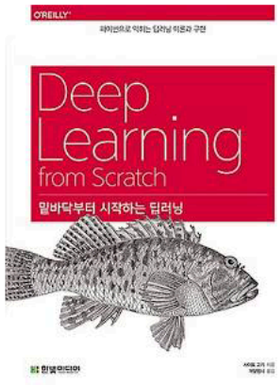
ABC to Deep Learning

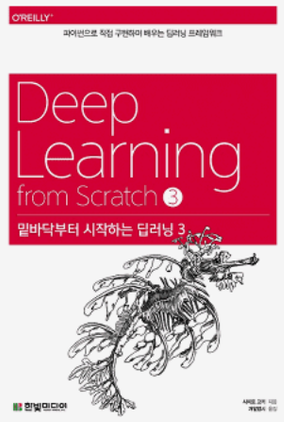
Myeonghun Park

쉬운 참고들

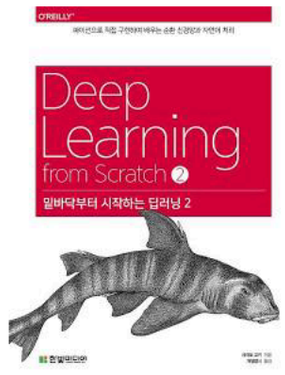
- Google
- 점프 투 파이썬 (https://wikidocs.net/2)



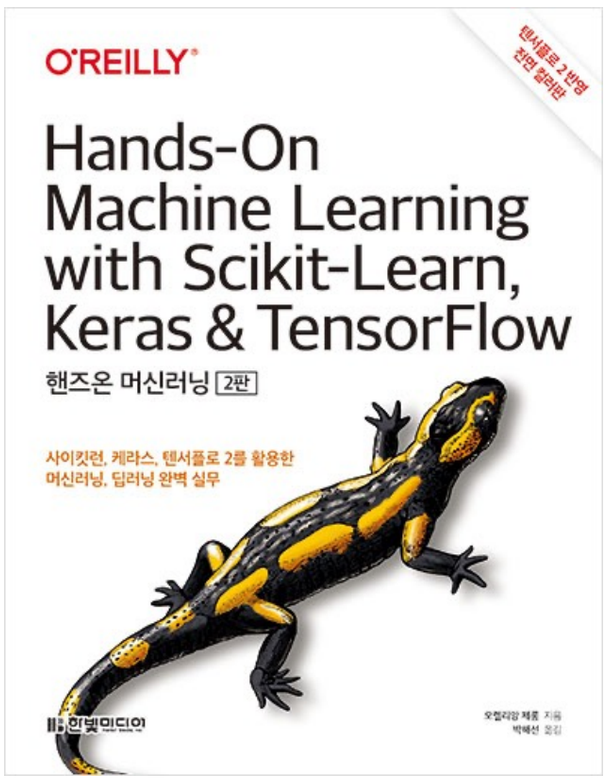
밑바닥부터 시작하는 딥러닝 1



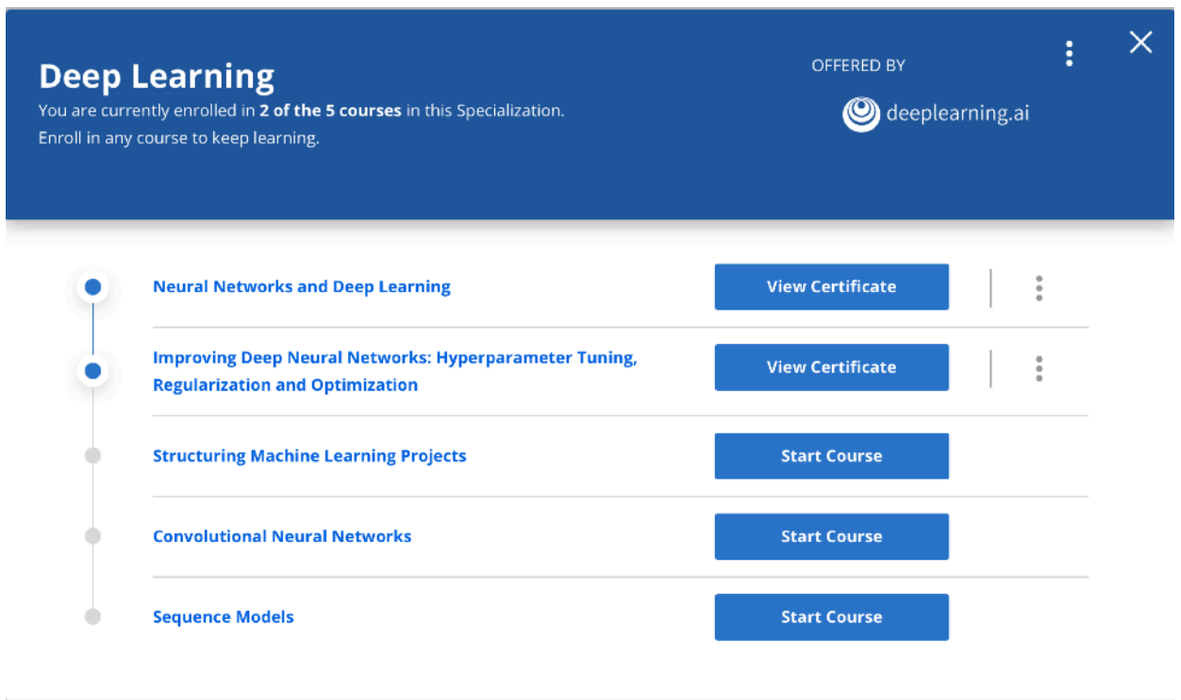
밑바닥부터 시작하는 딥러닝 3



밑바닥부터 시작하는 딥러닝 2



- Coursara: Deep Learning Specialization



인공지능의 성공을 인간의 수행 능력을 기준으로 측정	인공지능의 성공을 이성적 성과 (합리성) 기준으로 측정	
인간처럼 생각하기	**이성적으로 생각하기**	사고와 추론
인간처럼 행동하기	**이성적으로 행동하기**	행동방식

인간처럼 생각하기	이성적으로 생각하기
인간처럼 행동하기	이성적으로 행동하기

- 인간처럼 생각하기: 인간의 사고, 의사결정, 문제 풀기, 학습 등의 활동에 연관시킬 수 있는 활동들의 자동화 (인지과학, Bellman 1978)

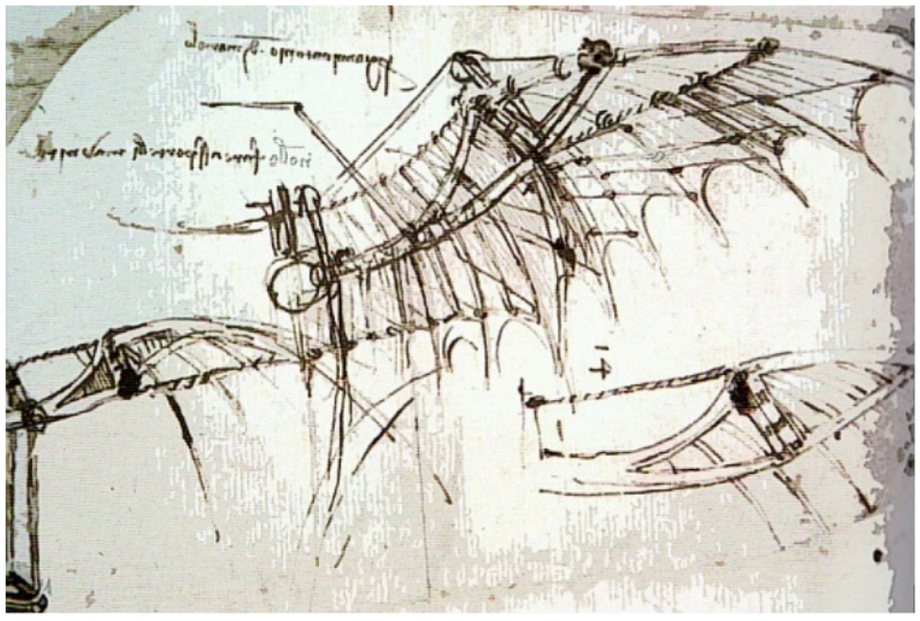
- 이성적으로 생각하기: 인지, 추론, 행동할 수 있도록 하는 컴퓨팅에 관한 연구 (Winston, 1992)

- 인간처럼 행동하기: "**튜링테스트**" 통과하기. 현재로서는 사람이 더 잘하는 것을 컴퓨터가 하게 만드는 방법에 대한 연구 (Rich, Knight, 1991)

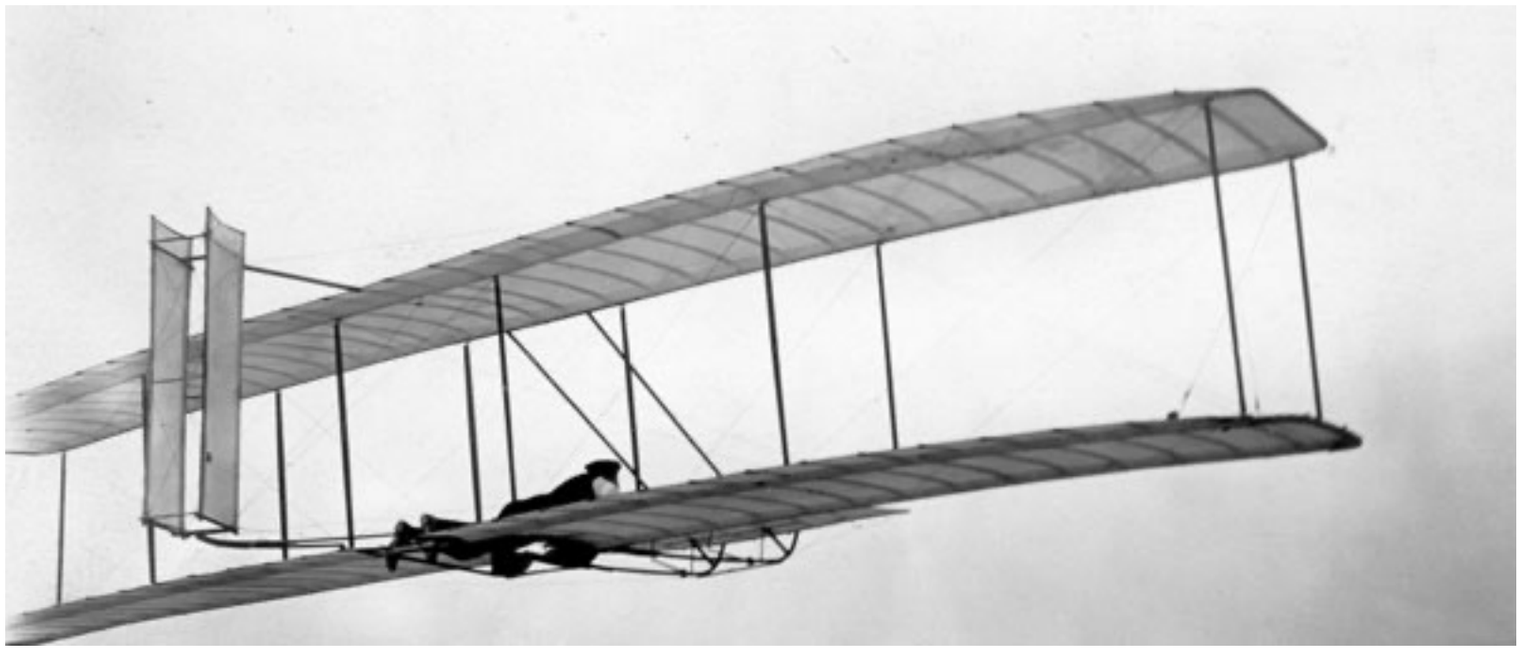
- 이성적으로 행동하기: 목적 달성을 위해 행동을 취하는 것 (게임을 이기는 행위)

인간처럼 행동하기

- 견본을 복제하는 것이 중요한가, 원리를 연구하는 것이 중요한가?



레오나르도 다 빈치의 날개 설계도

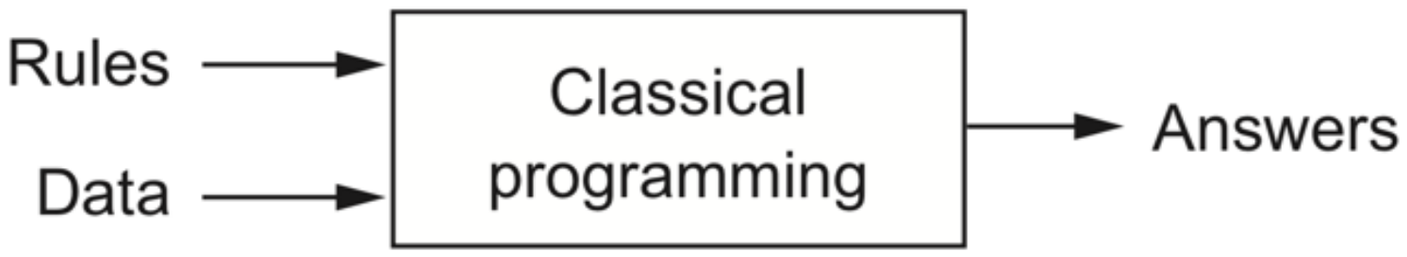


라이트 형제의 비행기

- 공기 역학의 목표가 "다른 비둘기들이 속을 정도로 비둘기처럼 날아다니는 기계를 만드는 것이 아님"

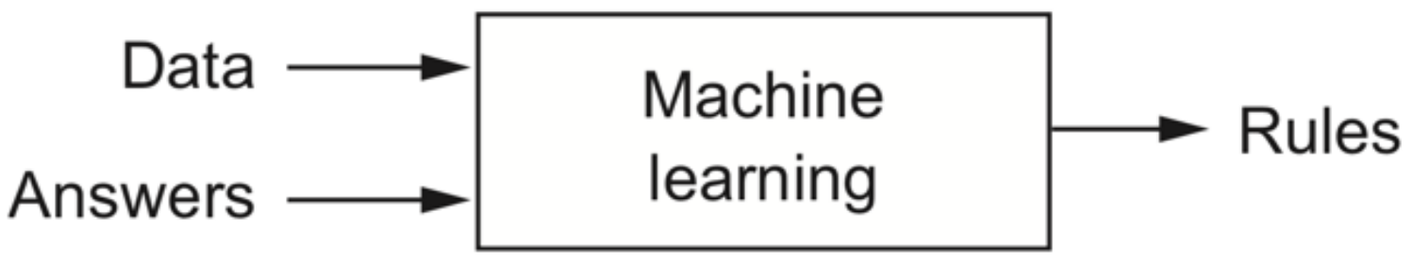
Knowledge based vs. **Data** based

- Expert system: 모든 가능한 규칙을 프로그래밍하여 **결과 산출**



- Symbolic AI: 많은 경우를 전부 고려하여, 데이터 처리 (체스게임)
- Expert system: 전문가의 know-how를 축적하여, 전문가와 동일한 문제해결 능력

- Machine Learning (기계학습): 데이터와 결과를 바탕으로, **규칙을 유출**



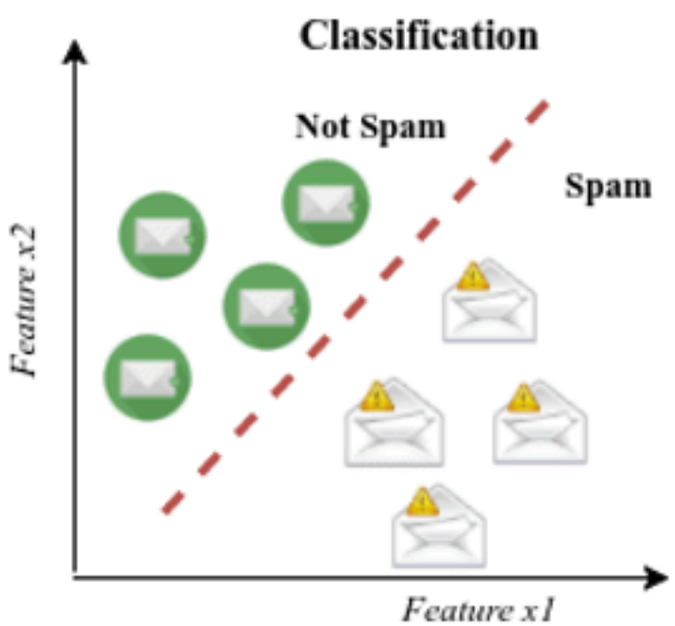
- 데이터 기반 학습
- 모호한 데이터 처리 가능

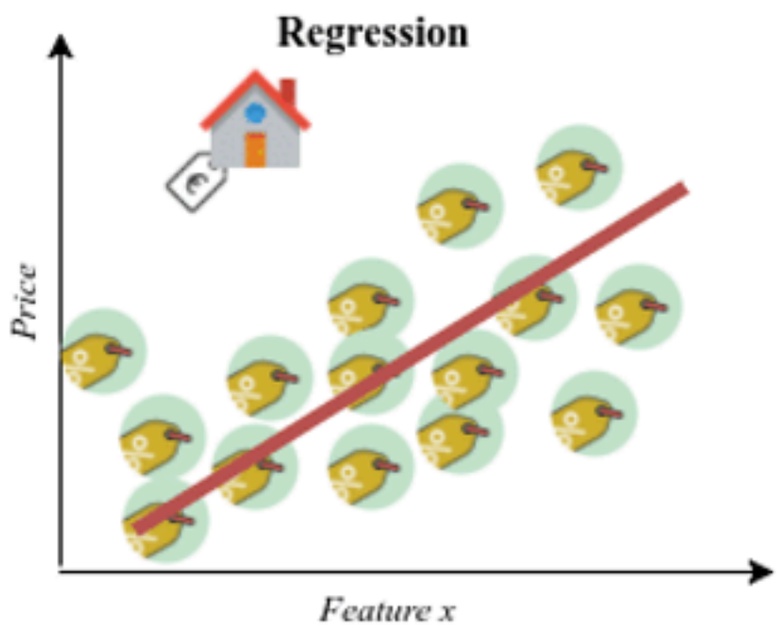
많은 양의 데이터 (빅데이터) 를 통한 학습:

예) 이미지 분석: $O(10^4)$ 개의 픽셀로 구성된 이미지를 수백만 개 학습 :

- 전통적인 통계 분석 방법을 적용하기 어려움.

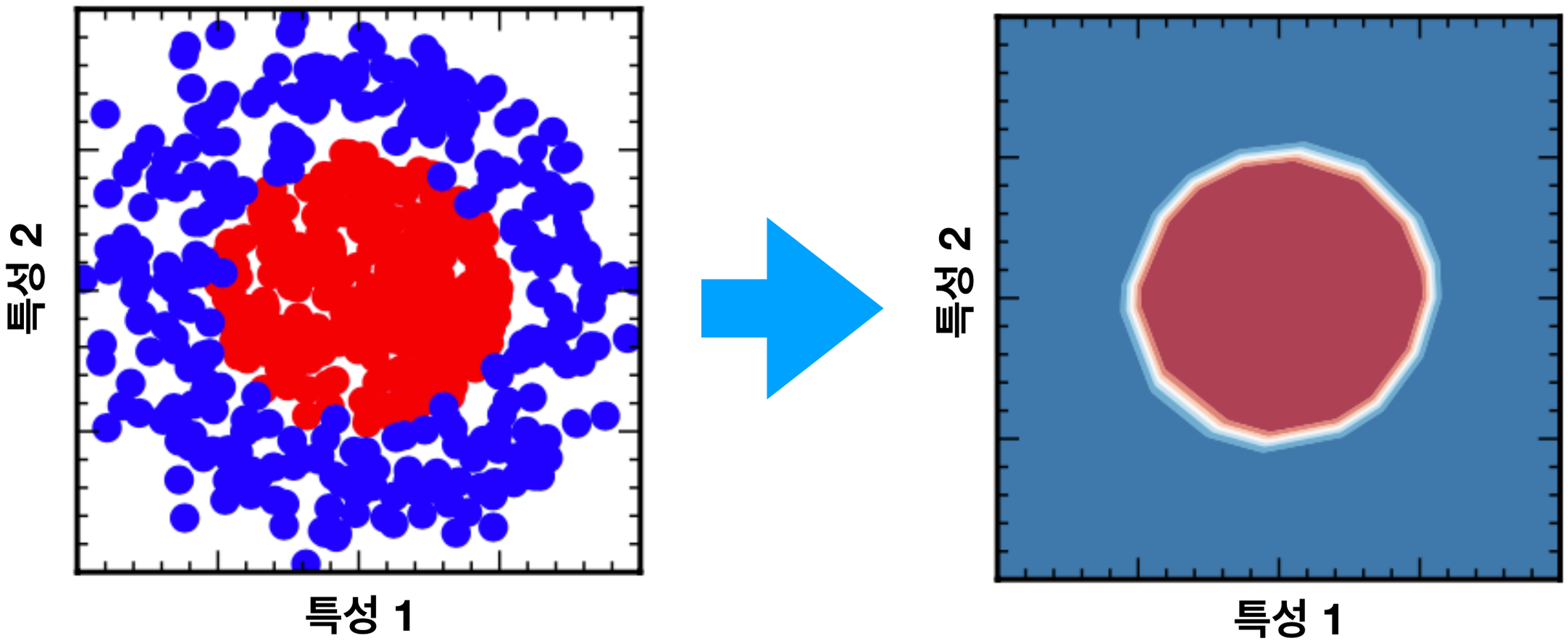
- 데이터를 학습시킬 때, 답(label)을 알려주는가, 알려주지 않는가...





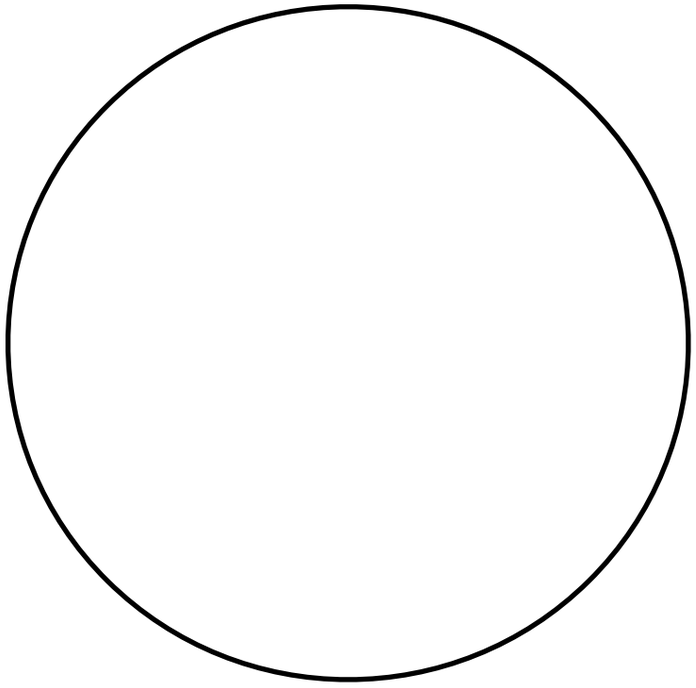
- Classification (분류문제): 데이터의 label이 이산적(discrete)인 경우
- Regression (회귀 문제): 데이터의 label이 연속적인 경우
- Clustering (군집 문제): 데이터에 label을 붙이지 않고, 컴퓨터가 **특성**만을 조사해서, 같은 특성을 갖는 데이터끼리 묶음.

- 분류문제에서, **특성들간의 상관관계**를 파악하는 것이 상당히 중요!

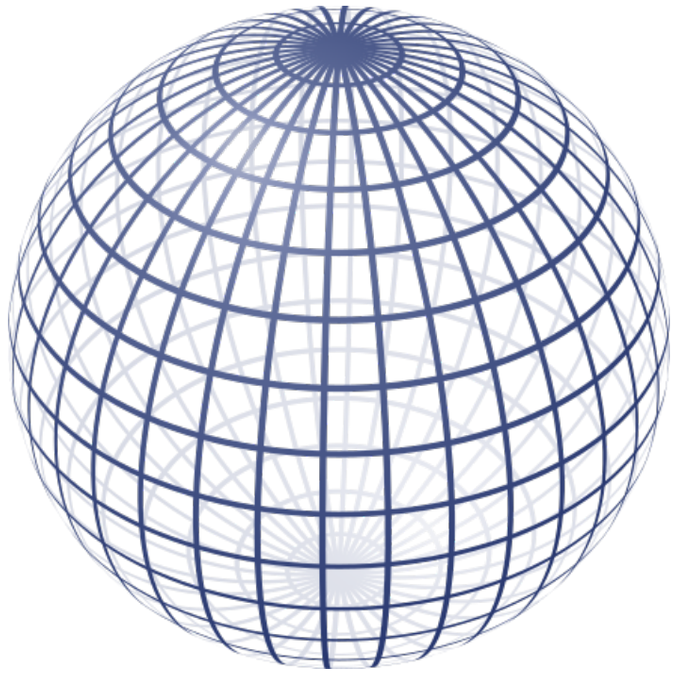


- 컴퓨터에게 비지도 학습을 통해 (왼쪽) 데이터를 학습시키면,
  오른쪽과 같이 서로 다른 데이터들을 구분할 수 있는 경계를 만들어 줌.

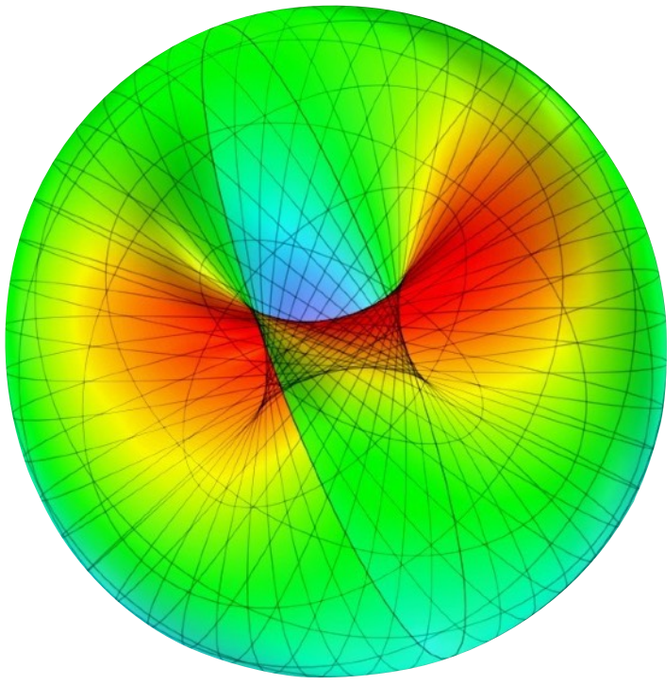
- 2차원 데이터 분포가 아닌, 높은 차원에 데이터가 분포되어 있다면 어떨까? 즉 두 개의 특성들만을 갖는 데이터가 아닌 3개, 4개, 그 이상의 특성들을 갖는 데이터들을 분류하는 경우



S^1



S^2



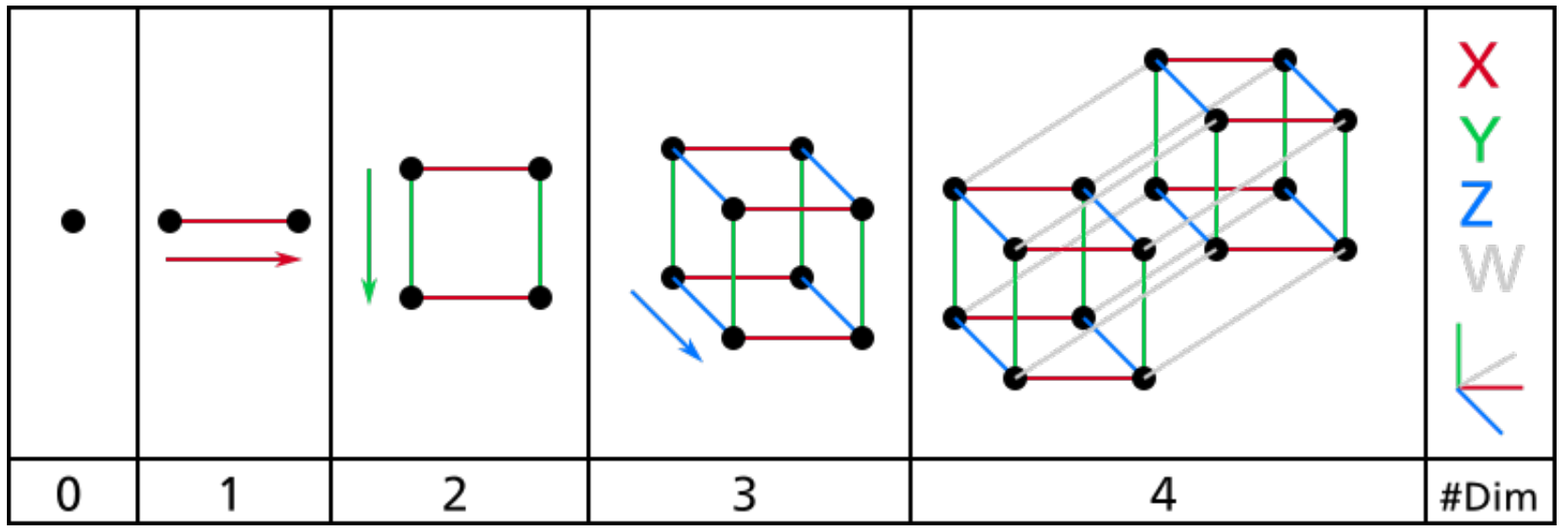
Projection of S^3 into R^3

- **컴퓨터가 효과적으로 데이터들을 구분할 수 있을까?**

특성 / 상관관계

- 데이터 기반 학습에서의 중요 문제
  Curse of dimensionality problem (차원의 저주) since 1957

변의 길이가 1인
n차원 단위 큐빅



두 점사이의 거리

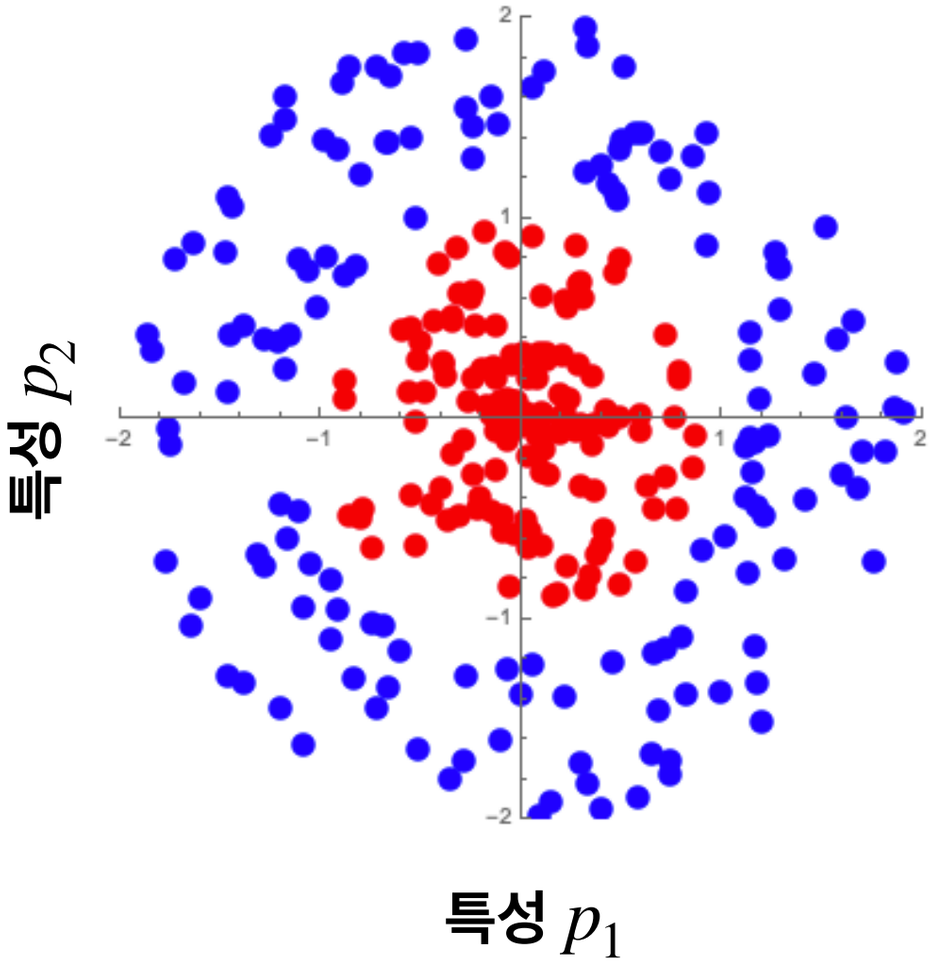
0.3332 0.5211 0.6616 0.7776 $\leq \sqrt{n/6}$

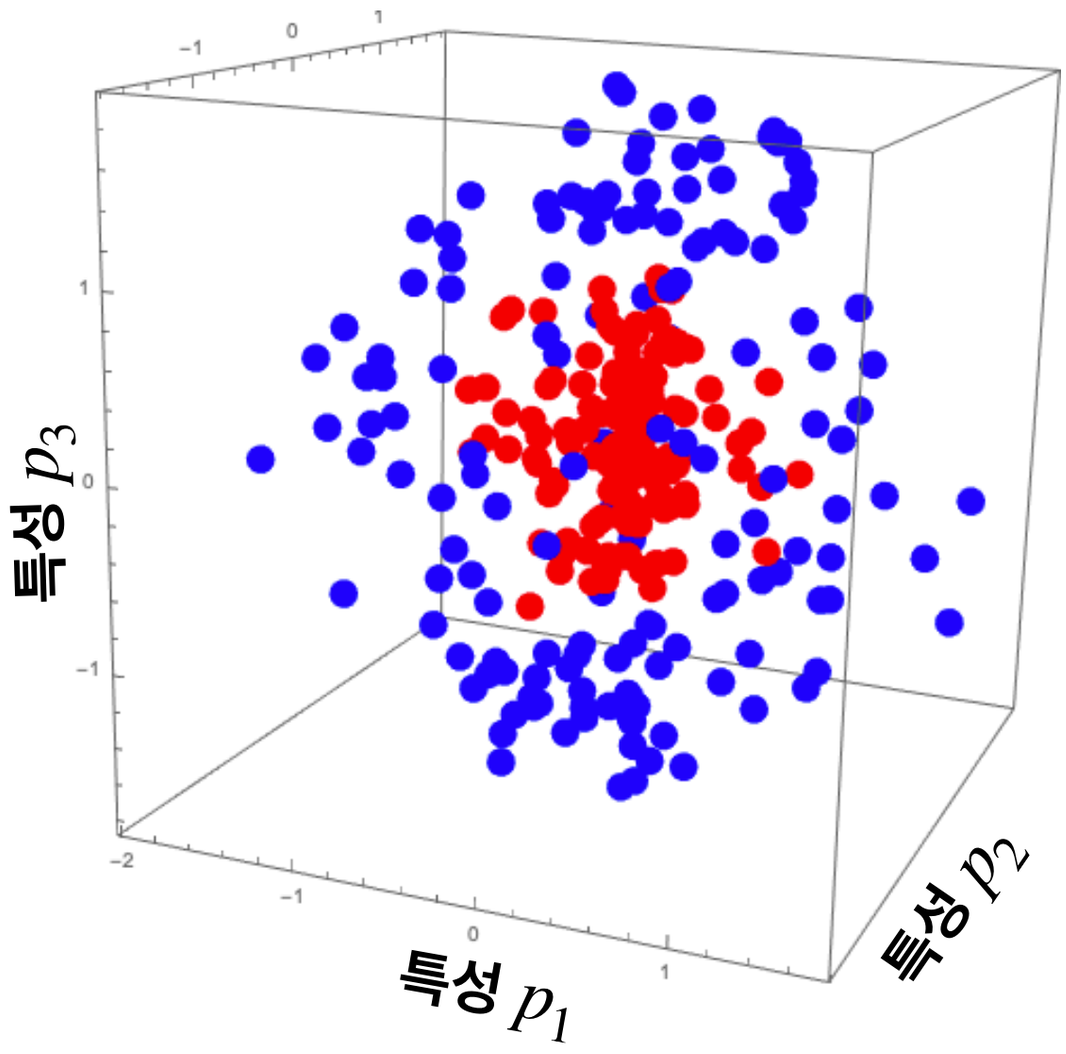
- 일반적인 격자 방법으로 두 점사이의 거리 d 를 유지시키려면
  n차원에서는 필요한 데이터의 개수는 d^n 으로 증가한다!

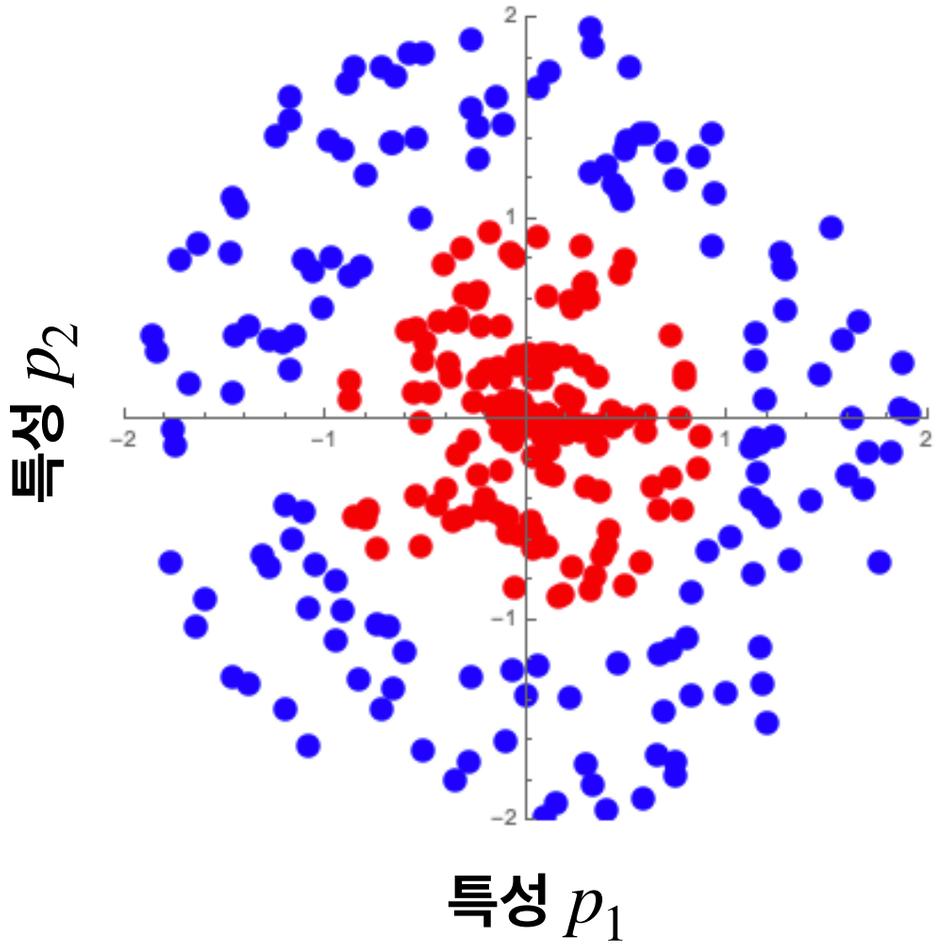
$$P||E_{\text{training}}(f_{\text{estimator}}) - E_{\text{test}}(f_{\text{estimator}})| > \epsilon|| \leq N_{\text{hypothesis}} e^{-2\epsilon^2 N_{\text{training}}}$$

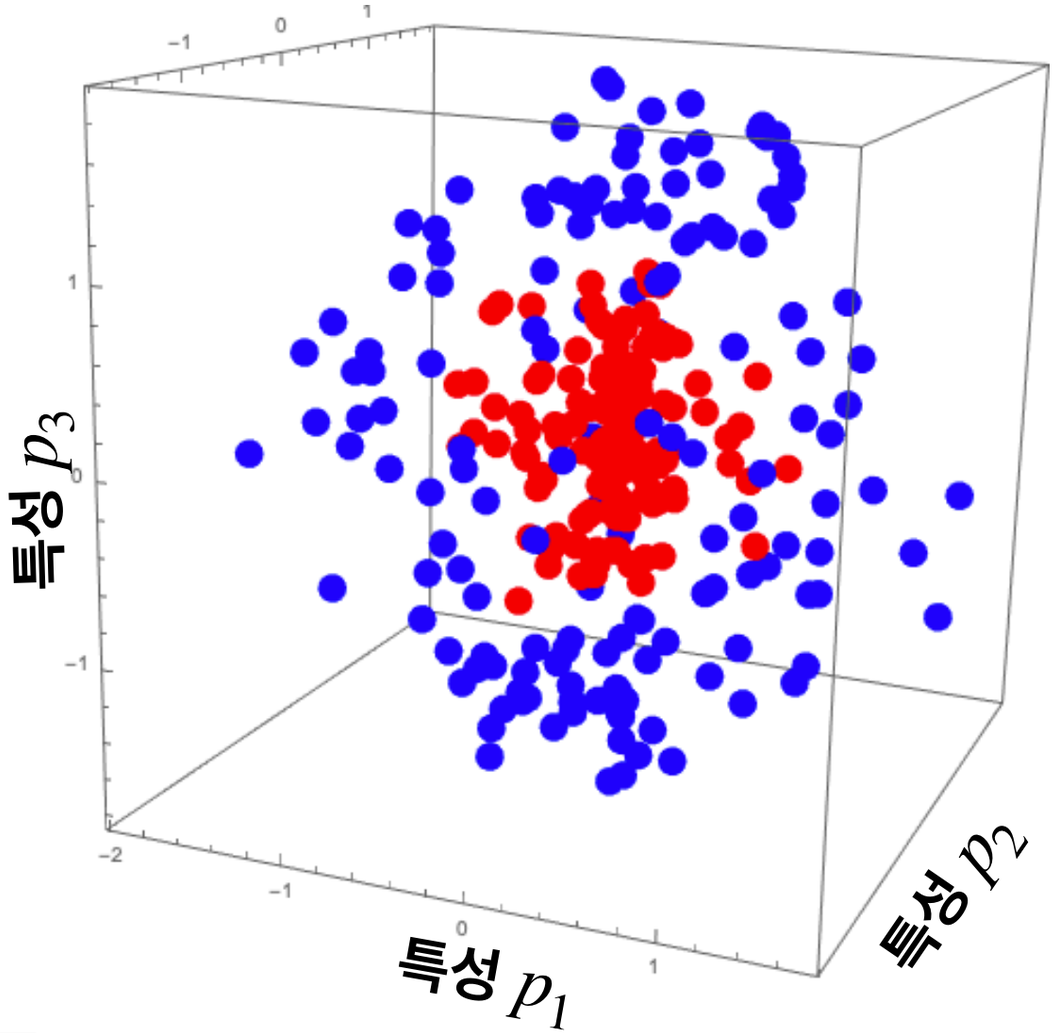
-"Hoeffding inequality"

- 차원이 높을 수록, 특성에 따른 분류가 어려워짐

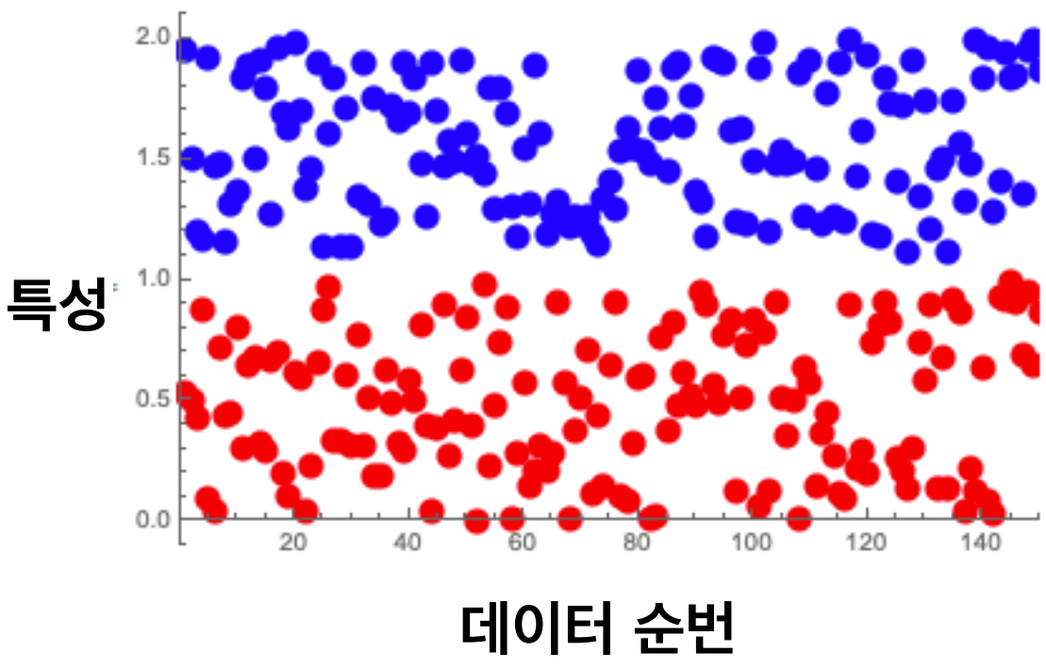


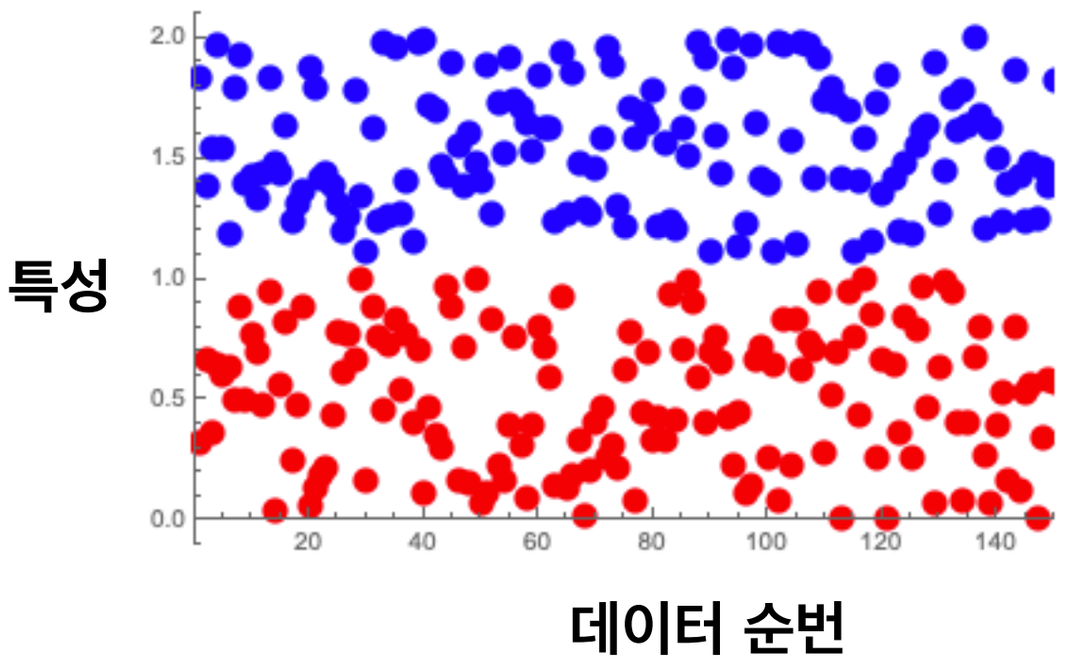






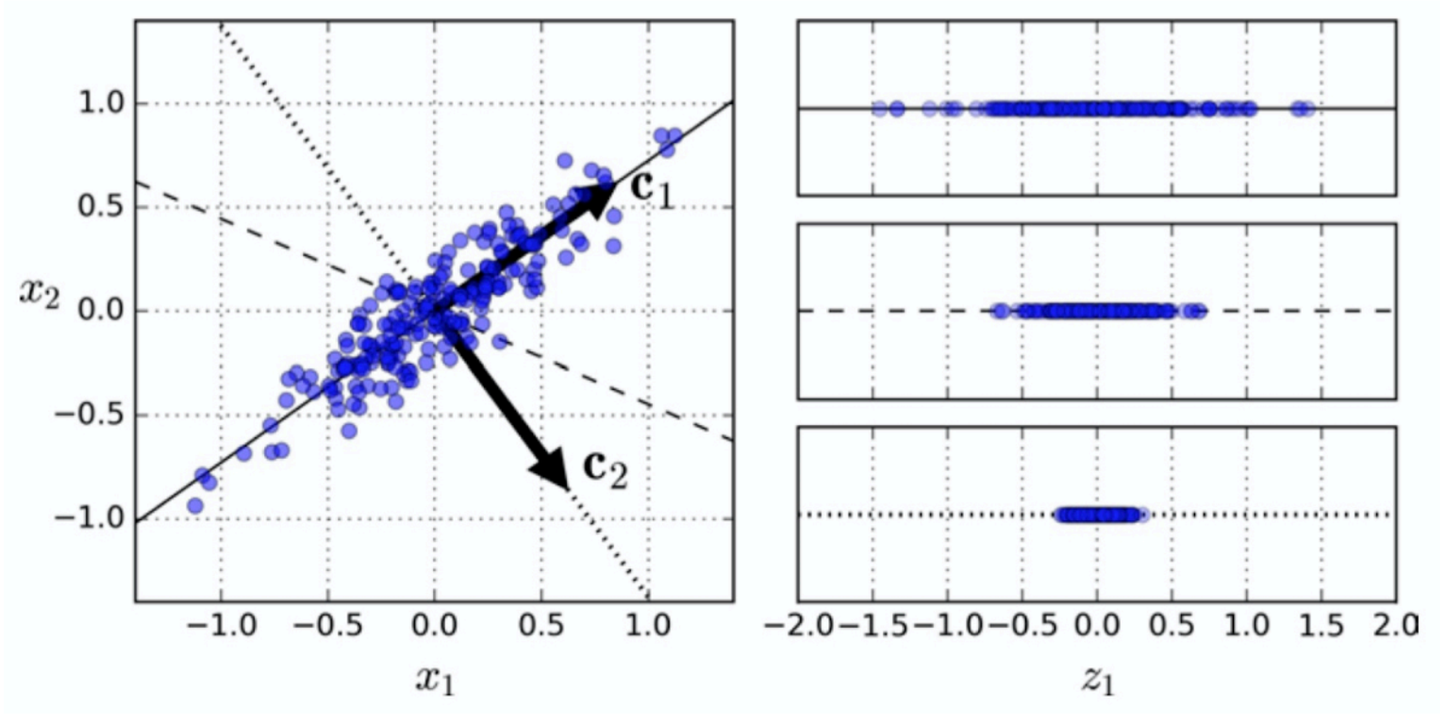
- 가장 중요 특성 $p = \sqrt{p_1^2 + p_2^2 + p_3^2}$ 고려하면, 분류가 상당히 쉬워짐





- 불필요한 차원을 (컴퓨터가) 줄임으로써, 특성을 쉽게 파악하는 방법

- Principal Component Analysis (PCA)

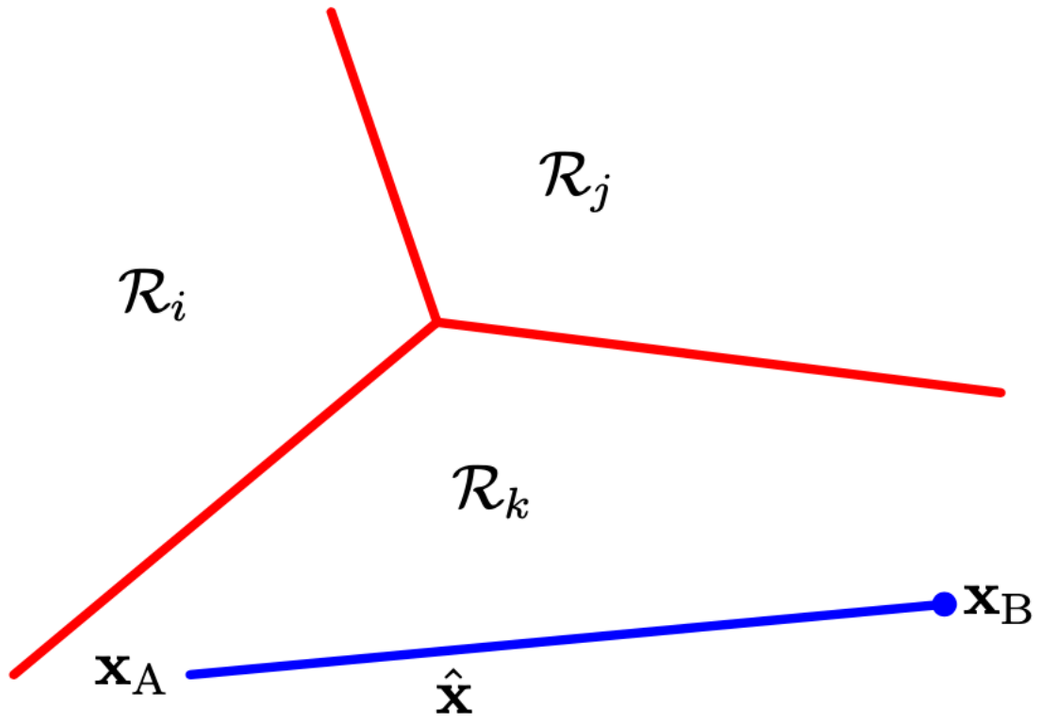


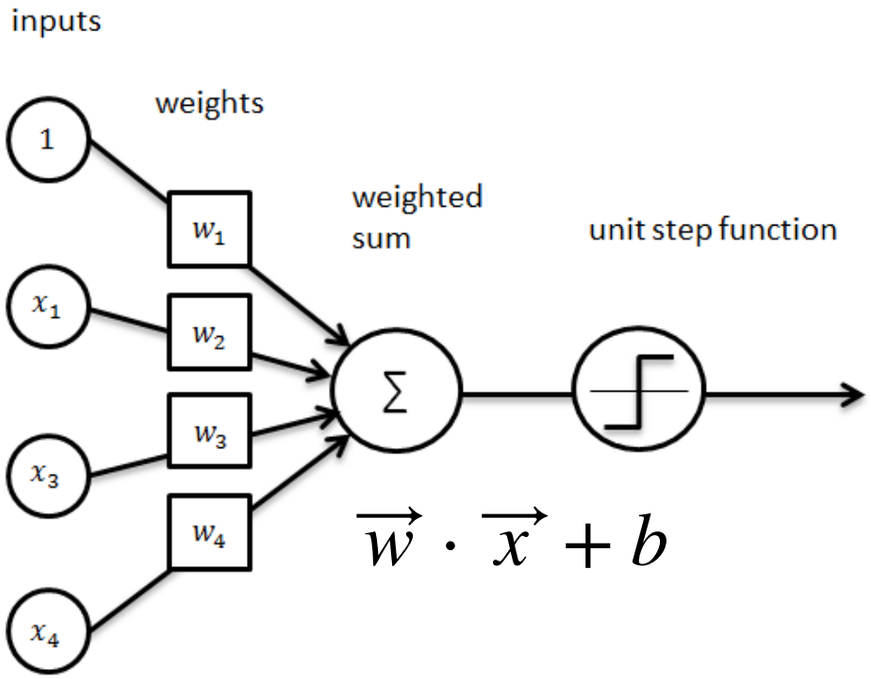
- Manifold learning

Kilian Q. Weinberger et.al. 2005

퍼셉트론 (Perceptron)

- 1958년 프랑크 로젠블라트(Rosenblatt)가 미해군의 지원을 받아 개발한 **선형 판별 모델** (클래스들이 **선형 결정 표면들을 바탕으로 정확하게 나누어** 질 수 있는 모델)



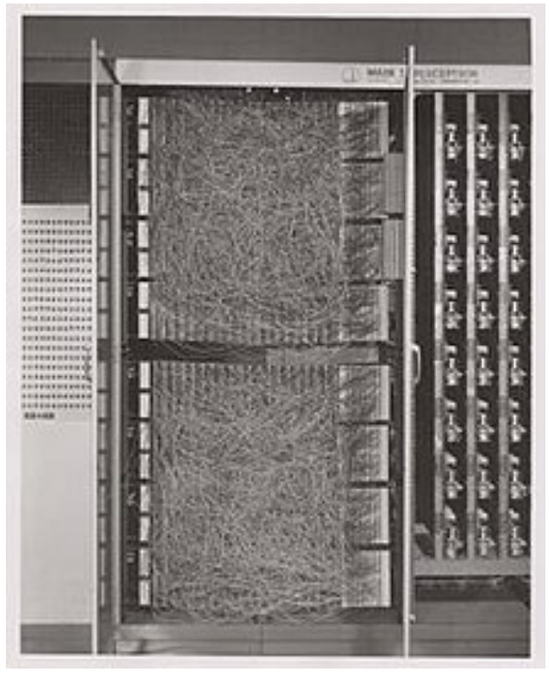


$$h\left(\vec{x}\right) = \textbf{sign}\left[\left(\sum_{i=1}^{d} w_i x_i\right) + b\right]$$

$\{x_i\} \equiv \vec{x}$: d개의 특성을 갖는 input

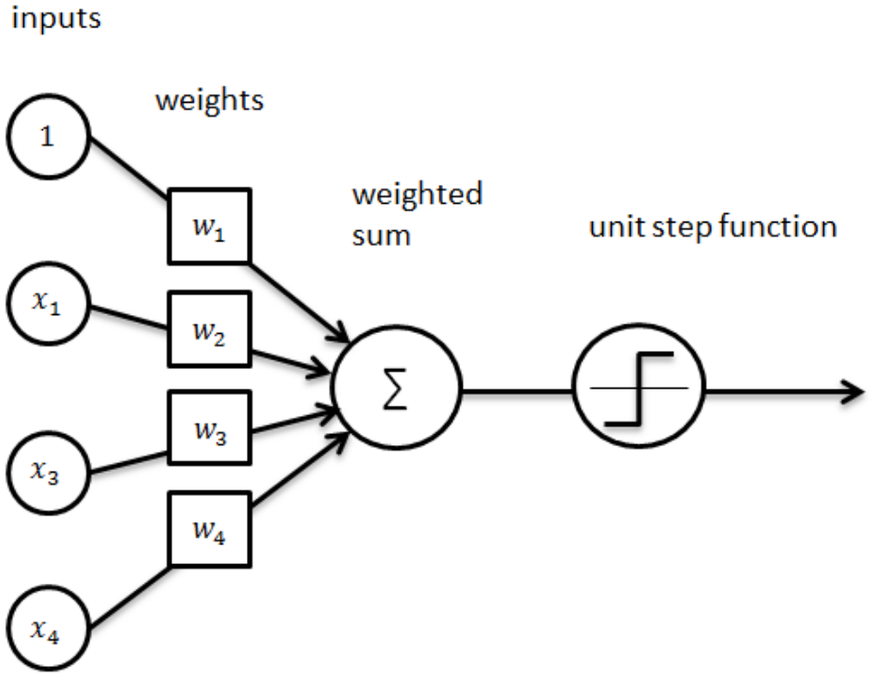
$\{w_i\} \equiv \vec{w}$: 가중치

b : 편향



- Mark 1 퍼셉트론:
  입력되는 이미지 (글자) 는 20×20,
  즉 400개의 포토셀을 통해 들어오고
  이 신호가 전선을 통해 뉴론들에 연결됨.

  각 가중치 w_i 는 회전식 가변저항(전기모터로 조절)을 통해 주어짐.



$$h\left(\vec{x}\right) = \textbf{sign}\left[\left(\sum_{i=1}^{d} w_i x_i\right) + b\right] = \textbf{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$$

$$\vec{w} = (w_1, w_2, \cdots, w_n)$$

$$\vec{x} = (x_1, x_2, \cdots, x_n)$$

$$\vec{w} = (w_0, w_1, w_2, \cdots, w_n)$$

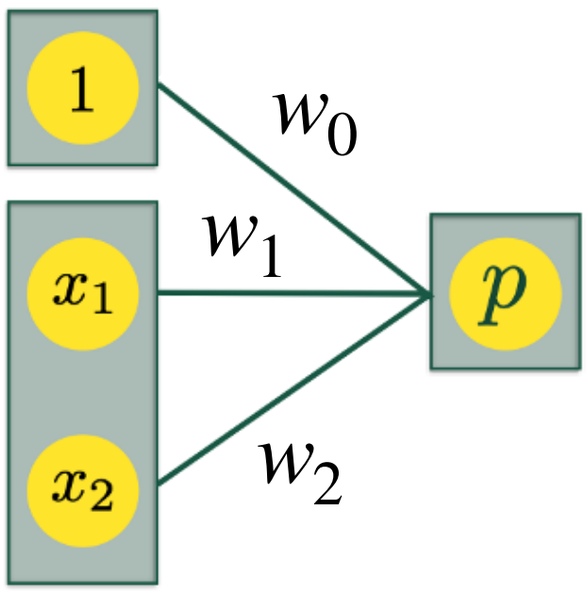
$$(w_0 = b)$$

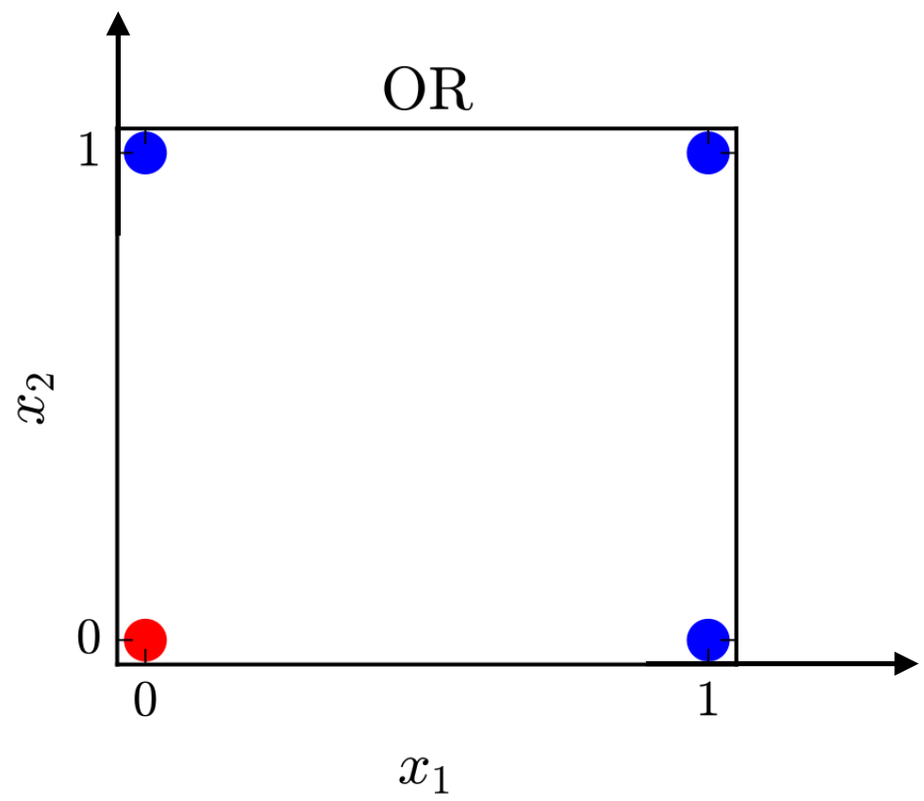
$$\vec{x} = (1, x_1, x_2, \cdots, x_n)$$

- 직선 (선형관계식)을 사용하여, OR-gate를 구현.

OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

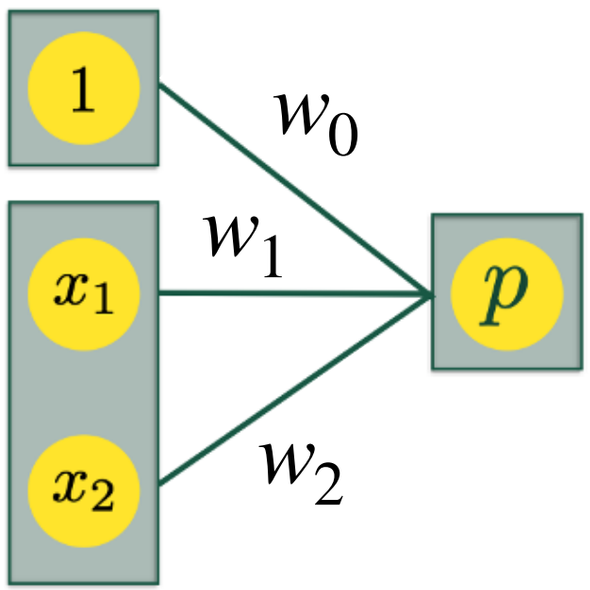




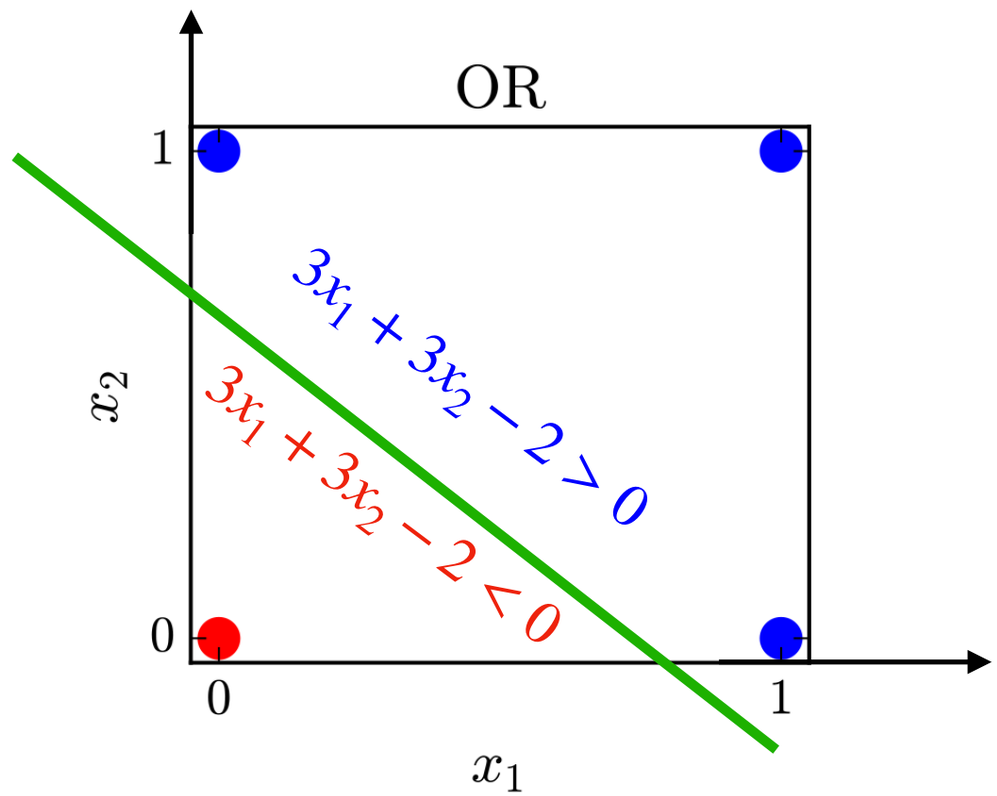
- 직선 (선형관계식)을 사용하여, OR-gate를 구현.

OR

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1



$$(w_0, w_1, w_2) = (-2,3,3)$$



$$h(\vec{x}) = \textbf{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right] = \textbf{sign}\left[3x_1 + 3x_2 - 2\right]$$

$3 \times 0 + 3 \times 0 - 2 = -2, \quad \textbf{sign} = -1$

$3 \times 0 + 3 \times 1 - 2 = 1, \quad \textbf{sign} = +1$

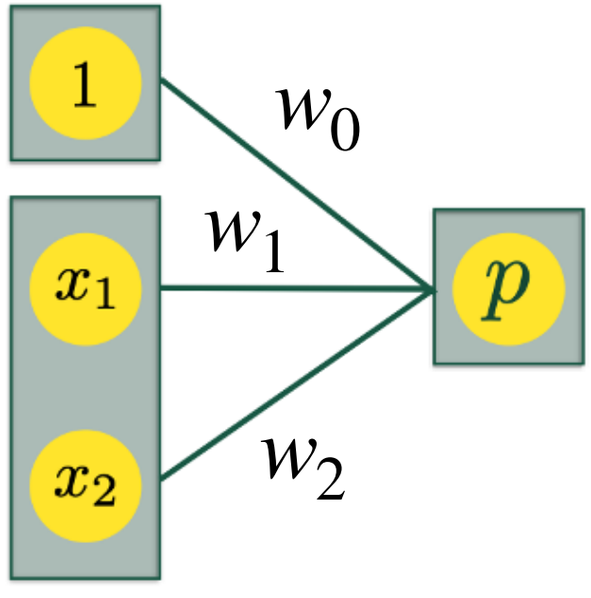
$3 \times 1 + 3 \times 0 - 2 = 1, \quad \textbf{sign} = +1$

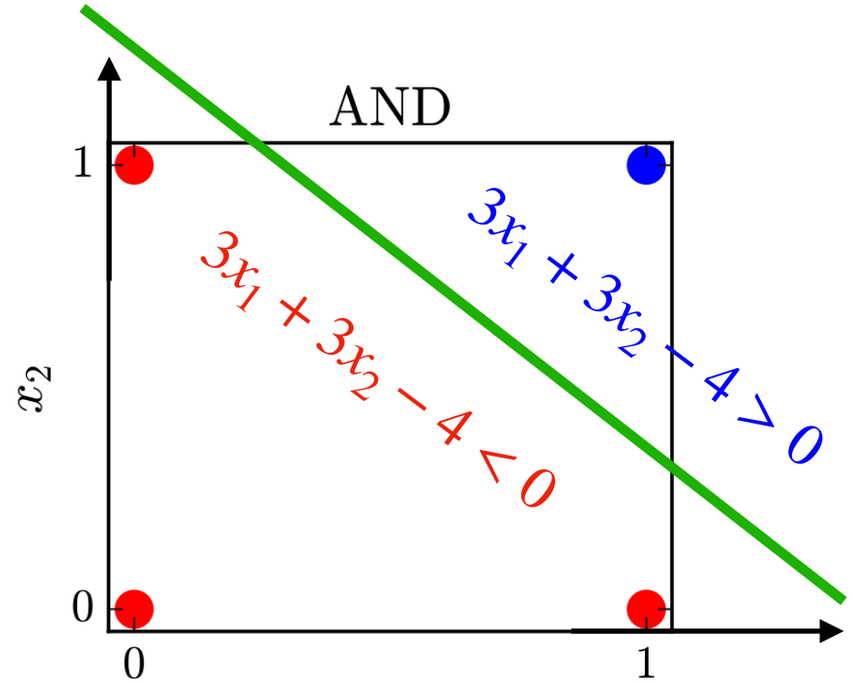
$3 \times 1 + 3 \times 1 - 2 = 4, \quad \textbf{sign} = +1$

- 직선 (선형관계식)을 사용하여, AND-gate를 구현.

AND

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1





$$(w_0, w_1, w_2) = (-4,3,3)$$

$$h(\vec{x}) = \textbf{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right] = \textbf{sign}\left[3x_1 + 3x_2 - 2\right]$$

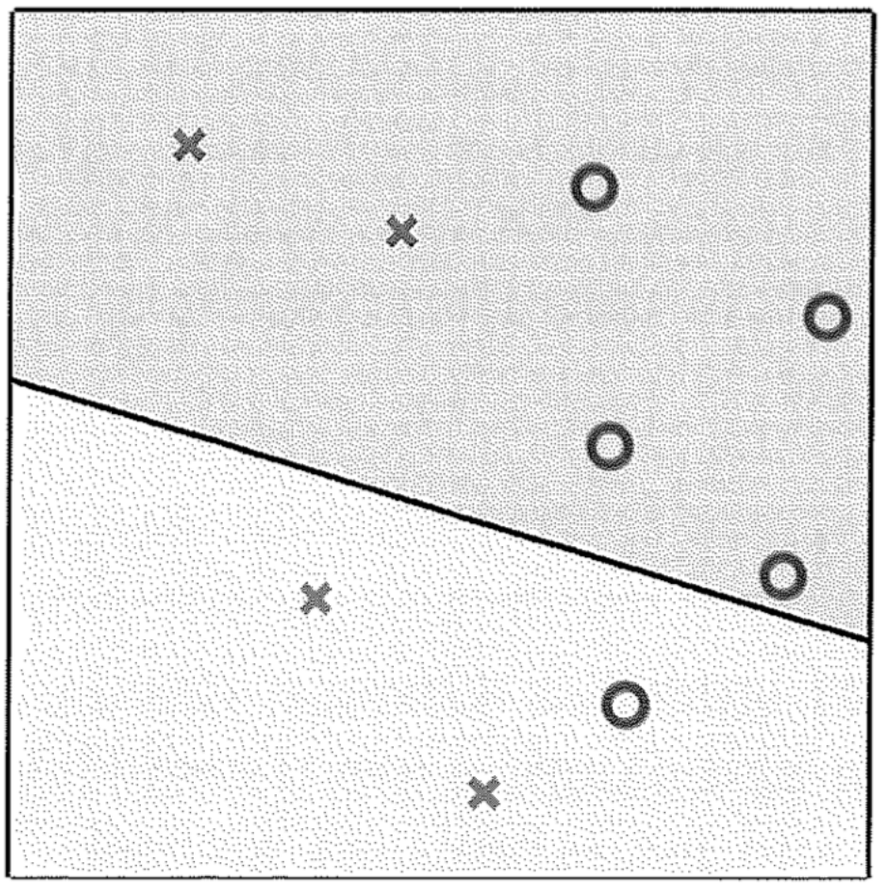
$3 \times 0 + 3 \times 0 - 4 = -4, \quad \textbf{sign} = -1$

$3 \times 0 + 3 \times 1 - 4 = -1, \quad \textbf{sign} = -1$

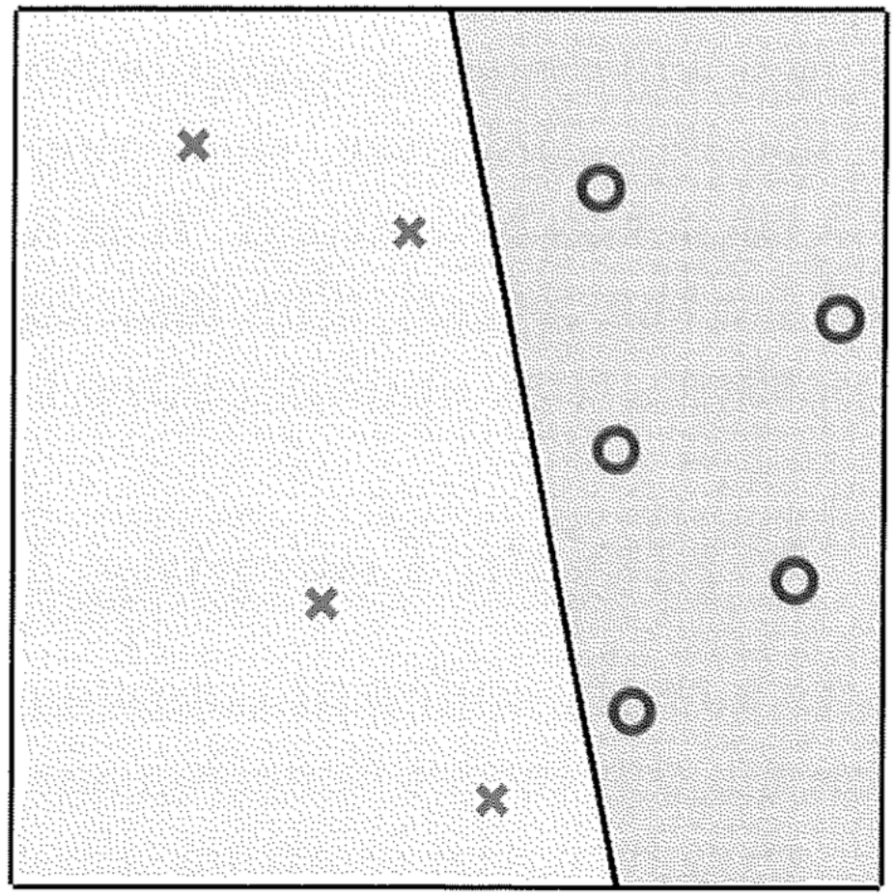
$3 \times 1 + 3 \times 0 - 4 = -1, \quad \textbf{sign} = -1$

$3 \times 1 + 3 \times 1 - 4 = 2, \quad \textbf{sign} = +1$

퍼셉트론을 이용한 지도학습



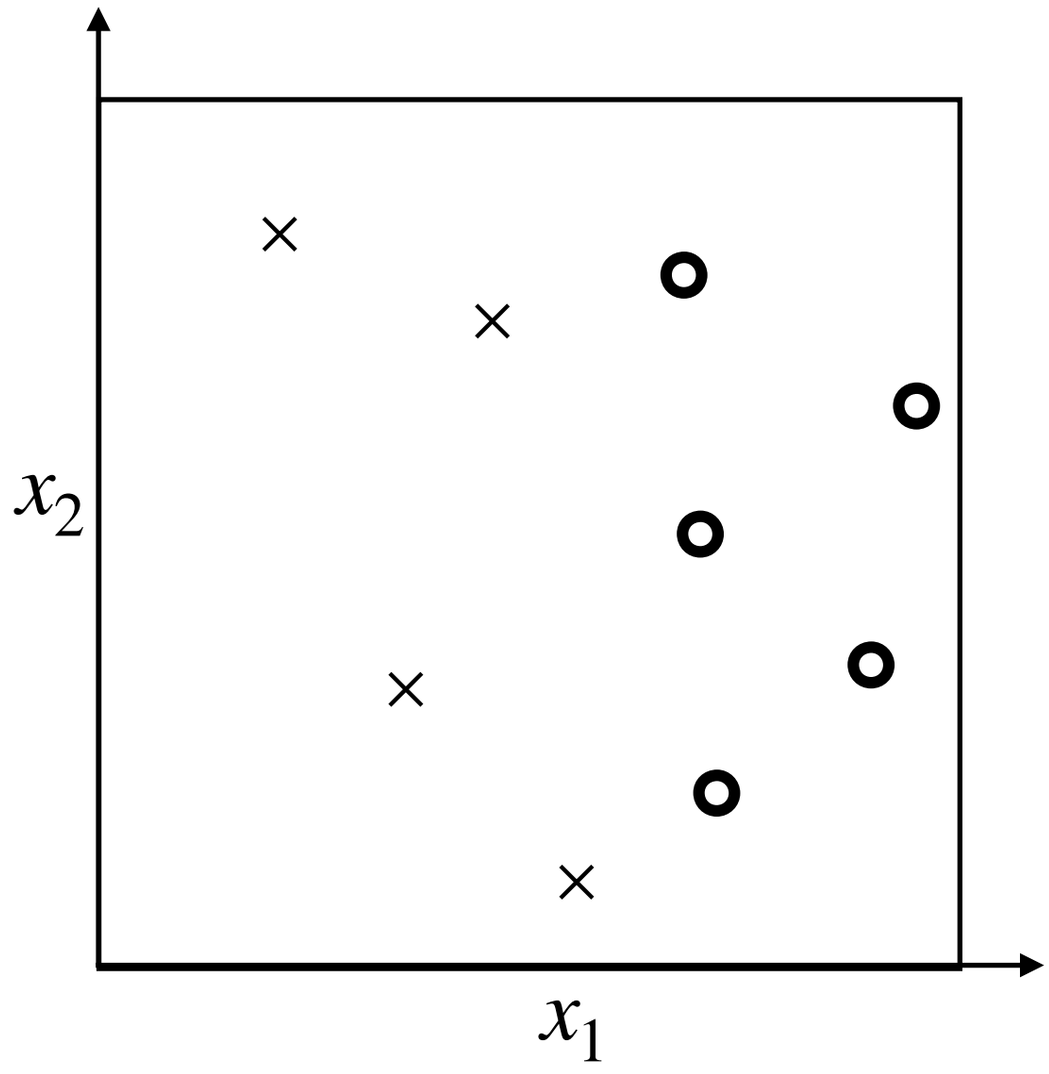
잘못된 선형 분류



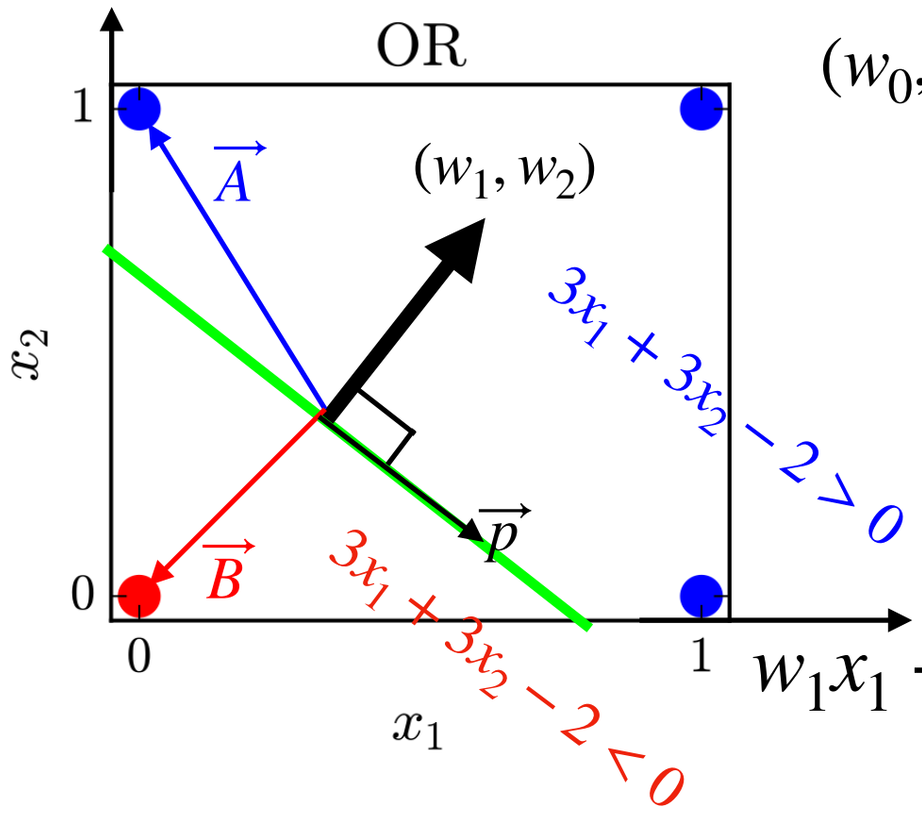
옳바른 선형 분류

- 퍼셉트론을 사용하여, 컴퓨터가 자동적으로 선형분류를 하도록 하는 방법은?

퍼셉트론을 이용한 지도학습



가중치 $\overrightarrow{w}$ 의 성질



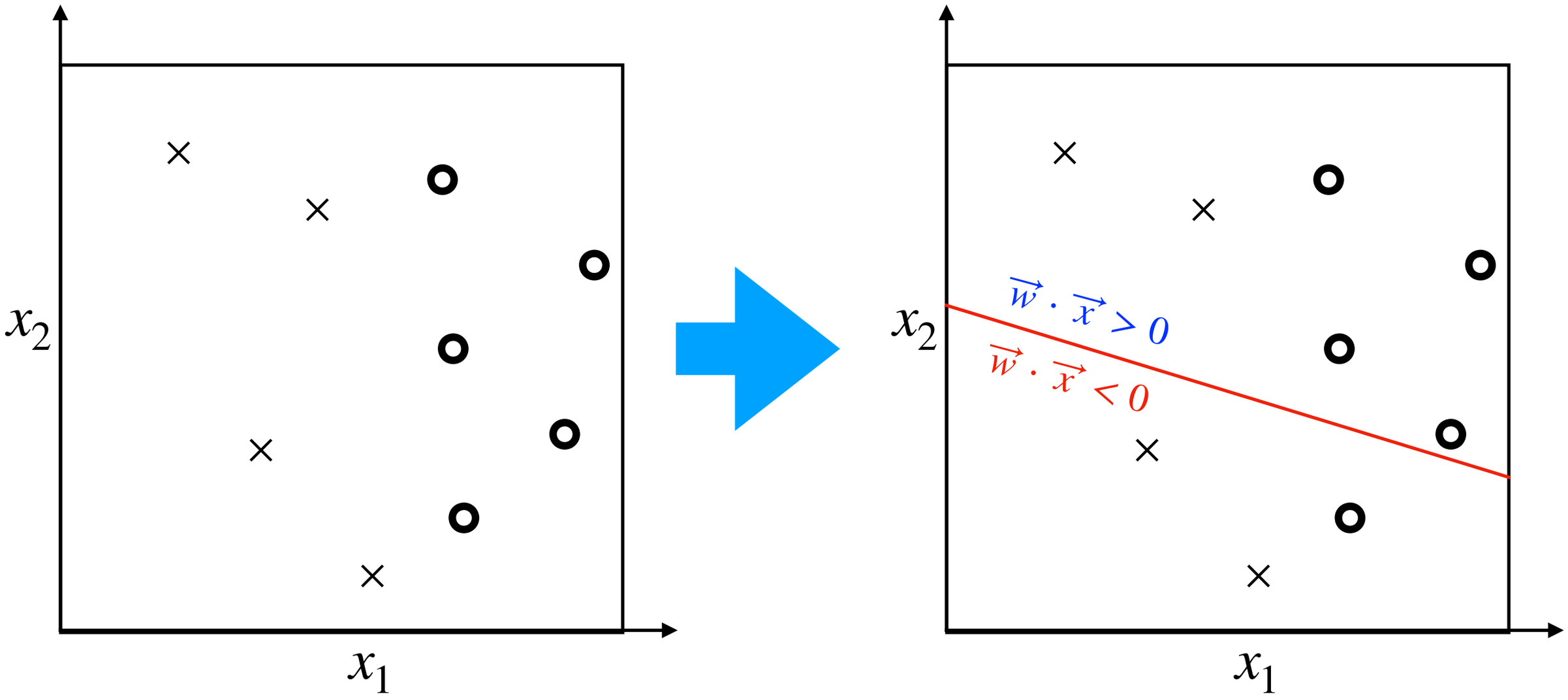
$$(w_0, w_1, w_2) = (-2,3,3)$$

$$w_1x_1 + w_2x_2 + w_0x_0 = 3x_1 + 3x_2 - 2 \times 1 = 0$$

- $\overrightarrow{W} = (w_1, w_2)$ 라 정의하면, $\overrightarrow{A} \cdot \overrightarrow{W} > 0$, $\overrightarrow{B} \cdot \overrightarrow{W} < 0$, $\overrightarrow{p} \cdot \overrightarrow{W} = 0$

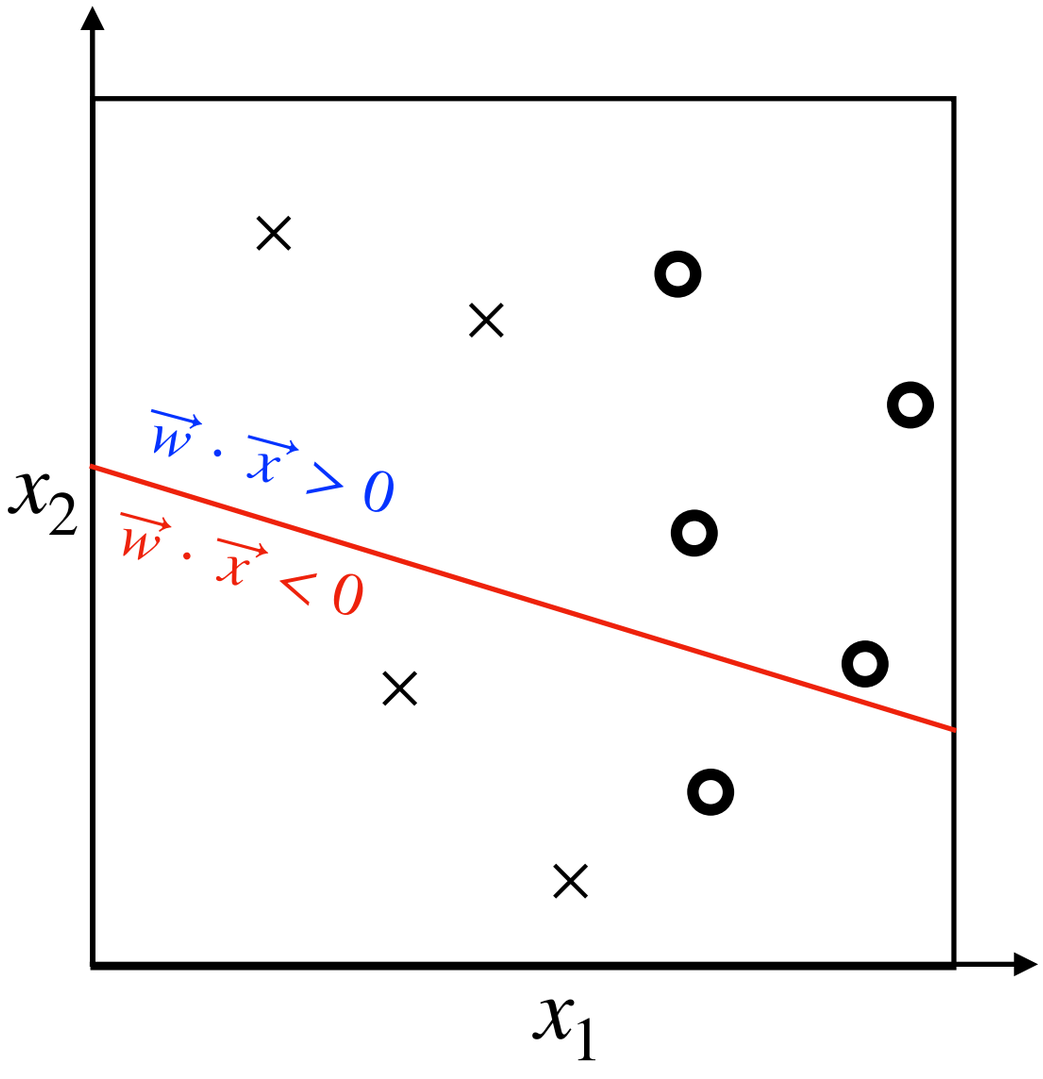
1. 임의의 $\vec{w}$ 를 설정하여 분류를 시도한다.

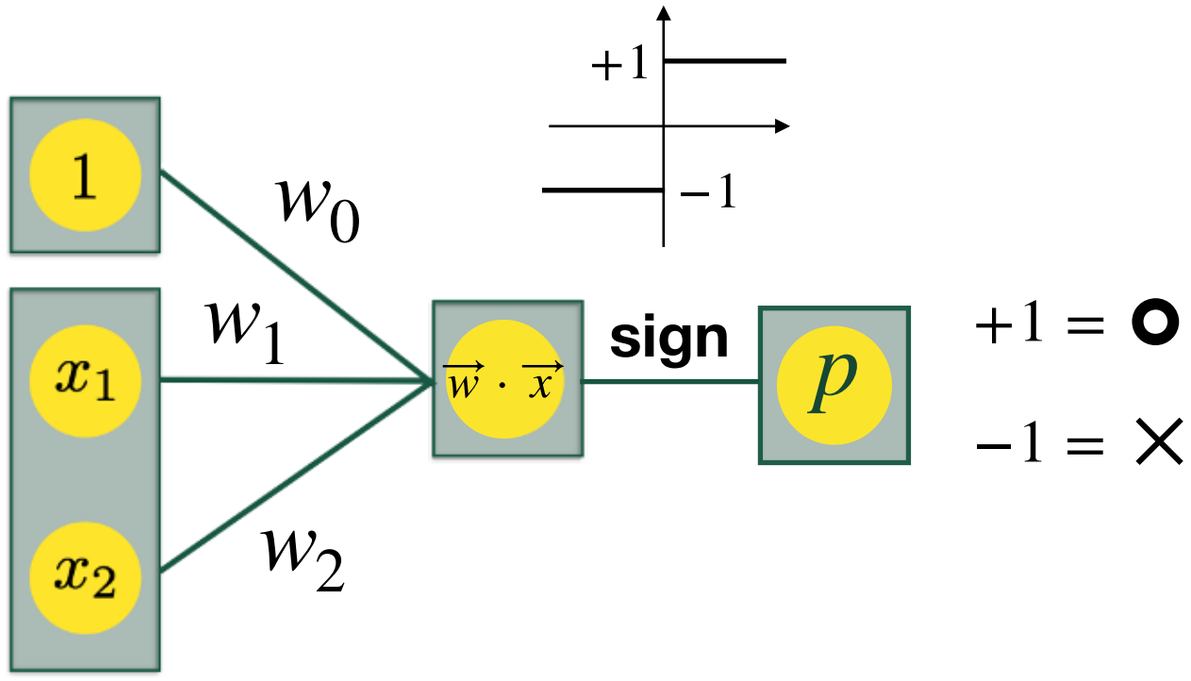
$$b + w_1x_1 + w_2x_2 = w_0 \times 1 + w_1x_1 + w_2x_2 = (w_0, w_1, w_2) \cdot (1, x_1, x_2) = \vec{w} \cdot \vec{x}$$



1. 임의의 $\vec{w}$ 를 설정하여 분류를 시도한다.

$$b + w_1x_1 + w_2x_2 = w_0 \times 1 + w_1x_1 + w_2x_2 = (w_0, w_1, w_2) \cdot (1, x_1, x_2) = \vec{w} \cdot \vec{x}$$

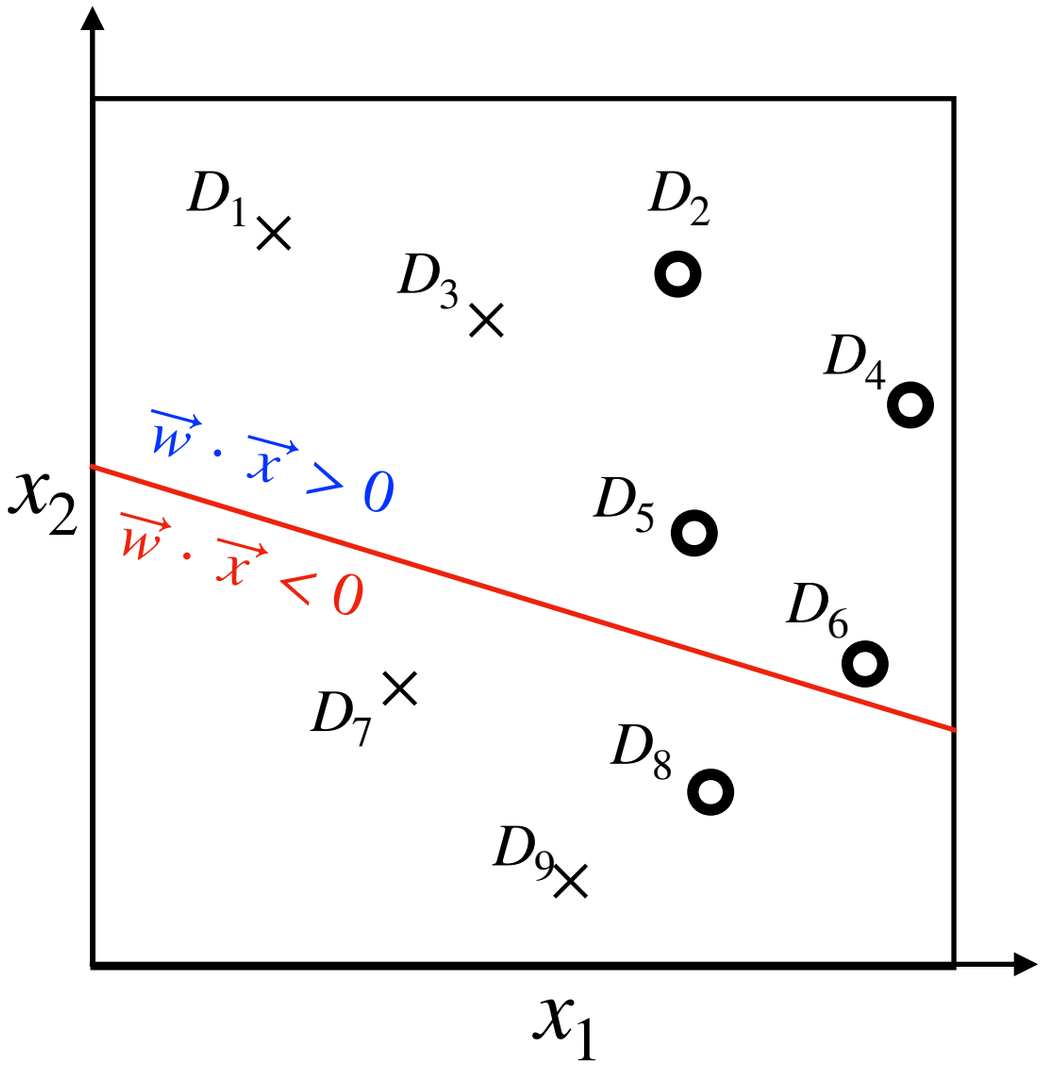




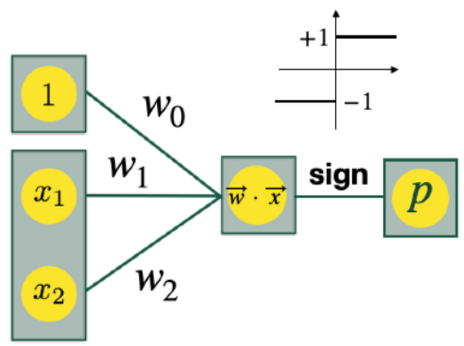
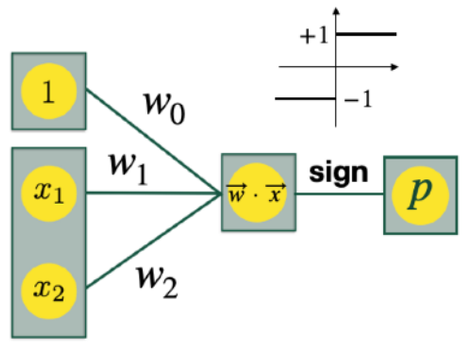
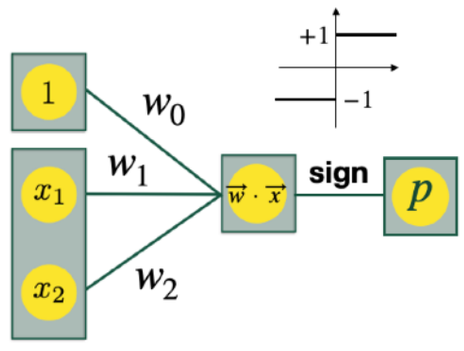
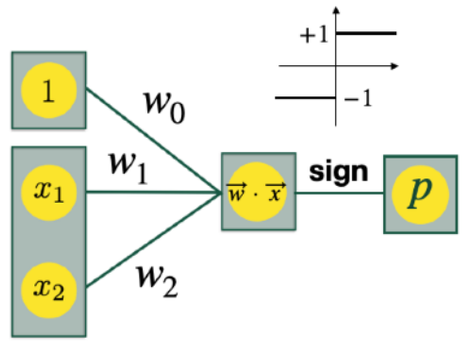
$$P(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{2} w_i x_i\right)\right]$$

2. 라벨을 활용하여, 퍼셉트론의 계산 값과 비교를 한다.

$$P(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{2} w_i x_i\right)\right]$$

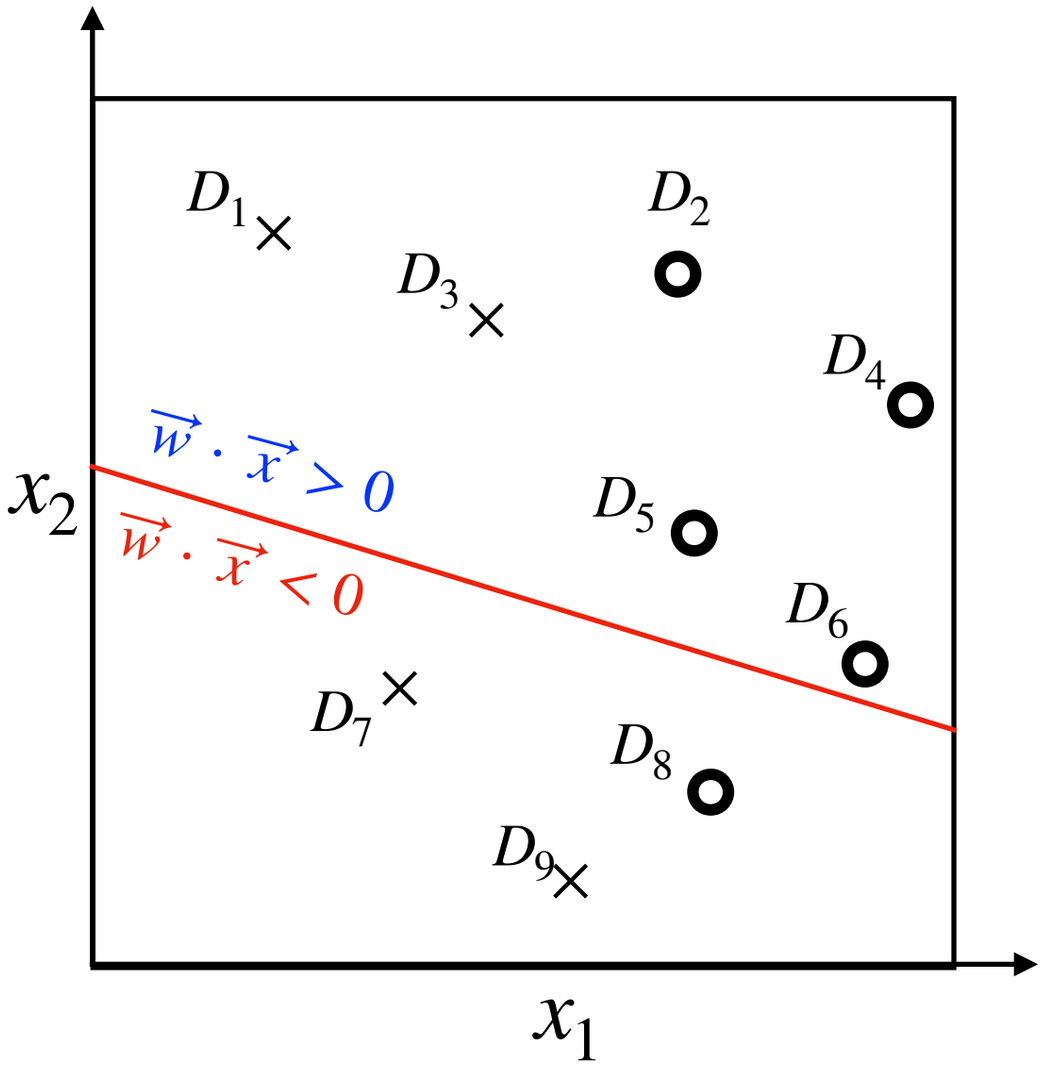


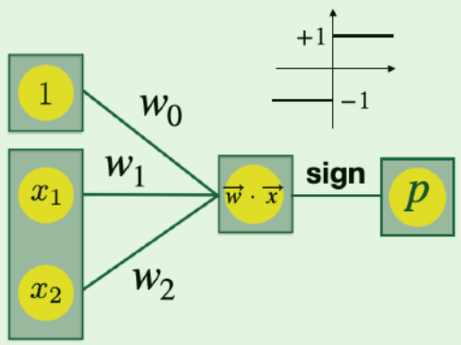
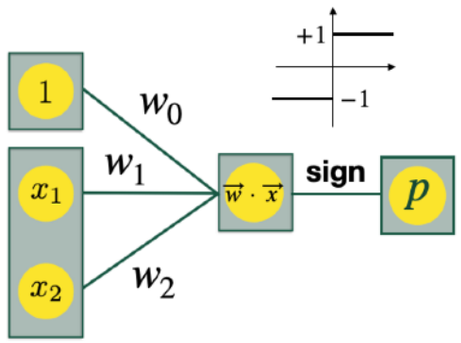
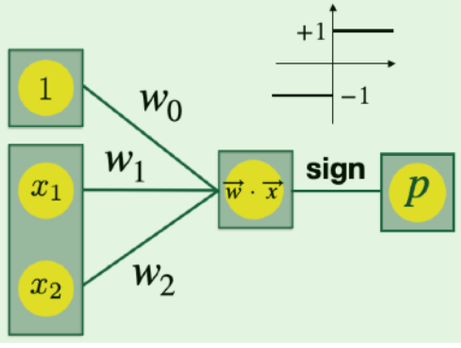
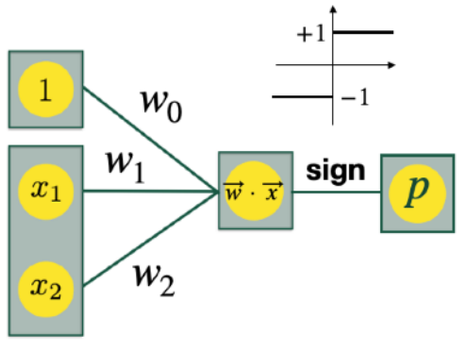
(지도학습)

데이터 D_i	퍼셉트론 계산	계산 값 P_i	실제 라벨 y_i
D_1		$+1$	-1
D_2		$+1$	$+1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
D_8		-1	$+1$
D_9		-1	-1

3. 틀린 답을 확인하고, 이를 반영하여 $\vec{w}$ 를 고친다.

(지도학습)



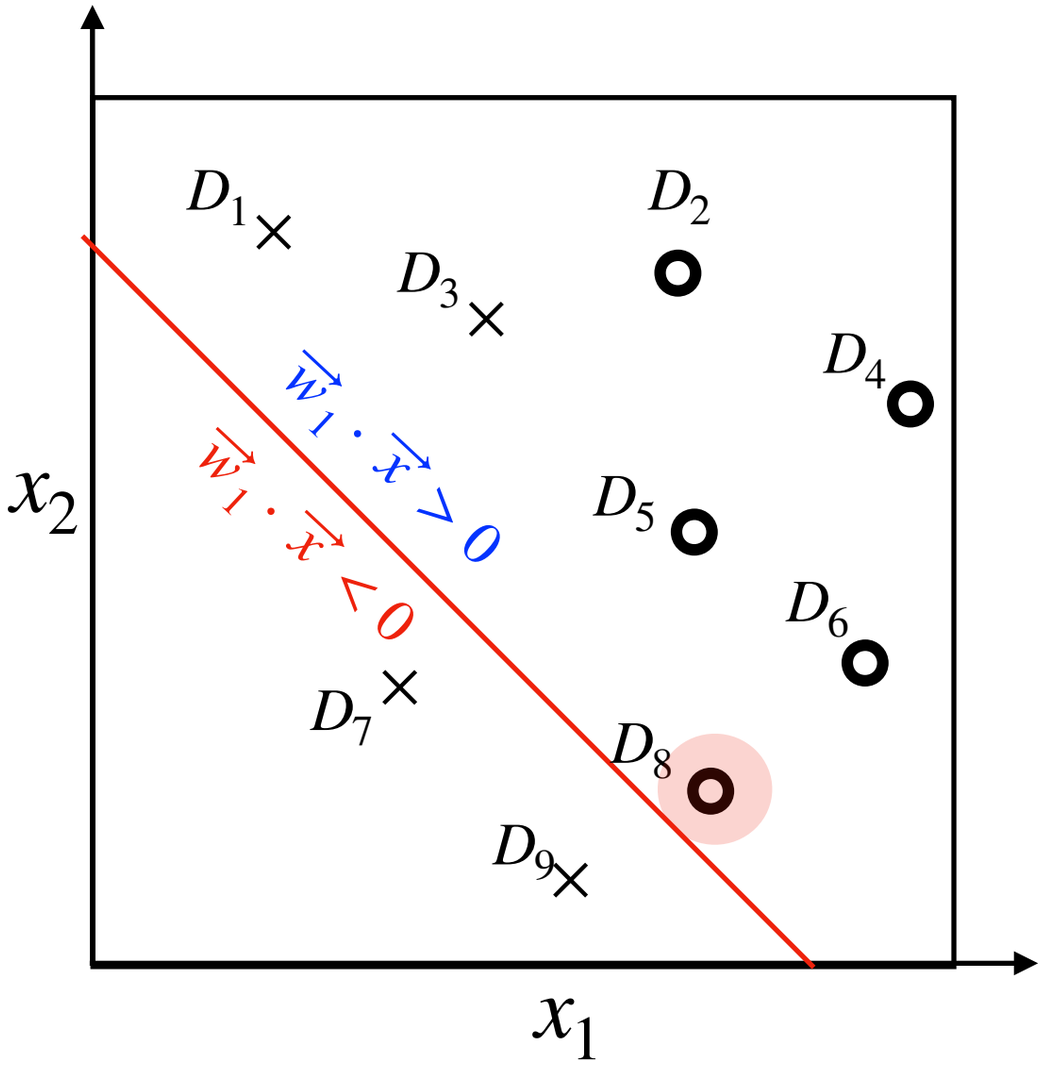
데이터 D_i	퍼셉트론 계산	계산 값 P_i	실제 라벨 y_i
D_1		$+1$	-1
D_2		$+1$	$+1$
•••	•••	•••	•••
D_8		-1	$+1$
D_9		-1	-1

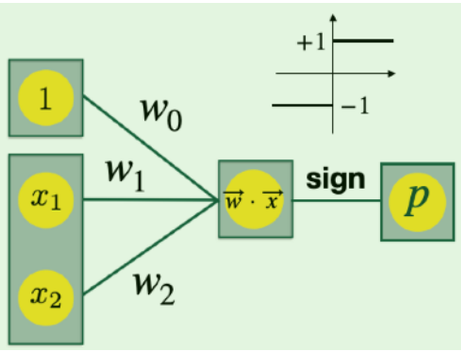
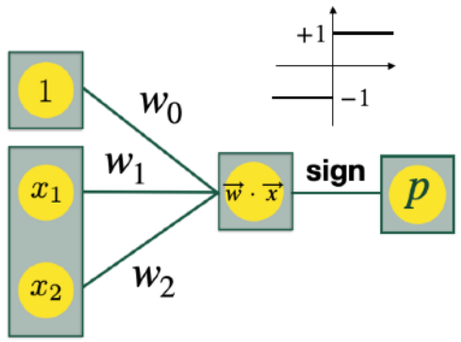
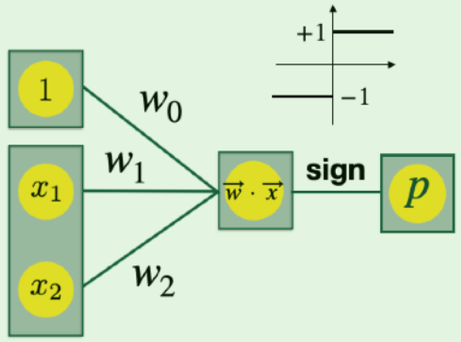
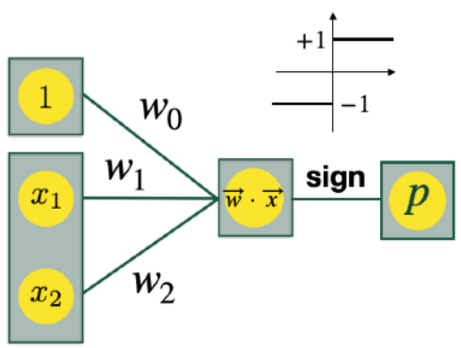
3. 틀린 답을 확인하고, 이를 반영하여 $\vec{w}$ 를 업데이트 한다.

(지도학습)

확률적 방법 (Stochastic method)

$\{D_1, D_3, D_8\}$중에 random 하게 한개를 골라서, 가중치를 $\vec{w} \rightarrow \vec{w}_1$ 로 업데이트



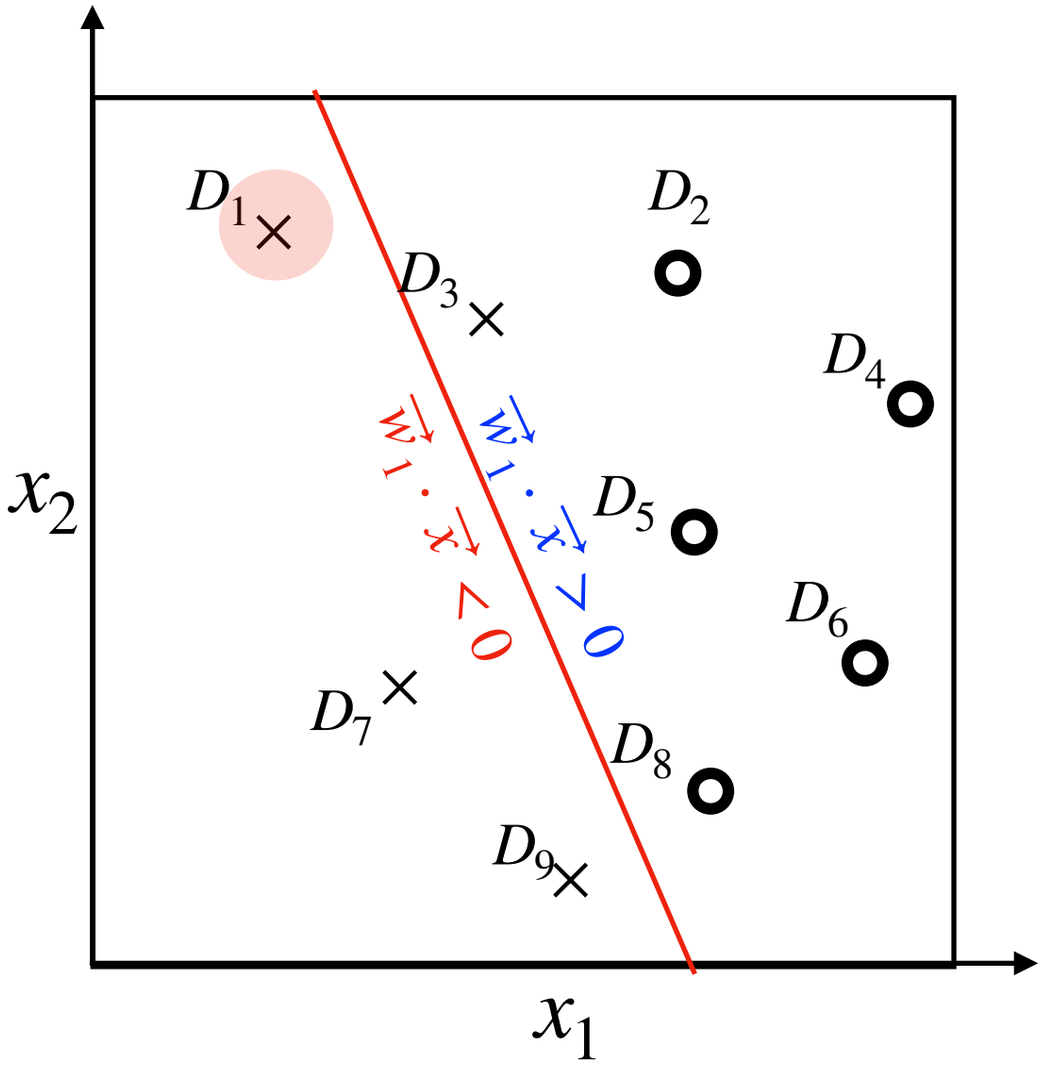
데이터 D_i	퍼셉트론 계산	계산 값 P_i	실제 라벨 y_i
D_1		$+1$	-1
D_2		$+1$	$+1$
...	...	...	...
D_8		-1	$+1$
D_9		-1	-1

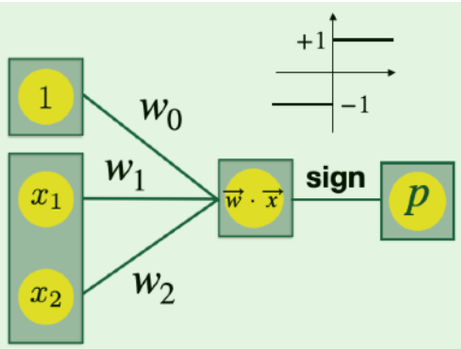
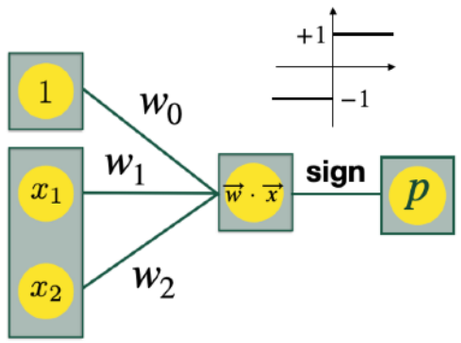
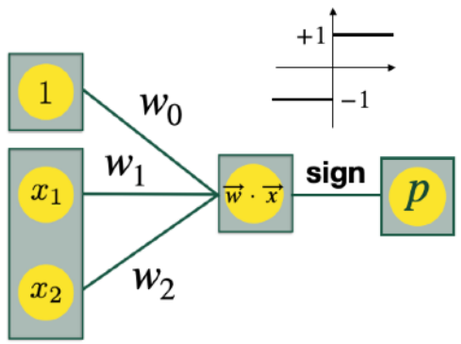
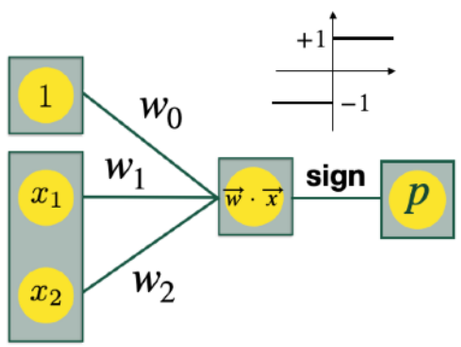
3. 틀린 답을 확인하고, 이를 반영하여 $\vec{w}$ 를 업데이트 한다.

(지도학습)

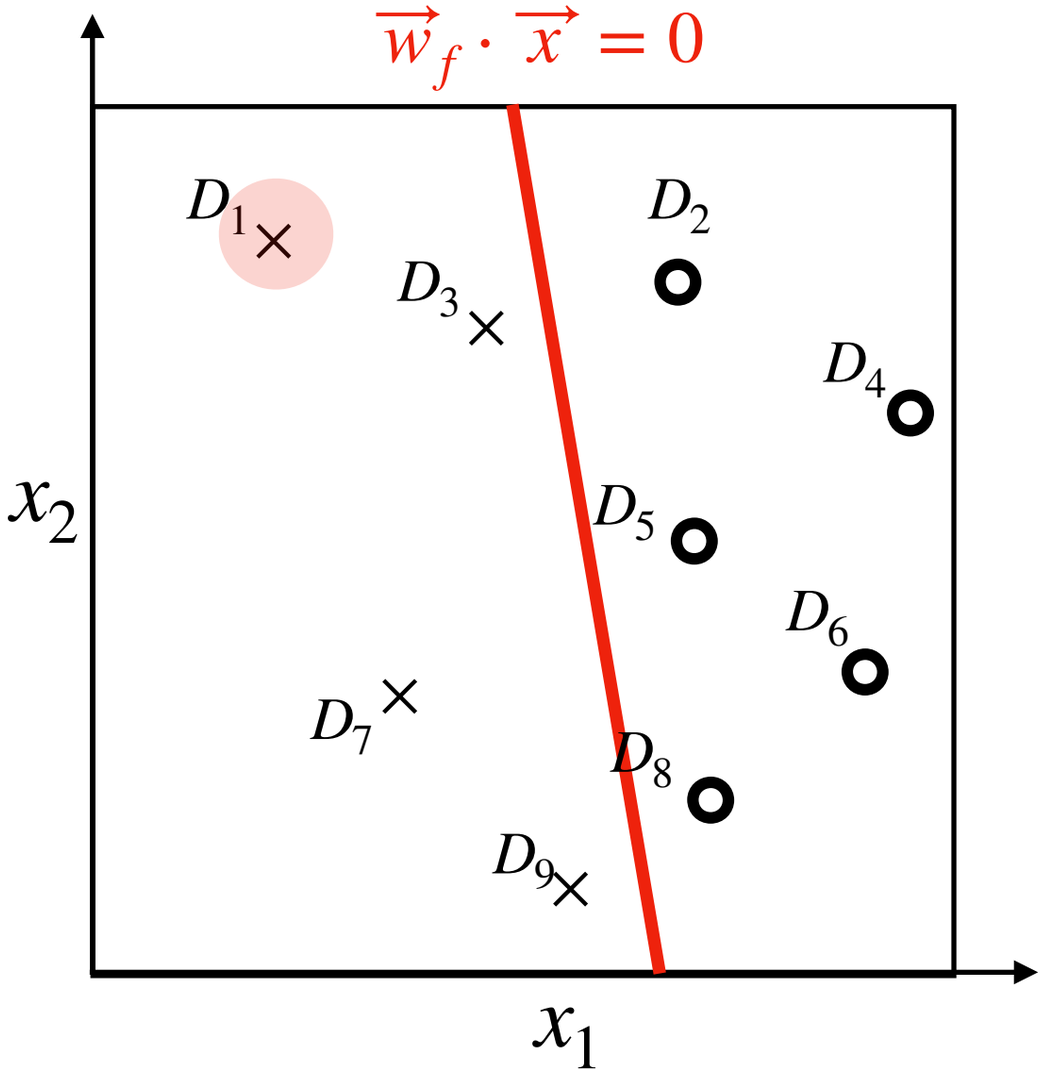
확률적 방법 (Stochastic method)

$\{D_1, D_3\}$중에 random 하게 한개를 골라서, 가중치를 $\vec{w}_1 \rightarrow \vec{w}_2$ 로 업데이트

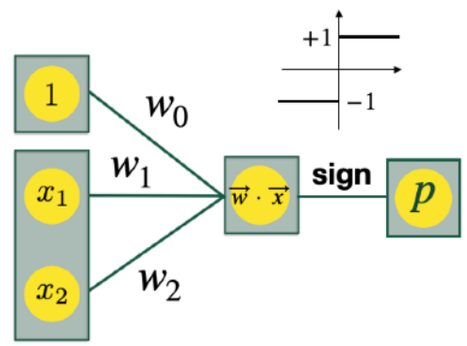
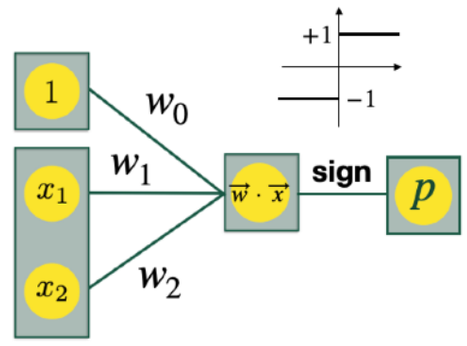
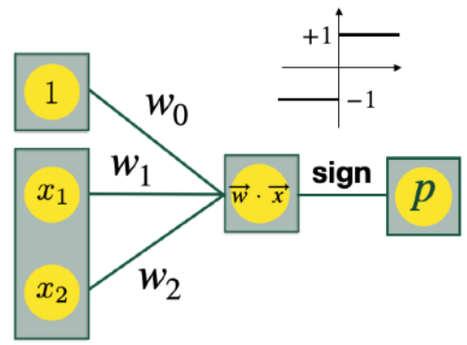
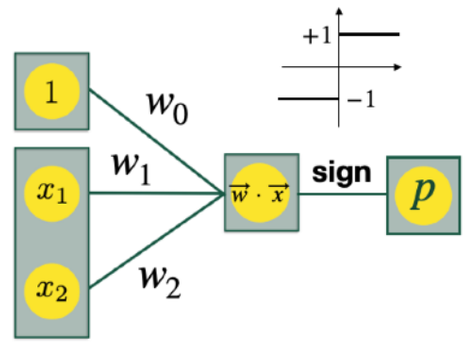


데이터 D_i	퍼셉트론 계산	계산 값 P_i	실제 라벨 y_i
D_1		$+1$	-1
D_2		$+1$	$+1$
$\cdots$	$\cdots$	$\cdots$	$\cdots$
D_8		$+1$	$+1$
D_9		-1	-1

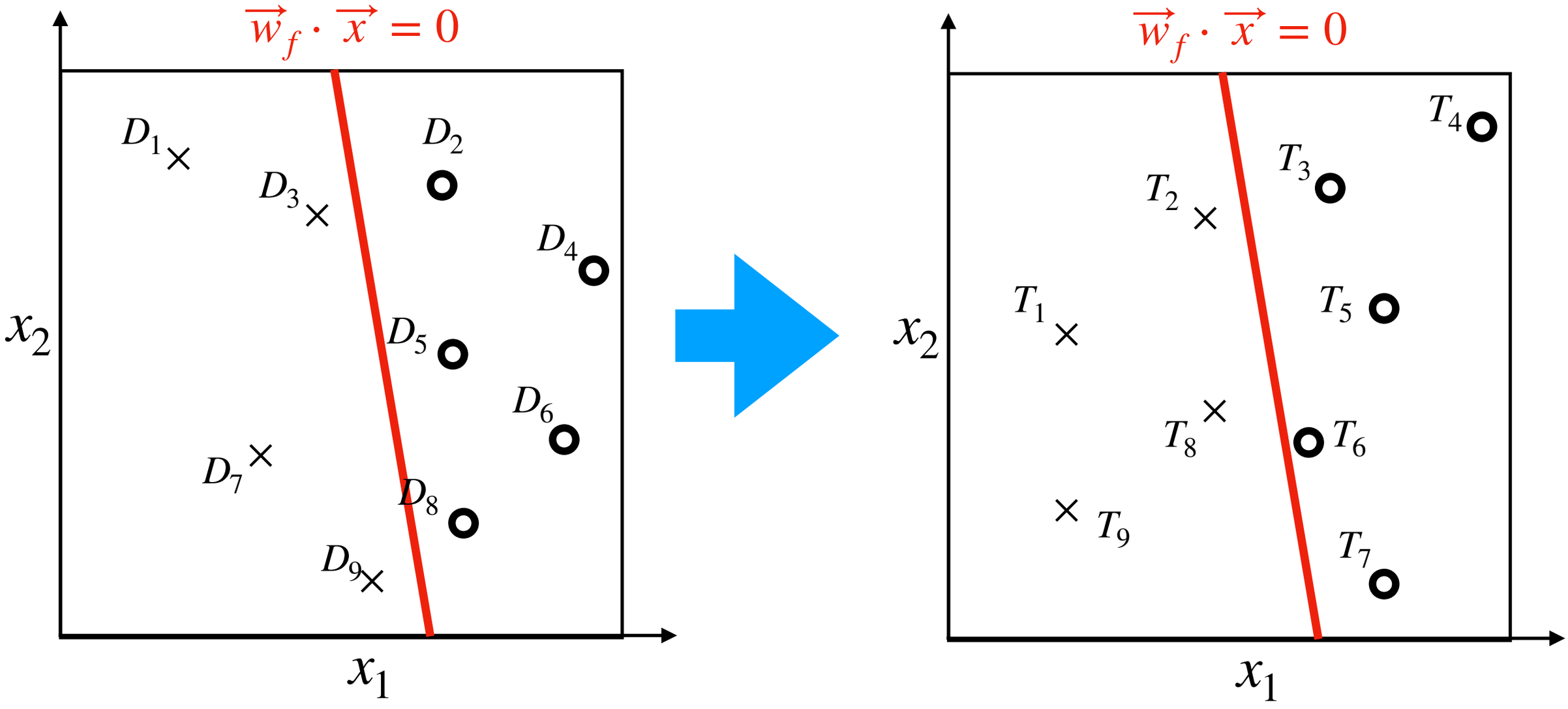
4. 최종적으로 학습 (분류)이 완성.



(지도학습)

데이터 D_i	퍼셉트론 계산	계산 값 P_i	실제 라벨 y_i
D_1		-1	-1
D_2		$+1$	$+1$
•••	•••	•••	•••
D_8		$+1$	$+1$
D_9		-1	-1

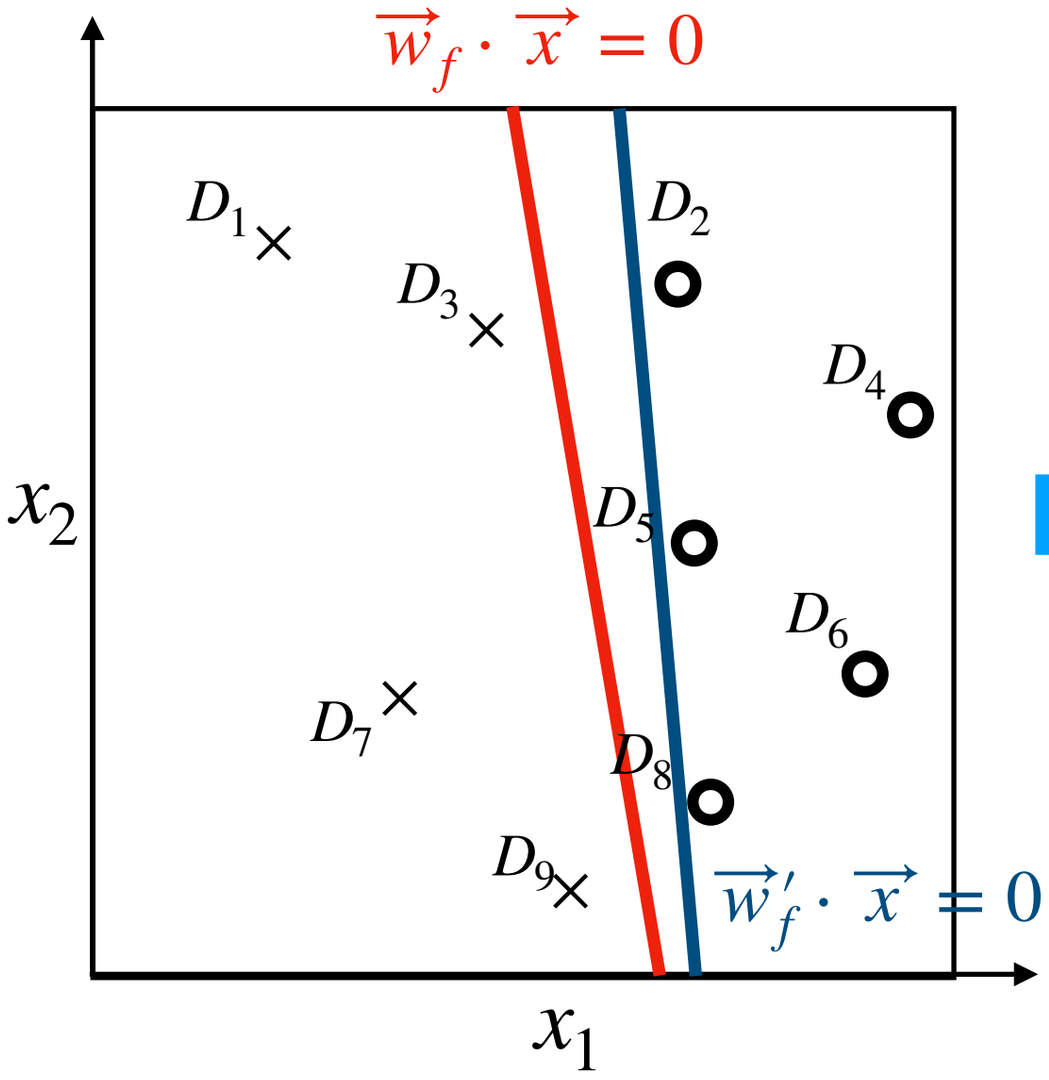
5. 학습 결과를 시험: Test data 로 학습한 결과를 테스트



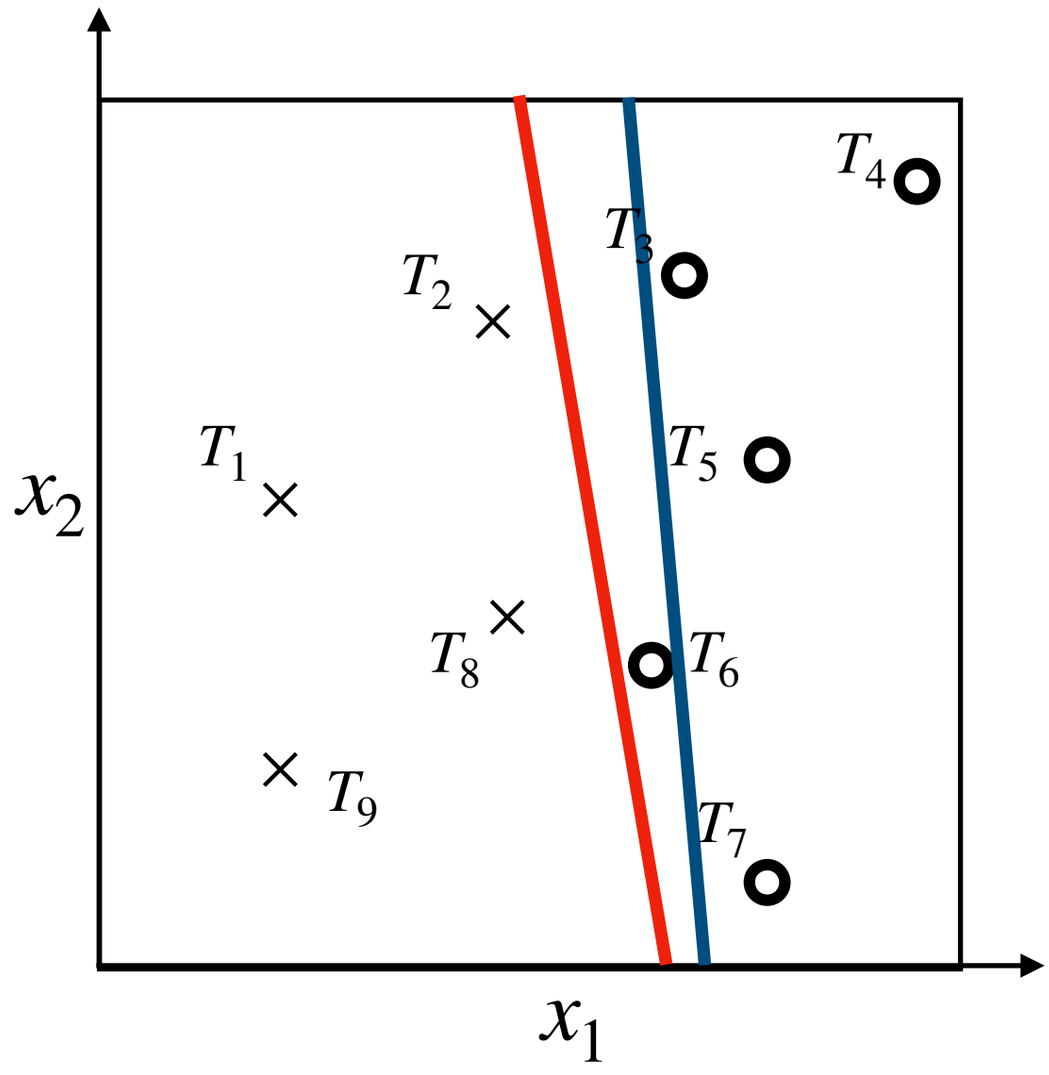
학습 **테스트 (시험)**

5. 학습 결과를 시험: Test data 로 학습한 결과를 테스트

학습을 너무 tight하게 할 경우, 시험에서 실패할 수도 있음.



학습



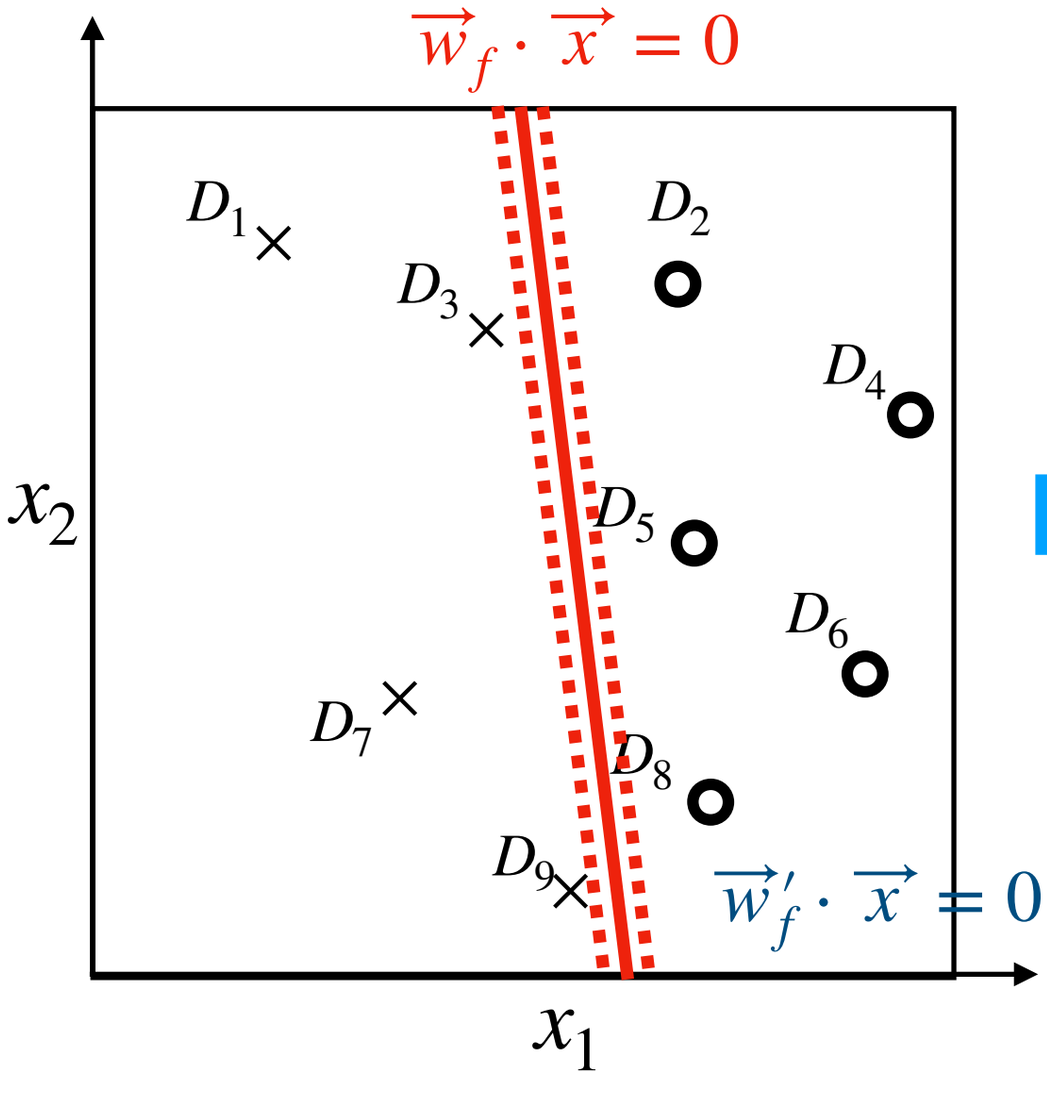
테스트 (시험)

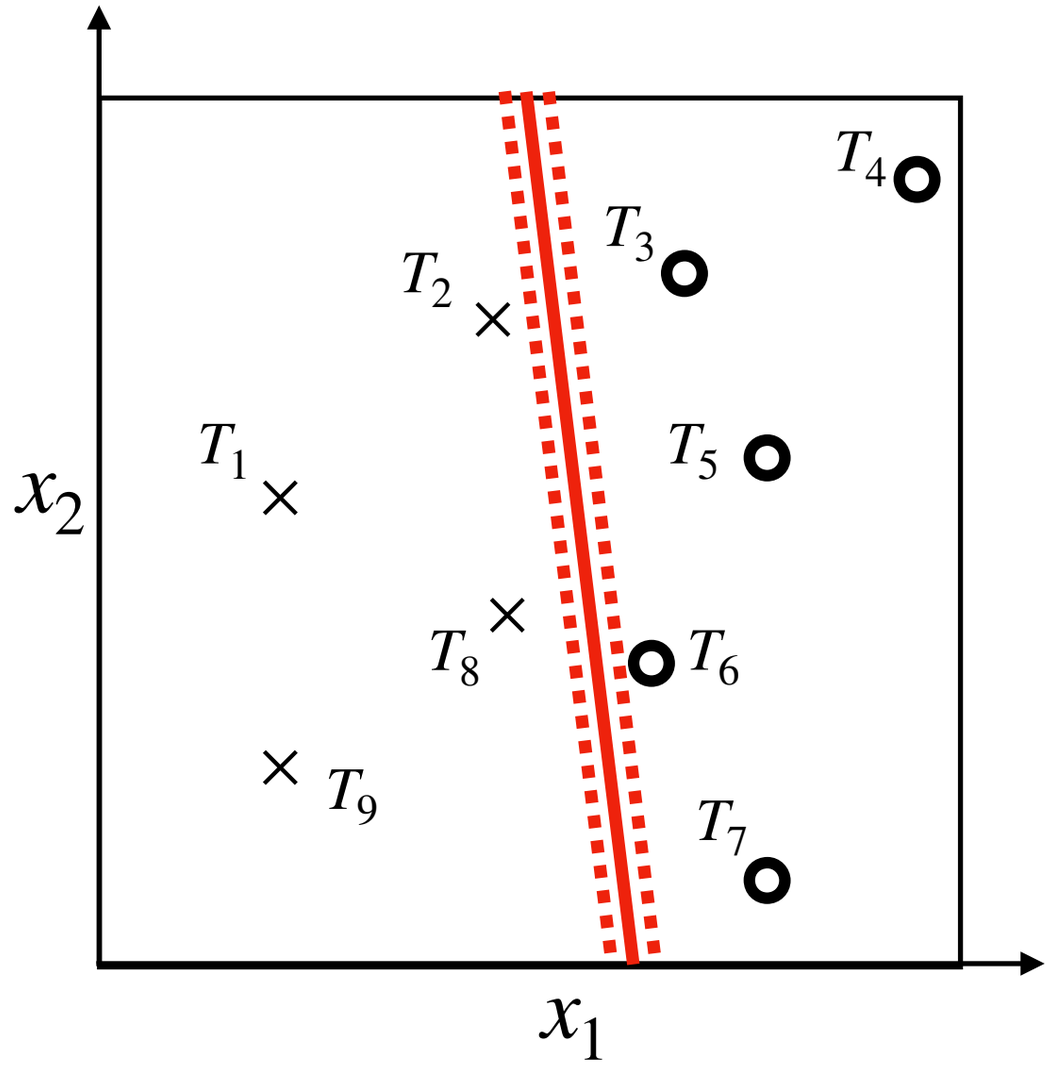
5. 학습 결과를 시험: Test data 로 학습한 결과를 테스트

학습을 너무 tight하게 할 경우, 시험에서 실패할 수도 있음.

= margin (여유) 를 고려하여 경계면을 설정

SVM (Support Vector Machine) : 학습할 때, **여유를 최대화** 할 수 있는 경계면 설정





학습

테스트 (시험)

퍼셉트론 구현

가중치 설정 방법 (**확률적 경사하강법**을 사용하는 알고리즘):

1. 임의로 가중치 $\vec{w}$ 를 설정한다.

2. $\vec{w}$에 의해 잘못 분류된 값인 **임의의** $\vec{x}_k$ 를 선택한다.
   이 경우, 잘못 분류된 값은 $y_k\vec{w} \cdot \vec{x}_k < 0$ 을 만족한다.

3. $\vec{w}' = \vec{w} + y_k\vec{x}_k$ 로 가중치 $\vec{w}$ 를 업데이트 한다.

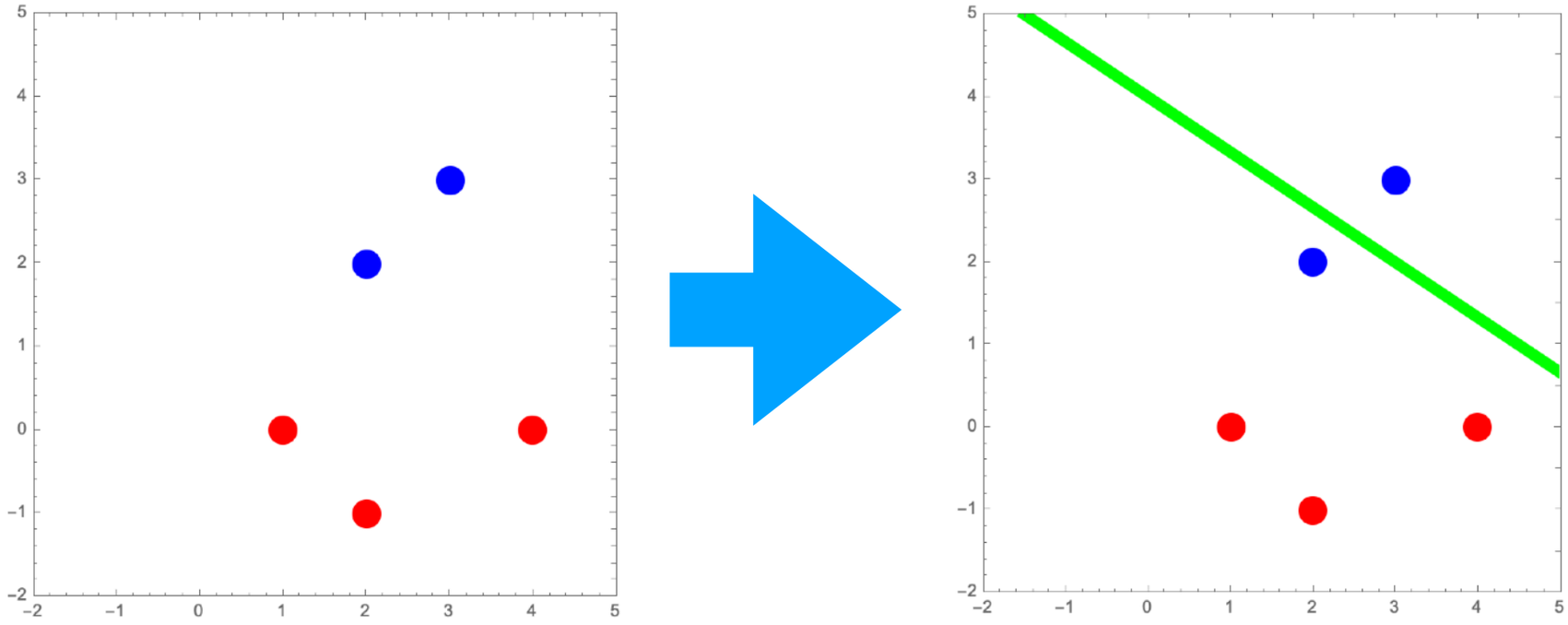
4. 업데이트된 가중치 $\vec{w}'$은 $y_k\vec{w}' \cdot \vec{x}_k > y_k\vec{w} \cdot \vec{x}_k$ 을 만족한다.
   즉, 업데이트된 가중치는 분류의 옳바른 방향으로 경계면을 움직인다.

가중치 구하기: $h(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$ 의 $\vec{w} = (w_0, w_1, w_2, \cdots . w_d)$

를 점진적 (iteration) 방법으로 구하기

- 예) 특성이 (x_1, x_2), 즉 2개인 데이터 5개가 있다.
  $\vec{x} = (x_0, x_1, x_2) = (1, x_1, x_2)$

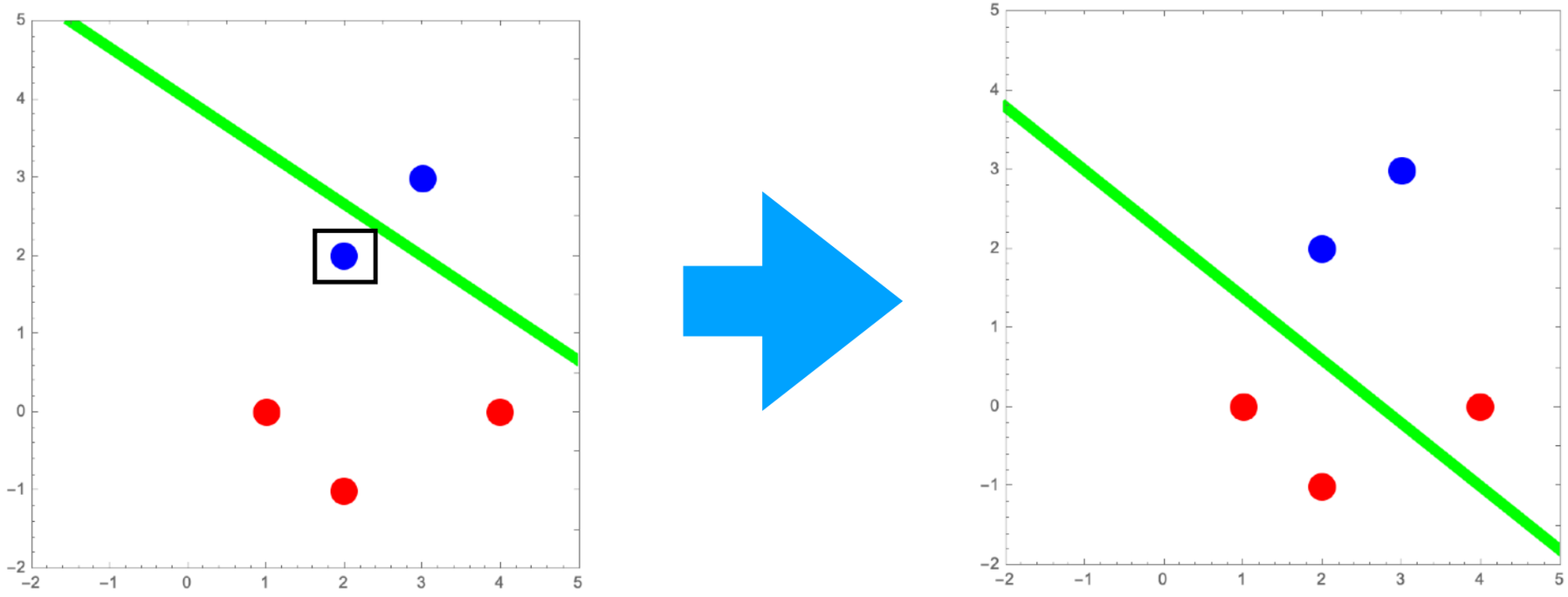
  처음에, 랜덤하게 가중치 $\vec{w} = (w_0, w_1, w_2) = (-12, 2, 3)$ 을 주어, 선형 경계
  $\vec{w} \cdot \vec{x} = w_0 x_1 + w_1 x_1 + w_2 x_2 = -12 + 2x_1 + 3x_2$ 를 만들었다.



가중치 구하기: $h(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$ 의 $\vec{w} = (w_0, w_1, w_2, \cdots . w_d)$

를 점진적 (iteration) 방법으로 구하기 $\vec{w}' = \vec{w} + y_k \vec{x}_k$

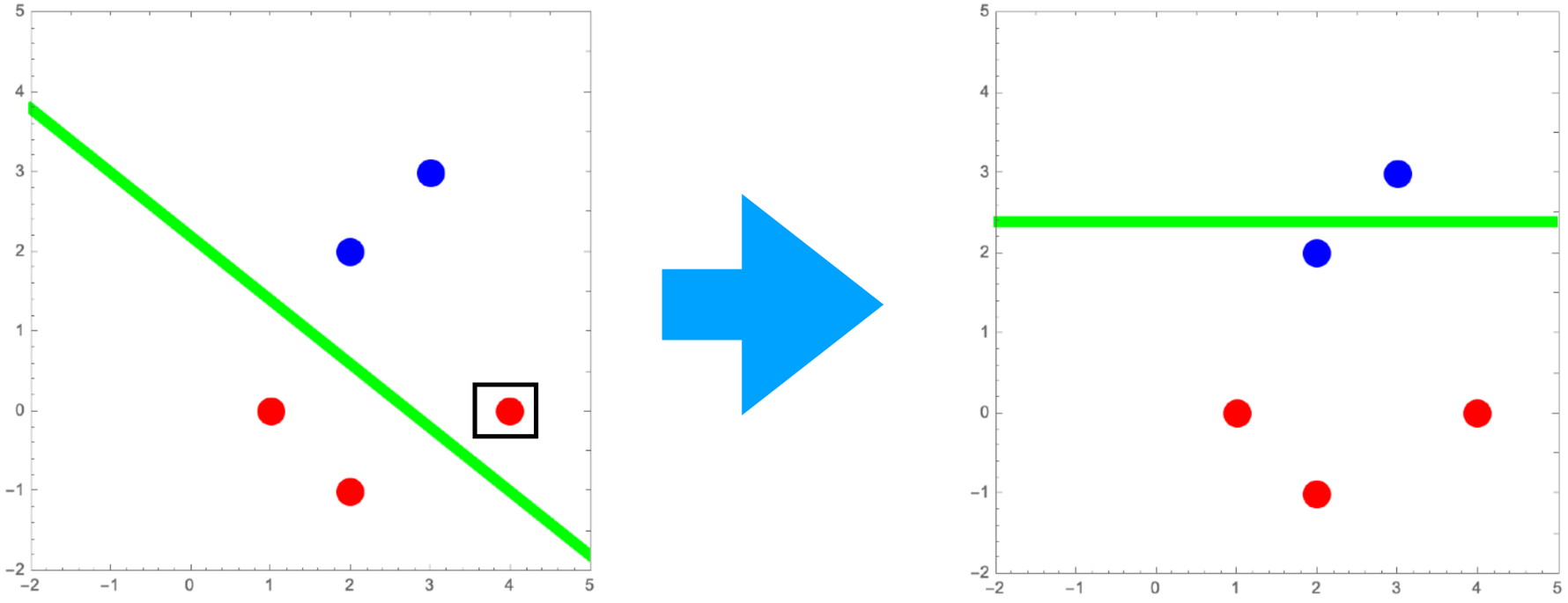
- 분류를 만족하지 않는 한 점 $\vec{x}_k = (1, x_1, x_2)$ 를 선택하여, 가중치를 $\vec{w} \rightarrow \vec{w} + y_k \vec{x}_k$ 로 업데이트 한다.
  즉, $(1, w_1, w_2)' = (-12, 2, 3) + (+1) \times (1, 2, 2) = (-11, 4, 5)$



가중치 구하기: $h(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$ 의 $\vec{w} = (w_0, w_1, w_2, \cdots . w_d)$

를 점진적 (iteration) 방법으로 구하기 $\vec{w}' = \vec{w} + y_k \vec{x}_k$

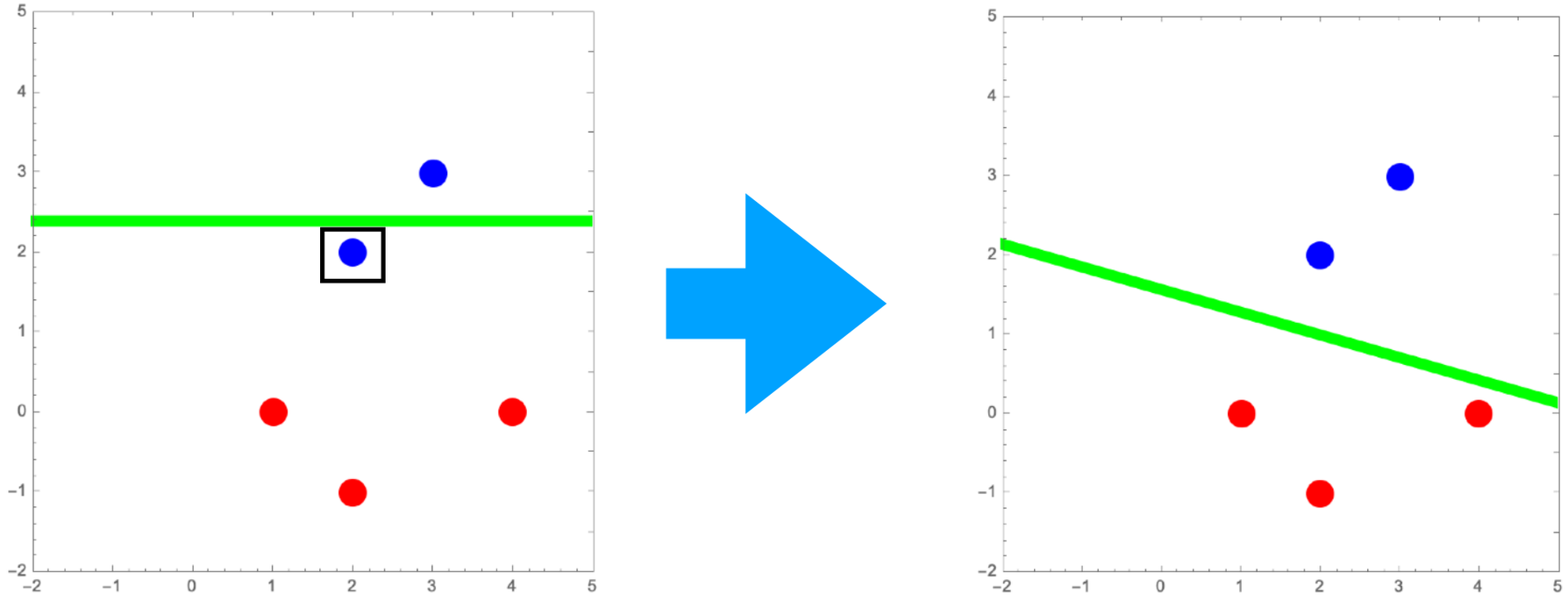
- 분류를 만족하지 않는 한 점 $\vec{x}_k = (1, x_1, x_2)$ 를 선택하여, 가중치를 $\vec{w} \rightarrow \vec{w} + y_k \vec{x}_k$ 로 업데이트 한다.
  즉, $(1, w_1, w_2)' = (-11, 4, 5) + (-1) \times (1, 4, 0) = (-12, 0, 5)$



가중치 구하기: $h(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$ 의 $\vec{w} = (w_0, w_1, w_2, \cdots . w_d)$

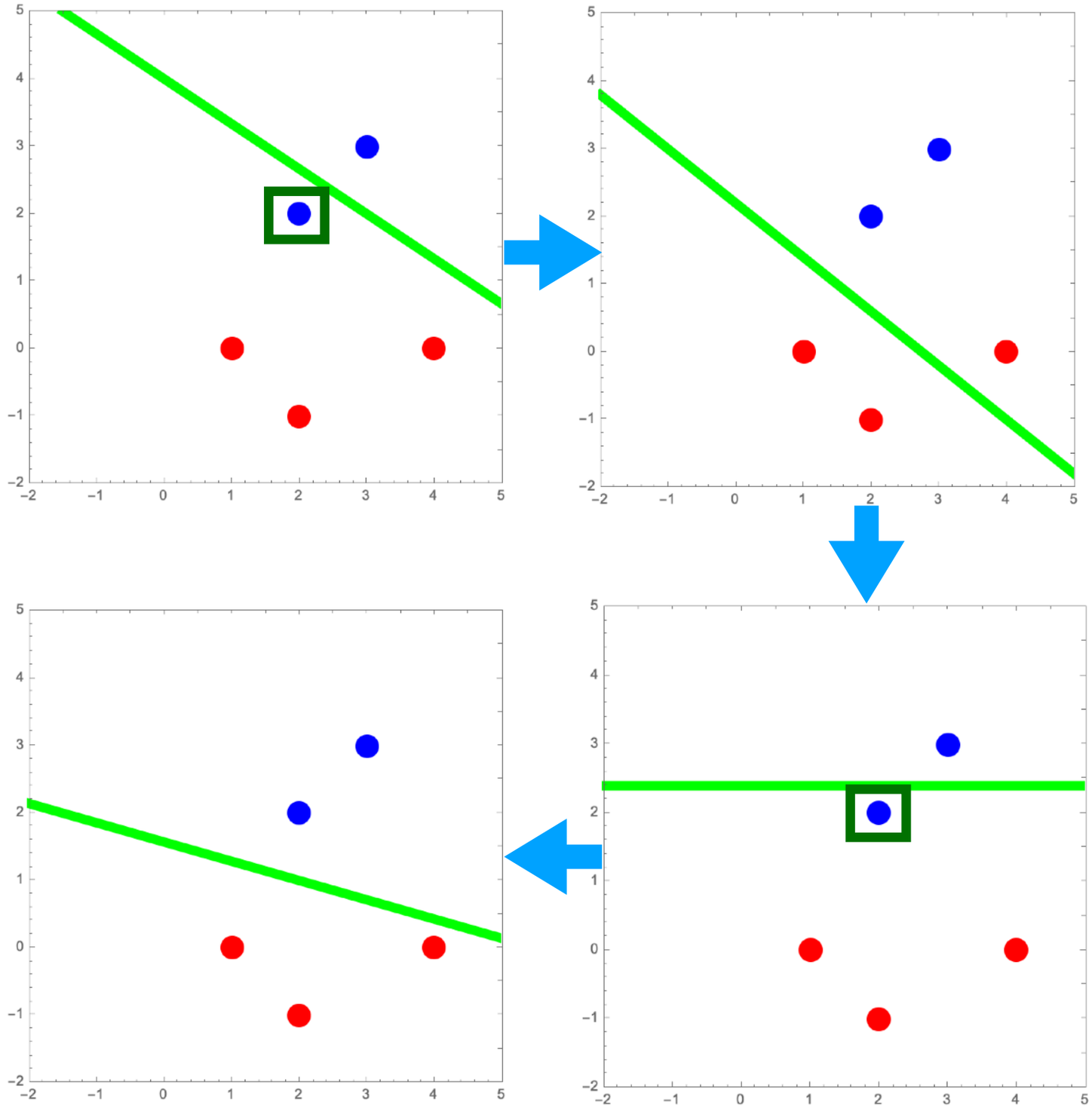
를 점진적 (iteration) 방법으로 구하기 $\vec{w}' = \vec{w} + y_k \vec{x}_k$

- 분류를 만족하지 않는 한 점 $\vec{x}_k = (1, x_1, x_2)$ 를 선택하여, 가중치를 $\vec{w} \rightarrow \vec{w} + y_k \vec{x}_k$ 로 업데이트 한다.
  즉, $(1, w_1, w_2)' = (-12, 0, 5) + (+1) \times (1, 2, 2) = (-11, 2, 7)$



가중치 구하기: $h(\vec{x}) = \text{sign}\left[\left(\sum_{i=0}^{d} w_i x_i\right)\right]$ 의 $\vec{w} = (w_0, w_1, w_2, \cdots . w_d)$

를 점진적 (iteration) 방법으로 구하기 $\vec{w}' = \vec{w} + y_k \vec{x}_k$



1. $y_k \vec{w} \cdot \vec{x}_k < 0$ 인 $\vec{x}_k$ 를 선택한다. ($\vec{w}$에 의해 잘못 분류된 값)

2. $\vec{w}' = \vec{w} + y_k \vec{x}_k$ 로 $\vec{w}$ 를 업데이트 한다.

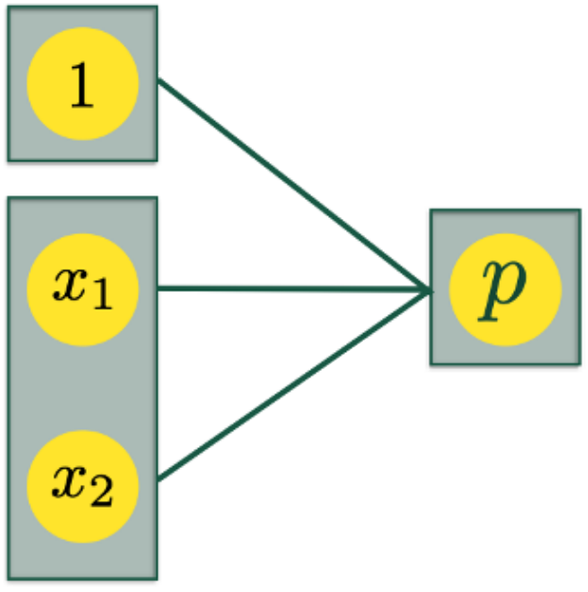
3. $y_k \vec{w}' \cdot \vec{x}_k > y_k \vec{w} \cdot \vec{x}_k$

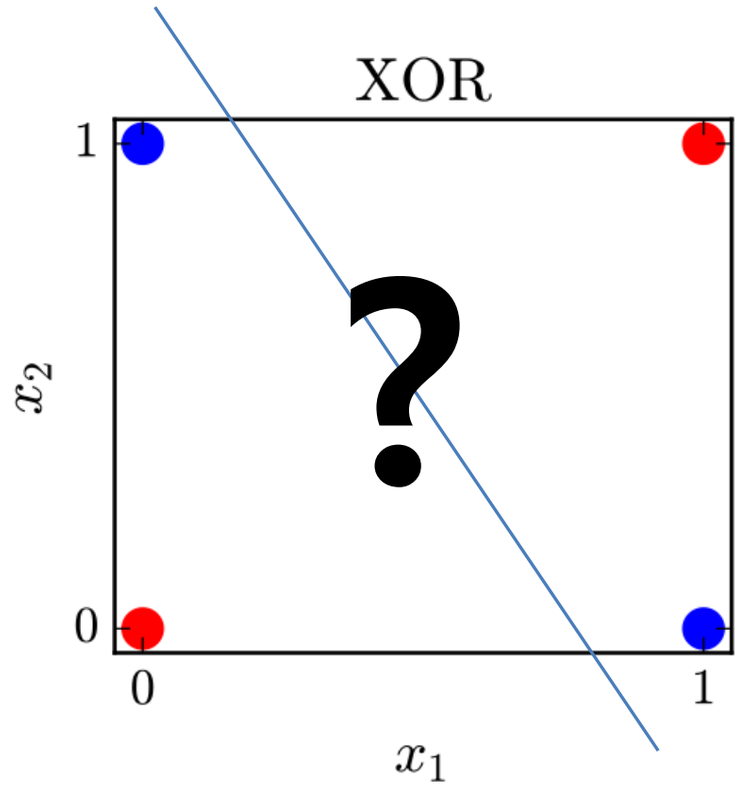
퍼셉트론의 한계

- 퍼셉트론은 선형 분류 방법으로, 그 한계가 있다.
  "Perceptrons: an introduction to computational geometry"
  Marvin Minsky & Seymour Papert (1969)

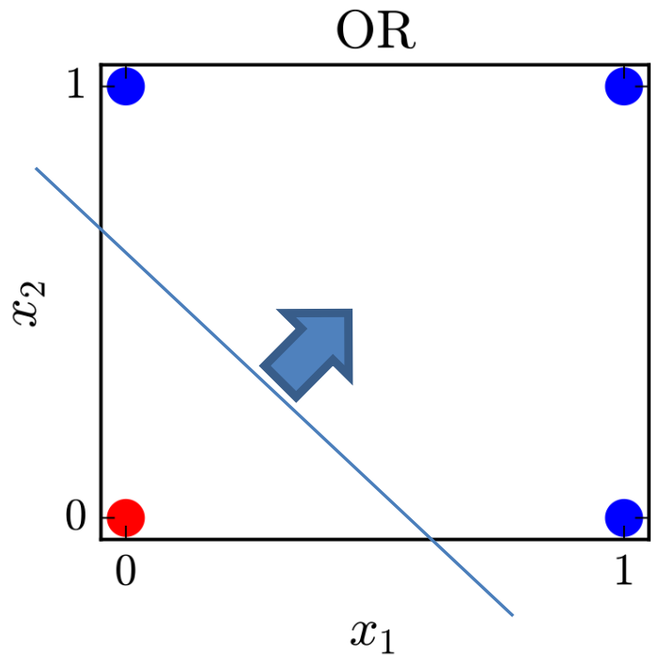
XOR

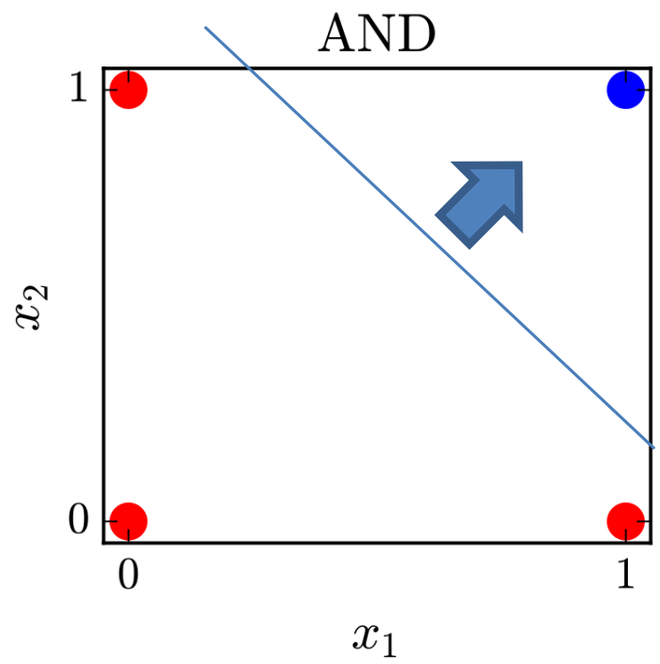
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

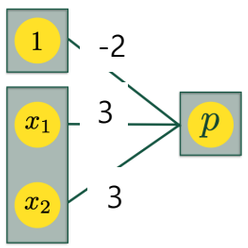


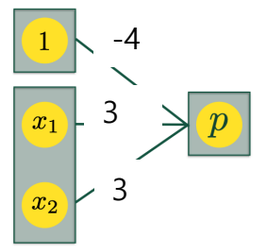


- 퍼셉트론을 확장 시킬 수 있는 방법은 무엇인가?

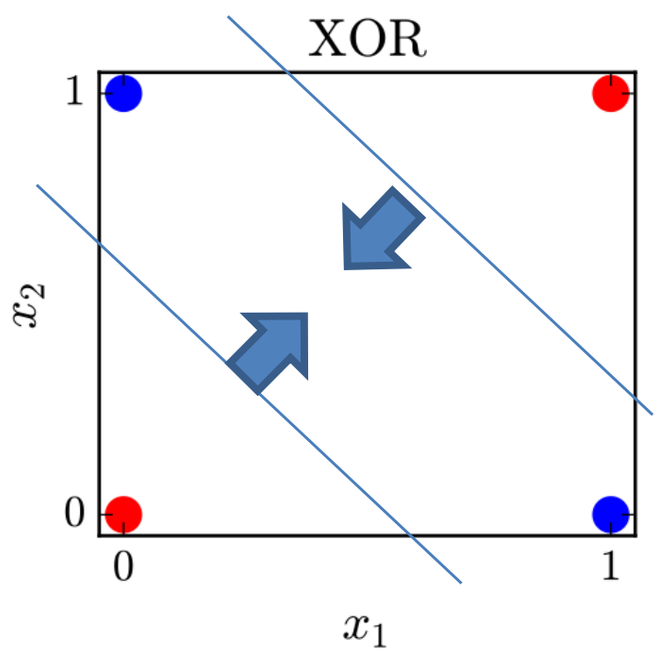


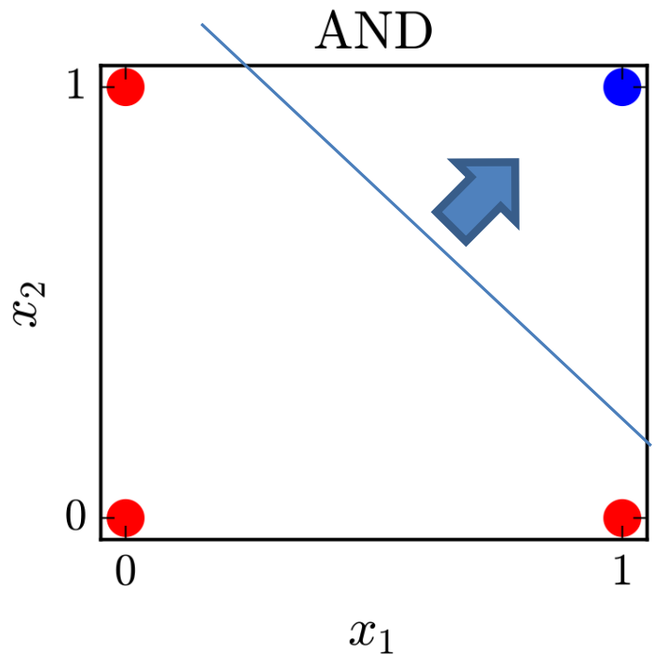


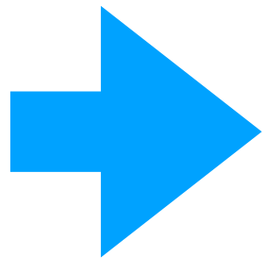


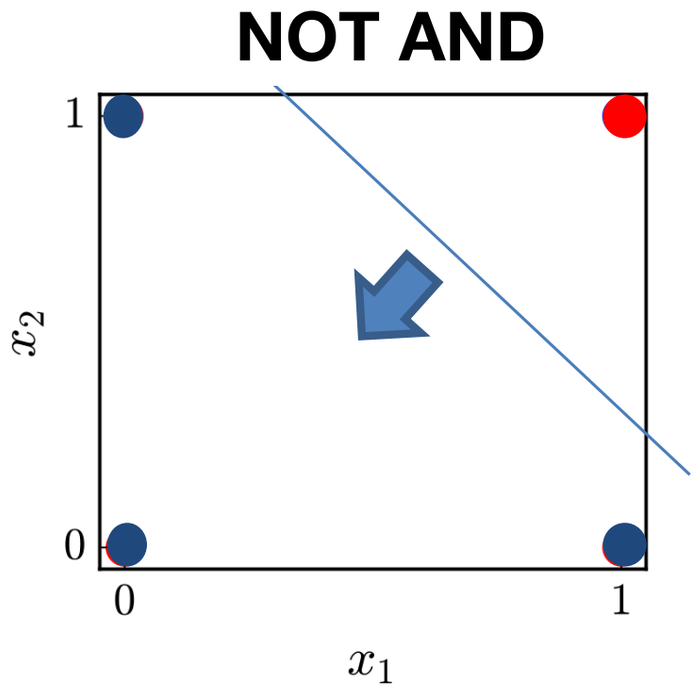


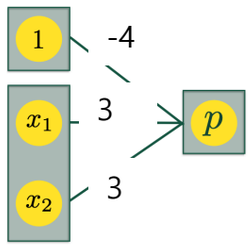
XOR 은 다음의 조건을 만족시키는 **경계들**을 찾으면 된다.

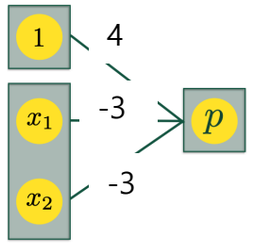




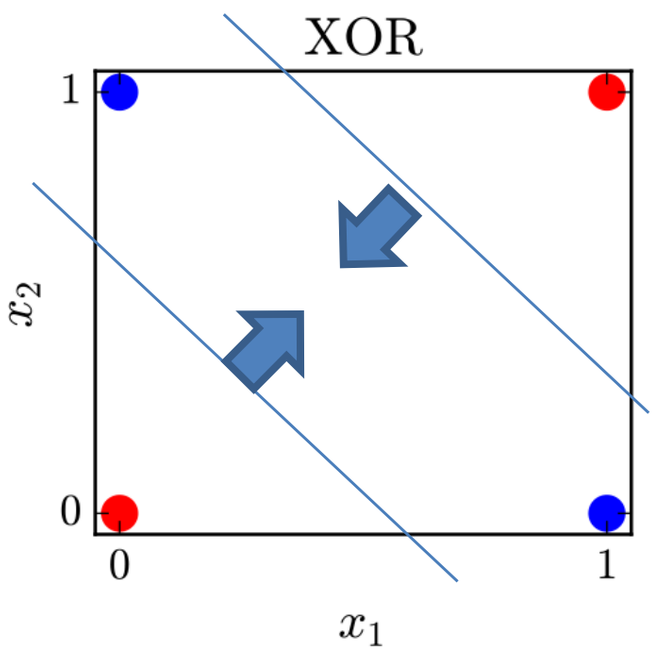




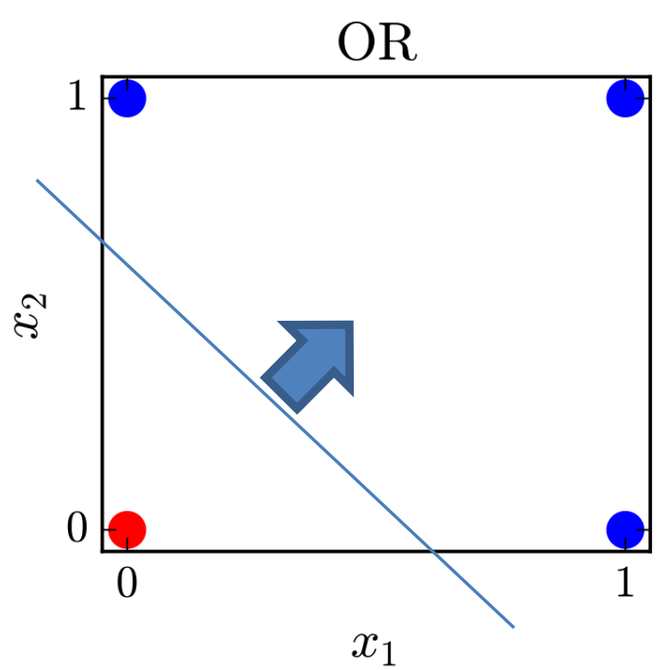




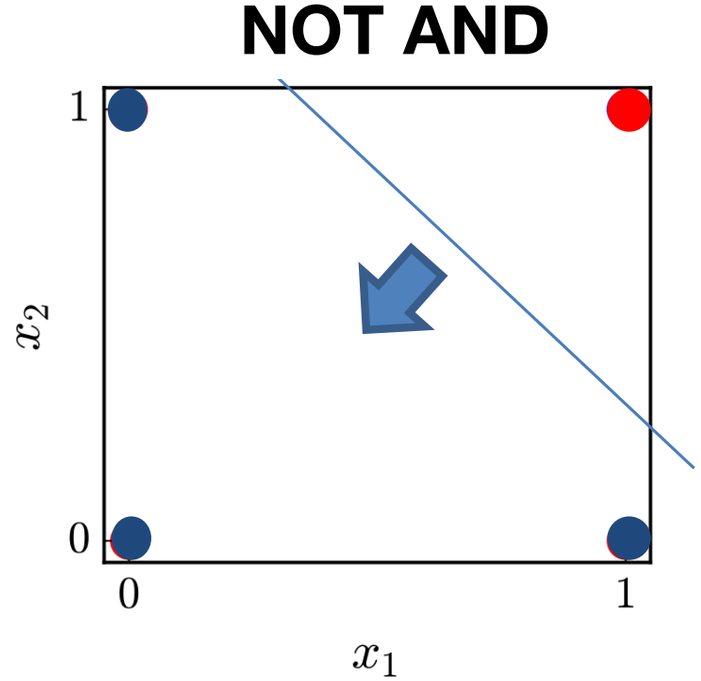
XOR 은 다음의 조건을 만족시키는 **경계들**을 찾으면 된다.



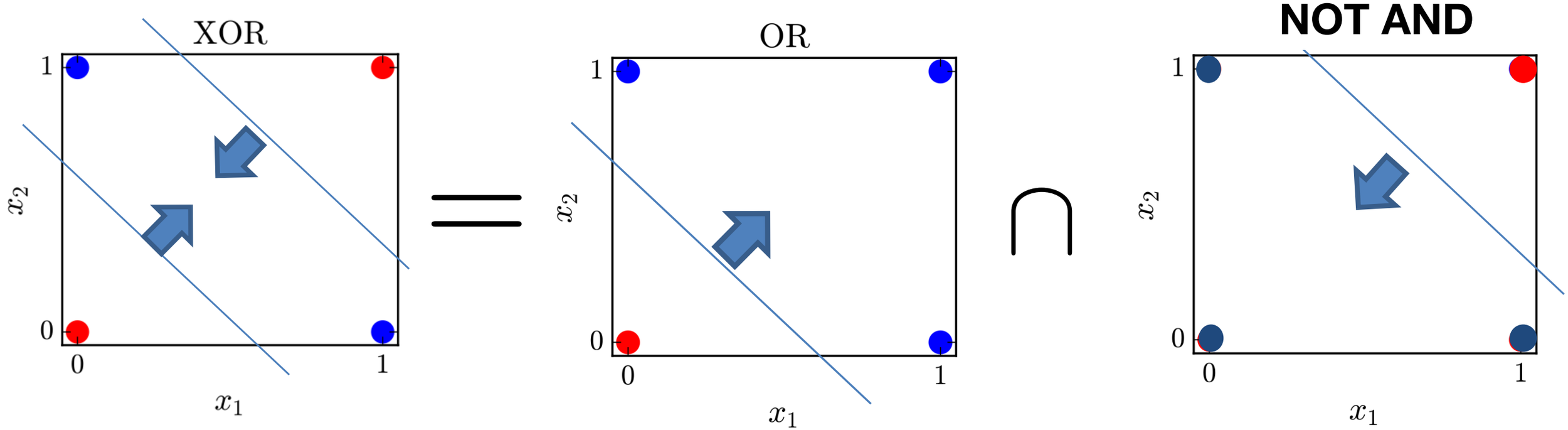
=

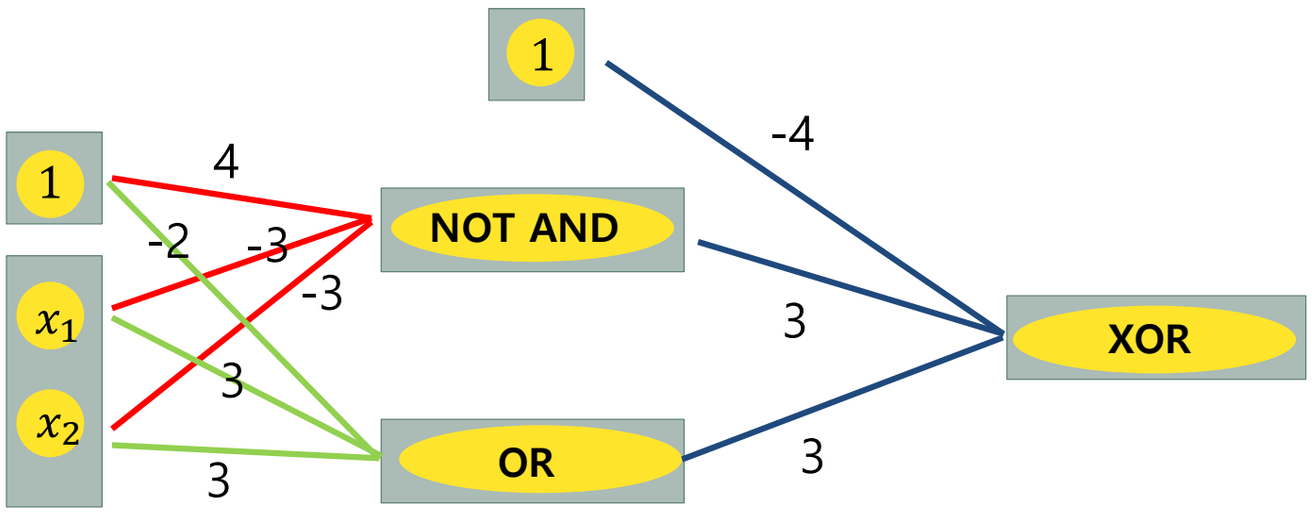


∩



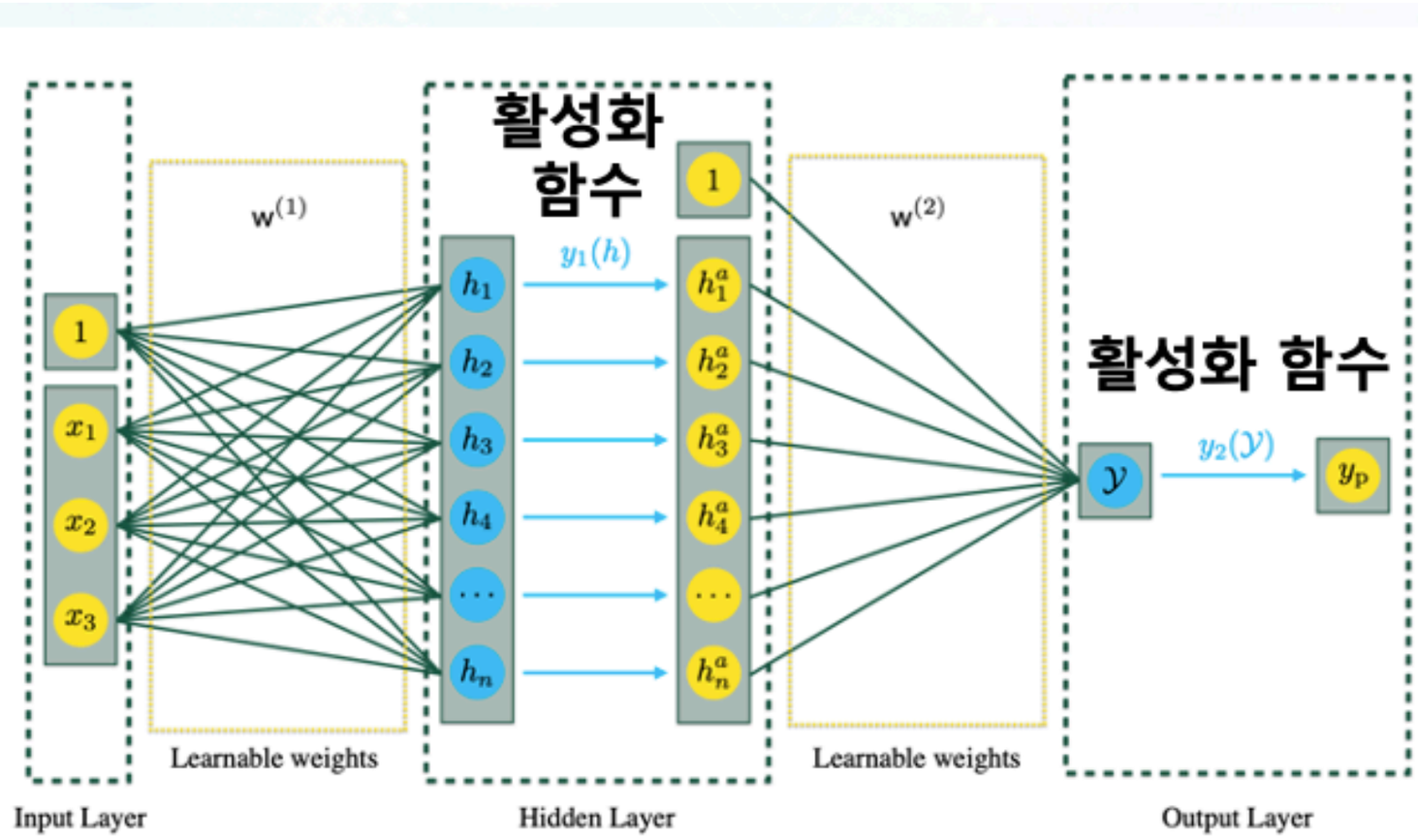
XOR = OR 과 NOT AND 를 AND 로 묶는, **다층신경망**을 사용하면 됨



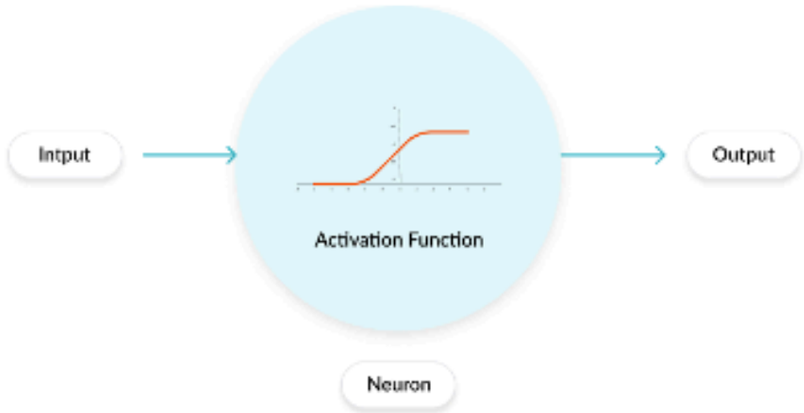


x_1	x_2	OR	NOT AND	XOR
0	0	0 (-2)	1 (+4)	0 (-1)
0	1	1 (+2)	1 (+1)	1 (+2)
1	0	1 (+2)	1 (+1)	1 (+2)
1	1	1 (+4)	0 (-2)	0 (-1)

- **신경망의 구조**

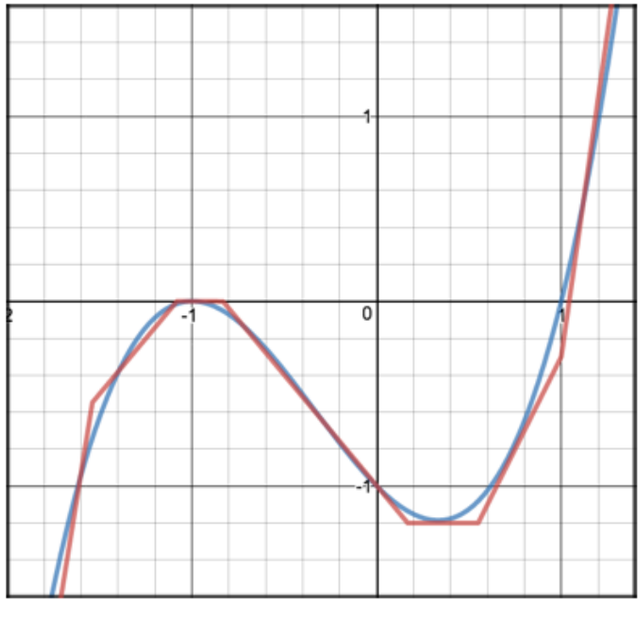


활성화 (activation)함수



- 출력 뉴론의 반응 (activation) 을 표시
- 비선형 관계 표시
  (일차함수의 합성 = 일차함수.)

- 활성화 함수를 통해 **다양한 함수를 근사**시킬 수 있다.

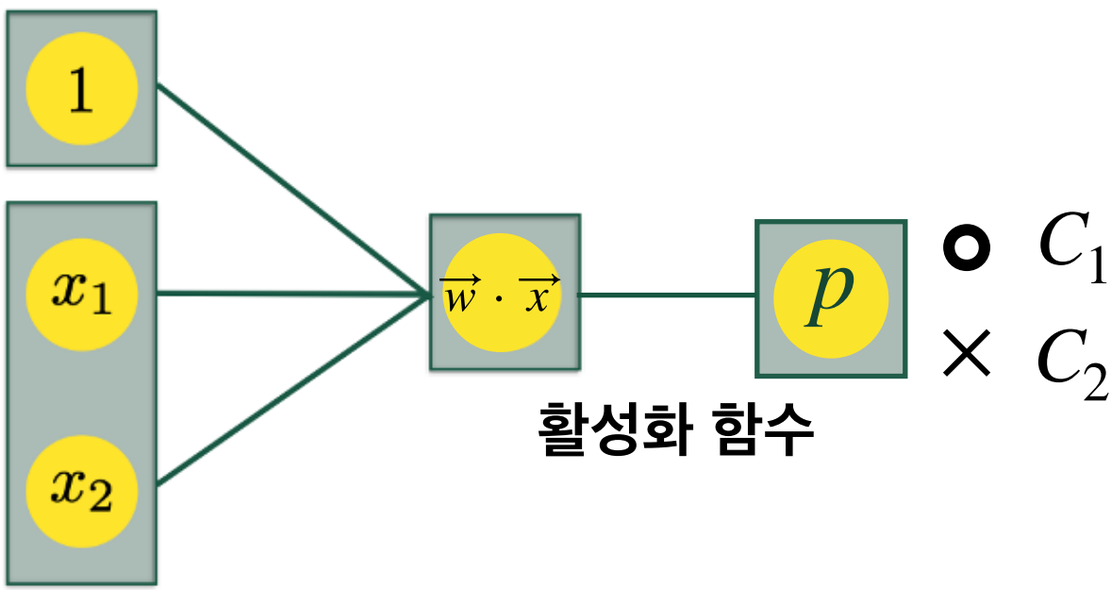


$$n_1(x) = Relu(-5x - 7.7)$$
$$n_2(x) = Relu(-1.2x - 1.3)$$
$$n_3(x) = Relu(1.2x + 1)$$
$$n_4(x) = Relu(1.2x - .2)$$
$$n_5(x) = Relu(2x - 1.1)$$
$$n_6(x) = Relu(5x - 5)$$

$$Z(x) = -n_1(x) - n_2(x) - n_3(x) + n_4(x) + n_5(x) + n_6(x)$$

- 베이즈 정리 (Bayes' theorem)

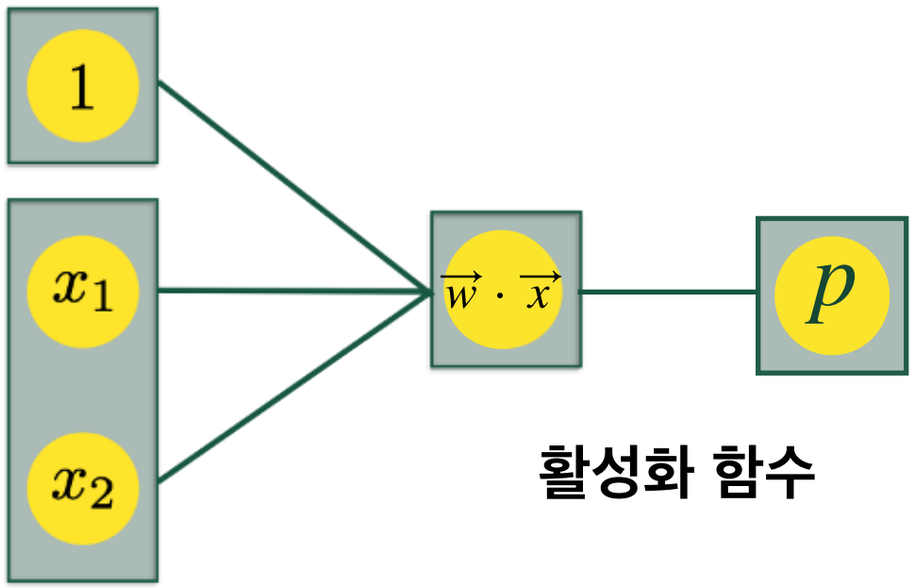
$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)}$$



$$P(C_1|x) = \frac{P(x|C_1)P(C_1)}{P(x)} \quad , P(x) = P(x|C_1)P(C_1) + P(x|C_2)P(C_2)$$

- $a = \log\left(\frac{P(x|C_1)P(C_1)}{P(x|C_2)P(C_2)}\right)$ 로 정의를 하면,

$$P(C_1|x) = \frac{1}{1 + exp(-a)} \equiv \sigma(a)$$ **(로지스틱 시그모이드 함수)**

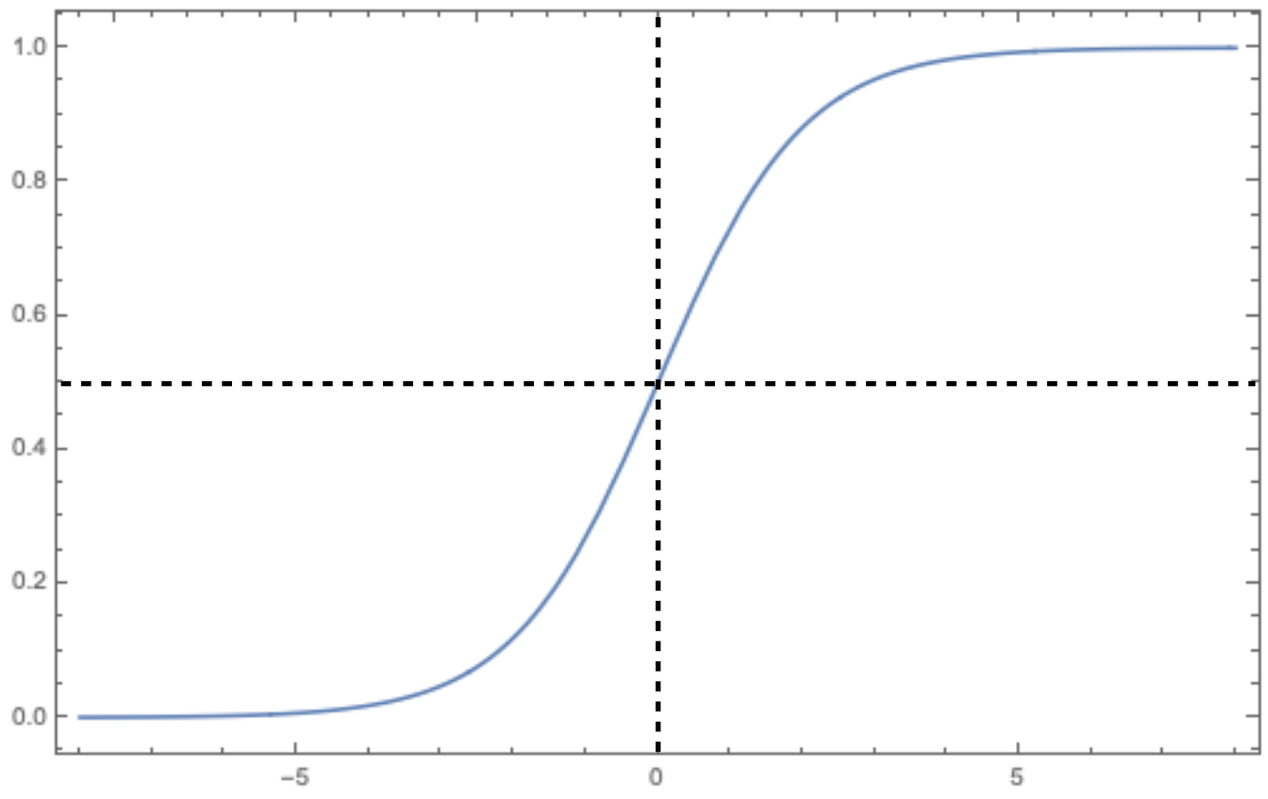


$+1 =$ ○ C_1

$0 =$ × C_2

활성화 함수

(로지스틱 시그모이드 함수)

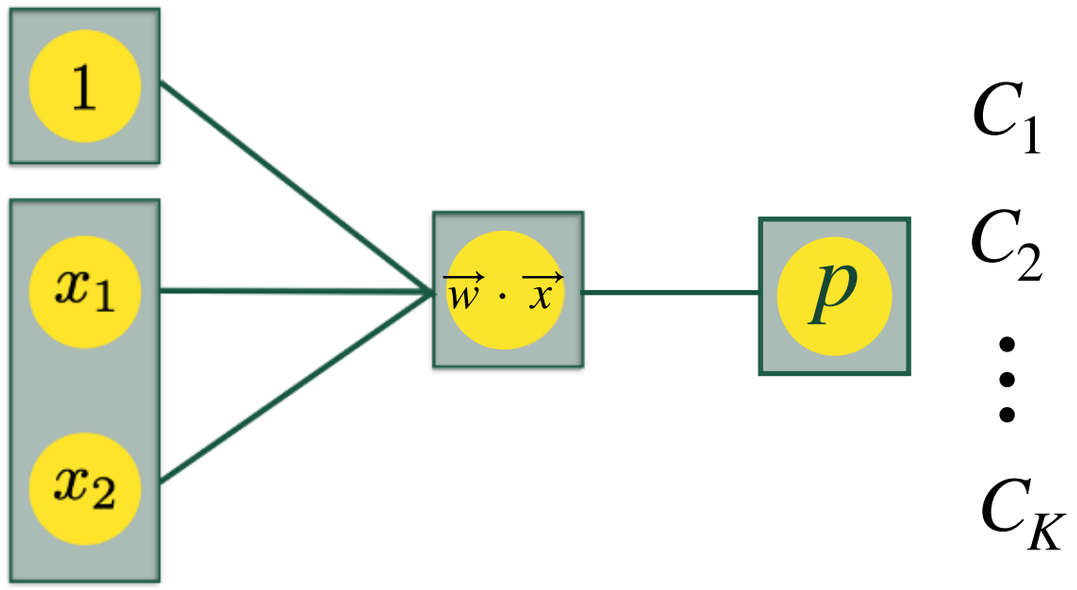


$$\sigma(a) = \frac{1}{1 + e^{-a}}$$

$$\sigma(-a) = 1 - \sigma(a)$$

$$P(C_2 \mid x) = 1 - P(C_1 \mid x)$$

- 클래스 (K) 가 여러개가 있는 경우, 즉 $K > 2$



- 소프트맥스 (softmax) 함수

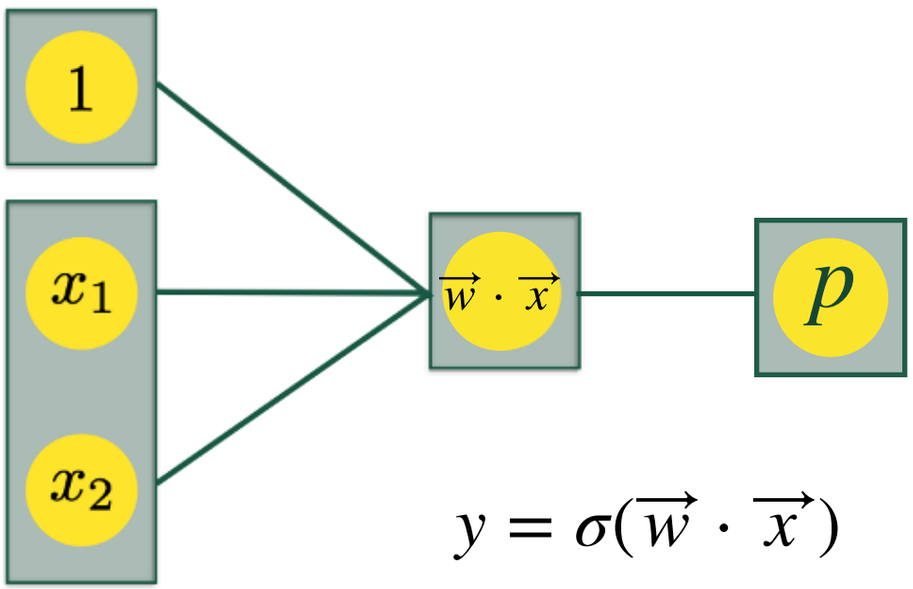
$$P(C_i \mid x) = \frac{P(x \mid C_i)\, P(C_i)}{\sum_j P(x \mid C_j)\, P(C_j)} = \frac{exp(a_i)}{\sum_j exp(a_j)}$$

$$a_i = \log \left(p(x \mid C_i) P(C_i) \right)$$

손실함수의 통계적 해석

- 손실함수 (Loss function) : 지도학습에서 예측값과 관측값의 차이

$$\mathscr{L}_i = (y_i - t_i)^2$$



$$y = \sigma(\vec{w} \cdot \vec{x})$$

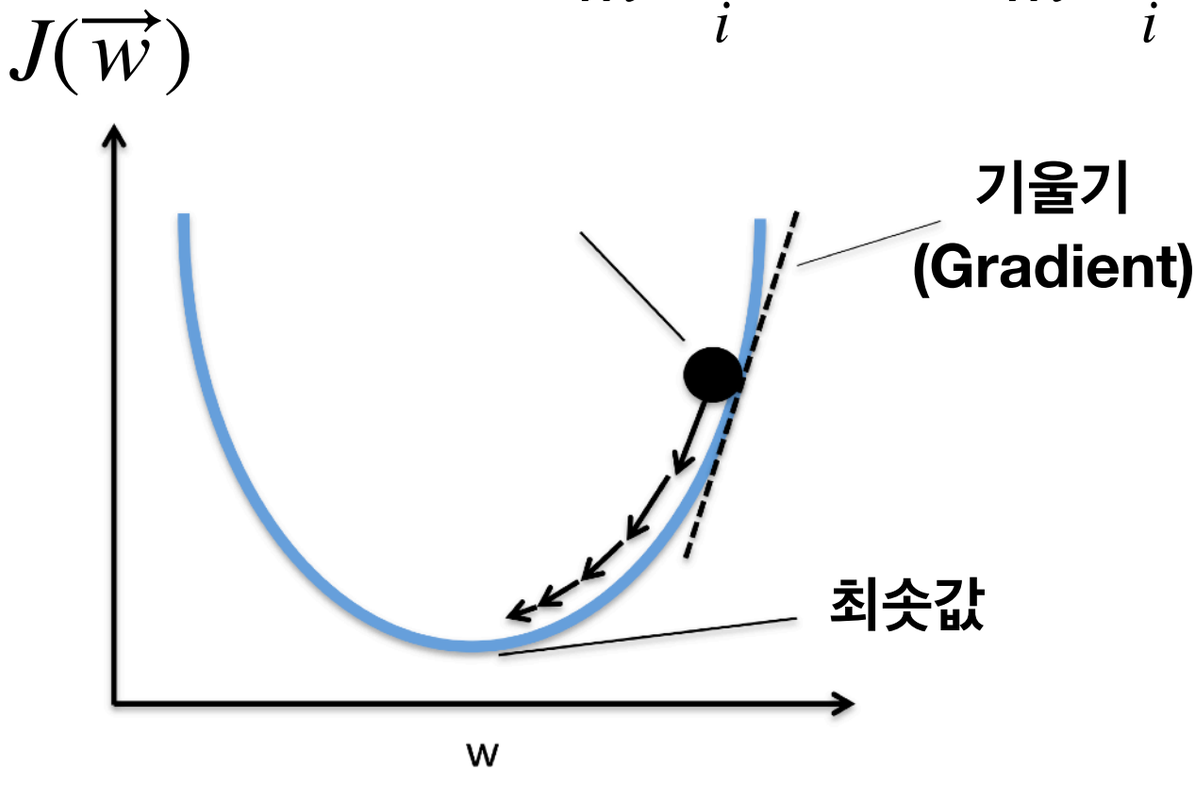
$$\begin{matrix} y_1 = \sigma(\vec{w} \cdot \vec{x}_1) \\ y_2 = \sigma(\vec{w} \cdot \vec{x}_2) \\ \vdots \\ y_m = \sigma(\vec{w} \cdot \vec{x}_m) \end{matrix}$$

예측값

$$\begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix}$$

관측값

비용함수 $J(\vec{w}) = \frac{1}{m}\sum_i \mathscr{L}_i = \frac{1}{m}\sum_i (y_i - t_i)^2$ 를 최소화



경사하강법 (Gradient Descent)

$$\vec{w} \leftarrow \vec{w} - \alpha \nabla_{\vec{w}} J$$

$$w_i \leftarrow w_i - \alpha \frac{\partial J}{\partial w_i}$$

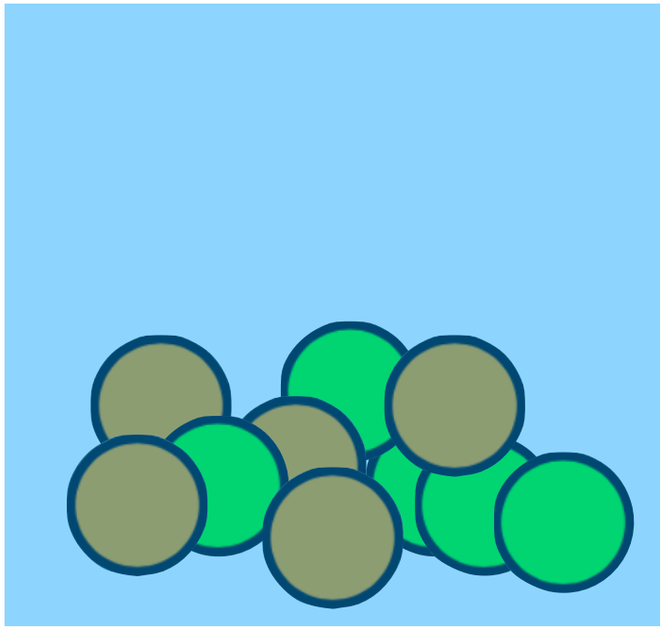
- **최대 가능도법** (Maximum likelihood method)

관측된 데이터를 바탕으로 모델의 **파라메터 (parameter) 를 추측함**.

How: 파라메터에 따른 모델이 예측하는 것과, 관측된 데이터와 **일치할 가능성을 최대**.
즉, **관측된 사건의 발생확률을 최대**로 만드는 모델 파라메터를 찾는 과정

예) 박스에서 무작위로 5개를 꺼냈더니, 그 중 2개가 사과이고 3개가 오렌지였다.
박스에서 과일을 꺼내는 각 과정이 서로 영향을 주지 않을 때, 과일을 꺼낼 때, 사과를 추출한 확률은 ?

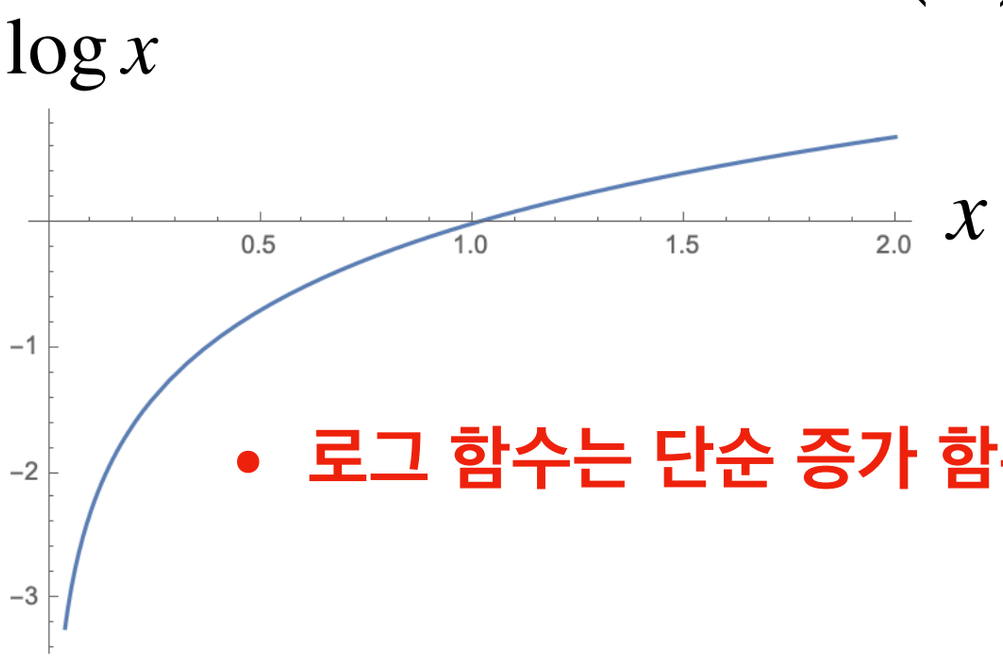
$$P(t=a) = \frac{2}{5} ???$$



사과를 추출할 확률을 $P(t=a) = w$ 라 하면,
5개 추출 시, 2개가 사과이고 3개가 오렌지 경우를 발생시키는
확률 $P(t=\{a,a,o,o,o\}) = \binom{5}{2} w^2 (1-w)^{(5-2)}$

이를 최대화 시키는, 즉 $\frac{dP}{dw} = 0$ 인 w 를 찾으면 됨.

$P(t = \{a, a, o, o, o\}) = \binom{5}{2} w^2 (1-w)^{(5-2)}$ 를 w 로 미분은 복잡.



- **로그 함수는 단순 증가 함수이므로, P 의 최대값 경우 = $\log P$ 의 최대값 경우**

대신, 로그값 $\log P = \log \binom{5}{2} + 2 \log w + 3 \log(1-w)$ 를 미분하는 것은 쉬움.

$$\frac{d \log P}{dw} = \frac{2}{w} - \frac{3}{1-w} = 0 \rightarrow w = \frac{2}{5}$$

베르누이 분포

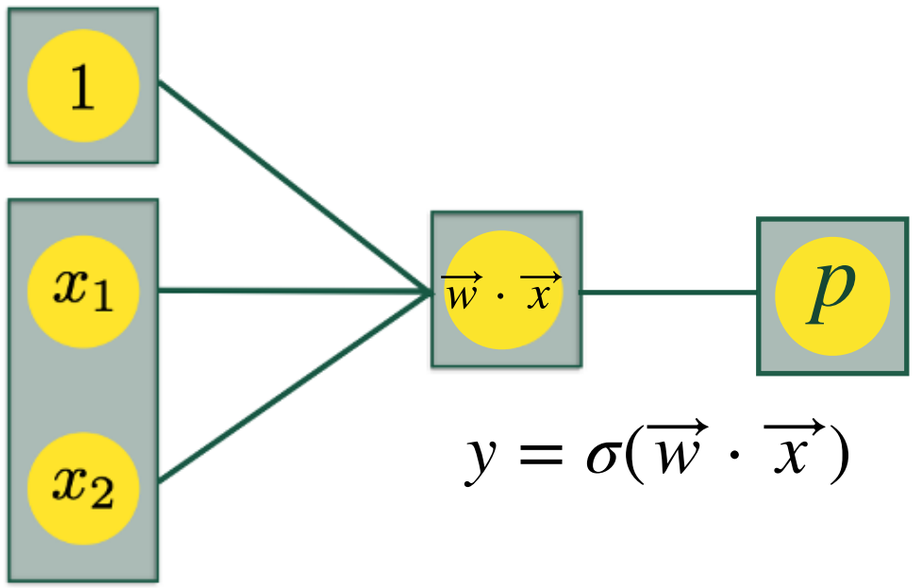
- 클래스 $t \in \{0,1\}$ 일 때, $P(t=1 \mid x) = y$ 라 하면,
  이 클래스의 생성확률 $P(t \mid x) = y^t (1-y)^{(1-t)}$ 이다.

  $P(1 \mid x) = y^1 (1-y)^{(1-1)} = y$

  $P(0 \mid x) = y^0 (1-y)^{(1-0)} = 1-y$

- 일반적으로 클래스 $t \in \{t_1, t_2, \cdots, t_m\}$ 이면, 생성확률
  $P(t \mid \overrightarrow{x}) = \Pi_i \, y_i^{t_i} (1-y_i)^{(1-t_i)}$

교차 엔트로피



$$y = \sigma(\vec{w} \cdot \vec{x})$$

$$\begin{aligned} y_1 &= \sigma(\vec{w} \cdot \vec{x}_1) \\ y_2 &= \sigma(\vec{w} \cdot \vec{x}_2) \\ &\vdots \\ y_m &= \sigma(\vec{w} \cdot \vec{x}_m) \end{aligned}$$

예측값

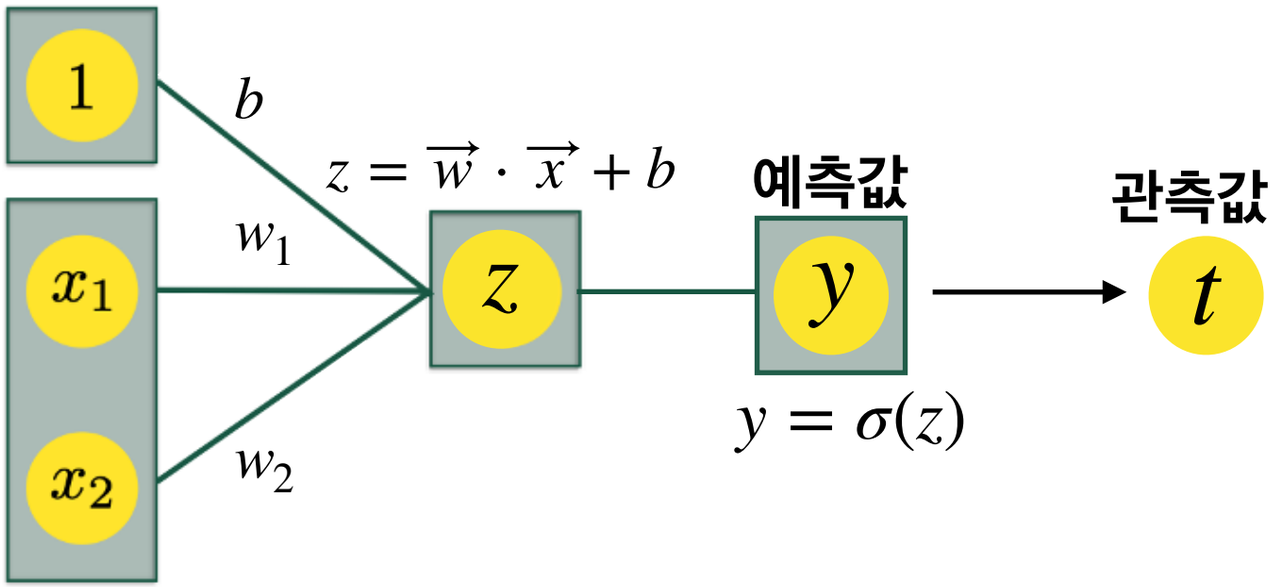
$$\begin{matrix} t_1 \\ t_2 \\ \vdots \\ t_m \end{matrix}$$

관측값

- 최대 가능도법의 생성확률 $P(t \mid \vec{x}) = y^t(1-y)^{(1-t)}$ 을 최대화
- **교차엔트로피** $\mathscr{L} = -\log P(t \mid \vec{x}) = -\left\{ t \log y + (1-t)\log(1-y) \right\}$ **를 최소화**

비용함수 $J = \dfrac{1}{m}\sum_{i=1}^{m} \mathscr{L}_i = -\dfrac{1}{m}\sum_{u=1}^{m} \left[t_i \log y_i + (1-t_i)\log(1-y_i) \right]$ **을 최소화**

단층신경망 구현



손실함수

$$\mathscr{L} = -\left\{ t \log y + (1-t)\log(1-y) \right\}$$

- 손실함수 최소화

$$dw_1 \equiv \frac{d\mathscr{L}}{dw_1}$$

$$dw_2 \equiv \frac{d\mathscr{L}}{dw_2}$$

$$db \equiv \frac{d\mathscr{L}}{db}$$

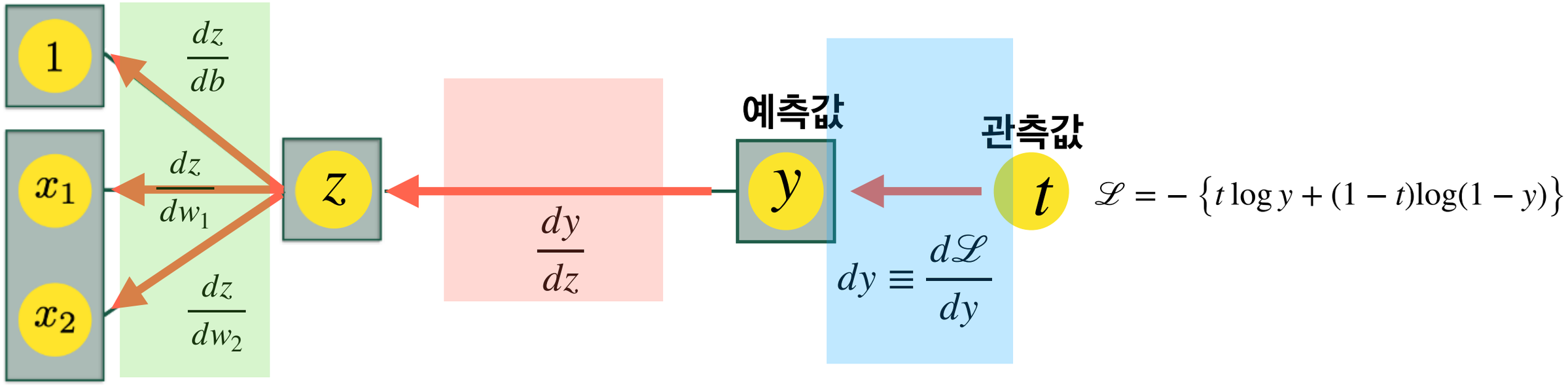
경사하강법

$$w_1 \leftarrow w_1 - \alpha(dw_1)$$

$$w_2 \leftarrow w_2 - \alpha(dw_2)$$

$$b \leftarrow b - \alpha(db)$$

- **합성 함수의 미분을 활용 (연쇄법칙 Chain Rule)**

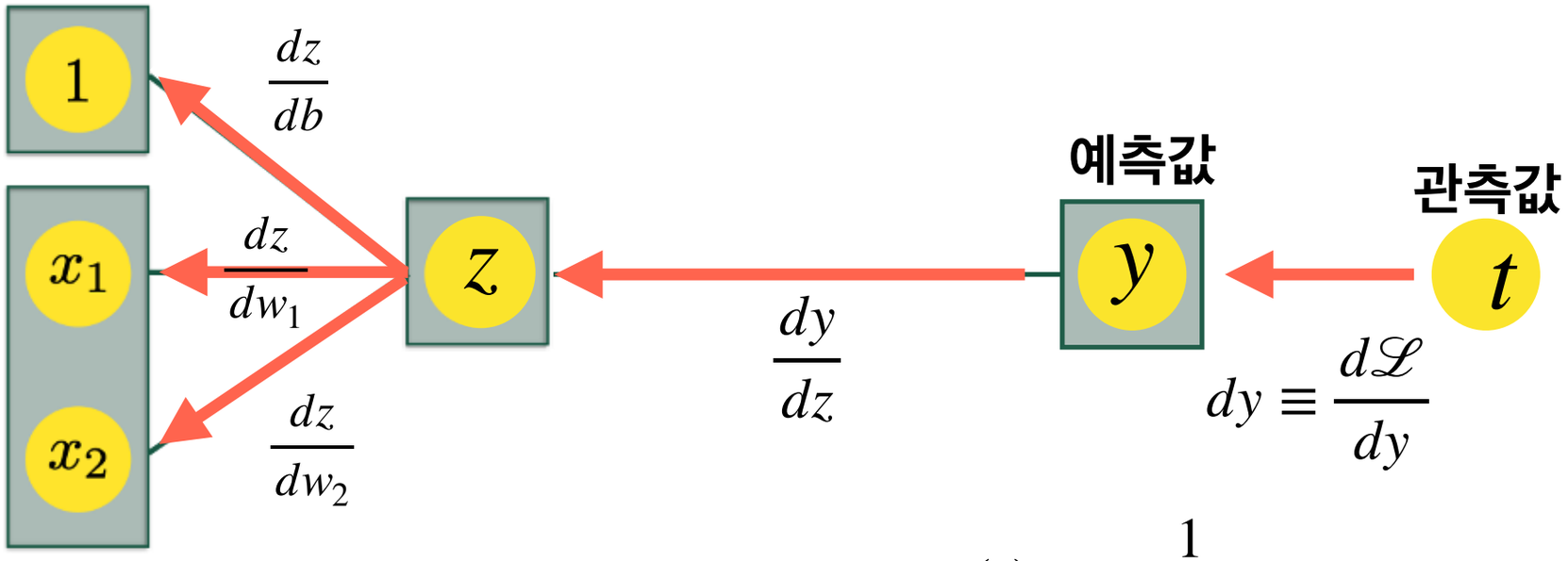


$$dw_1 = \frac{d\mathscr{L}}{dw_1} = \frac{d\mathscr{L}}{dy} \cdot \frac{dy}{dz} \cdot \frac{dz}{dw_1}$$

$$dw_2 = \frac{d\mathscr{L}}{dw_2} = \frac{d\mathscr{L}}{dy} \cdot \frac{dy}{dz} \cdot \frac{dz}{dw_2}$$

$$db = \frac{d\mathscr{L}}{db} = \frac{d\mathscr{L}}{dy} \cdot \frac{dy}{dz} \cdot \frac{dz}{db}$$

- **합성 함수의 미분을 활용 (연쇄법칙 Chain Rule)**



$$z = \vec{w} \cdot \vec{x} + b = w_1 x_1 + w_2 x_2 + b$$

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

$$\mathscr{L} = -\left\{ t \log y + (1 - t) \log(1 - y) \right\}$$

$$\frac{dz}{dw_1} = x_1 (dz)$$

$$\frac{dz}{dw_2} = x_2 (dz)$$

$$\frac{dz}{db} = dz$$

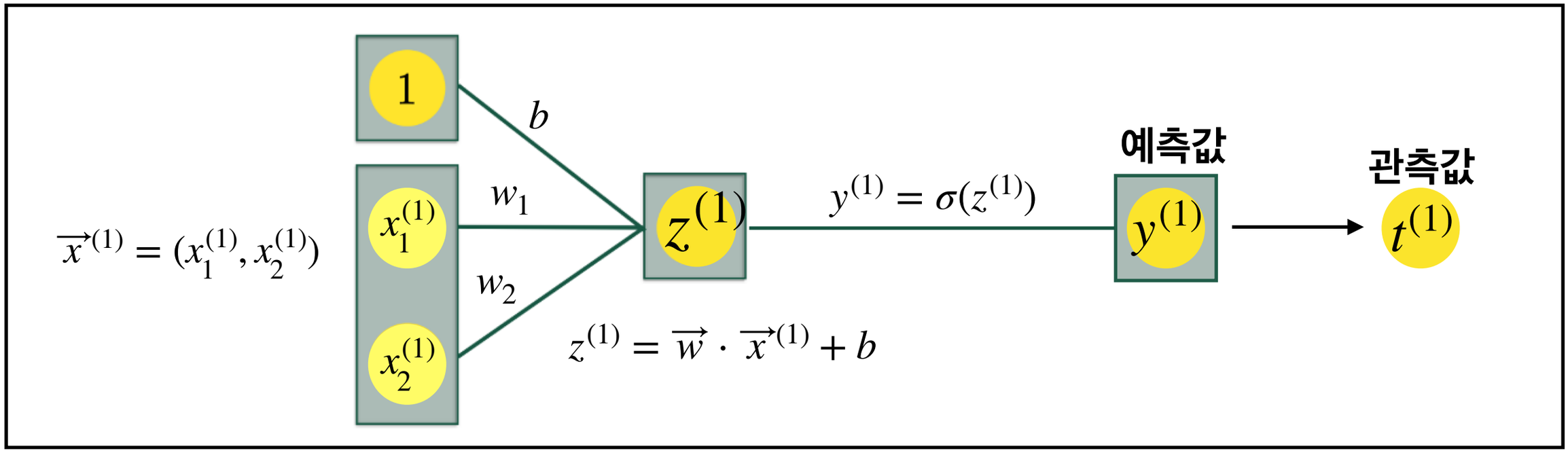
$$\frac{dy}{dz} = \frac{e^{-z}}{(1 + e^{-z})^2} = (1 - y) y$$

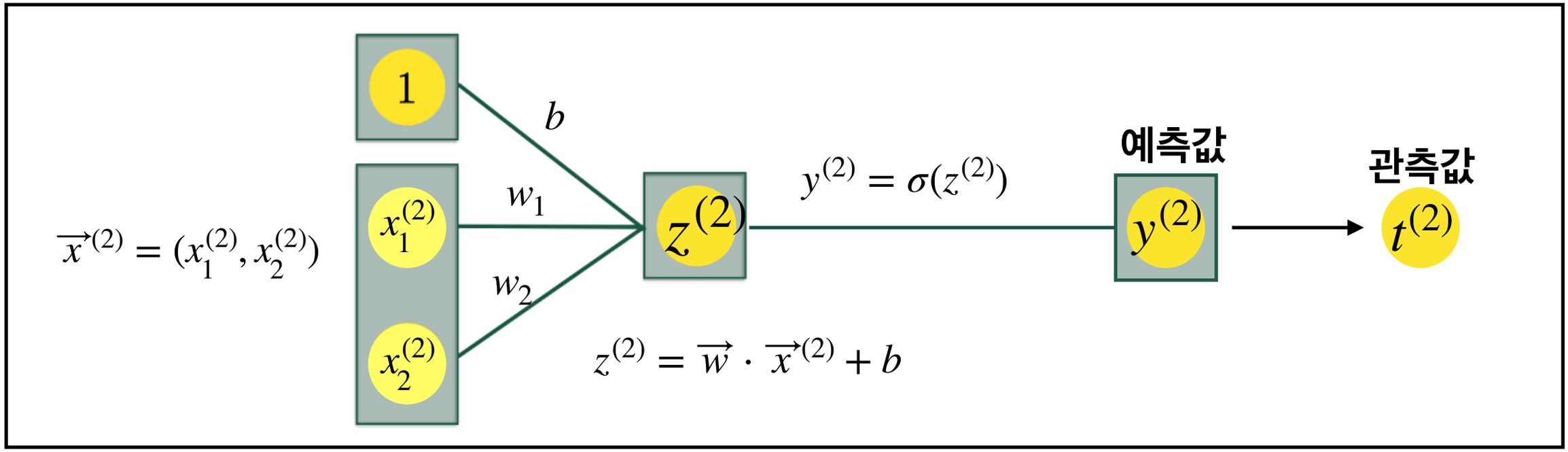
$$\frac{d\mathscr{L}}{dy} = \frac{1 - t}{1 - y} - \frac{t}{y}$$

$$dz = \frac{d\mathscr{L}}{dy} \frac{dy}{dz} = \left(\frac{1 - t}{1 - y} - \frac{t}{y} \right) \times (1 - y) y = y - t$$

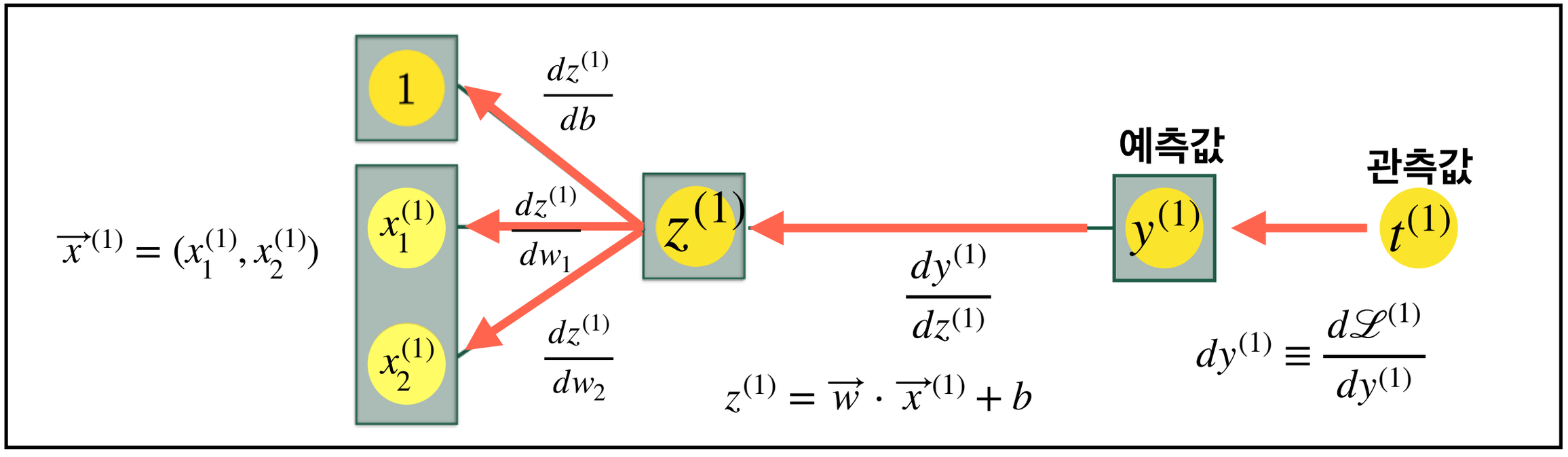
$$dw_1 = \frac{d\mathscr{L}}{dz} \frac{dz}{dw_1} = x_1 (y - t) \quad , \quad dw_2 = \frac{d\mathscr{L}}{dz} \frac{dz}{dw_2} = x_2 (y - t) \quad , \quad db = \frac{d\mathscr{L}}{dz} \frac{dz}{db} = (y - t)$$

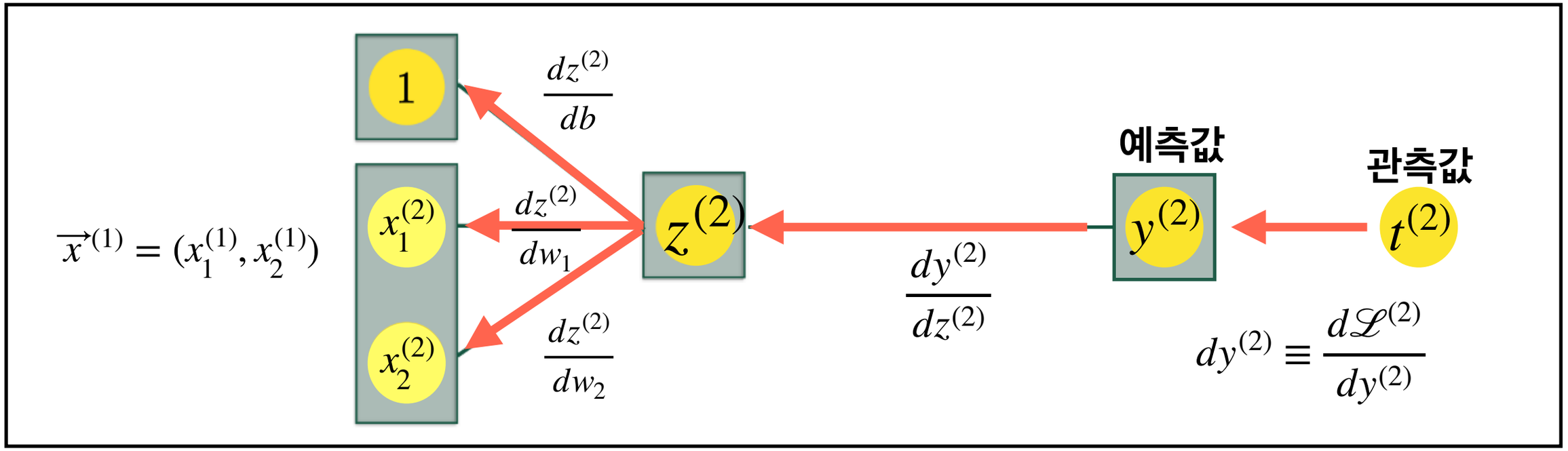
- 데이터가 2개가 있을 때: 각 데이터 당 손실함수의 총 합인 **비용함수**





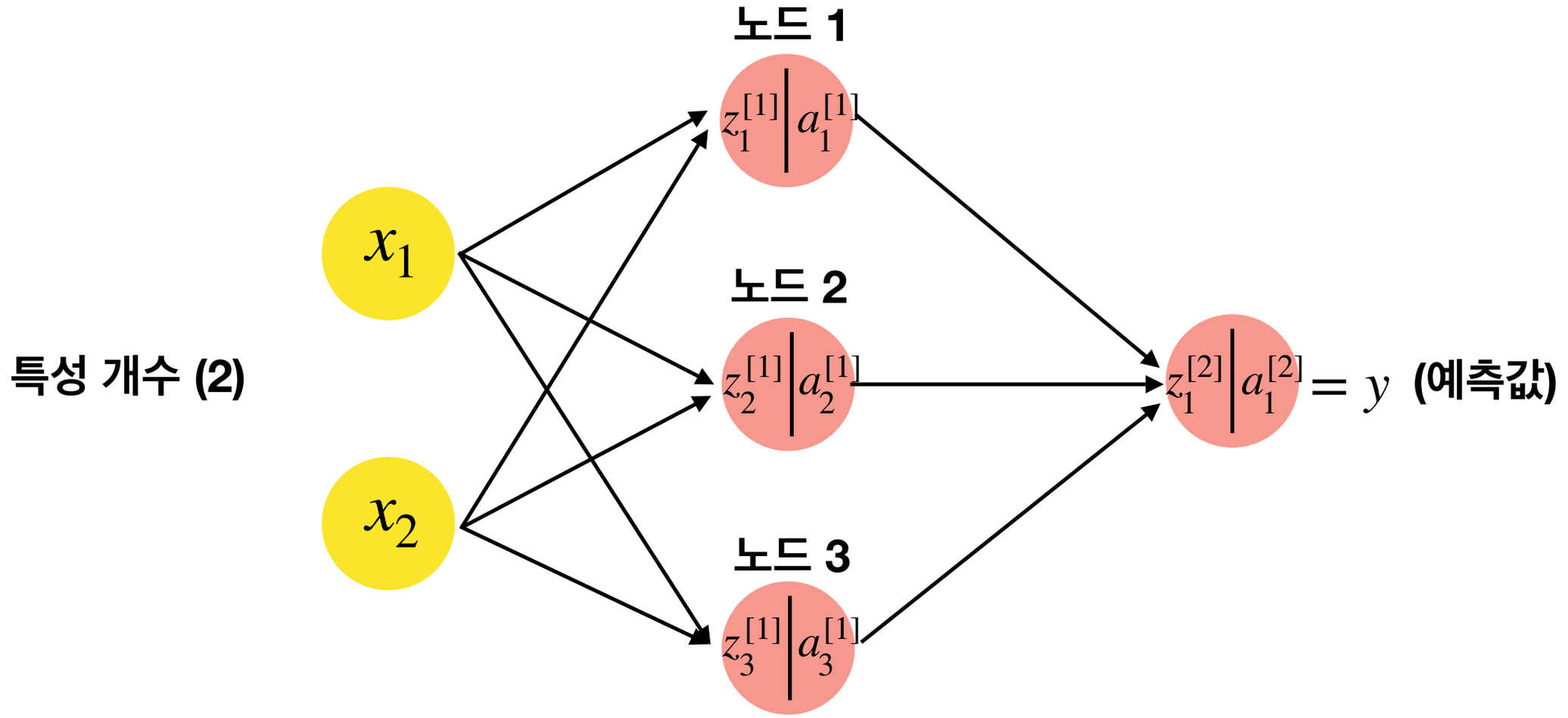
- 데이터가 2개가 있을 때: 각 데이터 당 손실함수의 총 합인 **비용함수**



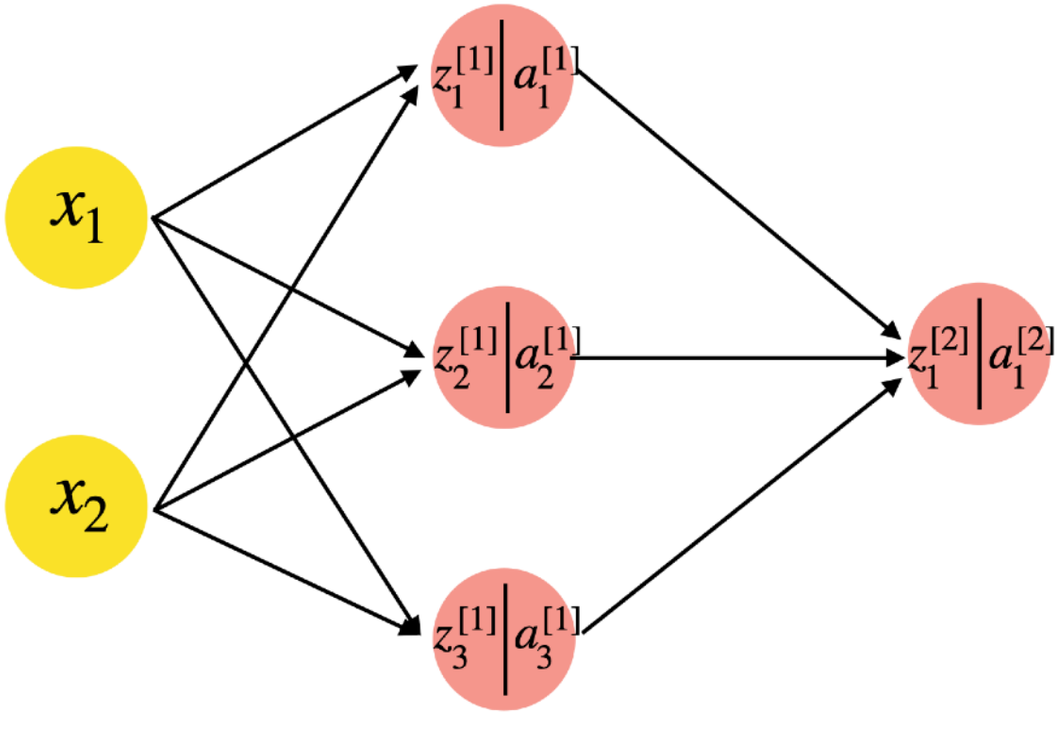


$$J = \frac{1}{2}\sum_{i=1}^{2} \mathscr{L}^{(i)} \quad \Rightarrow \quad dw_1 = \frac{dJ}{dw_1} = \frac{1}{2}\left(\frac{d\mathscr{L}^{(1)}}{dw_1} + \frac{d\mathscr{L}^{(2)}}{dw_1}\right) = \frac{1}{2}\left(dw_1^{(1)} + dw_1^{(2)}\right)$$

다층 신경망



- **표준 표기법**: $z_j^{[i]}$: i 번째 층. j 번째 노드
- 활성화 함수 $a = g(z)$
- 예측값 y , 측정값 t



$$\overrightarrow{x} = (x_1, x_2, \cdots, x_{n_x}) \qquad n_x : \textbf{특성 개수 (2)}$$

$$\overrightarrow{w}_i^{[j]} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots, w_{(i,n_x)}^{[j]} \right)$$

$$z_1^{[1]} = w_1^{[1]\,t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right)$$

$$z_2^{[1]} = w_2^{[1]\,t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right)$$

$$z_3^{[1]} = w_3^{[1]\,t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)$$

$$z_1^{[2]} = \overrightarrow{w}_1^{[2]} \cdot \overrightarrow{a} + b^{[2]}$$

$$\overrightarrow{a} = (a_1, a_2, a_3)$$

- **벡터**의 내적

$$\overrightarrow{w}_1^{[1]} \cdot \overrightarrow{x} = (w_{(1,1)}^{[1]}, w_{(1,2)}^{[1]}, \cdots, w_{(1,n_x)}^{[1]}) \cdot (x_1, x_2, \cdots, x_{n_x}) = w_{(1,1)}^{[1]} x_1 + w_{(1,2)}^{[1]} x_2 \quad (n_x = 2 \text{ 인 경우})$$

$$z_1^{[1]} = \overrightarrow{w}_1^{[1]} \cdot \overrightarrow{x} + b^{[1]} \ , \qquad z_1^{[2]} = \overrightarrow{w}_1^{[2]} \cdot \overrightarrow{a} + b^{[2]}$$

- **벡터 연산을 행렬 연산으로 바꿔서 사용할 예정**

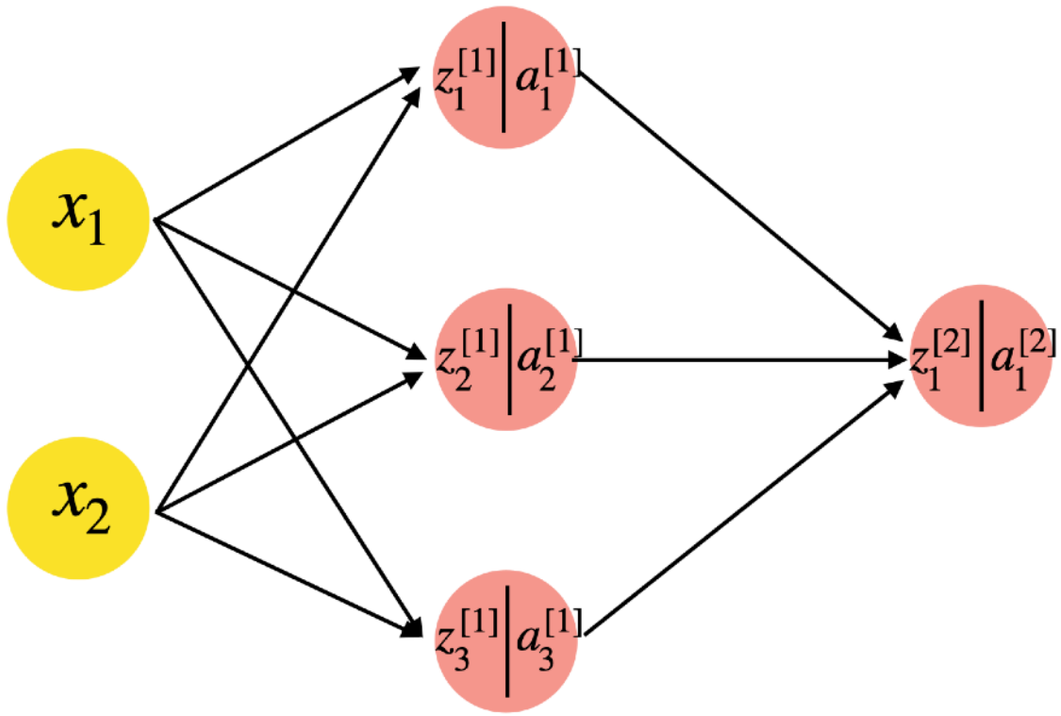
- **행렬**의 곱 $AB = C$: 행렬의 크기 $[A] = (i, j), \quad [B] = (j, k) \rightarrow [C] = (i, k)$

$$AB = \begin{pmatrix} \boxed{a_{11}\ a_{12}} \\ a_{21}\ a_{22} \end{pmatrix} \begin{pmatrix} \boxed{b_{11}} b_{12} \\ \boxed{b_{21}} b_{22} \end{pmatrix} = \begin{pmatrix} \boxed{a_{11}b_{11}+a_{12}b_{21}} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} \boxed{a_{11}\ a_{12}} \\ a_{21}\ a_{22} \end{pmatrix} \begin{pmatrix} b_{11} \boxed{b_{12}} \\ b_{21} \boxed{b_{22}} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21} & \boxed{a_{11}b_{12}+a_{12}b_{22}} \\ a_{21}b_{11}+a_{22}b_{21} & a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11}\ a_{12} \\ \boxed{a_{21}\ a_{22}} \end{pmatrix} \begin{pmatrix} \boxed{b_{11}} b_{12} \\ \boxed{b_{21}} b_{22} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ \boxed{a_{21}b_{11}+a_{22}b_{21}} & a_{21}b_{12}+a_{22}b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11}\ a_{12} \\ \boxed{a_{21}\ a_{22}} \end{pmatrix} \begin{pmatrix} b_{11} \boxed{b_{12}} \\ b_{21} \boxed{b_{22}} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} \\ a_{21}b_{11}+a_{22}b_{21} & \boxed{a_{21}b_{12}+a_{22}b_{22}} \end{pmatrix}$$



$$z_1^{[1]} = w_1^{[1]t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right)$$

$$z_2^{[1]} = w_2^{[1]t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right)$$

$$z_3^{[1]} = w_3^{[1]t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \qquad n_x : \textbf{특성 개수 (2)} \qquad w_i^{[j]} = \begin{pmatrix} w_{(i,1)}^{[j]} \\ w_{(i,2)}^{[j]} \\ \vdots \\ w_{(i,n_x)}^{[j]} \end{pmatrix}$$

w 의 전치행렬
(Transpose)

$$\rightarrow w_i^{[j]t} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots w_{(i,n_x)}^{[j]}\right)$$

- **행렬**의 곱 $AB = C$: 행렬의 크기 $[A] = (i,j), \quad [B] = (j,k) \rightarrow [C] = (i,k)$

$$\left[w_1^{[1]}\right] = (2,1) \qquad w_1^{[1]}x = ?$$

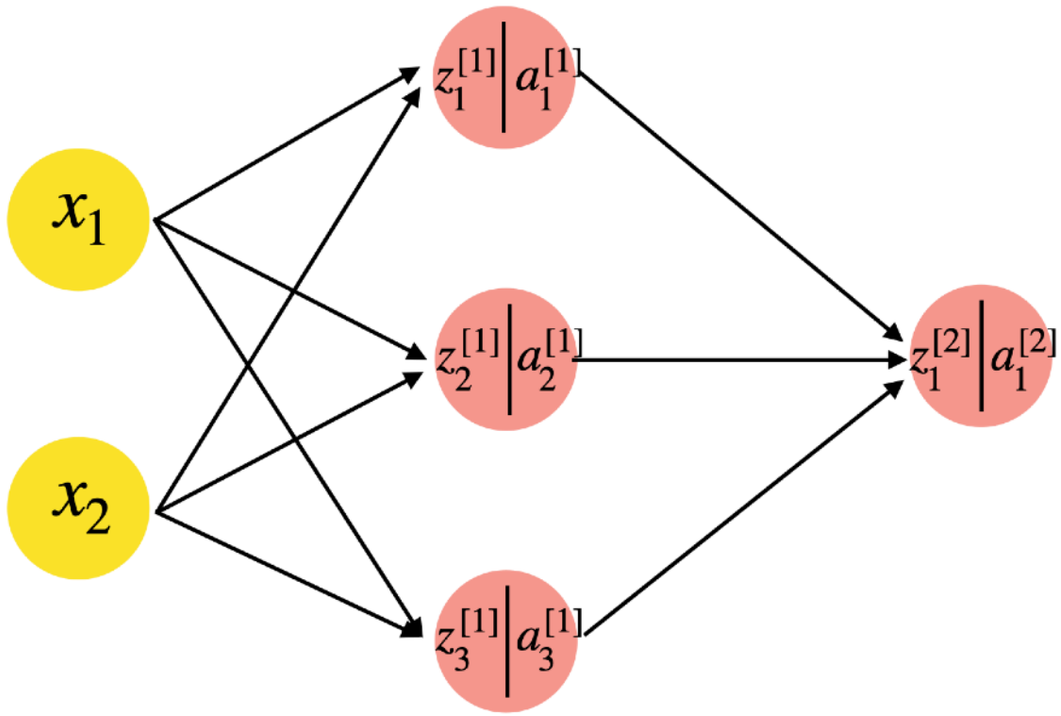
$$[x] = (2,1)$$

$$\left[w_1^{[1]t}\right] = (1,2)$$

$$[x] = (2,1)$$

$$w_1^{[1]t} x = \underline{(1,1)\ \textbf{행렬}}$$

= 스칼라



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]\,t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right) \\
z_2^{[1]} &= w_2^{[1]\,t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right) \\
z_3^{[1]} &= w_3^{[1]\,t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)
\end{aligned}
$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

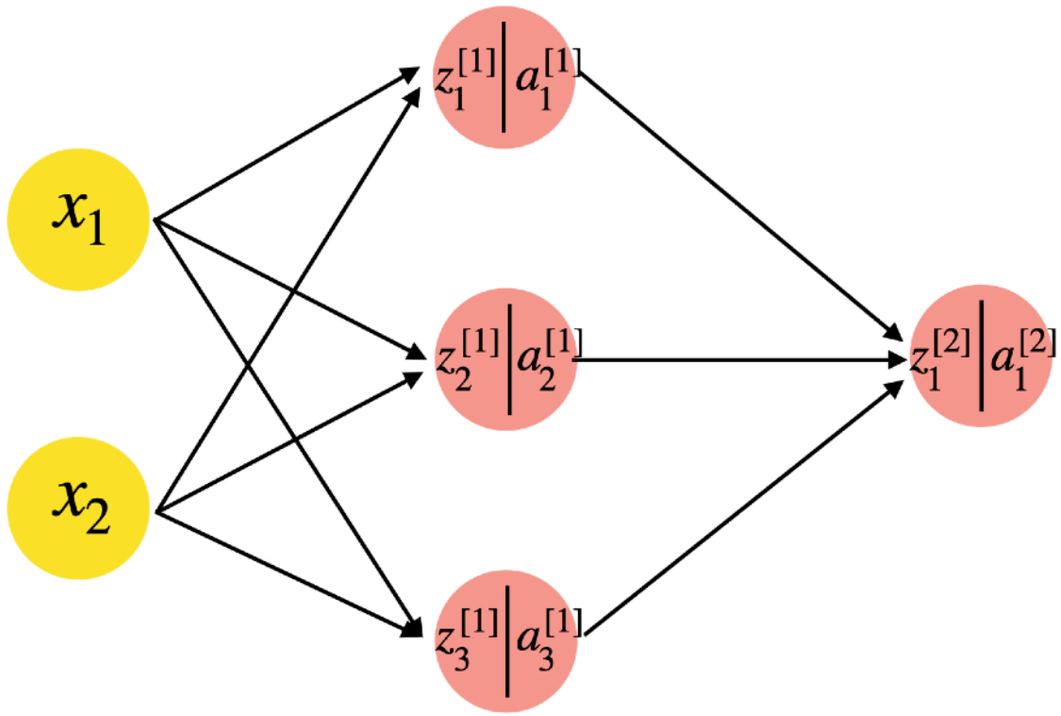
n_x : **특성 개수 (2)**

$$w_i^{[j]} = \begin{pmatrix} w_{(i,1)}^{[j]} \\ w_{(i,2)}^{[j]} \\ \vdots \\ w_{(i,n_x)}^{[j]} \end{pmatrix}$$

w 의 전치행렬 (Transpose)

$$\rightarrow w_i^{[j]\,t} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots w_{(i,n_x)}^{[j]} \right)$$

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,n_x)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,n_x)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,n_x)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]\,t} \\ w_2^{[1]\,t} \\ w_3^{[1]\,t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \equiv W^{[1]} x + b^{[1]}$$



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]\,t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right) \\
z_2^{[1]} &= w_2^{[1]\,t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right) \\
z_3^{[1]} &= w_3^{[1]\,t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)
\end{aligned}
$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

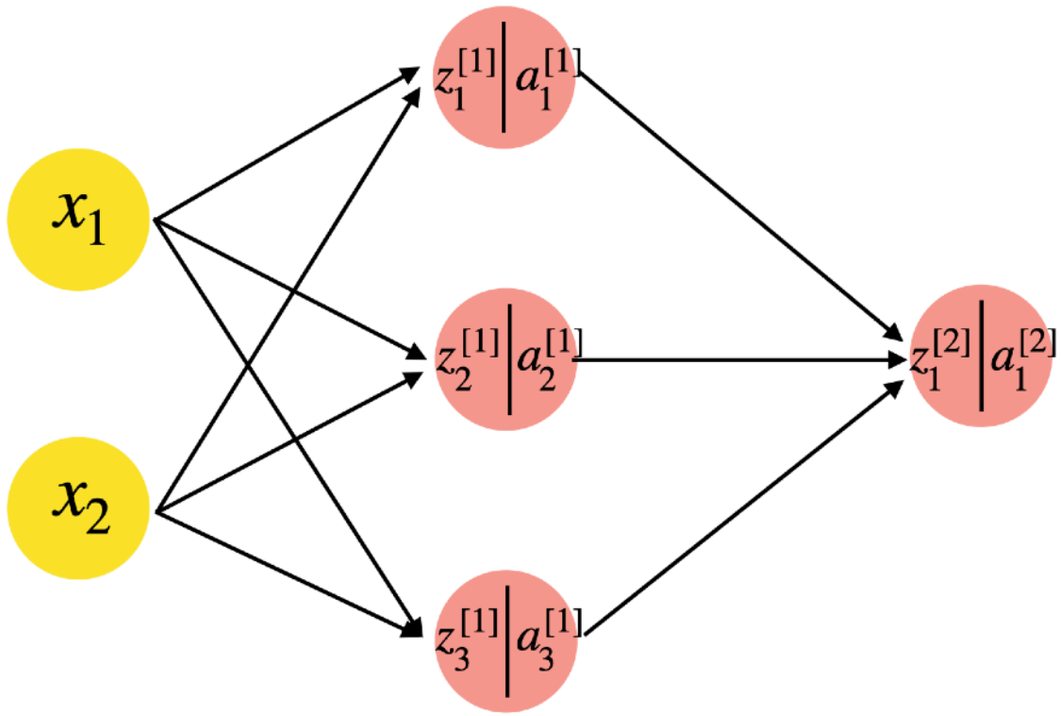
n_x : **특성 개수 (2)**

$$w_i^{[j]} = \begin{pmatrix} w_{(i,1)}^{[j]} \\ w_{(i,2)}^{[j]} \\ \vdots \\ w_{(i,n_x)}^{[j]} \end{pmatrix}$$

w 의 전치행렬 (Transpose)

$$\rightarrow w_i^{[j]\,t} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots w_{(i,n_x)}^{[j]} \right)$$

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,n_x)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,n_x)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,n_x)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]\,t} \\ w_2^{[1]\,t} \\ w_3^{[1]\,t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \equiv W^{[1]} x + b^{[1]}$$



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]\,t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right) \\
z_2^{[1]} &= w_2^{[1]\,t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right) \\
z_3^{[1]} &= w_3^{[1]\,t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)
\end{aligned}
$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

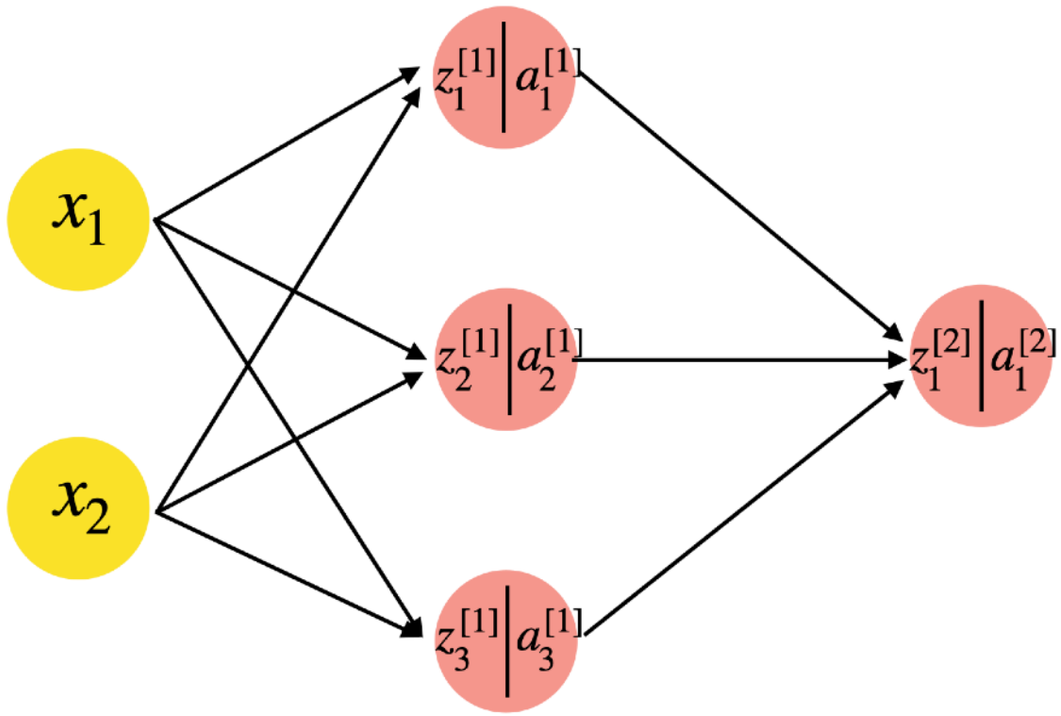
n_x : **특성 개수 (2)**

$$w_i^{[j]} = \begin{pmatrix} w_{(i,1)}^{[j]} \\ w_{(i,2)}^{[j]} \\ \vdots \\ w_{(i,n_x)}^{[j]} \end{pmatrix}$$

w 의 전치행렬 (Transpose)

$$\rightarrow w_i^{[j]\,t} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots w_{(i,n_x)}^{[j]} \right)$$

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,n_x)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,n_x)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,n_x)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]\,t} \\ w_2^{[1]\,t} \\ w_3^{[1]\,t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \equiv W^{[1]} x + b^{[1]}$$



$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]\,t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right) \\
z_2^{[1]} &= w_2^{[1]\,t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right) \\
z_3^{[1]} &= w_3^{[1]\,t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)
\end{aligned}
$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

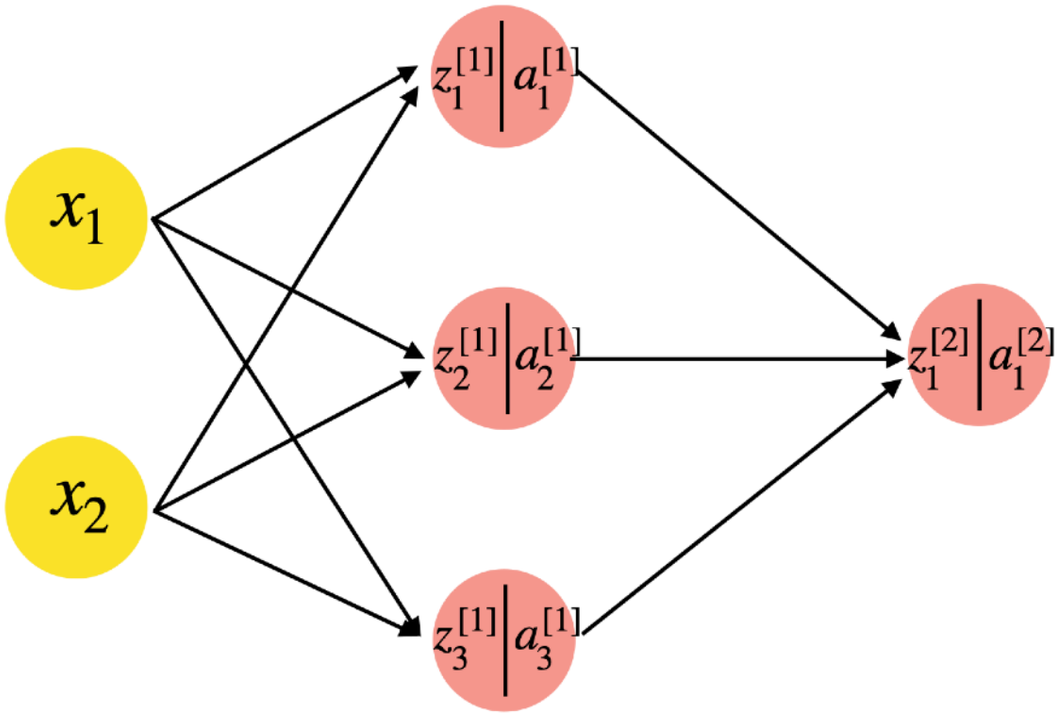
n_x : **특성 개수 (2)**

$$w_i^{[j]} = \begin{pmatrix} w_{(i,1)}^{[j]} \\ w_{(i,2)}^{[j]} \\ \vdots \\ w_{(i,n_x)}^{[j]} \end{pmatrix}$$

w 의 전치행렬 (Transpose)

$$\rightarrow w_i^{[j]\,t} = \left(w_{(i,1)}^{[j]}, w_{(i,2)}^{[j]}, \cdots w_{(i,n_x)}^{[j]} \right)$$

$$z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,n_x)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,n_x)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,n_x)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]\,t} \\ w_2^{[1]\,t} \\ w_3^{[1]\,t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \equiv W^{[1]} x + b^{[1]}$$



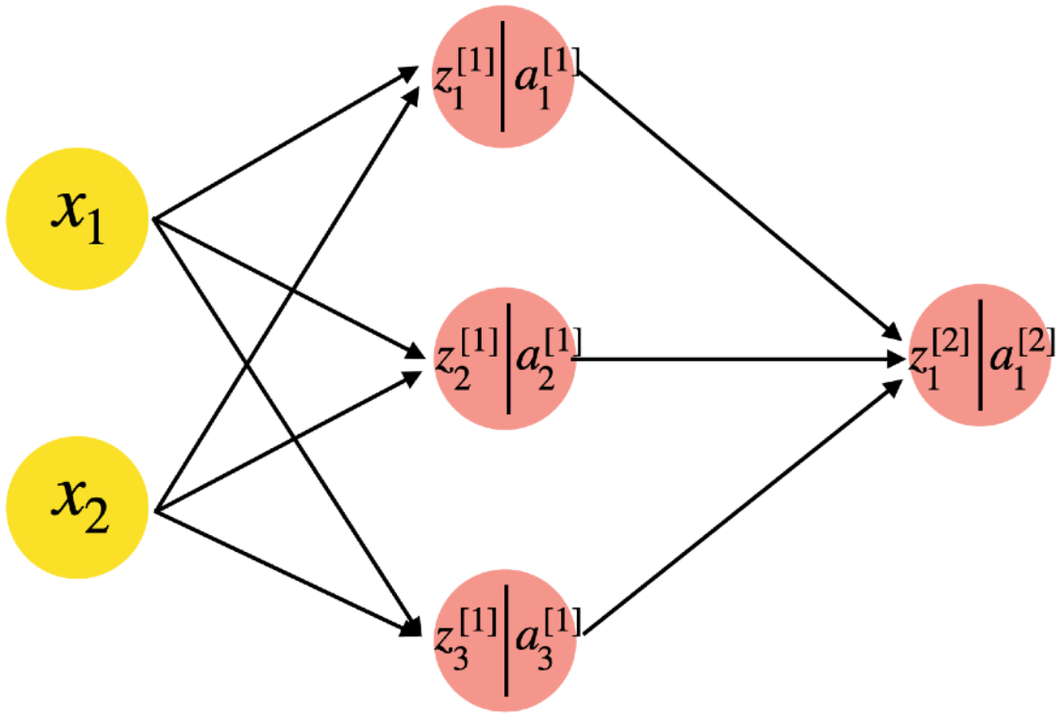
$$
\begin{aligned}
z_1^{[1]} &= w_1^{[1]t} x + b_1^{[1]} \quad \rightarrow a_1^{[1]} = g^{[1]}\left(z_1^{[1]}\right) \\
z_2^{[1]} &= w_2^{[1]t} x + b_2^{[1]} \quad \rightarrow a_2^{[1]} = g^{[1]}\left(z_2^{[1]}\right) \\
z_3^{[1]} &= w_3^{[1]t} x + b_3^{[1]} \quad \rightarrow a_3^{[1]} = g^{[1]}\left(z_3^{[1]}\right)
\end{aligned}
$$

$$
z^{[1]} = \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_{(1,1)}^{[1]} & w_{(1,2)}^{[1]} & \cdots & w_{(1,n_x)}^{[1]} \\ w_{(2,1)}^{[1]} & w_{(2,2)}^{[1]} & \cdots & w_{(2,n_x)}^{[1]} \\ w_{(3,1)}^{[1]} & w_{(3,2)}^{[1]} & \cdots & w_{(3,n_x)}^{[1]} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} = \begin{pmatrix} w_1^{[1]t} \\ w_2^{[1]t} \\ w_3^{[1]t} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{n_x} \end{pmatrix} + \begin{pmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{pmatrix} \equiv W^{[1]}x + b^{[1]}
$$

$$
a^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} = \begin{pmatrix} g^{[1]}(z_1^{[1]}) \\ g^{[1]}(z_2^{[1]}) \\ g^{[1]}(z_3^{[1]}) \end{pmatrix} = g^{[1]}(z^{[1]}) \quad \textbf{(Vectorizing)}
$$

$$
z^{[2]} = \left(z_1^{[2]}\right) = \begin{pmatrix} w_{(1,1)}^{[2]} & w_{(1,2)}^{[2]} & w_{(1,3)}^{[2]} \end{pmatrix} \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} + \left(b_1^{[2]}\right) = \left(w_1^{[2]t}\right) \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} + \left(b_1^{[2]}\right) \equiv W^{[2]}a^{[1]} + b^{[2]}
$$

- 데이터가 m 개가 있는 경우
  (예: 테스트 데이터가 1,000 개가 있는 경우 $m = 1000$)



$$z_1^{[1](k)} = w_1^{[1]\,t}\, x^{(k)} + b_1^{[1]} \rightarrow a_1^{[1](k)} = g^{[1]}\left(z_1^{[1](k)}\right)$$

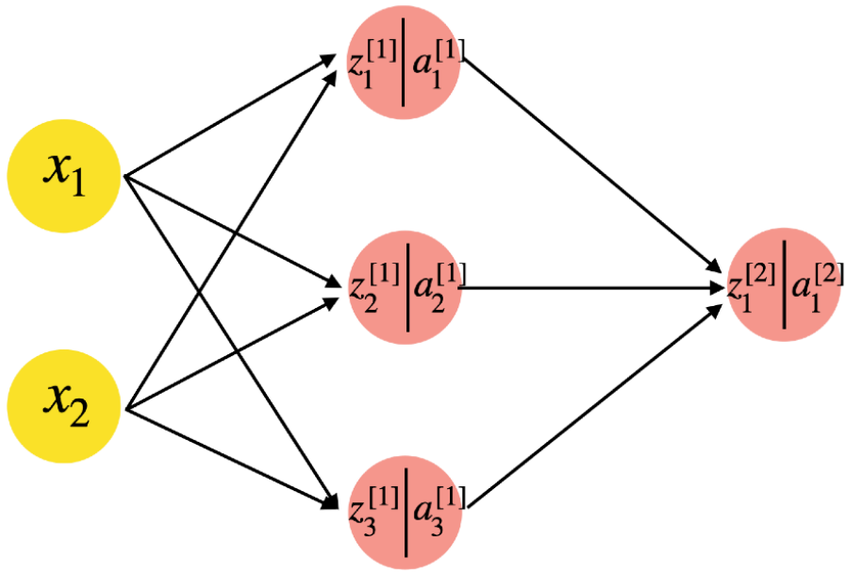
$$z_2^{[1](k)} = w_2^{[1]\,t}\, x^{(k)} + b_2^{[1]} \rightarrow a_2^{[1](k)} = g^{[1]}\left(z_2^{[1](k)}\right)$$

$$z_3^{[1](k)} = w_3^{[1]\,t}\, x^{(k)} + b_3^{[1]} \rightarrow a_3^{[1](k)} = g^{[1]}\left(z_3^{[1](k)}\right)$$

- 표준 표기법:
  $x_i^{(k)}$: i 번째 특성, k 번째 데이터
  $z_j^{[i](k)}$: i 번째 층. j 번째 노드, k 번째 데이터

- 가중치와 편향은 모든 데이터에 대해 **동일값**

- 데이터가 m 개가 있는 경우



$$z_1^{[1](k)} = w_1^{[1]\,t}\, x^{(k)} + b_1^{[1]} \rightarrow a_1^{[1](k)} = g^{[1]}\left(z_1^{[1](k)}\right)$$

$$z_2^{[1](k)} = w_2^{[1]\,t}\, x^{(k)} + b_2^{[1]} \rightarrow a_2^{[1](k)} = g^{[1]}\left(z_2^{[1](k)}\right)$$

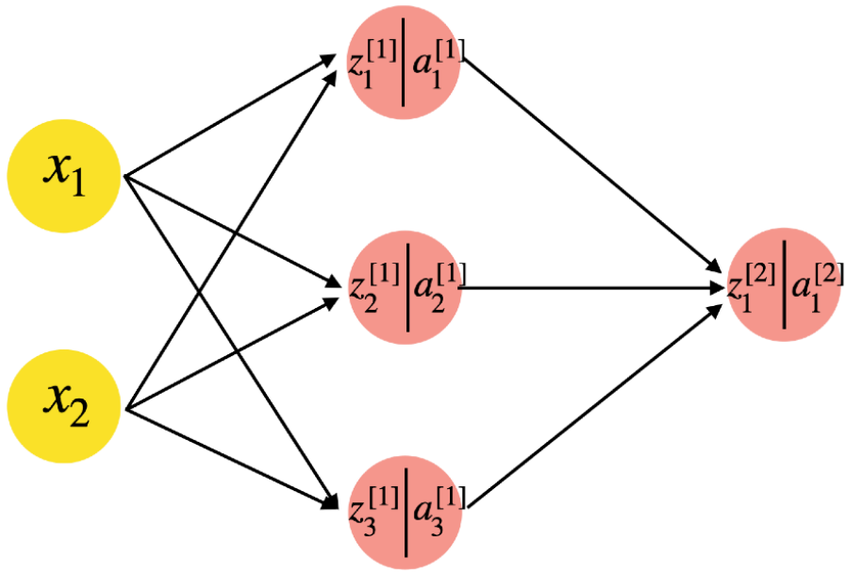
$$z_3^{[1](k)} = w_3^{[1]\,t}\, x^{(k)} + b_3^{[1]} \rightarrow a_3^{[1](k)} = g^{[1]}\left(z_3^{[1](k)}\right)$$

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \Rightarrow \quad X = \begin{pmatrix} x_1^{(1)} & x_1^{(2)} & \cdots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \cdots & x_2^{(m)} \end{pmatrix} = \begin{pmatrix} x^{(1)} & x^{(2)} & \cdots & x^{(m)} \end{pmatrix}$$

데이터 1개

데이터 m개

- 데이터가 m 개가 있는 경우



$$z_1^{[1](k)} = w_1^{[1]\,t}\, x^{(k)} + b_1^{[1]} \rightarrow a_1^{[1](k)} = g^{[1]}\left(z_1^{[1](k)}\right)$$

$$z_2^{[1](k)} = w_2^{[1]\,t}\, x^{(k)} + b_2^{[1]} \rightarrow a_2^{[1](k)} = g^{[1]}\left(z_2^{[1](k)}\right)$$

$$z_3^{[1](k)} = w_3^{[1]\,t}\, x^{(k)} + b_3^{[1]} \rightarrow a_3^{[1](k)} = g^{[1]}\left(z_3^{[1](k)}\right)$$

$$Z^{[1]} = \begin{pmatrix} z_1^{[1](1)} & z_1^{[1](2)} & \cdots & z_1^{[1](m)} \\ z_2^{[1](1)} & z_2^{[1](2)} & \cdots & z_2^{[1](m)} \\ z_3^{[1](1)} & z_3^{[1](2)} & \cdots & z_3^{[1](m)} \end{pmatrix} = \begin{pmatrix} z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \end{pmatrix}$$

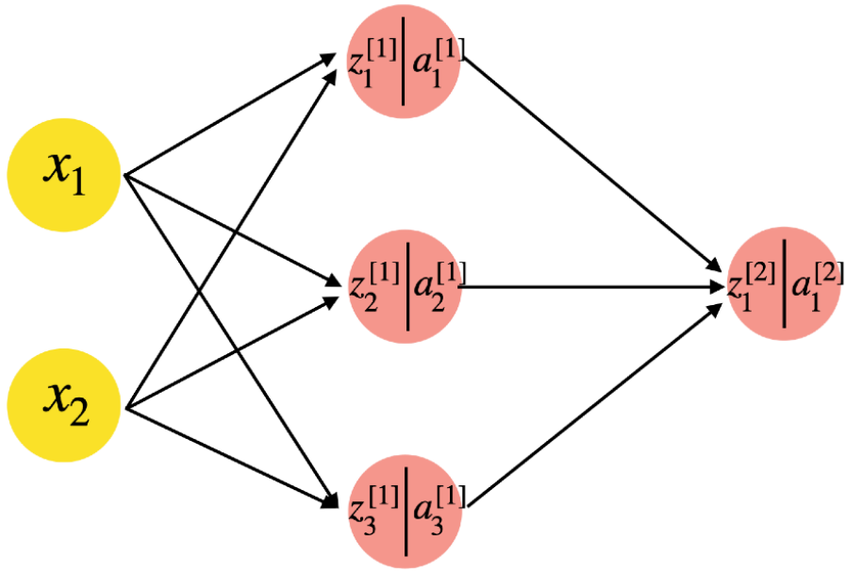
$$= \begin{pmatrix} W^{[1]}x^{(1)} & W^{[1]}x^{(2)} & \cdots & W^{[1]}x^{(m)} \end{pmatrix} + b^{[1]}$$

(Overriding)

$$= W^{[1]}X + b^{[1]}$$

(Vectorizing)

- 데이터가 m 개가 있는 경우



$$z_1^{[1](k)} = w_1^{[1]\,t}\, x^{(k)} + b_1^{[1]} \rightarrow a_1^{[1](k)} = g^{[1]}\left(z_1^{[1](k)}\right)$$

$$z_2^{[1](k)} = w_2^{[1]\,t}\, x^{(k)} + b_2^{[1]} \rightarrow a_2^{[1](k)} = g^{[1]}\left(z_2^{[1](k)}\right)$$

$$z_3^{[1](k)} = w_3^{[1]\,t}\, x^{(k)} + b_3^{[1]} \rightarrow a_3^{[1](k)} = g^{[1]}\left(z_3^{[1](k)}\right)$$

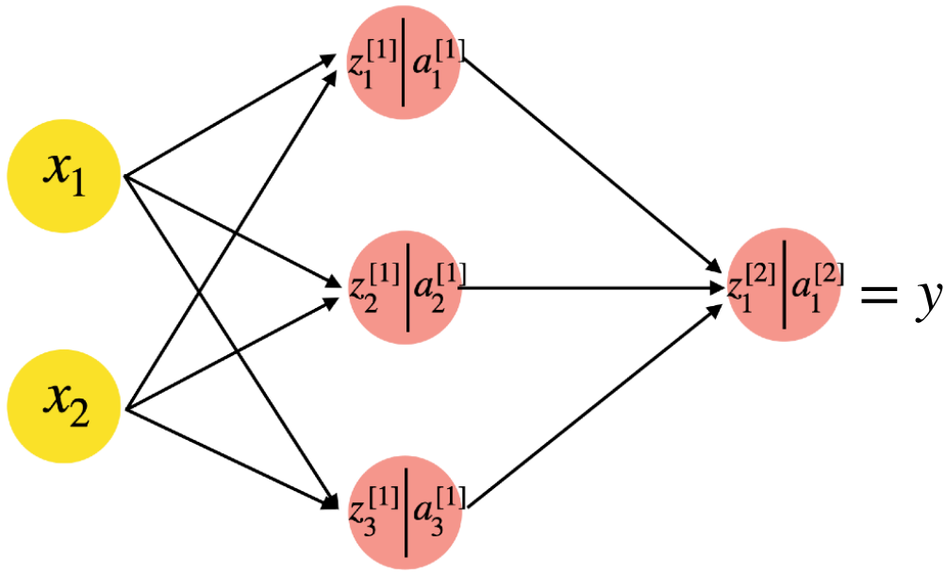
데이터 1개 $\quad a^{[1]} = \begin{pmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{pmatrix} = \begin{pmatrix} g^{[1]}(z_1^{[1]}) \\ g^{[1]}(z_2^{[1]}) \\ g^{[1]}(z_3^{[1]}) \end{pmatrix} = g^{[1]} \begin{pmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \end{pmatrix} = g^{[1]}(z^{[1]})$

데이터 m개

$$A^{[1]} \equiv \begin{pmatrix} a_1^{[1](1)} & a_1^{[1](2)} & \cdots & a_1^{[1](m)} \\ a_2^{[1](1)} & a_2^{[1](2)} & \cdots & a_2^{[1](m)} \\ a_3^{[1](1)} & a_3^{[1](2)} & \cdots & a_3^{[1](m)} \end{pmatrix} = \begin{pmatrix} a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \end{pmatrix} = g^{[1]}(Z^{[1]})$$

(Vectorizing)

- 데이터가 m 개가 있는 경우



$$z_1^{[2](k)} = w_1^{[2]\,t}\, a^{[1](k)} + b_1^{[2]} \rightarrow a_1^{[2](k)} = g^{[2]}\left(z_1^{[2](k)}\right)$$

$$Z^{[2]} = \left(z_1^{[2](1)} \quad z_1^{[2](2)} \quad \cdots \quad z_1^{[2](m)}\right) = \left(z^{[2](1)} \quad z^{[2](2)} \quad \cdots \quad z^{[2](m)}\right)$$

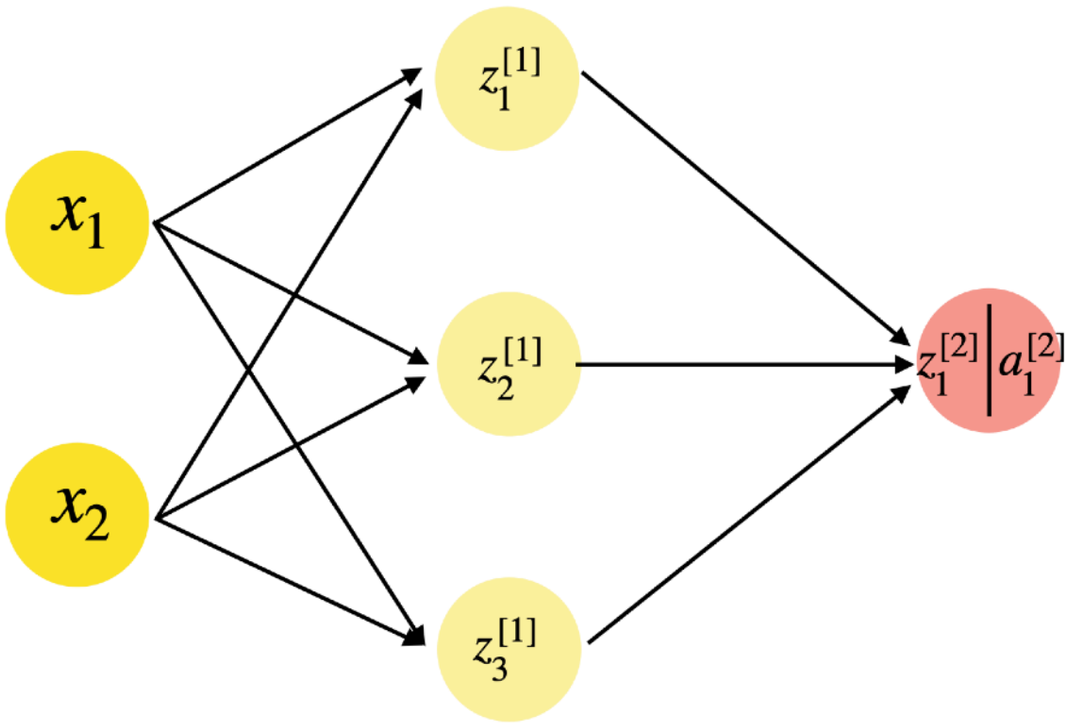
$$= \left(W^{[2]}a^{[1](1)} \quad W^{[2]}a^{[1](2)} \quad \cdots \quad W^{[2]}a^{[1](m)}\right) + b^{[2]}$$

(Overriding)

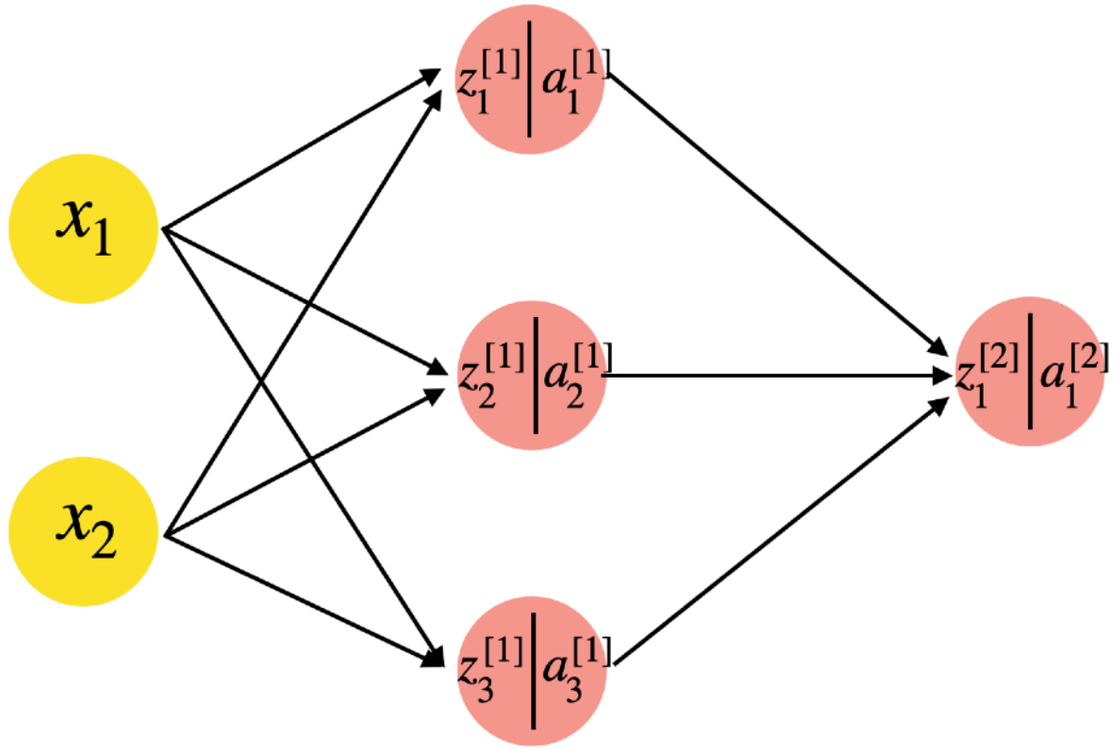
$$= W^{[2]}A^{[1]} + b^{[2]}$$

(Vectorizing)

$$(1,m) = (1,3)(3,m) + (1,1)$$



V. S.



- 은닉층에 활성화 함수가 필요한가?

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$\left[W^{[1]}\right] = (3,2) \qquad \left[b^{[1]}\right] = (3,1)$$

$$z^{[2]} = W^{[2]}z^{[1]} + b^{[2]}$$

$$\left[W^{[2]}\right] = (1,3) \qquad \left[b^{[2]}\right] = (1,1)$$

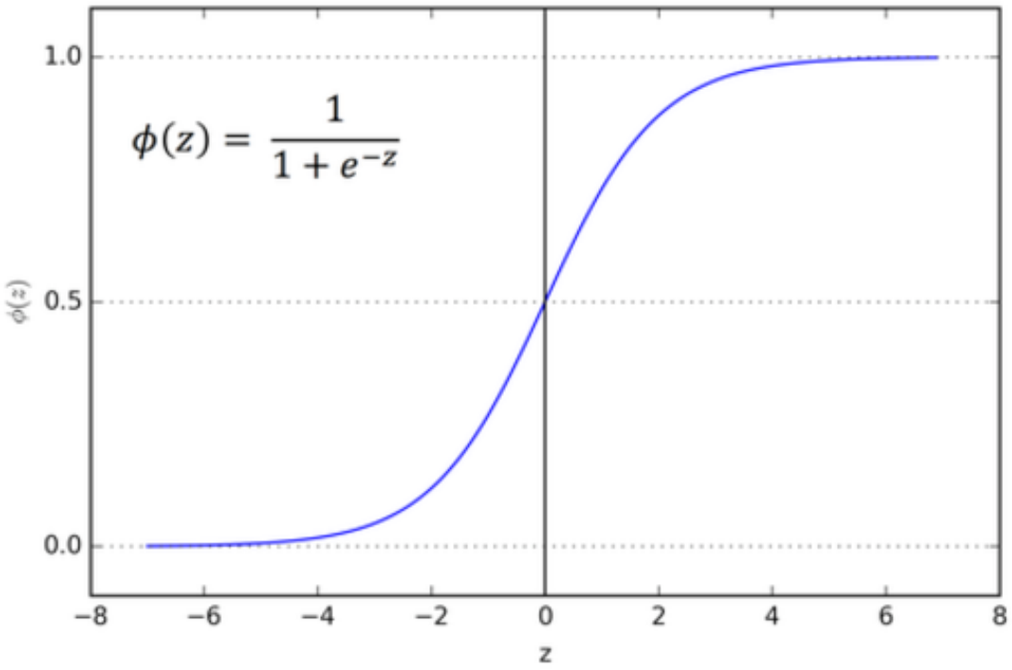
$$z^{[2]} = W^{[2]}\left(W^{[1]}x + b^{[1]}\right) + b^{[2]} = \left(W^{[2]}W^{[1]}\right)x + \left(W^{[2]}b^{[1]} + b^{[2]}\right)$$

$$= Wx + b$$

$$[W] = (1,2) \qquad \left[b\right] = (1,1)$$

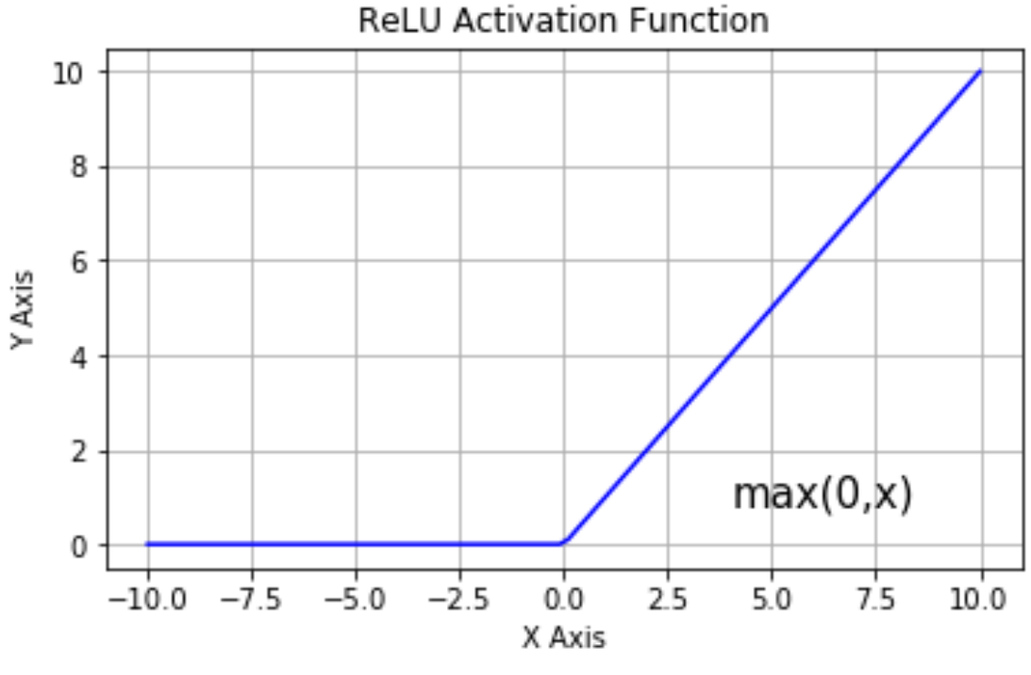
- (Deep-network) 은닉층에 어떠한 활성화 함수가 좋은가?

Sigmoid



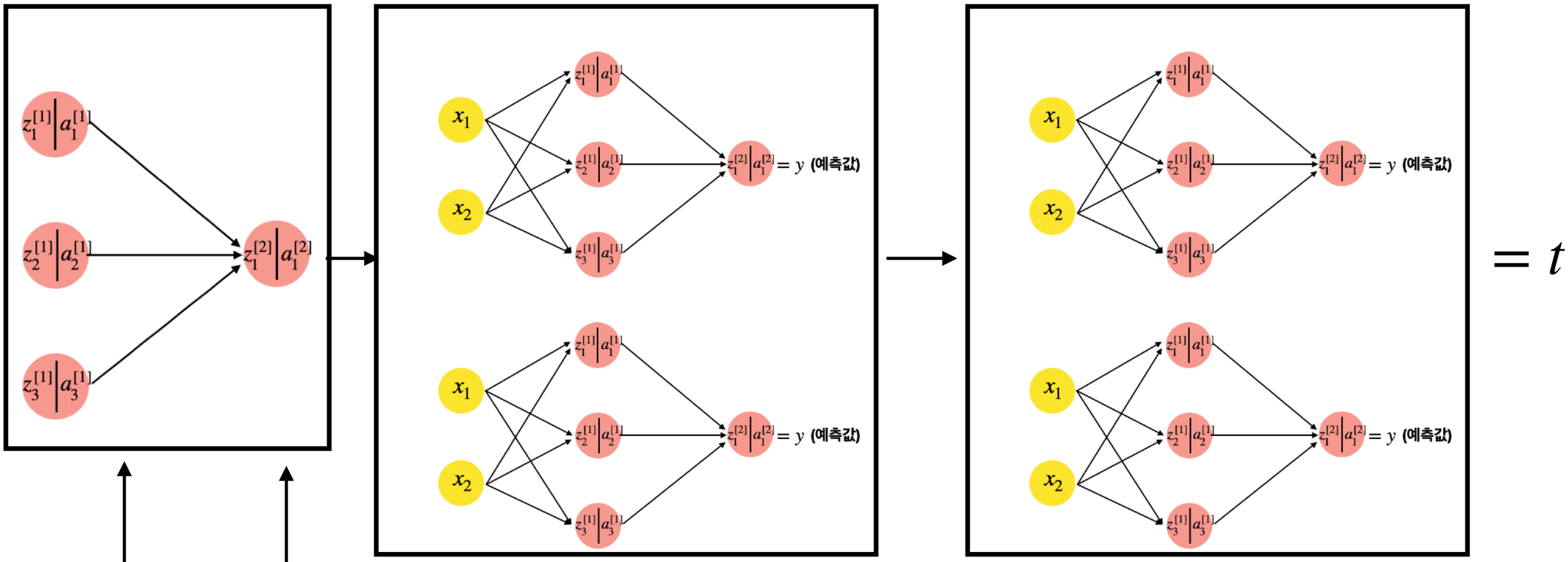
- 확률적인 해석을 가능하게 해줌.
- 큰 값에서 Vanishing Gradient 문제 발생
- 분류 문제의 경우, 출력층에서 사용

ReLU (Rectified Linear Unit)

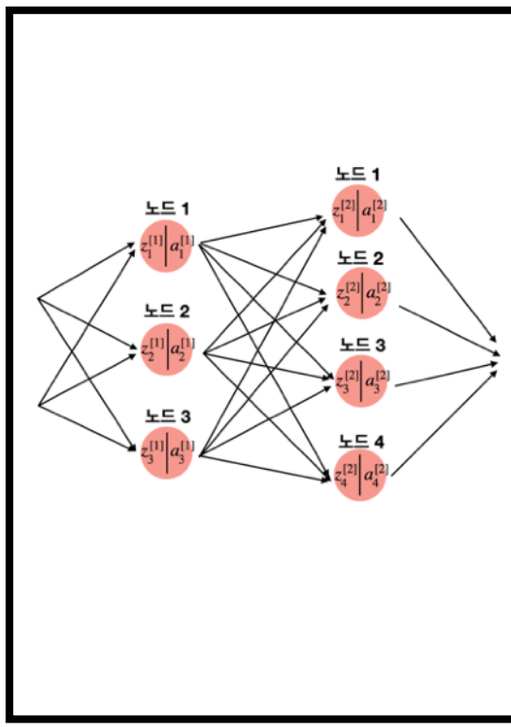


- Vanishing Gradient 문제에서 자유로움
- Hidden Layer (은닉층) 에서 사용

- **Shallow Network (은닉층 1개) 를 학습시키기**



- **Deep Network (은닉층 $L-1$개) 를 학습시키기**



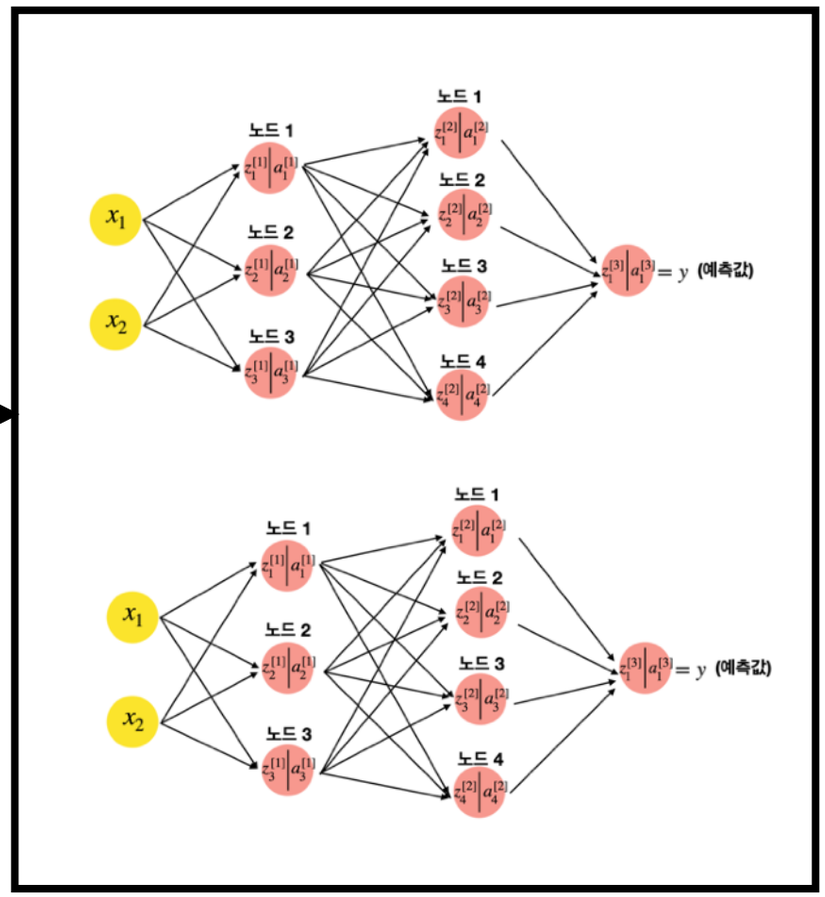
네트워크 초기화

$(W^{[1]}, b^{[1]})$

$(W^{[2]}, b^{[2]})$

• • •

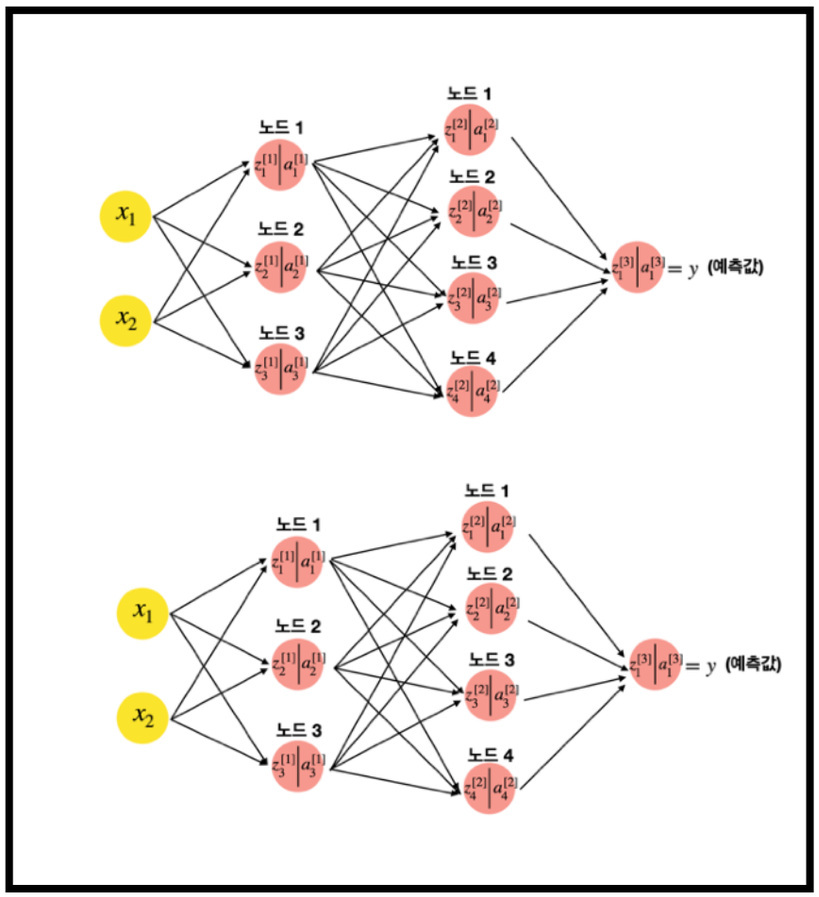
$(W^{[L]}, b^{[L]})$



데이터를 입력하여
전파과정을 통해 예측값 산출

$(Z^{[1]}, A^{[1]}, \cdots, Z^{[L]}, A^{[L]})$ 저장

$(W^{[1]}, b^{[1]}, \cdots, W^{[L]}, b^{[L]})$ 저장



예측값과 답안 비교 (비용함수 계산)

역전파를 통해
미분값 계산

$\left(dW^{[1]}, db^{[1]}, \cdots, dW^{[L]}, db^{[L]}\right)$ 저장

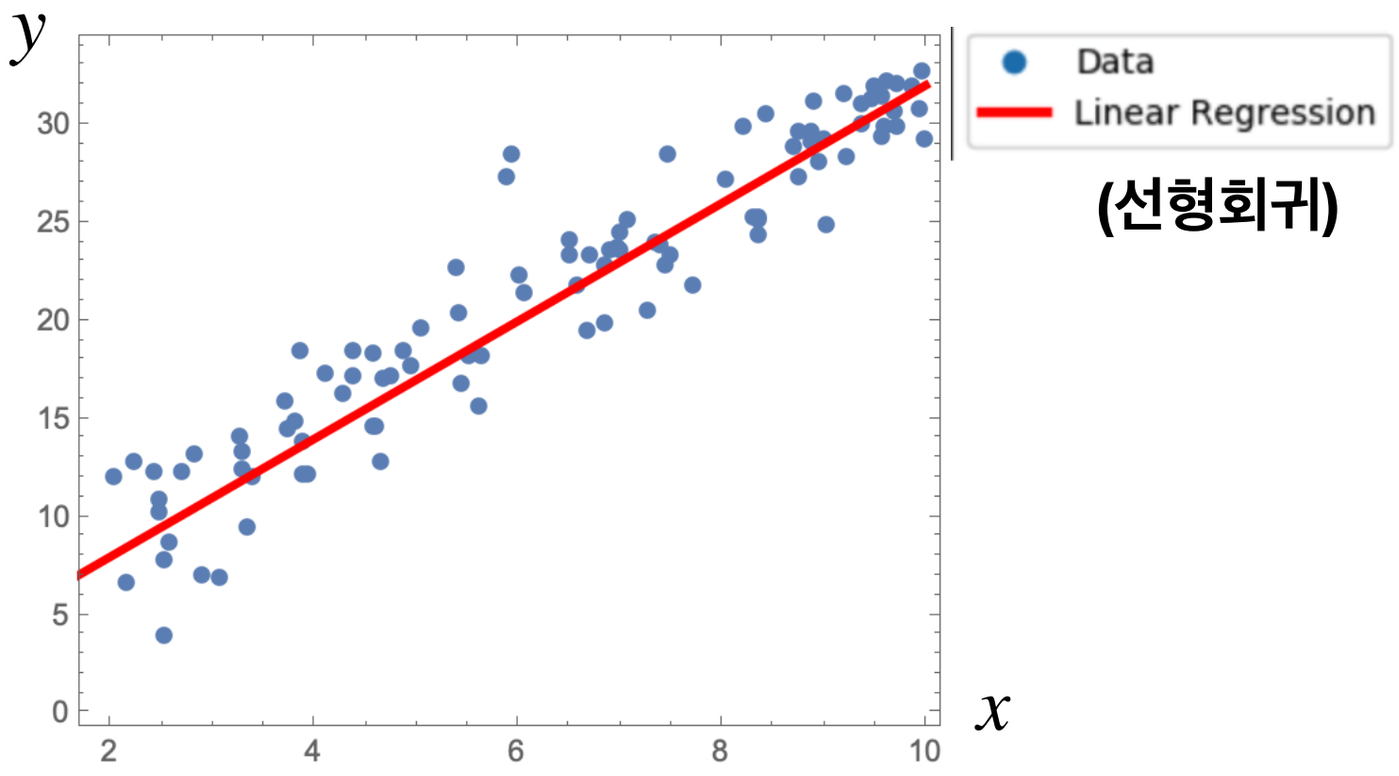
Parameter 업데이트

$$\left(W^{[1]}, b^{[1]}, \cdots, W^{[L]}, b^{[L]}\right) \leftarrow \left(W^{[1]}, b^{[1]}, \cdots, W^{[L]}, b^{[L]}\right) - \alpha \left(dW^{[1]}, db^{[1]}, \cdots, dW^{[L]}, db^{[L]}\right)$$

$= t$

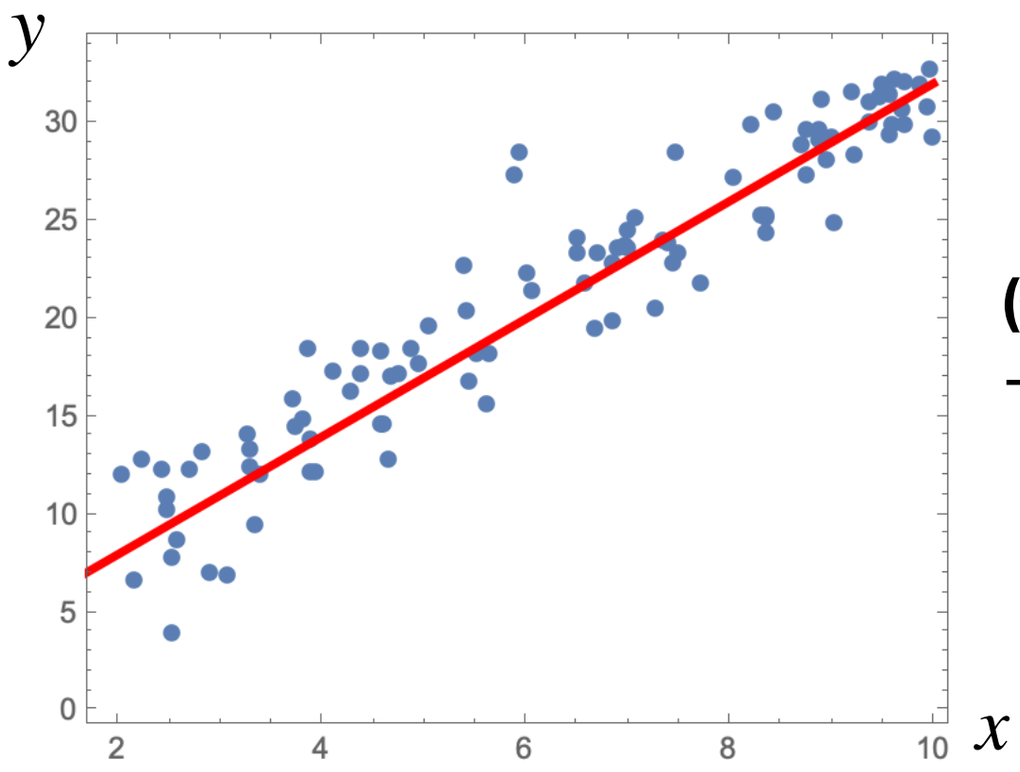
기계학습과 함수의 최적화

- 회귀 (regression)



(선형회귀)

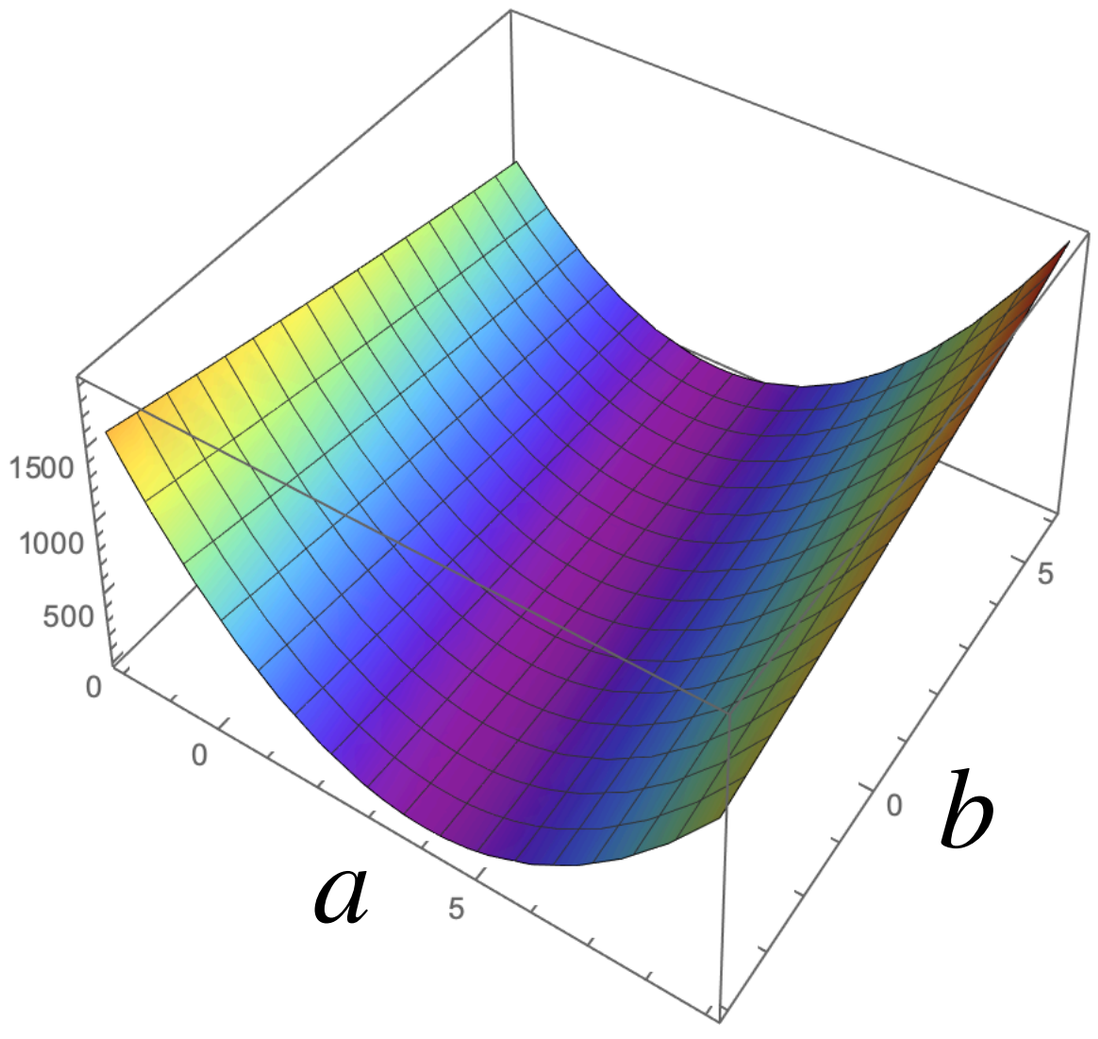
두 변수 (x, y) 의 관계 $y = f(x)$ 를 선형 $f(x) = ax + b$으로 예측하는 경우, **데이터** (x_i, y_i) **와 예측 값** $(x_i, ax_i + b)$ 의 **차이를 최소화** $L = \frac{1}{N}\sum_i \left(y_i - (ax_i + b)\right)^2$ 하는 (a, b) 를 찾아야 함.



회귀
(관계 예측)

$$L = \frac{1}{N}\sum_i \left(y_i - (ax_i + b)\right)^2$$

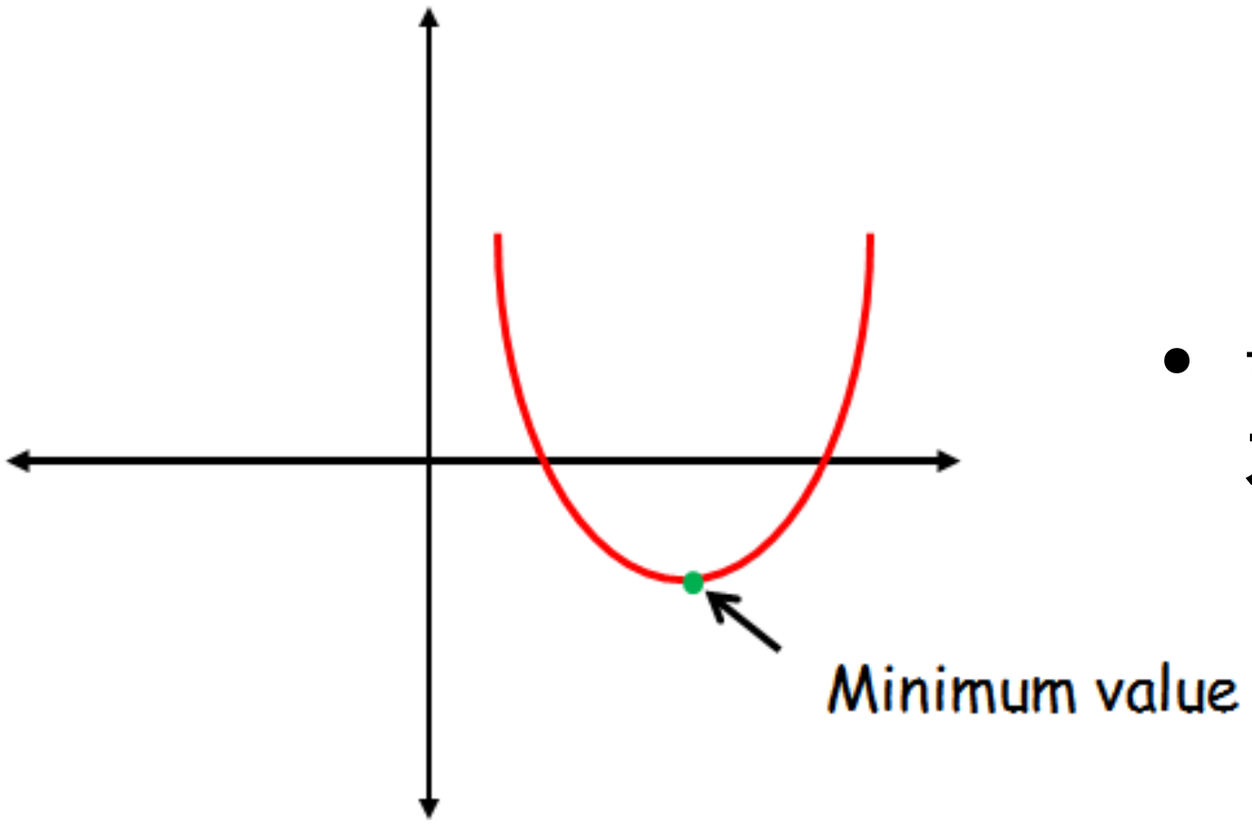
예측의 오류 최소화



- 데이터를 잘 설명하는 예측을 찾는 과정:
  함수 $L(a, b)$ 의 최솟값을 찾음.

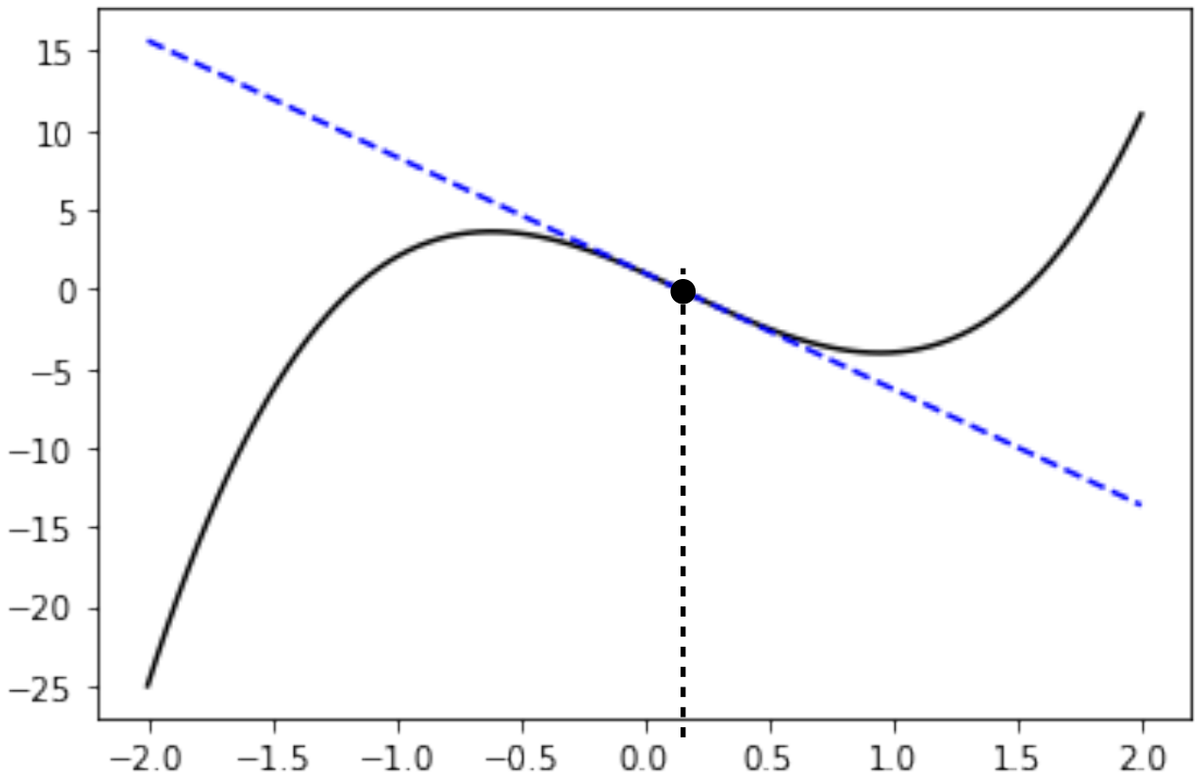
  이변수 함수 $f : \mathbb{R}^2 \to \mathbb{R}$ **미분을 활용**

함수의 극소/극대값



- 극소/극대값 주변에서는 **접선의 기울기가 작다**

경사하강법을 이용한 최솟값 찾기

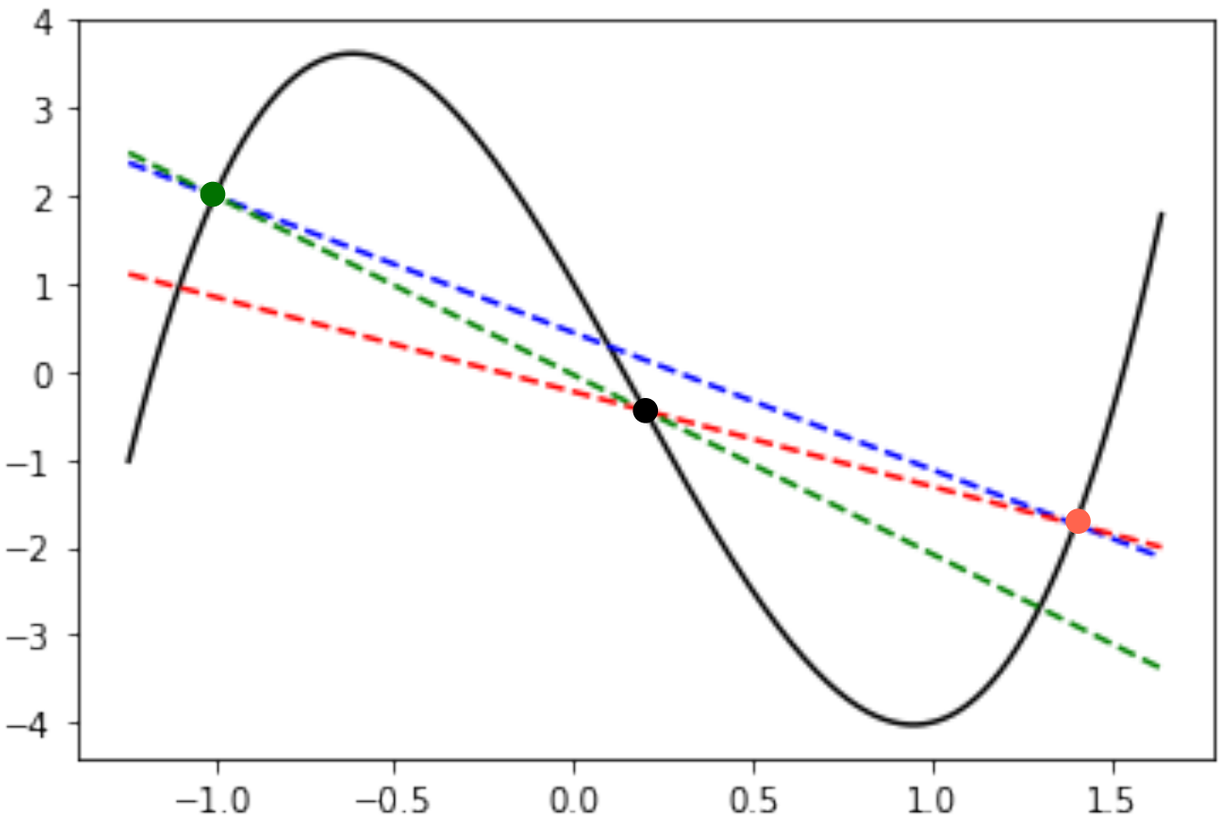


- 점 $\big(x_1, f(x_1)\big), \big(x_2, f(x_2)\big)$ 를 지나는 선분의 기울기

$$= \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

- 점 $\big(x, f(x)\big)$에서의 접선의 기울기

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



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

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

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

$$\frac{f(x+h)-f(x)}{(x+h)-x}$$

$$\frac{f(x+h)-f(x-h)}{(x+h)-(x-h)}$$

$$\frac{f(x)-f(x-h)}{x-(x-h)}$$

```python
def numerical_diff_front(f,x):
    h=1e-4 # 0.0001
    return (f(x+h)-f(x))/h

def numerical_diff_mid(f,x):
    h=1e-4 # 0.0001
    return (f(x+h)-f(x-h))/(2*h)

def numerical_diff_back(f,x):
    h=1e-4 # 0.0001
    return (f(x)-f(x-h))/h
```

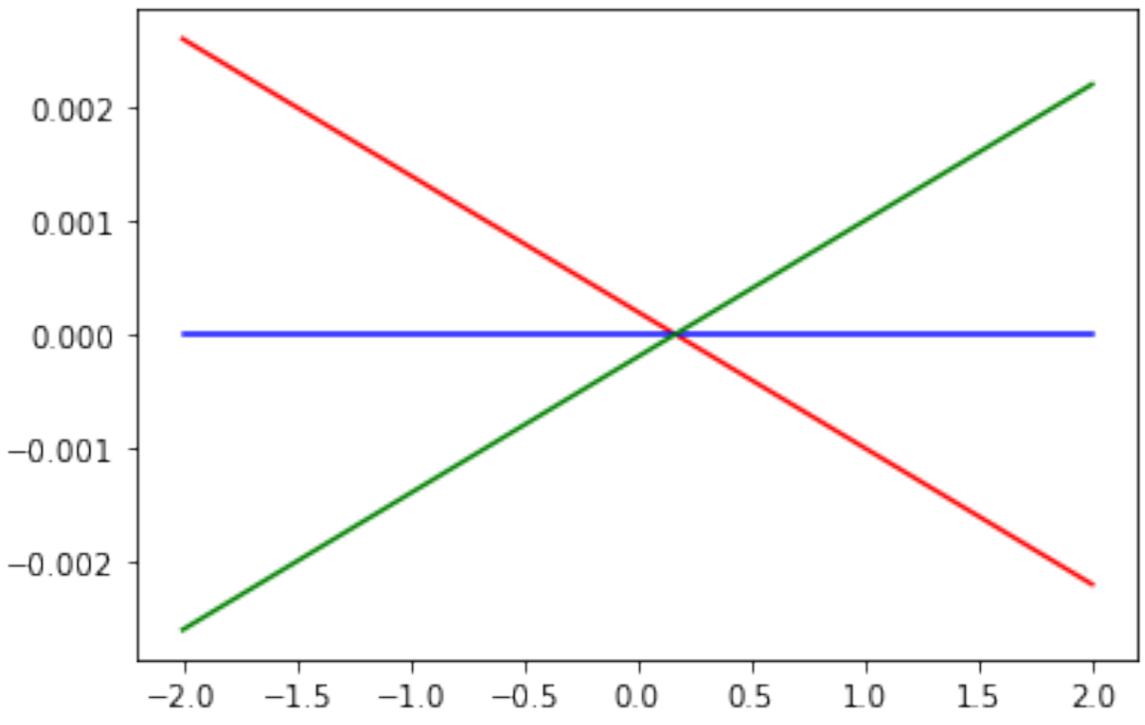
전방차분

중심차분

후방차분

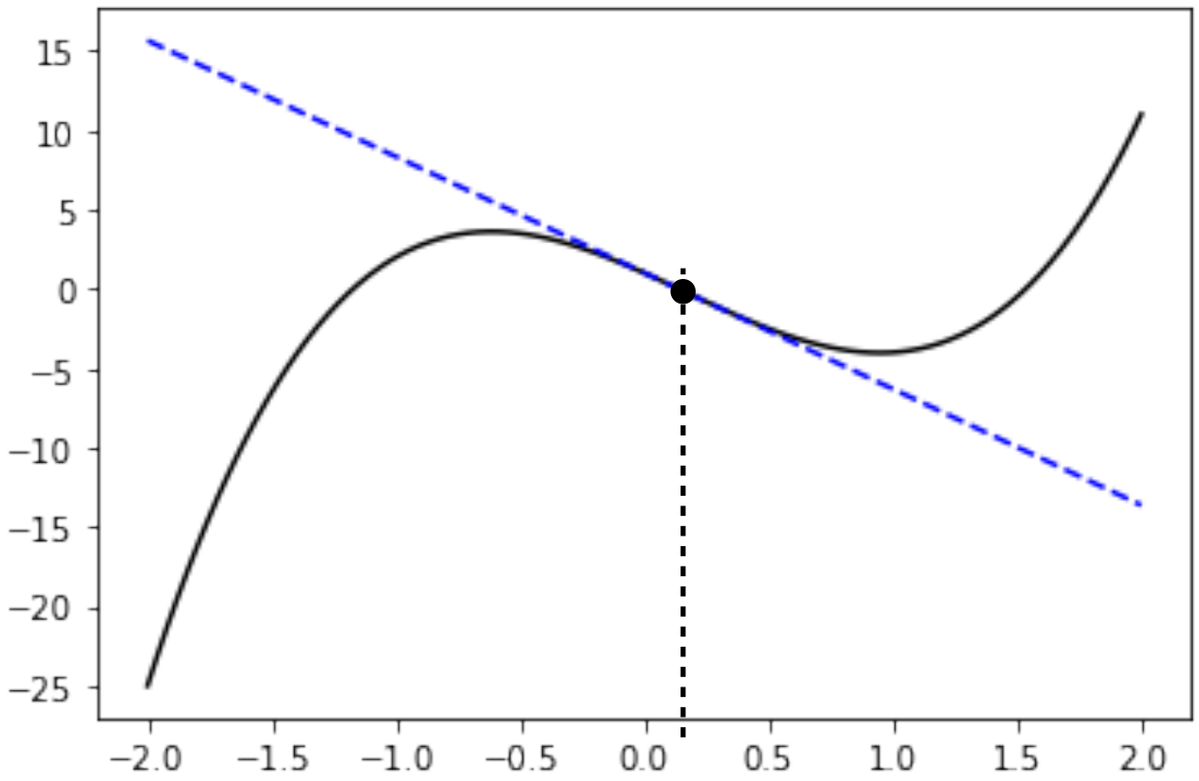
$$f(x) = 4x^3 - 2x^2 - 7x + 1$$

$f'(x)$— numerical diff



$$f'(x) = \frac{df(x)}{dx} = 12x^2 - 4x - 7$$

- 수치 미분 ($h \ll 1$ 이지만 $h \neq 0$)의 경우, 실제 미분값과 수치 미분사이에 오차가 존재.
- 중심 차분을 이용한 수치 미분이 오차가 제일 적음.
- 전방/후방 차분의 경우 오차가 $\mathcal{O}(h)$ 이지만, 중심차분의 경우 오차가 $\mathcal{O}(h^2)$ 임.



- 점 (x_0, y_0)에서의 접선의 기울기 = $f'(x_0)$

  이 접선의 방정식 $y = f'(x)(x - x_0) + y_0$

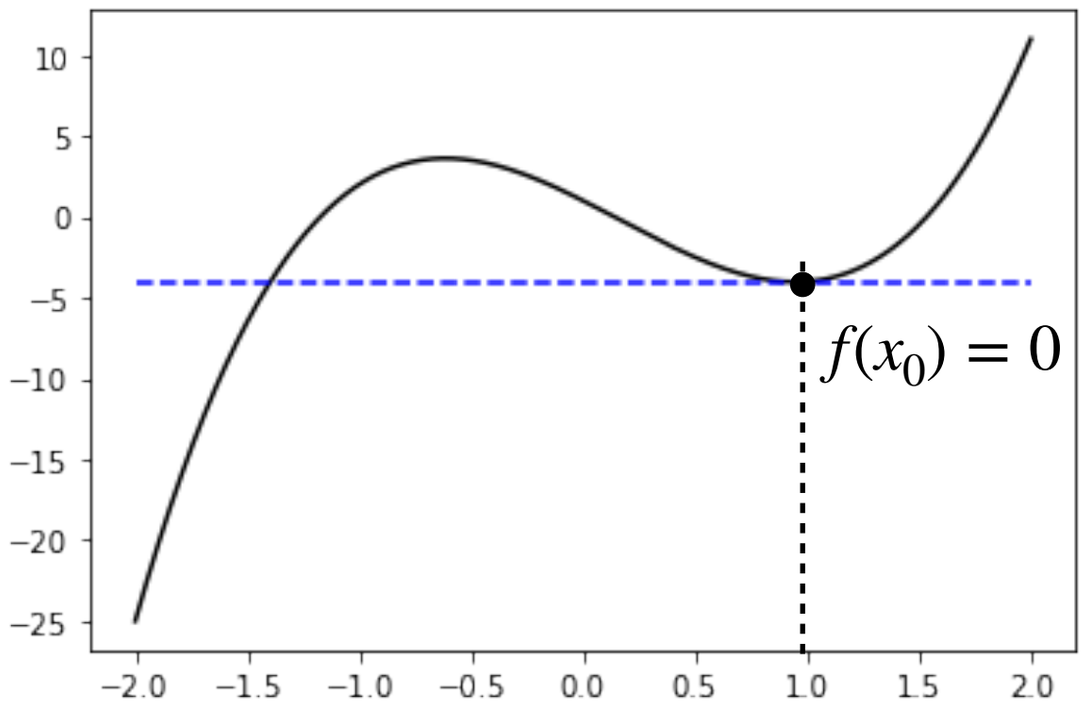
- 점 (x_0, y_0) 근처에서 함수 $f(x)$ 는 그 점에서 접선과 비슷함

$$y = f(x) \simeq f'(x_0)(x - x_0) + y_0$$

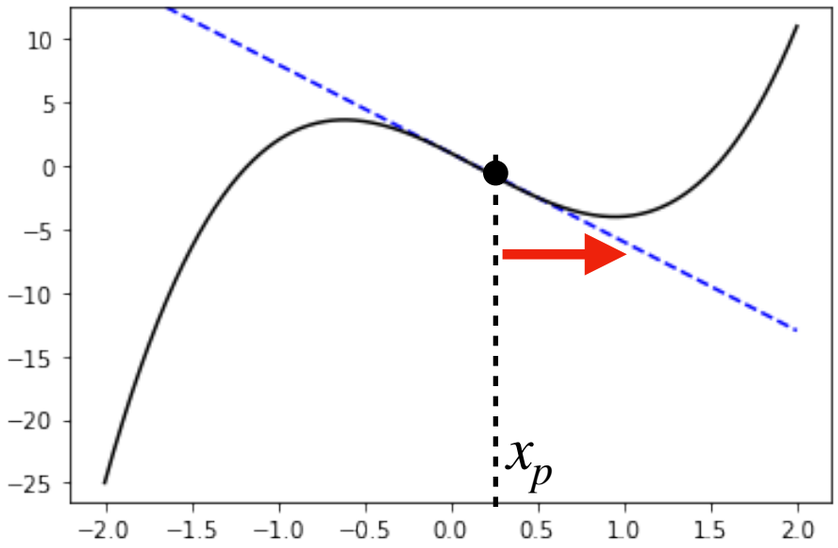
$$\Delta y = y - y_0 \ll 1$$

$$\Delta x = x - x_0 \ll 1$$

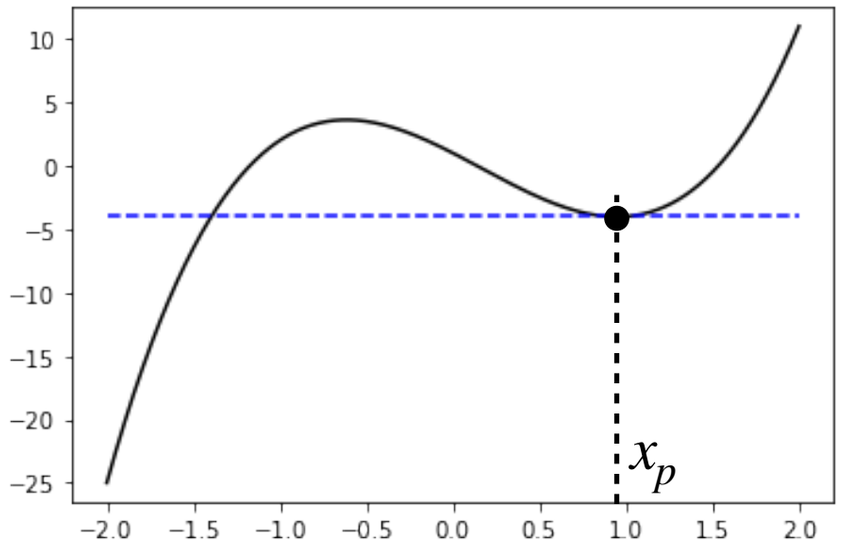
- 즉, $\Delta y \simeq f'(x_0) \times \Delta x$ 임. 따라서 $f'(x_0) = 0$ 이면, 그 근방에서 함수값의 변화가 없음: 극소/극대값.



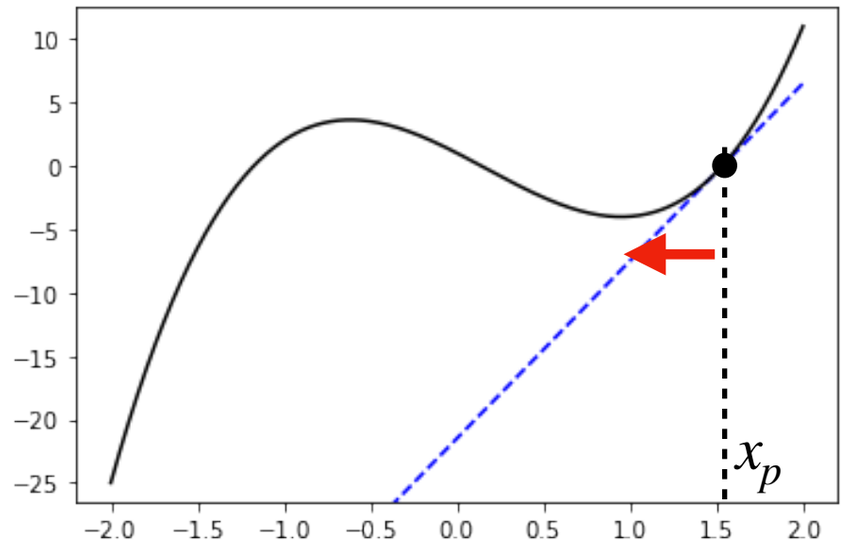
경사하강법: 극솟값 찾기



- $x_p < x_0$ 인 경우, $f'(x_p) < 0$



- $x_p = x_0$ 인 경우, $f'(x_p) = 0$



- $x_p > x_0$ 인 경우, $f'(x_p) > 0$

- $\Delta y \simeq f'(x_p) \times \Delta x$ 이므로,

  $f'(x_p) < 0$ 이면, $\Delta x > 0$ 인 경우, $\Delta y < 0$: 즉, 함수값이 감소함

  $f'(x_p) > 0$ 이면, $\Delta x < 0$ 인 경우, $\Delta y < 0$: 즉, 함수값이 감소함

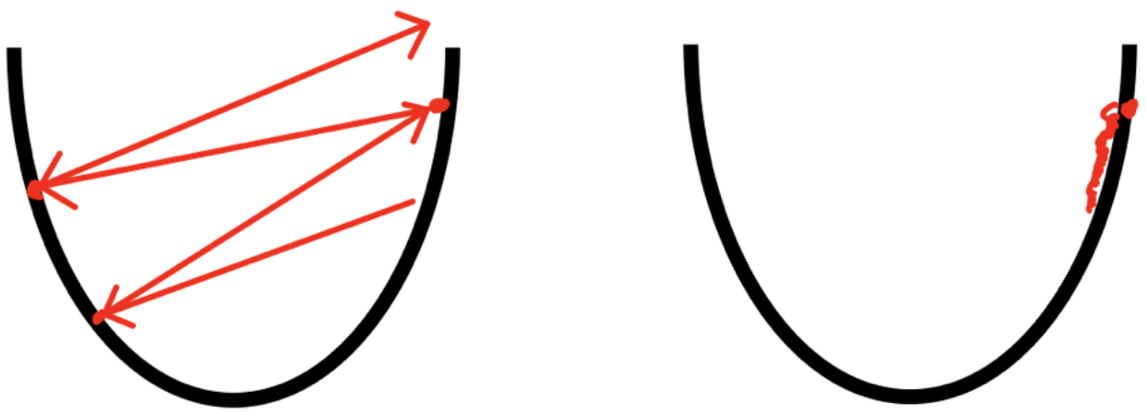
$$\Delta x \propto -f'(x)$$

$$\Delta x = -\alpha \times f'(x) \quad \Rightarrow \quad x \leftarrow x - \alpha \times f'(x)$$

학습률

경사하강법의 주의점

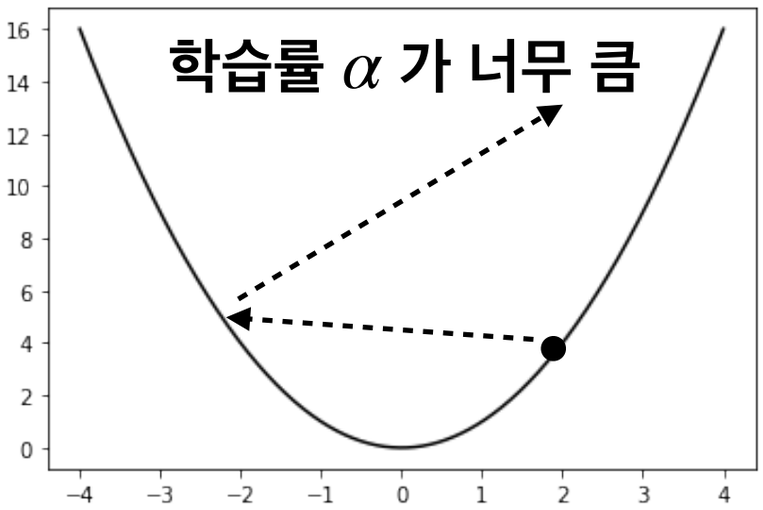
$$x_{n+1} \leftarrow x_n - \alpha \times f'(x_n)$$



$\alpha \gg 1$ $\alpha \ll 1$

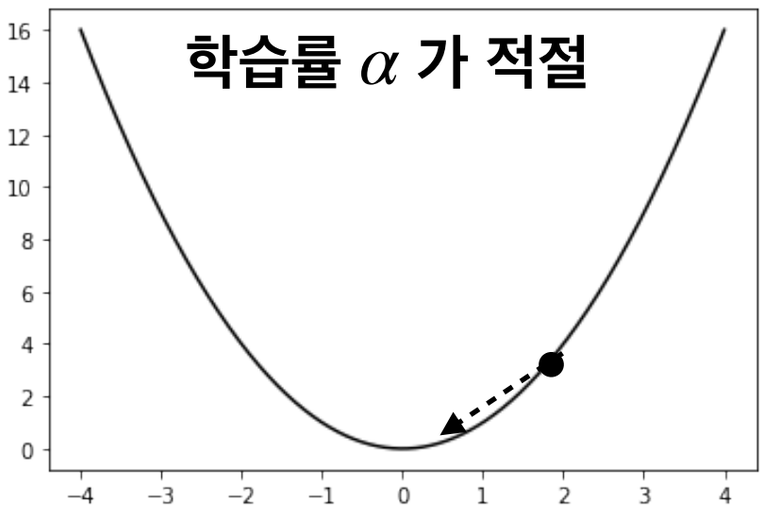
- 학습률이 너무 큰 경우, x_n 이 발산할 가능성이 큼
- 학습률이 너무 작은 경우 극솟값에 다다르는 속도가 너무 느리게 됨 (비효율성)
- **극소값에 가까워 질 수록** $f'(x) \ll 1$ 이 되어, **비효율적이 됨**

$$x_{n+1} \leftarrow x_n - \alpha \times f'(x_n) \simeq x_n$$



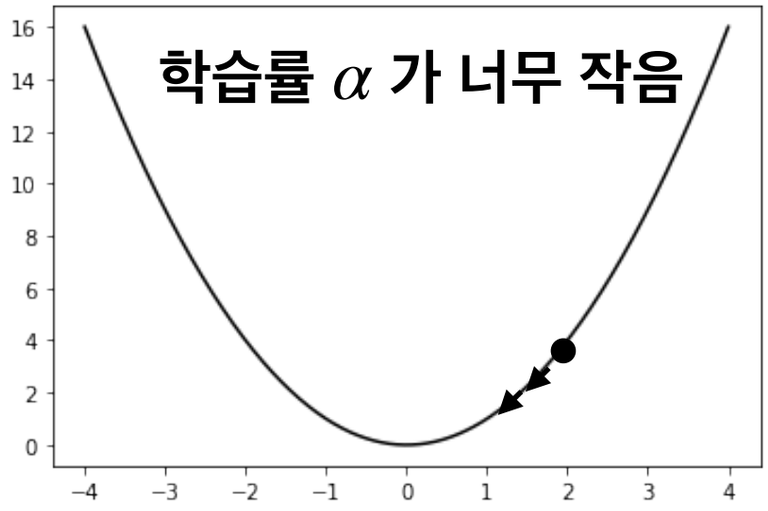

```
xp = 2
print("x_%i = %s" %(0, xp))
alpha=1.5
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

```
x_0 = 2
x_1 = -4.0
x_2 = 8.0
x_3 = -16.0
x_4 = 32.0
x_5 = -64.0
x_6 = 128.0
x_7 = -256.0
x_8 = 512.0
x_9 = -1024.0
x_10 = 2048.0
x_11 = -4096.0
x_12 = 8192.0
x_13 = -16384.0
x_14 = 32768.0
x_15 = -65536.0
x_16 = 131072.0
x_17 = -262144.0
x_18 = 524288.0
x_19 = -1048576.0
x_20 = 2097152.0
```




```
xp = 2
print("x_%i = %s" %(0, xp))
alpha=0.4
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

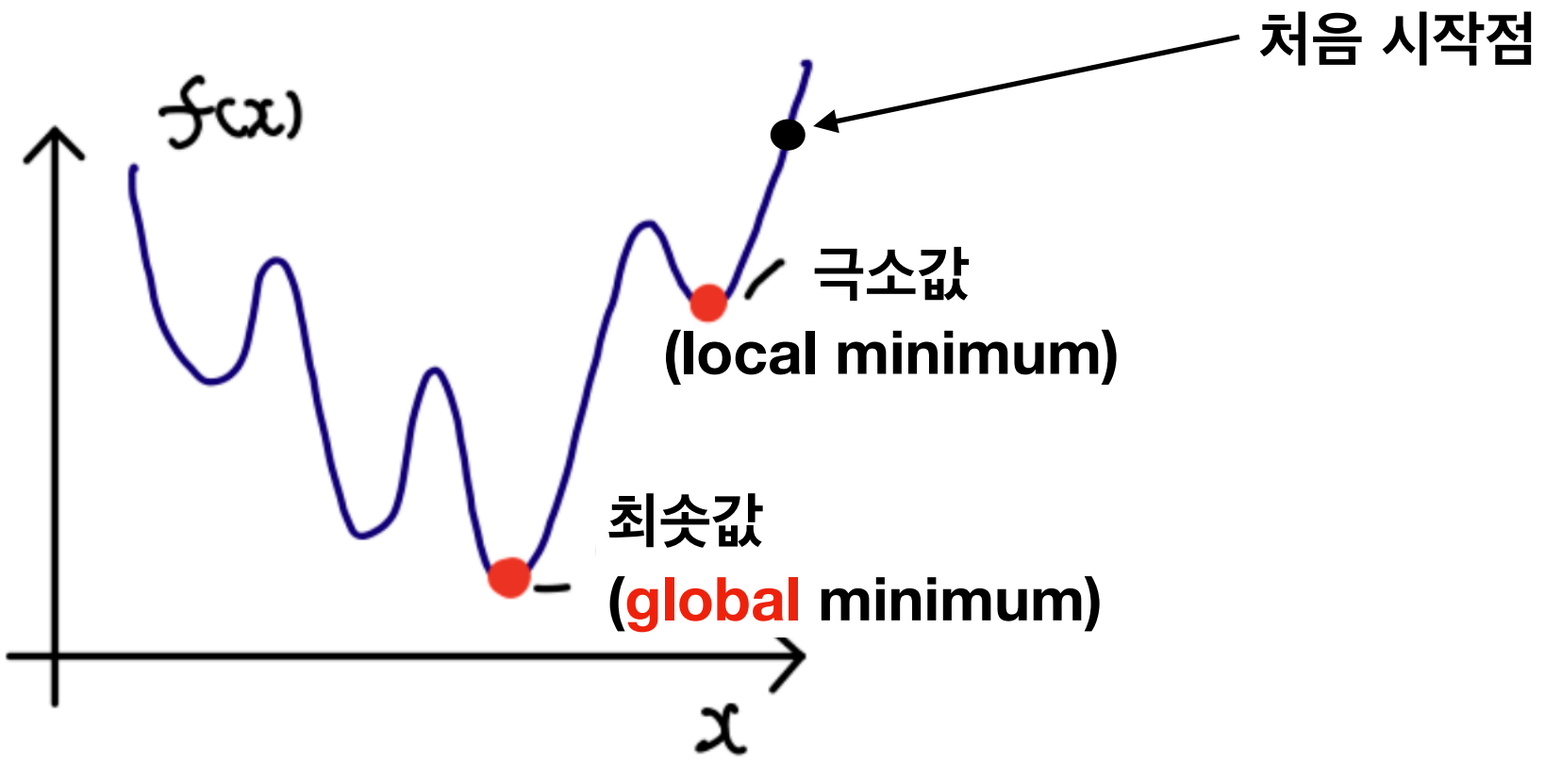
```
x_0 = 2
x_1 = 0.3999999999999999
x_2 = 0.07999999999999996
x_3 = 0.015999999999999986
x_4 = 0.0031999999999999963
x_5 = 0.0006399999999999991
x_6 = 0.00012799999999999975
x_7 = 2.5599999999999945e-05
x_8 = 5.119999999999988e-06
x_9 = 1.0239999999999973e-06
x_10 = 2.047999999999994e-07
x_11 = 4.095999999999986e-08
x_12 = 8.191999999999972e-09
x_13 = 1.6383999999999938e-09
x_14 = 3.276799999999986e-10
x_15 = 6.553599999999972e-11
x_16 = 1.3107199999999942e-11
x_17 = 2.6214399999999877e-12
x_18 = 5.242879999999976e-13
x_19 = 1.0485759999999949e-13
x_20 = 2.0971519999999888e-14
```




```
xp = 2
print("x_%i = %s" %(0, xp))
alpha=0.001
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

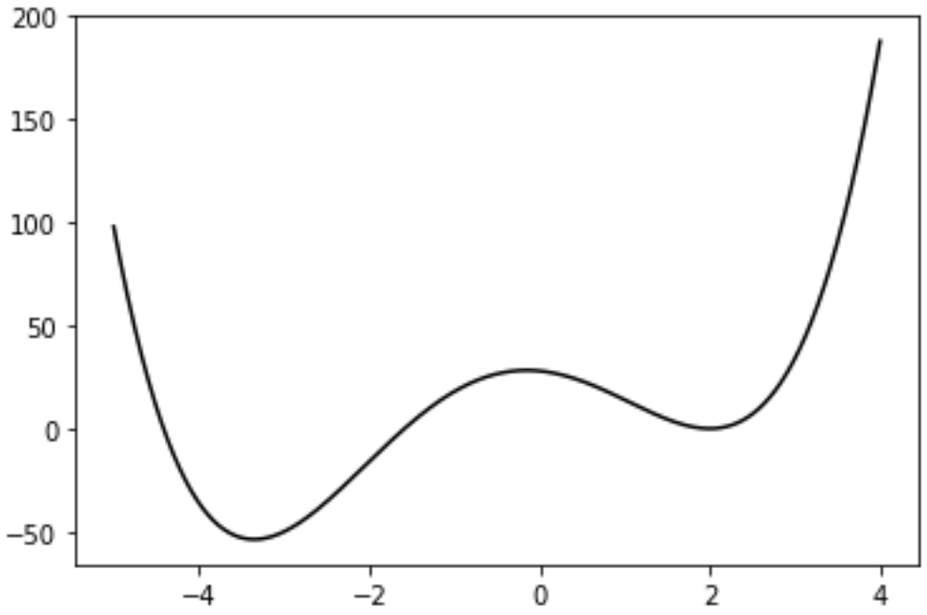
```
x_0 = 2
x_1 = 1.996
x_2 = 1.992008
x_3 = 1.988023984
x_4 = 1.984047936032
x_5 = 1.980079840159936
x_6 = 1.9761196804796162
x_7 = 1.972167441118657
x_8 = 1.9682231062364197
x_9 = 1.9642866600239468
x_10 = 1.960358086703899
x_11 = 1.9564373705304912
x_12 = 1.95252449578943
x_13 = 1.9486194467978513
x_14 = 1.9447222079042557
x_15 = 1.9408327634884472
x_16 = 1.9369510979614704
x_17 = 1.9330771957655475
x_18 = 1.9292110413740164
x_19 = 1.9253526192912684
x_20 = 1.9215019140526859
```

경사하강법의 취약점



- 처음 시작점 (보통 딥러닝에서 처음 시작점은 무작위로 주어짐) 이 최솟값보다 극소값에서 가까운 경우, 경사하강법으로 극소값에 도달하면, $f'(x) = 0$ 이 되어, 더이상의 업데이트가 안됨

$$x \leftarrow x - \alpha \times f'(x) \quad \Rightarrow \quad x \leftarrow x$$



$$f(x) = (x-2)^2(x+3)^2 - 2(x-2)^2$$

$$\frac{df(x)}{dx} = 0 \rightarrow x \in \{-3.35, -0.15, 2\}$$

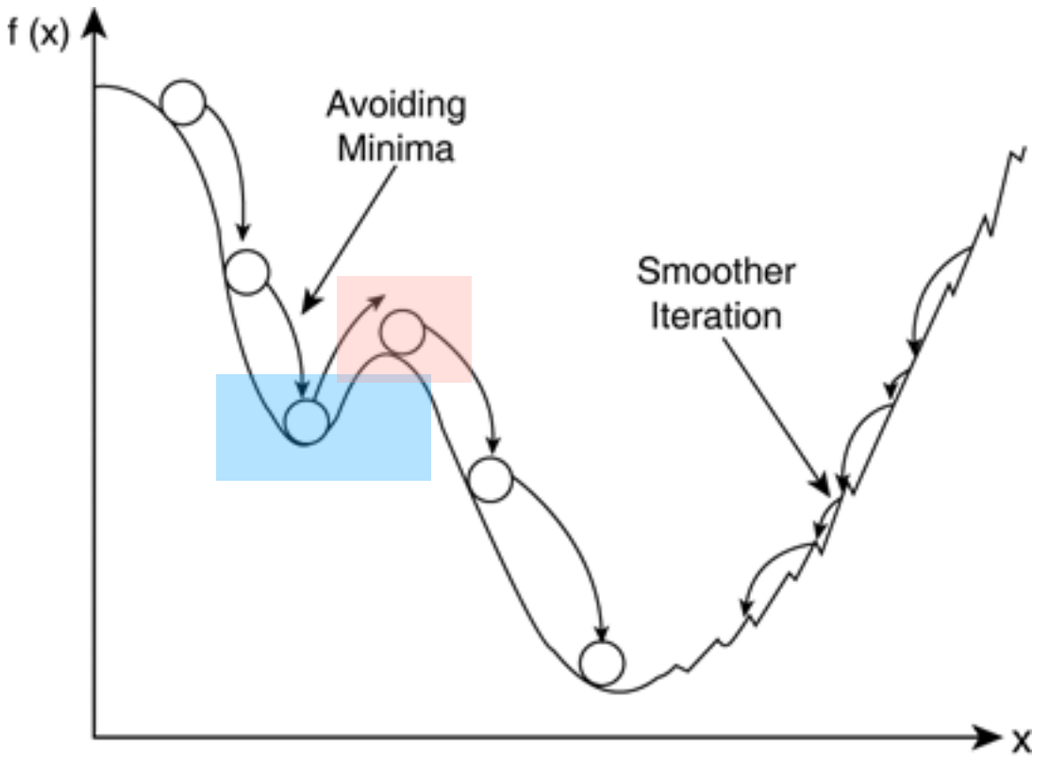
```
xp = 3
print("x_%i = %s" %(0, xp))
alpha=0.02
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

```
x_0 = 3
x_1 = 1.4
x_2 = 1.75328
x_3 = 1.9449413876409958
x_4 = 1.9937897931322763
x_5 = 1.9994800626095377
x_6 = 1.9999582428190736
x_7 = 1.9999966583793345
x_8 = 1.999999732663647
x_9 = 1.999999978613049
x_10 = 1.9999999982890437
x_11 = 1.9999999998631235
x_12 = 1.9999999999890499
x_13 = 1.999999999999124
x_14 = 1.9999999999999298
x_15 = 1.9999999999999944
x_16 = 1.9999999999999996
x_17 = 2.0
x_18 = 2.0
x_19 = 2.0
x_20 = 2.0
```

```
xp = 3
print("x_%i = %s" %(0, xp))
alpha=0.045
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

```
x_0 = 3
x_1 = -0.5999999999999996
x_2 = -1.1803199999999991
x_3 = -2.46145989967282
x_4 = -4.112817823095517
x_5 = -0.7894410574482342
x_6 = -1.6127969523421908
x_7 = -3.266957794648962
x_8 = -3.514720908301343
x_9 = -2.9670364427439253
x_10 = -3.9338105487727217
x_11 = -1.5770485588571086
x_12 = -3.2077026754479245
x_13 = -3.617905727574734
x_14 = -2.6809349763872192
x_15 = -4.1098129051771135
x_16 = -0.8036767469550297
x_17 = -1.6449340128907677
x_18 = -3.318917360325944
x_19 = -3.4156136661143823
x_20 = -3.209179142181531
x_21 = -3.6154603262064997
x_22 = -2.688125447623336
x_23 = -4.107857274317206
x_24 = -0.8129225651438969
x_25 = -1.665771034739621
x_26 = -3.3519266859785644
x_27 = -3.3483920632374096
x_28 = -3.355749879317858
x_29 = -3.3403902233884777
x_30 = -3.3722649230603525
x_31 = -3.3053059451120395
x_32 = -3.4423643599518403
x_33 = -3.14691209820452
x_34 = -3.71309603367323
x_35 = -2.385233762991075
x_36 = -4.089405994816705
x_37 = -0.8994239370999506
x_38 = -1.8592018752050916
x_39 = -3.630974928899266
x_40 = -2.642164452730028
```

모멘텀 (momentum) 방법



- 경사하강법의 경우 $f'(x) \simeq 0 \rightarrow \Delta x = -\alpha f'(x) \simeq 0$

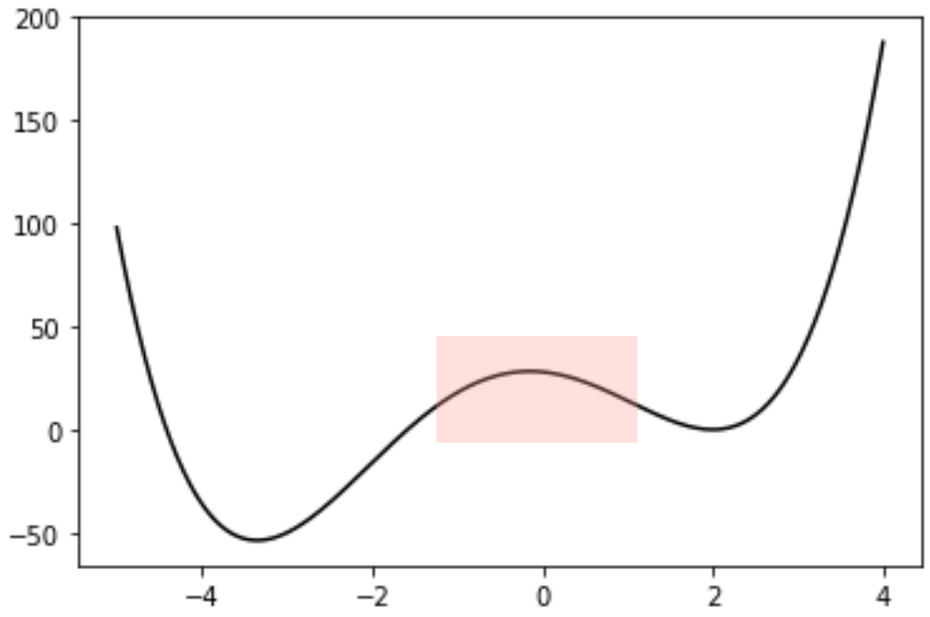
- 전단계의 변화율 (속도) 을 반영 $v_n \equiv x_{n+1} - x_n$

$$v_n = \beta v_{n-1} - \alpha f'(x_n)$$

$$x_{n+1} - x_n = v_n = \beta(x_n - x_{n-1}) - \alpha f'(x_n)$$

: 대략 $\beta = 0.9$ 정도로 두는 것이 일반적.

- 참고로 경사하강법: $v_n = x_{n+1} - x_n = -\alpha f'(x_n)$: $x_{n+1} = x_n - \alpha f'(x_n)$



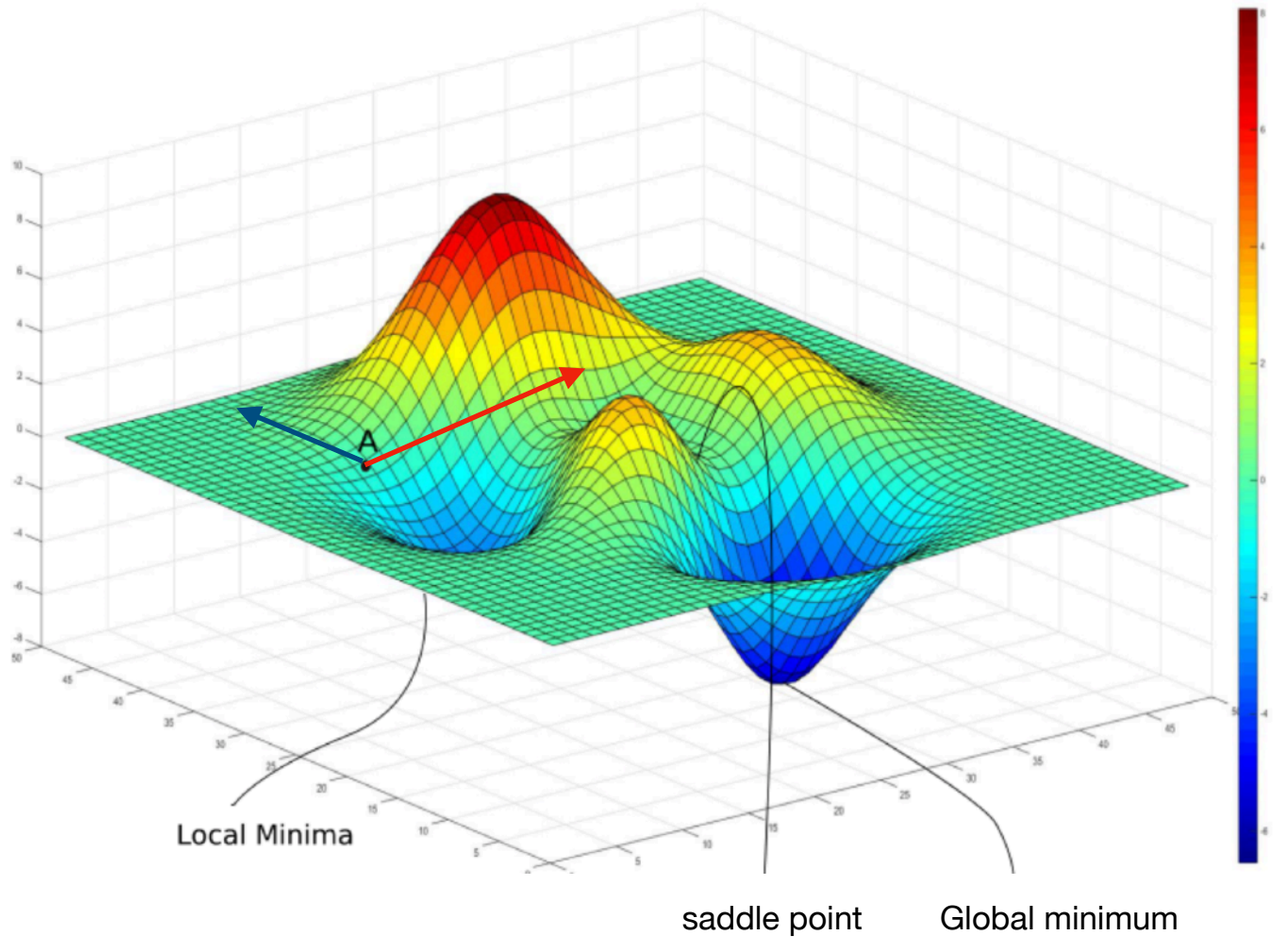

```python
xp = 3
print("x_%i = %s" %(0, xp))
alpha=0.02
for i in range(20):
    xp= xp - alpha*diff_f(xp)
    print("x_%i = %s" %(i+1, xp))
```

```
x_0 = 3
x_1 = 1.4
x_2 = 1.75328
x_3 = 1.9449413876409958
x_4 = 1.9937897931322763
x_5 = 1.9994800626095377
x_6 = 1.9999582428190736
x_7 = 1.9999966583793345
x_8 = 1.9999997326636 47
x_9 = 1.9999999786130 49
x_10 = 1.9999999982890437
x_11 = 1.9999999998631235
x_12 = 1.9999999999890499
x_13 = 1.9999999999999124
x_14 = 1.9999999999999298
x_15 = 1.9999999999999944
x_16 = 1.9999999999999996
x_17 = 2.0
x_18 = 2.0
x_19 = 2.0
x_20 = 2.0
```

```python
xp = 3
print("x_%i = %s" %(0, xp))
alpha=0.02
v=0
beta=0.9
for i in range(20):
    v = beta*v - alpha*diff_f(xp)
    xp = xp + v
    print("x_%i = %s" %(i+1, xp))
```

```
x_0 = 3
x_1 = 1.4
x_2 = 0.3132799999999998
x_3 = -0.4360994561759646
x_4 = -1.2734995252188481
x_5 = -2.638766436929365
x_6 = -4.525318703609472
x_7 = -3.5400600761547603
x_8 = -2.368871785031992
x_9 = -2.076558008372518
x_10 = -2.6143920260944684
x_11 = -3.76857286454931
x_12 = -4.109506094413966
x_13 = -2.9477336351869767
x_14 = -2.3485958361632866
x_15 = -2.5761799069894527
x_16 = -3.4692362164620314
x_17 = -4.100914586718611
x_18 = -3.2226301269464575
x_19 = -2.596733134048547
x_20 = -2.712103333004605
x_21 = -3.4329797246738853
x_22 = -3.964450558622487
x_23 = -3.3256130241749418
x_24 = -2.7847191365302355
x_25 = -2.868963711907946
x_26 = -3.455214906293401
x_27 = -3.8321645080250972
x_28 = -3.3442273759695094
x_29 = -2.914036210795086
x_30 = -3.0015678046053957
x_31 = -3.478902383682647
x_32 = -3.721518011115125
x_33 = -3.3336734580165635
x_34 = -3.0078589506893336
x_35 = -3.1073585627656213
x_36 = -3.4911238273007283
x_37 = -3.6304802479548526
x_38 = -3.3173064512078434
x_39 = -3.0805622696561947
x_40 = -3.1894392208702786
```

이변수 함수의 미분



- 함수 $z = f(\vec{X}) = f(x, y)$ 에 대해 특정 방향 $\vec{u} = (a, b)$ 에 대한 점 $\vec{X}_0 = (x_0, y_0)$ 에서 함수의 방향도 함수

$$D_{\vec{u}} f(\vec{X}_0) = \lim_{h \to 0} \frac{f(\vec{X}_0 + h\vec{u}) - f(\vec{X}_0)}{h} = \lim_{h \to 0} \frac{f(x_0 + ha, y_0 + hb) - f(x_0, y_0)}{h}$$

- 함수 $z = f(\vec{X}) = f(x, y)$ 에 대해 특정 방향 $\vec{u} = (a, b)$ 에 대한 점 $\vec{X}_0 = (x_0, y_0)$ 에서 함수의 방향도 함수

$$D_{\vec{u}} f(\vec{X}_0) = \lim_{h \to 0} \frac{f(\vec{X}_0 + h\vec{u}) - f(\vec{X}_0)}{h} = \lim_{h \to 0} \frac{f(x_0 + ha, y_0 + hb) - f(x_0, y_0)}{h}$$

- $\vec{u} = (1,0) = \hat{i}$ 이면 $D_{(1,0)} f(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0 + h, y_0) - f(x_0, y_0)}{h} \equiv \left. \frac{\partial f(x, y)}{\partial x} \right|_{(x_0, y_0)}$ (x -편미분)

- $\vec{u} = (0,1) = \hat{j}$ 이면 $D_{(0,1)} f(x_0, y_0) = \lim_{h \to 0} \frac{f(x_0, y_0 + h) - f(x_0, y_0)}{h} \equiv \left. \frac{\partial f(x, y)}{\partial y} \right|_{(x_0, y_0)}$ (y- 편미분)

- 기울기 벡터 (gradient) $\nabla f(x, y) = (f_x, f_y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$

- 일변수 함수 $y = f(x)$ 에서, $dy = f'(x) \cdot dx = \dfrac{df}{dx} \cdot dx \ \rightarrow \Delta y \simeq f'(x) \cdot \Delta x$

- 이변수 함수 $z = f(x, y)$ 에서 , $dz = \dfrac{\partial f}{\partial x}dx + \dfrac{\partial f}{\partial y}dy = \nabla f \cdot (dx, dy)$

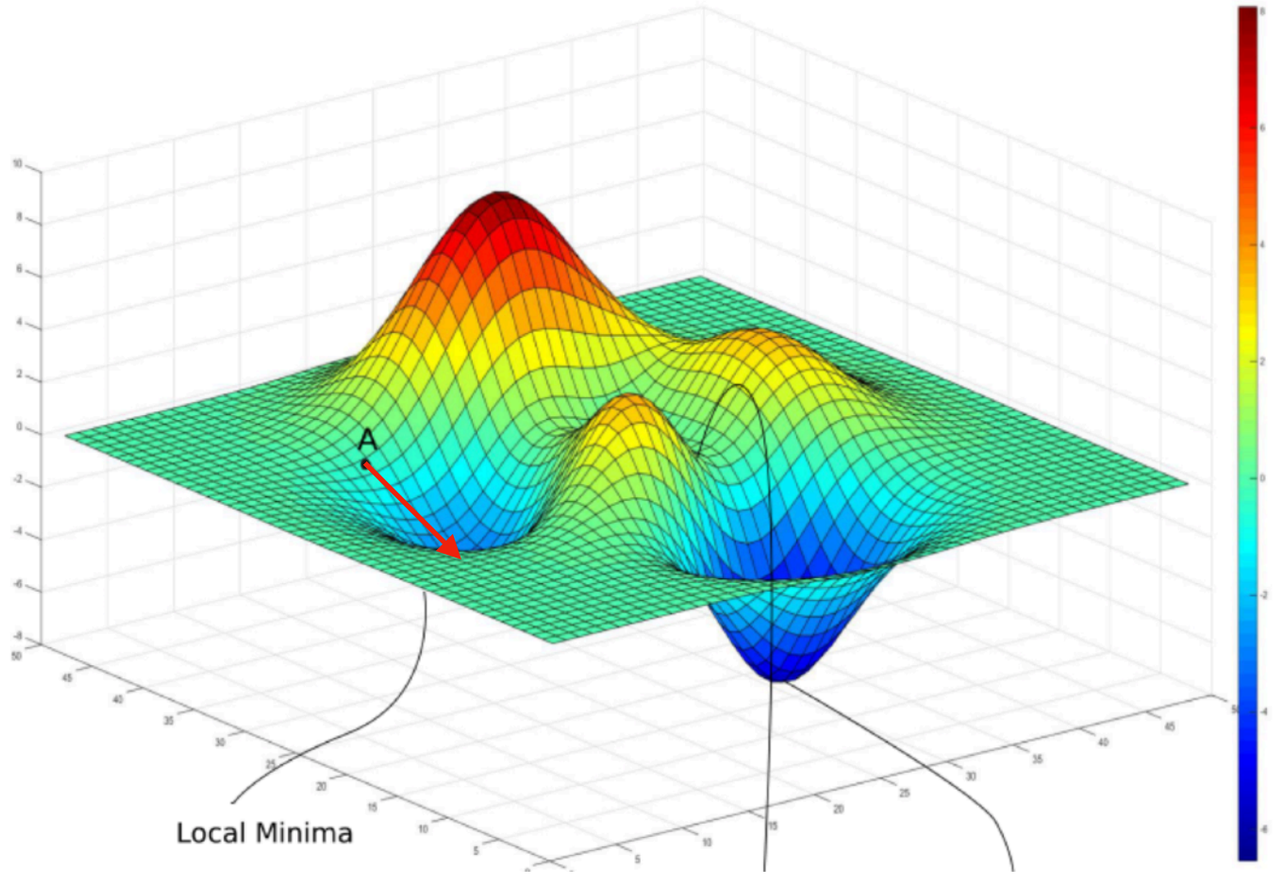
- 이변수 함수 $z = f(x, y)$ 에서 , $dz = \dfrac{\partial f}{\partial x}dx + \dfrac{\partial f}{\partial y}dy = \nabla f \cdot (dx, dy)$

$$\Delta z \simeq \nabla f \cdot (\Delta x, \Delta y)$$

- $\vec{u} = (a, b)$ 방향으로 움직일 때, 즉 $\Delta x = a$ 이고, $\Delta y = b$ 일 때,
  함수 $f(x, y)$ 의 증가분 $= \nabla f \cdot (a, b) = \nabla f \cdot \vec{u}$

- 내적의 성질로 부터
  1. Δz 가 빨리 증가하는 방향 $\vec{u} \propto \nabla f$
  2. Δz 가 빨리 감소하는 방향 $\vec{u} \propto -\nabla f$

경사 하강법



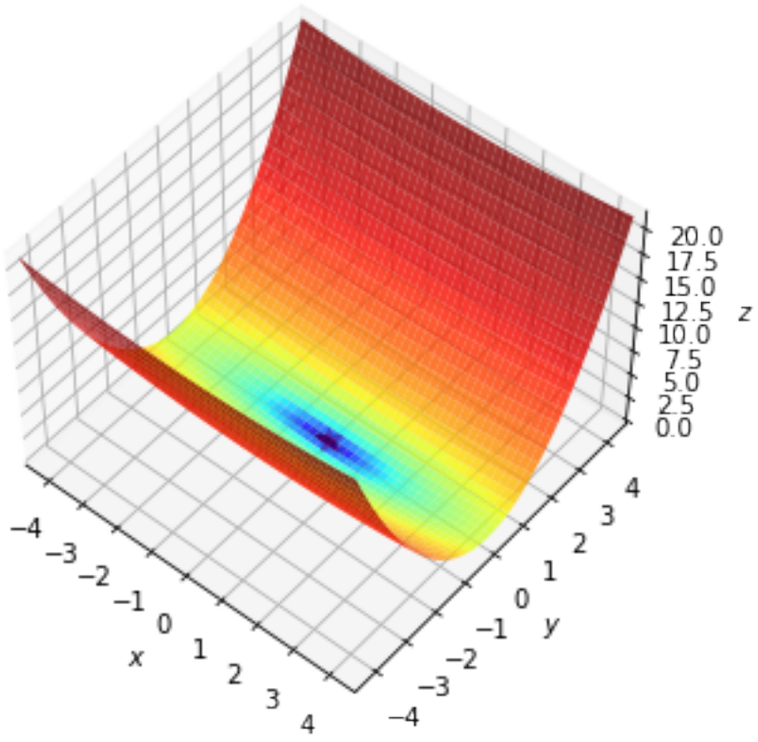
$$d\vec{X} = -\alpha \nabla f$$

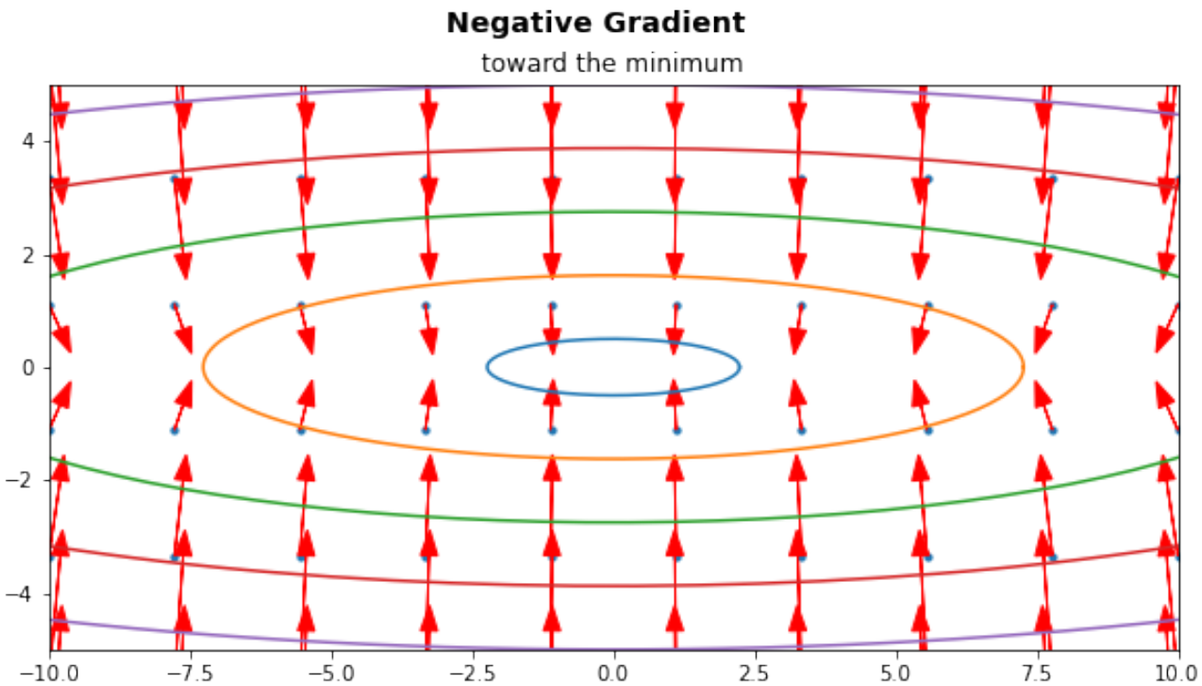
$$(x, y) \leftarrow (x, y) - \alpha \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right)$$

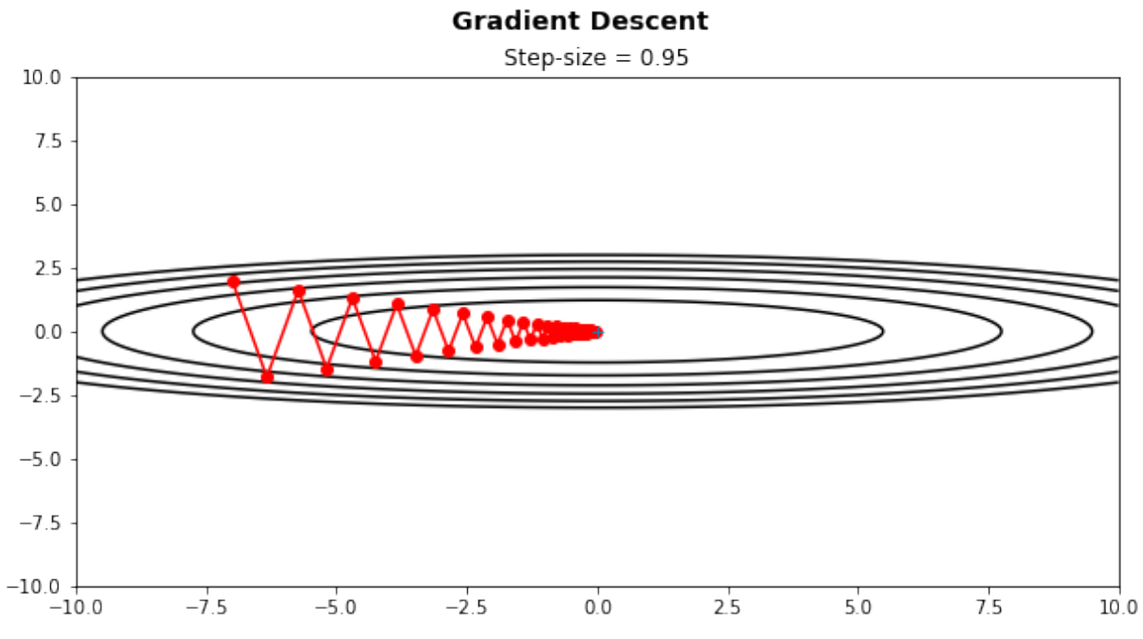
$$\vec{X}_{n+1} = \vec{X}_n - \alpha \nabla f(\vec{X}_n)$$

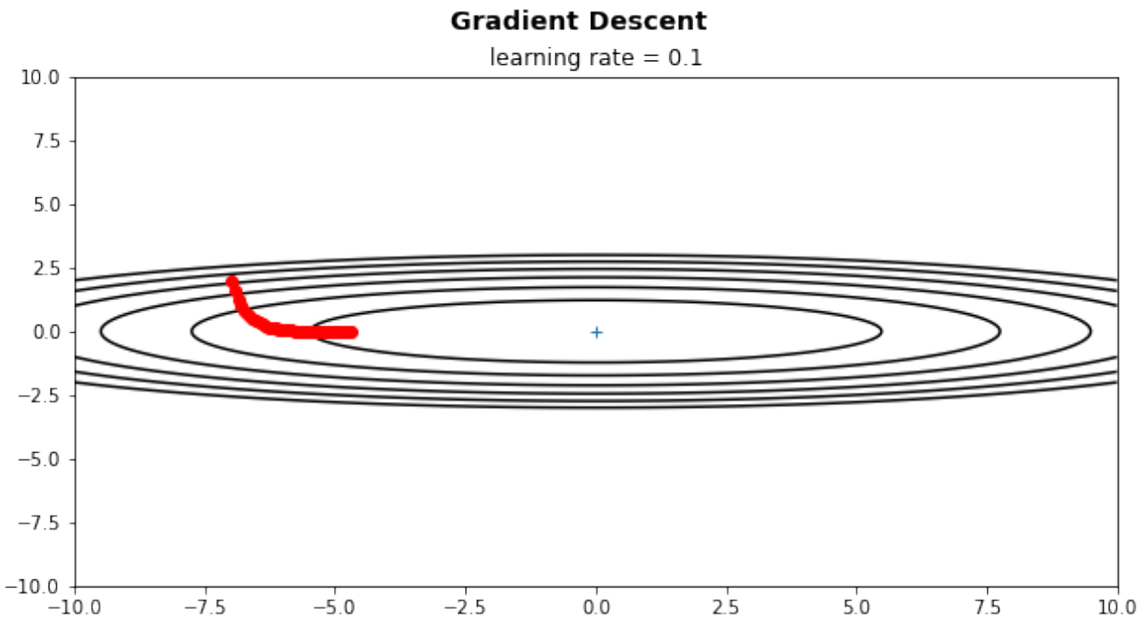
$$f(x,y) = \frac{1}{20}x^2 + y^2$$

$$-\nabla f(x,y) = -\left(\frac{1}{10}x, 2y\right)$$









iteration: 40회

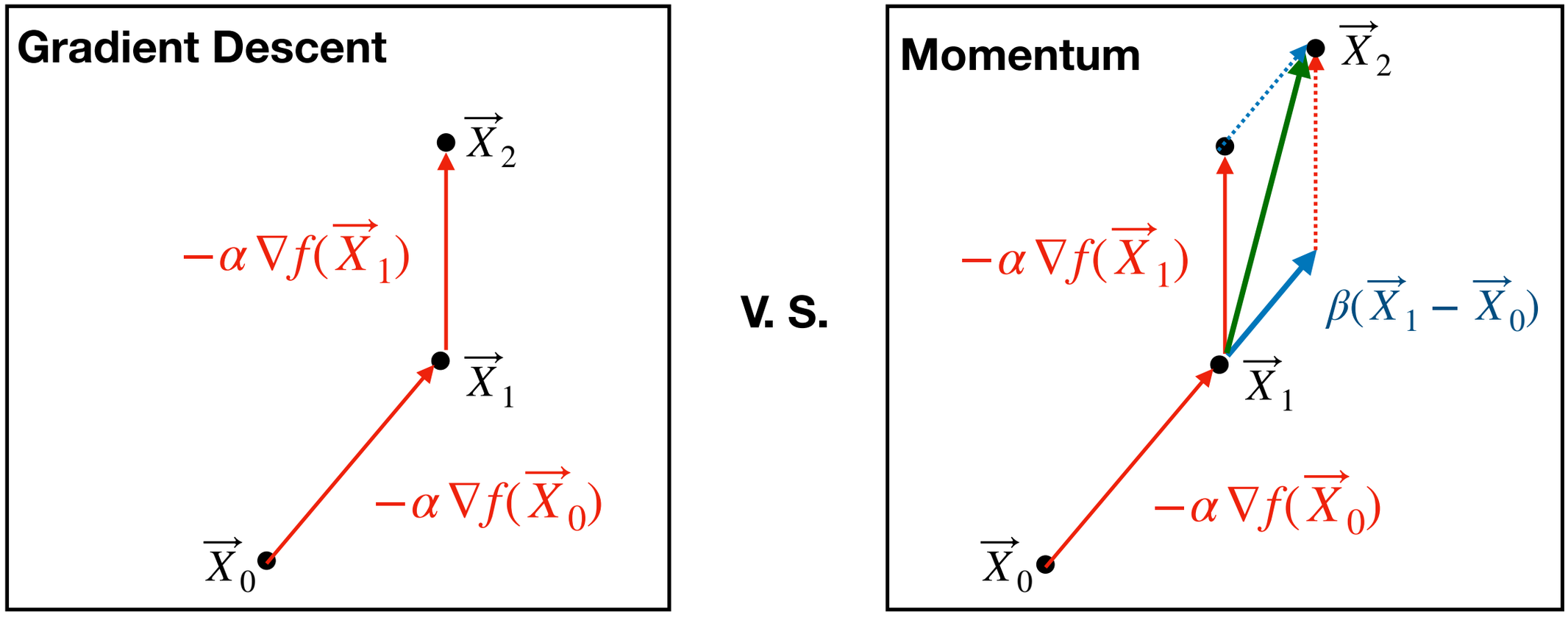
모멘텀 (momentum) 방법

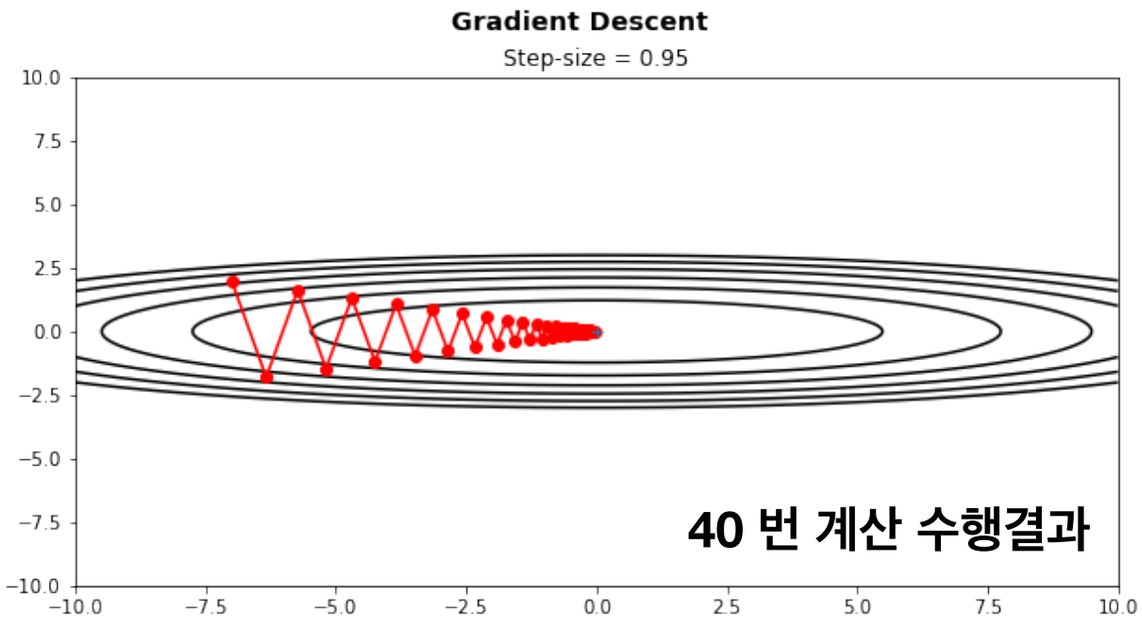
- 일변수 함수의 경우, $v_n = \beta v_{n-1} - \alpha f'(x_n)$

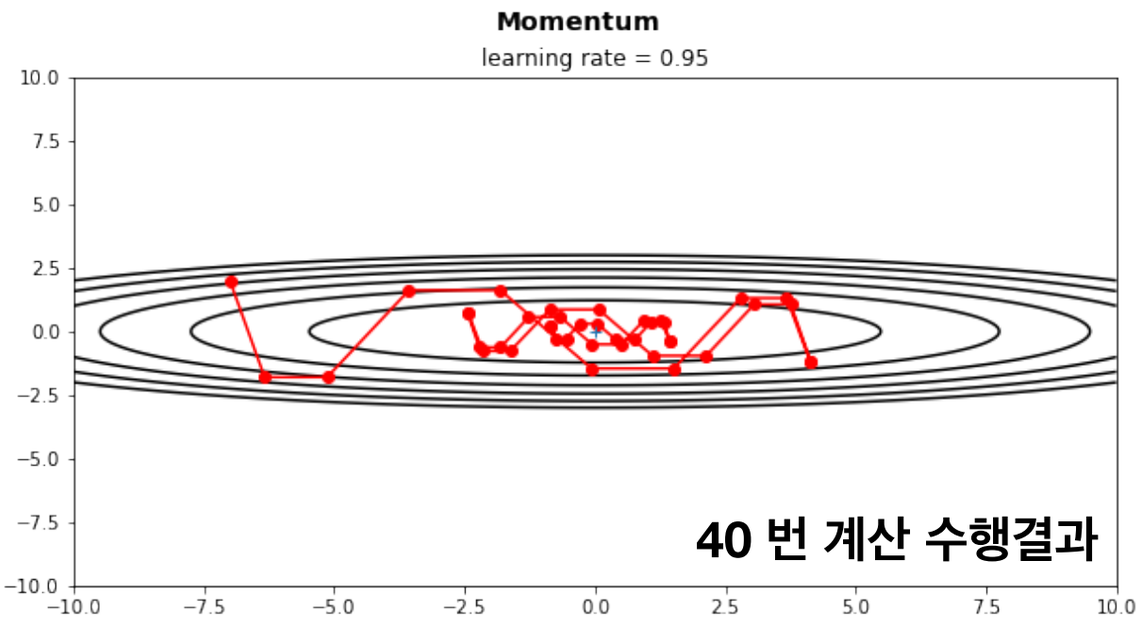
$$x_{n+1} - x_n = v_n = \beta(x_n - x_{n-1}) - \alpha f'(x_n)$$

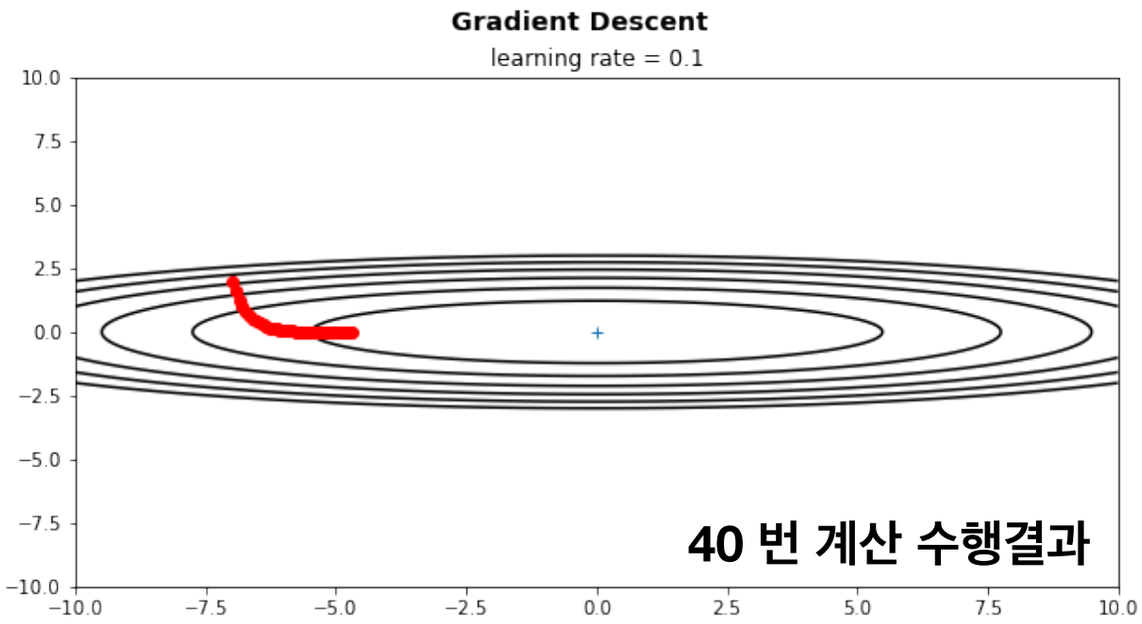
- 이변수 함수의 경우, $\vec{v}_n = \beta \vec{v}_{n-1} - \alpha \nabla f(\vec{X}_n)$

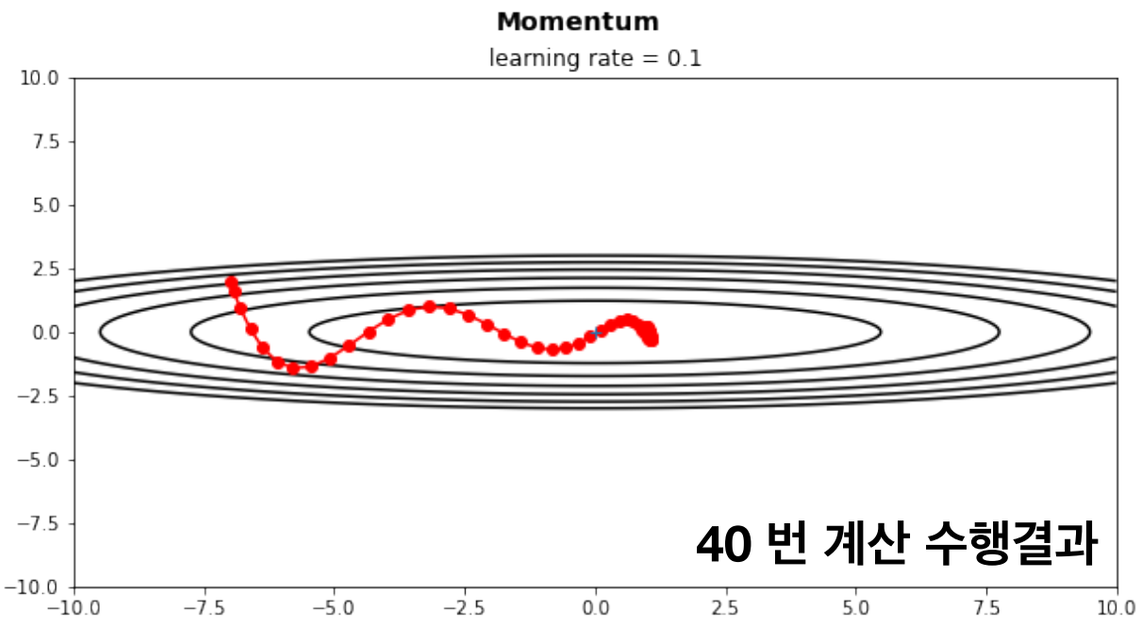
$$\vec{X}_{n+1} - \vec{X}_n = \vec{v}_n = \beta(\vec{X}_n - \vec{X}_{n-1}) - \alpha \nabla f(\vec{X}_n)$$




Gradient Descent
Step-size = 0.95
40 번 계산 수행결과


Momentum
learning rate = 0.95
40 번 계산 수행결과


Gradient Descent
learning rate = 0.1
40 번 계산 수행결과


Momentum
learning rate = 0.1
40 번 계산 수행결과

Adaptive Method: AdaGrad

- 일반적인 Gradient Method 의 경우에는, 모든 파라메터에 대해 동일한 step-size (학습률) 이 적용됨.

$$\vec{X}_n \leftarrow \vec{X}_{n-1} - \alpha \nabla f(\vec{X}_{n-1}) \quad \Rightarrow \quad \begin{aligned} x_n &\leftarrow x_{n-1} - \alpha f_x(x_{n-1}, y_{n-1}) \\ y_n &\leftarrow y_{n-1} - \alpha f_y(x_{n-1}, y_{n-1}) \end{aligned}$$

- "Adaptive" 방법은, 각각의 매개변수에 맞춰서 학습률 변화
  학습을 진행하면서, 학습률을 점차 줄여가는 방법 (학습률 감소 **learning rate decay**)

$$(g_n, h_n) \leftarrow (g_{n-1}, h_{n-1}) + \left(f_x(x_{n-1}, y_{n-1})^2, f_y(x_{n-1}, y_{n-1})^2 \right)$$

$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{g_n} + \epsilon} \cdot f_x(x_{n-1}, y_{n-1})$$

$$y_n \leftarrow y_{n-1} - \frac{\alpha}{\sqrt{h_n} + \epsilon} \cdot f_y(x_{n-1}, y_{n-1})$$

초기 값 $g_0 = h_0 = 0$

g **와** h **가 먼저 업데이트 되고**
(x, y) **를 업데이트 함**

- "Adaptive" 방법은, 각각의 매개변수에 맞춰서 학습률 변화
  학습을 진행하면서, 학습률을 점차 줄여가는 방법 (학습률 감소 **learning rate decay**)

$$(g_n, h_n) \leftarrow (g_{n-1}, h_{n-1}) + \left(f_x(x_{n-1}, y_{n-1})^2, f_y(x_{n-1}, y_{n-1})^2\right)$$

$$g_1 = g_0 + f_x(x_0, y_0)^2 = f_x(x_0, y_0)^2$$

$$g_2 = g_1 + f_x(x_1, y_1)^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2$$

$$g_3 = g_2 + f_x(x_2, y_2)^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2 + f_x(x_2, y_2)^2$$

$$\vdots$$

$$g_n = g_{n-1} + f_x(x_{n-1}, y_{n-1})^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2 + \cdots + f_x(x_{n-1}, y_{n-1})^2 = \sum_{i=0}^{n-1} f_x(x_i, y_i)^2$$

$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{g_n} + \epsilon} \cdot f_x(x_{n-1}, y_{n-1})$$

$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{\sum_{i=0}^{n-1} f_x(x_i, y_i)^2} + \epsilon} \cdot f_x(x_{n-1}, y_{n-1})$$

: n 이 커지면, 분모가 커짐.

:미분값이 작으면 학습률이 커짐

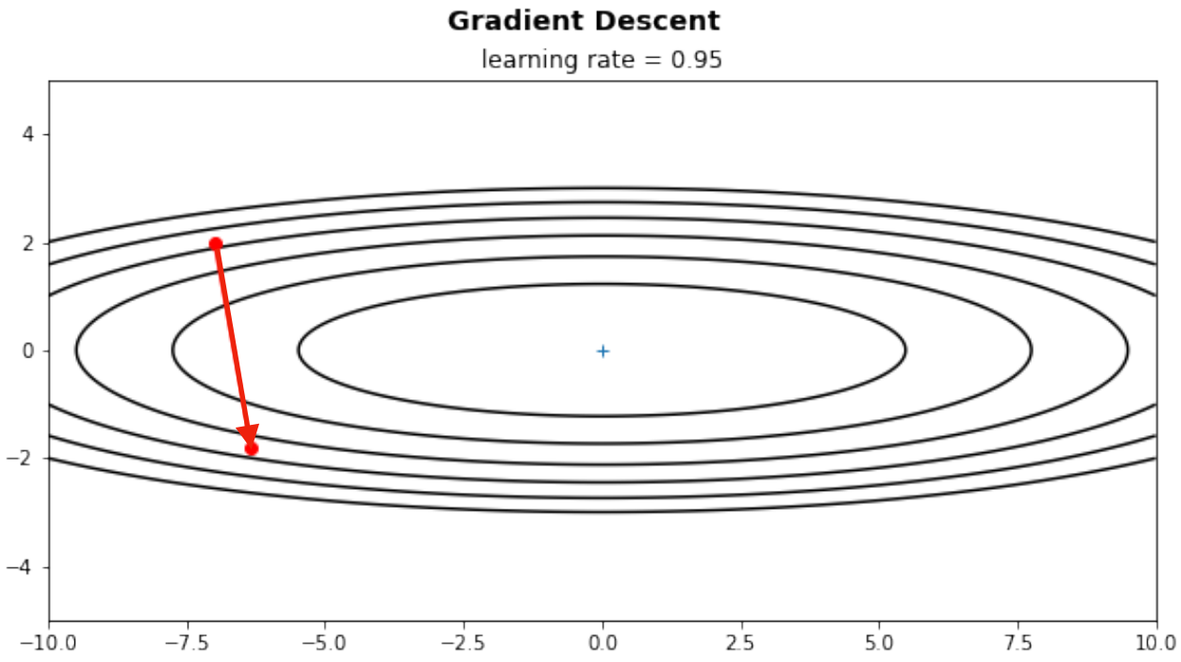
$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{\sum_{i=0}^{n-1} f_x(x_i, y_i)^2 + \epsilon}} \cdot f_x(x_{n-1}, y_{n-1})$$

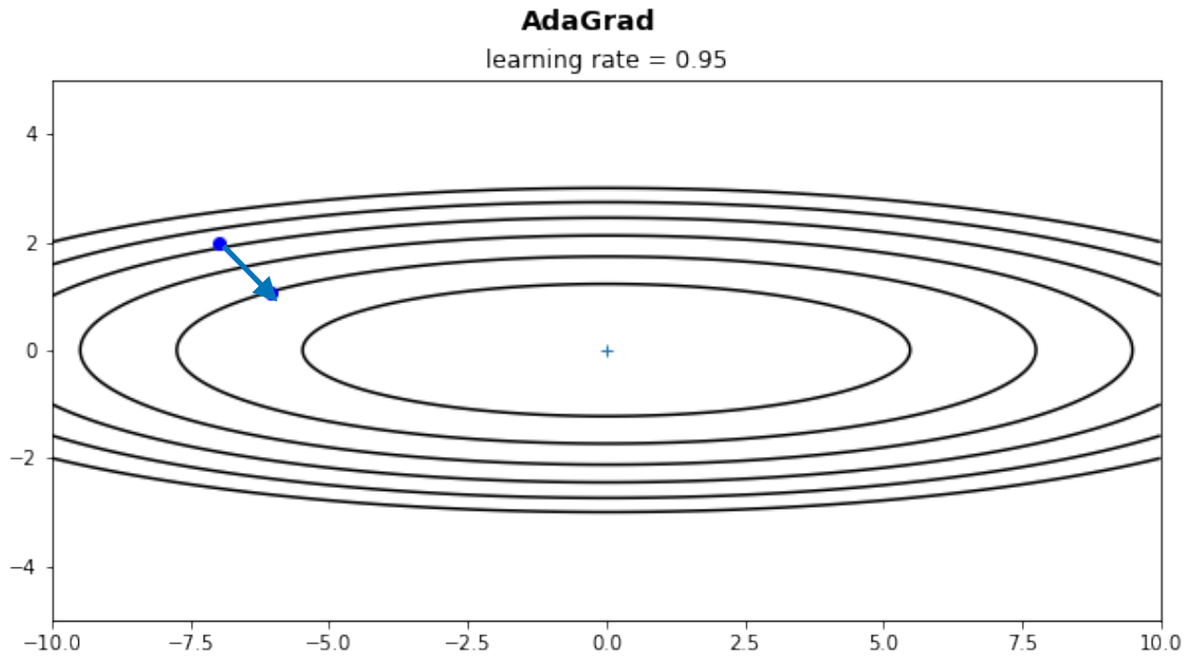
: n 이 커지면, 분모가 커짐.

:미분값이 작으면 학습률이 커짐

$$-\nabla f(x, y) = -\left(\frac{1}{10}x, 2y\right)$$

- **x 축 방향이 미분값이 작음 (변화가 적음)**
- **y 축 방향이 미분값이 큼 (급격히 변함)**





(0.95, -0.95)

(0.665, -3.80) **급격히 변하는 방향으로 쏠림**

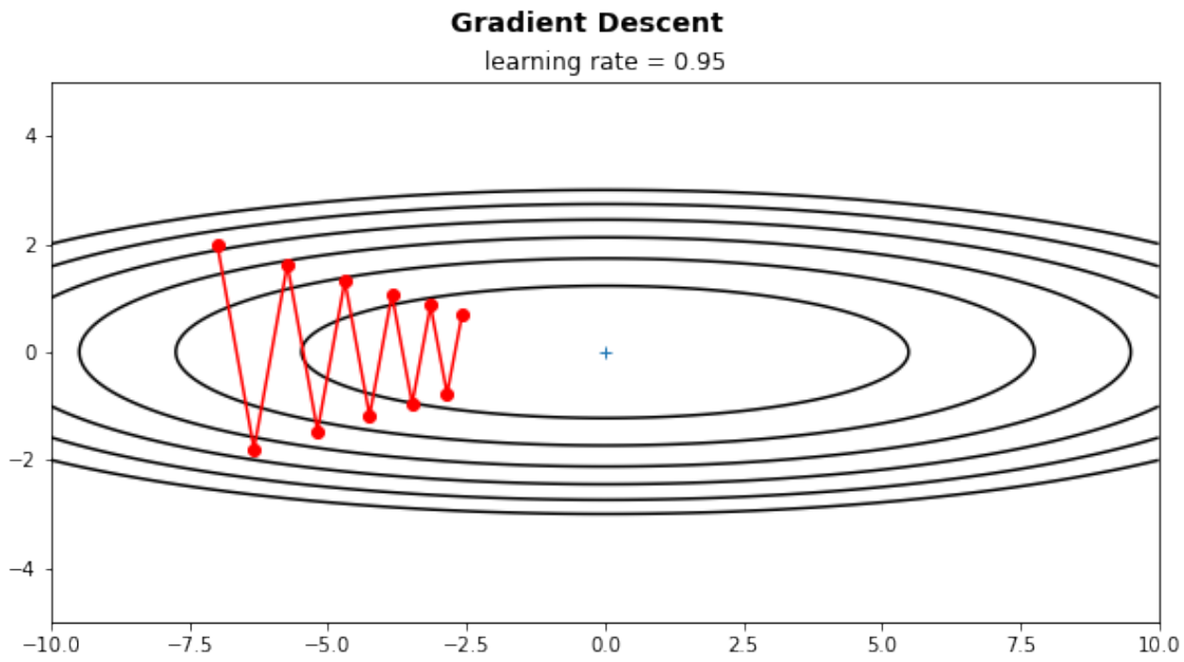
$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{\sum_{i=0}^{n-1} f_x(x_i, y_i)^2 + \epsilon}} \cdot f_x(x_{n-1}, y_{n-1})$$

: n 이 커지면, 분모가 커짐.

:미분값이 작으면 학습률이 커짐

$$-\nabla f(x, y) = -\left(\frac{1}{10}x, 2y\right)$$

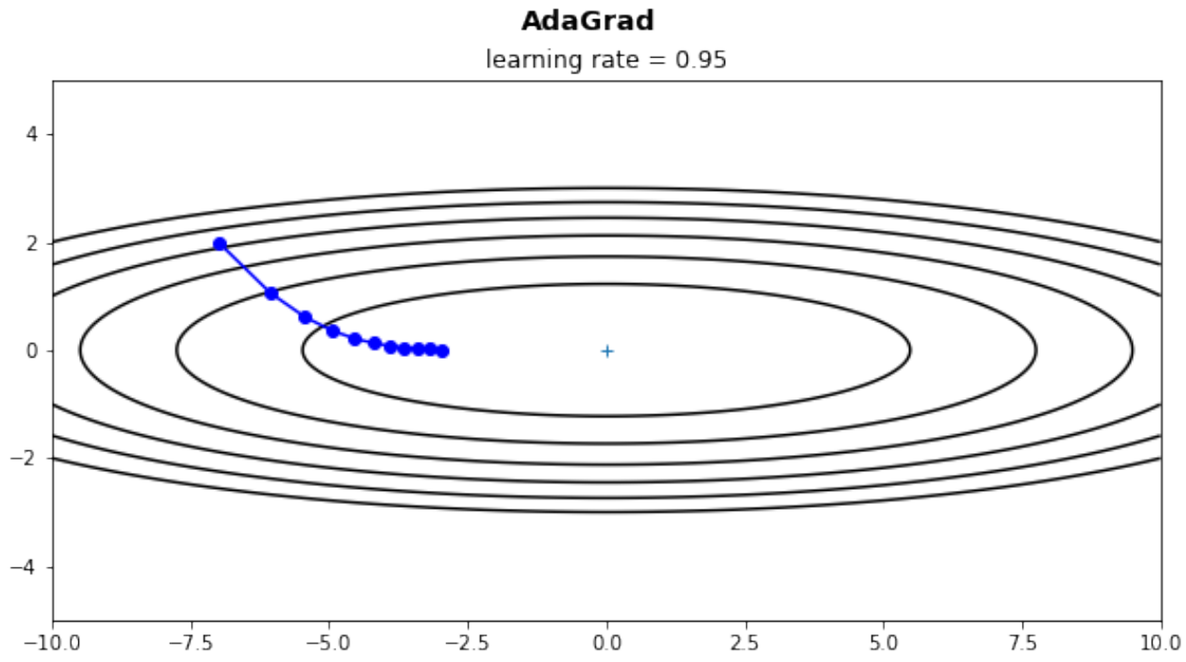
- **x 축 방향이 미분값이 작음 (변화가 적음)**
- **y 축 방향이 미분값이 큼 (급격히 변함)**



불안정하게 최솟값을 찾아감

(0.95, -0.95)

(0.665, -3.80) **급격히 변하는 방향으로 쏠림**



안정적으로 최솟값을 찾아감.

다만, 횟수가 증가할 수록 학습률이 작아짐

- Numpy array의 element-wise 연산을 통하여 코드를 간단화 시킬 수 있음.

$$(g_n, h_n) \leftarrow (g_{n-1}, h_{n-1}) + \left(f_x(x_{n-1}, y_{n-1})^2, f_y(x_{n-1}, y_{n-1})^2\right)$$

$$x_n \leftarrow x_{n-1} - \frac{\alpha}{\sqrt{g_n} + \epsilon} \cdot f_x(x_{n-1}, y_{n-1})$$

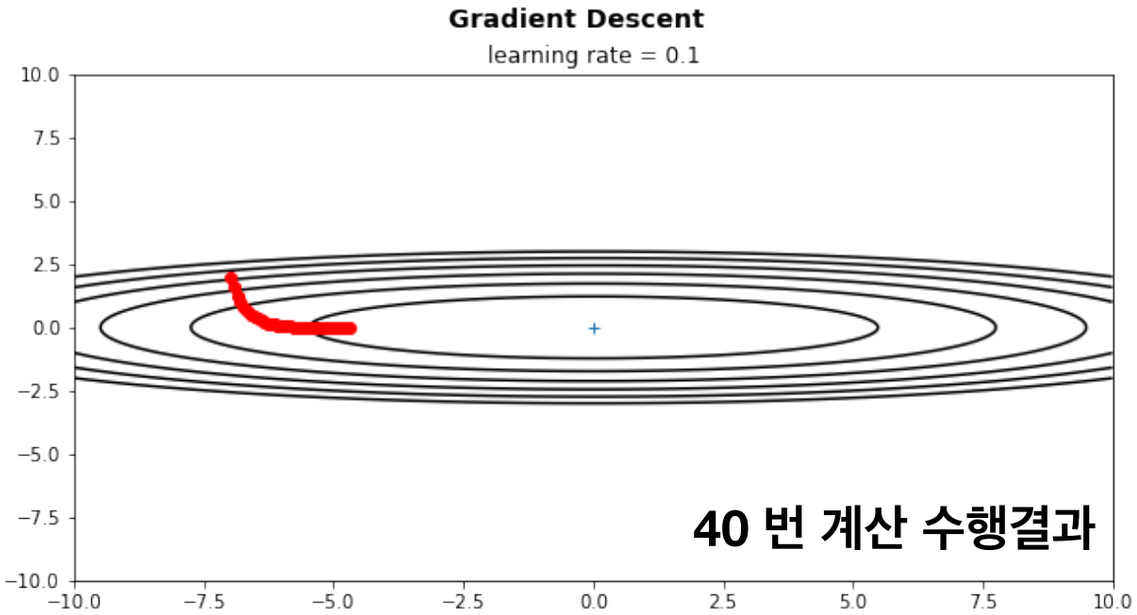
$$y_n \leftarrow y_{n-1} - \frac{\alpha}{\sqrt{h_n} + \epsilon} \cdot f_y(x_{n-1}, y_{n-1})$$

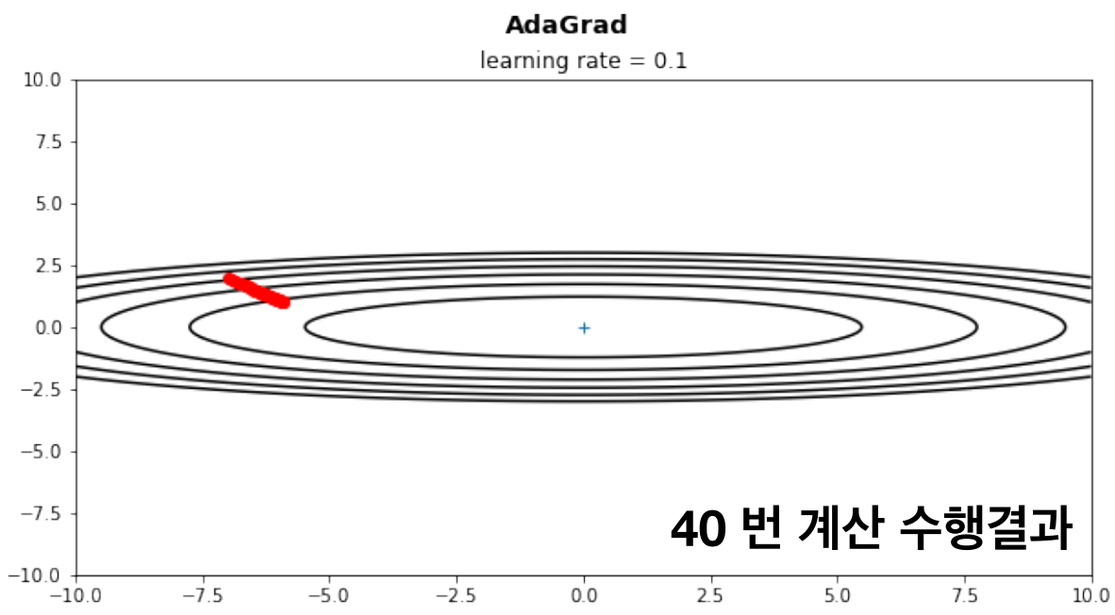
$$\vec{G}_n \leftarrow \vec{G}_{n-1} + \nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$$

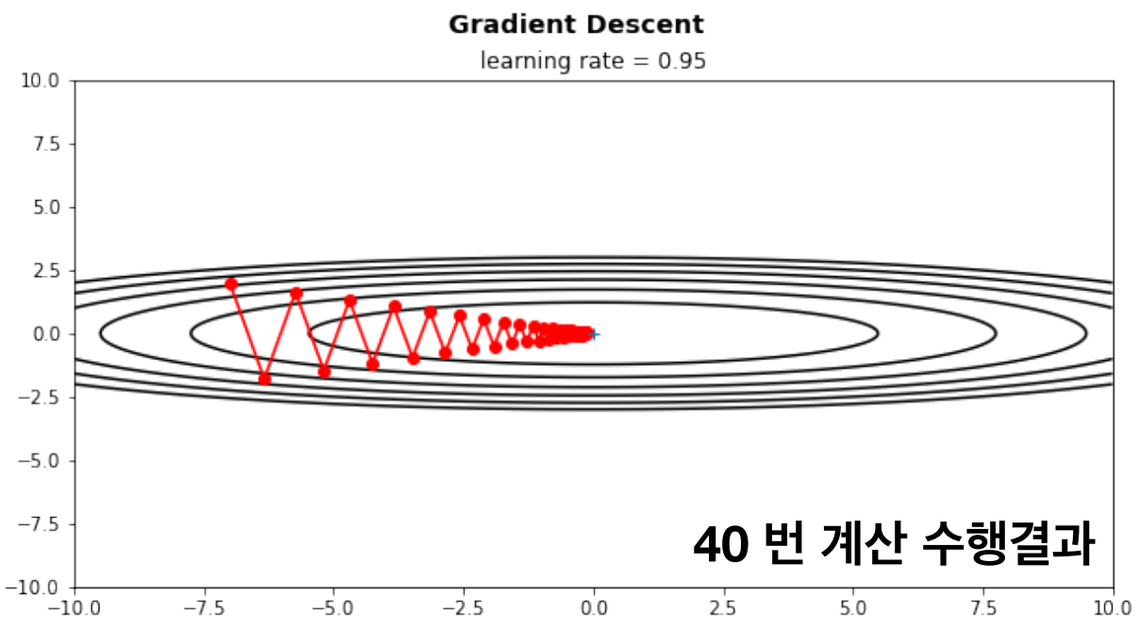
$$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\vec{G}_n} + \epsilon} \odot \nabla f(\vec{X}_{n-1})$$

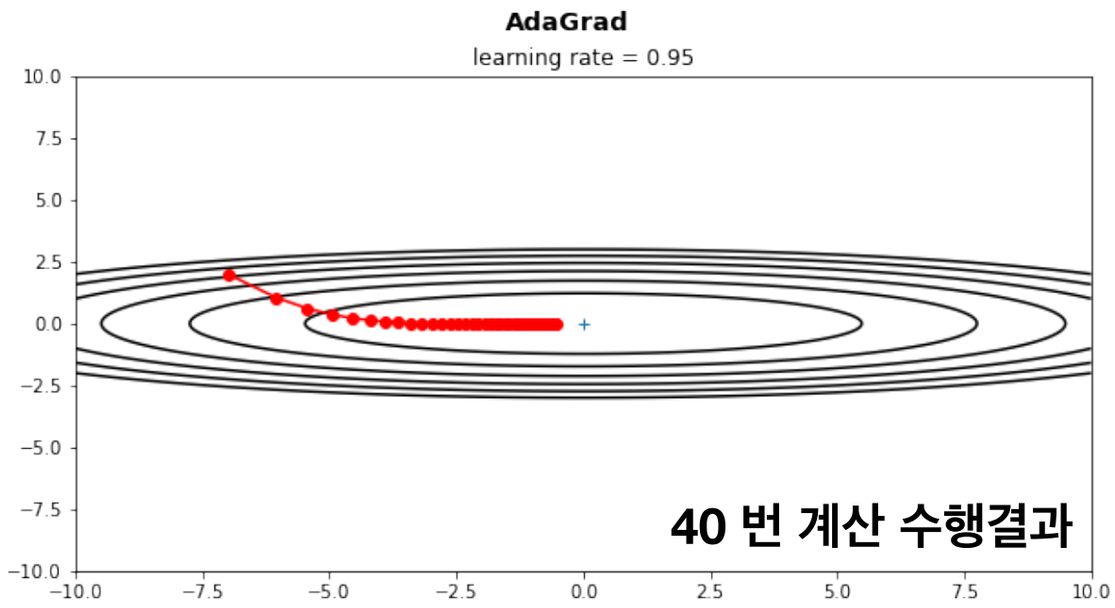
여기서 $\odot$ 는 element-wise 연산임

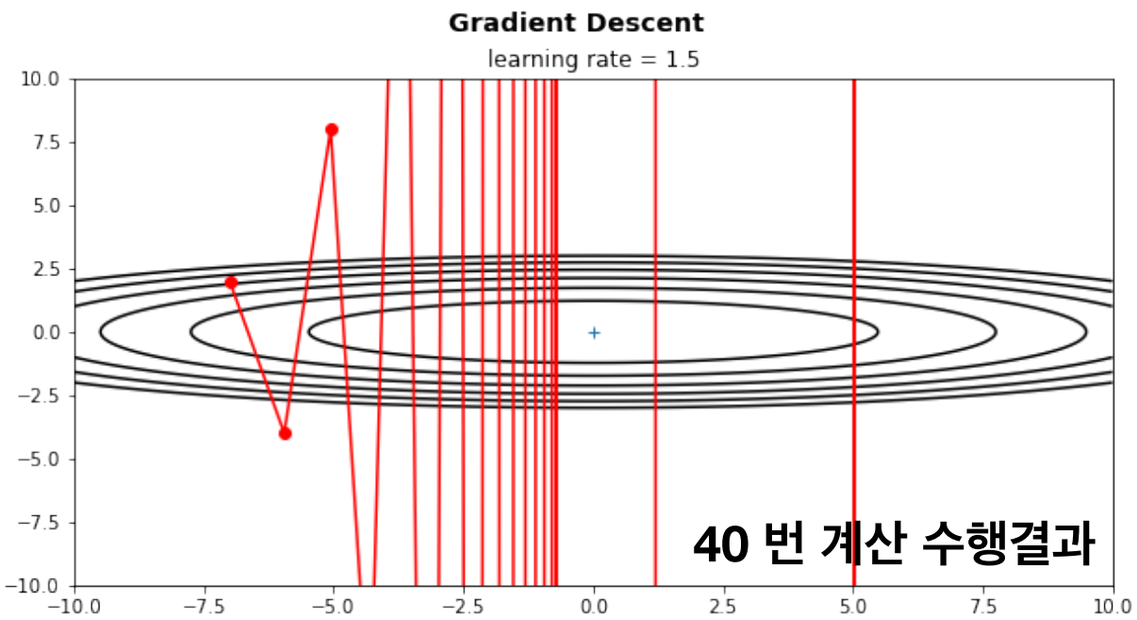
$\epsilon = 10^{-8}$ 정도로 잡음

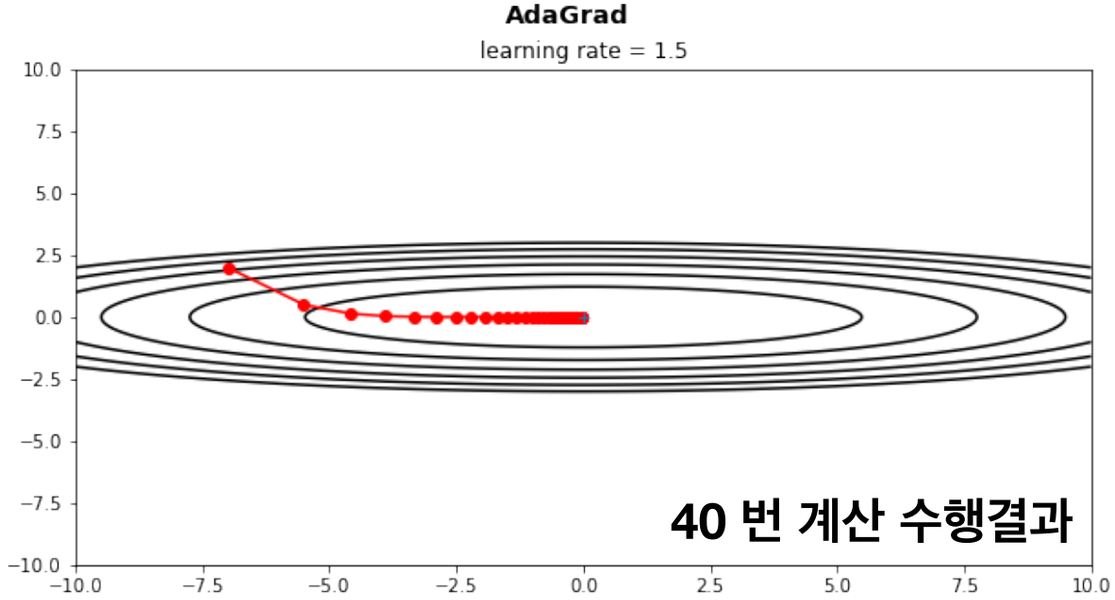

Gradient Descent
learning rate = 0.1
40 번 계산 수행결과


AdaGrad
learning rate = 0.1
40 번 계산 수행결과


Gradient Descent
learning rate = 0.95
40 번 계산 수행결과


AdaGrad
learning rate = 0.95
40 번 계산 수행결과


Gradient Descent
learning rate = 1.5
40 번 계산 수행결과


AdaGrad
learning rate = 1.5
40 번 계산 수행결과

Root Mean Square Propagation : RMSProp

- 학습이 진행될 수록,
  AdaGrad 의 학습률이 너무 작아지는 문제가 있음

$$\vec{G}_n \leftarrow \vec{G}_{n-1} + \nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$$

$$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\vec{G}_n + \epsilon}} \odot \nabla f(\vec{X}_{n-1})$$

- AdaGrad 의 학습률이 너무 작아지는 문제를 해결하고자 함.

- 학습이 진행될 수록, 과거의 기록의 영향을 감소시킴 (등비수열) $(\gamma < 1)$

$$\vec{G}_n \leftarrow \gamma\vec{G}_{n-1} + (1-\gamma)\nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$$

$$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\vec{G}_n + \epsilon}} \odot \nabla f(\vec{X}_{n-1})$$

$$g_1 = g_0 + f_x(x_0, y_0)^2 = f_x(x_0, y_0)^2$$

$$g_2 = g_1 + f_x(x_1, y_1)^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2$$

$$g_3 = g_2 + f_x(x_2, y_2)^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2 + f_x(x_2, y_2)^2$$

$$\vdots$$

$$g_n = g_{n-1} + f_x(x_{n-1}, y_{n-1})^2 = f_x(x_0, y_0)^2 + f_x(x_1, y_1)^2 + \cdots + f_x(x_{n-1}, y_{n-1})^2 = \sum_{i=0}^{n-1} f_x(x_i, y_i)^2$$

과거의 기록들이 동등하게 합해짐

- 학습이 진행될 수록, 과거의 기록의 영향을 감소시킴 ($\gamma < 1$ 등비수열 이용)

$$g_1 = \gamma g_0 + (1-\gamma) f_x(x_0, y_0)^2 = (1-\gamma) f_x(x_0, y_0)^2$$

$$g_2 = \gamma g_1 + (1-\gamma) f_x(x_1, y_1)^2 = \gamma(1-\gamma) f_x(x_0, y_0)^2 + (1-\gamma) f_x(x_1, y_1)^2$$

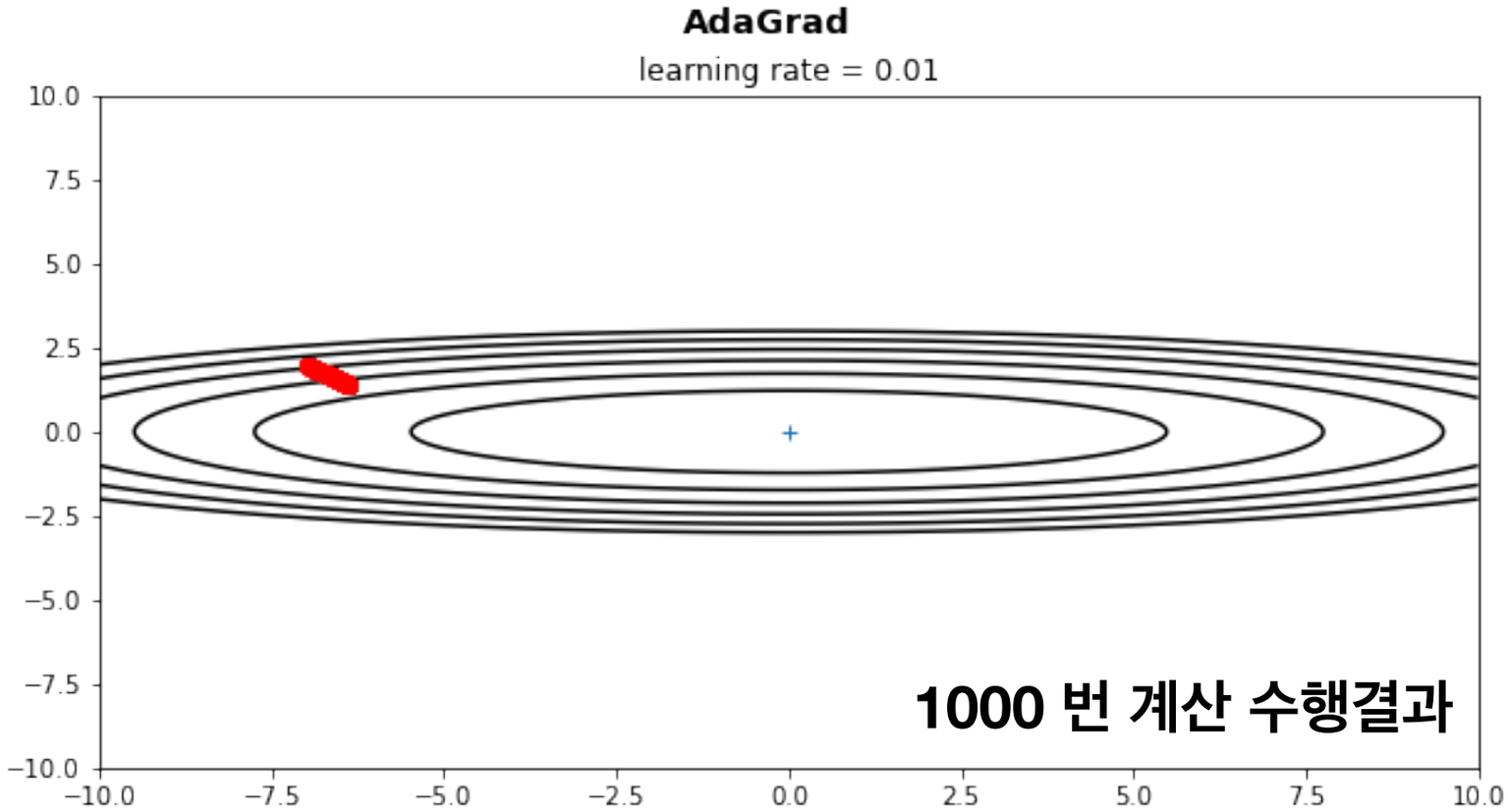
$$g_3 = \gamma g_2 + (1-\gamma) f_x(x_2, y_2)^2 = \gamma^2(1-\gamma) f_x(x_0, y_0)^2 + \gamma(1-\gamma) f_x(x_1, y_1)^2 + (1-\gamma) f_x(x_2, y_2)^2$$

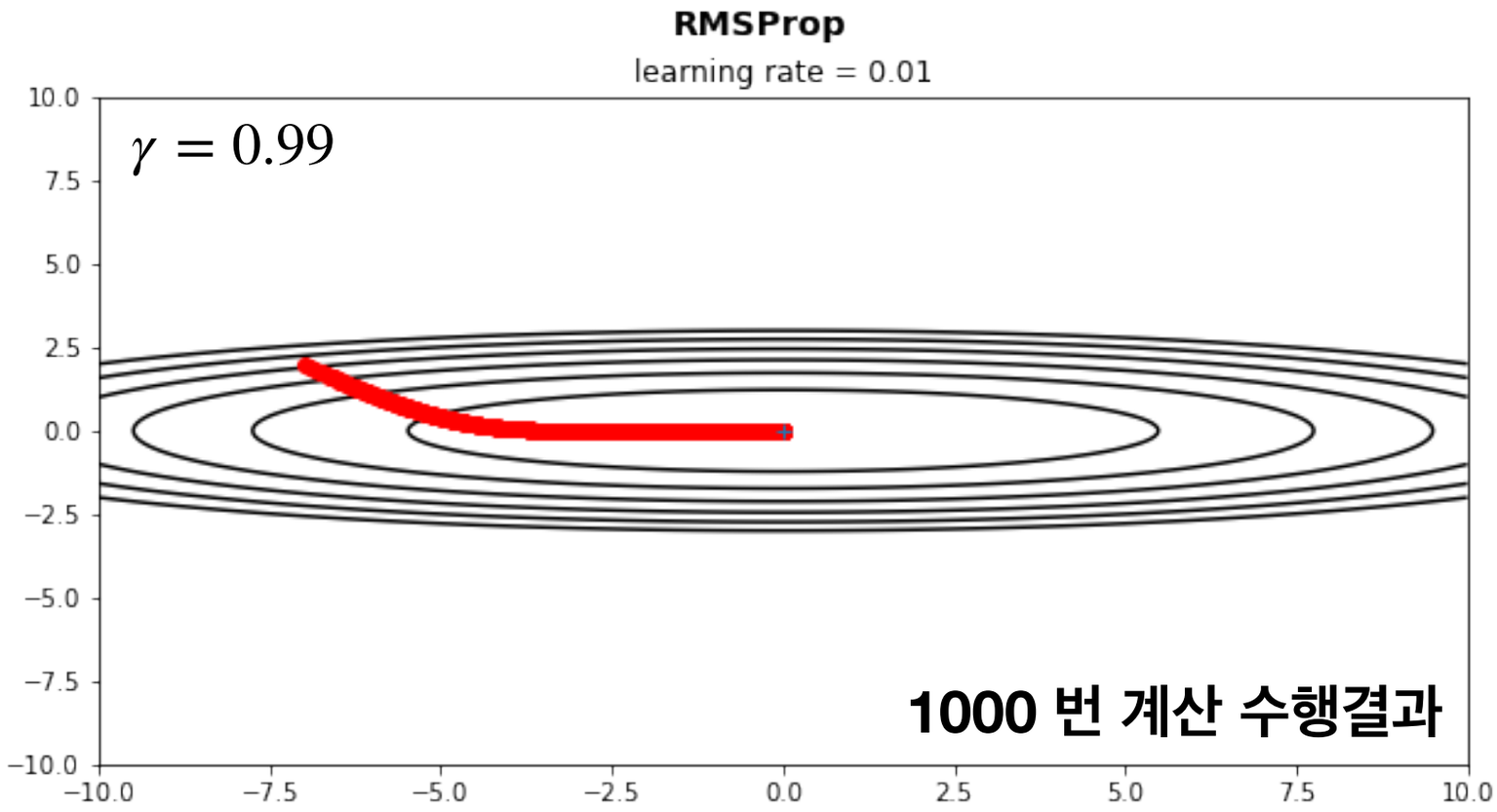
$$\vdots$$

$$g_n = \gamma^{n-1}(1-\gamma) f_x(x_0, y_0)^2 + \gamma^{n-2}(1-\gamma) f_x(x_1, y_1)^2 + \cdots + \gamma(1-\gamma) f_x(x_{n-2}, y_{n-2})^2 + (1-\gamma) f_x(x_{n-1}, y_{n-1})^2$$

$$= (1-\gamma) \sum_{i=0}^{n-1} \gamma^{(n-1)-i} f_x(x_i, y_i)^2$$

작은 γ 값일 수록, 최근의 Gradient에 더욱 집중을 함


AdaGrad
learning rate = 0.01
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
1000 번 계산 수행결과


RMSProp
learning rate = 0.01
$\gamma = 0.99$
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0
1000 번 계산 수행결과

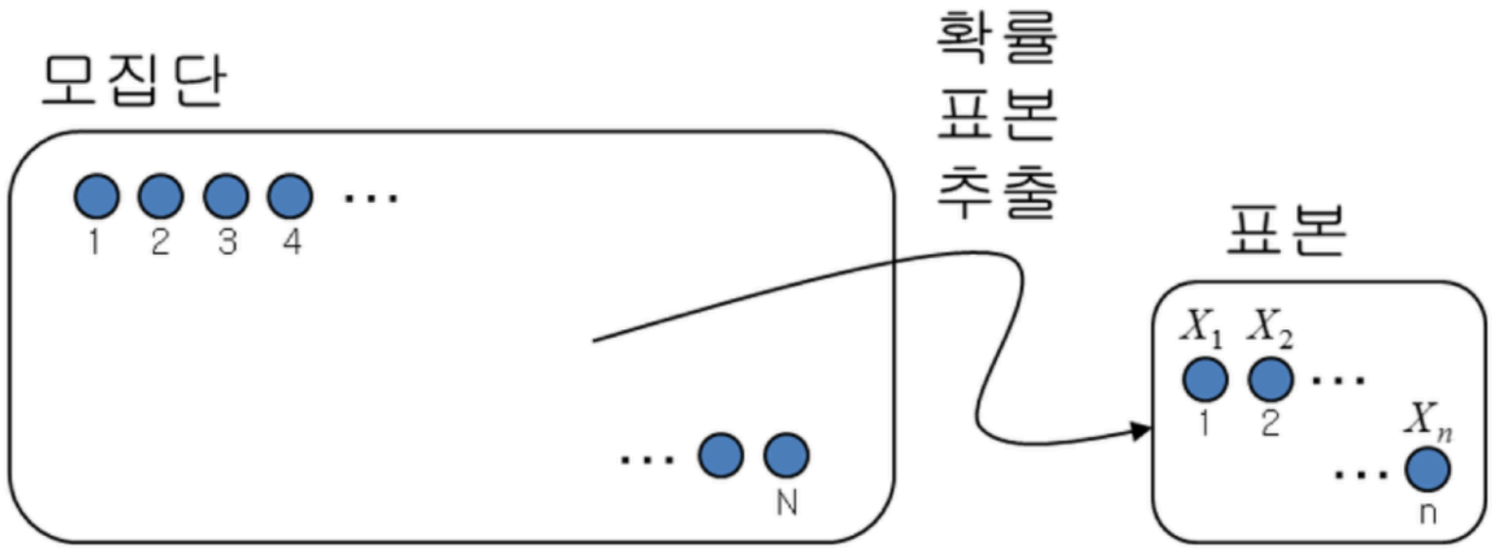
Adaptive Moment Estimation= Adam

- AdaGrad 와 RMSProp 의 장점을 합침.
  - Sparse gradient 에 작동 잘함 (성분 중에 0 값이 많은 벡터)
  - Non-stationary 문제에 효과적임 (최근의 변화에 더욱 강조를 둠)

$$\vec{G}_n \leftarrow \gamma \vec{G}_{n-1} + (1-\gamma)\nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$$

$$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\vec{G}_n + \epsilon}} \odot \nabla f(\vec{X}_{n-1})$$

- 실제, 데이터를 가지고 함수를 최적화 시킬 때, 추출된 "표본"이 "모집단"을 잘 표현하는지가 관건



- 확률변수 X 의 n 차 모멘트 = $\mathbb{E}[X^n] = \sum X^n p(X)$

- $\nabla f(\overrightarrow{X})$ 의 1차 모멘트: $\mathbb{E}[\nabla f(\overrightarrow{X})]$ 예측

$$\overrightarrow{M}_n \leftarrow \beta_1 \overrightarrow{M}_{n-1} + (1-\beta_1)\nabla f(\overrightarrow{X}_{n-1})$$

- $\nabla f(\overrightarrow{X})$ 의 2차 모멘트: $\mathbb{E}[\nabla f(\overrightarrow{X}) \odot \nabla f(\overrightarrow{X})]$ 예측

$$\overrightarrow{V}_n \leftarrow \beta_2 \overrightarrow{V}_{n-1} + (1-\beta_2)\nabla f(\overrightarrow{X}_{n-1}) \odot \nabla f(\overrightarrow{X}_{n-1})$$

- 수열 $\overrightarrow{M}_n \leftarrow \beta_1 \overrightarrow{M}_{n-1} + (1-\beta_1)\nabla f(\overrightarrow{X}_{n-1})$을 이용하여, $\overrightarrow{M}_n$ 을 구함

0. $\overrightarrow{M}_0 = \overrightarrow{0}$

1. $\overrightarrow{M}_1 = \beta_1 \overrightarrow{M}_0 + (1-\beta_1)\nabla f(\overrightarrow{X}_0) = (1-\beta_1)\nabla f(\overrightarrow{X}_0)$

2. $\overrightarrow{M}_2 = \beta_1 \overrightarrow{M}_1 + (1-\beta_1)\nabla f(\overrightarrow{X}_1) = \beta_1(1-\beta_1)\nabla f(\overrightarrow{X}_0) + (1-\beta_1)\nabla f(\overrightarrow{X}_1)$

3. $\overrightarrow{M}_3 = \beta_1 \overrightarrow{M}_2 + (1-\beta_1)\nabla f(\overrightarrow{X}_2) = \beta_1^2(1-\beta_1)\nabla f(\overrightarrow{X}_0) + \beta_1(1-\beta_1)\nabla f(\overrightarrow{X}_1) + (1-\beta_1)\nabla f(\overrightarrow{X}_2)$

따라서, $\overrightarrow{M}_n = (1-\beta_1)\sum_{i=0}^{n-1} \beta_1^{(n-1)-i}\nabla f(\overrightarrow{X}_i)$

- 비슷하게, 수열 $\overrightarrow{V}_n \leftarrow \beta_2 \overrightarrow{V}_{n-1} + (1-\beta_2)\nabla f(\overrightarrow{X}_{n-1}) \odot \nabla f(\overrightarrow{X}_{n-1})$을 이용하면,

$$\overrightarrow{V}_n = (1-\beta_2)\sum_{i=0}^{n-1} \beta_2^{(n-1)-i}\left[\nabla f(\overrightarrow{X}_i) \odot \nabla f(\overrightarrow{X}_i)\right]$$

$\zeta = \mathbb{E}\left[(1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i}\left(\nabla f(\overrightarrow{X}_i) - \nabla f(\overrightarrow{X}_{n-1})\right)\right]$ 으로 정의하면,

1) i 가 작을 때는 $\beta_1^{(n-1)-i} \ll 1$

2) $i \simeq n$ 일 때는 $\nabla f(\overrightarrow{X}_i) - \nabla f(\overrightarrow{X}_{n-1}) \ll 1$

이 되므로 $\zeta \ll 1$ (작은수) 로 간주될 수 있다.

$$\zeta = \mathbb{E}\left[(1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i}\left(\nabla f(\overrightarrow{X}_i) - \nabla f(\overrightarrow{X}_{n-1})\right)\right] = \mathbb{E}\left[(1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i}\nabla f(\overrightarrow{X}_i)\right] - \mathbb{E}\left[(1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i}\nabla f(\overrightarrow{X}_{n-1})\right]$$

$$(1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i} = (1-\beta_1)\frac{1-\beta^n}{1-\beta_1} = 1-\beta_1^n$$

$$= \mathbb{E}\left[\overrightarrow{M}_n\right] - (1-\beta_1)\sum_{i=0}^{n-1}\beta_1^{(n-1)-i} \times \mathbb{E}\left[\nabla f(\overrightarrow{X}_{n-1})\right]$$

$$= \mathbb{E}\left[\overrightarrow{M}_n\right] - (1-\beta_1^n)\mathbb{E}\left[\nabla f(\overrightarrow{X}_{n-1})\right]$$

- 따라서, $\mathbb{E}\left[\dfrac{\overrightarrow{M}_n}{1-\beta_1^n}\right] \simeq \mathbb{E}\left[\nabla f(\overrightarrow{X}_{n-1})\right]$ 가 된다.

- $$\mathbb{E}\left[\frac{\vec{M}_n}{1-\beta_1^n}\right] \simeq \mathbb{E}\left[\nabla f(\vec{X}_{n-1})\right].$$

- 비슷한 방식으로 $$\mathbb{E}\left[\frac{\vec{V}_n}{1-\beta_2^n}\right] \simeq \mathbb{E}\left[\nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})\right].$$

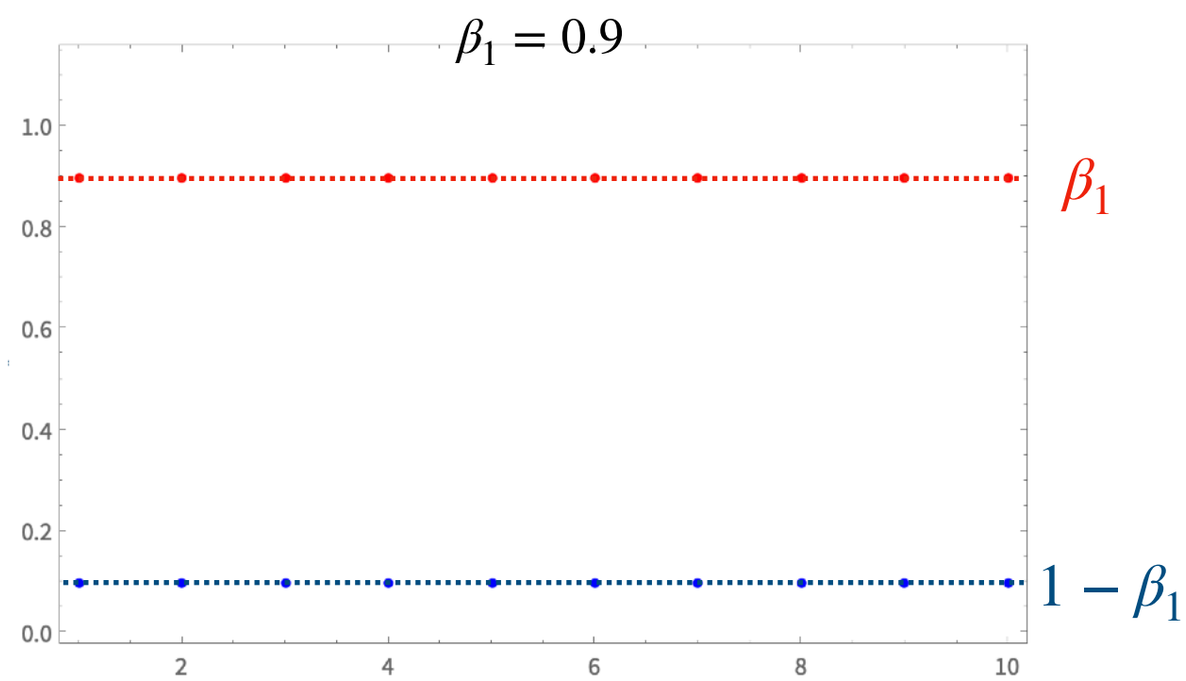
이를 Bias Correction 이라고 함.

- $\vec{M}_0 = \vec{V}_0 = \vec{0}$ 으로 초기화 되기 때문에, (for $\beta_1 \lesssim 1$)

$$\vec{M}_1 \leftarrow \beta_1 \vec{M}_0 + (1-\beta_1)\nabla f(\vec{X}_0)$$

$$\vec{M}_2 \leftarrow \beta_1 \vec{M}_1 + (1-\beta_1)\nabla f(\vec{X}_1)$$

$$\vec{M}_n \leftarrow \beta_1 \vec{M}_{n-1} + (1-\beta_1)\nabla f(\vec{X}_{n-1})$$



- $$\mathbb{E}\left[\frac{\vec{M}_n}{1-\beta_1^n}\right] \simeq \mathbb{E}\left[\nabla f(\vec{X}_{n-1})\right].$$

- 비슷한 방식으로 $$\mathbb{E}\left[\frac{\vec{V}_n}{1-\beta_2^n}\right] \simeq \mathbb{E}\left[\nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})\right].$$

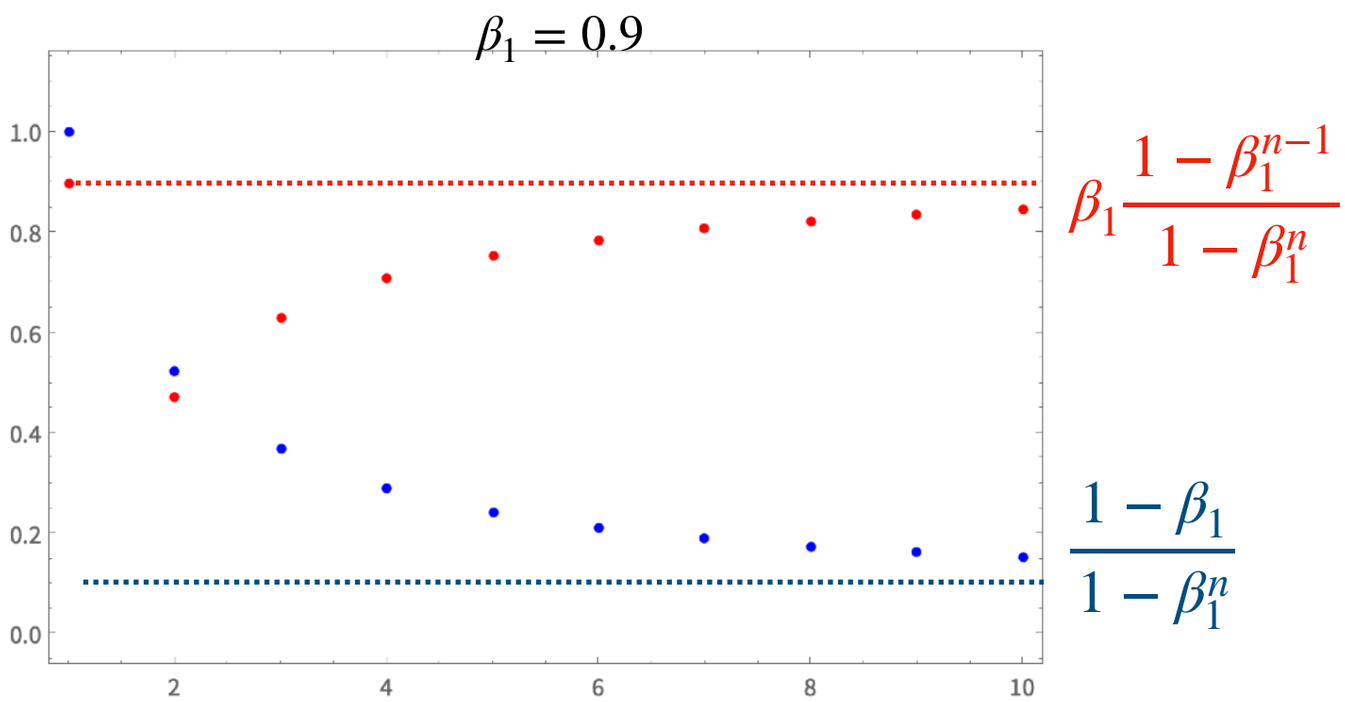
이를 Bias Correction 이라고 함.

- $\vec{M}_0 = \vec{V}_0 = \vec{0}$ 의 초기화 영향을 받지 않음 (for $\beta_1 \lesssim 1$)

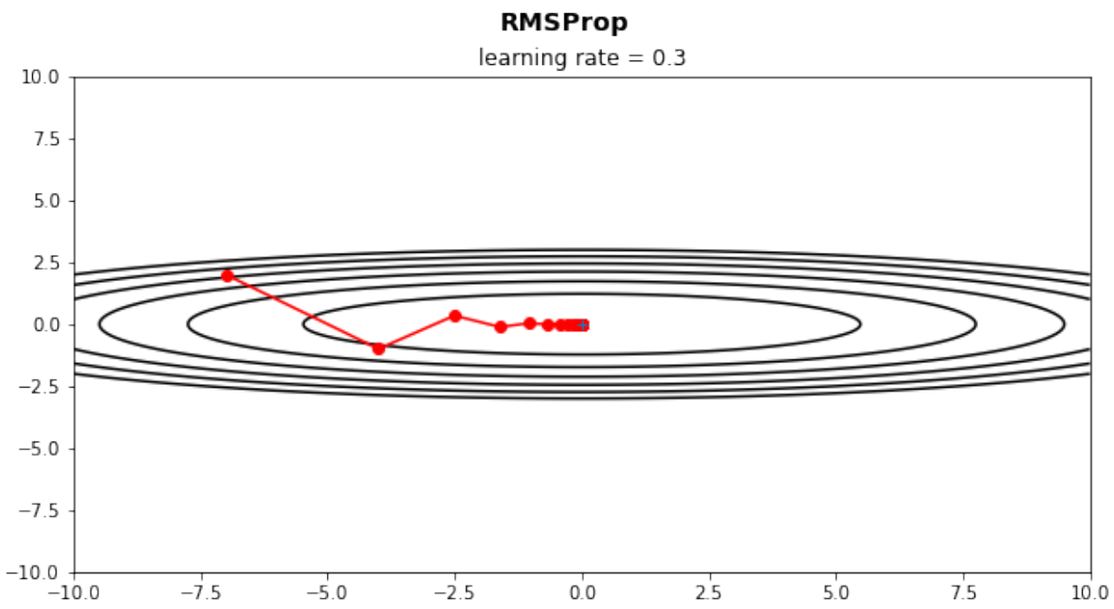
$$\hat{M}_1 \leftarrow \beta_1 \hat{M}_0 + \frac{1-\beta_1}{1-\beta_1}\nabla f(\vec{X}_0)$$

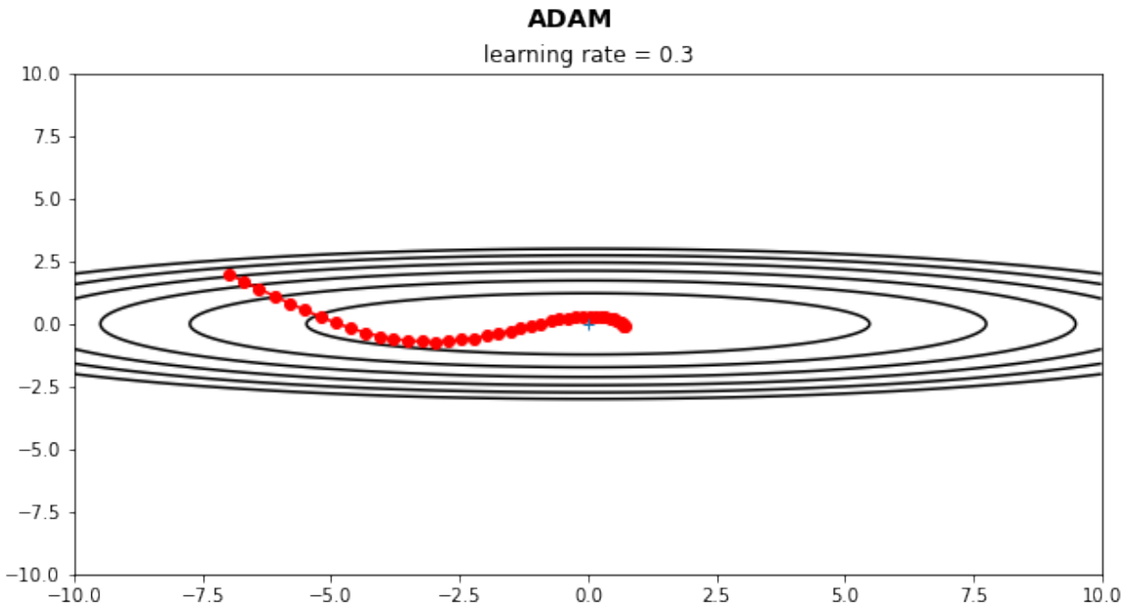
$$\hat{M}_2 \leftarrow \beta_1 \frac{1-\beta_1}{1-\beta_1^2}\hat{M}_1 + \frac{1-\beta_1}{1-\beta_1^2}\nabla f(\vec{X}_1)$$

$$\hat{M}_n \leftarrow \beta_1 \frac{1-\beta_1^{n-1}}{1-\beta_1^n}\hat{M}_{n-1} + \frac{1-\beta_1}{1-\beta_1^n}\nabla f(\vec{X}_{n-1})$$



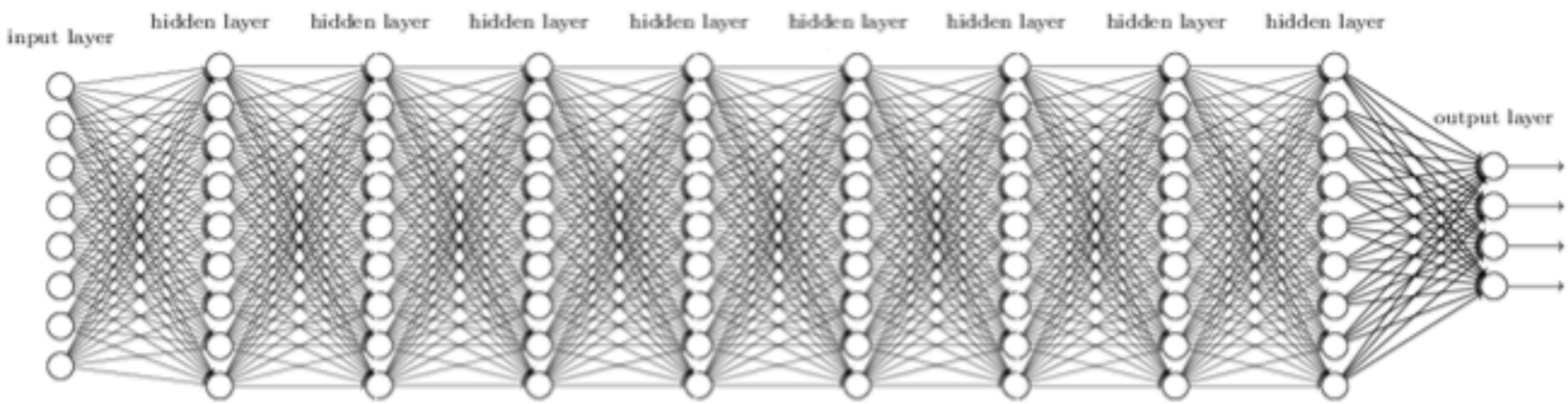
RMSProp	ADAM
$\vec{G}_n \leftarrow \vec{G}_{n-1} + \nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$	$\vec{M}_n \leftarrow \beta_1 \vec{M}_{n-1} + (1-\beta_1)\nabla f(\vec{X}_{n-1})$ $\vec{V}_n \leftarrow \beta_2 \vec{V}_{n-1} + (1-\beta_2)\nabla f(\vec{X}_{n-1}) \odot \nabla f(\vec{X}_{n-1})$
	$\hat{M}_n = \frac{\vec{M}_{n-1}}{1-\beta_1^n}, \quad \hat{V}_n = \frac{\vec{V}_{n-1}}{1-\beta_2^n}$
$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\vec{G}_n + \epsilon}} \odot \nabla f(\vec{X}_{n-1})$	$\vec{X}_n \leftarrow \vec{X}_{n-1} - \frac{\alpha}{\sqrt{\hat{V}_n + \epsilon}} \odot \hat{M}_n$



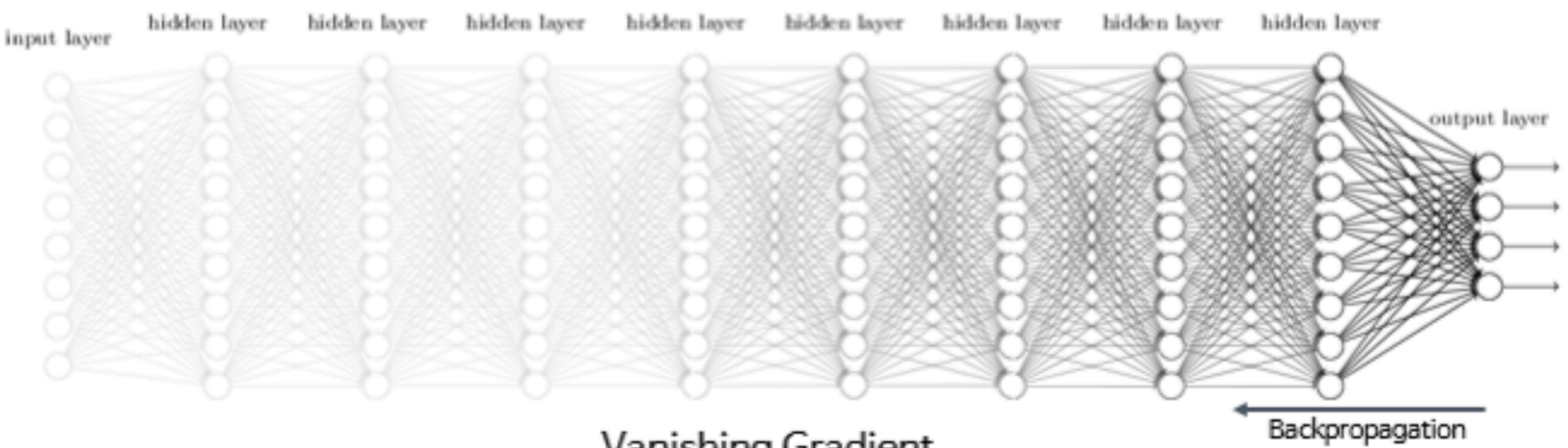


- **기울기 손실 문제 (Vanishing Gradient)**

  역전파 알고리즘에 따른 입력층 Gradient 가 점점 작아져, 결국 가중치 매개변수 업데이트가 제대로 이루어지지 않는 경우

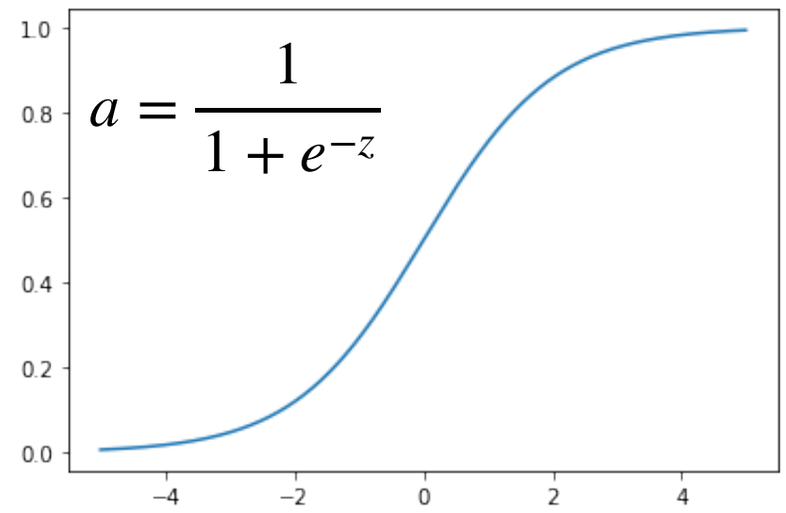


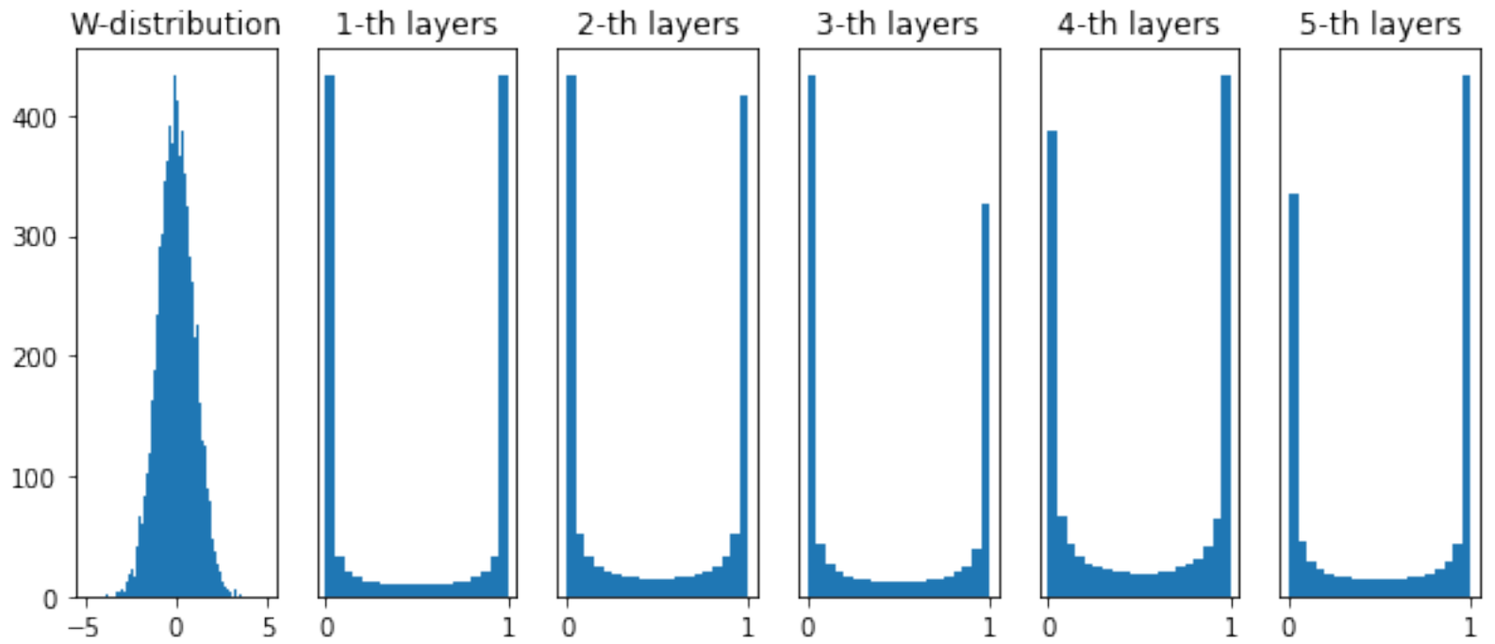
Deep Neural Network

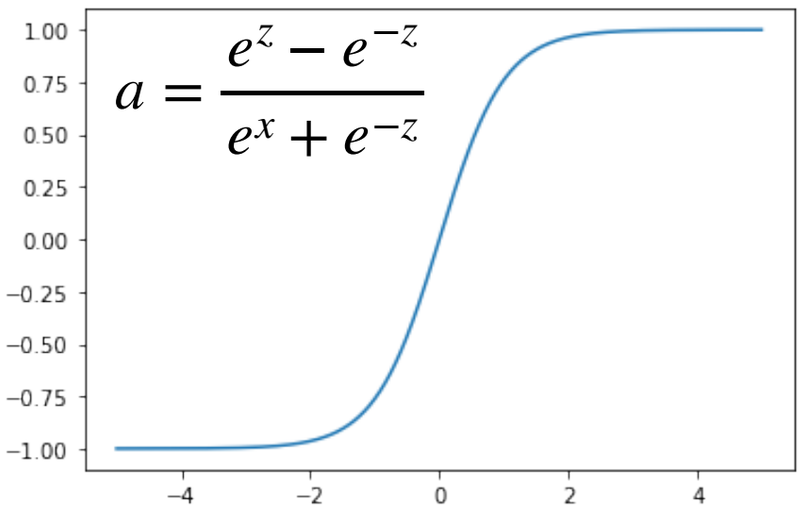


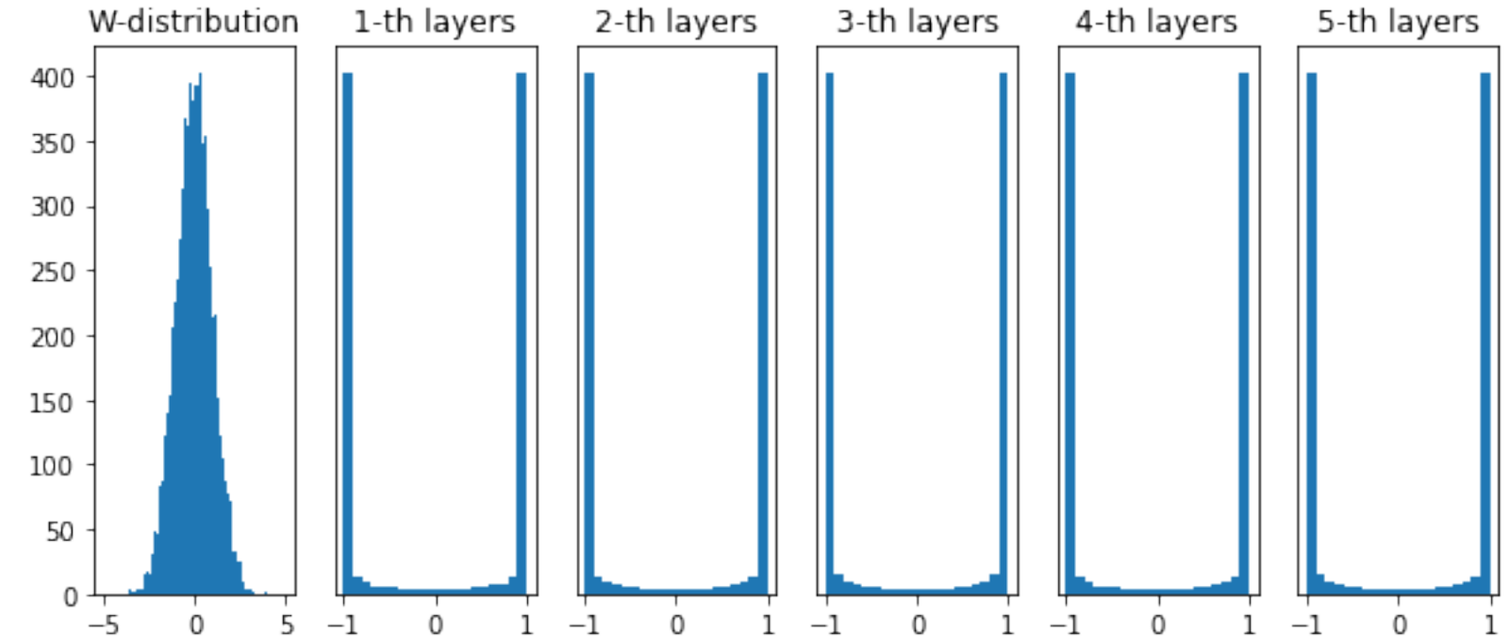
Vanishing Gradient

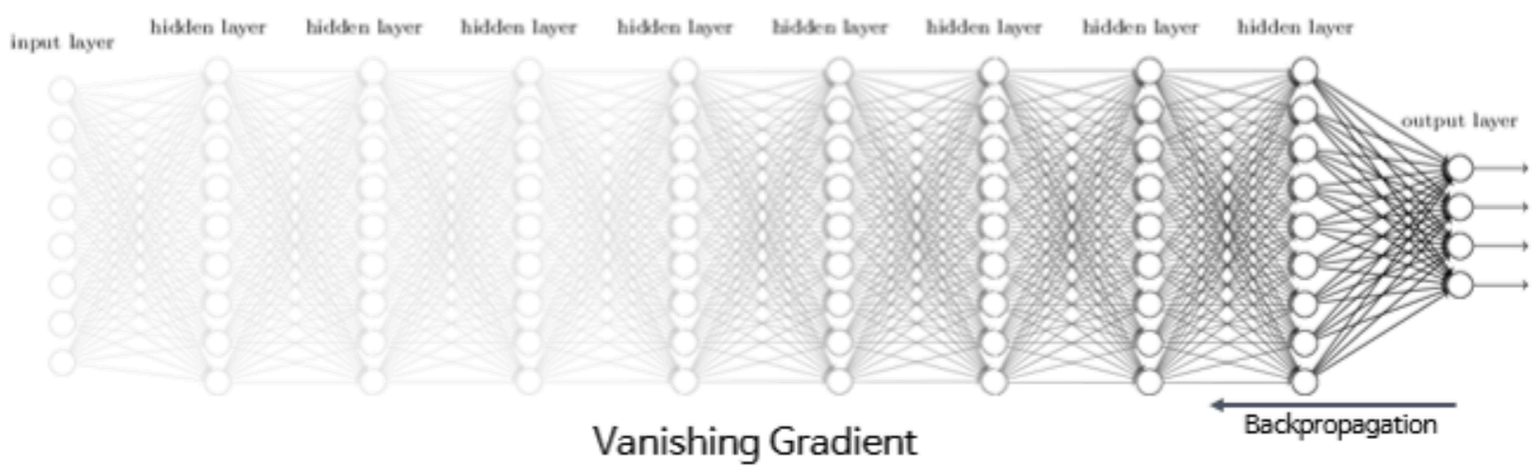
w=np.random.randn*1



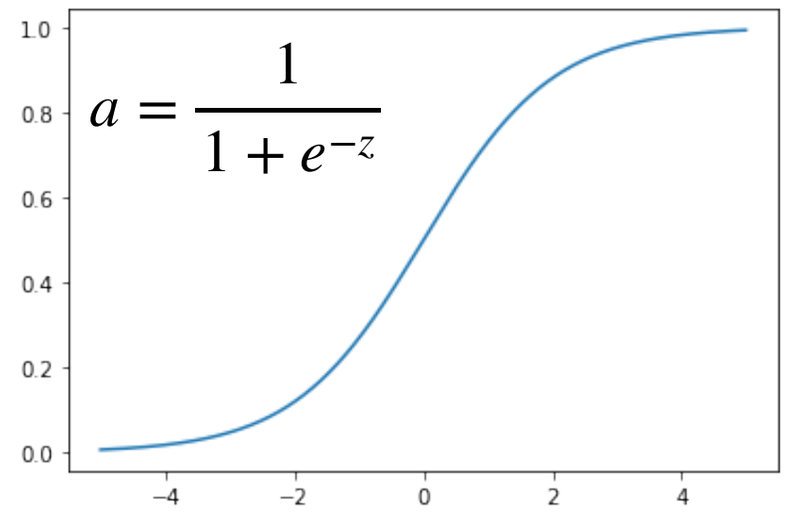


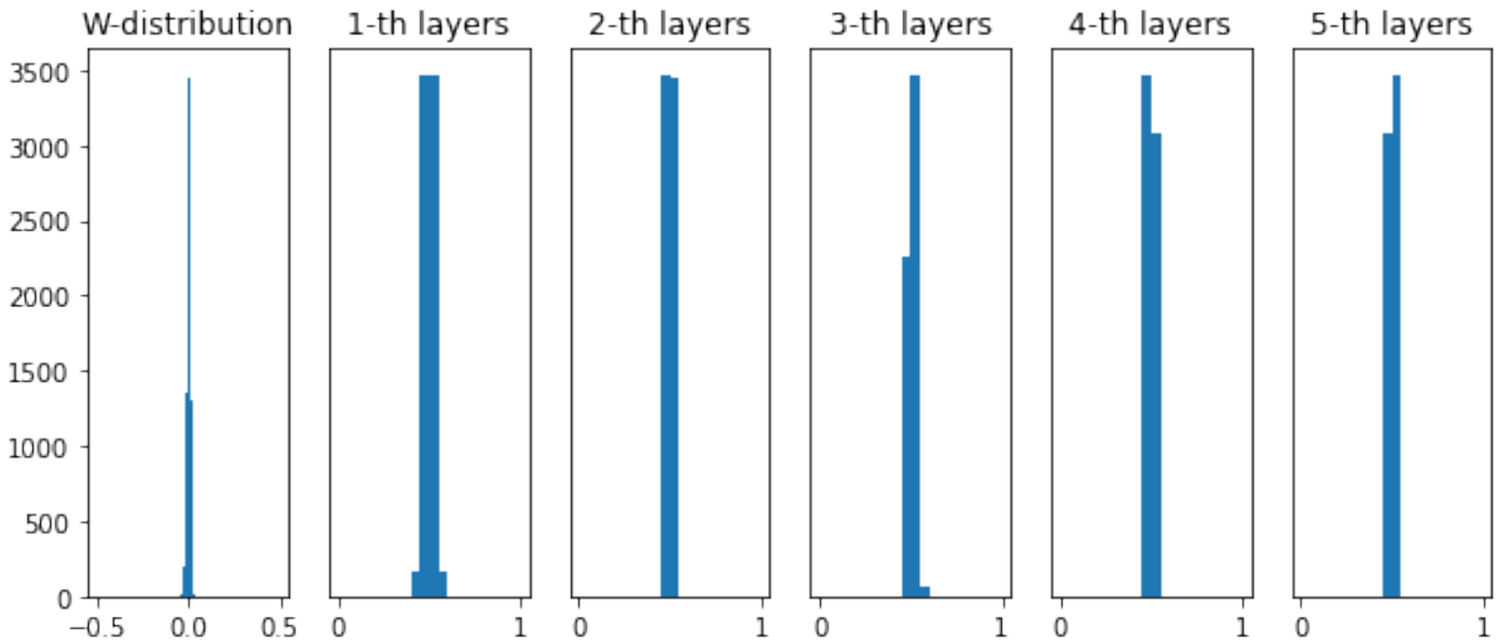


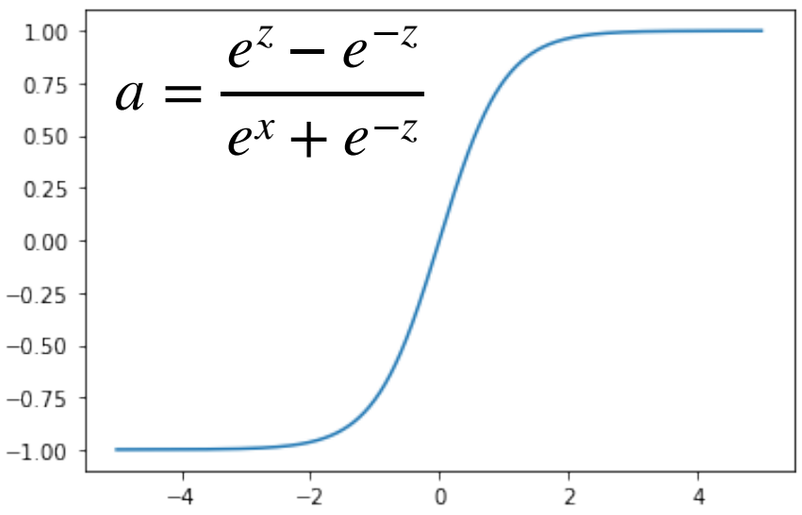


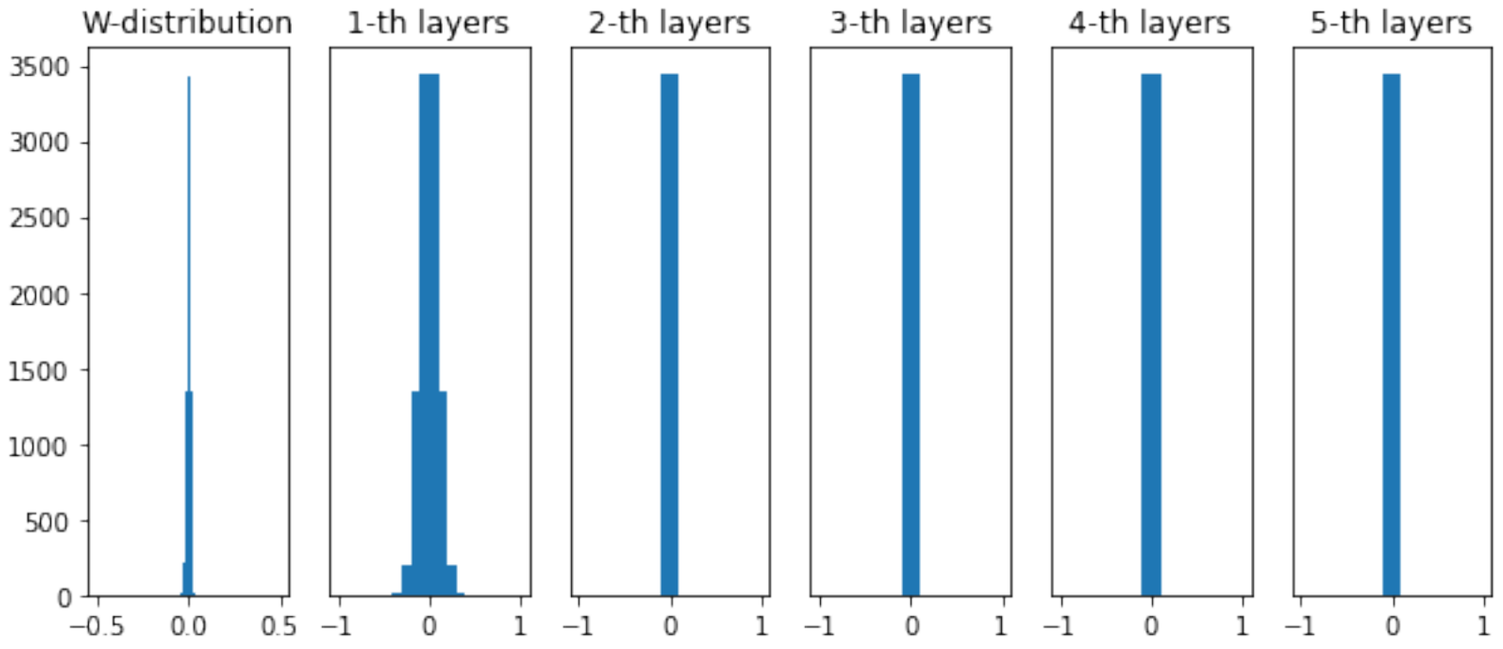


w=np.random.randn*0.01



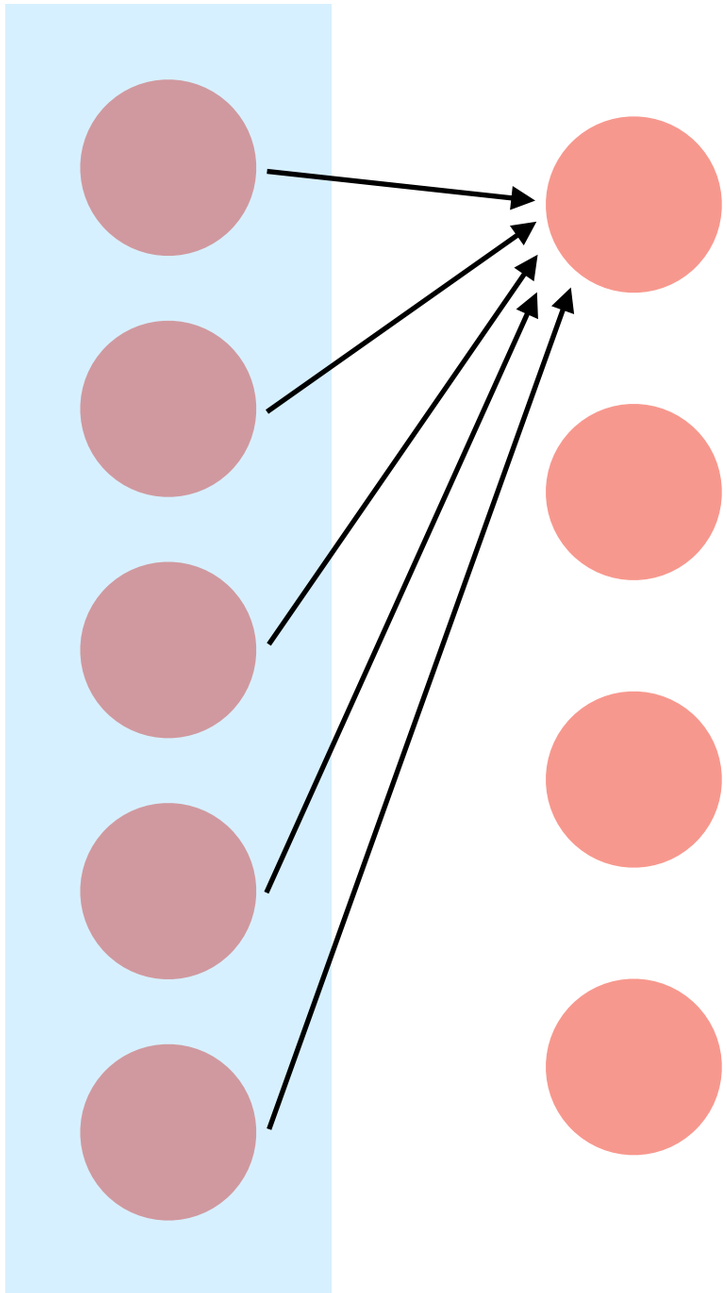






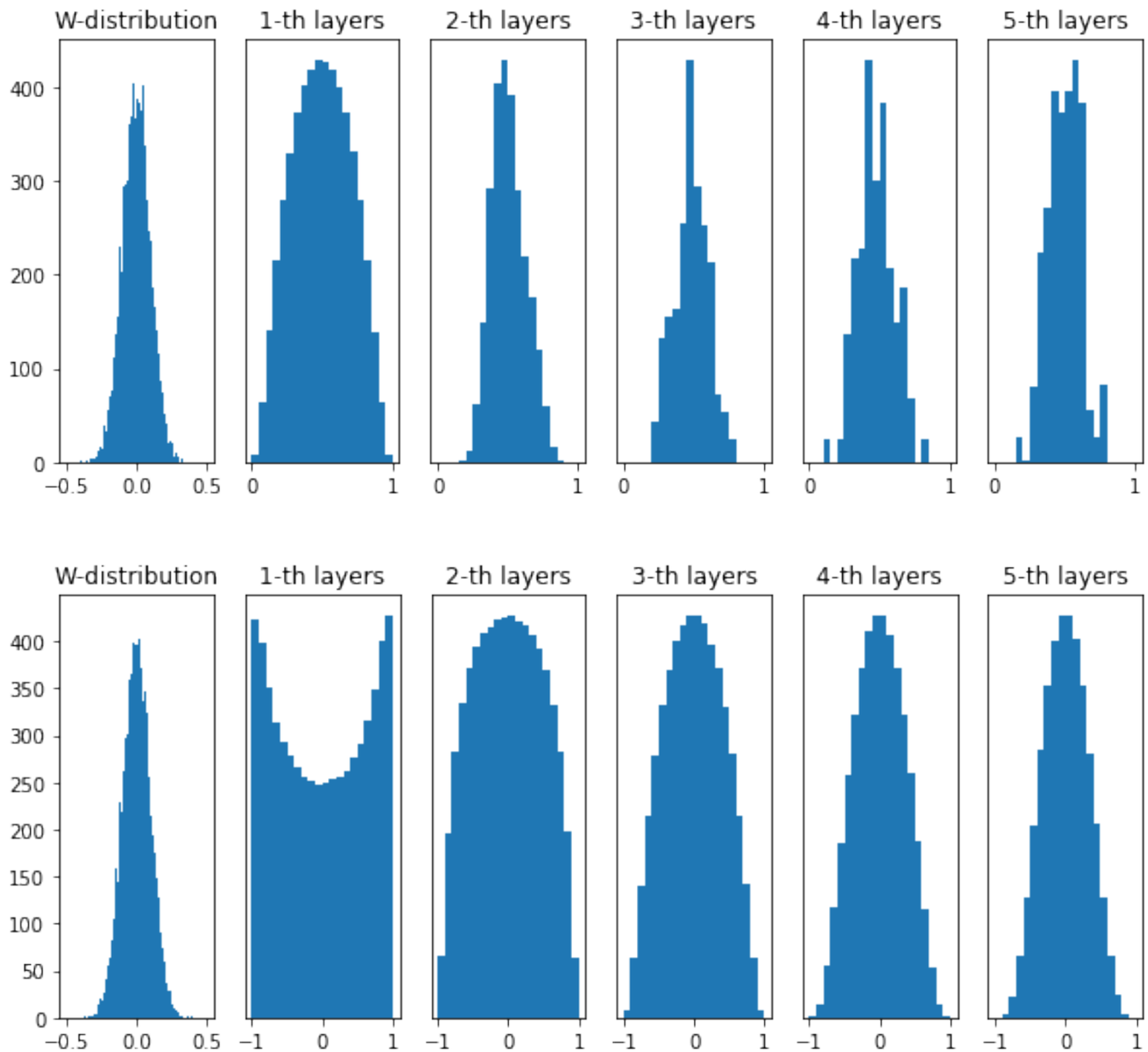
각 층의 활성화 노드 값들이 하나의 값을 갖게 되어, 노드를 늘린 효과가 없어짐.

- **기울기 손실 문제 해결: Xavier 초기값**

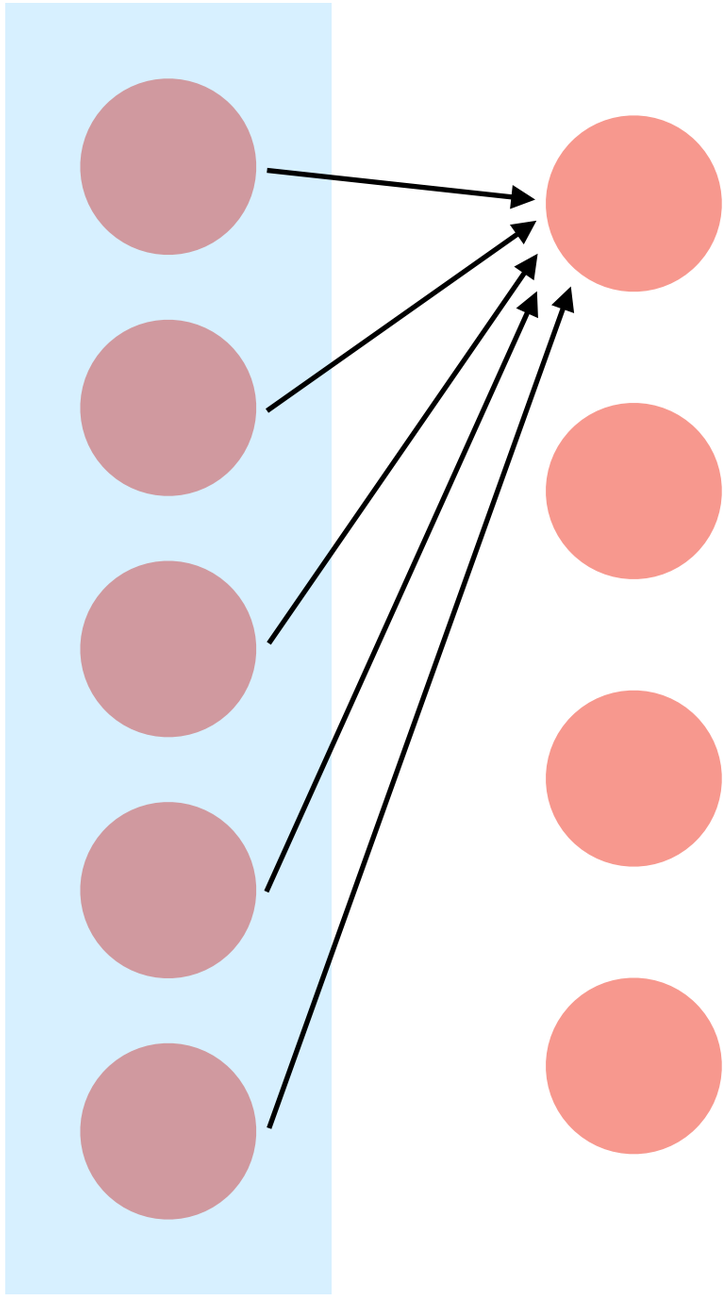


n 개 노드

가중치를 표준편차가 $\frac{1}{\sqrt{n}}$ 인 정규분포로 초기화

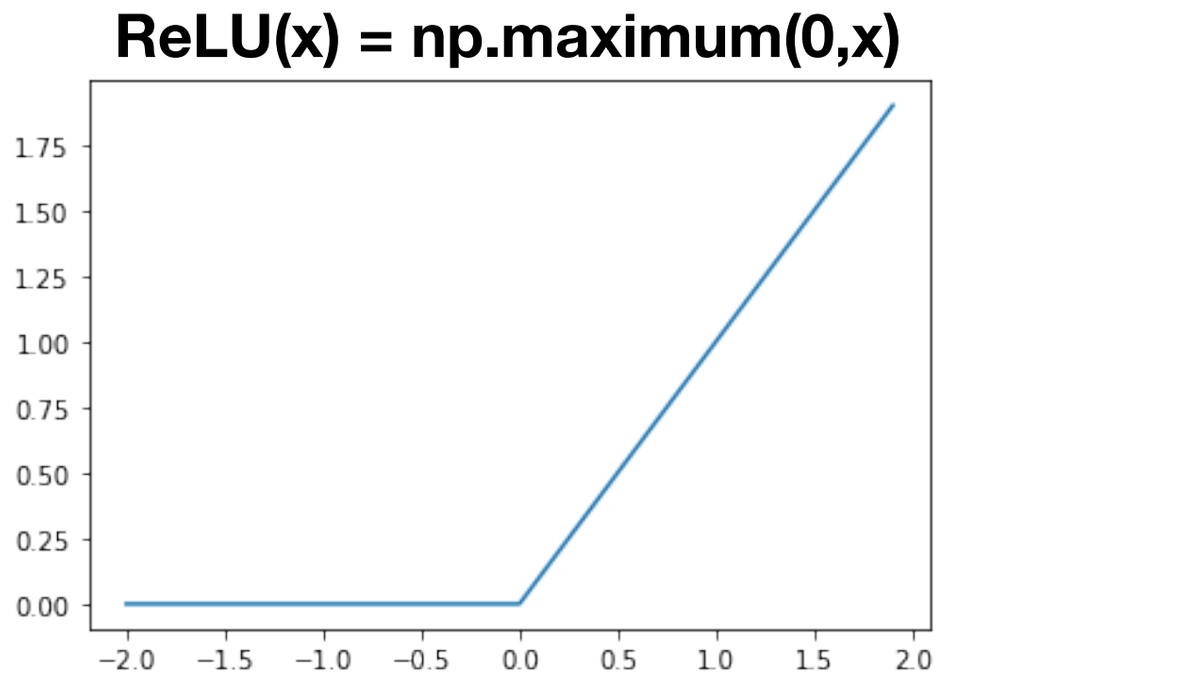


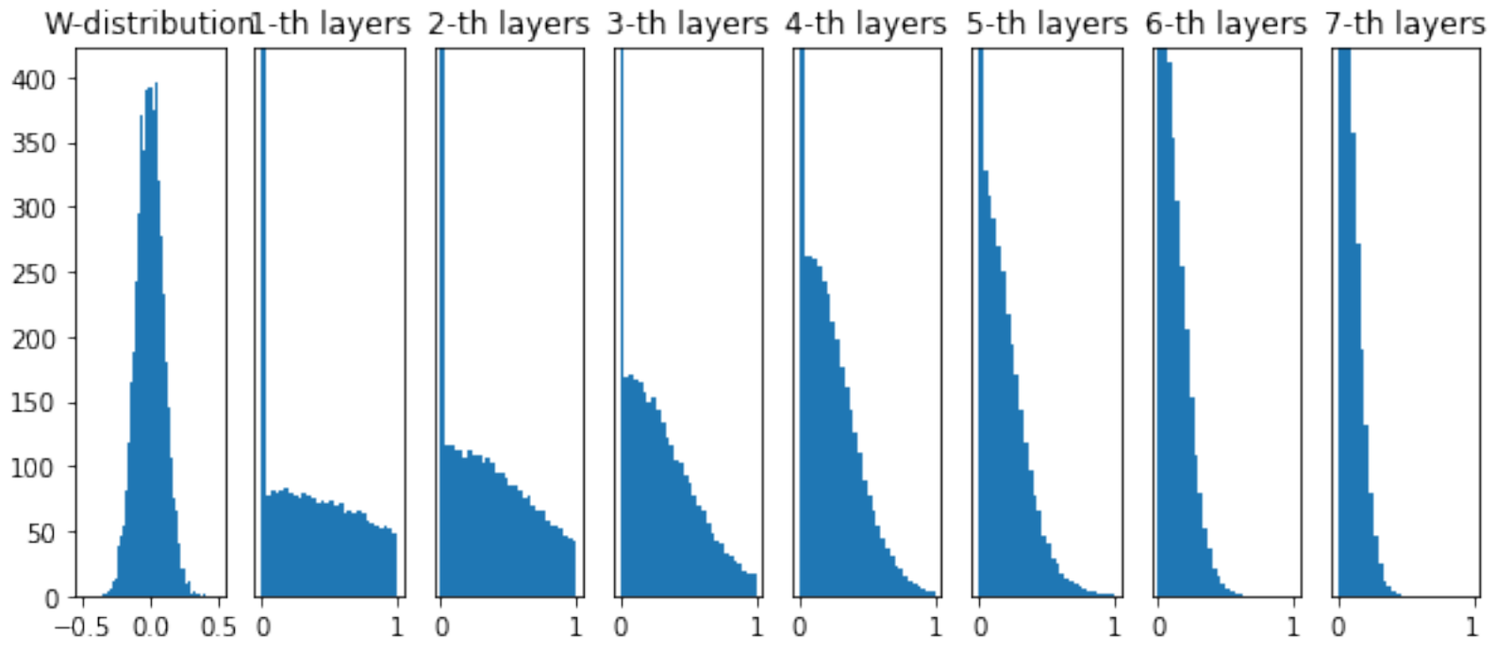
- **기울기 손실 문제 해결: Xavier 초기값 ???**



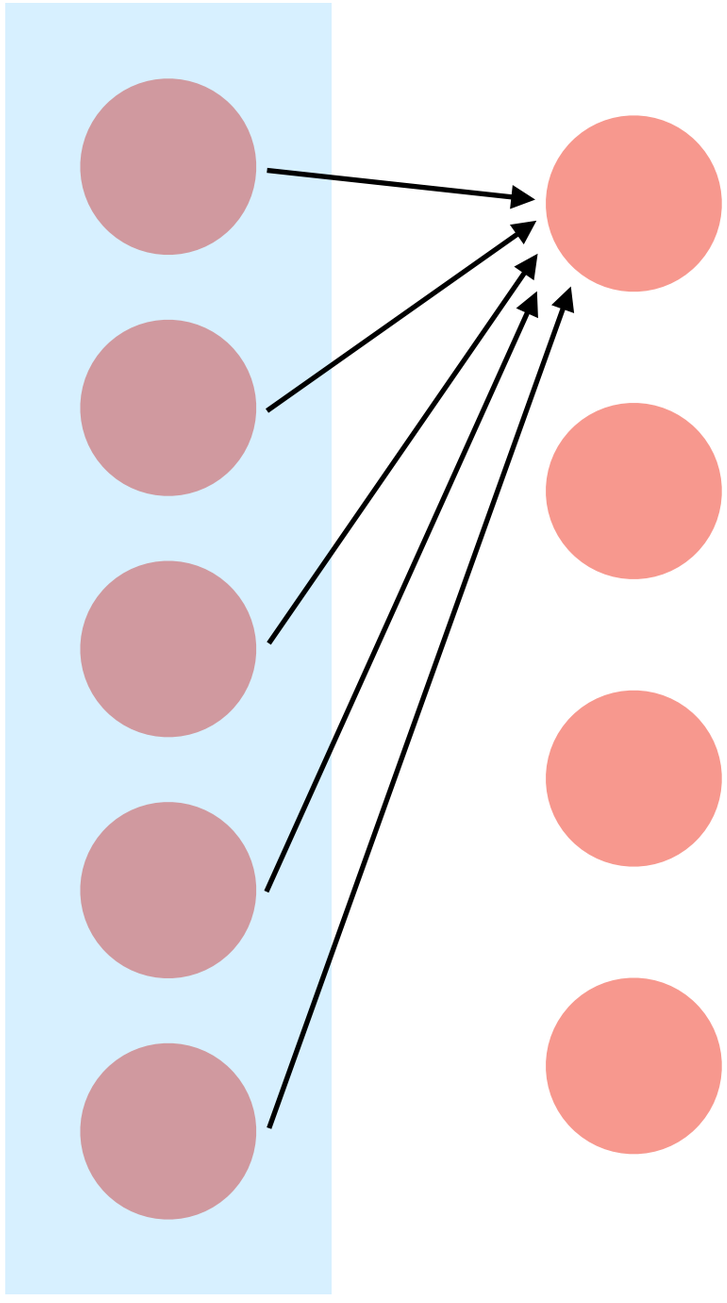
n 개 노드

가중치를 표준편차가 $\frac{1}{\sqrt{n}}$ 인 정규분포로 초기화

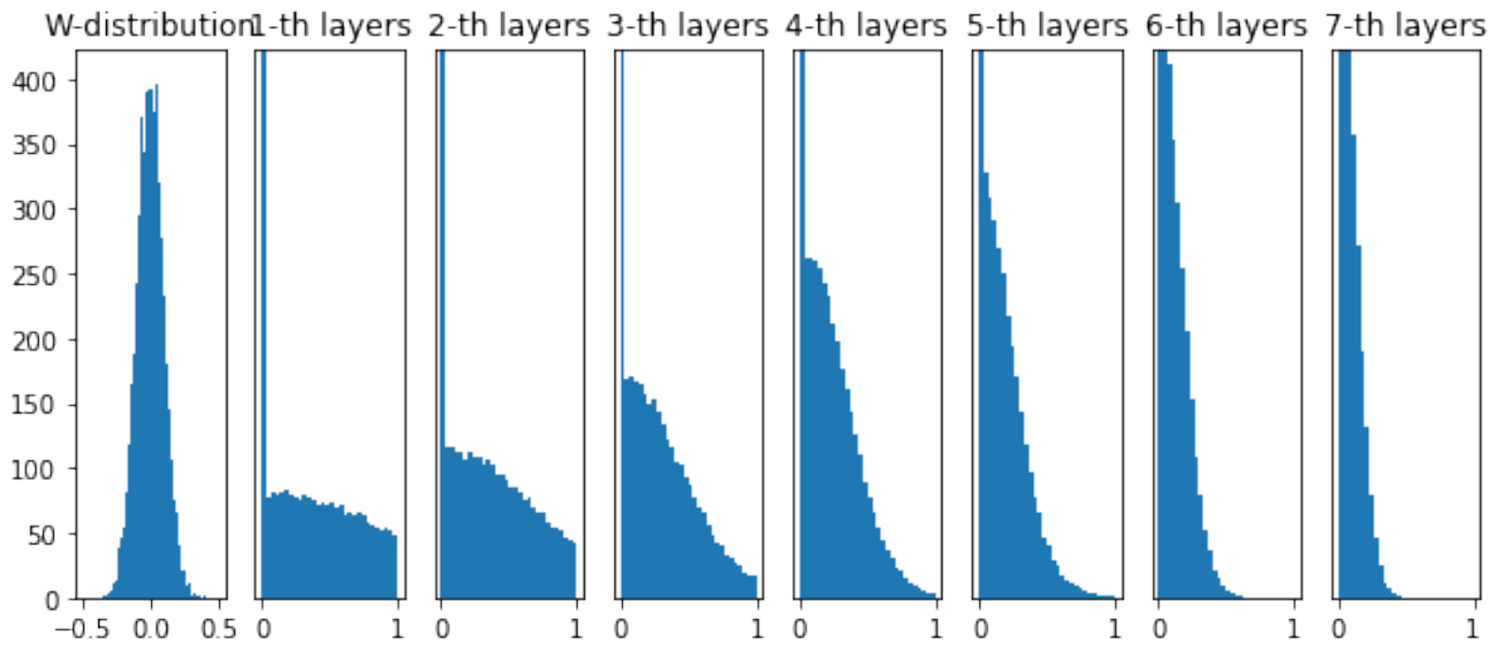




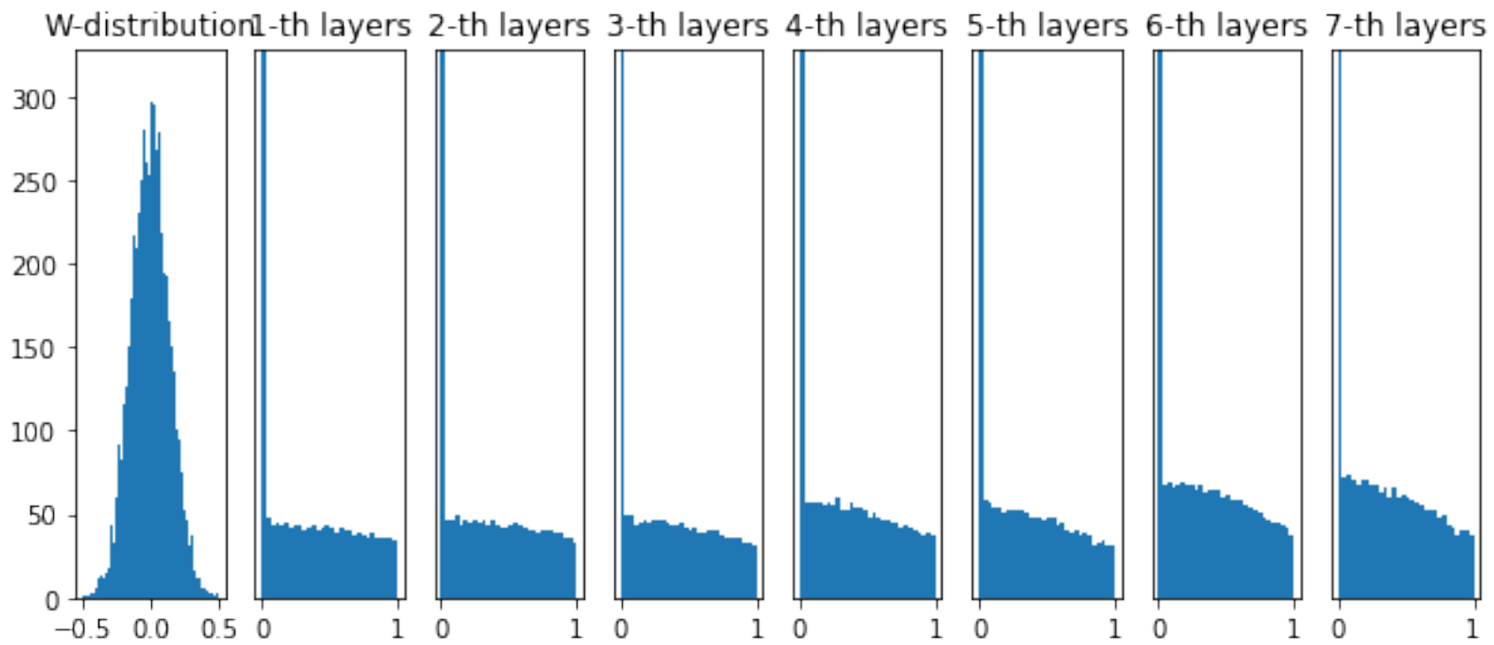
- **기울기 손실 문제 해결: Xavier 초기값 → He 초기값 (ReLU 인 경우)**



n 개 노드



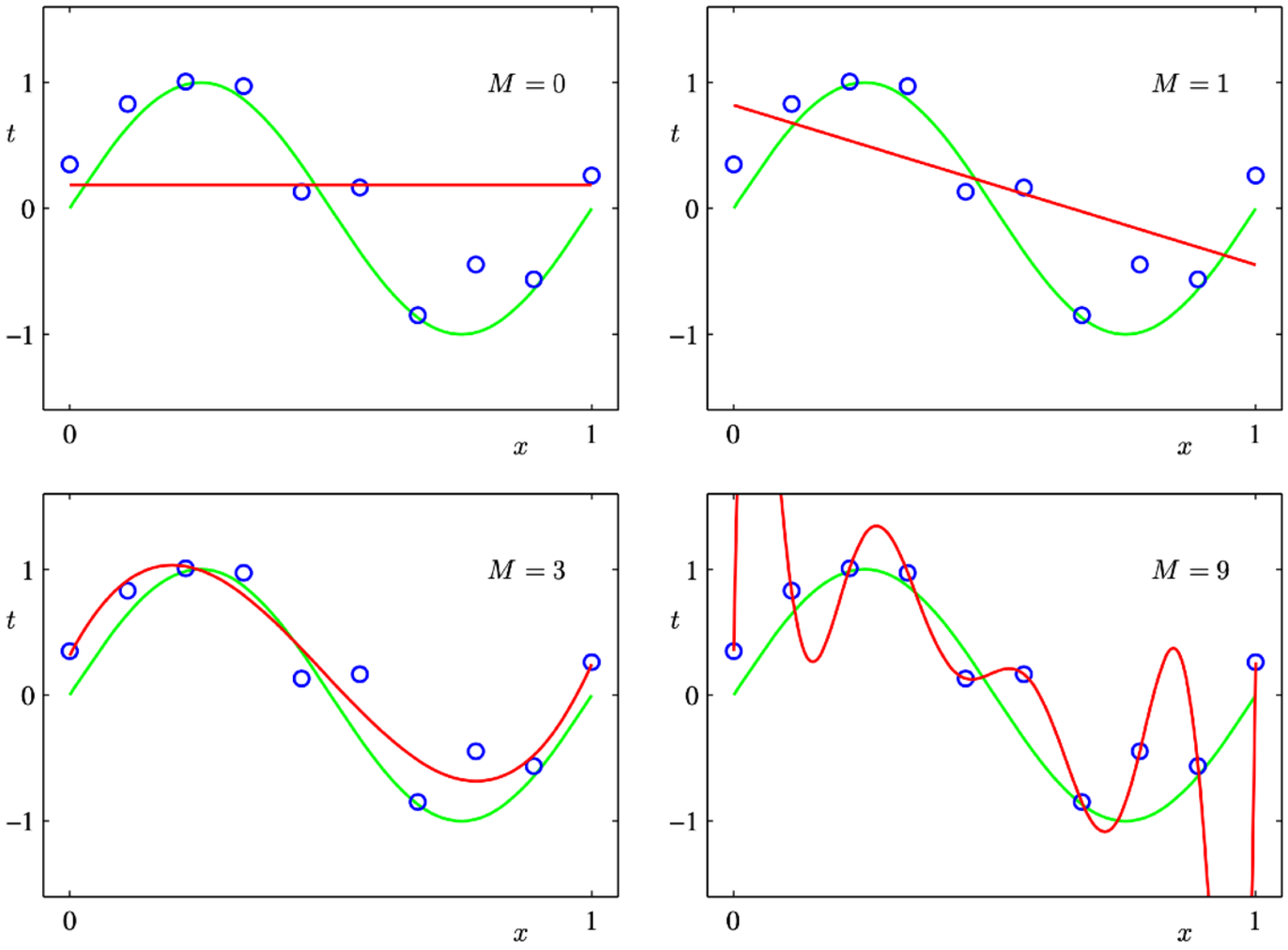
가중치를 표준편차가 $\frac{1}{\sqrt{n/2}}$ 인 정규분포로 초기화

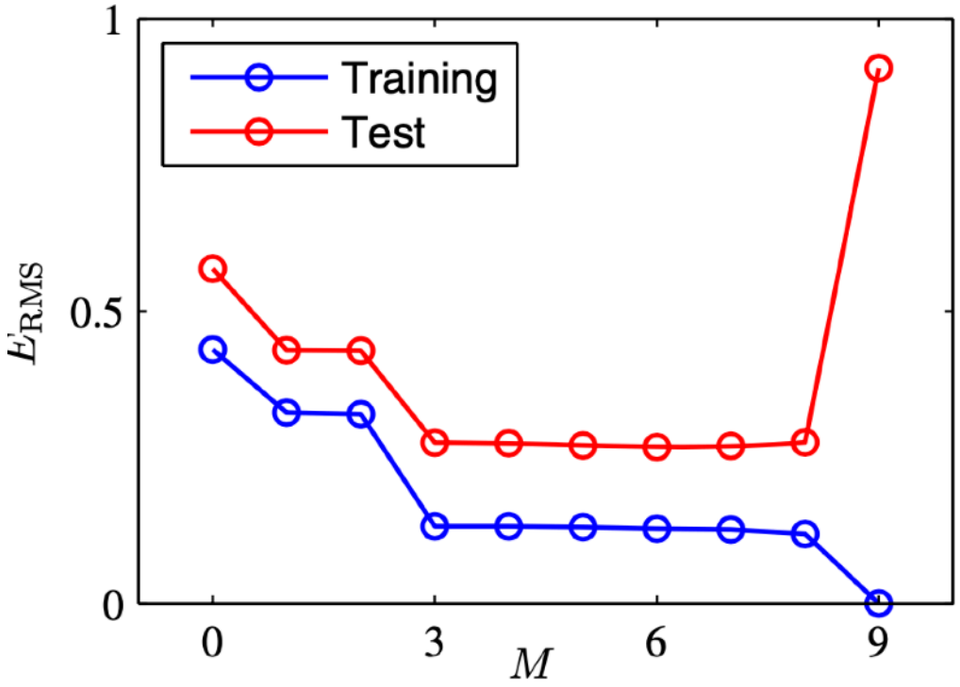


- **기울기 폭발 문제 (Exploding Gradient)**

1. 기울기 업데이트가 계속적으로 커져서, NaN 문제를 발생하는 경우.
2. 오버피팅 (Overfitting) 을 방지하는 목적

$$y(x, \omega) = \omega_0 + \omega_1 x + \omega_2 x^2 + \cdots + \omega_M x^M$$





	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

- **기울기 폭발 문제 (Exploding Gradient) : 가중치 감소 (weight decay)**

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}_i + \frac{\lambda}{2m}\sum_{l=1}^{L} \parallel \omega^{[l]} \parallel_F^2 \quad \equiv J_0 + \Omega_F$$

λ : regularization parameter 가 클수록, ω 를 작게 찾음.

$$\parallel \omega^{[l]} \parallel_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}\left(\omega_{ij}^{[l]}\right)^2$$ **(Frobenius sum)**

가중치 업데이트:

$$dW^{[n]} = \frac{1}{m} dZ^{[n]} A^{[n-1]t} \rightarrow ?$$

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}_i + \frac{\lambda}{2m}\sum_{l=1}^{L} \parallel \omega^{[l]} \parallel_F^2 \quad \equiv J_0 + \Omega_F$$

$$\parallel \omega^{[l]} \parallel_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}}\left(\omega_{ij}^{[l]}\right)^2 \quad \textbf{(Frobenius sum)}$$

가중치 업데이트:

$$dW^{[n]} = \frac{dJ}{dW^{[n]}} = \frac{dJ}{dZ^{[n]}}\frac{dZ^{[n]}}{dW^{[n]}} = \frac{dJ_0}{dZ^{[n]}}\frac{dZ^{[n]}}{dW^{[n]}} + \frac{d\Omega_F}{dW^{[n]}}$$

$$\frac{d\Omega_F}{dW^{[n]}} : \frac{d\Omega_F}{d\omega_{ij}^{[n]}} = \frac{d}{d\omega_{ij}^{[n]}}\left[\frac{\lambda}{2m}\sum_{n}\sum_{i,j}\left(\omega_{ij}^{[n]}\right)^2\right] = \frac{\lambda}{m}\omega_{ij}^{[n]}$$

$$dW^{[n]} = \frac{dJ_0}{dZ^{[n]}}\frac{dZ^{[n]}}{dW^{[n]}} + \frac{d\Omega_F}{dW^{[n]}} = \frac{1}{m}dZ^{[n]}A^{[n-1]t} + \frac{\lambda}{m}W^{[n]}$$

- **Type of "Regularization"**

L2 - regularization:

$$J = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}_i + \frac{\lambda}{2m}\sum_{l=1}^{L} \| \omega^{[l]} \|_F^2 \quad \equiv J_0 + \Omega_F$$

$$\| \omega^{[l]} \|_F^2 = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}} \left(\omega_{ij}^{[l]}\right)^2$$ **(Frobenius sum)**

L1 - regularization:

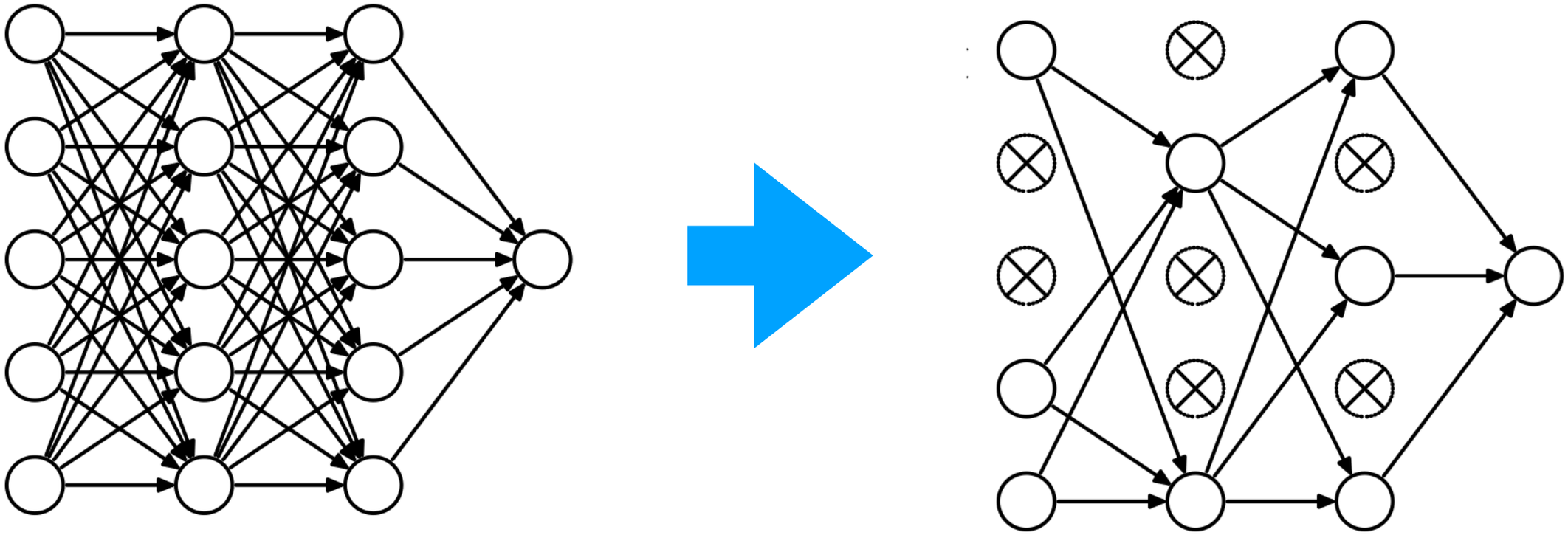
$$J = \frac{1}{m}\sum_{i=1}^{m}\mathscr{L}_i + \frac{\lambda}{2m}\sum_{l=1}^{L} | \omega^{[l]} |$$

가중치 업데이트:

$$| \omega^{[l]} | = \sum_{i=1}^{n^{[l]}}\sum_{j=1}^{n^{[l-1]}} | \omega_{ij}^{[l]} |$$

$$dW^{[n]} = \frac{1}{m} dZ^{[n]} A^{[n-1]t} \rightarrow ?$$ **(확인해 볼 것)**

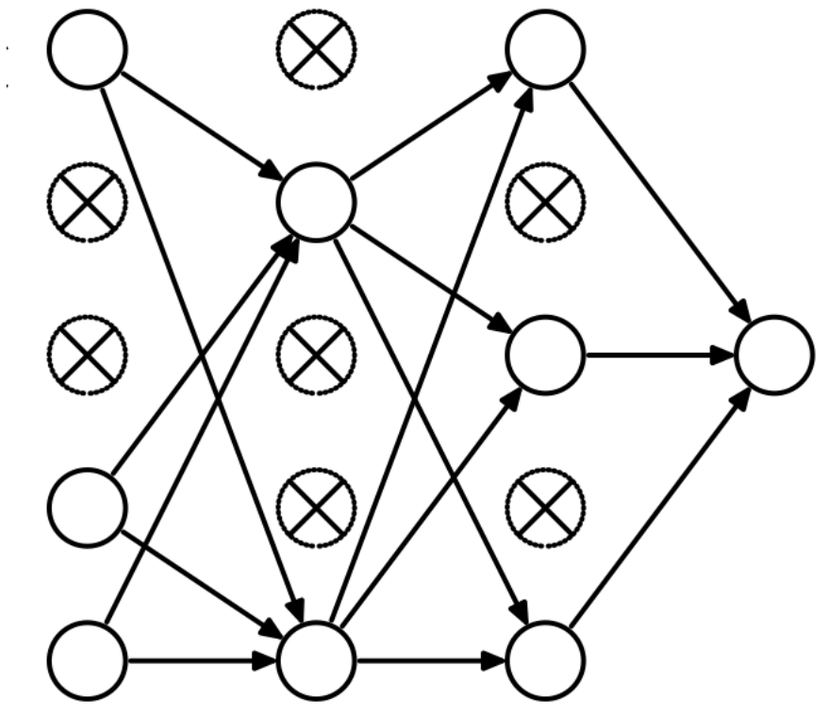
- **드롭 아웃 (Dropout) : 과적합을 피하는 방법**

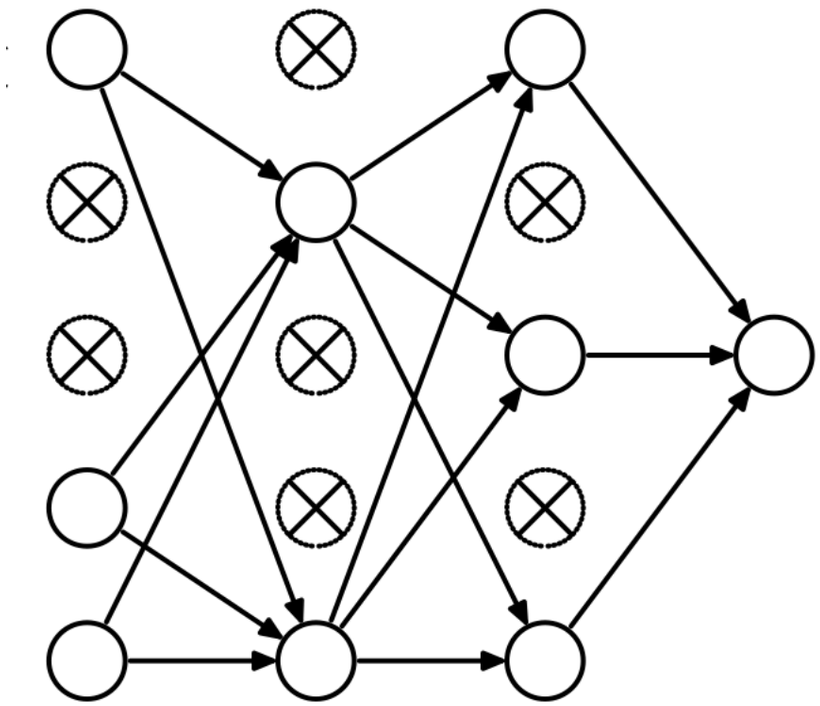


- 임의로 노드를 삭제하면서 학습하는 방법.
  훈련시 Hidden layer (은닉층)의 노드를 무작위로 삭제.

- 삭제하는 비율: Hyper-parameter (조절하는 변수) 임.

- **앙상블 학습** (Ensemble learning): 개별적으로 학습시킨 여러 모델의 출력을 평균내어 추론





$$A^{[n]} = A^{[n]} * D^{[n]}$$

$$A^{[n]} = A^{[n]} / \textbf{keep-prob}$$

$$dA^{[n]} = dA^{[n]} * D^{[n]}$$

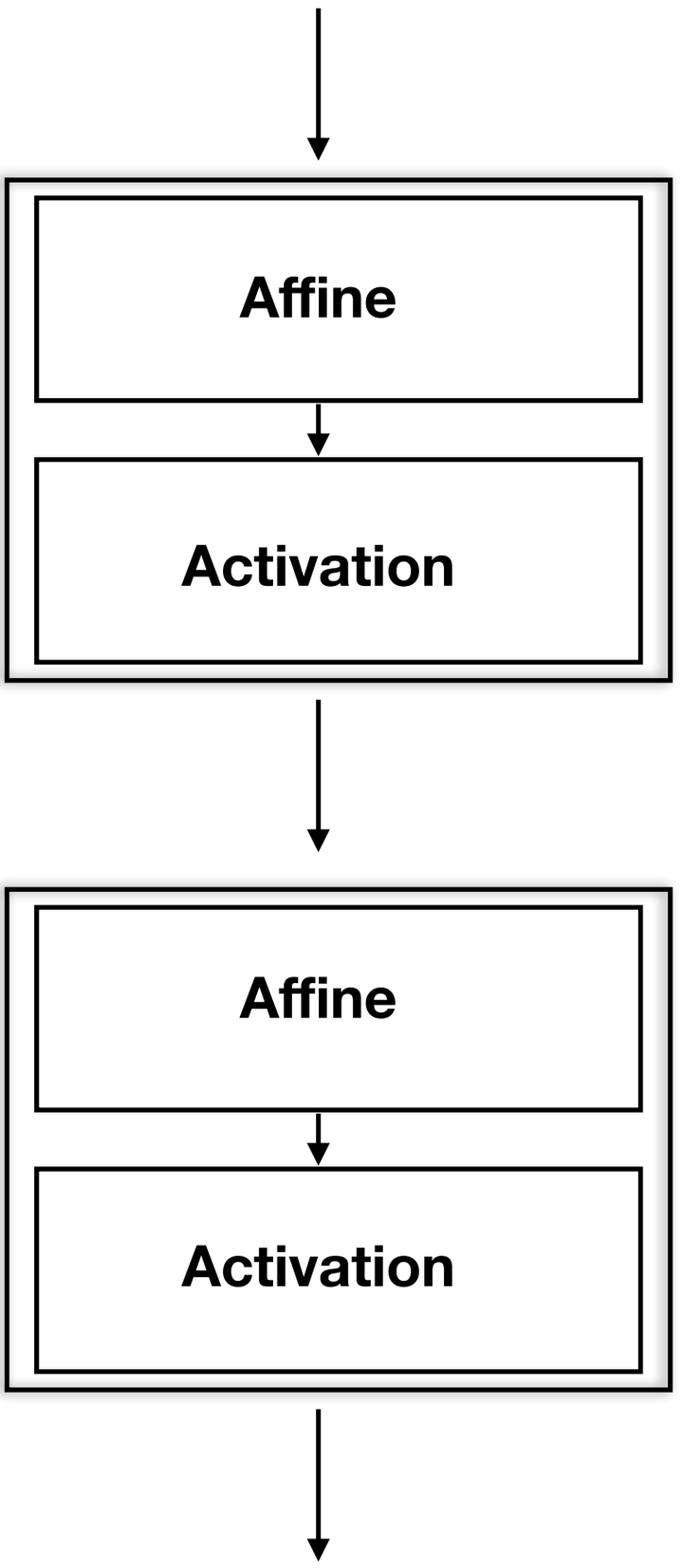
$$dA^{[n]} = dA^{[n]} / \textbf{keep-prob}$$

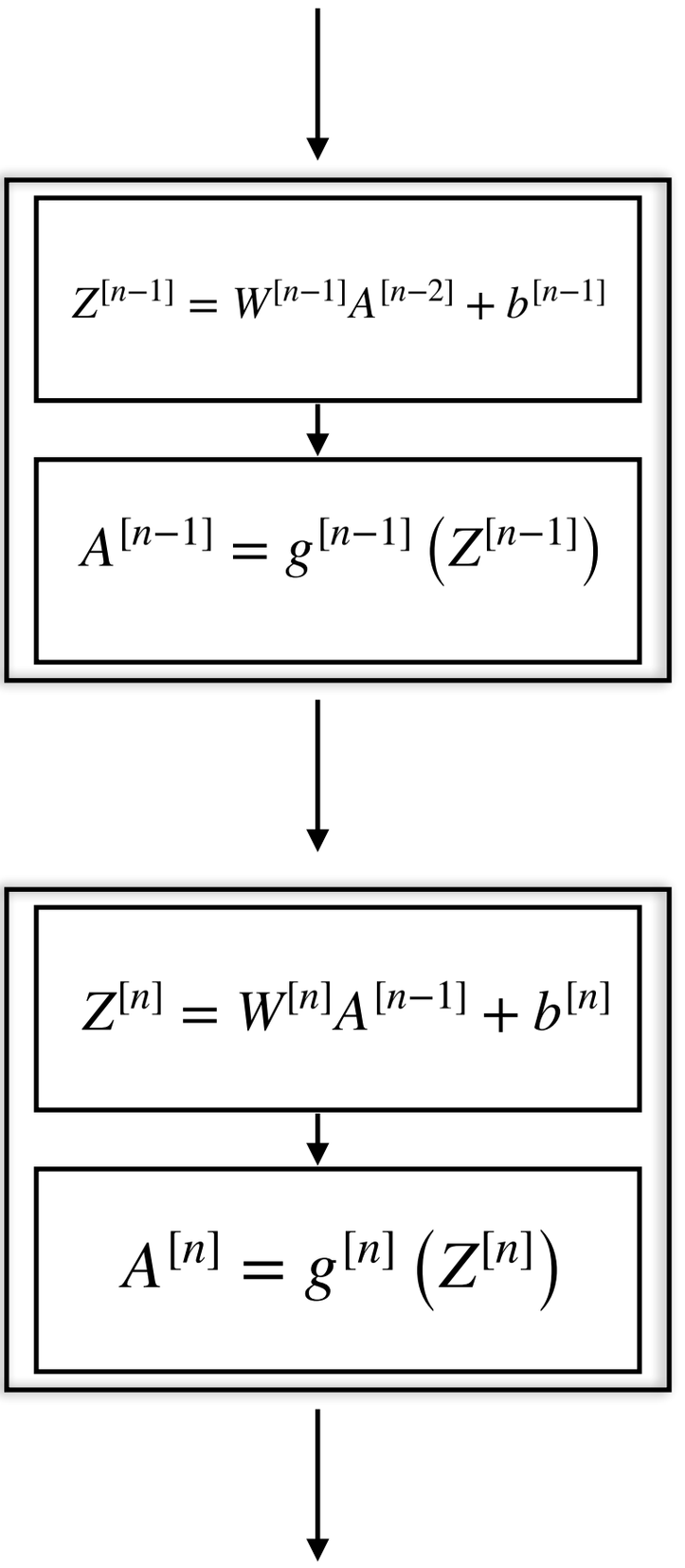
$$x' = \alpha x$$

$$A' = \alpha A$$

$$\frac{dy}{dx'} = \frac{dy}{dx}\frac{dx'}{dx} = \alpha\frac{dy}{dx}$$

$$dA' = \alpha(dA)$$

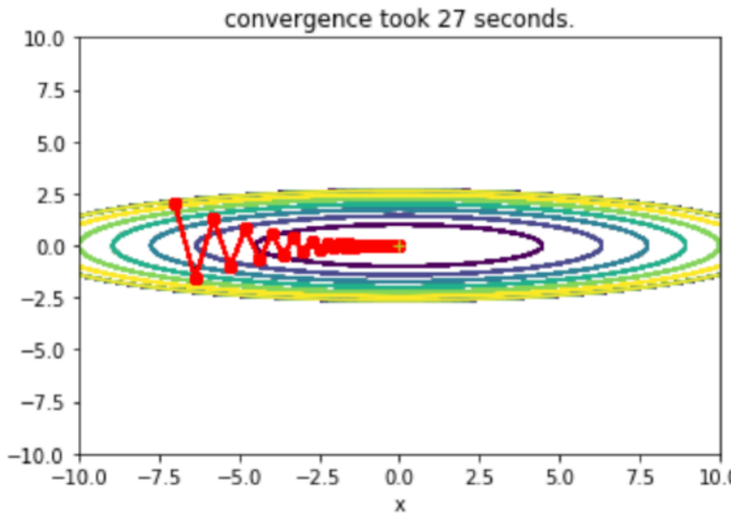

Affine
Activation
Affine
Activation


$Z^{[n-1]} = W^{[n-1]}A^{[n-2]} + b^{[n-1]}$
$A^{[n-1]} = g^{[n-1]}\left(Z^{[n-1]}\right)$
$Z^{[n]} = W^{[n]}A^{[n-1]} + b^{[n]}$
$A^{[n]} = g^{[n]}\left(Z^{[n]}\right)$

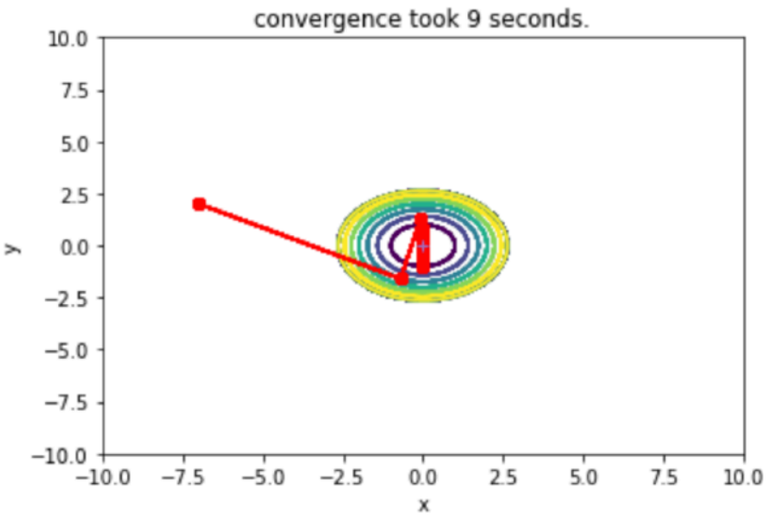
- **배치 정규화 (Batch - Normalization)**

각 층의 활성화 값을 적절히 분포시키는 것

un-Normalized



Normalized



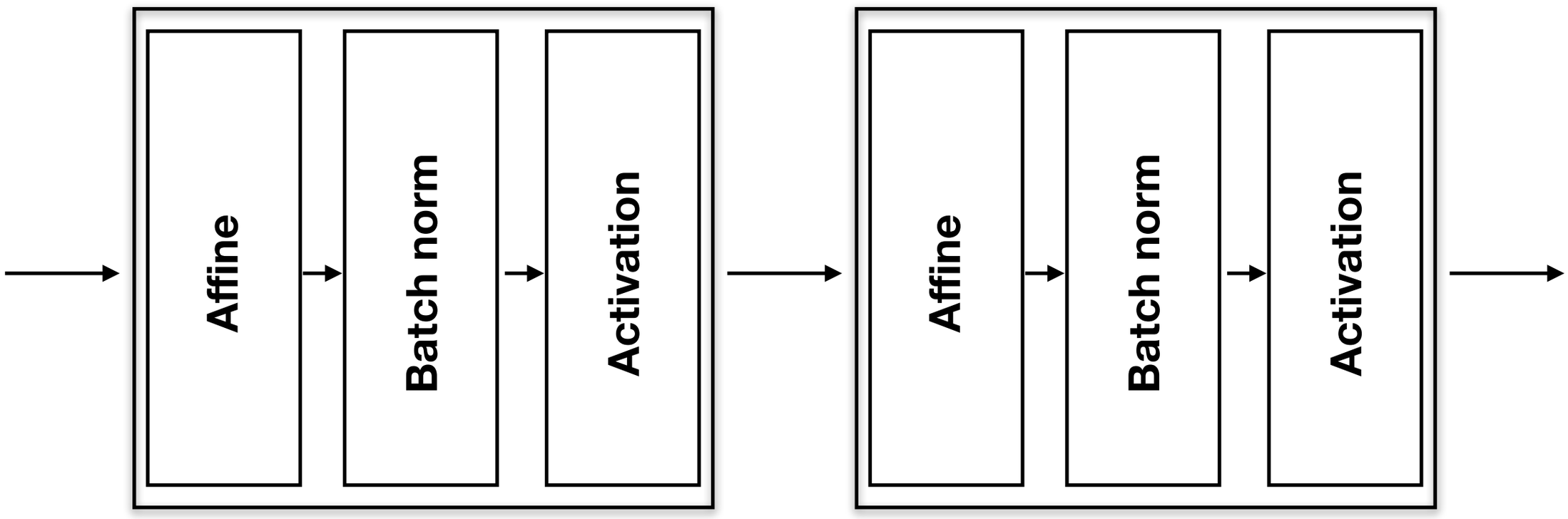
Batch norm

$$\mu^{[n]} = \frac{1}{m}\sum_{i=1}^{m} z^{[n](i)}$$

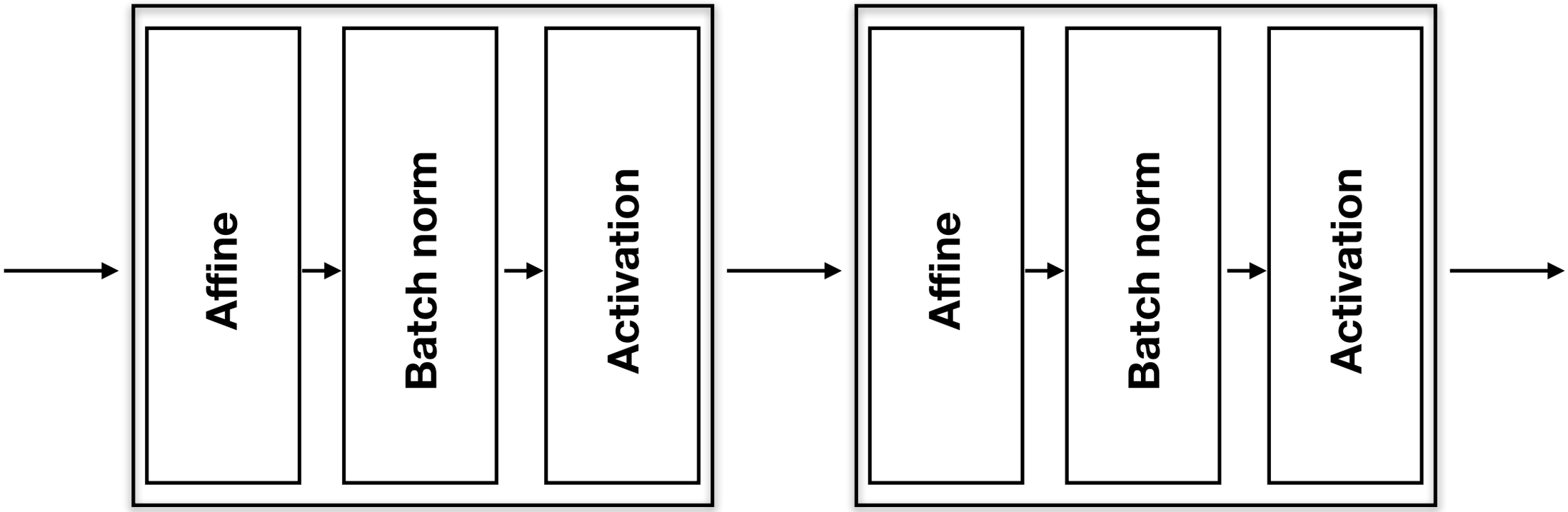
$$(\sigma^2)^{[n]} = \frac{1}{m}\sum_{i=1}^{m}\left(z^{[n](i)} - \mu^{[n]}\right)^2$$

$$Z^{[n]}_{norm} = \frac{Z^{[n]} - \mu^{[n]}}{\sqrt{(\sigma^2)^{[n]} + \epsilon}}$$

$$\tilde{Z}^{[n]} = \gamma^{[n]} Z^{[n]}_{norm} + \beta^{[n]}$$



- **배치 정규화 (Batch - Normalization)**



- 학습 변수: Affine (ω, b) 및 Batch norm에서 (γ, β)

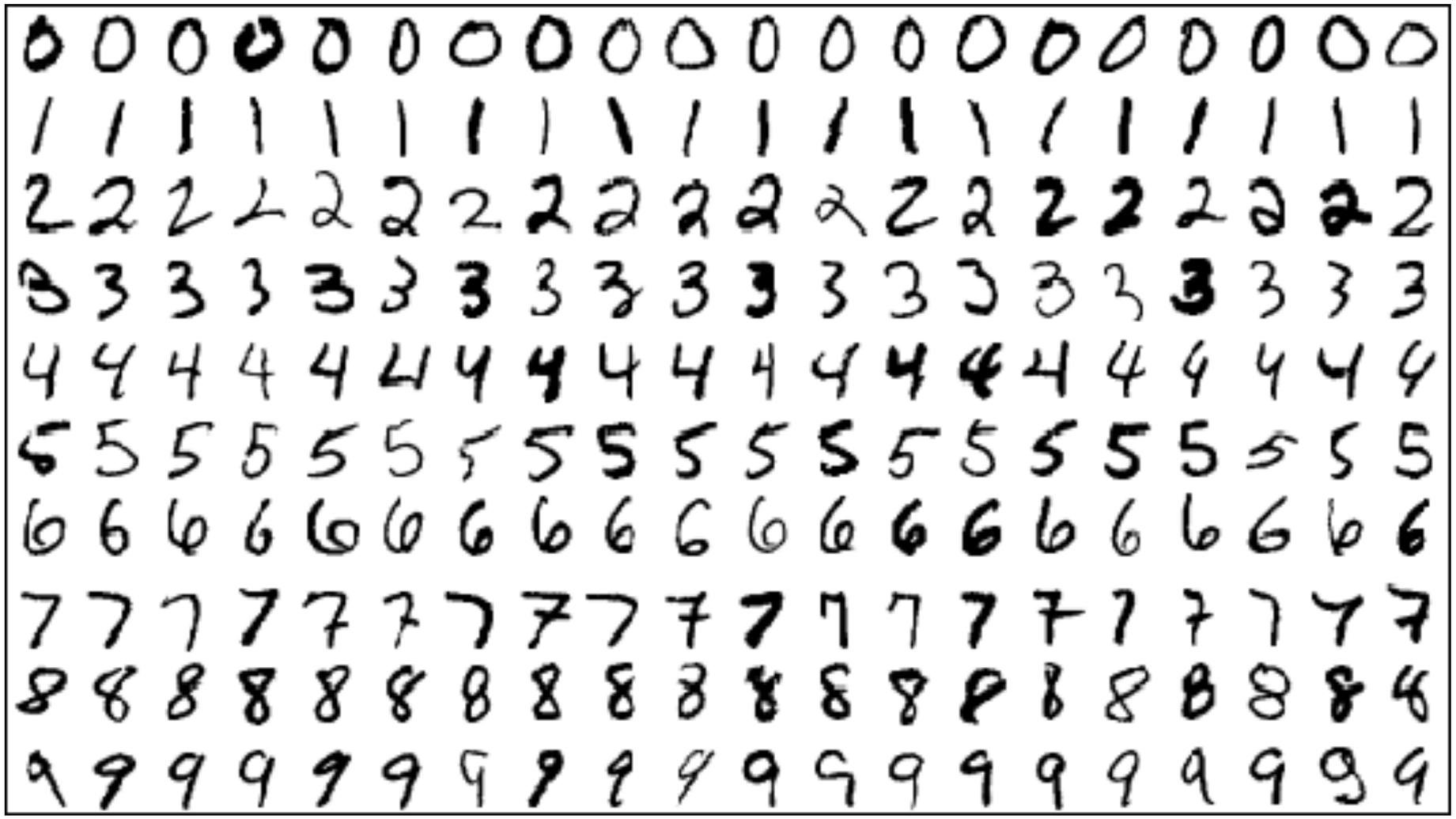
$$W \leftarrow W - \alpha(dW)$$

$$b \leftarrow b - \alpha(db)$$

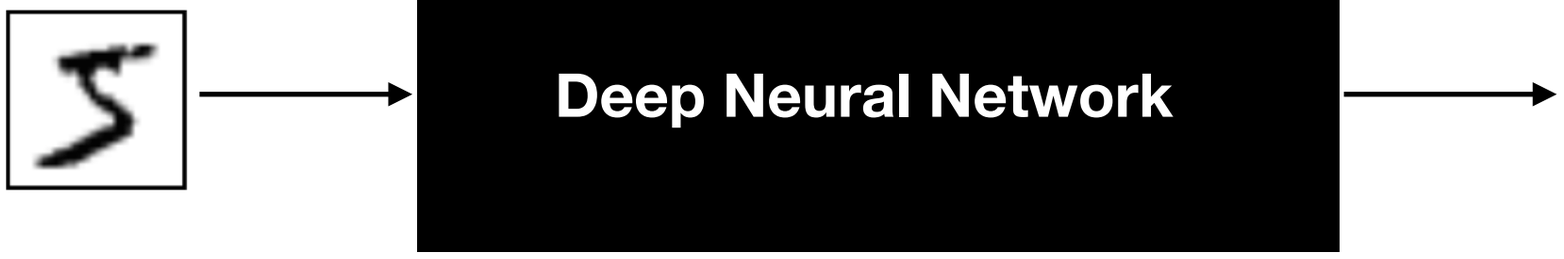
$$\gamma \leftarrow \gamma - \alpha(d\gamma)$$

$$\beta \leftarrow \beta - \alpha(d\beta)$$

- **여러개를 구분하는 경우**

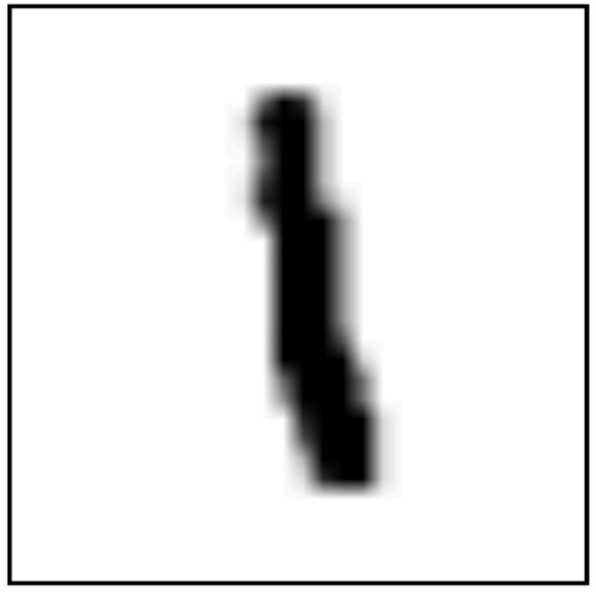


데이터로 supervised learning



확률

0	0.1
1	0.0
2	0.0
3	0.0
4	0.04
5	0.8
6	0.05
7	0.0
8	0.0
9	0.01

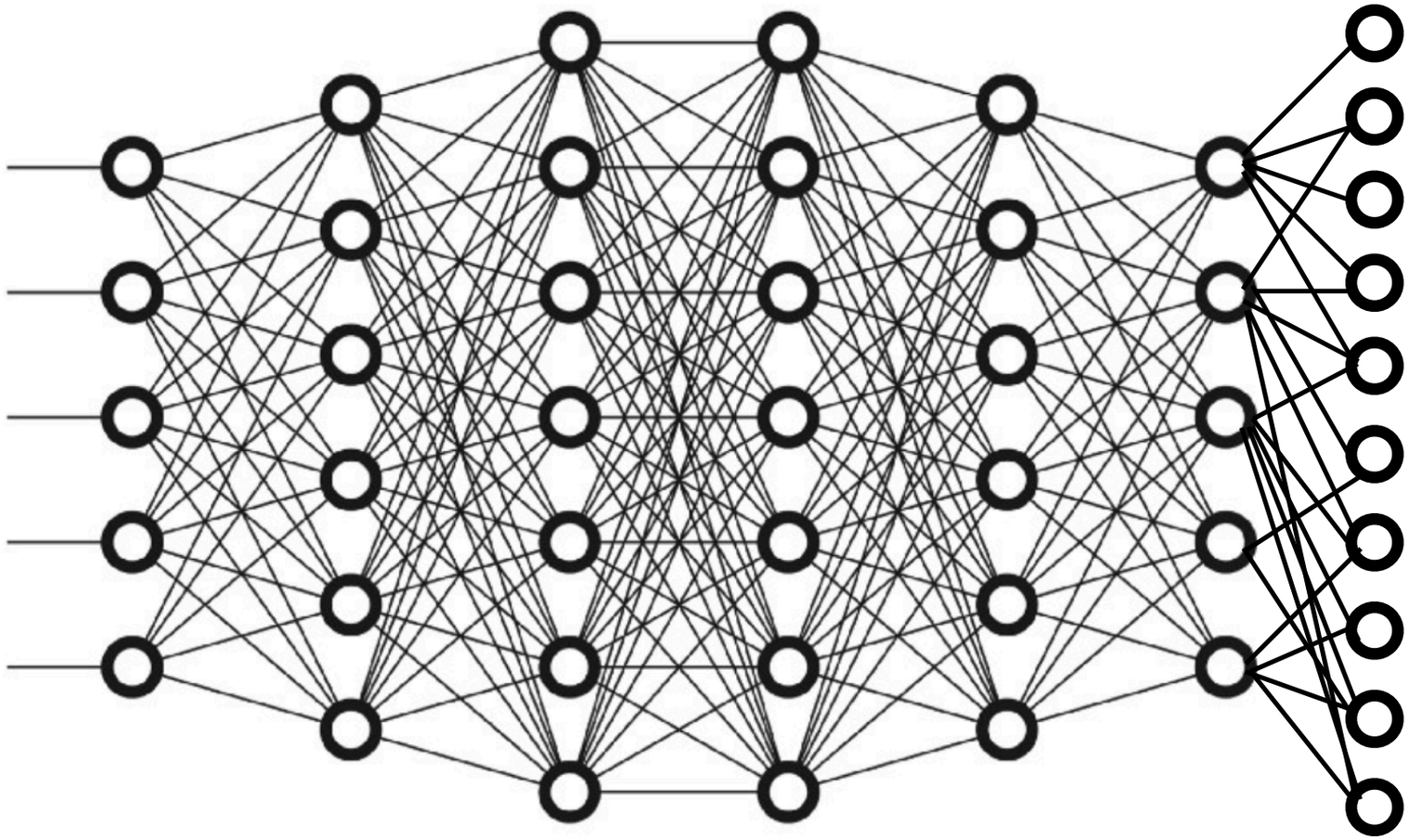


$$\approx \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

$$28 \times 28 = 784$$

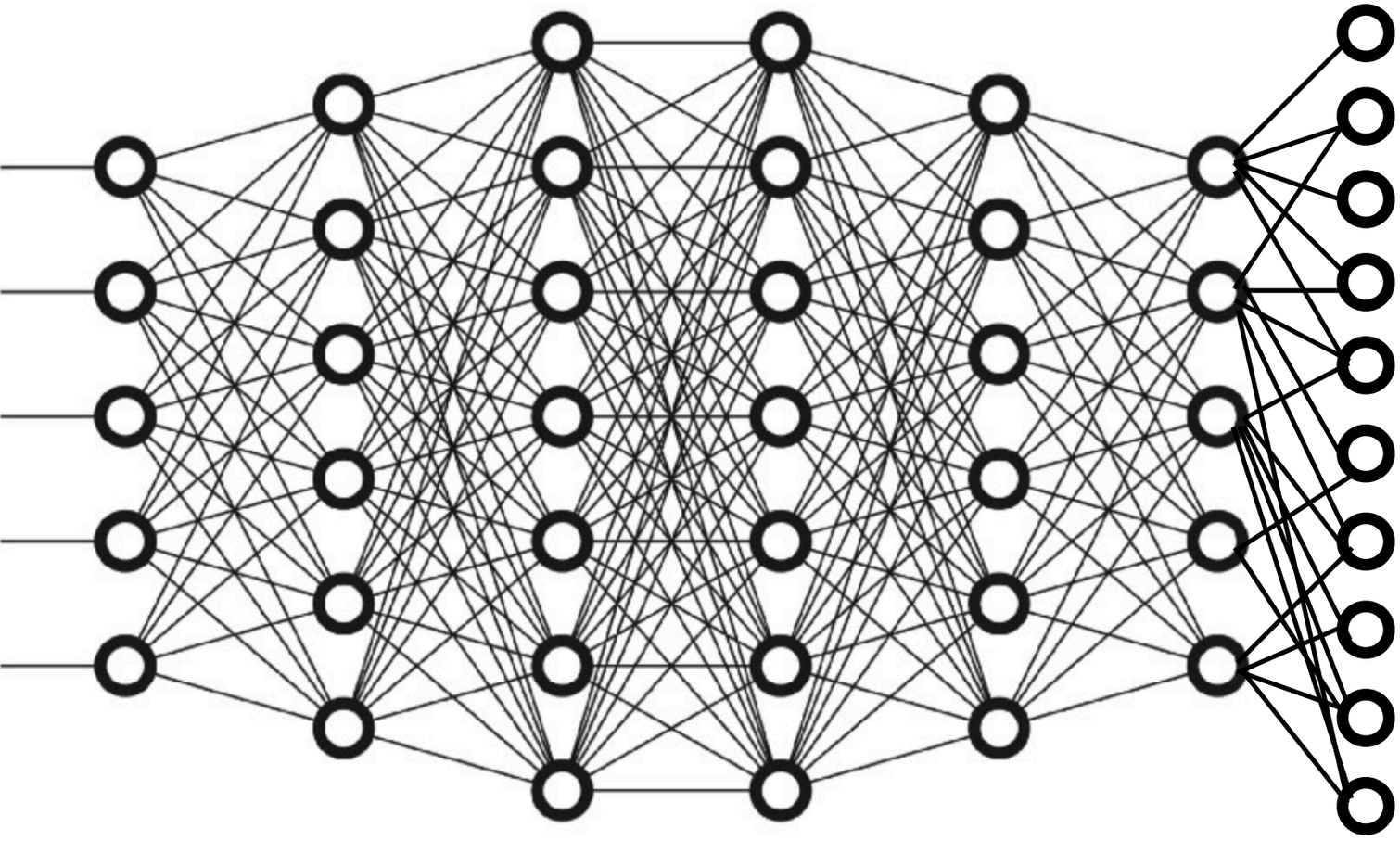
784

features (nodes)



10

output

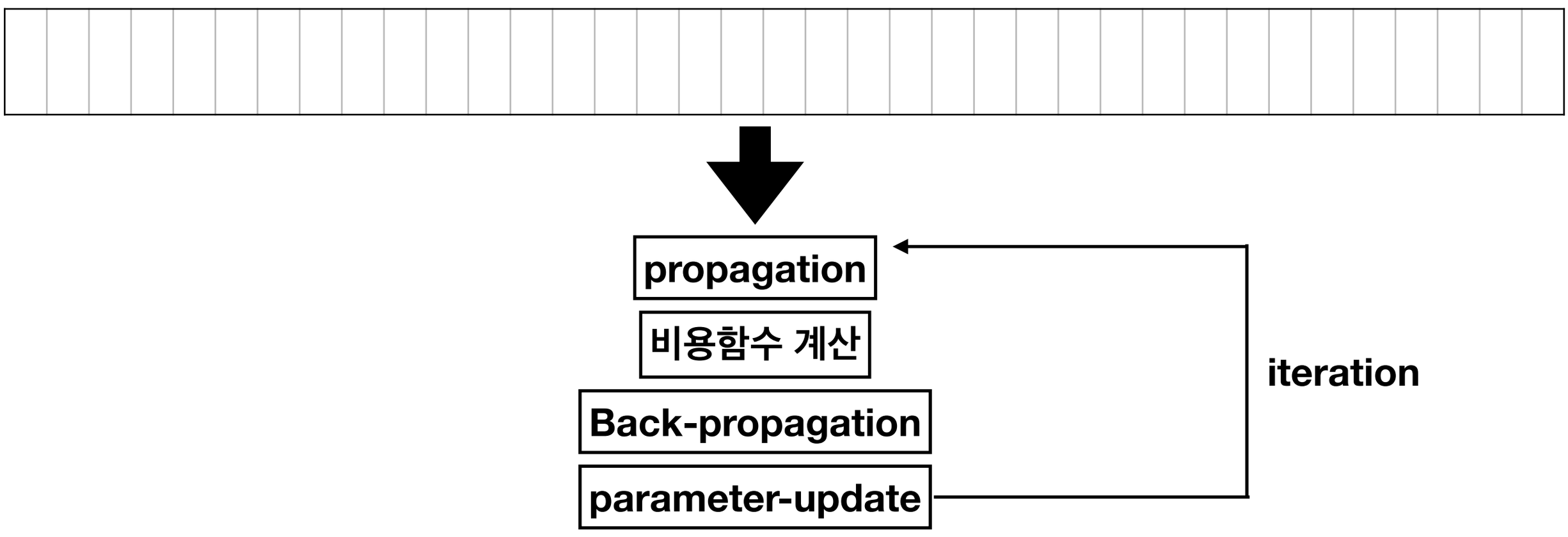


$$y_k = \frac{exp(z_k)}{\sum_{i=0}^{9} exp(z_i)}$$

$$\mathscr{L} = -\sum_{i=0}^{9} t_i \log y_i$$

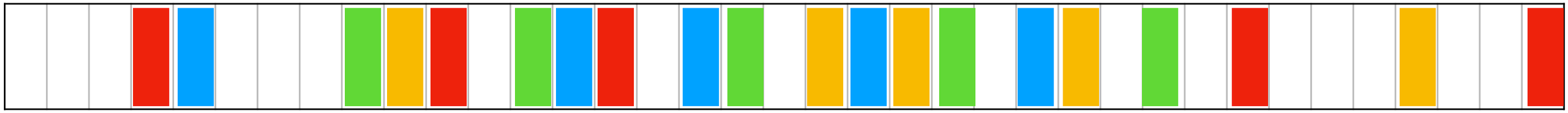
- **데이터 배치 학습하기**

메모리 및 속도등의 문제로, 학습 데이터 전체를 한번에 학습하는 경우, 속도 저하 문제 발생.

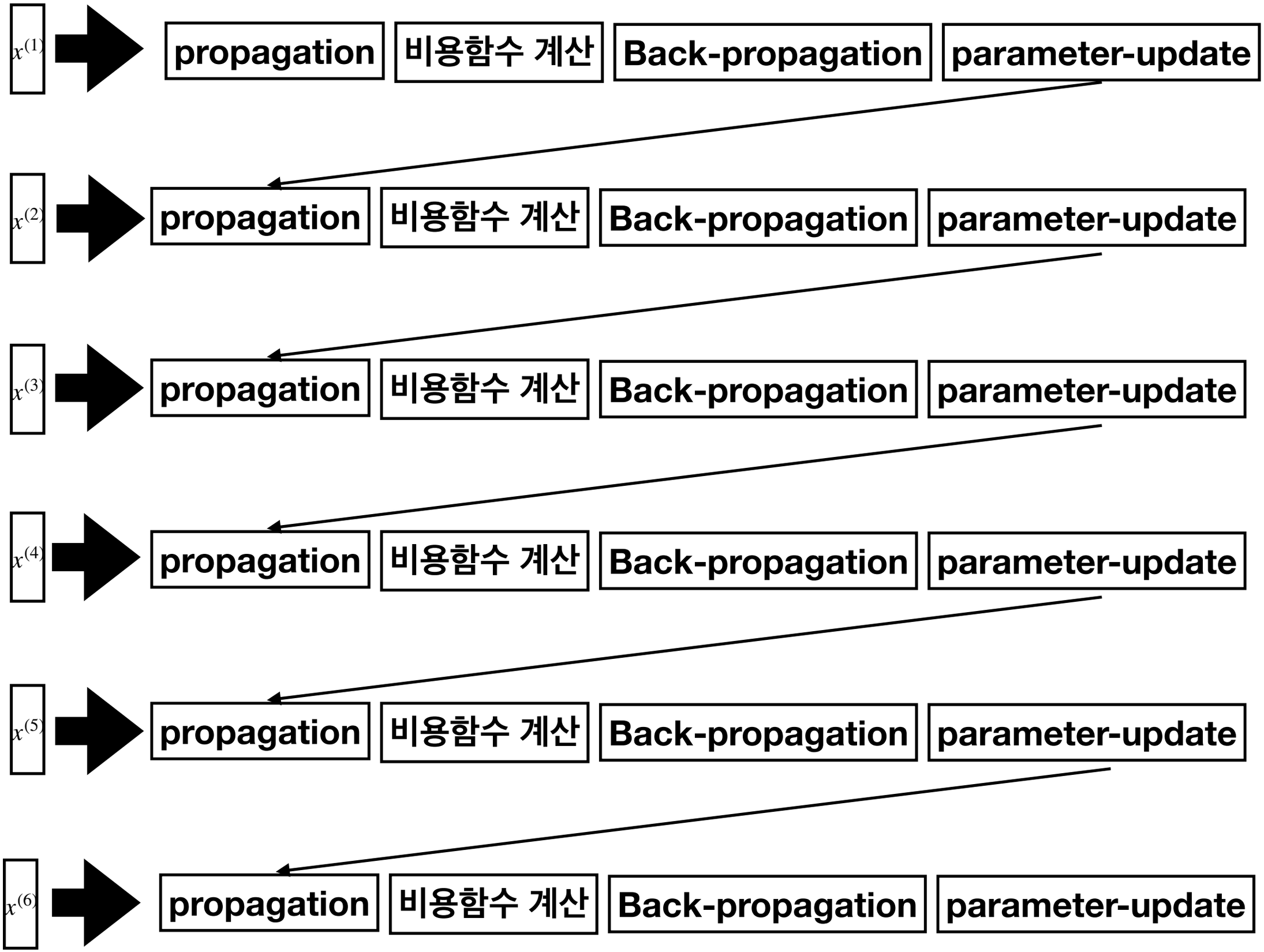


- **미니 배치 학습하기**

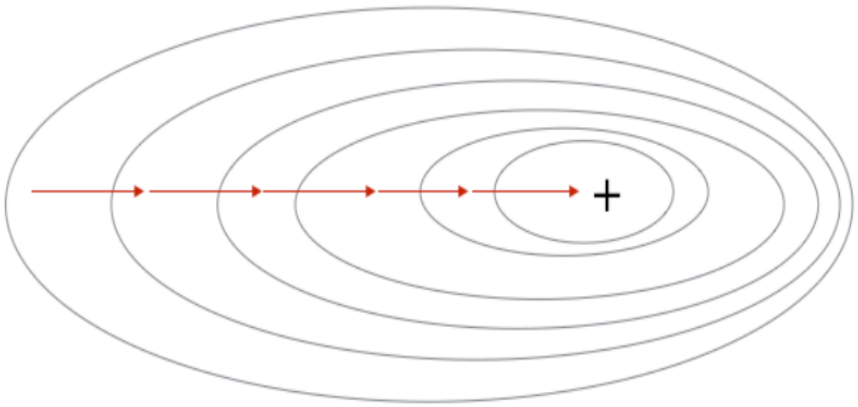
메모리 및 속도등의 문제로, 학습 데이터 전체를 한번에 학습하지 않고,
데이터 중 일부를 무작위로 꺼내어 그 미니배치에 대해 학습



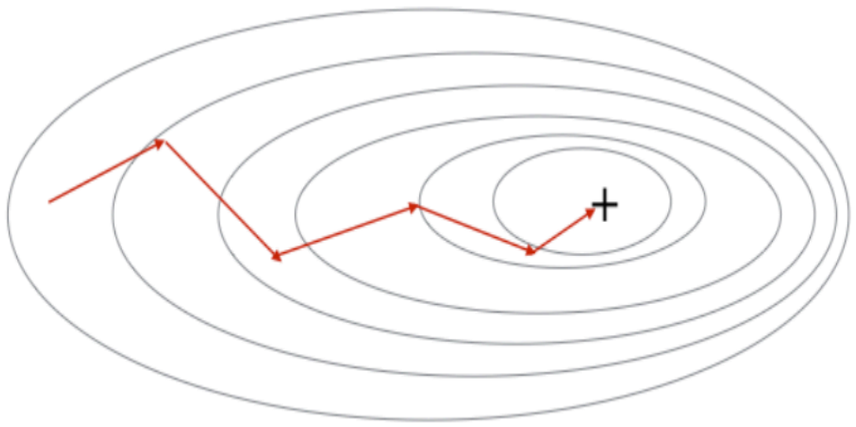
- **Stochastic method**



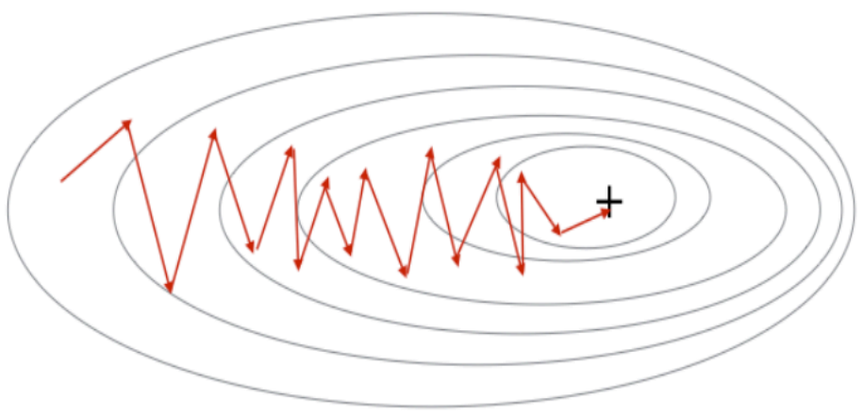
Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



- 에폭 (epoch): 1에폭 =
훈련 데이터를 모두 소진하였을 때의 횟수.
예) 100,000 개의 데이터가 있을 때, 미니배치 크기가 100인 경우. 1000회 반복 = 1 에폭