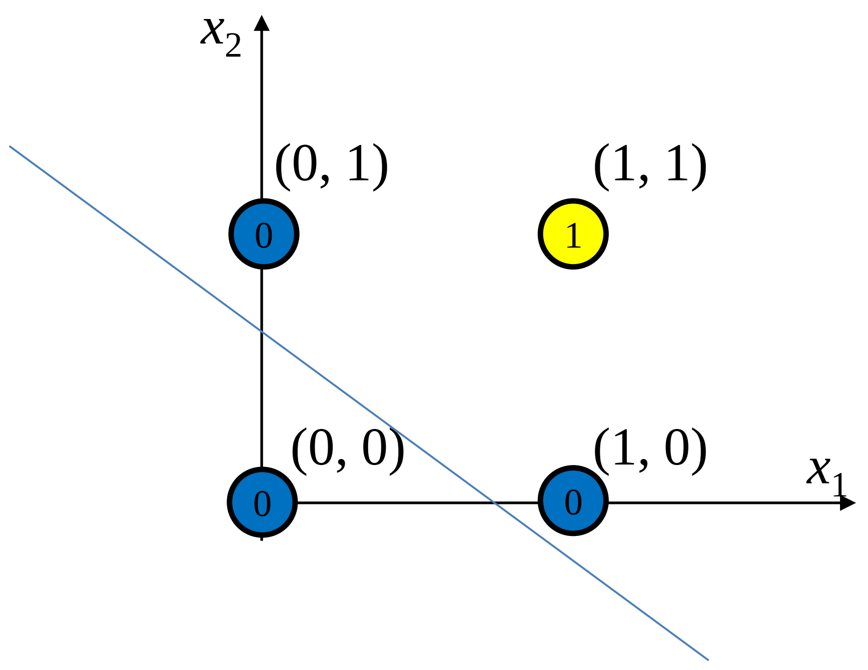
$$net = w_0 x_0 + w_1 x_1 + w_2 x_2$$
$$= -1 + 2x_1 + 3x_2 = 0$$

$$x_2 = -2/3\ x_1 + 1/3$$

Adapted values

- AND gate





$$net = w_0 x_0 + w_1 x_1 + w_2 x_2$$
$$= -1.5 + 2x_1 + 2.5x_2 = 0$$

$$x_2 = -2/2.5\, x_1 + 1.5/2.5$$
$$x_2 = -0.8\, x_1 + 0.6$$

Training example

ex) x = [1 1] → y = 0

- initial c= 0.1, x_0=1
- w_0=0.1 w_1=0.2 w_2=-0.1
- net = 0.1 + 0.2 – 0.1 = 0.2
- output = 1 (if net > 0 output = 1, else output = 0)
- delta = 0-1 = -1
- Δw0 = 0.1 × -1 × 1 = -0.1   w0=0
- Δw1 = 0.1 × -1 × 1 = -0.1   w1=0.1
- Δw2 = 0.1 × -1 × 1 = -0.1   w2= - 0.2
- net = 0 + 0.1 – 0.2 = -0.1   output = 0

Training 의 원리

- Training 시작: **random weights**
- Training 과정: weights 수정
- Weights 는 기울기의 반대 방향으로 수정 → **steepest descent**.

Delta rule

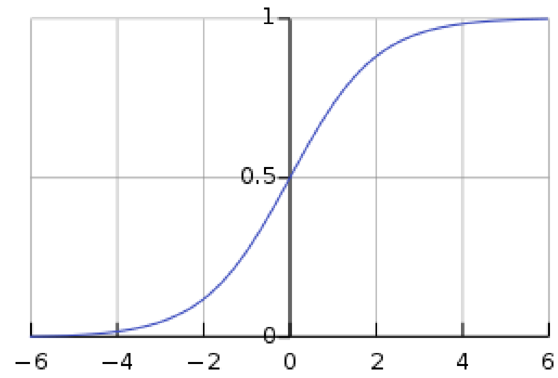
- 미분 가능한 activation function 필요 → logistic formula
- The delta rule learning formula for weight adjustment

$$\Delta w_k = c(t_i - f(net_j))f'(net_j)x_k$$

- c = learning rate
- t_i = desired output
- $f(x)$ = actual output values of the ith node.
- f' = The derivative
- x_k = the kth input to node i.

The derivative of the sigmoid activation function

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



$$f'(net_j) = \frac{\exp(-net_j)}{(1+\exp(-net_j))^2}$$

$$= \frac{1}{1+\exp(-net_j)}\left(1 - \frac{1}{1+\exp(-net_j)}\right)$$

$$= f(net_j)[1 - f(net_j)]$$

$$\Delta w_k = \mathrm{c}(t_i - f\left(net_j\right))f(net_i)(1 - f(net_i))x_k$$