



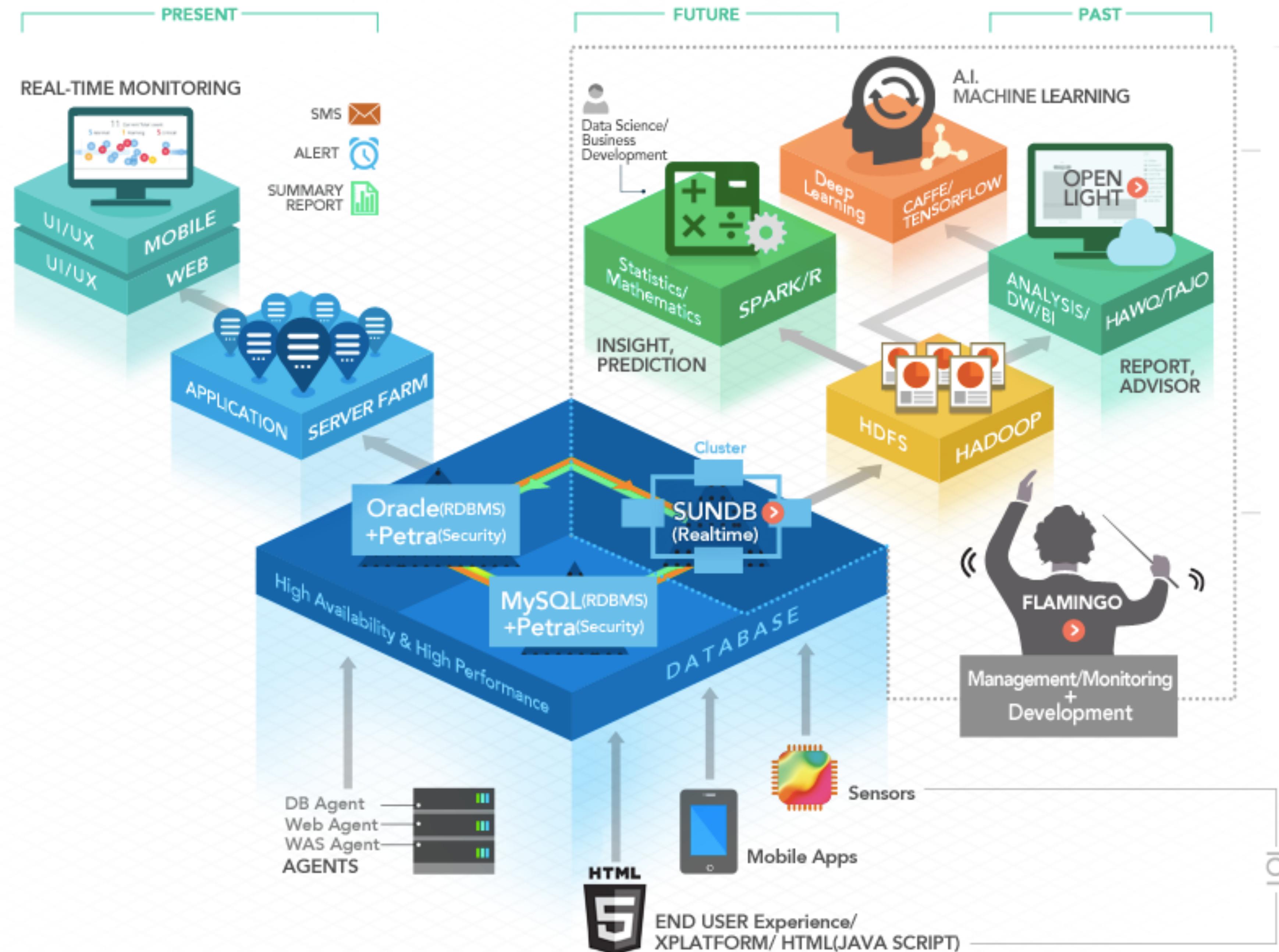
[171121] 엑셈아카데미

# 딥러닝 이론에서 실습까지



엑셈-포스텍 R&D 센터  
머신러닝팀 이도엽

# (주) 엑셈



## □ Coursera

Machine Learning (Andrew Ng.)

Neural Network for Machine Learning (Jeff. Hinton)

## □ Standford Course

cs231n (Convolutional Neural Networks for Visual Recognition)

cs224d (Deep Learning for Natural Language Processing)

cs229 (Machine Learning)

cs20si (Tensorflow, Student Lecture)

## □ 모두를 위한 머신러닝/딥러닝 (김성훈 교수)

## □ Facebook - Tensorflow KR (TF-KR) Group

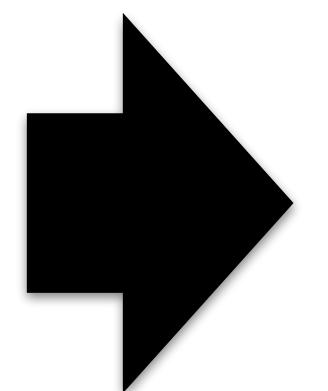
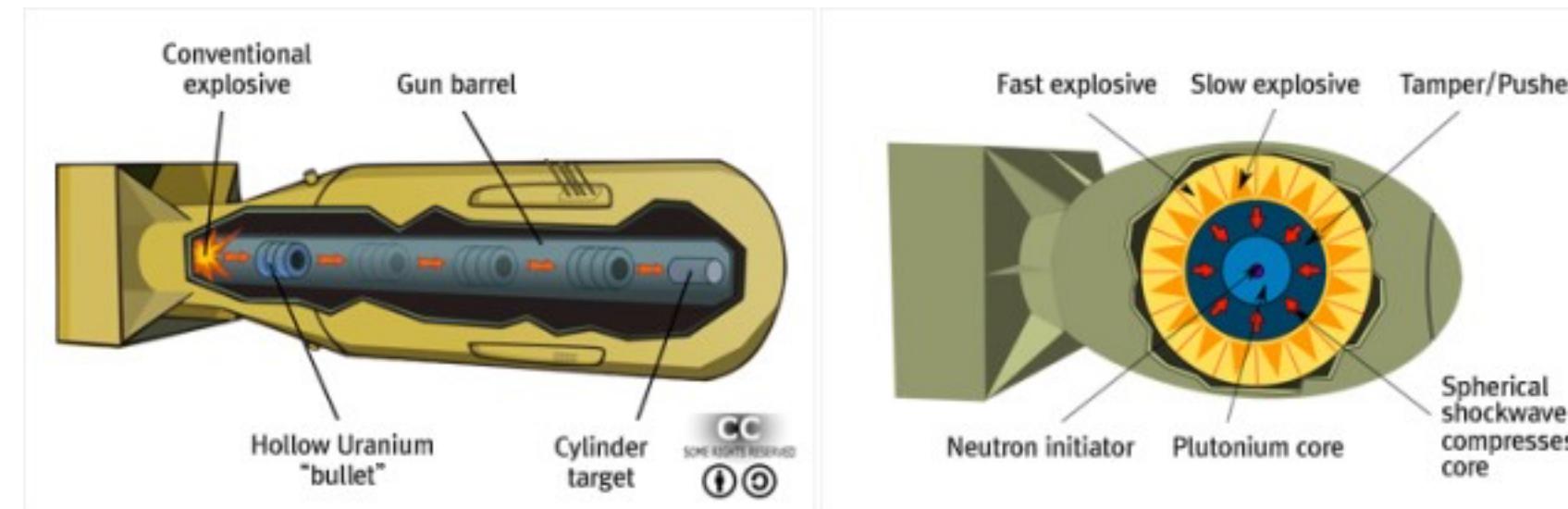
# Artificial Intelligence Machine Learning and Deep Learning

# Manhattan Project

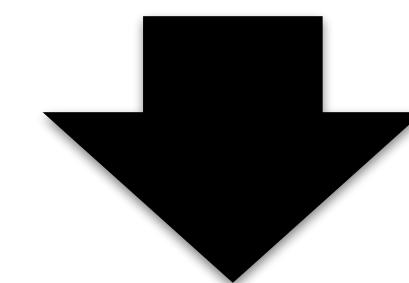
## ▣ 맨해튼 프로젝트 (Manhattan Project)

- 히틀러 나치 정권에 앞서 원자폭탄을 만들기 위한 미국의 비밀 프로젝트
- 미국 내 최고 과학자들이 모두 참여하여, 원자폭탄을 만드는 연구에 참여

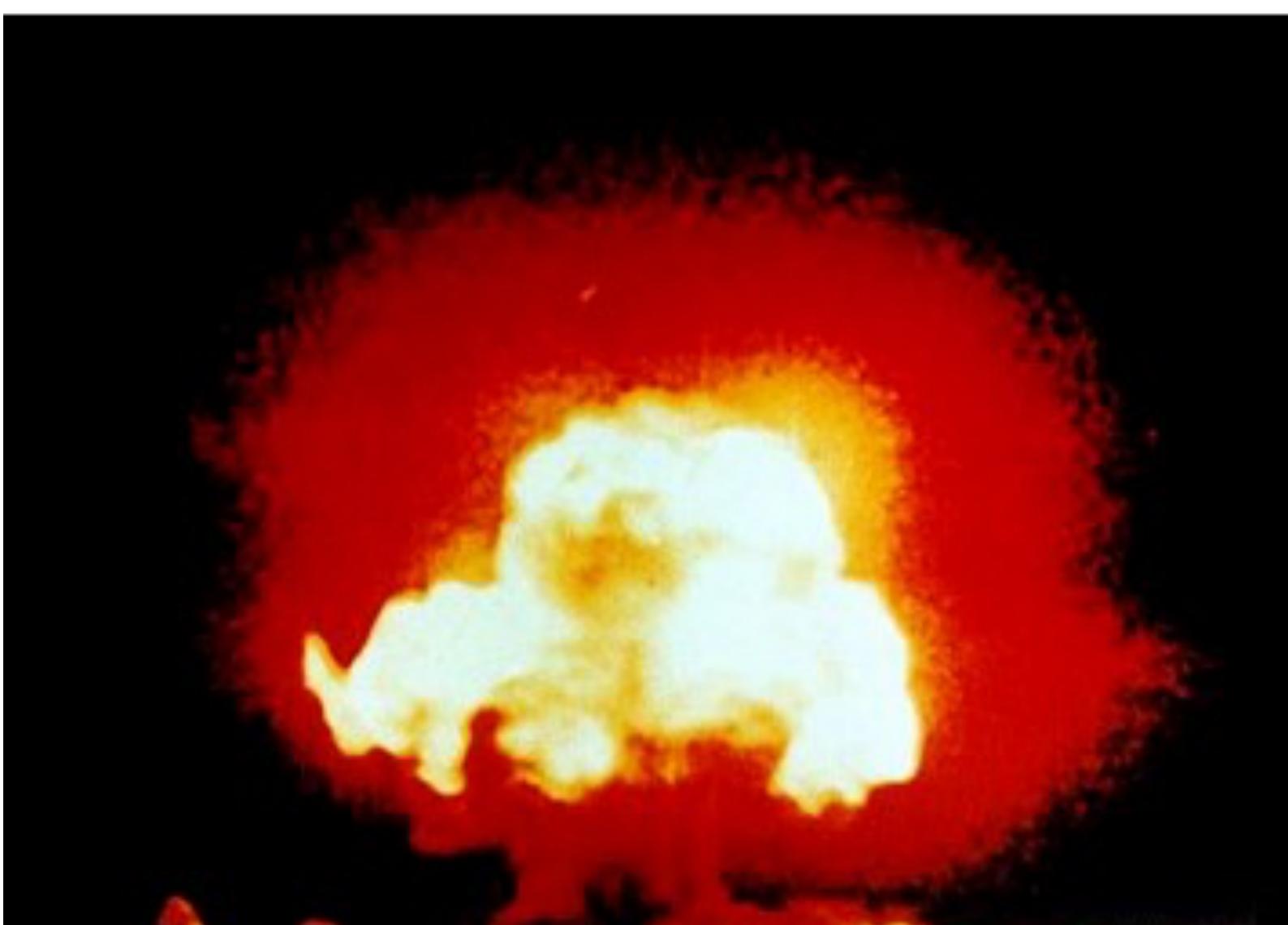
<최초의 원자폭탄 구성>



- 연구비 20억 달러
- 고용인원 최대 13만명



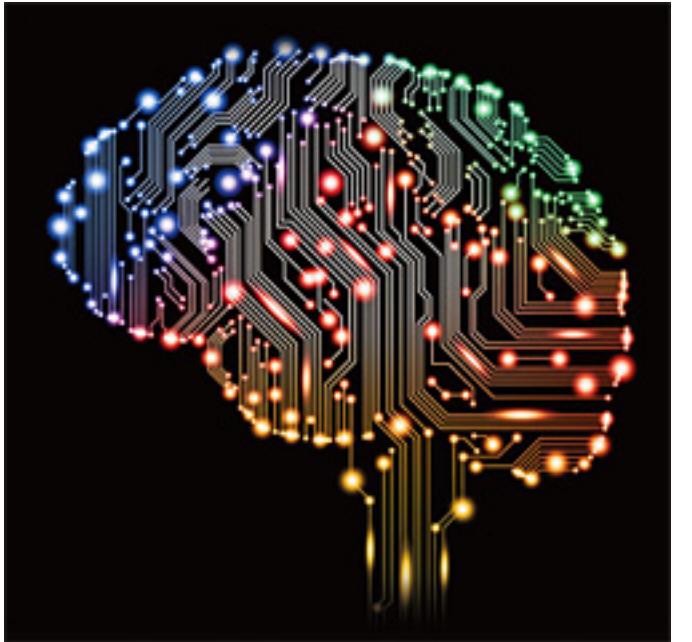
태평양 전쟁 승리  
냉전 체제에서 미국의 우위



# AI Manhattan Project of Google

## □ 인공지능 맨해튼 프로젝트 (AI Manhattan Project)

- 인공지능 기술 개발을 위한 구글의 세기적 프로젝트
- 관련 IT 기업과 전문가들을 확보하기 위한 거대 인수합병을 진행

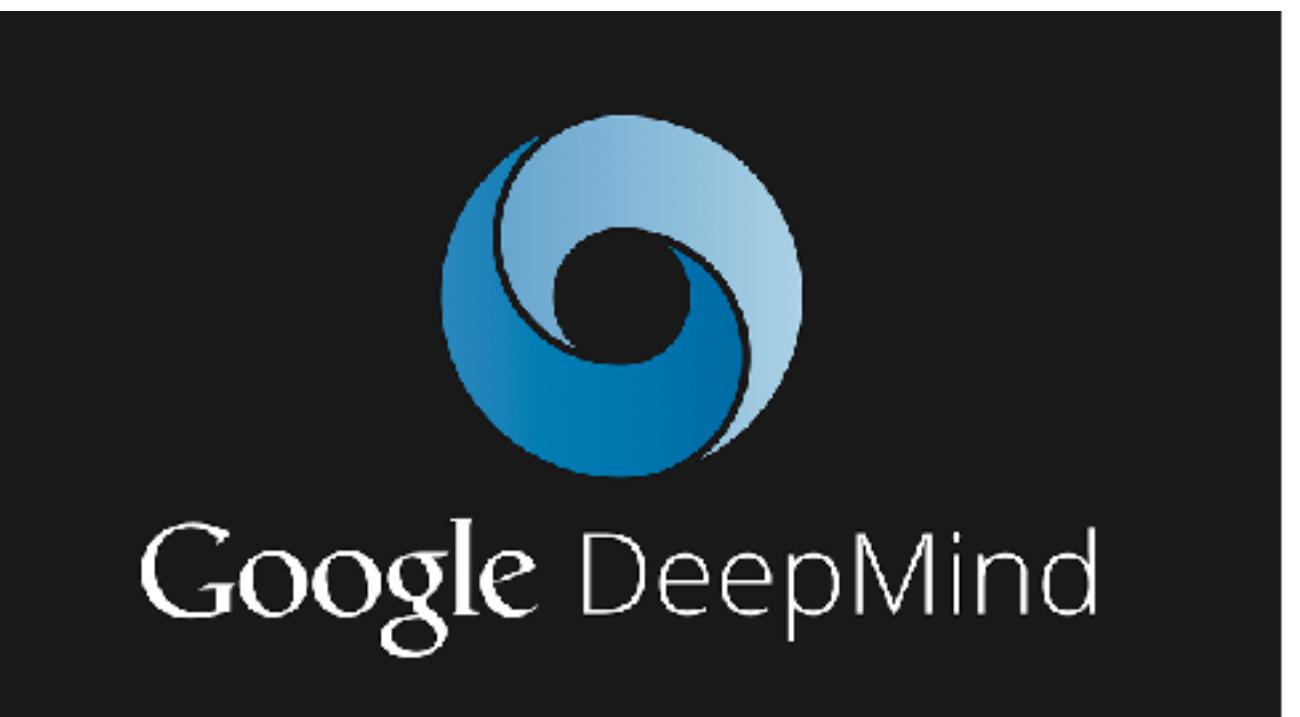


□ 1000명 이상의 인력이 구글의 인공지능/머신러닝 연구 진행 중 (아주경제)

□ 2014년 1월 Deep Mind 인수 (6억 5000만 달러)

□ 2014년 1월 Nest Labs 인수 (32억 달러)

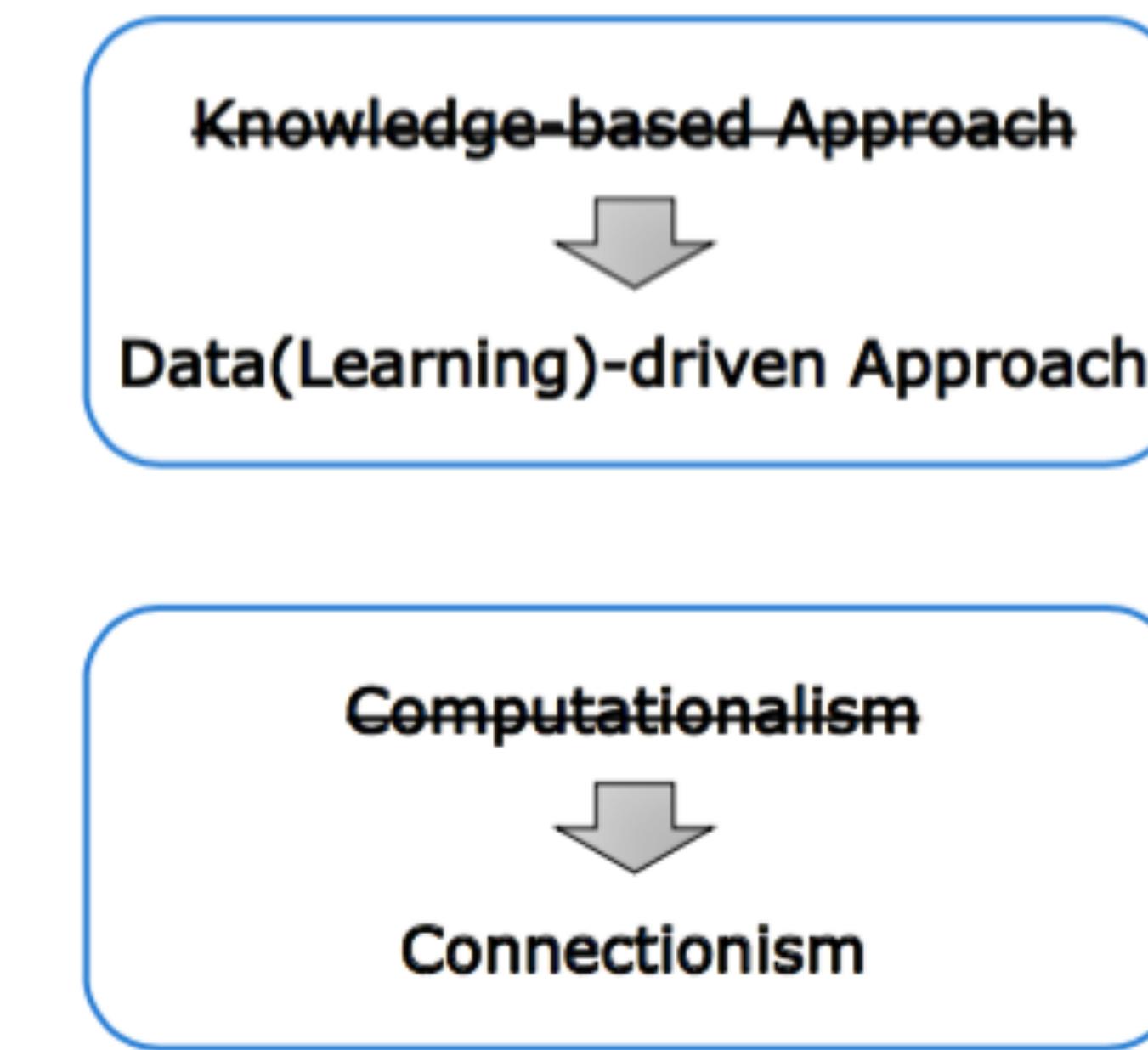
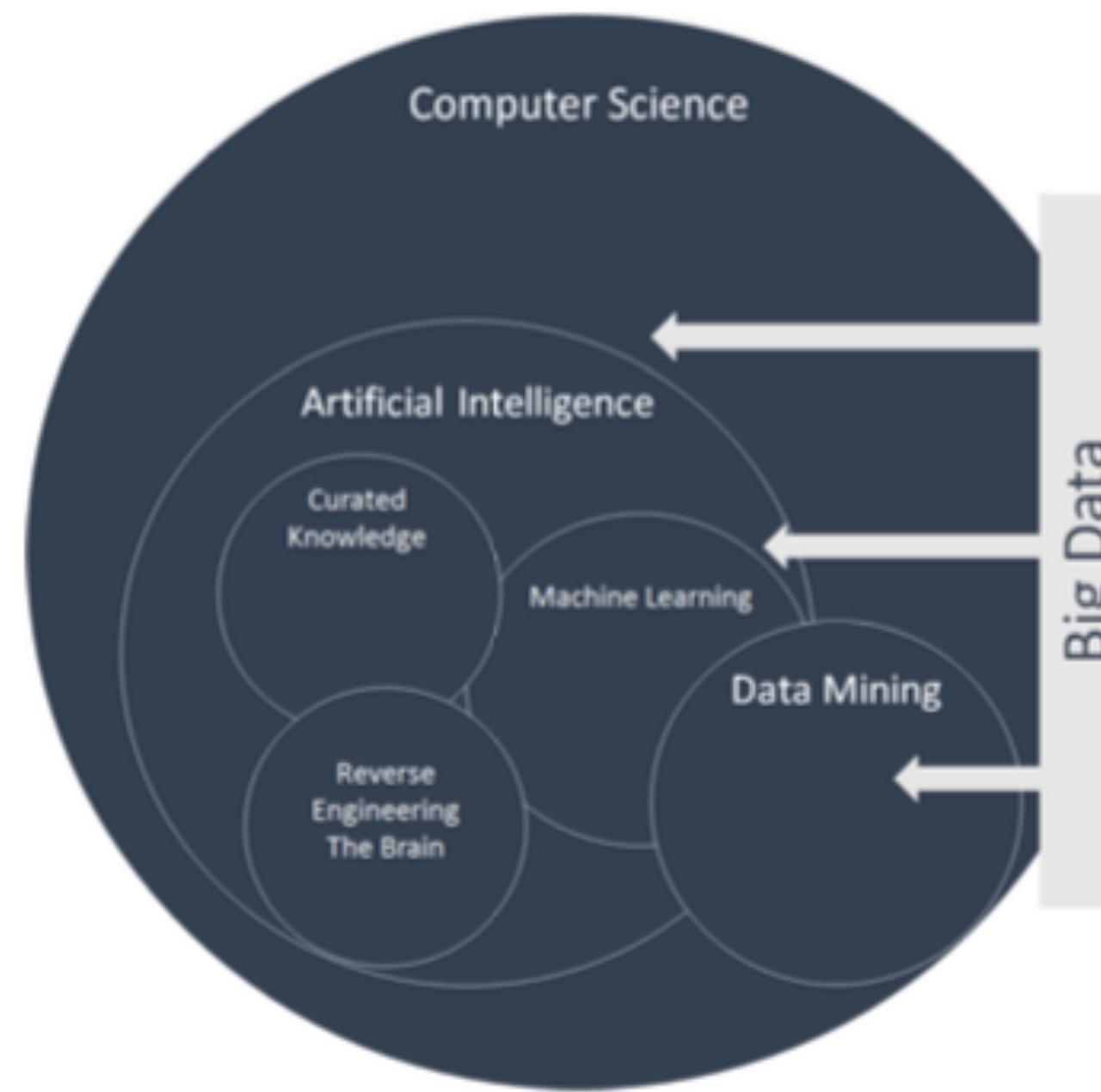
□ 기타 50여 개의 인공지능 기업들이 맨해튼 프로젝트에 참여, 연구 진행



# Background

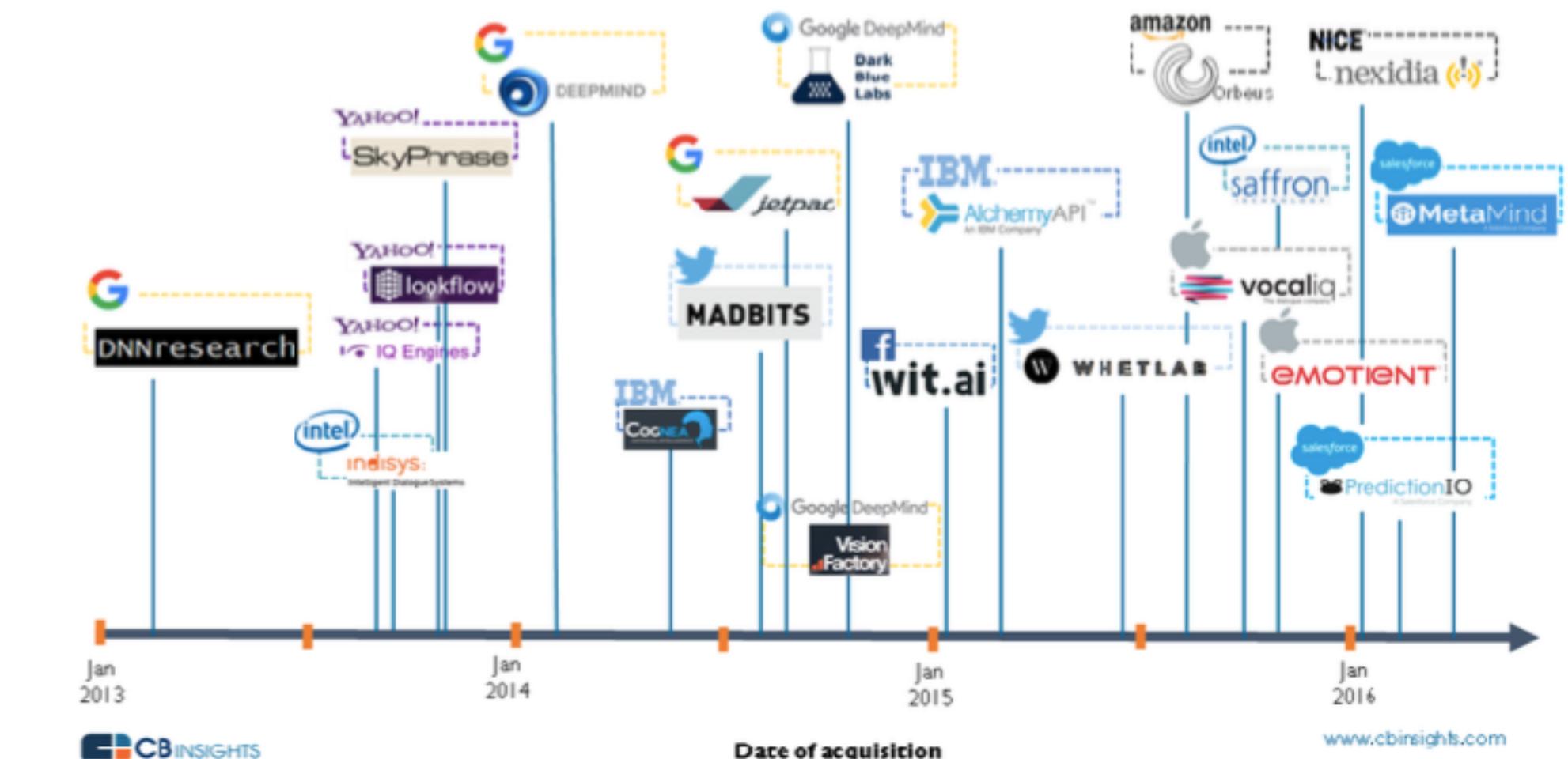
- **빅데이터(Big Data) 기술의 확산과 컴퓨팅 속도의 향상과 함께, 인공지능 시장은 빠른 속도로 성장하고 있음**

## <Artificial Intelligence Approach>



## <AI Tech Giants>

Race To AI: Major Acquisitions In Artificial Intelligence

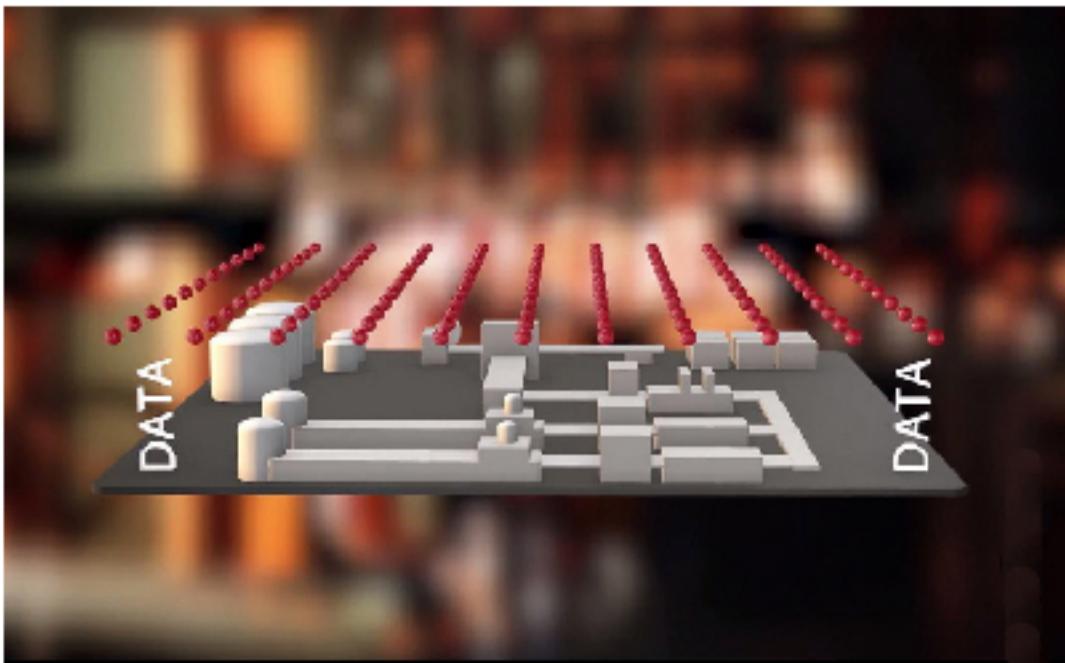


수많은 기업들이 인공지능 분야 머신러닝 / 딥러닝 기술 확보 진행을 위해 노력하고 있음

# Application of AI

## □ 현재 인공지능 기술은 분야를 가리지 않고 다양하게 이용되고 있음

<스마트 팩토리 적용 사례>



이미지: 크크원 오토메이션, 빙진·본지

<구글의 자율주행자동차>



<금융 지표 예측>

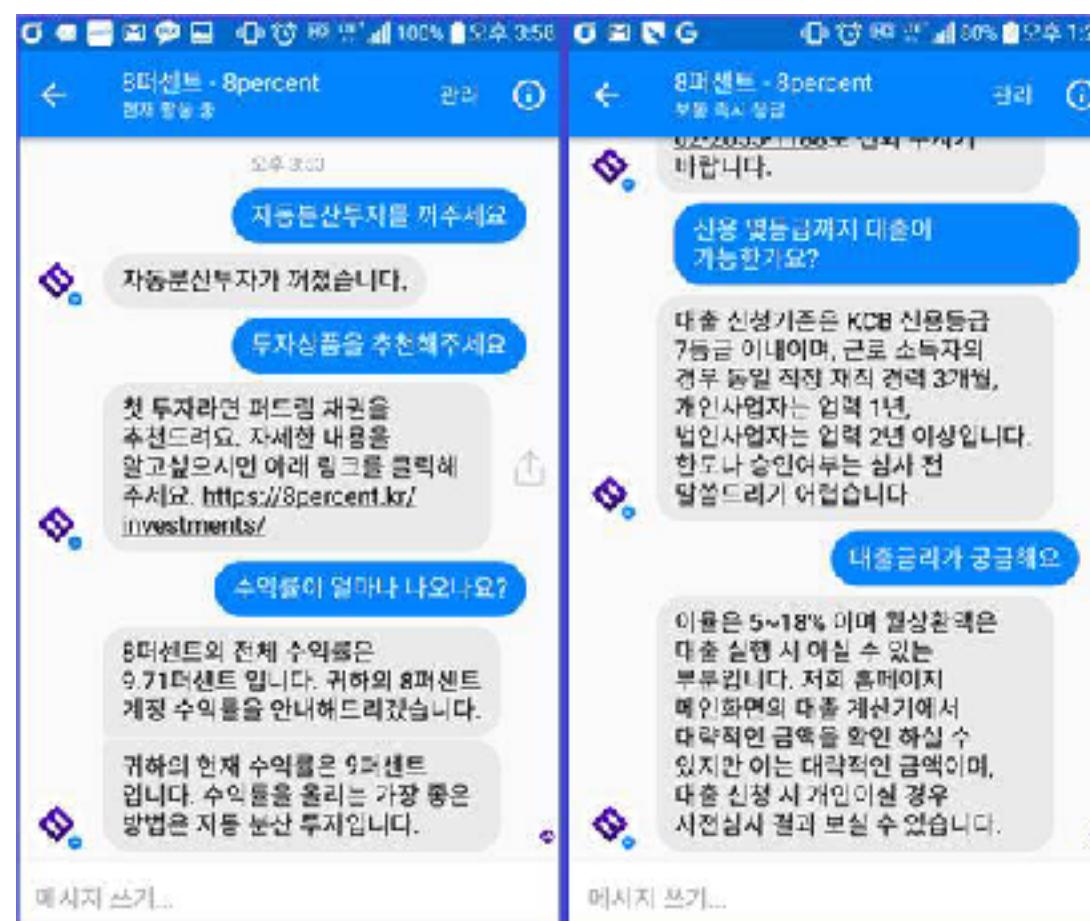


<영유아 발달 관리>



인공지능으로 영유아 발달 관리 하는 '씨모거어 AI 키친(사진: 엘트리시아)

<핀테크 업체 챗봇 예시>



<음악 작곡 등 예술 창작>



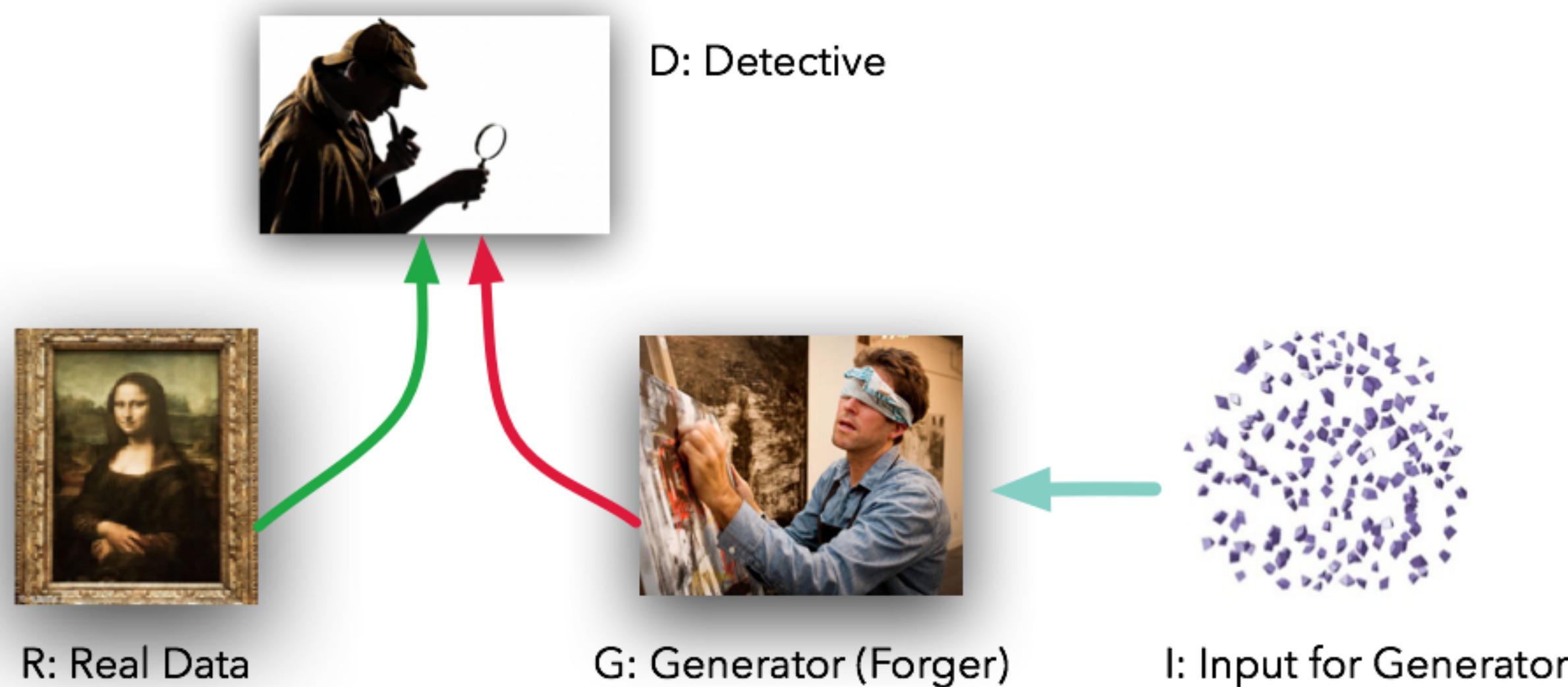
# 인공지능 기술 발전의 현재

- 현재 인공지능 기술은 비전(Vision)과 자연어 처리 (Natural Language Processing), 음성인식 (Speech Recognition) 등의 분야를 필두로 빠르게 발전하고 있음
- Discriminative Model의 급격한 발전과 함께 Generative Model의 성능 또한 빠르게 발전하고 있는 추세



# Generative Model Examples

## □ GAN (Generative Adversarial Networks)



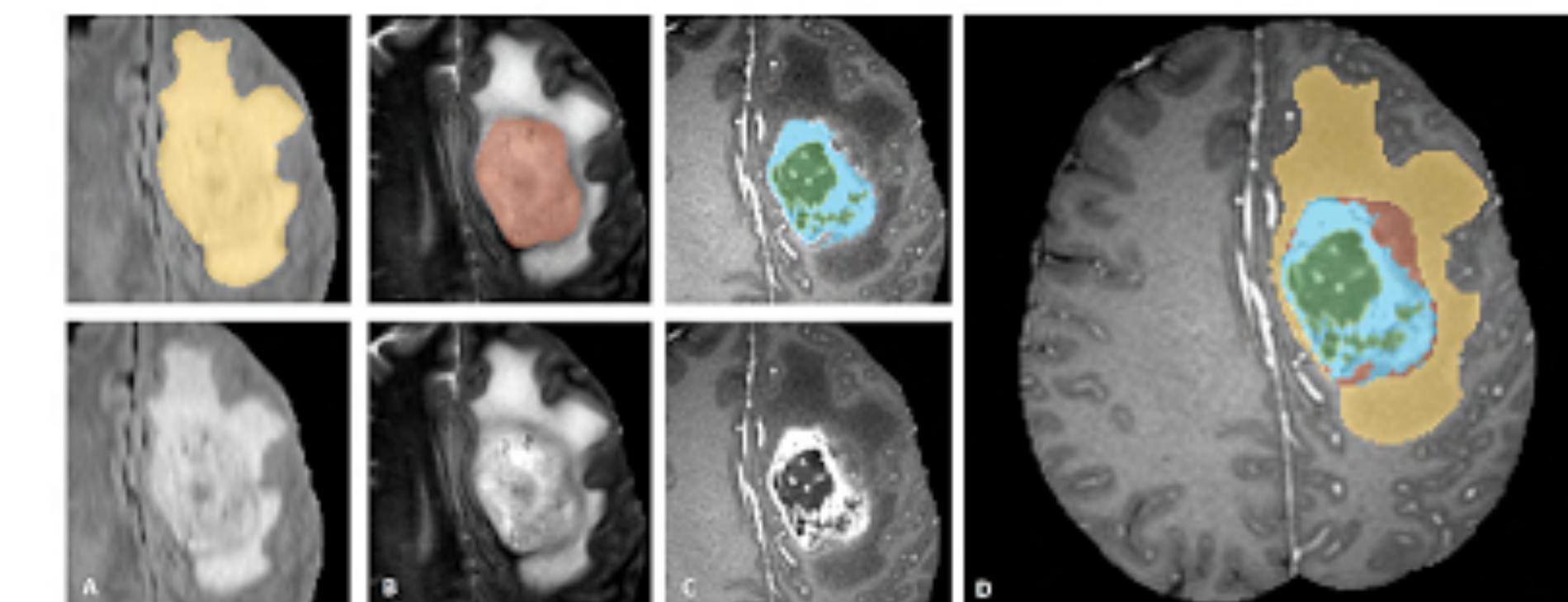
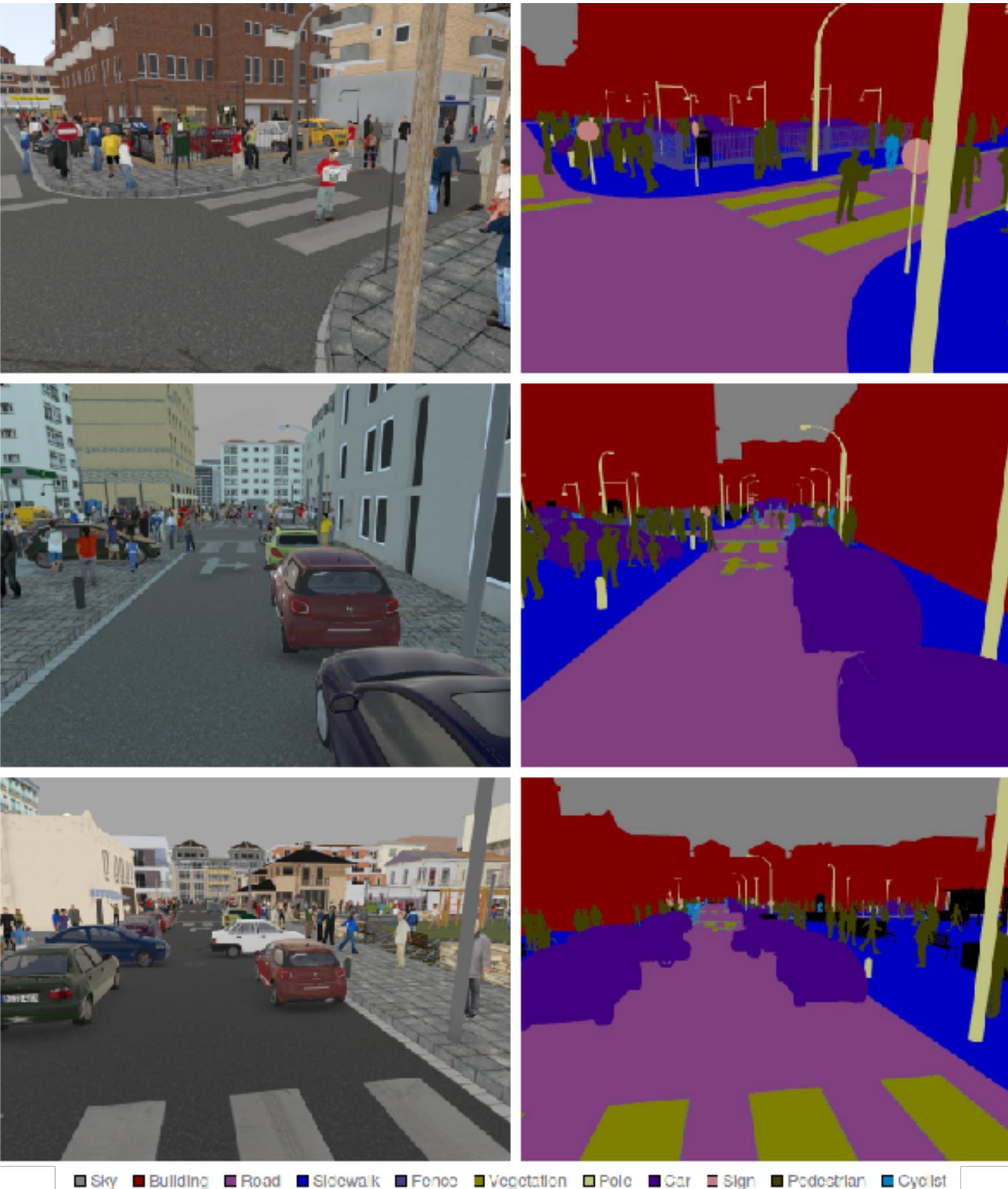
# Vision Application of Deep Learning

## □ Image Detection and Recognition



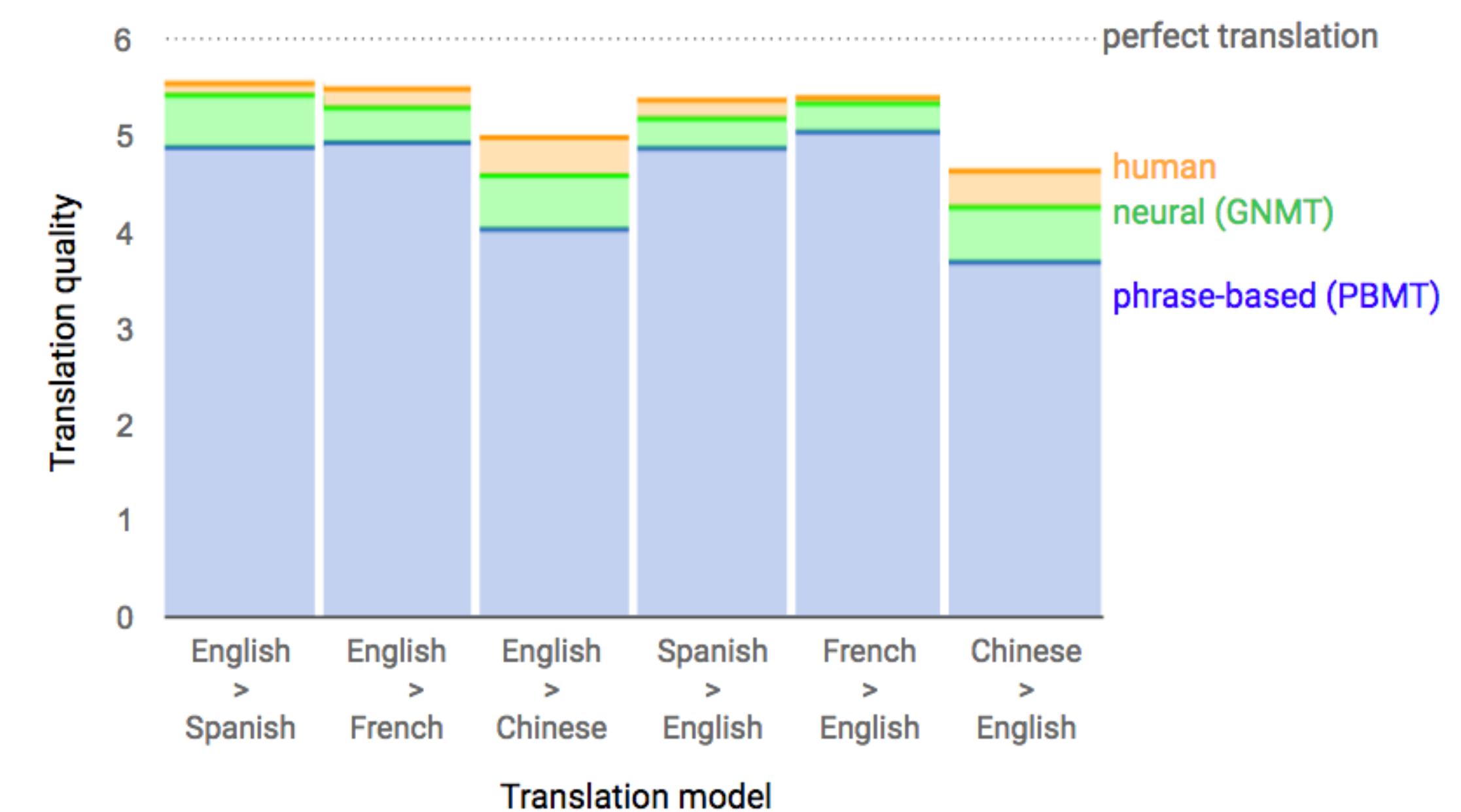
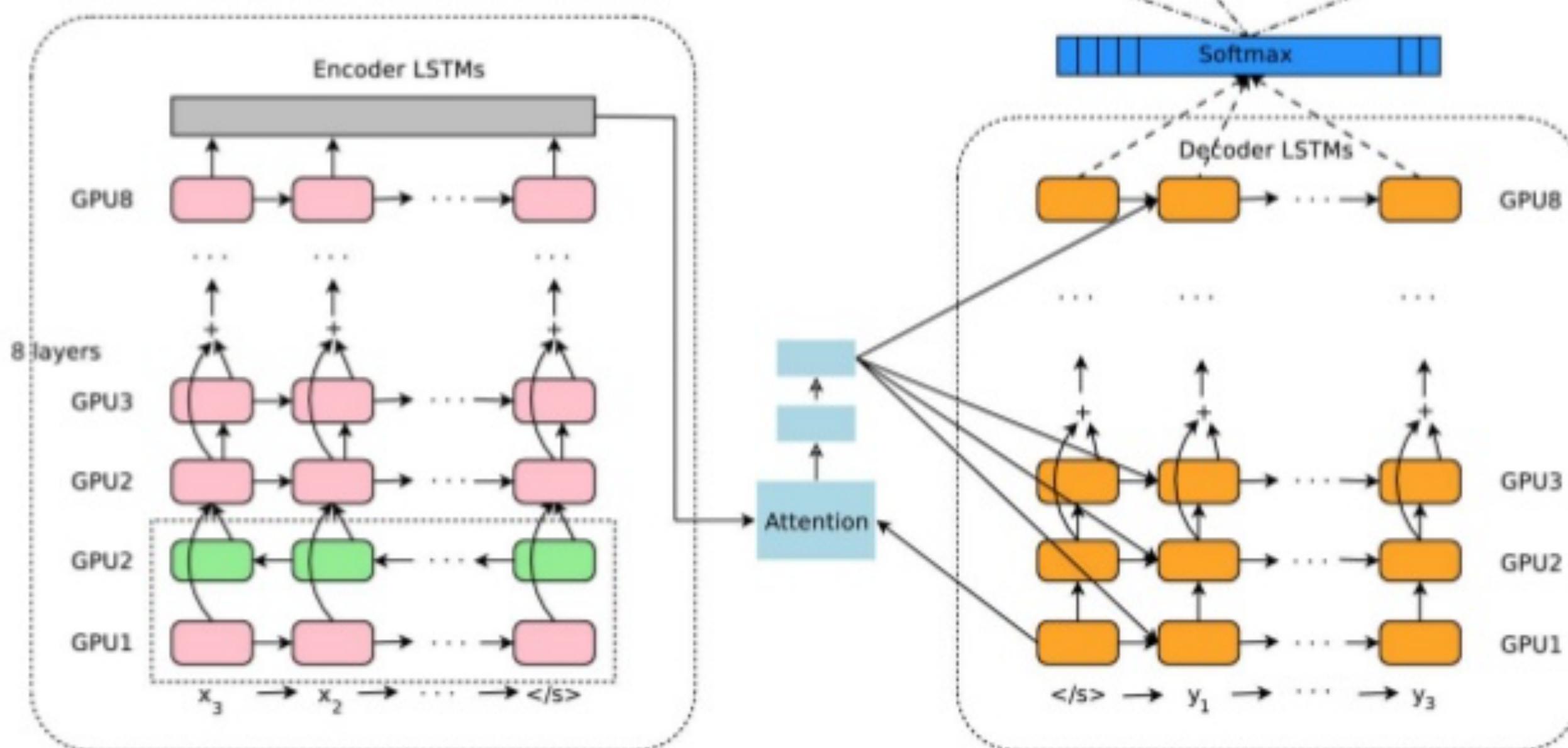
# Image Segmentation and Applications

## □ Image Segmentation



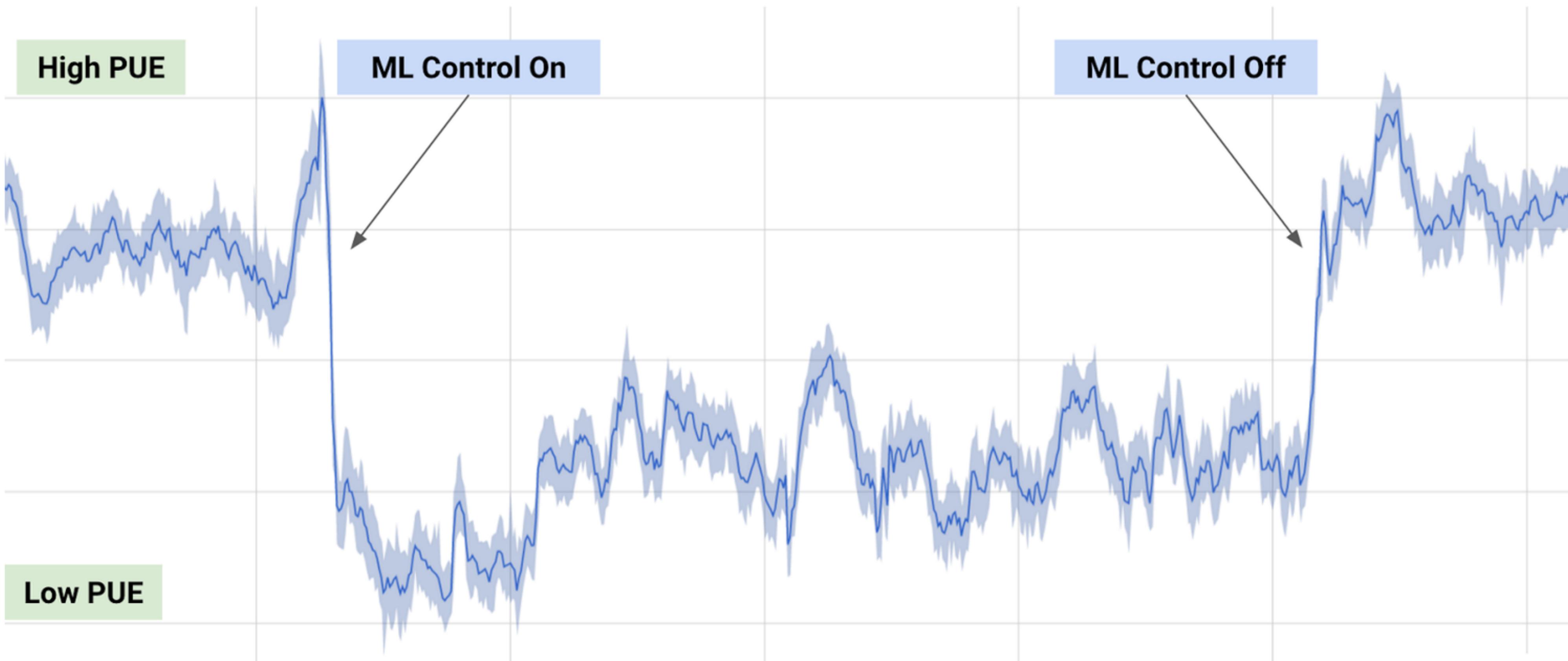
## □ Google's Neural Machine Translator

### Google NMT Architecture



# Reinforcement Learning for Data Center Control

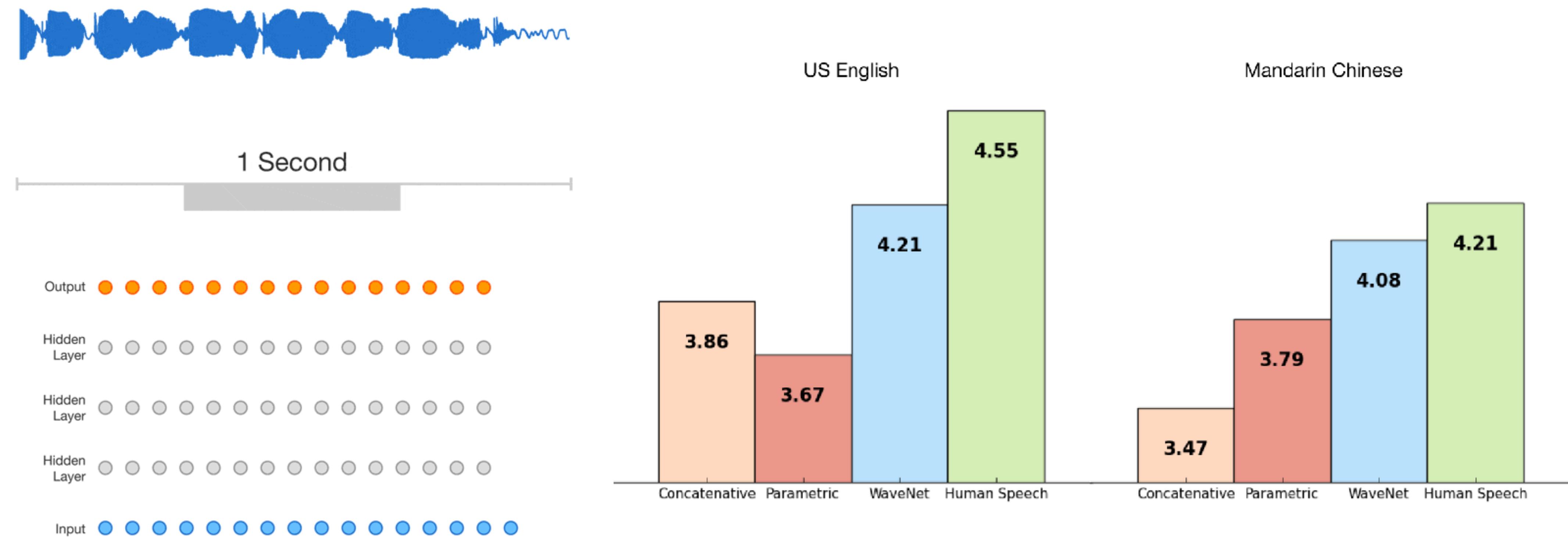
- 데이터 센터의 발열량을 조절하기 위해서 인공지능 기술이 적용되기도 함



# Generative Model for Audio (WaveNet)

- Deep Mind가 발표한 WaveNet 모델은 음성 합성 (Speech Synthesis) 또는 TTS (Text-to-Speech) 분야에서 높은 성능을 보여줌

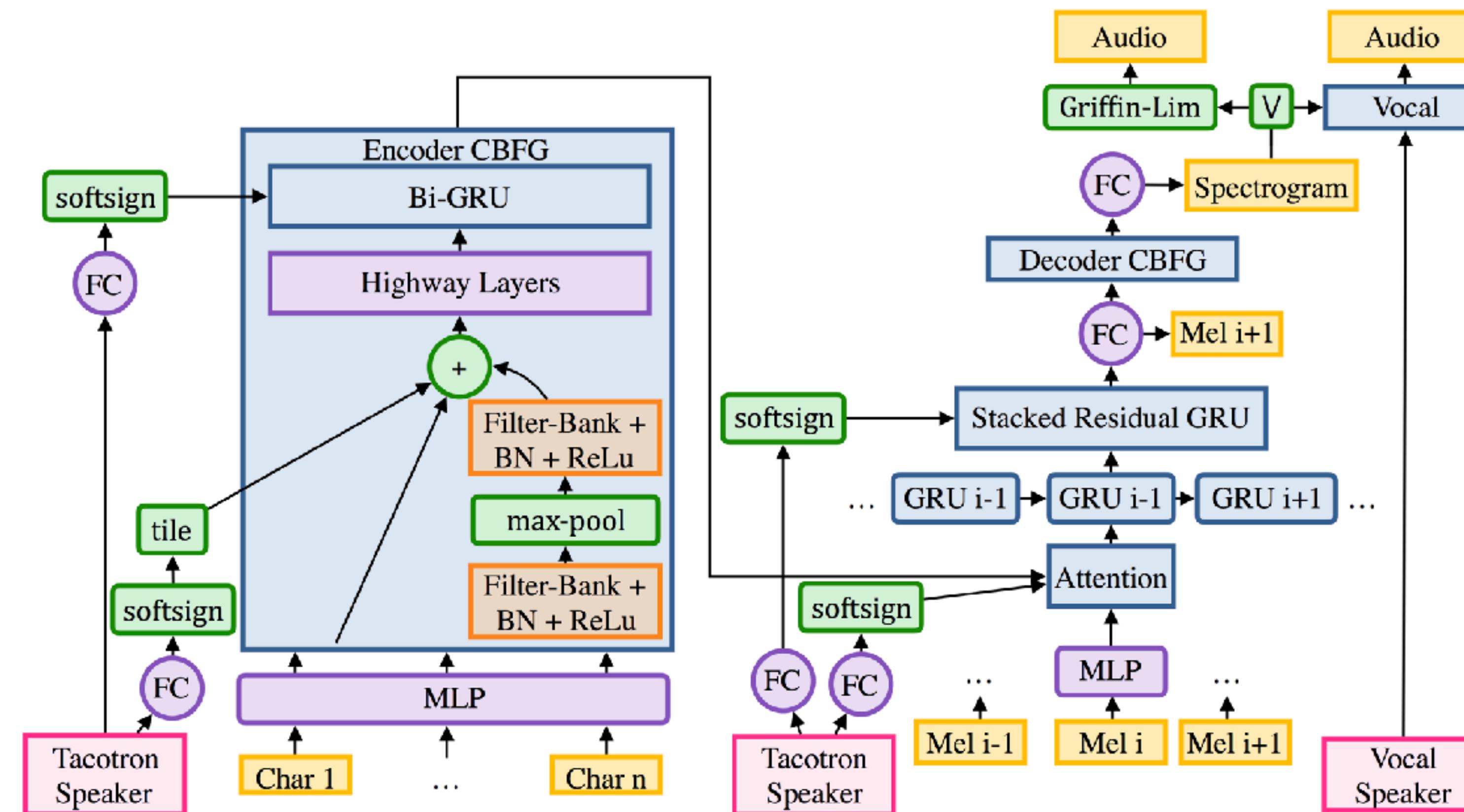
<https://carpedm20.github.io/tacotron/>



# Generative Model for Audio (Deep Voice)

- Deep Mind가 발표한 WaveNet 모델은 음성 합성 (Speech Synthesis) 또는 TTS (Text-to-Speech) 분야에서 높은 성능을 보여줌

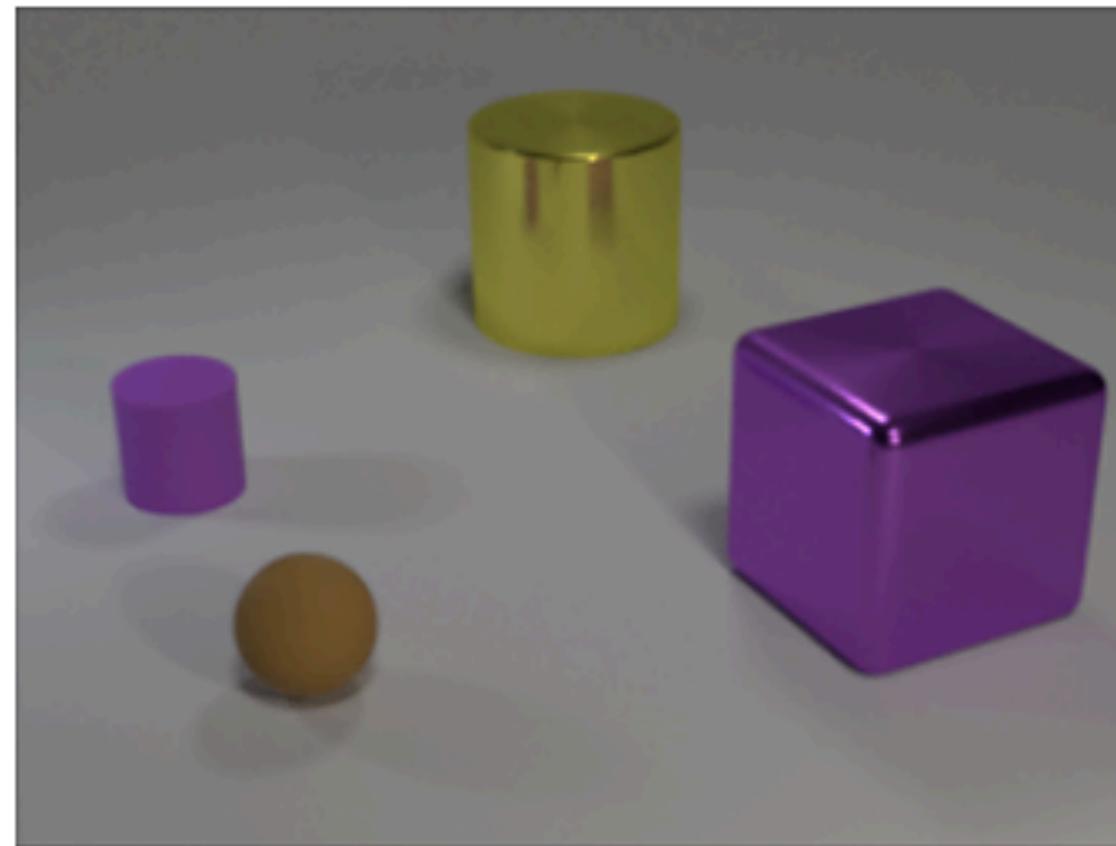
## <Deep Voice 3 Architecture>



# Neural Network for Relational Reasoning

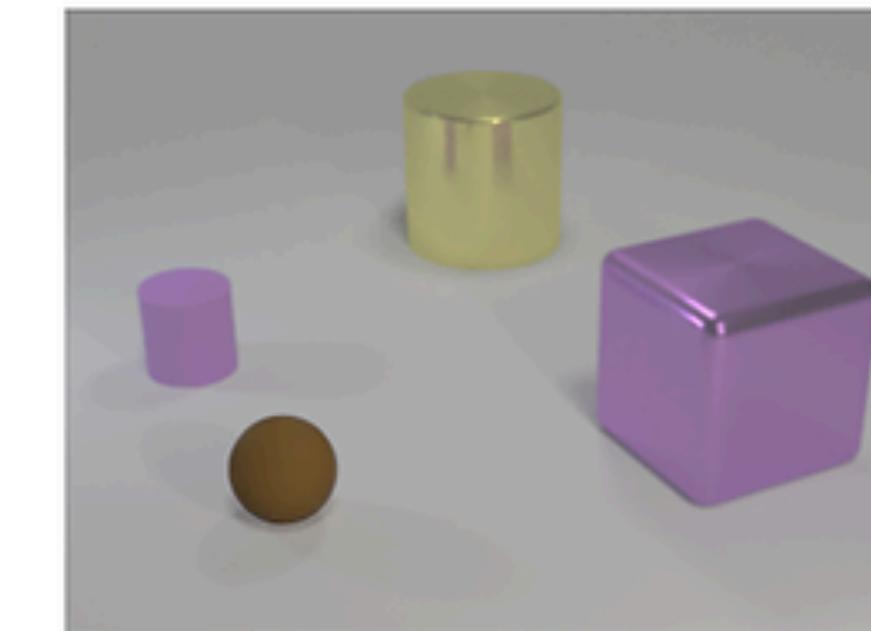
- 최근 Neural Network 기반의 모델은 관계형 추론 (Relational Reasoning) 분야에서도 높은 성능을 보여주기 시작

**Original Image:**



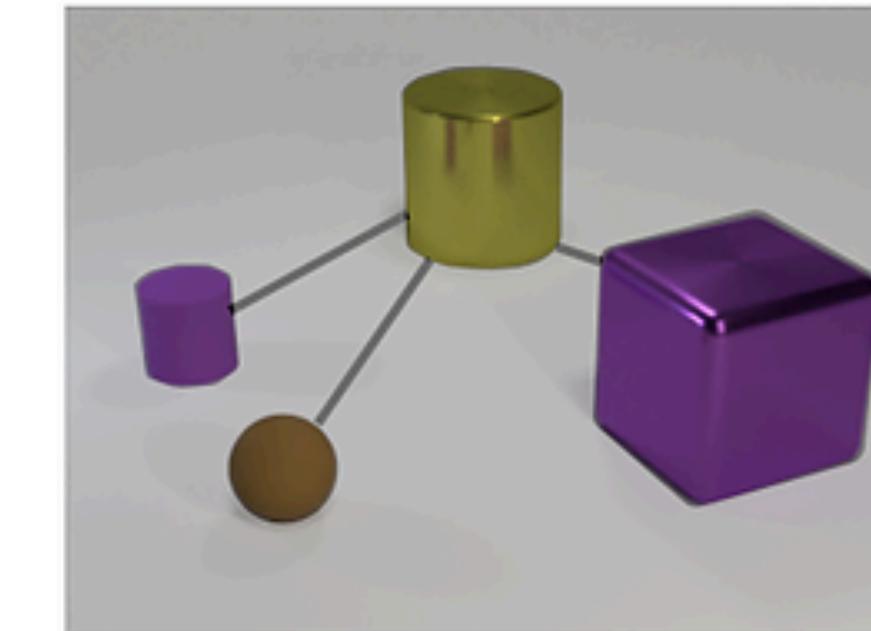
**Non-relational question:**

What is the size of  
the brown sphere?



**Relational question:**

Are there any rubber  
things that have the  
same size as the yellow  
metallic cylinder?



# AlphaGo Zero

□ 이번에 Deep Mind에 의해 공개된 AlphaGo Zero는 “Self-play” Reinforcement Learning으로 발전이 핵심

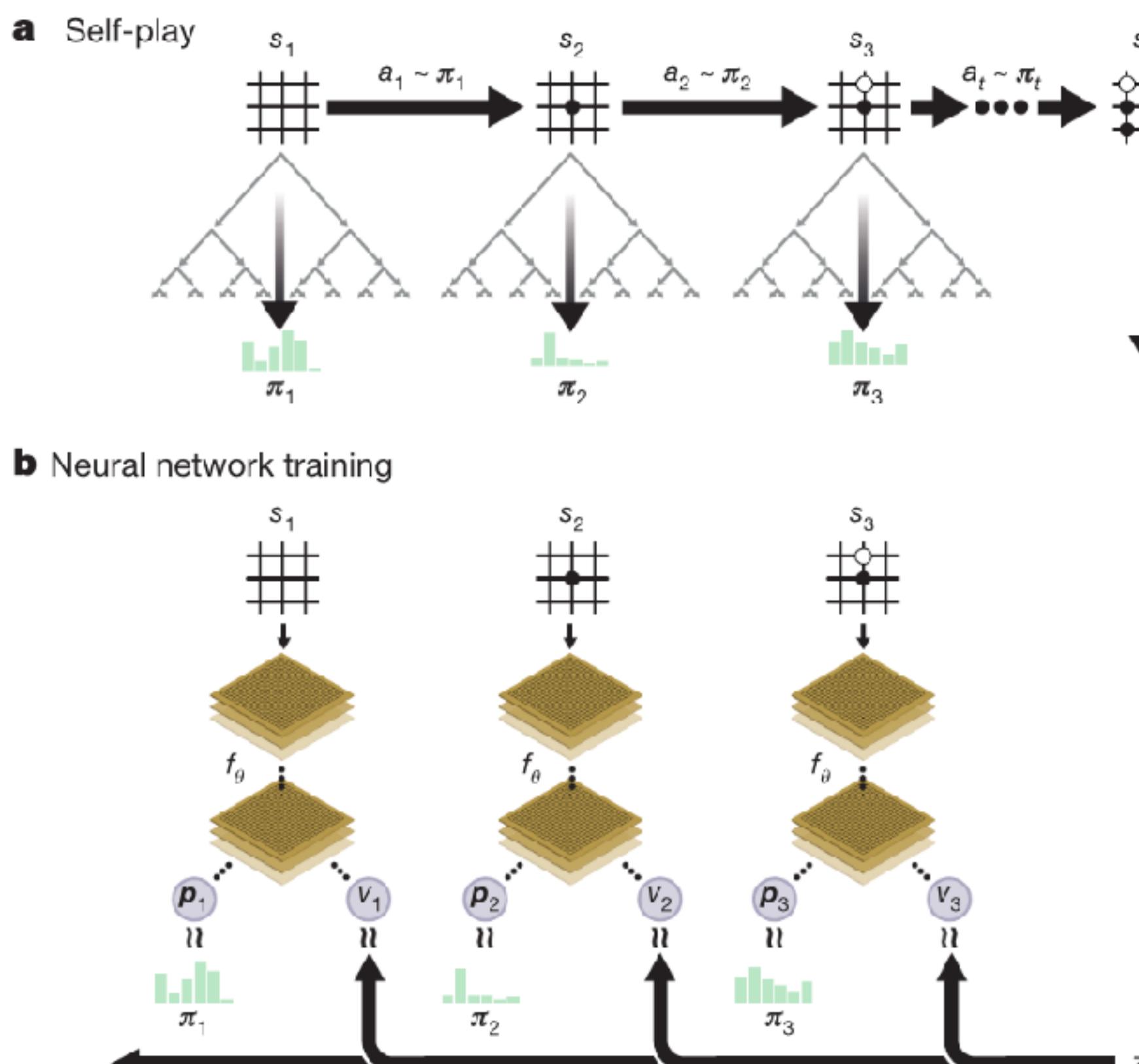
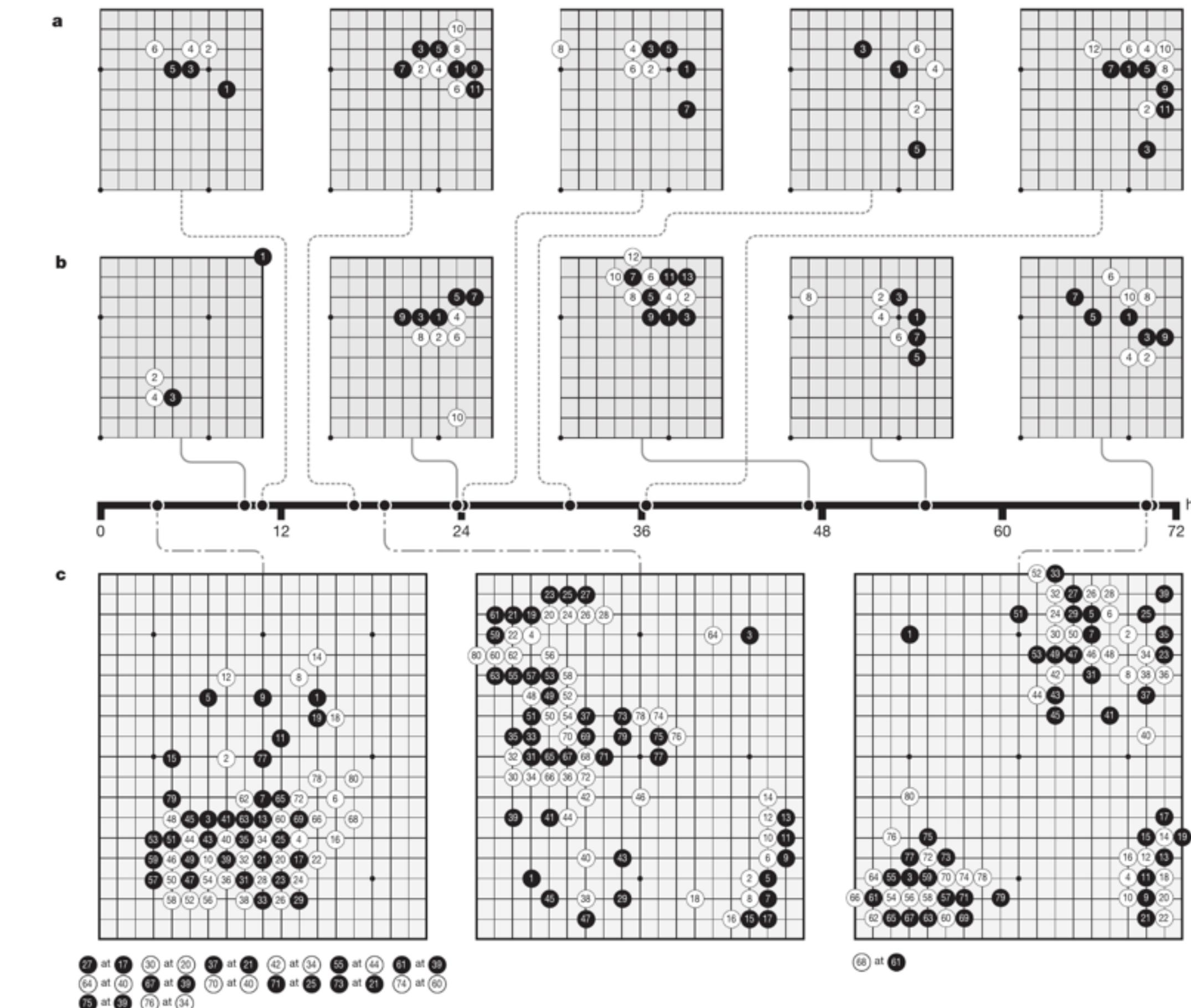


Figure 1 | Self-play reinforcement learning in AlphaGo Zero. a, The



- 4차 산업혁명은 인공지능을 중심으로 ICT 융합을 통해 이루어짐
- 인공지능을 바탕으로한 4차 산업혁명의 의미는 기존과는 다르며, 현재 진행형임

## - 4차 산업혁명

: 사물인터넷(IoT)와 인공지능(AI)를 바탕으로 제품과 제조공정이 지능화되는 것

**'AI 기술개발·인재육성'...미래부 지능정보 추진안 발표(종합)**

국가기전뉴스통신사  
연합뉴스

기사입력 2016-12-15 10:36 | 최종수정 2016-12-15 10:42 | 기사원문 | 10 | 3

지능정보기술로 만한 국내 경제효과가 2030년 기준으로 최대 460조 원에 이르고 기존 일자리의 노동시간 중 49.7%가 자동화되리라는 것이 정부의 전망이다.

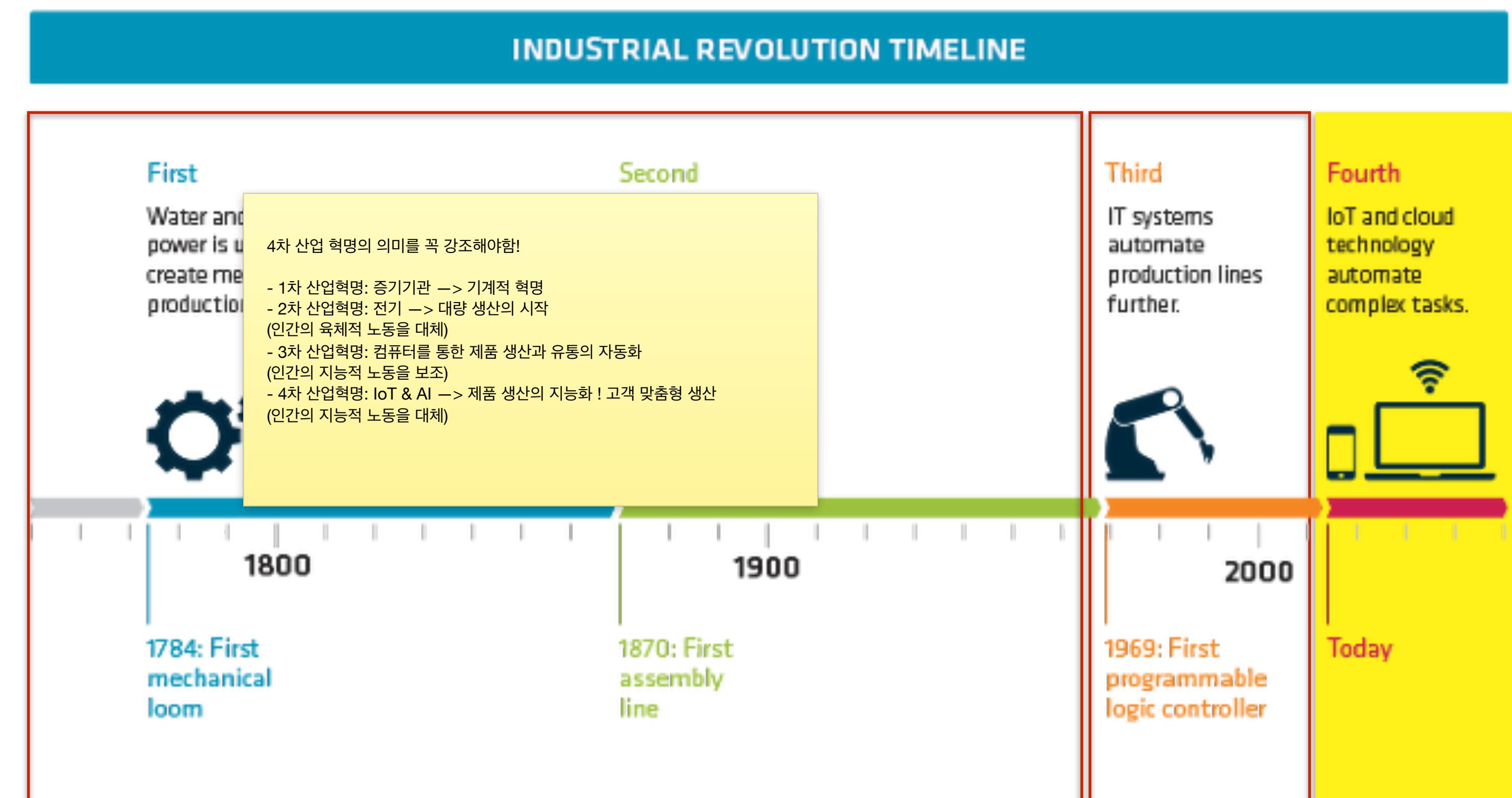
**세계 R&D, 4차 산업혁명 대응..총 4조1335억 확정**

이데일리

기사입력 2017-01-01 12:00 | 최종수정 2017-01-01 12:14 | 기사원문 | 70 | 20

◇인공지능 및 ICBM 투자 확대

인공지능 및 시각지능 등 실용화 제품 개발을 포함한 인공지능 분야와 딥러닝 등 차세대 지능정보 처리 등을 위한 기반 SW 컴퓨팅에 대한 투자가 534 억원에서 736억원으로 늘었다.



## □ 인공 지능 (Artificial Intelligence)

- Artificial intelligence (AI) is **intelligence exhibited by machines**. (**contrast to natural intelligence**)
- In computer science, an **ideal "intelligent" machine is a flexible rational agent** that
  - 1) **perceives its environment** and 2) **takes actions** that 3) **maximize its chance** of success at some goal. (Wikipedia)

The screenshot shows the Google Translate interface. At the top, it says "번역" (Translate). Below that, there are language selection boxes for "한국어" (Korean), "영어" (English), "독일어" (German), and "언어 갑지" (More languages). On the right, there are buttons for "영어" (English), "한국어" (Korean), "일본어" (Japanese), and a "번역하기" (Translate) button. The main area contains two text boxes. The left box contains the English text: "Artificial intelligence (AI) is intelligence exhibited by machines. In computer science, an ideal "intelligent" machine is a flexible rational agent that perceives its environment and takes actions that maximize its chance of success at some goal." The right box contains the Korean translation: "인공 지능 (AI)은 기계가 나타내는 정보입니다. 컴퓨터 과학에서 이상적인 "지능형" 기계는 환경을 인식하고 목표 달성을 기회를 극대화하는 유연한 합리적인 에이전트입니다." At the bottom, there are icons for keyboard, microphone, and a save button labeled "수정 제안하기".

# Types of Artificial Intelligence

- ‘지능’의 규정 범위에 따라 1) Artificial Narrow Intelligence (ANI), 2) AGI (General), 3) ASI (Super) 분류

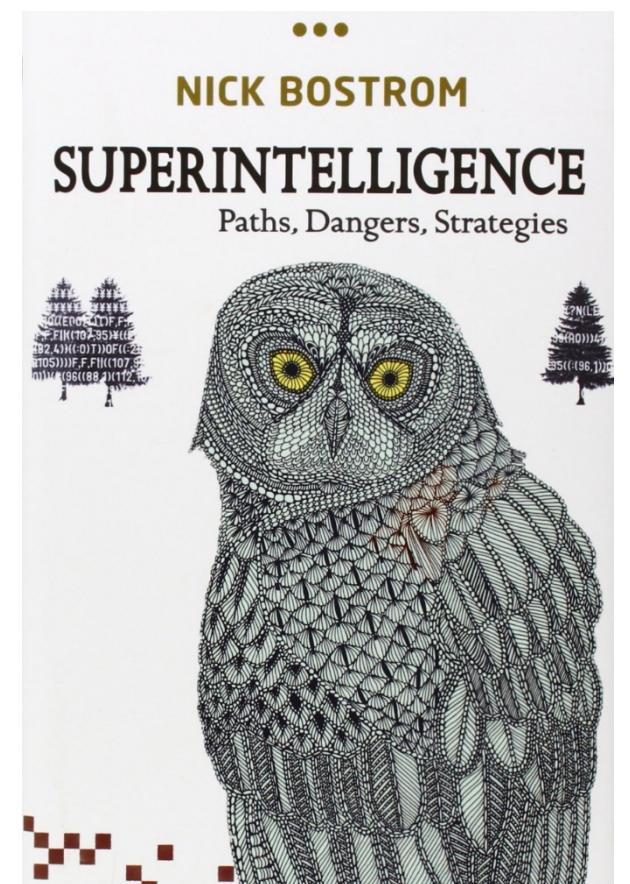
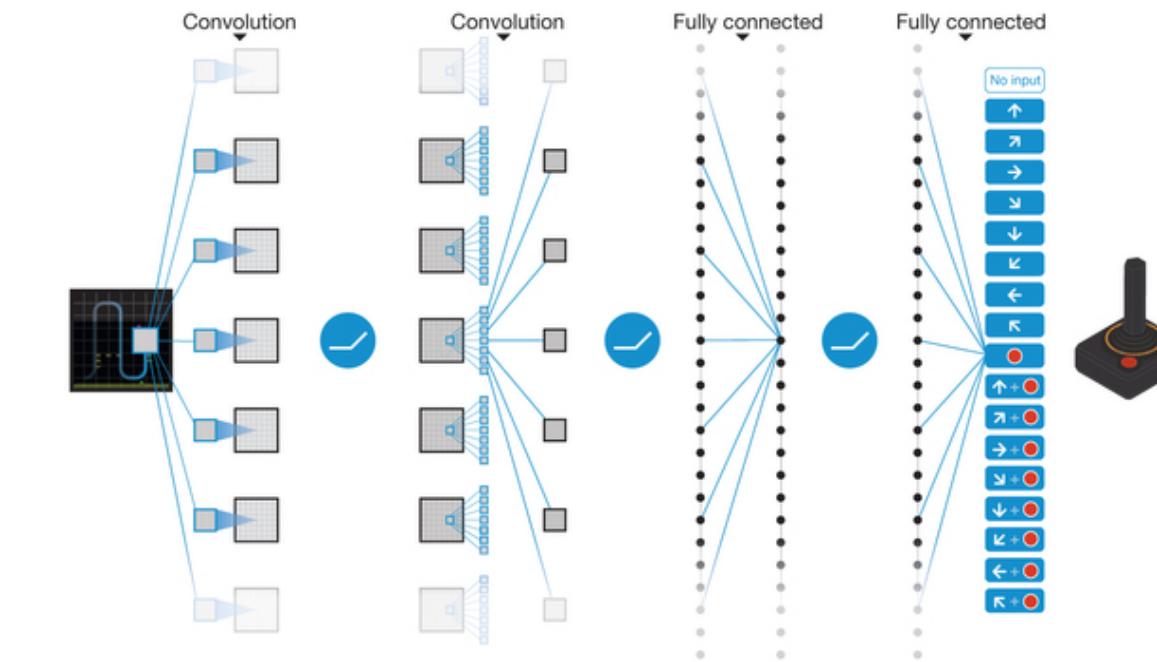
## Artificial Narrow Intelligence

- 정의된 목적을 달성하고 문제를 해결하기 위한 인공지능 (Trained/Programmed Intelligence)
- 스스로 문제를 정의하는 것이 아닌, 정의된 문제를 학습을 통해 해결하는 기계적 지능



## Artificial General Intelligence

- 기초 학습을 바탕으로 문제를 사고하고 해결할 수 있는 인공지능 (Educated Intelligence)
- 스스로 문제를 정의하고, 문제를 해결해내는 능력을 통해 지속적인 학습을 진행하는 인간과 유사한 지능



## Artificial Super Intelligence

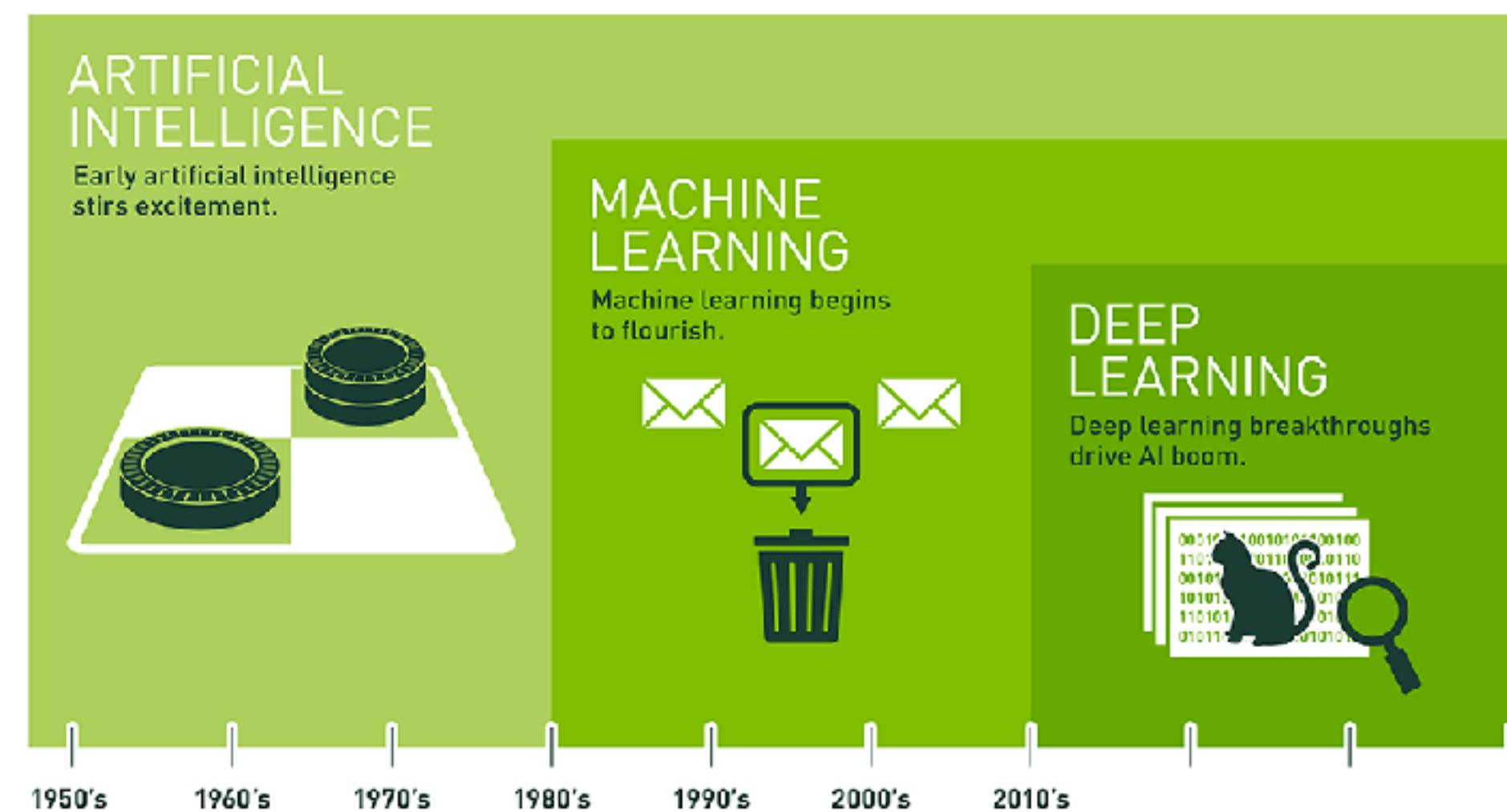
- 모든 분야에서 사람보다 뛰어난 능력을 보이는 인공지능
- 과학 기술의 창조, 일반적인 지식, 사회적 능력 등을 포함한 모든 영역에서 인류의 두뇌보다 총명한 지능 (Nick Bostrom)

# Machine Learning

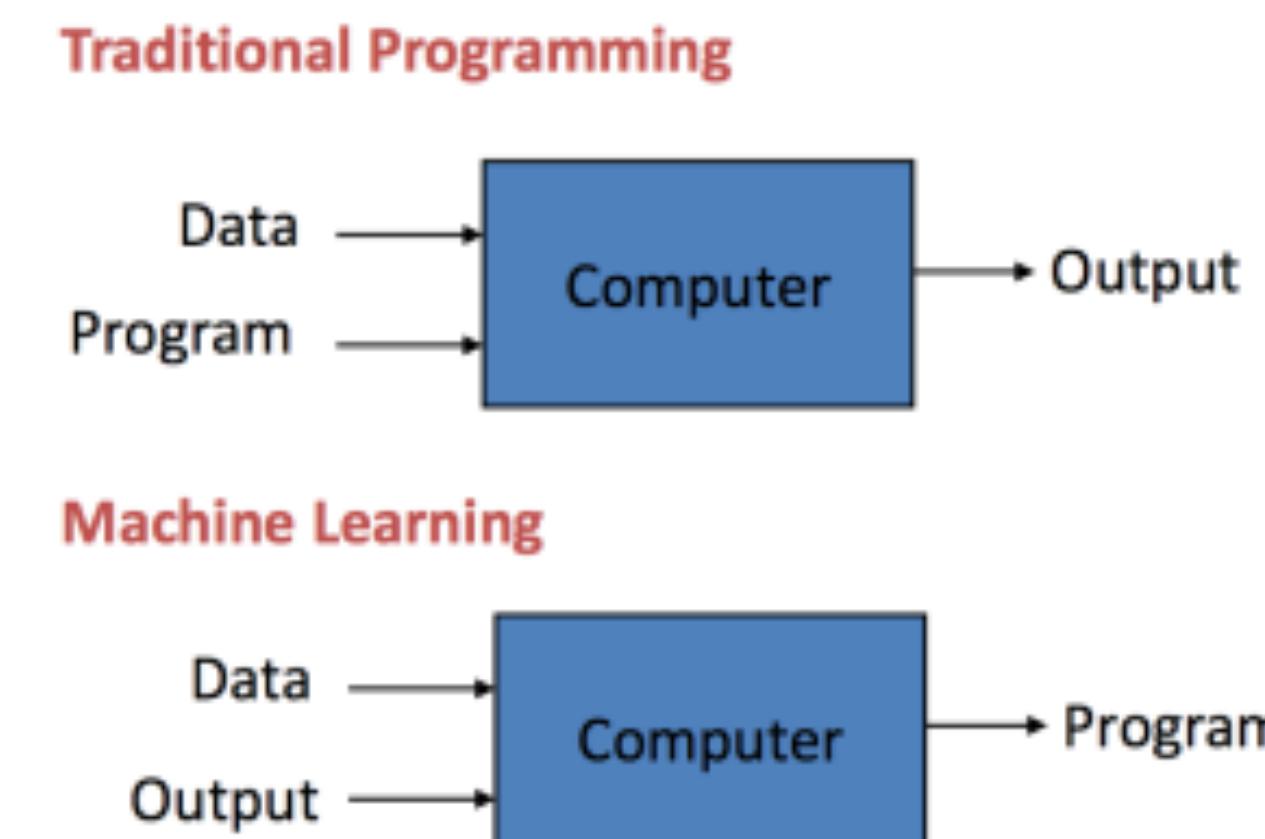
## □ Machine Learning (기계학습)

- 인공 지능의 한 분야. 컴퓨터가 학습할 수 있게 하는 알고리즘 분야 (Wikipedia)
- Field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel, 1959)
- A computer program is said to **learn from experience E** with respect to **some task T** and some **performance measure P**, if its performance on T, as measured by P, improves with experience E. (Tom Mitchell, 1998)

## <AI & Machine Learning 차이>



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

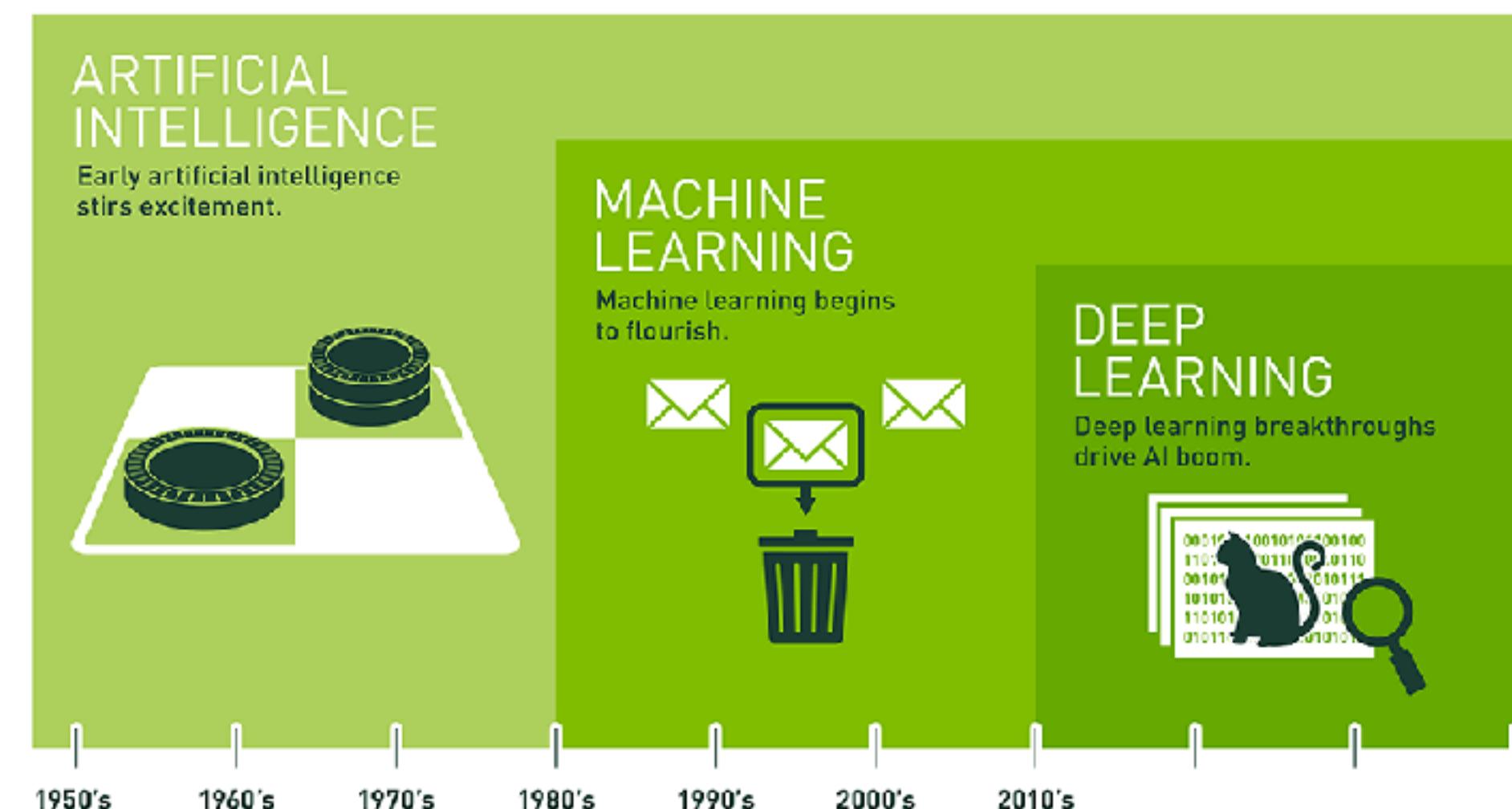


# Machine Learning

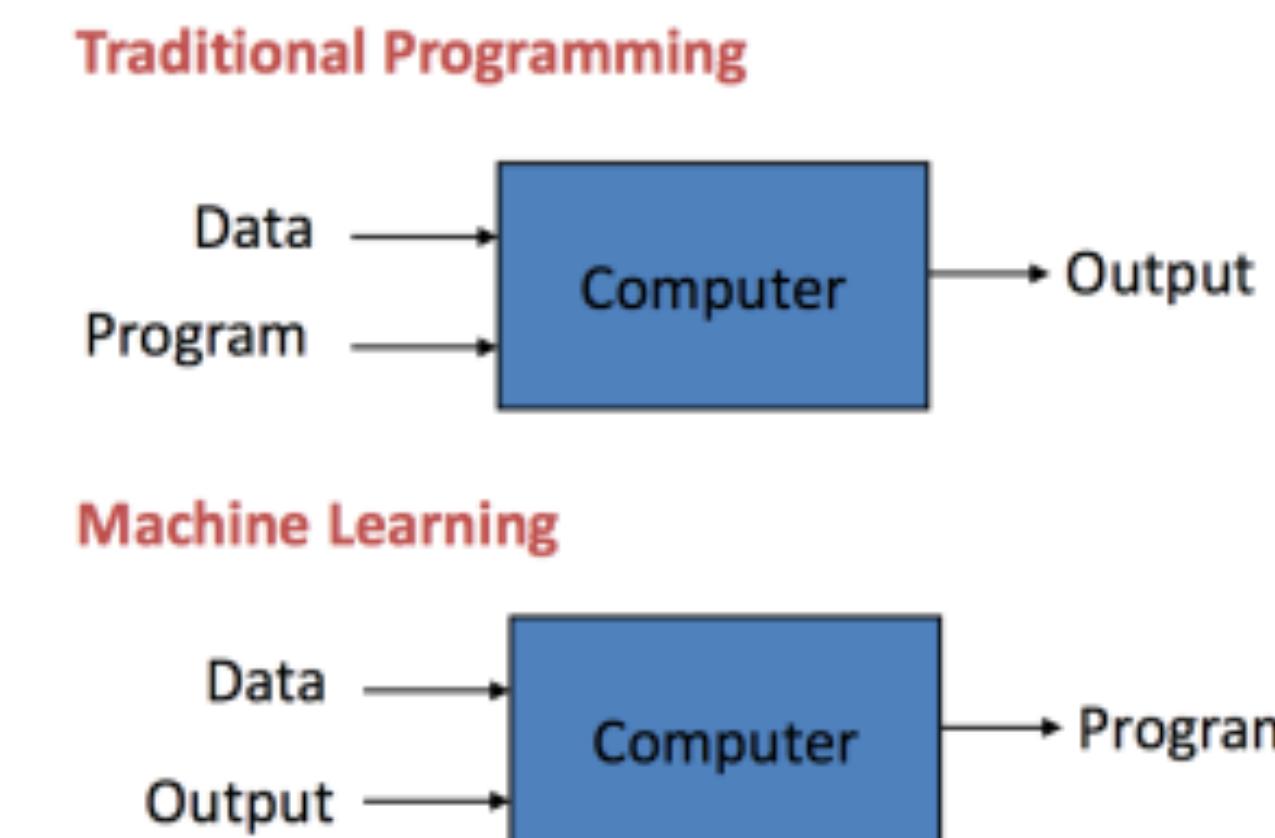
## □ Machine Learning (기계학습)

- 인공 지능의 한 분야. 컴퓨터가 학습할 수 있게 하는 알고리즘 분야 (Wikipedia)
- Field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel, 1959)
- A computer program is said to **learn from experience E** with respect to **some task T** and some **performance measure P**, if its performance on T, as measured by P, improves with experience E. (Tom Mitchell, 1998)

## <AI & Machine Learning 차이>



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

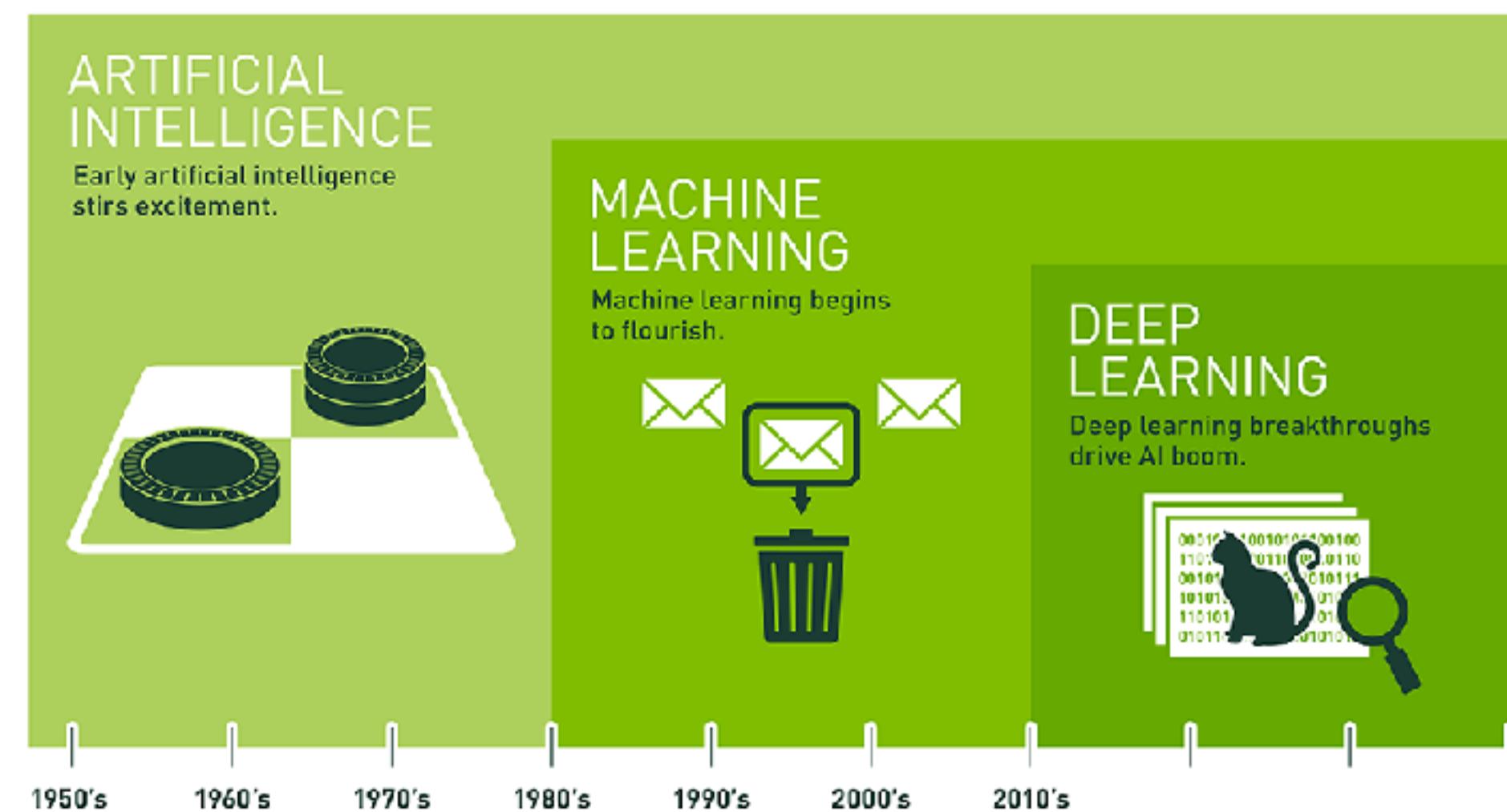


# Machine Learning

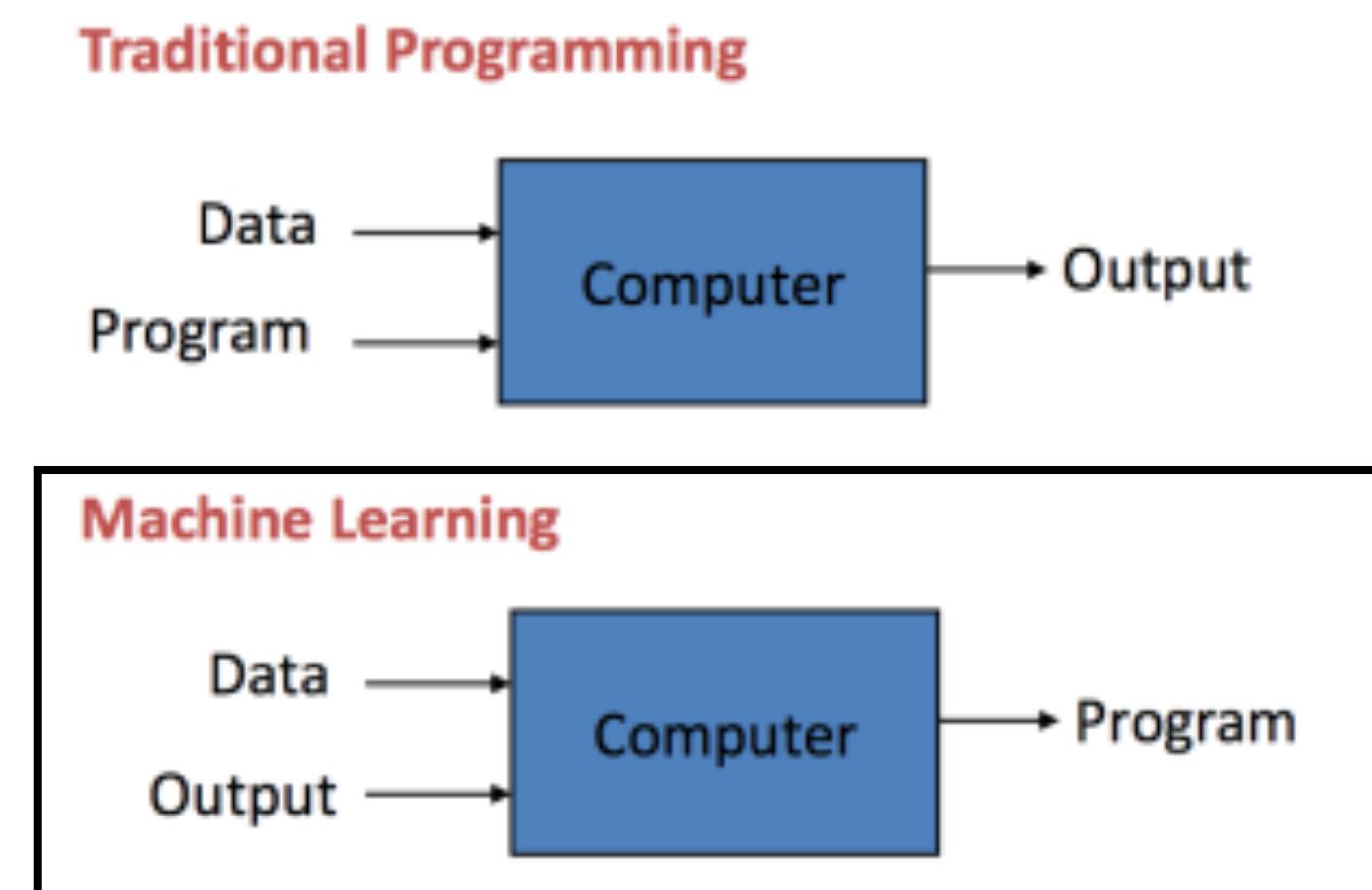
## □ Machine Learning (기계학습)

- 인공 지능의 한 분야. 컴퓨터가 학습할 수 있게 하는 알고리즘 분야 (Wikipedia)
- Field of study that gives computers the ability to learn without being explicitly programmed. (Arthur Samuel, 1959)
- A computer program is said to **learn from experience E** with respect to **some task T** and some **performance measure P**, if its performance on T, as measured by P, improves with experience E. (Tom Mitchell, 1998)

### <AI & Machine Learning 차이>



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.



조금 더 수학적으로 설명하면...

Machine Learning이 하는 작업은  
Input Data에 따른 Output을 잘 설명하는  
파라미터를 조정하는 것

**exem**

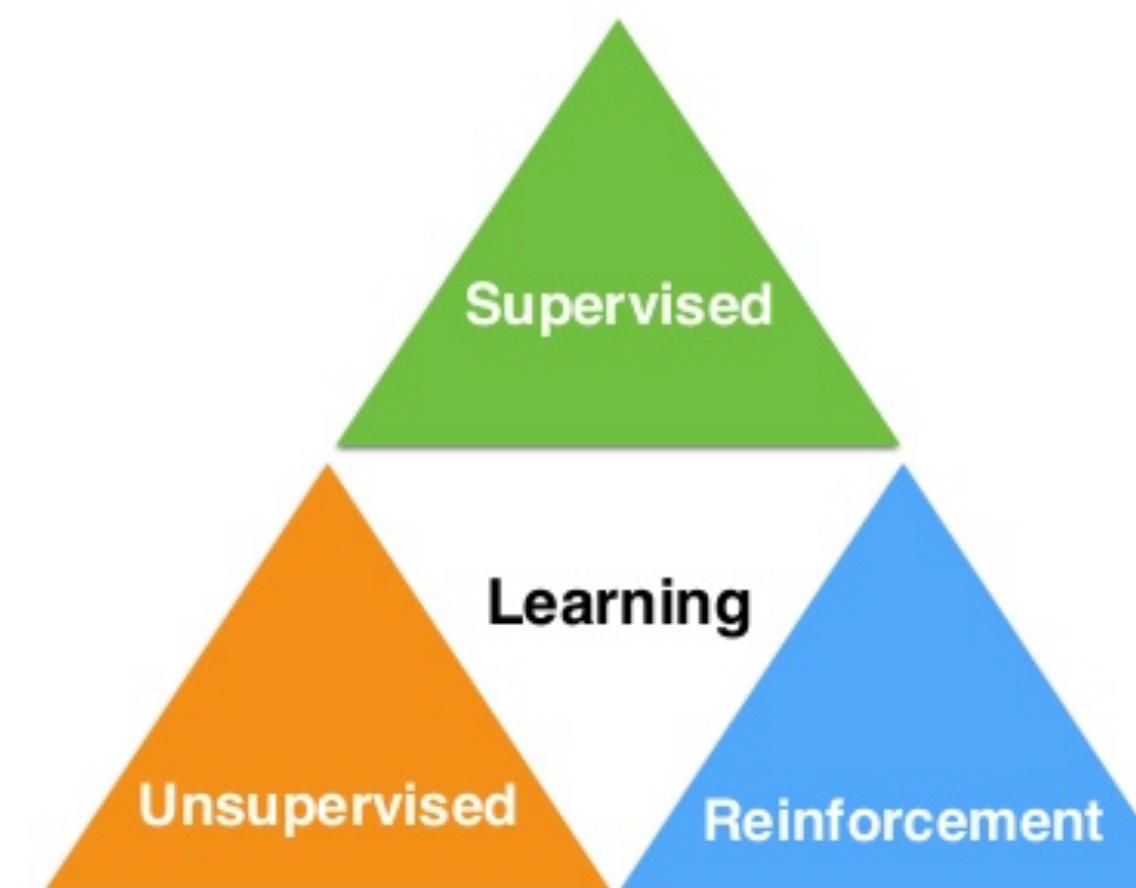
# Types of Machine Learning

- ❑ 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



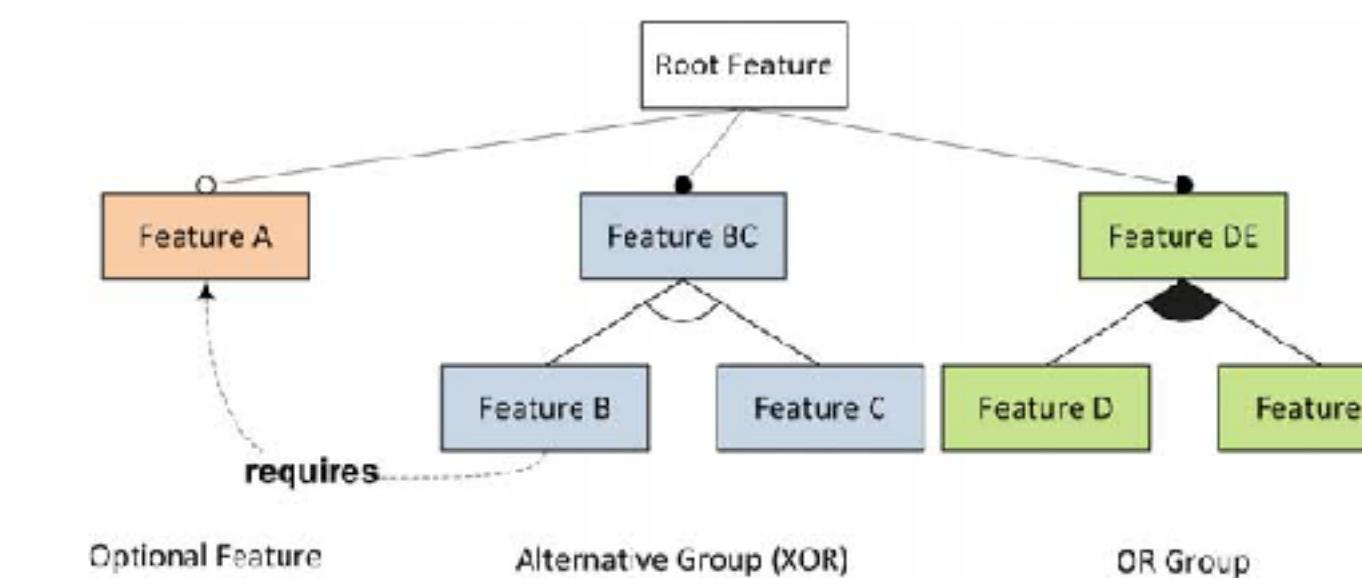
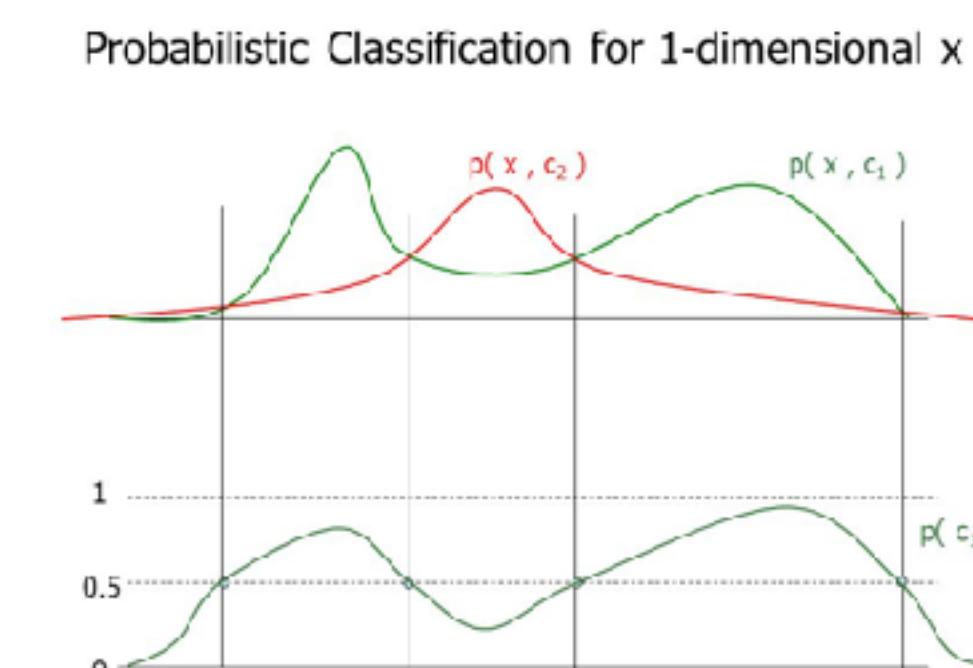
- Labeled data
- Direct feedback
- Predict outcome/future



- No labels
- No feedback
- "Find hidden structure"

- Decision process
- Reward system
- Learn series of actions

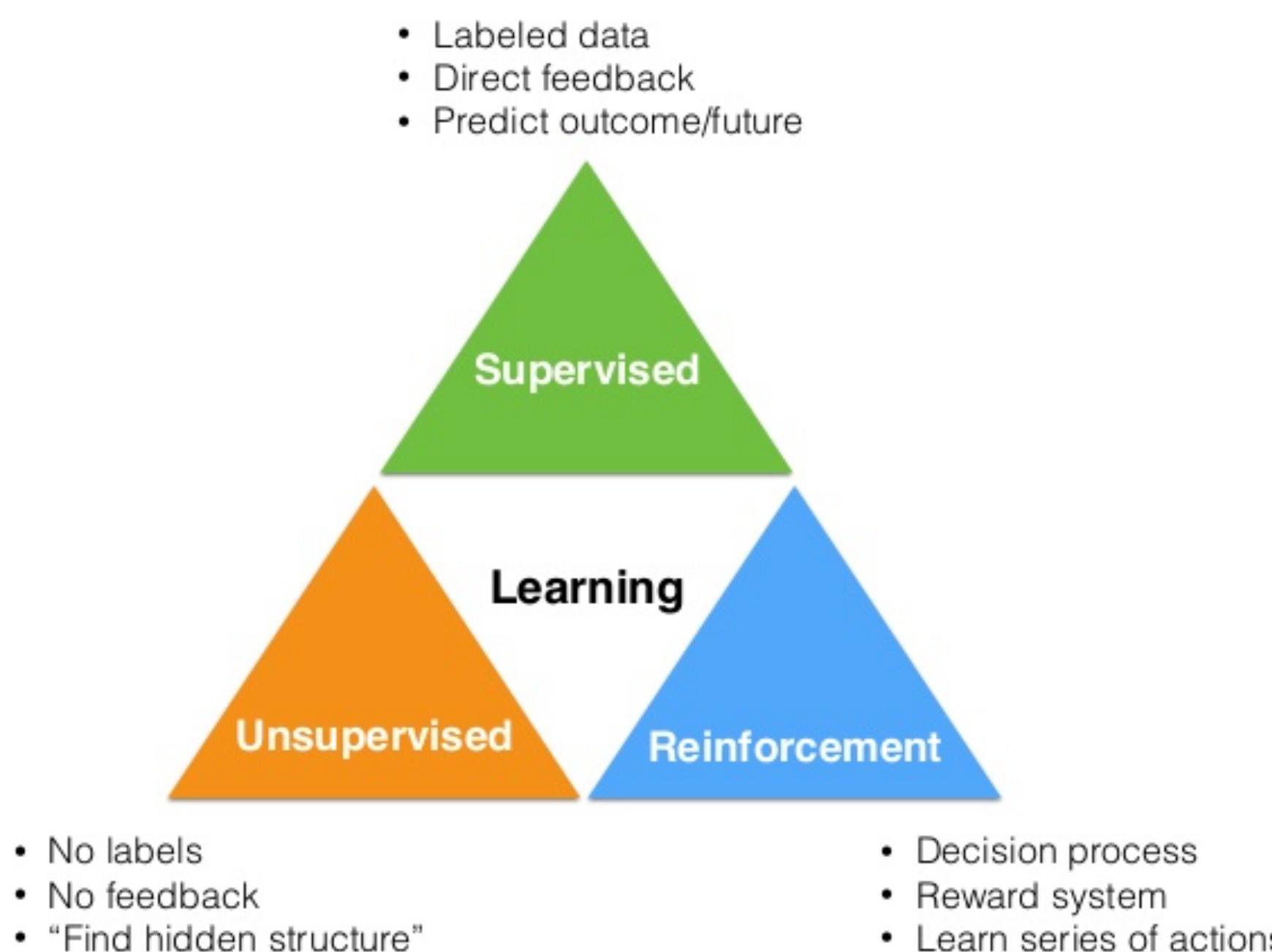
모델 특성에 따른 분류



# 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



## <Supervised Learning>

- 특정 Input에 대한 Output을 예측하기 위해 학습하는 것
- **Labeled Data**를 다룸 -> 정답 존재
- Target Output에 따라 2 타입으로 나뉨
  - Regression: real number or vector
  - Classification: class label

$$\underline{y} = f(\underline{x}; \underline{W})$$

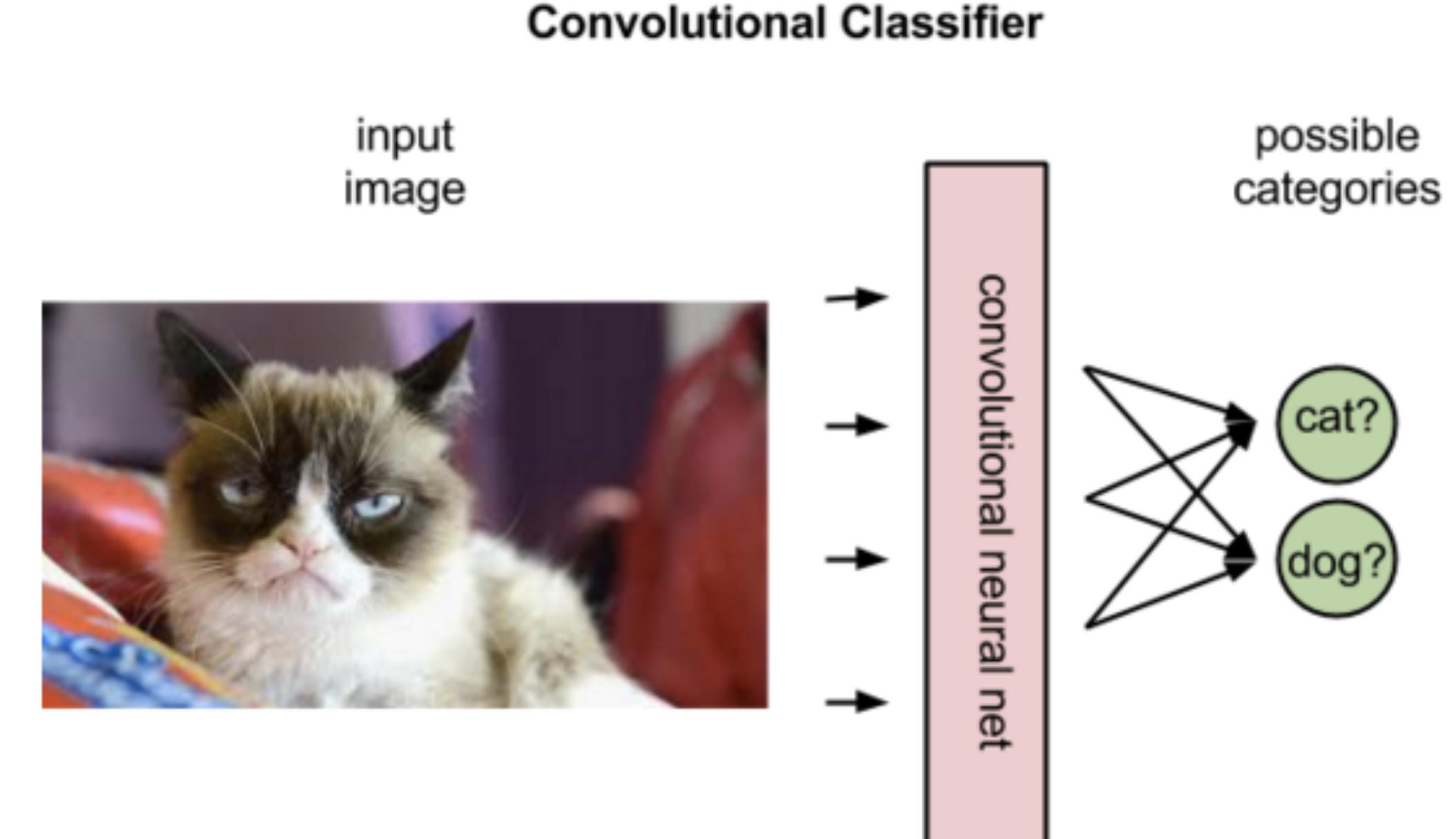
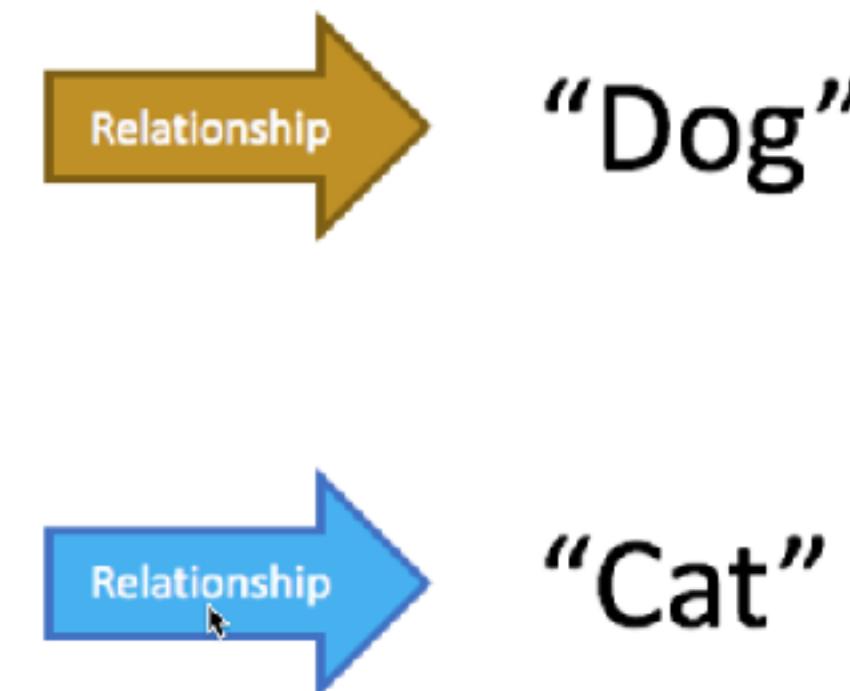
target output      input      learned parameter

<https://www.youtube.com/watch?v=3jCaGDIY6VM>

# 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

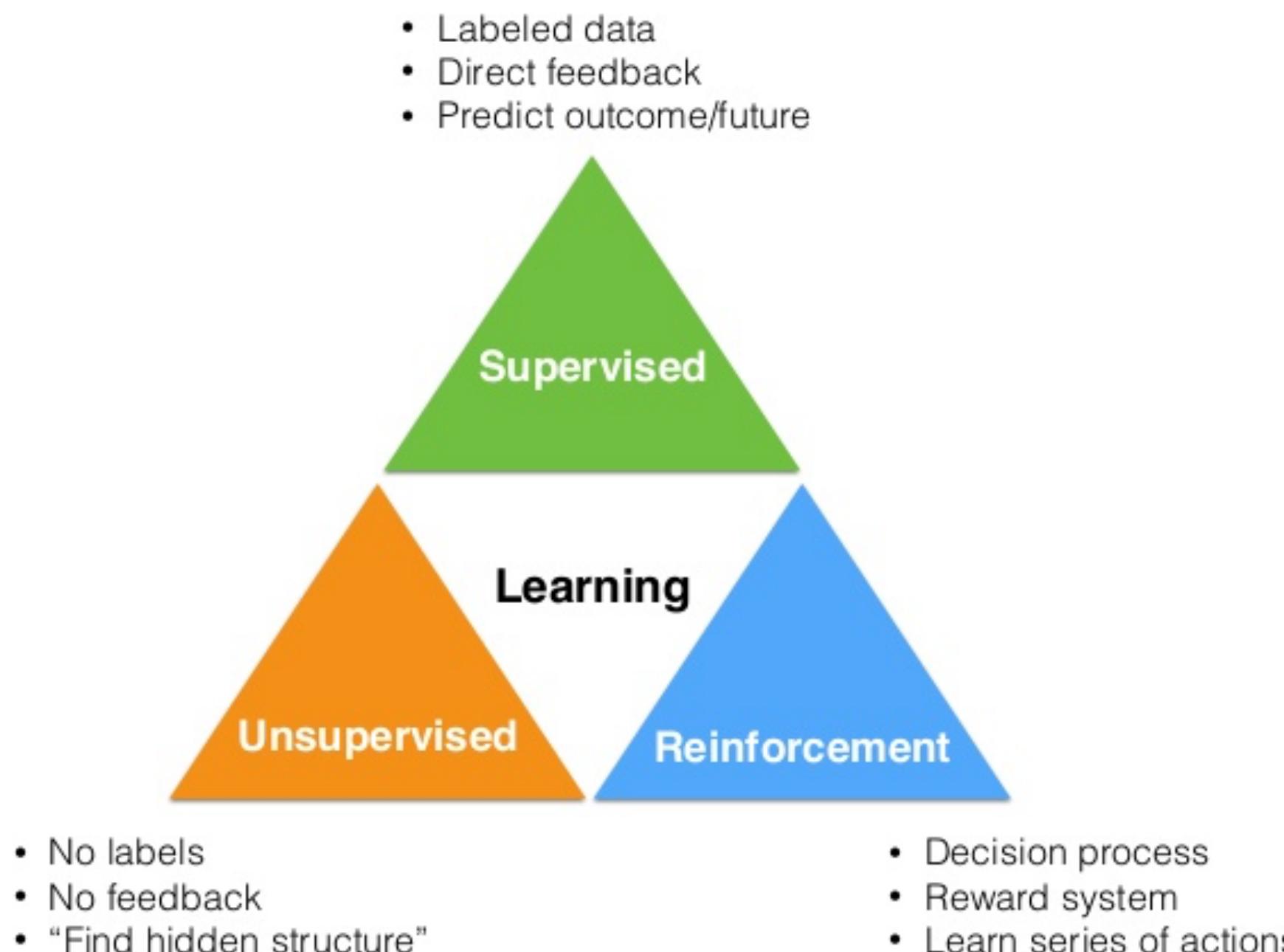
## <Labeled Data Example>



# Types of Machine Learning

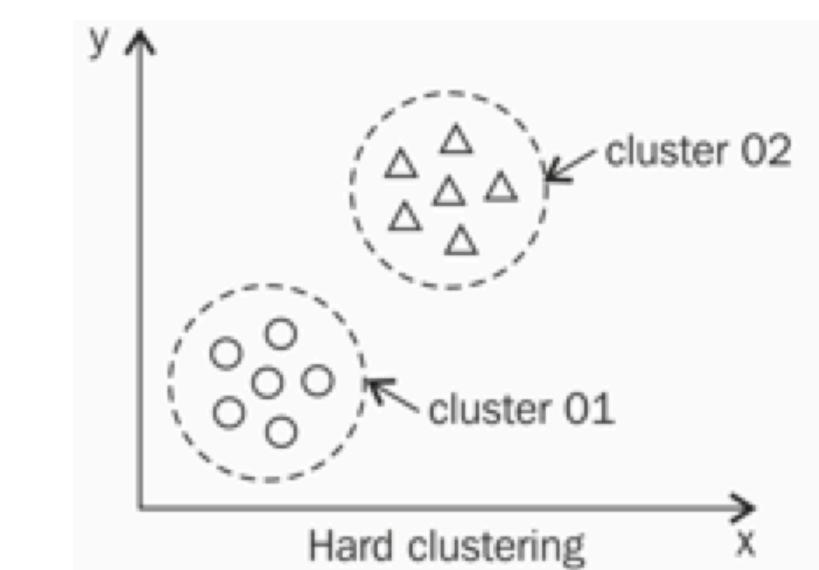
- ❑ 머신러닝은 다양한 기준으로 세부 분류가 가능함
- ❑ 학습 특성에 따른 분류: 1) Supervised, 2) Unsupervised, 3) Reinforcement Learnings
- ❑ 모델 특성에 따른 분류: 1) Geometric, 2) Probabilistic, 3) Logical Models

## 학습 특성에 따른 분류



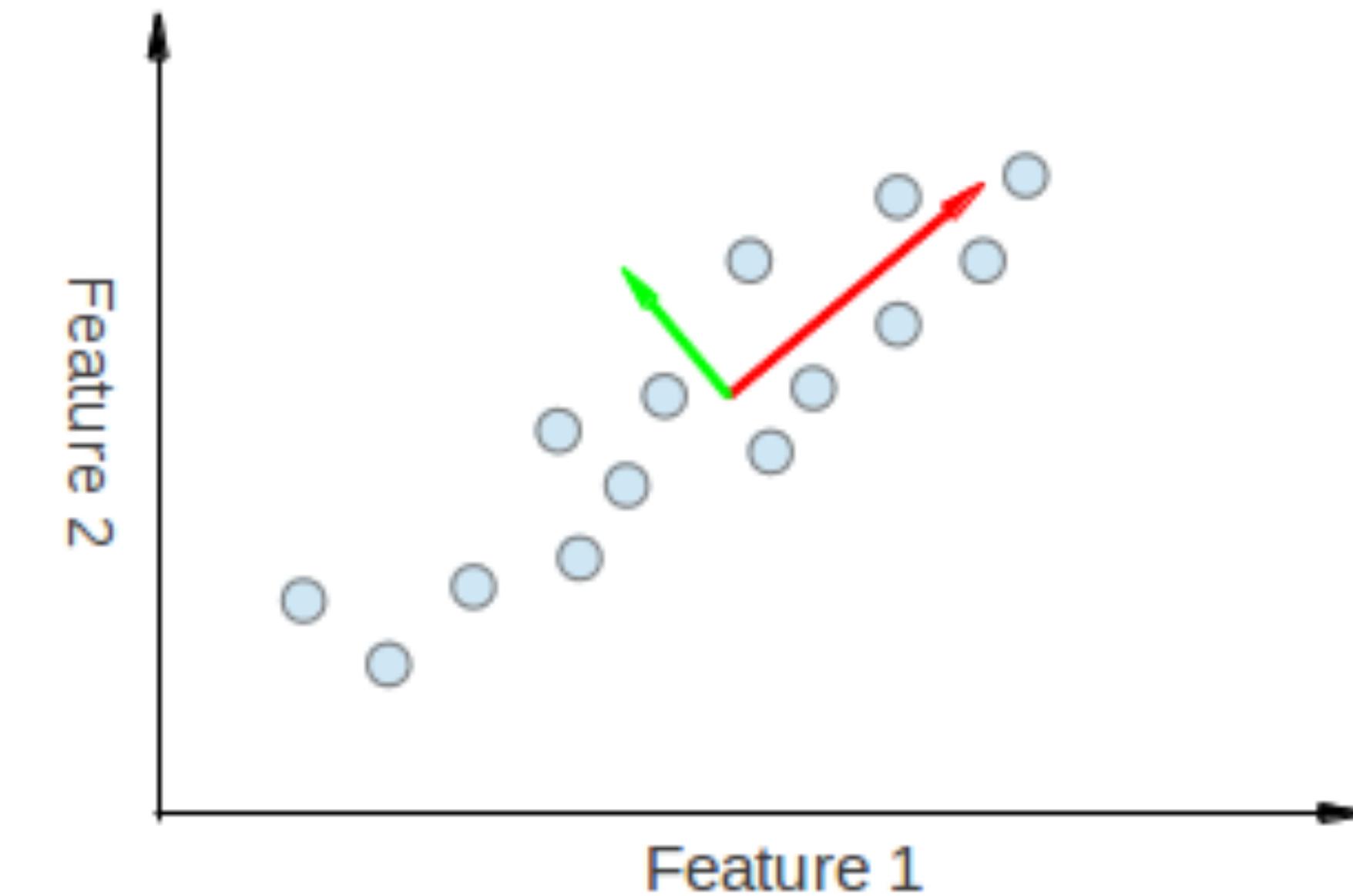
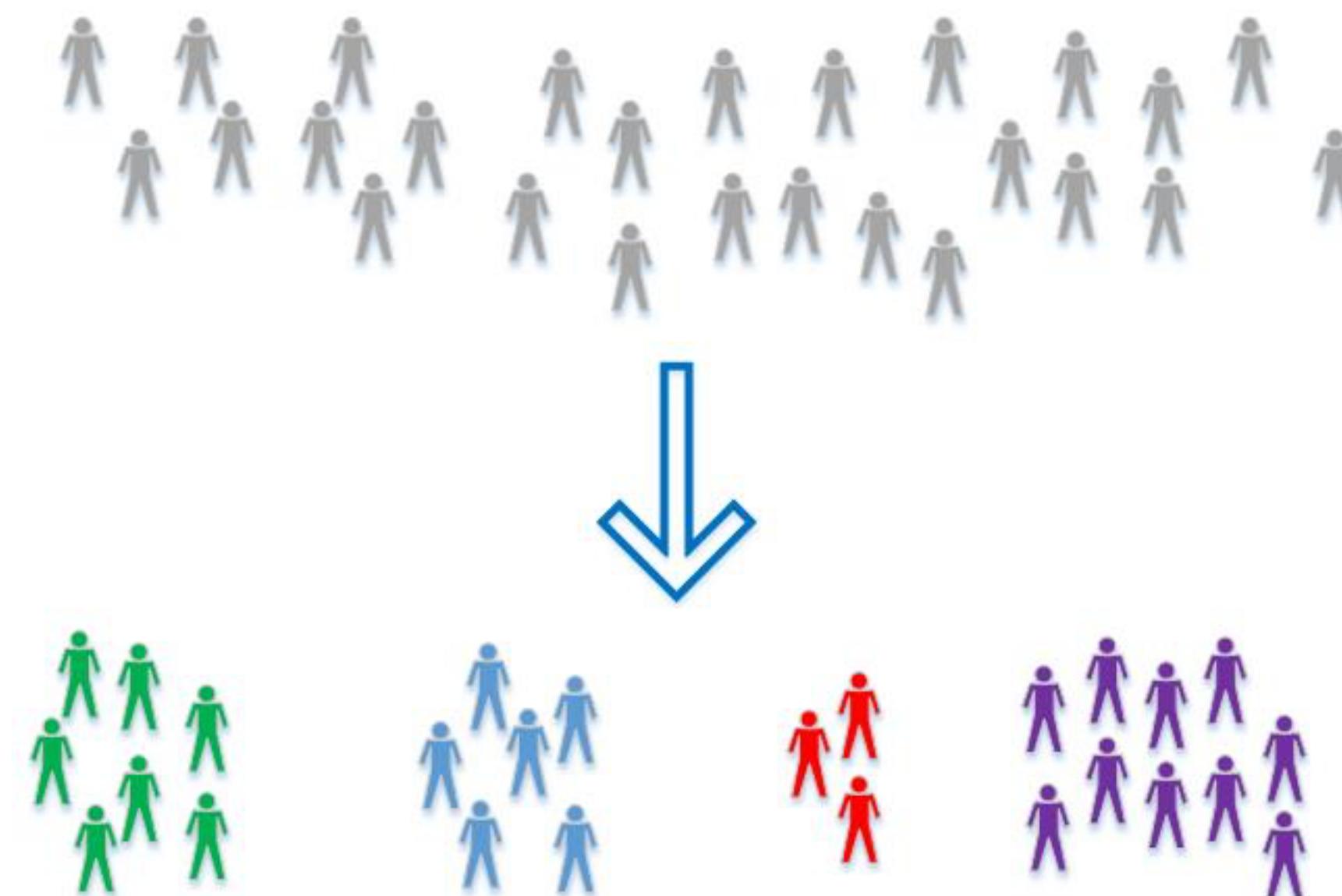
## <Unsupervised Learning>

- Input 데이터를 잘 설명하는 내부적 표현을 발견하는 것
- **Unlabeled Data**를 다룸 → 정답이 없음
- Unsupervised Learning의 범위에 대한 논란이 많았으나,  
현재에는 다양한 목적으로 사용함
  - Reduction of Dimensionality
  - Finding meaningful representation
  - Clustering Algorithm



# Types of Machine Learning

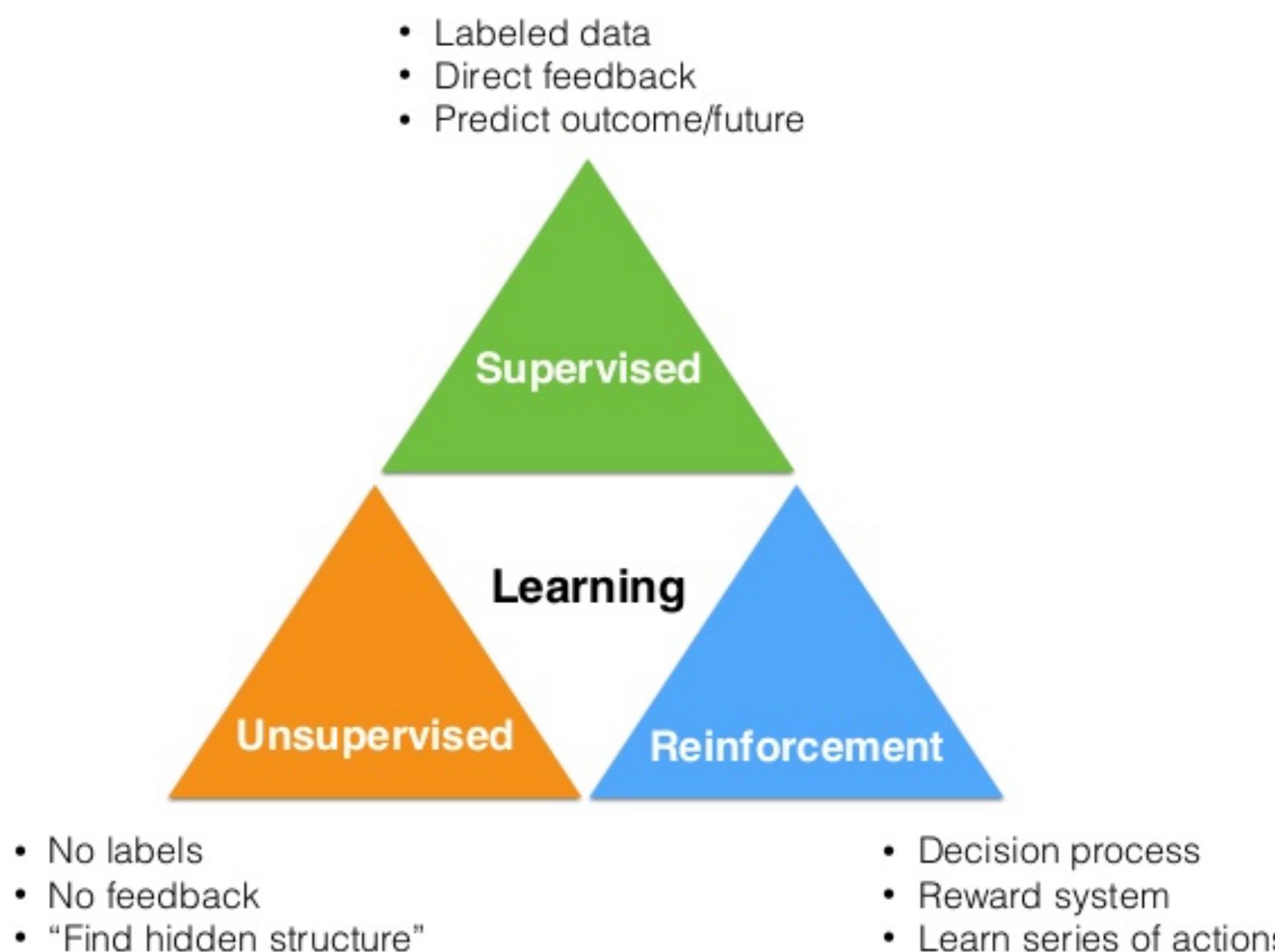
- ❑ 머신러닝은 다양한 기준으로 세부 분류가 가능함
- ❑ 학습 특성에 따른 분류: 1) Supervised, 2) Unsupervised, 3) Reinforcement Learnings
- ❑ 모델 특성에 따른 분류: 1) Geometric, 2) Probabilistic, 3) Logical Models



# Types of Machine Learning

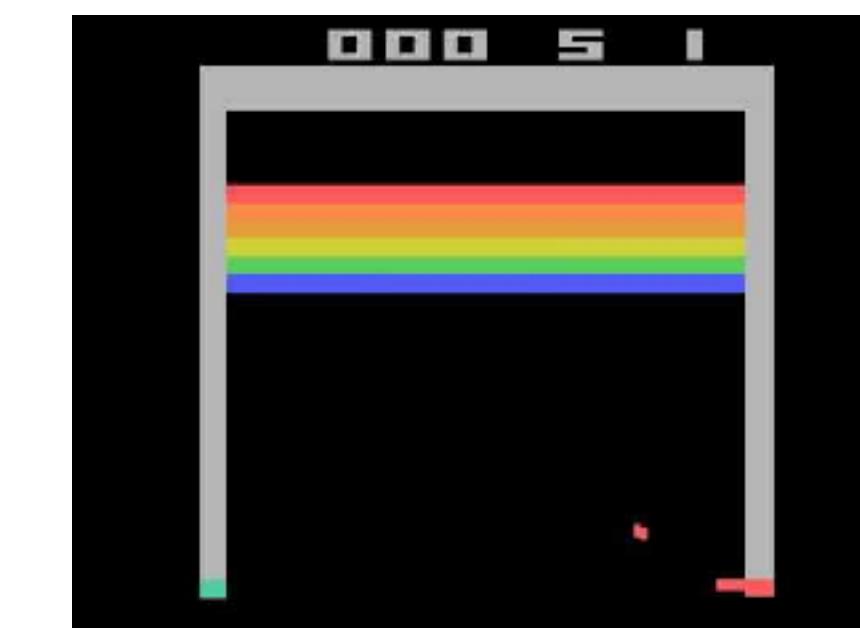
- ❑ 머신러닝은 다양한 기준으로 세부 분류가 가능함
- ❑ 학습 특성에 따른 분류: 1) Supervised, 2) Unsupervised, 3) Reinforcement Learnings
- ❑ 모델 특성에 따른 분류: 1) Geometric, 2) Probabilistic, 3) Logical Models

## 학습 특성에 따른 분류



## <Reinforcement Learning>

- 학습 주체 Agent, 주위 상황 Environment, 학습 기준 Value로 구성
- 미래의 기대 보상을 최대화하기 위한 Agent의 행동 방침(policy)를 주위 환경의 상호작용을 바탕으로 학습



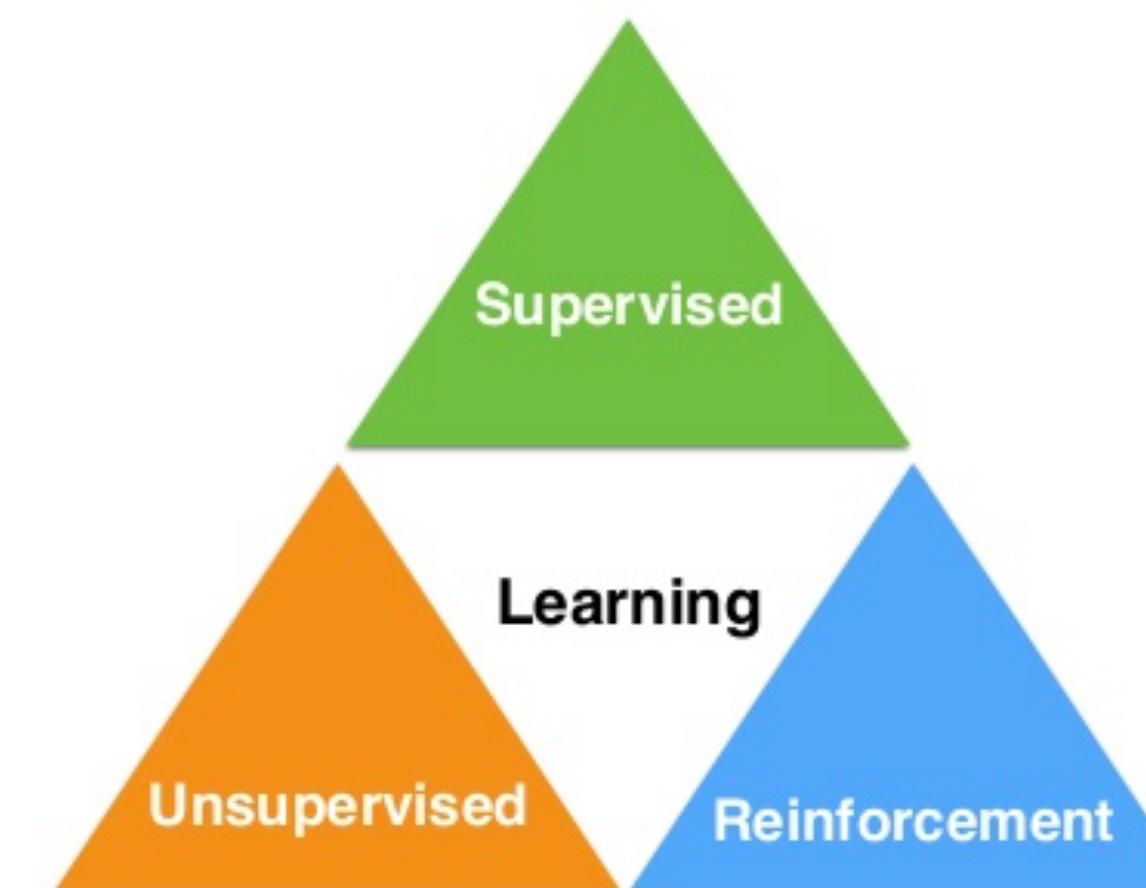
# Types of Machine Learning

- ❑ 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

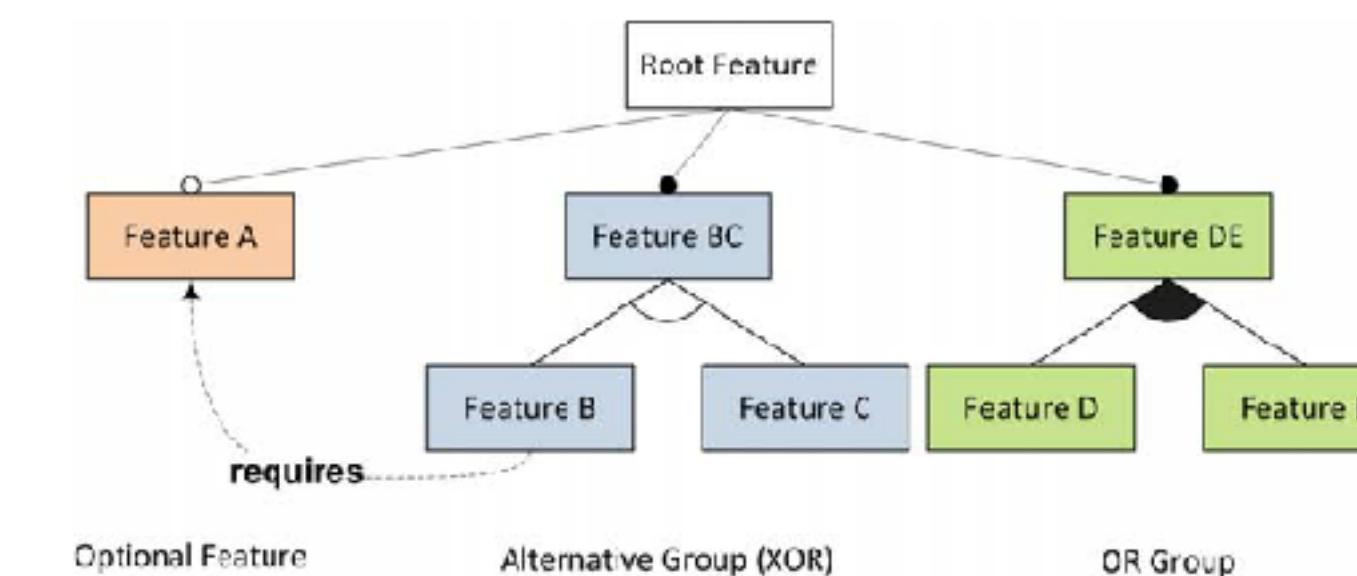
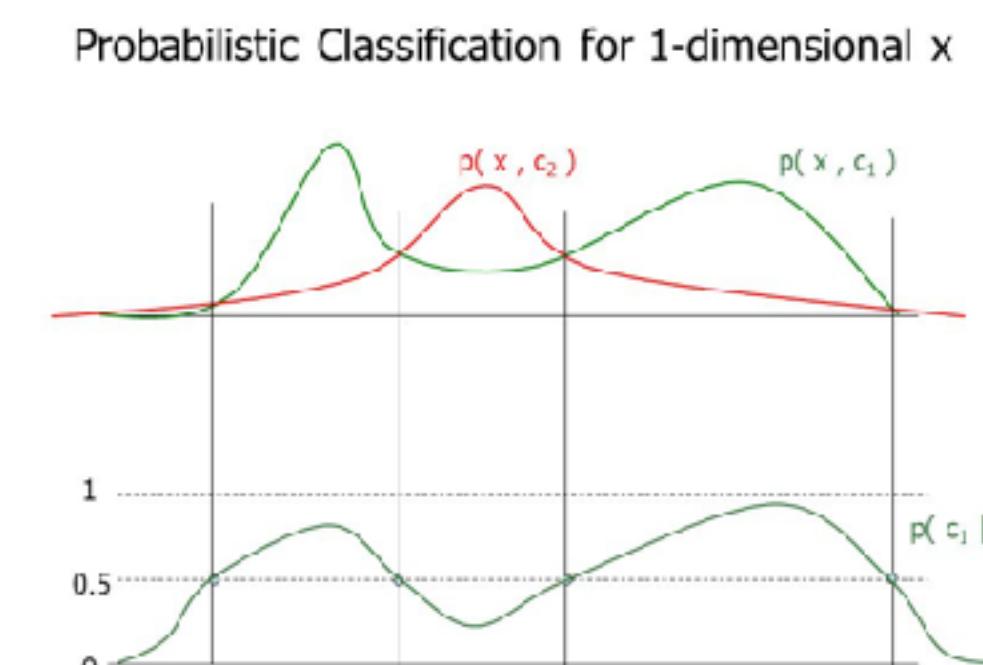


- Labeled data
- Direct feedback
- Predict outcome/future



- No labels
  - No feedback
  - "Find hidden structure"
- Decision process
  - Reward system
  - Learn series of actions

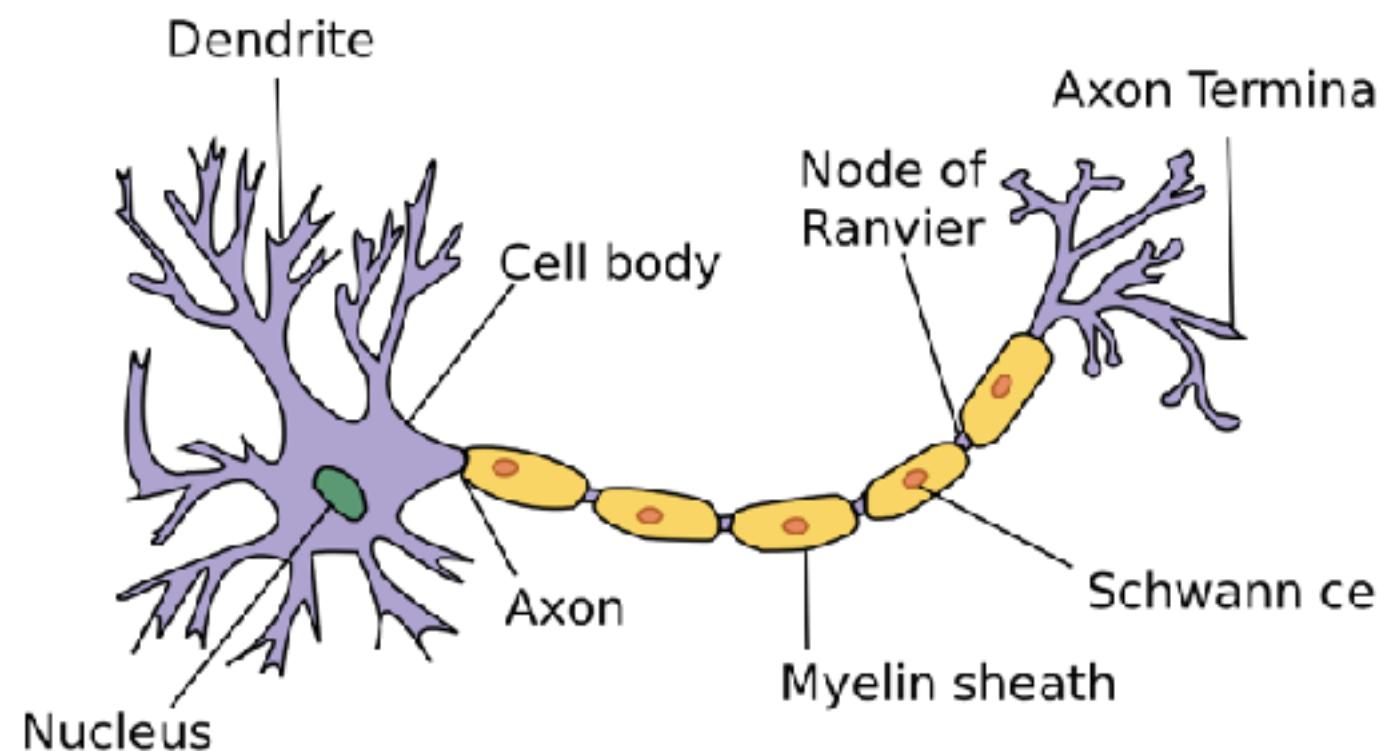
모델 특성에 따른 분류



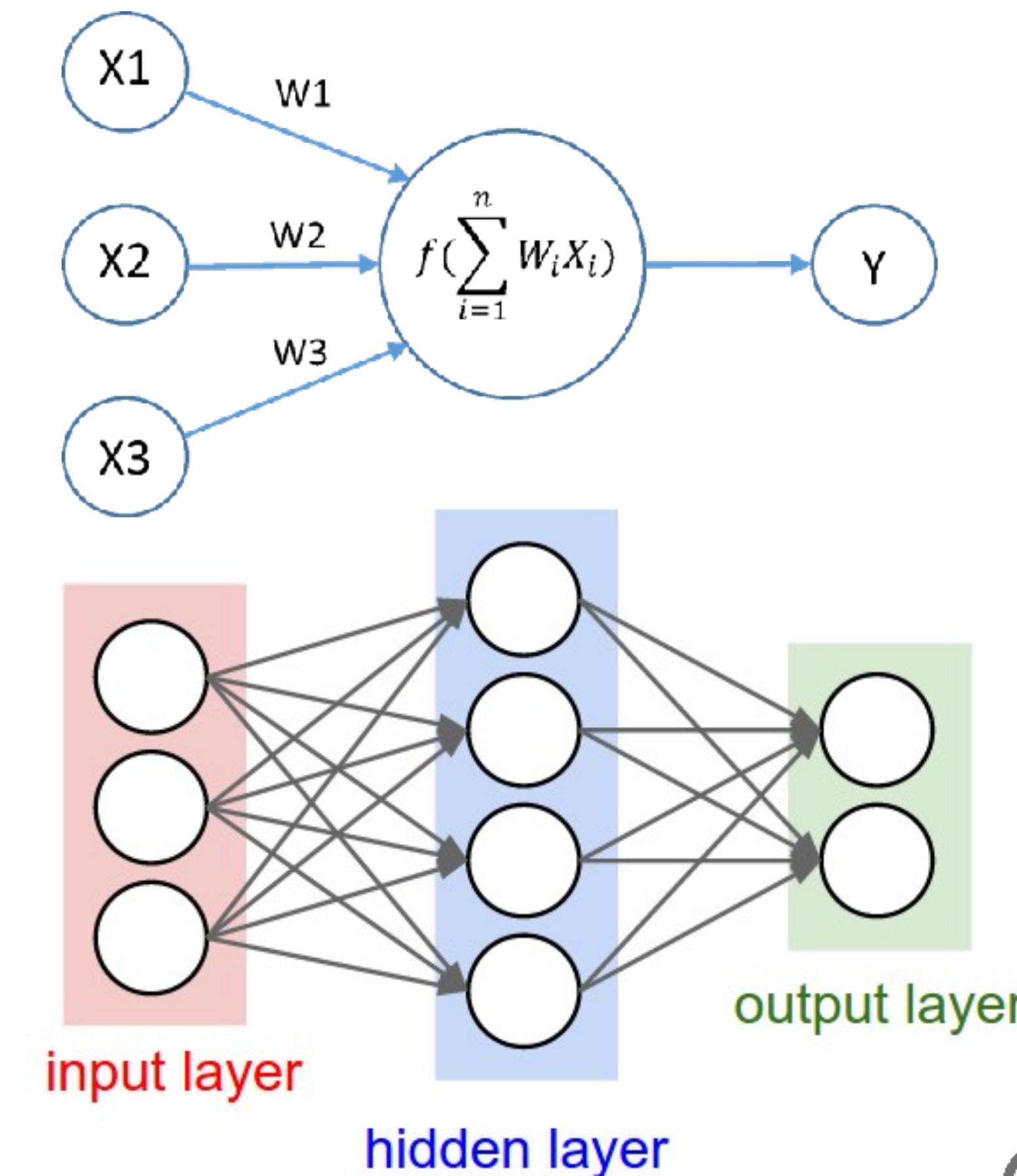
# Neuron and Neural Network

- 인간의 뇌는 뉴런을 단위로 뉴런의 연결을 통해 복잡한 계산을 효과적으로 수행함

## <Typical Cortical Neuron >



## <Overview of Neural Network>

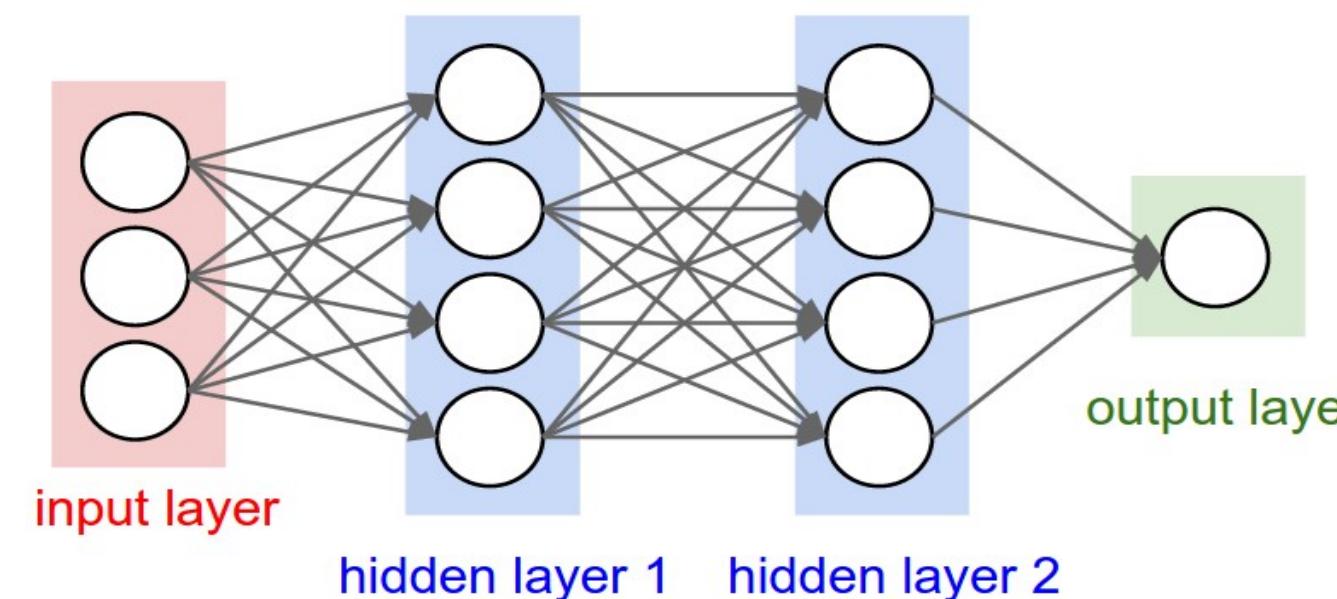
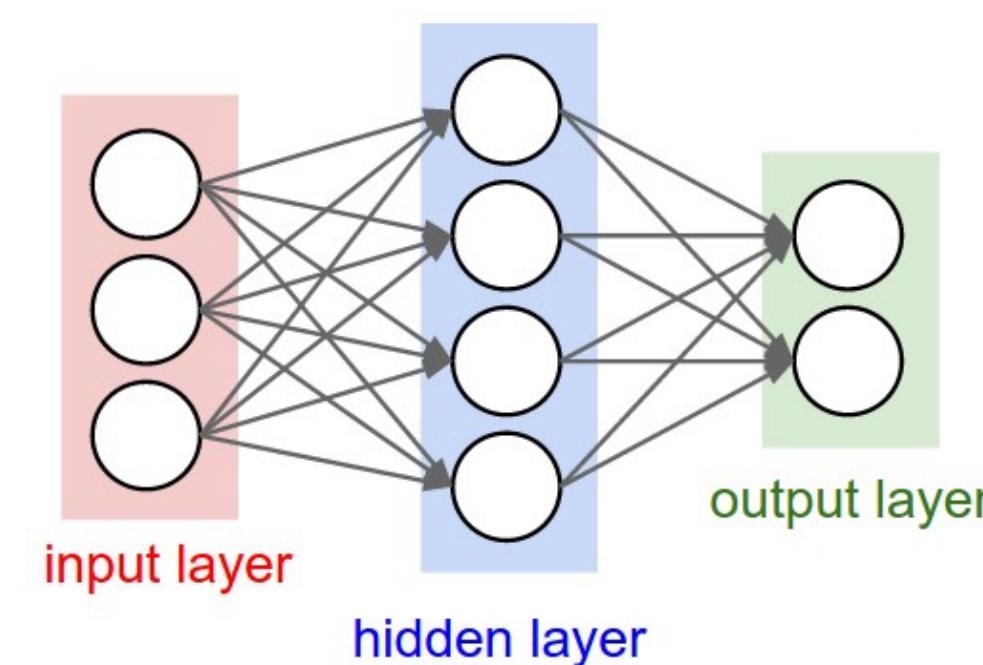


# Deep Learning

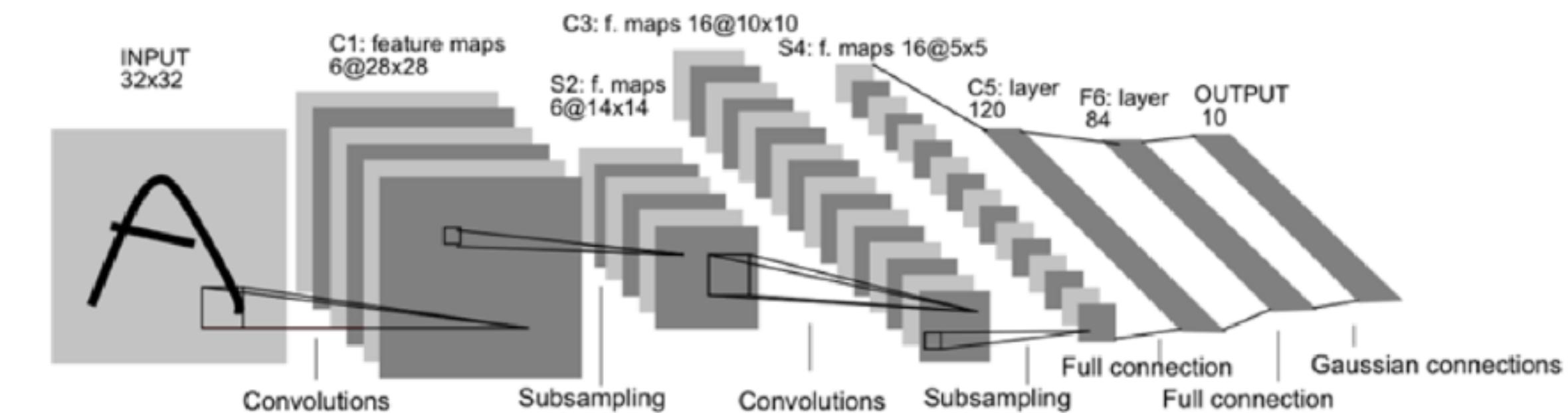
## □ Deep Learning

- “Deep” Neural Network를 활용하여 학습하는 머신러닝 알고리즘을 의미
- Deep : Hidden Layer의 수가 **2개 이상**일 경우

<DNN 예시(Below)>



<Convolutional Neural Network LeNet5>



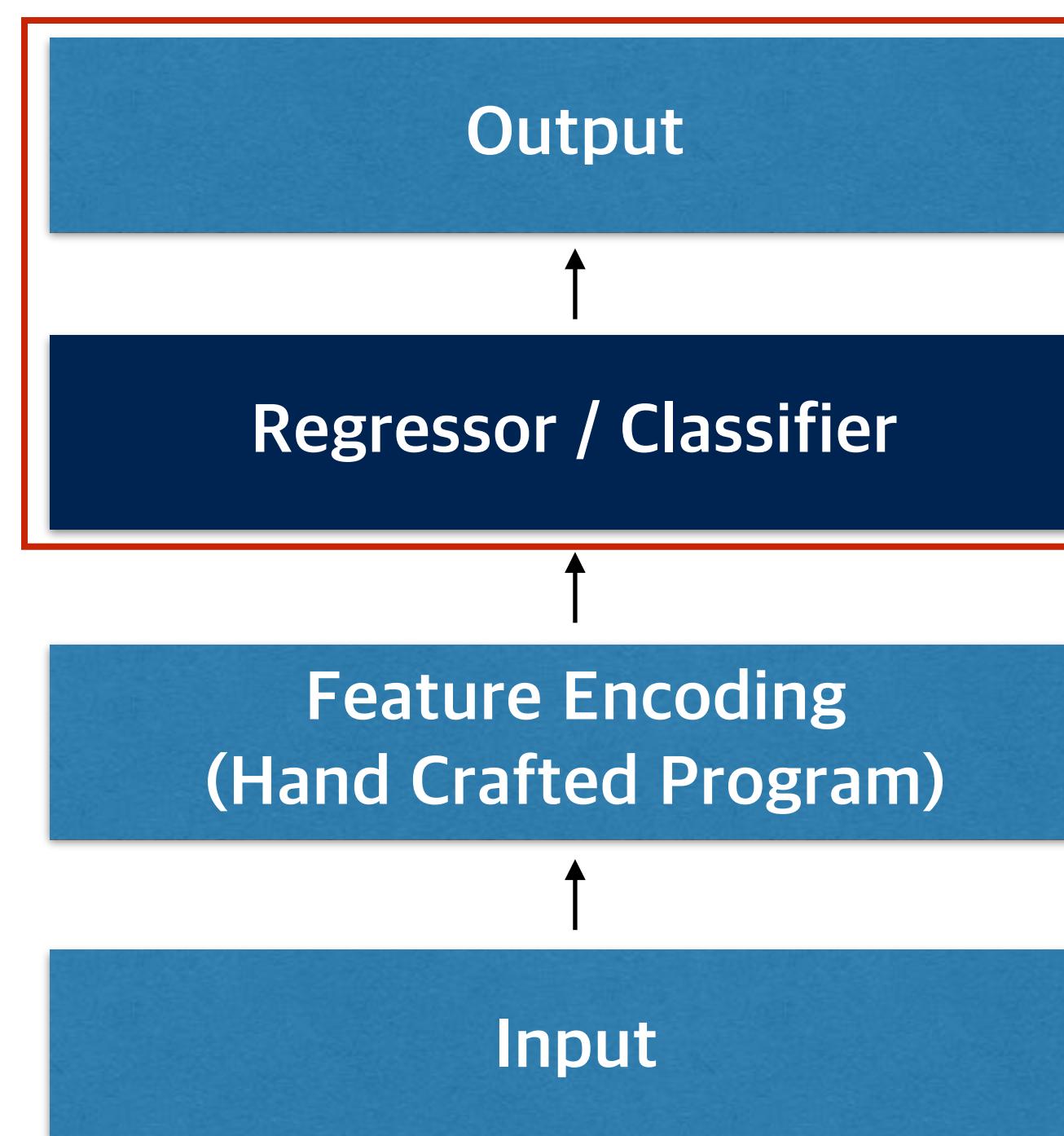
<Object Detection>



# Traditional ML vs. Deep Learning

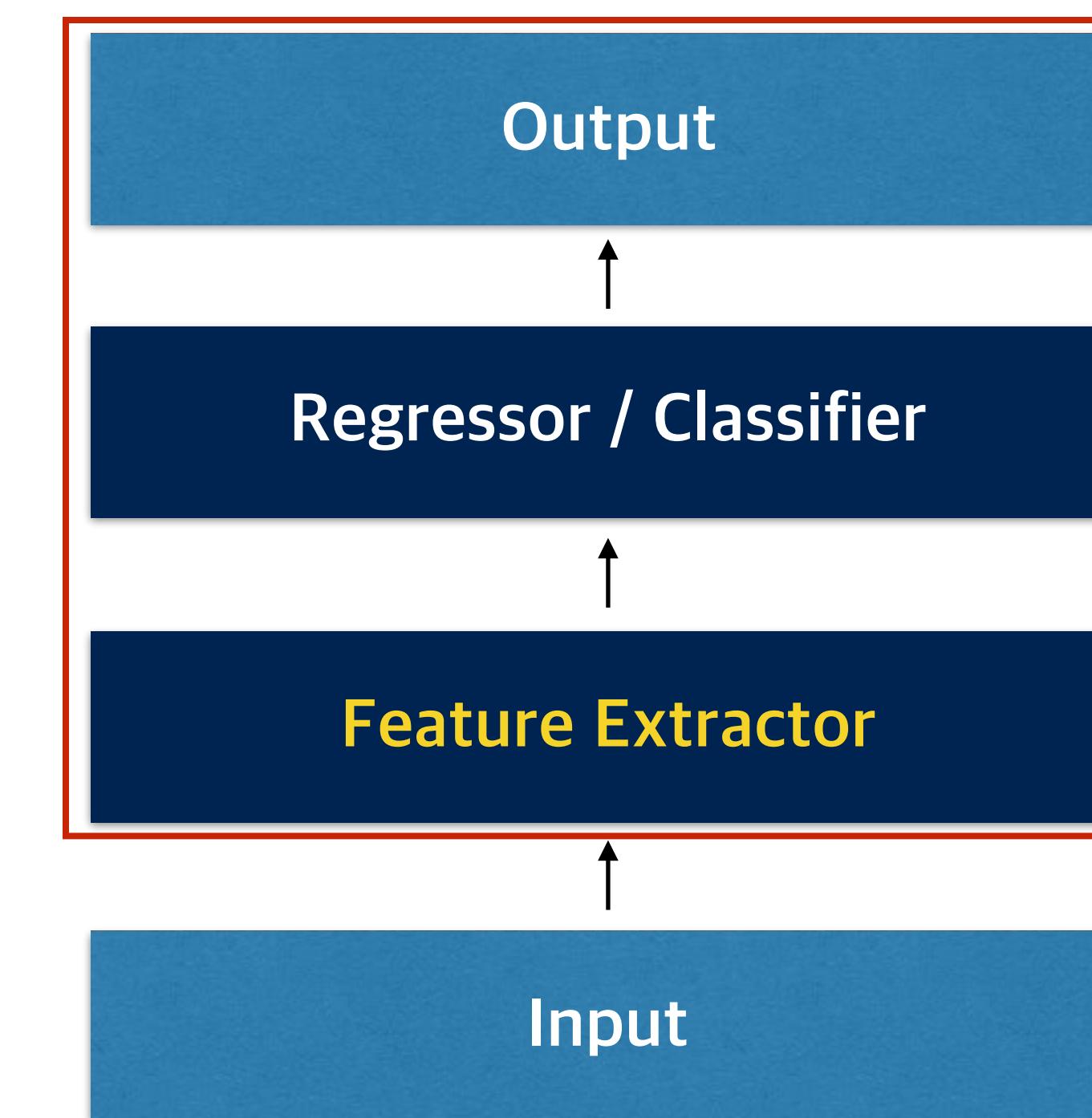
- ❑ 전통적인 머신러닝은 기본적으로 직접 Feature Encoding 후, 이를 바탕으로 학습을 진행함
- ❑ 딥러닝의 경우, 모델 내부에 **Feature Extractor**가 포함되며 함께 학습을 진행

<Traditional Model>



Learning Part

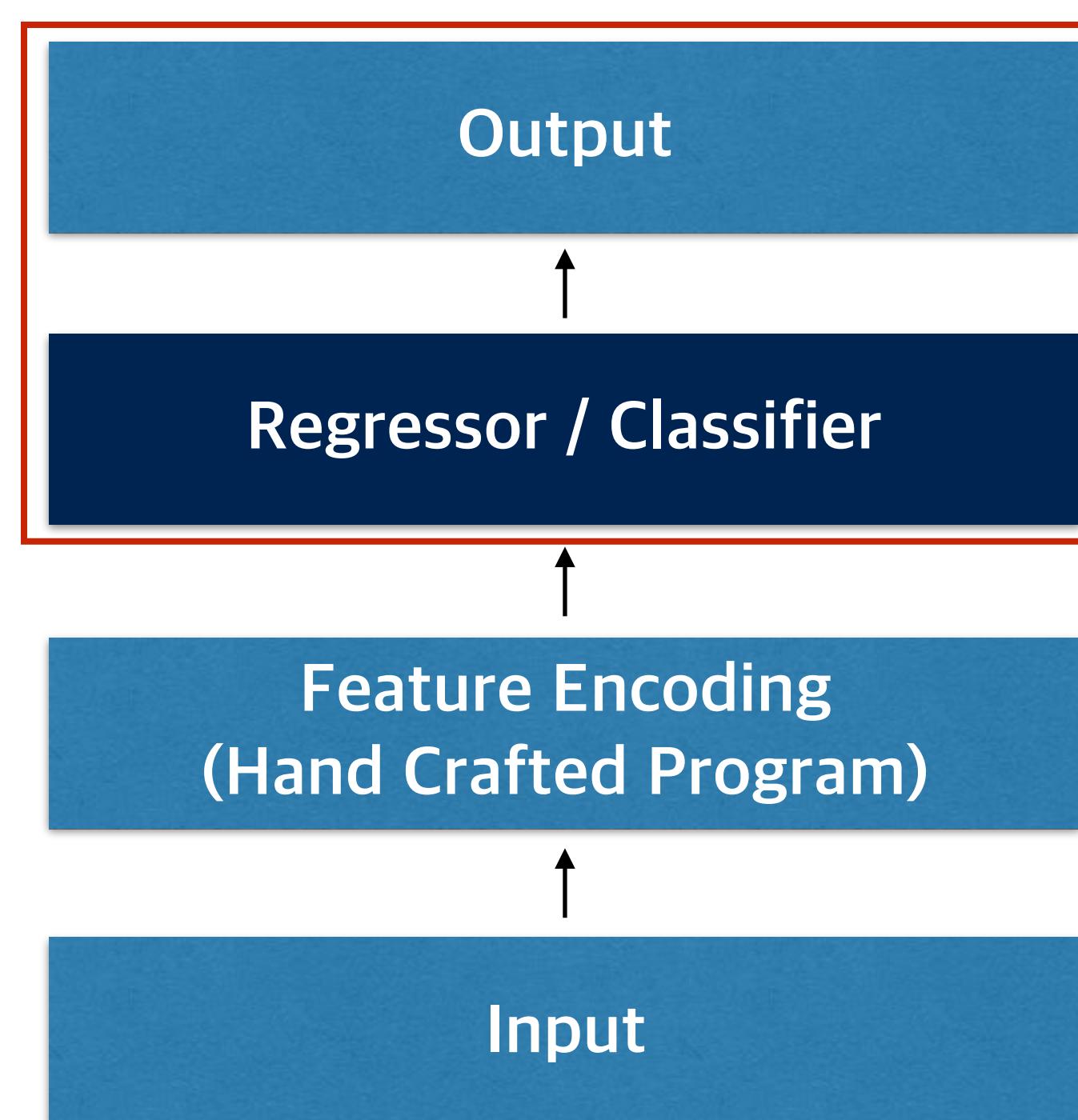
<Deep Learning Model>



# Traditional ML vs. Deep Learning

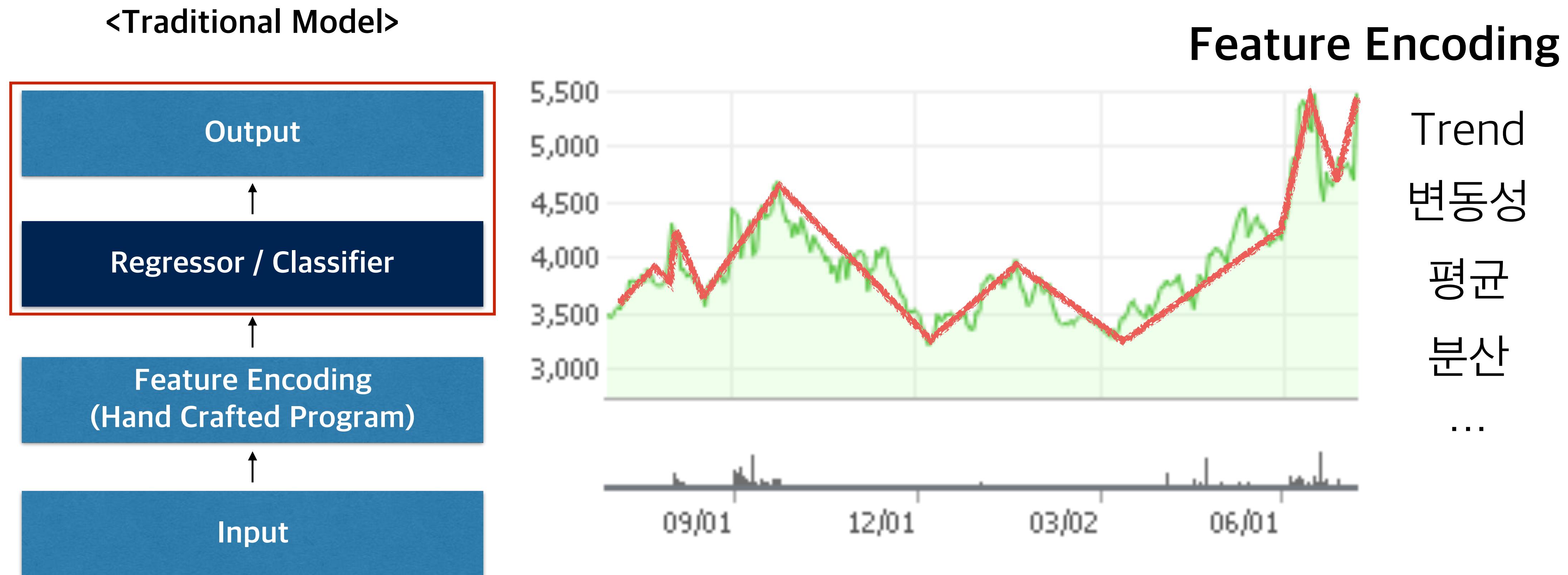
- ❑ 전통적인 머신러닝은 기본적으로 직접 Feature Encoding 후, 이를 바탕으로 학습을 진행함
- ❑ 딥러닝의 경우, 모델 내부에 **Feature Extractor**가 포함되며 함께 학습을 진행

## <Traditional Model>



# Traditional ML vs. Deep Learning

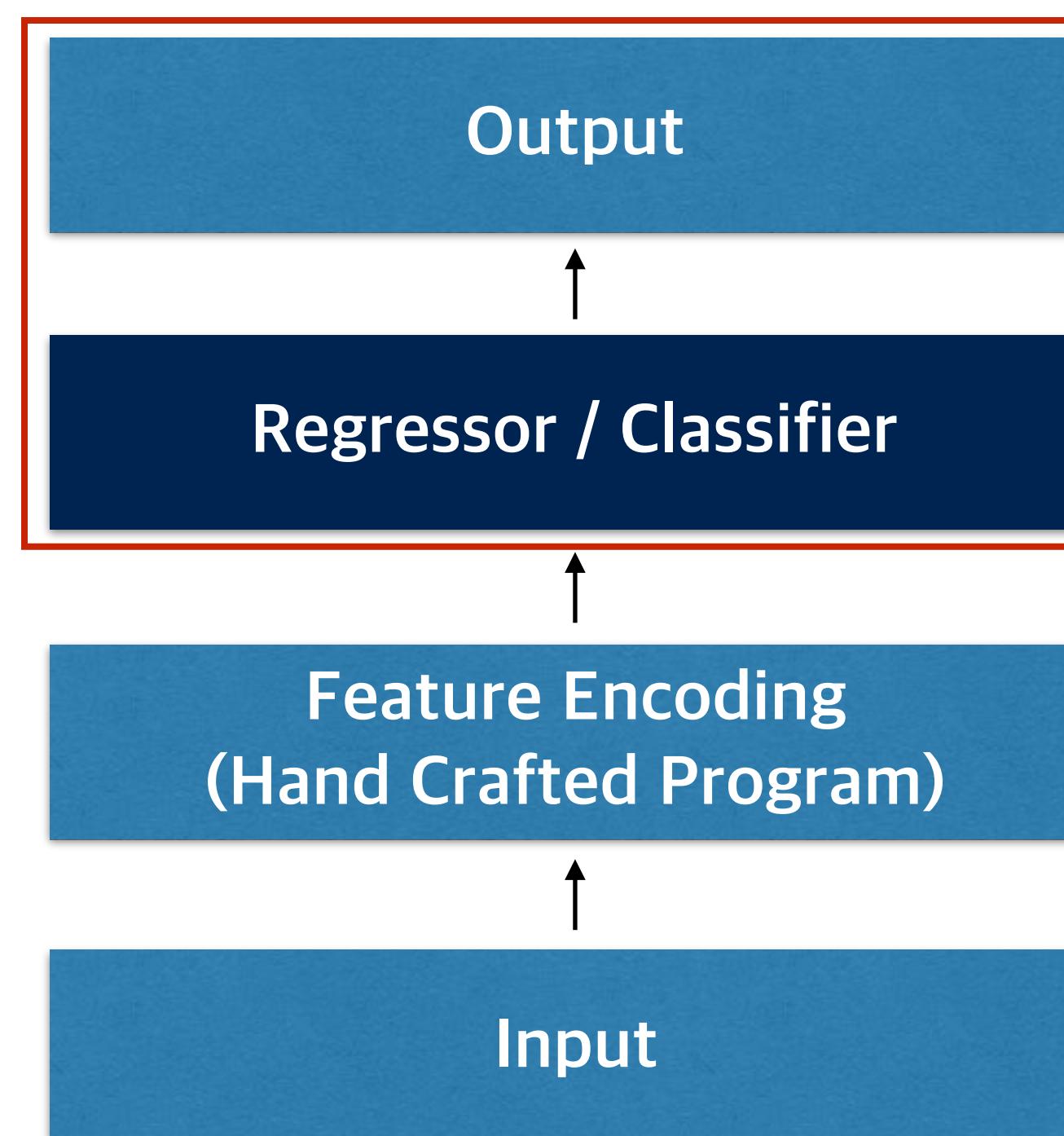
- ❑ 전통적인 머신러닝은 기본적으로 직접 Feature Encoding 후, 이를 바탕으로 학습을 진행함
- ❑ 딥러닝의 경우, 모델 내부에 **Feature Extractor**가 포함되며 함께 학습을 진행



# Traditional ML vs. Deep Learning

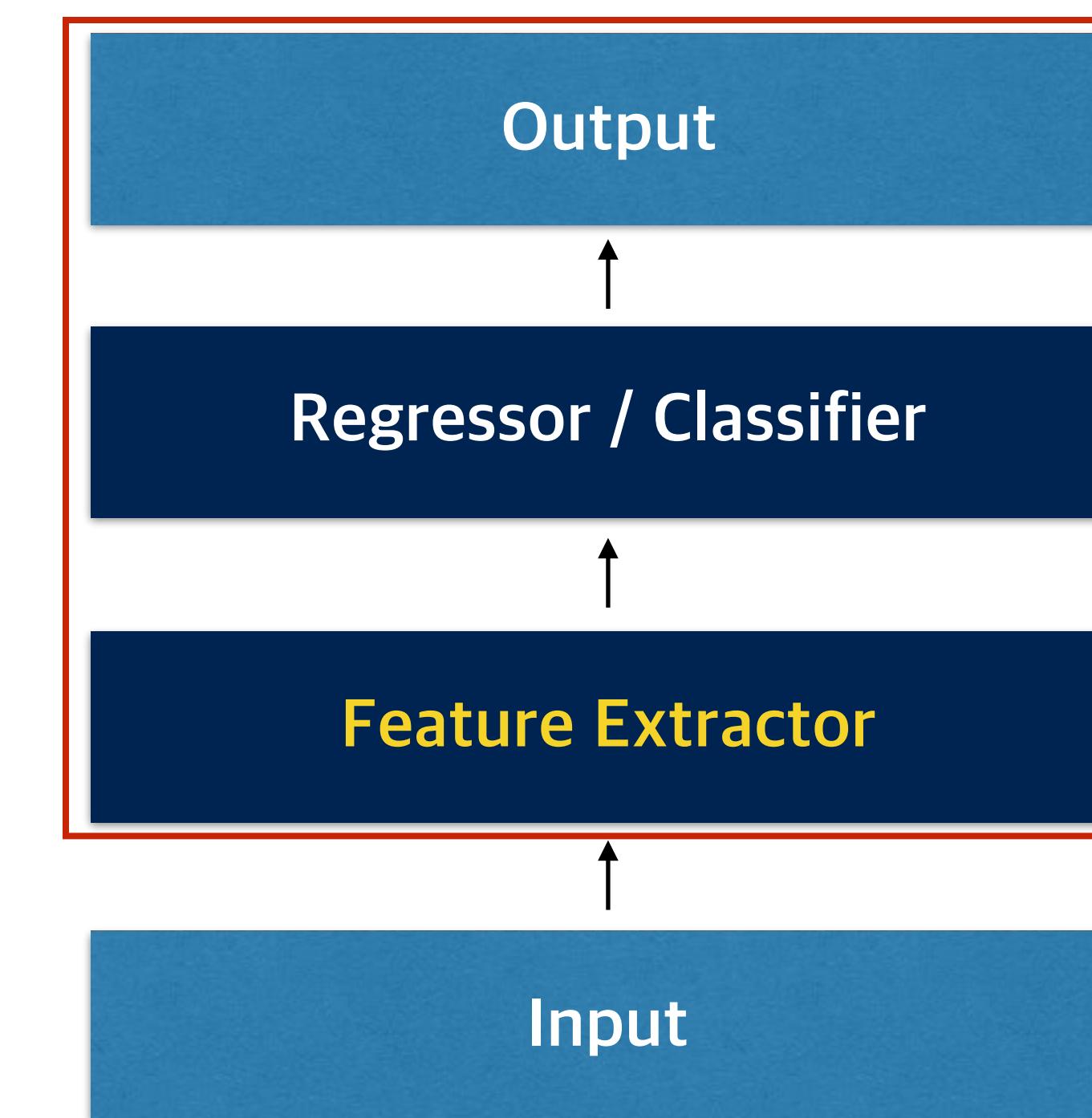
- ❑ 전통적인 머신러닝은 기본적으로 직접 Feature Encoding 후, 이를 바탕으로 학습을 진행함
- ❑ 딥러닝의 경우, 모델 내부에 **Feature Extractor**가 포함되며 함께 학습을 진행

<Traditional Model>



Learning Part

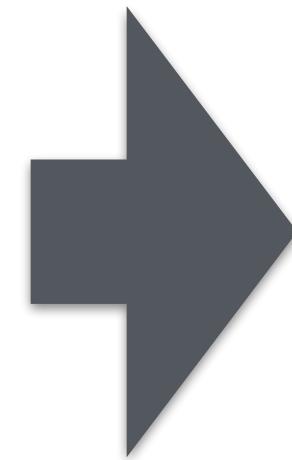
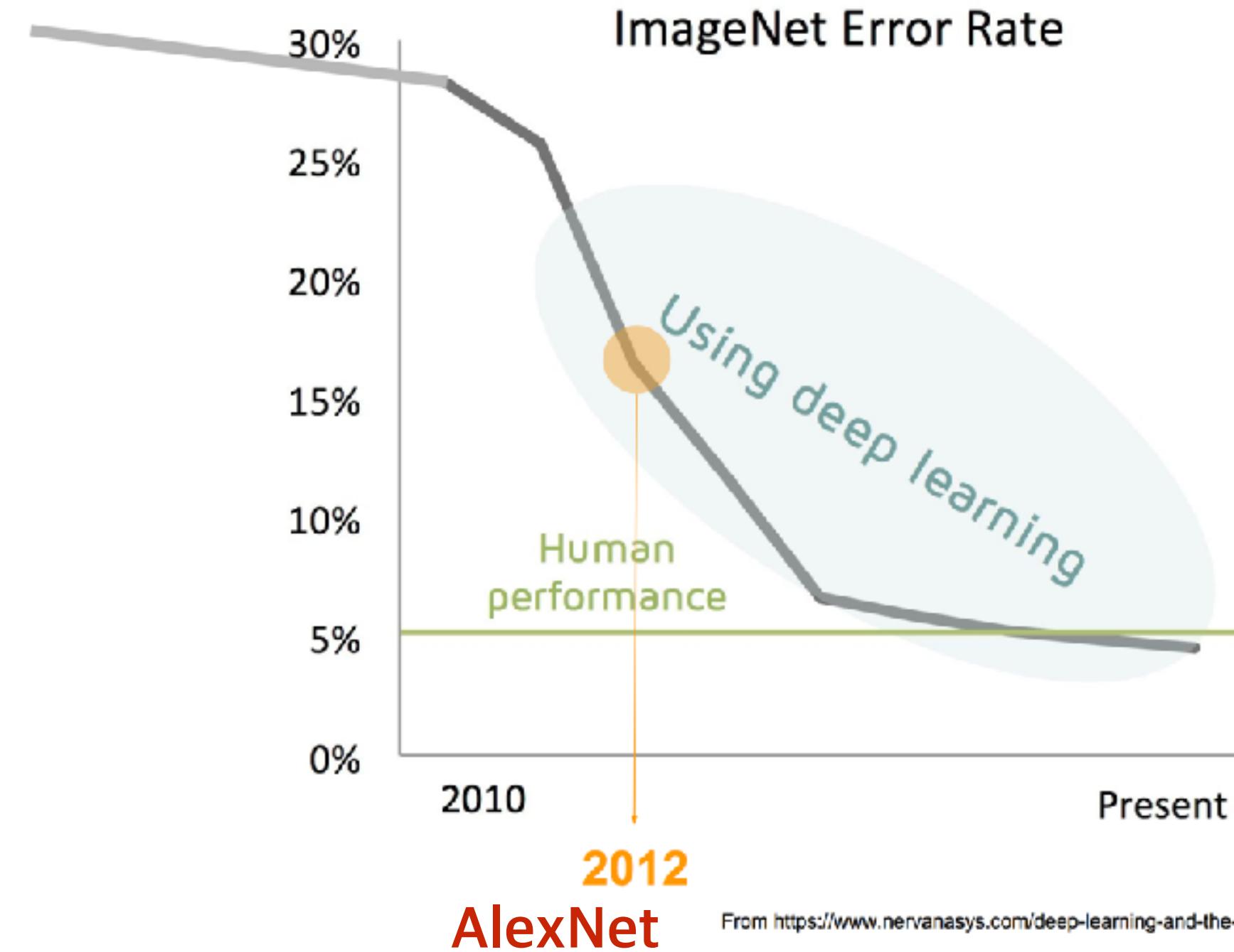
<Deep Learning Model>



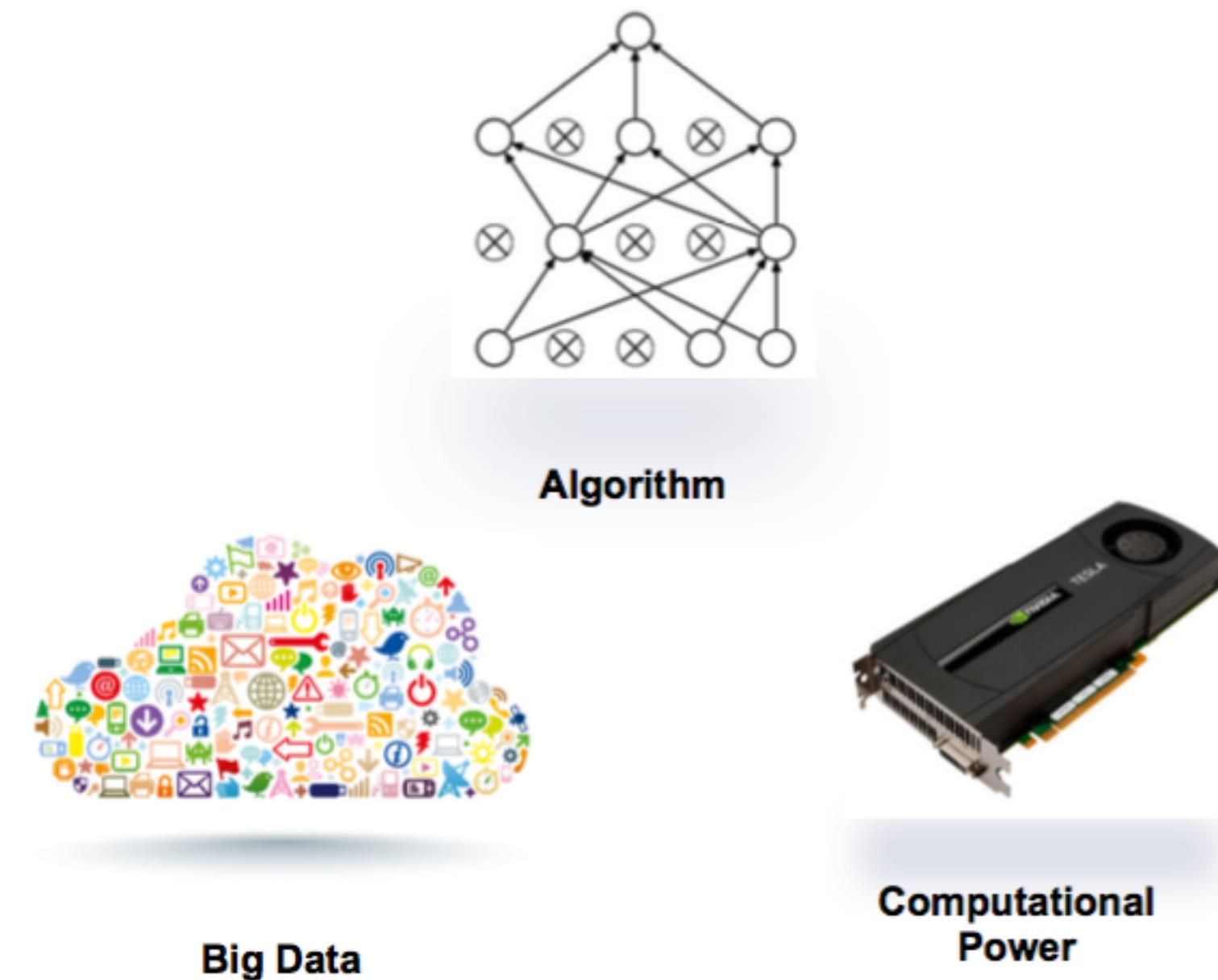
# Why Now?

## □ Deep Learning 알고리즘 자체는 최신이 아님

- Alexnet 2012 based on CNN (LeCunn, 1989)
- Alpha Go based on RL and MCTS (Sutton, 1998)



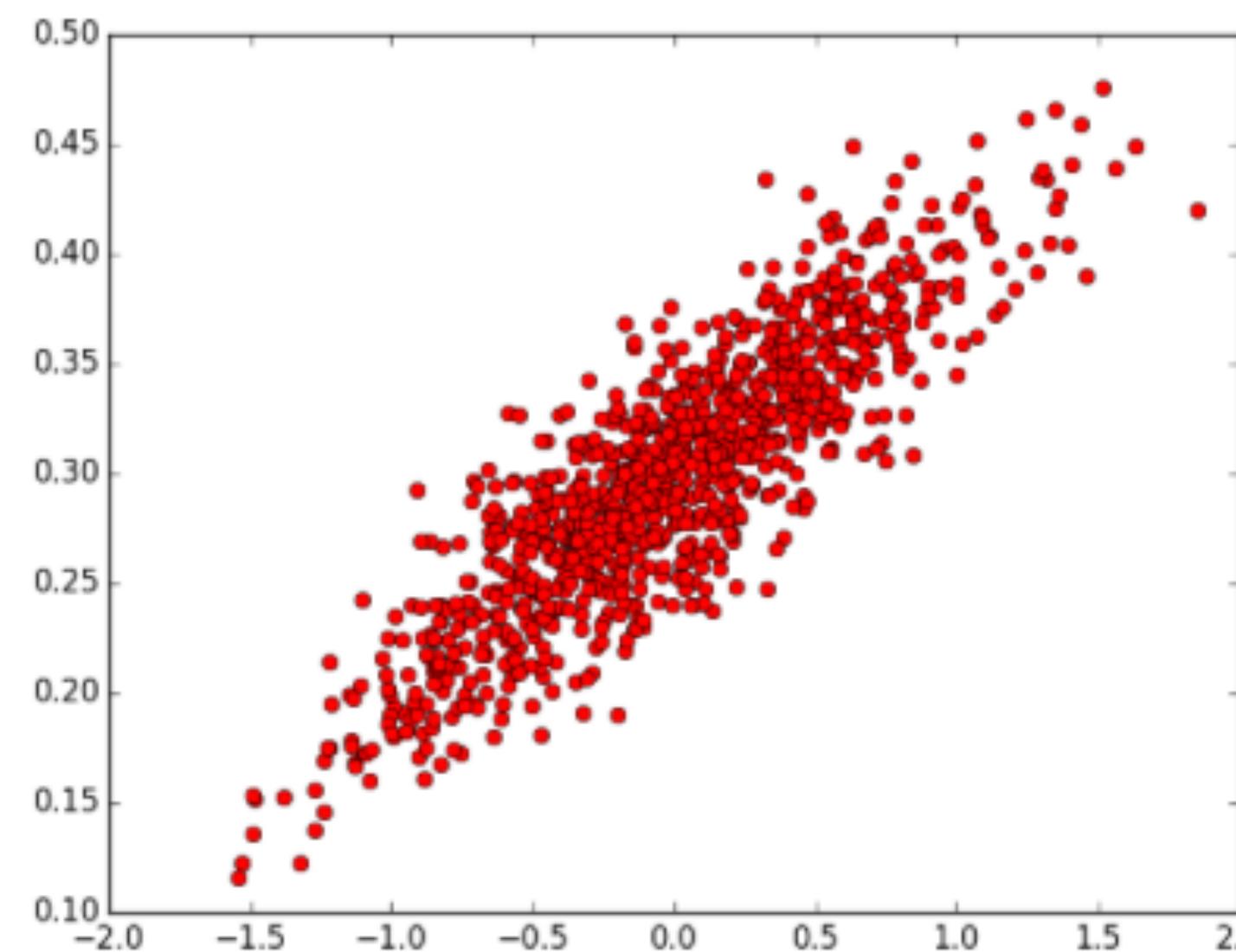
# Why Now?



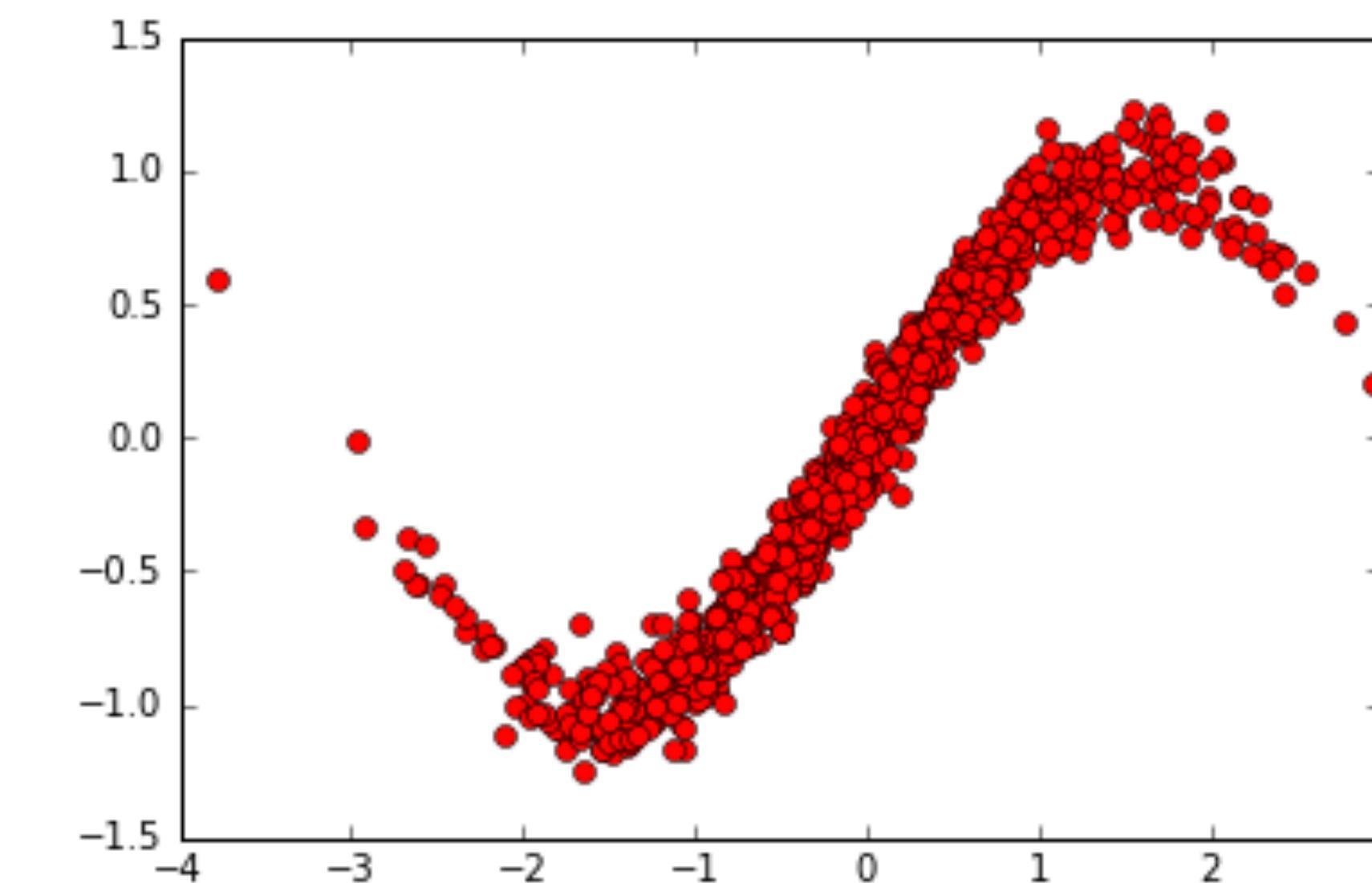
# Basic Idea Understanding Neural Network

- Neural Network의 기본 아이디어는 선형 모델 (linear model)의 한계에서 시작함
- 복잡한 관계를 가진 데이터는 선형 모델로 설명할 수 없음 (then.... how?)

<Sample Dataset 1>



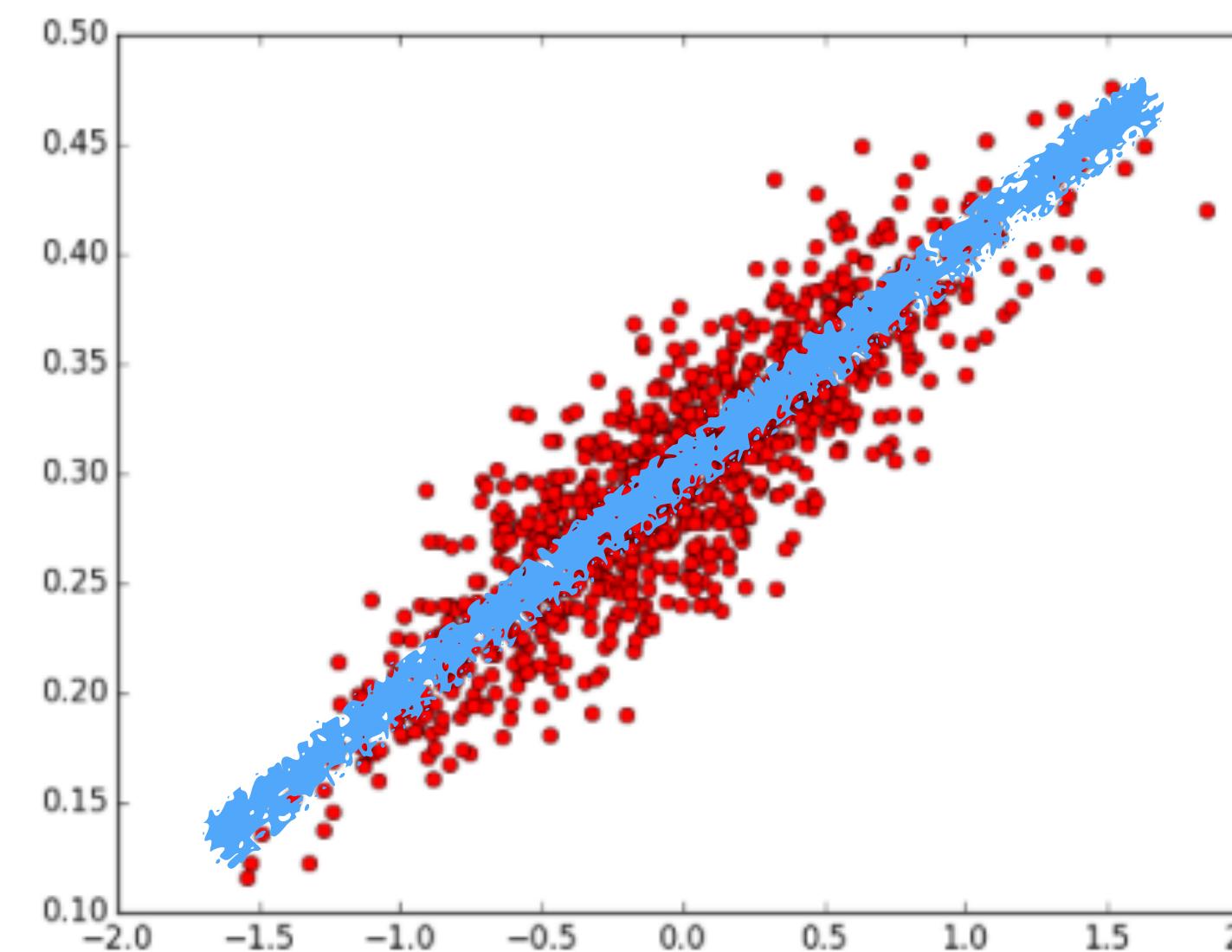
<Sample Dataset 2>



# Basic Idea Understanding Neural Network (Cont'd)

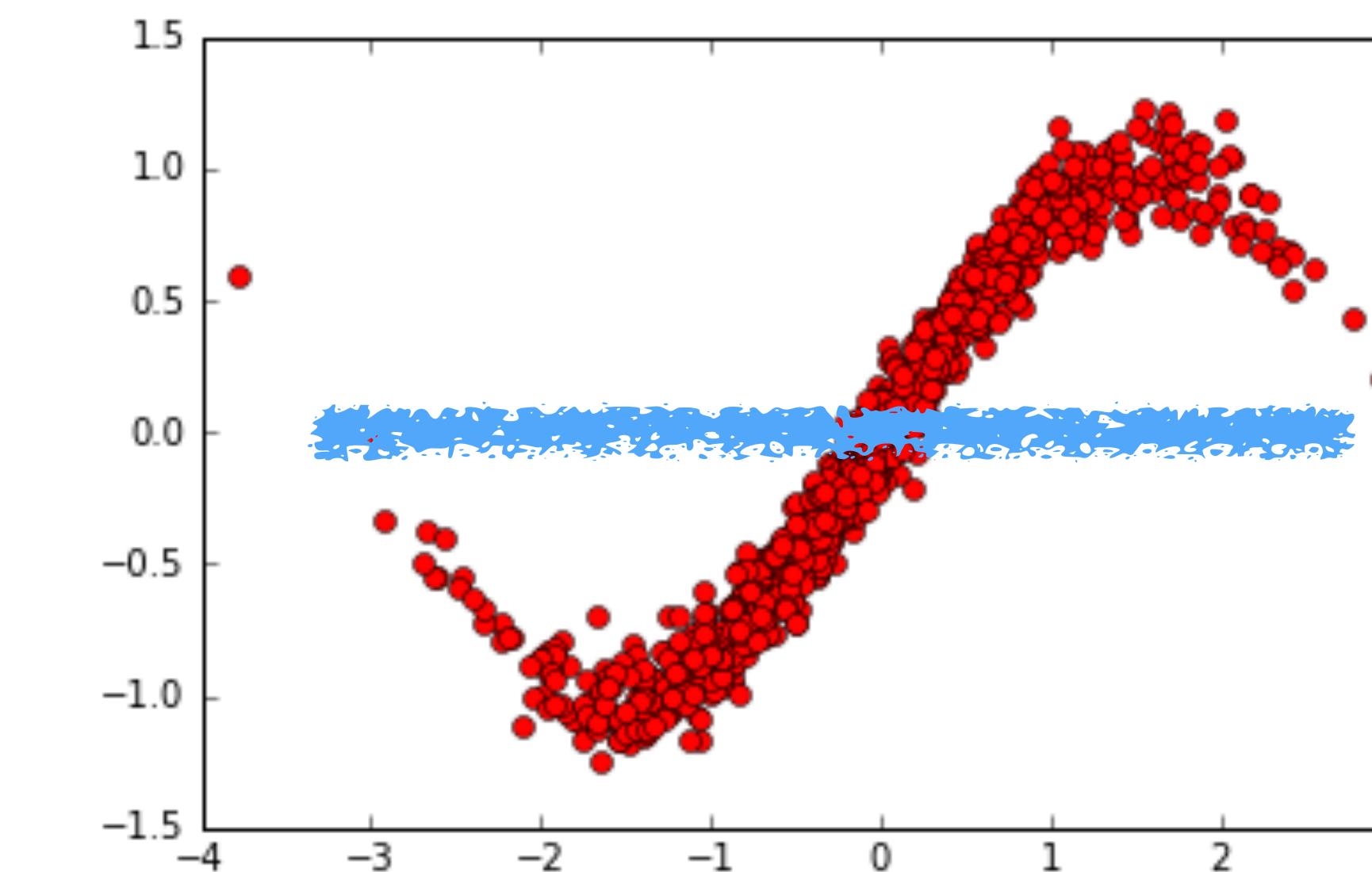
- ❑ Neural Network의 기본 아이디어는 선형 모델 (linear model)의 한계에서 시작함
- ❑ 복잡한 관계를 가진 데이터는 선형 모델로 설명할 수 없음 (then.... how?)

<Sample Dataset 1>



Feasible on Linear Model

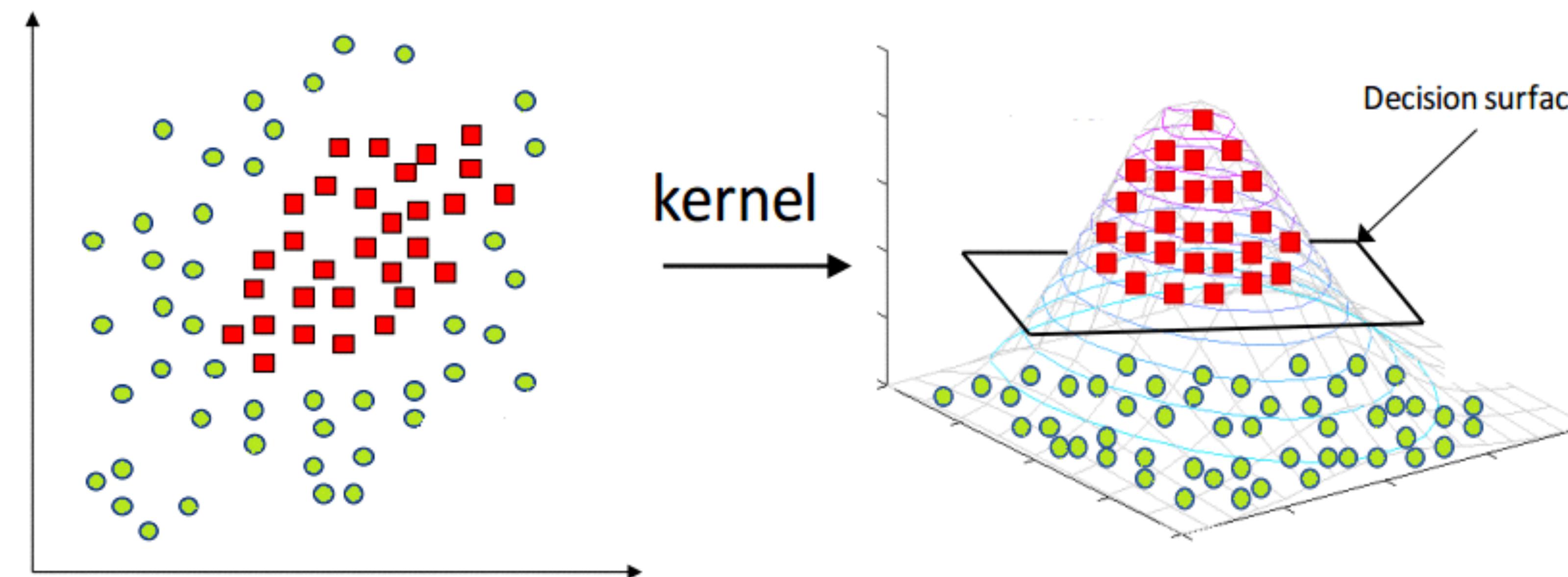
<Sample Dataset 2>



Infeasible on Linear Model

## Basic Idea Understanding Neural Network (Cont'd)

- 복잡한 관계를 가진 데이터를 설명하기 위한 데이터 변형 (Transformation) 방법에 대해 연구를 진행
- 데이터 분석의 데이터 전처리(data preprocessing) 단계에서 적절한 형태로 데이터를 변형함
- Kernel Method가 복잡한 데이터를 변형하기 위한 주된 방법 중 하나

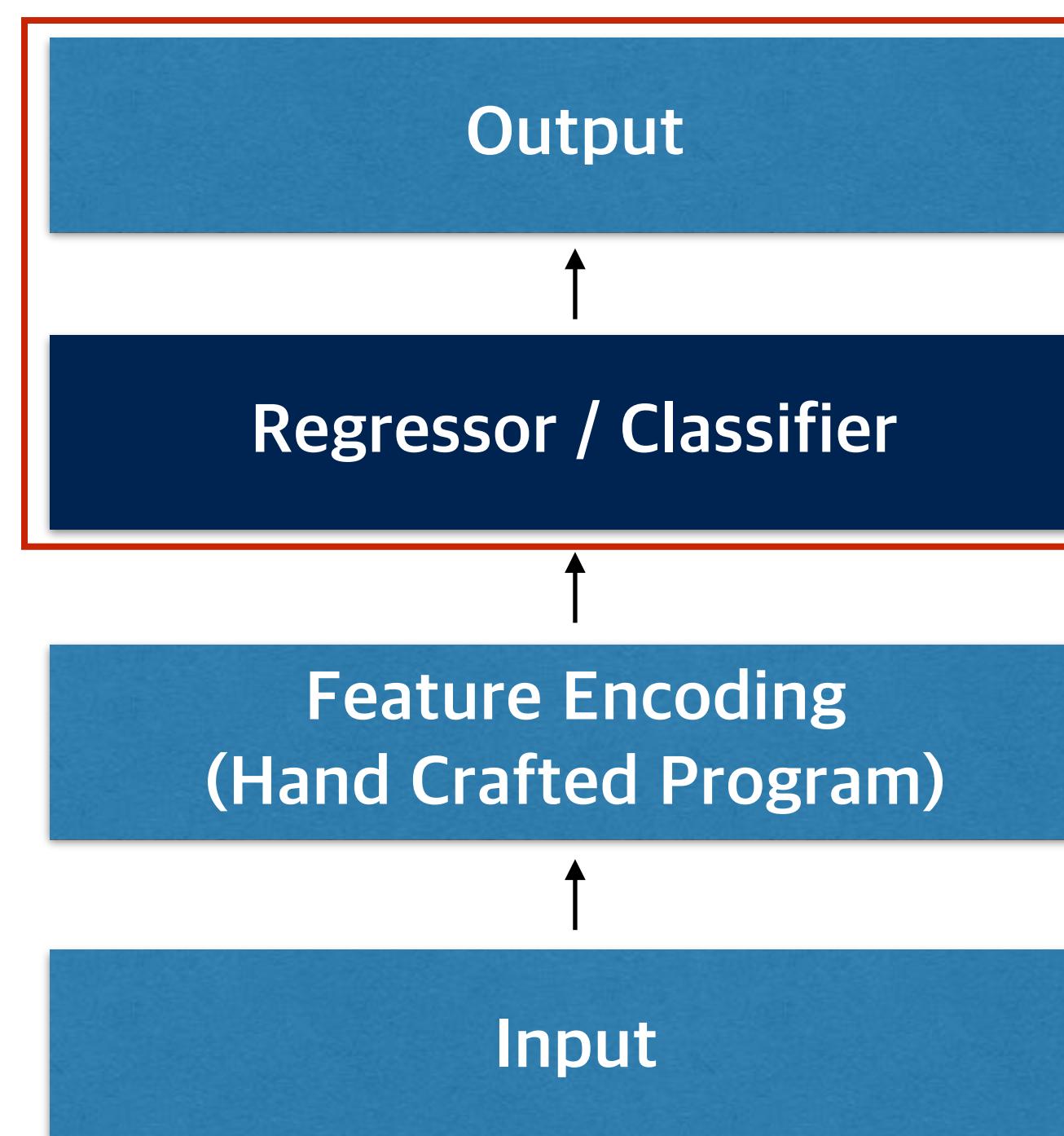


데이터에 적합한 적절한 Kernel을 사람이 직접 설정해야함 (기존 머신 러닝의 Feature Encoding)

# Traditional ML vs. Deep Learning

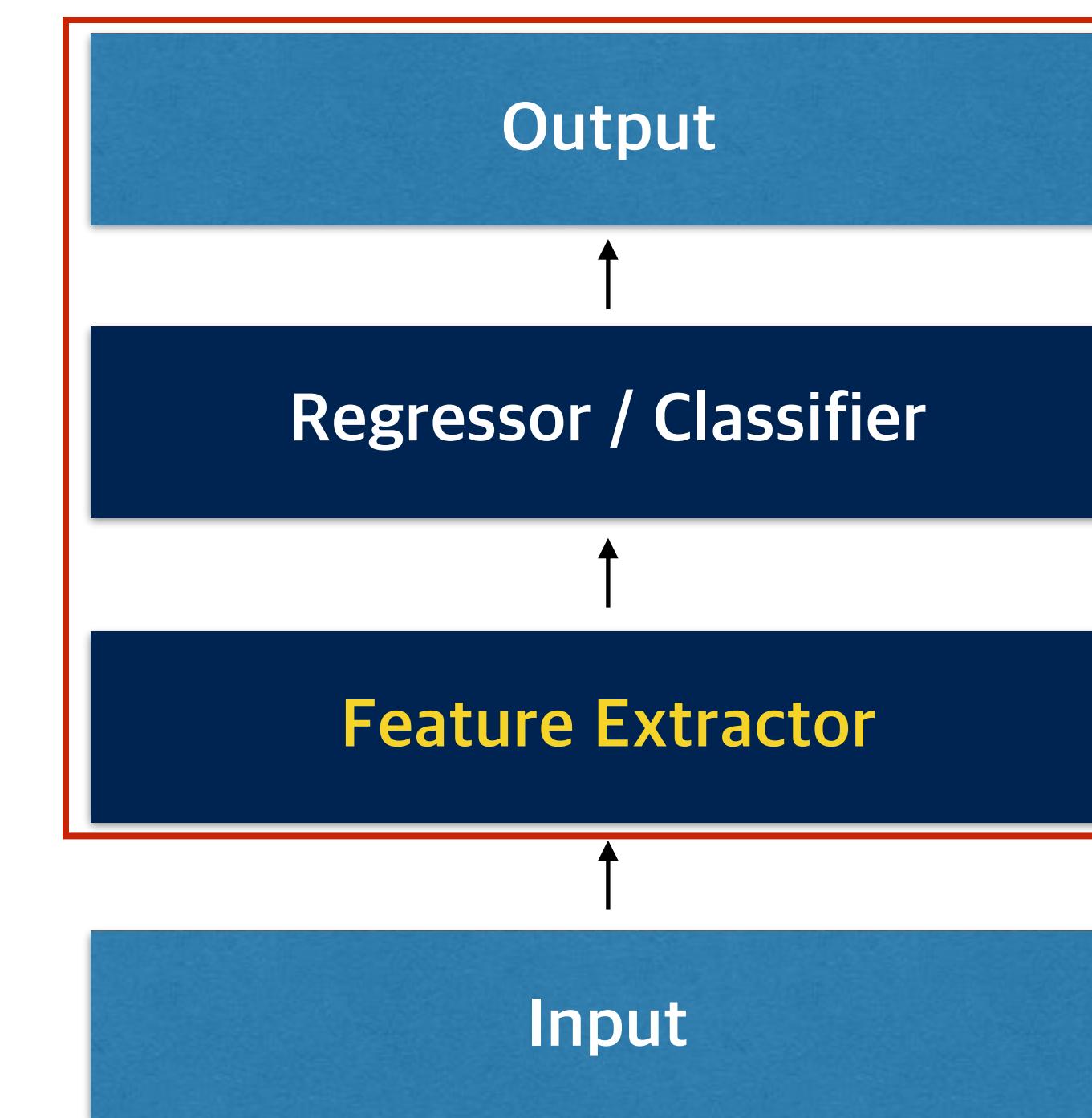
- ❑ 전통적인 머신러닝은 기본적으로 직접 Feature Encoding 후, 이를 바탕으로 학습을 진행함
- ❑ 딥러닝의 경우, 모델 내부에 **Feature Extractor**가 포함되며 함께 학습을 진행

<Traditional Model>



Learning Part

<Deep Learning Model>



# 값을 예측하는 가장 기본적인 방법은?

(ex. 비트코인 캐쉬 가격, 엑셈 주가, 부동산 가격…)

# Linear Regression

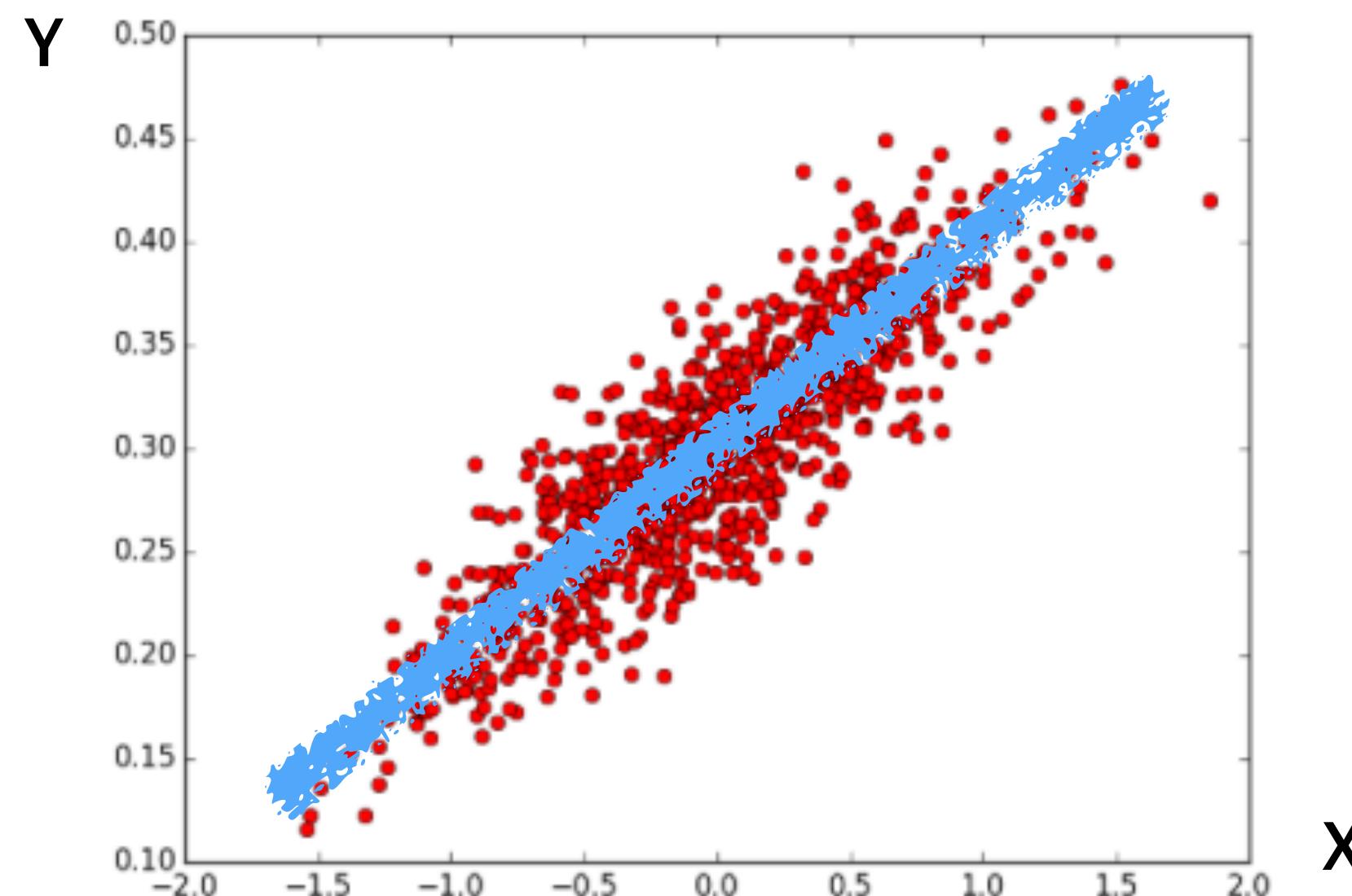
# Regression

- Regression은 종속 변수  $y$ 와 한 개 이상의 독립변수  $x$ 와의 상관관계를 모델링하는 기법 중 하나임 (Supervised Learning)
- 독립 변수  $X$ 를 입력 값(input)으로 종속 변수  $Y$ 를 1) 예측하거나 2) 설명하기 위하여 사용되며, 머신 러닝에서는 예측이 주목적
- $X$ 와  $Y$  사이의 관계를 선형 모델(linear model)로 추정하는 것을 “Linear Regression”이라고 함
  - 비선형 관계를 모델링 하기 위해 다양한 기저 함수(basis function) 경우에도 linear regression으로 표현

# Regression

- Regression은 종속 변수  $y$ 와 한 개 이상의 독립변수  $x$ 와의 상관관계를 모델링하는 기법 중 하나임 (Supervised Learning)
- 독립 변수  $X$ 를 입력 값(input)으로 종속 변수  $Y$ 를 1) 예측하거나 2) 설명하기 위하여 사용되며, 머신 러닝에서는 예측이 주목적
- $X$ 와  $Y$  사이의 관계를 선형 모델(linear model)로 추정하는 것을 “Linear Regression”이라고 함
  - 비선형 관계를 모델링 하기 위해 다양한 기저 함수(basis function) 경우에도 linear regression으로 표현

<Linear Regression Example>

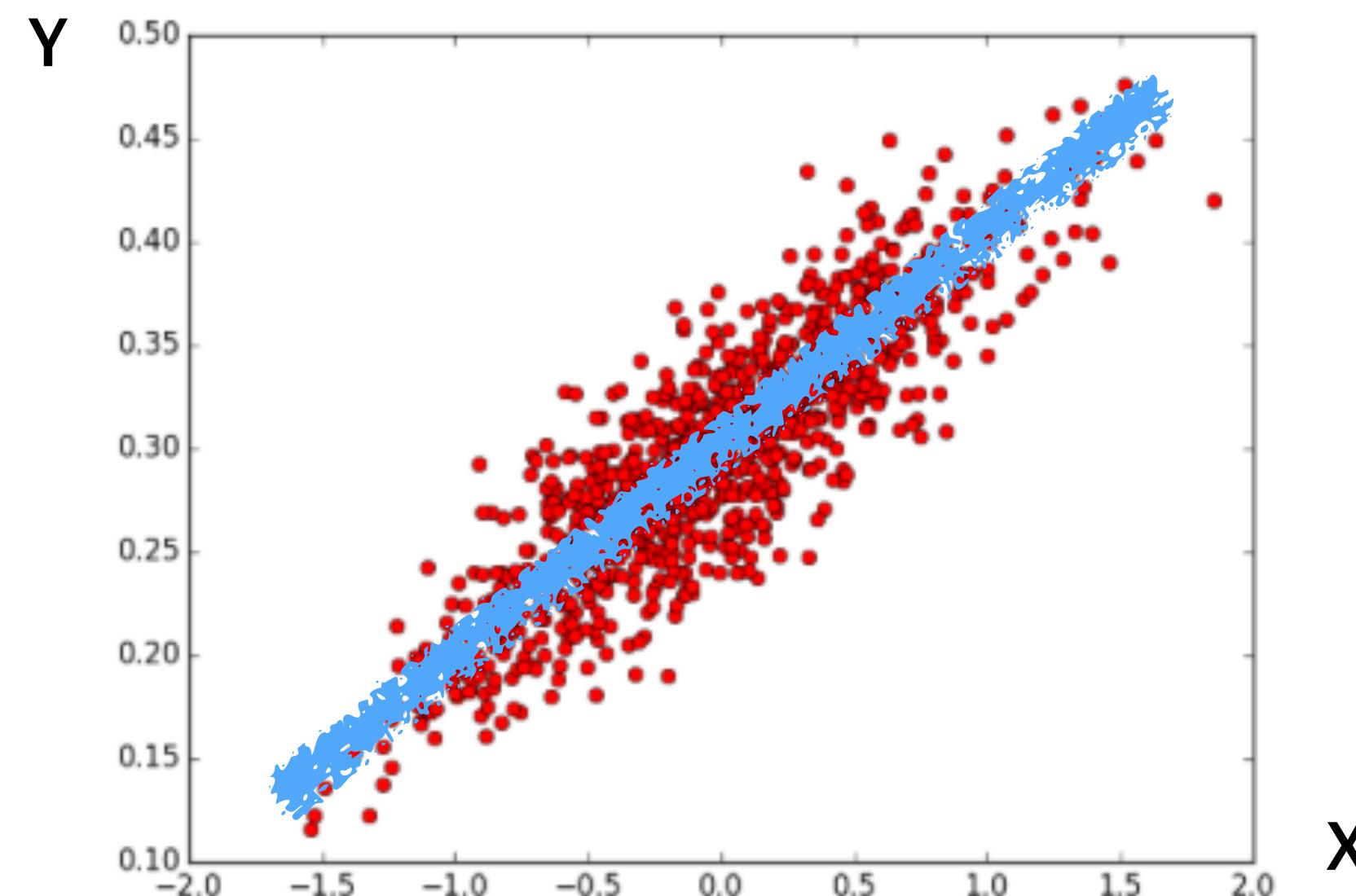


$$y = 0.1x + 0.3$$

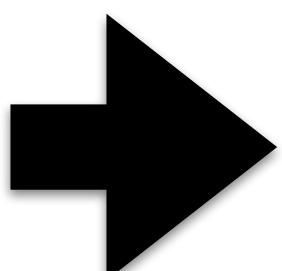
# Regression

- Regression은 종속 변수  $y$ 와 한 개 이상의 독립변수  $x$ 와의 상관관계를 모델링하는 기법 중 하나임 (Supervised Learning)
- 독립 변수  $X$ 를 입력 값(input)으로 종속 변수  $Y$ 를 1) 예측하거나 2) 설명하기 위하여 사용되며, 머신 러닝에서는 예측이 주목적
- $X$ 와  $Y$  사이의 관계를 선형 모델(linear model)로 추정하는 것을 “Linear Regression”이라고 함
  - 비선형 관계를 모델링 하기 위해 다양한 기저 함수(basis function) 경우에도 linear regression으로 표현

<Linear Regression Example>



$$y = 0.1x + 0.3$$

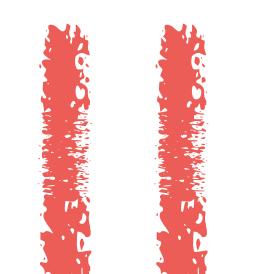


Input:



Independent Variable:  $X$

Parameter:



Weight Vector:  $W$

(vector, 기울기)

Bias Term:  $b$

(scalar, 절편)

Output:

Response:  $Y$

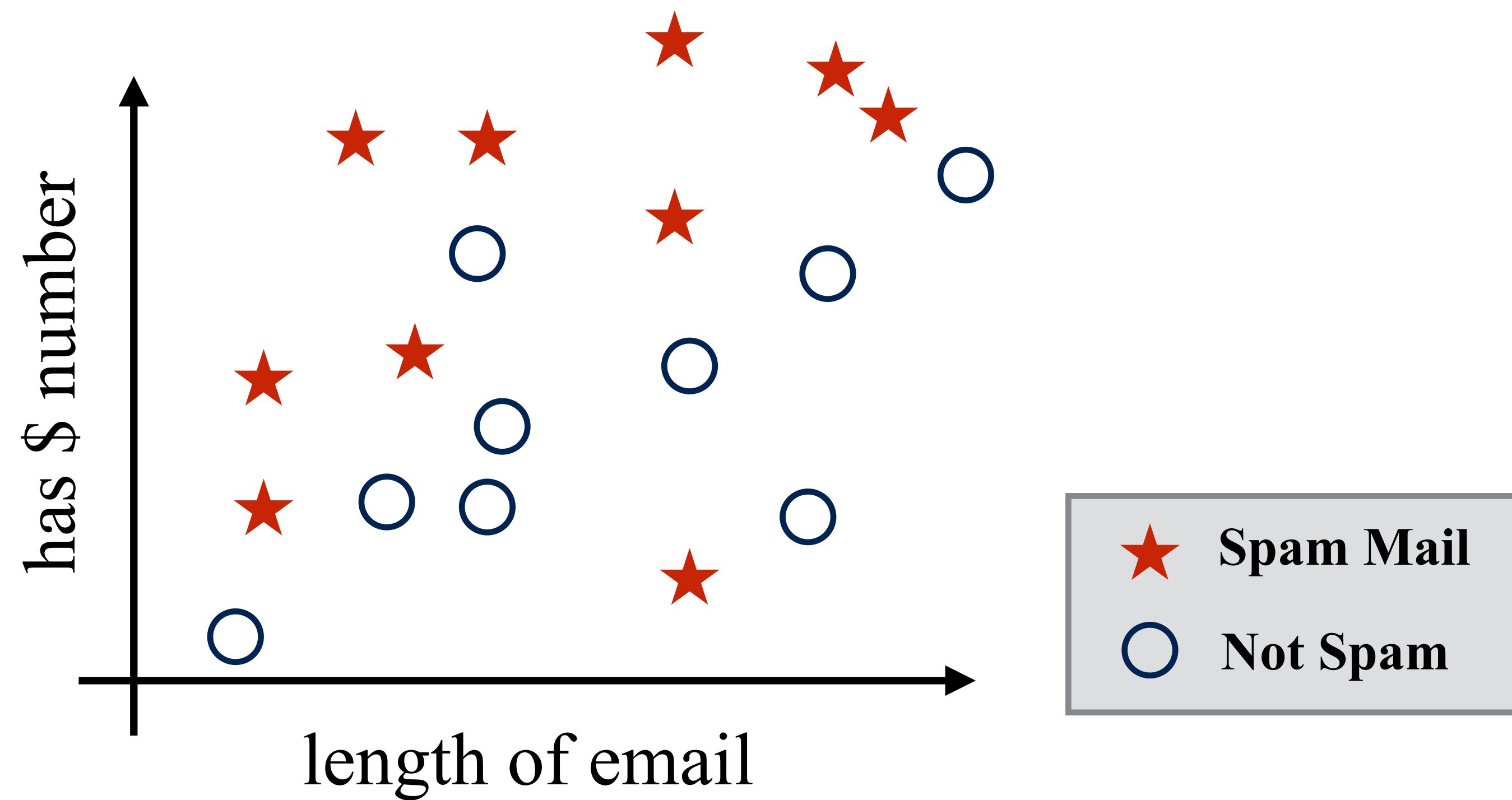
$$Y = W^T X + b$$

**Class를 예측하는 가장 기본적인 방법은?**

# Logistic & Softmax

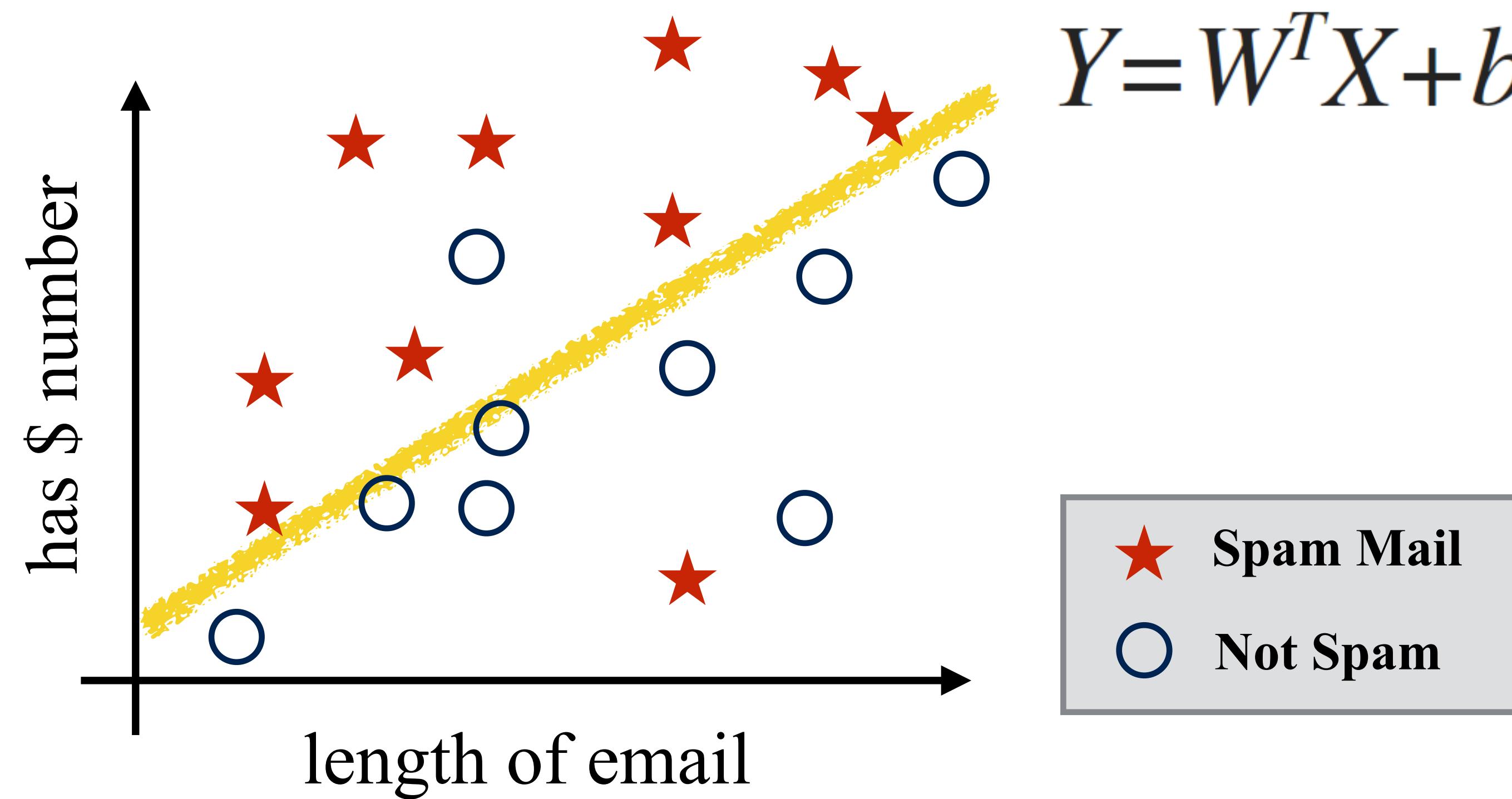
# Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
  - ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



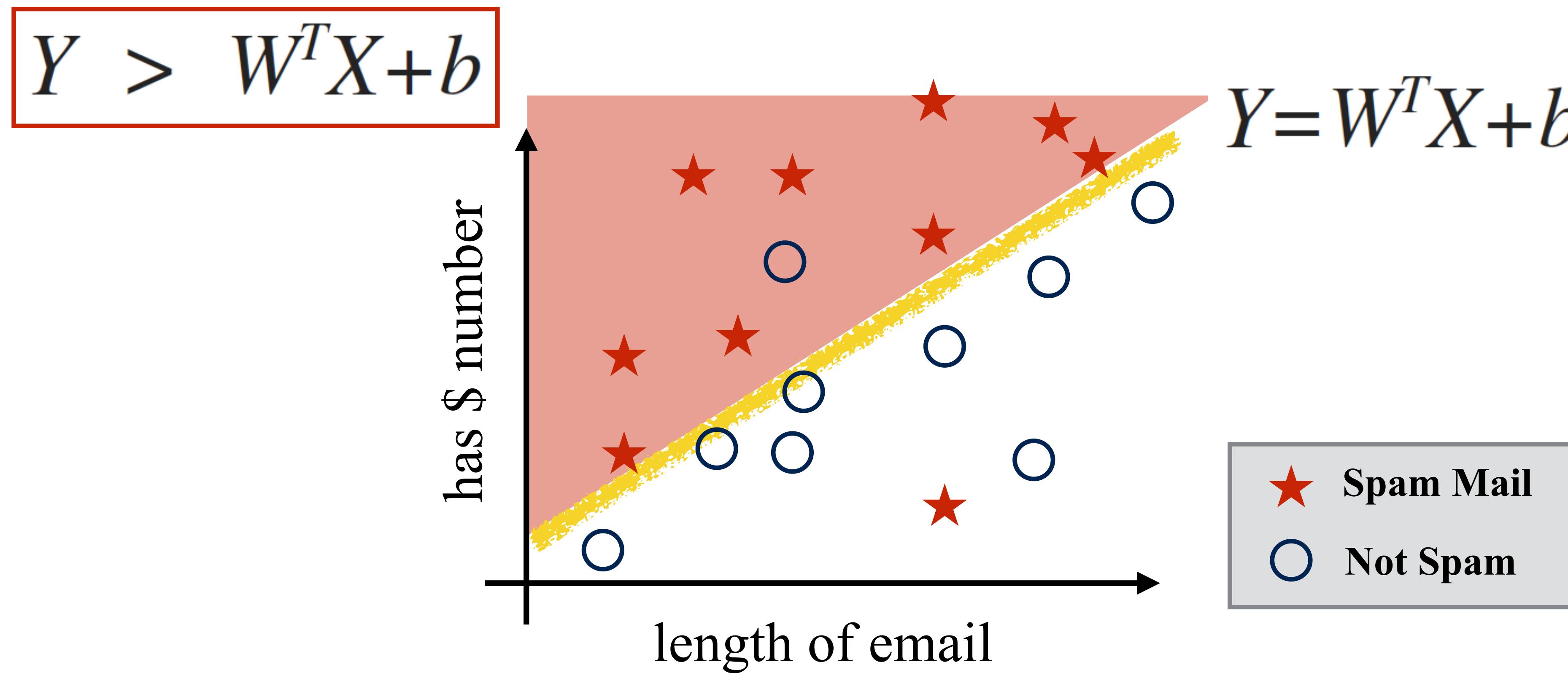
## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



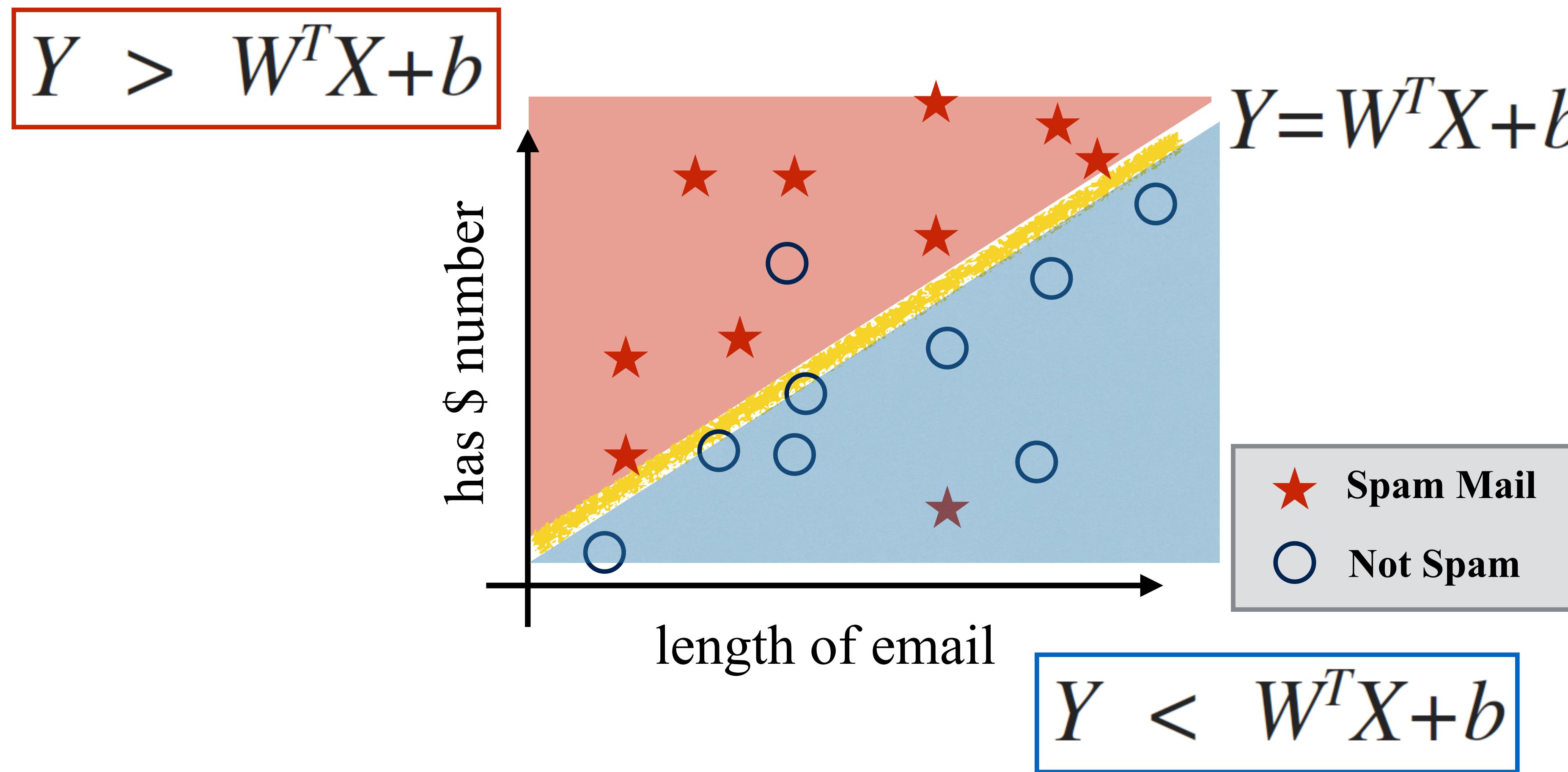
## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



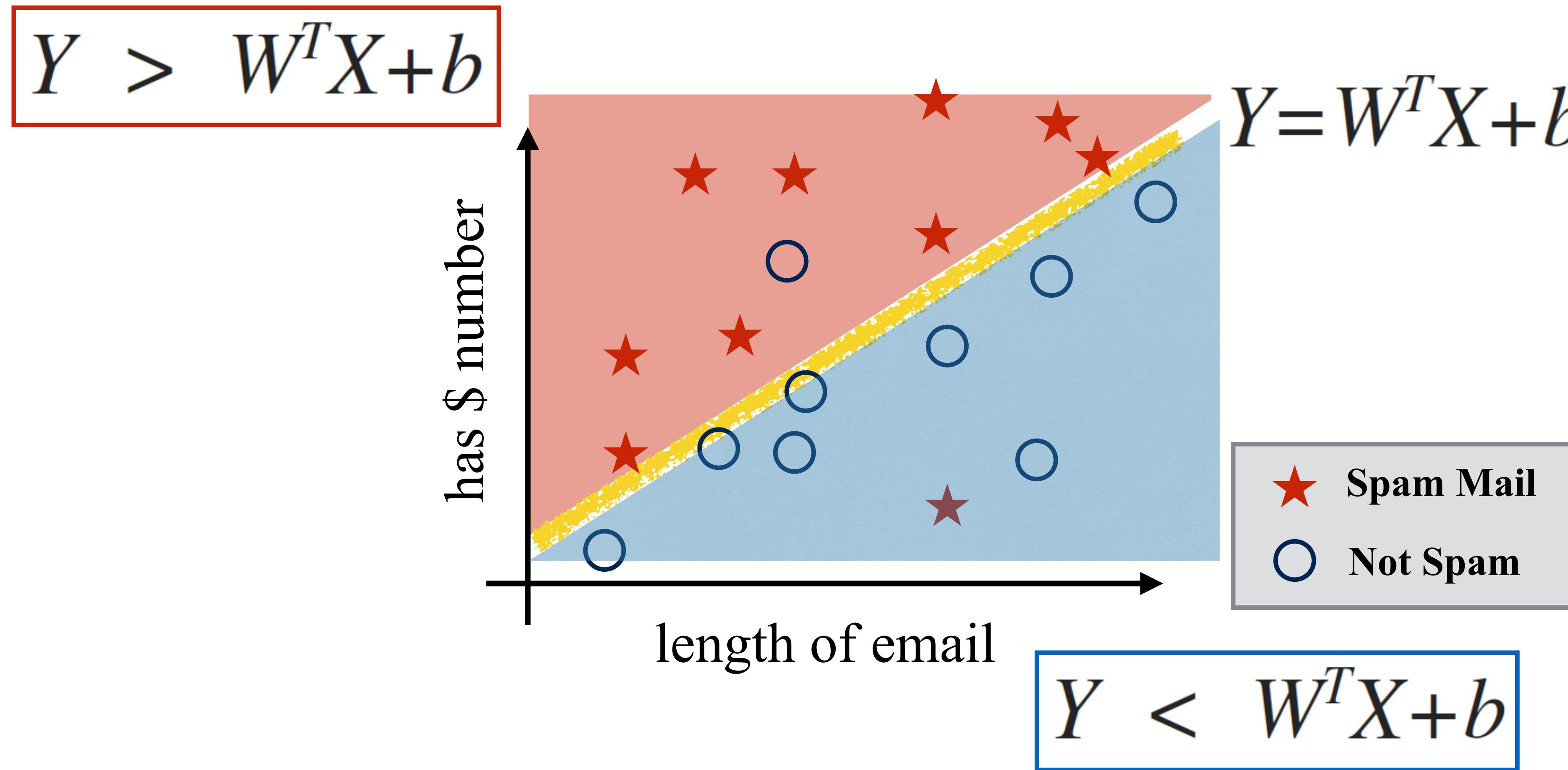
**In Machine Learning World.....**

In Machine Learning World.....

Classification is (always) with “Probability”

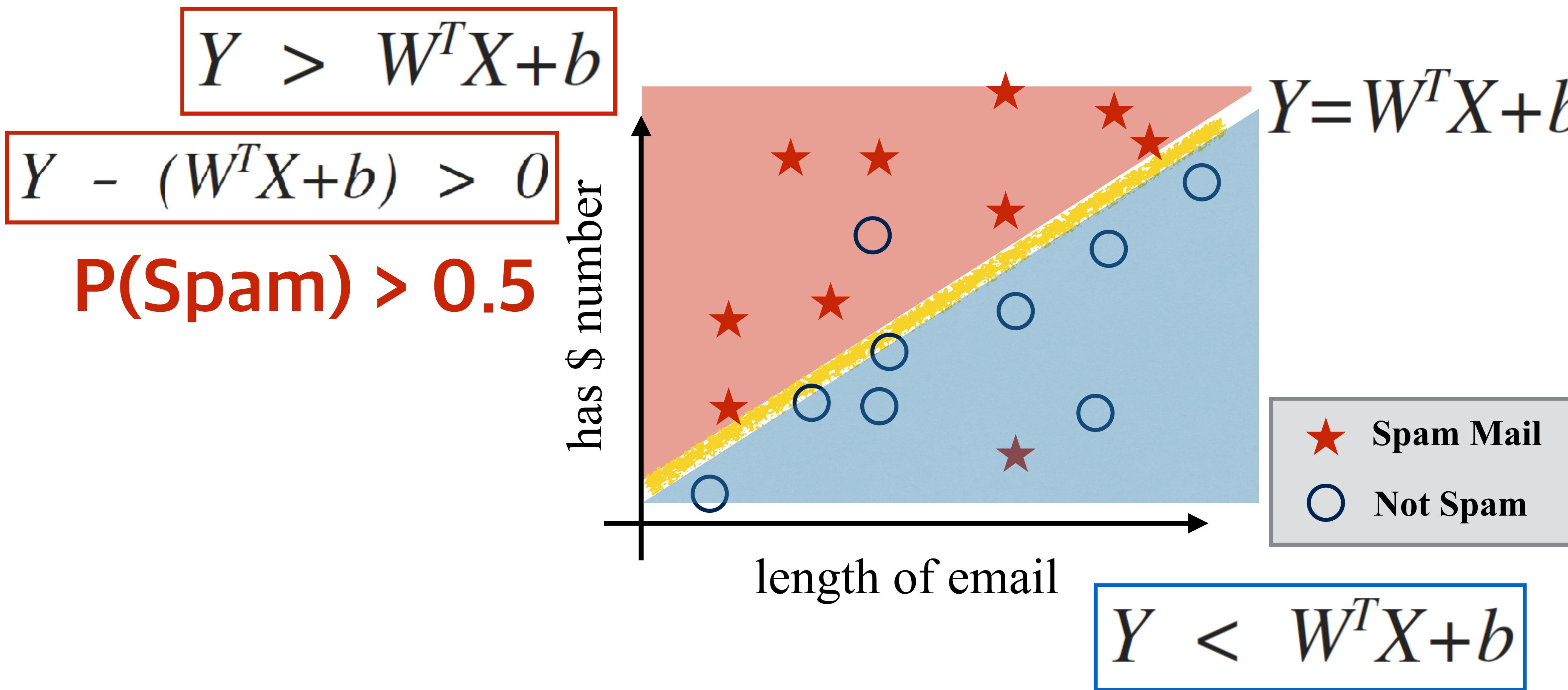
## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?



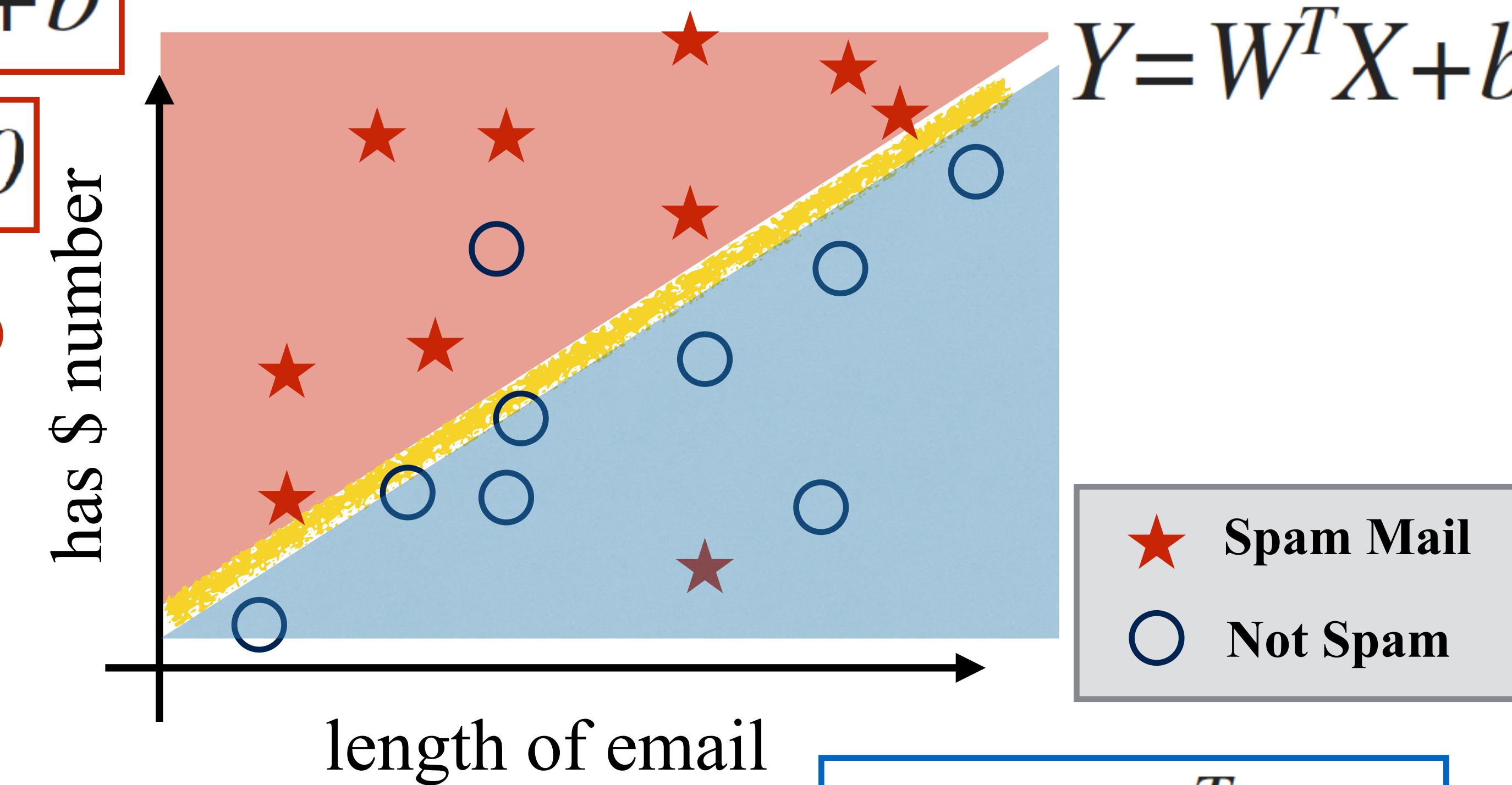
## Logistic Regression Intuition

- ❑ Spam 메일 여부는 1) 메일에 포함된 '\$' 개수 (has \$ number)와 2) 이메일 길이 (length of email)로 결정된다고 가정
- ❑ 아래 상황에서 Spam & Not Spam 메일을 구분하기 위해 사용할 수 있는 가장 간단한 방법은?

$$Y > W^T X + b$$

$$Y - (W^T X + b) > 0$$

P(Spam) > 0.5



$$Y = W^T X + b$$

★ Spam Mail  
○ Not Spam

P(Spam) < 0.5

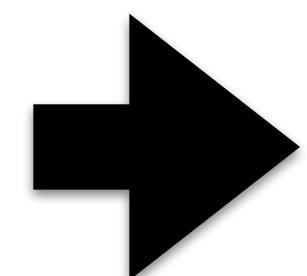
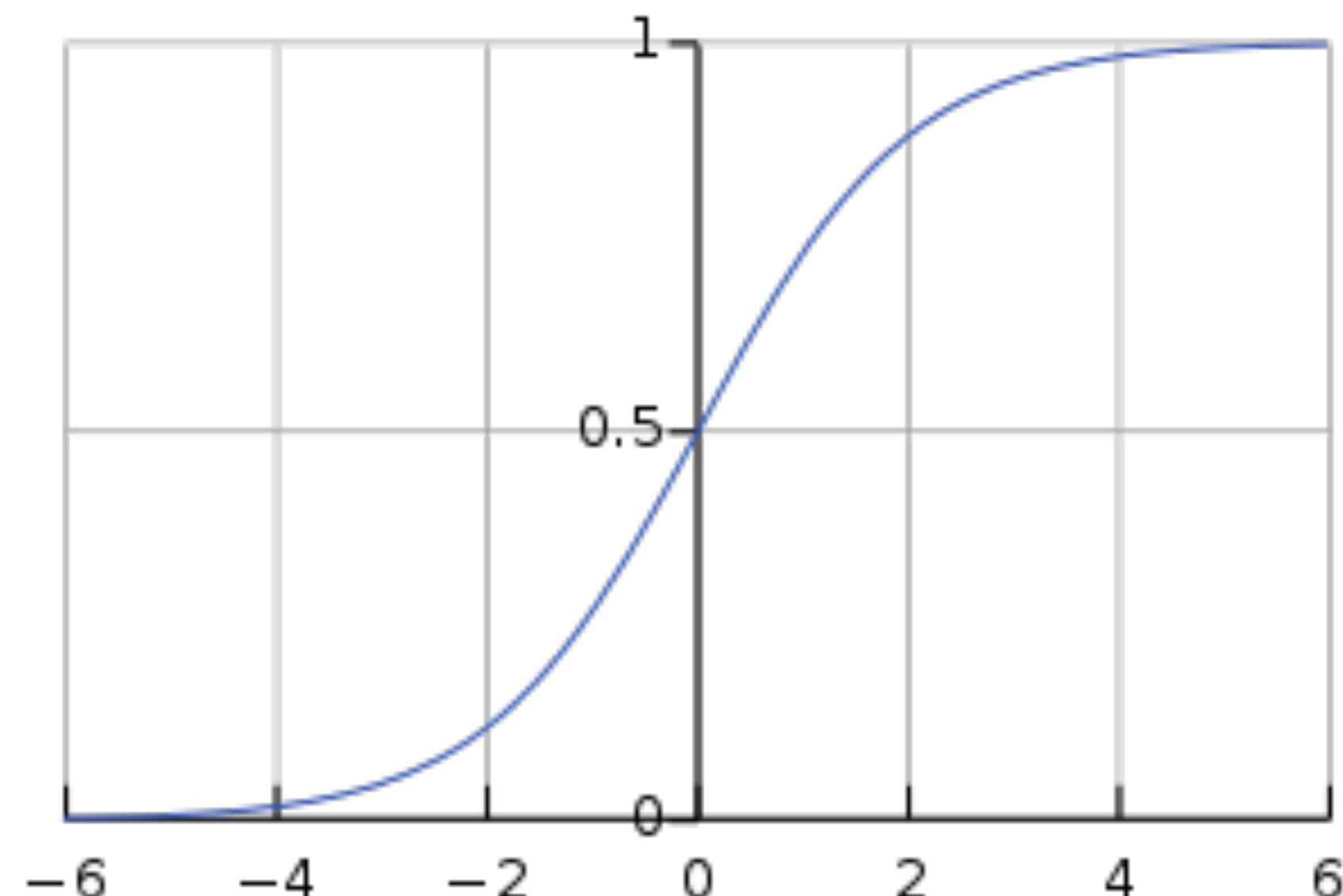
$$Y < W^T X + b$$

$$Y - (W^T X + b) < 0$$

# Logistic Regression

- Logistic Regression은 Sigmoid Function을 이용하여 특정 입력값이 양성(positive) class에 속할 확률을 계산
- $X$ 의 선형 모델의 값이 0 보다 크면 positive class, 0 보다 작으면 negative class에 분류

<Sigmoid Function>



$$\text{output} = p(\text{Class}_{(+)}|X) = \frac{1}{1+e^{-W^T X}}$$

If  $W^T X > 0$ ,  
then  $p(\text{Class}_{(+)}|X) > 0.5$  and positive class

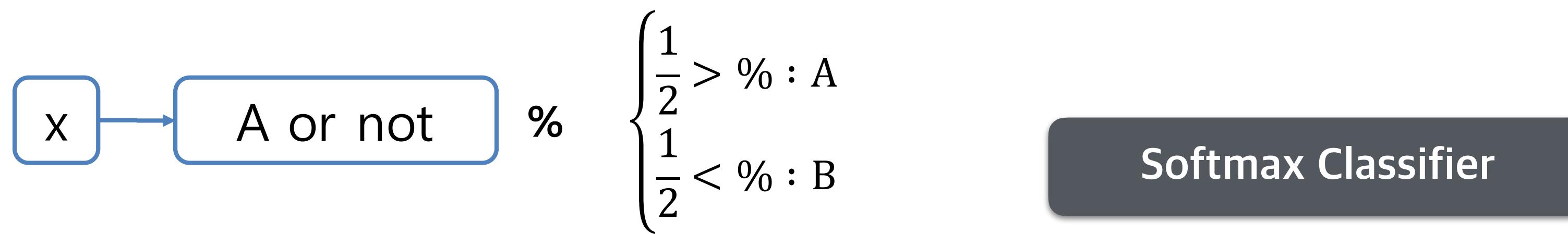
If  $W^T X < 0$ ,  
then  $p(\text{Class}_{(+)}|X) < 0.5$  and negative class

이진 분류(binary classification)을 넘어, 다중 클래스 분류 문제(multi-class)를 해결하기 위한 모델이 필요!

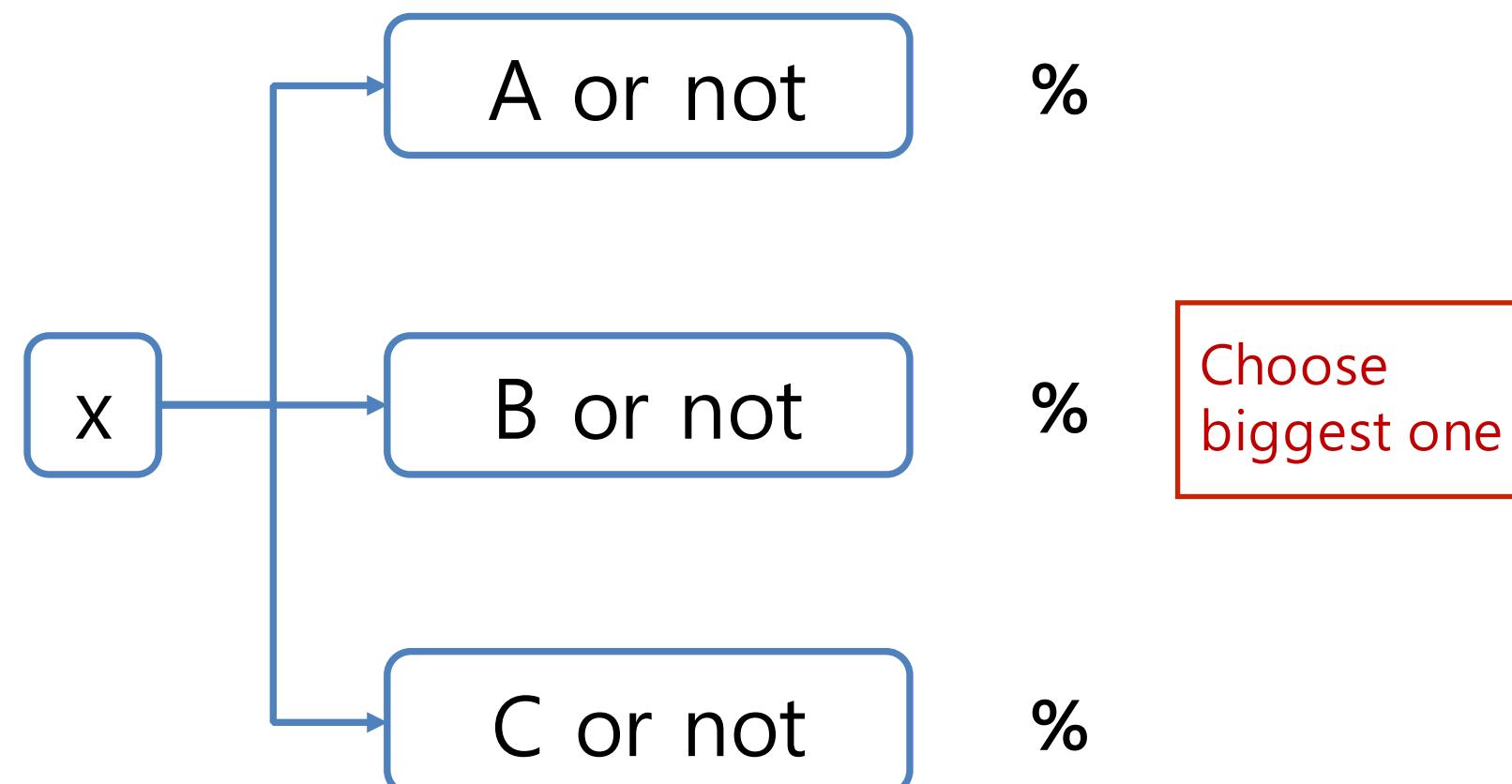
# Softmax

- 다중 클래스 (multi-class) 분류 문제는 각 클래스에 소속될 확률이 가장 높은 클래스를 분류 결과로 도출함
- Logistic regression의 아이디어를 유지한 방법으로 다중 클래스 문제를 해결하기 위해 일반화된 함수 사용: **Softmax**

## <Binary Case>



## <Multiple Case>



$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

$$y_i = w_i^T x = w_{i1}x_1 + w_{i2}x_2 + \dots$$

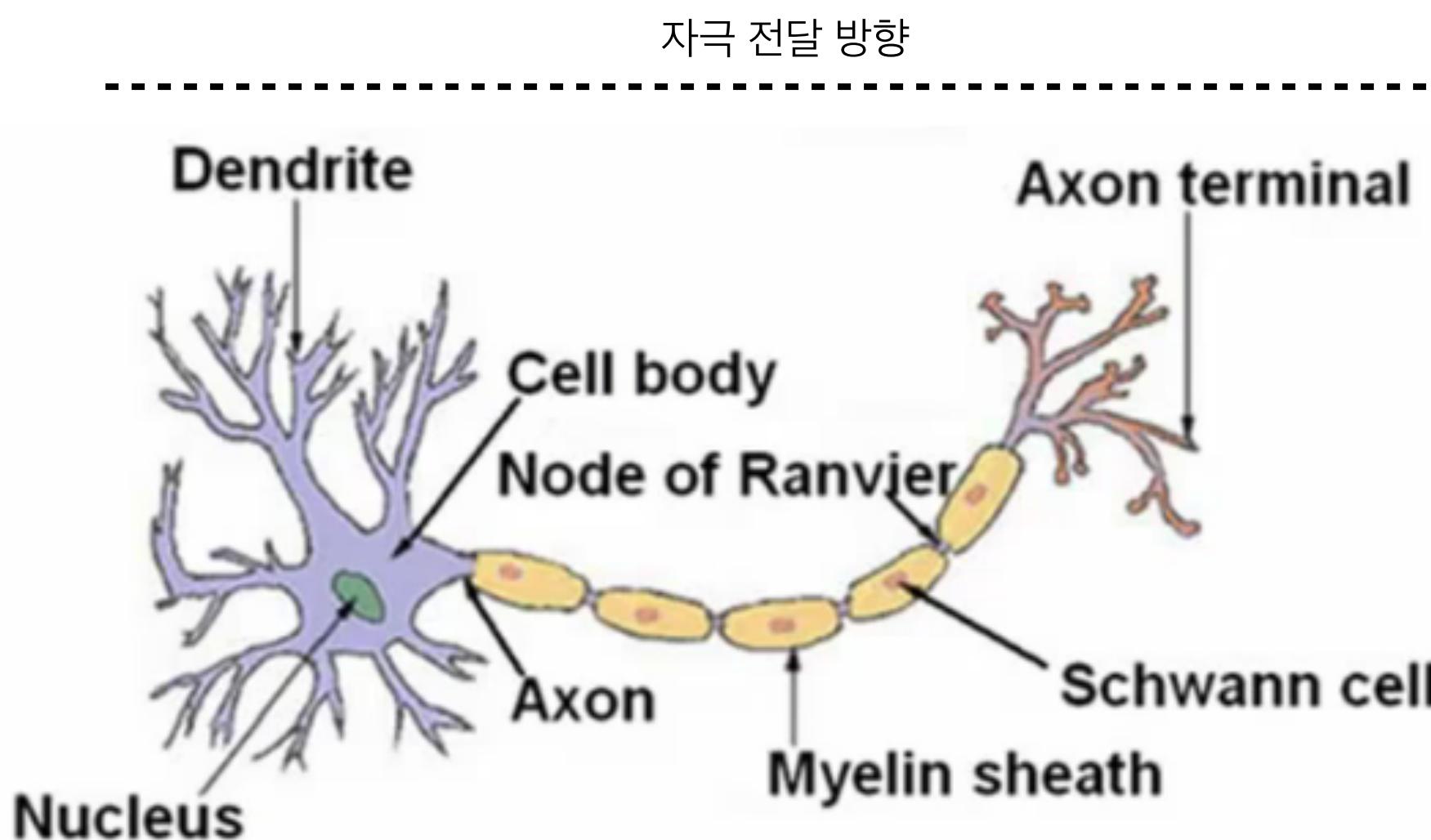
# Neural Network

# Principle of Cortical Neuron

## □ Neuron (신경세포, 뉴런)

- 신경계를 구성하는 구조적 및 기능적 단위이며, 전기적인 방법으로 신호를 전달함
- 수상돌기(Dendrite), 핵 (Nucleus), 축색돌기(Axon) 등으로 구성됨

<Cortical Neuron Structure>



- **Dendrite:**

이전 뉴런의 전기적 자극을 입력받는 수용체

- **Nucleus & Cell Body:**

수용된 자극을 바탕으로 다음 뉴런에게 전달한 자극을 생성

- **Axon:**

Cell body로부터 생성된 자극을 발생(Spike)시키고 다음 뉴런에게 전달

뇌가 처리하는 복잡한 인지적 작용들은 모두 간단한 계산의 뉴런들의 연결을 통해 이루어짐

# Awesome Characteristics of Neuron

- 인간의 뇌에 있는 뉴런의 작동 및 계산 방식은 컴퓨터 프로그램 등의 방식에 비해 다양한 장점이 있음
- 1) 높은 효율/효과의 계산 속도와 2) 모듈 단위의 적응(Adaptation)이 대표적인 장점으로 꼽힘

## <효율적인 계산 속도>

- 뉴런의 연결인 시냅스는 매우 적은 전력으로 신호를 주고 받음
- 인간의 뇌는 약  $10^{11}$  개의 뉴런을 가지고 있으며, 한 뉴런당 평균  $10^4$  개의 시냅스를 가지고 있음
- 이는 매우 높은 차원의 Neural Network 모델의 Weight 개수보다 훨씬 많은 개수임

Human's Synapse

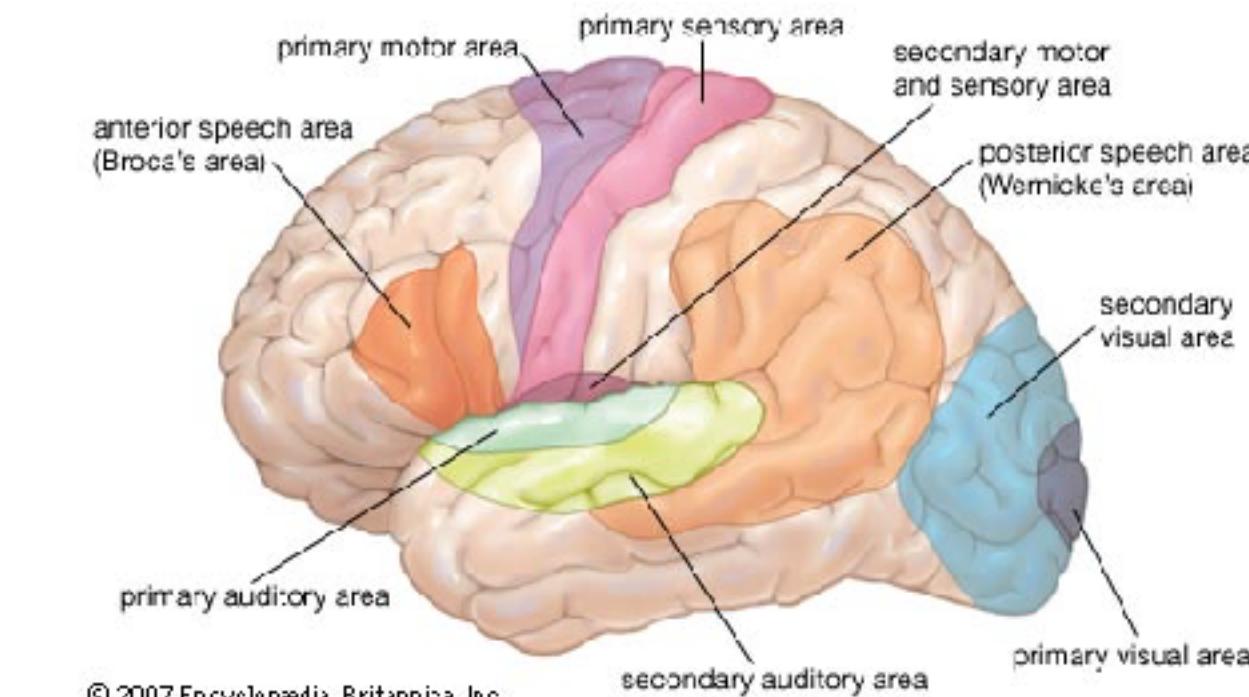
$10^{15}$

Neural Network's  
Synapse (Max)

$10^{11}$

## <모듈 단위의 적응(Adaptation)>

- Brain Learning Process Experiment
  - 대뇌 피질의 시각을 담당하는 부분(Primary visual area)가 손상되어 시신경(Optic nerve)를 Auditory cortex에 연결
  - 뉴런의 적응 이 후, 시각 자극에 대해 기존 Auditory cortex 부분이 새롭게 시각 자극에 대한 반응을 학습함



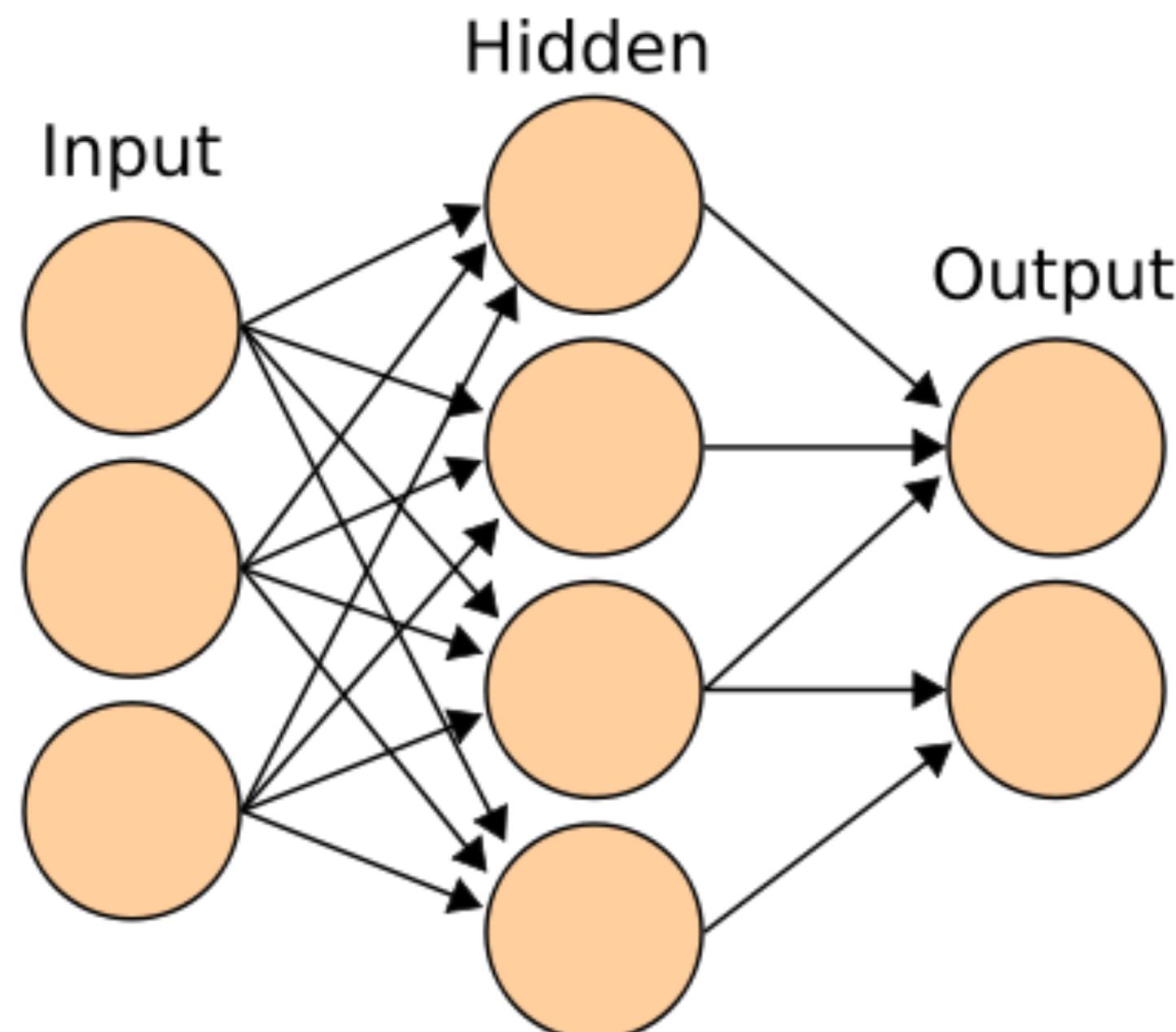
© 2007 Encyclopædia Britannica, Inc.

# Concept of Neural Network

## □ Neural Network (인공신경망)

- 신경세포의 간단하고 효과적인 처리 방식에 착안해 구현된 머신러닝 알고리즘의 한 종류
- 다량의 뉴런(Neuron, Unit)들이 층(Layer)으로 연결되어 간단한 계산과 연결을 통해 복잡한 문제를 해결함

<Neural Network Example>

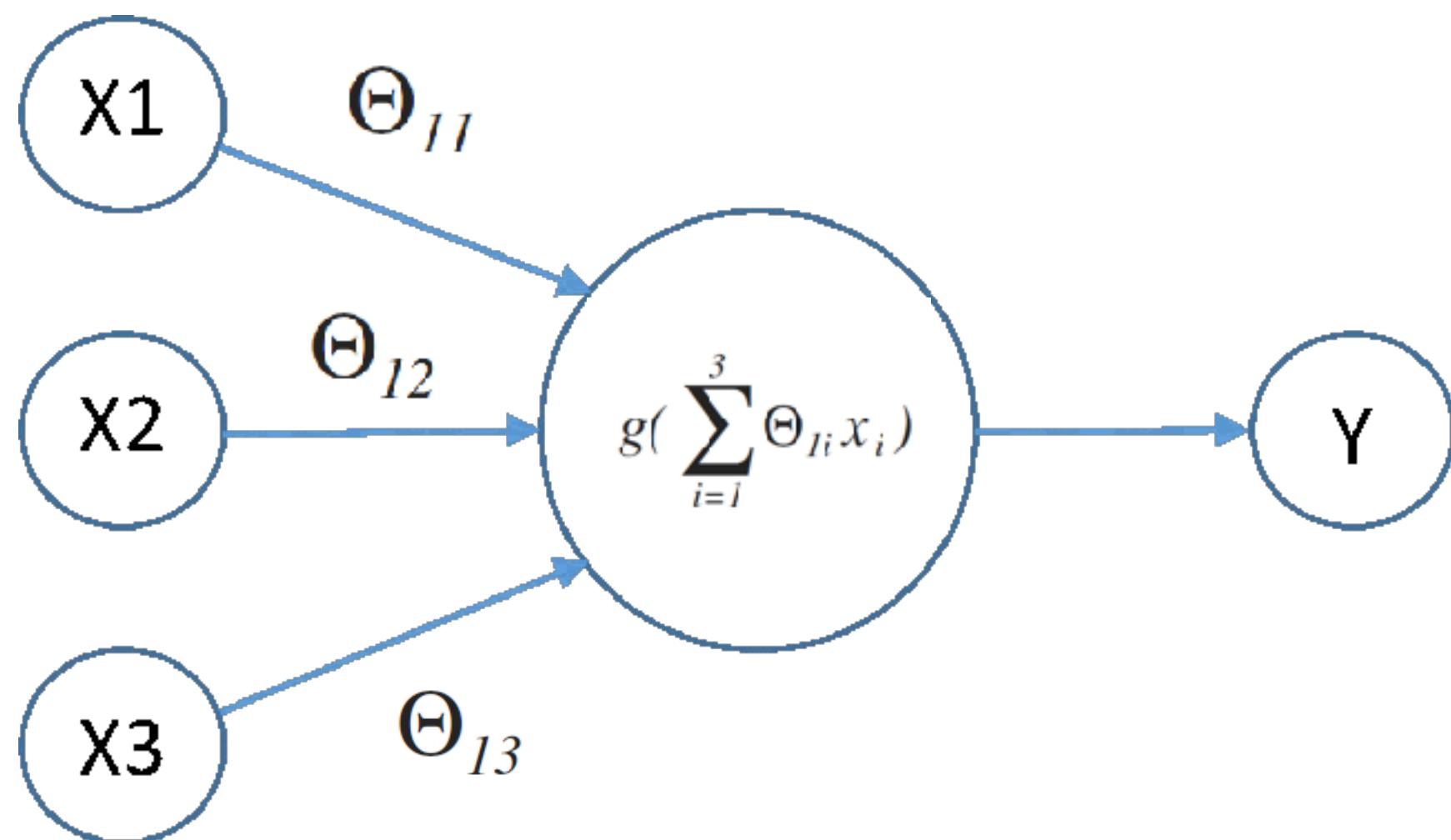


- **Neuron:** 각 뉴런이 **계산한 결과** 데이터 (Value)를 의미
- **Synapse :** 뉴런 결과값 사이의 **weight** 값을 의미
- **Input Layer:**  
초기 입력값을 받는 가장 첫번째 Layer ( $\# \text{ of Hidden} = 1$ )
- **Hidden Layer:**  
중간 단계의 모든 Layer를 의미함 ( $\# \text{ of Hidden} \geq 1$ )
- **Output Layer:**  
가장 마지막 단계의 Layer로 모델의 출력값을 계산 ( $\# \text{ of Output} = 1$ )

# Neuron in Neural Network

- 각 뉴런의 계산 과정은 1) 연결된 이전 뉴런과의 Linear Combination, 2) Activation 두 가지로 진행됨

<뉴런 기본 계산 프로세스>



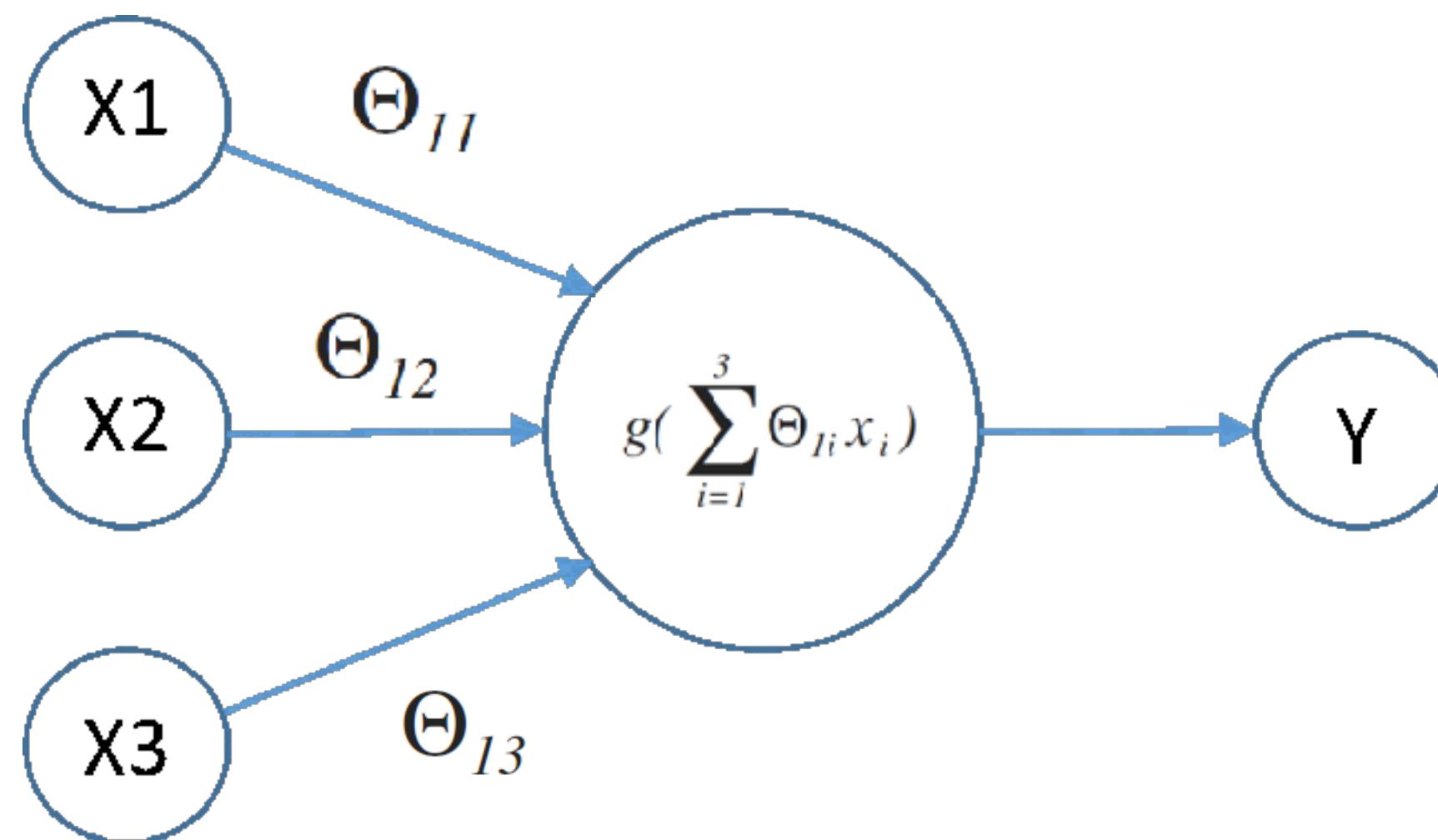
$\Theta_{13}$  (주의) 아래 첨자 숫자의 순서는 반대입니다!

(앞쪽) 3번째 뉴런 → (뒤쪽) 1번째 뉴런  
으로 연결된 weight

# Neuron in Neural Network

- 각 뉴런의 계산 과정은 1) 연결된 이전 뉴런과의 Linear Combination, 2) Activation 두 가지로 진행됨

<뉴런 기본 계산 프로세스>



## 1) 연결된 뉴런들의 Linear Combination

- 이전 Layer의 연결된 뉴런들의 출력값과 연결 Weight들 사이의 Linear Combination

$$\Theta_{11}x_1 + \Theta_{12}x_2 + \Theta_{13}x_3$$

$\Theta_{13}$

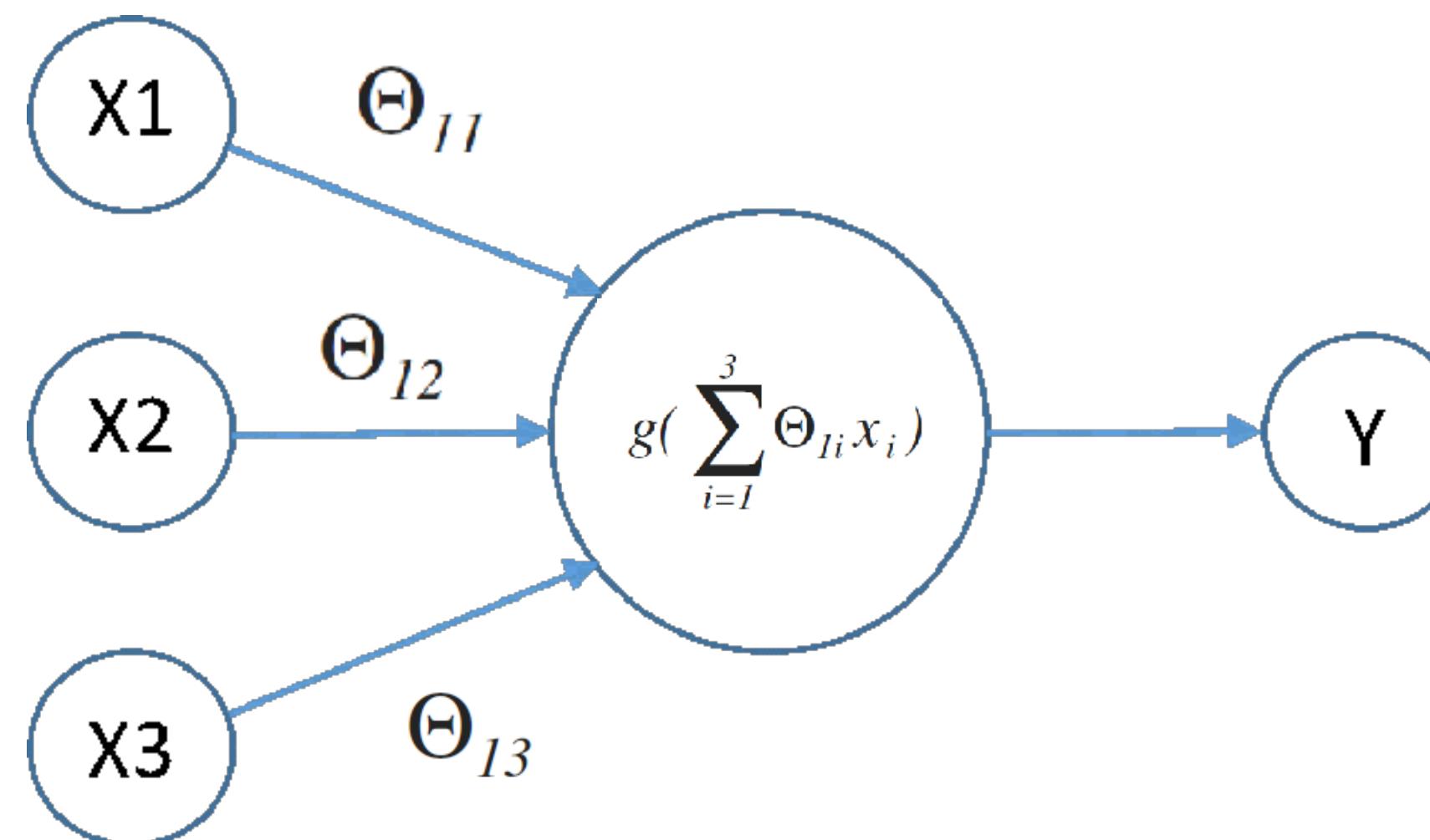
(주의) 아래 첨자 숫자의 순서는 반대입니다!

(앞쪽) 3번째 뉴런 → (뒤쪽) 1번째 뉴런  
으로 연결된 weight

# Neuron in Neural Network

▣ 각 뉴런의 계산 과정은 1) 연결된 이전 뉴런과의 Linear Combination, 2) Activation 두 가지로 진행됨

<뉴런 기본 계산 프로세스>



$\Theta_{13}$

(주의) 아래 첨자 숫자의 순서는 반대입니다!

(앞쪽) 3번째 뉴런 → (뒤쪽) 1번째 뉴런  
으로 연결된 weight

## 1) 연결된 뉴런들의 Linear Combination

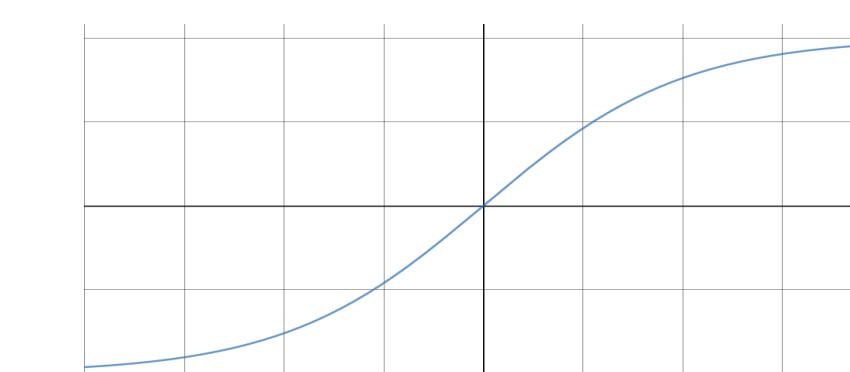
- 이전 Layer의 연결된 뉴런들의 출력값과 연결 Weight들 사이의 Linear Combination

$$\Theta_{11}x_1 + \Theta_{12}x_2 + \Theta_{13}x_3$$

## 2) Activation

- 연결된 뉴런들과의 Linear Combination 값을 Non-linear Function을 통해 Activation
- Activation Function:
  - Identify
  - Sigmoid
  - Hyperbolic Tangent (tanh)
  - Rectified Linear Unit (ReLU)

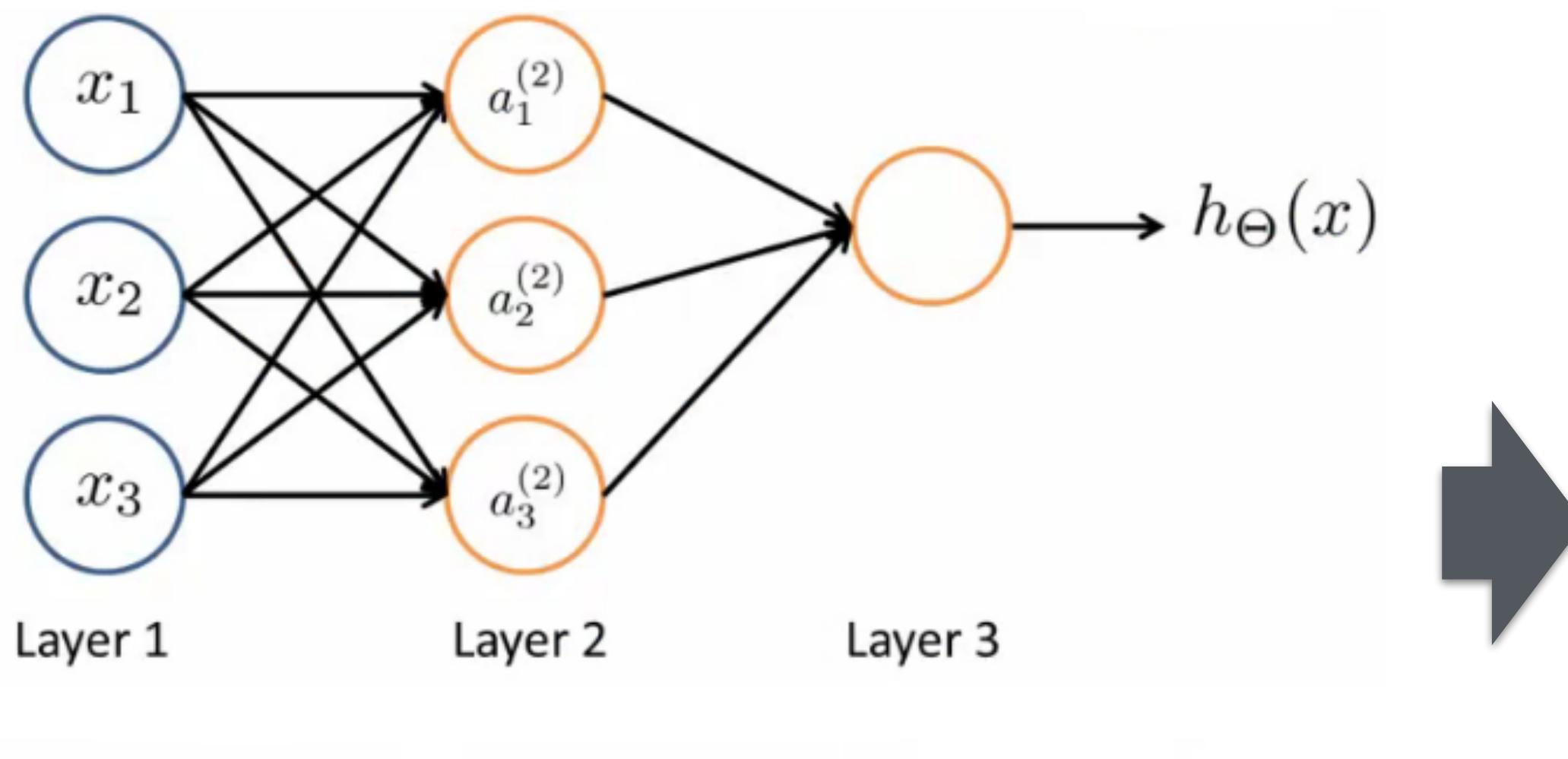
$$g\left(\sum_{i=1}^3 \Theta_{1i} x_i\right)$$



# Forward Propagation of Neural Network

- **Forward Propagation:** Input Layer에서 시작하여 Output Layer의 출력값을 순차적으로 얻어가는 과정을 의미함
- $a_i^{(j)}$  : activation of unit  $i$  in layer  $j$
- $\Theta^{(j)}$  : matrix of parameters controlling the function mapping from layer  $j$  to layer  $j + 1$

## <Forward Propagation of Neural Network>



$$\begin{aligned}a_1^{(2)} &= g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3) \\a_2^{(2)} &= g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3) \\a_3^{(2)} &= g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3) \\h_{\Theta}(x) &= a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})\end{aligned}$$

## Forward Propagation in Vectorized Implementation

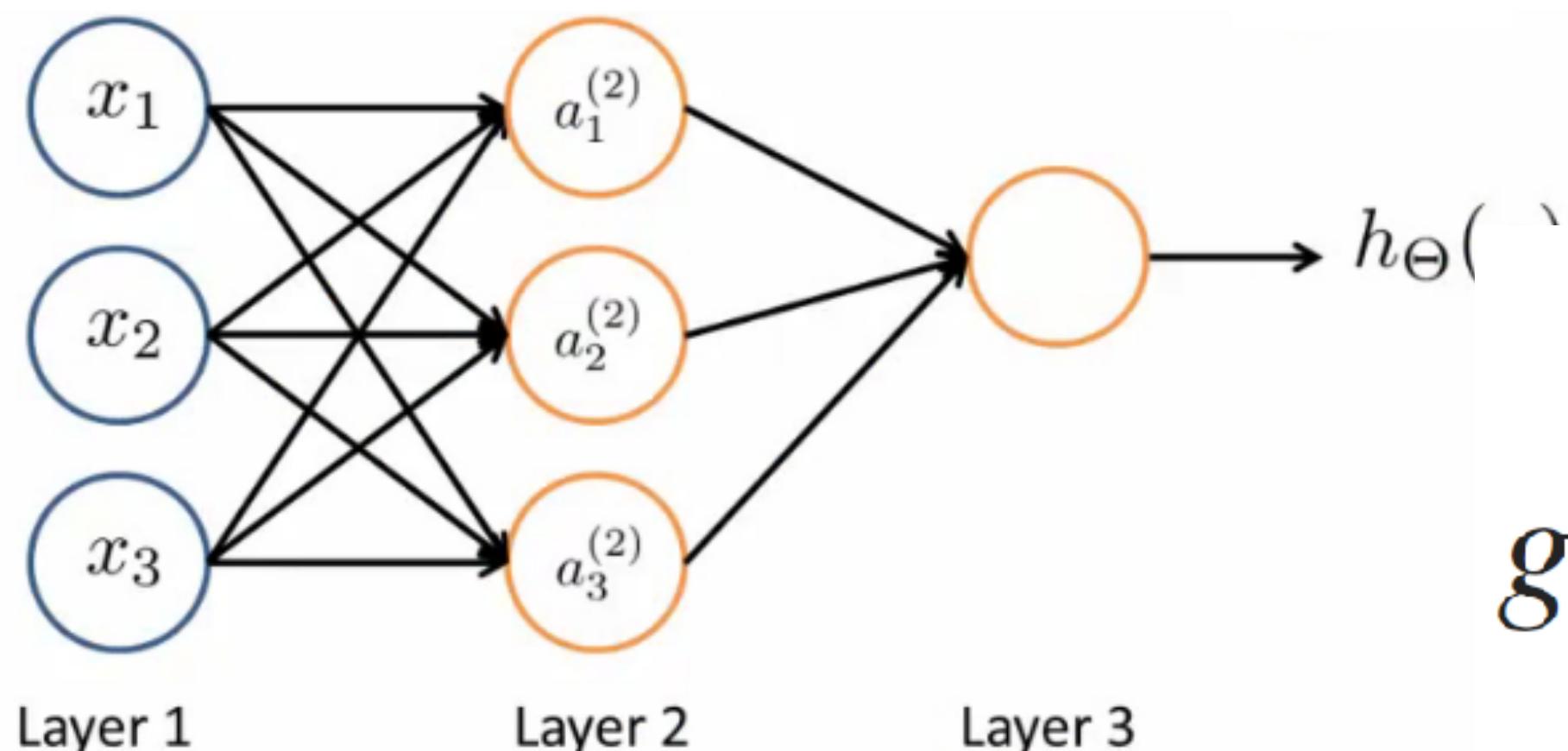
- 왼쪽 그림의 Forward Propagation의 표현 값들은 **Vectorization**을 통해 행렬의 형태로 간단하게 표현 가능함

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix} \quad \begin{aligned}z^{(2)} &= \Theta^{(1)}x \\a^{(2)} &= g(z^{(2)}) \\h_{\Theta}(x) &= a^{(3)} = g(z^{(3)})\end{aligned}$$

# Forward Propagation of Neural Network

- **Forward Propagation:** Input Layer에서 시작하여 Output Layer의 출력값을 순차적으로 얻어가는 과정을 의미함
- $a_i^{(j)}$  : activation of unit  $i$  in layer  $j$
- $\Theta^{(j)}$  : matrix of parameters controlling the function mapping from layer  $j$  to layer  $j + 1$

<Forward Propagation of Neural Network>



Forward Propagation in Vectorized Implementation

$$g\left(\begin{bmatrix} \Theta_{11} & \Theta_{12} & \Theta_{13} \\ \Theta_{21} & \Theta_{22} & \Theta_{23} \\ \Theta_{31} & \Theta_{32} & \Theta_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}\right) = \begin{bmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{bmatrix}$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

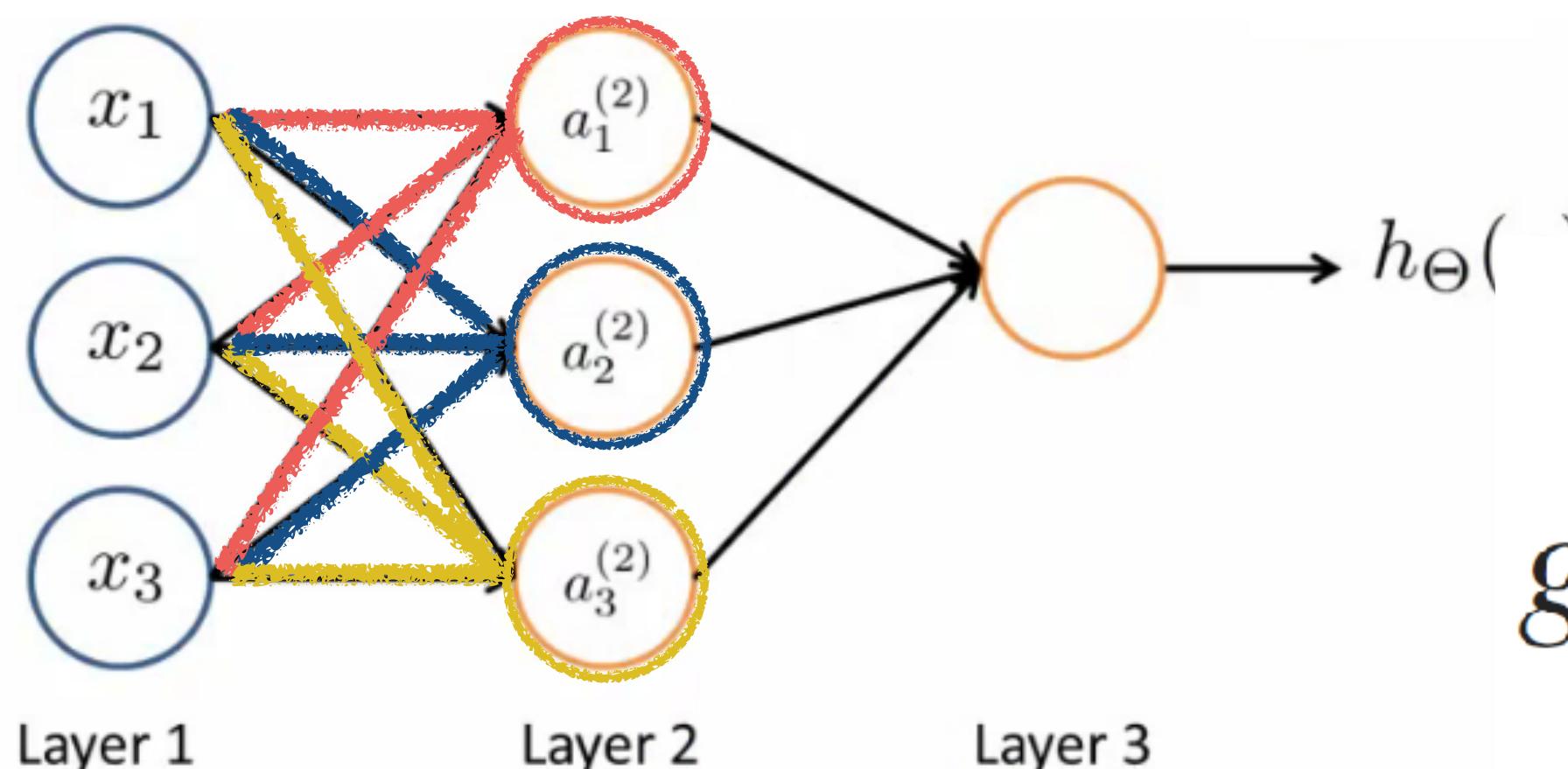
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_{\Theta}(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

# Forward Propagation of Neural Network

- **Forward Propagation:** Input Layer에서 시작하여 Output Layer의 출력값을 순차적으로 얻어가는 과정을 의미함
- $a_i^{(j)}$  : activation of unit  $i$  in layer  $j$
- $\Theta^{(j)}$  : matrix of parameters controlling the function mapping from layer  $j$  to layer  $j + 1$

<Forward Propagation of Neural Network>



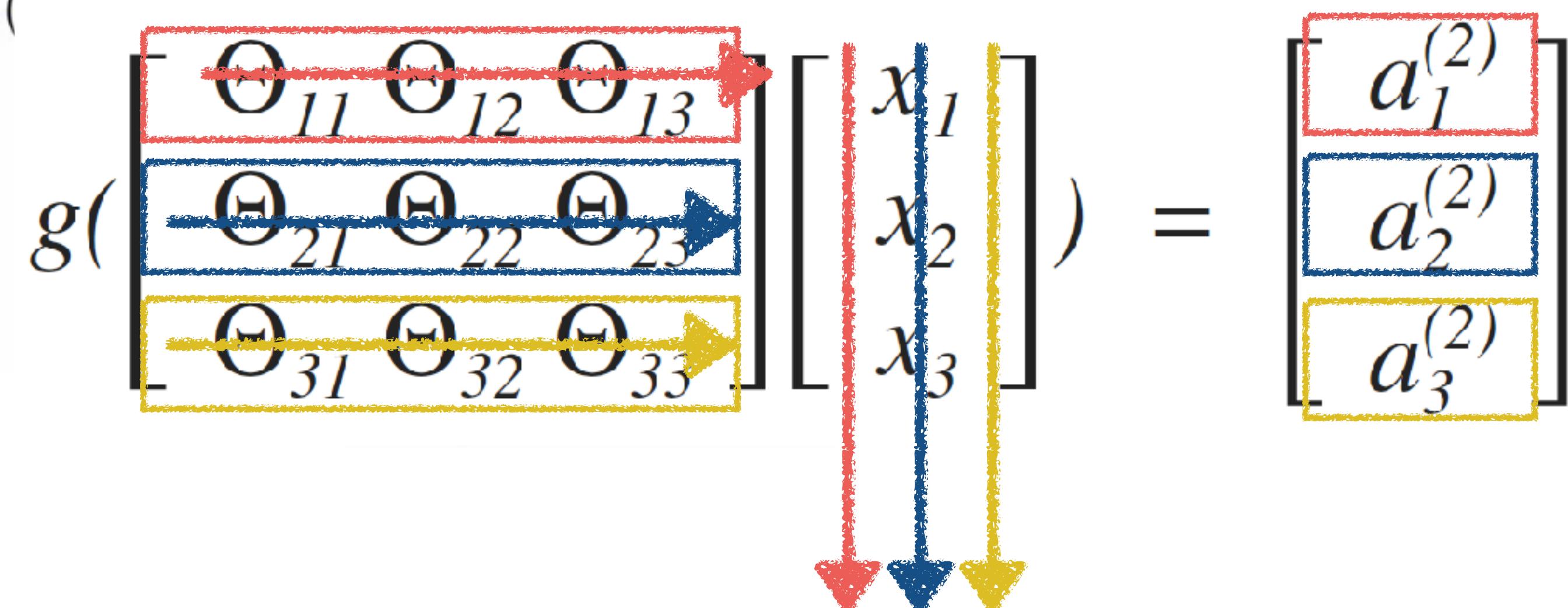
$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

Forward Propagation in Vectorized Implementation



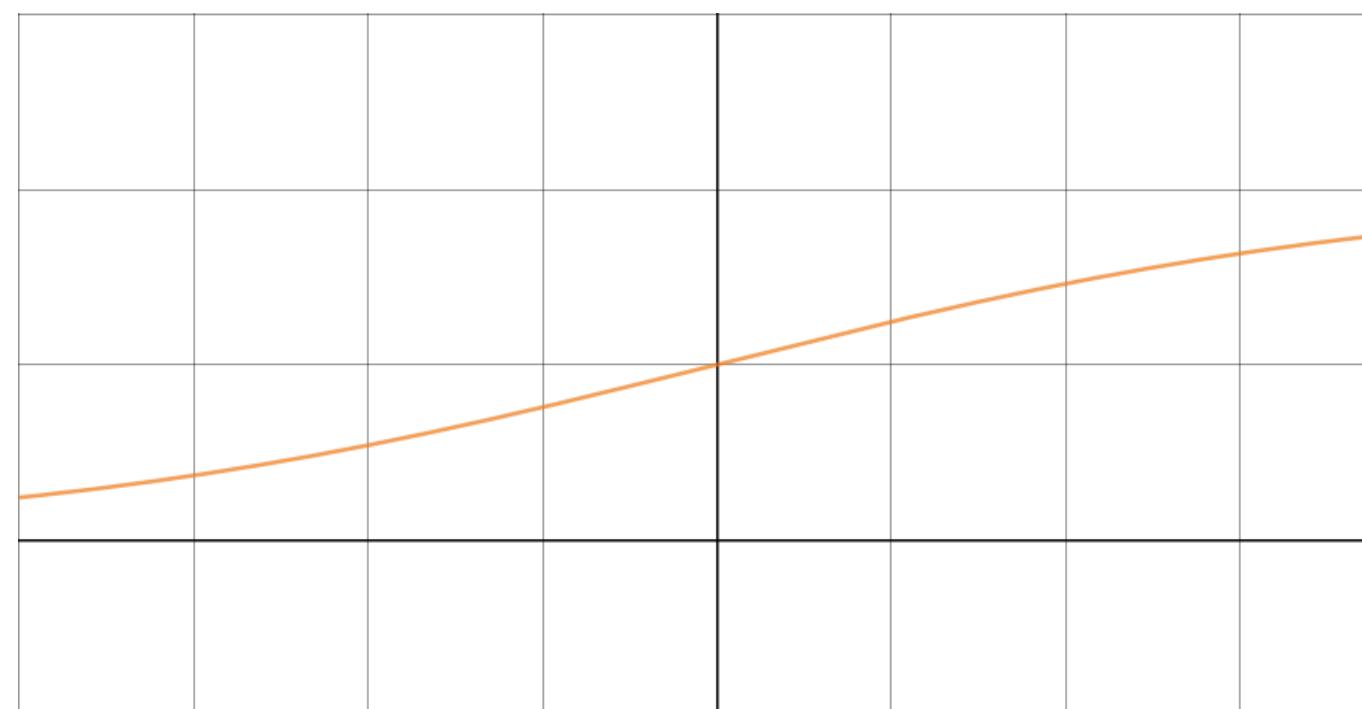
# Activation Function in Neuron (Cont'd)

□ 주로 Sigmoid, Hyperbolic Tangent (tanh), Rectified Linear Unit (ReLU) 등을 Activation Function으로 이용

<Sigmoid Function>

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

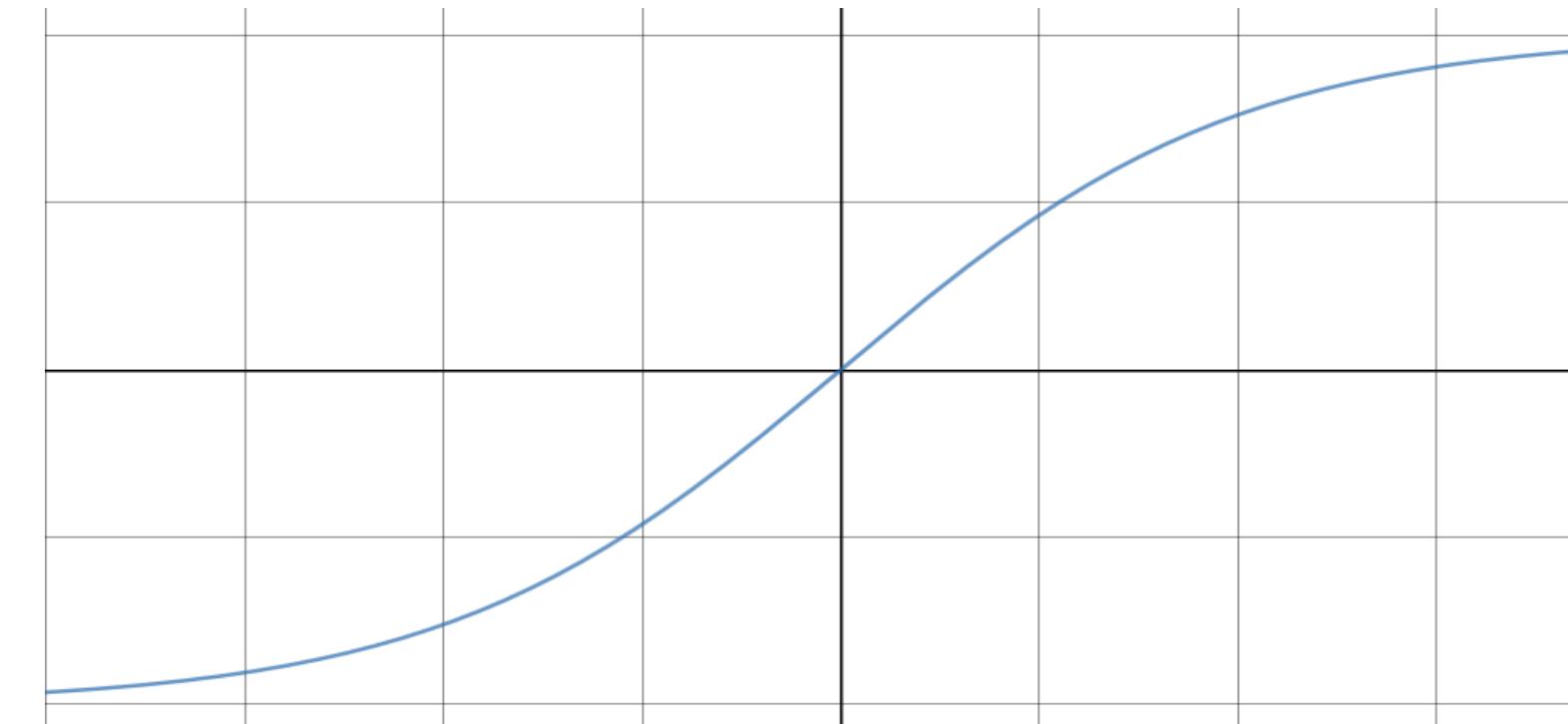
$$f'(x) = f(x)(1-f(x))$$



<tanh Function>

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

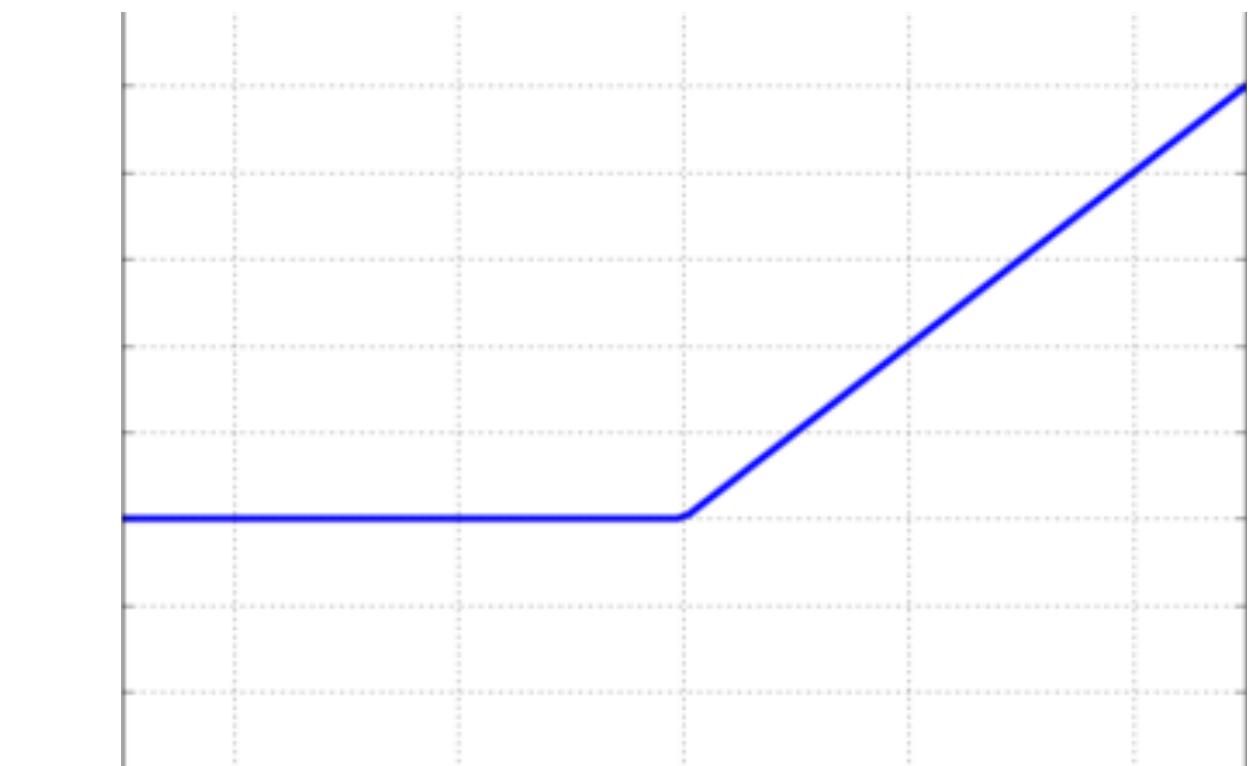
$$f'(x) = 1 - f(x)^2$$



<ReLU Function>

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

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

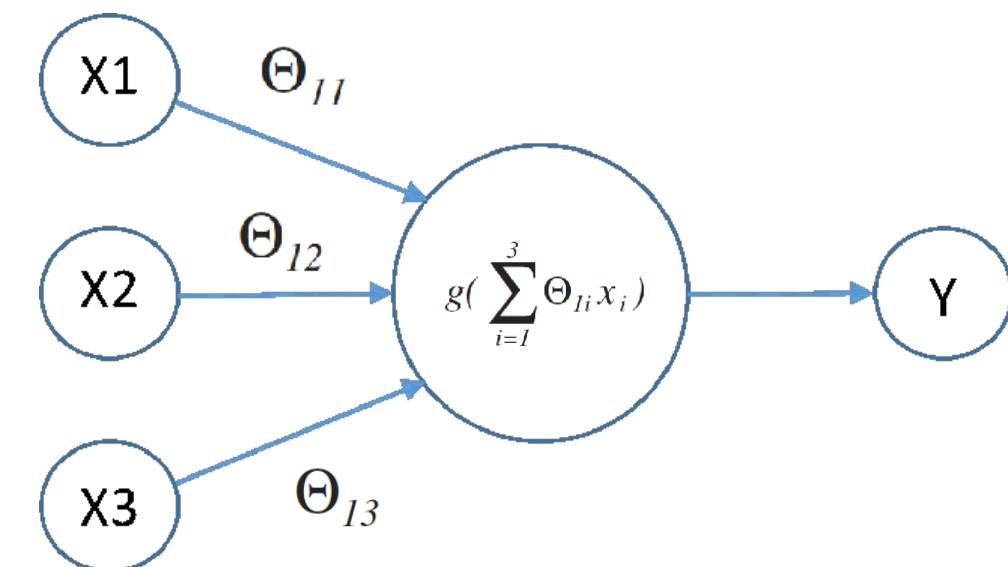


- 미분값이 변수의 변화를 왜곡하는 문제를 해결하기 위해 이용
- Computer Vision 분야에서 주로 이용하기 시작

# Activation Function in Neuron

## Activation Function

- 연결된 뉴런들의 Linear Combination 이 후, 합성하는 Non-linear Function
- 입력값들의 숨겨진 Non-linear feature를 발견하기 위해 사용
- Non-linear Function을 Activation Function으로 사용하지 않는다면, 여러 Layer를 사용하는 것이 무의미!



### Non-linearity (비선형성)

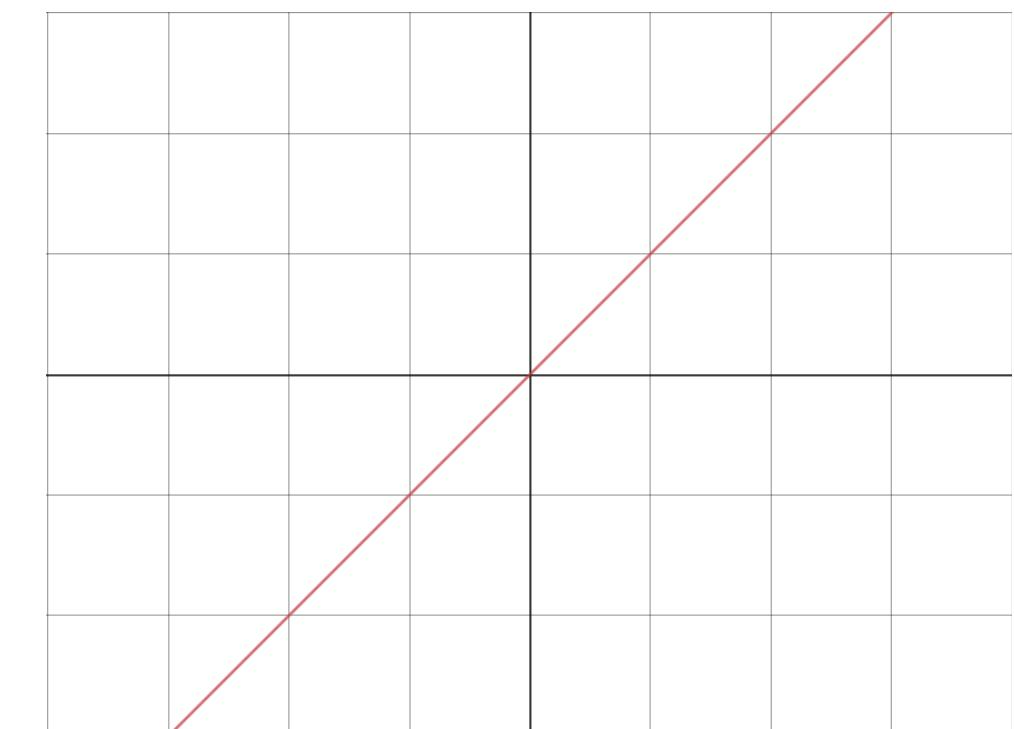
- 선형적이지 않은 함수
- Hidden Layer가 2개 이상일 경우 필수적인 조건
- Linear Function을 Activation으로 사용할 경우 Deep Neural Network의 의미가 사라짐

### Connectivity & Smooth

- Activation Function은 연속적이고 Smooth Curve를 가지는 것을 의미
- Backpropagation 학습 과정에서 미분값을 주로 사용하기 때문에 중요함

### <Identity Activation Function>

$$g(x) = x$$



- Derivative: 1
- Range: (-∞, ∞)

### Saturation (포화)

- Activation Function이 최대값과 최소값을 가지고 있는 것을 의미
- 출력값을 확률로 나타낼 경우, 특히 유용함
- 이미지 분야에서 ReLU 등의 사용을 통한 성능 향상으로 필수적인 조건은 아님

### Monotonicity

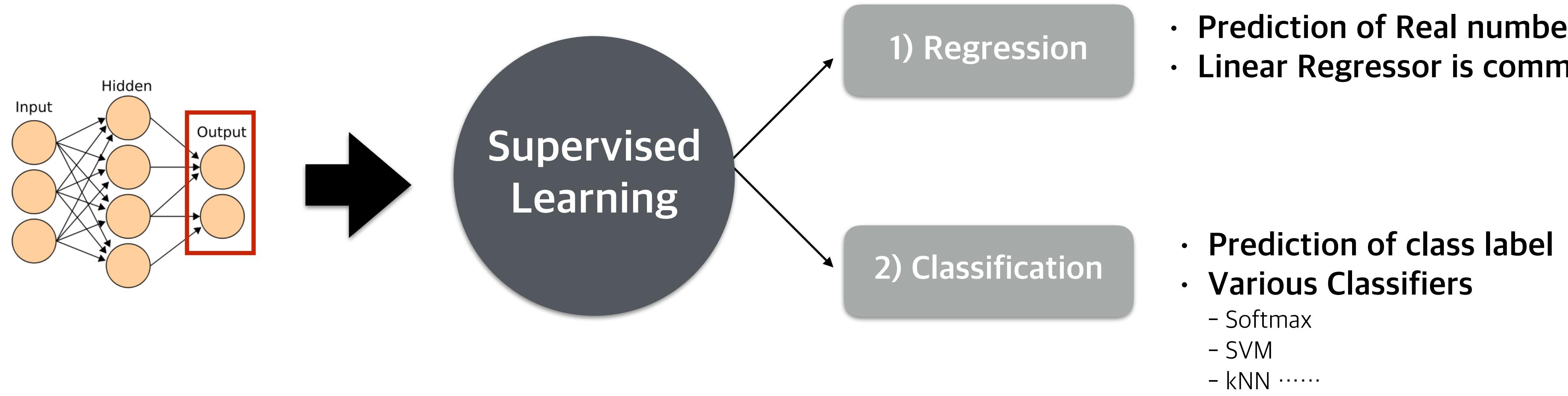
- Activation Function이 단조 증가하거나 단조 감소하는 것을 의미
- 단조성이 없을 경우, Error Surface에 다양한 Local Minimum 값이 추가될 수 있음

$$g\left(\sum \Theta^{(2)} g\left(\sum \Theta^{(1)} x^{(1)}\right)\right) = \sum \Theta x^{(1)}$$

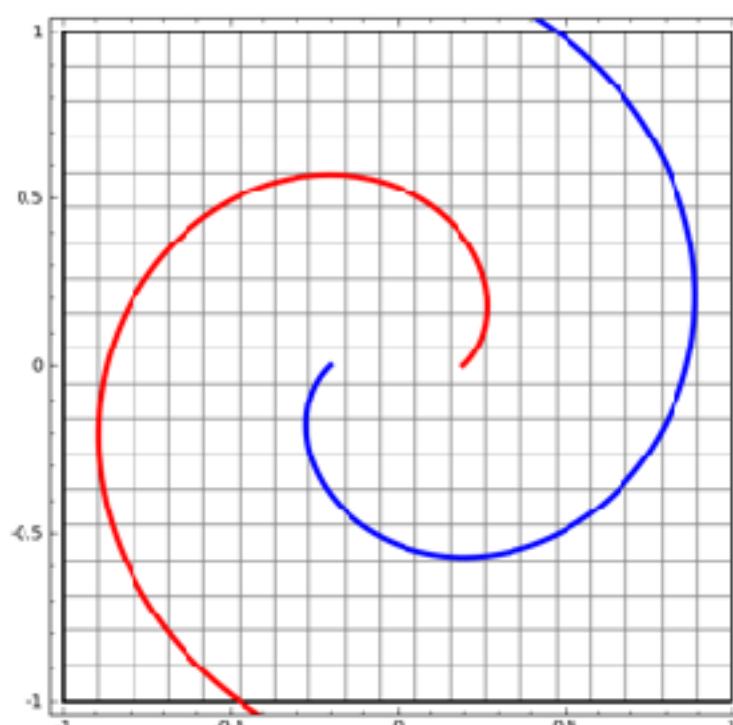
Identity를 Activation으로 사용할 경우, 다층 Layer 무의미

# Function of Last Layer

- Neural Network의 마지막 Layer는 모델의 목적에 맞게 설계되어 기능을 수행함
- Supervised Learning의 경우, 1) Regression, 2) Classification 두 가지 기능을 수행



## <Hidden Feature Extraction>

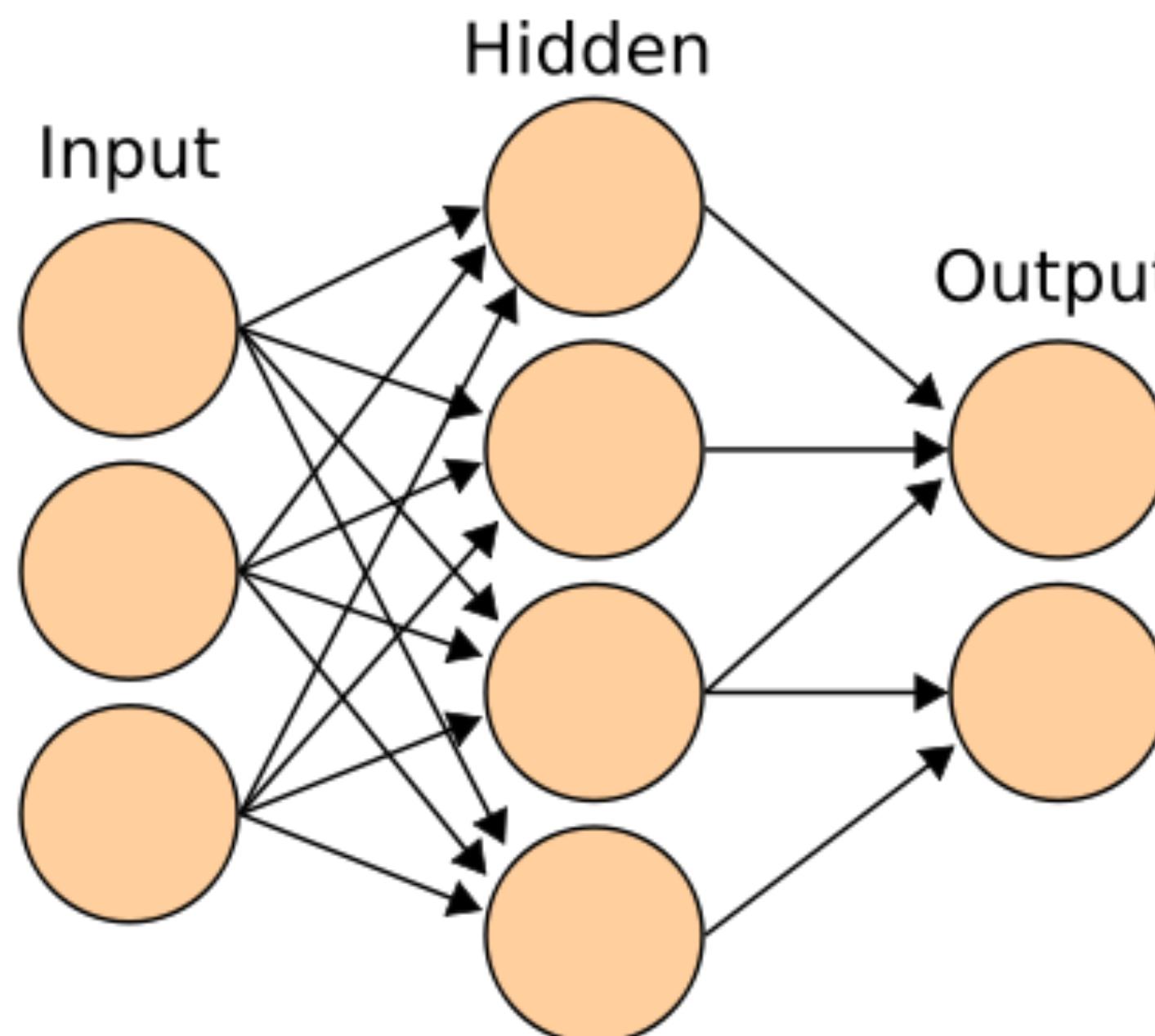


- Output Layer가 Linear Regressor, Logistic, SVM 등일 경우  
Hidden Layer는 Input data의 Non-linear hidden feature를  
Linear feature로 바꾸어주는 Kernel 역할을 함

# Review of NN and Forward Propagation

- Neural Network는 뉴런의 간단한 계산과 뉴런 사이의 연결을 통해 Input에 대한 Output을 계산하는 방식

## <Neural Network Example>



- **Input Layer:**

초기 입력값을 받는 가장 첫번째 Layer ( $\# \text{ of Hidden} = 1$ )

- **Hidden Layer:**

중간 단계의 모든 Layer를 의미함 ( $\# \text{ of Hidden} \geq 1$ )

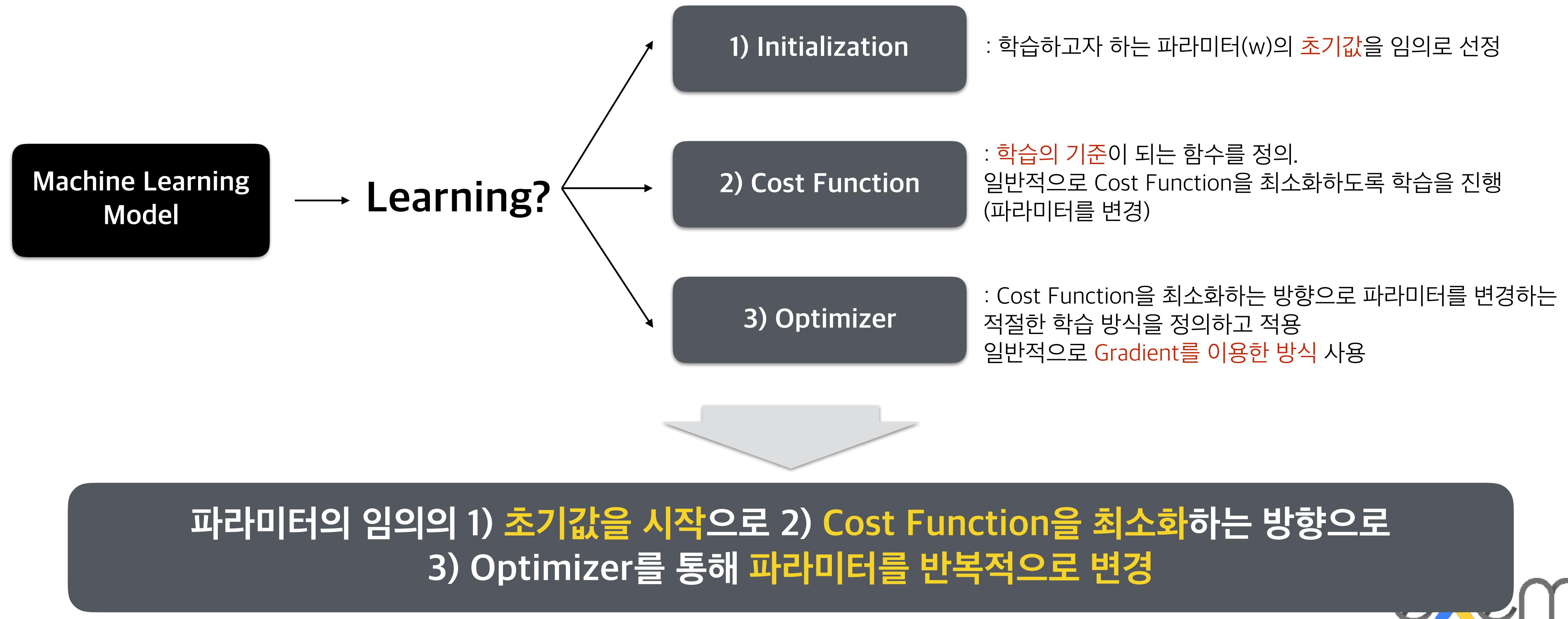
- **Output Layer:**

가장 마지막 단계의 Layer로 모델의 출력값을 계산 ( $\# \text{ of Output} = 1$ )

How can NN be trained for desirable output with each input?

# Neural Network Learning Process

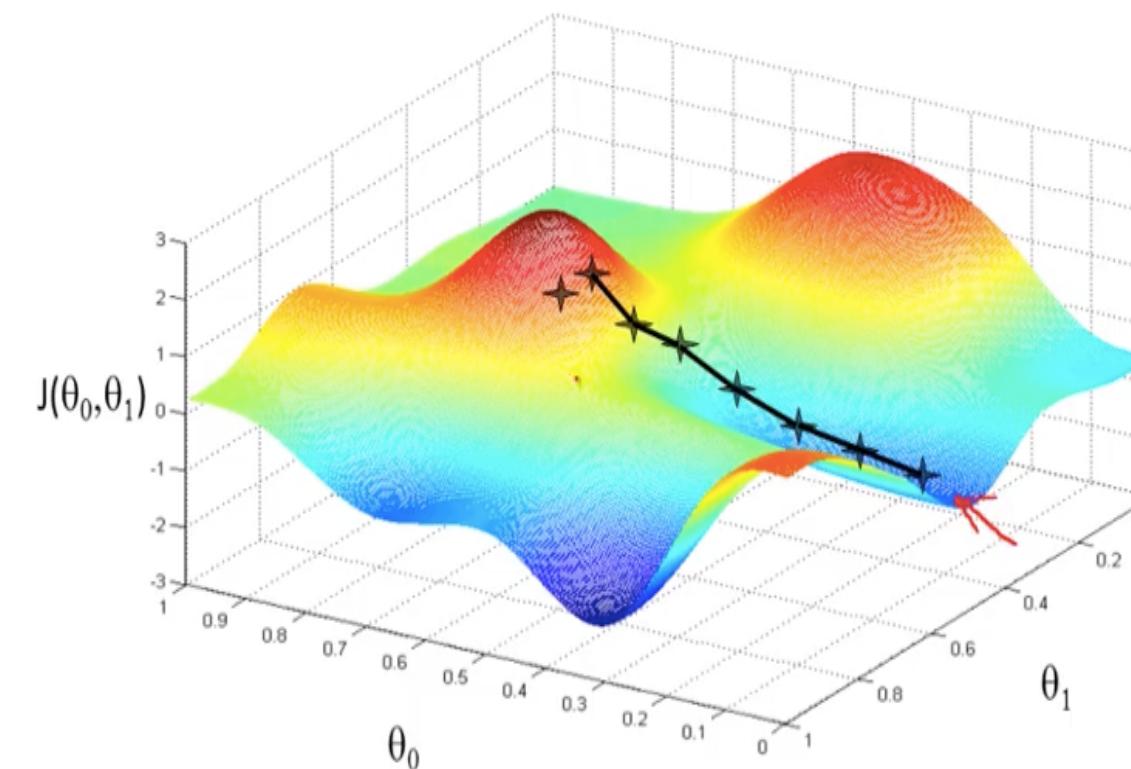
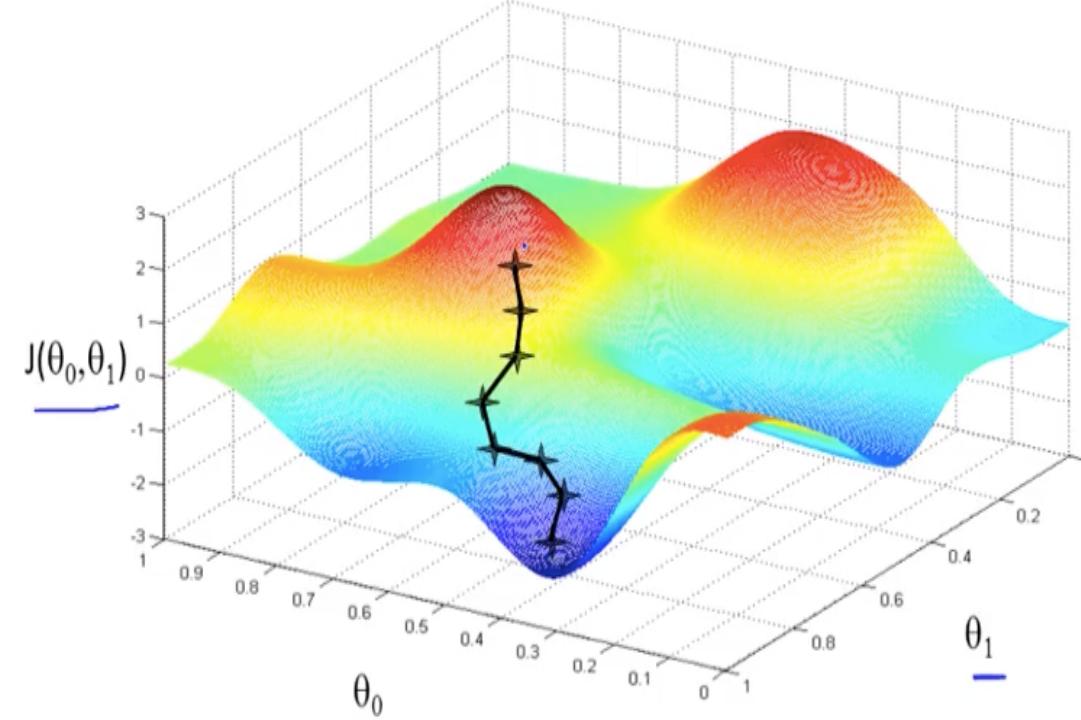
- ❑ 머신러닝 모델이 원하는 Output을 나타내도록 학습(learning)하기 위해서는 1) Initialization, 2) Cost Function, 3) Optimizer의 정의가 필요함



# Initialization of Neural Network

- ❑ 학습을 진행하기 전,  $\theta$  (weights)의 초기 값을 설정해야함
- ❑ 초기값 설정은 1) Training Speed, 2) 실제 Learning 가능 여부를 결정하는 중요한 역할을 함

<초기값에 따른 학습 결과 차이>



## Random Initialization

- Same Initialization (**Don't do**)
- Small number random Initialization
- Calibrating with variance  $1/\sqrt{n}$  Initialization
- **Xavier Initialization** (**Experimentally powerful**)

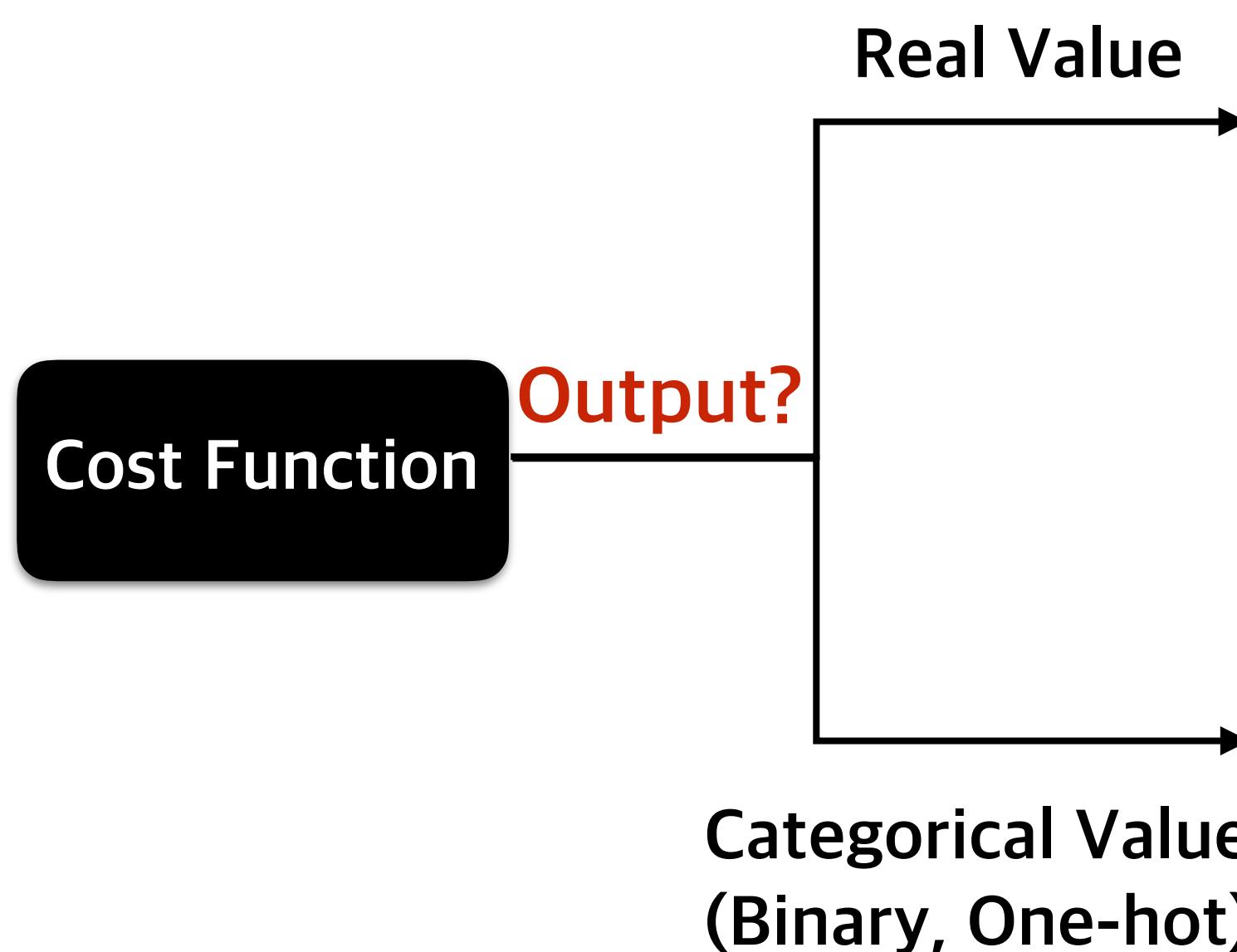
각각의 파라미터를 일정 범위 안의 임의의 값으로 초기값으로 Initialization하는 것이 공통적임

# Cost Functions

- Neural Network는 학습을 통해 성능을 향상시키며, 학습은 적절하게 **parameters(weights)**를 조절하는 것을 의미함
- 학습의 기준이 되는 Cost Function을 정의한 후, Cost Function 최소화를 위해 학습을 진행
- 일반적으로 용도에 따라 1) Least Square Method 또는 2) Cross-Entropy를 사용함

## Types of Cost Function

- 실제  $y$  값과 모델 출력값의 차이를 바탕으로 계산한 비용 함수



### <Least Square Method>

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

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

### <Cross-Entropy>

- 오분류에 대한 확률적인 비용을 의미
- Classifier (특히, softmax) 모델과 주로 사용
- $y$  값은 0 또는 1을 나타냄

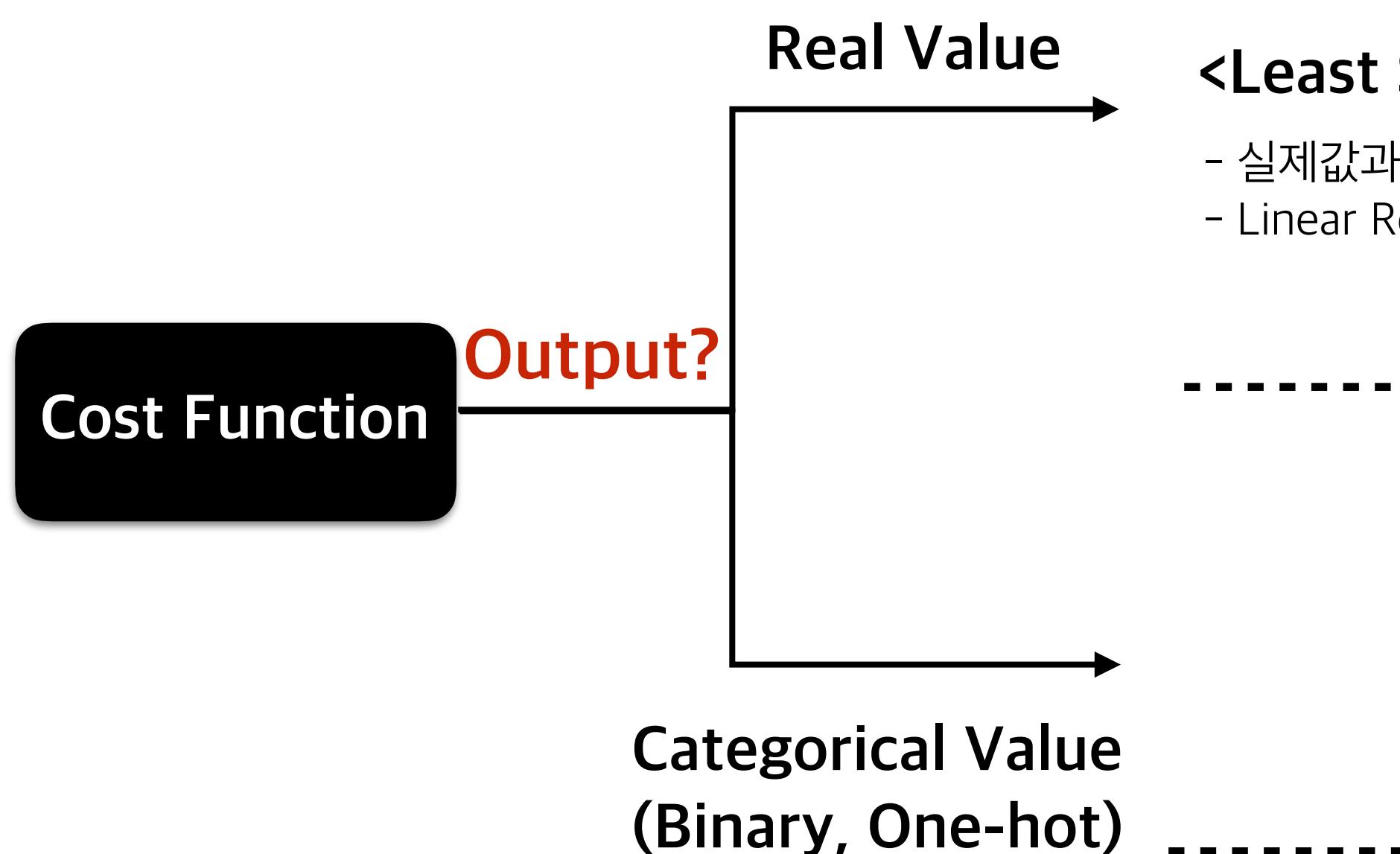
$$J(\theta) = -\frac{1}{m} \sum_i y^{(i)} \log h_{\theta}(x^{(i)})$$

# Cost Functions

- Neural Network는 학습을 통해 성능을 향상시키며, 학습은 적절하게 **parameters(weights)**를 조절하는 것을 의미함
- 학습의 기준이 되는 **Cost Function**을 정의한 후, **Cost Function** 최소화를 위해 학습을 진행
- 일반적으로 용도에 따라 1) Least Square Method 또는 2) Cross-Entropy를 사용함

## Types of Cost Function

- 실제  $y$  값과 모델 출력값의 차이를 바탕으로 계산한 비용 함수



### <Least Square Method>

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

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

모델 예측값과 실제값 사이의 차이

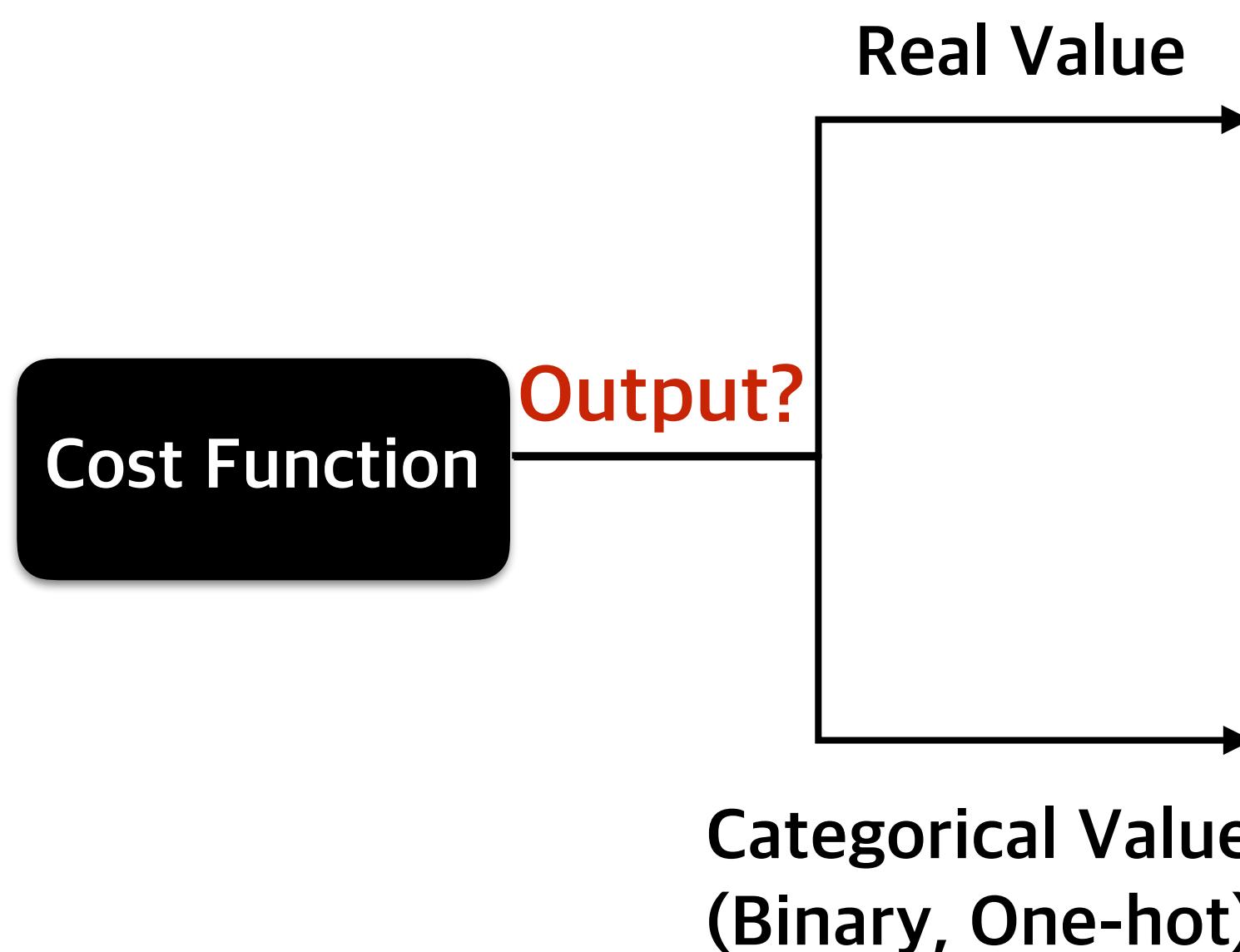
(오차 제곱 평균)

# Cost Functions

- Neural Network는 학습을 통해 성능을 향상시키며, 학습은 적절하게 **parameters(weights)**를 조절하는 것을 의미함
- 학습의 기준이 되는 Cost Function을 정의한 후, Cost Function 최소화를 위해 학습을 진행
- 일반적으로 용도에 따라 1) Least Square Method 또는 2) Cross-Entropy를 사용함

## Types of Cost Function

- 실제  $y$  값과 모델 출력값의 차이를 바탕으로 계산한 비용 함수



실제 Class 결과(0 또는 1)와 예측 결과의 차이에 대한 비용

Input 데이터가 각 Class에 속할 확률을 계산 (확률 분포의 차이를 계산)

### <Cross-Entropy>

- 오분류에 대한 확률적인 비용을 의미
- Classifier (특히, softmax) 모델과 주로 사용
- $y$  값은 0 또는 1을 나타냄

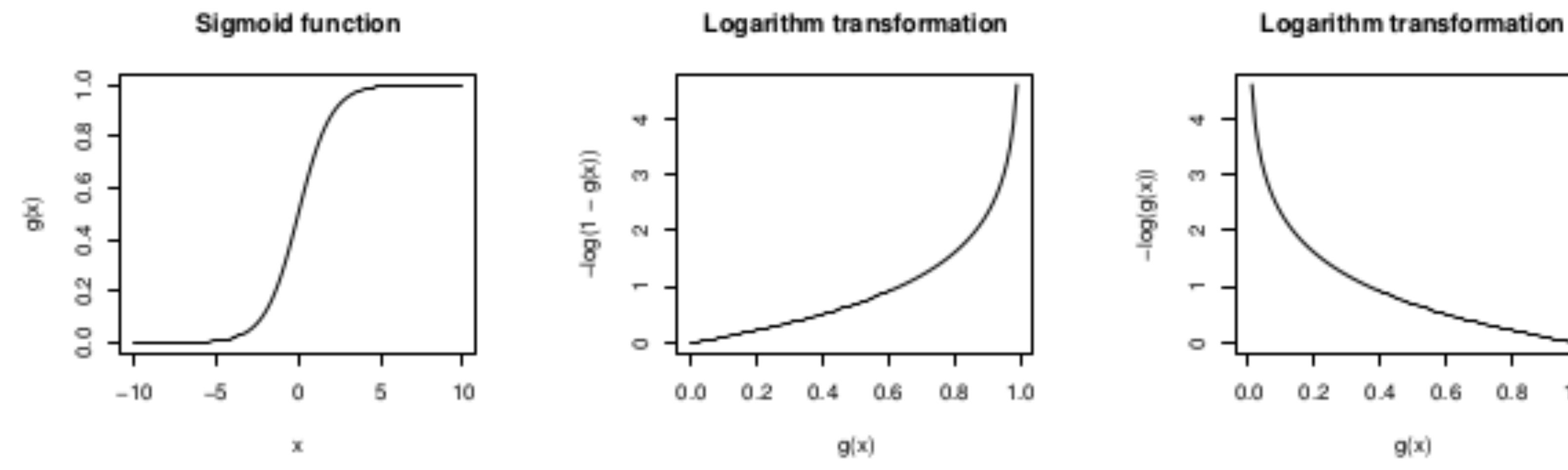
$$J(\theta) = -\frac{1}{m} \sum_i^m y^{(i)} \log h_{\theta}(x^{(i)})$$

# Cost Function and Optimizer

- 실제로 logistic regression의 파라미터는 반복적인(iterative) 방법을 통해 추정됨
- 분류 문제이므로 **Cross-entropy**를 비용 함수로 설정하고 **Gradient-based optimizer**를 통해 학습을 진행함

## <Cost Function>

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



(a) Sigmoid function.

(b) Cost for  $y = 0$ .

(c) Cost for  $y = 1$ .

Figure B.1: Logarithmic transformation of the sigmoid function.

# Learning Method of Neural Network

- Neural Network는 학습을 통해 성능을 향상시키며, 학습은 적절하게 **parameters(weights)**를 조절하는 것을 의미함
- 학습의 기준이 되는 **Cost Function**을 정의한 후, **Cost Function** 최소화를 위해 학습을 진행
- 대부분의 Deep Learning 모델은 **Gradient-based Learning**

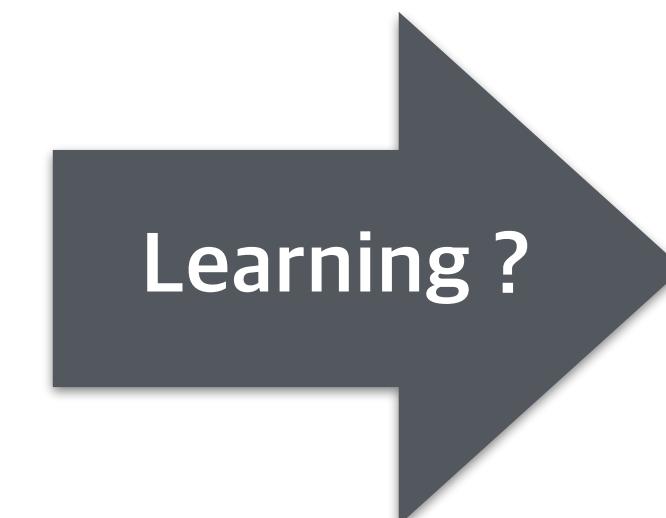
## Types of Cost Function

- (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

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



## Gradient-based Learning

### <Cross-Entropy>

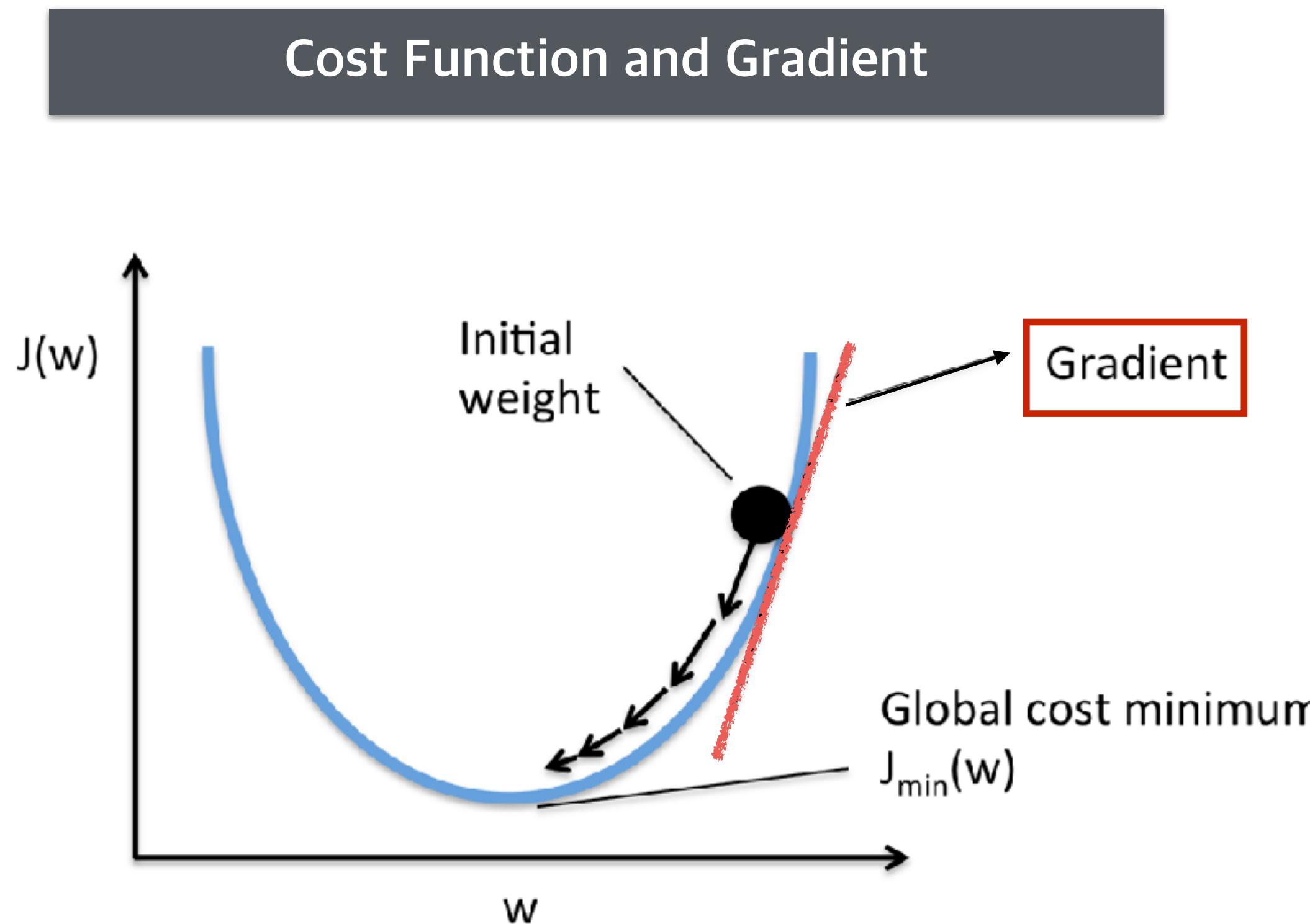
- 오분류에 대한 확률적인 비용을 의미
- Classifier (특히, softmax) 모델과 주로 사용

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

# Learning Method of Neural Network

## □ Neural Network 모델의 Optimizer들은 일반적으로 학습을 위해 Gradient를 이용함

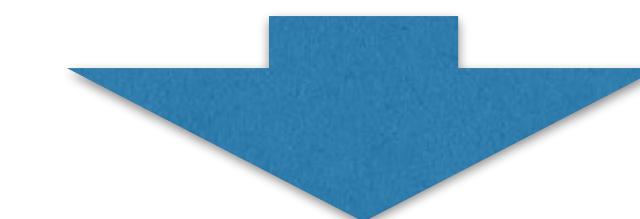
- 1) 특정 파라미터에 대한 Cost Function의 Gradient 계산
- 2) Cost Function이 작아지는 방향으로 파라미터를 업데이트



$$\frac{\partial J(\Theta)}{\partial \Theta}$$

: Parameter 변화에 따른 Cost 변화량

- Gradient  $> 0$  : 파라미터 값이 증가하면 비용 함수 증가
- Gradient  $< 0$  : 파라미터 값이 증가하면 비용 함수 감소



Gradient의 반대 방향으로 파라미터 업데이트

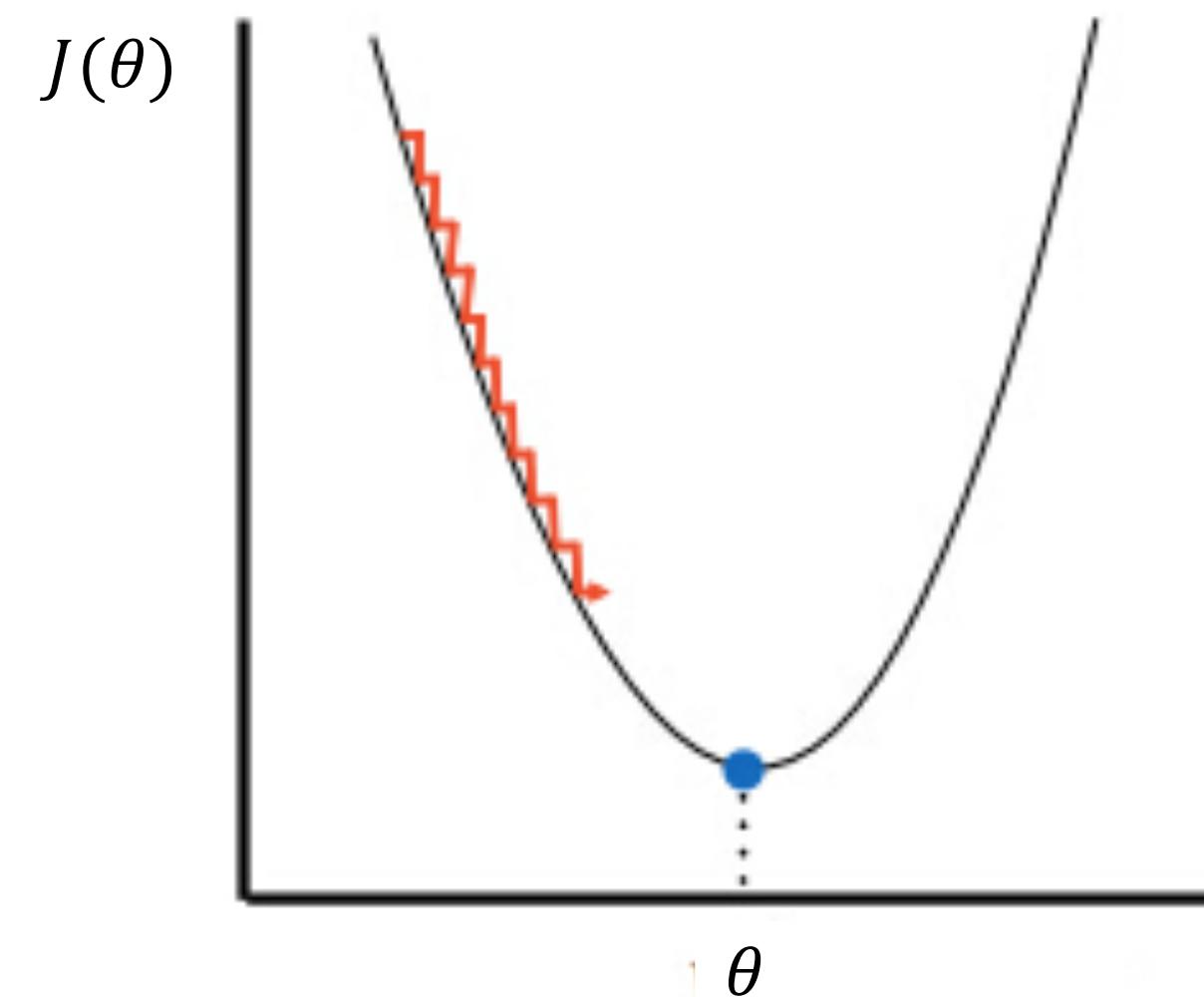
$$\theta_j := \theta_j - \underline{\alpha} \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$$

( $\alpha = \text{learning rate}$ )

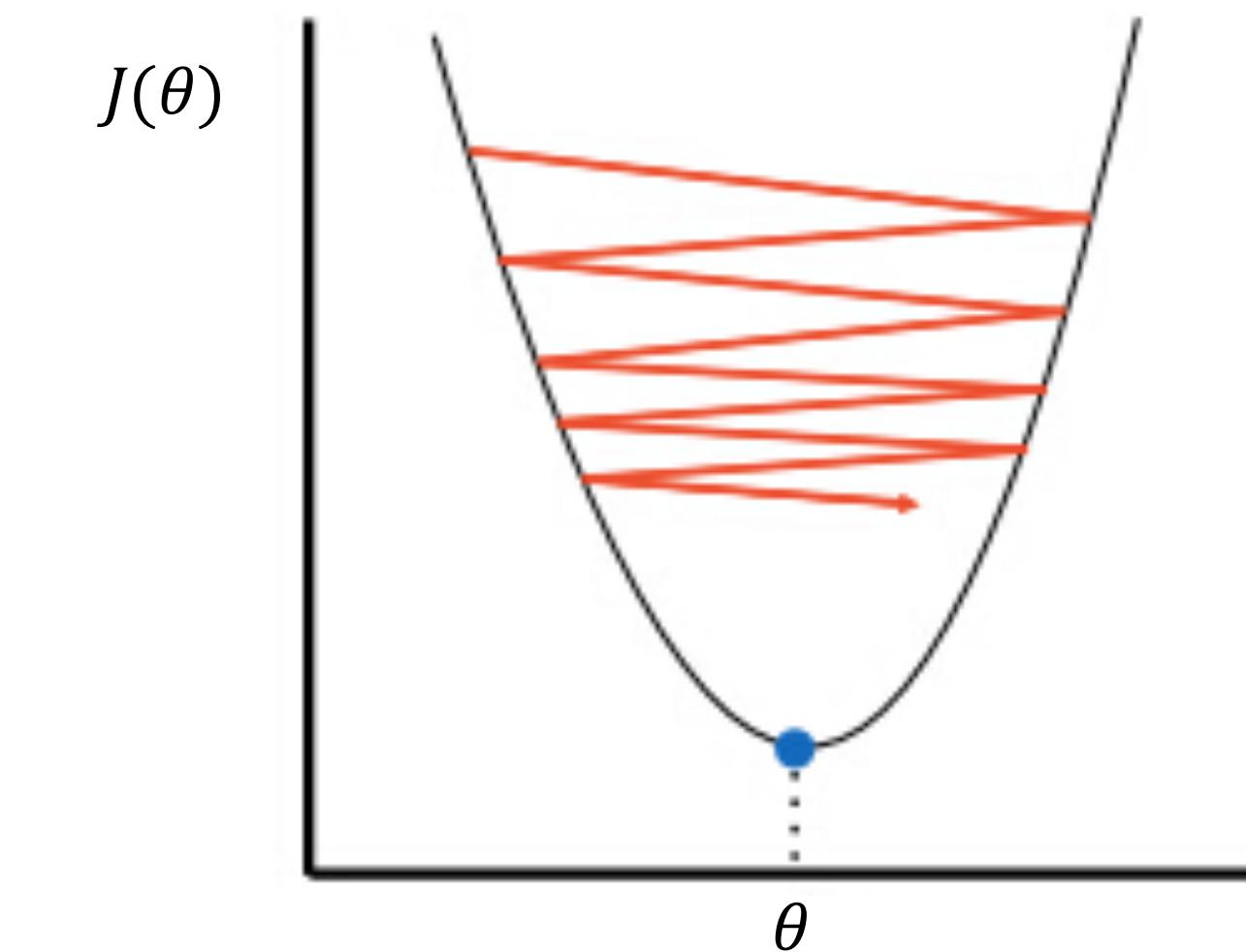
- Gradient Descent Method -

# Learning Process with Different Learning Rate

- ❑ 학습률 (Learning Rate)의 조절에 따라 학습의 결과 또는 수렴 여부가 크게 달라질 수 있음
- ❑ 데이터의 특징, 모델 파라미터의 수에 따라 적절한 크기의 Learning Rate을 결정하는 것이 중요!
  - Too small learning rate: 정교하지만 수렴 속도가 너무 느려, 학습이 제대로 되지 않을 수 있음
  - Too big learning rate: 학습이 빠르지만 수렴하지 않고 발산할 가능성이 있음



Too small: converge  
very slowly



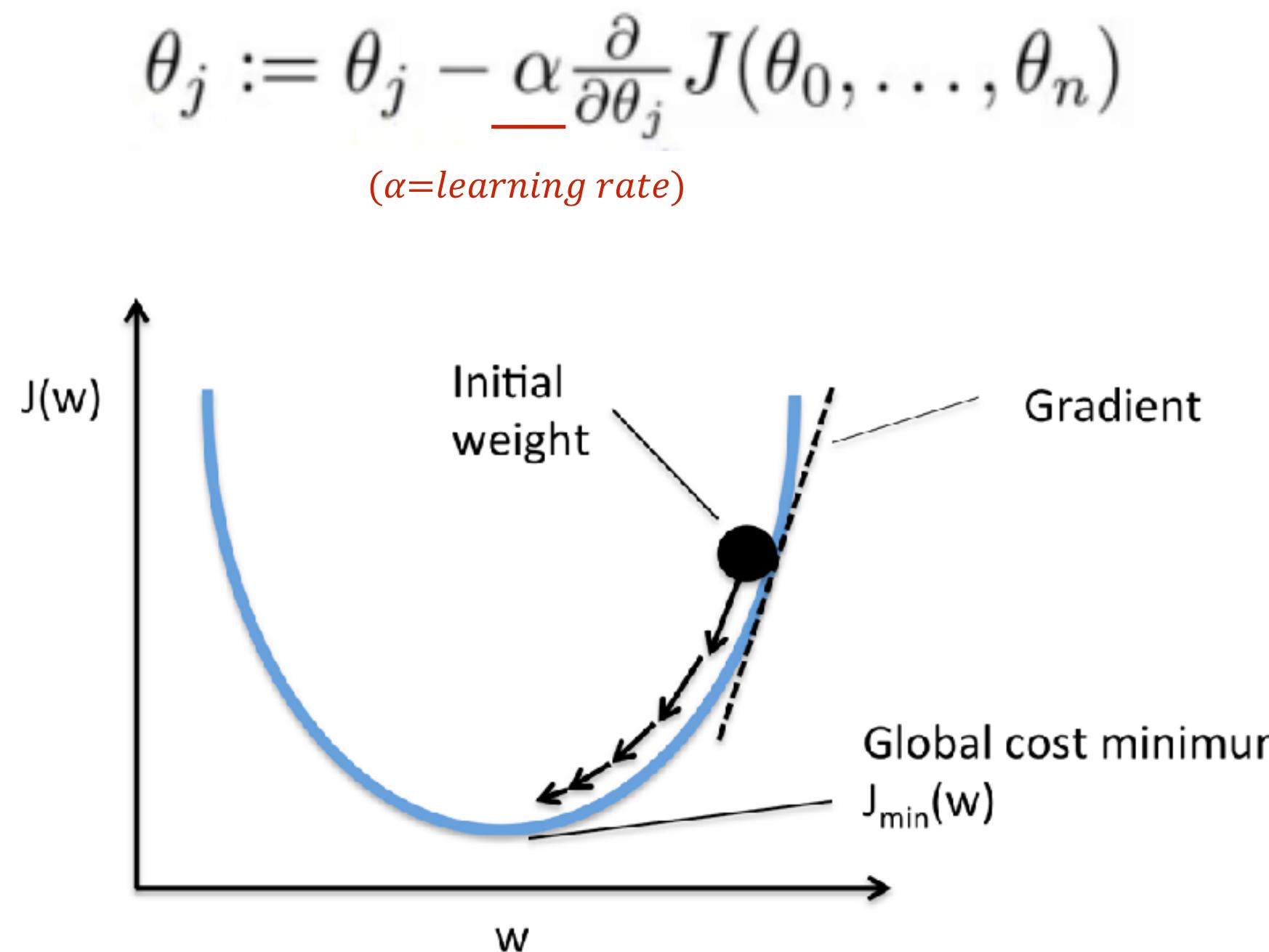
Too big: overshoot and  
even diverge

# Gradient Descent Learning Algorithm

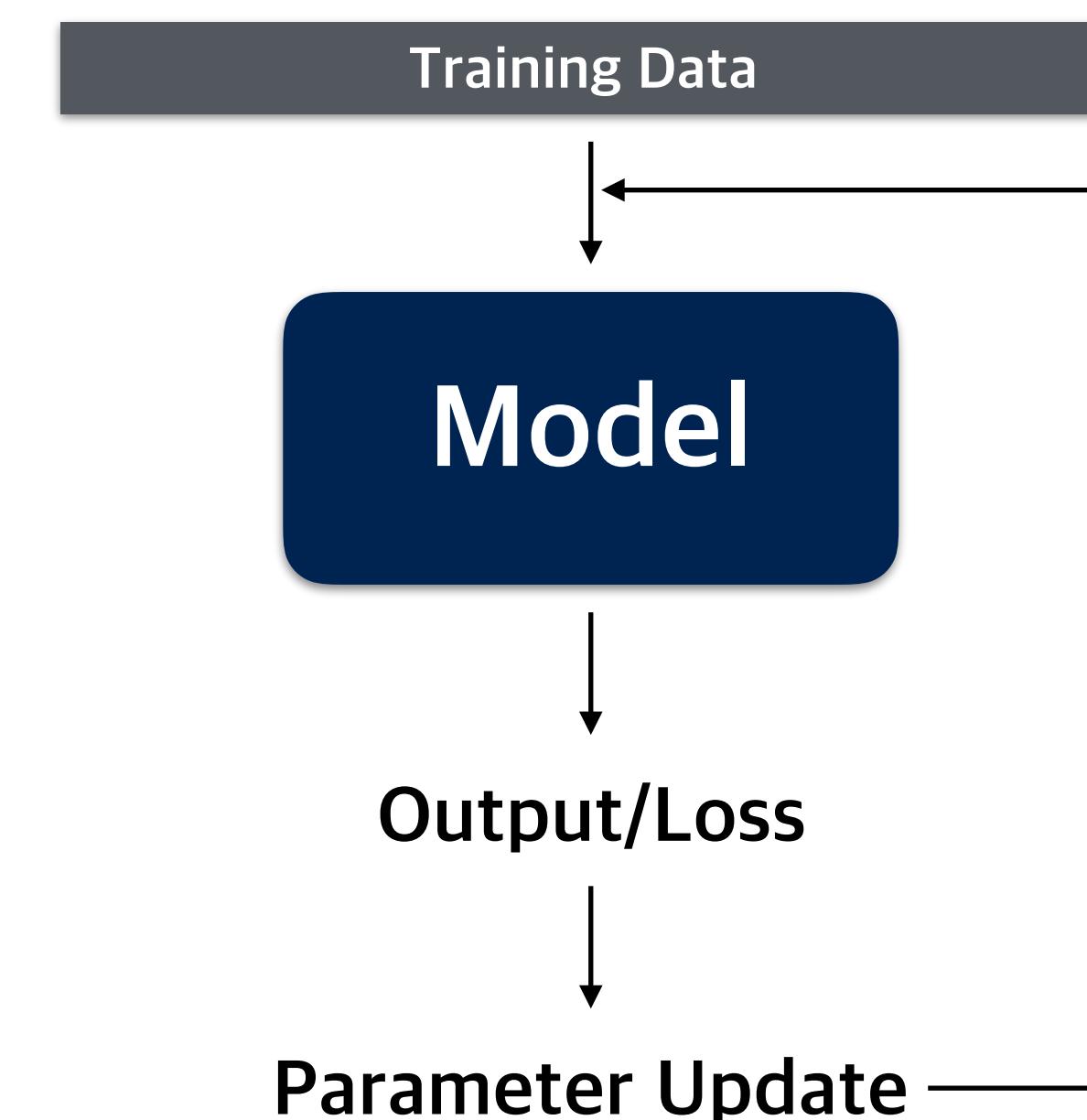
## □ Gradient Descent Method (경사강하법)

- 모델 학습을 위해 가장 대표적으로 사용되는 최적화 기법
- Cost의 gradient를 계산한 후, cost가 감소하는 방향으로 반복적으로 parameter를 조절하며 최적해에 수렴하는 방법

<Gradient Descent Method>

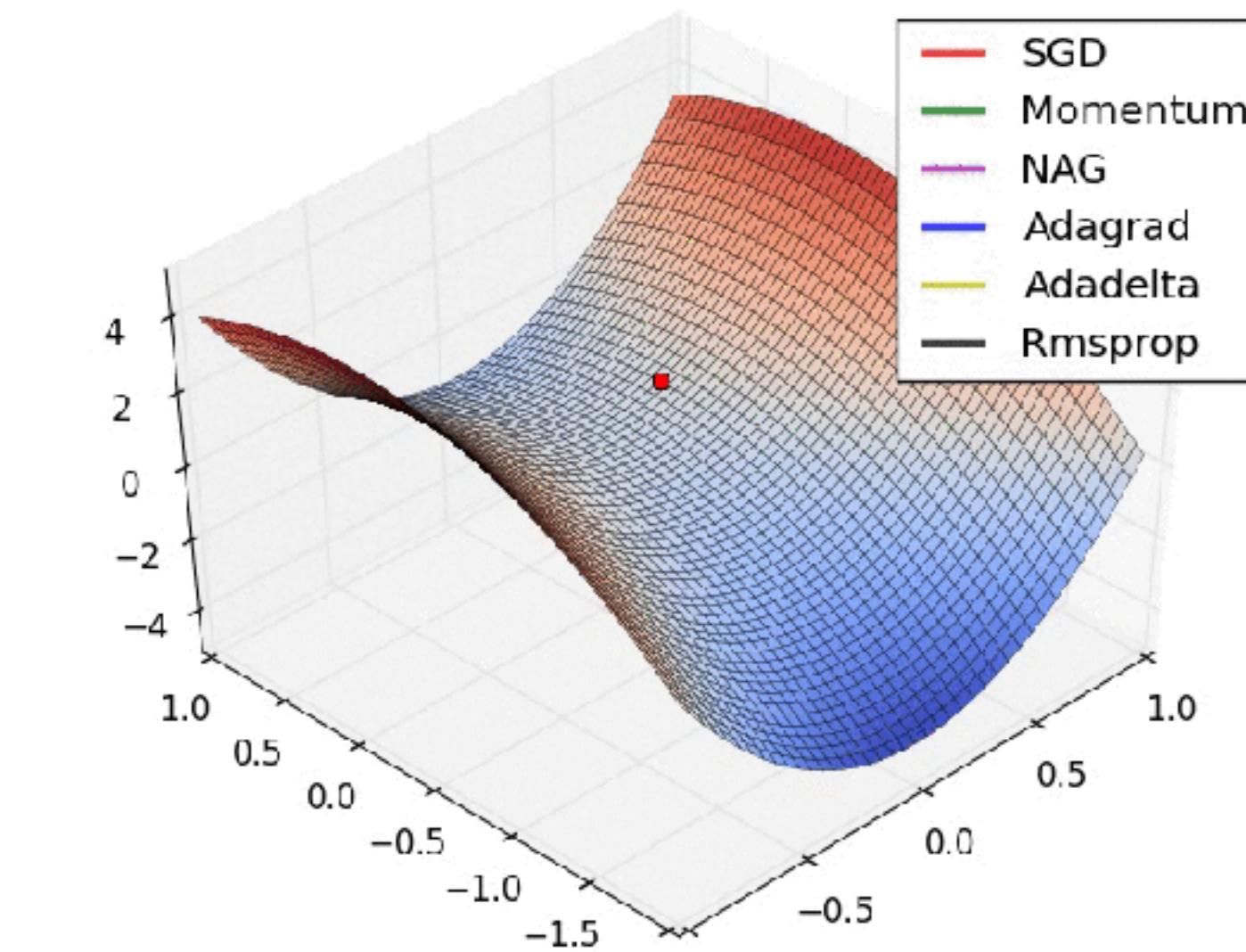
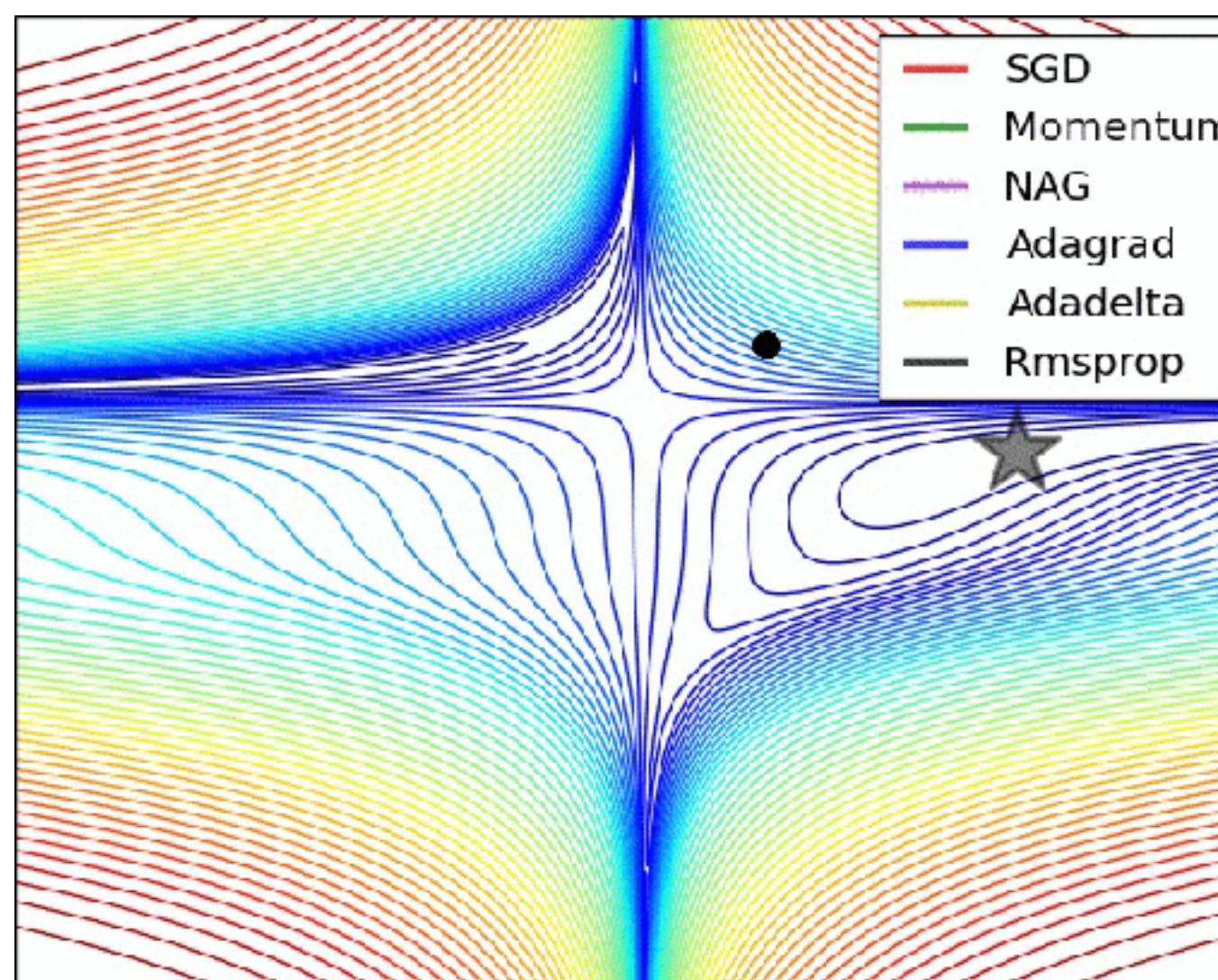


<Full Batch Learning 예시>



# Gradient Descent Learning Algorithm

- ❑ Gradient Descent Method의 성능을 개선한 다양한 방식의 Optimizers이 연구되었으며, Gradient를 기반으로하는 학습 방법임은 동일함
- ❑ Momentum, Nesterov momentum, Adagrad, Adadelta, RMSprop, Adam (Adaptive momentum) 등을 주로 사용하며, 일반적으로 Adam optimizer를 우선적으로 사용하는 추세
  - 모든 상황에서 잘 작동하는 Optimizer는 존재하지 않으며, 상황 별로 다름 (No Free Lunch!!)



## Calculation Difficulty of Gradient

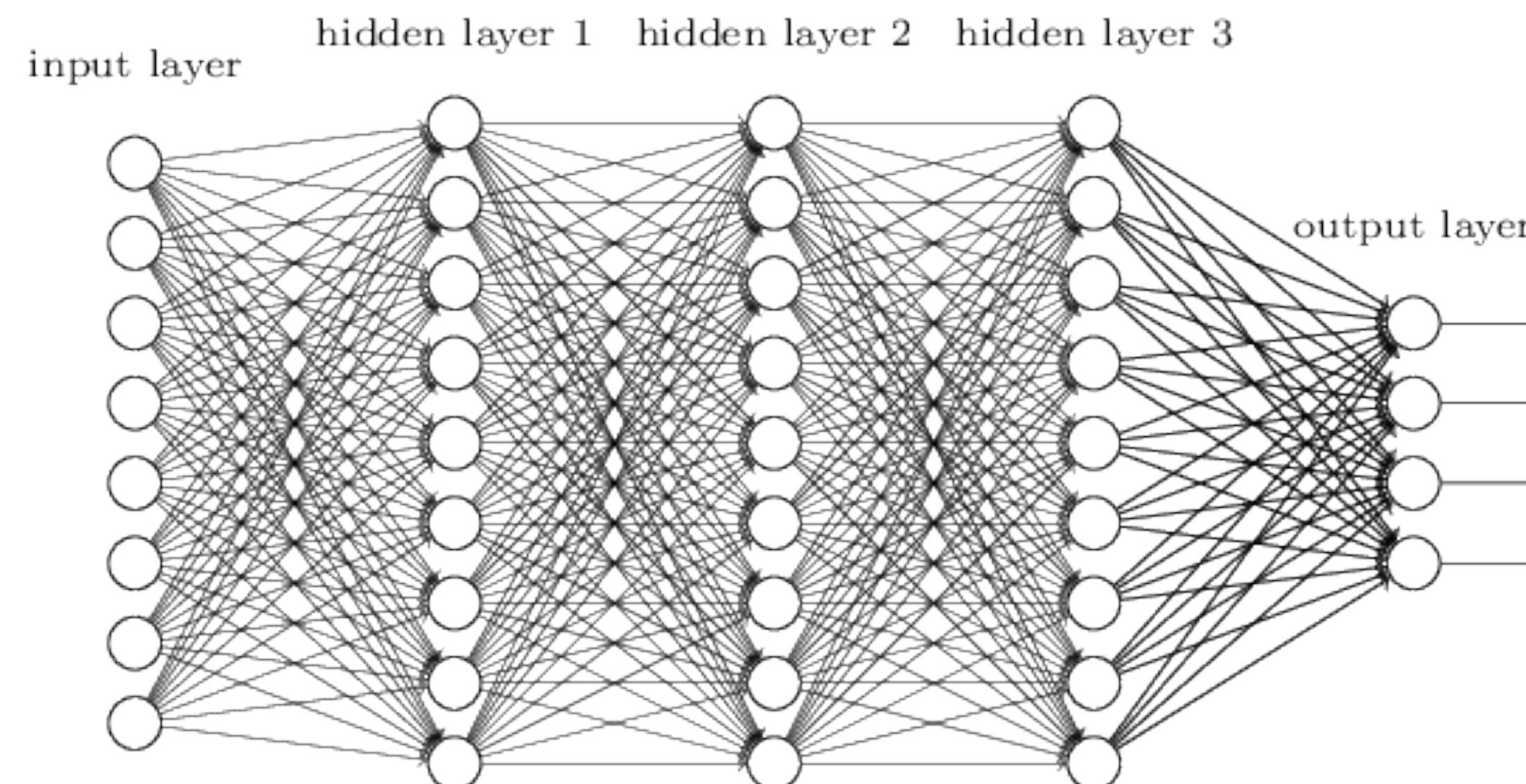
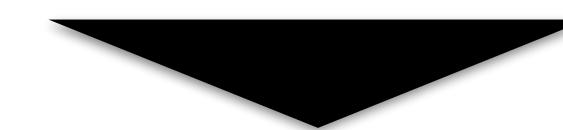
- 파라미터 업데이트를 위해 매 반복(Iteration)마다 Gradient를 직접 구하는 것은 계산적으로 매우 비효율적임

$$\frac{\partial J(\Theta)}{\partial \Theta} = \frac{J(\Theta+h) - J(\Theta-h)}{2h}$$

# Calculation Difficulty of Gradient

- 파라미터 업데이트를 위해 매 반복(Iteration)마다 Gradient를 직접 구하는 것은 계산적으로 매우 비효율적임

$$\frac{\partial J(\Theta)}{\partial \Theta} = \frac{J(\Theta+h) - J(\Theta-h)}{2h}$$

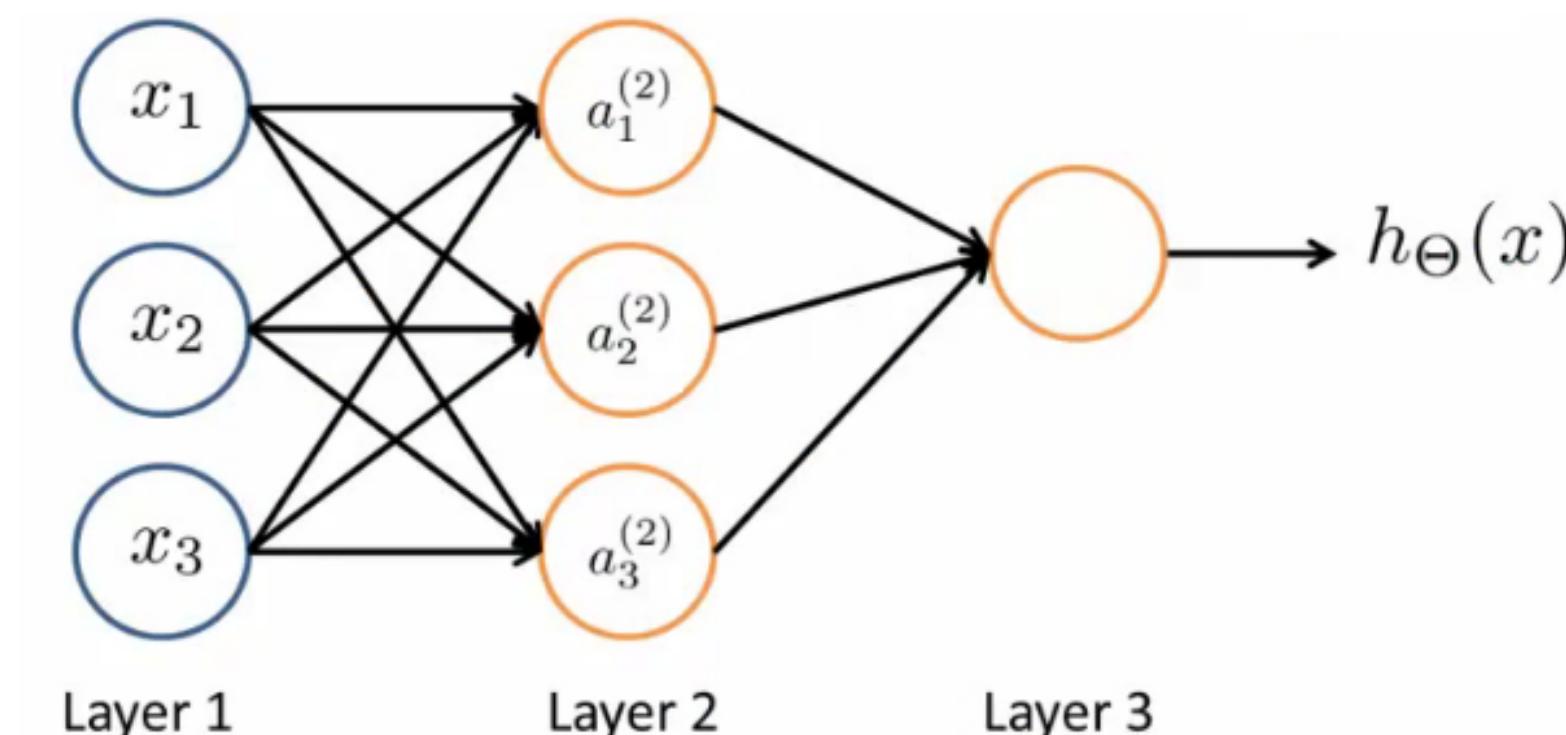
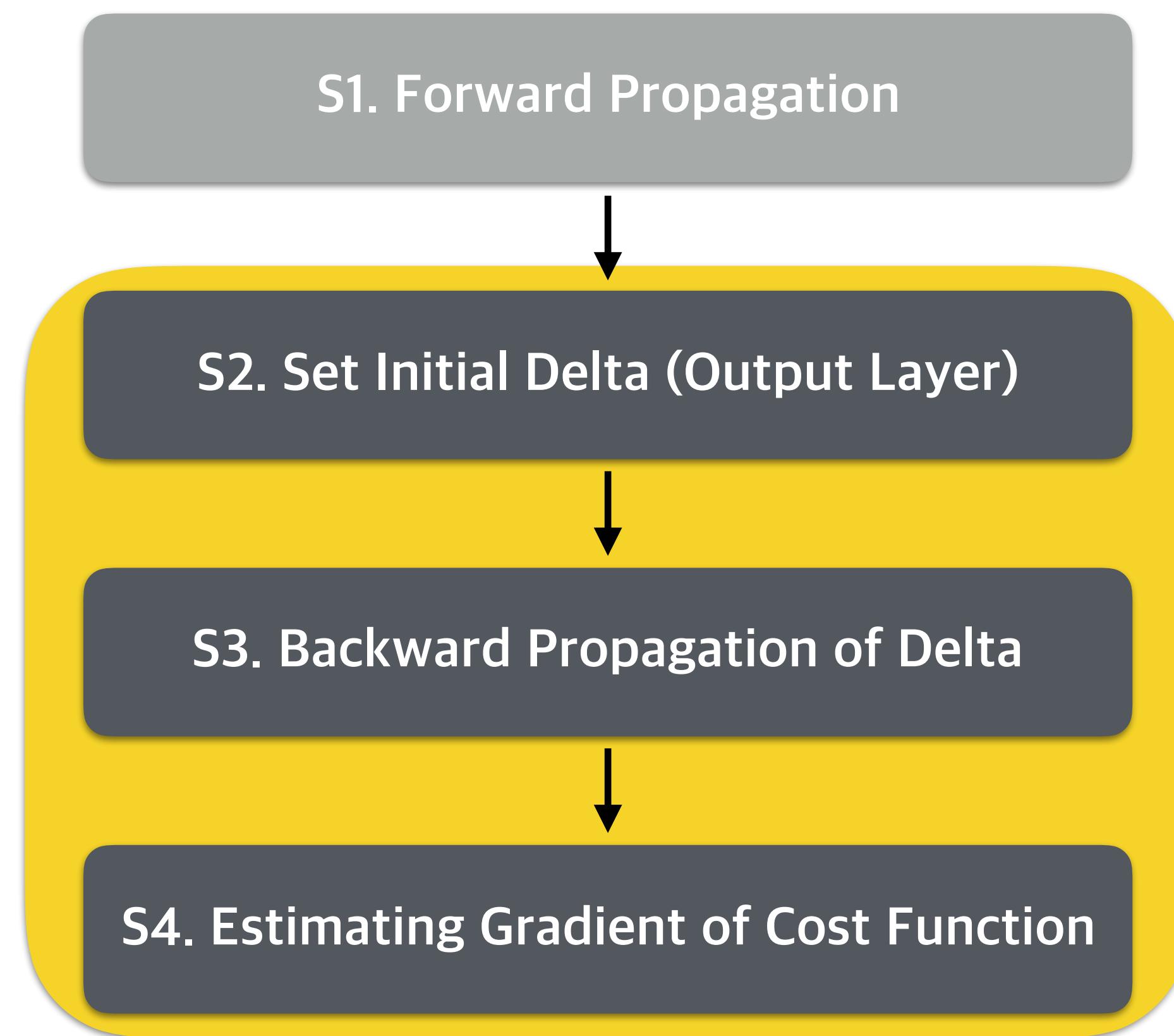


270 # of  
calculation (parameters)  
per each Iteration

# Backpropagation

- Deep Neural Network의 복잡한 과정만큼, Gradient를 직접 계산하는 것은 매우 복잡한 과정이 필요함
- Backpropagation (or Delta Rule)을 이용하면 간단한 계산으로 효율적으로 Gradient를 계산할 수 있음

<Backpropagation Algorithm>



Layer 1      Layer 2      Layer 3

S3. Backward Propagation

S2. Initial Delta (Output Layer)

$$\delta_i^{(last)} = y_i - a_i^{(last)}$$

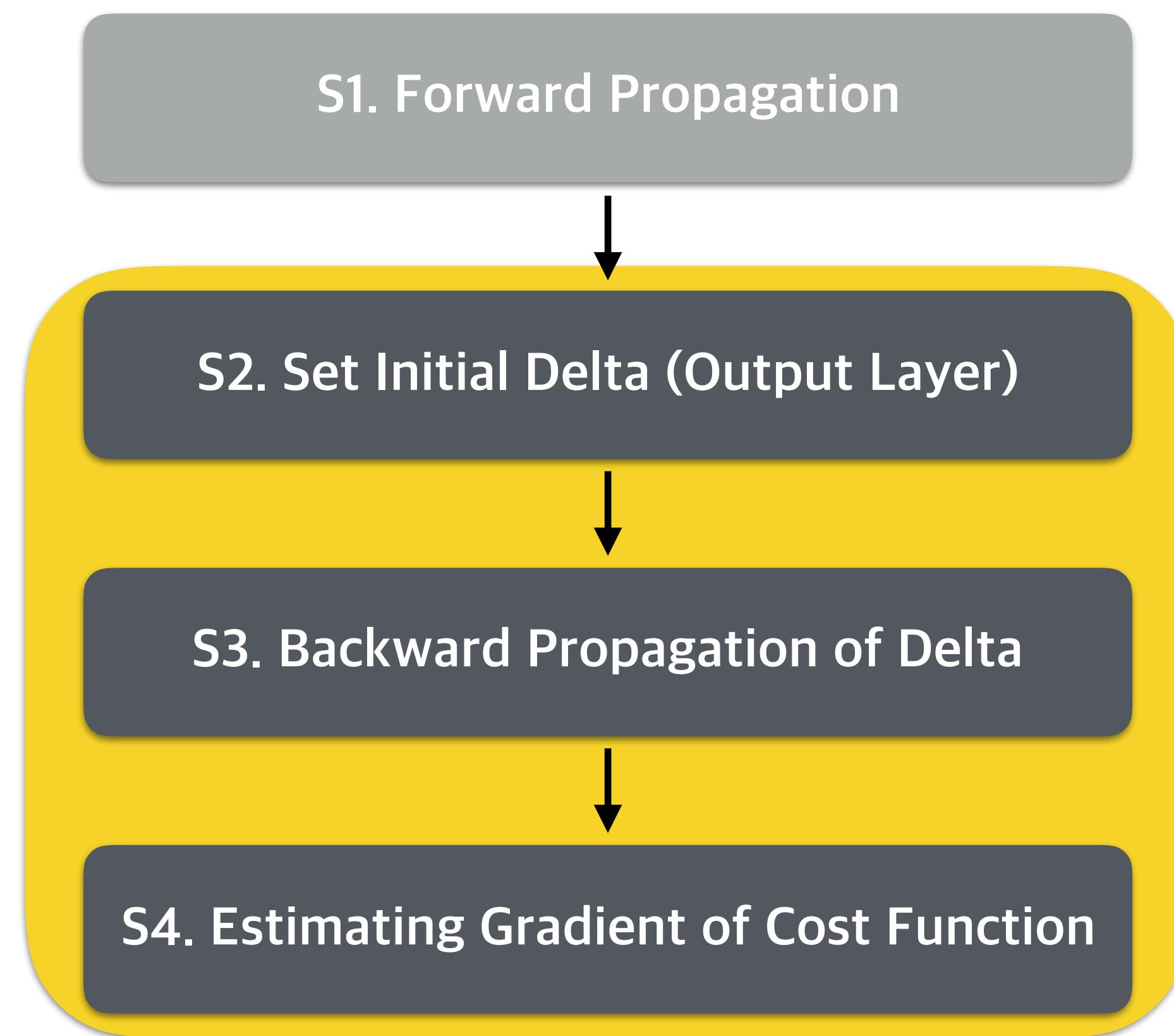
S4. Gradient Estimation

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$

# Backpropagation

- Deep Neural Network의 복잡한 과정만큼, Gradient를 직접 계산하는 것은 매우 복잡한 과정이 필요함
- Backpropagation (or Delta Rule)을 이용하면 간단한 계산으로 효율적으로 Gradient를 계산할 수 있음

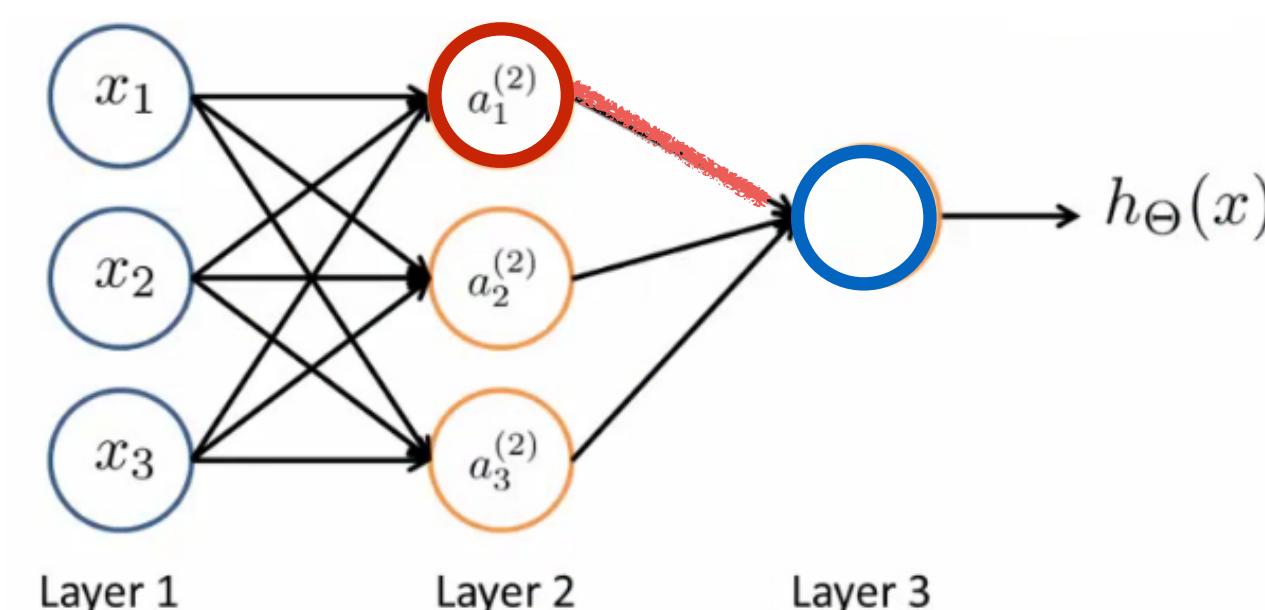
<Backpropagation Algorithm>



## S4. Gradient Estimation

$$\frac{\partial}{\partial \theta_{ij}^{(l)}} J(\theta) = a_j^{(l)} \delta_i^{(l+1)}$$

결국 특정 weight에 대한 Cost Function의 Gradient는  
(앞쪽 뉴런의 Activation 값) \* (뒷쪽 뉴런의 Delta값)  
으로 계산한 결과와 동일하다는 것이 핵심!

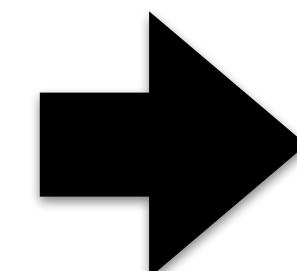


# Backpropagation

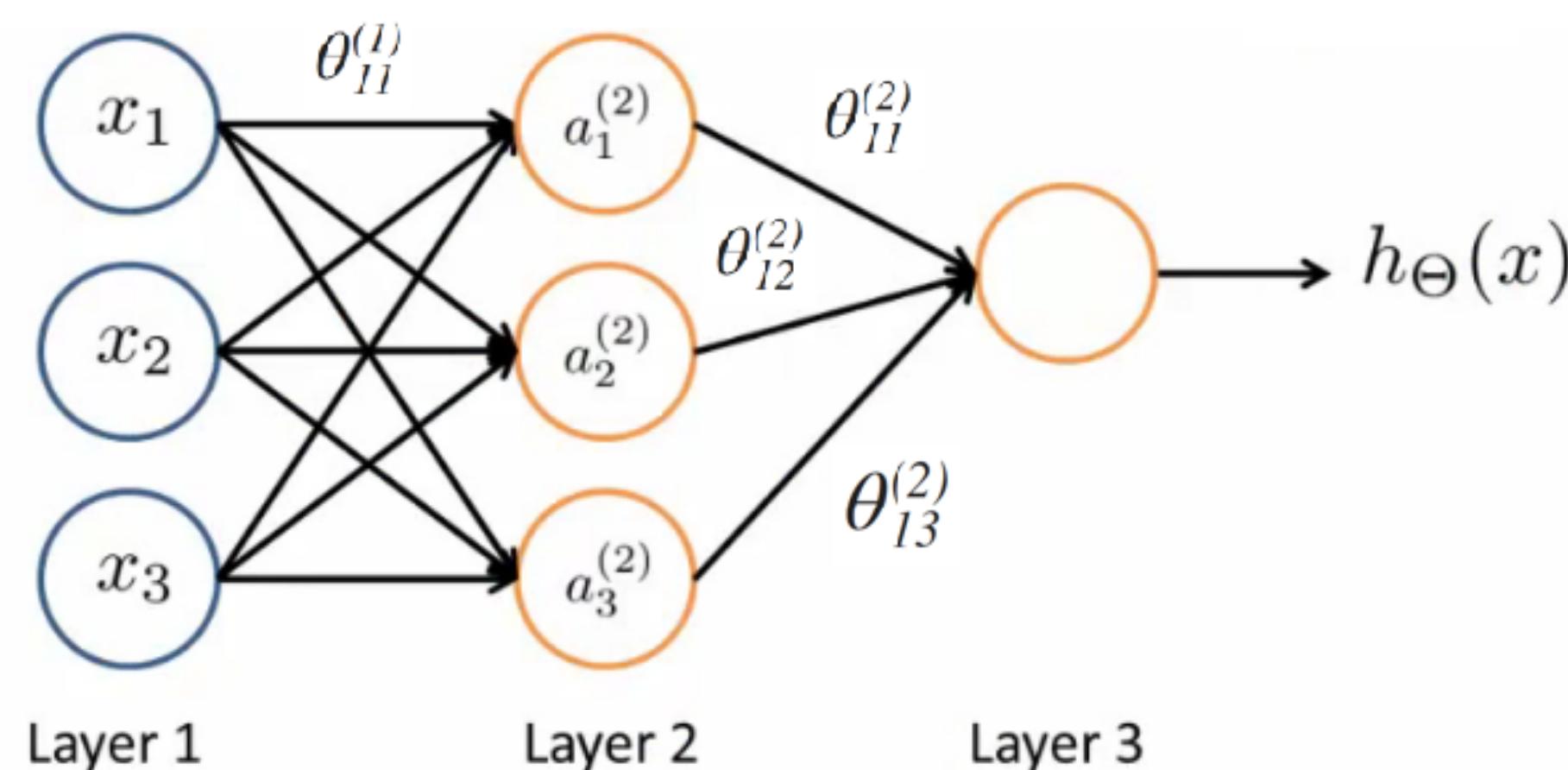
- Deep Neural Network의 복잡한 과정만큼, Gradient를 직접 계산하는 것은 매우 복잡한 과정이 필요함
- Backpropagation (or Delta Rule)을 이용하면 간단한 계산으로 효율적으로 Gradient를 계산할 수 있음
- Backpropagation은 Chain Rule을 바탕으로 Gradient를 계산하는 것에서 시작함

<Chain Rule>

$$\frac{\Delta J(\theta)}{\Delta \theta} = \frac{\Delta J(\theta)}{\Delta y} \frac{\Delta y}{\Delta \theta}$$

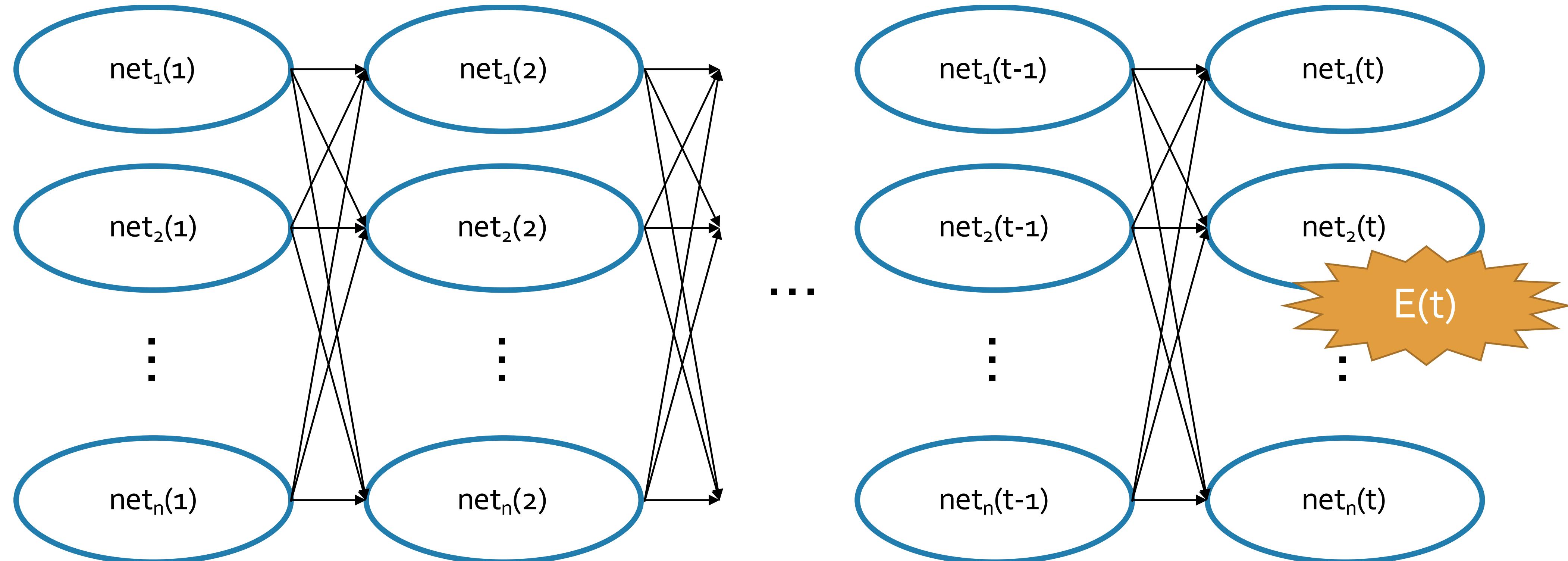


$$\frac{\Delta J(\theta)}{\Delta \theta_{II}^{(2)}} = \frac{\Delta J(\theta)}{\Delta a_I^{(3)}} \frac{\Delta a_I^{(3)}}{\Delta \theta_{II}^{(2)}}$$

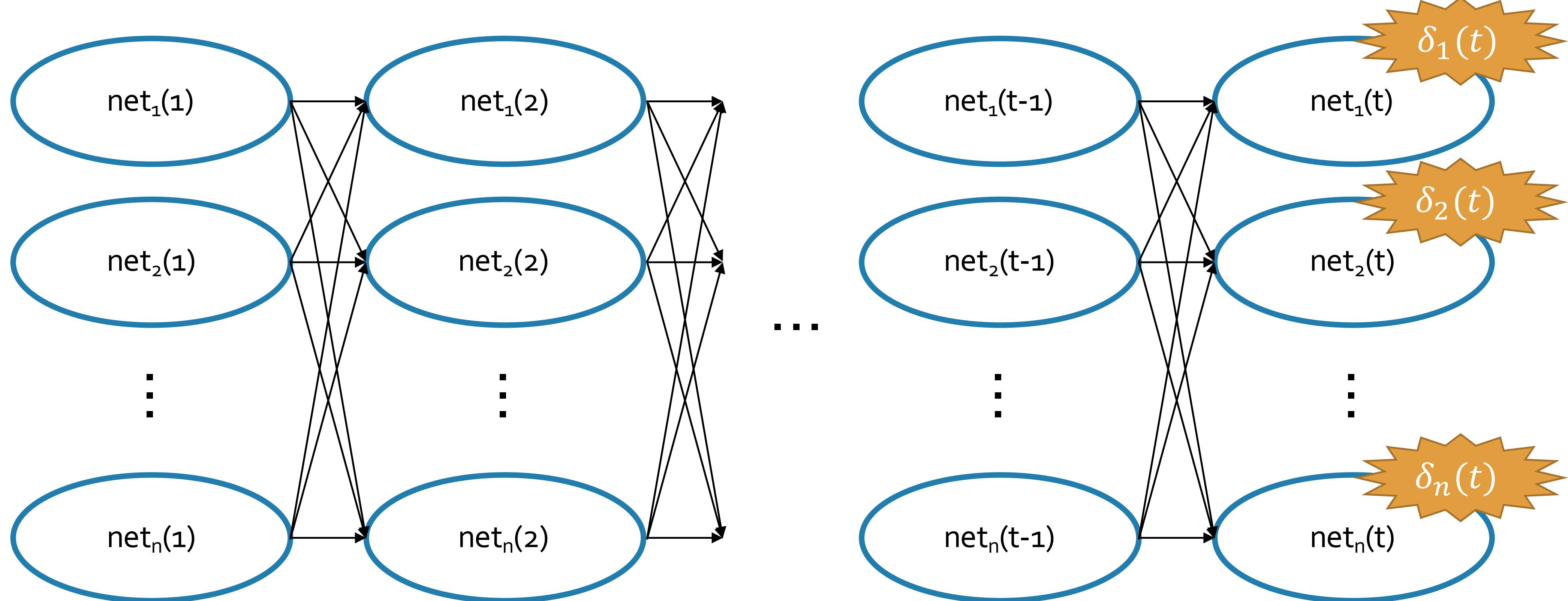


$$\frac{\Delta J(\theta)}{\Delta \theta_{II}^{(1)}} = \frac{\Delta J(\theta)}{\Delta a_I^{(2)}} \frac{\Delta a_I^{(2)}}{\Delta a_I^{(3)}} \frac{\Delta a_I^{(3)}}{\Delta \theta_{II}^{(1)}} \delta_I^{(2)}$$

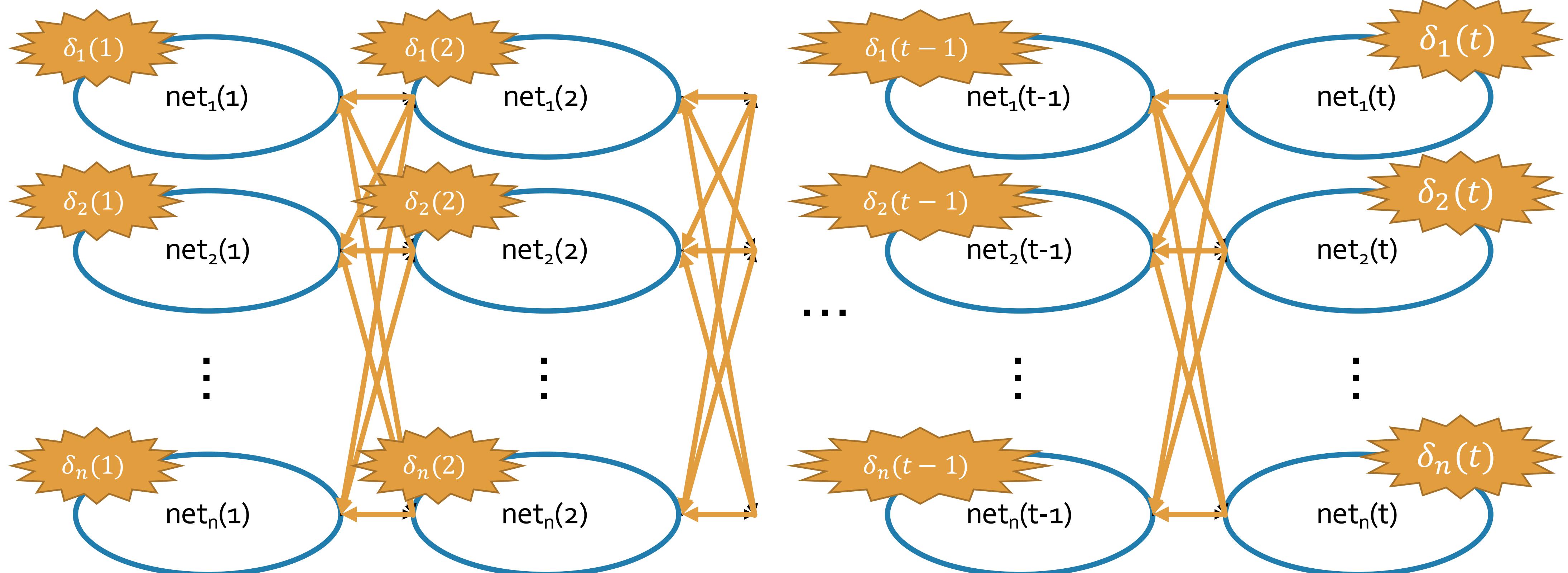
# Backward Propagation Process



## Backward Propagation Process (Cont'd)



## Backward Propagation Process (Cont'd)



# Review of Neural Network

## ❑ Neural Network Components

1) **Architecture:** Input, Hidden, Output Layer

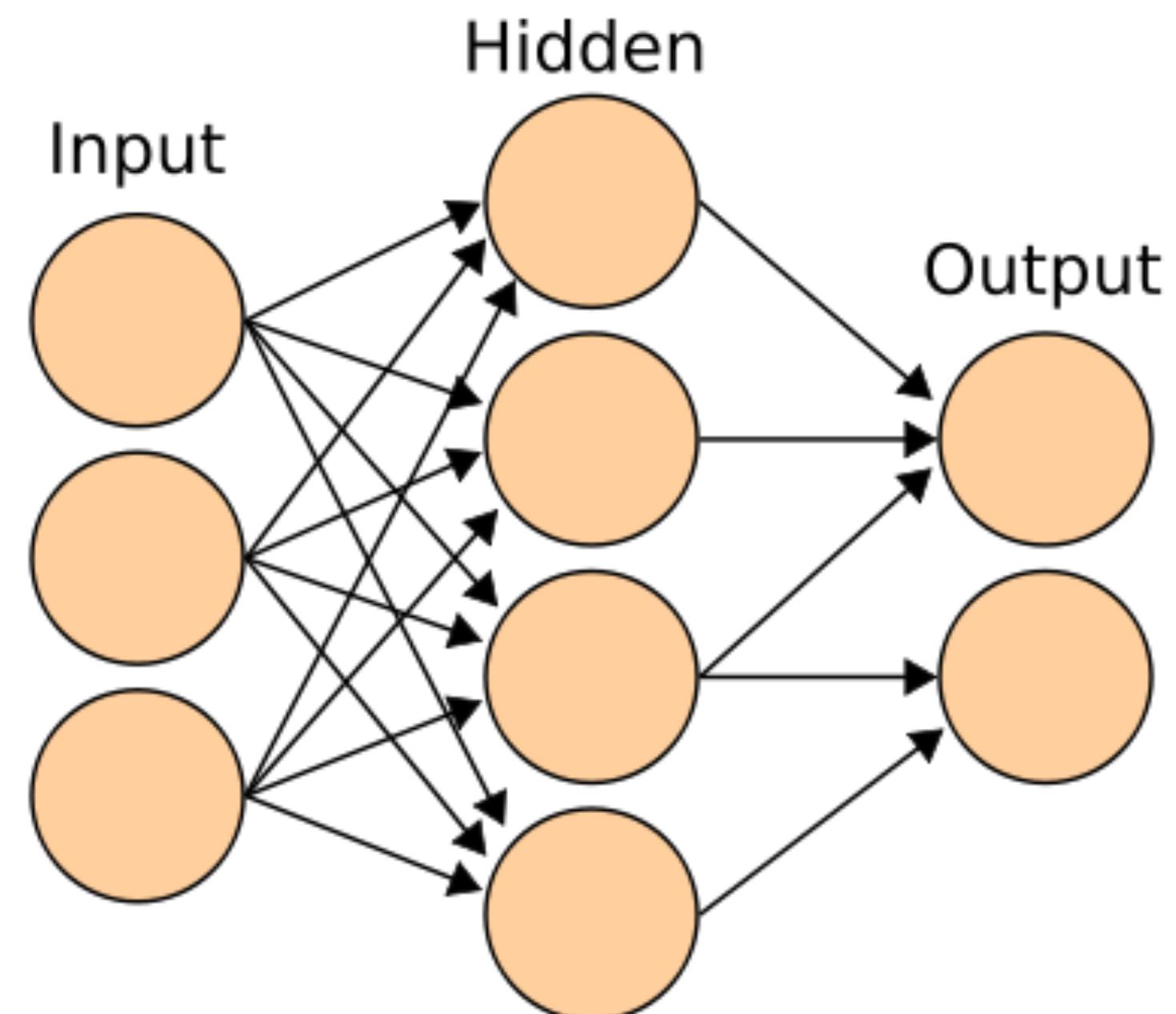
2) **Calculation:** Linear Combination & Activation Function

3) **Initialization:** (based on) Random Initialization

4) **Cost Function**

5) **(Gradient-based) Optimizer**

- Learning Rate
- Backpropagation



**We have “1+” more issues**

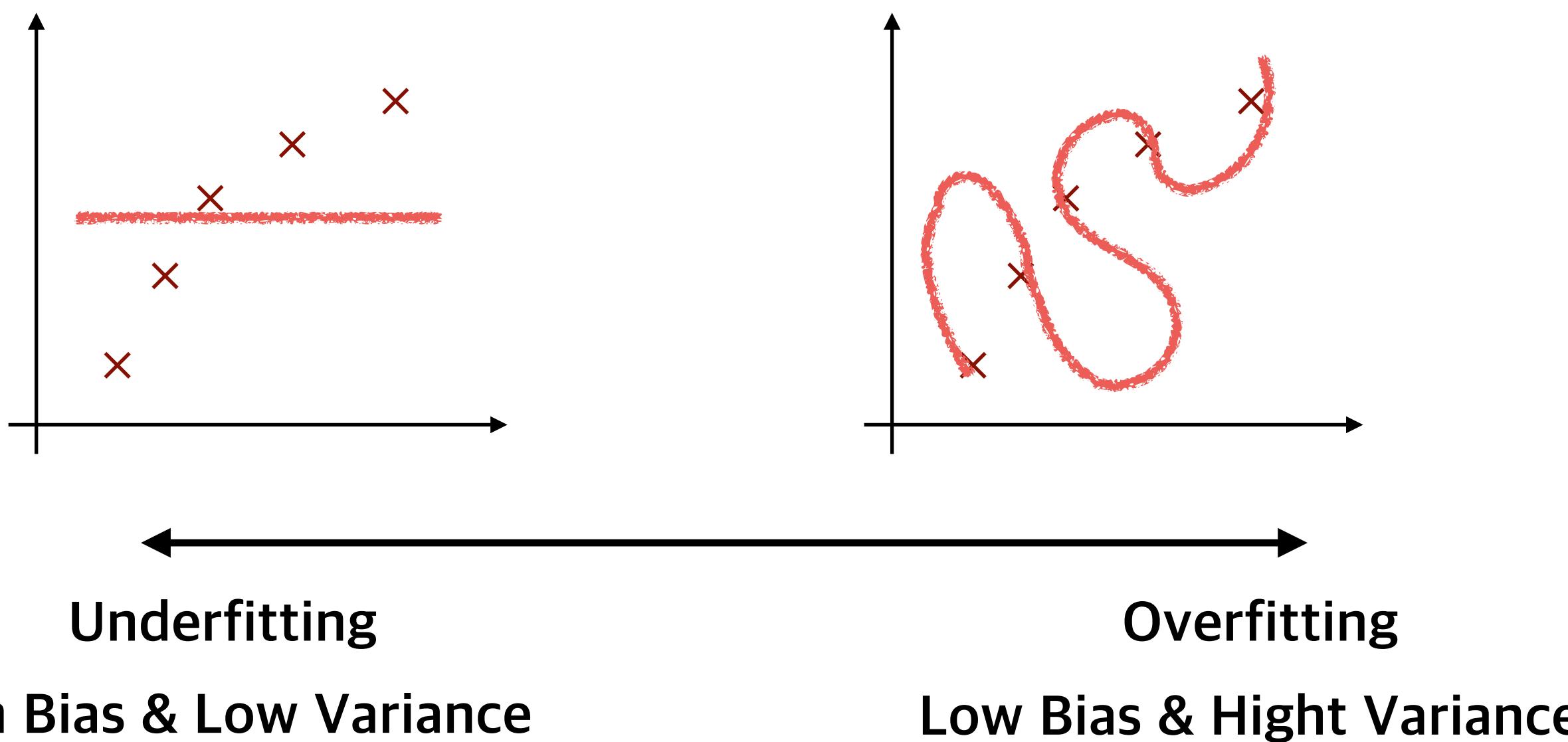
We have “1+” more issues

- Generalization

# Overfitting and Generalization

- 인공지능의 목적 중 하나는 **훈련되지 않은 새로운 상황**에 대하여 높은 성능을 유지하는 것
- 모델 학습시, Training data set에 편향되어 학습되어지는 **Overfitting 현상**이 종종 발생함
- 특히, 딥러닝을 구성하는 인공 신경망(neural network)는 모델이 복잡해질 수록 overfitting 현상이 빈번하게 발생하며, 이를 해결하는 것이 중요한 과제 중 하나

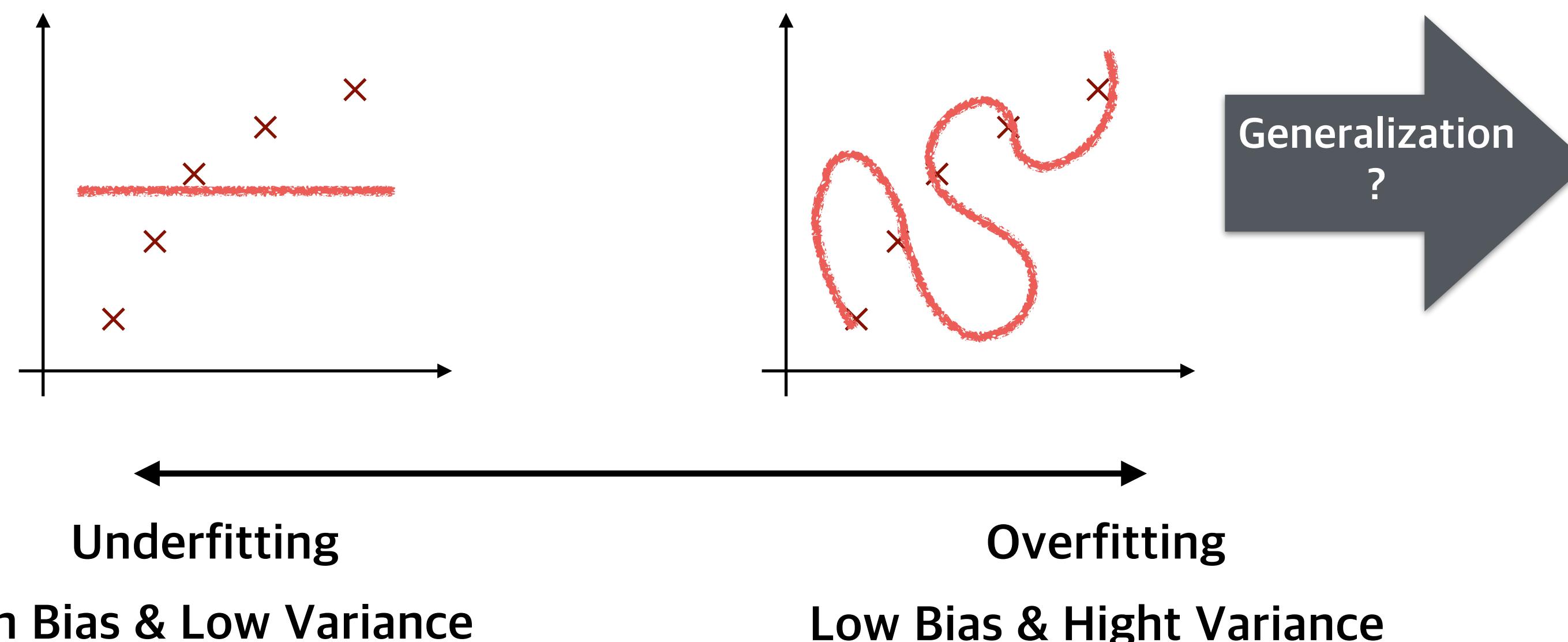
## <Overfitting Example>



# Overfitting and Generalization

- 인공지능의 목적 중 하나는 **훈련되지 않은 새로운 상황**에 대하여 높은 성능을 유지하는 것
- 모델 학습시, Training data set에 편향되어 학습되어지는 **Overfitting 현상**이 종종 발생함
- 특히, 딥러닝을 구성하는 인공 신경망(neural network)는 모델이 복잡해질 수록 overfitting 현상이 빈번하게 발생하며, 이를 해결하는 것이 중요한 과제 중 하나

<Overfitting Example>



Cost Function-based Approach

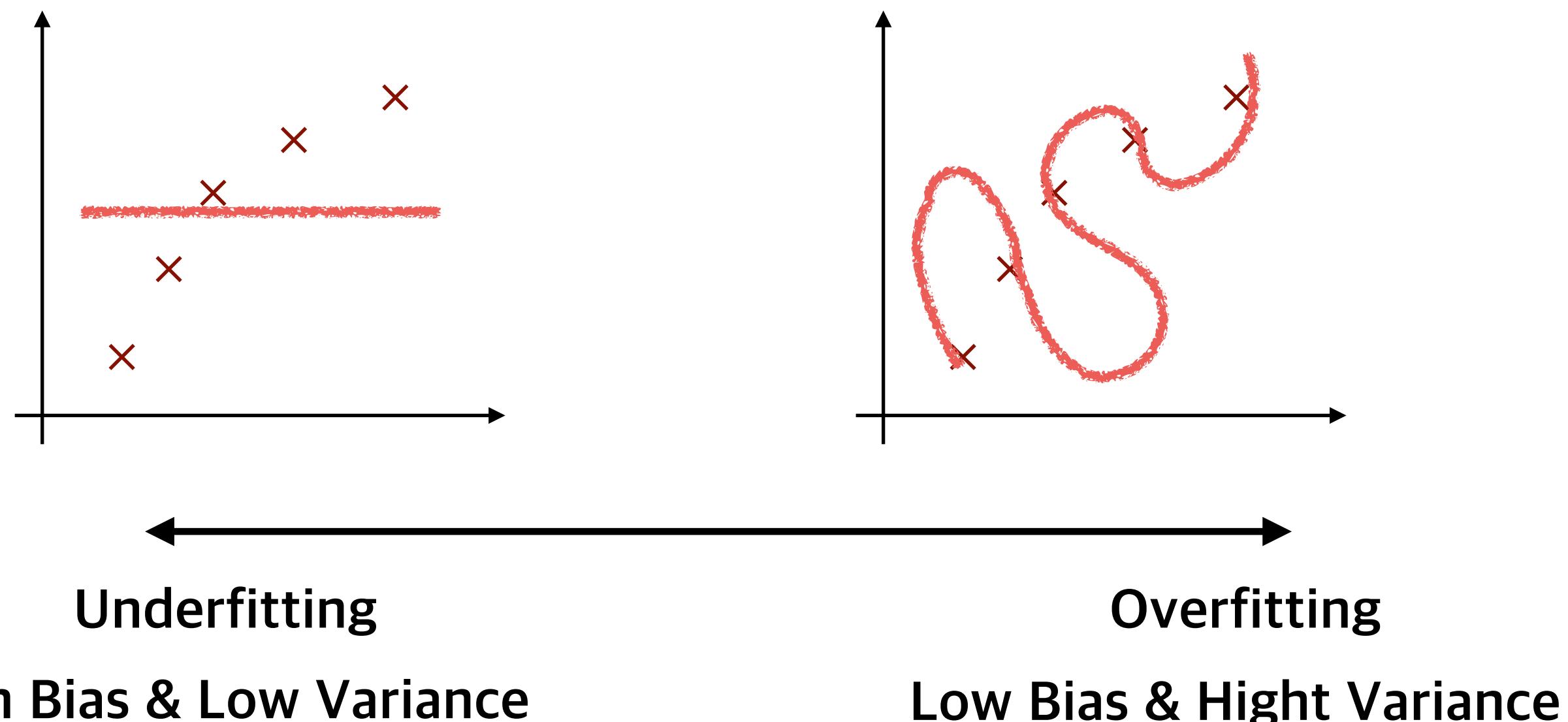
Validation-based Approach (Data)

Model-based Approach

# Overfitting and Generalization

- 인공지능의 목적 중 하나는 **훈련되지 않은 새로운 상황**에 대하여 높은 성능을 유지하는 것
- 모델 학습시, Training data set에 편향되어 학습되어지는 **Overfitting 현상**이 종종 발생함
- 특히, 딥러닝을 구성하는 인공 신경망(neural network)는 모델이 복잡해질 수록 overfitting 현상이 빈번하게 발생하며, 이를 해결하는 것이 중요한 과제 중 하나

## <Overfitting Example>



## Regularization

$$J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$$

Regularization Term

- L1 regularization:  $\lambda |\Theta|$
- L2 regularization:  $\lambda |\Theta^2|$

(Regularization in 3 Layer NN)

$$\frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

- $\lambda$  가 커질수록 Underfitting 되는 효과가 있음 (Why?)
- Regularization is “Weight Decaying”
  - Regularization Term은 학습시 자기 자신의 Weight 크기 만큼 변화를 감소시키는 역할을 함
  - 학습에 필수적인 Weights는 감소 효과를 극복하고 지속적으로 Weights를 업데이트하여 학습 진행

# Cross Validation Data

- 보유하고 있는 데이터를 1) Training, 2) Validation, 3) Test Data로 나누어 검증하여 일반화 성능을 높이는 방법
- Validation 데이터를 통해 학습 중인 모델의 일반화 검증을 진행하고,  
가장 높은 일반화 성능 (**lowest validation error**) 을 갖는 모델을 최종 모델로 선정

<Data Split with No Validation >



Test

Training Data

<Data Split with Validation>

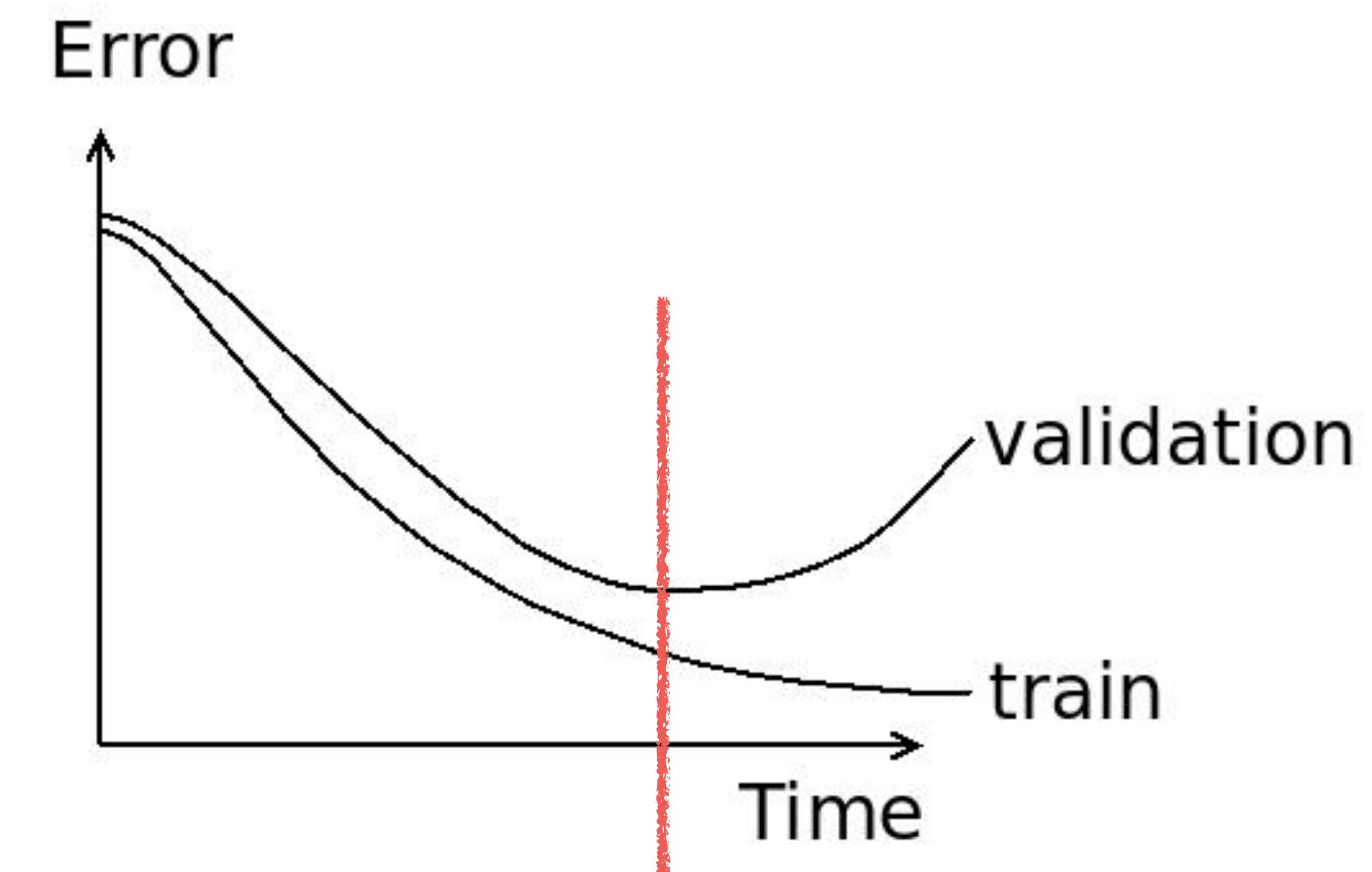


Test

Validation

\*Test 데이터는 학습 완료 후 최종 성능을 확인하기 위해서만 사용됨

<Learning Curve>



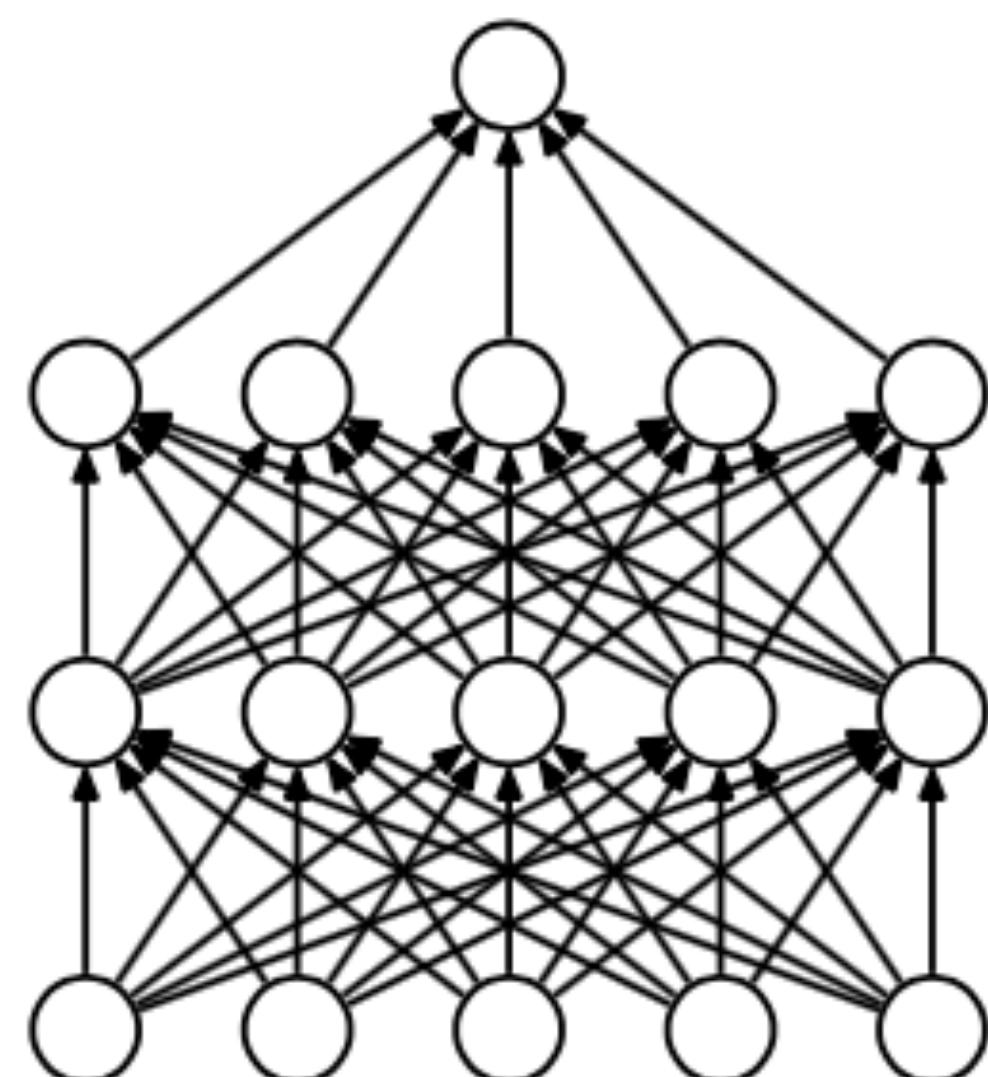
**Early Stop**

# Cross Validation Data

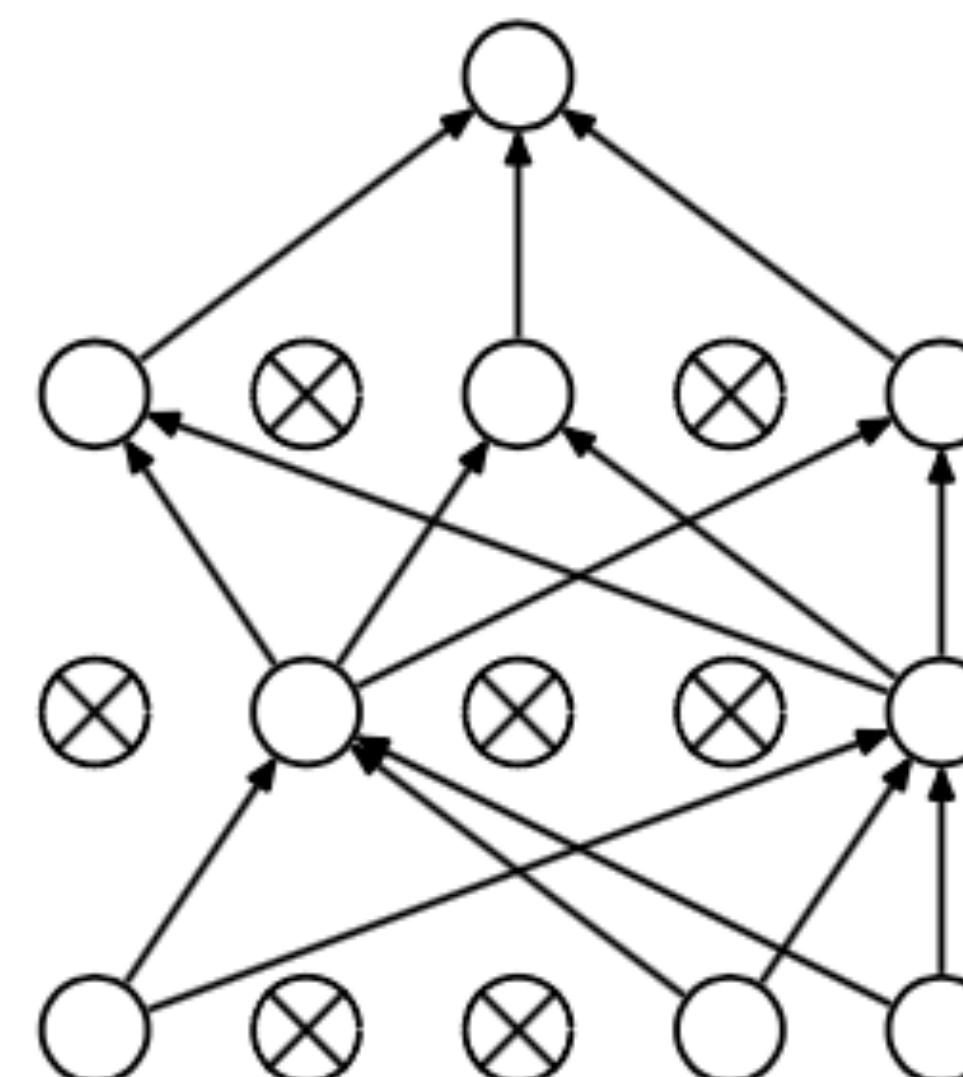
□ 모델의 일반화(generalization)을 위해 모델 차원에서 접근하는 방식은 다양하지만

1) Dropout, 2) Batch Normalization 두 가지 방법을 주로 사용함 (in Neural Networks)

<Dropout Architecture>



(a) Standard Neural Net



(b) After applying dropout.

<BN Algorithm in Training>

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;

Parameters to be learned:  $\gamma, \beta$

**Output:**  $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

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

$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{y}^{(l)} &= r^{(l)} * y^{(l)}, \end{aligned}$$

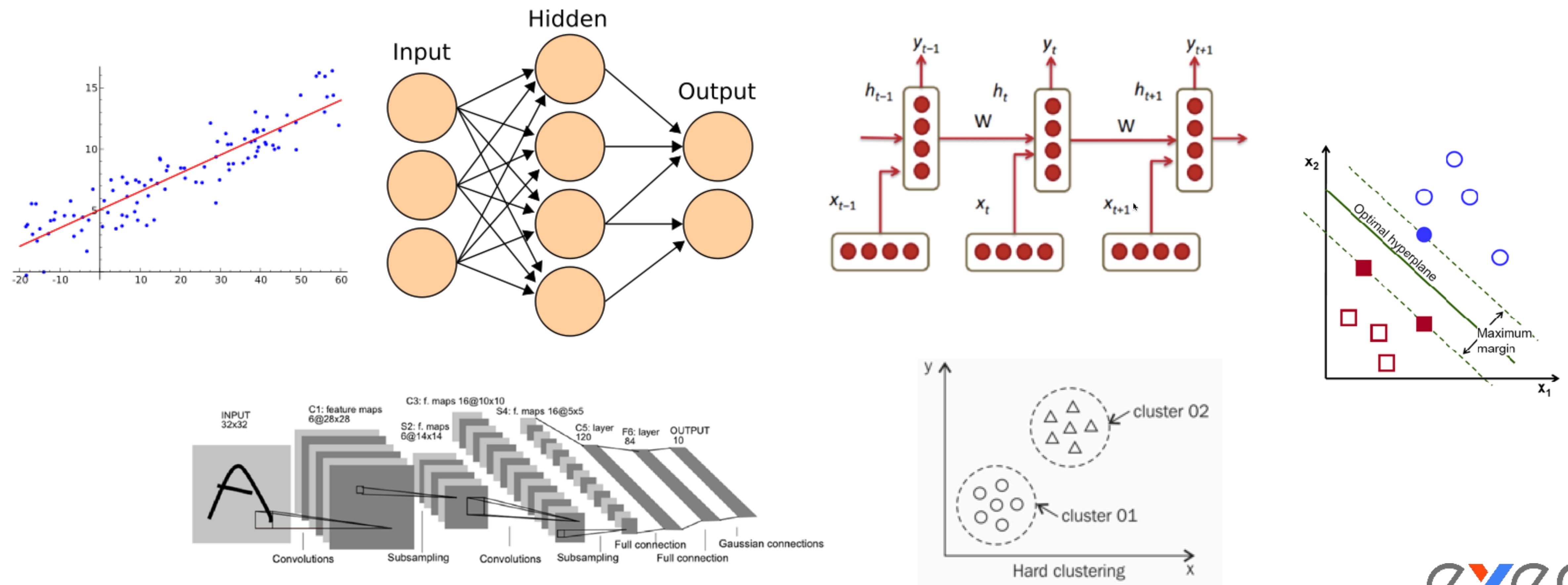
출석 체크

12시 30분부터 자유롭게 출입

# Tensorflow Introduction

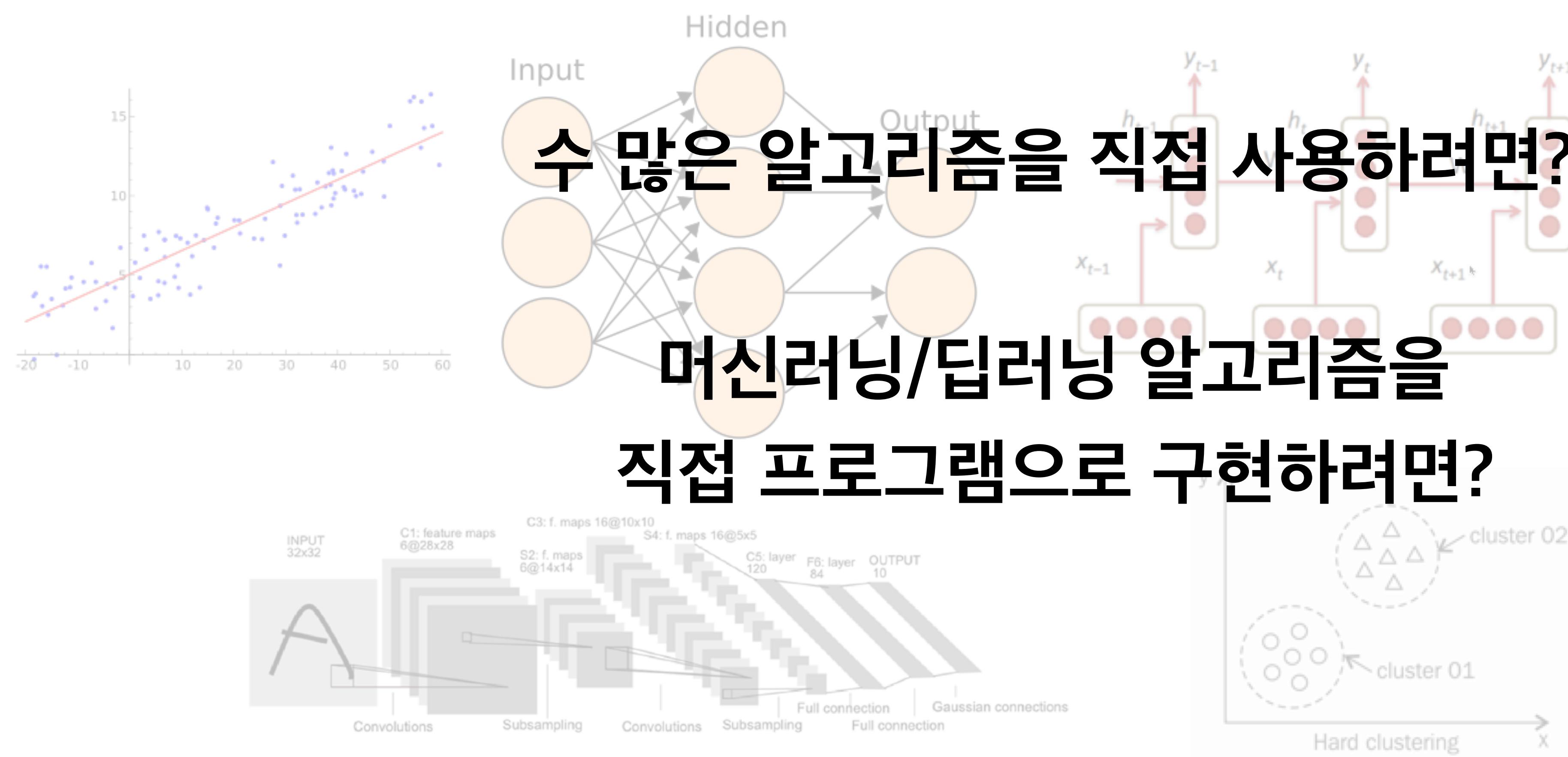
# Introduction

- We studied lots of algorithms in machine learning, especially Deep Learning !  
ex) Regression, Neural Network, Classification & Clustering Algorithms, Conv. Net, RNN, Hopfield Network, RBM, Deep Belief Network, and lots of RL algorithms....



# Introduction

- We studied lots of algorithms in machine learning, especially Deep Learning !  
ex) Regression, Neural Network, Classification & Clustering Algorithms, Conv. Net, RNN, Hopfield Network, RBM, Deep Belief Network, and lots of RL algorithms....



# Deep Learning Library

- ❑ 머신러닝과 딥러닝을 구현하기 위해 다양한 개발 환경이 존재함
- ❑ 모델 구현을 위해 다양한 언어를 사용할 수 있지만, 대개 특정 언어에서 오픈 라이브러리를 이용함
- ❑ 주요 머신러닝 모델에 따라 사용하는 딥러닝 라이브러리는 달라질 수 있음

## <Various Software and Library for Deep Learning>

Caffe



DL4J  
Deeplearning4j



Microsoft  
CNTK



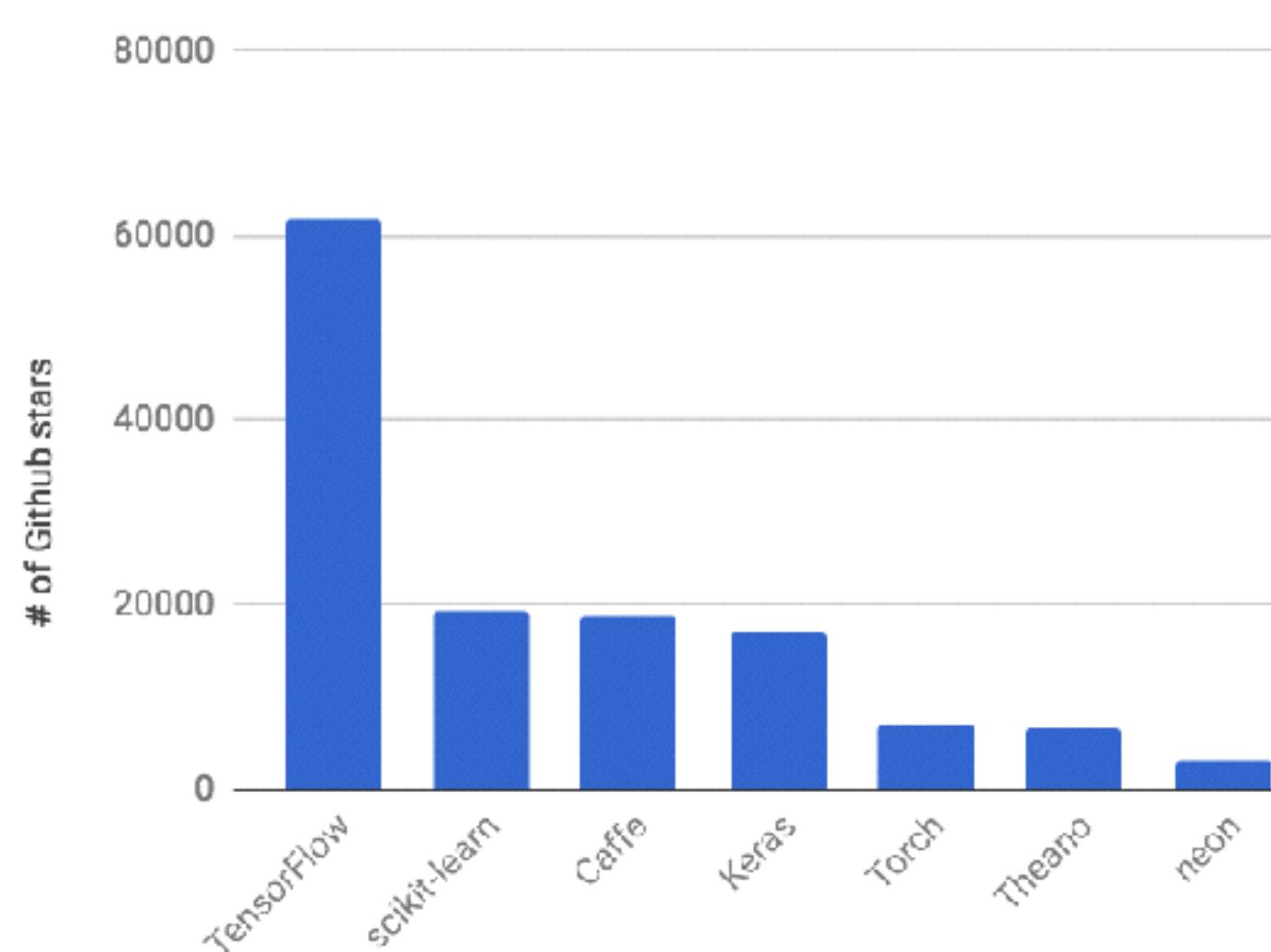
MINERVA



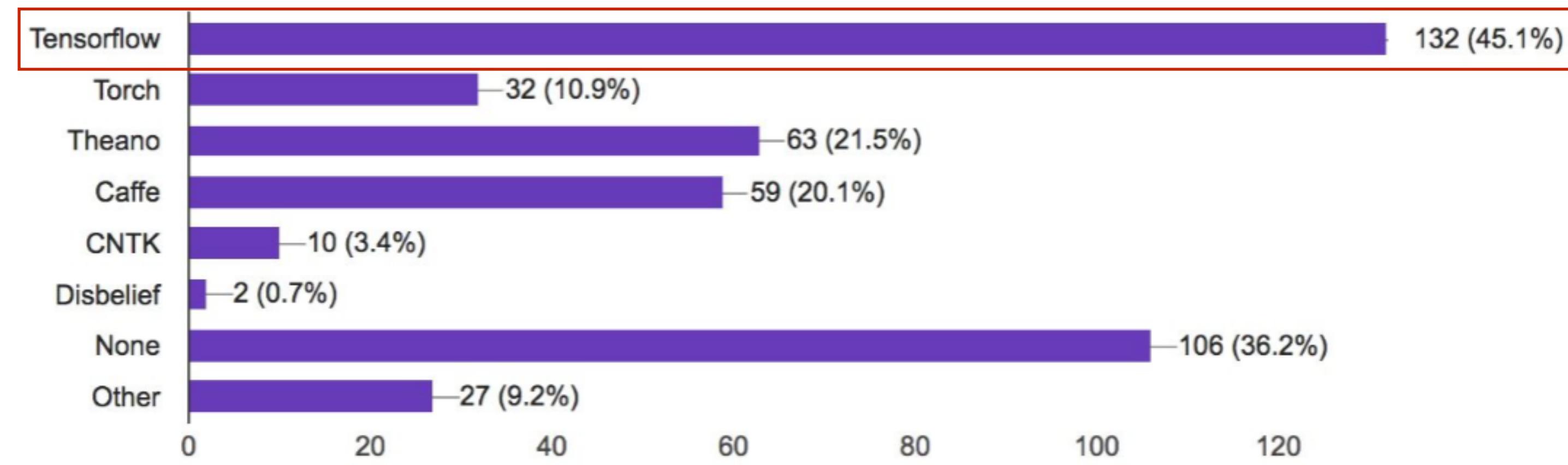
theano



Github Stars per Deep Learning Github Repository (as of June 2017)



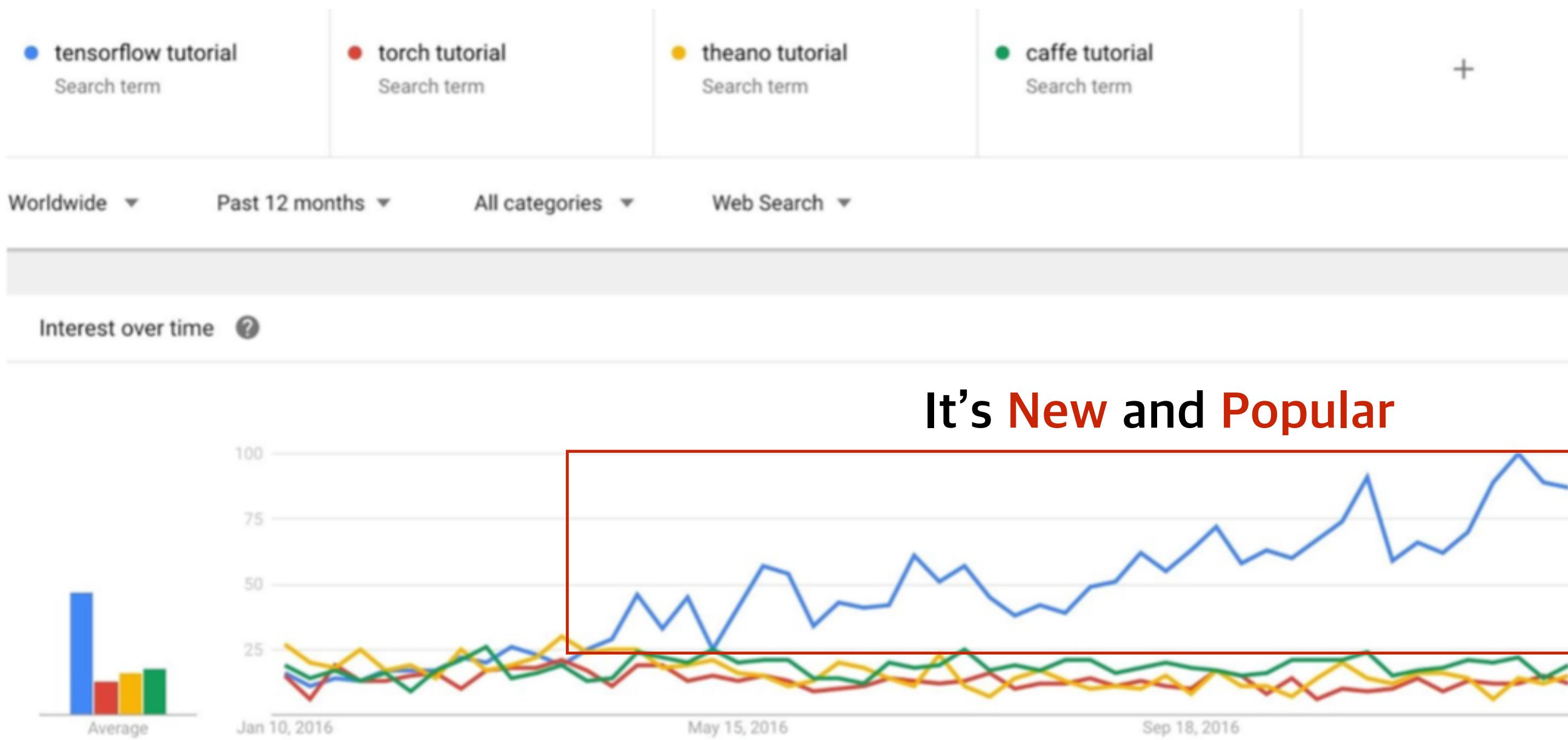
## <Stanford Students' Deep Learning Library Usage Experience >



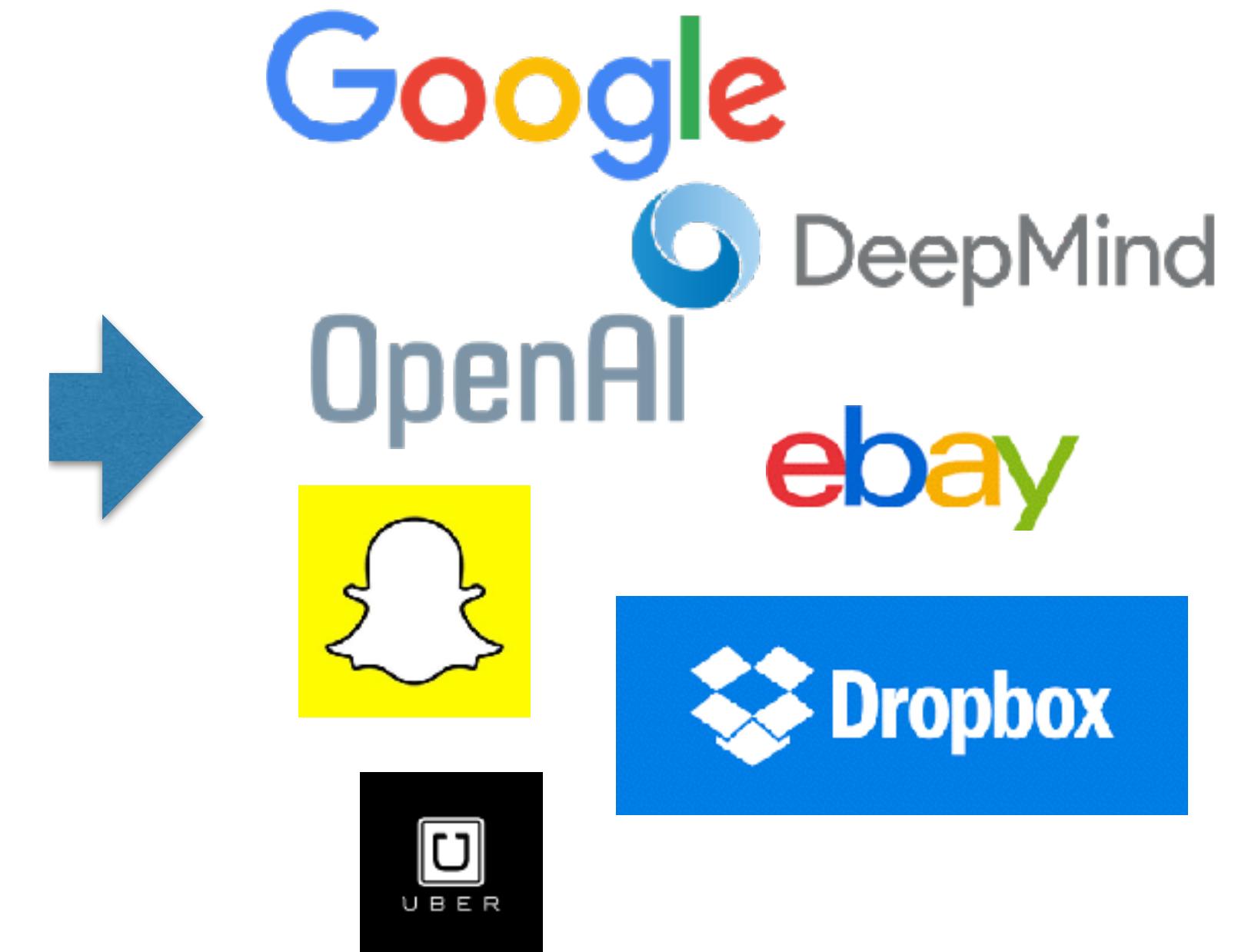
# Usage Tendency of TensorFlow

▣ TensorFlow는 다른 오픈 라이브러리에 비해 늦게 출시되었지만, 현재 가장 유명한 Deep Learning 라이브러리

## <Search Trend of Each Keywords>



## <Companies using TensorFlow>



Then, What is on earth TensorFlow?

## ■ TensorFlow

- 2015년 10월 Google Brain Team이 개발하고 공개한 머신러닝/딥러닝 오픈소스 라이브러리 (Apache 2.0)
- “TensorFlow is an open source software library for **numerical computation using data flow graphs.**”
- 1개 이상의 CPU 또는 GPU를 다양한 디바이스에서 사용 가능하도록 구조화되어 있음
- 1) Building Dataflow Graph, 2) Executing the Graph Model 두 단계를 통해 프로그램을 구현하고 실행함

**(중요\*) Programming Process**

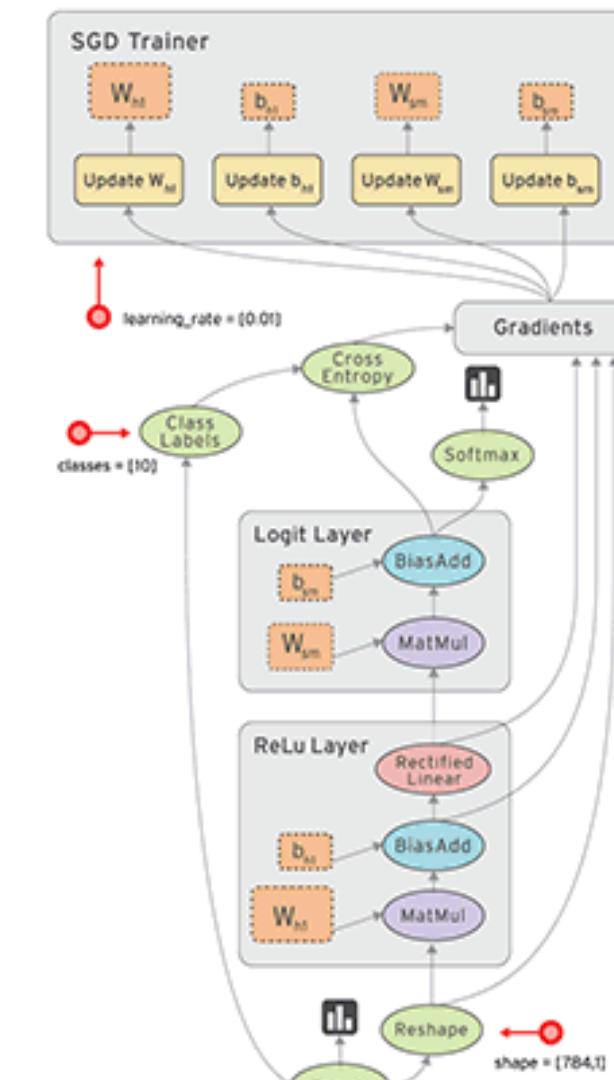
- 1) (Only) Build Graph**
  - 머신러닝 모델과 데이터의 흐름들의 계층적인 구조 (hierarchical structure)를 구현
- 2) Executing the Graph Model**
  - 구현된 Graph에 따라 데이터를 흘려보내고, 계산을 실행



# What is TensorFlow

## TensorFlow

- 2015년 10월 Google Brain Team이 개발하고 공개한 머신러닝/딥러닝 오픈소스 라이브러리 (Apache 2.0)
- “TensorFlow is an open source software library for **numerical computation using data flow graphs.**”
- 1개 이상의 CPU 또는 GPU를 다양한 디바이스에서 사용가능하도록 구조화되어 있음
- 1) Building Dataflow Graph, 2) Executing the Graph Model 두 단계를 통해 프로그램을 구현하고 실행함



## ❑ TensorFlow의 다양한 장점은 많은 사용자들을 유입시키는 이유가 되었음

- **Python API**
- **Portability**: deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API
- **Flexibility**: from Raspberry Pi, Android, Windows, iOS, Linux to server farms
- **Visualization** (TensorBoard is da bomb)
- **Checkpoints** (for managing experiments)
- **Auto-differentiation autodiff** (no more taking derivatives by hand. Yay)
- **Large community** (> 10,000 commits and > 3000 TF-related repos in 1 year)
- Awesome projects already using TensorFlow ([Next Slide](#))

# Awesome Projects with TensorFlow

## ❑ Awesome Projects Examples with TensorFlow



Paint your own style with the 1

Style transfer: writing the 1

Paint your own style with the 1

Paint your own style with the 1



1 Second

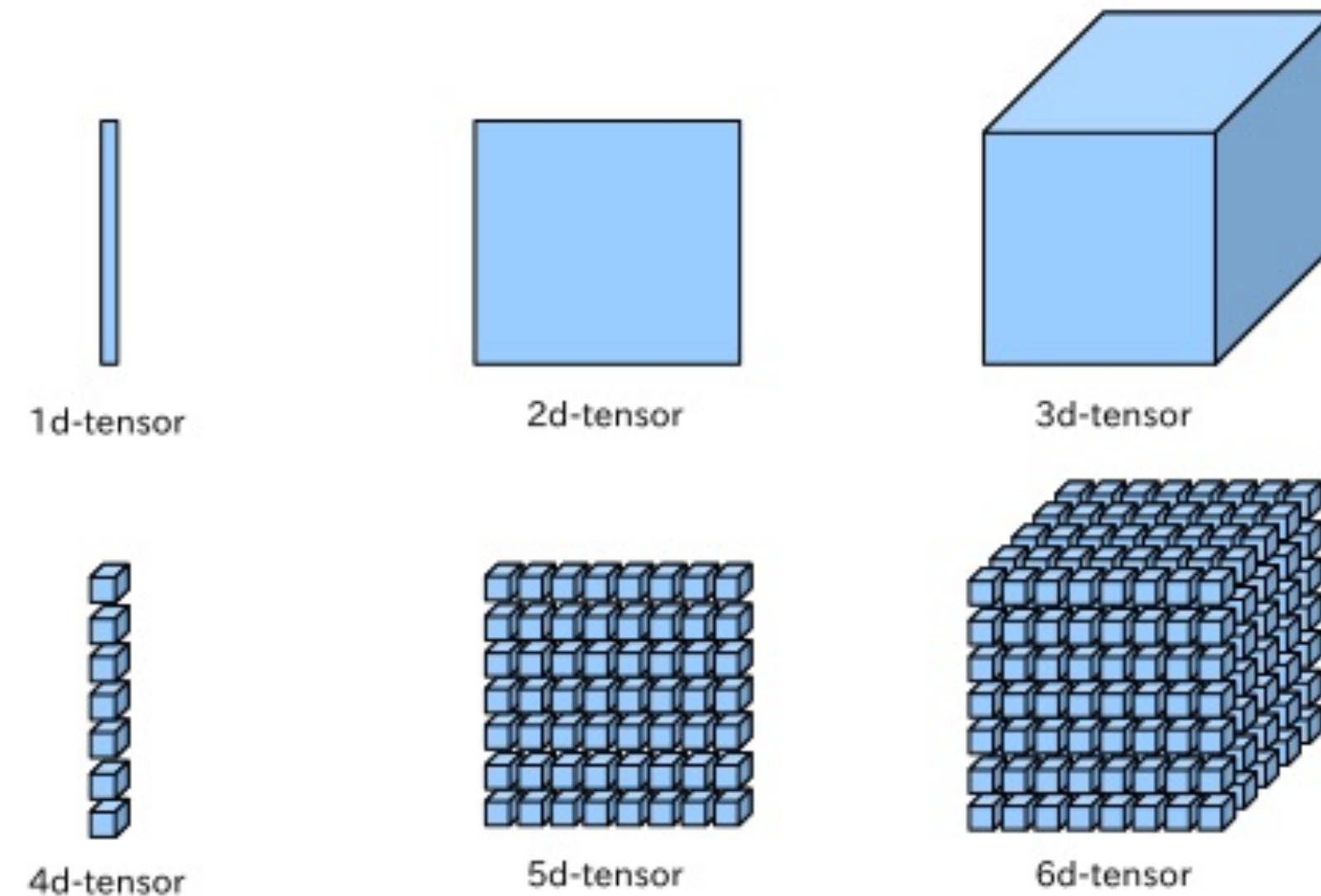
- \* “Image Style Transfer Using Convolutional neural networks” by Leon A. Gatys et al. (2016)
- \* “Generative handwriting using LSTM Mixture Density Network with TensorFlow” by hardmaru@GitHub (2016)
- \* “Wavenet: A generative model for raw audio” by Aaron van den Oord et al. (2016)

# What is Tensor

## □ Tensor (텐서)

- TensorFlow 내에서 모든 데이터를 표현하는 기본 자료 구조
- 동적 크기를 갖는 다차원 데이터 배열으로써 불리언(bool), 문자열, 정수형, 실수형 숫자 등 다양한 자료형의 데이터를 다룸
- **N-dimensional Matrix**

<Tensor Examples>



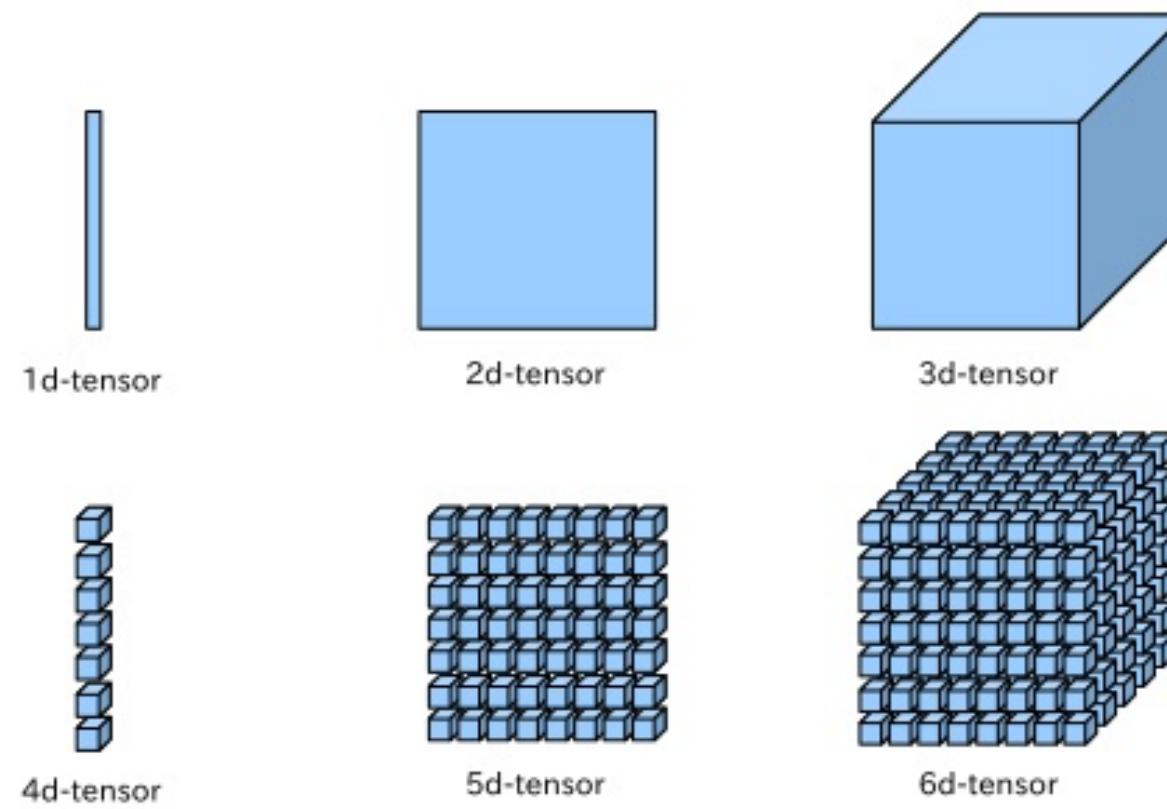
- **1D Matrix:** Vector
- **2D Matrix:** Matrix
- **Above 3D Matrix:** **Tensor!**

# Representation of Tensor Dimension

## Dimension of Tensor

- 구조 (Shape), 랭크 (Rank), 차원번호 (Dimension number) 세 종류의 명칭을 통해 텐서의 차원을 표기

### <Tensor Examples>

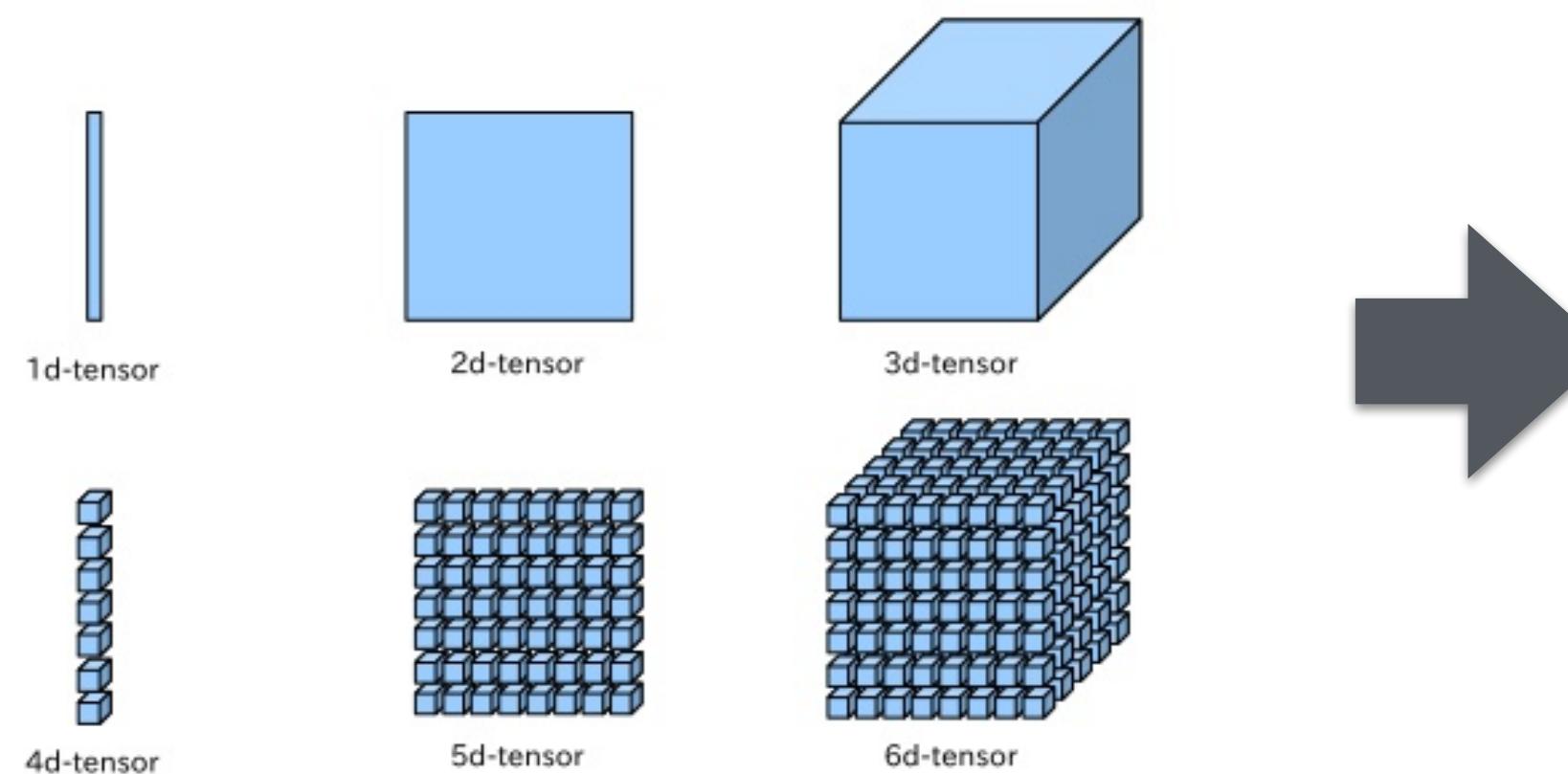


# Representation of Tensor Dimension

## Dimension of Tensor

- 구조 (Shape), 랭크 (Rank), 차원번호 (Dimension number) 세 종류의 명칭을 통해 텐서의 차원을 표기

### <Tensor Examples>

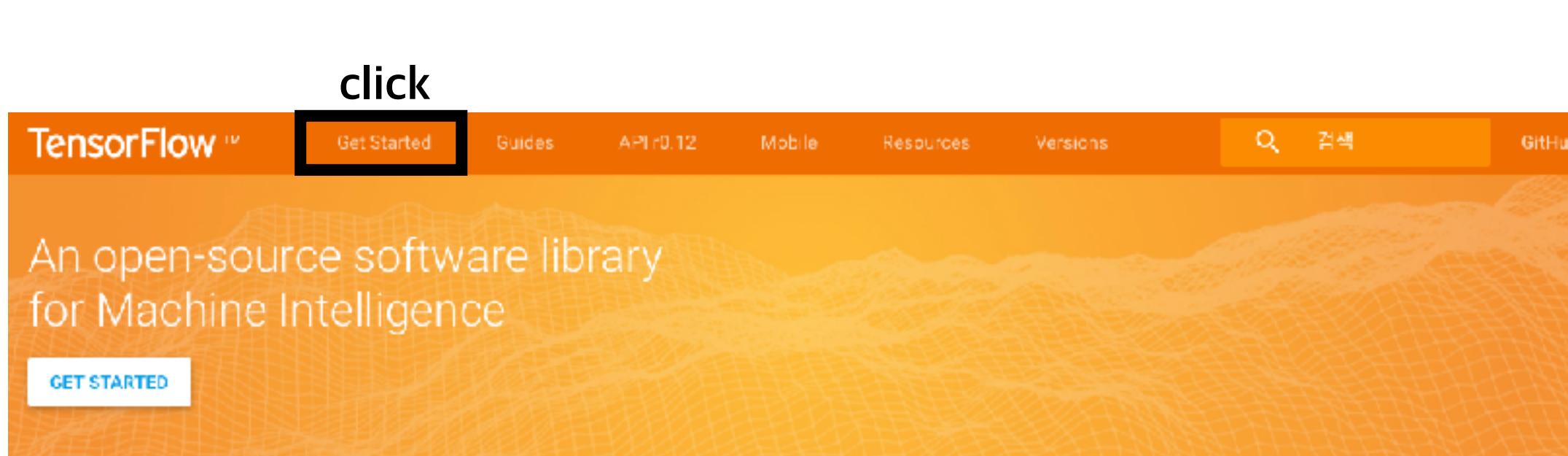


Python Code	Rank	Shape	Math Entity	Dimension number
483	0	[]	Scalar	0-D
[1.1, 2.2, 3.3]	1	[3]	Vector	1-D
[[1, 2, 3], [4, 5, 6]]	2	[2, 3]	Matrix	2-D
[[[2], [4], [6]], [[8], [10], [12]]]	3	[2, 3, 1]	3-Tensor	3-D
...	n	[D0, D1, ⋯ , Dn-1]	n-Tensor	n-D

# Installation

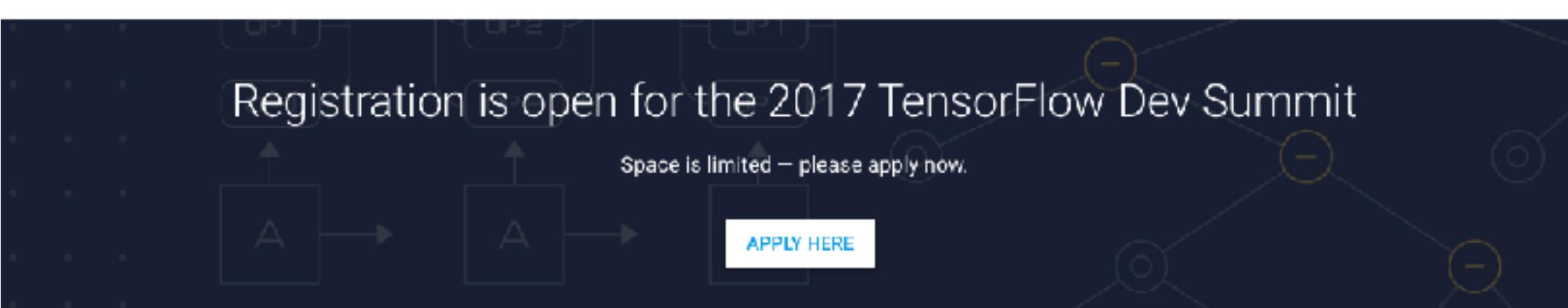
## □ TensorFlow 관련 대부분 매뉴얼은 TensorFlow 공식 홈페이지에서 접근 가능함

<https://www.tensorflow.org>



### About TensorFlow

TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.



### we will use this

#### Overview

We support different ways to install TensorFlow:

- [Pip install](#): Install TensorFlow on your machine, possibly upgrading previously installed Python packages. May impact existing Python programs on your machine.
- [Virtualenv install](#): Install TensorFlow in its own directory, not impacting any existing Python programs on your machine.
- [Anaconda install](#): Install TensorFlow in its own environment for those running the Anaconda Python distribution. Does not impact existing Python programs on your machine.
- [Docker install](#): Run TensorFlow in a Docker container isolated from all other programs on your machine.
- [Installing from sources](#): Install TensorFlow by building a pip wheel that you then install using pip.

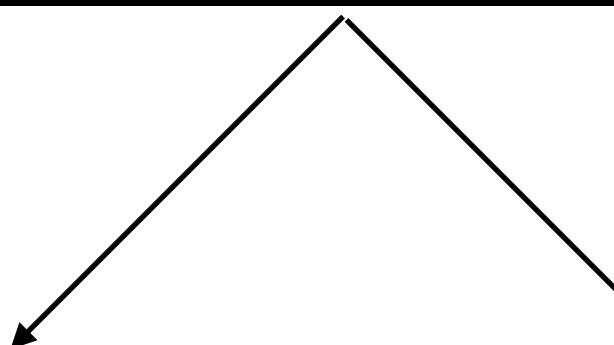
If you are familiar with Pip, Virtualenv, Anaconda, or Docker, please feel free to adapt the instructions to your particular needs. The names of the pip and Docker images are listed in the corresponding installation sections.

If you encounter installation errors, see [common problems](#) for some solutions.

## Import TensorFlow as tf

- Python 환경에서 TensorFlow는 대개 tf를 약어로 import 하여 사용함
- TensorFlow와 Numpy를 주로 동시에 import 하여 데이터 가공과 모델 구축을 동시에 진행
- Tensor 형태에서 입력/출력되는 데이터 타입은 Numpy의 Array와 일치하기 때문에 두 라이브러리를 함께 활용함

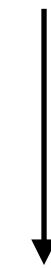
```
idoyeob-ui-MacBook-Pro:~ LDY$ python
Python 2.7.12 (default, Jun 29 2016, 14:05:02)
[GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow as tf
>>> import numpy as np
>>> █
```



# Various Operations in TensorFlow

## TensorFlow 내 다양한 연산들이 존재함

Operation	Description
tf.add	sum
tf.sub	subtraction
tf.mul	multiplication
tf.div	division
tf.mod	module
tf.abs	return the absolute value
tf.neg	return negative value
tf.sign	return the sign
tf.inv	returns the inverse
tf.square	calculates the square
tf.round	returns the nearest integer
tf.sqrt	calculates the square root
tf.pow	calculates the power
tf.exp	calculates the exponential
tf.log	calculates the logarithm
tf.maximum	returns the maximum
tf.minimum	returns the minimum
tf.cos	calculates the cosine
tf.sin	calculates the sine



# Various Operations in TensorFlow

## TensorFlow 내 다양한 연산들이 존재함

Operation	Description
tf.add	sum
tf.sub	subtraction
tf.mul	multiplication
tf.div	division
tf.mod	module
tf.abs	return the absolute value
tf.neg	return negative value
tf.sign	return the sign
tf.inv	returns the inverse
tf.square	calculates the square
tf.round	returns the nearest integer
tf.sqrt	calculates the square root
tf.pow	calculates the power
tf.exp	calculates the exponential
tf.log	calculates the logarithm
tf.maximum	returns the maximum
tf.minimum	returns the minimum
tf.cos	calculates the cosine
tf.sin	calculates the sine

모든 함수의 종류  
사용법을 외우고 자유자재로 사용하는 것은  
매우 숙련되지 않으면 사실상 불가능



구현시 필요한 결과에 따라  
알맞은 함수를 검색 + 사용법 검색

## □ TensorFlow 내 다양한 연산들이 존재함

tf.add

tf.add

```
add(  
    x,  
    y,  
    name=None  
)
```

Defined in [tensorflow/python/ops/gen\\_math\\_ops.py](#).

See the guide: [Math > Arithmetic Operators](#)

Returns  $x + y$  element-wise.

*NOTE:* `Add` supports broadcasting. `AddN` does not. More about broadcasting [here](#)

Args:

- `x`: A `Tensor`. Must be one of the following types: `half`, `float32`, `float64`, `uint8`, `int8`, `int16`, `int32`, `int64`, `complex64`, `complex128`, `string`.
- `y`: A `Tensor`. Must have the same type as `x`.
- `name` : A name for the operation (optional).

Returns:

A `Tensor`. Has the same type as `x`.

# Various Operations in TensorFlow

## TensorFlow 내 다양한 연산들이 존재함

tf.add

tf.add

```
add(  
    x,  
    y,  
    name=None  
)
```

Defined in [tensorflow/python/ops/gen\\_math\\_ops.py](#).

See the guide: [Math > Arithmetic Operators](#)

Returns  $x + y$  element-wise.

NOTE: `Add` supports broadcasting. `AddN` does not. More about broadcasting [here](#)

Args:

- `x`: A `Tensor`. Must be one of the following types: `half`, `float32`, `float64`, `uint8`, `int8`, `int16`, `int32`, `int64`, `complex64`, `complex128`, `string`.
- `y`: A `Tensor`. Must have the same type as `x`.
- `name` : A name for the operation (optional).

Returns:

A `Tensor`. Has the same type as `x`.

Tensorflow 홈페이지에 모든 정보가 존재함  
해당 문서를 읽고 이해할 수 있는 능력이 중요

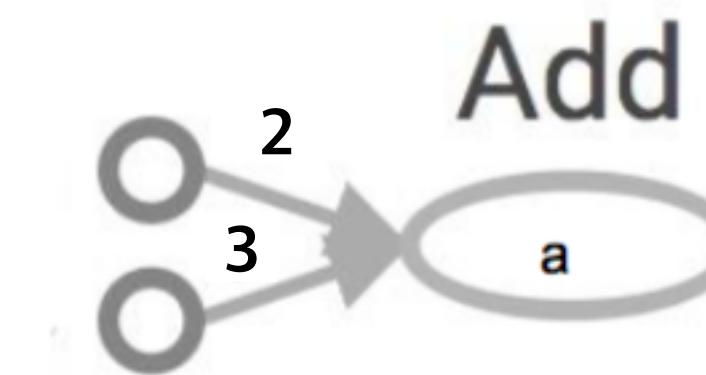
# Operating Process Example

- 더하기 연산을 하는 간단한 TensorFlow 프로그램이 있다고 가정하자 ( $2+3 = 5$ )

```
>>> a = tf.add(2,3)  
>>> print a
```

What is the Result?

```
>>> print a  
Tensor("Add_1:0", shape=(), dtype=int32)
```



→ It is the **Type of Result Tensor**

**Don't Forget**  
the Characteristics of TensorFlow!

## Programming Process

### 1) (Only) Build Graph

- 머신러닝 모델과 데이터의 흐름들의 계층적인 구조 (hierarchical structure)를 구현

### 2) Executing the Graph Model

- 구현된 Graph에 따라 데이터를 흘려보내고, 계산을 실행

→ use  
**tf.Session**

## tf.Session()

### □ (\*중요) tf.Session()을 통해 실제 구현된 머신러닝 모델의 그래프를 실행시킬 수 있음

- 다양한 Operation 객체들이 실행되고 Tensor 값이 계산되도록 하는 객체
- tf.Session 내 run() 메서드를 호출함으로써 모델이 실행됨
- 하나의 Session은 하나의 창 또는 프로그램을 의미한다고 생각하는게 바람직함

(예를 들어 한 프로그램 내에서 여러 개의 Session을 선언하고 이용하는 것은 해당 그래프 모델을 실행시키는 여러 개의 창을 띄우는 것과 같음)

```
>>> a = tf.add(2,3)
>>> sess = tf.Session()
>>> sess.run(a)
5
>>> a
<tf.Tensor 'Add_2:0' shape=() dtype=int32>
```

```
>>> a = tf.add(2,3)
>>> with tf.Session() as sess:
...     sess.run(a)
...
5
```

# tf.Session.run()

- tf.Session.run의 대상인 fetches는 하나의 오퍼레이션, 텐서 등을 포함해 리스트(list)와 딕셔너리(dictionary) 형식으로 실행할 수 있음

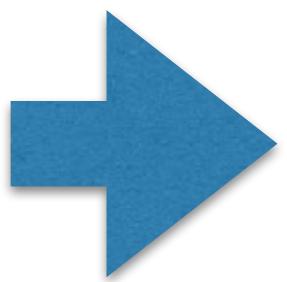
## <tf.Session.run(args) 목록>

run

```
run(  
    fetches,  
    feed_dict=None,  
    options=None,  
    run_metadata=None  
)
```

Args:

- fetches : A single graph element, a list of graph elements, or a dictionary whose values are graph elements or lists of graph elements (described above).
- feed\_dict : A dictionary that maps graph elements to values (described above).
- options : A [ RunOptions ] protocol buffer
- run\_metadata : A [ RunMetadata ] protocol buffer



## <Example of sess.run with list>

```
a = 1  
b = 2
```

```
_add = tf.add(a,b)  
_mul = tf.mul(a,b)  
sess = tf.Session()  
  
#print 'add' and 'mul' (3 2)  
print sess.run(_add), sess.run(_mul)  
print sess.run([_add,_mul])
```

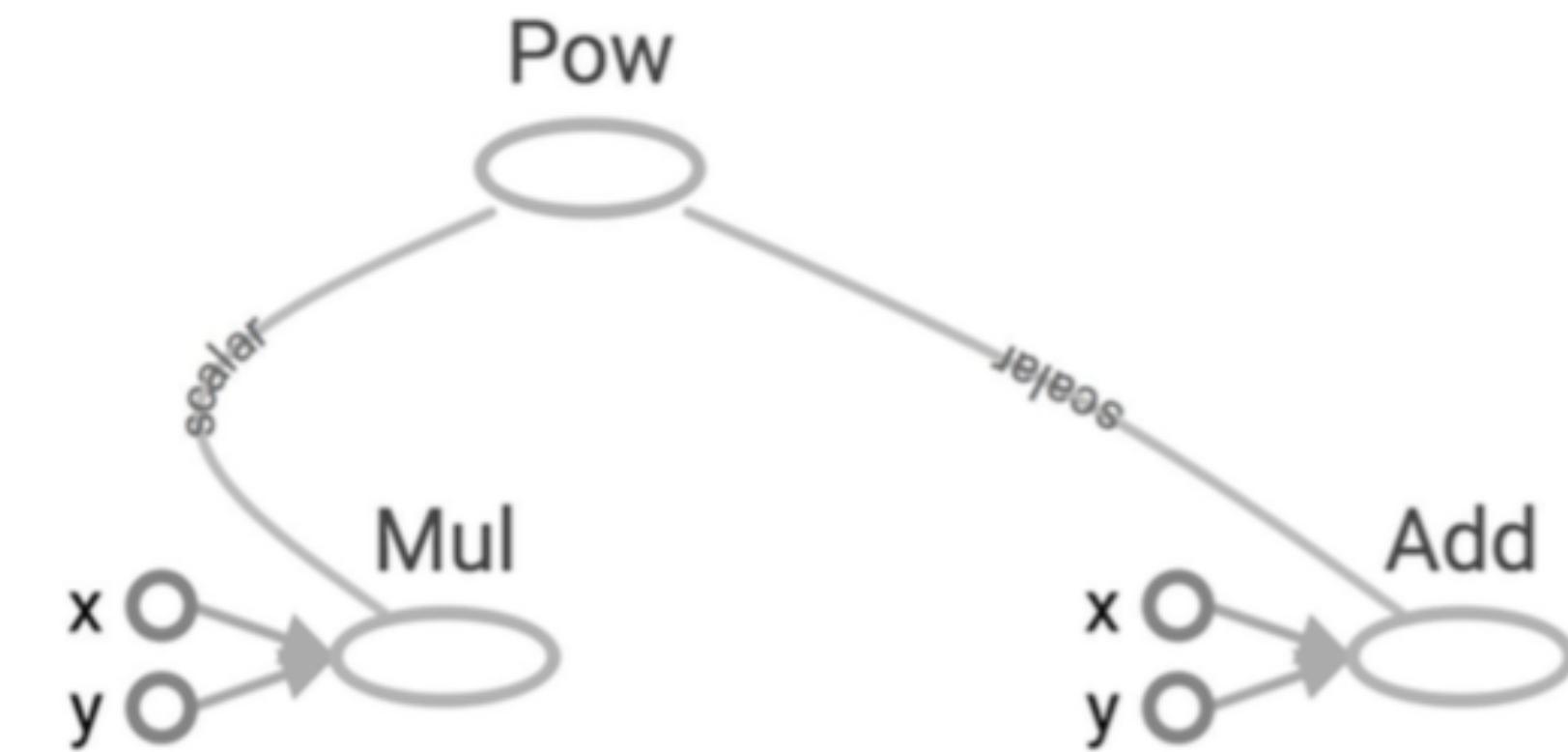
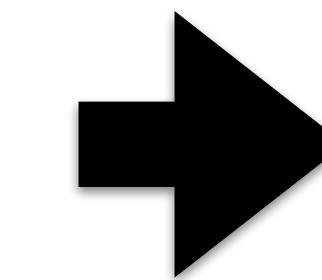
```
MDOM1207-2:~ LDY$ python tf_test.py  
W tensorflow/core/platform/cpu_feature_g  
but these are available on your machine  
W tensorflow/core/platform/cpu_feature_g  
but these are available on your machine  
W tensorflow/core/platform/cpu_feature_g  
these are available on your machine and  
W tensorflow/core/platform/cpu_feature_g  
t these are available on your machine and  
W tensorflow/core/platform/cpu_feature_g  
these are available on your machine and  
3 2  
[3, 2]
```

# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```

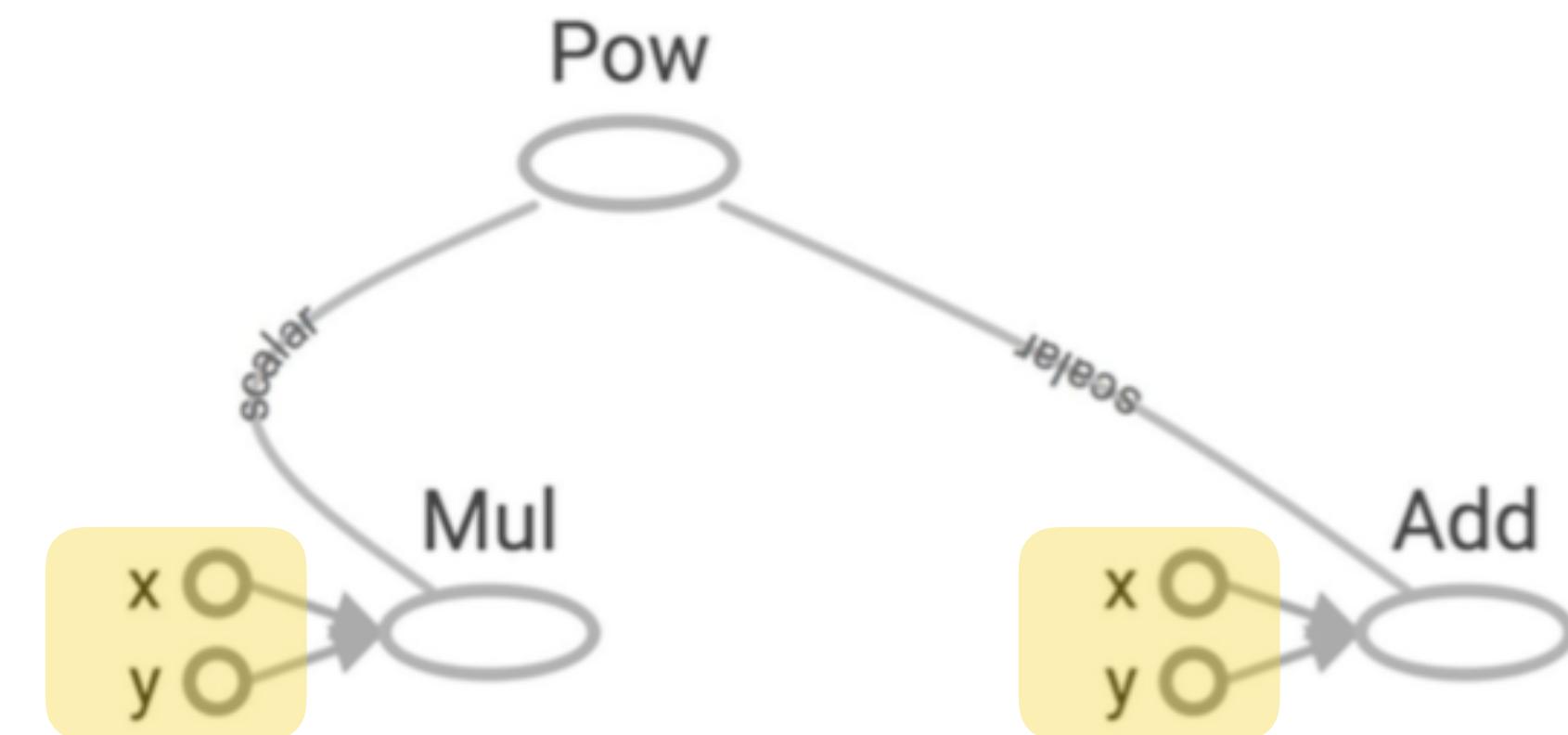
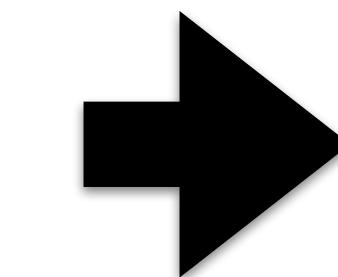


# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```



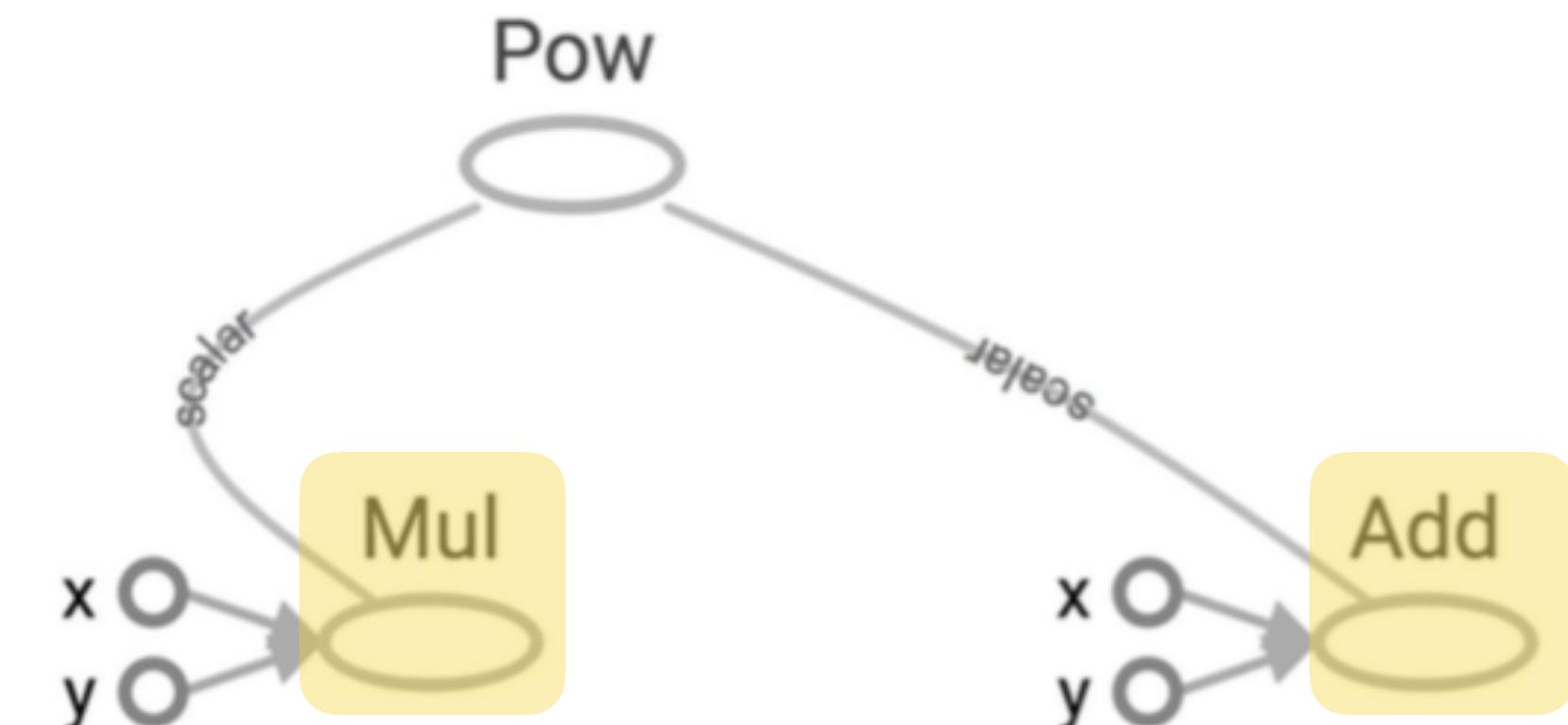
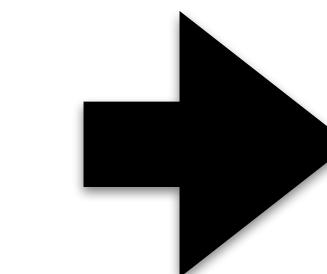
# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

- **Node:** Operation
- **Edge:** Tensor

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```



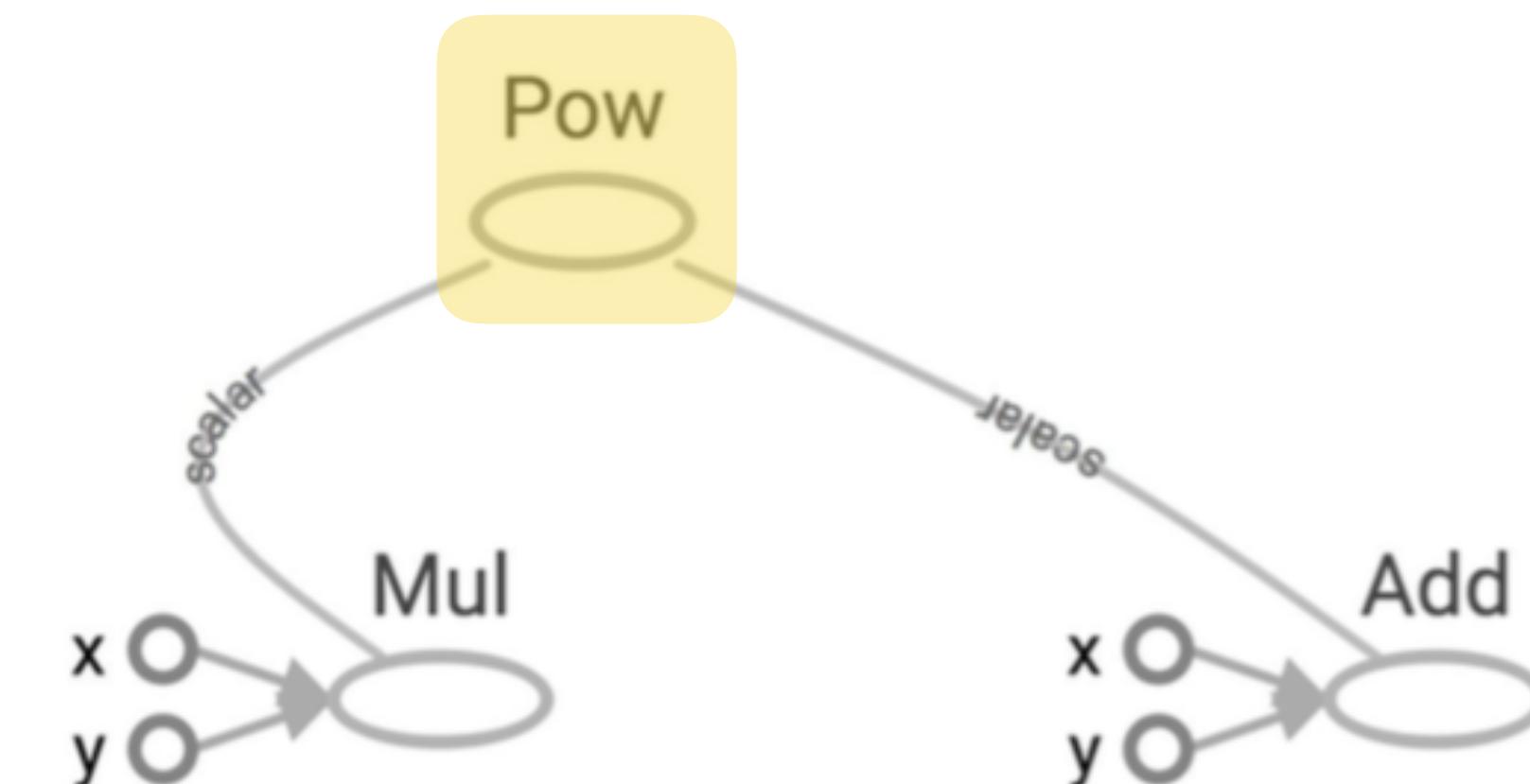
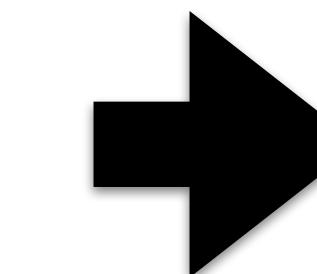
# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

- **Node:** Operation
- **Edge:** Tensor

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```



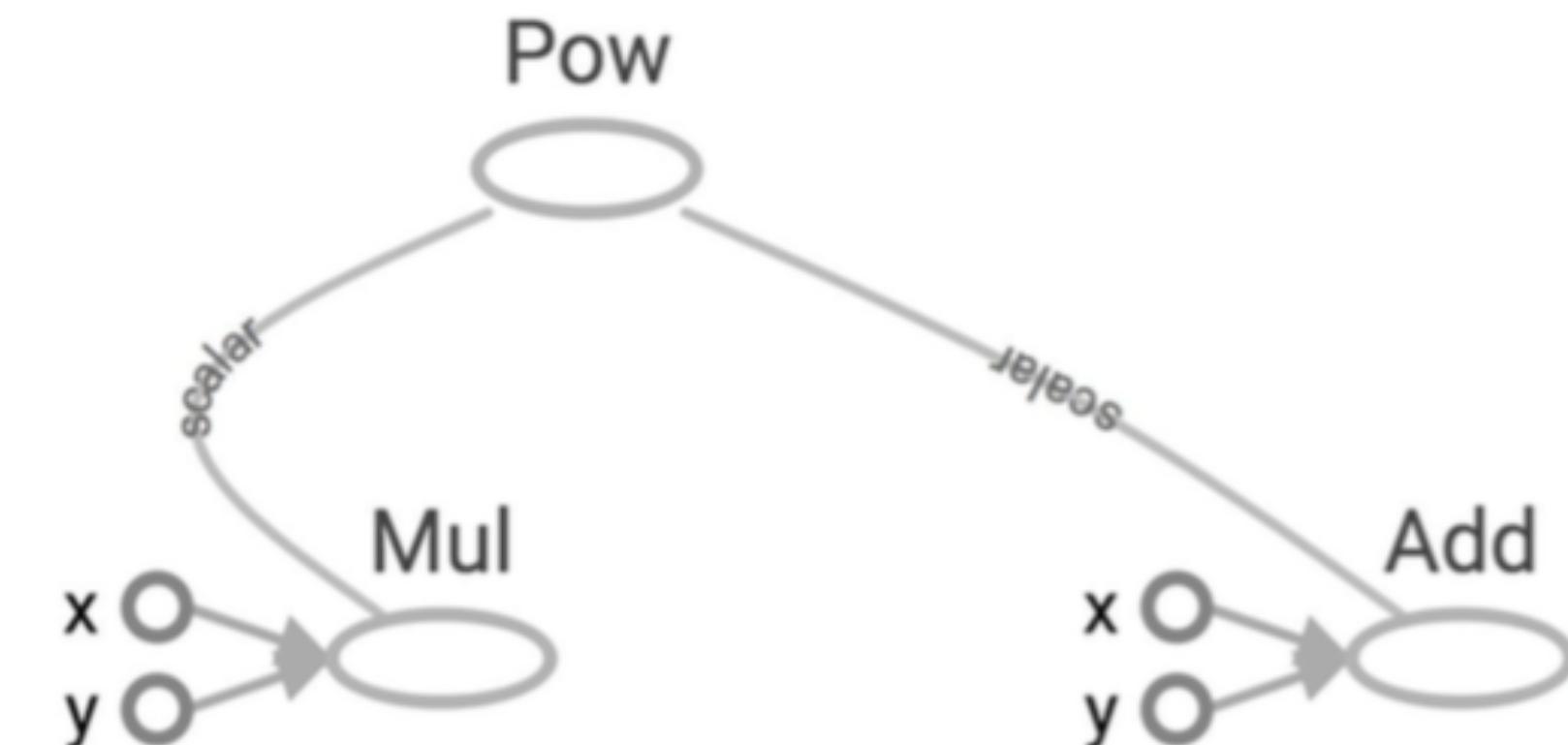
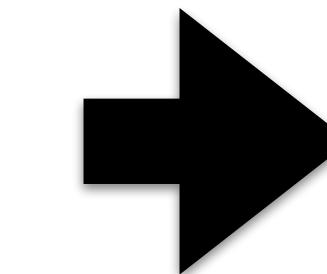
# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

- **Node:** Operation
- **Edge:** Tensor

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```



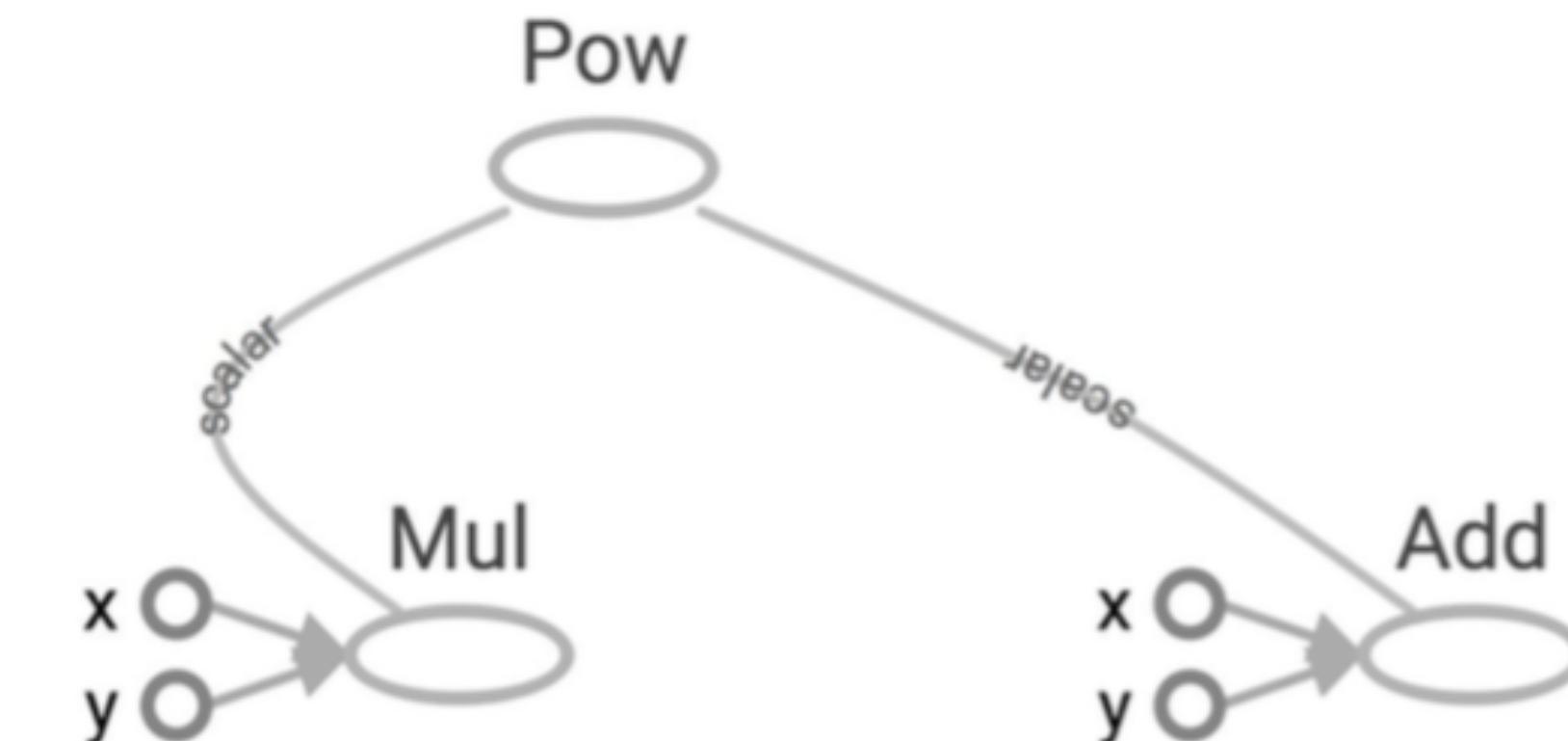
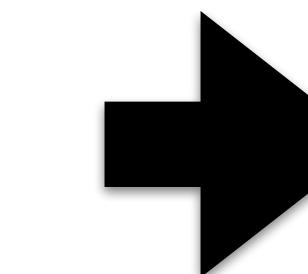
# Example of More Graphs in TensorFlow (1)

## □ 다양한 과정의 계산 예시

- 1) x, y 값을 순차적으로 입력 받음
- 2) x, y 값의 합(add)과 곱(mul)을 구함
- 3) 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

- **Node:** Operation
- **Edge:** Tensor

```
>>> import tensorflow as tf
>>> x = 2
>>> y = 3
>>> op1 = tf.add(x,y)
>>> op2 = tf.mul(x,y)
>>> op3 = tf.pow(op2,op1)
>>> with tf.Session() as sess:
...     op3 = sess.run(op3)
...
>>> op3
7776
```



\*We **only run op3** (not all op3, op2, and op1),  
because op3 is connected to op2 and op1.

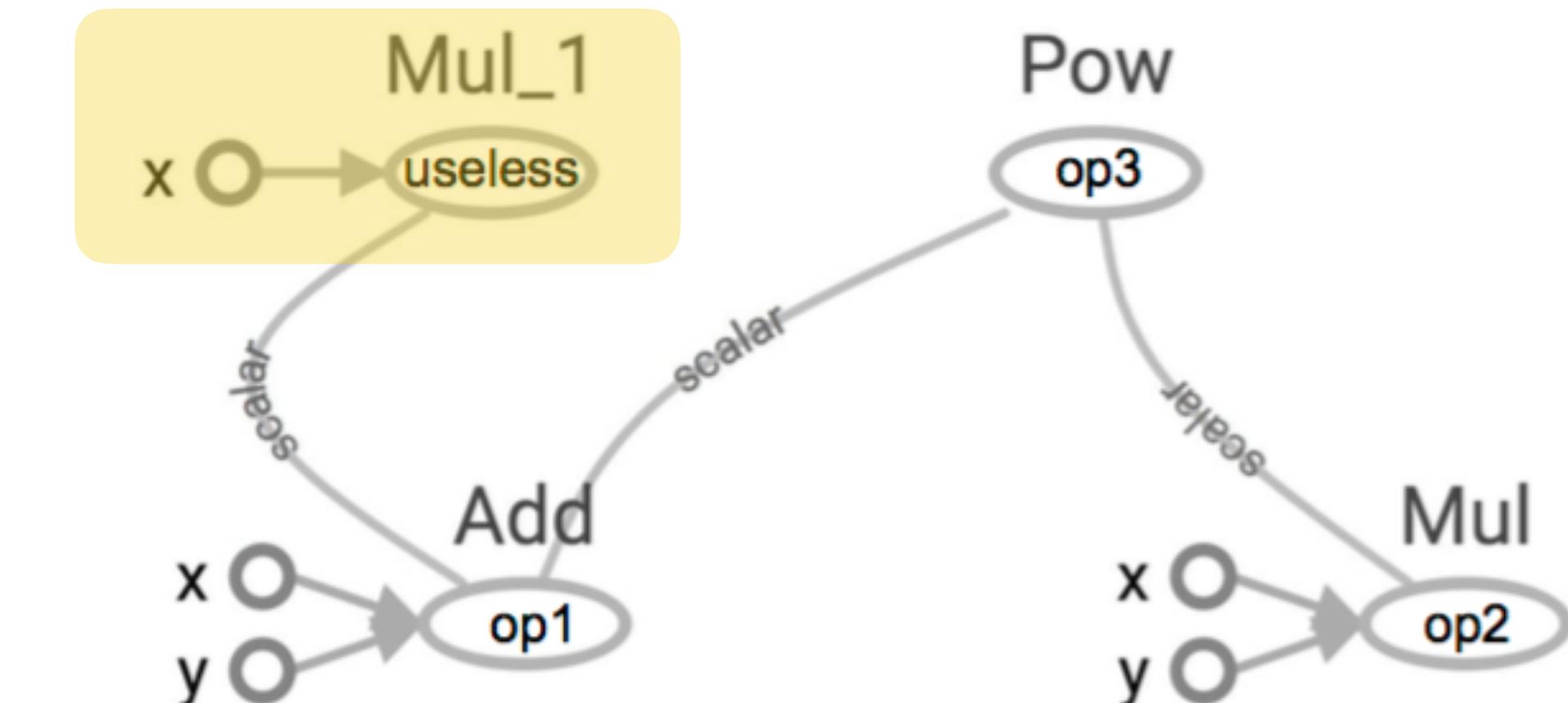
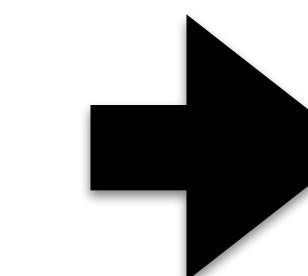
>> op3를 계산하기 위하여 op2와 op1가 연결되어 (우선적으로) 실행됨

## Example of More Graphs in TensorFlow (2)

- TensorFlow Session 실행시 계산은 필요에 의해 효율적으로 계산함
- X와 op1 (Add) 결과를 통해 곱셈을 계산하는 Useless 연산을 그래프에 추가했을 경우

- x, y 값을 순차적으로 입력 받음
- x, y 값의 합(add)과 곱(mul)을 구함
- x와 2)의 합(add) 사이의 곱인 **useless**를 계산함
- 곱(mul)의 합(add) 거듭 제곱 값을 계산 ( $\text{mul}^{\text{add}}$ )

```
>>> import tensorflow as tf  
>>> x = 2  
>>> y = 3  
[REDACTED]  
>>> op1 = tf.add(x,y)  
>>> op2 = tf.mul(x,y)  
>>> useless = tf.mul(x, op1)  
>>> op3 = tf.pow(op2,op1)  
>>>  
[REDACTED]  
>>> with tf.Session() as sess:  
...     op3 = sess.run(op3)  
...  
>>> op3  
7776
```



Should we calculate  $\text{useless} = \text{tf.mul}(x, \text{op1})$  in left program?

\*Not Calculate Useless in Session!  
(If we want)

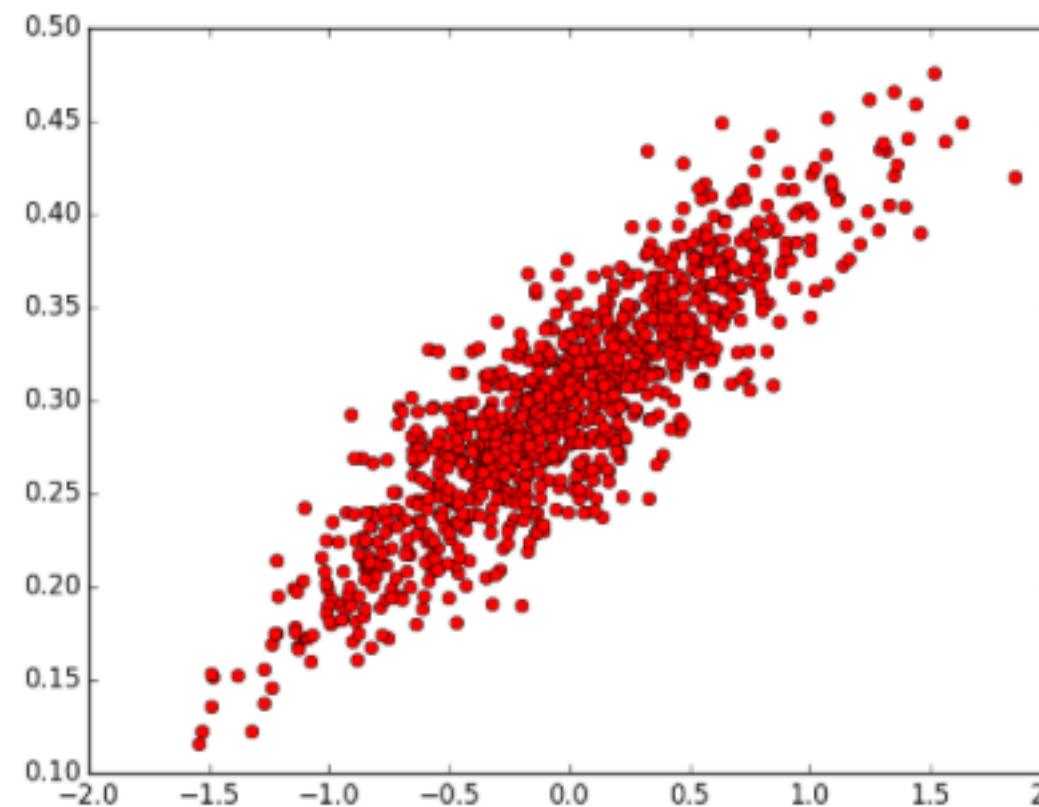
`tf.Session.run(fetches, feed_dict=None,  
options=None, run_metadata=None)`

```
>>> with tf.Session() as sess:  
...     op3, not_useless = sess.run([op3,useless])  
...  
>>> op3  
7776  
>>> not_useless  
10
```

# Linear Regression in TensorFlow

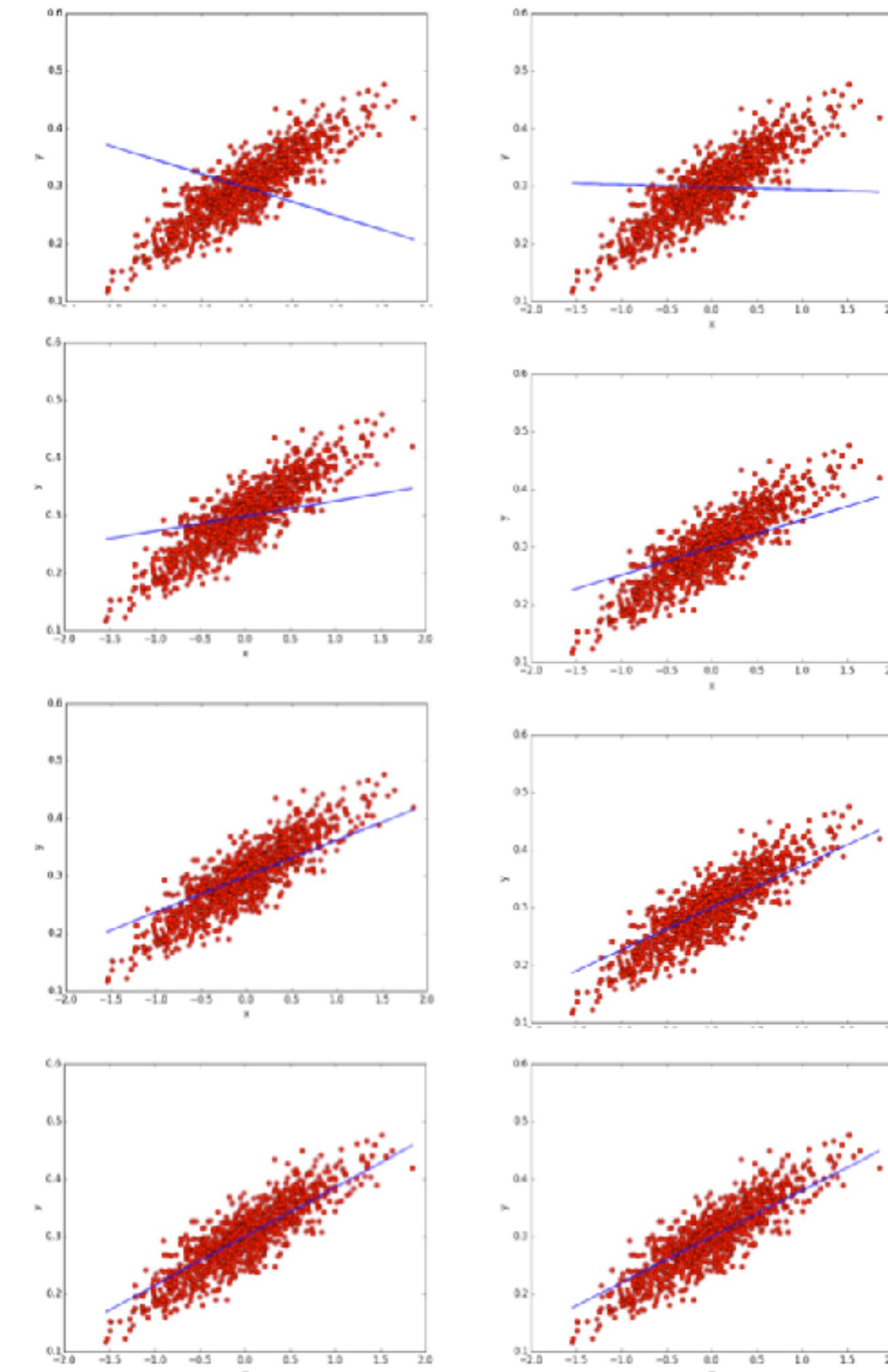
- Dataset의 변수들 사이의 관계를 가장 잘 설명하고 예측하는 선형회귀식(Linear Regressor)을 찾아보자

<Sample Dataset>



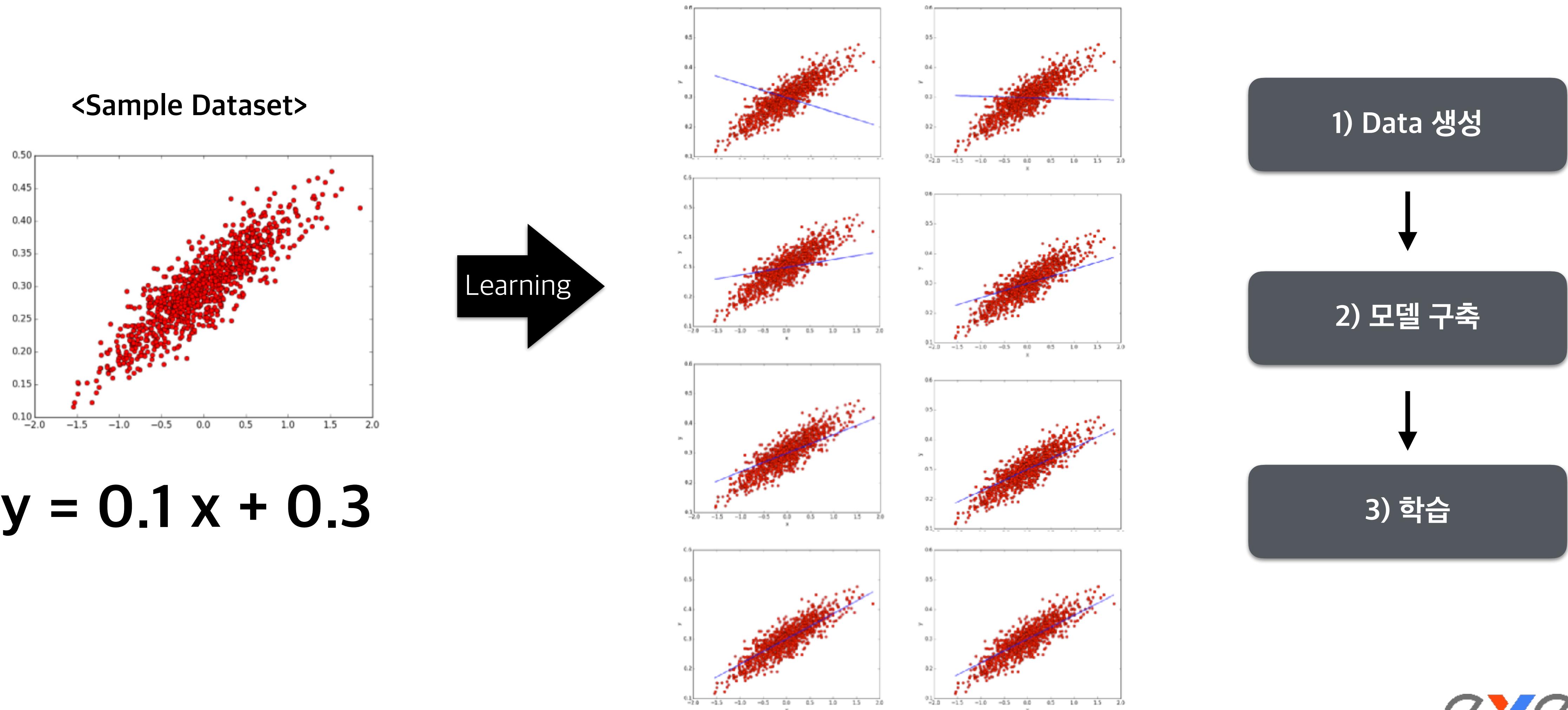
Learning

$$y = 0.1x + 0.3$$



# Linear Regression in TensorFlow

- Dataset의 변수들 사이의 관계를 가장 잘 설명하고 예측하는 선형회귀식(Linear Regressor)을 찾아보자



# Sample Code for Linear Regression

## □ Data Generation 단계

```
In [1]: import numpy as np

num_points = 1000
vectors_set = []
for i in range(num_points):
    x1=np.random.normal(0.0, 0.55)
    y1 = x1* 0.1 + 0.3 + np.random.normal(0.,0.03)
    vectors_set.append([x1,y1])

x_data = [v[0] for v in vectors_set]
y_data = [v[1] for v in vectors_set]

import matplotlib.pyplot as plt

plt.plot(x_data, y_data, 'ro')
plt.legend()
plt.show()
```

# Sample Code for Linear Regression

## □ Data Generation 단계

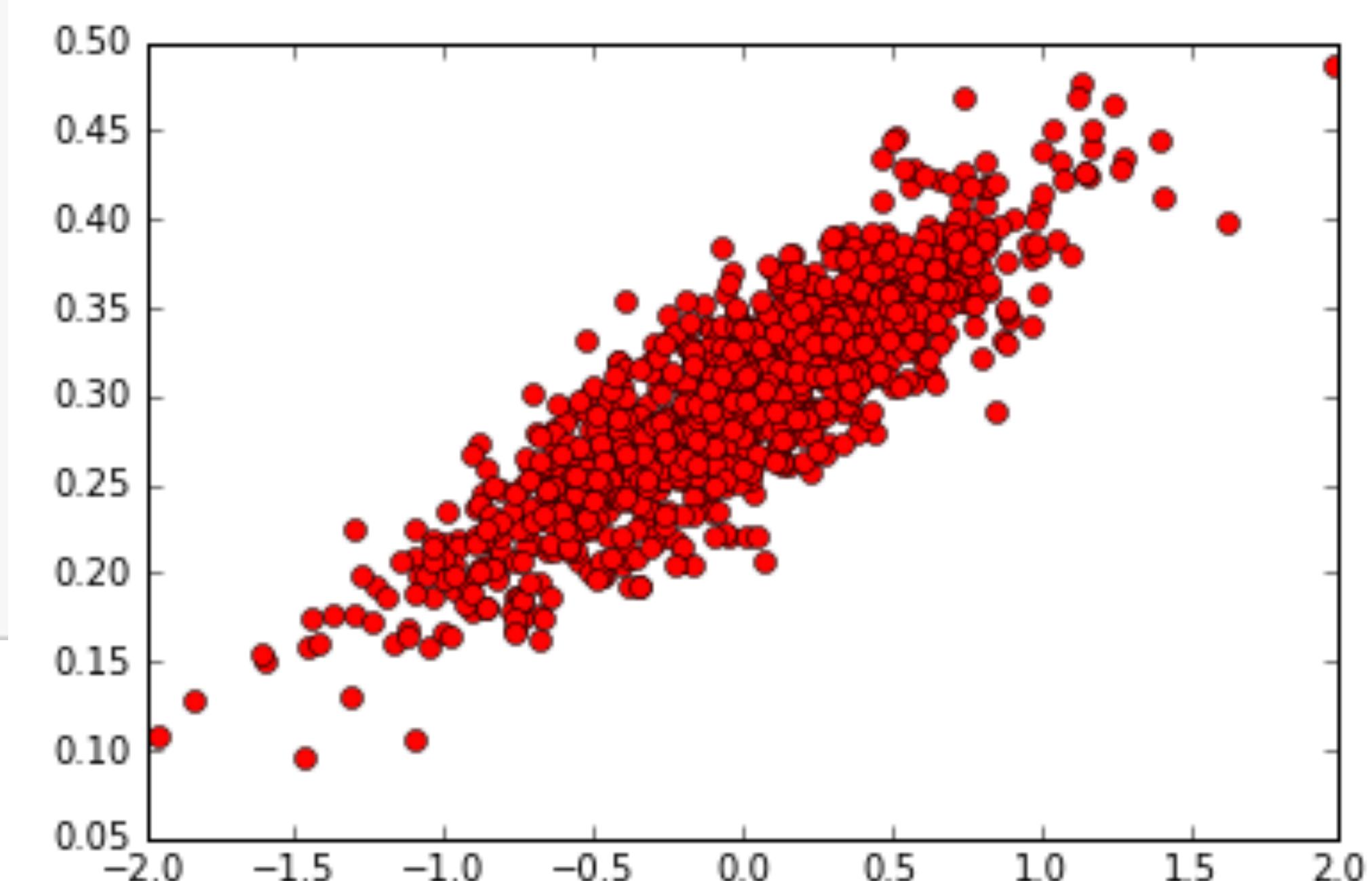
```
In [1]: import numpy as np

num_points = 1000
vectors_set = []
for i in range(num_points):
    x1=np.random.normal(0.0, 0.55)
    y1 = x1* 0.1 + 0.3 + np.random.normal(0.,0.03)
    vectors_set.append([x1,y1])

x_data = [v[0] for v in vectors_set]
y_data = [v[1] for v in vectors_set]

import matplotlib.pyplot as plt

plt.plot(x_data, y_data, 'ro')
plt.legend()
plt.show()
```



# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
In [5]: import tensorflow as tf  
  
w = tf.Variable(tf.random_uniform([1], -1.0, 1.0))  
b = tf.Variable(tf.zeros([1]))  
y = W*x_data + b  
  
loss = tf.reduce_mean(tf.square(y-y_data))  
optimizer = tf.train.GradientDescentOptimizer(0.5)  
train = optimizer.minimize(loss)  
  
init = tf.global_variables_initializer()  
  
sess= tf.Session()  
sess.run(init)
```

기울기 W

절편 b

**tf.Variable**로 선언  
(학습의 대상)

# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

In [5]:

```
import tensorflow as tf

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

기울기 W

절편 b

**tf.Variable**로 선언  
(학습의 대상)



변수

초기값 (constant)  
설정 필요  
(Initialization)

# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
In [5]: import tensorflow as tf

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

x\_data

Model

Output

# Sample Code for Linear Regression

□ 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

In [5]:

```
import tensorflow as tf

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

(중요)

Learning Part

- Cost Function (Loss)
- Optimizer (GD)
- Optimization Criteria

# Sample Code for Linear Regression

□ 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
In [5]: import tensorflow as tf

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

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

(중요)  
변수 초기화  
실행

- **tf.global\_variables\_initializer()**  
: tf.Variable 들이 지정된 초기값으로  
초기화를 실행하는 메소드

## Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
In [5]: import tensorflow as tf  
  
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))  
b = tf.Variable(tf.zeros([1]))  
y = W*x_data + b
```

# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

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

# Sample Code for Linear Regression

- 모델 구축 단계 (\*Tensorflow!) >> 일단 사용법을 받아들여 봅시다!

```
In [5]: import tensorflow as tf
```

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b
```

모델 구축

```
loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)
```

학습 환경 설정  
(Loss, Optimizer)

```
init = tf.global_variables_initializer()

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

변수 초기화

# Sample Code for Linear Regression

## □ 학습 단계

```
In [6]: #Train several steps  
  
for step in range(10):  
    sess.run(train)  
    print("Step\t W\t b\t")  
    print(step, sess.run(W), sess.run(b))  
    print("Loss Value")  
    print(step, sess.run(loss))
```

**train** [=optimizer.minimize(loss)]  
만 실행시켜 주면 됩니다!

# Sample Code for Linear Regression

## □ 학습 단계

```
In [6]: #Train several steps

for step in range(10):
    sess.run(train)
    print("Step\t W\t b\t")
    print(step, sess.run(W), sess.run(b))
    print("Loss Value")
    print(step, sess.run(loss))
```

**train** [=optimizer.minimize(loss)]  
만 실행시켜 주면 됩니다!

```
In [5]: import tensorflow as tf

W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b

loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

# Sample Code for Linear Regression

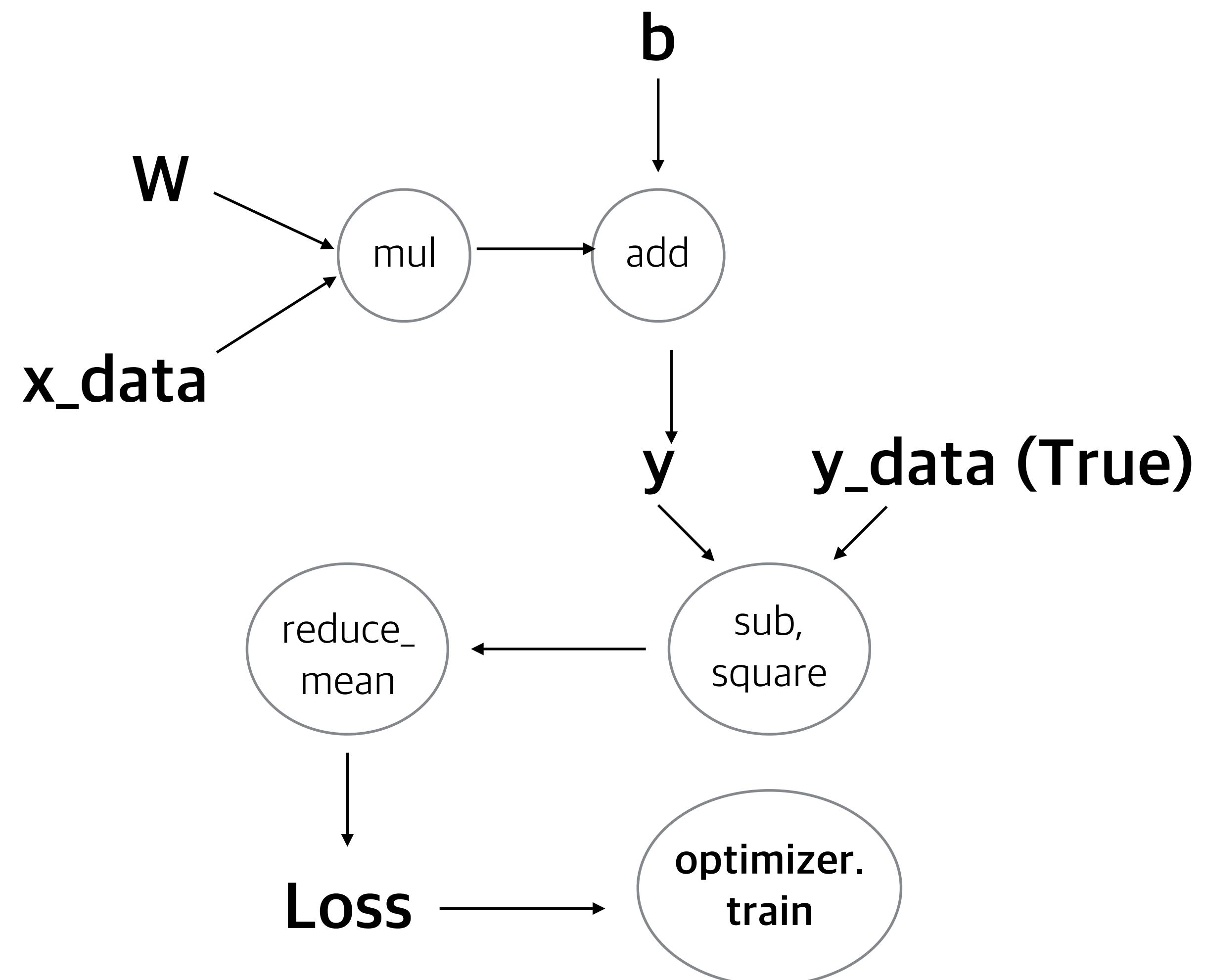
## □ 학습 단계

```
In [6]: #Train several steps  
  
for step in range(10):  
    sess.run(train)  
    print("Step\tW\tb\t")  
    print(step, sess.run(W), sess.run(b))  
    print("Loss Value")  
    print(step, sess.run(loss))
```

**train** [=optimizer.minimize(loss)]  
만 실행시켜 주면 됩니다!

```
In [5]: import tensorflow as tf  
  
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))  
b = tf.Variable(tf.zeros([1]))  
y = W*x_data + b  
  
loss = tf.reduce_mean(tf.square(y - y_data))  
optimizer = tf.train.GradientDescentOptimizer(0.5)  
train = optimizer.minimize(loss)  
  
init = tf.global_variables_initializer()  
  
sess = tf.Session()  
sess.run(init)
```

## <구축된 그래프 모델 예시>



# Sample Code for Linear Regression

## 학습 단계

```
In [6]: #Train several steps
```

```
for step in range(10):
    sess.run(train)
    print("Step\tW\tb\t")
    print(step, sess.run(W), sess.run(b))
    print("Loss Value")
    print(step, sess.run(loss))
```

**train** [=optimizer.minimize(loss)]  
만 실행시켜 주면 됩니다!

```
In [5]: import tensorflow as tf
```

```
W = tf.Variable(tf.random_uniform([1], -1.0, 1.0))
b = tf.Variable(tf.zeros([1]))
y = W*x_data + b
loss = tf.reduce_mean(tf.square(y-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

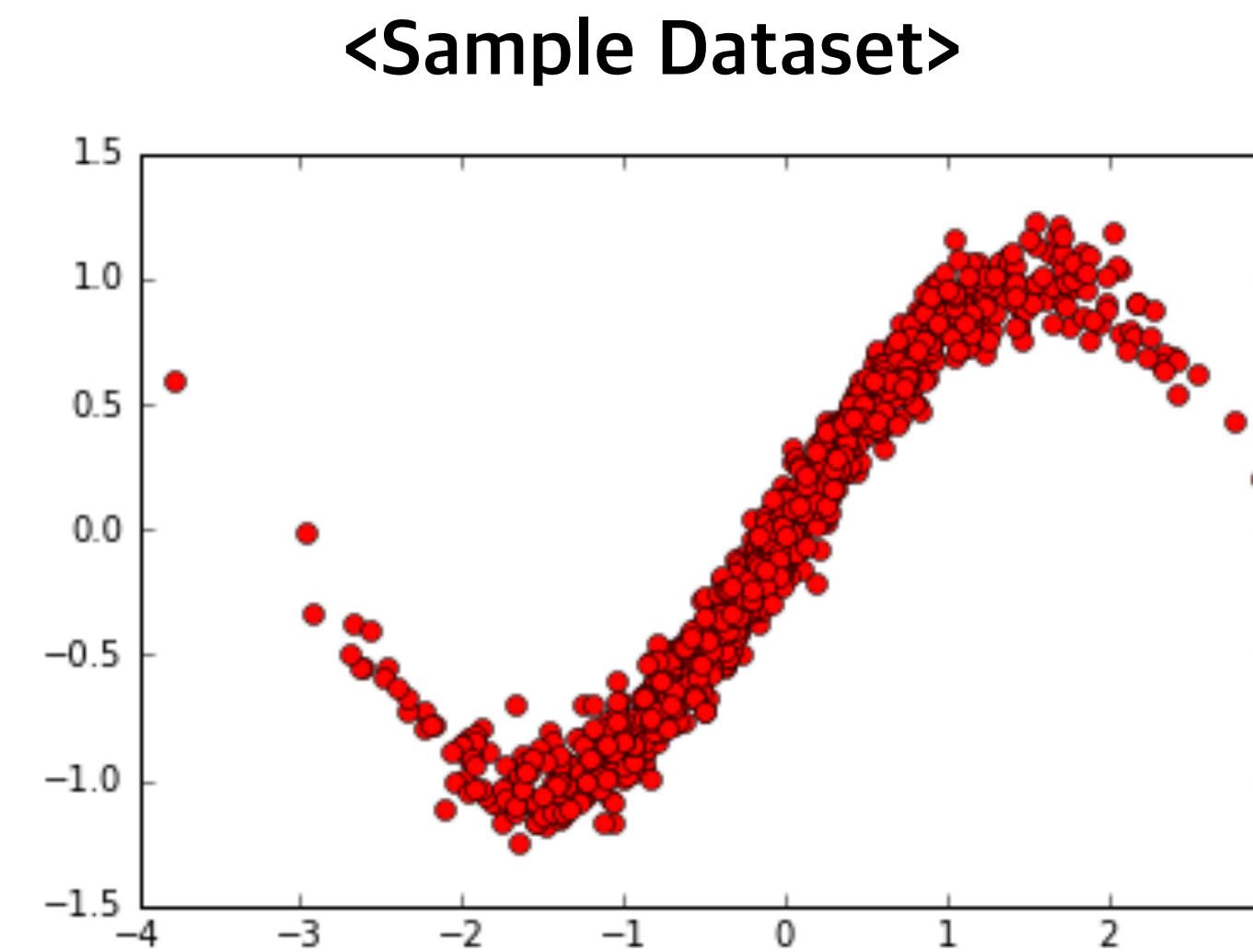
```
Step      W      b
(0, array([-0.34587264], dtype=float32), array([ 0.29445165], dtype=float32))
Loss Value
(0, 0.058301292)
Step      W      b
(1, array([-0.21731669], dtype=float32), array([ 0.29642794], dtype=float32))
Loss Value
(1, 0.030005457)
Step      W      b
(2, array([-0.12582], dtype=float32), array([ 0.29787126], dtype=float32))
Loss Value
(2, 0.01567189)
Step      W      b
(3, array([-0.06069896], dtype=float32), array([ 0.29889852], dtype=float32))
Loss Value
(3, 0.0084110601)
Step      W      b
(4, array([-0.01435029], dtype=float32), array([ 0.29962966], dtype=float32))
Loss Value
(4, 0.0047330046)
Step      W      b
(5, array([ 0.0186375], dtype=float32), array([ 0.30015004], dtype=float32))
Loss Value
(5, 0.0028698463)
Step      W      b
(6, array([ 0.04211594], dtype=float32), array([ 0.30052039], dtype=float32))
Loss Value
(6, 0.0019260412)
Step      W      b
(7, array([ 0.05882628], dtype=float32), array([ 0.30078399], dtype=float32))
Loss Value
(7, 0.0014479463)
Step      W      b
(8, array([ 0.07071954], dtype=float32), array([ 0.30097163], dtype=float32))
Loss Value
(8, 0.0012057624)
Step      W      b
(9, array([ 0.07918435], dtype=float32), array([ 0.30110514], dtype=float32))
Loss Value
(9, 0.0010830811)
```

1 more time !

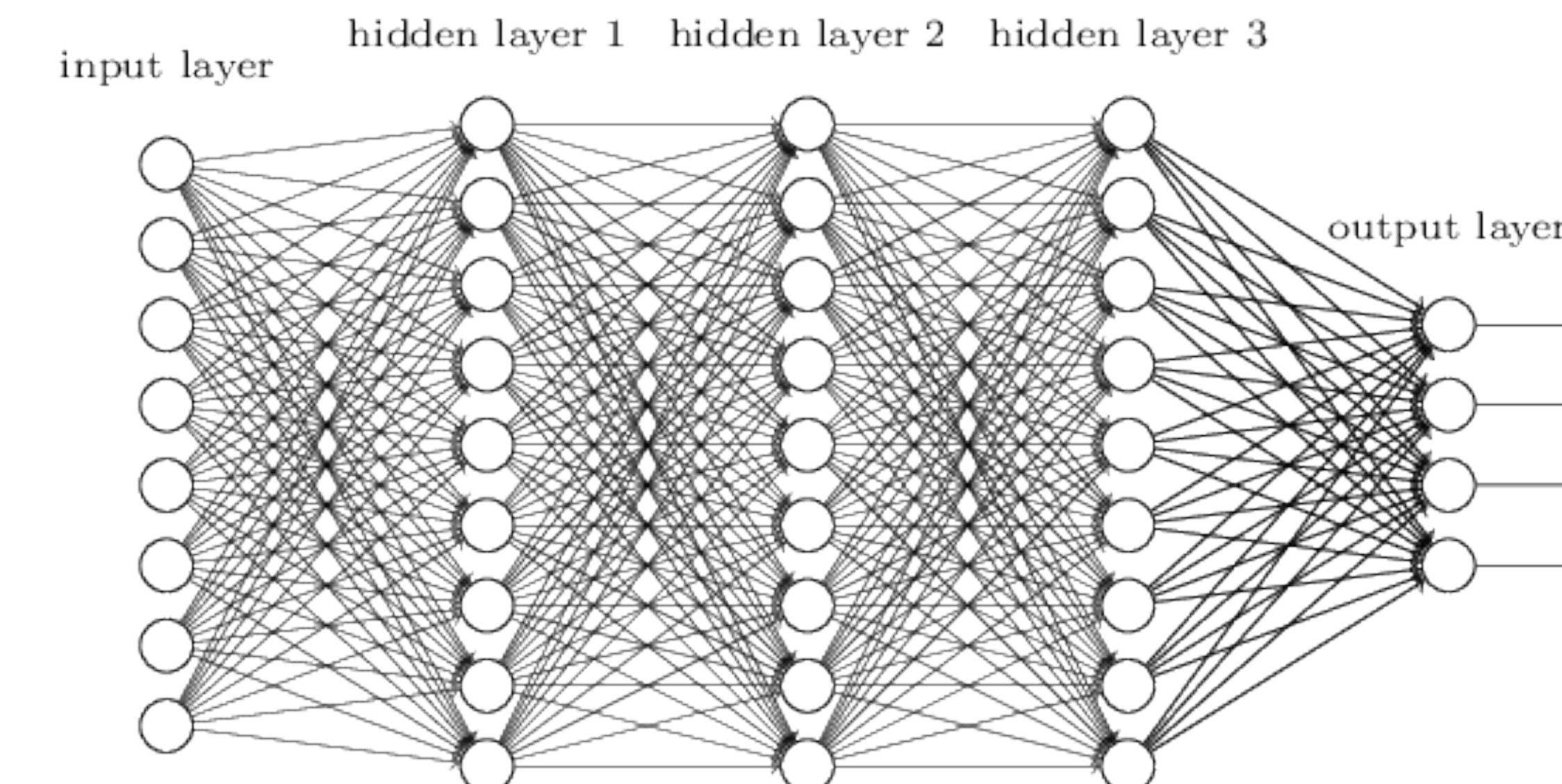
지금까지 내용을 다시 익혀볼 겸!

다른 데이터로 한 번 더 해볼까요~~~~?

- Dataset의 변수들 사이의 관계를 가장 잘 설명하고 예측하는 인공 신경망(Neural Network)을 찾아보자



$$y = \sin(x)$$



## □ 데이터 준비

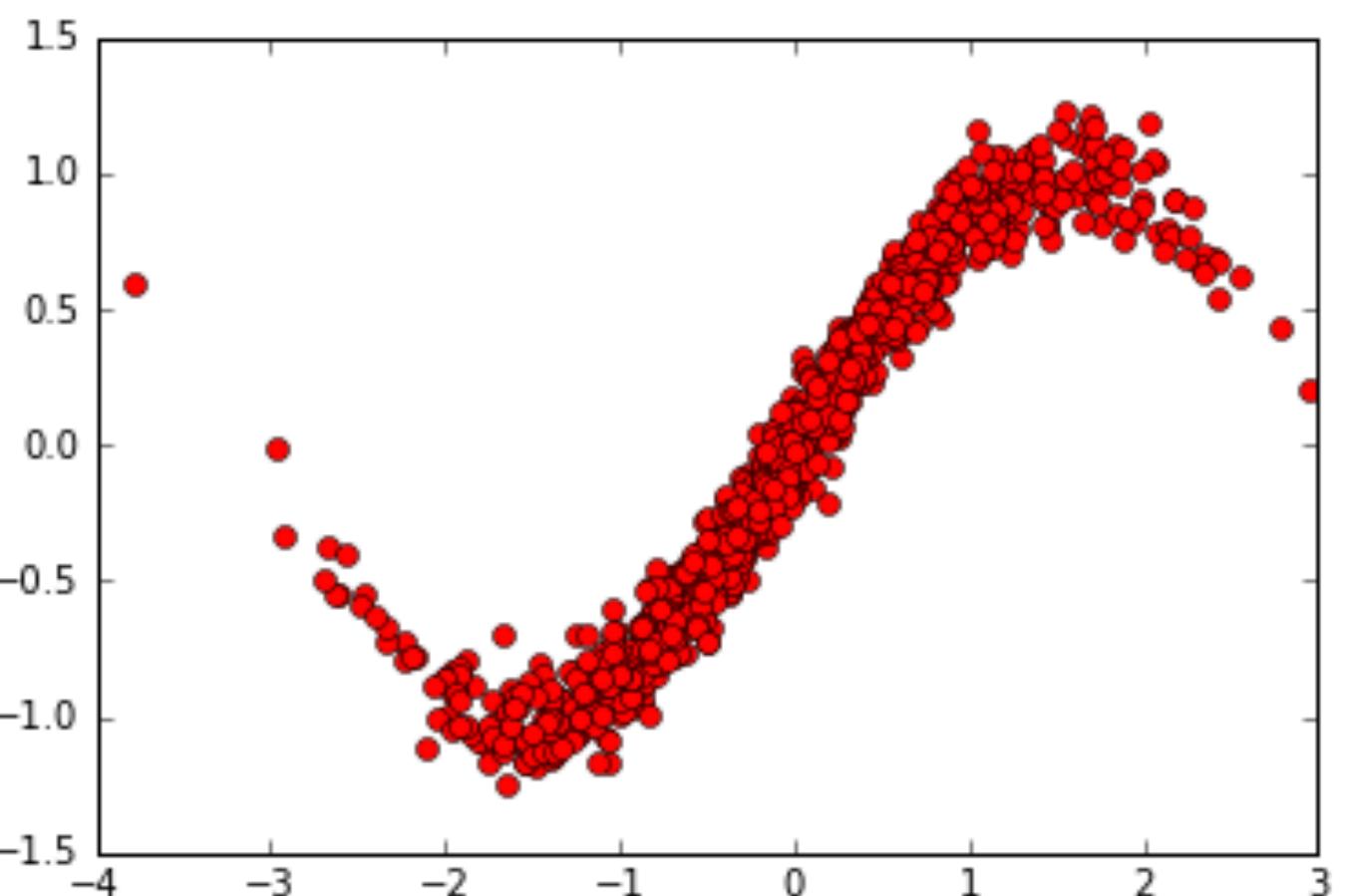
```
import numpy as np

num_points = 1000
vectors_set = []
for i in range(num_points):
    x1=np.random.normal(.0, 1.0)
    y1=np.sin(x1) + np.random.normal(0.,0.1)
    vectors_set.append([x1,y1])

x_data = [v[0] for v in vectors_set]
y_data = [v[1] for v in vectors_set]

import matplotlib.pyplot as plt

plt.plot(x_data, y_data, 'ro')
plt.legend()
plt.show()
```



모델 구축

```
import tensorflow as tf
_x_data = tf.expand_dims(x_data,1)

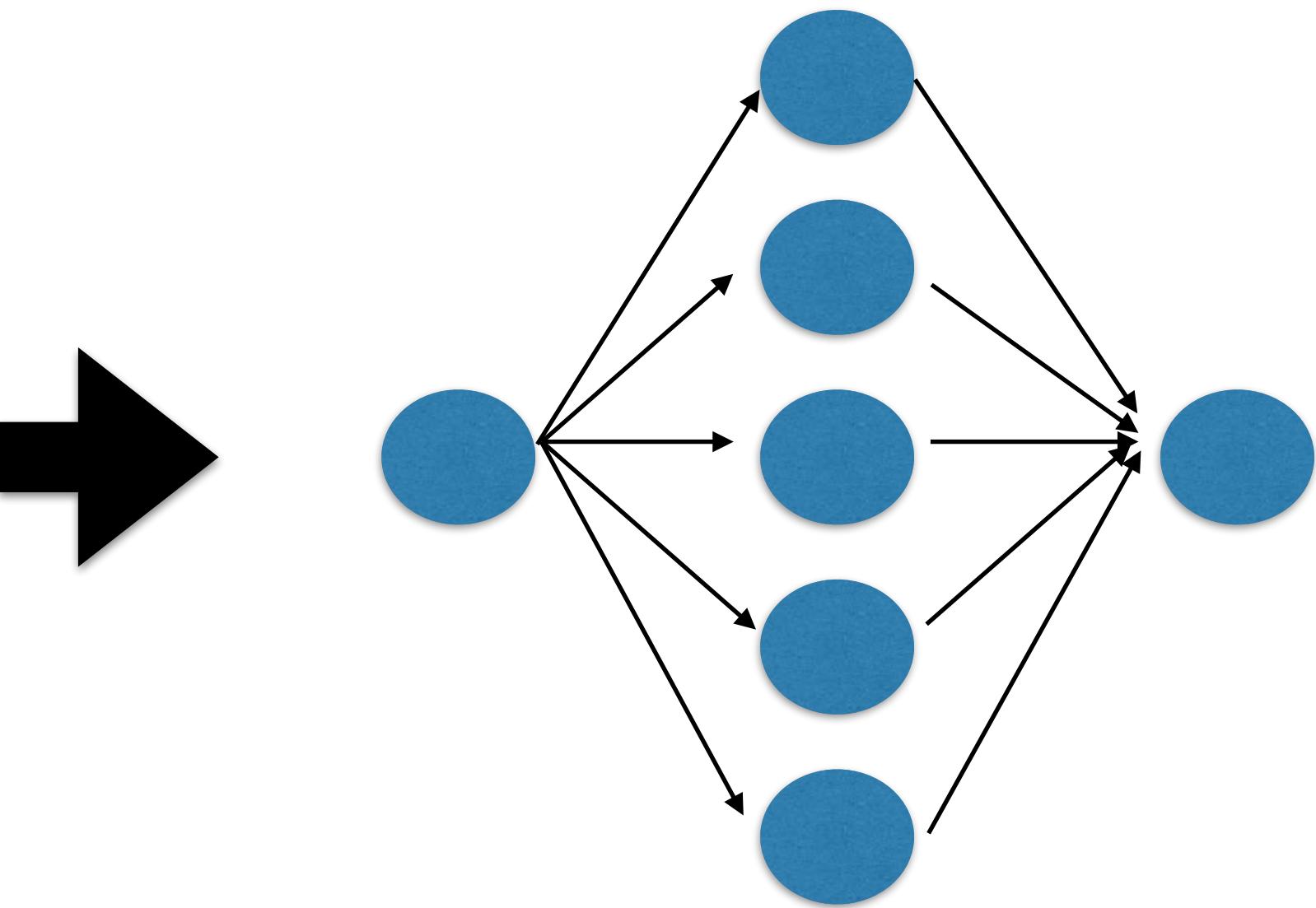
W = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))
W_out = tf.Variable(tf.random_uniform([5,1], -1.0, 1.0))

hidden = tf.nn.tanh(tf.matmul(_x_data,W))
output = tf.matmul(hidden, W_out)

loss = tf.reduce_mean(tf.square(output-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

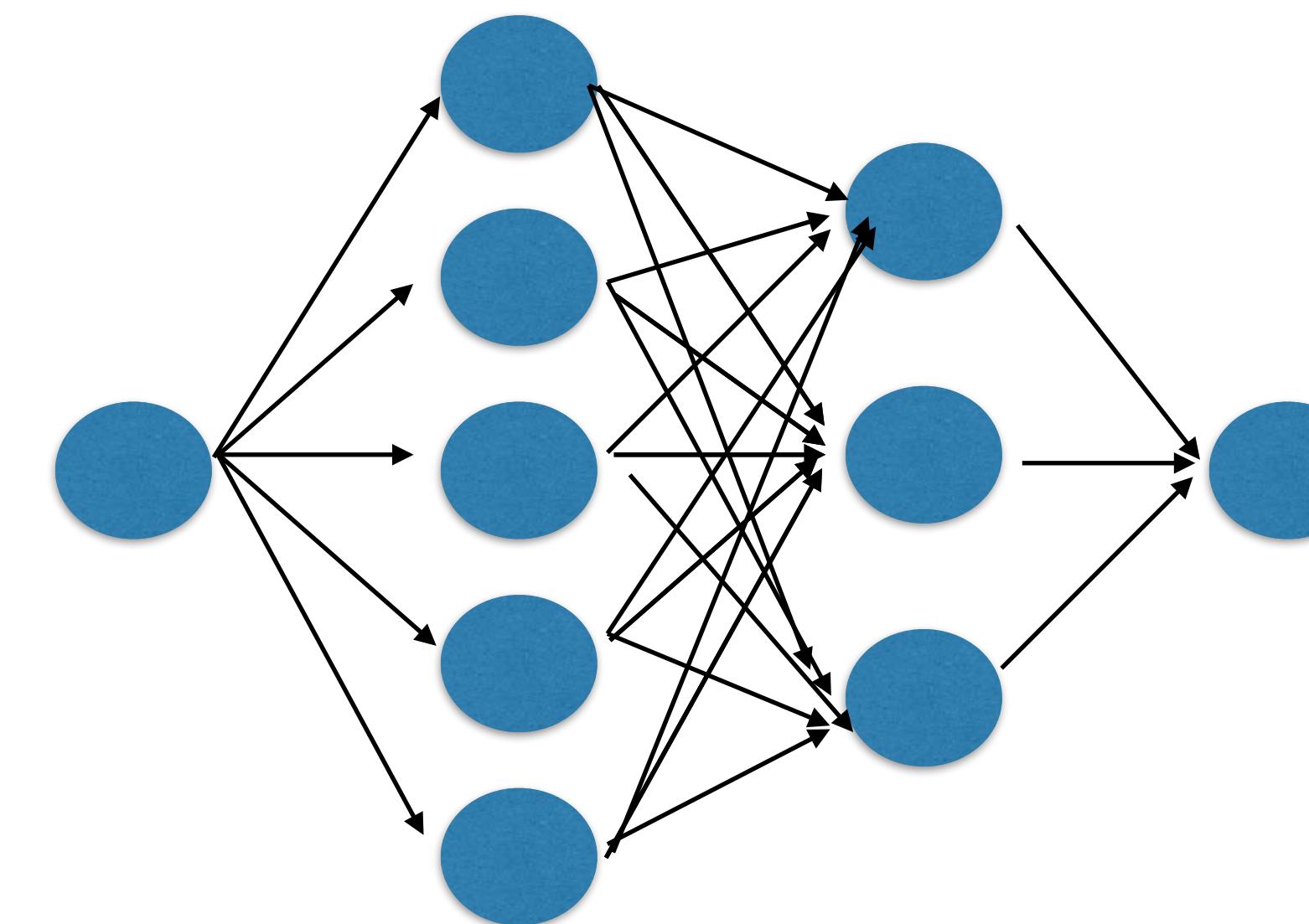
sess= tf.Session()
sess.run(init)
```



모델 구축 (2 Hidden Layer)

```
W1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0))
W_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0))

hidden1 = tf.nn.sigmoid(tf.matmul(_x_data,W1))
hidden2 = tf.nn.sigmoid(tf.matmul(hidden1,W2))
output = tf.matmul(hidden2, W_out)
```



## □ 모델 구축 (2 Hidden Layer)

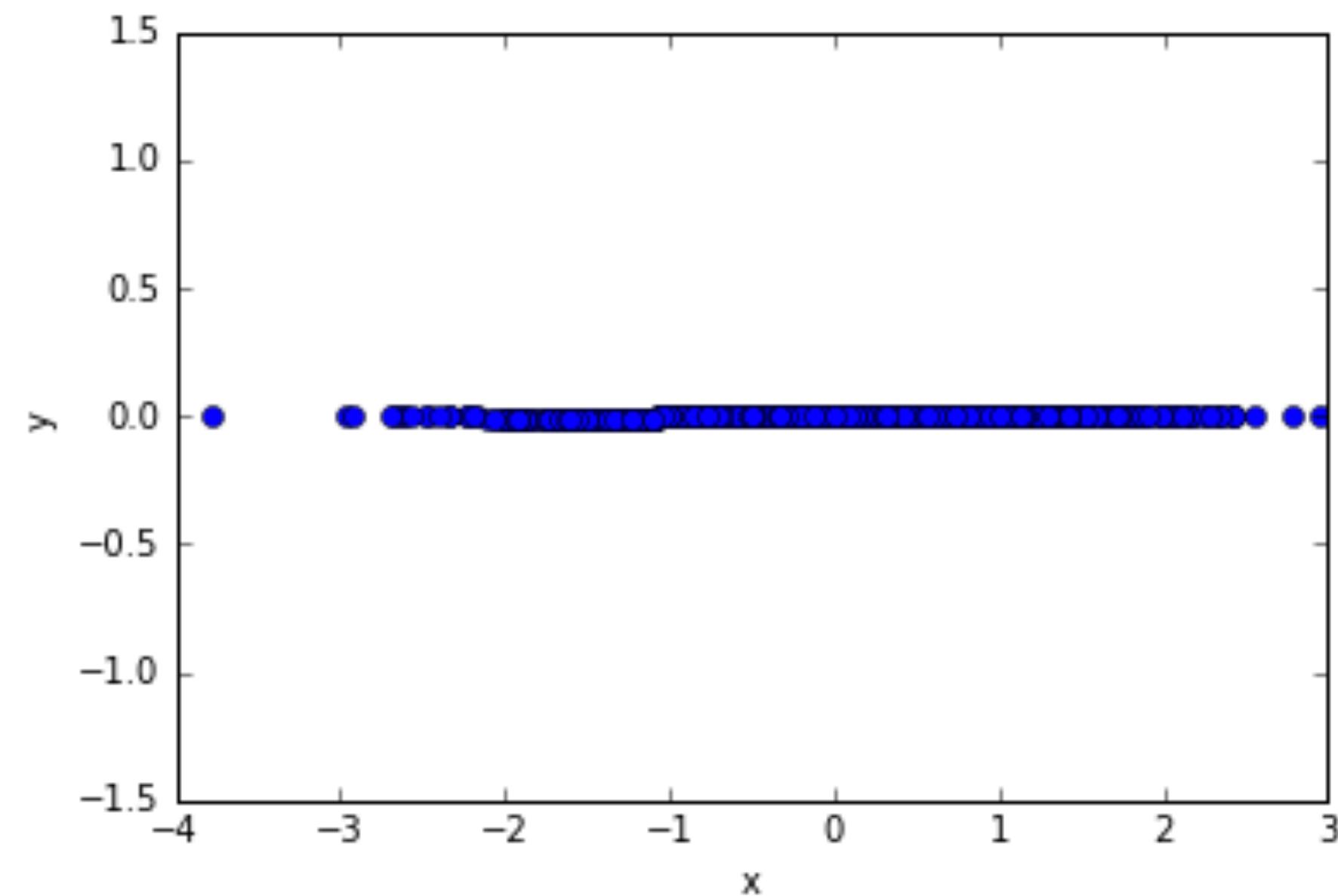
```
loss = tf.reduce_mean(tf.square(output-y_data))
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

init = tf.global_variables_initializer()

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

## □ 결과확인

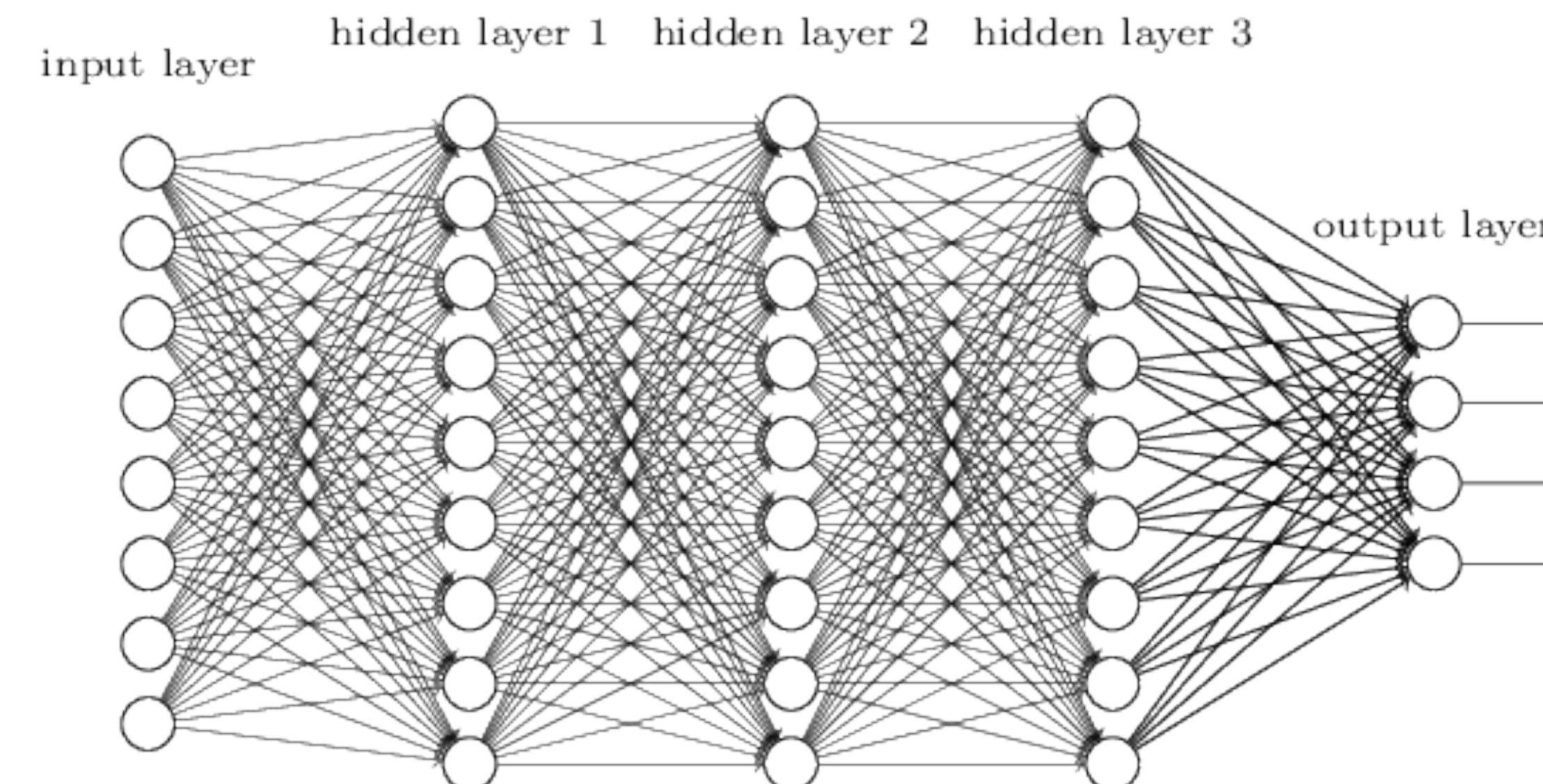
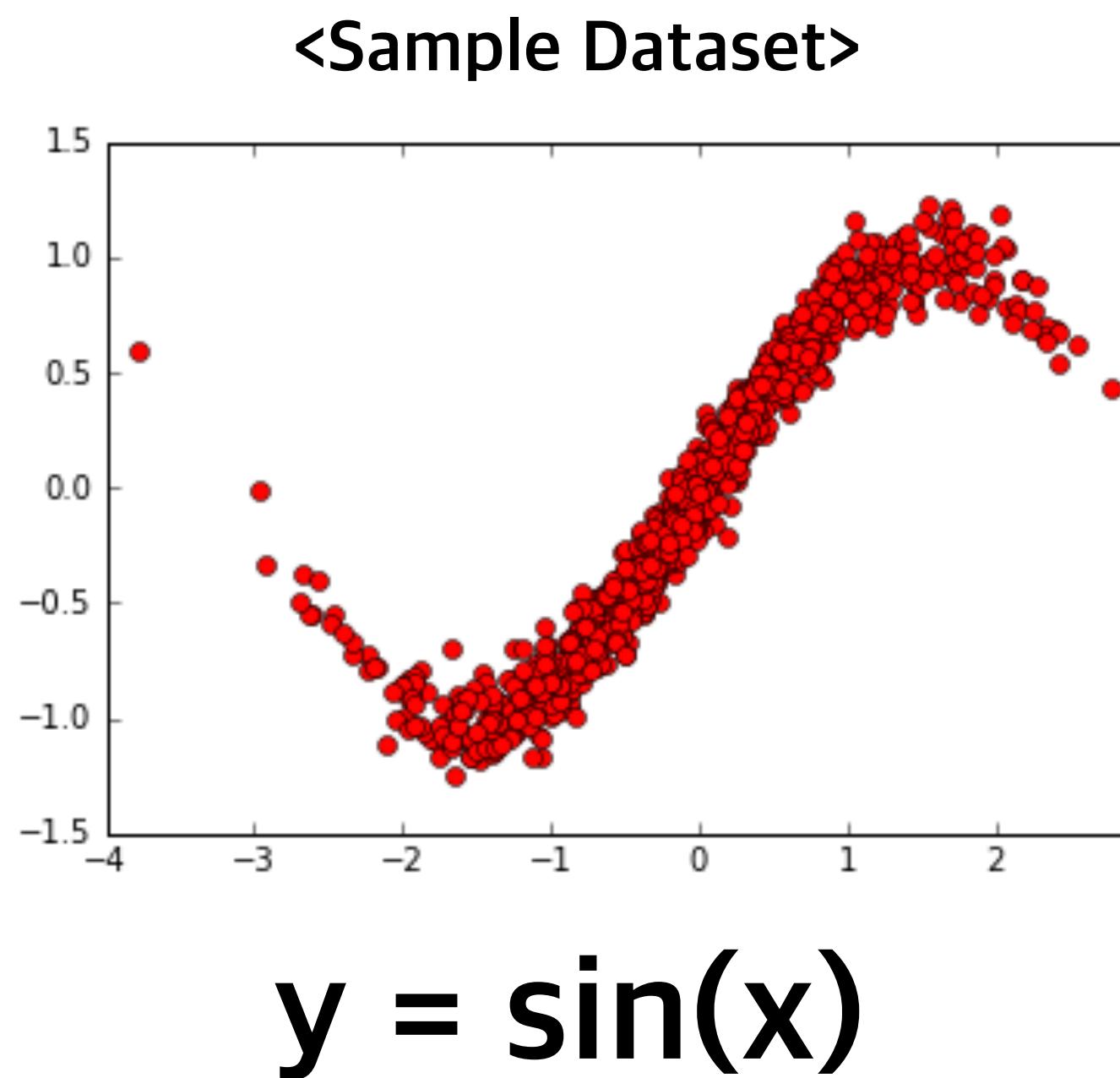
```
#plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, sess.run(output), 'bo')
plt.xlabel('x')
plt.xlim(-4,3)
plt.ylabel('y')
plt.ylim(-1.5,1.5)
plt.legend()
plt.show()
```



Why The Result is Garbage????

# Tensorflow Basics

- Dataset의 변수들 사이의 관계를 가장 잘 설명하고 예측하는 인공 신경망(Neural Network)을 찾아보자



1) Data 생성

2) 모델 구축

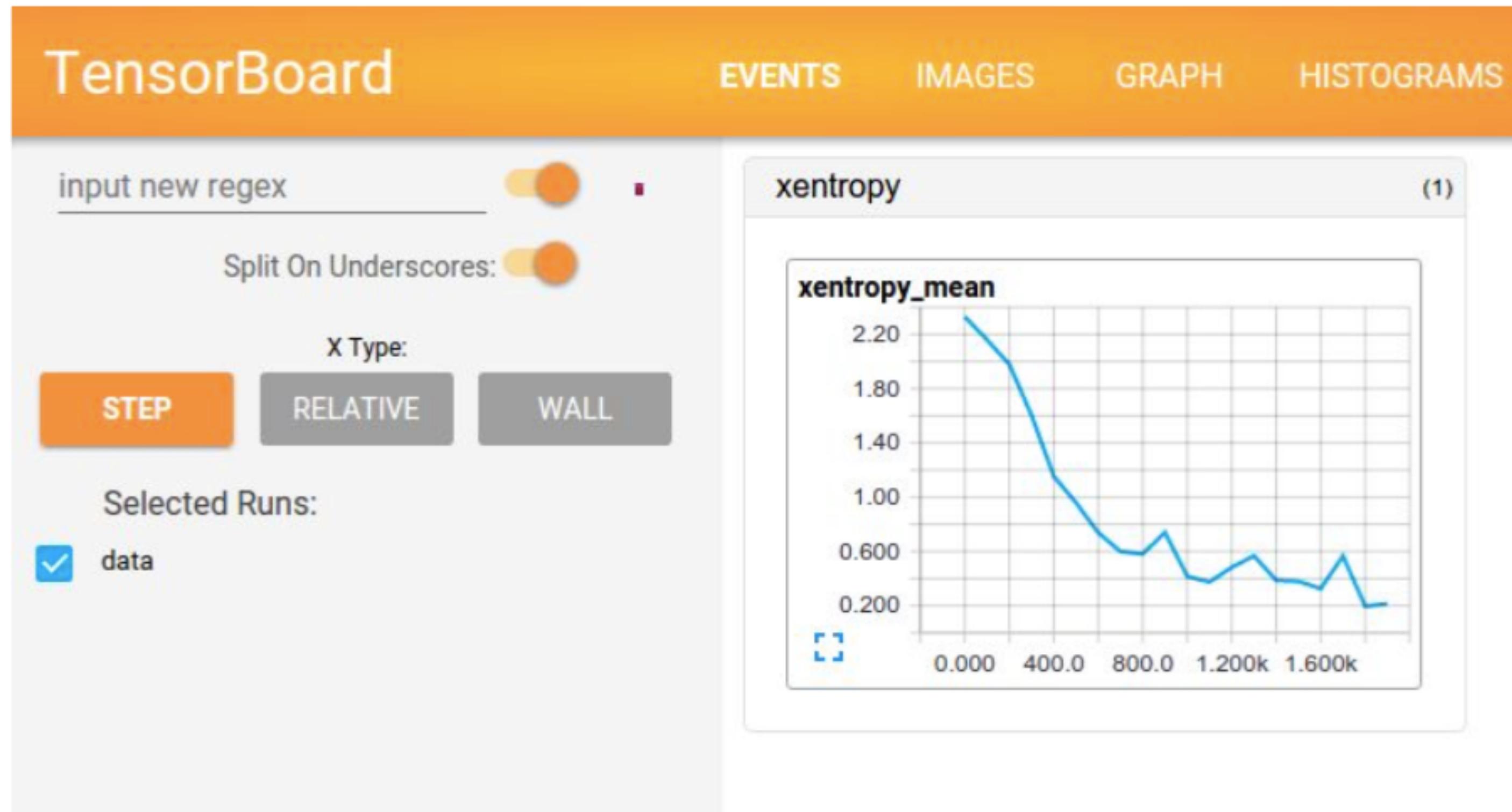
3) 학습

Tensorflow의 가장 큰 장점 중 하나는 **Visualization** 기능

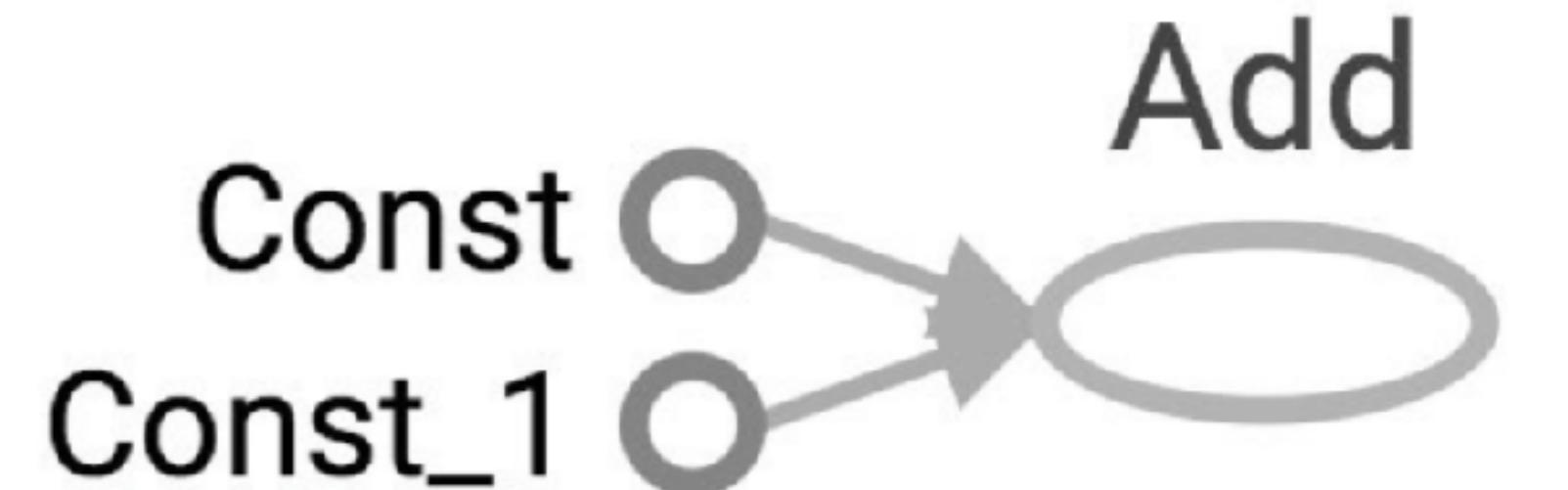
# What is tensorBoard?

- TensorBoard :Tensorflow가 포함하고 있는 Graph visualization 소프트웨어

< TensorBoard>



```
a = tf.constant(2)  
b = tf.constant(3)  
x = tf.add(a, b)
```



# Tensorboard

- `tf.summary` 내의 메소드(method)를 활용하여 그래프, 상수, 히스토그램을 텐서보드에 입력할 수 있음
- `tf.summary.FileWriter`를 통해 `tensorboard` 파일을 생성하고 내부의 메소드 주로 활용함
- 기타 `tf.summary.scalar`, `tf.summary.histogram` 등 추가 사용 (TF 홈페이지 참고)

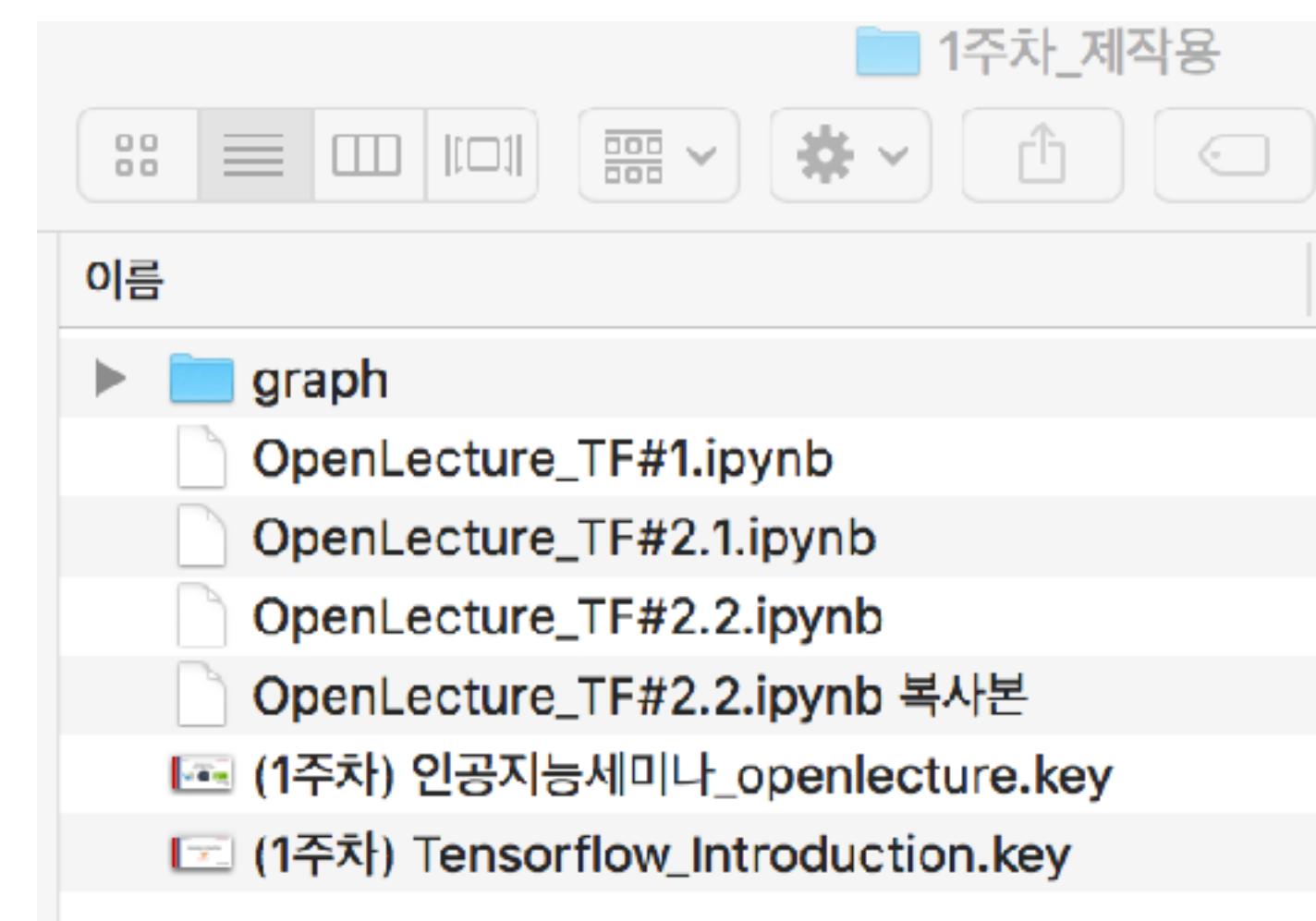
목차

```
class tf.summary.  
FileWriter  
  
Methods  
  
    __init__  
    add_event  
    add_graph  
    add_meta_graph  
    add_run_metadata  
    add_session_log  
    add_summary  
    close  
    flush  
    get_logdir  
    reopen
```

In [6]: `writer = tf.summary.FileWriter('./graph', sess.graph')`

2) 저장할 그래프

1) Tensorboard 파일  
저장 디렉토리



LDY\$ tensorboard --logdir="./graph"

tensorboard -- logdir="디렉토리 위치" (-- port 6006)



eXem

# Tensorboard Example

TensorBoard

SCALARS

IMAGES

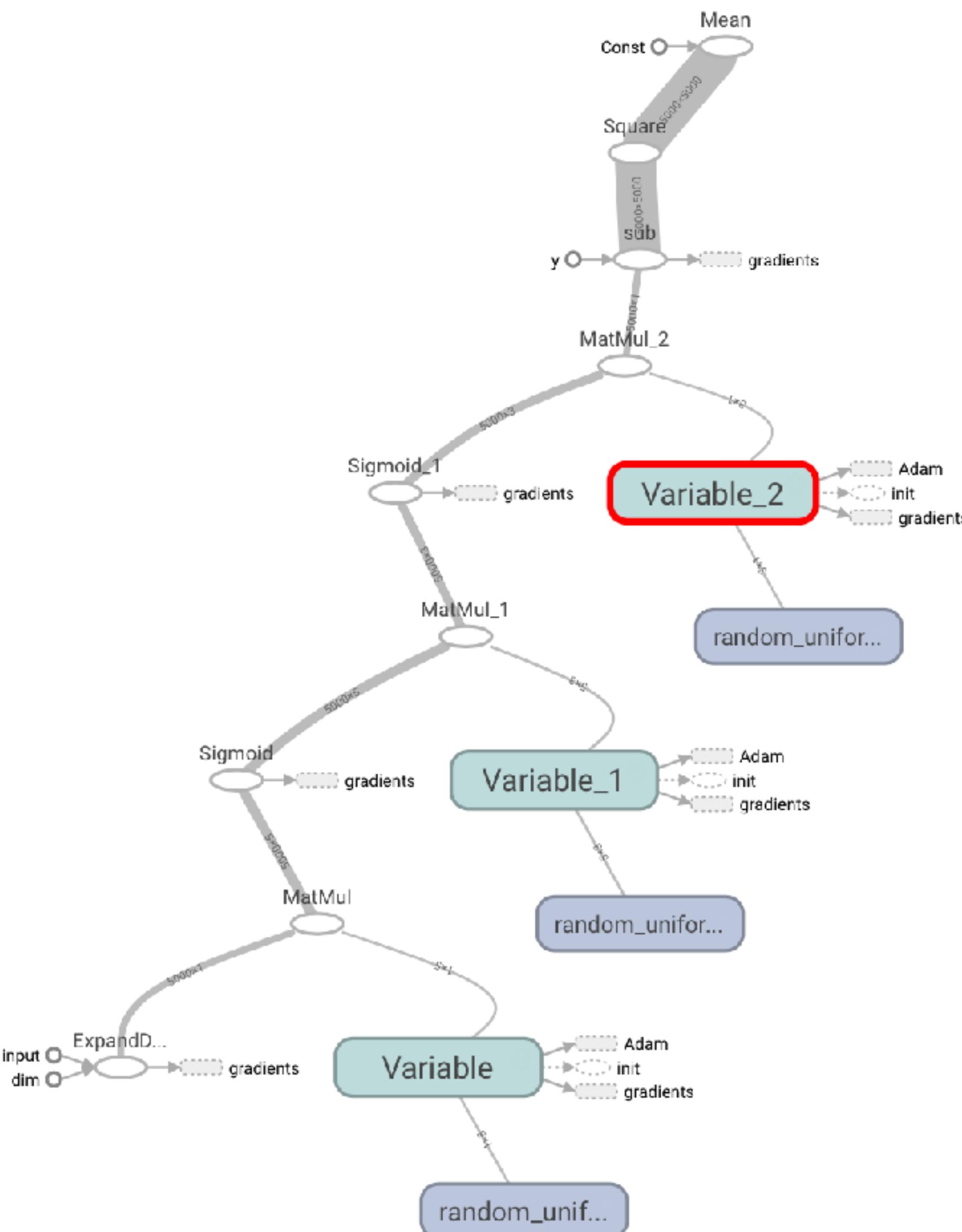
AUDIO

GRAPHS

DISTRIBUTIONS

HISTOGRAMS

EMBEDDINGS



아직 예쁘게 다듬어지지 않음!  
Namespace 필요

eXem

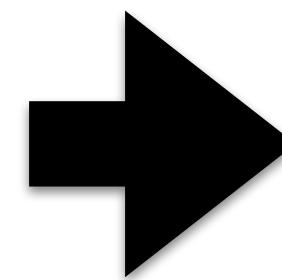
## tf.name\_scope("name")

- tf.name\_scope를 통해 이름을 가진 각 노드들의 범주를 지정해줄 수 있음

```
w1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))
w2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0))
w_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0))

hidden1 = tf.nn.sigmoid(tf.matmul(_x_data,w1))
hidden2 = tf.nn.sigmoid(tf.matmul(hidden1,w2))
output = tf.matmul(hidden2, w_out)

loss = tf.reduce_mean(tf.square(output-y_data))
optimizer = tf.train.AdamOptimizer(0.5)
train = optimizer.minimize(loss)
```



```
with tf.name_scope("Model"):
    w1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))
    w2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0))
    w_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0))

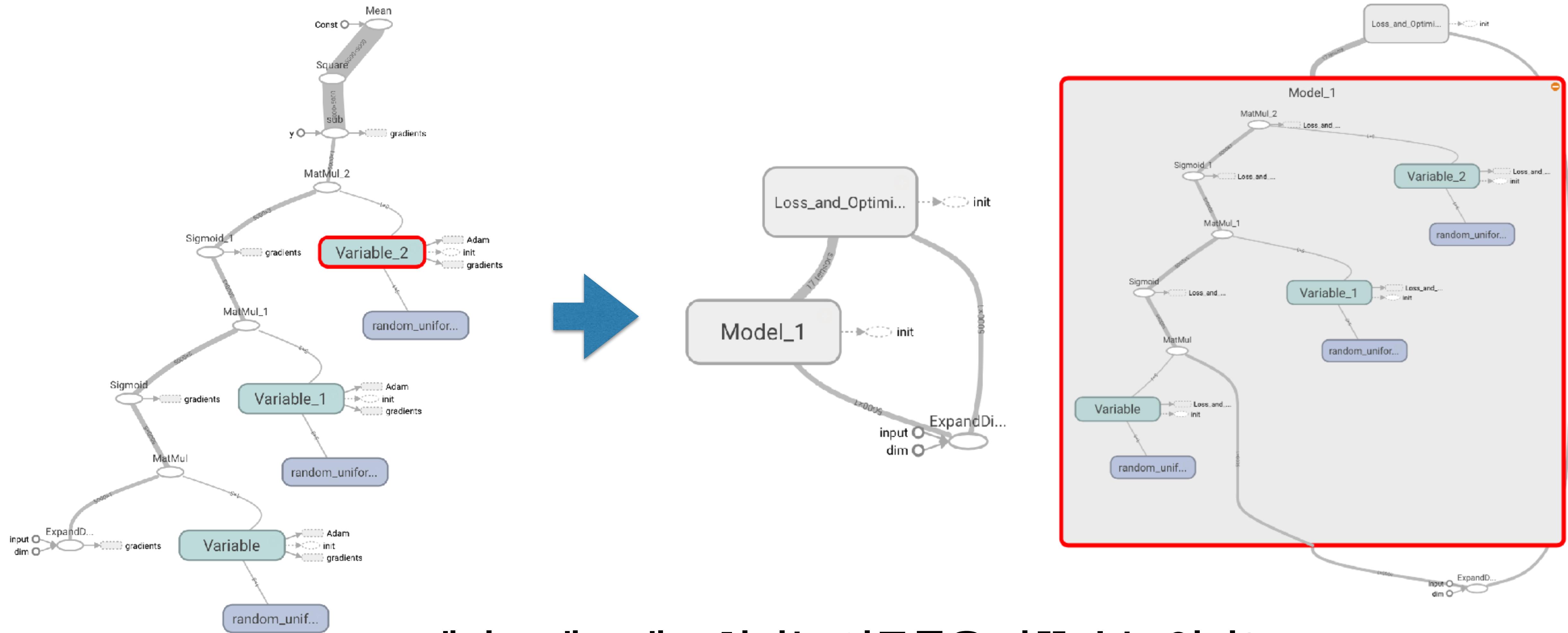
    hidden1 = tf.nn.sigmoid(tf.matmul(_x_data,w1))
    hidden2 = tf.nn.sigmoid(tf.matmul(hidden1,w2))
    output = tf.matmul(hidden2, w_out)

with tf.name_scope("Loss_and_Optimizer"):
    loss = tf.reduce_mean(tf.square(output-y_data))
    optimizer = tf.train.AdamOptimizer(0.5)
    train = optimizer.minimize(loss)
```

```
LDY$ tensorboard --logdir="./graph"
```

# tf.name\_scope("name")

- Tensorboard를 통해 확인해보면 각 노드들이 name\_scope에 따라서 큰 모듈로 묶여있는 것을 볼 수 있음



텐서 그래프 내 표현되는 이름들을 바꿀 수는 없나?



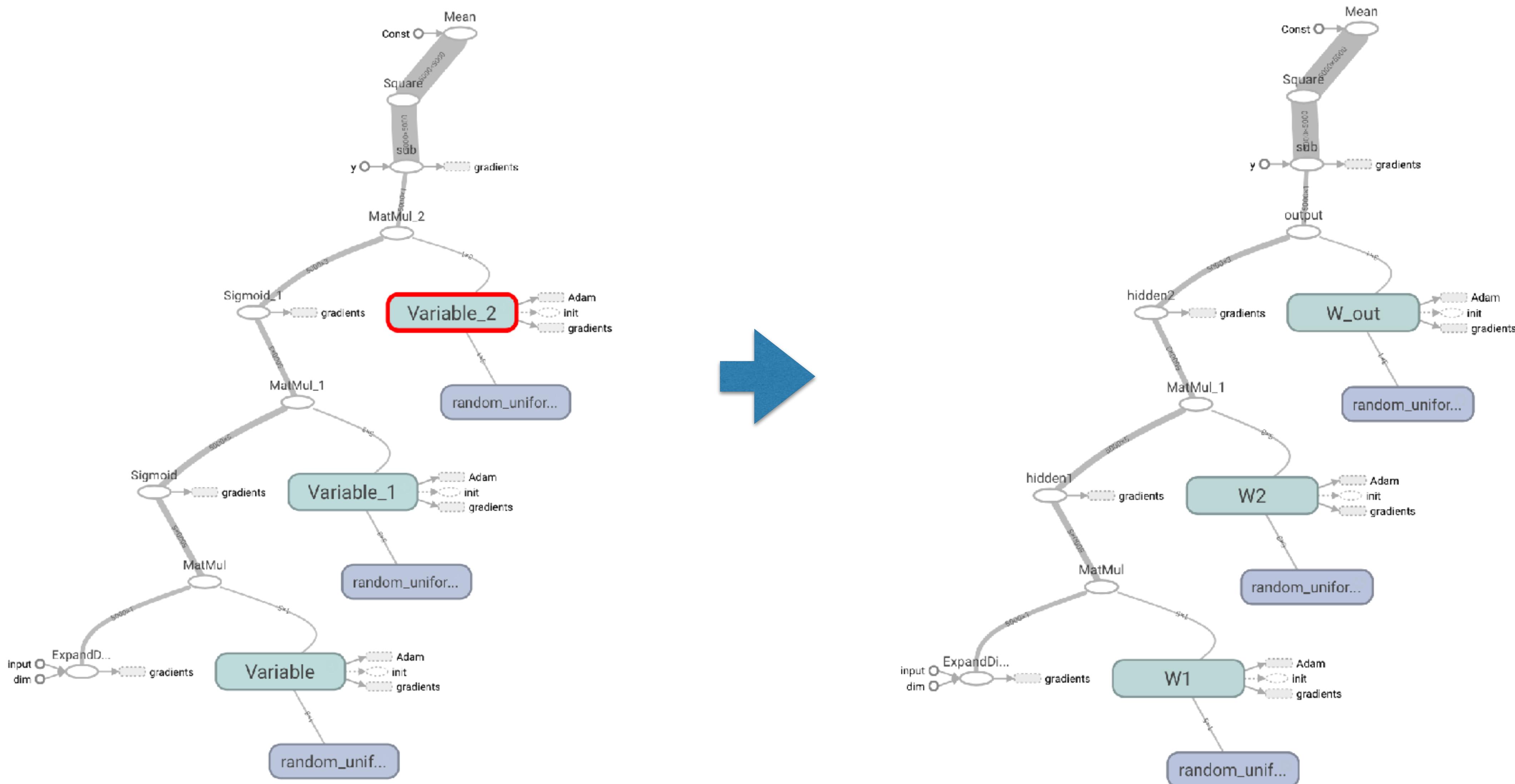
□ Tensorflow에서 사용하는 Tensor, Operation, Variable 등의 args에서 말하는 ‘name’은 모두 그래프 표현과 관련

```
W1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0), name='W1')
W2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0), name='W2')
W_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0), name='W_out')

hidden1 = tf.nn.sigmoid(tf.matmul(_x_data,W1), name='hidden1')
hidden2 = tf.nn.sigmoid(tf.matmul(hidden1,W2), name='hidden2')
output = tf.matmul(hidden2, W_out, name='output')
```

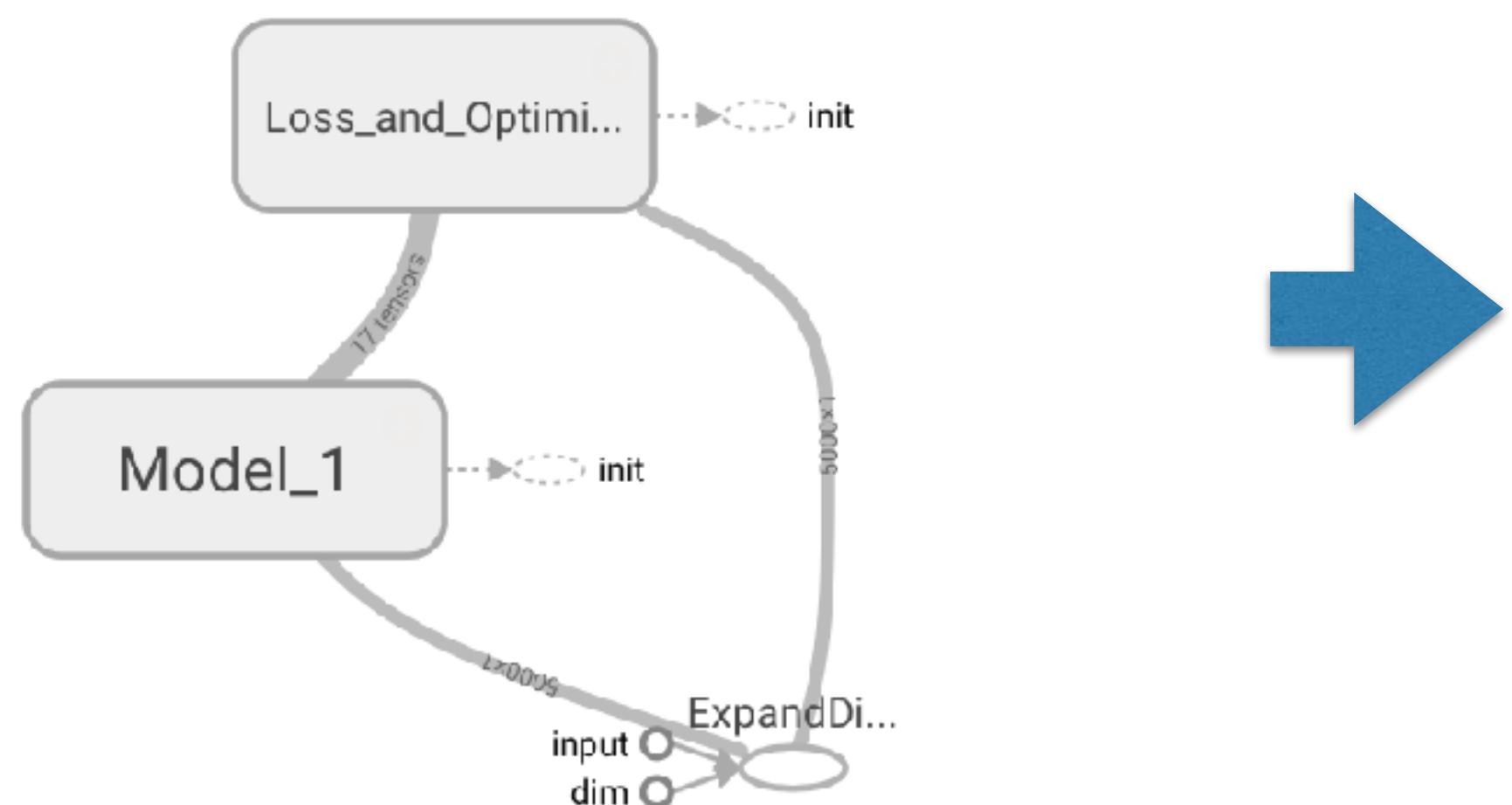
# name을 이용한 표현 바꾸기

□ Tensorflow에서 사용하는 Tensor, Operation, Variable 등의 args에서 말하는 ‘name’은 모두 그래프 표현과 관련



- **tf.Graph** 의 선언을 통해 그래프를 정의하여 사용할 수 있음
- 특별한 선언이 없을 경우, Tensorflow 내에서 기본적으로 포함하는 그래프를 디폴트 그래프(default graph)로 사용함

<그래프 선언 없이 생성>



<Tensorflow 내 작동 순서>

```
main_graph = tf.Graph()  
with main_graph.as_default():  
    ...
```

를 사용하여 구현한 것과 같음  
(실제로 main\_graph라고 존재하지 않음)

# Multiple Graph with tf.Graph()

- tf.Graph를 사용하면 여러 그래프를 구조화하여 사용할 수 있음
- 한 프로그램에서 여러 그래프를 선언하여 사용할 경우, **with tf.Graph.as\_default():** 문을 통해 그래프의 범위 지정 필수
- (Error!) 사용자가 직접 그래프를 선언하고, **as\_default()**를 사용하지 않을 경우 에러 발생

<Correct Case>

```
import tensorflow as tf  
  
g1 = tf.Graph()  
g2 = tf.Graph()  
  
with g1.as_default():  
    #Define graph model in g1  
    #...  
  
with g2.as_default():  
    #Define graph model in g2  
    #...
```

<Incorrect Case>

```
import tensorflow as tf  
  
g1 = tf.Graph()  
  
#Define graph model in g1  
#...  
  
with g1.as_default():  
    #Define graph model in g1  
    #...
```

tf.Graph()를 사용하면  
관련이 없는 여러 모델을 동시에 사용할 때 효과적으로 관리 가능

Tensorflow로 데이터를 표현하는 법?

- Constant
- Variable
- Placeholder

# tf. constant vs. Variable

## □ Tensorflow에서 데이터를 표현하는 방법은 크게 2가지가 있음

1) tf.constant: 상수를 생성 (변하지 않는 특정한 데이터를 의미)

2) tf.Variable: 변수를 생성 (상황에 따라 변하는, 업데이트되는 데이터를 의미)

### tf.constant

- 입력한 Value의 값과 모양(shape)을 갖는 Tensor를 반환
- 편리하게 constant를 선언하는 다양한 함수들 존재

### tf.constant

```
constant(  
    value,  
    dtype=None,  
    shape=None,  
    name='Const',  
    verify_shape=False  
)
```

```
import tensorflow as tf  
a = tf.constant(3)  
print a
```

Tensor("Const:0", shape=(), dtype=int32)

```
import tensorflow as tf  
a = tf.constant([5,10,15], shape=(3,1))  
print a
```

Tensor("Const\_2:0", shape=(3, 1), dtype=int32)

tf.zeros

tf.zeros\_like

tf.ones

tf.ones\_like

tf.fill

tf.constant

tf.random\_normal

tf.truncated\_normal

tf.random\_uniform

tf.random\_shuffle

tf.random\_crop

tf.multinomial

tf.random\_gamma

tf.set\_random\_seed

## tf.constant

```
tf.constant(value, dtype=None, shape=None, name='Const', verify_shape=False)
```

```
a=tf.constant(2, dtype="int32", name="a") → 2
```

```
a=tf.constant(2, dtype="float", name="a") → 2.0
```

```
a=tf.constant([2.0,3.0, 2.0,3.0], dtype="float", shape=[2,2], name="a") → [[ 2.  3.]  
[ 2.  3.]]
```

```
a=tf.constant([2.0,3.0], dtype="float", shape=[2,2], name="a") → [[ 2.  3.]  
[ 3.  3.]]
```

```
a=tf.constant(1, shape=[2], dtype="float", name="a", verify_shape=False) → [ 1.  1.]
```

```
a=tf.constant(1, shape=[2], dtype="float", name="a", verify_shape=True) → Error!!
```

# tf. constant vs. Variable

## □ Tensorflow에서 데이터를 표현하는 방법은 크게 2가지가 있음

- 1) tf.constant: 상수를 생성 (변하지 않는 특정한 데이터를 의미)
- 2) tf.Variable: 변수를 생성 (상황에 따라 변하는, 업데이트되는 데이터를 의미)

### tf.constant

- 입력한 Value의 값과 모양(shape)을 갖는Tensor를 반환
- 편리하게 constant를 선언하는 다양한 함수들 존재

```
tf.zeros  
tf.zeros_like  
tf.ones  
tf.ones_like  
tf.fill  
tf.constant  
tf.random_normal  
tf.truncated_normal  
tf.random_uniform  
tf.random_shuffle  
tf.random_crop  
tf.multinomial  
tf.random_gamma  
tf.set_random_seed
```

### Random Number 생성 역시 Constant의 일종

```
In [2]: const = tf.random_uniform([1], -1.0, 1.0)  
sess = tf.Session()  
  
In [3]: const  
  
Out[3]: <tf.Tensor 'random_uniform:0' shape=(1,) dtype=float32>  
  
In [4]: sess.run(const)  
  
Out[4]: array([ 0.17192531], dtype=float32)
```

# Generating random constants from certain distribution

- 특정 분포를 이용해서 random하게 수들을 생성할 수 있음

- Normal Distribution

```
tf.random_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None, name=None)  
tf.truncated_normal(shape, mean=0.0, stddev=1.0, dtype=tf.float32, seed=None,  
name=None)
```

- Uniform Distribution

```
tf.random_uniform(shape, minval=0, maxval=None, dtype=tf.float32, seed=None,  
name=None)
```

- Shuffle

```
tf.random_shuffle(value, seed=None, name=None)
```

b=tf.random\_shuffle(value=[2,3,4,5]) → [3 5 4 2]

- Multinomial Distribution

```
tf.multinomial(logits, num_samples, seed=None, name=None)
```

- Gamma Distribution

```
tf.random_gamma(shape, alpha, beta=None, dtype=tf.float32, seed=None, name=None)
```

## Constant with Specific Values

```
tf.zeros(shape, dtype=tf.float32, name=None)
```

a=tf.zeros(shape=[2,3], dtype="int32",name="a") → [[0 0 0]  
[0 0 0]]

a=tf.zeros(shape=[2], dtype='float32',name="a") → [ 0. 0.]

```
tf.ones(shape, dtype=tf.float32, name=None)
```

a=tf.ones([2,3], dtype="int32",name="a") → [[1 1 1]  
[1 1 1]]

a=tf.ones([1,2], dtype="float32",name="a") → [[ 1. 1.]]

```
tf.fill(dims, value, name=None)
```

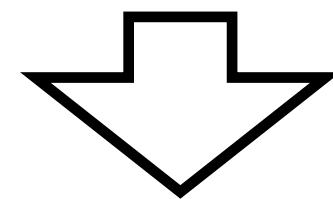
a=tf.fill(dims=[2,3], value=1,name="a") → [[1 1 1]  
[1 1 1]]

# Sequence Generation with Constant

- 숫자들 사이에 간격이 일정한 수열을 반환하도록 Constant를 생성할 수 있음

```
tf.linspace(start, stop, num, name=None)
```

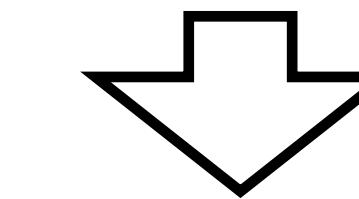
- start: 시작 값, stop: 종료 값, num: 생성 개수
- 종료 값을 포함하도록 생성
- Type of start & stop: **float\***
- Type of num: integer



```
tf.linspace(10.0, 13.0, 4, name="linspace")  
==> [10.0 11.0 12.0 13.0]
```

```
tf.range(start, limit=None, delta=1, dtype=None, name='range')
```

- start: 시작 값, limit: 종료 값, delta = 숫자간 간격
- 끝 값(stop)을 포함하지 않고 수열을 생성함.



```
a=tf.range(start=3,limit=17,delta=3, name="a")  
a=tf.range(start=3,limit=18,delta=3, name="a")  
==> [3, 6, 9, 12, 15]
```

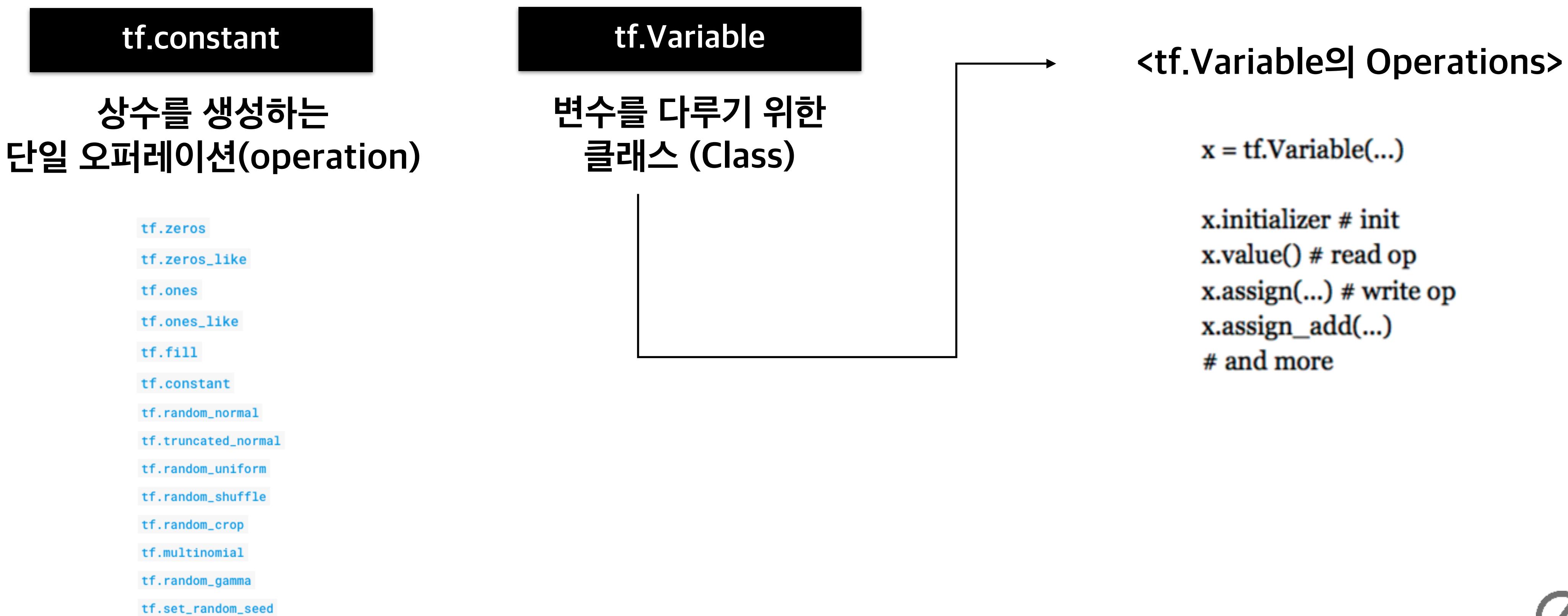
(주의) tf.linspace, tf.range 등으로 생성된 Sequence는 반복문에서 사용할 수 없음

# Contant vs. Variable: 구조적 관점의 차이

- tf.constant와 tf.Variable은 TF 내 구조적으로 차이를 가지고 있음

1) tf.constant: 단일 오퍼레이션 (Operation) 형태로 구현되어 있음

2) tf.Variable: 클래스 (Class) 형태로 구현되어 있음 —> 생성자, 멤버변수, 멤버함수 등 포함하고 있음!



# tf. constant vs. Variable

## □ Tensorflow에서 데이터를 표현하는 방법은 크게 2가지가 있음

- 1) tf.constant: 상수를 생성 (변하지 않는 특정한 데이터를 의미)
- 2) tf.Variable: 변수를 생성 (상황에 따라 변하는, 업데이트되는 데이터를 의미)

### tf.Variable

- Tensorflow 내 Class로 정의되어 있음
- 일반적으로 학습에 따라 변경되는 파라미터를 tf.Variable()로 선언

#### `__init__`

```
__init__(  
    initial_value=None,  
    trainable=True,  
    collections=None,  
    validate_shape=True,  
    caching_device=None,  
    name=None,  
    variable_def=None,  
    dtype=None,  
    expected_shape=None,  
    import_scope=None  
)
```

### Variable 선언

### Initialization 실행

- 초기값으로 설정할 constant를 지정하며 선언

```
w = tf.Variable(tf.random_uniform([1], -1.0, 1.0))  
b = tf.Variable(tf.zeros([1]))
```

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

(변수를 하나하나 초기화할 경우에는 tf.variables\_initializer 사용)

# tf. constant vs. Variable

## □ Tensorflow에서 데이터를 표현하는 방법은 크게 2가지가 있음

- 1) tf.constant: 상수를 생성 (변하지 않는 특정한 데이터를 의미)
- 2) tf.Variable: 변수를 생성 (상황에 따라 변하는, 업데이트되는 데이터를 의미)

### tf.Variable

- Tensorflow 내 Class로 정의되어 있음
- 일반적으로 학습에 따라 변경되는 파라미터를 tf.Variable()로 선언

### \_\_init\_\_

```
__init__(  
    initial_value=None,  
    trainable=True,  
    collections=None,  
    validate_shape=True,  
    caching_device=None,  
    name=None,  
    variable_def=None,  
    dtype=None,  
    expected_shape=None,  
    import_scope=None  
)
```

→ 기본적으로 Optimizer를 훈련(Train)시키는 함수를 실행시키면,  
관련된 Variable은 모두 업데이트됨

## tf.assign()

- tf.Variable의 값이 항상 Optimizer에 의해 변경되는 것은 아님
- tf.assign()을 통해 변수에 직접 값을 지정할 수 있음

```
# create a variable whose original value is 2
a = tf.Variable(2, name="scalar")

# assign a * 2 to a and call that op a_times_two
a_times_two = a.assign(a * 2)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    sess.run(init)
    # have to initialize a, because a_times_two op depends on the value of a
    sess.run(a_times_two) # >> 4
    sess.run(a_times_two) # >> 8
    sess.run(a_times_two) # >> 16
```

지정한 값을 해당 변수에  
1) 지정함(assign)과  
2) 해당 변수를 리턴함 (주의)

## Variable.assign\_add(sub)

- 단순히 variable에 더하고 빼는 과정을 할 때, Variable.assign\_add(sub) 사용

```
W = tf.Variable(10)

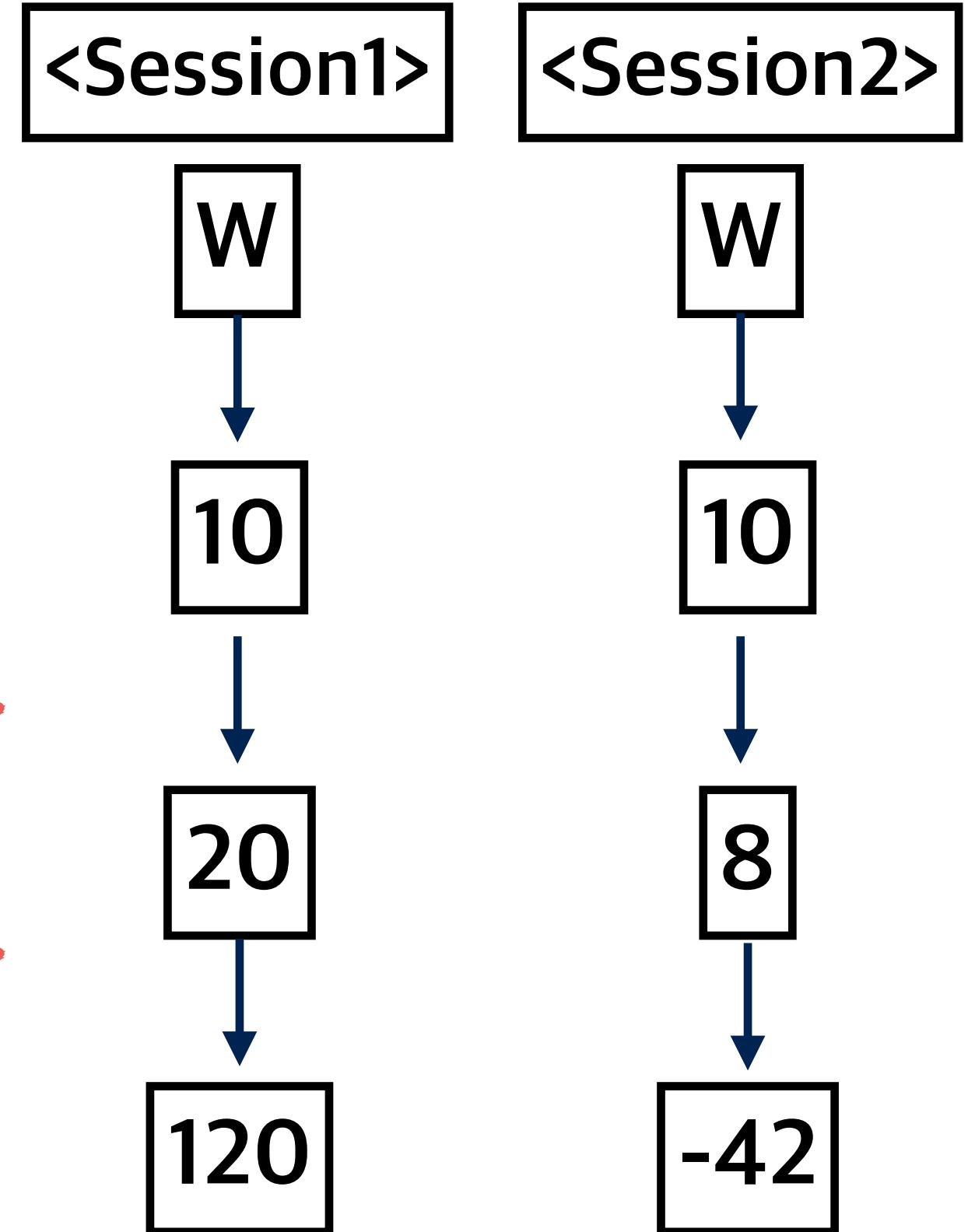
sess1 = tf.Session()
sess2 = tf.Session()

sess1.run(W.initializer)
sess2.run(W.initializer)

print sess1.run(W.assign_add(10)) # >> 20
print sess2.run(W.assign_sub(2)) # >> 8

print sess1.run(W.assign_add(100)) # >> 120
print sess2.run(W.assign_sub(50)) # >> -42

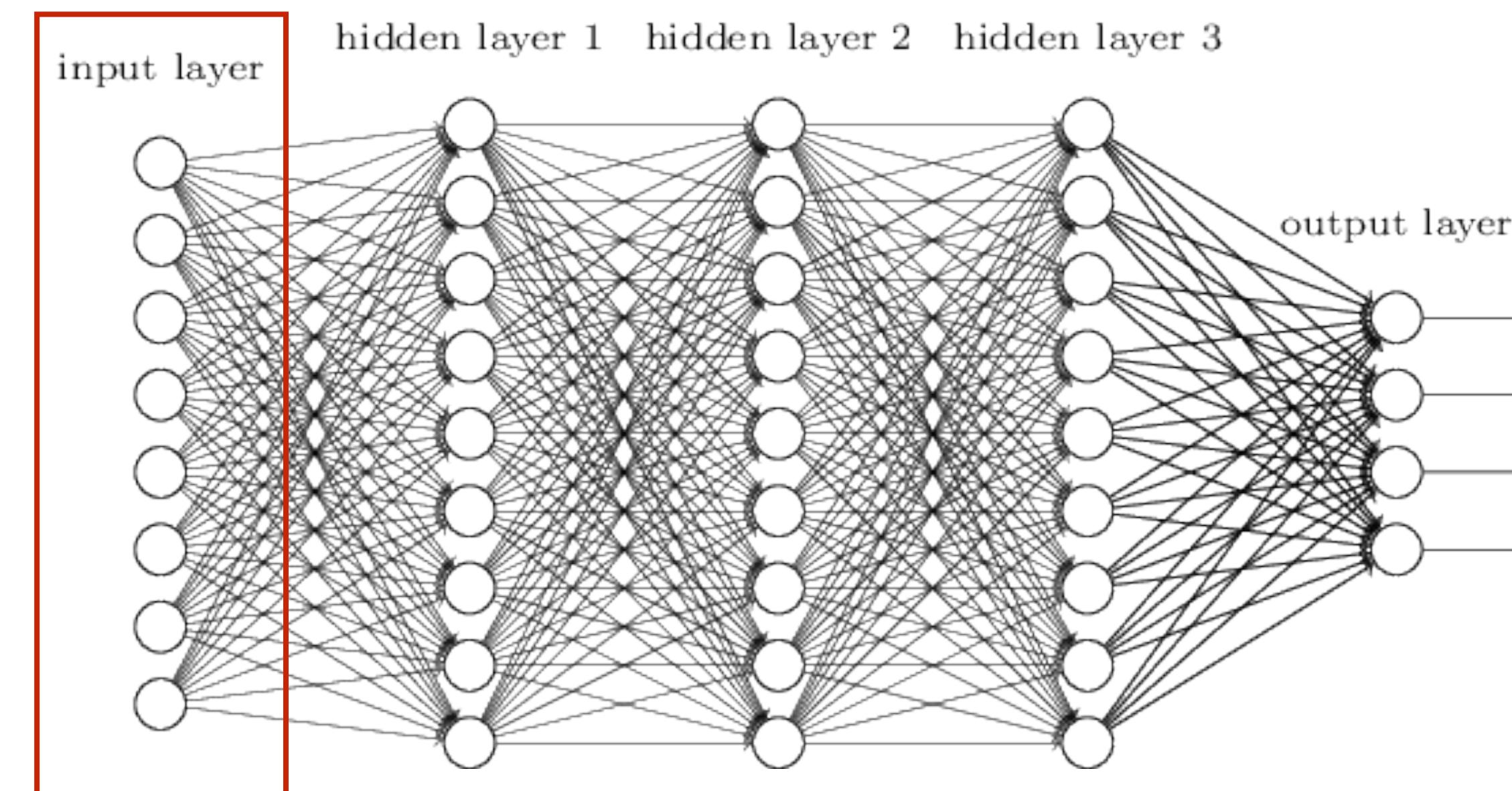
sess1.close()
sess2.close()
```



\*(주의) Session은 프로그램 한 개와 같은 의미  
>> Session이 다르면 다른 프로그램에서 독립적으로 실행되는 것

# Data representation with Placeholder

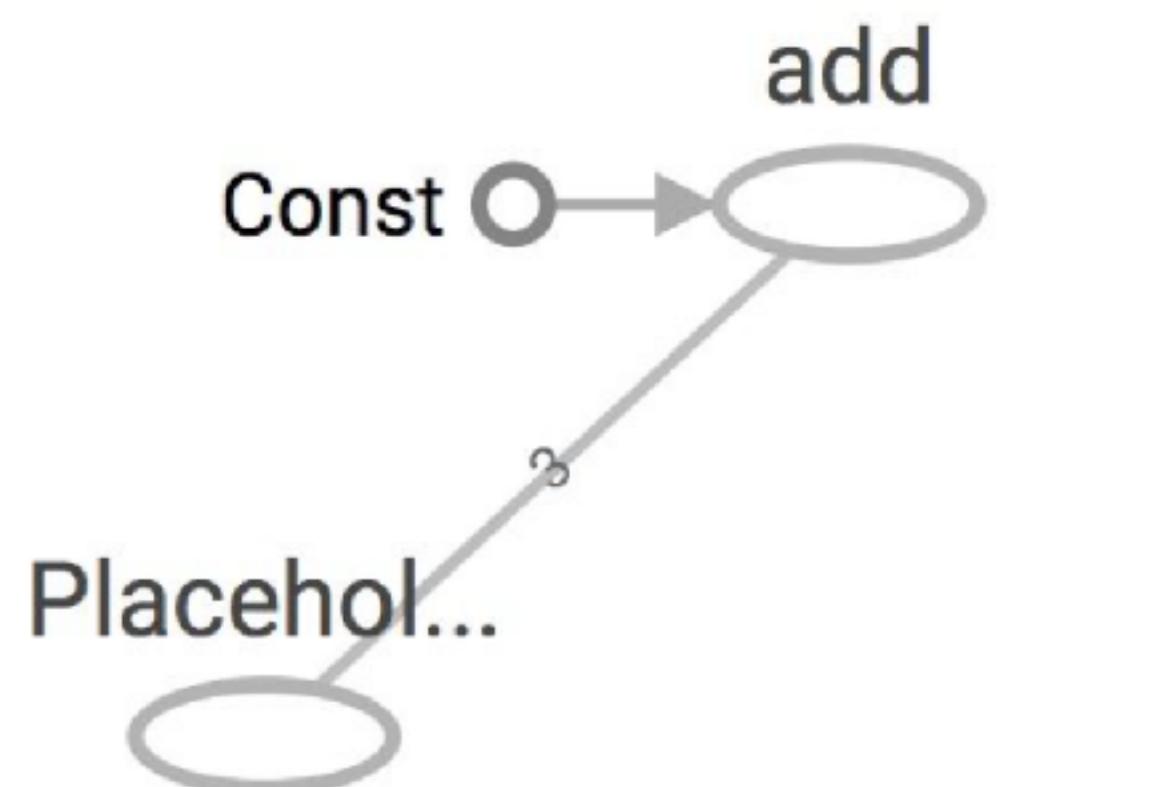
- ❑ 우리가 만드는 머신러닝/딥러닝 모델은 다양한 데이터를 이용하는 것이 일반적
- ❑ Input에 해당하는 데이터가 특정하게 정해지는 경우보다는 정해지지 않는 것이 일반적
- ❑ 학습을 위해 입력값으로 사용하는 데이터는 반복에 따라 달라질 수 있음



We need “Placeholder”

# Data representation with Placeholder

- ❑ Placeholder는 특정한 데이터 텐서를 표현하는 것이 아닌 데이터가 포함되는 공간 (Place(공간) + Holder(공간, 용기))
- ❑ **dtype, shape, name**을 전달 인자로 가지고 있음
- ❑ Placeholder의 shape를 결정할 때 **None**을 사용할 수 있음 (주의사항 존재)



## tf.placeholder

```
placeholder(  
    dtype,  
    shape=None,  
    name=None  
)
```

- 데이터 타입

```
place_holder1 = tf.placeholder(tf.int32)  
place_holder2 = tf.placeholder(tf.int64)  
place_holder3 = tf.placeholder(tf.float32)
```

- 데이터 타입, 모양

```
place_holder1 = tf.placeholder(tf.int32, shape=None)  
place_holder2 = tf.placeholder(tf.int64, shape=(2,2))  
place_holder3 = tf.placeholder(tf.float32, shape=(None,2))
```

## <Placeholder 선언 예시>

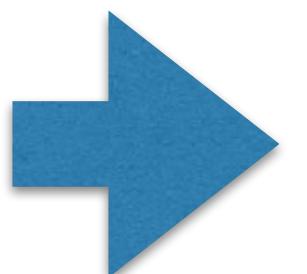
# Data representation with Placeholder

- ❑ Placeholder는 특정한 데이터 텐서를 표현하는 것이 아닌 데이터가 포함되는 공간 (Place(공간) + Holder(공간, 용기))
- ❑ dtype, shape, name을 전달 인자로 가지고 있음
- ❑ Placeholder의 shape를 결정할 때 **None**을 사용할 수 있음 (주의사항 존재)

## <Placeholder 선언 예시>

- 데이터 타입, 모양

```
place_holder1 = tf.placeholder(tf.int32, shape=None)
place_holder2 = tf.placeholder(tf.int64, shape=(2,2))
place_holder3 = tf.placeholder(tf.float32, shape=(None,2))
```



- shape = (2,2) : (2,2) 고정된 모양의 공간
- shape = None : 모양이 정해져 있지 않음
- shape = (None, 2) : (?,2) 모양의 고정되지 않은 공간
  - 일반적으로 Input의 feature 개수는 고정하지만 데이터 개수는 고정하지 않고 유동적으로 조절

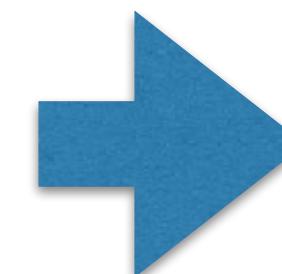
# Data representation with Placeholder

- ❑ Placeholder는 특정한 데이터 텐서를 표현하는 것이 아닌 데이터가 포함되는 공간 (Place(공간) + Holder(공간, 용기))
- ❑ dtype, shape, name을 전달 인자로 가지고 있음
- ❑ Placeholder의 shape를 결정할 때 **None**을 사용할 수 있음 (주의사항 존재)

## <Placeholder 선언 예시>

- 데이터 타입, 모양

```
place_holder1 = tf.placeholder(tf.int32, shape=None)
place_holder2 = tf.placeholder(tf.int64, shape=(2,2))
place_holder3 = tf.placeholder(tf.float32, shape=(None,2))
```



- shape = (2,2) : (2,2) 고정된 모양의 공간
- shape = None : 모양이 정해져 있지 않음
- shape = (None, 2) : (?,2) 모양의 고정되지 않은 공간
  - 일반적으로 Input의 feature 개수는 고정하지만 데이터 개수는 고정하지 않고 유동적으로 조절

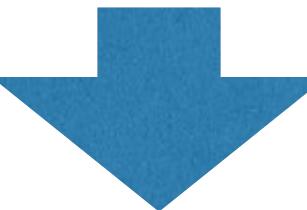
## Placeholder 공간에 어떻게 텐서를 넣나?

# Feed Dictionary

- ❑ Placeholder에 텐서를 입력하기 위해 구성하는 데이터 Dictionary
- ❑ sess.run 을 실행하기 전에 feed\_dict에 넣을 dictionary를 구성하고, 실행할 때 입력값으로 사용

```
input_data = tf.placeholder(tf.float32, shape=[2])
data_yes = np.array([4,3])

feed_dict = {input_data: data_yes}
print sess.run(input_data, feed_dict=feed_dict)
```

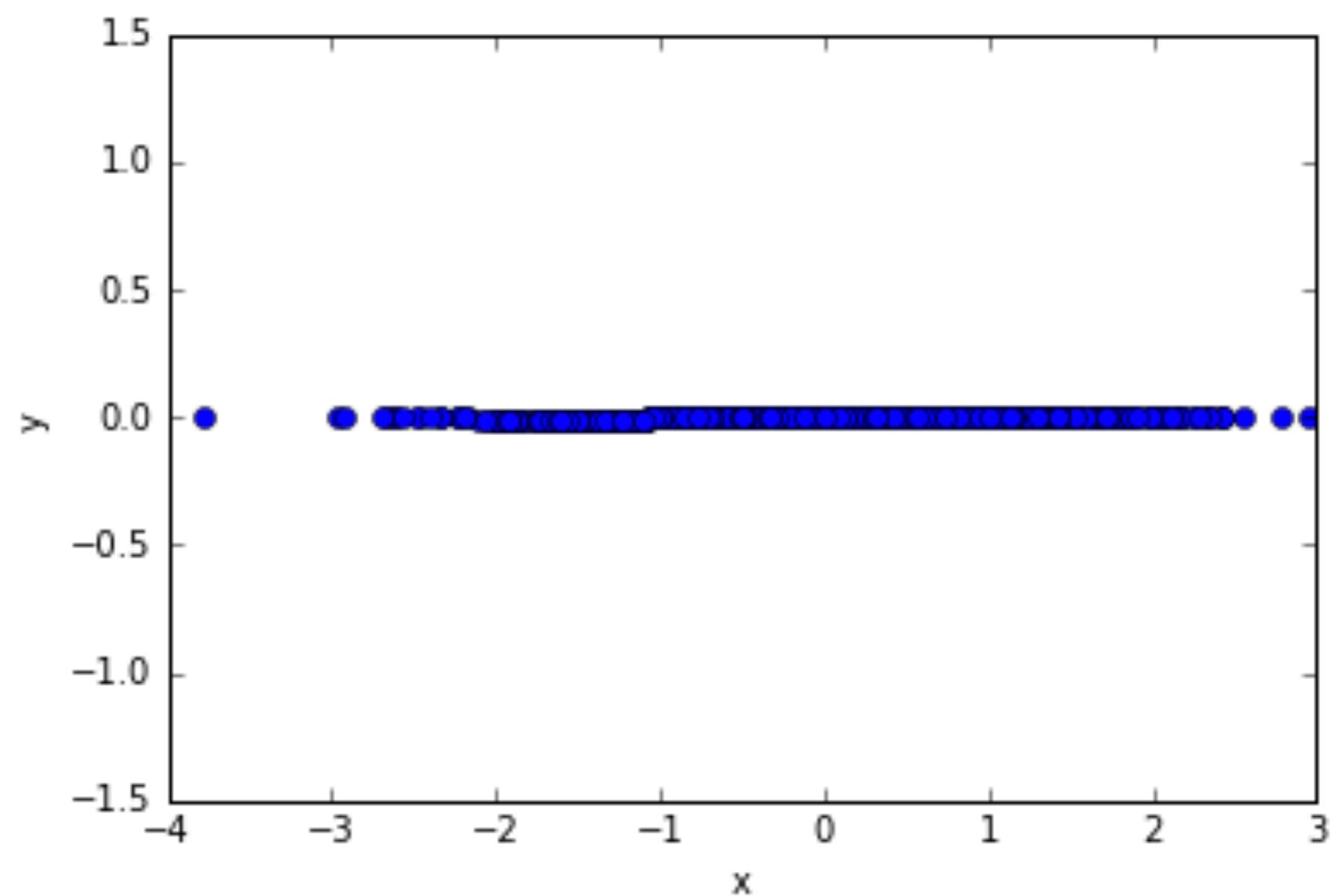


```
MDOM1207-2:~ LDY$ python tf_test.py
W tensorflow/core/platform/cpu_feature_guard.cc:128] CPU
eed up CPU computations.
W tensorflow/core/platform/cpu_feature_gua
eed up CPU computations.
W tensorflow/core/platform/cpu_feature_gua
d up CPU computations.
W tensorflow/core/platform/cpu_feature_gua
ed up CPU computations.
W tensorflow/core/platform/cpu_feature_gua
d up CPU computations.
[ 4.  3.]
```

저번 시간에 구현 예시였던 Neural Network는  
왜 Sin curve 조차 예측하지 못하는가? 무엇이 문제인가?

## □ 결과확인

```
#plt.plot(x_data, y_data, 'ro')
plt.plot(x_data, sess.run(output), 'bo')
plt.xlabel('x')
plt.xlim(-4,3)
plt.ylabel('y')
plt.ylim(-1.5,1.5)
plt.legend()
plt.show()
```



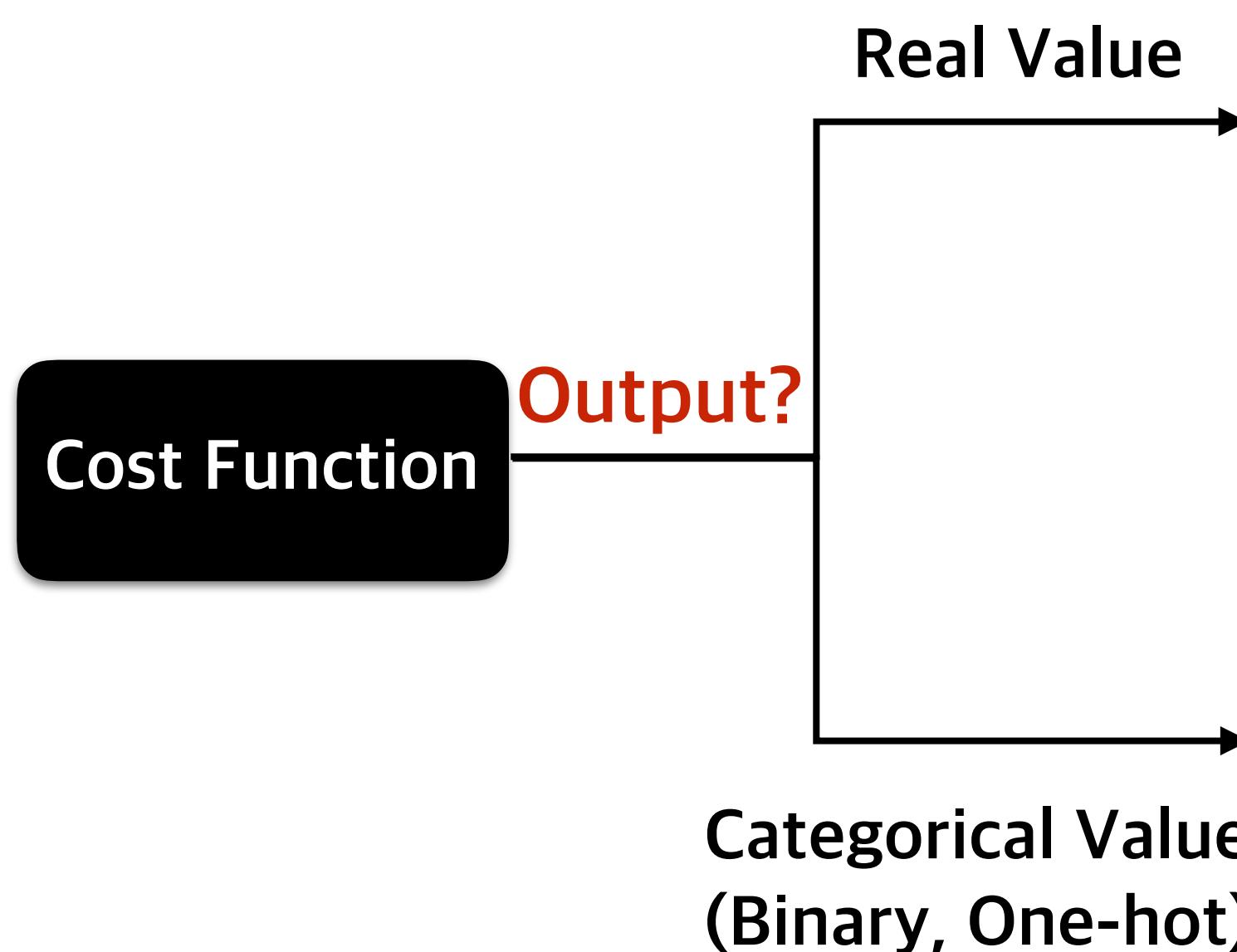
Why The Result is Garbage????

# Cost Functions

- Neural Network는 학습을 통해 성능을 향상시키며, 학습은 적절하게 **parameters(weights)**를 조절하는 것을 의미함
- 학습의 기준이 되는 Cost Function을 정의한 후, Cost Function 최소화를 위해 학습을 진행
- 일반적으로 용도에 따라 1) Least Square Method 또는 2) Cross-Entropy를 사용함

## Types of Cost Function

- 실제  $y$  값과 모델 출력값의 차이를 바탕으로 계산한 비용 함수



### <Least Square Method>

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

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

### <Cross-Entropy>

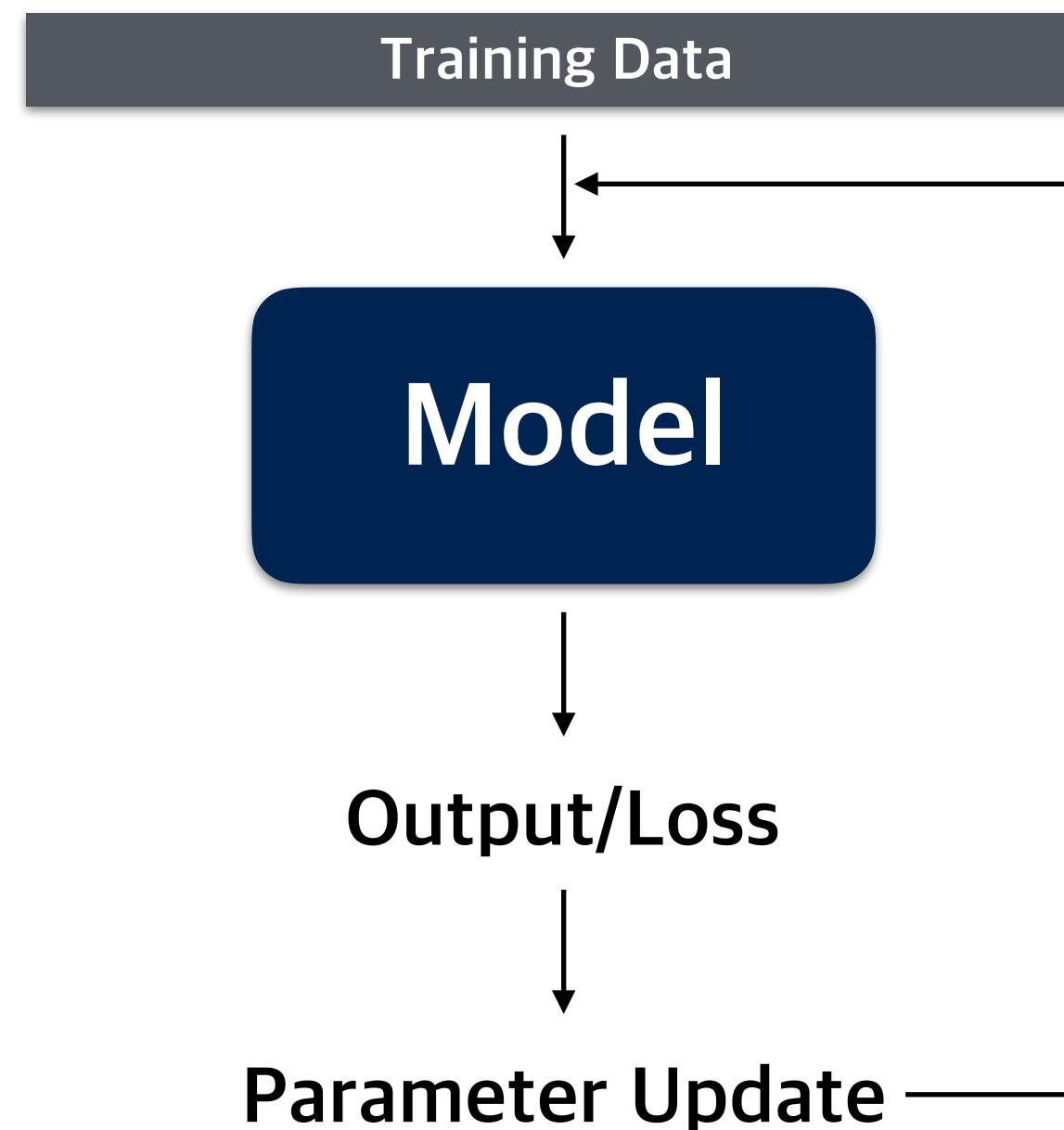
- 오분류에 대한 확률적인 비용을 의미
- Classifier (특히, softmax) 모델과 주로 사용
- $y$  값은 0 또는 1을 나타냄

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

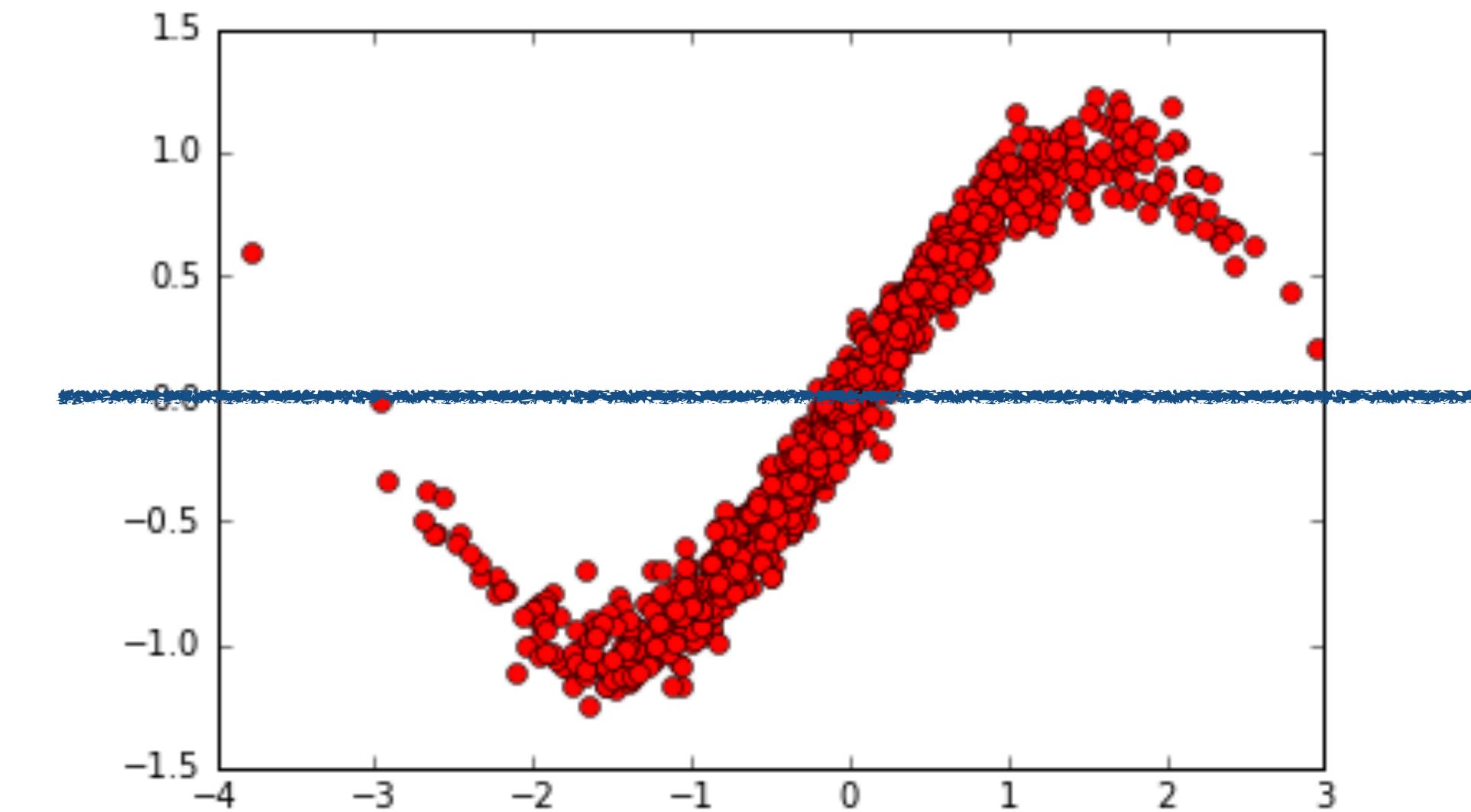
# Causes of Garbage Learning

- 이전 시간에 구현한 Neural Network는 (full) batch learning 방식의 학습을 진행함
  - Full Batch Learning: 전체 트레이닝 데이터를 파라미터 1회 업데이트에 사용
- 반복적인(iterative) 방식으로 파라미터를 업데이트하는 상황에서 심각한 Local Optimum Problem에 빠진 것으로 해석
  - 1) 데이터가 (0,0)을 중심으로 정규분포를 따르도록 골고루 퍼져있어 모든  $x$ 에 대한 예측을 0으로 하는 것에서 벗어나지 못함
  - 2) 골고루 퍼져있는 데이터 전체에 대한 비용 함수의 평균으로 파라미터를 업데이트함

<Full Batch Learning 예시>



<예측하고자 하는 데이터 분포>

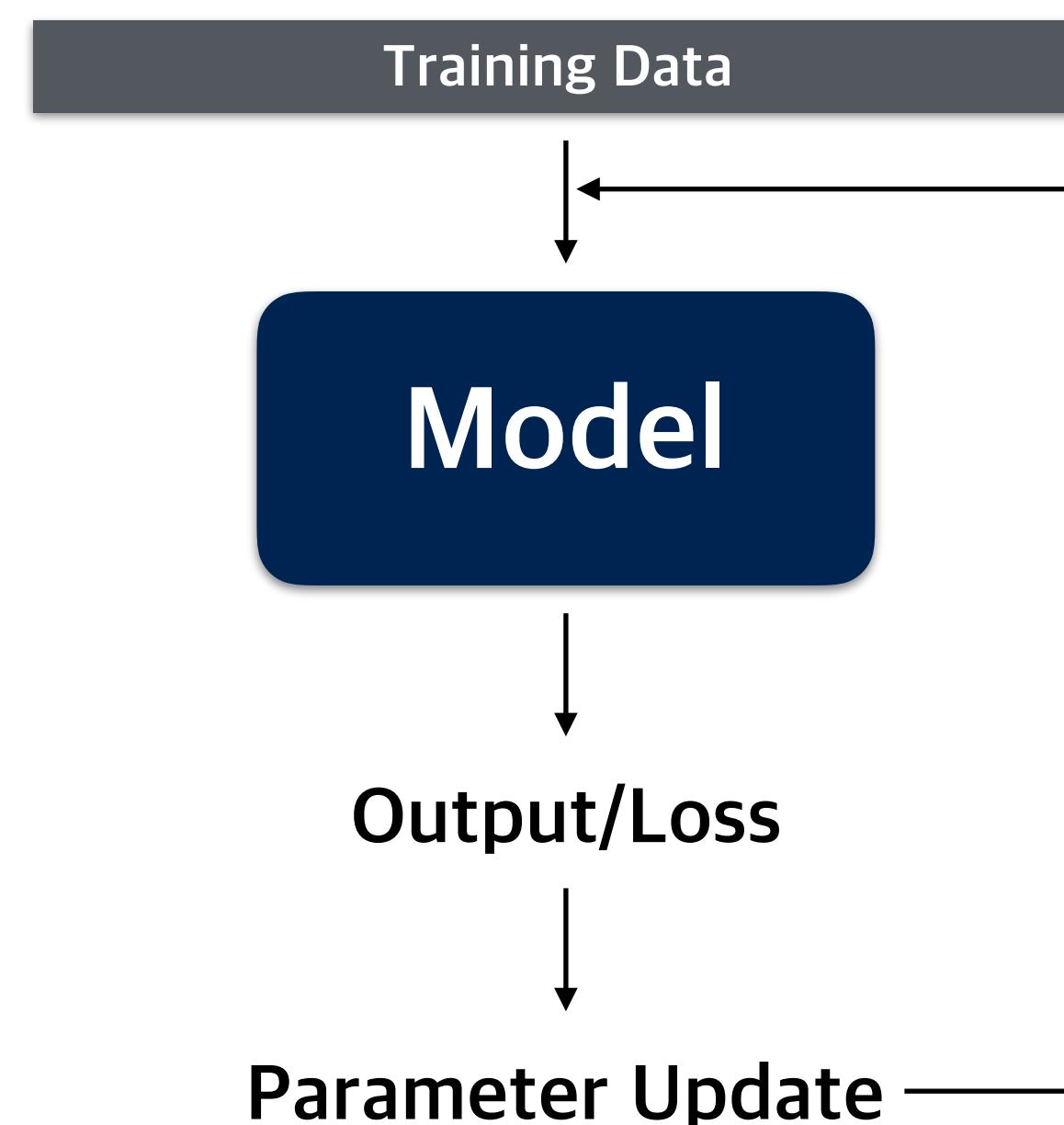


# Mini-batch Learning for Effective Learning

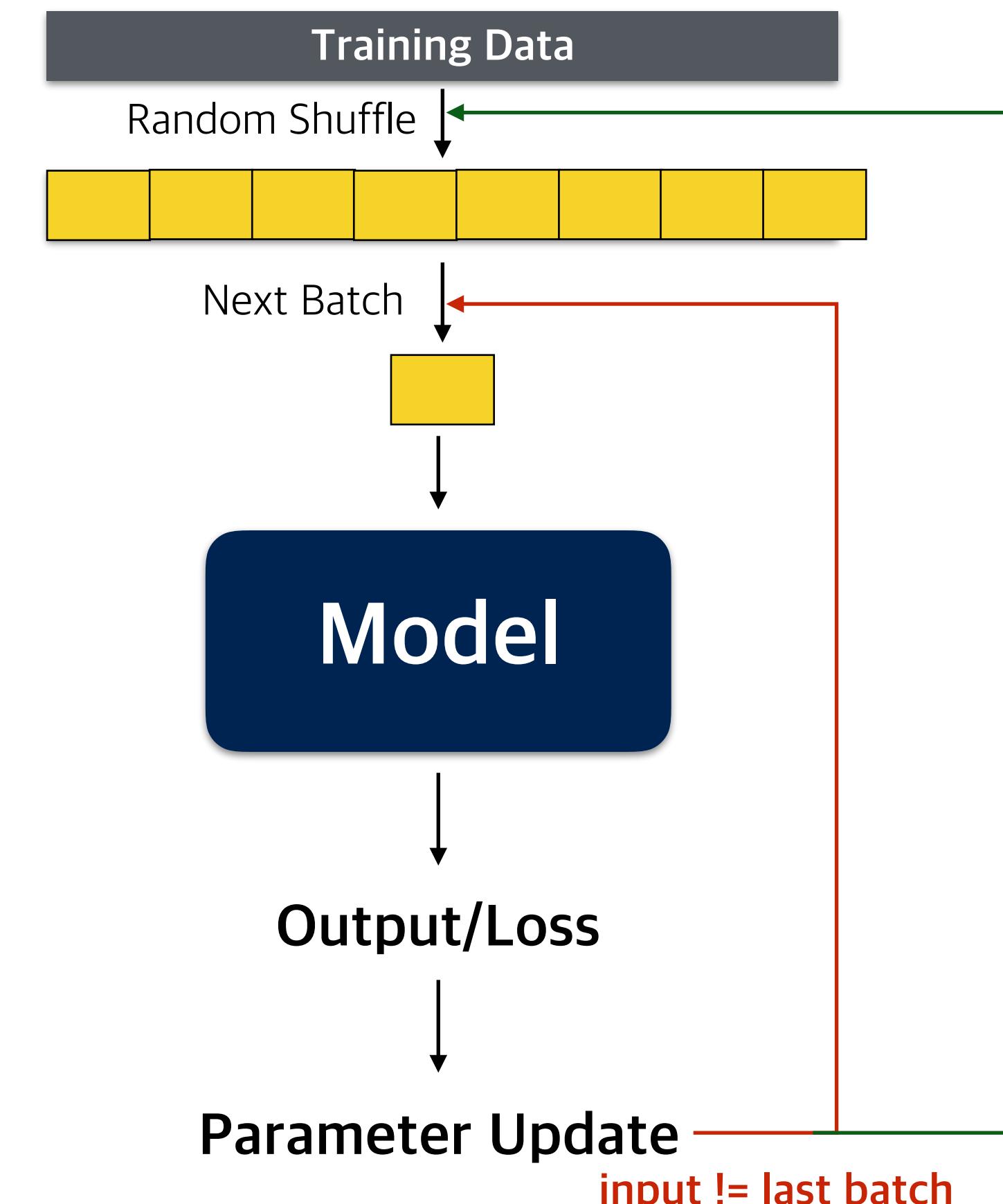
## ❑ Mini Batch Learning을 통해 학습의 속도를 높이고 local optimum에 빠지는 문제를 어느정도 해결 가능

- 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>



input = last batch

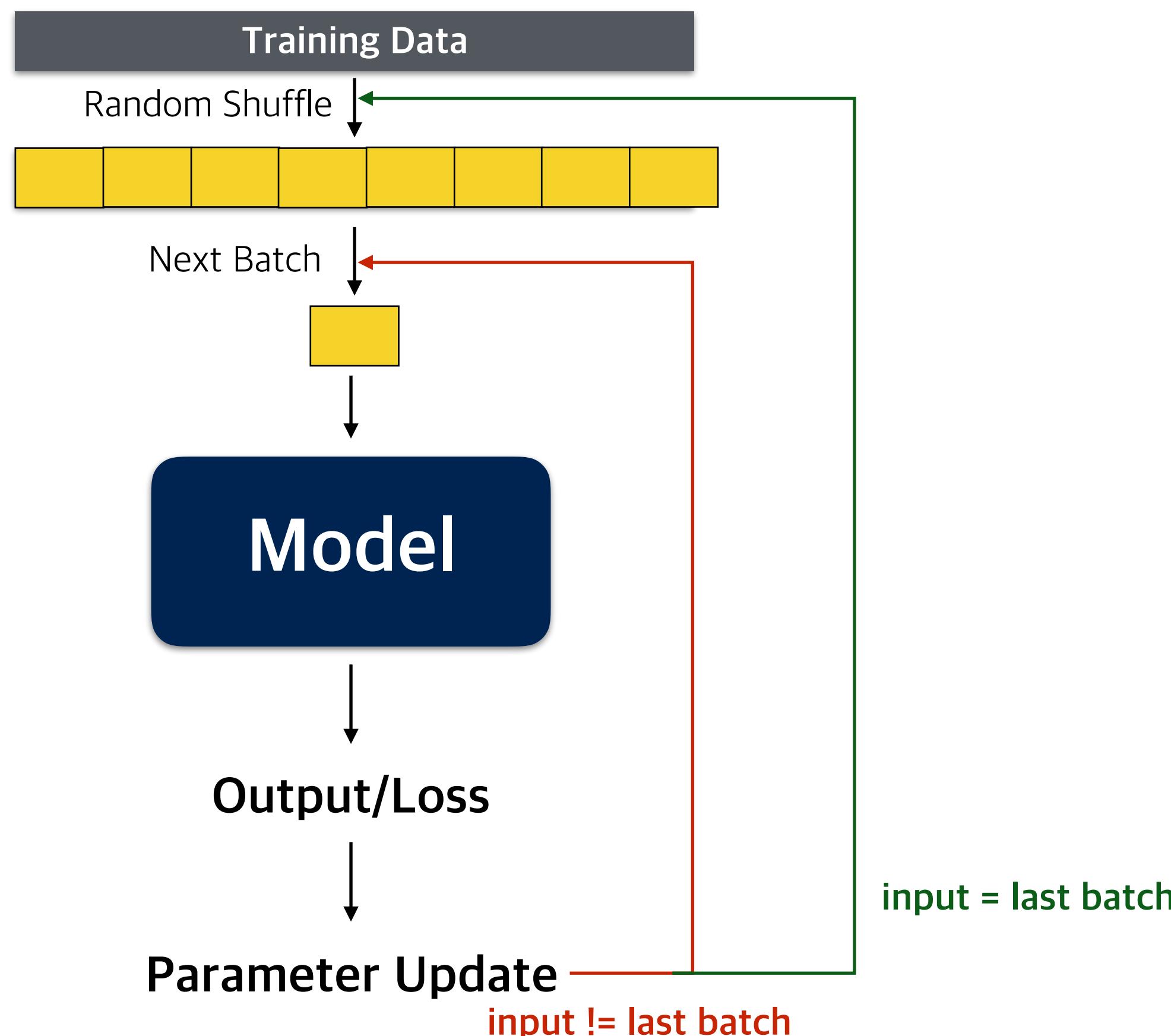
eXem

# Implementation of mini-batch

## ❑ 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

<mini-Batch Learning 예시>



# Implementation of mini-batch

## □ 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

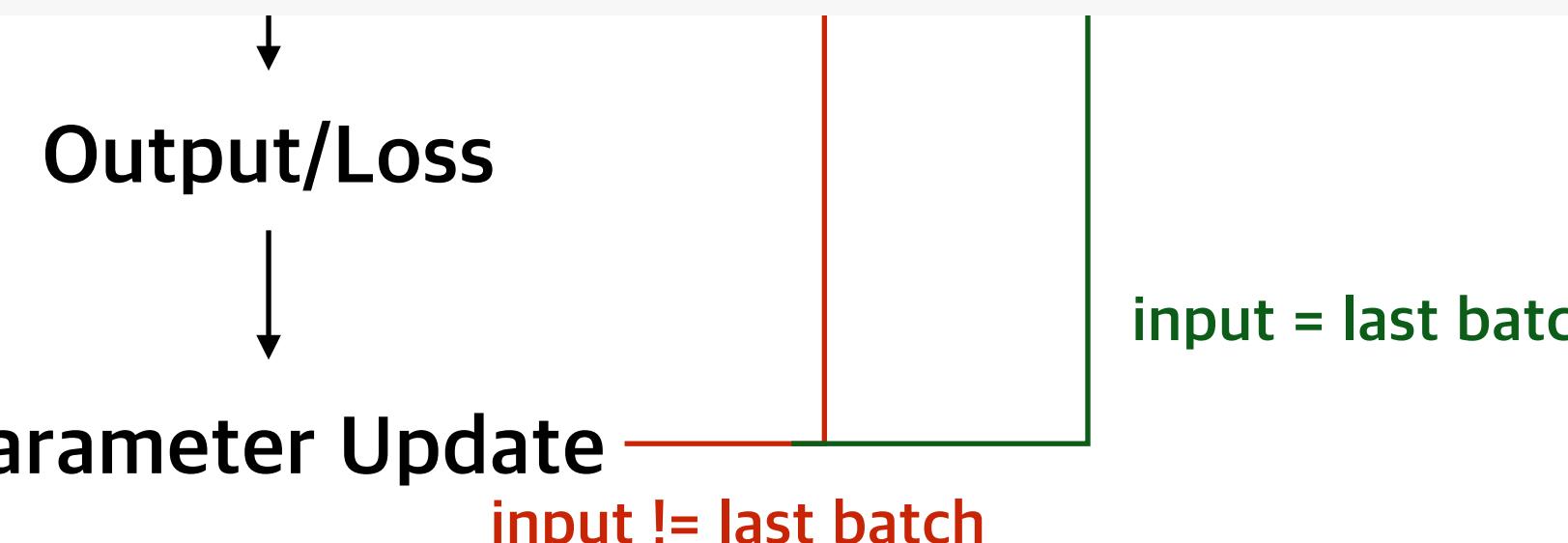
<mini-Batch Learning 예시>

BATCH\_SIZE에 따라  
한 번에 입력되는 데이터의 수가 달라지므로  
tf.placeholder 사용

Training Data

```
input_data = tf.placeholder(tf.float32, shape=[None, 1])  
output_data = tf.placeholder(tf.float32, shape=[None, 1])
```

```
W1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))  
W2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0))  
W_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0))
```

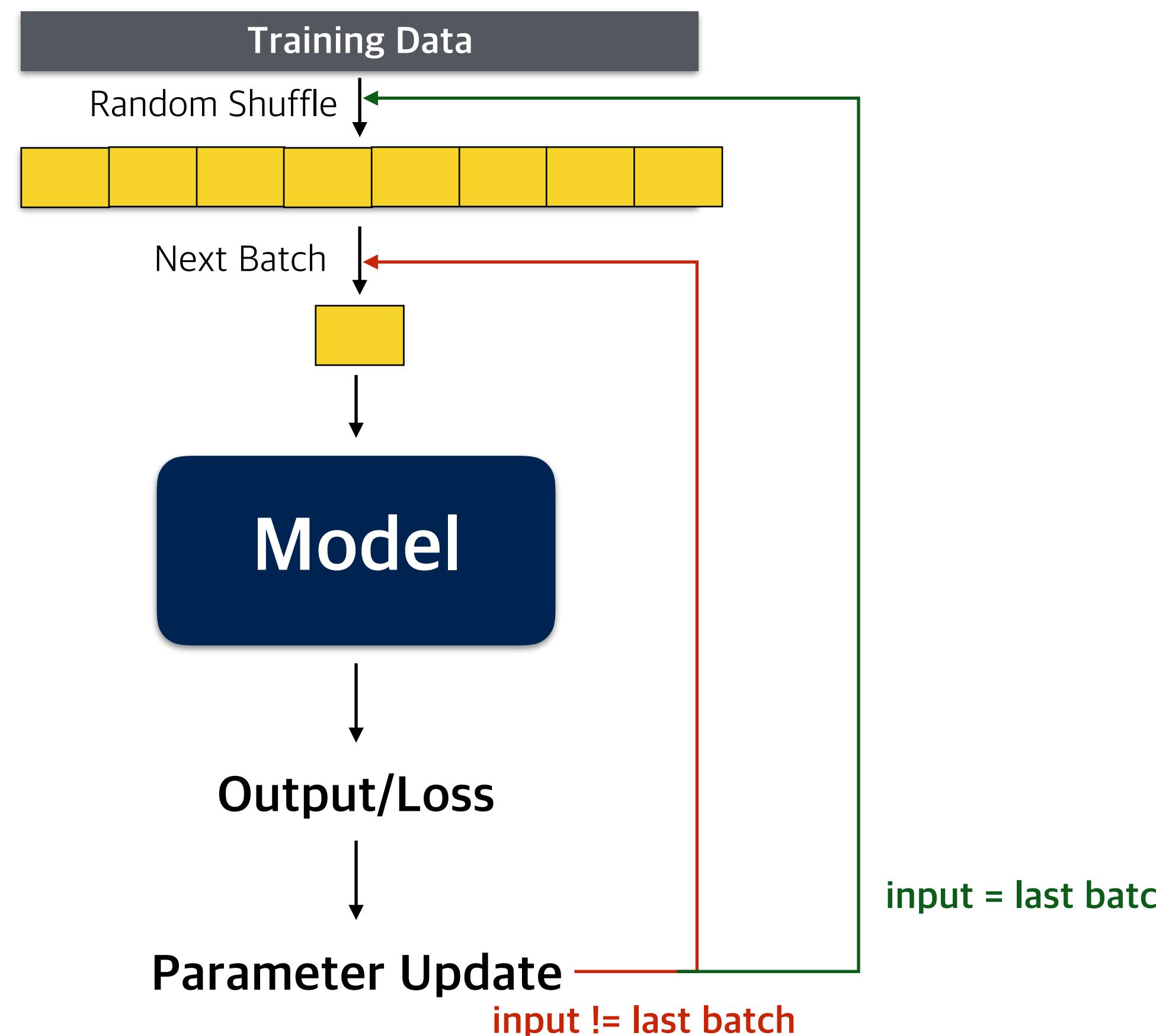


# Implementation of mini-batch

## □ 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

### <mini-Batch Learning 예시>



```
input_data = tf.placeholder(tf.float32, shape=[None,1])
output_data = tf.placeholder(tf.float32, shape=[None,1])

W1 = tf.Variable(tf.random_uniform([1,5], -1.0, 1.0))
W2 = tf.Variable(tf.random_uniform([5,3], -1.0, 1.0))
W_out = tf.Variable(tf.random_uniform([3,1], -1.0, 1.0))

#Train several steps

for step in range(5000):
    index = 0
    x_data, y_data = shuffle_data(x_data, y_data)
    for batch_iter in range(BATCH_NUM-1):
        feed_dict = {input_data: x_data[index:index+BATCH_SIZE],
                    output_data: y_data[index:index+BATCH_SIZE]}
        sess.run(train, feed_dict = feed_dict)
        print("Step, Loss Value")
        print(step, sess.run(loss, feed_dict = feed_dict))
        index += BATCH_SIZE
```

BATCH\_SIZE에 따라  
한 번에 입력되는 데이터의 수가 달라지므로  
tf.placeholder 사용

1 epoch 시작시 데이터 random shuffle  
이 후, BATCH\_SIZE 단위로 파라미터 업데이트

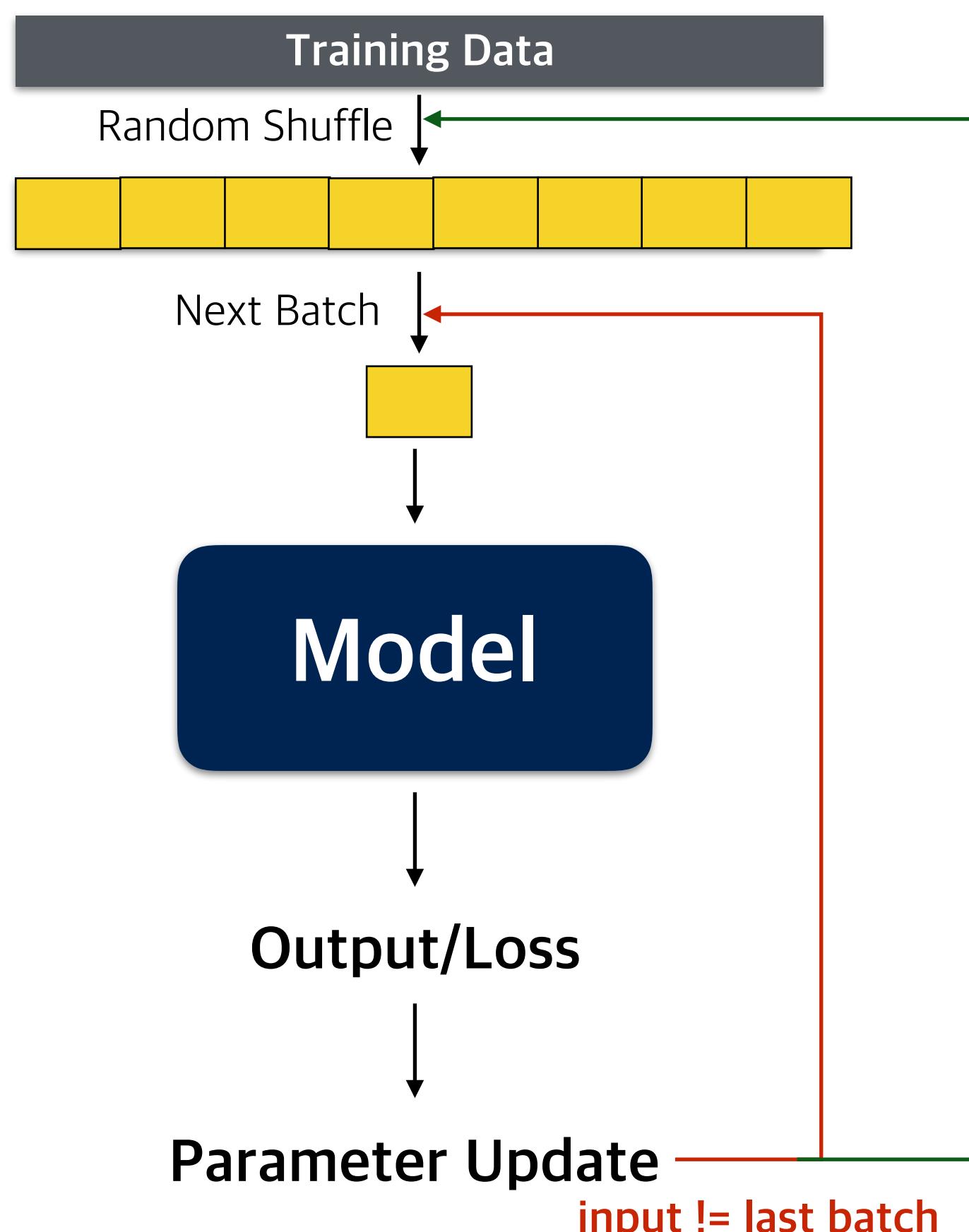
# Implementation of mini-batch

## □ 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

<mini-Batch Learning 예시>

BATCH\_SIZE에 따라  
한 번에 입력되는 데이터의 수가 달라지므로  
tf.placeholder 사용



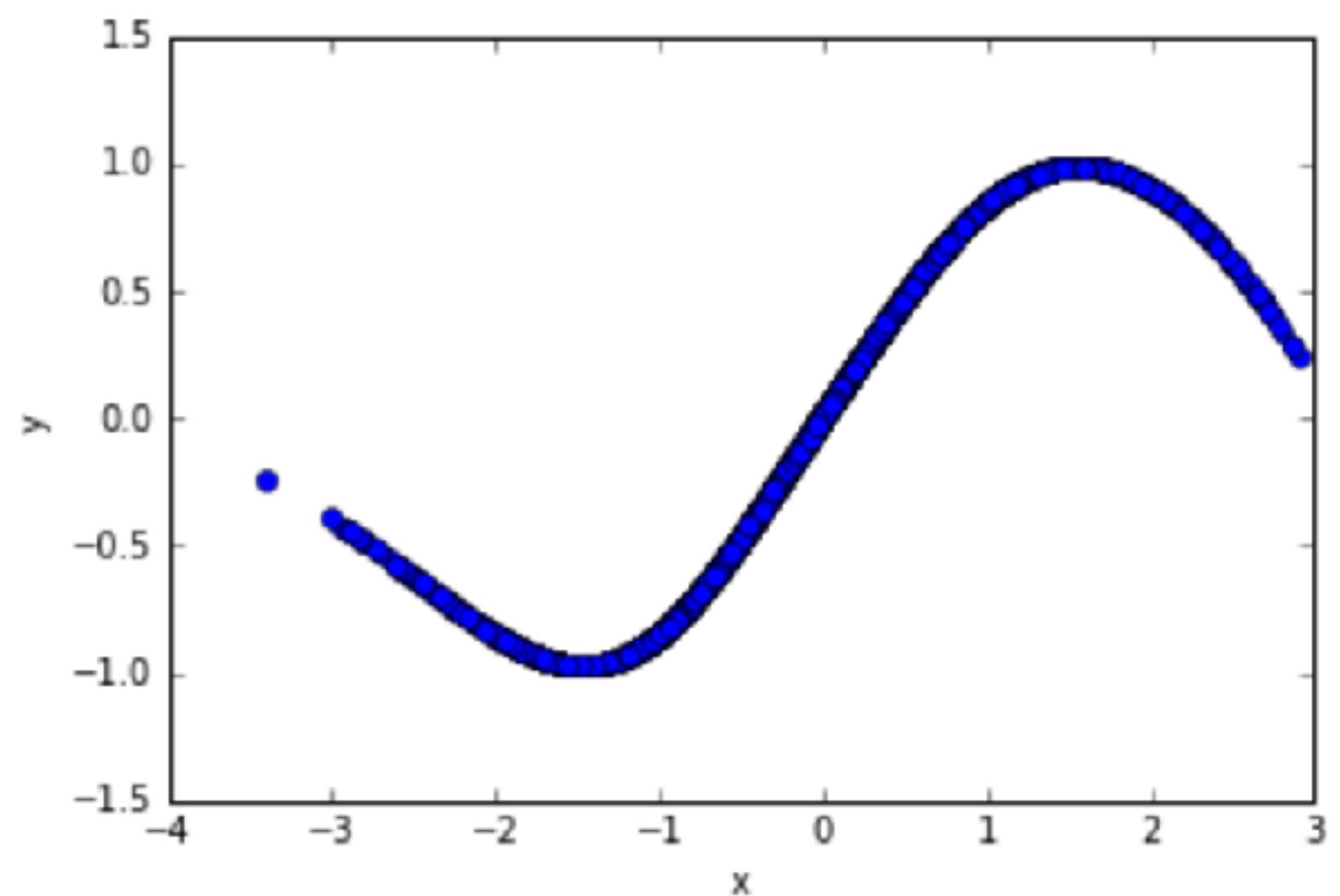
#Train several steps

```
for step in range(5000):
    index = 0
    x_data, y_data = shuffle_data(x_data, y_data)
    for batch_iter in range(BATCH_NUM-1):
        feed_dict = {input_data: x_data[index:index+BATCH_SIZE],
                    output_data: y_data[index:index+BATCH_SIZE]}
        sess.run(train, feed_dict = feed_dict)
        print("Step, Loss Value")
        print(step, sess.run(loss, feed_dict = feed_dict))
        index += BATCH_SIZE
```

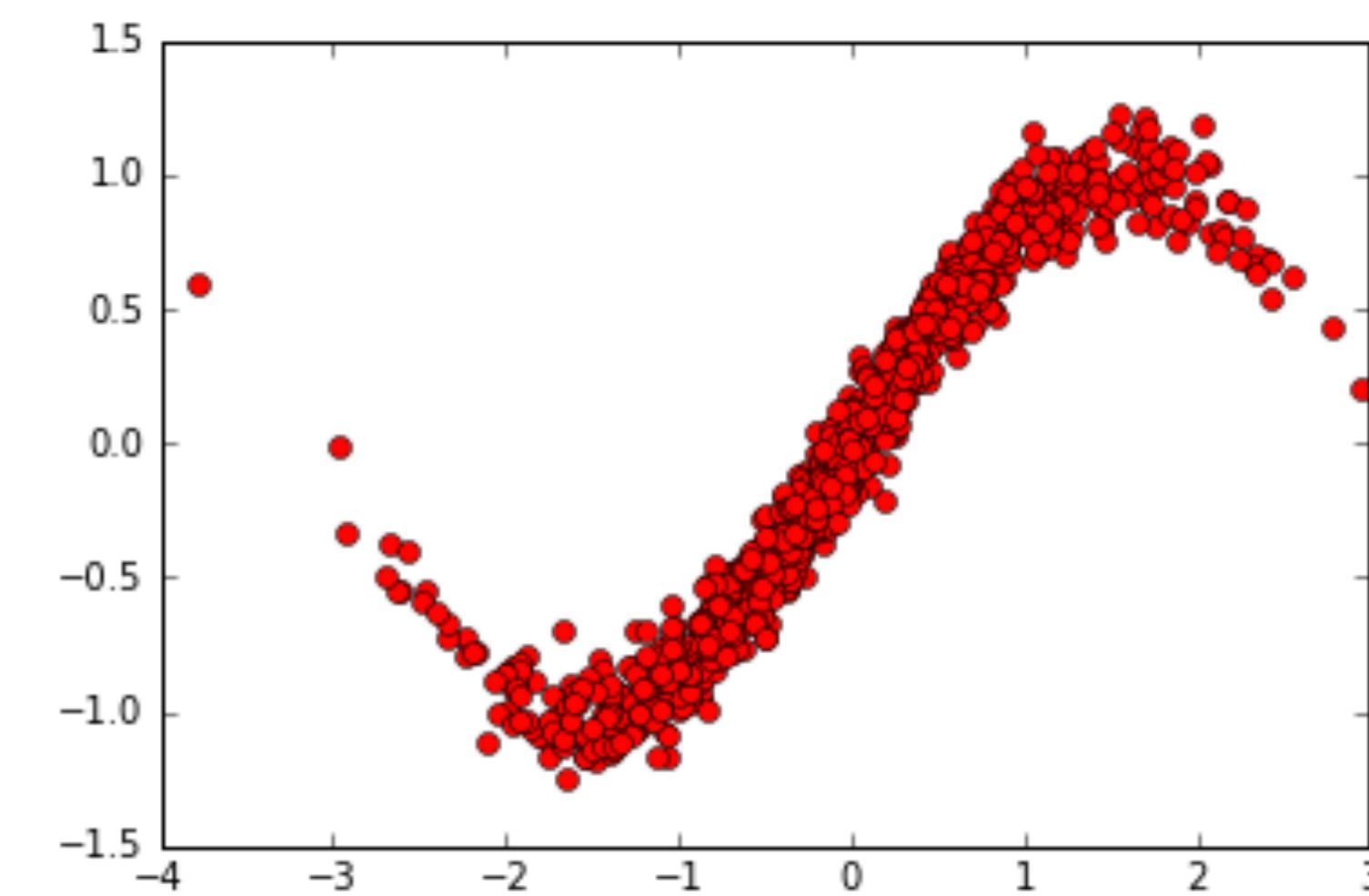
input = last batch

# Successful Result

```
#plt.plot(x_data, y_data, 'ro')
feed_dict = {input_data: x_data}
plt.plot(x_data, sess.run(output, feed_dict=feed_dict), 'bo')
plt.xlabel('x')
plt.xlim(-4,3)
plt.ylabel('y')
plt.ylim(-1.5,1.5)
plt.legend()
plt.show()
```



<Prediction Data>

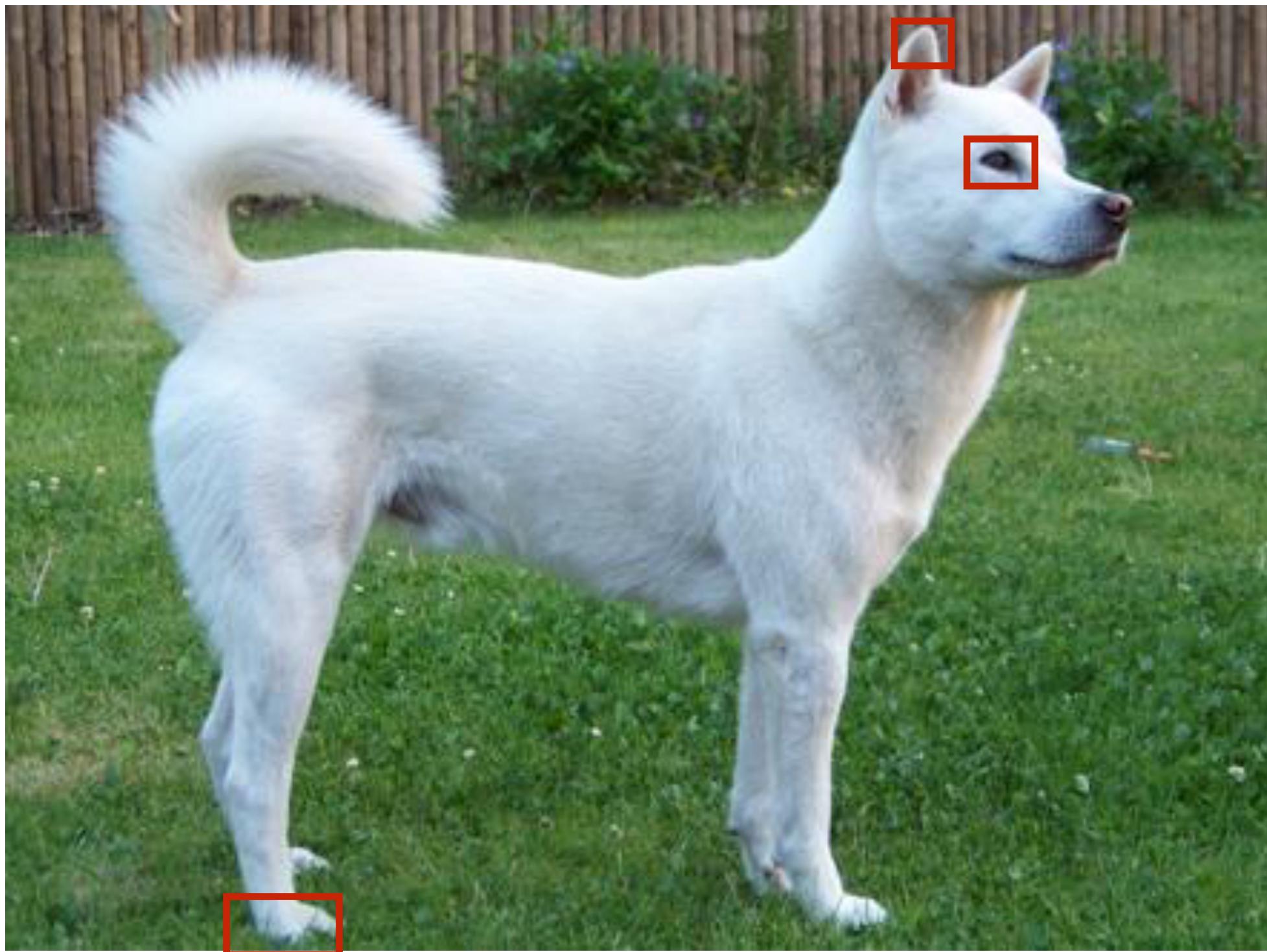


<Original Data>

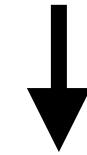
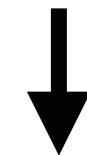
# Convolutional Neural Network

# Introduction

## □ Pop Quiz !



Q: “Which one is in the left picture,  
Dog or Cat?”



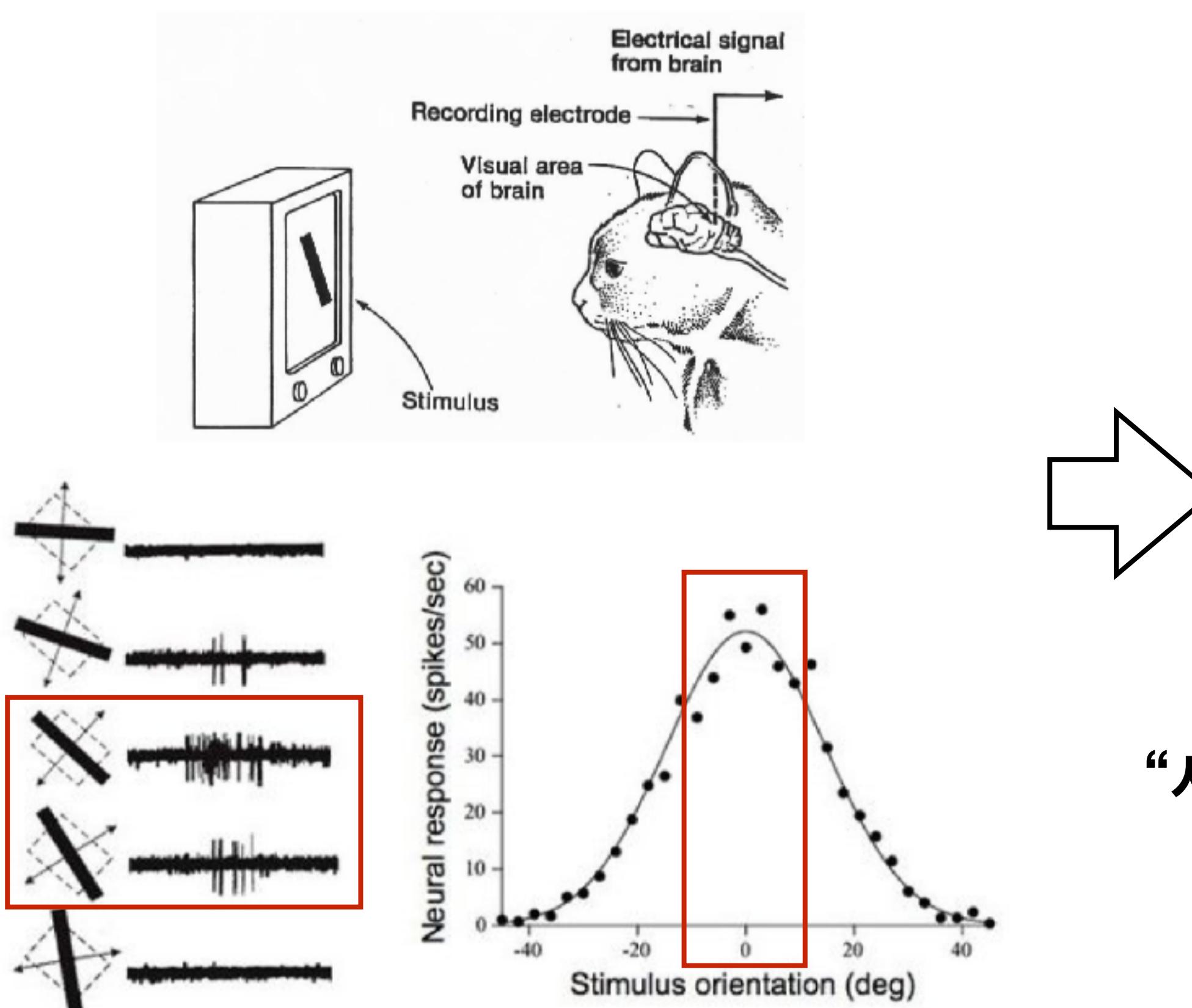
A: “Dog!!”

인간은 이미지 전체 보다는 이미지의 **특정 부분**을 통해 시각적 특징을 인지함

# Introduction

## □ Hubel & Wiesel Experiment (1968)

- 고양이의 수용 영역(receptive field) 내 **자극체(stimulus)**에 따른 대뇌 시각 피질의 **뉴런의 활성도 변화**를 확인한 실험
- 수용 영역을 고정시키고, 선(line, stimulus)을 회전시켜가며 뉴런의 활성도를 모니터링



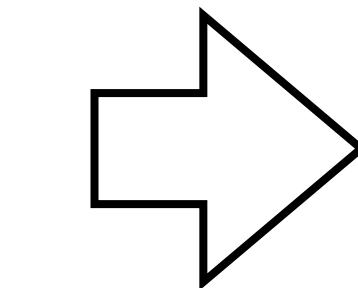
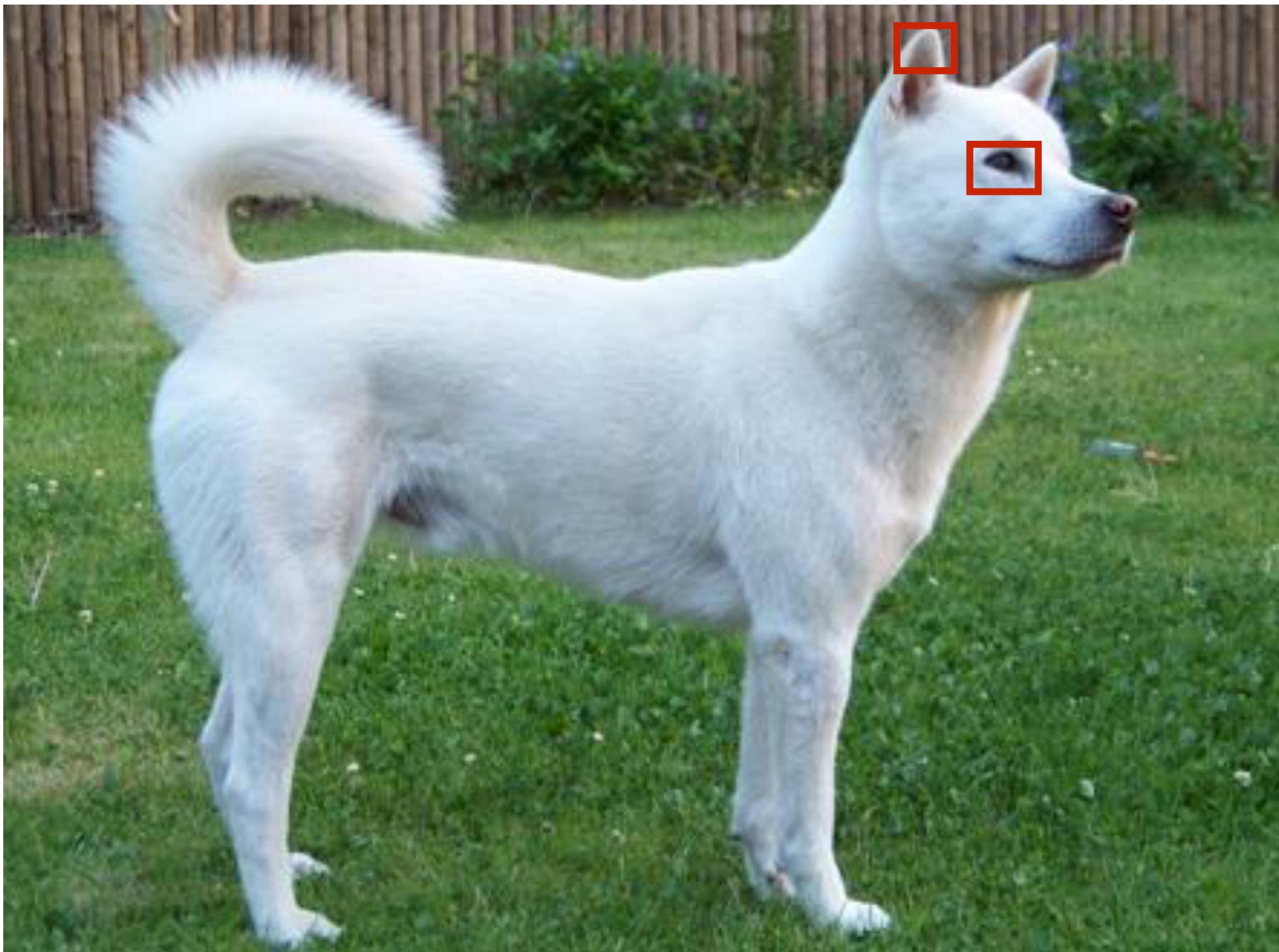
### Experiment Result

- 뉴런은 선이 망막 내 특정 부분에 위치할 때만 활성화됨을 관찰
- 뉴런의 활성 여부는 회전에 따른 선의 방향에 의존하여 결정됨
- 뉴런은 선이 특정 방향으로 움직일 때만 활성화됨

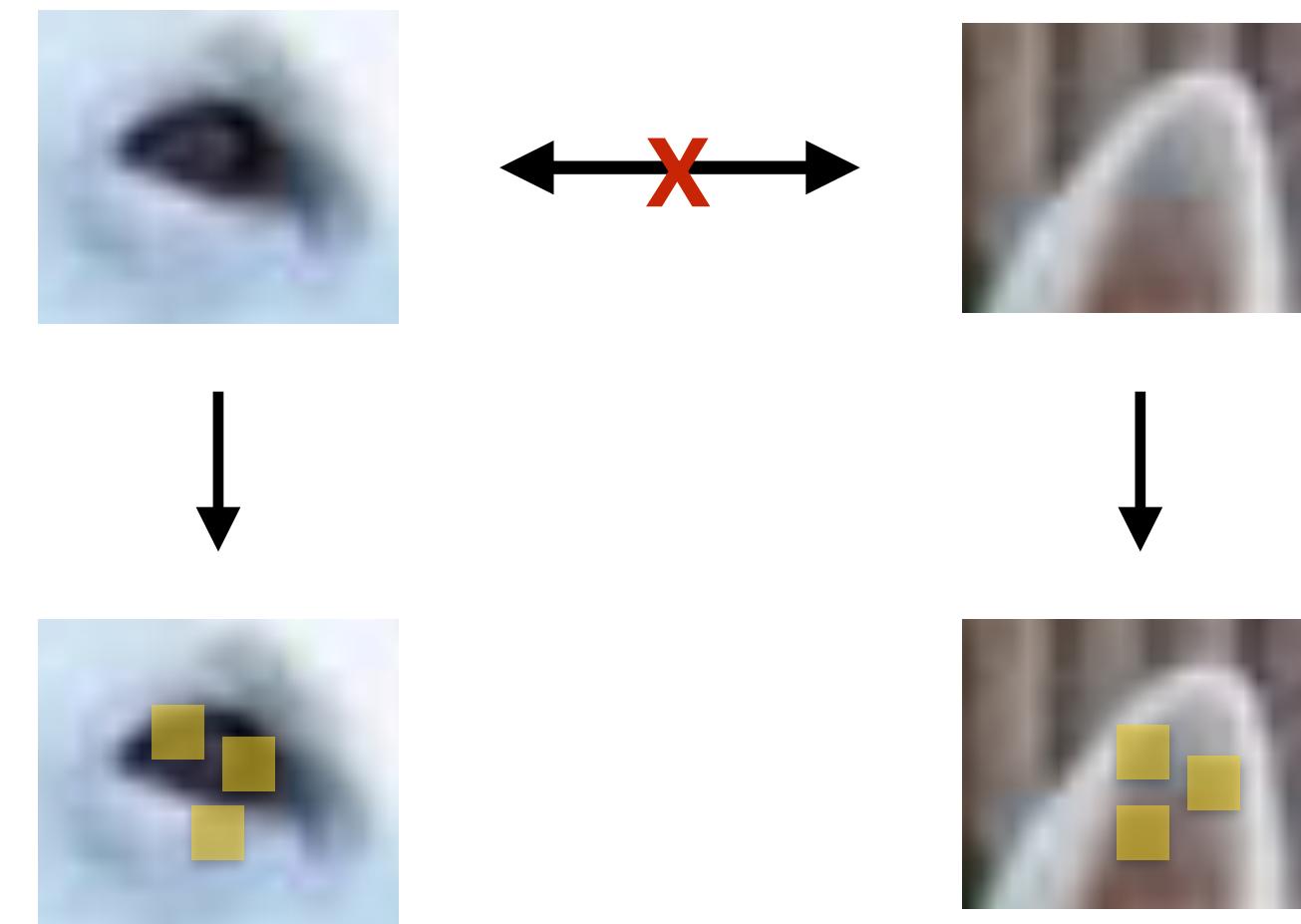
“**시각령(visual cortex)의 뉴런들은 이미지 전체가 아니라 이미지의 특정한 모양이나 특징에 반응하고 활성화됨**”

## Introduction

- 이미지 내 모든 픽셀(pixel)들이 서로 연관 관계를 가지고 있는 것은 아님
- 이미지 내 픽셀들은 지역적으로 관련이 높으며(**locally related**), 멀리 떨어진 경우 수치적 관련성이 매우 낮음



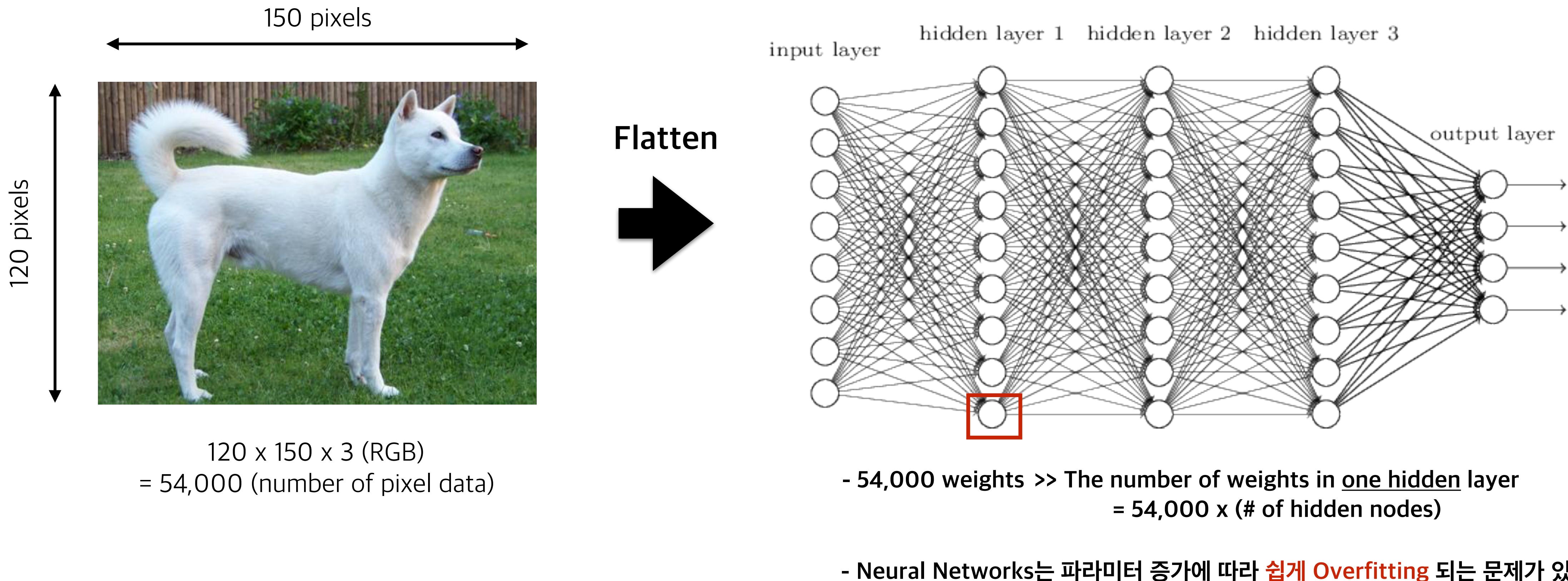
“The spatial correlation is local”



인접한 픽셀들은  
높은 지역적 관계(local relationship)를 갖으며  
공간적인 상관관계(spatial correlation)를 나타냄

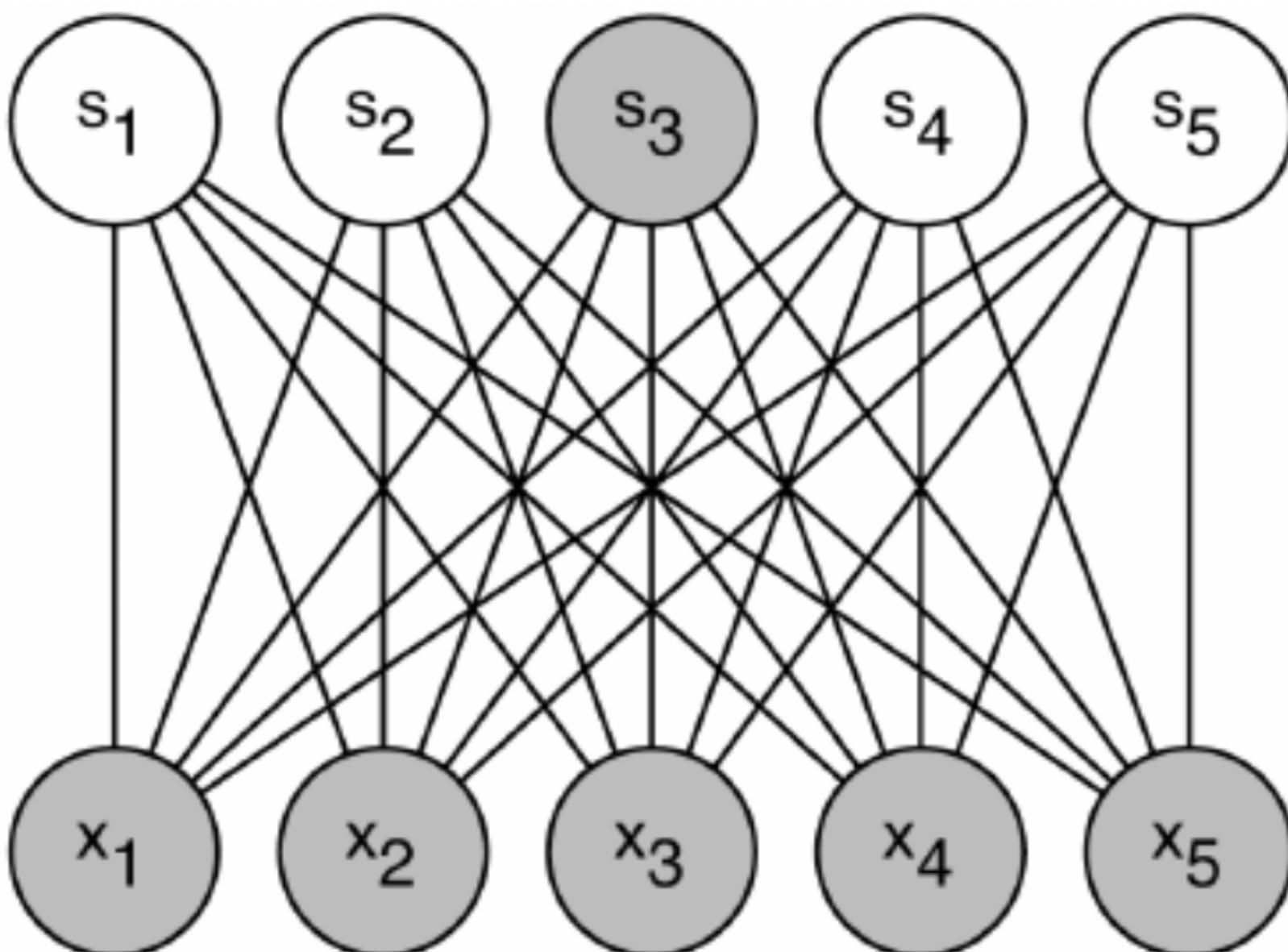
# Limitations of Artificial Neural Network

- ▣ 기존의 Neural Network는 이미지를 처리하거나 시각적인 특징을 다루는 상황에서 한계를 가지고 있음
  - 1) 이미지를 처리하는데, 극도로 **많은 수의 파라미터**가 필요함 (Extremely large number of parameters)
  - 2) 기존 Neural Network은 **local relationship**을 고려하지 않음



# Limitations of Artificial Neural Network

- ▣ 기존의 Neural Network는 이미지를 처리하거나 시각적인 특징을 다루는 상황에서 한계를 가지고 있음
  - 1) 이미지를 처리하는데, 극도로 많은 수의 파라미터가 필요함 (Extremely large number of parameters)
  - 2) 기존 Neural Network은 local relationship을 고려하지 않음

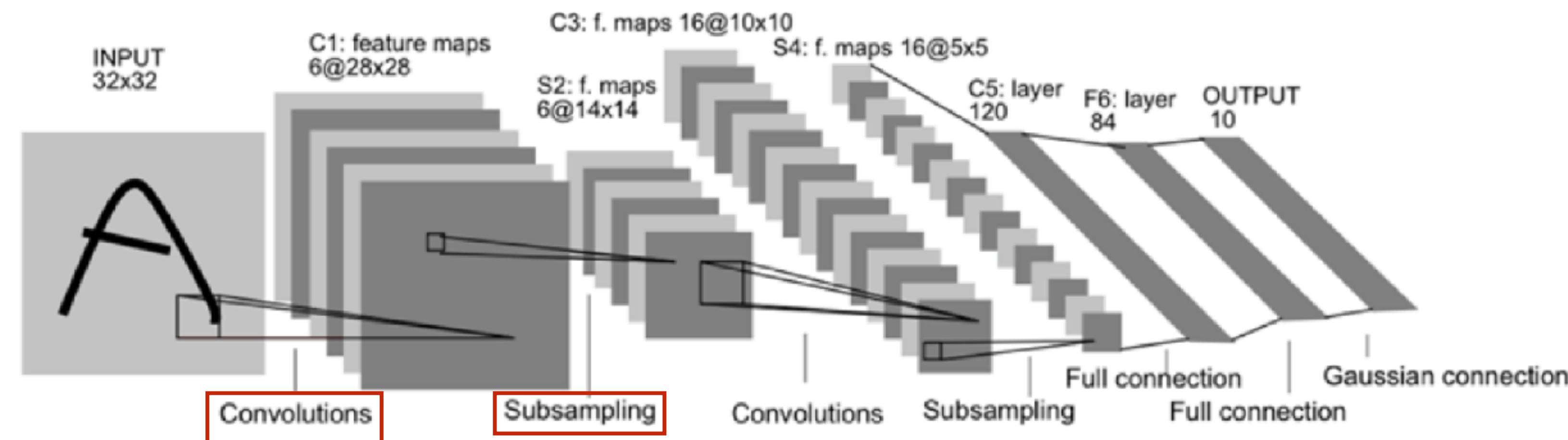


- 모든 Input 뉴런들이  $s_3$ 를 계산할 때 고려되어야 함
- 시각적 특징은 모든 Input에 의해 결정되기보다, 인접한 데이터들의 지역적 관계로 결정됨
- 기존 Neural Network는 지역적 관계를 가지는 데이터에 대하여 계산적으로 매우 비효율적임

# Convolutional Neural Network

- Convolutional Neural Network (ConvNet)은 앞서 언급된 기존 Neural Network의 단점을 보완하는 구조임
- 위치에 관계없이 시각적 특징을 추출하기 위하여 제안되었으며, 지역적 연결(local connection)과 파라미터 공유(shared weight)을 포함한 Neural Network 구조를 지님 (Lecun, 1989).
- 1) convolution, 2) pooling 필터를 통해 local connection과 shared weight가 구현됨

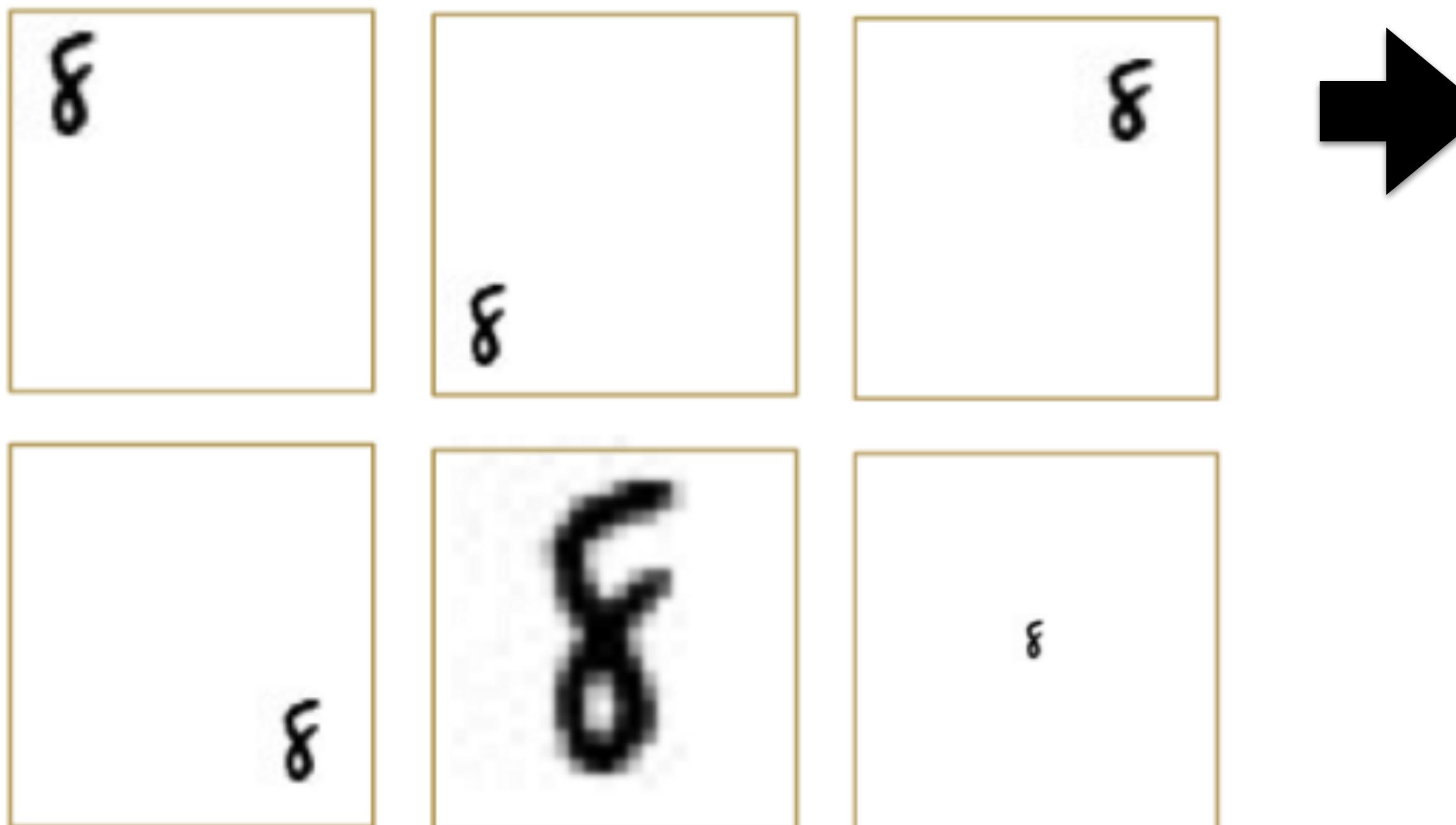
<Convolutional Neural Network LeNet5>



- 2d, 3d의 Input 구조를 유지하고,  
작은 크기의 receptive field를 사용하여  
픽셀 사이의 이미지 특징을 학습하고, spatial relationship을 유지함
- 모서리, 윤곽선 등의 작고 의미 있는 특징을 필터를 시작으로  
복잡한 특징을 검출하는 필터를 학습을 통해 도출함

# Invariant Recognition of Handwrite digit

- ConvNet은 이미지의 특징을 위치에 관계없이 추출할 수 있다는 장점이 있음 (invariant recognition)



ConvNet은 숫자의 위치에 초점을 맞추는 것이 아닌  
이미지에 포함된 시각적 특징에 초점을 맞춤



“Focus on **red box** information,  
not its location.”

# Architecture of ConvNet

# Main Components of Convolutional Neural Network

## □ ConvNet의 계산은 크게 4가지 구성요소로 이루어짐

### Convolutional Layer

- Filter, Kernel, Weights, Convolution matrix, Mask
- 작은 크기의 필터를 통해 입력 데이터에 대하여 Convolution 연산을 진행함

### Activation Layer

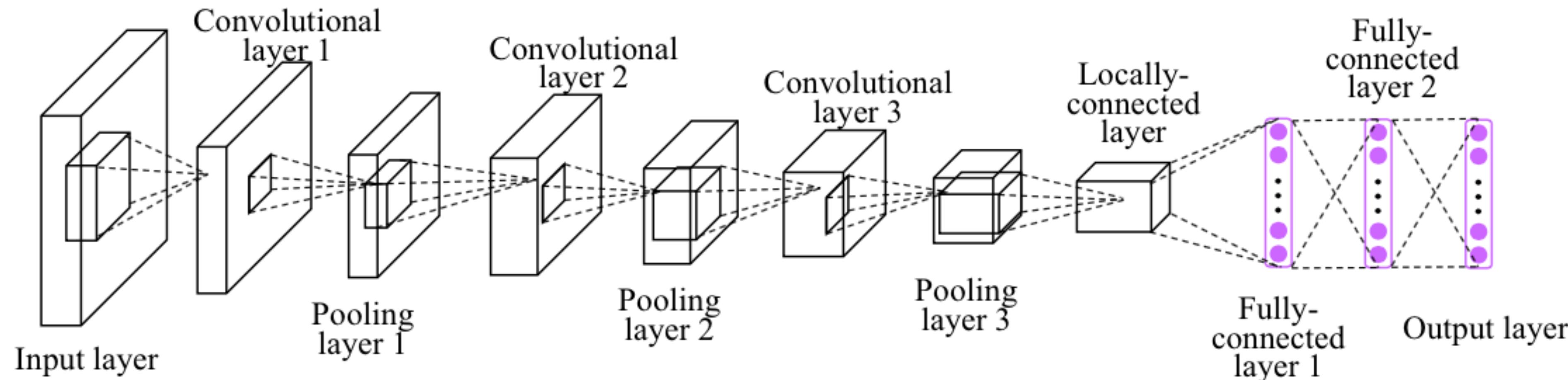
- Convolutional Layer의 결과는 Linear Combination과 같음
- 비선형 특징을 추출하기 위해 activation function 이용
- 특히 ReLU (Rectified Linear Unit) 사용

### Pooling Layer

- 입력 데이터의 크기를 줄이기 위한 Subsampling
- Overfitting 예방하는 효과

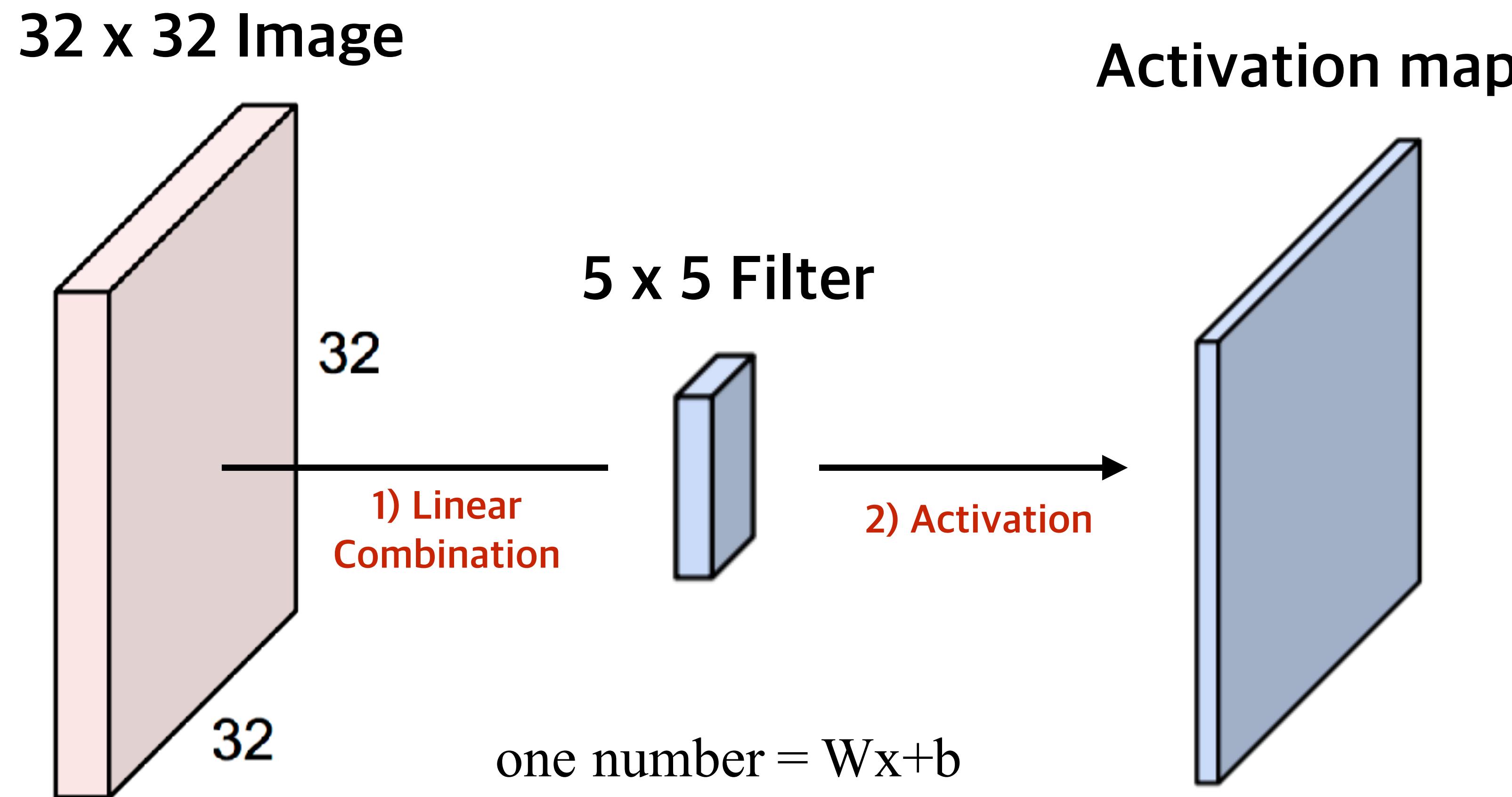
### Full Connected Layer

- ConvNet을 통해 추출된 특징을 종합하여 연산함
- 모델의 마지막 부분에 위치



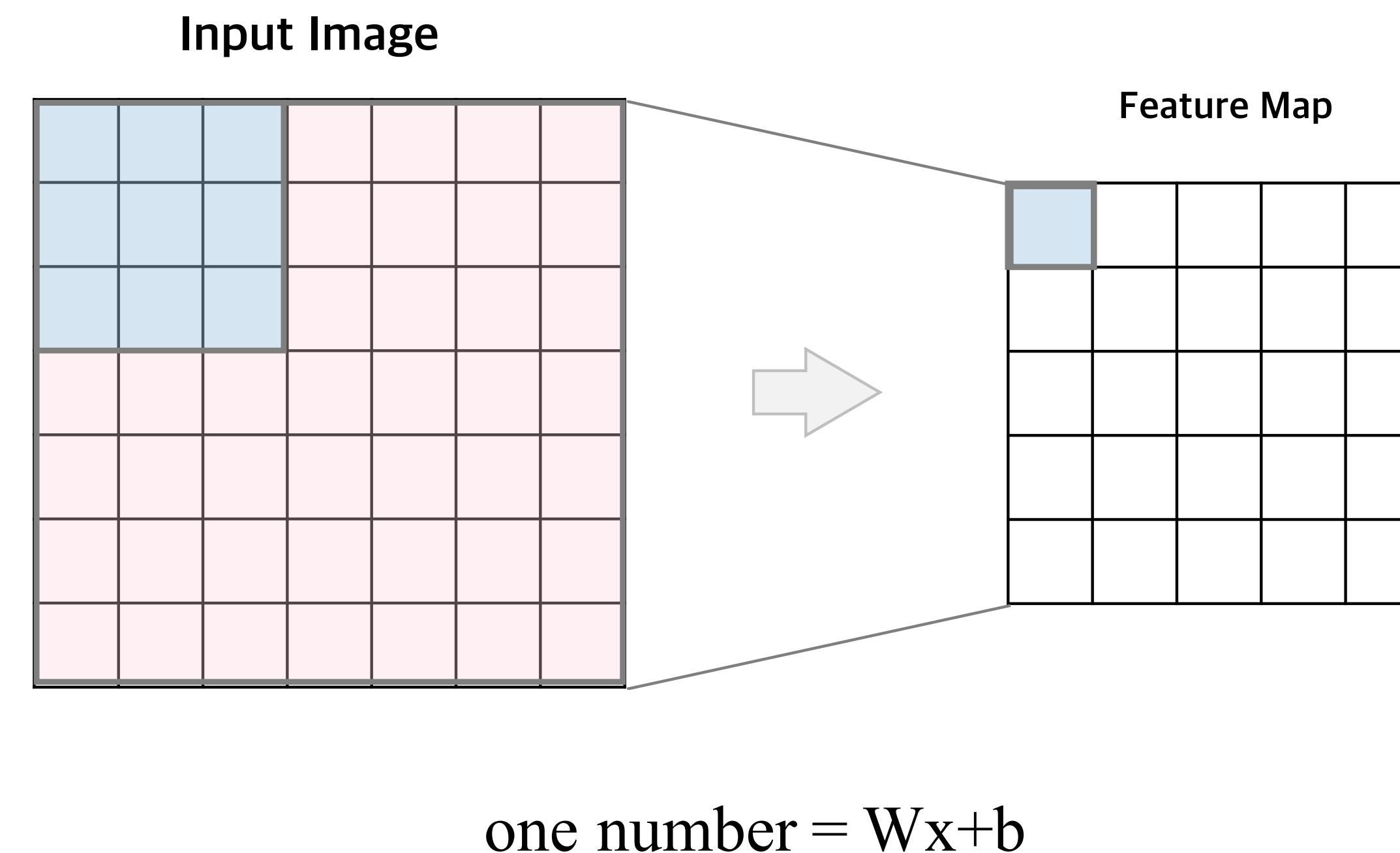
## Convolutional Filter

- Convolutional layer는 다양한 filter(kernel)로 구성되어 있으며, 다양한 시각적 특징(visual feature)를 찾는 역할을 함
- 특정 Input에 대하여 하나의 convolutional filter는 하나의 activation map을 결과로 구성함
- 기본 계산 프로세스는 기존 Neural Network와 동일: 1) Linear Combination, 2) Activation



# Calculating Kernels on Image: Linear Combination

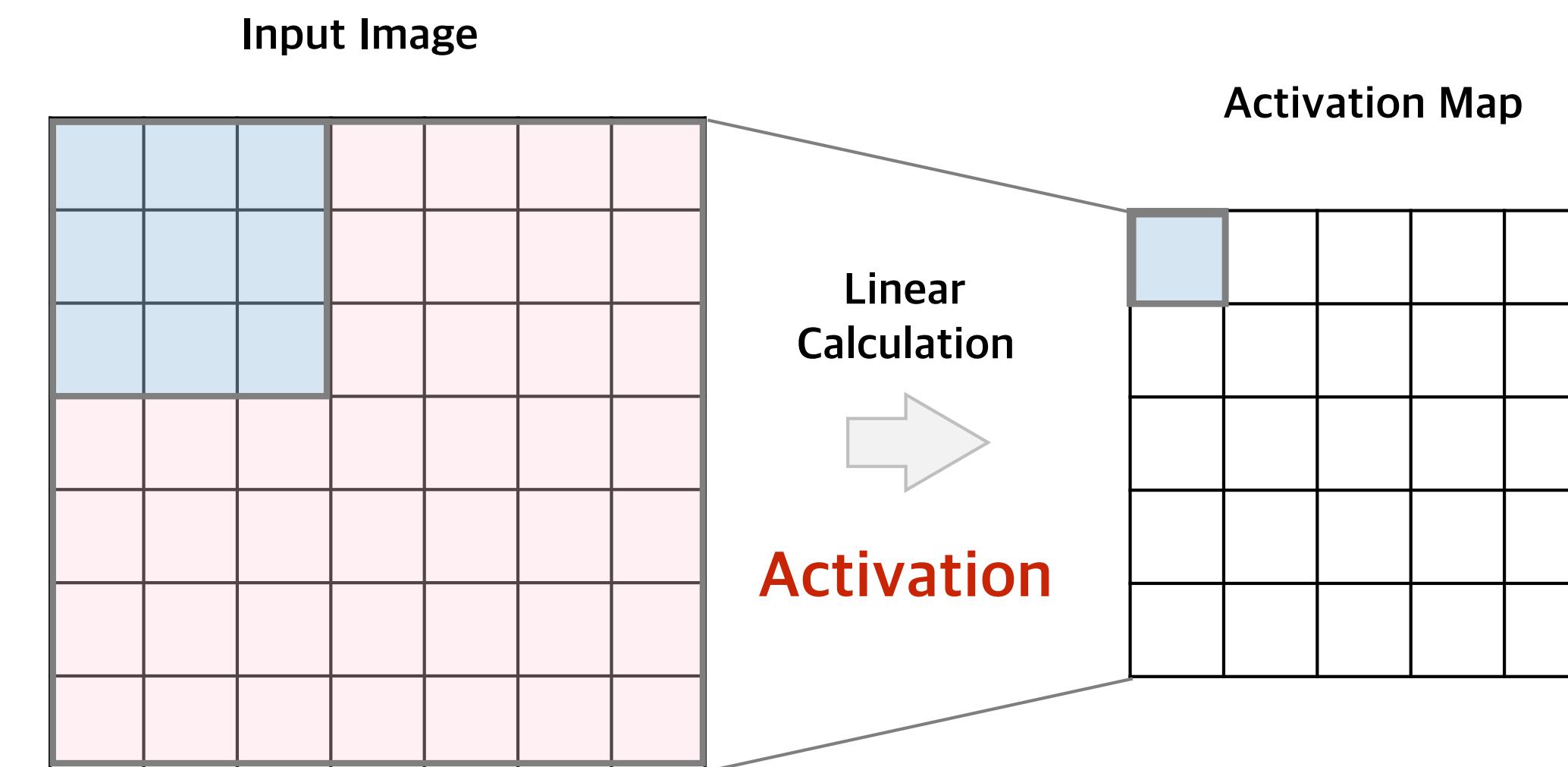
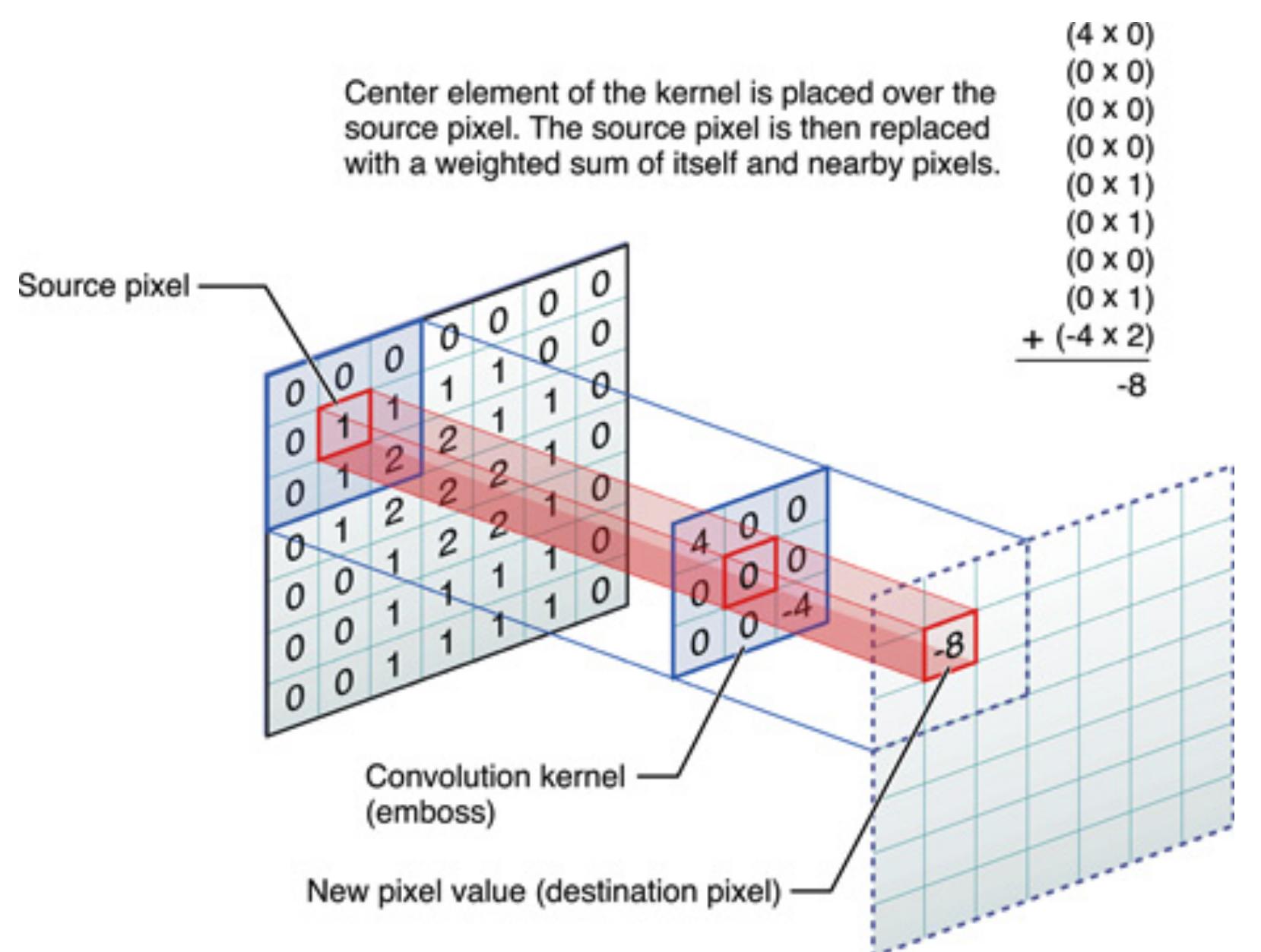
- Receptive field의 픽셀 값들은 위치에 따라 각각의 필터의 값들과 대응됨
- 특정 receptive filed에서의 필터 계산은 대응되는 값을 기준으로 픽셀과 필터 값의 선형 조합(linear combination)을 통해 이루어짐



# Calculating Filters on Image

- 각 필터는 입력 이미지의 픽셀 전체를 차례로 훑고 지나가며 linear combination을 진행하고 Feature Map을 구성함
- 필터의 linear combination 이 후, activation을 통하여 비선형 특징을 포함한 Activation Map으로 재구성

## <Linear Calculation of Filters on Image>



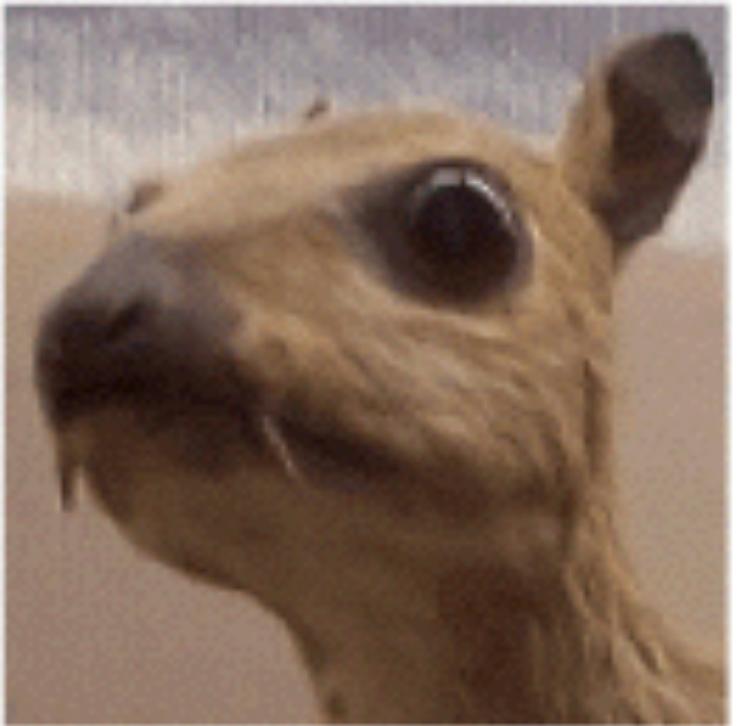
(Recall) In this example, only 9 value of weights are used to make activation map! → Weight Sharing

# Filter Result Example

- ❑ 필터의 계산은 이미지에 존재하는 특정한 특징을 추출하는데 사용됨

<Example of Edge Detection Filter>

Input image



Convolution  
Kernel

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

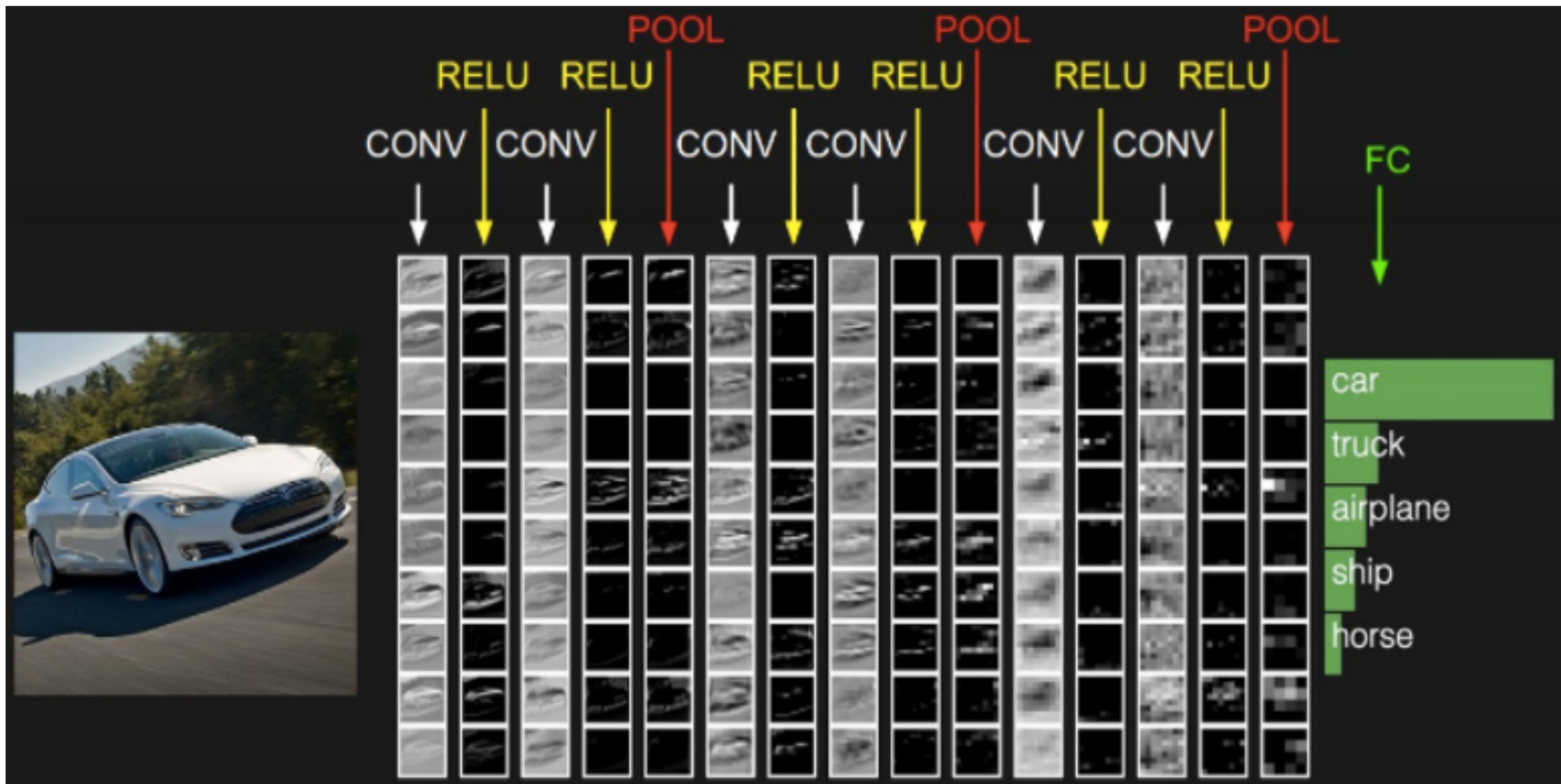
Feature map



But we want to know  
more **complex, nonlinear** relationship  
by learning.

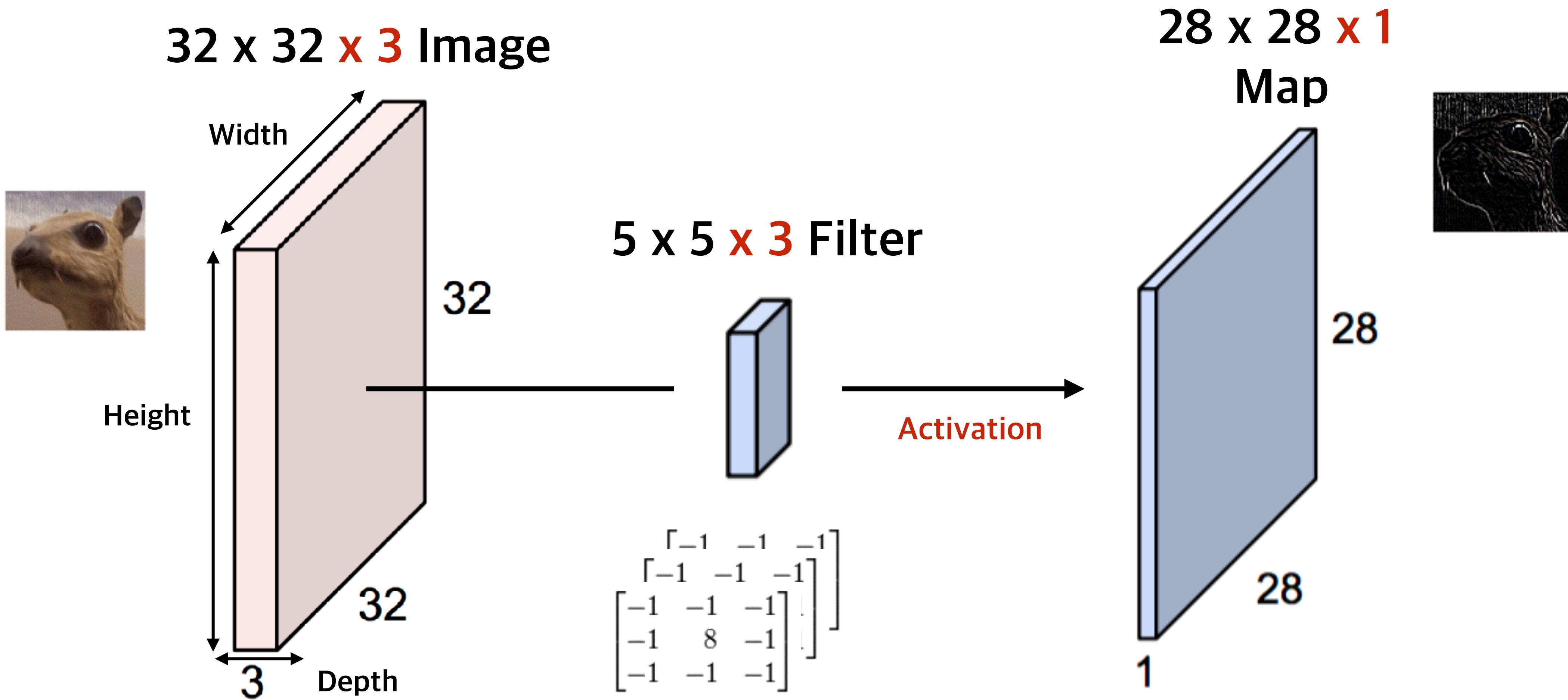


# Filter Result Example



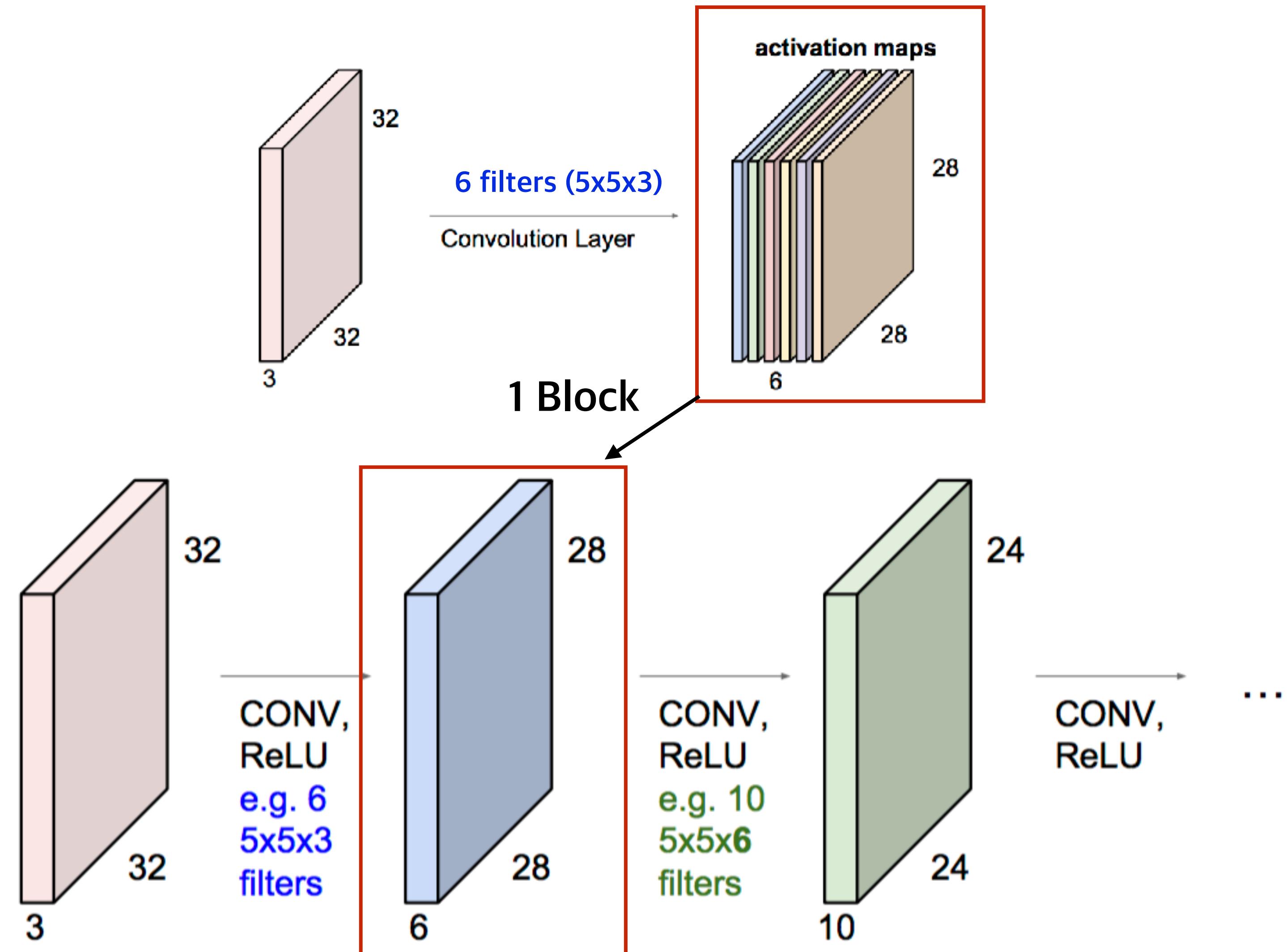
## Depth of Input and Filters

- ❑ 일반적인 이미지의 픽셀은 RGB 컬러를 가지는 3차원 벡터이며 이미지는 Height x Width x Depth 모양의 구조를 가짐
- ❑ 이미지의 Depth 크기와 필터의 Depth 크기는 일치해야함 (RGB 이미지의 경우, depth = 3)



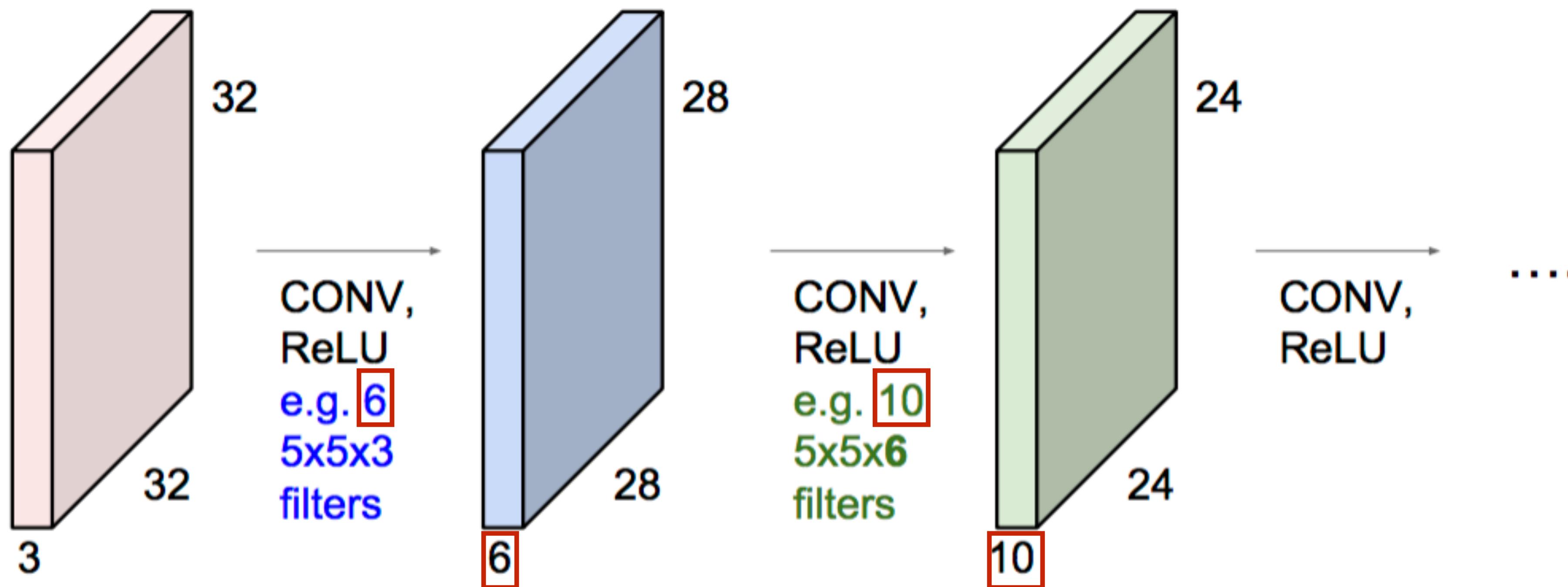
## Depth of Input and Filters

- 다양한 종류의 필터를 통해 여러 개의 activation map을 구성한 후, 이들을 하나의 이미지 블록(block)으로 모음
- 모아진 블록의 depth는 직전 layer에서 사용한 필터의 개수와 동일함



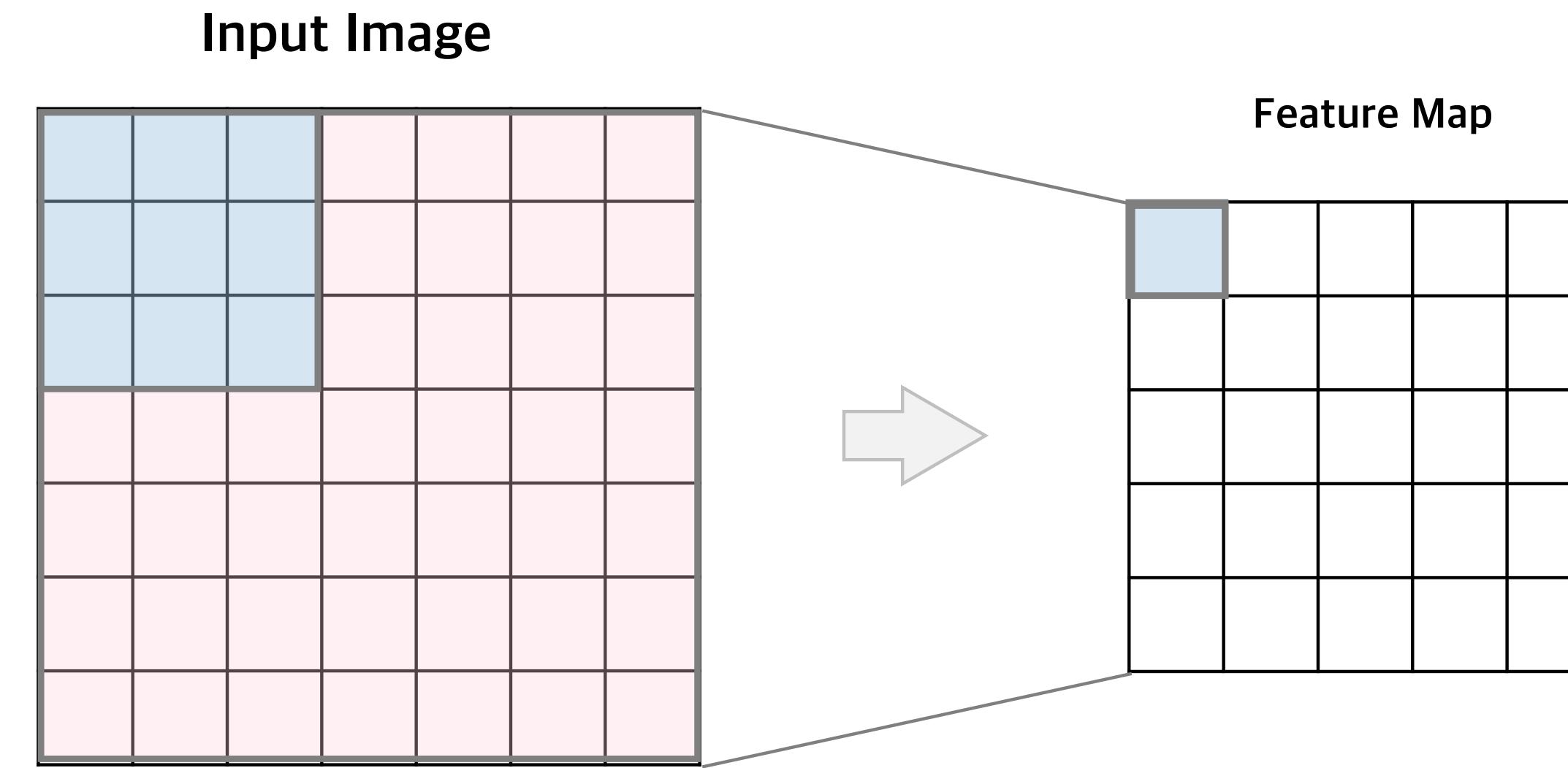
## Depth of Input and Filters

- 다양한 종류의 필터를 통해 여러 개의 activation map을 구성한 후, 이들을 하나의 이미지 블록(block)으로 모음
- 모아진 블록의 depth는 직전 layer에서 사용한 필터의 개수와 동일함



# Calculating Kernels on Image: Linear Combination

- ❑ 필터에 투영되는 receptive field를 얼마나 움직일 지에 따라 feature map 결과의 크기가 결정됨



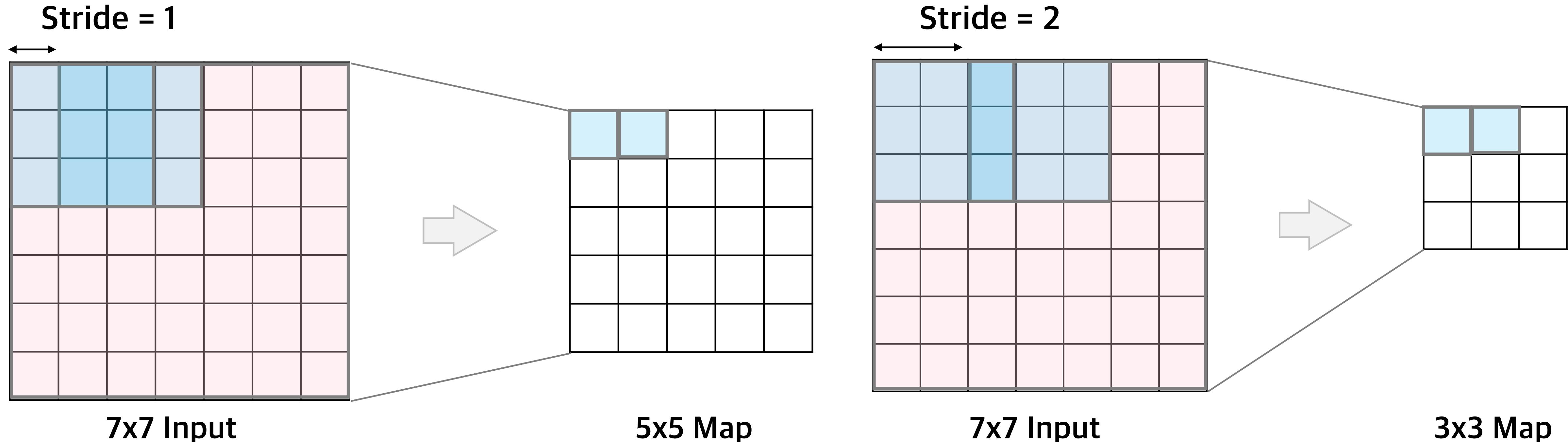
$$\text{one number} = Wx + b$$

Q) 항상 receptive field를 1칸씩 이동해야 하나?

A) No

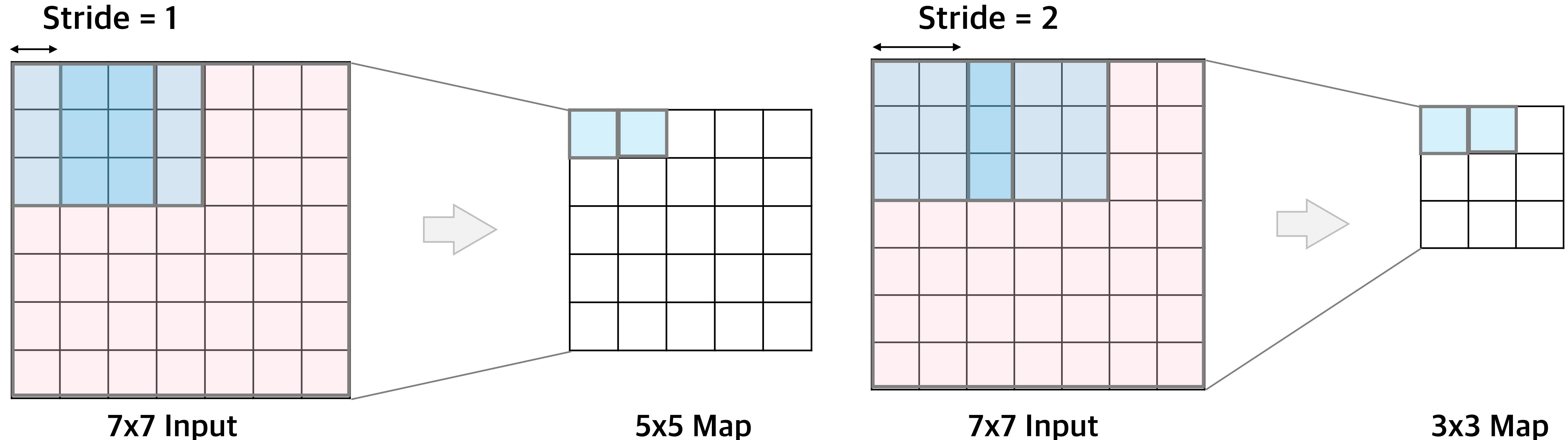
## Stride

- “필터의 계산 과정에서 얼마나 receptive field를 겹치며 이동하는지”에 대한 결정은 결과의 사이즈를 결정함
- Stride: Convolution 계산 과정에서 각 receptive field 사이의 간격 길이



## Stride

- “필터의 계산 과정에서 얼마나 receptive field를 겹치며 이동하는지”에 대한 결정은 결과의 사이즈를 결정함
- Stride: Convolution 계산 과정에서 각 receptive field 사이의 간격 길이



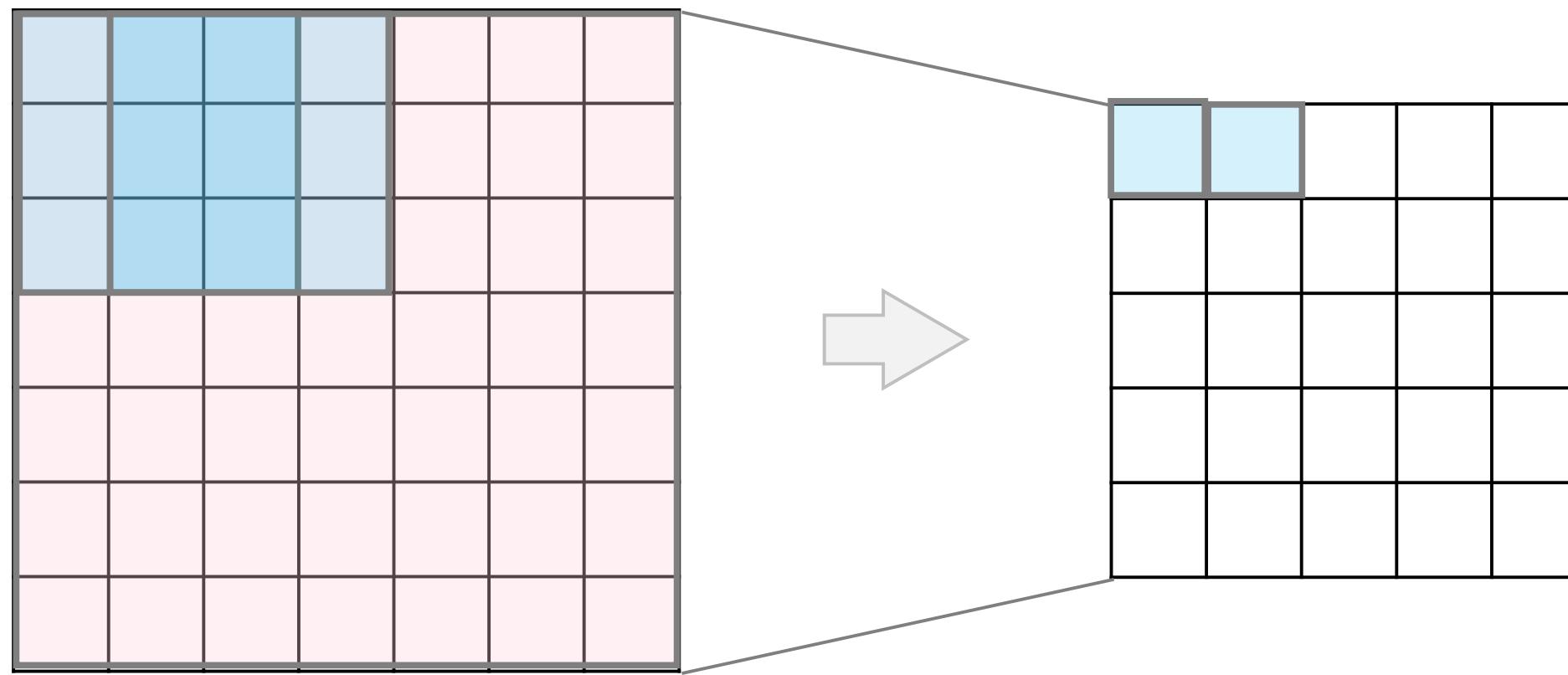
Q) Convolutional layer는 항상 입력보다  
작은 크기의 결과를 도출하는가?  
A) No!

# Padding

❑ **Padding** : 입력 이미지의 주변을 특정한 값으로 추가로 덧붙여 채우는 과정

- 1) Convolutional layer에서 필터 움직임의 stride 크기에 따른 결과의 차원 감소를 방지하기 위해 사용됨 (SAME)
- 2) 설정한 convolutional layer 내 필터의 stride에 따라 이미지 경계 부분의 계산을 진행하기 위해 사용됨 (VALID)

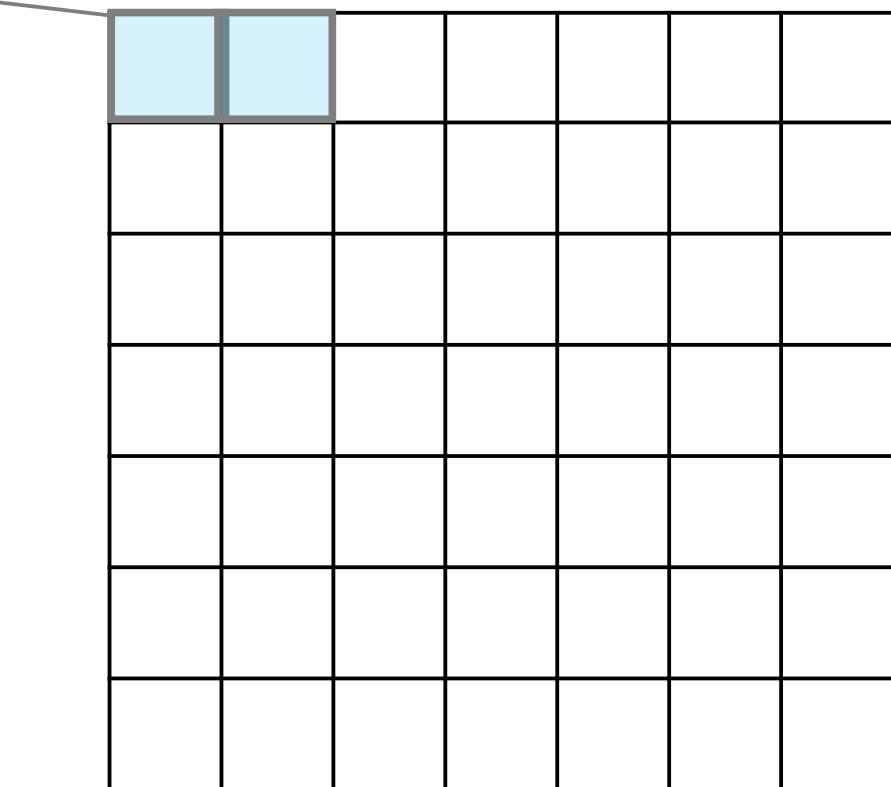
<Without Padding, Stride=1>



<With Zero Padding, Stride=1>

Zero Padding= (Filter Size -1)/2

0	0	0	0	0	0	0	0	0
0								0
0								0
0								0
0								0
0								0
0								0
0	0	0	0	0	0	0	0	0



"Without padding, the size of feature maps become smaller and smaller when the layer is deep, eventually becomes 1x1."

# Summary of Conv Layer Statistics

**Summary.** To summarize, the Conv Layer:

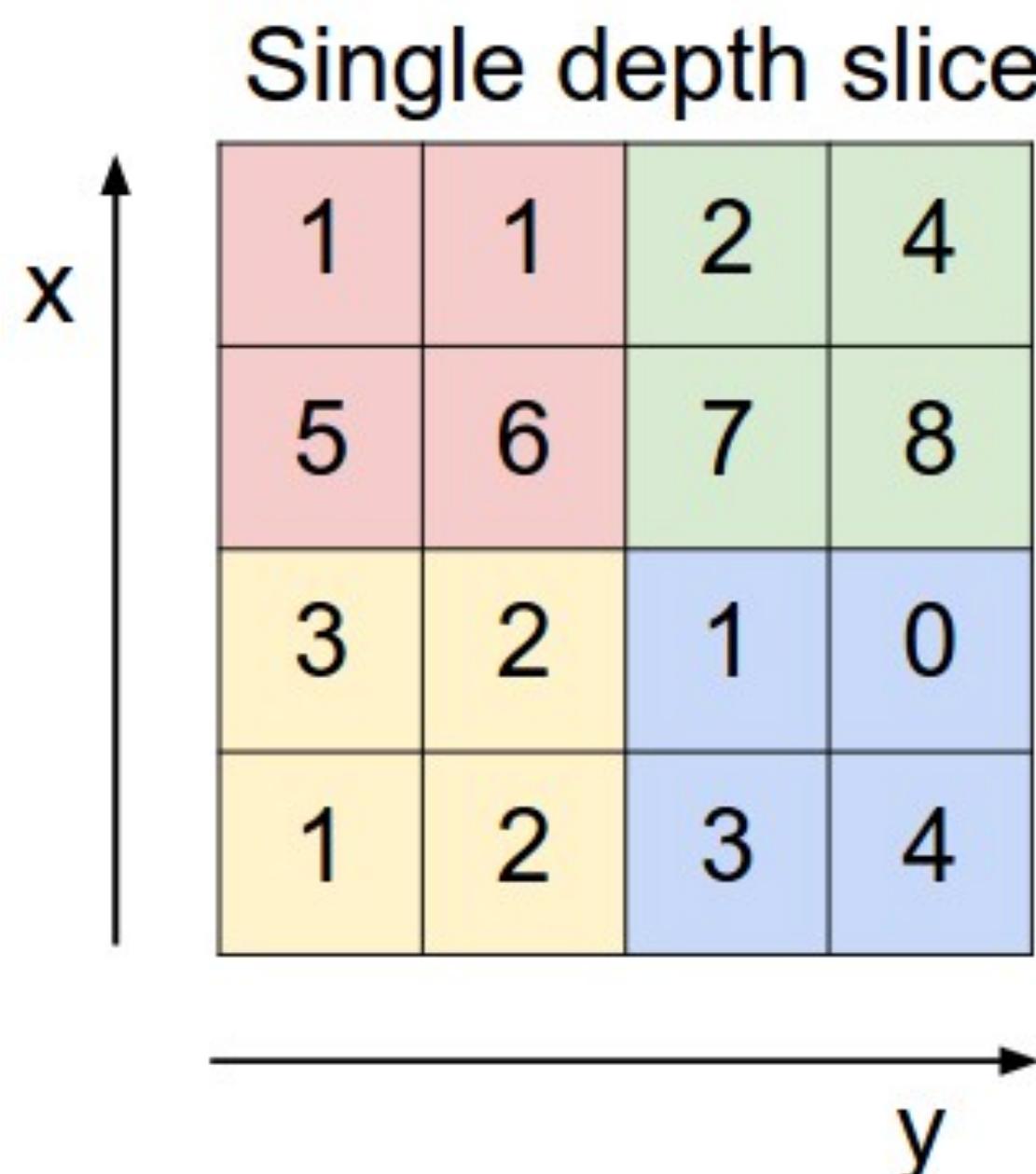
- Accepts a volume of size  $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters  $K$ ,
  - their spatial extent  $F$ ,
  - the stride  $S$ ,
  - the amount of zero padding  $P$ .
- Produces a volume of size  $W_2 \times H_2 \times D_2$  where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$  (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces  $F \cdot F \cdot D_1$  weights per filter, for a total of  $(F \cdot F \cdot D_1) \cdot K$  weights and  $K$  biases.
- In the output volume, the  $d$ -th depth slice (of size  $W_2 \times H_2$ ) is the result of performing a valid convolution of the  $d$ -th filter over the input volume with a stride of  $S$ , and then offset by  $d$ -th bias.

(Causion) They must be **Integer Value**

# Pooling Layer

## ■ Pooling Layer

- Pooling layer는 직전 레이어의 activation map의 크기를 subsampling을 통해 효과적으로 줄임
- 중요한 정보는 유지하면서 입력의 크기를 줄임으로써 과적합(overfitting)을 예방하고 계산량을 감소시키는 효과를 가짐
- Subsampling 방법에 따라 1) Mean(Average) Pooling, 2) Max Pooling, 3) L<sup>p</sup> Pooling 등으로 나뉨



max pool with 2x2 filters  
and stride 2

Max pooling  
Average polling  
L<sup>p</sup> polling

6	8
3	4

<Various Pooling Example>

$$g(x) = \begin{cases} \frac{\sum_{k=1}^m x_k}{m}, & \frac{\partial g}{\partial x} = \frac{1}{m} \\ \max(x), & \frac{\partial g}{\partial x_i} = \begin{cases} 1 & \text{if } x_i = \max(x) \\ 0 & \text{otherwise} \end{cases} \\ \|x\|_p = \left( \sum_{k=1}^m |x_k|^p \right)^{1/p}, & \frac{\partial g}{\partial x_i} = \left( \sum_{k=1}^m |x_k|^p \right)^{1/p-1} |x_i|^{p-1} \\ \text{or any other differentiable } \mathbf{R}^m \rightarrow \mathbf{R} \text{ functions} \end{cases}$$

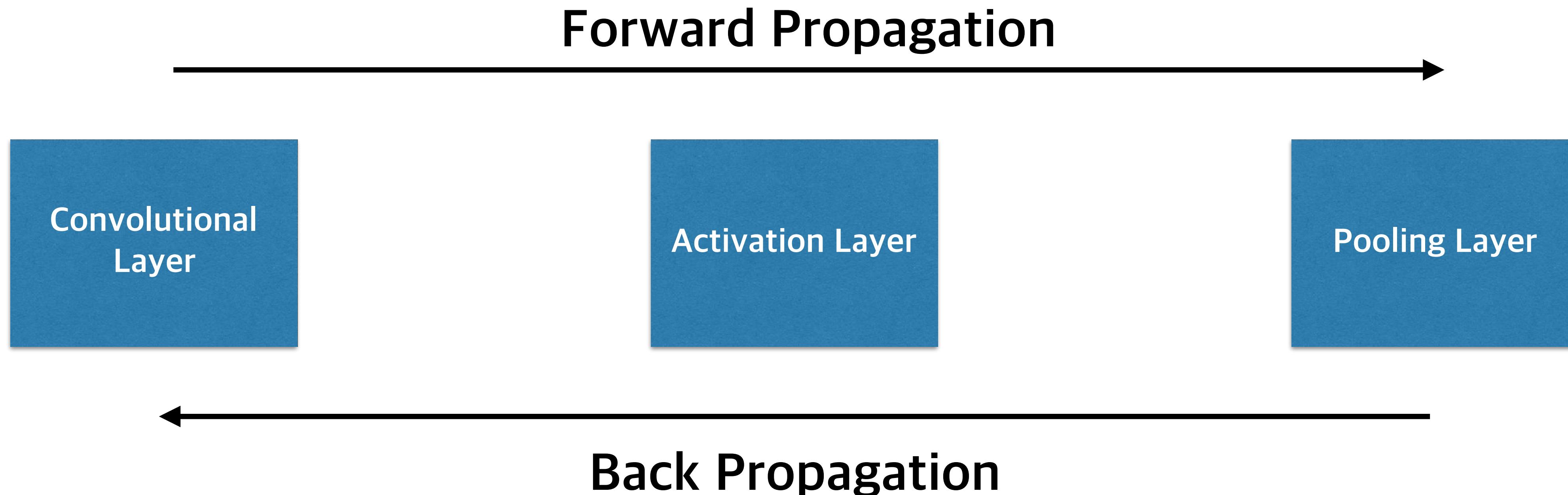
mean pooling

max pooling

L<sup>p</sup> pooling

# Backpropagation in Practice

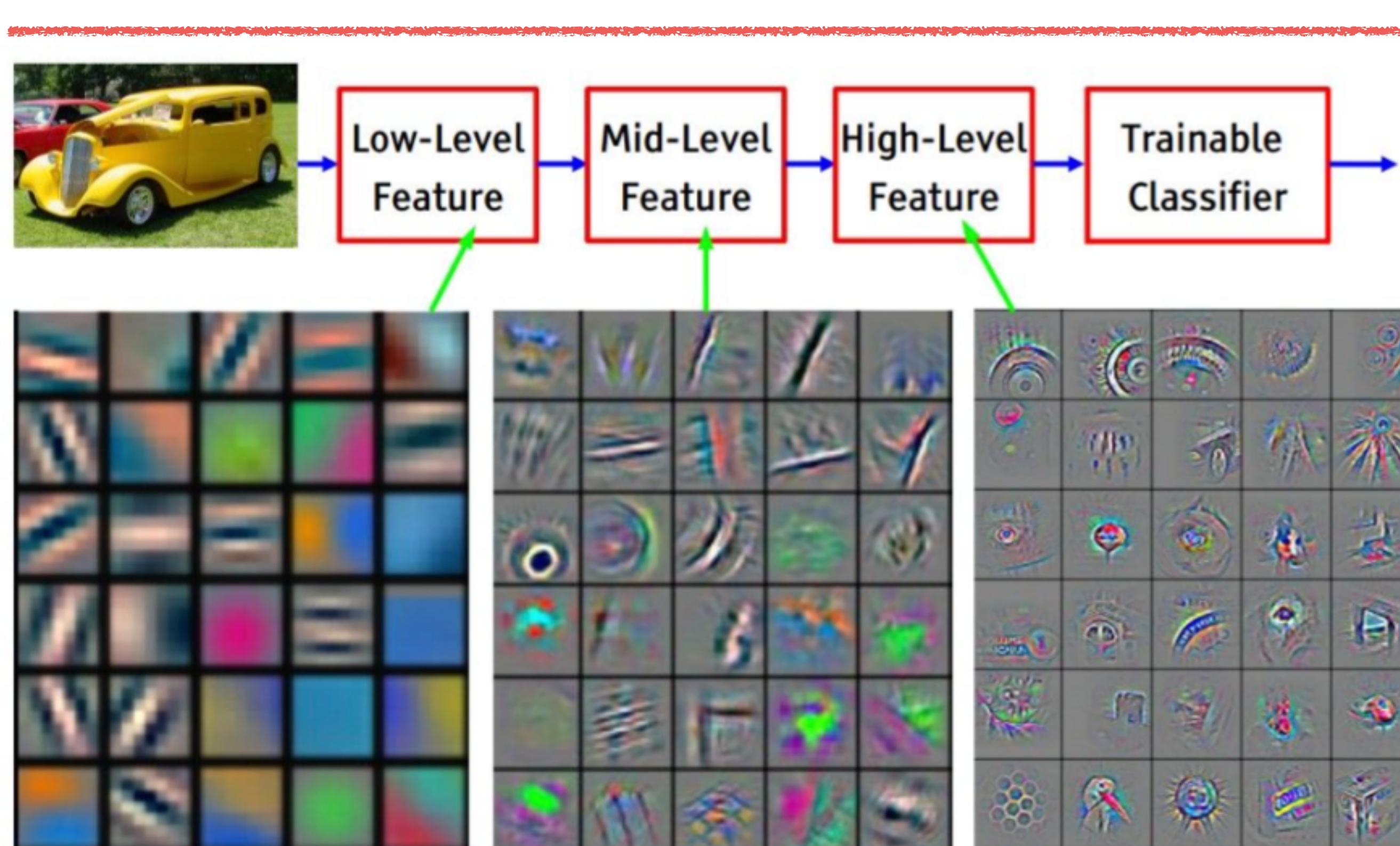
- ❑ 일반적으로 Convolution - Activation - Pooling layer 순서로 모델을 구성함
- ❑ Pooling layer는 딥 러닝 모델의 hidden layer 개수가 증가함에 따라 사용의 빈도가 감소하는 추세임



# Feature Visualization Examples with Conv. Net

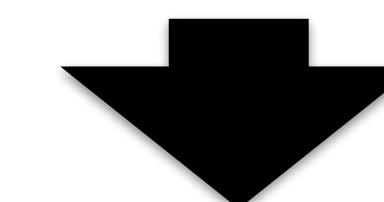
## □ Feature visualization example

### What is the change?



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

- 뒤쪽 Layer로 갈수록 feature를 나타내는 activation map이 sparse image를 포함하는 것을 볼 수 있음
- Sparse activation map을 가지는다는 것은 각각의 map이 고차원의 시각적 특징에 관한 **특정 정보만** 포함하고 있다는 것을 의미
- 해당 시각적 특징(visual feature)를 제외하면, 나머지 값은 모두 0으로 되어있음

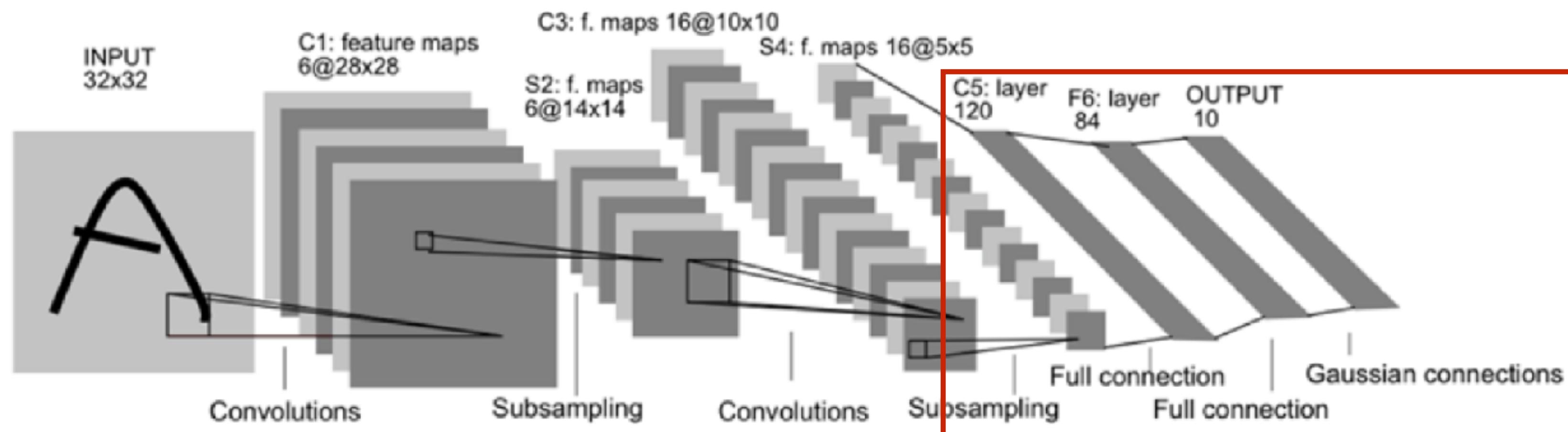


**“Utilize extracted visual features  
in order to ……”**

# Convolutional Neural Network

- Convolutional Neural Network를 통해 시각적 이미지에 포함된 고차원의 숨은 표현(hidden representation)을 추출
- 추출된 정보를 바탕으로 다양한 목적을 달성하기 위해 일반적으로 후반부 layer를 **full connected layer**로 설정함

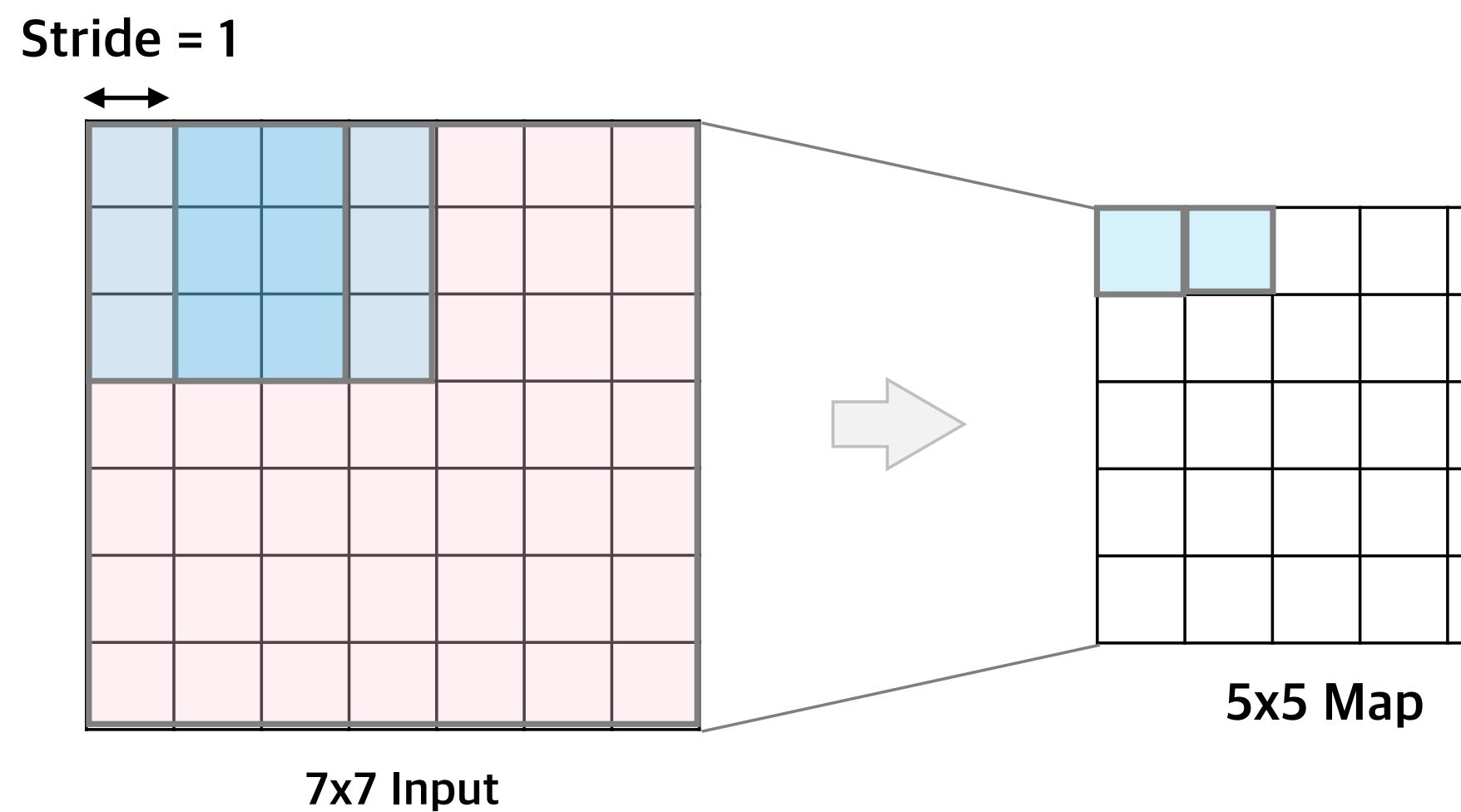
## <Convolutional Neural Network LeNet5>



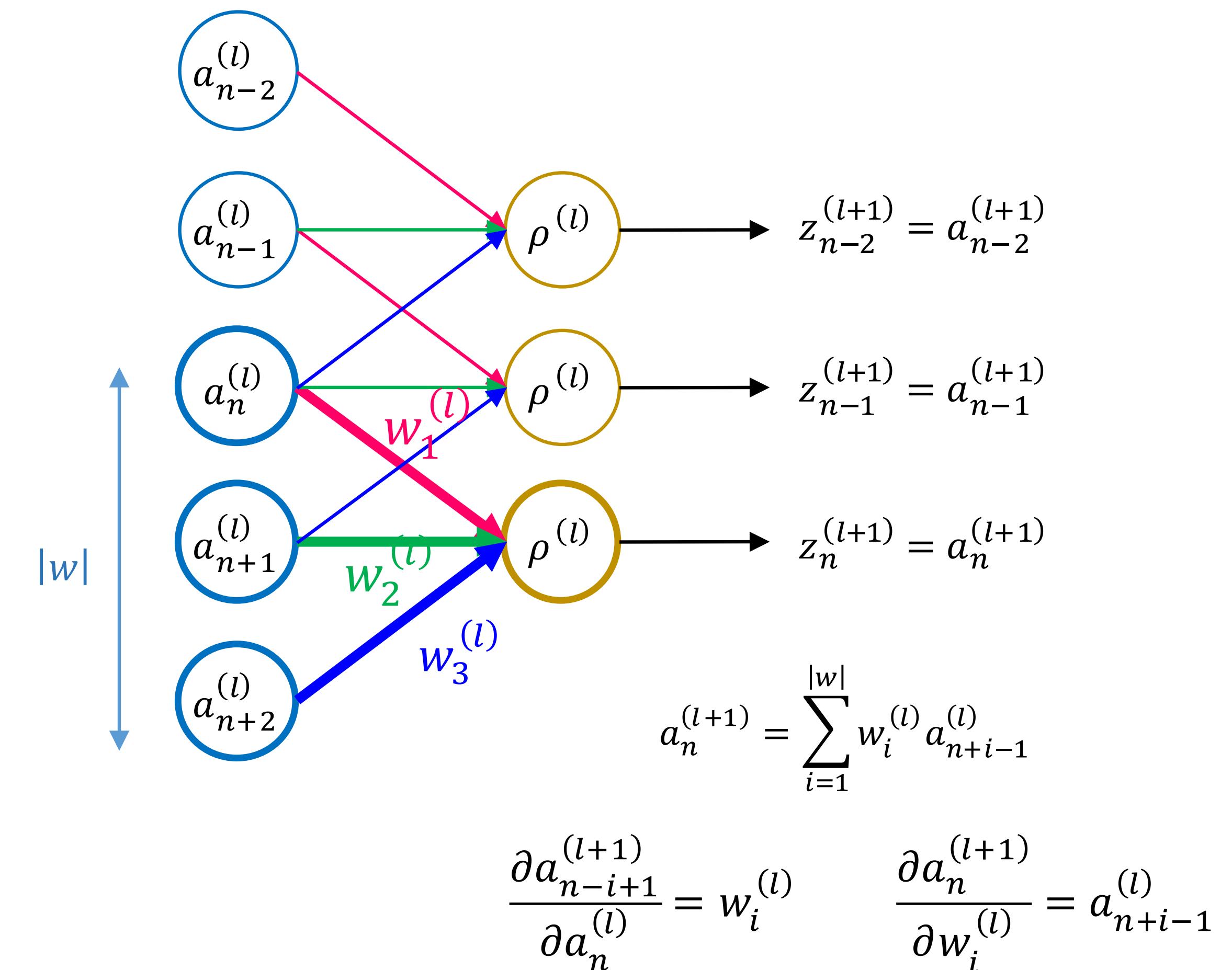
- 3d 구조의 activation map ( $H \times W \times C$ )을 full connected layer의 input으로 사용
- 모든 값들을 일렬로 이어 붙임 (concatenate)

# Trainning Weights in Conv. Net

- ❑ Backpropagation (BP) algorithm also used to obtain the gradient w.r.t each weight.



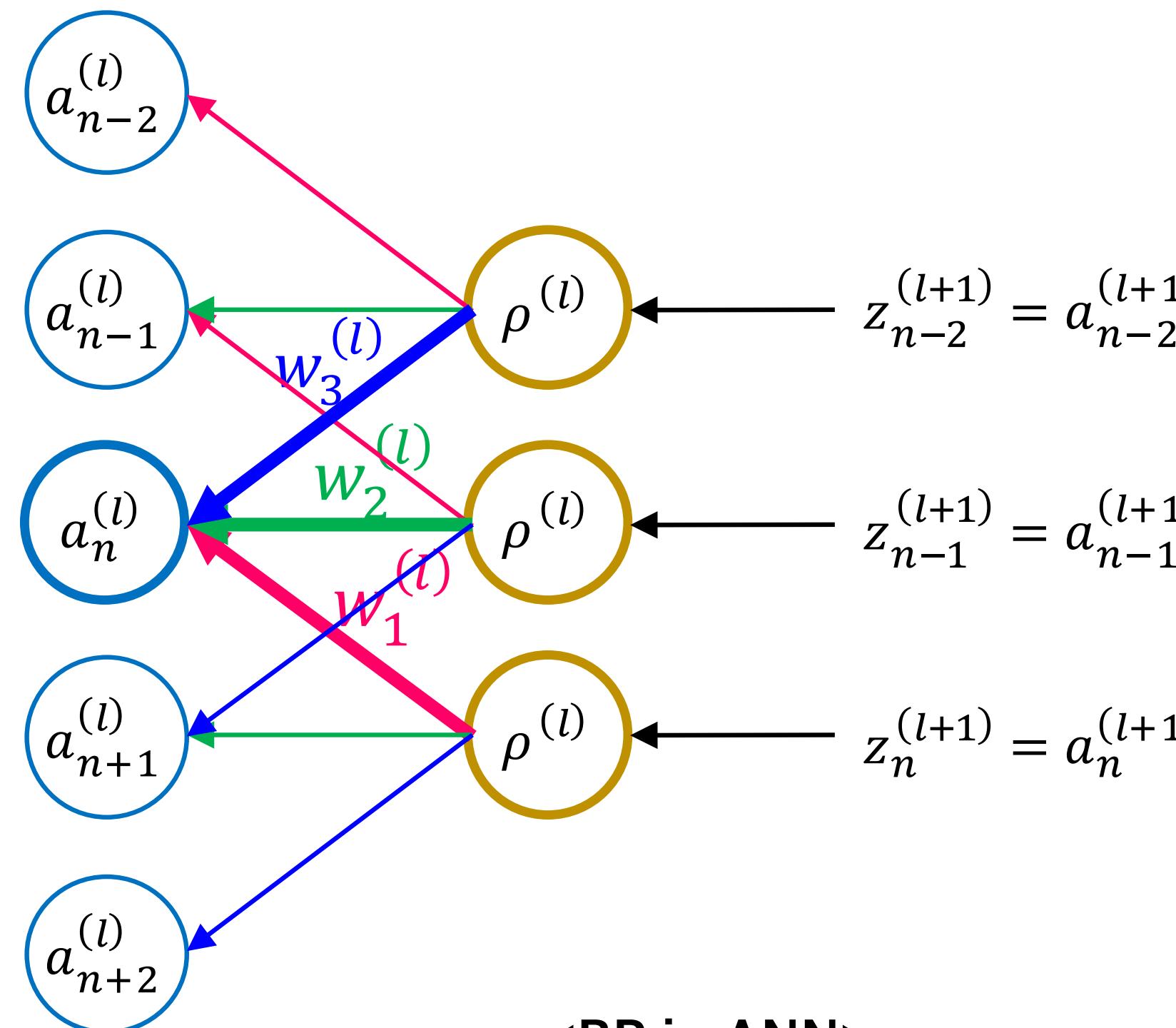
<Forward Propagation of Convolutional Layer>



# Trainning Weights in Conv. Net

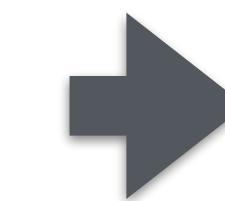
- ❑ Backpropagation (BP) algorithm also used to obtain the gradient w.r.t each weight.

<Backward Propagation of Convolutional Layer>



<BP in ANN>

$$\frac{\partial C}{\partial w_i^{(l)}} = \frac{\partial C}{\partial a_n^{(l+1)}} \frac{\partial a_n^{(l+1)}}{\partial w_i^{(l)}} = \delta_n^{(l+1)} a_{n+i}^{(l)}$$



Define delta

$$\delta_n^{(l+1)} = \frac{\partial C}{\partial z_n^{l+1}} = \frac{\partial C}{\partial a_n^{l+1}}$$

Backward Propagation (similar with ANN case)

$$\frac{\partial C}{\partial a_n^{(l)}} = \sum_{i=1}^{|w|} \frac{\partial C}{\partial a_{n-i+1}^{(l+1)}} \frac{\partial a_{n-i+1}^{(l+1)}}{\partial a_n^{(l)}} = \sum_{i=1}^{|w|} \delta_{n-i+1}^{(l+1)} w_i^{(l)}$$

$$a_n^{(l+1)} = \sum_{i=1}^{|w|} w_i^{(l)} a_{n+i-1}^{(l)}, \quad \frac{\partial a_{n-i+1}^{(l+1)}}{\partial a_n^{(l)}} = w_i^{(l)}$$

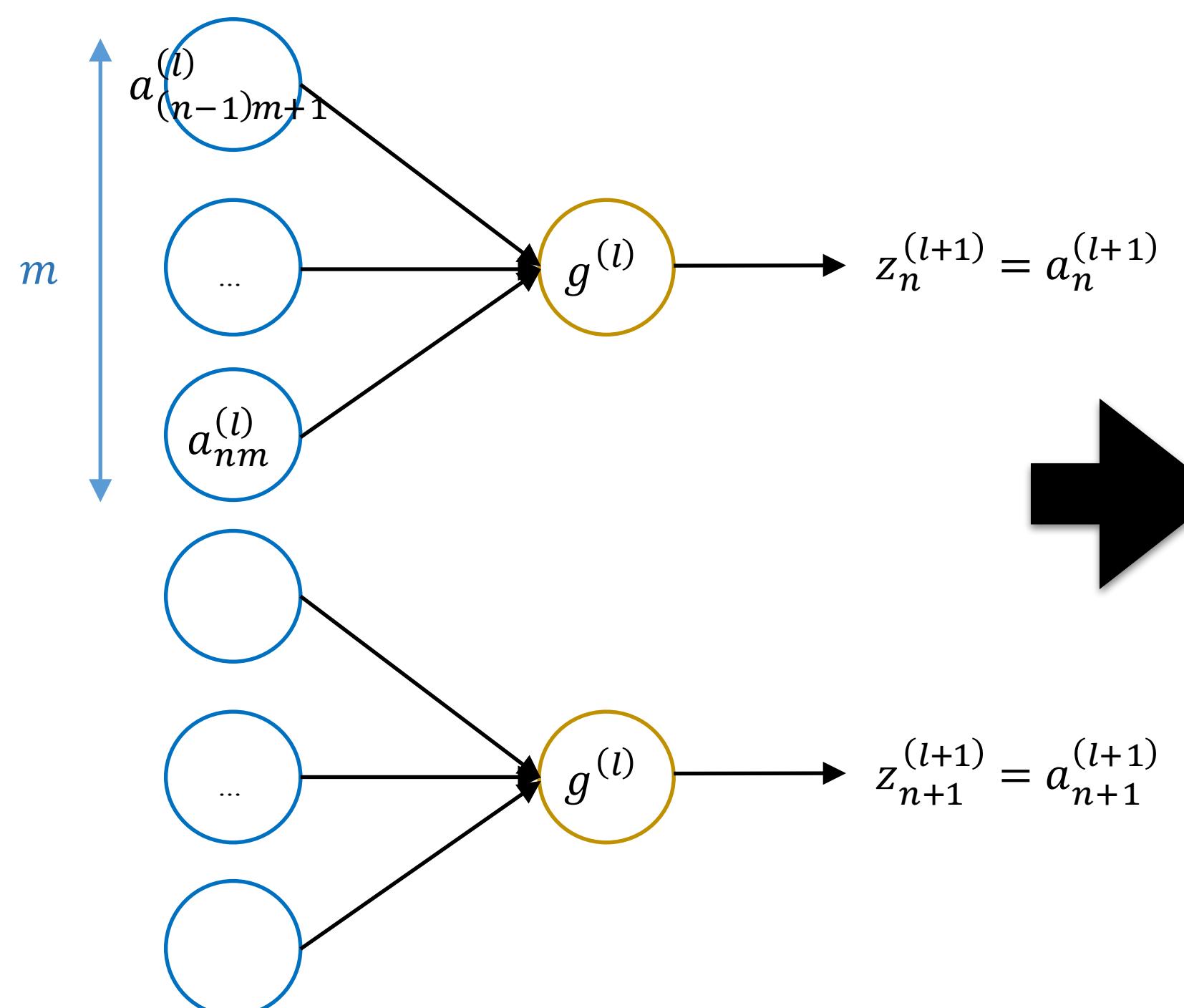
Calculate Weight Gradient

Not a value, but summation!  
(It is **shared weight**)

$$\frac{\partial C}{\partial w_i^{(l)}} = \sum_{n=1}^{N^{(l+1)}} \frac{\partial C}{\partial a_n^{(l+1)}} \frac{\partial a_n^{(l+1)}}{\partial w_i^{(l)}} = \sum_{n=1}^{N^{(l)} - |w| + 1} \delta_n^{(l+1)} a_{n+i-1}^{(l)}$$

# Backpropagation in Pooling Layer

- The types of pooling function determine the result of back propagation in pooling layer.



$$\frac{\partial C}{\partial a_{(n-1)m+1:nm}^{(l)}} = \frac{\partial C}{\partial z_n^{(l+1)}} \frac{\partial z_n^{(l+1)}}{\partial a_{(n-1)m+1:nm}^{(l)}} = \delta_n^{(l+1)} \frac{\partial z_n^{(l+1)}}{\partial a_{(n-1)m+1:nm}^{(l)}} \triangleq \delta_n^{(l+1)} \cdot (g^{(l)})'_{(n-1)m+1:nm}$$

$$g(x) = \begin{cases} \frac{\sum_{k=1}^m x_k}{m}, & \frac{\partial g}{\partial x} = \frac{1}{m} \\ \max(x), & \frac{\partial g}{\partial x_i} = \begin{cases} 1 \text{ if } x_i = \max(x) \\ 0 \text{ otherwise} \end{cases} \\ \|x\|_p = \left( \sum_{k=1}^m |x_k|^p \right)^{1/p}, & \frac{\partial g}{\partial x_i} = \left( \sum_{k=1}^m |x_k|^p \right)^{1/p-1} |x_i|^{p-1} \end{cases}$$

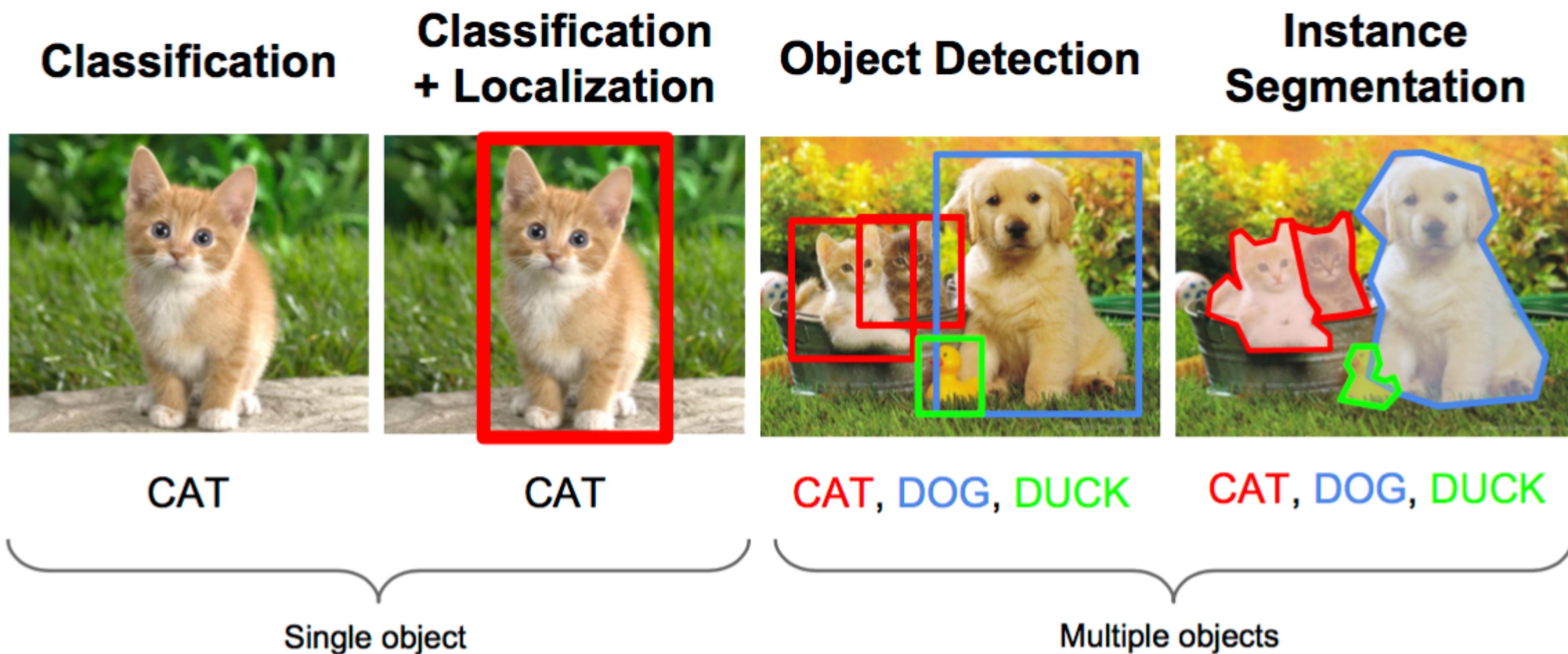
or any other differentiable  $\mathbf{R}^m \rightarrow \mathbf{R}$  functions

mean pooling  
max pooling  
 $L^p$  pooling

# Applications of Conv. Net

# Conv. Net Applications in Vision Area

- ❑ Convolutional neural network can be utilized to various applications in vision area
- ❑ Many architectures were designed and improved its performance.



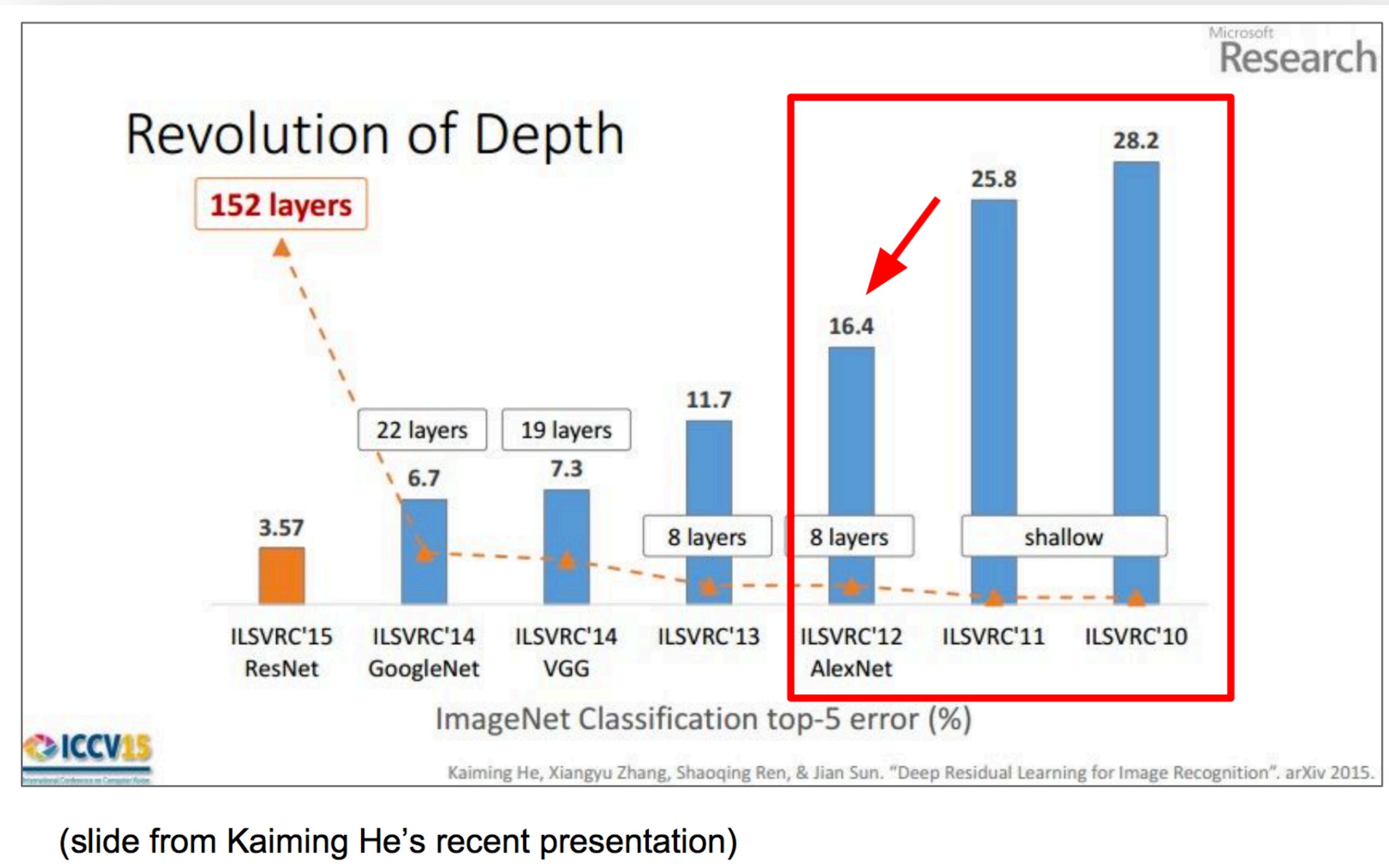
# Applications on Image Classification

- ❑ After AlexNet (2012 Winner), various architectures were designed with Conv. Net.
- ❑ With computational power of GPU, the model tends to be deeper and deeper.

Classification



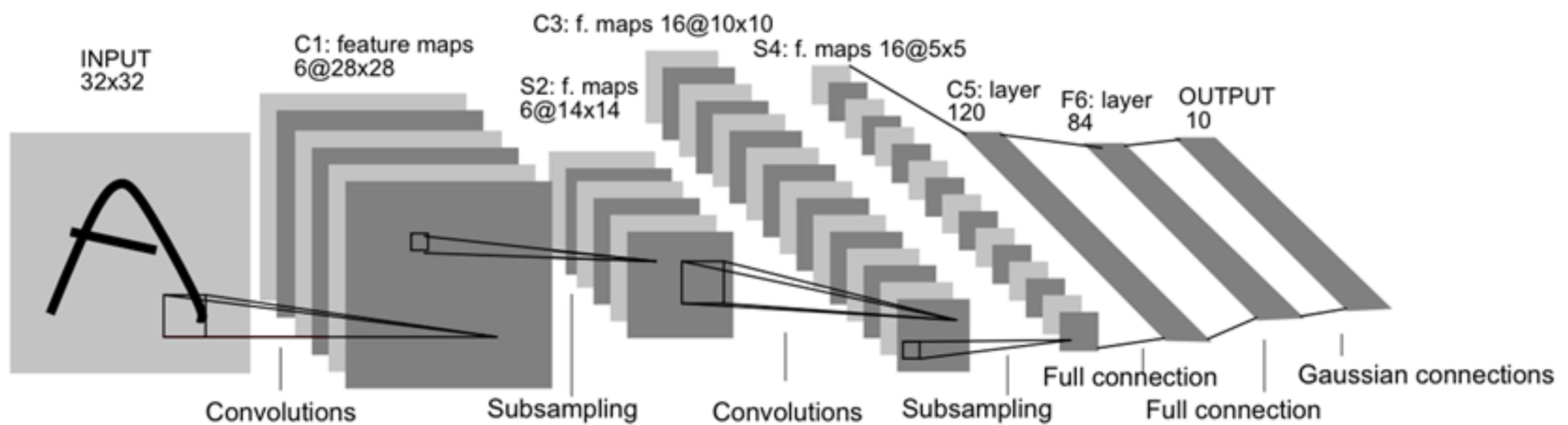
CAT



# LeNet-5

## □ LeNet-5

- Early convolutional neural network architecture (Le Cunn, 1998)
- LeNet-5 was used in order to recognize handwritten digit images.



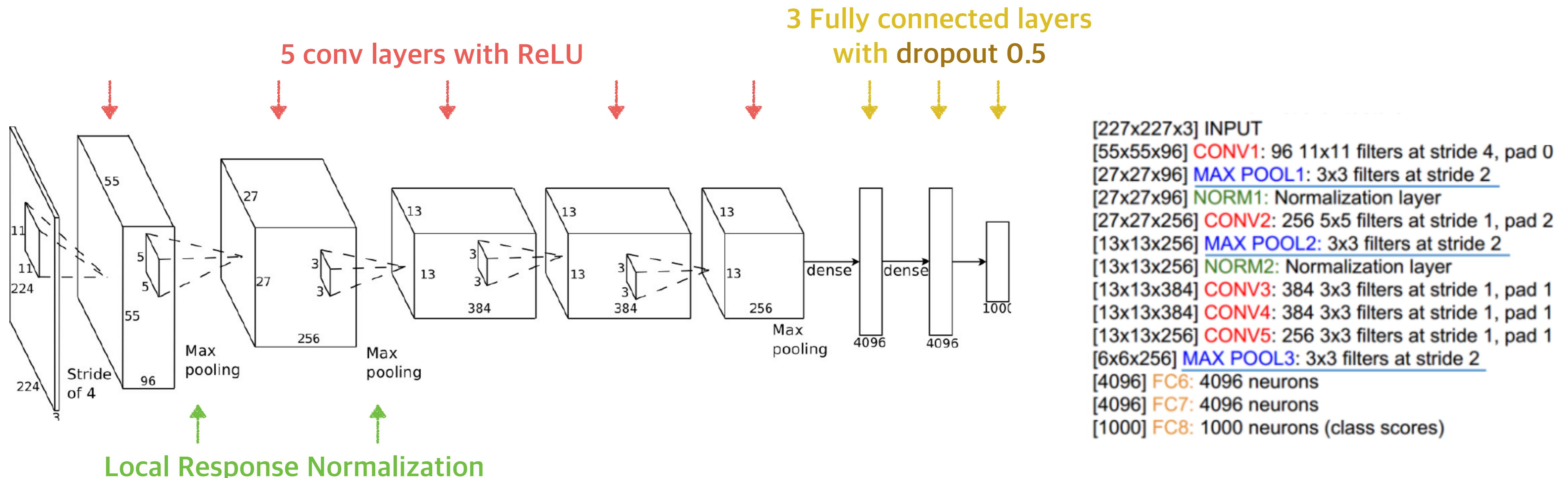
- **Sigmoid Activation**
- **Subsampling means only reduction of size, not max pooling**

An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

# AlexNet (The ILSVRC 2012 Winner)

## ❑ AlexNet (Krizhevsky et al. 2012)

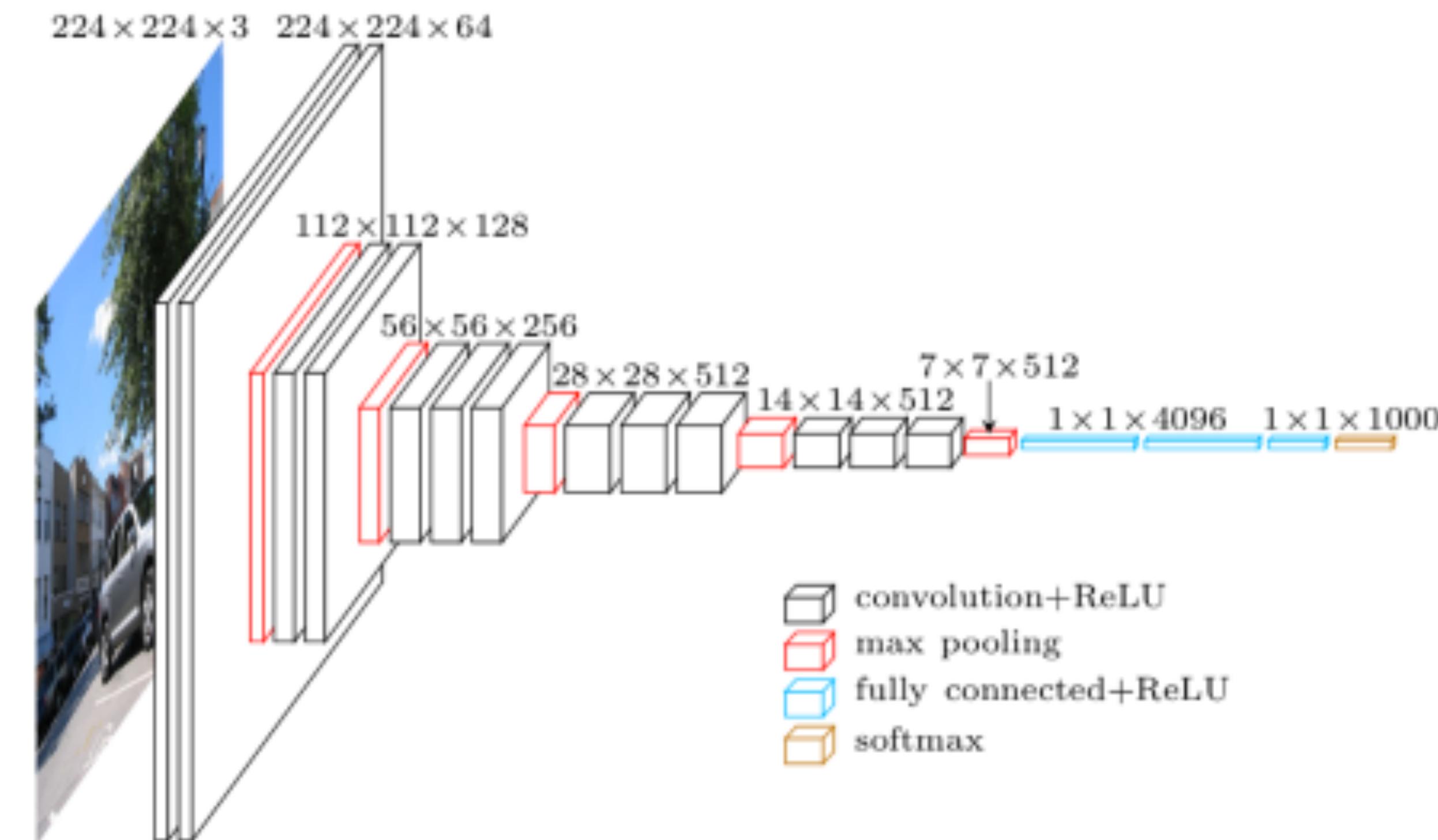
- No difference in the view of network components, but advance of computing technology
- ReLU function is used for activation (first use)
- Local Response Normalization (helps generalization)
- Drop Out (helps generalization)



# VGG (The ILSVRC 2014 2nd Winner)

## ❑ Very Deep CNN (Simonyan et al., 2014) or VGG (Visual Geometry Group)

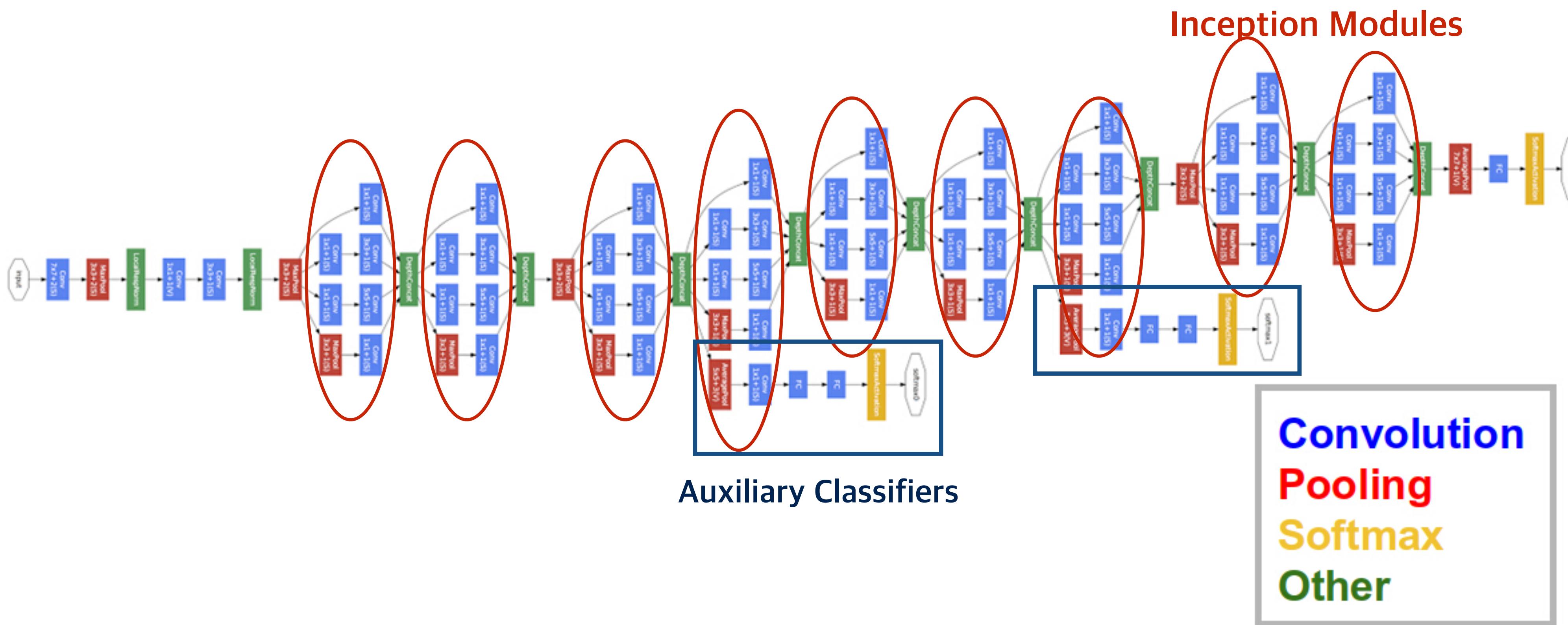
- VGG는 작은 크기의 convolutional filter( $3 \times 3$ , stride=1)를 사용하는 대신, 매우 깊게 모델을 구조화함  
(VGG: 16 or 19 layers, AlexNet: 8 layers)
- 3-4 개의 convolutional layers를 연속으로 위치시킨 후, pooling layer를 위치시킴
- 다른 모델에 비해 **매우 간단한 구조**로 이해 및 구현하기 쉽고 성능이 높아 대중적으로 많이 사용됨



# GoogLeNet” (The ILSVRC 2014 Winner)

## □ GoogLeNet (called “Inception”, Szegedy et al., 2015)

- 22개 layer 구조로 ILSVRC 2014 우승 모델
- Layer by Layer 구조의 일반화로 네트워크 내 네트워크를 배치하는 “Inception” 모듈을 사용
- 깊은 구조의 학습을 위해 Auxiliary Classifiers & Loss (보조적인 분류기와 손실)를 사용

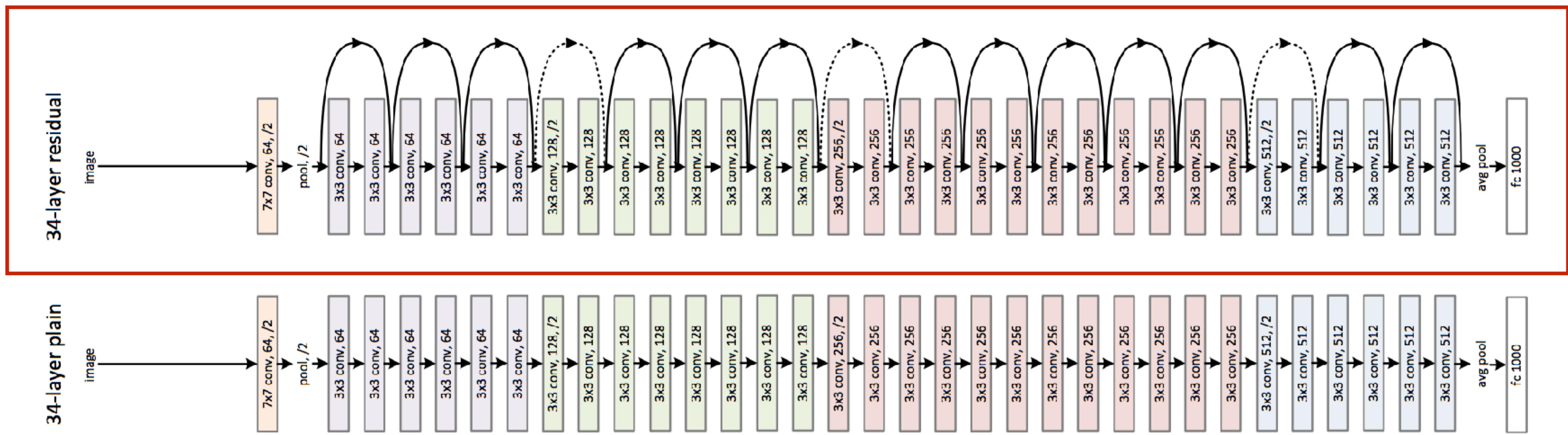


# ResNet (The ILSVRC 2015 Winner)

## □ ResNet (Deep Residual Network, Kaiming et al., MS Research)

- ResNet은 기존 모델들과 다르게 매우 많은 hidden layer를 포함하는 모델 (152 layers)
- “Skip connection” 구조를 통하여 기존 feature map이 아닌 입력값과 사이의 **잔차(residual)**를 학습하는 방식으로 구현
- Skip connection을 통해 매우 많은 hidden layer를 사용하더라도, gradient vanishing 현상을 극복하고 학습이 가능

<ResNet and Plain Architectures>



# DenseNet (The ILSVRC 2015 Winner)

## □ Densely Connected Convolutional Networks (2017, Gao et. al.)

- 기존 Skip Connection을 모든 Layer에 연결하는 것으로 확장하여 여러 레벨의 feature를 종합적으로 활용
- 이전 Layer의 Feature Map을 이후 모든 Layer의 Feature Map에 “Concatenate”하여 feature map을 구성

### <DenseNet Architectures>

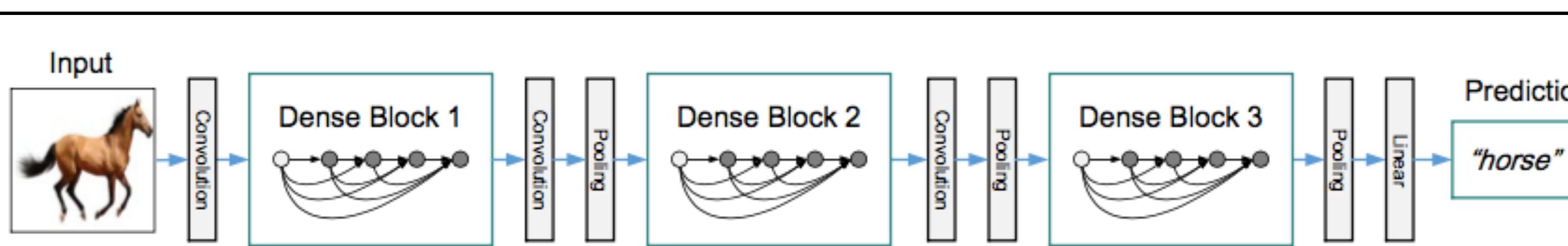


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

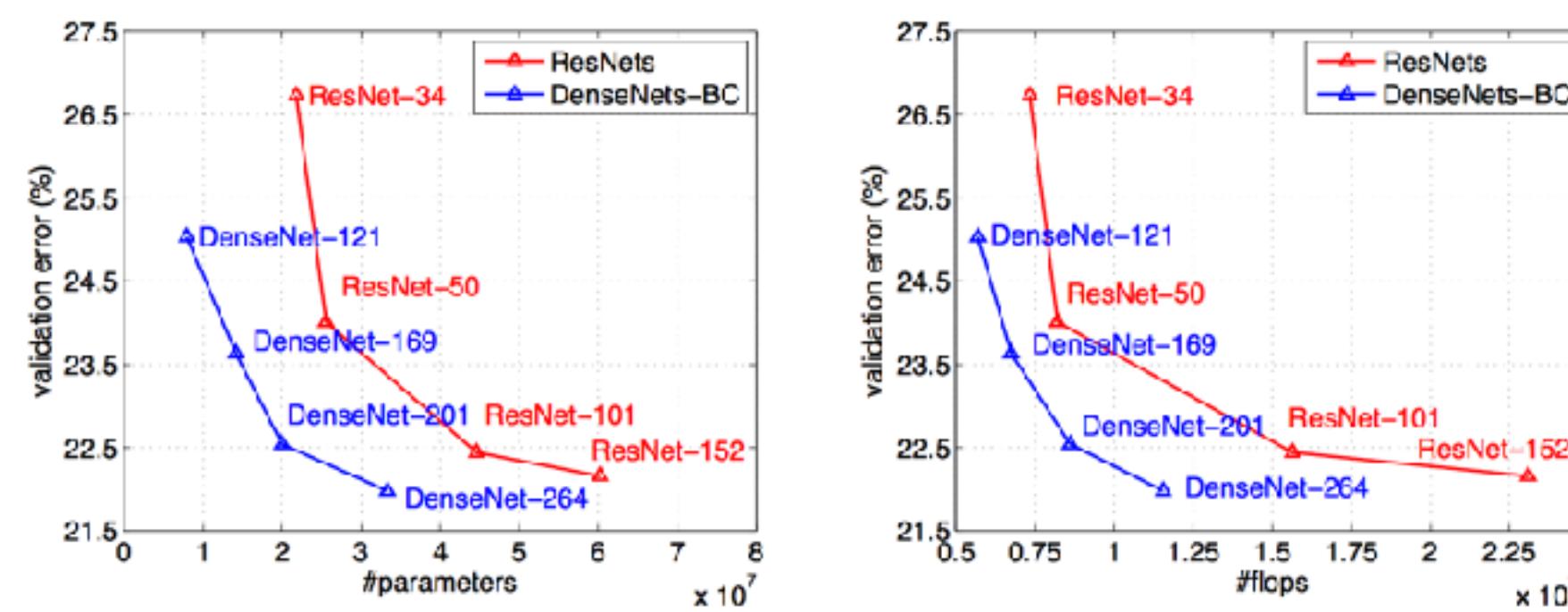


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (left) and FLOPs during test-time (right).

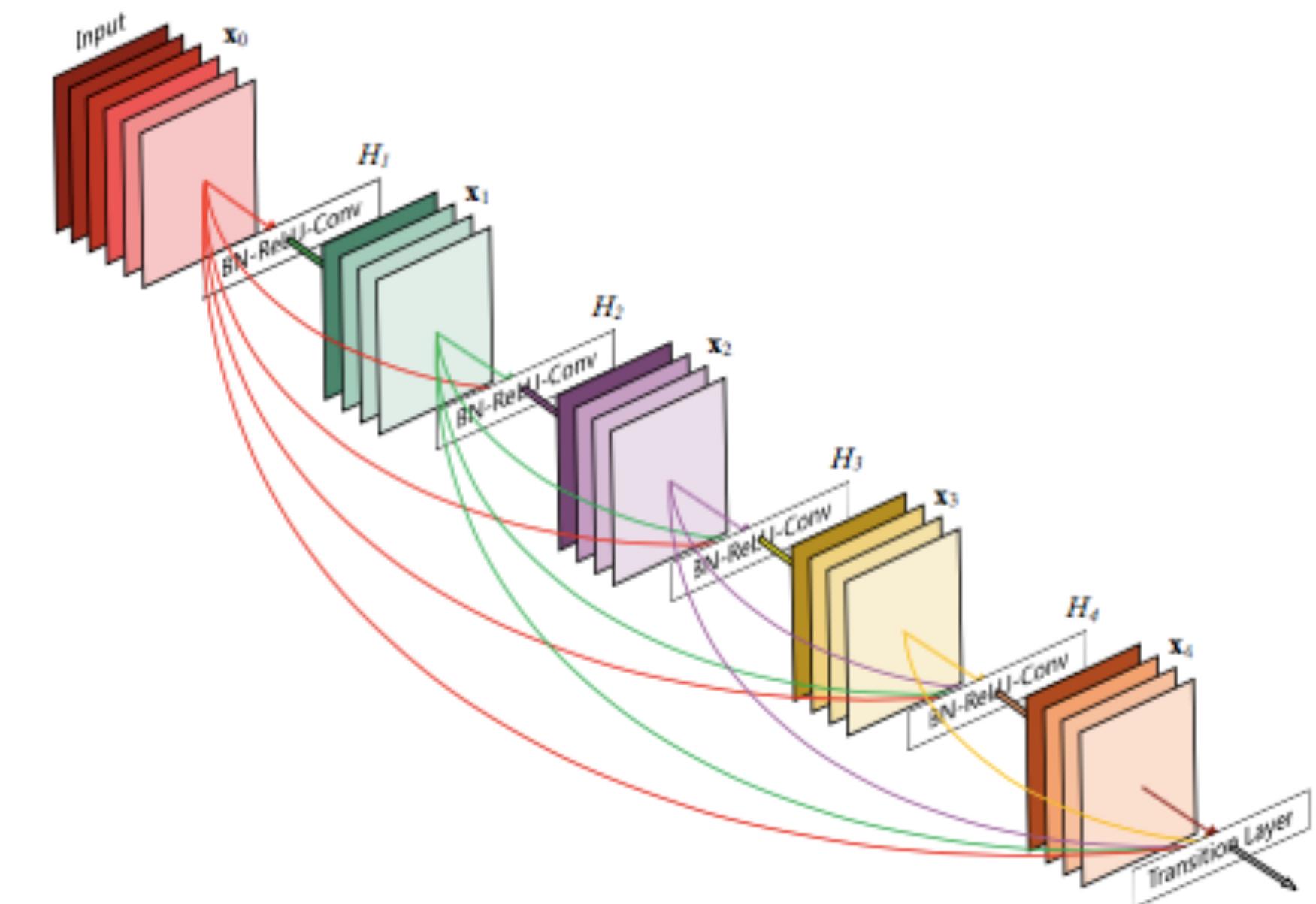


Figure 1: A 5-layer dense block with a growth rate of  $k = 4$ . Each layer takes all preceding feature-maps as input.

# ConvNet in Tensorflow

# Convnet in Tensorflow

- Tensorflow를 이용하여 convnet의 layer들을 쉽게 구현할 수 있음.
  - tf.nn.conv2d: Convolutional layer를 구현
  - tf.nn.max\_pool(tf.nn.avg\_pool): Max(or average) pooling layer를 구현.
  - tf.nn.relu(tf.sigmoid, tf.tanh): relu (sigmoid, tanh) activation을 구현.

# tf.nn.conv2d

- tf.nn.conv2d: Convolutional layer를 구현하는 명령어.

```
tf.nn.conv2d(input, filter, strides, padding, use_cudnn_on_gpu=None, data_format=None, name=None)
```

- **input**: 처음 투입되는 자료로 [batch size, input height, input width, input channel]의 4차원의 shape으로 입력받음
  - 예) [batch size=128, input height=28, input width=28, input channel=1]: 배치 크기 128, 높이 28 pixel, 너비 28 pixel, 채널 1(흑백)).  
컬러 자료의 경우 input channel=3 (R,G,B).
- **filter**: filter(kernel)의 크기를 결정하며 [filter height, filter width, input channel, output channel]의 형식을 가짐.
  - 예) [5,5,1,32]: 너비, 높이가 5인 kernel이며 input channel이 1인 자료를 32의 채널을 가진 자료로 변환.
- **strides**: stride의 크기를 결정하며 Input data의 각 dimension별로 각각 설정.
  - 예) [1,1,1,1]: 각 dimension 별로 1의 크기로 kernel을 움직임 (stride=1).
  - 첫 번째와 마지막 차원의 값은 (거의) 항상 1로 설정(why?)
- **padding**: padding의 알고리즘을 결정하며 'SAME', 'VALID'의 두 가지로 나뉨
  - SAME: 입력과 출력의 width, height 크기가 같도록 PADDING
  - VALID: 특정 stride 크기에 대해 마지막까지 연산이 가능하도록 PADDING
- **data\_format**: Input, output data의 형식을 결정하며, 'NHWC', 'NCHW'의 형식 중 하나 (기본값은 'NHWC').
  - 'NHWC': [batch size, input height, input width, input channel], 'NCHW': [batch size, input channels, input heights, input width].
- **name**: 해당 operation의 이름 설정(선택사항).

## tf.nn.max\_pool

- tf.nn.max\_pool(tf.nn.avg\_pool): Pooling layer를 구현하는 명령어.

```
tf.nn.max_pool(value, ksize, strides, padding, data_format="NHWC", name=None)
tf.nn.avg_pool(value, ksize, strides, padding, data_format="NHWC", name=None)
```

- **value**: [batch size, height, width, channels]의 형식을 가진 자료.
  - convolutional layer의 뒤에서 사용될 경우, conv layer의 결과물을 value로 입력.
- **ksize**: pooling layer의 창의 크기를 설정하며, 자료형식은 value 항목과 동일.
  - 예) ksize=[1,2,2,1]: height 2, width 2인 창 형성.
- **strides**: stride의 크기를 결정하며 Input data의 각 dimension별로 각각 설정.
  - 예) [1,2,2,1]: 각 dimension에 해당하는 크기로 pooling layer의 창을 움직임 (stride=2).
- **padding**: padding의 알고리즘을 결정하며 'SAME', 'VALID'의 두 가지로 나뉨 (VALID: zero padding).
- **data\_format**: Input, output data의 형식을 결정하며, 'NHWC', 'NCHW'의 형식 중 하나 (기본값은 'NHWC').
  - 'NHWC': [batch size, input height, input width, input channel], 'NCHW': [batch size, input channels, input heights, input width].
- **name**: 해당 operation의 이름 설정(선택사항).

# FC layer and activation

## □ Full-Connected Layer

- 사용하고자 하는 activation 함수와 matmul을 통해 직접 계산 또는 `tf.contrib.layers.fully_connected(layer, output_neuron, activation)` 을 통해 계산
- FC layer의 계산 전, 2차원 구조로 되어있는 결과들을 **1차원으로 flatten** 시켜야함!

```
with tf.variable_scope('fc') as scope:  
    input_features = height * width * channel  
    pool = tf.reshape(pool, [-1, input_features])  
  
    w = tf.get_variable('weights', [input_features, 1024],  
                        initializer=tf.truncated_normal_initializer())  
    b = tf.get_variable('biases', [1024],  
                        initializer=tf.random_normal_initializer())  
  
    fc = tf.nn.relu(tf.matmul(pool2, w) + b, name='relu')
```

- Pooling layer와 연결되어있는, 1024개의 node를 가지고 있는 FC layer 생성.
    - 기존 4차원 자료를 2차원으로 변화.
    - Pooling layer와 연결되는 weight, bias 설정.
    - Relu 함수를 이용하여 activation.
- 이외에도 sigmoid(`tf.sigmoid`), tanh(`tf.tanh`) 등의 함수를 사용.

```
with tf.variable_scope('softmax') as scope:  
    w = tf.get_variable('weights', [1024, N_CLASSES],  
                        initializer=tf.truncated_normal_initializer())  
    b = tf.get_variable('biases', [N_CLASSES],  
                        initializer=tf.random_normal_initializer())  
    logits = tf.matmul(fc, w) + b  
    preds = tf.nn.softmax(logits)
```

- FC layer의 결과를 이용하여 자료의 class를 구할 때 사용 가능한 코드.
- Softmax 함수(`tf.nn.softmax`)를 이용하여 해당 class에 속할 확률 계산.
- 1024개의 node를 가진 FC layer를 softmax layer와 연결.

## Drawback of tf.Variable

### □ tf.Variable만을 이용할 경우의 단점:

- 많은 수의 변수를 직접 정의해야함: 모델이 더 복잡해질 경우, 관리가 힘듦.
- 변수의 재사용, 공유 불가: 같은 함수를 사용하여도 다른 결과가 도출됨.

```
In [1]: import tensorflow as tf  
import numpy as np
```

```
In [2]: def variable_test(input):  
    var1 = tf.Variable(1)  
    print var1
```

```
In [3]: variable_test(1)  
variable_test(2)
```

```
Tensor("Variable/read:0", shape=(), dtype=int32)  
Tensor("Variable_1/read:0", shape=(), dtype=int32)
```

→ 같은 Operation이지만 함수를 실행할 때마다 변수를 새로 생성

# Drawback of tf.Variable

## □ tf.Variable만을 이용할 경우의 단점:

- 많은 수의 변수를 직접 정의해야함: 모델이 더 복잡해질 경우, 관리가 힘듦.
- 변수의 재사용, 공유 불가: 같은 함수를 사용하여도 다른 결과가 도출됨.

```
def my_image_filter(input_images):  
    conv1_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),  
        name="conv1_weights")  
    conv1_biases = tf.Variable(tf.zeros([32]), name="conv1_biases")  
    conv1 = tf.nn.conv2d(input_images, conv1_weights,  
        strides=[1, 1, 1, 1], padding='SAME')  
    relu1 = tf.nn.relu(conv1 + conv1_biases)  
  
    conv2_weights = tf.Variable(tf.random_normal([5, 5, 32, 32]),  
        name="conv2_weights")  
    conv2_biases = tf.Variable(tf.zeros([32]), name="conv2_biases")  
    conv2 = tf.nn.conv2d(relu1, conv2_weights,  
        strides=[1, 1, 1, 1], padding='SAME')  
    return tf.nn.relu(conv2 + conv2_biases)
```

- tf.Variable을 이용하여 convnet 모델의 일부를 구현.
- 2개의 convolutional layer에 각 2개씩의 변수 정의.
- Layer가 늘어날수록, 변수의 양이 많아질수록 효과적인 관리가 힘들어짐.

```
# First call creates one set of variables.  
result1 = my_image_filter(image1)  
# Another set is created in the second call.  
result2 = my_image_filter(image2)
```

- result1과 result2는 서로 다른 결과값 도출: 변수의 공유가 이루어지지 않음.
- 변수의 공유가 필요한 경우( 예: training model과 test model간의 변수 공유) 문제가 생길 수 있음.

## tf.variable\_scope

### □ tf.variable\_scope('scope name'): 변수들의 이름 설정.

- tf.name\_scope()와 비슷한 역할을 변수에 대해 수행.
- tf.get\_variable() 함수를 통해 변수 설정: 기존 tf.Variable()의 공유 문제 해결.
- 변수들에 대해 접두사로 작용: scope 내에 만들어진 변수의 경우, 같은 scope의 소속이 됨 (상위 디렉토리의 개념과 비슷).  
→ 변수의 이름이 같아도 scope가 다르다면 충돌하지 않음.

예) scope name=conv일 경우, scope 내에 만들어진 weight variable의 이름: conv/weight.

```
with tf.variable_scope('conv1') as scope:
```

```
    kernel = tf.get_variable('kernel', [5, 5, 1, 32],  
                            initializer=tf.truncated_normal_initializer())  
    biases = tf.get_variable('biases', [32],  
                            initializer=tf.random_normal_initializer())  
    conv = tf.nn.conv2d(images, kernel, strides=[1, 1, 1, 1], padding='SAME')  
    conv1 = tf.nn.relu(conv + biases, name=scope.name)
```

```
with tf.variable_scope('conv2') as scope:
```

```
    kernel = tf.get_variable('kernel', [5, 5, 32, 64],  
                            initializer=tf.truncated_normal_initializer())  
    biases = tf.get_variable('biases', [64],  
                            initializer=tf.random_normal_initializer())  
    conv = tf.nn.conv2d(conv1, kernel, strides=[1, 1, 1, 1], padding='SAME')  
    conv2 = tf.nn.relu(conv + biases, name=scope.name)
```

- conv1과 conv2로 variable scope 지정.
- 같은 이름의 variable이 정의되어있지만, 서로 충돌하지 않음  
: 서로 속해있는 scope가 다르기 때문 (conv1/kernel, conv2/kernel).
- tf.get\_variable()을 이용하여 변수 설정:  
이후에 변수의 공유가 가능.



# tf.get\_variable vs. tf.Variable

## □ tf.Variable과 tf.get\_variable의 비교:

tf.Variable()

```
tf.Variable(initial_value, trainable=True, name=None, dtype=None, ...)
```

- 변수를 설정하고 지정.
- 원하는 값을 직접 지정 가능.  
예) `tf.Variable(3)`: 초기화될 때 해당 변수를 3으로 할당.
- 간단한 모델을 작성할 때 주로 쓰임.
- `tf.variable_scope()`의 영향을 받지 않음.
- 변수의 공유, 재사용이 불가능.

tf.get\_variable()

```
tf.get_variable(name, shape=None, dtype=None, initializer=None, ...)
```

- 변수를 설정하고 지정.
- 원하는 값을 직접 할당하는 것이 불가능:
  - 변수는 `initializer(shape)`의 값으로 초기화됨.  
예) 변수를 3으로 할당 때 명령어: `tf.get_variable(name="v", shape=[1], initializer=tf.constant_initializer(3))`
- `tf.variable_scope()`의 영향을 받음.
- `tf.variable_scope()`의 `reuse` 설정값에 의해 해당 변수의 생성, 재사용여부 결정.
  - `reuse=False`: `tf.Variable()`와 같이 변수 생성.

전체 변수 이름은 `tf.variable_scope()`의 영향을 받음.

같은 이름이 variable scope에 이미 있을 경우 오류 메세지 생성.

- `reuse=True`: 기존에 있는 변수를 찾아 사용.

입력 변수의 이름이 variable scope 안에 없는 경우

오류 메세지 생성.

## tf.get\_variable

- tf.get\_variable은 tf.variable\_scope 범위 내의 reuse 설정 여부에 따라서 다르게 작동함
- tf.get\_variable\_scope를 통해 현재 소속된 variable\_scope에 대한 정보를 얻을 수 있음
  - reuse == False: tf.Variable과 같이 새로운 변수를 생성
  - reuse == True: 같은 scope 안에 같은 이름을 갖는 변수를 찾아서 해당 변수를 재사용

```
tf.get_variable(name, shape=None, dtype=None, initializer=None, ...)
```

<reuse == False>

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1])  
assert v.name == "foo/v:0"
```

<reuse == True>

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1])  
with tf.variable_scope("foo", reuse=True):  
    v1 = tf.get_variable("v", [1])  
assert v1 == v
```

```
with tf.variable_scope("foo"):  
    v = tf.get_variable("v", [1])
```

```
tf.get_variable_scope().reuse_variables()
```

```
v1 = tf.get_variable("v", [1])
```

```
assert v1 == v
```

**False >> True**

(True >> False 설정은 할 수 없음)



# Xavier Initializer

- ❑ Xavier Initializer가 Random Initializer으로 실험적으로 모델의 좋은 성능을 보여주는 것으로 알려져 있음
- ❑ Random Initializer 관점에서 좋은 성능은 안정적인 Cost Function의 변동을 의미 (Variance of Cost Function)
- ❑ 각 뉴런에 연결된 weight 수 (fan-in, fan-out)를 바탕으로 파라미터 초기화를 수행함

$$W \sim U \left[ -\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}} \right]$$

↑                      ↑  
Fan-in    Fan-out

```
xavier_initializer(  
    uniform=True,  
    seed=None,  
    dtype=tf.float32  
)
```



```
-initial = tf.contrib.layers.xavier_initializer()  
tf.get_variable(name, shape, initializer=initial)
```

uniform=False로 설정시  
 $\sqrt{3/(fan\_in + fan\_out)}$ 의 표준편차를 갖는  
정규 분포를 바탕으로 초기화 진행

# Loss Functions in Tensorflow

- 다양한 Loss Functions을 계산할 수 있도록 Tensorflow 내 구현되어 있음
- tf.losses 모듈 안에 다양한 함수들 존재함

## Functions

`absolute_difference(...)` : Adds an Absolute Difference loss to the training procedure.

`add_loss(...)` : Adds a externally defined loss to the collection of losses.

`compute_weighted_loss(...)` : Computes the weighted loss.

`cosine_distance(...)` : Adds a cosine-distance loss to the training procedure.

`get_losses(...)` : Gets the list of losses from the loss\_collection.

`get_regularization_loss(...)` : Gets the total regularization loss.

`get_regularization_losses(...)` : Gets the list of regularization losses.

`get_total_loss(...)` : Returns a tensor whose value represents the total loss.

`hinge_loss(...)` : Adds a hinge loss to the training procedure.

`huber_loss(...)` : Adds a Huber Loss term to the training procedure.

`log_loss(...)` : Adds a Log Loss term to the training procedure.

`mean_pairwise_squared_error(...)` : Adds a pairwise-errors-squared loss to the training procedure.

`mean_squared_error(...)` : Adds a Sum-of-Squares loss to the training procedure.

`sigmoid_cross_entropy(...)` : Creates a cross-entropy loss using `tf.nn.sigmoid_cross_entropy_with_logits`.

`softmax_cross_entropy(...)` : Creates a cross-entropy loss using `tf.nn.softmax_cross_entropy_with_logits`.

`sparse_softmax_cross_entropy(...)` : Cross-entropy loss using `tf.nn.sparse_softmax_cross_entropy_with_logits`.

## `tf.losses.mean_squared_error`

```
mean_squared_error(  
    labels,  
    predictions,  
    weights=1.0,  
    scope=None,  
    loss_collection=tf.GraphKeys.LOSSES,  
    reduction=Reduction.SUM_BY_NONZERO_WEIGHTS  
)
```

- **labels**: 실제 참 값 (ground-truth)
- **predictions**: 모델 예측 값
- **weights**: 가중치 평균 여부
  - 1.0으로 설정: 산술 평균 계산
  - labels과 같은 rank의 tensor 설정: 해당 가중치에 맞게 가중치 평균

tf 내 구현된 함수들을 사용하기도 하지만,  
직접 계산 식을 입력하여 계산하는 경우가 많음..

# Optimizers in Tensorflow

- Tensorflow 내 다양한 Optimizer를 사용할 수 있도록 구현되어 있음
- 필요에 따라 원하는 Optimizer를 선언하여 사용함

## Optimizers

- tf.train.Optimizer
- tf.train.GradientDescentOptimizer
- tf.train.AdadeltaOptimizer
- tf.train.AdagradOptimizer
- tf.train.AdagradDAOptimizer
- tf.train.MomentumOptimizer
- tf.train.AdamOptimizer
- tf.train.FtrlOptimizer
- tf.train.ProximalGradientDescentOptimizer
- tf.train.ProximalAdagradOptimizer
- tf.train.RMSPropOptimizer

tf.train.AdamOptimizer

`__init__`

```
__init__(  
    learning_rate=0.001,  
    beta1=0.9,  
    beta2=0.999,  
    epsilon=1e-08,  
    use_locking=False,  
    name='Adam'  
)
```

`minimize`

```
minimize(  
    loss,  
    global_step=None,  
    var_list=None,  
    gate_gradients=GATE_OP,  
    aggregation_method=None,  
    colocate_gradients_with_ops=False,  
    name=None,  
    grad_loss=None  
)
```

Methods

`_init_`  
`apply_gradients`  
`compute_gradients`  
`get_name`  
`get_slot`  
`get_slot_names`  
`minimize`

Class Members

GATE\_GRAPH  
GATE\_NONE  
GATE\_OP

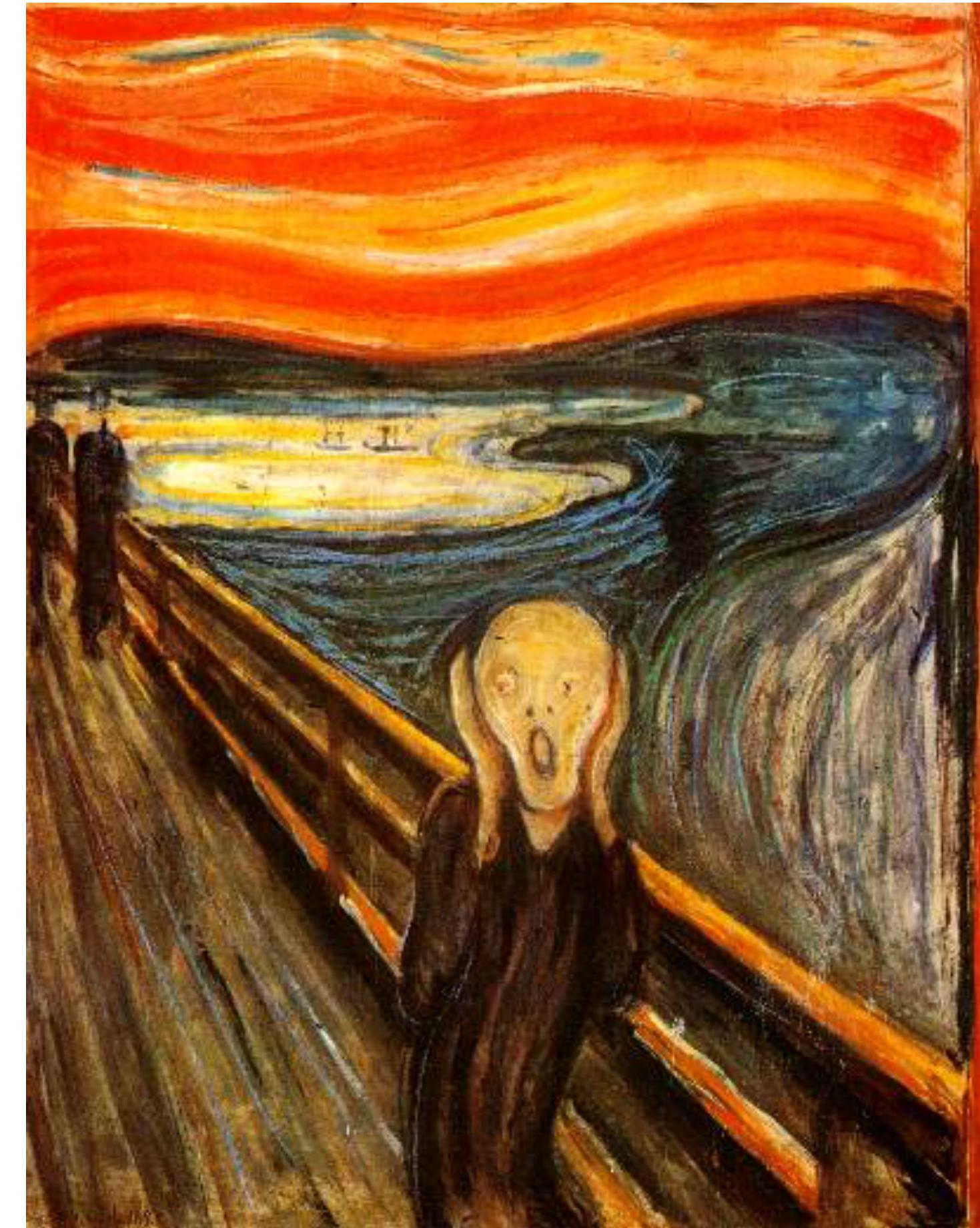
## Example

```
optimizer = tf.train.AdamOptimizer(LEARNING_RATE).minimize(loss,  
                                         global_step=global_step)
```

# Saving Current Information

# Importance of Saving

- 모델이 복잡할 경우, 또는 투입자료가 많아질 경우, 학습시간은 길어짐.
- 세상은 넓고 예상하지 못한 일들은 언제나 일어남...
  - 작업을 중간에 중지해야 할 경우?
  - 갑자기 컴퓨터가 꺼지면?
  - (특히) ssh, aws 등 외부로 접속하여 사용하는 경우, 통신이 끊기면??



## tf.train.Saver()

- **tf.train.Saver.save()**: 주기적으로 일정 step(반복 횟수)마다 모델의 parameter 저장
- 저장할 때의 step을 **checkpoint**라고 부름.
- **Metagraph**: tensorflow의 graph 정보(모델 실행에 필요한 정보)를 저장

```
tf.train.Saver.save(sess, save_path, global_step=None, latest_filename=None,  
meta_graph_suffix='meta', write_meta_graph=True, write_state=True)
```

- `global_step`: 저장 파일이름에 학습의 반복 횟수 추가.
- `latest_filename`: 가장 최근의 checkpoint를 가지고 있는 파일 지정. Checkpoint 폴더에 있을 경우 프로그램이 자동으로 찾아냄.
- `meta_graph_suffix`: MetaGraphDef protocol buffer 설정(이전의 process들을 이어가기 위한 정보들을 포함하는 형식).
- `write_meta_graph`: metagraph파일을 만들지의 여부 설정.
- `write_state`: CheckpointStateProto의 기록 여부 (Checkpoint의 상태(저장 폴더, 경로 등)를 보여주는 정보)

## tf.train.Saver.save() code

- 일반적으로 모델의 저장은 학습을 반복하는 for문 중간에 추가함

- Optimizer의 일정 step마다 저장.

### 저장 관리 오퍼레이션 선언

```
saver=tf.train.Saver()  
  
with tf.Session() as sess:  
    for step in range(training_step):  
        sess.run([optimizer])  
        if (step + 1) % 1000 == 0:  
            saver.save(sess, 'checkpoints/skip-gram', global_step=model.global_step)
```

- 반복 횟수가 1000의 배수일 때 파일 저장
- checkpoints 폴더의 filename(skip-gram)-global step의 이름으로 저장됨.

- session을 저장함 (why?)
  - session은 쉽게 생각해 프로그램 단위!
  - 그래프 정보와 variable 정보 등 모두 포함

### Global Step:

일반적으로 프로그램 구현시 진행 중인 학습 반복 수를 변수로 저장함 (why?)

## Global Step

- Global Step: 학습의 step(반복 횟수) 저장
  - Global step의 설정을 통해 **저장 당시의 step**을 알 수 있음
  - 모델을 학습시키며 저장은 한 번이 아닌 수시로 일어나며, 저장한 파일을 불러들여 학습을 이어나갈 때도 학습 진행 정도에 대한 정보가 필요!

# Global Step

## □ Global Step: 학습의 step(반복 횟수) 저장

- Global step의 설정을 통해 **저장 당시의 step**을 알 수 있음
- 모델을 학습시키며 저장은 한 번이 아닌 수시로 일어나며, 저장한 파일을 불러들여 학습을 이어나갈 때도 학습 진행 정도에 대한 정보가 필요!

## □ Global Step의 적용법

1. 초기 설정: 초기값 0, optimizer에 의해 학습되지 않도록(trainable=False) 설정

```
self.global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
```

2. Optimizer 함수의 변수로 적용: 학습을 반복할 때마다 1씩 증가.

Optimizer가 자동으로 변수 업데이트함

```
self.optimizer = tf.train.GradientDescentOptimizer(self.lr).minimize(self.loss, global_step=self.global_step)
```

3. **tf.train.Saver.save()**에 적용: 저장 당시의 step을 파일명 뒤에 표시.

```
tf.train.Saver.save(sess, 'checkpoints/skip-gram', global_step=model.global_step)
```

## Global Step: checkpoint folder

- Global step을 적용할 경우, 저장 폴더(여기서는 checkpoints)에 저장되는 파일(skip-gram) :

checkpoint	265 bytes
skip-gram-1000.data-00000-of-00001	51.4 MB
skip-gram-1000.index	261 bytes
skip-gram-1000.meta	87 KB
skip-gram-2000.data-00000-of-00001	51.4 MB
skip-gram-2000.index	261 bytes
skip-gram-2000.meta	87 KB
skip-gram-3000.data-00000-of-00001	51.4 MB
skip-gram-3000.index	261 bytes
skip-gram-3000.meta	87 KB
skip-gram-4000.data-00000-of-00001	51.4 MB
skip-gram-4000.index	261 bytes
skip-gram-4000.meta	87 KB

저장 주기: 1000회

- **meta file:** describes the saved graph structure, includes GraphDef, SaverDef, and so on; then apply `tf.train.import_meta_graph('/tmp/model.ckpt.meta')`, will restore `Saver` and `Graph`.
- **index file:** it is a string-string immutable table(tensorflow::Table). Each key is a name of a tensor and its value is a serialized BundleEntryProto. Each BundleEntryProto describes the metadata of a tensor: which of the "data" files contains the content of a tensor, the offset into that file, checksum, some auxiliary data, etc.
- **data file:** it is TensorBundle collection, save the values of all variables.

Then, How to Restore?

## Restore

- **tf.train.Saver.restore(sess,file\_path):** tf.train.Saver로 저장해놓은 파일을 다시 불러들임.

- 저장한 부분부터 이어서 모델 학습이 가능.

```
tf.train.Saver.restore(sess, 'skip-gram-10000')
```

- Global step=10000일 때의 parameter 정보를 불러들임.
- 단점: 저장한 파일의 이름을 알고 있어야 함, 파일의 이름을 직접 입력하여 불러들임

## Restore

### □ **tf.train.Saver.restore(sess,file\_path): tf.train.Saver.save로 저장해놓은 파일을 다시 불러들임.**

- 저장한 부분부터 이어서 모델 학습이 가능.

```
tf.train.Saver.restore(sess, 'skip-gram-10000')
```

- Global step=10000일 때의 parameter 정보를 불러들임.
- 단점: 저장한 파일의 이름을 알고 있어야 함, 파일의 이름을 직접 입력하여 불러들임.

### □ **tf.train.get\_checkpoint\_state('directory name'): 입력한 폴더에 저장 파일이 있는지를 체크.**

- 저장파일을 발견할 경우, 경로를 지정하고 그 중 **가장 최신의 파일을 최종 경로로** 지정.

```
tf.train.get_checkpoint_state('directory')
```

```
model_checkpoint_path: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-59999"
all_model_checkpoint_paths: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-51999"
all_model_checkpoint_paths: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-53999"
all_model_checkpoint_paths: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-55999"
all_model_checkpoint_paths: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-57999"
all_model_checkpoint_paths: "/Users/hch/PycharmProjects/tensorflow_practice/data/checkpoints/skip-gram-59999"
```

tf.train.get\_checkpoint\_state의 결과

### □ **두 명령어를 조합: 입력 폴더에 있는 최신의 저장 파일을 자동으로 불러들임.**

```
ckpt = tf.train.get_checkpoint_state(os.path.dirname('directory'))
if ckpt and ckpt.model_checkpoint_path:
    saver.restore(sess, ckpt.model_checkpoint_path)
```

## Save & Restore code

- 코드의 진행: 먼저 directory에서 불러온 뒤, 학습을 수행하면서 파일을 저장.

```
saver=tf.train.Saver()

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    ckpt = tf.train.get_checkpoint_state('checkpoints')
    # if that checkpoint exists, restore from checkpoint
    if ckpt and ckpt.model_checkpoint_path:
        saver.restore(sess, ckpt.model_checkpoint_path)

    for step in range(training_step):
        sess.run([optimizer])
        if (step + 1) % 1000 == 0:
            saver.save(sess, 'checkpoints/skip-gram', global_step=model.global_step)
```

: 불러오는 부분. checkpoints 폴더에 저장 파일이 있는지 확인하고 있으면 최신의 파일으로 불러들임.

: 저장하는 부분. 반복 횟수가 1000의 배수에 해당할 때마다 주기적으로 checkpoints 폴더에 파일 저장.

# Example:MNIST

# MNIST



MNIST(Mixed National Institute of Standards and Technology database)

- 손으로 쓴 0 ~ 9 사이의 숫자들의 흑백 image들과 이에 대응되는 label들로 이루어진 자료.
  - 각 숫자 image들은 모두  $28 \times 28 \times 1$ 의 pixel로 이루어져 있음(마지막 1: 흑백을 의미).
  - 자료의 수: 55,000개의 training image, 5,000개의 validation image, 10,000개의 test image.
  - Tensorflow 안에 tutorial로 자료가 존재(tensorflow.examples.tutorials.mnist): 바로 읽어들일 수 있음.

0  
1  
2  
3  
4  
5  
6  
7  
8  
9 9

# Model

□ 목표: Convnet을 통해 주어진 MNIST 자료를 알맞게 분류하는 것.

- 각 image들이 가리키는 숫자를 찾아내는 것이 목표.

□ 사용 Convnet 모델

- 2개의 convolutional layer 이용.

- 활성화 함수: Relu function.

- 최종 classification: softmax function을 이용.

- Overfitting 예방: - 마지막 FC layer에서 dropout 적용.

- 각 convolution layer 뒤에 pooling layer(max pooling) 배치.

Original Image 28 x 28 x 1 	Conv1 Filter: 5 x 5 x 1 x 32 Stride: 1, 1, 1, 1 Out: 28 x 28 x 32 Relu Maxpool (2 x 2 x 1) Out: 14 x 14 x 32	Conv2 Filter: 5 x 5 x 32 x 64 Stride: 1, 1, 1, 1 Out: 14 x 14 x 64 Relu Maxpool (2 x 2 x 1) Out: 7 x 7 x 64	Fully connected W: 7*7*64 x 1024 Out: 1 x 1024 Relu Out: 1 x 1024	Softmax W: 1024 x 10 Out: 1 x 10 Softmax 1 x 10
---	--	---	---	---

# Model Construction: Preliminary work

- Preliminary: 자료 읽기, initial parameter 설정.

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

```
LEARNING_RATE = 0.001
BATCH_SIZE = 128
SKIP_STEP = 10
DROPOUT = 0.75
N_EPOCHS = 10
N_CLASSES = 10
```

: MNIST 자료 읽어들이기.

: 초기 parameter 설정

- SKIP\_STEP: 저장, loss 알림 주기 설정 (계산의 반복횟수가 10의 배수일 때 parameter 저장 & loss 알림).
- DROPOUT: Dropout의 비율 설정(FC layer의 node 중 75%는 최종 계산에 참여. 25%는 계산에서 누락됨).
- N\_EPOCHES: training data의 학습 횟수 설정 (모든 training data를 10번 반복하여 학습).
- N\_CLASS: 최종 class의 수 (0 ~ 9의 class가 있으므로 총 10개).

# Model Construction : Placeholder

- Image와 label의 자료를 투입할 placeholder 생성.

```
with tf.name_scope('data'):
    X = tf.placeholder(tf.float32, [None, 784], name="X_placeholder")
    Y = tf.placeholder(tf.float32, [None, 10], name="Y_placeholder")

dropout = tf.placeholder(tf.float32, name='dropout')
```

- X: image feature의 정보(28×28 pixel을 784-dim vector로 변환).  
Y: image feature에 대응되는 label.
- X의 shape: [None, 784]: None부분은 batch size 의미.
  - 직접 숫자를 명시하지 않음으로써 batch size의 변화에 자동으로 대응(tensorflow가 자동으로 size 계산).
  - Y의 shape 역시 같은 의미를 가짐.
- dropout: dropout 확률을 투입하는 placeholder.

# Model Construction : Conv & Pool

## □ Convolution & Pooling layer의 생성: tf.variable\_scope를 통해 각 layer 구분.

```
with tf.variable_scope('conv1') as scope:  
    images = tf.reshape(X, shape=[-1, 28, 28, 1])  
    kernel = tf.get_variable('kernel', [5, 5, 1, 32],  
                            initializer=tf.truncated_normal_initializer())  
    biases = tf.get_variable('biases', [32],  
                            initializer=tf.random_normal_initializer())  
    conv = tf.nn.conv2d(images, kernel, strides=[1, 1, 1, 1], padding='SAME')  
    conv1 = tf.nn.relu(conv + biases, name=scope.name)
```

```
with tf.variable_scope('pool1') as scope:  
    pool1 = tf.nn.max_pool(conv1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],  
                          padding='SAME')
```

```
with tf.variable_scope('conv2') as scope:  
    kernel = tf.get_variable('kernels', [5, 5, 32, 64],  
                            initializer=tf.truncated_normal_initializer())  
    biases = tf.get_variable('biases', [64],  
                            initializer=tf.random_normal_initializer())  
    conv = tf.nn.conv2d(pool1, kernel, strides=[1, 1, 1, 1], padding='SAME')  
    conv2 = tf.nn.relu(conv + biases, name=scope.name)
```

```
with tf.variable_scope('pool2') as scope:  
    pool2 = tf.nn.max_pool(conv2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],  
                          padding='SAME')
```

첫번째 convolutional layer: input을 처음으로 거치는 layer(conv1).

- images: input data의 크기: BATCH\_SIZE×28×28×1  
(-1은 None과 같은 의미를 가짐).
- kernel, biases: kernel(5×5×1×32)과 bias(32)의 크기 설정.
- Output size: BATCH\_SIZE×28×28×32.

: Activation function으로 Relu 사용.

첫번째 pooling layer: conv1을 거친 자료를 처리(pool1).

- max pooling 방법을 이용하여 자료 축소.
- width, height size를 절반으로 축소.
- Output size : BATCH\_SIZE×14×14×32.

두번째 convolutional layer: pooling layer를 거친 자료 처리(conv2).

- 투입되는 자료의 크기가 다르므로(BATCH\_SIZE×14×14×32) kernel과 bias의 크기 변화: 5×5×32×64 for kernel, 64 for bias.
- Output size : BATCH\_SIZE×14×14×64.

: Activation function으로 Relu 사용.

두번째 pooling layer: conv2을 거친 자료를 처리(pool2).

- max pooling 방법을 이용하여 자료 축소.
- Output size : BATCH\_SIZE×7×7×64.

# Model Construction: FC

- Pooling layer를 거친 자료를 마지막으로 처리하는 layer.

```
with tf.variable_scope('fc') as scope:  
    input_features = 7 * 7 * 64  
    w = tf.get_variable('weights', [input_features, 1024],  
                        initializer=tf.truncated_normal_initializer())  
    b = tf.get_variable('biases', [1024],  
                        initializer=tf.random_normal_initializer())  
  
    pool2 = tf.reshape(pool2, [-1, input_features])  
    fc = tf.nn.relu(tf.matmul(pool2, w) + b, name='relu')  
  
    fc = tf.nn.dropout(fc, dropout, name='relu dropout')
```

pool2를 거친 자료를 처리.

- Input size: BATCH\_SIZE×7×7×64.
- Output size: BATCH\_SIZE×1024.

 : 쉬운 자료 처리를 위해 input size를 matrix로 변화.  
BATCH\_SIZE×input\_features (input\_features= 7×7×64).

 : Dropout 작업 수행  
일정 확률(1-DROPOUT)로 마지막 layer의 node를 삭제: overfitting 예방.

```
with tf.variable_scope('softmax_linear') as scope:  
    w = tf.get_variable('weights', [1024, N_CLASSES],  
                        initializer=tf.truncated_normal_initializer())  
    b = tf.get_variable('biases', [N_CLASSES],  
                        initializer=tf.random_normal_initializer())  
    logits = tf.matmul(fc, w) + b
```

마지막으로 자료 처리.

- Input size: BATCH\_SIZE×1024.
- Output size: BATCH\_SIZE×10.
- 각 class에 대한 최종 결과값 결정.
- 값이 클수록 해당 class에 속할 확률이 높다는 것을 의미.
- Softmax 함수는 적용하지 않은 상태.

# Model Construction: Loss, Optimizer, and Summary

- Loss function: Cross entropy 이용.
- Optimizer: Adam optimizer를 이용하여 계산 진행.
- Summary: Loss curve, loss histogram을 통해 학습 진행 과정을 시각화.

```
with tf.name_scope('loss'):
    entropy = tf.nn.softmax_cross_entropy_with_logits(logits, Y)
    loss = tf.reduce_mean(entropy, name='loss')
```

```
global_step = tf.Variable(0, dtype=tf.int32, trainable=False, name='global_step')
optimizer = tf.train.AdamOptimizer(LEARNING_RATE).minimize(loss,
    global_step=global_step)
```

```
with tf.name_scope("summary"):
    tf.summary.scalar("loss", loss)
    tf.summary.histogram("histogram loss", loss)

    summary_op = tf.summary.merge_all()
```

Loss function 정의

- softmax\_linear layer의 값(logit)과 label을 이용, cross entropy를 구함.
- tf.reduce\_mean: Cross entropy의 batch-sample에서의 평균을 loss function으로 정의.

Optimizer 설정: 목적은 Cross entropy의 평균의 최소화.

- Global step 설정: 반복횟수를 저장.
- Adam optimizer를 채택: learning rate=0.001

Summary operation 생성

- Loss graph와 histogram 생성: 학습 과정 시각화.

# Summary visualization

## Tensorboard를 통하여 모델의 학습상황을 확인 가능.

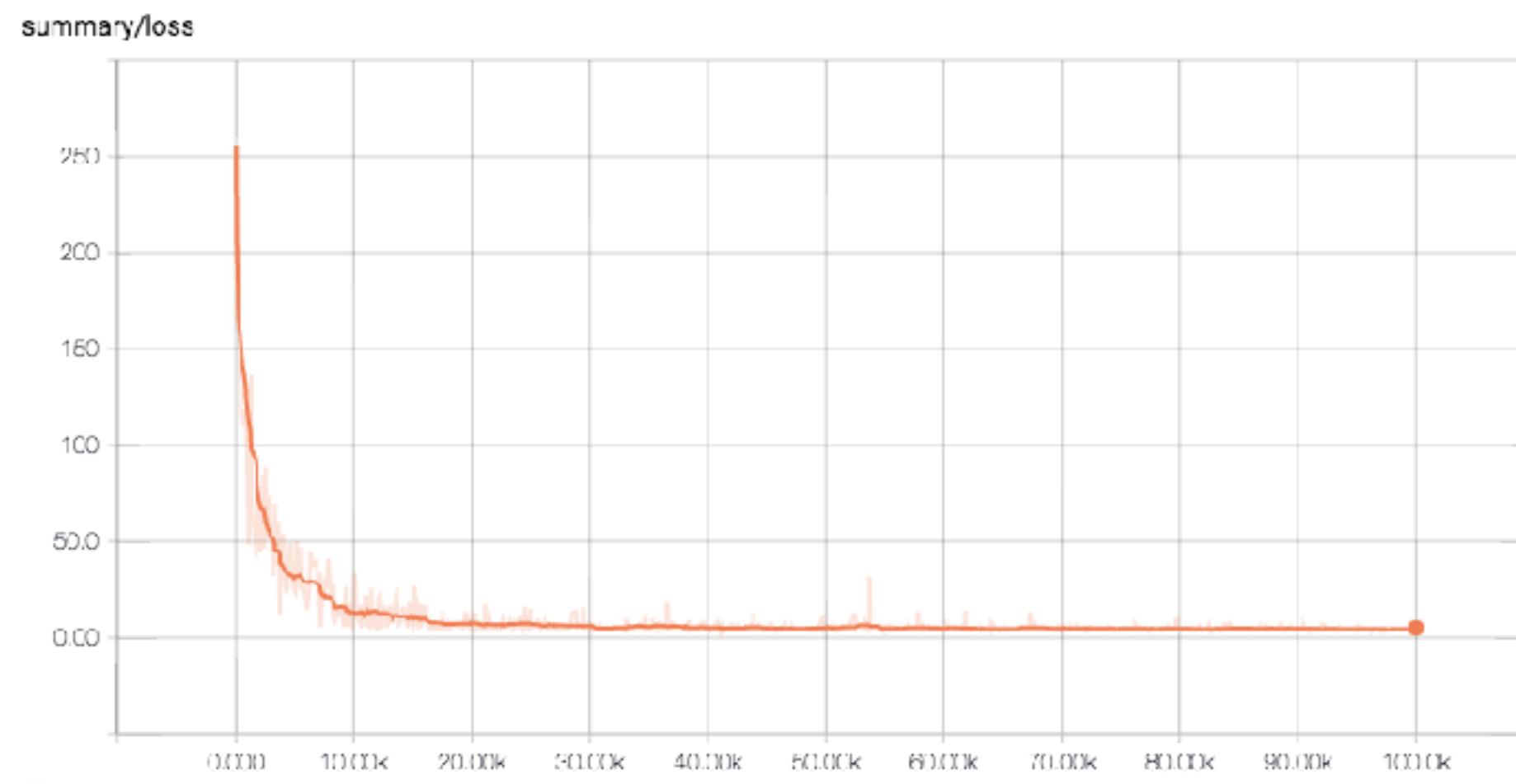
- tf.Summary() 명령어를 사용 (별도의 Output이 존재하지 않으니 주의! **오퍼레이션 선언만 존재**)

tf.Summary.scalar("loss", self.loss): loss scalar의 반복횟수에 따른 변화를 나타냄.

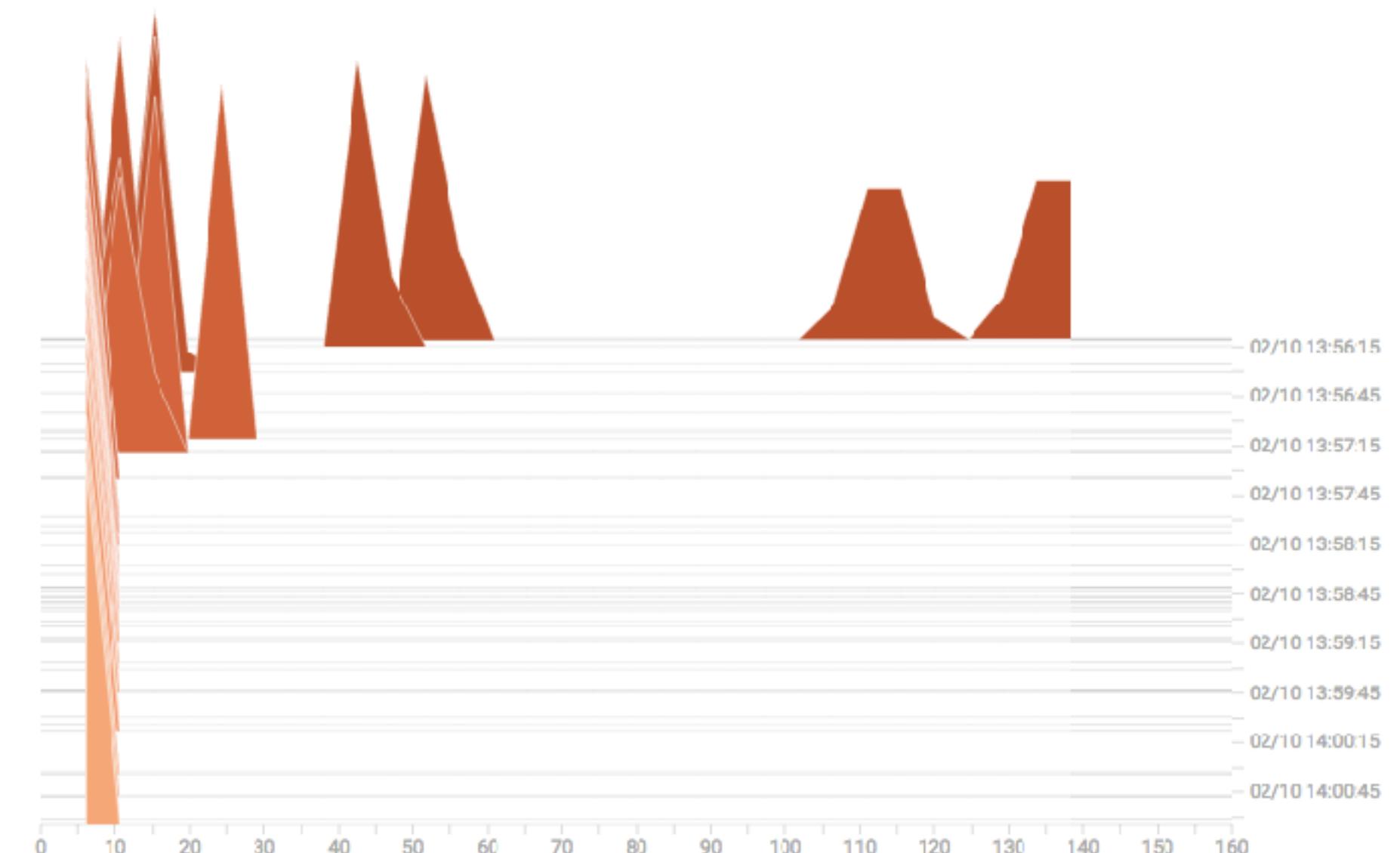
tf.Summary.scalar("accuracy", self.accuracy): accuracy scalar의 반복횟수에 따른 변화를 나타냄.

tf.Summary.histogram("histogram loss", self.loss): loss의 히스토그램을 나타냄.

tf.Summary.scalar("loss",self.loss)



tf.Summary.histogram("histogram loss",self.loss)



# Summary Process

## □ Summary 출력 과정

### 1. Summary 생성

```
def _create_summaries(self):
    with tf.name_scope("summary"):
        tf.summary.scalar("loss", self.loss)
        tf.summary.histogram("histogram loss", self.loss)
        # because you have several summaries, we should merge them all
        # into one op to make it easier to manage
        self.summary_op = tf.summary.merge_all()
```

- Name scope를 통해 summary operation들을 묶음.
- tf.summary.merge.all(): 여러개의 summary operation들을 하나로 통합

### 2. Summary 실행 (Summary는 operation이므로 tf.Session.run()으로 실행)

- Summary operation은 학습과 무관하므로 별도로 실행시켜줘야함!

```
loss_batch, _, summary = sess.run([model.loss, model.optimizer, model.summary_op],
                                  feed_dict=feed_dict)
```

### 3. Summary 기록: FileWriter를 이용하여 Summary 그래프를 출력.

```
writer = tf.summary.FileWriter('./graph directory')
writer.add_summary(summary, global_step=model.global_step)
```

- Flow graph가 저장된 곳에 저장됨.

# Computation: Learning from Training Data

- 작성된 모델을 이용하여 training data 학습.

```
with tf.Session() as sess:  
    sess.run(tf.global_variables_initializer())  
    saver = tf.train.Saver()  
    writer = tf.summary.FileWriter('./my_graph/mnist', sess.graph)  
  
    ckpt = tf.train.get_checkpoint_state('/checkpoints/convnet_mnist_new')  
    if ckpt and ckpt.model_checkpoint_path:  
        saver.restore(sess, ckpt.model_checkpoint_path)  
  
    initial_step = global_step.eval()  
  
    start_time = time.time()  
    n_batches = int(mnist.train.num_examples / BATCH_SIZE)  
  
    total_loss = 0.0  
    for index in range(initial_step, n_batches * N_EPOCHS): # train the model n_epochs times  
        X_batch, Y_batch = mnist.train.next_batch(BATCH_SIZE)  
        _, loss_batch, summary = sess.run([optimizer, loss, summary_op],  
                                         feed_dict={X: X_batch, Y:Y_batch, dropout: DROPOUT})  
        writer.add_summary(summary, global_step=index)  
        total_loss += loss_batch  
        if (index + 1) % SKIP_STEP == 0:  
            print('Average loss at step {}: {:.5f}'.format(index + 1, total_loss / SKIP_STEP))  
            total_loss = 0.0  
            saver.save(sess, '/checkpoints/convnet_mnist_new/mnist-convnet', index)  
  
    print("Optimization Finished!") # should be around 0.35 after 25 epochs  
    print("Total time: {} seconds".format(time.time() - start_time))
```

저장된 파일의 유무 체크

- 파일이 있을 경우 파일을 불러옴.

학습 수행

- 횟수: (n\_batches \* N\_EPOCHS) – 최신 global step.  
N\_EPOCHS=1일 경우 training data 전체를 한번 학습.  
N\_EPOCHS: training data를 몇번 학습하는지를 나타냄.
- Summary 저장: loss curve, histogram 저장.
- Parameter 저장:
  - 파일 이름: mnist-convnet-(global step)의 꼴.
  - checkpoints/convnet\_mnist\_new 폴더에 저장.
- . MNIST의 training data 이용.

## Calculation: Test set

### □ Training data에서 학습한 모델을 test set에 적용.

- Training data 학습 때와 같은 session 이용.
- tf.variable\_scope, tf.get\_variable: test data가 학습된 parameter를 공유할 수 있도록 함.

```
n_batches = int(mnist.test.num_examples/BATCH_SIZE)
total_correct_preds = 0
for i in range(n_batches):
    X_batch, Y_batch = mnist.test.next_batch(BATCH_SIZE)
    _, loss_batch, logits_batch = sess.run([optimizer, loss, logits],
                                           feed_dict={X: X_batch, Y:Y_batch, dropout: DROPOUT})
    preds = tf.nn.softmax(logits_batch)
    correct_preds = tf.equal(tf.argmax(preds, 1), tf.argmax(Y_batch, 1))
    accuracy = tf.reduce_sum(tf.cast(correct_preds, tf.float32))
    total_correct_preds += sess.run(accuracy)

print("Accuracy {0}".format(total_correct_preds/mnist.test.num_examples))
```

Test 수행: train에서 구한 parameter를 test set에 적용.

정확도 측정

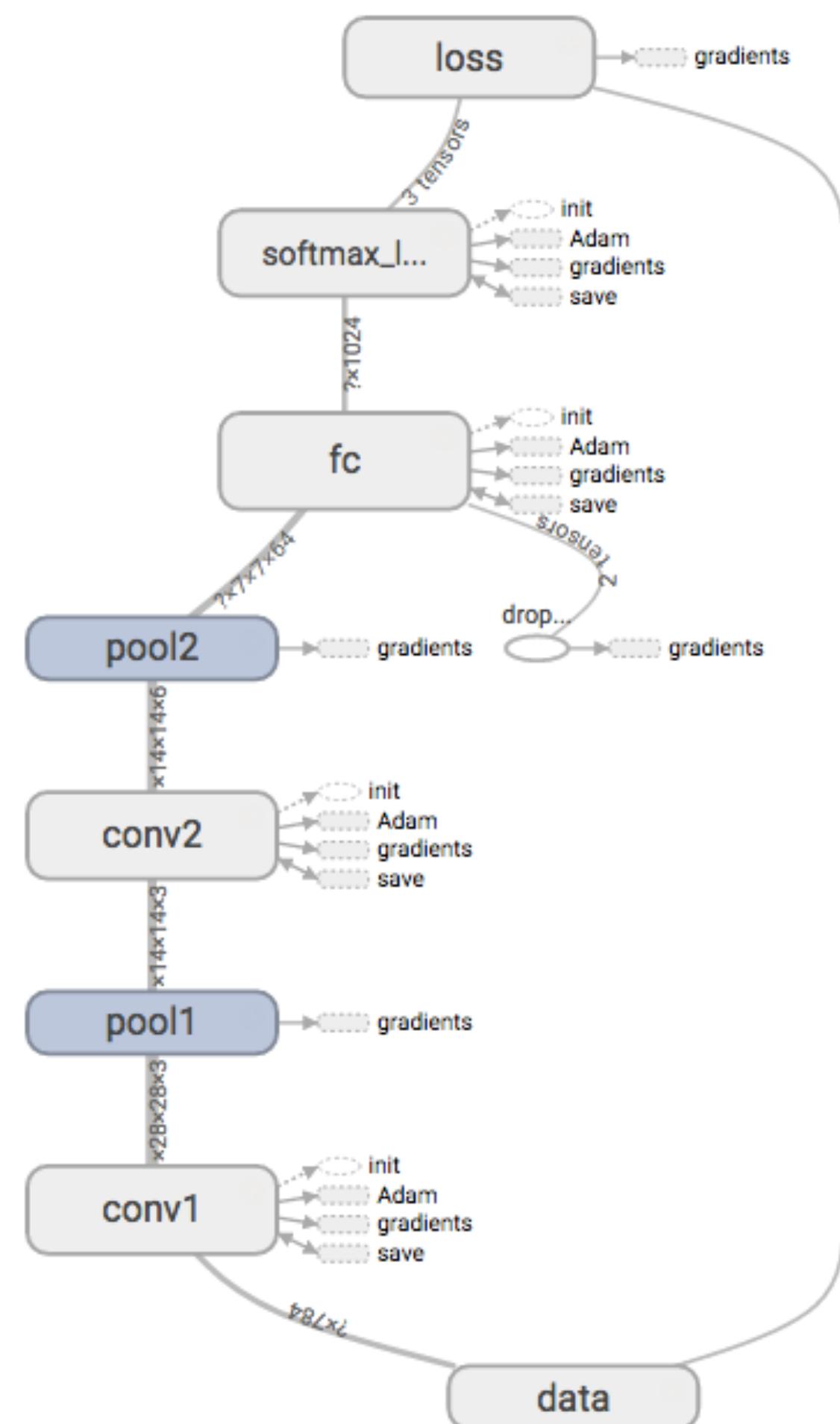
- preds: test set의 image가 각 class에 속할 확률 계산.
- total\_correct\_preds: 실제 label과 계산값과의 일치 개수.

# Flow Graph of the model

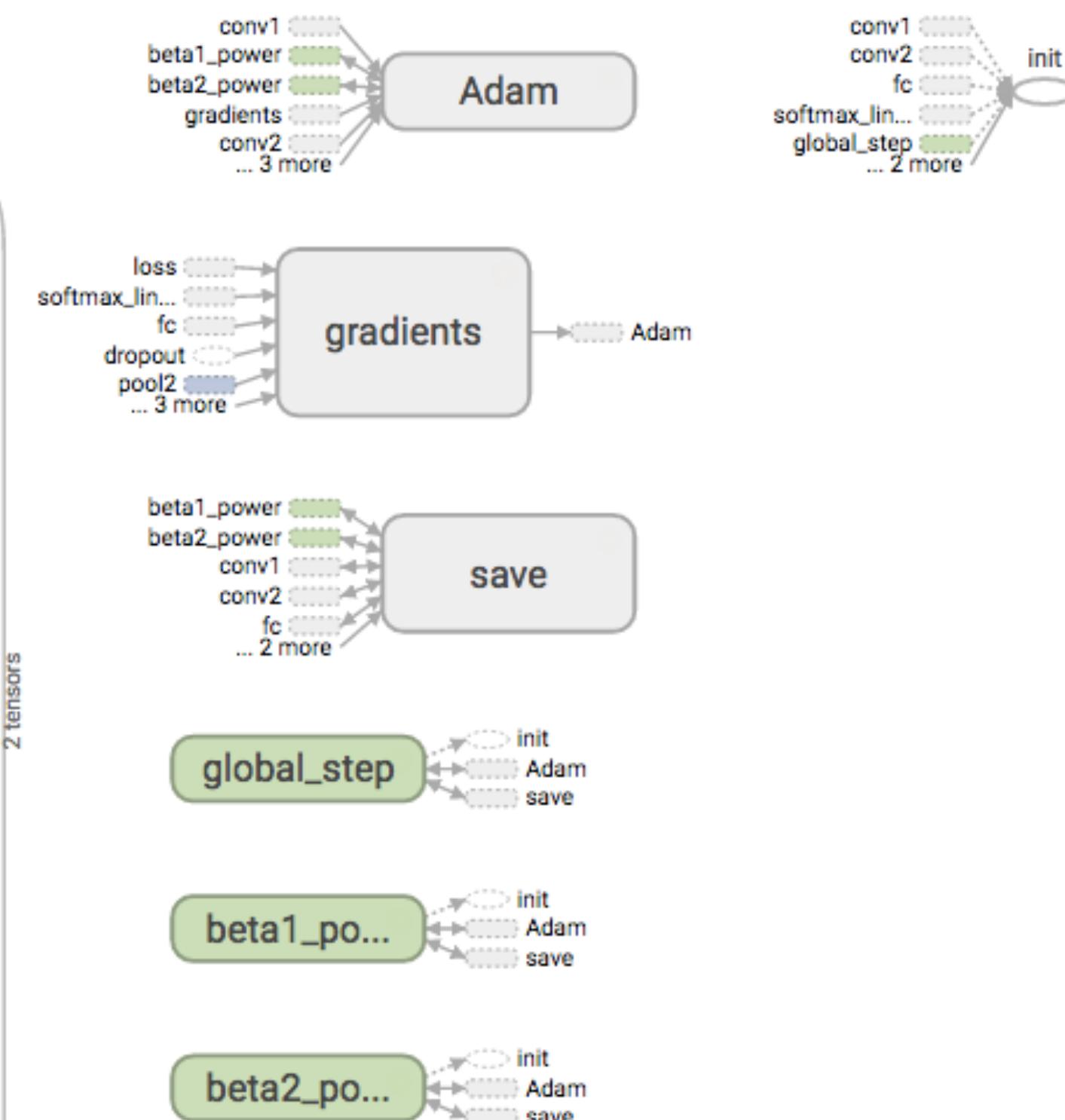
□ Tensorboard를 통해 모델의 흐름도를 그림으로 확인 가능.

- tf.name\_scope, tf.variable\_scope를 통해 여러 variable을 보기 쉽게 묶음.

Main Graph

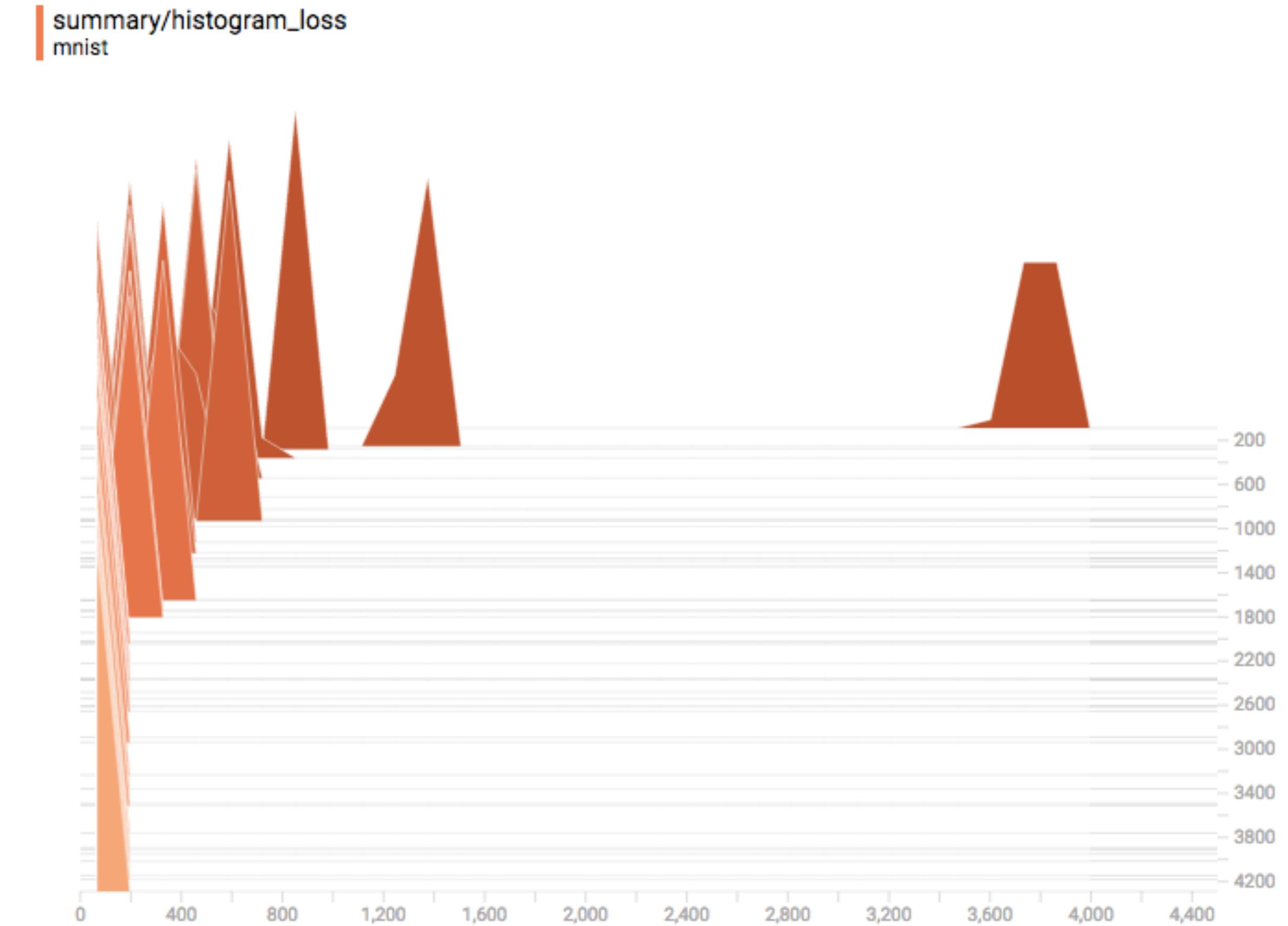
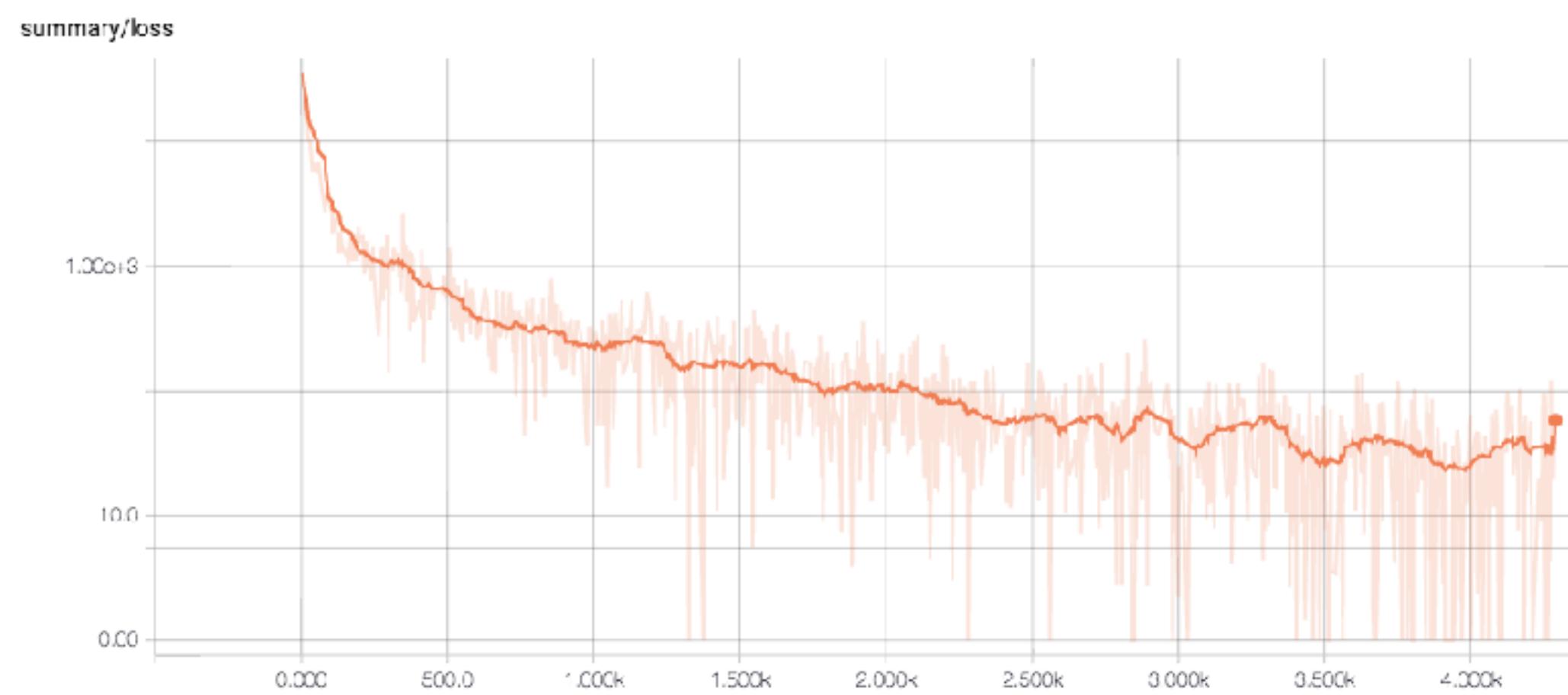
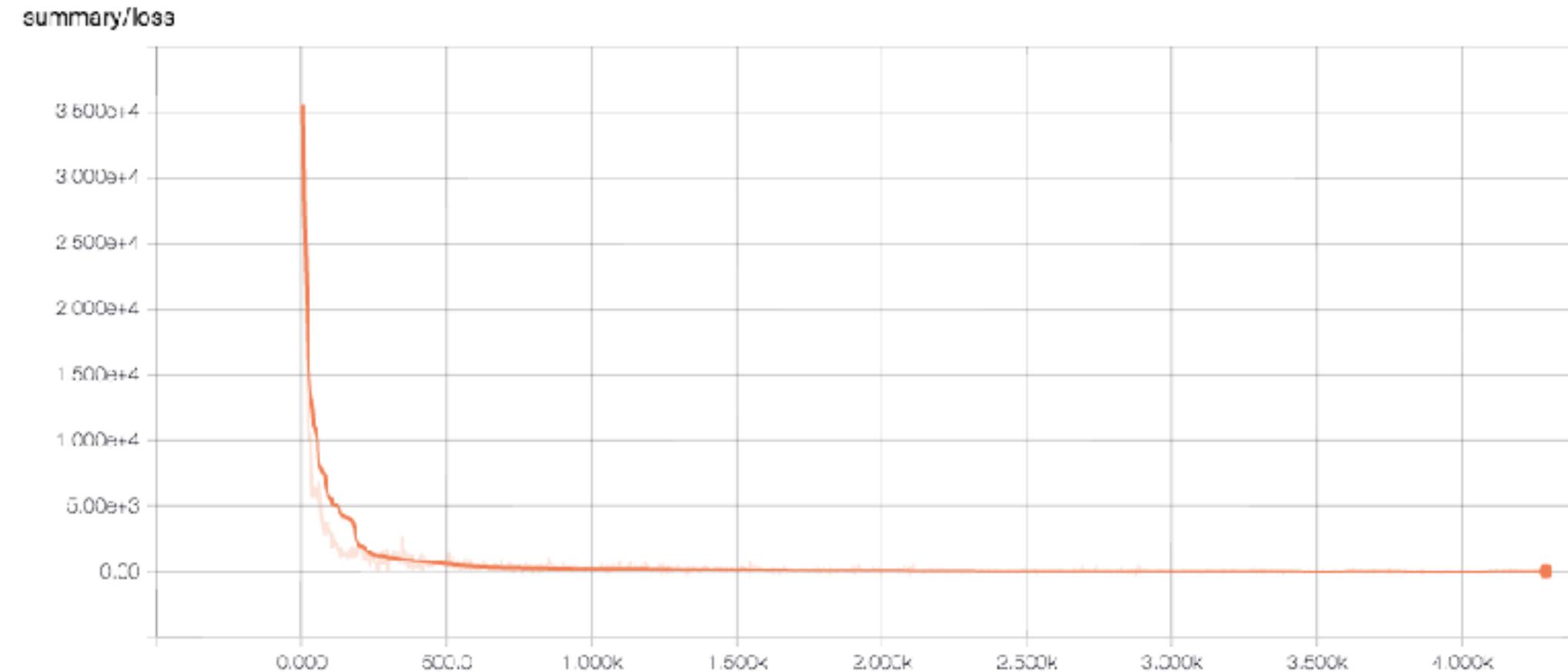


Auxiliary Nodes



# Result: Learning curve and Histogram

- Tensorboard를 이용하여 learning curve와 loss histogram을 관찰 가능.
  - 학습이 잘 이루어지고 있음을 보여줌.



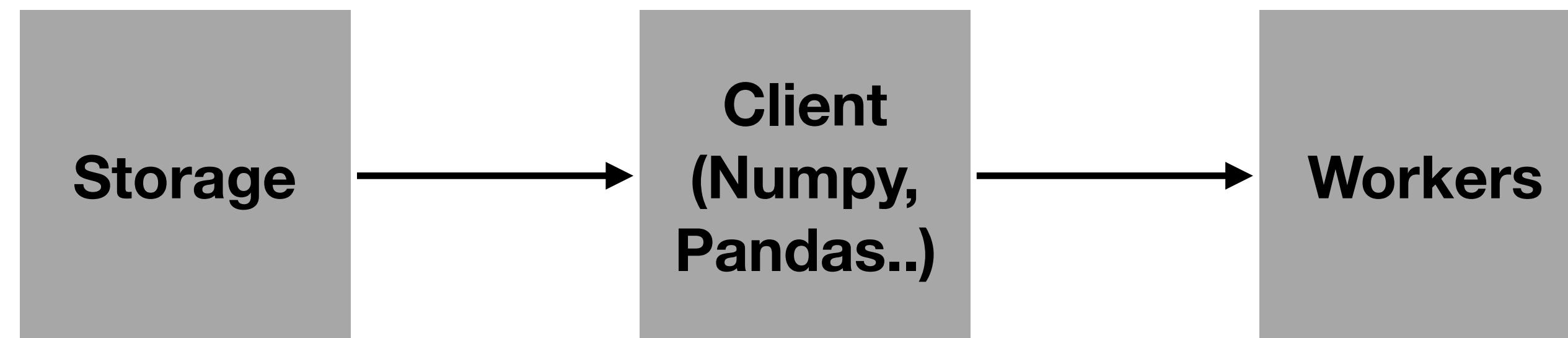
# Input PipeLines in Tensorflow

# Types of Input Pipeline

- Tensorflow 내에서 Input data를 생성하는 방법은 크게 3가지로 나뉨

1) tf.constant: 프로그램 내에서 데이터를 직접 생성하여 사용하는 경우

2) Feed Dictionary: 외부 클라이언트(client)에서 파일을 입력받은 후, Tensorflow로 넘겨주는 경우 (most of cases)



3) Data Readers: tensorflow 내 구현되어 있는 Data Reader 클래스 사용하는 경우 (관련 코드 별도 제공)



# Data Readers

- 외부 Client를 거치지 않고 데이터를 받아오는 경우 사용
- 데이터 종류에 따라 사용하는 Data Reader가 다름

**tf.TextLineReader**

Outputs the lines of a file delimited by newlines

E.g. text files, CSV files

**tf.FixedLengthRecordReader**

Outputs the entire file when all files have same fixed lengths

E.g. each MNIST file has 28 x 28 pixels, CIFAR-10 32 x 32 x 3

**tf.WholeFileReader**

Outputs the entire file content. This is useful when each file contains a sample

**tf.TFRecordReader**

Reads samples from TensorFlow's own binary format (TFRecord)

**tf.ReaderBase**

Allows you to create your own readers

```
filename_queue = tf.train.string_input_producer(filenames)
reader = tf.TextLineReader(skip_header_lines=1) # skip the first line in the file
_, value = reader.read(filename_queue)

record_defaults = [[1.0] for _ in range(N_FEATURES)]

content = tf.decode_csv(value, record_defaults=record_defaults)
```



# Recurrent Neural Network

# Recurrent Neural Network

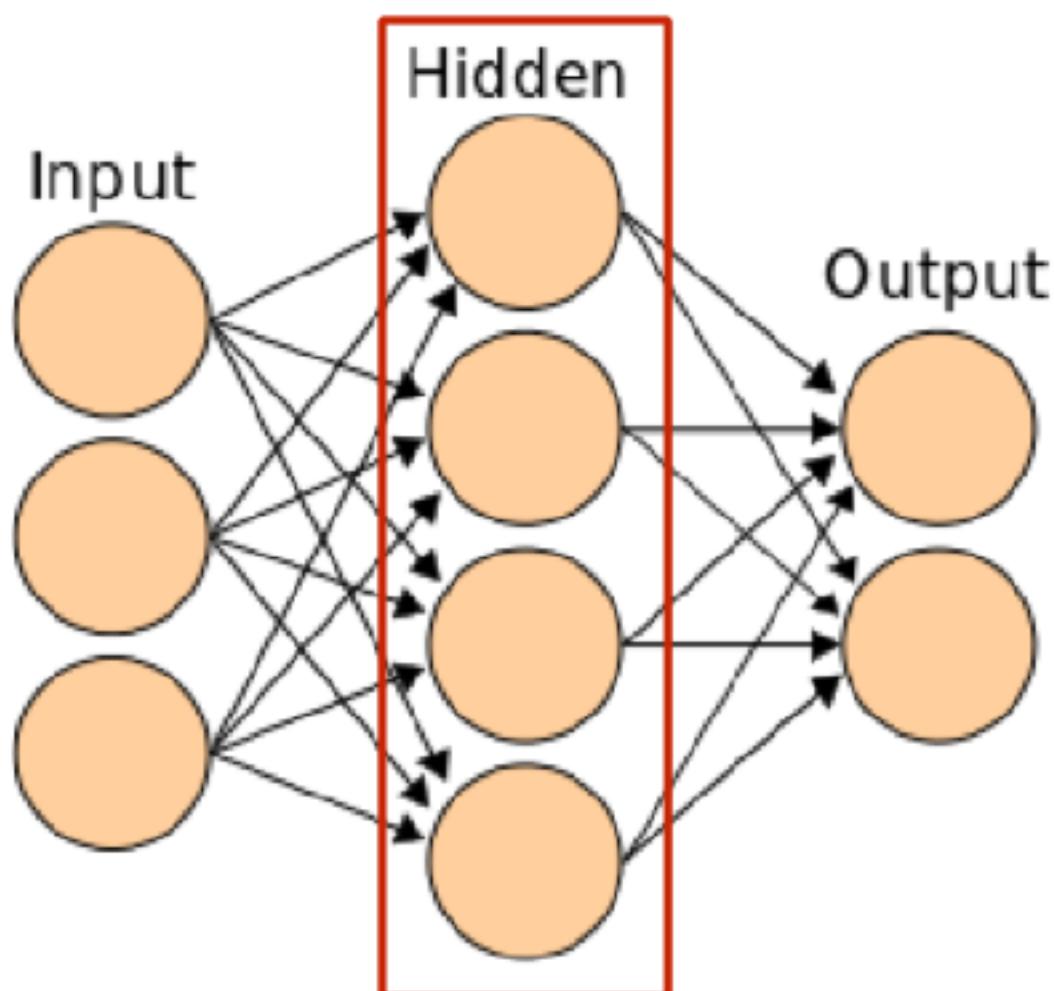
**RNN(Recurrent Neural Network)**은 Layer 내 연결이 없던 기존 Neural Network의 구조를 변형한 구조

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

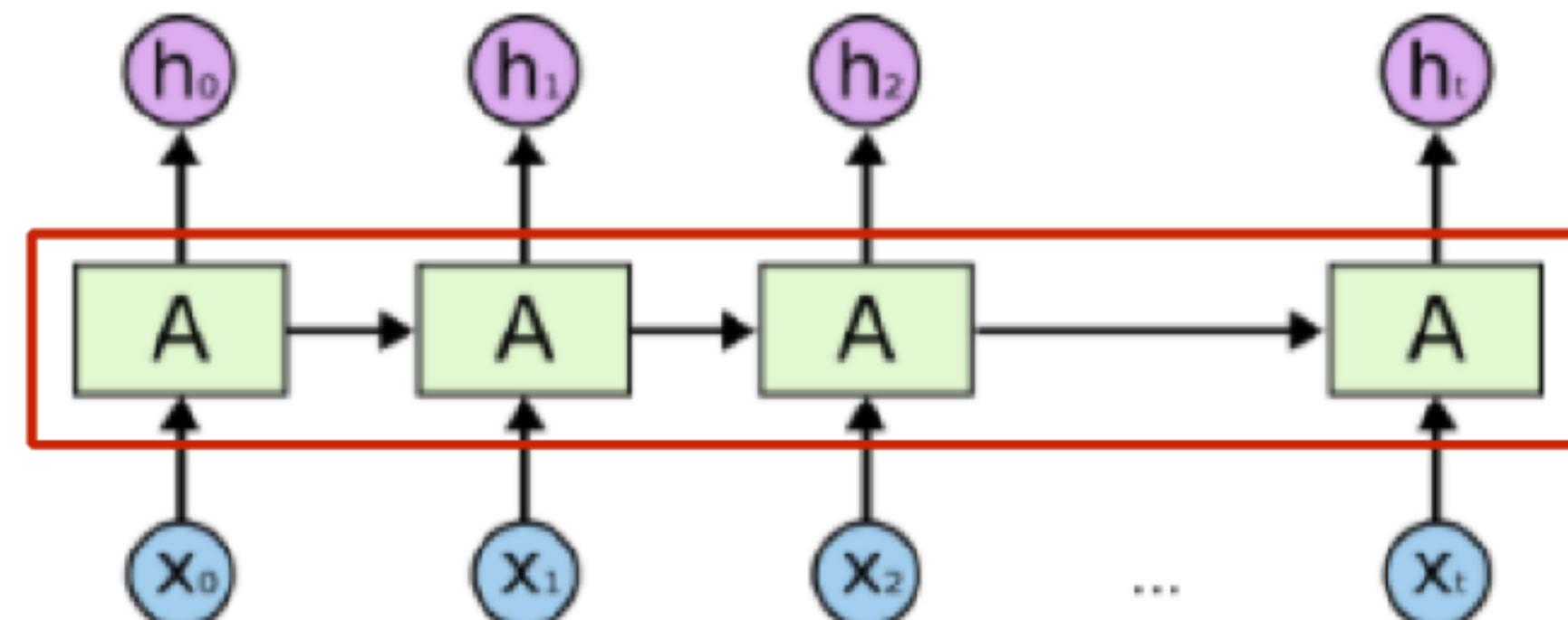
## <Artificial Neural Network>

- No interaction in same layer



## <Recurrent Neural Network>

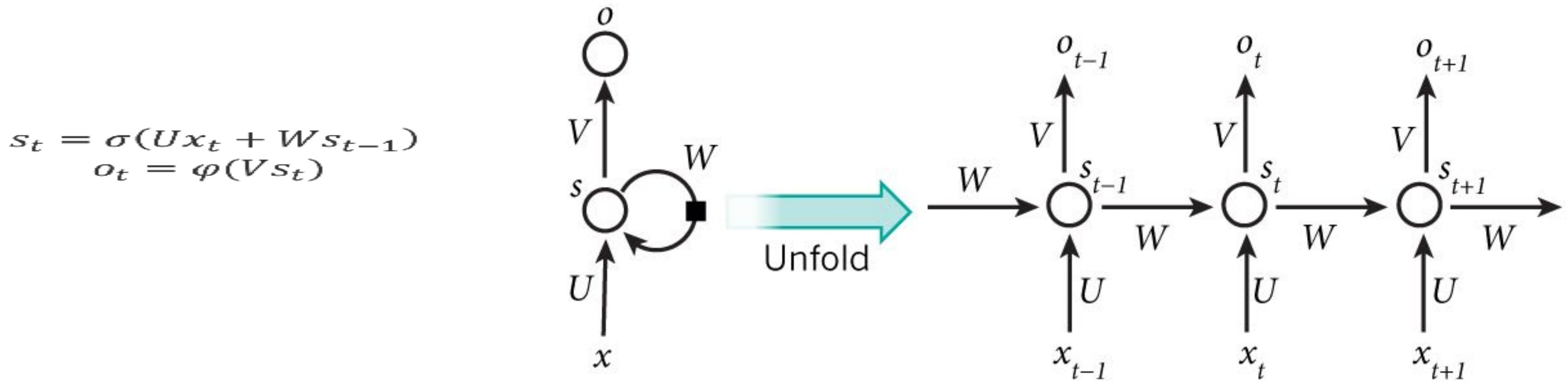
- 계산과정에서 Input Sequence가 고려됨
- Input과 이전 Input의 계산 결과값을 동시에 고려해가며 Output 계산
- Input-Hidden, Hidden-Hidden, Hidden-Output Weight **Sharing!**



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)  $\sigma$  와  $\varphi$  를 적절히 선정

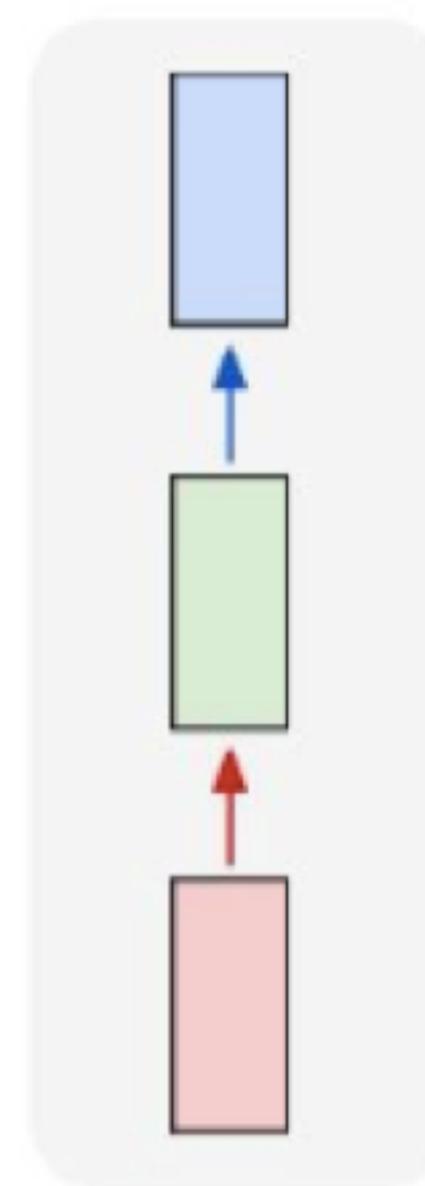


<basic RNN model>

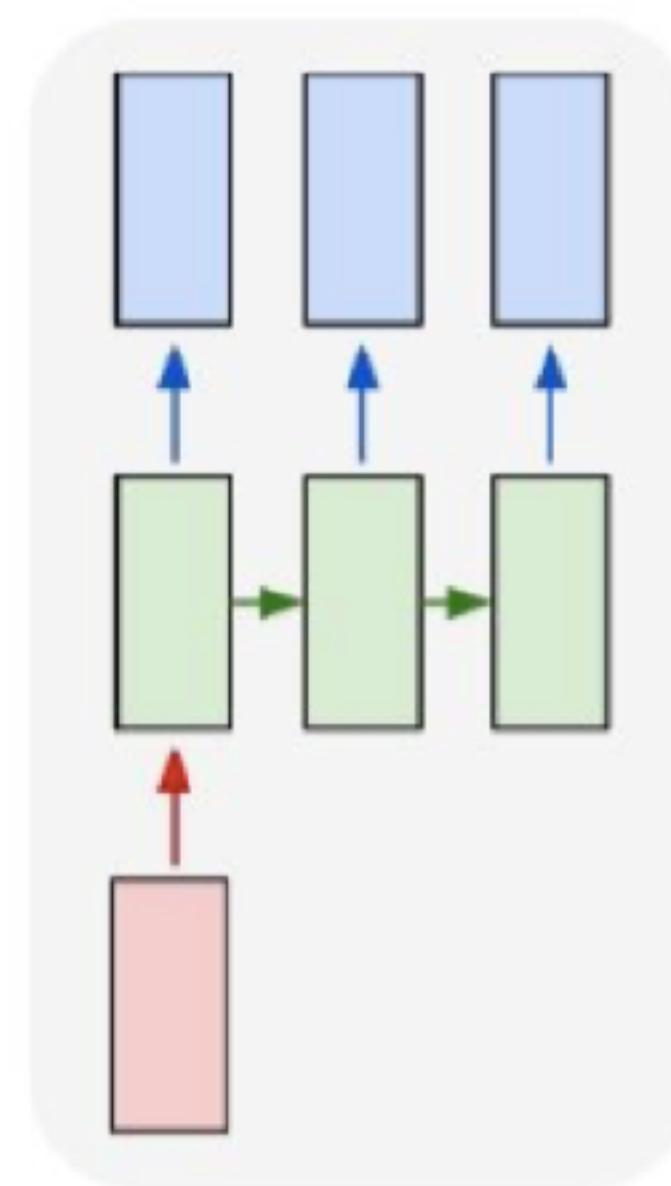
# Different I/O Structure of RNN

RNN은 원하는 Input/Output을 다양하게 하여 모델 선택 가능

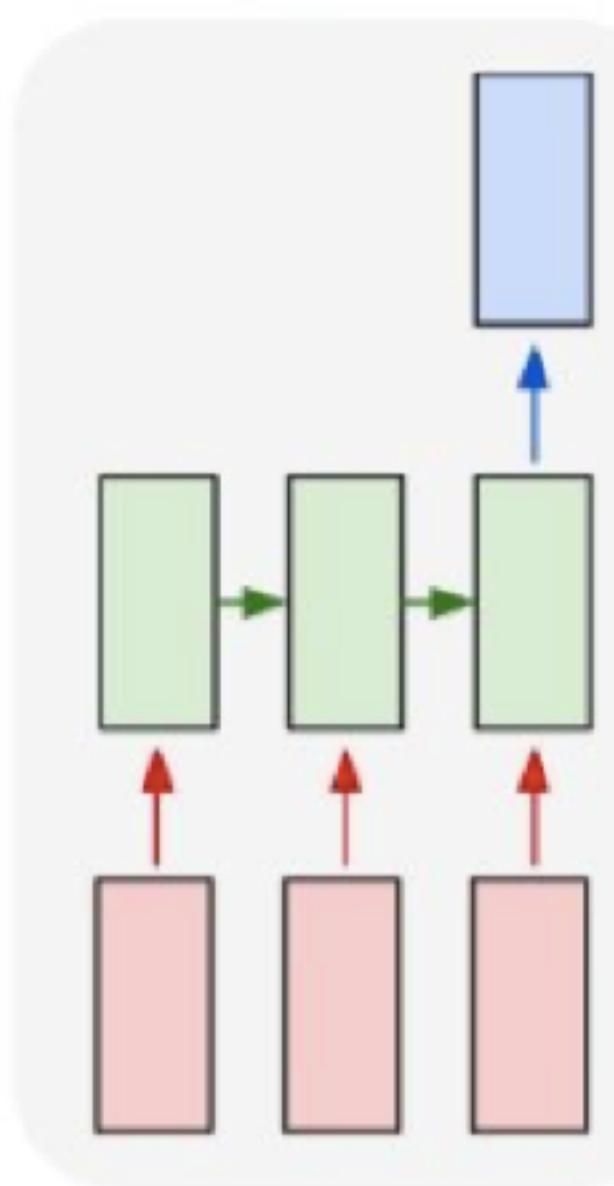
one to one



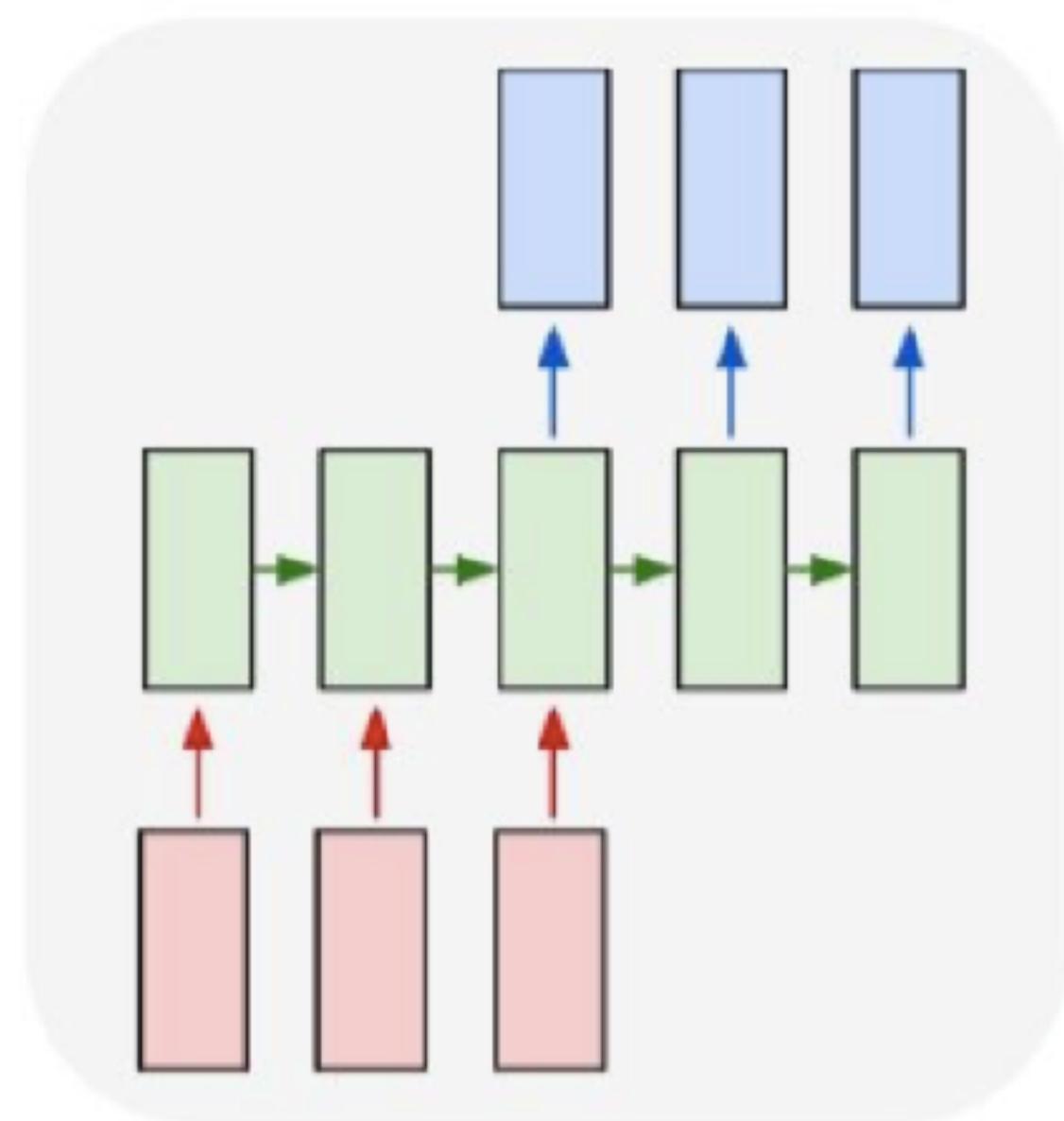
one to many



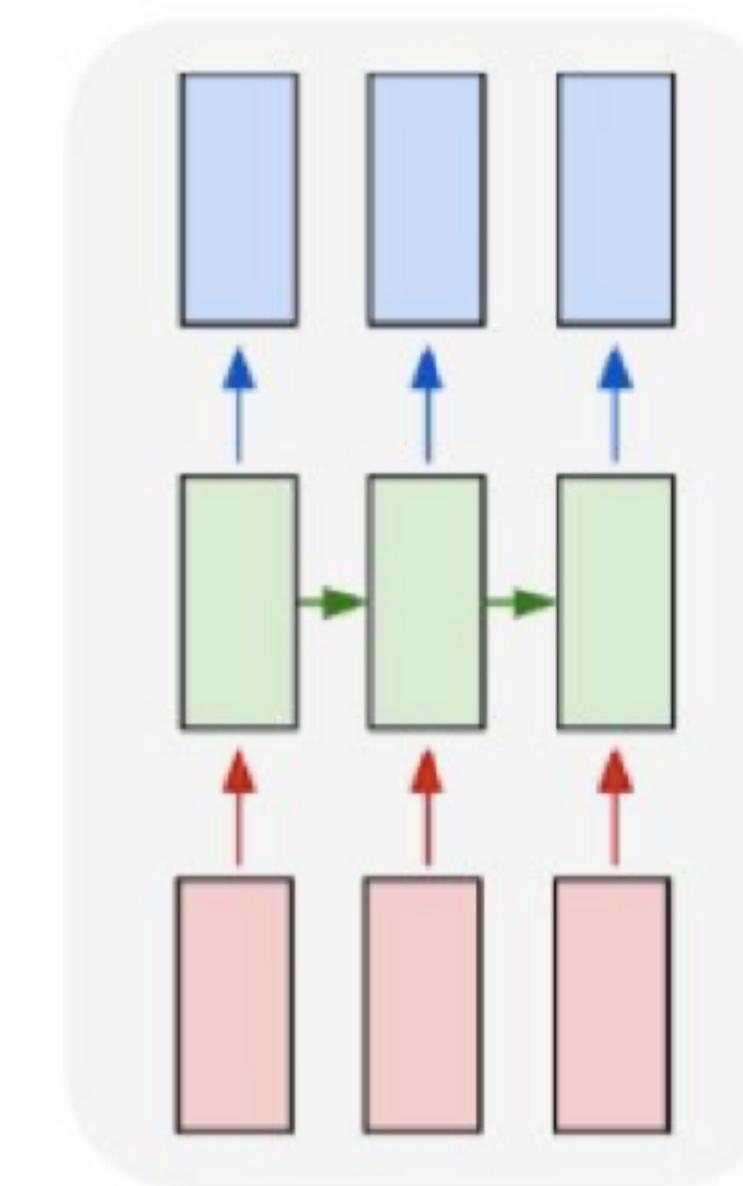
many to one



many to many



many to many



Vanilla Neural Networks

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

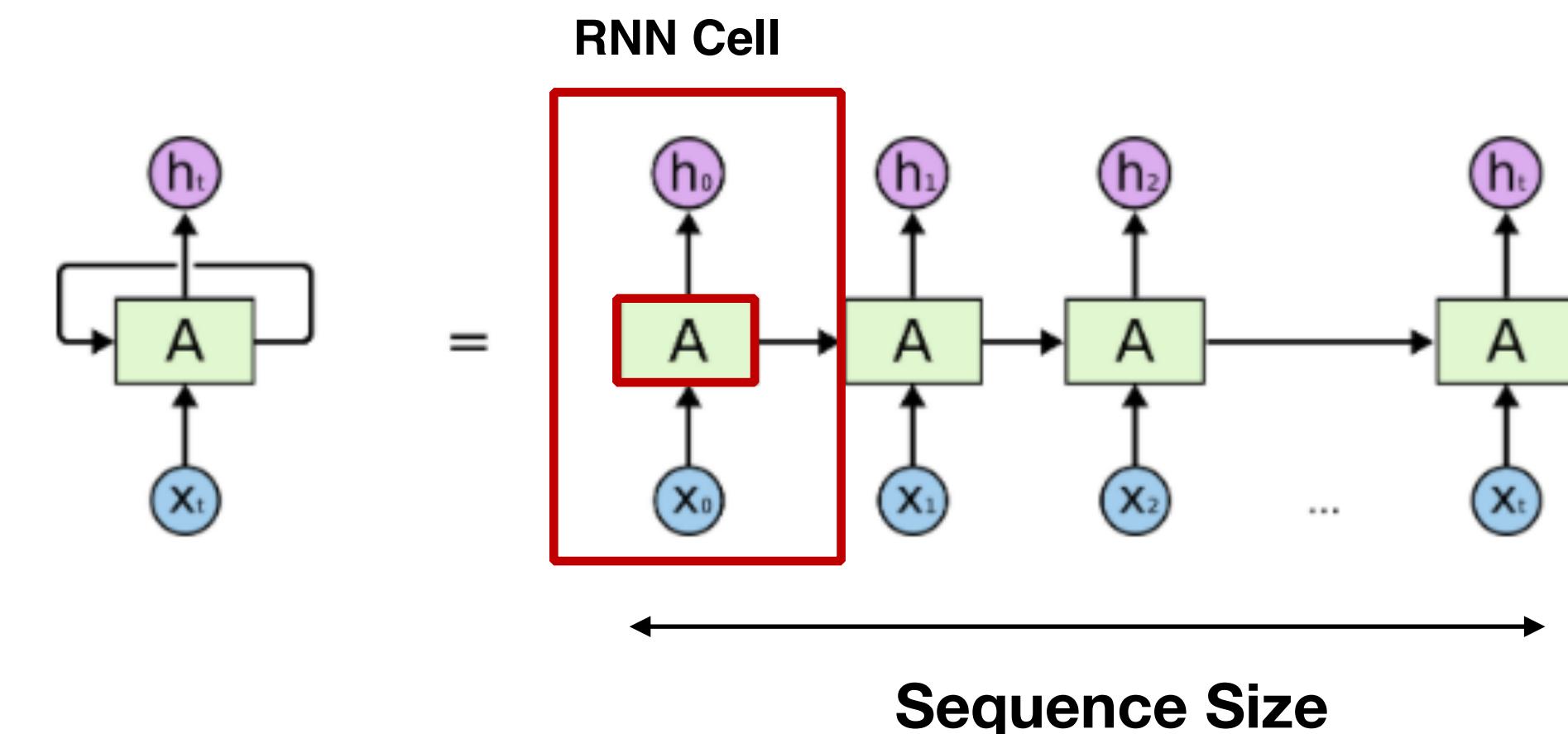
e.g. **Sentiment Classification**  
sequence of words -> sentiment

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

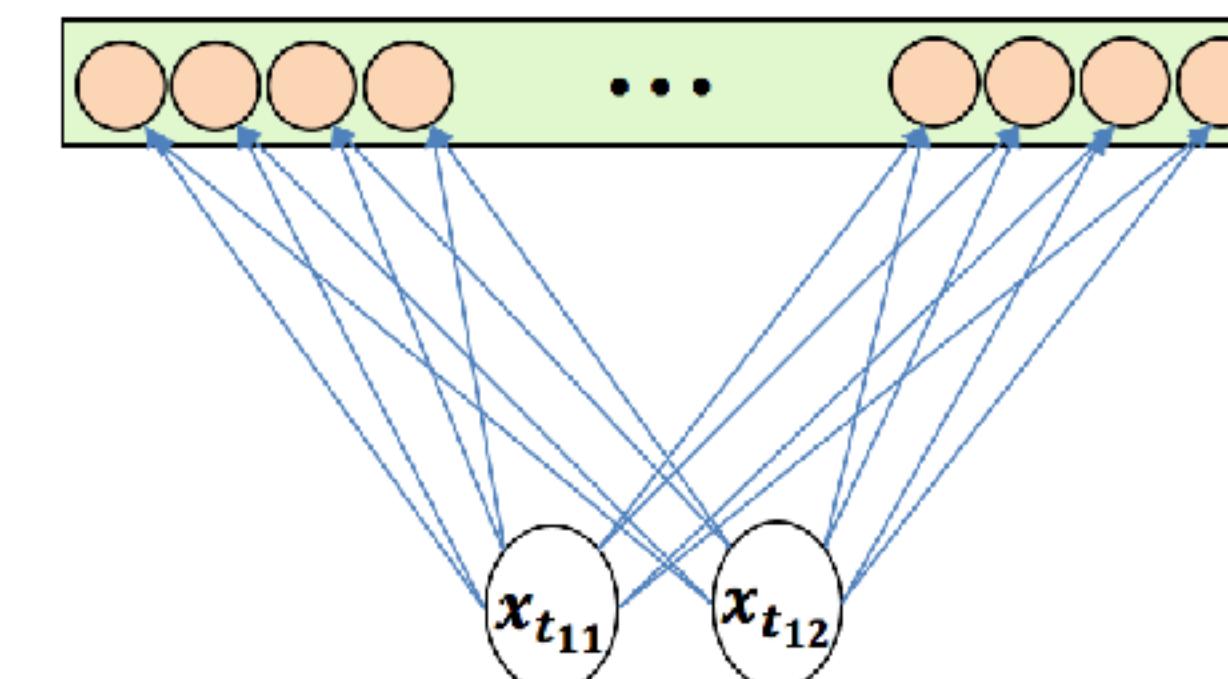
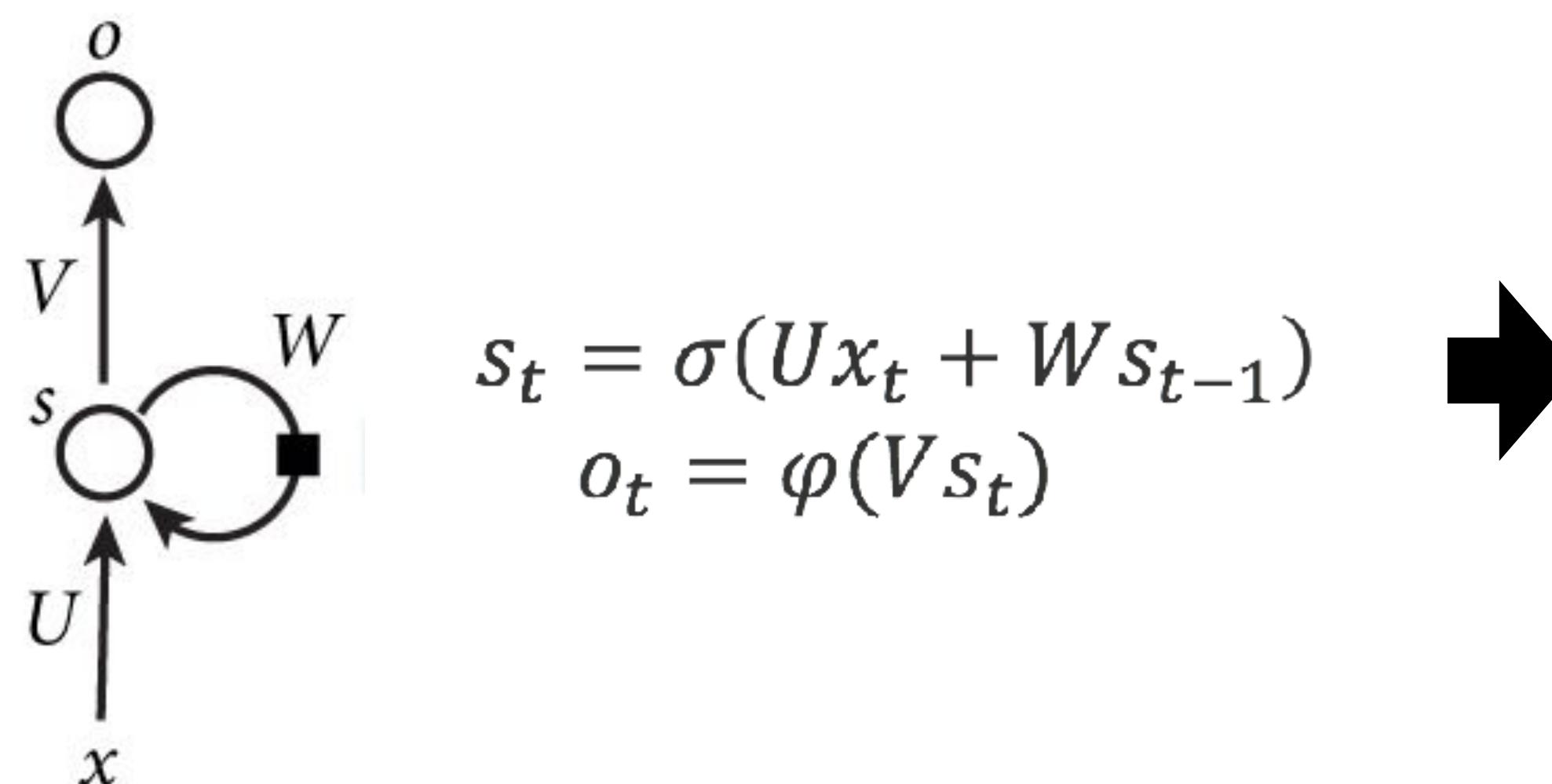
e.g. **Video classification on frame level**

# 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

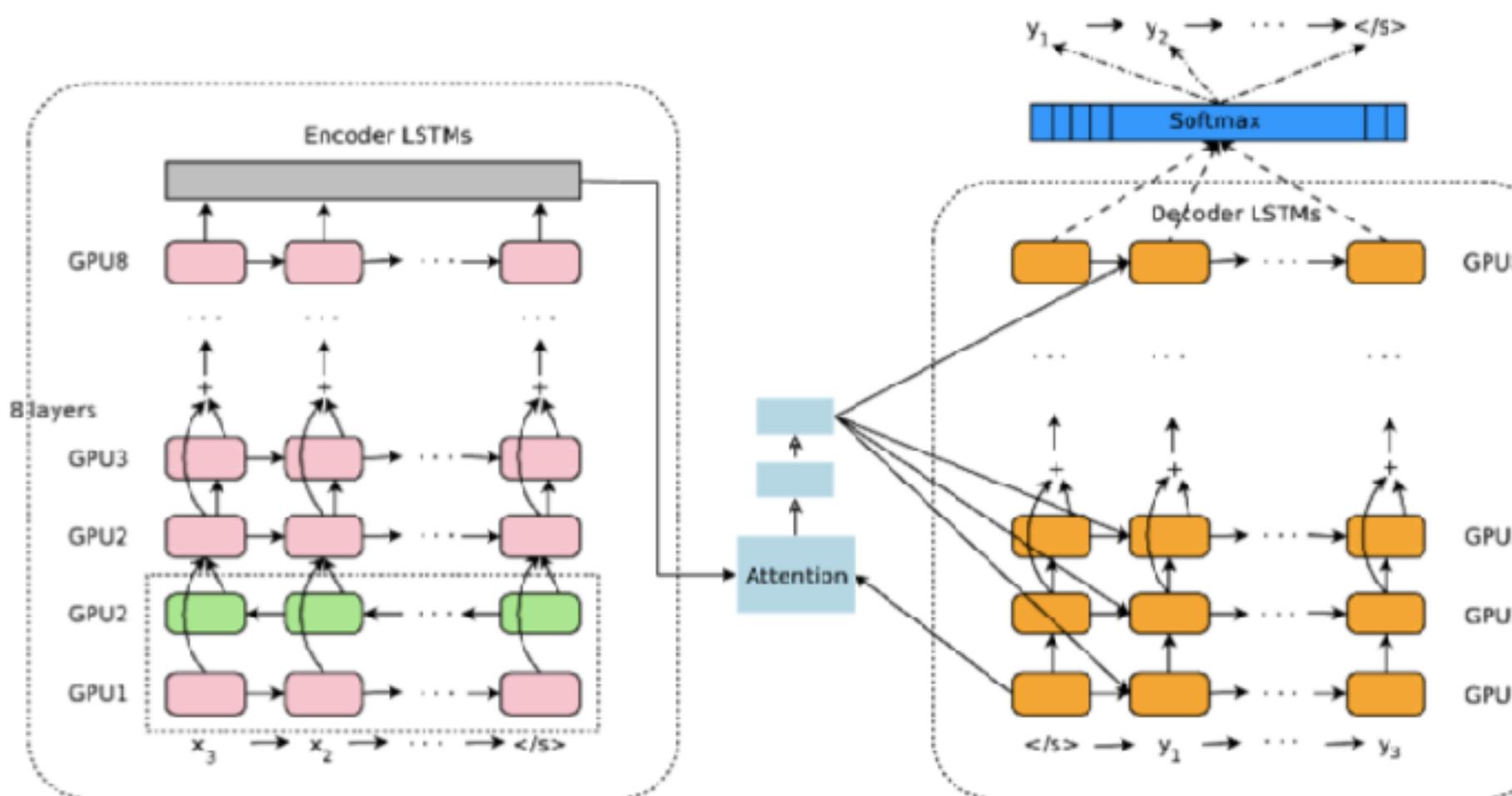
Recurrent Neural Network는 Sequential Data를 다루는 다양한 분야에 적용되고 있음

## <Translator>



Google  
Translate

Break through language barriers.



## <image Captioning>

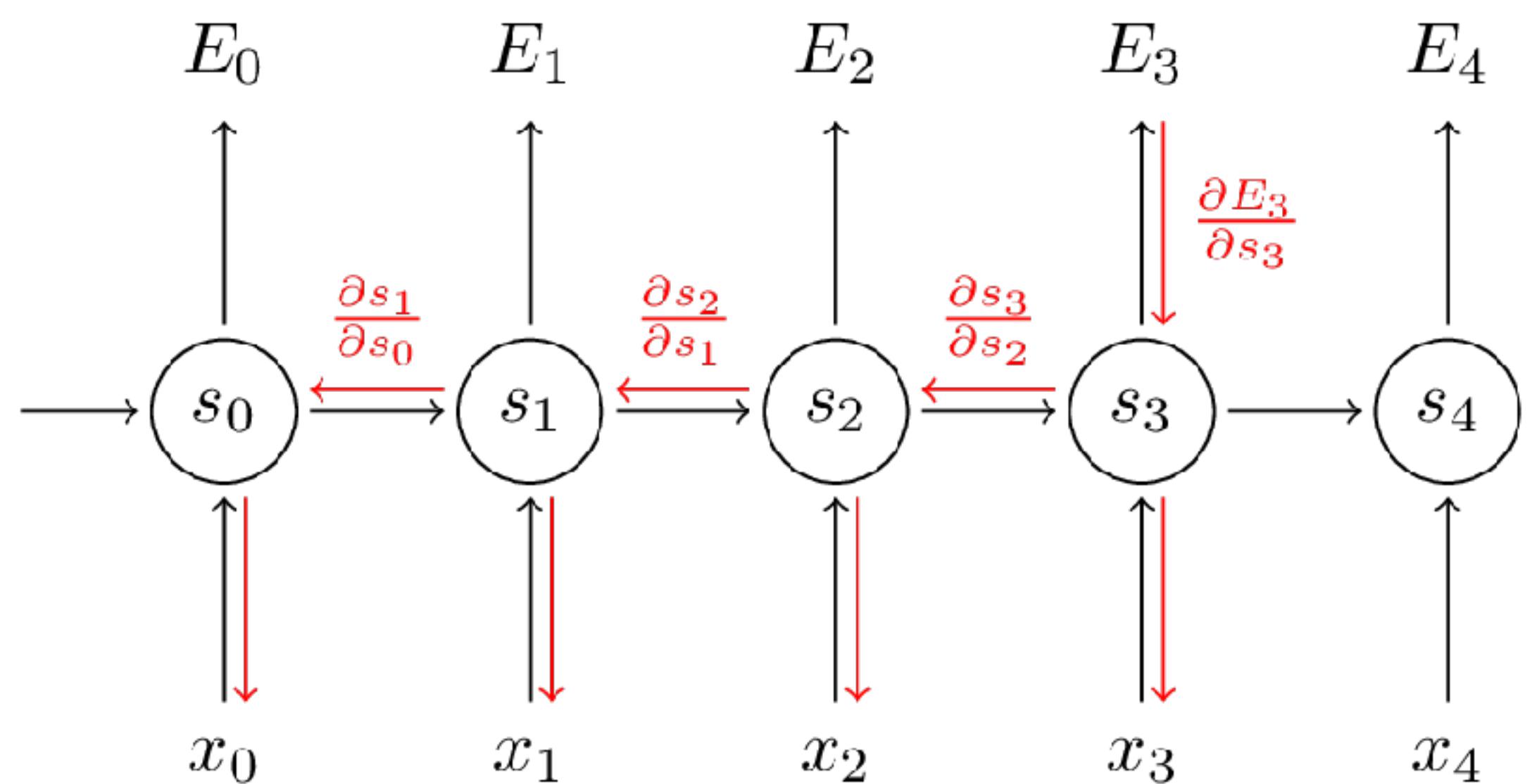


# LSTM & GRU

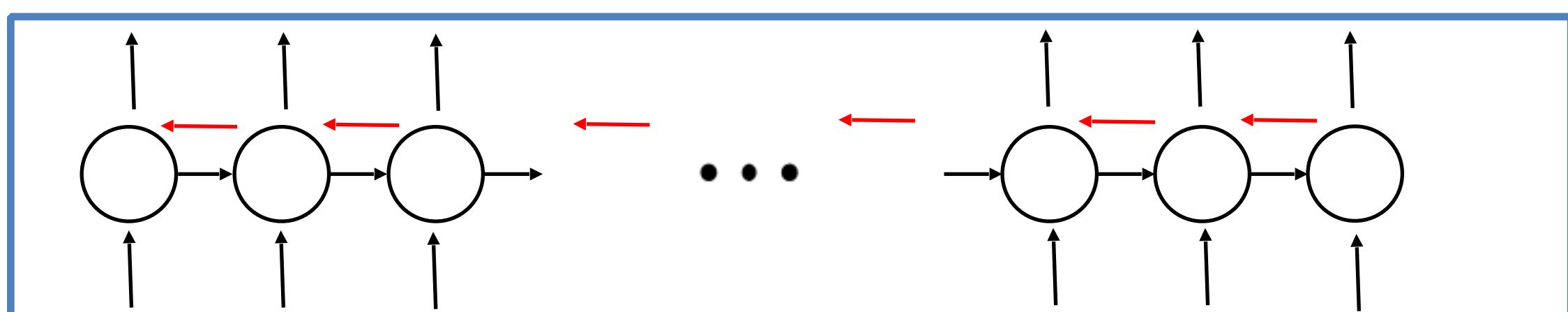
# Gradient Vanishing in RNN

- ▣ 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

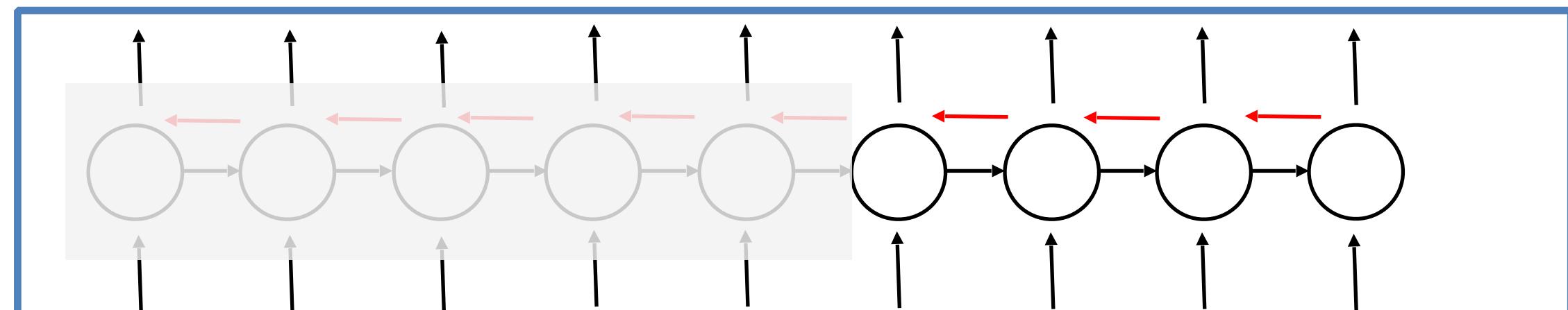
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$



<Long Term Memory>



<Short Term Memory>



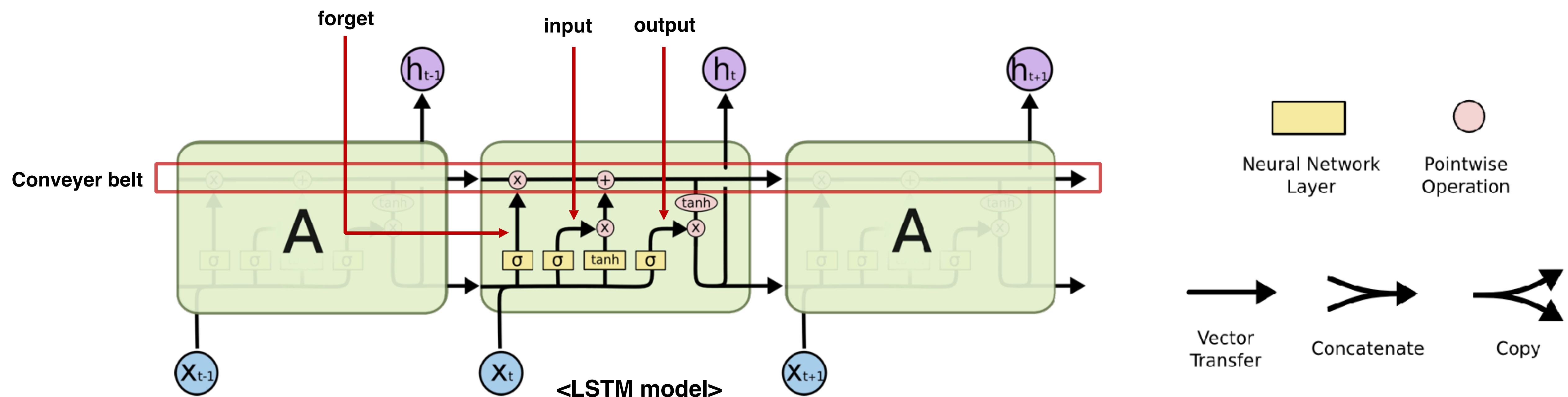
# Long Short Term Memory

□ 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

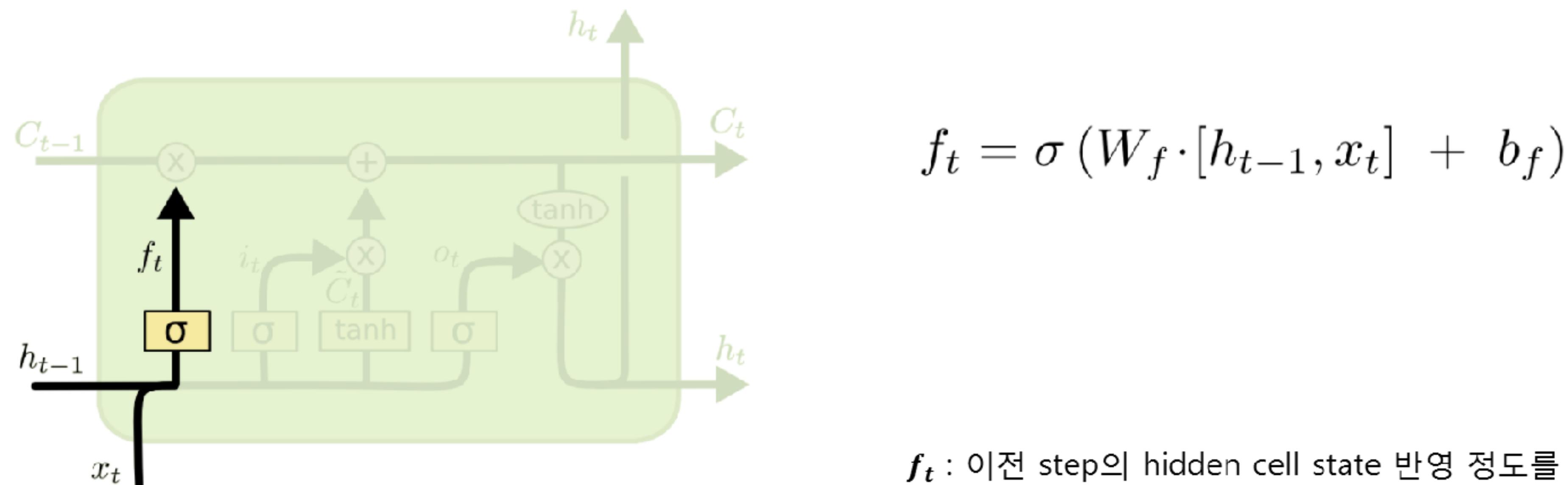
□ Conveyer belt에서 hidden state를 +연산자를 통해 갱신하여, gradient 전파에 용이하도록 modeling



# 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

$\sigma$  : activation function (sigmoid function)

$h_{t-1}$  : 이전 step의 output

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

$b_f$  : bias value

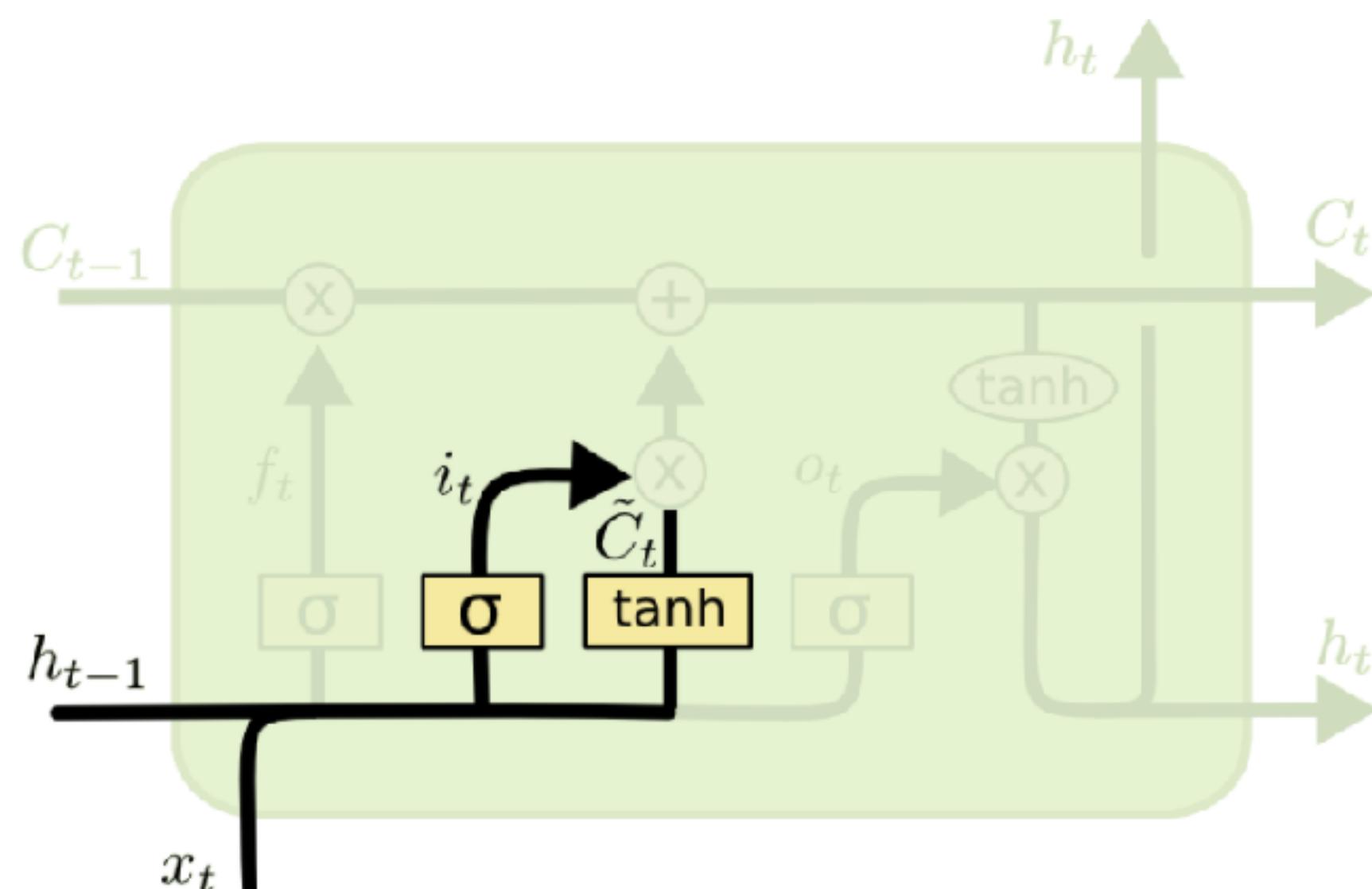
## LSTM step.2

### □ 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

$\sigma$  : activation function (sigmoid function)

$h_{t-1}$  : 이전 step의 output

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

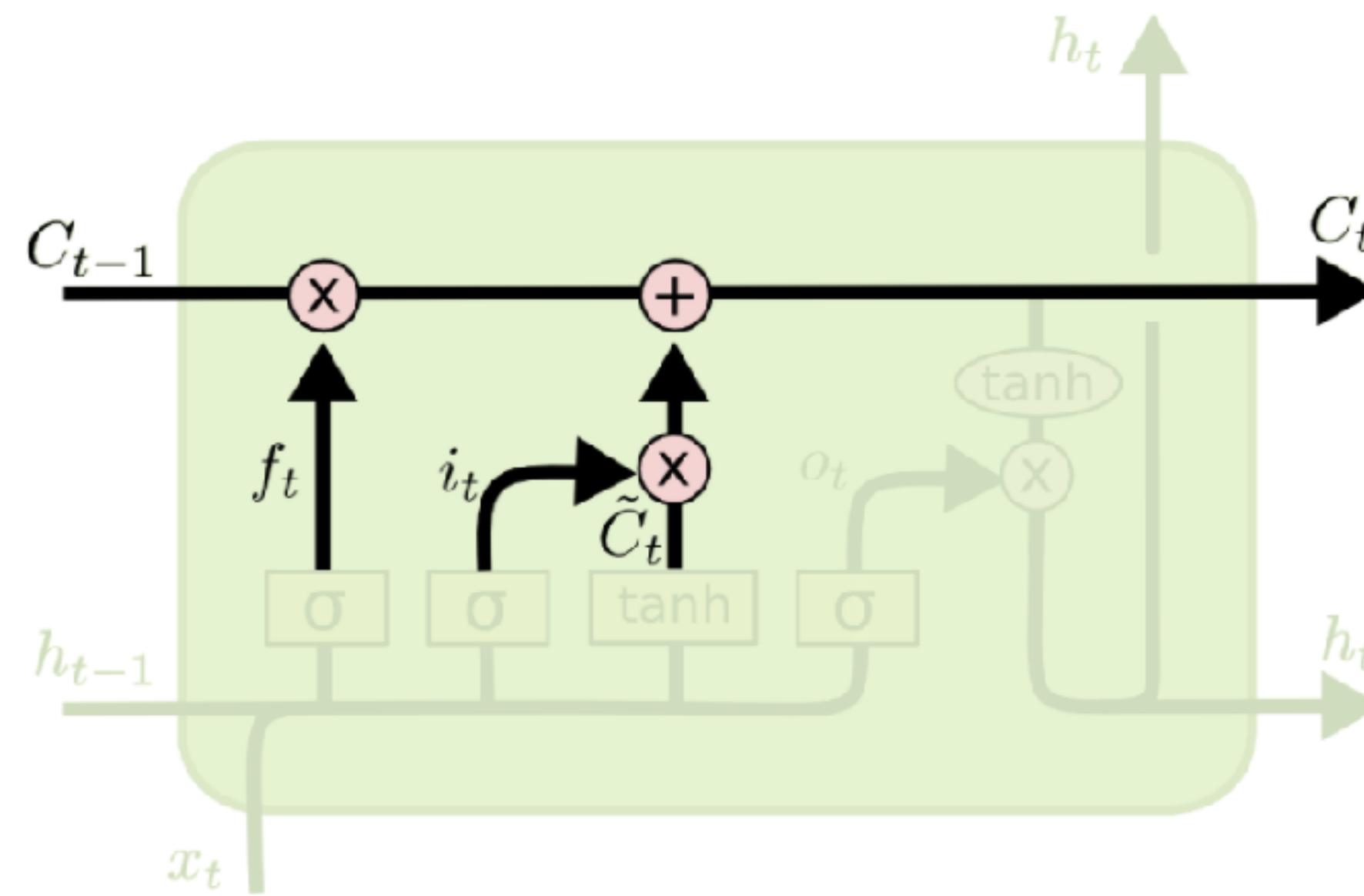
$\tilde{C}_t$  : 현재 step의 input과  $h_{t-1}$  를 통해 예측된 cell state

$b_{i,C}$  : 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



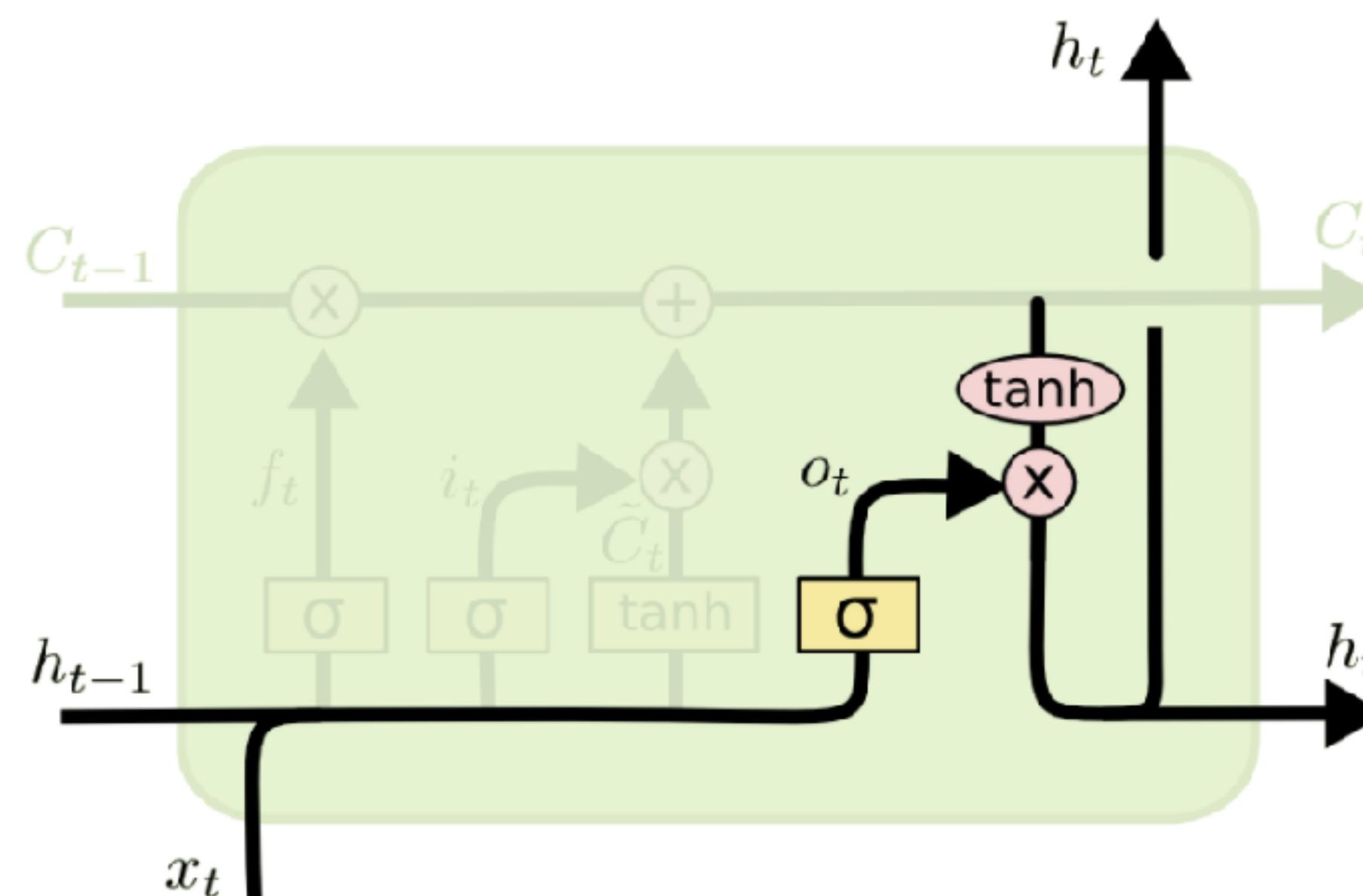
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

$f_t$  : 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

$\sigma$  : activation function (sigmoid function)

$h_{t-1}$  : 이전 step의 output

$C_t$  : 새롭게 구해진 현재 step의 hidden cell state

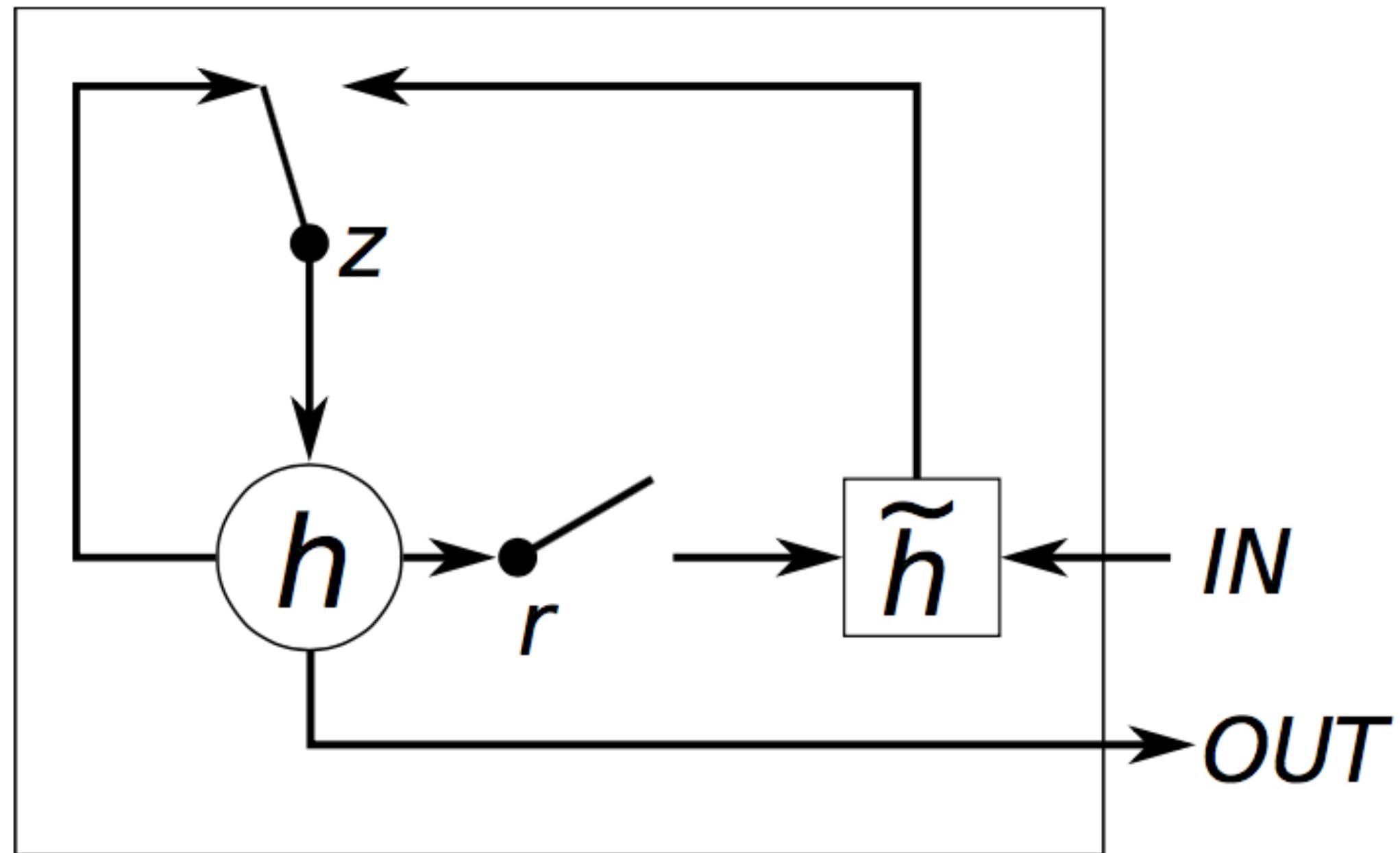
$h_t$  : 새로운 output

$b_o$  : 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>

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

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

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

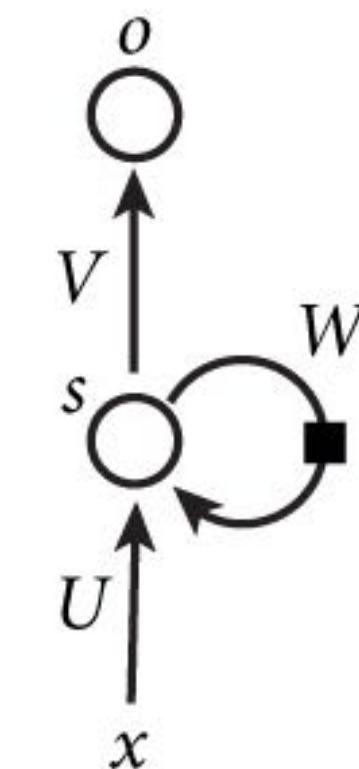
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

# Recurrent Models in Tensorflow

# RNN structure

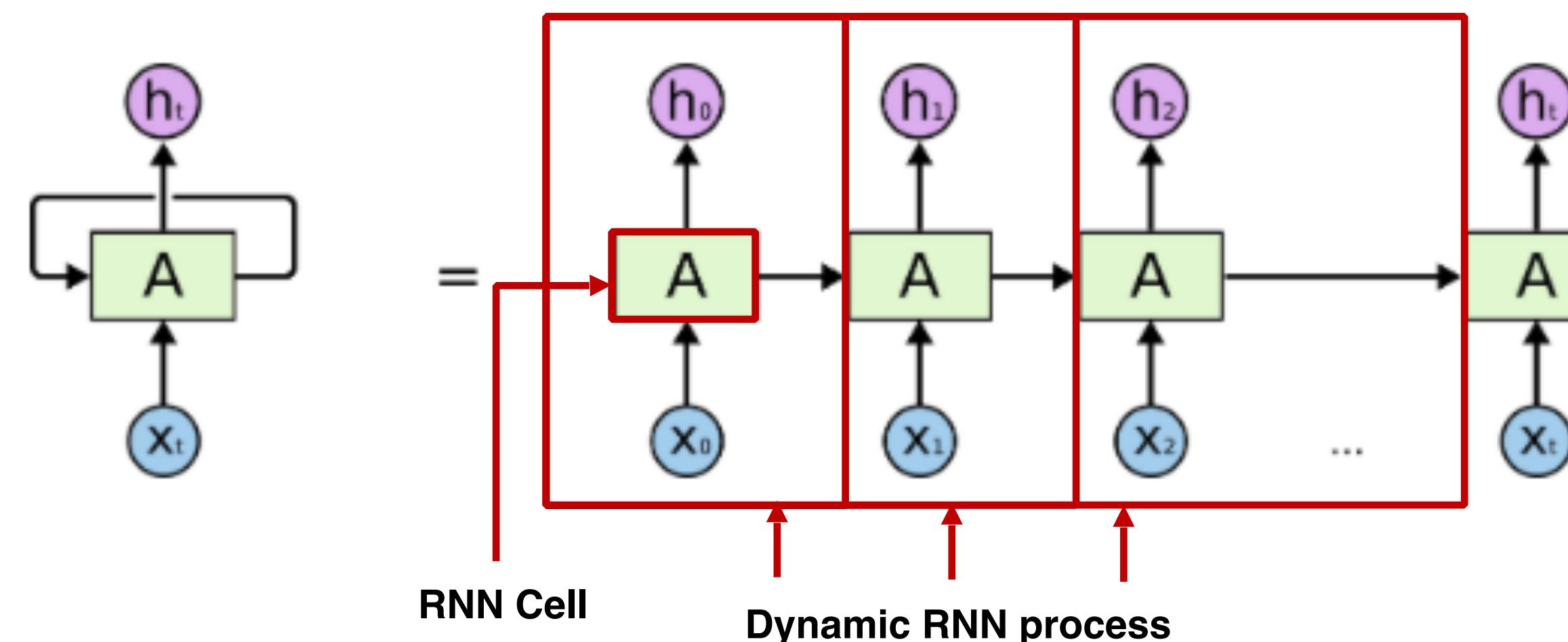
## □ RNN Cell : 기존의 Neural Network에서 layer의 역할을 수행하는 Cell

- 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

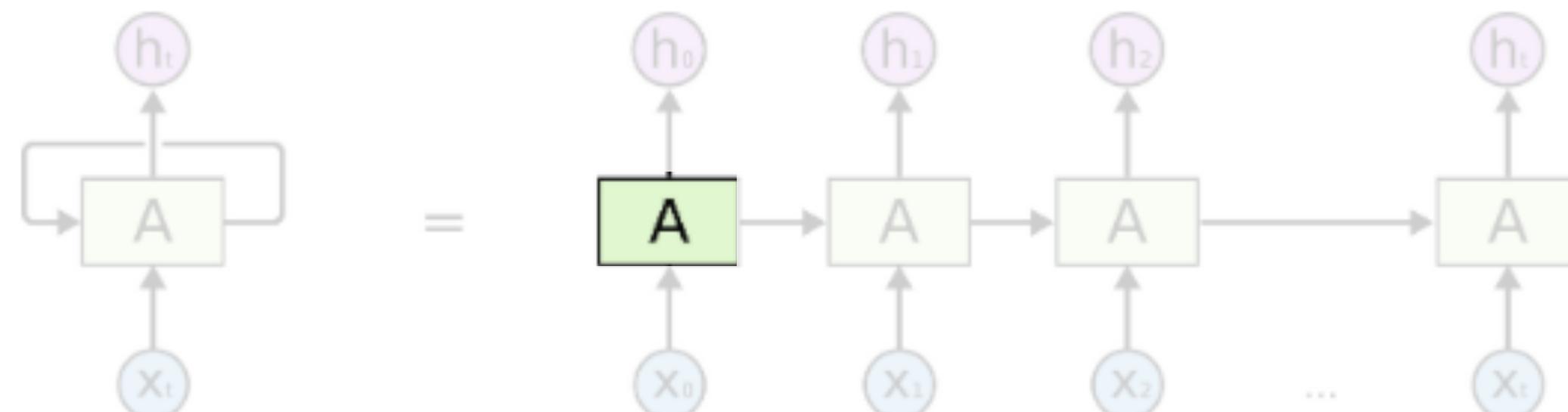
## □ 기본 모델의 단위인 BasicRNNCell을 생성하기 위한 클래스

### □ \_\_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

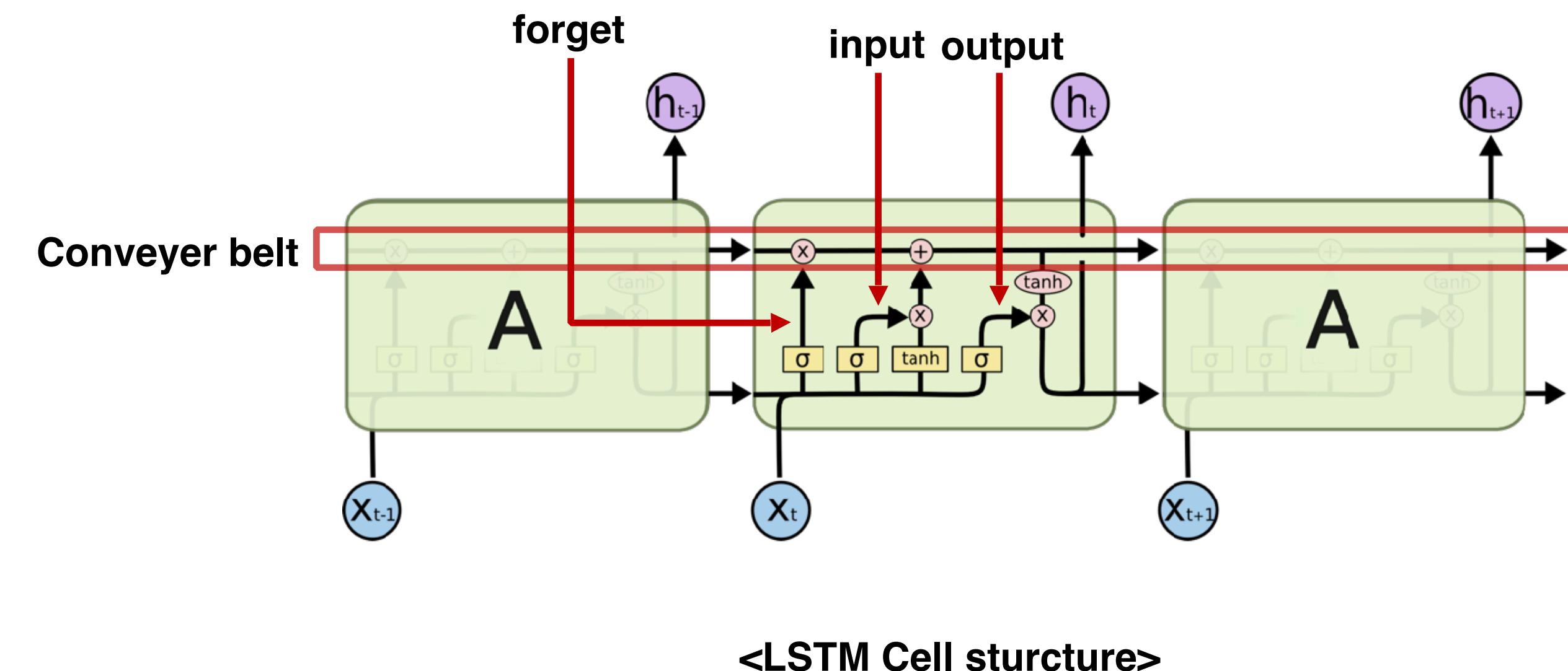
### □ \_\_init\_\_

- 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\_\_

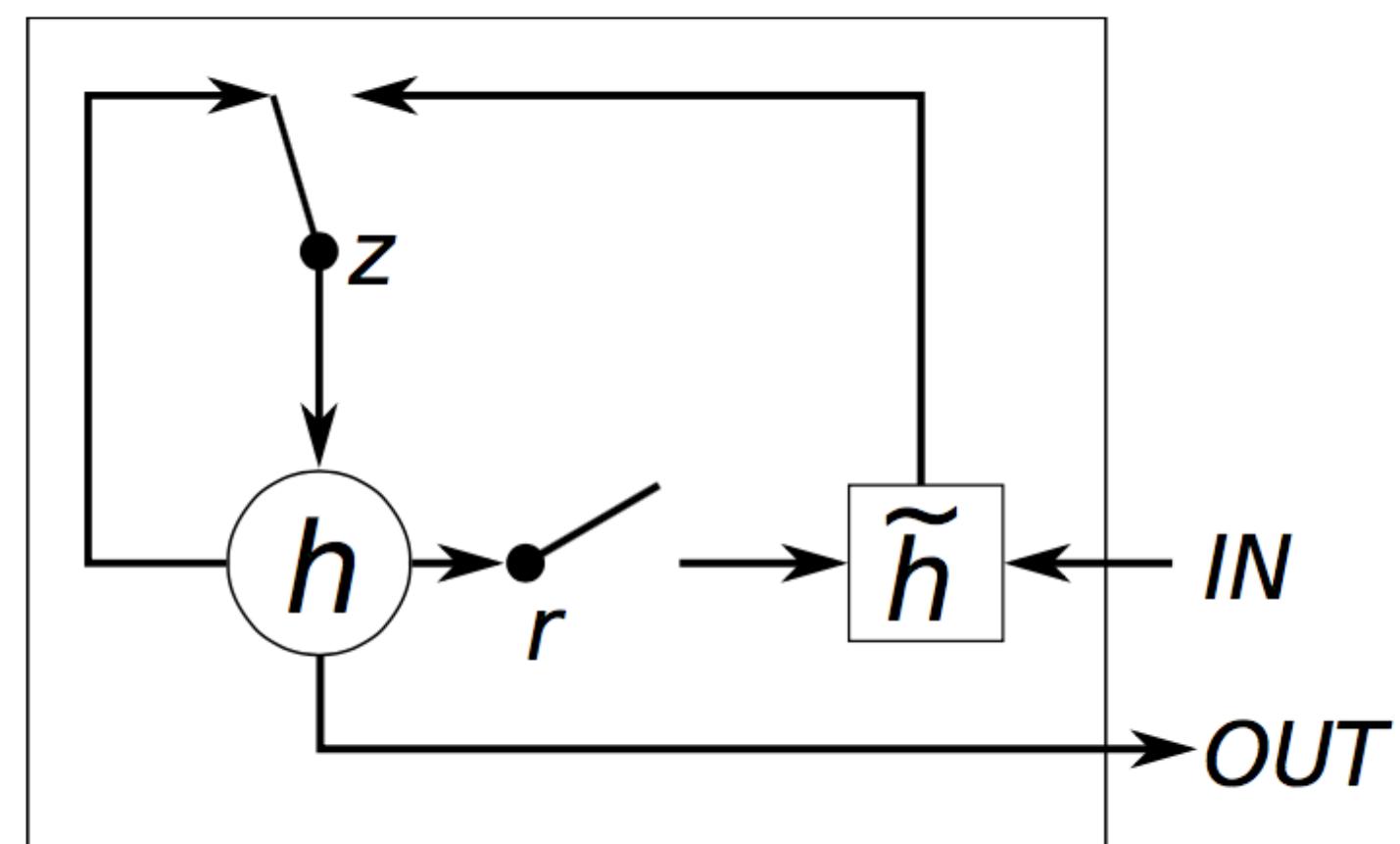
- num\_units : hidden\_dimension의 값을 입력
- activation : activation function으로 사용할 함수를 지정(default : tanh)
- reuse : 가중치 U, W 변수의 재사용 여부를 결정
- kernel(bias)\_initializer : 가중치의 초기값을 설정하기 위한 Initializer를 별도로 선정 가능

(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  
)
```



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

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

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

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

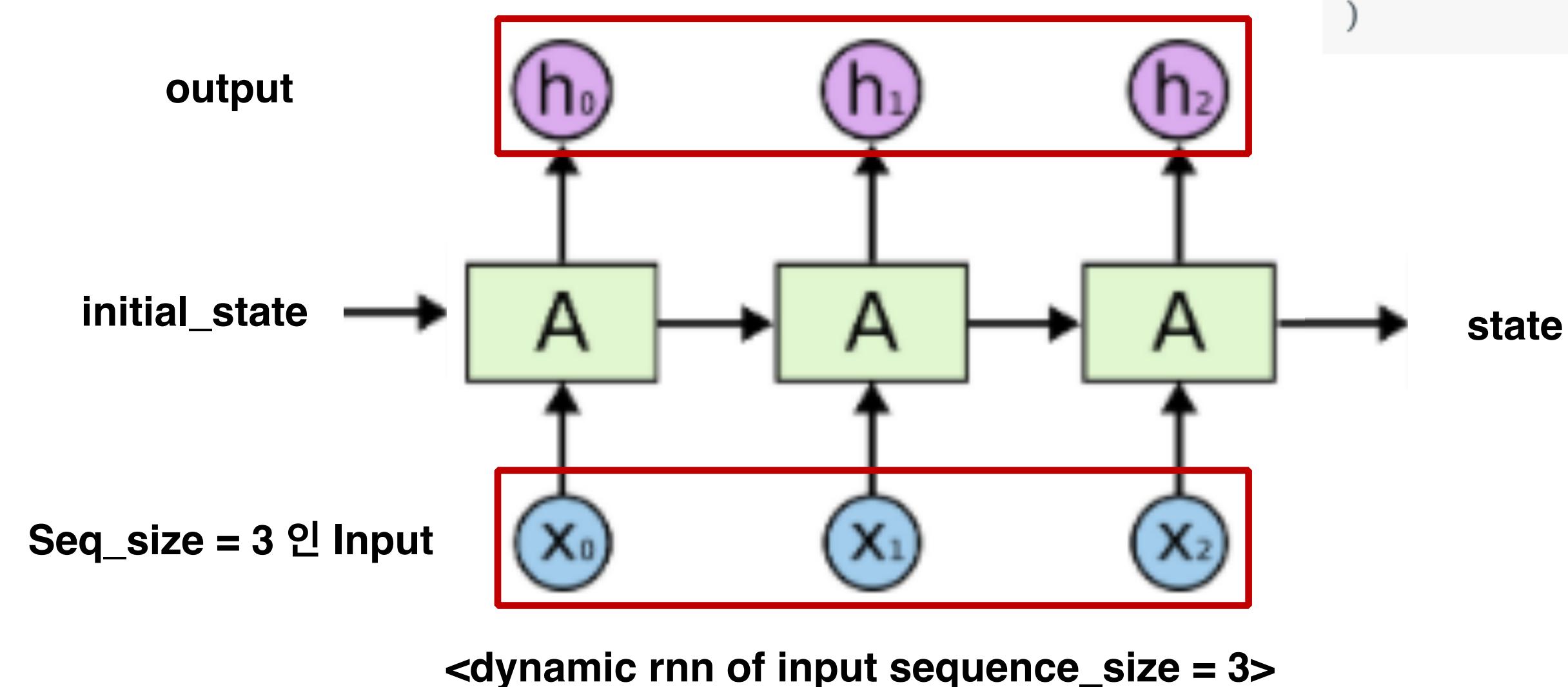
## tf.nn.dynamic\_rnn

- 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

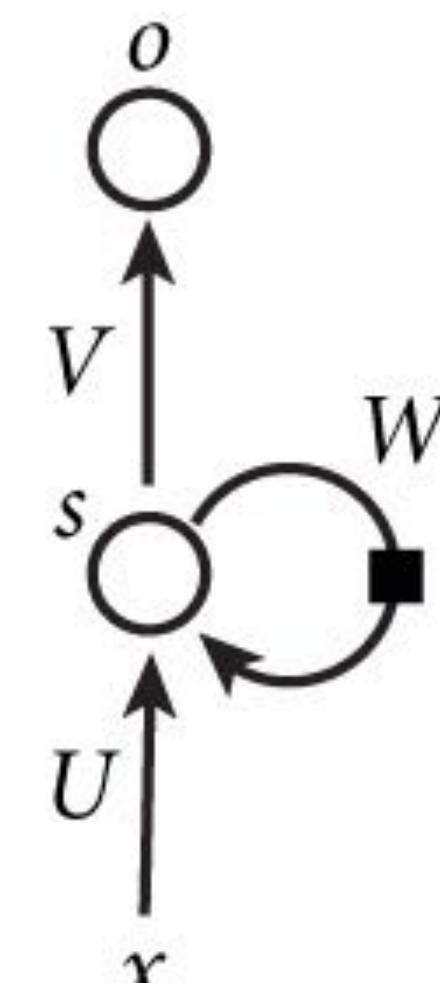
```
dynamic_rnn(  
    cell,  
    inputs,  
    sequence_length=None,  
    initial_state=None,  
    dtype=None,  
    parallel_iterations=None,  
    swap_memory=False,  
    time_major=False,  
    scope=None  
)
```

## tf.nn.dynamic\_rnn

- 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]

eXem

## 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

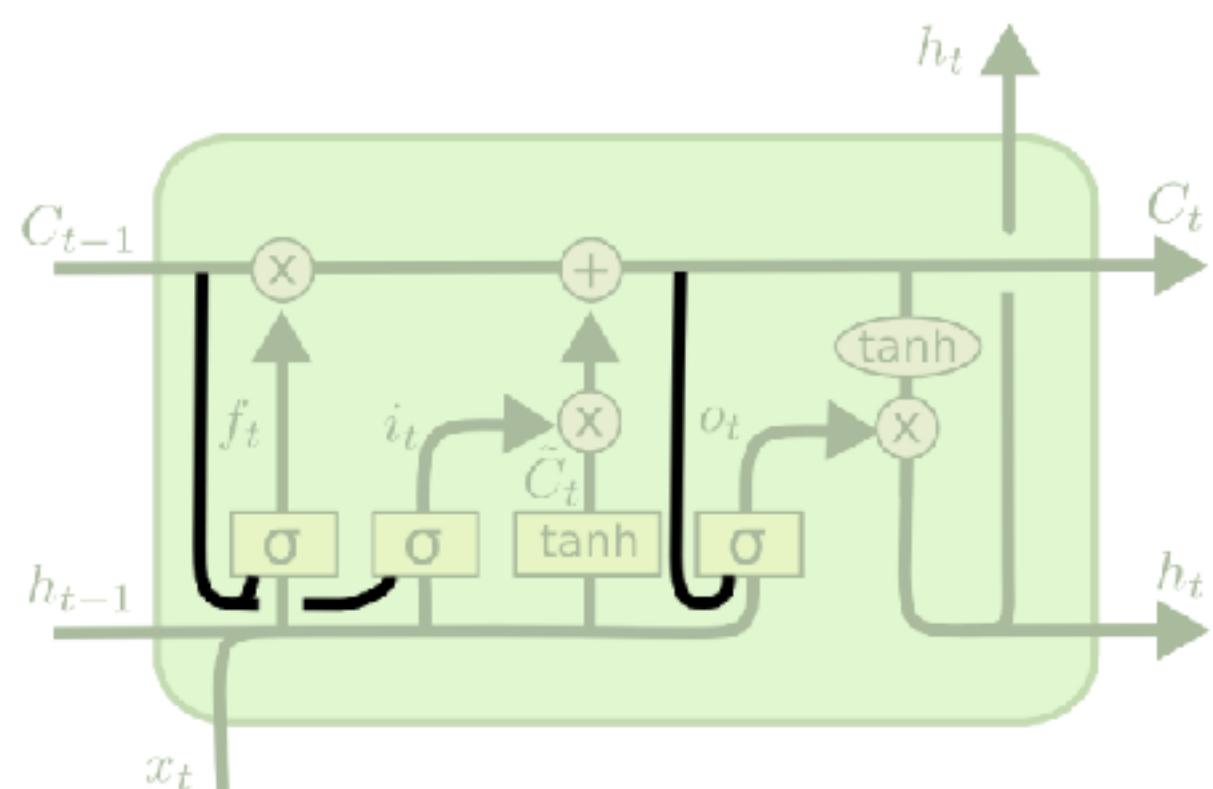
□ \_\_init\_\_

- `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  
)
```

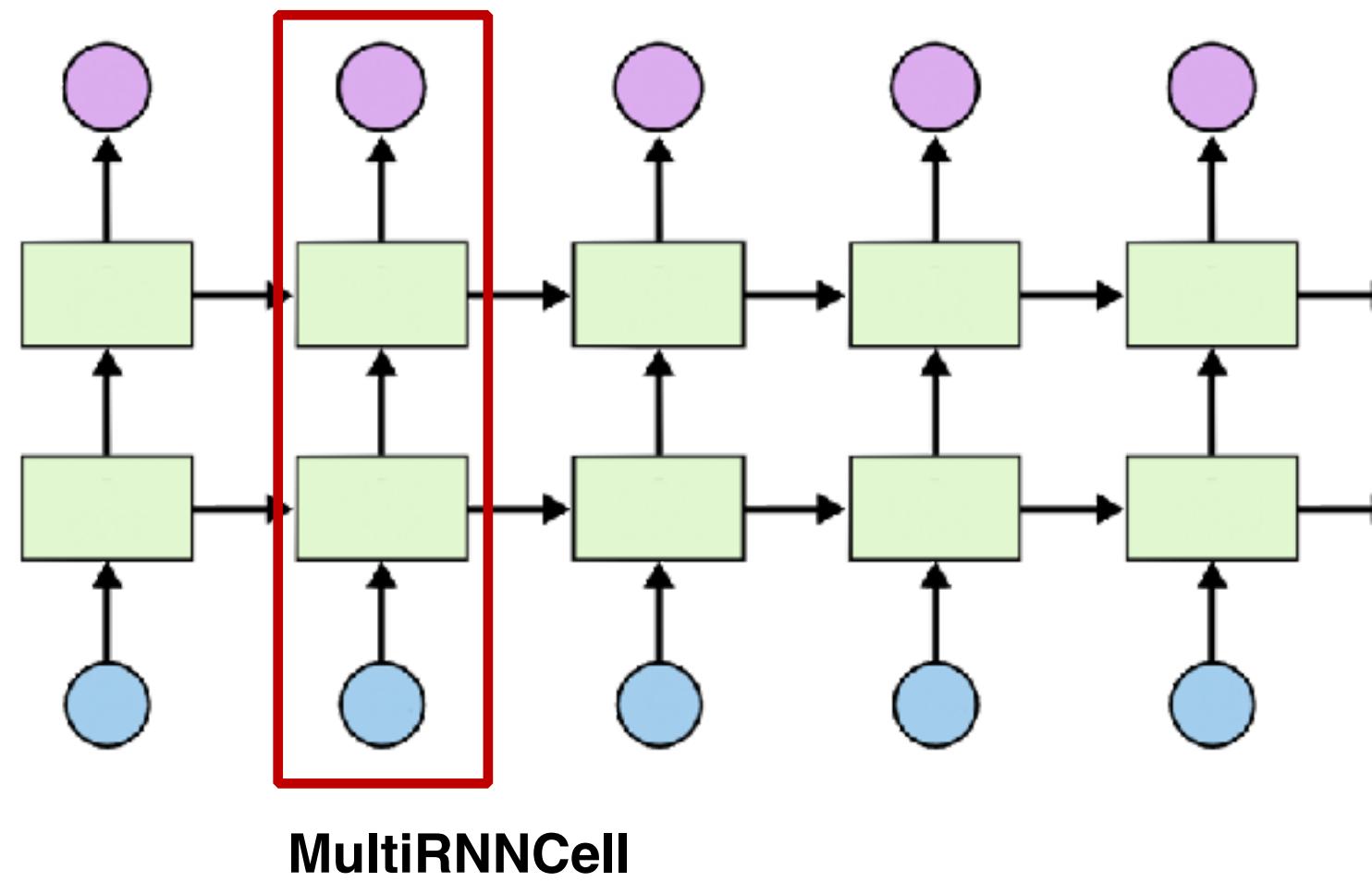


$$\begin{aligned} f_t &= \sigma(W_f \cdot [C_{t-1}, h_{t-1}, x_t] + b_f) \\ i_t &= \sigma(W_i \cdot [C_{t-1}, h_{t-1}, x_t] + b_i) \\ o_t &= \sigma(W_o \cdot [C_t, h_{t-1}, x_t] + b_o) \end{aligned}$$

<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

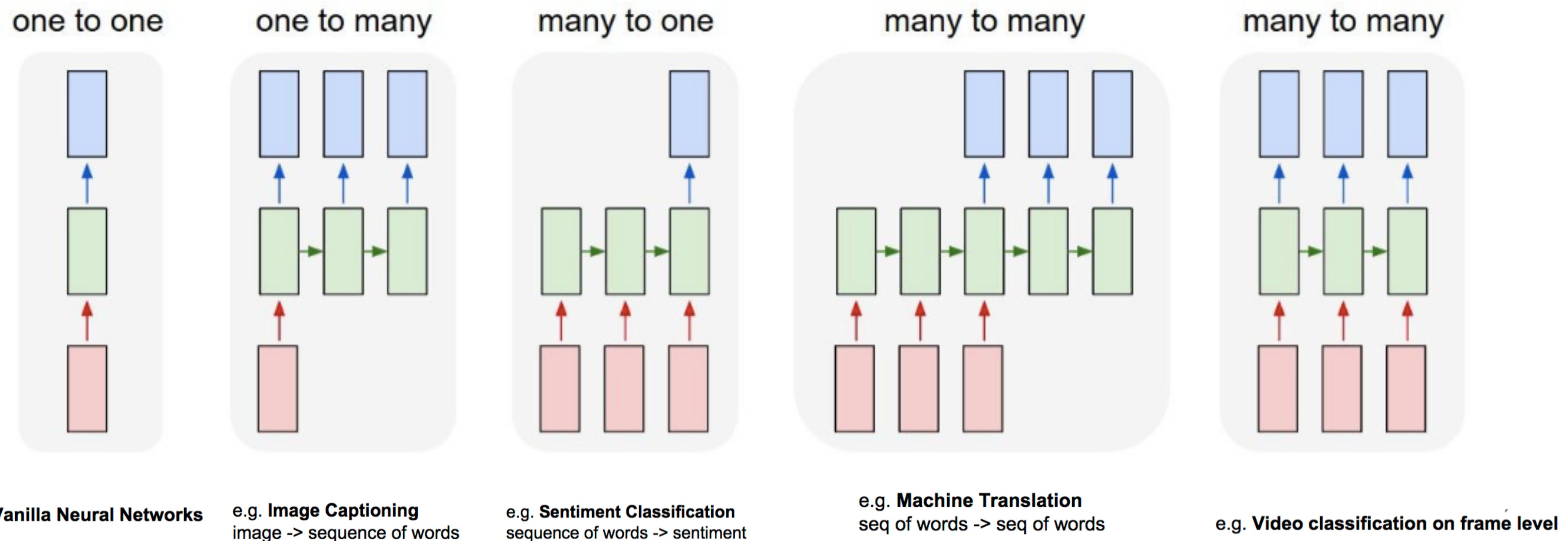
```
#1st recurrent layer
cell=tf.nn.rnn_cell.LSTMCell(num_units=self.hidden_dim, initializer = tf.contrib.layers.xavier_initializer(uniform=False), state_is_tuple=False)

#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



# Management of Variables

# Model Implementation using Class

## □ 모델이 복잡해 질수록, 모델 결과와 파라미터 등에 자유롭게 접근하는 것이 힘듦

(스크립트 식으로 구현해놓았다가, 갑자기 특정 파라미터가 필요하다면? 어떤 결과를 가지고 분석하고자 한다면?)

## □ 효과적인 모델 관리를 위해 Class 형식으로 모델을 구현하고 관리함

```
-def __init__(self, sess, x_data, y_data):
    print "Initialize New Model with whole data"
    self.sess = sess
    self.train_epoch = 0
    self.train_loss = []
    self.validation_loss = []
    self.min_valid_loss = 9999.999
    self.x_data = x_data
    self.y_data = y_data
    self.train_valid_test_split()
    print "Data is splitted into train, valid, test data"
    self.dir_LO = SAVE_DIRECTORY_LOSS_OUT
    self.dir_Paras = SAVE_DIRECTORY_PARAS
    self.dir_Checks = SAVE_DIRECTORY_CHECKS
```

학습, 테스트 데이터, 저장 경로 등  
모델 사용에 필요한 정보를 멤버 변수로 관리함

```
class Model(object):
    def __init__(self, sess): ...
    def _create_place_holders(self): ...
    def _inference(self): ...
    def _create_loss(self, loss_type='cross_entropy', regularization='L2'): ...
    #Types of optimizer = 'GD(Gradient Descent)', 'ADAM', and 'AdaDelta'
    def _create_optimizer(self, type_optimizer='GD', learning_rate = LEARNING_RATE, ra ...
    #Run Optimizer in order to minimize loss function and return its loss_value
    def batch_train(self, feed_dict): ...
    #return loss value of feed_dict input
    def batch_loss(self, feed_dict): ...
    #return output result of feed_dict input
    def batch_out(self, feed_dict): ...
    def get_data(self, X_data, Y_data): ...
    def train_valid_test_split(self): ...
    def run_train_epoch(self): ...
    def run_validation(self): ...
    def use_optimal_paras(self): ...
    def run_test(self, use_optimal_parameter=True): ...
    def build_graph(self): ...
    def initialize_variables(self): ...
    def update_optimal_variables(self): ...
    def save_loss_output(self): ...
    def save_paras(self): ...
    def save_checks(self): ...
    def run(self): ...
sess = tf.Session()
x_train, y_train, x_val, y_val, x_test, y_test = get_CIFAR10_data()
model1 = Model(sess, x_train, y_train, x_val, y_val, x_test, y_test)
model1.run()
```

모델 구축,

모델 학습 및 테스트에 필요한 함수 등을  
멤버 함수로 구현하고

클래스 내부에서 멤버 변수에 자유롭게 접근

# Model Implementation using Class

- 모델을 구성하는 파라미터, Layer 등은 **Dictionary 자료형**으로 클래스 멤버 변수로 관리함
- 파라미터를 Dictionary의 Key 값으로 통합하여 관리할 수 있으며, 직관적인 사용에도 용이함

```
def _inference(self):
    #How the Inference Graph (Main Model) is described?
    #If you want to change the model, change this part and use proper loss function.
    print "Create Inference Graph of Main Model"

    self.paras={}
    self.opt_paras={}
    self.layers={}

    with tf.variable_scope('BATCH_NORM1') as scope:
        batch_mean1, batch_var1 = tf.nn.moments(self.x_inputs, [0])
        self.paras['scale_bn1'] = weight_variable('scale_bn1', [1], initializer = 'normal')
        self.paras['offset_bn1'] = bias_variable('offset_bn1', [1])
        self.layers['bn1'] = tf.nn.batch_normalization(self.x_inputs, batch_mean1, batch_var1, self.paras['offset_bn1'], self.paras['scale_bn1'], 0.0001)

        self.layers['reshape_bn1'] = tf.reshape(self.layers['bn1'], shape=[-1,IMAGE_WIDTH, IMAGE_HEIGHT, IMAGE_DEPTH])
        with tf.variable_scope('CONV1') as scope:
            self.paras['W1'] = weight_variable('W1',[3,3,3,64], initializer = 'Xavier')
            self.paras['b1'] = bias_variable('b1',[64])
            self.layers['conv1'] = tf.nn.conv2d(self.layers['reshape_bn1'],self.paras['W1'],strides=[1,1,1,1],padding='SAME')
            self.layers['relu_conv1'] = tf.nn.relu(self.layers['conv1']+self.paras['b1'], name=scope.name)

    with tf.variable_scope('BATCH_NORM2') as scope:
        batch_mean2, batch_var2 = tf.nn.moments(self.layers['relu_conv1'],[0])
        self.paras['scale_bn2'] = weight_variable('scale_bn2', [1], initializer = 'normal')
        self.paras['offset_bn2'] = bias_variable('offset_bn2', [1])
        self.layers['bn2'] = tf.nn.batch_normalization(self.layers['relu_conv1'], batch_mean2, batch_var2, self.paras['offset_bn2'], self.paras['scale_bn2'], 0.0001)
```

# tf.GraphKeys

- Graph 내의 기본적인 변수 (Variable) 및 오퍼레이션 (Operation) 등에 대한 정보를 포함
- GraphKeys 내 멤버들은 각 Collection에 해당하는 Key 목록을 포함하고 있음
- 실제 호출은 tf.get\_collection 함수를 통해 가능함

tf.GraphKeys	Class Members		
class tf.GraphKeys	ACTIVATIONS	READY_FOR_LOCAL_INIT_OP	자동 저장하지 않음
	ASSET_FILEPATHS	READY_OP	
	BIASES	REGULARIZATION_LOSSES	
	CONCATENATED_VARIABLES	RESOURCES	
	COND_CONTEXT	SAVEABLE_OBJECTS	
	EVAL_STEP	SAVERS	
	GLOBAL_STEP	SUMMARIES	
	GLOBAL_VARIABLES	SUMMARY_OP	
	INIT_OP	TABLE_INITIALIZERS	
	LOCAL_INIT_OP	TRAINABLE_RESOURCE_VARIABLES	
	LOCAL_RESOURCES	TRAINABLE_VARIABLES	
	LOCAL_VARIABLES	TRAIN_OP	
	LOSSES	UPDATE_OPS	
	MODEL_VARIABLES	VARIABLES	
	MOVING_AVERAGE_VARIABLES	WEIGHTS	
	QUEUE_RUNNERS	WHILE_CONTEXT	

## tf.get\_collection

- 해당하는 Key를 가진 변수 및 오퍼레이션의 리스트를 반환하는 함수

```
In [1]: import tensorflow as tf  
  
a = tf.Variable(1, name='exem')  
b = tf.Variable(2, name='postech')  
  
init = tf.global_variables_initializer()  
sess = tf.Session()  
  
sess.run(init)
```

```
In [2]: print sess.run([a,b])  
  
[1, 2]
```

```
In [3]: collect = tf.GraphKeys.GLOBAL_VARIABLES
```

전역 변수로 선언된 모든 변수들의 목록

```
In [4]: show = tf.get_collection(collect)
```

전역 변수들의 값을 리스트 형태로 리턴

```
In [5]: sess.run(show)
```

```
Out[5]: [1, 2]
```

# Collection Setting

- 지정되지 않은 Key를 사용할 때는 변수 선언시에 직접 Collections을 사용해야함
- Collection은 list 형태로 지정하며, 여러 Key 값을 갖도록 설정할 수 있음

```
In [1]: import tensorflow as tf

a = tf.Variable(1, name='exem')
b = tf.Variable(2, name='postech')
c = tf.Variable(3, name='Tensorflow',
                collections=[tf.GraphKeys.GLOBAL_VARIABLES, 'Hi'])

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

sess.run(init)
```

```
In [2]: print sess.run([a,b,c])
[1, 2, 3]
```

```
In [3]: collect = tf.GraphKeys.GLOBAL_VARIABLES
```

```
In [4]: show = tf.get_collection(collect)
```

```
In [5]: sess.run(show)
```

```
Out[5]: [1, 2, 3]
```

```
In [6]: show2 = tf.get_collection("Hi")
```

```
In [7]: sess.run(show2)
```

```
Out[7]: [3]
```

# Regularization with GraphKeys

- tf.get\_collection을 활용하여 Regularization Term을 Cost Function에 추가할 수 있음
- Variable 선언시, regularizer를 직접 선언해주면 정규화된 값들이 GraphKeys.REGULARIZATION\_LOSSES에 추가

## <Regularization with Manual>

```
if REGULARIZATION == True:  
    #Choose Regularizaton Type: L1, L2  
    vars = tf.trainable_variables()  
    if regularization == 'L2':  
        loss = loss + tf.add_n([ tf.nn.l2_loss(v) for v in vars ]) * REGULARIZATION_PARA  
    if regularization == 'L1':  
        loss = loss + tf.add_n([ tf.nn.l1_loss(v) for v in vars ]) * REGULARIZATION_PARA  
  
return loss
```

## <Regularization with GraphKeys>

```
regularizer = tf.contrib.layers.l2_regularizer(REGULARIZATION_PARA)  
initial = tf.contrib.layers.xavier_initializer()  
return tf.get_variable(name, shape, initializer=initial, regularizer=regularizer)
```

- tf.contrib 내 regularizer를 종류에 맞게 선언
- 변수 선언시 regularizer를 함께 선언



```
if REGULARIZATION == True:  
    loss = tf.add(loss, tf.reduce_sum(tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)))
```

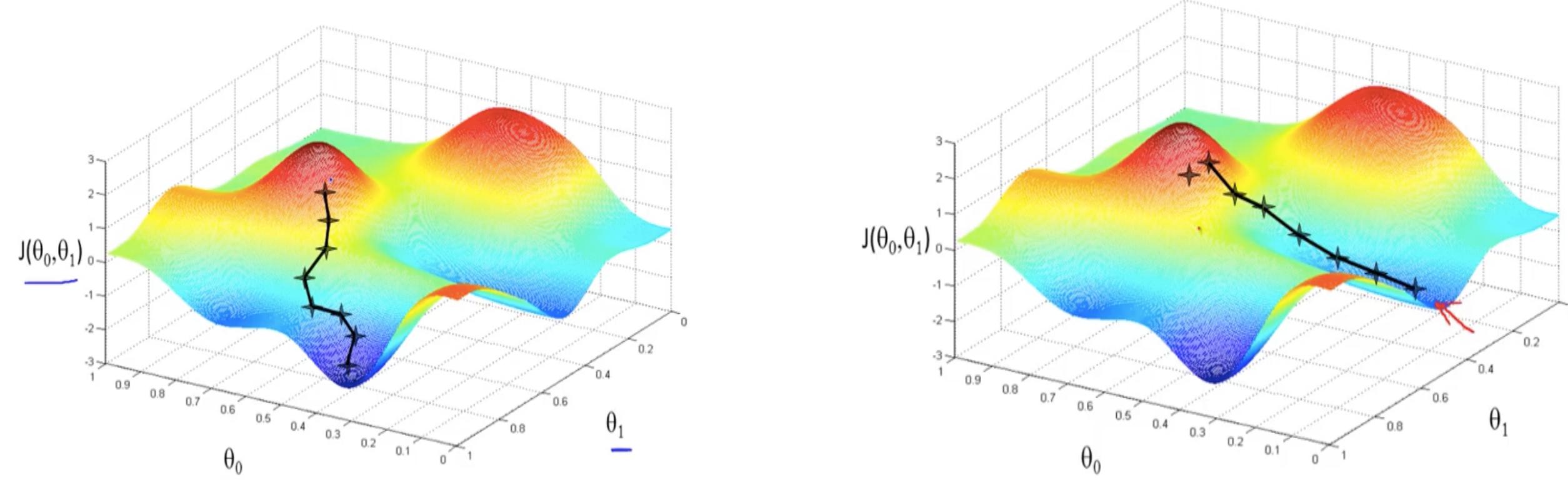
- REGULARIZATION\_LOSSES 내 변수들을 collection한 후 모두 더해줌

# Control Randomization

# Control Randomization

- Parameter의 initialization: 정해진 확률 분포에서 임의로 추출됨 → 제대로 된 통제가 불가능.
- 모델의 개선 여부 측정, 디버깅을 위해 파라미터 초기값을 고정한 후 모델의 성능을 비교할 필요가 있음

<초기값에 따른 학습 결과 차이>



Random Initializer로 구현하더라도  
초기 발생 값을 고정할 필요가 있음

# Random Seed

- 컴퓨터 프로그램이 난수를 생성하는 가장 기본적인 방법은 **seed** 값에 따라 난수를 생성하는 방법임
- Seed 값에 따라 난수 발생 순서가 결정됨 (일반적으로 ms 등 실시간으로 변하는 값을 seed로 사용)
- Seed를 특정한 값으로 지정해주면, 해당 Seed에 해당하는 정해진 순서로 난수가 발생됨

```
>>> c=tf.random_uniform([-10,10,seed=2])
```

Seed 지정

```
>>> sess.run(c)
3.5749321
>>> sess.run(c)
-5.9731865
>>> sess.run(c)
-0.028779984
>>> sess.run(c)
4.2479515
>>> sess.run(c)
9.682888
>>> sess.run(c)
9.1325684
```

```
>>> sess2.run(c)
3.5749321
>>> sess2.run(c)
-5.9731865
>>> sess2.run(c)
-0.028779984
>>> sess2.run(c)
4.2479515
>>> sess2.run(c)
9.682888
>>> sess2.run(c)
9.1325684
```

새로운 Session >> 초기 난수부터 다시 발생

# Random Seed

- 컴퓨터 프로그램이 난수를 생성하는 가장 기본적인 방법은 **seed** 값에 따라 난수를 생성하는 방법임
- **Seed** 값에 따라 난수 발생 순서가 결정됨 (일반적으로 ms 등 실시간으로 변하는 값을 **seed**로 사용)
- **Seed**를 특정한 값으로 지정해주면, 해당 **Seed**에 해당하는 정해진 순서로 난수가 발생됨

## Operation 수준에서 설정

- 국소적인 방법: 각 operation node마다 따로 설정.

```
c=tf.random_uniform([],-10,10,seed=2)
with tf.Session() as sess:
    print sess.run(c) >>> result: 3.57493
?   print sess.run(c) >>> result: -5.97319
```

- Session이 다시 시작될 경우 난수도 초기화.

```
c=tf.random_uniform([],-10,10,seed=2)
with tf.Session() as sess:
    print sess.run(c) >>> result: 3.57493
with tf.Session() as sess:
    print sess.run(c) >>> result: 3.57493
```

## 그래프 수준에서 설정(**tf.set\_random\_seed(seed)**)

- **tf.set\_random\_seed**를 통해 그래프 전체에 해당하는 seed 설정이 가능함
- 다른 그래프에서도 같은 결과를 내고자 할 때 유용.
  - 같은 코드를 가지고 있는 그래프는 같은 결과를 산출.

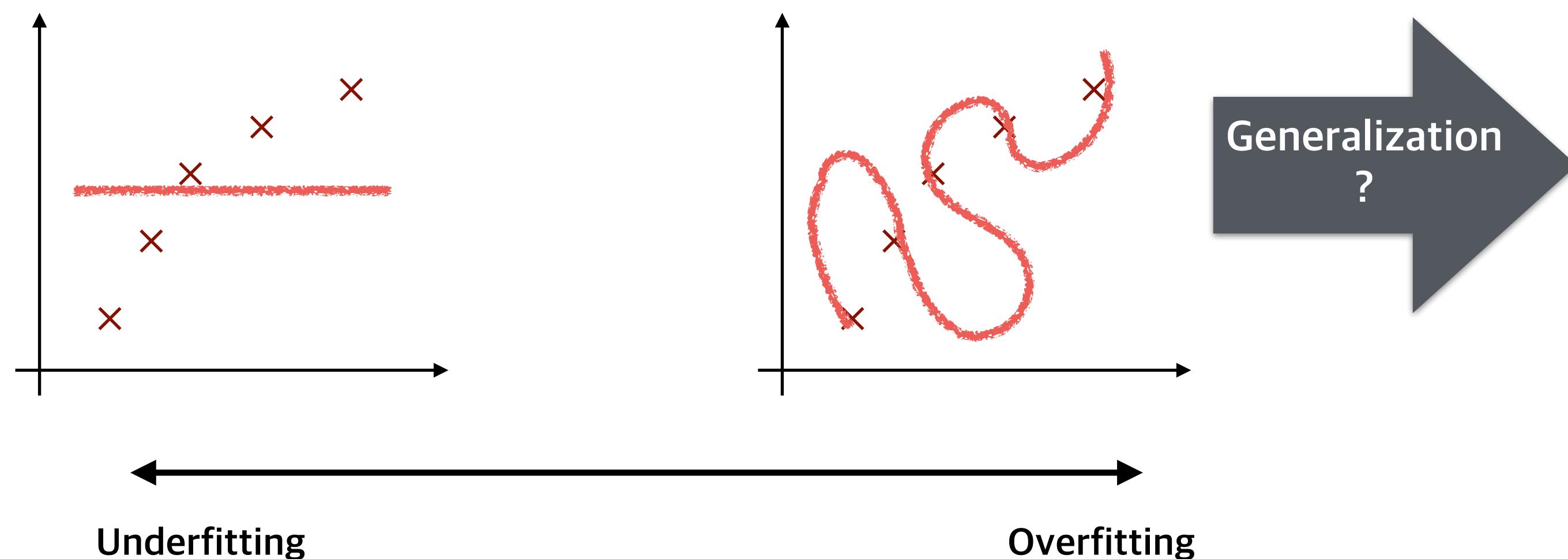
```
tf.set_random_seed(2)
c=tf.random_uniform([],-10,10)
d=tf.random_uniform([],-10,10)
with tf.Session() as sess:
    print sess.run(c) >>> result: -4.00752
    print sess.run(d) >>> result: -2.98339
```

# Avoiding Overfitting

# Regularization Term in Cost Function

- 모델 학습시, Training data set에 편향되어 학습되어지는 **Overfitting 현상**이 종종 발생함
- 파라미터의 수가 많고, 각 파라미터의 크기가 커질수록 Overfitting 현상이 심화됨
- Overfitting을 방지하기 위한 방법으로 Cost Function에 **Regularization Term**을 추가하는 방법이 있음

## <Overfitting Example>



$$J(\Theta) = \frac{1}{m} \sum L_i + \lambda R(\Theta)$$

Regularization Term

- L1 regularization:  $\lambda |\Theta|$  (Regularization in 3 Layer NN)
- L2 regularization:  $\lambda |\Theta^2|$  
$$\frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$
- $\lambda$  가 커질수록 Underfitting 되는 효과가 있음 (Why?)
- Regularization is “Weight Decaying”
  - Regularization Term은 학습시 자기 자신의 Weight 크기 만큼 변화를 감소시키는 역할을 함
  - 학습에 필수적인 Weights는 감소 효과를 극복하고 지속적으로 Weights를 업데이트하며 학습 진행

# Cross Validation Data

- 인공지능의 목적 중 하나는 **훈련되지 않은 새로운 상황**에 대하여 높은 성능을 유지하는 것
- 보유하고 있는 데이터를 1) Training, 2) Validation, 3) Test Data로 나누어서 성능을 검증

<Data Split with No Validation >

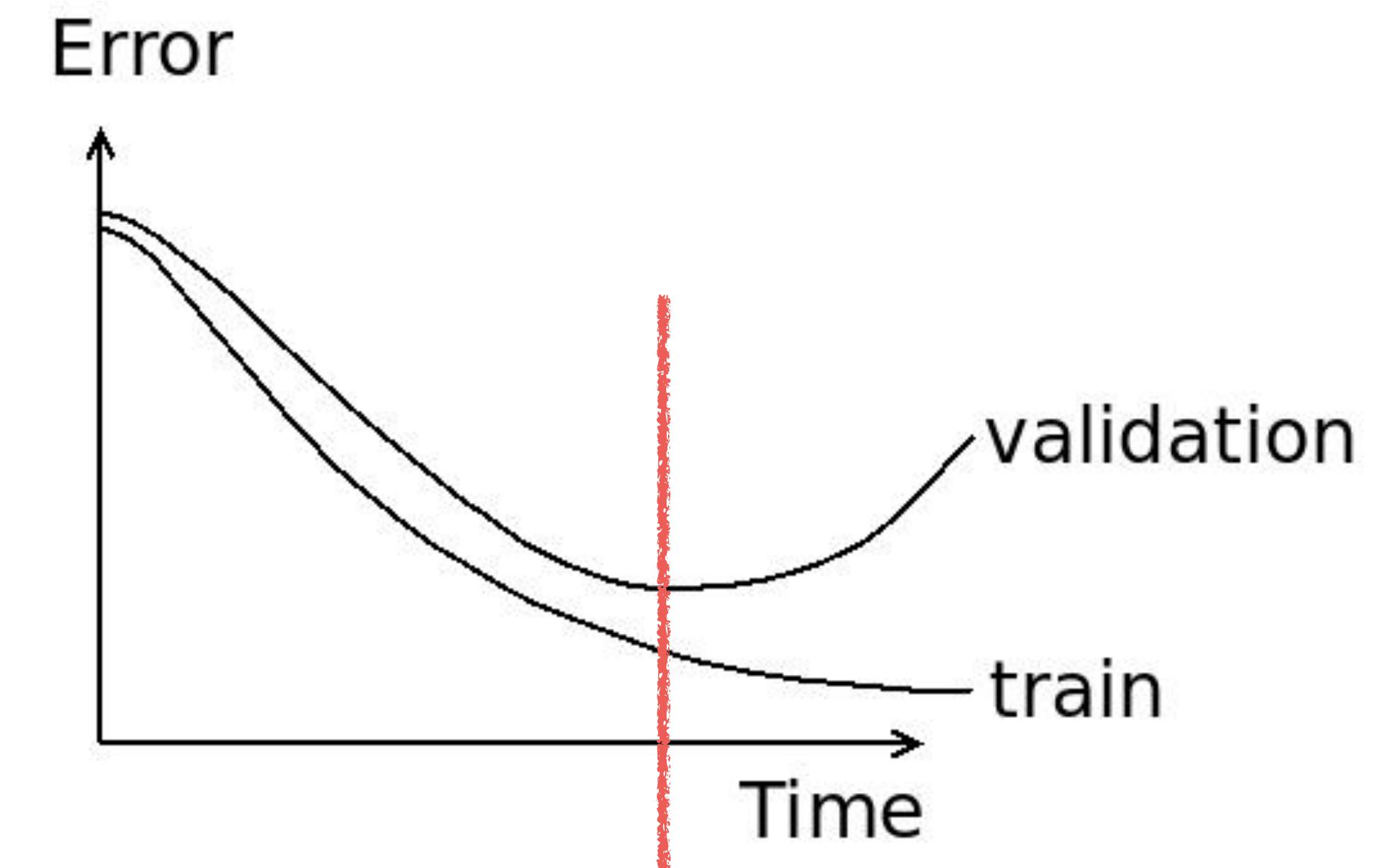


<Data Split with Validation>



\*Test 데이터는 학습 완료 후 **최종 성능을 확인**하기 위해서만 사용됨

<Learning Curve>

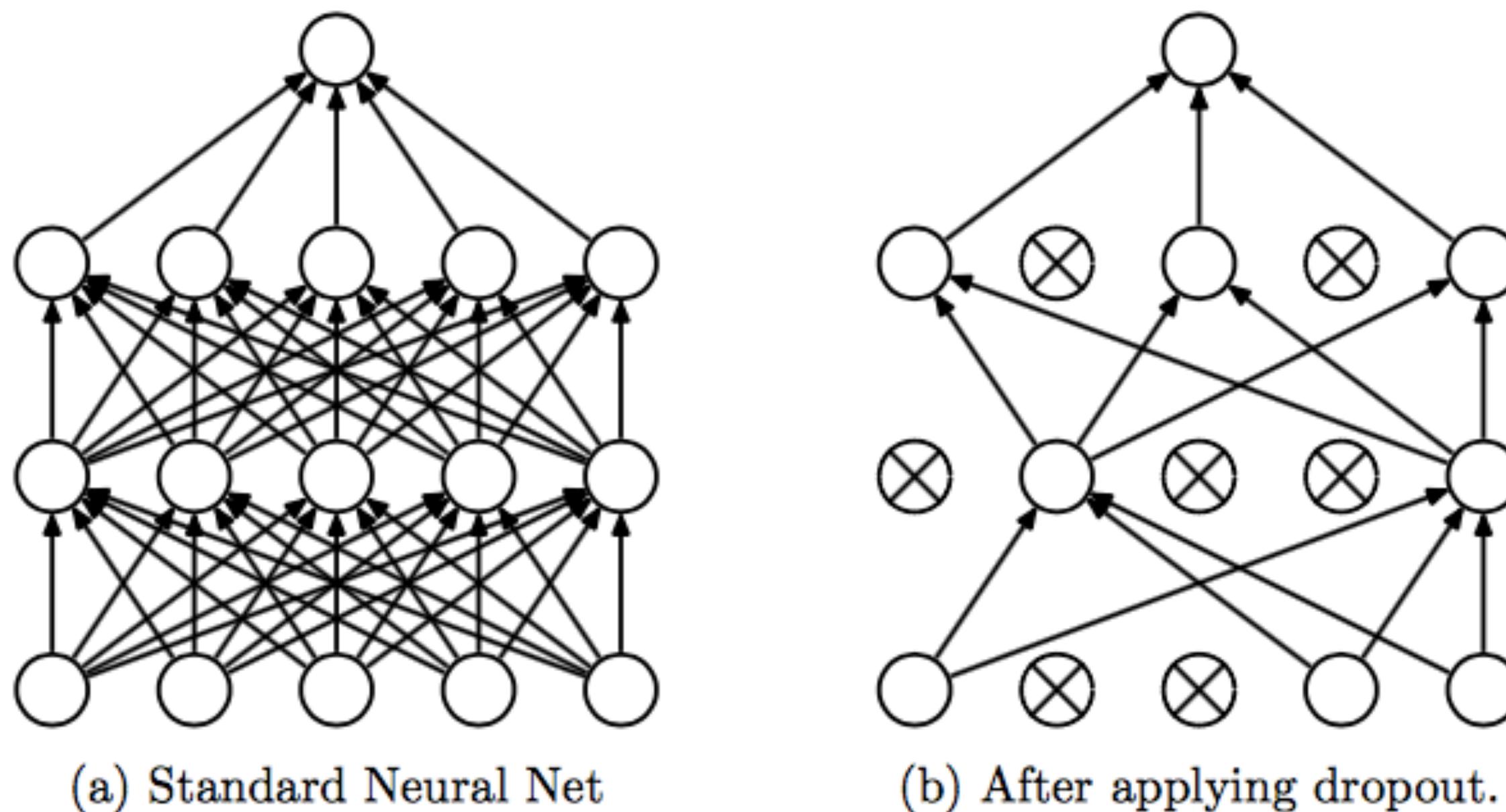


**Early Stop**

# Dropout

- Dropout은 overfitting을 예방하기 위해 사용되는 가장 대표적인 방법 중 하나
- 학습 과정에서 각 뉴런을 **(1-p)**의 확률로 생략하여, 생략된 뉴런을 고려하지 않고 Feedforward/Backward 연산을 진행
- Test 시에는 dropout을 사용하지 않고, 기존과 같은 방법으로 모든 뉴런에 대하여 결과를 계산함

<Dropout Architecture>

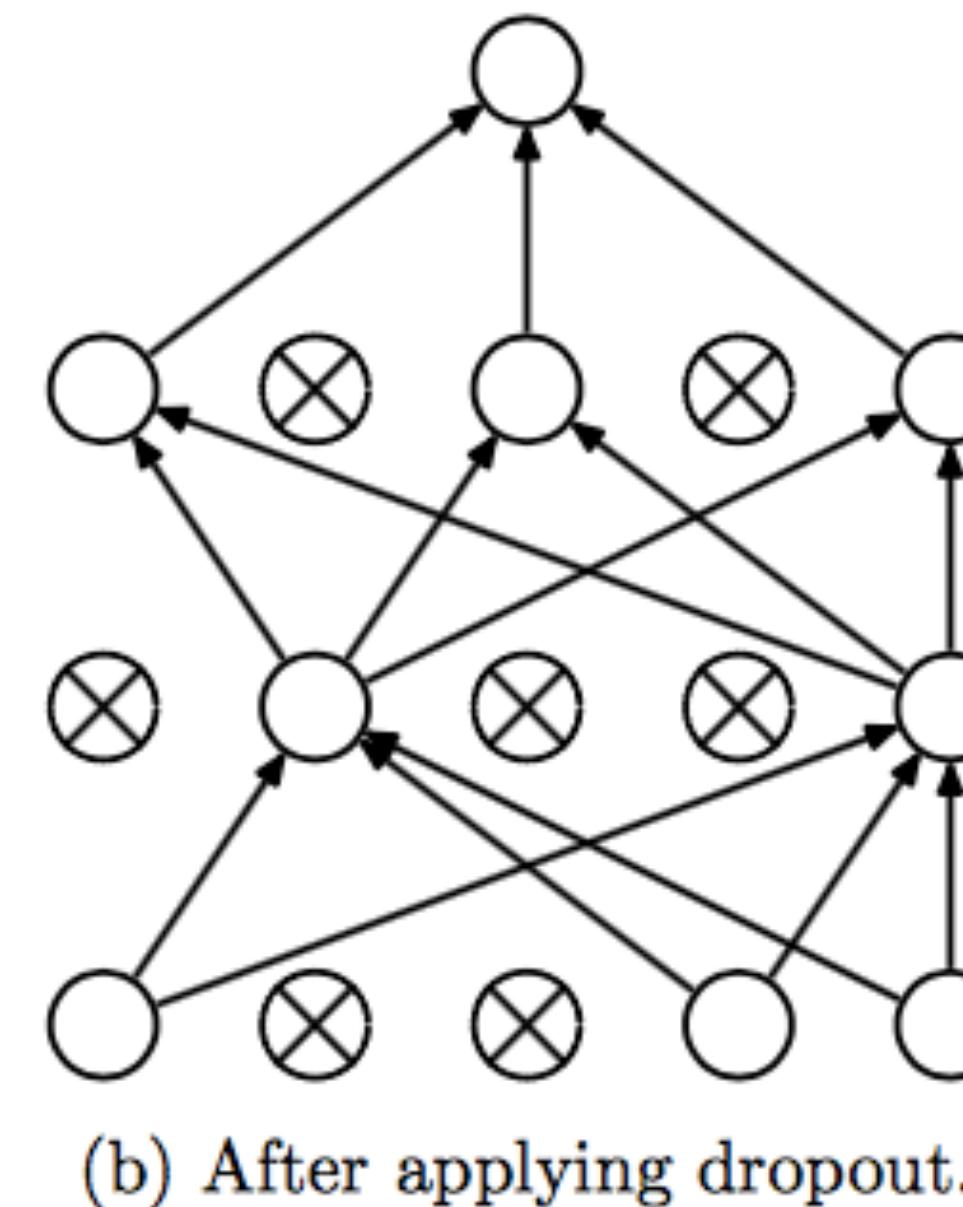
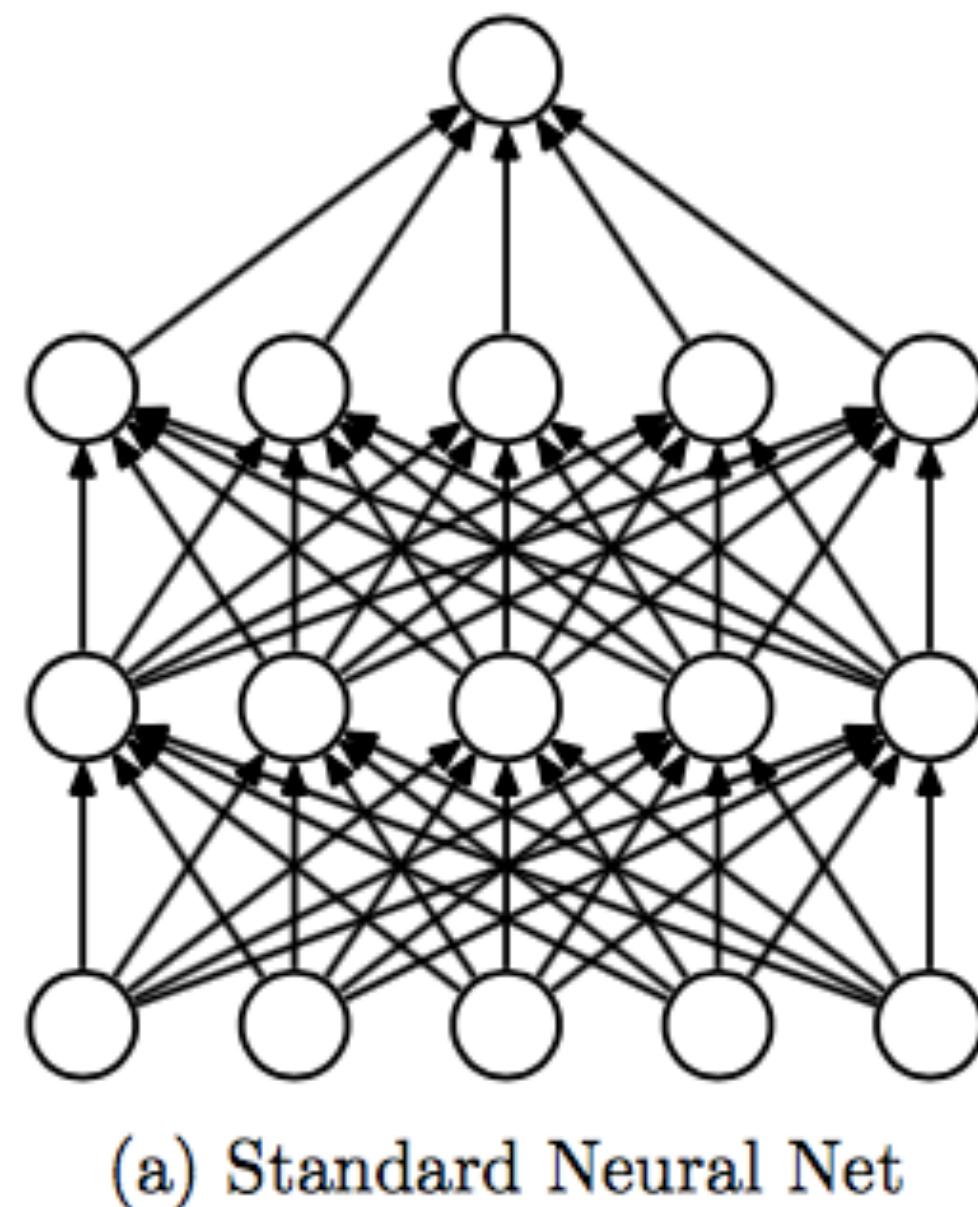


$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \end{aligned}$$

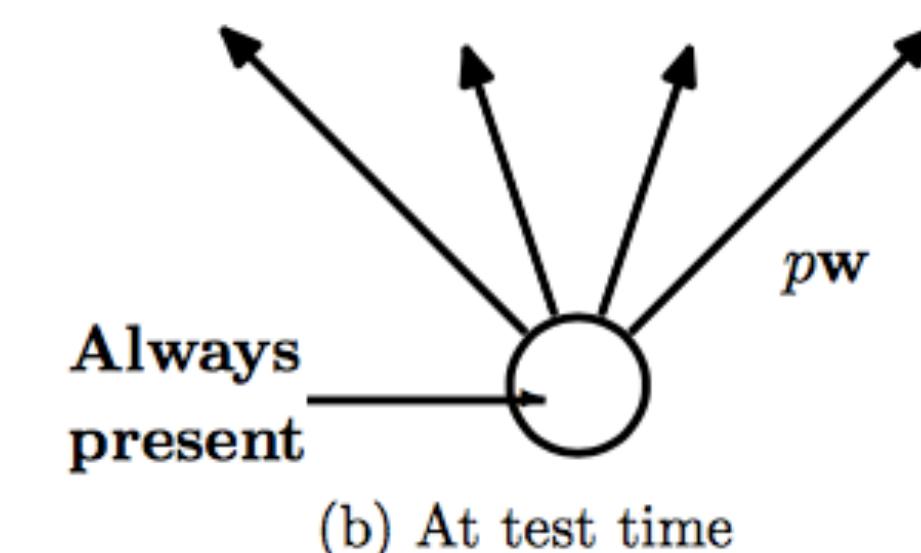
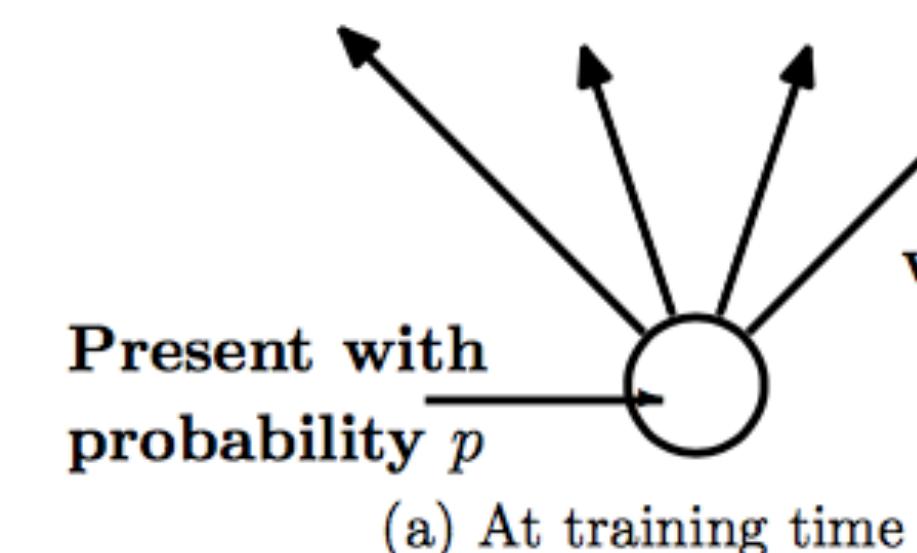
# Dropout

- Dropout은 overfitting을 예방하기 위해 사용되는 가장 대표적인 방법 중 하나
- 학습 과정에서 각 뉴런을 **(1-p)**의 확률로 생략하여, 생략된 뉴런을 고려하지 않고 Feedforward/Backward 연산을 진행
- Test 시에는 dropout을 사용하지 않고, 기존과 같은 방법으로 모든 뉴런에 대하여 결과를 계산함

<Dropout Architecture>



<Training & Test Comparision>

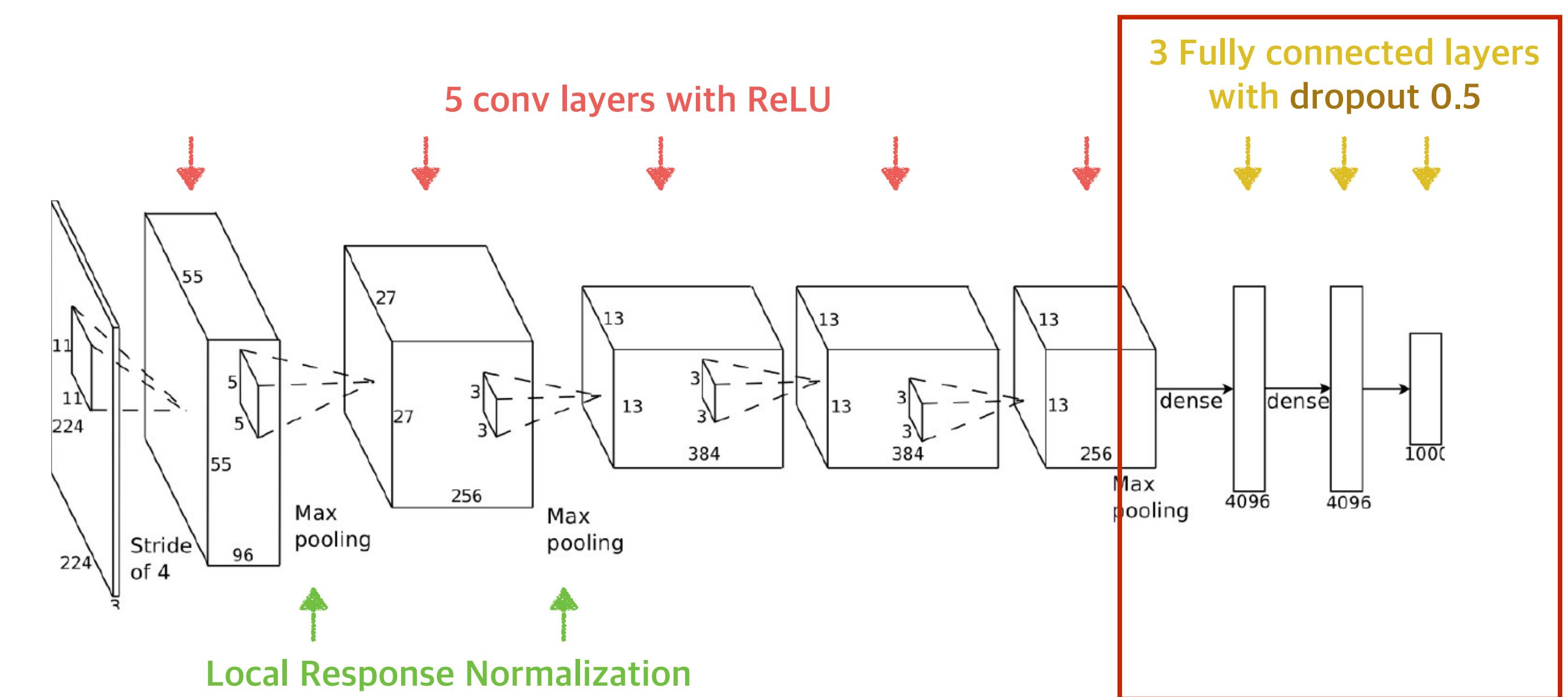
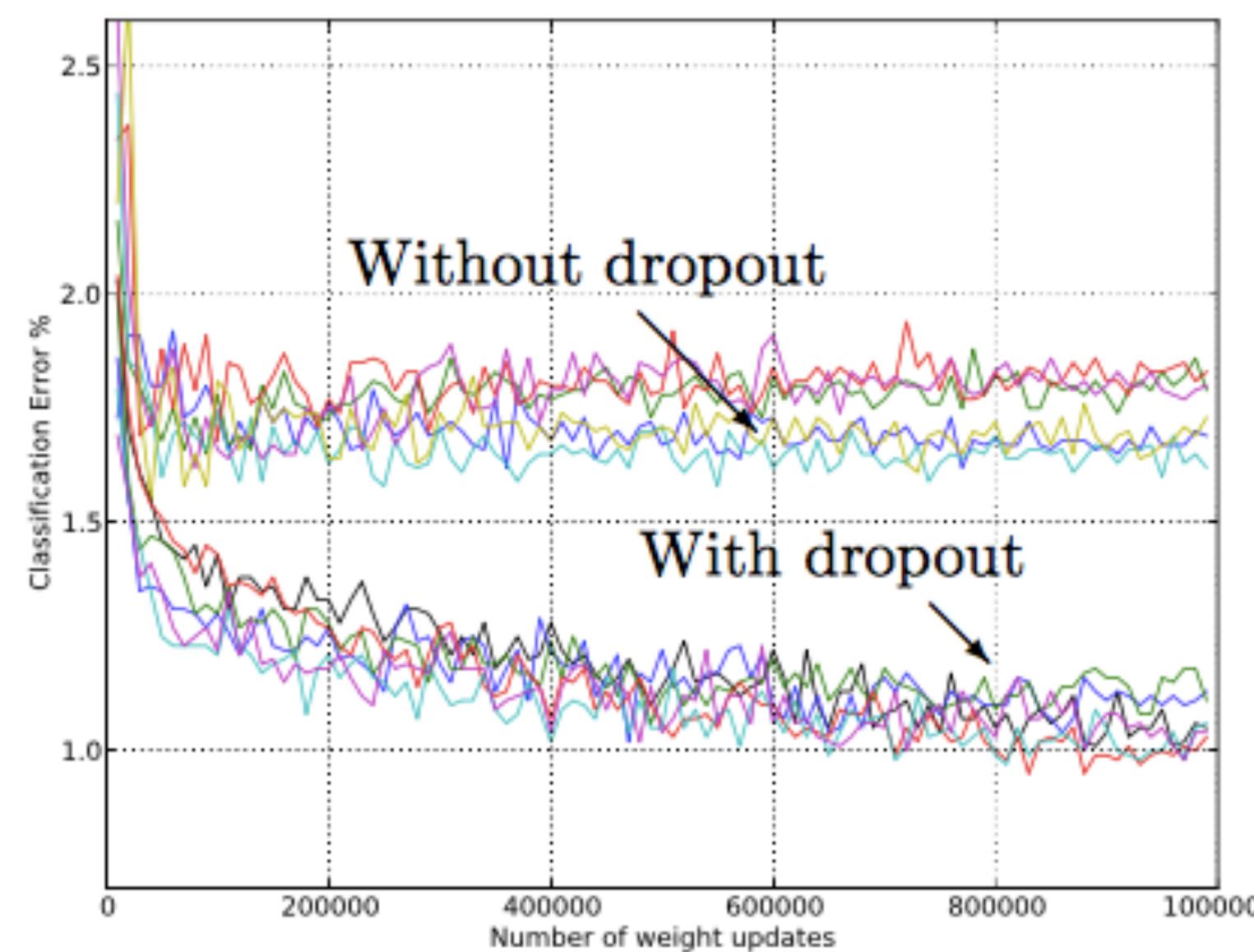


$$\begin{aligned} r_j^{(l)} &\sim \text{Bernoulli}(p), \\ \tilde{\mathbf{y}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{y}^{(l)}, \end{aligned}$$

# Dropout

- 적절한 dropout 확률  $p$ 를 사용하여 모델을 학습할 경우, 모델의 성능을 향상시키는 것으로 나타남
- AlexNet (The ILSVRC 2012 Winner)의 Fully connected layer에서 dropout 사용함 ( $p=0.5$ )

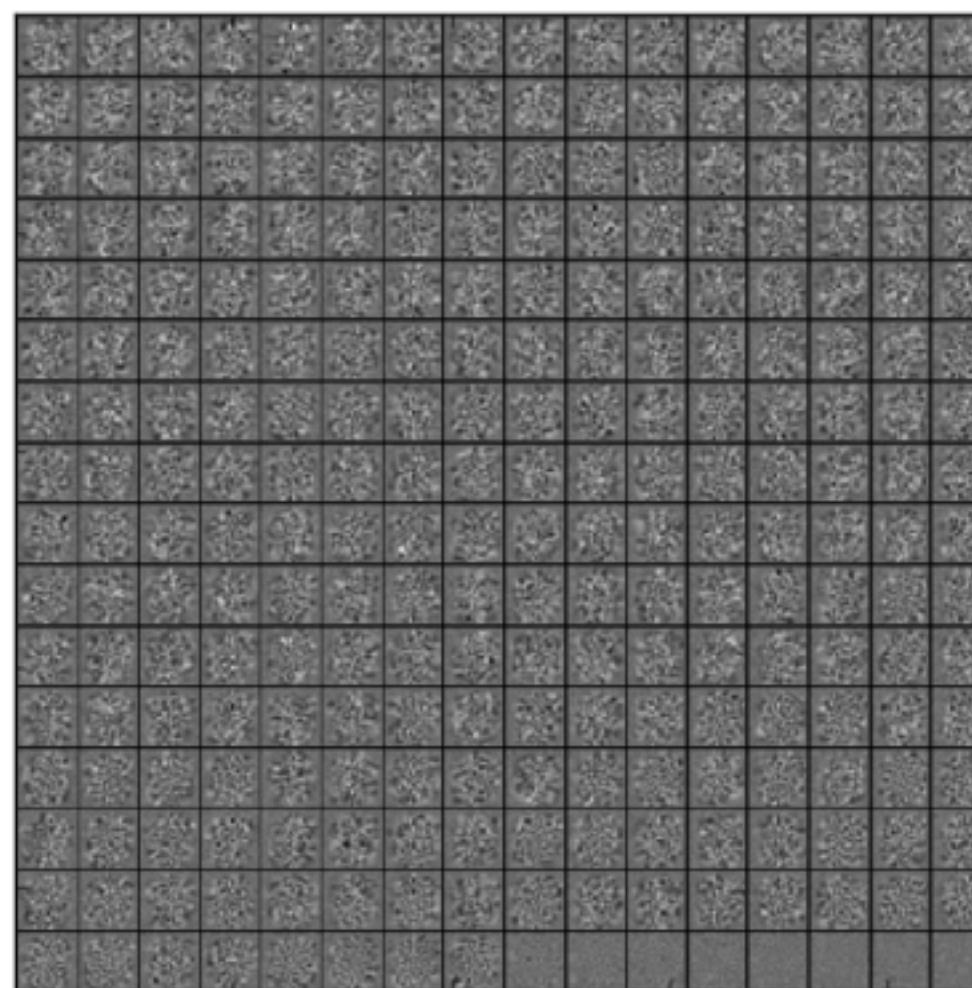
<MNIST Classification Result>



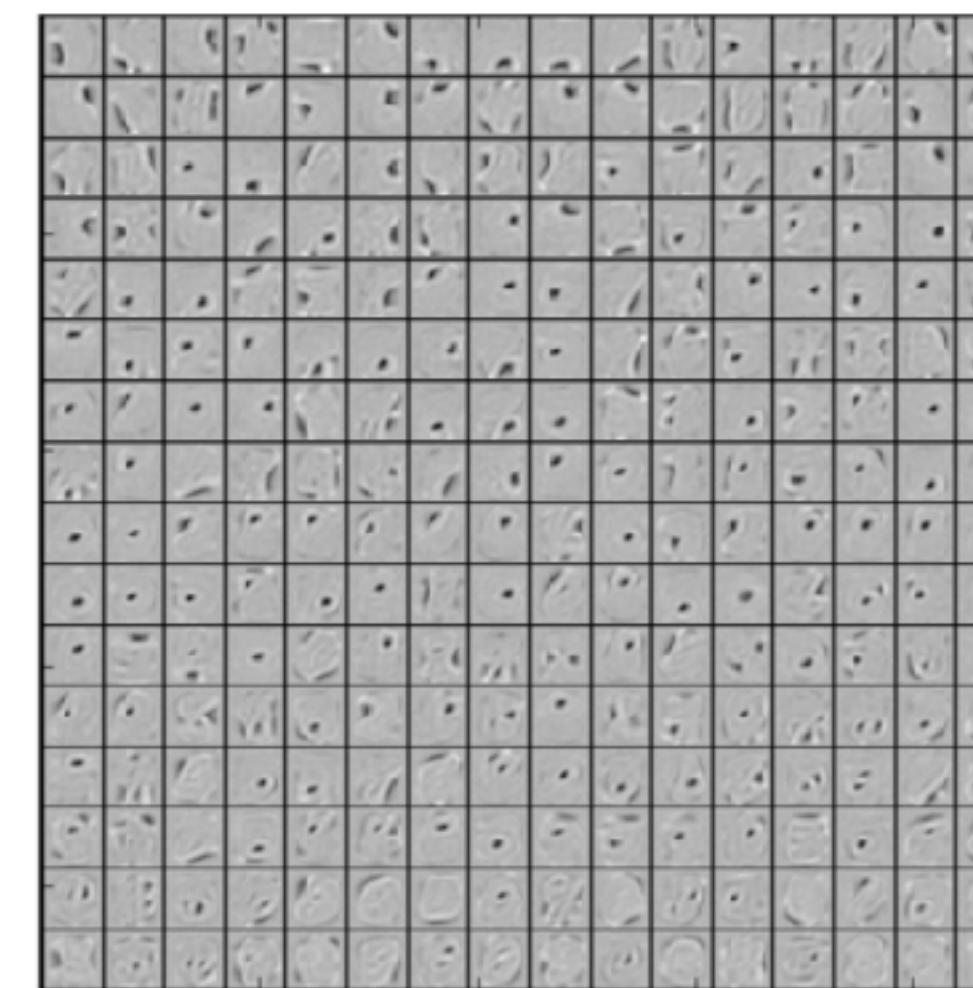
# Dropout Effect

- Dropout을 사용하면 뉴런들 사이의 **co-adaption** 현상을 방지할 수 있음
  - Co-adaption: 주위 뉴런의 반응에 동조하여 함께 반응하는 현상, Overfitting의 주 원인 중 하나
- Dropout을 통해 뉴런 사이의 연관관계를 최소화하고, **activation의 sparsity**를 유지; 각기 다른 특징에 대하여 반응
- 다양한 모델을 동시에 학습하는 것으로 해석할 수 있으며, 앙상블(Ensemble)을 사용하는 것과 유사한 효과

<Features Learned on MNIST>

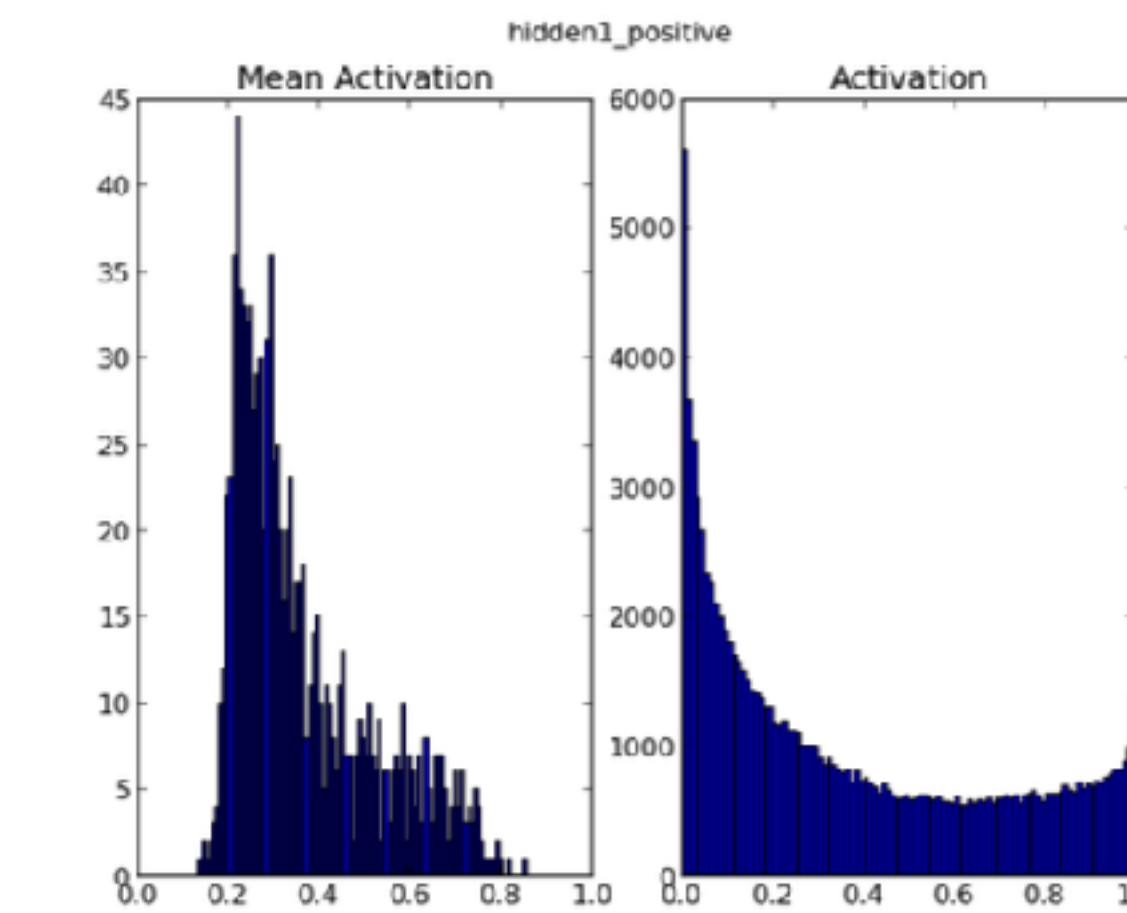


(a) Without dropout

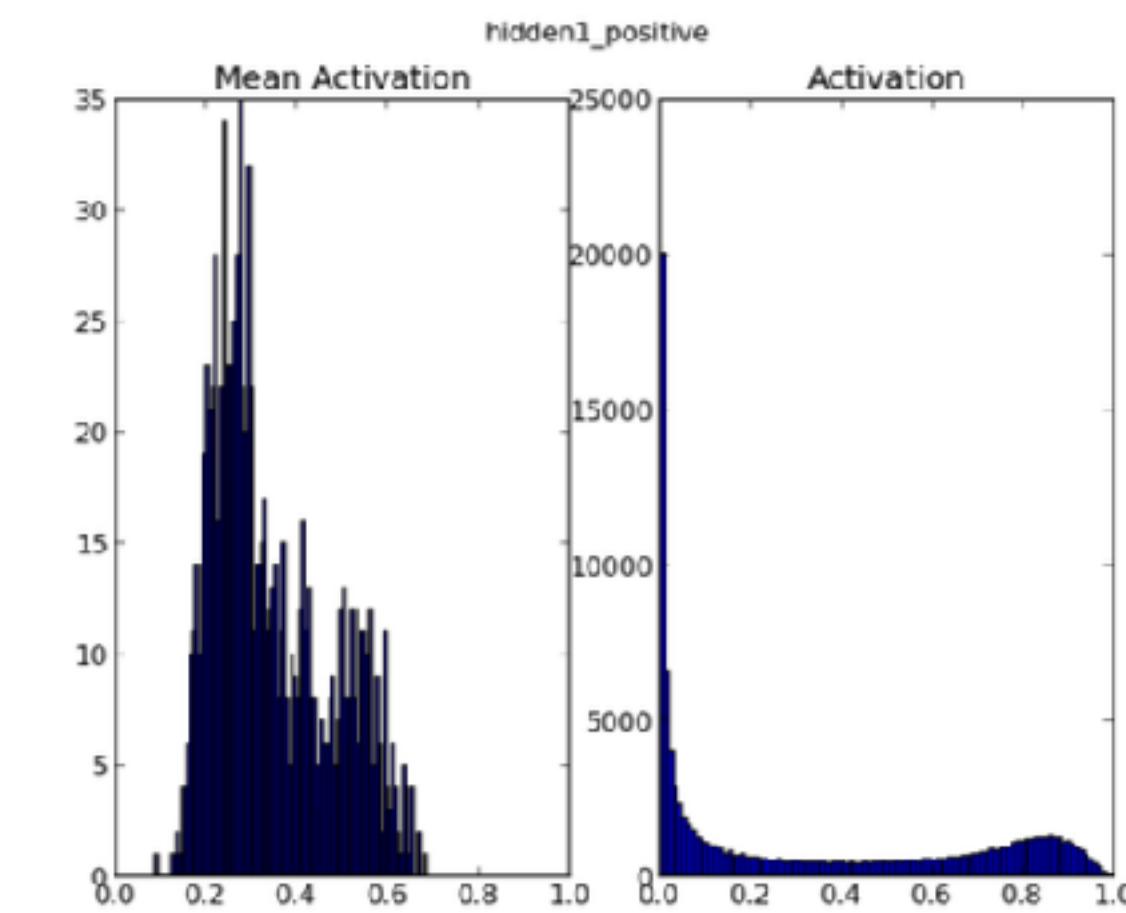


(b) Dropout with  $p = 0.5$ .

<Dropout Effect on Sparsity>



(a) Without dropout



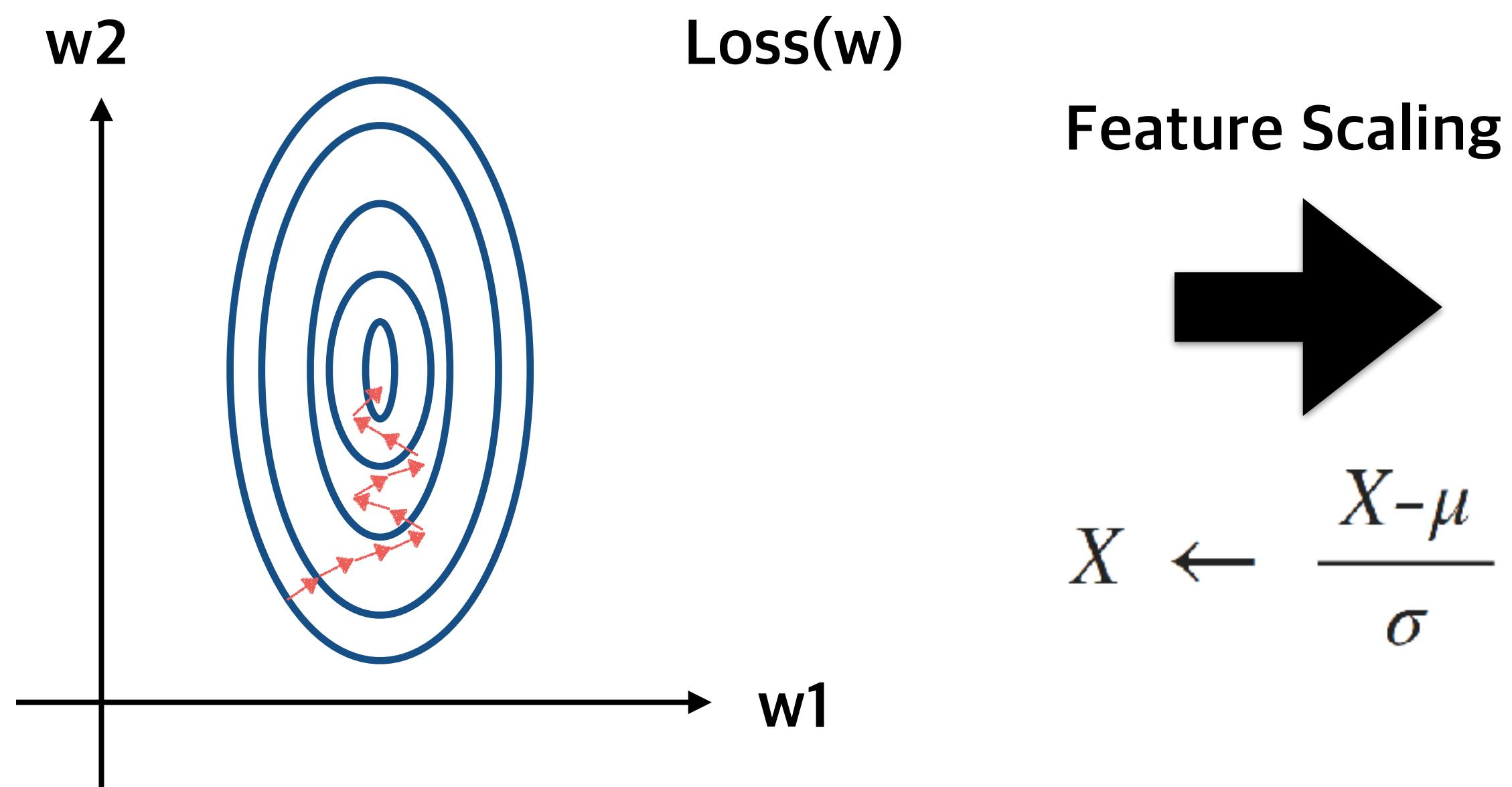
(b) Dropout with  $p = 0.5$ .

# Learning Much Faster

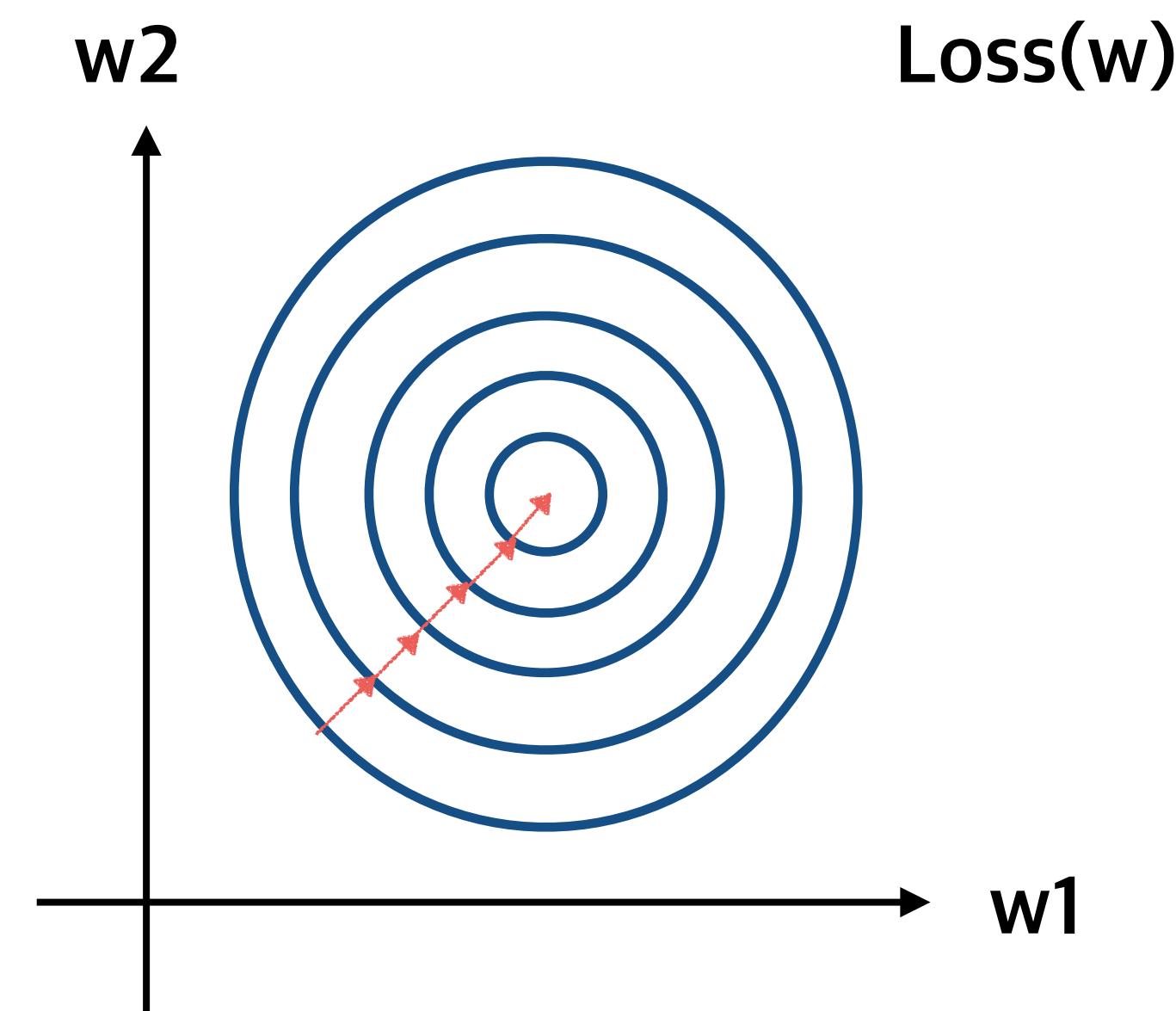
# Feature Scaling

- 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>



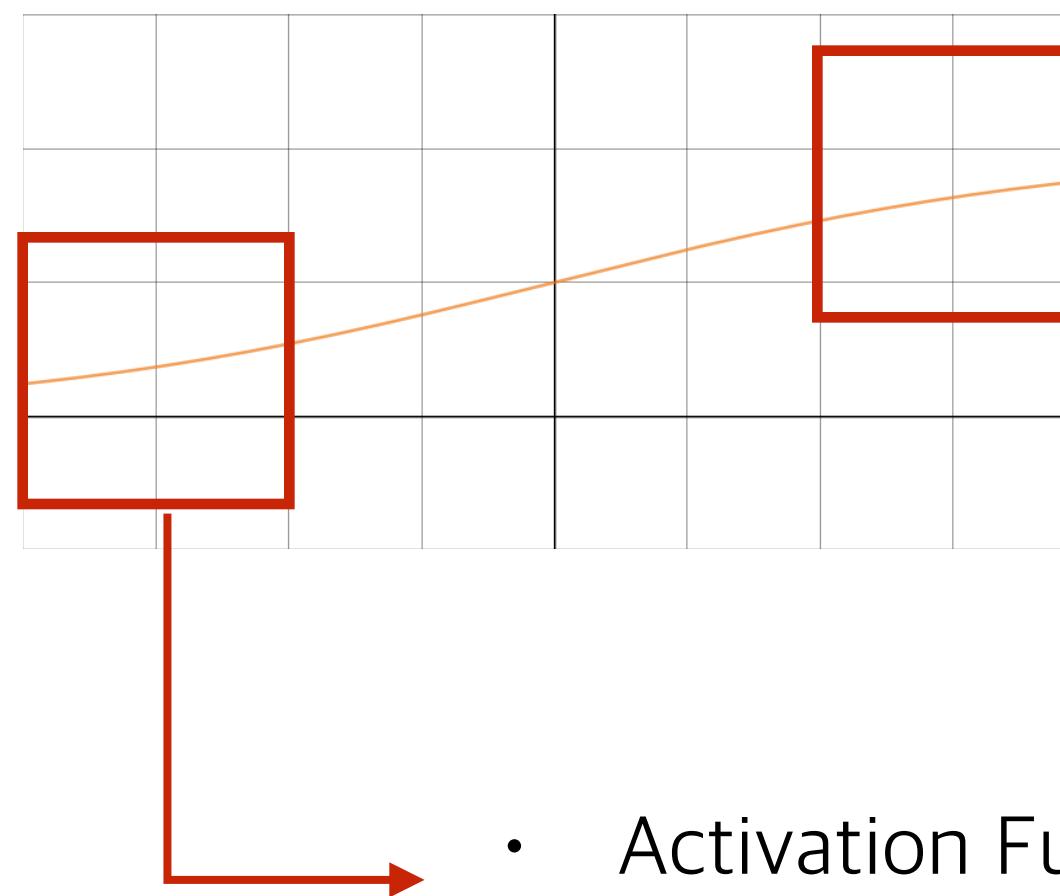
# Feature Scaling

- 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>

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

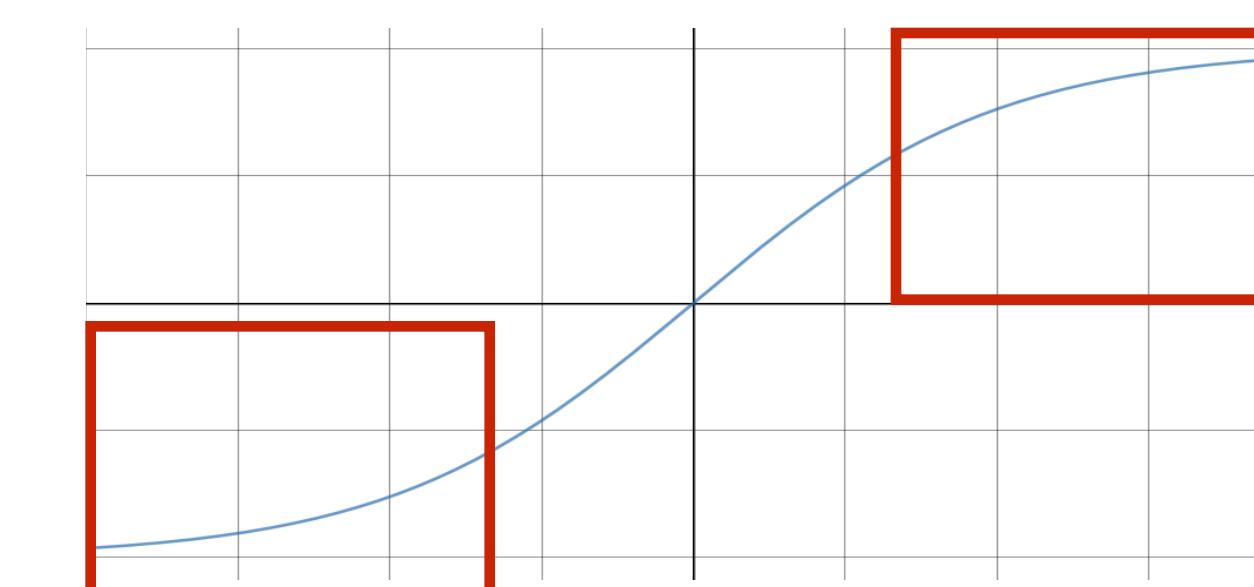
$$f'(x) = f(x)(1-f(x))$$



<tanh Function>

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

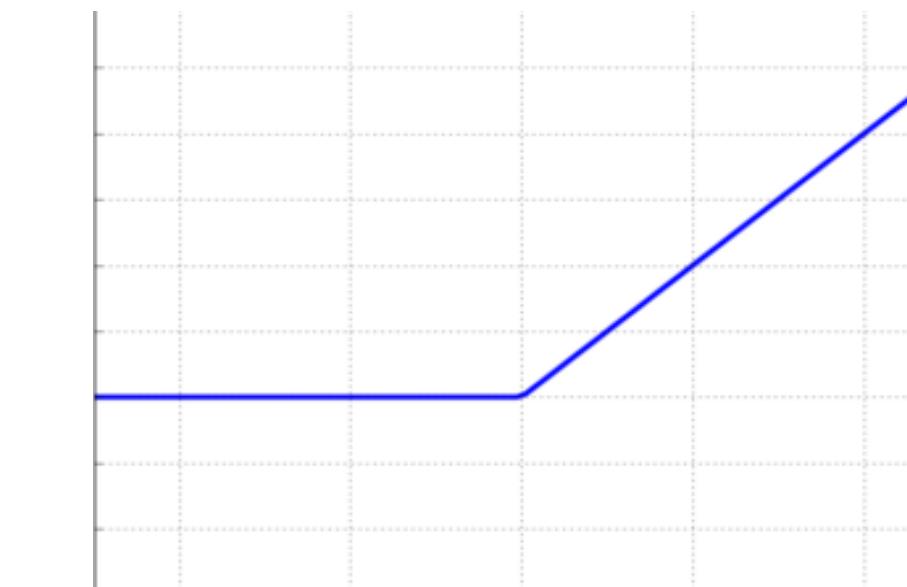
$$f'(x) = 1 - f(x)^2$$



<ReLU Function>

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

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



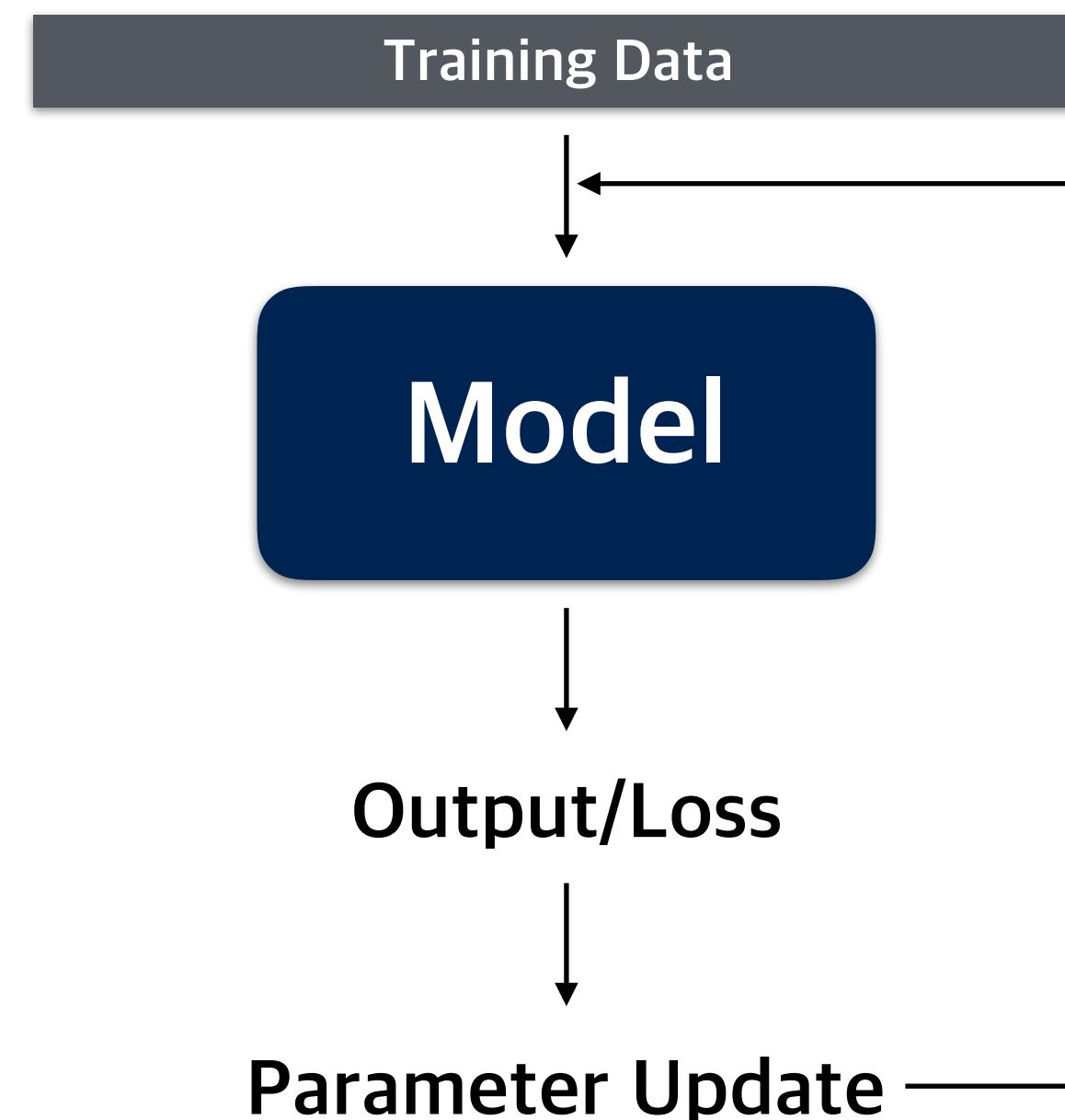
- Activation Function의 포화(Saturation) 부분
- 큰 값의 Scale을 가지는 Feature가 존재할 경우, 계산 결과가 이 부분에 해당할 가능성 높음
- Gradient 값이 0에 가까워, 학습이 거의 진행되지 않음

# Mini-batch Learning for Effective Learning

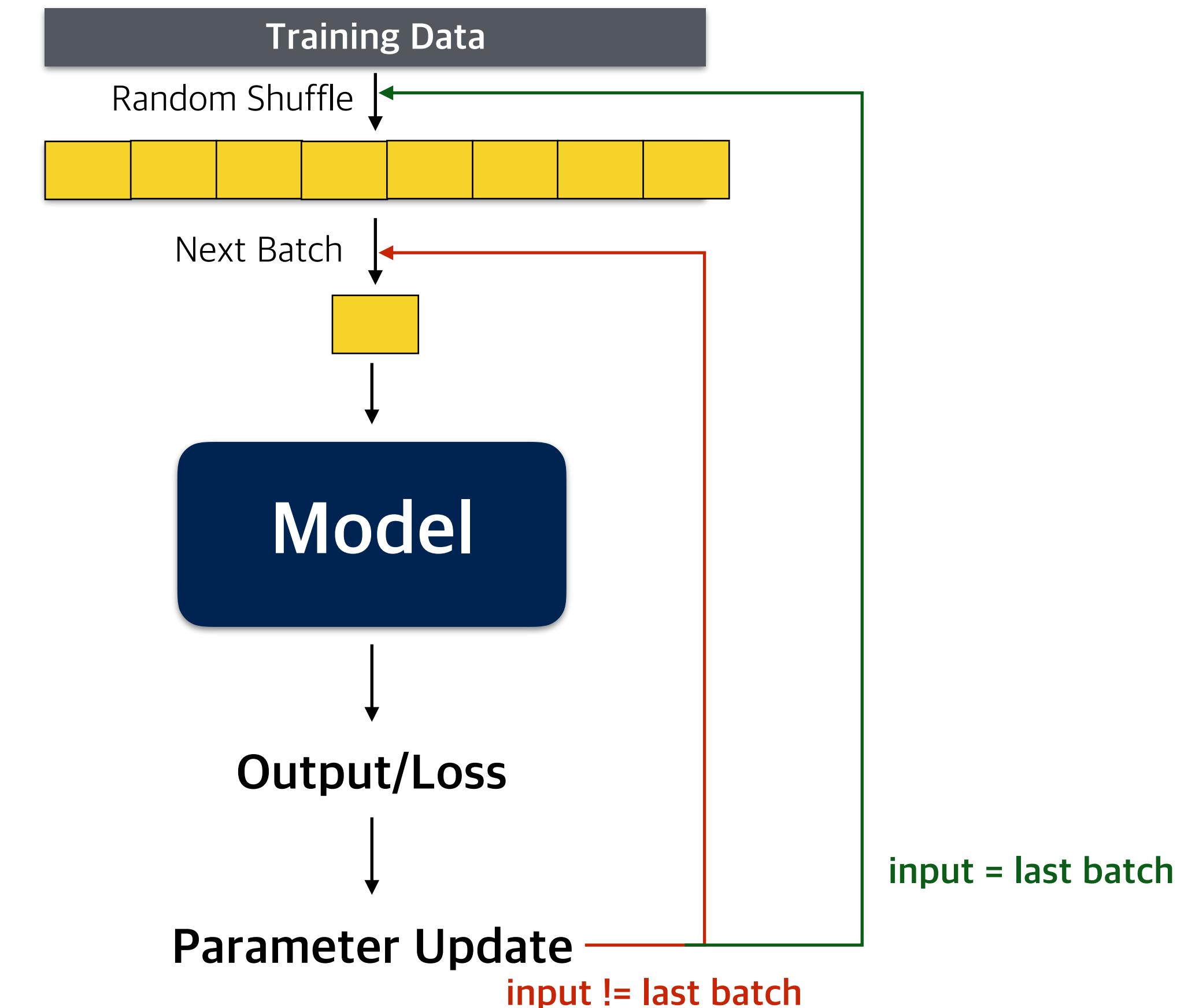
## ❑ Mini Batch Learning을 통해 학습의 속도를 높이고 local optimum에 빠지는 문제를 어느정도 해결 가능

- 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>

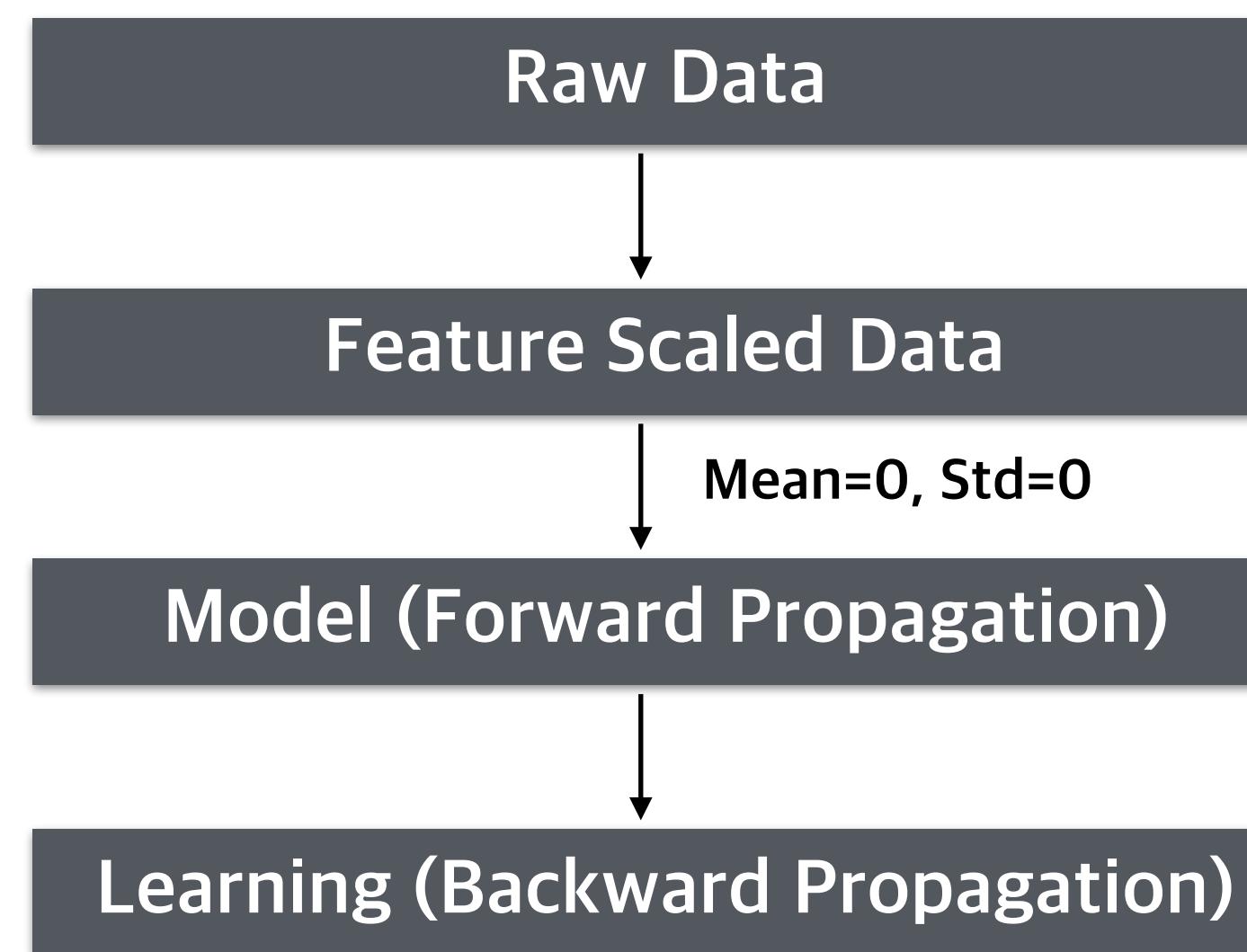


Mini-batch 방식으로 학습을 진행할 때,  
앞에서 해결하고자 했던 Feature Scale 관련 문제가 다시 발생……  
(Why?)

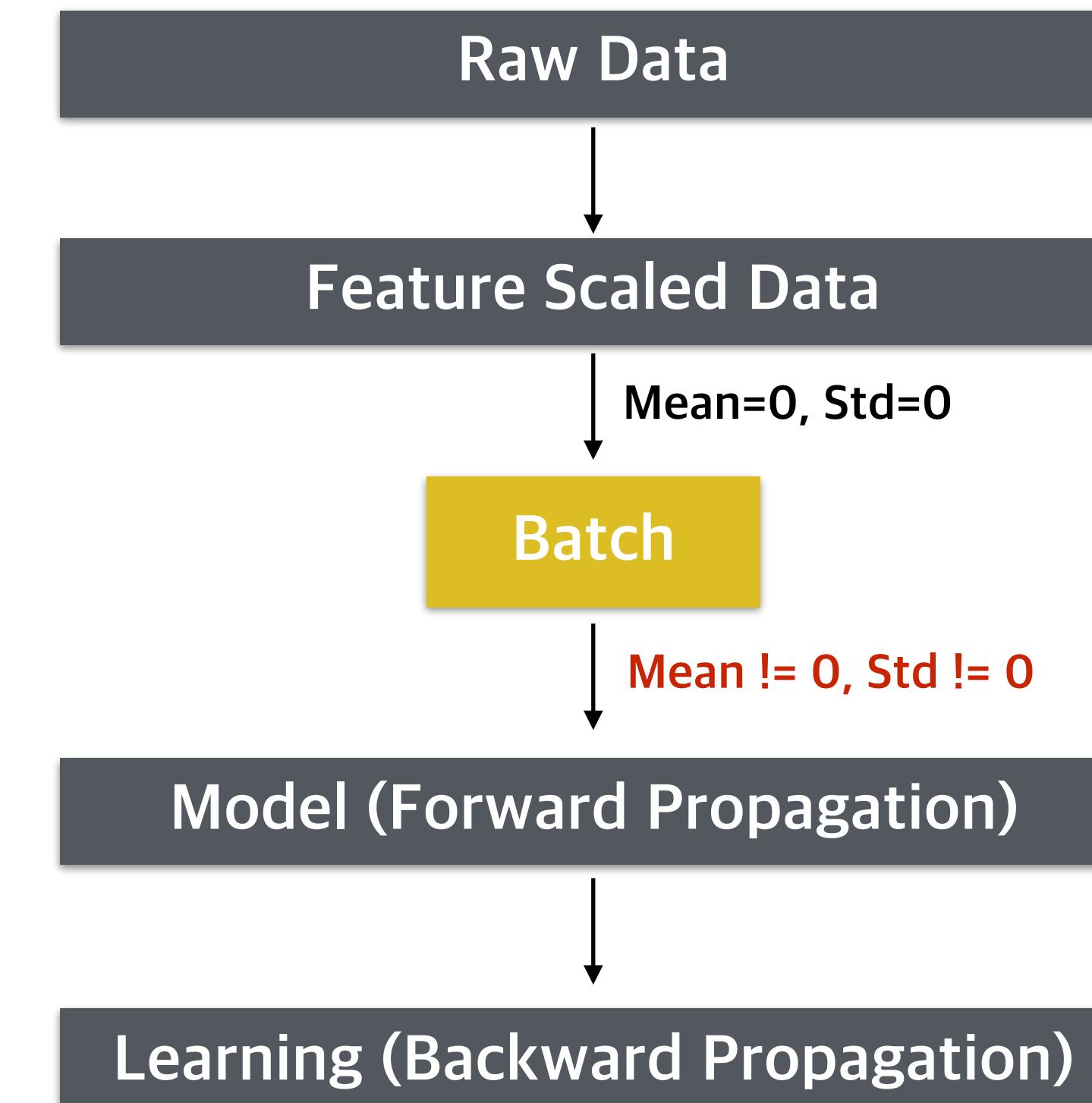
# Mini-Batch Learning and Normalization

- 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>

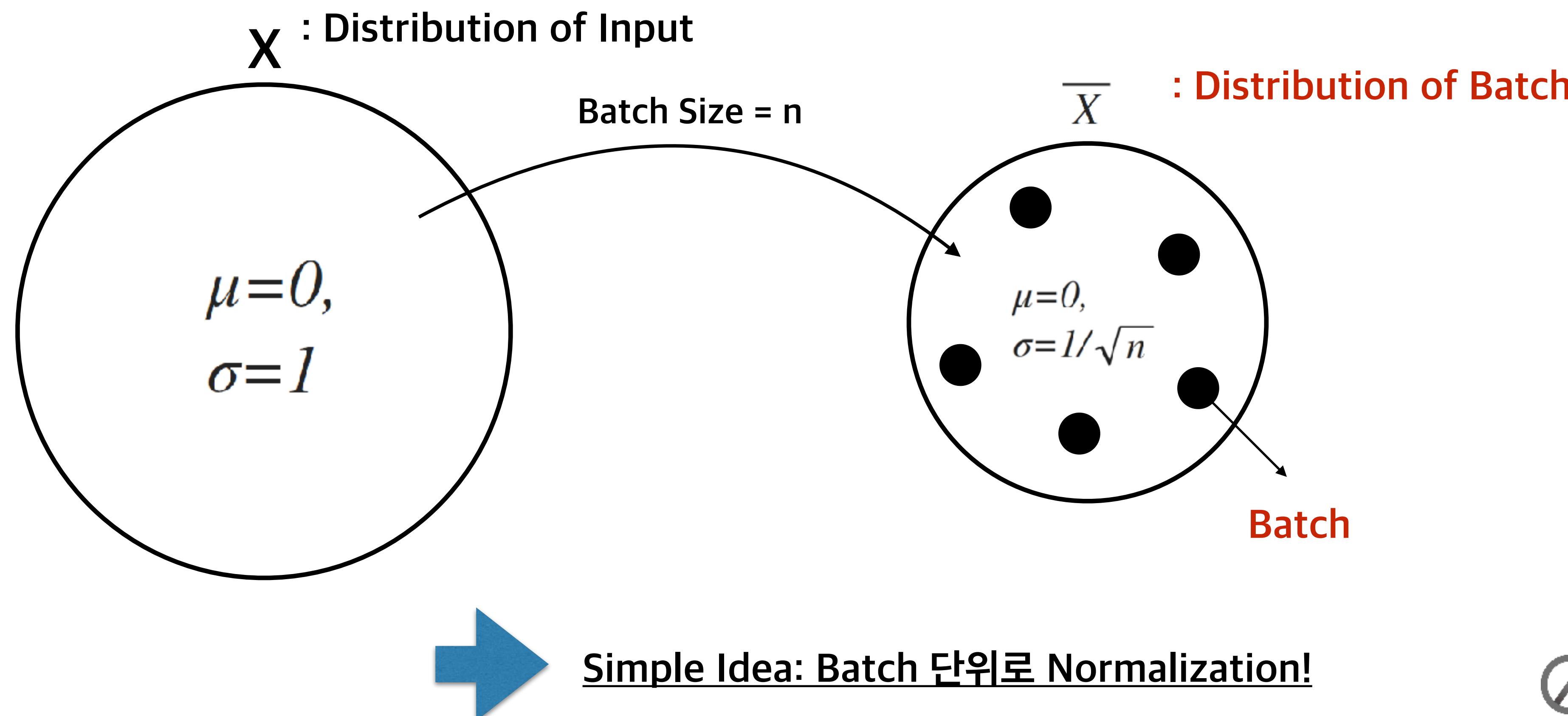


<Mini-batch Learning Process>



# 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)**

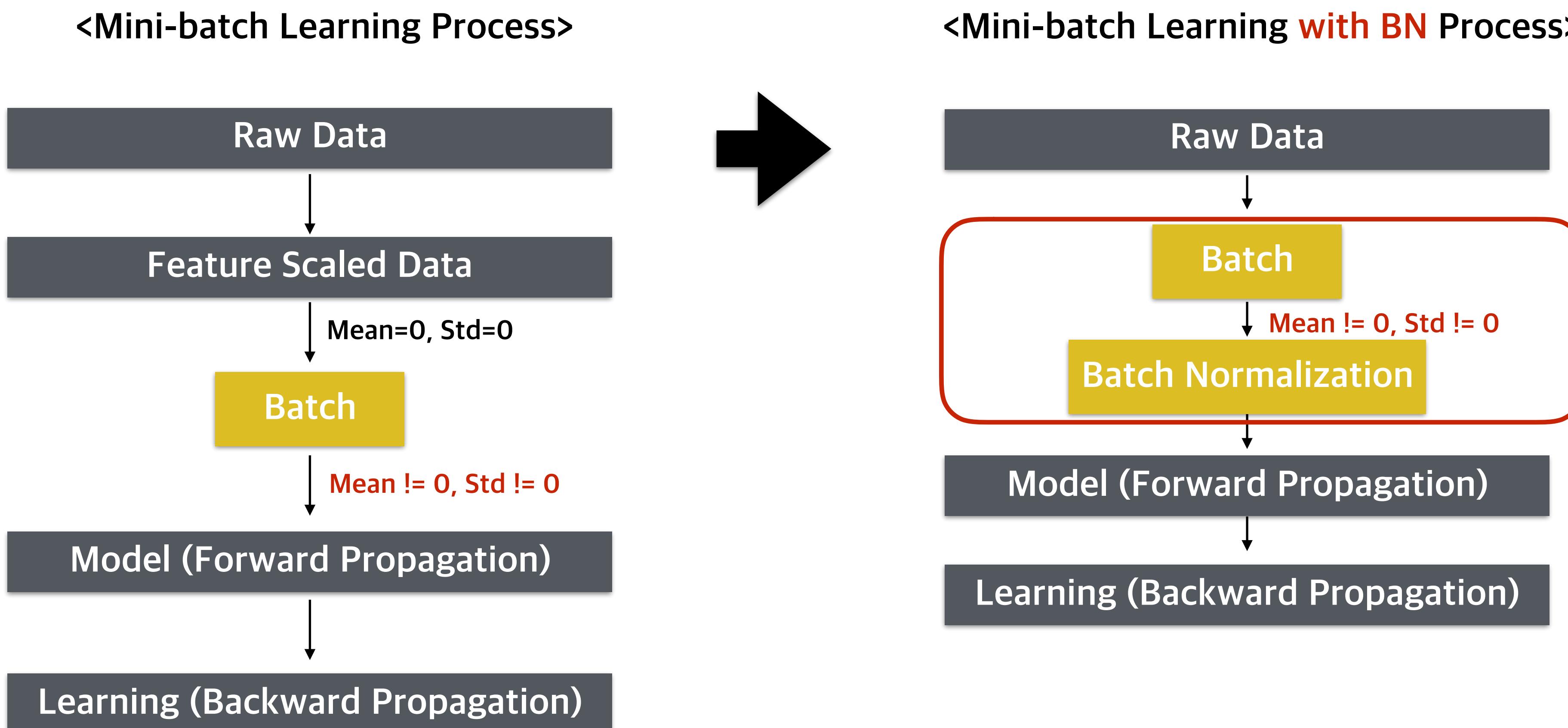


# Batch Normalization

## 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:  $\gamma, \beta$   
**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:  $\gamma, \beta$   
**Output:**  $\{y_i = BN_{\gamma, \beta}(x_i)\}$

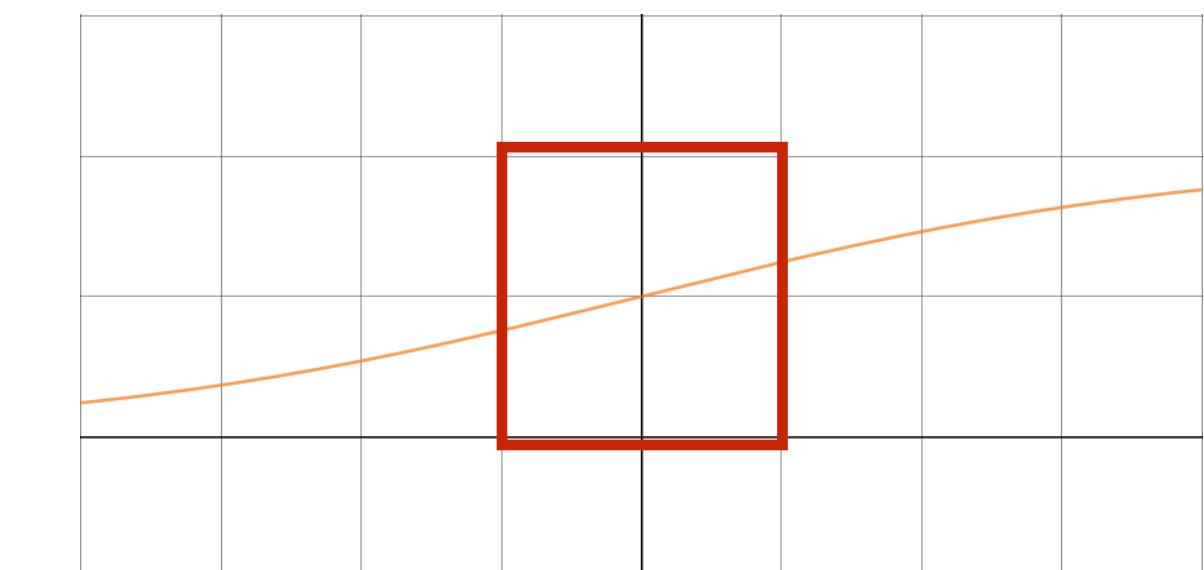
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2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

**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

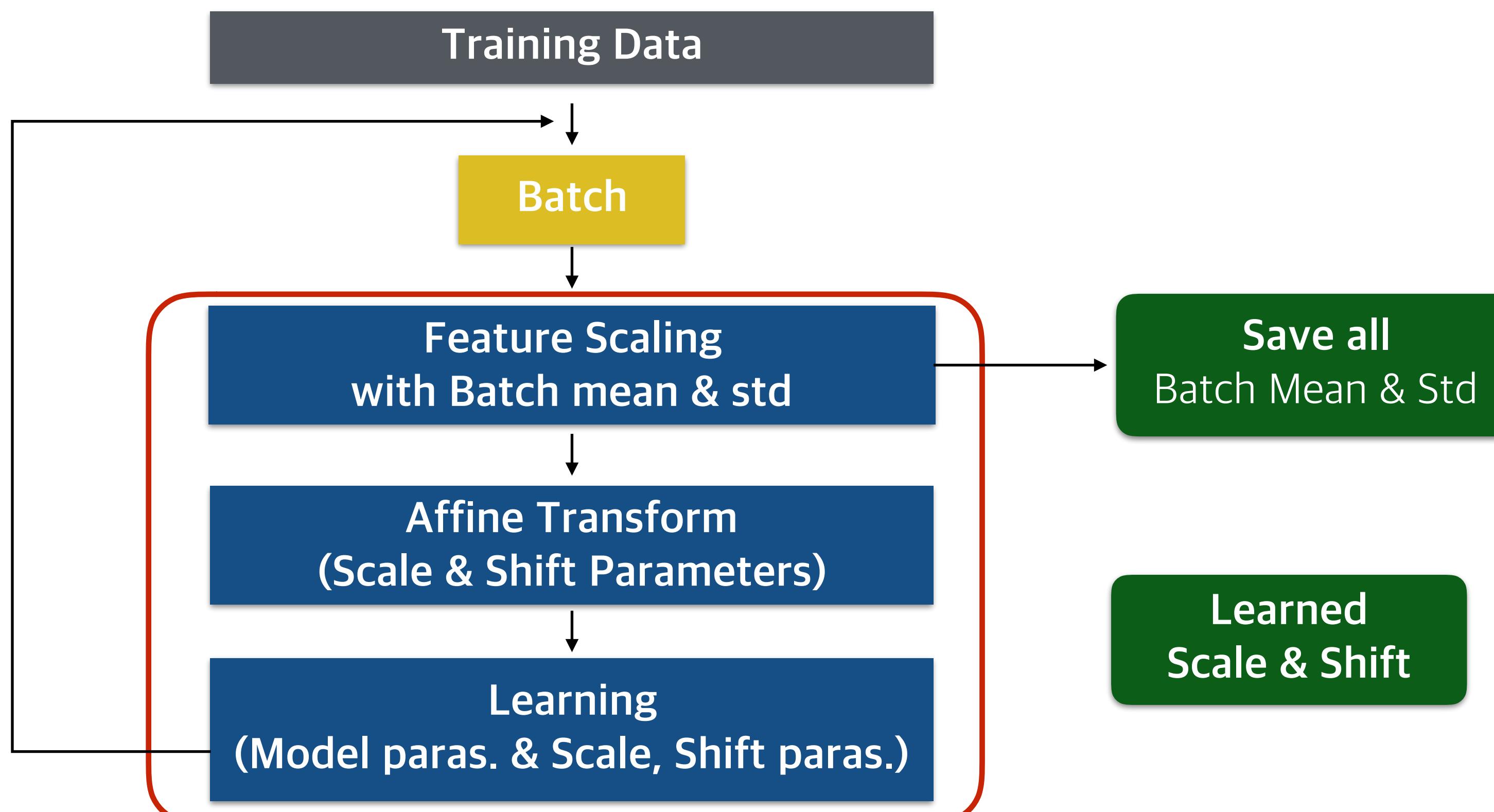
(오른쪽 논문에서는 이를 정리하여 표기한 것!)

# 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

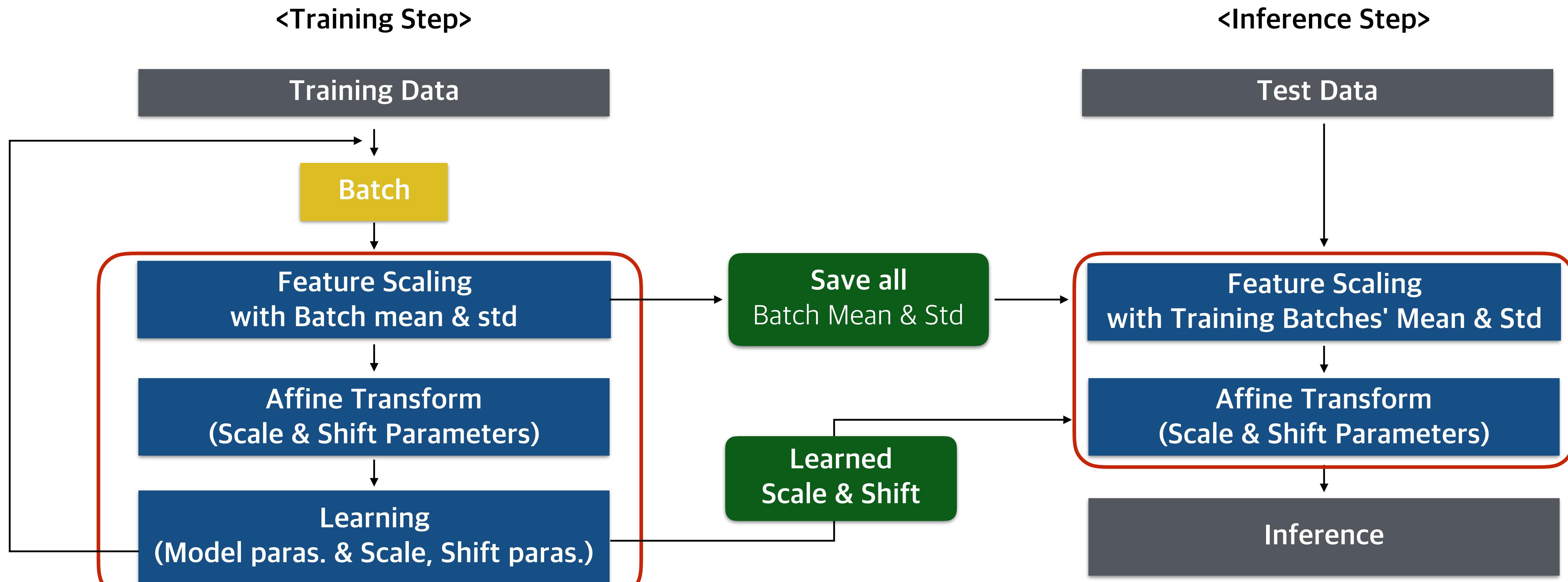
### <Training Step>



# Batch Normalization Algorithm Summary

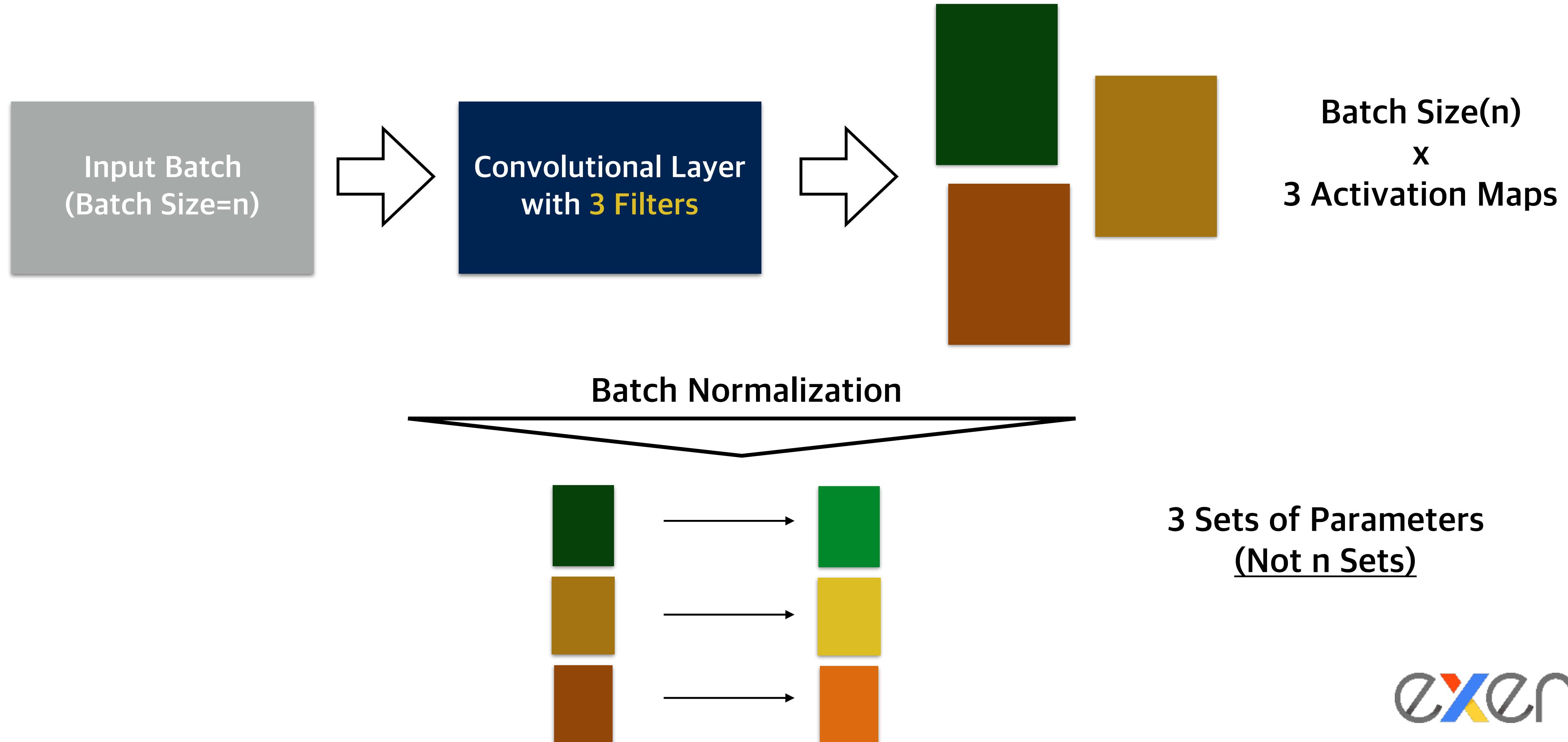
## □ Batch Normalization 알고리즘의 순서를 정리하면 아래와 같음

- 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

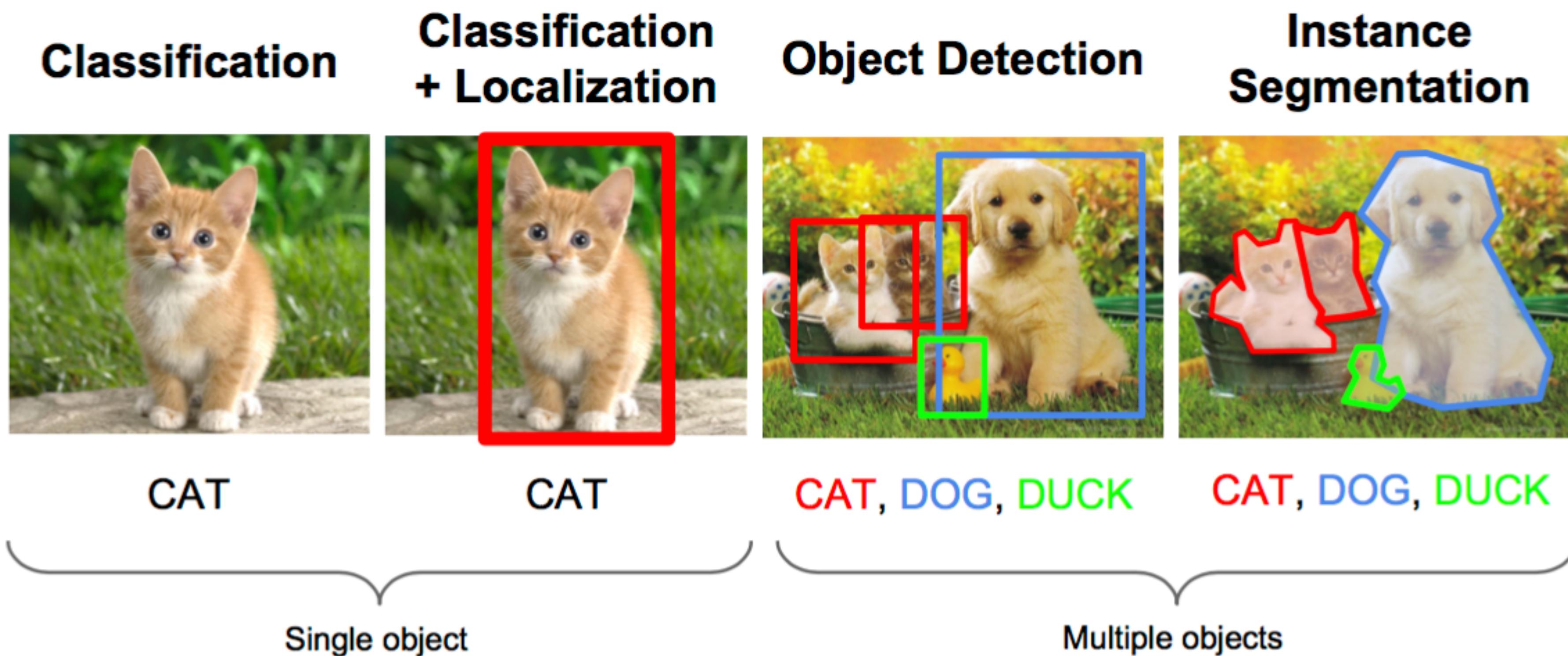
### - Remove Dropout & Regularization Parameter (Regularization)

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

# Applications of Conv. Net

# Conv. Net Applications in Vision Area

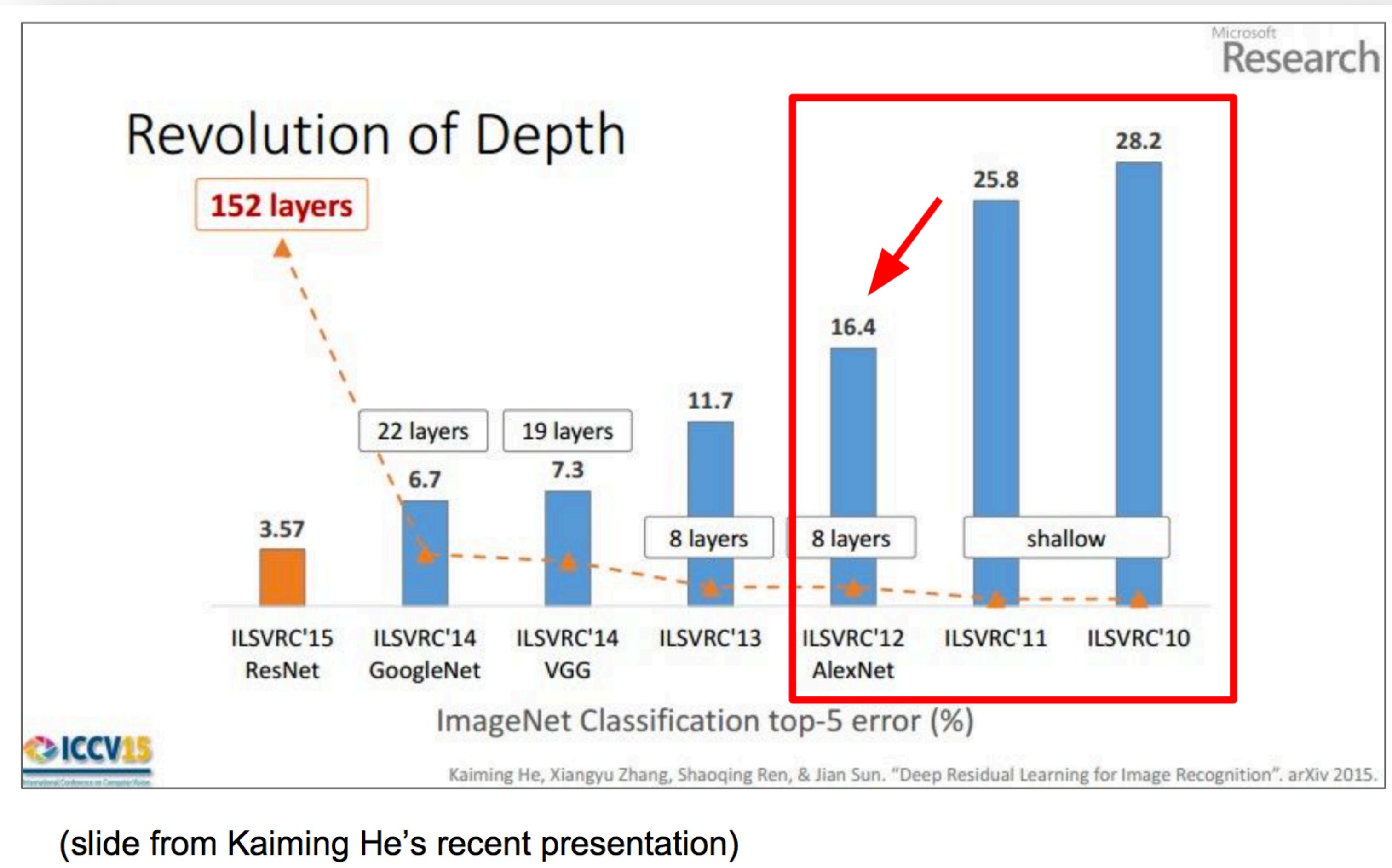
- ❑ Convolutional neural network can be utilized to various applications in vision area
- ❑ Many architectures were designed and improved its performance.



# Applications on Image Classification

- ❑ After AlexNet (2012 Winner), various architectures were designed with Conv. Net.
- ❑ With computational power of GPU, the model tends to be deeper and deeper.

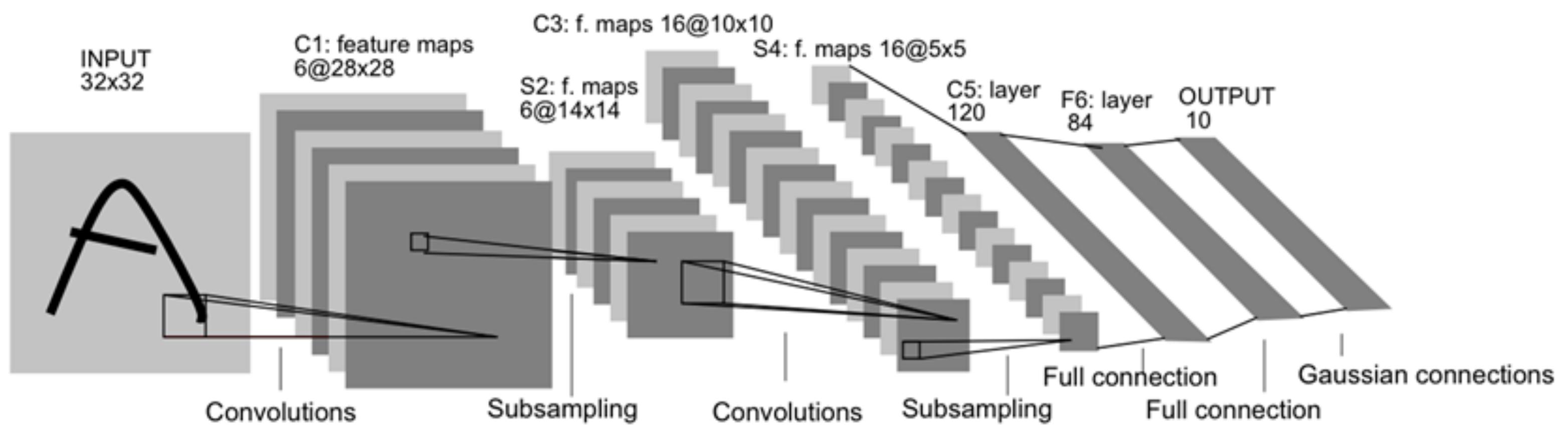
Classification



# LeNet-5

## □ LeNet-5

- Early convolutional neural network architecture (Le Cunn, 1998)
- LeNet-5 was used in order to recognize handwritten digit images.



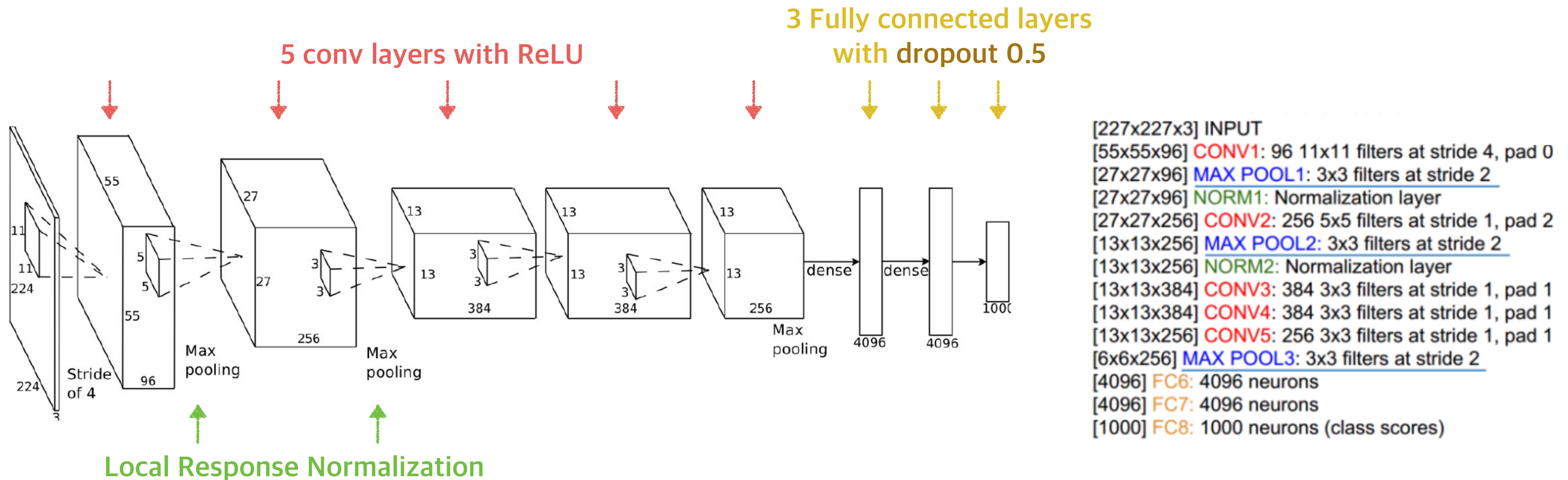
- **Sigmoid Activation**
- **Subsampling means only reduction of size, not max pooling**

An early (Le-Net5) Convolutional Neural Network design, LeNet-5, used for recognition of digits

# AlexNet (The ILSVRC 2012 Winner)

## ❑ AlexNet (Krizhevsky et al. 2012)

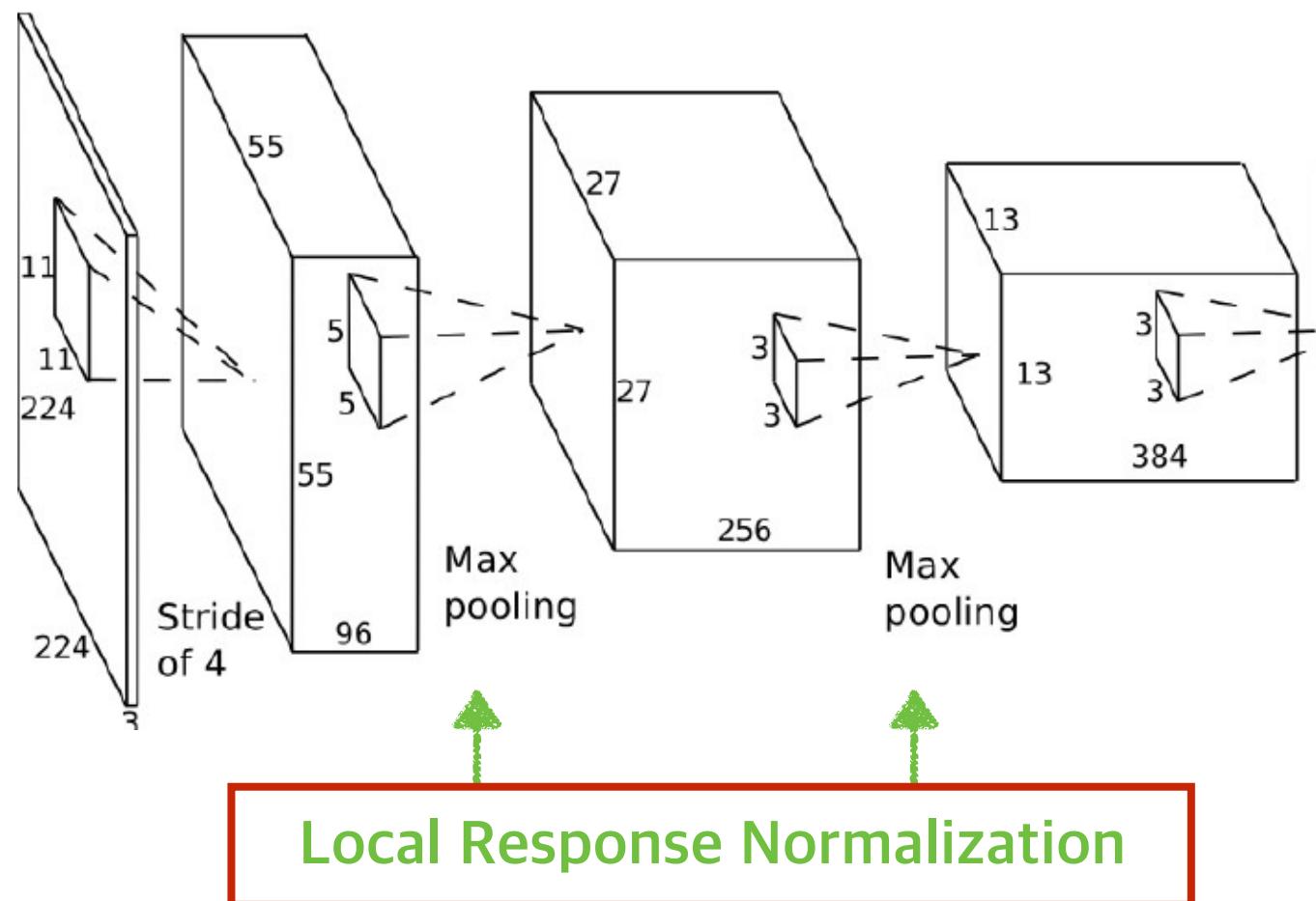
- No difference in the view of network components, but advance of computing technology
- ReLU function is used for activation (first use)
- Local Response Normalization (helps generalization)
- Drop Out (helps generalization)



# AlexNet (The ILSVRC 2012 Winner)

## □ Local Response Normalization

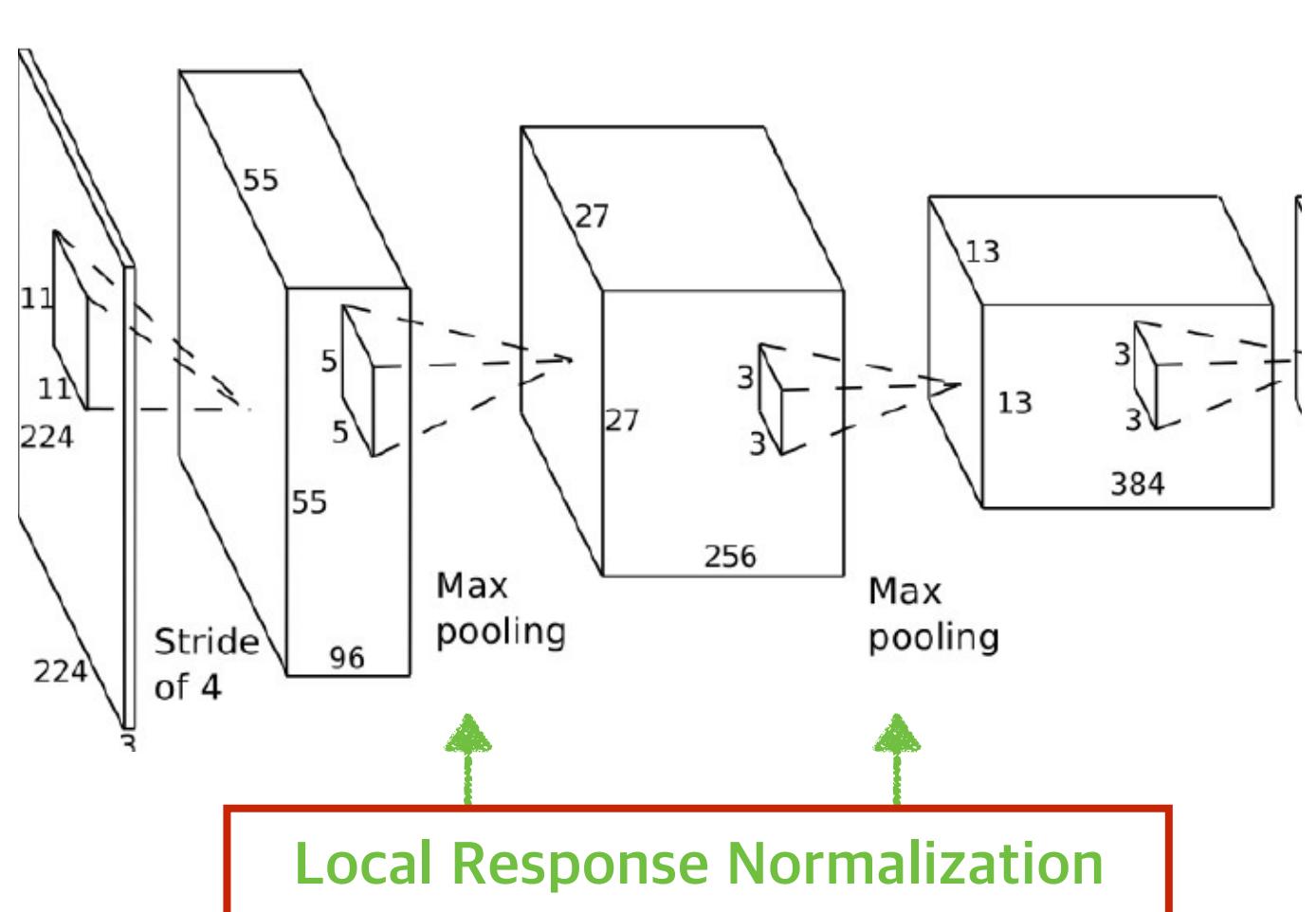
- ReLU activation을 이용하면 sigmoid, tanh 함수 이용시와 다르게 입력값을 정규화할 필요는 없음
- 실험적으로 normalization이 모델 일반화(generalization)에 도움이 되는 것을 확인해 local response normalization 적용



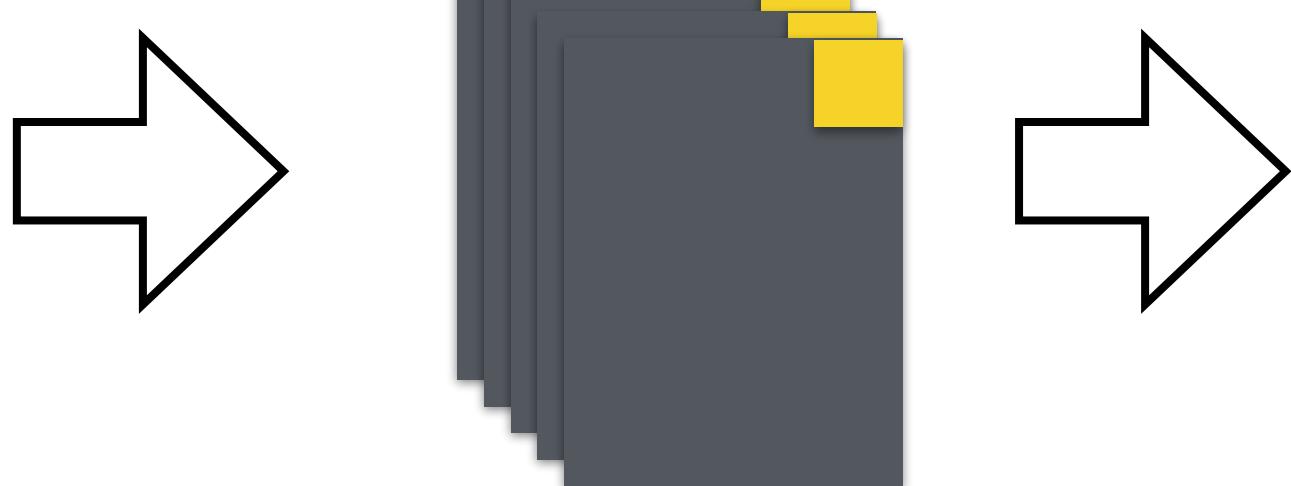
# AlexNet (The ILSVRC 2012 Winner)

## □ Local Response Normalization

- ReLU activation을 이용하면 sigmoid, tanh 함수 이용시와 다르게 입력값을 정규화할 필요는 없음
- 실험적으로 normalization이 모델 일반화(generalization)에 도움이 되는 것을 확인해 local response normalization 적용



Spatial Position을 기준으로  
Local Normalization



< i 번 째 필터의 x,y 위치의 값 >

$$b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta$$

$k = 2, n = 5, \alpha = 10^{-4}$ , and  $\beta = 0.75$

(validation set을 통해 실험적으로 도출)

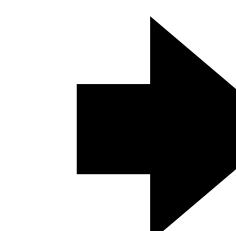
# AlexNet (The ILSVRC 2012 Winner)

## □ Local Response Normalization

- ReLU activation을 이용하면 sigmoid, tanh 함수 이용시와 다르게 입력값을 정규화할 필요는 없음
- 실험적으로 normalization이 모델 일반화(generalization)에 도움이 되는 것을 확인해 local response normalization 적용



Lateral Inhibition  
(Generalization)

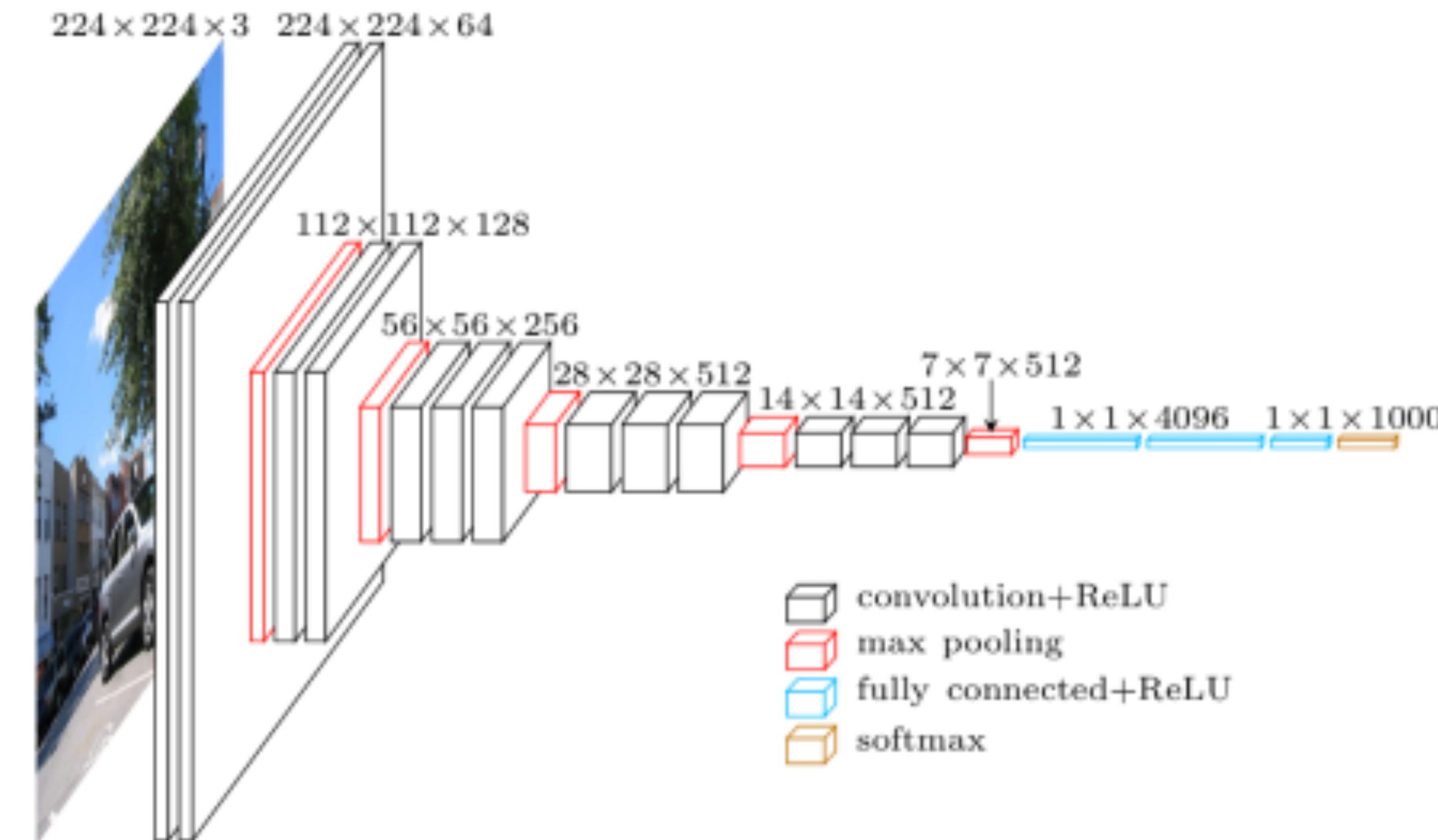


Error Rate  
Performance Improvement  
(top-1: -1.4%, top-5: -1.2%)

# VGG (The ILSVRC 2014 2nd Winner)

## ❑ Very Deep CNN (Simonyan et al., 2014) or VGG (Visual Geometry Group)

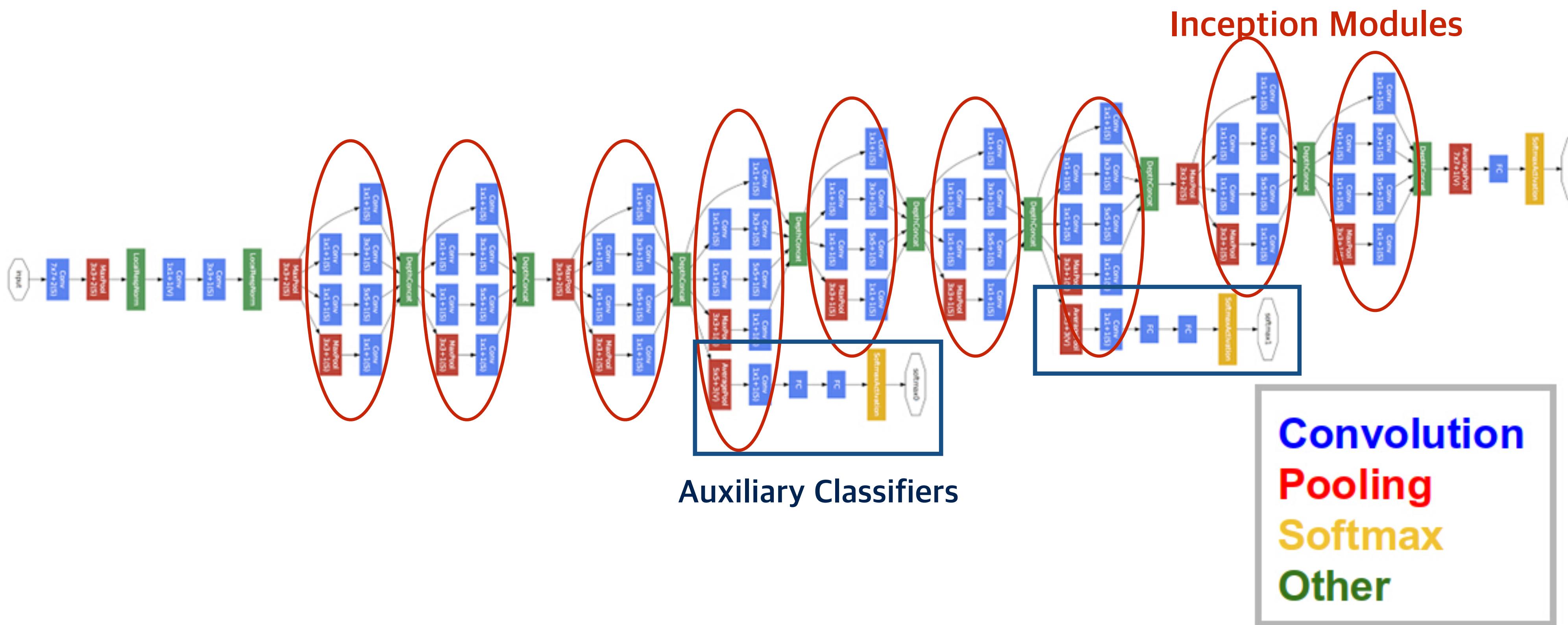
- VGG는 작은 크기의 convolutional filter( $3 \times 3$ , stride=1)를 사용하는 대신, 매우 깊게 모델을 구조화함  
(VGG: 16 or 19 layers, AlexNet: 8 layers)
- 3-4 개의 convolutional layers를 연속으로 위치시킨 후, pooling layer를 위치시킴
- 다른 모델에 비해 **매우 간단한 구조**로 이해 및 구현하기 쉽고 성능이 높아 대중적으로 많이 사용됨



# GoogLeNet” (The ILSVRC 2014 Winner)

## □ GoogLeNet (called “Inception”, Szegedy et al., 2015)

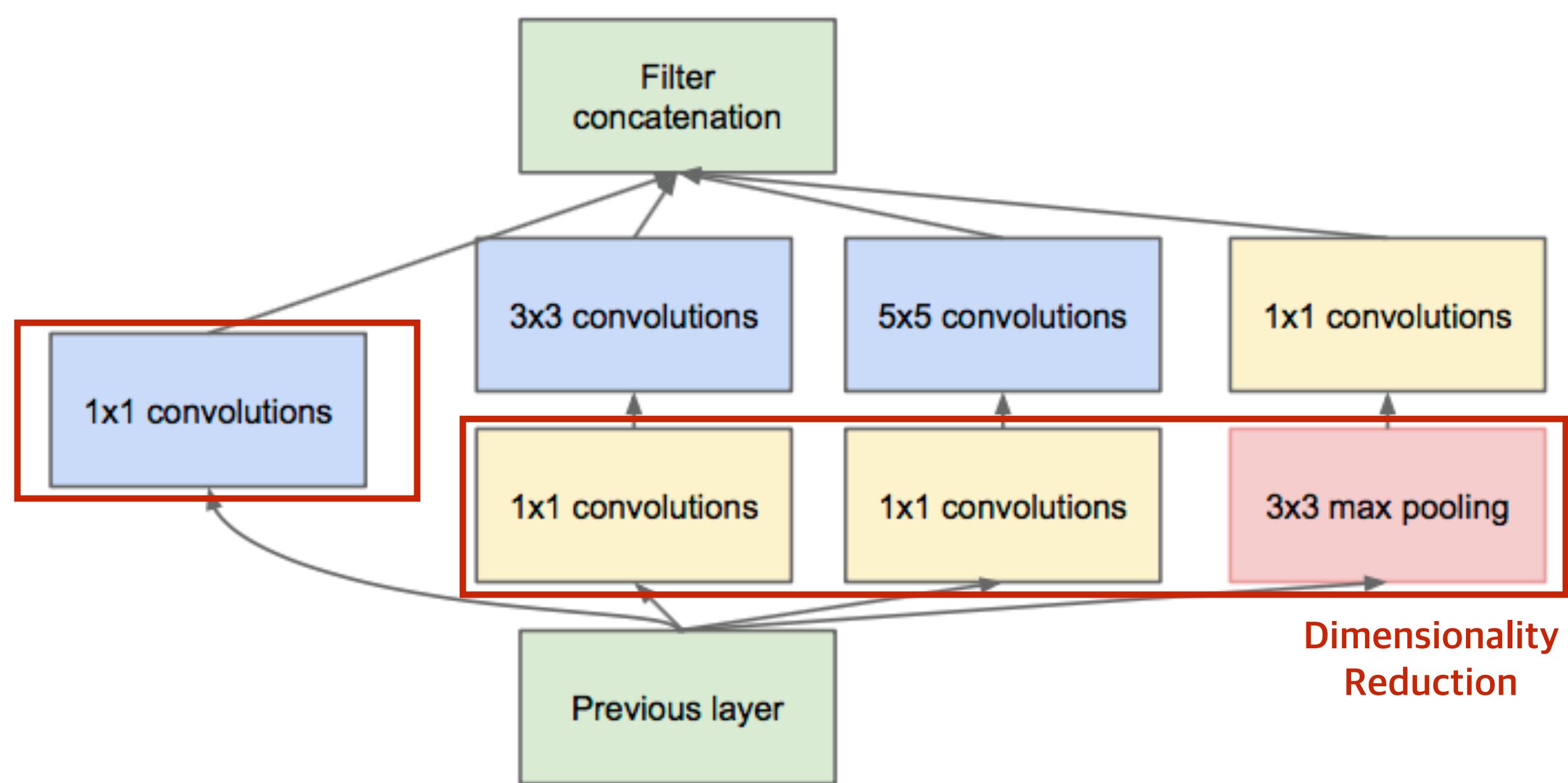
- 22개 layer 구조로 ILSVRC 2014 우승 모델
- Layer by Layer 구조의 일반화로 네트워크 내 네트워크를 배치하는 “Inception” 모듈을 사용
- 깊은 구조의 학습을 위해 Auxiliary Classifiers & Loss (보조적인 분류기와 손실)를 사용



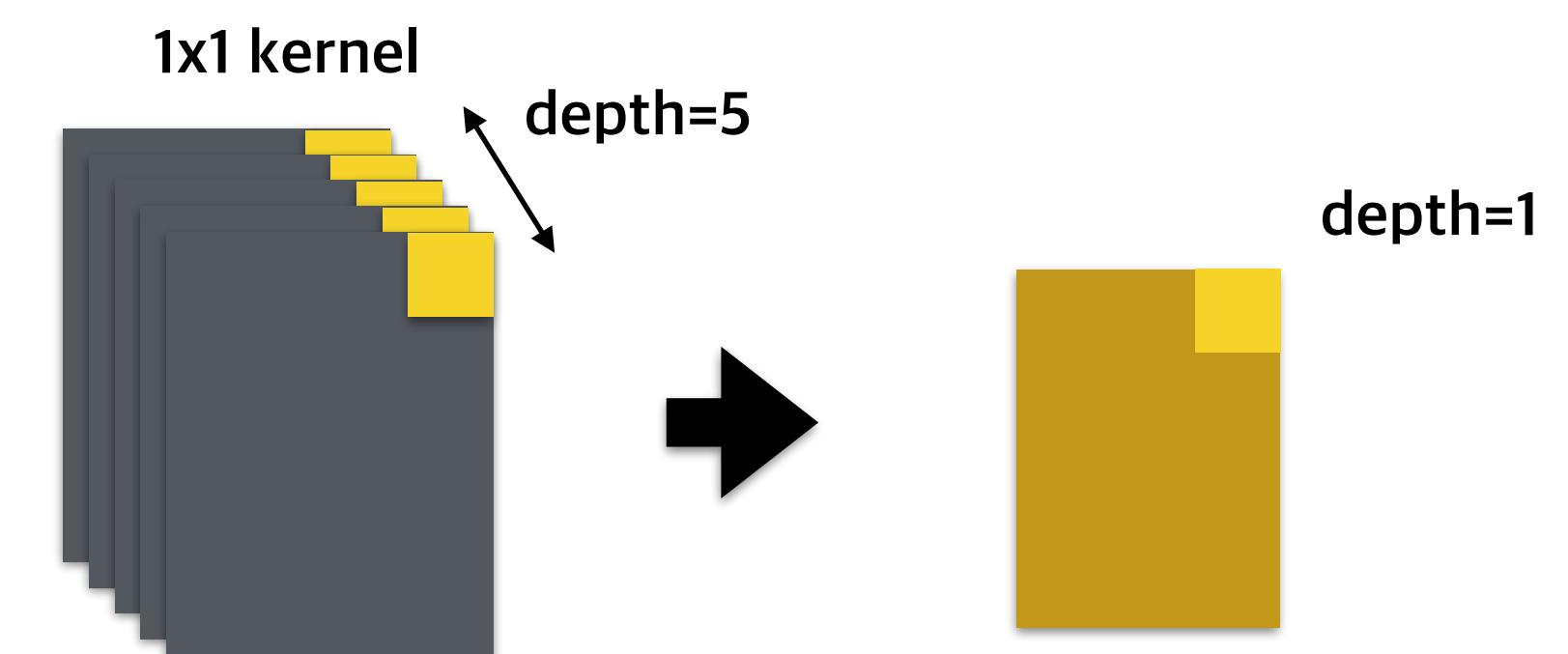
## ❑ Inception Module

- Feature map을 구성하기 위해 단일 크기의 필터(kernel)을 사용하는 것이 아닌, 여러 크기의 필터로 구성된 네트워크 사용
- 여러 크기의 필터로 구성된 convolution 연산 이 후, 모든 결과를 이어 붙여(concatenate) 새로운 feature map 구성
- 파라미터가 극심하게 커지는 계산 상의 문제를 해결하기 위해 **dimensionality reduction (1x1 convolution, pooling)**을 우선적으로 사용함

<Inception Module>

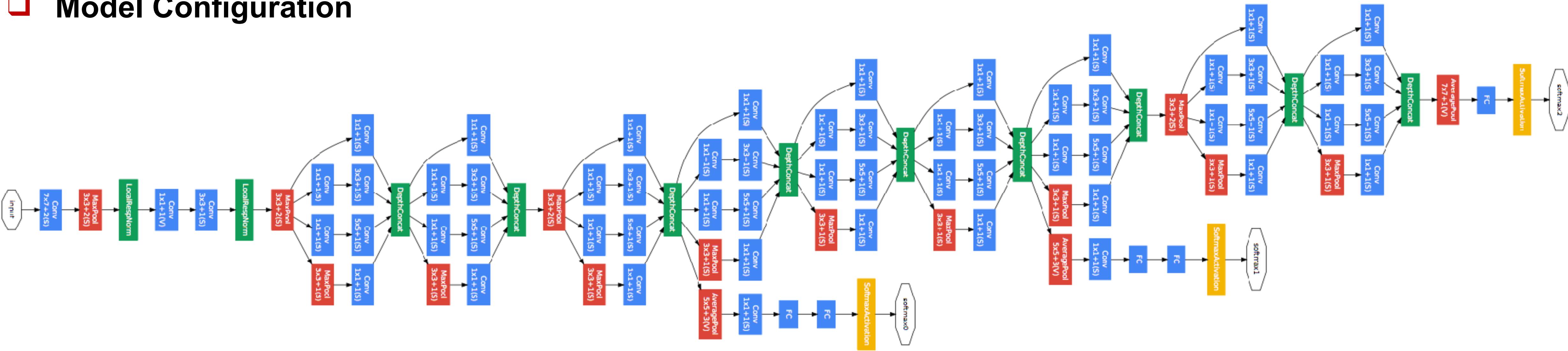


<1x1 Convolution Operation>



# GoogLeNet (Cont'd)

## Model Configuration



type	patch size/ stride	output size	depth	#1x1	#3x3 reduce	#3x3	#5x5 reduce	#5x5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0		64	96	128	16	32	159K	128M
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Table 1: GoogLeNet incarnation of the Inception architecture

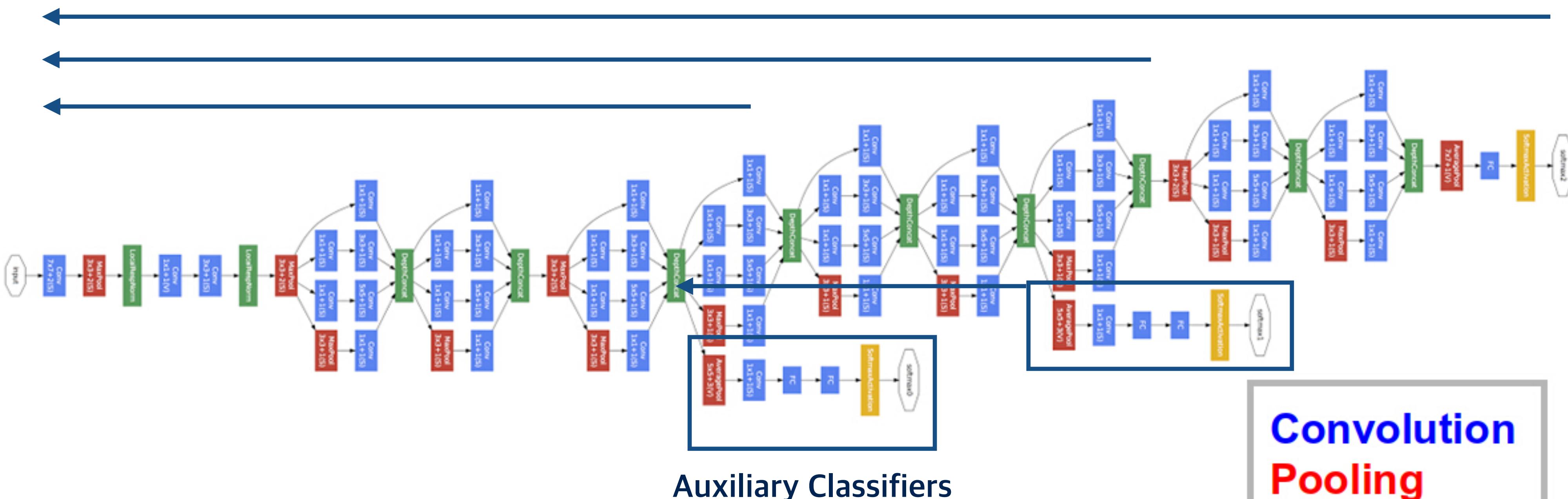
Height & Width 지속적으로 감소  
Depth(Channel) 지속적으로 증가

# GoogLeNet (Cont'd)

# Auxiliary Classifiers

- 모델의 구조가 깊어질수록 마지막 layer의 error signal은 앞 쪽 layer에 전달되기 힘듦 (gradient vanishing problem)
  - 위의 문제를 해결하기 위해 모델 중간에 Auxiliary Classifier를 배치하고 Loss를 추가로 계산하여, 학습에 활용
  - 학습 과정에서 사용되는 Loss는 모든 Classifier들의 loss의 가중치 합이며, 테스트 과정에서는 마지막 layer의 결과만 사용

# Error Signals in backpropagation



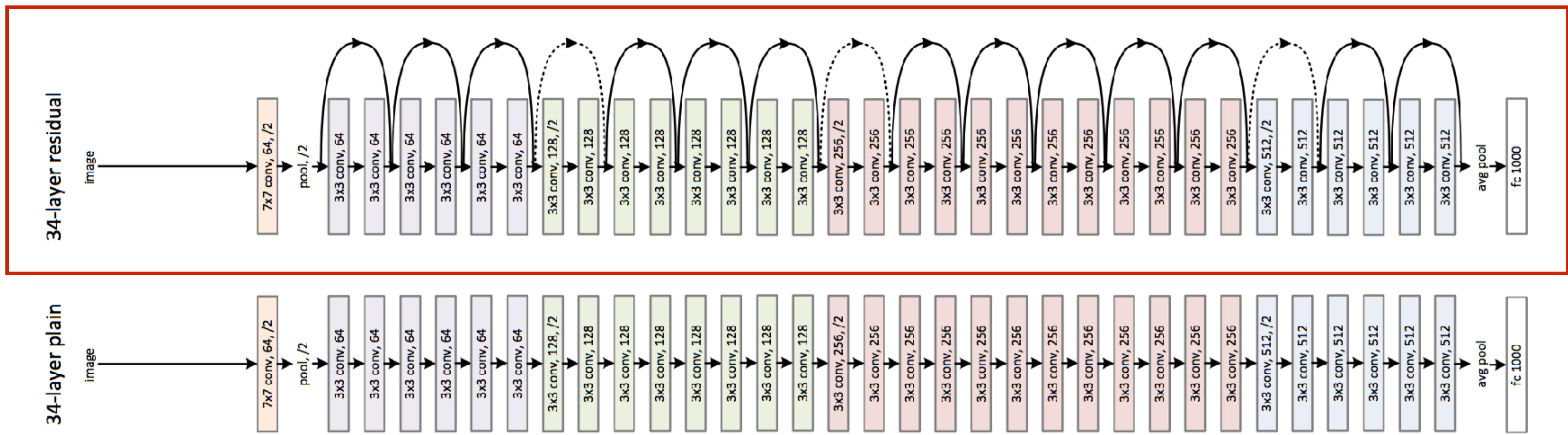
**Convolution**  
**Pooling**  
**Softmax**  
**Other**

# ResNet (The ILSVRC 2015 Winner)

## □ ResNet (Deep Residual Network, Kaiming et al., MS Research)

- ResNet은 기존 모델들과 다르게 매우 많은 hidden layer를 포함하는 모델 (152 layers)
- “Skip connection” 구조를 통하여 기존 feature map이 아닌 입력값과 사이의 **잔차(residual)**를 학습하는 방식으로 구현
- Skip connection을 통해 매우 많은 hidden layer를 사용하더라도, gradient vanishing 현상을 극복하고 학습이 가능

<ResNet and Plain Architectures>



# Background of ResNet

- ▣ 기존에 제안된 다양한 방식이 깊은 구조의 딥러닝 모델에서 gradient vanishing 문제 등을 해결하였음  
(ReLU activation function, Batch Normalization, Initialization techniques...)
- ▣ 하지만 MS Research 팀은 그 깊이가 **매우 깊어질 경우**에는 여전히 문제가 있다는 것을 제기함

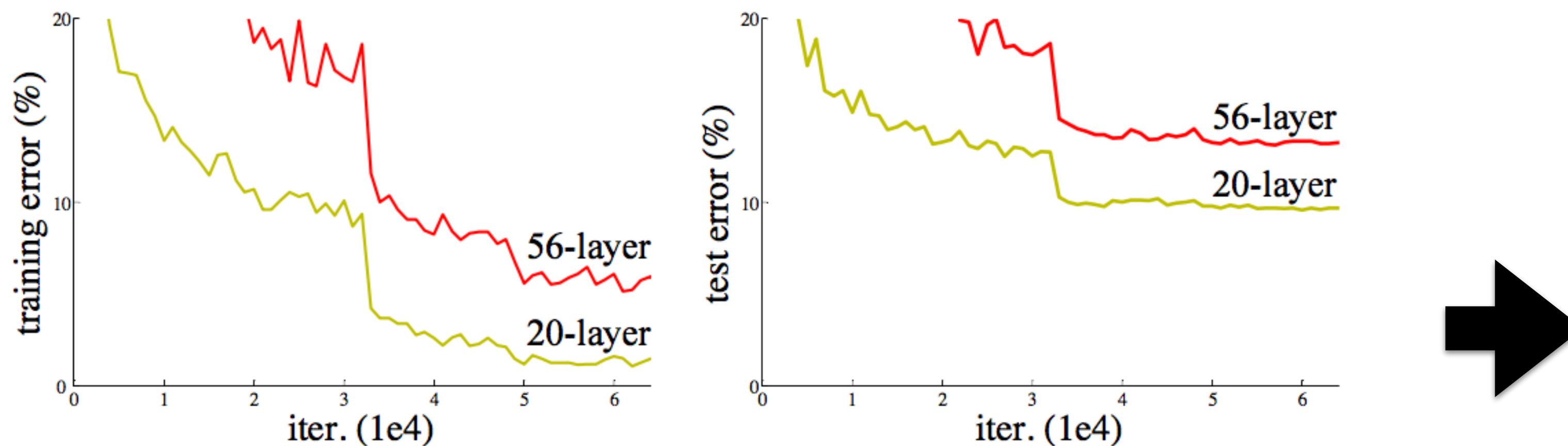


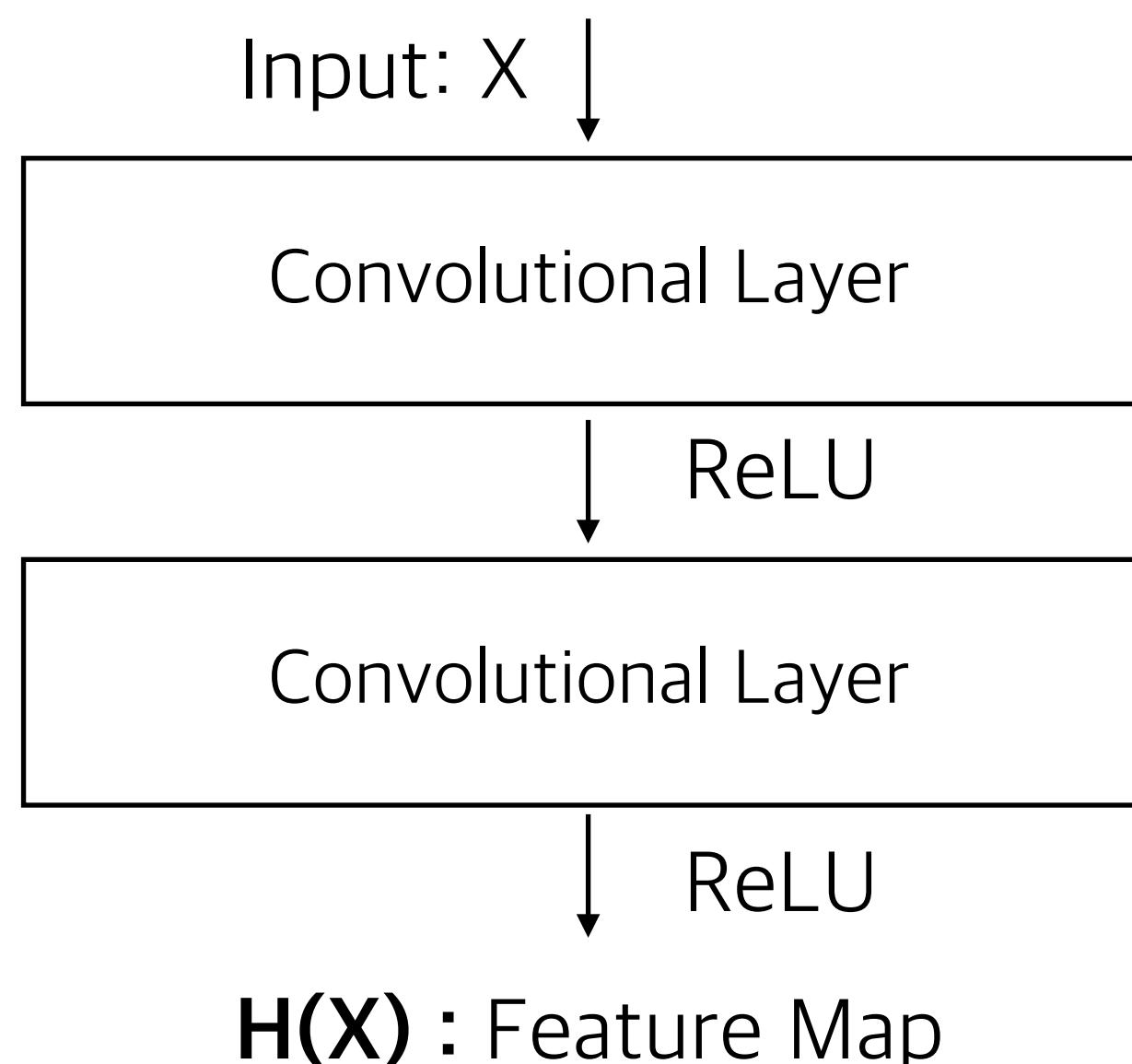
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

VGG의 층 수를 무수히 높일 경우  
오히려, 그 **성능이 저하**되는 것을 확인

# Skip Connection in ResNet

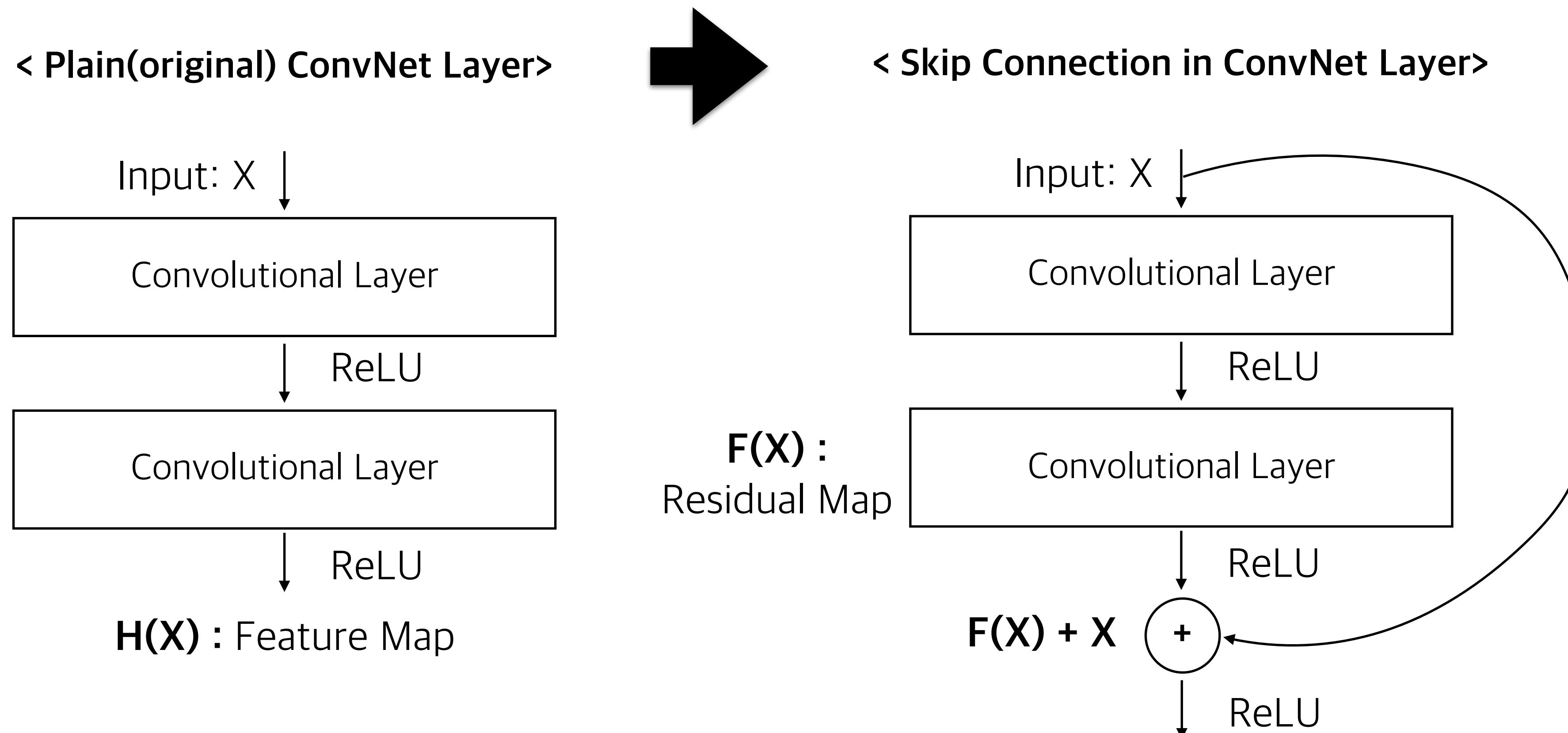
- ResNet은 기존의 학습 대상을 전환하여, 같은 작업을 반대로 생각하는 것에서 시작
- Skip Connection을 통해 모델의 학습 대상을 입력을 잘 표현하는 feature map이 아닌, 입력과의 차이를 학습

< Plain(original) ConvNet Layer>



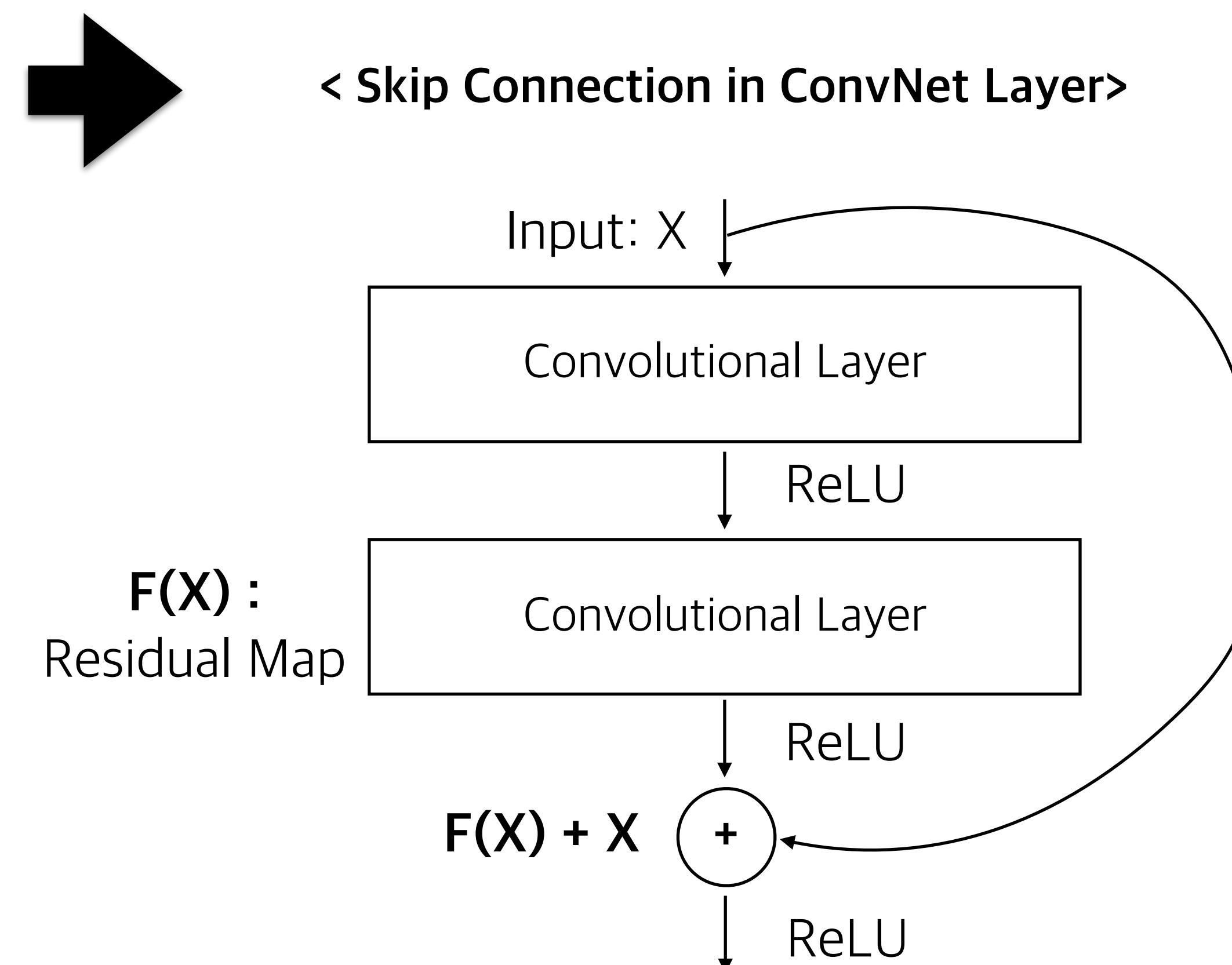
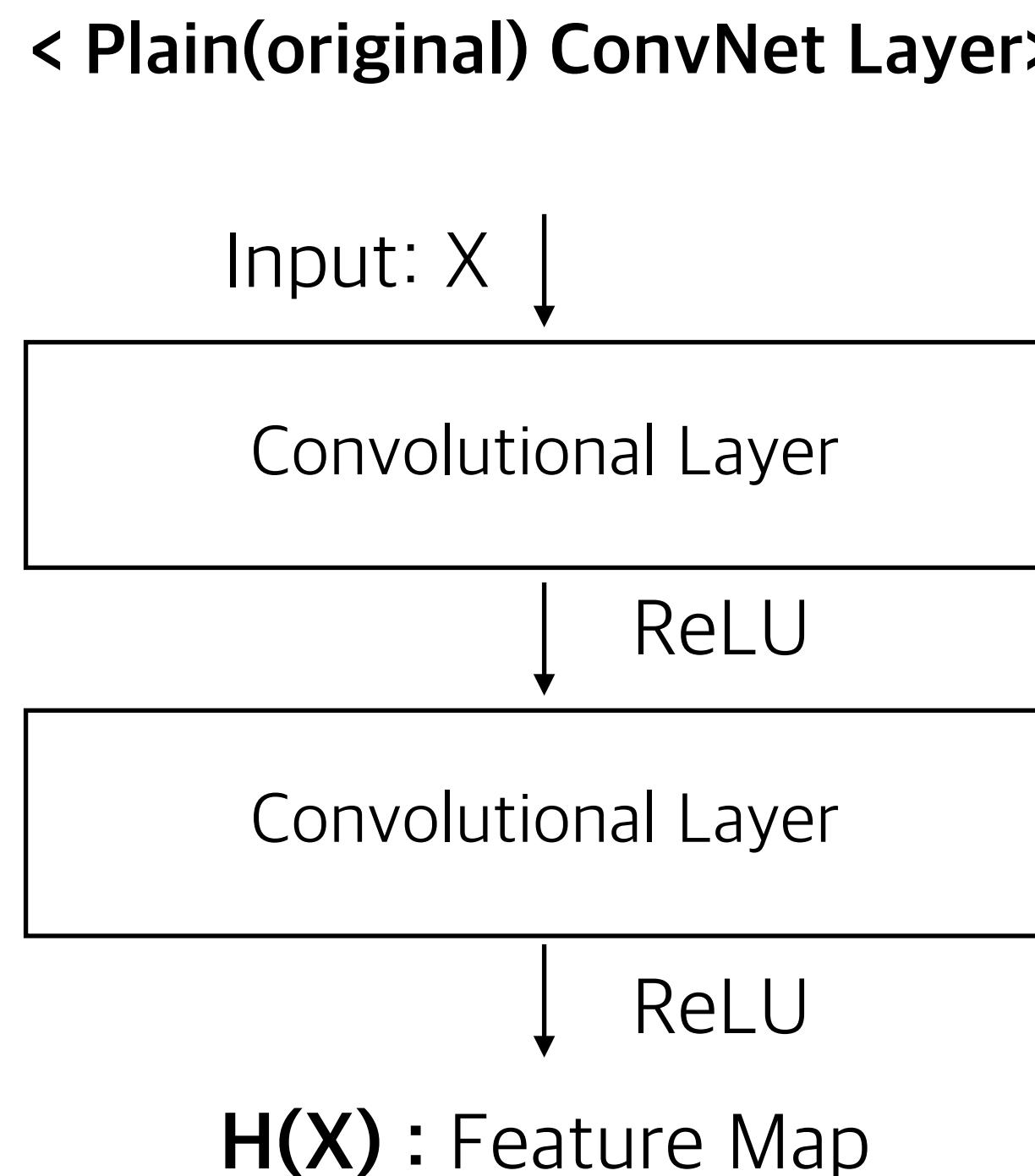
# Skip Connection in ResNet

- ResNet은 기존의 학습 대상을 전환하여, 같은 작업을 반대로 생각하는 것에서 시작
- Skip Connection을 통해 모델의 학습 대상을 입력을 잘 표현하는 feature map이 아닌, 입력과의 차이를 학습



# Skip Connection in ResNet

- ResNet은 기존의 학습 대상을 전환하여, 같은 작업을 반대로 생각하는 것에서 시작
- Skip Connection을 통해 모델의 학습 대상을 입력을 잘 표현하는 feature map이 아닌, 입력과의 차이를 학습



If we know  $H(X)$  and  $X$ ,  
also know Residual  
 $F(X) = H(X) - X$

Then, Learn not  $H(X)$  but  $F(X)$

If we know  $F(X)$  and  $X$ ,  
also know  $H(X) = X + F(X)$

# Simple Experiment Result Comparison

## □ 18 layers & 32 layers 를 포함한 Plain 모델과 Skip Connection 모델에 대하여 ImageNet 데이터 실험

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	<b>25.03</b>

Table 2. Top-1 error (%), 10-crop testing) on ImageNet validation. Here the ResNets have no extra parameter compared to their plain counterparts. Fig. 4 shows the training procedures.

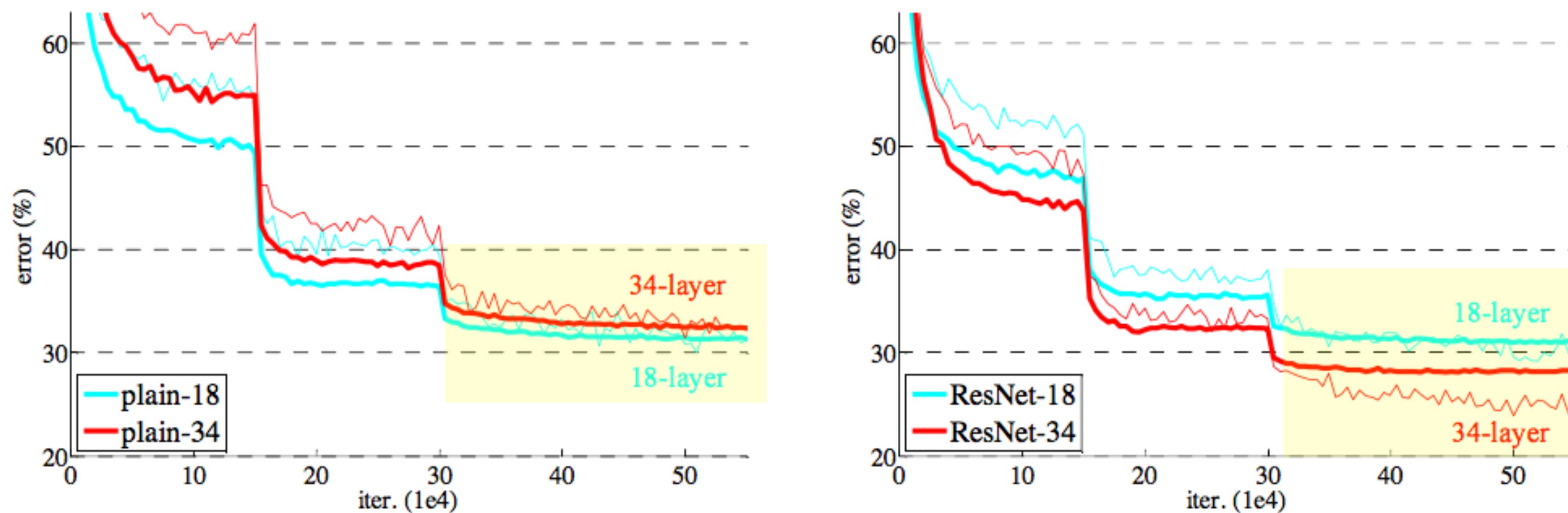
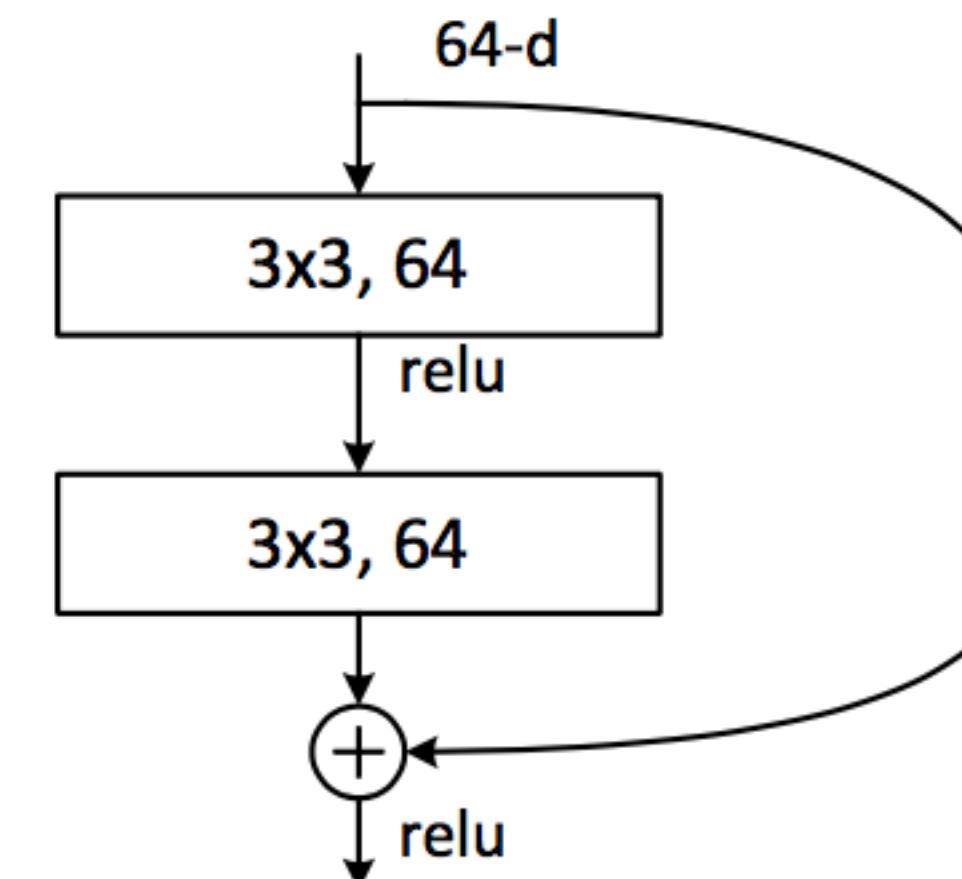


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

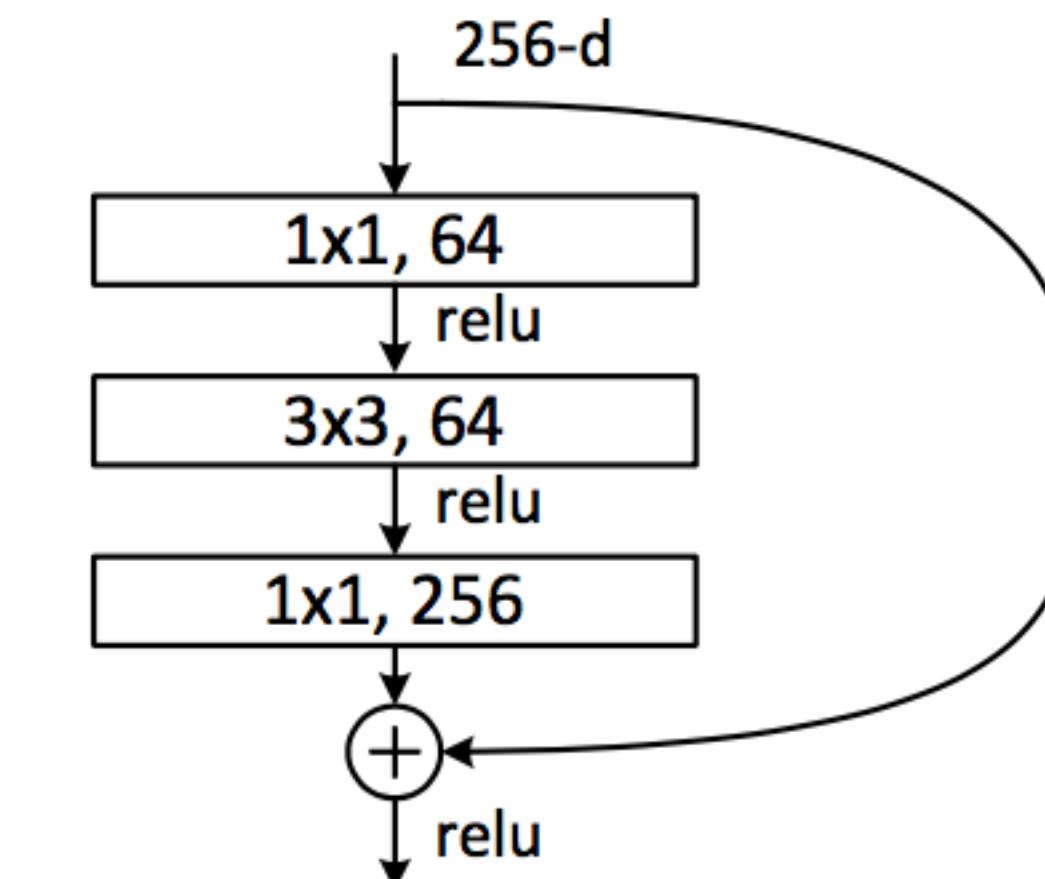
# Bottleneck Structure for Extremely Large Model

- 50 layers 이상의 매우 깊은 구조의 모델을 사용할 경우 Skip Connection 부분의 연산을 일부 수정
- Bottleneck 구조를 이용하며, **1x1 convolution**을 통해 차원 축소한 후 **3x3 convolution** 진행
- 이 후, 다시 **1x1 convolution**을 통해 입력값의 차원으로 재구성하고 **Skip Connection** 연산 진행

<기존 Skip Connection>



<50개 이상 Layer 사용시  
Skip Connection >



CIFAR-10 데이터셋에 대하여

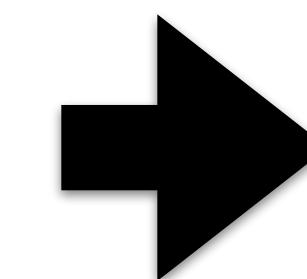
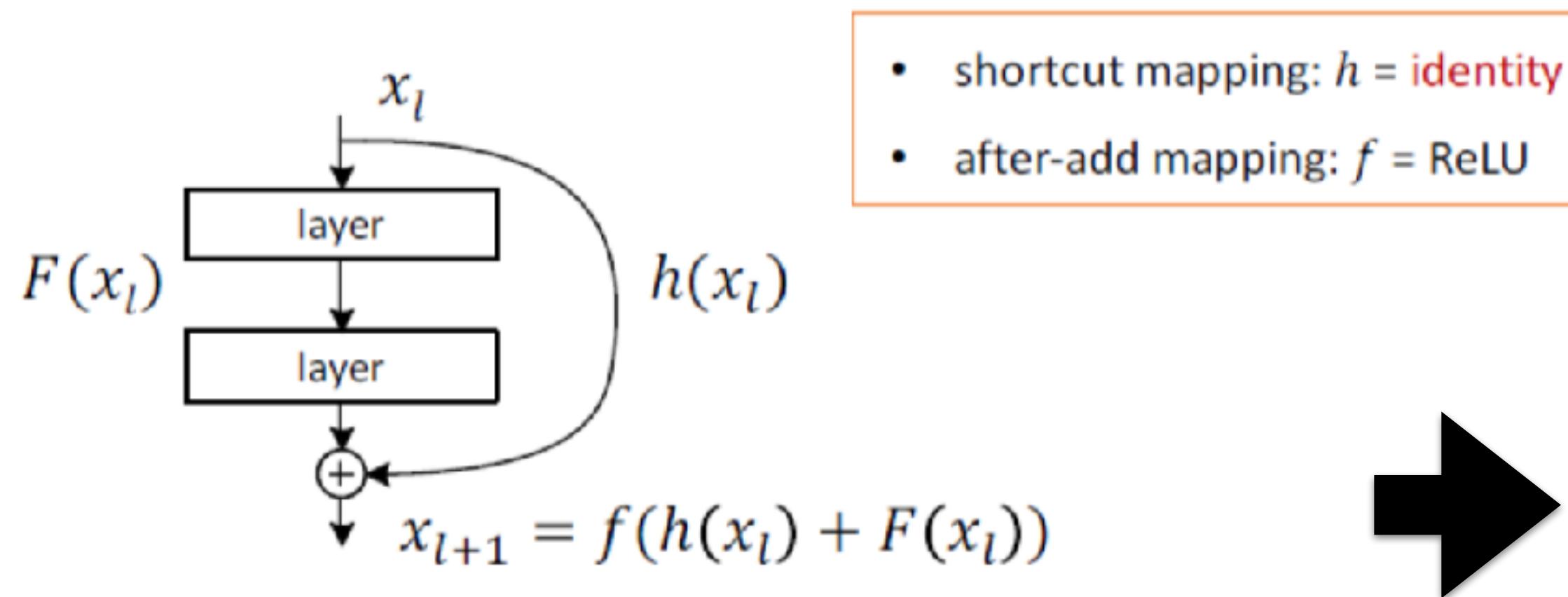
110개의 layer를 사용했을 때, 가장 높은 성능을 보임을 확인  
(20, 32, 44, 56, 110 layers 비교)

1000개 이상의 layer에 대하여도  
높은 성능의 결과 확인

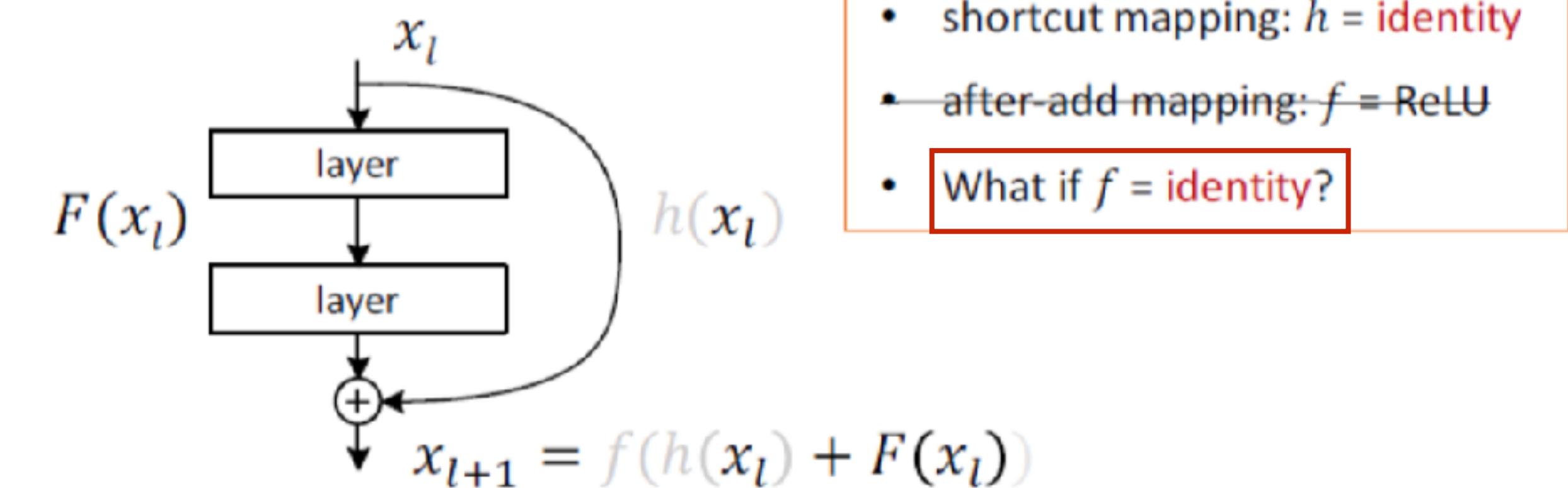
# Revision of Activation Location in ResNet

- ImageNet 대회 이 후, ResNet의 모델 일부를 수정함
- Skip Connection 이 후, ReLU 함수를 합성한 값을 새로운 입력으로 사용하지 않고,  $F(x) + X$  를 그대로 입력으로 사용

## <Original ResNet>

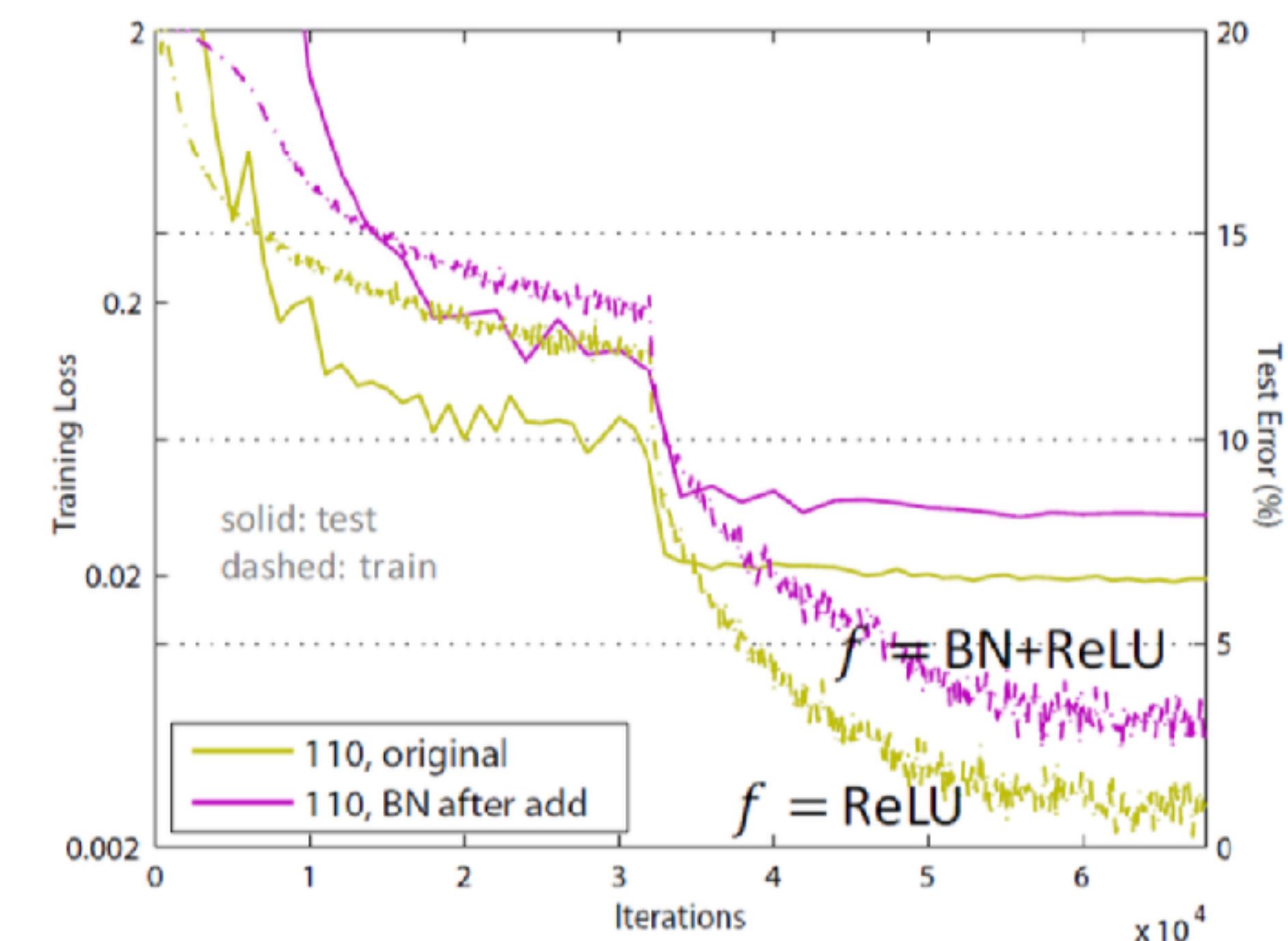
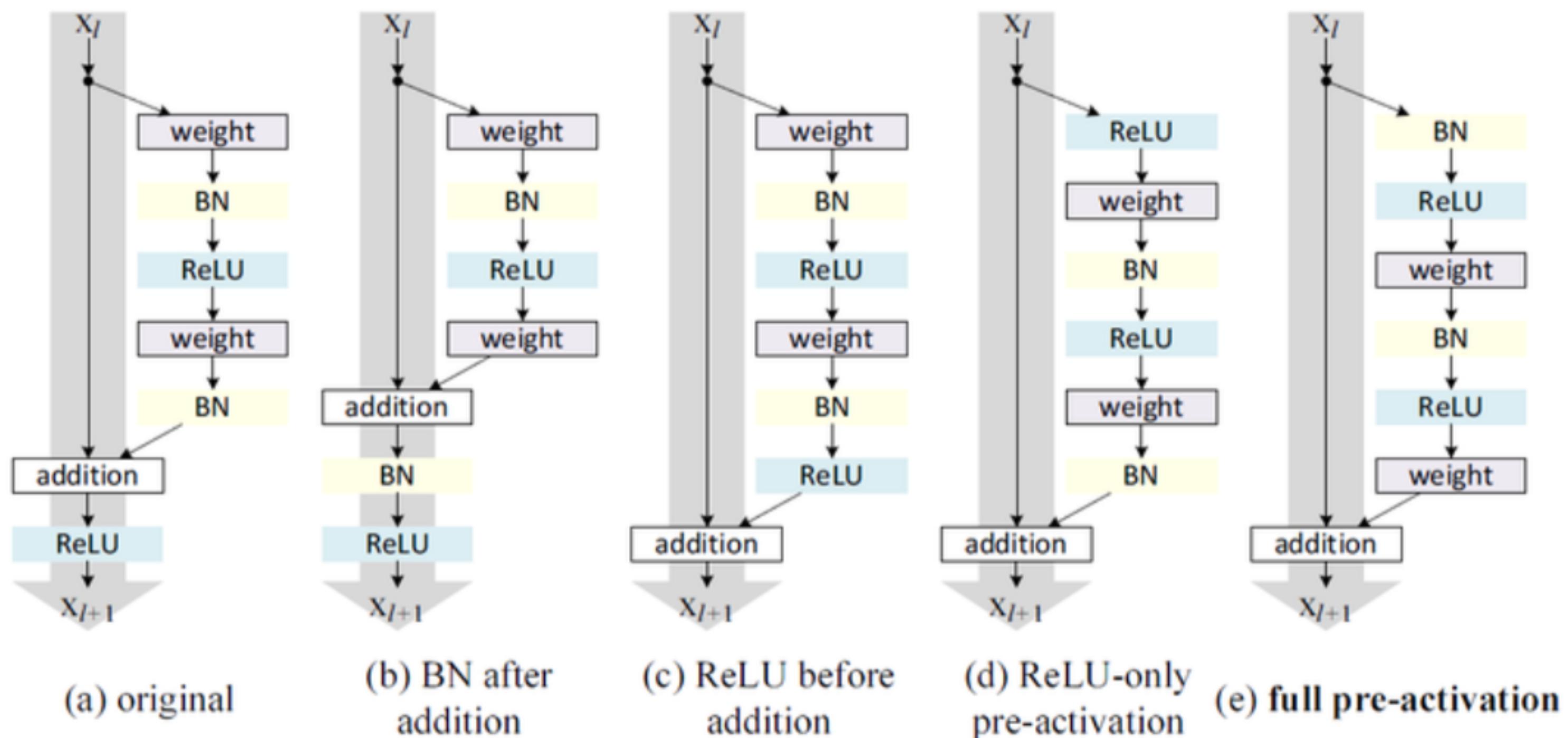


## <Revised ResNet>



# Full Pre-Activation in ResNet

- Activation 위치에 따라 성능 비교를 추가적으로 진행
- Full pre-activation 모델의 우수한 성능 확인



Now You Are  
Ready To Start  
Tensorflow!

# k-Nearest Neighbor

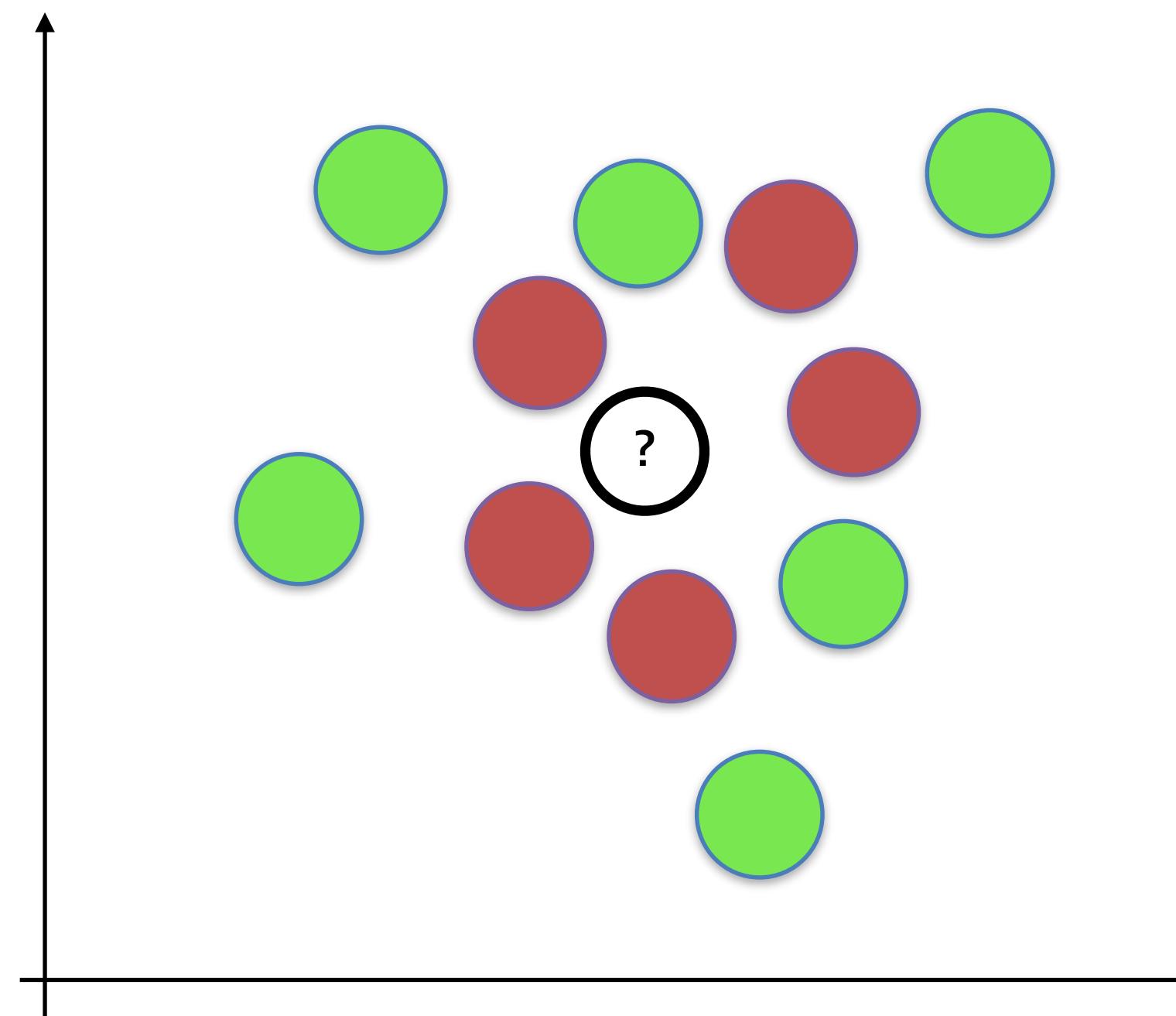
# Simple Example for Classification

## □ Simple Idea for Classification

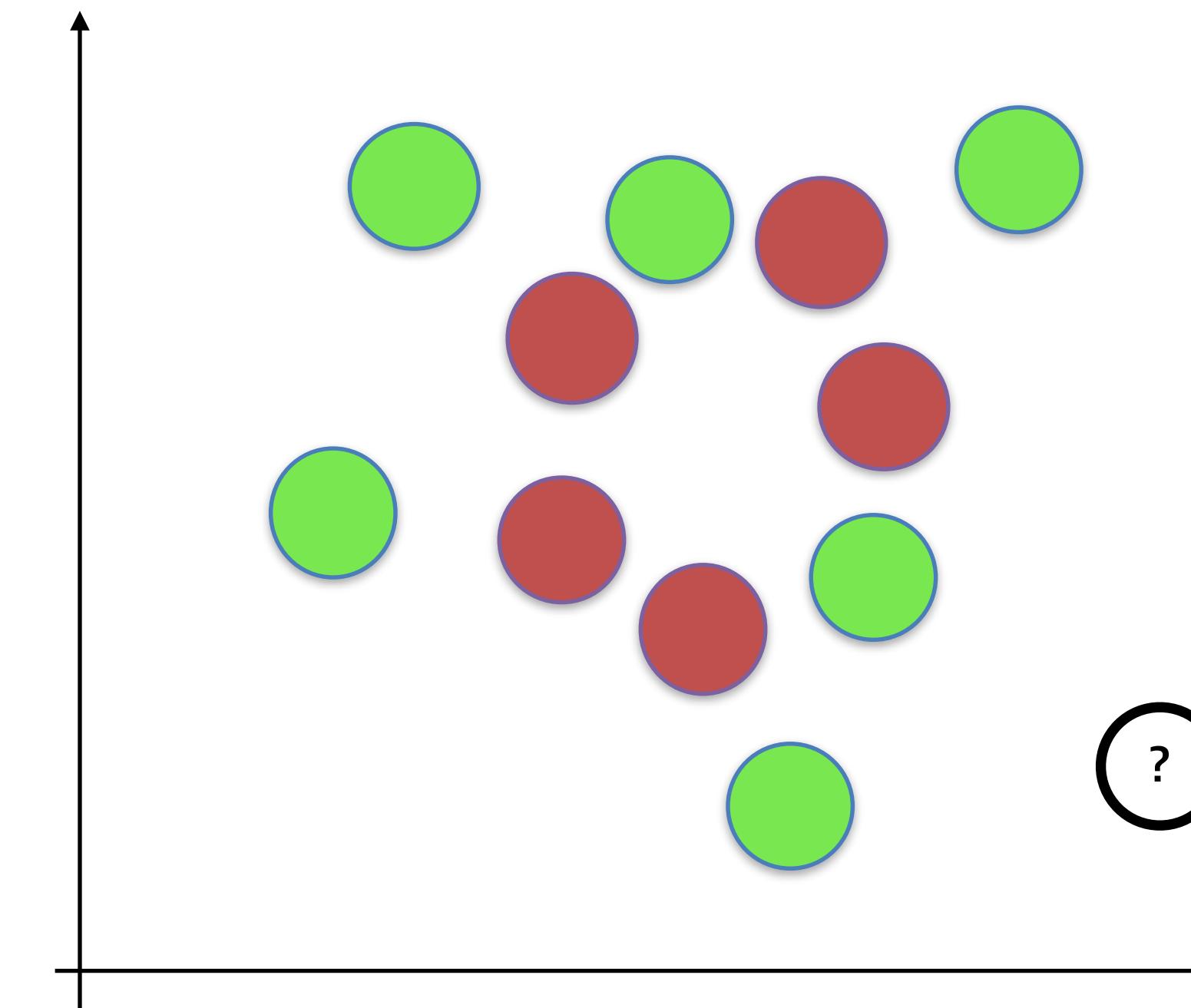
- 현재 가지고 있는 데이터는 위치에 따라 색깔(Green/Red)의 Class로 분류가 가능하다고 가정

## □ Class(색깔)을 모르는 임의의 점에 대하여 색깔을 분류하는 가장 간단한 방법은?

<Problem #1>



<Problem #2>



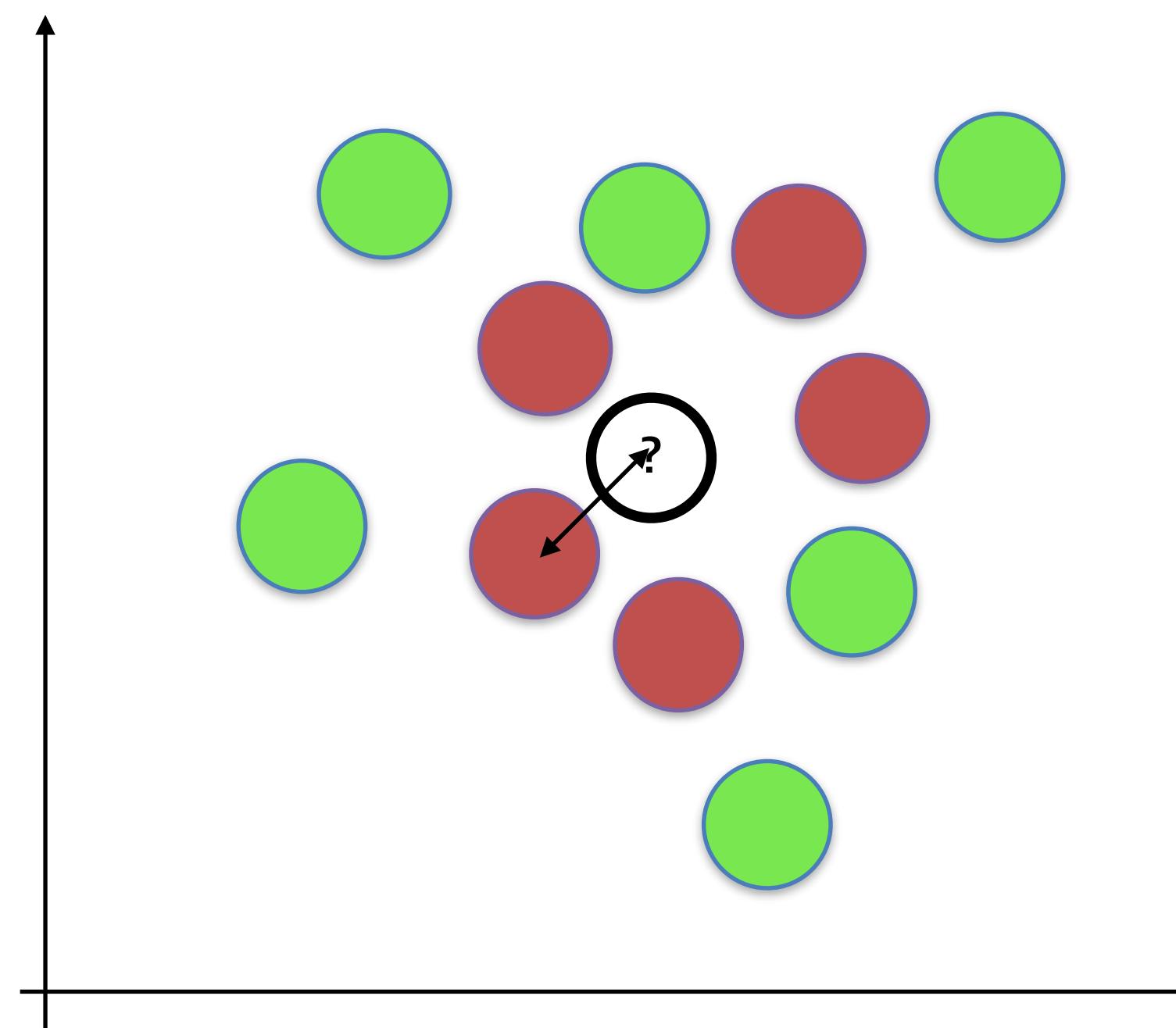
# Simple Example for Classification

## □ Simple Idea for Classification

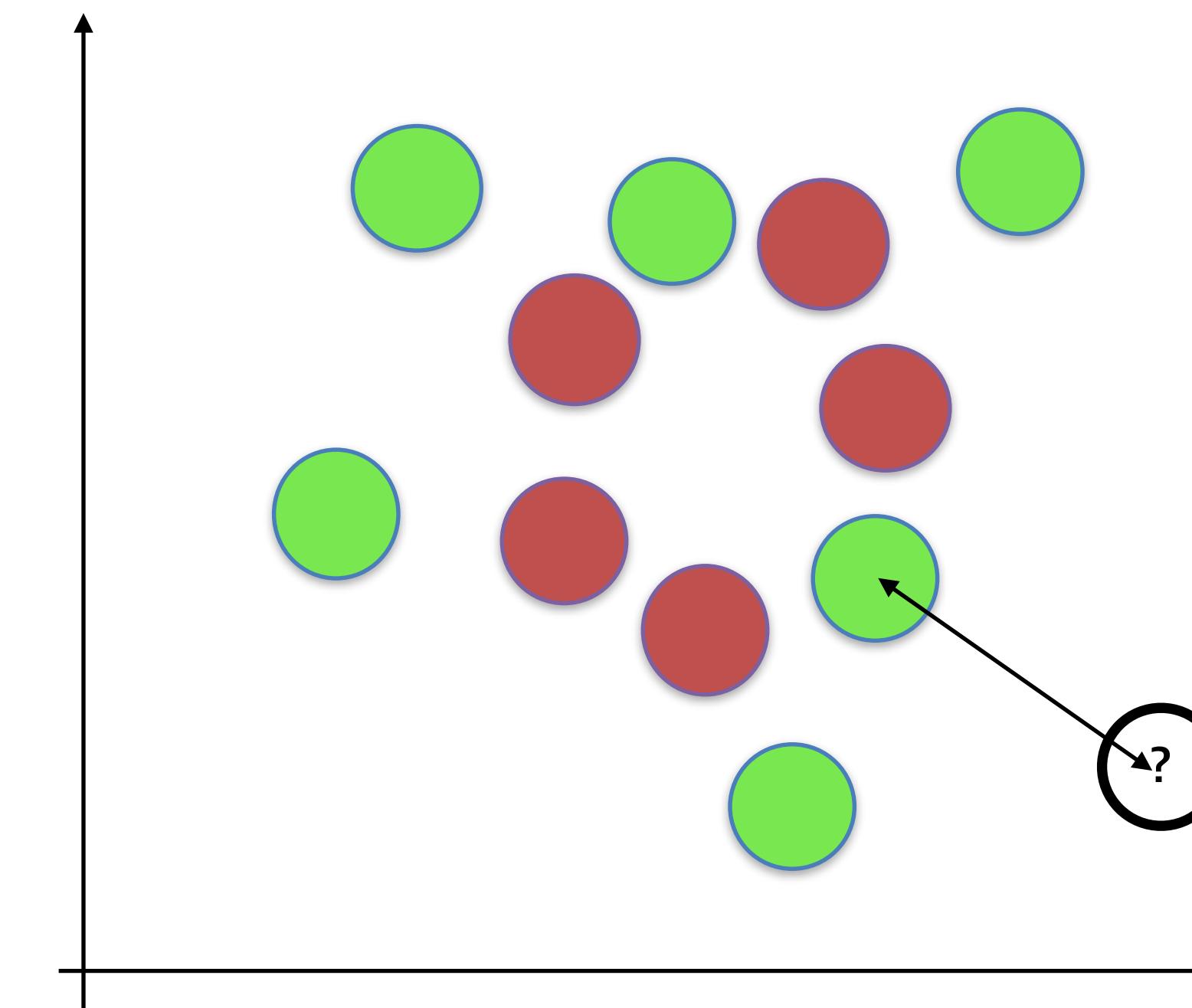
- 현재 가지고 있는 데이터는 위치에 따라 색깔(Green/Red)의 Class로 분류가 가능하다고 가정

## □ Class(색깔)을 모르는 임의의 점에 대하여 색깔을 분류하는 가장 간단한 방법은? 가장 가까운 점의 색깔을 부여

<Problem #1>



<Problem #2>



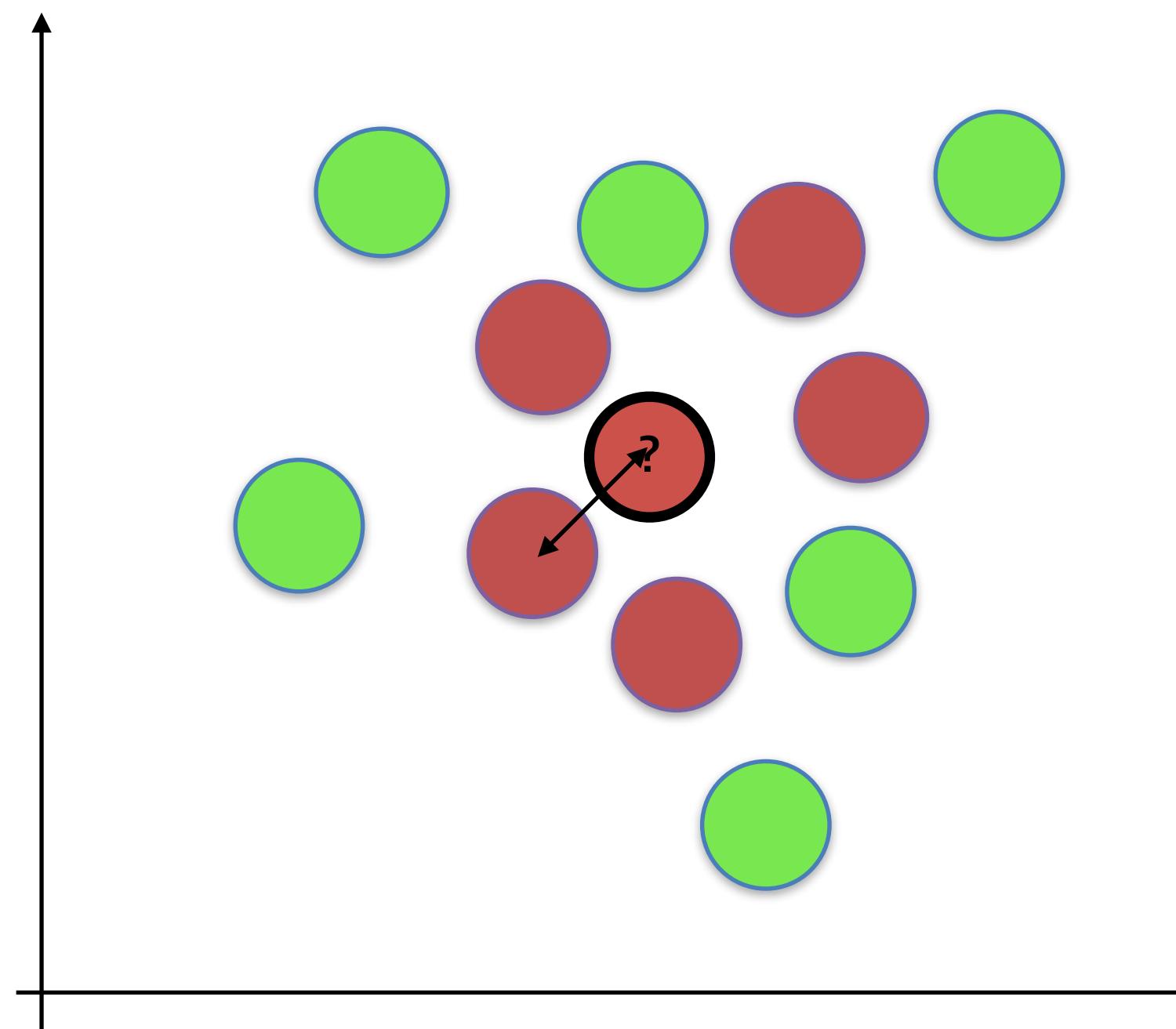
# Simple Example for Classification

## □ Simple Idea for Classification

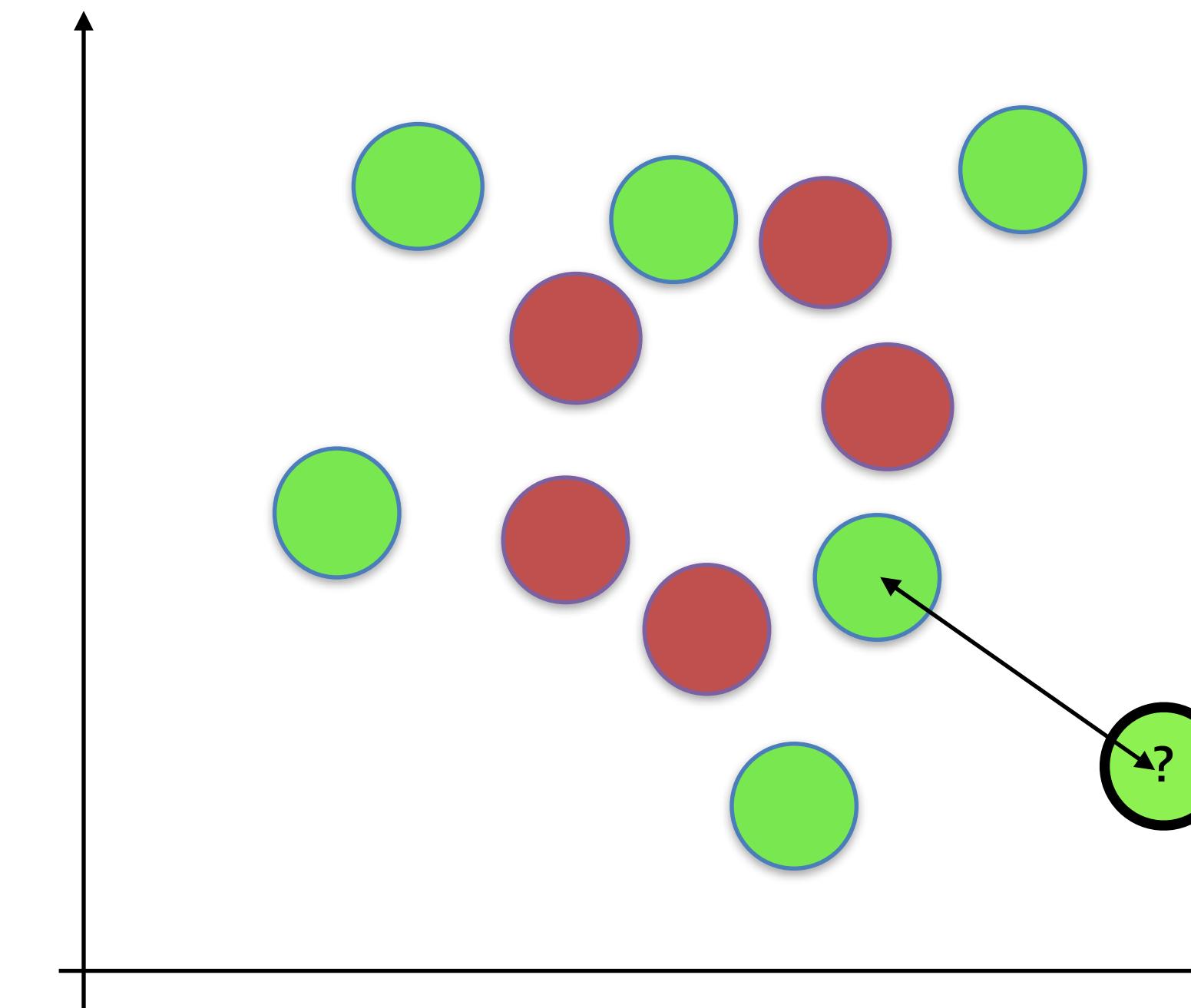
- 현재 가지고 있는 데이터는 위치에 따라 색깔(Green/Red)의 Class로 분류가 가능하다고 가정

## □ Class(색깔)을 모르는 임의의 점에 대하여 색깔을 분류하는 가장 간단한 방법은? 가장 가까운 점의 색깔을 부여

<Problem #1>



<Problem #2>



# k Nearest Neighbor (kNN)

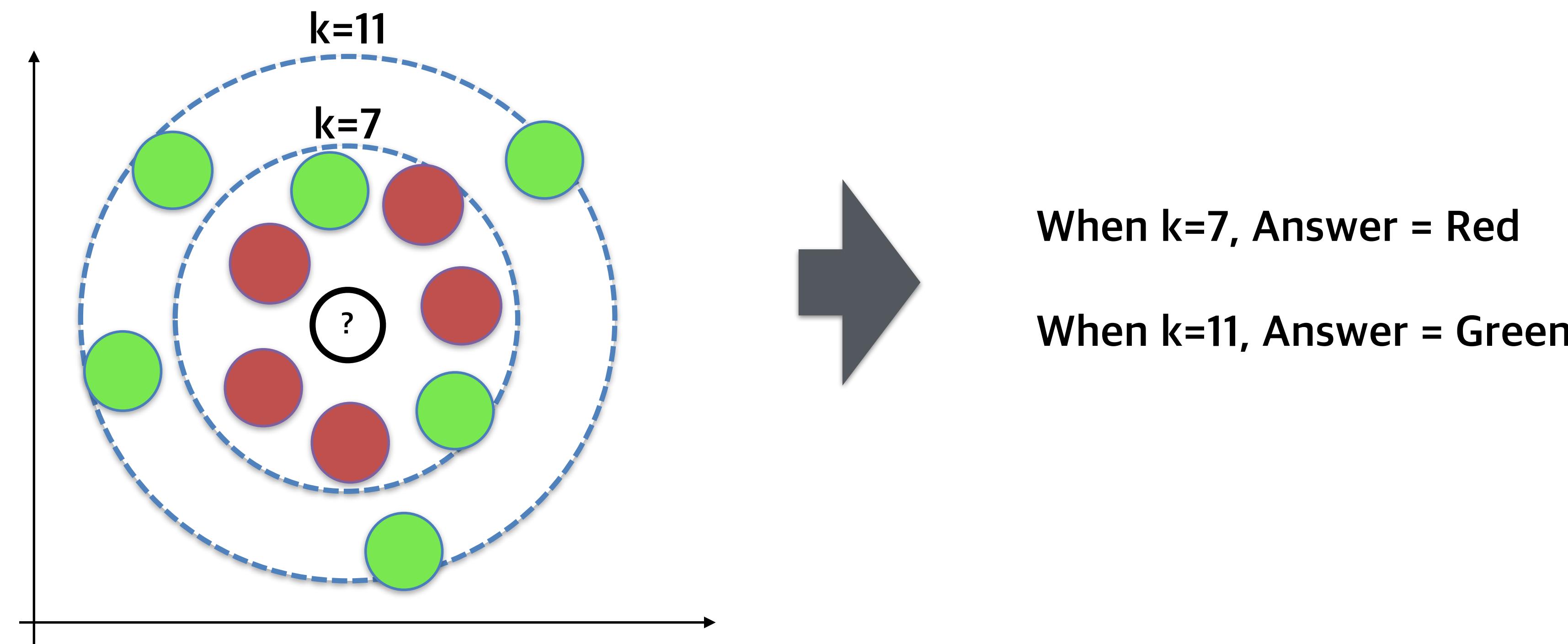
## ❑ k Nearest Neighbor (kNN) Classifier

- Class를 모르는 임의의 입력 데이터와 가장 거리가 가까운 **k 개의 데이터**가 가장 많이 소속된 class로 분류하는 방법

❑ 간단하지만 보편적으로 좋은 성능을 보이는 **분류(classification)** 방법으로 분류 문제에 관한 연구의 대조군으로 사용되기도 함

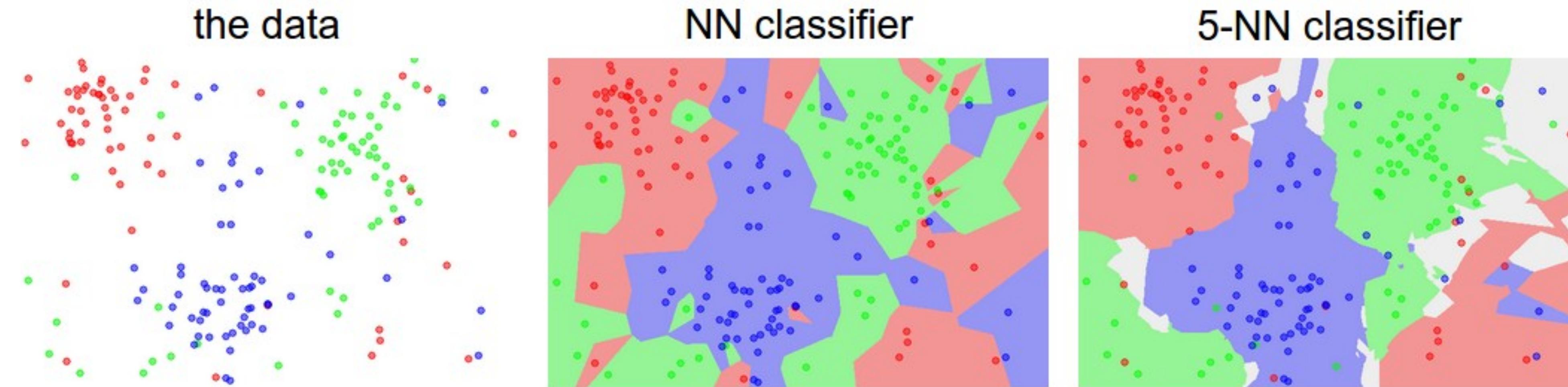
❑ k는 모델 사용자가 **직접 결정해주어야하는 하이퍼 파라미터(hyper parameter)**이며 설정에 따라 결과가 달라질 수 있음

(학습의 대상이 아님)



# Decision Boundary of kNN

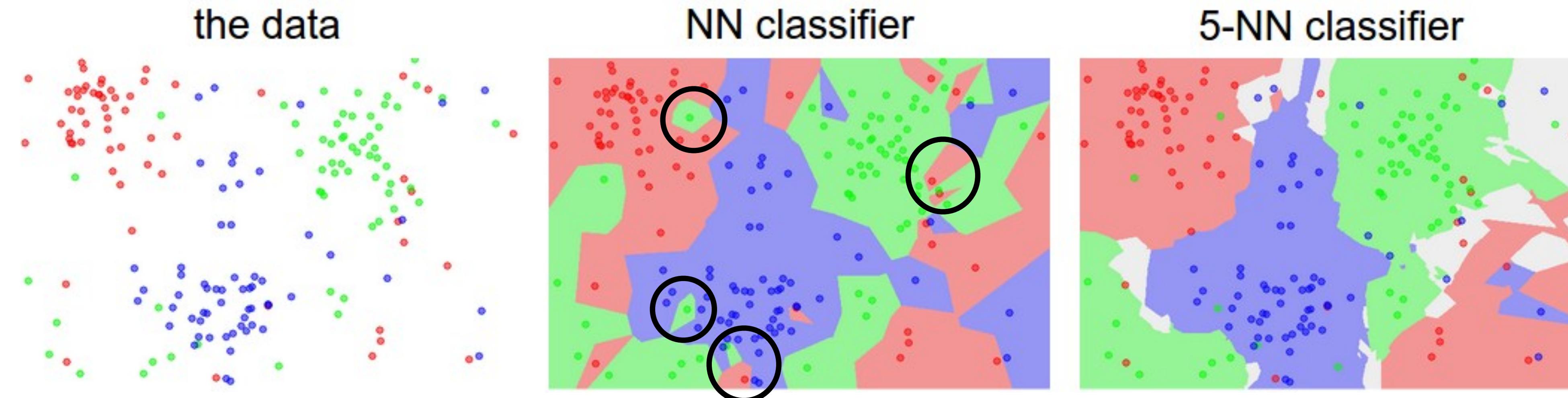
- 현재 보유하고 있는 정답을 알고 있는 데이터(labeled data, training data)와 k를 바탕으로 판정 경계를 결정할 수 있음
- k 값이 증가할수록 판정 경계가 부드러워지고, 모델의 과적합(overfitting)을 막을 수 있음



- 데이터 포인트 = Training Data
- 색 영역 = Decision Area

# Decision Boundary of kNN

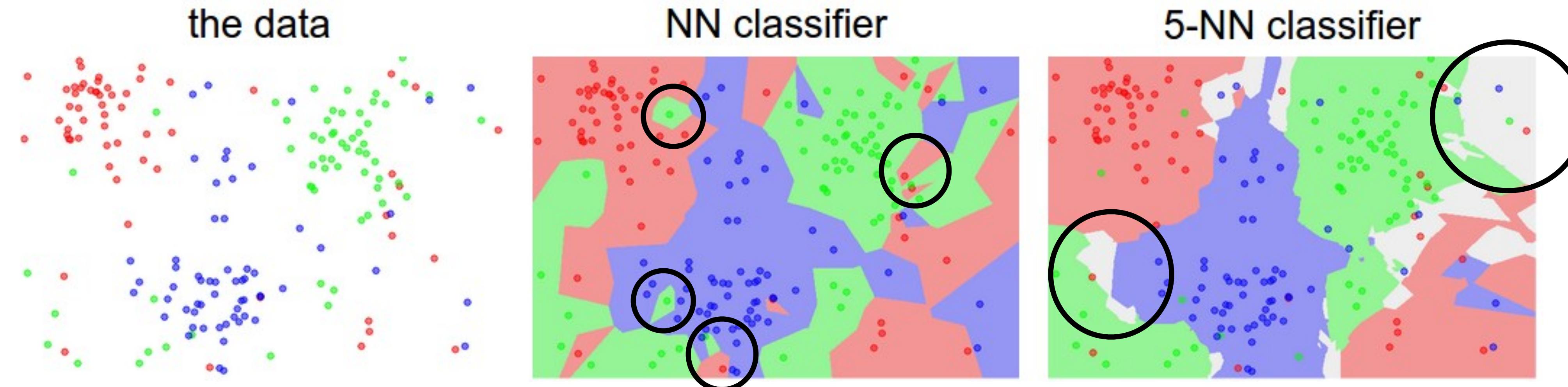
- 현재 보유하고 있는 정답을 알고 있는 데이터(labeled data, training data)와 k를 바탕으로 판정 경계를 결정할 수 있음
- k 값이 증가할수록 판정 경계가 부드러워지고, 모델의 과적합(overfitting)을 막을 수 있음



- 데이터 포인트 = Training Data
- 색 영역 = Decision Area
- Outlier에 대한 민감도가 높음

# Decision Boundary of kNN

- 현재 보유하고 있는 정답을 알고 있는 데이터(labeled data, training data)와 k를 바탕으로 판정 경계를 결정할 수 있음
- k 값이 증가할수록 판정 경계가 부드러워지고, 모델의 과적합(overfitting)을 막을 수 있음

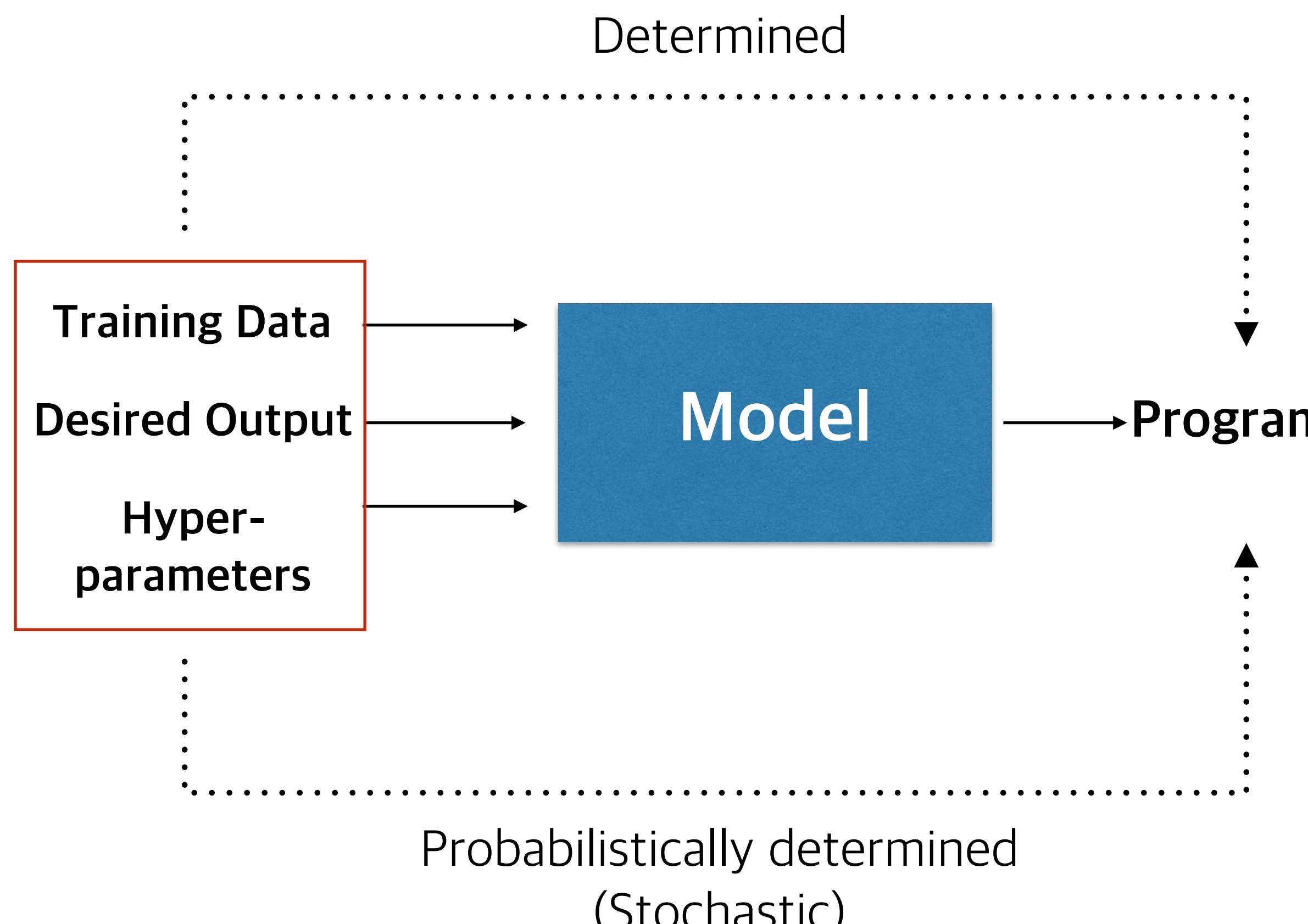


- 데이터 포인트 = Training Data
- 색 영역 = Decision Area
- Outlier에 대한 민감도가 높음
- Gray Area = 판정 불가 영역 (Tie score)

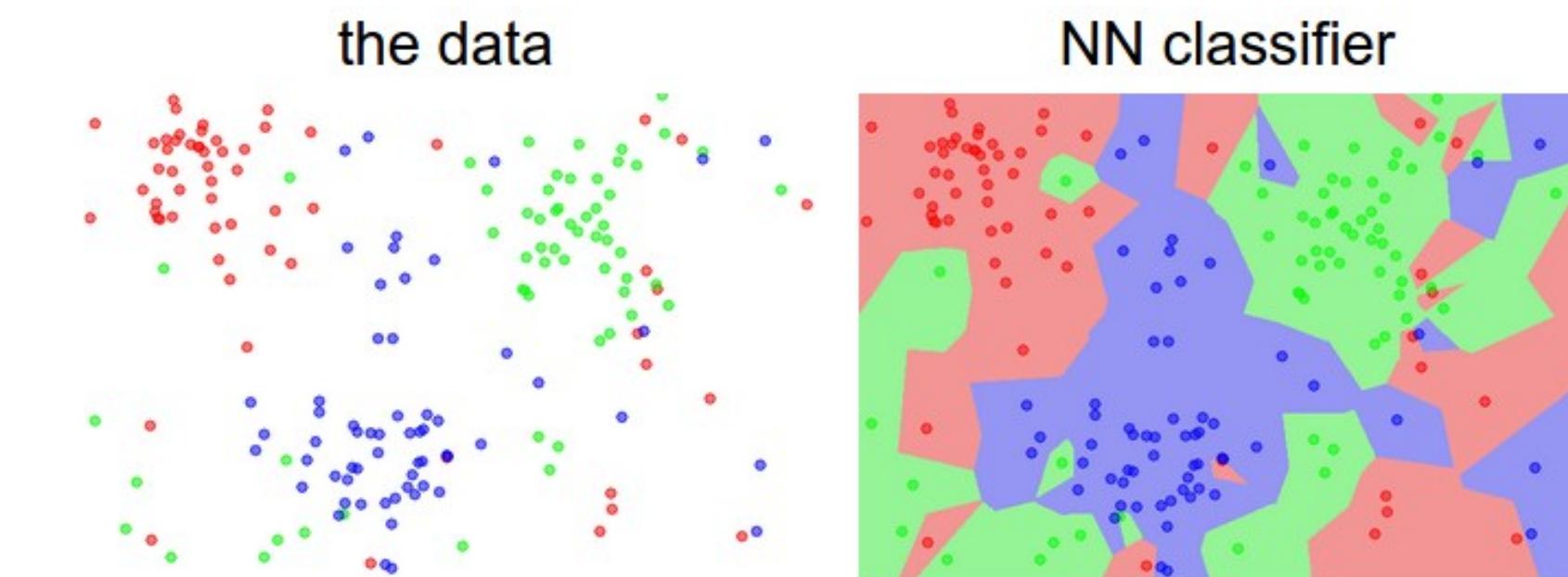
# Deterministic vs. Stochastic Model

## ❑ kNN 분류기는 training data와 k 값에 대하여 결정적인(deterministic) 모델

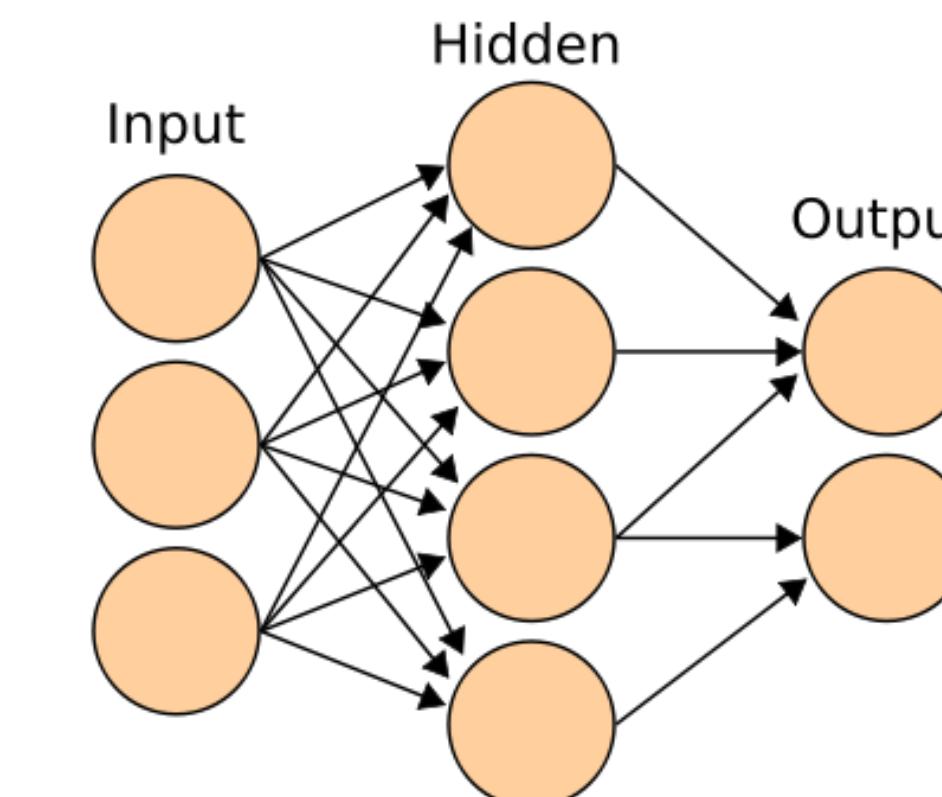
- Training data와 k 값이 같다면 학습 반복에 관계없이 같은 결과를 제공



<Deterministic Model Example>



<Stochastic Model Example>



# Characteristics of kNN

## ❑ Advantages of kNN

- 구현이 간단하고, 일반적인 성능이 높음
- 모델의 계산 알고리즘과 결과의 이해가 직관적임
- 데이터의 분포에 대한 사전 지식 없이 사용 가능 (non-parameteric model)

## ❑ Disadvantages of kNN

- 학습 데이터가 많을 경우, 계산시간이 매우 높음  
(ex. 학습 데이터 100만개 --> 1개 테스트 데이터마다 100만개 사이 거리 계산)
- $k$ 의 값이 성능 차이를 결정하며, 이 값을 직접 결정해주어야함  
( $k$  값에 대하여 적절한 기준이 존재하는 것이 아님. 가지고 있는 데이터에 따라 다를 수 있음)

## Determination of k

- ❑ Neighbor 수 k를 결정하는 방법은 1) Rule of thumb 또는 2) Cross-validation 을 사용함

### 1) Rule of Thumb

$$k = \sqrt{n}, \text{ where } n: \# \text{ of training data}$$

# Determination of k

- ❑ Neighbor 수 k를 결정하는 방법은 1) Rule of thumb 또는 2) Cross-validation 을 사용함

## 1) Rule of Thumb

$$k = \sqrt{n}, \text{ where } n: \# \text{ of training data}$$

## 2) Cross-validation

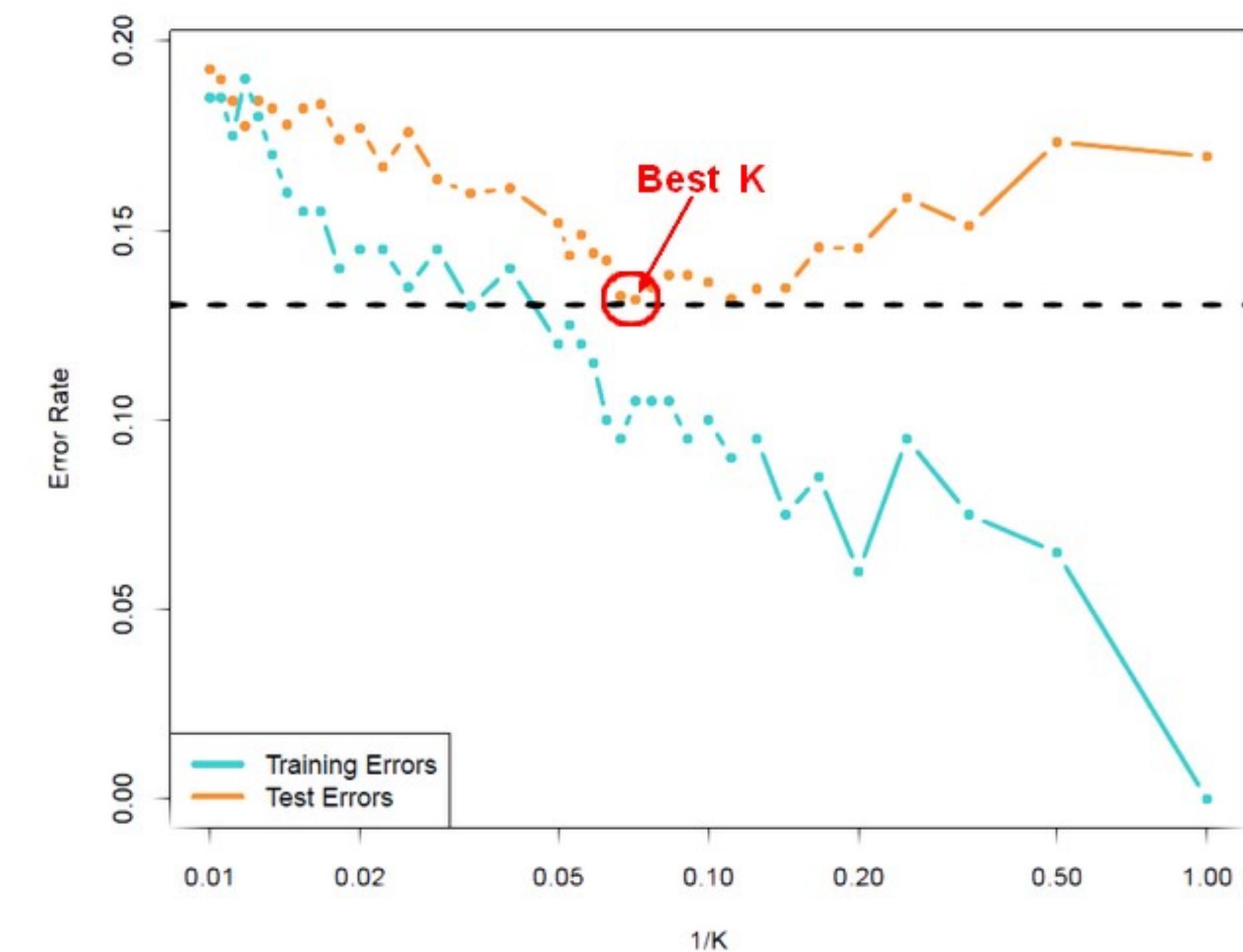
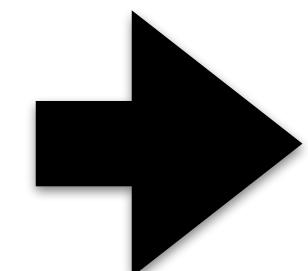
<Data Split with No Validation >



<Data Split with Validation>

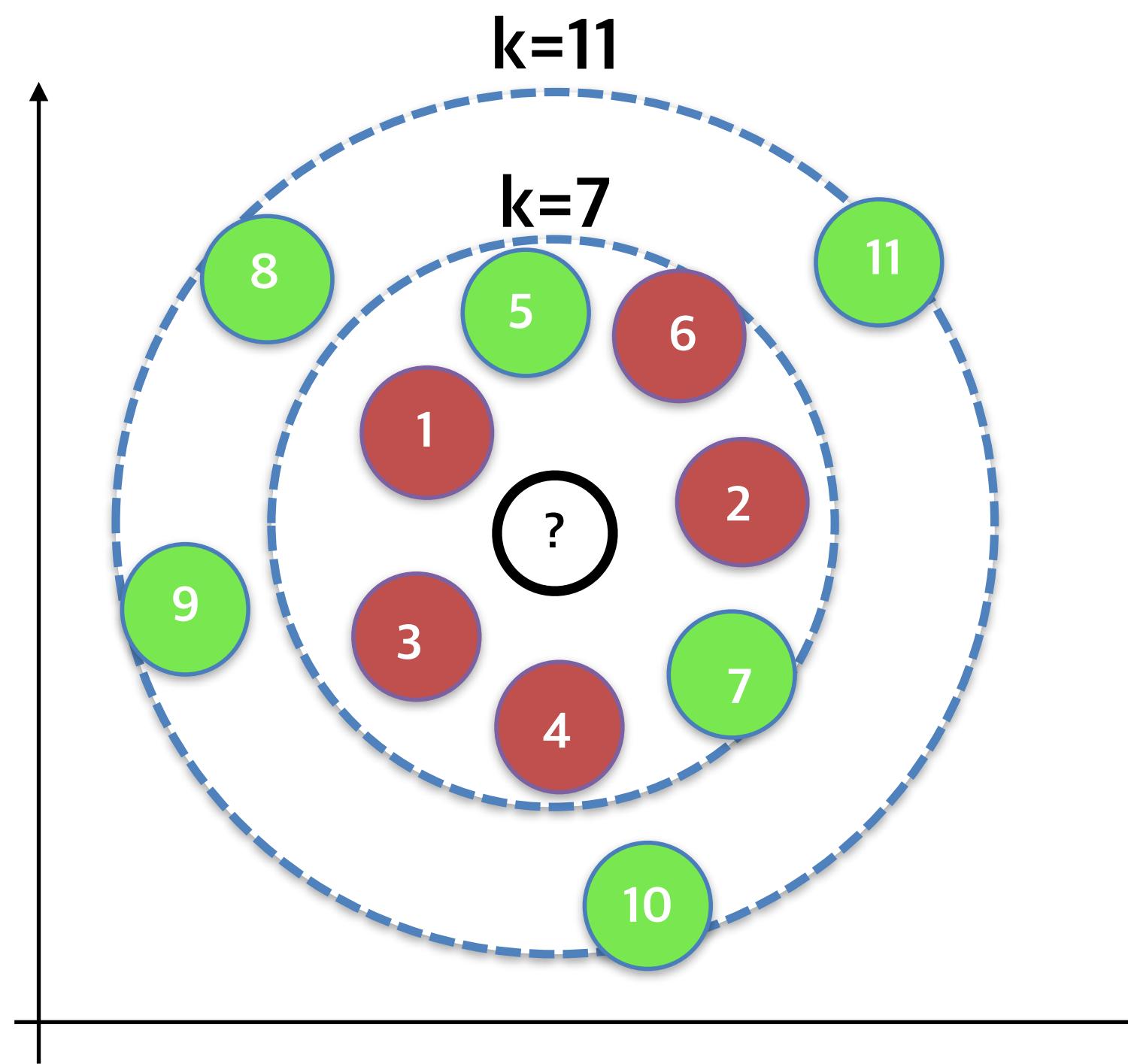


학습 반복마다 일반화(Generalization)에 대한 검증



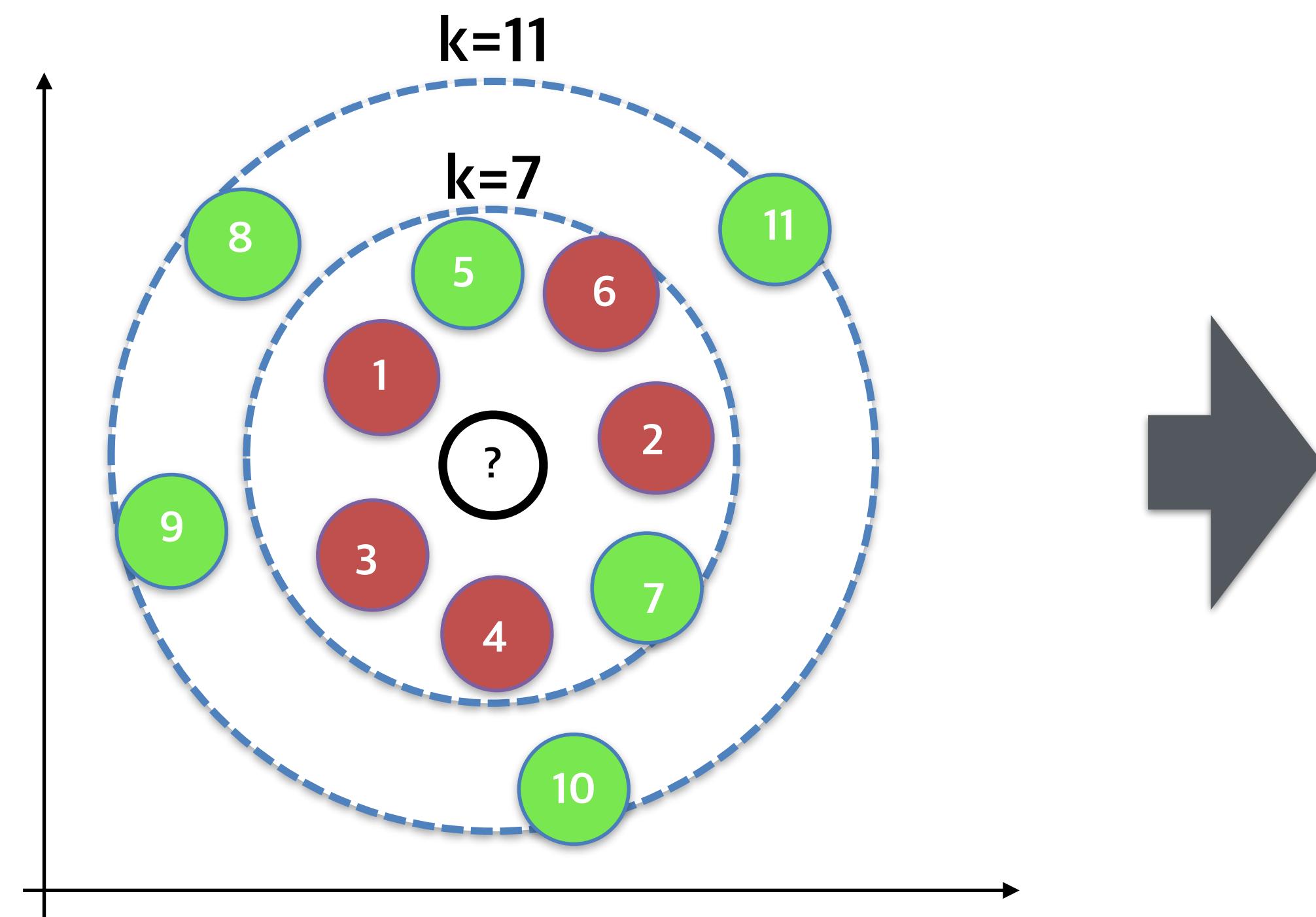
## Inverted Indexes of kNN

- **k** 값이 클 때, 데이터 분포 특성에 따라 분류기 모델의 정확도가 낮아질 수 있는 가능성 존재
- Inverted Index를 사용하여 거리에 따른 패널티를 부여하는 방식으로 해결 가능 (penalizing)
- 거리 순서에 따라 Index를 매기고, Index의 역수를 Score로 사용



# Inverted Indexes of kNN

- $k$  값이 클 때, 데이터 분포 특성에 따라 분류기 모델의 정확도가 낮아질 수 있는 가능성 존재
- Inverted Index를 사용하여 거리에 따른 패널티를 부여하는 방식으로 해결 가능 (penalizing)
- 거리 순서에 따라 Index를 매기고, Index의 역수를 Score로 사용



<Original kNN>    When  $k=7$ , Answer = Red  
                          When  $k=11$ , Answer = Green

## <kNN with Inverted Indexes>

When  $k=7$ ,

$$\begin{aligned} \text{Sum of Inverted Index (Red)} &= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{6} = \frac{27}{12} \\ \text{Sum of Inverted Index (Greedy)} &= \frac{1}{5} \end{aligned}$$

Answer: Red

When  $k=11$ ,

$$\begin{aligned} \text{Sum of Inverted Index (Red)} &= \frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{6} = \frac{27}{12} \\ \text{Sum of Inverted Index (Greedy)} &= \frac{1}{5} + \frac{1}{7} + \frac{1}{9} + \frac{1}{10} + \frac{1}{11} = 0.645 \end{aligned}$$

Answer: Red