

# 딥러닝팀

1팀

정승민  
변석주  
이정환  
송승현  
최용원

# CONTENTS

---

1. 딥러닝

2. 퍼셉트론

3. 신경망

4. 성능 향상 기법

1

딥러닝

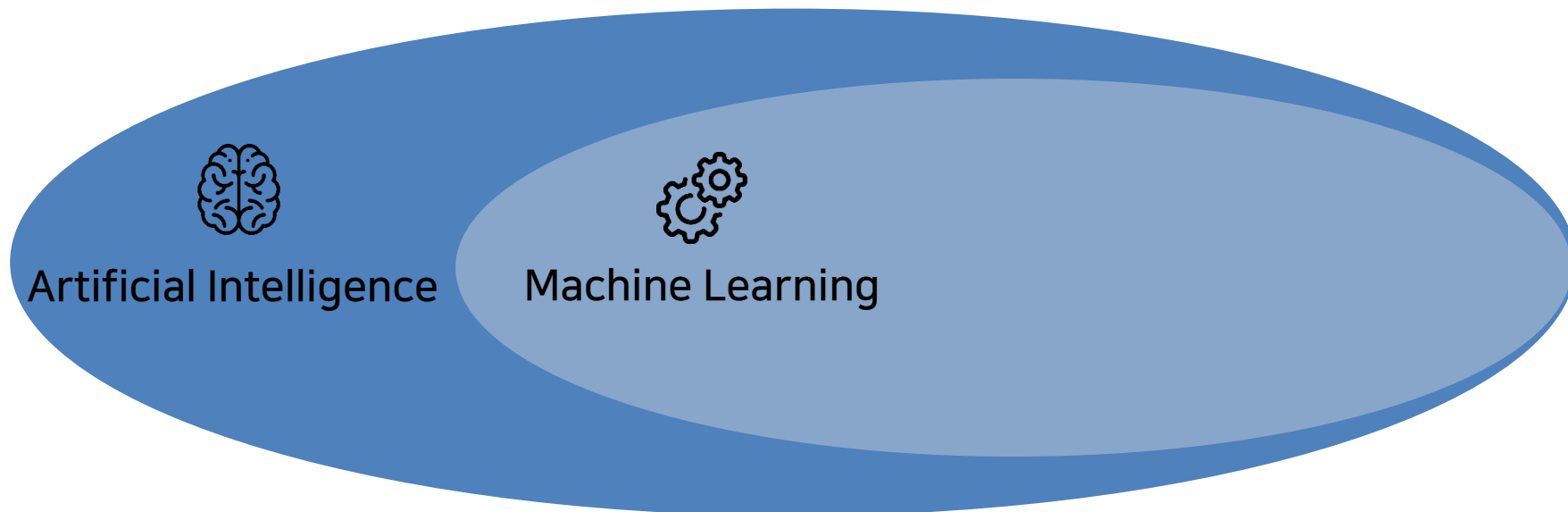
## 머신러닝 Machine Learning



Artificial Intelligence

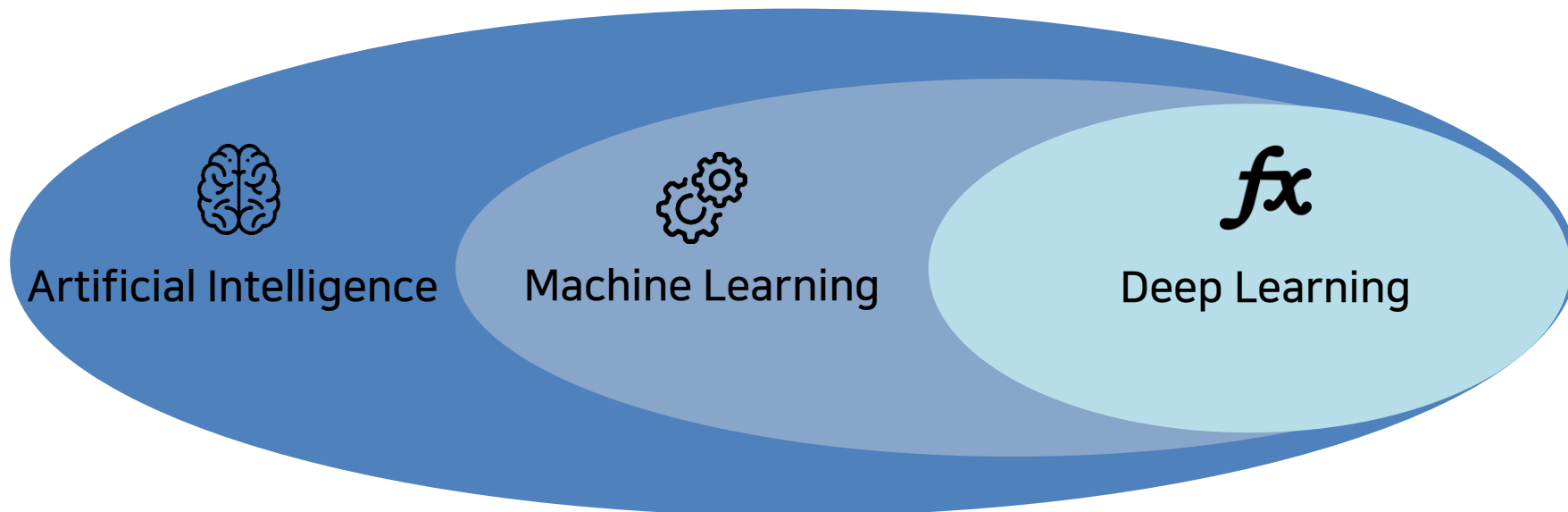
인간이 다양한 경험을 기반으로 새로운 행동 패턴과 지식을 습득하듯,  
컴퓨터가 데이터형태로 얻는 경험에서 **스스로 학습**하고 지식을 추론하는 것

## 머신러닝 Machine Learning



머신러닝은 지도/비지도와 강화학습으로 나눌 수 있으며,  
정확한 결과를 예측하도록 다양한 알고리즘을 통해 학습

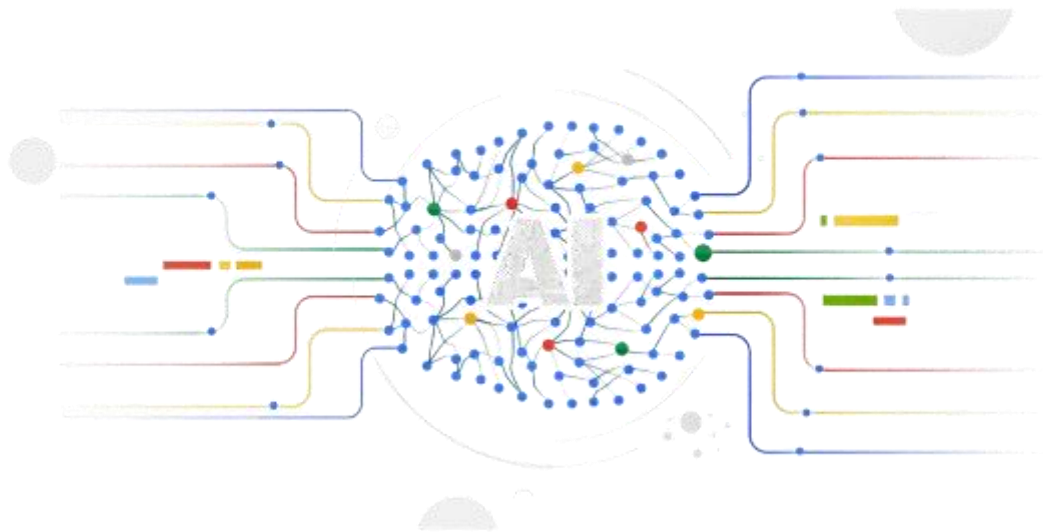
## 머신러닝 Machine Learning



머신러닝에서 발전된 형태로  
사람이 학습할 데이터를 입력하지 않아도 스스로 학습하고 예측

## 머신러닝의 한계

차원의 저주



비정형 데이터 적용의 한계  
고차원 공간에서 성능 하락

## 머신러닝의 한계

### 차원의 저주

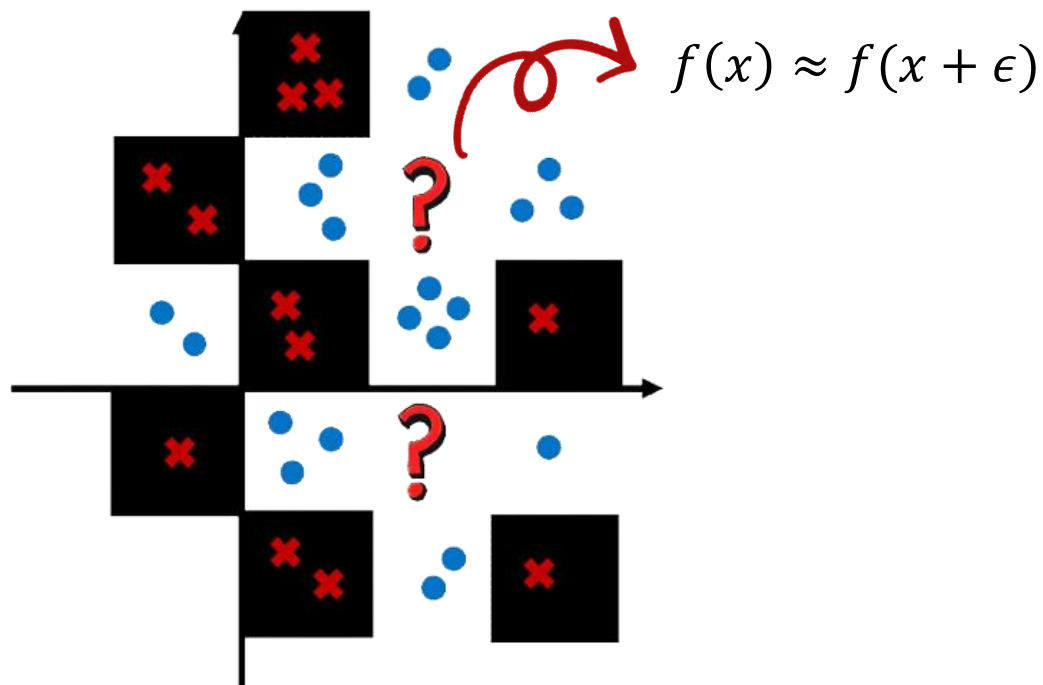


자료의 차원이 높을 때 해결하기 어려운 과제  
변수의 수 증가 → 구성 요소의 수 증가 (지수적)



## 머신러닝의 한계

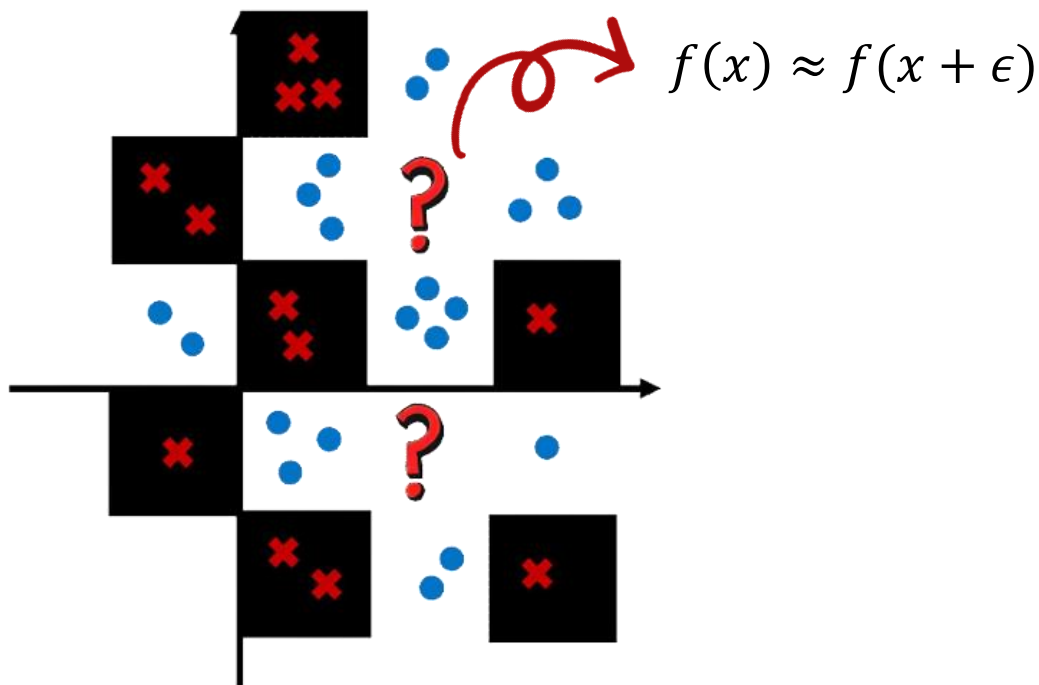
국소 일치성



함수가 작은 영역 안에서는 크게 변하면 안됨

## 머신러닝의 한계

국소 일치성

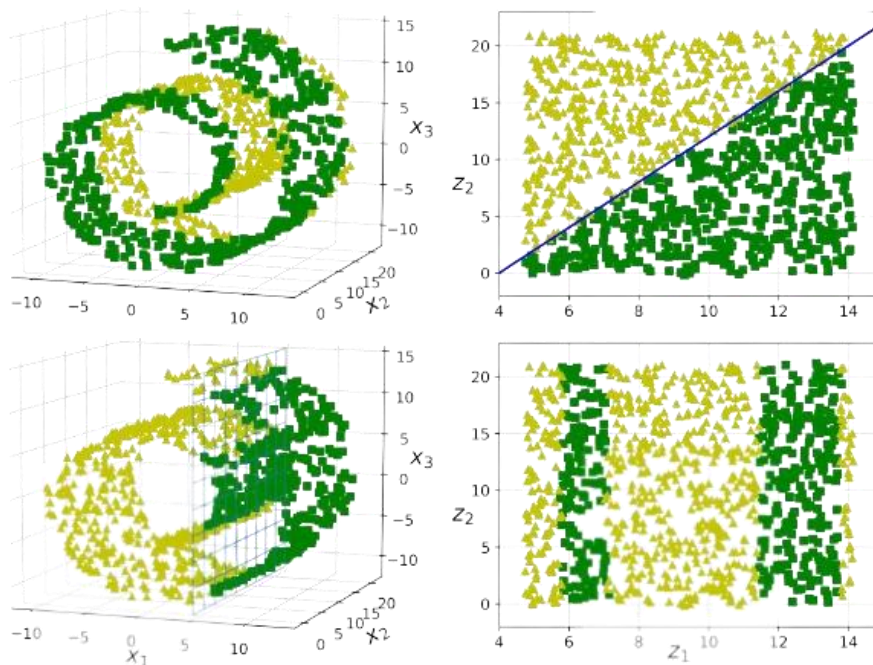


위 예시에서는 **주기함수** 사용시 해결 가능

But 일반적으로 해결 가능한 방법은 아님

## 머신러닝의 한계

다양체 가설



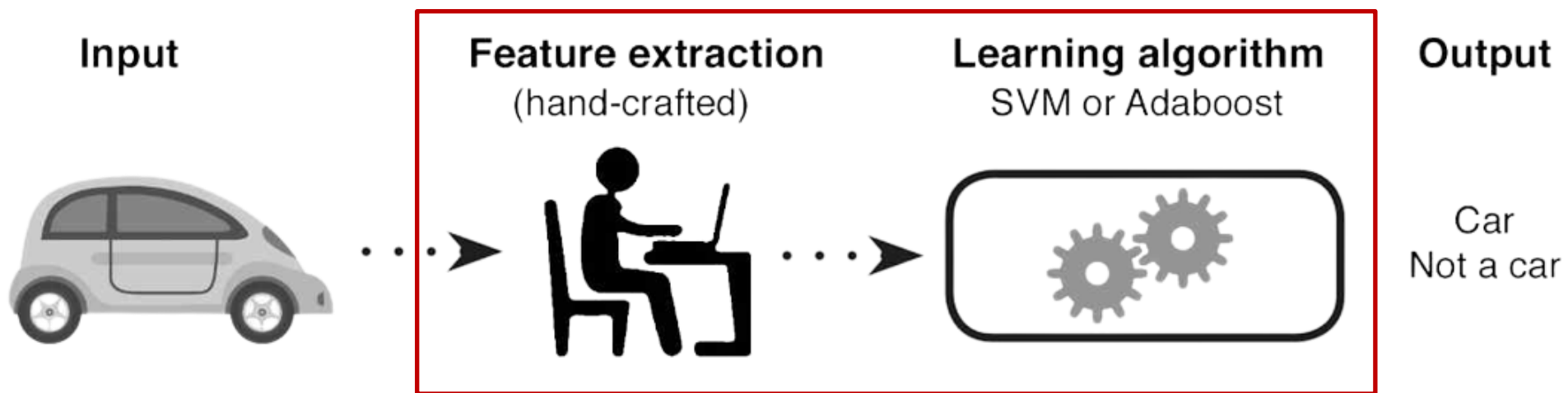
다양체: 국소적으로 유클리드 공간과 닮은 위상공간

고차원을 저차원에서 표현

텍스트, 이미지 데이터에서 두드러짐

## 머신러닝 VS 딥러닝

End-to-end learning



머신러닝은 Feature Extraction 과정에 따라  
결과가 천차만별로 달라짐

Input

Feature extraction - Classification

Output

## 머신러닝 VS 딥러닝

End-to-end learning

Input

Feature extraction  
(hand-crafted)

Learning algorithm  
SVM or Adaboost

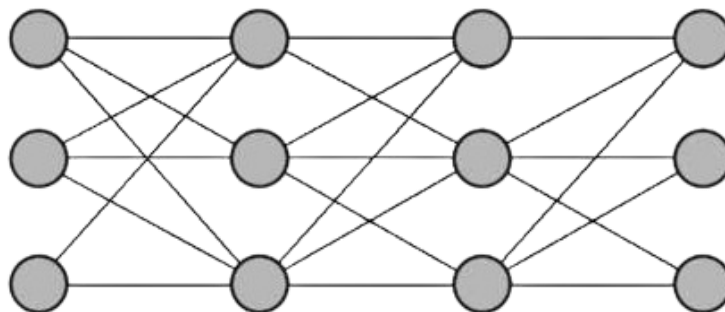
Output

딥러닝은 **Feature Engineering**의 중요성이 낮아,  
사람이 개입하지 않아도 된다는 강점을 가짐

Car  
Not a car



Input



Feature extraction - Classification



Car  
Not a car

Output

## 머신러닝 VS 딥러닝

딥러닝 활용 사례

지도학습



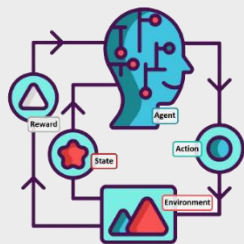
얼굴 인식  
질병 진단

비지도학습



오토인코더  
이미지 생성

강화학습



Deep Q-Network  
알파고

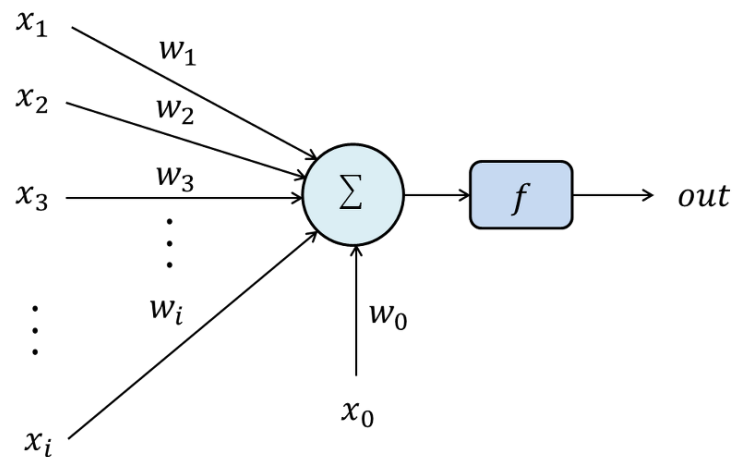
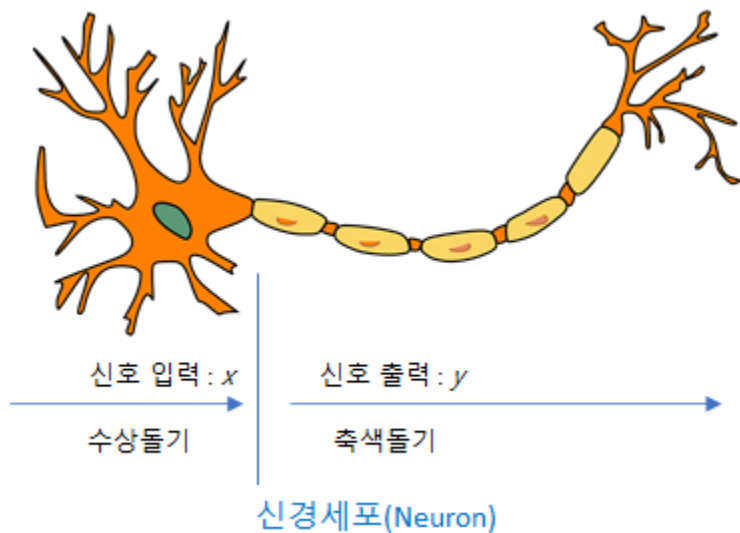
# 2

---

퍼셉트론

## 퍼셉트론

Perceptron

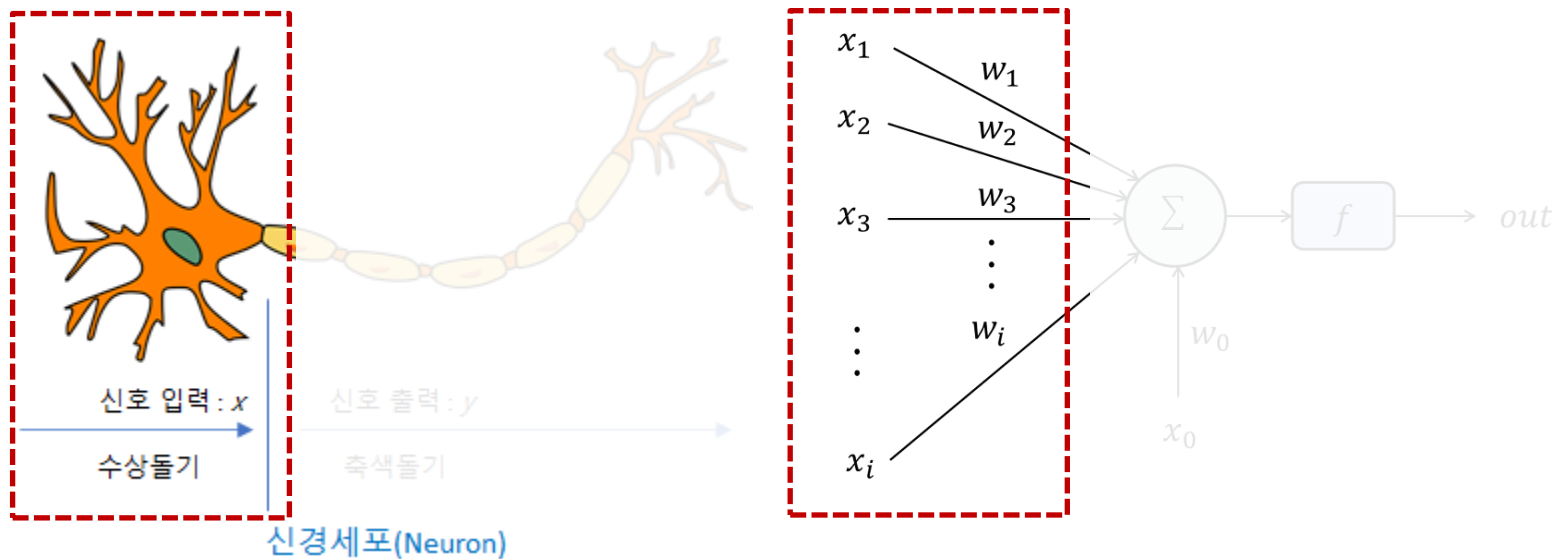


신경망의 **최소 단위**이자  
뉴런을 수학적으로 모델링한 것



## 퍼셉트론

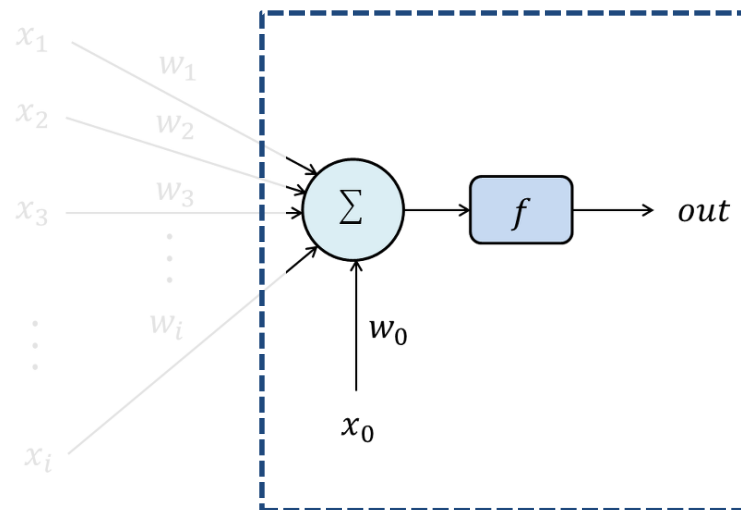
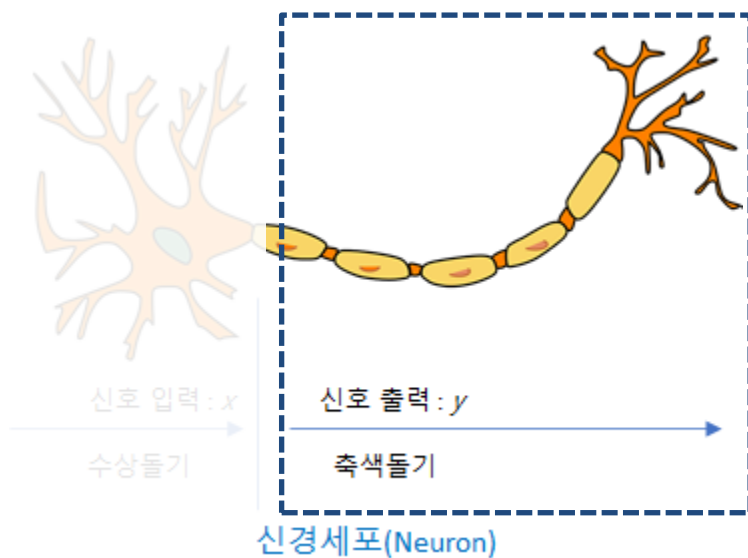
Perceptron



뉴런과 같이 다수의 입력에 대해 하나의 출력을 반환

## 퍼셉트론

Perceptron

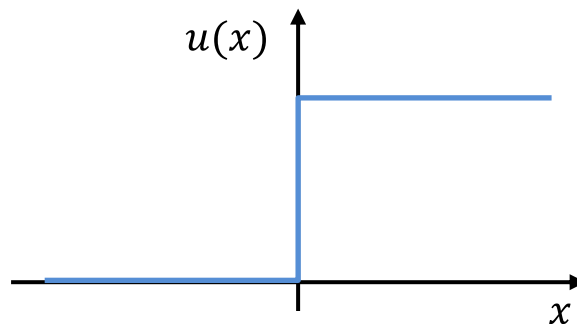


뉴런과 같이 **다수**의 입력에 대해 **하나**의 출력을 반환

## 퍼셉트론

Perceptron

$$g_w(x) = \begin{cases} 0 & \left( \sum_{i=1}^n w_i x_i + b \leq \theta \right) \\ 1 & \left( \sum_{i=1}^n w_i x_i + b \geq \theta \right) \end{cases}$$



가중합을 계산한 후 계단함수를 통해  
0 혹은 1의 출력값을 반환

## 퍼셉트론

동작 알고리즘

$$w_{j+1} := w_j + a(y^{(i)} - g_w(x^{(i)}))x_j^{(i)}$$

$(x^{(i)}, y^{(i)})$ :  $x$ 와  $y$ 의  $i$ 번째 관측치

$x_j^{(i)}$ :  $i$ 번째 관측치  $x$  중  $j$ 번째  $x$

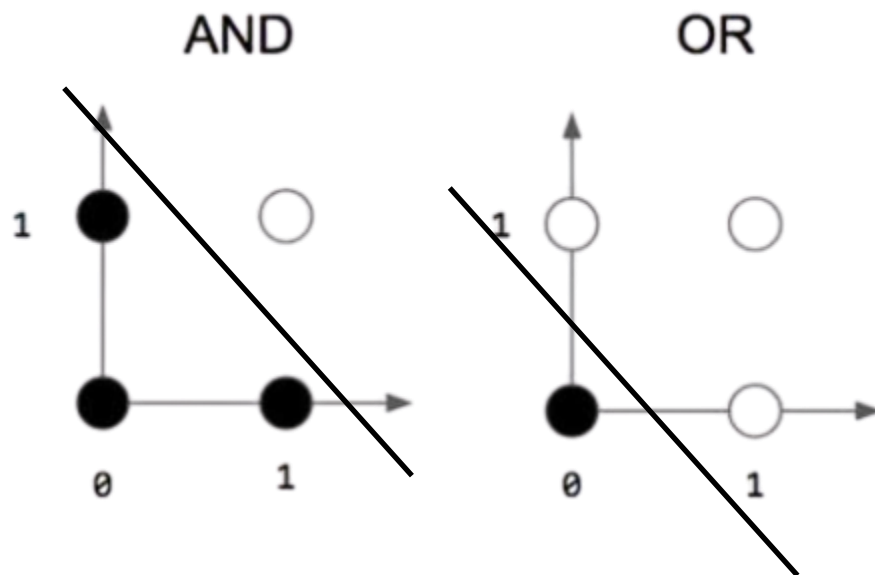
$w_j$ :  $x_j$ 와 곱해지는 가중치

$y^{(i)} - g_w(x^{(i)})$ : 오차

1.  $x_i$ 를  $g_w(x)$ 에 대입해 예측값 도출
2.  $w_j$  업데이트 ( $w_{j+1} = \text{예측값} - \text{실제 } y^{(i)} + w_j$ )
3. 각 관측치마다 동일 과정 반복

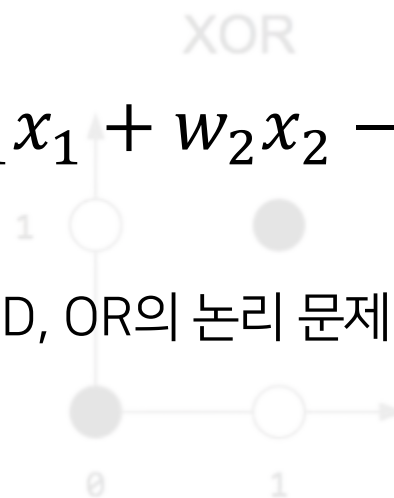
## 퍼셉트론

논리 연산과 한계



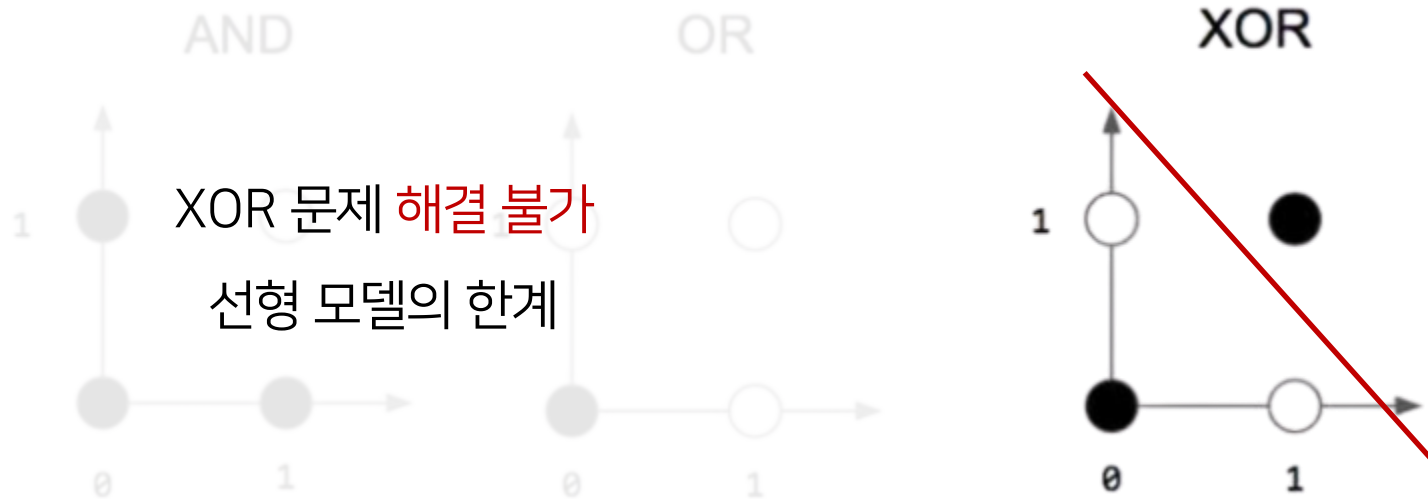
$$w_1x_1 + w_2x_2 - b = 0$$

AND, OR의 논리 문제 해결 가능



## 퍼셉트론

논리 연산과 한계



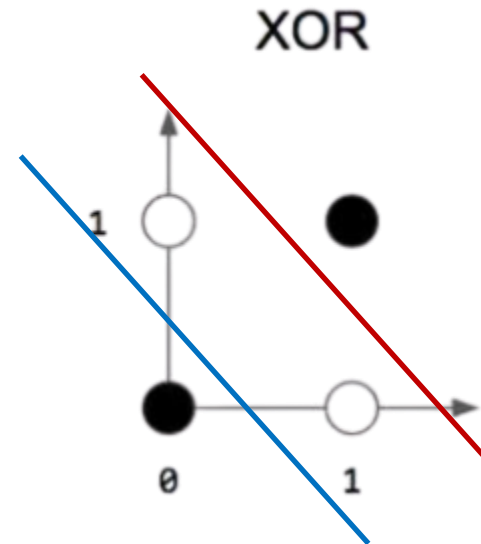
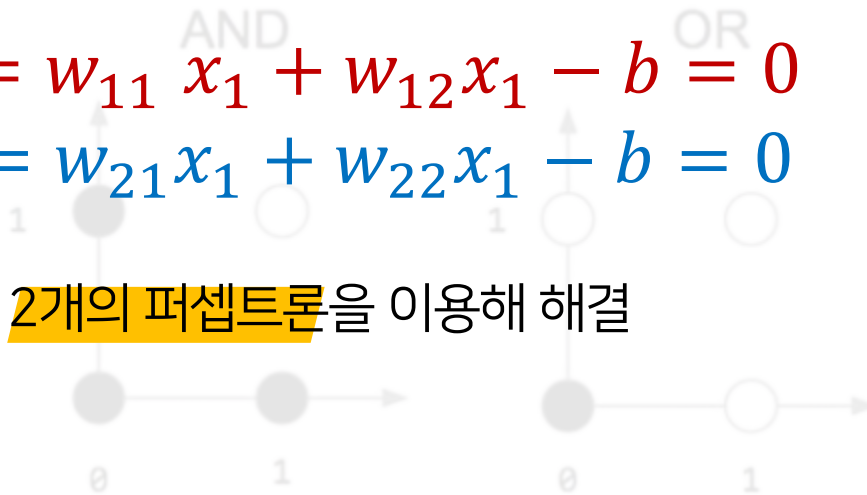
## 퍼셉트론

논리 연산과 한계

$$s_1 = w_{11}x_1 + w_{12}x_2 - b = 0$$

$$s_2 = w_{21}x_1 + w_{22}x_2 - b = 0$$

2개의 퍼셉트론을 이용해 해결



## 퍼셉트론

논리 연산과 한계



해결책 : 다층 퍼셉트론

$$s_1 = w_{11}x_1 + w_{12}x_2 - b = 0$$

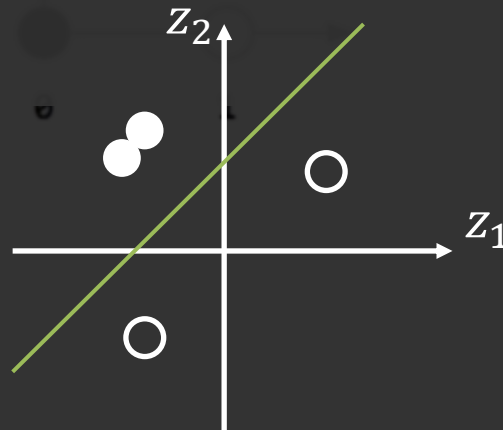
$$s_2 = w_{21}x_1 + w_{22}x_2 - b = 0$$

기존 좌표  $(x_1, x_2)$ 의 변환 :  $z_1 = f(s_1), z_2 = f(s_2)$ 

XOR 문제 해결 방법

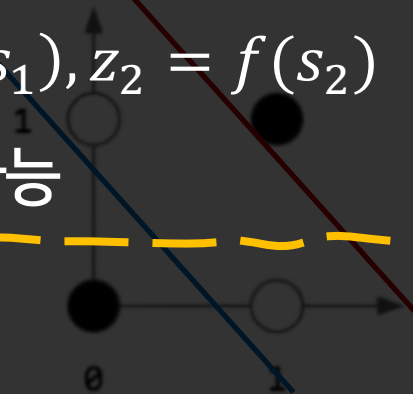
선형적인 구분선 작성 가능

선형 모델의 한계



$$v_1 z_1 + v_2 z_2 - b = 0$$

XOR





## 퍼셉트론

논리 연산과 한계



해결책 : 다층 퍼셉트론

$$s_1 = w_{11}x_1 + w_{12}x_2 - b = 0$$

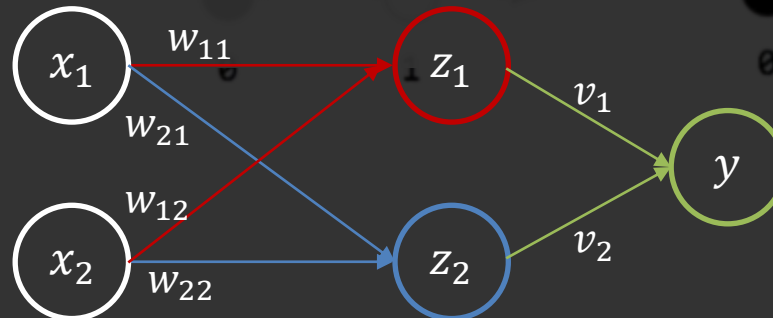
$$s_2 = w_{21}x_1 + w_{22}x_2 - b = 0$$

기존 좌표  $(x_1, x_2)$ 의 변환 :  $z_1 = f(s_1), z_2 = f(s_2)$ 

XOR 문제 해결 방법

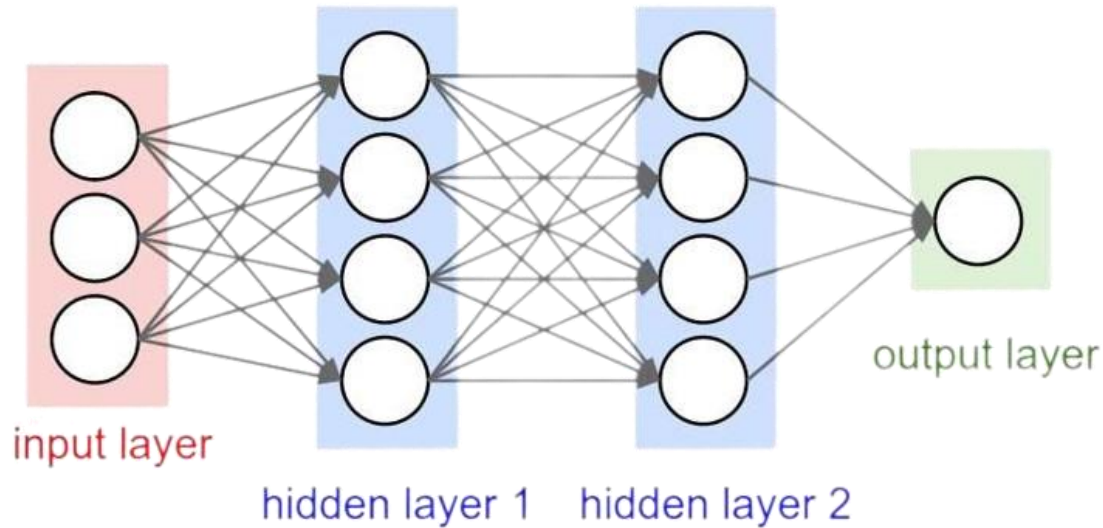
선형적인 구분선 작성 가능

선형 모델의 한계



## 다층 퍼셉트론

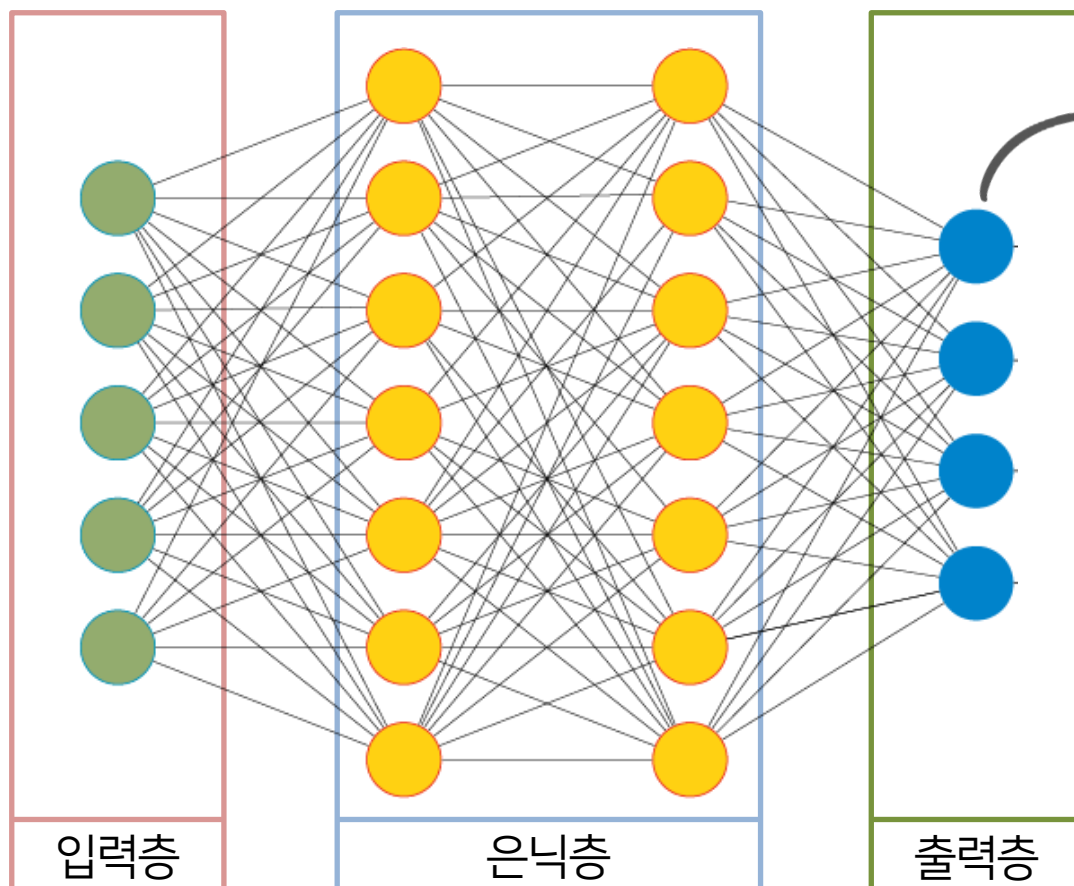
Multi-layer Perceptron



여러 개의 퍼셉트론을 쌓아 올린 형태  
심층 순방향 신경망 (deep feedforward [neural] network)

## 다층 퍼셉트론

퍼셉트론을 여러 층에 걸쳐 쌓은 형태



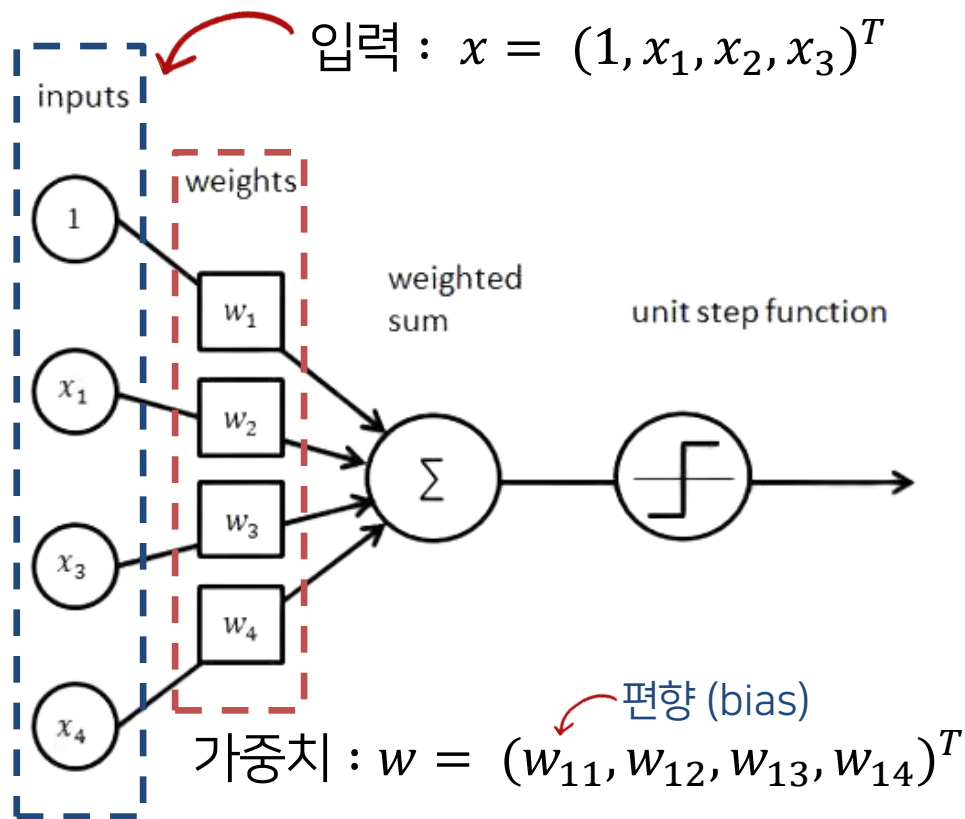
노드 (Node)

각 층의 동그라미  
입력과 출력에 해당하는 원

## 다층 퍼셉트론

## 1 번째 노드의 연산

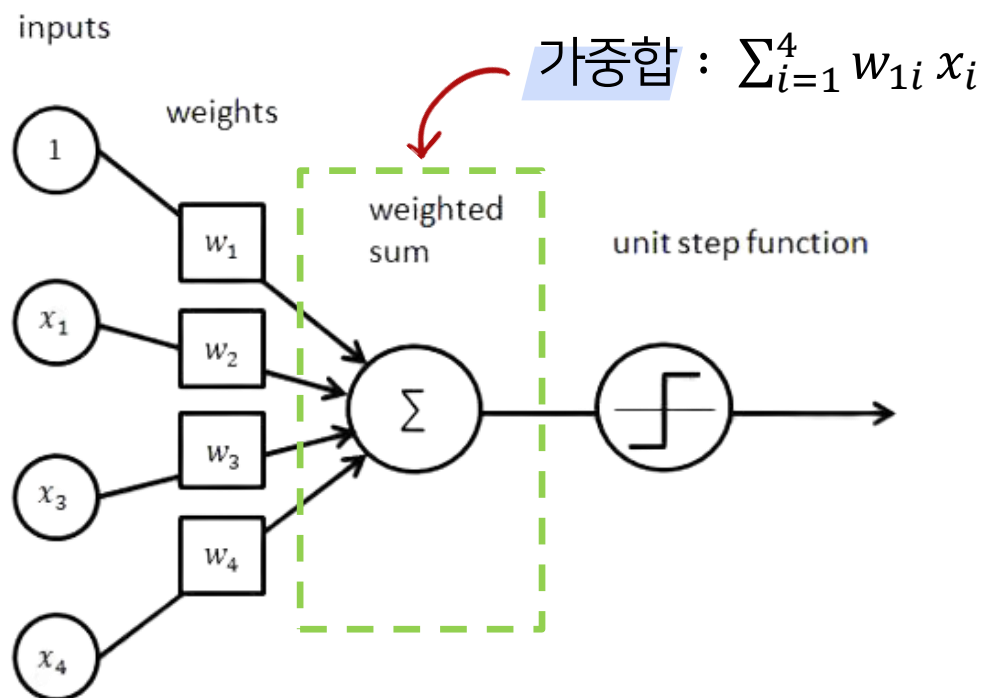
1. input에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 시그모이드 함수에 통과
4. 출력값을 다음 노드에 전달
5. 2번째 노드에 대해 반복...



## 다층 퍼셉트론

### 1 번째 노드의 연산

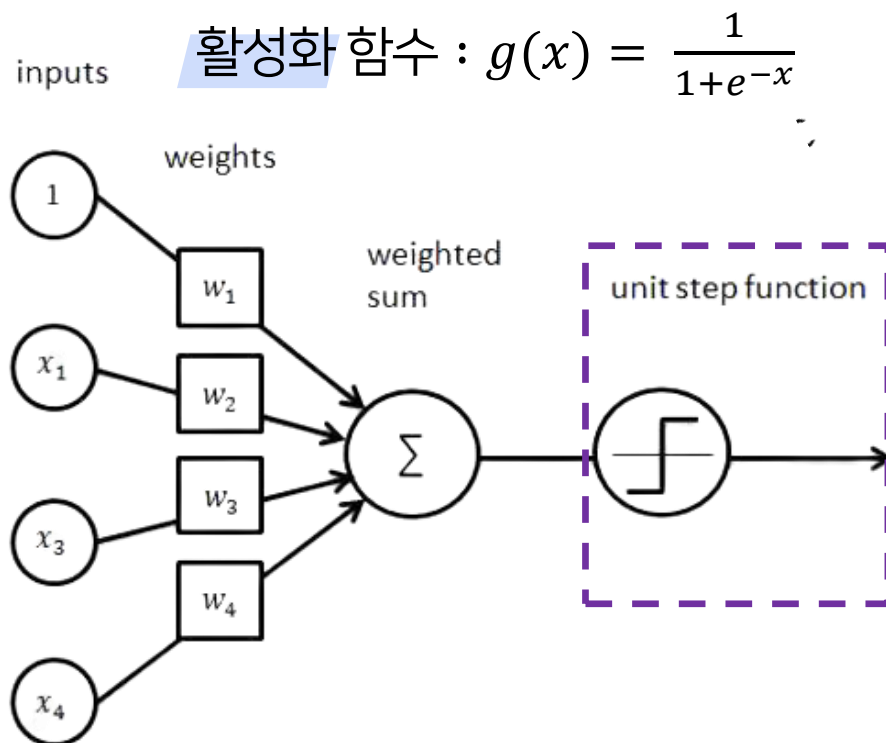
1. input에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 시그모이드 함수에 통과
4. 출력값을 다음 노드에 전달
5. 2번째 노드에 대해 반복...



## 다층 퍼셉트론

### 1 번째 노드의 연산

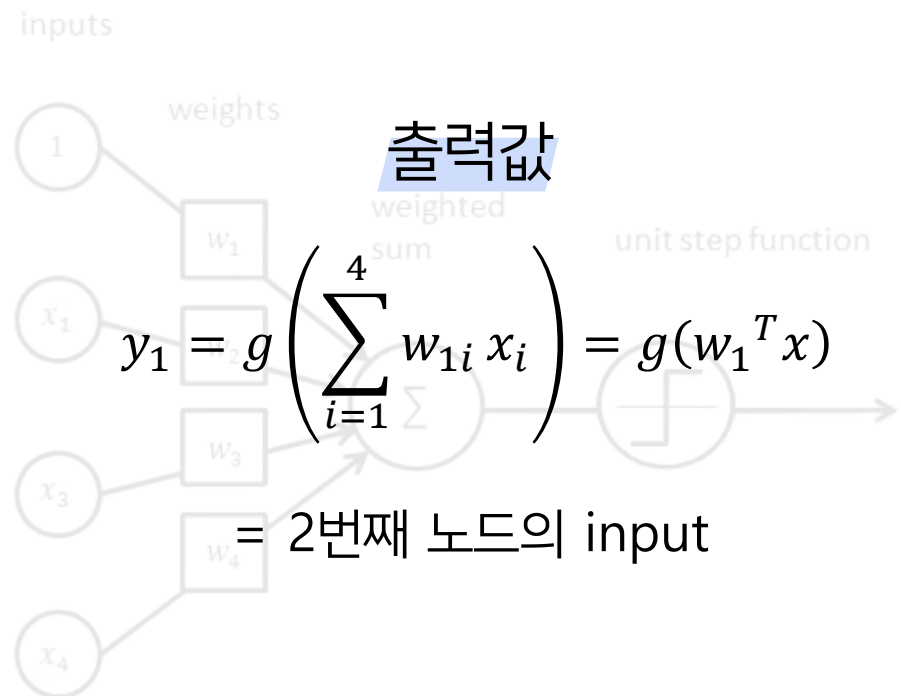
1. Input 에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 시그모이드 함수에 통과
4. 출력값을 다음 노드에 전달
5. 2번째 노드에 대해 반복...



## 다층 퍼셉트론

### 1 번째 노드의 연산

1. input에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 시그모이드 함수에 통과
4. 출력값을 다음 노드에 전달
5. 2번째 노드에 대해 반복...

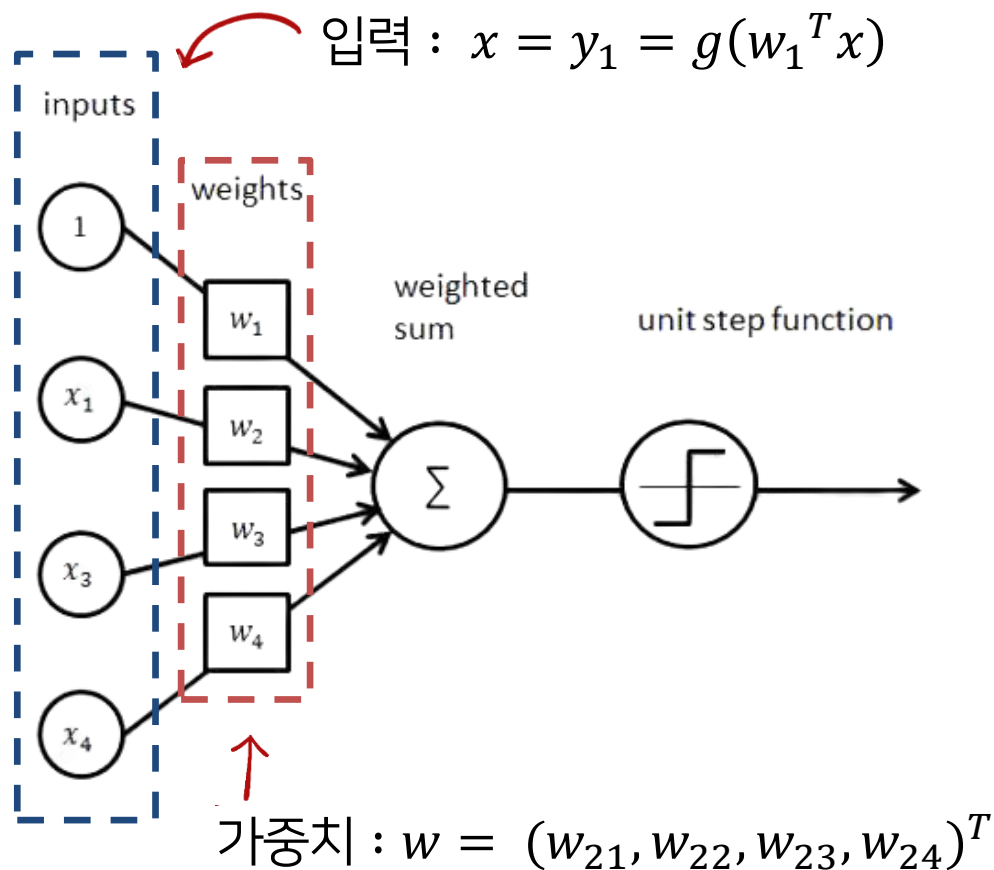


## 다층 퍼셉트론

## 2 번째 노드의 연산

1. input에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 시그모이드 함수에 통과
4. 출력값을 다음 노드에 전달
5. 2번째 노드에 대해 반복...

1번째 노드의 출력값이  
2번째 노드의 입력값으로





## 다층 퍼셉트론

노드의 연산

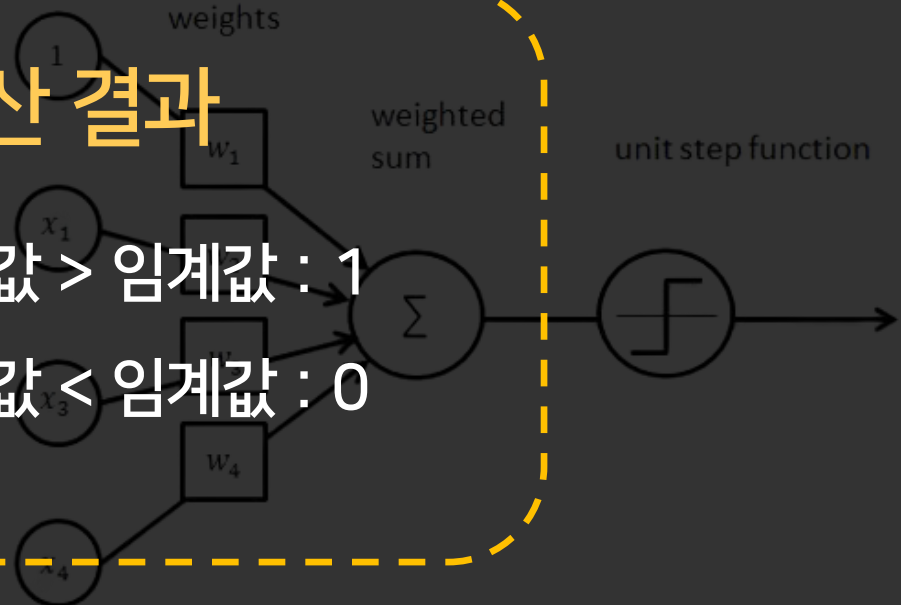


1. 입력(1 포함)에 가중치 부여
2. 입력과 가중치를 모두 곱해 합산
3. 계단 함수에 통과
4. 출력값을 다음 노드에 전달
5. 다음 노드에 대해 반복...

연산 결과

최종 출력값 &gt; 임계값 : 1

최종 출력값 &lt; 임계값 : 0

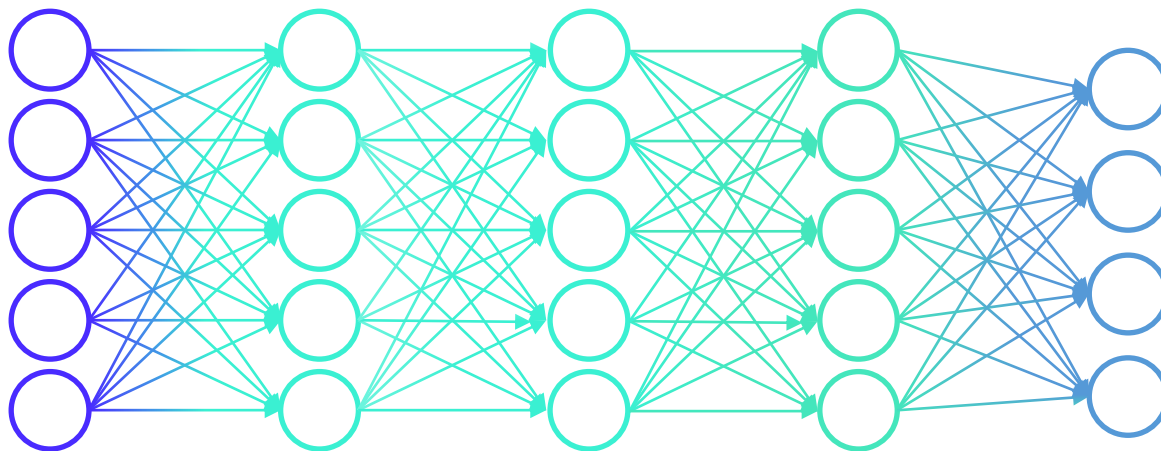


3

신경망

## 신경망

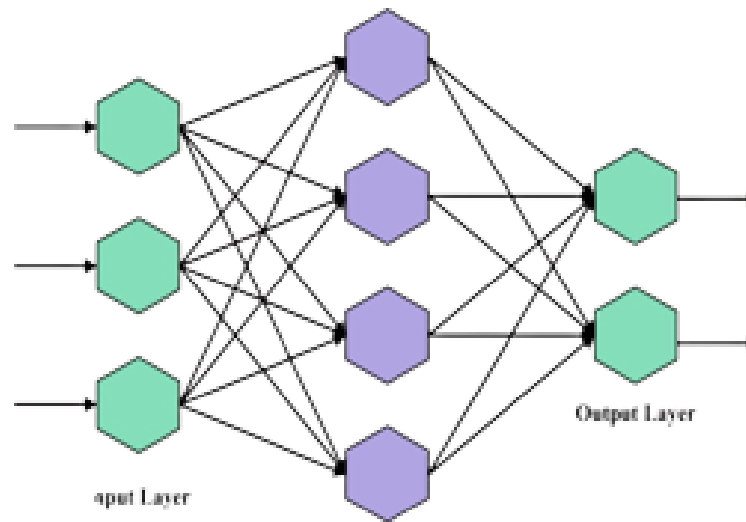
Neural Network



생물의 뇌에서 일어나는 학습을 본뜬 계산모형

## 순전파

## Feedforward Propagation



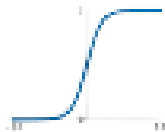
데이터를 계산하여  
입력층부터 출력층까지 전달하는 과정

## 활성화 함수

### Activation Function

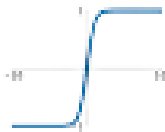
**Sigmoid**

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



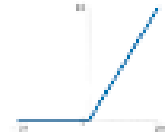
**tanh**

$$\tanh(x)$$



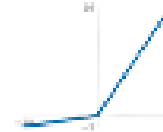
**ReLU**

$$\max(0, x)$$



**Leaky ReLU**

$$\max(0, \alpha x, x)$$

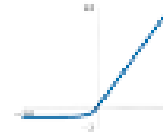


**Maxout**

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

**ELU**

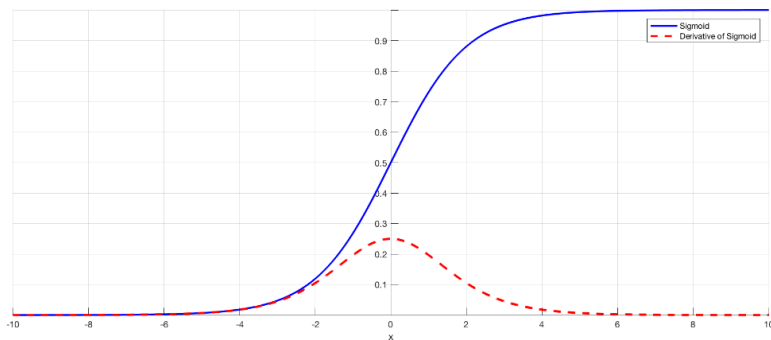
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



가중합이 임계치를 넘었을 때 출력을 반환  
함수의 **비선형성** 부과하는 역할

## 활성화 함수

### 시그모이드 함수



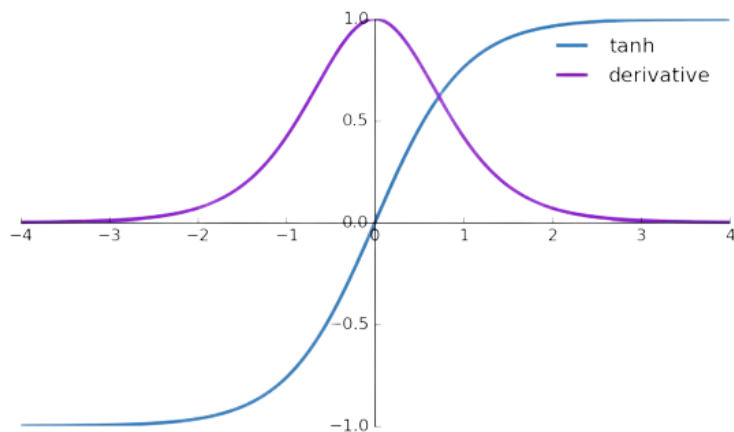
$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



기초적인 활성화 함수  
미분이 가능하여 역전파 가능  
0과 1 사이의 값을 가짐

## 활성화 함수

하이퍼볼릭 탄젠트 함수



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

$$\tanh(x) = 2\sigma(2x) - 1$$



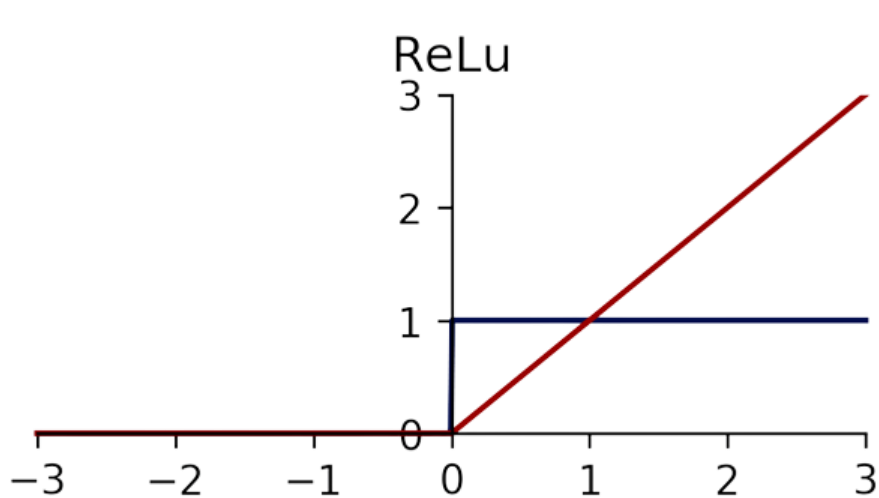
-1과 1 사이의 값을 가짐

0 근처에서 벗어나면 미분값이 0으로 수렴

지수함수 계산으로 인해 연산이 느림

## 활성화 함수

ReLU(Rectified Linear Unit) 함수



장점

은닉층에 주로 사용  
기울기 소실 문제 해결



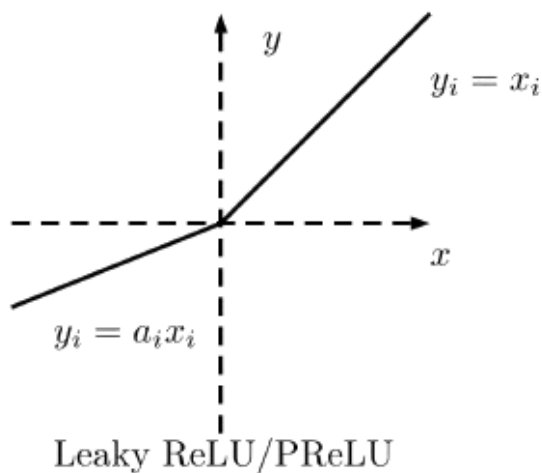
단점

Knockout Problem 발생



## 활성화 함수

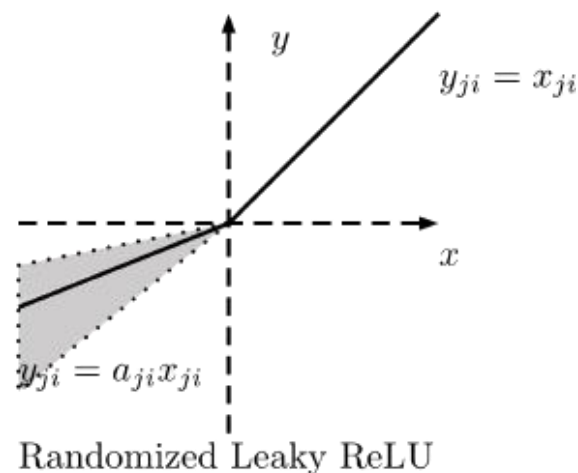
Leaky ReLU/ PReLU / Randomized Leaky ReLU



### Leaky ReLU

Knockout Problem 해결

PReLU는 매개변수  $a$ 를 직접 학습



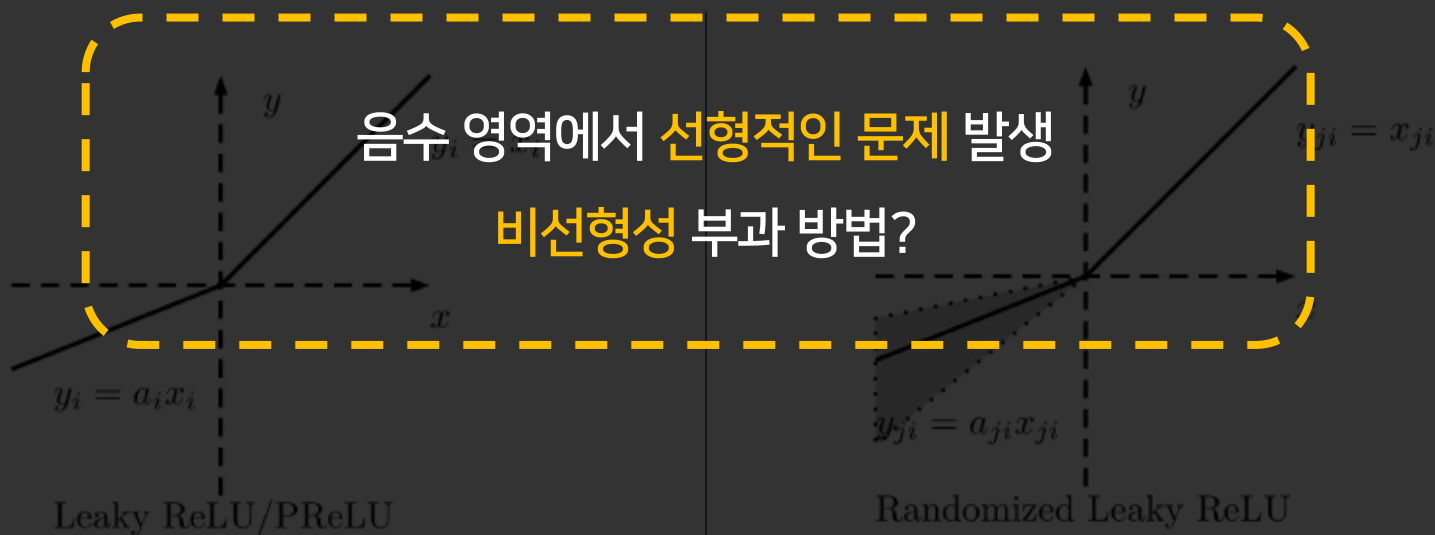
### Randomized

노드별로  $a$ 를 다르게 초기화

과적합 방지

## 활성화 함수

Leaky ReLU/ PReLU / Randomized Leaky ReLU



### Leaky ReLU

Knockout Problem 해결

PReLU는 매개변수  $a$ 를 직접 학습

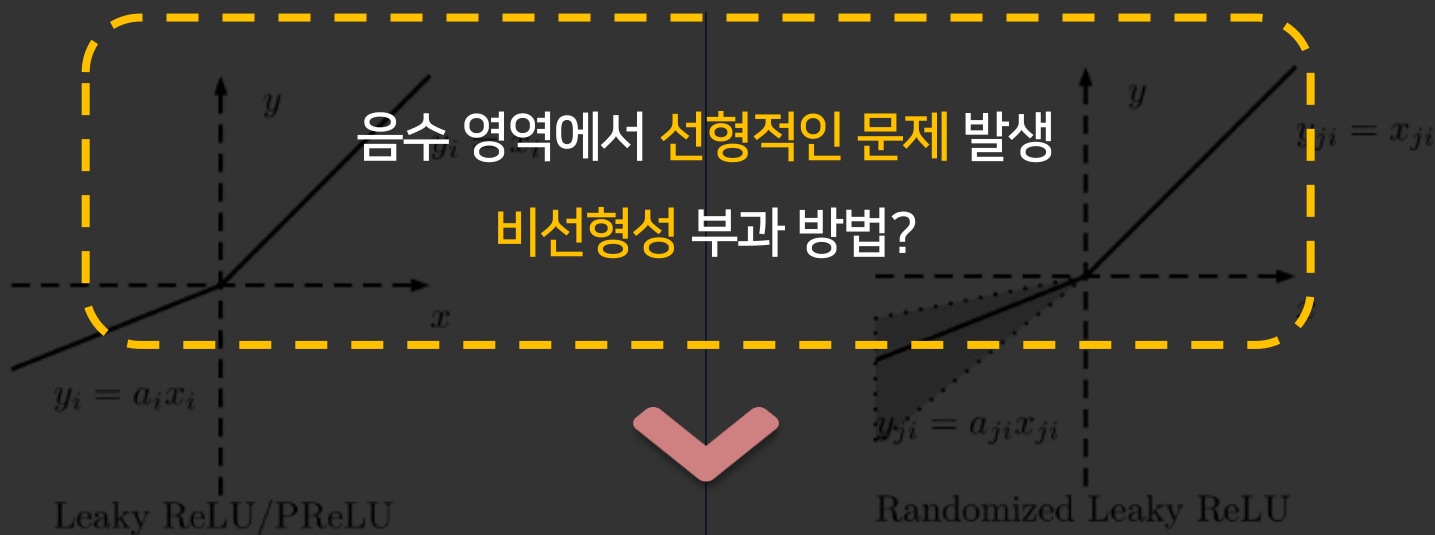
### Randomized

노드별로  $a$ 를 다르게 초기화

과적합 방지

## 활성화 함수

Leaky ReLU/ PReLU / Randomized Leaky ReLU



**ELU** Randomized

Leaky ReLU

Knockout Problem 해결

PReLU는 매개변수  $a$ 를 직접 학습

노드별로  $a$ 를 다르게 초기화

과적합 방지

## 활성화 함수

### ELU 함수

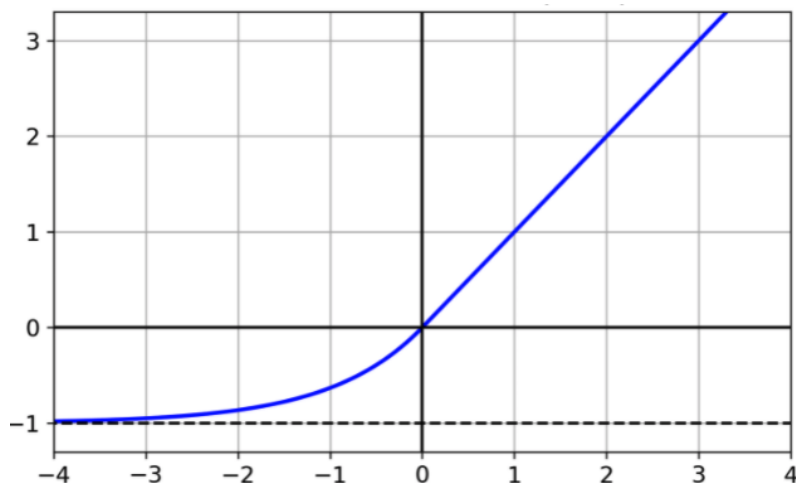


Figure 11-3. ELU activation function



#### 장점

Knockout Problem 해결

음수 영역에서 **비선형적**

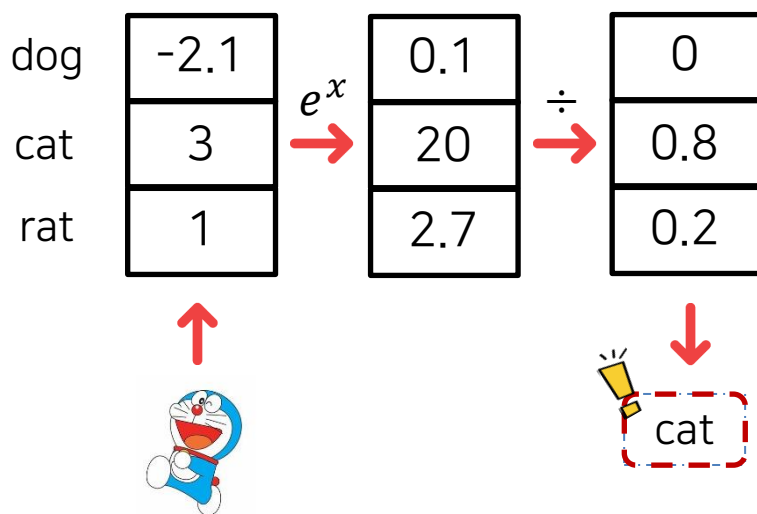


#### 단점

ReLU와 비슷한 성능을 보임  
지수함수로 인한 **연산량 증가**

## 활성화 함수

소프트맥스(Softmax) 함수



$$p_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}}$$

where  $j$  is # of input,  $j = 1, 2, \dots, K$



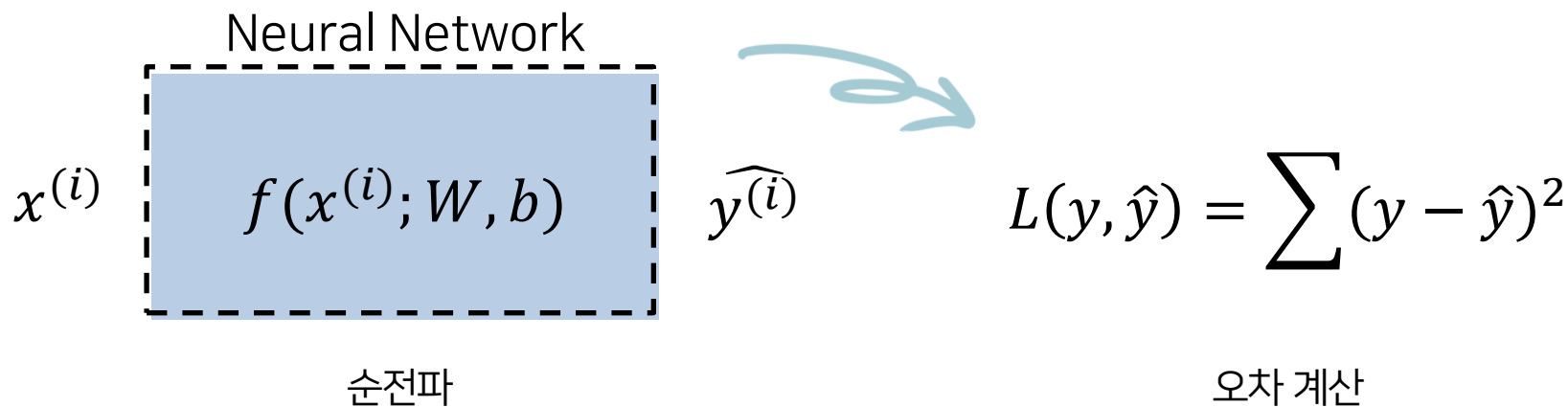
다중 분류에 사용

반환된 값을 확률처럼 해석 가능

Cross entropy 함수를 사용해 오차 계산 가능

## 손실 함수

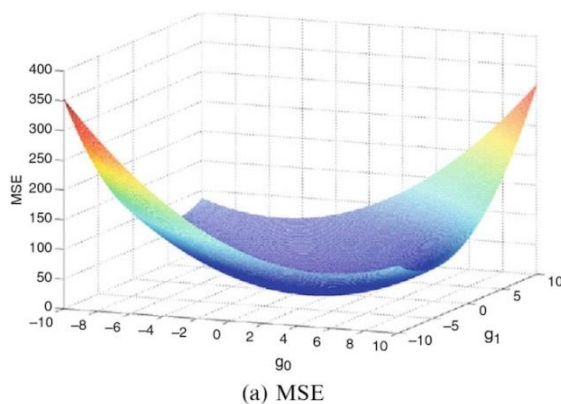
Loss Function



예측 값과 실제 값의 차이를 수치화하는 함수

## 손실 함수

평균 제곱 오차(Mean Squared Error, MSE)



$$L(\hat{y}; \theta) = \frac{1}{2n} \sum_{k=1}^n (y_k - \hat{y}_k)^2$$

$\frac{1}{2}$ 은 미분했을 때 계산의 편의성을 위해서

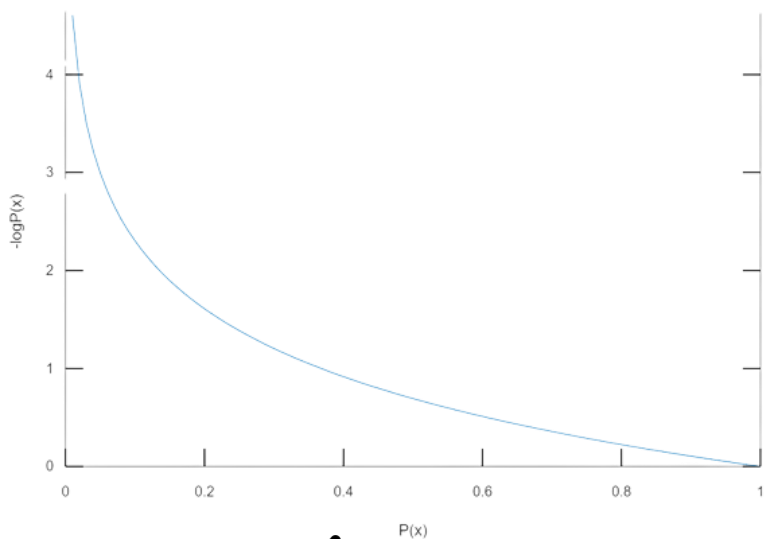


회귀 문제에서 주로 사용

Convex하기 때문에 언제나 극소에서 최솟값을 가짐

## 손실 함수

교차 엔트로피 오차(Cross Entropy Loss)



$$L(\hat{y}; \theta) = - \sum_{i=1}^{\text{output size}} y_i \log \hat{y}_i$$

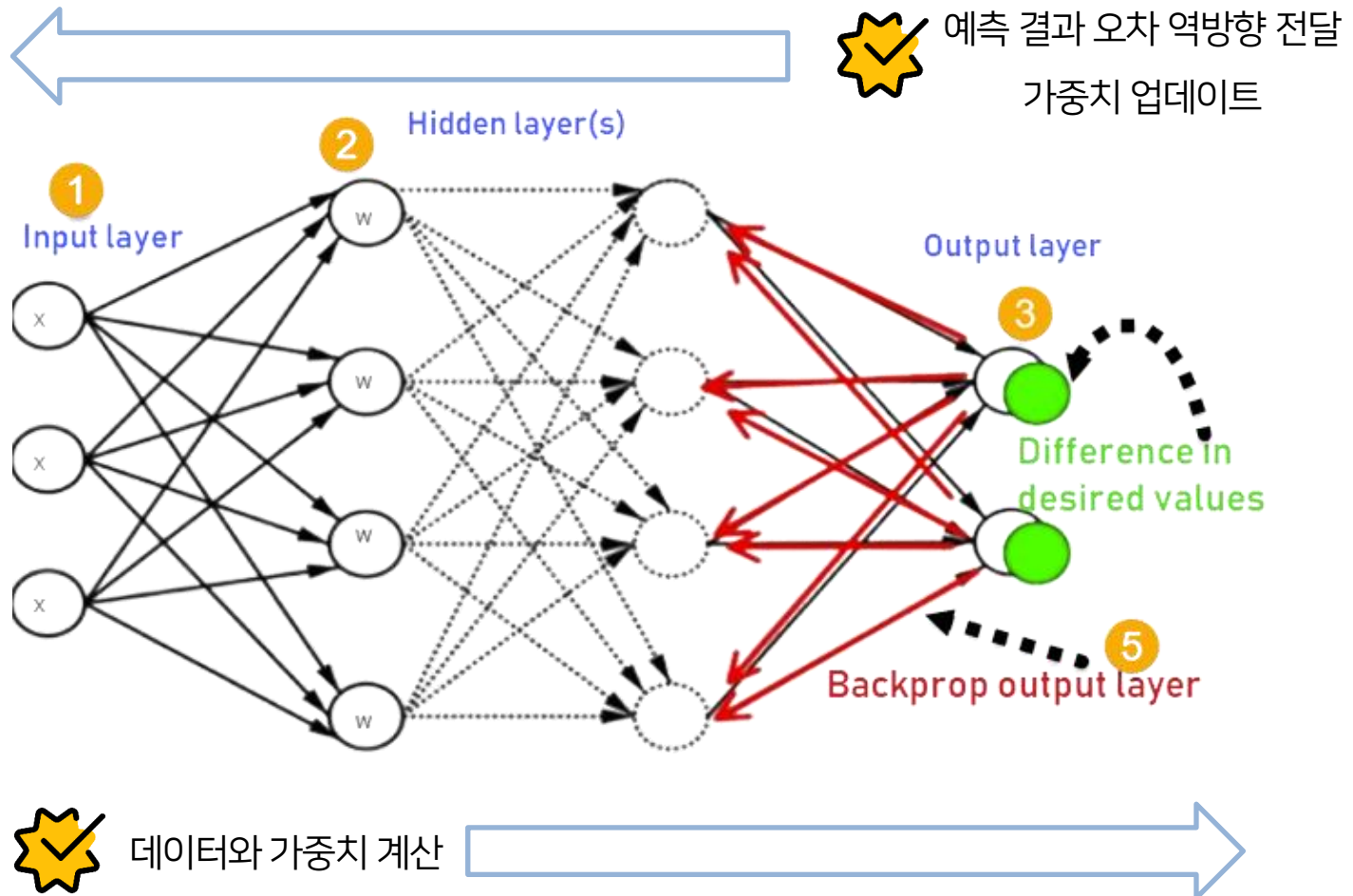


분류 문제에서 주로 사용  
정답과 예측값의 차이가 커질수록 **큰 오차** 반환



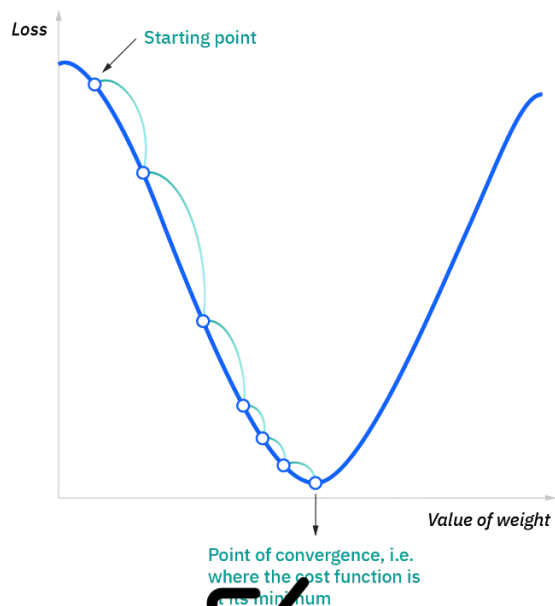
## 역전파

Back Propagation



## 경사하강법

### Gradient Descent



$$x_{i+1} = x_i - \alpha \frac{\partial f(x)}{\partial x}$$

$$W \leftarrow W - \alpha \left( \frac{\partial L}{\partial w} \right)$$

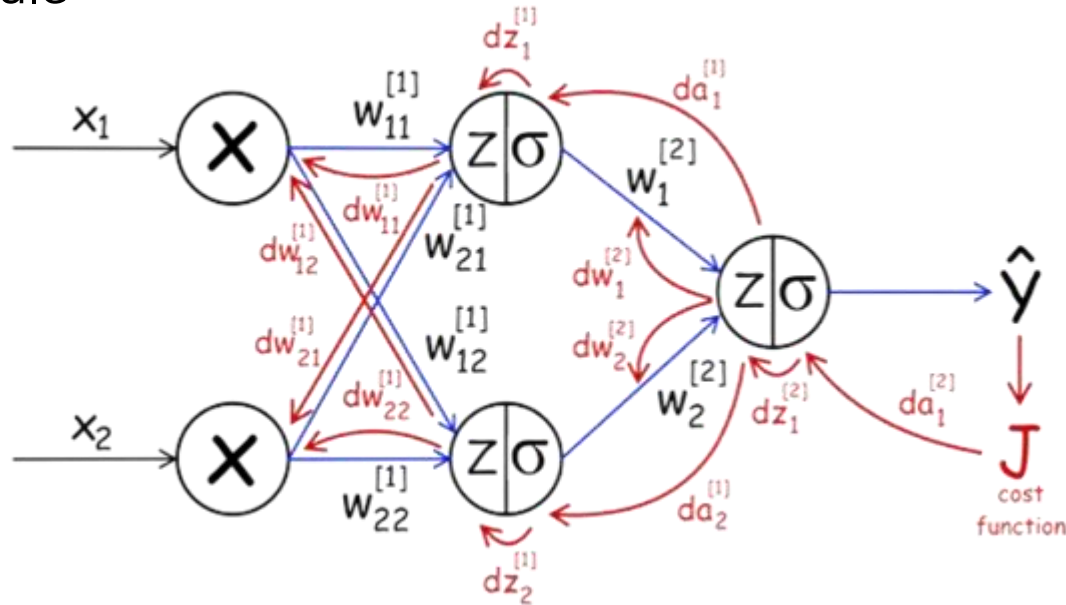
$$b \leftarrow b - \alpha \left( \frac{\partial L}{\partial b} \right)$$



손실함수 값의 기울기를 통해 **가중치 업데이트**  
손실함수가 최저가 될 때 학습 종료

## 연쇄법칙

Chain Rule

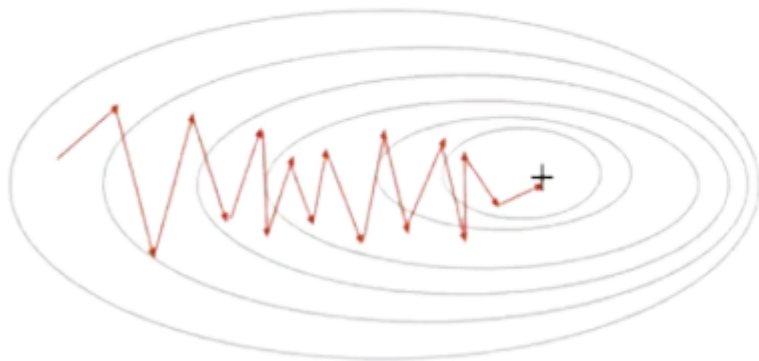


다수의 층으로 구성된 딥러닝 모델 특성 상  
미분의 연쇄법칙을 통해 손실함수의 편미분 값 전달

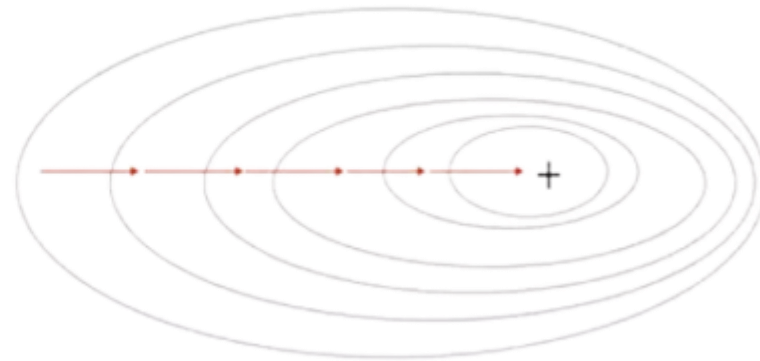
## 확률적 경사 하강법

Stochastic Gradient Descent

Stochastic Gradient Descent



Gradient Descent



모든 데이터가 아닌

batch 단위로 경사하강법 진행



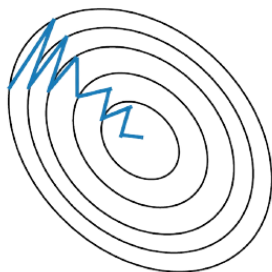
학습 속도 증가,

최적값에 근사적으로 도달

## 모멘텀 : Momentum



Stochastic Gradient  
Descent **without**  
Momentum



Stochastic Gradient  
Descent **with**  
Momentum

관성에 대한 항

$$V(t) = m * V(t - 1) - \alpha \frac{\partial}{\partial w} Cost(w)$$

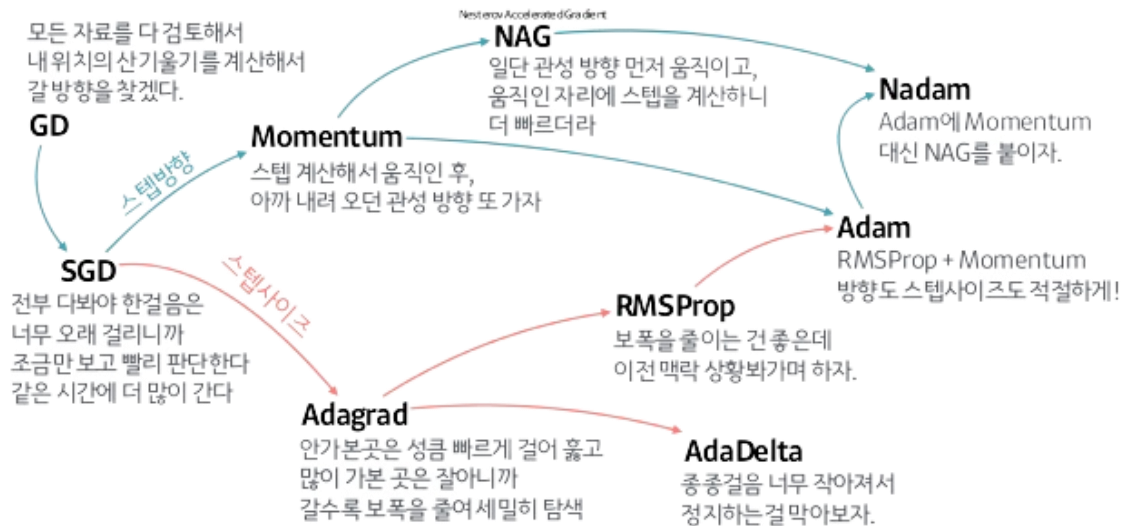
$$W(t + 1) = W(t) + V(t)$$

확률적 경사 하강법에  
관성의 아이디어 도입



Local minima 해소

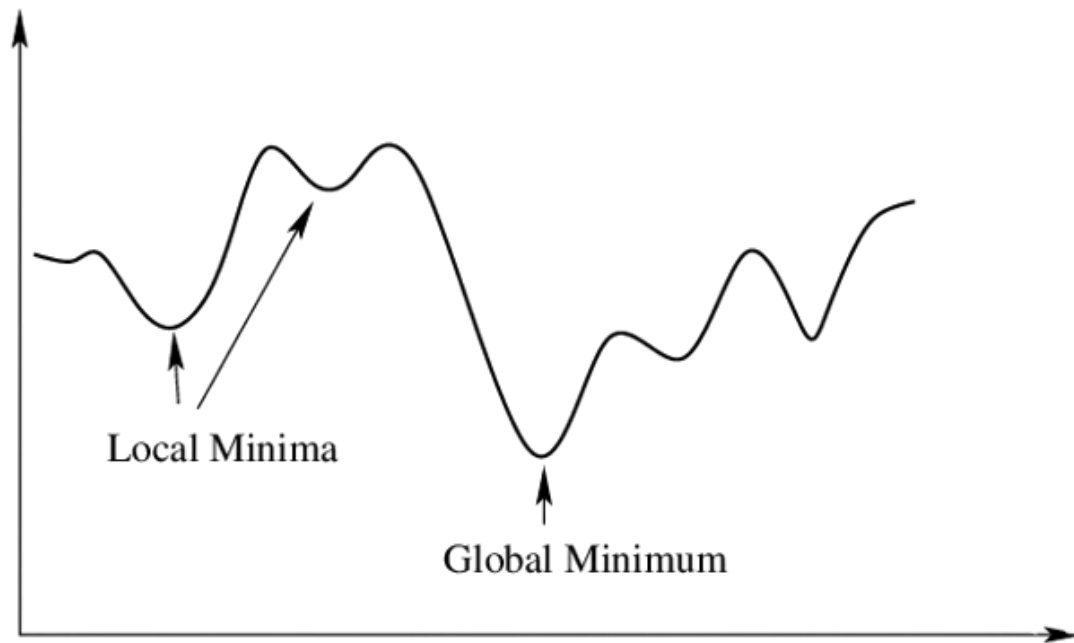
## 다양한 Optimizer



다양한 Optimizer가 존재하며  
각 기법마다 장단점 존재

## Optimizer에서 발생하는 문제

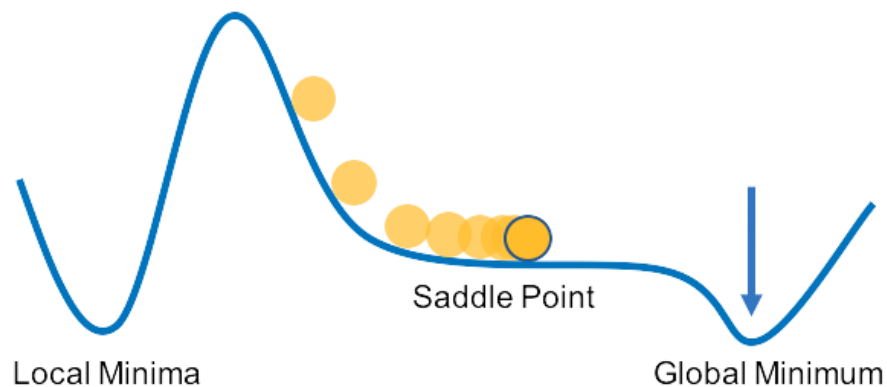
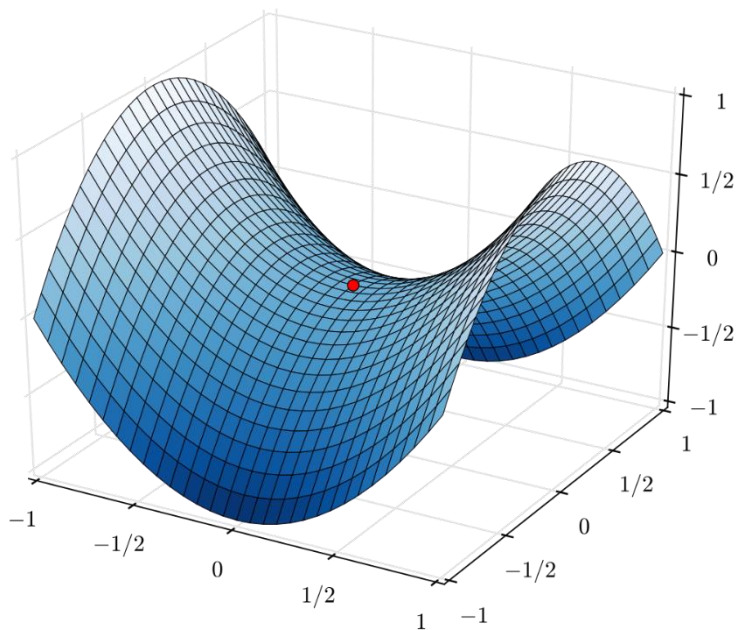
Local Minima



함수에 극소 지점이 여러 개 존재할 때  
미분계수가 0인 곳에서 **학습 종료**

## Optimizer에서 발생하는 문제

### Saddle Point



모든 방향으로 기울기가 0이면서 극점이 아닌 점

고차원 함수에서 주로 나타나는 문제



## Optimizer에서 발생하는 문제

기울기 소실 문제 (Gradient Vanishing Problem)

경사 하강법

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

$$\frac{df}{dx}(x_i) = \frac{df}{dz} \frac{dz}{dy} \frac{dy}{dx} = 0.2 \times \cdots \times 0.8 \approx 0$$

## Optimizer에서 발생하는 문제

기울기 소실 문제 (Gradient Vanishing Problem)

경사 하강법

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

$$\frac{df}{dx}(x_i) = \frac{df}{dz} \frac{dz}{dy} \frac{dy}{dx} = 0.2 \times \dots \times 0.8 \approx 0$$

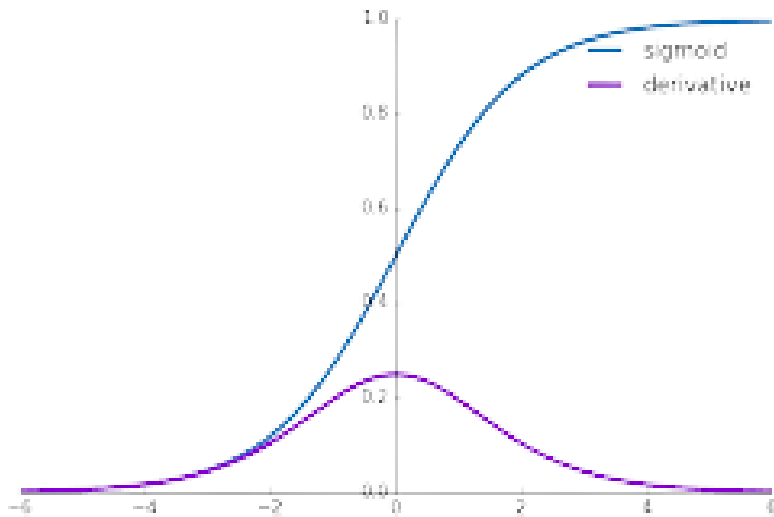


역전파 마지막 노드에서 연쇄법칙으로 인해  
기울기가 0에 가까워짐

## Optimizer에서 발생하는 문제

기울기 소실 문제 (Gradient Vanishing Problem)

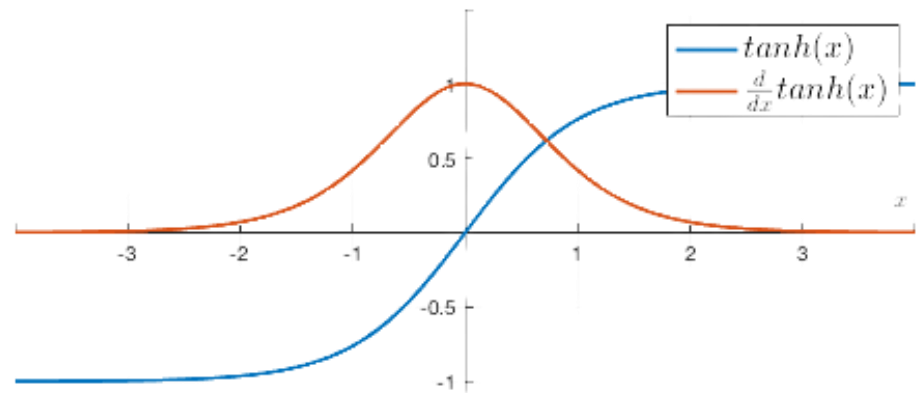
시그모이드 함수의 미분값



도함수의 최대값 0.25

0을 벗어나면 미분값이 0에 수렴

$\tanh$  함수의 미분값

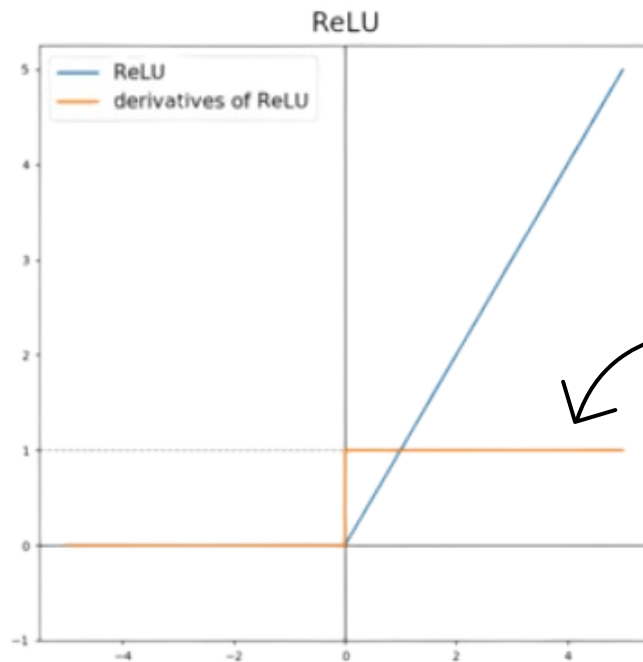


도함수의 최대값 1

시그모이드 함수와 같은 문제 발생

## Optimizer에서 발생하는 문제

기울기 소실 문제 (Gradient Vanishing Problem)



0 또는 1의  
계단함수 형태



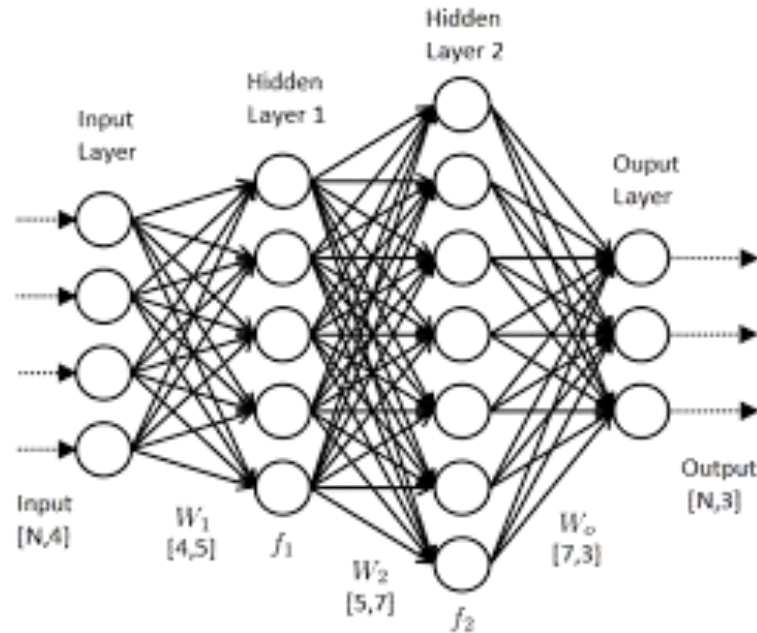
임계점을 넘었을 때 상대적으로 큰 미분 계수를  
가지고 있어 기울기 소실문제 해소

# 4

## 성능 향상 기법

## 가중치 초기화

Weight Initialization



Ex) 경사하강법

초기값 설정에 따라 **수렴 속도** 차이  
학습 시작 시점의 **가중치 설정**의 중요성

## 가중치 초기화

Weight Initialization

### Zero Initialization

파라미터의 가중치 값을  
0으로 초기화



역전파로 갱신했을 때 같은 가중치의 값  
각 노드 역할의 중복성

### Random Initialization

파라미터의 값을 정규분포 혹은  
균일분포를 따르는 값으로 설정



시그모이드 활성화 함수 사용 시  
기울기 소실 문제 발생

## 가중치 초기화

Weight Initialization

### Zero Initialization

파라미터의 가중치 값을  
0으로 초기화



역전파로 갱신했을 때 같은 가중치의 값  
각 노드 역할의 중복성

### Random Initialization

파라미터의 값을 정규분포 혹은  
균일분포를 따르는 값으로 설정



시그모이드 활성화 함수 사용 시  
기울기 소실 문제 발생



## 가중치 초기화


Weight Initialization

### Xavier Initialization

$\frac{2}{\sqrt{n+m}}$  을 표준편차로 하는  
정규분포로 초기화



n: 이전 은닉층 노드  
m: 현재 은닉층 노드

 Sigmoid 활성화 함수 사용 시

층의 노드 개수 반영하여 **강건 (robust)**

### He Initialization

$\sqrt{\frac{2}{n}}$  을 표준편차로 하는  
정규분포로 초기화



 ReLU 활성화 함수 사용 시

활성값의 분포 치우침 방지

## 가중치 초기화

### Weight Initialization

#### Xavier Initialization

$\frac{2}{\sqrt{n+m}}$  을 표준편차로 하는  
정규분포로 초기화



n: 이전 은닉층 노드  
m: 현재 은닉층 노드



층의 노드 개수 반영하여 강건 (robust)  
Sigmoid 활성화 함수 사용 시

#### He Initialization

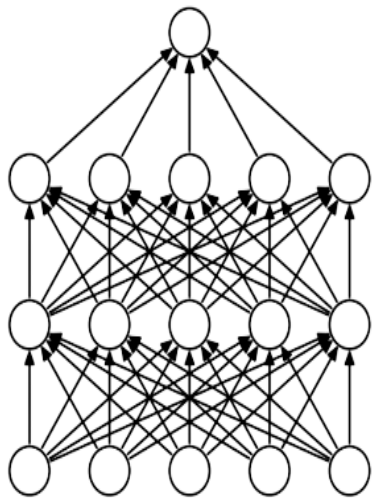
$\sqrt{\frac{2}{n}}$  을 표준편차로 하는  
정규분포로 초기화



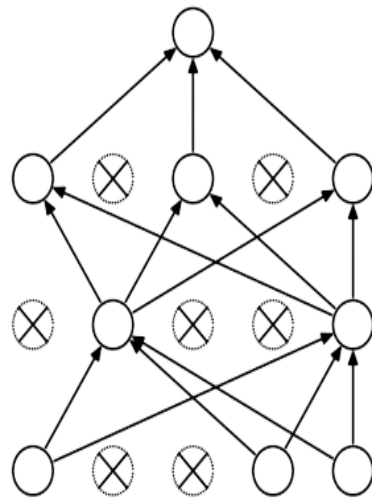
ReLU 활성화 함수 사용 시  
활성값의 **분포 치우침** 방지

## 드롭아웃

Dropout



(a) Standard Neural Net



(b) After applying dropout.

일정한 비율의 노드를 **버리고** 학습하는 방법

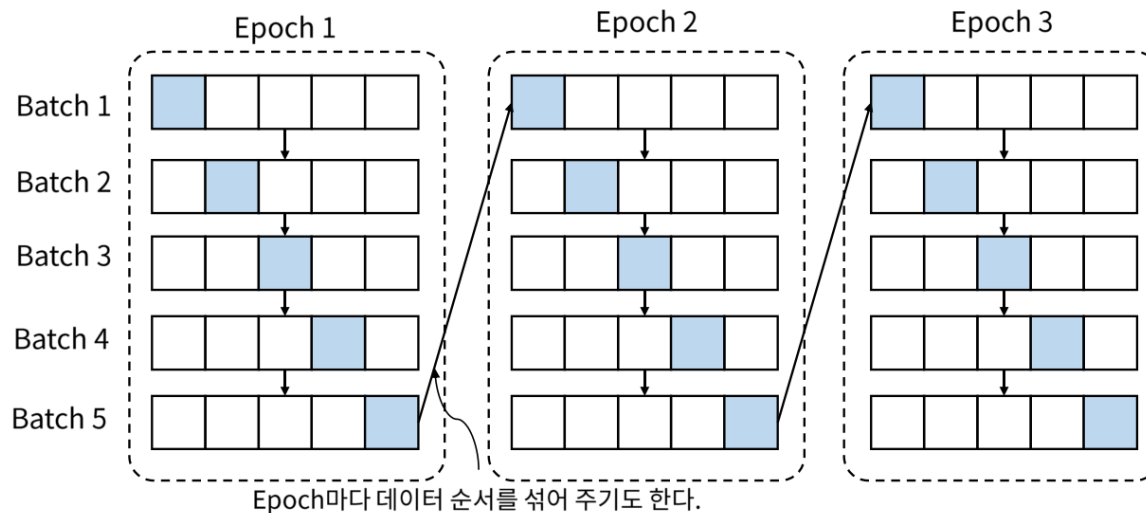


여러 모델 앙상블한 효과

**과적합 방지**

## 배치 정규화

## Batch

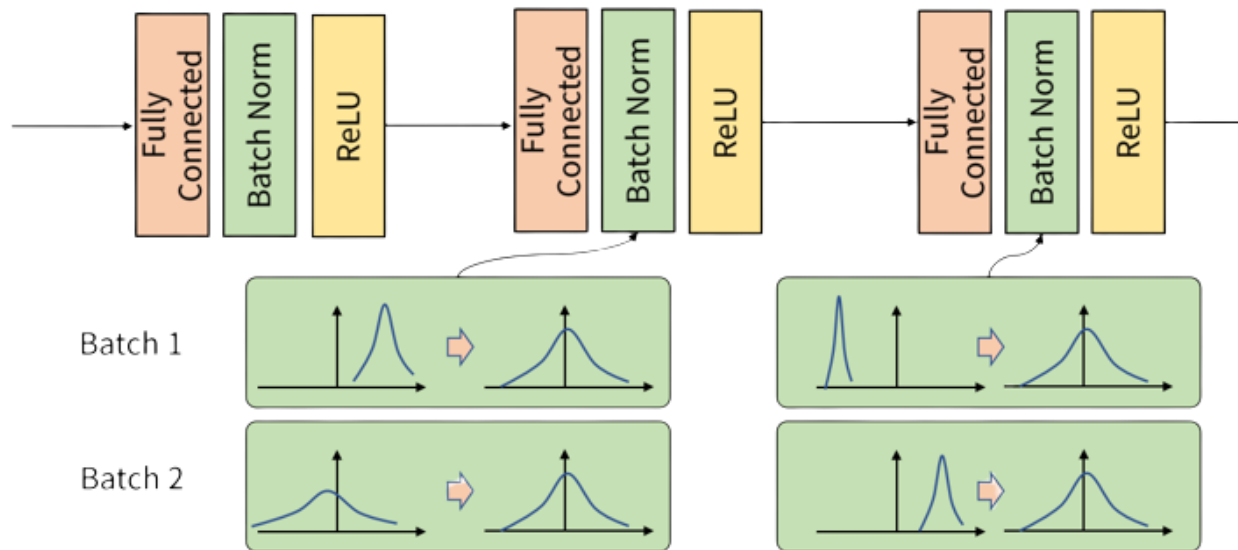


전체 데이터 셋이 1000개를 10개로 나누면 한 batch당 100개의 데이터!

학습 데이터가 많은 경우 한 번에 계산이 어려워  
전체 데이터셋을 **batch** 단위로 나눔

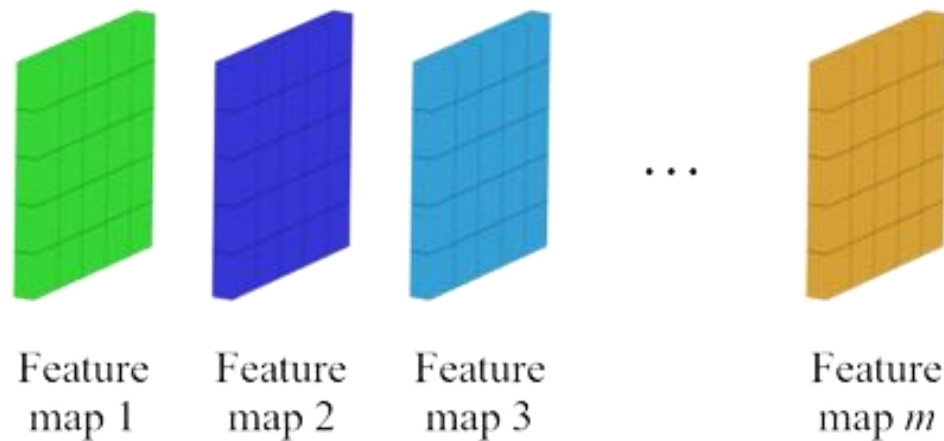
## 배치 정규화

### Batch Normalization



batch별 데이터를 평균과 분산으로 정규화  
예측과정에서는 학습 시 얻은 batch의 **평균** 사용

## Layer Normalization



feature 차원에서 정규화 진행

Sequence에 강건

**Sequential** 데이터를 다루는 RNN에 적용



THANK YOU

