

딥러닝 이론에서 실습까지



POSTECH
창의IT융합공학과
이도엽

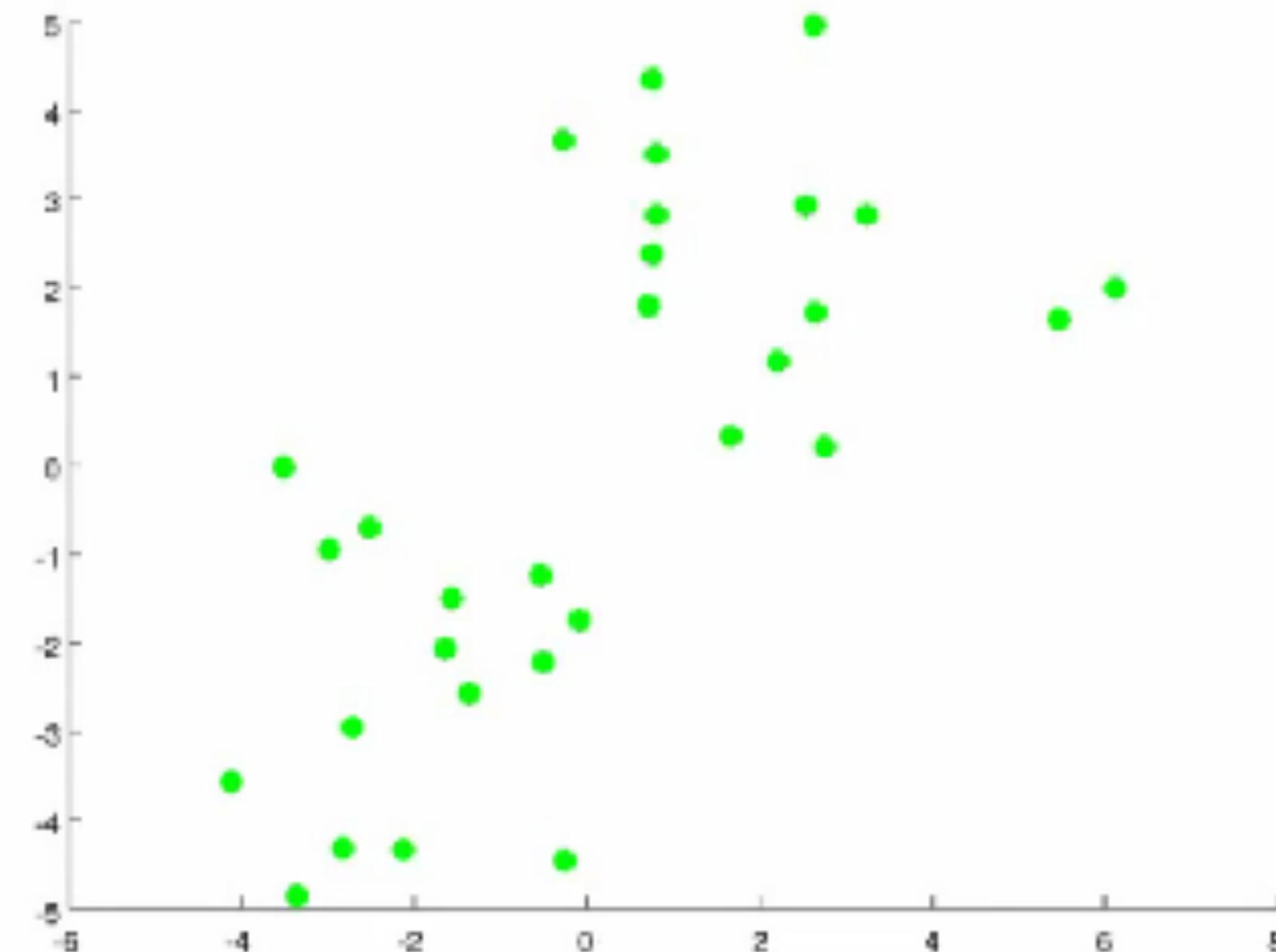
Unsupervised Learning

K-mean Clustering

What is Clustering?

- 군집화 (Clustering): 데이터를 유사한 집단 또는 그룹으로 나누어 묶어주는 것
- 일반적으로 **Unlabeled Data**를 유사도 (거리)에 따라 K 개의 군집으로 나눈다는 점에서 **Unsupervised Learning**
- 결과에 대한 정답은 정해져 있지 않음 (but. 기준은 존재)

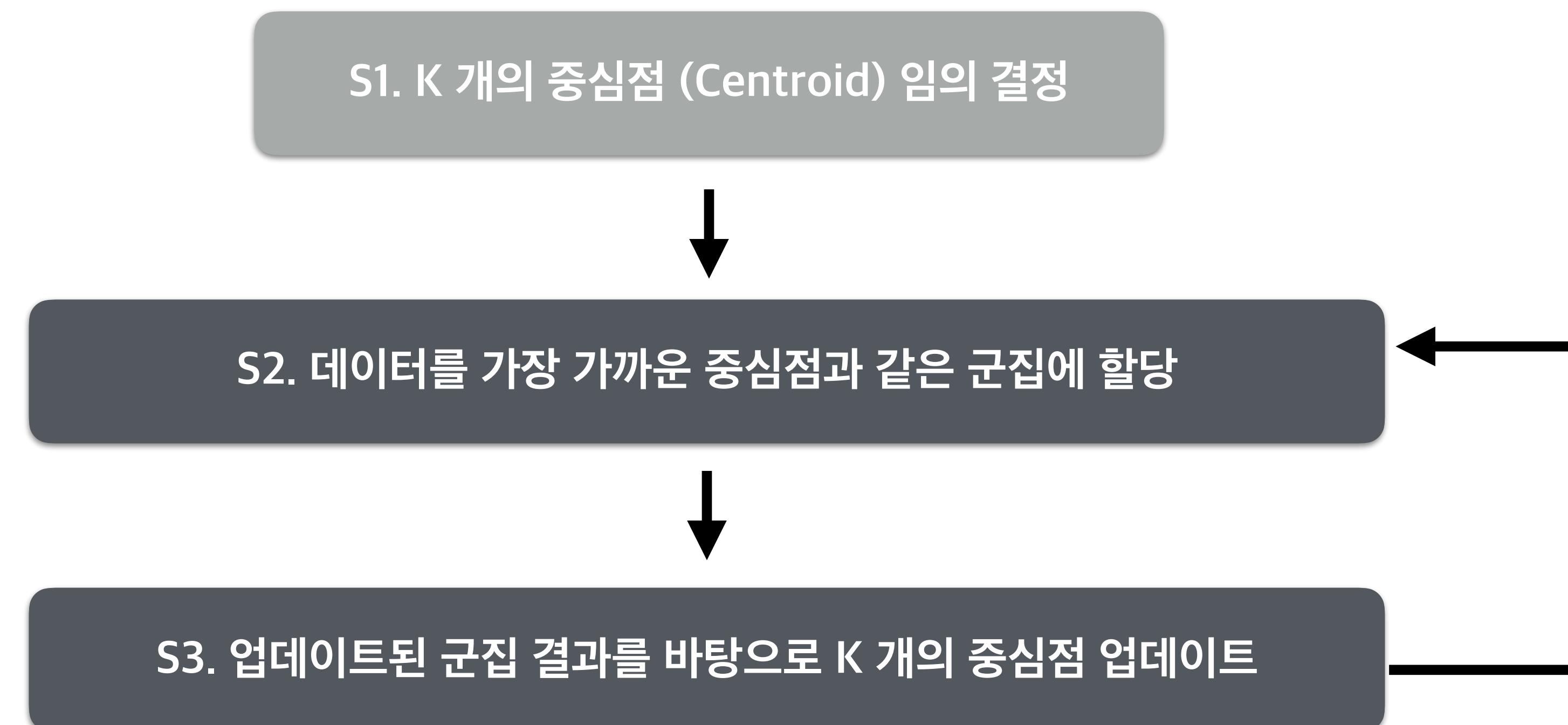
<Unlabeled Data 예시>



How can we cluster these data?

Example

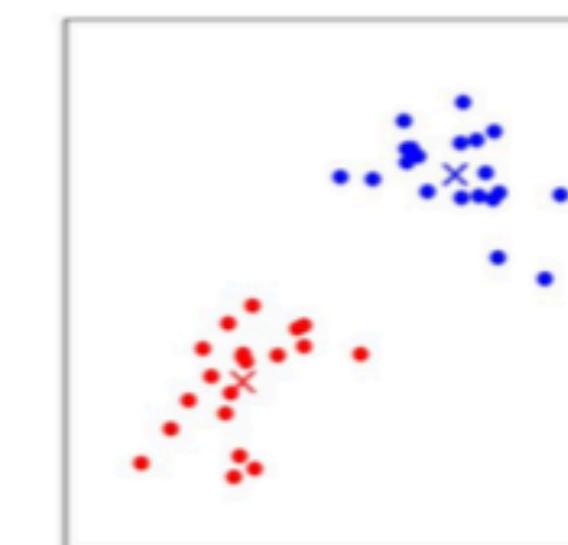
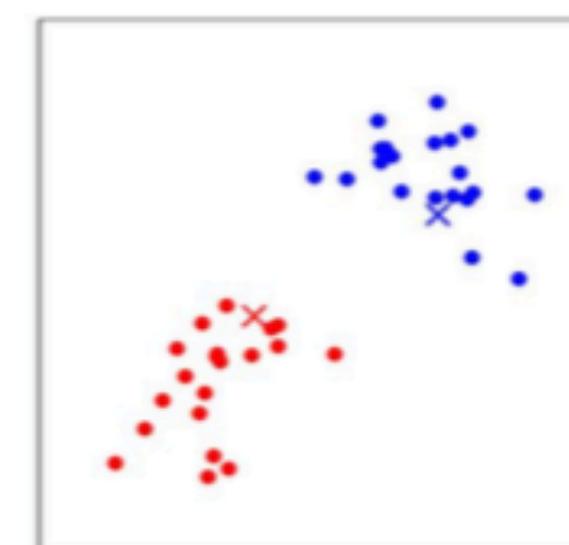
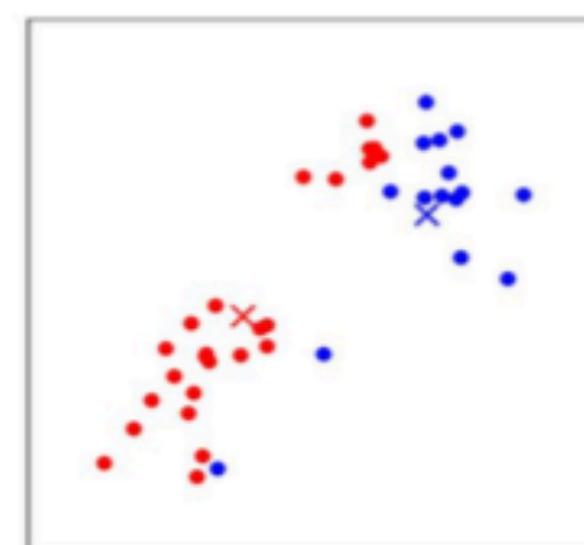
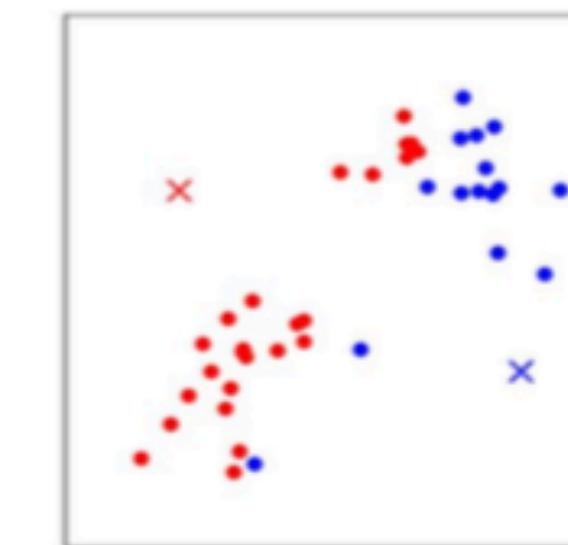
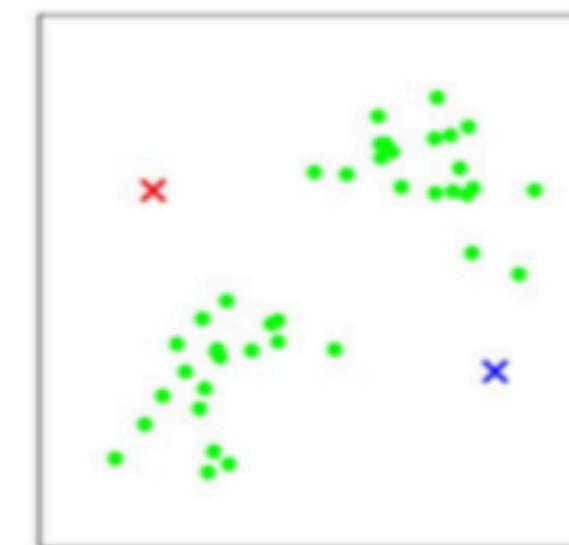
- K-means Clustering Algorithm은 Unlabeled Data를 Clustering에 간단하고 적합한 방법 중 하나임
- 군집에 대한 기준점(Centroids)를 선정하고, 각 데이터를 가장 가까운 기준점과 같은 군집으로 할당하는 방법
- 군집 할당 이후 기준점을 다시 계산하여 업데이트하고, 업데이트된 기준점을 바탕으로 다시 데이터 군집 할당하는 과정을 반복



수렴할 때까지
반복

Example

- K-means Clustering Algorithm은 Unlabeled Data를 Clustering에 간단하고 적합한 방법 중 하나임
- 군집에 대한 기준점(Centroids)를 선정하고, 각 데이터를 가장 가까운 기준점과 같은 군집으로 할당하는 방법
- 군집 할당 이후 기준점을 다시 계산하여 업데이트하고, 업데이트된 기준점을 바탕으로 다시 데이터 군집 할당하는 과정을 반복



a) Original dataset(●)

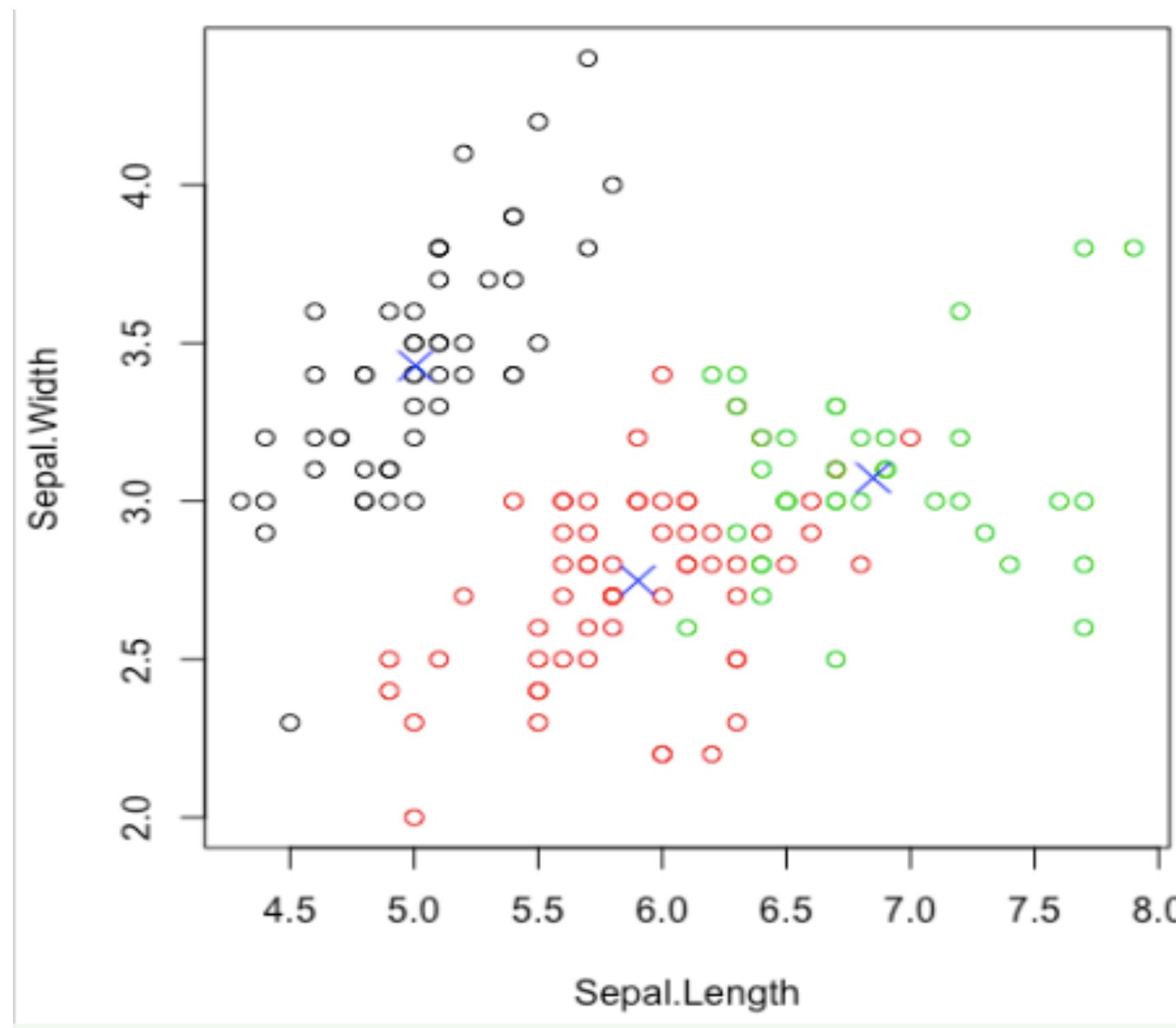
b) Random initial cluster centroids(✕, ✕):
임의로 초기 기준점(centroids)를 선정

c-f) k-mean clustering:
각 데이터를 가까운 centroid와 같은 군집에 할당함

K-means Clustering

- K-means Clustering은 n개의 객체를 k개의 군집으로 나눌수 있는 모든 가능한 방법을 점검해 최적화된 군집을 형성

<K-means Clustering Algorithm>



1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ randomly.

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

For each j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

}

1. Centroids 초기화

2. 소속 Cluster 설정 및 Centroids 업데이트

K-means Clustering on Image Segmentation

$K = 2$



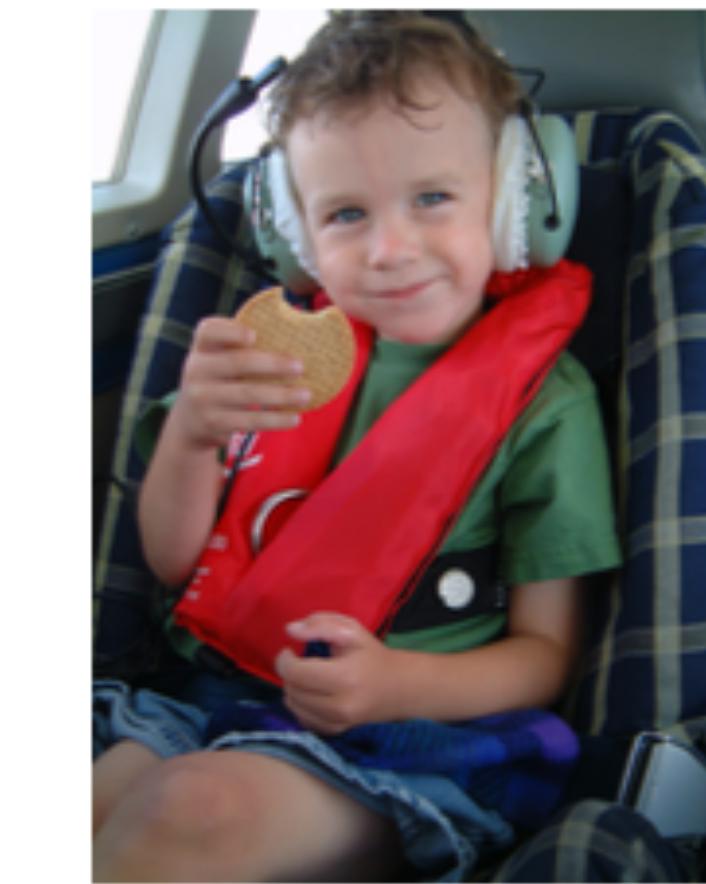
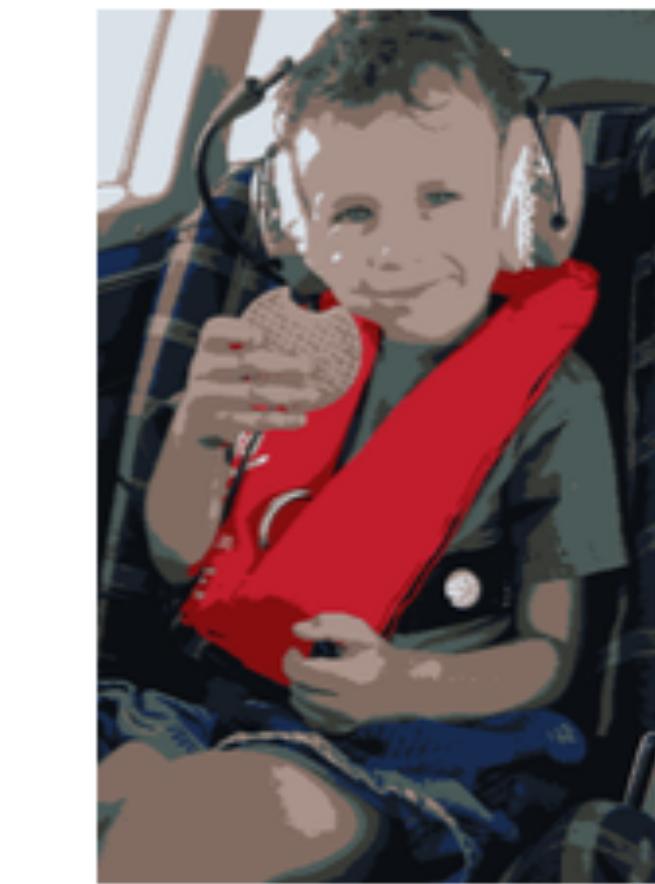
$K = 3$



$K = 10$



Original image



Optimization in k-means

- Unsupervised Learning의 모델 선정의 최적화를 위한 객관적인 지표가 필요함.
- Training example $x^{(i)}$ 와 그에 대응 되어 있는 cluster centroid $\mu_{c^{(i)}}$ 차이의 제곱의 합을 측정함
- Objective(Distortion) Function of K-means Clustering Optimization

$$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

Optimization in k-means

- Unsupervised Learning의 모델 선정의 최적화를 위한 객관적인 지표가 필요함.
- Training example $x^{(i)}$ 와 그에 대응 되어 있는 cluster centroid $\mu_{c^{(i)}}$ 차이의 제곱의 합을 측정함
- Objective(Distortion) Function of K-means Clustering Optimization

$$\min_{c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K} J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

1. Initialize cluster centroids $\mu_1, \mu_2, \dots, \mu_k \in \mathbb{R}^n$ randomly.

2. Repeat until convergence: {

For every i , set

$$c^{(i)} := \arg \min_j \|x^{(i)} - \mu_j\|^2.$$

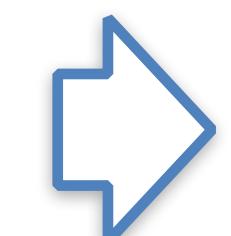
- minimize $J(\dots)$ with respect to c

For each j , set

$$\mu_j := \frac{\sum_{i=1}^m 1\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{c^{(i)} = j\}}.$$

- minimize $J(\dots)$ with respect to μ

}



- $J(\dots)$: monotonic decrease
- $J(\dots)$ converges

2 Kinds of Problem in K-means Clustering

□ K-means Clustering은 참값이 존재하지 않는 Unsupervised Learning의 특성상 2가지 문제가 발생 가능함.

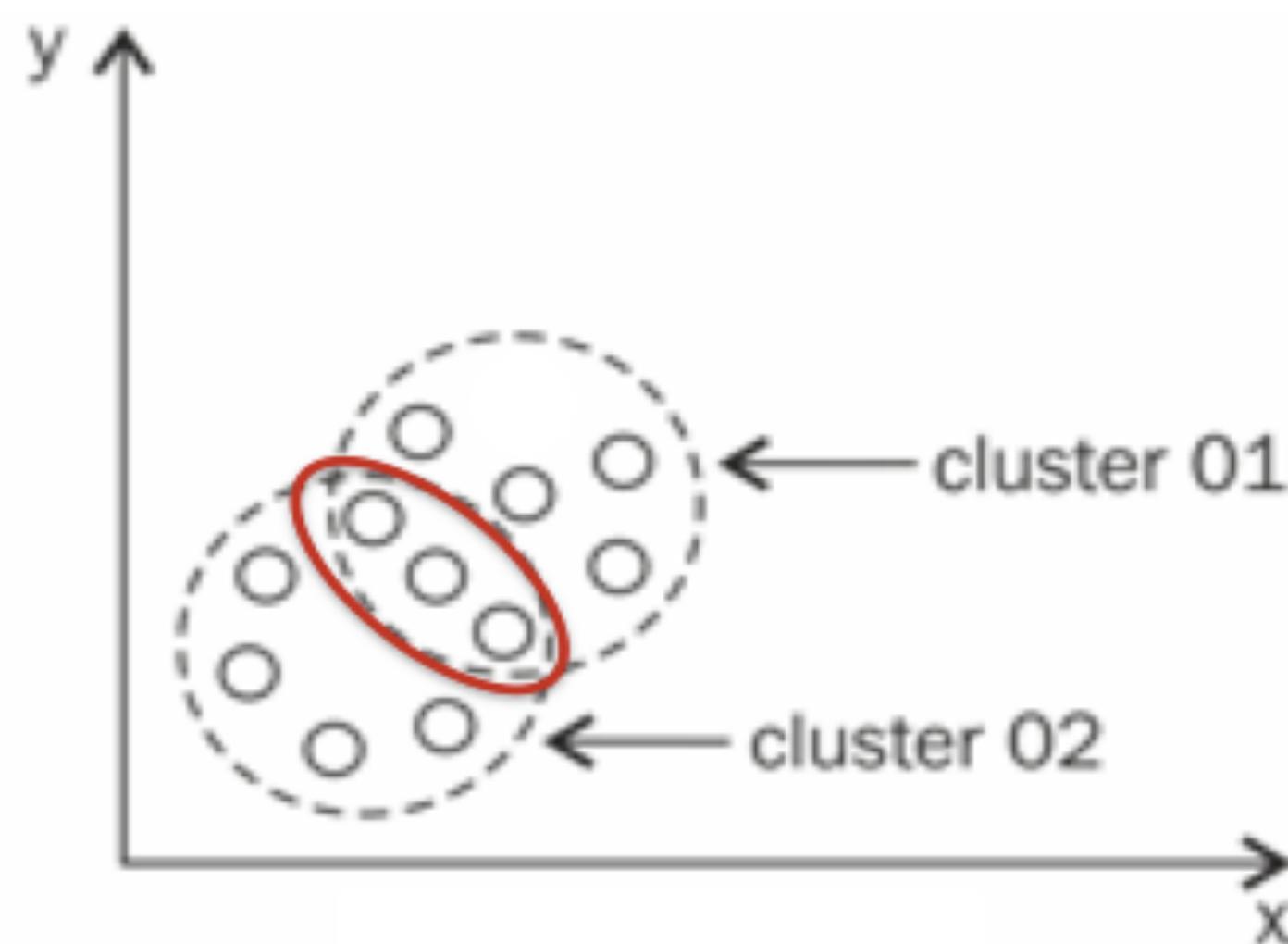
1) No Assigned Centroid Problem: 특정 Cluster에 어떤 Data도 포함되지 않는 경우 (비어있는 Cluster)

- Remove the centroid
- Randomly reinitialize

2) Non-separated Data Problem: Data의 Cluster 기준이 명확하지 않는 경우

- Pre-knowledge of the Domain
- Fuzzy K-means Clustering

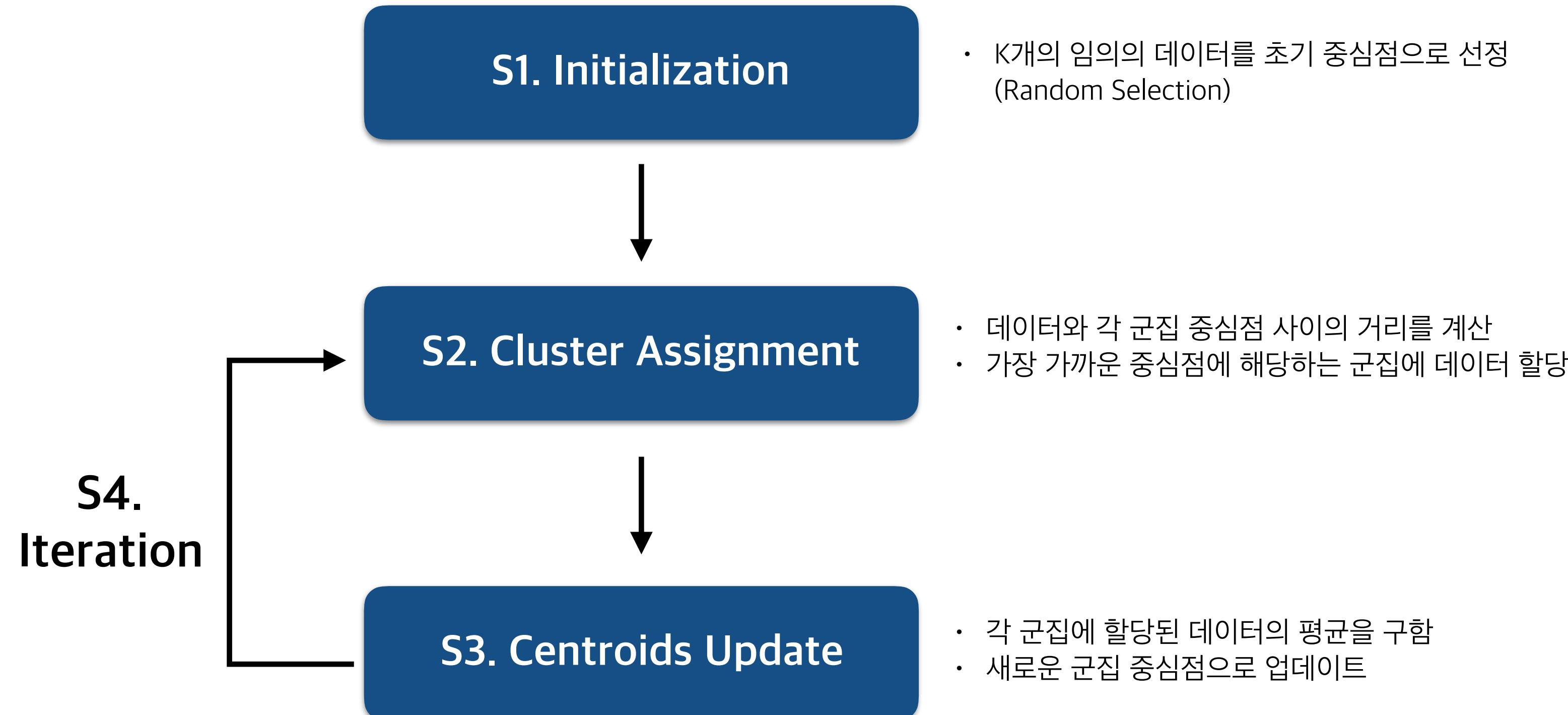
<cluster 기준이 명확하지 않음>



P1.S2. K-means Clustering in Detail

❑ K-means Clustering Algorithm은 Centroids와 Cluster Assignment의 계산을 반복하여 K 개의 군집 도출

<K-means Clustering Algorithm>

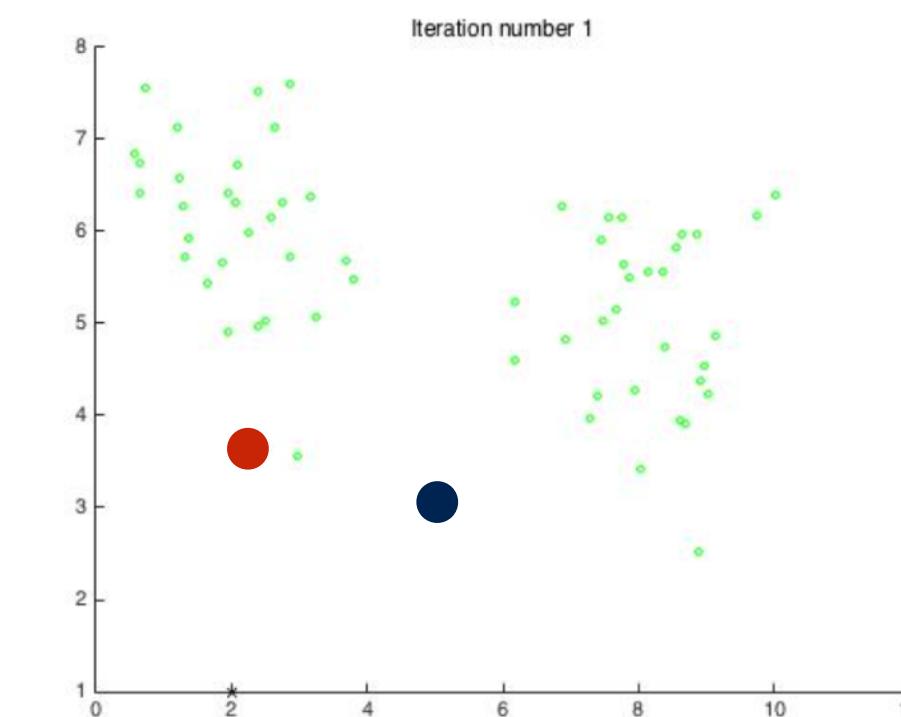


- K개의 임의의 데이터를 초기 중심점으로 선정 (Random Selection)

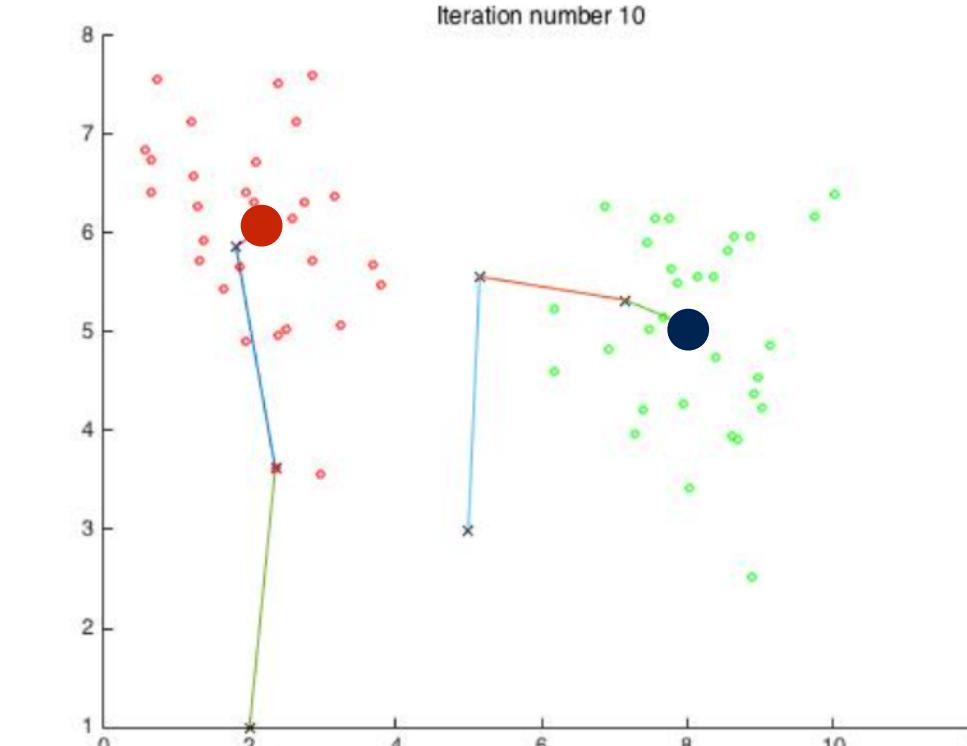
- 데이터와 각 군집 중심점 사이의 거리를 계산
- 가장 가까운 중심점에 해당하는 군집에 데이터 할당

- 각 군집에 할당된 데이터의 평균을 구함
- 새로운 군집 중심점으로 업데이트

<K=2, Initialization >



<K=2, Result >

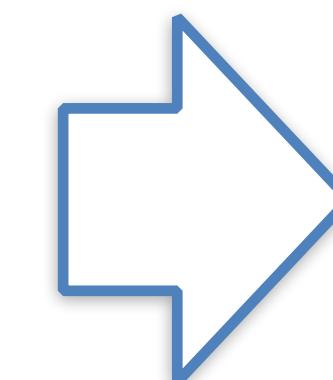
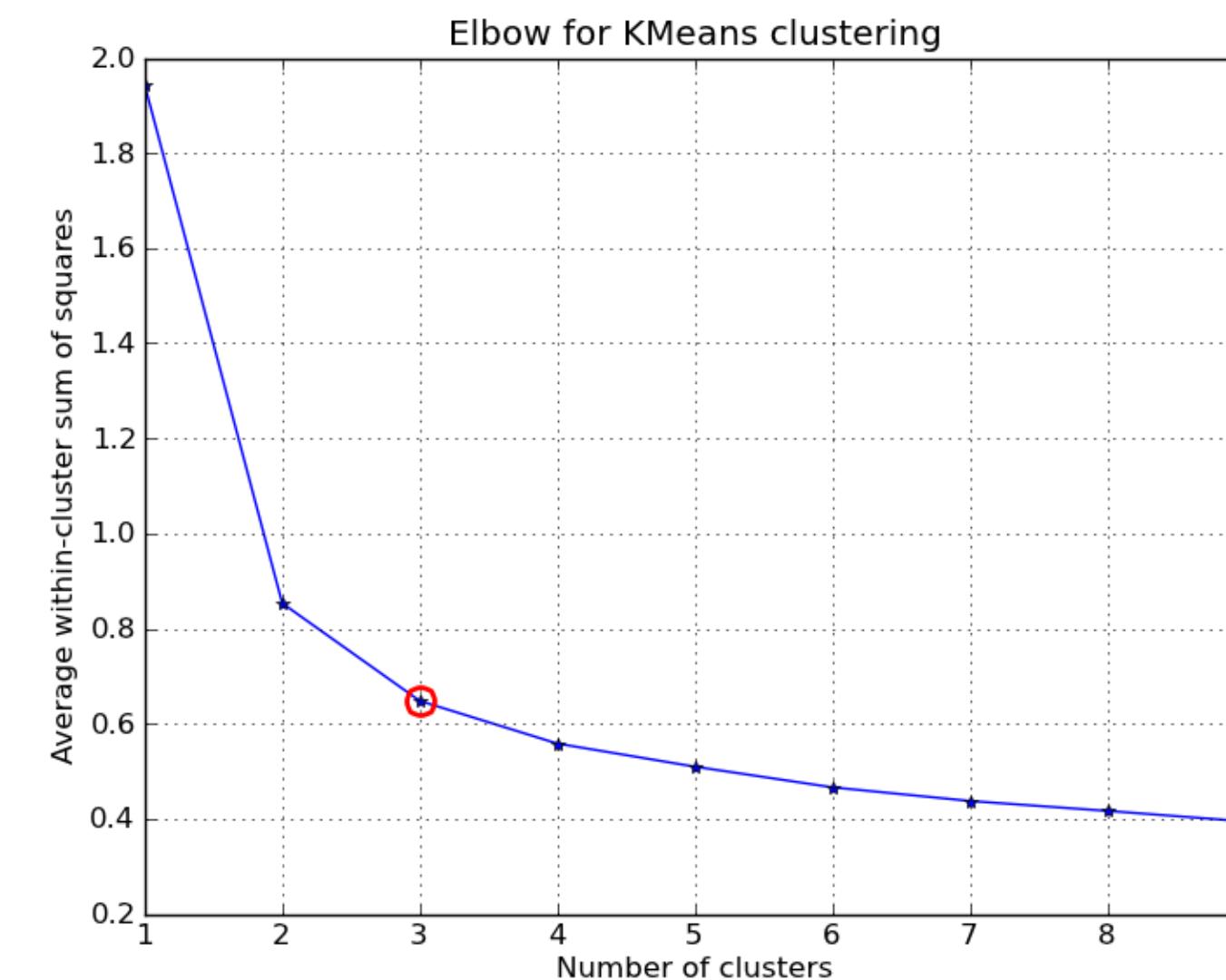


K-means Clustering은 군집 수 K를 사람이 직접 결정해야하는 문제가 있으며, 이는 Automation 관점에서 볼 때 문제가 있음



Choosing K in K-means Clustering

- K-means Clustering에서 Cluster 수 (K)는 사용자가 직접 정해줘야함.
- Cluster 수, K를 결정하는 것은 모델의 성능을 결정하는 중요한 요소임.
- Cluster 수 (K) 결정은 Distortion Function 최소값과 K 크기의 Trade-off 문제임



- Cluster 수 (K) 결정 방법
 1. Elbow method
 2. Using the market criterion

- $J(K)$ 는 K 증가에 따라 단조 감소함
- K 가 크면 클수록 $J(K)$ 값은 감소
- K 가 증가에 따라 현실적인 비용 발생

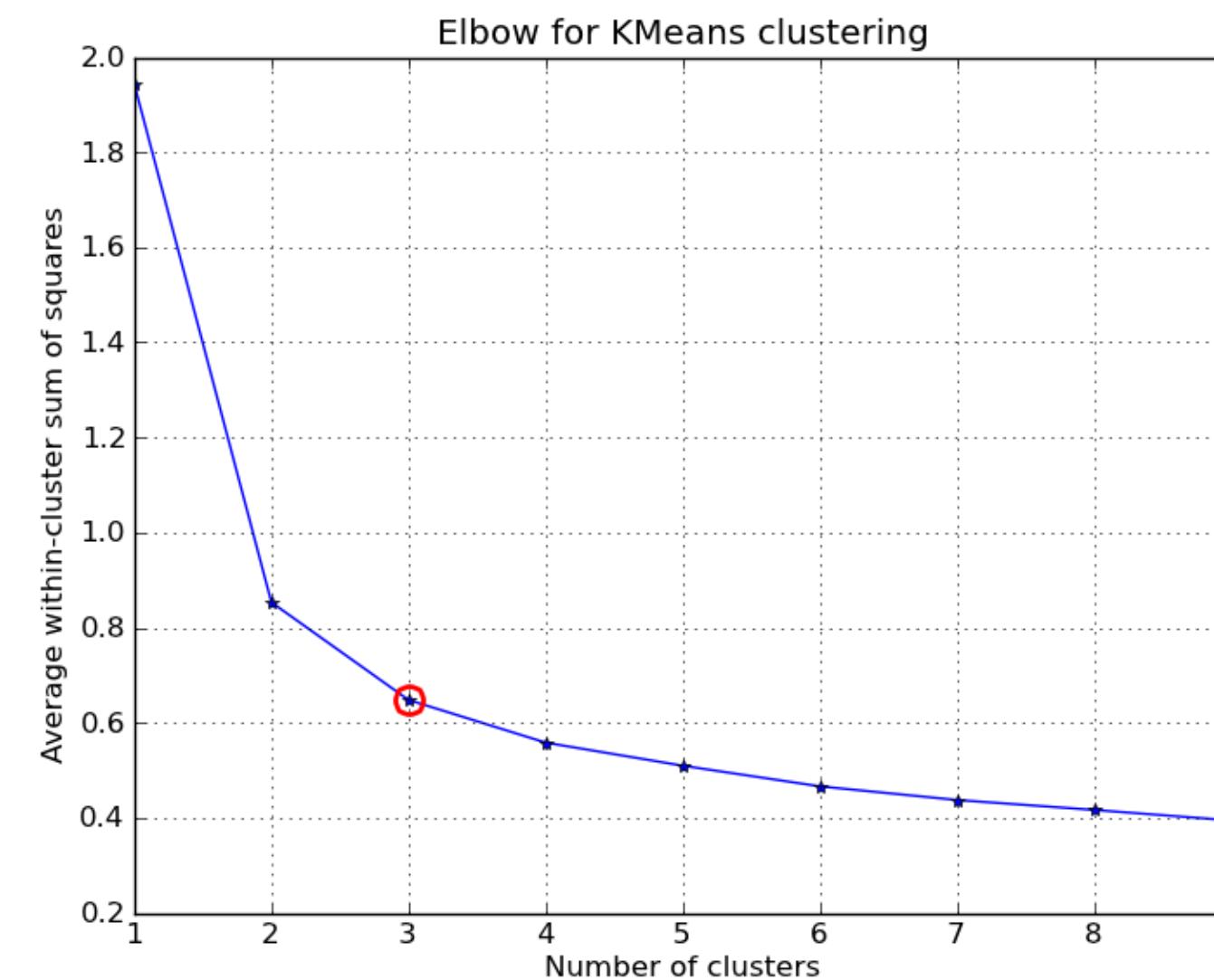
Selecting K: Elbow method

□ Elbow Method

- 다양한 K에 대하여 K-means Clustering을 진행하고 $J(K)$ 를 계산함
- $J(K)$ 가 급격히 줄어드는 것을 확인하고, K 증가에 따라 그 변화가 미미한 지점을 찾아 K를 직접 결정해주는 방법

□ Plot of the (K vs $J(K)$)

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

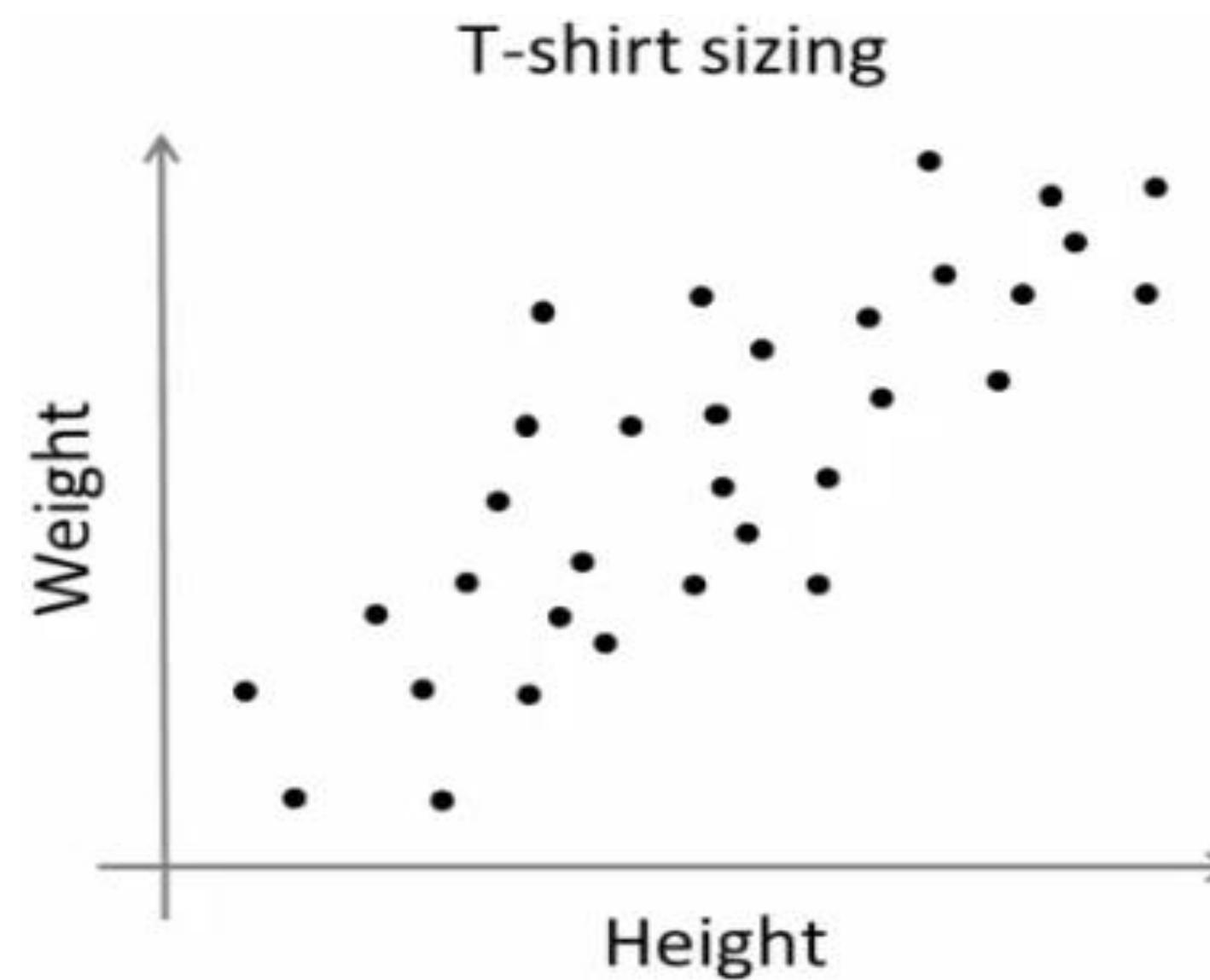


- ## □ 문제점: $J(K)$ 의 변화가 위의 그림처럼 뚜렷하게 나타나지 않는 경우, K를 선정하는 것이 애매모호함

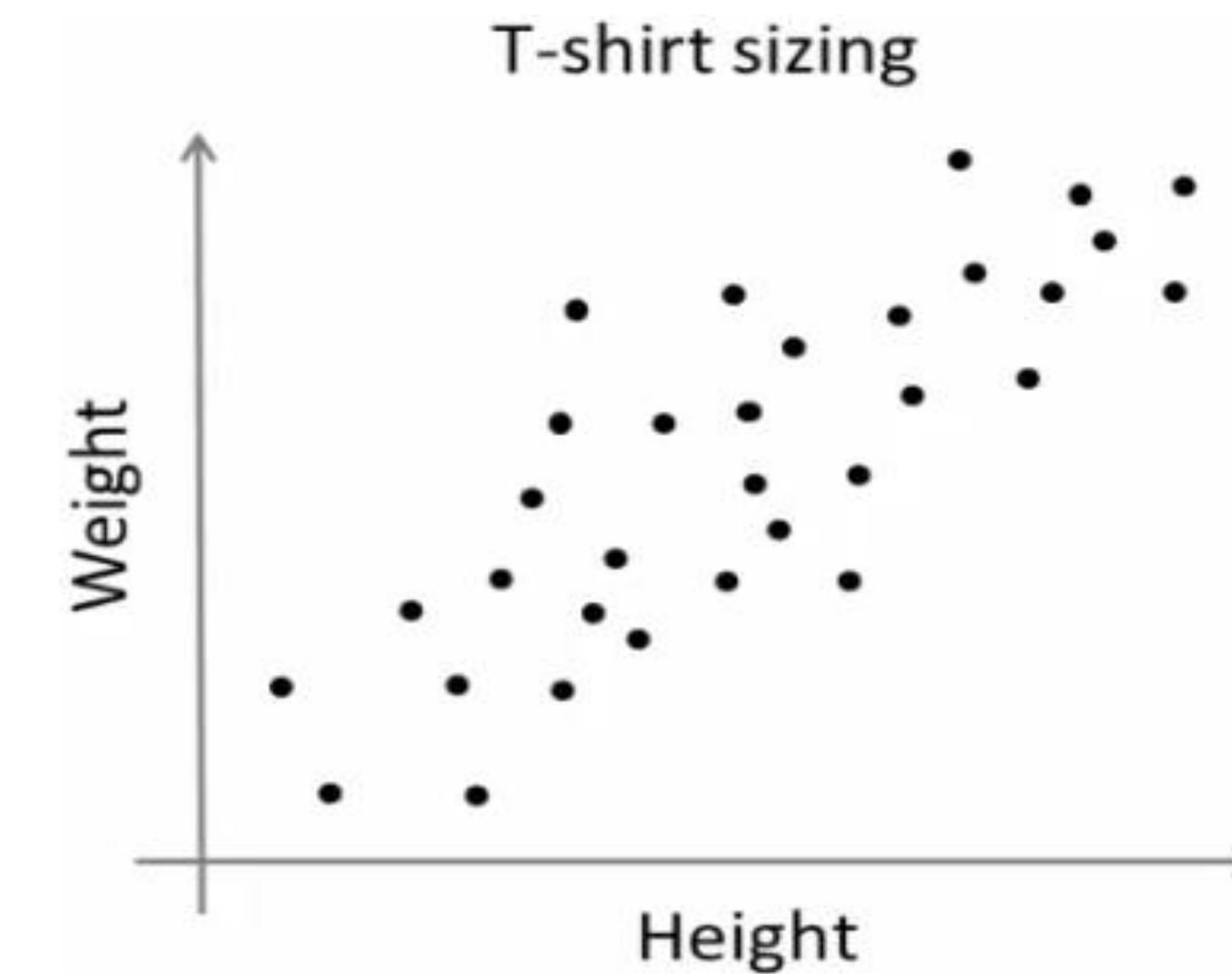
Using the market criterion

- K를 결정하는데 있어, 해당 데이터의 특성을 고려하여 일반적으로 통용되는 군집 수를 채택
- Ex) T-shirt size

1) 3size(S,M,L)



2) 5 size(XS,S,M,L,XL)



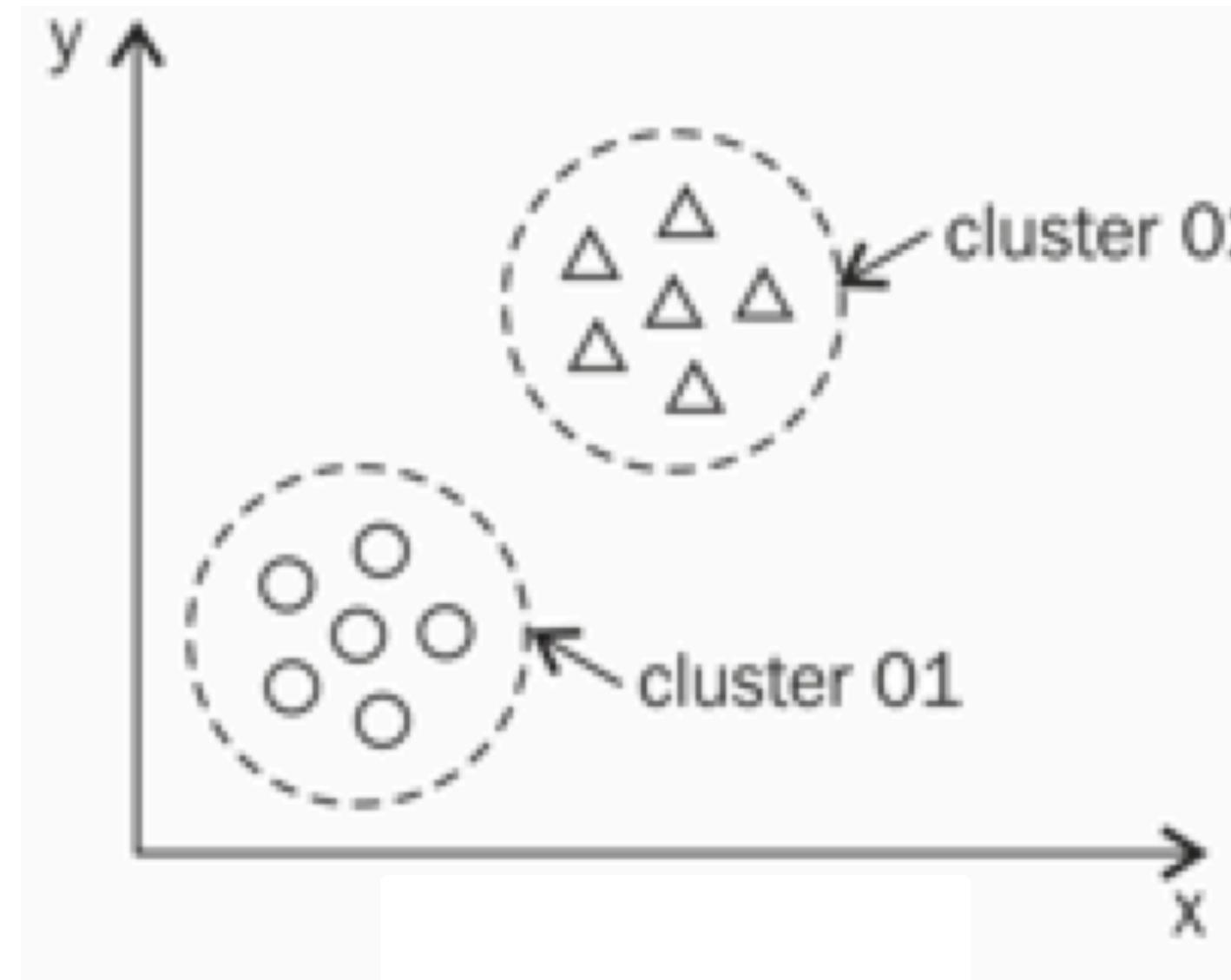
→Market segmentation을 위해 K-means 사용

Fuzzy k-means Clustering

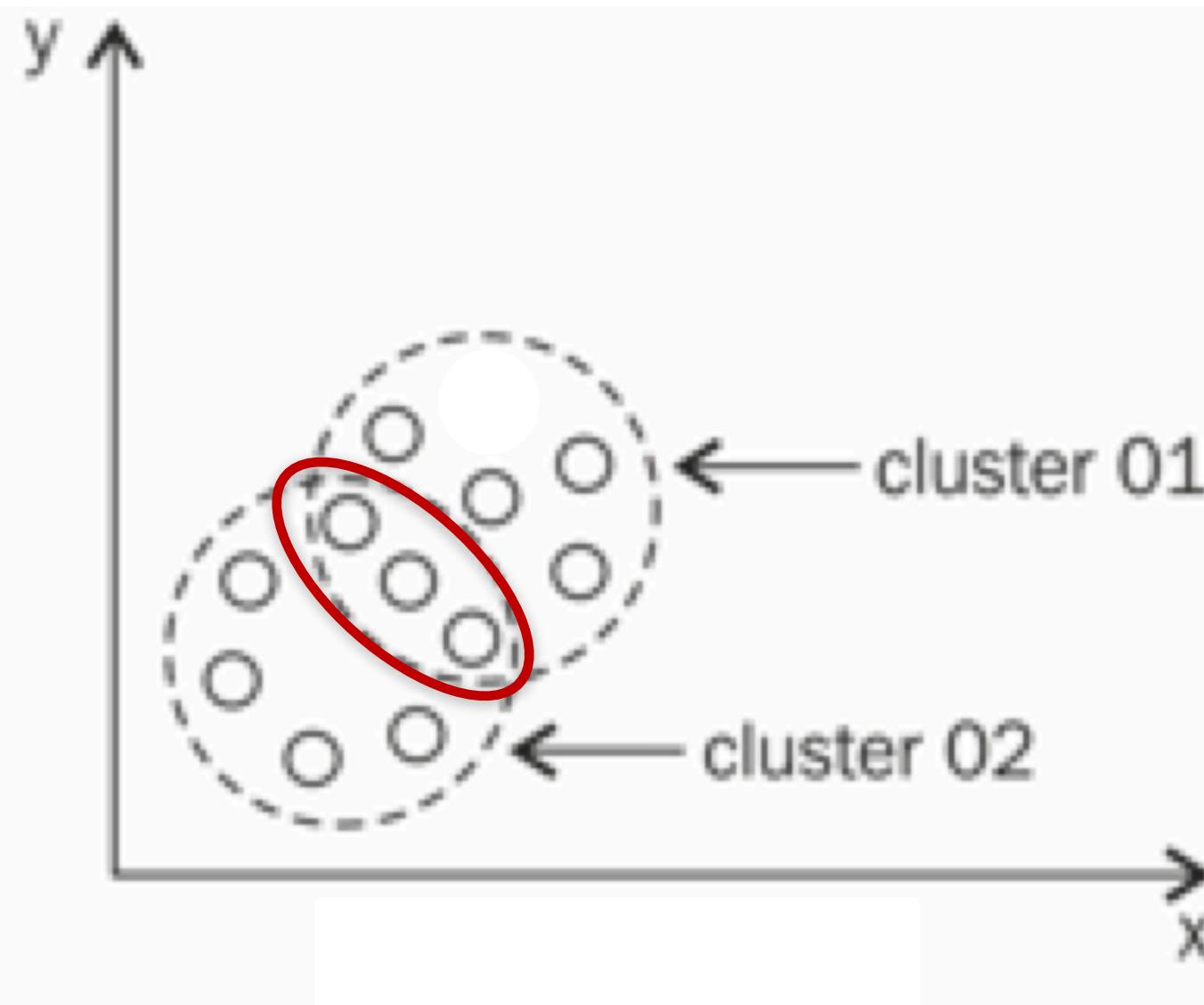
Fuzzy k-means Clustering: Introduction

- K-means clustering에서 centroids update는 해당 군집에 할당된 데이터에 의해서만 이루어짐
- 데이터의 위치가 군집 사이의 경계선에 있을 경우, 해당 데이터가 어느 군집에 포함되었느냐에 따라 계산 결과가 달라질 수 있음
- 초기 centroids에 따라 계산 결과가 달라지는 문제가 심화될 수 있음

<K-means clustering 효과적 사례>



<K-means clustering 비효과적 사례>

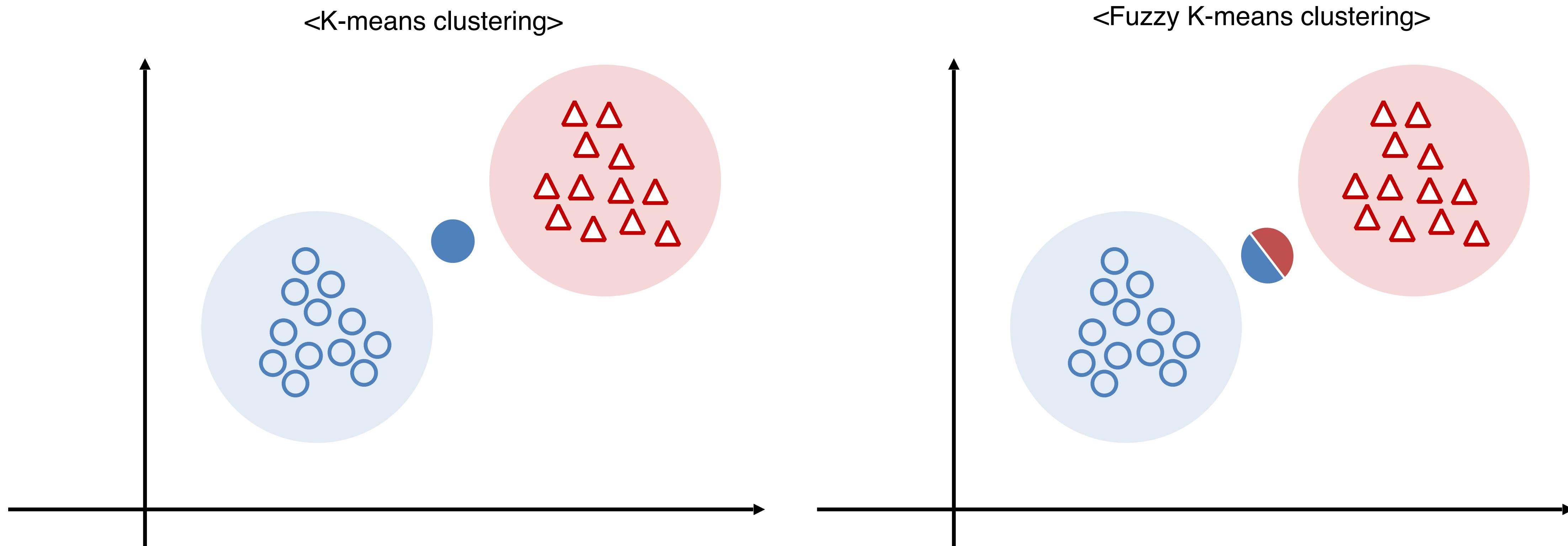


- 각 군집 경계선에 걸쳐 있는 관측치들이 존재.
- 경계선 주위의 관측치를 분류하는 기준의 ‘애매함’ 존재.
- 시작 기준에 따라서 잘못된 분석을 할 수가 있음.
- 군집 양상이 뚜렷하지 않은 데이터에 대해 효과적으로 군집화하기 위한 추가적인 방법 필요.

Fuzzy k-means Clustering: Introduction

▣ Fuzzy K-means Clustering

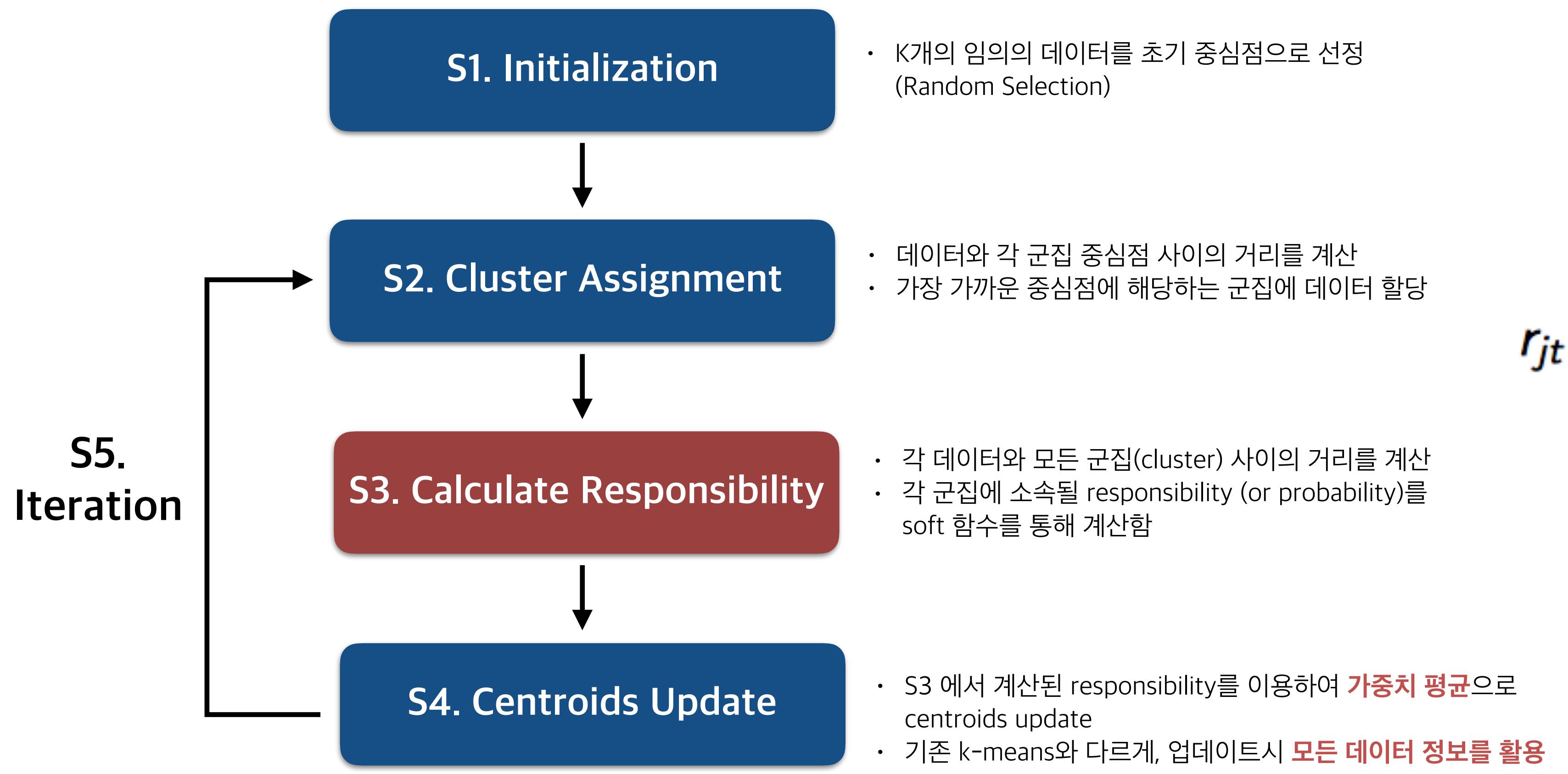
- 각 관측치들이 군집에 할당될 확률 (responsibility)를 계산하여 군집화를 진행
- ‘애매모호’한 관측치의 **모호한 정도**를 고려하여 계산.
- 데이터에 대한 정보를 전혀 가지고 있지 않은 경우 주로 사용



Algorithm

- Fuzzy K-means Clustering 알고리즘은 Centroids Update 계산 방식에서 차이가 있음
- 각 데이터와 모든 Cluster Centroids 사이의 거리를 바탕으로 Responsibility를 계산함

<Fuzzy K-means Clustering Algorithm>



< Cluster Responsibility >

$$r_{jt} = \frac{\exp \left\{ -\beta \| \mathbf{x}_t - \boldsymbol{\mu}_j \|^2 \right\}}{\sum_l \exp \left\{ -\beta \| \mathbf{x}_t - \boldsymbol{\mu}_l \|^2 \right\}}.$$

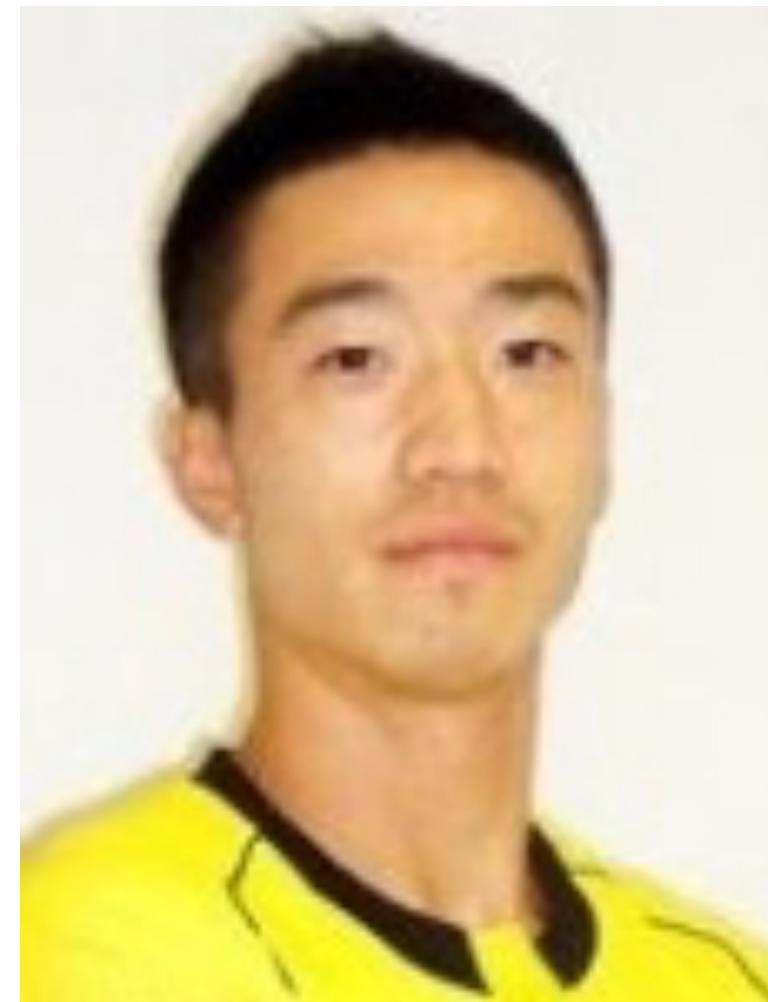
< Centroids Update >

$$\boldsymbol{\mu}_j = \frac{\sum_{t=1}^N r_{jt} \mathbf{x}_t}{\sum_{t=1}^N r_{jt}}.$$

Curse of Dimensionality

Introduction

□ 왜 차원의 수를 줄여야하는 상황이 생길까? 어떤 문제가 생길까?



고차원 축구선수

출생 1986년 4월 30일, 서울특별시

신체 169cm, 69kg

소속팀 수원 삼성 블루윙즈 (MF 미드필더)

학력 아주대학교 학사

데뷔 2009년 전남 드래곤즈 입단

경력 2013.07~ 수원 삼성 블루윙즈

사이트 [공식사이트](#)

Q [수원삼성의 고차원 선수 조기축구회 선수인가요?](#) 2015.05.16.

수원과 제주 경기를 보고 있는데요. 고차원 선수가 오른쪽 윙으로 나왔는데 혹시 이 선수 조기축구회 선수인가요? 어떻게 프로선수가 된거죠?

A 생활체육에서는 사실 선수들과 레벨이 틀려서 프로로 가기는 현실적으로 힘들죠.. 고차원의 경우는 인터넷상에서 확인되는 경력만 놓고본다면, 아주대-전남-상주를 거쳐서 수원으로 오게 되었네요..

구기스포츠, 레저 > 축구 > 축구 선수, 감독 | 1 · 0

□ 왜 차원의 수를 줄여야하는 상황이 생길까? 어떤 문제가 생길까?

□ 많은 특징들을 이용하는 경우는 생각보다 많음

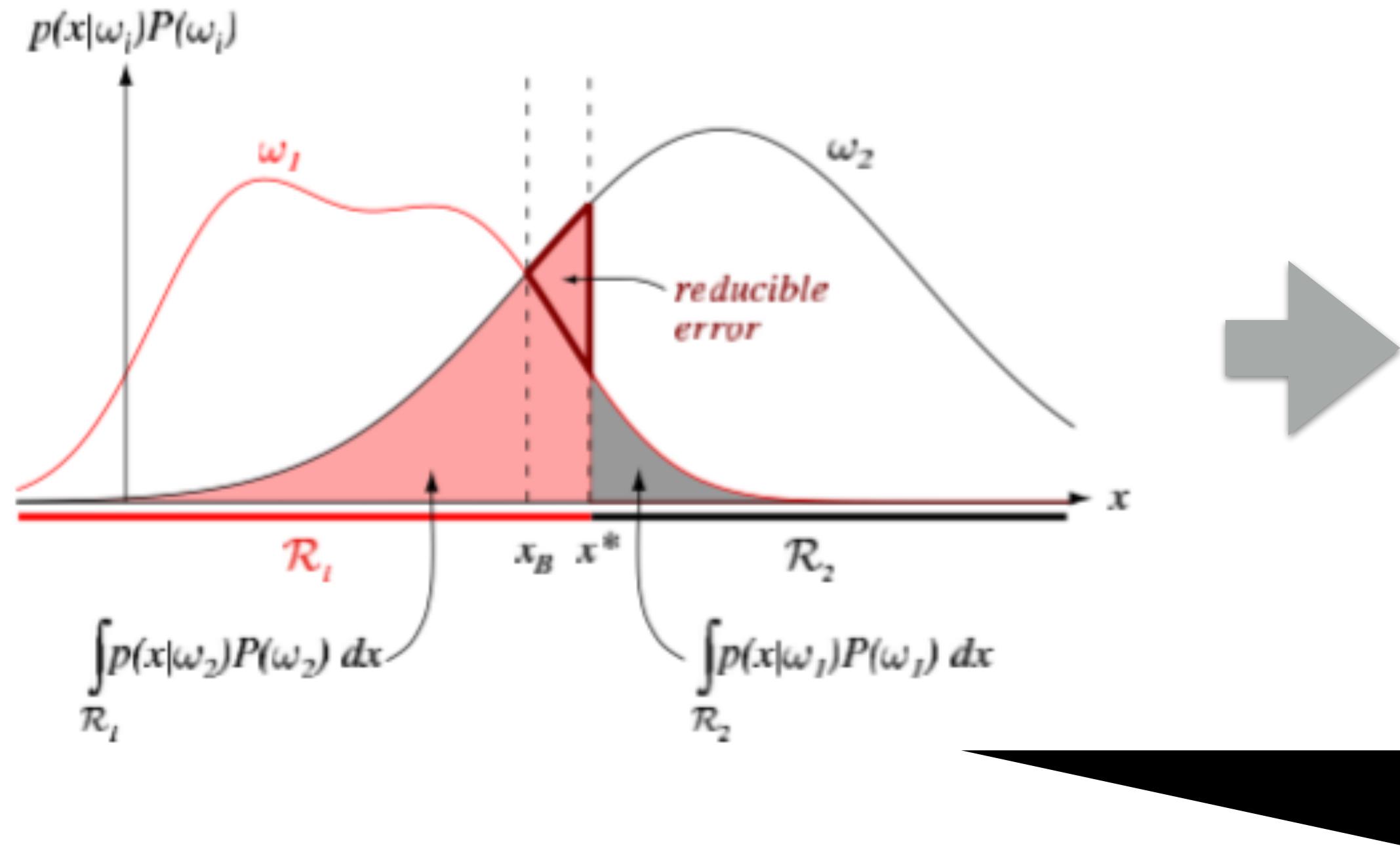
□ 차원의 수가 너무 클 경우, **두가지 문제**가 발생할 수 있음

- Error can increase (Computation Time)

- Training Data Scale

Problem of Error with High Dimensionality

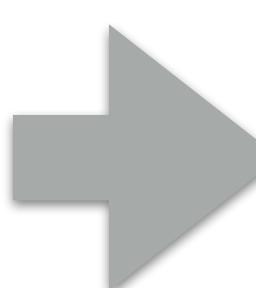
- 이론적으로 볼 때, 차원의 수가 증가할수록 에러는 작아져야 함
- 각 클래스의 사전 확률이 같고, 특징의 분포가 정규분포라고 가정할 때, 에러 확률은 마할라노비스 거리로 표현될 수 있음



$$P(\text{error}) = \int_{\mathcal{R}_2} p(\mathbf{x}|\omega_1)P(\omega_1) d\mathbf{x} + \int_{\mathcal{R}_1} p(\mathbf{x}|\omega_2)P(\omega_2) d\mathbf{x}$$

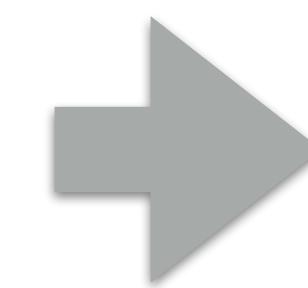
$p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \Sigma)$, and $P(\omega_i) = \frac{1}{2}$, $i = 1, 2$,

$$P(e) = \frac{1}{\sqrt{2\pi}} \int_{\frac{r}{2}}^{\infty} e^{-\frac{1}{2}u^2} du$$



$$\Sigma = \begin{bmatrix} \sigma_1^2 & & 0 \\ & \ddots & \\ 0 & & \sigma_d^2 \end{bmatrix}$$

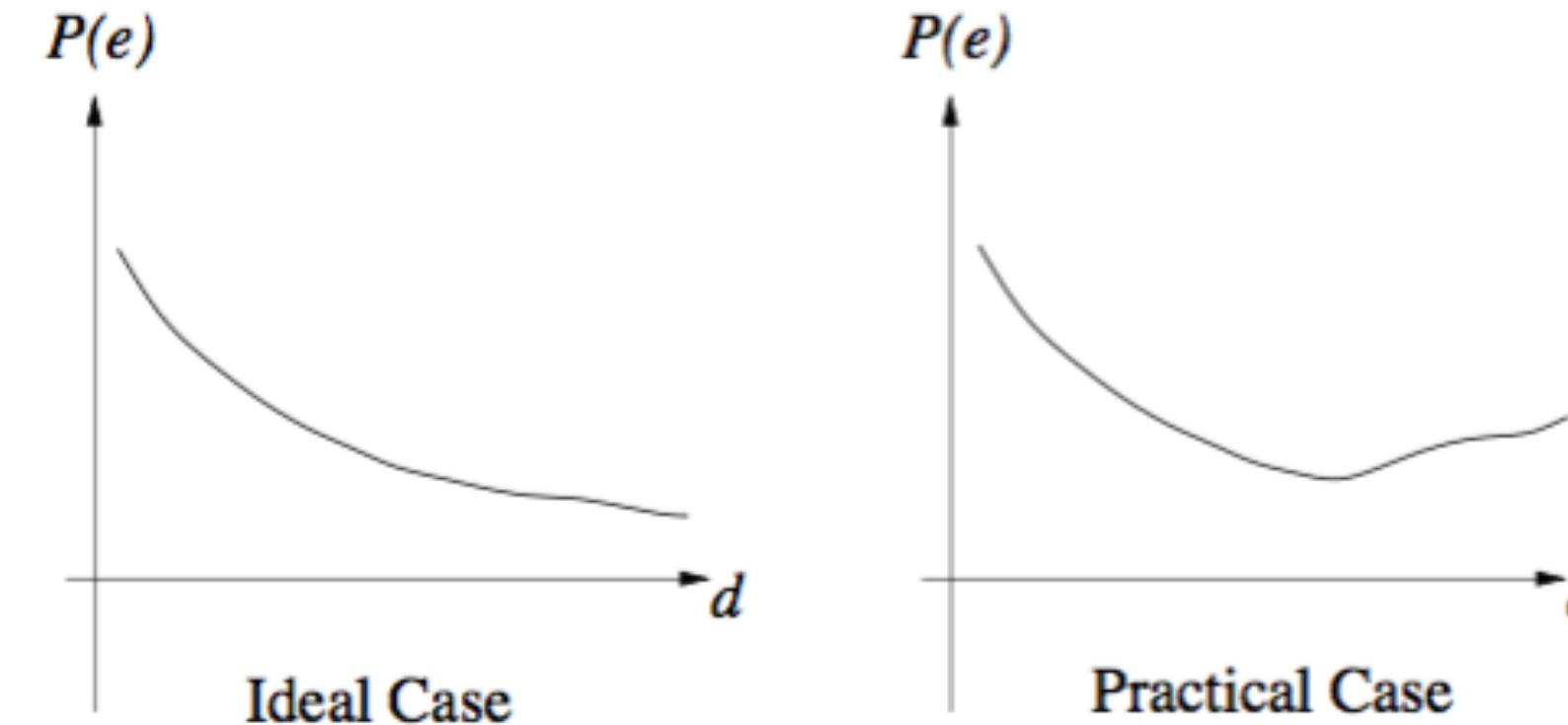
$$\gamma^2 = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = \sum_{i=1}^d \left(\frac{\mu_{i1} - \mu_{i2}}{\sigma_i} \right)^2$$



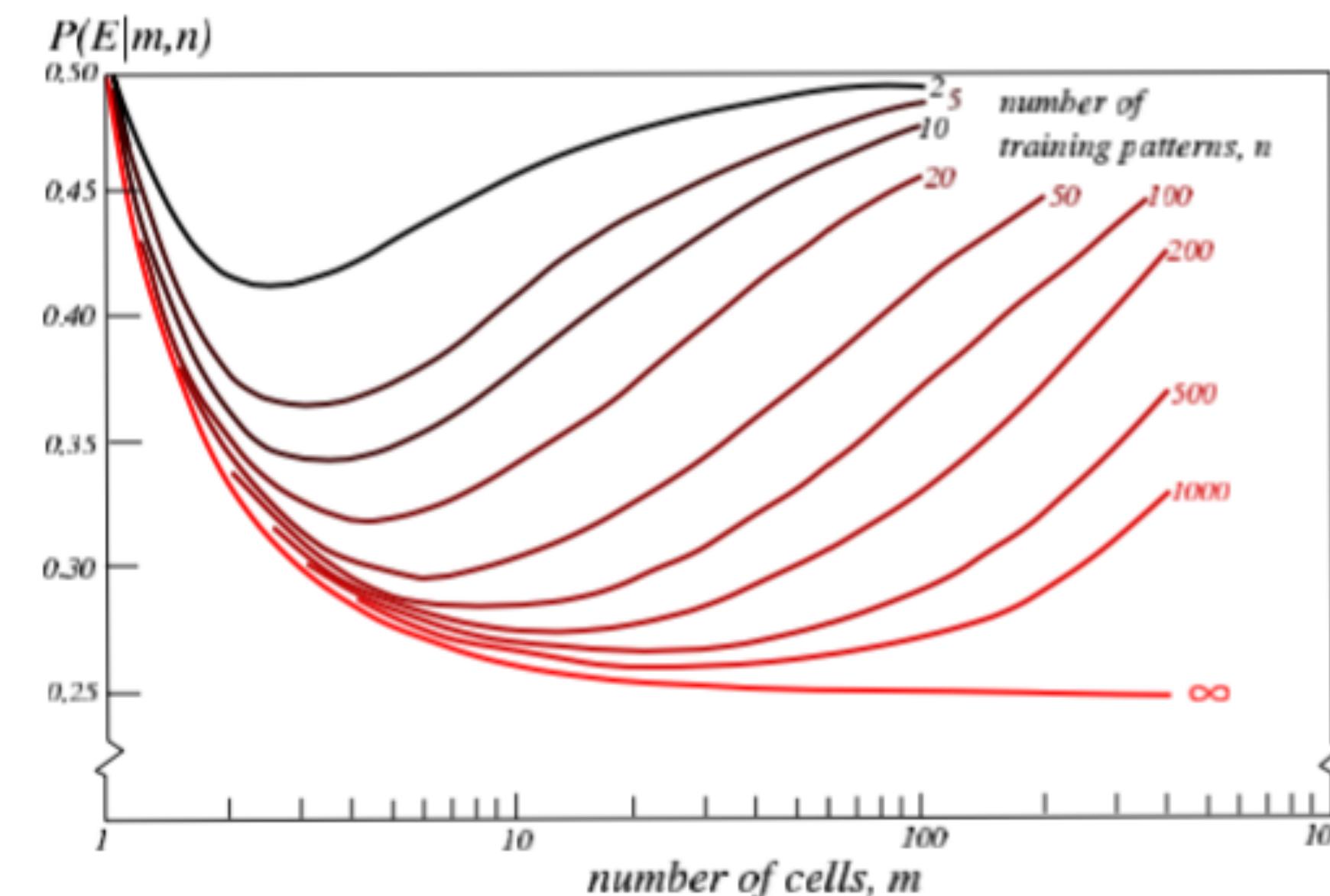
$$\lim_{\gamma \rightarrow \infty} P(e) = 0.$$

Problem of Error with High Dimensionality

- 실제 문제에서 차원을 증가하면 증가할수록 오히려 에러가 커지는 상황이 발생함



- 모순적인 상황은 훈련 샘플(Training Sample)의 수가 한정적이기 때문

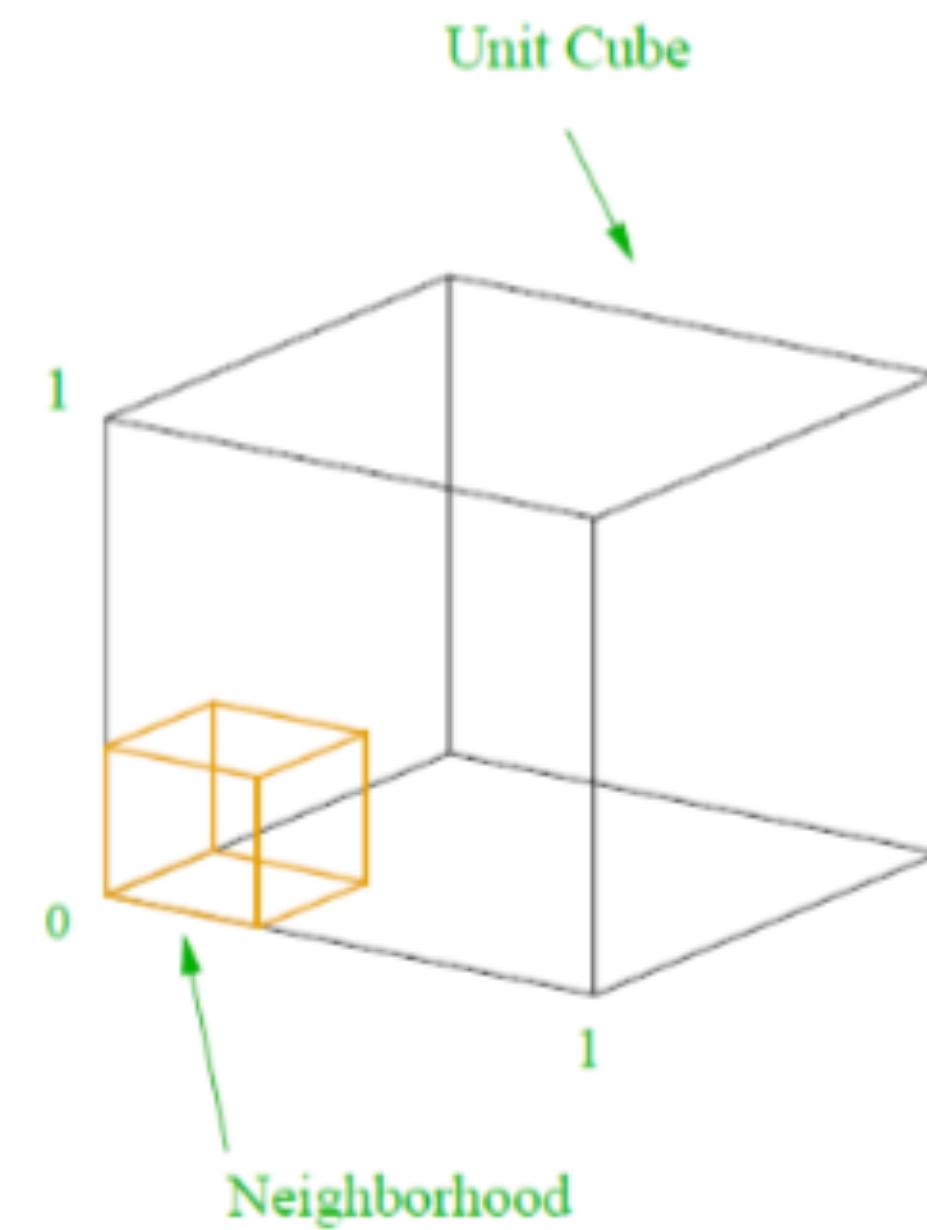


Problem of Training Set

▣ 훈련 데이터의 분포가 단위 길이의 Hypercube에 균등하게 퍼져있다고 생각해보자

- Hypercube의 전체 부피는 총 확률 값으로 1을 의미함
- Hypercube의 차원은 특징의 수(Dimensionality)를 의미함

▣ Q) 각 특징에 대한 정보의 양과 전체 정보량 사이의 관계는?



$$L = (0.01)^{1/10} = 0.63.$$

$$10\% : \rightarrow (0.1)^{1/10} = 0.8$$

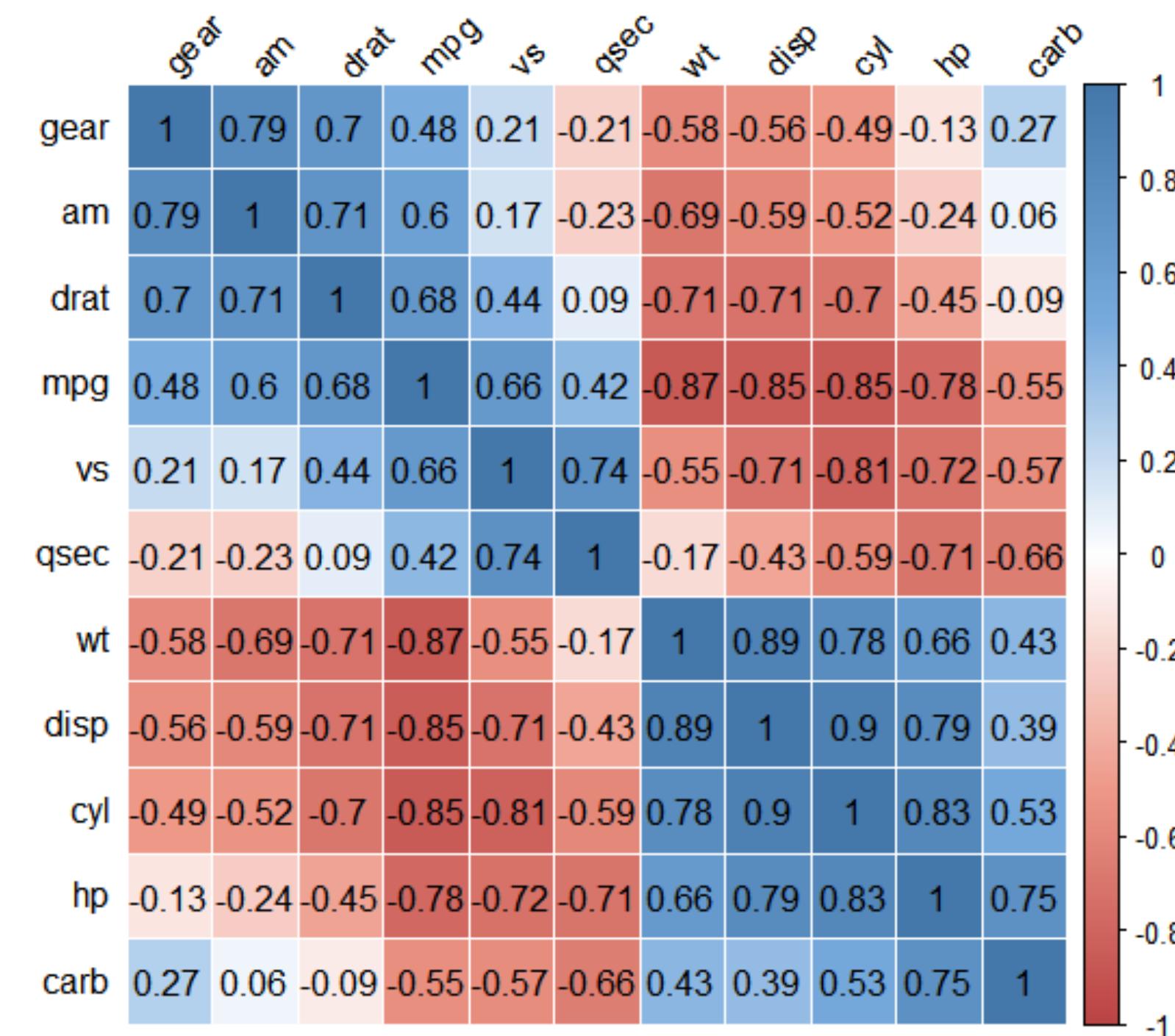
- 전체 데이터의 1% 정보를 알기 위해서 각 차원의 정보가 63%씩 필요
- 전체 데이터의 10% 정보를 알기 위해서 각 차원의 정보가 80% 필요

This is why Dimensionality Reduction is Important!
(because # of data is not infinite...)

Common Factor Analysis

□ 공통 요인 분석

- Correlation Matrix를 구한 후 상관계수가 높은 두 요인을 합치는 것을 반복
- 변수의 차원을 줄이는 것에 주요 목표

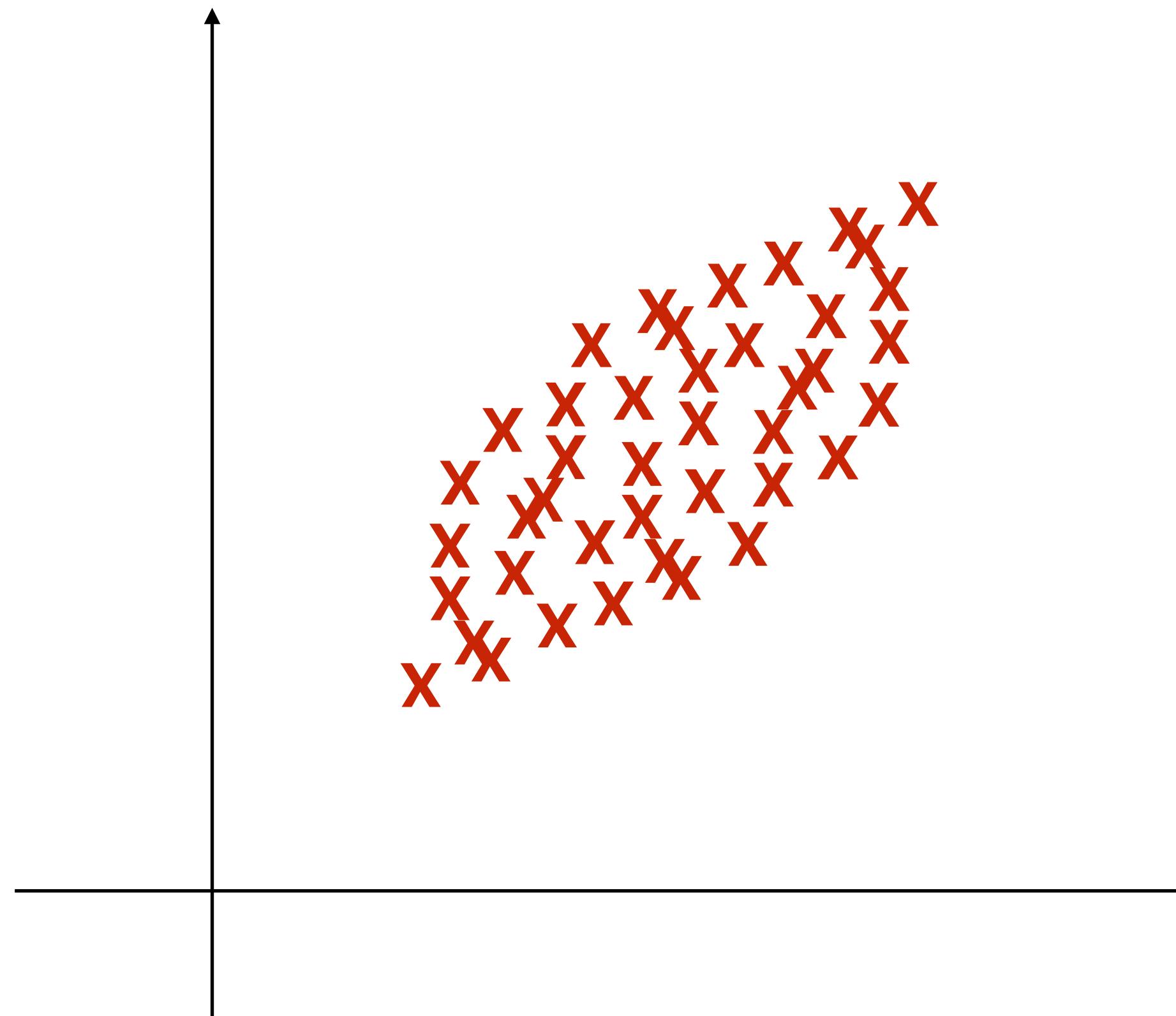


Principal Component Analysis

Principal Component Analysis

□ 주성분분석(Principal Component Analysis, PCA)

- 제시된 변수들의 **선형 조합(Linear Combination)**으로 이루어진 변수를 통해 적은 양의 변수(주성분)으로 전체 변동을 설명하는 방법
- 전체 변동(variation)을 가장 잘 설명하는 성분을 순차적으로 뽑아냄



데이터가 포함하고 있는 정보(Information)을
데이터의 분산(Variance)로 해석

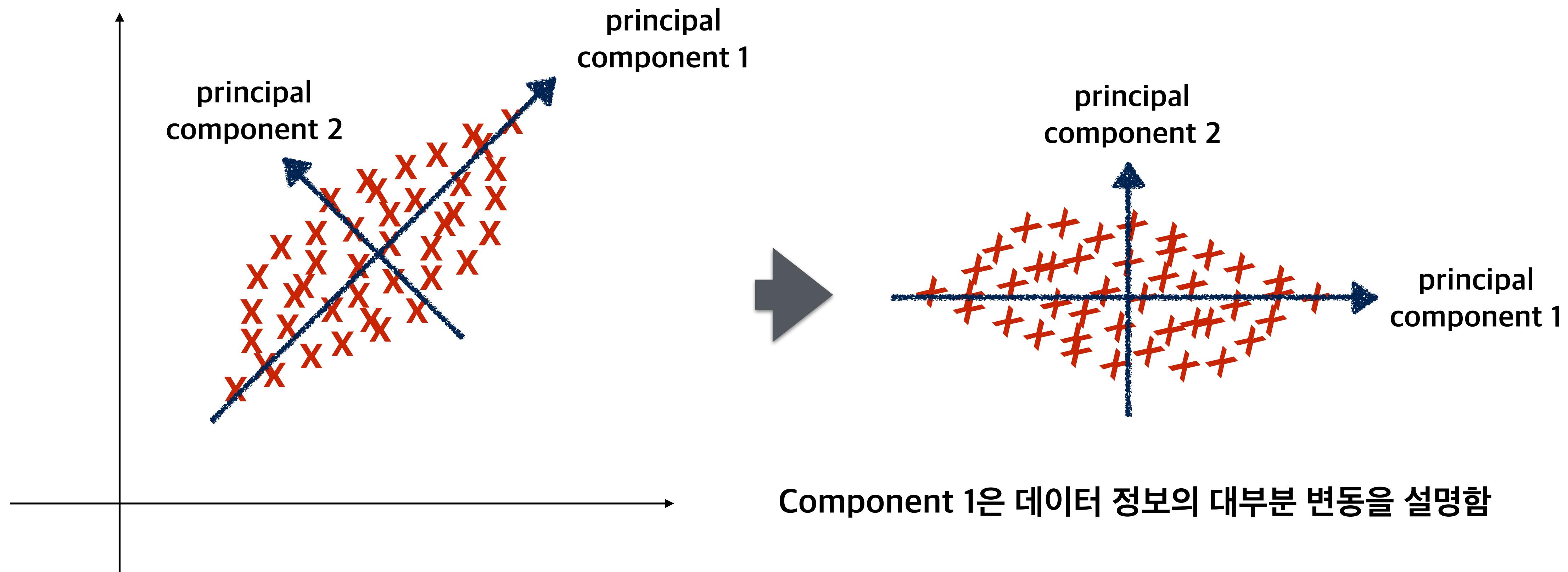


데이터의 분산을 최대화하는 방향으로
차원을 축소

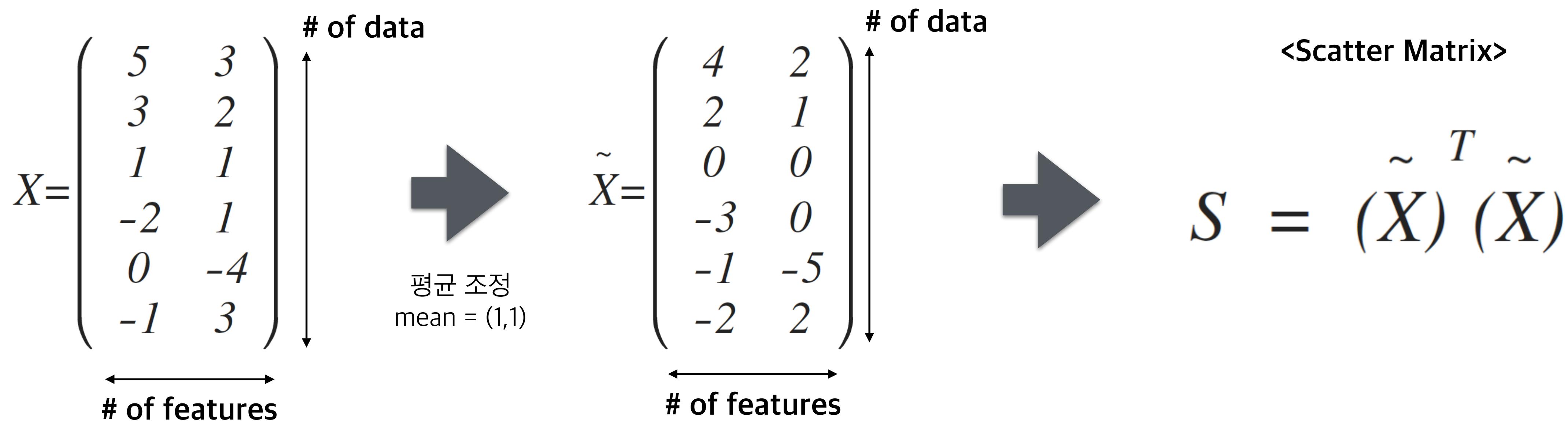
Principal Component Analysis

□ 주성분분석(Principal Component Analysis, PCA)

- 제시된 변수들의 **선형 조합(Linear Combination)**으로 이루어진 변수를 통해 적은 양의 변수(주성분)으로 전체 변동을 설명하는 방법
- 전체 변동(variation)을 가장 잘 설명하는 성분을 순차적으로 뽑아냄



- 1) 데이터 행렬 X 를 특징에 따라 평균 조정한 후, 산포 행렬(Scatter Matrix)을 구함
- 2) 산포 행렬, S 를 Spectral Decomposition을 통해 분해하고 주성분을 구함



- 1) 데이터 행렬 X 를 특징에 따라 평균 조정한 후, 산포 행렬(Scatter Matrix)을 구함
- 2) 산포 행렬, S 를 Spectral Decomposition을 통해 분해하고 주성분을 구함 (표기상 편의를 위해 X 로 표현)

<Spectral Decomposition of Scatter Matrix>

$$X^T X = P \Lambda P^T$$

↔ (Symmetric Matrix) ↓ → Eigenvector Matrix
 Eigenvalue Matrix

- 1) 데이터 행렬 X 를 특징에 따라 평균 조정한 후, 산포 행렬(Scatter Matrix)을 구함
- 2) 산포 행렬, S 를 Spectral Decomposition을 통해 분해하고 주성분을 구함 (표기상 편의를 위해 X 로 표현)

<Spectral Decomposition of Scatter Matrix>

$$X^T X = P \Lambda P^T$$

↔ (Symmetric Matrix) ↓ → Eigenvector Matrix

Eigenvalue Matrix

$$P^T X^T \boxed{XP} = \Lambda$$

$$T^T T = \Lambda$$

Linear Projection of X on Eigenspace

$$T = XP$$

- 1) 데이터 행렬 X 를 특징에 따라 평균 조정한 후, 산포 행렬(Scatter Matrix)을 구함
- 2) 산포 행렬, S 를 Spectral Decomposition을 통해 분해하고 주성분을 구함 (표기상 편의를 위해 X 로 표현)

<Spectral Decomposition of Scatter Matrix>

$$X^T X = P \Lambda P^T$$

↔ (Symmetric Matrix) ↓ → Eigenvector Matrix
Eigenvalue Matrix

$$P^T X^T X P = \Lambda$$

$$T^T T = \Lambda$$

Linear Projection of X on Eigenspace

$$T = X P$$

Scatter Matrix of Projected Data

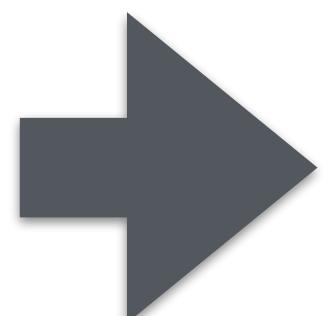
- Eigenspace로 투영된 데이터의 변동(variation)의 합은 기존 데이터 산포 행렬의 고유값의 합과 같음
- 각 고유벡터가 설명하는 변동은 이에 대응되는 고유값의 크기와 같음

Linear Projection of X on Eigenspace

$$T = X P$$

$$P^T X^T X P = \Lambda$$

$$T^T T = \Lambda$$



(Don't forget)

- # of eigenvector = min(# of data, # of features)
- Eigenvectors are orthonormal (inner product = 0)

What we want to do is **dimensionality reduction!**

- Eigenspace로 투영된 데이터의 변동(variation)의 합은 기존 데이터 산포 행렬의 고유값의 합과 같음
- 각 고유벡터가 설명하는 변동은 이에 대응되는 고유값의 크기와 같음

<Dimensionality Reduction>

1) Choose m number of eigenvector



2) Project data X with selected ones

of X's features (in general)

$$P = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & \dots & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

Selected vectors

$$P' = \begin{pmatrix} * & * \\ * & * \\ * & * \\ * & * \end{pmatrix}$$

- Eigenspace로 투영된 데이터의 변동(variation)의 합은 기존 데이터 산포 행렬의 고유값의 합과 같음
- 각 고유벡터가 설명하는 변동은 이에 대응되는 고유값의 크기와 같음

<Dimensionality Reduction>

1) Choose m number of eigenvector



2) Project data X with selected ones

of X's features (in general)

$$P = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & \dots & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}$$

Selected vectors

$$P' = \begin{pmatrix} * & * \\ * & * \\ * & * \\ * & * \\ * & * \end{pmatrix}$$

<차원 축소된 최종 데이터>

$$T' = X P'$$

Selection of # of Components

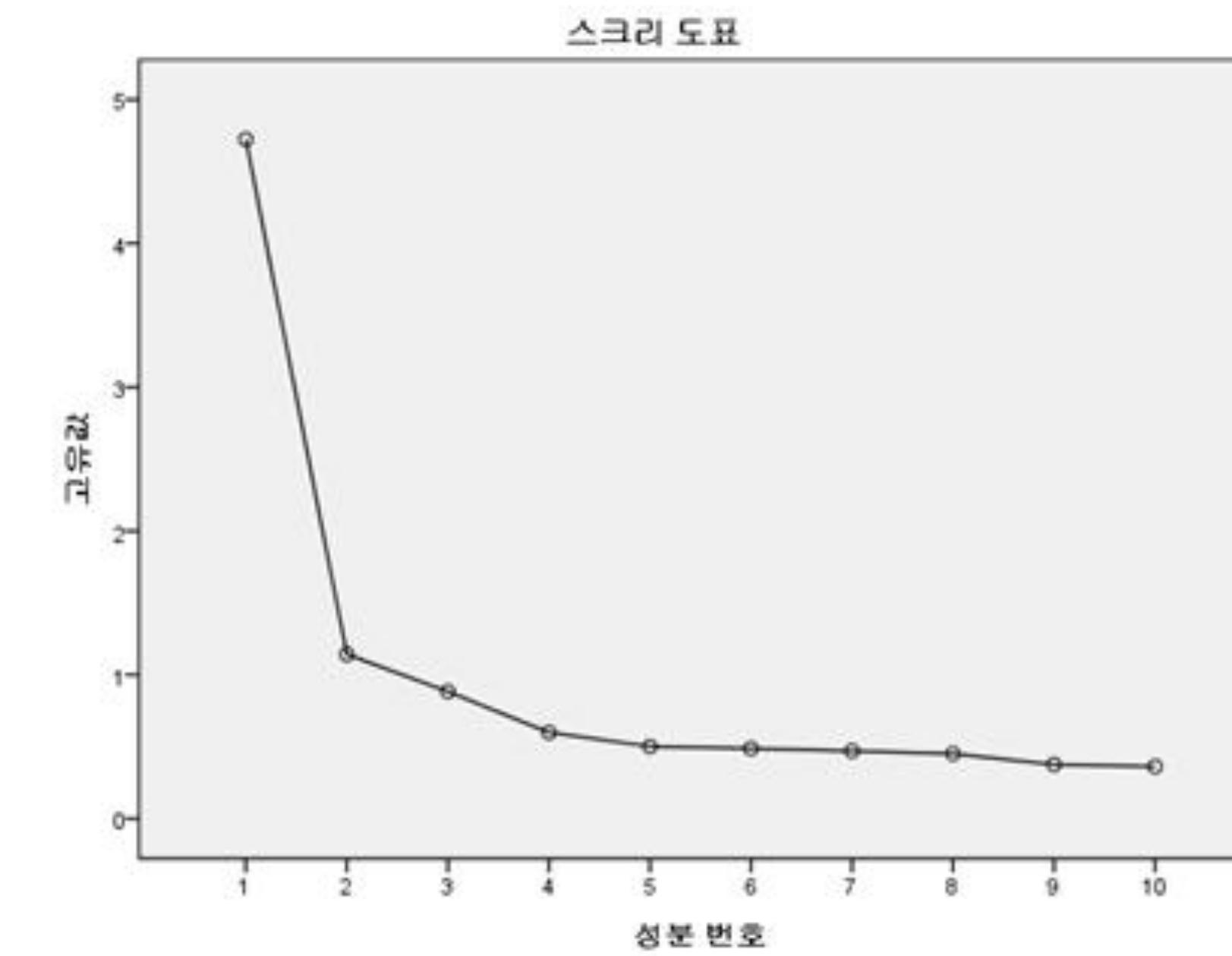
- 각 주성분이 전체 변동의 어느 정도를 설명하는지를 나타내는 척도인 기여율(Contribution) 사용
 - 기여율(Contribution) = (해당 고유값) / (전체 고유값의 합)
- 전체 변동의 70% 이상을 설명할 수 있도록 주성분의 개수를 설정하는 것이 일반적
- 내림 차순으로 고유값을 표시한 스크리 도표(Scree Plot)를 보고 적정한 개수를 선정하기도 함

<Using Contribution>

성분	초기 고유값		
	합계	% 분산	% 누적
1	4.724	47.242	47.242
2	1.143	11.433	58.675
3	.884	8.836	67.510
4	.599	5.992	73.502
5	.502	5.019	78.522
6	.487	4.874	83.396
7	.471	4.709	88.105
8	.451	4.511	92.615
9	.376	3.759	96.375
10	.363	3.625	100.000

추출방법: 주성분

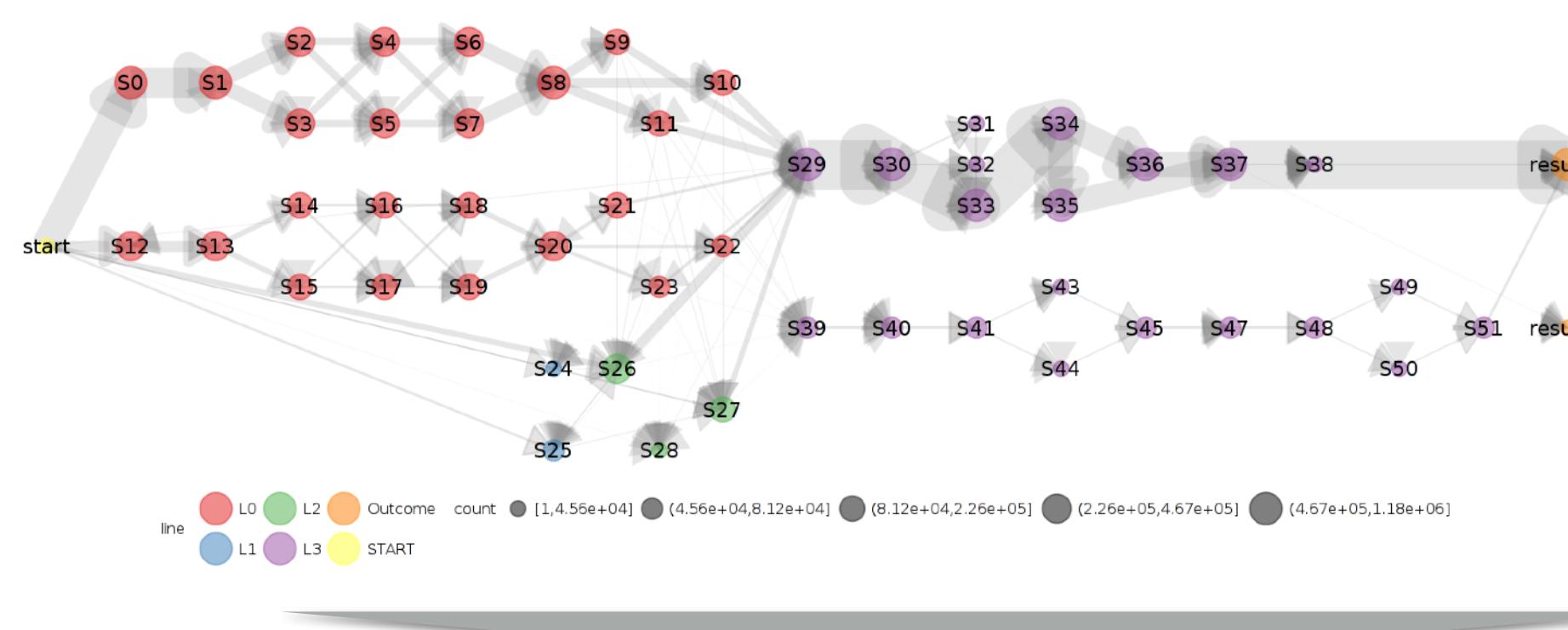
<Scree Plot>



Application of PCA: Bosch Production Line Performance (Kaggle)

- PCA는 일반적으로 데이터가 가지고 있는 특징(feature)의 수가 많은 경우 활용

<Bosch Production Line>



A	B	C	L	M	N	O	P	Q	R	AG	AH	AI	AJ
id	LO_S0_D1	LO_S0_D3	LO_S0_D21	LO_S0_D23	LO_S1_D26	LO_S1_D30	LO_S2_D34	LO_S2_D38	LO_S2_D42	LO_S3_D102	LO_S4_D106	LO_S4_D111	LO_S5_D115
4	82.24	82.24	82.24	82.24	82.24	82.24	82.24	82.24	82.24	82.26	82.26		
6													
7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7	1618.7				1618.72
9	1149.2	1149.2	1149.2	1149.2	1149.2	1149.2	1149.21	1149.21	1149.21		1149.22	1149.22	
11	602.64	602.64	602.64	602.64	602.64	602.64				602.64	602.66	602.66	
13	1331.66	1331.66	1331.66	1331.66	1331.66	1331.66				1331.67	1331.68	1331.68	
14													
16													
18	517.64	517.64	517.64	517.64	517.64	517.64	517.64	517.64	517.64	517.65	517.65		
23													
27	392.85	392.85	392.85	392.85	392.85	392.85				392.85			392.87
28	55.44	55.44	55.44	55.44	55.44	55.44				55.44	55.47	55.47	
31	98.99	98.99	98.99	98.99	98.99	98.99	99	99	99		99.01	99.01	
34	354.51	354.51	354.51	354.51	354.51	354.51	354.52	354.52	354.52				354.59
38	1633.8	1633.8	1633.8	1633.8	1633.8	1633.8	1633.8	1633.8	1633.8	1633.82	1633.82		
41													
44	1532.42	1532.42	1532.42	1532.42	1532.42	1532.42	1532.42	1532.42	1532.42				1532.44

Feature 개수가 너무 많고,
각각의 Feature를 해석하기 힘듦

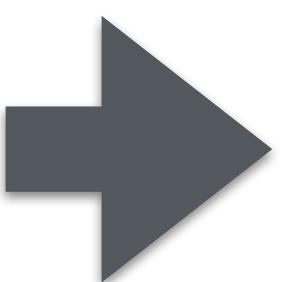
Principal Component Analysis

Classifier

결과 도출

Application of PCA: EigenFace

- 이미지 처리 분야에서도 PCA를 활발하게 사용
- 작은 차원의 특징을 바탕으로 이미지 처리 가능



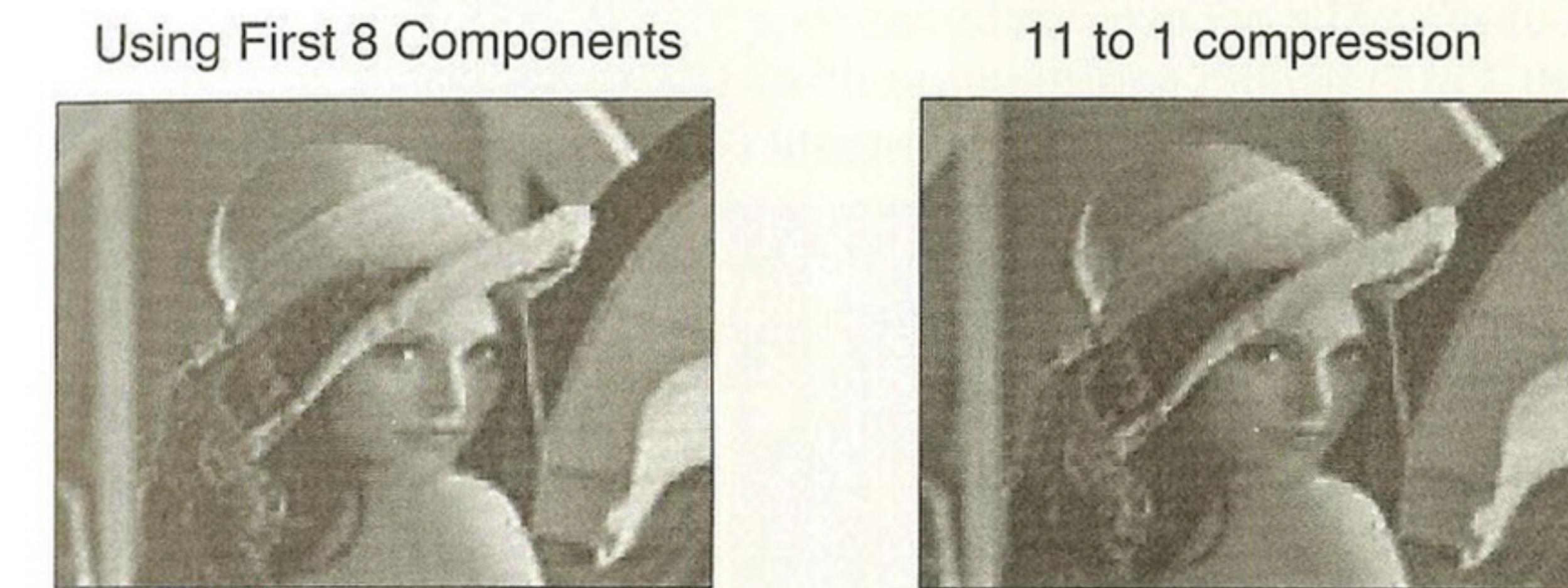
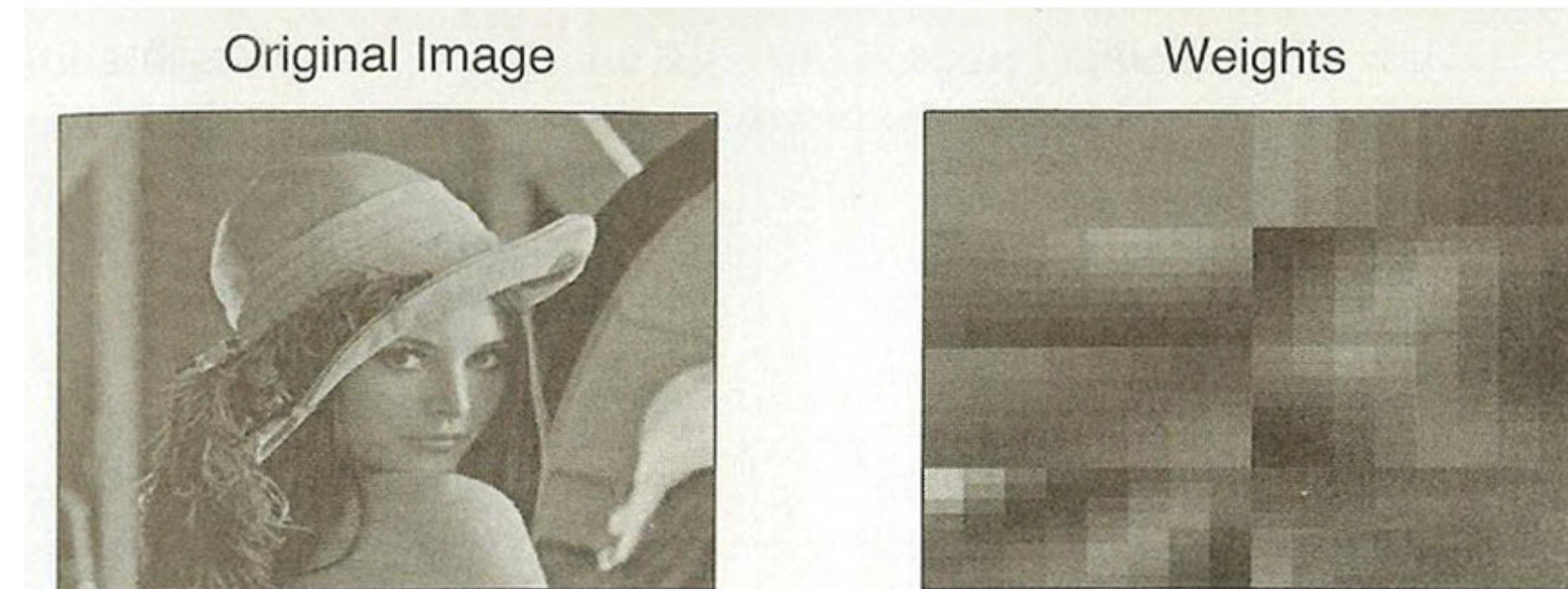
Principal Component Analysis

Reconstruction with
Principal Component

$$\text{Face} = a_1 \times \text{Eigenface}_1 + a_2 \times \text{Eigenface}_2 + a_3 \times \text{Eigenface}_3 + a_4 \times \text{Eigenface}_4 + \dots + a_n \times \text{Eigenface}_n$$

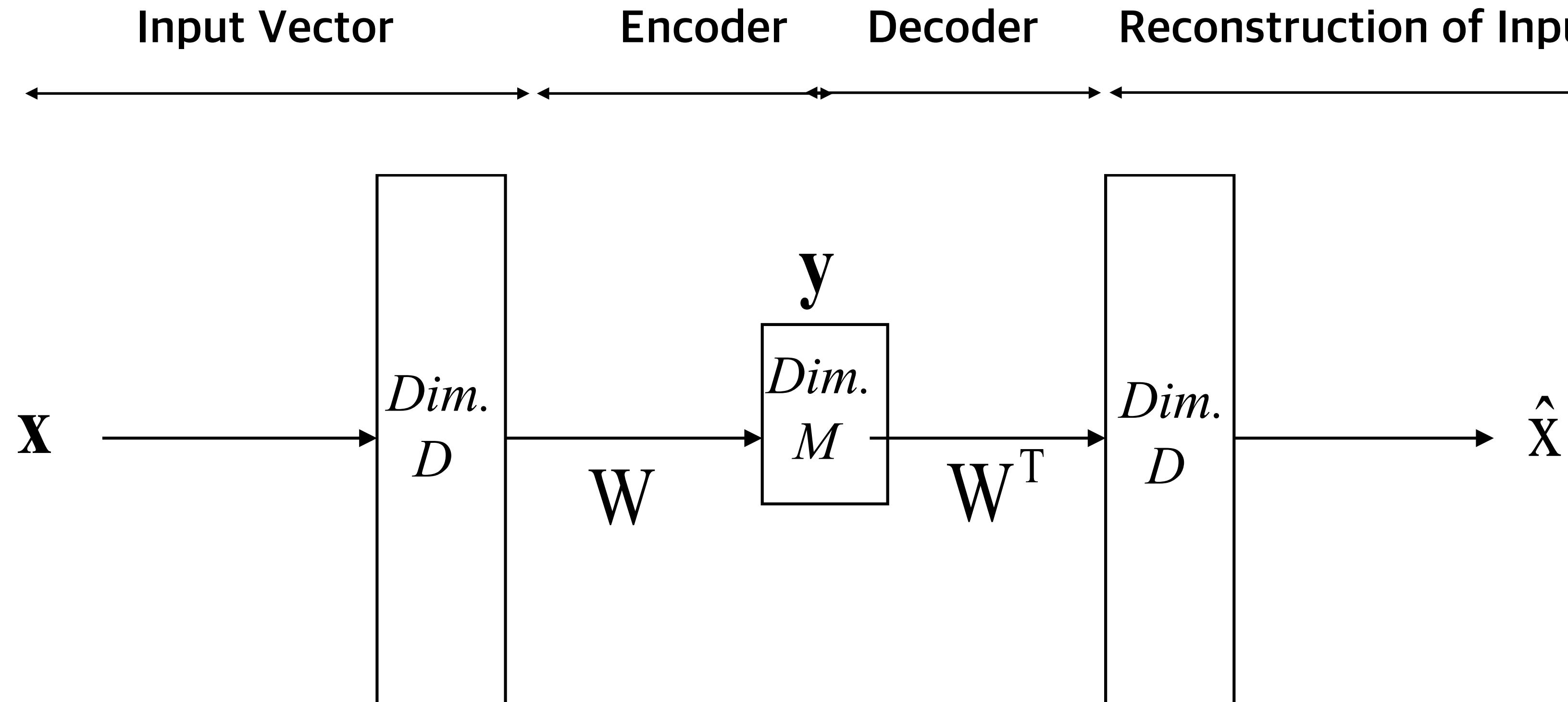
Application of PCA: Image Compression and Reconstruction

- PCA로 도출된 결과를 바탕으로 이미지를 압축(compression)하거나 재건축(reconstruction) 가능



PCA Neural Network

□ PCA는 Neural Network를 통해서도 구현이 가능함 (PCA Neural Network, Linear Autoassociator)



Learning

$$w_j \rightarrow e_j, \quad Var(y_j) \rightarrow \lambda$$

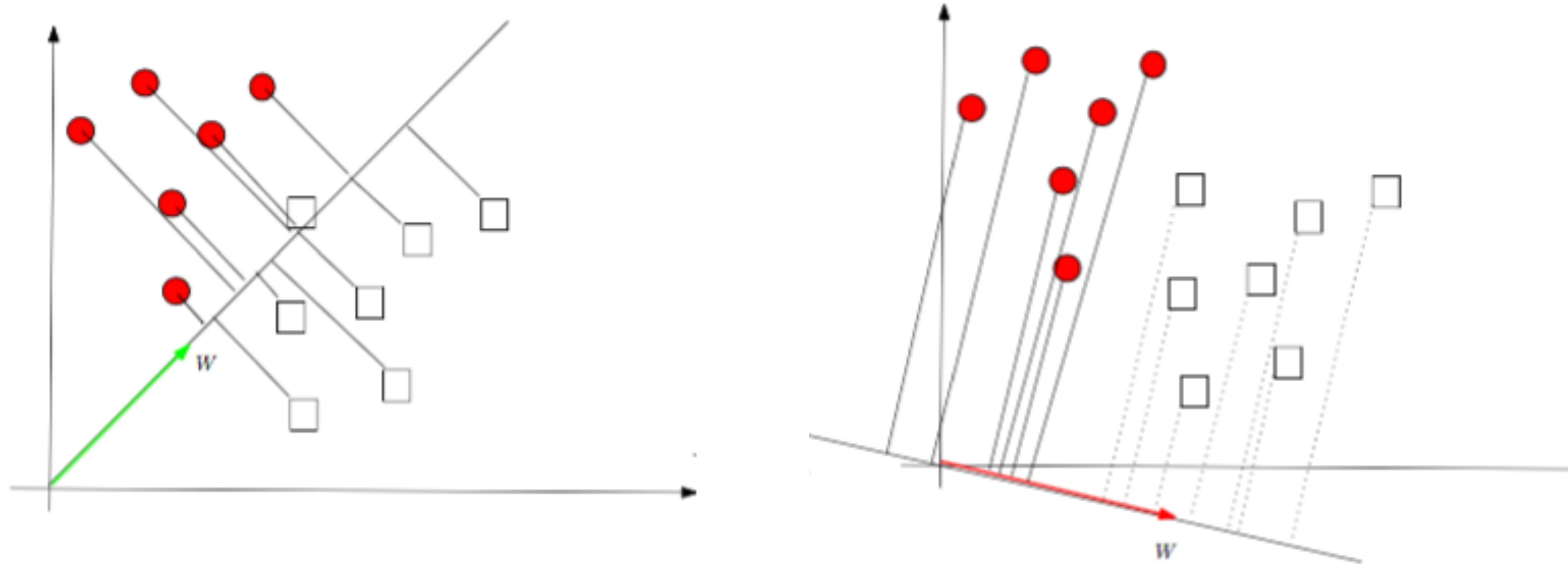
$$\hat{x} = \sum_{j=1}^M y_j e_j = \text{Least Square Estimate of } x$$

Linear Discriminant Analysis

Fisher Linear Discriminant

■ Fisher Linear Discriminant

- 클래스간 산포(Between-class)과 클래스내 산포(With in-class)을 고려하여 차원을 축소시키는 선형 판별 방법
- 분류(classification) 문제와 관련한 차원 축소 방법으로 주로 쓰임



Fisher Linear Discriminant

❑ Fisher Linear Discriminant

- 클래스간 분산(Between-class)과 클래스내 분산(With in-class)을 고려하여 차원을 축소시키는 선형 판별 방법

- 기존 클래스의 평균

$$y = \underbrace{\mathbf{w}^T \mathbf{x}}$$

Let $\mathbf{m}_i = \frac{1}{n_i} \sum_{x \in D_i} \mathbf{x}$.

1) 선형 판별 방법 적용 이후 각 클래스간 분산 (Between-class)

$$\tilde{m}_i = \frac{1}{n_i} \sum_{y \in Y_i} y = \frac{1}{n_i} \sum_{x \in X_i} \mathbf{w}^T \mathbf{x} = \mathbf{w} \cdot \mathbf{m}_i$$

$$|\tilde{m}_i - \tilde{m}_j| = |\underbrace{\mathbf{w}^T}_{\text{Magnitude depends on } \mathbf{w}(\text{scaling})} (\mathbf{m}_1 - \mathbf{m}_2)|$$

Minimize:

$$J(\mathbf{w}) = \frac{|\tilde{m}_1 - \tilde{m}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}$$

2) 선형 판별 방법 적용 이후 각 클래스내 분산 (Within-class)

$$\text{Scatter : } \tilde{s}_i^2 = \sum_{y \in Y_i} (y - \tilde{m}_i)^2$$

$\tilde{s}_1^2 + \tilde{s}_2^2$: Within -class scatter

$\frac{1}{n} \{ \tilde{s}_1^2 + \tilde{s}_2^2 \}$: An estimate of the variance of the pooled data.

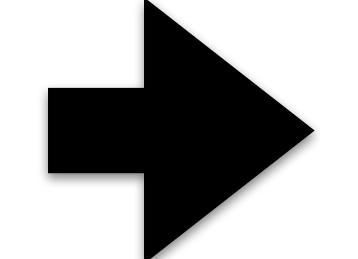
- Linear mapping 이 후, 클래스내 분산은 기존 클래스내 분산과 linear mapping, w 로 표현 가능

각 클래스별 Scatter Matrix (within-class)

Define scatter matrices S_i and S_w

$$S_i = \sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T$$

$$S_w = S_1 + S_2$$

$$y = \underbrace{\mathbf{w}^T \mathbf{x}}$$


Scatter Matrix of Projection Data (within-class)

$$\begin{aligned} \text{Then } \tilde{s}_i^2 &= \sum_{\mathbf{x} \in D_i} (\mathbf{w} \cdot \mathbf{x} - \mathbf{w} \cdot \mathbf{m}_i)^2 \\ &= \sum_{\mathbf{x} \in D_i} \mathbf{w}^T (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T \mathbf{w} \\ &= \mathbf{w}^T \left[\sum_{\mathbf{x} \in D_i} (\mathbf{x} - \mathbf{m}_i)(\mathbf{x} - \mathbf{m}_i)^T \right] \mathbf{w} \\ &= \mathbf{w}^T S_i \mathbf{w} \end{aligned}$$

$$\begin{aligned} \tilde{s}_1^2 + \tilde{s}_2^2 &= \mathbf{w}^T S_1 \mathbf{w} + \mathbf{w}^T S_2 \mathbf{w} = \mathbf{w}^T [S_1 + S_2] \mathbf{w} \\ &= \boxed{\mathbf{w}^T S_w \mathbf{w}} \end{aligned}$$

- Linear mapping 이 후, 클래스간 분산(between-class) 역시 유사한 방법으로 정리 가능

클래스간 분산 (between-class)

$$\begin{aligned}(\tilde{m}_1 - \tilde{m}_2)^2 &= (\mathbf{w}^T \mathbf{m}_1 - \mathbf{w}^T \mathbf{m}_2)^2 \\&= \mathbf{w}^T (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{w} \\&= \boxed{\mathbf{w}^T S_B \mathbf{w}}, \quad S_B = (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T\end{aligned}$$

LDA

□ 결국 판별함수는, 기울기 벡터에 대한 함수로 표현이 가능함

- 우리의 목적은 **클래스간 산포가 크고, 클래스내 산포는 적은** Projection 방향을 찾고 싶은 것임

$$J(\mathbf{w}) = \frac{|\tilde{\mathbf{m}}_1 - \tilde{\mathbf{m}}_2|^2}{\tilde{s}_1^2 + \tilde{s}_2^2} = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}$$

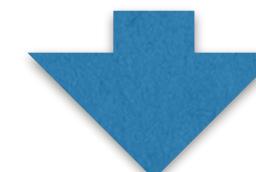
$$\begin{aligned} J(\mathbf{w}) &= \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}} = \frac{\alpha}{\beta} \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{2S_B \mathbf{w}}{\beta} - \frac{2\alpha}{\beta^2} S_w \mathbf{w} = \mathbf{0} \end{aligned}$$

$$S_B \mathbf{w} = \frac{\alpha}{\beta} S_w \mathbf{w} \rightarrow S_B \mathbf{w} = \lambda S_w \mathbf{w}$$

$$S_B \mathbf{w} = \lambda S_w \mathbf{w} \Rightarrow S_w^{-1} S_B \mathbf{w} = \lambda \mathbf{w}$$

- 결국 고유값, 고유벡터를 찾는 문제로 귀결……

$$\begin{aligned} S_B \mathbf{w} &= (\mathbf{m}_1 - \mathbf{m}_2)(\mathbf{m}_1 - \mathbf{m}_2)^T \mathbf{w} \\ &= (\mathbf{m}_1 - \mathbf{m}_2) [\mathbf{w}^T (\mathbf{m}_1 - \mathbf{m}_2)]^T \\ &= (\mathbf{m}_1 - \mathbf{m}_2) (\underbrace{\tilde{\mathbf{m}}_1 - \tilde{\mathbf{m}}_2}_{\text{Scalar values}})^T = K(\mathbf{m}_1 - \mathbf{m}_2) \end{aligned}$$



$$\left. \begin{aligned} S_w^{-1} S_B \mathbf{w} &= \lambda \mathbf{w} \\ S_B \mathbf{w} &= K(\mathbf{m}_1 - \mathbf{m}_2) \end{aligned} \right\} \begin{aligned} S_w^{-1} K(\mathbf{m}_1 - \mathbf{m}_2) &= \lambda \mathbf{w} \\ \mathbf{w} &= \underbrace{\frac{K}{\lambda}}_{\text{scaling factor}} S_w^{-1} (\mathbf{m}_1 - \mathbf{m}_2) \end{aligned}$$

Fisher linear discriminant : $y = \mathbf{w} \cdot \mathbf{x}$

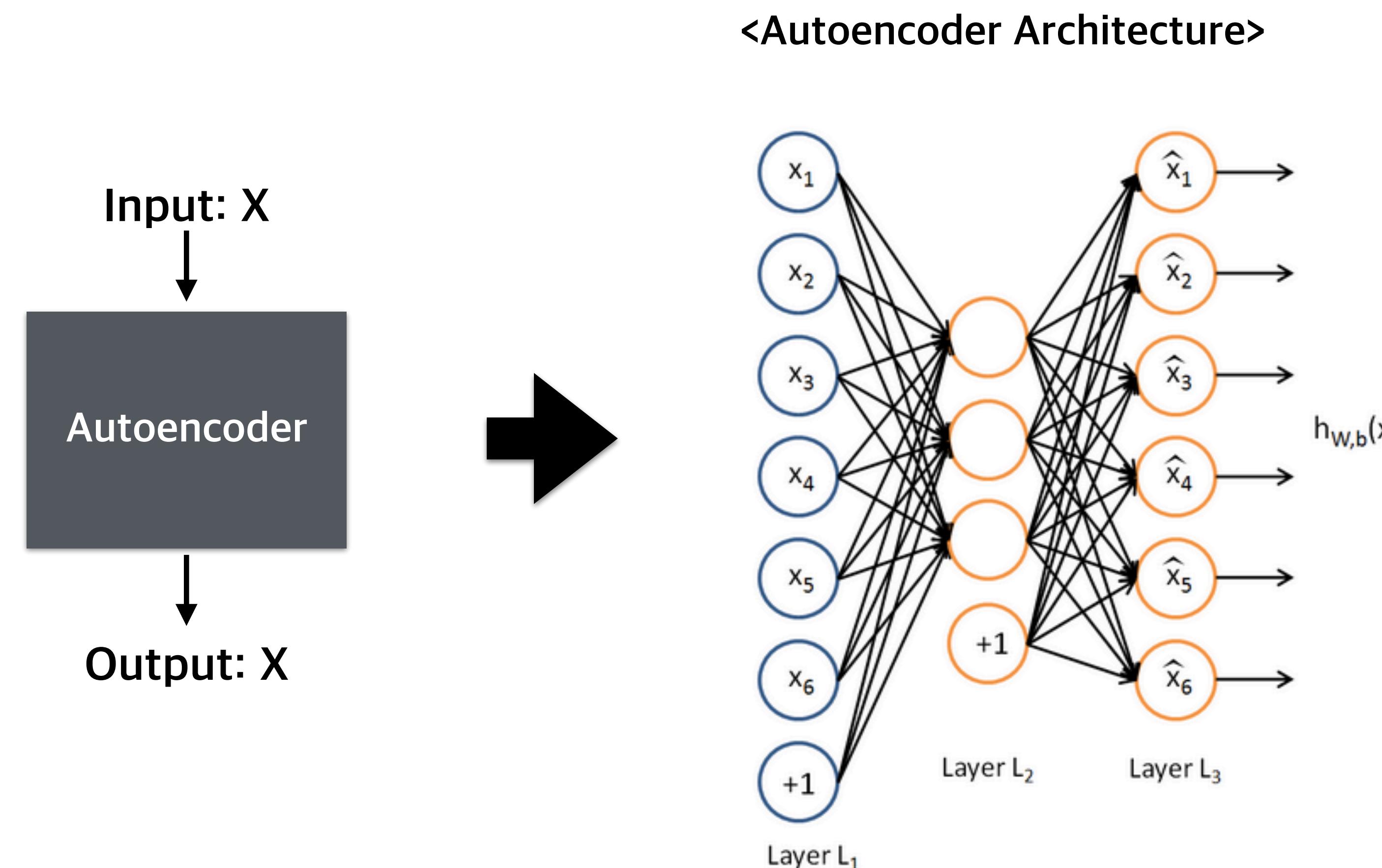
$$\mathbf{w} = S_w^{-1} (\mathbf{m}_1 - \mathbf{m}_2)$$

- ❑ PCA는 데이터를 잘 설명하는 방향(variance)으로 Projection을 시키는 것이 목적
- ❑ LDA(Fisher 등..)는 데이터를 잘 분리하는 방향(discrimination)으로 Projection을 시키는 것이 목적
- ❑ 두 방법 모두 결국 Linear Model of Dimensionality Reduction !
- ❑ (Nonlinear dimensionality reduction?)
 - Kernel PCA
 - Add polynomial terms
 - Autoencoder

Autoencoder

Autoencoder

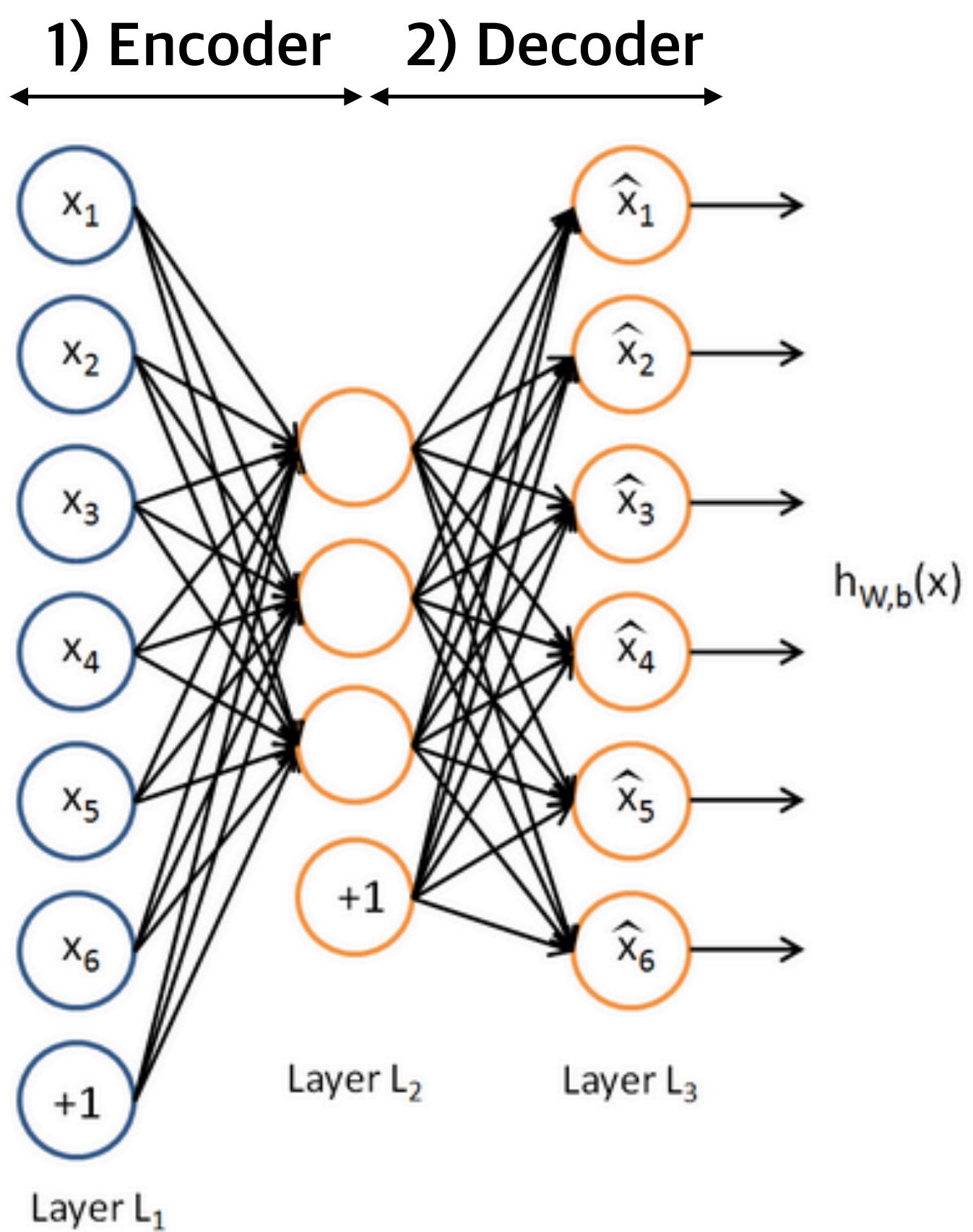
- ❑ Neural Network를 이용한 Unsupervised Learning 방식으로 Encoder-Decoder 구조를 가진 Neural Network
- ❑ Input X 에 대하여 Output을 Input과 동일하게 설정한 후, 이를 추정하는 Neural Network 모델을 학습
- ❑ Input 데이터 X 를 잘 설명하는 **hidden representation (latent variable, 잠재 변수)**를 얻는 것을 목적으로함



Autoencoder Terminology

- Autoencoder 모델은 1) encoder, 2) decoder로 구성되어 있음
- 일반적으로 Input의 차원보다 낮은 개수의 hidden neuron을 사용하여, 차원 축소 및 데이터 압축에 활용

<Autoencoder Architecture>



1) Encoder

- Input X를 Latent variable에 대한 표현으로 변경하는 부분
- 일반적으로 고차원의 X를 저차원의 Z로 embedding하는 것을 의미
- 차원 축소 및 데이터 압축에 사용할 수 있음

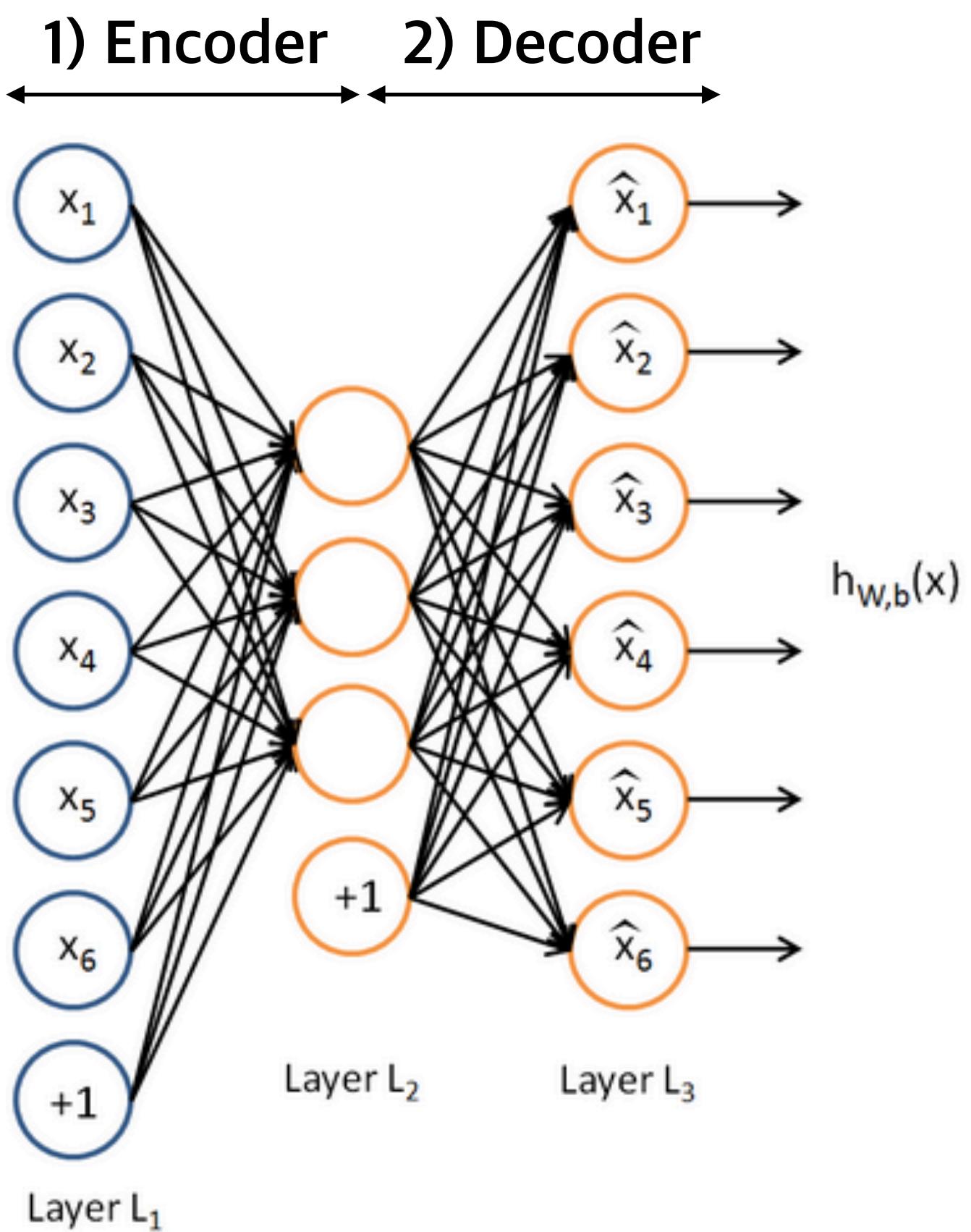
2) Decoder

- Latent variable의 표현을 기존 데이터 X의 방식으로 표현하는 부분
- 일반적으로 저차원의 Z를 고차원의 X로 복구(reconstruction)하는 것을 의미
- Latent variable의 표현이 복구하고자 하는 데이터에 대한 충분한 정보를 포함하고 있다는 것을 가정

Autoencoder Terminology

- Autoencoder 모델은 1) encoder, 2) decoder로 구성되어 있음
- 일반적으로 Input의 차원보다 낮은 개수의 hidden neuron을 사용하여, 차원 축소 및 데이터 압축에 활용

<Autoencoder Architecture>



1) Encoder

- Input X를 Latent variable에 대한 표현으로 변경하는 부분
- 일반적으로 고차원의 X를 저차원의 Z로 embedding하는 것을 의미
- 차원 축소 및 데이터 압축에 사용할 수 있음

2) Decoder

- Latent variable의 표현을 기존 데이터 X의 방식으로 표현하는 부분
- 일반적으로 저차원의 Z를 고차원의 X로 복구(reconstruction)하는 것을 의미
- Latent variable의 표현이 복구하고자 하는 데이터에 대한 충분한 정보를 포함하고 있다는 것을 가정

$$a^{(2)} = f(W^T X) \longrightarrow \text{Encoding}$$

$$\hat{X} = g(W'^T a^{(2)}) \longrightarrow \text{Decoding}$$

$$W' = W^T \quad (\text{Optional})$$

Cost Function of Autoencoder

- Autoencoder는 일반적으로 **Average Reconstruction Error**를 Loss로 설정한 후, 최소화하는 방향으로 학습
- 결과가 확률 정보와 관련된 경우 reconstruction cross-entropy를 사용하기도 함

$$\begin{aligned} W, W' &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(x^{(i)}, \hat{x}^{(i)}) \\ &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\hat{x}^{(i)} - x^{(i)})^2 \end{aligned}$$

Cost Function of Autoencoder

- Autoencoder는 일반적으로 **Average Reconstruction Error**를 Loss로 설정한 후, 최소화하는 방향으로 학습
- 결과가 확률 정보와 관련된 경우 reconstruction cross-entropy를 사용하기도 함

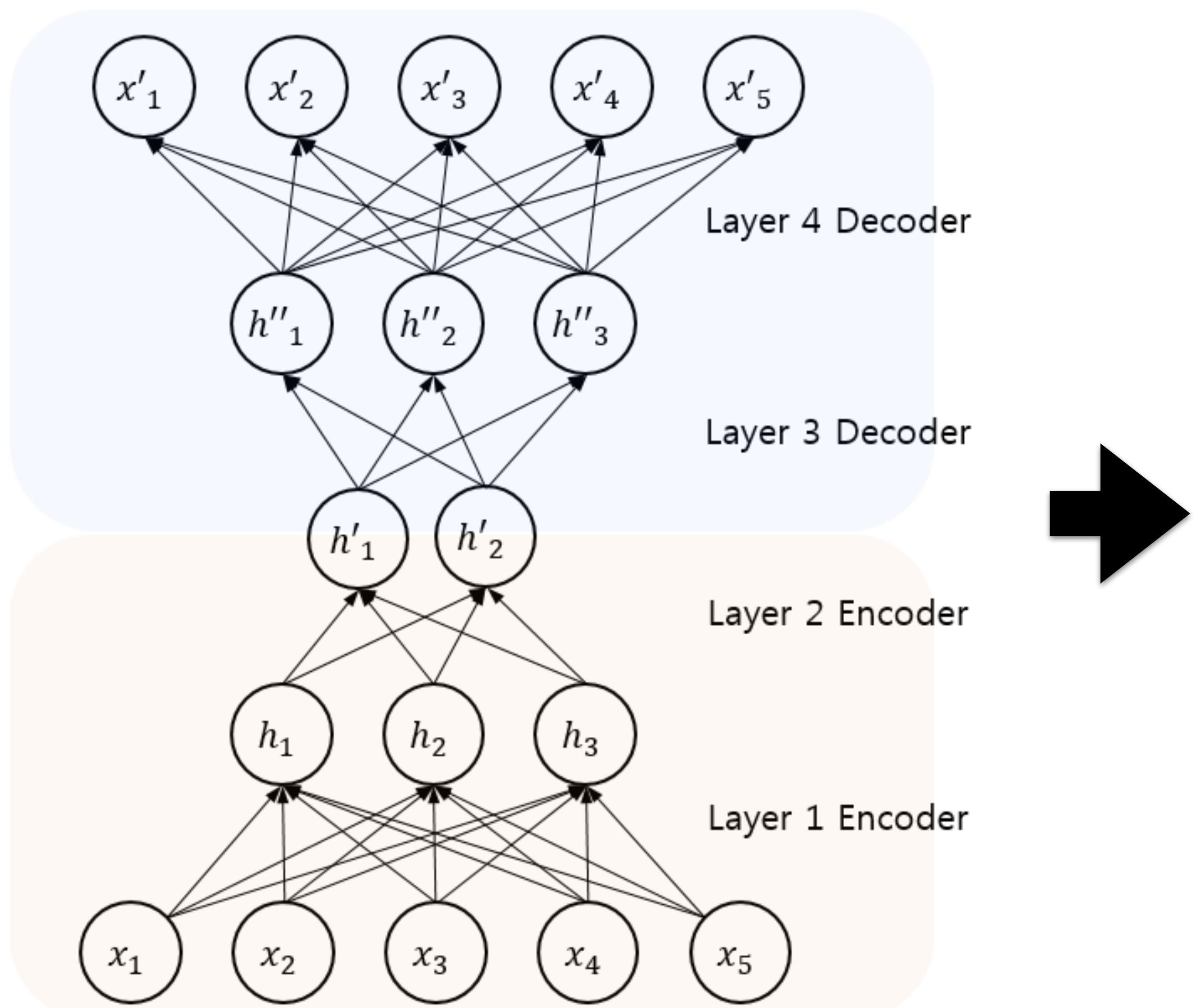
$$\begin{aligned} W, W' &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(x^{(i)}, \hat{x}^{(i)}) \\ &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\hat{x}^{(i)} - x^{(i)})^2 \end{aligned}$$

Q) How can we extract
more complex & nonlinear latent representation?

A) Use deeper layer

Stacked Autoencoder

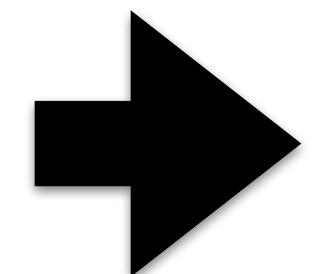
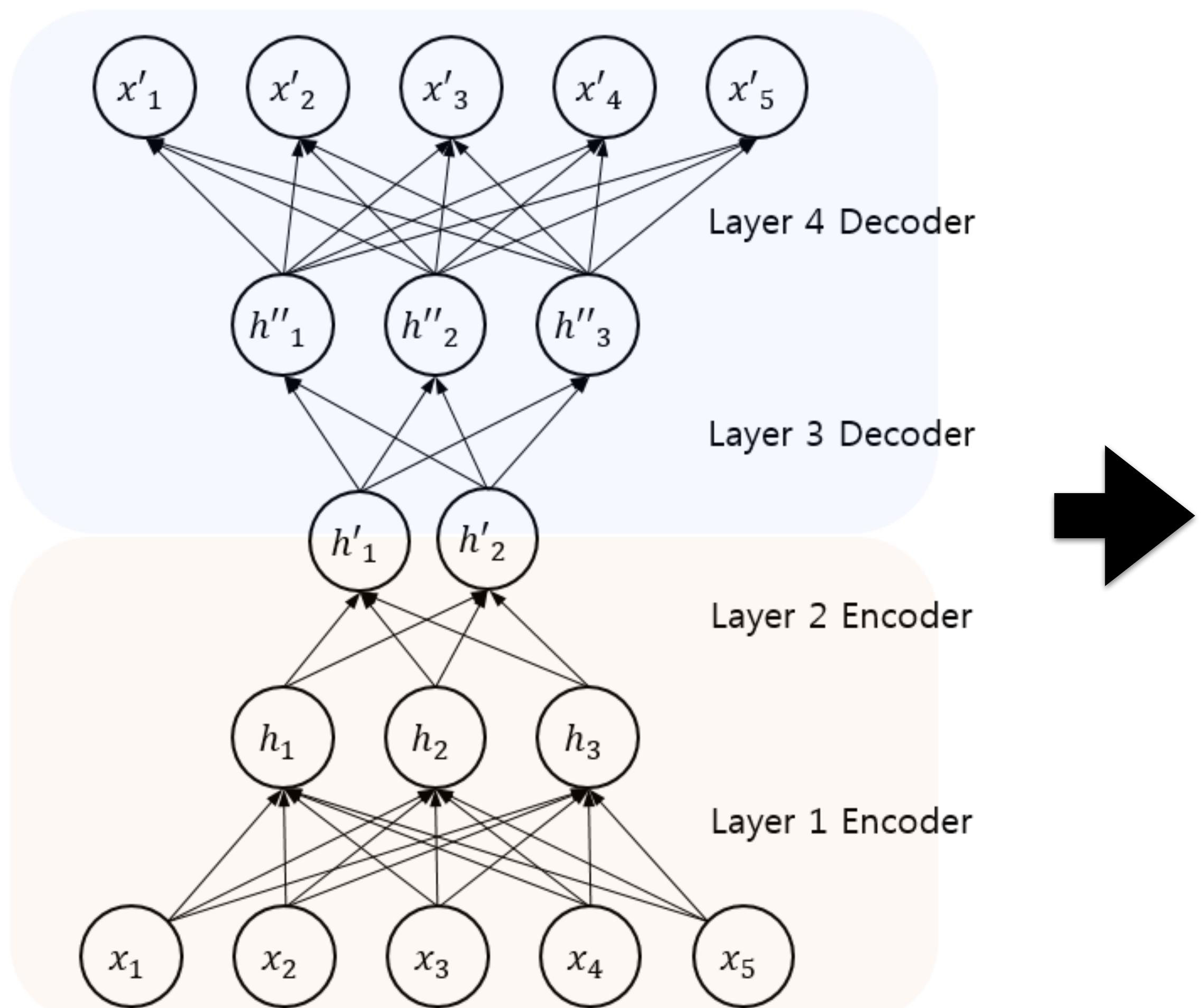
- Encoder, decoder가 여러 층으로 구성되어 있는 Autoencoder
- 기존 Autoencoder보다 더 복잡한 정보를 포함한 latent representation을 얻을 수 있음



Dimensionality Reduction가 아니더라도
(Stacked) Autoencoder는
NN의 중요한 모델 중 하나! (Why?)

Stacked Autoencoder

- Encoder, decoder가 여러 층으로 구성되어 있는 Autoencoder
- 기존 Autoencoder보다 더 복잡한 정보를 포함한 latent representation을 얻을 수 있음



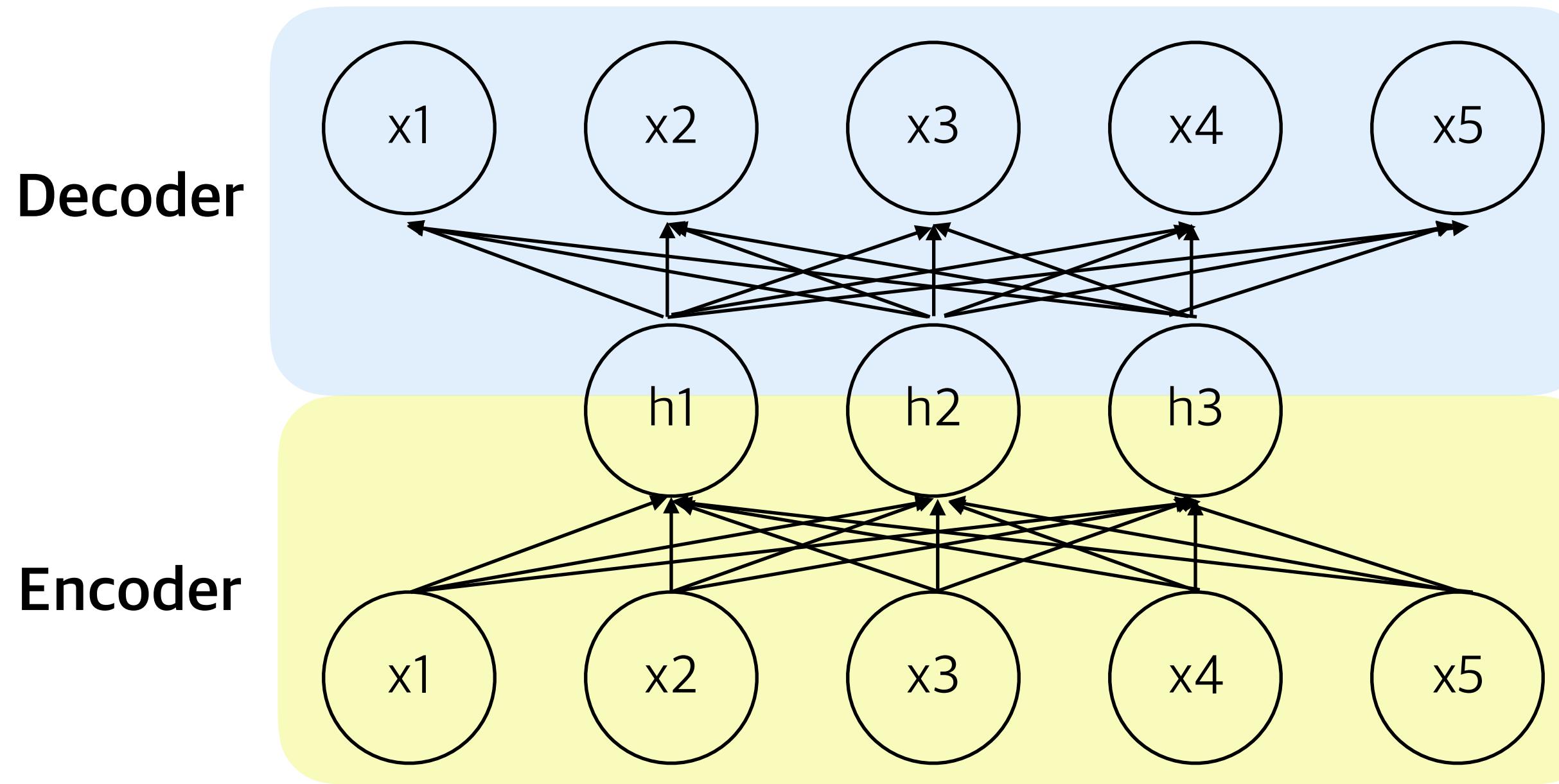
Dimensionality Reduction가 아니더라도
(Stacked) Autoencoder는
NN의 중요한 모델 중 하나! (Why?)

(Stacked) Autoencoder is
good initialization for fine tuning!

Initialization & Fine Tuning

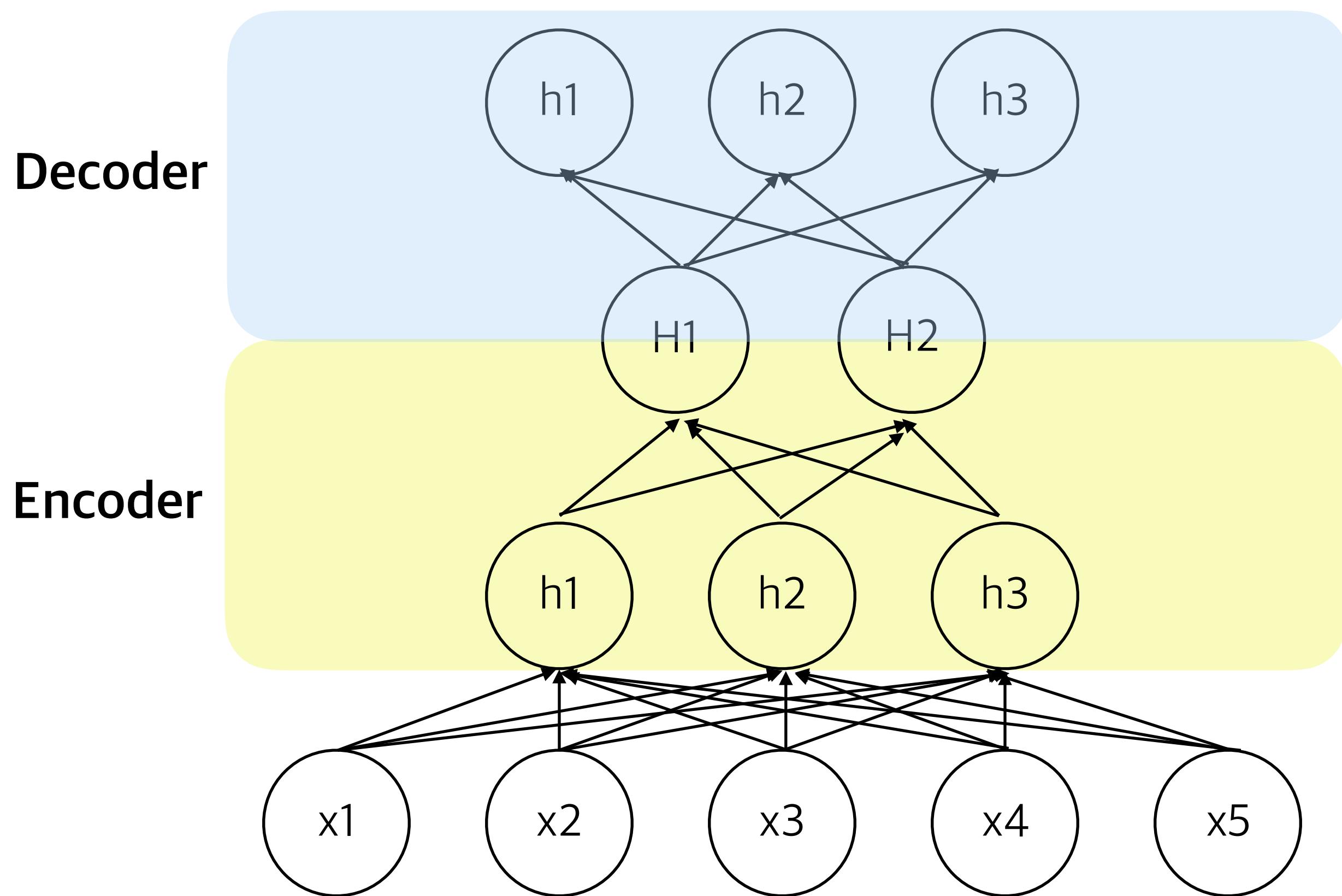
- 각 Layer를 **직전 layer 결과값**을 입력으로 하는 Autoencoder로 생각하여 학습을 진행함 (**Pre-training**)
- 최종적으로 학습된 Encoders를 바탕으로 (특정) 모델을 설정하고, **Fine Tuning** 후 사용

<Pre-training for Initialization>



- 각 Layer를 **직전 layer 결과값**을 입력으로 하는 Autoencoder로 생각하여 학습을 진행함 (**Pre-training**)
- 최종적으로 학습된 Encoders를 바탕으로 (특정) 모델을 설정하고, **Fine Tuning** 후 사용

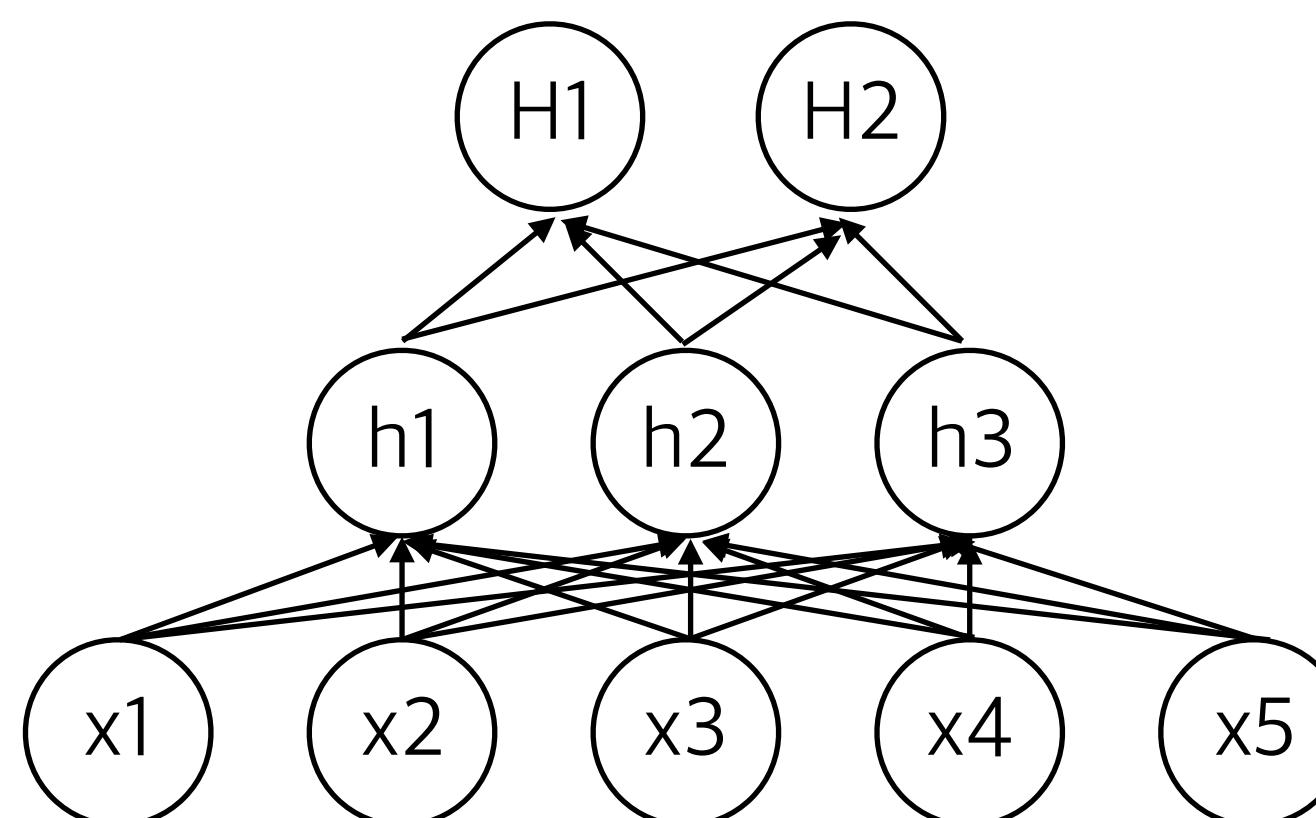
<Pre-training for Initialization>



Initialization & Fine Tuning

- 각 Layer를 **직전 layer 결과값**을 입력으로 하는 Autoencoder로 생각하여 학습을 진행함 (**Pre-training**)
- 최종적으로 학습된 Encoders를 바탕으로 (특정) 모델을 설정하고, **Fine Tuning** 후 사용

<Pre-training for Initialization>

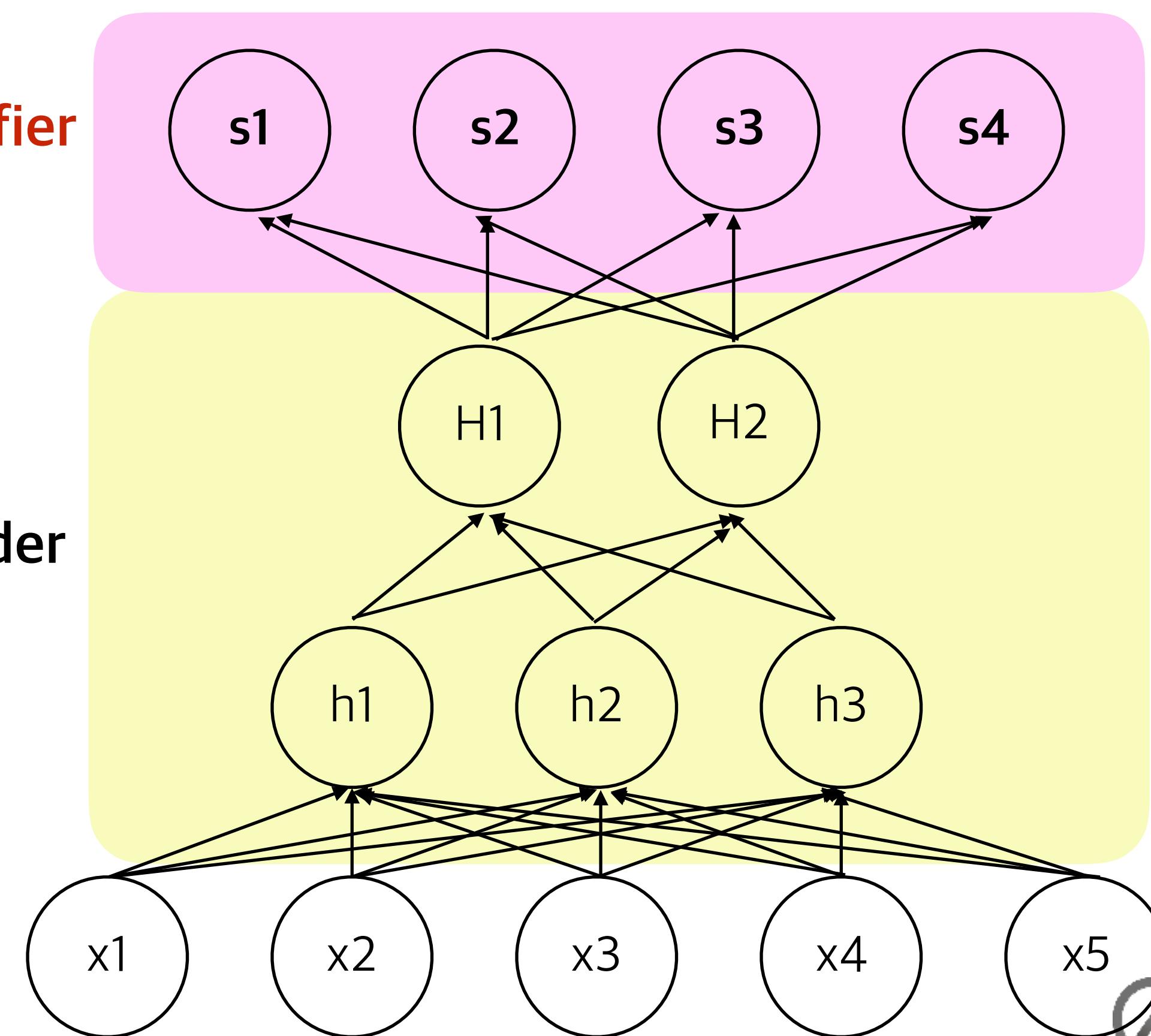


Encoder
Initialization

Classifier

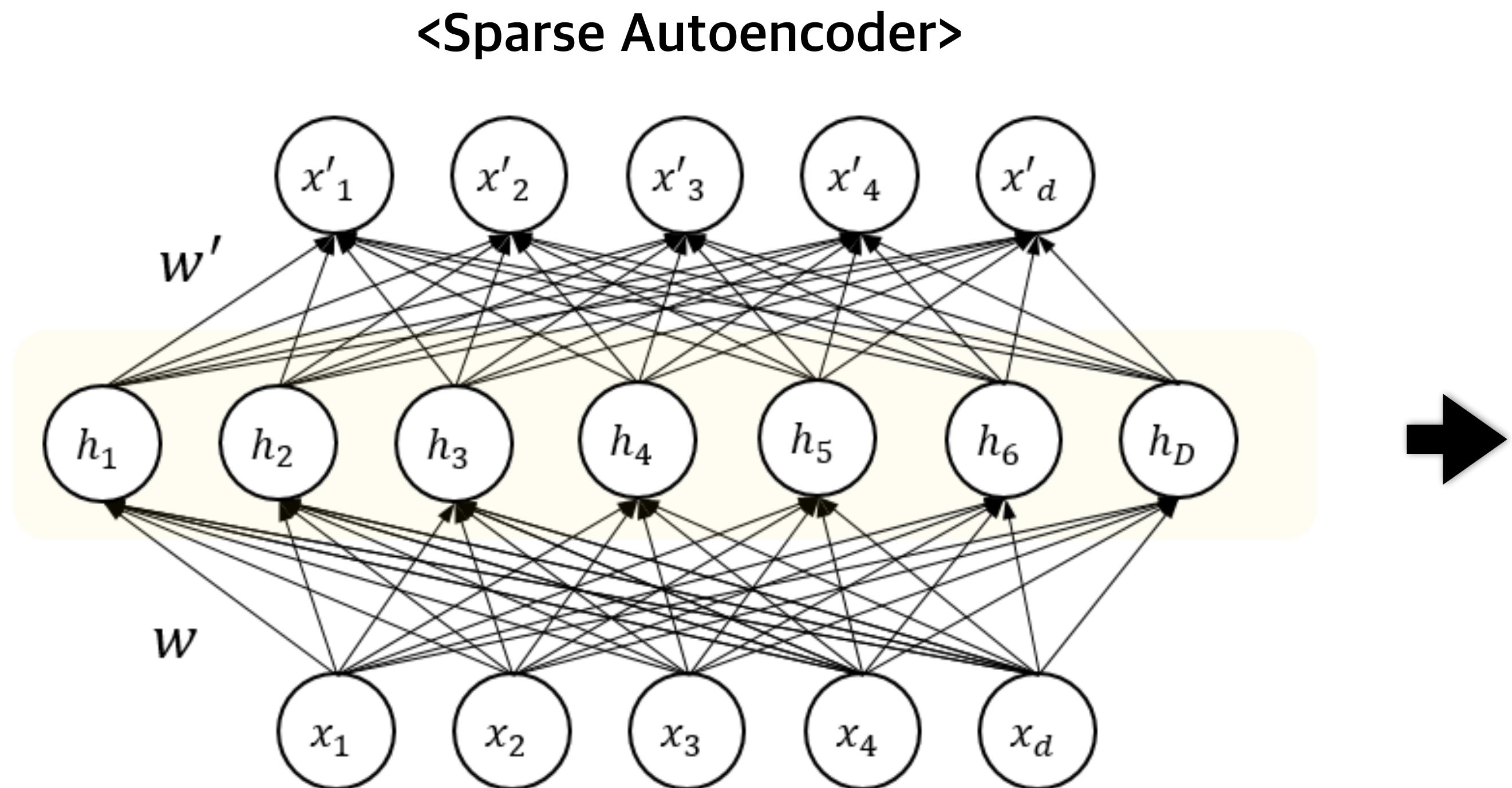
Encoder

<Fine Tuning for Classification>



Sparse Autoencoder

- Hidden Layer 내 뉴런의 수가 Input 데이터의 차원의 수보다 많은 구조의 Autoencoder
- Hidden Neuron의 Activation 정도를 제한하여 **Sparsity**를 유지하고, 이를 통해 Input의 특수한 구조를 파악함



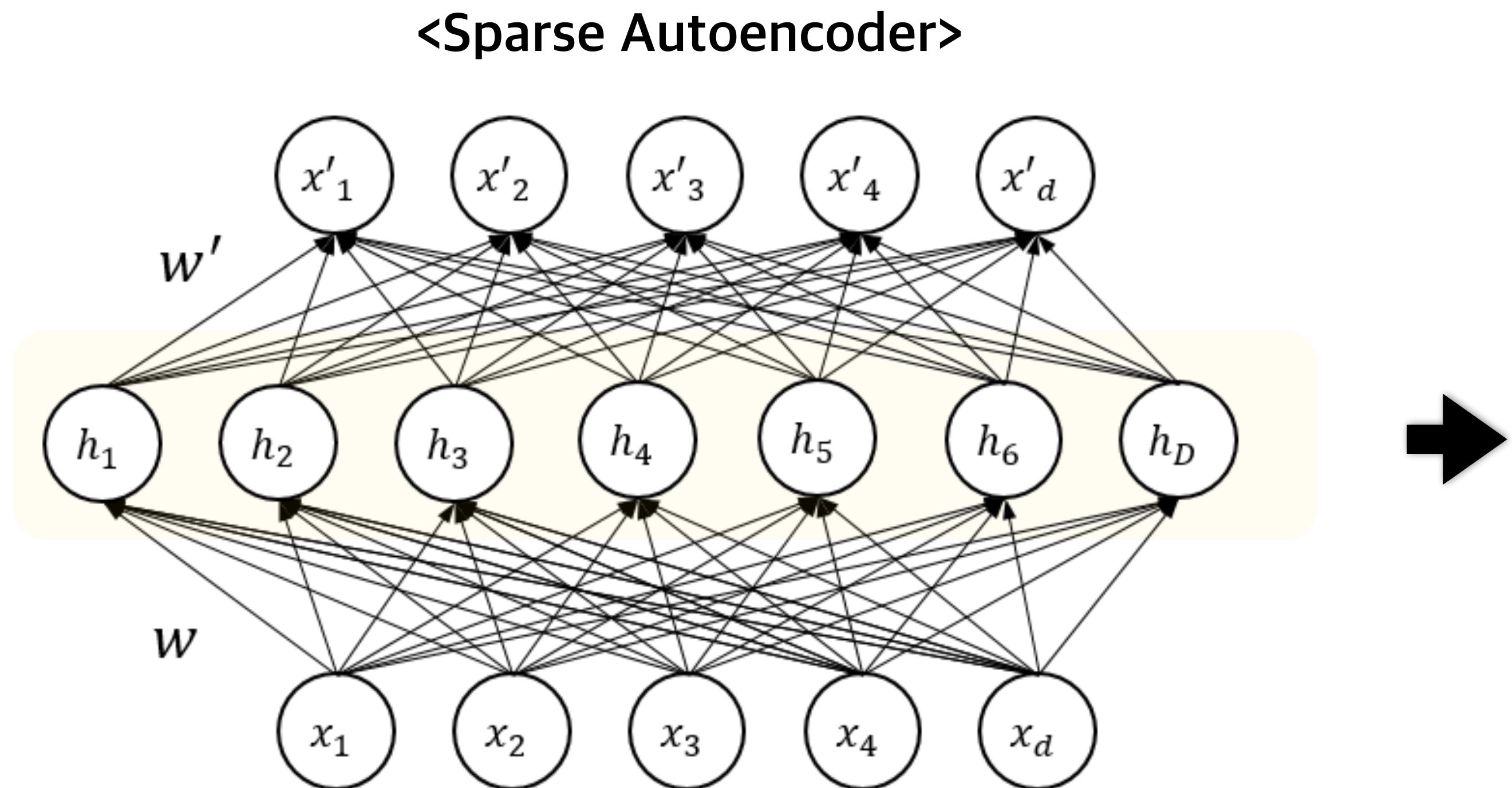
<Activation Sparsity>

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m h_j(x^{(i)})$$

m개의 input 데이터에 대하여
j 번째 hidden activation 값 평균

Sparse Autoencoder

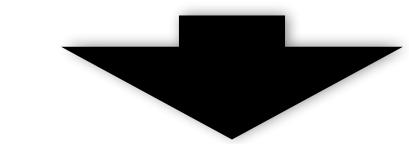
- Hidden Layer 내 뉴런의 수가 Input 데이터의 차원의 수보다 많은 구조의 Autoencoder
- Hidden Neuron의 Activation 정도를 제한하여 **Sparsity**를 유지하고, 이를 통해 Input의 특수한 구조를 파악함



<Activation Sparsity>

$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^m h_j(x^{(i)})$$

m개의 input 데이터에 대하여
j 번째 hidden activation 값 평균



<Sparsity Constraint>

$$\hat{\rho}_j = \rho$$

- 일반적으로 0.2 정도로 설정
- 모든 데이터에 대하여 각 뉴런의 Activation은 0.2를 넘지 않음
- 각 뉴런이 특별한 특징에 대해서만 Activation 하도록 제약

Loss Function of Sparse Autoencoder

- ❑ 기존 autoencoder의 loss에 sparsity 제약 조건을 더한 함수를 loss function으로 사용
- ❑ 학습 중인 모델의 Activation 평균과 설정한 sparsity 제약 조건 사이의 KL-divergence로 수식화하여 표현

Cost Function of Sparse AE

$$W, W' = \underset{W, W'}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{n=1}^n L(x^{(i)}, \hat{x}^{(i)}) + \lambda \sum_{j=1}^{h_d} \overline{KL(\rho \parallel \hat{\rho}_j)} \right\}$$

sparsity constraint

Loss Function of Sparse Autoencoder

- 기존 autoencoder의 loss에 sparsity 제약 조건을 더한 함수를 loss function으로 사용
- 학습 중인 모델의 Activation 평균과 설정한 sparsity 제약 조건 사이의 KL-divergence로 수식화하여 표현

Cost Function of Sparse AE

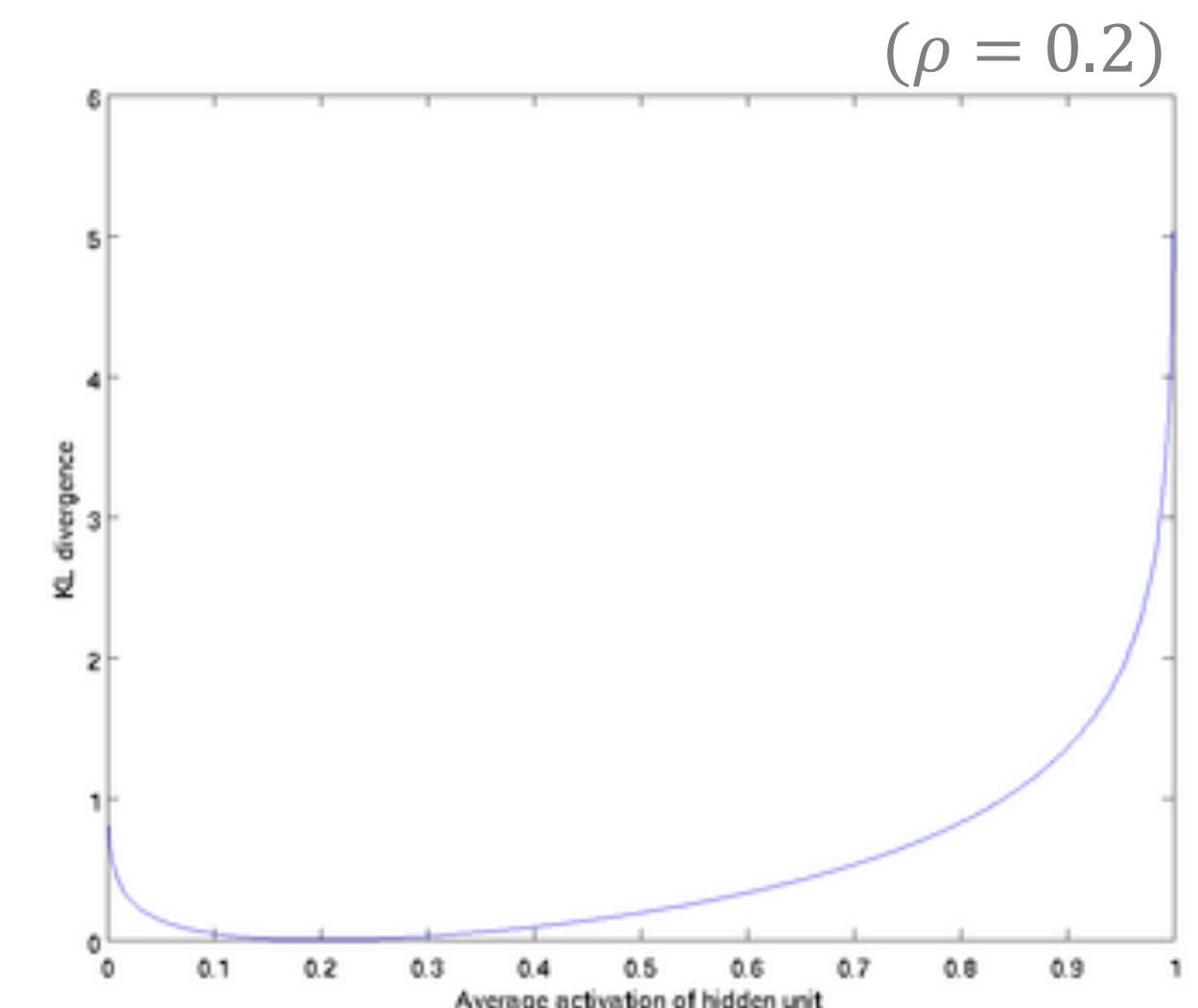
$$W, W' = \underset{W, W'}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{n=1}^n L(x^{(i)}, \hat{x}^{(i)}) + \lambda \sum_{j=1}^{h_d} KL(\rho \parallel \hat{\rho}_j) \right\}$$

sparsity constraint

<KL-divergence with sparsity constraint>

$$\sum_{j=1}^{h_d} KL(\rho \parallel \hat{\rho}_j) = \sum_{j=1}^{h_d} \left[\rho \log \left(\frac{\rho}{\hat{\rho}_j} \right) + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j} \right]$$

<KL-divergence Example>



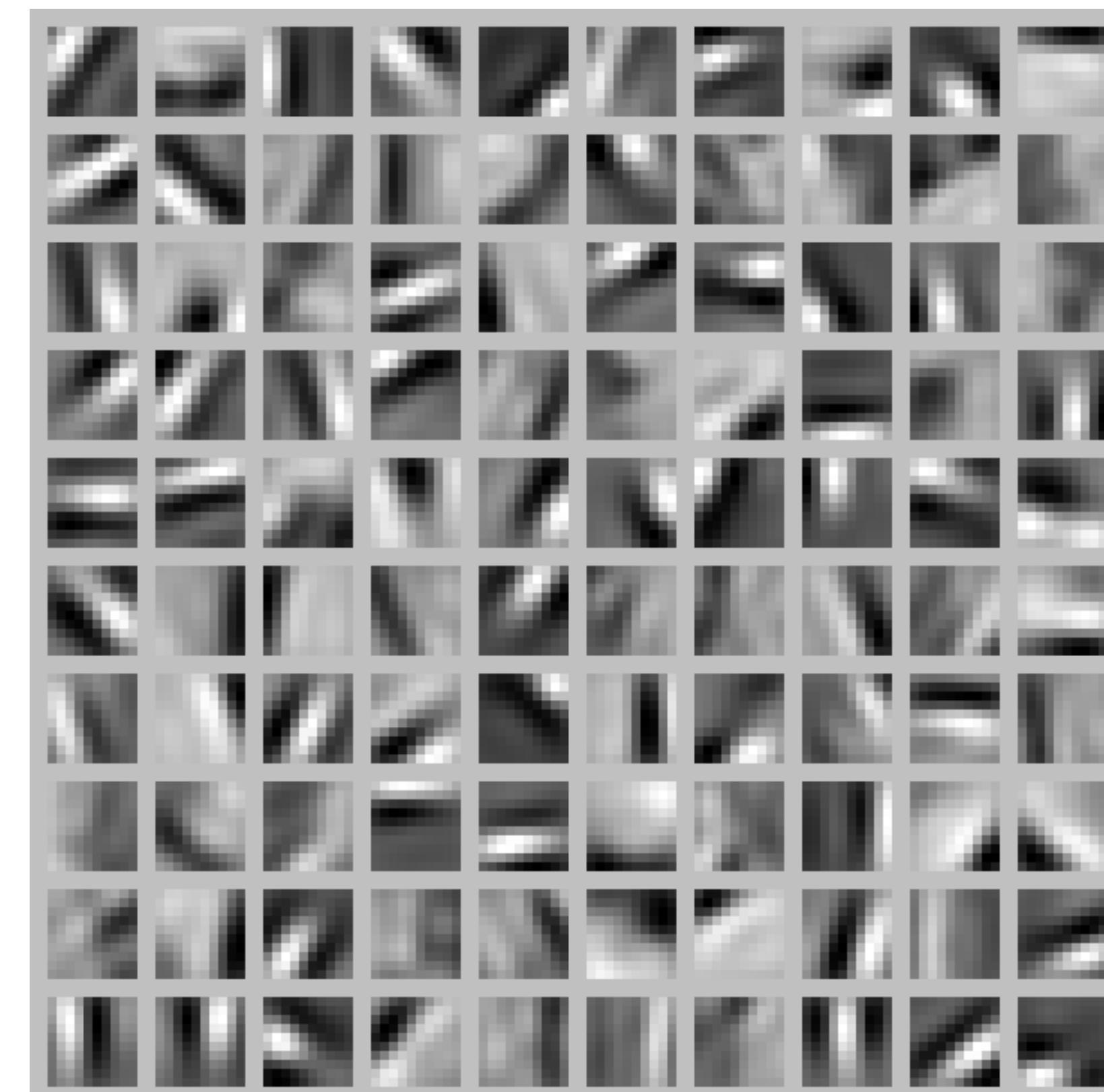
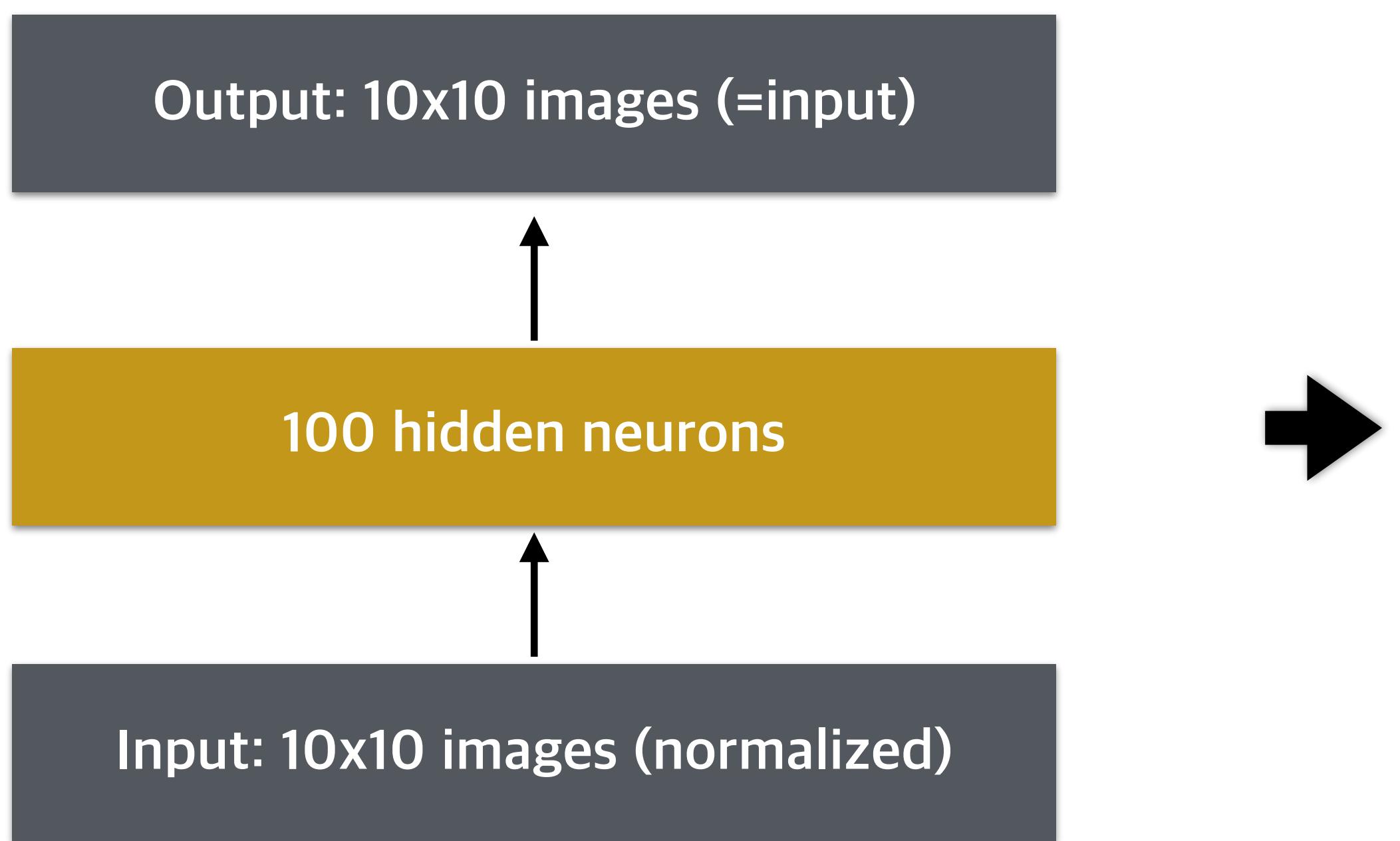
- 설정한 sparsity 값과 차이가 커질 경우 증가
- Sparsity 값을 평균으로 갖는

Bernoulli random distribution 사이의 차이를 의미

Sparse Autoencoder Result Example

- 10x10 이미지를 입력값으로 하고 100개의 hidden neuron을 갖는 sparse autoencoder 학습에 대한 결과 예시
- 각 100개의 hidden neuron의 activation을 최대화하는 Input Image를 계산하여 확인해본 결과는 아래와 같음
- 서로 다른 뉴런은 이미지의 서로 다른 특징에 반응하도록 학습이 됨을 알 수 있음

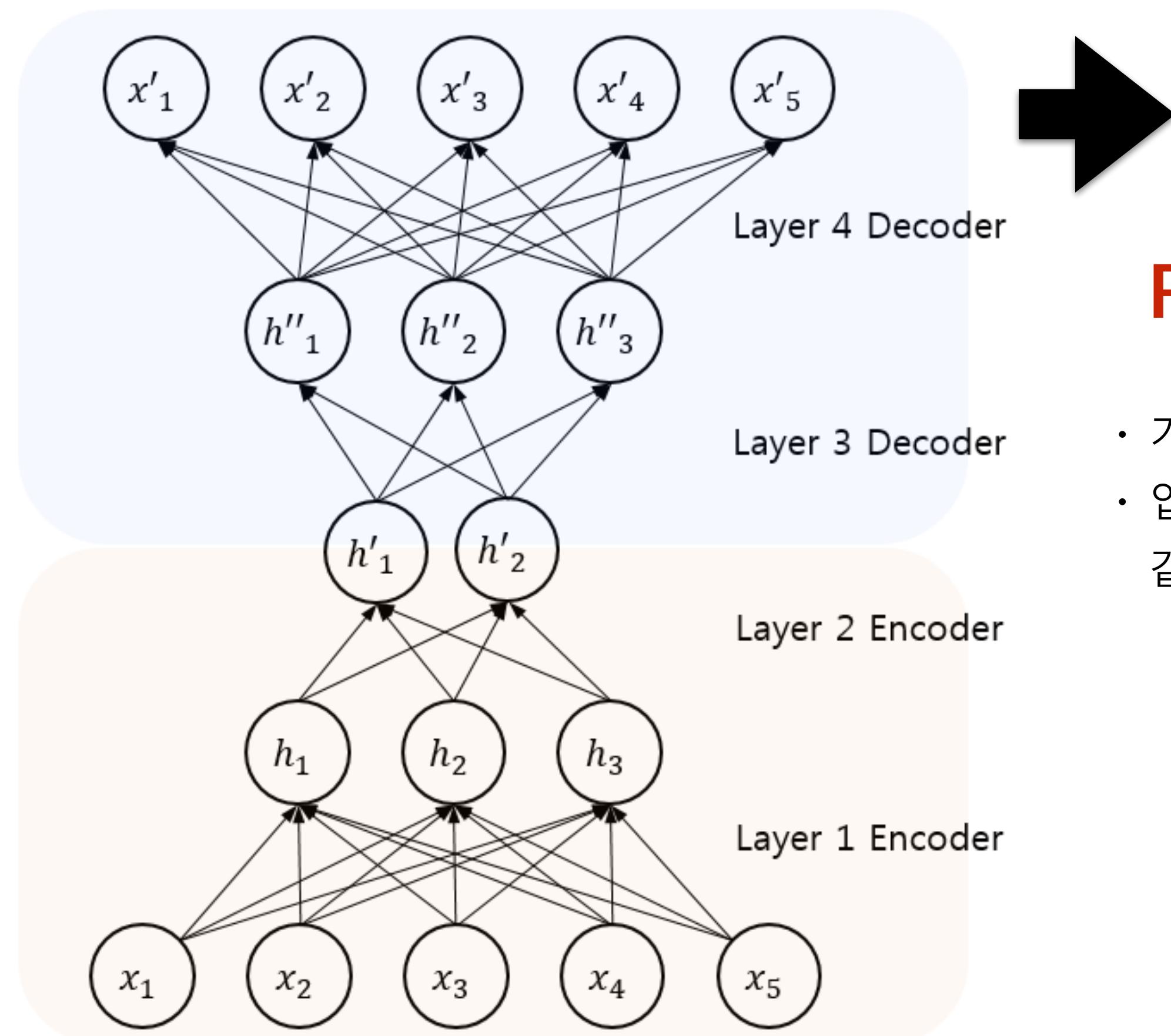
<Input Images Maximizing Each Hidden Neurons>



One more important Autoencoder

New Criteria for Autoencoder

- Autoencoder는 입력 데이터를 잘 설명하는 숨은 표현(latent representation)을 추출하는 것이 근본적인 목적
- 입력 데이터가 어느 정도 손상되거나 변형되더라도 핵심적인 정보를 잘 추출하는 것 또한 성능의 한 기준으로 볼 수 있음



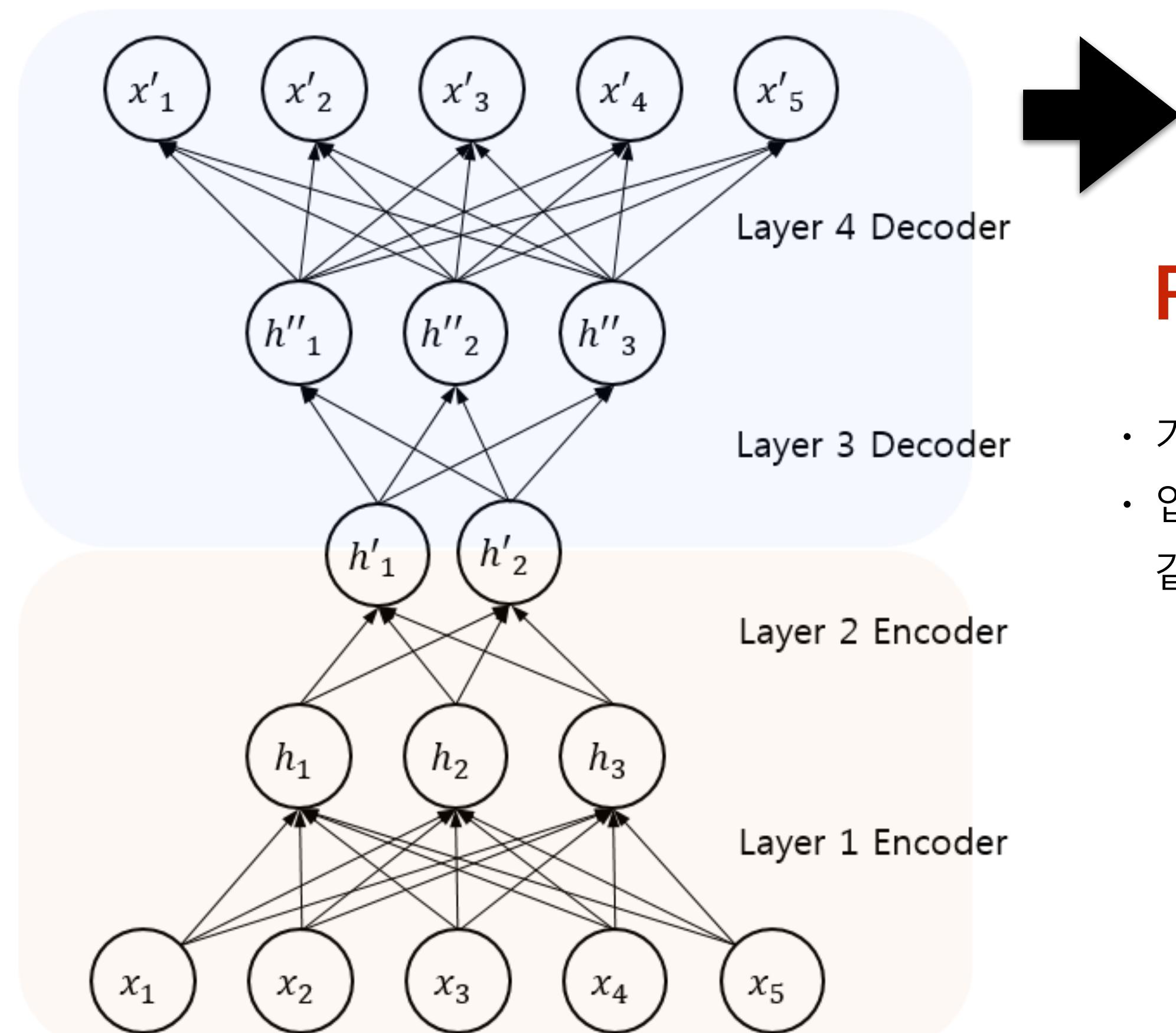
New Criteria:

Robustness to partial destruction of the input

- 기존 Autoencoder는 입력된 데이터의 전체를 이용해서 유의미한 latent representation을 도출
- 입력 데이터의 특징을 잘 추출하는 AE는 **입력값이 어느 정도 손상되거나 Noise가 섞이더라도 같은 정보를 얻을 수 있어야함 (robustness to partial destruction)**

New Criteria for Autoencoder

- Autoencoder는 입력 데이터를 잘 설명하는 숨은 표현(latent representation)을 추출하는 것이 근본적인 목적
- 입력 데이터가 어느 정도 손상되거나 변형되더라도 핵심적인 정보를 잘 추출하는 것 또한 성능의 한 기준으로 볼 수 있음



New Criteria:

Robustness to partial destruction of the input

- 기존 Autoencoder는 입력된 데이터의 전체를 이용해서 유의미한 latent representation을 도출
- 입력 데이터의 특징을 잘 추출하는 AE는 **입력값이 어느 정도 손상되거나 Noise가 섞이더라도 같은 정보를 얻을 수 있어야함 (robustness to partial destruction)**

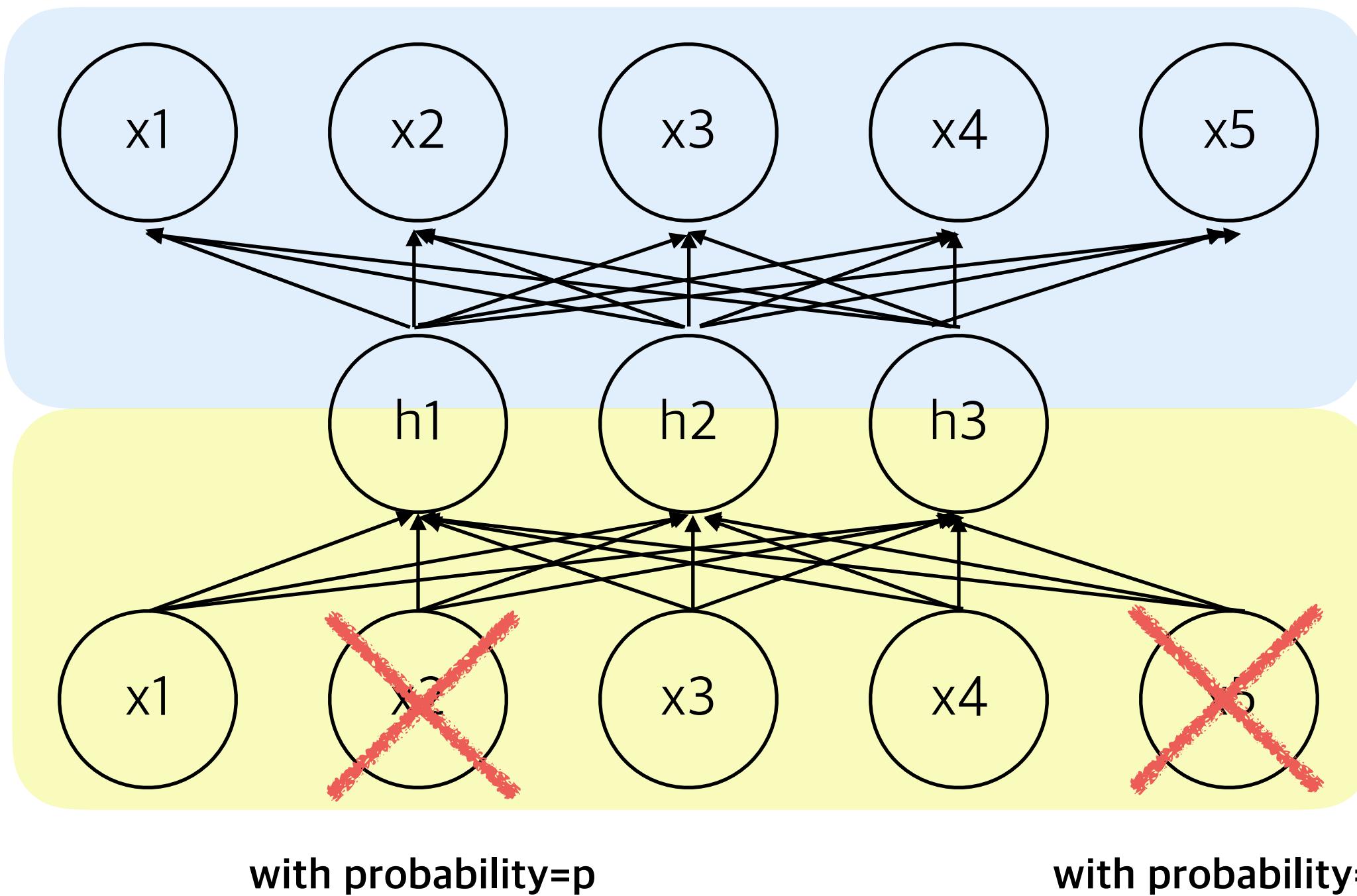
Denoising Autoencoder

Denoising Autoencoder

- 입력 데이터 X 가 포함하는 각 차원의 값을 임의의 확률 p 로 생략한 후, Autoencoder의 입력값으로 사용
- Denoising Autoencoder의 Loss는 기존 입력 데이터와 노이즈 입력 데이터로 복원된 결과 사이의 복원 에러 (reconstruction error)
- Stacked Denoising AE는 기존과 동일하게 layer 단위로 학습을 시키며, 매 layer 학습마다 input corruption을 적용함

<Denoising Autoencoder Architecture>

Decoder

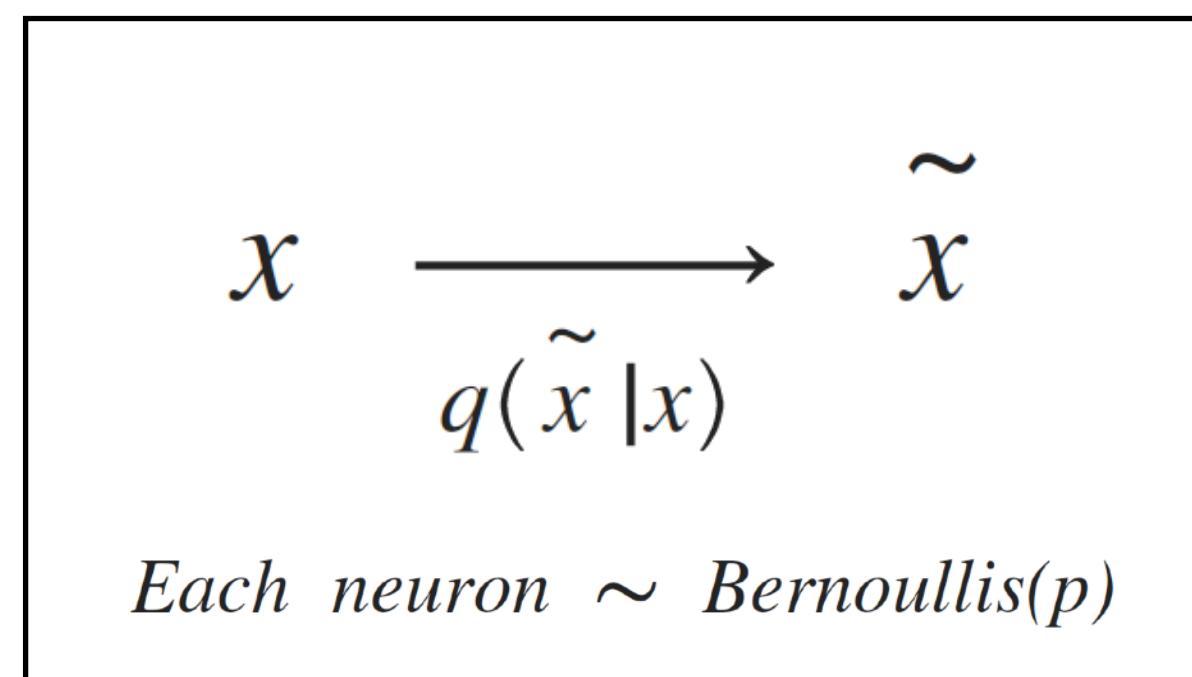


Encoder

with probability=p

with probability=p

<Corruption Distribution of x >



<Loss Function of Denoising Autoencoder>

$$W, W' = \underset{W, W'}{\operatorname{argmin}} E_{q(X, \tilde{X})} [L(X, g(W'^T f(W^T \tilde{X})))]$$

Other Interpretation of Denoising Autoencoder

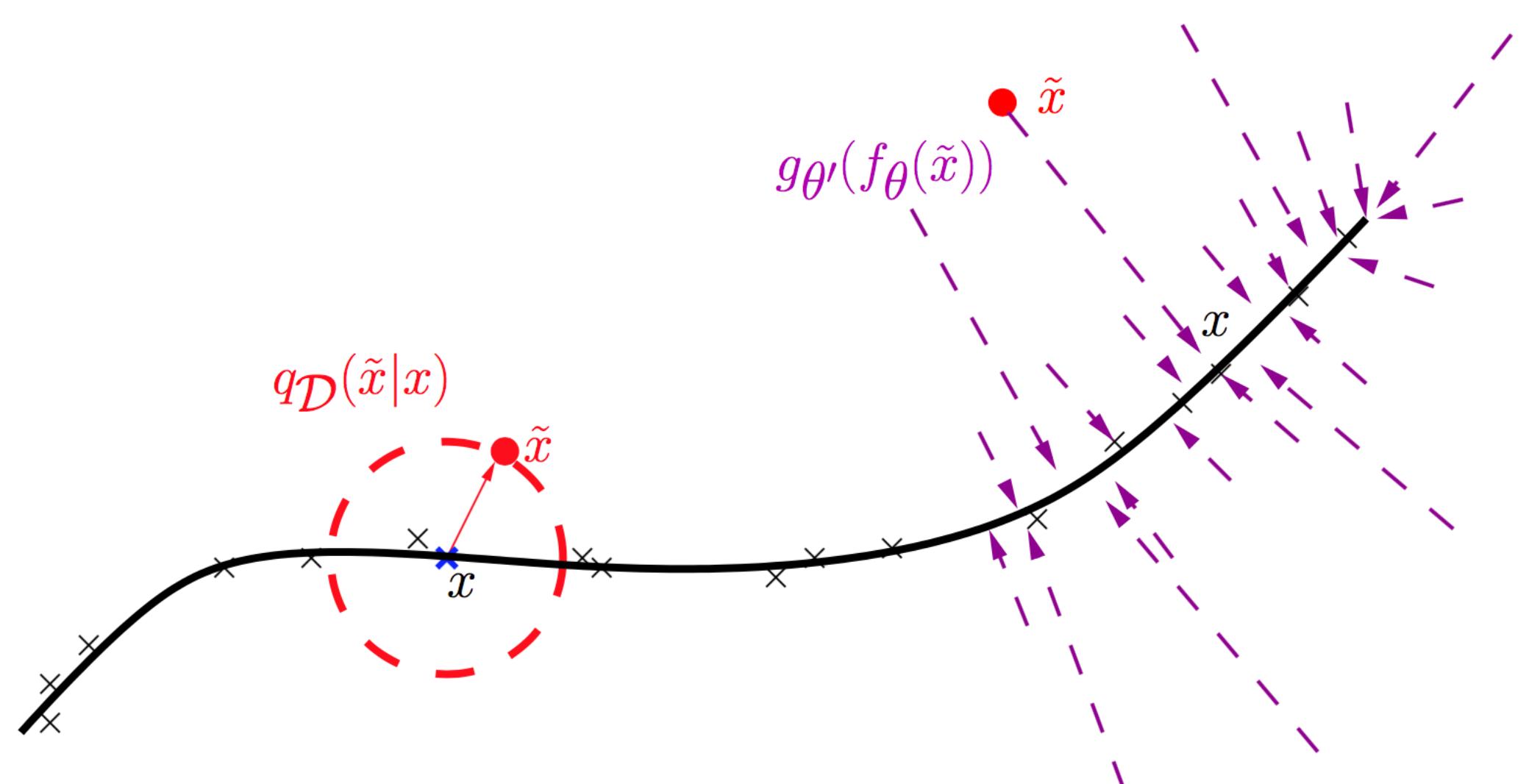
□ Denoising Autoencoder의 학습과 그 결과를 1) Manifold Learning, 2) Generative Model의 관점에서 해석 가능

1) Manifold Learning: AE는 training 데이터의 manifold를 학습하고,

Denoising AE는 noise 데이터가 manifold에 도달할 수 있도록 하는 방향을 학습하는 것으로 해석

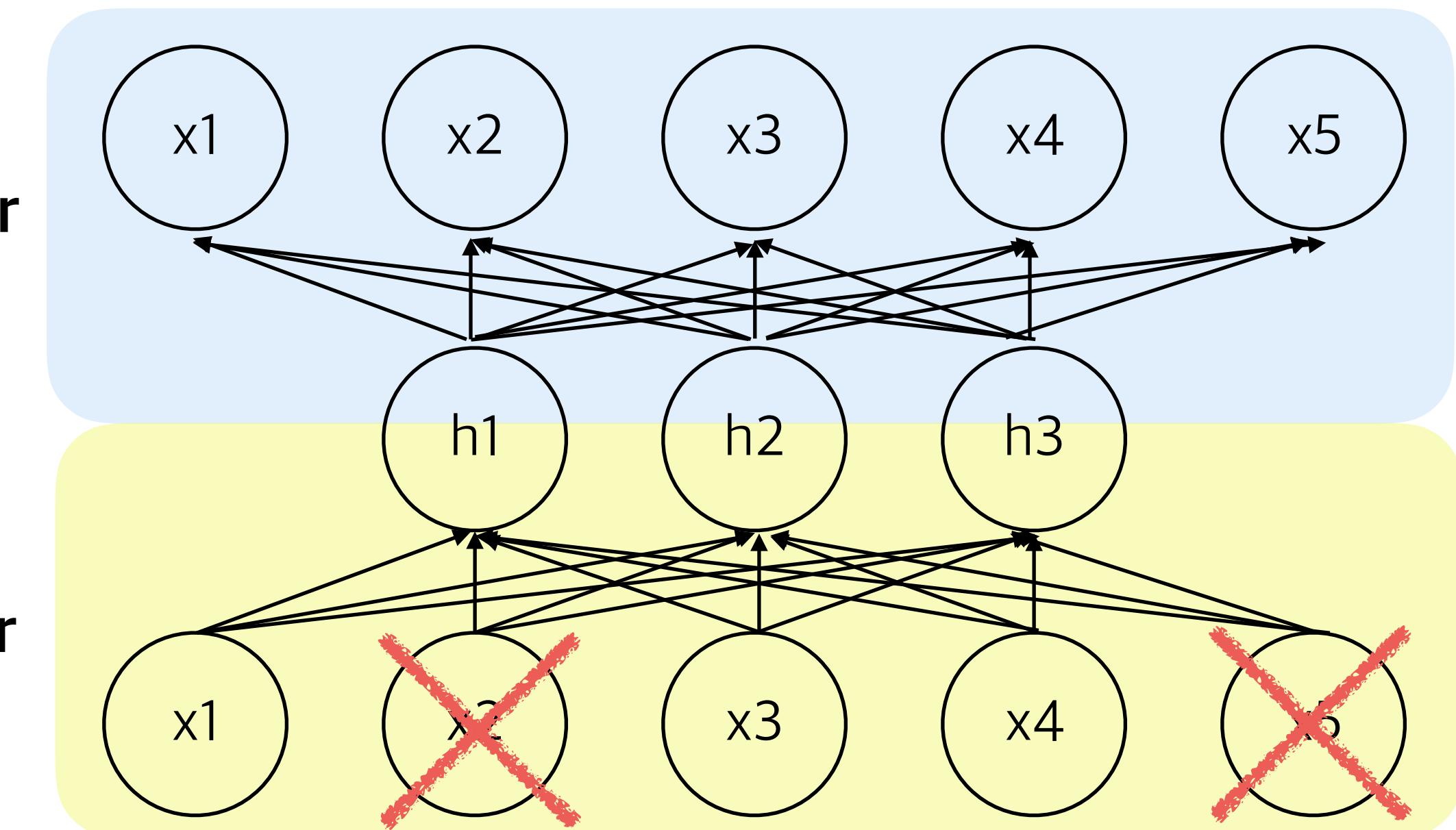
2) Generative Model: 특정 generative model의 variational bound를 최대화하는 것으로 Loss Function을 해석 (자세한 내용 생략)

<Manifold Learning Perspective>



<Generative Model Perspective>

Decoder



Encoder

with probability= p

with probability= p

eXem

Summary of Autoencoder

- Autoencoder는 Input 데이터와 동일한 Output을 갖는 Neural Network 모델
- 데이터에 핵심적이고 숨어있는 정보 또는 표현(latent representation)을 찾는 것을 목적으로하는 Unsupervised Learning
- 일반적으로 Input 데이터보다 낮은 수의 hidden neuron을 사용하고, Dimensionality Reduction에 활용
- Sparsity를 기준으로 사용할 경우, 더 많은 수의 hidden neuron 또한 사용 가능하고 각 뉴런은 각기 다른 표현에 반응함
- Autoencoder를 연속으로 학습시키며 여러 layer로 구성된 encoder-decoder 모델을 만들 수 있음(Stacked Model)
- Latent representation을 찾은 후, 일반적인 모델의 Initialization으로 사용하며 Fine Tuning 필요
- Noise에 대한 Robustness를 위하여 Denoising Autoencoder 사용 가능

Recurrent Neural Network

Recurrent Neural Network

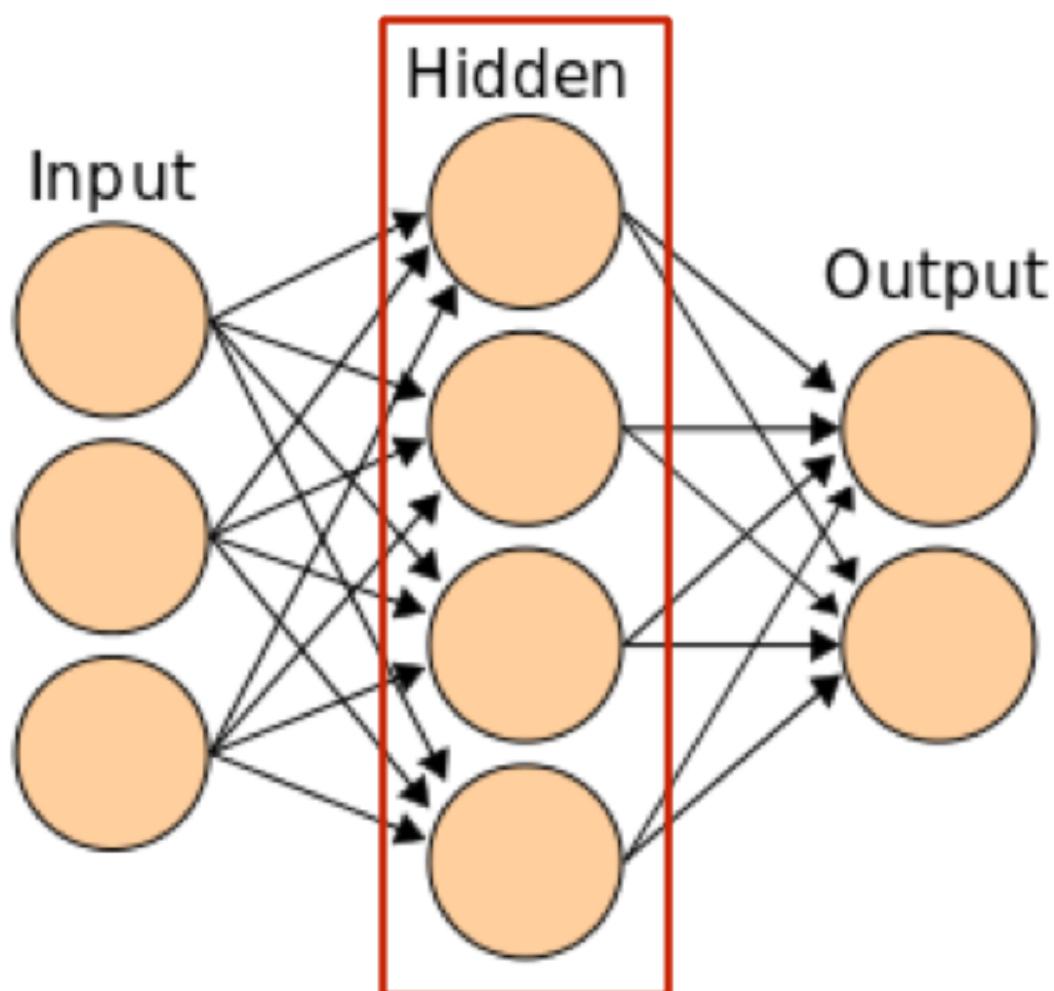
RNN(Recurrent Neural Network)은 Layer 내 연결이 없던 기존 Neural Network의 구조를 변형한 구조

RNN은 layer 내 순환 구조의 연결을 통해 Input의 연속적 특성(Sequential Characteristics)를 고려함

주로 음성, 음악, 문자열, 동영상 등 순차적인 정보 연속적 특성을 가진 데이터를 다룰 때 적합한 모델임

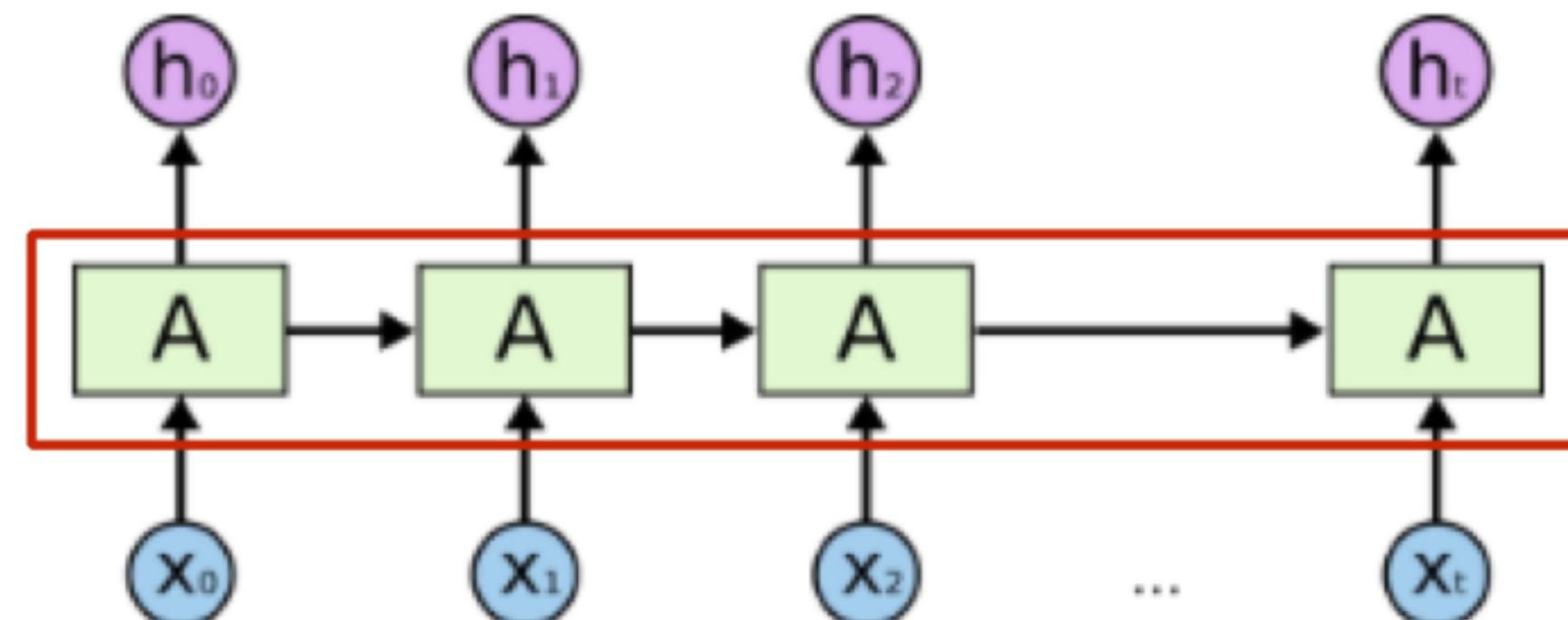
<Artificial Neural Network>

- No interaction in same layer



<Recurrent Neural Network>

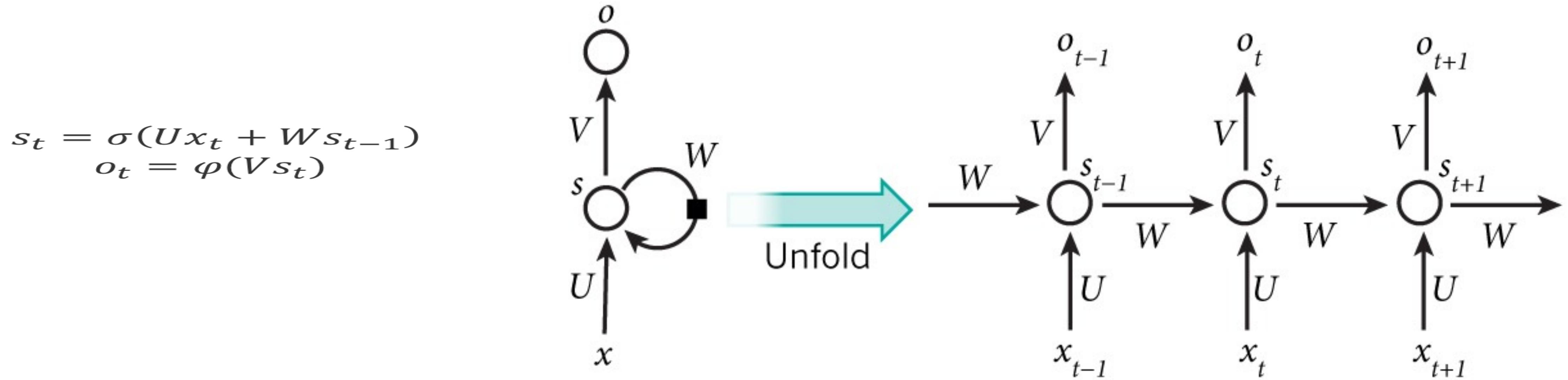
- 계산과정에서 Input Sequence가 고려됨
- Input과 이전 Input의 계산 결과값을 동시에 고려해가며 Output 계산
- Input-Hidden, Hidden-Hidden, Hidden-Output Weight Sharing!



RNN은 **Input Sequence**가 중요한 데이터를 다루기 적합한 구조를 가짐

Recurrent Neural Network

- 연속적인 관계를 갖는 Sequence data를 처리할 수 있도록 고안된 인공 신경망 (Neural Network) 모델
- 각 input은 t에 따른 순서(sequence)가 존재하고 이 결과값(state)을 다시 자신에게 input으로 주어 이전의 data들이 학습, 판단하는데 영향을 줌 (recursion, recurrent 재귀)
- 시간 Step에 관계 없이 parameter(U, V, W)를 공유하는 형태
- U 와 W 값을 통해 새로운 input(x_t)와 이전까지의 결과값 state(s_{t-1})를 얼마나 반영하여 계산할지 결정
- 원하는 학습 model에 따라 활성화 함수(activation function) σ 와 φ 를 적절히 선정

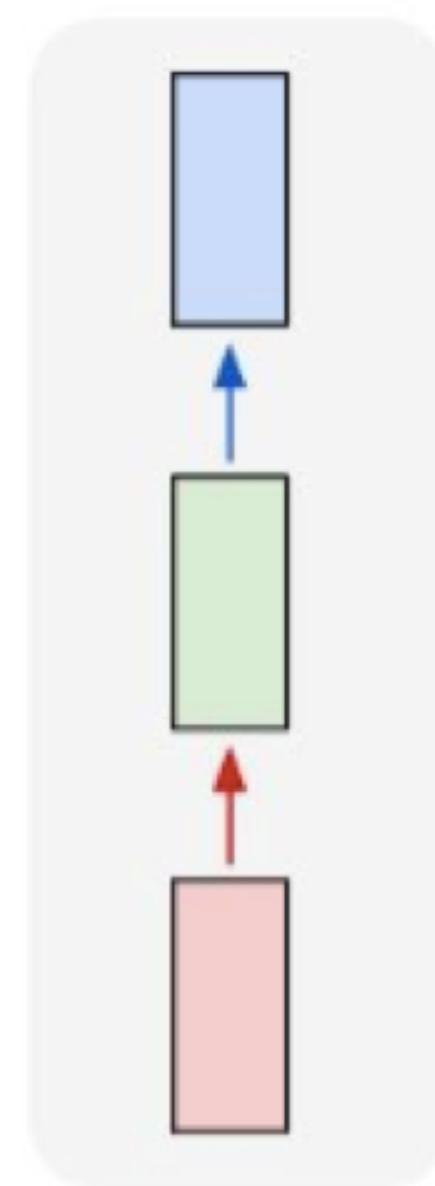


<basic RNN model>

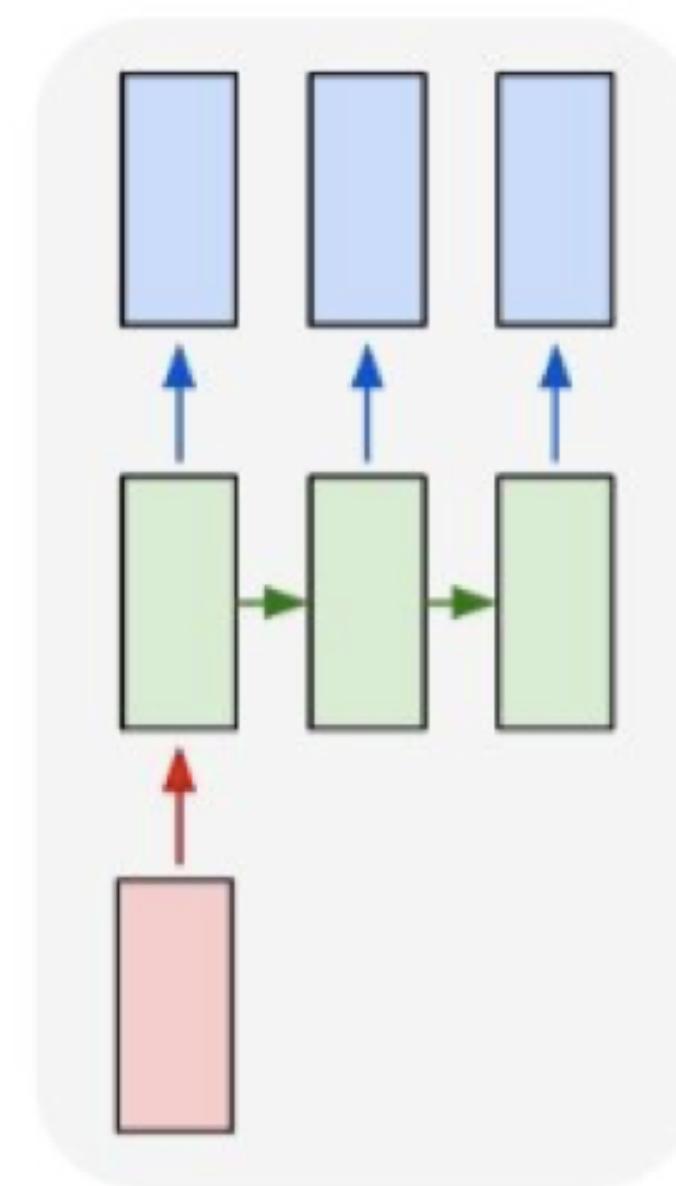
Different I/O Structure of RNN

RNN은 원하는 Input/Output을 다양하게 하여 모델 선택 가능

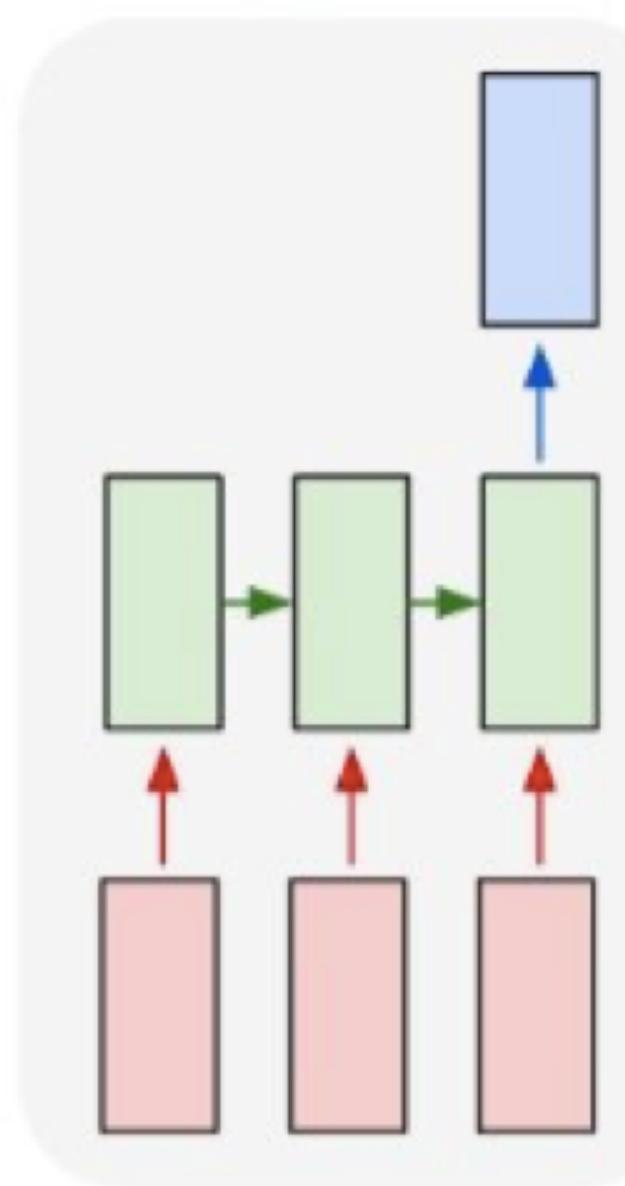
one to one



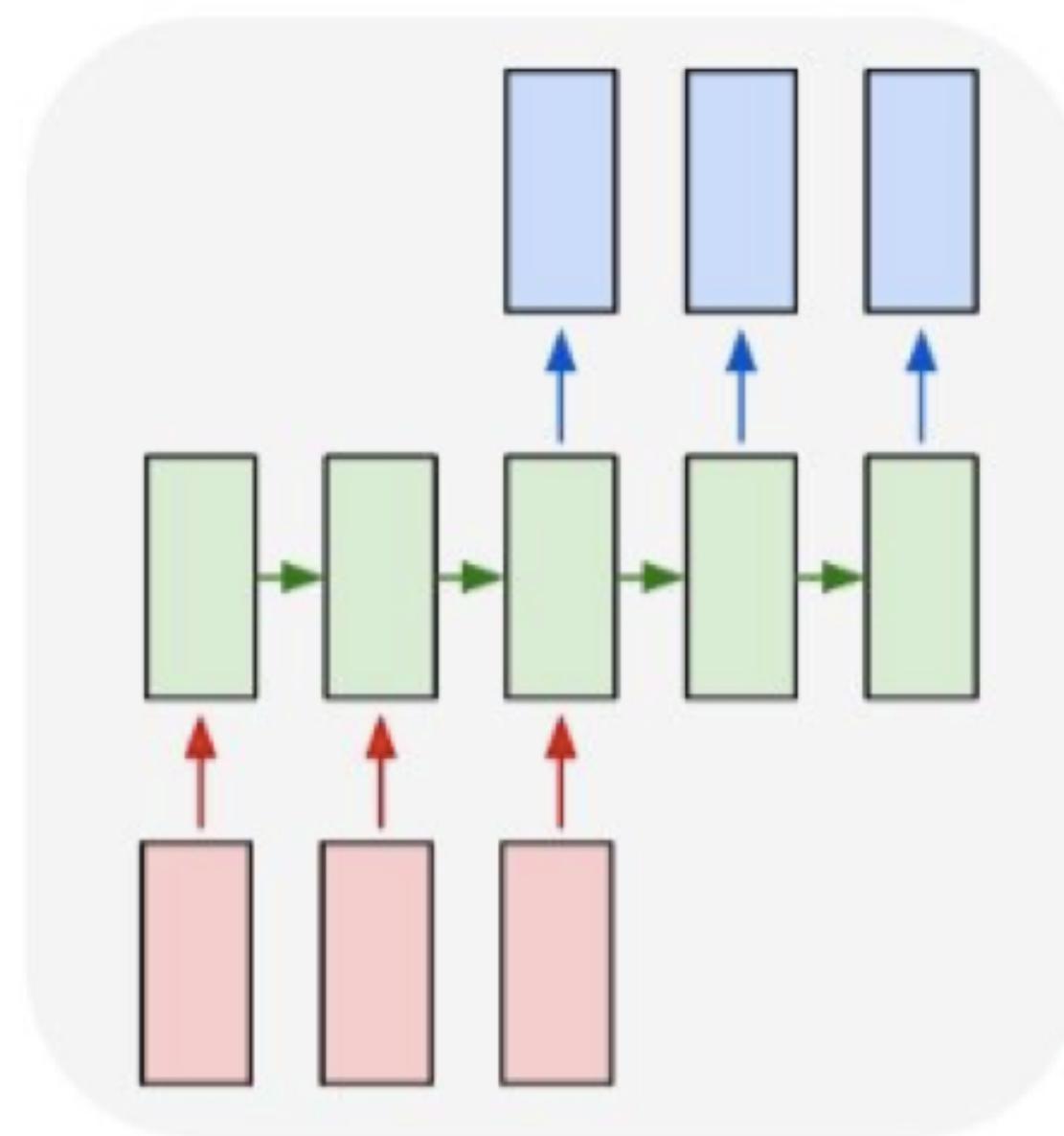
one to many



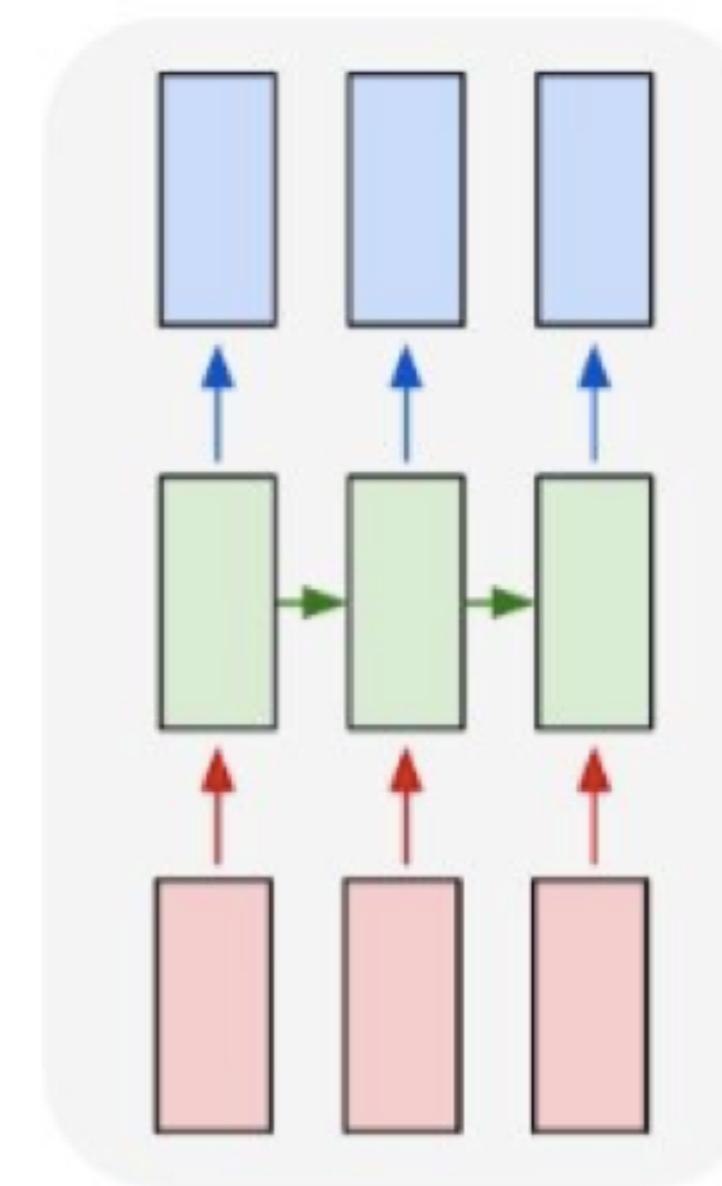
many to one



many to many



many to many



Vanilla Neural Networks

e.g. Image Captioning
image -> sequence of words

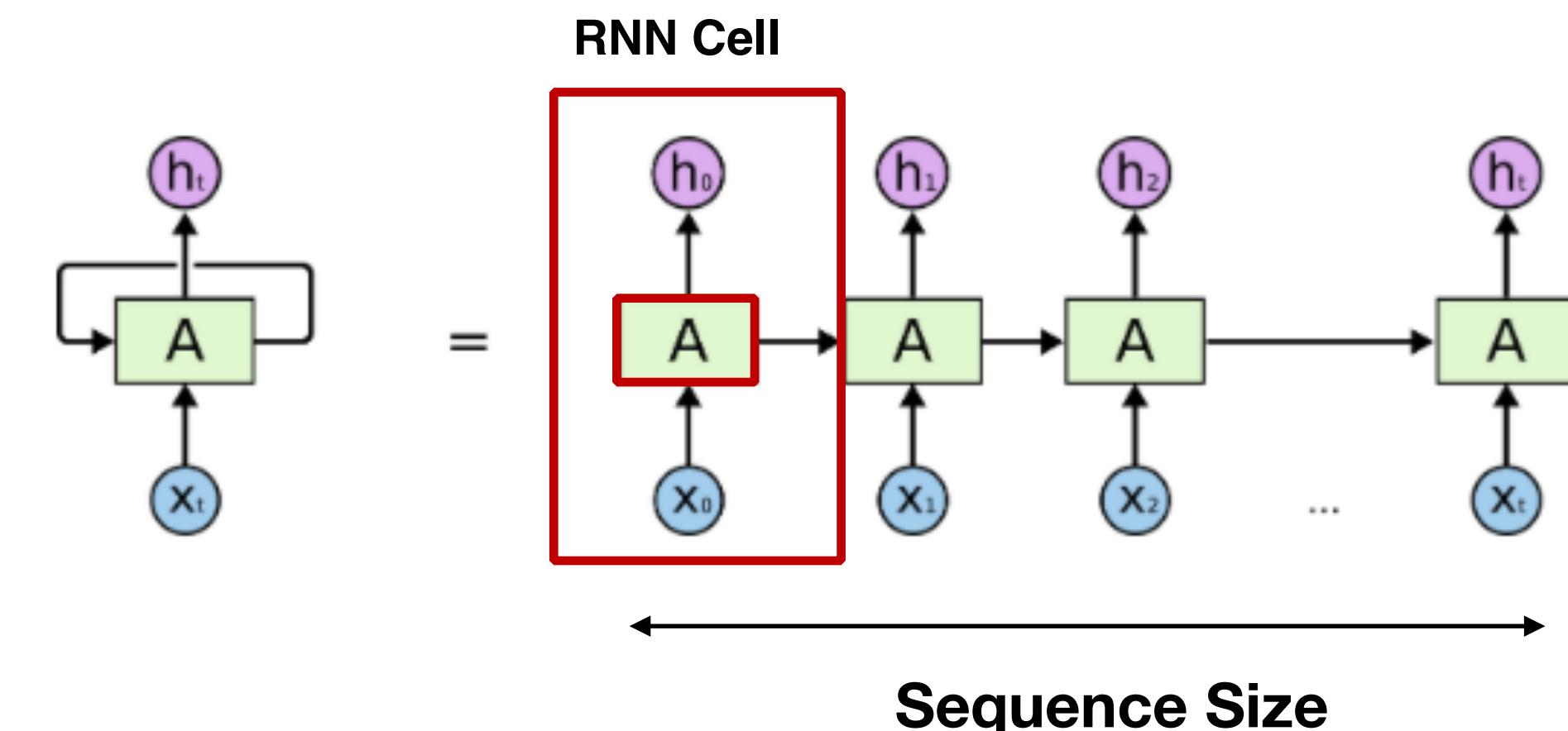
e.g. Sentiment Classification
sequence of words -> sentiment

e.g. Machine Translation
seq of words -> seq of words

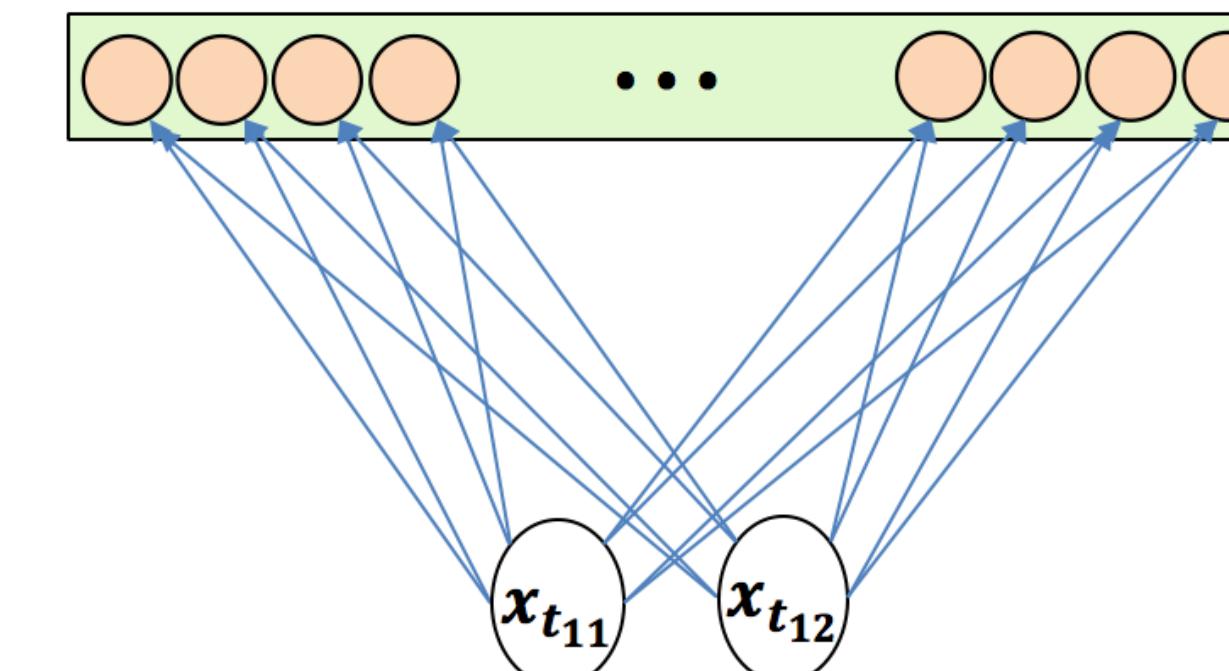
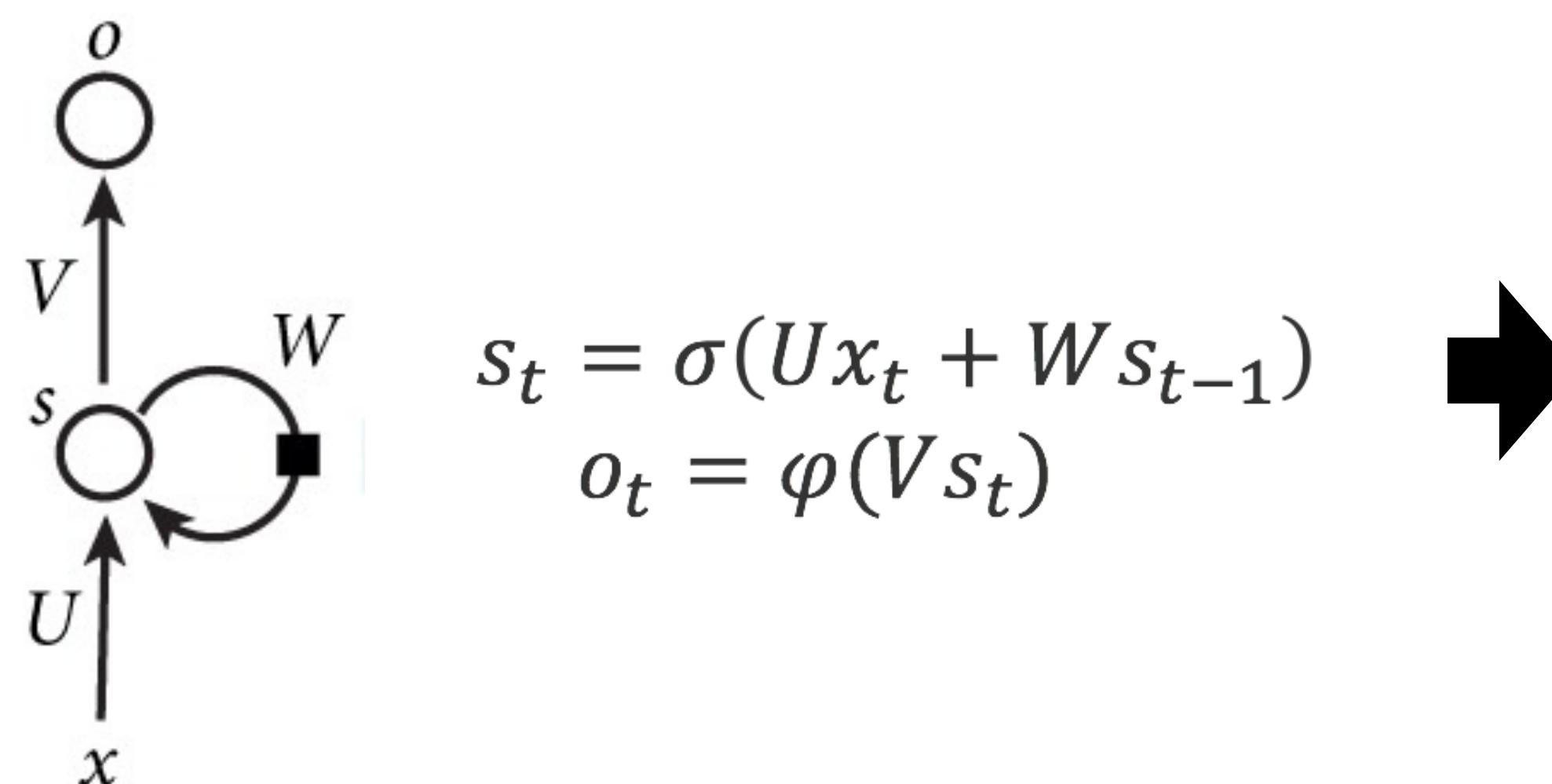
e.g. Video classification on frame level

Preliminary Knowledge for RNN

- RNN 모델은 구성 기본 단위인 RNN cell의 연속적인 구조로 이루어져 있음
- (Input) x_t : time step t에 해당하는 입력값
- (output) o_t : time step t에 해당하는 출력값
- (hidden state) s_t : time step t에 해당하는 상태값



<Calculation in RNN>



of : # of hidden state
of : input feature dimension

Application of Recurrent Neural Network

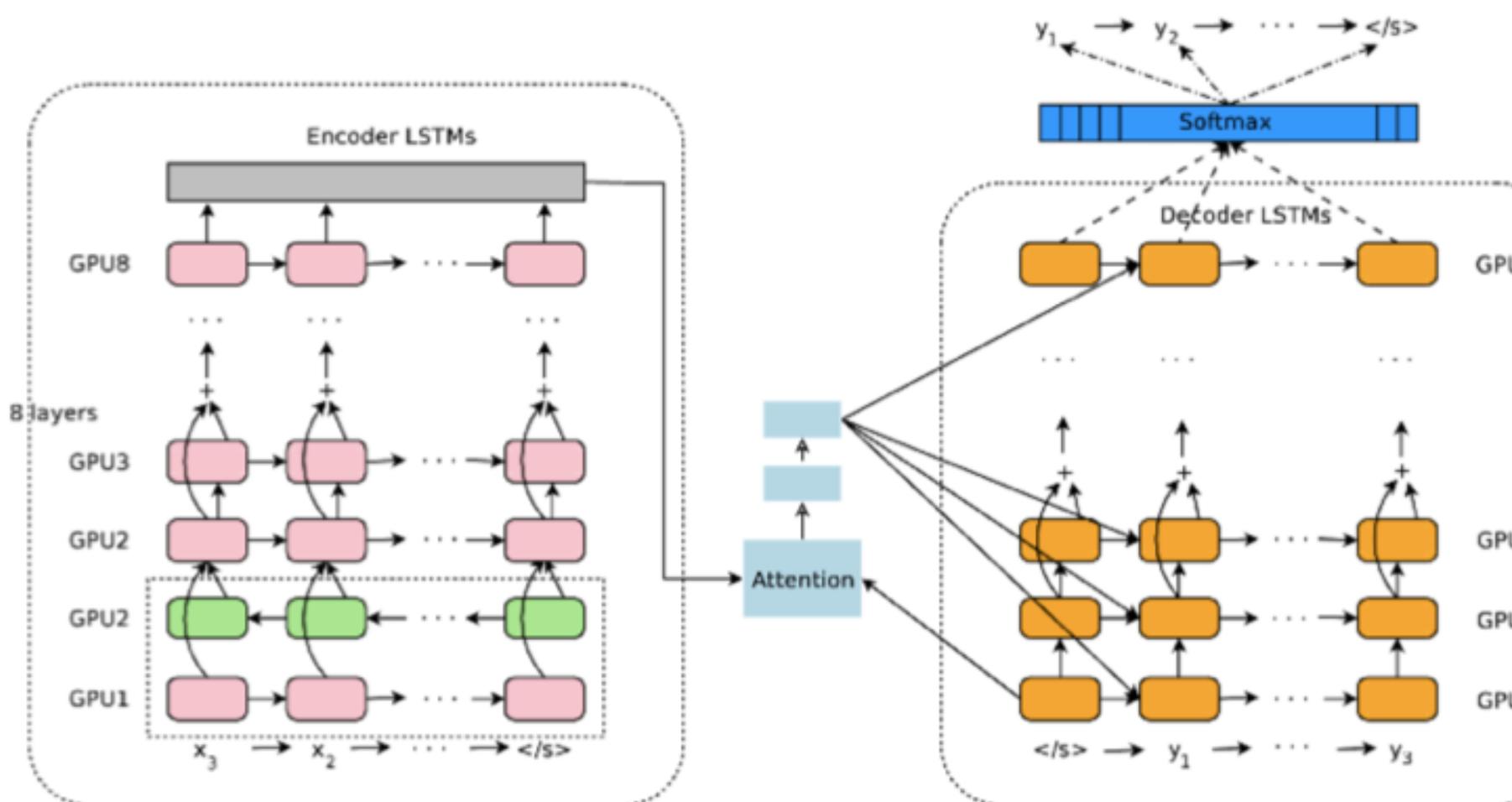
Recurrent Neural Network는 Sequential Data를 다루는 다양한 분야에 적용되고 있음

<Translator>



Google
Translate

Break through language barriers.



<image Captioning>

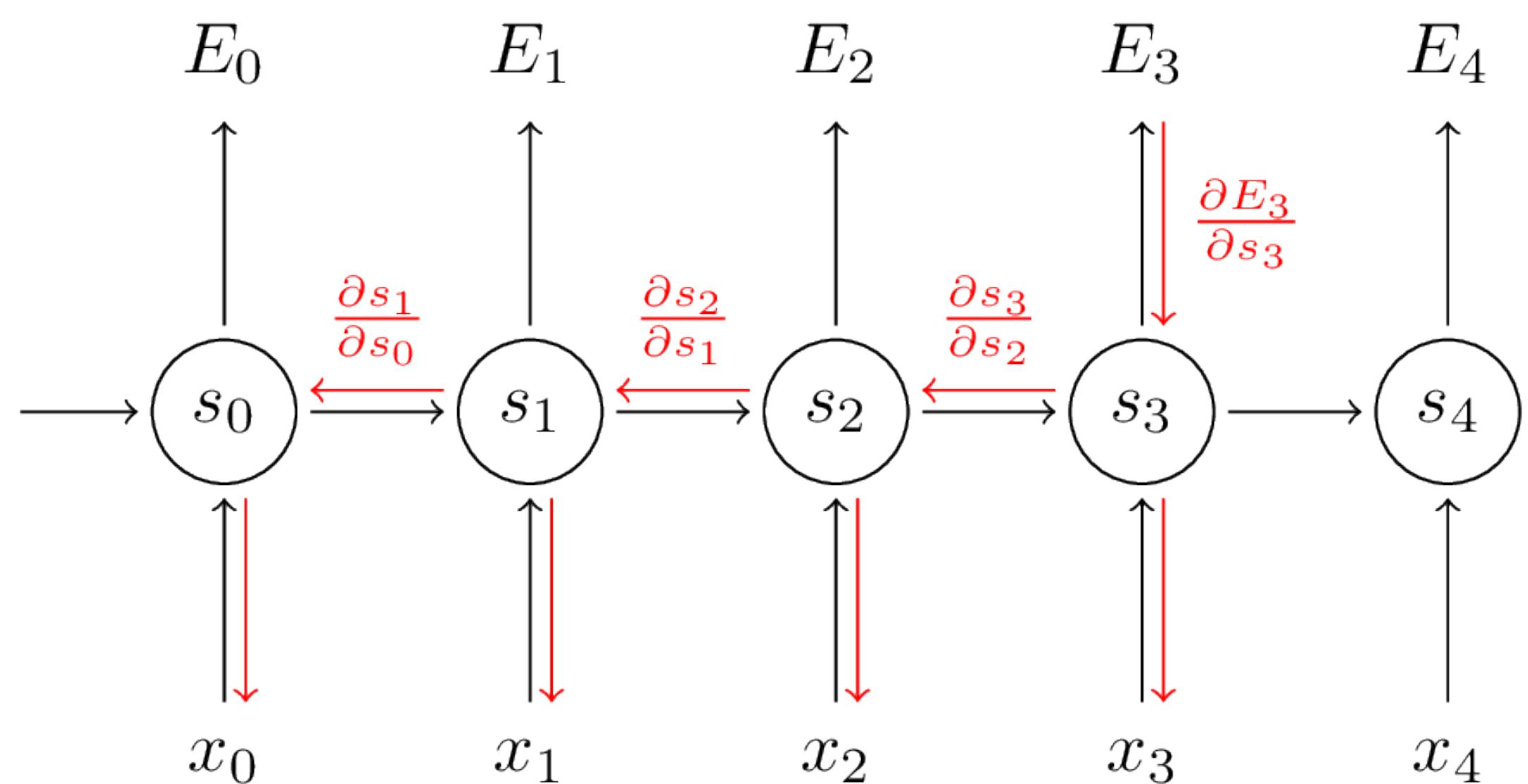


LSTM & GRU

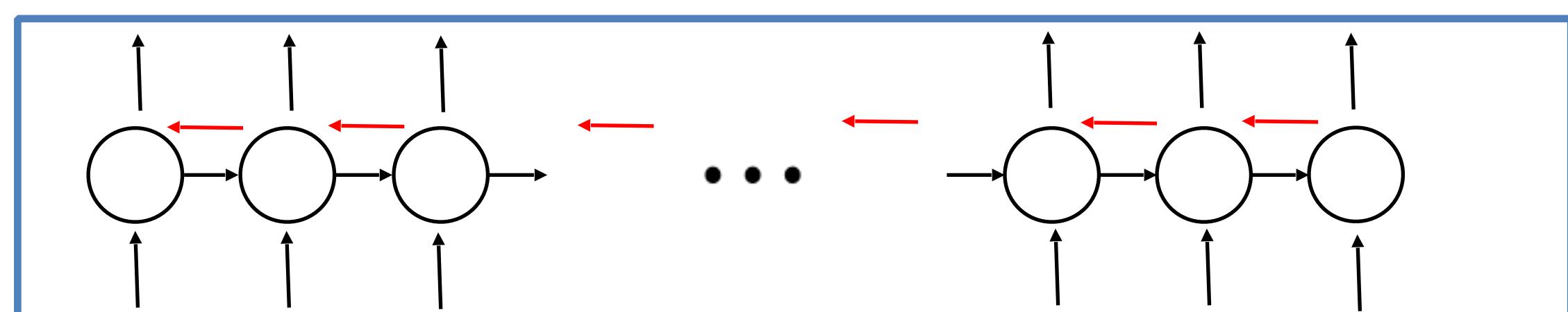
Gradient Vanishing in RNN

- ▣ 기존 RNN 구조는 시간에 따른 가중치 W 를 학습할 때 Gradient Vanishing 현상으로 인해 장기적인 관계가 학습되지 않는 문제를 가지고 있음
- ▣ RNN 구조는 Short Term Memory 만 반영하여 학습되는 경향이 있음

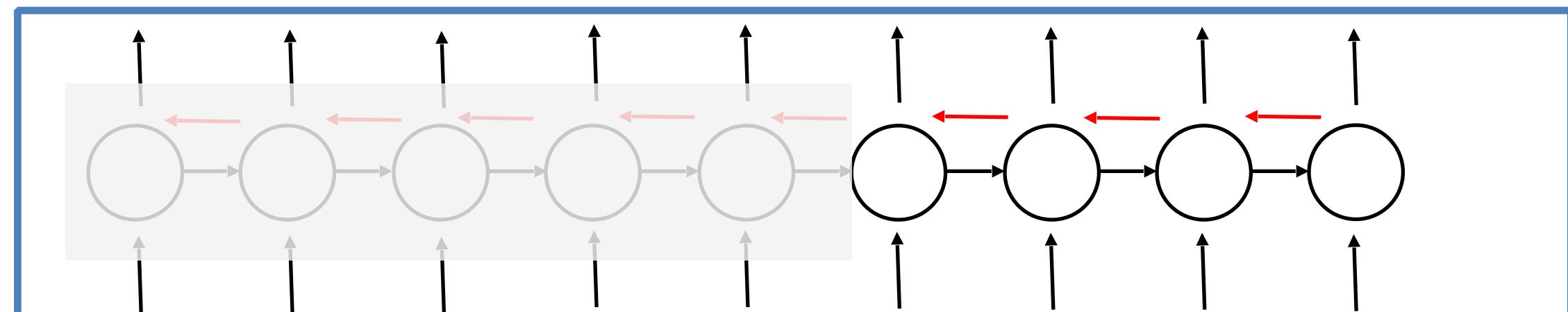
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$



<Long Term Memory>



<Short Term Memory>



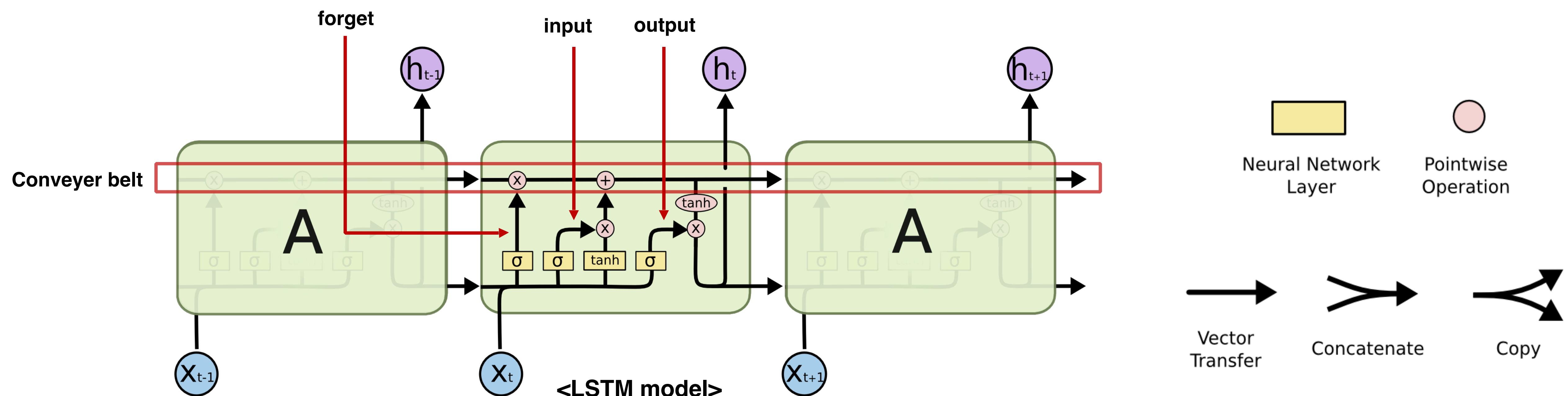
Long Short Term Memory

□ RNN model에서의 Gradient vanishing 문제를 해결하도록 디자인 된 model

□ 기존 RNN에서 3개의 Gate가 추가된 구조로 변경

- input gate : ‘현재 정보를 기억하기’위한 gate
- forget gate : ‘과거 정보를 잊기’위한 gate
- output gate : 현재 step의 cell state를 output에 얼마나 반영할 지 가중치를 결정

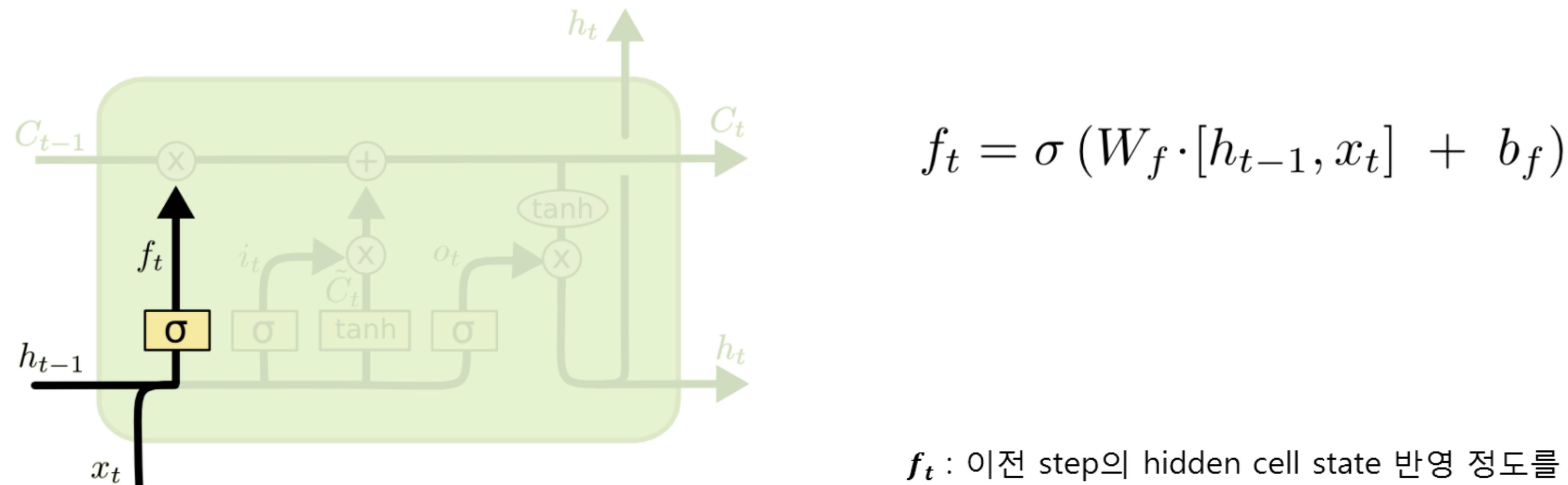
□ Conveyer belt에서 hidden state를 +연산자를 통해 갱신하여, gradient 전파에 용이하도록 modeling



LSTM step.1

□ Forget gate(f_t)

- 이전 Time step 의 output(C_{t-1})을 time step의 hidden state(C_t)에 반영(forget)하는 정도(0~1)를 결정하는 gate
- h_{t-1} 와 x_t 를 W_f 와 곱한 뒤 이를 sigmoid를 통해 forget gate의 가중치 값을 계산
- 계산된 forget gate 값이 1에 가까울수록 C_{t-1} 를 현재 step의 값 계산에 반영



f_t : 이전 step의 hidden cell state 반영 정도를 나타내는 가중치

W_f : f_t 를 구하기 위한 parameter

σ : activation function (sigmoid function)

h_{t-1} : 이전 step의 output

C_{t-1} : 이전 step으로부터의 hidden cell state

b_f : bias value

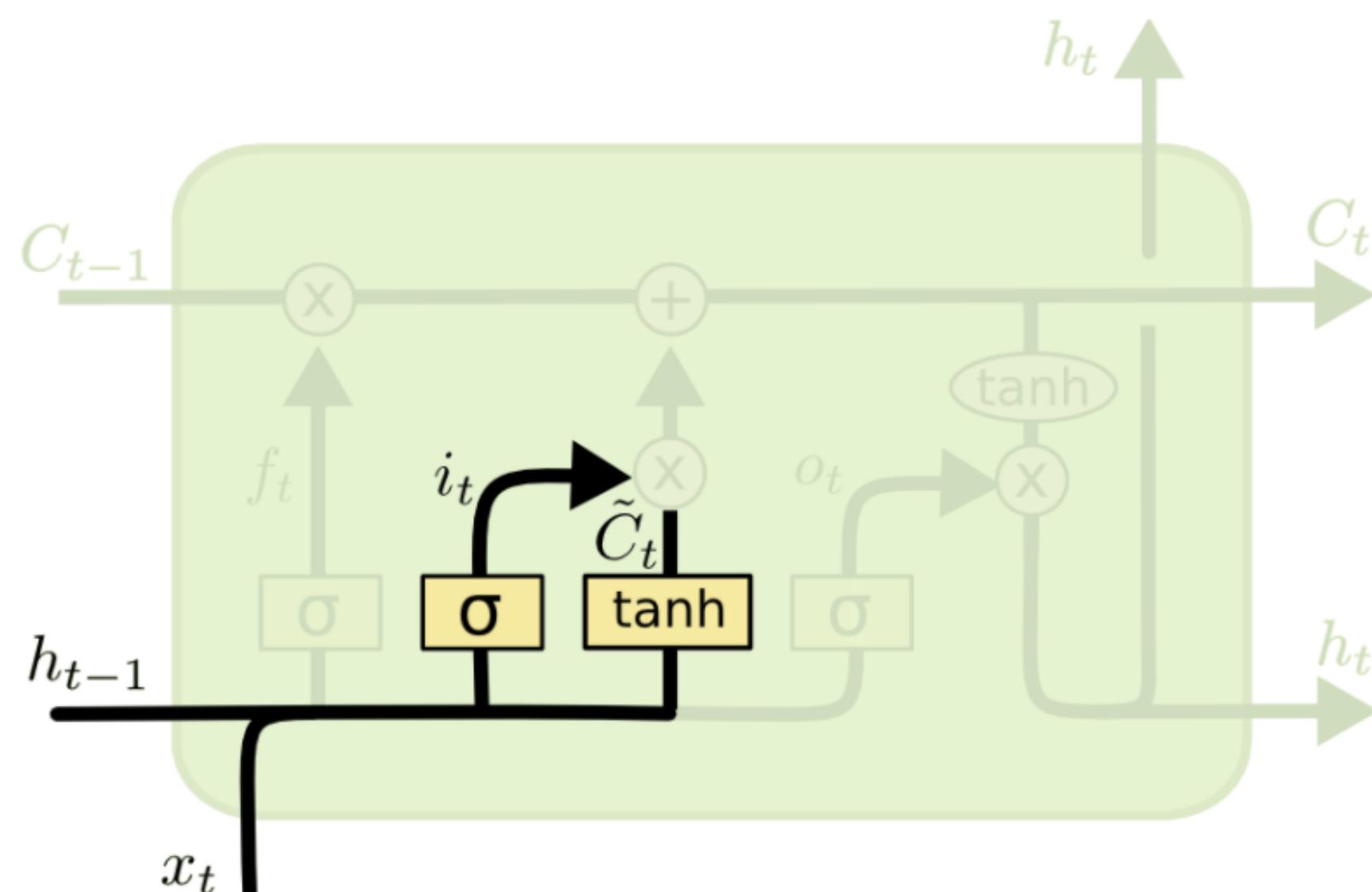
LSTM step.2

□ Predict cell state(\tilde{C}_t)

- input gate를 통해 예측된 새로운 cell state
- 기존의 RNN에서는 이를 output으로 사용하였음

□ Input gate(i_t)

- new cell state(\tilde{C}_t)을 현재 time step의 hidden state(C_t)에 반영(forget)하는 정도(0~1)를 결정하는 gate
- h_{t-1} 와 x_t 를 W_i 에 곱한 뒤 sigmoid를 취하여 new cell state를 얼마나 반영할 지 결정
- 계산된 input gate 값이 1에 가까울수록 \tilde{C}_t 를 현재 step의 hidden state(C_t)값 계산에 반영



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

i_t : 현재 step에서 들어온 input의 반영 정도를 나타내는 가중치

W_i : i_t 를 구하기 위한 parameter

σ : activation function (sigmoid function)

h_{t-1} : 이전 step의 output

C_{t-1} : 이전 step으로부터의 hidden cell state

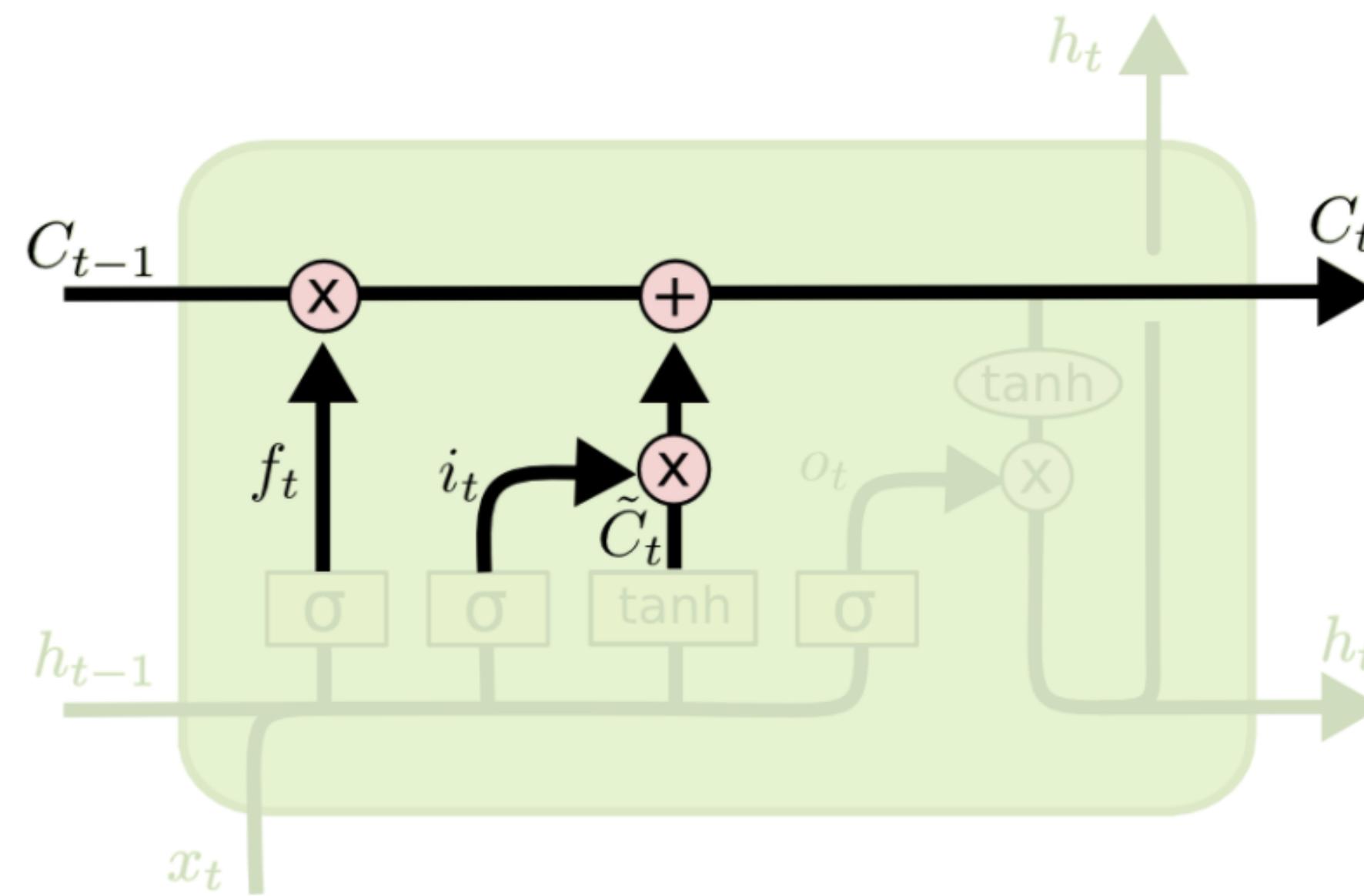
\tilde{C}_t : 현재 step의 input과 h_{t-1} 를 통해 예측된 cell state

$b_{i,C}$: bias value

LSTM step.3

□ Update C_t

- step1, step2에서 구했던 forget 가중치 f_t 와 input 가중치 i_t , predict cell state \tilde{C}_t 를 이용해 new cell state C_t 결정
- step2에서의 \tilde{C}_t 와 다름을 주의



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

f_t : 이전 step의 hidden cell state 반영 정도를 나타내는 가중치

i_t : 현재 step에서 들어온 input의 반영 정도를 나타내는 가중치

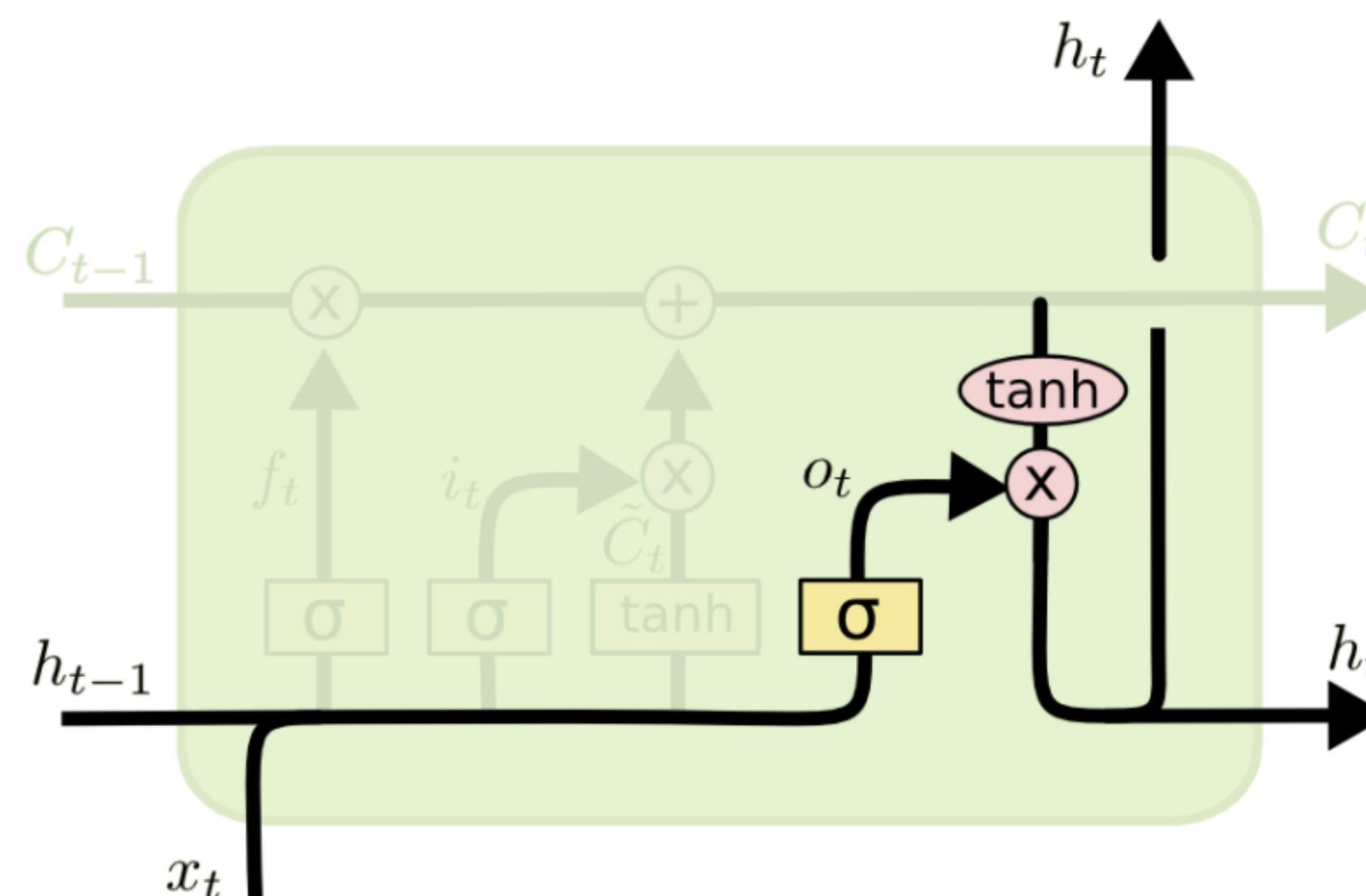
C_{t-1} : 이전 step으로부터의 hidden cell state

\tilde{C}_t : 현재 step의 input과 h_{t-1} 를 통해 예측된 new cell state

LSTM step.4

□ Output gate(o_t)

- C_t 를 filtered한 new hidden cell state에 tanh를 취한 뒤 실제 output에 반영하는 정도를 결정하는 gate
- h_{t-1} 와 x_t 를 W_o 에 곱한 뒤 sigmoid를 취하여 C_t 를 h_t 에 얼마나 반영할 지 결정
- C_t 를 tanh에 넣어 -1과 1사이의 값을 갖도록 만든 뒤 o_t 를 곱하여 h_t 결정(output)



$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh (C_t)$$

o_t : 현재 step에서 들어온 input의 반영 정도를 나타내는 가중치
 W_o : o_t 를 구하기 위한 parameter

σ : activation function (sigmoid function)

h_{t-1} : 이전 step의 output

C_t : 새롭게 구해진 현재 step의 hidden cell state

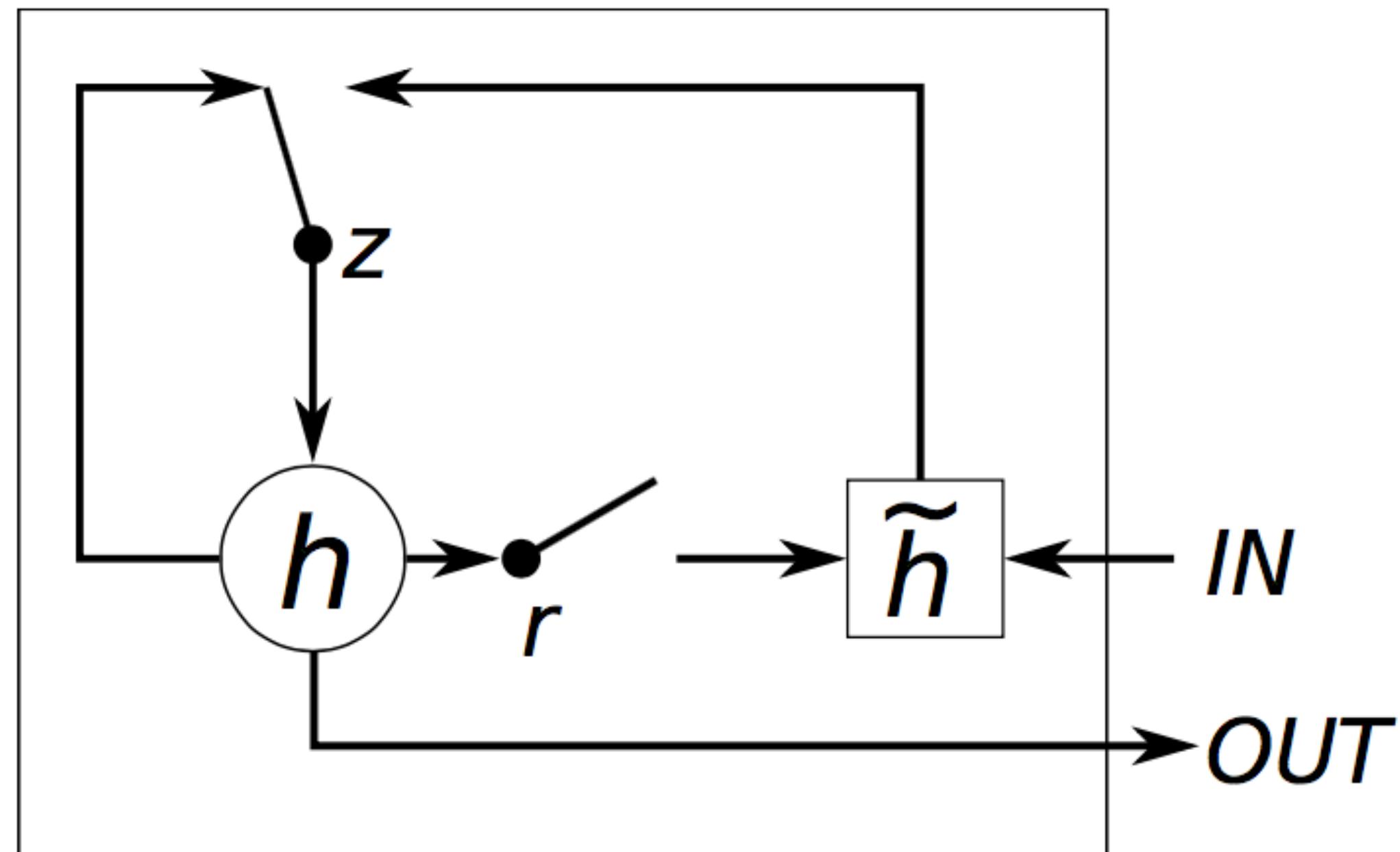
h_t : 새로운 output

b_o : bias value

Gated Recurrent Unit

□ GRU

- LSTM model에서 output gate를 생략한 모델 (step4 생략)
- memory cell에서 보이는 값과 hidden state 값이 같음(memory cell에서 결과 출력)



<GRU model>

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

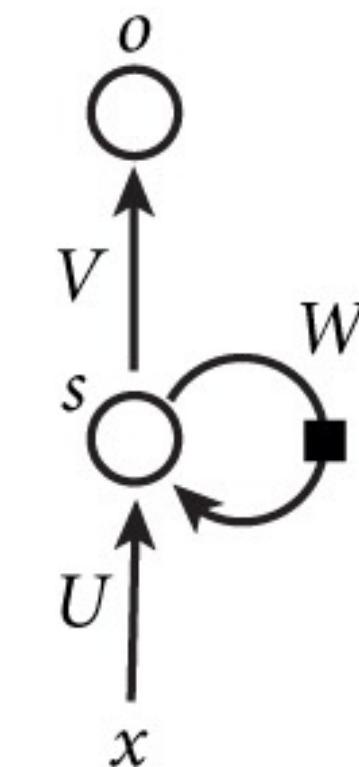
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

Recurrent Models in Tensorflow

RNN structure

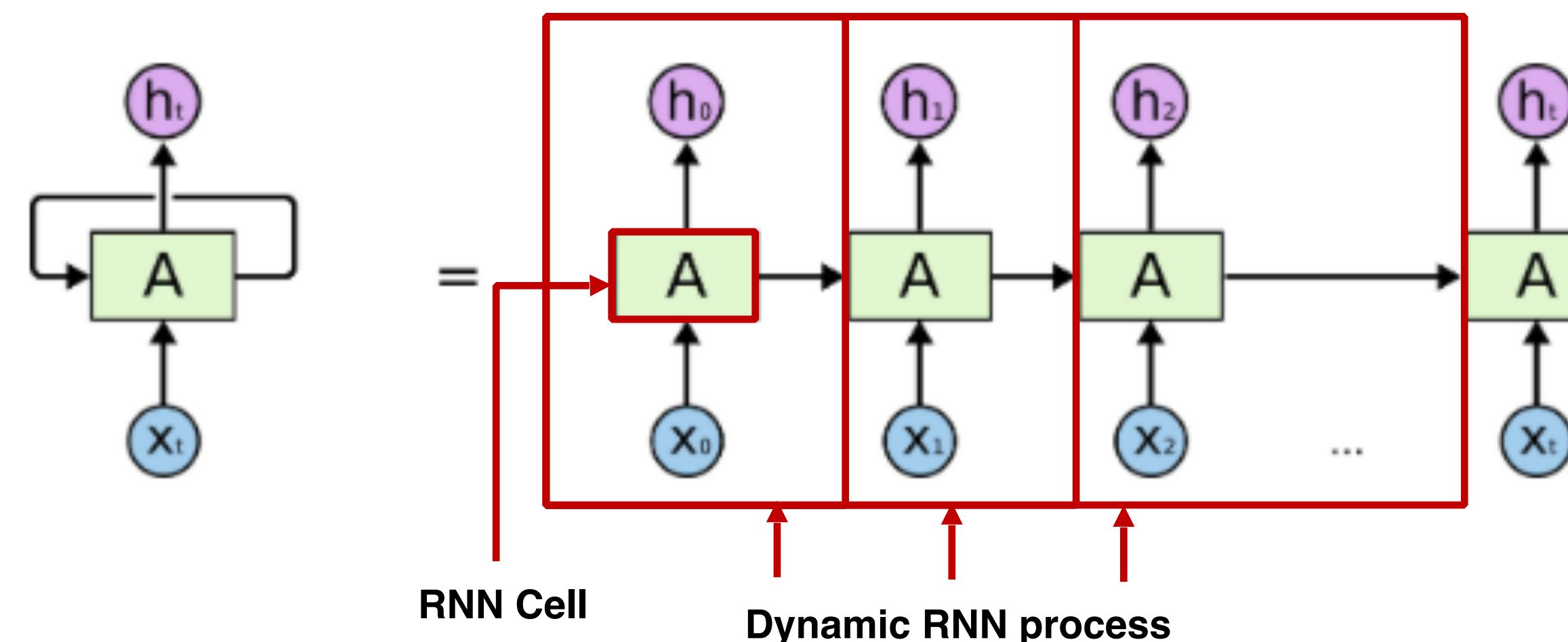
□ RNN Cell : 기존의 Neural Network에서 layer의 역할을 수행하는 Cell

- RNN structure의 기본 단위
- BasicRNN, LSTM and GRU 등의 structure가 존재
- 호출 시 입력한 hidden dimension에 따라 가중치 U , W , V 의 dimension을 정하고, Cell 안에서 자동으로 생성



□ Dynamic RNN : 기본적인 RNN Cell 구조를 이용하여 여러 시간의 결과값을 연속적으로 출력할 수 있는 프로세스 구조

- 사용하고 싶은 Recurrent Cell을 입력하여 여러 시간의 출력값을 한번에 계산할 수 있도록 구현되어 있음
- 입력된 input의 seq_size 크기와 같은 크기를 갖는 output과 hidden state를 리턴함



▣ RNN과 관련한 모든 모듈은 tf.contrib.rnn에 포함되어 있음 (tf 1.0 버전 이상부터 contrib에 통합되었으니 검색하여 사용할 경우 주의)

Classes

`class AttentionCellWrapper` : Basic attention cell wrapper.

`class BasicLSTMCell` : Basic LSTM recurrent network cell.

`class BasicRNNCell` : The most basic RNN cell.

`class BidirectionalGridLSTMCell` : Bidirectional GridLstm cell.

`class CompiledWrapper` : Wraps step execution in an XLA JIT scope.

`class CoupledInputForgetGateLSTMCell` : Long short-term memory unit (LSTM) recurrent network cell.

`class DeviceWrapper` : Operator that ensures an RNNCell runs on a particular device.

`class DropoutWrapper` : Operator adding dropout to inputs and outputs of the given cell.

`class EmbeddingWrapper` : Operator adding input embedding to the given cell.

`class FusedRNNCell` : Abstract object representing a fused RNN cell.

`class FusedRNNCellAdaptor` : This is an adaptor for RNNCell classes to be used with `FusedRNNCell`.

`class GLSTMCell` : Group LSTM cell (G-LSTM).

`class GRUBlockCell` : Block GRU cell implementation.

`class GRUCell` : Gated Recurrent Unit cell (cf. <http://arxiv.org/abs/1406.1078>).

`class GridLSTMCell` : Grid Long short-term memory unit (LSTM) recurrent network cell.

`class HighwayWrapper` : RNNCell wrapper that adds highway connection on cell input and output.

`class InputProjectionWrapper` : Operator adding an input projection to the given cell.

`class IntersectionRNNCell` : Intersection Recurrent Neural Network (+RNN) cell.

`class LSTMBlockCell` : Basic LSTM recurrent network cell.

`class LSTMBlockFusedCell` : FusedRNNCell implementation of LSTM.

`class LSTMBlockWrapper` : This is a helper class that provides housekeeping for LSTM cells.

`class LSTMCell` : Long short-term memory unit (LSTM) recurrent network cell.

`class LSTMStateTuple` : Tuple used by LSTM Cells for `state_size`, `zero_state`, and output state.

`class LayerNormBasicLSTMCell` : LSTM unit with layer normalization and recurrent dropout.

`class MultiRNNCell` : RNN cell composed sequentially of multiple simple cells.

`class NASCell` : Neural Architecture Search (NAS) recurrent network cell.

`class OutputProjectionWrapper` : Operator adding an output projection to the given cell.

`class PhasedLSTMCell` : Phased LSTM recurrent network cell.

`class RNNCell` : Abstract object representing an RNN cell.

`class ResidualWrapper` : RNNCell wrapper that ensures cell inputs are added to the outputs.

`class TimeFreqLSTMCell` : Time-Frequency Long short-term memory unit (LSTM) recurrent network cell.

`class TimeReversedFusedRNN` : This is an adaptor to time-reverse a FusedRNNCell.

`class UGRNNCell` : Update Gate Recurrent Neural Network (UGRNN) cell.

tf.contrib.rnn.BasicRNNCell

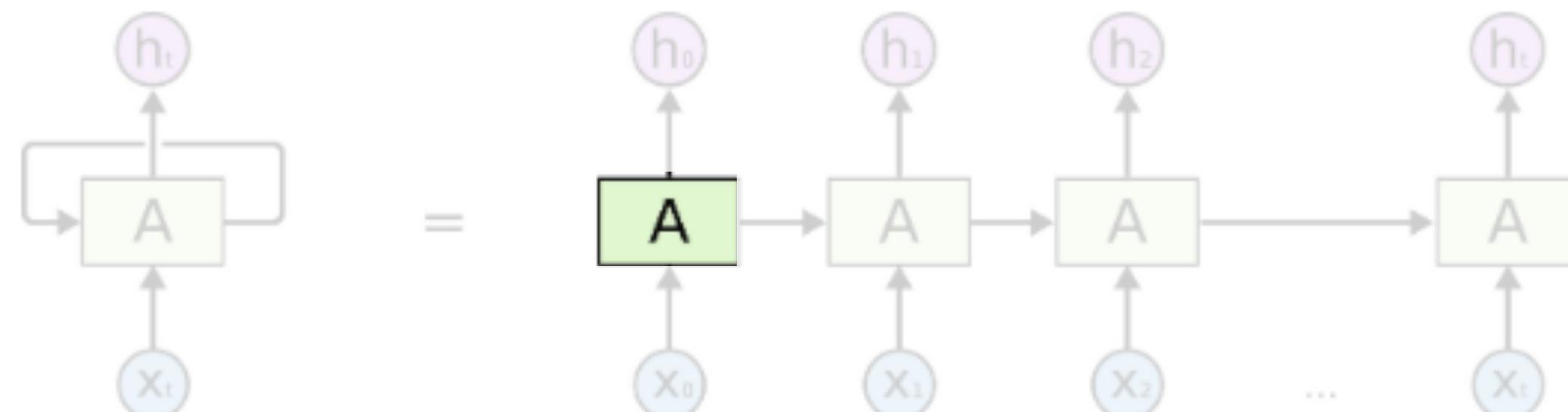
□ 기본 모델의 단위인 BasicRNNCell을 생성하기 위한 클래스

□ __init__

- num_units : Cell이 포함할 hidden state(unit)의 개수를 입력
- activation : activation function으로 사용할 함수를 지정(default : tanh)
- reuse : 가중치 U, W 변수의 재사용 여부를 결정

□ __call__

- BasicRNNCell을 호출할 때, 입력값(input)과 이전 상태(state)를 입력으로 받음
- dynamicRNN 구조를 사용하지 않고, Cell 단위로 직접 사용할 때 이용함



tf.contrib.rnn.BasicRNNCell

__init__

```
__init__(  
    num_units,  
    activation=None,  
    reuse=None  
)
```

__call__

```
__call__(  
    inputs,  
    state,  
    scope=None  
)
```

tf.contrib.rnn.BasicLSTMCell

□ 기본 모델의 단위인 BasicLSTMCell을 생성하기 위한 클래스

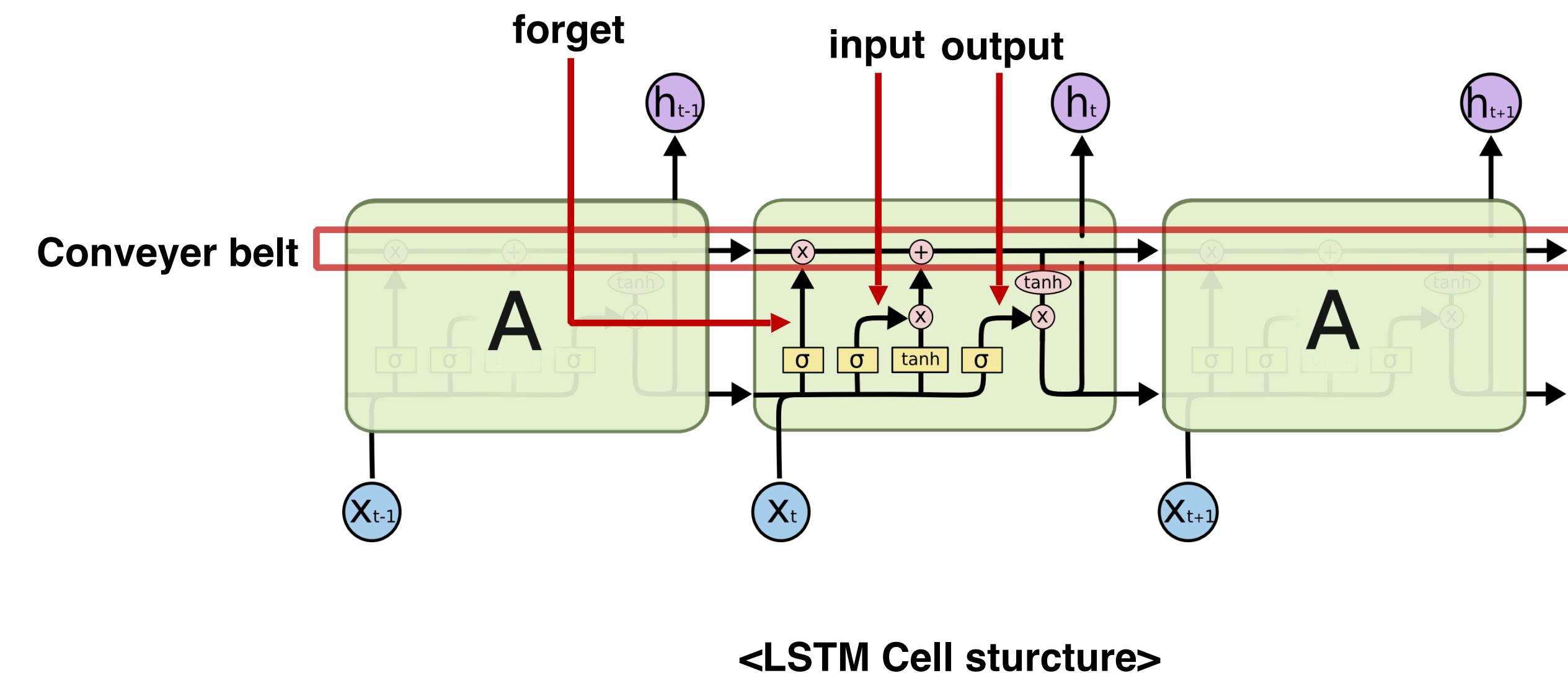
□ __init__

- num_units : hidden_dimension의 값을 입력
- forget_bias : training 초기에 forget scale을 설정
- activation : activation function으로 사용할 함수를 지정(default : tanh)
- reuse : 가중치 U, W 변수의 재사용 여부를 결정

tf.contrib.rnn.BasicLSTMCell

__init__

```
__init__(  
    num_units,  
    forget_bias=1.0,  
    state_is_tuple=True,  
    activation=None,  
    reuse=None  
)
```



tf.contrib.rnn.GRUCell

□ 기본 모델의 단위인 GRUCell을 생성하기 위한 클래스

□ __init__

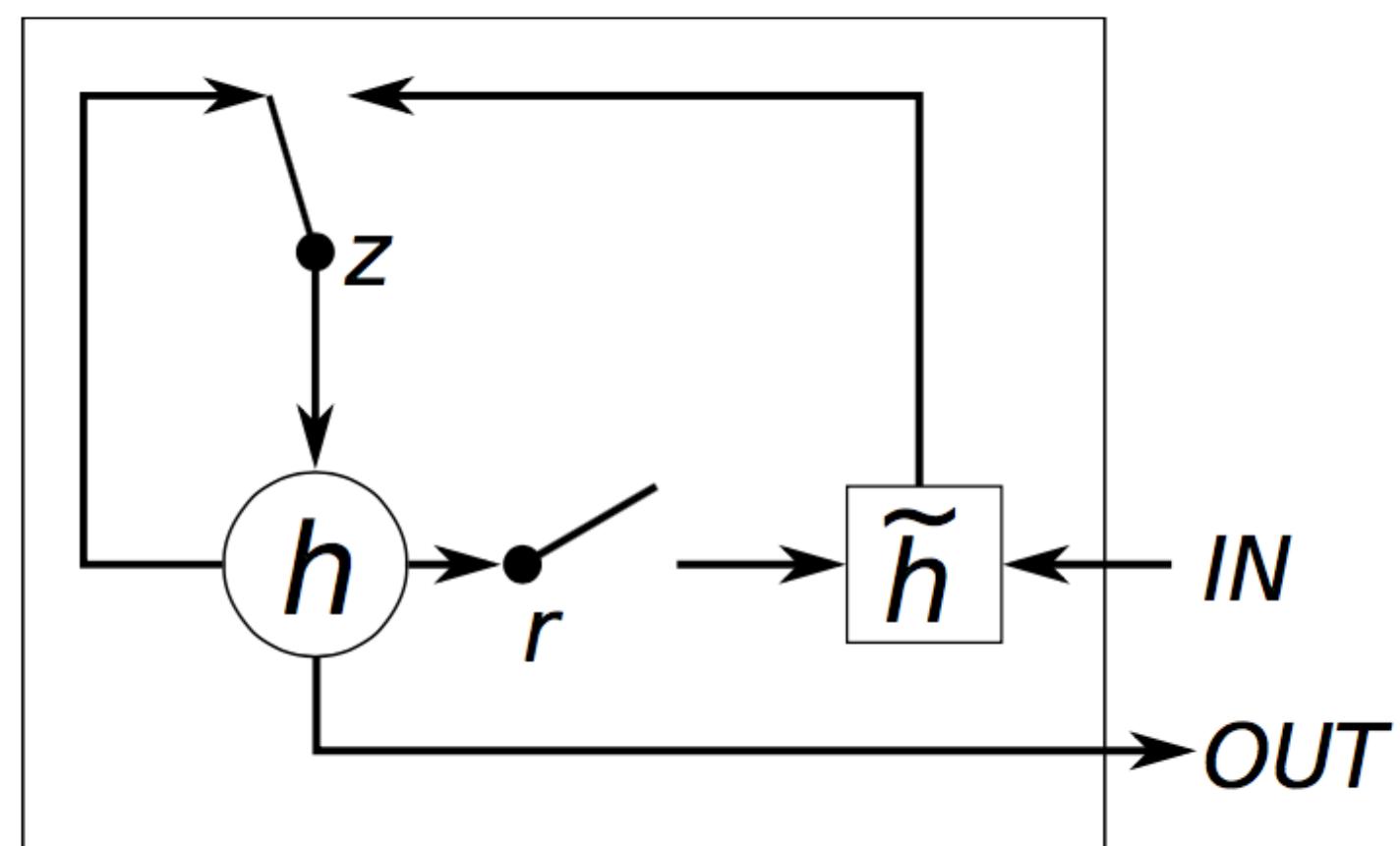
- num_units : hidden_dimension의 값을 입력
- activation : activation function으로 사용할 함수를 지정(default : tanh)
- reuse : 가중치 U, W 변수의 재사용 여부를 결정
- kernel(bias)_initializer : 가중치의 초기값을 설정하기 위한 Initializer를 별도로 선정 가능

(BasicRNNCell, BasicLSTMCell은 Glorot Initializer (Xavier)를 기본으로 사용하도록 설정되어 있음)

tf.contrib.rnn.GRUCell

__init__

```
--init__(  
    num_units,  
    activation=None,  
    reuse=None,  
    kernel_initializer=None,  
    bias_initializer=None  
)
```



$$z = \sigma(x_t U^z + s_{t-1} W^z)$$

$$r = \sigma(x_t U^r + s_{t-1} W^r)$$

$$h = \tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$

$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

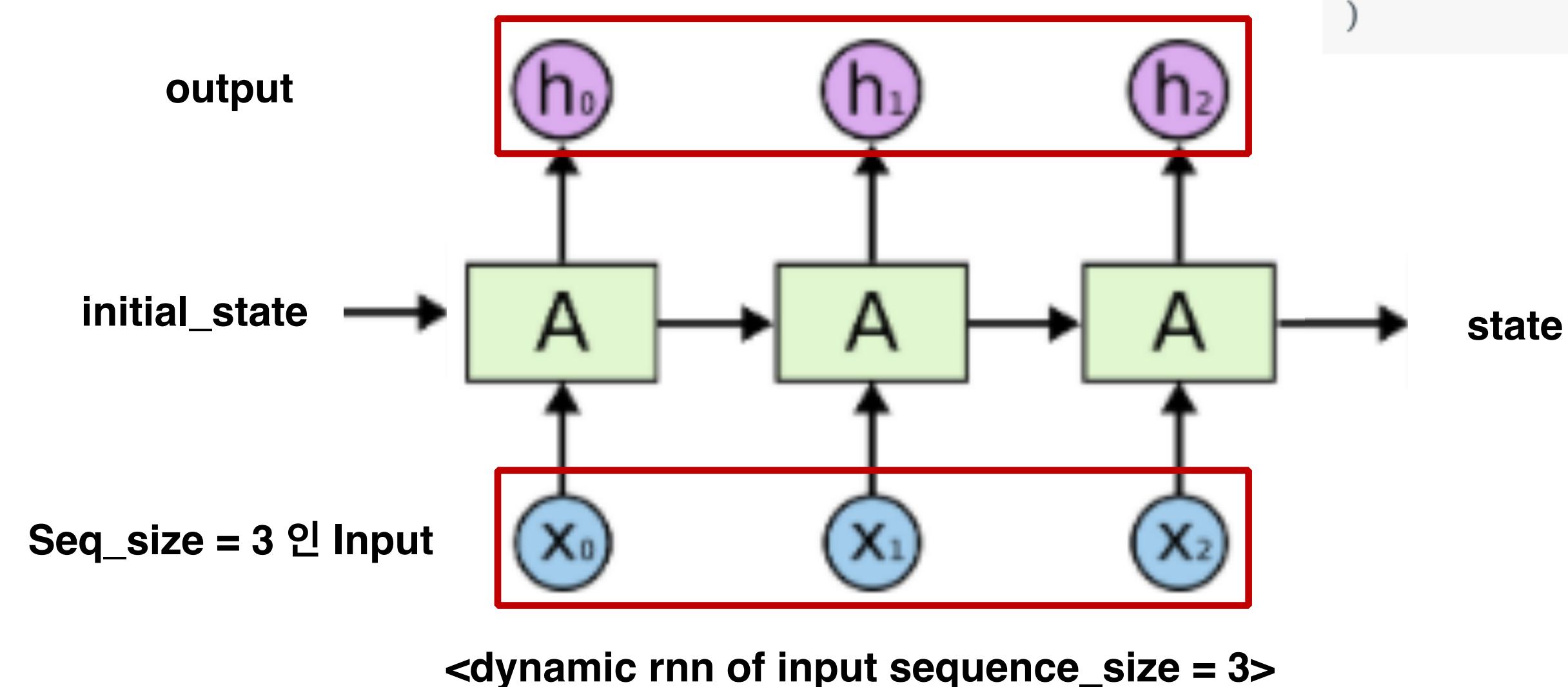
tf.nn.dynamic_rnn

- 입력받은 Cell 구조를 이용하여 여러 시간의 결과값을 연속적으로 출력할 수 있도록 구현해놓은 함수
- 여러 시간 단위의 입력값을 바탕으로 내부에서 cell을 호출(call)하여 연속적으로 결과를 계산
- Arguments

- cell : 기본 단위 구조인 Cell (BasicRNN or LSTM or GRU etc..)
- inputs : Sequence size를 갖는 입력값
- initial_state : 초기 결과값 계산에 활용할 이전 State의 계산값
(None으로 설정시 default 값으로 0을 이용)

tf.nn.dynamic_rnn

```
dynamic_rnn(  
    cell,  
    inputs,  
    sequence_length=None,  
    initial_state=None,  
    dtype=None,  
    parallel_iterations=None,  
    swap_memory=False,  
    time_major=False,  
    scope=None  
)
```



tf.nn.dynamic_rnn

Arguments

- cell : 기본 단위 구조인 Cell (BasicRNN or LSTM or GRU etc..)

- inputs : Sequence size를 갖는 입력값

- initial_state : 초기 결과값 계산에 활용한 이전 State의 계산값

(None으로 설정시 default 값으로 0을 이용)

- parallel_iterations: 계산을 parallel하게 run하기 위해 사용되는 반복수 (32)

- swap_memory: Swap memory 사용 여부

- time_major

□ True: [Max Time, Batch Size, Depth] (해당 구조가 계산상 더 빠름)

□ False: [Batch Size, Max Time, Depth]

- scope: RNN을 지정할 Variable Scope

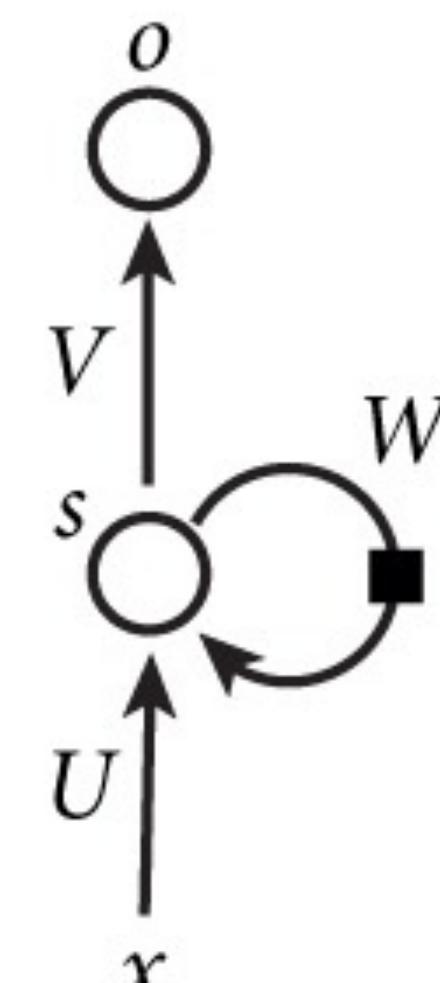
tf.nn.dynamic_rnn

```
dynamic_rnn(  
    cell,  
    inputs,  
    sequence_length=None,  
    initial_state=None,  
    dtype=None,  
    parallel_iterations=None,  
    swap_memory=False,  
    time_major=False,  
    scope=None  
)
```

- Returns: [output, (final) state]
- 기본적으로 모든 Cell들은 output과 state를 같은 값으로 출력함
- 각 Time step의 hidden state가 (output, state) 형식으로 두번 출력됨
- 아래 Output을 계산하기 위해서 별도로 V 변수를 선언한 후, 추가적인 계산 필요

<Call in BasicRNNCell>

```
def call(self, inputs, state):
    """Most basic RNN: output = new_state = act(W * input + U * state + B)."""
    output = self._activation(_linear([inputs, state], self._num_units, True))
    return output, output
```



$$s_t = \sigma(Ux_t + Ws_{t-1})$$

$$o_t = \varphi(Vs_t)$$

<Return Shape>

- Output Shape

time_major == True:

[max_time, batch_size, num_unit]

time_major == False (default):

[batch_size, max_time, num_unit]

- (Final) State Shape

[batch_size, num_unit]

Example of Dynamic RNN

- BasicRNN Cell을 만들고, dynamic rnn에서 신경망을 구축한 다음 outputs과 states를 받는다.

```
cell = rnn.BasicRNNCell(self.hidden_dim, reuse=tf.get_variable_scope().reuse)
outputs, states = tf.nn.dynamic_rnn(cell, self.x, dtype=tf.float32)
```

```
cell = rnn.BasicLSTMCell(self.hidden_dim, reuse=tf.get_variable_scope().reuse)
outputs, states = tf.nn.dynamic_rnn(cell, self.x, dtype=tf.float32)
```

```
cell = rnn.GRUCell(self.hidden_dim, reuse=tf.get_variable_scope().reuse)
outputs, states = tf.nn.dynamic_rnn(cell, self.x, dtype=tf.float32)
```

tf.contrib.rnn.OutputProjectionWrapper

- BasicRNNCell의 리턴값인 output과 state을 동일하게 사용하지 않고, 추가적으로 output 계산을 진행하고 싶을 경우 tf.contrib.rnn.OutputProjectionWrapper 를 통해 구현 가능
- 일반적인 경우는 사용이 흔치 않음
(일반적으로 리턴된 output들을 바탕으로 full-connected layer 구축)
- DynamicRNN을 사용하기전, OutputProjectionWrapper를 통해 BasicRNNCell 추가 계산

tf.contrib.rnn.OutputProjectionWrapper

`__init__`

```
__init__(  
    cell,  
    output_size,  
    activation=None,  
    reuse=None  
)  
..._cell = tf.contrib.rnn.BasicRNNCell(self.hidden_dim, reuse=get_variable_scope.reuse())  
cell = tf.contrib.rnn.OutputProjectionWrapper(_cell, self.output_dim, reuse=get_variable_scope.reuse())  
...output, state = tf.nn.dynamic_rnn(cell, self.x_inputs, dtype=tf.float32)
```

tf.contrib.rnn.LSTMCell

□ 기본 모델의 단위인 **LSTMCell**을 생성하기 위한 클래스

□ 기존 **BasicLSTMCell**에 비해 다양한 설정값을 제공

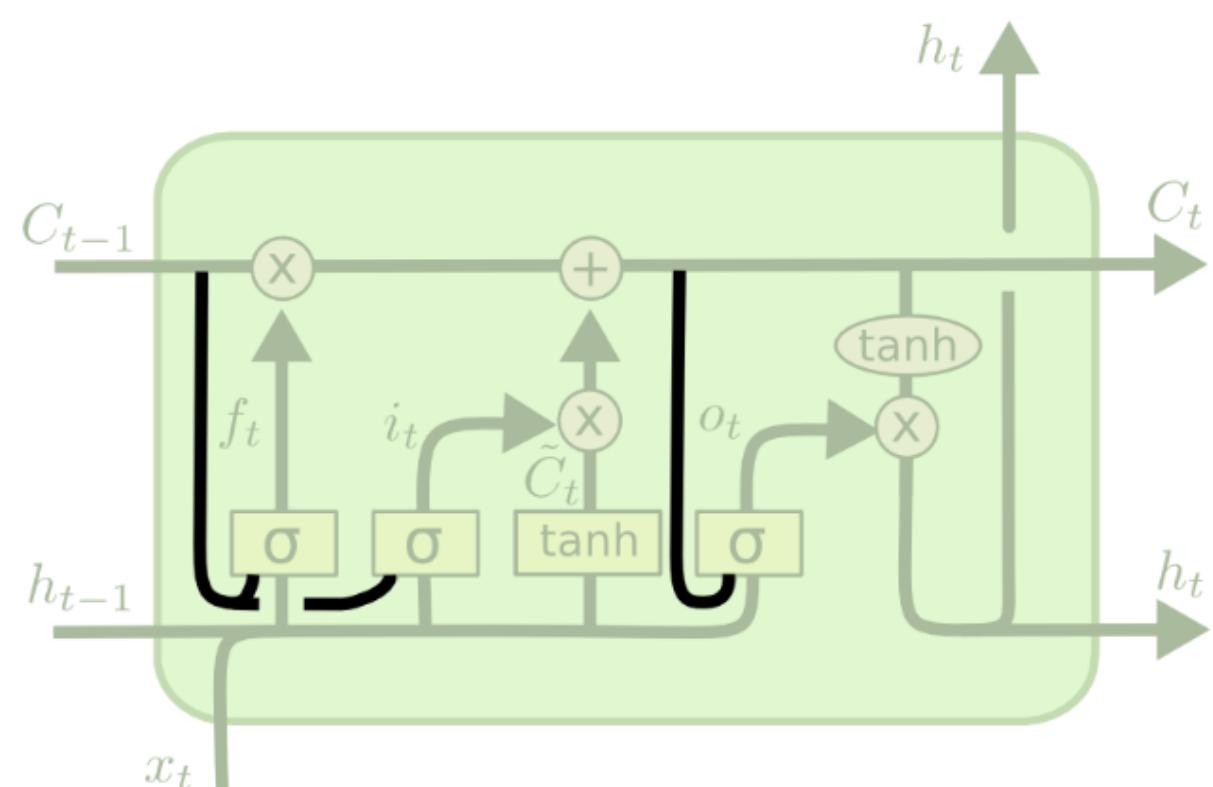
□ __init__

- `use_peepholes`: LSTM 내 Peephole connection 사용 여부 설정
- `cell_clip`: activation 절댓값의 최대 범위를 지정 (optional)
- `num_proj`: output을 projection할 경우 차원 결정 (optional, `OutputProjectionWrapper`와 유사)
- `proj_clip`: (projection 사용할 경우) projection 절댓값의 최대 범위 지정 (optional)

tf.contrib.rnn.LSTMCell

__init__

```
__init__(  
    num_units,  
    use_peepholes=False,  
    cell_clip=None,  
    initializer=None,  
    num_proj=None,  
    proj_clip=None,  
    num_unit_shards=None,  
    num_proj_shards=None,  
    forget_bias=1.0,  
    state_is_tuple=True,  
    activation=None,  
    reuse=None  
)
```

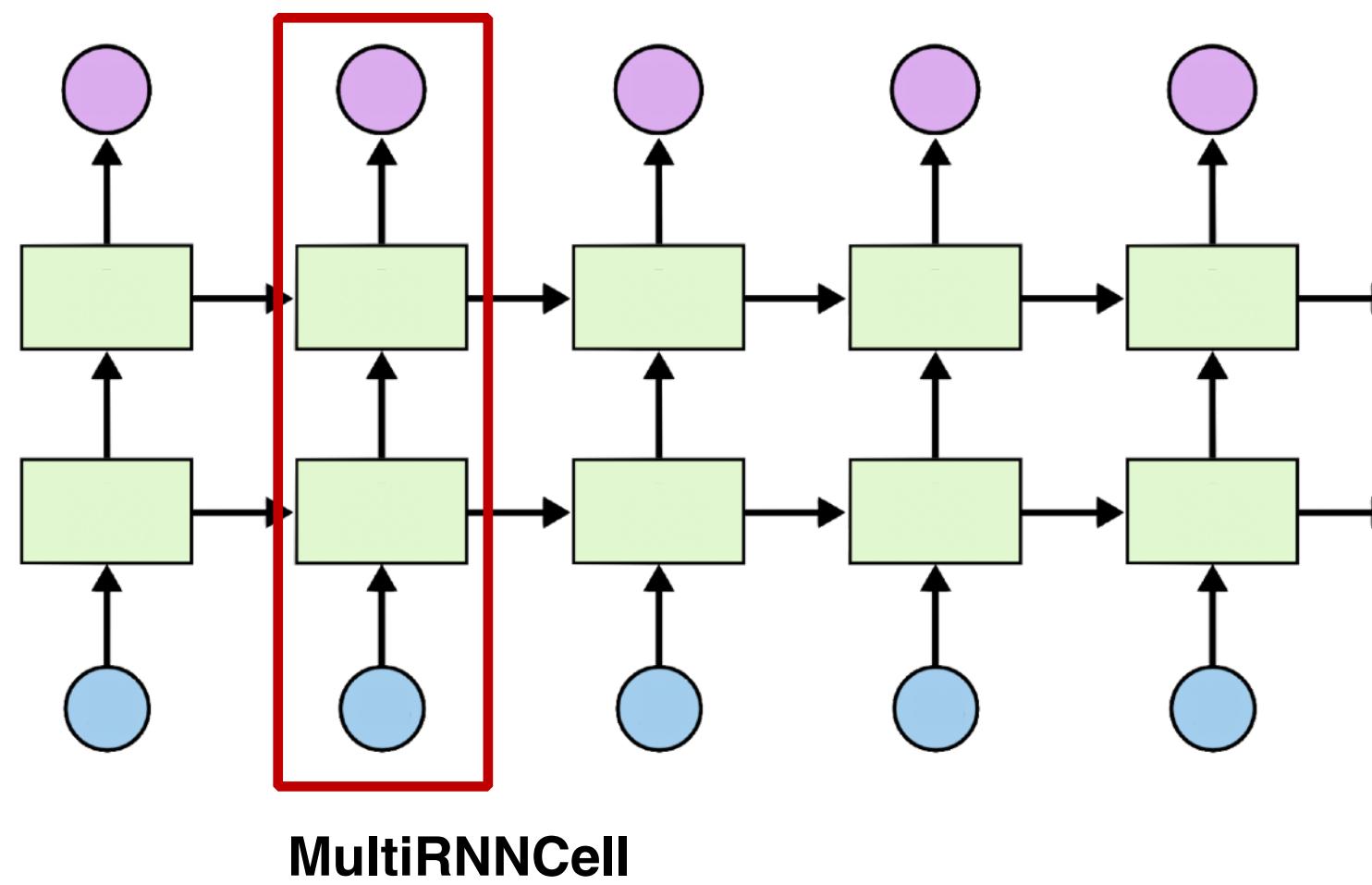


$$\begin{aligned}f_t &= \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\i_t &= \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\o_t &= \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o)\end{aligned}$$

<LSTM with Peephole Connection>

tf.contrib.rnn.MultiRNNCell

- 여러 층의 recurrent layer를 사용하는 Stacked RNN을 구현하고 싶은 경우, MultiRNNCell을 이용하여 Cell 생성
- 쌓고자 하는 순서로 Cell들을 list 형태로 선언한 후, MultiRNNCell의 입력값으로 사용



tf.contrib.rnn.MultiRNNCell
__init__(cells, state_is_tuple=True)

Create a RNN cell composed sequentially of a number of RNNCells.
- cells: 구성하고자 하는 순서대로 Cell을 담은 list
- state_is_tuple: state의 출력 형식

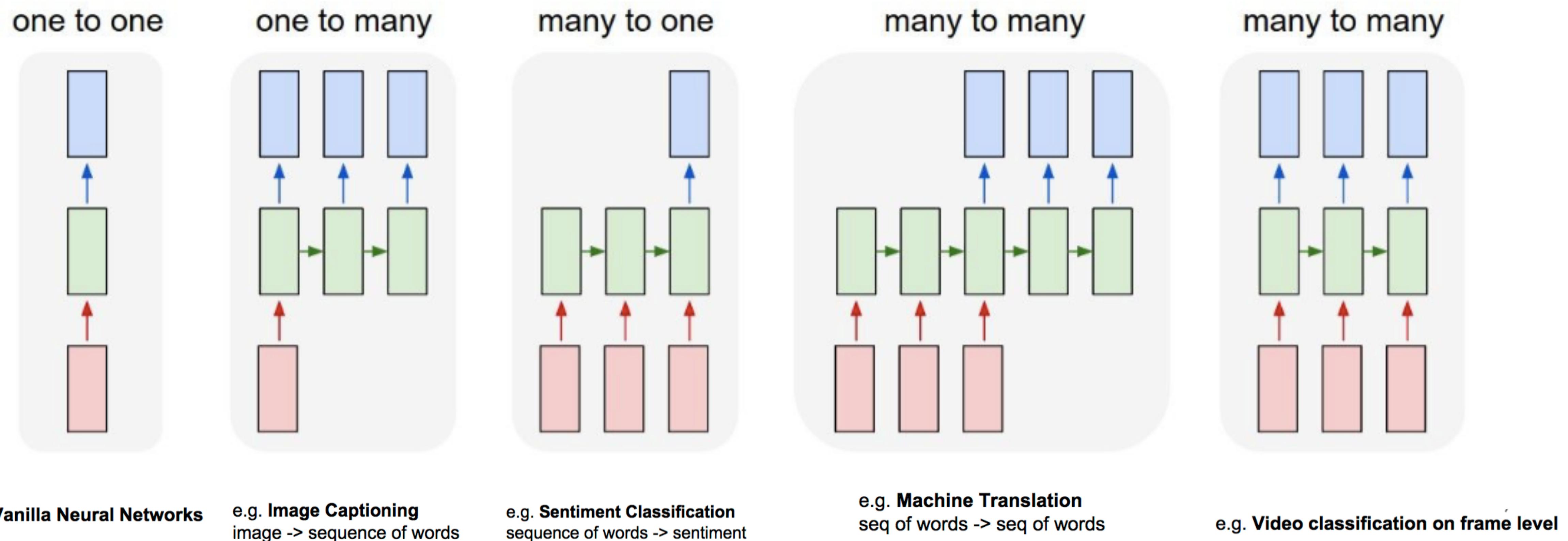
```
#1st recurrent layer
cell=tf.nn.rnn_cell.LSTMCell(num_units=self.hidden_dim, initializer = tf.contrib.layers.xavier_initializer(uniform=False), state_is_tuple=False)

#Stacked RNN Cells
stack_rnn =[cell] * self.num_layers

#stacked rnn cell
cells=tf.nn.rnn_cell.MultiRNNCell(stack_rnn,state_is_tuple=True)
```

Various Architectures of RNN

- 기타 다양한 구조의 Recurrent Model은 RNNCell 단위로 직접 호출하여 output, state를 계산함
- 반복문 (for 문)을 통하여 계산 결과를 연속적으로 계산함

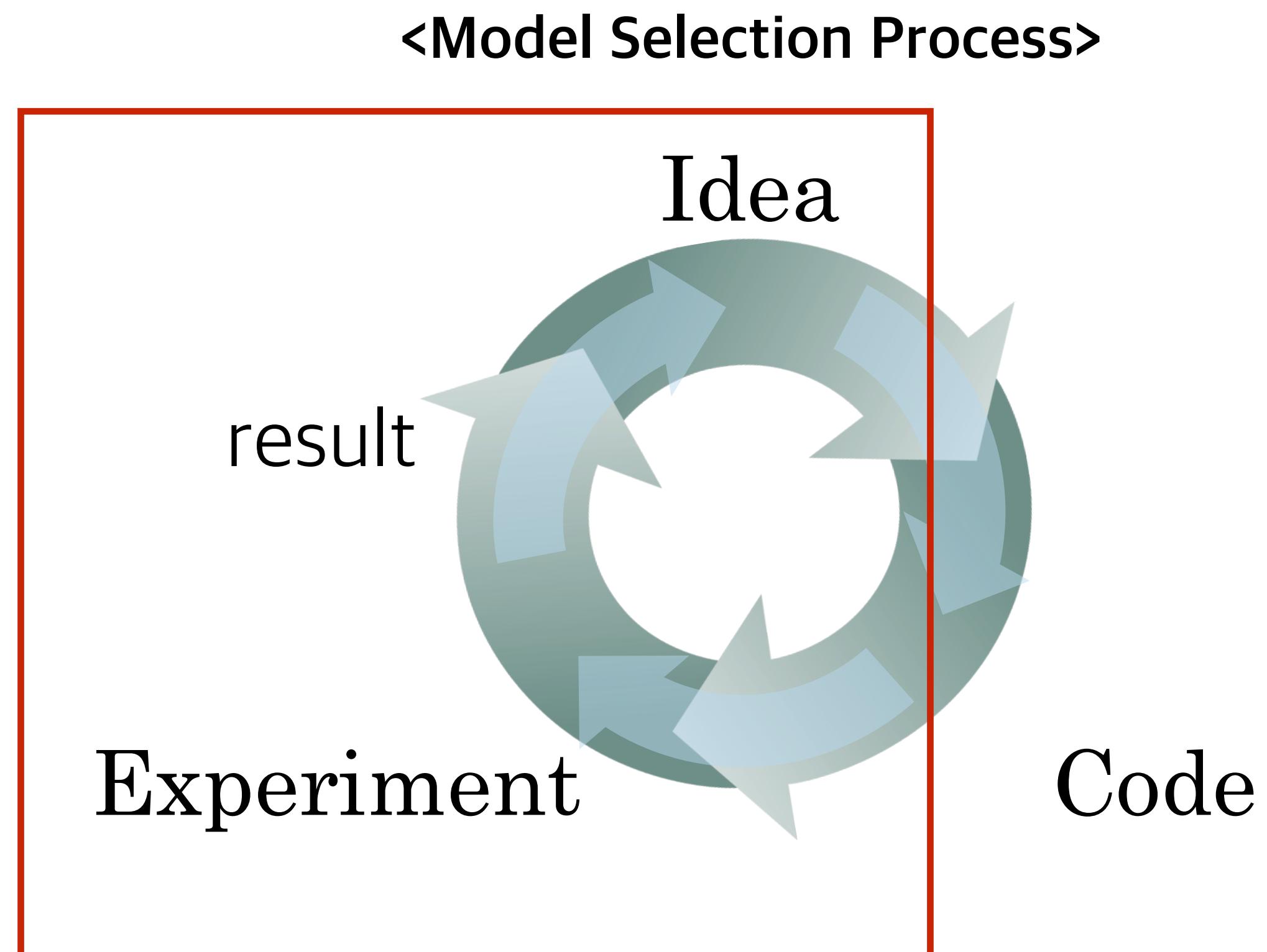


Recurrent Models Practice with TF

Model Selection

Introduction: Model Selection Process

- 모델 선택 프로세스는 크게 1) Idea Generation, 2) Code Implementation, 3) Experiment 순서로 이루어짐
- 정의된 문제의 해결을 위해 아이디어를 내고, 이를 코드로 실현하고 실험을 하며 나오는 결과를 바탕으로 모델을 수정함



How can we change model?

How can we change hyper-parameters?

Performance Measure

- 모델의 성능을 객관적으로 판단하기 위해, 이를 측정할 수 있는 성능 척도 (Performance Measure)가 필요함
 - 모델 학습에 사용하는 Cost Function과 별도로, 학습된 모델이 원하는 목적을 얼마나 잘 달성하는지를 판단하는 기준
 - 상황에 따라 다양한 성능 척도가 사용될 수 있음

Types of Cost Function

- (In supervised learning), 실제 측정값과 모델 출력값의 차이
 - Cost를 최소화하도록 지속적으로 모델을 학습함

<Least Square Method>

- 실제값과 모델 출력값 차이의 제곱들의 합
 - Linear Regression 모델에서 주로 사용

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

<Cross-Entropy>

- 오분류에 대한 확률적인 비용을 의미
 - Classifier (특히, softmax) 모델과 주로 사용

$$J(\theta) = -\frac{1}{m} \sum_j^m y^{(j)} \log h_{\theta}(x^{(j)})$$

Types of Performance Measure

- Regression의 경우 R-square, Classification의 경우 Accuracy 를 기본 성능 척도로 사용함

<R-square>

- 학습된 모델이 데이터를 설명하는 정도를 의미
 - 데이터의 전체 변동 중, 학습된 모델이 설명하는 정도를 0~1 사이로 계산

<Accuracy>

- 학습된 모델이 얼마나 정확하게 분류를 예측하는 지에 대한 척도
 - 전체 데이터 개수 중, 모델이 정확하게 분류한 데이터의 개수의 비율

Etc.

- ROC AUC, F-score, False Alarm Rate, Precision, Recall, True Positive Rate....

Coefficient of Determination (결정 계수)

- 결정 계수 (Coefficient of Determination)는 R-square로 표기하며, Regression 모델의 성능을 측정하는 지표 중 하나
- 학습된 모델이 예측하고자 하는 데이터의 변동을 얼마나 잘 설명하는지에 대한 정도를 나타냄

<R-square Measure>

$$R^2 = \frac{SSR}{SST} = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$
$$= 1 - \frac{SSE}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$



*데이터의 변동 ~ 정보의 양으로 바라볼 때 !

$$\frac{\text{우리 모델이 설명하는 변동}}{\text{데이터의 전체 변동}} = \frac{(\text{모델 예측값} - \text{전체 평균}) \text{ 제곱 합}}{(\text{데이터 실제 값} - \text{전체 평균}) \text{ 제곱 합}}$$
$$= 1 - \frac{\text{우리 모델이 가지는 에러 변동}}{\text{데이터의 전체 변동}} = 1 - \frac{(\text{데이터 실제 값} - \text{모델 예측값}) \text{ 제곱 합}}{(\text{데이터 실제 값} - \text{전체 평균}) \text{ 제곱 합}}$$

Accuracy (정확도)

- 분류 모델의 대표적인 성능 척도로 정확도 (Accuracy)를 사용함
- 전체 테스트 데이터 중 학습된 모델이 정확하게 분류한 데이터의 비율을 의미함

$$\text{Accuracy} = \frac{\# \text{ of } \text{ accurate } \text{ results}}{\# \text{ of } \text{ test } \text{ data}}$$

분류 모델의 성능을
판단할 때
항상 적합할까?

Example)
의료 이미지에서
악성 종양 유무를 분류한다면?

Accuracy를 성능 척도로 사용할 때의
잠재적인 가정은……

분류하고자 하는 데이터의
클래스들이 균형적이라는 것!

그렇지 않다면…?

Confusion Matrix

- 분류 모델의 성능을 상황에 따라 복합적으로 분석하기 위해 Confusion Matrix를 활용함
- 실제 데이터의 클래스와 모델이 예측한 결과에 따라 구분하여 성능을 기록한 행렬을 의미함
- 각각의 의미에 따라 True/False + Positive/Negative로 구분할 수 있음

<Confusion Matrix>

	Actual Negative	Actual Positive
Model Negative	True Negative (TN)	False Negative (FN)
Model Positive	False Positive (FP)	True Positive (TP)

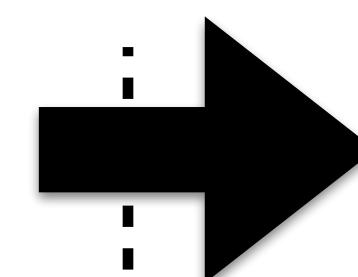
True/False + Negative/Positive

(모델이)
맞혔다/틀렸다

(모델의 대답은)
Neg/Pos 클래스였다

예시

- True Positive: 모델이 Positive라고 해서 맞추었다.
- False Positive: 모델이 Positive라고 해서 틀렸다.
(실제로는 Negative)



$$\text{Accuracy} = (\text{TN}+\text{TP})/(\text{TN}+\text{TP}+\text{FN}+\text{FP})$$

$$\begin{aligned}\text{True Positive Rate (Sensitivity)} \\ = (\text{TP})/(\text{TP}+\text{FN})\end{aligned}$$

$$\begin{aligned}\text{True Negative Rate (Specificity)} \\ = (\text{TN})/(\text{TN}+\text{FN})\end{aligned}$$

$$\begin{aligned}\text{Precision} \\ = (\text{TP})/(\text{TP}+\text{FP})\end{aligned}$$

Module: tf.metrics

Defined in [tensorflow/python/ops/metrics.py](#).

Evaluation-related metrics.

Functions

[accuracy\(...\)](#): Calculates how often predictions matches labels.

[auc\(...\)](#): Computes the approximate AUC via a Riemann sum.

[false_negatives\(...\)](#): Computes the total number of false negatives.

$$\begin{aligned}\text{False Positive Rate} \\ = (\text{FP})/(\text{FP}+\text{TN})\end{aligned}$$

Training/Validation Dataset

- 적합한 모델을 선정하기 위해 일반적으로 Training/Validation 데이터셋을 이용할 수 있음
- Train/Valid 데이터의 최종 성능을 확인하고, 이를 바탕으로 상황에 따라 적절하게 모델을 수정하는 방향으로 실험을 반복
- 데이터 구분 비율은 기본적으로 7:3, 5:3:2 등을 주로 사용하였으나, 데이터가 많아질수록 Training 데이터 비율을 늘림

<Data Split with No Validation >



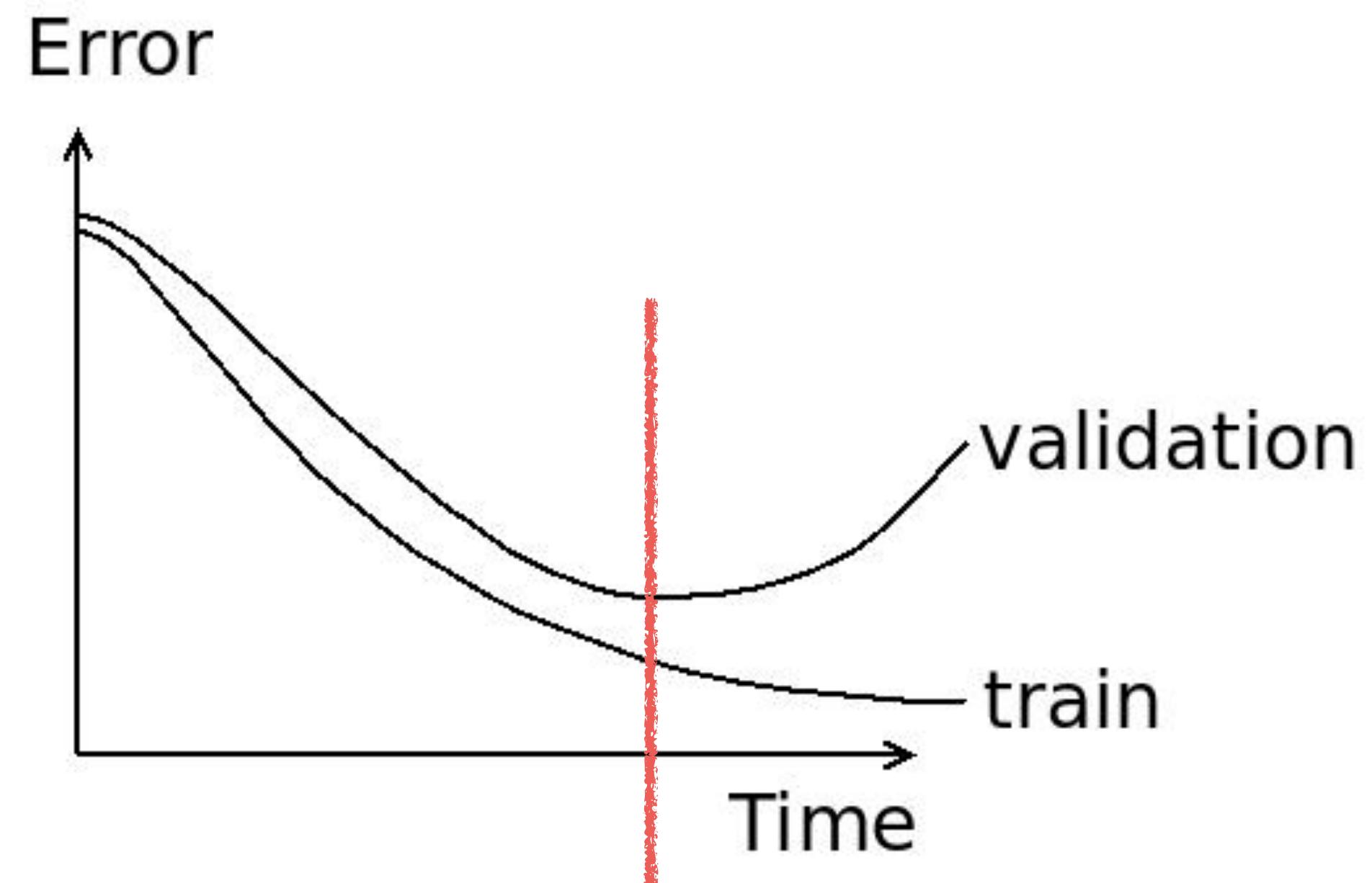
→ Training Data에 과적합된 학습 가능성 높음

<Data Split with Validation>



↓
학습 반복마다 일반화(Generalization)에 대한 검증

<Learning Curve>

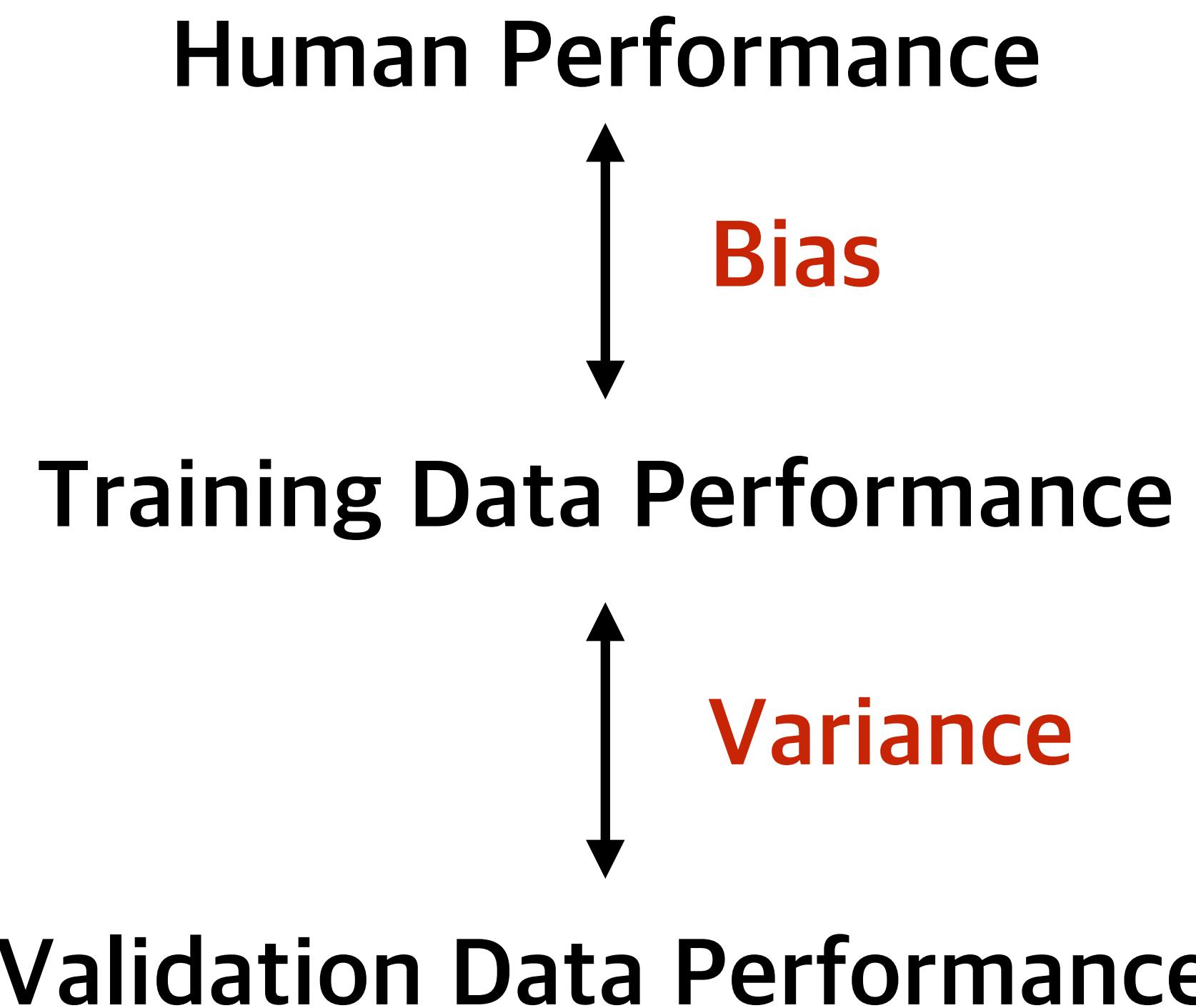


*Test 데이터는 학습 완료 후 최종 성능을 확인하기 위해서만 사용됨

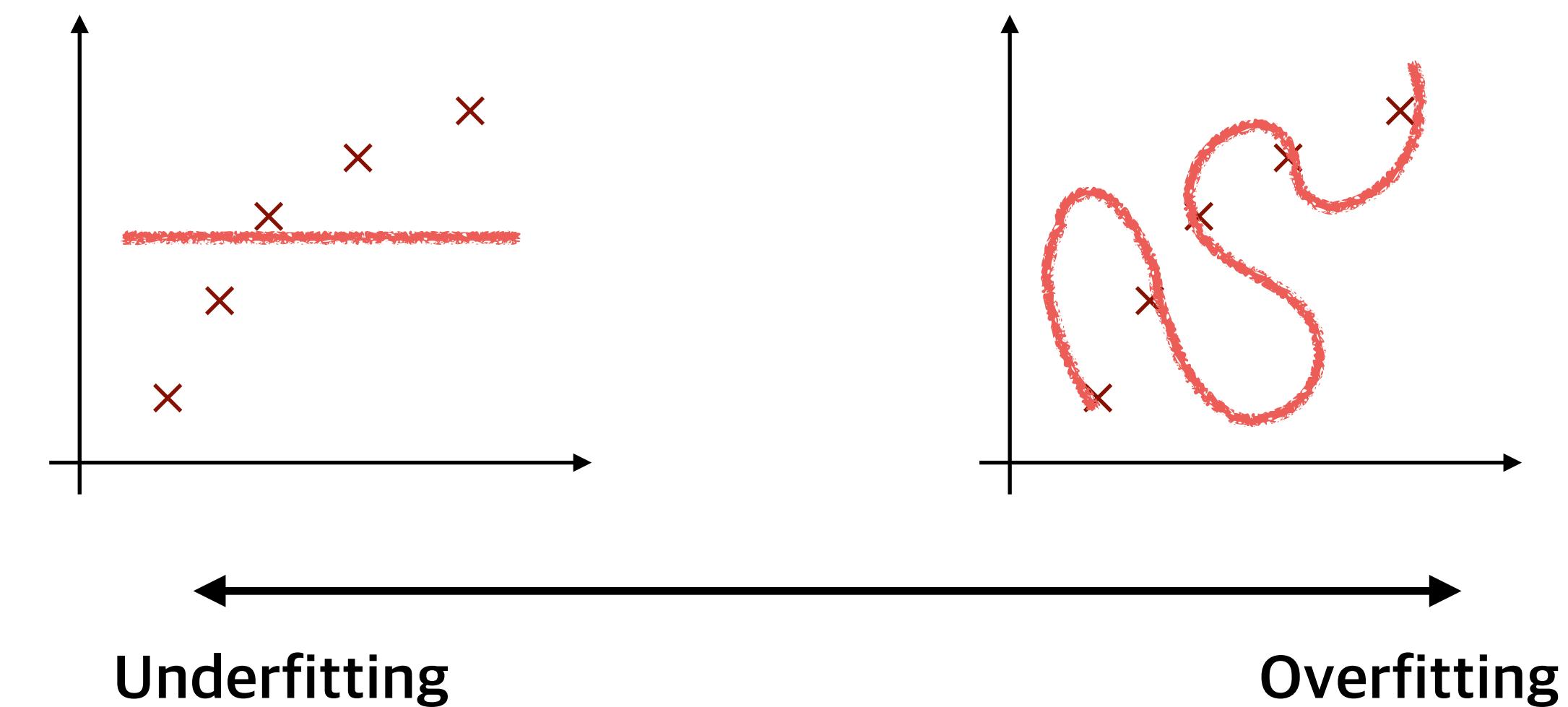
Bias and Variance Concept

❑ Bias & Variance

- Bias: 우리의 모델이 원하는 성능에 비해 얼마나 편향되어 있나? (안좋은 방향으로)
- Variance: 우리의 모델이 얼마나 과적합 되어 있나?

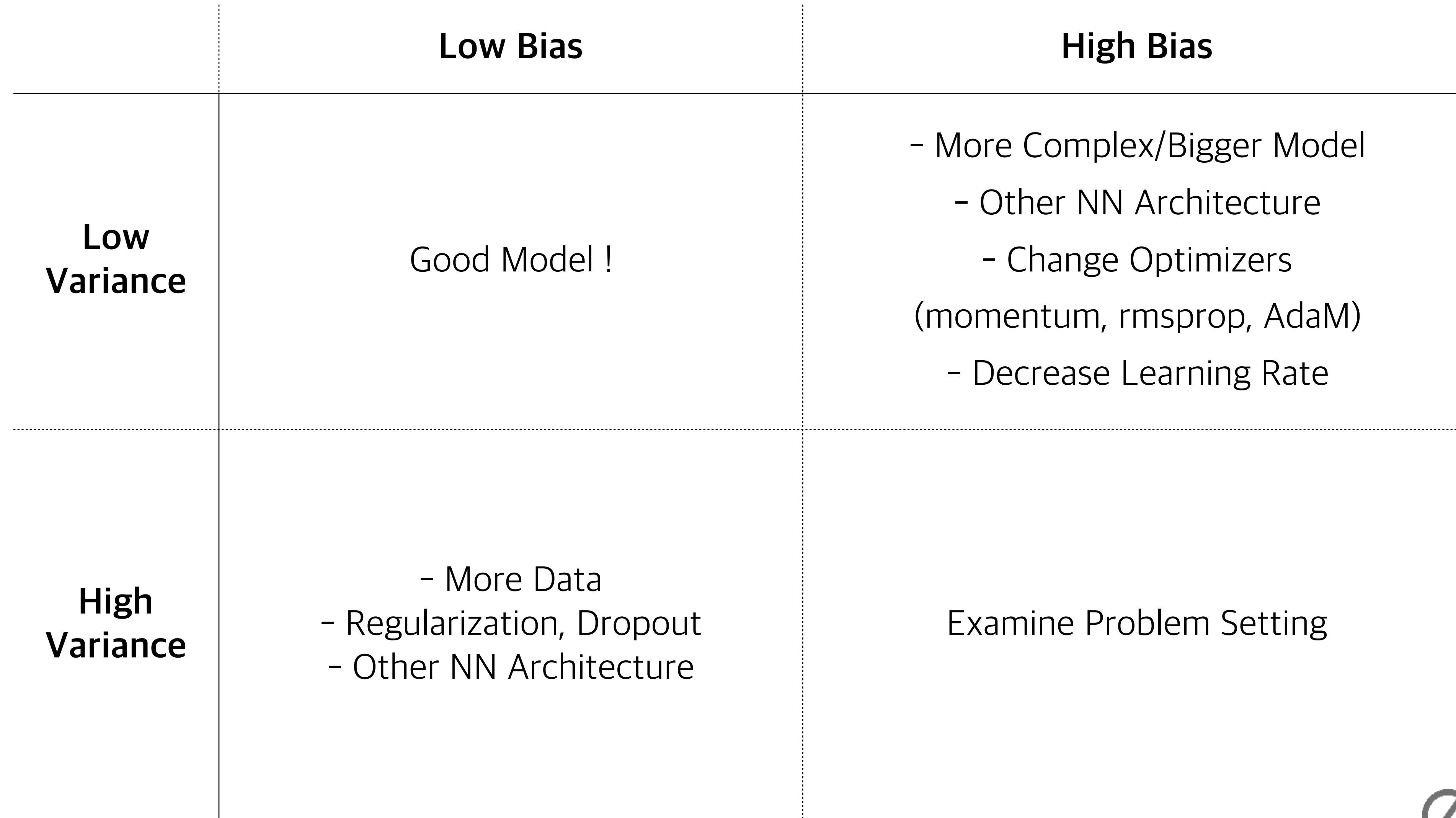


<Overfitting Example>



How to select model?

- ❑ Bias, Variance 정도에 따라 실행할 수 있는 행동은 아래와 같음



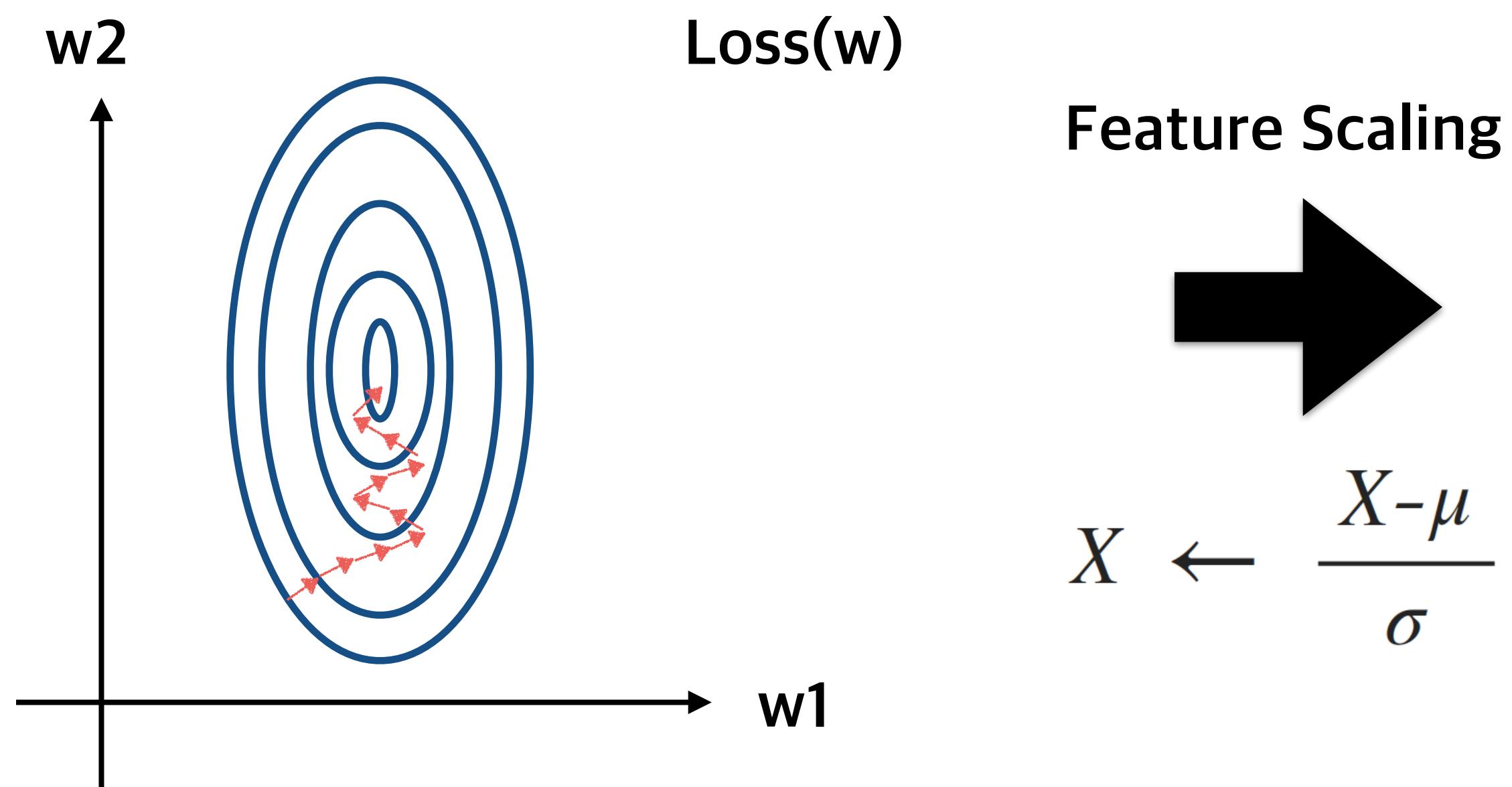
Learning Methods in Deep Learning

Learning Much Faster

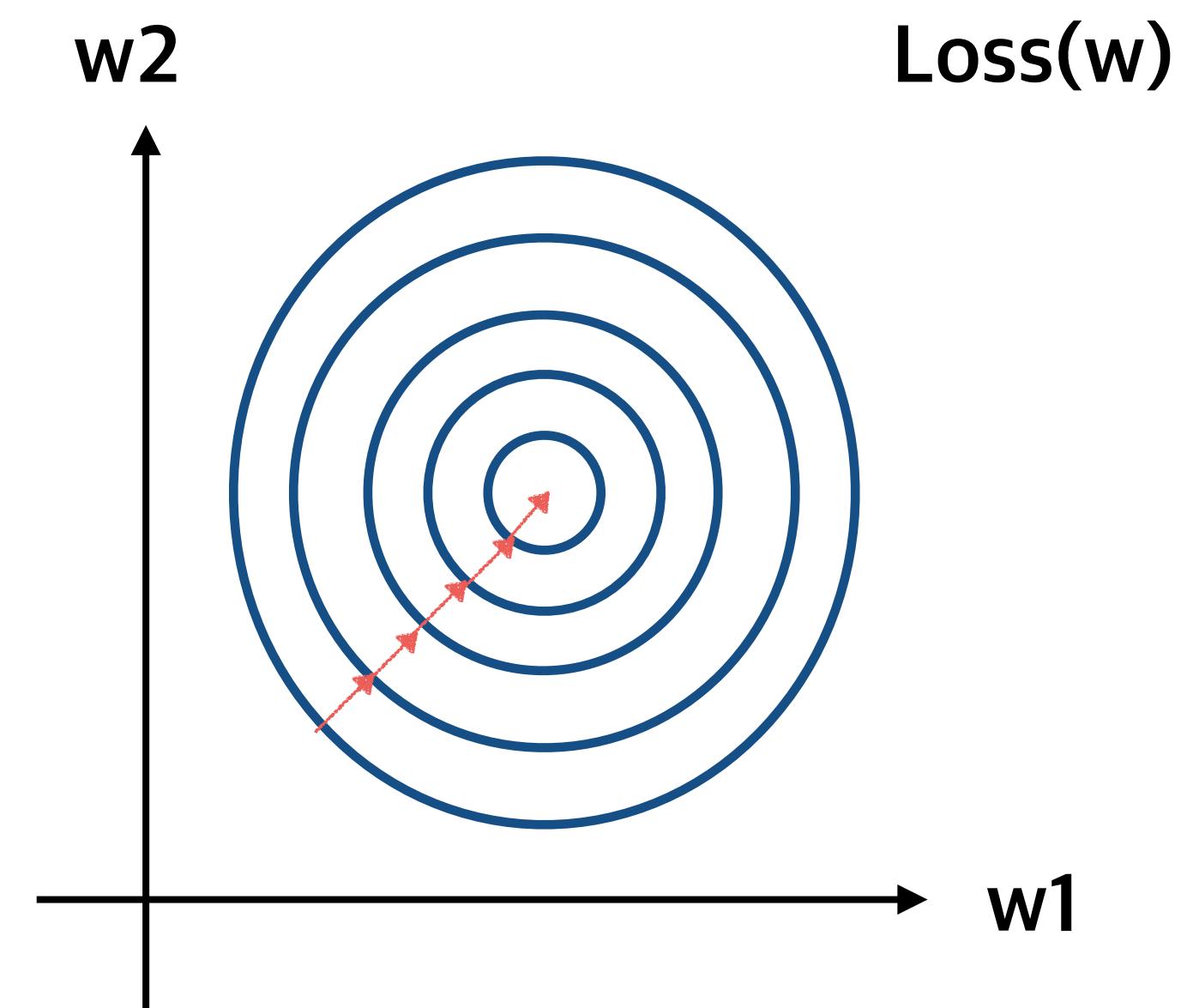
Feature Scaling

- 입력 데이터 X 의 특징(Feature)을 표현하는 값의 범위와 스케일(scale)이 다르면, 실질적으로 학습이 오래 걸리는 문제 발생
- 각 Feature 별 평균(mean)과 편차(standard deviation or range)를 가지고 정규화(normalization)하는 것이 일반적
 - 모든 Feature의 평균을 0, 편차를 1로 맞추어 주는 것을 의미

<Learning with Different Scale of Features>



<Learning with Similar Scale of Features>



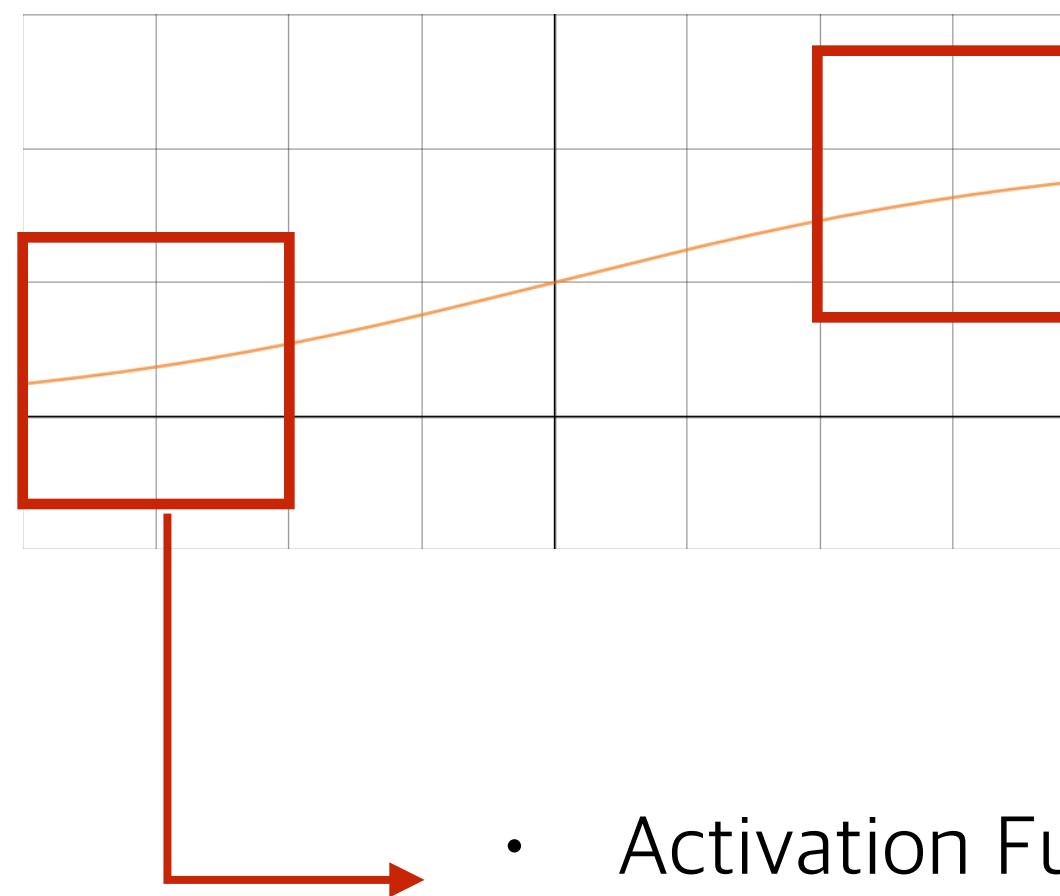
Feature Scaling

- Feature scaling은 sigmoid 또는 tanh 함수를 activation으로 사용할 때 필수적임 (Saturation of activation function)
- ReLU를 쓰는 경우에도 activation 값이 너무 커지는 현상을 방지하기 때문에 실용적으로 유용하다고 알려져 있음

<Sigmoid Function>

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

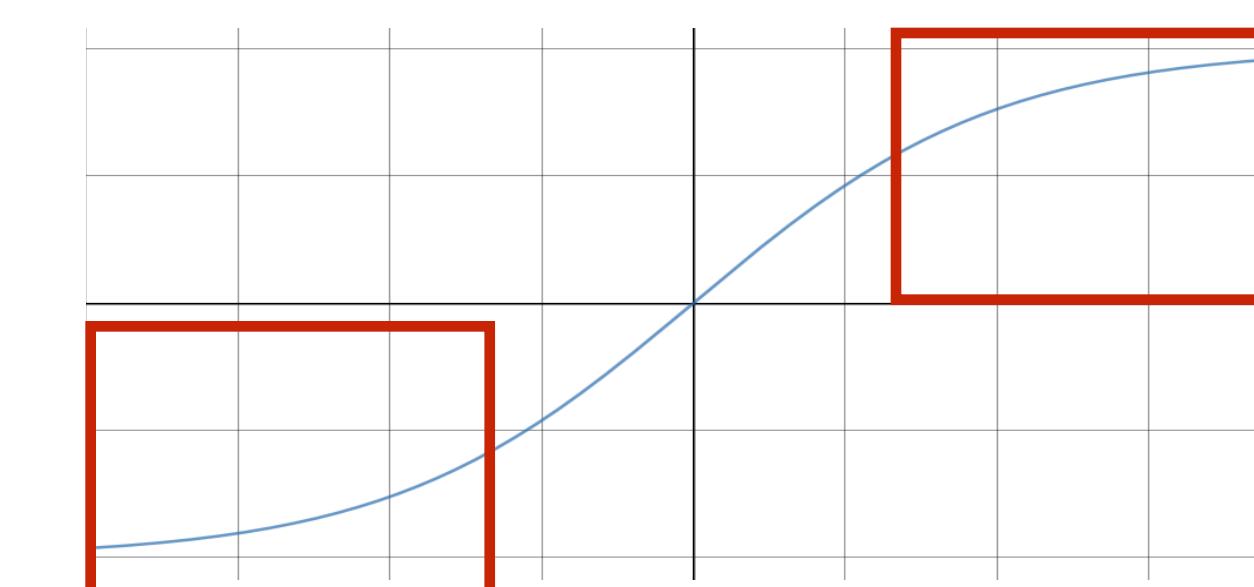
$$f'(x) = f(x)(1-f(x))$$



<tanh Function>

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

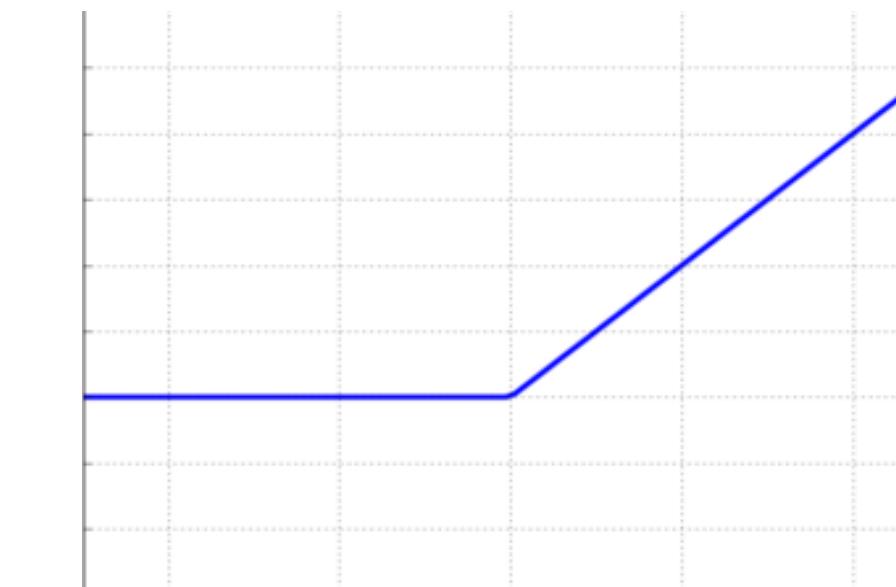
$$f'(x) = 1 - f(x)^2$$



<ReLU Function>

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$



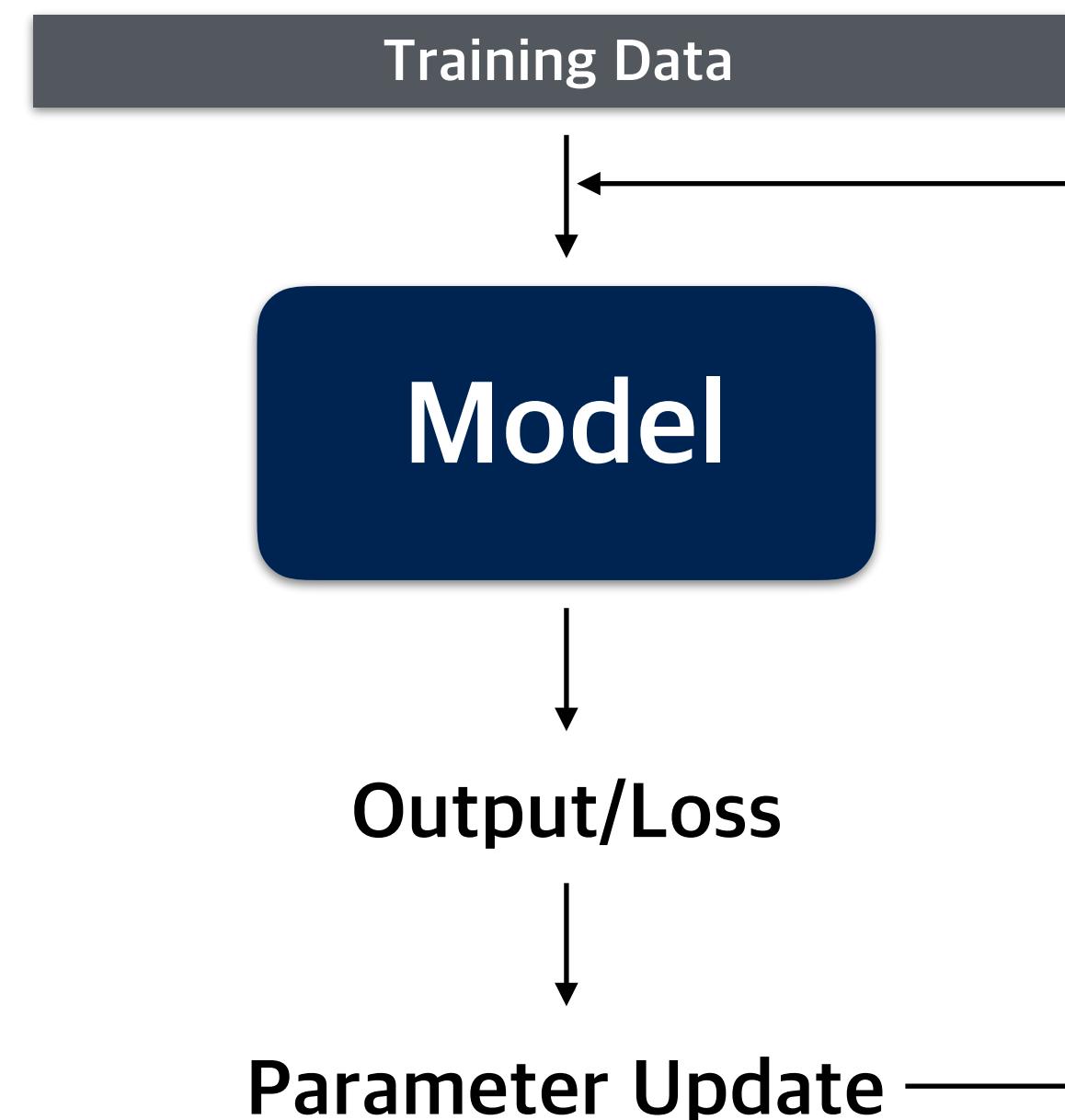
- Activation Function의 포화(Saturation) 부분
- 큰 값의 Scale을 가지는 Feature가 존재할 경우, 계산 결과가 이 부분에 해당할 가능성 높음
- Gradient 값이 0에 가까워, 학습이 거의 진행되지 않음

Mini-batch Learning for Effective Learning

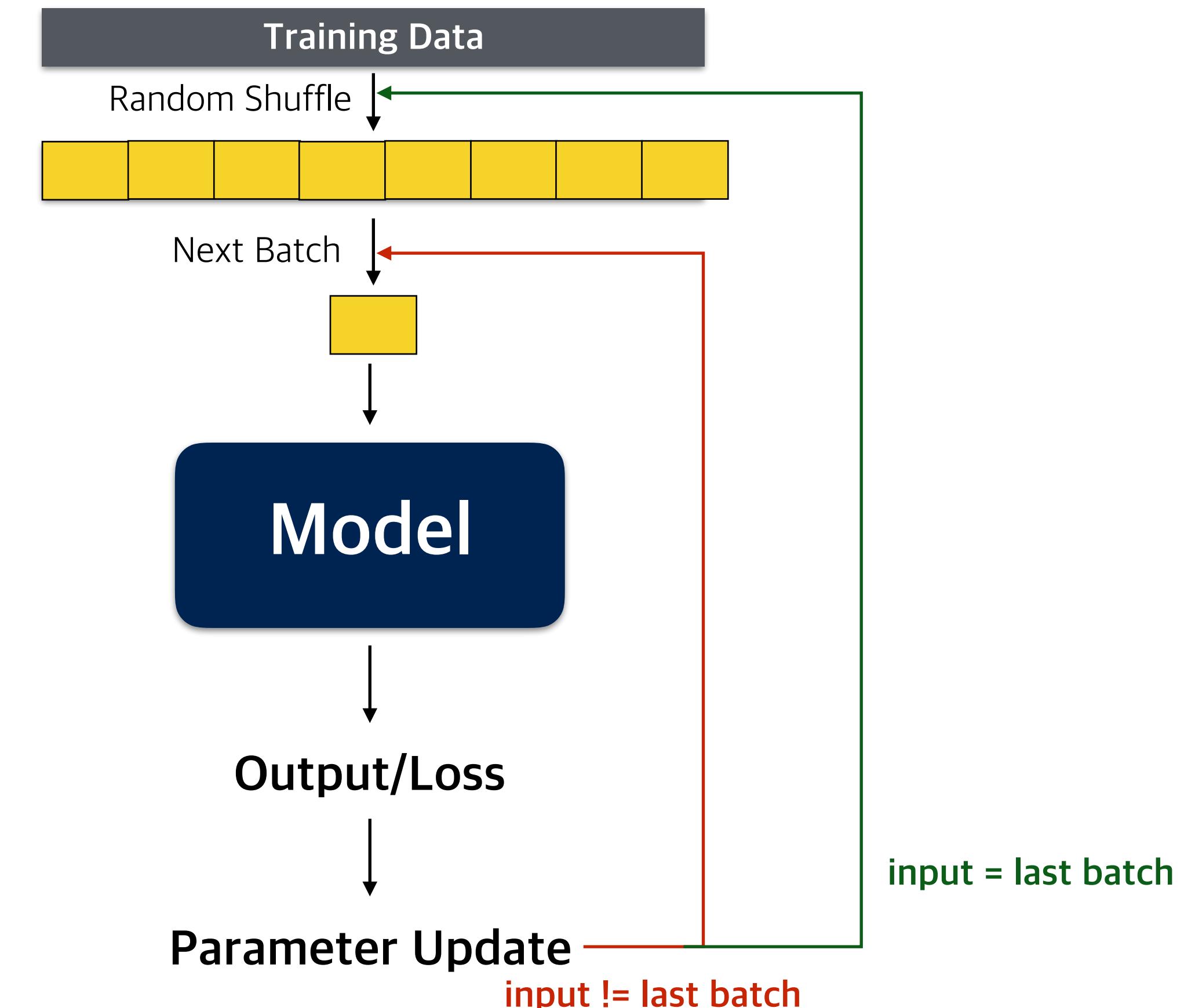
❑ Mini Batch Learning을 통해 학습의 속도를 높이고 local optimum에 빠지는 문제를 어느정도 해결 가능

- 작은 단위로 입력을 사용하기 때문에 1회 파라미터 업데이트의 속도가 빠름
- 고정된 트레이닝 데이터 양을 가지고 많은 수의 업데이트가 가능

<Full Batch Learning 예시>



<mini-Batch Learning 예시>

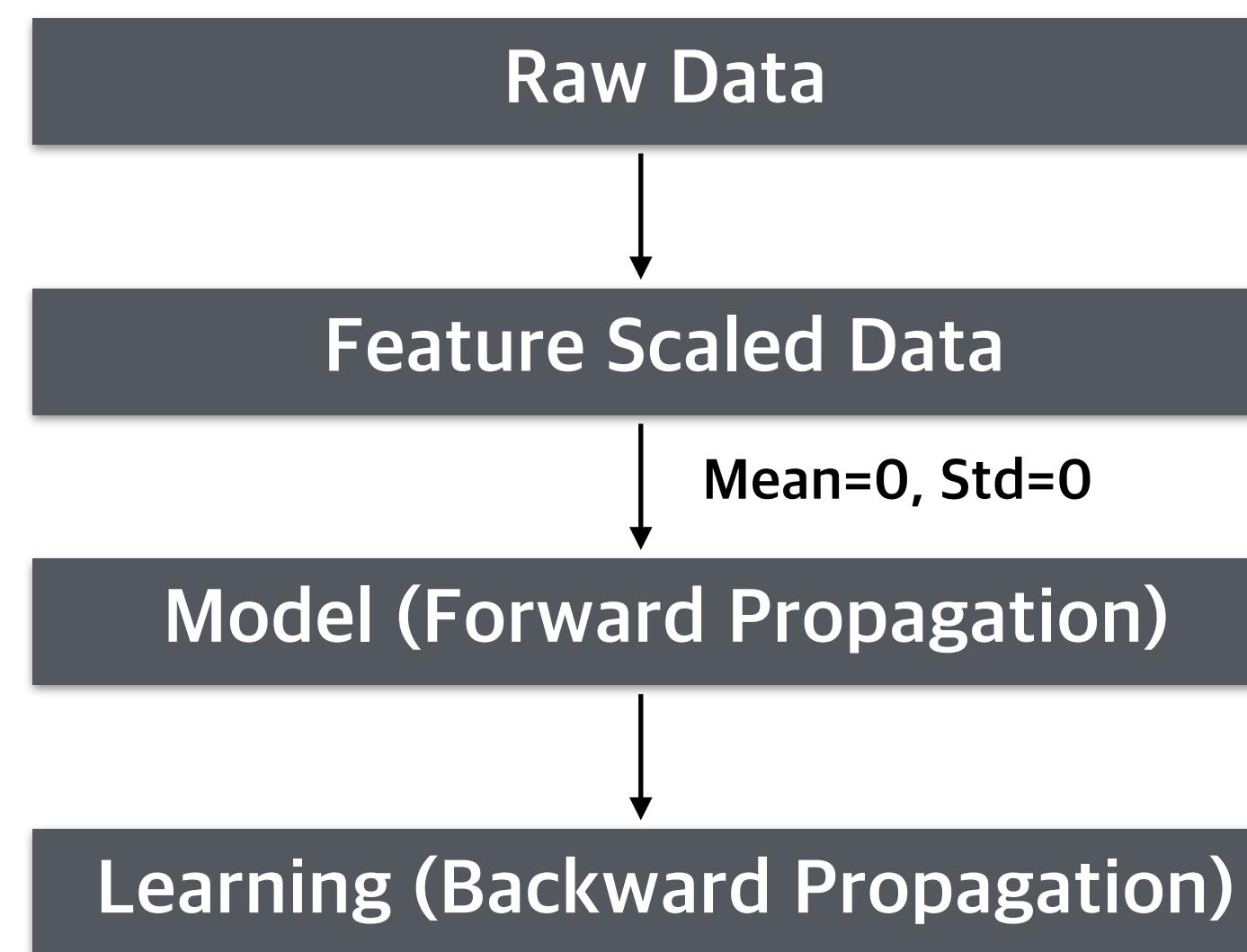


Mini-batch 방식으로 학습을 진행할 때,
앞에서 해결하고자 했던 Feature Scale 관련 문제가 다시 발생……
(Why?)

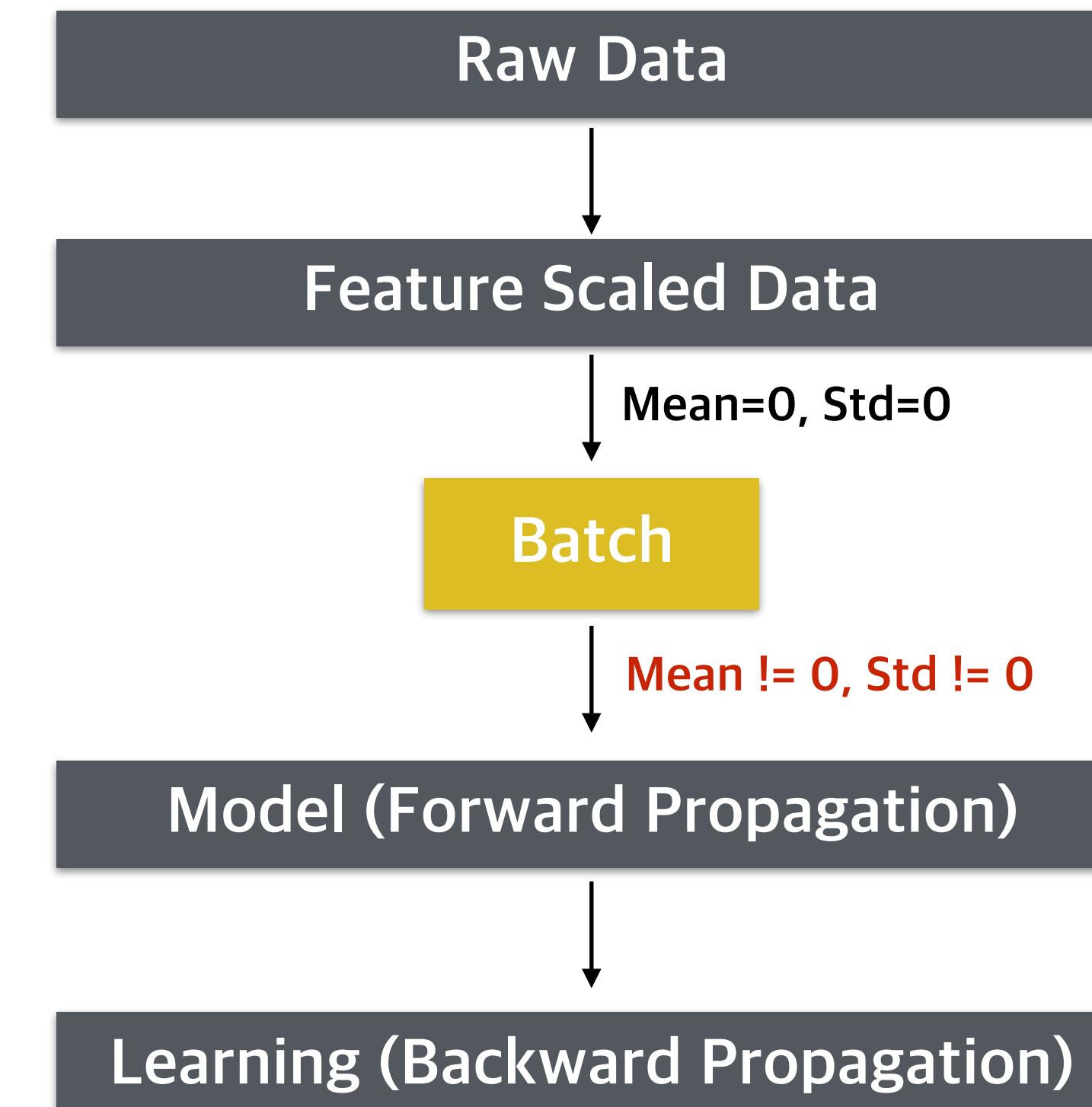
Mini-Batch Learning and Normalization

- Mini-batch 방식으로 학습을 진행할 때, 각 배치 단위로 파라미터 업데이트를 진행
- Feature Scaling 후, mini-batch를 사용하면 샘플링의 편향으로 인해 각 배치의 평균과 편차가 달라지는 현상이 발생할 수 있음

<Full Batch Learning Process>

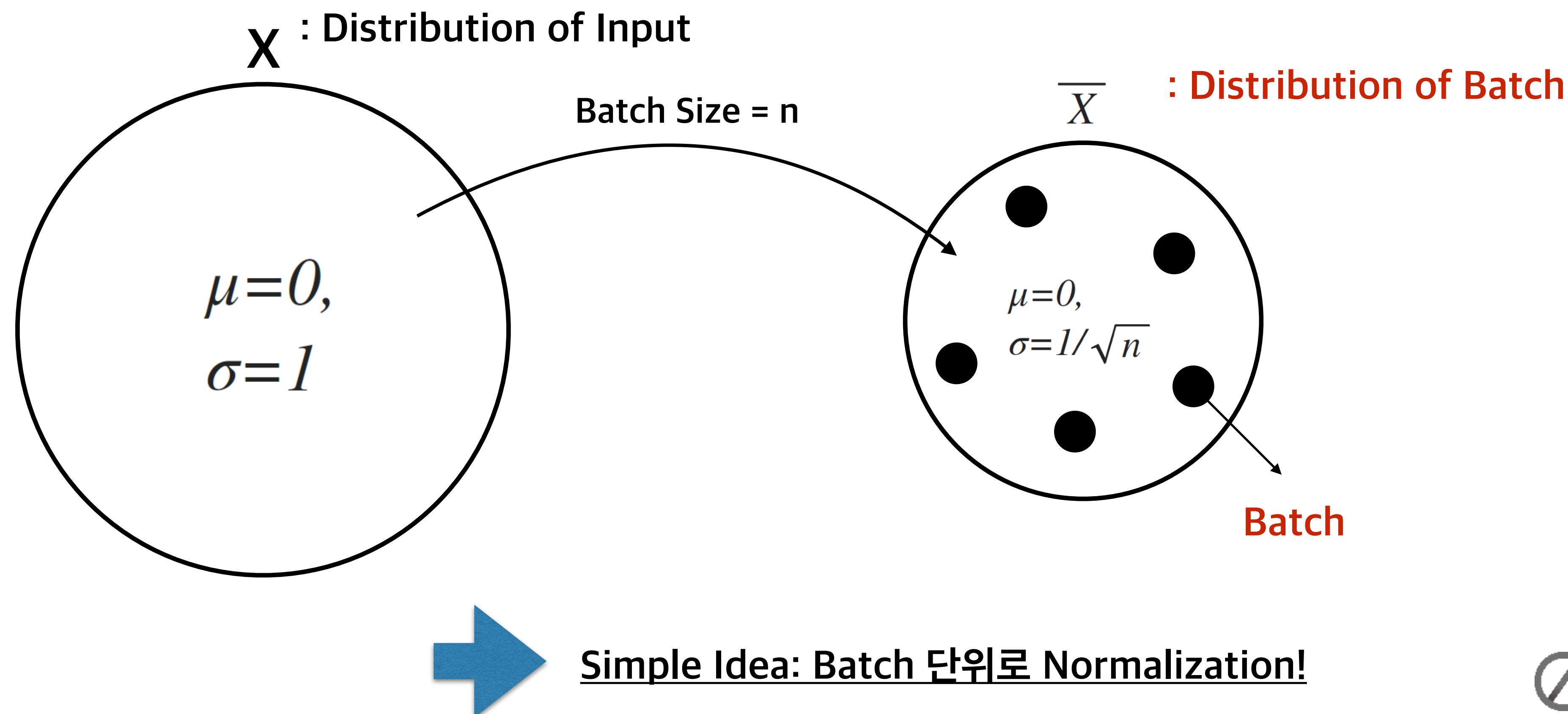


<Mini-batch Learning Process>



Mini-Batch Learning and Normalization

- 평균 0, 분산 1 모집단에서 n 개 단위로 샘플링을 진행한다고 가정
- 표본평균 집단의 분포는 평균 0, 분산 $1/n$ 을 갖지만, 샘플링된 각각의 표본들은 다양한 값을 가짐
- Batch의 분포는 평균 0, 분산 $1/n$ 을 따르지만, **각각의 Batch는 다양한 평균과 분산을 가짐 (심지어 분포의 형태도 바뀜)**

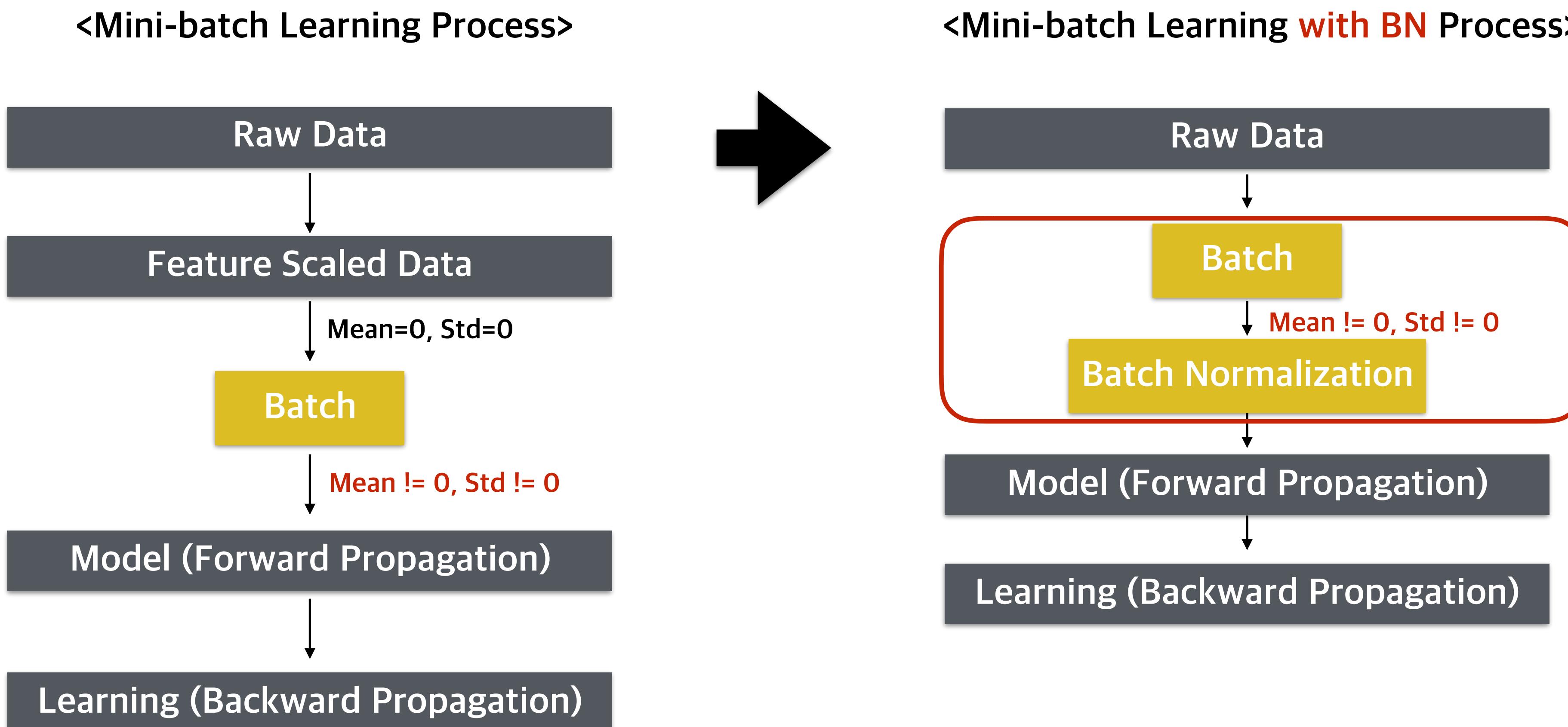


Batch Normalization

Internal Covariate Shift 현상을 근본적으로 방지하기 위해 고안된 알고리즘

- Internal Covariate Shift: 학습 시 모델의 각 layer의 input distribution이 달라지는 현상을 의미 (Recall ML is function estimator)

각 Batch 단위로 Normalization을 진행하여 Internal Covariate Shift 현상 완화



Batch Normalization Algorithm - Training

- Batch Normalization의 알고리즘은 Training 과정과 Inference(Testing) 과정으로 나뉘고 차이가 존재함
- 단순히 Batch 단위로 Normalization만 진행하는 것이 아닌, Scale & Shift(offset) 파라미터를 통한 재조정 및 학습을 진행

<BN Algorithm in Training>

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Batch 단위
Normalization

Affine Transform

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch Normalization Algorithm - Training

- Batch Normalization의 알고리즘은 Training 과정과 Inference(Testing) 과정으로 나뉘고 차이가 존재함
- 단순히 Batch 단위로 Normalization만 진행하는 것이 아닌, Scale & Shift(offset) 파라미터를 통한 재조정 및 학습을 진행

<BN Algorithm in Training>

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;
Parameters to be learned: γ, β
Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

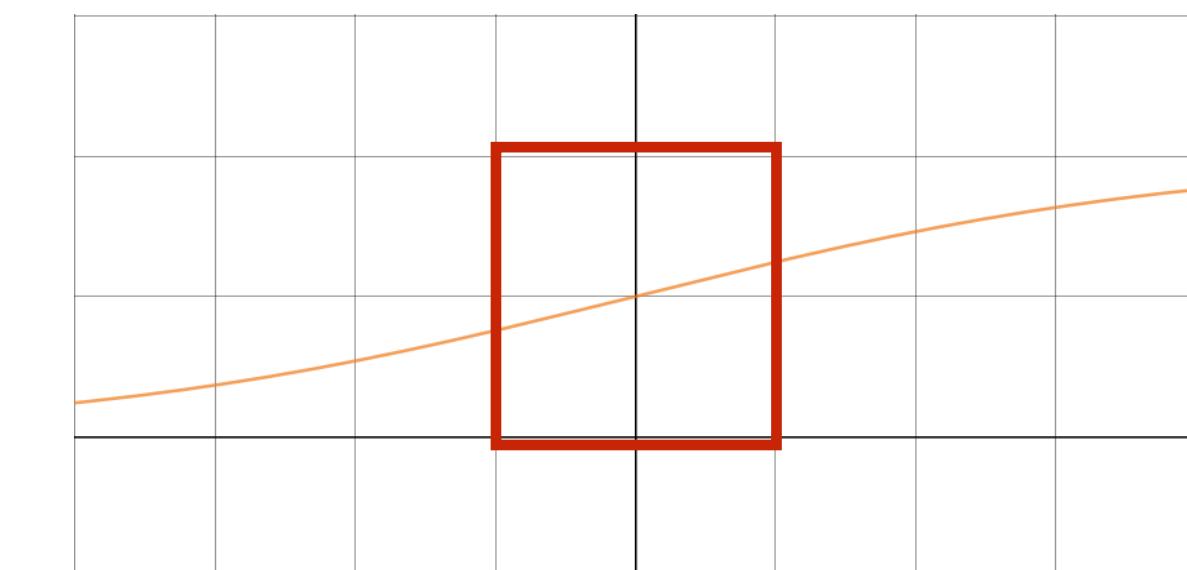
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$
$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Batch 단위
Normalization

Affine Transform

<Sigmoid Function>



- Normalization만 진행할 경우 모든 Batch의 데이터가 위의 영역에만 한정되는 현상 발생
- Nonlinear Transform을 거의 진행하지 못함
(선형에 가까움, BN을 모델 중간중간에 계속 사용 불가)
- Scale & Shift 파라미터를 통해 해당 영역을 확장
(Affine Transform)
- (*중요) Scale & Shift 파라미터는 학습의 대상

Batch Normalization Algorithm - Training

- Test 데이터는 mini-batch 단위의 기준이 존재하지 않고, test 데이터 분포에 대한 정보를 모른다고 가정해야함
(No available on statistics of test data. 말그대로 test 데이터는 단순 test를 위한 것이니까 정보를 활용하면 안됨)
- Test 데이터에 적용할 평균과 편차를 구하기 위해, 학습에 사용한 배치들의 평균과 편차 정보를 활용하여 계산

<BN Algorithm in Inference>

```
for k = 1 ... K do
    // For clarity,  $x \equiv x^{(k)}$ ,  $\gamma \equiv \gamma^{(k)}$ ,  $\mu_B \equiv \mu_B^{(k)}$ , etc.
    Process multiple training mini-batches  $B$ , each of
    size  $m$ , and average over them:
         $E[x] \leftarrow E_B[\mu_B]$ 
         $\text{Var}[x] \leftarrow \frac{m}{m-1} E_B[\sigma_B^2]$ 
    In  $N_{\text{BN}}^{\text{inf}}$ , replace the transform  $y = \text{BN}_{\gamma, \beta}(x)$  with
     $y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$ 
end for
```

학습에 사용한 Batch들의 평균&편차들의 평균을
Test 데이터의 계산에 활용

(Inference 과정에서 Test 데이터에 대한 정보가 없다고 가정해야함!)

Batch Normalization Algorithm - Training

- Test 데이터는 mini-batch 단위의 기준이 존재하지 않고, test 데이터 분포에 대한 정보를 모른다고 가정해야함
(No available on statistics of test data. 말그대로 test 데이터는 단순 test를 위한 것이니까 정보를 활용하면 안됨)
- Test 데이터에 적용할 평균과 편차를 구하기 위해, 학습에 사용한 배치들의 평균과 편차 정보를 활용하여 계산

<BN Algorithm in Inference>

```
for k = 1 ... K do
    // For clarity,  $x \equiv x^{(k)}$ ,  $\gamma \equiv \gamma^{(k)}$ ,  $\mu_B \equiv \mu_B^{(k)}$ , etc.
    Process multiple training mini-batches  $B$ , each of
    size  $m$ , and average over them:
         $E[x] \leftarrow E_B[\mu_B]$ 
         $\text{Var}[x] \leftarrow \frac{m}{m-1} E_B[\sigma_B^2]$ 
    In  $N_{\text{BN}}^{\text{inf}}$ , replace the transform  $y = \text{BN}_{\gamma, \beta}(x)$  with
        
$$y = \frac{\gamma}{\sqrt{\text{Var}[x]+\epsilon}} \cdot x + \left(\beta - \frac{\gamma E[x]}{\sqrt{\text{Var}[x]+\epsilon}}\right)$$

end for
```

학습된 Scale & Shift 파라미터와 위에서 계산한 평균&편차 정보를 가지고
Training 과정과 동일하게 계산

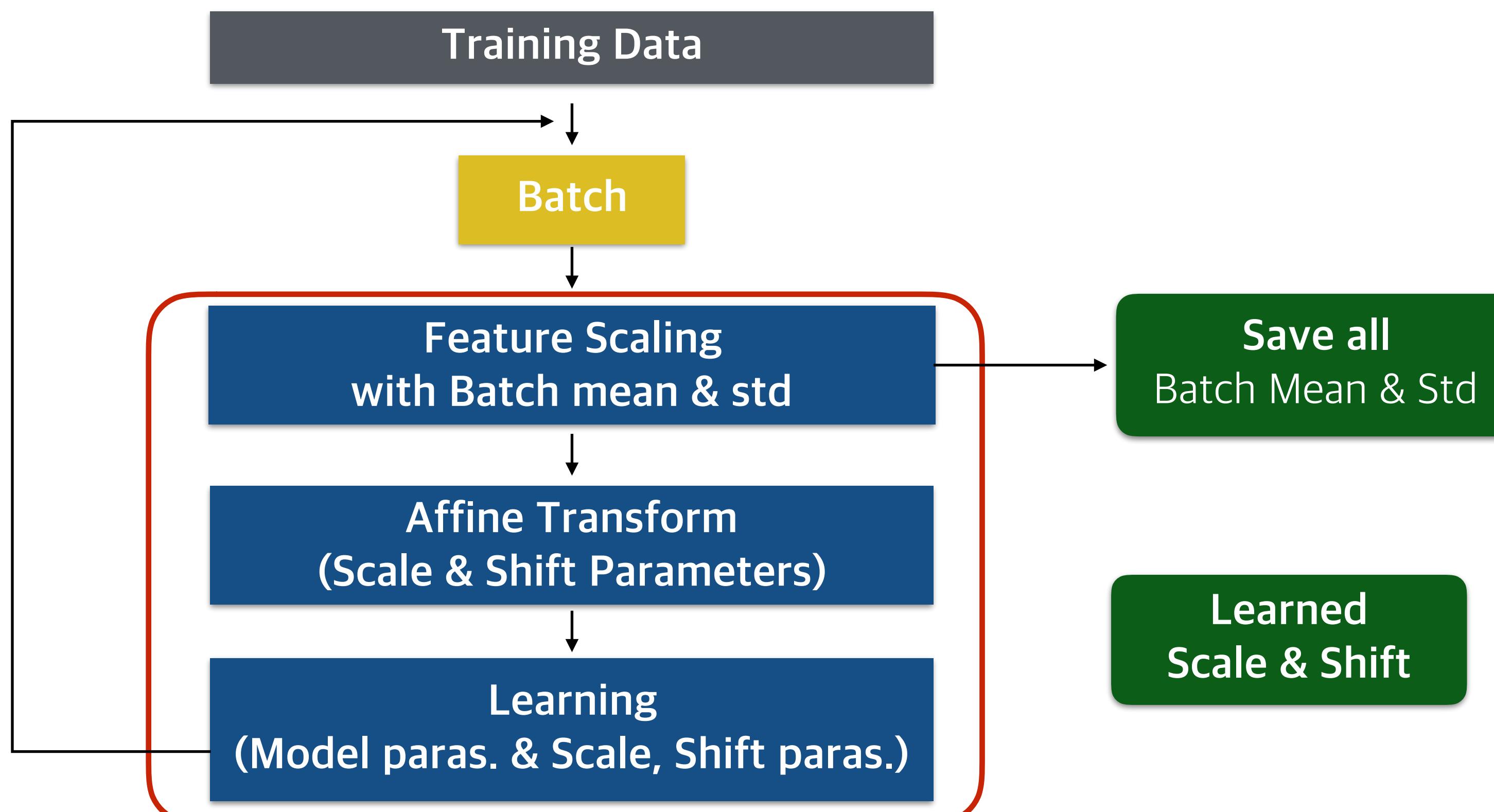
(오른쪽 논문에서는 이를 정리하여 표기한 것!)

Batch Normalization Algorithm Summary

□ Batch Normalization 알고리즘의 순서를 정리하면 아래와 같음

- 학습 도중 사용되는 모든 Batch의 평균과 편차를 메모리에 저장하는 것을 주의
- Scale & Shift 파라미터는 학습의 대상임을 주의

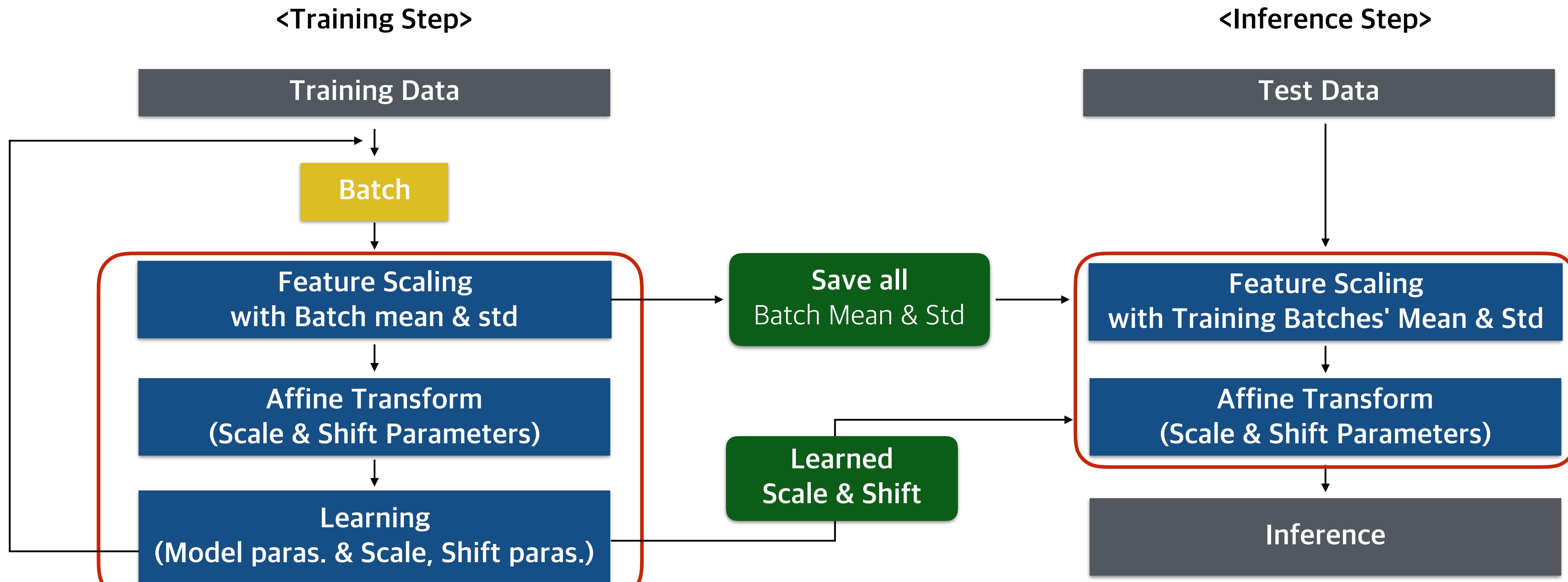
<Training Step>



Batch Normalization Algorithm Summary

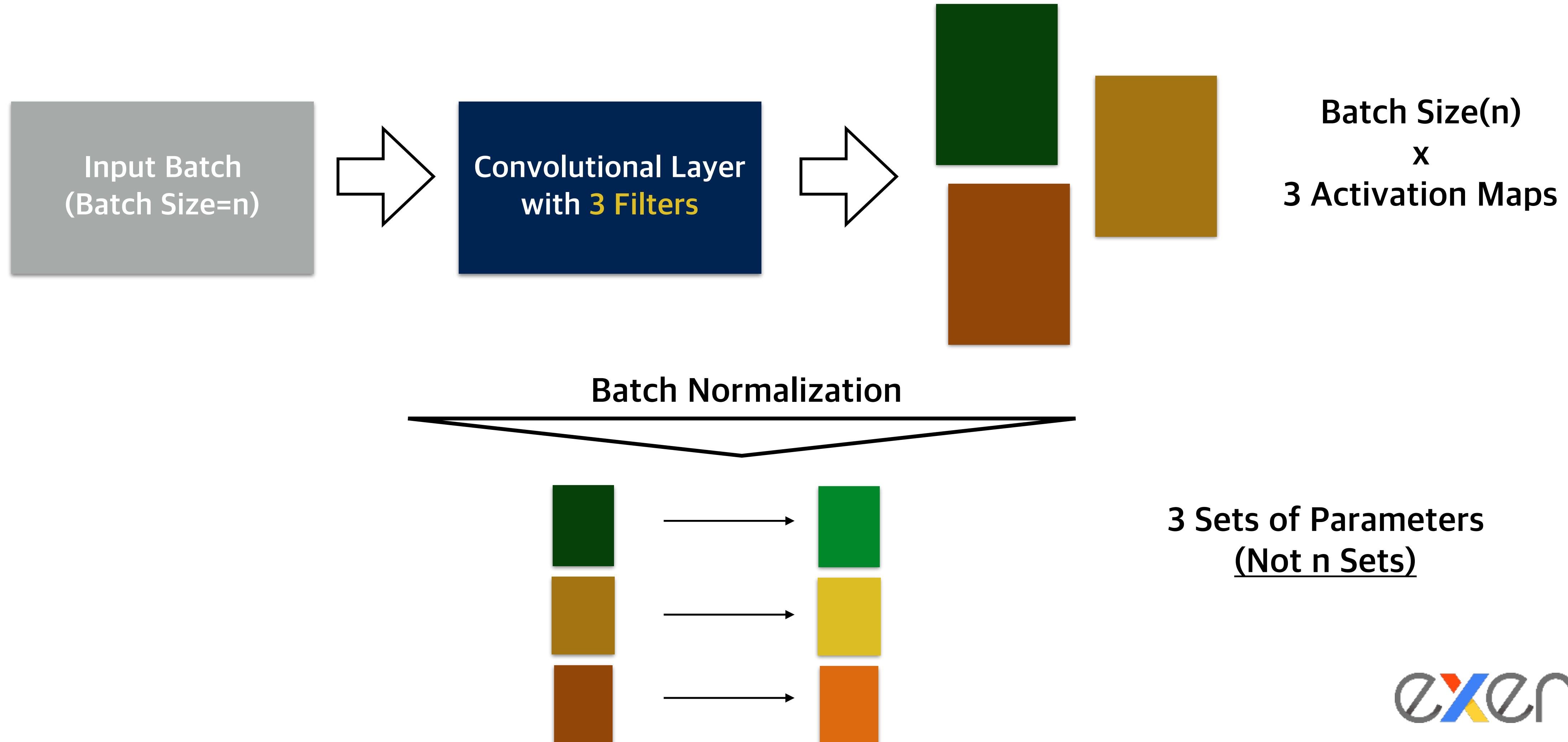
□ Batch Normalization 알고리즘의 순서를 정리하면 아래와 같음

- 학습 도중 사용되는 모든 Batch의 평균과 편차를 메모리에 저장하는 것을 주의
- Scale & Shift 파라미터는 학습의 대상임을 주의



Batch Normalization in Convolutional Neural Network

- Batch Normalization을 ConvNet에서 사용할 경우, Feature Map을 기준로 평균과 편차를 구함
- Scale & Shift 파라미터 역시 Feature Map 단위로 적용



Why Batch Normalization?

□ Batch Normalization를 사용할 경우 크게 두가지 효과를 볼 수 있음

- Increase Learning Rate (Speed Up)

: 높은 학습률과 함께 사용할 경우 효과가 커지는 것을 확인

- Remove Dropout & Regularization Parameter (Regularization)

: Dropout layer를 제거하고, regularization parameter의 크기를 줄였을 때 성능이 향상되는 것을 확인

Optimization for Training Deep Learning

Learning Deep Learning Model needs out Learning

□ Deep Learning 모델을 학습하는 것은 몇가지 문제로 인해 생각보다 까다로움

- 1) Trapped in Local Area: 최소 지점에 도달하는 것이 아닌 지역 최소점(local minimum)에 도달한 후 빠져나오지 못하는 현상
- 2) Inappropriate Parameter Update: Gradient, Hessian 값을 진행한 파라미터 업데이트가 적절하지 못한 현상

1) Trapped in Local Area

- 국소 정보의 한계로 cost의 최소점을 찾지 못하는 문제.
- 다음과 같은 요소로 인해 문제가 생김:
 - Local minima, saddle points, plateaus
 - Poor Correspondence between local & global structure
(Importance of initial point)

2) Inappropriate Parameter Update

- Gradient, Hessian의 문제로 parameter의 정확한 갱신 방향을 얻지 못하는 문제.
- 다음과 같은 요소로 문제가 생김:
 - Ill-condition
 - Long term dependency (Gradient exploding/vanishing)

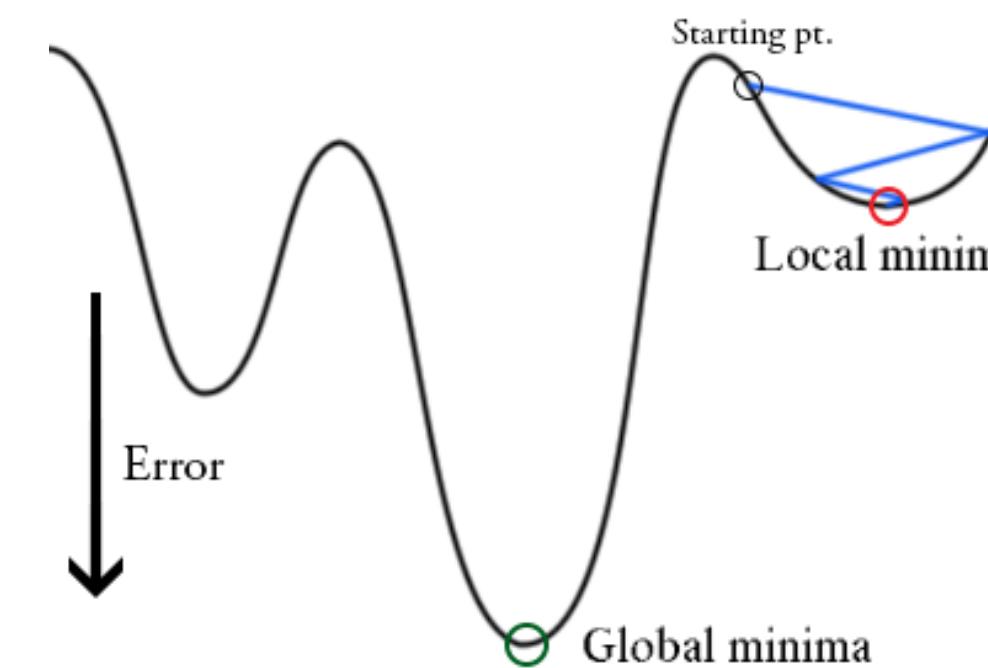
Learning Deep Learning Model needs out Learning

□ Deep Learning 모델을 학습하는 것은 몇가지 문제로 인해 생각보다 까다로움

- 1) Trapped in Local Area: 최소 지점에 도달하는 것이 아닌 지역 최소점(local minimum)에 도달한 후 빠져나오지 못하는 현상
- 2) Inappropriate Parameter Update: Gradient, Hessian 값을 진행한 파라미터 업데이트가 적절하지 못한 현상

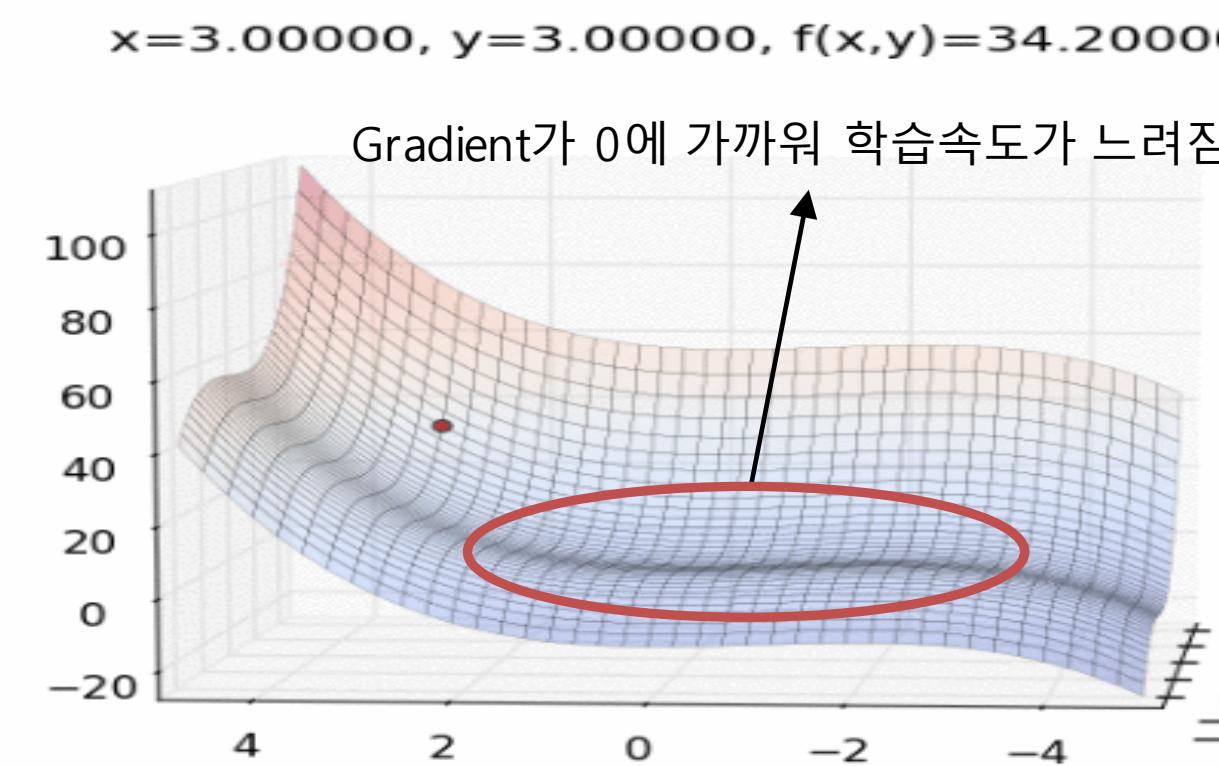
1) Local minima

- 함수 f 에 대해, x 주변에서의 함수값이 $f(x)$ 보다 작은 경우 x 를 local minimum이라고 부름.
- x 가 local minimum인 경우: gradient=0, Hessian matrix: positive semidefinite.
- 주변정보로는 더이상 학습이 불가능: 학습 종료.



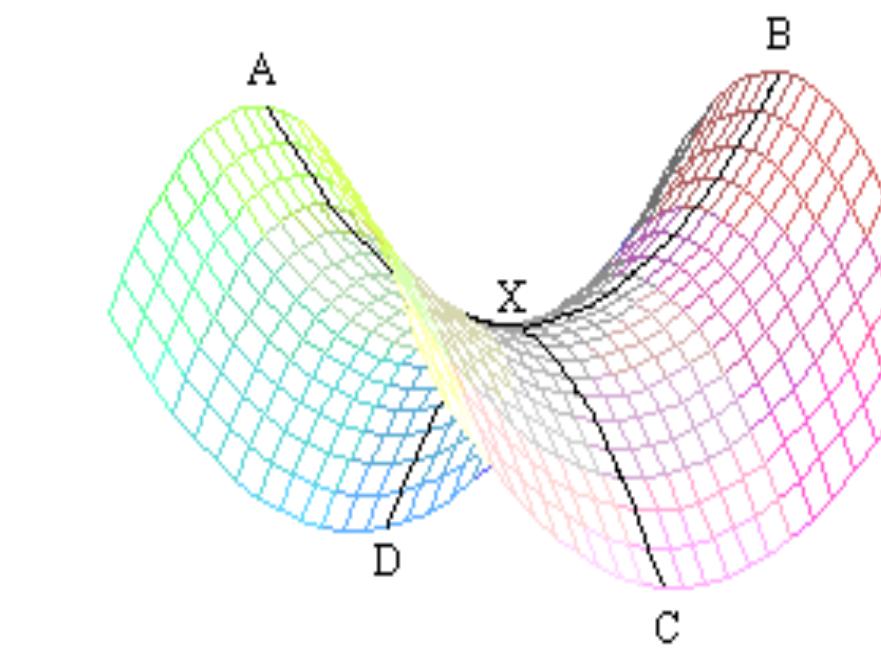
2) Plateaus

- Plateaus는 cost function의 평평한 부분을 말함.
- Cost의 변화가 거의 없어 gradient가 거의 0에 가까움 → 학습속도가 매우 느림.
- Cost가 최솟값이 아닌에도 불구하고 학습을 종료하게 될 가능성이 있음.



3) Saddle Points

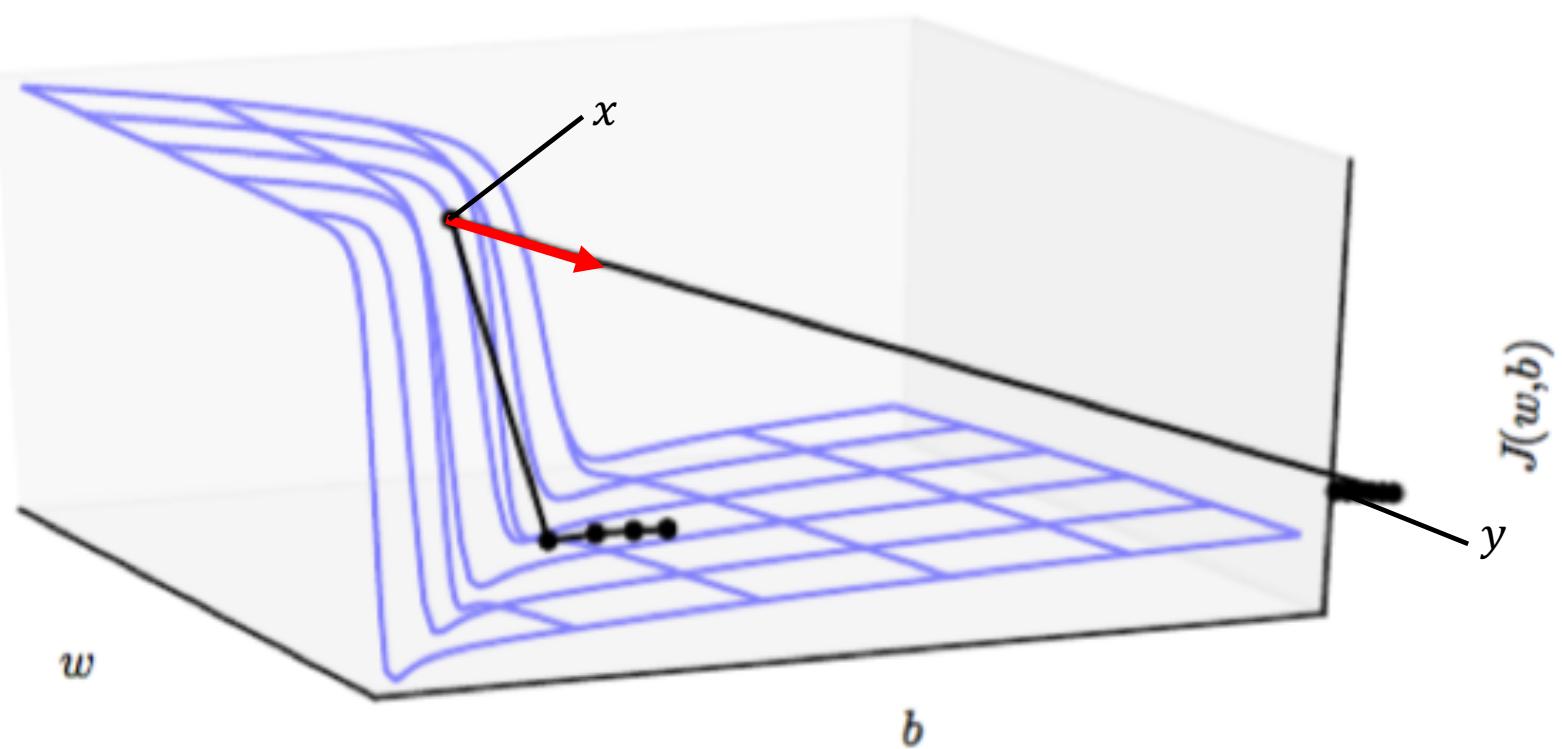
- Saddle point는 gradient는 0이지만 주변에 더 작은 값들이 있을 경우.
- Hessian matrix가 indefinite임.
- Local minima를 지나는 선과 local maxima를 지나는 선이 만나는 지점으로 해석 가능.
- Newton's method: 빠져나가기 힘듦.
- Gradient descent method: saddle point의 주변을 지나가 saddle point에서 멈추지 않음.



Learning Deep Learning Model needs out Learning

□ Deep Learning 모델을 학습하는 것은 몇가지 문제로 인해 생각보다 까다로움

- 1) Trapped in Local Area: 최소 지점에 도달하는 것이 아닌 지역 최소점(local minimum)에 도달한 후 빠져나오지 못하는 현상
- 2) Inappropriate Parameter Update: Gradient, Hessian 값을 진행한 파라미터 업데이트가 적절하지 못한 현상
 - 특히 Layer 수가 많은 경우 + 파라미터의 크기가 지나치게 커질경우 : **Gradient Exploding!**



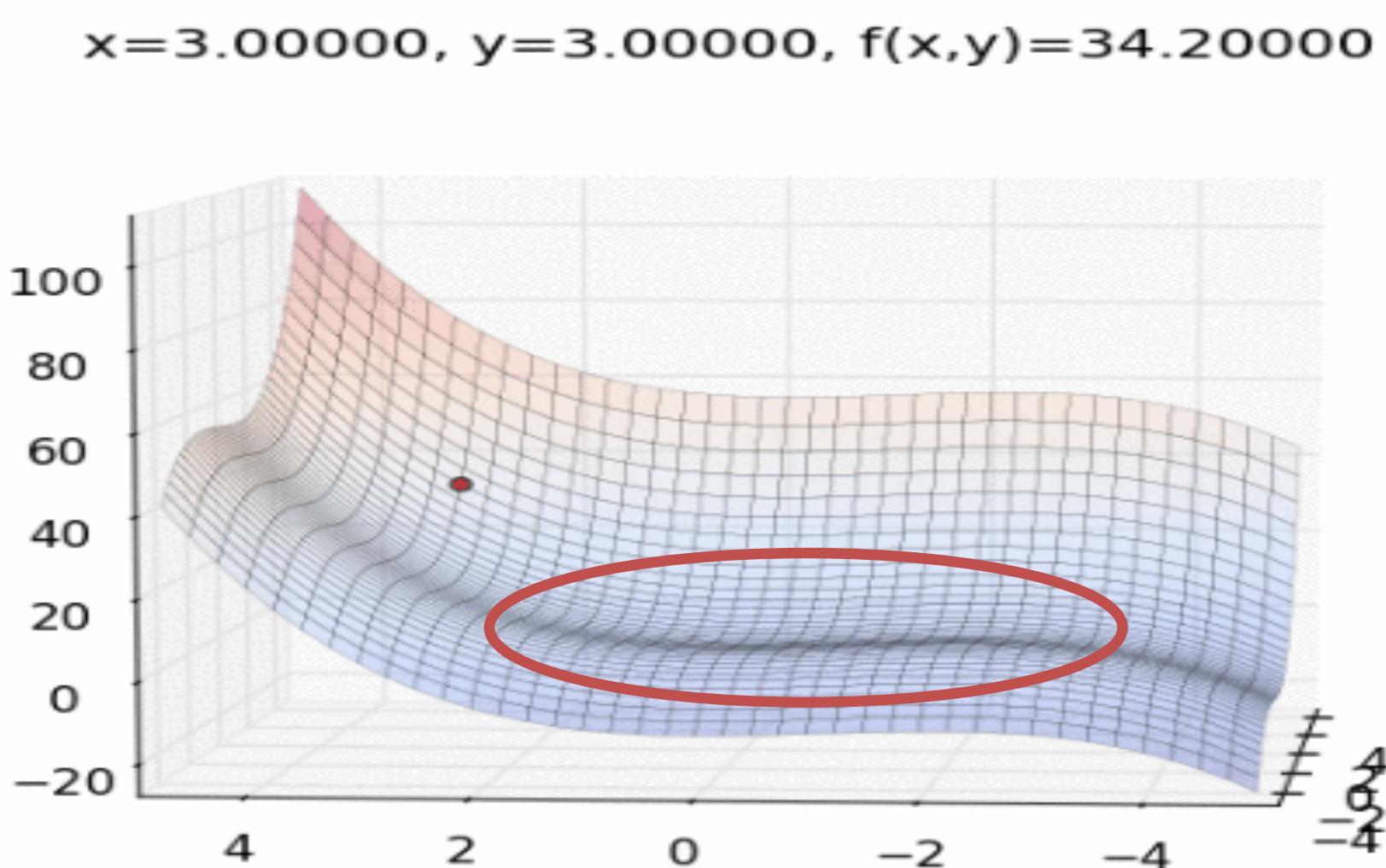
- Parameter가 cliff 근처에 있을경우
→ 현재 위치에서 매우 먼 지점으로 parameter를 간신힘.
- 최적화 지점에서 멀리 떨어짐 → 수렴에 더 많은 시간이 소요됨.

<Cliff: An example of the problem>

Critical Problems in Gradient Descent Method

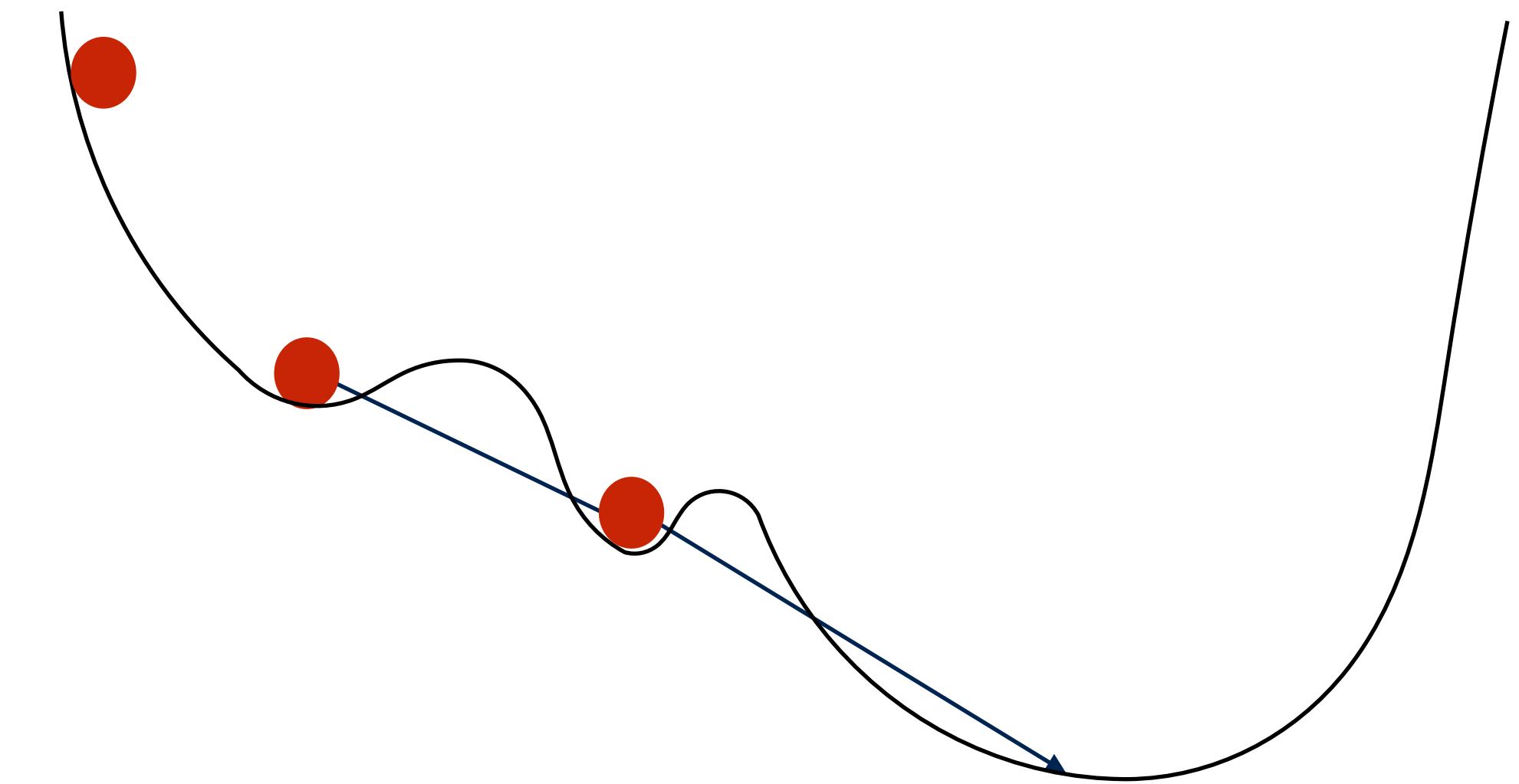
- Gradient Descent를 학습에 이용할 경우, Plateaus와 같은 평평한 부분에서 학습이 진행되지 않는 문제가 있음
- Local Optimum에 수렴할 경우, 빠져나오지 못하는 현상 발생

<Plateaous Example>



Gradient가 0은 아니지만 매우 작은 값을 가져서
학습이 느리게 이루어짐

<Trapped in Local Optimum>



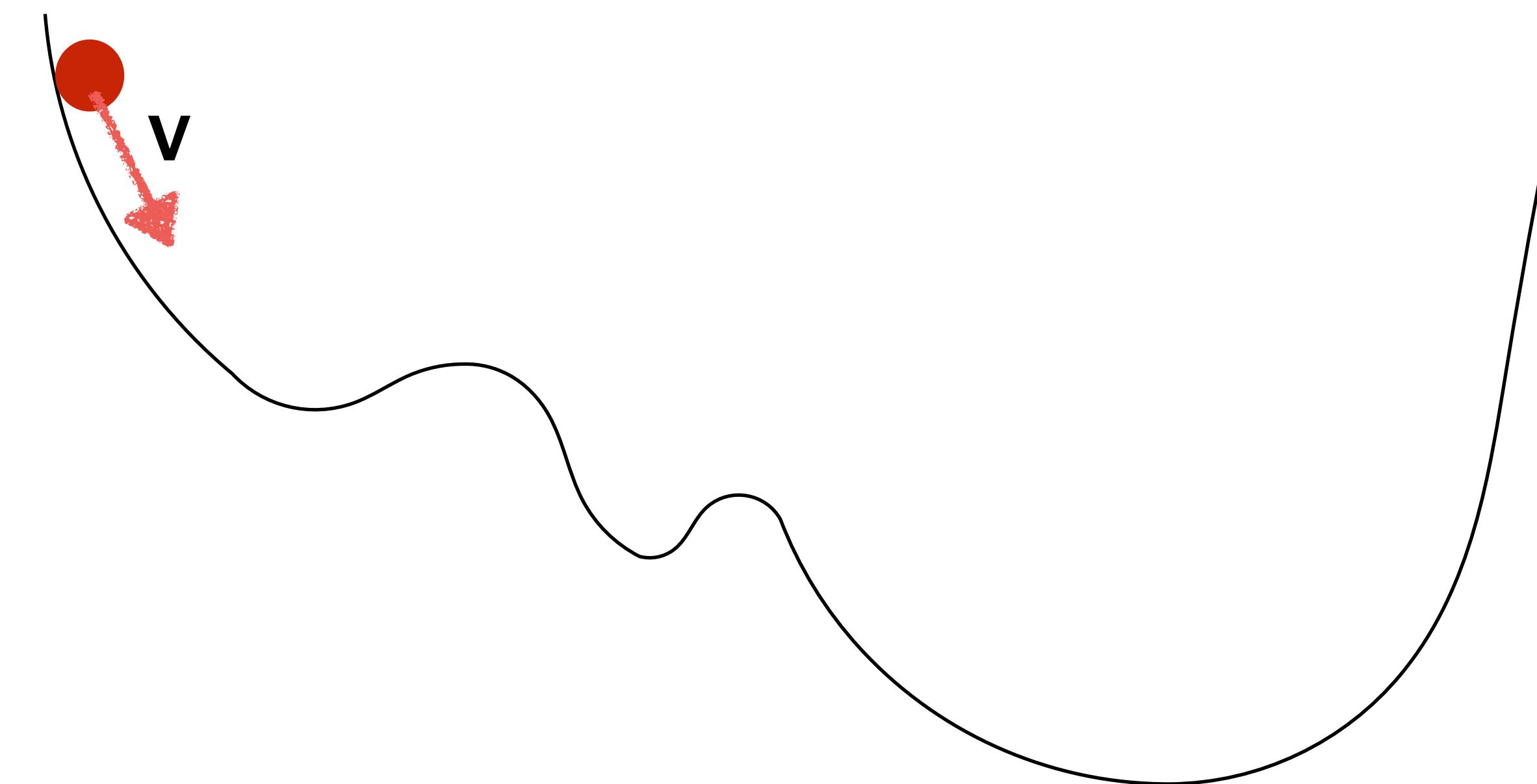
Local Optimum에 빠질 경우
Gradient = 0 이 되어 학습이 이루어지지 않음

How can we solve this problem?

Simple Solution: Momentum

- Gradient Descent Method는 단계별로 위치를 변화시키며 파라미터를 업데이트하는 방법임
- (Idea*) Loss function 위에서 공을 굴린다고 생각한다면?
 - 공이 내려갈 때 위치 변화에 따라 속도를 가지며 이동하게됨
 - 큰 속력을 가지고 내려가는 도중, local optimum 지점에 도달해도 관성으로 인해 지나쳐 언덕을 넘어갈 수 있음

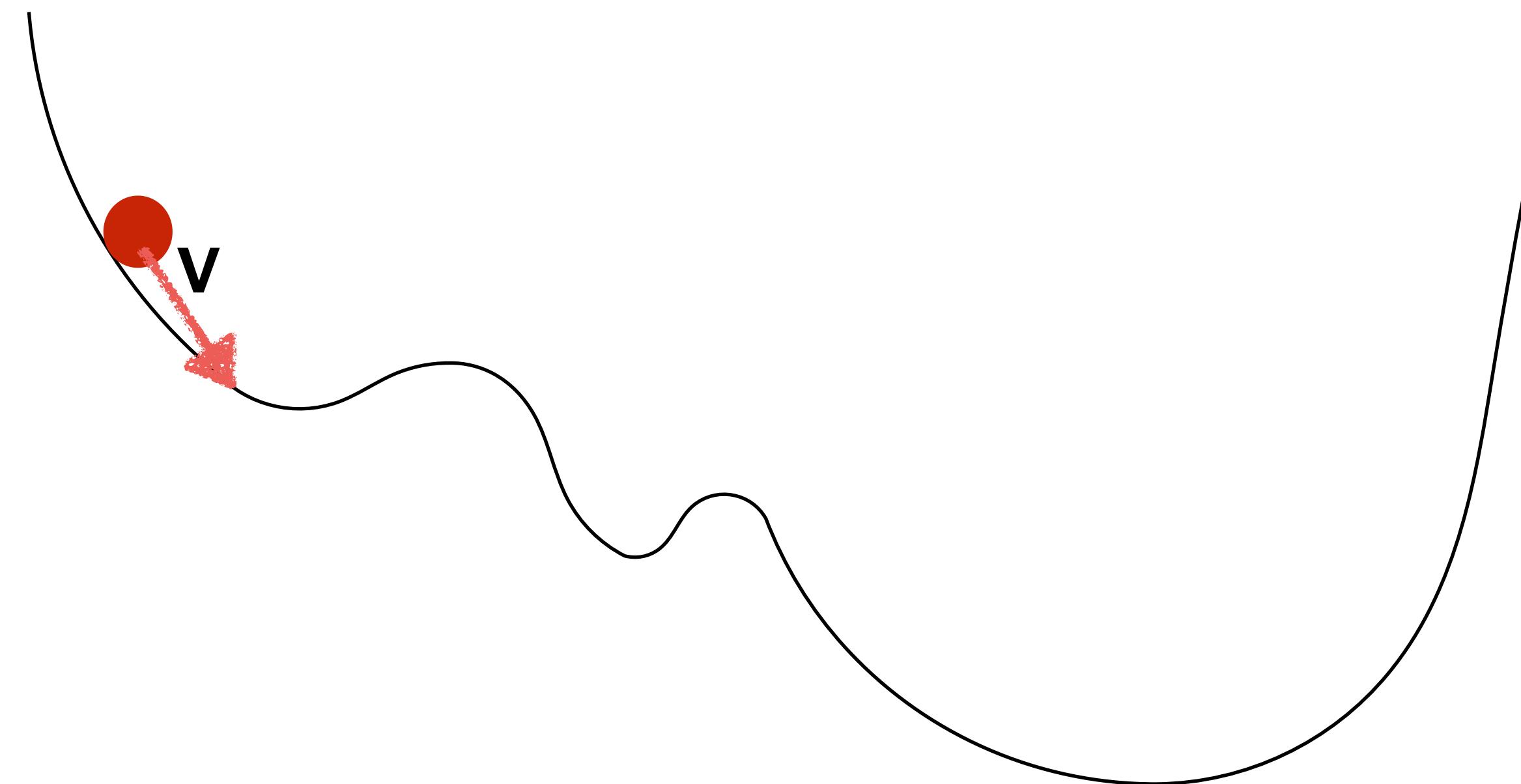
<Parameter Update with Momentum>



Simple Solution: Momentum

- Gradient Descent Method는 단계별로 위치를 변화시키며 파라미터를 업데이트하는 방법임
- (Idea*) Loss function 위에서 공을 굴린다고 생각한다면?
 - 공이 내려갈 때 위치 변화에 따라 속도를 가지며 이동하게됨
 - 큰 속력을 가지고 내려가는 도중, local optimum 지점에 도달해도 관성으로 인해 지나쳐 언덕을 넘어갈 수 있음

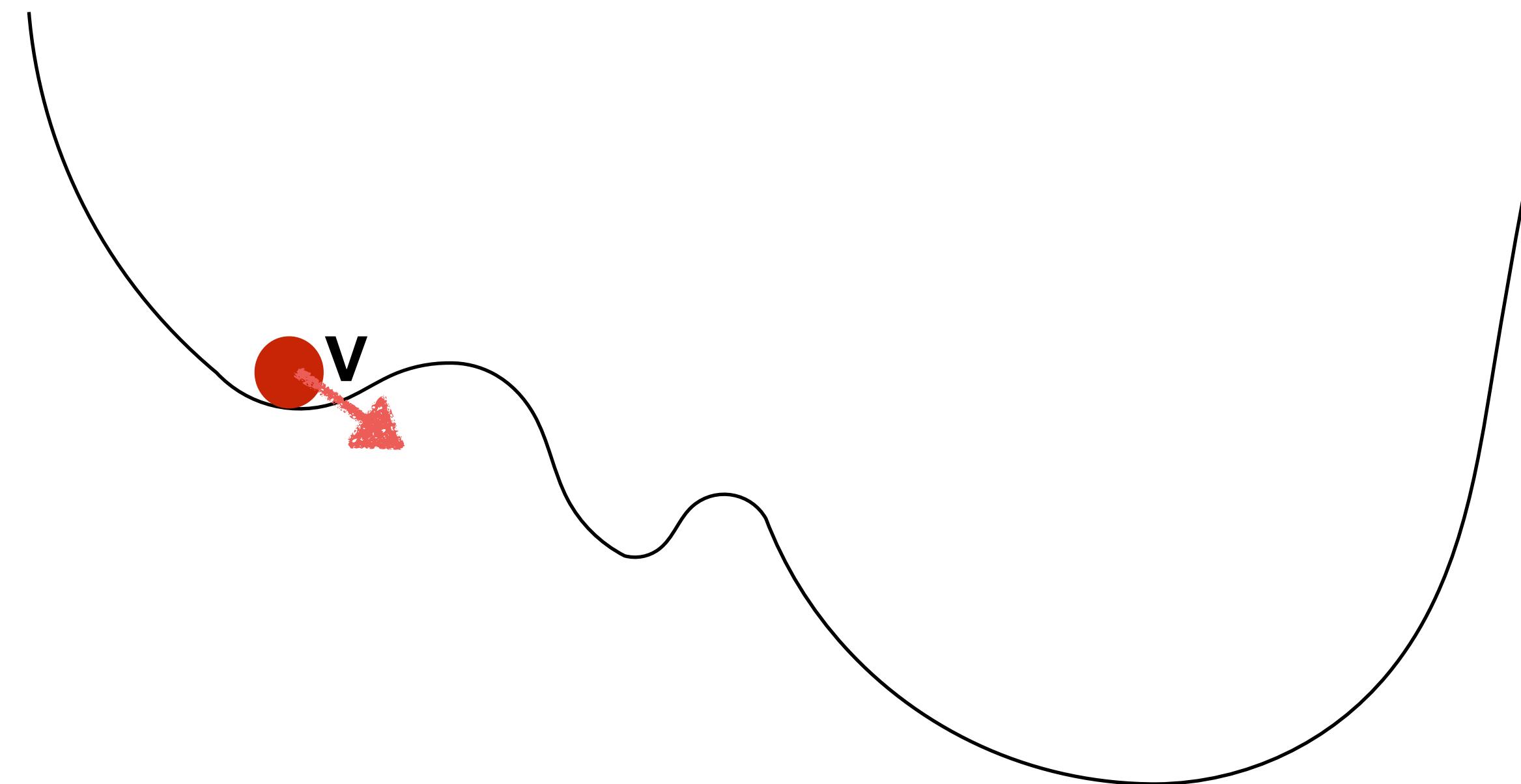
<Parameter Update with Momentum>



Simple Solution: Momentum

- Gradient Descent Method는 단계별로 위치를 변화시키며 파라미터를 업데이트하는 방법임
- (Idea*) Loss function 위에서 공을 굴린다고 생각한다면?
 - 공이 내려갈 때 위치 변화에 따라 속도를 가지며 이동하게됨
 - 큰 속력을 가지고 내려가는 도중, local optimum 지점에 도달해도 관성으로 인해 지나쳐 언덕을 넘어갈 수 있음

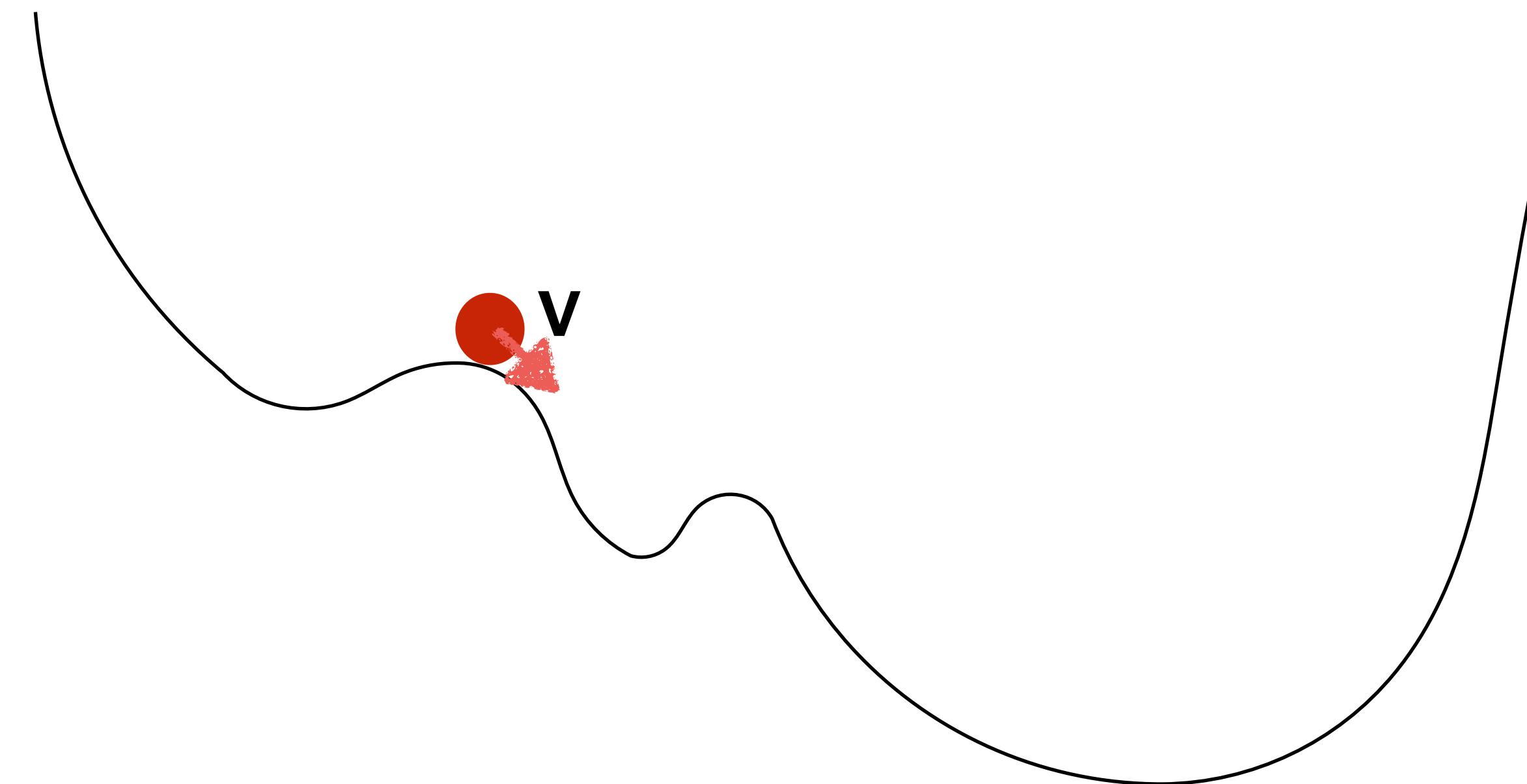
<Parameter Update with Momentum>



Simple Solution: Momentum

- Gradient Descent Method는 단계별로 위치를 변화시키며 파라미터를 업데이트하는 방법임
- (Idea*) Loss function 위에서 공을 굴린다고 생각한다면?
 - 공이 내려갈 때 위치 변화에 따라 속도를 가지며 이동하게됨
 - 큰 속력을 가지고 내려가는 도중, local optimum 지점에 도달해도 관성으로 인해 지나쳐 언덕을 넘어갈 수 있음

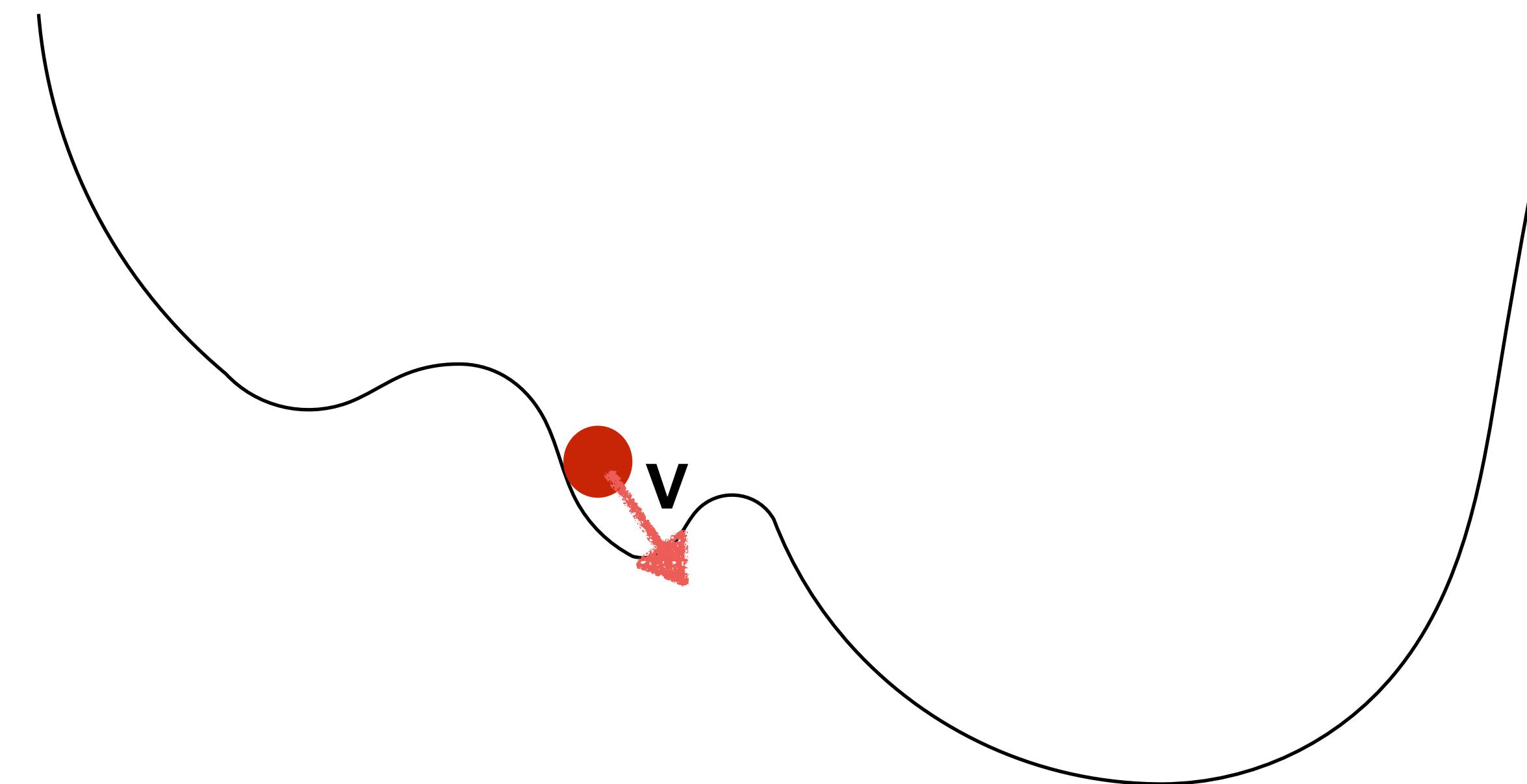
<Parameter Update with Momentum>



Simple Solution: Momentum

- Gradient Descent Method는 단계별로 위치를 변화시키며 파라미터를 업데이트하는 방법임
- (Idea*) Loss function 위에서 공을 굴린다고 생각한다면?
 - 공이 내려갈 때 위치 변화에 따라 속도를 가지며 이동하게됨
 - 큰 속력을 가지고 내려가는 도중, local optimum 지점에 도달해도 관성으로 인해 지나쳐 언덕을 넘어갈 수 있음

<Parameter Update with Momentum>



Momentum Algorithm

- Loss의 파라미터에 대한 Gradient를 바탕으로 속도를 정의하고, 이를 파라미터 업데이트에 이용
- 과거 위치 변화 정도(속도)의 영향을 고려하여 파라미터 업데이트를 진행하며, 어느 정도의 local optimum 문제 해결 가능

<Momentum Algorithm>

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ , momentum parameter α .

Require: Initial parameter θ , initial velocity v .

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

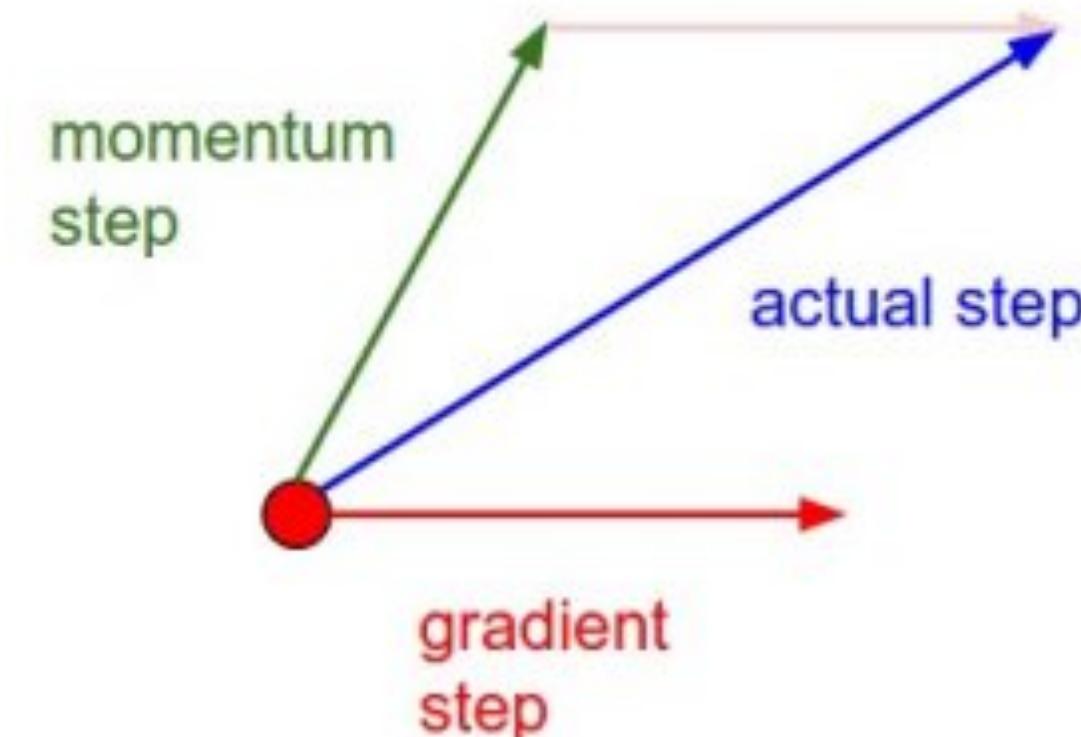
 Apply update: $\theta \leftarrow \theta + \mathbf{v}$

end while

누적된 과거 위치 변화 정도

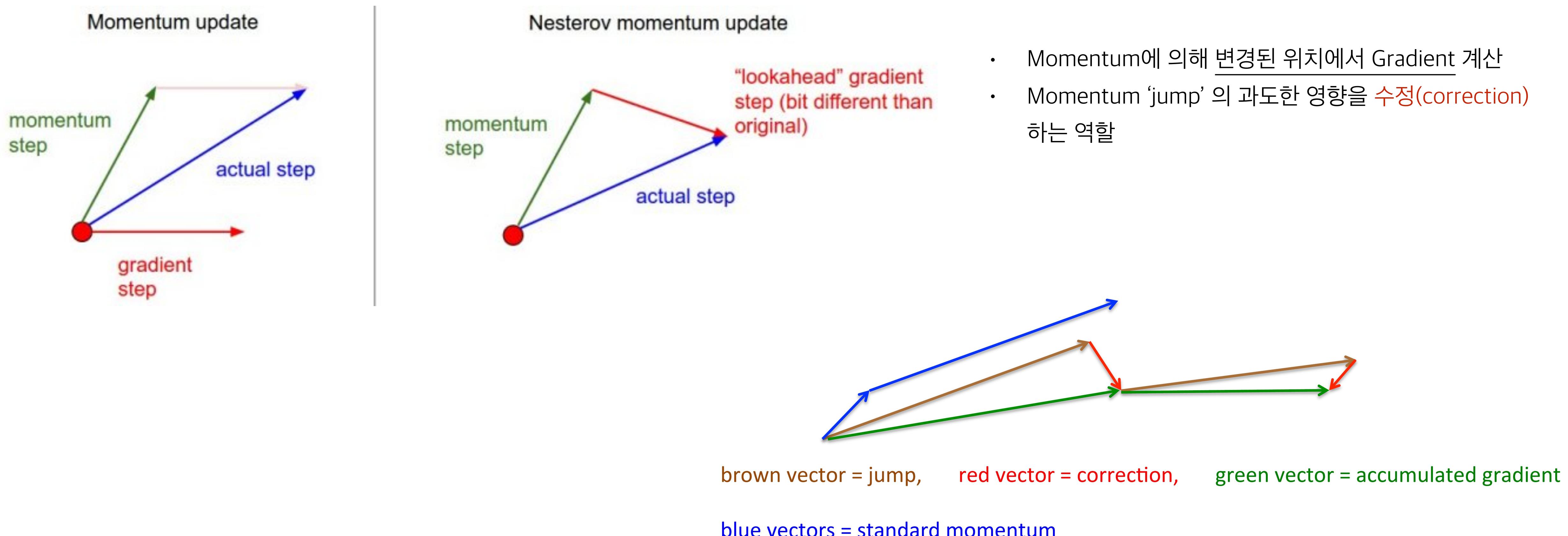
현재 위치에서
변화하고자 하는 위치 변화 정도

Momentum update



Nesterov Momentum

- ▣ 기존 Momentum Method의 변형된 알고리즘
- ▣ 파라미터 업데이트에 사용되는 gradient 계산의 위치를 변경한 방법으로 현재 속도에 의한 ‘jump’의 영향을 높임
 - The Momentum Step: ‘Jump’
 - The Gradient Step: ‘Correction’



Nesterov Momentum Algorithm

❑ Nestrerov Momentum 알고리즘

- Momentum 만을 고려한 파라미터 업데이트 지점 계산
- 해당 지점에서 Gradient 계산
- 기존 Momentum과 변경 지점에서의 Gradient를 통해, 최종 속도를 업데이트

Algorithm 8.3 Mini-batch Gradient Descent with Nesterov momentum

Require: Learning rate ϵ , momentum parameter α .

Require: Initial parameter θ , initial velocity v .

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding labels $\mathbf{y}^{(i)}$.

 Apply interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

 Compute gradient (at interim point): $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(\mathbf{x}^{(i)}; \tilde{\theta}), \mathbf{y}^{(i)})$

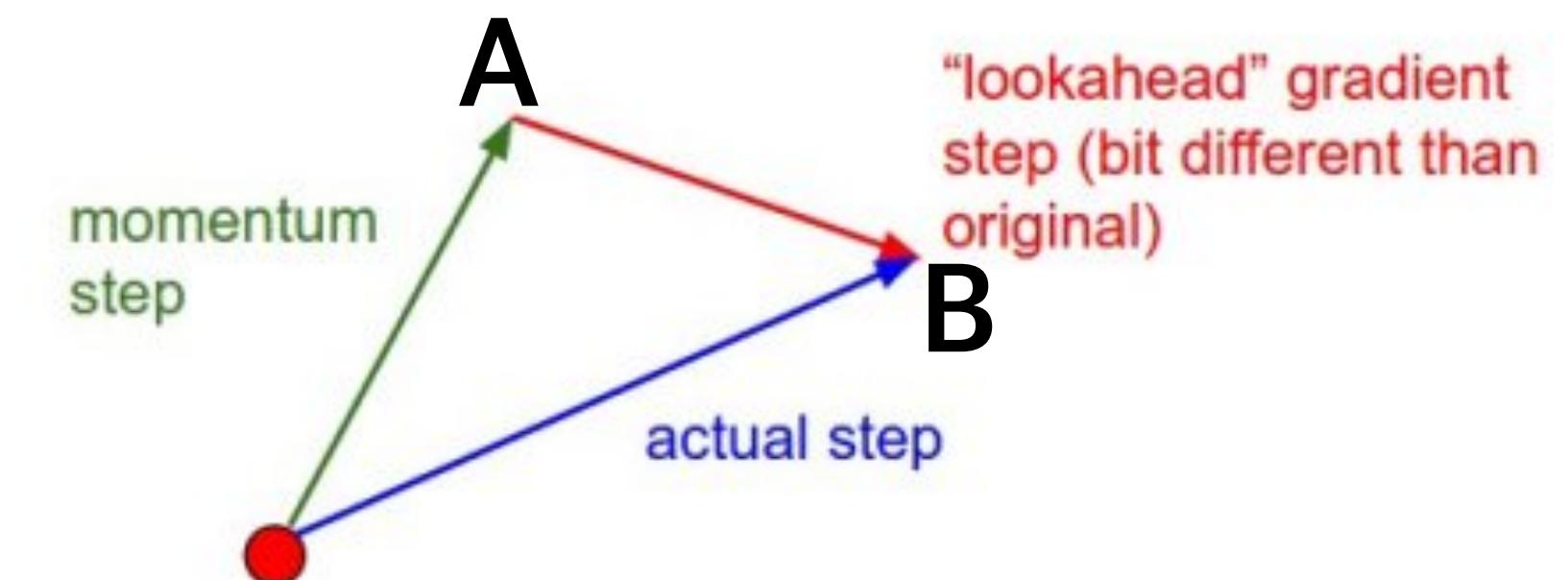
 Compute velocity update: $v \leftarrow \alpha v - \epsilon g$

 Apply update: $\theta \leftarrow \theta + v$

end while

A
B

Nesterov momentum update

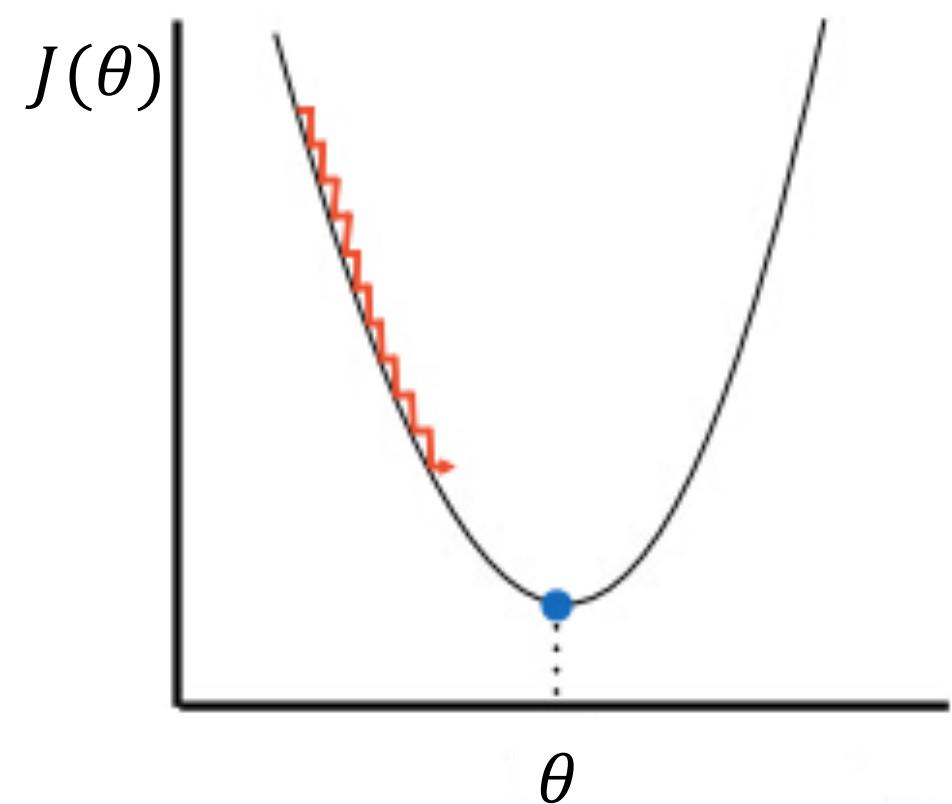


- ▣ 기존 Gradient Descent Method는 고정된 Learning Rate을 사용
- ▣ Learning Rate의 크기는 학습에 큰 영향을 미치며, 이를 결정하는 것이 학습 속도 및 학습 여부에 있어 매우 중요

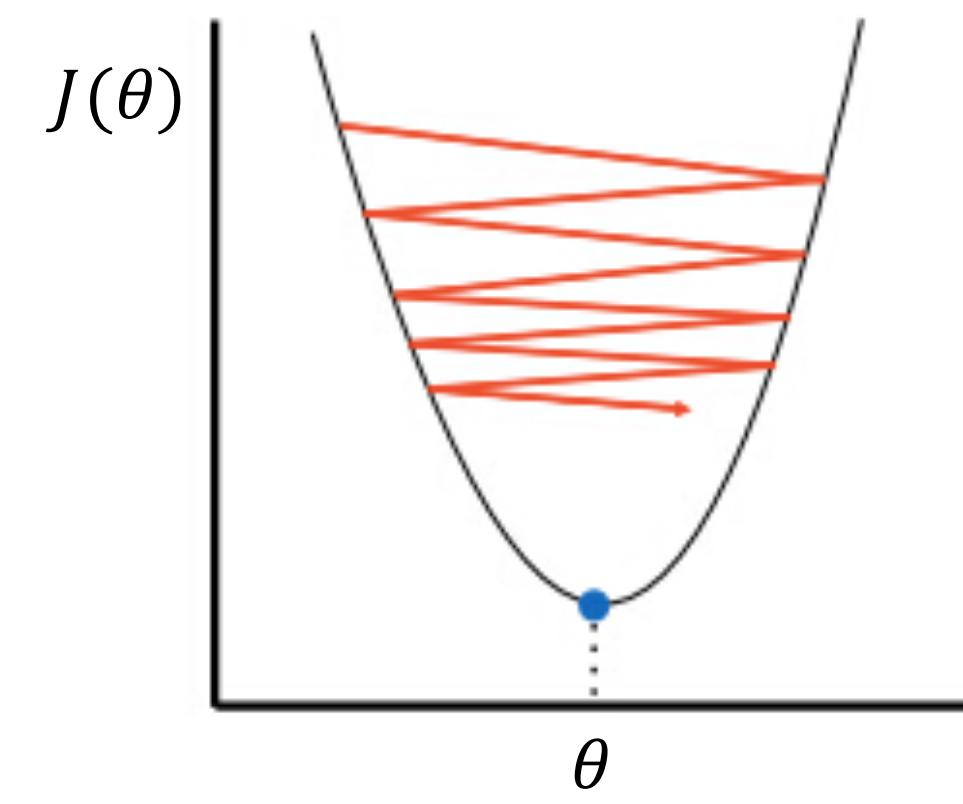
<Gradient Descent Method>

$$\theta_j := \theta_j - \underline{\alpha} \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

(α =learning rate)



Too small: converge
very slowly



Too big: overshoot and
even diverge

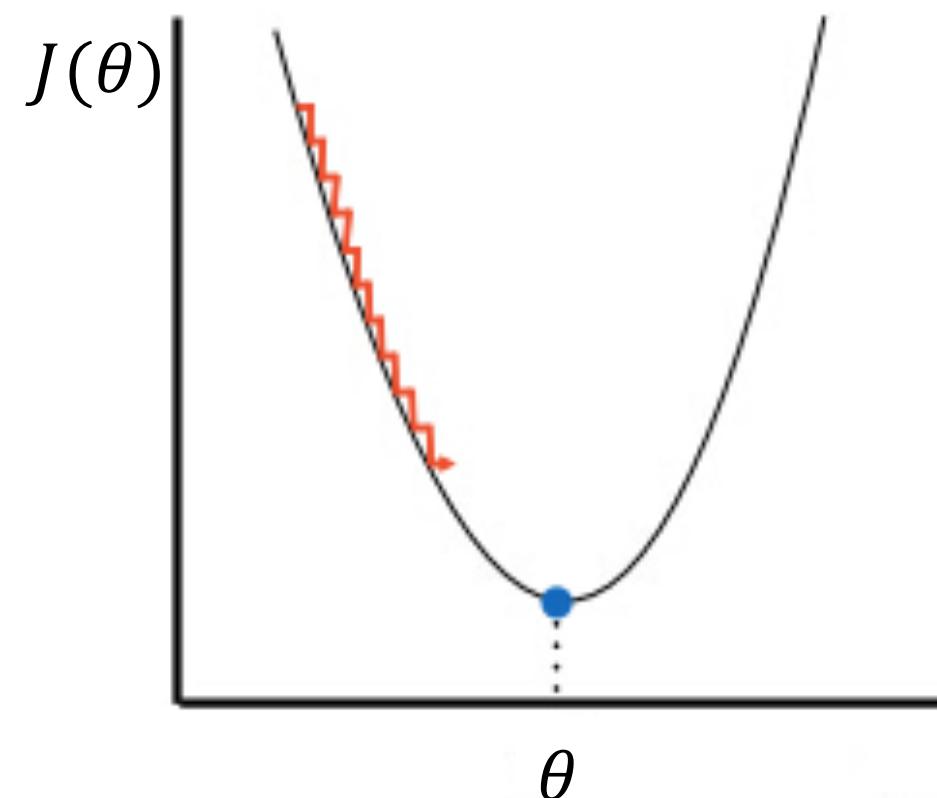
Adaptive Learning Rate

- 기존 Gradient Descent Method는 고정된 Learning Rate을 사용
- Learning Rate의 크기는 학습에 큰 영향을 미치며, 이를 결정하는 것이 학습 속도 및 학습 여부에 있어 매우 중요

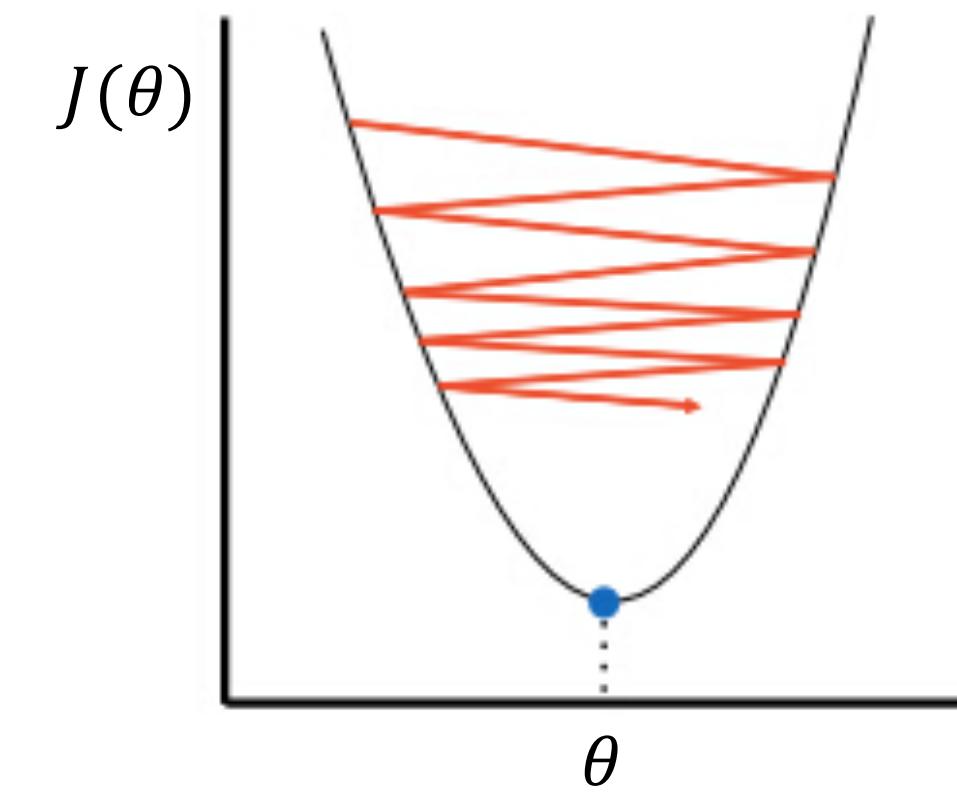
<Gradient Descent Method>

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

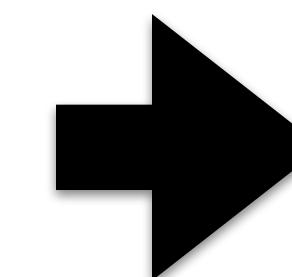
(α=learning rate)



Too small: converge very slowly



Too big: overshoot and even diverge



(Idea*) 학습 초기에 큰 Learning Rate으로 빠르게 학습
학습 후반에는 작은 Learning Rate으로 정교하게 학습

<Adaptive Learning>

학습 진행에 따라
학습률(또는 gradient)의 크기를 조절해가며 학습하는 방법

Simple Adaptive Learning: Annealing the Learning Rate

□ Adaptive Learning을 구현하는 가장 간단한 방법은 시간에 따라서 지속적으로 Learning Rate를 줄여주는 것

- Step Decay: 특정 epoch 수마다 learning rate을 일정 비율 줄여나가는 방식

- Exponential Decay: Exponential 비율로 지속적으로 learning rate 조절

$$\alpha = \alpha_0 e^{-kt}$$

- 1/t Decay: $1/t$ 비율로 지속적으로 learning rate 조절

$$\alpha = \alpha_0 / (1 + kt)$$

AdaGrad Optimizer

- AdaGrad Optimizer는 gradient의 이전 값들을 모두 고려하여 learning rate을 조절하는 방식
- 이전 gradient의 제곱의 합의 누적을 바탕으로 learning rate을 매 스텝마다 줄여나감

$$\Delta\theta = -\epsilon \left(\frac{g_{1t}}{\sqrt{\sum_{i=1}^{t-1} g_{1i}^2}}, \dots, \frac{g_{nt}}{\sqrt{\sum_{i=1}^{t-1} g_{ni}^2}} \right)^T$$

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $r = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

 Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}$. (Division and square root applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

- Global Learning rate ϵ 설정.
 - Parameter θ , 축적 변수 $r = 0$ 설정.
- (batch size) m 개의 표본을 training set에서 추출
 - Gradient $\mathbf{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ 계산.
 - Gradient 축적: $\mathbf{r} := \mathbf{r} + \mathbf{g} \odot \mathbf{g}$
각 element의 제곱을 축적.
 - Parameter의 변화량 설정: $\Delta\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}$.
Gradient의 사이즈를 element별로 scaling.
 - θ 값 갱신: $\theta := \theta + \Delta\theta$.
 - 다음 m 개의 자료를 뽑아 같은 작업 반복: 수렴할 때까지.

AdaGrad Optimizer

- AdaGrad Optimizer는 gradient의 이전 값들을 모두 고려하여 learning rate을 조절하는 방식
- 이전 gradient의 제곱의 합의 누적을 바탕으로 learning rate을 매 스텝마다 줄여나감

$$\Delta\theta = -\epsilon \left(\frac{g_{1t}}{\sqrt{\sum_{i=1}^{t-1} g_{1i}^2}}, \dots, \frac{g_{nt}}{\sqrt{\sum_{i=1}^{t-1} g_{ni}^2}} \right)^T$$

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ

Require: Initial parameter θ

Require: Small constant δ , perhaps 10^{-7} , for numerical stability

Initialize gradient accumulation variable $r = 0$

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$

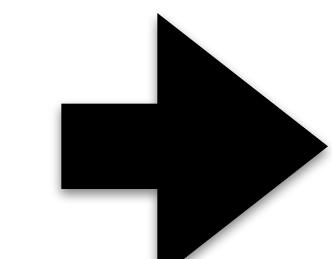
 Compute update: $\Delta\theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}$. (Division and square root applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

- Global Learning rate ϵ 설정.
 - Parameter θ , 축적 변수 $r = 0$ 설정.
- (batch size) m 개의 표본을 training set에서 추출
 - Gradient $\mathbf{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ 계산.
 - Gradient 축적: $\mathbf{r} := \mathbf{r} + \mathbf{g} \odot \mathbf{g}$
각 element의 제곱을 축적.
 - Parameter의 변화량 설정: $\Delta\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot \mathbf{g}$.
Gradient의 사이즈를 element별로 scaling.
 - θ 값 갱신: $\theta := \theta + \Delta\theta$.
 - 다음 m 개의 자료를 뽑아 같은 작업 반복: 수렴할 때까지.

모든 Gradient 값들이 축적되기 때문에



시간이 갈수록 파라미터 업데이트가 더뎌지는 문제

>> 학습 완료 이전에 학습이 끝나는 문제 발생

RMSProp Optimizer

- 과거 모든 Gradient의 값이 누적되는 부분을 변경한 최적화 방법
- Decay rate을 추가하여 오래된 gradient 정보를 삭제하며, AdaGrad 이용
- 현재 Neural Network 분야에서 많이 쓰이는 Optimizer 중 하나

<Comparison of AdaGrad and RMSProp>

Accumulate Squared Gradient

$$\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \quad \rightarrow \quad \mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$$

RMSProp Optimizer

- 과거 모든 Gradient의 값이 누적되는 부분을 변경한 최적화 방법
- Decay rate을 추가하여 오래된 gradient 정보를 삭제하며, AdaGrad 이용
- 현재 Neural Network 분야에서 많이 쓰이는 Optimizer 중 하나

<Comparison of AdaGrad and RMSProp>

Accumulate Squared Gradient

$$\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g} \quad \rightarrow \quad \mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$$

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ , decay rate ρ .

Require: Initial parameter θ

Require: Small constant δ , usually 10^{-6} , used to stabilize division by small numbers.

Initialize accumulation variables $\mathbf{r} = 0$

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Accumulate squared gradient: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

 Compute parameter update: $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta+r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{\delta+r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

- Global Learning rate ϵ , decay rate ρ 설정.
- Parameter θ , 축적 변수 $r = 0$ 설정

1. (batch size) m 개의 표본을 training set에서 추출
2. Gradient $\mathbf{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ 계산.
3. Gradient 축적: $r := \rho r + (1 - \rho) \mathbf{g} \odot \mathbf{g}$
각 element의 제곱을 decay rate ρ 로 scaling하여 축적.
4. Parameter의 변화량 설정: $\Delta\theta = -\frac{\epsilon}{\sqrt{\delta+r}} \odot \mathbf{g}$.
Gradient의 사이즈를 element별로 scaling.
5. θ 값 갱신: $\theta := \theta + \Delta\theta$.
6. 다음 m 개의 자료를 뽑아 같은 작업 반복: 수렴할 때까지.

RMSProp with Nesterov Momentum

- ❑ RMSProp의 더 높은 효과를 위해 Nesterov momentum과 결합하여 사용하기도 함
- ❑ RMSProp의 Accumulate Gradient를 계산할 때 Momentum에 의해 “Jump”한 지점의 값을 이용

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ , decay rate ρ , momentum coefficient α .

Require: Initial parameter θ , initial velocity v . Momentum 사용.

Initialize accumulation variable $r = 0$

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.

 Compute interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

Nesterov method

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$

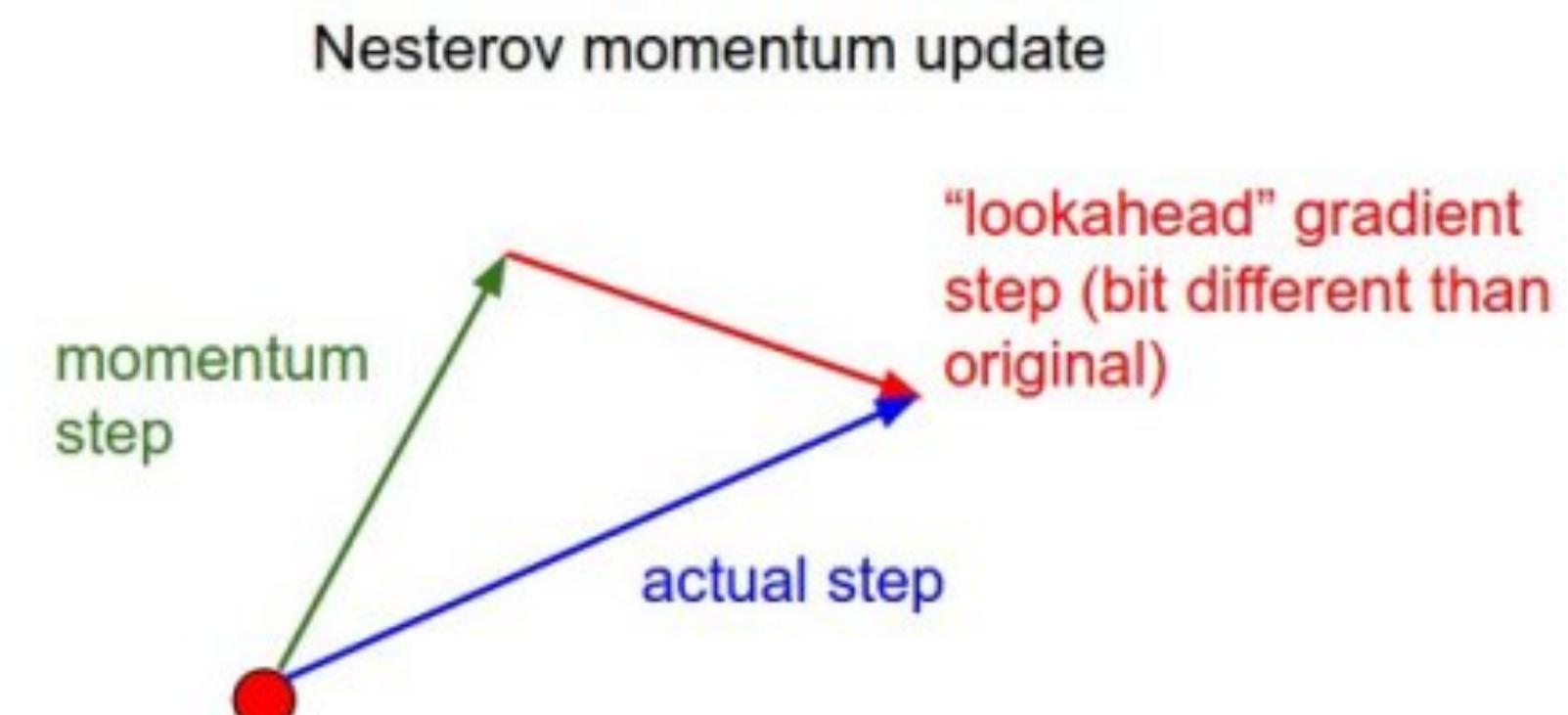
 Accumulate gradient: $r \leftarrow \rho r + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

 Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + v$

end while

- Gradient 제곱의 합을 누적하여 Learning Rate 감소
- 오래된 Gradient 정보 제거
- Momentum을 이용한 파라미터 업데이트
- Momentum “Jump” 지점의 Gradient 정보 사용



RMSProp with Nesterov Momentum

- ❑ RMSProp의 더 높은 효과를 위해 Nesterov momentum과 결합하여 사용하기도 함
- ❑ RMSProp의 Accumulate Gradient를 계산할 때 Momentum에 의해 “Jump”한 지점의 값을 이용

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ , decay rate ρ , momentum coefficient α .

Require: Initial parameter θ , initial velocity v . Momentum 사용.

Initialize accumulation variable $r = 0$

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{x^{(1)}, \dots, x^{(m)}\}$ with corresponding targets $y^{(i)}$.

 Compute interim update: $\tilde{\theta} \leftarrow \theta + \alpha v$

Nesterov method

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\theta}} \sum_i L(f(x^{(i)}; \tilde{\theta}), y^{(i)})$

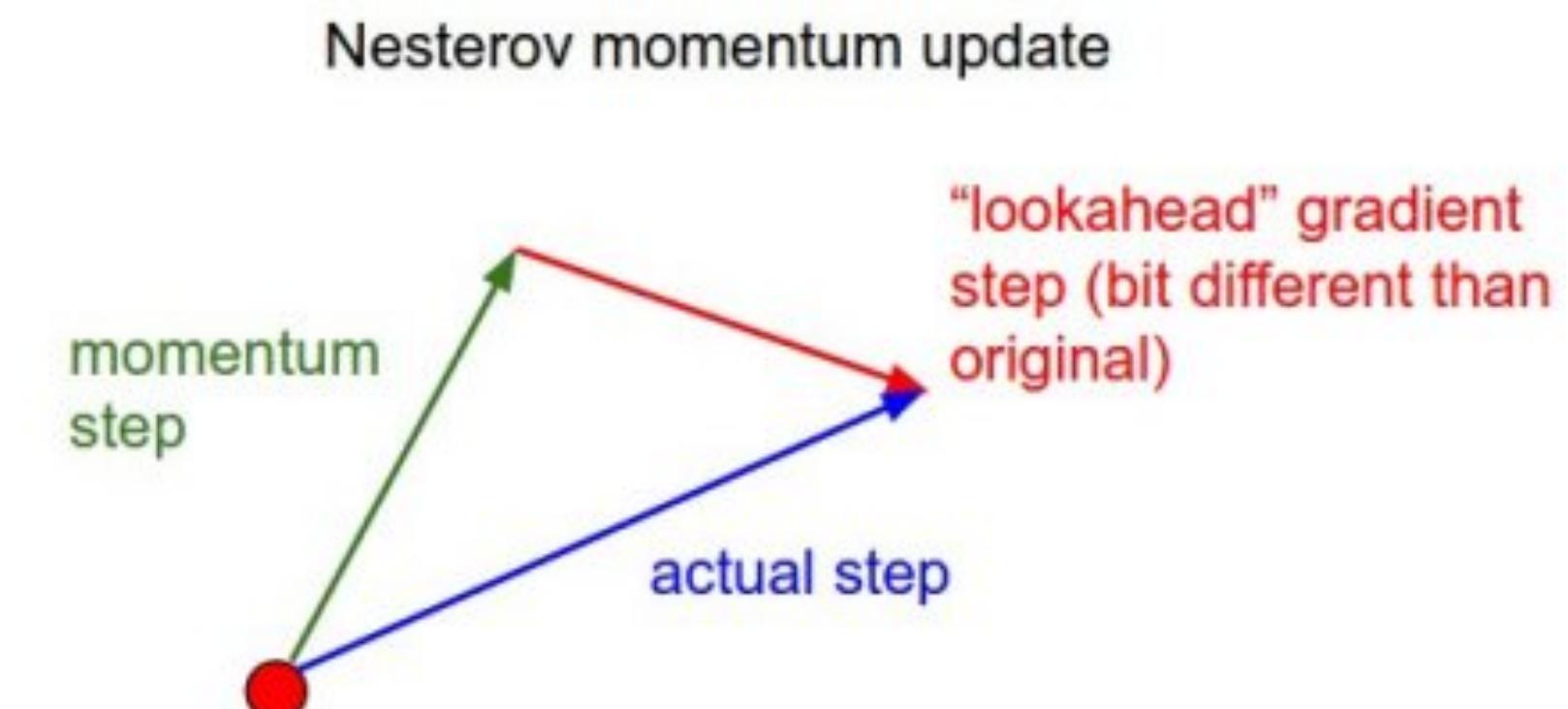
 Accumulate gradient: $r \leftarrow \rho r + (1 - \rho) \mathbf{g} \odot \mathbf{g}$

 Compute velocity update: $v \leftarrow \alpha v - \frac{\epsilon}{\sqrt{r}} \odot \mathbf{g}$. ($\frac{1}{\sqrt{r}}$ applied element-wise)

 Apply update: $\theta \leftarrow \theta + v$

end while

- Gradient 제곱의 합을 누적하여 Learning Rate 감소
- 오래된 Gradient 정보 제거
- Momentum을 이용한 파라미터 업데이트
- Momentum “Jump” 지점의 Gradient 정보 사용



→ Velocity (1st Momentum) 과 RMSProp (2nd Momentum)를
결합하니 성능이 향상되는 것을 확인…!

ADAM: Adaptive Momentum Optimizer

- 1st Momentum (Velocity)와 2nd Momentum (RMSProp)을 동시에 고려한 Adaptive Learning 알고리즘
- 기존 RMSProp과 Momentum method의 결합의 변형으로 이해할 수 있음
- 최근 많은 deep learning 모델의 학습 방법으로 채택하여 사용하는 추세

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$.
(Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default:
 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $s = 0, r = 0$

Initialize time step $t = 0$

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with
 corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
 $t \leftarrow t + 1$

 Update biased first moment estimate: $\hat{s} \leftarrow \rho_1 s + (1 - \rho_1) \mathbf{g}$

 Update biased second moment estimate: $\hat{r} \leftarrow \rho_2 r + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 Correct bias in first moment: $\hat{s} \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$

 Correct bias in second moment: $\hat{r} \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$

 Compute update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r} + \delta}}$ (operations applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

<Parameter Update Rule>

$$\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r} + \delta}}$$

1st Momentum

Adaptive Learning Rate (RMSProp)

- Momentum method로 업데이트를 하는데,
그 정도를 조절하는 것으로 이해 가능
- 시간이 갈수록, 각 momentum들의 값이 작아짐 (bias 조절)

ADAM: Adaptive Momentum Optimizer

- 1st Momentum (Velocity)와 2nd Momentum (RMSProp)을 동시에 고려한 Adaptive Learning 알고리즘
- 기존 RMSProp과 Momentum method의 결합의 변형으로 이해할 수 있음
- 최근 많은 deep learning 모델의 학습 방법으로 채택하여 사용하는 추세

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1]$.
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant δ used for numerical stabilization. (Suggested default:
 10^{-8})

Require: Initial parameters θ

Initialize 1st and 2nd moment variables $s = 0, r = 0$

Initialize time step $t = 0$

while stopping criterion not met do

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with
 corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

 Update biased first moment estimate: $\hat{s} \leftarrow \rho_1 s + (1 - \rho_1)g$

 Update biased second moment estimate: $\hat{r} \leftarrow \rho_2 r + (1 - \rho_2)g \odot g$

 Correct bias in first moment: $\hat{s} \leftarrow \frac{\hat{s}}{1 - \rho_1^t}$

 Correct bias in second moment: $\hat{r} \leftarrow \frac{\hat{r}}{1 - \rho_2^t}$

 Compute update: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (operations applied element-wise)

 Apply update: $\theta \leftarrow \theta + \Delta\theta$

end while

1. (batch size) m 개의 표본을 training set에서 추출.
2. Gradient $\mathbf{g} = \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ 계산.
3. Time step 증가: $t := t + 1$.
4. First moment 갱신: $s := \rho_1 s + (1 - \rho_1)g$ **(Momentum method)**
5. Second moment 갱신: $r := \rho_2 r + (1 - \rho_2)g \odot g$. **(RMSProp)**
6. 각 moment들의 bias 조절: $\hat{s} := \frac{s}{1 - \rho_1^t}, \hat{r} = \frac{r}{1 - \rho_2^t}$.
7. Parameter의 변화량 설정: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$
8. θ 값 갱신: $\theta := \theta + \Delta\theta$.

• Momentum method로 업데이트를 하는데,

그 정도를 조절하는 것으로 이해 가능

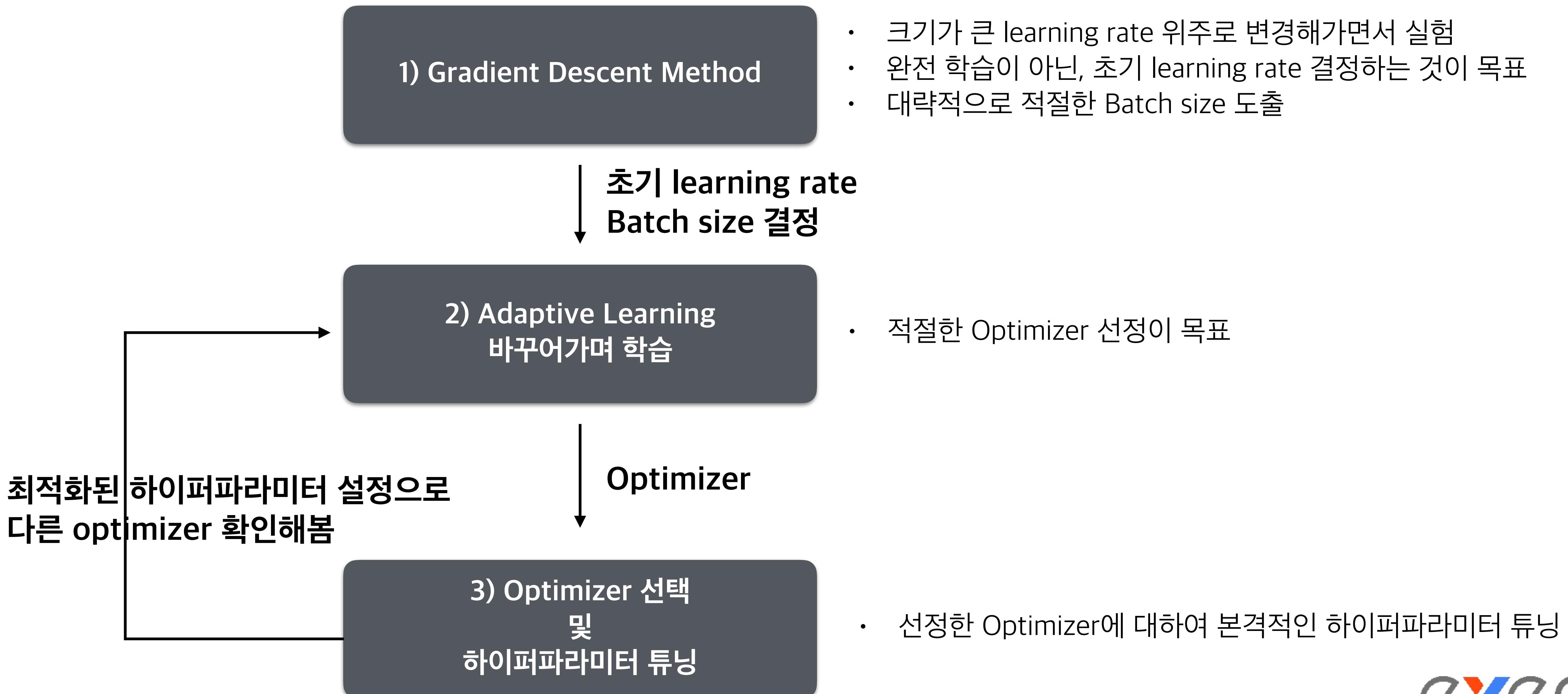
• 시간이 갈수록, 각 momentum들의 값이 작아짐 (bias 조절)

Choosing Proper Optimization Algorithm

- 모든 상황에서 좋은 성능을 보여주는 알고리즘은 없음 (No Free Lunch)
- 일반적으로 Mini-batch Learning과 함께 RMSProp, RMSProp with momentum, ADAM 등이 주로 쓰임
- 알고리즘 선택은 결국 연구자가 유동적으로 선택 (실험을 반복하며)
- 왜 이렇게 복잡한 Optimizer들을 이해하고 있어야하나? (그냥 가져다 쓰면 되는 것 아닌가?)
 - 실제로 사용할 때 한 번에 최적의 학습이 이뤄지지 않음...
 - 연구자는 학습이 잘 되지 않는 상황에서 그 원인을 정확하게 추론할 수 있어야함!
(ex. 단순히 “어떤 optimizer 쓰니 않돼네??” 가 아닌, “이 optimizer를 썼는데 이런 부분 때문에 안되는 것 같아....” 해당 원인이 정확하면 우리 상황에 맞게 optimizer 자체를 수정하거나 개발할 수도 있음)

Personal Experience (극도로 개인적임)

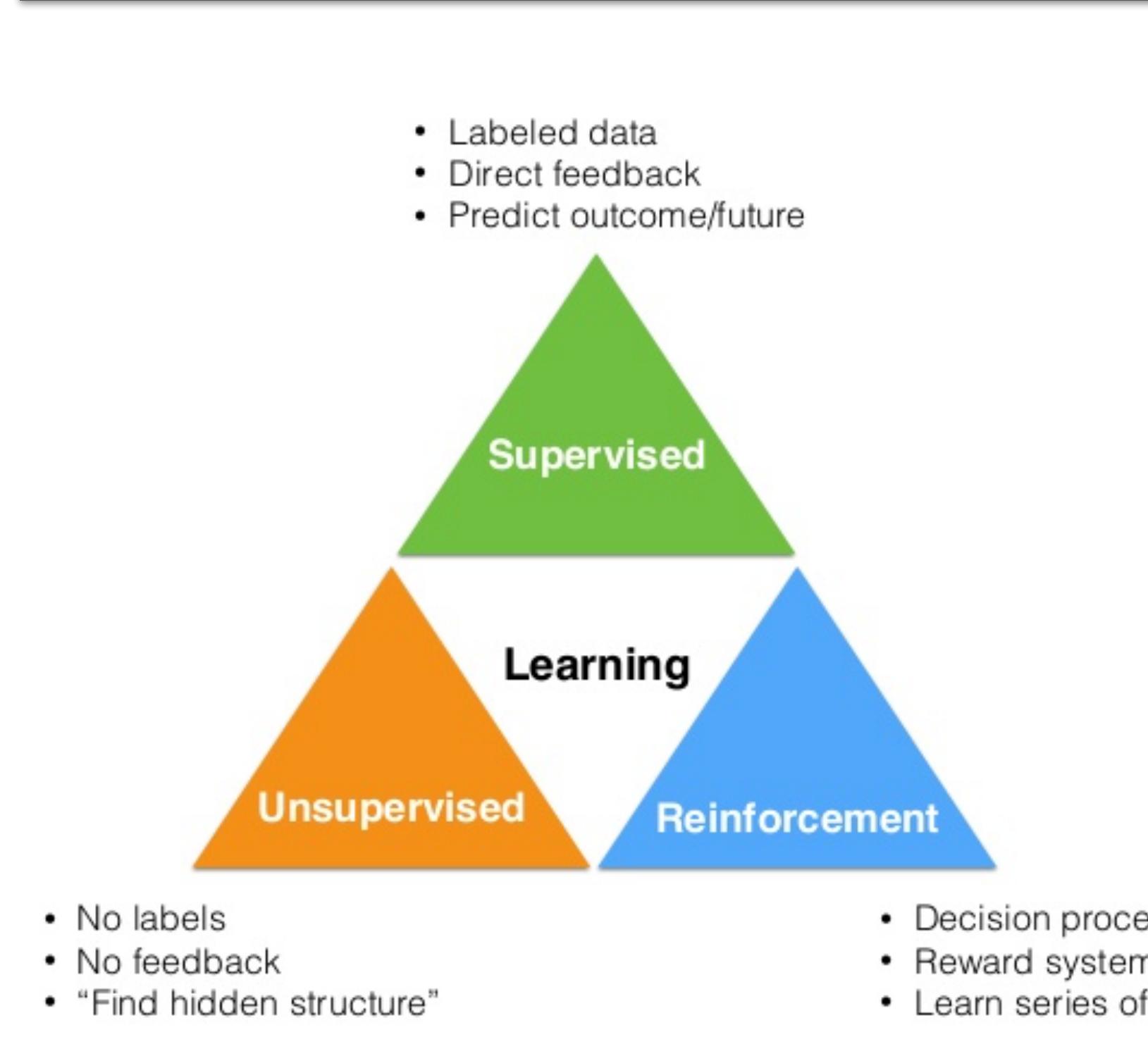
▣ 학습을 반복하여 실험해보며 개인적인 경험으로 취득한 optimizer 결정 방법



Types of Machine Learning

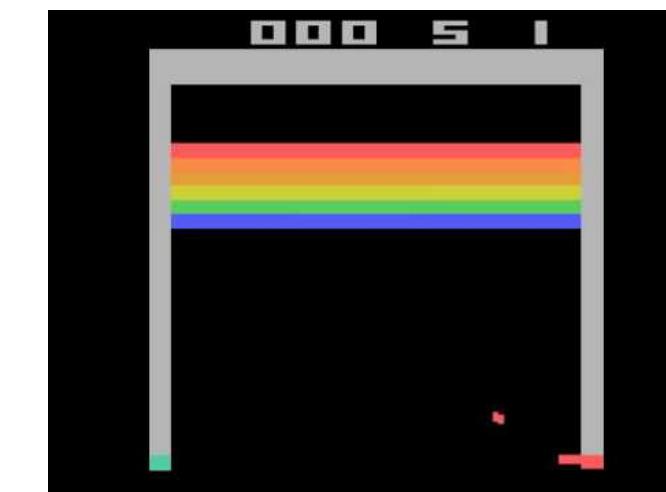
- ❑ 머신러닝은 다양한 기준으로 세부 분류가 가능함
- ❑ 학습 특성에 따른 분류: 1) Supervised, 2) Unsupervised, 3) Reinforcement Learnings
- ❑ 모델 특성에 따른 분류: 1) Geometric, 2) Probabilistic, 3) Logical Models

학습 특성에 따른 분류



<Reinforcement Learning>

- Policy, Reward, (Future) Value, Environment로 구성됨
- 미래의 기대 보상을 최대화하기 위한 행동 방침(policy)를 주위 환경과의 상호작용을 바탕으로 학습
- Different from other learning paradigms
 - Supervisor가 아닌, Reward Signal의 존재
 - Action에 대한 Feedback이 즉각적이지 않으며 지연됨(delayed)
 - Time is really matter (not i.i.d. data, but sequential)
 - Agent의 행동이 환경에 영향을 미침 (closed-loop)



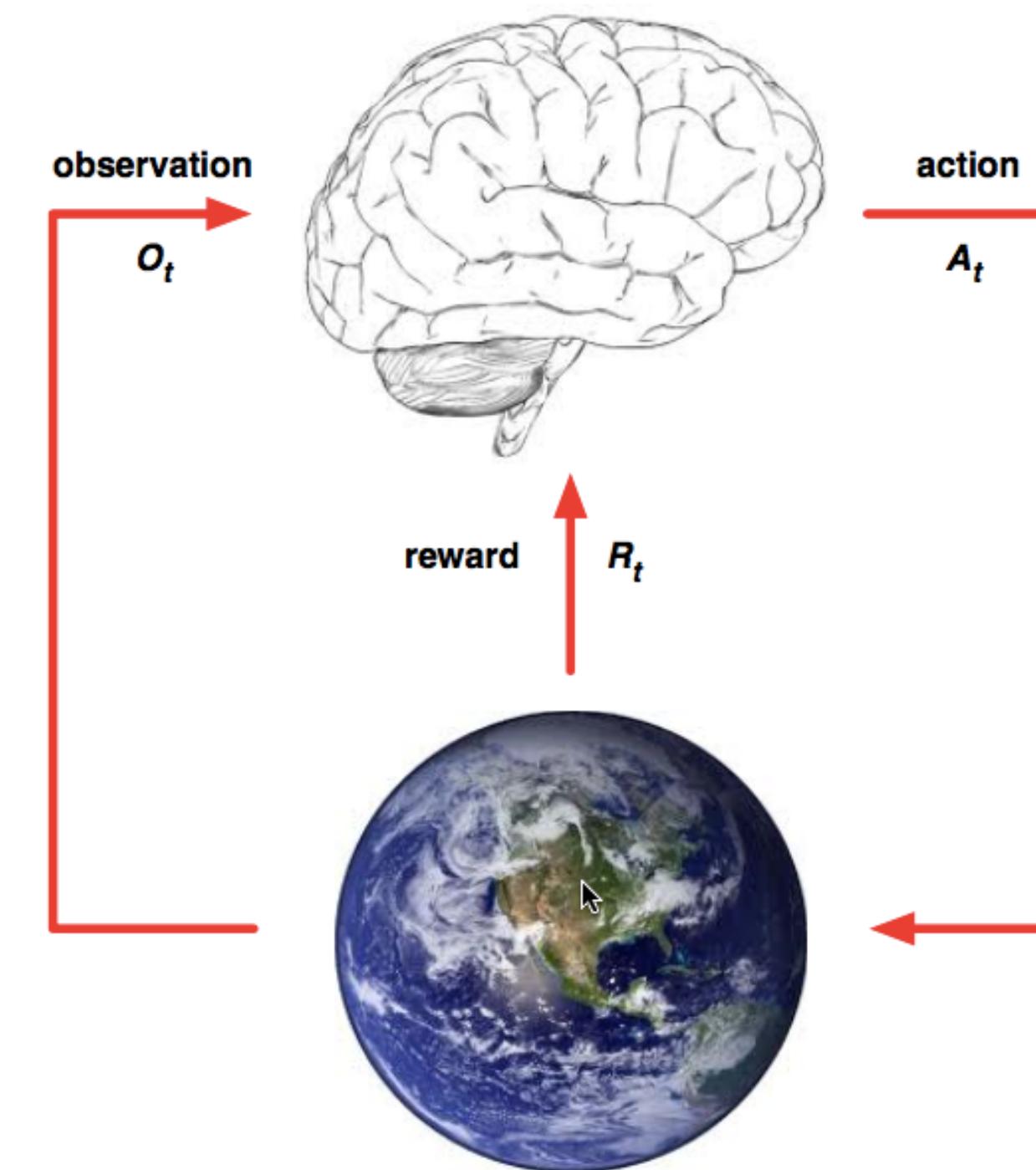
AlphaGo

What is Reinforcement Learning

□ Reinforcement Learning (강화학습)

- Numerical cumulative reward(value)를 최대화하기 위한 행동을 학습하는 방식과 관련한 머신러닝의 한 분야
- Agent의 Optimal Policy(행동 방침)을 구하는 것을 목적으로함
- 환경과의 상호작용(Interaction)을 통한 목적 지향적인(goal-directed) 학습방식
- “Reinforcement learning is defined not by characterizing learning models, but **by characterizing a learning problem.**” (Sutton)

<Reinforcement Learning System>



Agent

- 목적을 가지고 행동을 학습하는 주체 (What we involved in)
- 목적을 달성하기 위해 행동을 결정하고 Reward를 바탕으로 학습
- **At each time step t**
 - 1) 특정 행동 A_t 를 취하고
 - 2) Environment로부터 Reward R_t , Observation O_t 를 얻음

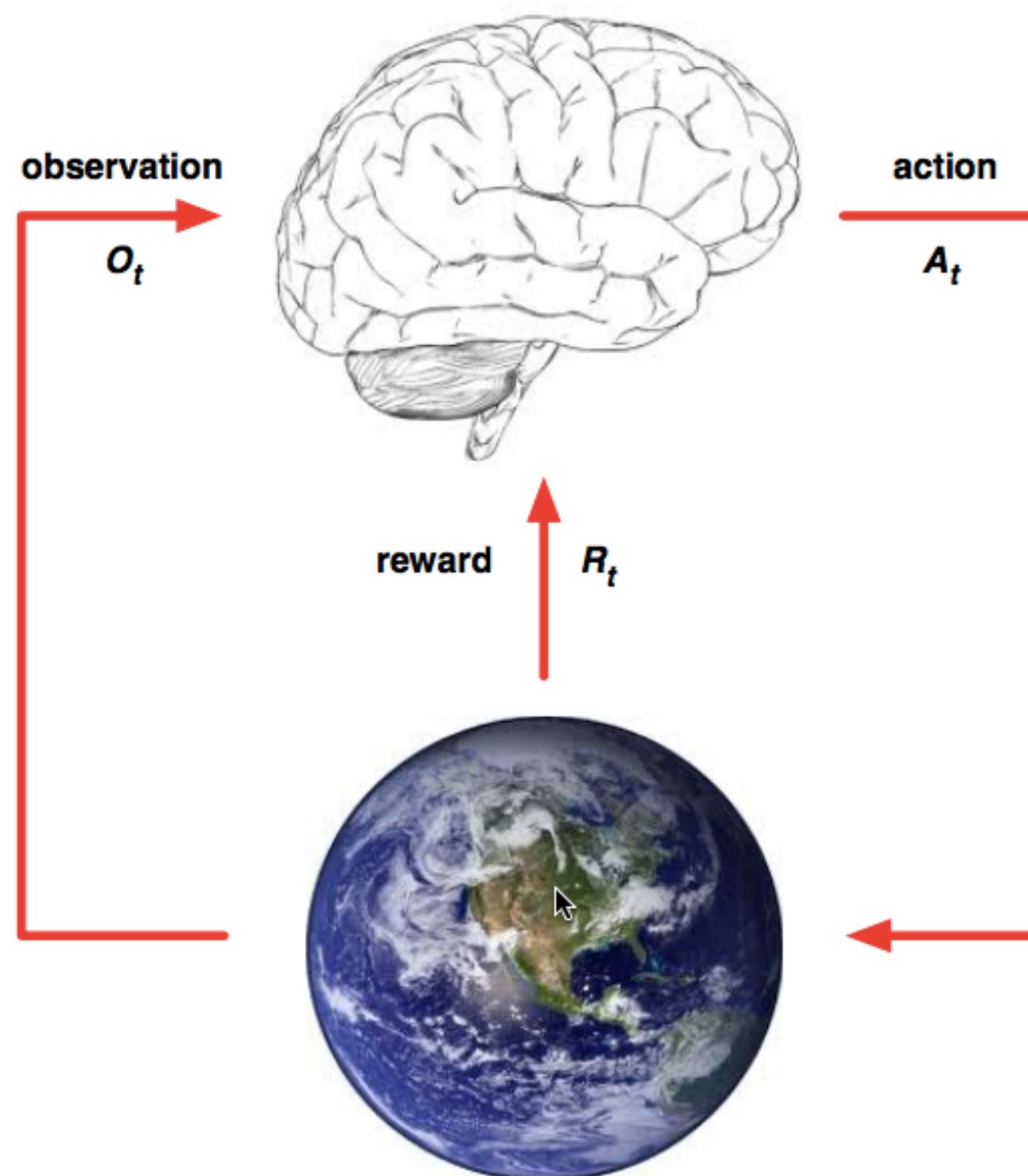
Environment

- Agent가 상호작용(Interaction)하는 대상
- Agent와 관련한 다양한 정보를 포함하는 시스템 전반을 의미
- **At each time step t**
 - 1) Agent로부터 특정 행동 A_t 를 인식하고
 - 2) Agent에게 Reward $R(t+1)$, Observation $O(t+1)$ 을 제공

What is Reinforcement Learning

- Agent와 Environment는 1) Action, 2) Reward, 3) Observation을 주고 받는 구조를 가지고 있음

<Reinforcement Learning System>



Action

Reward

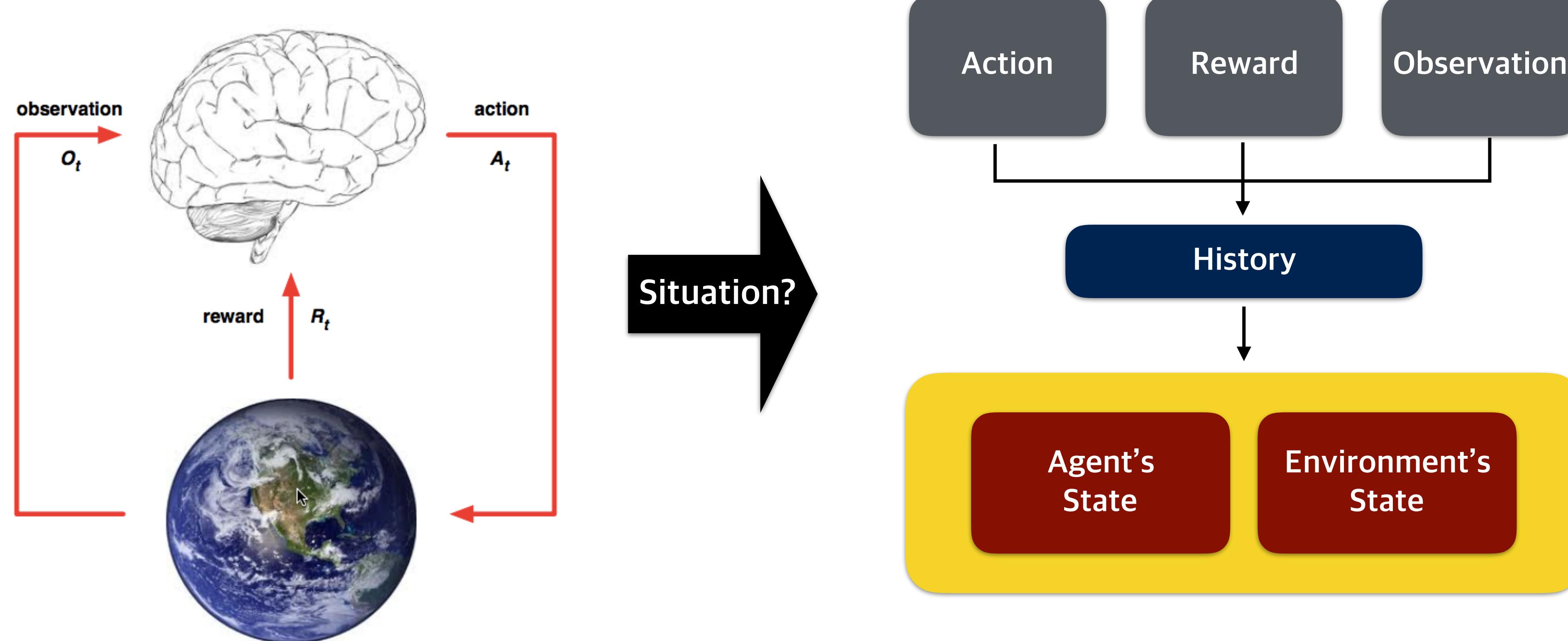
Observation

- 각 단계별로 Agent가 목적 달성을 위해 취한 **동작의 결과**
 - Agent의 Action은 Environment에 상호작용을 하며 영향을 주고, 이 후 **상황을 변화**시킴
 - Agent가 가지고 있는 **Policy**에 따라 Action을 취함
-
- Agent의 Action에 대한 Scalar Feedback.
 - Agent가 달성하고자 하는 목적은 각 단계의 Reward에 의해 정의됨
 - All goal can be described by the maximization of expected cumulative reward. (Reward Hypothesis)
-
- Agent의 Action에 의해 Environment로부터 제공되는 정보에 대한 종합적인 표현
 - Observation의 전체 또는 일부가 agent에게 제공되며, 이후 state를 결정

What is Reinforcement Learning

- 과거 Action, Reward, Observation의 모든 기록은 특정 시간의 History로 정의됨
- History는 과거의 정보를 바탕으로 Agent와 Environment의 상황(or State)을 정의하는 입력(Input)으로 작용함

<Reinforcement Learning System>

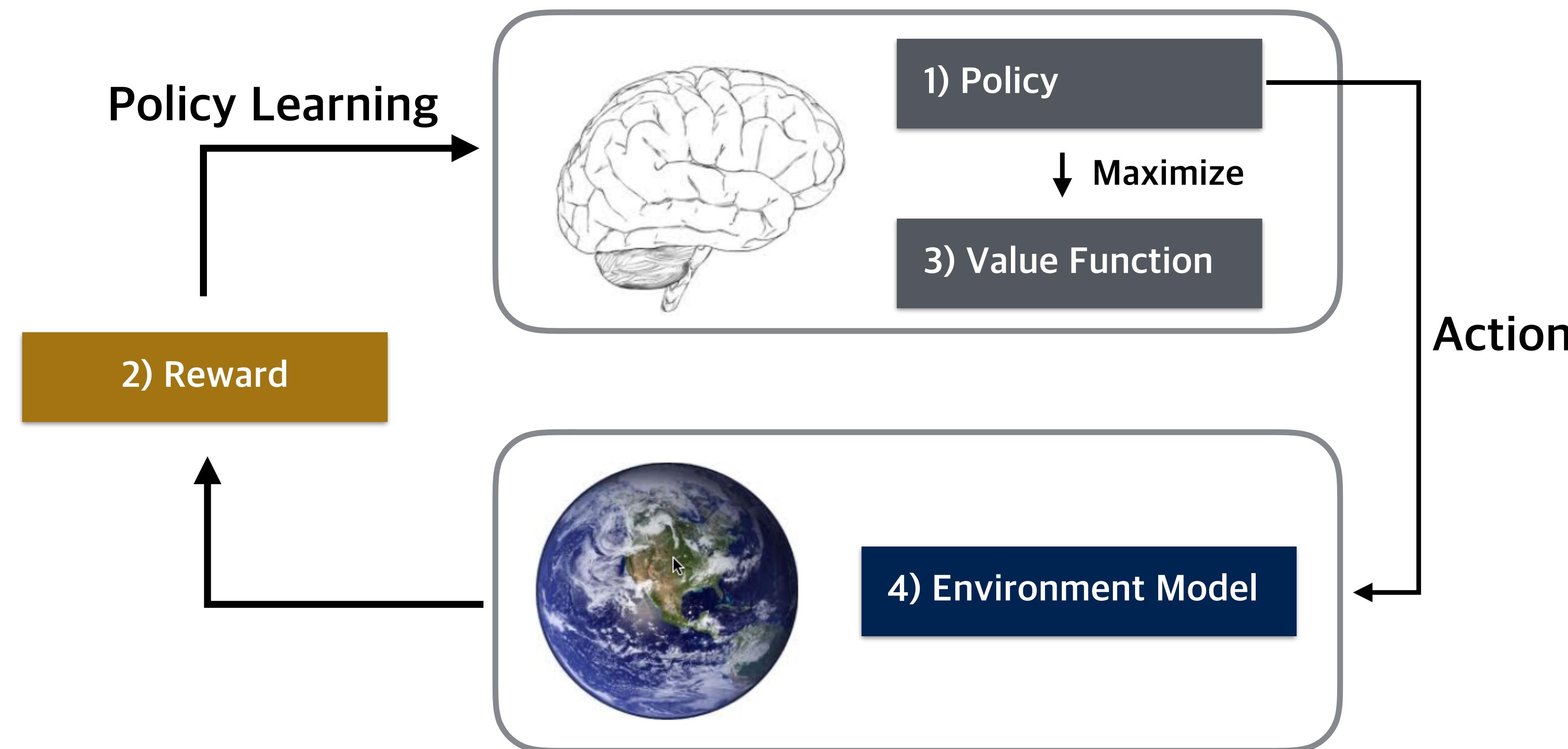


Elements of Reinforcement Learning

❑ Reinforcement Learning Model

: 1) Policy, 2) Reward Signal, 3) Value Function, 4) Environment Model (optional) 4가지로 구성됨

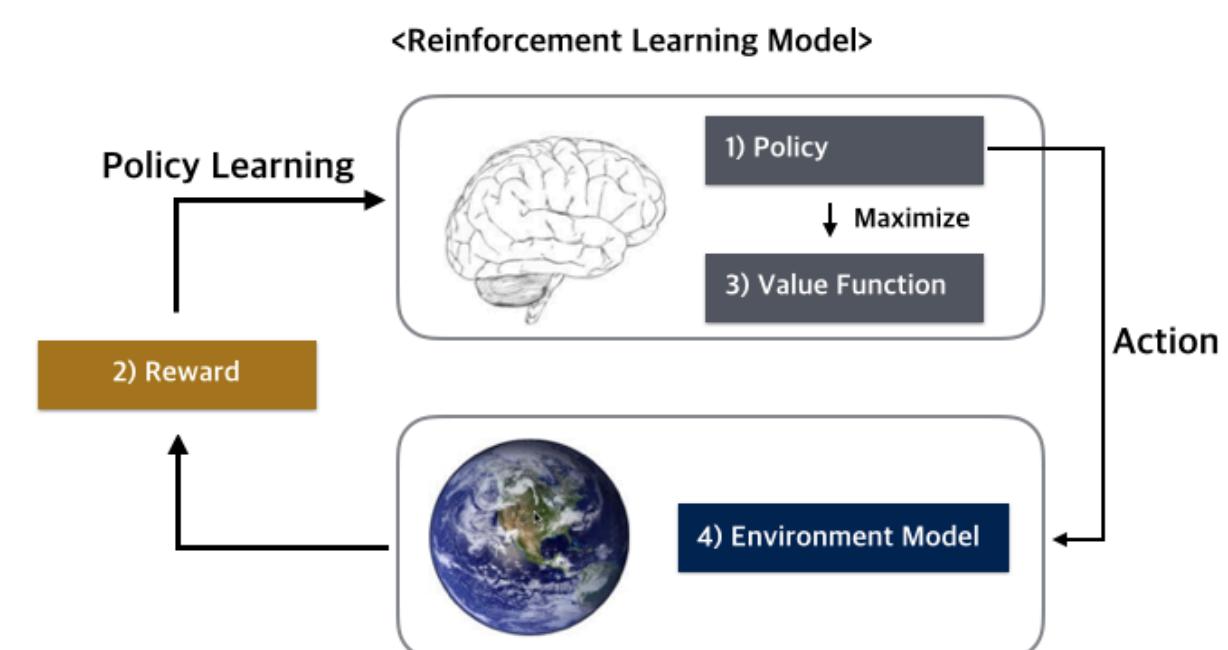
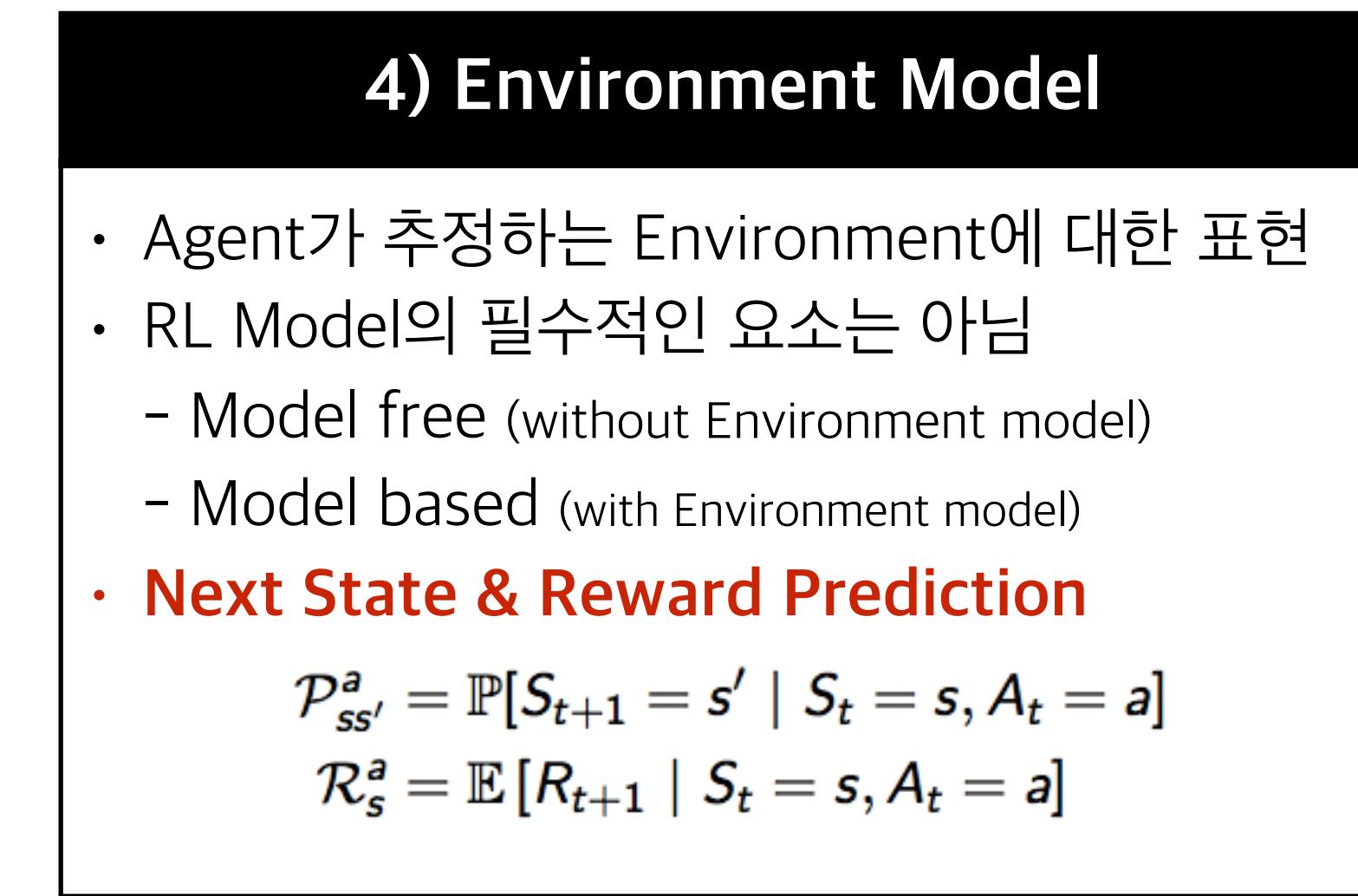
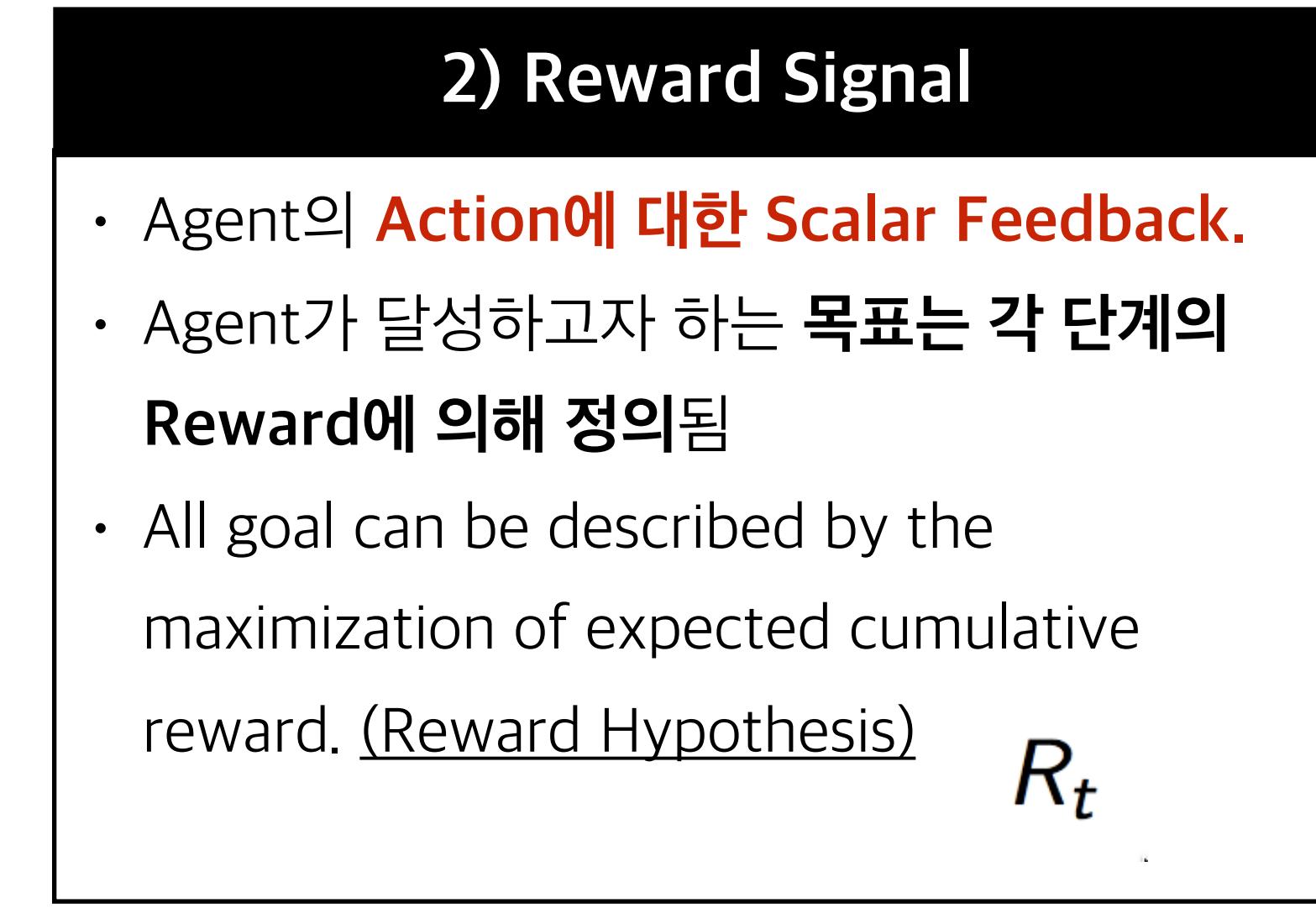
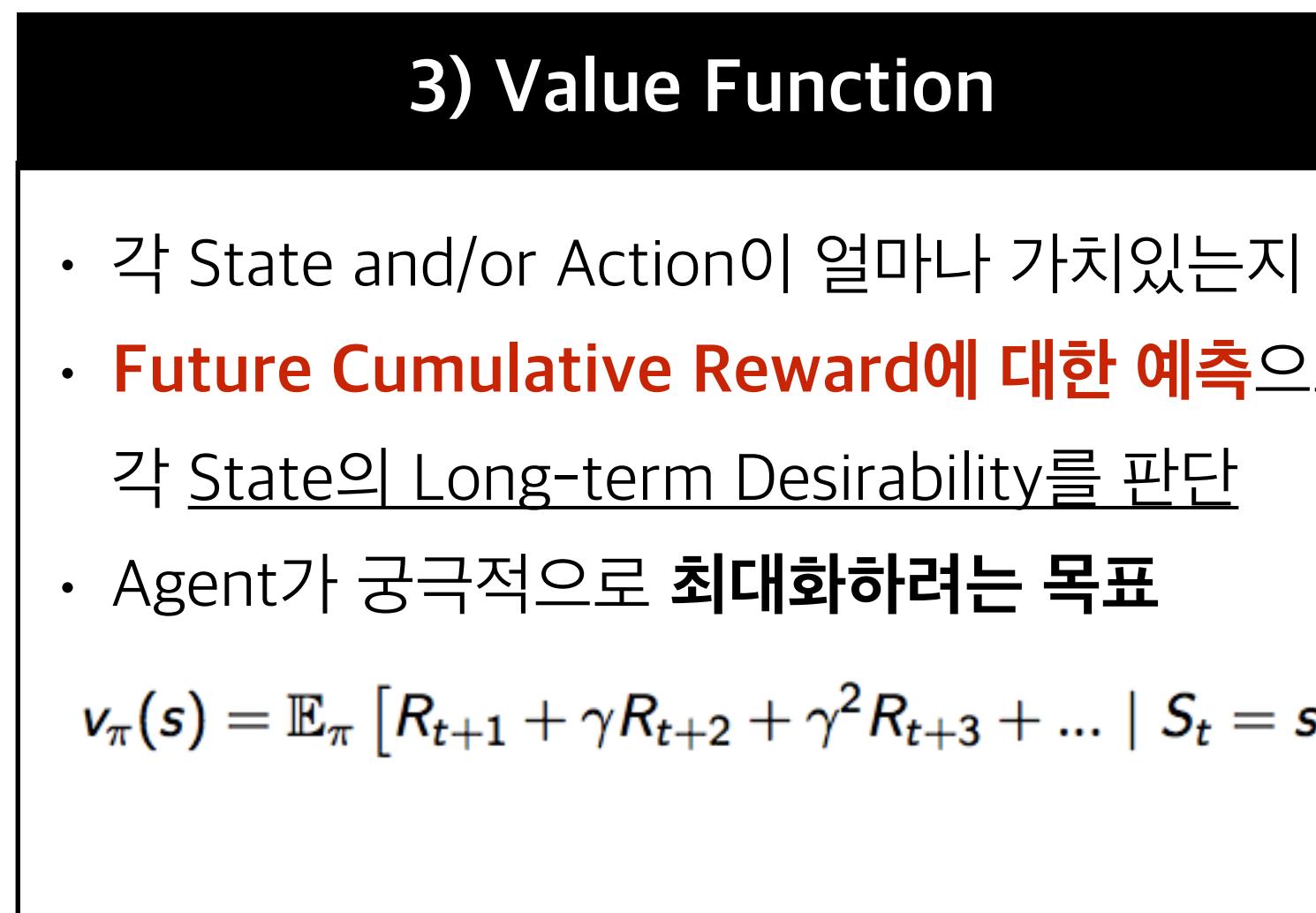
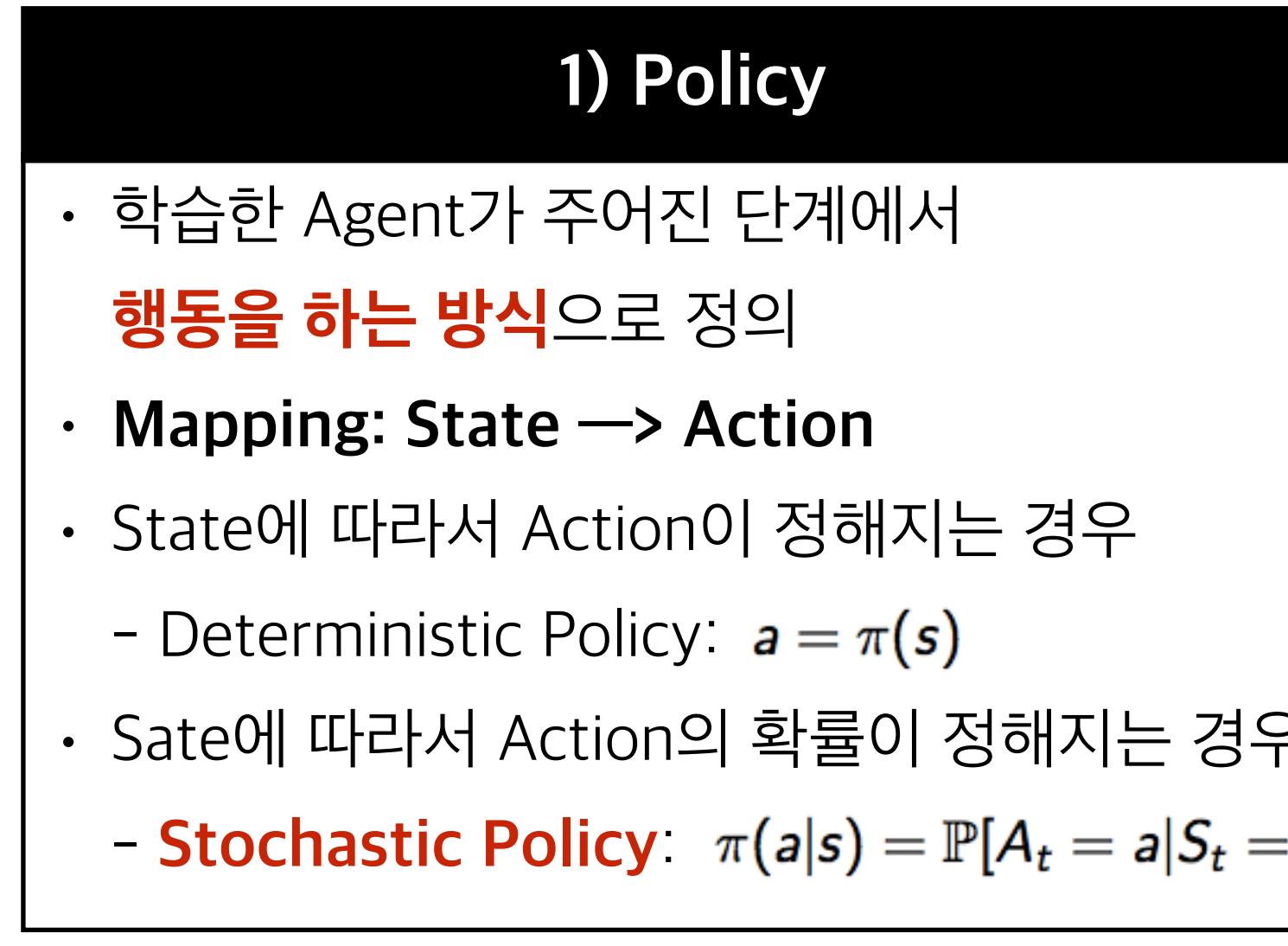
<Reinforcement Learning Model>



Elements of Reinforcement Learning

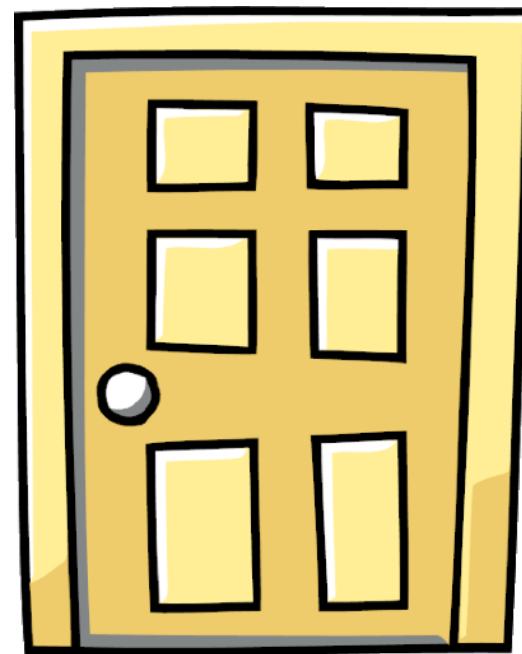
□ Reinforcement Learning Model

: 1) Policy, 2) Reward Signal, 3) Value Function, 4) Environment Model (optional) 4가지로 구성됨



Simple Example for Adopting Exploration

Robot Vacuum Cleaner Example



Takes 19% of Battery
in order to clean New Room
which **the robot has never gone**



30%

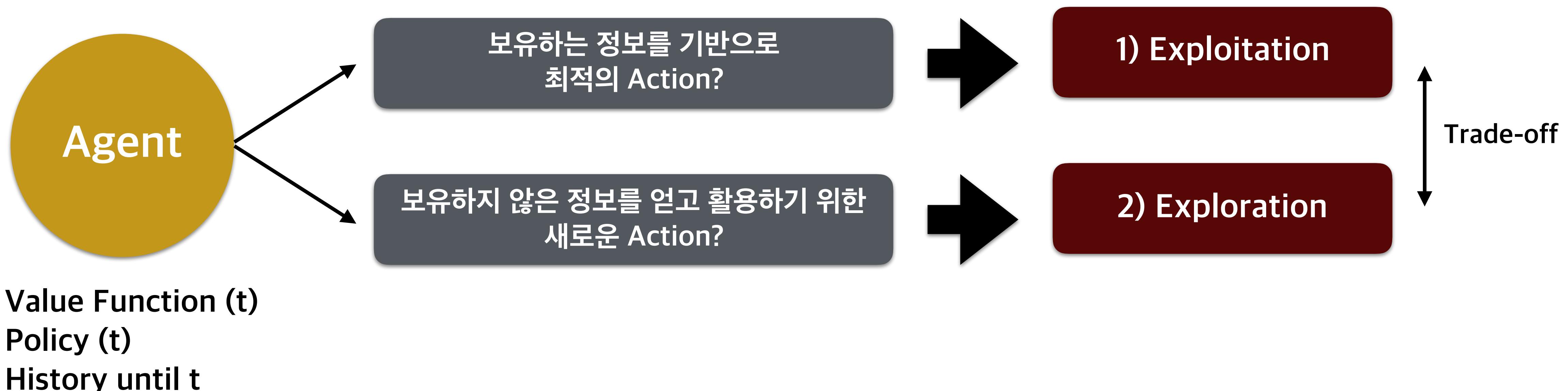


Takes 10% of Battery
from present location to outlet

How can the robot **decide to explore** the new room in order to optimize its action?

Exploration vs. Exploitation

- Agent가 가지고 있는 정보는 불완전/불확실(incomplete/uncertain)함을 기반으로 함
 - Agent는 문제에 대한 결정적인 솔루션(deterministic solution)을 가지고 있지 않음
 - History에 의해 정의된 State와 Value Function을 바탕으로 Policy를 학습해나감 >> Policy는 과거에 기반함
- 과거 History에 포함되지 않은 새로운 Action을 통한 정보의 습득은 미래의 가치를 최대화할 수 있음
- Agent는 Action 선택을 위해 보유 정보 이용과 새로운 정보 습득 사이의 Trade-off 문제에 직면함



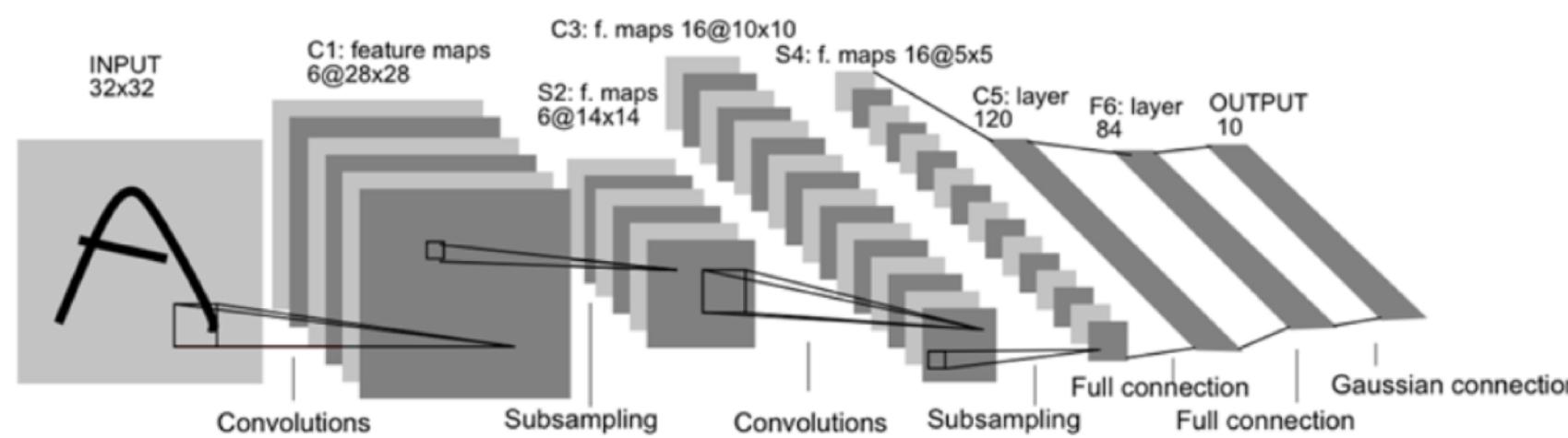
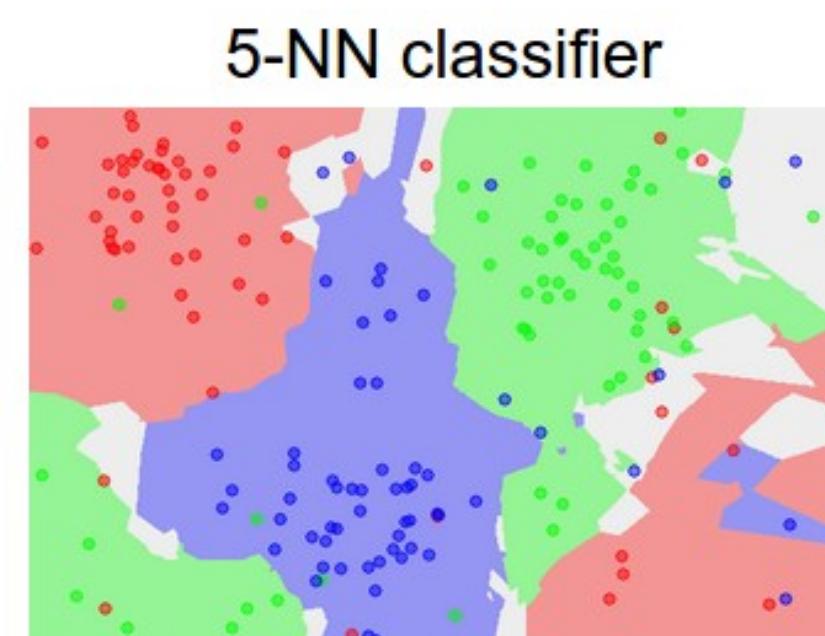
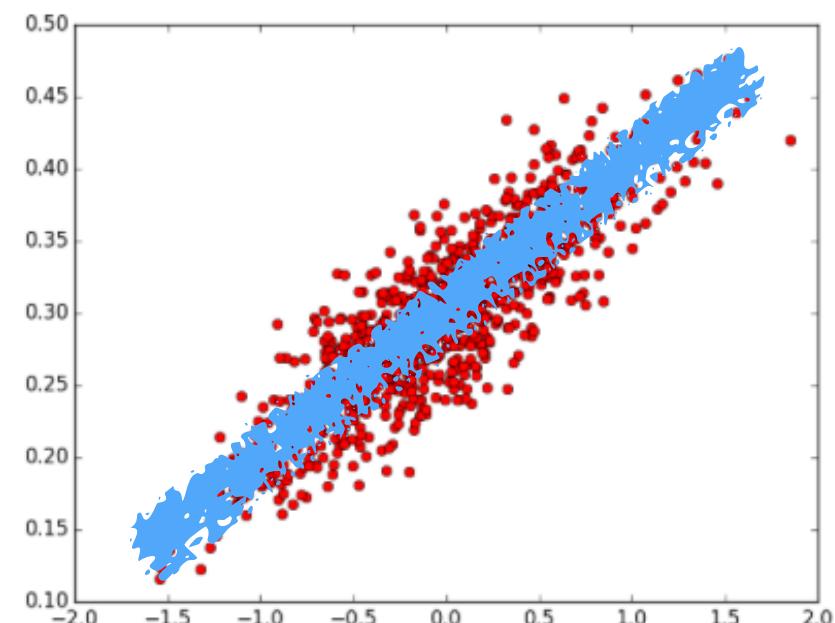
□ Q-Learning (deterministic)

GAN

Discriminative vs. Generative Model

- ❑ 머신러닝 모델은 목적에 따라 크게 분별 (Discriminative) 모델과 생성 (Generative) 모델로 분류할 수 있음
 - Discriminative model: 기존 관찰된 데이터를 분석하여, 새로운 데이터의 정보를 알아내는 것 (정답을 판별)
 - Generative model: 기존 관찰된 데이터를 분석하여, 새로운 데이터를 생성해내는 것

Discriminative Model



Generative Model

We didn't study yet...

- Gaussian Mixture Model (MoG)
- Hidden Markov Model
- Native Bayes
- Restricted Boltzmann Machine (RBM)
- Generative Adversarial Network

Boom !!!

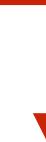
Generative Adversarial Networks

□ GAN (Generative Adversarial Network)

- 2014년 Goodfellow et al. 에 의해 처음 제안된 Generative Model
- 모델이 공개된 이 후, 세계의 폭발적인 관심과 함께 추가적인 연구와 이를 기반으로한 새로운 시도들이 나오고 있음

<GAN 이름의 해석>

Generative (뭔가를 생성하구나) + Adversarial (경쟁적인?) + Network (Neural Net을 의미하겠죠?)



무한 경쟁 사회에서 무엇을 경쟁시킨다는 걸까요?



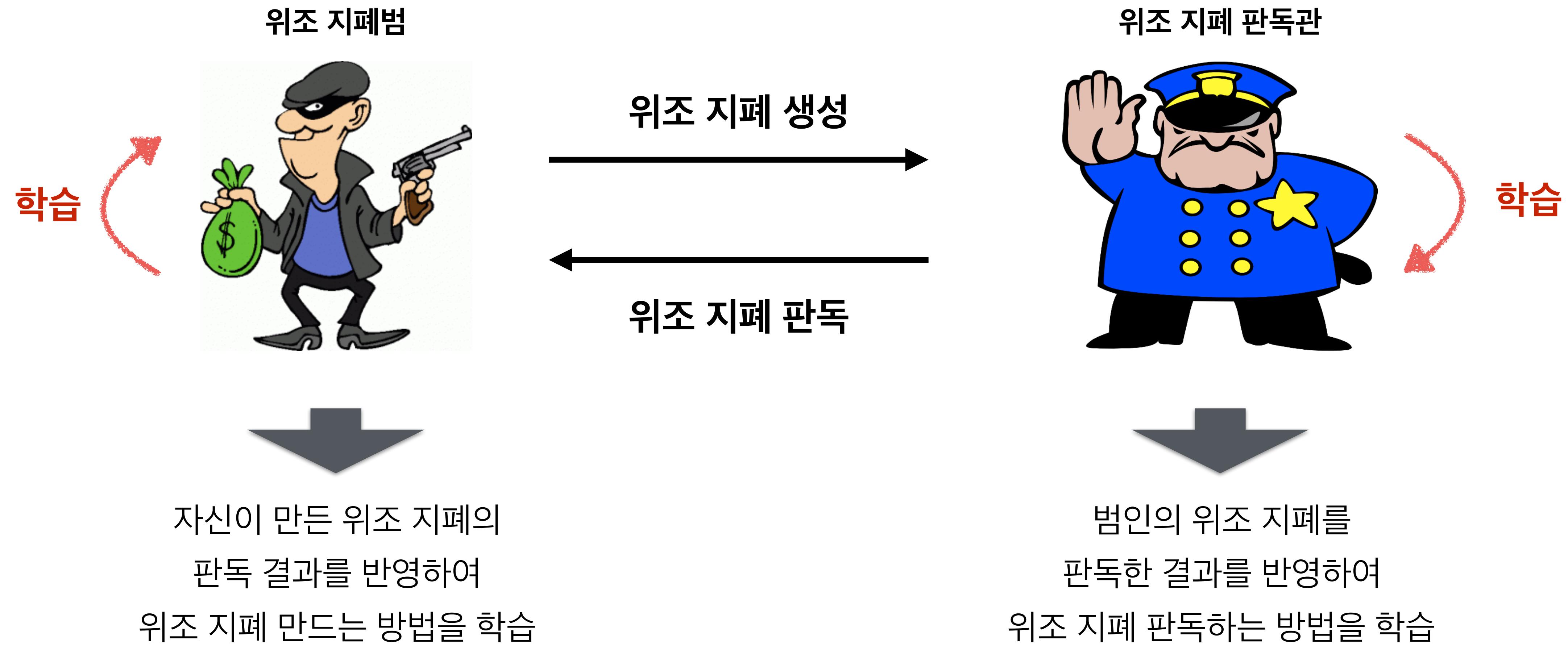
머신 러닝 모델은 기본적으로 모델의 결과를 바탕으로 학습을 하는 모델인데..



뭔가를 경쟁적으로 학습시키면서 뭔가를 만들어 내고 싶은거구나?

GAN Intuition

- GAN 모델의 아이디어는 매우 간단함: 위조 지폐범과 판독관의 예시



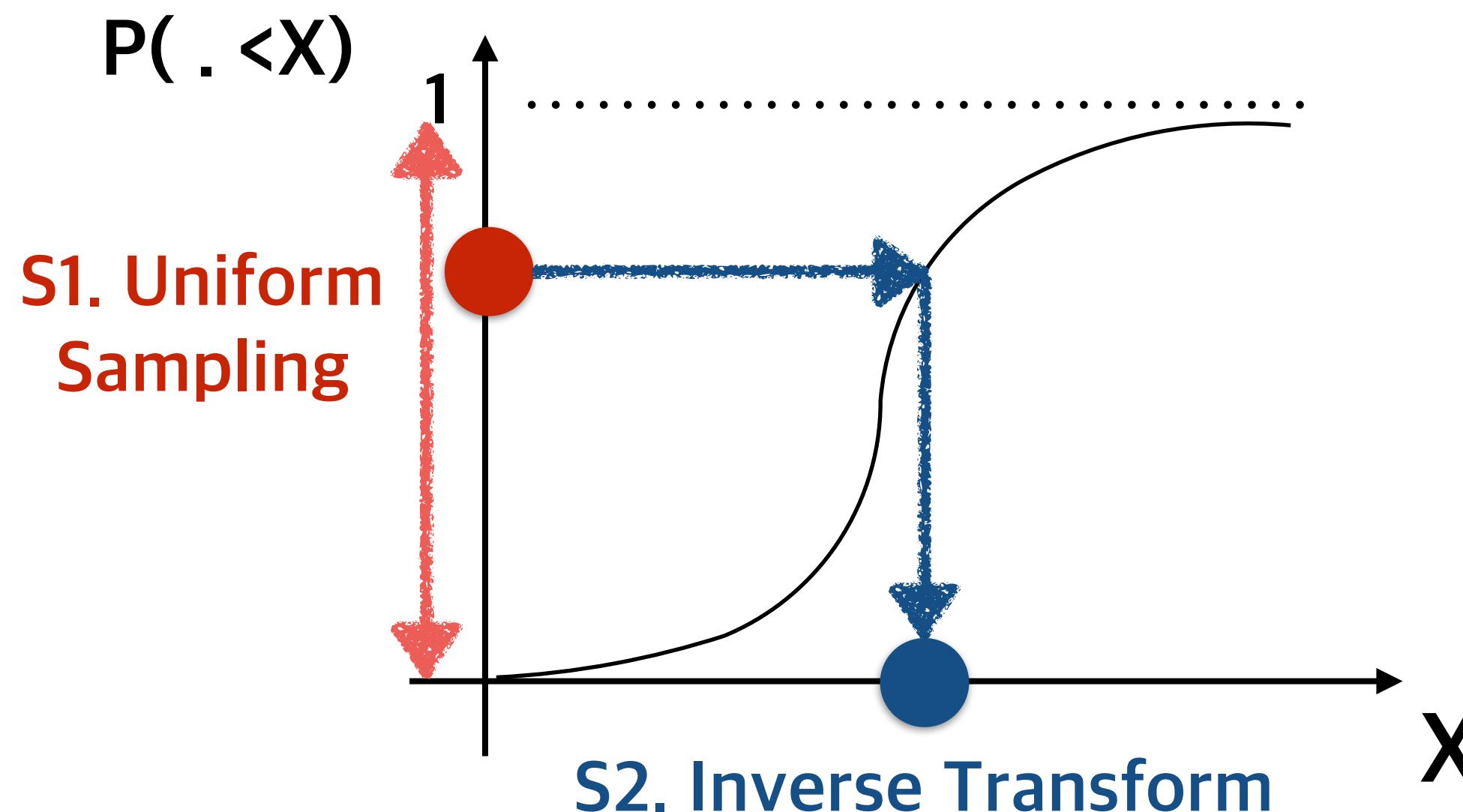
위조 지폐범 (Generator)와 판독관 (Discriminator) 모두 훌륭한 성능을 갖게 될 것으로 기대

What is Generative Process?

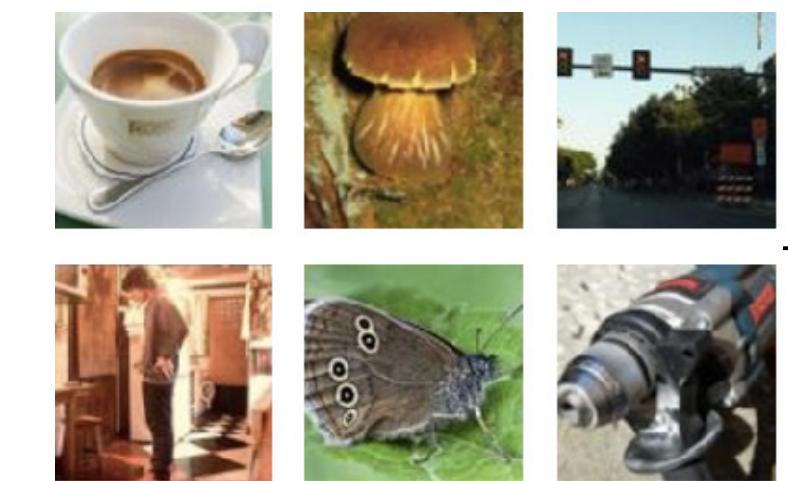
- ❑ 머신 러닝 모델이 무엇인가를 생성(generation)한다는 것의 의미는 생각보다 간단하며, 데이터의 분포와 관련이 있음
- ❑ 데이터의 누적 확률 분포 (Cumulative Probability Distribution, CDF)를 알고 있다면, 이를 기반으로 새로운 임의의 값을 생성할 수 있음 (Inverse Transform Sampling)

<Inverse Transform Sampling>

- 특정 확률 분포를 따르는 데이터를 임의로 생성하는 방법 중 하나
- 0~1 사이의 난수(random number)를 발생시킨 후, 특정 확률 분포의 CDF의 역함수를 통해 데이터를 생성함



우리가 오른 쪽과 같은
이미지들을 만들고 싶으면?



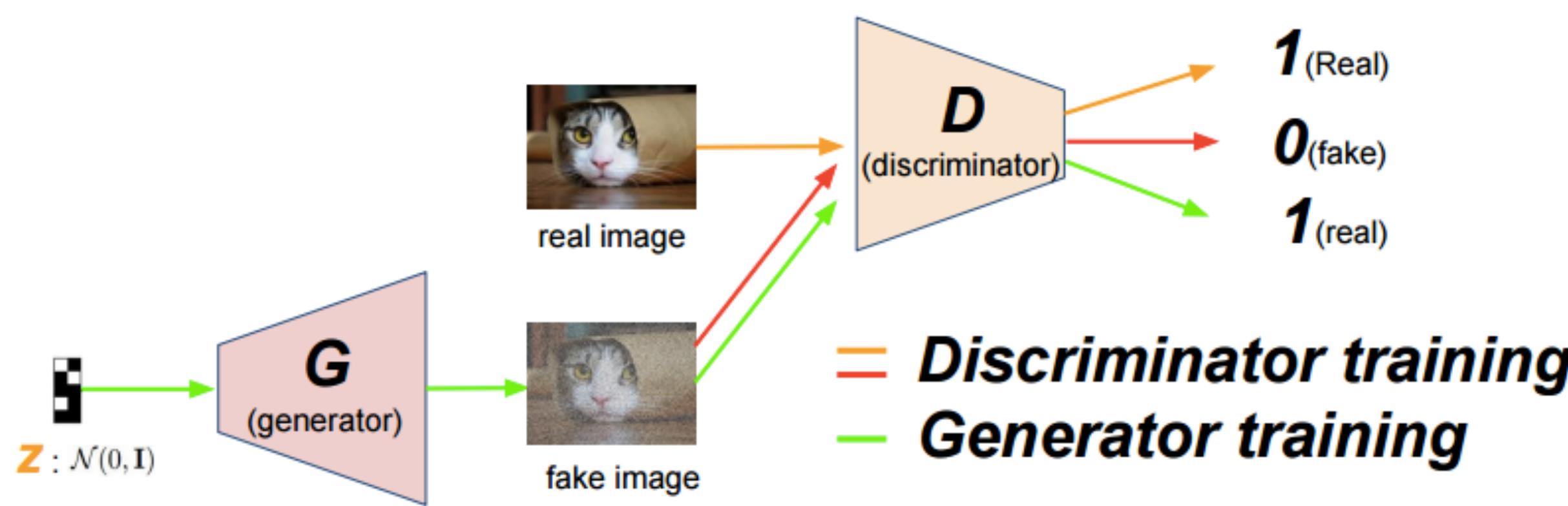
Random Number (Noise)

이미지들의 확률 분포
(모르지만, 일단 알고 있다고 가정.
모르면 학습하면 되니까요.)

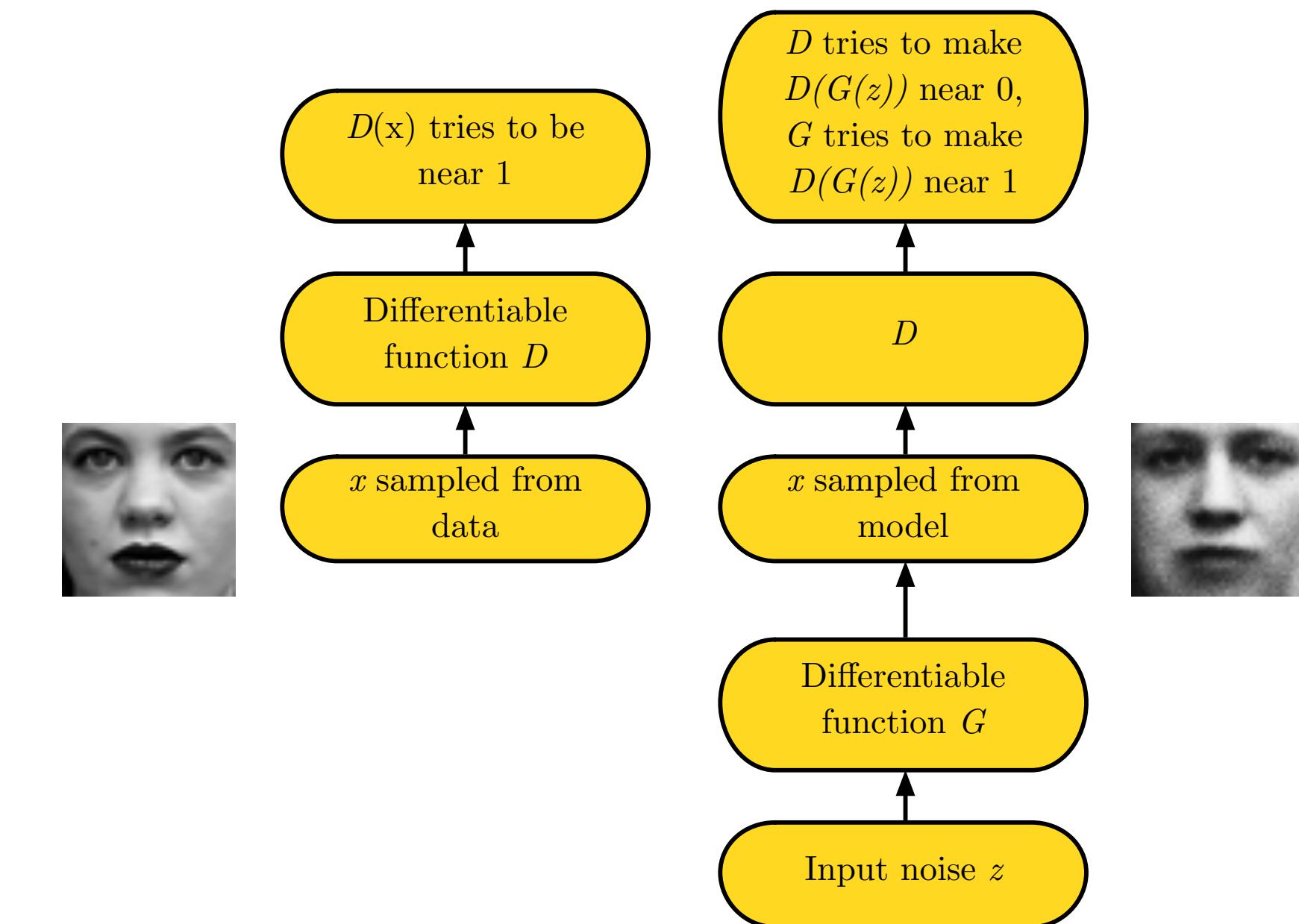
New Image !

Generative Adversarial Network Architecture

- Generator는 임의의 Random Noise로부터 가짜 이미지를 생성해냄
- Discriminator는 Generator가 생성한 이미지는 가짜로, 실제 이미지는 진짜로 판별하도록 설정되어 있음
- 위의 두 가지 프로세스가 진행된 후, Discriminator의 판정 결과에 따라 각각의 모델을 수정하는 것을 반복함



Adversarial Nets Framework



(Goodfellow 2016)

What is the Loss Functions?

eXem

Loss Functions of GAN

- GAN에서 Discriminator는 Real/Fake를 분류하는 binary-classifier로 볼 수 있음: Logistic Regression과 유사

$$\left[-\frac{1}{m} \sum_{i=1}^m [y_i \log(p_i) + (1-y_i) \log(1-p_i)] \right]$$

positive class일 때 negative class일 때

- GAN에서 Loss Function은 Minimax Game (zero-sum)에 기반하여 만들어짐 (수식에 겁먹지 말고 의미를 해석하기)

Total

$$\min_G \max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1-D(G(z)))]$$

Generator와 Discriminator의 학습을 동시에 표현한 식
(어려우면 아래 따로 따로 구성된 식을 보기)

Discriminator

$$\max_D V(D, G) = E_{x \sim p_{data}(x)}[\log D(x)] + E_{z \sim p_z(z)}[\log(1-D(G(z)))]$$

진짜 사진을 분별할 확률을 최대화하고 가짜 사진을 진짜라고 말할 확률을 최소화하자

Generator

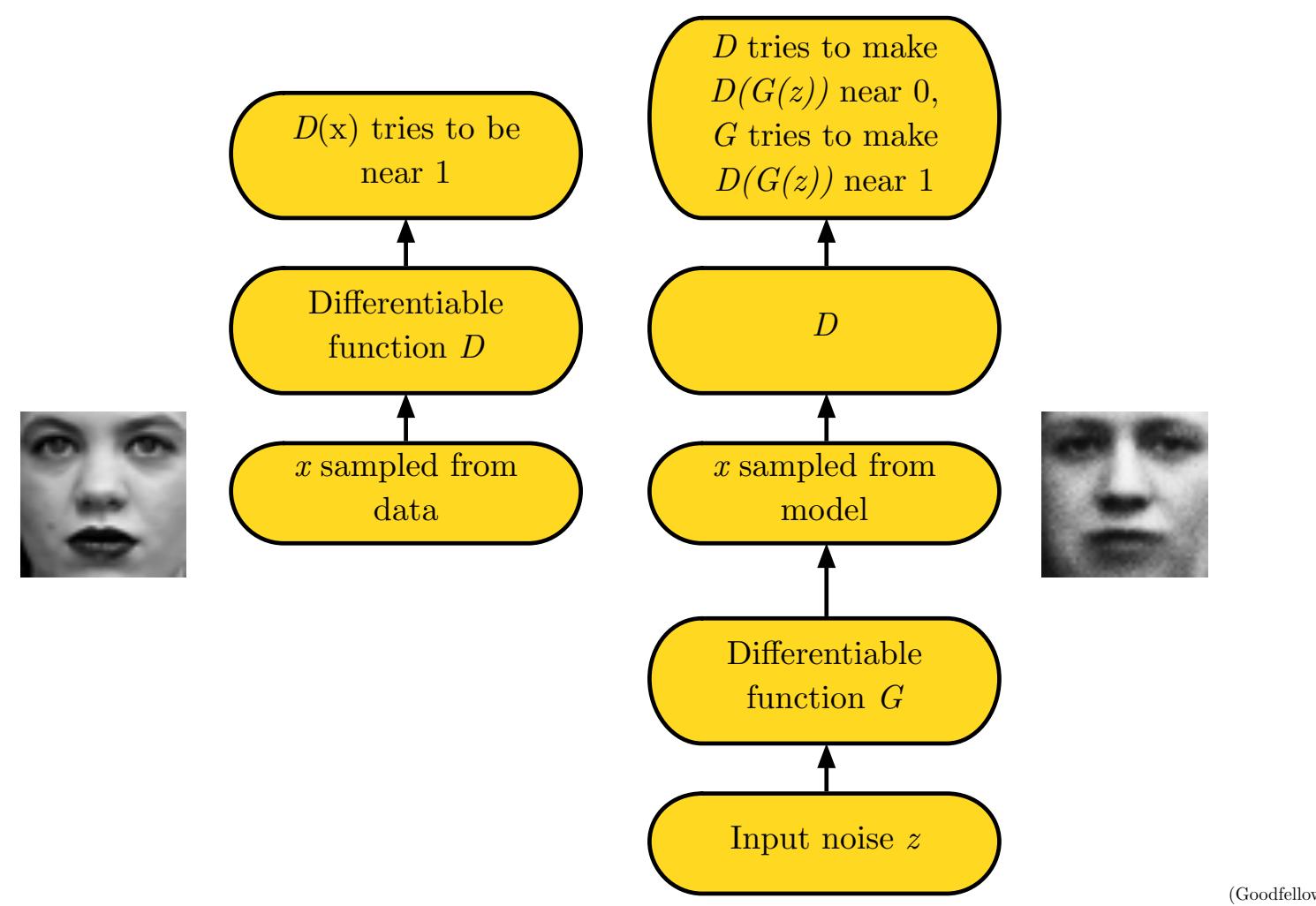
$$\min_G V(D, G) = E_{z \sim p_z(z)}[\log(1-D(G(z)))] \simeq \max_G E_{z \sim p_z(z)}[\log(D(G(z)))]$$

가짜 사진을 진짜라고 말할 확률을 최대화하자

GAN Learning Algorithm Detail

□ GAN에서 Discriminator는 Real/Fake를 분류하는 binary-classifier로 볼 수 있음: Logistic Regression과 유사

Adversarial Nets Framework



Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k , is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for  $k$  steps do
        • Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
        • Sample minibatch of  $m$  examples  $\{x^{(1)}, \dots, x^{(m)}\}$  from data generating distribution  $p_{\text{data}}(x)$ .
        • Update the discriminator by ascending its stochastic gradient:
```

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D(x^{(i)}) + \log (1 - D(G(z^{(i)}))) \right].$$

```
end for
• Sample minibatch of  $m$  noise samples  $\{z^{(1)}, \dots, z^{(m)}\}$  from noise prior  $p_g(z)$ .
• Update the generator by descending its stochastic gradient:
```

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log (1 - D(G(z^{(i)}))).$$

```
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

G 고정,
D 업데이트

D 고정,
G 업데이트

Results Example

- GAN을 이용하여 새롭게 만들어낸 이미지는 아래와 같음

GAN이 만든 이미지

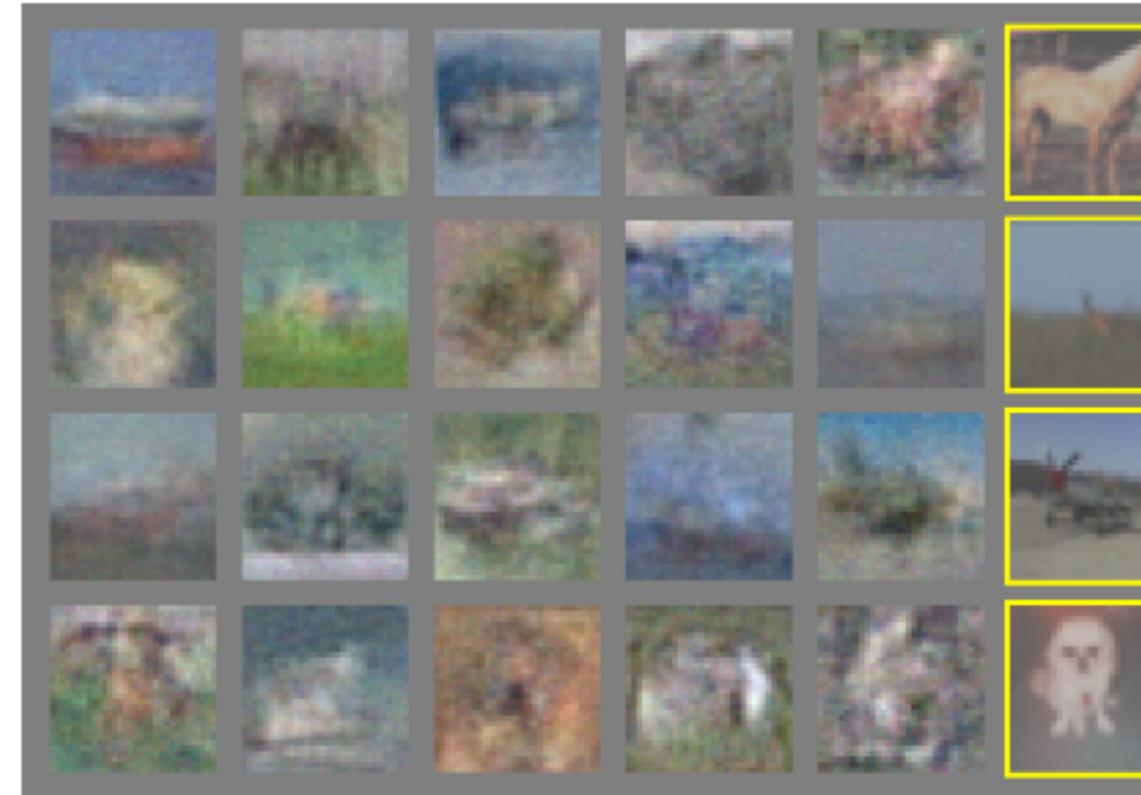
1	3	9	3	9	9
1	1	0	6	0	0
0	1	9	1	2	2
6	3	2	0	8	8

a)

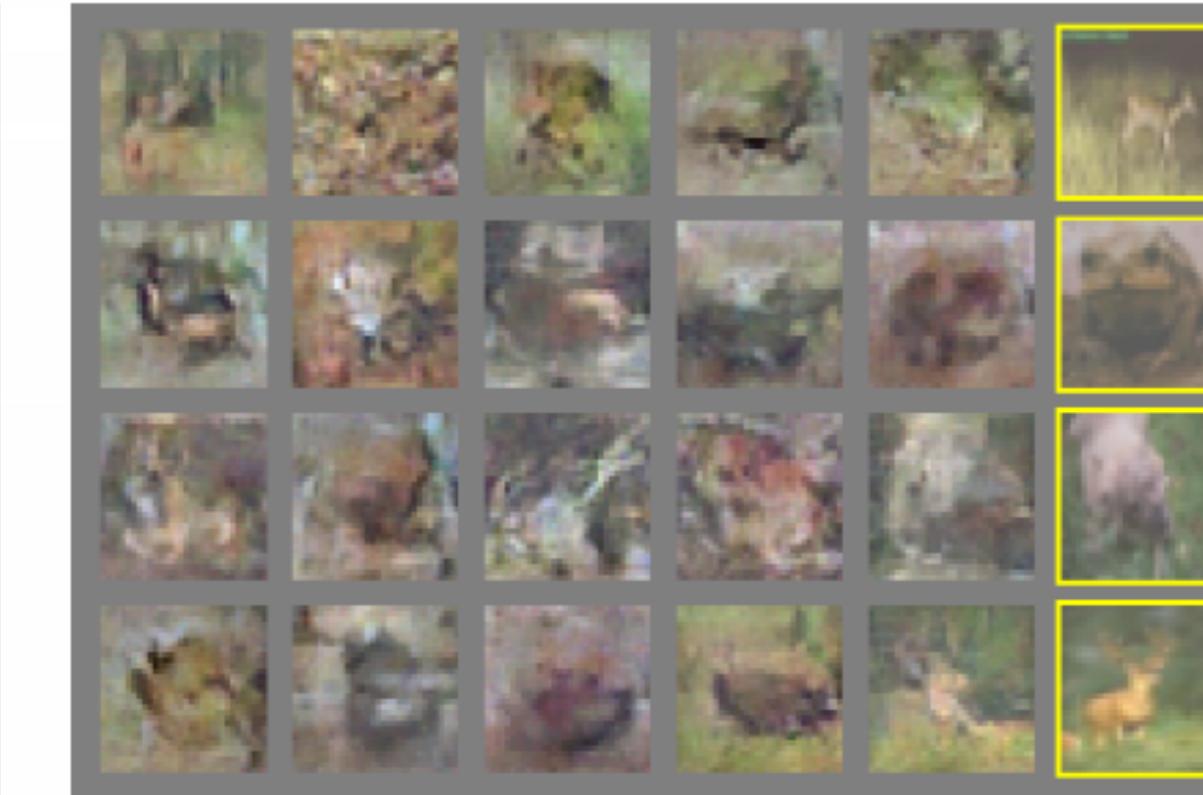
실제 데이터 중 GAN이 만든 이미지와 가장 비슷한 이미지



b)



c)



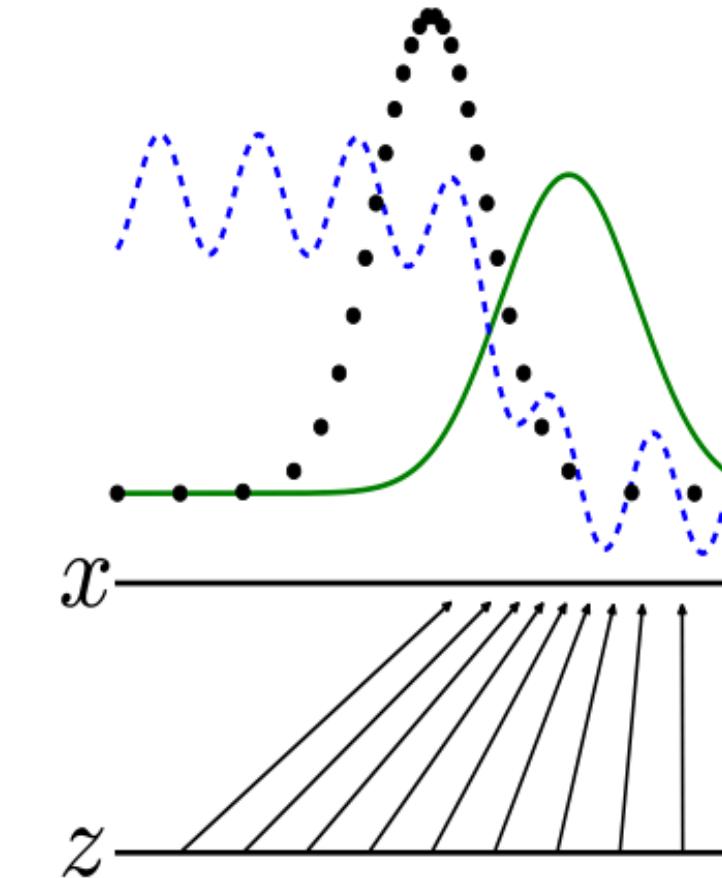
d)

Distribution Learning of GAN

■ GAN의 반복적인 학습은 결국 생성하고자 하는 이미지의 확률 분포를 학습하는 것과 같음

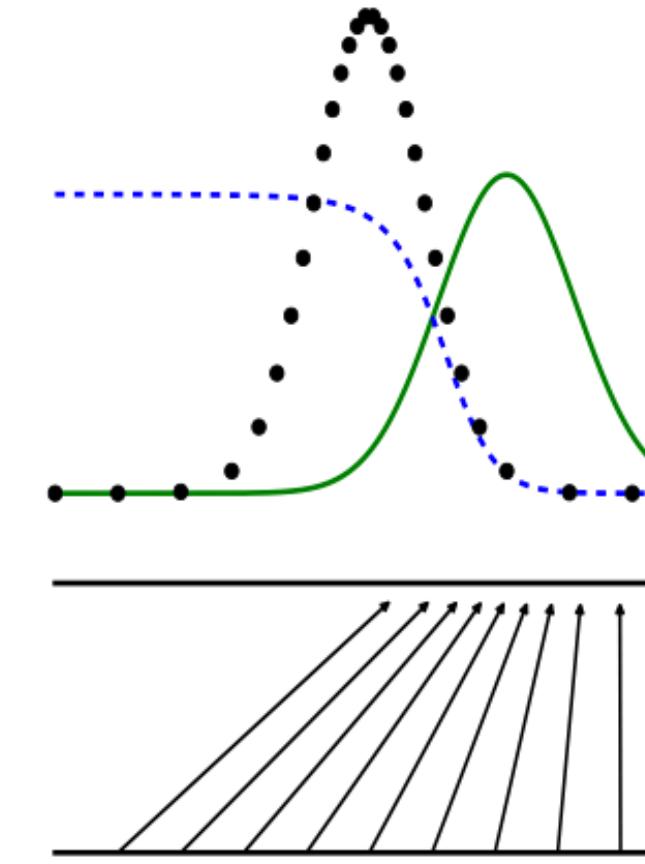
Blue dotted: D
Green solid: G
Black dots: Real samples

G와 Pdata는 꽤나 비슷하다
초록색 G가 약간 오른쪽으로
치우쳐있다.



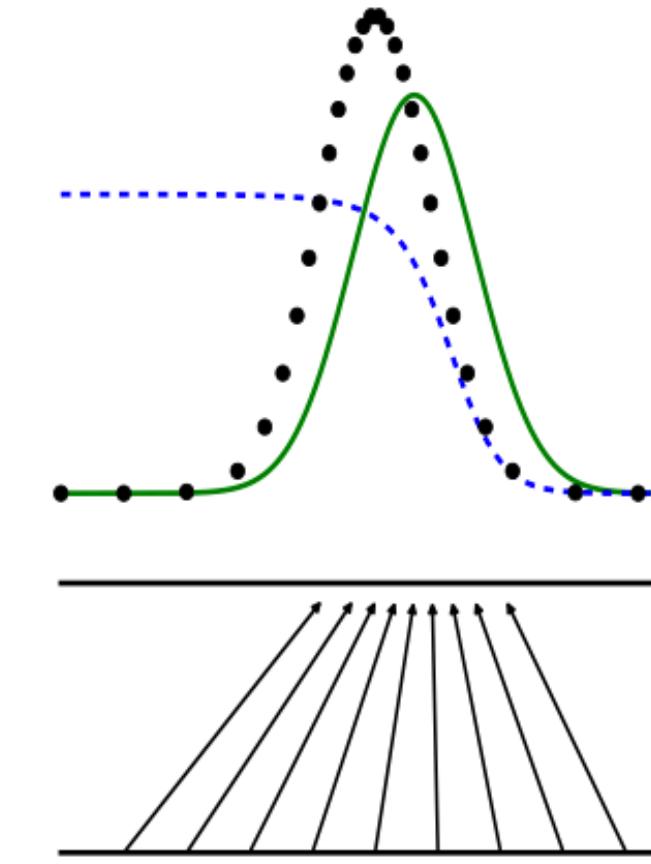
(a)

D는 Pg와 Pdata를 구분하기 위해서 왼쪽에 높은 값을 할당하게 된다.



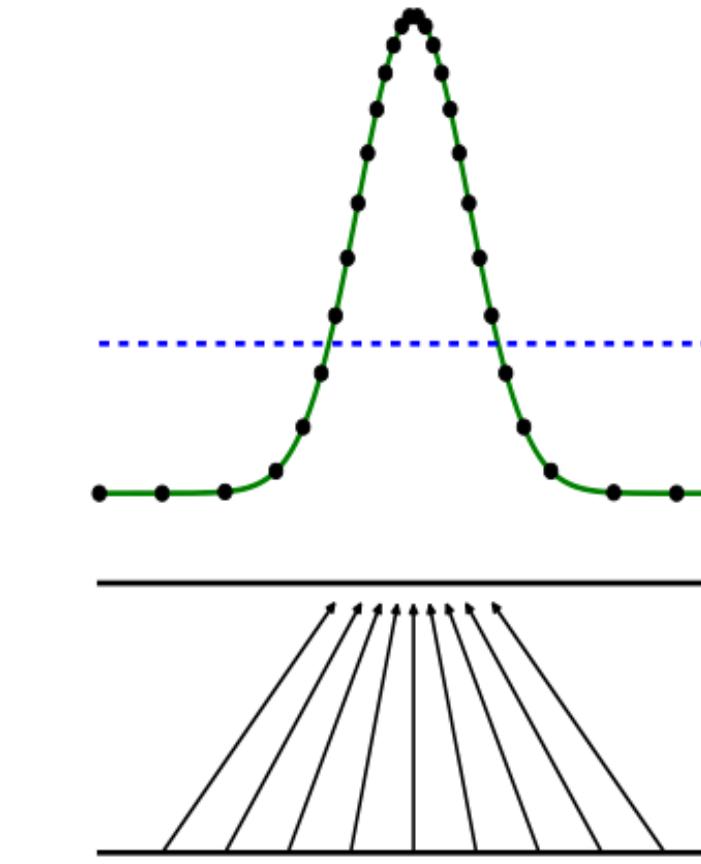
(b)

G는 D가 높아진 왼쪽 영역에 많은 mass를 가하게 되고, 자연스레 mode가 왼쪽으로 밀려간다



(c)

이를 반복하면 결국 Pg와 Pdata가 같아지게 된다. (hopefully)



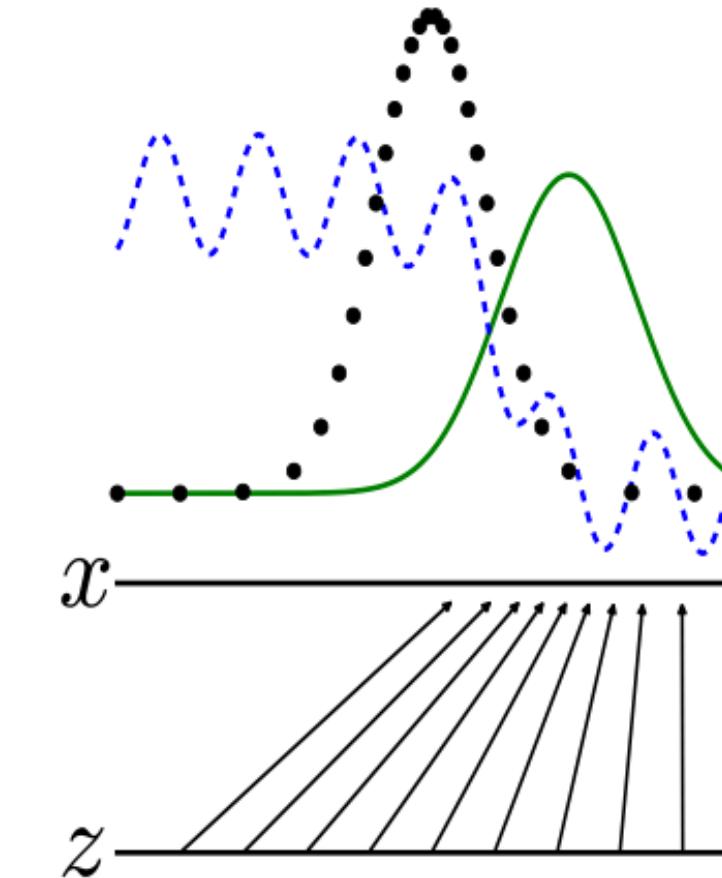
(d)

Distribution Learning of GAN

GAN의 반복적인 학습은 결국 생성하고자 하는 이미지의 확률 분포를 학습하는 것과 같음

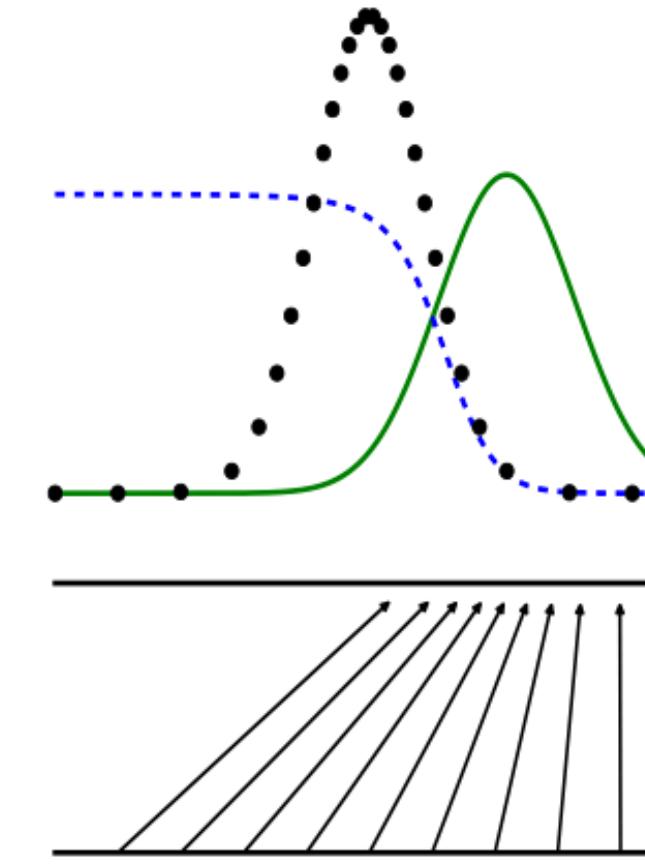
Blue dotted: D
Green solid: G
Black dots: Real samples

G와 Pdata는 꽤나 비슷하다
초록색 G가 약간 오른쪽으로
치우쳐있다.



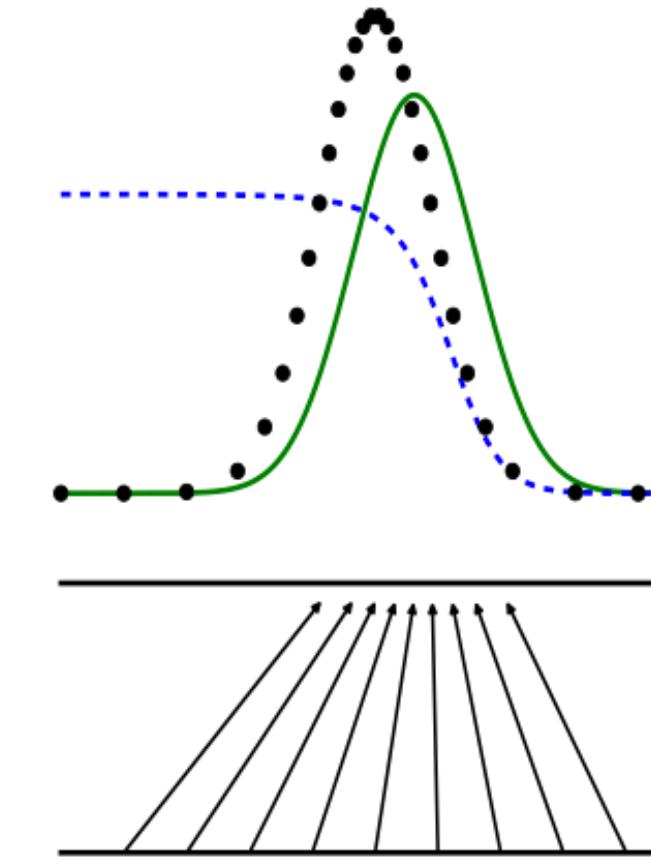
(a)

D는 Pg와 Pdata를 구분하기 위해서 왼쪽에 높은 값을 할당하게 된다.



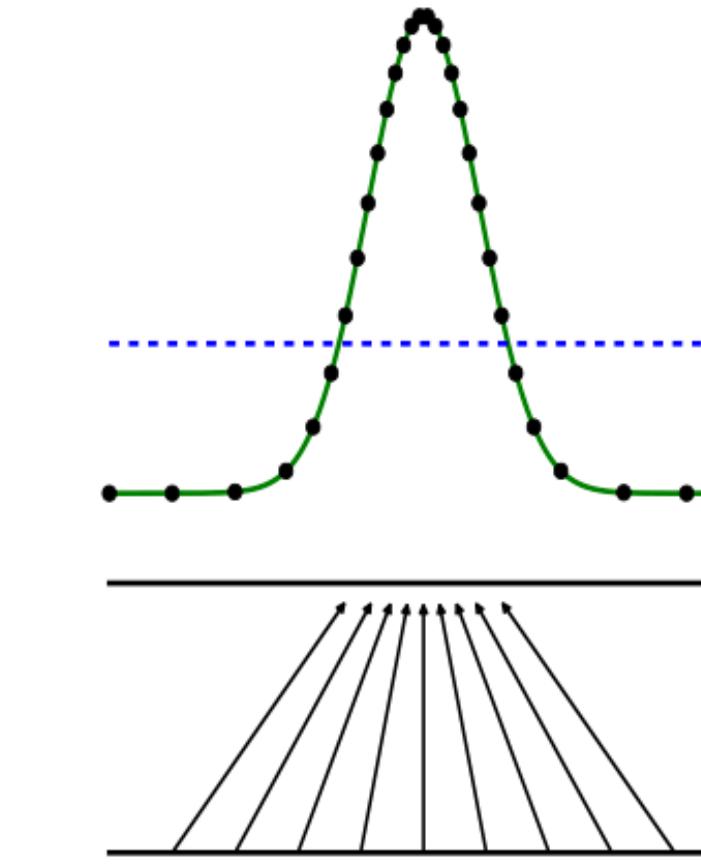
(b)

G는 D가 높아진 왼쪽 영역에 많은 mass를 가하게 되고, 자연스레 mode가 왼쪽으로 밀려간다



(c)

이를 반복하면 결국 Pg와 Pdata가 같아지게 된다. (hopefully)

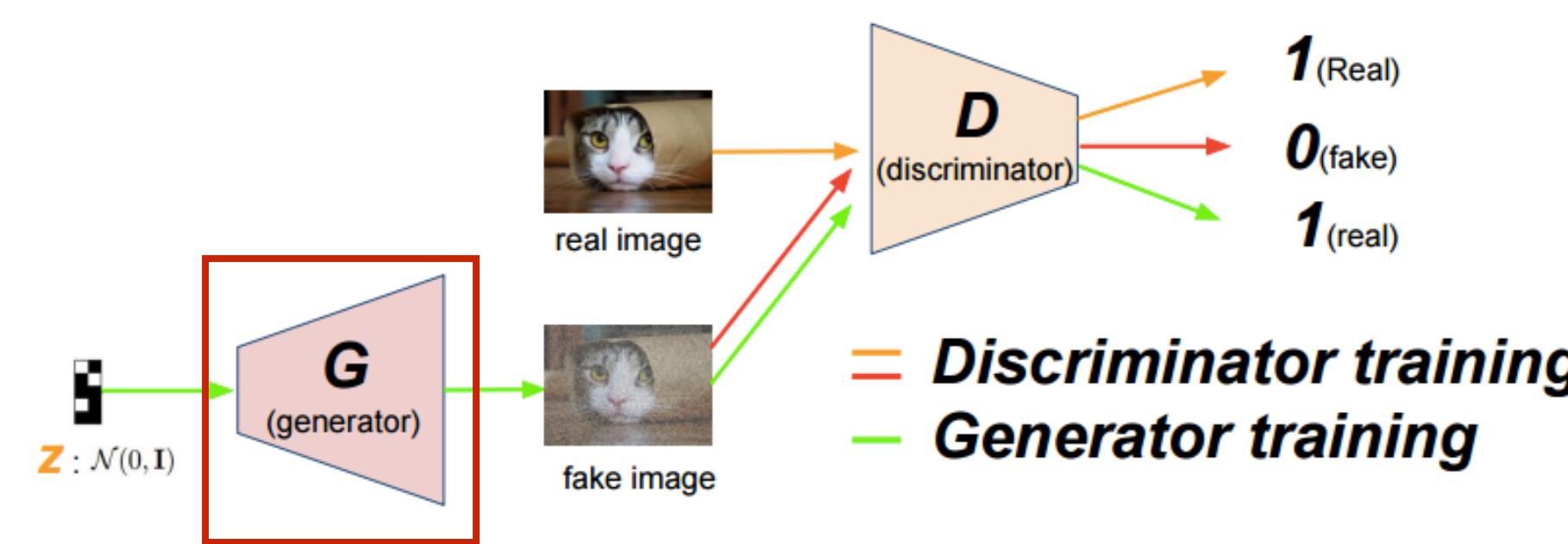


(d)

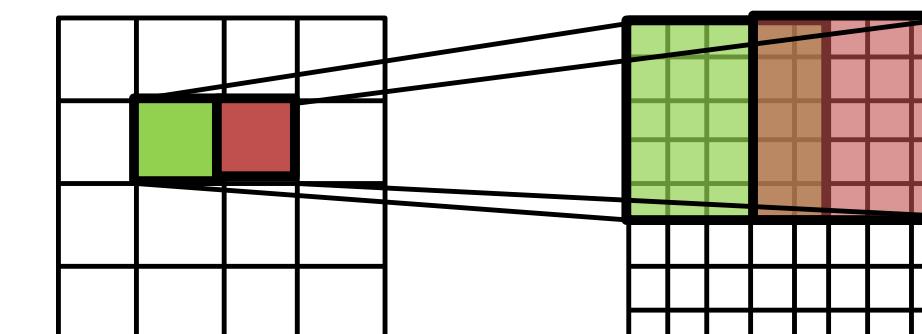
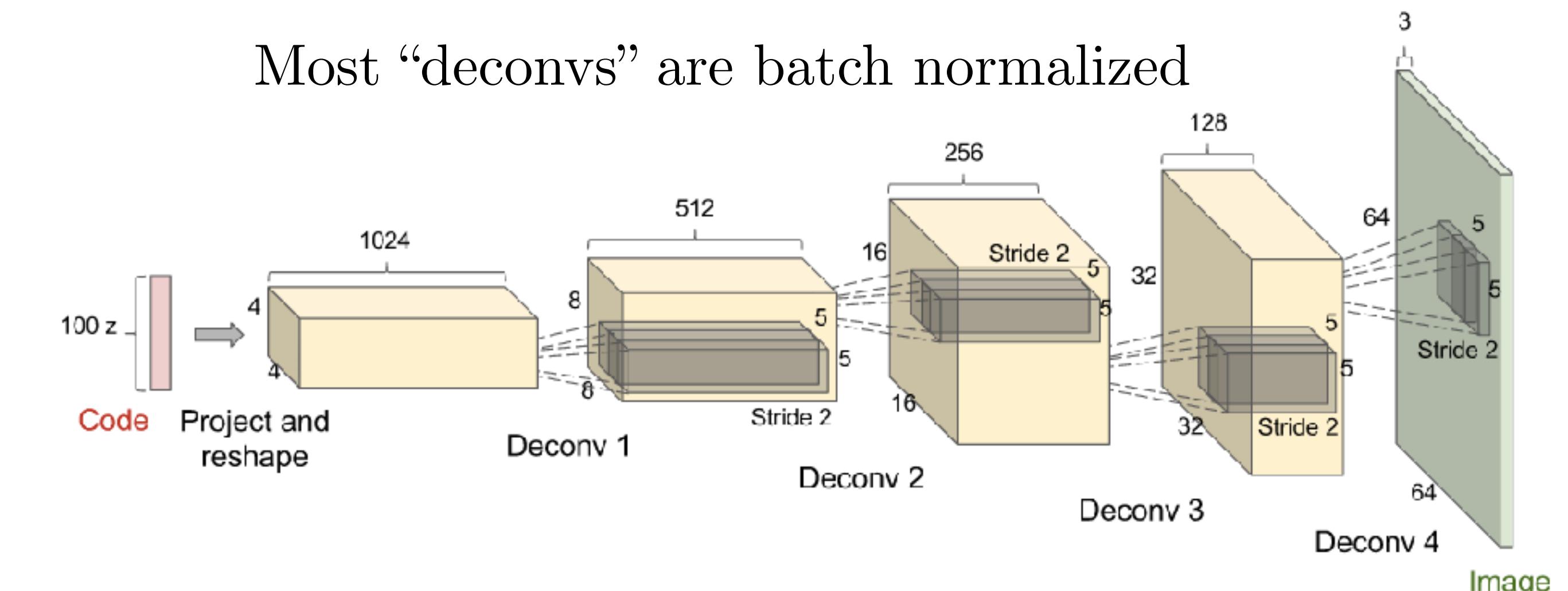
DCGAN

■ DCGAN (Deep Convolutional GAN)

- GAN의 구성 모델을 ConvNet으로 대체한 것으로 높은 해상도의 이미지를 안정적인 학습을 통해서 얻을 수 있는 장점이 있음

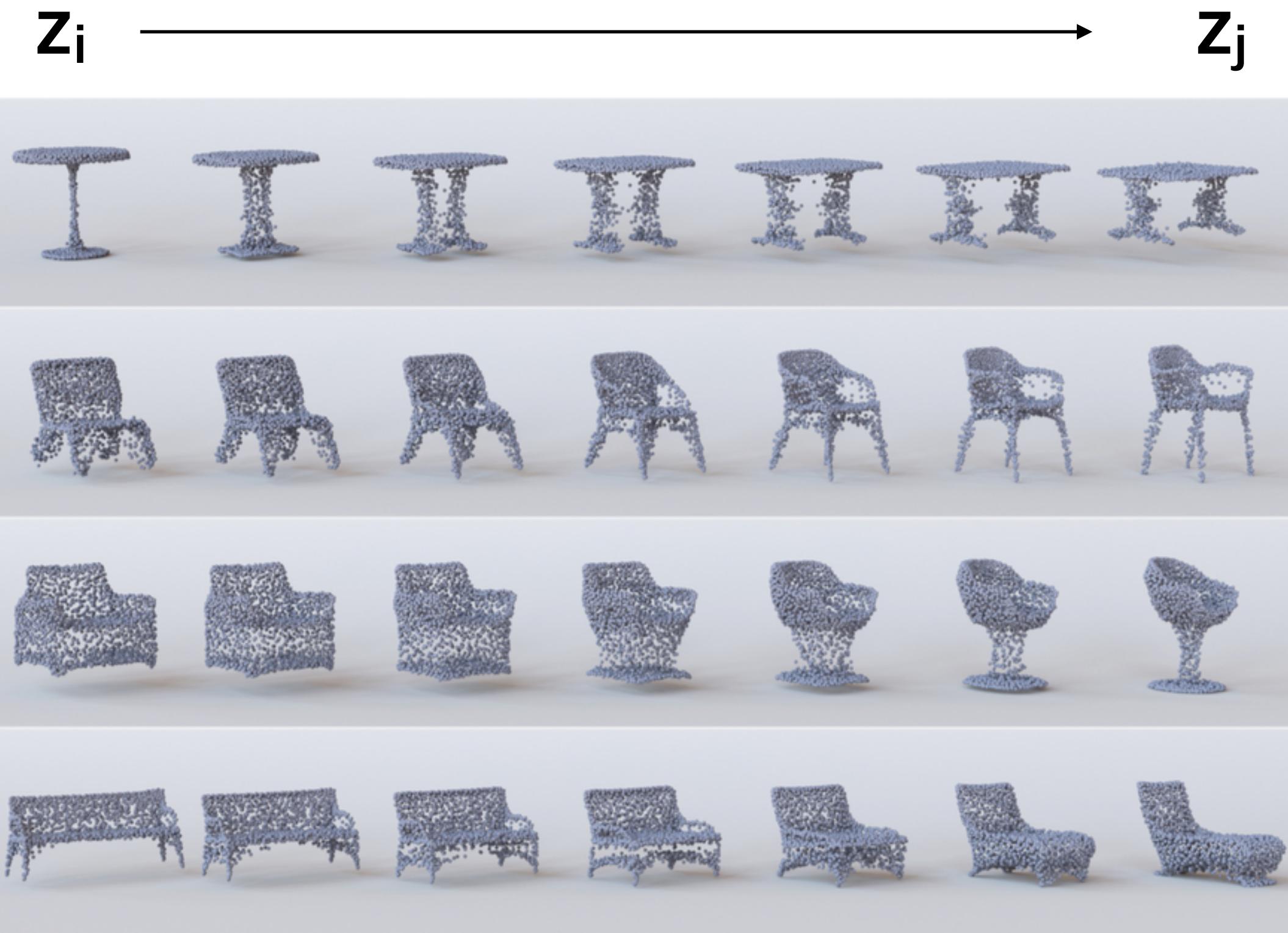


Most “deconvs” are batch normalized



Interpolation in Latent Space

- Random Noise를 이미지의 확률 분포에 Mapping하고, 해당 이미지를 생성하는 방식으로 모델이 작동함
- 유사한 이미지를 만드는 데 사용된 Input Noise들 사이의 이미지들은 어떤 의미가 있을까? (Interpolation)



Interpolation in Latent Space

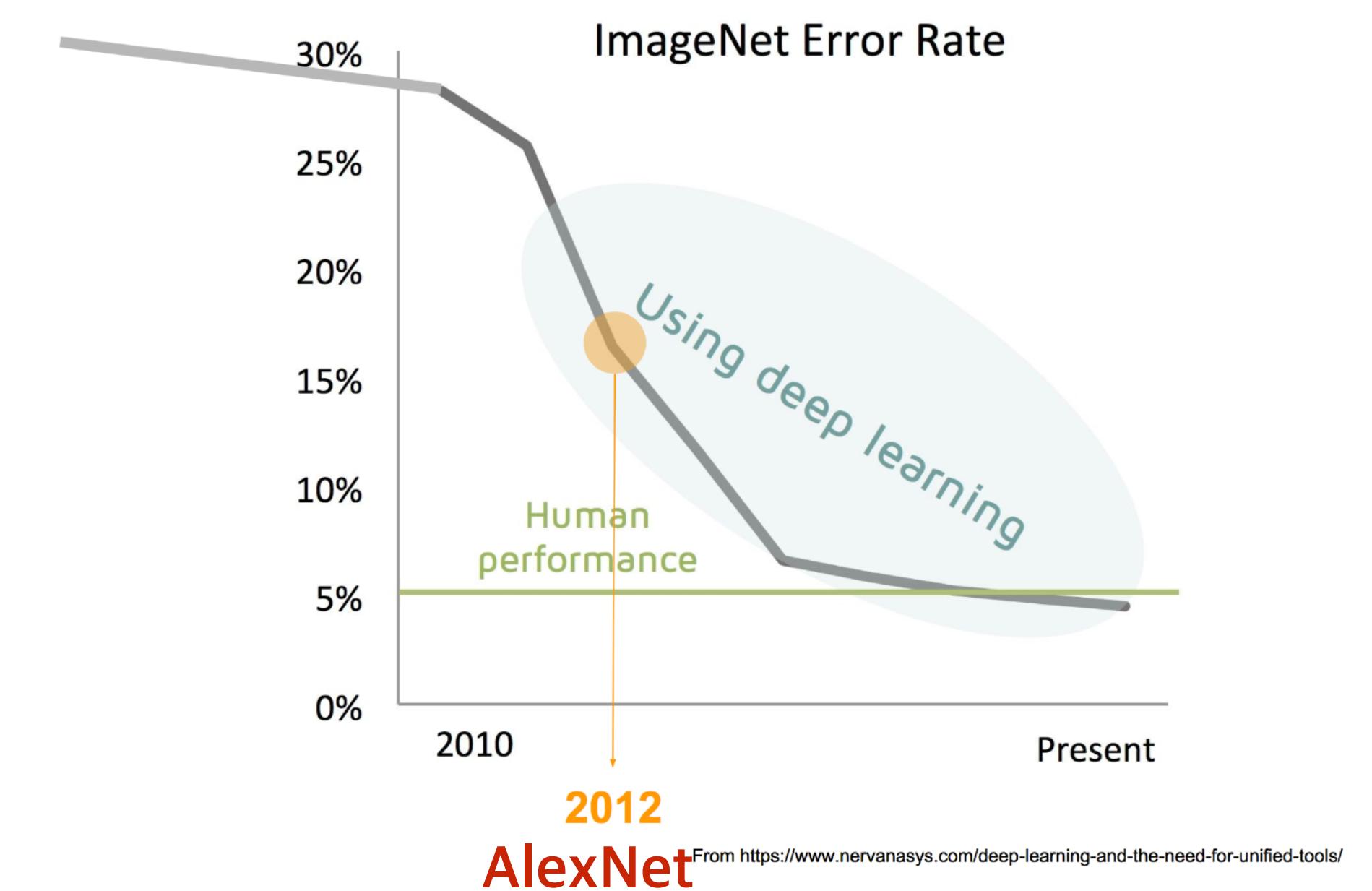
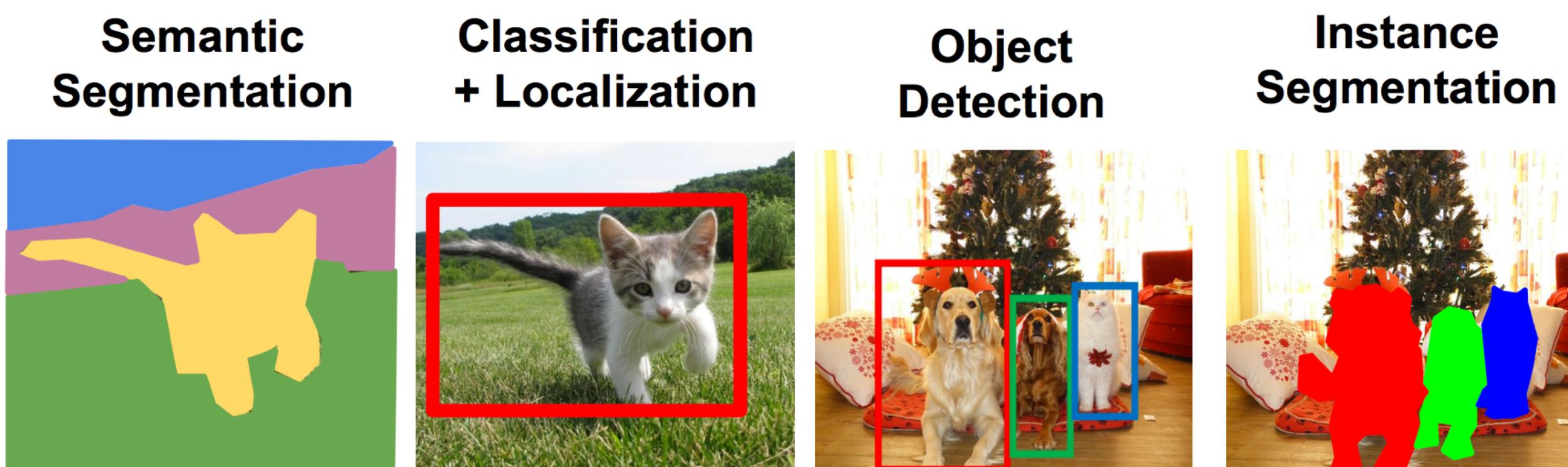
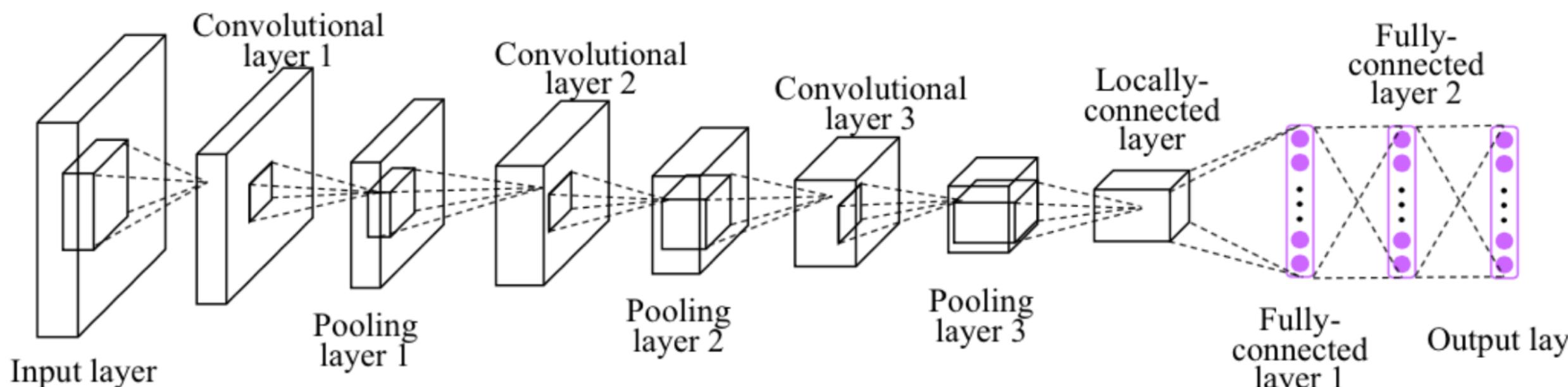
- Random Noise를 이미지의 확률 분포에 Mapping하고, 해당 이미지를 생성하는 방식으로 모델이 작동함
- 유사한 이미지를 만드는 데 사용된 Input Noise들 사이의 이미지들은 어떤 의미가 있을까? (Interpolation)



Style Transfer

Introduction

- Convolutional Neural Network (ConvNet) 기반의 딥러닝 모델은 이미지 처리 분야에서 높은 성능을 보여주었음
- ConvNet의 Hidden layer에서 이미지를 잘 나타내는 특징(representative feature)을 학습하여 표현함
- 과거에는 이미지의 특징을 찾는 Feature Encoding에 집중하여 연구를 진행해왔음



Content Generation with NN

- 이미지의 특징을 잘 계산할 수 있다면, 특징을 바탕으로 이미지를 다시 만들어 수 있지 않을까 (Reconstruction)?
- Feature Inversion

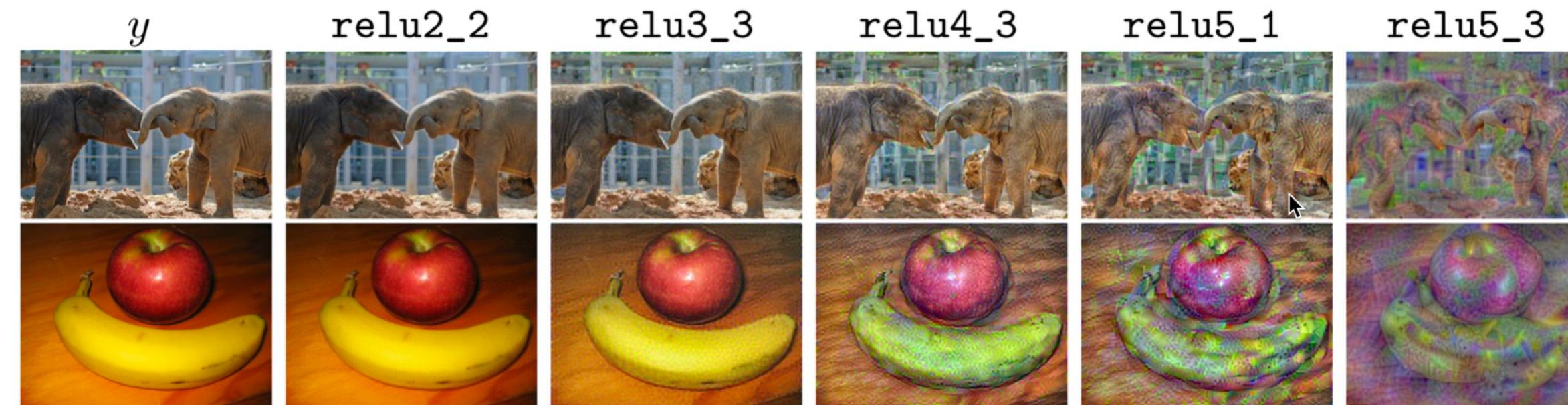
: 특정한 ConvNet feature vector가 주어졌을 때 1) 주어진 특징과 잘 맞고 (match), 2) 어색하지 않고 자연스러운 이미지를 생성하는 것

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathbb{R}^{H \times W \times C}}{\operatorname{argmin}} \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda \mathcal{R}(\mathbf{x})$$

$$\ell(\Phi(\mathbf{x}), \Phi_0) = \|\Phi(\mathbf{x}) - \Phi_0\|^2$$

$$\mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}} \quad : \text{Total Variation Regularizer}$$

<Reconstructing from Different Layers of VGG-16>



Basic Assumption of Style Transfer

- ❑ 사진이나 그림은 표현의 대상인 **내용(Content)**과 표현 방식인 **스타일(Style)**이 존재하며 이는 **구분 가능하다고 전제함**
- ❑ Question: 내용은 그대로 유지하면서 스타일만 바꿀 수 있지 않을까?



+



=



내용: 하늘, 강, 건물들…

스타일: 경계가 뚜렷, 사실적, 낮 …

내용: 마을, 성당, 별, 하늘 …

스타일: 추상적인, 고흐 풍…

내용: 하늘, 강, 건물들…

스타일: 추상적인, 고흐 풍…

General Approach for Neural Style Transfer

- Content 이미지와 Style 이미지를 Input Image로 받고, 목적에 잘 맞는 새로운 이미지를 찾는 방식으로 이미지 생성
- 학습의 대상은 ConvNet 파라미터가 아닌 모델이 생성하고자 이미지 (input X)

Content Input



Generated
Image
(X)

Style Transfer Model
(Image Generator)

Style Input



Learn “X” with minimum loss

- X's content is similar with Content Input
- X's style is similar with Style Input

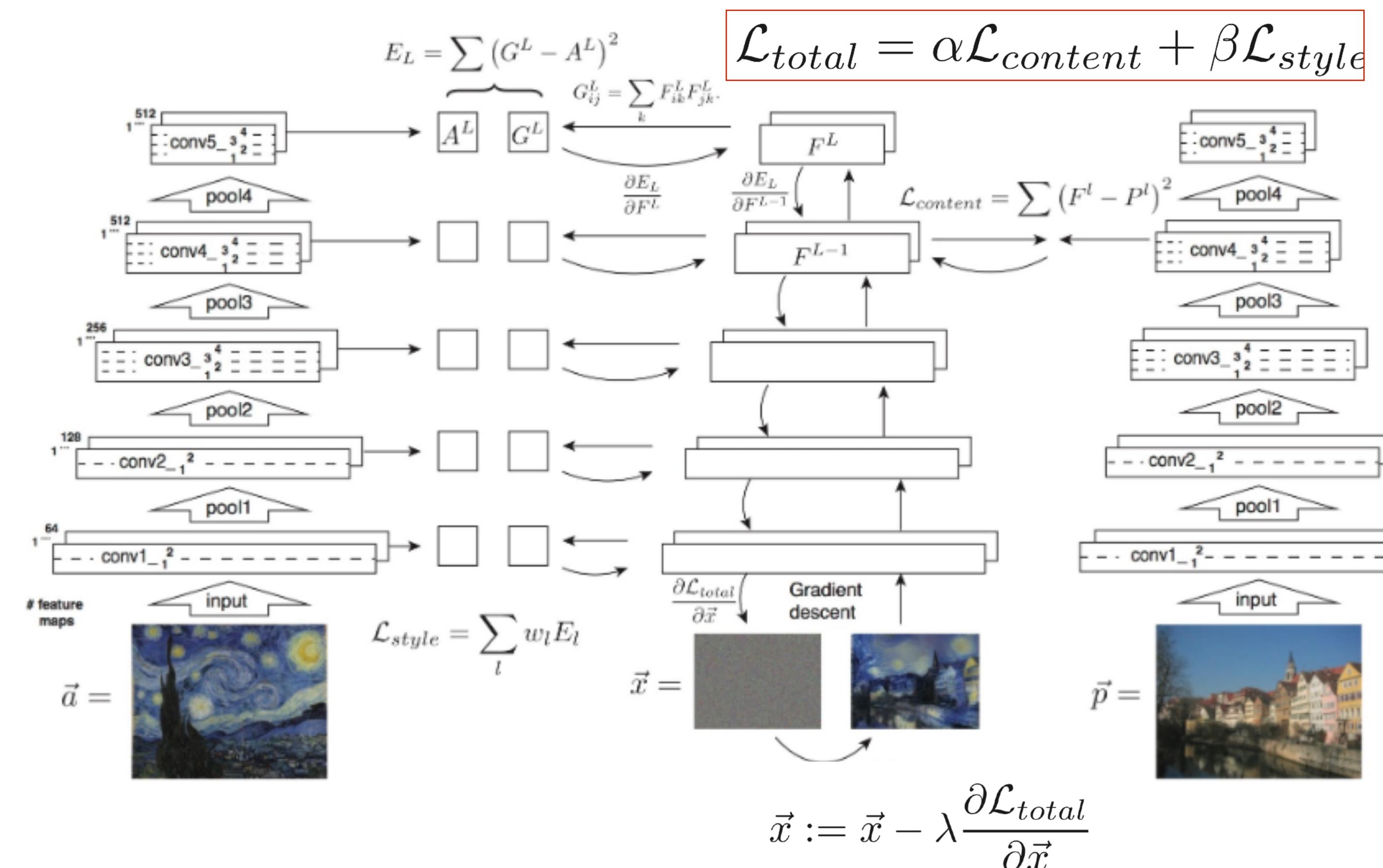
$\vec{x} =$



Loss Function를 어떻게 정의할 것인가?

Neural Style Transfer Framework

- 이미지와 관련하여 미리 학습된 ConvNet 모델을 이용하여 Style Transfer를 진행함 (주로 VGG-16, VGG-19 이용)
- 생성한 이미지에 대하여 Content Loss와 Style Loss를 정의하고, 둘의 합을 Total Loss로 정의
- 입력된 Content & Style Image에 대하여 Total Loss를 최소화하는 이미지를 반복적으로 학습하여 생성함



Content Representation: Loss Function

- 1 번째 layer의 Content Loss는 아래와 같이 정의됨
- Content image의 feature map과 Input image의 feature map 사이의 SSE (Sum of Square Error)

<l-th Content Loss Definition>

$$\mathcal{L}_{\text{content}}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

$$\frac{\partial \mathcal{L}_{\text{content}}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & \text{if } F_{ij}^l > 0 \\ 0 & \text{if } F_{ij}^l < 0 \end{cases}$$

\vec{x} : Input Image (모델이 생성하는 이미지, 학습의 대상)

\vec{p} : Input Content Image

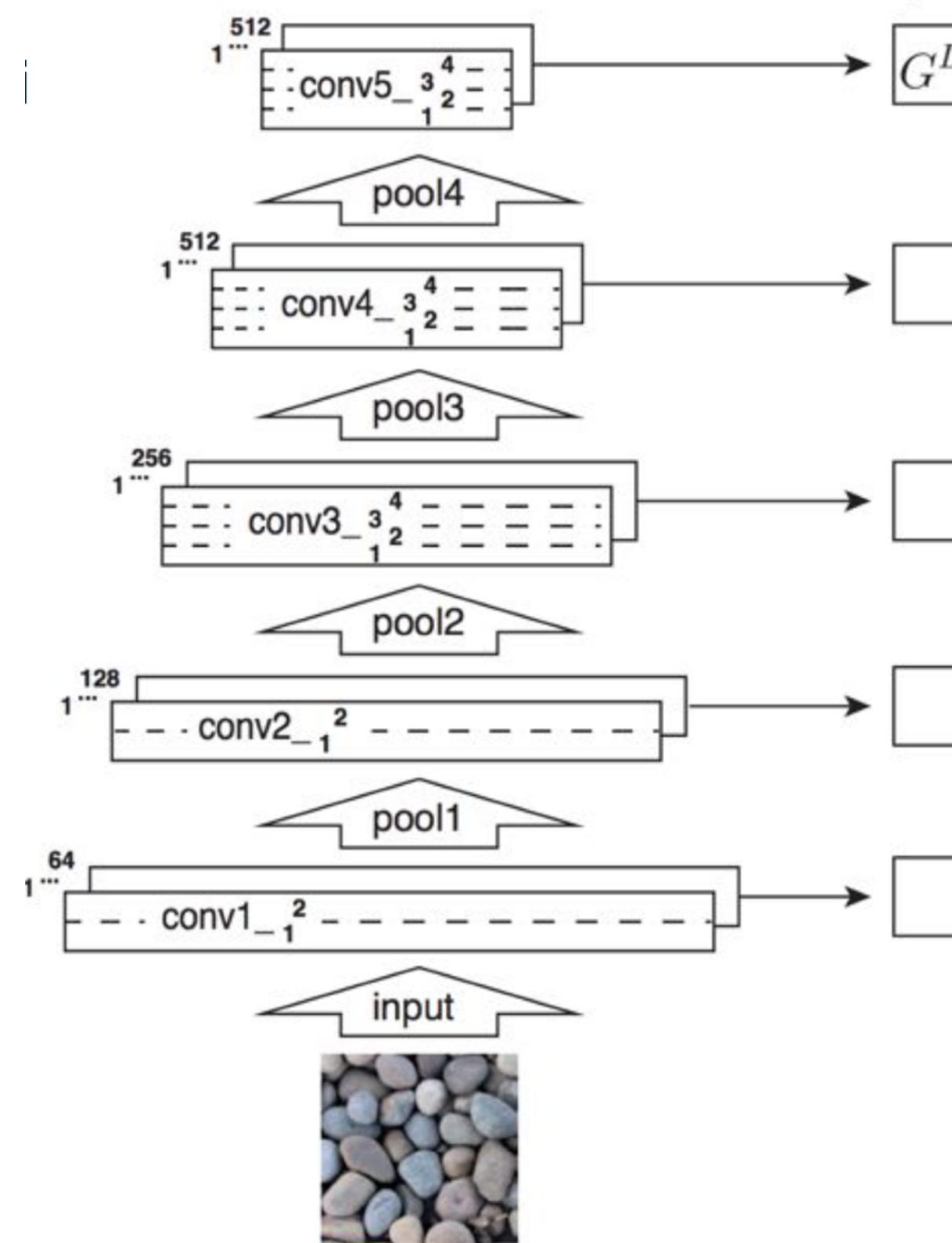
M_l : l-th layer의 feature map size (height * width)

N_l : l-th layer의 feature map 개수

F_{ij}^l : l-th layer에서
i-th feature map의
j-th 위치에 있는 activation 값

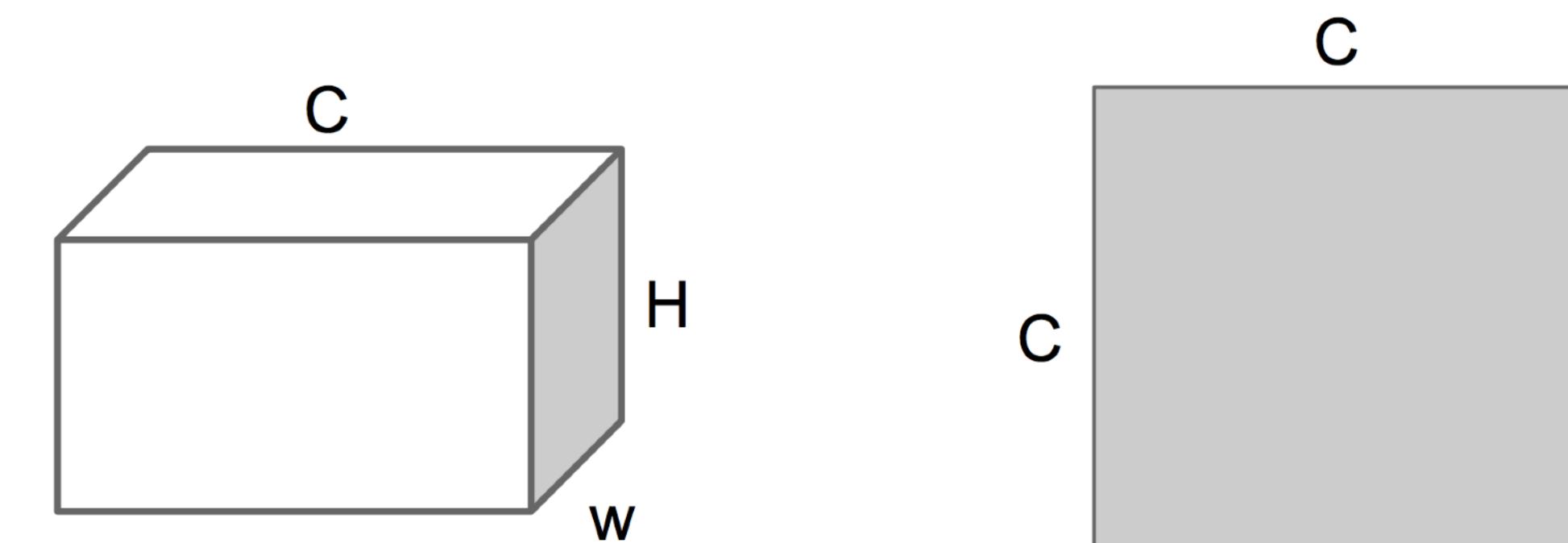
Style Representation: Gram Matrix

- Feature maps 사이의 Correlation을 나타내는 Gram Matrix를 통해 이미지의 Style을 표현함
- Input image와 Style image의 Gram matrix를 유사하게 만들도록 style loss를 설정하고 input image X를 학습함



$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l$$

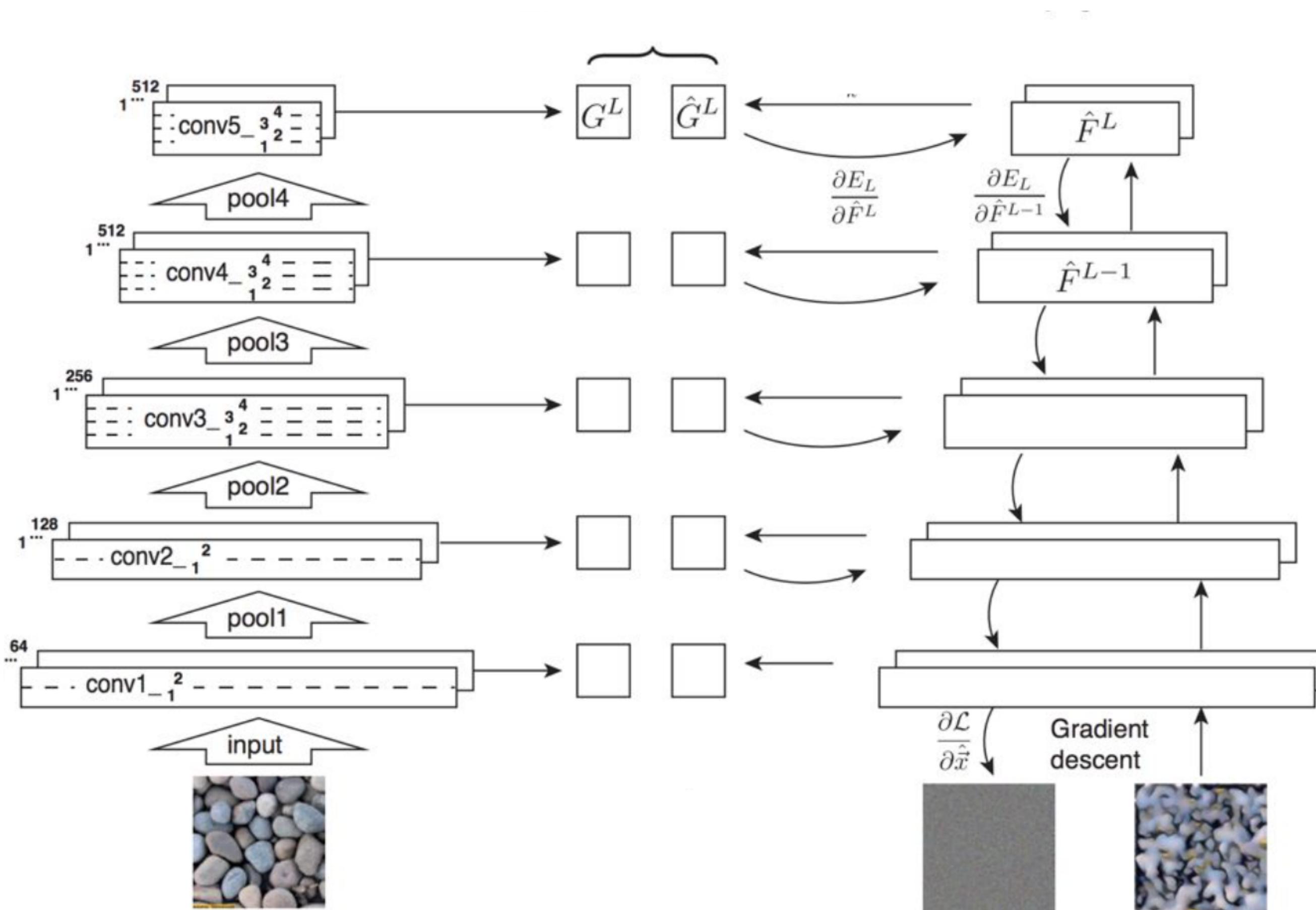
l -th layer에서 i 번째 feature와 j 번째 feature 사이의 correlation을 의미함.
(각각의 feature map을 vector 형태로 reshape)



Efficient to compute; reshape features from
 $C \times H \times W$ to $=C \times HW$
then compute $G = FF^\top$

Style Representation: Loss Function

- Input image X 와 Style Input Image의 Gram matrix의 차이를 Style Loss로 정의함
- Layer 별로 style loss를 계산하고, 모든 layer의 style loss 가중치 합을 전체 style loss로 정의함



<l-th Style Loss Definition>

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

<Overall Style Loss Definition>

$$\mathcal{L}_{\text{style}}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l,$$

Loss Function Summary

- Total loss는 Content loss와 Style loss의 가중치 합으로 정의됨
- Content와 Style 사이의 일치 정도를 weight으로 조절하며 trade-off를 진행함

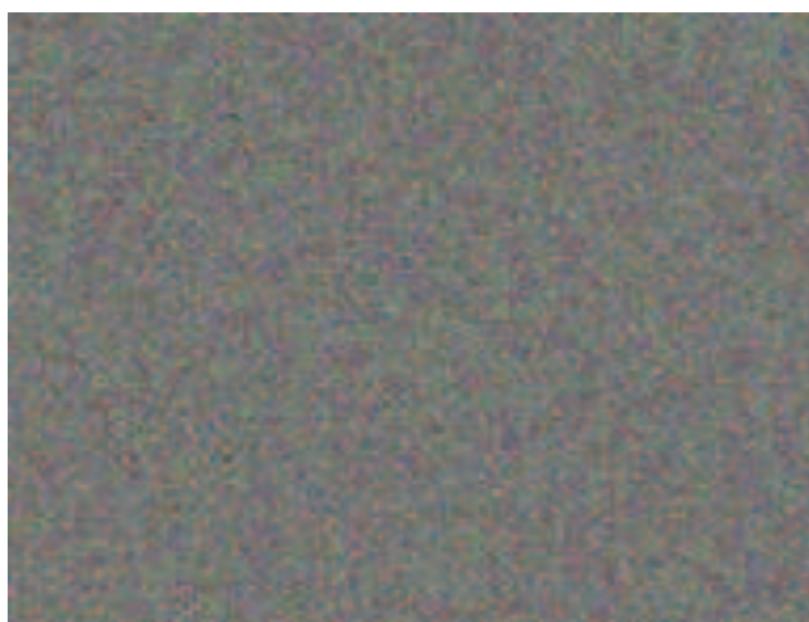
$$\mathcal{L}_{\text{total}}(\vec{p}, \vec{a}, \vec{x}) = \alpha \underline{\mathcal{L}_{\text{content}}(\vec{p}, \vec{x})} + \beta \underline{\mathcal{L}_{\text{style}}(\vec{a}, \vec{x})}$$

Input X와 Content Input의
Feature map 값 차이

Input X와 Style Input의
Feature map의 Gram matrix 차이

Learning? ↓

random noise image X



trained image X



$\vec{x} := \vec{x} - \lambda \frac{\partial \mathcal{L}_{\text{total}}}{\partial \vec{x}}$

X를 random noise로 초기화한 후,
total loss를 최소화하는 방향으로 반복적으로 X를 학습 (픽셀 단위)

Results

- 다양한 Style Image들에 대하여 원활하게 transfer하는 것을 확인할 수 있음
- Content & Style loss 사이의 weight 변수를 조절함에 따라 content & style matching 정도 조절 가능
(일반적으로 alpha/beta = 0.001)



More weight to content loss More weight to style loss

The Effect of Matching Content Representation in Different Layers

- 상위 layer의 feature map을 content representation으로 사용할 수록 왜곡 정도가 심해지는 것을 확인 가능



Photorealistic Style Transfer Result

□ 해당 연구는 Artistic style transfer를 목적으로 만들어졌지만, photorealistic style transfer에도 이용할 수 있음

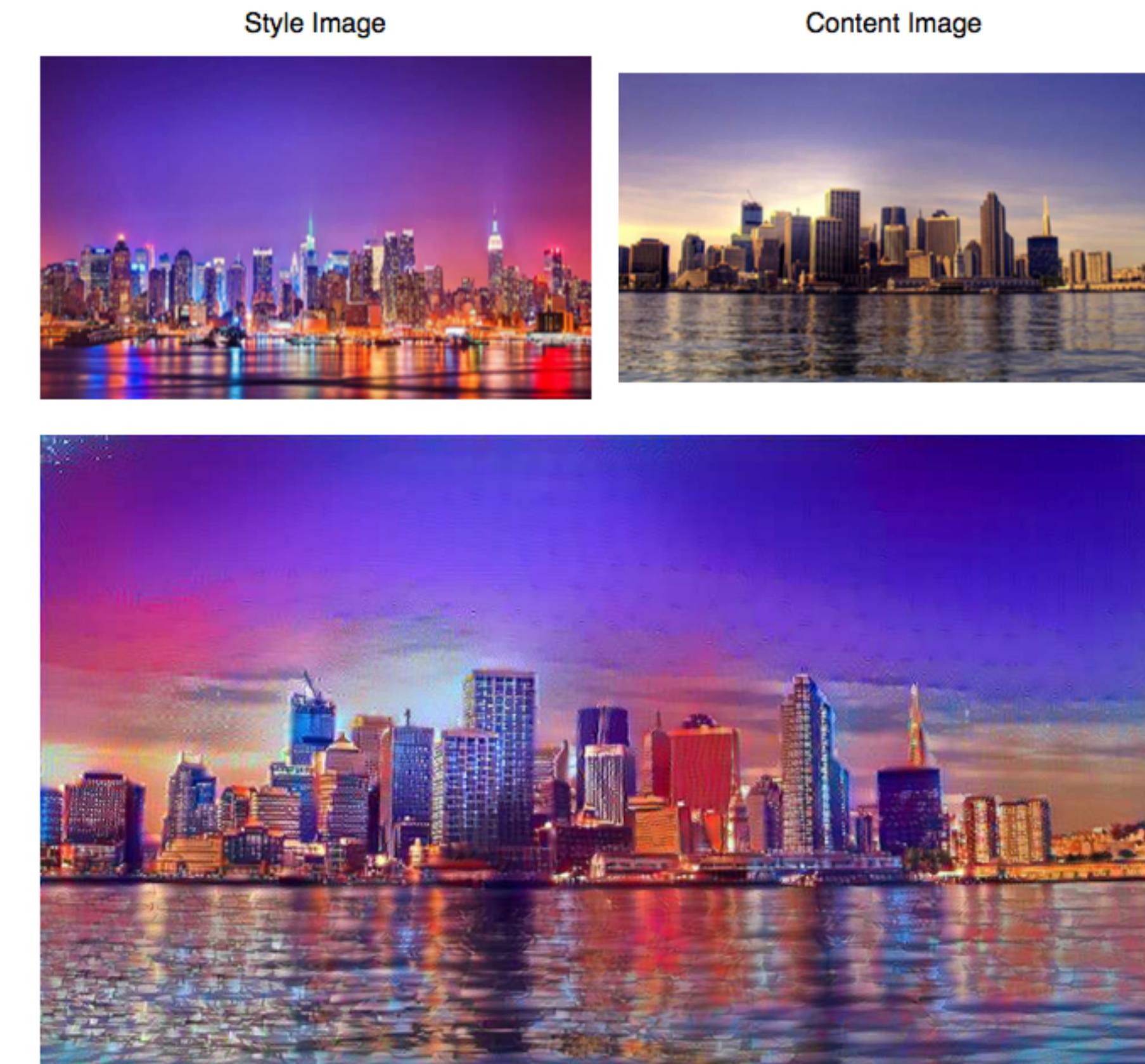
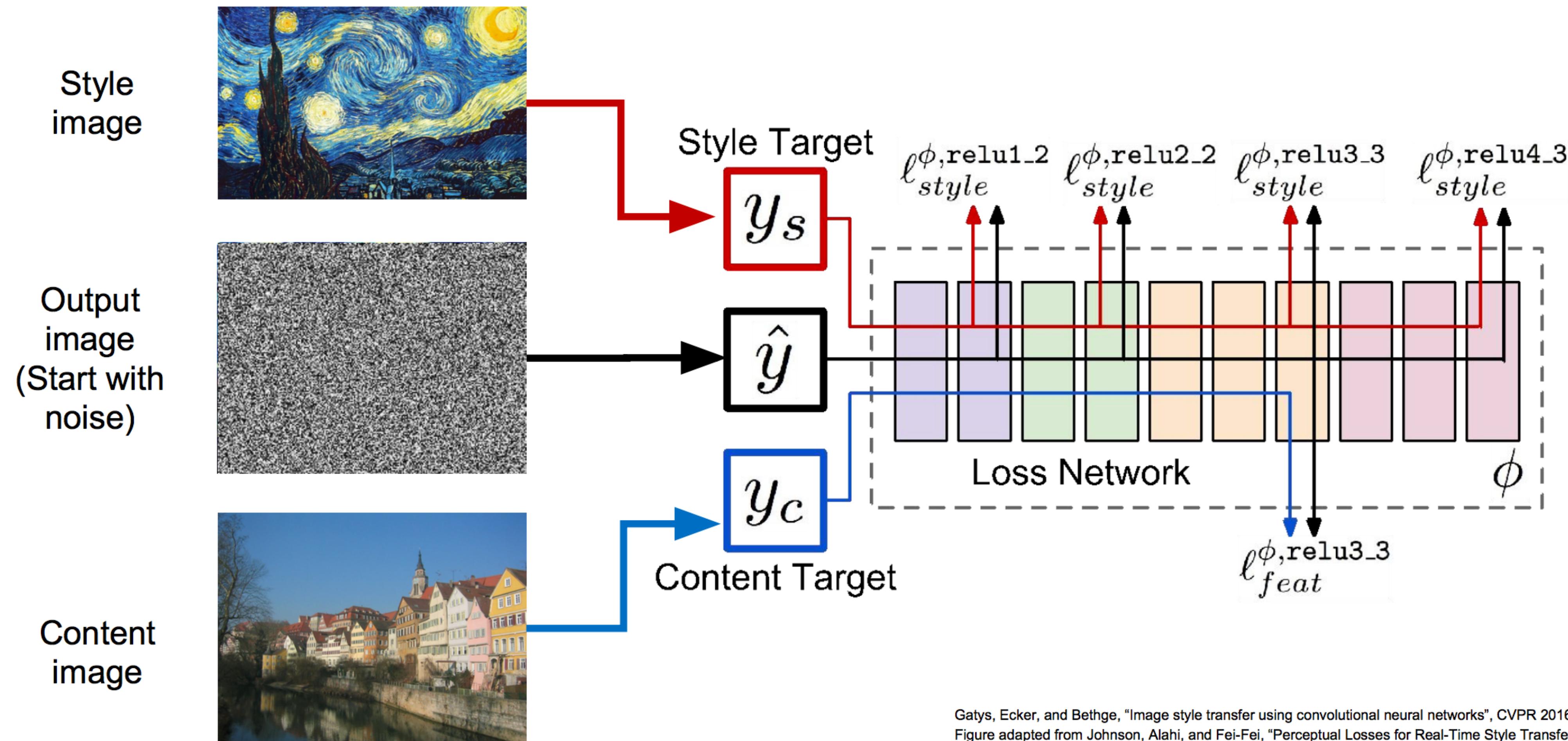


Figure 7. Photorealistic style transfer. The style is transferred from a photograph showing New York by night onto a picture showing London by day. The image synthesis was initialised from the content image and the ratio α/β was equal to 1×10^{-2}

Summary



Gatys, Ecker, and Bethge, "Image style transfer using convolutional neural networks", CVPR 2016
Figure adapted from Johnson, Alahi, and Fei-Fei, "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Q & A