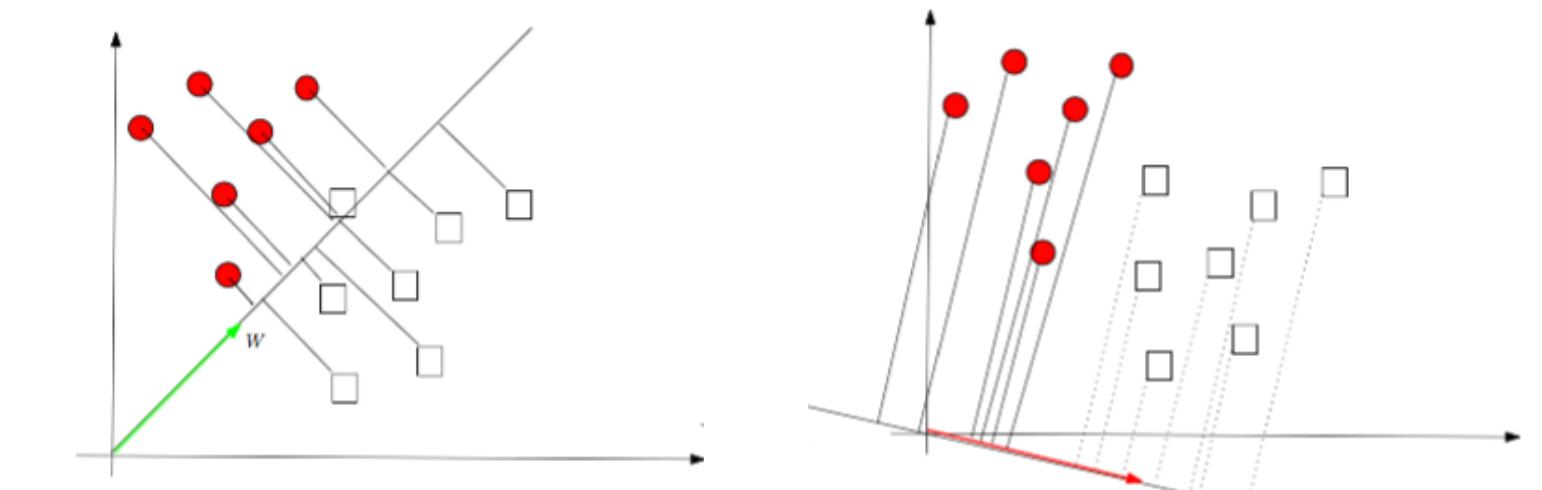
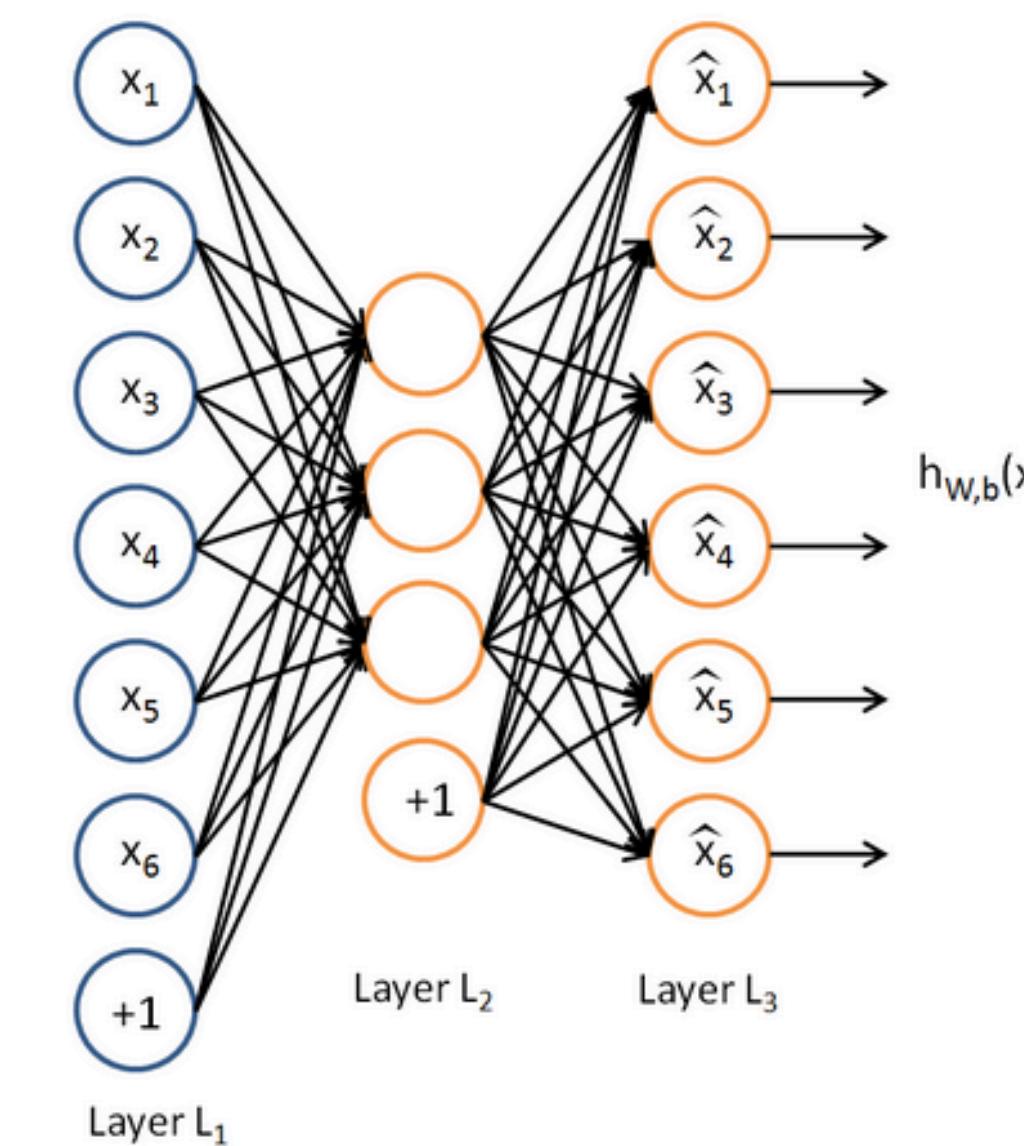
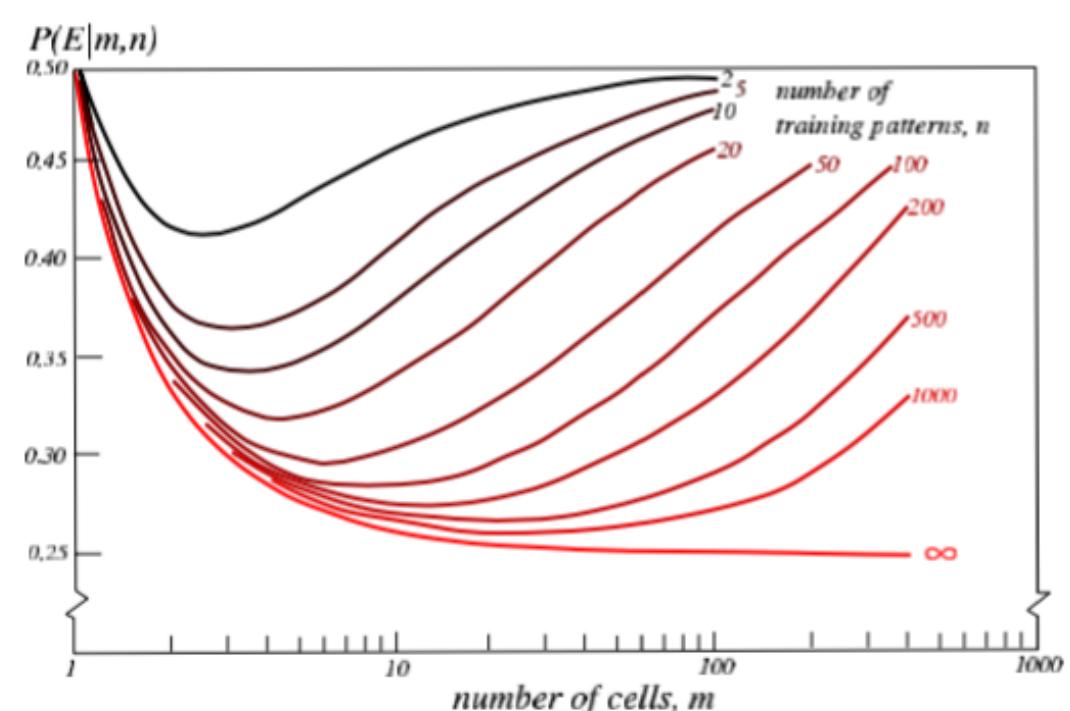


[170720] 인공지능 Open Lecture 3주차

# Unsupervised Learning for Dimensionality Reduction



엑셈-포스텍 R&D 센터  
이도업

---

# **Contents**

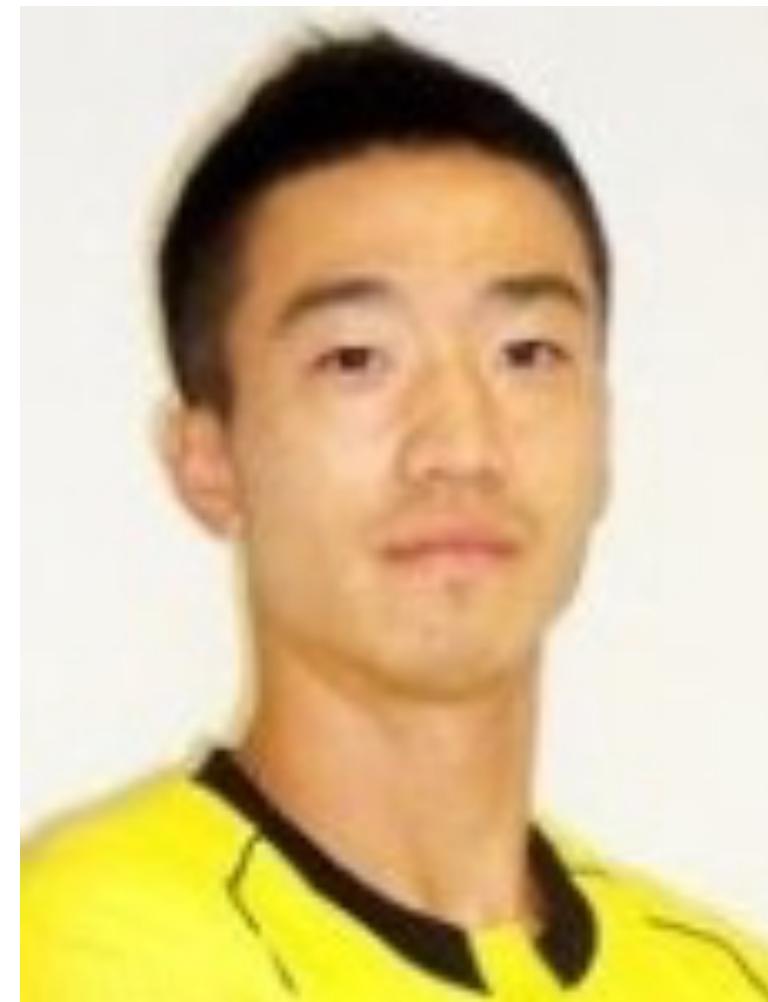
---

- 1. Introduction**
- 2. Principal Component Analysis**
- 3. Linear Discriminant Analysis**
- 4. Autoencoder**

# 1. Introduction

# 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 둘려서 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

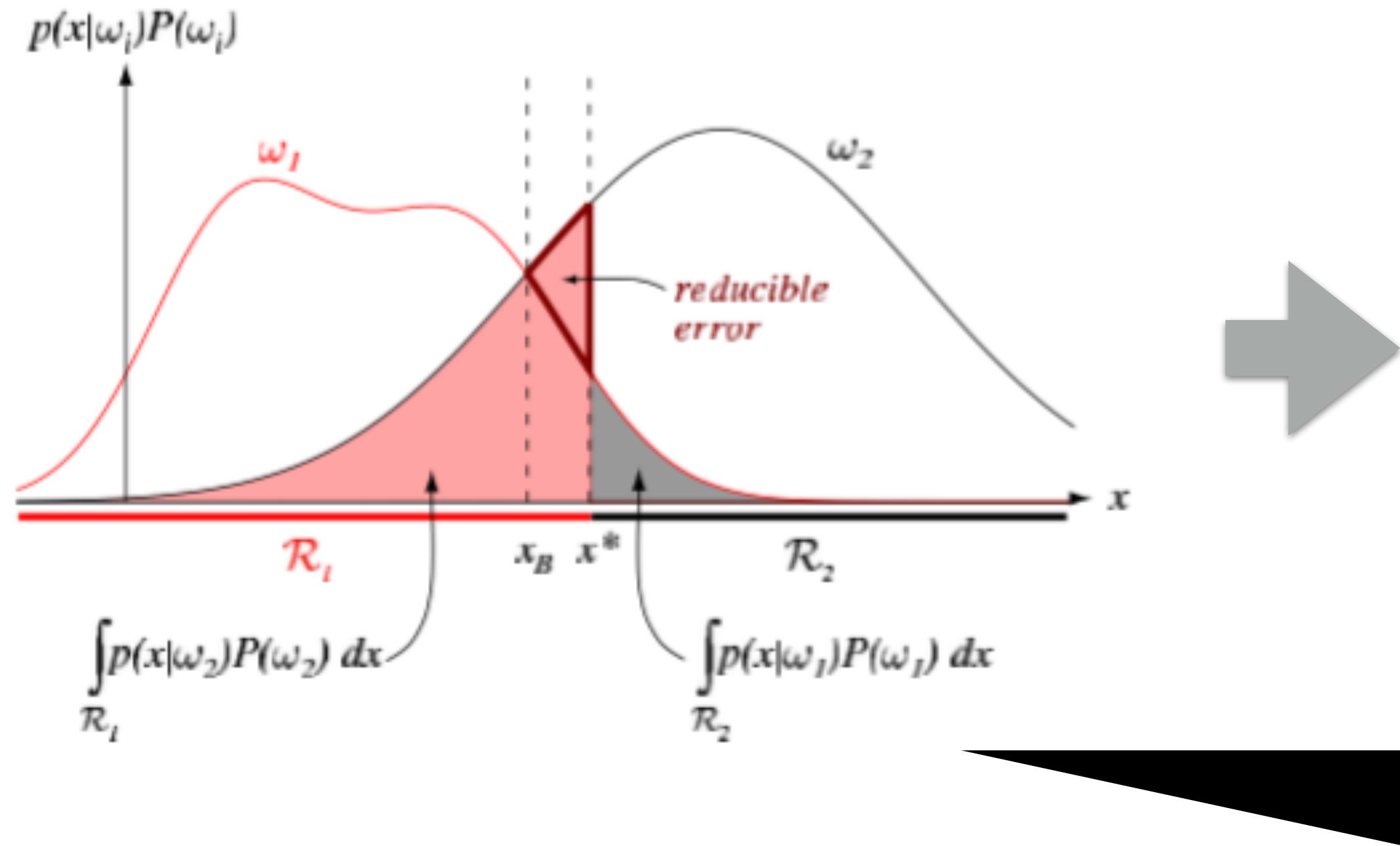
□ 차원의 수가 너무 클 경우, **두가지 문제**가 발생할 수 있음

- Error can increase (Computation Time)

- Training Data Scale

# Problem of Error with High Dimensionality

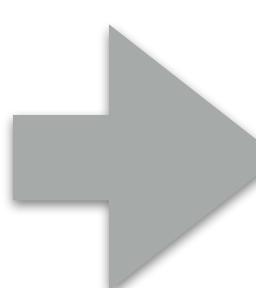
- 이론적으로 볼 때, 차원의 수가 증가할수록 에러는 작아져야 함
- 각 클래스의 사전 확률이 같고, 특징의 분포가 정규분포라고 가정할 때, 에러 확률은 마할라노비스 거리로 표현될 수 있음



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

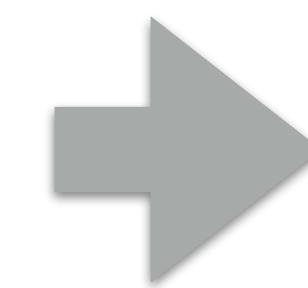
$p(\mathbf{x}|\omega_i) \sim N(\boldsymbol{\mu}_i, \Sigma)$ , and  $P(\omega_i) = \frac{1}{2}$ ,  $i = 1, 2$ ,

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



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

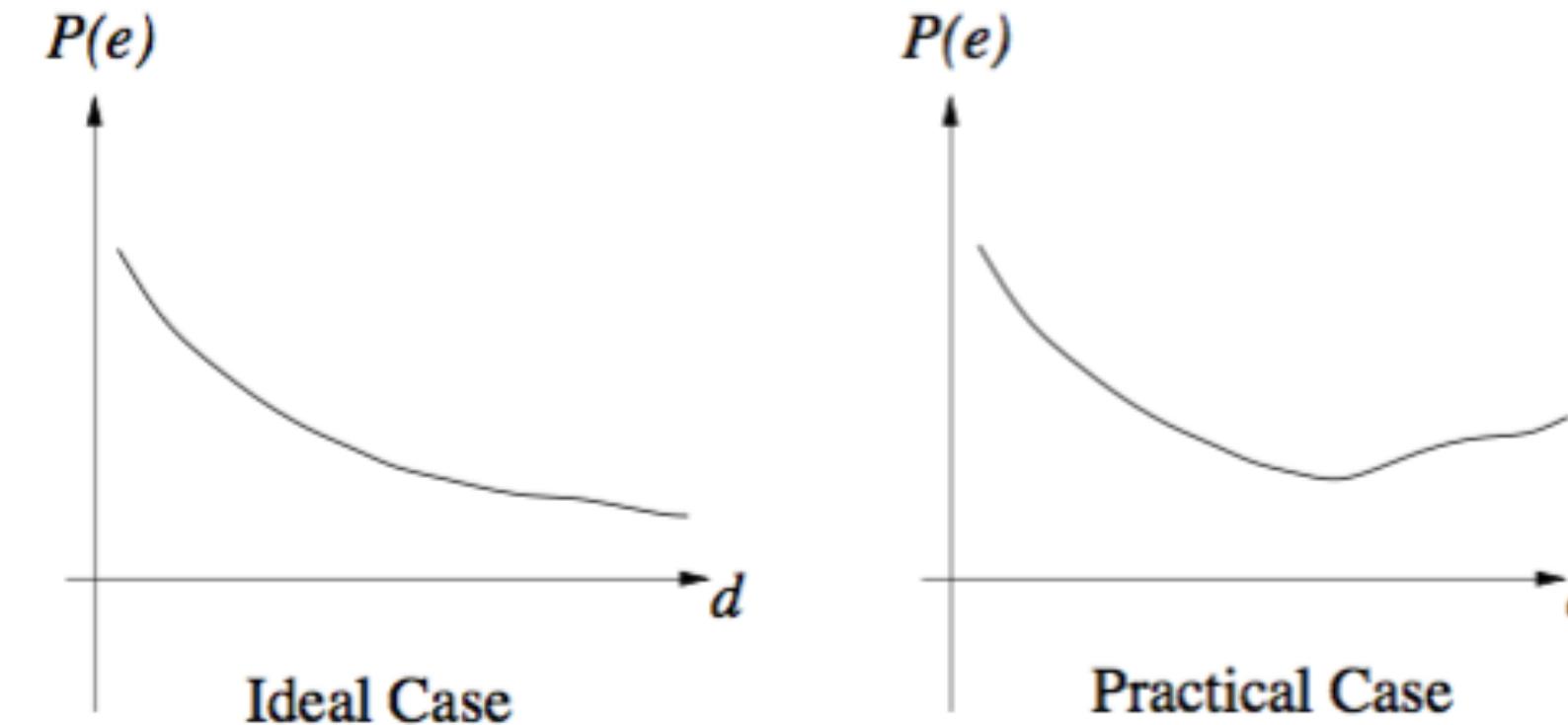
$$\gamma^2 = (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)^T \Sigma^{-1} (\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) = \sum_{i=1}^d \left( \frac{\mu_{i1} - \mu_{i2}}{\sigma_i} \right)^2$$



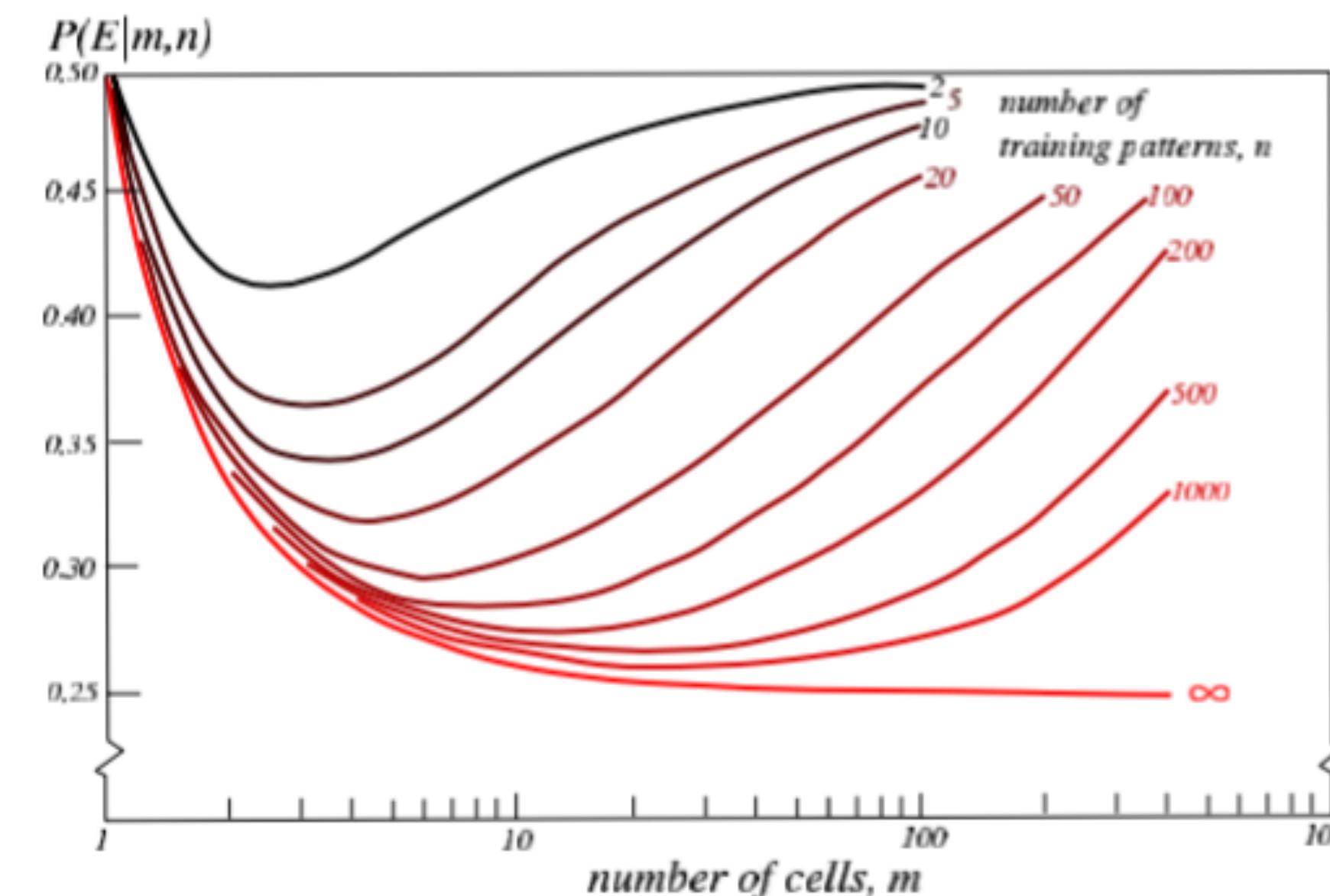
$$\lim_{\gamma \rightarrow \infty} P(e) = 0.$$

# 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

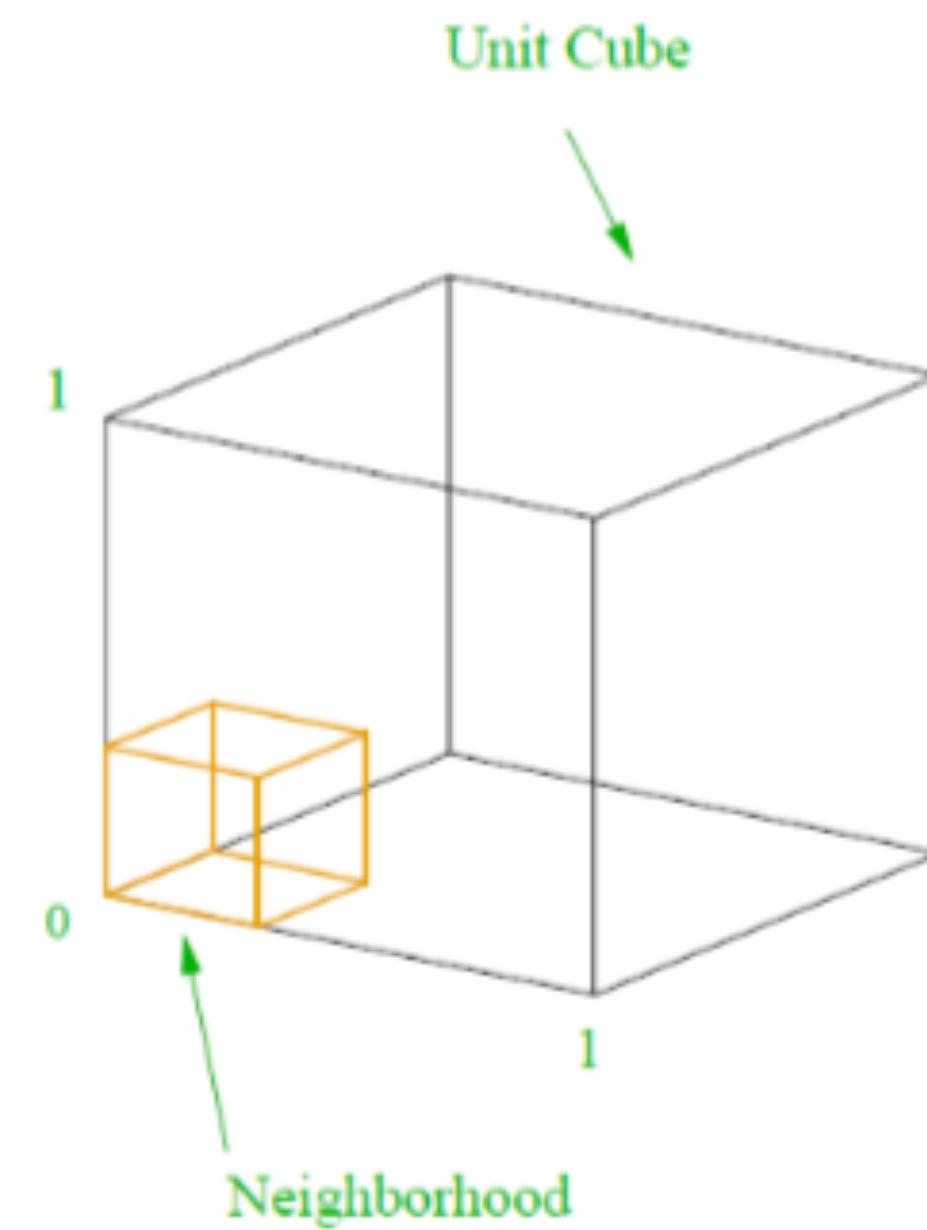


# Problem of Training Set

## ▣ 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?



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

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

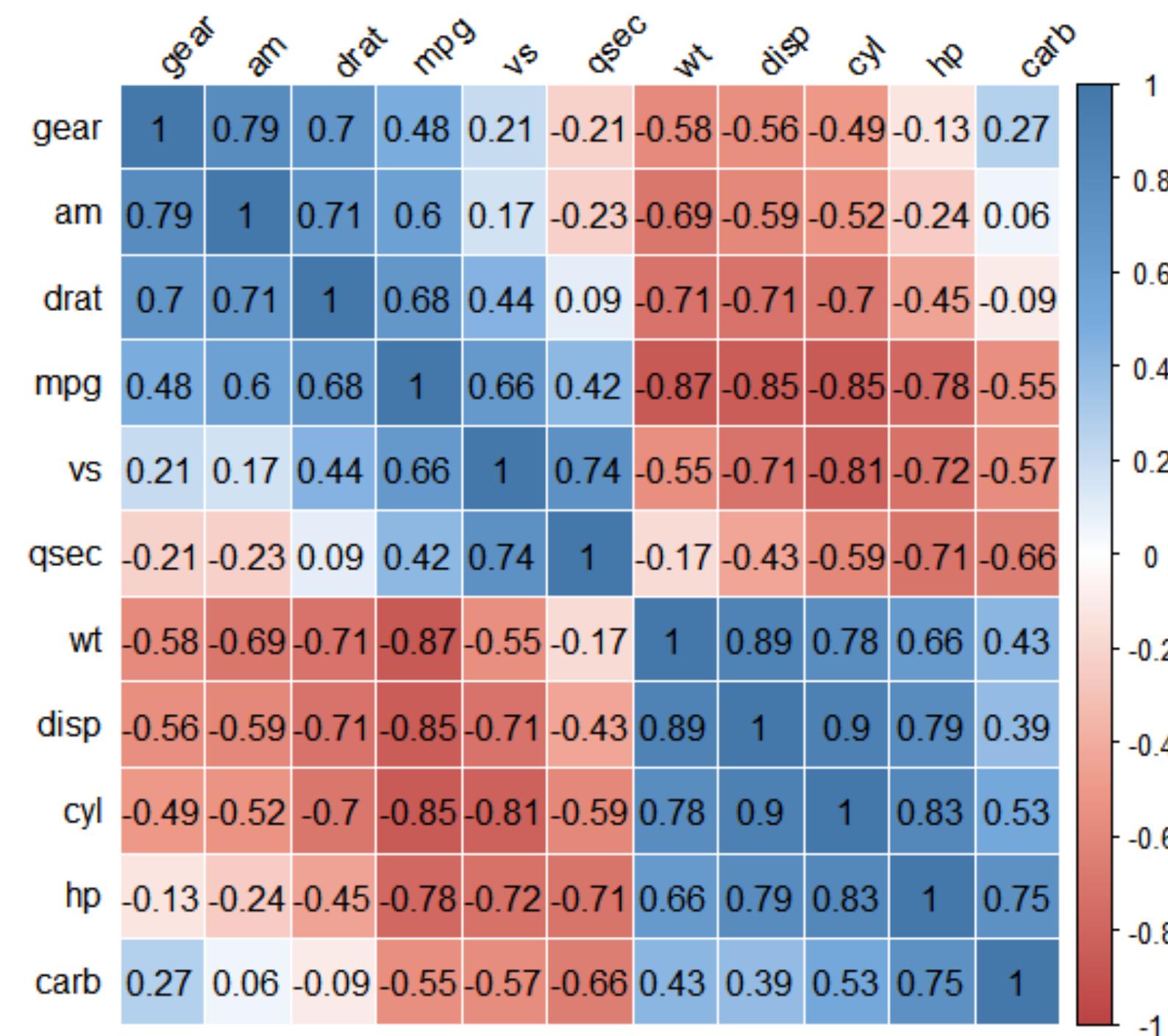
- 전체 데이터의 1% 정보를 알기 위해서 각 차원의 정보가 63%씩 필요
- 전체 데이터의 10% 정보를 알기 위해서 각 차원의 정보가 80% 필요

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

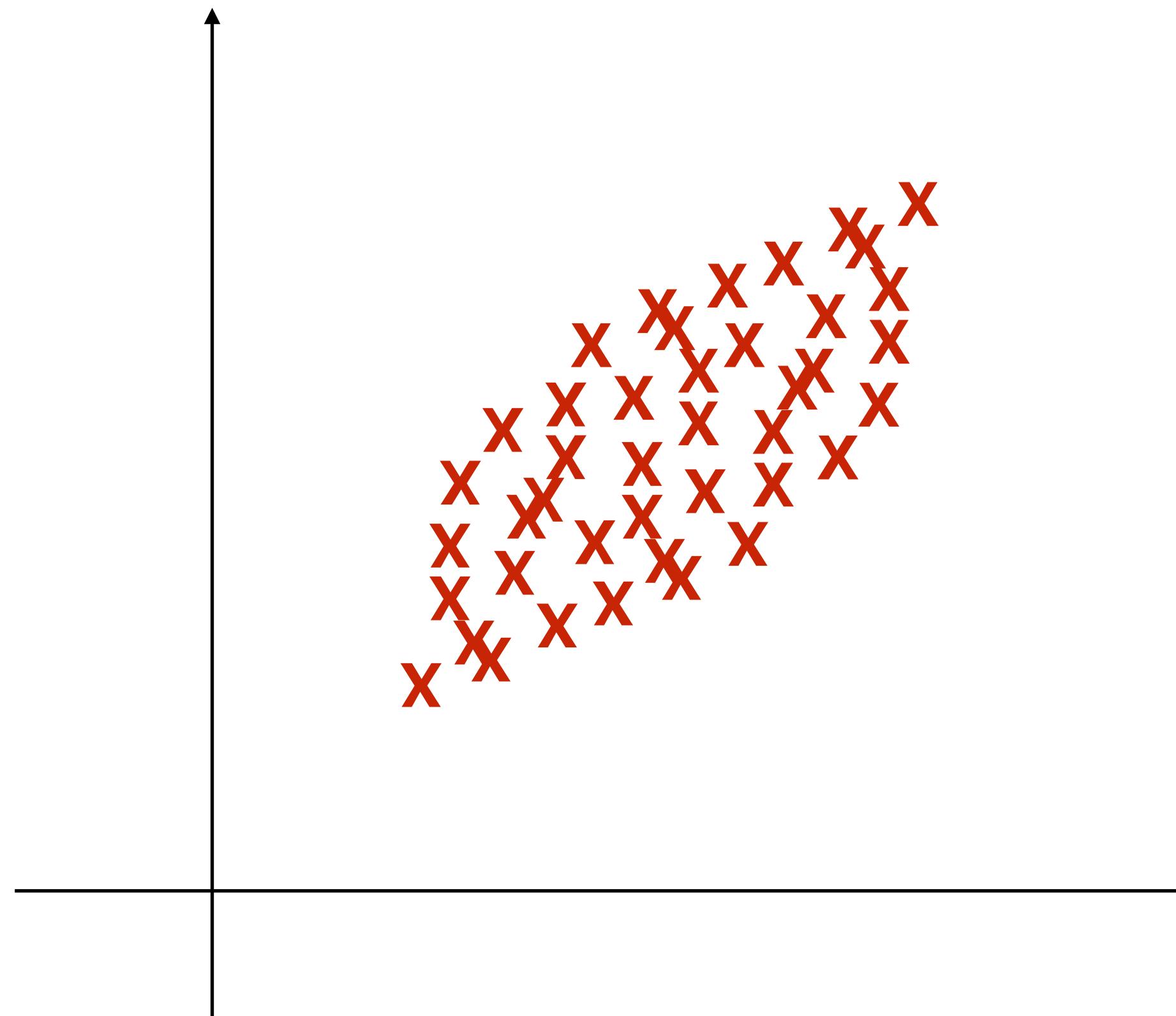


# 2. Principal Component Analysis

# 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

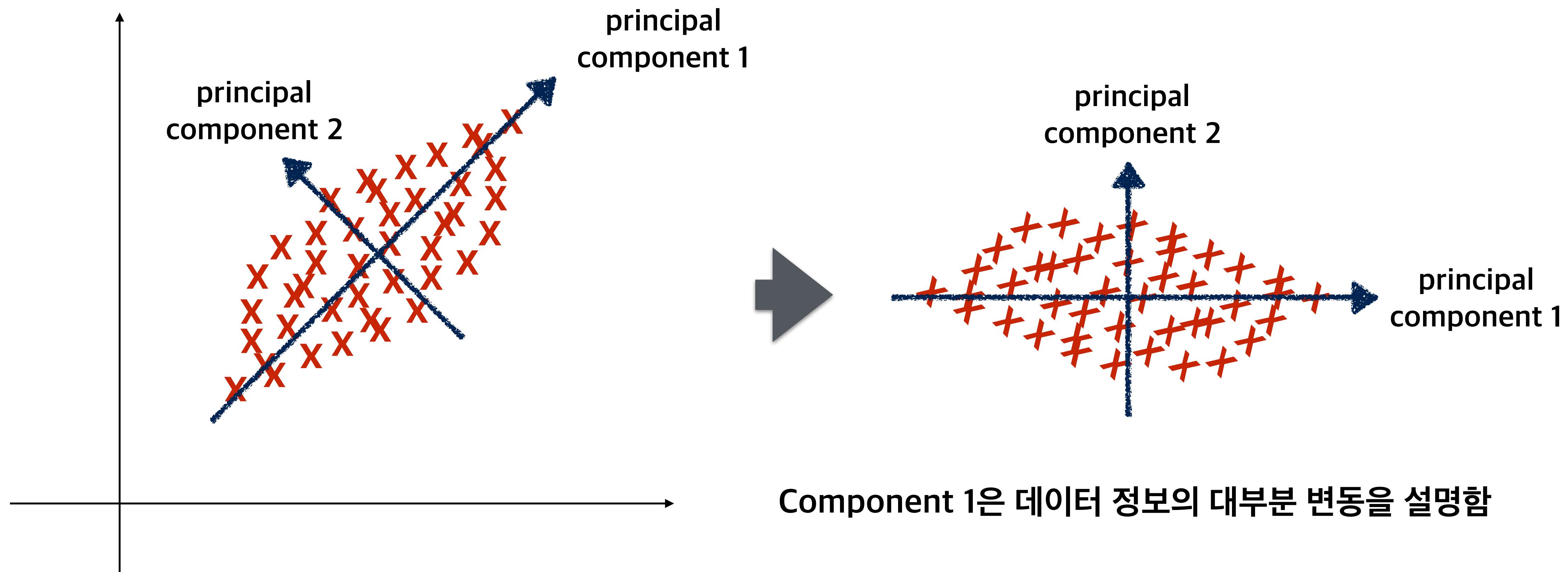


데이터의 분산을 최대화하는 방향으로  
차원을 축소

# 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

The diagram illustrates the mean subtraction process for a dataset  $X$  and its transformation into a centered dataset  $\tilde{X}$ .

**Dataset  $X$ :**

$$X = \begin{pmatrix} 5 & 3 \\ 3 & 2 \\ 1 & 1 \\ -2 & 1 \\ 0 & -4 \\ -1 & 3 \end{pmatrix}$$

**Mean:** mean = (1,1)

**Centered Dataset  $\tilde{X}$ :**

$$\tilde{X} = \begin{pmatrix} 4 & 2 \\ 2 & 1 \\ 0 & 0 \\ -3 & 0 \\ -1 & -5 \\ -2 & 2 \end{pmatrix}$$

**Scatter Matrix:**

$$S = (\tilde{X})^T (\tilde{X})$$

- 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>

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



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

$$T^T T = \Lambda$$

# 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

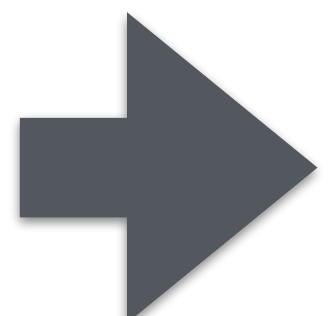
$$T = X P$$


---


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


---


$$T^T T = \Lambda$$



(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)

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

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

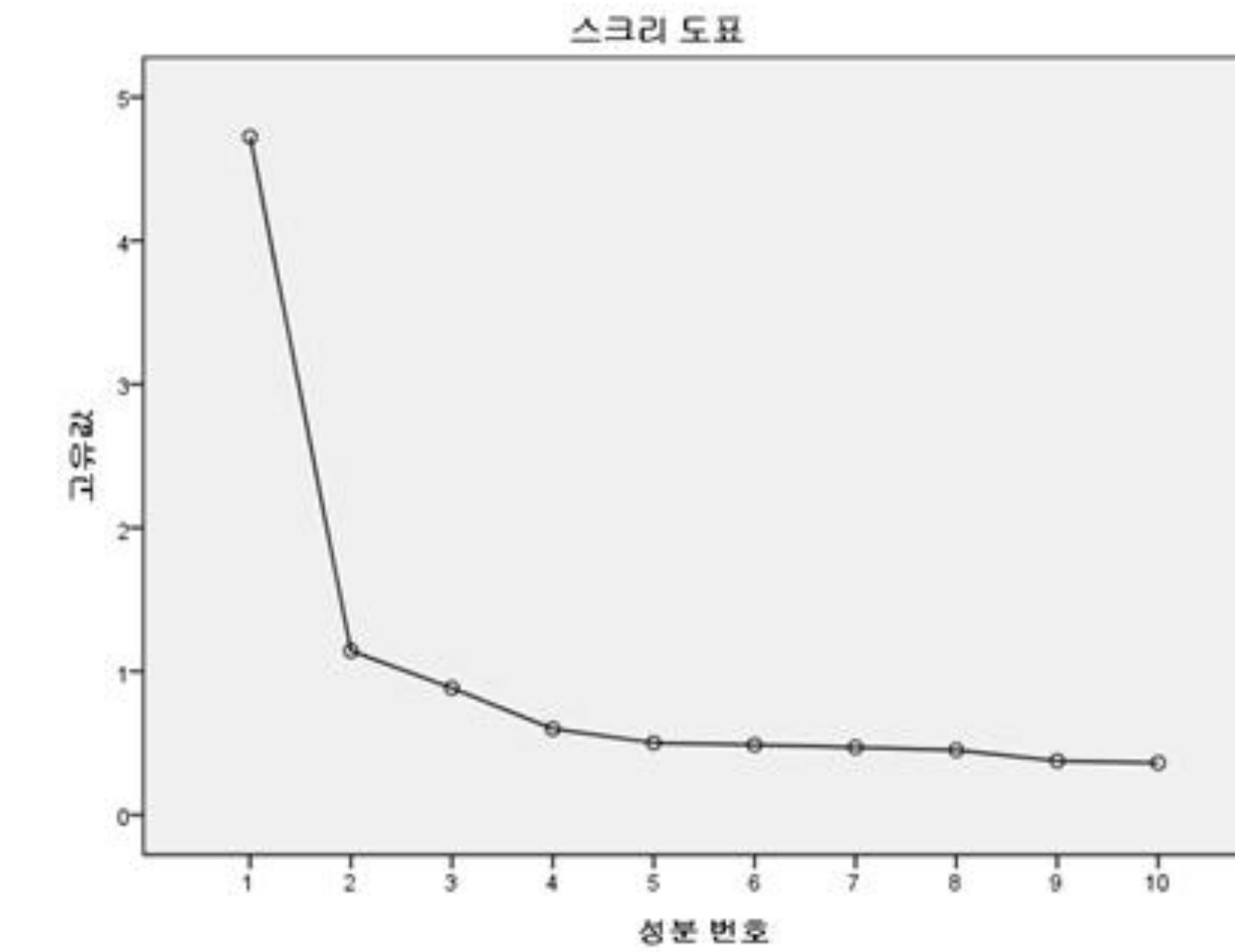
- 각 주성분이 전체 변동의 어느 정도를 설명하는지를 나타내는 척도인 기여율(Contribution) 사용
  - 기여율(Contribution) = (해당 고유값) / (전체 고유값의 합)
- 전체 변동의 70% 이상을 설명할 수 있도록 주성분의 개수를 설정하는 것이 일반적
- 내림 차순으로 고유값을 표시한 스크리 도표(Scree Plot)를 보고 적정한 개수를 선정하기도 함

<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

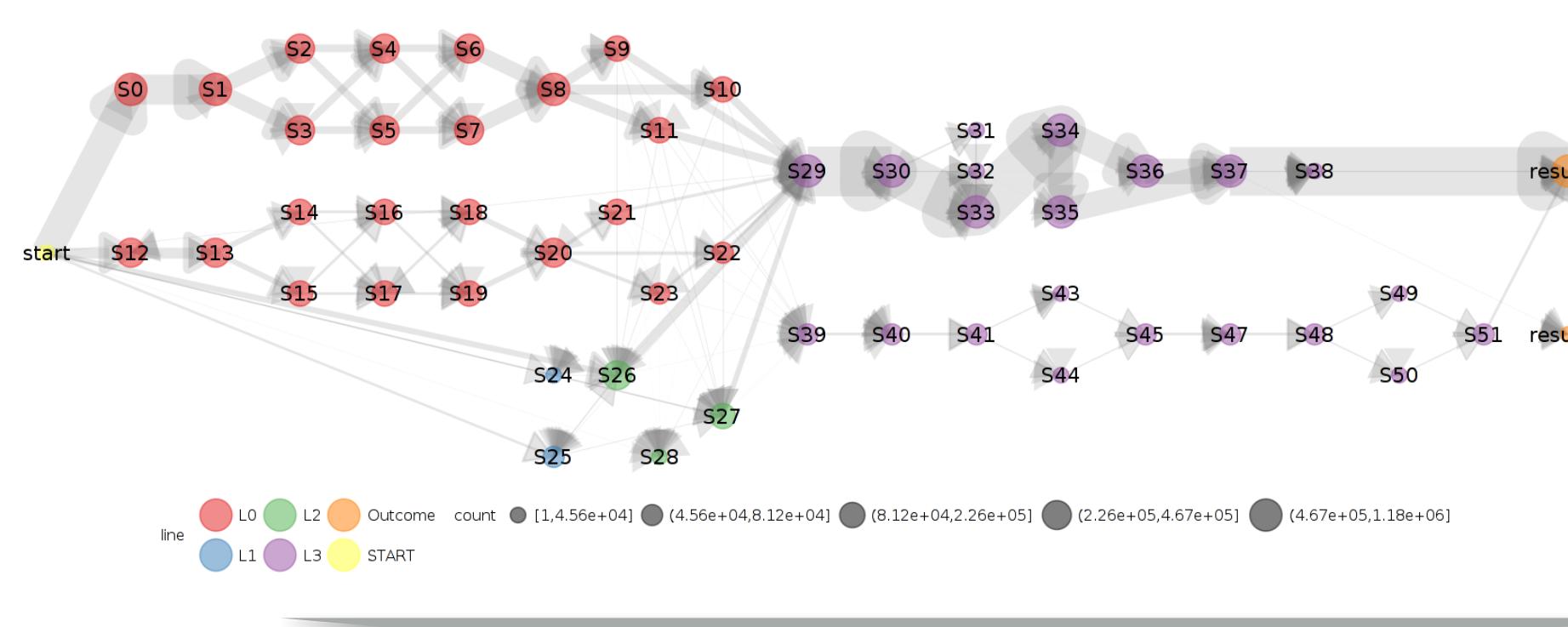
<Scree Plot>



# 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

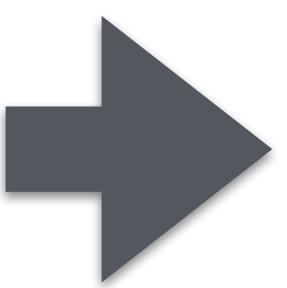
Principal Component Analysis

Classifier

결과 도출

# Application of PCA: EigenFace

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



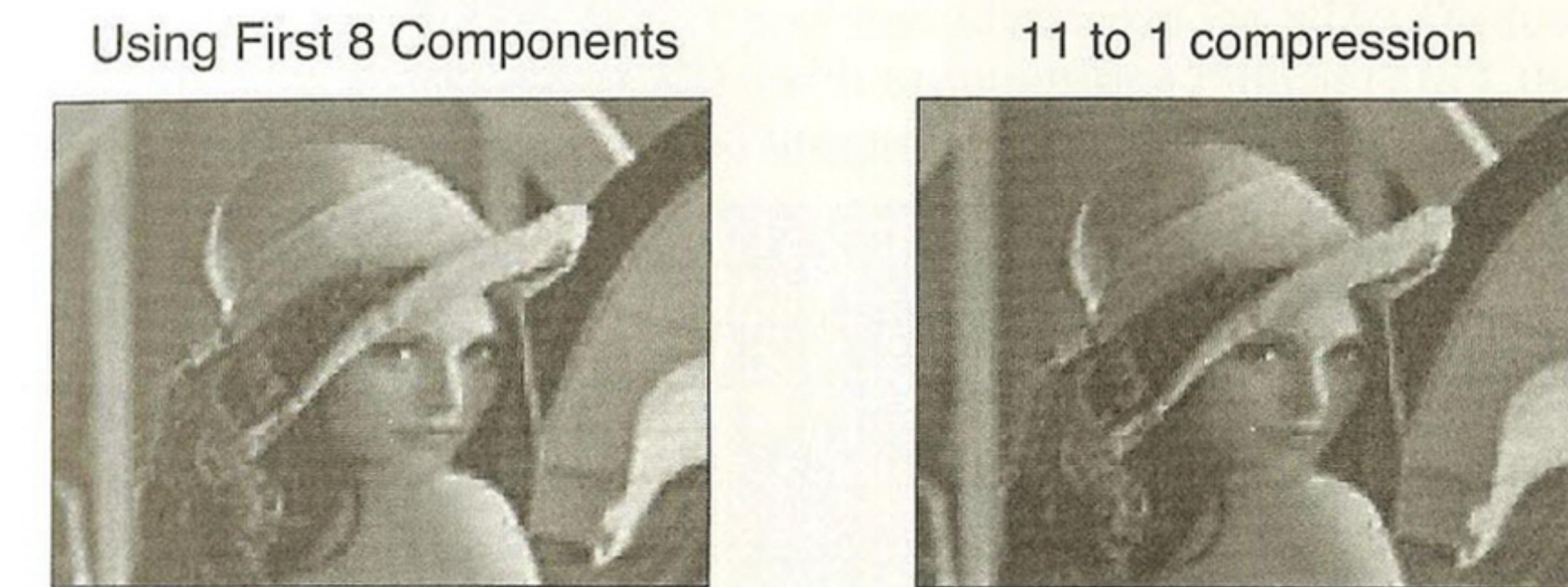
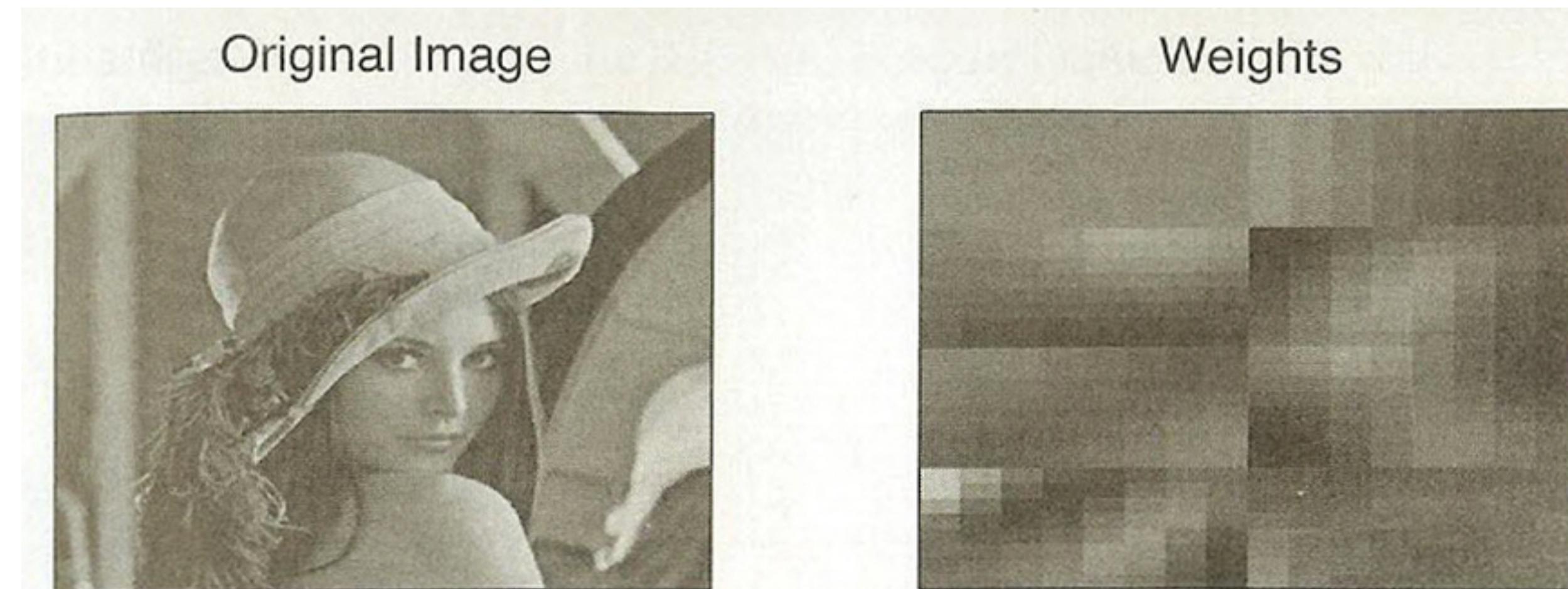
Principal Component Analysis

Reconstruction with  
Principal Component

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

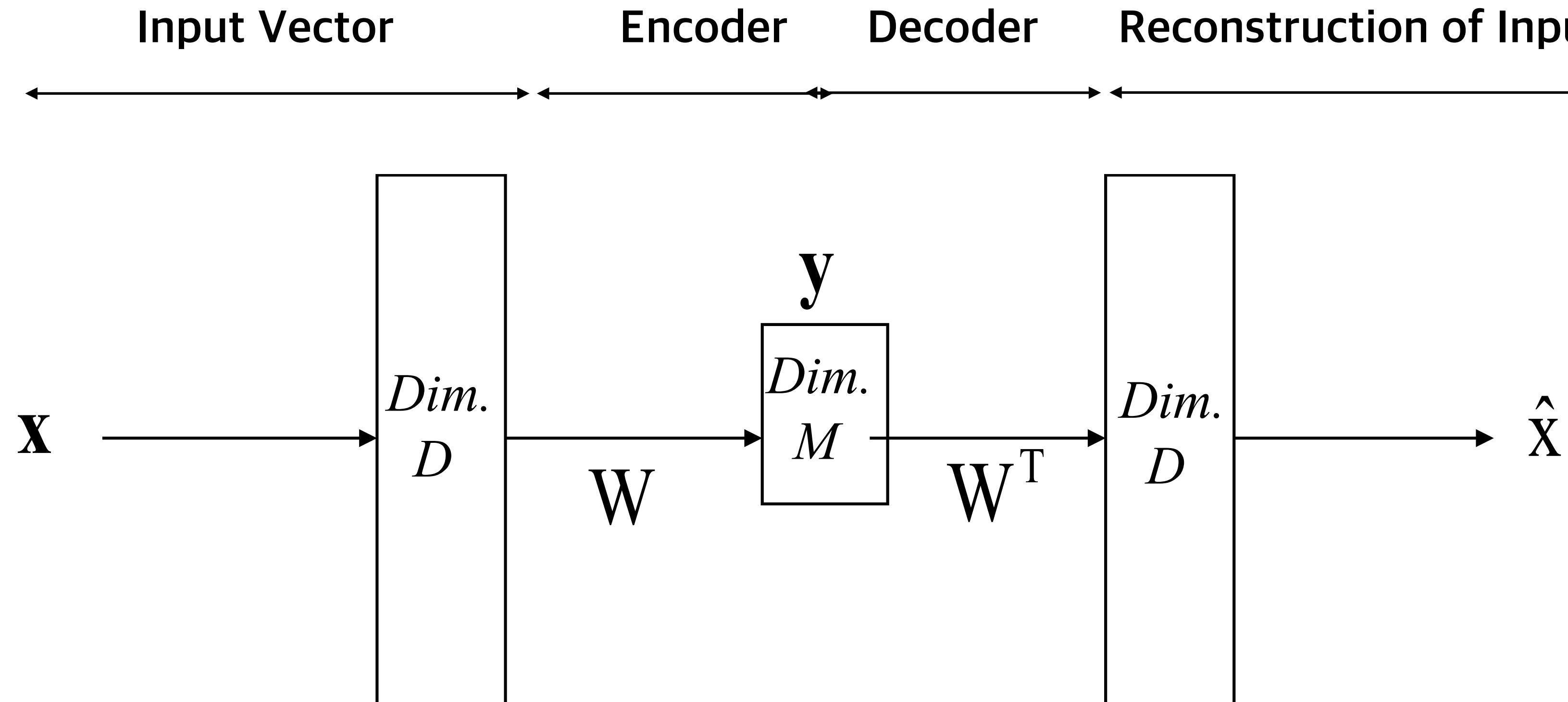
# Application of PCA: Image Compression and Reconstruction

- 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

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

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

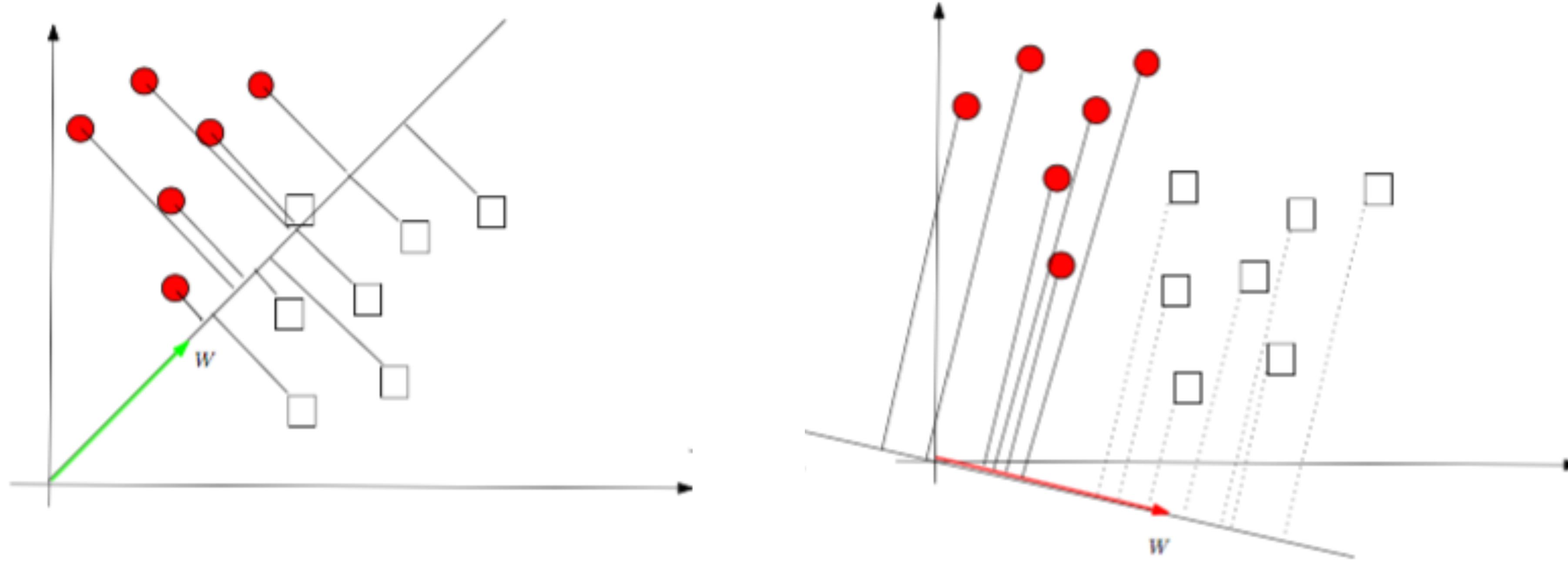
# Discriminant Analysis

## 3. Linear

# 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)

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

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

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

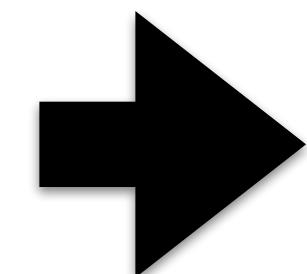
$$\text{Scatter : } \tilde{s}_i^2 = \sum_{y \in Y_i} (y - \tilde{m}_i)^2$$

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

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

Minimize:

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



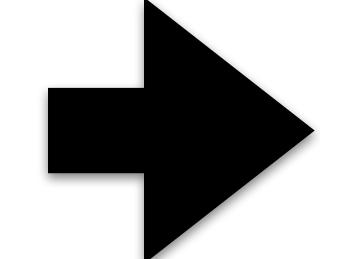
- Linear mapping 이 후, 클래스내 분산은 기존 클래스내 분산과 linear mapping, w로 표현 가능

### 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

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

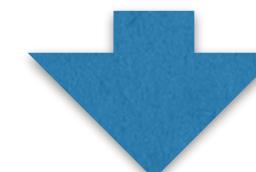
$$\begin{aligned} J(\mathbf{w}) &= \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}} = \frac{\alpha}{\beta} \\ \frac{\partial J(\mathbf{w})}{\partial \mathbf{w}} &= \frac{2S_B \mathbf{w}}{\beta} - \frac{2\alpha}{\beta^2} S_w \mathbf{w} = \mathbf{0} \end{aligned}$$

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

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

- 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 :  $y = \mathbf{w} \cdot \mathbf{x}$

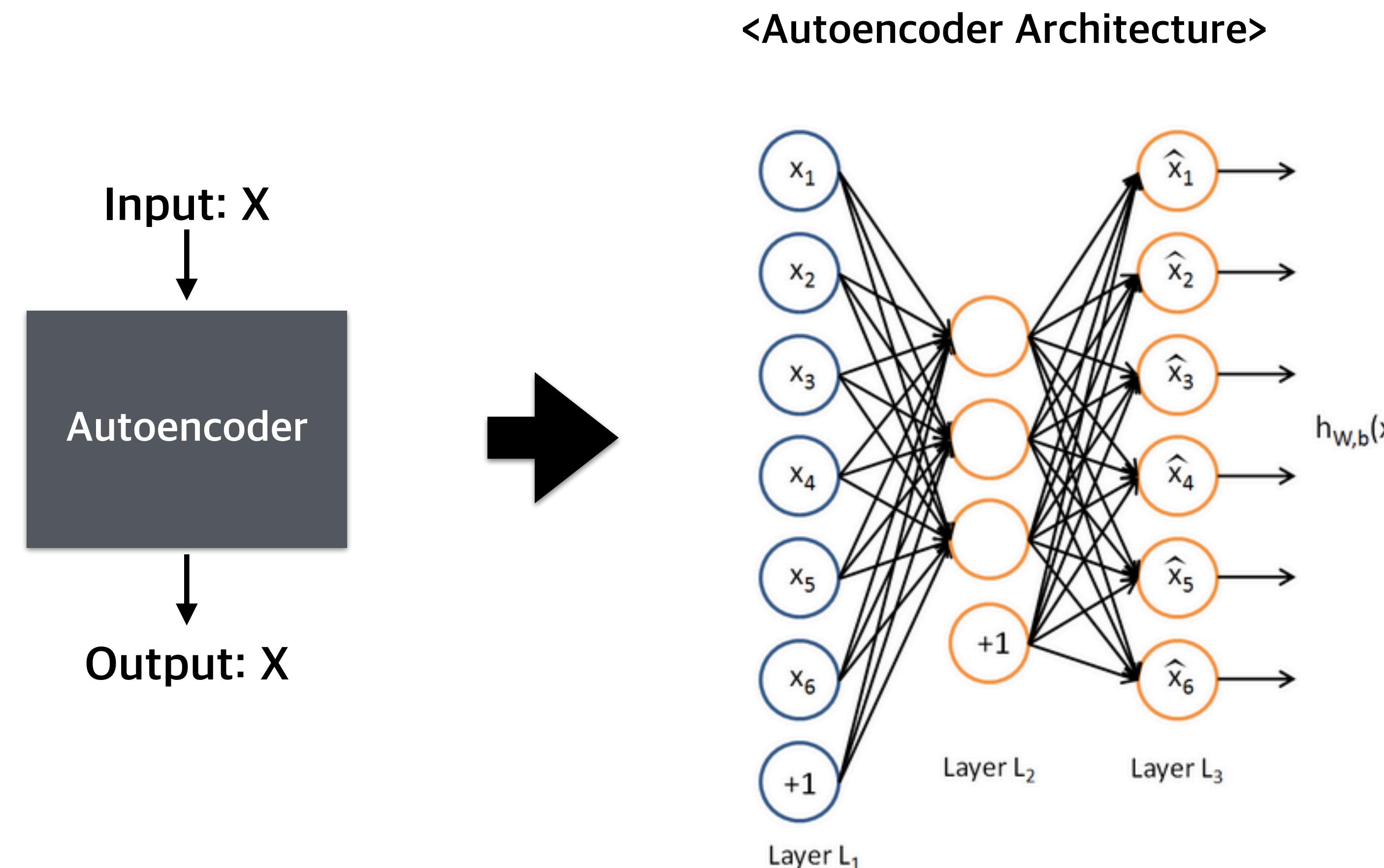
$$\mathbf{w} = S_w^{-1} (\mathbf{m}_1 - \mathbf{m}_2)$$

- ❑ 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

# 4. Autoencoder

# Autoencoder

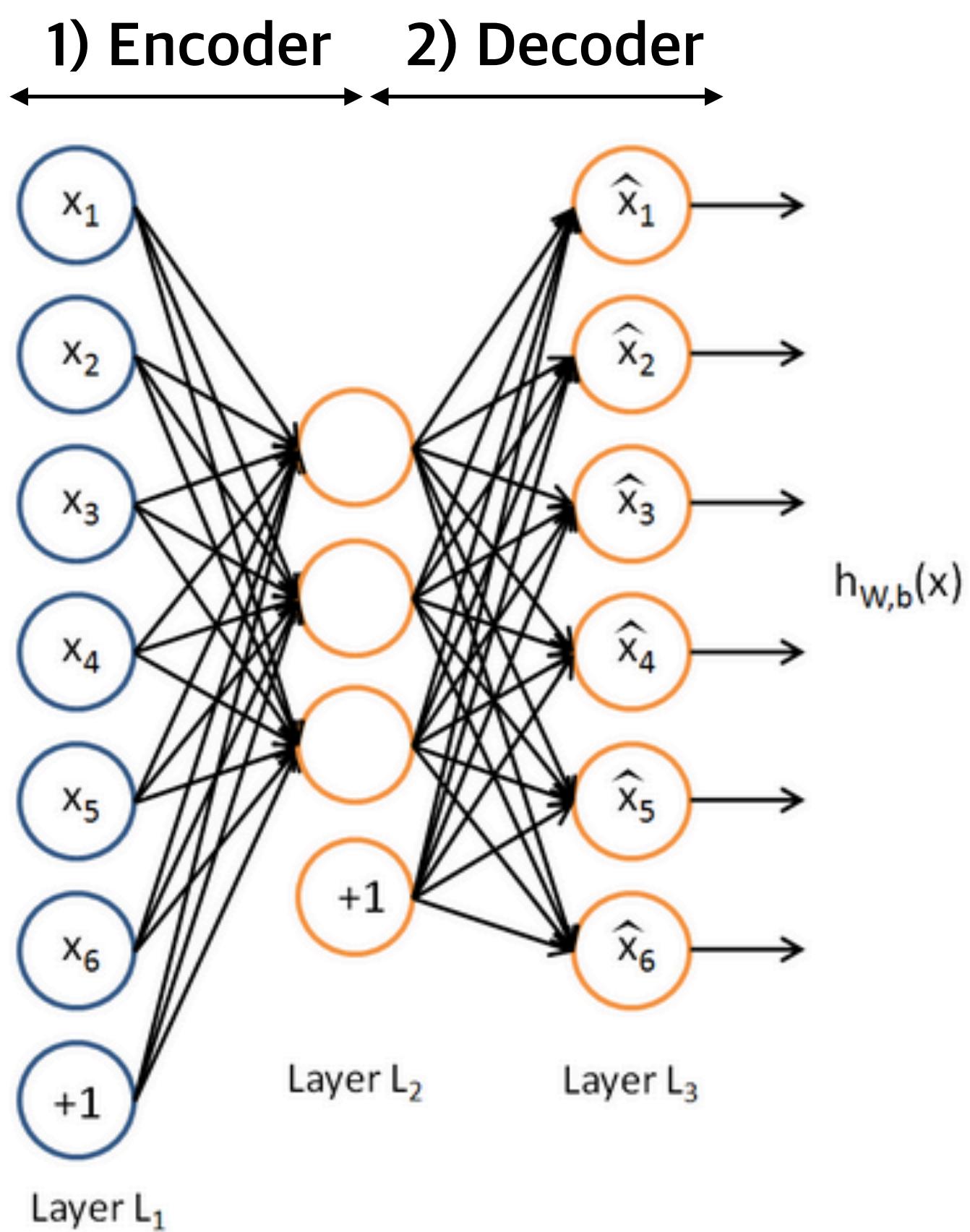
- ❑ Neural Network를 이용한 Unsupervised Learning 방식으로 Encoder-Decoder 구조를 가진 Neural Network
- ❑ Input  $X$ 에 대하여 Output을 Input과 동일하게 설정한 후, 이를 추정하는 Neural Network 모델을 학습
- ❑ Input 데이터  $X$ 를 잘 설명하는 **hidden representation (latent variable, 잠재 변수)**를 얻는 것을 목적으로함



# 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

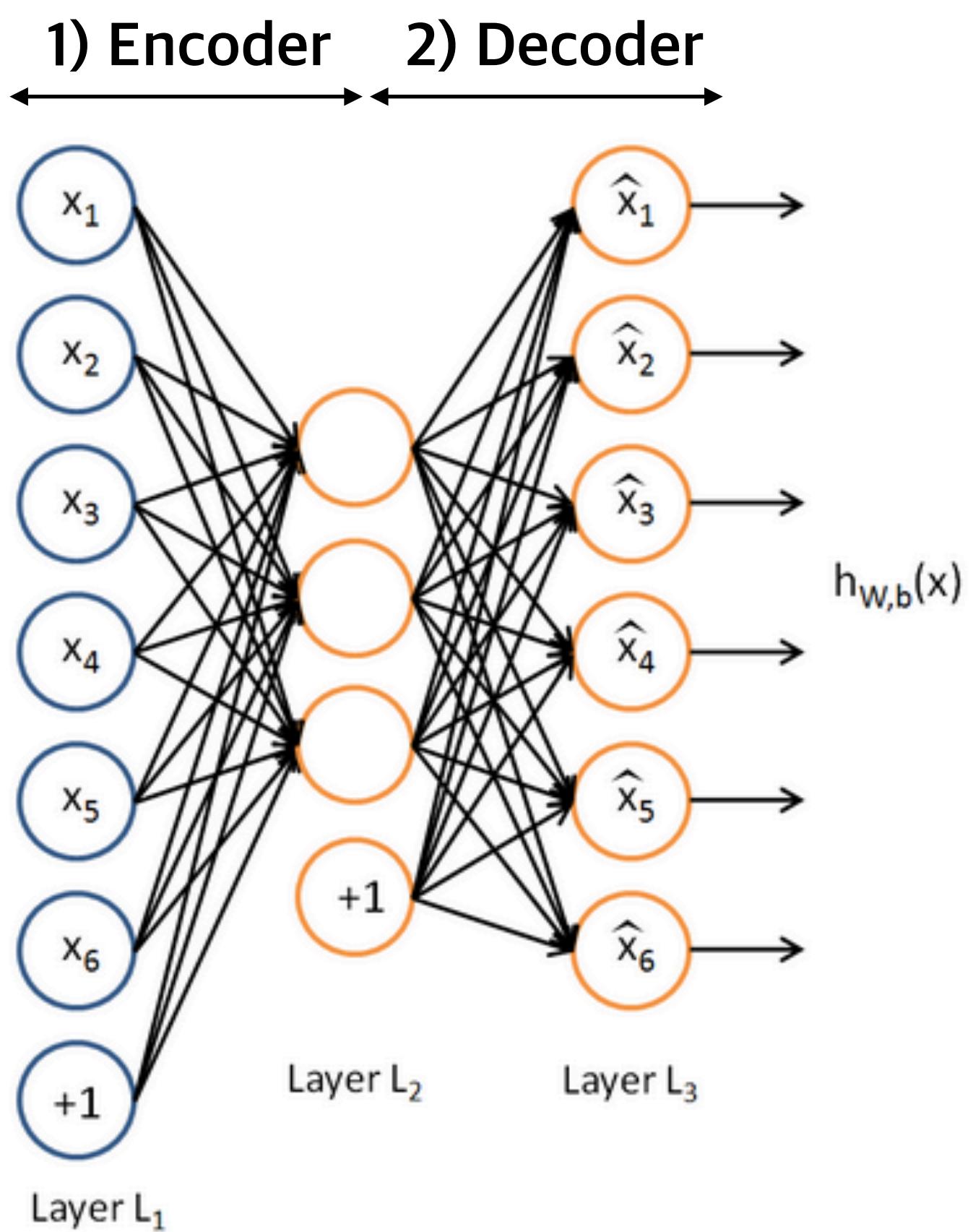
## 2) Decoder

- 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

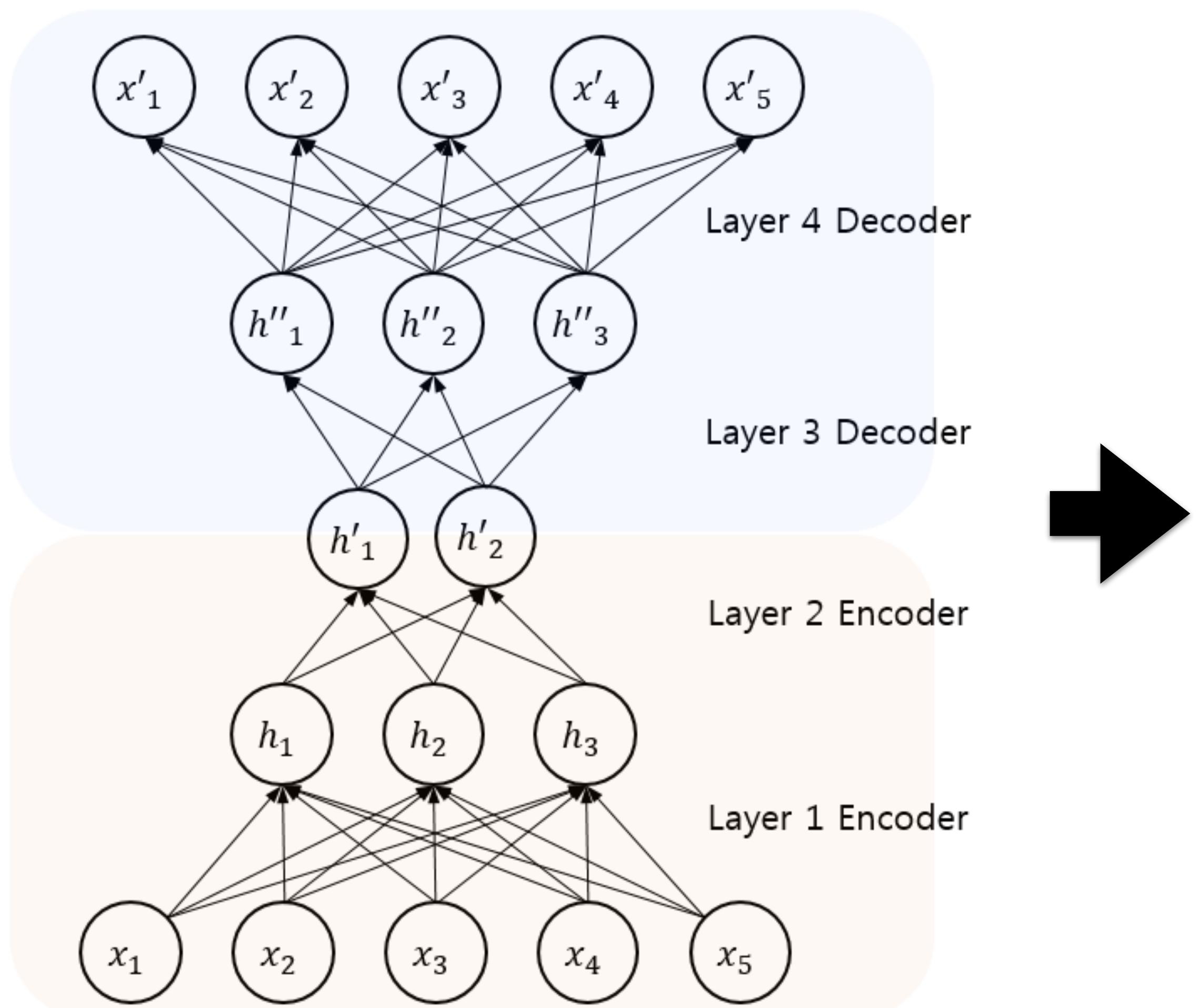
$$\begin{aligned} W, W' &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(x^{(i)}, \hat{x}^{(i)}) \\ &= \underset{W, W'}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\hat{x}^{(i)} - x^{(i)})^2 \end{aligned}$$

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

A) Use deeper layer

# Stacked Autoencoder

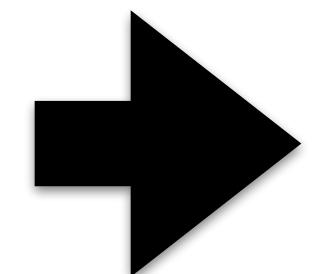
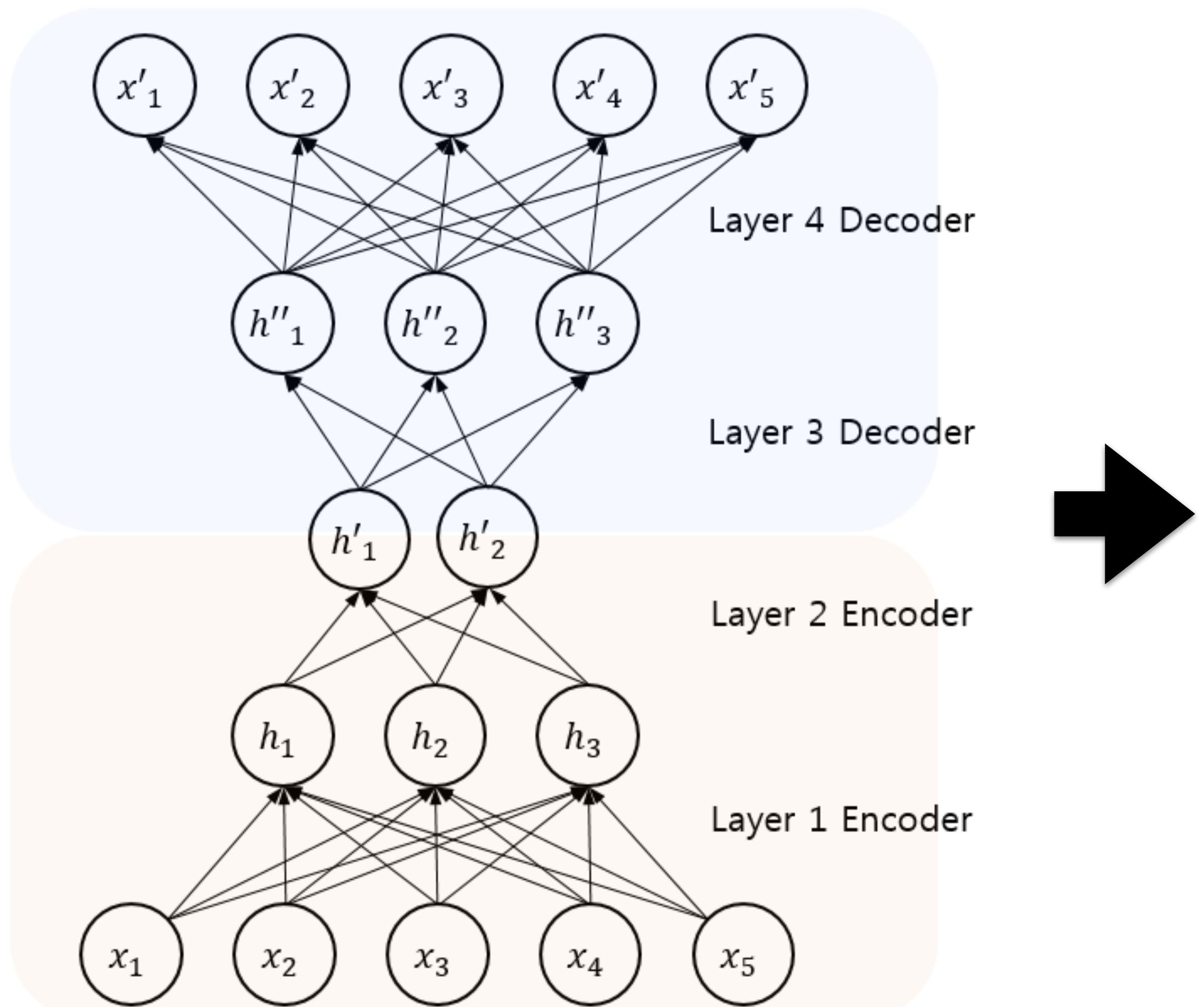
- Encoder, decoder가 여러 층으로 구성되어 있는 Autoencoder
- 기존 Autoencoder보다 더 복잡한 정보를 포함한 latent representation을 얻을 수 있음



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



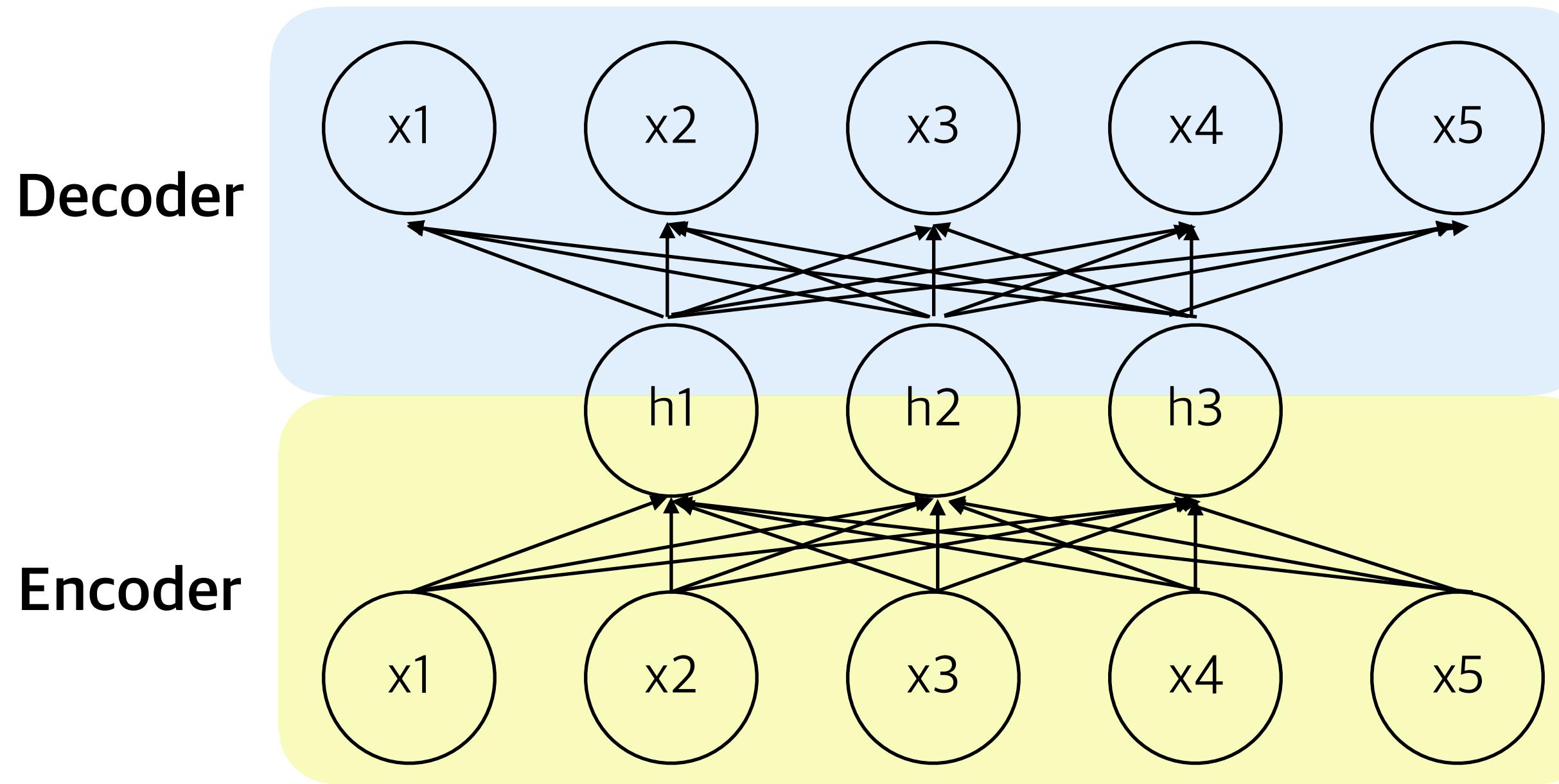
Dimensionality Reduction가 아니더라도  
(Stacked) Autoencoder는  
NN의 중요한 모델 중 하나! (Why?)

(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

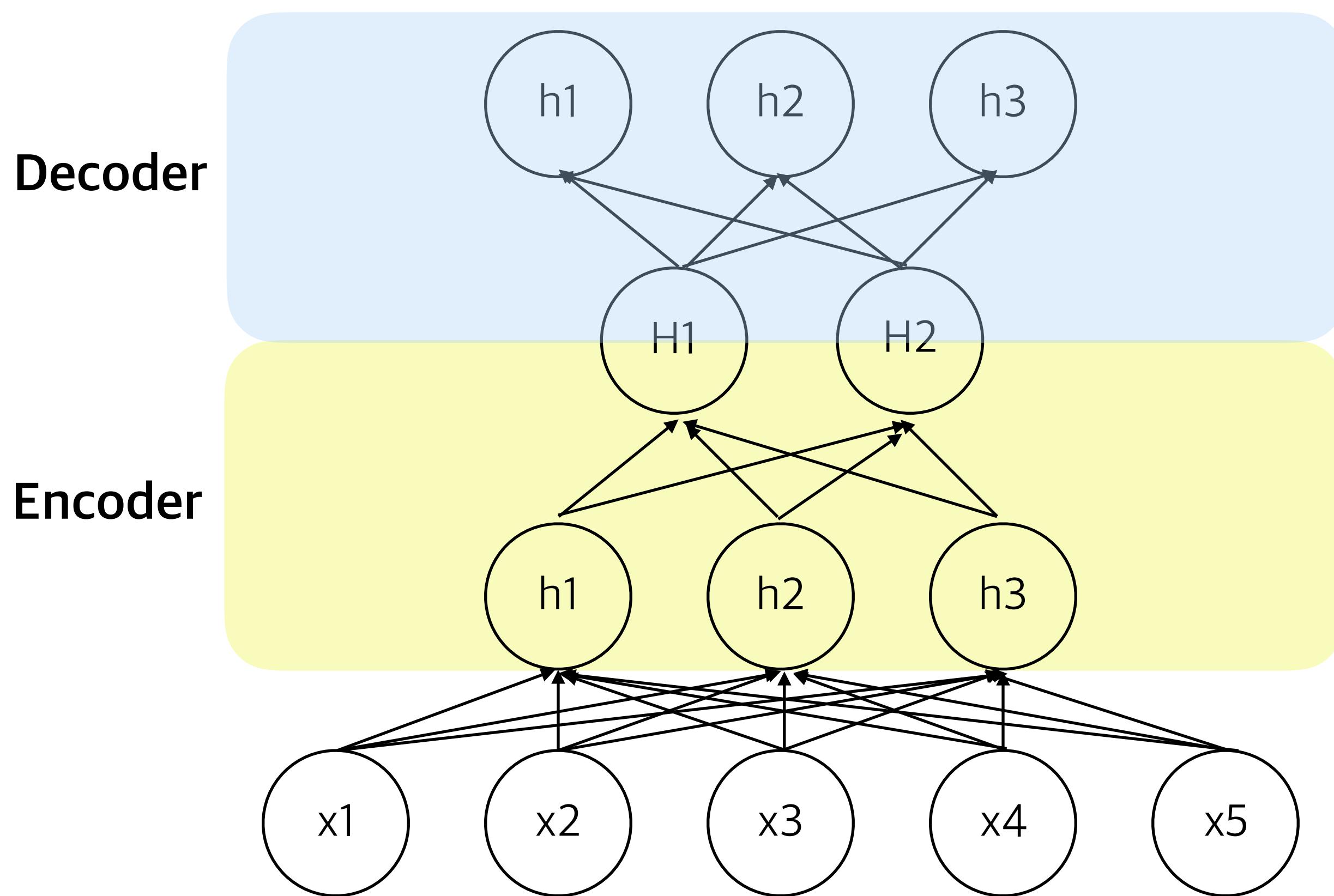
<Pre-training for Initialization>



## 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

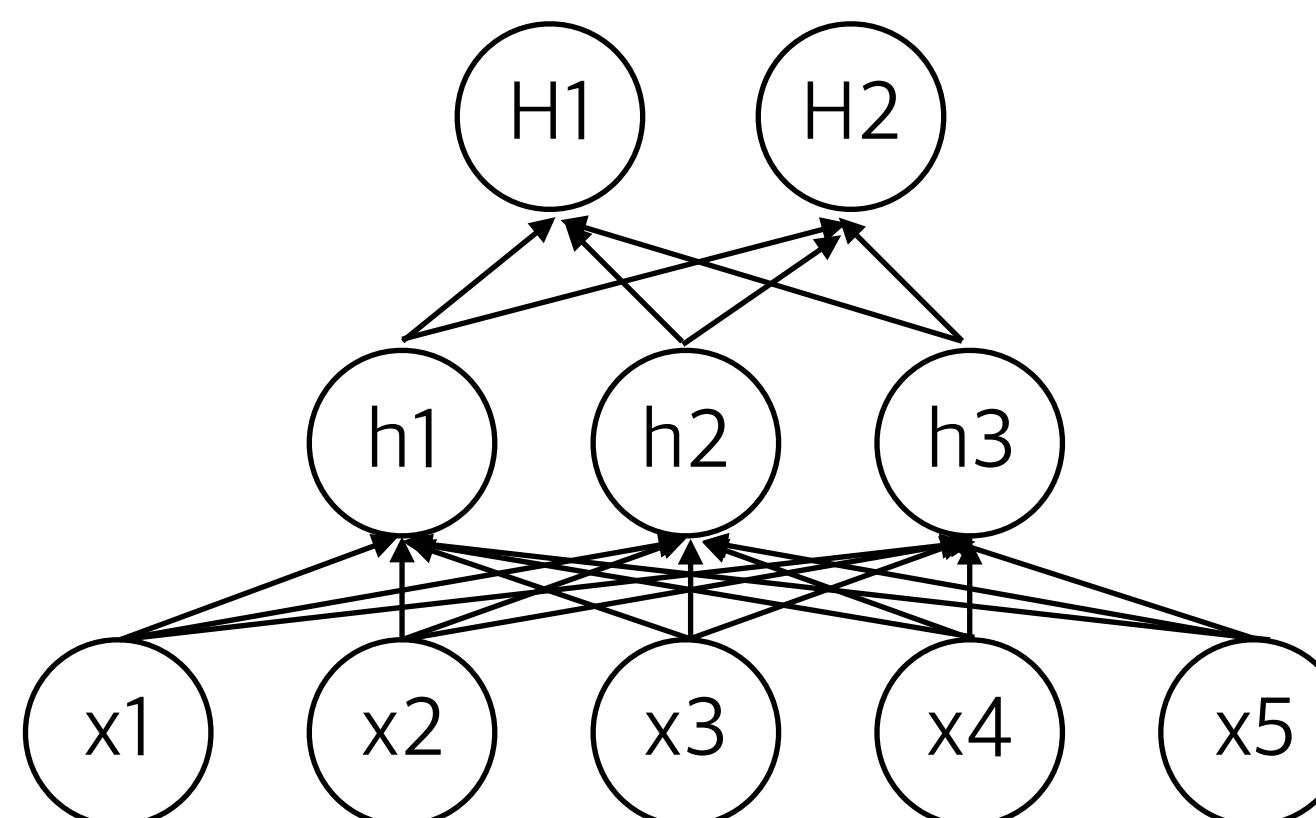
<Pre-training for Initialization>



# 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>



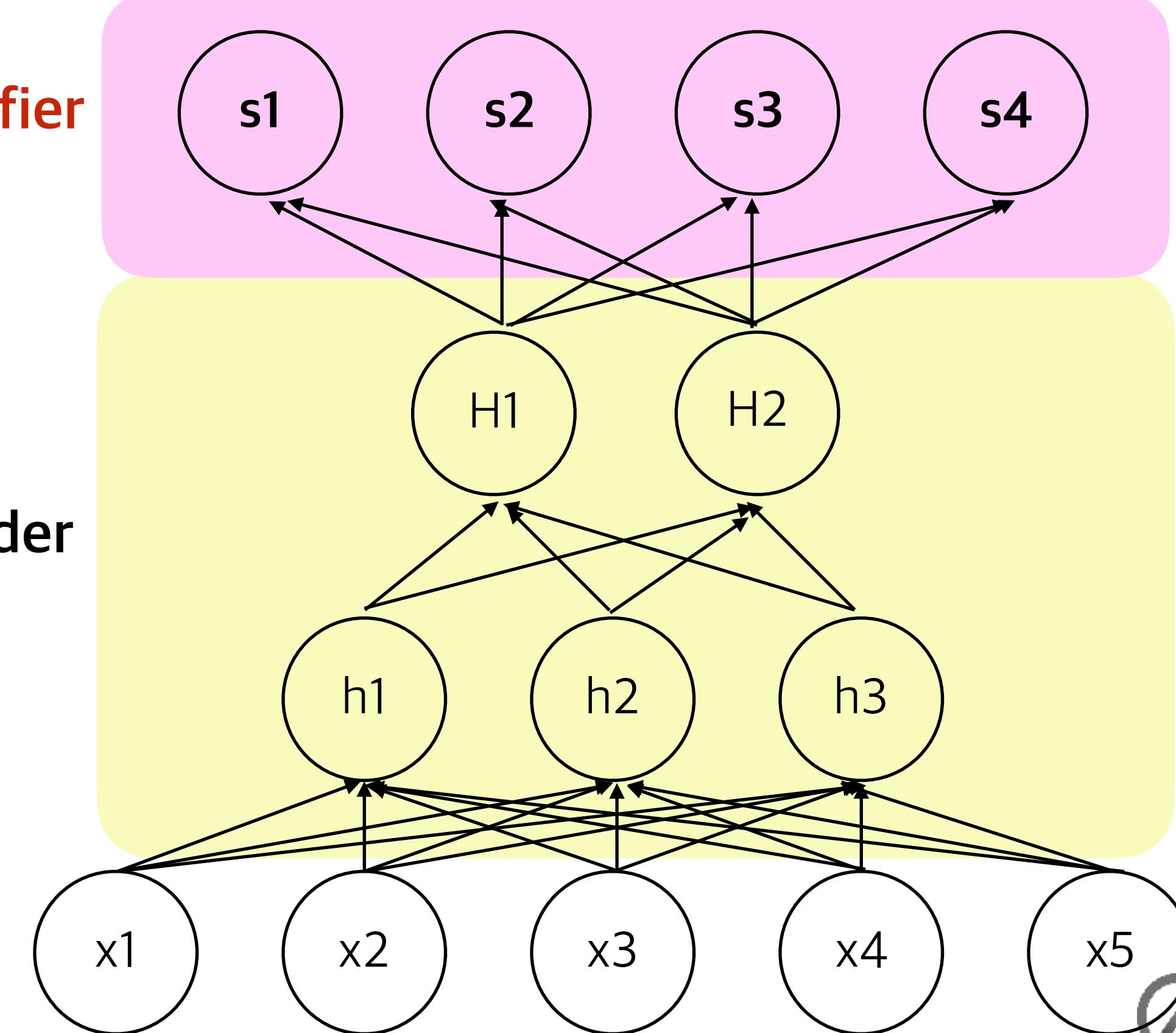
Encoder  
Initialization

Classifier

Encoder

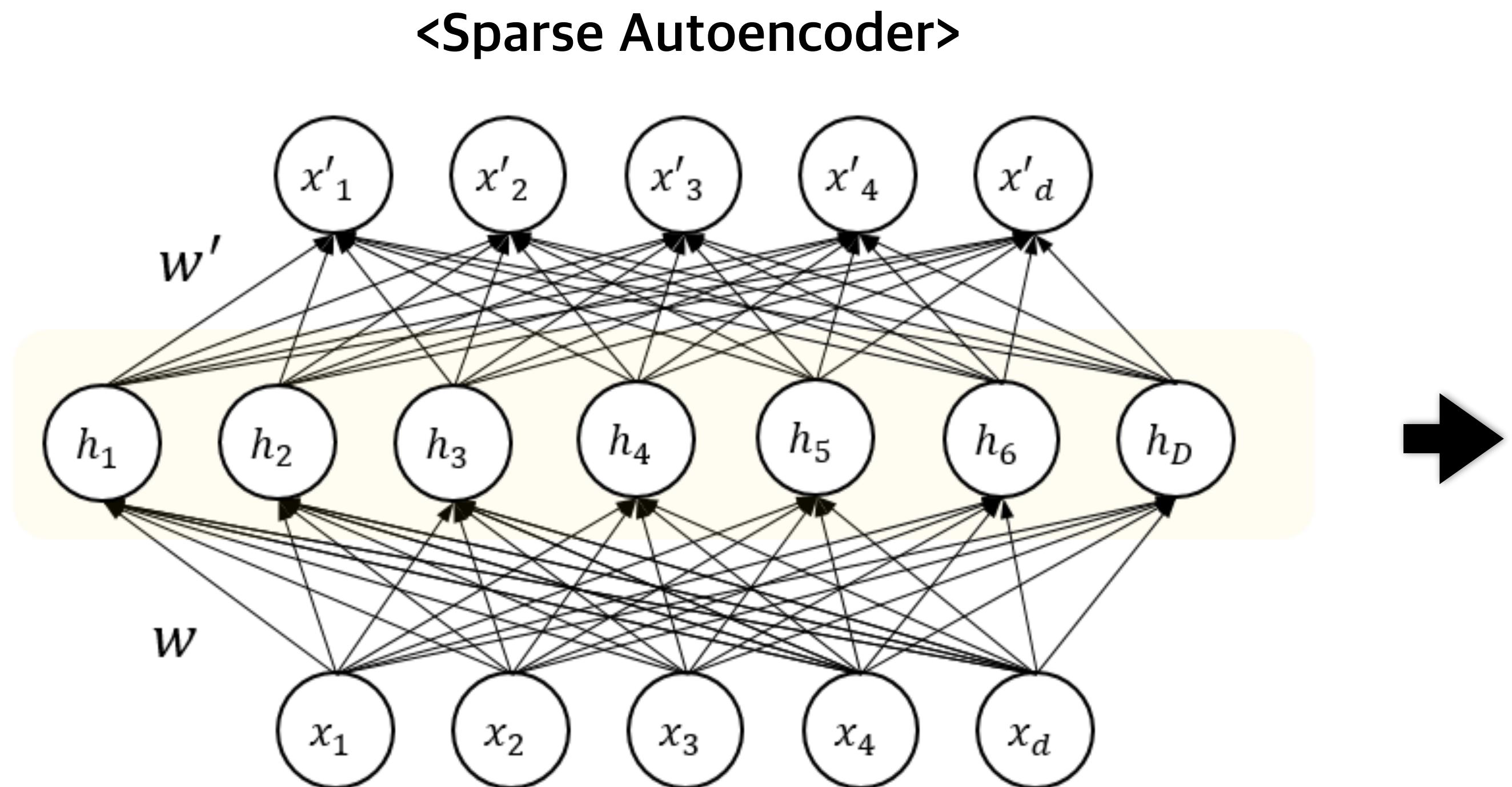
<Fine Tuning for Classification>

Learning



# 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



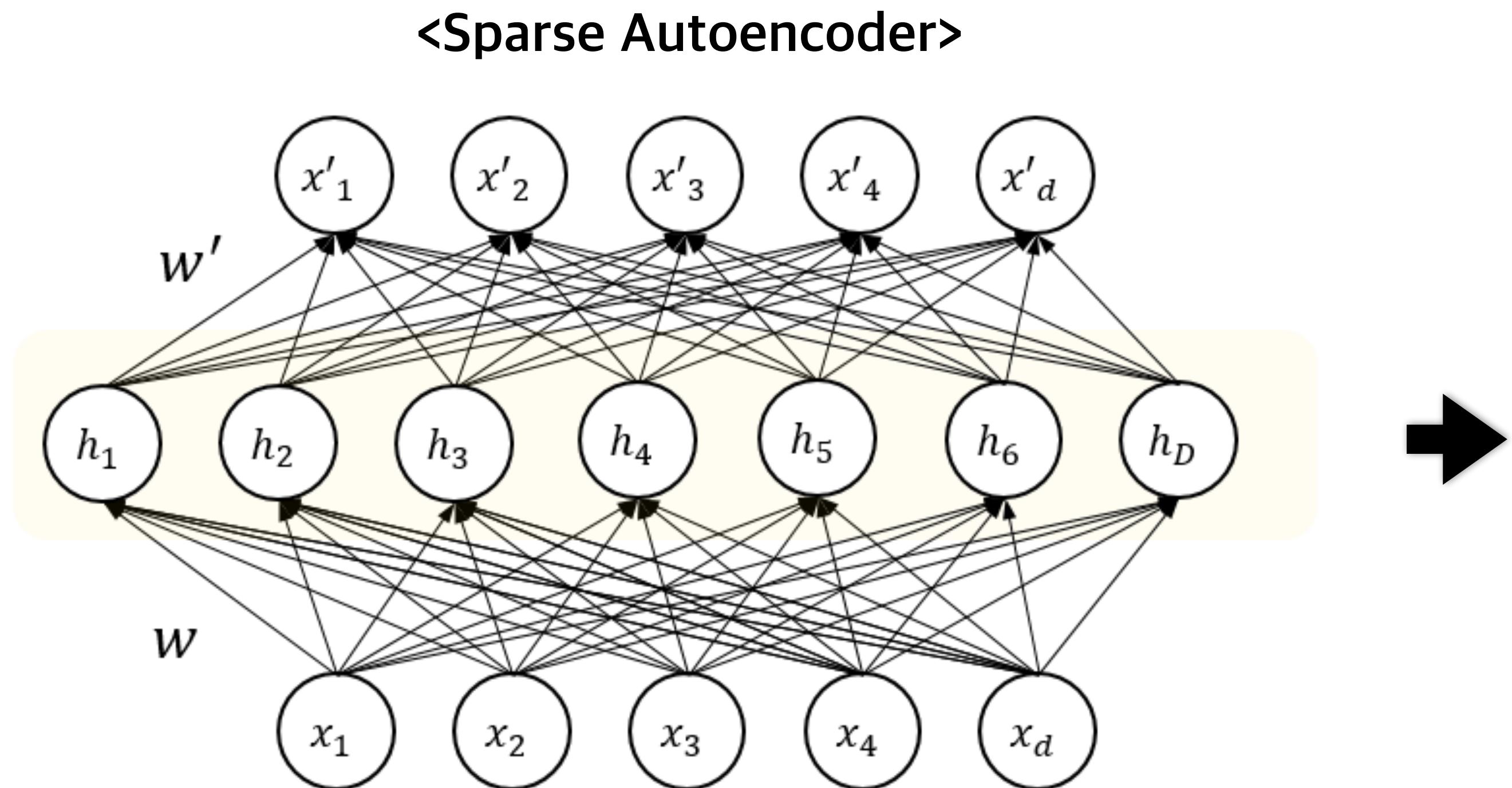
**<Activation Sparsity>**

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

m개의 input 데이터에 대하여  
j 번째 hidden activation 값 평균

# Sparse Autoencoder

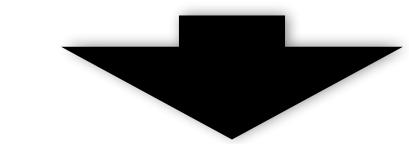
- Hidden Layer 내 뉴런의 수가 Input 데이터의 차원의 수보다 많은 구조의 Autoencoder
- Hidden Neuron의 Activation 정도를 제한하여 **Sparsity**를 유지하고, 이를 통해 Input의 특수한 구조를 파악함



**<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

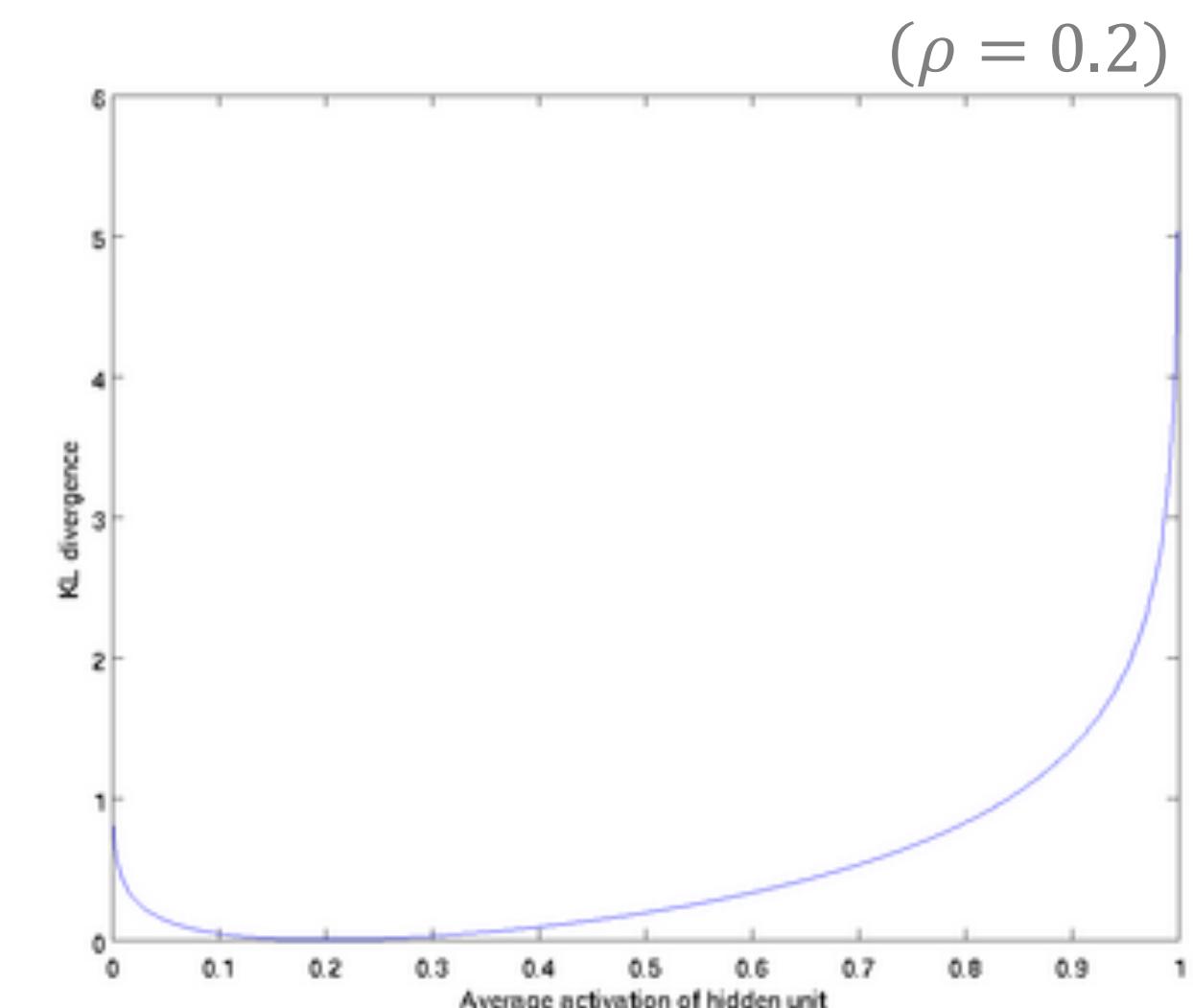
$$W, W' = \underset{W, W'}{\operatorname{argmin}} \left\{ \frac{1}{n} \sum_{n=1}^n L(x^{(i)}, \hat{x}^{(i)}) + \lambda \sum_{j=1}^{h_d} KL(\rho \parallel \hat{\rho}_j) \right\}$$

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>



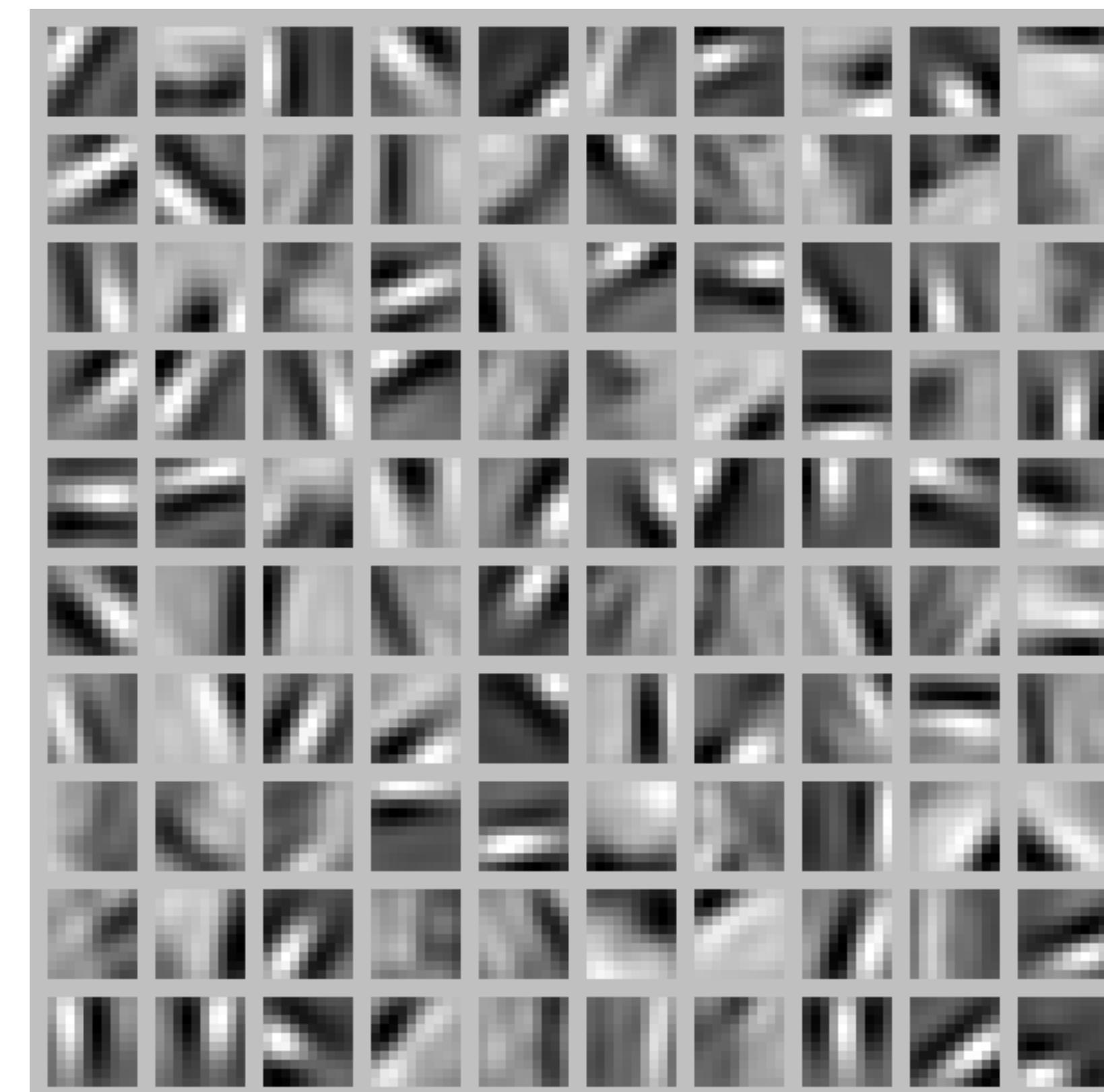
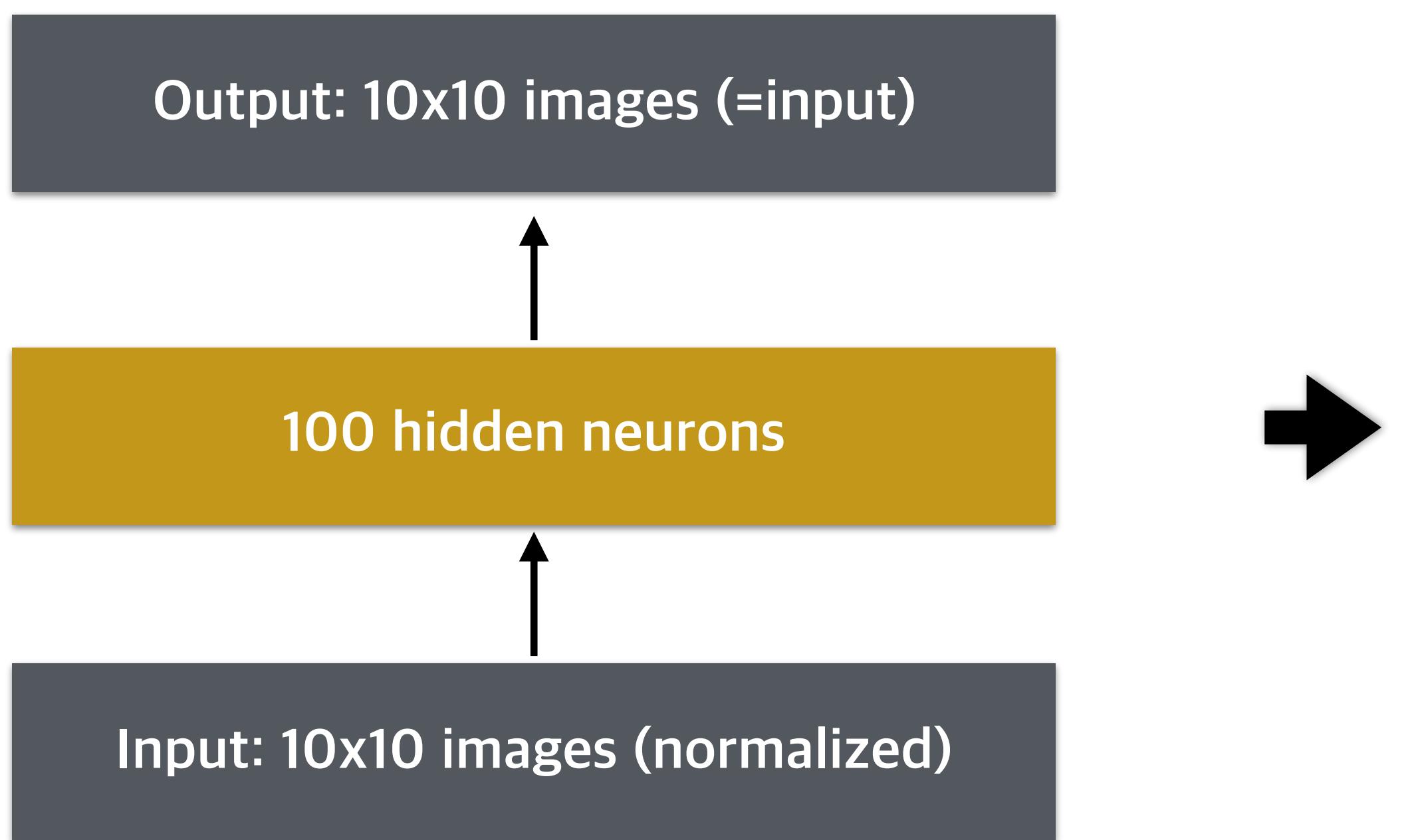
- 설정한 sparsity 값과 차이가 커질 경우 증가
- Sparsity 값을 평균으로 갖는

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

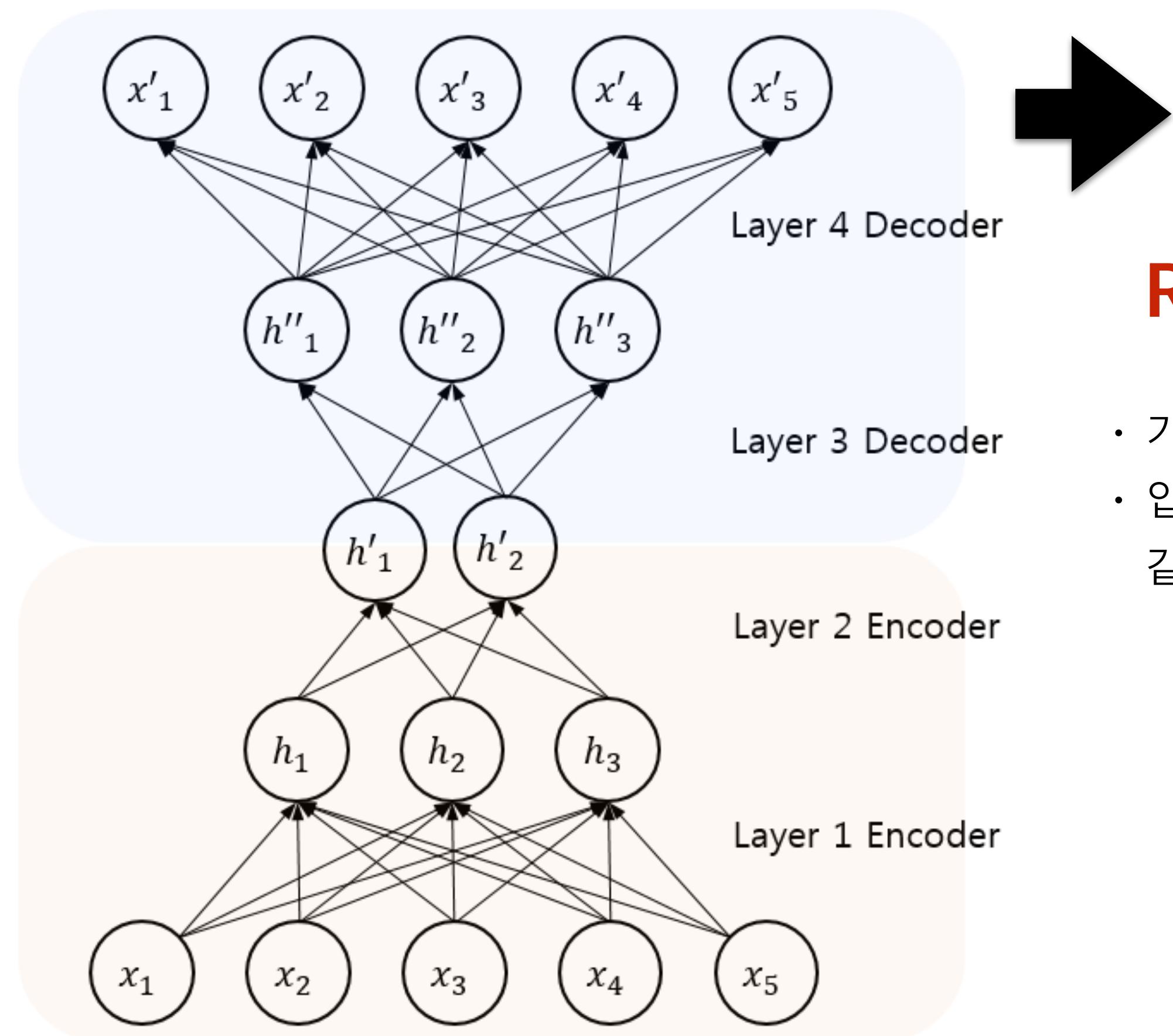
<Input Images Maximizing Each Hidden Neurons>



# One more important Autoencoder

# New Criteria for Autoencoder

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



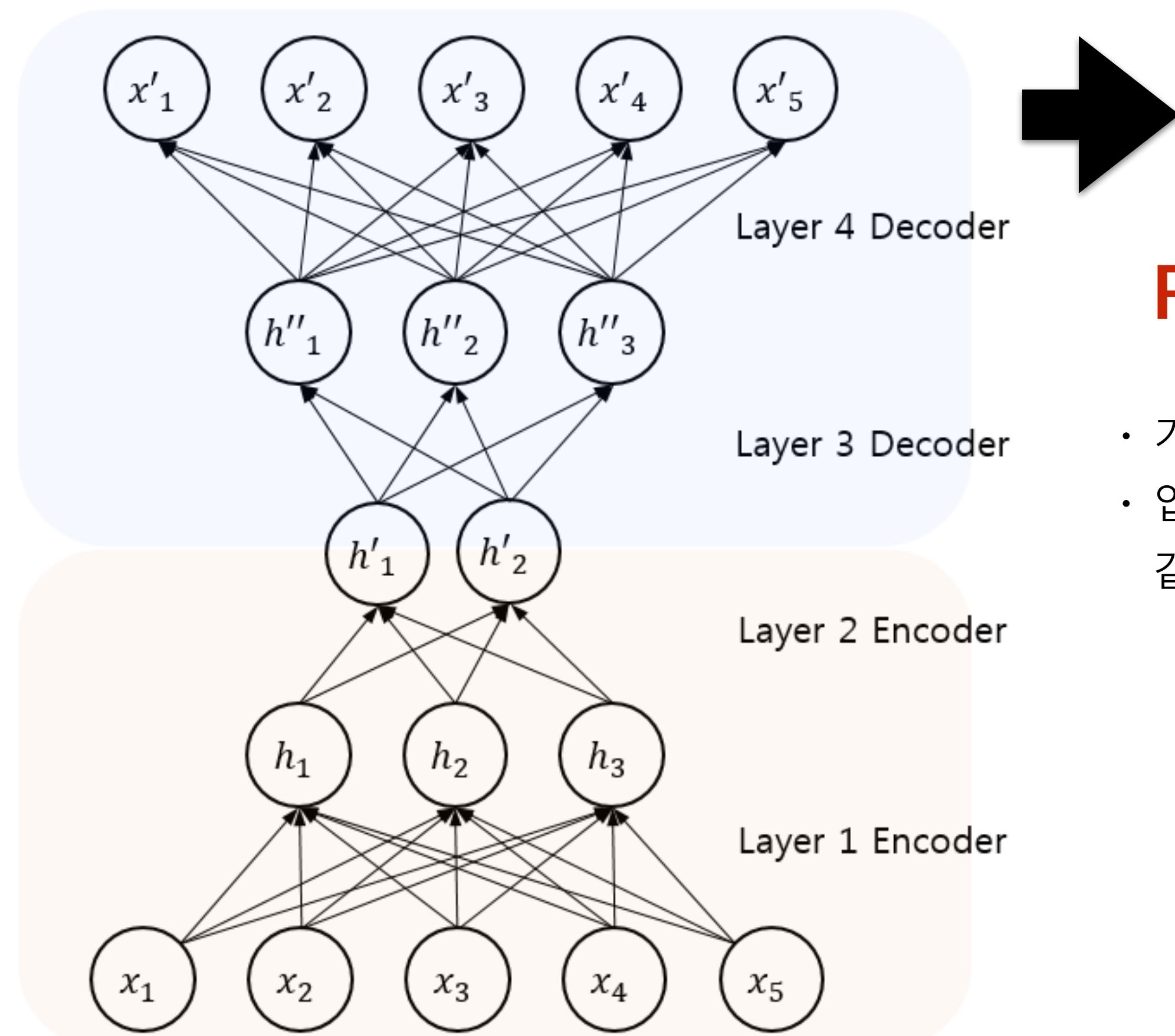
New Criteria:

## 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)**

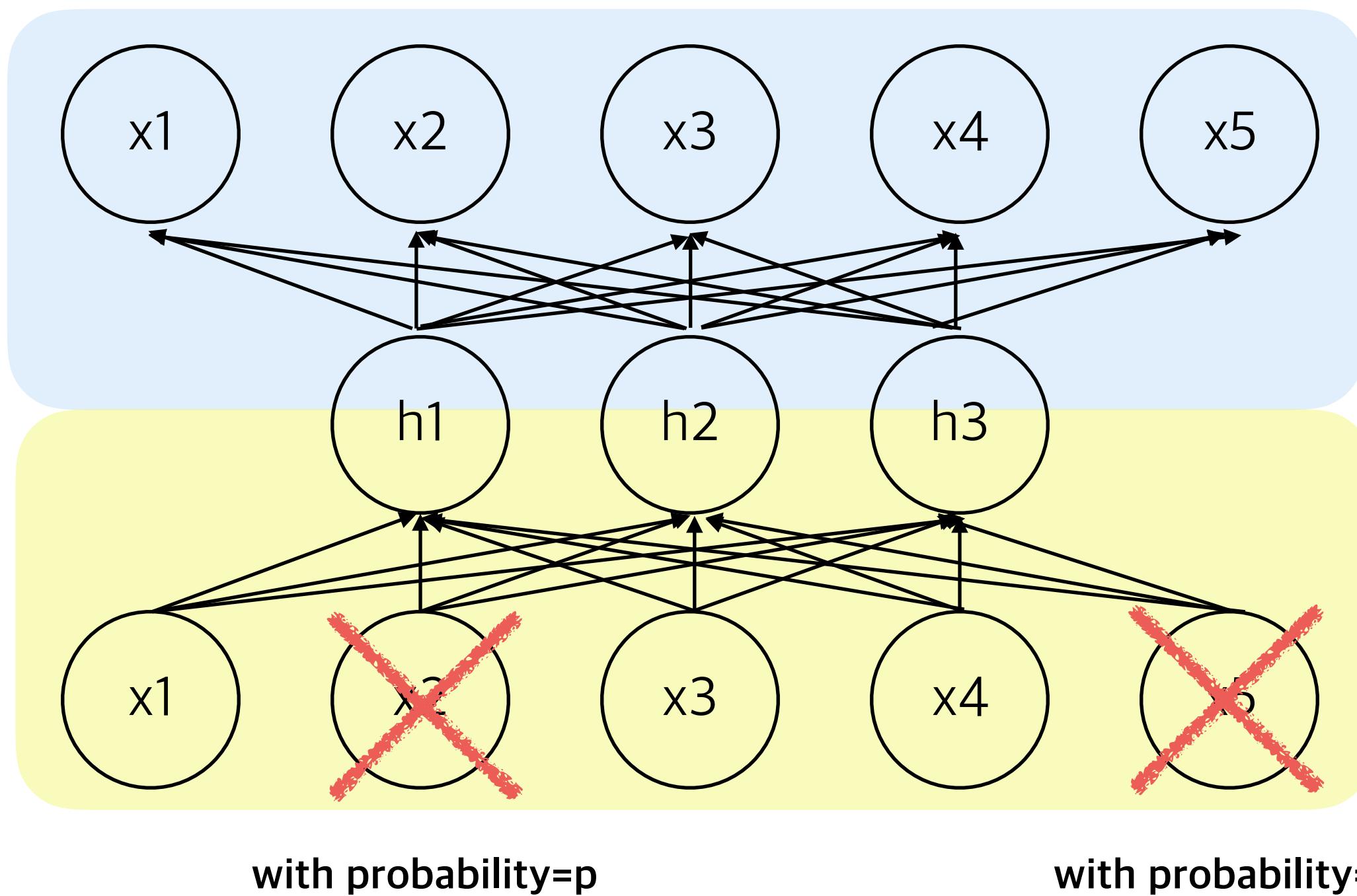
Denoising Autoencoder

# 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

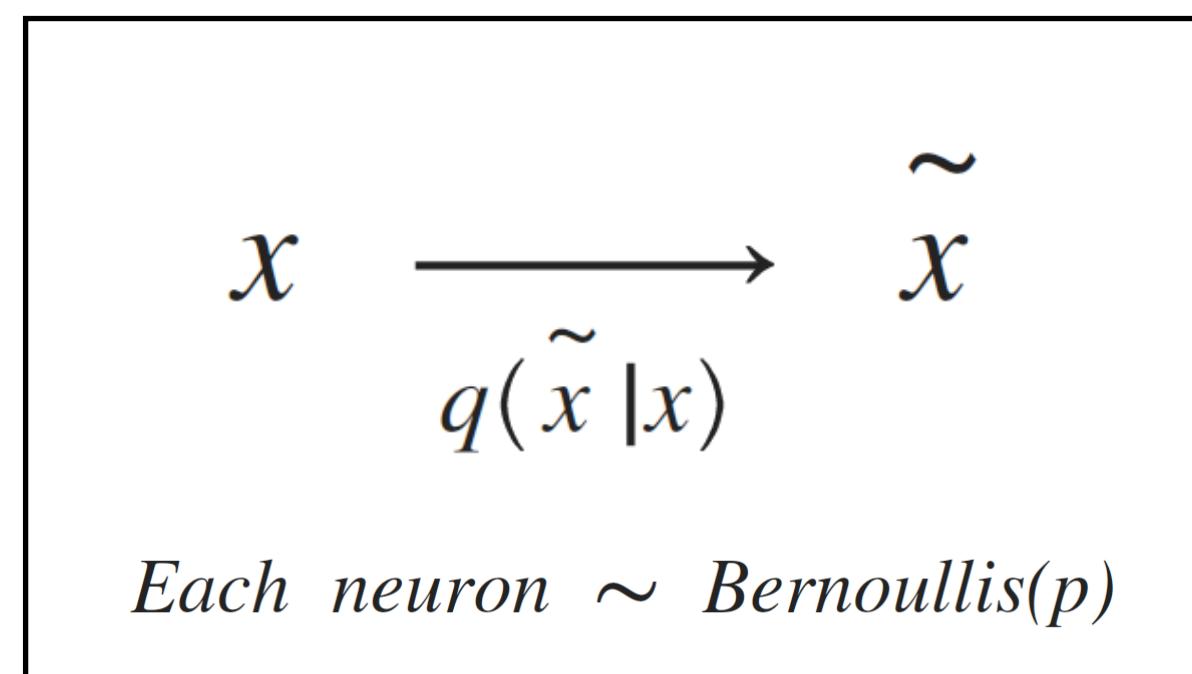


Encoder

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

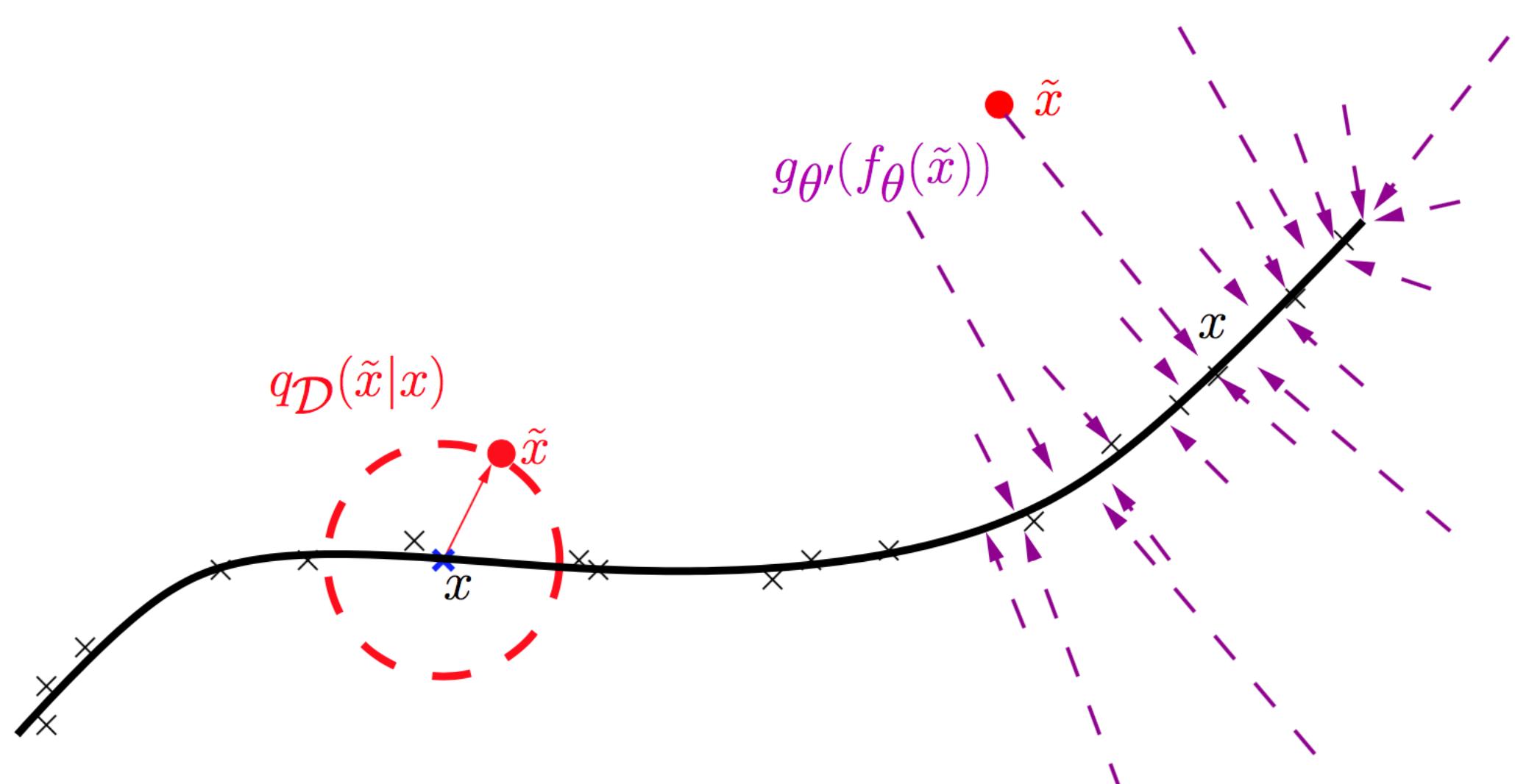
## □ Denoising Autoencoder의 학습과 그 결과를 1) Manifold Learning, 2) Generative Model의 관점에서 해석 가능

1) Manifold Learning: AE는 training 데이터의 manifold를 학습하고,

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

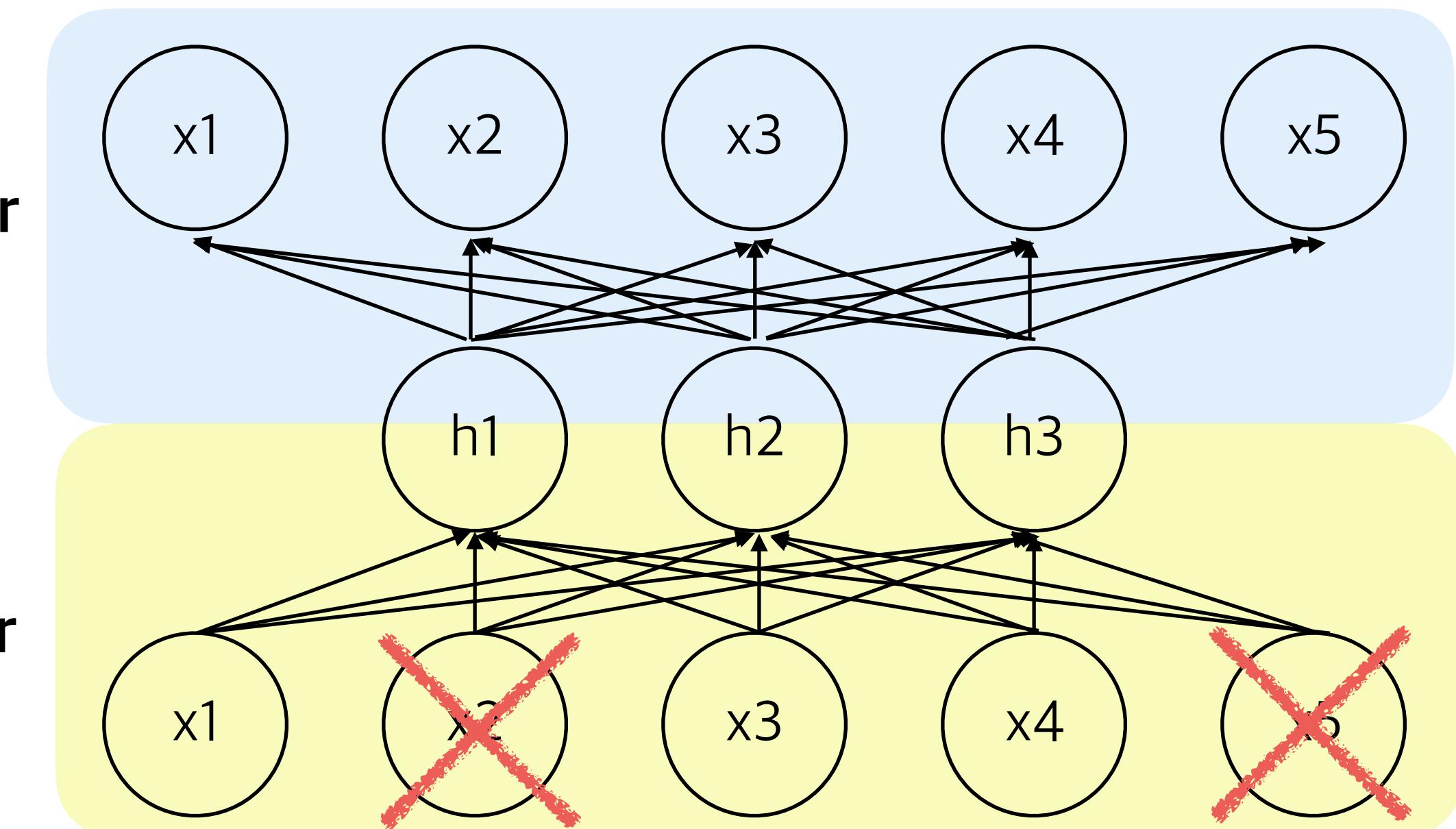
2) Generative Model: 특정 generative model의 variational bound를 최대화하는 것으로 Loss Function을 해석 (자세한 내용 생략)

<Manifold Learning Perspective>



<Generative Model Perspective>

Decoder



Encoder

with probability= $p$

with probability= $p$

**exam**

## 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