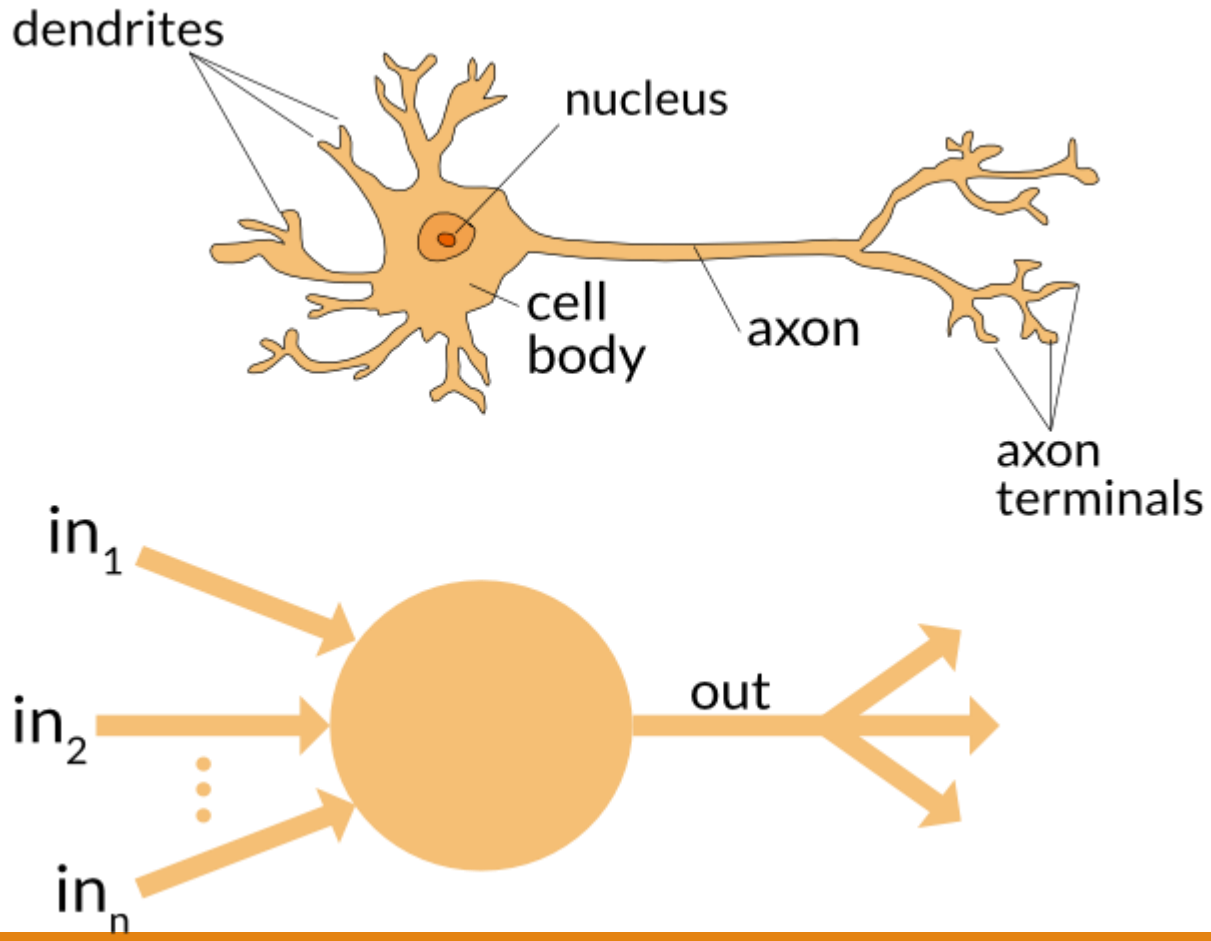
Multi Layer Perceptron

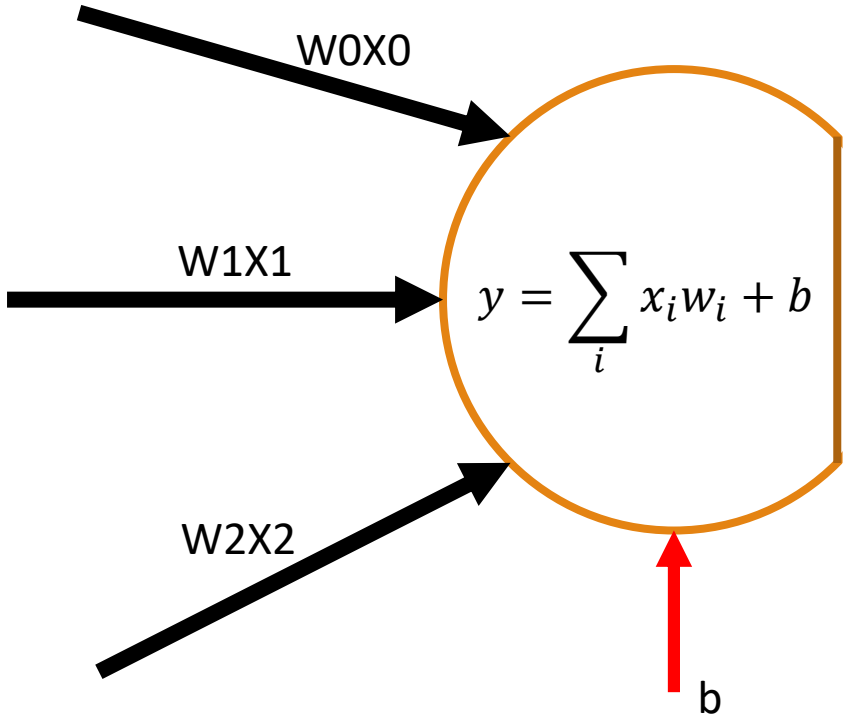
목차

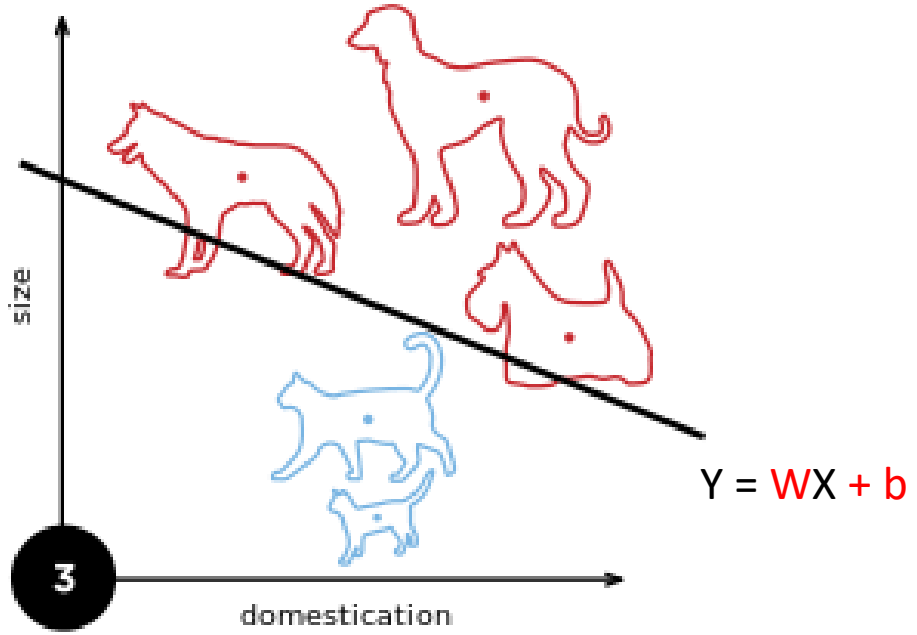
퍼셉트론

뇌의 뉴런을 모방



단일 퍼셉트론

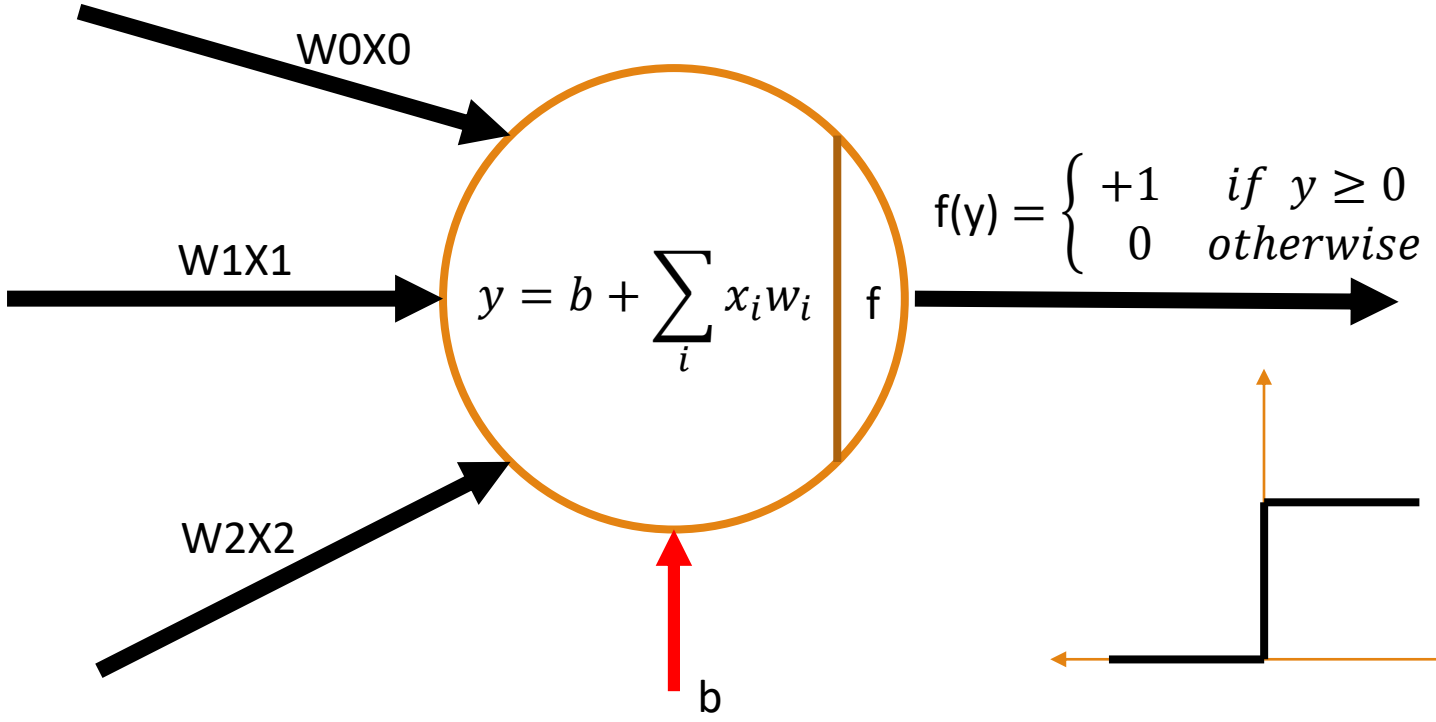




Activation function

뉴런에서 계산된 값이 임계치보다 크면 1을 출력 작은 경우 0을 출력.

다음으로 신호를 보낼지 말지를 결정.



Activation function : step function

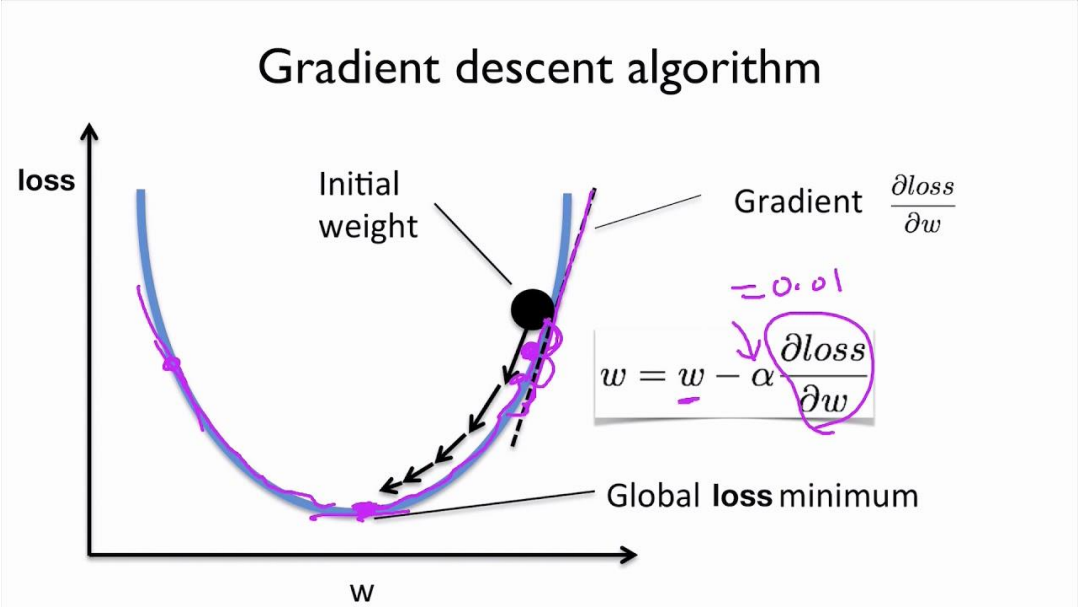
학습 알고리즘 – Delta Rule

출력 값이 우리가 원하는 목표 값(d)과 가까워지도록 가중치를 Gradient Descent(경사하강법)를 이용하여 조정.

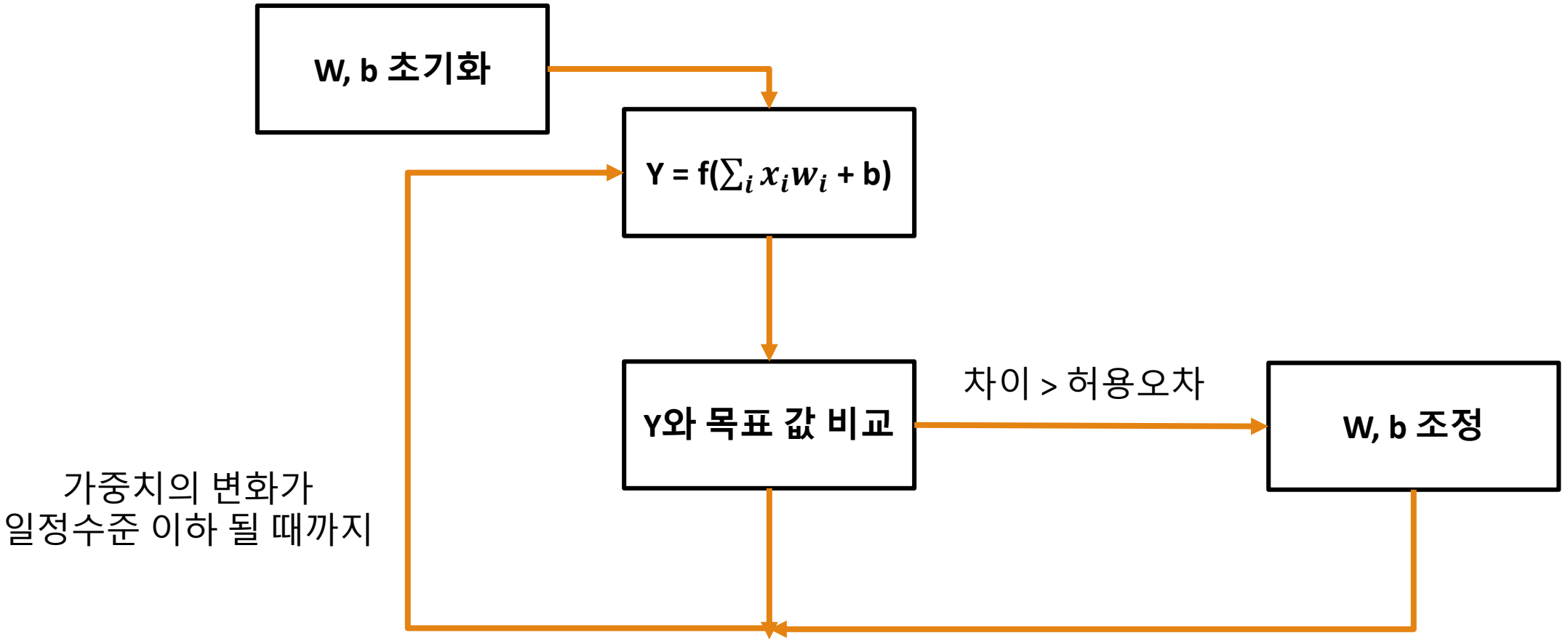
W(가중치)와 b(바이어스)를 임의의 값으로 초기화

하나의 학습 벡터에 대한 출력 값 계산

목표 값과 비교해서 허용 오차보다 크면 학습을 진행



학습 알고리즘



W, b 조정

출력 값과 목표 값의 차이가 허용 오차 보다 작을 때 W(가중치)와 b(bias) 조정

$$W_i(t+1) = W_i(t) + \alpha[d(t) - y(t)] \cdot X_i(t)$$

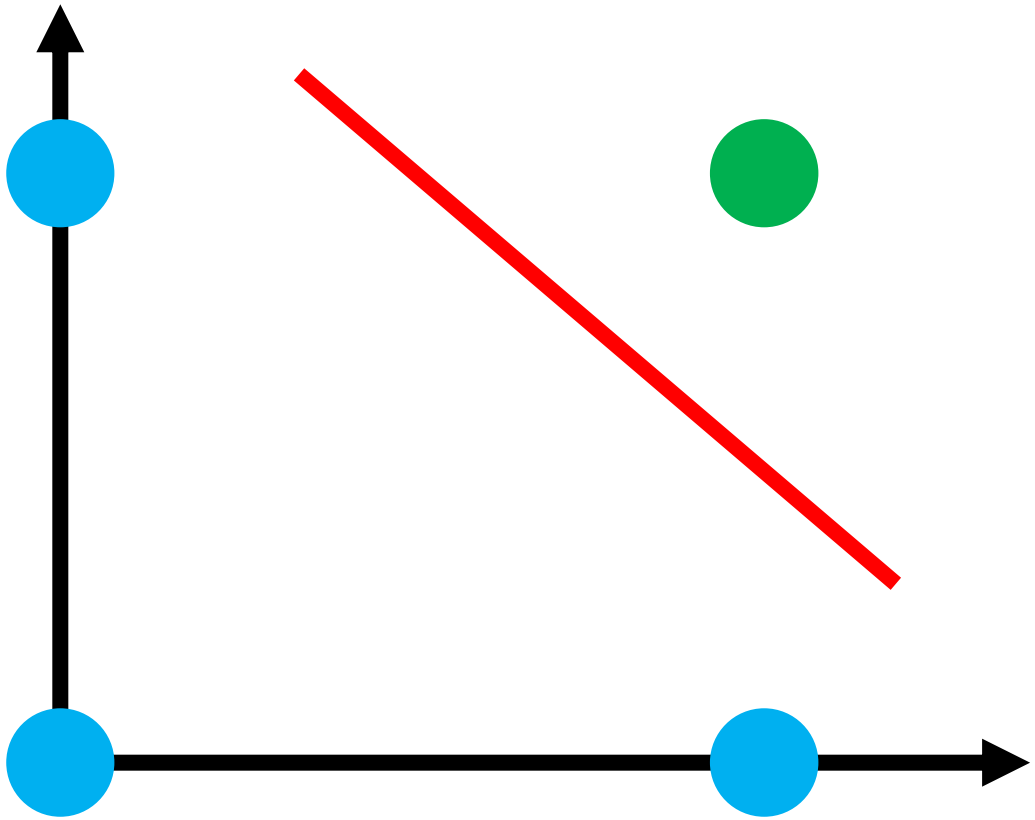
$$\mathrm{b} = b + \alpha[d(t) - y(t)] \cdot X_i(t)$$

α = learning rate

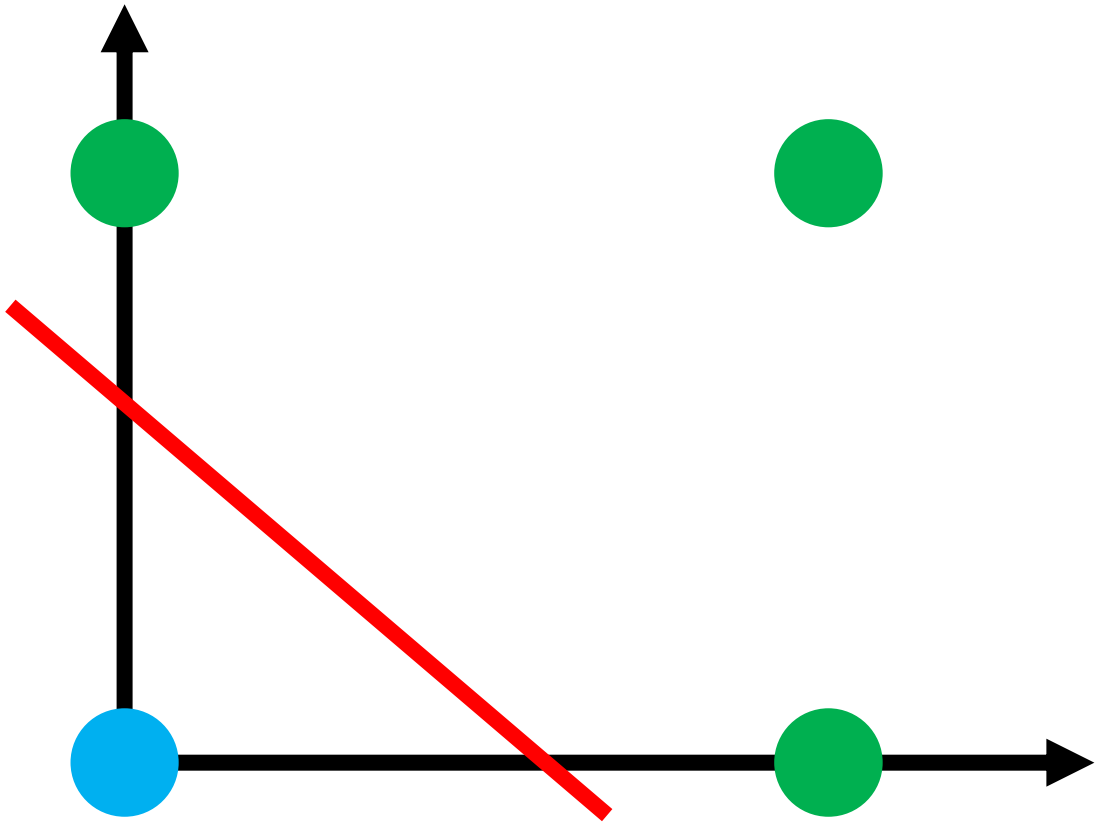
$d(t)$ = 목표 값

y(t) = 결과 값

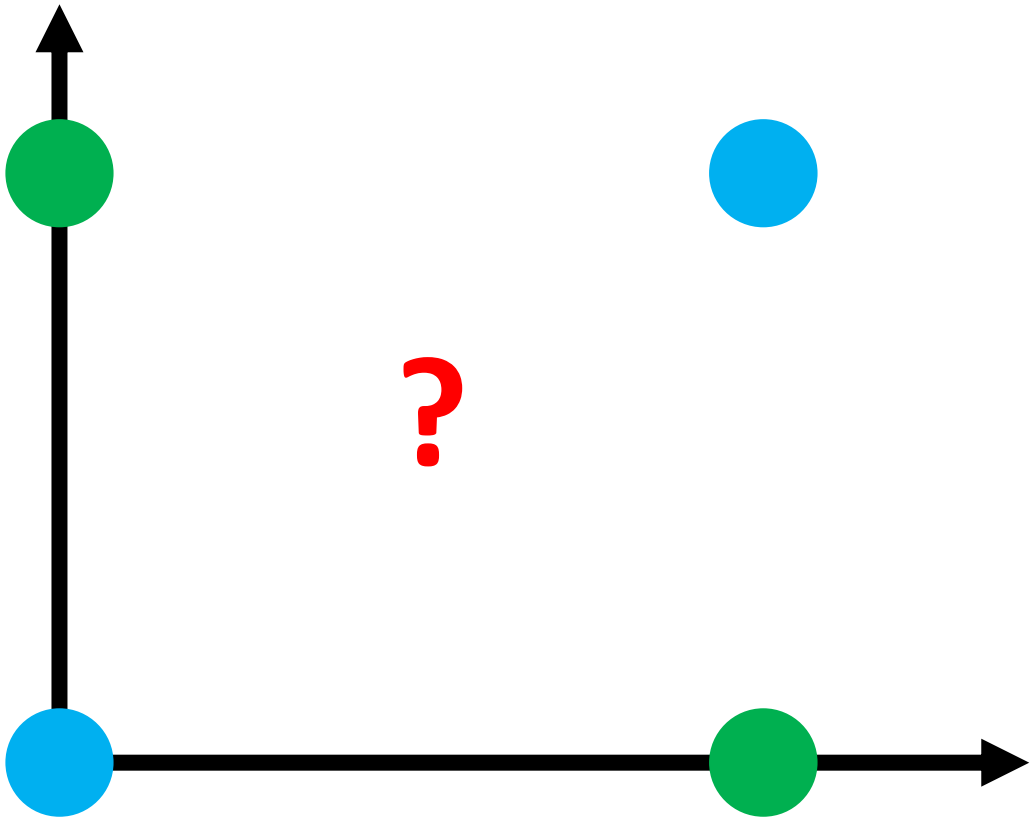
AND



OR



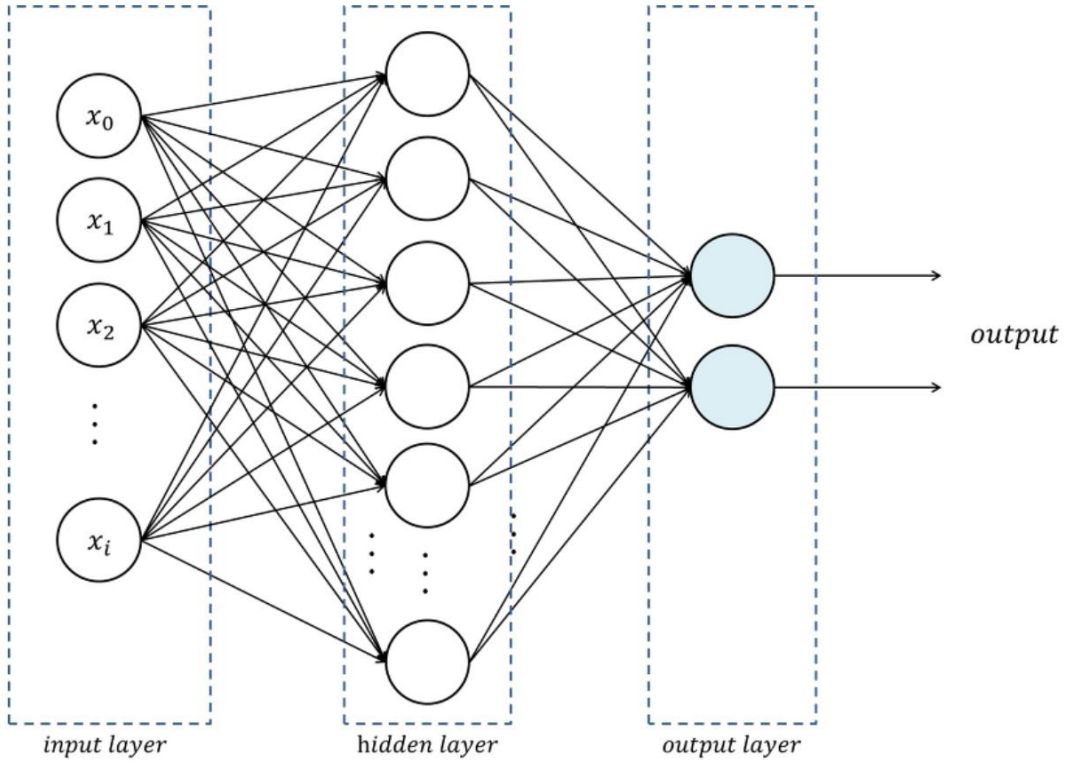
XOR?



다층 퍼셉트론

입력층과 출력층 사이에 1개 이상의 은닉층(hidden layer)을 추가

대부분 역전파 알고리즘을 이용하여 학습



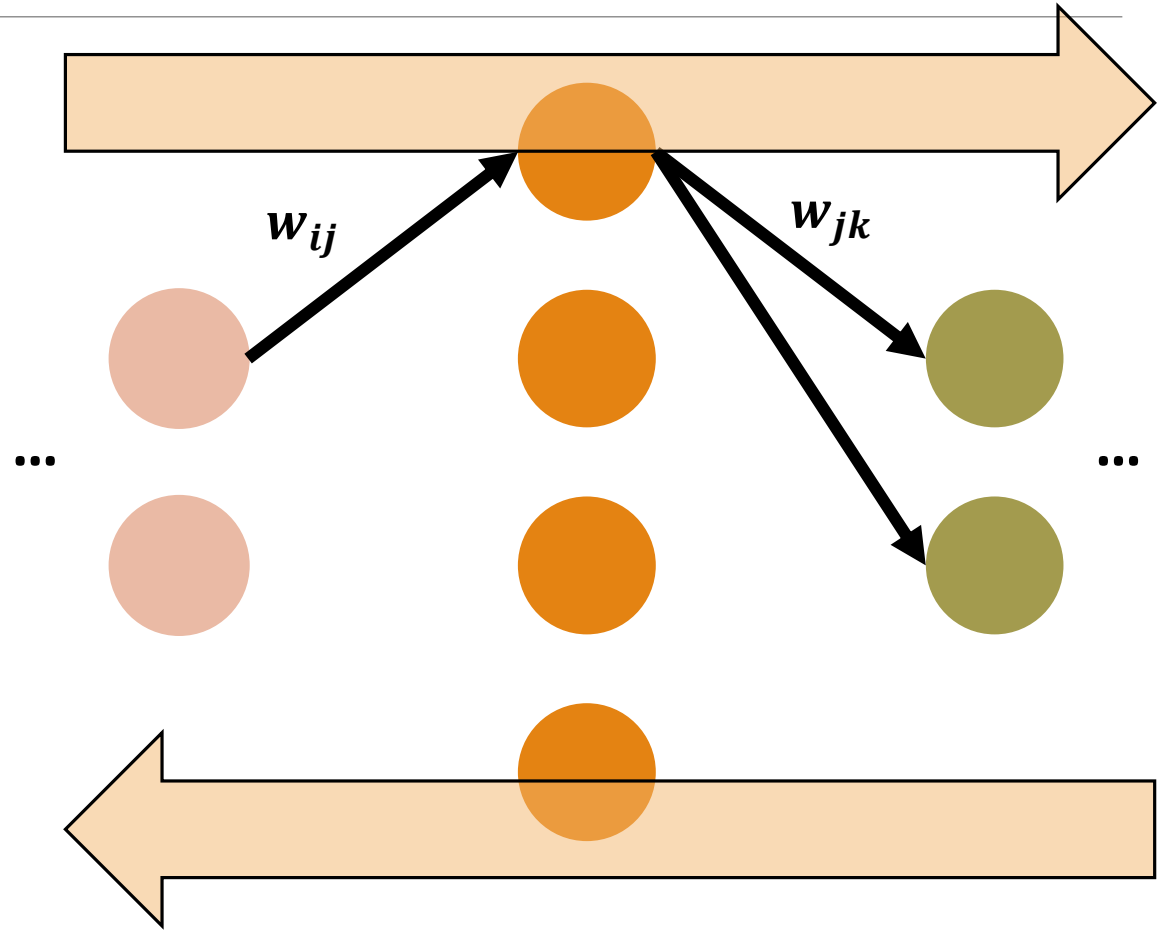
학습 알고리즘

1. 초기화 (initialization)

while(오차의 합계 < 허용 오차):

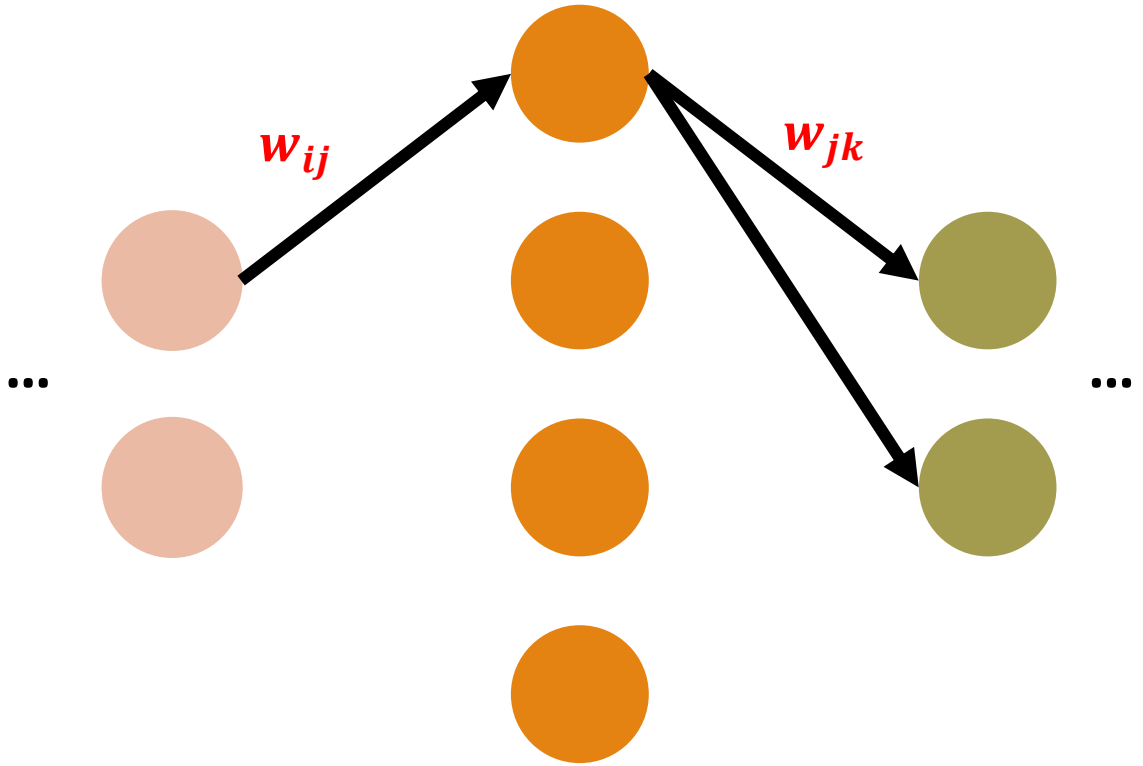
  2. 순전파 (feedforward)

  3. 역전파 (backpropagation)



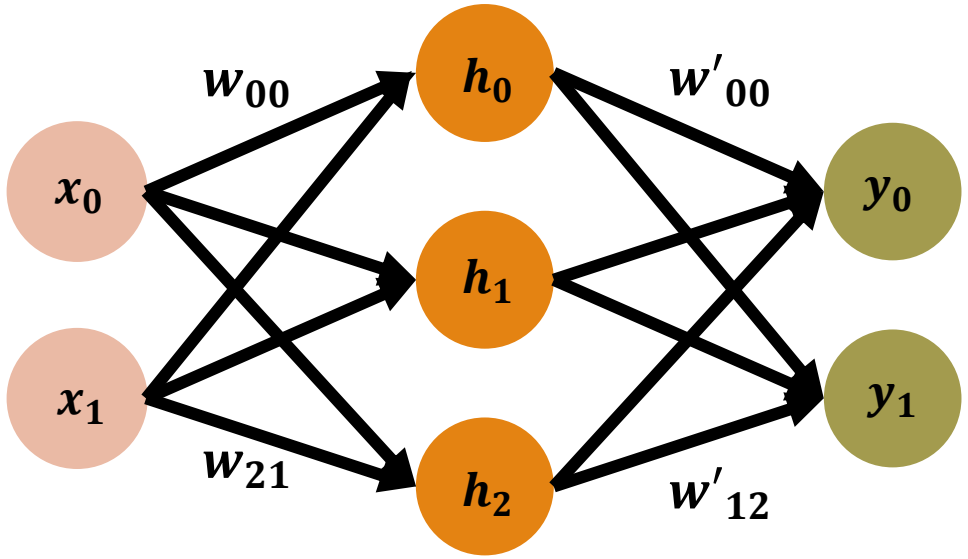
초기화 (initialization)

1. 신경망 구조 선택
   - 은닉층 (hidden layer) 개수
   - 은닉 뉴런 (hidden neuron)개수
2. 가중치, 임계 값 초기화
   - 임의의 값
   - 초기 값으로 학습 시간 결정



순전파 (feedforward)

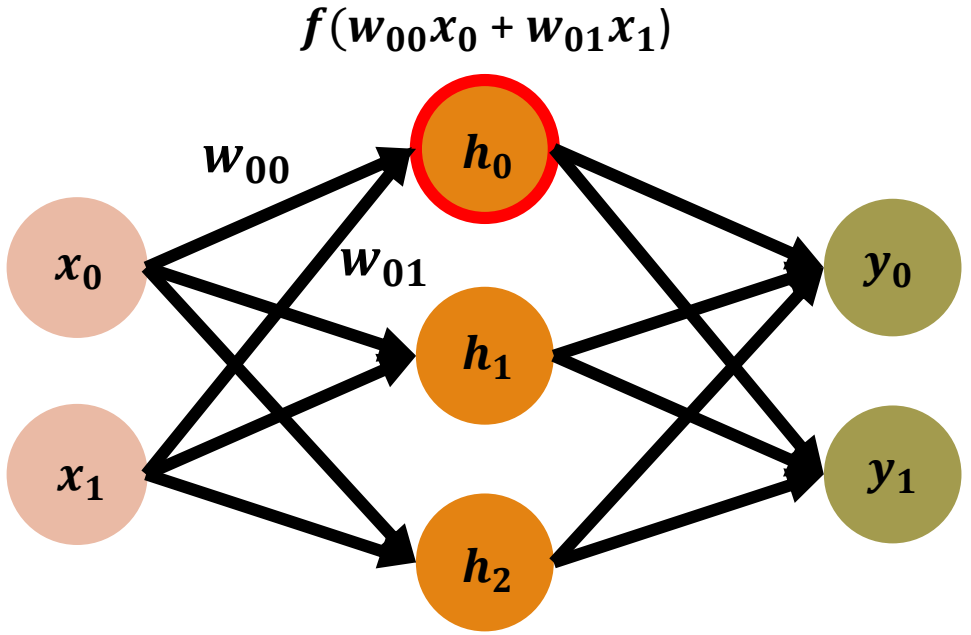
$$\begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \qquad \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} \qquad \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$



https://www.youtube.com/watch?v=iJ6Kj4XZBzA

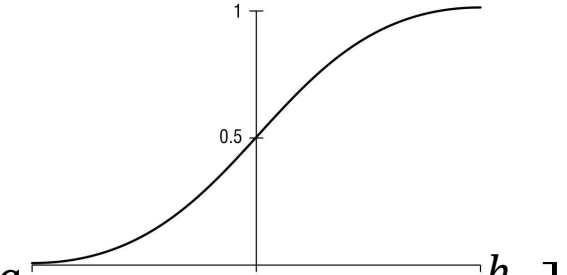
순전파 (feedforward)

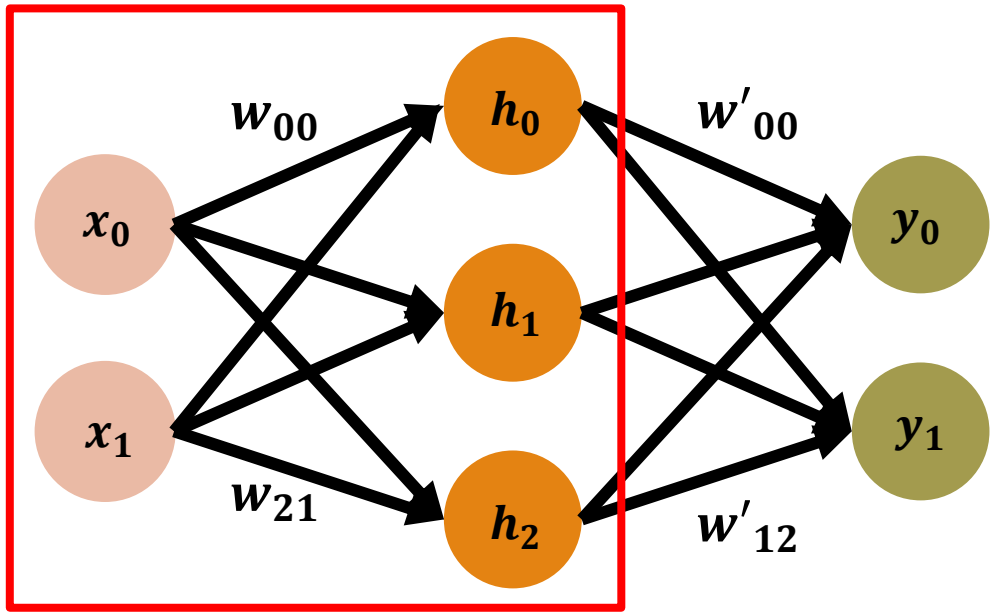
$$\begin{bmatrix} w_{00} & w_{01} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix}$$



순전파 (feedforward)

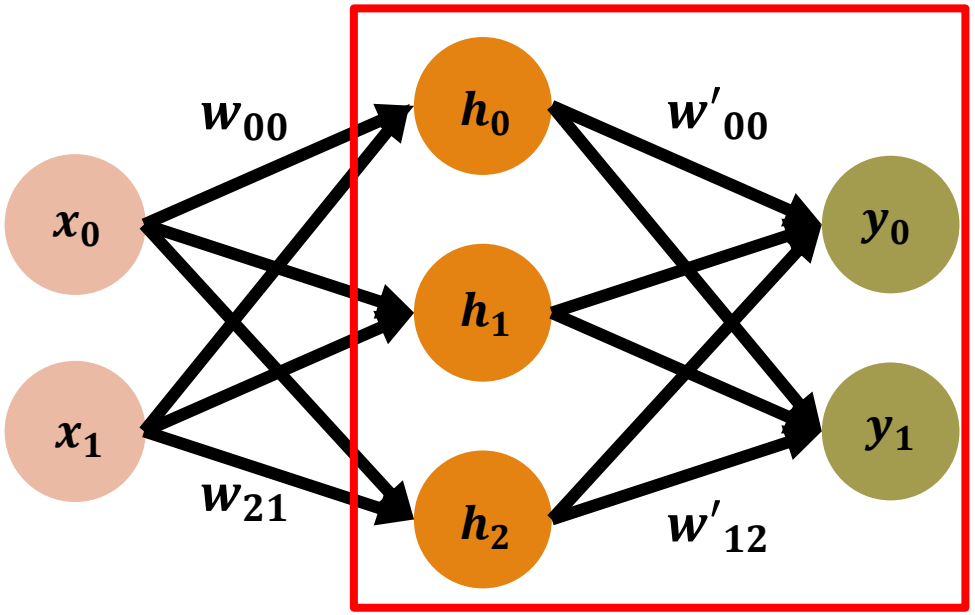
$$\begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \text{activation function (sigmoid)} \Rightarrow \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix}$$





순전파 (feedforward)

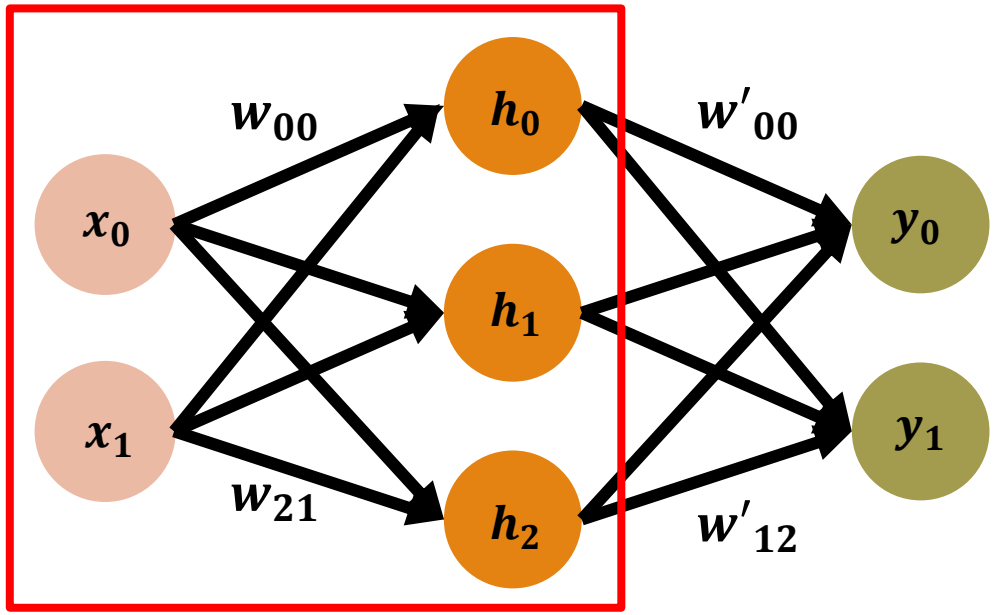
$$\begin{bmatrix} w'_{00} & w'_{01} & w'_{02} \\ w'_{10} & w'_{11} & w'_{12} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} a' \\ b' \end{bmatrix} \Rightarrow \text{activation function (sigmoid)} \Rightarrow \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$



순전파 (feedforward)

$$\begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \text{activation function (sigmoid)} \Rightarrow \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix}$$

W **X** **f()** **H**

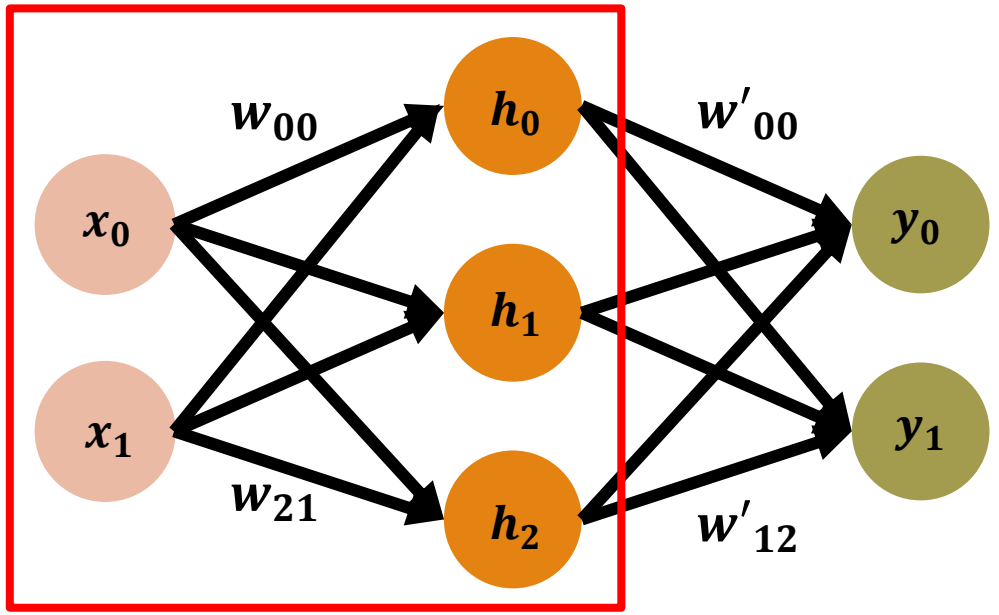


순전파 (feedforward)

$$\begin{bmatrix} w_{00} & w_{01} \\ w_{10} & w_{11} \\ w_{20} & w_{21} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix} \Rightarrow \text{activation function (sigmoid)} \Rightarrow \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix}$$

W **X** **f()** **H**

H = f(WX)

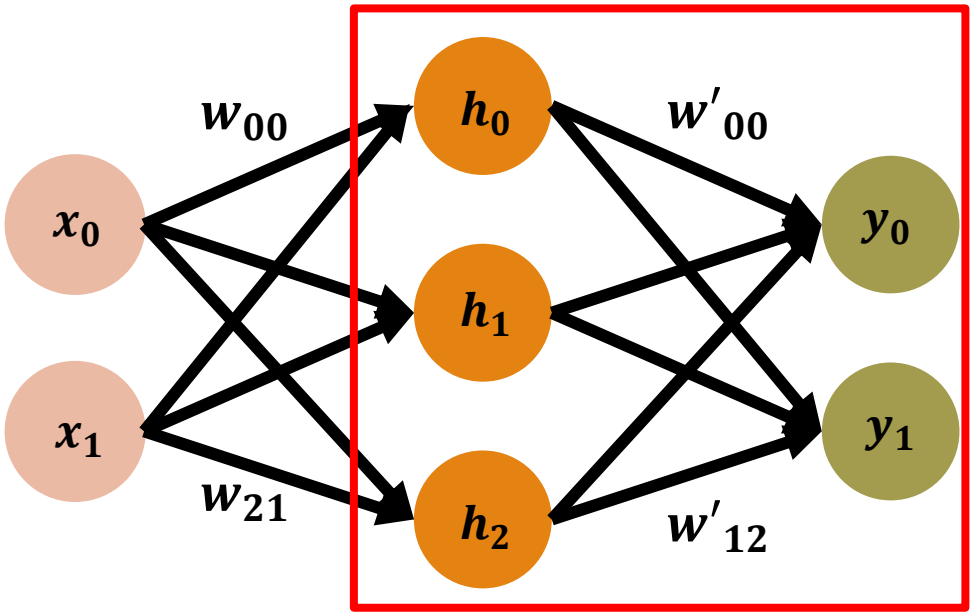


순전파 (feedforward)

$$\begin{bmatrix} w'_{00} & w'_{01} & w'_{02} \\ w'_{10} & w'_{11} & w'_{12} \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} a' \\ b' \end{bmatrix} \Rightarrow \text{activation function (sigmoid)} \Rightarrow \begin{bmatrix} y_0 \\ y_1 \end{bmatrix}$$

W’ H f() Y

Y = f(W’H)



역전파 (backpropagation)

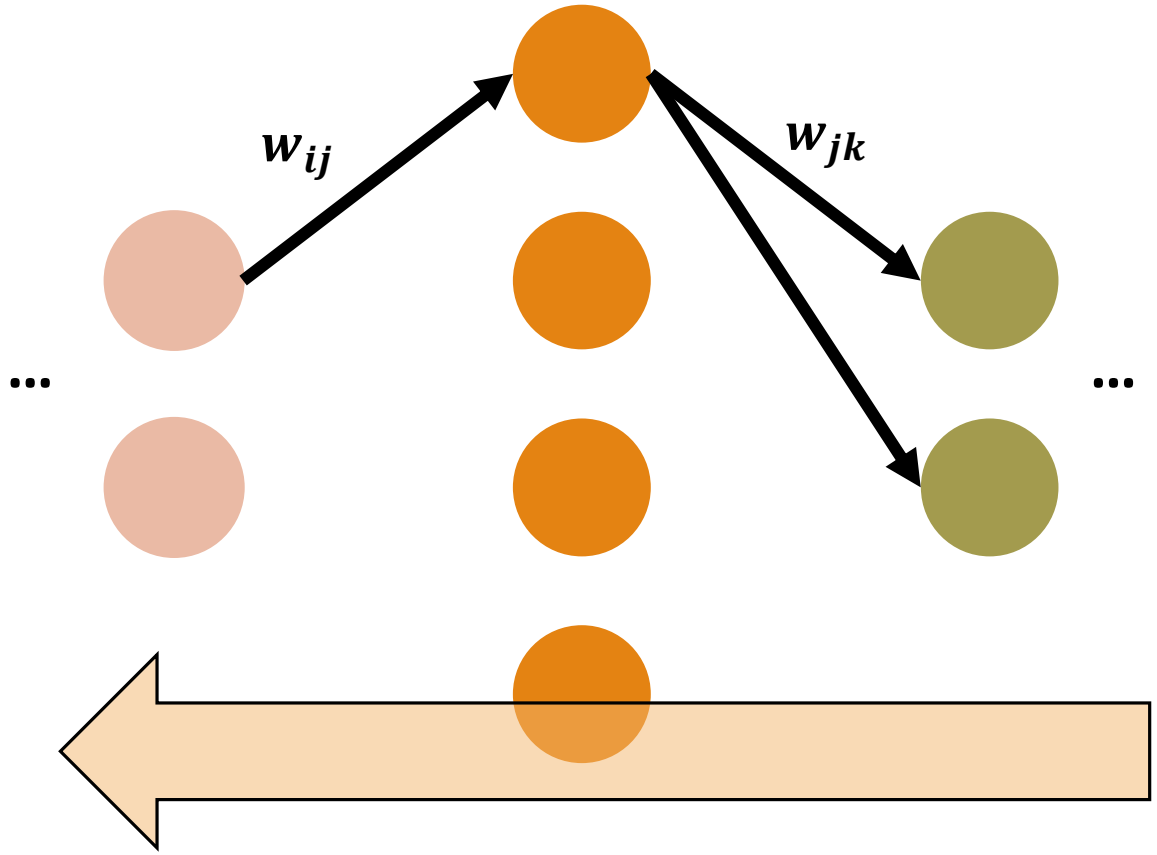
목표 : error을 줄이는 것 (loss function)

각 layer간의 weight에 대한 loss function의 gradient 를 weight 반대 방향으로 더해준다.

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

η : *learning rate*

$\frac{\partial E}{\partial w}$: *w의 변화에 따른 E의 변화*



역전파 (backpropagation)

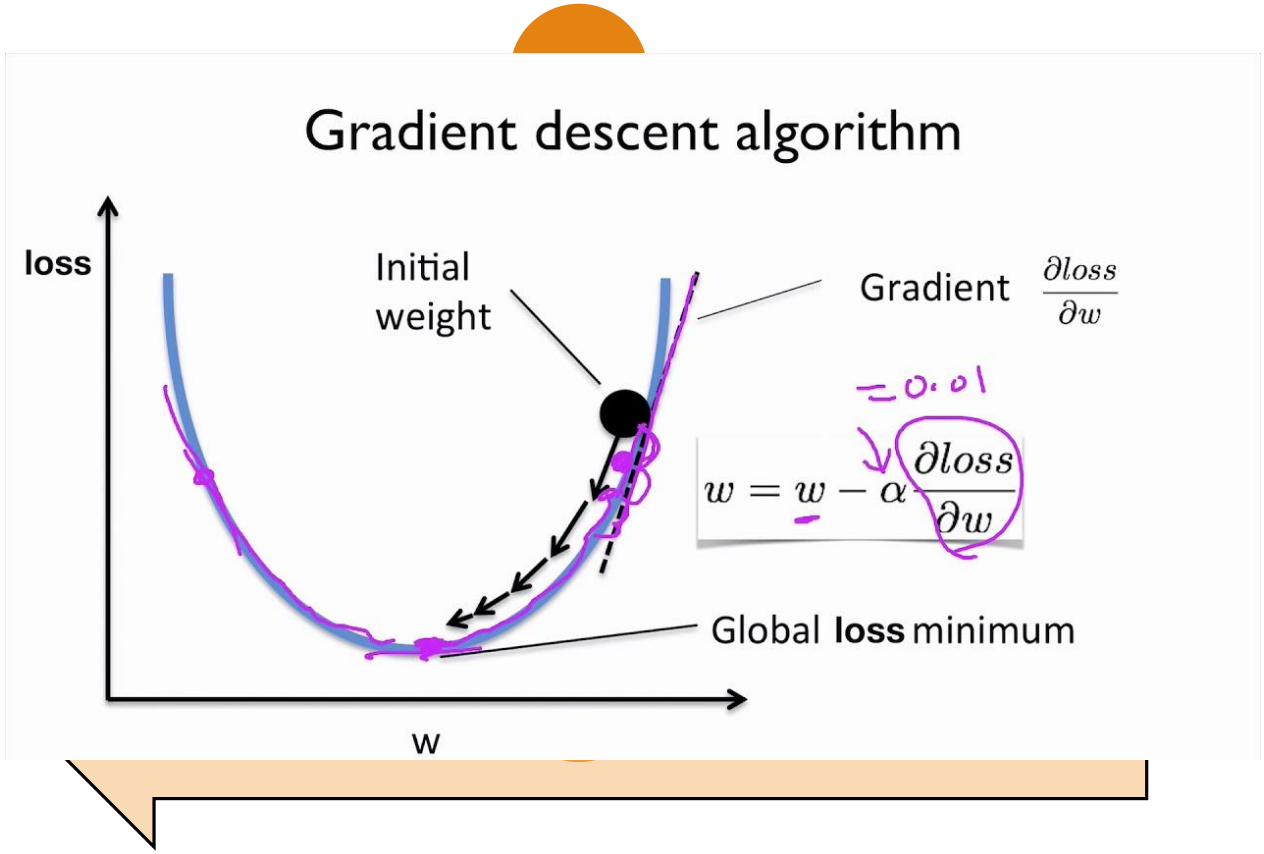
목표 : error을 줄이는 것 (loss function)

각 layer간의 weight에 대한 loss function의 gradient 를 weight 반대 방향으로 더해준다.

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

η : *learning rate*

$\frac{\partial E}{\partial w}$: *w의 변화에 따른 E의 변화*

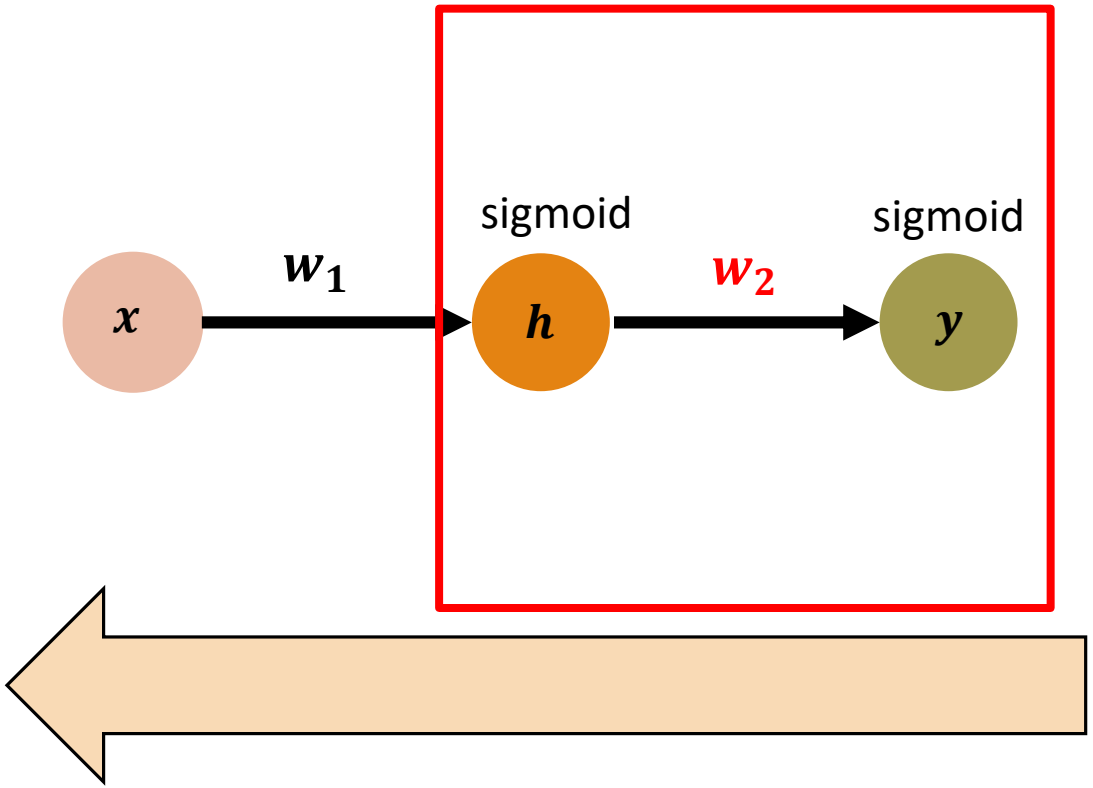


역전파 (backpropagation)

$E = \frac{1}{2}(y - y')2$ $\quad w_2 = w_2 - \eta * \frac{\partial E}{\partial w_2}$

$y' = desired\ value$

$\frac{\partial E}{\partial w_2} =$



https://www.youtube.com/watch?v=fhrORKjjU7w

역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w_2 = w_2 - \eta * \frac{\partial E}{\partial w_2}$$

$y' = desired\ value\ (training\ data)$

$$\frac{\partial E}{\partial w_2} = (y - y')\frac{\partial y}{\partial w_2}$$

$y = sig(w_2 h)$

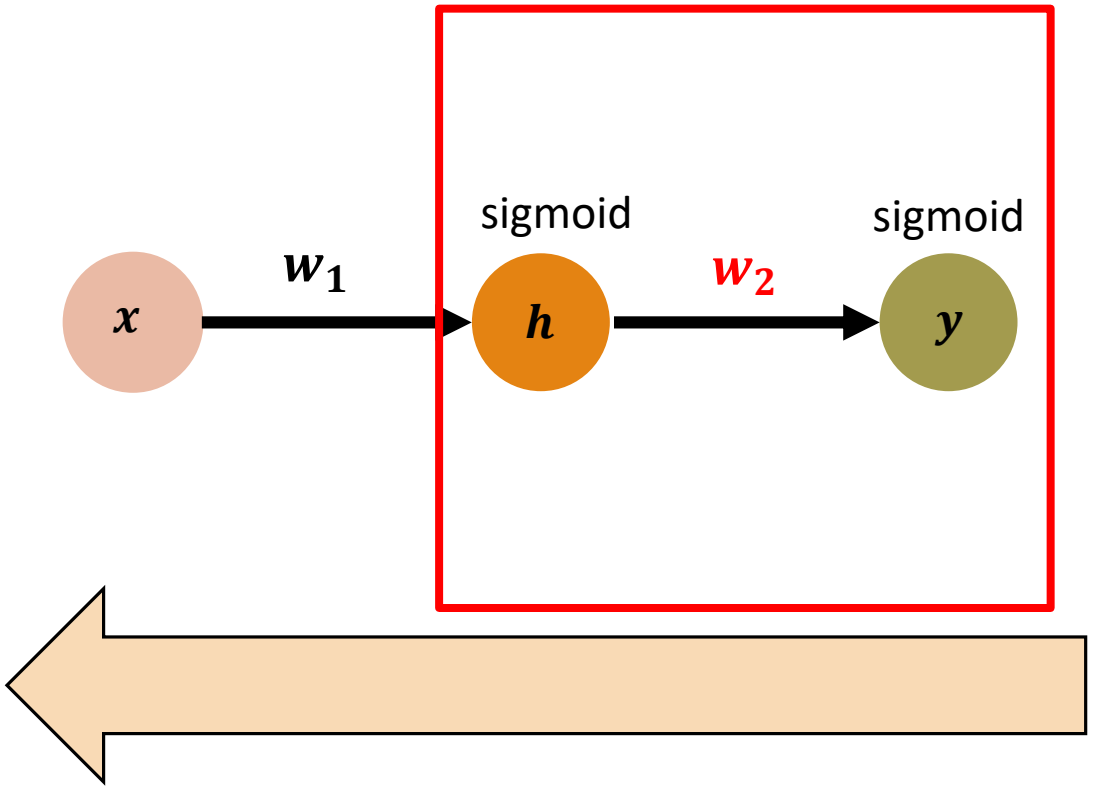
$$= (y - y')\frac{\partial}{\partial w_2} sig(w_2 h)$$

$sig(x)' = sig(x)(1 - sig(x))$

$$= (y - y')\, sig(w_2 h)(1 - sig(w_2 h))\frac{\partial w_2 h}{\partial w_2}$$

$$= (y - y')\, y(1 - y)\frac{\partial w_2 h}{\partial w_2}$$

$$= (y - y')y(1 - y)h$$

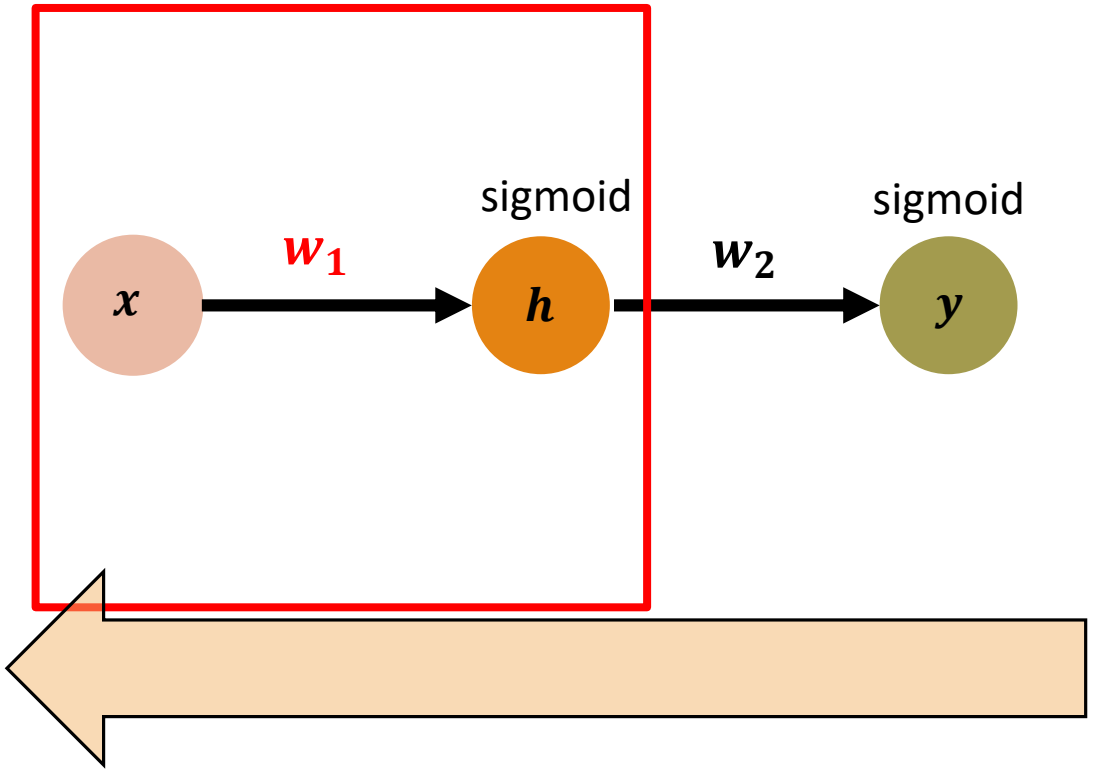


역전파 (backpropagation)

$E = \frac{1}{2}(y - y')2$ $\quad w = w_1 - \eta * \frac{\partial E}{\partial w_1}$

$y' = desired\ value$

$\frac{\partial E}{\partial w_1} =$



역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w = w_1 - \eta * \frac{\partial E}{\partial w_1}$$

$y' = desired\ value$

$$\frac{\partial E}{\partial w_1} = (y - y')\frac{\partial y}{\partial w_1}$$

$y = sig(w_2 h)$

$$= (y - y')\frac{\partial}{\partial w_1} sig(w_2 h)$$

$sig(x)' = sig(x)(1 - sig(x))$

$$= (y - y')\, y(1-y)\frac{\partial w_2 h}{\partial w_1}$$

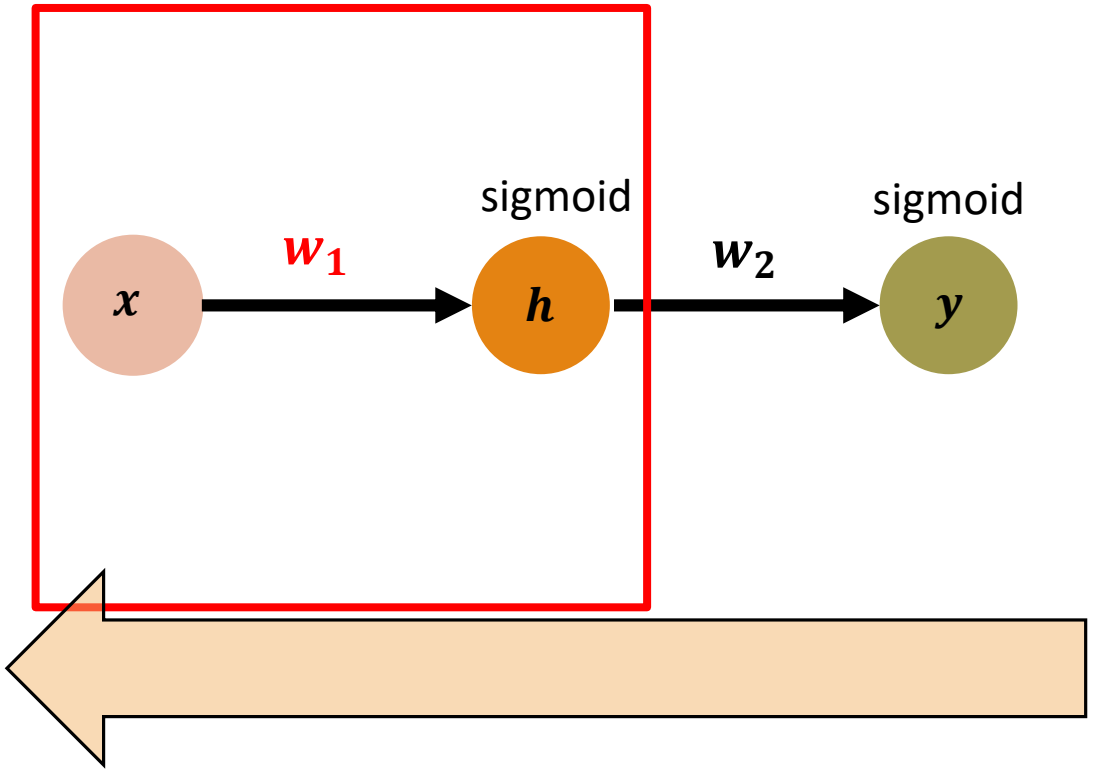
w_2 는 w_1 입장에서 상수

$$= (y - y')y(1-y)w_2\frac{\partial h}{\partial w_1}$$

$h = sig(w_1 x)$

$$= (y - y')y(1-y)w_2\frac{\partial}{\partial w_1} sig(w_1 x)$$

$$= (y - y')y(1-y)w_2 h(1-h)x$$

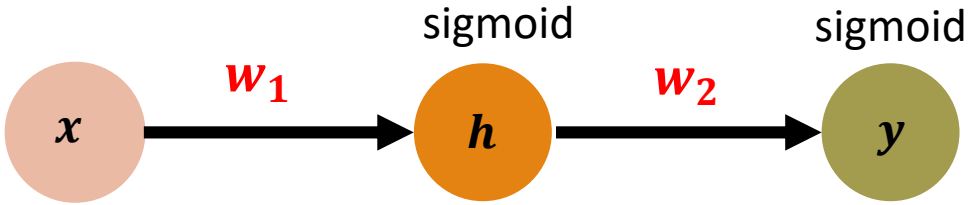


역전파 (backpropagation)

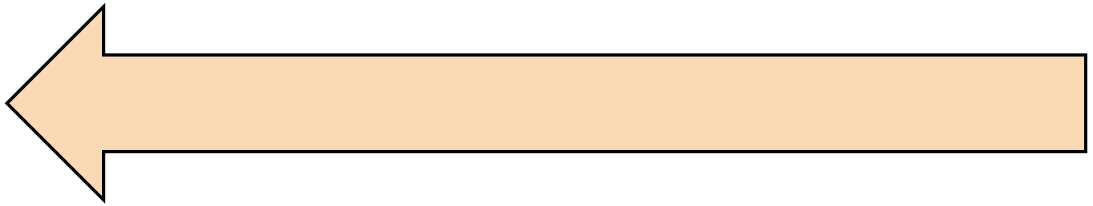
$$E = \frac{1}{2}(y - y')2 \qquad w = w - \eta * \frac{\partial E}{\partial w}$$

$$\frac{\partial E}{\partial w_2} = (y - y')y(1-y)h$$

$$\frac{\partial E}{\partial w_1} = (y - y')y(1-y)w_2h(1-h)x$$



즉, 뭐시기에 입력을 곱하면 loss의 기울기를 알 수 있음

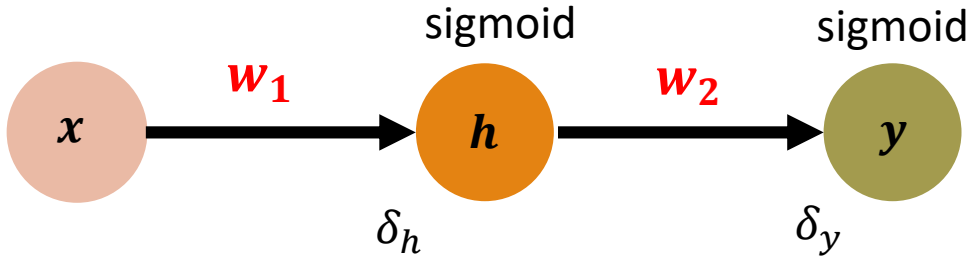


역전파 (backpropagation)

$$E = \frac{1}{2}(y - y')2 \qquad w = w - \eta * \frac{\partial E}{\partial w}$$

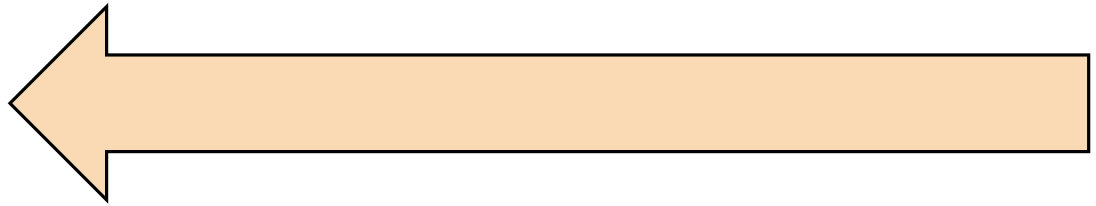
$$\frac{\partial E}{\partial w_2} = \underbrace{(y - y')y(1-y)}_{\delta_y} h$$

$$\frac{\partial E}{\partial w_1} = \underbrace{(y - y')y(1-y)w_2 h(1-h)}_{\delta_h} x$$



즉, 뭐시기에 입력을 곱하면 loss의 기울기를 알 수 있음

δ



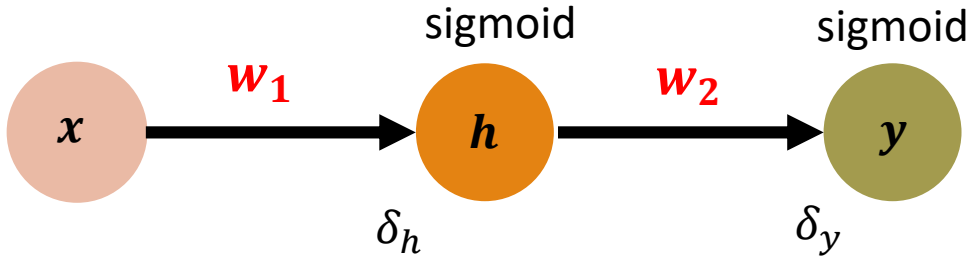
역전파 (backpropagation)

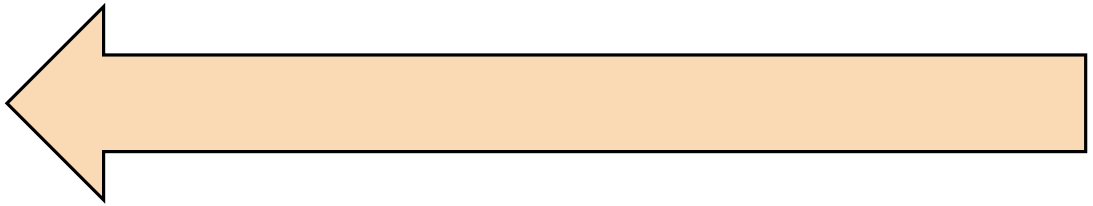
$$E = \frac{1}{2}(y - y')2 \qquad w = w - \eta * \frac{\partial E}{\partial w}$$

$$\frac{\partial E}{\partial w_2} = \underbrace{(y - y')y(1-y)}_{\delta_y} h$$

$$\frac{\partial E}{\partial w_1} = \underbrace{(y - y')y(1-y)w_2 h(1-h)}_{\delta_h} x$$

$$\delta_h = \delta_y \; sig(w_1 x)' \; w_2$$



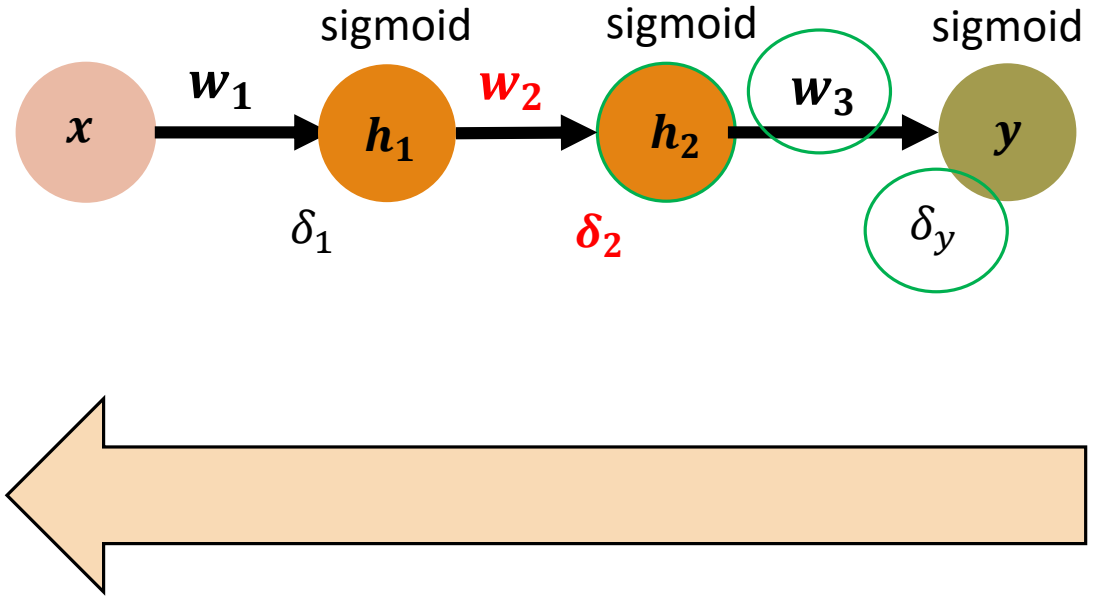


역전파 (backpropagation)

$\delta_2 = \delta_y \;\; sig(w_2 h_1)' \;\; w_3$

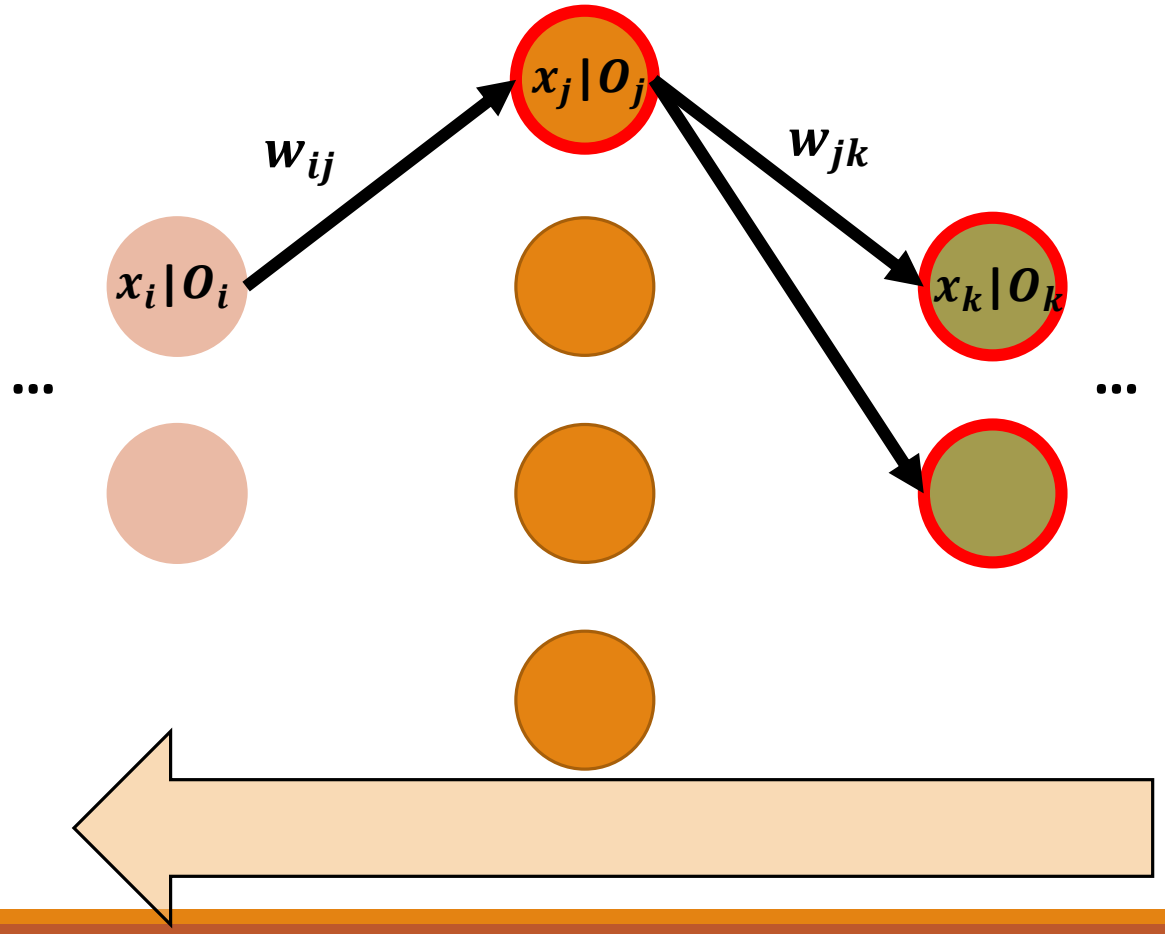
$\delta_1 = \delta_2 \;\; sig(w_1 x)' \;\; w_2$

…



역전파 (backpropagation) - 4

$$\delta_j = \sum_k \delta_k w_{jk} f'(O_j)$$

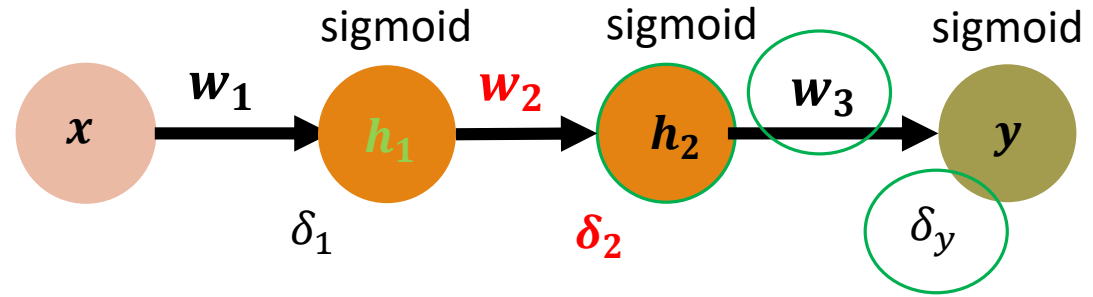


역전파 (backpropagation)

$\delta_2 = \delta_y \;\; sig(w_2 h_1)' \;\; w_3$

$\frac{\partial E}{\partial w_2} = \delta_2 h_1$

$w_2 = w_2 - \eta * (\delta_2 h_1)$



Activation function

네트워크에 비선형을 추가하기 위해 사용됨

Why?

- H(X) = f(g(i … (X))) -> H(x) = WX -> 아무리 층이 많아도 선형함수

Activation function 추가

- 각 층의 출력 결과를 비선형화 하고 다음 층에 넘겨줌.
- H(X) = activationfunction(f(activationfunction(g(activationfunction(i … (X))))))

단층 퍼셉트론에서는 출력값 영역을 나누는 선이 직선 형태로 표현되기 때문에 XOR을 표현할 수 없다.

그래서 층을 하나 더 쌓아서 `(input -> NAND,OR -> AND -> output)` XOR을 표현하게 되는데, 이게 가능했던 이유는 활성화 함수로 비선형 함수인 계단 함수를 사용해 결과값의 영역을 나누는 선이 곡선으로 표현될 수 있었기 때문이다.

이렇게 기본적인 연산을 조합해 더욱 복잡한 연산을 표현하기 위해서는 비선형 함수가 필요하다.

Activation function 종류

활성 함수에 따라 학습 성능이 달라진다.

step function

sigmoid

hyperbolic tangent

ReLU

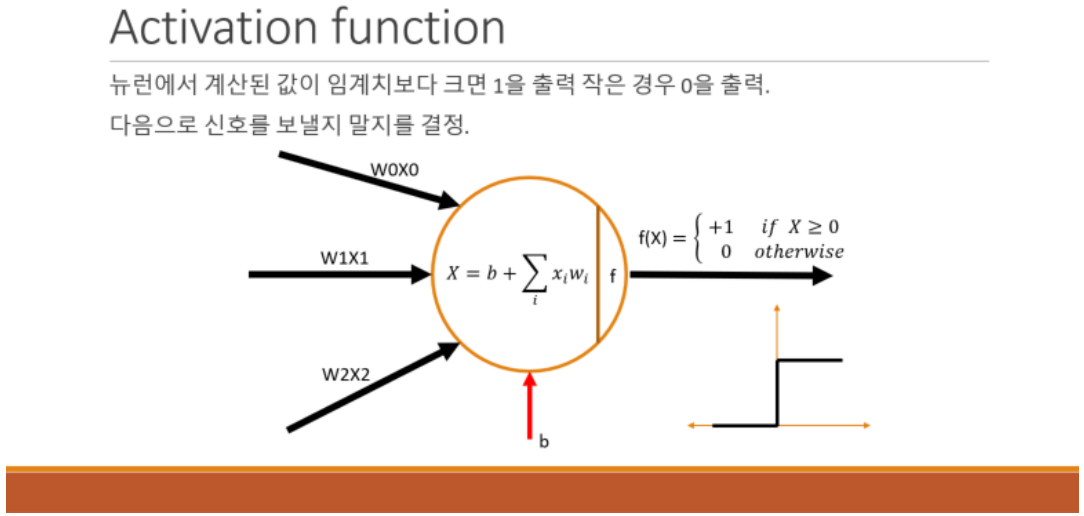
softmax

…

Activation function – step function

$$step(x) = \begin{cases} 1 & x \geq 0 \\ 0 & otherwise \end{cases}$$

임계 값 기준으로 활성화 되거나 혹은 비활성화 되는 형태



Activation function – sigmoid(1)

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

미분이 편하다 : sigmoid(x)’ = sigmoid(x)(1 - sigmoid(x))

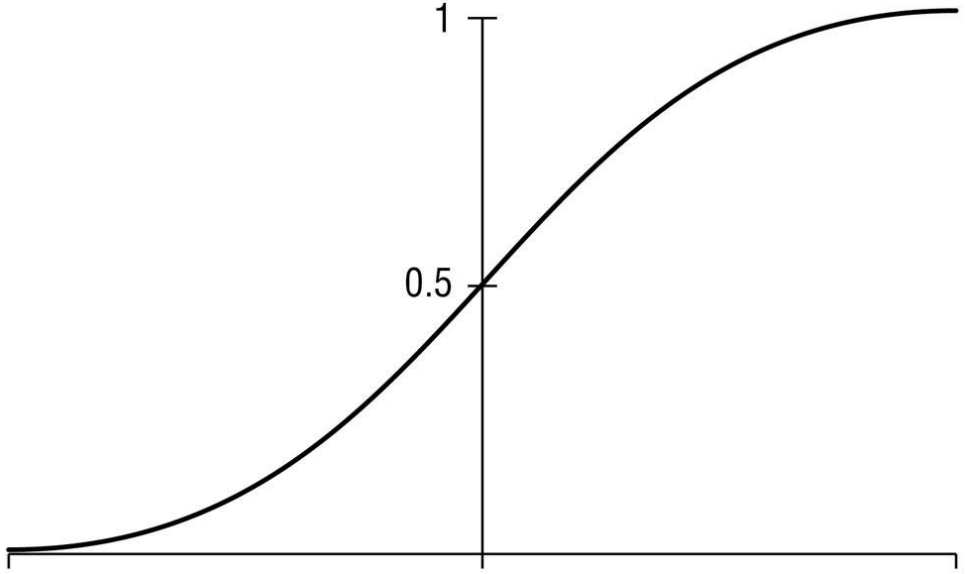
결과 값이 [0, 1] 사이로 제한 – 입력이 작을 때 0, 클 때 1에 수렴

- 확률 값을 리턴한다.

역전파 신경망에 많이 쓰임

가중치나 바이어스를 조금 변화 시켰을 때

출력이 조금씩 변화 하도록 만들 수 있다.



Activation function – sigmoid(2)

Gradient Vanishing (그래디언트가 죽는 현상 -> 학습이 되지 않는다)

지수함수라서 계산이 복잡하다.

Gradient Vanishing

- 이전 레이어로 전파되는 그래디언트가 0에 가까워지는 현상.
  - 양 극단의 미분값이 0에 가깝기 때문
- 레이어를 깊게 쌓으면 파라미터의 업데이트가 제대로 이루어지지 않음.

중심값이 0이 아니다.

Activation function – hyperbolic tangent

$tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ = 2sigmoid(2x) - 1

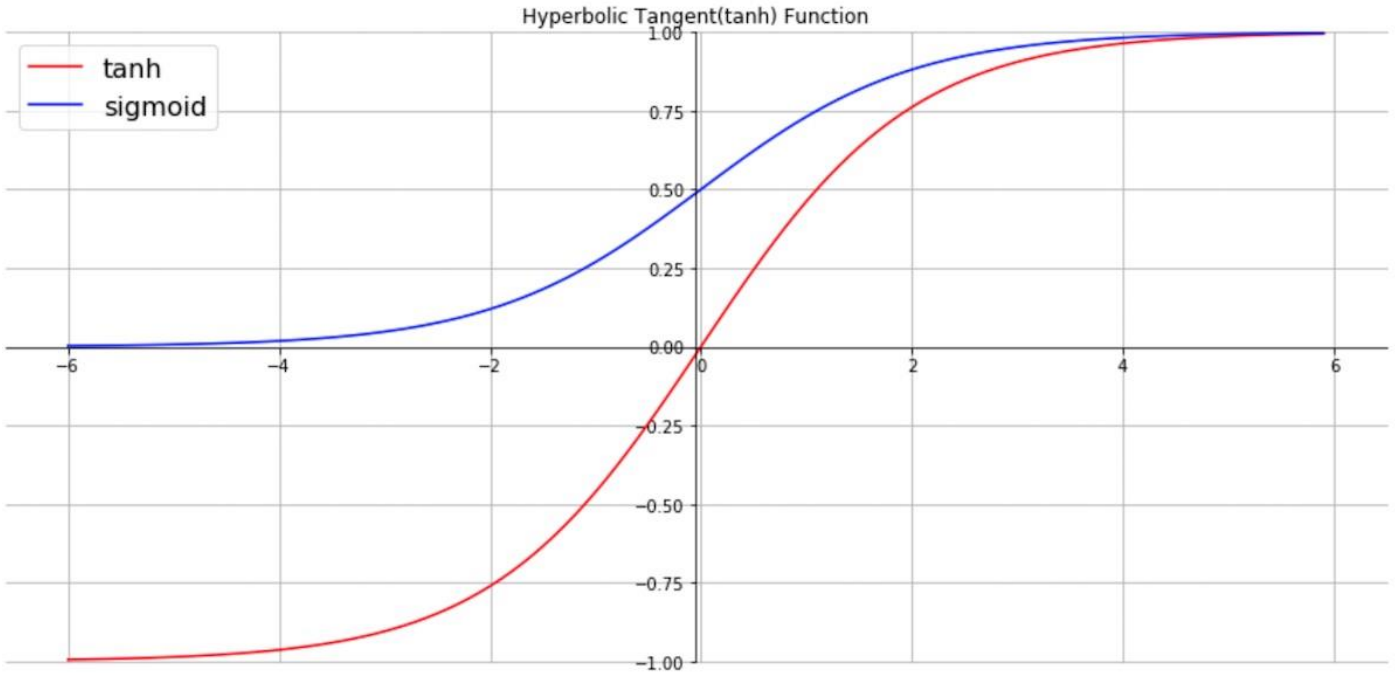
결과값 범위 : (-1,1)

시그모이드 확장버젼

출력범위가 sigmoid에 비해 더 넓고,
미분계수 최대값이 sigmoid의 4배.

=> 더 빠르게 수렴하여 학습

동일한 Gradient Vanishing 문제



Activation function – ReLU

$$ReLU(x) = \begin{cases} x & x \geq 0 \\ 0 & otherwise \end{cases}$$

0에서 확 꺾이기 때문에 비선형.

계산이 매우 효율적

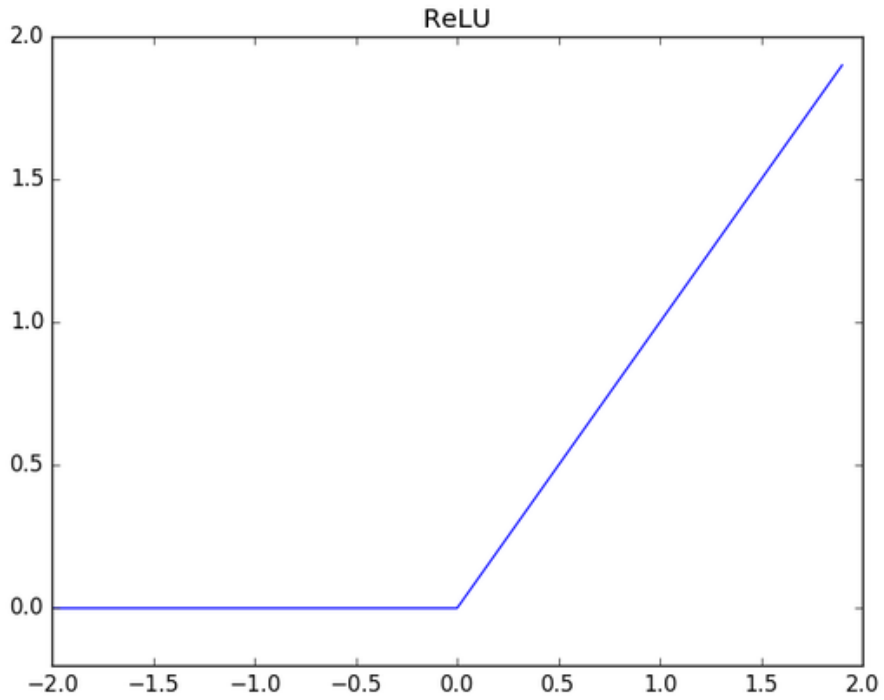
Sigmoid 보다 계산 속도가 빠르다

But. 음수가 나온 노드는 학습이 불가능 (dying ReLU)

-> 간단한 네트워크에서는 성능 저하

-> leaky ReLU와 같은 다른 ReLU 존재

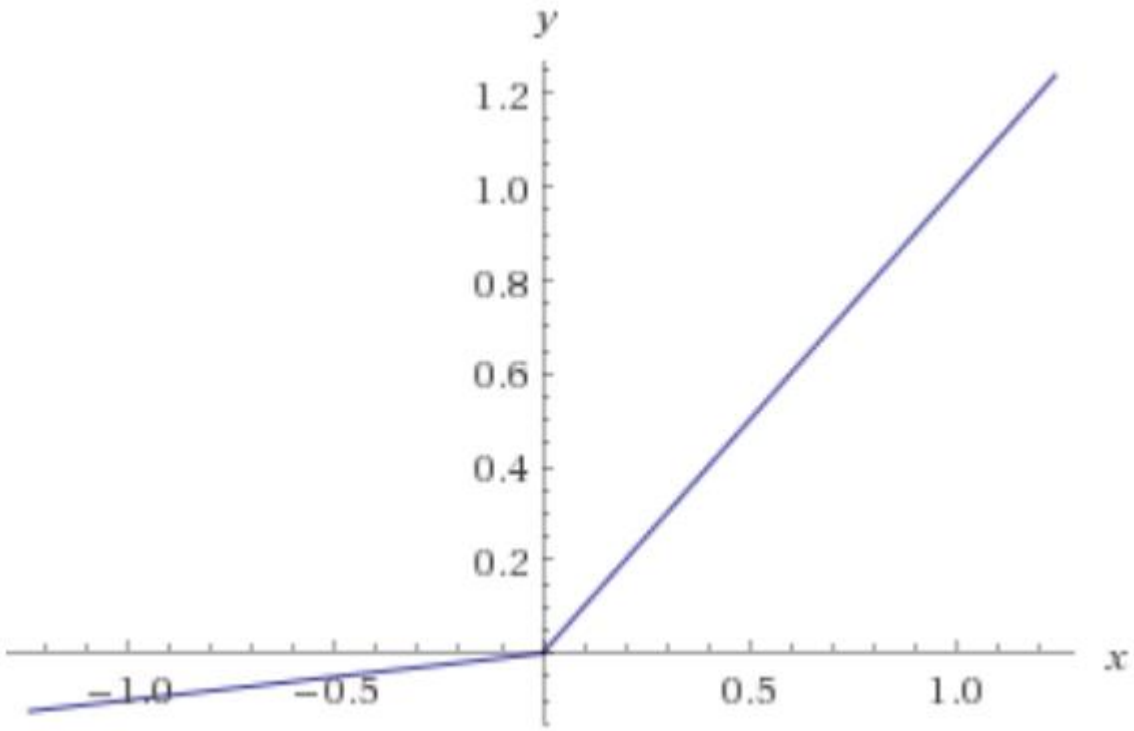
But. 0에서 미분 불가능



Activation function – Leaky ReLU

$$Leaky\ ReLU(x) = \begin{cases} x & x \geq 0 \\ 0.01x & otherwise \end{cases}$$

ReLU 함수의 변형으로 음수에 대해 1/10로 값을 줄여서 사용하는 함수



Activation function - softmax

$$\sigma(\boldsymbol{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \mathrm{j} = 1,2,\ldots,\mathrm{K}$$

입력 받은 값을 출력으로 0 ~ 1 사이의 정규화 된 값

출력 값들의 총합은 항상 1

주로 출력 노드에서 사용.

분류, 강화 학습에 사용된다.

vs sigmoid?

sigmoid는 해당 뉴런으로 들어오는 입력들과 bias에 의해 출력이 결정

softmax는 다른 뉴런의 출력 값들 과의 상대적인 비교를 통해 출력이 결정