# Deep Learning

Week 2 Session

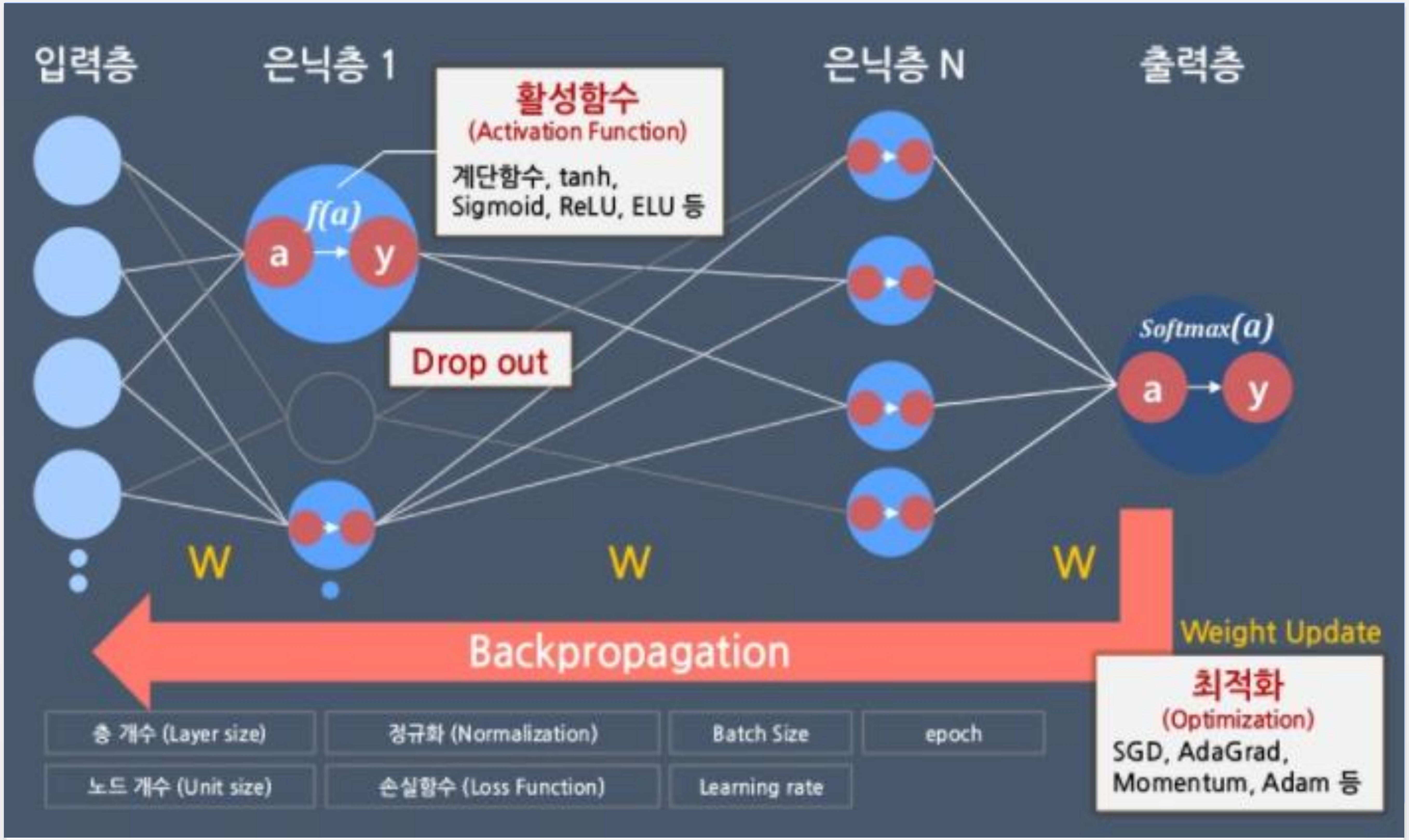
01 Deep Learning Structure

02 Optimization

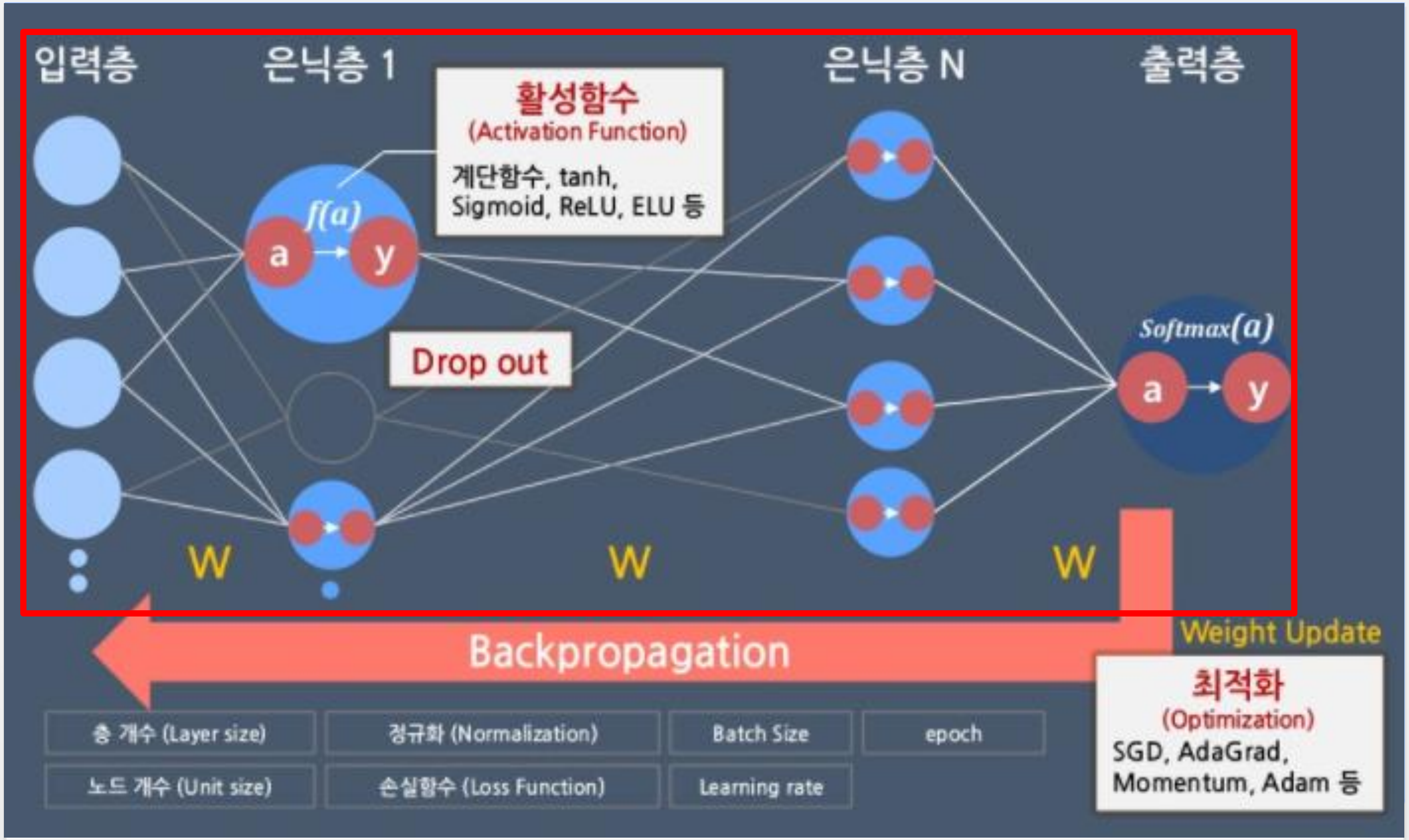
03 BackPropagation

# 01. Deep Learning Structure

# Week 2 & Week 3 내용 요약



# Week 2 & Week 3 내용 요약



# Deep Learning
# Graph Representation

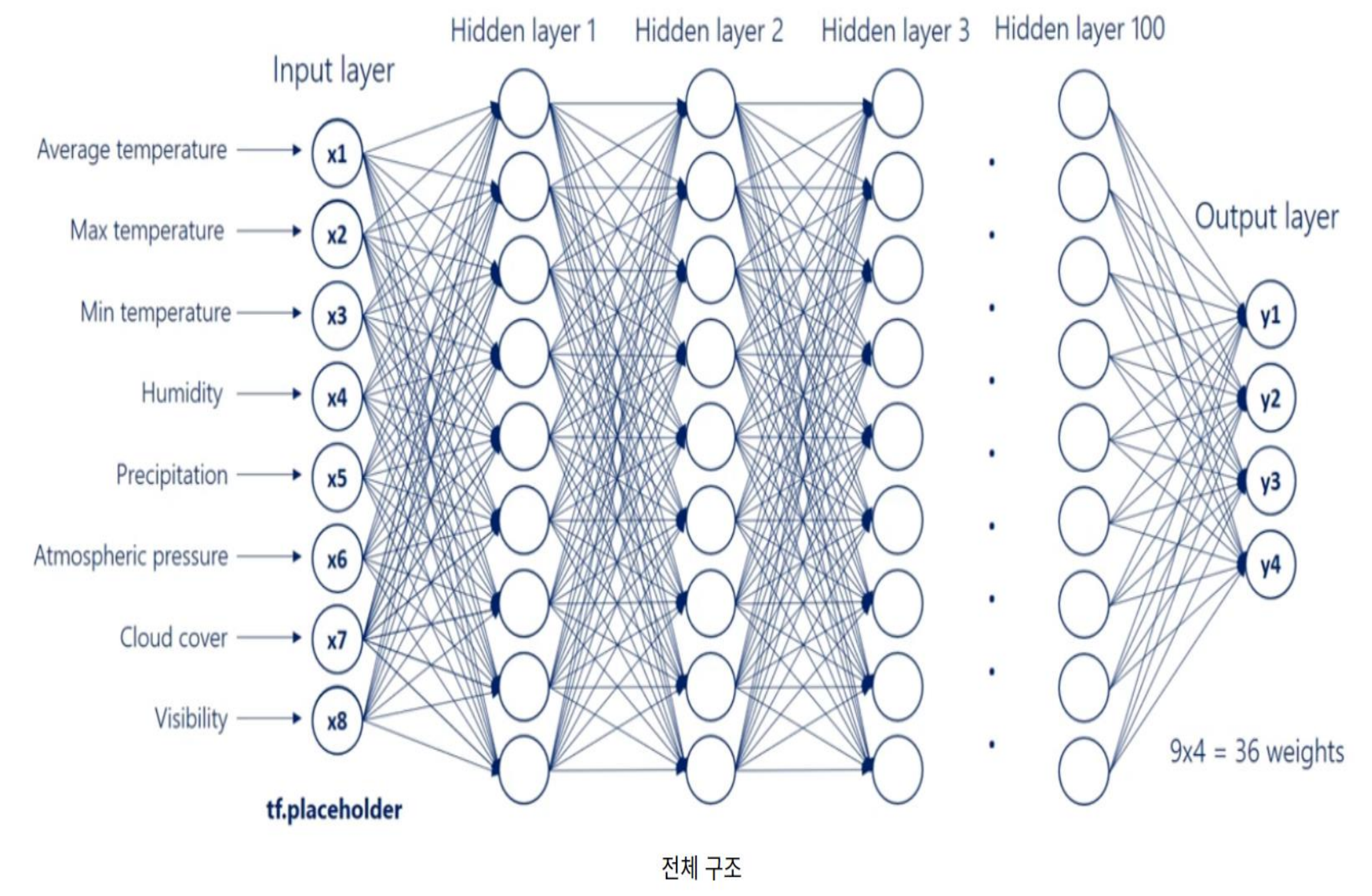
---

Input Layer : 데이터 입력 층
Hidden Layer : 은닉 층
Output Layer : 데이터 출력 층

Connection : weight를 반영

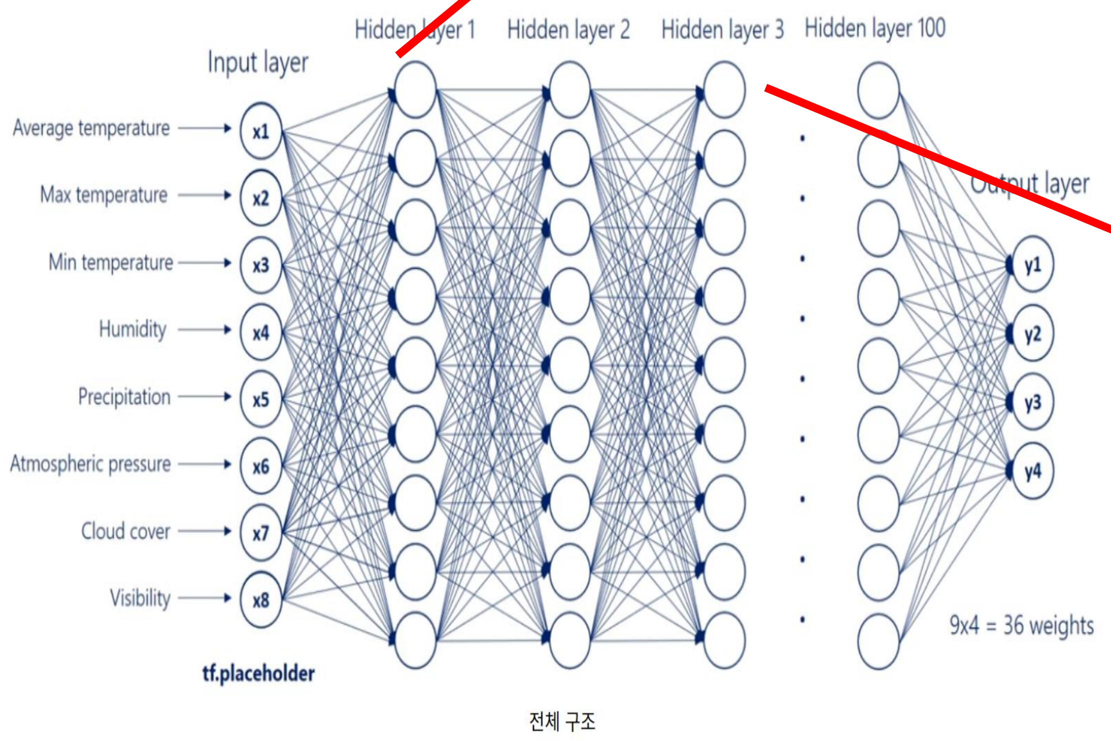
$$f^{(n)}\left(f^{n-1}\left(\ldots\left(f^{2}\left(f^{1}(x)\right)\right)\right)\right) = f(x)$$

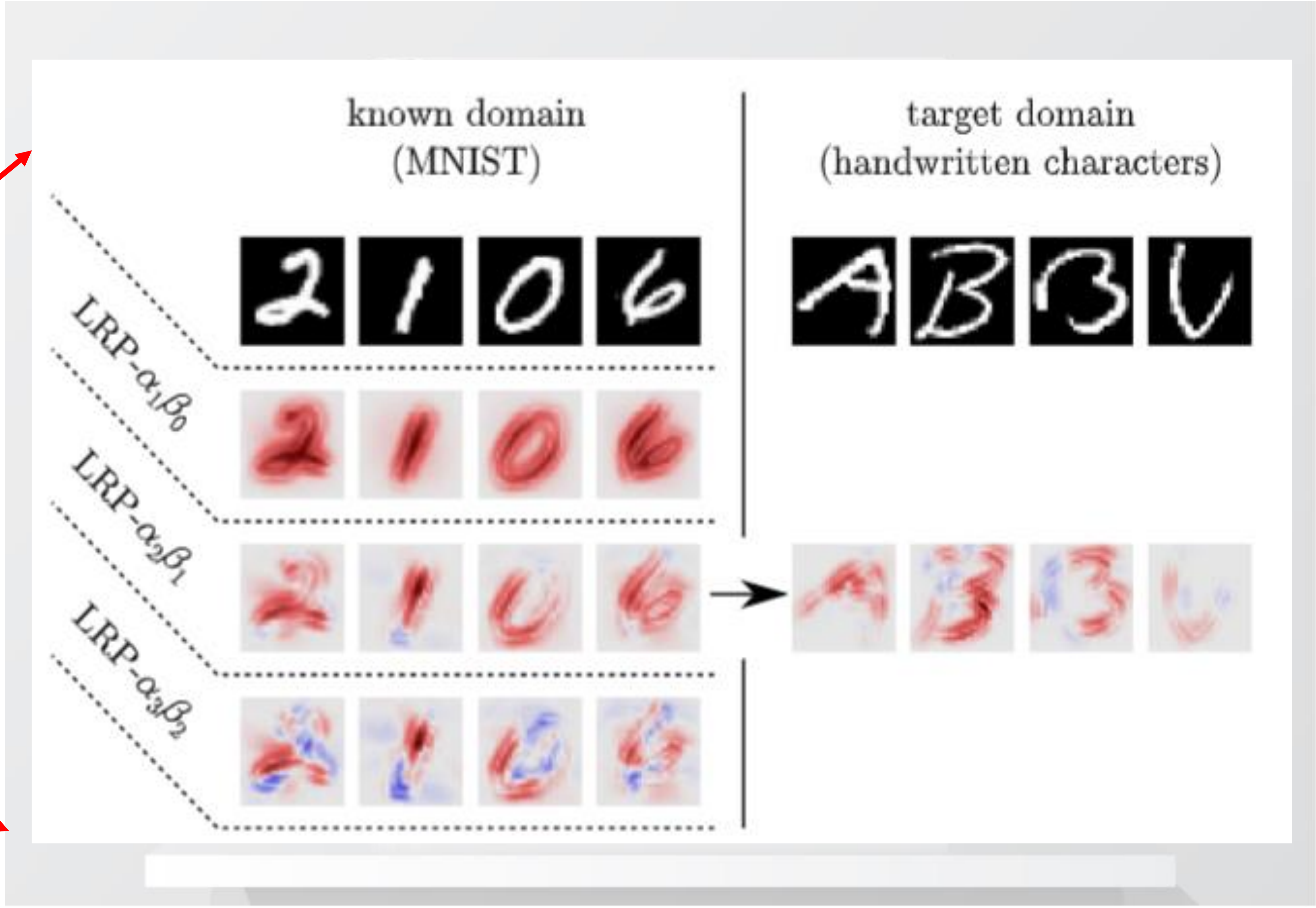
$$y \approx f^{*}(x) = \hat{y}$$



Neuron Importance
Neuron Interpretation

Deep Learning
Graph Representation



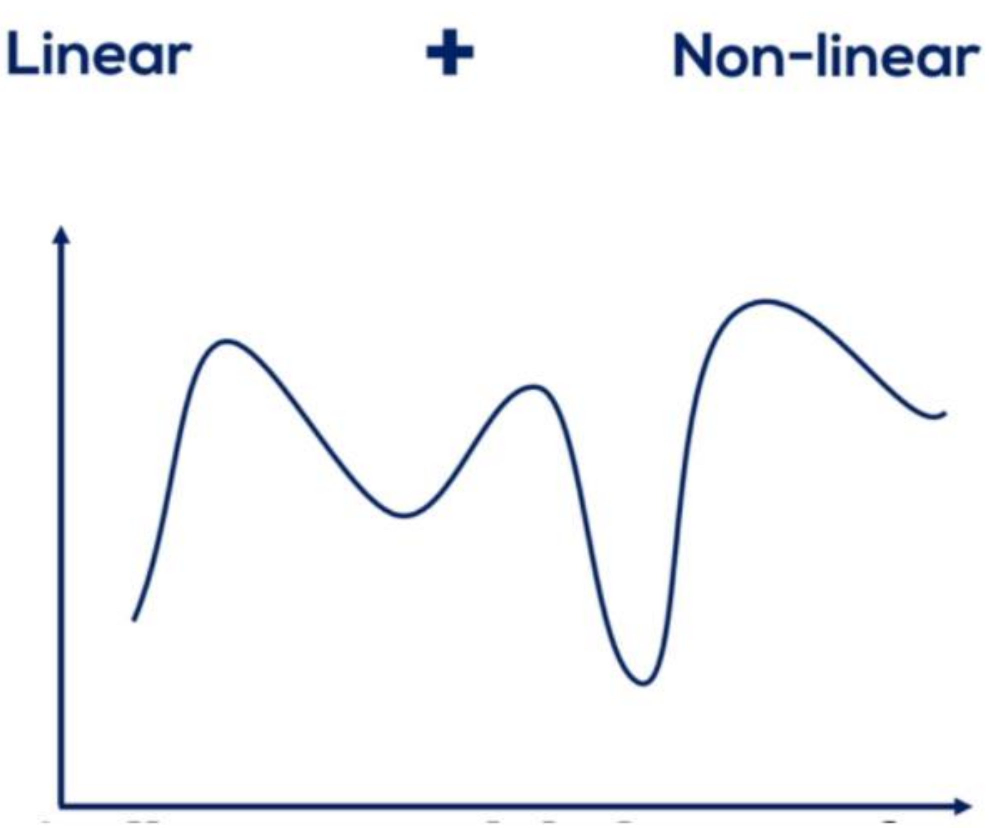

https://www.sciencedirect.com/science/article/pii/S1051200417302385

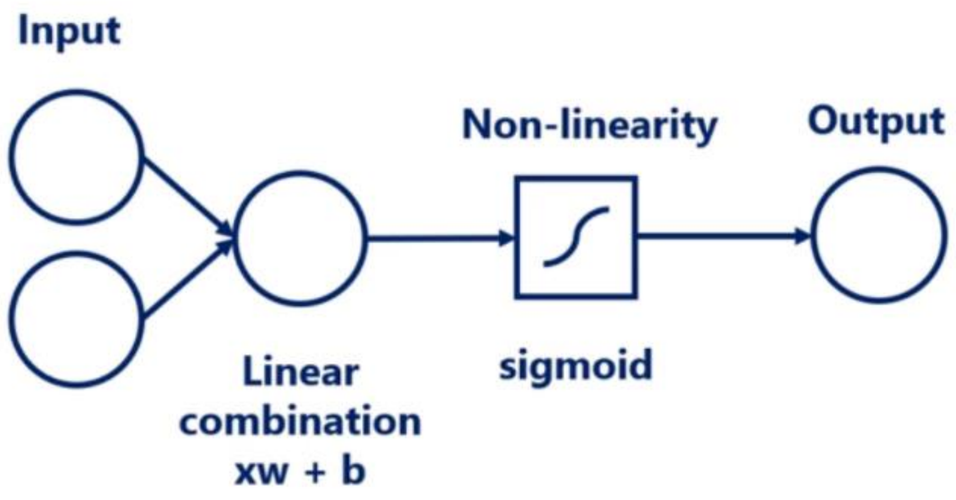
# 01

## Non-Linearity

$$y = f(x;\theta,w) = \phi(x;\theta)^T w$$

Deep Learning : Finding $\phi(x;\theta)$ (linear or non-linear) with proper parameter $\theta$, w




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

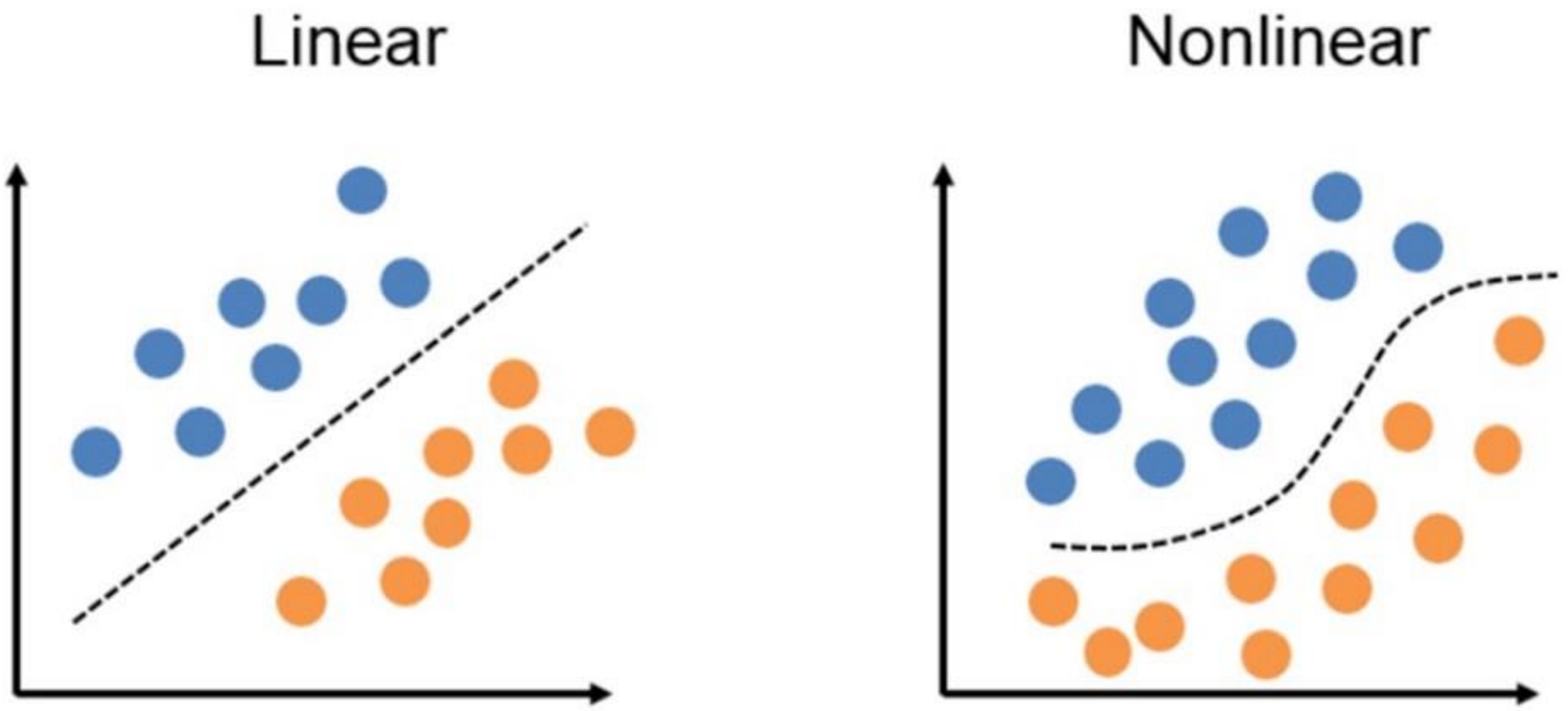
# 01

## Non-Linearity

$y = f(x;\theta,w) = \phi(x;\theta)^T w$

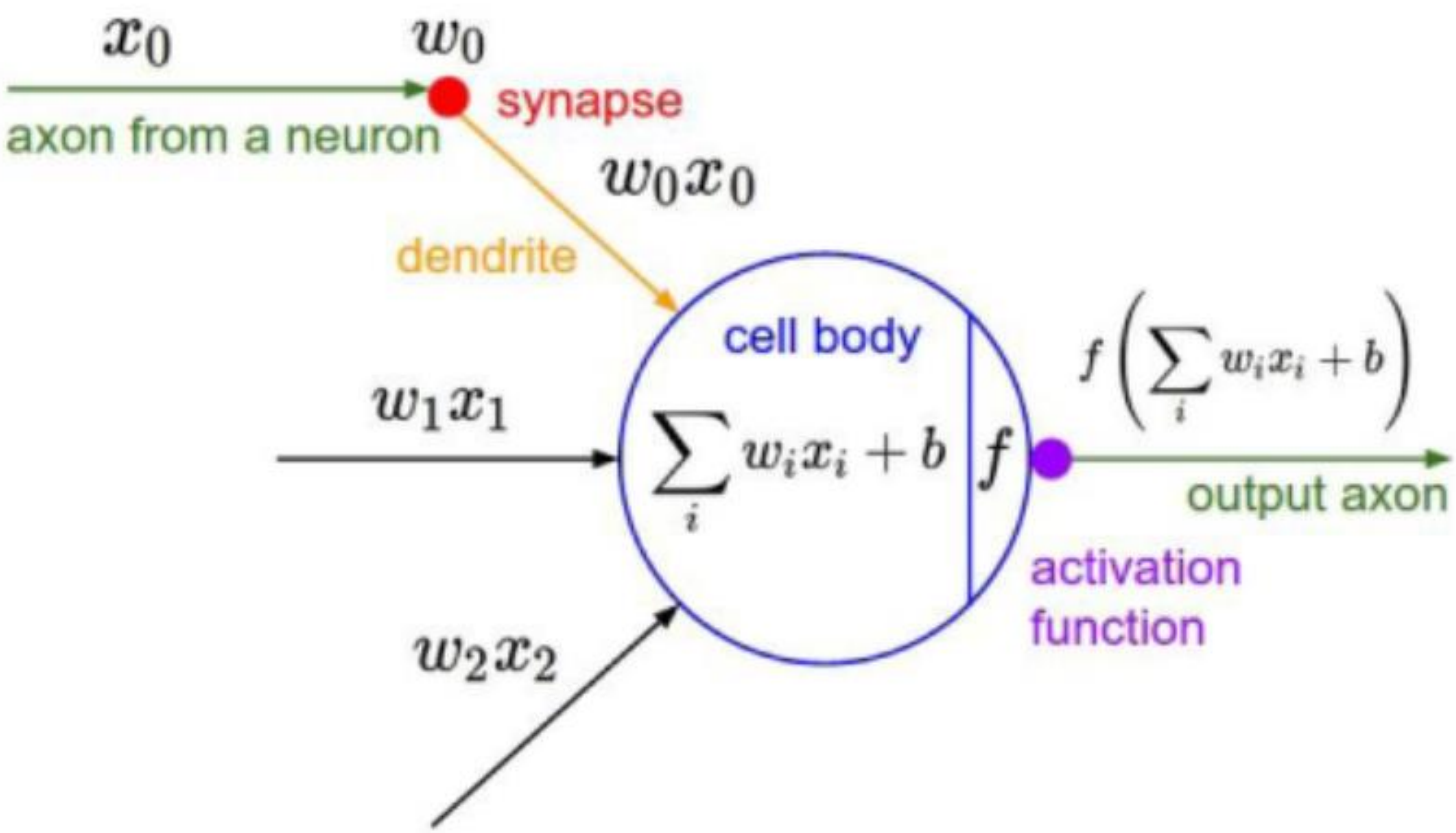
Non-linear $\phi(x;\theta)$

Convexity를 포기!



Hidden Layer Specification

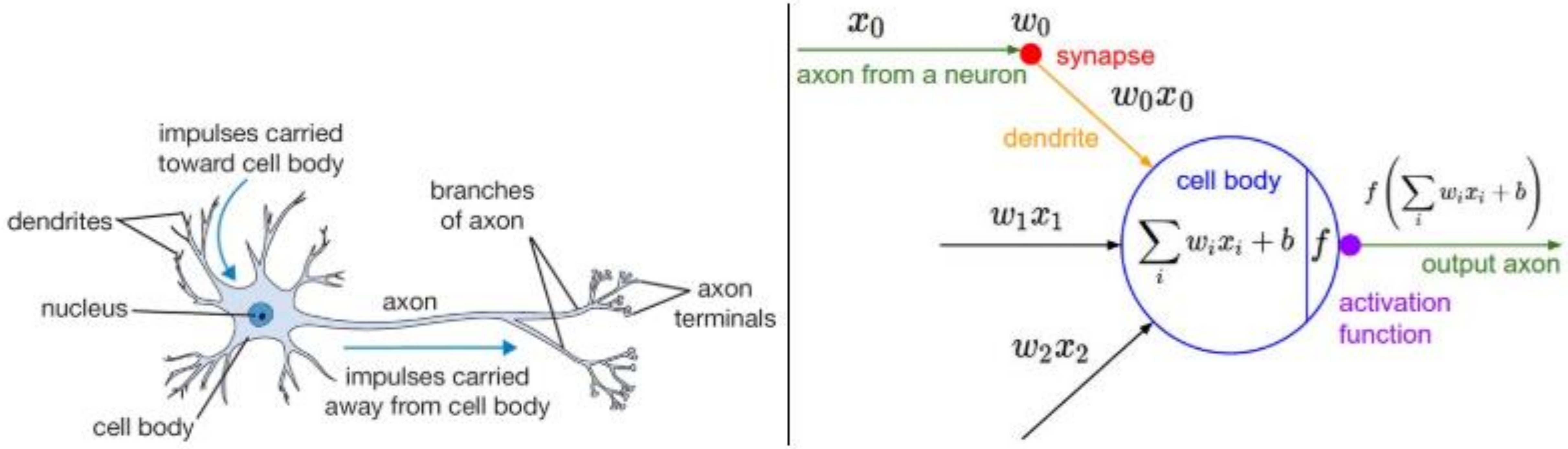
$$x \rightarrow f(W^T x + b)$$



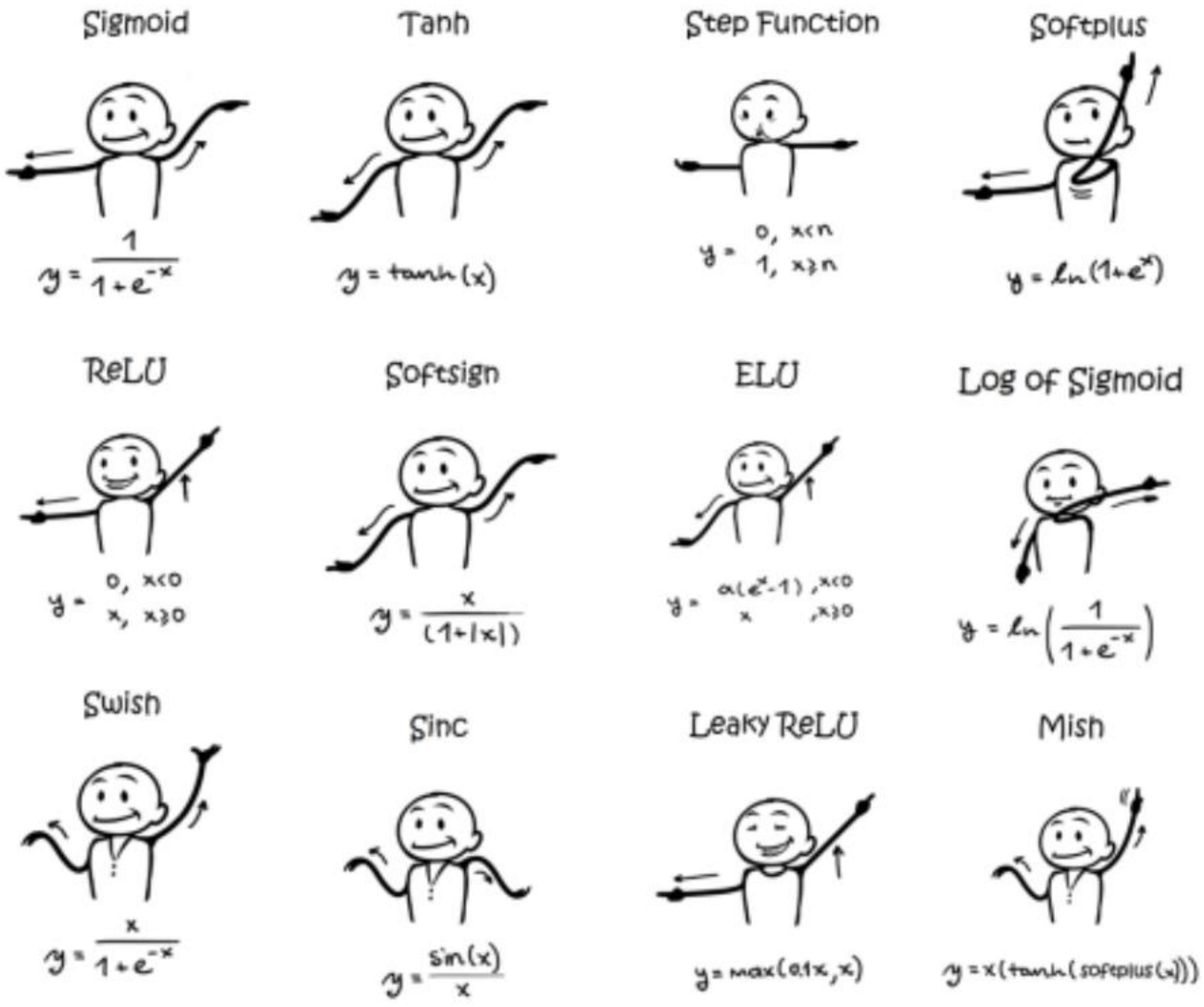
## Activation Function

Neuro-Science 개념에서 착안

Neuron이 다른 Neuron으로 신호를 변환하여 보내는 것과 비슷한 개념



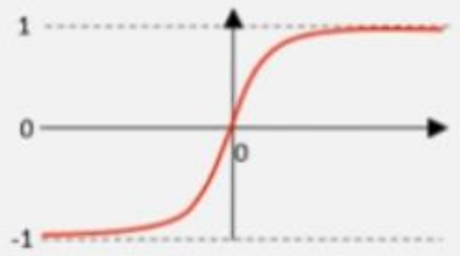
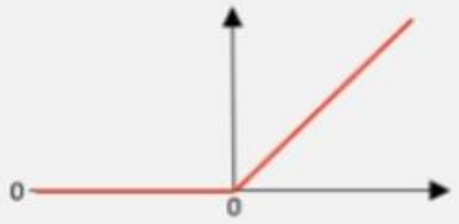

A cartoon drawing of a biological neuron (left) and its mathematical model (right).




# Activation Function

Layer별로 다른 Activation Function

나타내고 싶은 형태에 따라, 상황에 따라 선택!

| Name | Formula | Derivative | Graph | Range |
|---|---|---|---|---|
| **sigmoid (logistic function)** | $\sigma(a) = \frac{1}{1+e^{-a}}$ | $\frac{\partial \sigma(a)}{\partial a} = \sigma(a)(1-\sigma(a))$ |  | (0,1) |
| **TanH (hyperbolic tangent)** | $\tanh(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$ | $\frac{\partial \tanh(a)}{\partial a} = \frac{4}{(e^{a}+e^{-a})^2}$ |  | (-1,1) |
| **ReLu (rectified linear unit)** | relu(a) = max(0,a) | $\frac{\partial \, \mathrm{relu}(a)}{\partial a} = \begin{cases} 0, if\ a \le 0 \\ 1, if\ a > 0 \end{cases}$ |  | (0,∞) |
| **softmax** | $\sigma_i(\boldsymbol{a}) = \frac{e^{a_i}}{\sum_j e^{a_j}}$ | $\frac{\partial \sigma_i(\boldsymbol{a})}{\partial a_j} = \sigma_i(\boldsymbol{a})\left(\delta_{ij} - \sigma_j(\boldsymbol{a})\right)$ Where $\delta_{ij}$ is 1 if i=j, 0 otherwise | different every time | (0,1) |

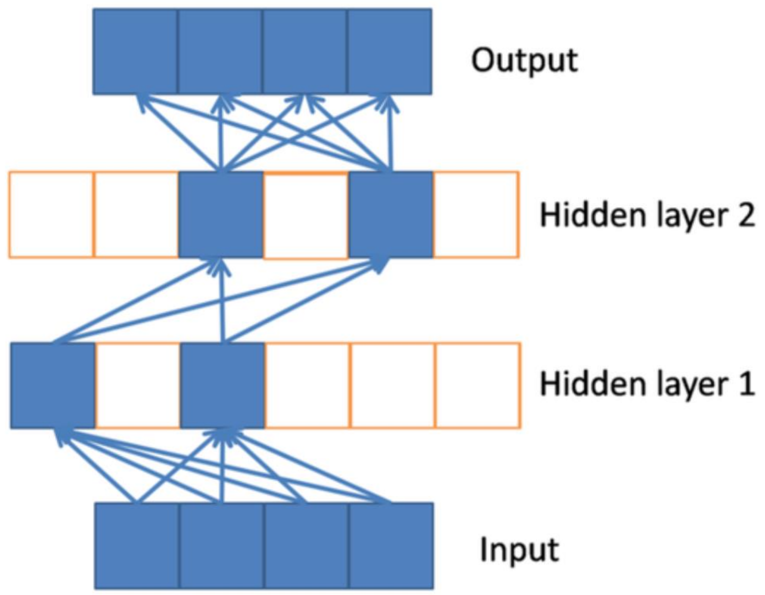
## Activation Function

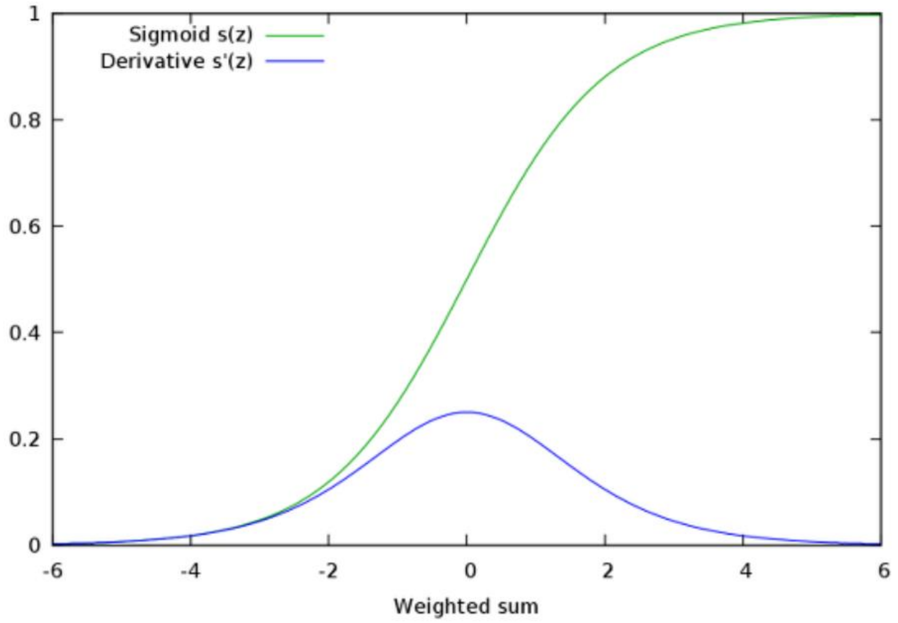
Layer별로 다른 Activation Function

나타내고 싶은 형태에 따라, 상황에 따라 선택!

## Hidden Layer

ReLU를 주로 사용!




1. Sparsity
(computationally efficient)
(Dropout과 비슷한 효과)

2. 미분 계산 빨라짐
(computationally efficient)
(fast convergence)

3. Vanishing Gradient
Exploding Gradient
문제 해결
"Deep" Learning

## Output Layer

출력하려는 Output의 형태에 따라!

Regression : ReLU

Binary Classification : Tanh / Sigmoid

Multiple Classification : Softmax

Layer 1 : $h^{(1)} = g^{(1)}(W^{(1)\,T}x + b^{(1)})$

Layer 2 : $h^{(2)} = g^{(2)}(W^{(2)\,T}h^{(1)} + b^{(2)})$

# Deep Learning Architecture Representation

$$f^{(n)}\left(f^{n-1}\left(\dots\left(f^{2}\left(f^{1}(x)\right)\right)\right)\right) = f(x) = y$$

Layer별로 다른 Activation Function
나타내고 싶은 형태에 따라,
상황에 따라 선택!

**Universal Approximation Theorem** (non-affine activation, arbitrary depth, constrained width). Let $\mathcal{X}$ be a compact subset of $\mathbb{R}^d$. Let $\sigma : \mathbb{R} \to \mathbb{R}$ be any non-affine continuous function which is continuously differentiable at at-least one point, with non-zero derivative at that point. Let $\mathcal{N}^{\sigma}_{d,D:d+D+2}$ denote the space of feed-forward neural networks with $d$ input neurons, $D$ output neurons, and an arbitrary number of hidden layers each with $d + D + 2$ neurons, such that every hidden neuron has activation function $\sigma$ and every output neuron has the identity as its activation function, with input layer $\phi$, and output layer $\rho$. Then given any $\varepsilon > 0$ and any $f \in C(\mathcal{X}, \mathbb{R}^D)$, there exists $\hat{f} \in \mathcal{N}^{\sigma}_{d,D:d+D+2}$ such that

$$\sup_{x \in \mathcal{X}} \left\| \hat{f}\,(x) - f(x) \right\| < \varepsilon.$$

In other words, $\mathcal{N}$ is dense in $C(\mathcal{X}; \mathbb{R}^D)$ with respect to the uniform topology.[*disambiguation needed*]

# Deep Learning
# Architecture Function Existence

Following the Universal Approximation Theorem

Deep Learning Architecture Function Implementation

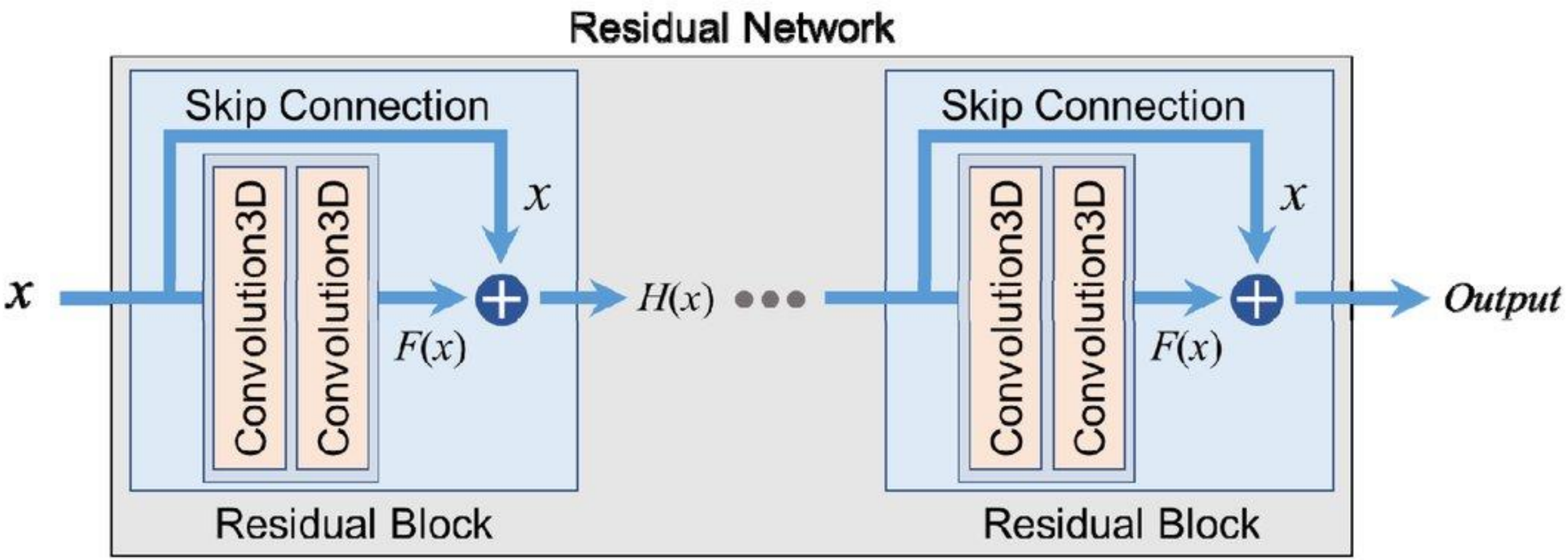
There are many limitations

존재성의 보장은 가능! But…

1. 최적화 알고리즘 => 최적의 파라미터를 찾지 못할수도!

2. Over-fitting : 다른 알고리즘

3. Theorem 자체가 깊이에 관한 조건을 제시해주지 않는다!
=> 효율성의 문제

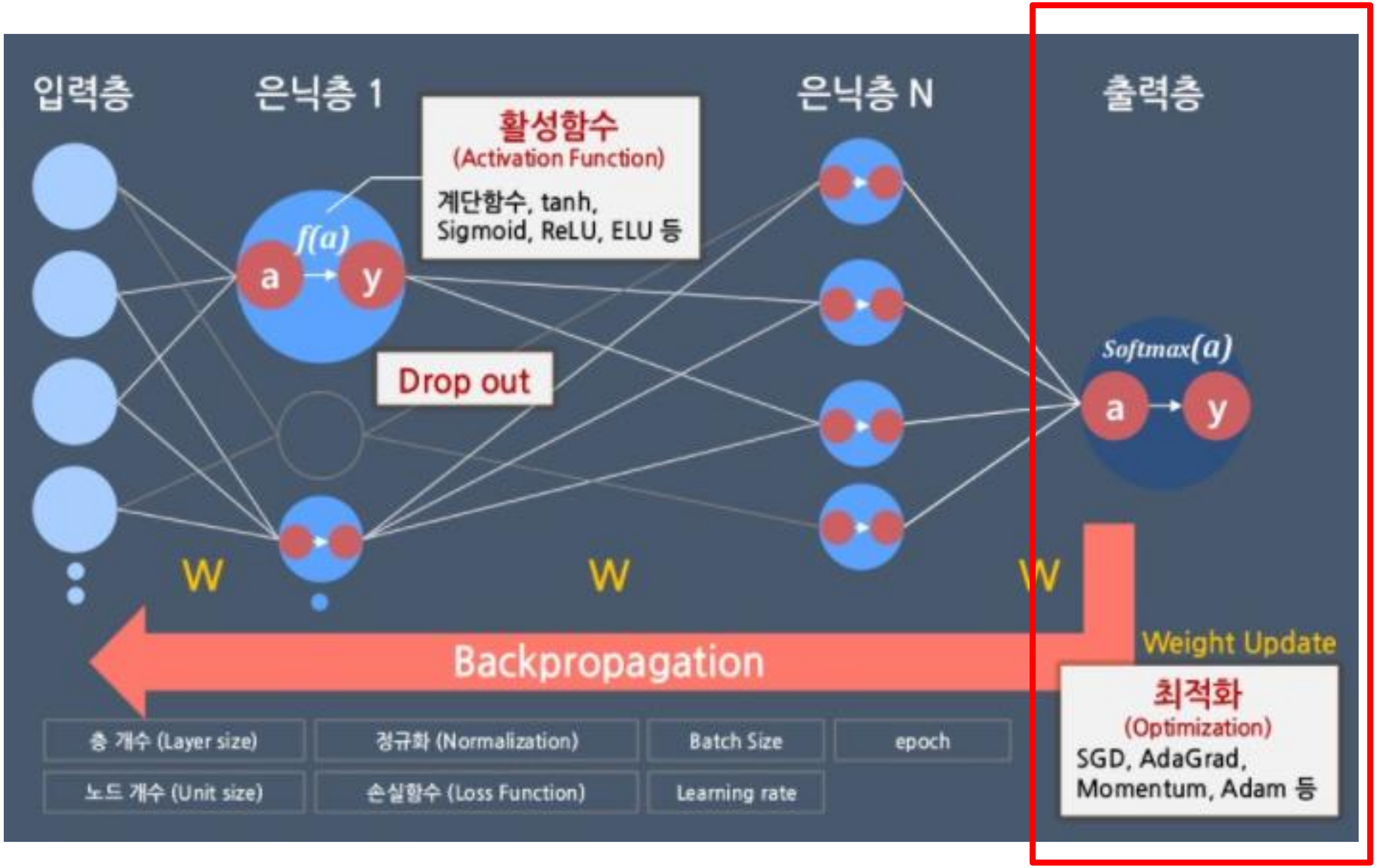



# Deep Learning Architecture

ResNet

Fully connected Layer is not necessary

# 02. Optimization




# Overview

Deep Learning Architecture

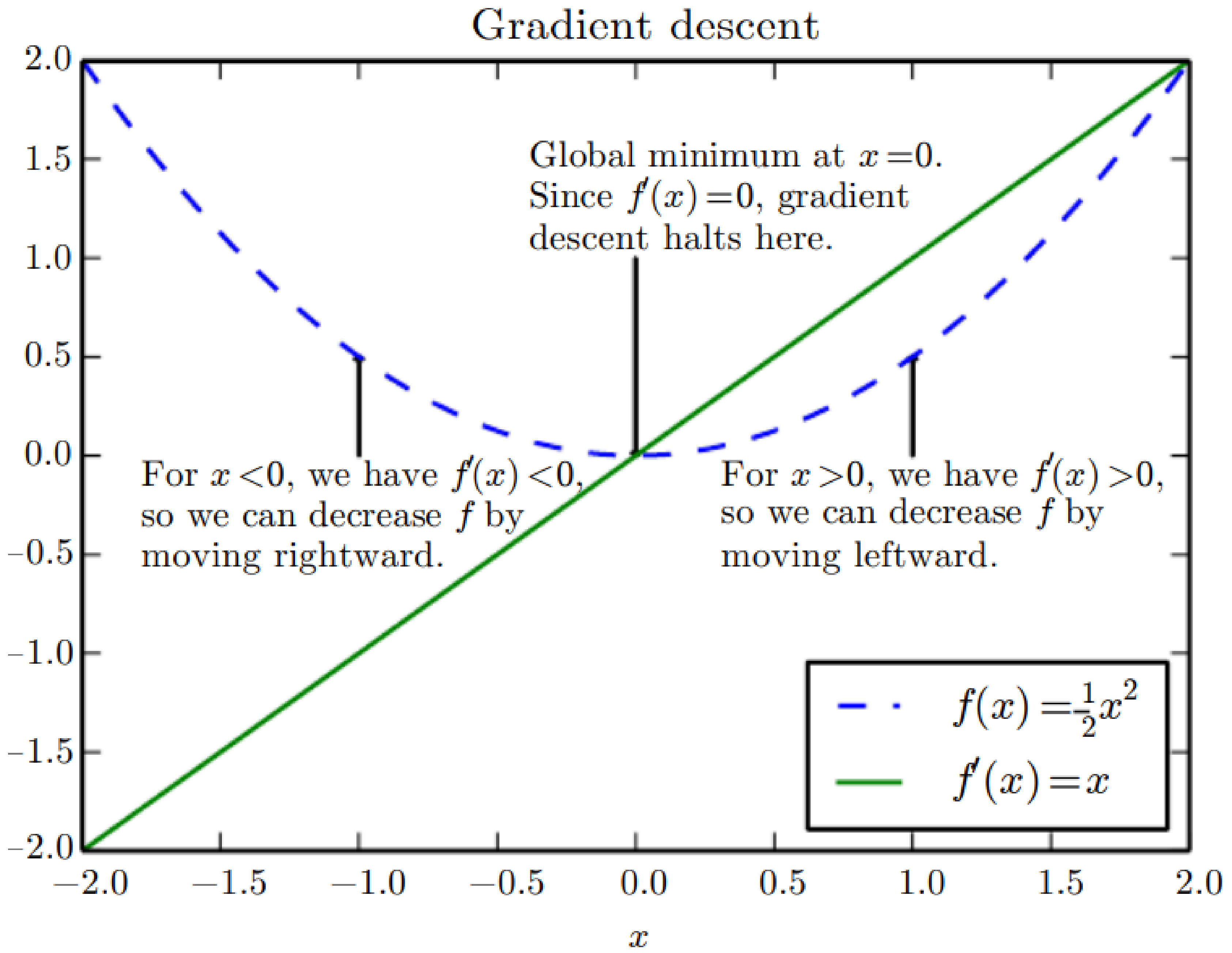
## Second-order Numerical Optimization

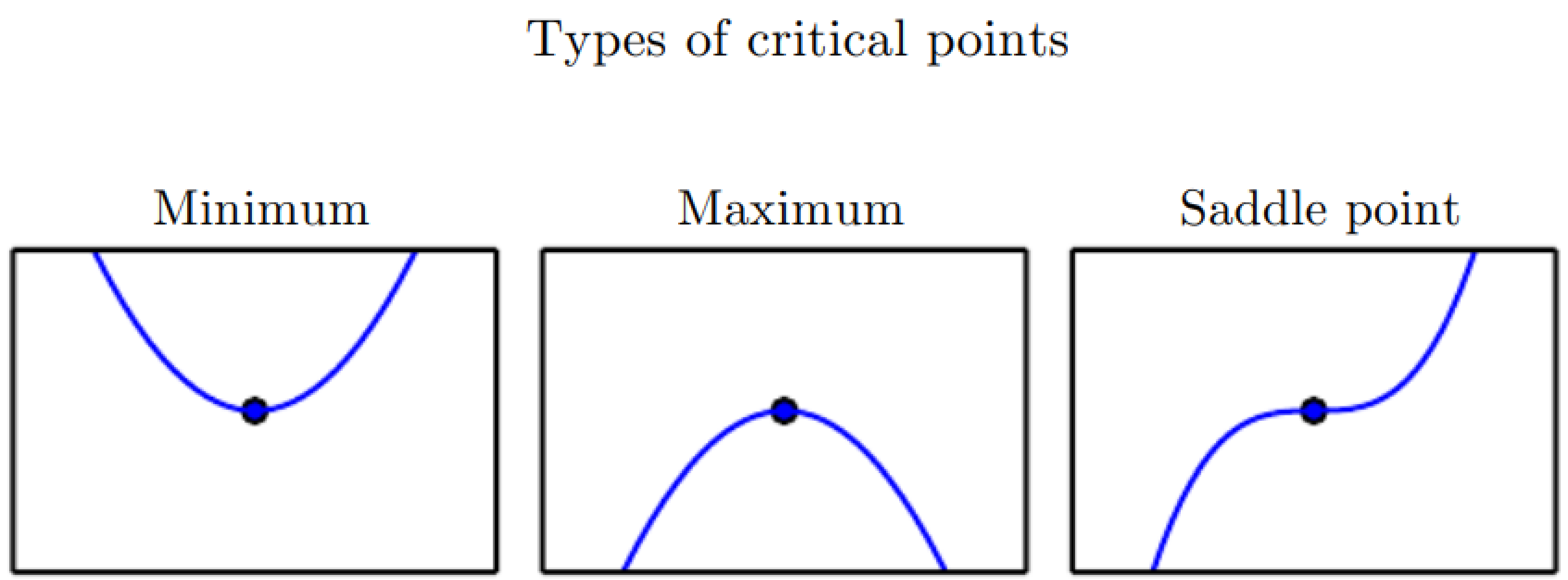
Newton’s method

## First-order Numerical Optimization

Gradient Descent

| 항목 | Newton's method | Gradient descent |
|---|---|---|
| Memory | $O(n^2)$($n \times n$의 hessian matrix) storage | $O(n)$($n$-dimensional gradient) storage |
| Computation | $O(n^3)$ flops($n \times n$의 선형시스템 계산) | $O(n)$ flops($n$-dimensional vector의 선형 결합) |
| Backtracking | $O(n)$ | $O(n)$ |
| Conditioning | Affine invariant 등, conditioning에 크게 영향받지 않음 | 큰 영향을 받을 가능성 존재 |
| Fragility | bugs나 numerical errors에 민감 | newton's method보다 비교적 강건 |

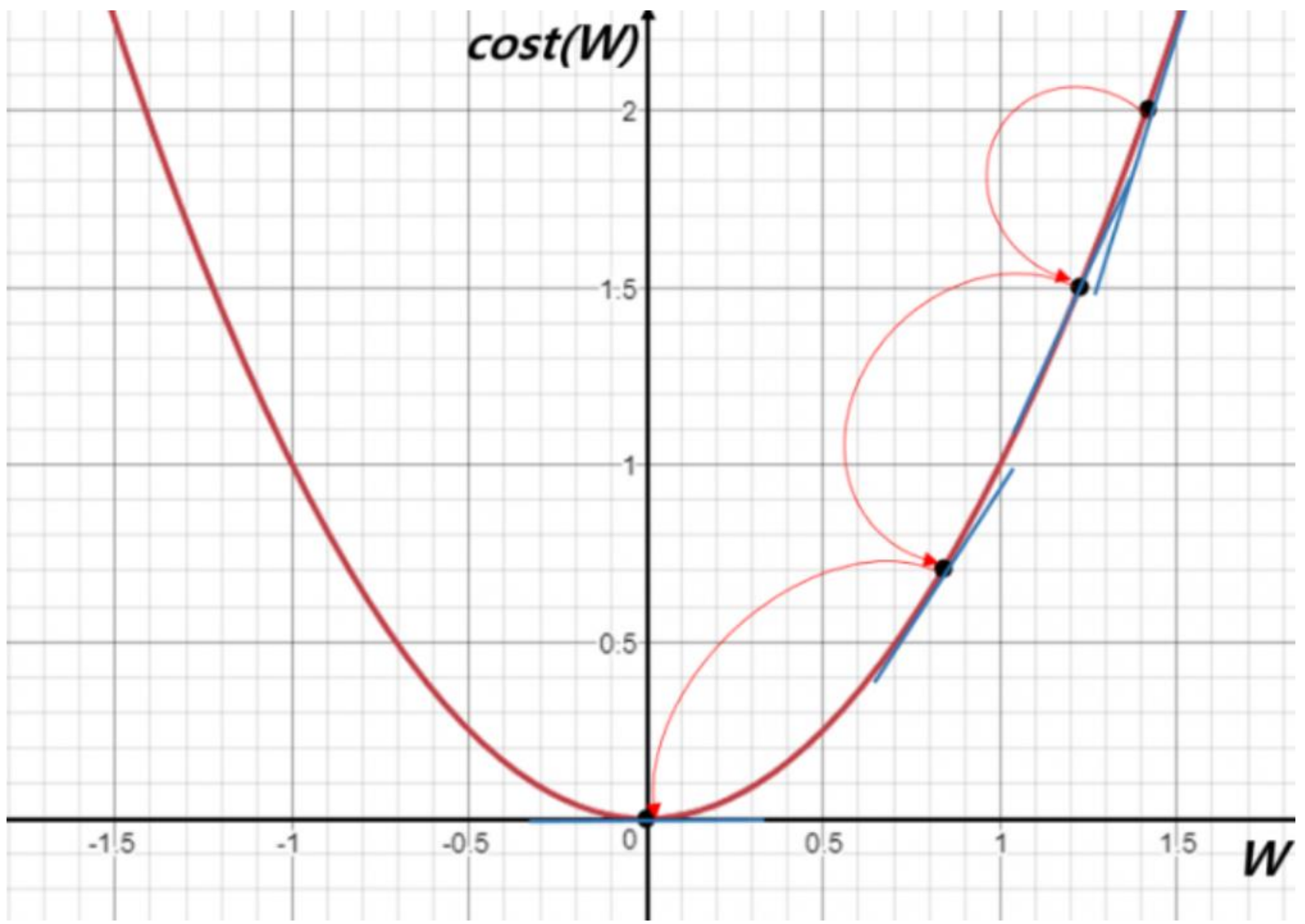



# Gradient Descent

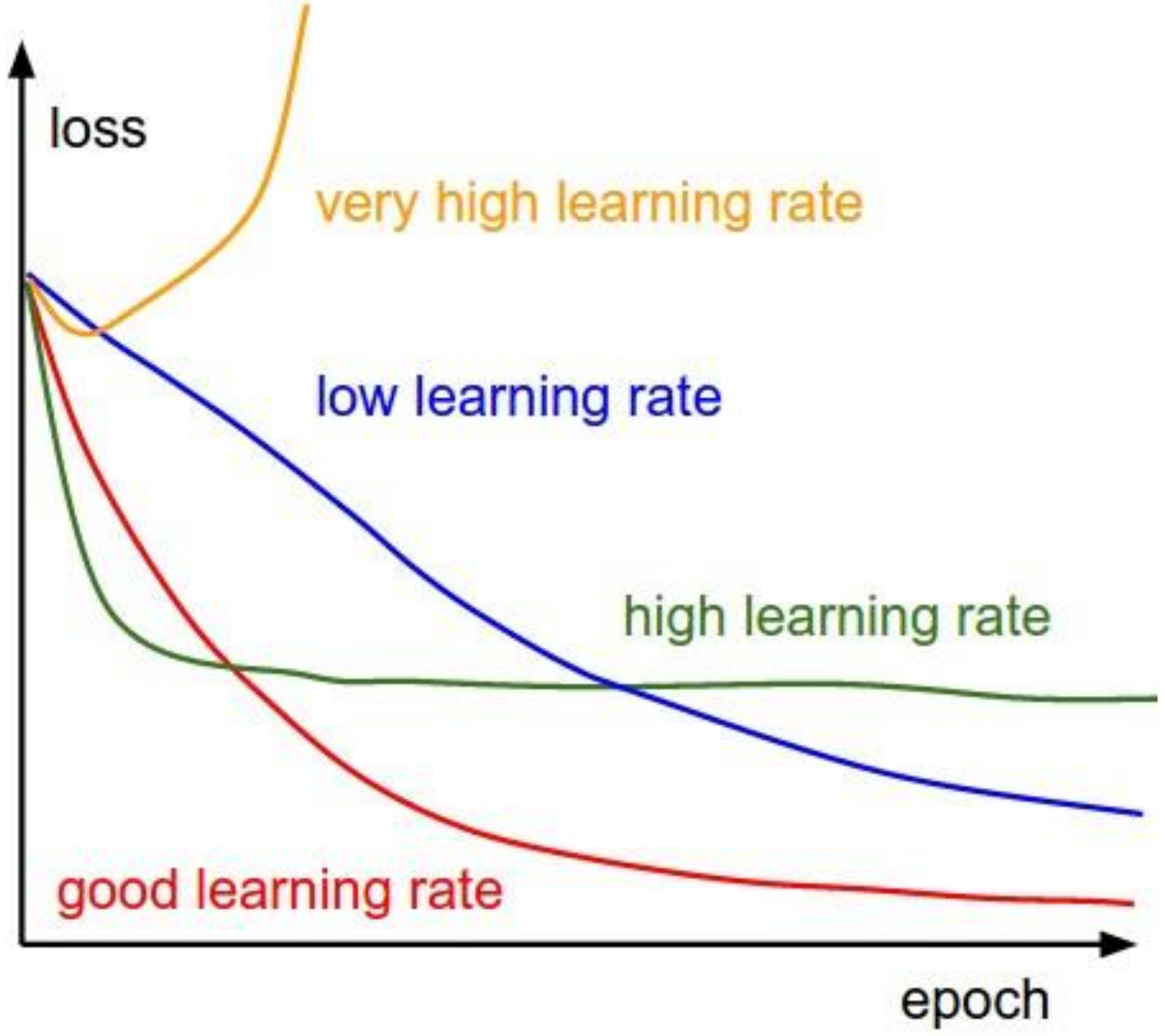
Deep Learning Optimization method

$$\theta = \theta - \eta \cdot \overbrace{\nabla_\theta J(\theta; x, y)}^{\text{Backpropagation}}$$



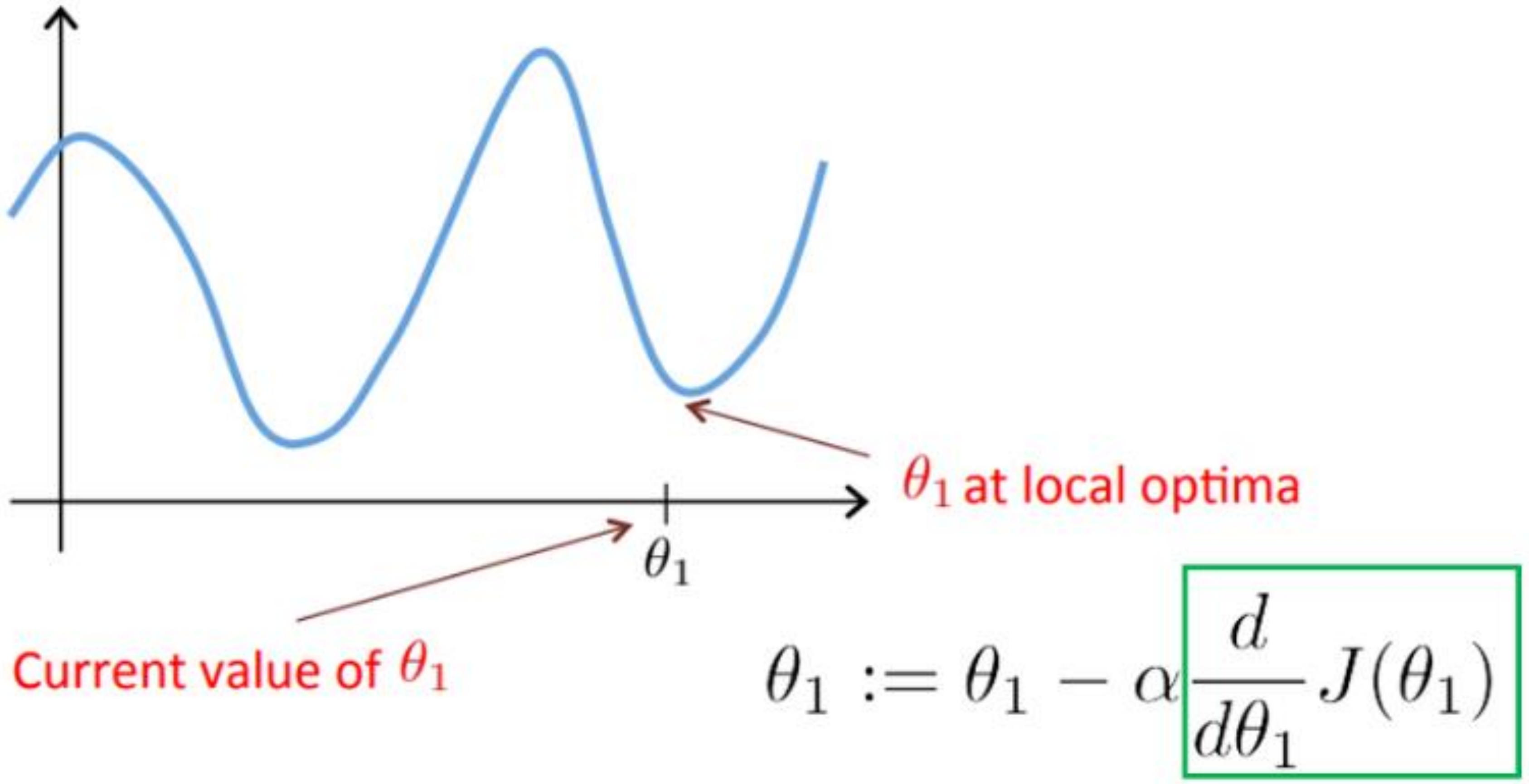
# Gradient descent algorithm

$$\text{repeat until convergence } \{$$
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$
$$(\text{for } j = 1 \text{ and } j = 0)$$
$$\}$$




## Gradient Descent

Deep Learning Optimization method



$$x^{(k)} = x^{(k-1)} - t_k \nabla f(x^{(k-1)}), k = 1, 2, 3, \ldots, t_k > 0$$

## Interpretation

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} \nabla^2 f(x) \parallel y - x \parallel_2^2$$

$$f(y) \approx \underbrace{f(x) + \nabla f(x)^T (y - x)}_{\text{Linear approximation of f}} + \underbrace{\frac{1}{2t} \parallel y - x \parallel_2^2}_{\text{Proximity Term}}$$

1/t를 eigenvalue로 갖는 hessian matrix를 2차항의 계수로 갖는 2차식으로 근사

$$\min_y f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} ||y - x||_2^{\ 2} \longrightarrow x^+ = x - t\nabla f(x) \quad \textit{when } \nabla f(y) = 0$$

$$x^+ = \underset{y}{\operatorname{argmin}}\ f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} \parallel y - x \parallel_2^2$$

## Convergence

*f is convex, differentiable, and Lipschitz continuous*

Gradient Descent

Convergence Analysis

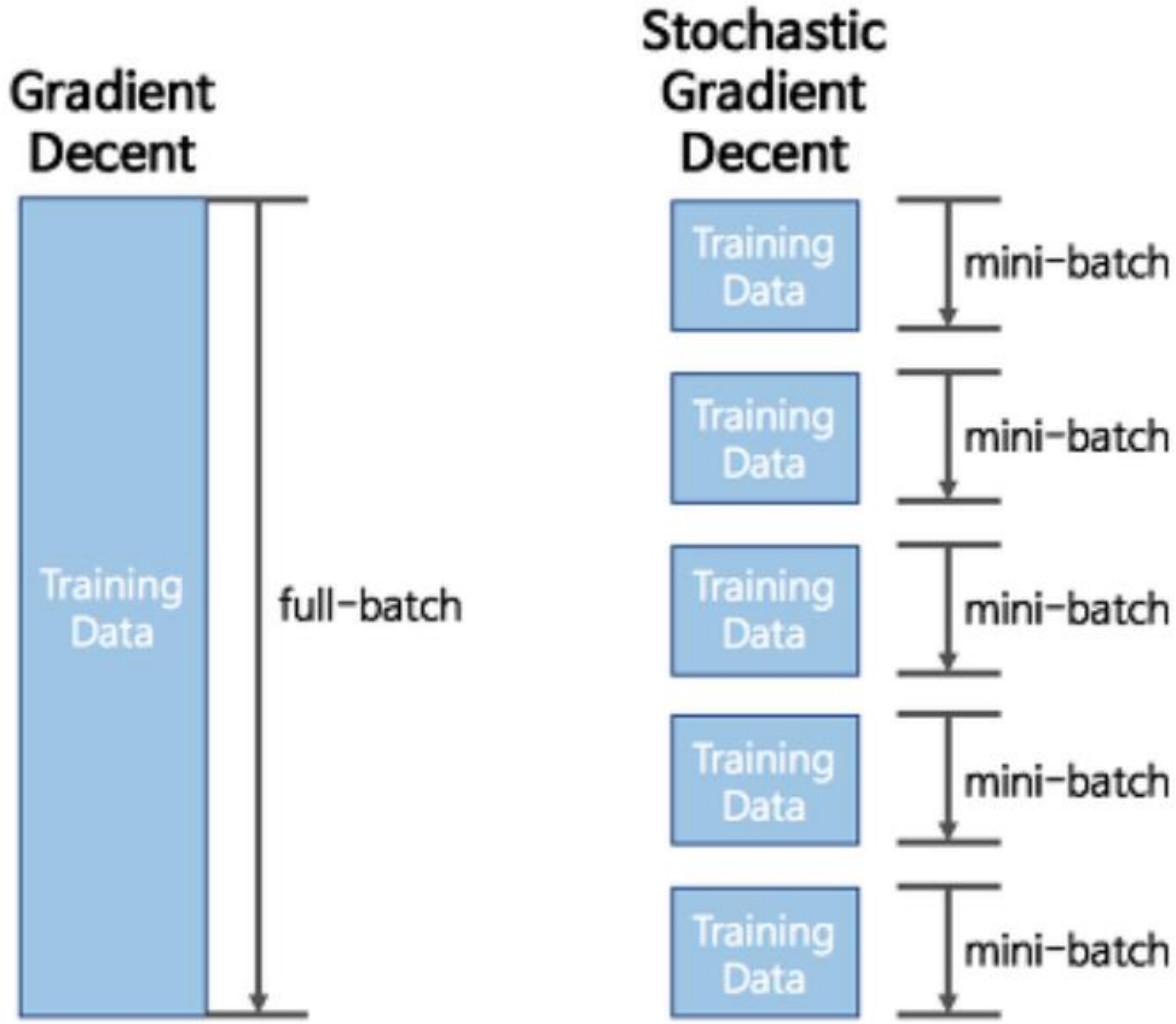
$$f(x^{(k)}) - f^* \le \frac{\|x^{(0)} - x^*\|_2^2}{2tk}$$

fixed step size $t \le 1/L$

convergence rate $O(1/k)$

when $f(x^{(k)}) - f^* \le \epsilon$

$O(1/\epsilon)$ Iterations needed




**Algorithm 8.1** Stochastic gradient descent (SGD) update at training iteration $k$

**Require:** Learning rate $\epsilon_k$.
**Require:** Initial parameter $\boldsymbol{\theta}$
  **while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  **end while**

$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

iteration $\tau$

$$\alpha = \frac{k}{\tau}$$

# Stochastic Gradient Descent

Deep Learning Optimization method

$$\sum_{k=1}^{\infty} \epsilon_k = \infty \qquad \sum_{k=1}^{\infty} \epsilon_k^2 < \infty$$

→ Sufficient Condition for Convergence

SGD Cycle rule : $x^{(k+1)} = x^{(k)} - t_k \cdot \sum_{i=1}^{m} \nabla f_i(x^{(k+i-1)})$

GD Batch : $x^{(k+m)} = x^{(k)} - t_k \cdot \sum_{i=1}^{m} \nabla f_i(x^{(k)})$

Difference : $\sum_{i=1}^{m} [\nabla f_i(x^{(k+i-1)}) - \nabla f_i(x^{(k)})]$

$If\ f_i\ is\ Lipschitz\ continuous,$

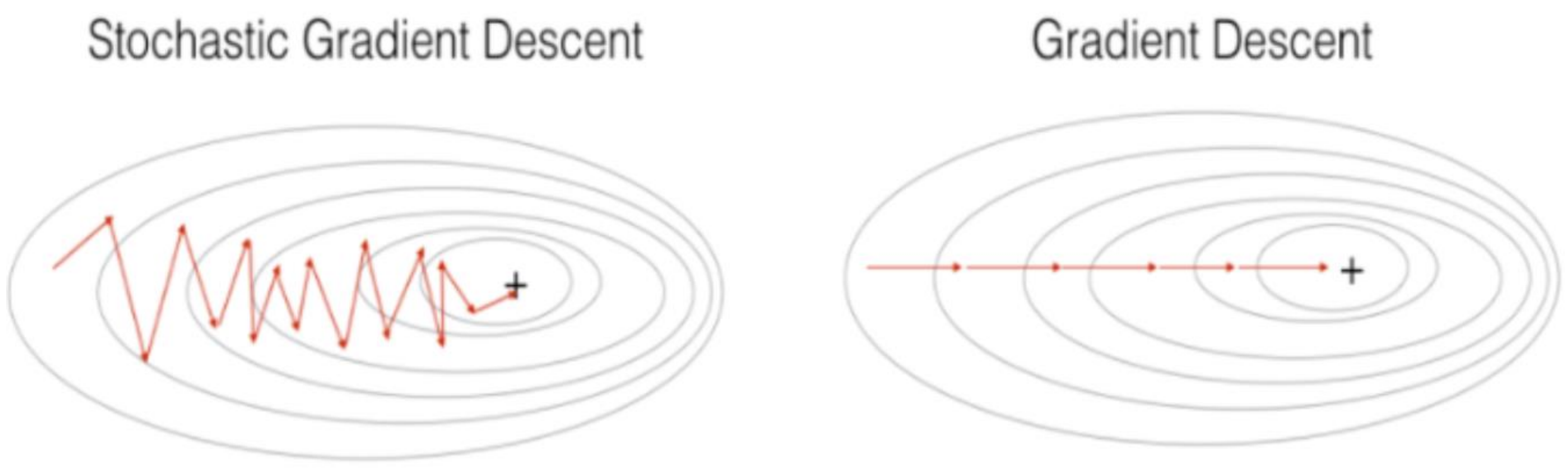
I.e. $f_i$ 가 x에 따라 크게 변하지 않는다면,

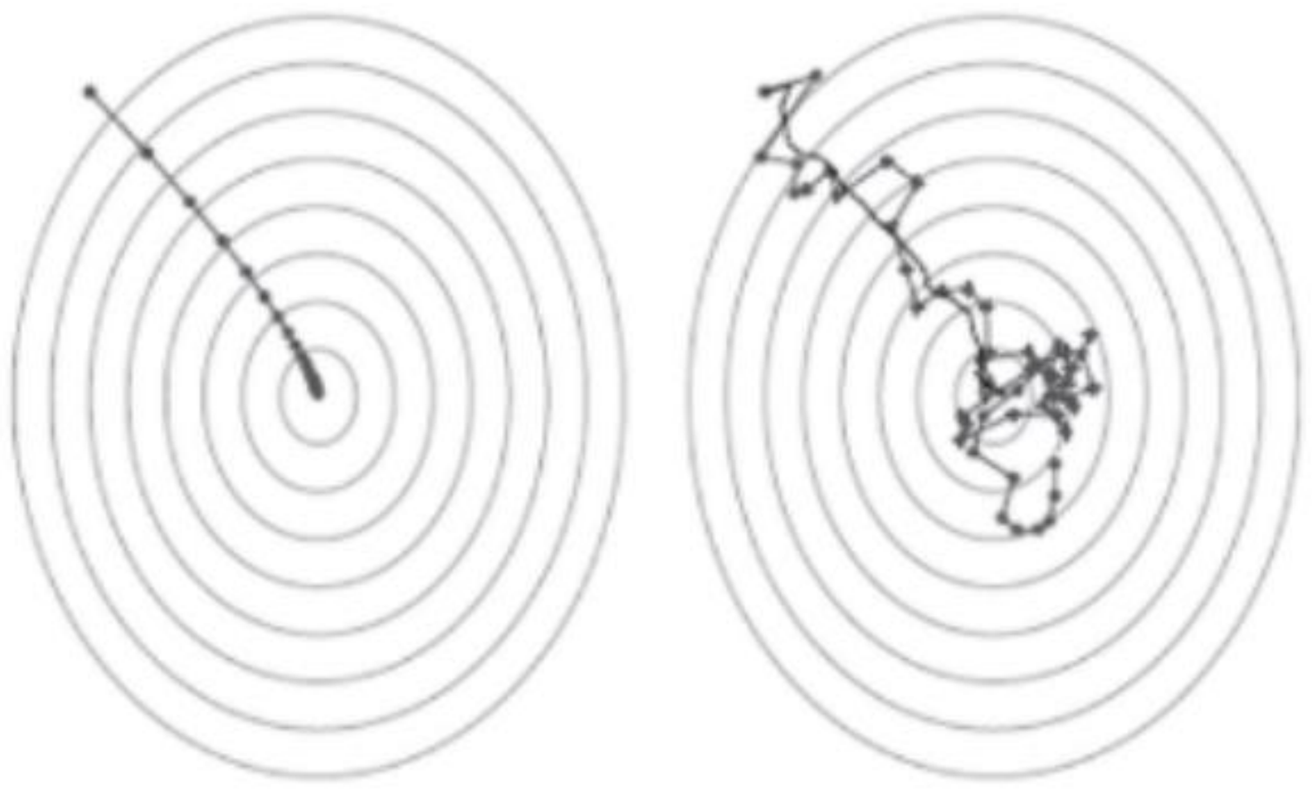
# Stochastic Gradient Descent

Convergence Analysis

**SGD도 GD처럼 Convergence**

경험적으로, SGD는 최적점 근처에 다다랐을 때, 잘 수렴하지 않는 단점!



<BGD, SGD>

# Stochastic Gradient Descent

Deep Learning Optimization method

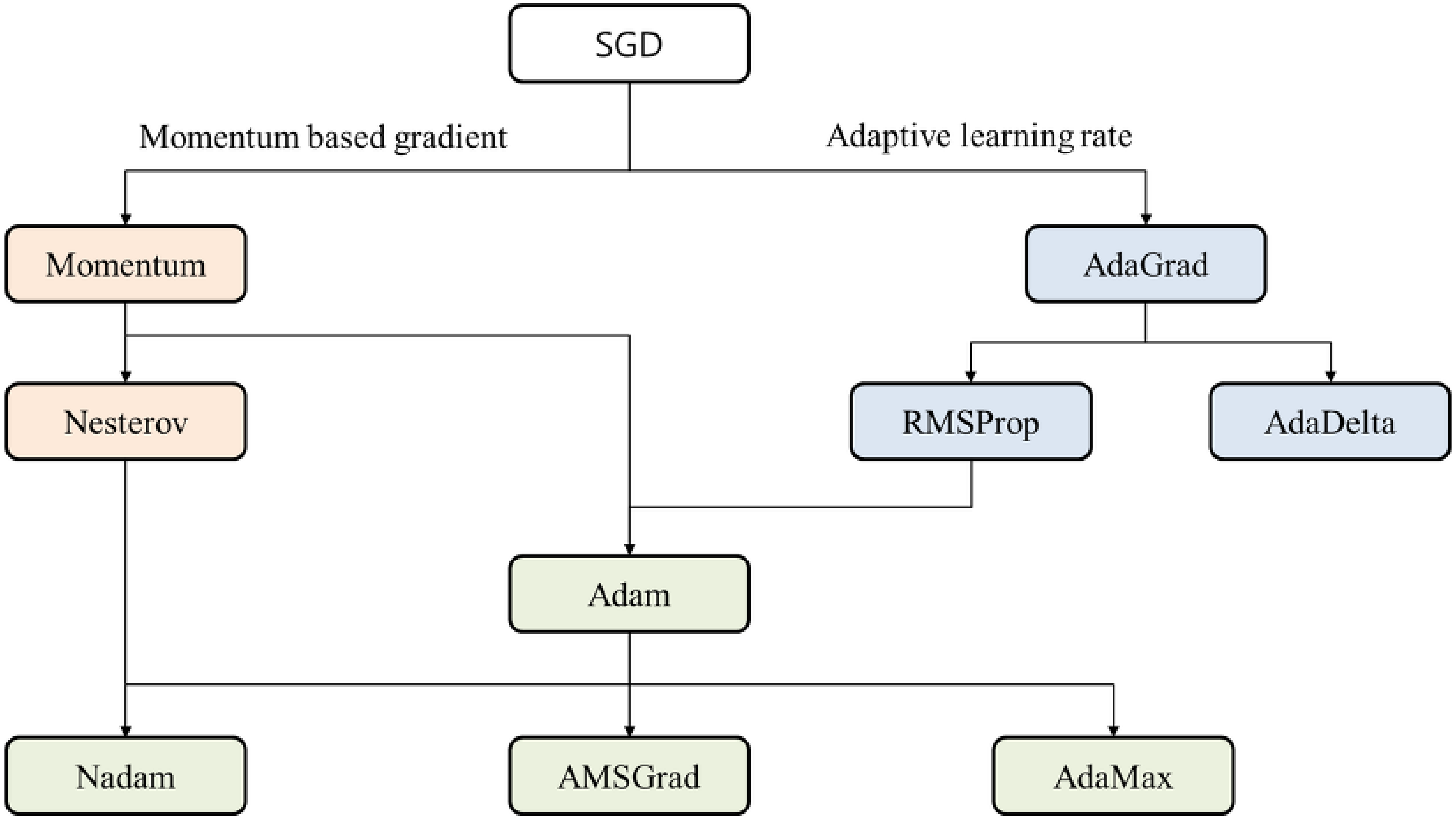
## SGD 장/단점

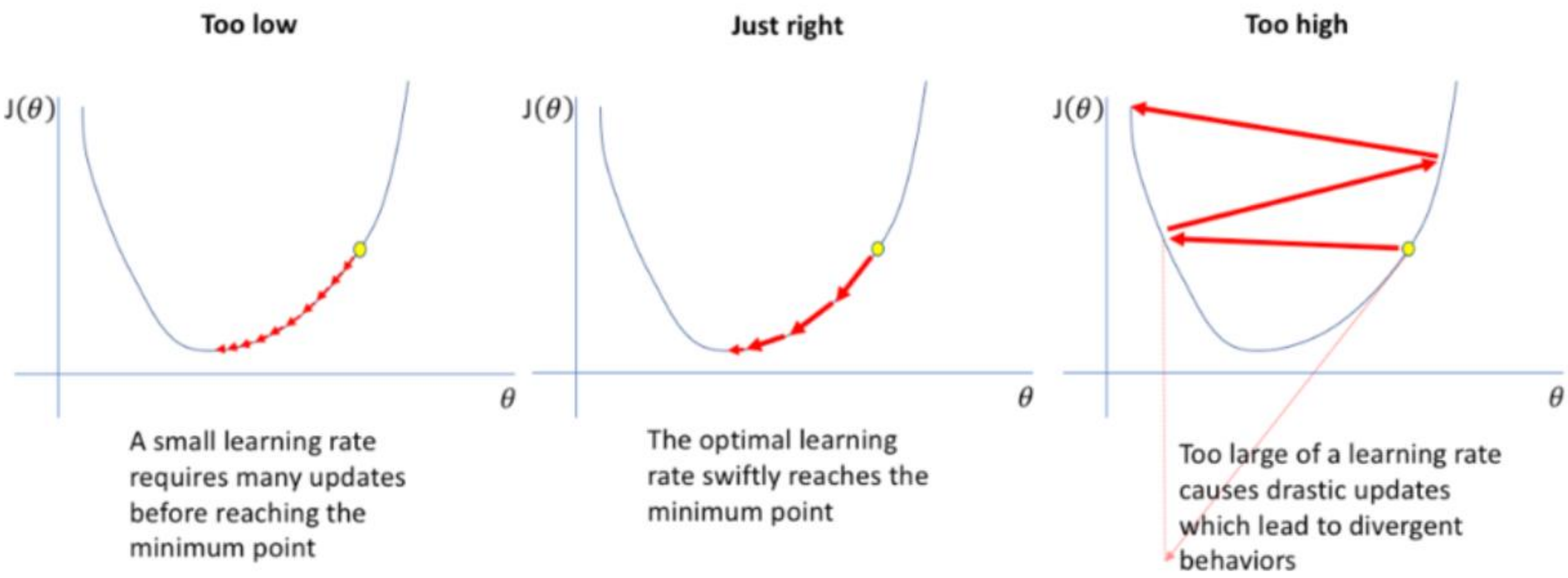
1. 장점
 - Local minimum에 빠질 가능성 적음
 - 걸리는 시간 단축, 빠른 수렴

2. 단점
 - global minimum을 못 찾을수도..
 - GPU 사용 제한 (하나씩 수행)

# Homework

**표에 나타난 Optimizer**들이 어떤 원리로 작동하는지 조사해주세요!



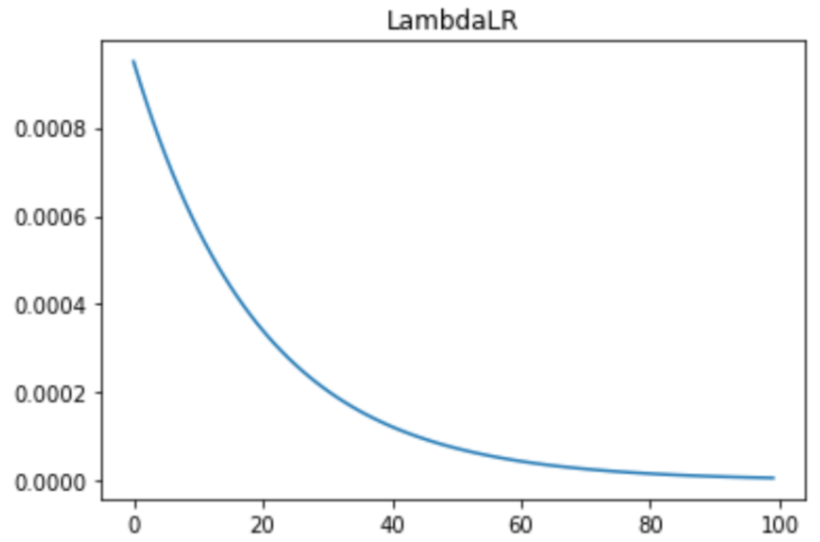



Learning rate Scheduler → Adaptive Learning rate Selection

Deep Learning Optimization

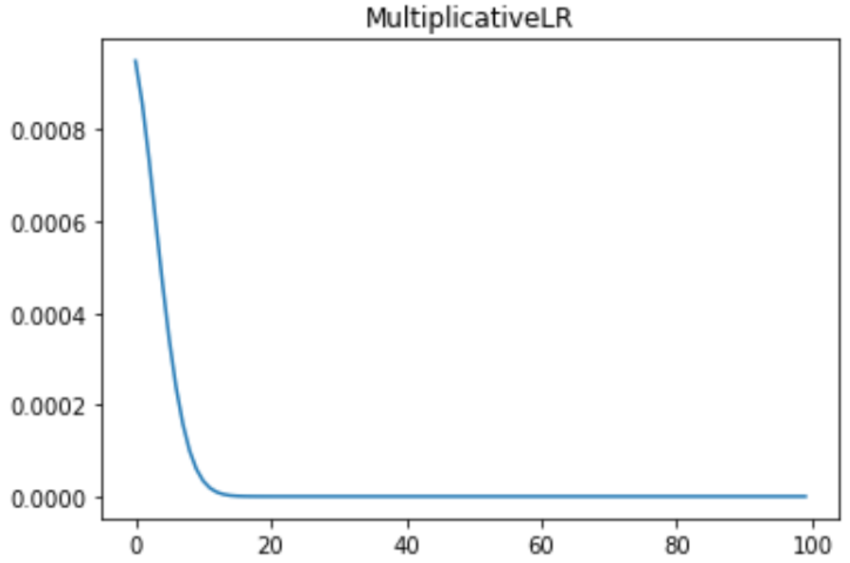
**LambdaLR**

$$lr_{\text{epoch}} = lr_{\text{initial}} * Lambda(epoch)$$



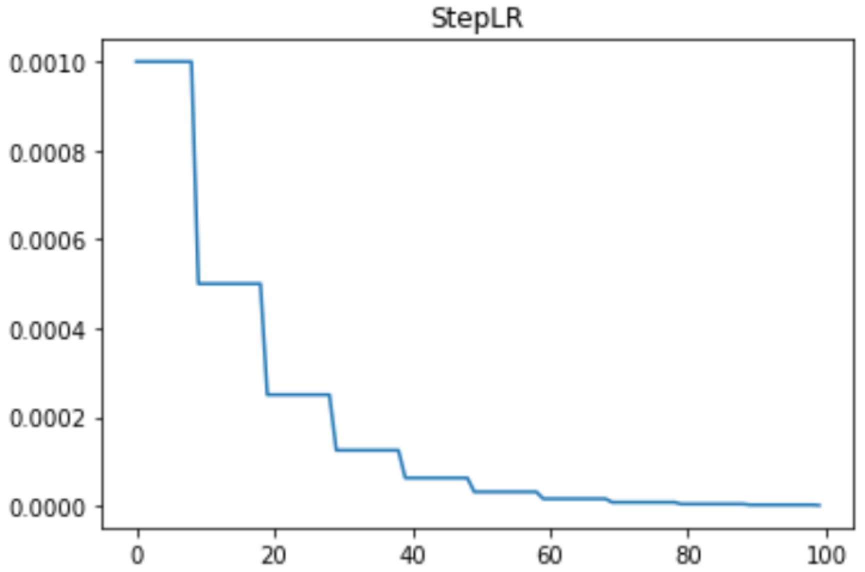

**MultiplicativeLR**

$$lr_{\text{epoch}} = lr_{\text{epoch - 1}} * Lambda(epoch)$$




**StepLR**

$$lr_{\text{epoch}} = \begin{cases} Gamma * lr_{\text{epoch - 1}}, & \text{if epoch \% step\_size} = 0 \\ lr_{\text{epoch - 1}}, & \text{otherwise} \end{cases}$$

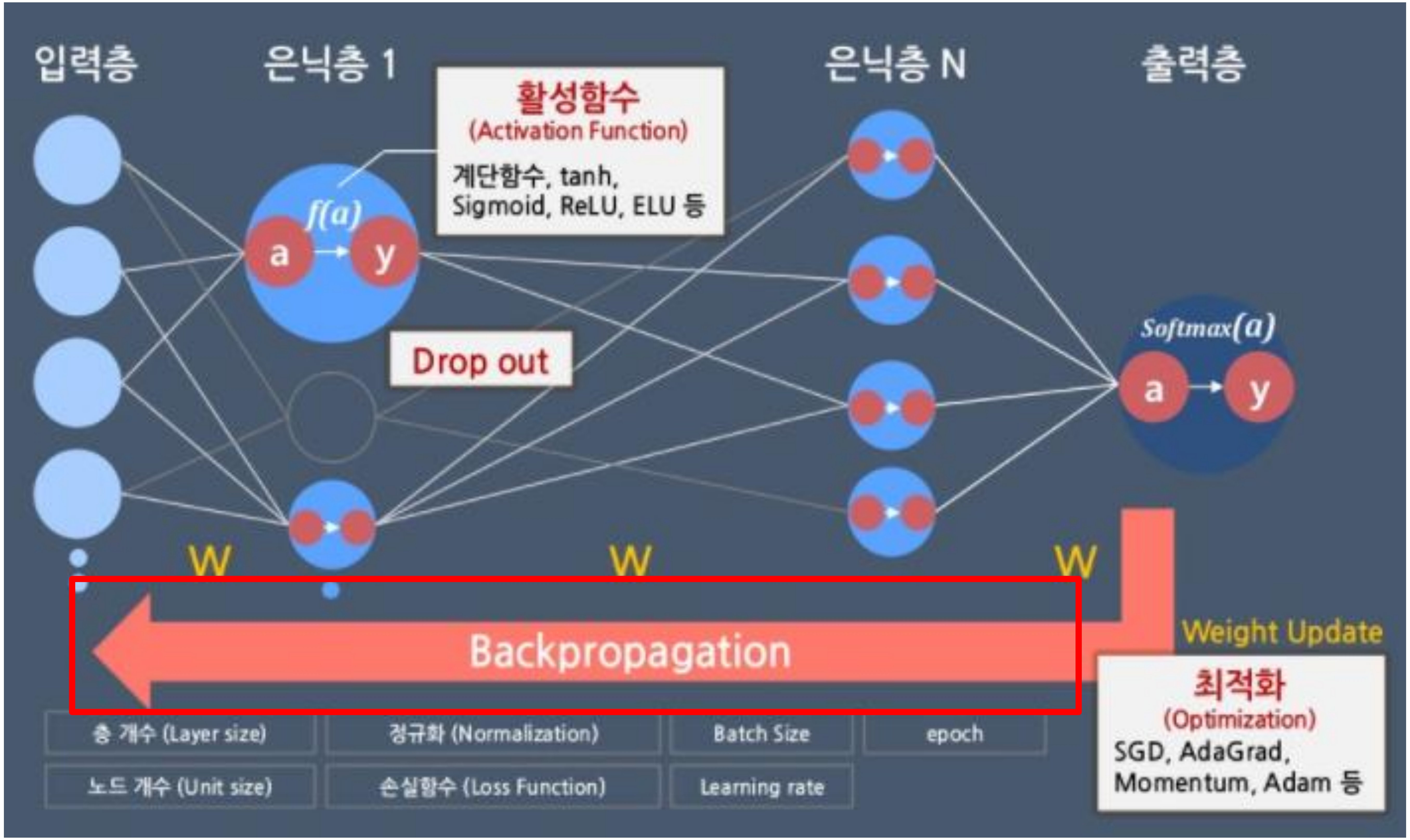


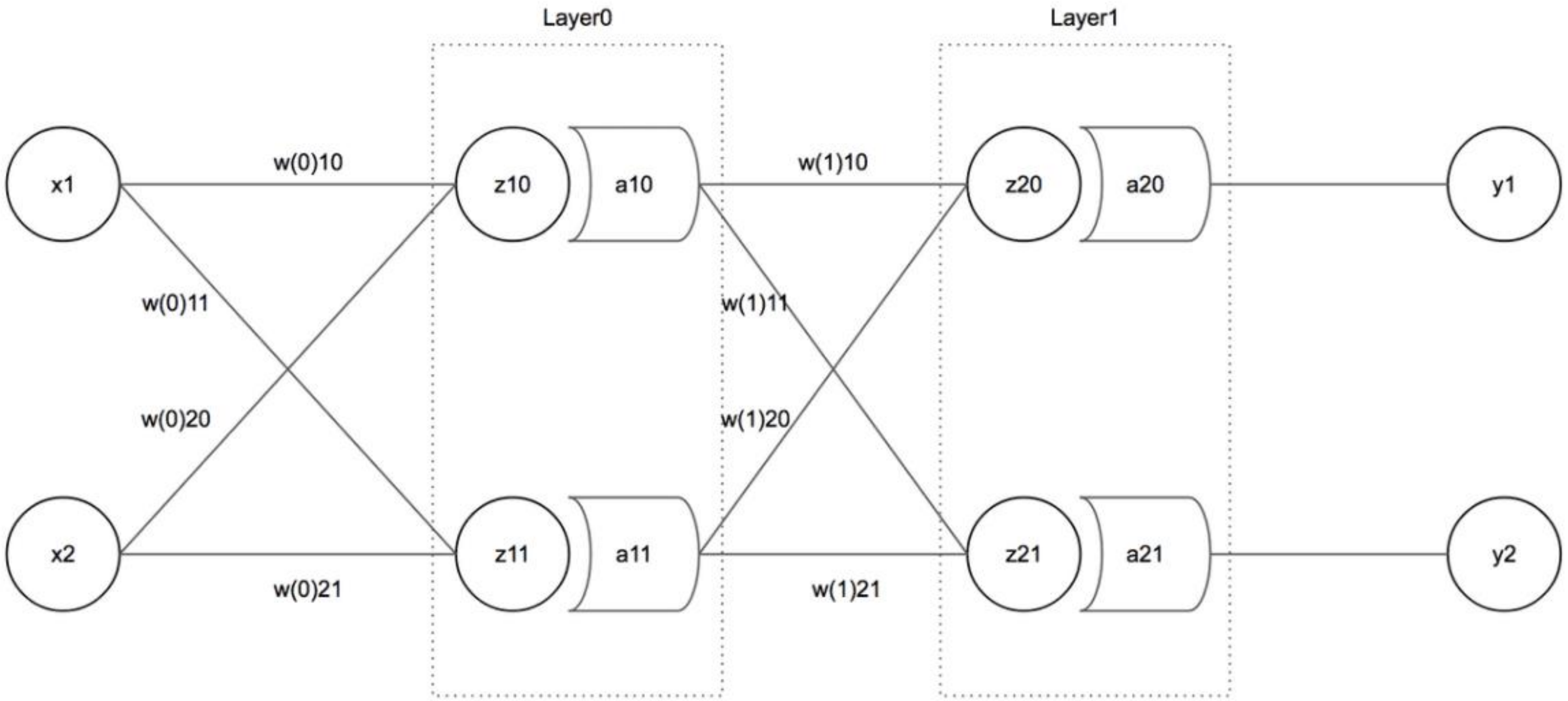

Learning rate Scheduler → Adaptive Learning rate Selection

# 03. BackPropagation

## Overview

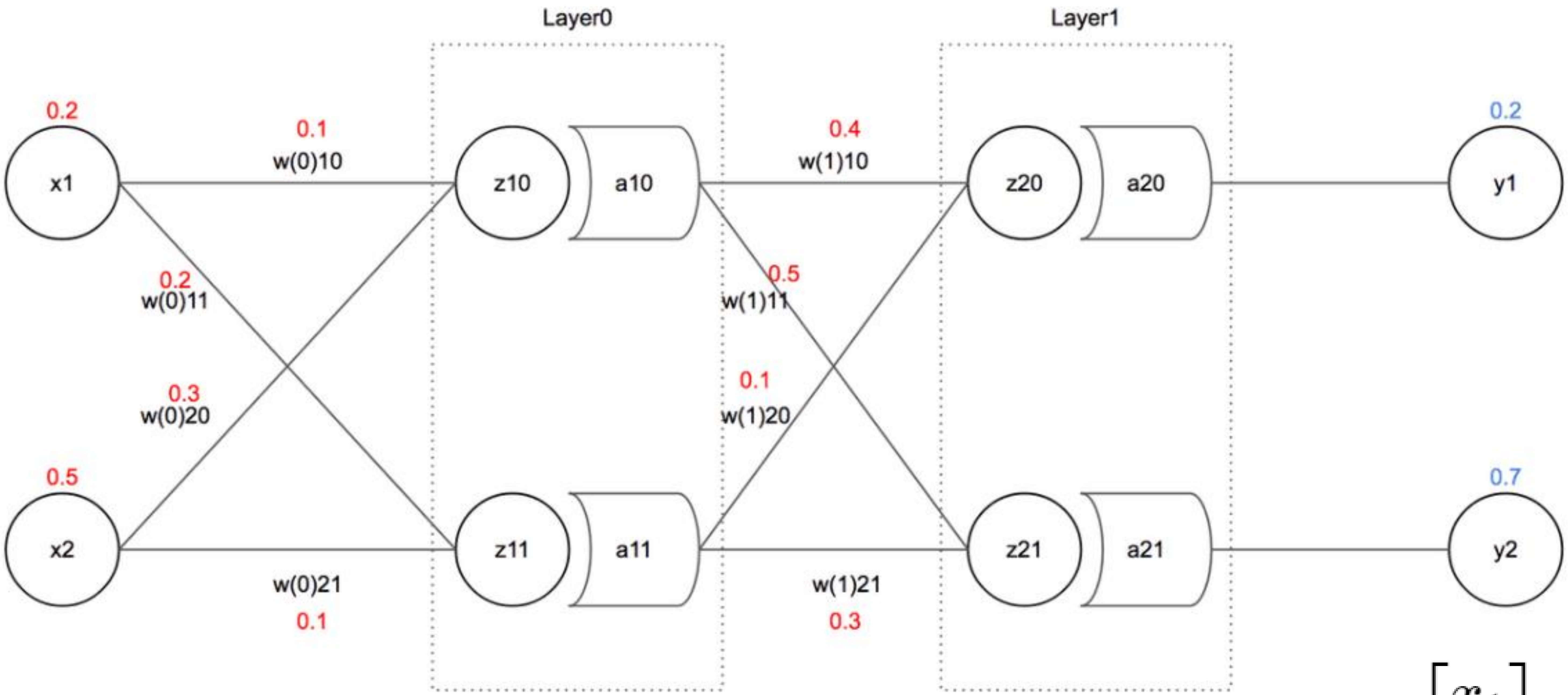
Deep Learning Architecture






# Backpropagation(역전파)

Deep Learning Architecture

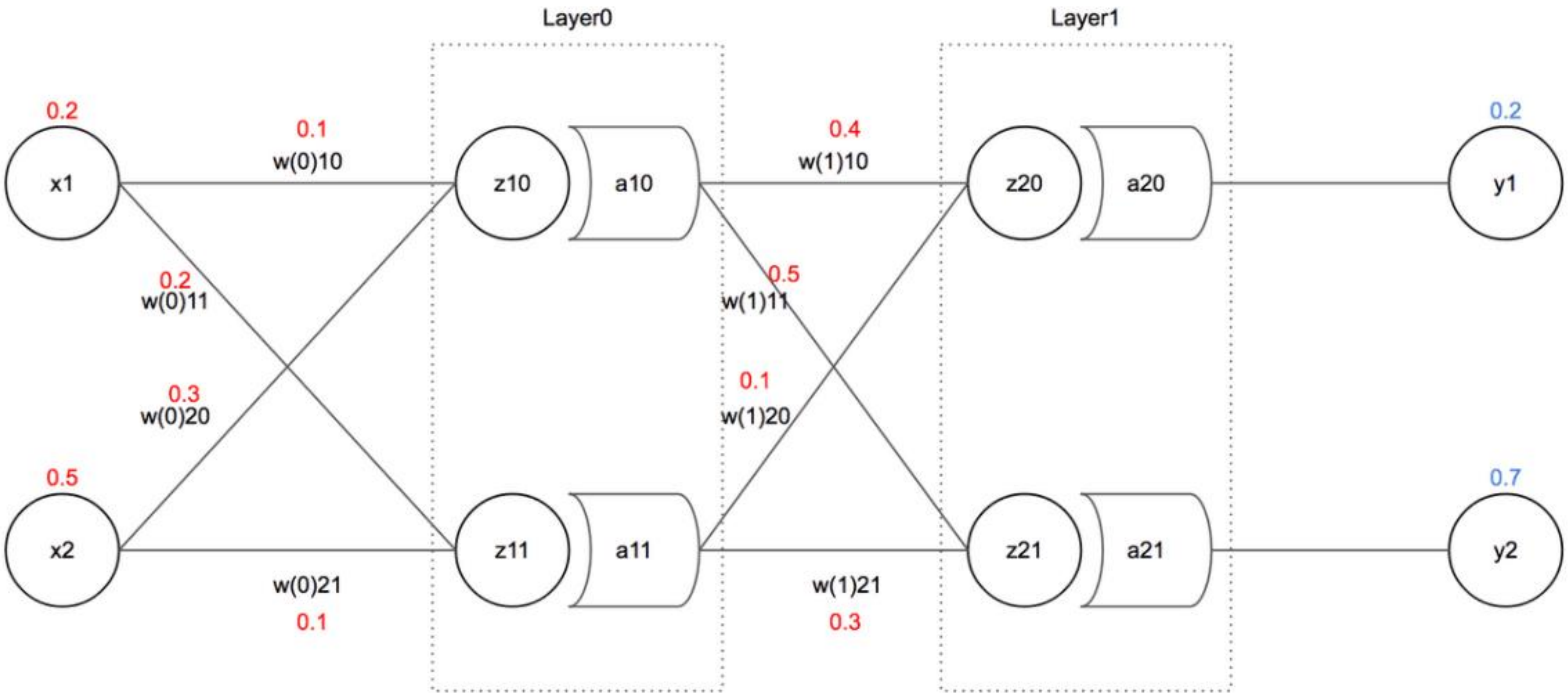



$$z_{10} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times \begin{bmatrix} w_{10}^0 & w_{20}^0 \end{bmatrix}$$

$$z_{11} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \times \begin{bmatrix} w_{11}^0 & w_{21}^0 \end{bmatrix}$$

# Backpropagation(역전파)

Deep Learning Architecture




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

$$z_{10} = x_1 w_{10}^0 + x_2 w_{20}^0 = (0.2 \times 0.1) + (0.5 \times 0.3) = 0.02 + 0.15 = 0.17$$

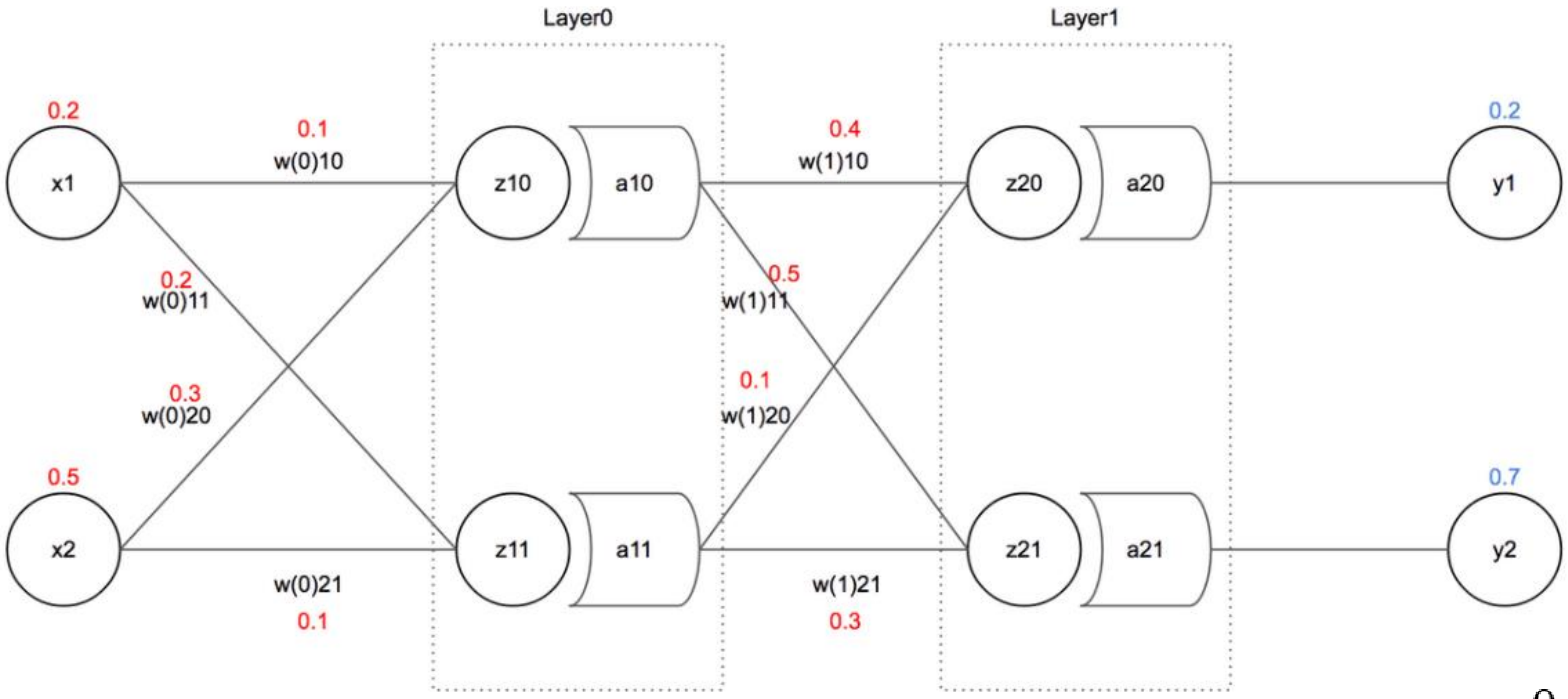
$$z_{11} = x_1 w_{11}^0 + x_2 w_{21}^0 = (0.2 \times 0.2) + (0.5 \times 0.1) = 0.04 + 0.05 = 0.09$$

# Backpropagation(역전파)

$$a_{10} = \sigma(z_{10}) = 0.54$$

Deep Learning Architecture

$$a_{11} = \sigma(z_{11}) = 0.52$$




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

$$z_{20} = 0.27$$

$$a_{20} = 0.57$$

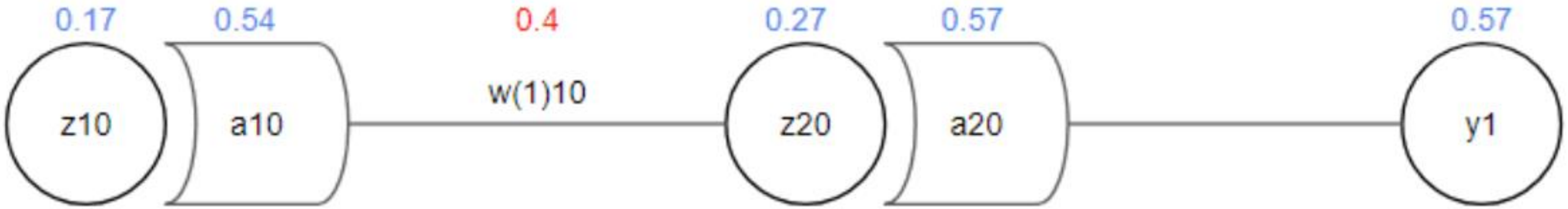
$$z_{21} = 0.43$$

$$a_{21} = 0.61$$

# Backpropagation(역전파)

Deep Learning Architecture

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



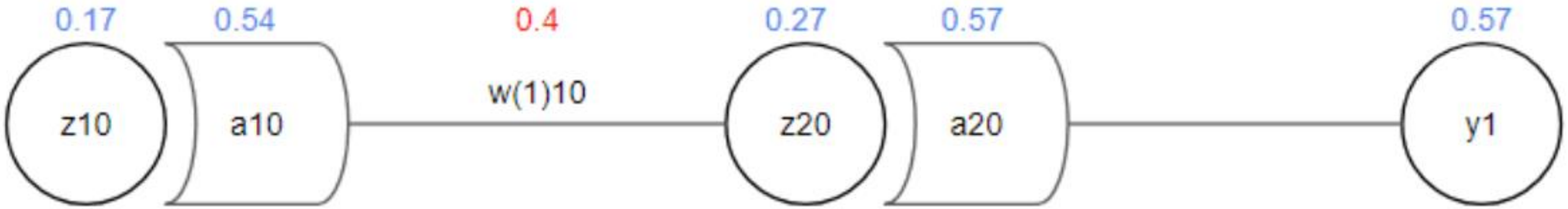

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E}{\partial w^1_{10}} = \boxed{\frac{\partial E}{\partial a_{20}}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial w^1_{10}}$$

$$\frac{\partial E}{\partial a_{20}} = (t_1 - a_{20}) * -1 + 0 = (0.2 - 0.57) \times -1 = 0.37$$

# Backpropagation(역전파)

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



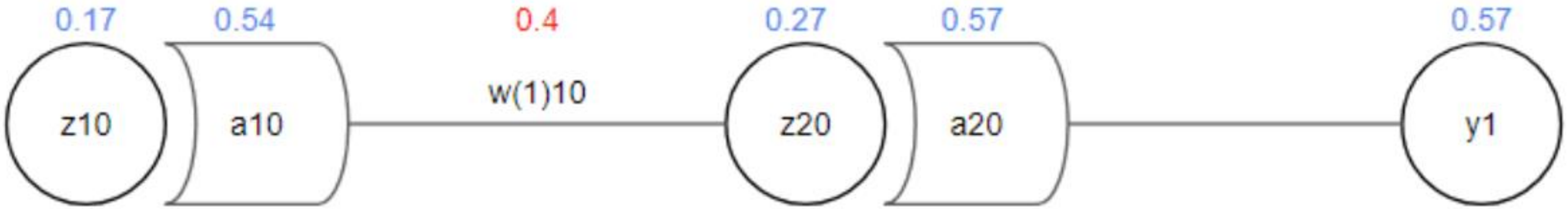

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E}{\partial w_{10}^1} = \frac{\partial E}{\partial a_{20}} \boxed{\frac{\partial a_{20}}{\partial z_{20}}} \frac{\partial z_{20}}{\partial w_{10}^1}$$

$$\frac{\partial a_{20}}{\partial z_{20}} = a_{20} \times (1 - a_{20}) = 0.57 \times (1 - 0.57) = 0.25$$

# Backpropagation(역전파)

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



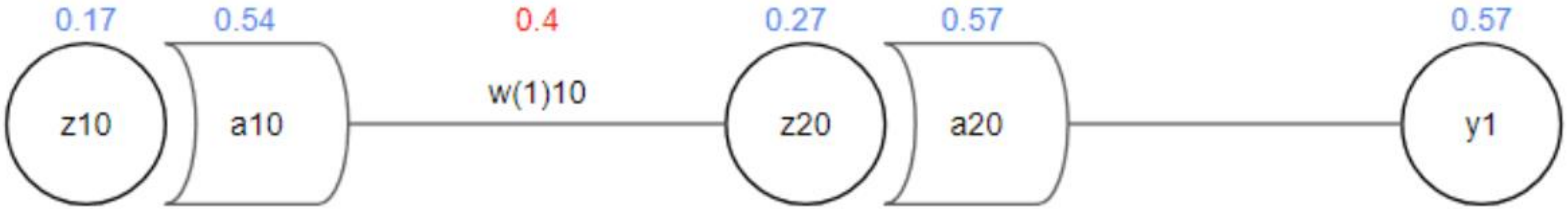

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E}{\partial w_{10}^1} = \frac{\partial E}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \boxed{\frac{\partial z_{20}}{\partial w_{10}^1}}$$

$$\frac{\partial z_{20}}{\partial w_{10}^1} = a_{10} + 0 = 0.54$$

# Backpropagation(역전파)

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



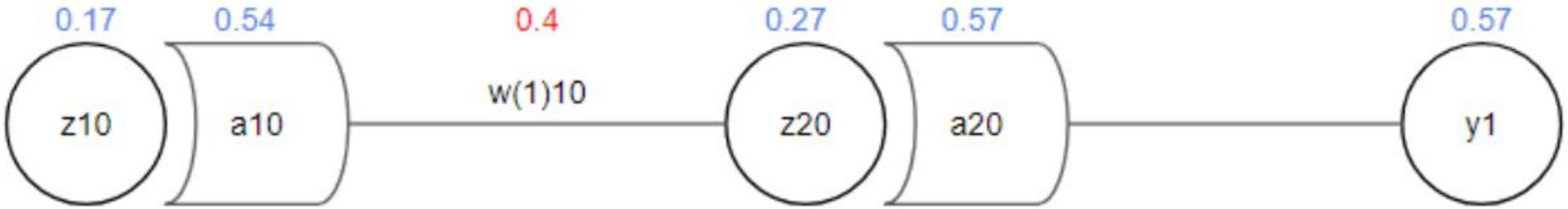

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E}{\partial w_{10}^1} = \frac{\partial E}{\partial a_{20}} \frac{\partial a_{20}}{\partial z_{20}} \frac{\partial z_{20}}{\partial w_{10}^1}$$

$$\boxed{\frac{\partial E}{\partial w_{10}^1} = 0.37 \times 0.25 \times 0.54 = 0.049}$$

## Backpropagation(역전파)

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

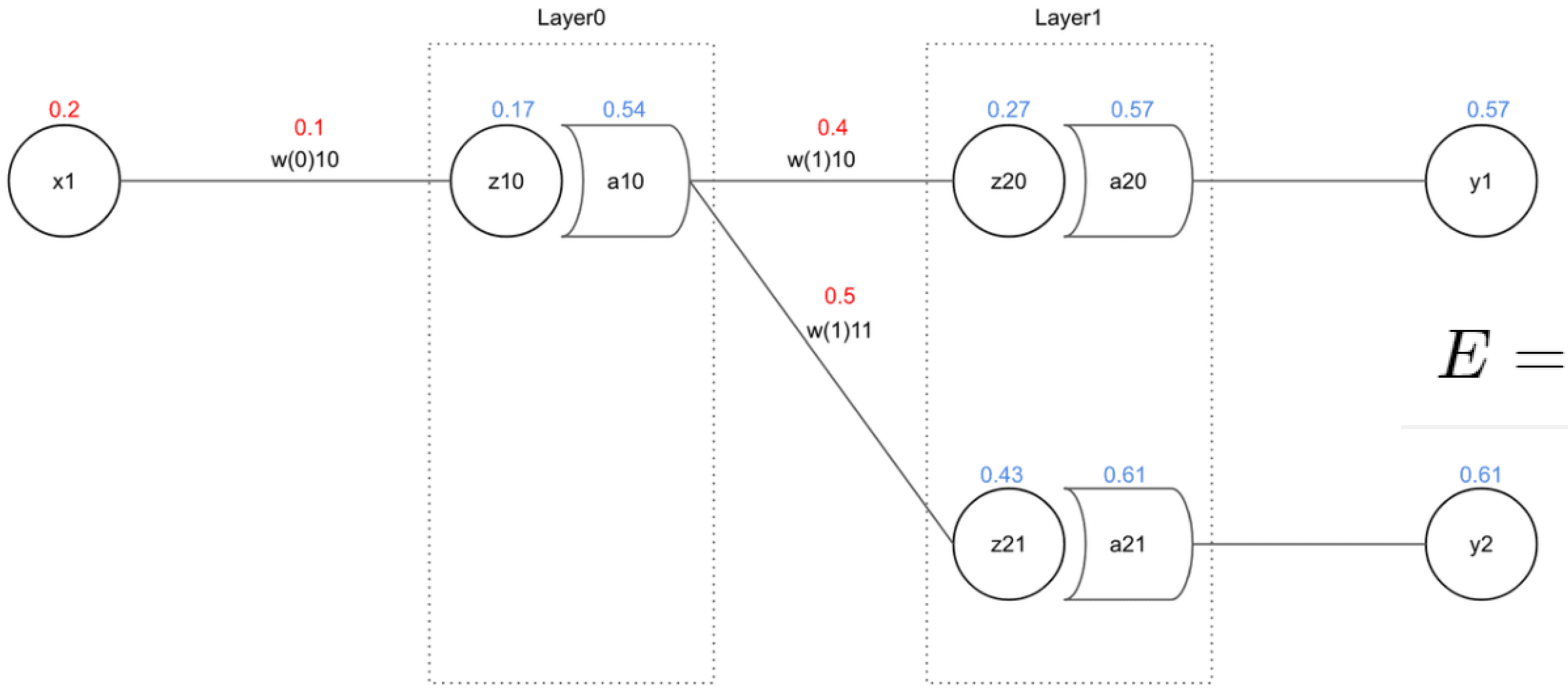



$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

By the Gradient Descent,

$$w_{10}^{1+} = w_{10}^{1} - (L * \frac{\partial E}{\partial w_{10}^{1}}) = 0.4 - (0.3 \times 0.049) = 0.3853$$

# Backpropagation(역전파)




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

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E_t}{\partial w^0_{10}} = (\boxed{\frac{\partial E_1}{\partial a_{10}}} + \frac{\partial E_2}{\partial a_{10}})\frac{\partial a_{10}}{\partial z_{10}}\frac{\partial z_{10}}{\partial w^0_{10}}$$

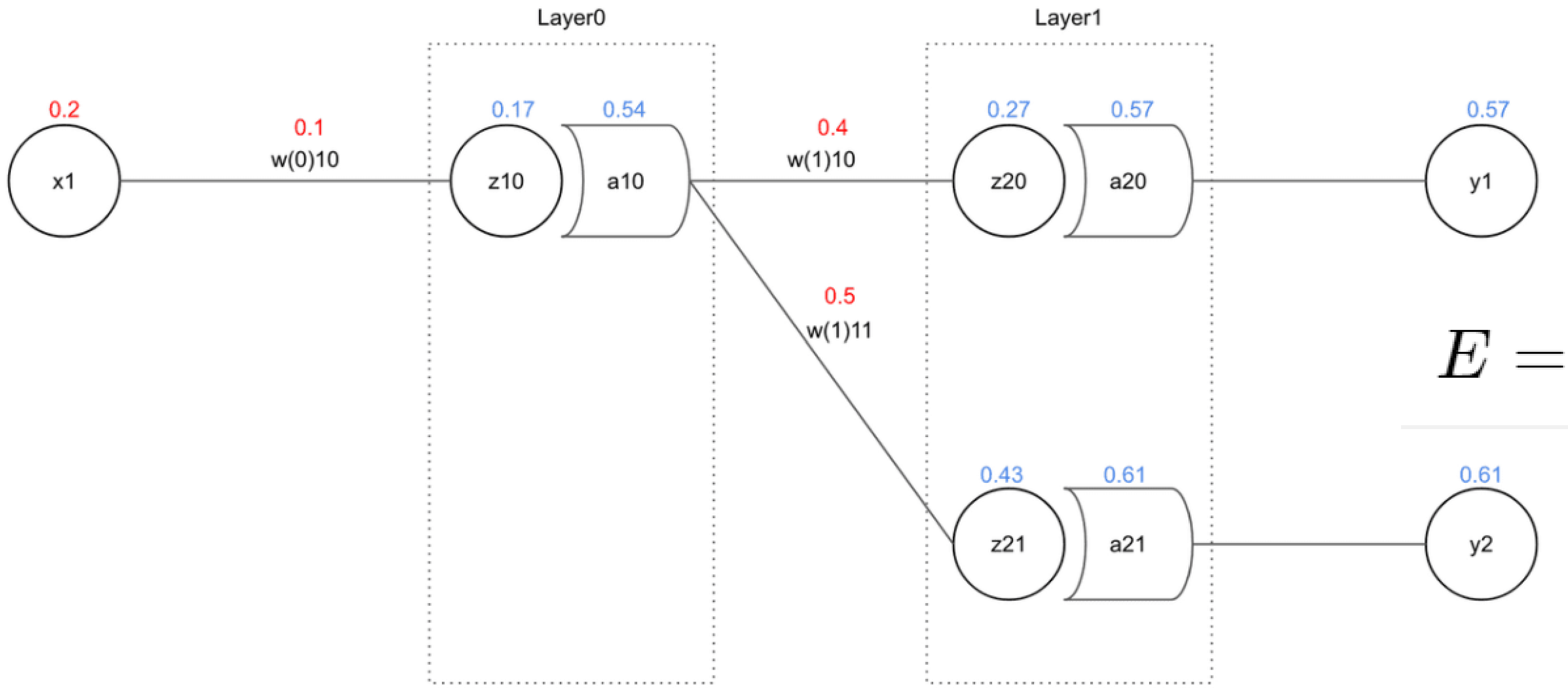
$$\frac{\partial E_1}{\partial a_{10}} = \frac{\partial E_1}{\partial a_{20}}\frac{\partial a_{20}}{\partial z_{20}}\frac{\partial z_{20}}{\partial a_{10}}$$

$$= -(t_1 - a_{20}) \times a_{20} \times (1 - a_{20}) \times w^1_{10}$$

$$= -(0.2 - 0.57) \times 0.57 \times (1 - 0.57) \times 0.4$$

$$= 0.03627$$

# Backpropagation(역전파)




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

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E_t}{\partial w^0_{10}} = \left(\frac{\partial E_1}{\partial a_{10}} + \boxed{\frac{\partial E_2}{\partial a_{10}}}\right)\frac{\partial a_{10}}{\partial z_{10}}\frac{\partial z_{10}}{\partial w^0_{10}}$$

$$\frac{\partial E_2}{\partial a_{10}} = \frac{\partial E_2}{\partial a_{21}}\frac{\partial a_{21}}{\partial z_{21}}\frac{\partial z_{21}}{\partial a_{10}}$$

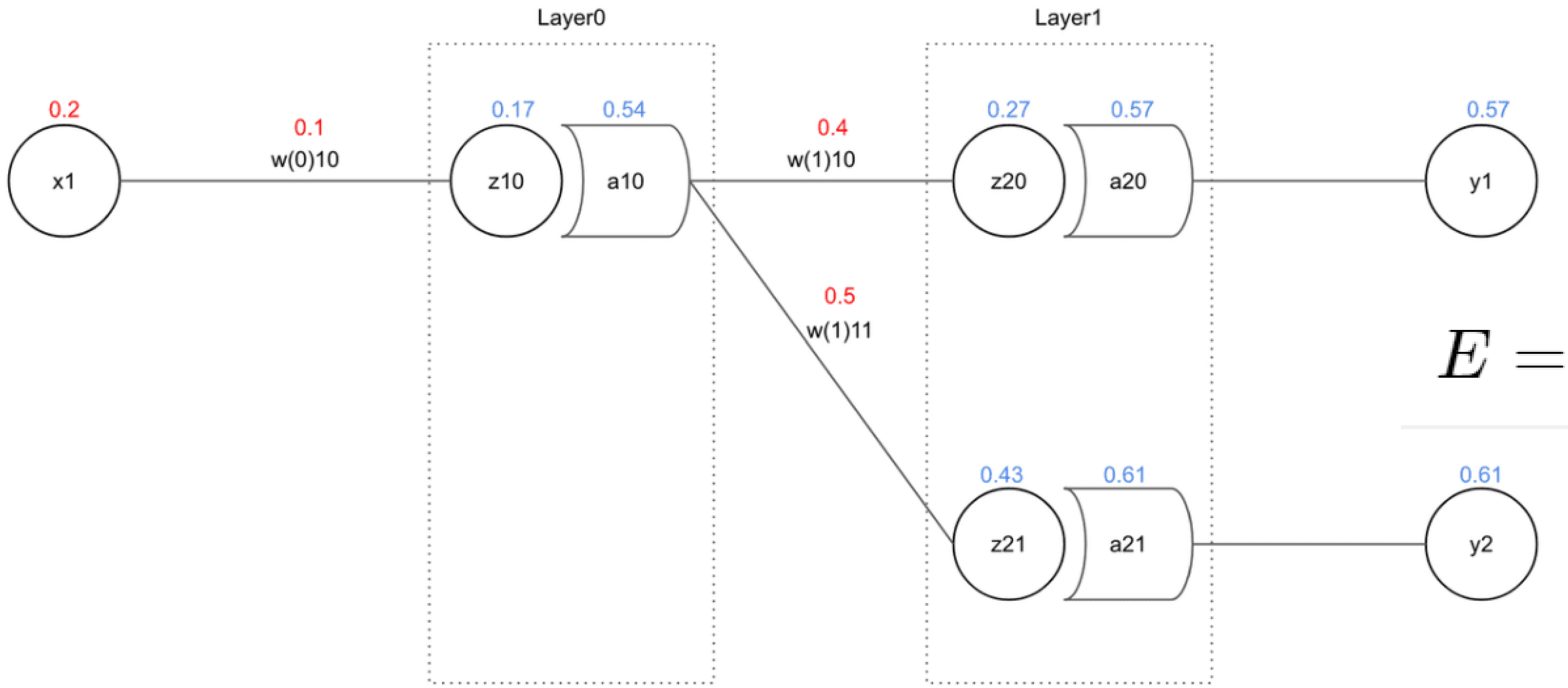
$$= -(t_2 - a_{21}) \times a_{21} \times (1 - a_{21}) \times w^1_{11}$$

$$= -(0.7 - 0.61) \times 0.61 \times (1 - 0.61) \times 0.5$$

$$= -0.0107$$

# Backpropagation(역전파)

Deep Learning Architecture




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

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

$$\frac{\partial E_t}{\partial w_{10}^0} = (\frac{\partial E_1}{\partial a_{10}} + \frac{\partial E_2}{\partial a_{10}})\frac{\partial a_{10}}{\partial z_{10}}\frac{\partial z_{10}}{\partial w_{10}^0}$$

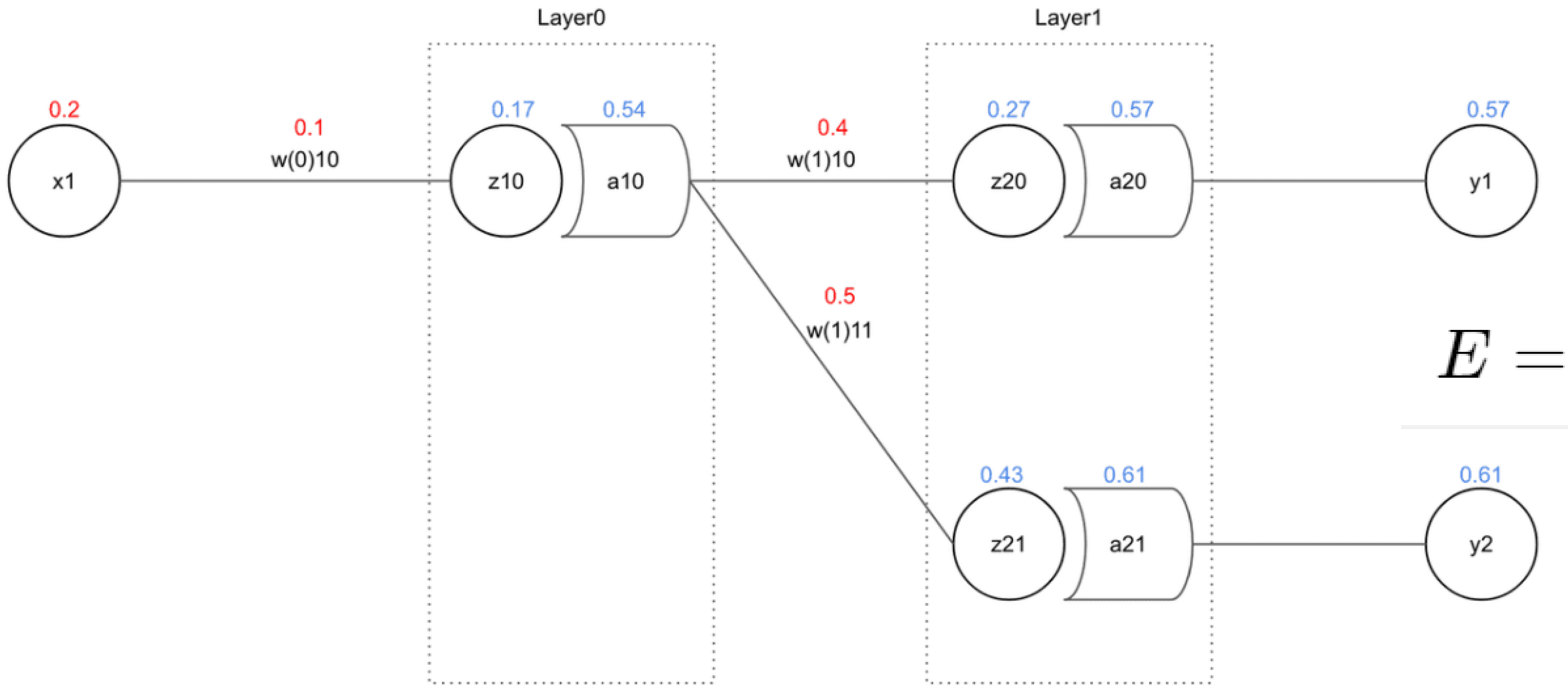
$$\frac{\partial E_t}{\partial w_{10}^0} = (\frac{\partial E_1}{\partial a_{10}} + \frac{\partial E_2}{\partial a_{10}})\frac{\partial a_{10}}{\partial z_{10}}\frac{\partial z_{10}}{\partial w_{10}^0}$$

$$= (0.03627 + (-0.0107)) \times 0.2484 \times 0.54$$

$$= 0.0034$$

# Backpropagation(역전파)

Deep Learning Architecture




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

$$E = \frac{1}{2}((t_1 - a_{20})^2 + (t_2 - a_{21})^2)$$

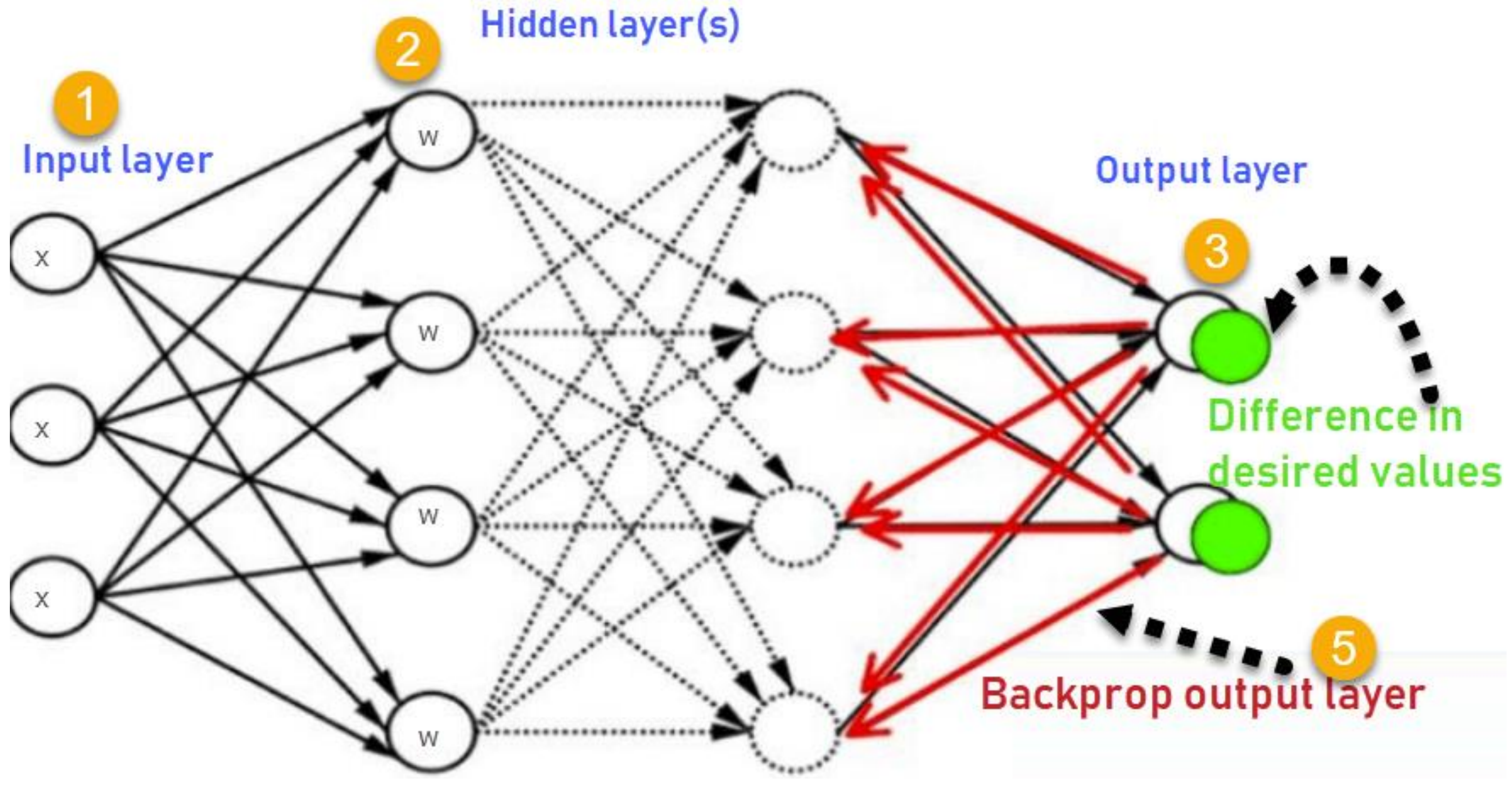
$$w_{10}^{0+} = w_{10}^{0} - (L * \frac{\partial E_t}{\partial w_{10}^{0}}) = 0.1 - (0.3 \times 0.0034) = 0.09897$$

# Backpropagation(역전파)

Deep Learning Architecture

# Backpropagation(역전파)

Deep Learning Architecture




$$\nabla_{\boldsymbol{x}} z = \left(\frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}}\right)^{\top} \nabla_{\boldsymbol{y}} z$$

$$\nabla_{\mathbf{X}} z = \sum_j (\nabla_{\mathbf{X}} Y_j) \frac{\partial z}{\partial Y_j}$$

$$\frac{\partial E}{\partial w_{ji}} = \sum_n \frac{\partial E_n}{\partial w_{ji}}$$

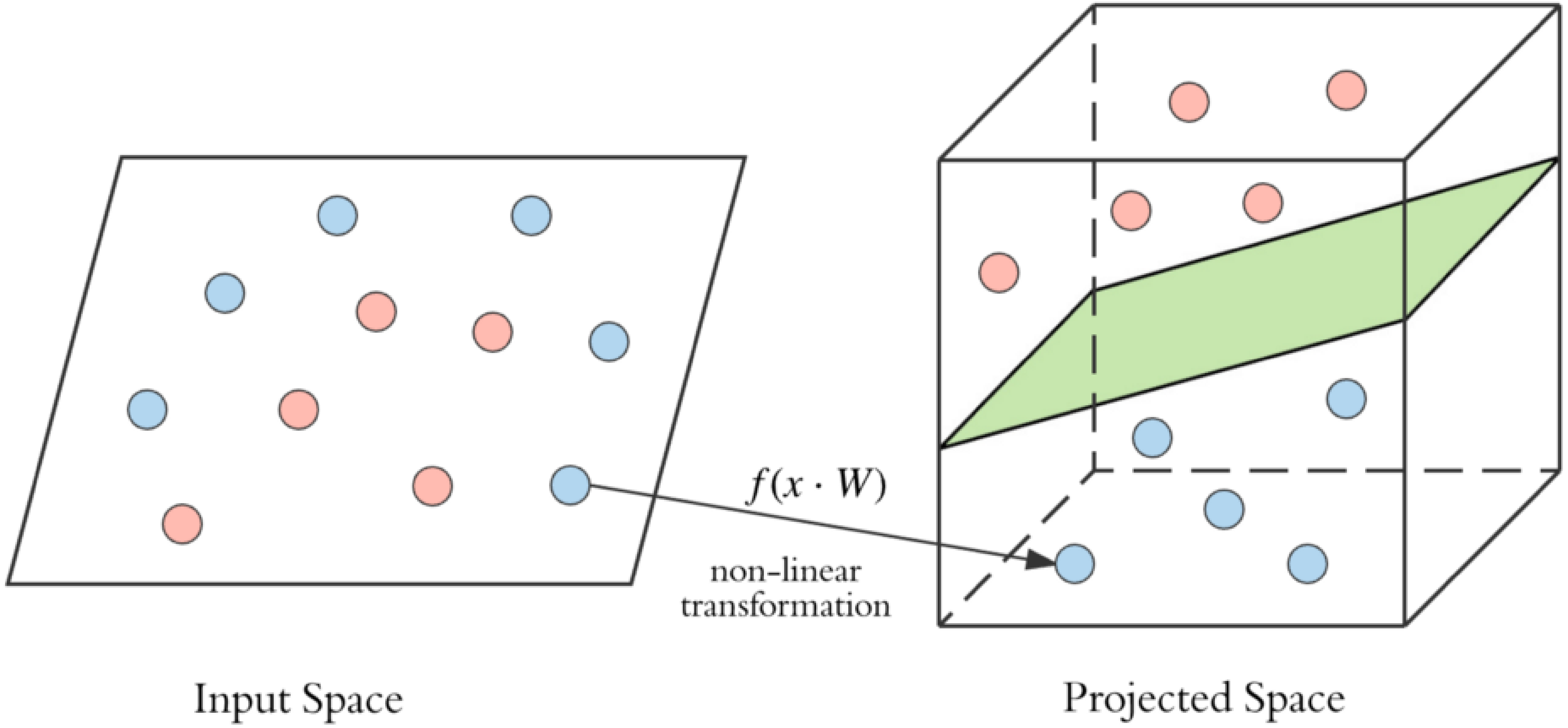
$$w_{ji}^{+} = w_{ji} - \gamma \frac{\partial E}{\partial w_{ji}}$$

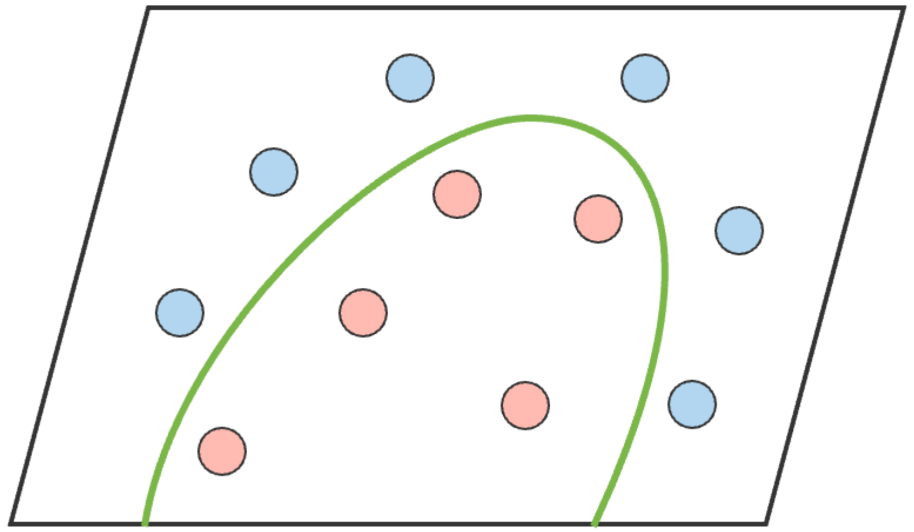
- Optimizer
- Learning rate
- Loss function

# Backpropagation(역전파)

Deep Learning Architecture

## XOR Problem



# THANK

# YOU