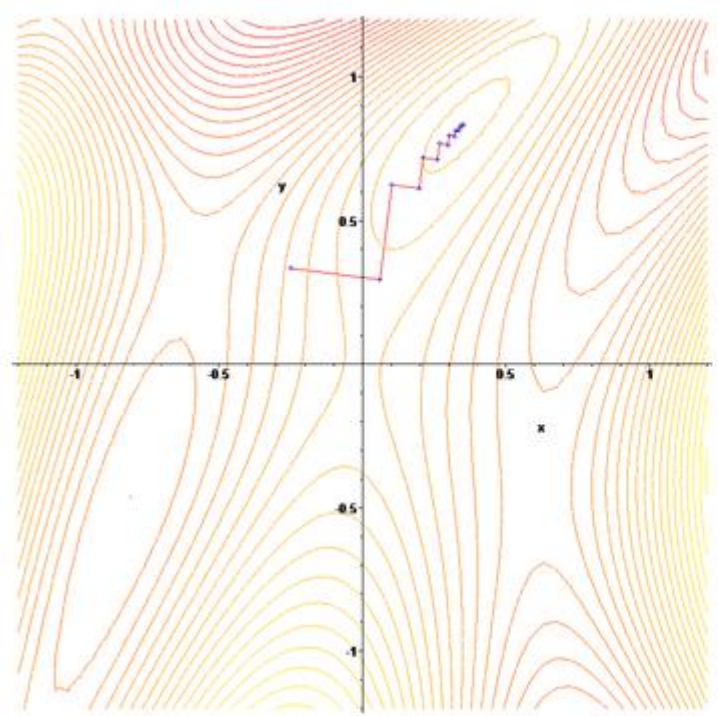
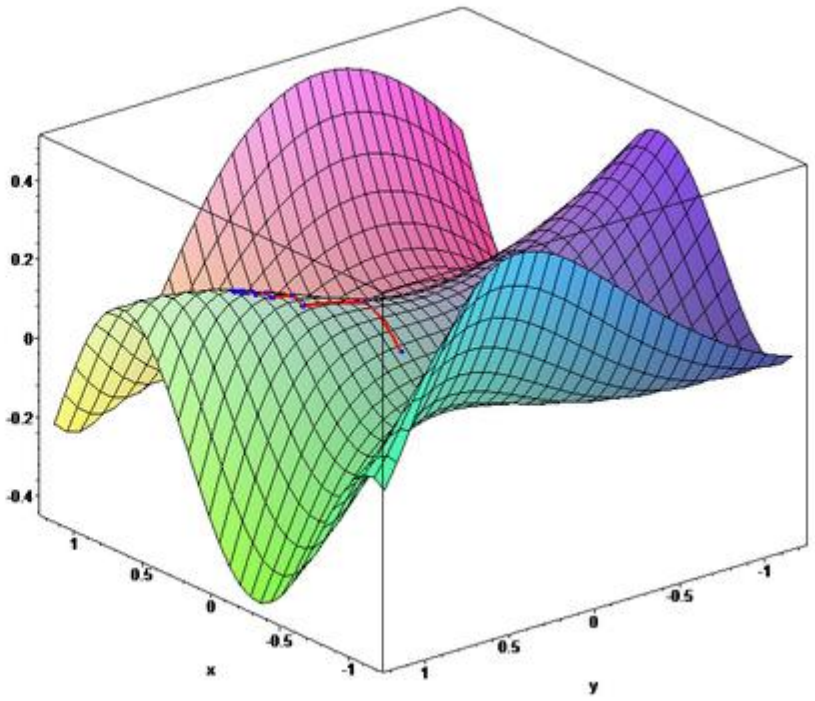
# 경사하강법 및 역전파 알고리즘

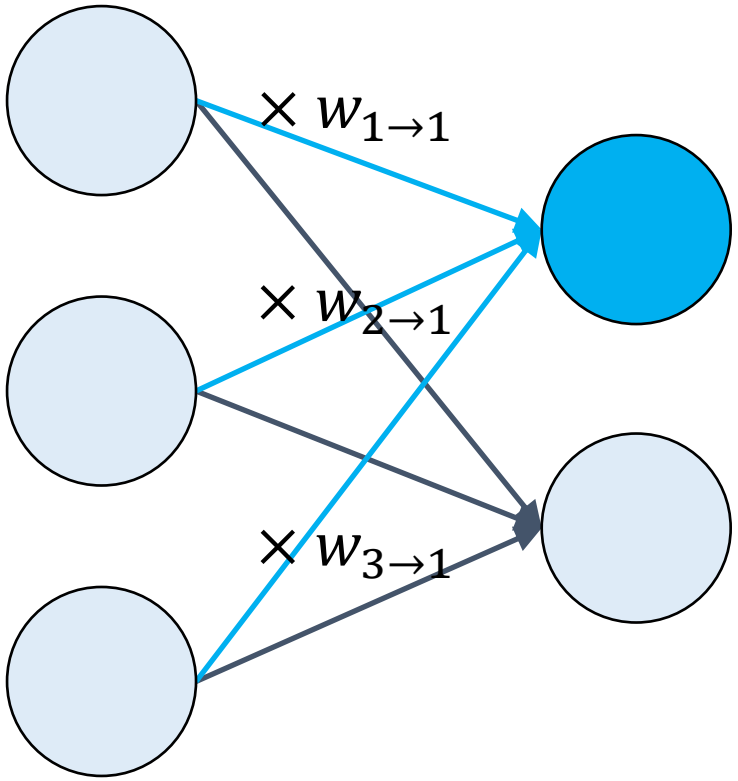


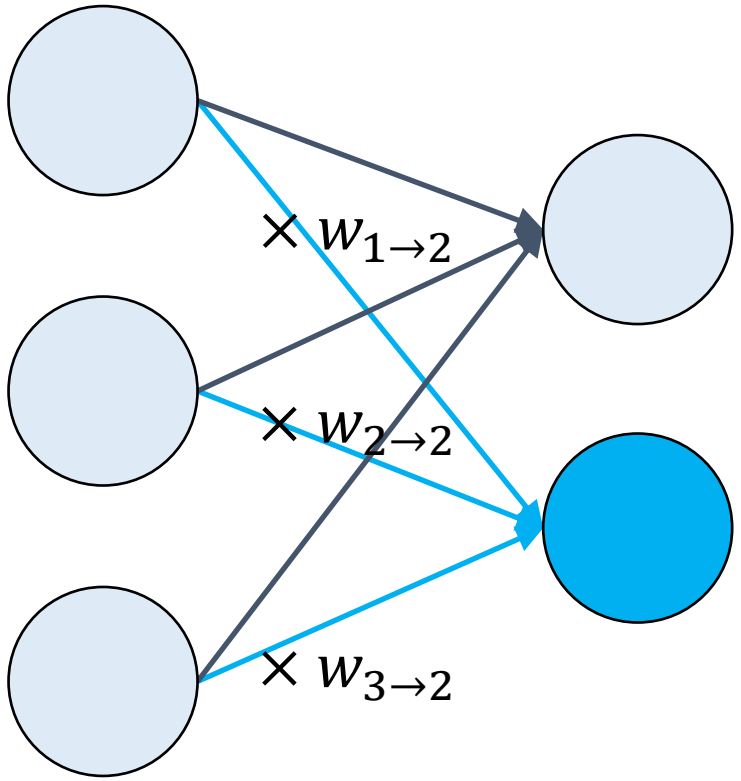




박경규

# Linear Layer

- 신경망의 가장 기본 구성 요소, FC(Fully Connected) Layer, 내부 파라미터에 따른 선형 변환을 수행하는 함수
- 각 입력 노드들에 weight(가중치)를 곱하고  모두 합친 뒤, bias(편향)을 더합니다.
- 행렬 곱으로 구현 가능하며, n차원에서 m차원으로의 선형 변환 함수입니다.
- 이 함수의 파라미터(W, b)를 잘 조절하면, 주어진 입력에 대해 원하는 출력을 만들 수 있습니다.

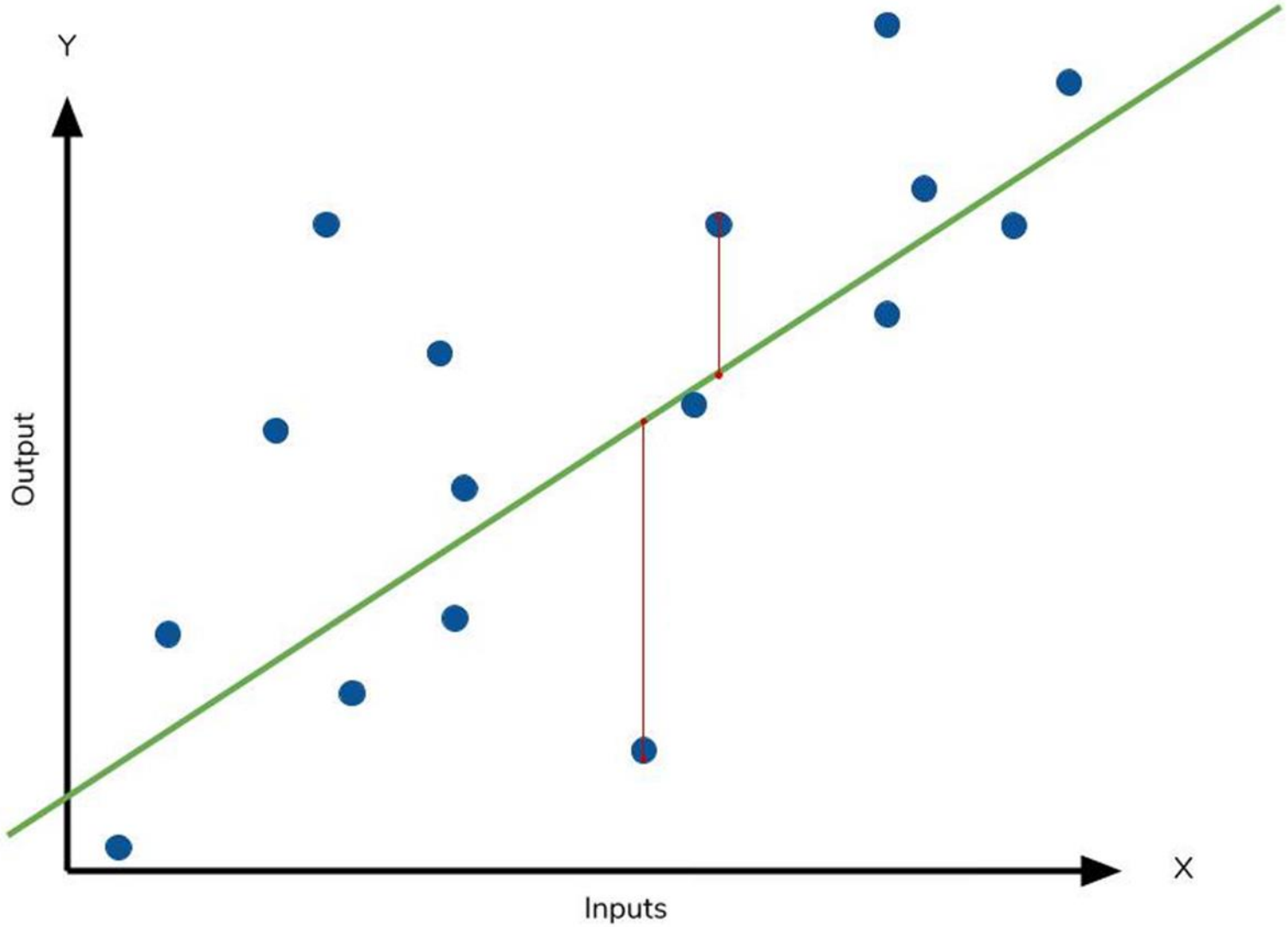



$$y = f(x) = x \cdot W + b$$

# Loss

- 원하는 출력값(target, $y$)과 실제 출력값(output, $\hat{y}$)의 차이의 합
- Linear Layer 파라미터(W, b)를 바꾸면서 Loss를 최소화 해야 합니다.



# Cost function

w*x + b = z

σ(z) = a

σ : 활성화함수(activation function)

$$C = \frac{1}{2n} \sum_x \|y(x) - a^L(x)\|^2$$

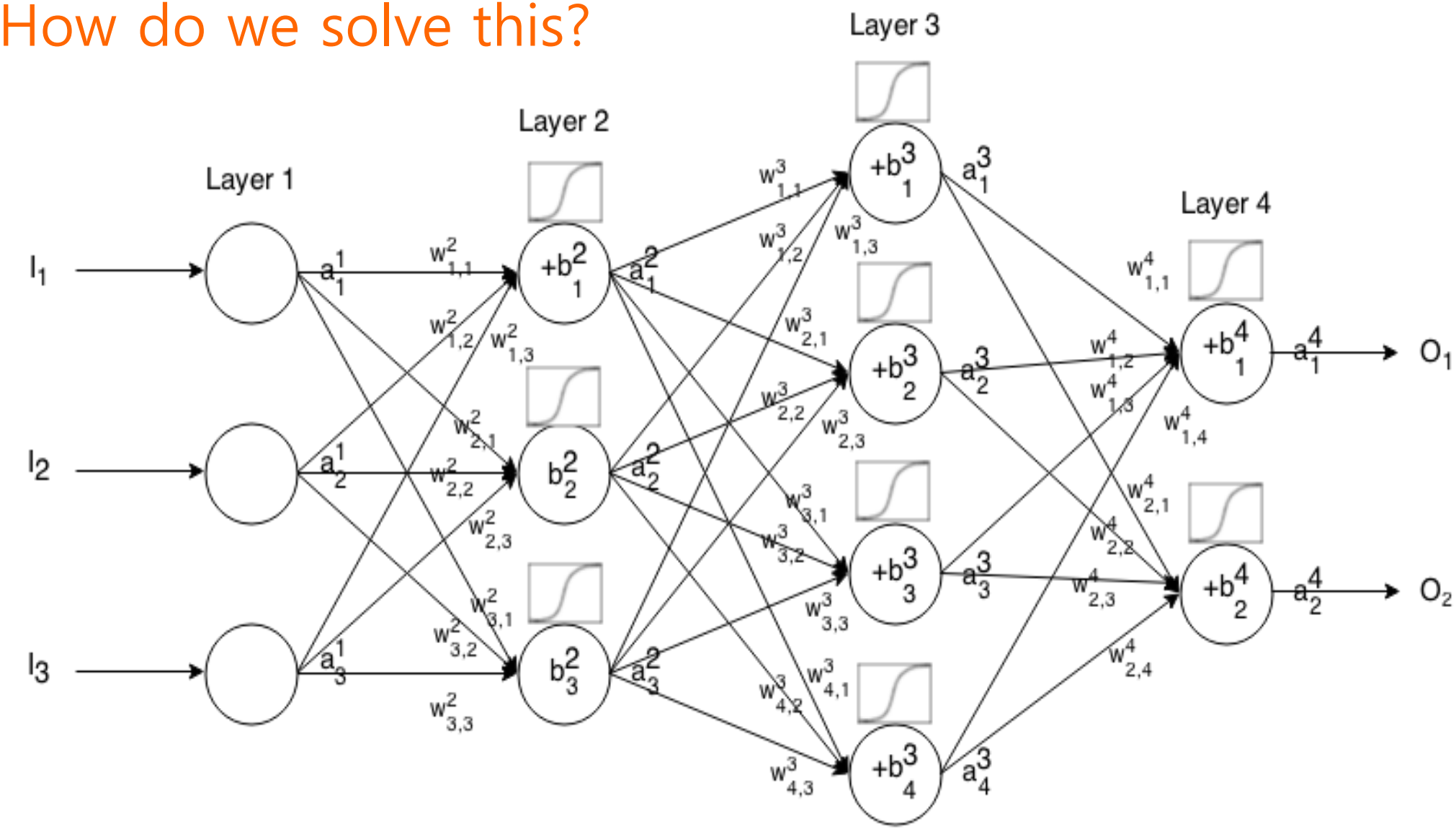
a(x)는 가중치(weights)와 편향(biases) 정보를 가지고 있다.

$$C(W, B, S^r, E^r)$$

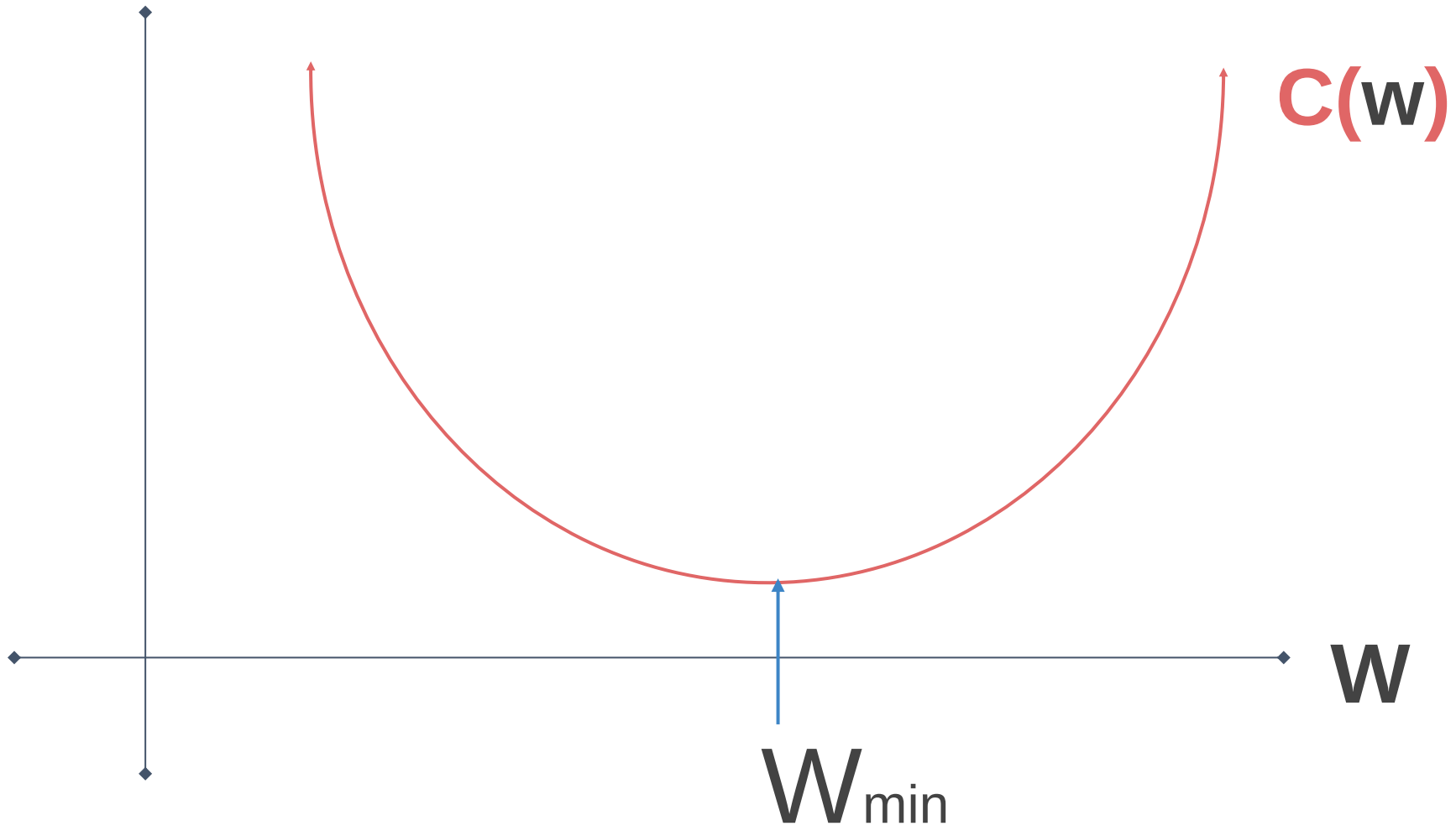
**W** : 가중치(weights)
**B** : 편향(biases)
**$S^r$** : 입력값(input of a single training sample)
**$E^r$** : 목표값(desired output of that training sample)

# Cost function

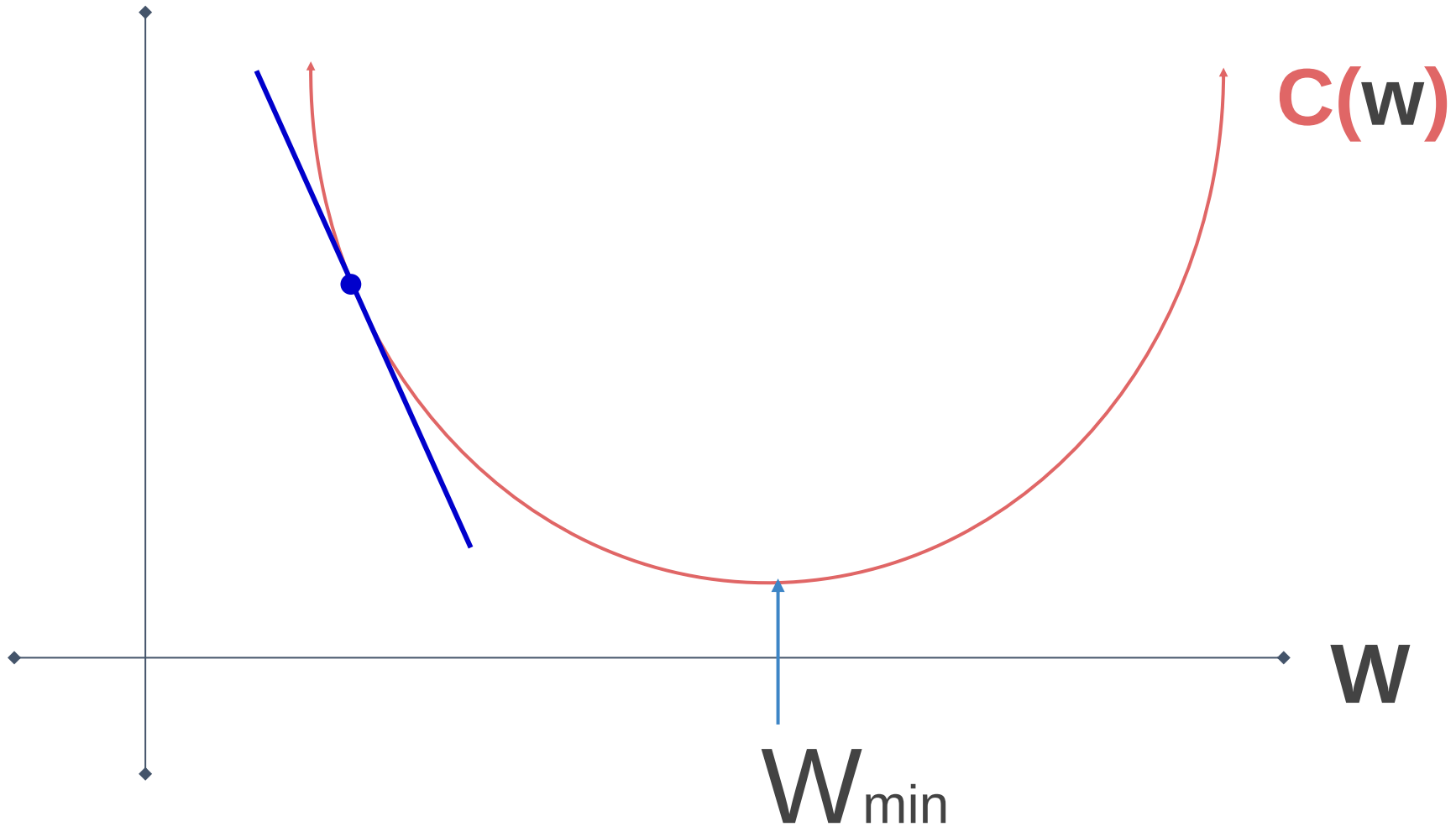
## How do we solve this?



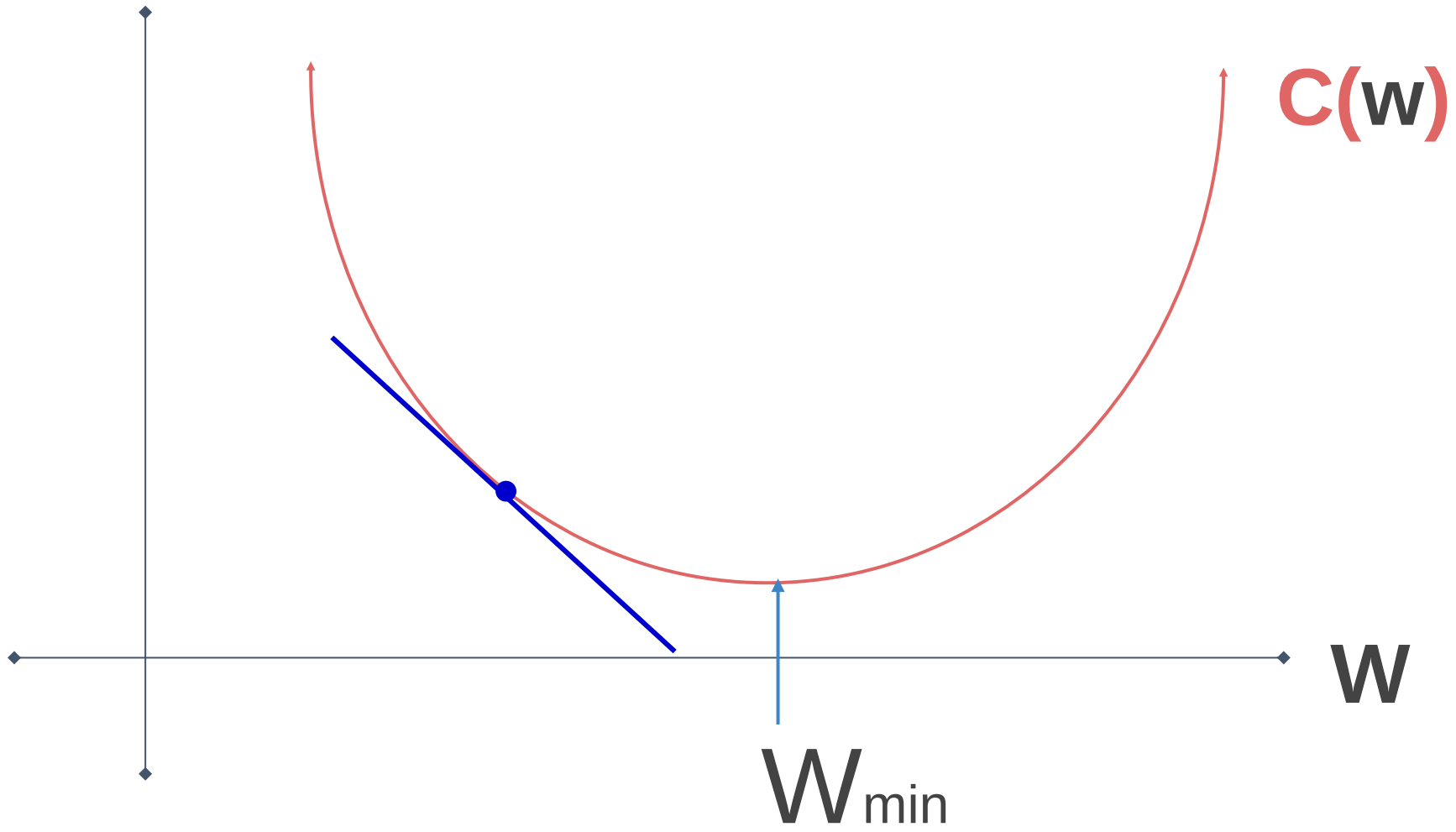
# Cost function



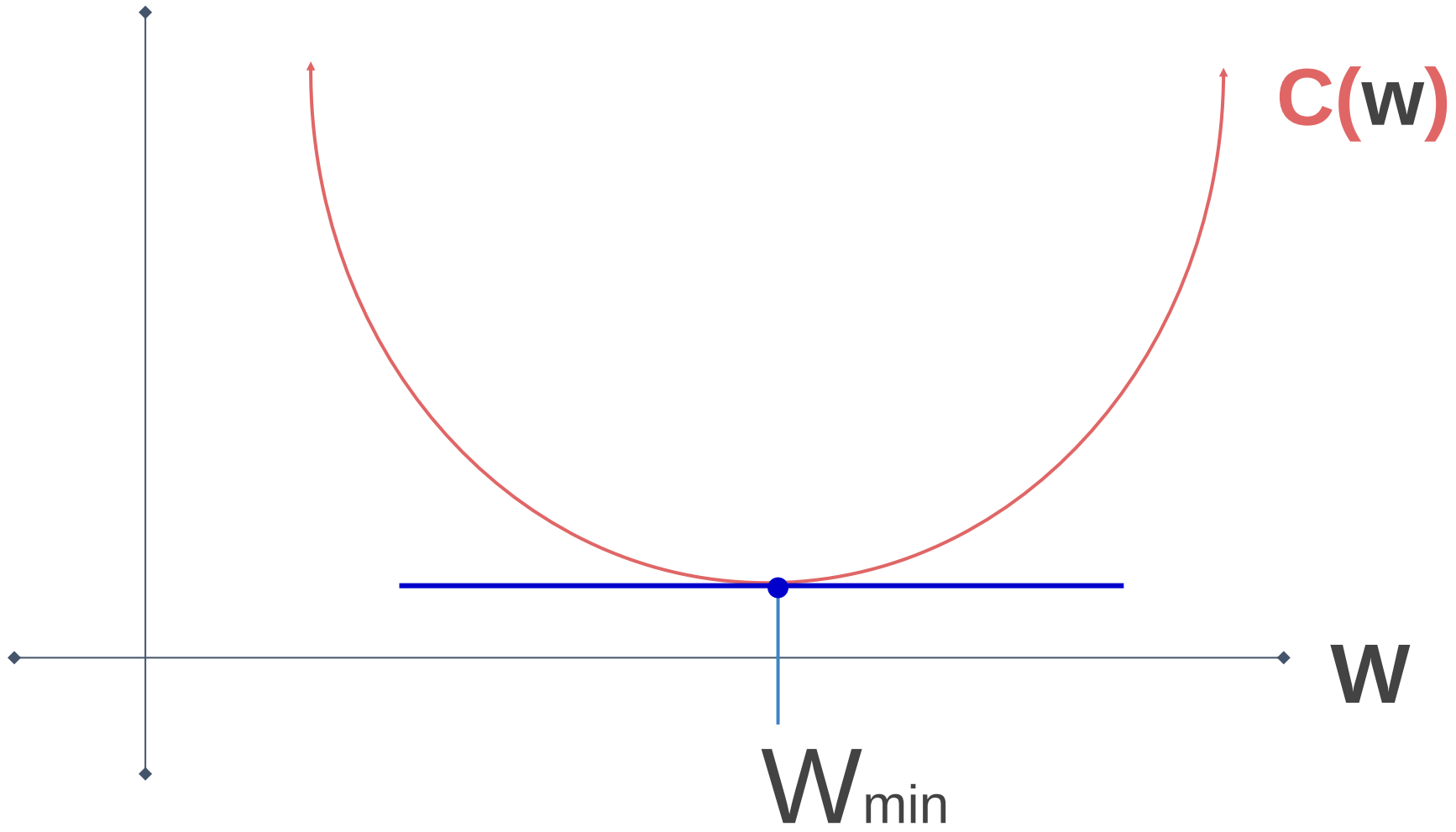
# Cost function



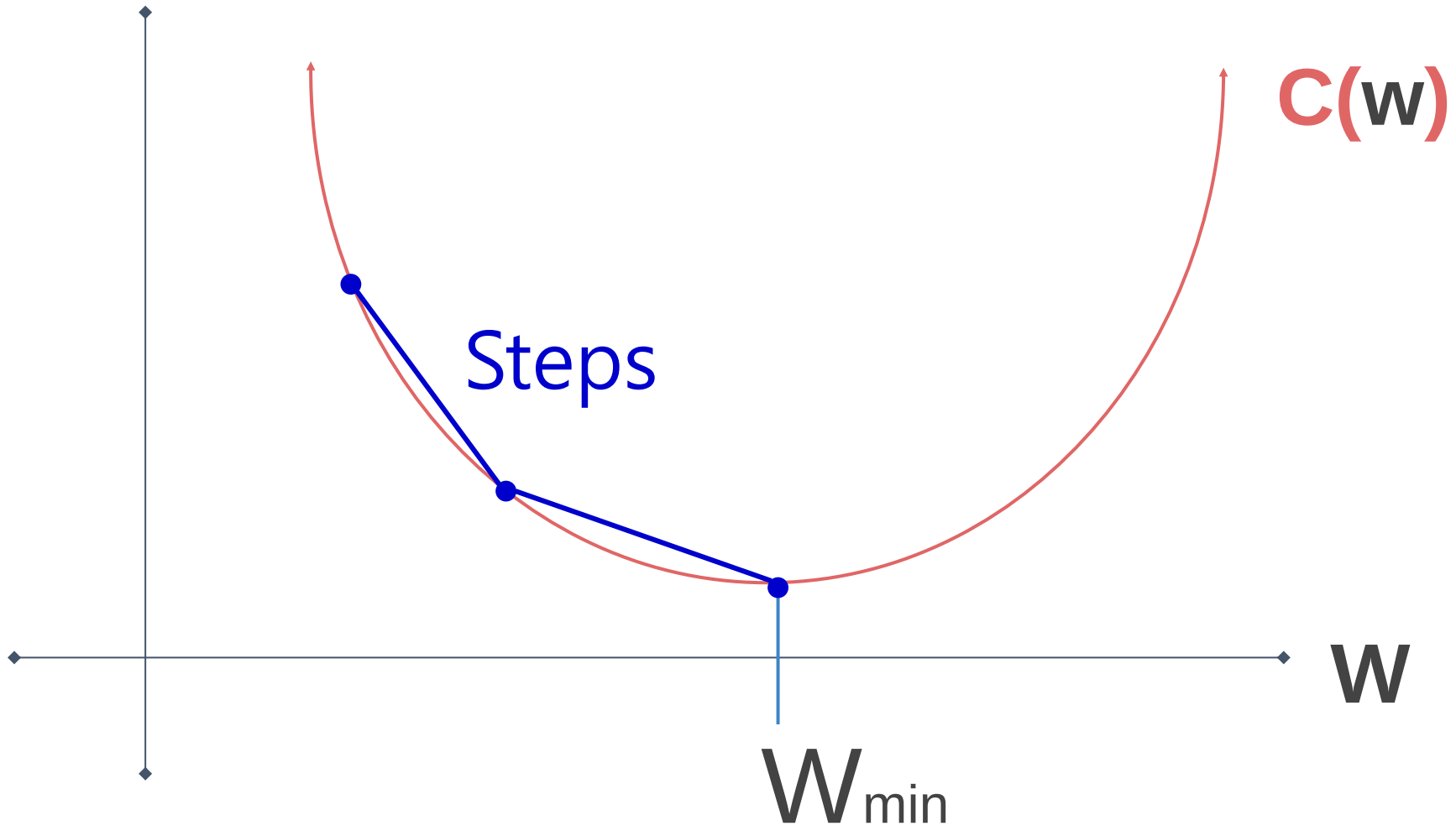
# Cost function



# Cost function

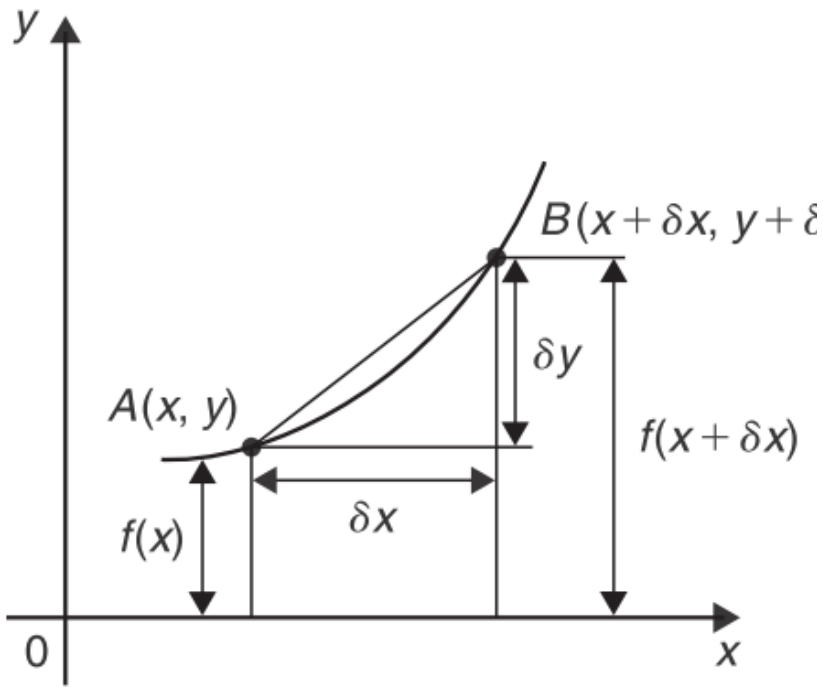


# Cost function

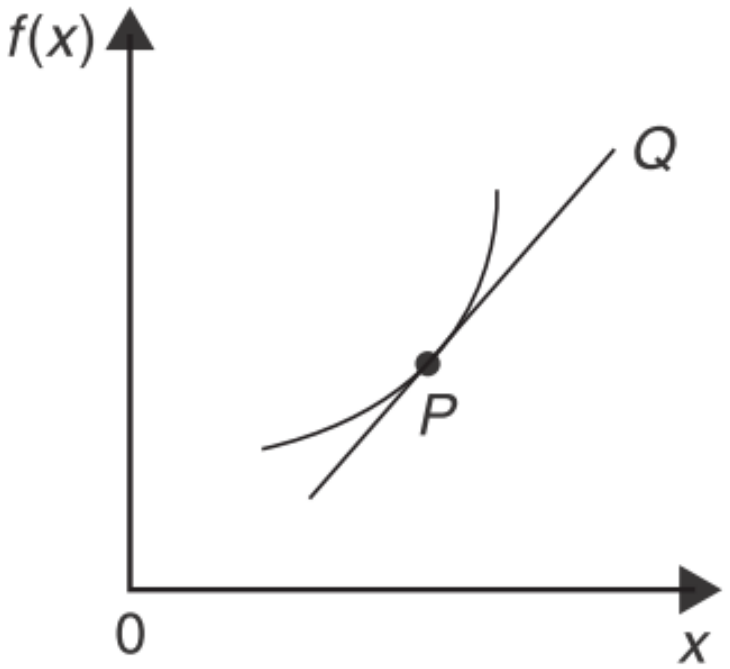


# 기울기

- 함수의 두 입력 값에 대한 출력 값의 변화량의 비율




$$\frac{\delta y}{\delta x} = \frac{f(x+\delta x) - f(x)}{\delta x}$$



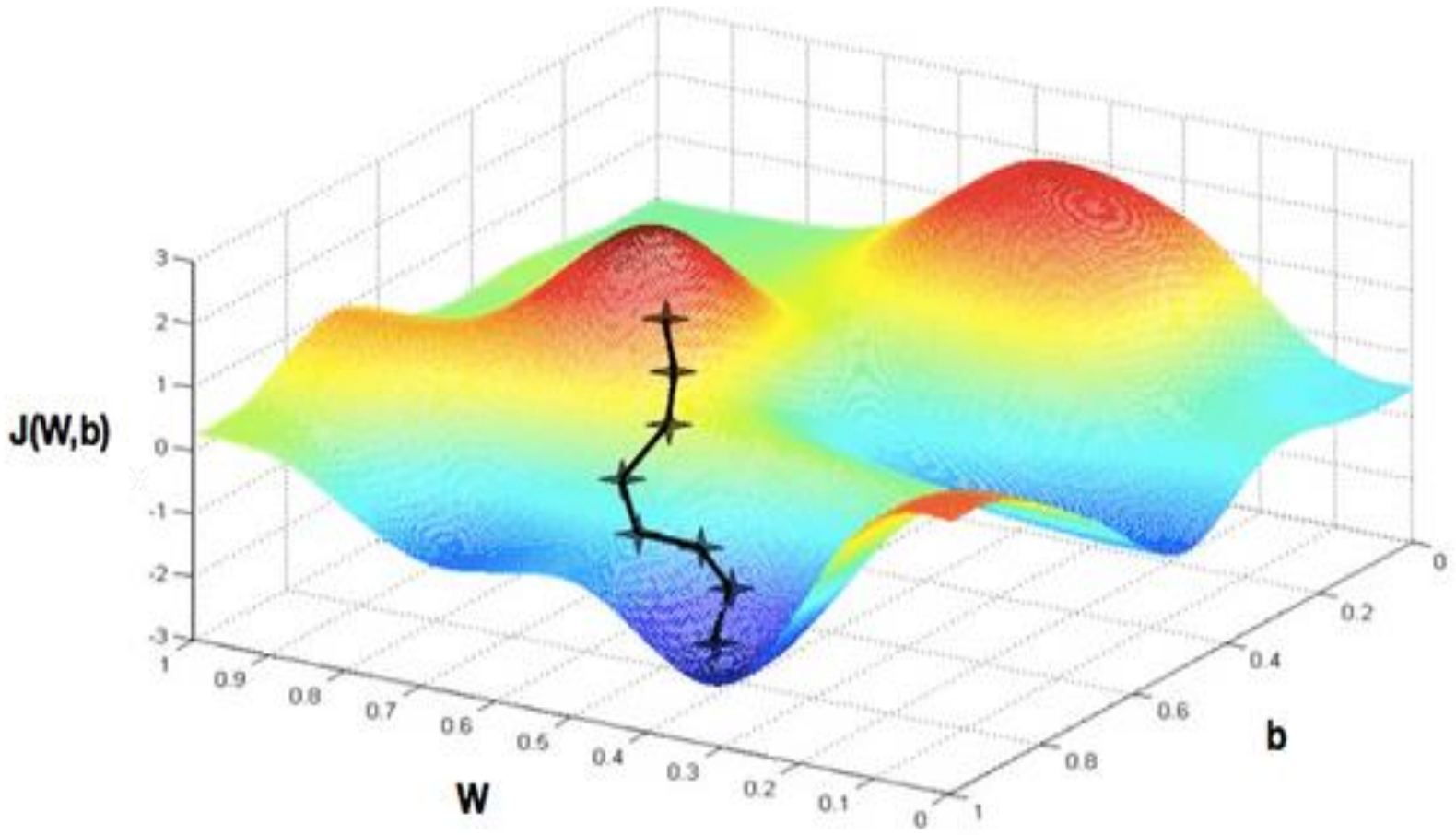

$$\lim_{\delta x \to 0} \left\{ \frac{f(x+\delta x) - f(x)}{\delta x} \right\}$$

$$\frac{dy}{dx} = \lim_{\delta x \to 0} \frac{\delta y}{\delta x}$$

$$'(x) = \lim_{\delta x \to 0} \left\{ \frac{f(x+\delta x) - f(x)}{\delta x} \right\}$$

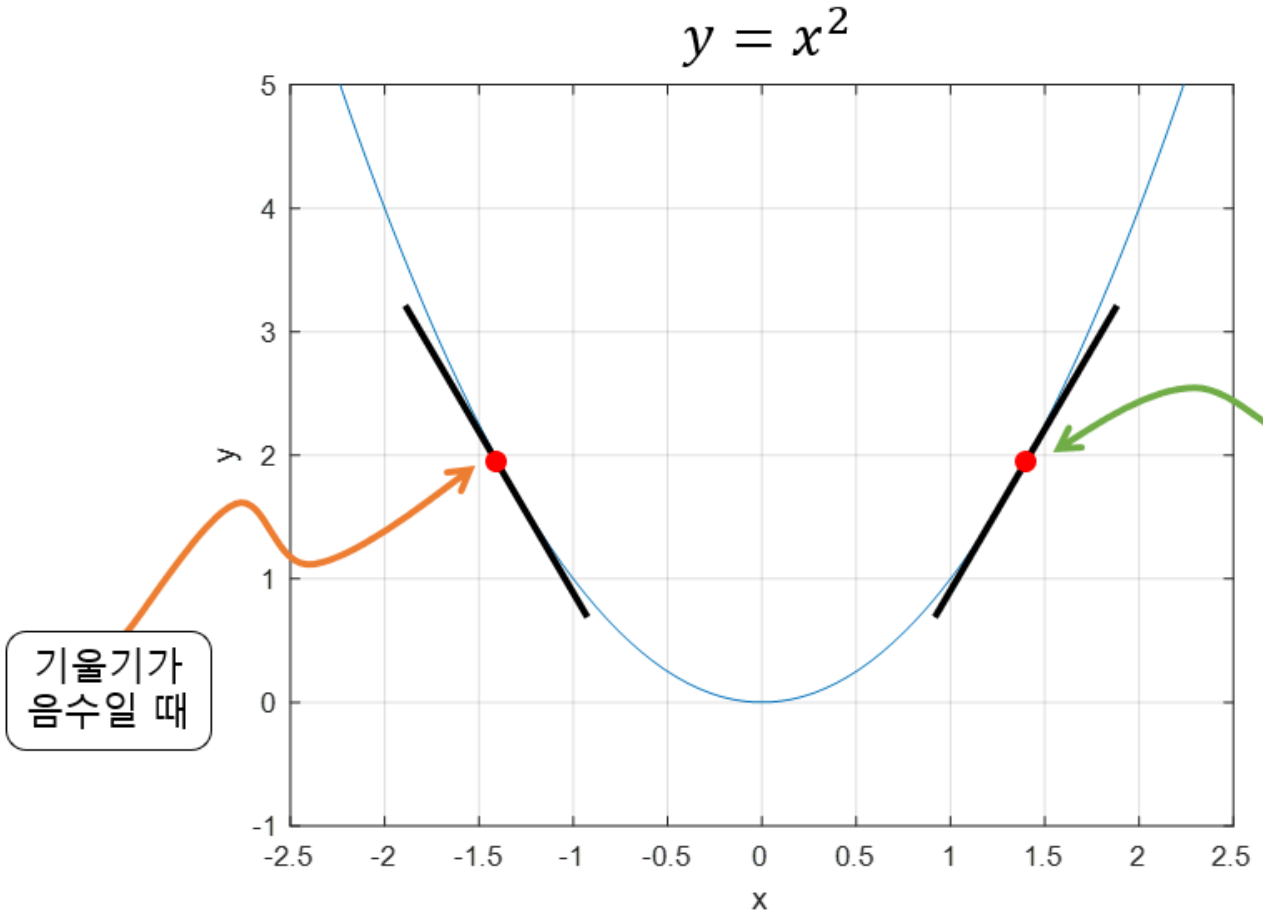
# 경사하강법(Gradient Descent)

앞이 보이지 않는 안개가 낀 산을 내려올 때는 모든 방향으로 산을 더듬어가며 산의 높이가 가장 낮아지는 방향으로 한 발씩 내디뎌갈 수 있으며, 이러한 방법이 steepest descent 방법입니다.



# 경사하강법(Gradient Descent)

Gradient Descent 방법은 steepest descent 방법이라고도 불리는데, 함수 값이 낮아지는 방향으로 독립 변수 값을 변형시켜가면서 최종적으로는 최소 함수 값을 갖도록 하는 독립 변수 값을 찾는 방법입니다.



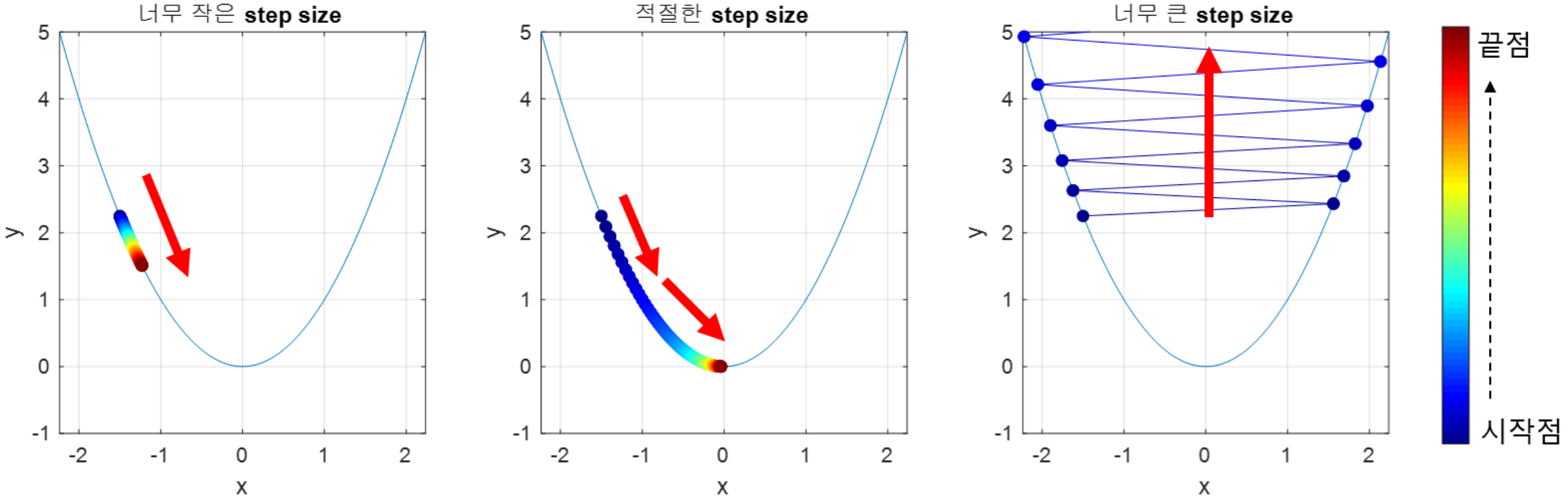

$$x_{i+1} = x_i - \text{이동거리} \times \text{기울기의 부호}$$

$$x_{i+1} = x_i - \alpha \frac{df}{dx}(x_i)$$

https://angeloyeo.github.io/2020/08/16/gradient_descent.html

# 경사하강법(Gradient Descent)

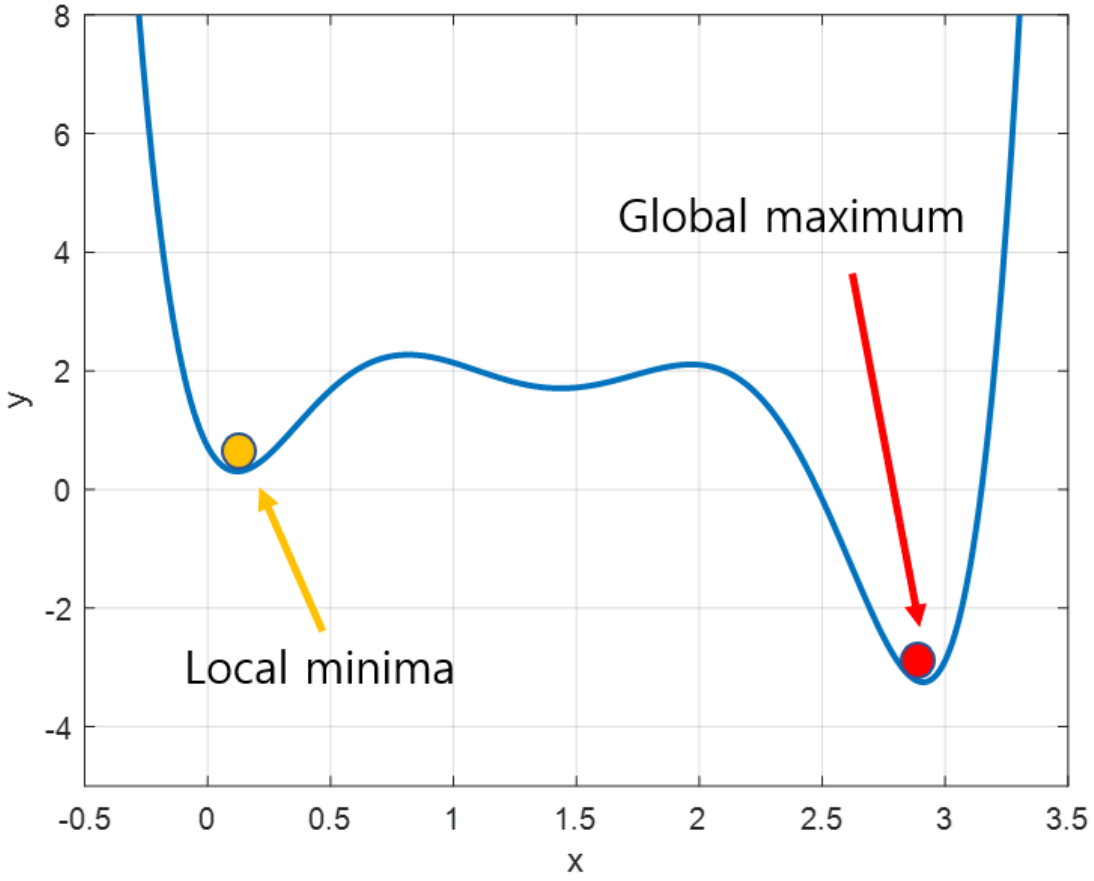
- step size(learning rate $\alpha$)가 큰 경우 한 번 이동하는 거리가 커지므로 빠르게 수렴할 수 있다는 장점이 있지만, 최소값을 계산하도록 수렴하지 못하고 함수 값이 계속 커지는 방향으로 최적화가 진행될 수 있습니다.
- step size가 너무 작은 경우 발산하지는 않을 수 있지만 최적의 x를 구하는데 소요되는 시간이 오래 걸립니다.




https://angeloyeo.github.io/2020/08/16/gradient_descent.html

# 경사하강법(Gradient Descent)

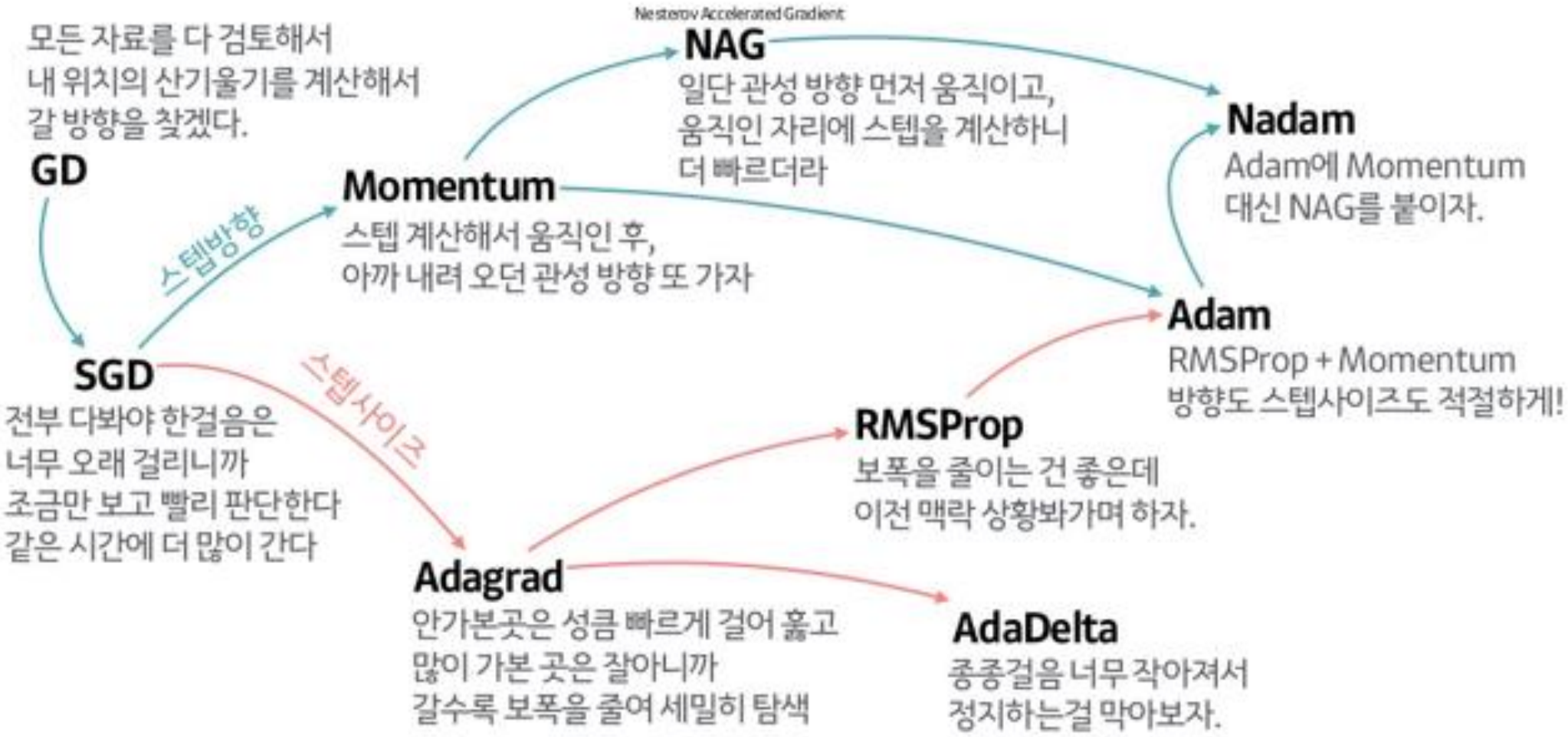
- Gradient descent는 어떤 경우에는 local minima에 빠져  계속 헤어나오지 못하는 경우도 발생합니다.
- 딥러닝은 수백만 차원의 loss 함수 surface에서 global minima를 찾는 문제로, 동시에 수 많은 차원에서 local minima를 위한 조건이 만족되기는 어려우므로 크게 걱정하지 않아도 됩니다.




https://angeloyeo.github.io/2020/08/16/gradient_descent.html

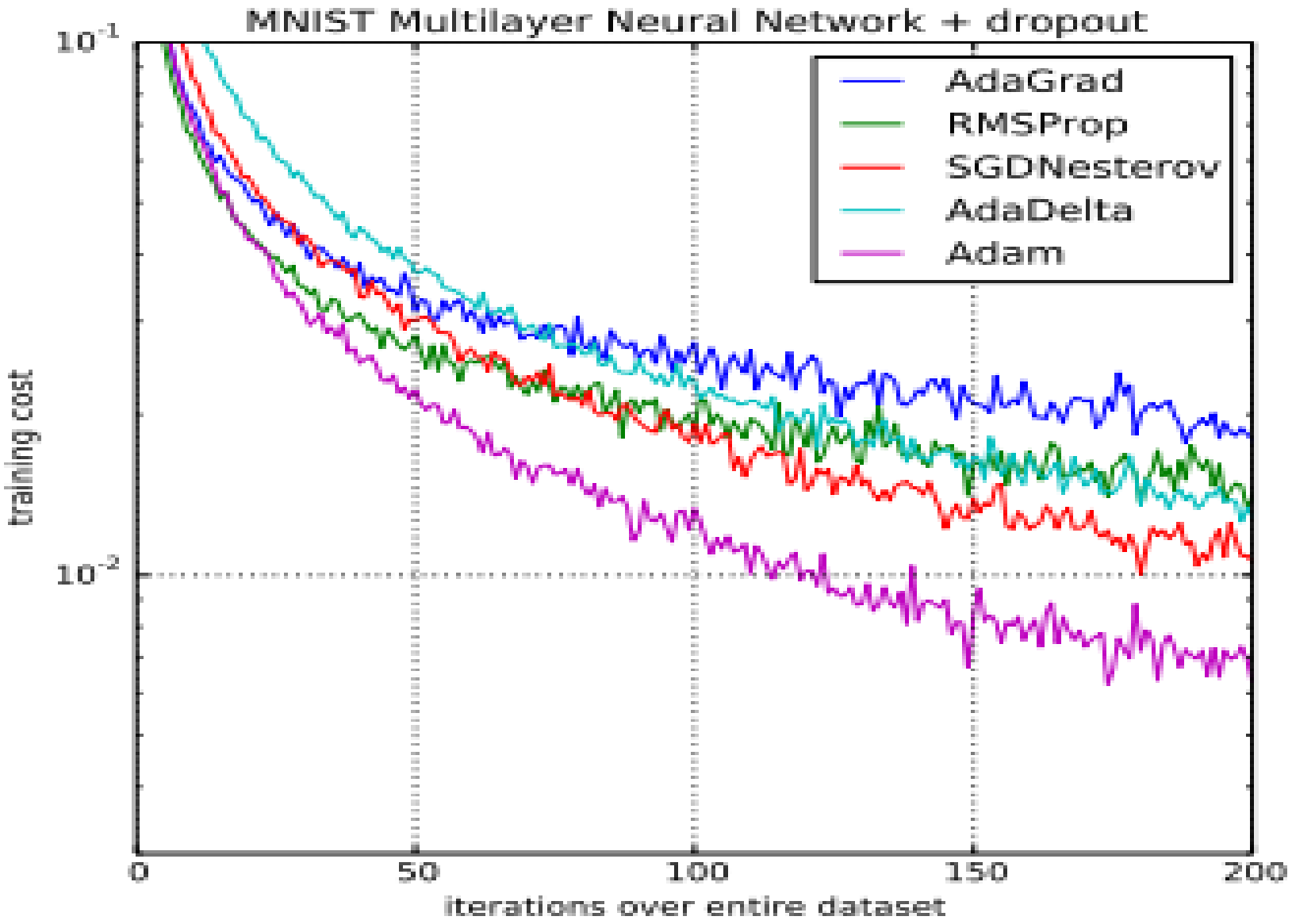
# 경사하강법(Gradient Descent)

**옵티마이저(Optimizer)는 손실함수를 최소화하는 방향으로 가중치를 갱신하는 알고리즘입니다.**




https://www.slideshare.net/yongho/ss-79607172

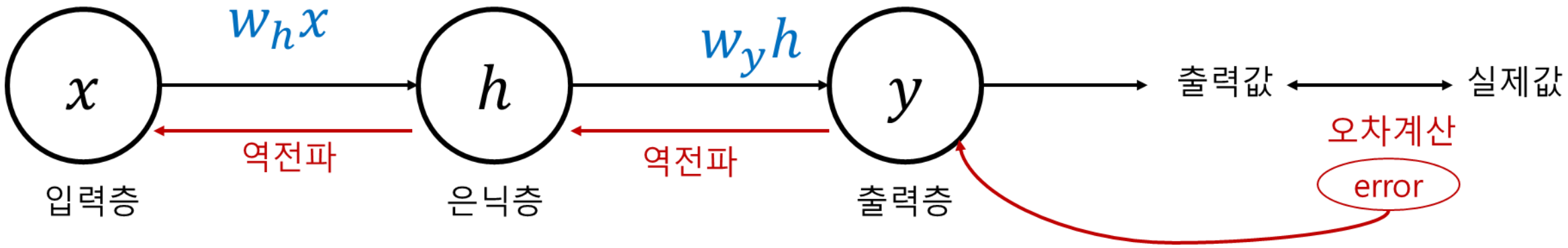
# 경사하강법(Gradient Descent)



# 역전파 알고리즘 이해

## ■ 역전파

역전파는 모델 결과값과 실제값의 오차를 구해서, 오차를 입력(input) 방향으로 보내서 가중치를 재업데이트 하는 과정입니다.
역전파 알고리즘은 심층신경망에서 복잡한 비용함수의 편미분을 효율적으로 계산하기 위한 방법입니다.
심층신경망은 전형적으로 고차원 특성공간에서 비롯된 대규모 가중치를 다루어야 하기 때문에 학습하기 어렵습니다.




## ■ 미분 연쇄법칙(Chain Rule)

$$\frac{d}{dx}\left[f\left(g(x)\right)\right] = f'\left(g(x)\right)g'(x)$$

# 역전파 알고리즘


**C(w1,b1,w2,b2,w3,b3)**

**z=wx+b**

**$z^L = w^L a^{L-1} + b^L$**

**$a^L = \sigma(z^L)$**

**$C_0(\ldots) = (a^L - y)^2$**

$$\frac{\partial C_0}{\partial w^L} = \frac{\partial z^L}{\partial w^L}\frac{\partial a^L}{\partial z^L}\frac{\partial C_0}{\partial a^L}$$

네트워크 앞으로 돌아가면서(backward) Gradient를 사용하여 가중치와 편향을 조정하여 마지막 출력층에서 오차벡터(error vector )의 크기를 최소화 할 수 있습니다.

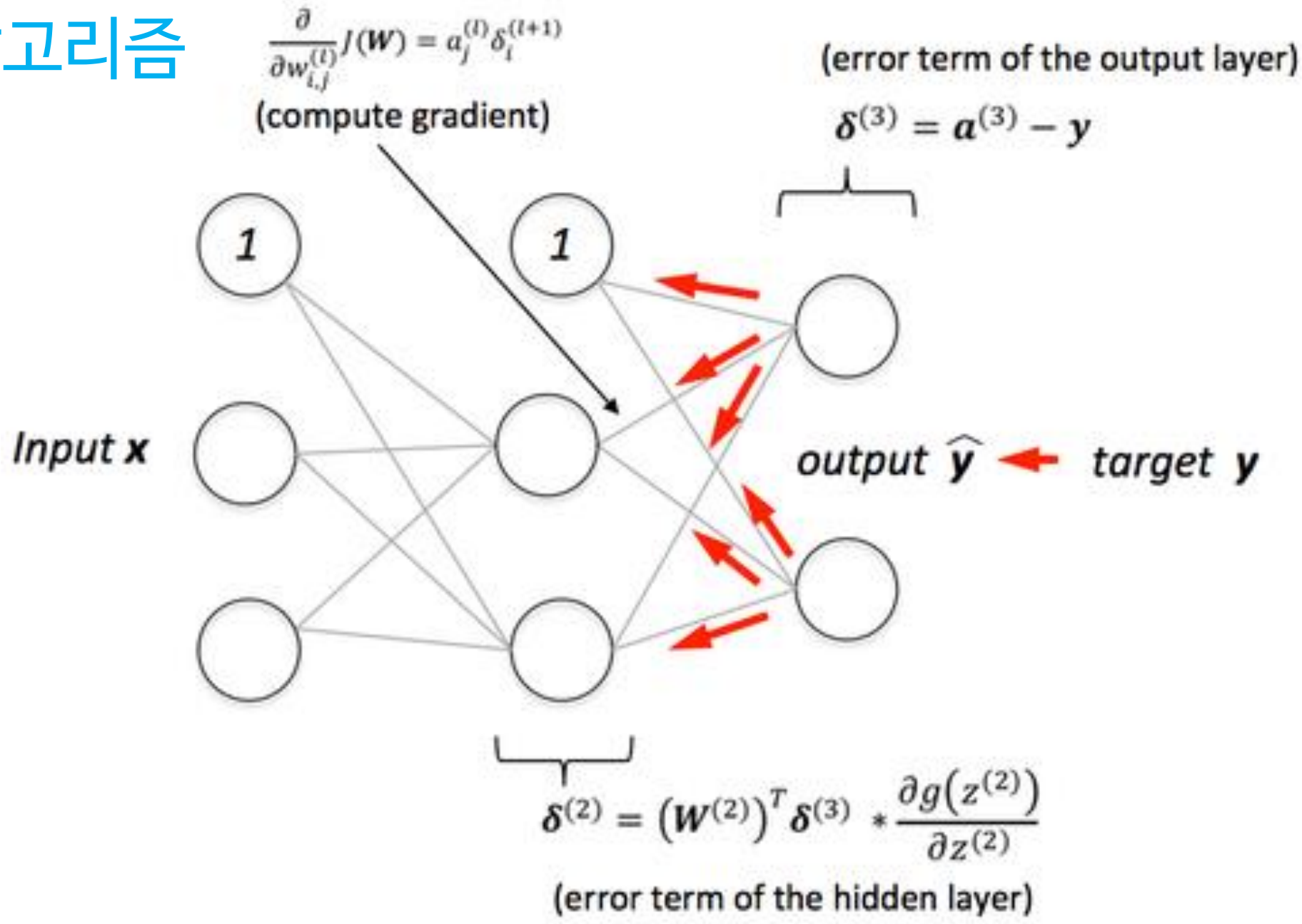
# 역전파 알고리즘

- 역전파(Error Backpropagation)는 인공신경망을 학습시키는 핵심 알고리즘입니다.
- 경사 하강법을 이용해서 파라미터($\theta$)들을 초기화한 initial 값부터 목적 함수(Objective Function) J ($\theta$)를 최소화하는 방향으로 지속적으로 업데이트합니다.

$$\theta = \theta - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

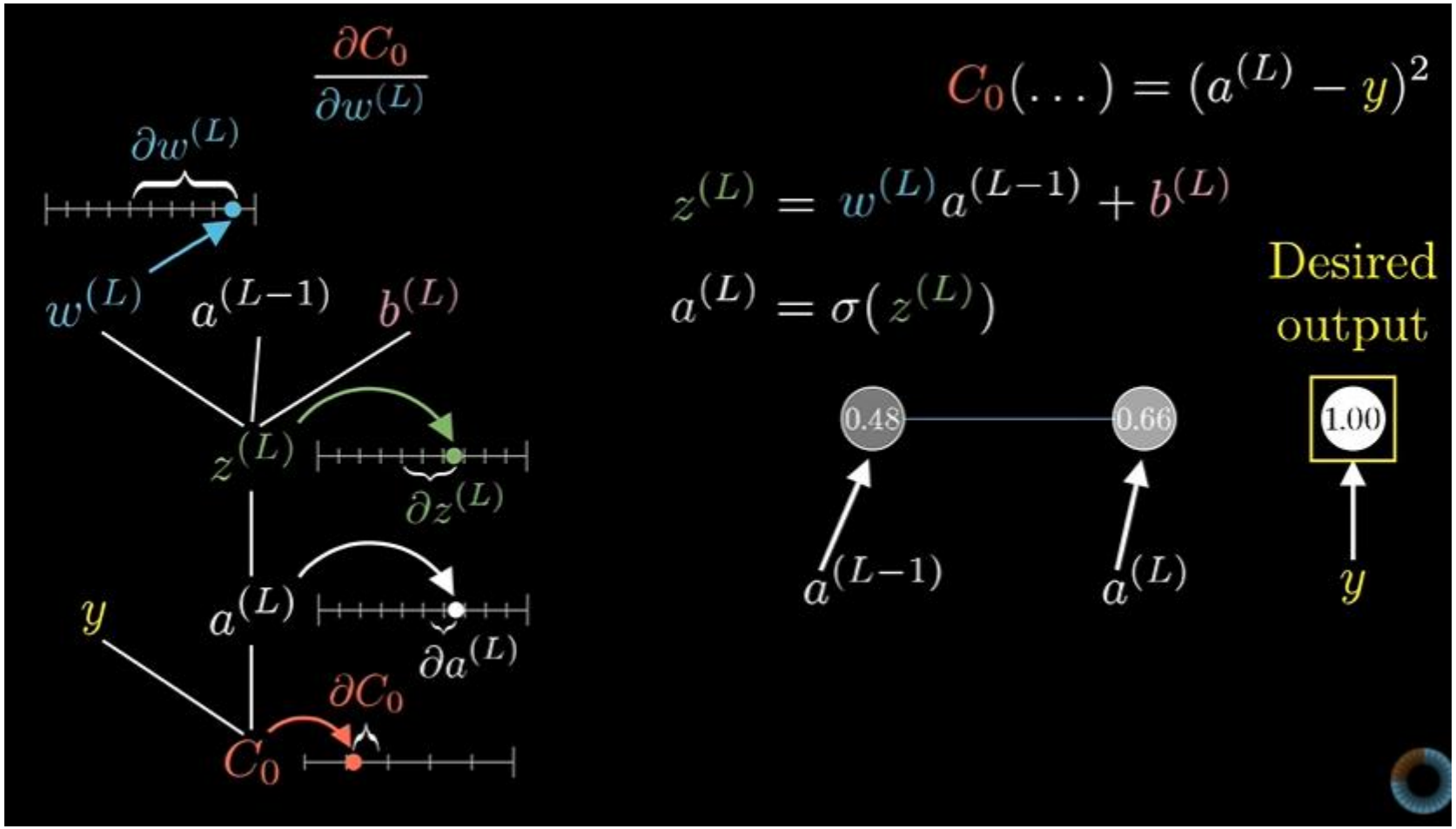
https://sebastianraschka.com/faq/docs/visual-backpropagation.html

# 역전파 알고리즘




https://sebastianraschka.com/faq/docs/visual-backpropagation.html

# 역전파 동작원리




https://www.youtube.com/watch?v=tIeHLnjs5U8

# Thank you

kgpark88@gmail.com