

KCC 2019 tutorial review

오토인코더 딥 러닝 모델 기초 이론 및 활용 실습

강연자: 최재영 교수님(한국외대)

발표자: 남궁 영

강연자 소개



최재영 교수
(한국외대)

2011 KAIST 전기및전자공학과 박사
2008-2012 토론토대학 연구원
2012-2013 펜실베이니아대학 연구원
2013-2014 삼성전자 책임연구원
2014-2016 중원대학교 의료공학과 조교수
2016-현재 한국외국어대학교 컴퓨터.전자공학부 조교수
관심분야 : 딥 러닝 기반 안면인식, 머신러닝, 패턴인식, 영상처리

- ❖ Prof. Jae Young Choi
- ❖ Pattern Recognition and Machine Intelligence Lab. (PMI)
- ❖ Hankuk University of Foreign Studies

강의 소개

강연제목

오토인코더 딥 러닝 모델 기초 이론 및 활용 실습

강연요약

딥 러닝 분야의 대표적인 학습모델 중 하나인 오토인코더를 강의한다. 오토인코더 개발 배경, 개념, 설계방법, 응용분야 등에 대해 수강생을 이해를 돕는 것을 목표로 한다. 오토인코더와 관련된 핵심 논문들을 분석하여 관련 이론을 수강생에게 이해시키고 텐서플로우, 케라스 딥 러닝 프레임워크에 기반한 구현실습을 수행하여 수강생들이 현업에서 활용할 수 있도록 한다.

시간별 강의계획

시간	주제	주요내용
1	오토인코더 이론 I	- 오토인코더 이해를 위한 배경 지식(비지도학습, 차원축소, 신경망) 강의 - 기본 오토인코더 이론 강의 (비지도학습, 차원축소, 신경망) 강의
2	오토인코더 이론 II	- 심층 오토인코더 이론 강의 - 디노이징(Denoising) 오토인코더 이론 강의
3	오토인코더 실습	- 심층 오토인코더 실습 강의 - 디노이징 오토인코더 실습 강의

Content

▪ Background-Unsupervised Learning

- Unsupervised learning
- Supervised learning
- Why not exploit unlabeled data?
- Unsupervised learning
- Examples of unsupervised learning
- unsupervised learning

▪ Motivation of Autoencoder

- Dimensionality reduction
- Reconstruction issue
- Principal component analysis
- PCA details
- PCA reconstruction
- Summary of PCA
- Limitation of PCA
- Alternative perspective
- Introduction of Autoencoder
- Alternative model – Autoencoder
- Low-dimensionaity embedding with examples

▪ Basic(simple) Autoencoder Theory

- Basic Autoencoder
- General autoencoder vs. Linear autoencoder

| 이론 I

▪ Deep Autoencoders

- Deep(stacked) Autoencoder
- Layerwise training vs. Whole network Training

▪ Denoising Autoencoder

- Denoising autoencoder – Introduction
- Applying autoencoders to eliminating noise
- Denoising autoencoder (DAE)
- Denoising autoencoder – Notations
- Denoising autoencoder – Manifold perspective
- Perspectives on denoising autoencoders
- Denoising autoencoder – Result
- Denoising autoencoder – Concluding remarks

| 이론 II

▪ Basic and Deep Autoencoder 실습 (Tensorflow)

- Basic autoencoder
- Deep autoencoder
- Denoising autoencoder

| 실습



Background- Unsupervised Learning

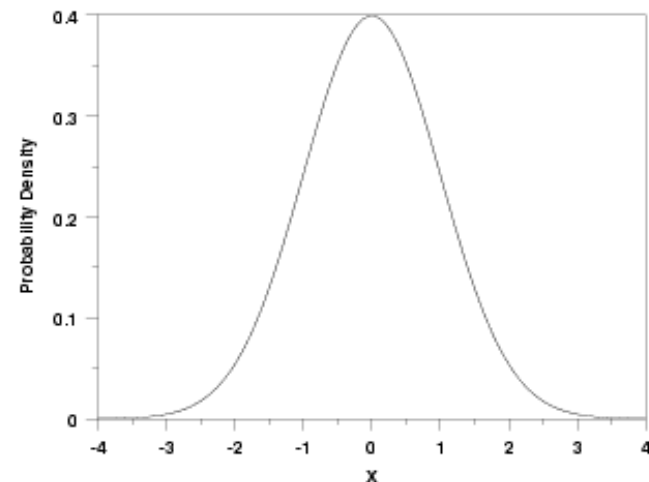
Unsupervised learning

❖ “Learning from unlabeled/unannotated data” (without supervision)



❖ What can we predict from unlabeled data?

- Density estimation
- Groups or clusters in the data
- Low-dimensional structure
 - Principal Component Analysis (PCA)
 - Linear Discriminant Analysis(LDA)
 - Manifold learning(non-linear)



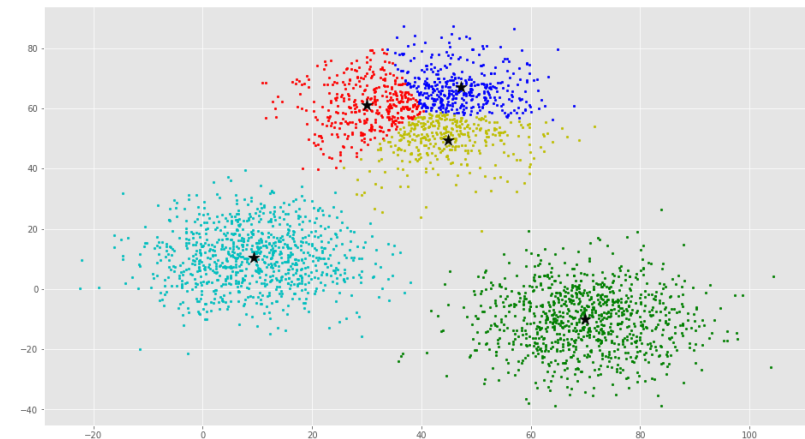
Unsupervised learning

❖ “Learning from unlabeled/unannotated data” (without supervision)



❖ What can we predict from unlabeled data?

- Density estimation
- Groups or clusters in the data
- Low-dimensional structure
 - Principal Component Analysis (PCA)
 - Linear Discriminant Analysis(LDA)
 - Manifold learning(non-linear)



Unsupervised learning

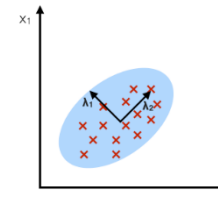
❖ “Learning from unlabeled/unannotated data” (without supervision)



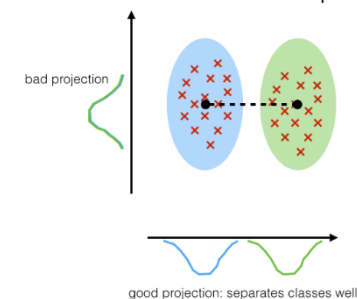
❖ What can we predict from unlabeled data?

- Density estimation
- Groups or clusters in the data
- Low-dimensional structure
 - Principal Component Analysis (PCA)
 - Linear Discriminant Analysis(LDA)
 - Manifold learning(non-linear)

PCA:
component axes that
maximize the variance



LDA:
maximizing the component
axes for class-separation



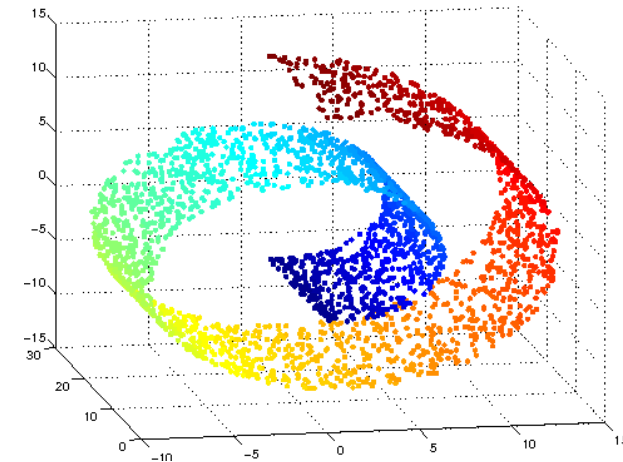
Unsupervised learning

❖ “Learning from unlabeled/unannotated data” (without supervision)



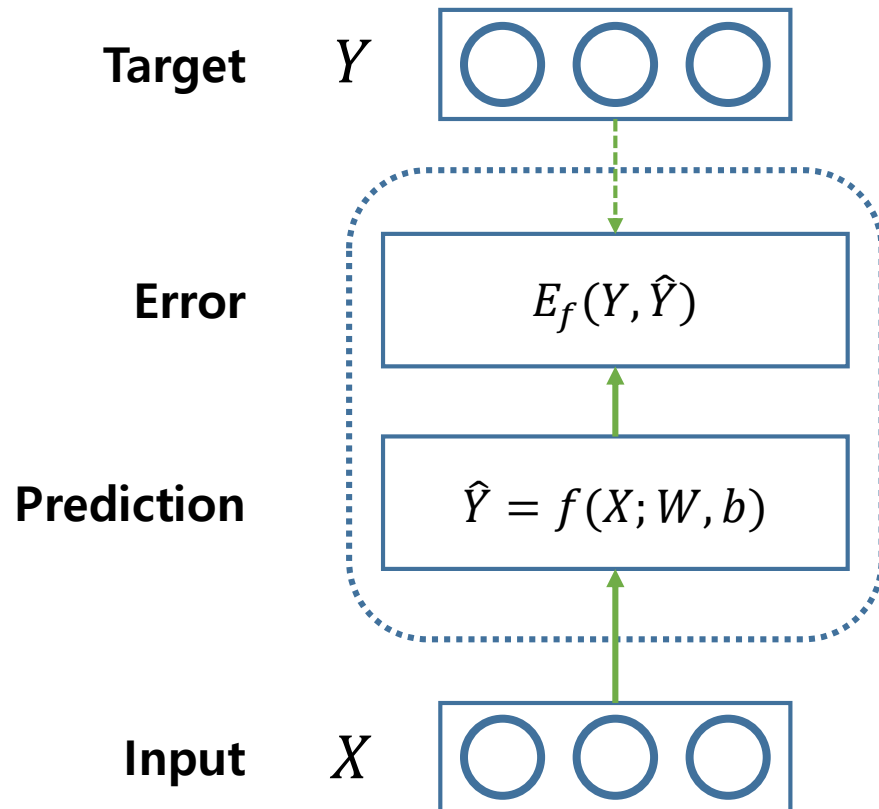
❖ What can we predict from unlabeled data?

- Density estimation
- Groups or clusters in the data
- Low-dimensional structure
 - Principal Component Analysis (PCA)
 - Linear Discriminant Analysis(LDA)
 - Manifold learning(non-linear)

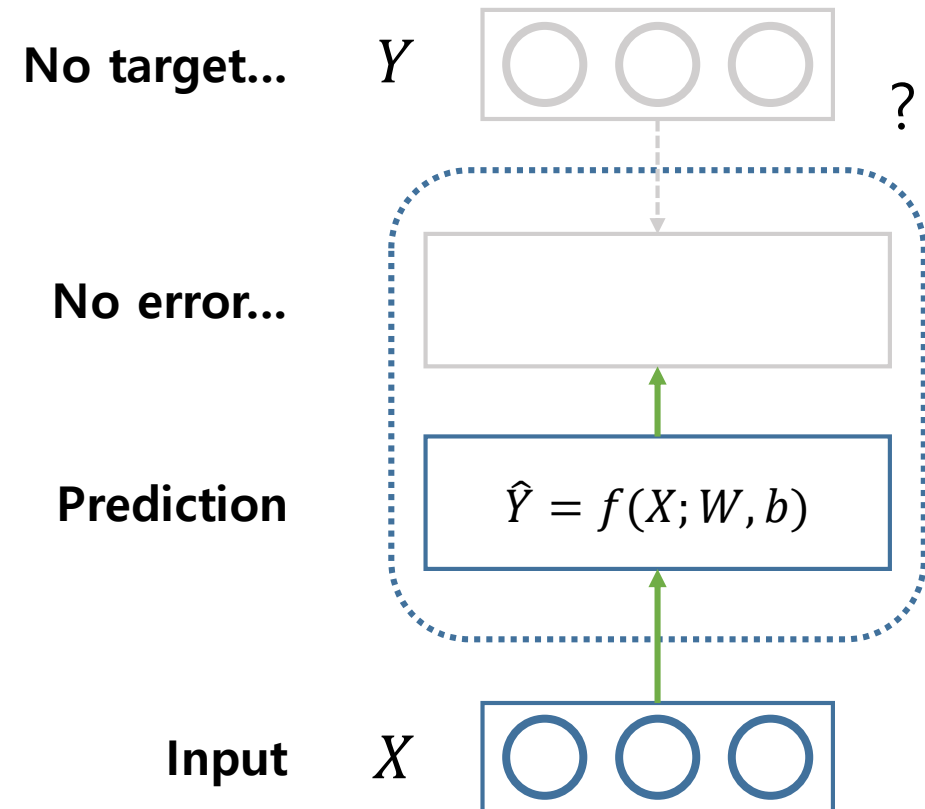


Supervised / Unsupervised learning

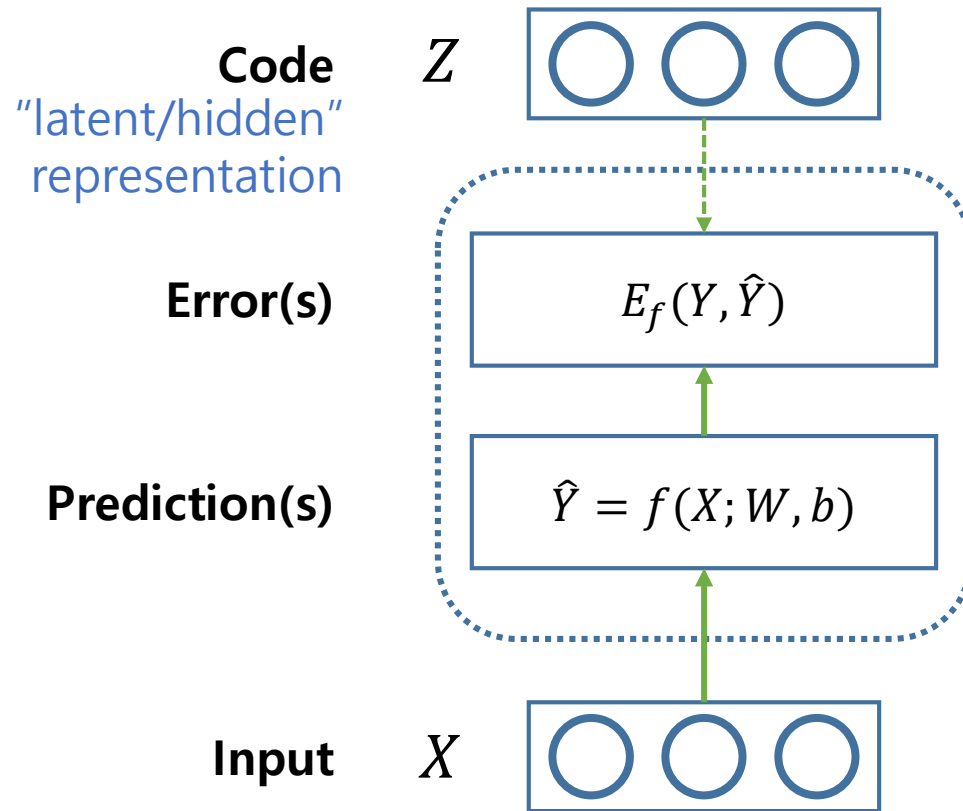
Supervised



Unsupervised



Unsupervised learning



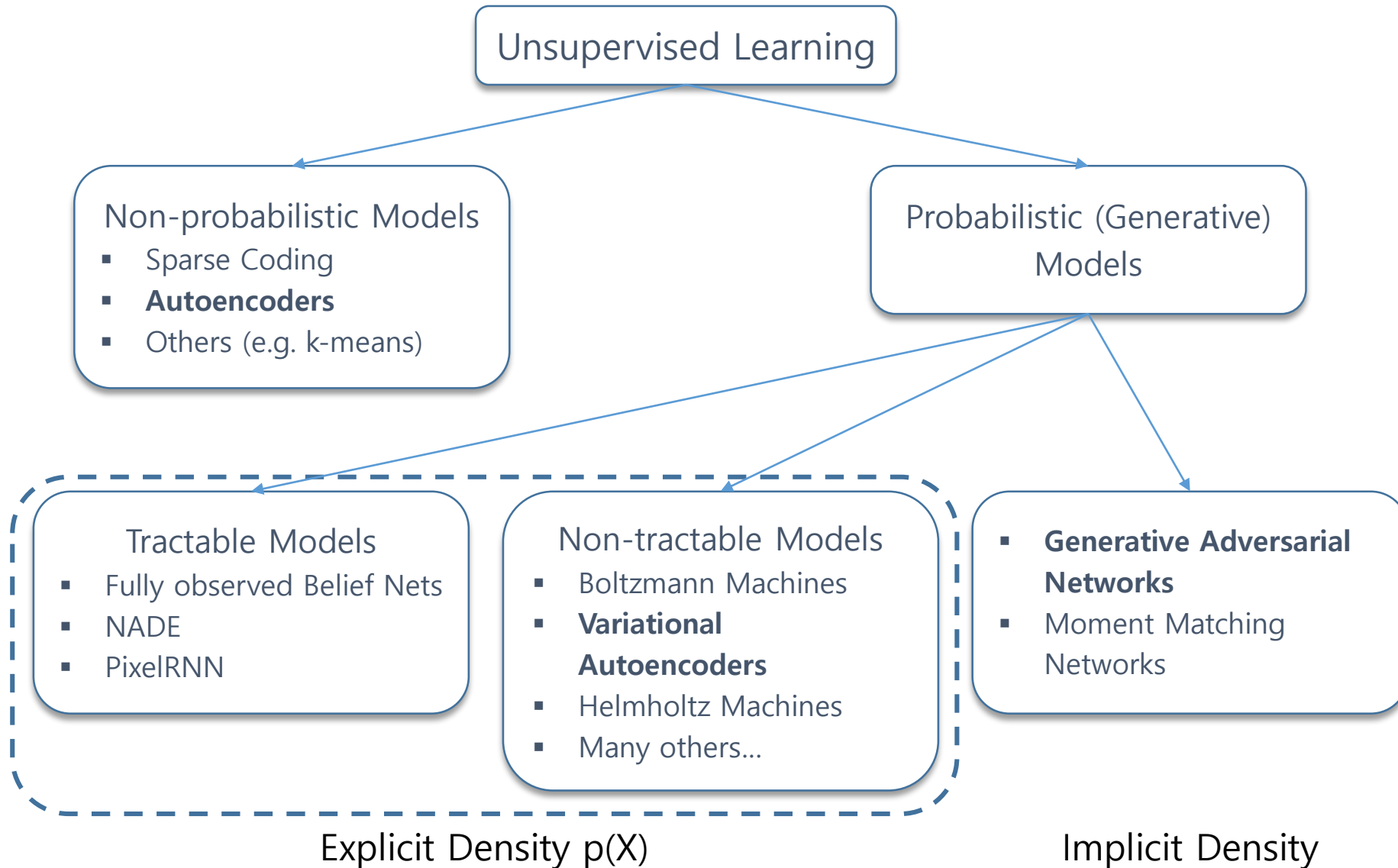
- ❖ We want the **codes** to represent the **inputs** in the dataset.
- ❖ The **code** should be a compact representation of the inputs: low-dimensional and/or sparse.

Unsupervised learning

❖ What's the point if we don't have labels?

- 감독 학습의 **precursor**가 될 수 있다.
 - 분류기의 사전 학습(pretraining)에 비감독 학습이 사용될 수 있다.
 - Restricted Boltzmann Machines, G. Hinton
- 데이터 양은 늘릴 수 있지만, label이 존재할 때 비감독 학습을 활용함.
- 차원 축소를 하면 알고리즘을 설계할 때 좀 더 단순화시킬 수 있고, 다른 훈련 데이터 합성에도 유리.
 - data augmentation
- 사람은 근본적으로 비감독 학습 방식으로 학습함.

Unsupervised learning



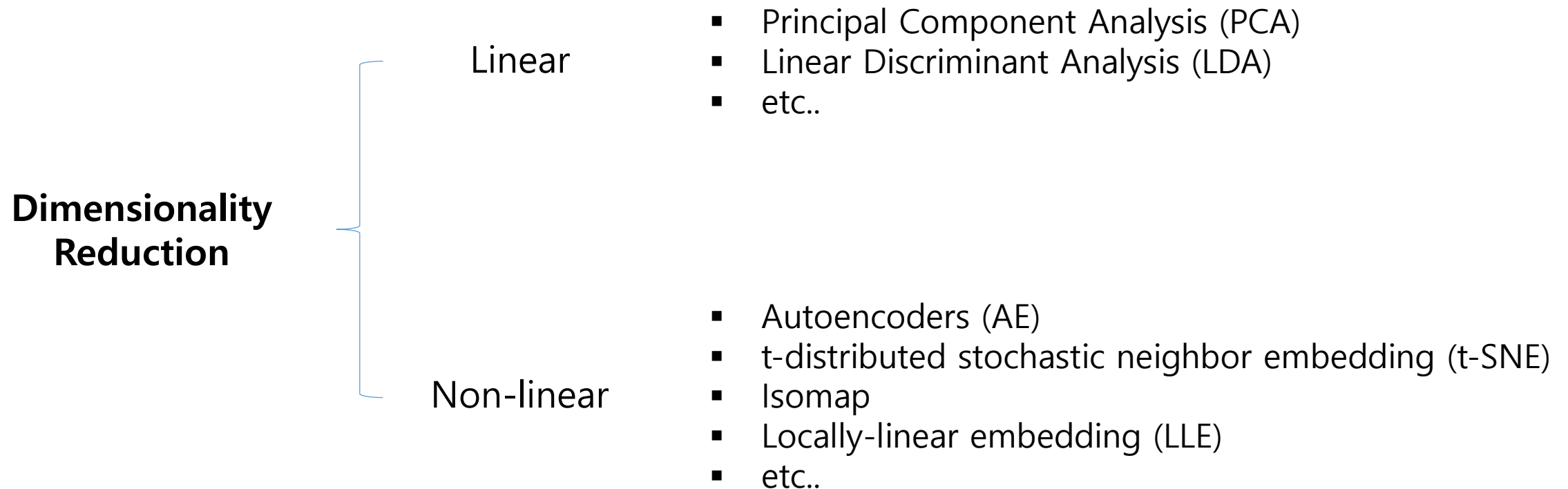


Motivation of Autoencoder

Dimensionality reduction

- 고차원의 특징을 저차원의 벡터로 표현.
 - 적절한 bottleneck vector를 찾을 수 있어야 함.
 - 이러한 알고리즘을 autoencoder에서는 encoder라고 함.
- 전통적인 기계 학습에서 감독 학습의 응용에 많이 활용.
 - 저차원의 데이터를 얻은 다음 분류기(k-NN, SVM, logistic regression...) 모델을 학습 하는 데 활용함.
 - 훈련 데이터 수집 → 특징 추출기 설계 → 차원 축소 → 분류기
(PCA, autoencoder)
- 해석에 유리: 데이터의 특성을 가지면서 시각화 할 수 있음.
 - 데이터의 분포 등을 시각화함으로써 데이터의 특징, 관계 등을 분석하기 쉬움.

Dimensionality reduction



Reconstruction issue

- 축소한 차원을 복원하는 것을 reconstruction이라고 함.
- 다음과 같은 두 transformations을 가정함.
 - The encoder ~ $enc: X \rightarrow Z$
 - The decoder ~ $dec: Z \rightarrow X$(X, Z : input, code)
(these are often simply \mathbb{R}^n and \mathbb{R}^m with $n > m$)
- 복원에 필요한 손실을 최소화 하는 encoder와 decoder의 parameters를 찾아야 함.

$$L(\vec{x}) = ||dec(enc(\vec{x})) - \vec{x}||^2$$

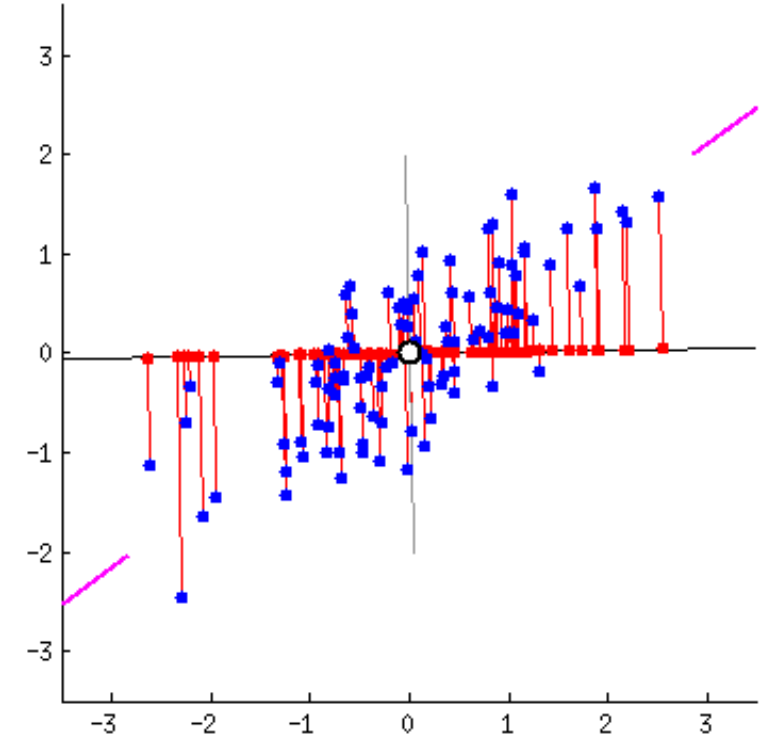
PCA (Principal component analysis)

- 차원 축소에서 가장 기본적인 모델.
 - autoencoder에서는 encoder가 차원 축소 역할을 함.
 - PCA는 비선형적인 데이터에 대해서는 차원 축소가 잘 안 됨.
- Encode by projecting the n -dimensional data onto a set of m orthogonal axes. ($n \geq m$)
 - n 차원의 데이터를 orthogonal한 공간에 projection을 시켜서 저차원을 만듦
- orthogonal 공간에 투영을 할 때 데이터의 손실이 거의 없게 차원 축소를 하고, 차원 축소한 데이터는 원래 데이터의 특성을 그대로 보존해야 함.
- 데이터를 최대한 보존시키기 위해서는 variance가 큰 m 개의 축에 투영시킴
= (principle component값이 큼)

PCA(principal Component Analysis) :: 주성분 분석

For 차원 축소
(Dimension Reduction)

- 데이터를 요약하는 방법
- 기존 정보를 최대한 살려 새로운 **특성** 설정
 - show as much variation across the data as possible
: **max** variance
 - reconstruct the original characteristics as well as possible
: **min** error



PCA details

- 직교 축에 투영시키는 것을 선형 operation이라고 할 때, PCA encoder는 간단히 행렬의 곱으로 나타낼 수 있음.

$$enc(\vec{X}) = W\vec{X}$$

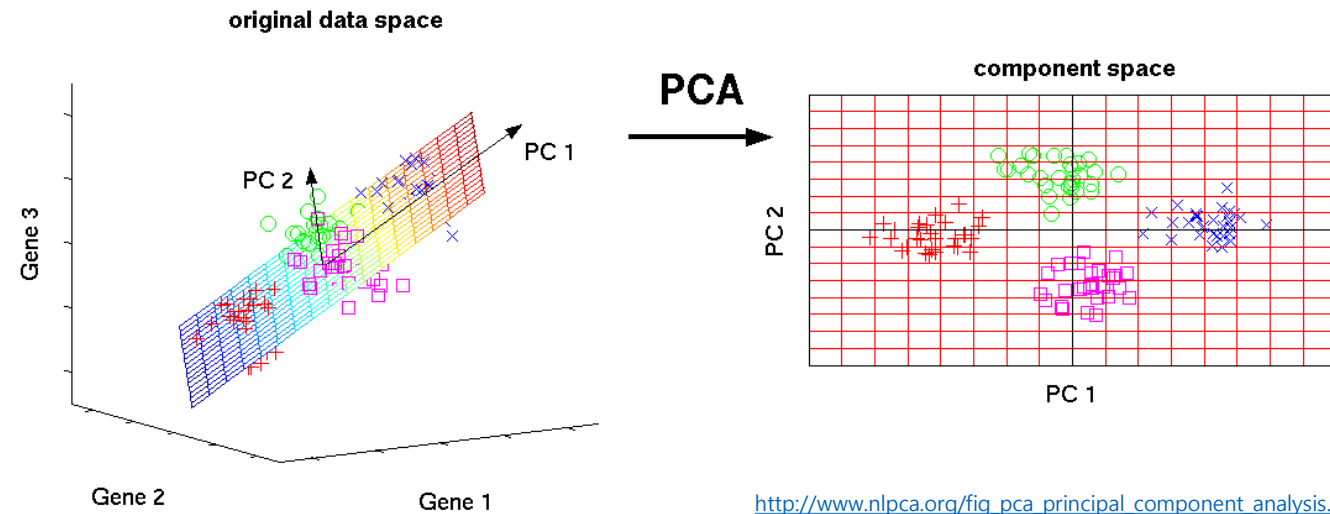
(where **W** is of size $m \times n$)

- encoder의 결과를 벡터 Z라고 하면, PCA에서 구한 W를 transpose 시켜서 Z에 곱함으로써 입력 X를 복원 시킴.

$$dec(\vec{Z}) = W^T\vec{Z}$$

- 즉, $\|\vec{X} - W^TW\vec{X}\|^2$ 를 최소화하는 W를 찾아야 함.
 - eigenvalue decomposition을 통해서 고유값을 찾는 문제로 풀면 됨.

Summary of PCA



http://www.nlpca.org/fig_pca_principal_component_analysis.png

❖ Finds k directions in which data has highest variance

- Principal directions (eigenvectors) W

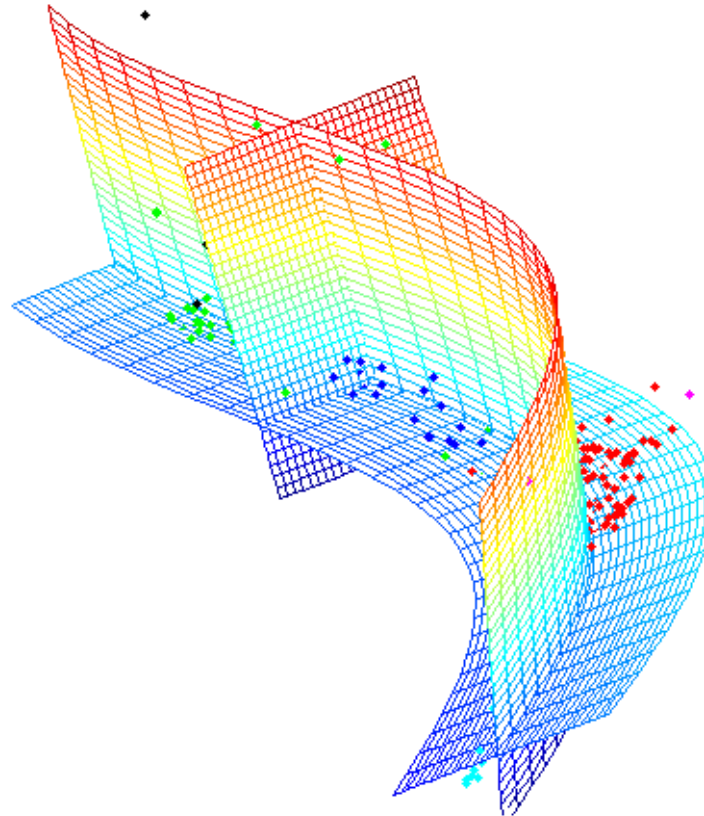
❖ Projecting inputs x on these vectors yields reduced dimension representation (&decorrelated)

- Principal components
- $h = f_{\theta}(x) = W(x - \mu)$ with $\theta = \{W, \mu\}$

❖ Why mention PCA?

- Prototypical unsupervised representation learning algorithm
- Related to autoencoders
- Prototypical manifold modeling algorithm

Limitation of PCA



Linear model \Rightarrow Incapable of capturing nonlinear manifolds!

Alternative perspective

- ❖ PCA처럼 matrix projection으로 모델을 구성하는 것이 아니라 두 개의 layer를 갖는 완전 연결 신경망을 생성.

- ❖ PCA의 동작 방식인데, backpropagation을 통해 학습 시킴.

$$\vec{Z} = enc(\vec{X}) = W_1\vec{X} + \vec{b}_1$$

$$\vec{X} = dec(\vec{Z}) = W_2\vec{Z} + \vec{b}_2$$

- ❖ 복원에 필요한 손실을 최적화 시킴.

$$L(\vec{X}') = ||\vec{X}' - \vec{X}||^2$$

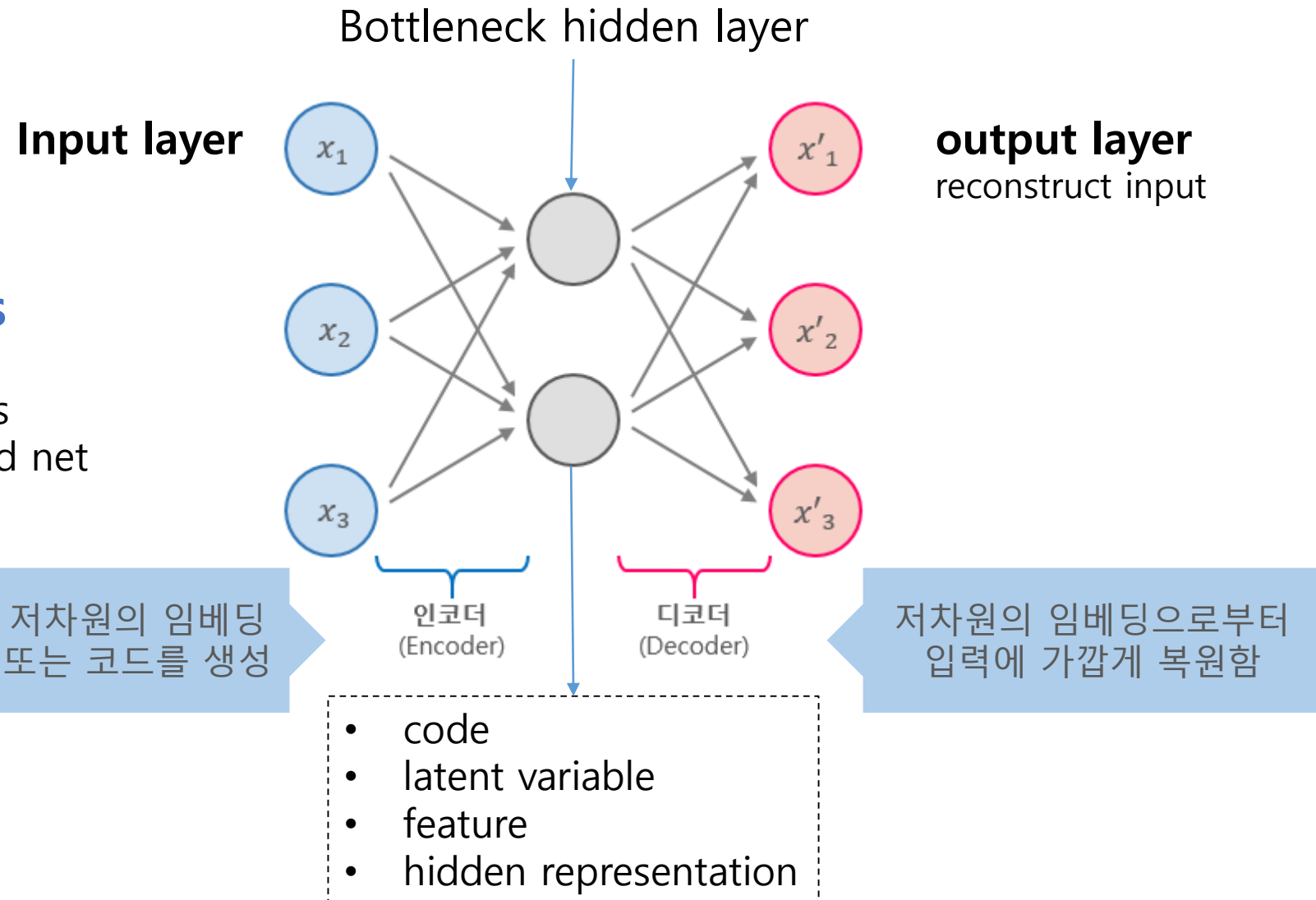
- ❖ 기본 신경망과 다른 오토 인코더의 특징:

- encoder – decoder
- 입력을 복원 시킨다는 관점에서의 loss를 구함.

Introduction of Autoencoder

Autoencoders

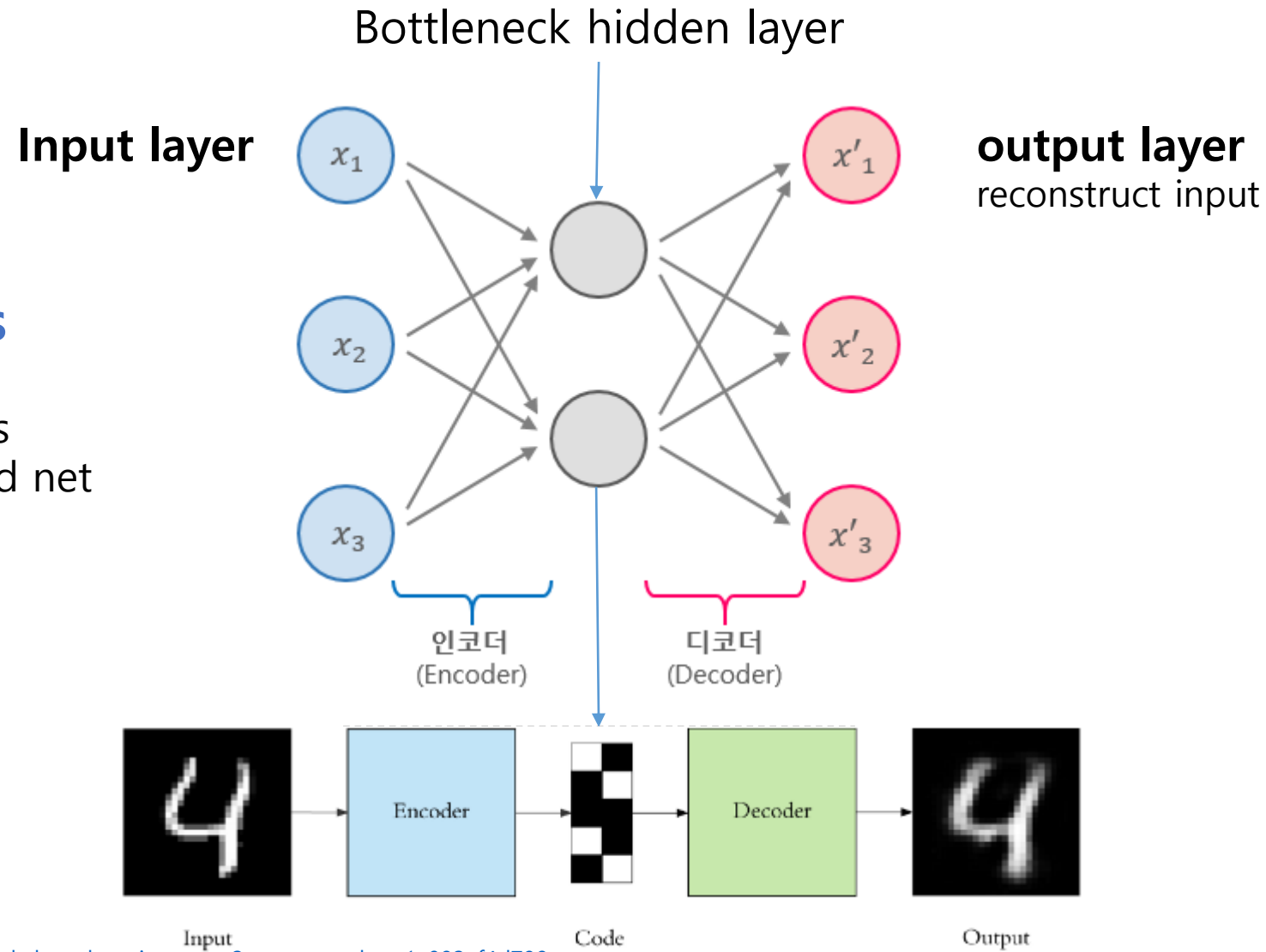
= auto-associators
= diablo networks
= sandglass-shaped net

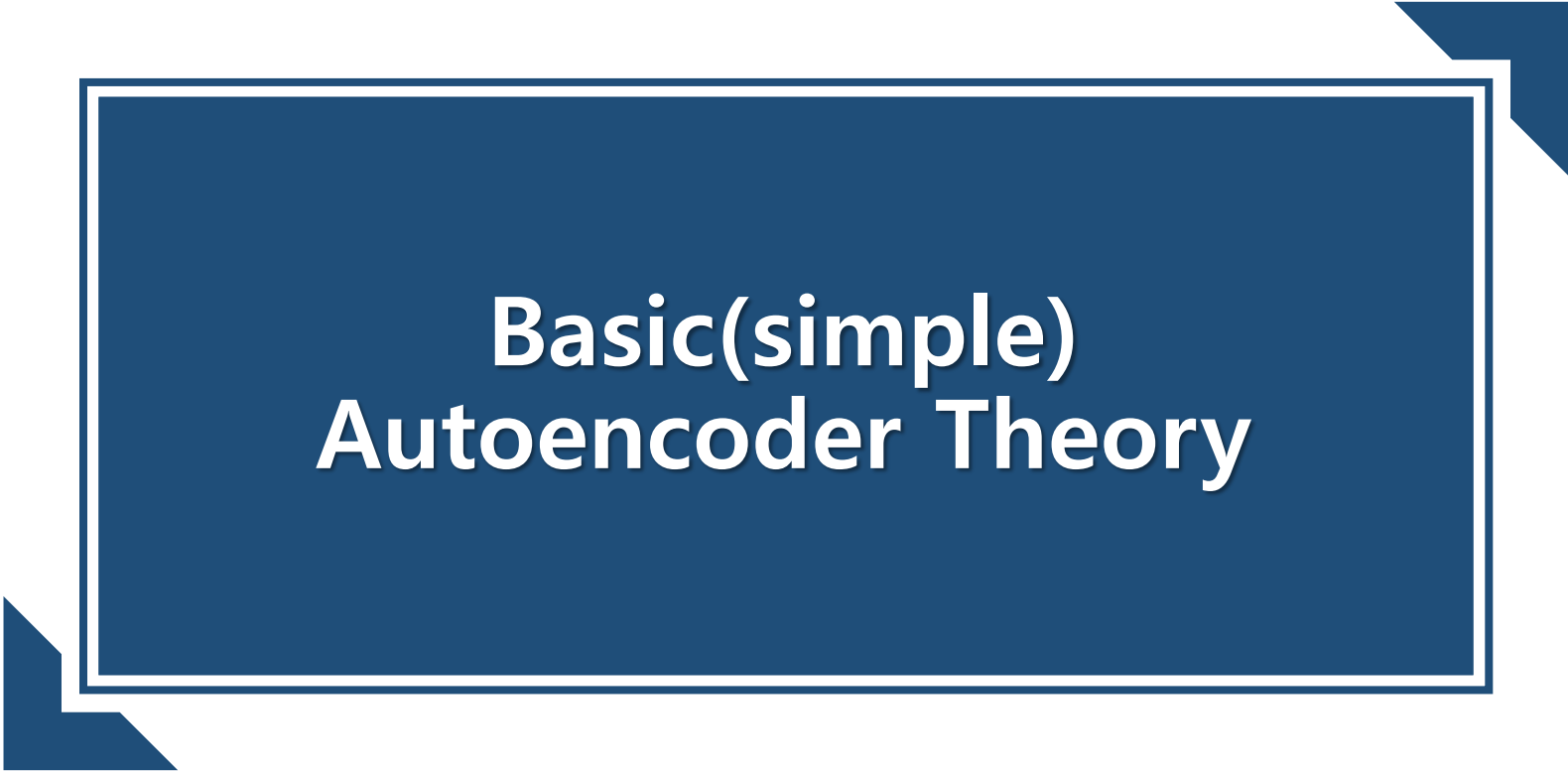


Introduction of Autoencoder

Autoencoders

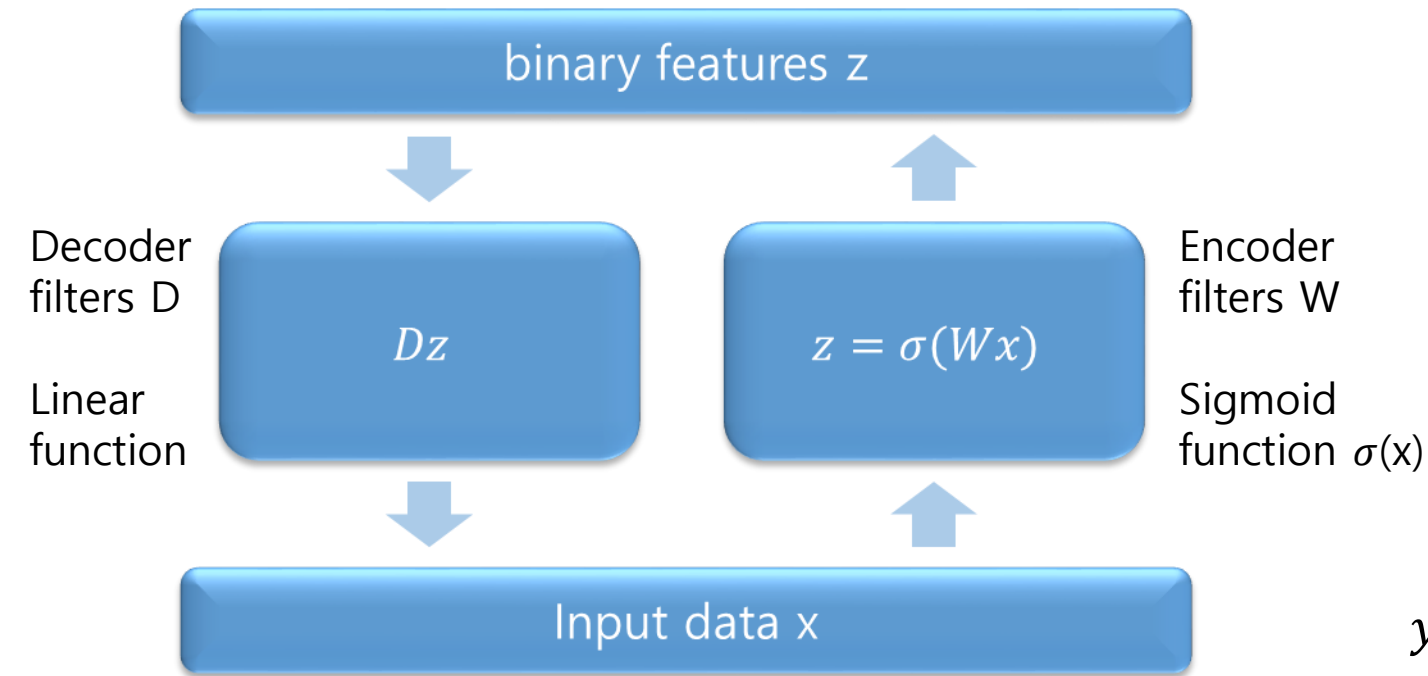
- = auto-associators
- = diablo networks
- = sandglass-shaped net





Basic(simple) Autoencoder Theory

Basic Autoencoder

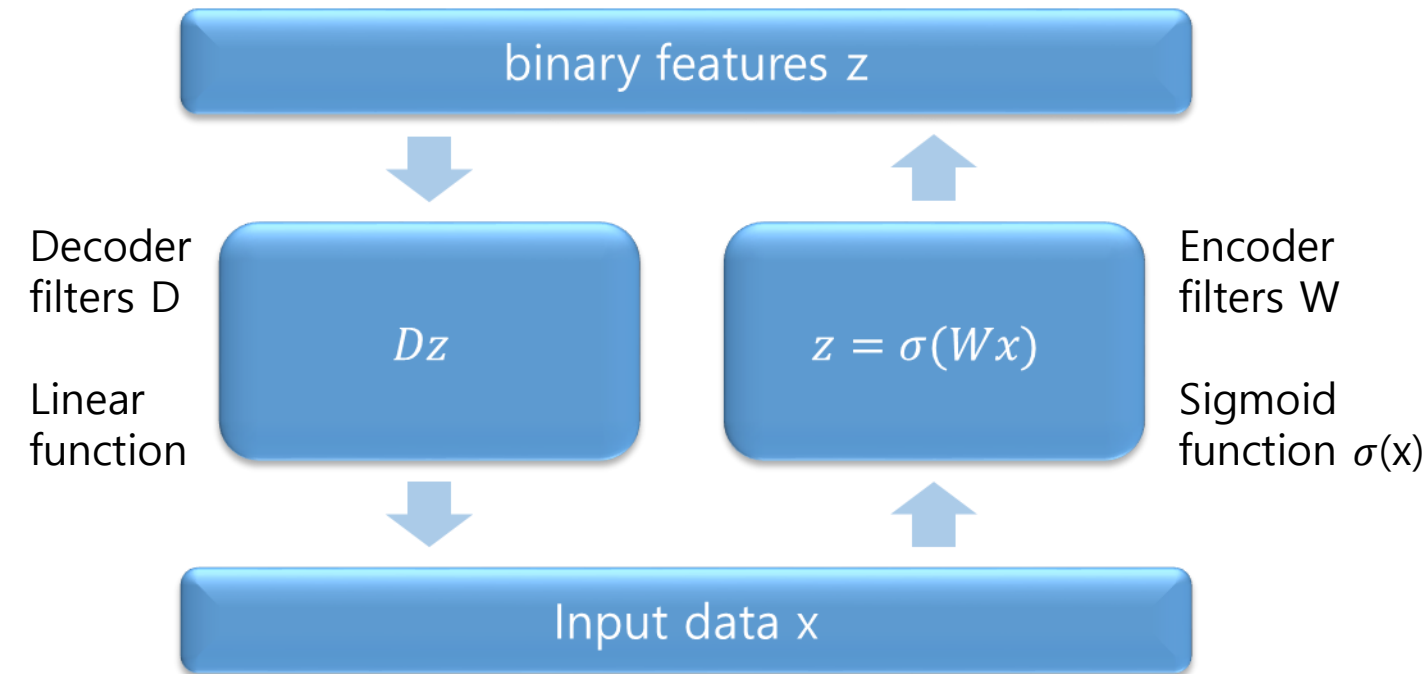


- 오토 인코더가 D 개의 입력, D 개의 출력, K 의 히든 유닛으로 이루어져 있을 때, $K < D$.
- 오토 인코더의 최종 목적은 D 개의 입력을 복원 시키는 것이다. (입력 D = 출력 D)
- 입력을 x 라 할 때, 복원 수식은 다음과 같다.

$$y_j(x, W, D) = \underbrace{\sum_{k=1}^K D_{jk}}_{\text{Decoder}} \underbrace{\sigma \left(\sum_{i=1}^D W_{ki} x_i \right)}_{\text{Encoder}}, i = 1, \dots, D$$

$$y_j = \sum_{k=1}^K D_{jk} z_k \quad z_k = \sigma \left(\sum_{i=1}^D W_{ki} x_i \right)$$

Basic Autoencoder

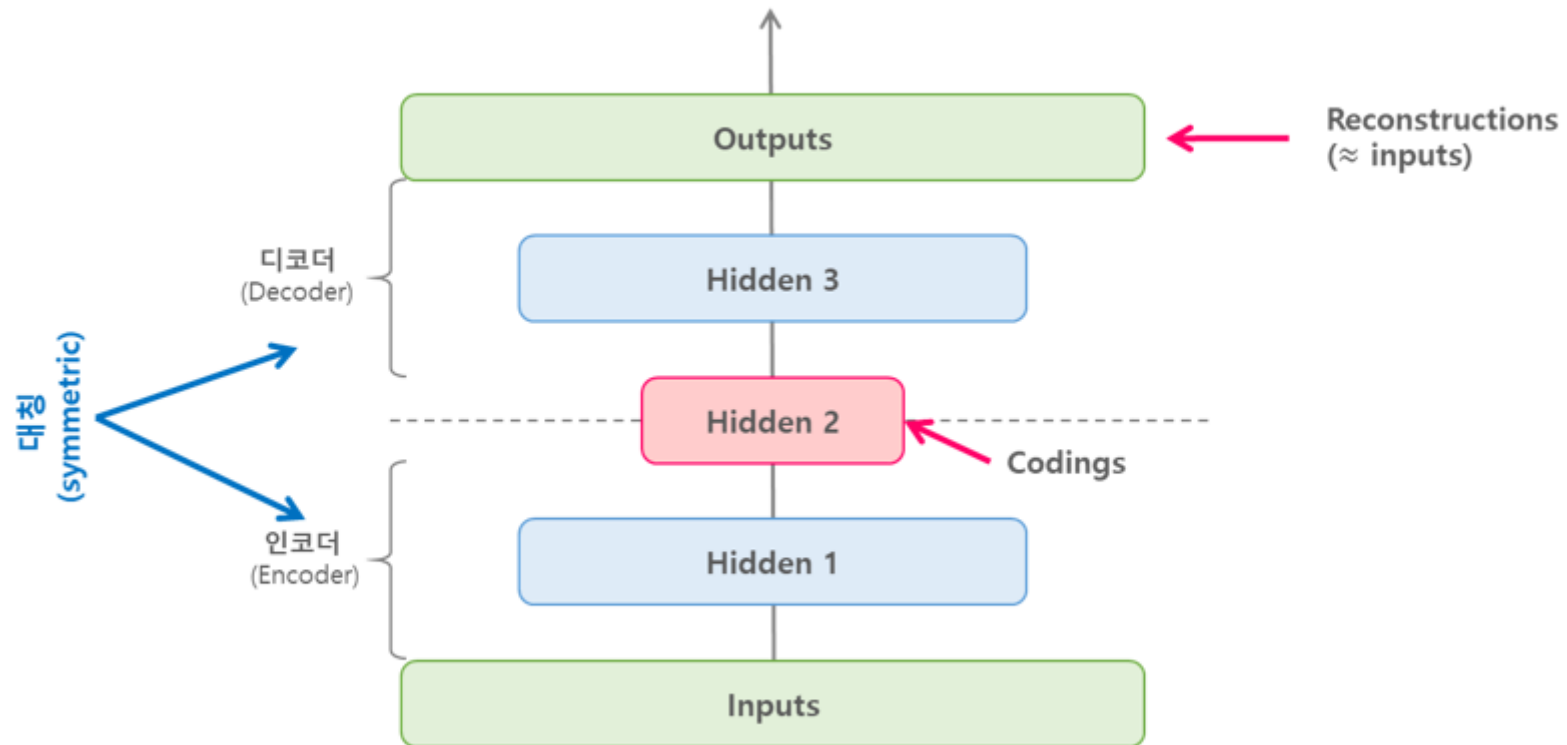


- reconstruction error를 최소화함으로써 네트워크 파라미터 W와 D를 구할 수 있다.

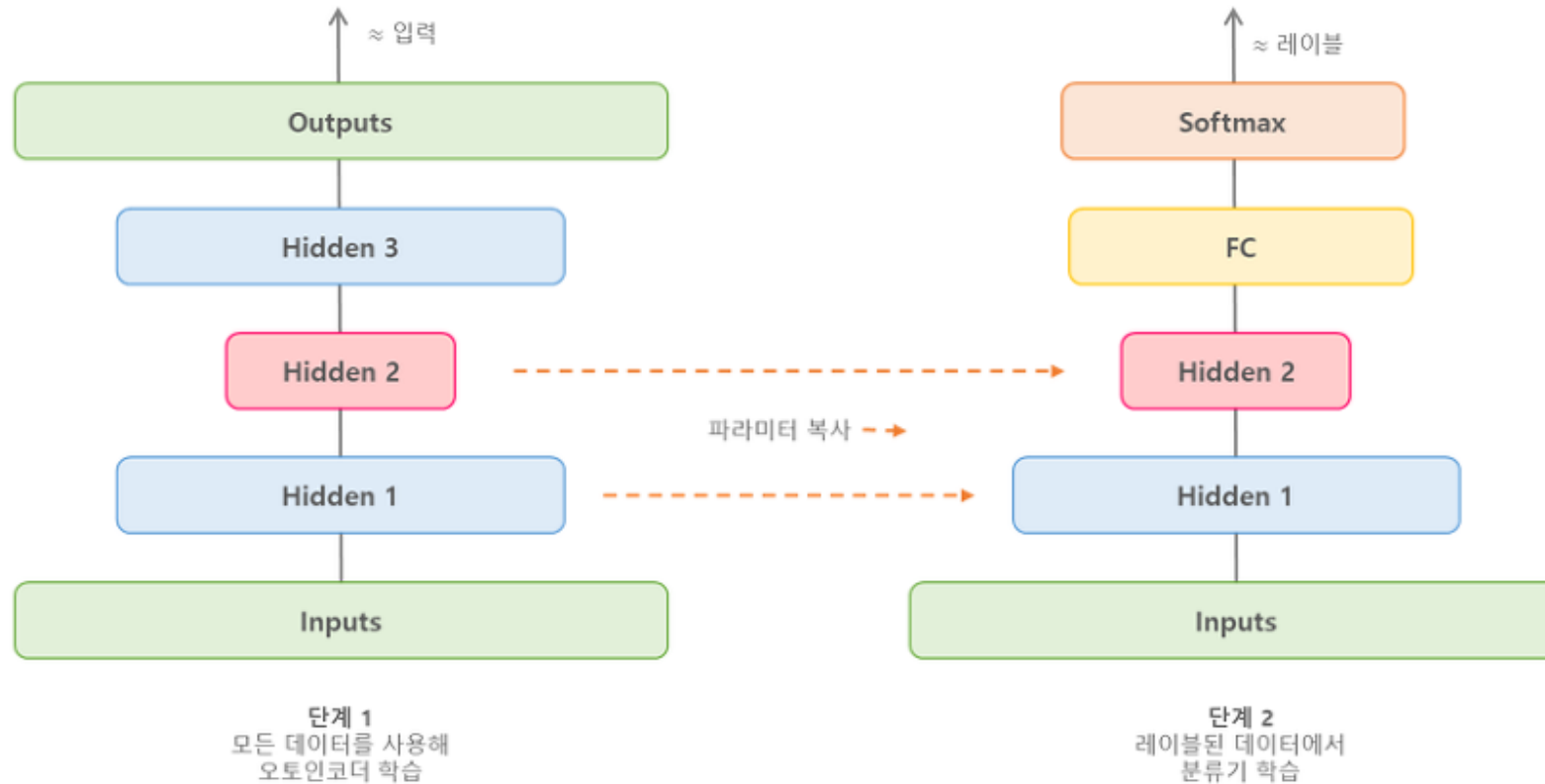
$$E(W, D) = \frac{1}{2} \sum_{n=1}^N \|y(x_n, W, D) - x_n\|^2$$

Deep Autoencoders

Deep (stacked) autoencoder



Deep (stacked) autoencoder를 이용한 비지도 사전학습



❖ 비감독 학습

- label y 없이 계속해서 새로운 입력을 잘 복원할 수 있도록만 학습함.

❖ 사전학습에 사용

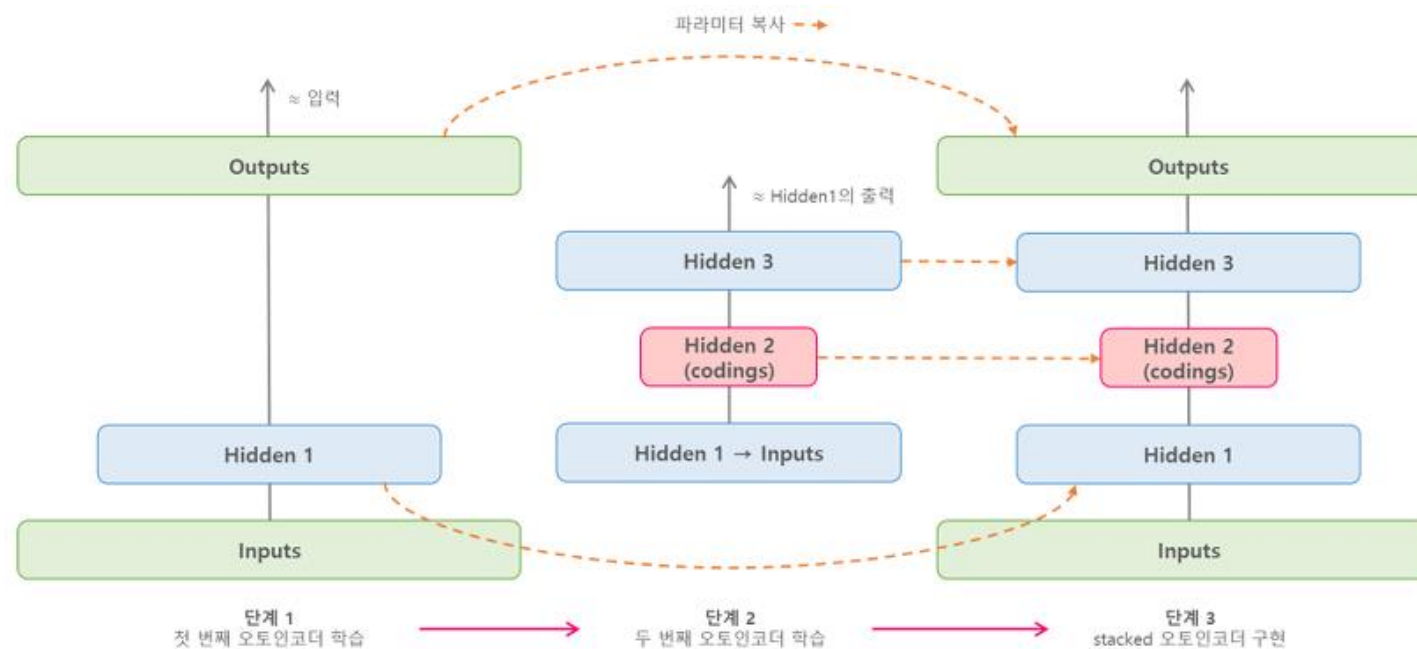
- 설계한 오토 인코더를 분류 문제에 사용할 수 있음.
- 주어진 deep autoencoder의 상층부에 softmax를 얹고 학습 → data가 혼재할 때는 이 방법으로 성능 향상 추구

Deep (stacked) autoencoder를 이용한 비지도 사전학습

- ❖ 심층 오토인코더가 좋은 인코더로 기능하도록 설계하면, 압축 과정에 따른 **정보 손실이 적어** 입력 데이터에 약간의 변화가 있어도 결과에 악영향을 미치지 않을 수 있다.
- ❖ 그러므로 **심층 오토인코더의 가중치는 분류나 인식 문제 같은 감독 학습 문제의 좋은 초기 가중치**가 될 수 있다. 이는 심층 오토인코더를 효과적으로 활용할 수 있는 방법 중 하나이다.

Layerwise training vs. Whole Network Training

- ❖ layerwise training을 할 때의 오토인코더는 감독학습을 위한 사전 단계로 구성되고 이용되었다.

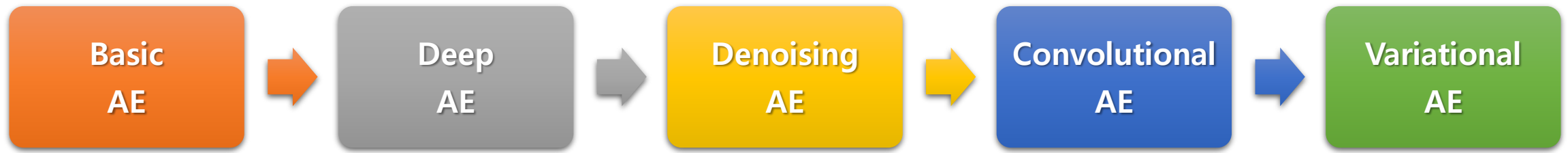


- ❖ 현재는 레이블이 있는 large scale data가 많음
→ 분류기까지 올려서 한번에 학습 가능
- ❖ whole network training이 deep neural network를 훈련하는 가장 흔한 방식이 됨.



Denoising Autoencoder

Denoising autoencoder - Introduction



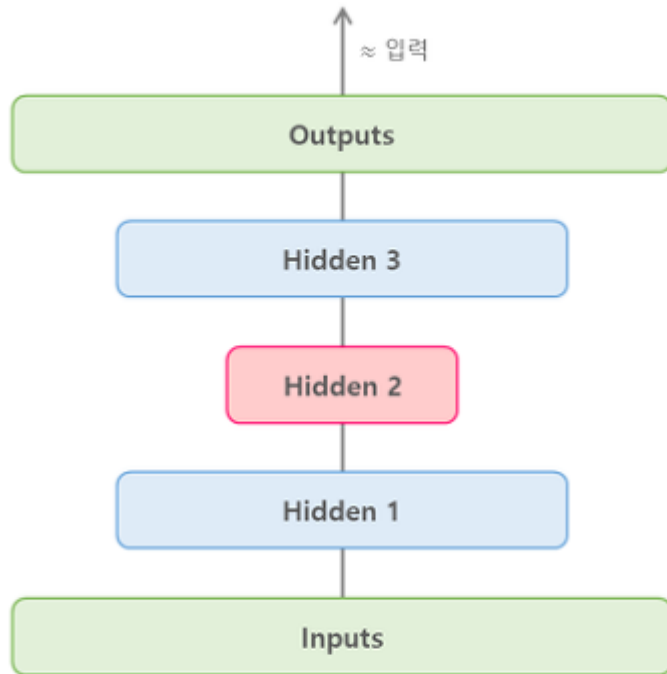
- 입력 패턴에 **corruption**을 줘서 원래 데이터를 변형시킴
- 복원 시킬 때는 corruption이 없는 **원본 데이터로 복원**
- 입력에 corruption이 있어도 인식이나 분류 문제에 견고한 성능을 보임
- **manifold learning**에 motivation을 받아서 설계됨
- **실세계의 데이터**는 대부분 잡음이 있으므로 깨끗한 데이터를 넣어봤자 의미가 없음.

Applying autoencoders to eliminating noise

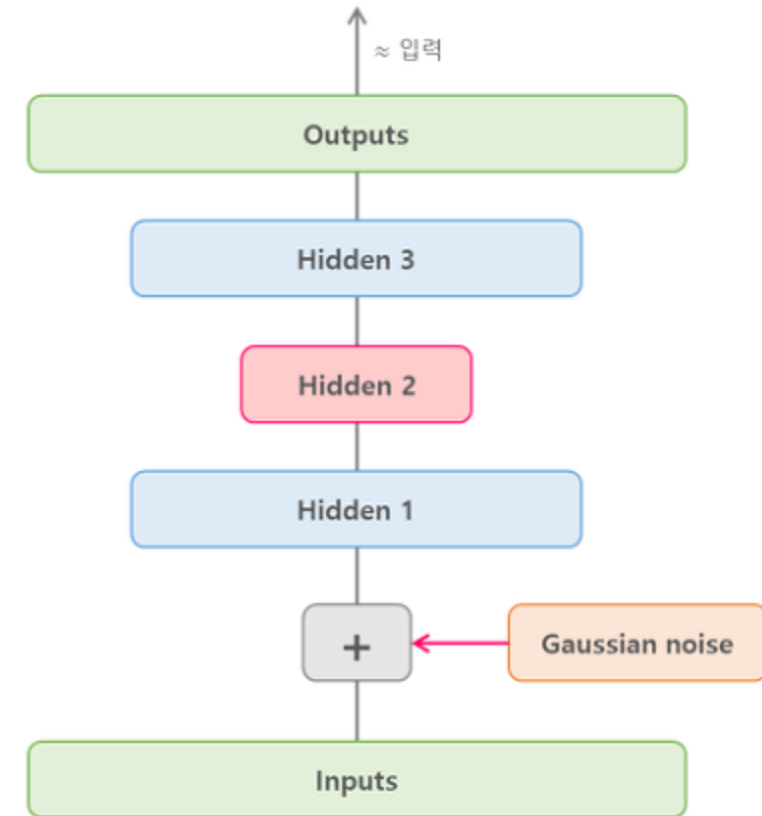
❖ This comes from several motivations:

- **corruption이 있어도 잘 복원이 될 수 있는 모델을 만들어 보자.**
 - 실세계에서 얻은 data는 대부분 노이즈로부터 자유롭지 못함.
- **훈련 데이터에 오버피팅되는 문제를 해결해 보자.**
 - 변형된 데이터로 훈련을 해서 모델이 훈련에 사용되지 않은 데이터에 대해서도 좀 더 강인한 성능을 갖게 하고자 함.
- **강인한 representation을 갖게 하자.**
 - 노이즈가 있거나 변형이 되도 encoder에서 나오는 hidden representation이 noise에 좀 더 강인하도록 설계

Denoising autoencoder (DAE)



< stacked autoencoder >



< denoising autoencoder >

Denoising autoencoder - Notations

- ❖ **input vector:** $x \in [0, 1]^d$, **hidden representation:** $y \in [0, 1]^{d'}$ 라고 할 때,
 $y = f_{\theta}(x) = s(W_x + b)$, $\theta = \{W, b\}$ (W : $d \times d'$ weight matrix, b : bias vector)
- ❖ **reconstructed vector:** $z \in [0, 1]^d$,
 $z = g_{\theta'}(y) = s(W'y + b')$, $\theta' = \{W', b'\}$
 - decoder: latent representation y 를 통해 z 로 복원
- ❖ **reconstruction loss를 최소화하기 위해 모델 파라미터를 최적화 시킴:**

$$\begin{aligned}\theta^* \theta'^* &= \operatorname{argmin}_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^n L(x^{(i)}, z^{(i)}) \\ &= \operatorname{argmin}_{\theta, \theta'} \frac{1}{n} \sum_{i=1}^n L(x^{(i)}, g_{\theta'}(f_{\theta}(x^{(i)})))\end{aligned}$$

loss function: $L(x, z) = ||x - z||^2$

Denoising autoencoder

- ❖ an alternative reconstruction loss: **cross-entropy**

$$L(x, z) = H(\beta_x || \beta_z)$$

$$= - \sum_{k=1}^d [x_k \log z_k + (1 - x_k) \log(1 - z_k)]$$

Denoising autoencoder

- ❖ 왜곡된 입력에 대해서 강건한 모델을 만들기 위해 기본 오토인코더를 수정
- ❖ stochastic mapping $\tilde{x} \sim q_D(\tilde{x}|x)$ 을 통해 초기 입력 x 를 왜곡한다.
- ❖ 주어진 입력을 얼마나 파괴시킬지에 대한 파라미터 v 를 이용하여,
주어진 입력 x 가 있을 때 vd 개의 컴포넌트만 랜덤하게 선택해 0으로 치환

Denoising autoencoder

- ❖ 파괴된 입력 \tilde{x} 를 기본 오토인코더에 넣어 hidden representation y 를 얻는다.

$$y = f_{\theta}(\tilde{x}) = s(W\tilde{x} + b)$$

- ❖ 파괴된 입력에 대한 복원값 z 를 얻는다.

$$z = g_{\theta'}(y) = s(W'y + b')$$

- ❖ average reconstruction error가 최소화 될 수 있도록 훈련한다.

$$L_{IH}(x, z) = H(\beta_x || \beta_z)$$

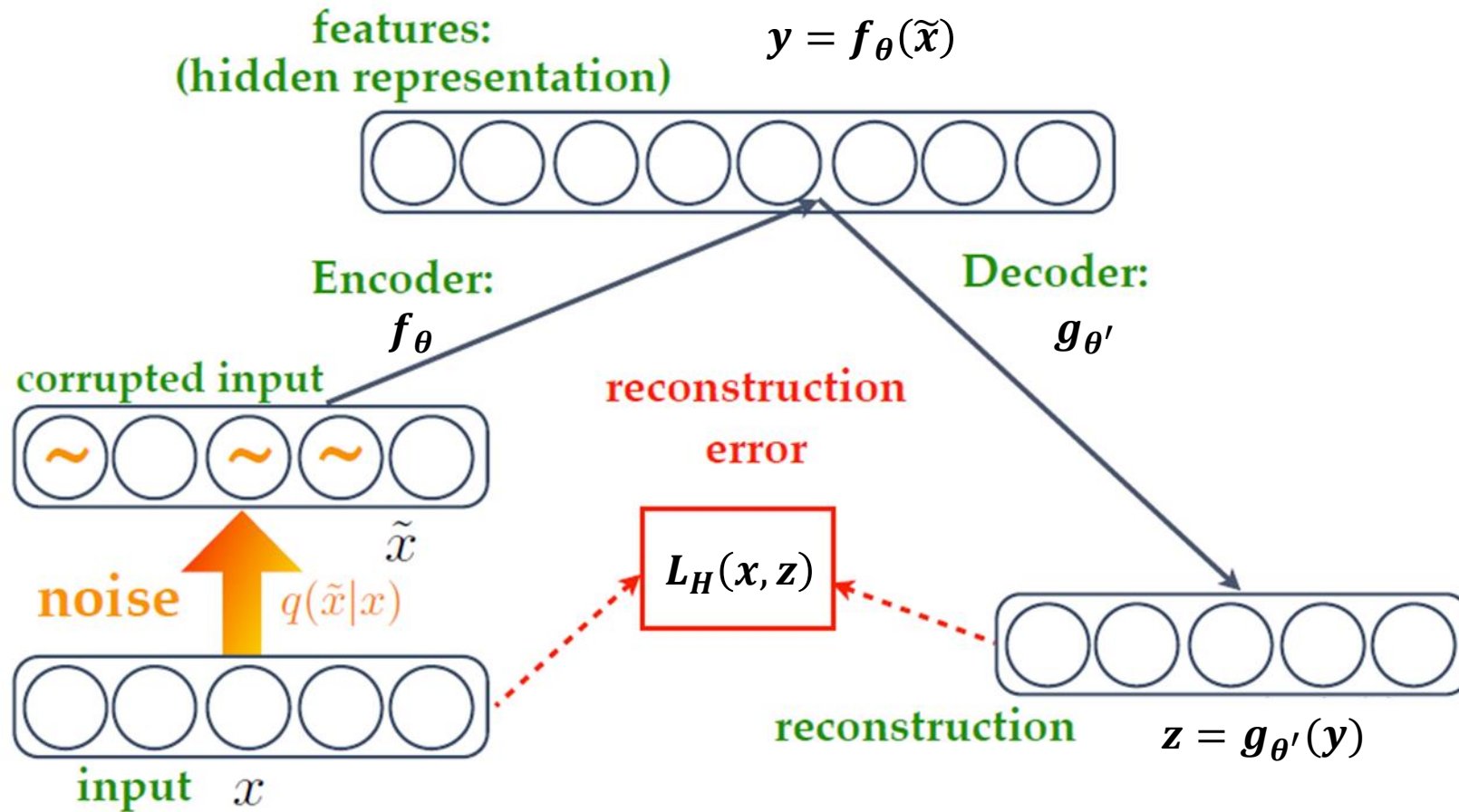
- ❖ 기존과의 큰 차이점은 z 가 \tilde{x} 의 deterministic function이라는 점이다.

Denoising autoencoder

- ❖ Stochastic gradient descent를 통해 목적 함수를 최소화:

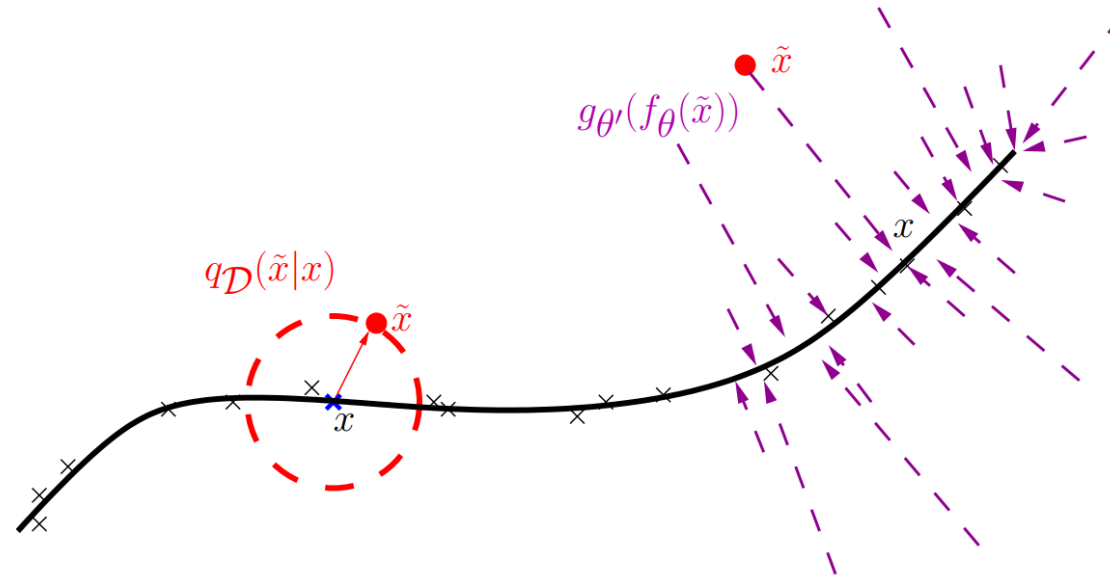
$$\operatorname{argmin}_{\theta, \theta'} \mathbb{E}_{q^0(X, \tilde{X})} [L_{IH}(X, g_{\theta'}(f_{\theta}(\tilde{X})))]$$

Denoising autoencoder



Perspectives on denoising autoencoders

Manifold learning perspective



❖ DAE는 **manifold**를 학습하는 방식의 하나로 볼 수 있다. :

- 훈련 데이터 (X) 가 저차원 매니폴드 공간에 있다고 할 때,
- $q_d(\tilde{X}|X)$ 에 의해 **corruption**이 일어난 입력(\cdot) 은 매니폴드 공간에서 벗어나게 된다.
- 따라서 왜곡된 데이터를 다시 매니폴드 공간으로 **projection**시키도록 모델을 학습($p(\tilde{X}|X)$) 시킨다.

Denoising autoencoder - Result

Dataset	SVM _{rbf}	SVM _{poly}	DBN-1	SAA-3	DBN-3	SdA-3 (ν)
<i>basic</i>	3.03±0.15	3.69±0.17	3.94±0.17	3.46±0.16	3.11±0.15	2.80±0.14 (10%)
<i>rot</i>	11.11±0.28	15.42±0.32	14.69±0.31	10.30±0.27	10.30±0.27	10.29±0.27 (10%)
<i>bg-rand</i>	14.58±0.31	16.62±0.33	9.80±0.26	11.28±0.28	6.73±0.22	10.38±0.27 (40%)
<i>bg-img</i>	22.61±0.37	24.01±0.37	16.15±0.32	23.00±0.37	16.31±0.32	16.68±0.33 (25%)
<i>rot-bg-img</i>	55.18±0.44	56.41±0.43	52.21±0.44	51.93±0.44	47.39±0.44	44.49±0.44 (25%)
<i>rect</i>	2.15±0.13	2.15±0.13	4.71±0.19	2.41±0.13	2.60±0.14	1.99±0.12 (10%)
<i>rect-img</i>	24.04±0.37	24.05±0.37	23.69±0.37	24.05±0.37	22.50±0.37	21.59±0.36 (25%)
<i>convex</i>	19.13±0.34	19.82±0.35	19.92±0.35	18.41±0.34	18.63±0.34	19.06±0.34 (10%)

- ❖ $\nu = 0\%$ 라면, SAA-3(basic autoencoder)와 SdA-3(denoising autoencoder)는 같다.
- ❖ 분류 및 인식과 같은 문제에서 denoising autoencoder가 더 좋다.

Denoising autoencoder – Concluding remarks

- ❖ DAE 는 데이터에 왜곡이 있을 때 비감독 학습을 통해서 이를 극복하고자 설계되었다.
- ❖ 감독 학습을 할 때 intermediate representation으로도 좋은 역할을 한다.
 - softmax 레이어를 추가해서 감독 학습기를 설계할 때에도 DAE의 hidden representation이 유용하게 사용될 수 있다.
- ❖ 입력에 왜곡이 있더라도 훈련 및 검증을 할 때 강건한 성능을 낼 수 있다.

References

- ❖ P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, **Extracting and Composing Robust Features with Denoising Autoencoders**, *Proceedings of the 25th International Conference on Machine Learning (ICML'2008)*, pp. 1096-1103, Omnipress, 2008.
 - paper
<https://www.cs.toronto.edu/~larocheh/publications/icml-2008-denoising-autoencoders.pdf>
 - presentation (4th CiFAR Summer School on Learning and Vision in Biology and Engineering, 2008)
http://ailab.jbnu.ac.kr/seminar_board/pds1_files/pascal_vincent_part_2.pdf

Autoencoder 구현실습

Development Environment

- ❖ `conda==4.5.4`
- ❖ `python==3.6.5`
- ❖ `tensorflow==1.11.0`
- ❖ `sklearn, argparse, numpy, matplotlib, pillow`



Basic and Deep Autoencoder 실습(Tensorflow)

Content

❖ Basic autoencoder

- 오토인코더의 기본 구조
- 필요한 모듈 호출 및 데이터셋 로드
- 오토 인코더 모델 하이퍼파라미터 설정
- 오토인코더 모델 구성 (인코더/디코더)
- 모델 학습시키기 (손실 함수와 최적화 함수 설정)
- 학습 실행 결과
- 모델 테스트 해보기(matplotlib을 이용한 이미지 출력)

❖ Deep autoencoder

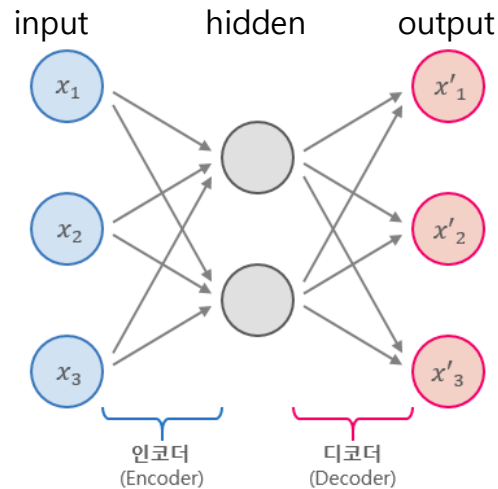
- 필요 라이브러리 설치 및 코드 다운로드
- 오토인코더 구조
- 모델 구조와 하이퍼파라미터 설정
- layer함수
- 인코더/디코더
- 손실 함수 정의 및 신경망 학습 알고리즘
- 모델 학습시키기
- 프로그램 실행 및 테스트 - 결과 및 손실값 변화

❖ Denoising autoencoder

- corrupt_input 함수
- 손상된 이미지 생성
- loss function
- 프로그램 실행하기
- 프로그램 실행하여 테스트하기
- 디노이징 오토인코더 결과 예시

Basic autoencoder

❖ 오토인코더의 기본 구조



- 입력값과 출력값을 같게 하는 신경망이며, 가운데 은닉 계층의 노드 수가 입력값보다 적다.
(데이터 압축, 노이즈 제거에 효과적)
- 입력을 히든 레이어로 인코딩하고 이를 디코딩하여 만들어진 출력 값을 원래의 입력값과 비슷해지도록 만드는 가중치를 찾아내는 것이 핵심이다.
(입력값이 압축되므로 입력에서 출력으로의 손실 없는 완벽한 복사는 일어날 수 없다.)
- 변이형 오토인코더, 잡음 제거 오토인코더 등 다양한 방식이 있다.

Basic autoencoder

❖ 필요한 모듈 호출 및 데이터셋 로드 (1)

- numpy – 행렬 조작과 연산에 필수라 할 수 있는 수치 해석용 파이썬 라이브러리
- matplotlib – 시각화를 위해 그래프를 쉽게 그릴 수 있도록 해주는 파이썬 라이브러리
- 밑에 두 줄처럼 텐서플로에 내장된 MNIST 모듈을 통해 MNIST 데이터셋을 다운받고 사용할 수 있다.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```





```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("./mnist/data/", one_hot=True)
```

- MNIST 데이터셋 – 손으로 쓴 숫자들의 이미지를 모아 놓은 데이터셋으로, 0부터 9까지의 숫자를 28X28픽셀 크기의 이미지로 구성해 놓음. 전 처리까지 잘 되어있어 머신러닝계의 Hello, World!라고도 볼 수 있다.
(총 70,000장 – train data: 60,000장, test data: 10,000장)
- one hot encoding – 데이터가 가질 수 있는 값들을 일렬로 나열한 배열을 만들고, 표현하려는 값을 뜻하는 인덱스의 원소만 1로 표기하고 나머지 원소는 모두 0으로 채우는 label 표기법

Basic autoencoder

❖ 필요한 모듈 호출 및 데이터셋 로드 (2)

- 프로젝트 폴더에 mnist 폴더가 생긴 것을 확인할 수 있다.

📁 > 내 PC > 로컬 디스크 (C:) > TF_Projects > mnist > data			
이름	수정한 날짜	유형	크기
 t10k-images-idx3-ubyte	2019-07-17 오후...	압축(GZ) 파일	1,611KB
 t10k-labels-idx1-ubyte	2019-07-17 오후...	압축(GZ) 파일	5KB
 train-images-idx3-ubyte	2019-07-17 오후...	압축(GZ) 파일	9,681KB
 train-labels-idx1-ubyte	2019-07-17 오후...	압축(GZ) 파일	29KB

- mnist 숫자 이미지



Basic autoencoder

❖ 오토인코더 모델 하이퍼 파라미터 설정

```
learning_rate = 0.01
training_epoch = 20
batch_size = 100
# 신경망 레이어 구성 옵션
n_hidden = 256 # 히든 레이어의 뉴런 갯수
n_input = 28*28 # 입력값 크기 - 이미지 픽셀 수
```

- learning_rate – 최적화 함수에서 사용할 학습률
- training_epoch – 전체 데이터를 학습할 총 횟수
- batch_size – 미니배치로 한 번에 학습할 데이터(이미지)의 개수

Basic autoencoder

❖ 오토인코더 모델 구성

```
#####  
# 신경망 모델 구성  
#####  
X = tf.placeholder(tf.float32, [None, n_input])
```

- 입력 X의 플레이스홀더를 설정해준다.
- 오토인코더 모델은 **비지도 학습**이므로 y(ground truth)값이 없다.
- X코드에서 텐서의 첫 번째 차원에 None이 있는 자리는 **한번에 학습시킬 mnist 이미지의 개수**를 지정하는 값이 들어간다. 즉, 배치 크기를 지정하는 자리.
- None은 크기가 정해지지 않았음을 의미한다. 한번에 학습할 개수를 계속 바꿔가면서 실험해보려는 경우에 None으로 넣어주면 텐서플로가 알아서 계산한다.

Basic autoencoder

❖ 오토인코더 모델 구성 - 인코더 만들기

```
W_encode = tf.Variable(tf.random_normal([n_input, n_hidden]))
b_encode = tf.Variable(tf.random_normal([n_hidden]))
# sigmoid 함수를 이용해 신경망 레이어를 구성
# sigmoid(X * W + b)
# 인코더 레이어 구성
encoder = tf.nn.sigmoid(
    tf.add(tf.matmul(X, W_encode), b_encode))
```

인코더 모델: $\sigma(X * W_{encode} + b_{encode})$

- 인코더 레이어의 가중치(weight)와 편향(bias) 변수를 원하는 뉴런의 개수만큼 설정하고 정규분포를 띄는 무작위 수로 초기화 한다.
- n_hidden개(256개)의 뉴런을 가진 은닉층을 만든다. output의 크기(은닉 층의 뉴런 개수)를 입력값(뉴런 784개)보다 적은 크기로 만들어 정보를 압축하여 특성을 뽑아낸다.
- 활성화 함수로 sigmoid 사용

Basic autoencoder

❖ 오토인코더 모델 구성 - 디코더 만들기

```
W_decode = tf.Variable(tf.random_normal([n_hidden, n_input]))
b_decode = tf.Variable(tf.random_normal([n_input]))
# 디코더 레이어 구성
# 이 디코더가 최종 모델
decoder = tf.nn.sigmoid(
    tf.add(tf.matmul(encoder, W_decode), b_decode))
```

디코더(최종 모델): $\sigma(\text{encoder} * W_{\text{decode}} + b_{\text{decode}})$

- 디코더도 인코더와 같은 구성이지만 **입력값을 은닉층의 크기로, 출력값을 입력층의 크기로** 만들어 입력과 똑같은 output을 만들어 내도록 한다.
- 활성화 함수로 sigmoid를 사용하여 최종 모델을 구성한다(decoder가 최종 출력 값이 됨).

Basic autoencoder

❖ 모델 학습시키기 - 손실 함수와 최적화 함수 설정

```
cost = tf.reduce_mean(tf.pow(X - decoder, 2))  
optimizer = tf.train.RMSPropOptimizer(learning_rate).minimize(cost)
```

- 입력 값인 X 를 평가를 위한 실측값(ground truth)으로 사용하고, 디코더가 내보낸 결과 값 $decode$ 와의 차이의 제곱을 **손실값**으로 설정한다.
- **비용(cost)**은 `tf.reduce_mean` 함수를 사용하여 모든 데이터에 대한 손실 값의 평균을 내어 구한다.
(MSE, Mean Square Error)
- 텐서플로가 기본적으로 제공하는 `tf.train.RMSPropOptimizer` 최적화 함수를 사용하여 가중치와 편향값을 변경해가면서 **손실값을 최소화하는 최적화된 가중치와 편향값**을 찾아주도록 한다.
- https://www.tensorflow.org/api_docs/python/tf/train

Basic autoencoder

❖ 모델 학습시키기

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

total_batch = int(mnist.train.num_examples / batch_size)

for epoch in range(training_epoch):
    total_cost = 0
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        _, cost_val = sess.run([optimizer, cost],
                                feed_dict={X: batch_xs})
        total_cost += cost_val
    print('Epoch: ', '%04d' % (epoch + 1),
          'Avg. cost = {:.4f}'.format(total_cost / total_batch))

print('최적화 완료!')
```

- 정의한 변수들을 초기화해주고, 학습 데이터의 총 개수인 `mnist.train.num_examples`를 앞서 설정했던 `batch_size`(미니배치 크기 = 100)로 나누어 **미니 배치가 총 몇 개인지**를 구한다. (`mnist.train`을 사용하면 학습 데이터를, `mnist.test`를 사용하면 테스트 데이터를 사용할 수 있다.)
- 학습 데이터 전체를 학습하는 일을 총 `training_epoch`만큼 반복한다.
- `mnist.train.next_batch(batch_size)`함수를 이용해 학습할 데이터를 배치 크기만큼 가져와서 `feed_dict`매개변수에 입력값이자 평가를 위한 실측값 `X`에 사용할 데이터를 넣어준다.
- `sess.run`을 이용하여 최적화시키고 손실값을 저장한 다음, 한 세대의 학습이 끝나면 학습한 세대의 평균 손실값을 출력한다.

Basic autoencoder

❖ 학습 실행 결과

- epoch이 늘어남에 따라 손실값이 성공적으로 점점 잘 줄어드는 것을 확인할 수 있다.

```
Epoch: 0001 Avg. cost = 0.2015
Epoch: 0002 Avg. cost = 0.0653
Epoch: 0003 Avg. cost = 0.0533
Epoch: 0004 Avg. cost = 0.0480
Epoch: 0005 Avg. cost = 0.0433
Epoch: 0006 Avg. cost = 0.0414
Epoch: 0007 Avg. cost = 0.0399
Epoch: 0008 Avg. cost = 0.0387
Epoch: 0009 Avg. cost = 0.0378
Epoch: 0010 Avg. cost = 0.0372
Epoch: 0011 Avg. cost = 0.0368
Epoch: 0012 Avg. cost = 0.0363
Epoch: 0013 Avg. cost = 0.0349
Epoch: 0014 Avg. cost = 0.0343
Epoch: 0015 Avg. cost = 0.0335
Epoch: 0016 Avg. cost = 0.0333
Epoch: 0017 Avg. cost = 0.0331
Epoch: 0018 Avg. cost = 0.0330
Epoch: 0019 Avg. cost = 0.0329
Epoch: 0020 Avg. cost = 0.0327
최적화 완료!
```

Basic autoencoder

❖ 모델 테스트 해보기

```
sample_size = 10

samples = sess.run(decoder,
                    feed_dict={X: mnist.test.images[:sample_size]})
```

- 먼저 앞에서부터 총 10개의 테스트 데이터를 가져와 feed_dict로 입력값 X에 넣어주고 디코더를 이용해 출력값으로 만든다.

Basic autoencoder

❖ 모델 테스트 해보기 – matplotlib을 이용한 이미지 출력

```
fig, ax = plt.subplots(2, sample_size, figsize=(sample_size, 2))

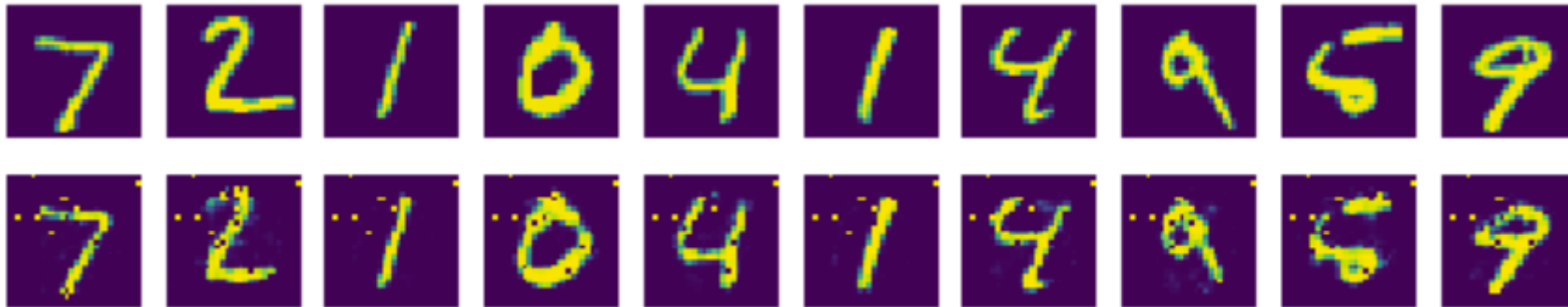
for i in range(sample_size):
    ax[0][i].set_axis_off()
    ax[1][i].set_axis_off()
    ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
    ax[1][i].imshow(np.reshape(samples[i], (28, 28)))

plt.show()
```

- numpy모듈을 이용해 mnist 데이터를 28x28크기의 이미지 데이터로 재구성한 뒤, matplotlib의 imshow 함수를 이용해 그래프에 이미지로 출력한다.
- 위쪽에는 입력 값의 이미지를, 아래쪽에는 오토인코더 신경망으로 생성한 이미지를 출력한다.

Basic autoencoder

❖ 모델 테스트 해보기 - 출력 결과



- 입력을 압축하여 약간의 노이즈가 끼 있지만 원본과 거의 유사하게 이미지를 생성해 낸 것을 확인할 수 있다.

Deep autoencoder

❖ 필요 라이브러리 설치 및 코드 다운로드

- 다음 명령어를 입력하여 실습에 필요한 라이브러리들을 설치한다.

관리자: 명령 프롬프트

```
(tensorflow_dae) C:\WTF_Projects>pip install sklearn argparse
```

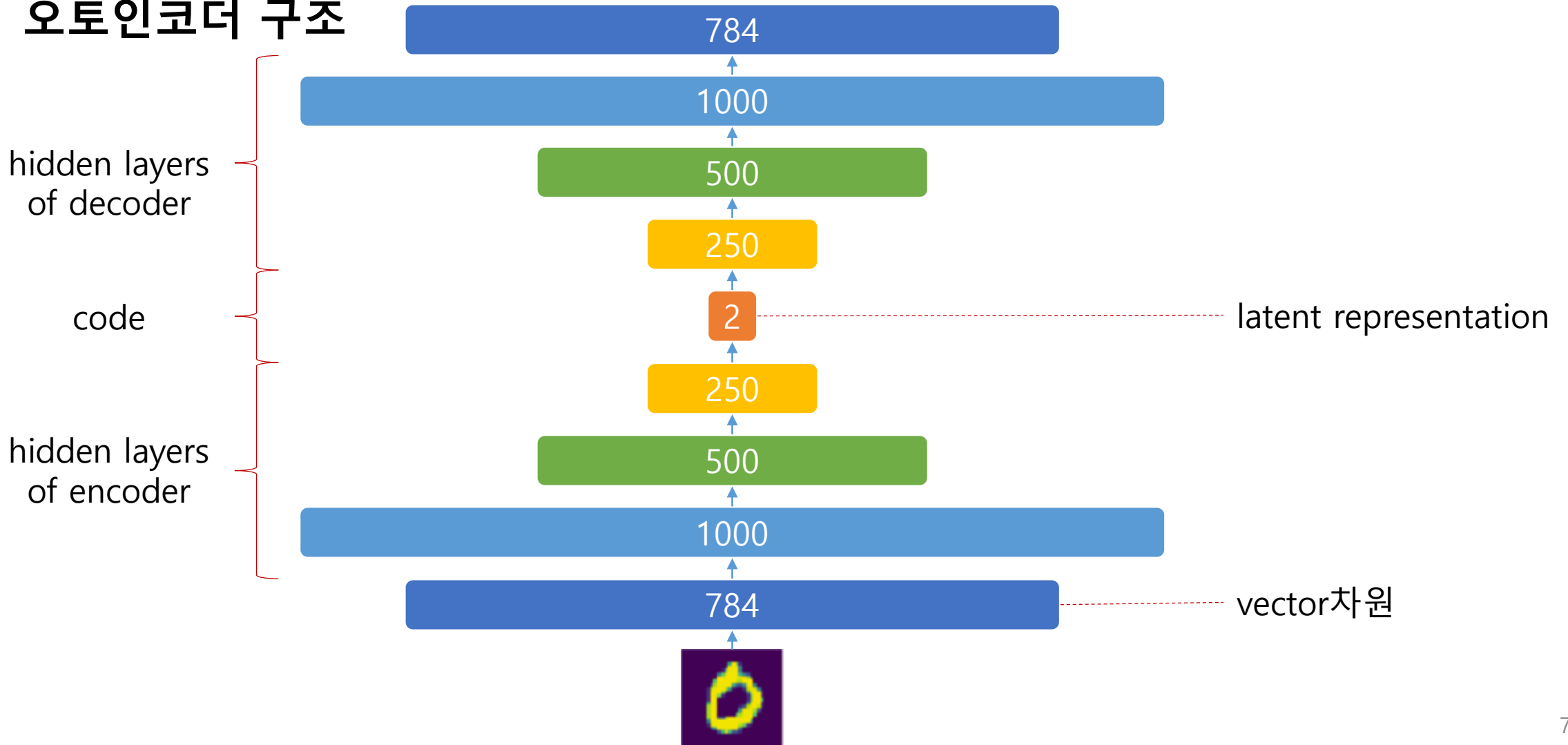
- 구글 드라이브에서 C에 다운받고 풀어주기
http://drive.google.com/open?id=1bRI8_Y-LjAG6EjFIN03GFmaeVBb2mDR

- 그 폴더에 쥬피터 노트북 들어가기
- 오른쪽과 같이 to do로 비워져 있는 부분 코드 채워넣기
- 저장은 필수!

```
54 def encoder(x, n_code, phase_train):
55     with tf.variable_scope("encoder"):
56         ##### to do #####
57
58
59
60
61
62     #####
```

Deep autoencoder

❖ 오토인코더 구조



Deep autoencoder

❖ 모델 구조와 하이퍼파라미터 설정 – autoencoder_mnist.py

```
# Architecture
n_encoder_hidden_1 = 1000
n_encoder_hidden_2 = 500
n_encoder_hidden_3 = 250
n_decoder_hidden_1 = 250
n_decoder_hidden_2 = 500
n_decoder_hidden_3 = 1000

# Parameters
learning_rate = 0.01
training_epochs = 1000
batch_size = 100
```

Deep autoencoder

❖ layer 함수 – autoencoder_mnist.py

```
def layer(input, weight_shape, bias_shape, phase_train):  
    weight_init = tf.random_normal_initializer(stddev=(1.0/weight_shape[0])**0.5)  
    bias_init = tf.constant_initializer(value=0)  
    W = tf.get_variable("W", weight_shape,  
                        initializer=weight_init)  
    b = tf.get_variable("b", bias_shape,  
                        initializer=bias_init)  
    logits = tf.matmul(input, W) + b  
    return tf.nn.sigmoid(layer_batch_norm(logits, weight_shape[1], phase_train))
```

- 인자로 받은 가중치와 편향을 초기화한다.
- 모델은 $WX + b$ 를 사용하고 활성화 함수로 sigmoid를 사용하여 나오는 출력값을 반환한다.

Deep autoencoder

❖ 인코더 구현하기 – autoencoder_mnist.py

```
def encoder(x, n_code, phase_train):
    with tf.variable_scope("encoder"):
        ##### to do #####
        with tf.variable_scope("hidden_1"):
            hidden_1 = layer(x, [784, n_encoder_hidden_1], [n_encoder_hidden_1], phase_train)
        with tf.variable_scope("hidden_2"):
            hidden_2 = layer(hidden_1, [n_encoder_hidden_1, n_encoder_hidden_2], [n_encoder_hidden_2], phase_train)
        with tf.variable_scope("hidden_3"):
            hidden_3 = layer(hidden_2, [n_encoder_hidden_2, n_encoder_hidden_3], [n_encoder_hidden_3], phase_train)
        with tf.variable_scope("code"):
            code = layer(hidden_3, [n_encoder_hidden_3, n_code], [n_code], phase_train)

    return code

#####
```

- layer(input, weight, bias, phase_train)
- 784 → 1000 → 500 → 250 → 2

Deep autoencoder

❖ 디코더 구현하기 – autoencoder_mnist.py

```
def decoder(code, n_code, phase_train):
    with tf.variable_scope("decoder"):
        ##### to do #####
        with tf.variable_scope("hidden_1"):
            hidden_1 = layer(code, [n_code, n_decoder_hidden_1], [n_decoder_hidden_1], phase_train)
        with tf.variable_scope("hidden_2"):
            hidden_2 = layer(hidden_1, [n_decoder_hidden_1, n_decoder_hidden_2], [n_decoder_hidden_2], phase_train)
        with tf.variable_scope("hidden_3"):
            hidden_3 = layer(hidden_2, [n_decoder_hidden_2, n_decoder_hidden_3], [n_decoder_hidden_3], phase_train)
        with tf.variable_scope("code"):
            output = layer(hidden_3, [n_decoder_hidden_3, 784], [784], phase_train)

    return output
#####
```

- layer(input, weight, bias, phase_train)
- $2 \rightarrow 250 \rightarrow 500 \rightarrow 1000 \rightarrow 784$

Deep autoencoder

❖ 손실 함수 정의 및 신경망 학습 알고리즘 – autoencoder_mnist.py

```
def loss(output, x):  
    with tf.variable_scope("training"):  
        ##### to do #####  
        l2 = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(output, x)), 1))  
        train_loss = tf.reduce_mean(l2)  
        #####  
        train_summary_op = tf.summary.scalar("train_cost", train_loss)  
        return train_loss, train_summary_op  
  
def training(cost, global_step):  
    optimizer = tf.train.AdamOptimizer(learning_rate=0.001, beta1=0.9, beta2=0.999, epsilon=1e-08,  
                                        use_locking=False, name='Adam')  
    train_op = optimizer.minimize(cost, global_step=global_step)  
    return train_op
```

- 손실값은 입력값과 재구성한 출력값 사이의 거리를 L2 norm으로 계산한다.
- L2 norm 수식: $\|I - O\| = \sqrt{\sum_i (I_i - O_i)^2}$
- 최종 손실값(train_loss)을 생성하기 위해 **미니배치 전체에 걸쳐 이 함수의 평균을 구한다.**
- Adam 최적화 알고리즘을 사용하고, 최종 손실값을 최소화하는 방향으로 신경망을 학습시킨다.

Deep autoencoder

❖ 모델 학습시키기 – autoencoder_mnist.py

```
# Training cycle
for epoch in range(training_epochs):

    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        minibatch_x, minibatch_y = mnist.train.next_batch(batch_size)
        # Fit training using batch data
        _, new_cost, train_summary = sess.run([train_op, cost, train_summary_op], feed_dict={x: minibatch_x, phase_train: True})
        train_writer.add_summary(train_summary, sess.run(global_step))
        # Compute average loss
        avg_cost += new_cost/total_batch

    # Display logs per epoch step
    if epoch % display_step == 0:
        print("Epoch:", '%04d' % (epoch+1), "cost =", "{:.9f}".format(avg_cost))

    saver.save(sess, "mnist_autoencoder_hidden=" + n_code + "_logs/model-checkpoint-" + '%04d' % (epoch+1), global_step=global_step)
```

- 학습의 최종 목적은 cost를 최소화 시키는 것.
- 1 epoch 당 평균 최종 손실값을 출력한다.
- phase_train: True면 학습, False면 테스트
- 각 epoch 당 코드층(인코더 마지막 층) 모델의 체크포인트 저장도구(saver)를 사용한다.

Deep autoencoder

❖ 프로그램 실행하기 – autoencoder_mnist.py

- 명령줄 파라미터를 받아들여 실행하기 때문에 명령 프롬프트에서 코드를 실행해야한다.
- Window + R키를 눌러 cmd를 관리자 권한으로 또 하나를 실행한다.
- 아래와 같이 입력하여 가상환경을 실행시키고 프로그램 파일이 있는 폴더로 다시 이동한다.
- 다음과 같은 입력으로 실행하여 오토인코더 모델을 학습시킨다.

```
E:\DenoisingAE>activate tensorflow_dae  
  
(tensorflow_dae) E:\DenoisingAE>python autoencoder_mnist.py 2
```

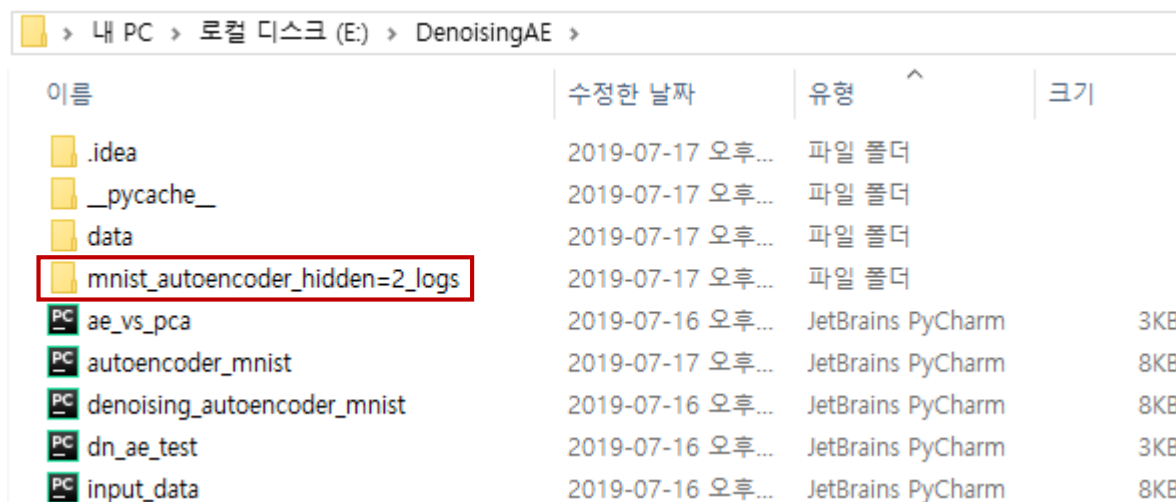
- .py 뒤에 붙은 2는 인코더의 마지막 출력으로 나오는 뉴런의 개수로 정해준 것이다. (code)

Deep autoencoder

❖ 실행 결과

```
Epoch: 0001 cost = 11.684345055
Validation Loss: 9.885018
Epoch: 0002 cost = 9.505442881
Validation Loss: 8.773004
Epoch: 0003 cost = 8.445695796
Validation Loss: 8.347751
Epoch: 0004 cost = 7.869748344
Validation Loss: 7.4505854
Epoch: 0005 cost = 7.405649343
Validation Loss: 7.3392873
Epoch: 0006 cost = 7.133315698
Validation Loss: 6.988289
Epoch: 0007 cost = 6.926423908
Validation Loss: 6.995893
```

- 다음과 같이 epoch 당 모델의 checkpoint가 저장된 폴더가 생성된다.



이름	수정한 날짜	유형	크기
.idea	2019-07-17 오후...	파일 폴더	
__pycache__	2019-07-17 오후...	파일 폴더	
data	2019-07-17 오후...	파일 폴더	
mnist_autoencoder_hidden=2_logs	2019-07-17 오후...	파일 폴더	
ae_vs_pca	2019-07-16 오후...	JetBrains PyCharm	3KB
autoencoder_mnist	2019-07-17 오후...	JetBrains PyCharm	8KB
denoising_autoencoder_mnist	2019-07-16 오후...	JetBrains PyCharm	8KB
dn_ae_test	2019-07-16 오후...	JetBrains PyCharm	3KB
input_data	2019-07-16 오후...	JetBrains PyCharm	8KB

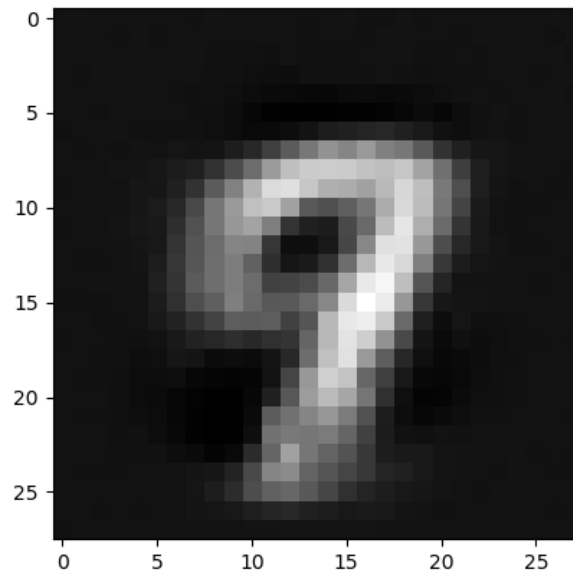
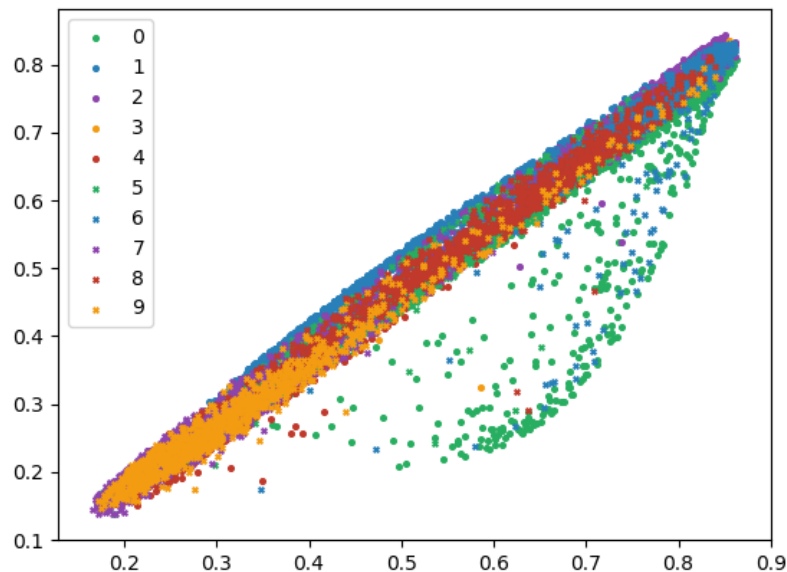
Deep autoencoder

❖ 프로그램 실행하여 테스트하기

- python ae_vs_pca.py [모델 체크포인트 절대 경로]

C:\> 관리자: 명령 프롬프트

```
(tensorflow_dae) C:\>python ae_vs_pca.py C:\>practive\ae_checkpoint\model-checkpoint-0971-53405
```



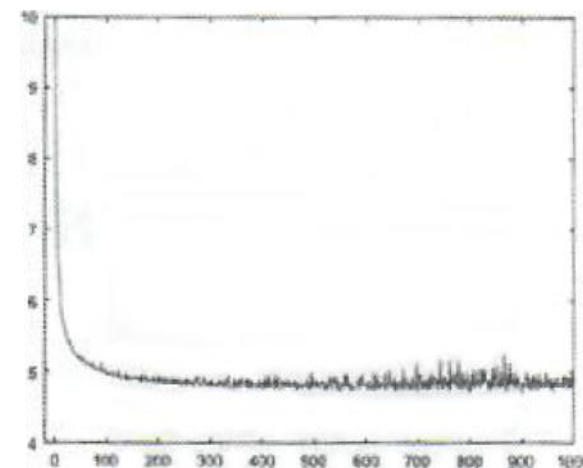
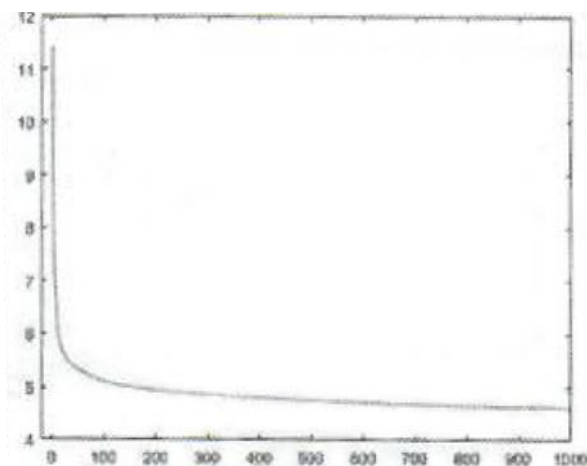
Deep autoencoder

❖ 실행 결과 - 손실값 변화

- 손실값이 성공적으로 점점 줄어드는 것을 확인할 수 있다.

```
Epoch: 0001 cost = 11.684345055  
Validation Loss: 9.885018  
Epoch: 0002 cost = 9.505442881  
Validation Loss: 8.773004  
Epoch: 0003 cost = 8.445695796  
Validation Loss: 8.347751  
Epoch: 0004 cost = 7.869748344  
Validation Loss: 7.4505854  
Epoch: 0005 cost = 7.405649343  
Validation Loss: 7.3392873  
Epoch: 0006 cost = 7.133315698  
Validation Loss: 6.988289  
Epoch: 0007 cost = 6.926423908  
Validation Loss: 6.995893
```

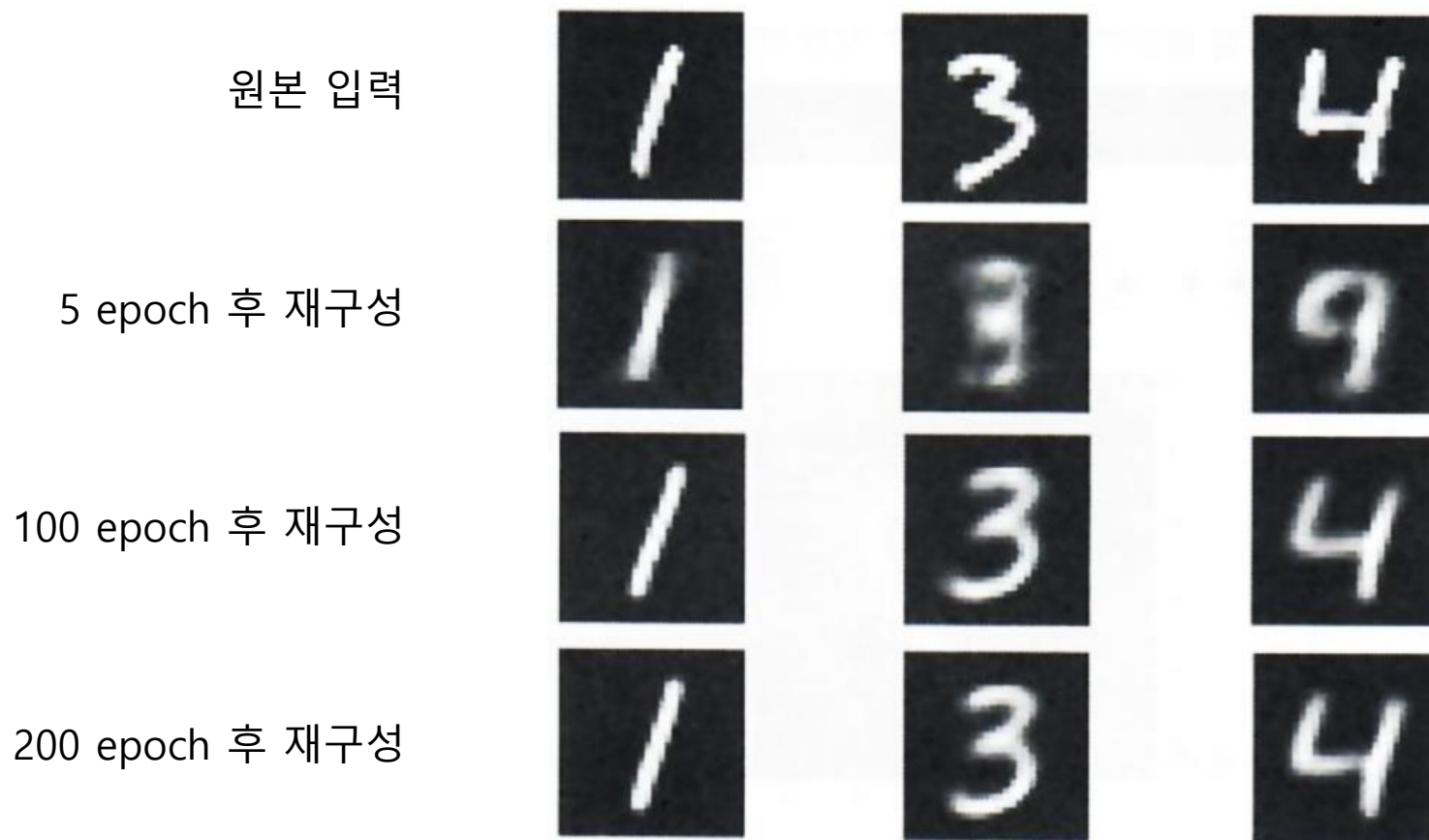
- 총 1000 epoch의 학습 기간에 걸친 손실 값 변화를 시각화하면 아래와 같다.
- 학습과 검증 비용 모두 점점 평평해질 때까지 감소한다.



Deep autoencoder

❖ 실행 결과

- 각 5, 100, 200 epoch마다 테스트 이미지를 재구성한 이미지들을 비교해본다.

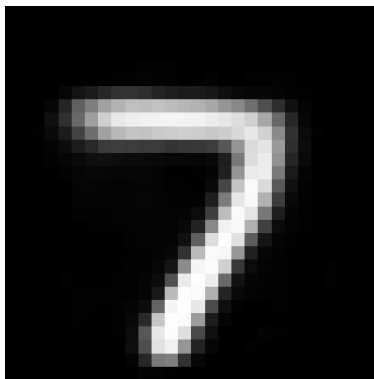


Denoising autoencoder

❖ corrupt_input 함수 – denoising_autoencoder_mnist.py

- 원래 input 이미지에 노이즈를 곱하여 손상된 이미지를 생성한다.
- 0~2 사이 정수 (0, 1, 2)를 tf.random_uniform 함수를 사용하여 균일 분포로 입력값 x 크기만큼 초기화해 준다.

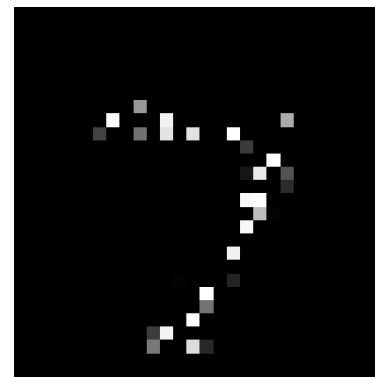
```
def corrupt_input(x):  
    ##### to do #####  
    corrupting_matrix = tf.random_uniform(shape=tf.shape(x), minval=0, maxval=2, dtype=tf.int32)  
    return x * tf.cast(corrupting_matrix, tf.float32)  
    #####
```



*



=



Denoising autoencoder

❖ 손상된 이미지 생성 – denoising_autoencoder_mnist.py

- Corrupt가 placeholder로 생성이 되는데, 1이면 corrupt_input이 c_x 가 되고, 0이면 x 가 c_x 가 된다. 즉, 1이면 입력을 손상시키고, 0이면 손상시키지 않는다.
- 인코더 신경망에 대한 입력이 x 대신 손상된 c_x 가 된다.

```
##### to do #####  
corrupt = tf.placeholder(tf.float32)  
  
c_x = (corrupt_input(x) * corrupt) + (x * (1 - corrupt))  
  
code = encoder(c_x, int(n_code), phase_train)  
#####
```

Denoising autoencoder

❖ loss function – denoising_autoencoder_mnist.py

- 기본 오토인코더와 손실 함수는 동일하다.
- 손실값을 구할 때 c_x 와 output의 차이값이 아닌 원본 이미지의 x 와 output의 차이로 정의하였기 때문에 손상 이미지가 아닌 원본 이미지와 유사하게 재구성하도록 학습시킨다.


```
def loss(output, x):  
    with tf.variable_scope("training"):  
        l2 = tf.sqrt(tf.reduce_sum(tf.square(tf.subtract(output, x)), 1))  
        train_loss = tf.reduce_mean(l2)  
        train_summary_op = tf.summary.scalar("train_cost", train_loss)  
    return train_loss, train_summary_op
```

```
c_x = (corrupt_input(x) * corrupt) + (x * (1 - corrupt))  
  
code = encoder(c_x, int(n_code), phase_train)  
#####  
  
output = decoder(code, int(n_code), phase_train)  
  
cost, train_summary_op = loss(output, x)
```


Denoising autoencoder

❖ 프로그램 실행하기 - denoising_autoencoder_mnist.py

- 다음과 같은 입력으로 실행하여 디노이징 오토인코더 모델을 학습시킨다.

 관리자: 명령 프롬프트

```
(tensorflow_dae) C:\practice>python denoising_autoencoder_mnist.py2
```

- .py 뒤에 붙은 '2'는 인코더의 마지막 출력으로 나오는 뉴런의 개수로 정해진 것이다. (encode)

Denoising autoencoder

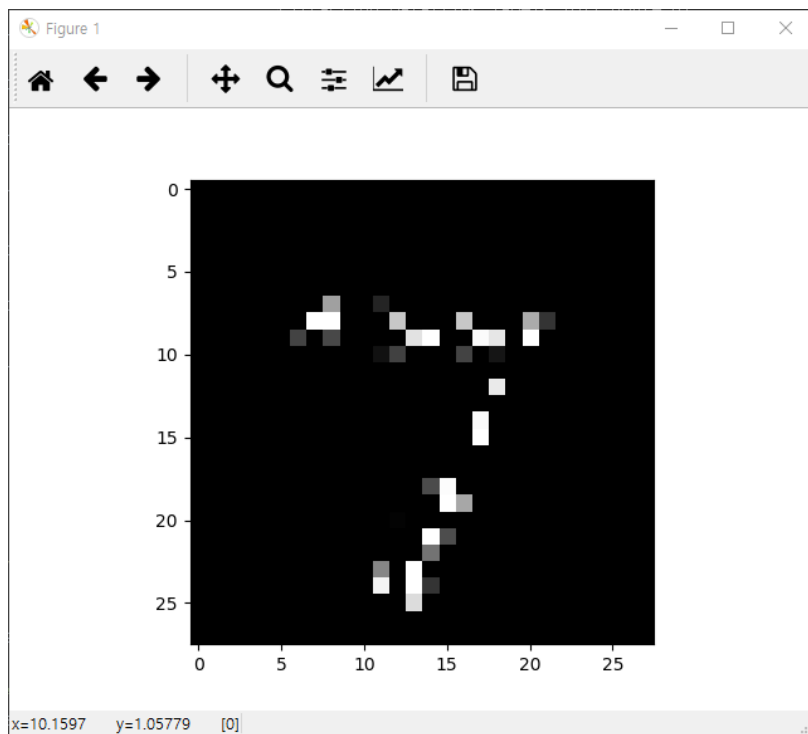
❖ 프로그램 실행하여 테스트하기

- python dn_ae_test.py [모델 체크포인트 절대 경로]

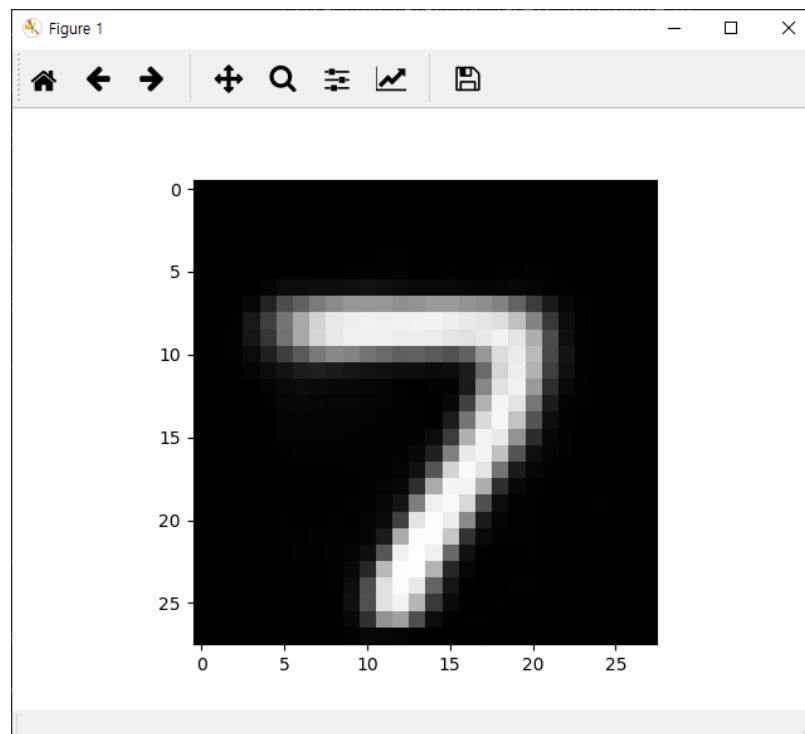
선택 관리자: 명령 프롬프트

```
(tensorflow_dae) C:\practice>python ae_vs_pca.py C:\practice\ae_checkpoint\model-checkpoint-0971-534050
```

noise_input



reconstruction



Denoising autoencoder

❖ 디노이징 오토인코더 결과 예시

원본 이미지



손상 이미지



재구성 이미지



- 데이터셋에 손상 작업을 적용하고 손상되지 않은 원본 이미지들을 재구성한 이미지 비교
- 빠진 픽셀들을 잘 채워주는 것을 확인할 수 있다.