

# Deep Learning for Biologists

EunAh Lee, PhD



# **Curriculum**

- **Introduction to BME**
- **Fundamentals of Biochemistry & Cell Biology**
- **Principles of Tissue Engineering & Regenerative Medicine**
- **Convergence Technology for Regenerative Medicine**
  - Materials Science
  - AI Technology

# Artificial Intelligence

Artificial Intelligence

Machine Learning

Neural Network

Deep Learning

## Artificial Intelligence, AI

인간의 지능을 기계로 구현하는 것 (기계가 알아서 일을 하도록 구현한 것)

사람이 수행하는 지능적인 작업을 컴퓨터가 모방하도록 하는 모든 기술

**심볼릭 AI/규칙 기반 AI** 규칙을 사람이 수동으로 만들어 컴퓨터에 입력하여 구현

**일반 인공지능/강인공지능** 다양한 일을 처리할 수 있는 인간 수준의 지능

**초지능** 인간의 수준을 훨씬 뛰어넘는 고차원적 사고를 가지는 지능

# AI 용어설명

**머신러닝** 기계가 지능을 가지도록 스스로 학습하는 것

Artificial Intelligence

Machine Learning

Neural Network

Deep Learning

**신경망/인공신경망** 머신러닝 알고리즘 중 인간의 뇌를 구성하는 뉴런에 영감을 받아 형성한 모델/ 입력층-은닉층-출력층

**퍼셉트론(Perceptron)** 신경망을 구성하는 기본 단위

**활성화 함수(Activation Function)** 입력층으로부터 주입된 데이터에서 규칙을 찾아내기 위해 입력 데이터를 적절하게 변환해주는 함수

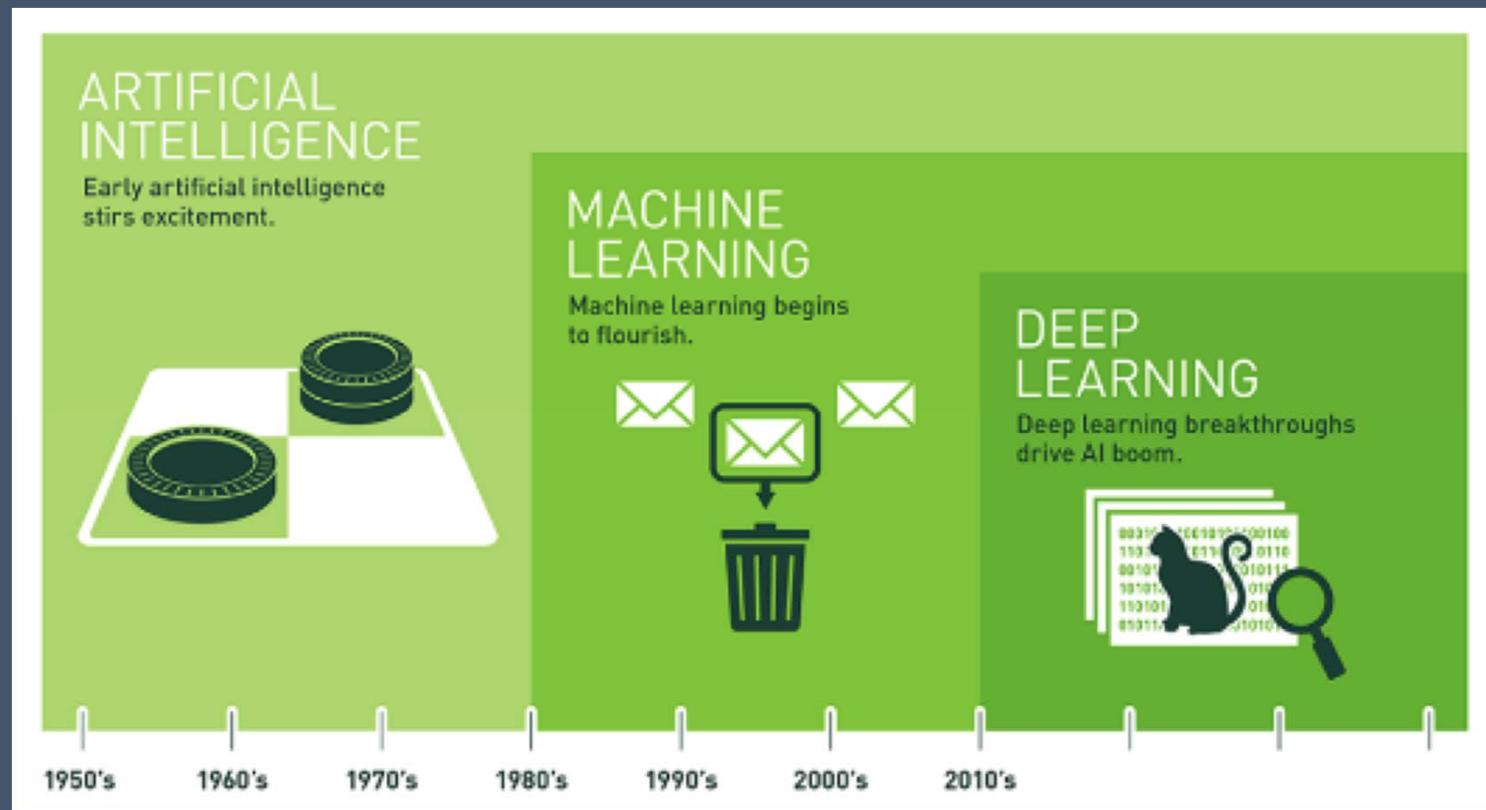
**손실 함수(Loss Function)**/ 비용함수(Cost Function)/목적 함수(Objective Function)/오차함수>Error Function) 신경망 학습이 잘 되고 있는지를 확인할 때 사용하는 함수

**경사하강법(Gradient Descent)** 신경망을 데이터에 최적화 하기 위해 손실함수의 경사를 구하고 기울기가 낮은 쪽으로 계속 이동시키는 방법

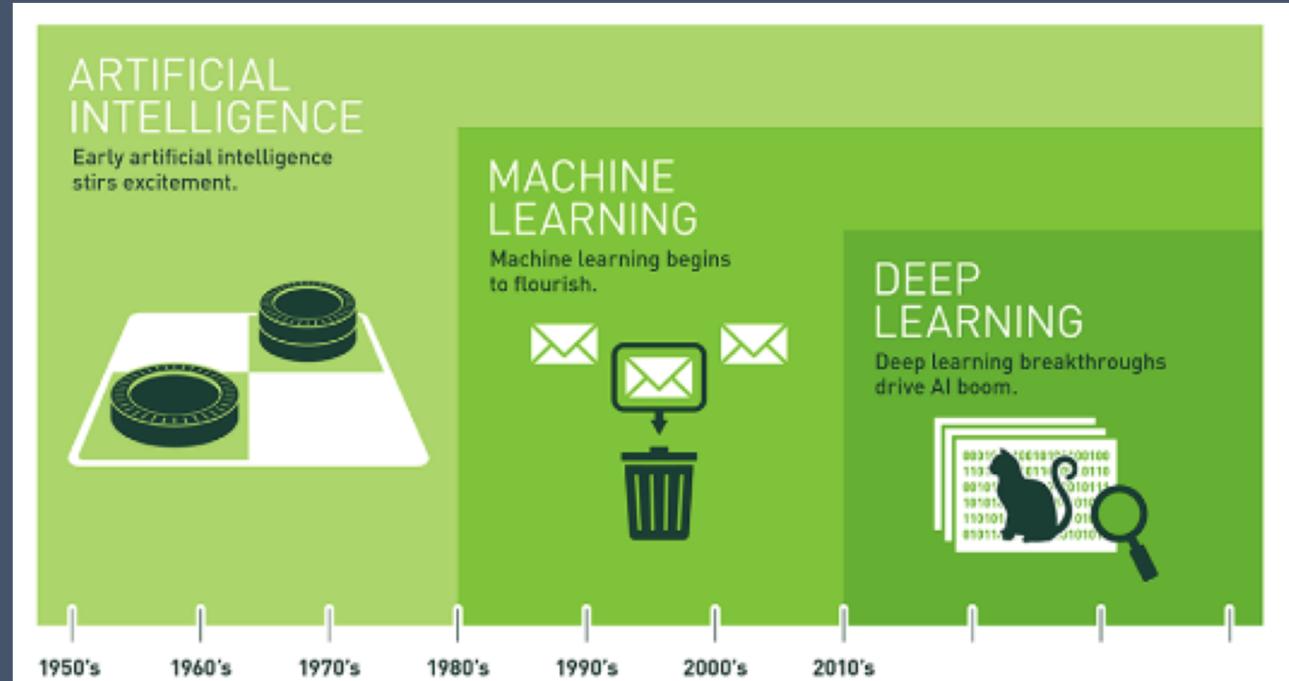
**역전파 알고리즘(Backpropagation Algorithm)** 딥러닝 네트워크 구조에서 입력을 통하여 나온 경사(미분값)를 역방향으로 다시 보내어 신경망을 업데이트하는 방법

**딥러닝** 은닉층을 깊게 쌓은 신경망 구조를 활용해 학습하는 방법

# AI, ML, DL



# 머신러닝(ML; Machine Learning)



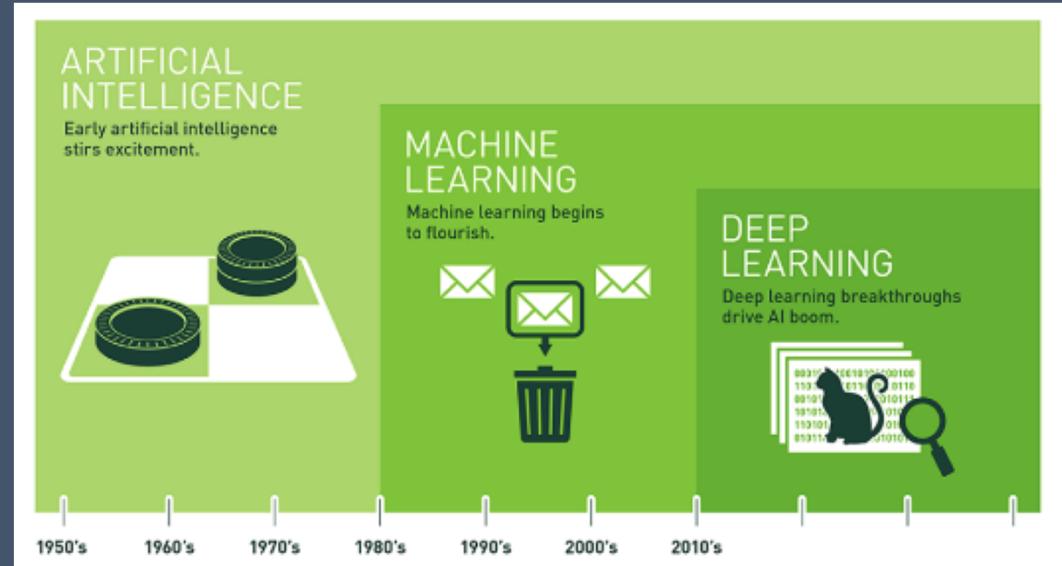
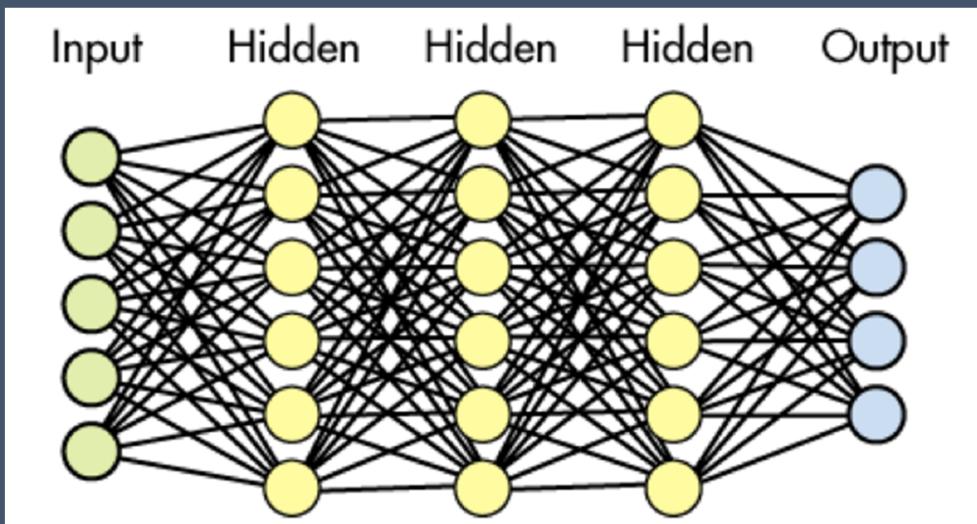
인공지능의 한 분야  
데이터에서 가치를 찾아내는 것  
아주 다양한 방법이 있다.

- SVM, 의사결정트리, Random Forest, Bayesian, K-Means Clustering, K-NN, Neural Network

# 딥러닝(DL; Deep Learning)

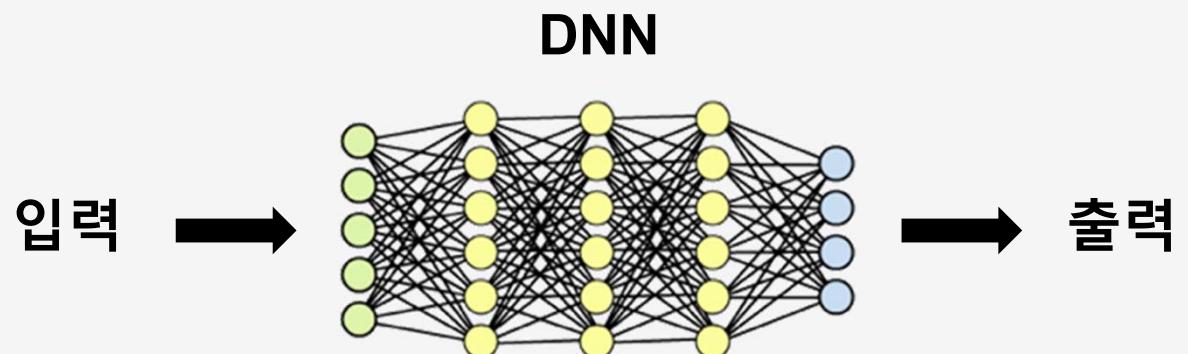
신경망(NN; Neural Network)을 사용한 머신러닝 방법

신경망의 은닉층이 많아서(deep) DNN(Deep NN)이라 부른다



# Universal Approximator (함수근사화)

- DNN은 어떠한 함수도 근사화 가능
- - 함수의 내부를 모르더라도 입력과 출력 데이터로 근사화 가능

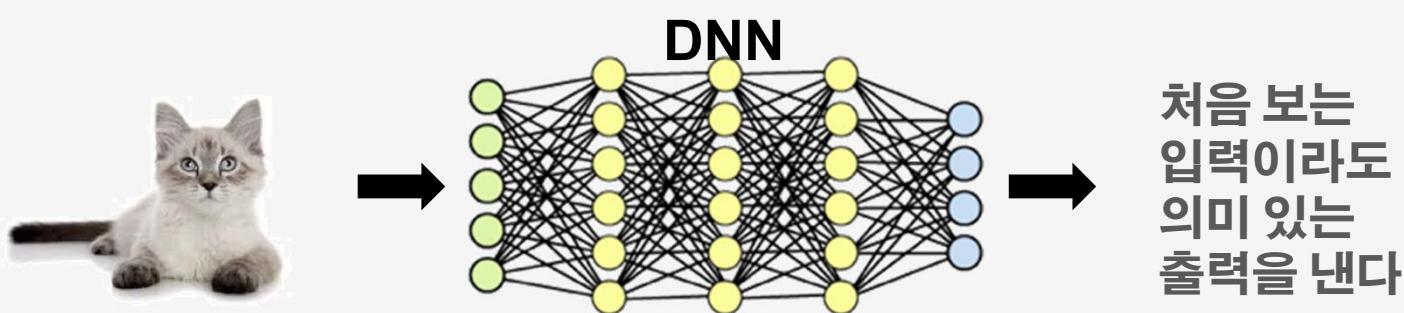


고양이, 강아지를 구분 함수의 예:

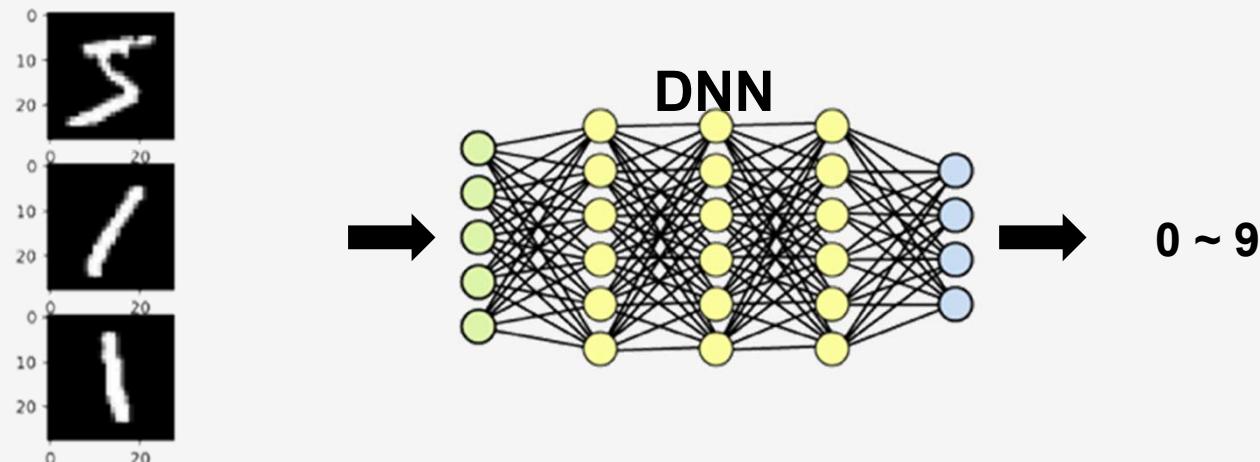
- 구분 방법은 정의하지 않음
- 정의하기는 힘들어도, 그 함수는 존재한다. 로직을 찾아내기 어려운 문제에 효과적
- 단순히 입출력 쌍 데이터로 반복하여 학습시켜 그 함수를 근사화 함

## 학습 과정

---

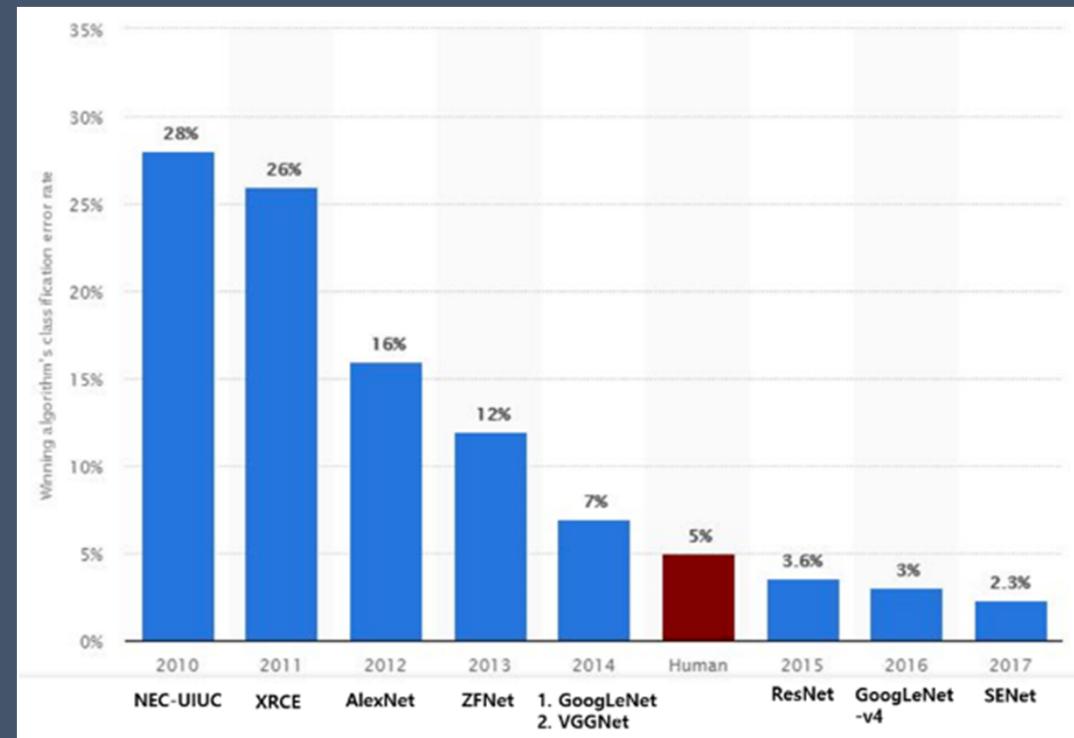
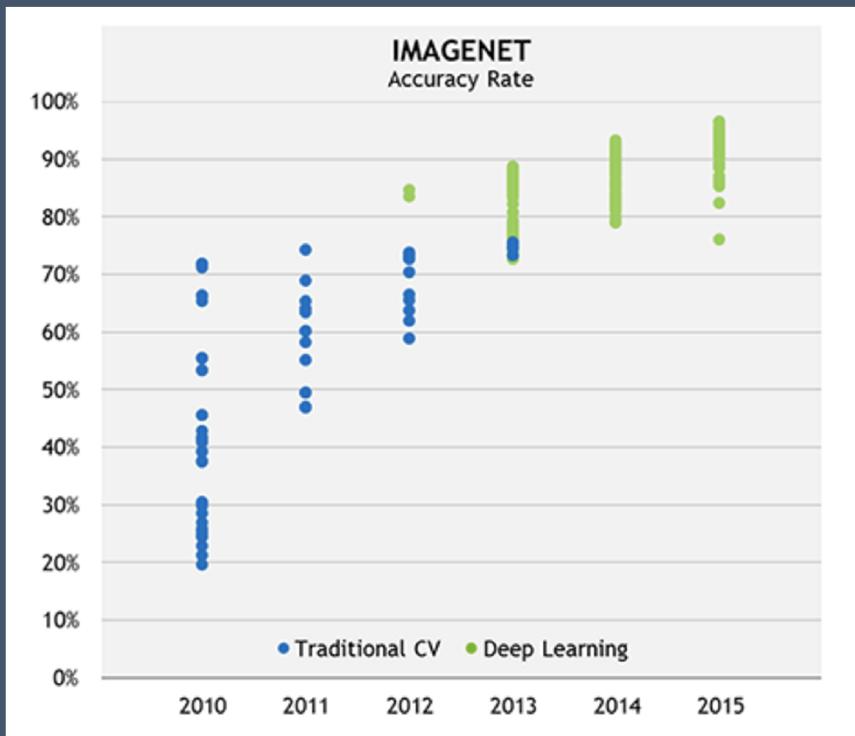


# 로직을 만들기 어려운 함수 근사화에 효과적

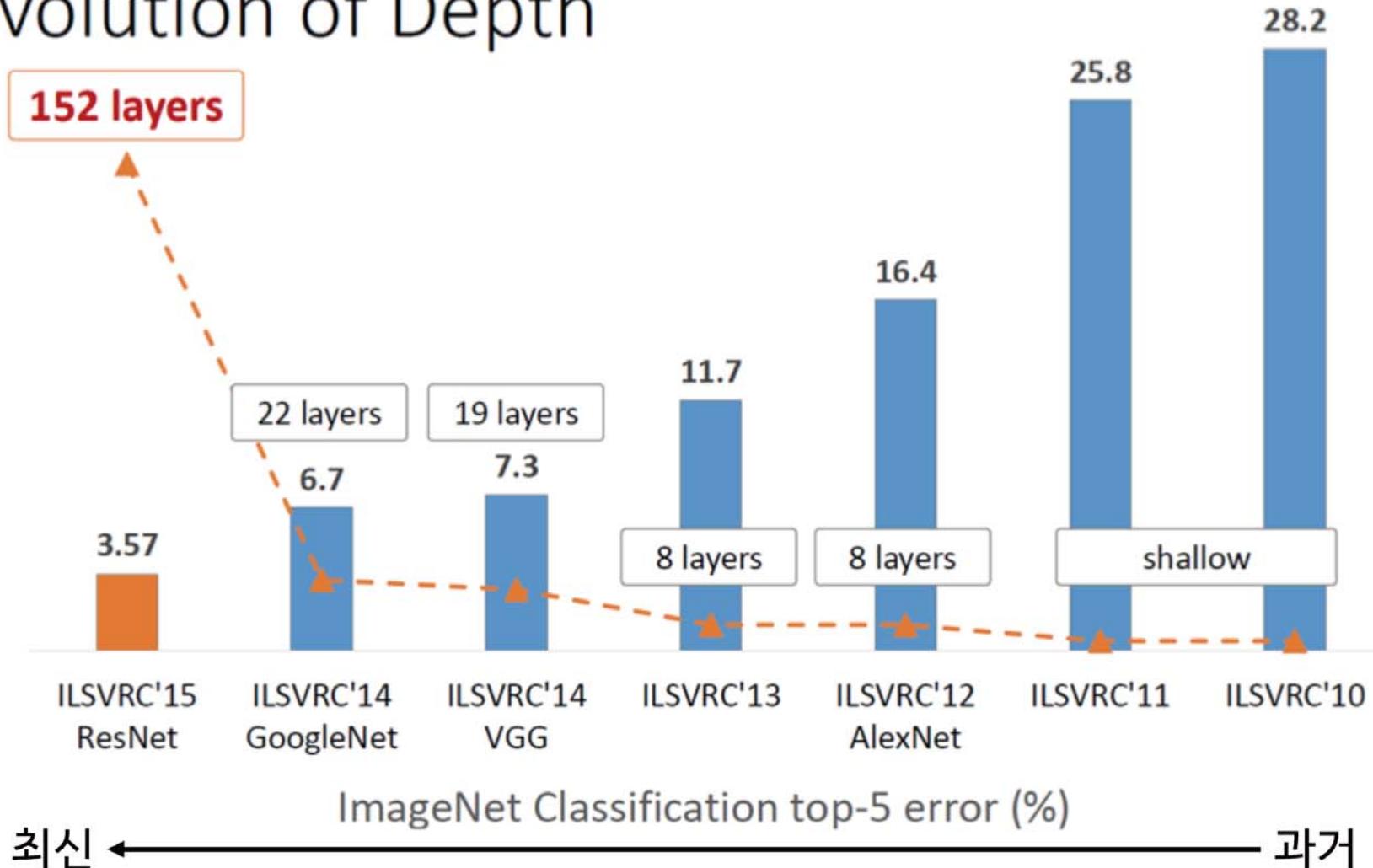


# 2012년 AI 부활

## The ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

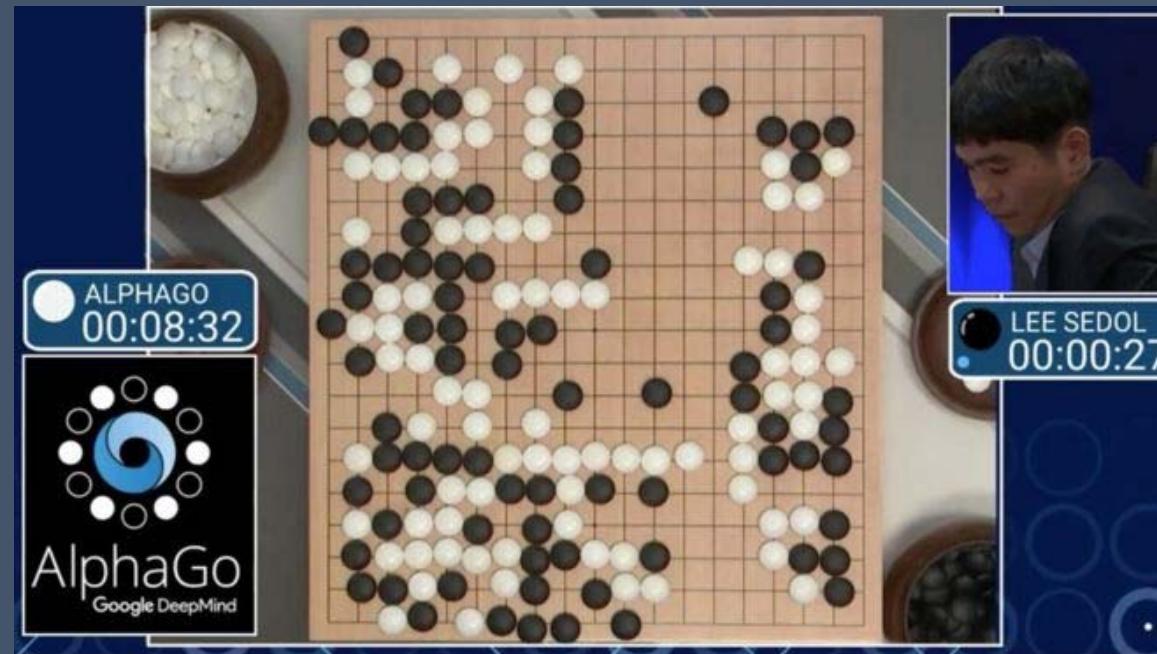


# Revolution of Depth



# 2016년 화려한 복귀

---



# Deep Learning 용어설명

---

**합성곱 신경망(Convolutional Neural Network, CNN)**

Yan Lecun 개발

이미지 처리 컴퓨터 비전에서 활용

**순환 신경망(Recurrent Neural Network, RNN)**

시계열 데이터와 언어처리분야에서 주로 활용

**적대적 생성 신경망(Generative Adversarial Network, GAN)**

실제 데이터와 구분하기 어려운 데이터를 만드는 데 사용되는 신경망



# 베이지안 네트워크 Bayesian Network

확률 기반의 그래프 모델

변수 간의 단순 상관성 뿐만 아니라 인과성까지 고려

사전 지식(사전 확률) 등을 반영하여 사용 가능

# 인공지능 학습방법

**지도학습(Supervised Learning)** 정답이 주어진 데이터를 사용해 컴퓨터를 학습하는 방법 (예: 강아지와 고양이 구분을 위한 labeled data 사용)

예측이 틀린 경우 피드백을 통해 수정하고 이 과정을 반복하면서 학습을 하고 수정을 거쳐 높은 정확도의 구분능력 구현 가능  
문자 인식, 음성 인식, 이미지 분류, 언어 번역 등 거의 모든 어플리케이션에서 활용

**비지도학습(Unsupervised Learning)** 정답이 주어지지 않은 데이터를 기반으로 모델 구조를 통해 학습하는 방법 (Ex. GAN)  
정답이 없으므로 예측에 대한 피드백이 불가능하고 교정도 불가능한 대신 비슷한 것들끼리 집단으로 분류하는 방식으로 작동  
대표적인 방법으로 clustering이 있으며 Facebook에서 특정 집단의 사람들을 그룹화 하는 알고리즘 들에서 쓰임  
지도학습에 비해 활용정도가 낮으나, 수십만 수천만개의 데이터를 label 하는 과정을 생략할 수 있으므로 ML의 발전에 필요한 방법론  
중 하나로 평가되고 있음

**강화학습(Reinforcement Learning)** 에이전트(agent)가 주어진 환경에서 어떤 행동을 취하고 이로부터 어떤 보상을 받으면서 학습을 진행 (환경과 동적으로 연동하여 레이블링 데이터를 취득) 시행착오를 통해 학습하는 방법 중 하나 유용한 어플리케이션의 사례는 아직 없음

\* 에이전트: 다른 사람을 대신해 업무 또는 교섭을 대행하는 사람. 컴퓨터에서는 특정 목적에 대해 사용자를 대신해 작업을 수행하는 자율적 프로세스를 의미(프로그램이나 시스템을 실행하는 주체-주로 컴퓨터-를 의미)

**자기지도학습(Self-supervised Learning)** 사람 개입 없이 입력 데이터로부터 정답을 얻음. Feature 구분을 통해 얻어진 정보를 활용하여 학습

**전이학습(Transfer Learning)** 이미 잘 훈련된 모델이 있고, 해당 모델과 유사한 문제를 해결하기 위해 특성 추출, 레이어 조정, 학습된 모델 등을 활용해 학습하는 방법 (예: 컵을 감지하는 시스템 모델을 주전자 감지 모델에 활용하기 위해 모델을 가져와 세부 조정을 거쳐 학습 하는 것)

# ImageNet

---

- Prof. Li Peipei
- 이미지를 판단할 수 있는 알고리즘을 개발하기 보다는 많은 이미지를 보여주어 기계학습하는 것이 낫다는 아이디어로 2007년부터 시작한 프로젝트
- 인터넷과 집단지성의 힘을 이용하여 5만명의 작업자가 10억장에 이르는 이미지 데이터를 모으고 정리, 분류 작업을 했으며, 이렇게 축적된 대량의 분류된 이미지는 컴퓨터 비전 분야의 발전에 많은 기여를 함
- 2012

# 인공지능 응용분야

## 영상처리(Computer vision)

- **객체 인식** 이미지 내에 존재하는 사물을 인식
- **객체 탐지** 어떤 객체가 어디에 있는지 까지 감지하는 기술
- **이미지 캡션** 이미지가 주어지면 이를 바탕으로 이미지를 설명하는 텍스트를 만드는 기술  
컴퓨터 비전 뿐만 아니라 자연어 처리 기술도 필요

## 자연어 처리(Natural Language Processing, NLP)

- 자연어(Natural Language) 인공적으로 만들어진 인공어와 대비되는 개념의 일상언어. 인간의 언어를 기계가 따라할 수 있도록 연구하고 구현하는 분야.  
머신러닝 이전부터 규칙기반 알고리즘으로 시작 머신러닝에 의해 크게 발전
- **기계번역** 인간이 사용하는 자연어를 컴퓨터를 사용해 다른 언어로 바꾸는 것
- **음성인식** 인간이 말하는 음성을 문장으로 바꾸고 더 나아가 그 의미를 파악하는 기술



HxqD k #hh# KHU  
이미지 캡션  
<https://unsplash.com>

# 인공지능 관련 개발 제품

## 자율주행차

1960년대에 제안, 1970년대 중후반부터 초보수준의 연구 시작  
최근 딥러닝의 발달로 활발하게 연구 진행

## 음성인식 비서

Siri (Apple), Alexa (Amazon), Google Assistant (Google), Jibo (Jibo Inc)

## IBM Watson

자연어 형식으로 된 질문들에 답할 수 있는 인공지능 컴퓨터 시스템  
David Peruci 주도 (IBM)  
2010년 Jeopardy!에서 사람을 이기고 우승  
최근 의료영역으로 확장. 이미지 분석에 사용

## AlphaGo & AlphaZero

데미스 허사비스 (DeepMind) 개발  
AlphaGo Fan: 2015. 10. 판후이 2단에게 승리  
AlphaGo Lee: 2016.03. 이세돌 9단에게 4:1로 승리  
AlphaGo Master: 커제9단에게 완승  
AlphaGo Zero: 2017년 Nature에 소개 (기보 없이 스스로 바둑을 두면서 학습, 알파고와의 대국에서 100전 100승, 알파고 마스터와의 대국에서 100전 89승 11패)

# 개발도구

---

- **TensorFlow** - Opensource ML Library
- **AutoML** - ML 전문가 없이도 ML 개발이 가능하도록 하기 위한 Google Cloud AI Service
- **CPU (Central Processing Unit)** - 컴퓨터가 동작하는 데 필요한 모든 계산을 처리
- **GPU (Graphics Processing Unit)** - 컴퓨터 그래픽을 처리하는 장치. 부동 소수점 계산에 뛰어난 성능. 인공지능 머신러닝 계산 수행에 적합
- **TPU (Tensor Processing Unit)** - 구글에서 만든 데이터분석 및 딥러닝 전용 하드웨어

# AI 관련 기업

**Google Brain** 구글의 인공지능 연구소

**Baidu** 중국 최대 검색엔진기업

**Tencent** 중국 종합 인터넷 회사. 아시아 게임계. 수퍼셀 인수

**Deep Mind** 데미스 허사비스 설립. 2014년 구글에 인수됨. 일반 인공지능 목표- AlphaGO, AlphaZero, AlphaStar

**Open AI** Elon Musk가 투자한 비영리 AI 연구단체. 일반인공지능 및 윤리문제 연구

**Allen Institute** Paul Allen (마이크로소프트 공동창업자)의 비영리 연구단체

- **앨런 뇌과학연구소** - 뇌지도 작성 프로젝트
- **앨런 인공지능연구소** - 공공선을 위한 인공지능 개발 목표. 인공지능에 상식을 구축하는 'Mosaic' 프로젝트 진행

**Future of Humanity Institute** 인공지능이 인류에 가져올 영향력에 대해 여러 분야에 걸쳐 연구

**Tesla** Elon Musk가 투자한 전기자동차 회사. 완전자율자동차 개발 목표

# AI로 인해 논의 되는 기타 주제들

## Turing Test

앨런 튜링이 제안한 테스트로 기계가 인간과 얼마나 비슷하게 대화할 수 있는지를 기준으로 기계에 지능이 있는지를 판별하고자 하는 테스트

## 일치문제(Alignment problem)/통제문제(Control problem)

인공지능의 발달로 인해 발생할 수 있는 문제의 유형

인류가 바라는 가치와 일치하지 않고 인간의 통제를 벗어나 인류에게 해를 끼칠 수 있는 초지능 발생의 문제

## 설명가능성(Explainability)

데이터를 가지고 인공지능이 예측을 했지만 어떻게 데이터를 분석했는지를 명확하게 설명할 수 있는 능력

## 역공학(Reverse engineering)

시스템을 역으로 추적해 분석하는 기법

장치 또는 시스템의 기술적 원리를 그 구조 분석을 통해 발견하는 과정

## 기본소득(Basic income)

모든 사회 구성원들에게 아무 조건 없이 정기적으로 지급하는 소득

# 딥러닝의 장단점

## 딥러닝의 장점: 대상함수의 내부를 몰라도 작동가능

DNN은 함수 근사화 능력이 있음

입출력 쌍을 반복적으로 제공하여 내부를 업데이트

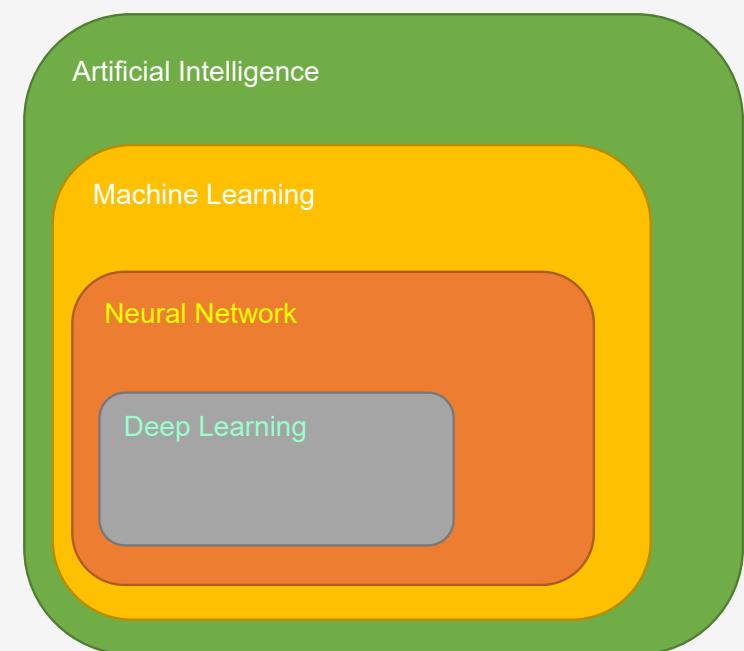
딥러닝: DNN으로 특정 함수 근사화하는 것

## 딥러닝의 단점: 고비용 (데이터 & 연산량)

충분한 학습 데이터(Training data) 개수 필요

- 입출력 데이터 쌍을 구하기 어렵다  
(특히 출력 데이터: Labeled data)

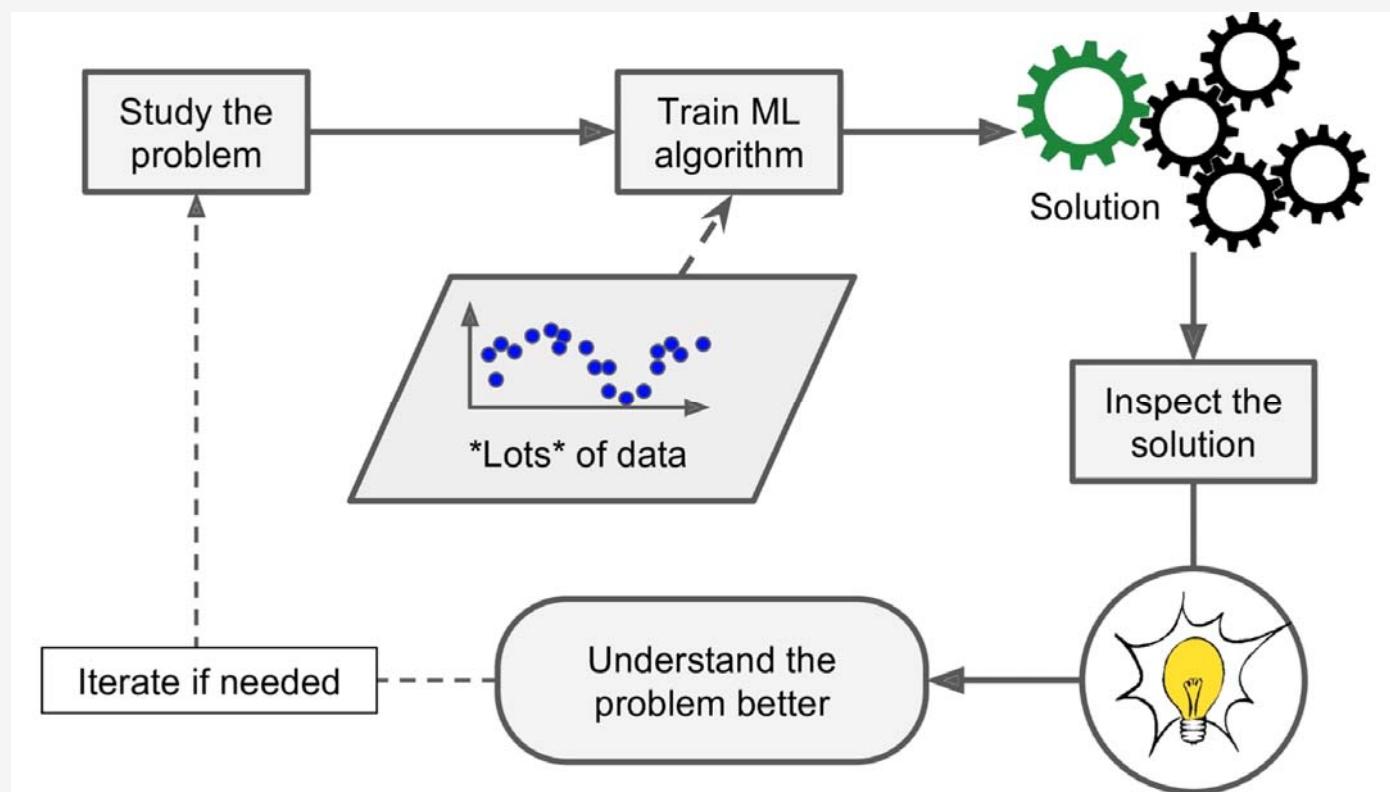
컴퓨팅 파워 필요



# Deep Learning

## Deep Learning의 Breakthrough:

- Geoffrey Hinton (2005): 역전파 알고리즘 (Back propagation)
- Andrew Ng (2012)

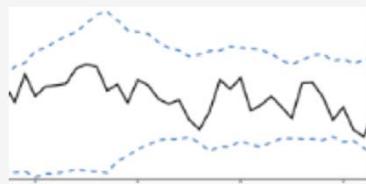


# Machine Learning or Deep Learning?

- 기존의 방법으로 이미 풀린 문제는 Machine Learning
- 기존의 방법으로 못 풀었는데 데이터가 많이 있으면 Deep Learning
  - 바둑, 얼굴인식, 물체인식, 음성인식, 번역



“hello” → 한국어



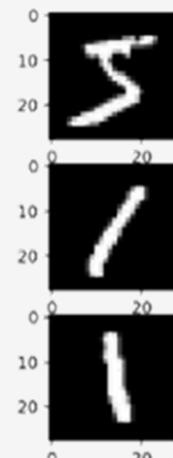
시스템 데이터



1200 × 900 - brunch.co.kr



음파



# 적절한 Training set의 갯수

- Data point  $10^4$ 개 이상: 이상적 수준
- $10^3$ 개 이상: 아쉽지만 학습이 가능한 수준
- $10^2$  단위: data 증강 필요. 어쩔 수 없이 생성하는 수준

# Deep Learning



# 딥러닝을 위한 데이터 규모의 문제

- 데이터 처리 관련 비용의 변화

- 100만 트랜지스터 당 비용

1992년 \$222.00

2012년 \$0.06

20년 만에  $2.7 \times 10^{-4}$  배 수준으로 감소

- 1GB 저장장치 비용

1985년 \$80,000.00

1995년 \$686.00

2005년 \$0.75

2015년 \$0.02 30년 만에  $1.25 \times 10^{-7}$  배 수준으로 감소

2020년 \$0.01 35년 만에  $6.25 \times 10^{-8}$  배 수준으로 감소

$10^n$	명칭	기호
$10^{24}$	Yotta	Y
$10^{21}$	Zetta	Z
$10^{18}$	Exa	E
$10^{15}$	Peta	P
$10^{12}$	Tera	T
$10^9$	Giga	G
$10^6$	Mega	M
$10^3$	Kilo	K
$10^2$	Hecto	H
$10^1$	Deca	da
$10^0$	-	-
$10^{-1}$	Deca	d
$10^{-2}$	Centi	c
$10^{-3}$	Milli	m
$10^{-6}$	Micro	$\mu$
$10^{-9}$	Nano	n
$10^{-12}$	Pico	p
$10^{-15}$	Femto	f
$10^{-18}$	Atto	a
$10^{-21}$	Zepto	z
$10^{-24}$	Yocto	y

“…위와 같은 이유로 웹의 총 용량은  
2 페타바이트에 달한다.”

-2001년

“…2012년의 경우 하루에 생성되는 데이터의 양은  
약 7.5 엑사바이트…”

-2012년

# 빅데이터

데이터의 입력 속도가 데이터의 처리 속도를 웃도는 상황

기존 방법론으로는 실시간 처리가 불가능한 데이터

## 클라우드

- 연산자원 제공 → 특징: 이용패턴 증가
- 유휴 시간에는 감소

## 클라우드와 빅 데이터

- 공간을 더 써서 속도를 올림
- 분산 처리에 적합하게 데이터를 가공하고 병렬화 함
- 클라우드에 적합
- 서비스 시간-유휴 시간에 따라 동작으로 데이터 수집/처리

# 클라우드

---

## 변천

- 메인 프레임(1980s) → Network Computer (1990s) → 데이터 서버 (2000s) → 클라우드 (2010s)

필요한 만큼의 양을 필요할 때 사용 & 연산 자원까지 제공

## 아마존 웹 서비스

- 자체적 서비스 backend를 외부에 공개하는 서비스로 시작함
- 현재는 아마존의 주요 서비스로 성장

# 클라우드 인프라 사용양상의 변화

대부분의 전산 infrastructure가 클라우드화 됨

- 더 이상 유휴자원의 재판매 개념이 아니며 클라우드 전문사업 등장

## 머신러닝 시대의 도래

- 기존에 필요하지 않았던 리소스가 중요해 졌음
- GPU, special-purpose ASIC (e.g., TPU)

## GPU 클라우드

- 폭발적인 수요 증가 추세 (2017~)
- 전성비 측면 /실사용 측면 기술적으로 엄청난 개선 진행 중

# GPU의 특징

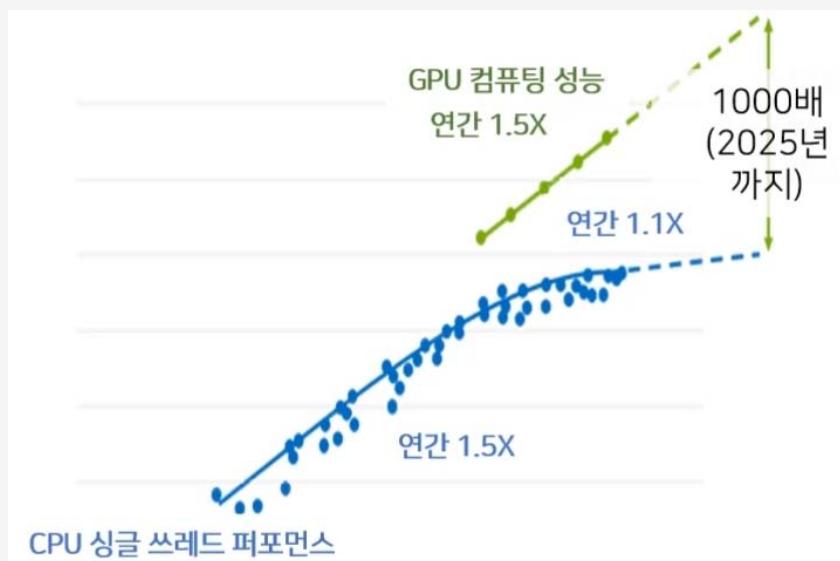
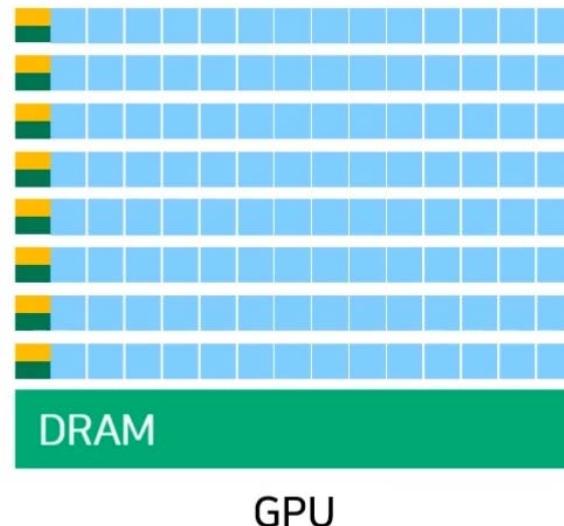
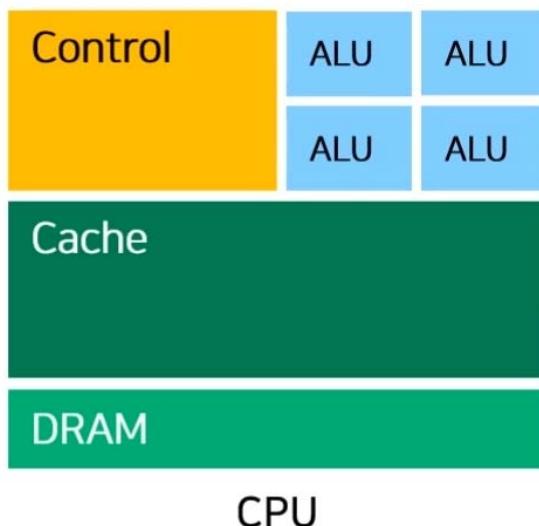


대규모 병렬연산(SPMD: Single Program Multiple Data)방식에 특화된 칩 구조

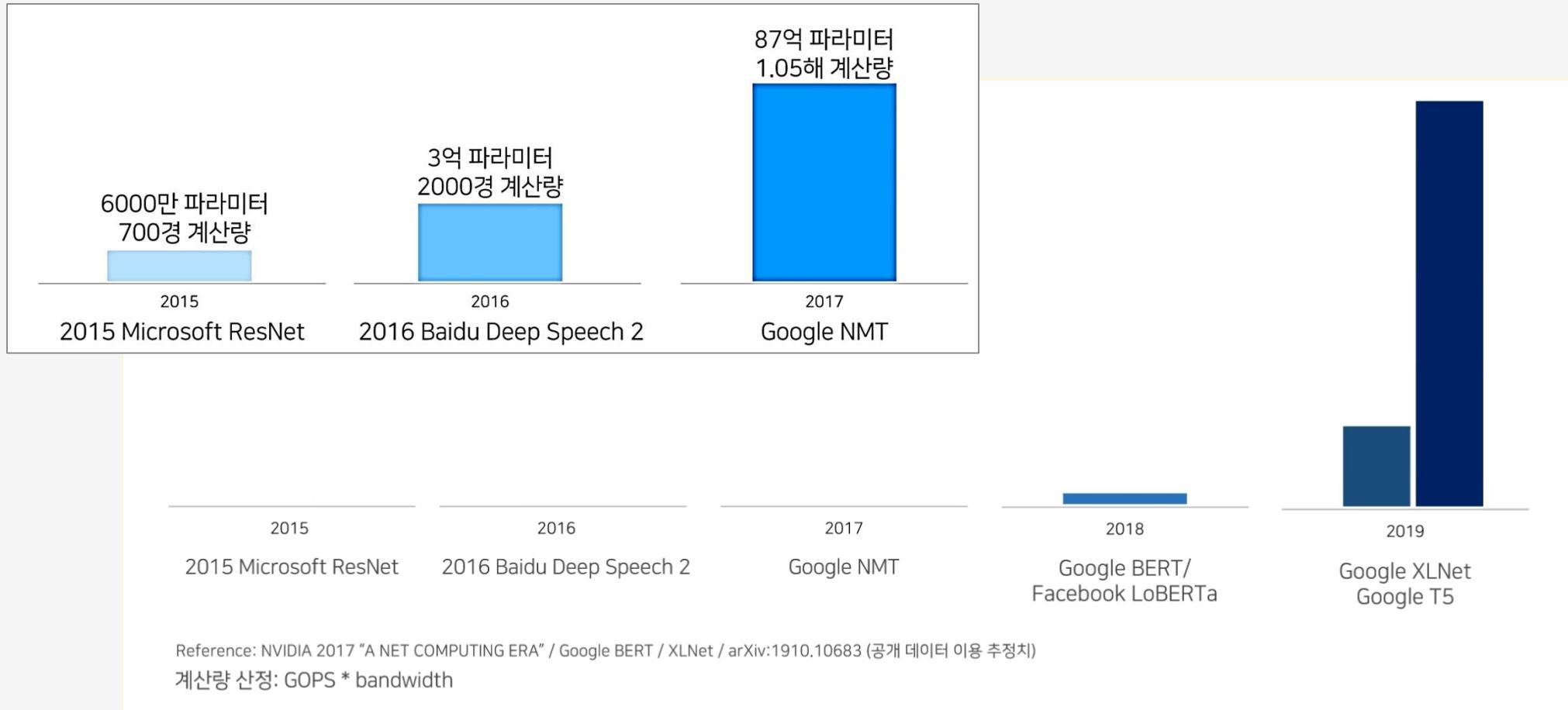
- 많은 수의 코어 & 병렬 연산 가능

2007년 NVIDIA CUDA, 2009년 OpenCL 기술의 등장으로 일반 C/C++ 개발환경을 그대로 쓰면서 GPU에서 돌아가는 코드를 쉽게 작성할 수 있게 됨

이후 수치연산, 머신러닝, 딥러닝 등에서 활용도가 높아지고 있음



# 딥 러닝 모델의 복잡도 증가



# 머신러닝과 GPU

GPU: 일반적인 그래픽 카드

딥러닝 학습 계산의 대부분은 단순 곱셈과 덧셈

많은 연산량: 많은 수의 노드가 개별적으로 계산됨

## Neural Network 기반 알고리즘의 특징

- 기본적으로 행렬 연산 (GPU에 적합)

## 환경 변화

- 머신러닝의 발전 → GPU 수요 증가 → GPU 클라우드 서비스 등장 → 대규모 GPU/ASIC 자원 등장

## AI Accelerator: ASIC 시장의 활성화

- Google TPU (2016~), Intel Nervana (2017), and Movidius (2017)

## GPU cluster

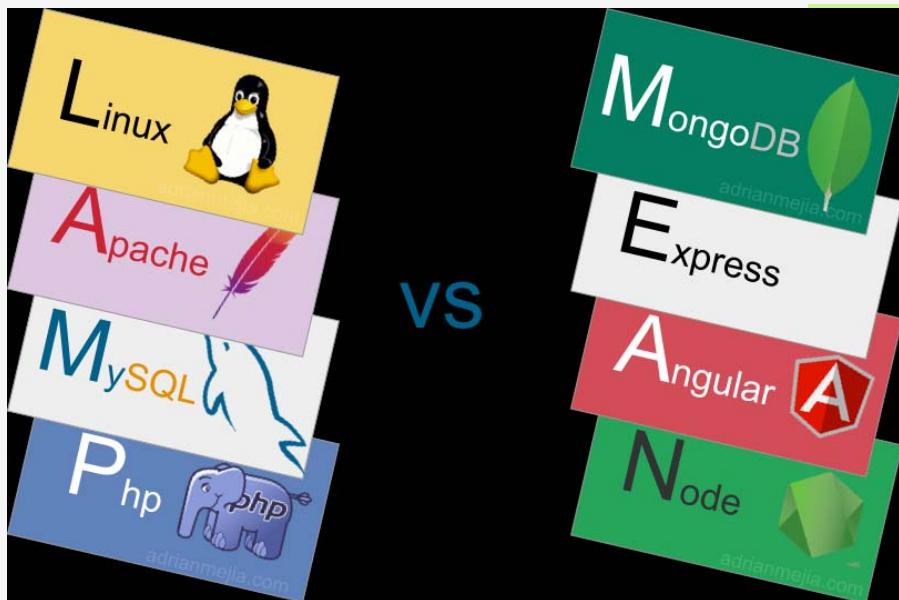
- 초대규모 연산자원(Ultra-large scale calculation resources)의 등장
- 일단 해보는(Trial & error) 식의 접근이 가능해짐

# Front- vs Back-End

---



# 개발 스택들



	LAMP	MEAN
<b>OS</b>	Linux	no need to mention, node is cross platform
<b>Database</b>	mySQL	mongoDB
<b>HTTP Server</b>	apache	node is its own server, Express makes it easier
<b>Serverside Code</b>	PHP	nodeJS applications, may link to angular clientside

# TIOBE Index for August 2020



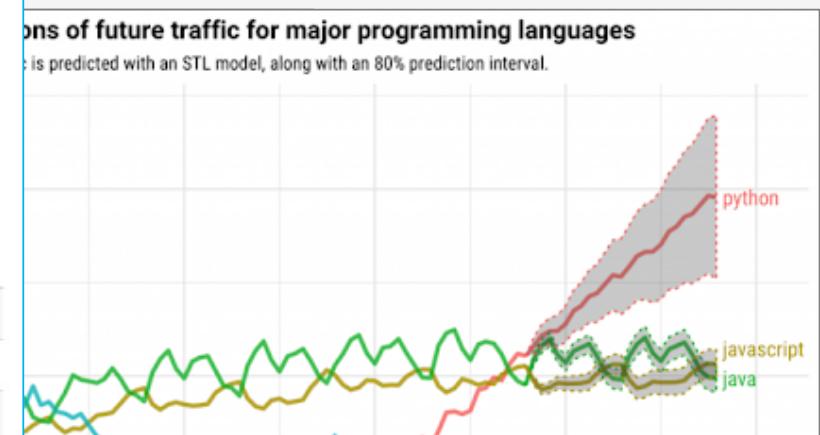
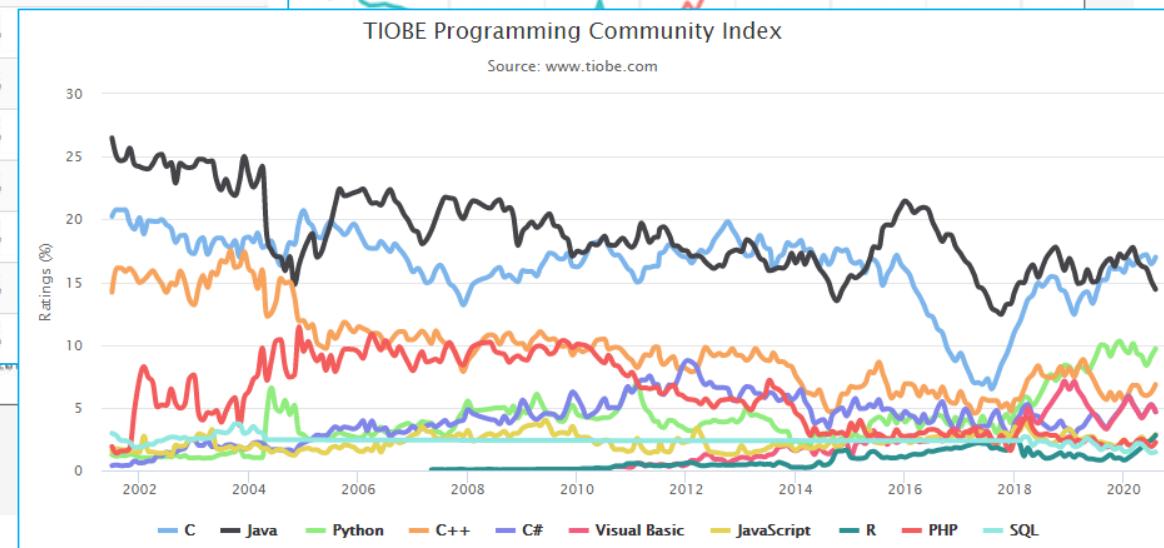
## August Headline: Holiday season in the programming language world

This month nothing much happened in the TIOBE index. The programming language R continues to rise and is on schedule to become TIOBE's programming language of the year 2020. It is also interesting to follow the on-going fight for position #10 in the TIOBE index between Go, Swift and SQL. Swift lost 2 positions this month (from #10 to #12). SQL took over and is back in the top 10 this time. Also worth noting is Groovy's re-entrance in the TIOBE index top 20 at the expense of Scratch and the fact that Hack entered the top 50 at position #44. - Paul Jansen CEO TIOBE Software

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found [here](#).

Aug 2020	Aug 2019	Change	Programming Language	Ratings	Change
1	2	▲	C	16.98%	+1.83%
2	1	▼	Java	14.43%	-1.60%
3	3		Python	9.69%	-0.33%
4	4		C++	6.84%	
5	5		C#	4.68%	
6	6		Visual Basic	4.66%	
7	7		JavaScript	2.87%	
8	20	▲	R	2.79%	
9	8	▼	PHP	2.24%	
10	10		SQL	1.46%	

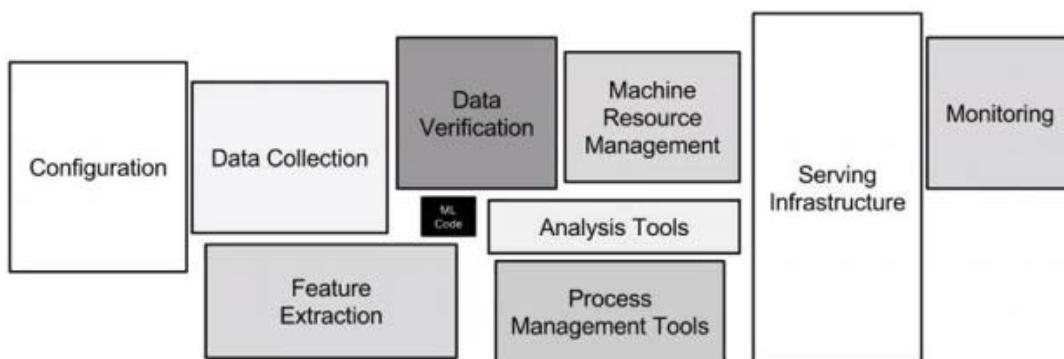


# 초 대규모 인프라 관리의 자동화



## Hidden Technical Debt in Machine Learning Systems

D. Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips  
{dsculley, gholt, dg, edavydov, toddphillips}@google.com  
Google, Inc.



“Only a fraction of real-world ML systems  
is composed of ML code”

- 비자동화된 부분이 상당히 많음
  - 설정, 데이터 수집, 검증, 특징 추출, 리소스 관리, 인프라스트럭처 관리 등
- 각 부분별 새로운 자동화 도입
  - 클라우드 벤더들의 서비스 제공
    - ✓ AWS
    - ✓ Google
    - ✓ Microsoft
  - MLFlow
  - AirFlow
  - KubeFlow

# 딥러닝 개발 스택

## TensorFlow와 Keras

- TensorFlow는 다소 난해
- Keras로 편하게 코드 작성하고  
Keras는 TensorFlow를 호출하여 실행
- 2019년 9월에 공식적으로  
TensorFlow에 Keras가 포함됨

사용자 코드

python application

편의성 좋은 프레임워크

Keras

## 구현 언어

- 개발된 OS 코드의 95% 정도가 Python
- 선택의 여지 없이 Python으로 개발 필요
- 프레임워크 60%가 Keras

백엔드 프레임워크

TensorFlow / PyTorch

하드웨어 사용  
라이브러리

Cuda / CuDNN

하드웨어

CPU / GPU

# Jupyter Notebook

웹을 기반으로 작동: 코드를 서버로 전달하여 실행하고 그 결과를 웹 화면으로 보여줌

- 작업 페이지 내에 문서도 작성하고 그림도 출력

사용자 컴에 개발 환경이 없어도 됨.

작업 결과를 파일로 저장 가능 (ipynb 파일)

## Colab

구글의 Jupyter 서비스. 무료로 GPU 사용 가능

구글 계정(gmail 계정) 생성 만으로 사용준비 끝

<https://colab.research.google.com>

The screenshot shows the Google Colab interface. At the top, there's a browser-like header with back, forward, and refresh buttons, followed by the URL 'colab.research.google.com/drive/1E0CG30zeIDFOT6ukDiJ9OW54v2dwufe4#scrollTo=' and a lock icon. Below the header are two tabs: '+ 코드' (Code) and '+ 텍스트' (Text). A large text area titled '주피터 노트북' (Jupyter Notebook) contains the following text:  
코드를 작성하고 실행시키면  
코드가 Jupyter 서버로 전송되고  
실행된 결과가 이 페이지에 보여집니다.  
Below this text is a code cell with a play button icon and the Python code 'print("hello Keras")'. The output of the cell is 'hello Keras'.

# Kaggle

---

머신러닝 공개 문제 서비스

문제와 데이터, 그리고 답이 있다.

무료 Jupyter를 사용할 수 있다.

<https://www.kaggle.com>

# 딥러닝 알고리즘과 자동화

---

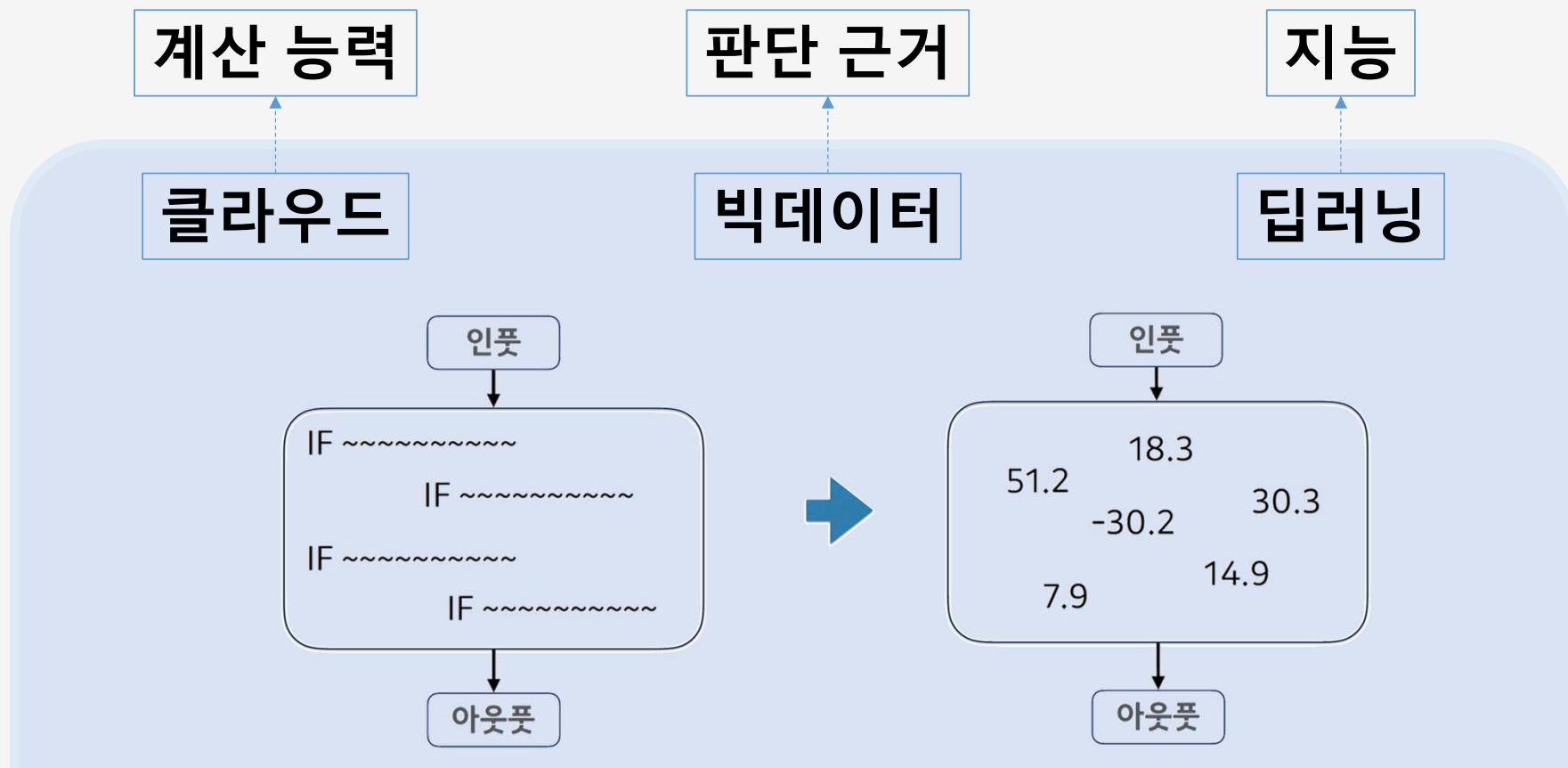
## 딥러닝 알고리즘

- 최근 관심이 높아진 머신러닝 방법론
- 연산자원 증가로 인한 혜택

## 딥러닝 알고리즘 자동화

- 아직 사람 손이 많이 감
- 자동화가 어려운 부분이 많음: 수동설계, 하이퍼 파라미터 튜닝, 데이터 최적화 등
- 도메인 전문가 + 머신러닝 전문가 + 연산 리소스 관리자 필요

# Paradigm Shift - Data-based Foresight



## Machine Learning

Application을 수정하지 않고도 data를 기반으로 pattern을 학습하고 결과를 예측하는 알고리즘 기법을 통칭

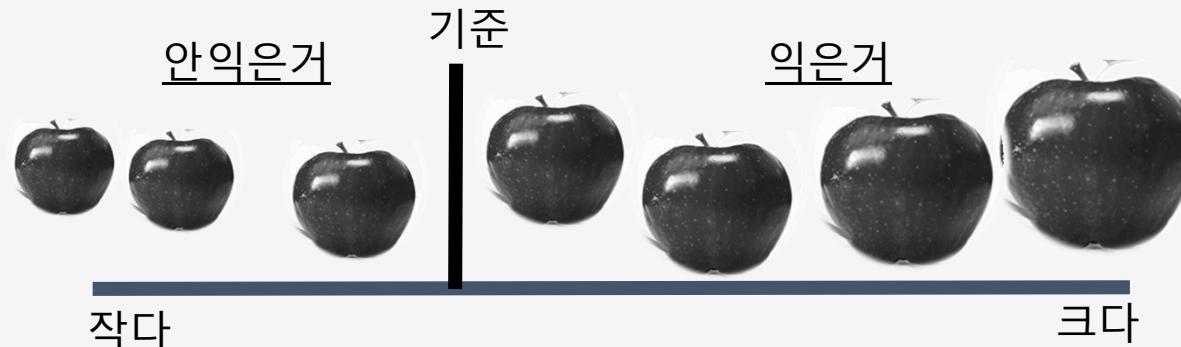
# Machine Learning 원리 이해



# 사과 익은 것 분류하기

## 사람

사과를 보고 감을 동원하여 익은 것과 안 익은 것을 두개의 상자에 분류해 넣는다.



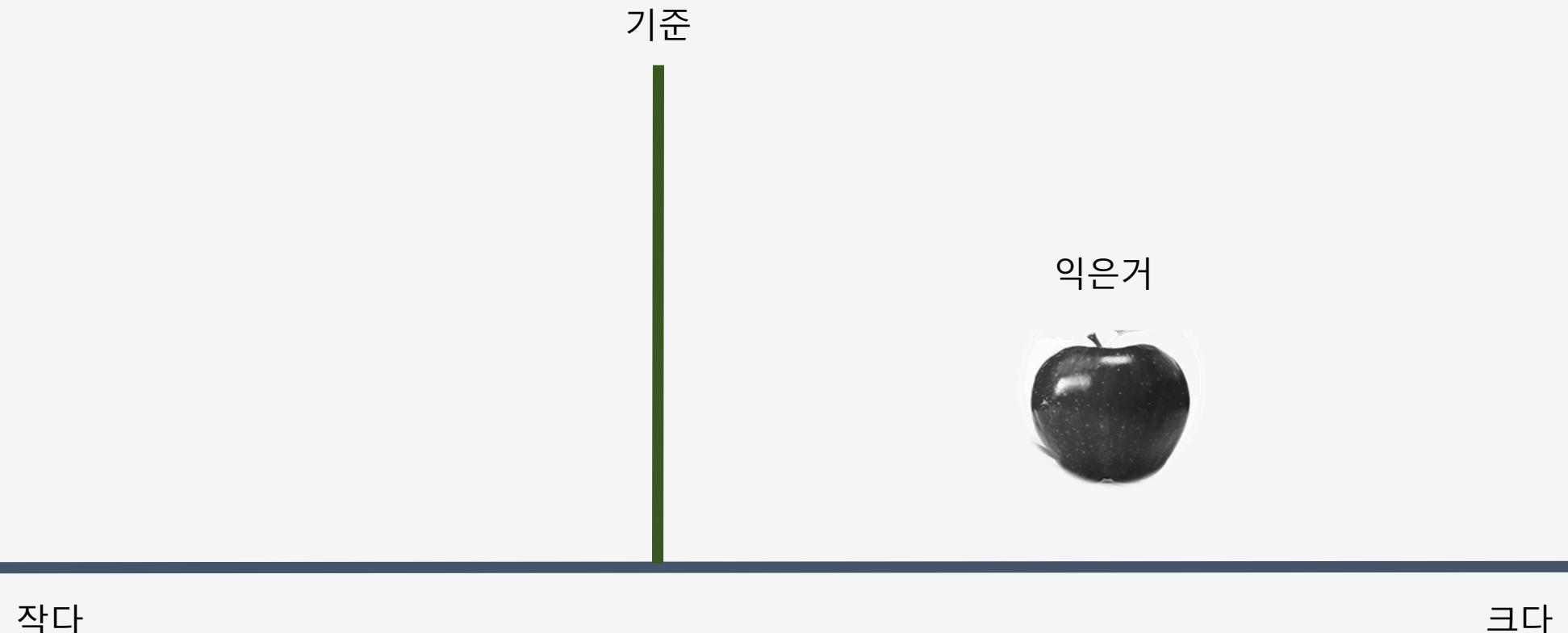
## 로봇

로봇을 위한 프로그램을 작성한다.

- ###이면 우측 상자에, 그렇지 않으면 좌측 상자에 넣는다.
- 분류할 때 사용한 기준을 코딩해 넣어야 한다.  
익은 사과의 크기를 기준으로 코딩  
'if(size > 7.5) then ...'
- 가능한 기준: 한 알의 크기, 색깔 등

# 사과 익은 것 스스로 기준 찾기

---



# 사과 익은 것 스스로 기준 찾기

기준

찾았음!

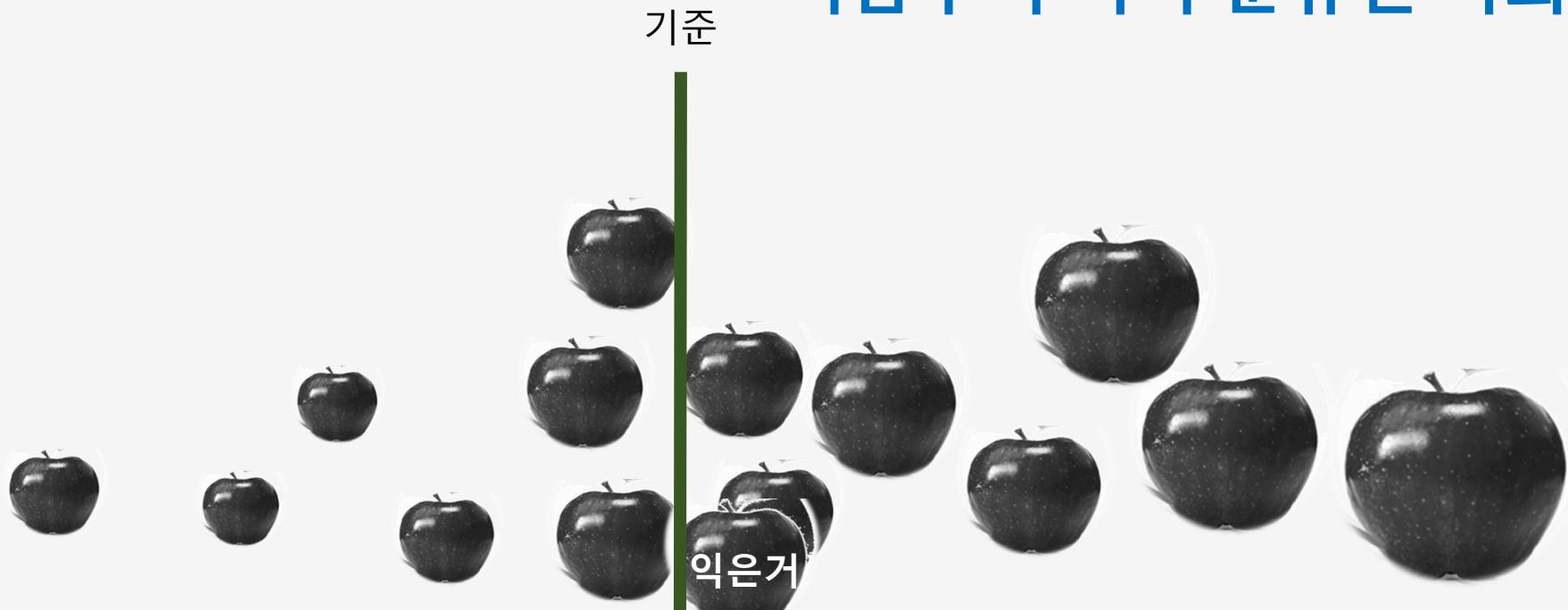


작다

크다

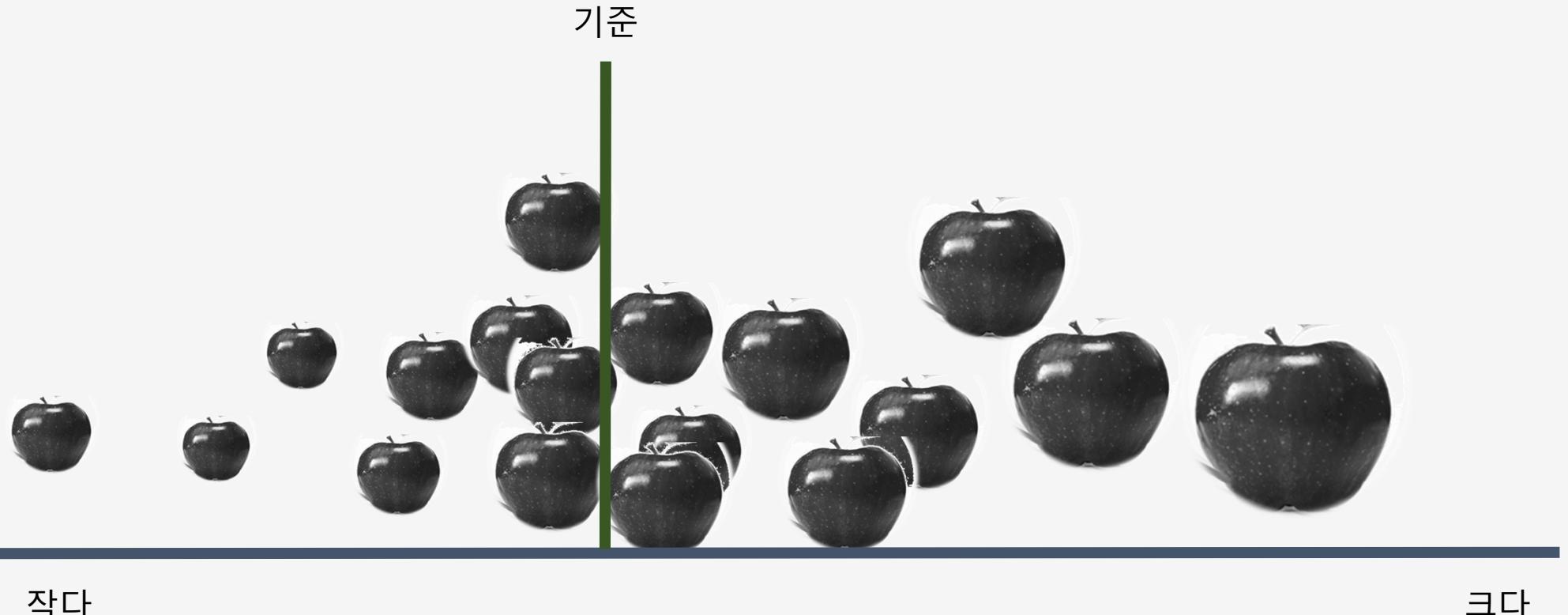
# 사과 익은 것 스스로 기준 찾기

지금부터 이미 분류된 사과로 검증



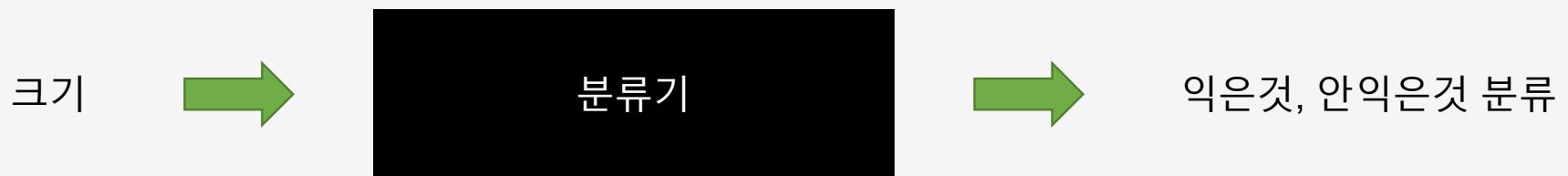
# 사과 익은 것 스스로 기준 찾기

---



# Classifier

입력 값에 따라 몇 개의 그룹으로 분류한다



## 용어정리

- Model: 분류를 위해 '크기를 재서 기준 크기를 가지고 구별'
- Feature: 사과를 구분하는데 사용된 특성 (크기)
- Training set: 학습시키기 위해 사용한 (분류되어 있던) 2개의 사과 상자
- Iteration, Step, Epoch: 학습을 반복하는 과정
- Learning, Training: 원하는 기준 크기를 찾아내는 과정
- Test set, Evaluation set: 검증하기 위한 사과

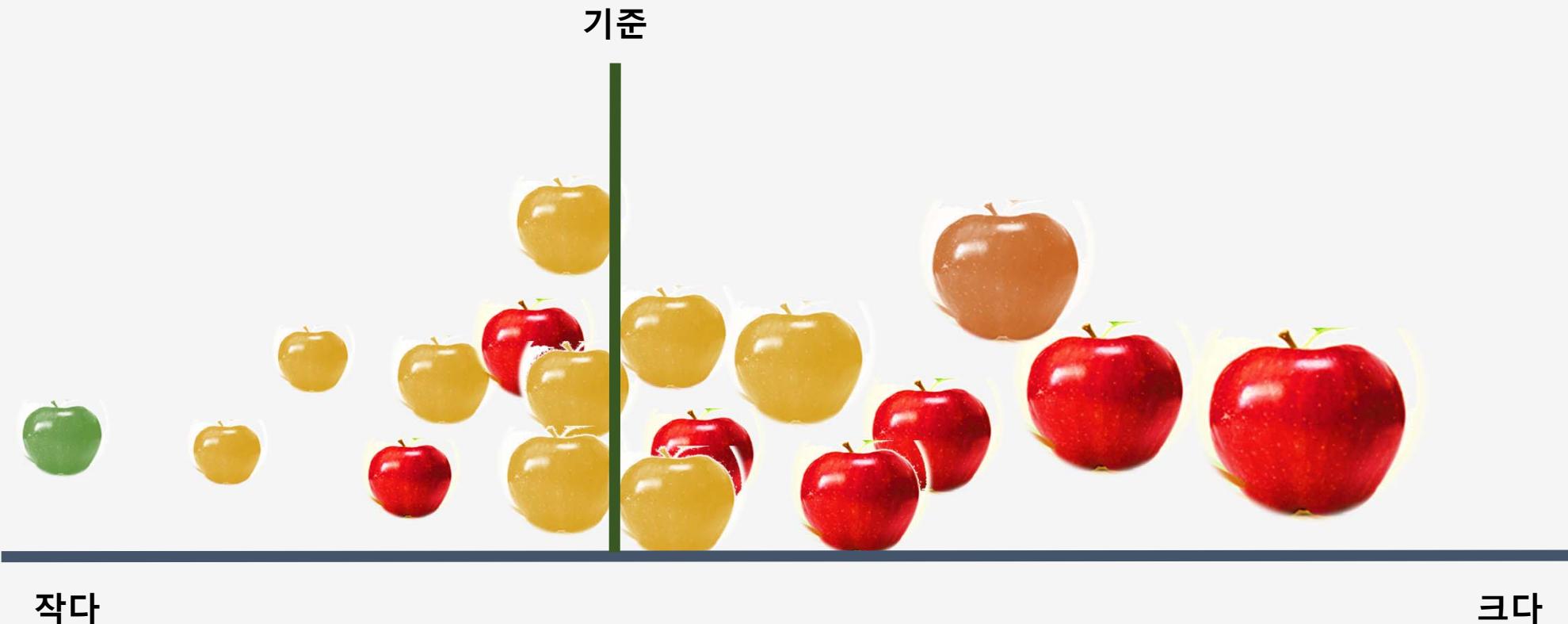
## 결과

---

- Training set으로는 100% 정확하게 학습했다 하더라도 test set으로는 결과가 안 좋을 수 있음
- 실제의 문제는 feature가 복잡하게 얹혀 있음

# 크기만으로는 익은 사과가 명확하게 분류가 안됨

분류해 놓은 것을 보니, 크기가 큰데도 덜 빨간 것이 있다.



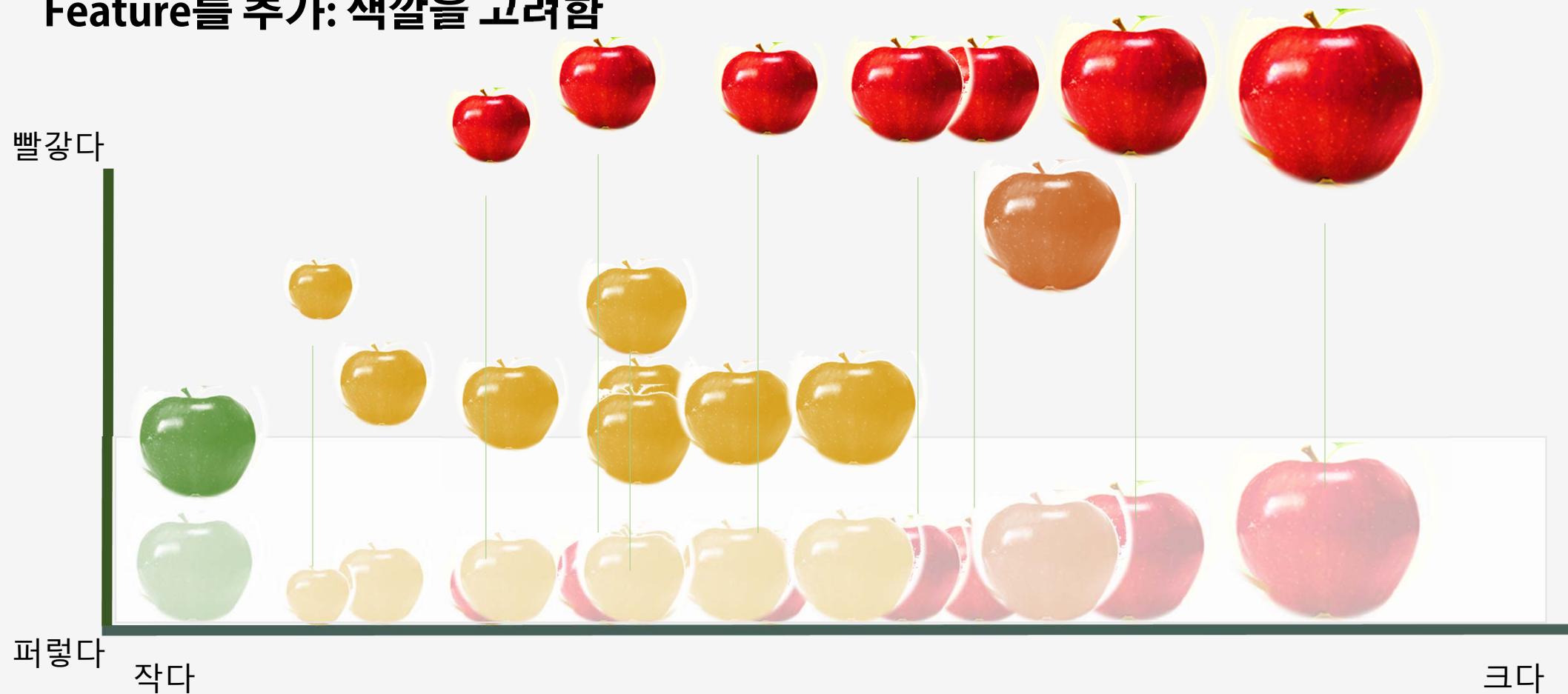
# 기준 다시 찾기

Feature를 추가: 색깔을 고려함



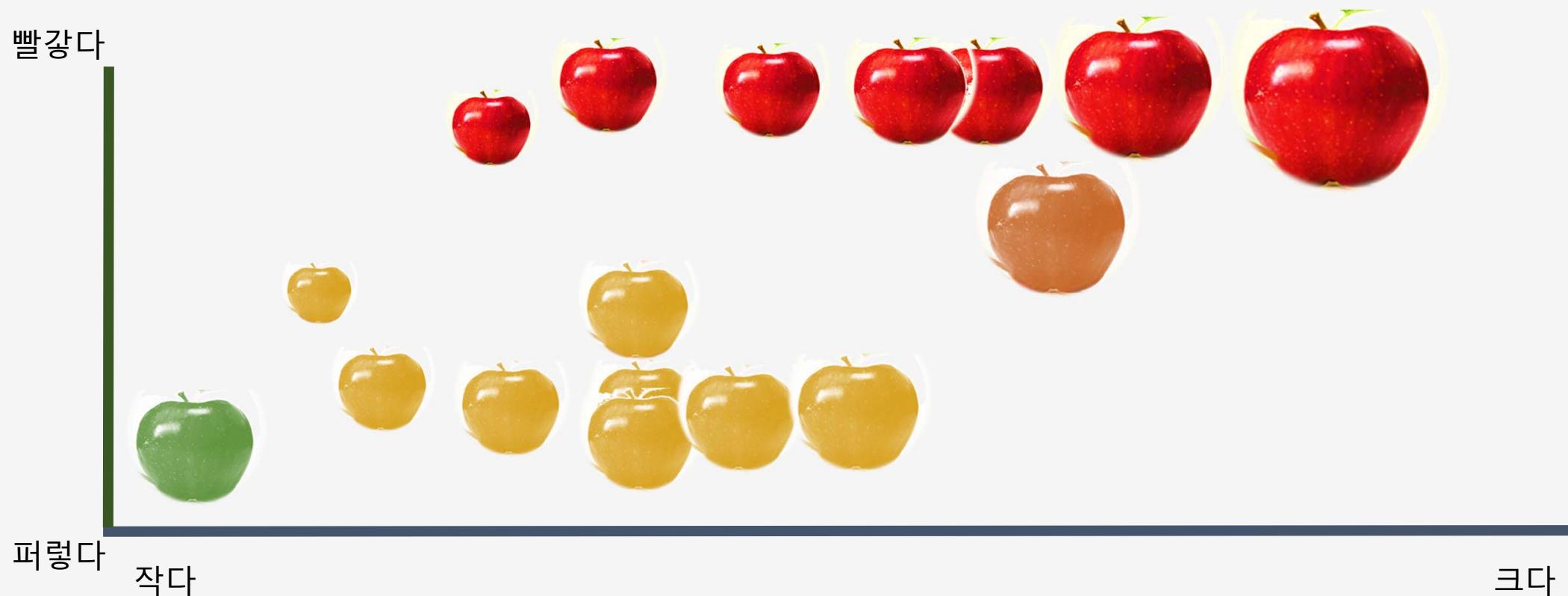
# 기준 다시 찾기

Feature를 추가: 색깔을 고려함



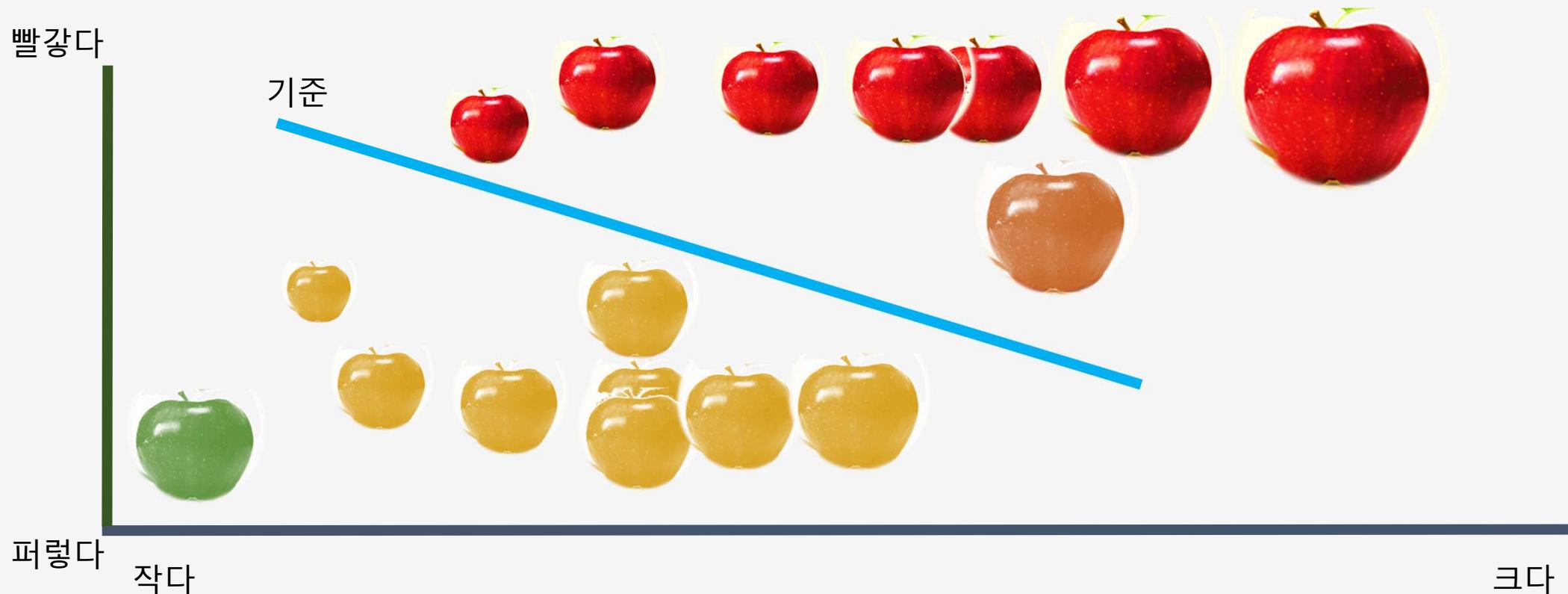
# 기준 다시 찾기

Feature를 추가: 색깔을 고려함

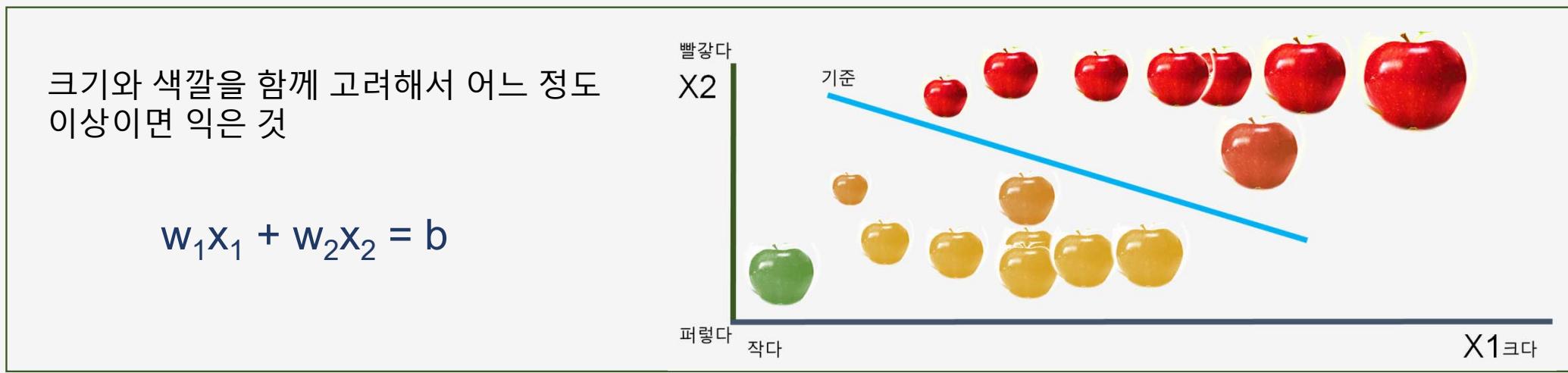
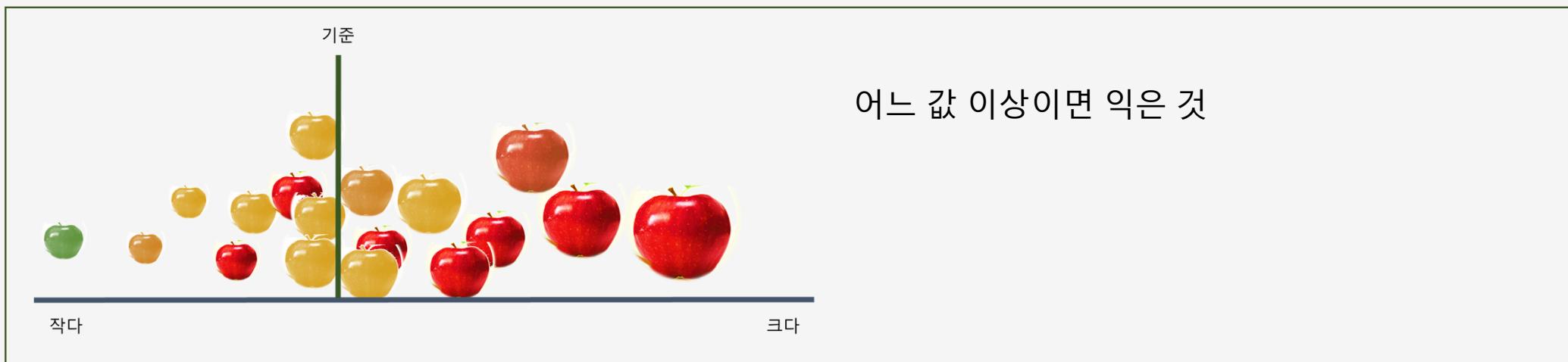


# 기준선 굿기

Feature를 추가: 색깔을 고려함



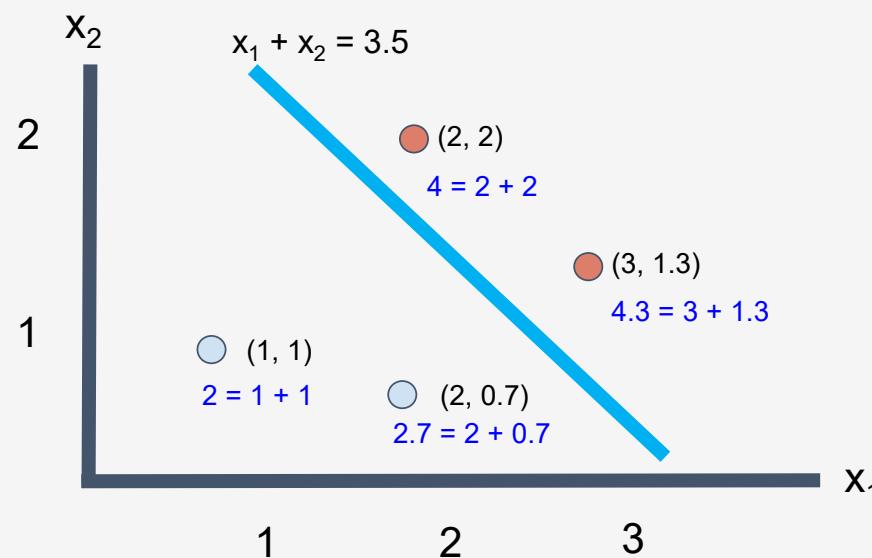
# 분류의 기준 - 값 vs 선



# 구분하는 선

$$x_1 + x_2 = 3.5$$

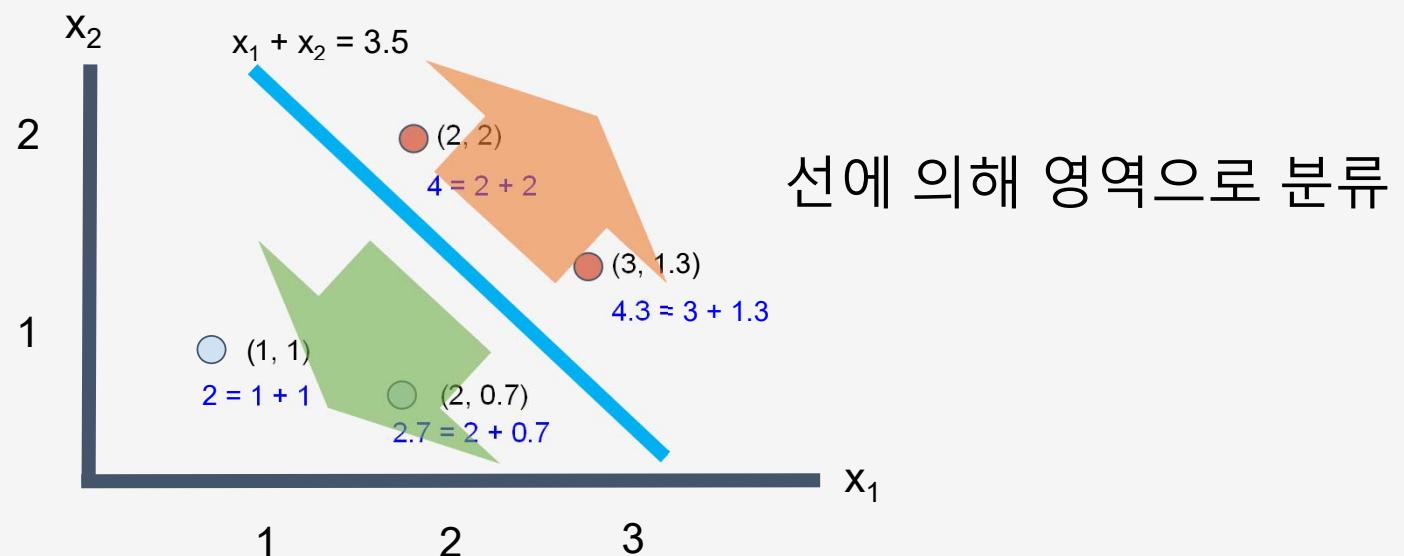
크기의 정도( $x_1$ )와 색깔 정도( $x_2$ )를 합쳐서 값이 3.5를 기준으로 익은 것과 안익은 것을 분류



# 구분하는 선

$$x_1 + x_2 = 3.5$$

크기의 정도( $x_1$ )와 색깔 정도( $x_2$ )를 합쳐서 값이 3.5를 기준으로 익은 것과 안익은 것을 분류



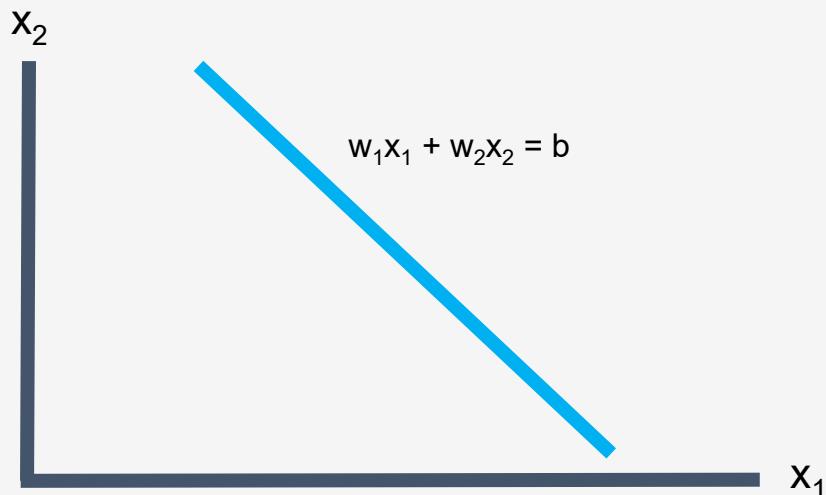
# 기계학습을 한다는 건 선을 구성하는 3개의 값을 찾는 것

- 2개 feature에 대한 가중치 ( $w_1, w_2$ )
- 기준 값 ( $b$ )

$$w_1x_1 + w_2x_2 = b$$

$$w_1x_1 + w_2x_2 - b = 0$$

- 계산 값이 0보다 큰지 작은지에 따라 구분 기준 설정

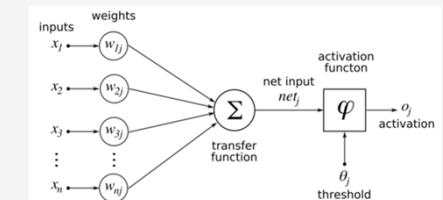
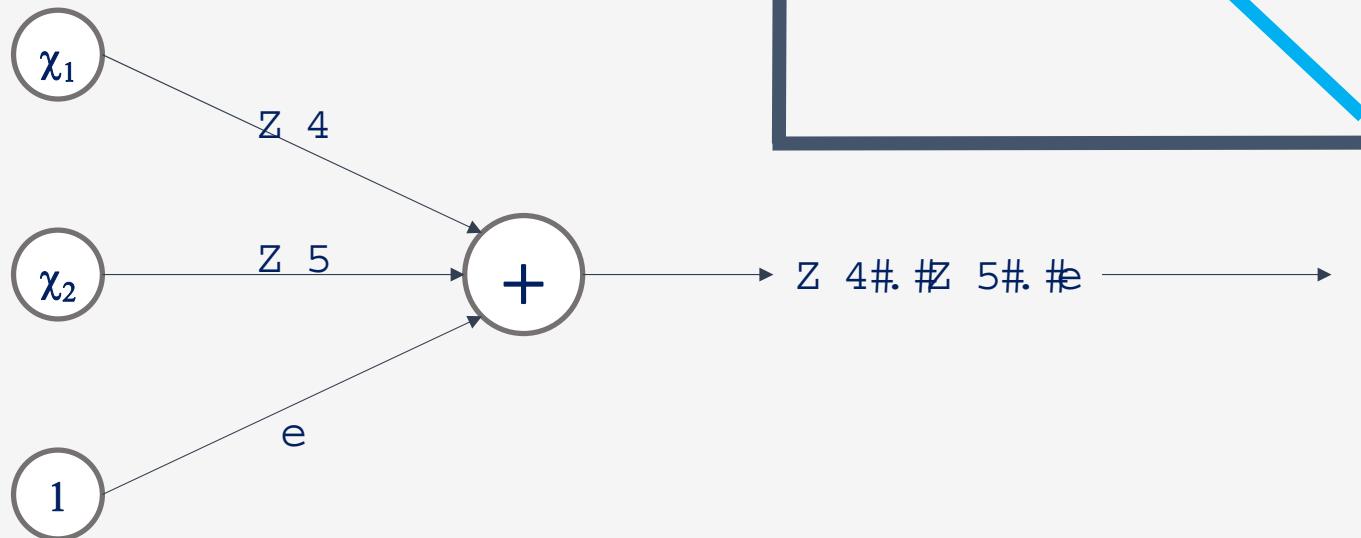


# 선에 대한 식을 달리 나타내면

계산 값이 0보다 크면 우측 (익은 사과)  
0보다 작으면 좌측 (안익은 사과)

$$w_1x_1 + w_2x_2 = b$$

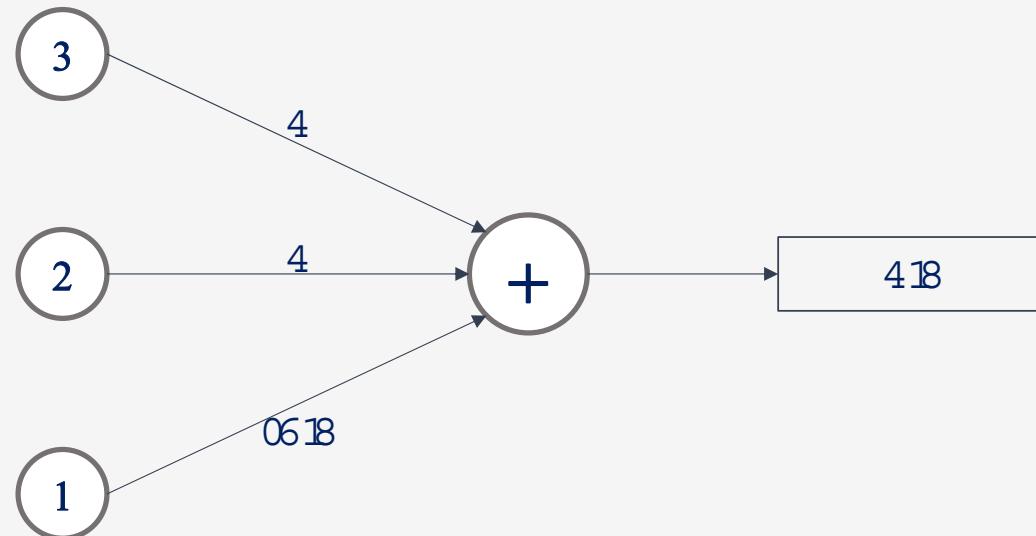
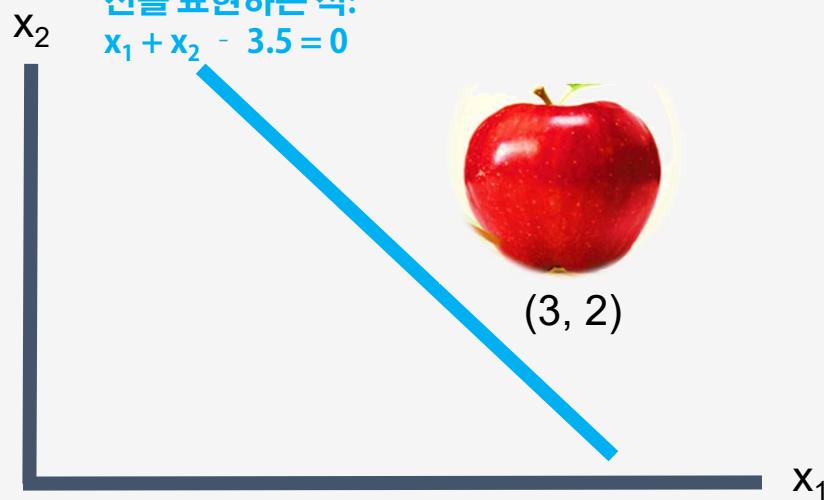
$$w_1x_1 + w_2x_2 - b = 0$$



# 익은 사과에 대한 그림 표현

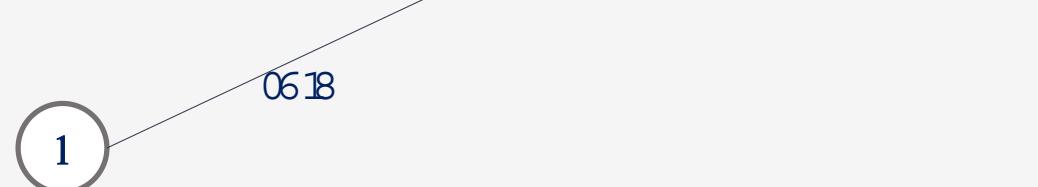
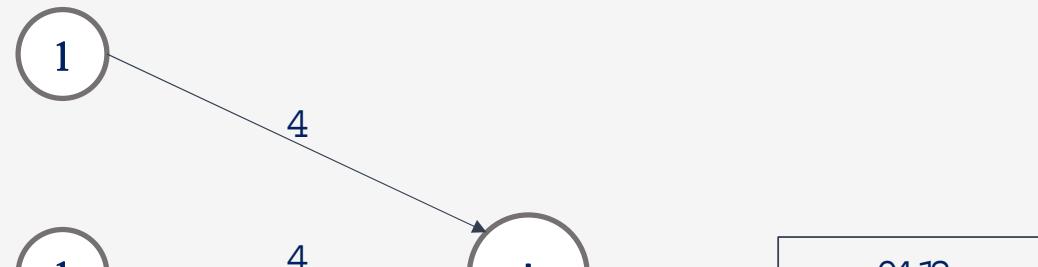
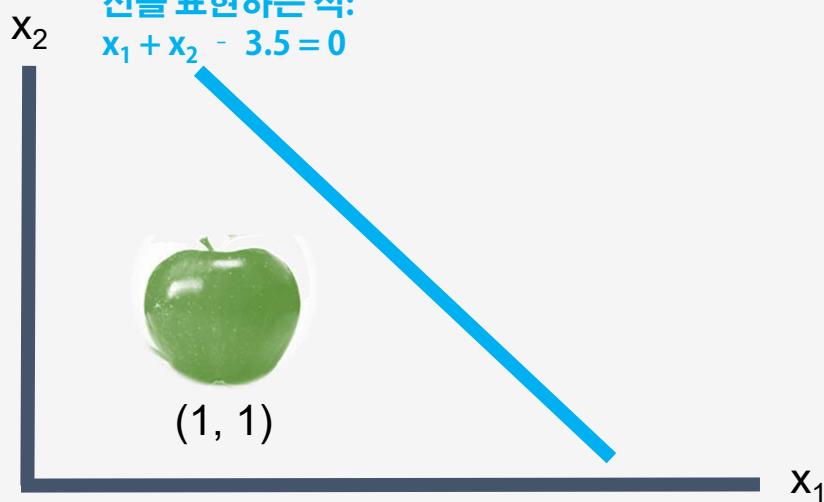
크기가 3이고, 색깔이 2인 경우 계산 값은 1.5: 사과는 선의 우측에  
있음

익은 사과



# 안익은 사과에 대한 그림 표현

크기가 1이고, 색깔이 1인 경우 계산 값은 -1.5: 사과는 선의 좌측에  
있음

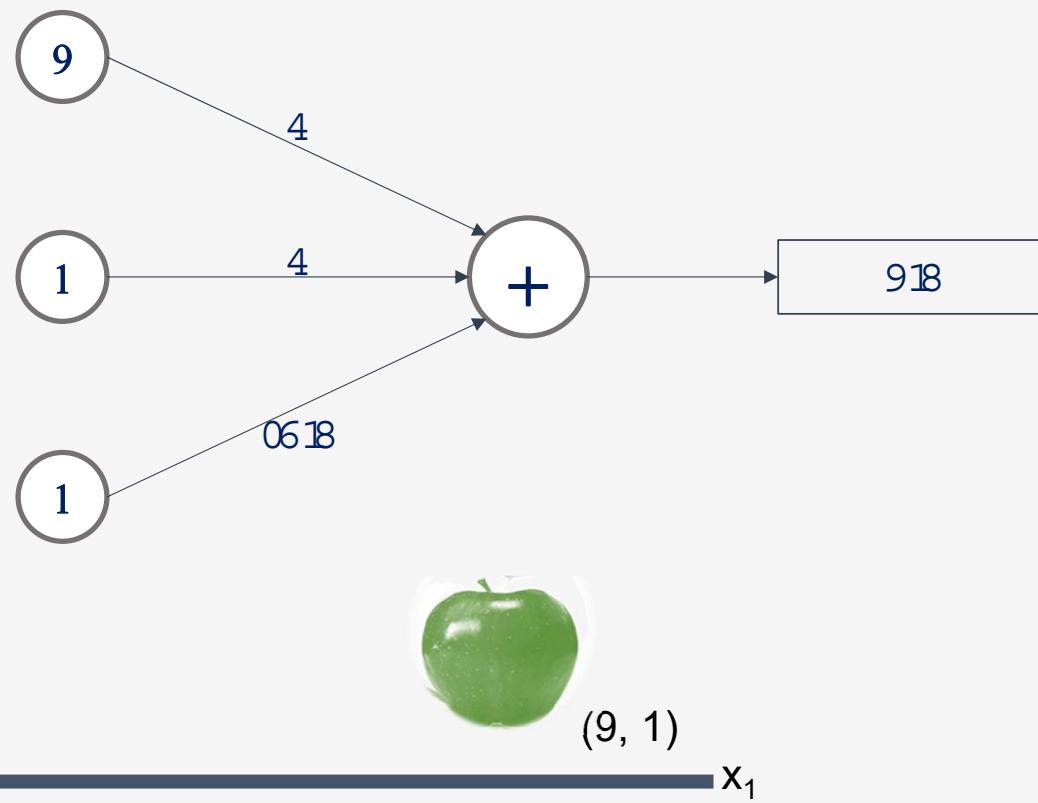
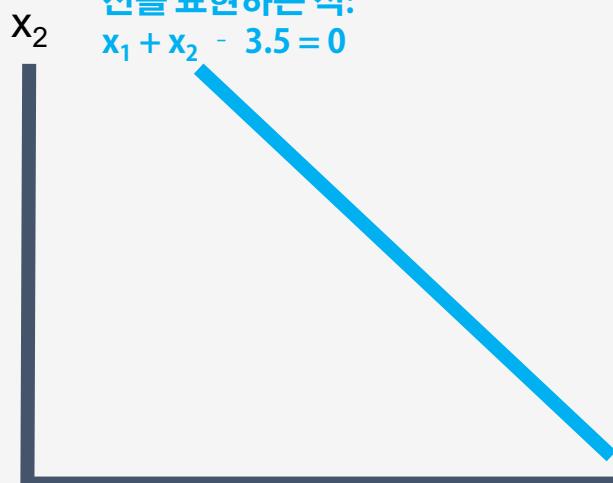


# 분류가 잘못된 경우

안익은 사과의 크기가 9이고, 색깔이 1인 경우 계산 값은 6.5:

??

안익은 사과가 선의 우측에 존재함!



# 바르게 분류되도록 선을 조정함

안익은 사과의 크기가 9이고, 색깔이 1인 경우 계산 값은 6.5:

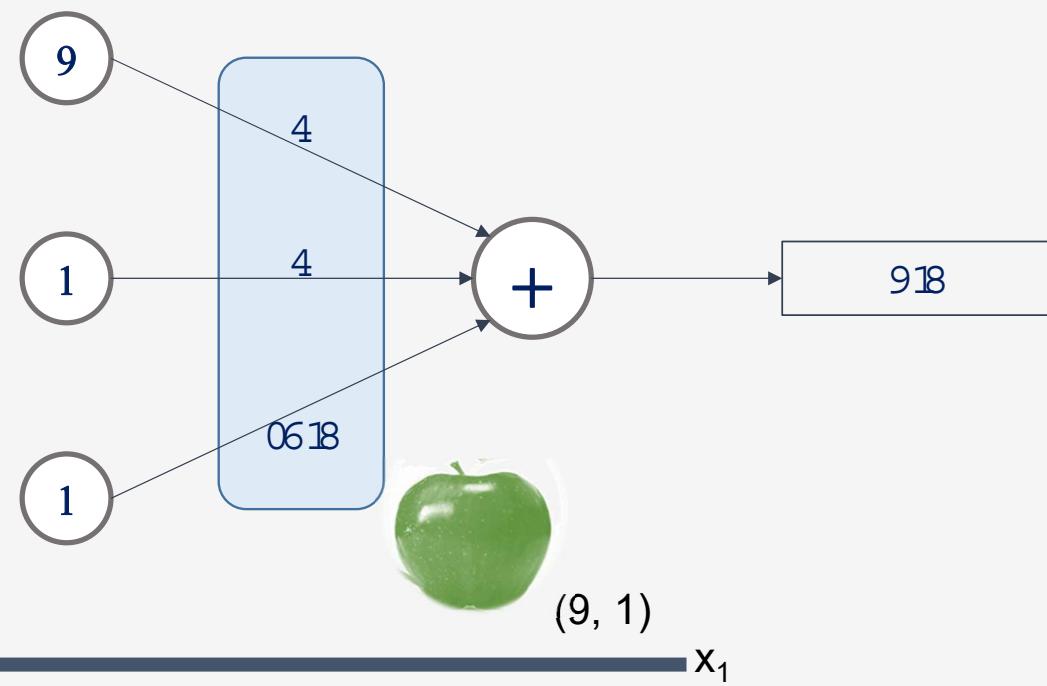
??

안익은 사과가 선의 우측에 존재함!

선을 나타내는  $1, 1, 3.5$ 의 값을 조정해서 계산 값이 0보다 작게 만들어야 함

$x_2$

선을 표현하는 식:  
 $x_1 + x_2 - 3.5 = 0$



# 바르게 분류되도록 선을 조정함

안익은 사과의 크기가 9이고, 색깔이 1인 경우 계산 값은 6.5:

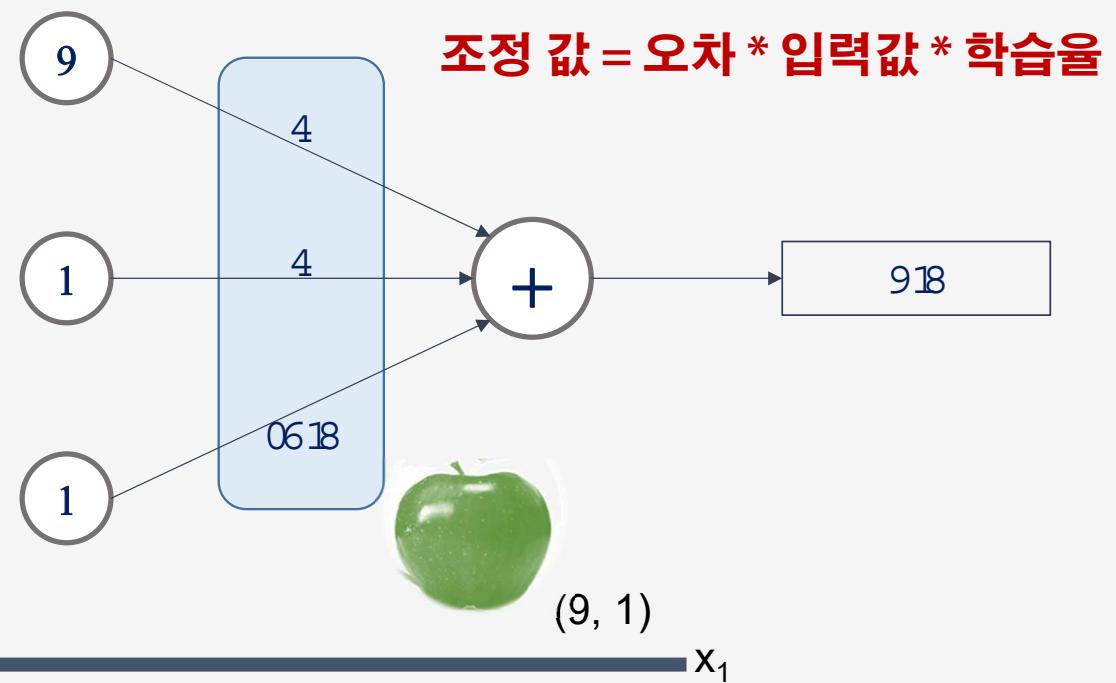
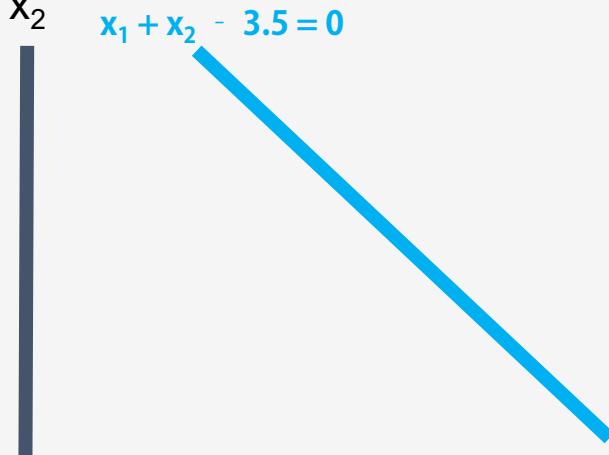
??

안익은 사과가 선의 우측에 존재함!

선을 나타내는  $1, 1, 3.5$ 의 값을 조정해서 계산 값이 0보다 작게 만들어야 함

→ 계산 값에 영향을 크게 미치는 '크기' 값이 더 크게 조정되도록 weight 값을 조절

선을 표현하는 식:  
 $x_1 + x_2 - 3.5 = 0$



# 선을 조정하는 방법

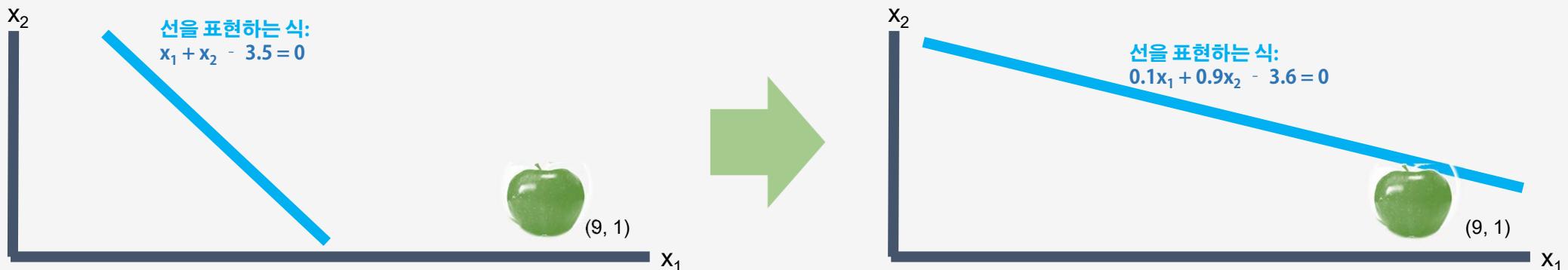
$$\text{Weight}_{\text{new}} = \text{Weight}_{\text{old}} + \text{adjustment}$$

$$\text{adjustment} = \text{error} * \text{input} * \text{learning rate}$$

크기 비중 조정:  $\text{New Weight}_{x_1} = \text{Old Weight}_{x_1} - (1 \times 9 \times 0.1) = 1 - (1 \times 9 \times 0.1) = 0.1$

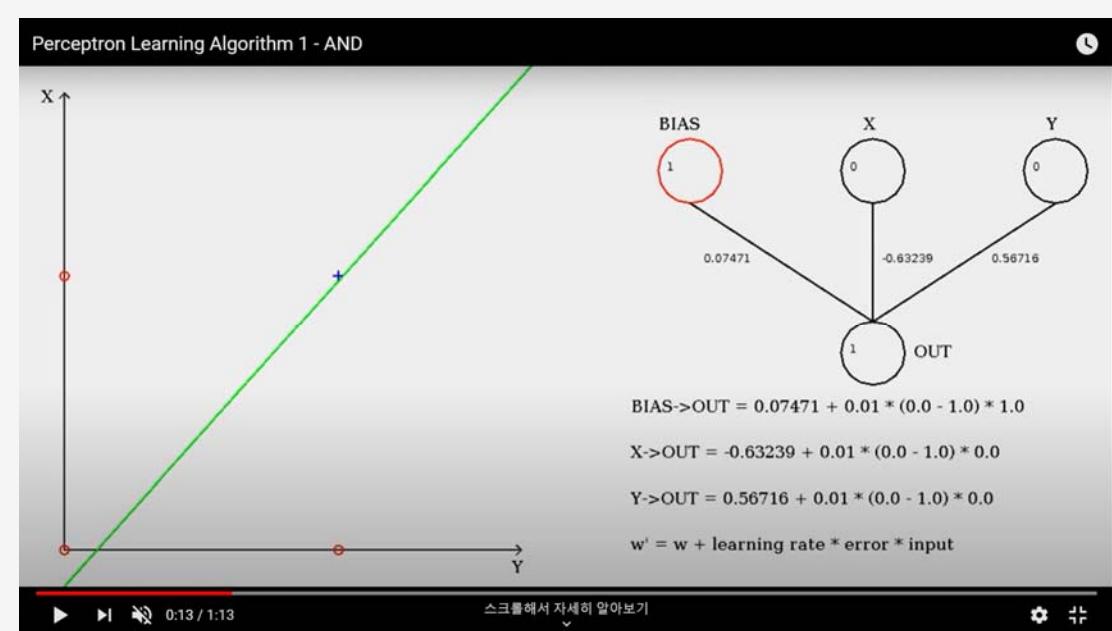
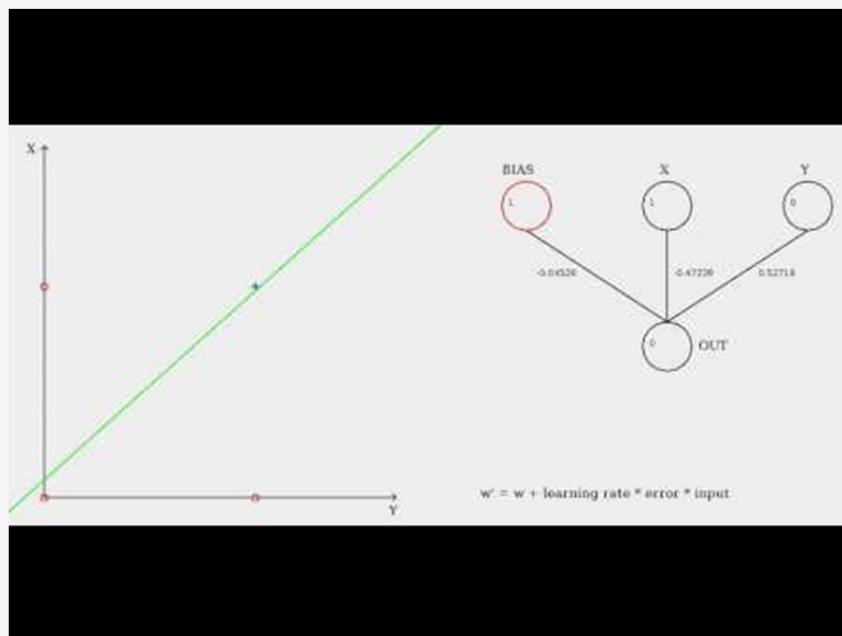
색깔 비중 조정:  $\text{New Weight}_{x_2} = \text{Old Weight}_{x_2} - (1 \times 1 \times 0.1) = 1 - (1 \times 1 \times 0.1) = 0.9$

기준값 조정:  $\text{New Bias} = \text{Old Bias} - (1 \times 1 \times 0.1) = -3.5 - (1 \times 1 \times 0.1) = -3.6$



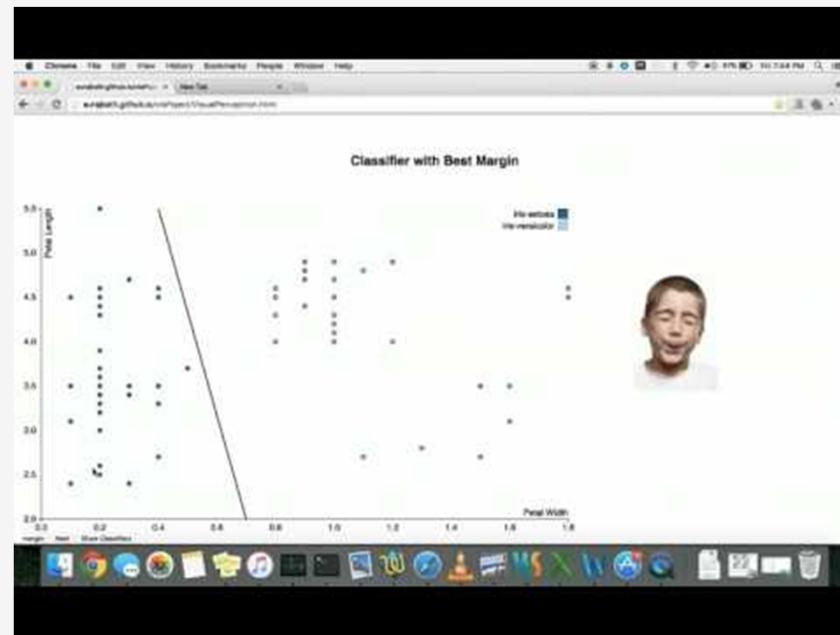
# 선근기 학습 실 예 1

<https://www.youtube.com/watch?v=tYxkIOTdeu8>

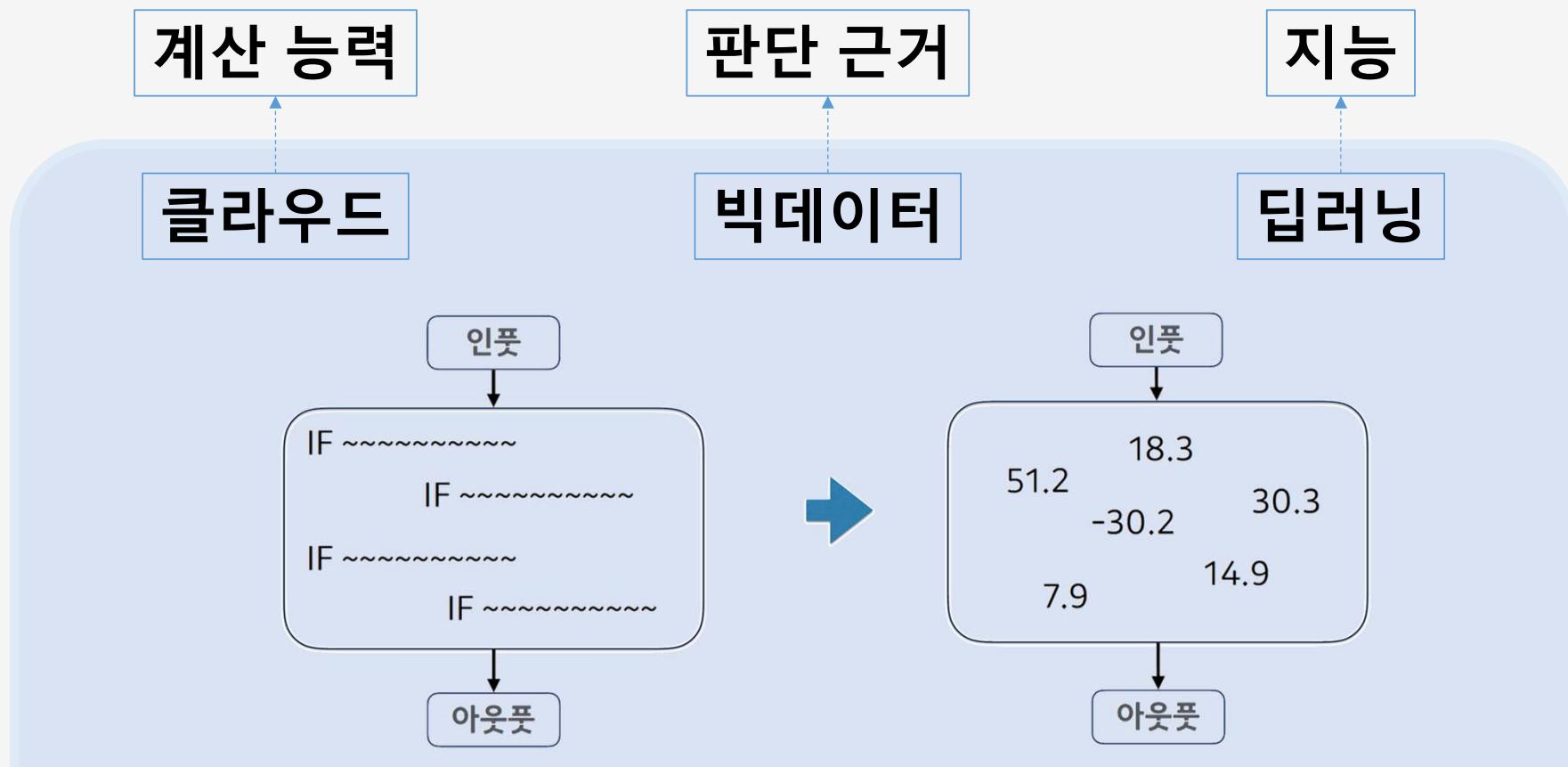


# 선긋기 학습 실 예 2

<https://www.youtube.com/embed/V2MMMyk7bdWY>



# Paradigm Shift - Data-based Foresight



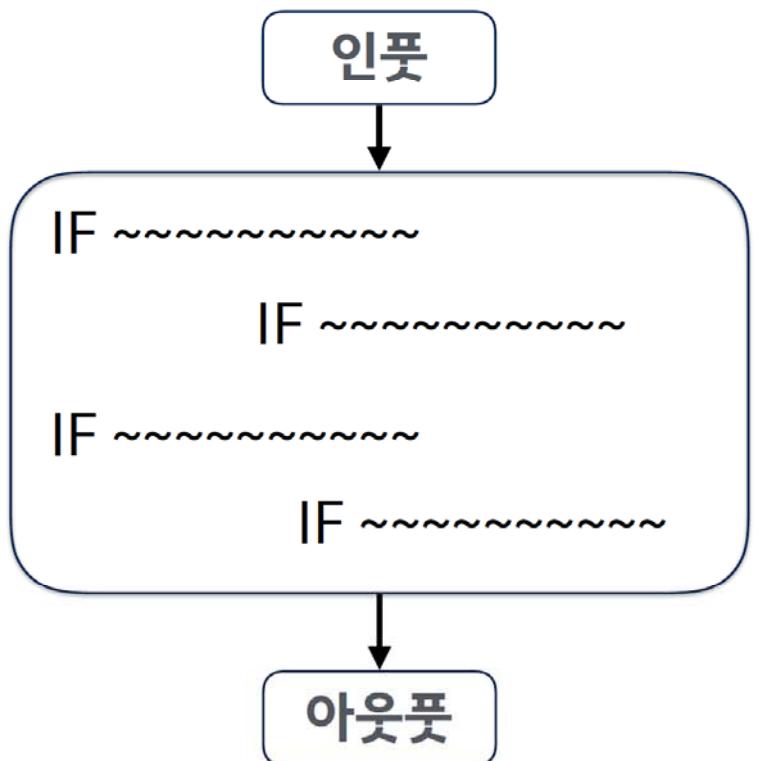
## Machine Learning

Application을 수정하지 않고도 data를 기반으로 pattern을 학습하고 결과를 예측하는 알고리즘 기법을 통칭

# 패러다임 쉬프트

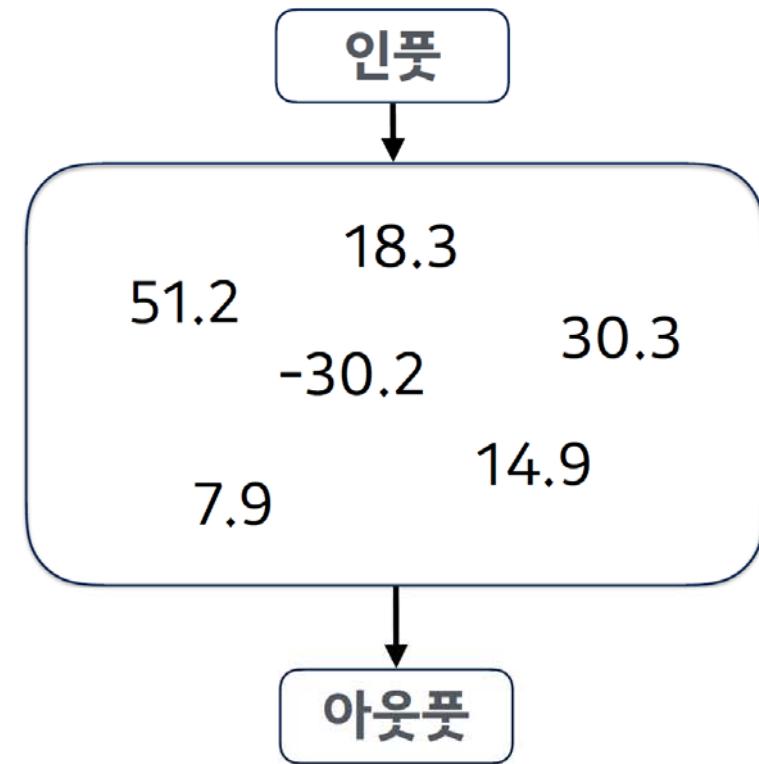
## 기존의 코딩:

조건을 라인바이라인으로 **긴 코드**로 써내려 가는 일



## 앞으로의 코딩:

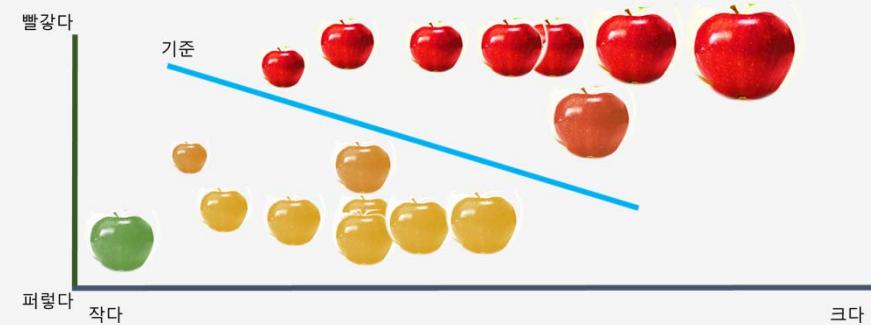
조건을 학습 **모델의 여러 가중치로 변환**하는 일



# 용어정리

## 간단한 로직을 반복하여 기준선을 찾는 과정: 학습

- 크기만으로 구분하니 오류 → 색깔을 추가하니 제대로 구분 (크기, 색깔: 입력, 특질(feature))
- 각 사과마다 2개 값의 특질이 있다. 크기 3, 색깔 2 (입력 벡터, 특질 벡터)
- 사과 예의 경우 2개 특질(크기, 색깔)을 사용 (입력 벡터가 2차원)
- 올바르게 구별하는 선을 구성하는 3개의 값을 찾는 과정 (학습)
- 각 입력에 곱해지는  $w_1, w_2$ 에 해당하는 값 (가중치(weight))
- Model: 분류를 위해 '크기와 색깔의 기준을 수립해서 구별'
- Feature: 사과를 구분하는데 사용된 특성 (크기)
- Training set: 학습시키기 위해 사용한 (분류되어 있던) 2개의 사과 상자
- Iteration, Step, Epoch: 학습을 반복하는 과정
- Learning, Training: 원하는 기준 크기를 찾아내는 과정
- Test set, Evaluation set: 검증하기 위한 사과



# AI 용어설명

Artificial Intelligence

Machine Learning

Neural Network

Deep  
Learning

**머신러닝** 기계가 지능을 가지도록 스스로 학습하는 것

**신경망/인공신경망** 머신러닝 알고리즘 중 인간의 뇌를 구성하는 뉴런에 영감을 받아 형성한 모델/ 입력층-은닉층-출력층

**딥러닝** 은닉층을 깊게 쌓은 신경망 구조를 활용해 학습하는 방법

**퍼셉트론(Perceptron)** 신경망을 구성하는 기본 단위

**활성화 함수(Activation Function)** 입력층으로부터 주입된 데이터에서 규칙을 찾아내기 위해 입력 데이터를 적절하게 변환해주는 함수

**손실 함수(Loss Function)**/ 비용함수(Cost Function)/목적 함수(Objective Function)/오차함수>Error Function) 신경망 학습이 잘 되고 있는지를 확인할 때 사용하는 함수

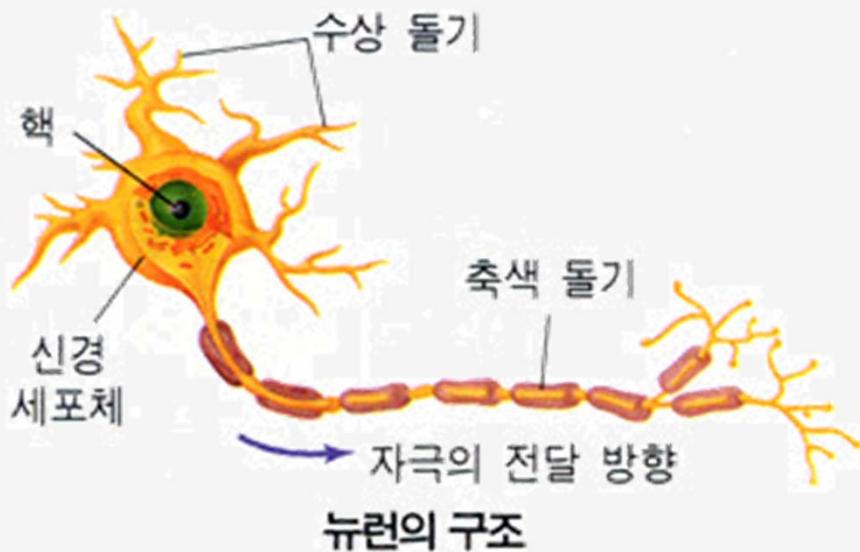
**경사하강법(Gradient Descent)** 신경망을 데이터에 최적화 하기 위해 손실함수의 경사를 구하고 기울기가 낮은 쪽으로 계 속 이동시키는 방법

**역전파 알고리즘(Backpropagation Algorithm)** 딥러닝 네트워크 구조에서 입력을 통하여 나온 경사(미분값)를 역방향으로 다시 보내어 신경망을 업데이트하는 방법

# Artificial Neural Network

## Neuronal Cell

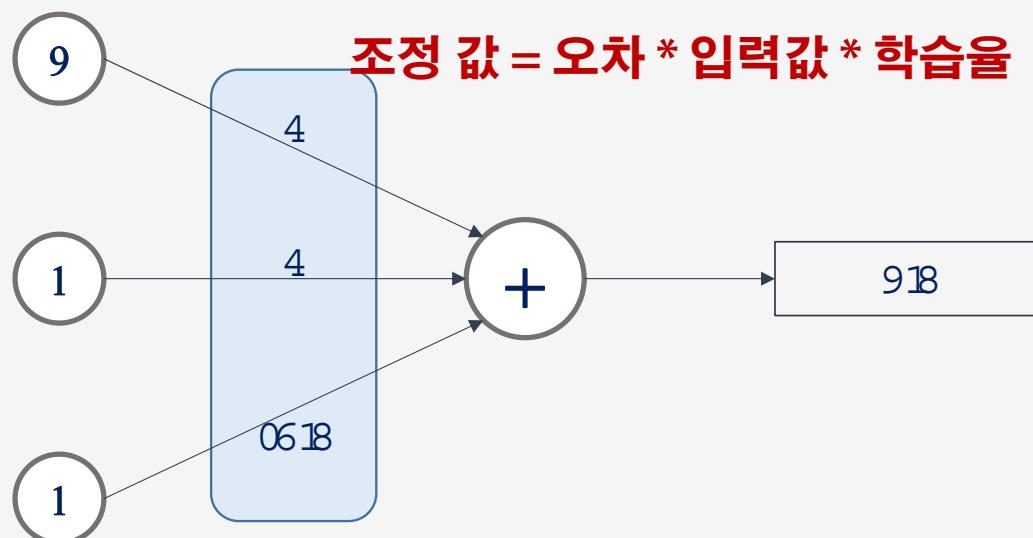
여러개의 수상돌기에서 자극이 합해져서  
그 값이 어느 값 이상일 경우  
축색돌기로 자극을 발생시킨다.



## Perceptron

신경세포와 유사함

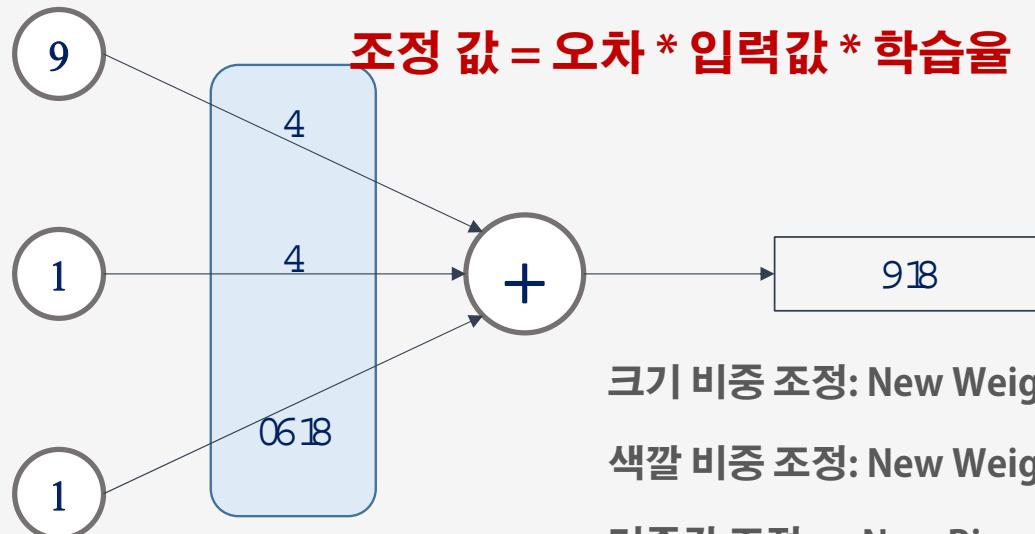
Input으로부터의 자극 중 선택하여 증폭&전달



# 선을 조정하는 방법

$$\text{Weight}_{\text{new}} = \text{Weight}_{\text{old}} + \text{adjustment}$$

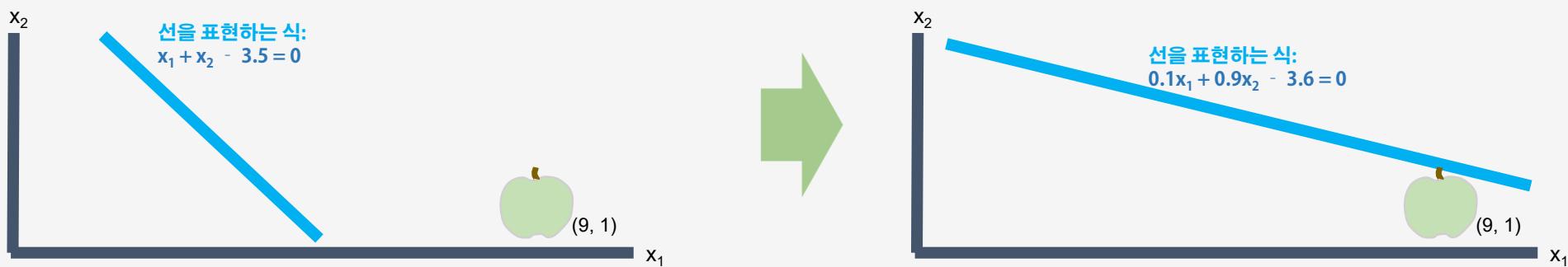
$$\text{adjustment} = \text{error} * \text{input} * \text{learning rate}$$



크기 비중 조정:  $\text{New Weight}_{x_1} = \text{Old Weight}_{x_1} - (1 \times 9 \times 0.1) = 1 - (1 \times 9 \times 0.1) = 0.1$

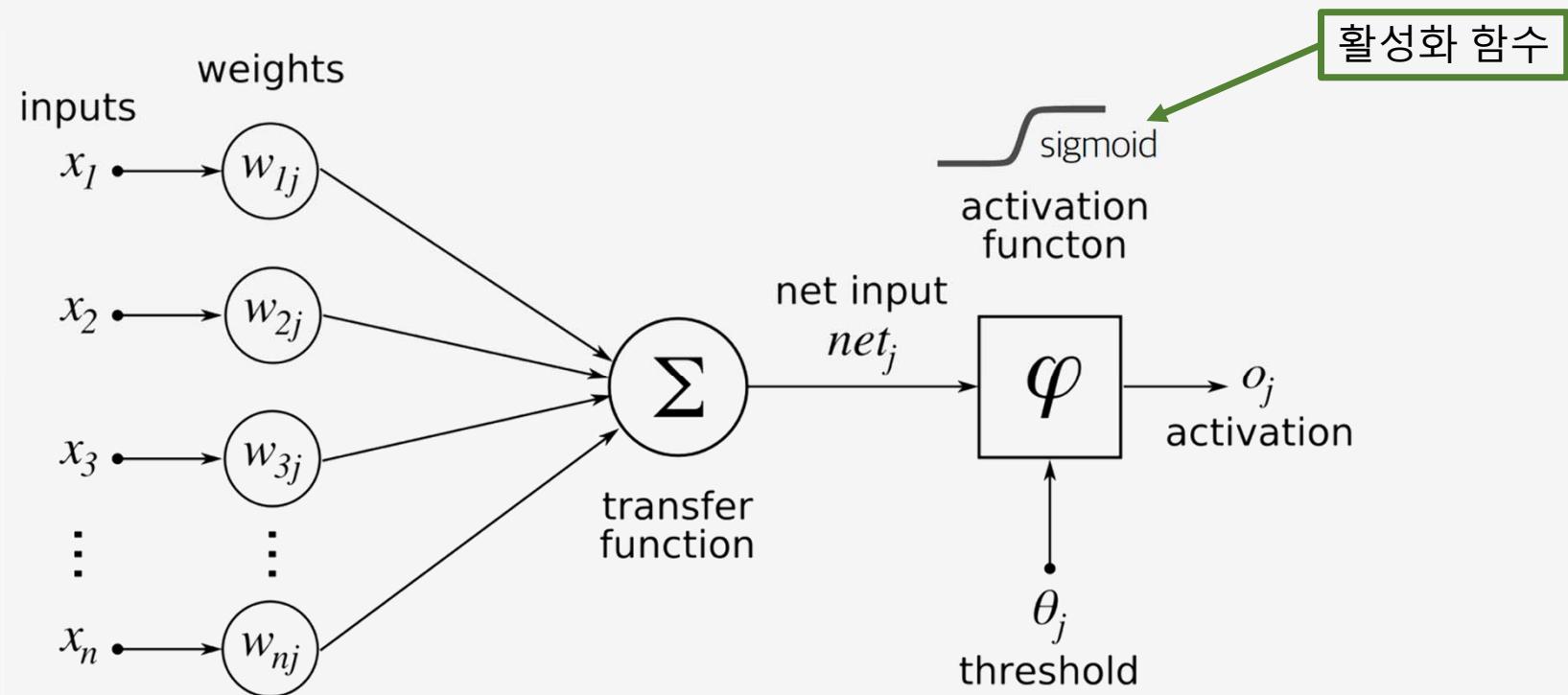
색깔 비중 조정:  $\text{New Weight}_{x_2} = \text{Old Weight}_{x_2} - (1 \times 1 \times 0.1) = 1 - (1 \times 1 \times 0.1) = 0.9$

기준값 조정:  $\text{New Bias} = \text{Old Bias} - (1 \times 1 \times 0.1) = -3.5 - (1 \times 1 \times 0.1) = -3.6$



# 퍼셉트론(Perceptron)

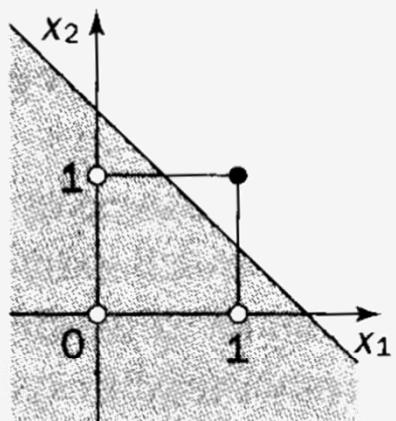
선형분리의 문제를 학습할 수 있다.



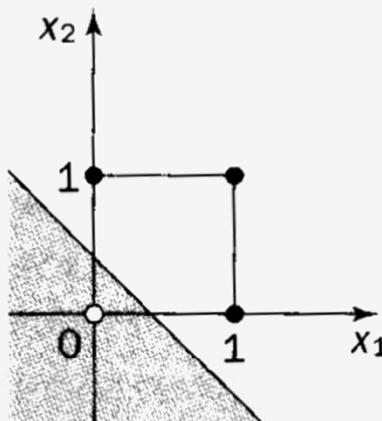
# 퍼셉트론의 한계

선형분리가 불가능한 것은 풀지 못함.

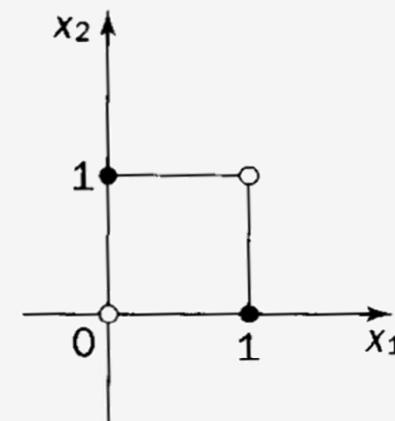
간단한 XOR(Exclusive OR, 배타적 논리합)도 못한다. (XOR는 선형분리로 풀지 못함)



(a) AND ( $x_1 \cap x_2$ )



(b) OR ( $x_1 \cup x_2$ )

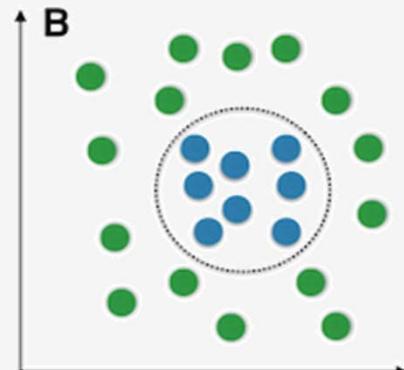
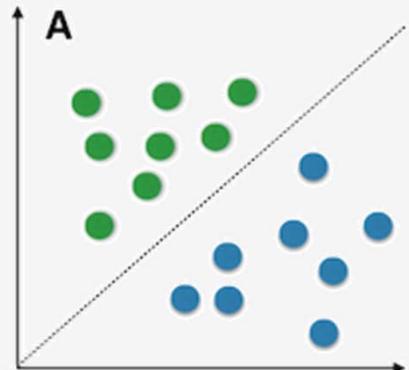


(c) Exclusive-OR  
( $x_1 \oplus x_2$ )



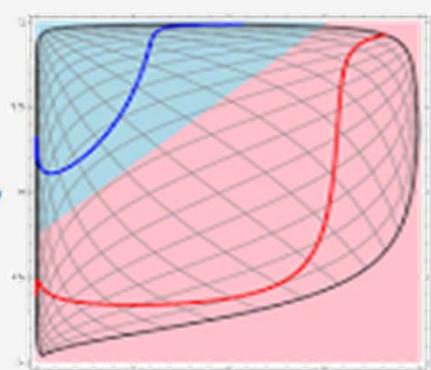
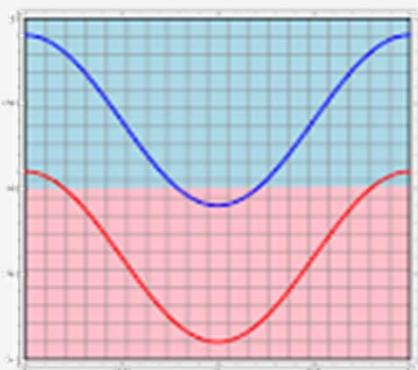
# 선형 분리 문제

## 선형 분리로 해결이 불가능한 문제



## 해결책

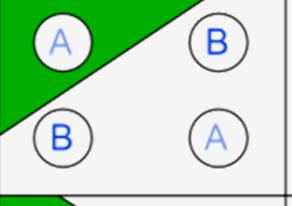
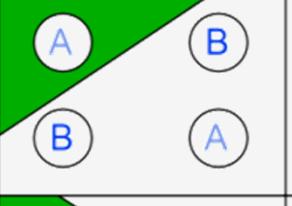
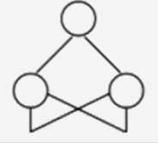
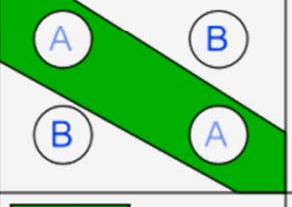
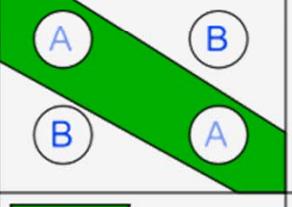
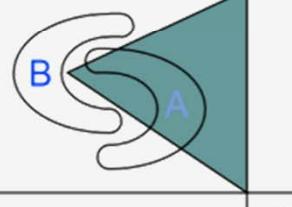
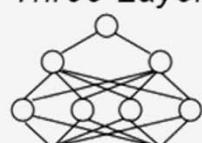
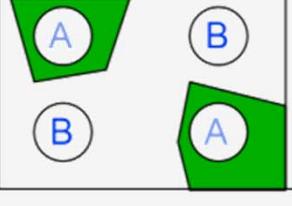
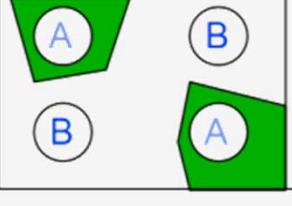
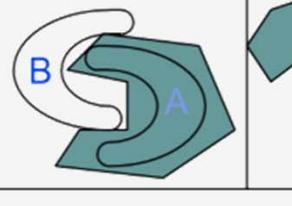
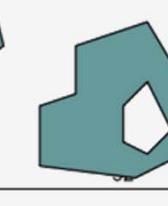
- 입력 차원을 늘린다.(예: 사과 분류 문제)
- 입력을 비선형으로 변환
- 혹은 MLP



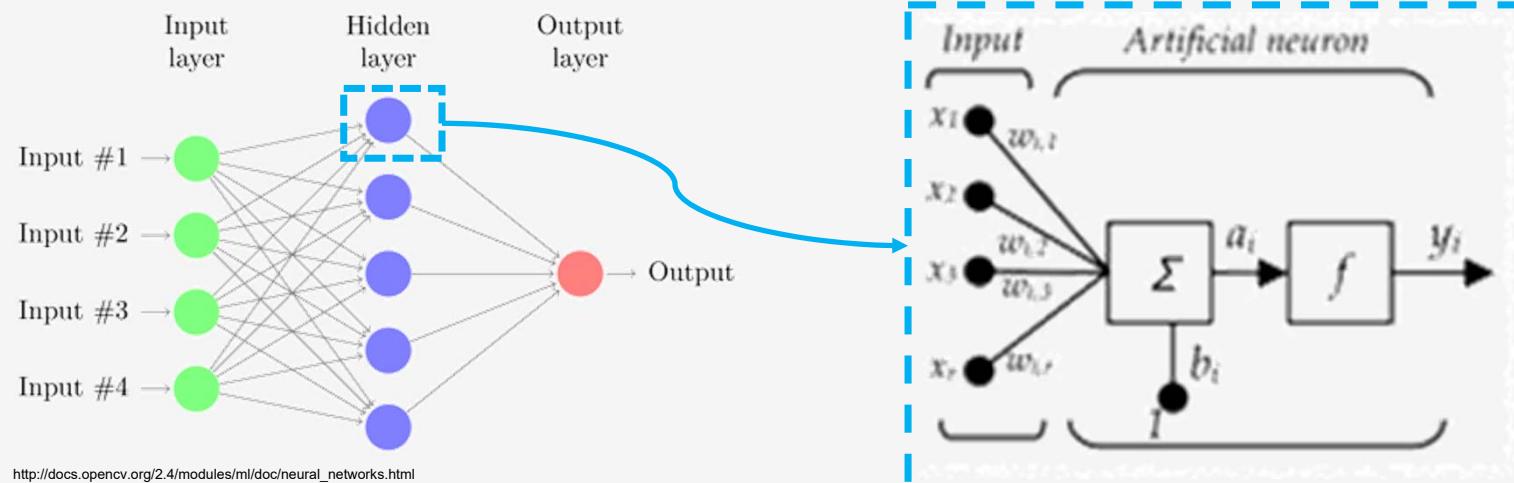
[http://sebastianraschka.com/Articles/2014\\_naive\\_bayes\\_1.html](http://sebastianraschka.com/Articles/2014_naive_bayes_1.html)

# 퍼셉트론의 능력

- 1개의 퍼셉트론은 1개의 선형 분리를 할 수 있다.
- 퍼셉트론의 결과를 다른 퍼셉트론의 입력으로 하면 여러개의 선으로 분리를 할 수 있다.

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane 			
Two-Layer 	Convex Open Or Closed Regions 			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes) 			

# MLP(Multi Layer Perceptron)

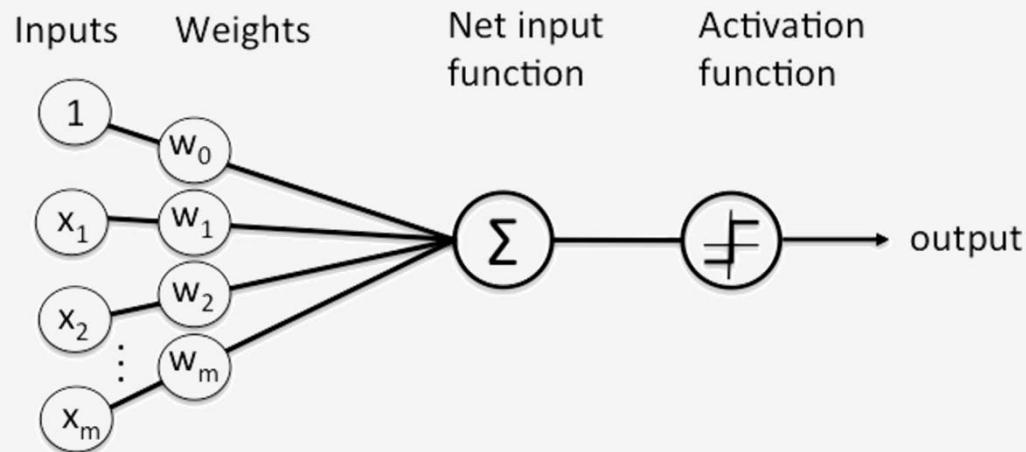


입력과 출력 사이에 층이 더 있다.

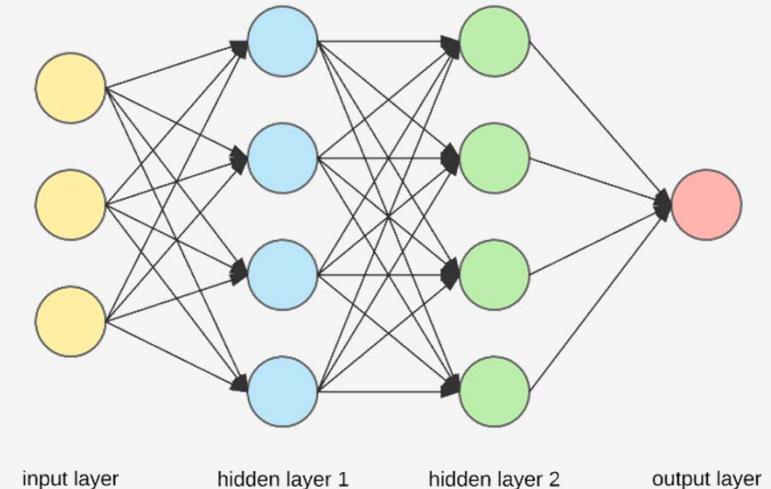
개별 perceptron의 결과를 다음 층의 입력으로 사용하고 결과적으로 선형 분리의 제약을 극복

# Node, Layer, Neural Network, and Model

## Node



## Multi Layer Perceptron

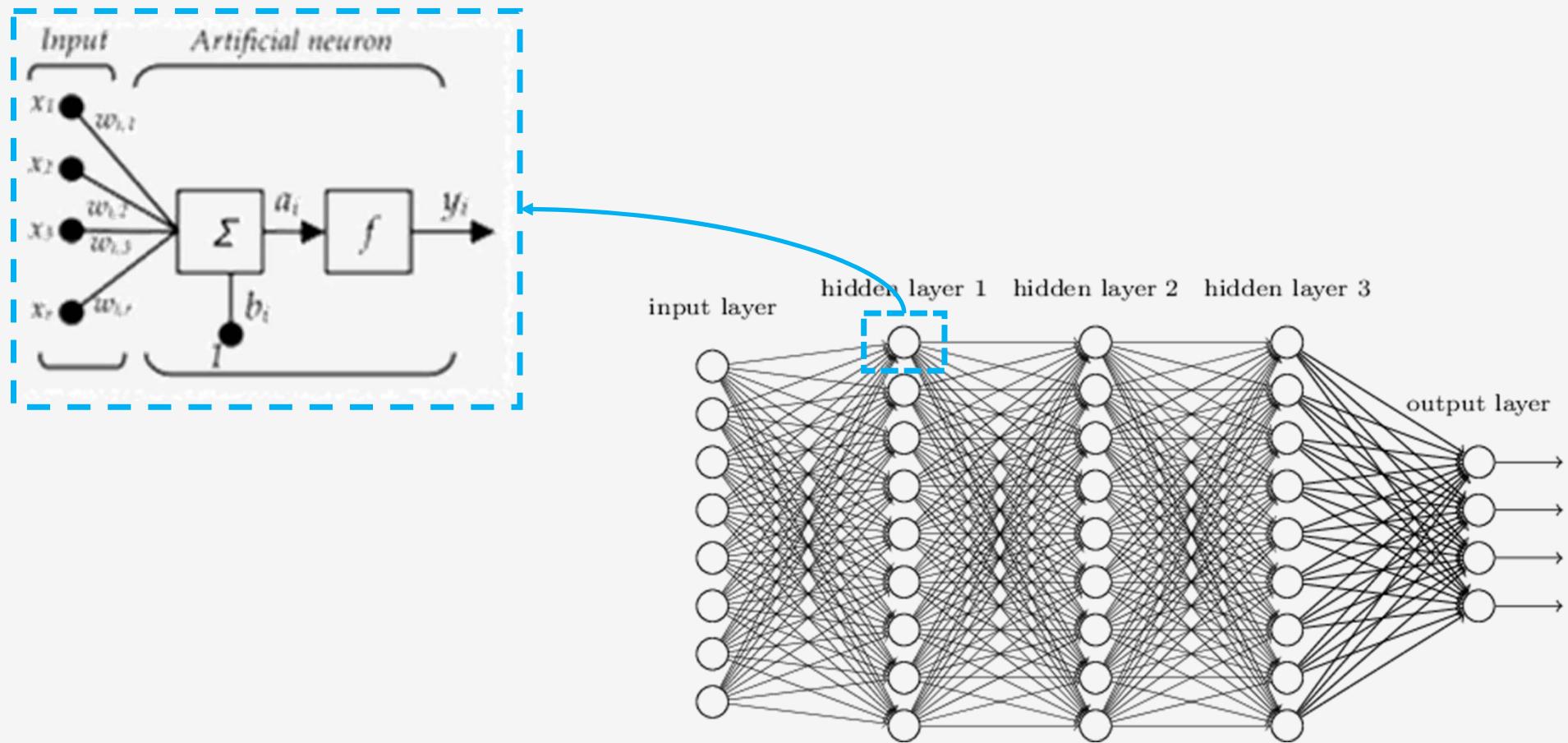


입력 값들은 가중치에 의해 출력이 결정됨

학습: 원하는 출력이 되도록 **가중치를 조절**하는 과정

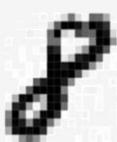
학습의 결과: 모델에 맞게 잘 조정된 가중치

# Deep Neural Network

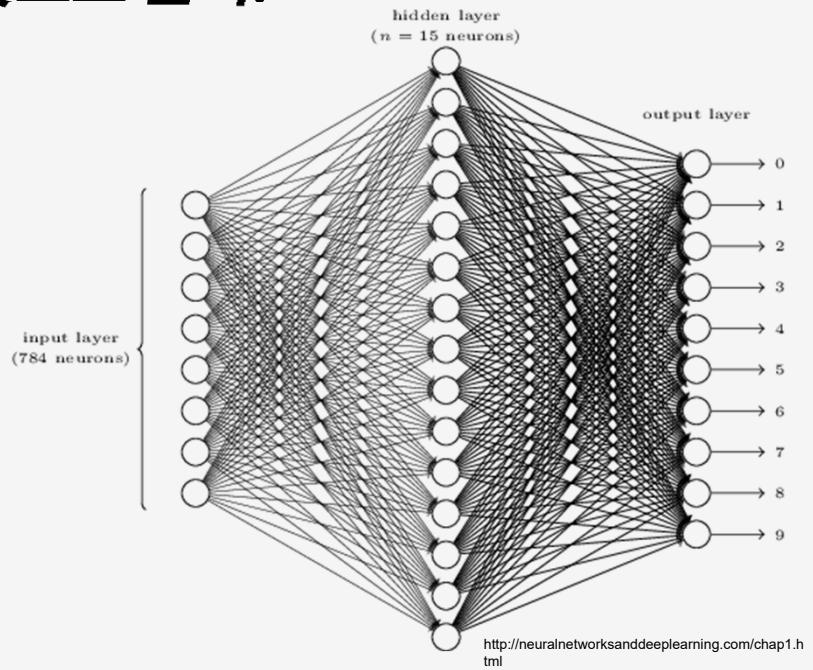


# 숫자 인식의 예

- 이미지를 구성하는 pixel의 각 값으로 구성된 입력벡터를 NN의 입력으로 한다.
  - 그리고 학습시 해당 출력 노드만 1로 하고, 나머지는 0으로 학습.
  - test 시에는 출력 노드중 최대 값을 가진 것으로 인식.



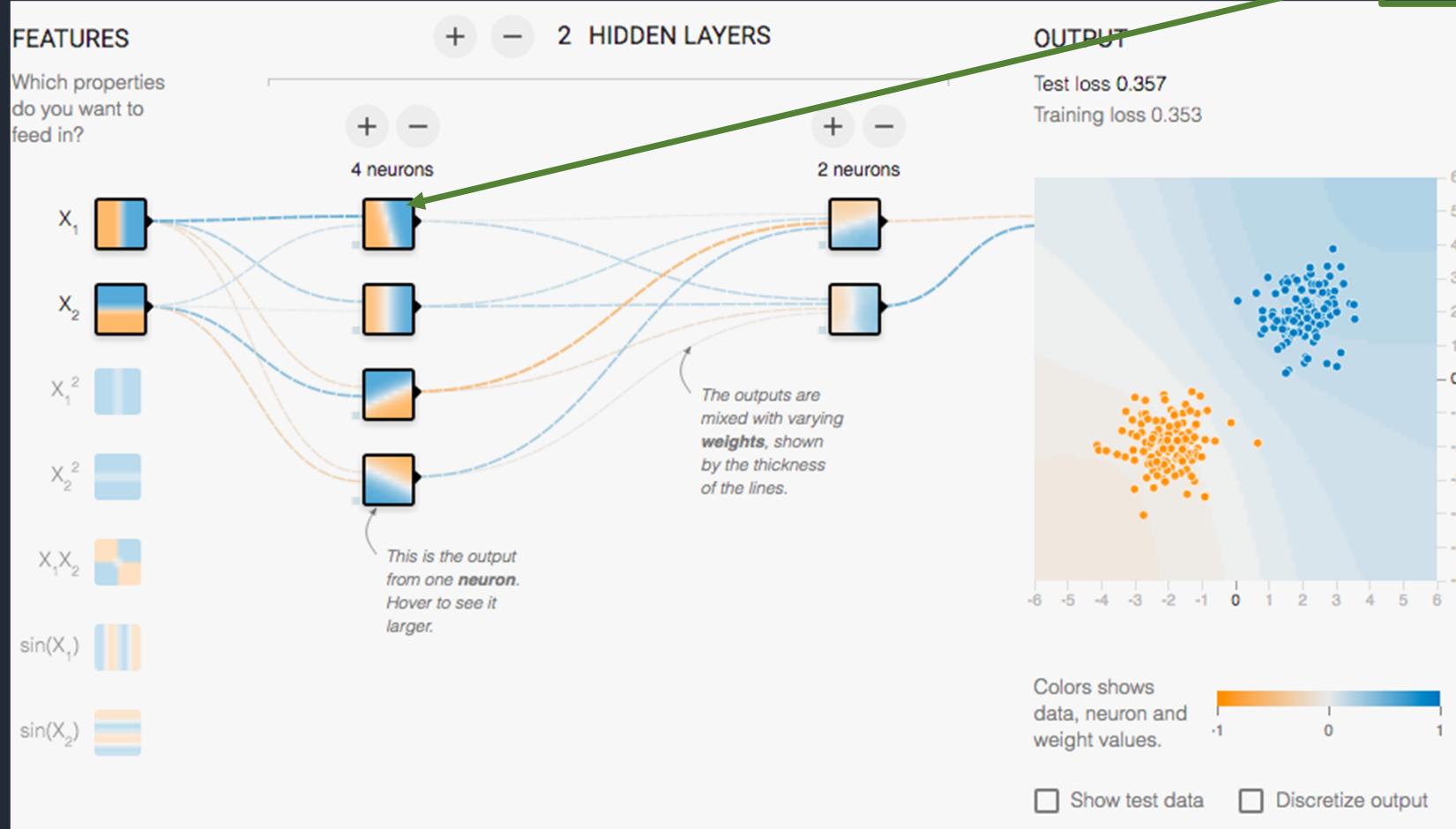
28\*28 image



# 온라인 시뮬레이션

<http://playground.tensorflow.org>

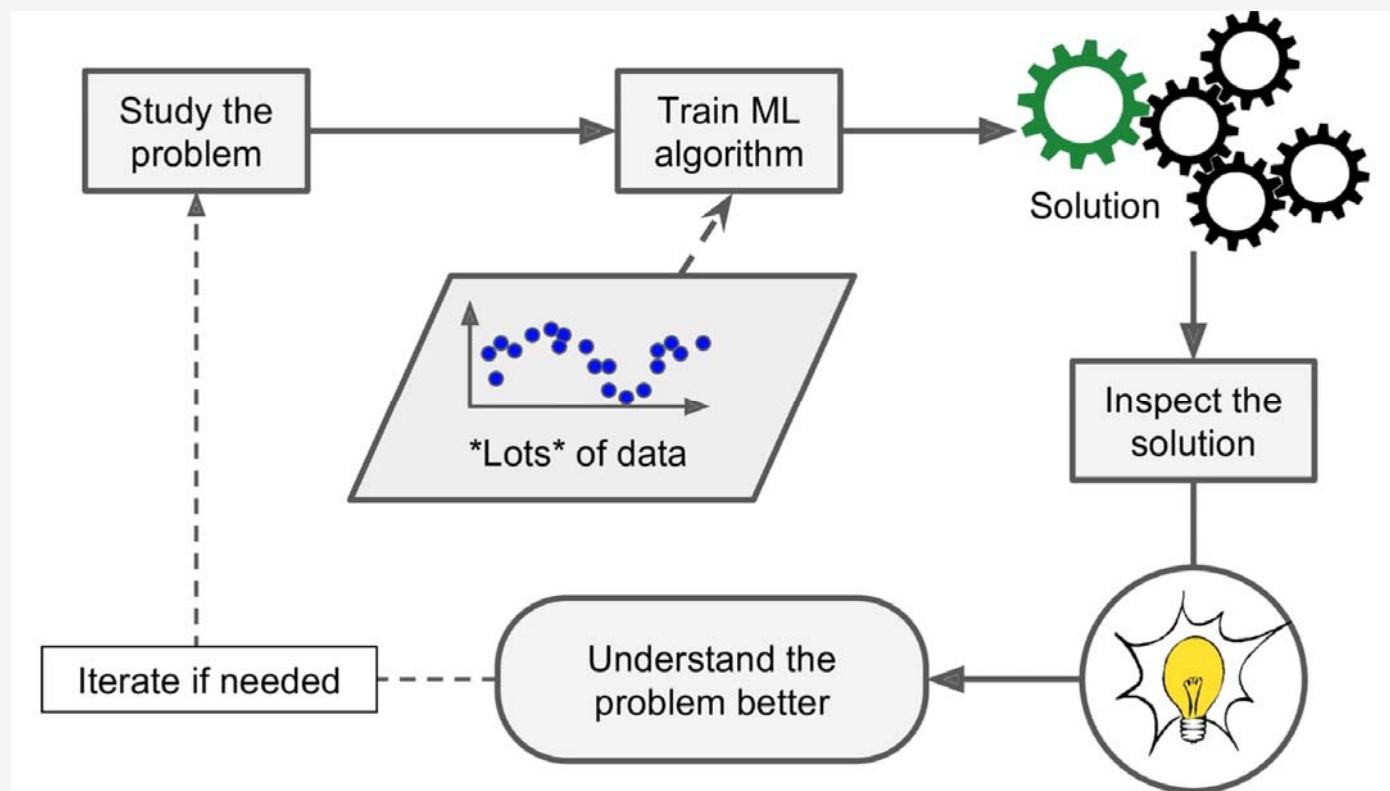
네모 하나하나가 퍼셉트론



# Deep Learning

## Deep Learning의 Breakthrough:

- Geoffrey Hinton (2005): 역전파 알고리즘 (Back propagation)
- Andrew Ng (2012)



하용호@kakao

# 세 가지 문제가 있었다.

덜하거나

**Underfitting** 학습이 잘 안돼!

느리거나

**Slow** 도대체 학습은 언제 끝나는 건가?

과하거나

**Overfitting** 겨우 되어도 융통성이 없다?!

# DNN의 문제점 I - Underfitting

덜하거나  
**Underfitting** 학습이 잘 안돼!

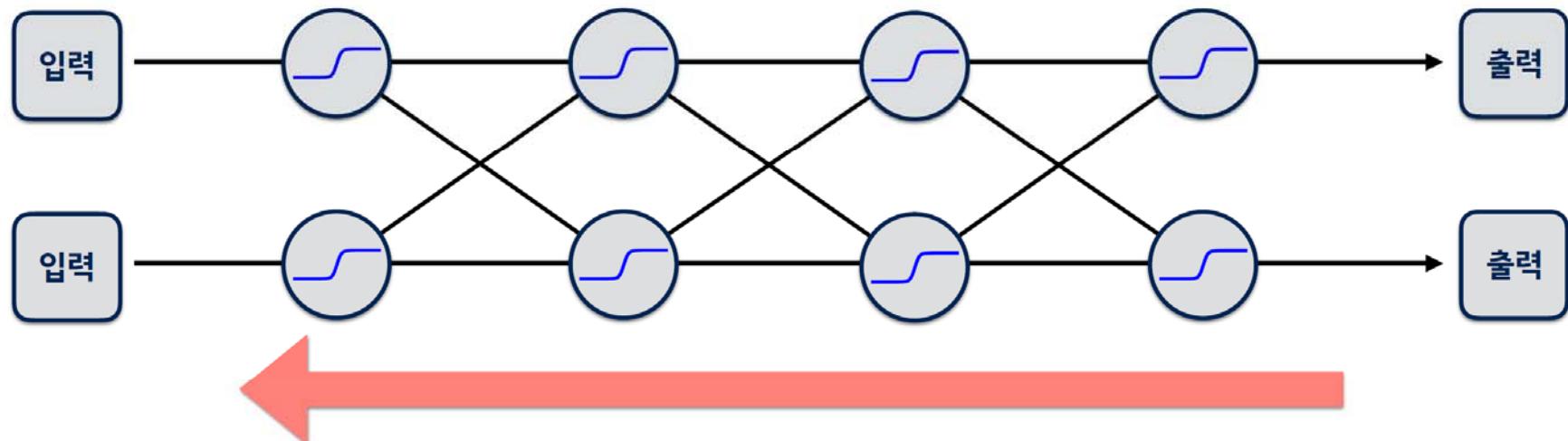
느리거나  
**Slow** 도대체 학습은 언제 끝나는 건가?

과하거나  
**Overfitting** 겨우 되어도 융통성이 없다?!

# 뉴럴넷의 학습방법 Back propagation

(사실 별거 없고 그냥 “뒤로 전달”)

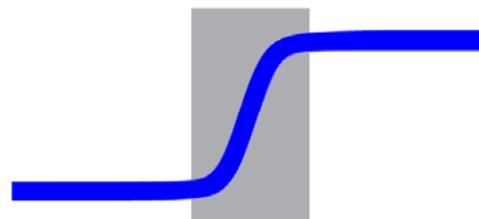
뭐를 전달하는가?  
현재 내가 틀린정도를 ‘미분(**기울기**)’ 한 거



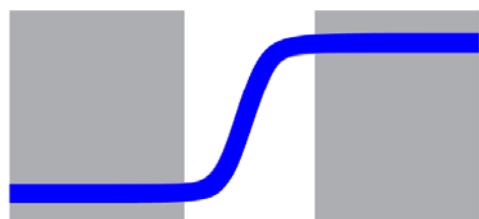
미분하고, **곱하고**, 더하고를 역방향으로 반복하며 업데이트한다.

# 근데 문제는?

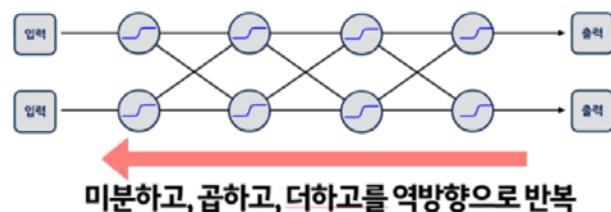
우리가 activation 함수로 sigmoid  를 썼다는 것



여기의 미분(기울기)는 뭐라도 있다. 다행



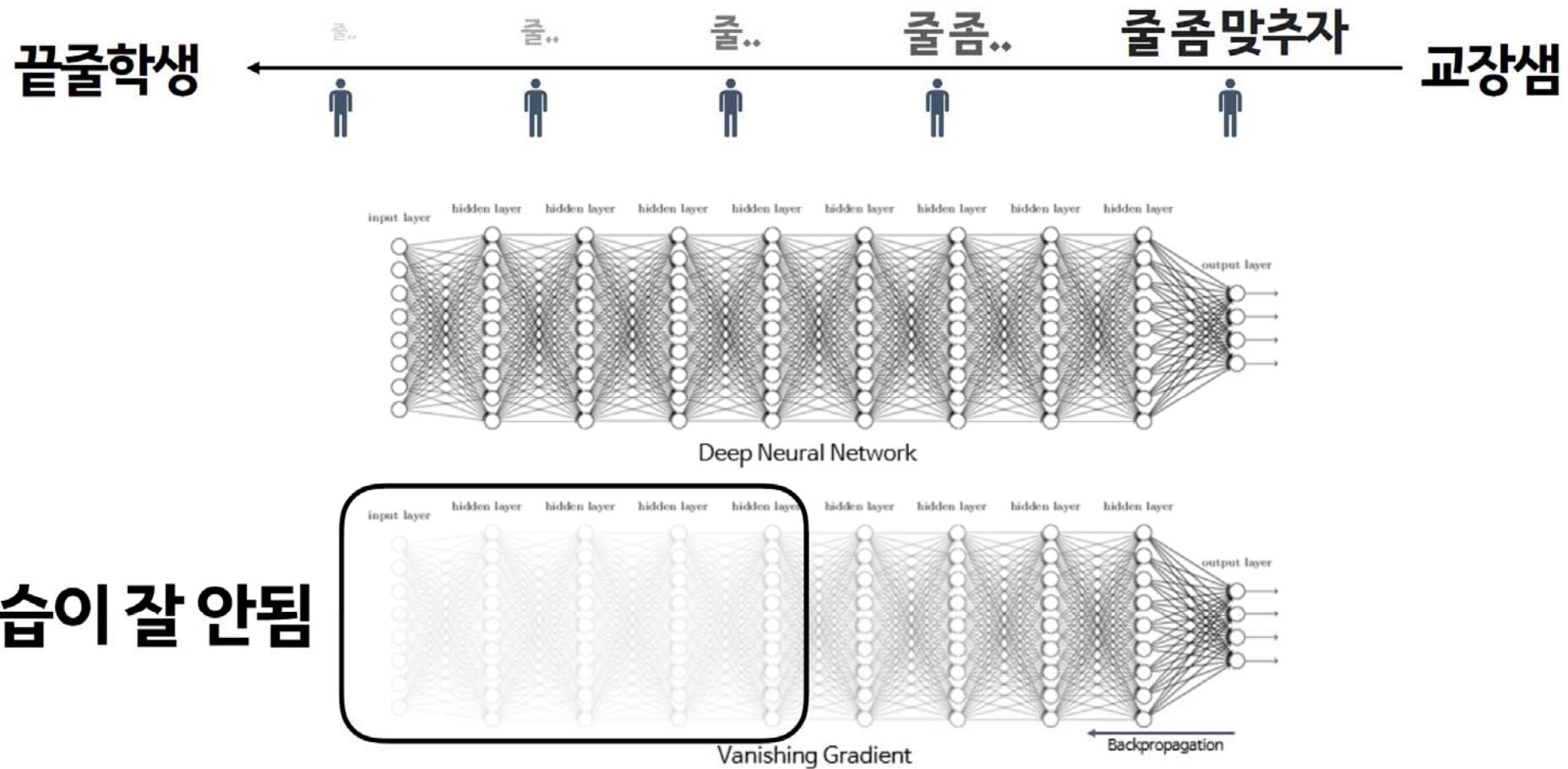
근데 여기는 기울기 0.. 이런거 중간에 곱하면 뭔가 뒤로 전달할게 없다?!



그런 상황에서 이걸 반복하면??????

미분하고, 곱하고, 더하고를 역방향으로 반복

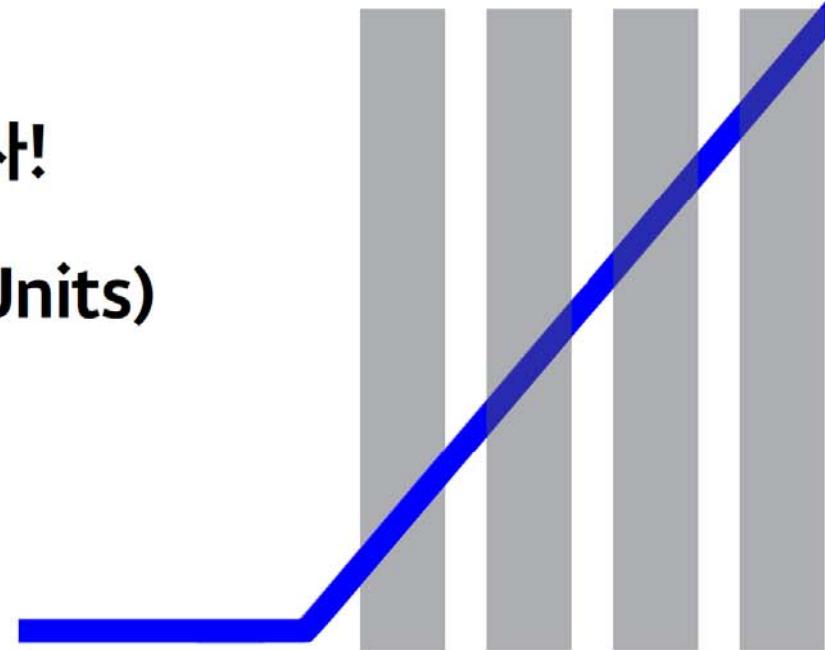
# Vanishing gradient 현상 : 레이어가 깊을 수록 업데이트가 사라져간다. 그래서 fitting이 잘 안됨(underfitting)



하용호@kakao

사그라드는 sigmoid 대신  
죽지 않는 activation func을 쓰자!

→ **ReLU (Rectified Linear Units)**



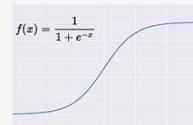
이 녀석은 양의 구간에서 전부 미분 값(1)이 있다!



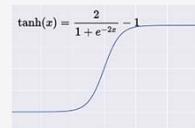
끌줄 학생까지 이야기가 전달이 잘 되고 위치를 고친다!

# 활성화 함수 종류

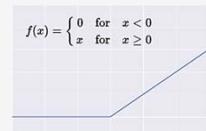
- sigmoid (= logistics)



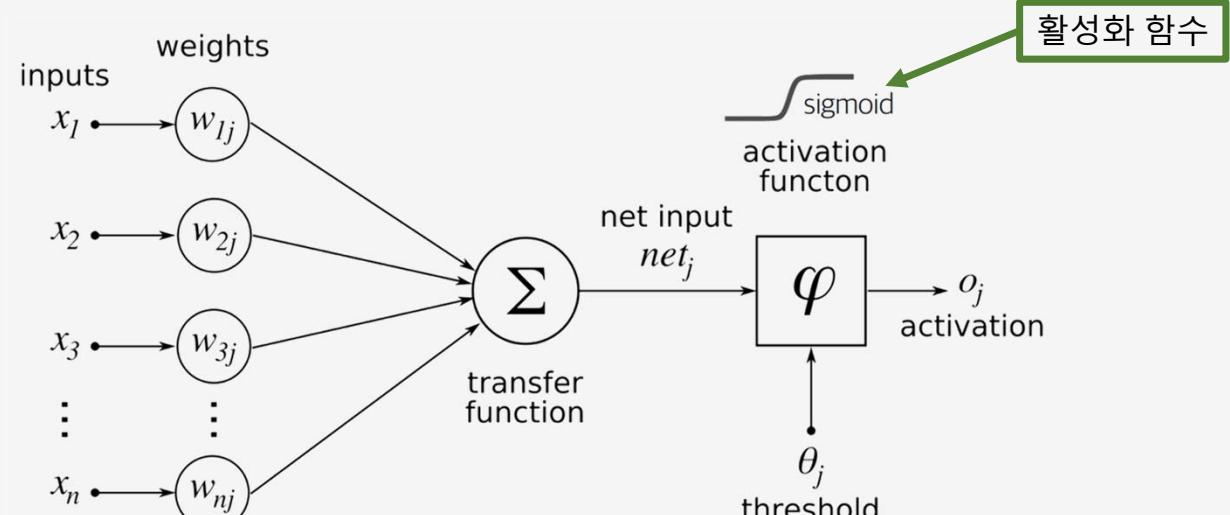
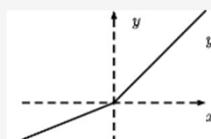
- Tanh



- ReLU



- Leaky ReLU



# SoftMax

activation function 중의 하나.

최종 출력층에 사용되며, 여러 개의 출력 노드의 합 중 비중의 값으로 나타내고 확률처럼 표현된다

$$\begin{bmatrix} w_{A1} & w_{A2} & w_{A3} \\ w_{B1} & w_{B2} & w_{B3} \\ w_{C1} & w_{C2} & w_{C3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} w_{A1}x_1 + w_{A2}x_2 + w_{A3}x_3 \\ w_{B1}x_1 + w_{B2}x_2 + w_{B3}x_3 \\ w_{C1}x_1 + w_{C2}x_2 + w_{C3}x_3 \end{bmatrix} = \begin{bmatrix} \bar{y}_A \\ \bar{y}_B \\ \bar{y}_C \end{bmatrix} \begin{bmatrix} 0.1 \\ 2.0 \\ 1.0 \\ 0.1 \end{bmatrix}$$

$x \rightarrow \boxed{-}$

Cost function

CROSS-ENTROPY

$$S(Y) = \bar{Y}$$

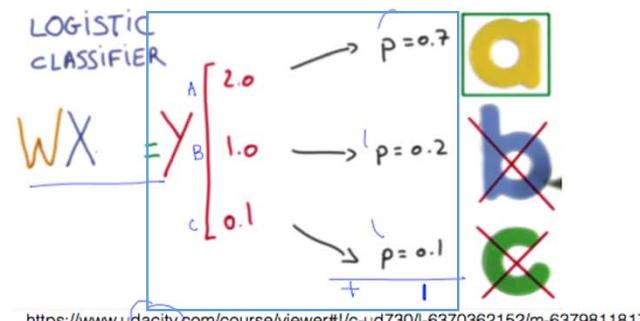
$$D(S, L) = - \sum_i L_i \log(S_i)$$

0.7	
0.2	
0.1	

1.0	
0.0	
0.0	

$L = \bar{Y}$

<https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817>



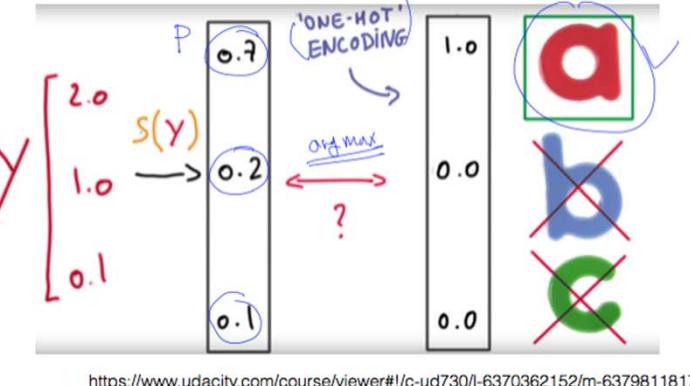
SOFTMAX

$$Y \left[ \begin{array}{c} 2.0 \\ 1.0 \\ 0.1 \end{array} \right] \xrightarrow{\text{Softmax}} \left[ \begin{array}{c} 0.7 \\ 0.2 \\ 0.1 \end{array} \right]$$

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

SCORES → PROBABILITIES

<https://www.udacity.com/course/viewer#!/c-ud730/l-6370362152/m-6379811817>



<https://pythonkim.tistory.com/20?category=573319>

문제?

전달하다가, 사그라져 버린다(vanishing gradient)

해결!

**sigmoid -> ReLU = 뒤로전달 오케이!**

# DNN의 문제점 II - Slow Learning

덜하거나

**Underfitting** 학습이 잘 안돼!

느리거나

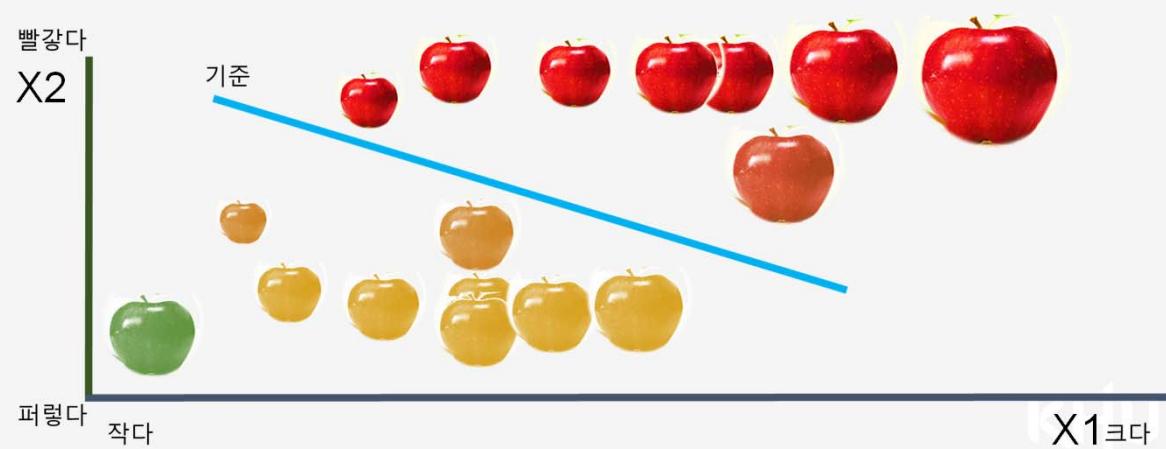
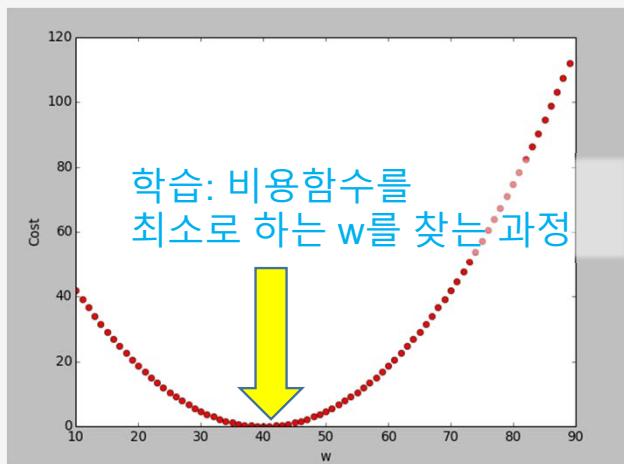
**Slow** 도대체 학습은 언제 끝나는 건가?

과하거나

**Overfitting** 겨우 되어도 융통성이 없다?!

# 비용 함수(Cost Function)와 학습

- 사과 분류의 예에서, 기준의 값에 따라 오차의 크기가 결정된다
- 결국 학습의 목표는 오차를 최소로, 혹은 오차함수의 값이 최소가 되는  $w$ 를 찾는 것
- 비용 함수는 모델을 구성하는 가중치( $w$ )의 함수이다
- 예에서의 기준값에 의해 발생하는 오차의 정도



# Cost Function의 종류

비용함수: 원래의 값과 오차가 가정 적은  $\theta$  값을 구하여 가설함수  $h$ 를 정하는데 사용되는 함수

## MSE (Mean Squared Error)

- 가장 보편적으로 사용

$$\bullet \quad MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

$\hat{Y}_i$  = predicted value

## CE (Cross Entropy)

- Entropy는 불확실성의 척도 (불확실성: 어떤 data가 나올지 예측하기 어려운 경우)
- Entropy가 높으면 정보가 많고 확률이 낮다는 것을 의미함
- 동전의 엔트로피: 약 0.693, 주사위의 엔트로피: 약 1.79 (주사위의 경우 무엇이 나올지 예측하기 더 어렵다.)

## KL-Divergence (Kullback - Leibler Divergence: 컬백-라이블러 발산)

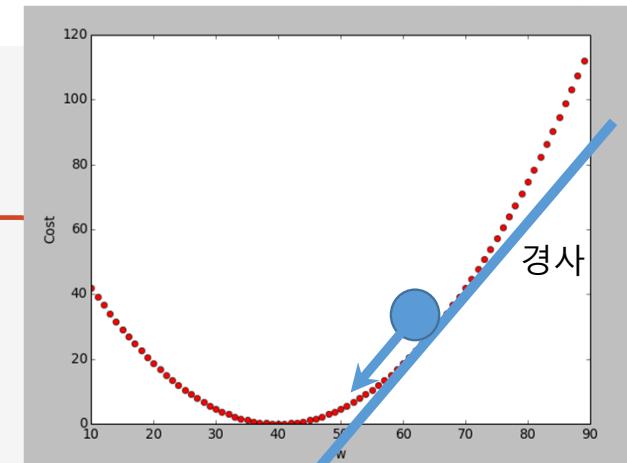
- 두 확률 분포의 차이를 계산하는 데에 사용하는 함수
- 어떤 이상적인 분포에 대해 그 분포를 근사하는 다른 분포를 사용해 샘플링을 한다면 발생할 수 있는 정보 엔트로피의 차이
- 상대 엔트로피, 정보 획득량, 인포메이션 다이버전스라고도 함

## MLE (Maximum Likelihood Estimation)

- 최대 가능성 방법/최대 우도법: 어떤 확률 변수에서 포집한 값들을 토대로 그 확률변수의 모수를 구하는 방법

# 경사 하강법(Gradient Descent)

함수가 학습될 바를 정의한 비용함수의 값이 최소로 되도록 가중치를 업데이트하기 위한 알고리즘



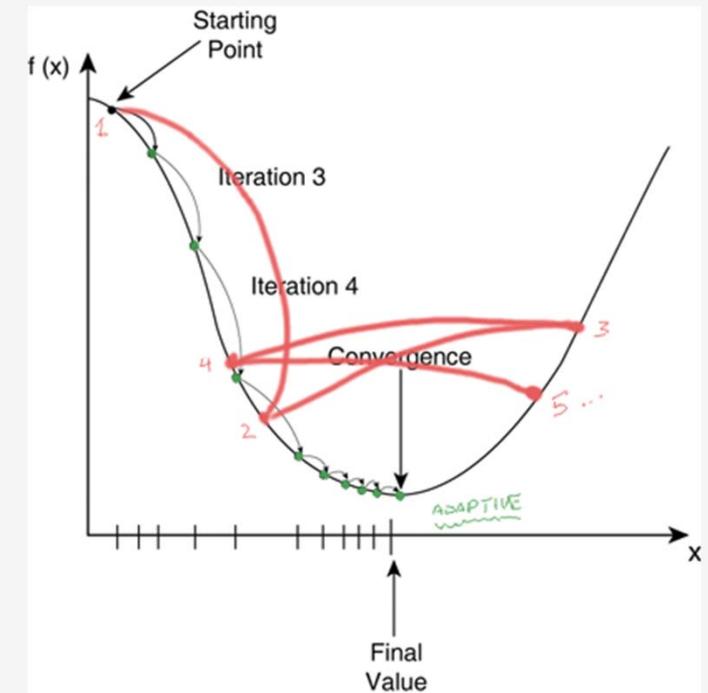
오차평면에서 공을 올려 놓았을 때 공이 굴러가는 방향(오차가 적어지는 방향)으로  $w$ 를 조정한다.

사과 분류의 예에서의 학습방법:

- 경사하강법: 비용함수를 구하고, 이를 각 weight 별로 편미분하여 각 weight 별로 수정할 다음 값을 구하는 방법
- 선형 모델( $w_1x_1 + w_2x_2 = b$ )에서는 경사하강법의 결과가 사과 분류의 예를 들어 직관적으로 설명했던 방법과 동일
- 사과 분류의 예에서 학습한 방법은 경사하강법을 사용한 것

# 학습율

## 가중치가 변경되는 정도



<https://www.quora.com/In-an-artificial-neural-network-algorithm-what-happens-if-my-learning-rate-is-wrong-too-high-or-too-low>

# Back Propagation

---

출력된 값과 원하는 값과의 차이를 가지고 그 전의  $w$  값을 변경하는 알고리즘.

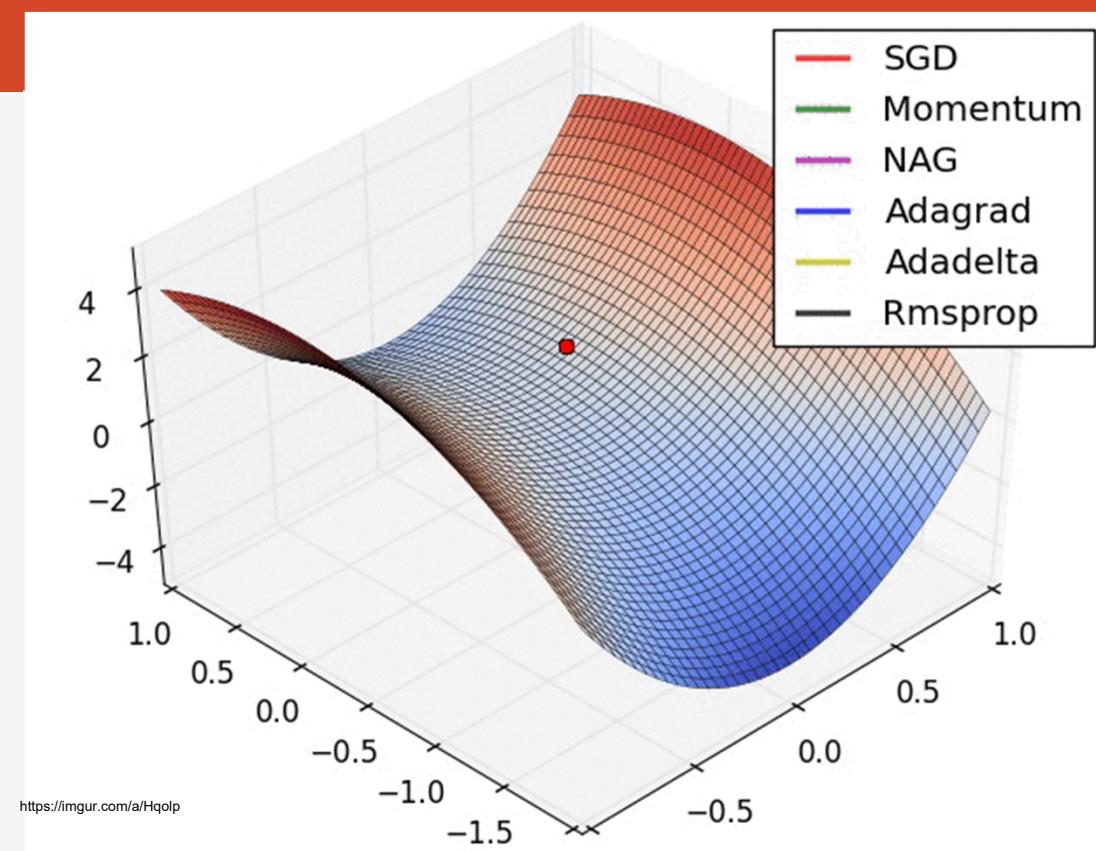
뒤에서부터 그 오차의 값이 전파된다는 이름.

실제 변경되는 값의 크기는 GD로 결정됨.

# Optimizer

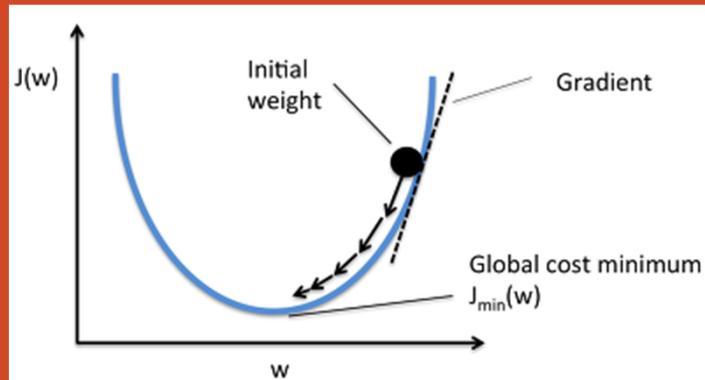
- 오차에 대하여 weight을 업데이트 시키는 알고리즘

- GD (Gradient Descent)
- Batch GD
- Mini-batch GD
- SGD (Stochastic GD)
- Momentum
- AdaGrad
- AdaDelta
- Adam
- RMSprop



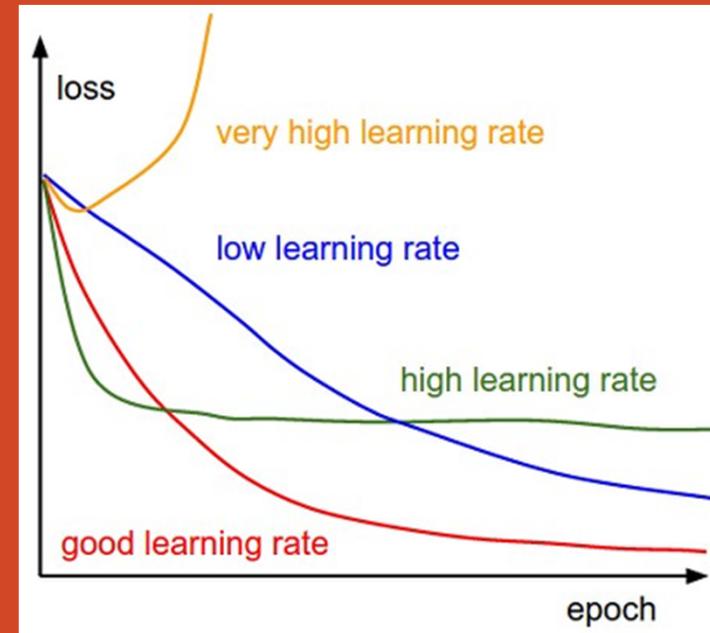
# Optimizer 사용의 목적

## 가장 빠르게 근사치를 찾기 위한 방법



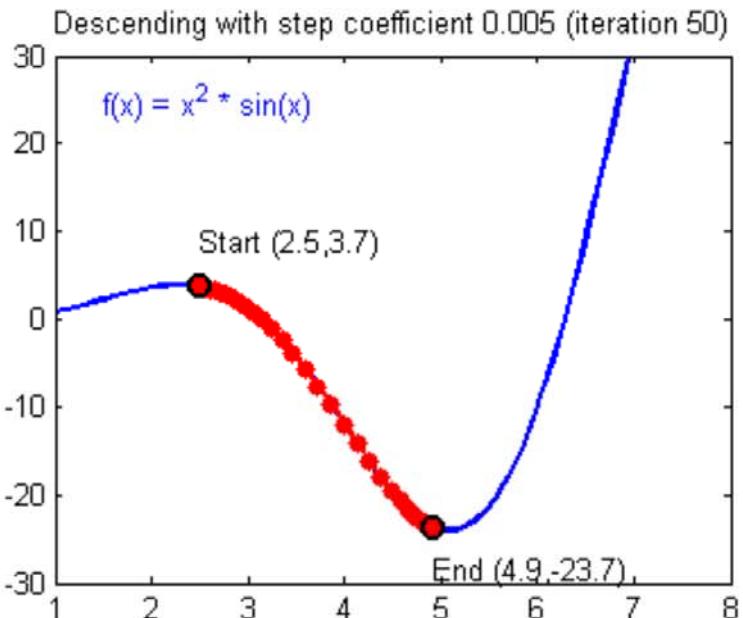
weight의 업데이트 = 에러 낮추는 방향  
(descent)  $\times$  한발자국 크기  
(learning rate)  $\times$  현 지점의 기울기  
(gradient)

$$- \gamma \nabla F(a^n)$$



### Gradient Descent – Weight Optimization Algorithm

- Loss function의 현 가중치에서의 기울기를 구해서 loss를 줄이는 방향으로 업데이트해 나가는 방법



뉴럴넷은 loss(or cost) function을 가지고 있습니다. 쉽게 말하면 “틀린 정도”

현재 가진 weight 세팅(내 자리)에서,  
내가 가진 데이터를 다 넣으면  
 전체 에러가 계산됩니다.

거기서 미분하면 에러를 줄이는 방향을 알 수 있습니다.  
 (내자리의 기울기 \* 반대방향)

그 방향으로 정해진 스텝량(learning rate)을  
 곱해서 weight을 이동시킵니다. 이걸 반복~~

weight의 업데이트 =

에러 낮추는 방향  
(decent)

$$-\gamma \nabla F(\mathbf{a}^n)$$

—

$\times$  한발자국 크기  
(learning rate)

$$\gamma$$

$\times$  현 지점의 기울기  
(gradient)

$$\nabla F(\mathbf{a}^n)$$

하용호@kakao

- Done is better than perfect -

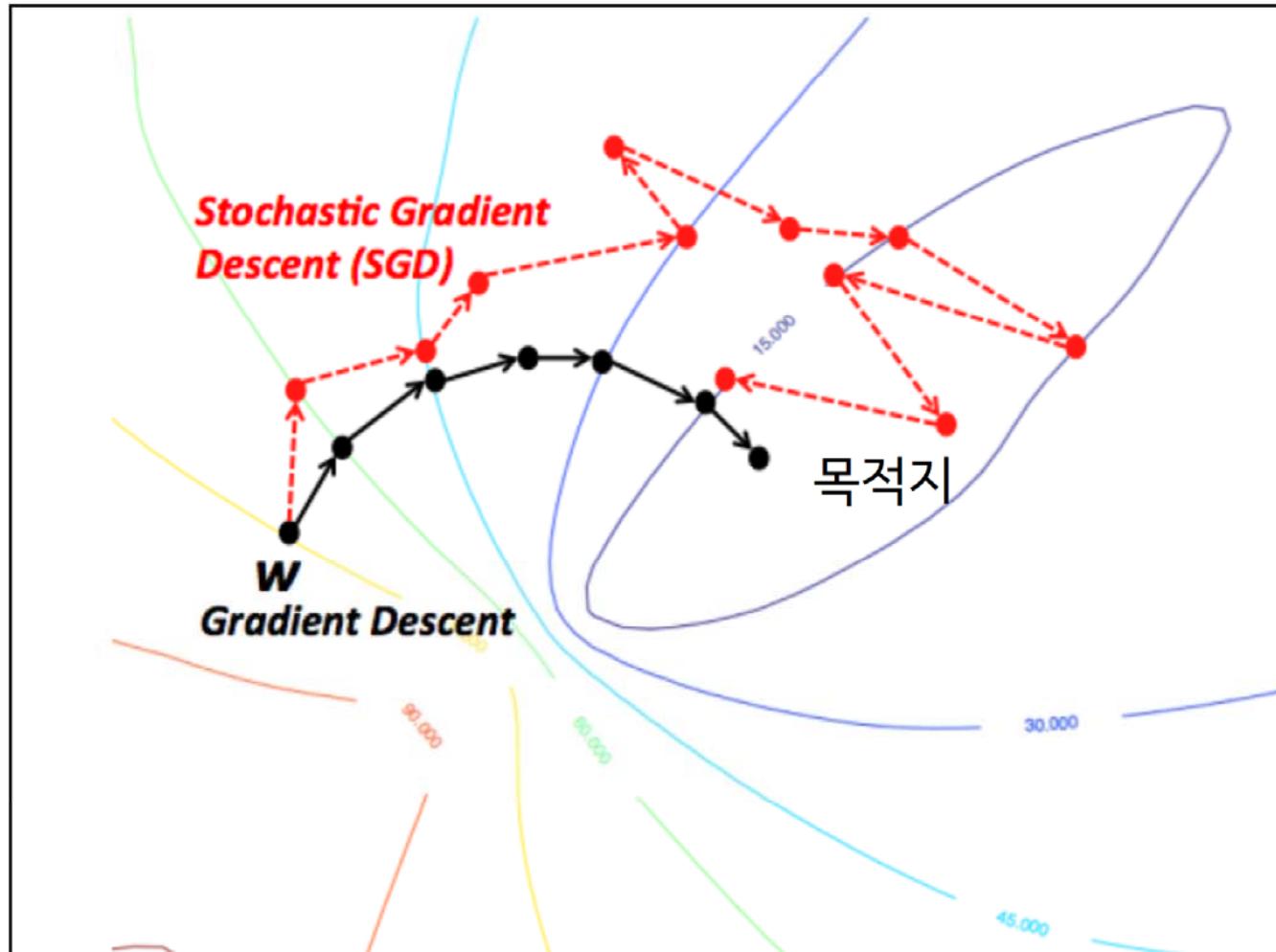


어느천년에 다하는가.  
GD보다 빠른 옵티마이저는 없을까?

# Stochastic Gradient Decent!!

조금씩 훑어보고 빨리 진행하는 방식

# GD vs SGD



## Gradient Decent

모든 걸 계산(1시간)  
후  
최적의 한스텝

6스텝 \* 1시간 = 6시간

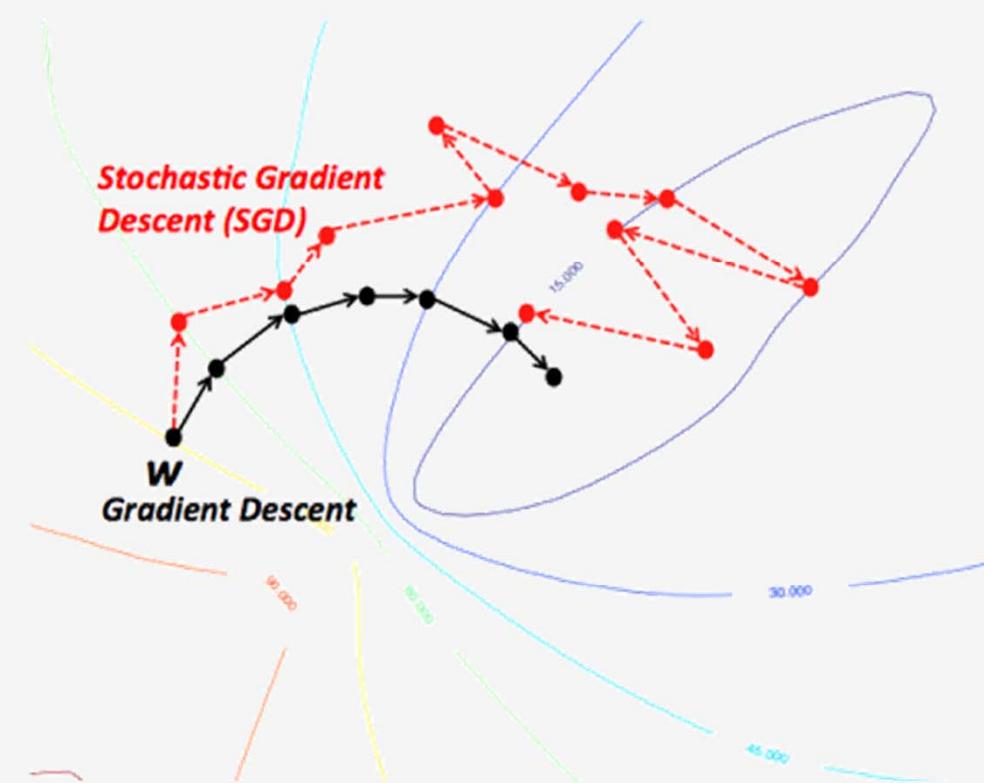
**최적인데 너무 느리다!**

## Stochastic Gradient Descent

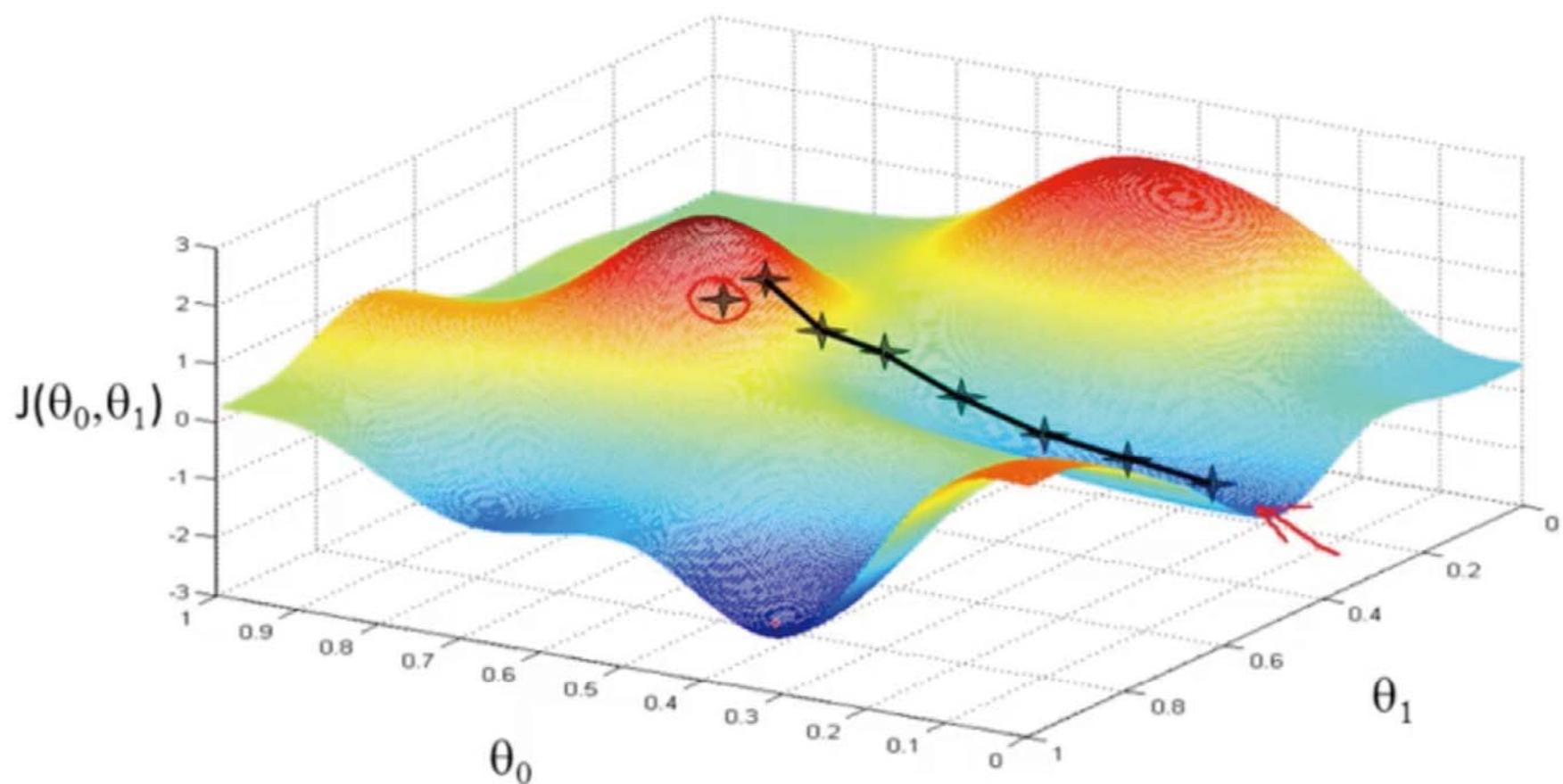
일부만 검토(5분)  
틀려도 일단 간다! 빠른 스텝!  
11스텝 \* 5분 = 55분 < 1시간

**조금 헤매도 어쨌든 인근에  
아주 빨리 갔다!**

# Gradient Decent vs Stochastic Gradient Decent

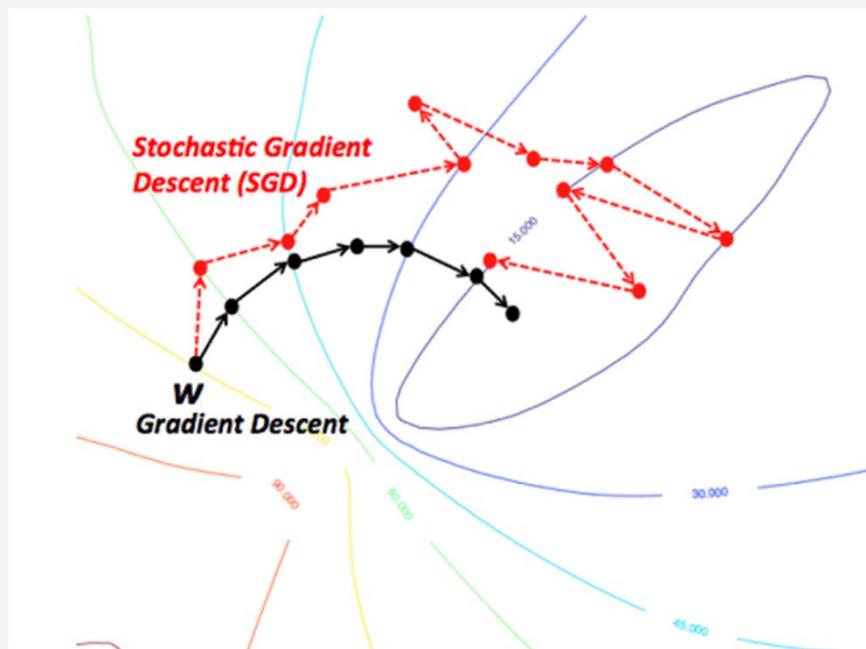
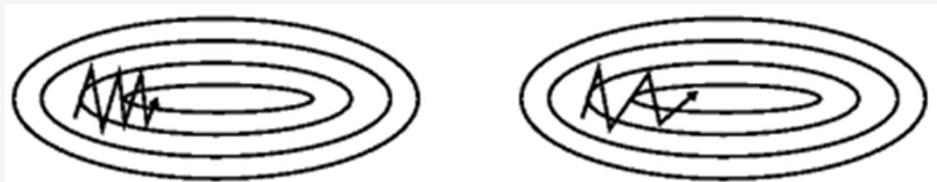


# 다시 생각해봐도 이건, 굴곡 많은 산을 좋은 오솔길을 찾아 잘 내려가는 일과 참 비슷

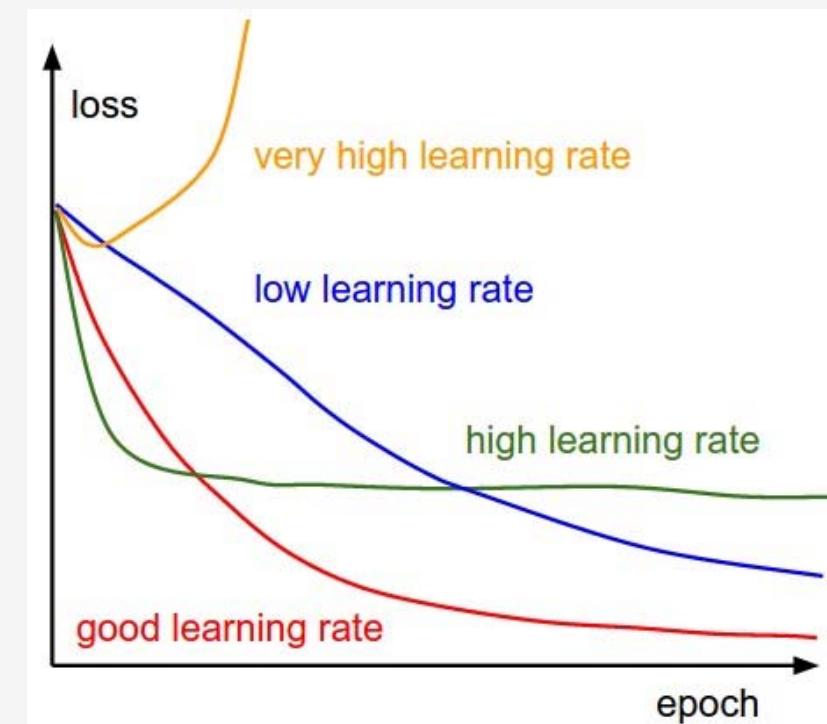


# GSD의 단점

## 방향 문제 – 빠른 반면 많이 헤멘다



## Step size 문제



<https://cs231n.github.io/neural-networks-3/>

–  $\gamma \nabla F(\mathbf{a}^n)$  산을 잘 타고 내려오는 것은  
 $\nabla F(\mathbf{a}^n)$  어느 방향으로 발을 디딜지  
 $\gamma$  얼마 보폭으로 발을 디딜지  
두 가지를 잘 잡아야 빠르게 타고 내려온다.

SGD를 더 개선한 멋진 optimizer가 많다!  
SGD의 개선된 후계자들

# 산내려오는 작은 오솔길 잘찾기(Optimizer)의 발달 계보

모든 자료를 다 검토해서  
내 위치의 산기울기를 계산해서  
갈 방향을 찾겠다.

**GD**

스텝방향

**SGD**

전부 다봐야 한걸음은  
너무 오래 걸리니까  
조금만 보고 빨리 판단한다  
같은 시간에 더 많이 간다

**Momentum**

스텝 계산해서 움직인 후,  
아까 내려 오던 관성 방향 또 가자

Nesterov Accelerated Gradient

**NAG**

일단 관성 방향 먼저 움직이고,  
움직인 자리에 스텝을 계산하니  
더 빠르더라

**Nadam**

Adam에 Momentum  
대신 NAG를 붙이자.

**Adam**

RMSProp + Momentum  
방향도 스텝사이즈도 적절하게!

**RMSProp**

보폭을 줄이는 건 좋은데  
이전 맥락 상황봐가며 하자.

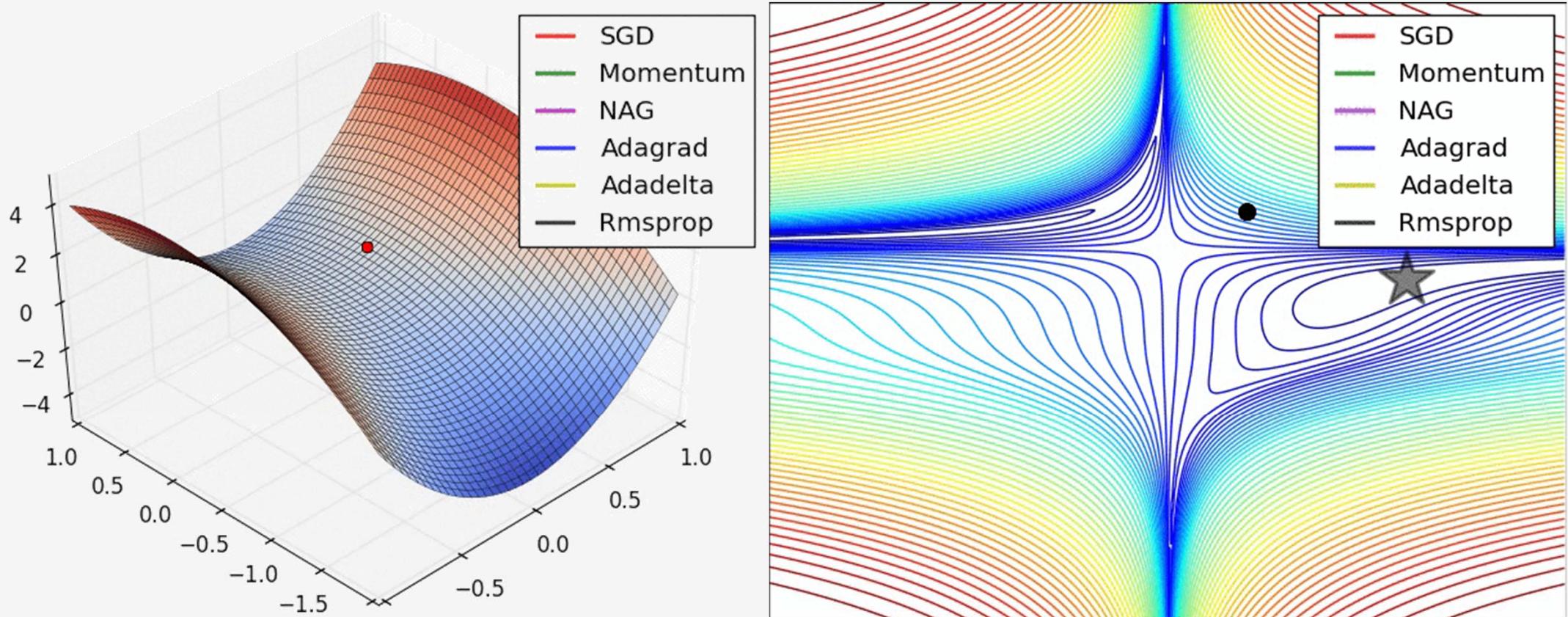
**Adagrad**

안가본 곳은 성큼 빠르게 걸어 훑고  
많이 가본 곳은 잘아니까  
갈수록 보폭을 줄여 세밀히 탐색

**AdaDelta**

종종 걸음 너무 작아져서  
정지하는 걸 막아보자.

해결!  
**SGD의 개선된 버전을 골라  
더 빠르고 더 정확하게!**



<http://imgur.com/NKsFHJb>

# Data 전처리

## 필수적 data 전처리

- 결측치
- Outlier
- Normalization/Standardization

# Normalization

- 입력이 여러 차원일 때 각 차원의 입력을 동일한 스케일로 맞춰주면 학습이 빠르게 진행됨
  - Normalization: 전체 data를 0~1 사이 값으로 변환
  - Standardization: 평균을 0, 표준편차를 1이 되도록 변환

# DNN의 문제점 III - Over Fitting

덜하거나

**Underfitting** 학습이 잘 안돼!

느리거나

**Slow** 도대체 학습은 언제 끝나는 건가?

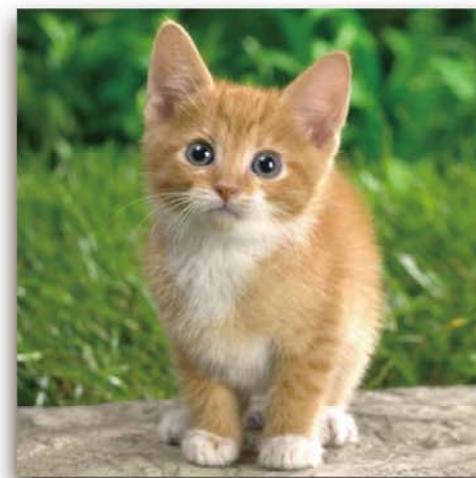
과하거나

**Overfitting** 겨우 되어도 융통성이 없다?!

열심히 뉴럴넷에게 고양이  
를 가르쳤더니..



뚱뚱하니까 고양이 아님



갈색이니까 고양이 아님

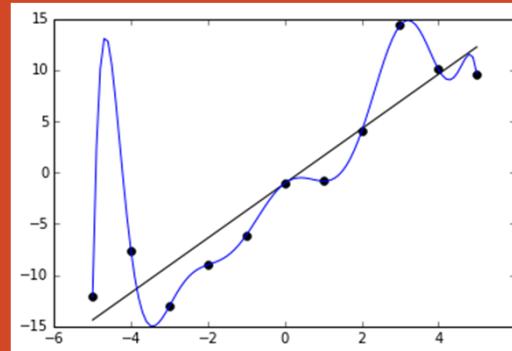
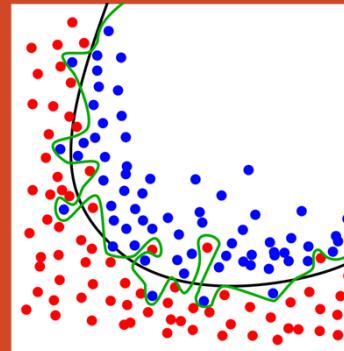


귀처졌으니까 고양이 아님

융통성이라곤 눈꼽 만큼도 없다!

**Overfitting**

# Overfitting



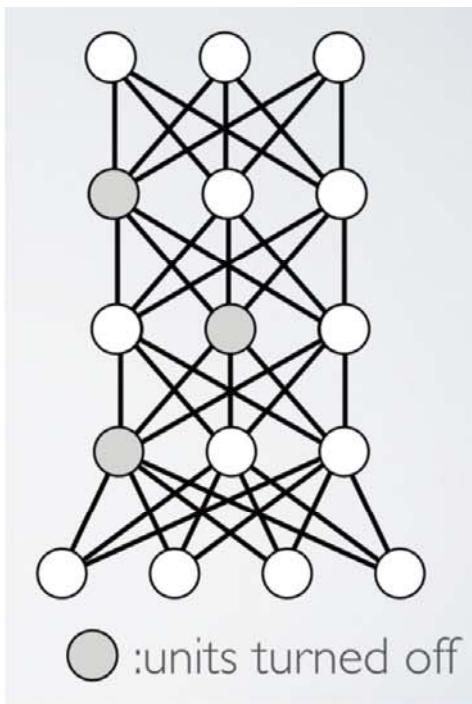
→ Overfitting된 graph:  
오히려 성능이 떨어짐

- Data에 나타나는 다양성을 심하게 받아들여 나타나는 현상
- Data가 충분치 못해 발생- data가 부족하면 일찍 발생함
- 알고리즘 상으로 완화시키는 기법으로 처리 (거의 필수적)
  - Dropout - 값의 추이를 둔화시키는 방법
  - Batch Normalization (가장 좋음) - 0-1 값으로 변형시켜주는 방법
  - Regularization
- 근본적으로 data가 적어서 나타나는 현상이므로 data augmentation으로 해결하기도 함

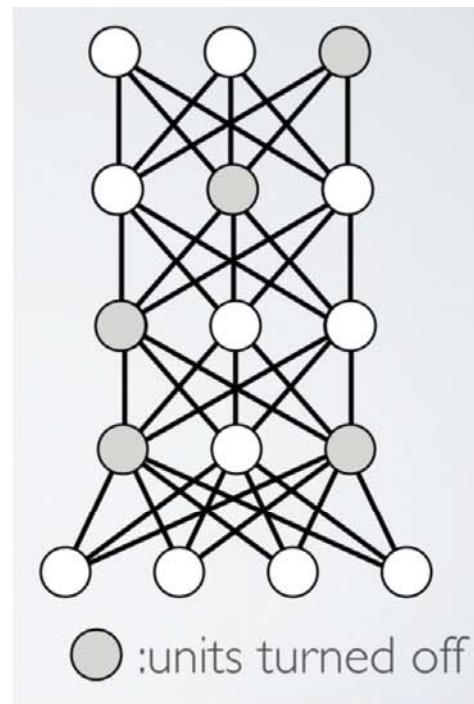
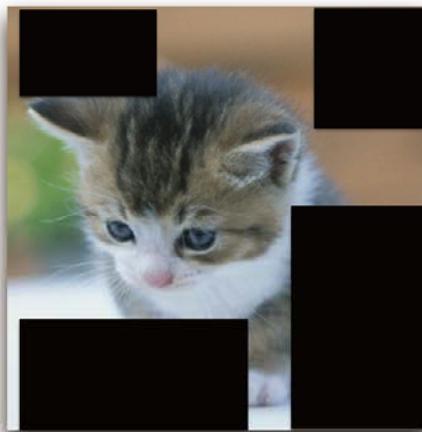
# 뉴럴넷에게 융통성을 기르는 방법은? 가르칠 때부터, 좀 가리면서 가르친다!

DropOut!

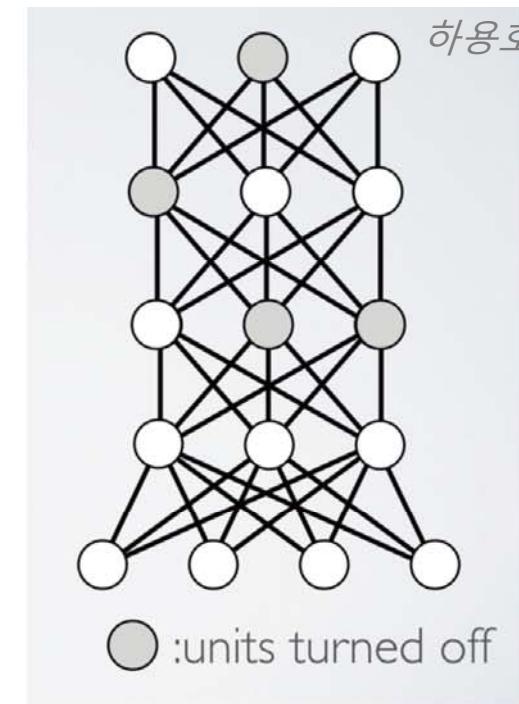
학습 시킬 때,  
일부러 정보를 누락 시키거나  
중간 중간 노드를 끈다.



얼굴위주



색지우고



귀 빼고



하용호@kakao

# DropOut Code

```
(train_x, train_y), (test_x, test_y) = get_sin_data (start=0, end=10, step=0.1)
```

```
model=keras.Sequential()  
model.add(Dense(10, activation='tanh', input_shape=(1,))  
model.add(Dropout(0.1)) #Dropout added  
model.add(Dense(10, activation='tanh'))  
model.add(Dense(1))
```

```
model.compile (optimizer='SGD', loss='mse', metrics=['mse'])  
model.summary()
```

```
model.fit(train_x, train_y, epochs=1000, verbose=0, batchsize=20)
```

```
y_=model.predict(test_x) # 예측데이터
```

**dropout으로  
일부에 집착하지 않고  
중요한 요소가 무엇인지  
터득해 나간다.**

문제?

과적합으로 융통성이 없다.

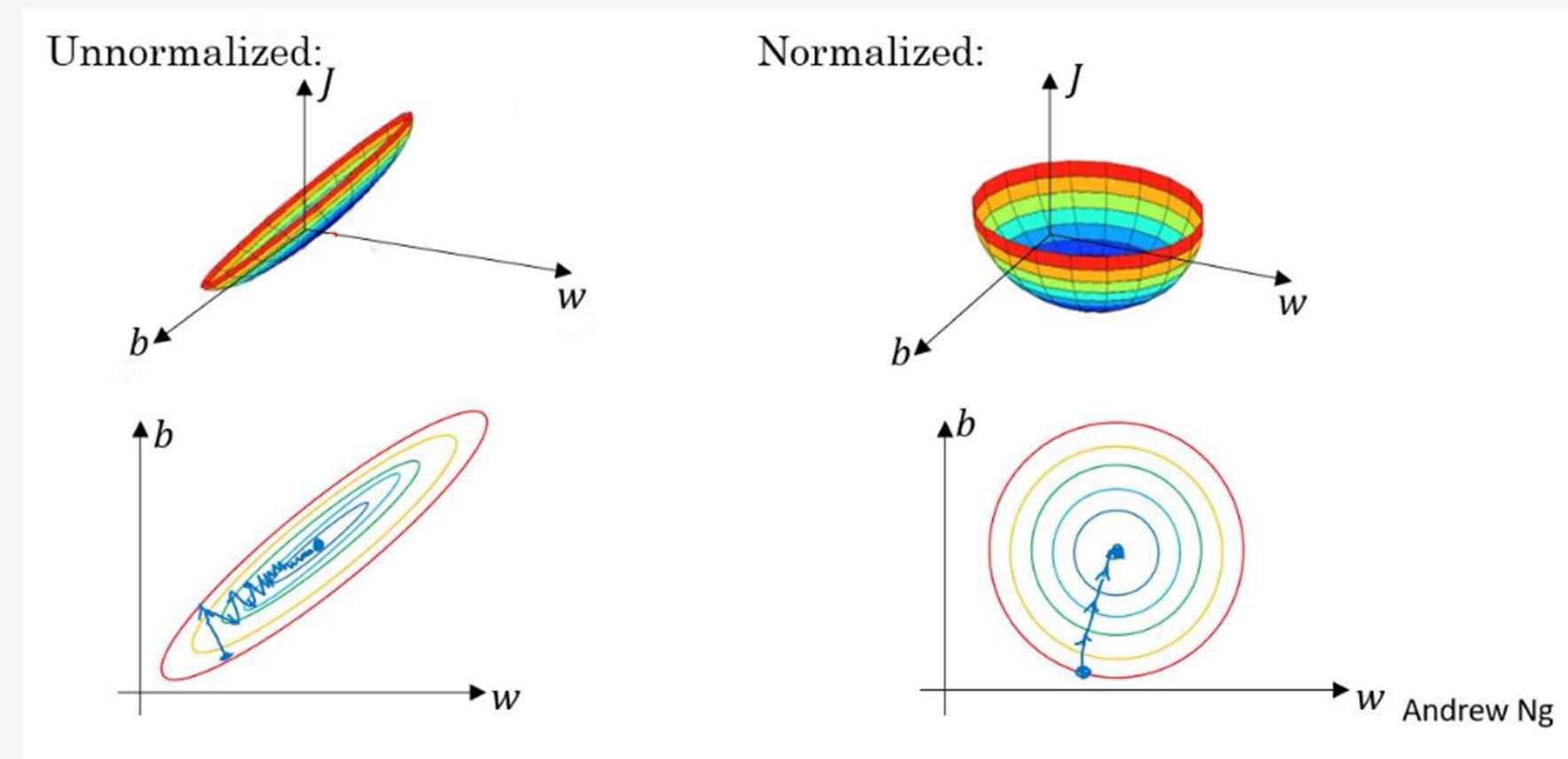


해결!

DropOut으로  
유연성을 획득시킨다.

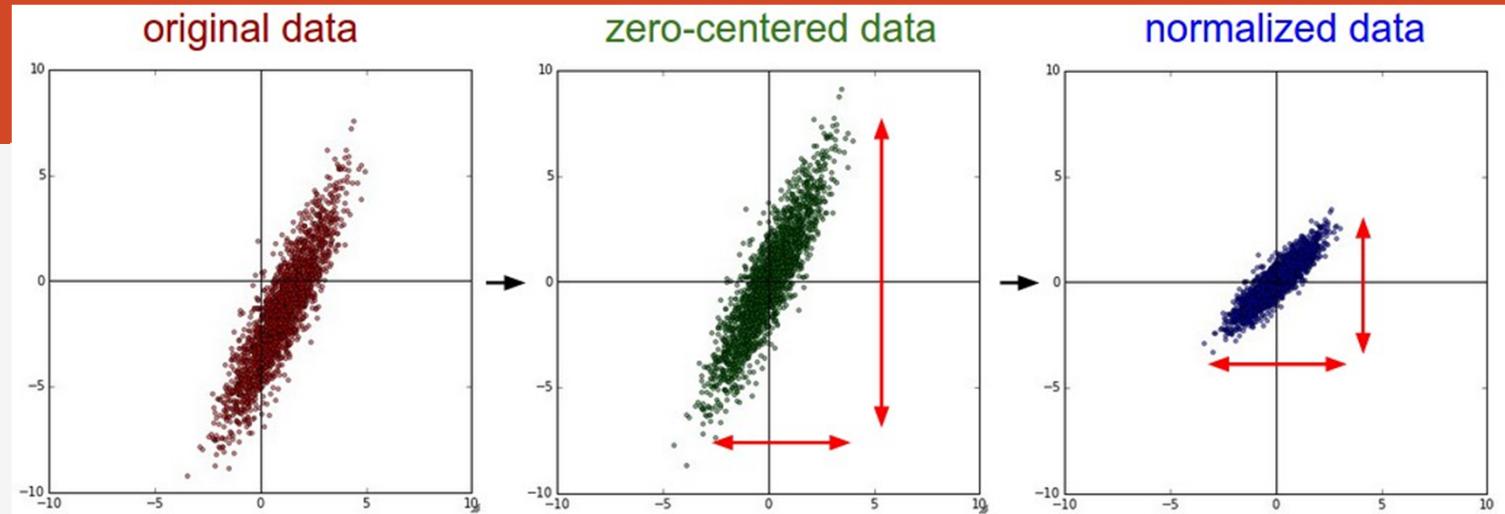
# Normalization

- 입력이 여러 차원일때 각 차원의 입력을 동일한 스케일로 맞춰주면 학습이 빠르게 진행됨
  - Normalization: 전체 데이터를 0~1 사이 값으로 변환
  - Standardization: 평균을 1, 표준편차를 10이 되도록 변환



정규화되지 않은 데이터셋은 타원모양으로 길게 늘어져 Learning Rate를 매우 **작게** 해야만 학습이 될 수 있음  
Learning rate가 충분히 작지 않으면 수평으로 이동할 때와 수직으로 이동할 때 불균형이 발생하여 Gradient Descent 알고리즘을 적용하기 어려울 수 있음  
(자료 출처: <https://gruuuuu.github.io/machine-learning/cifar10-cnn/#>)

# Standardization



- original data : 정규화 하기 전의 데이터 분포
- zero-centered data : 원 데이터에 평균을 뺌. 이로써 데이터의 분포가 가운데에 모이게 됩니다.
- normalized data : 표준편차를 나눠줌으로써 데이터의 분포가 일정해지는 효과를 얻게 됩니다. (가로 세로 길이가 같아짐)

# 문제 해결 완료!

덜하거나

**Underfitting** 학습잘되고

느리거나

**Slow** 이젠 빠르고

과하거나

**Overfitting** 융통성도 생겼다!

# 딥러닝 실제 코드 (Outline)

---

**model = build\_model() // 모델 구성**

**for i in range(1000): // 모든 데이터에 대하여 1000번 반복**

**model.fit(datas)**

**model.save( 'my\_model.h5' ) // 저장. 파일 크기는 보통 100메가 단위**

**model = load( 'my\_model.h5' ) // 저장된 모델 로딩**

**predicted = model.predict(input) // 예측 실행**

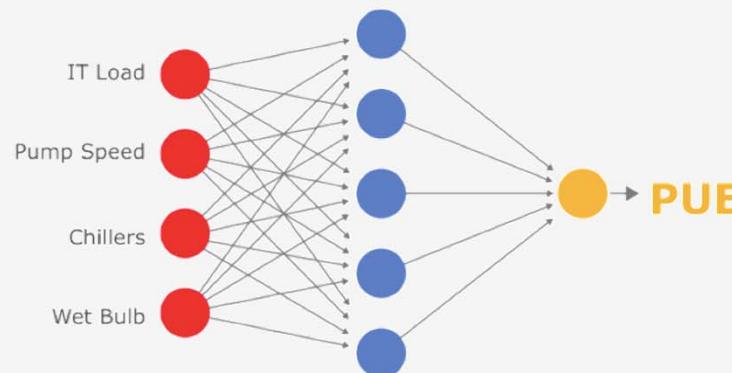
# DNN 적용의 예 - 구글 데이터 센터

## 제어값을 입력으로 하고, PUE\*를 출력으로 하여 학습 (실제 데이터 사용)

\*PUE(Power Usage Effectiveness) : 에너지 사용 효율도

- **입력**

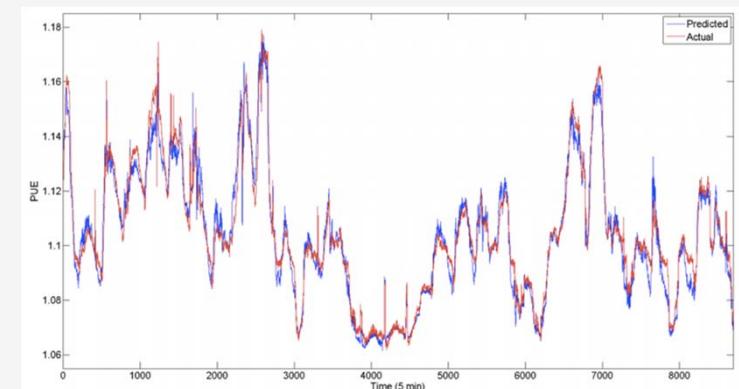
1. Total server IT load [kW]
2. Total Campus Core Network Room (CCNR) IT load [kW]
3. Total number of process water pumps (PWP) running
4. Mean PWP variable frequency drive (VFD) speed [%]
5. Total number of condenser water pumps (CWP) running
6. Mean CWP variable frequency drive (VFD) speed [%]
7. Total number of cooling towers running
8. Mean cooling tower leaving water temperature (LWT) setpoint [F]
9. Total number of chillers running
10. Total number of drycoolers running
11. Total number of chilled water injection pumps running
12. Mean chilled water injection pump setpoint temperature [F]
13. Mean heat exchanger approach temperature [F]
14. Outside air wet bulb (WB) temperature [F]
15. Outside air dry bulb (DB) temperature [F]
16. Outside air enthalpy [kJ/kg]
17. Outside air relative humidity (RH) [%]
18. Outdoor wind speed [mph]
19. Outdoor wind direction [deg]



- **학습된 DNN**
  - 제어값과 PUE의 관계를 학습
  - 임의의 제어값에 대한 PUE를 알 수 있다.
  - 학습된 DNN은 시뮬레이터로 사용할 수 있다.

- **예측 결과**

- 시뮬레이션이 가능하면 임의의 제어값에 대한 PUE를 미리 알 수 있다.
- 다양한 입력에 대한 시뮬레이션으로, 최선의 PUE를 찾을 수 있다.



# 기본코드를 이용한 DNN 실습

# Deep Learning 실습

Colab 사용

코드 구조 이해

- Train set으로 학습, test set으로 평가

Keras DNN 기본 Template 이용 실습

- Hidden layer 개수
- Node 개수

학습에 영향을 미치는 요소들

- Data Shuffle
- Batch size
- Epoch 수
- Data 규모

# Keras DNN Template

```

import numpy as np
import matplotlib.pyplot as plt

from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input

import time

#모델정의
model=keras.Sequential()
model.add(Dense(10, activation='tanh', input_shape=(1,)))
model.add(Dense(10, activation='tanh'))
model.add(Dense(1))

#모델 컴파일
model.compile(optimizer='SGD', loss='mse', metrics=['mse'])
model.summary()

#학습 & 시간표시
start_time=time.time()
model.fit(train_x, train_y, epochs=1000, verbose=0, batch_size=20)
print('elapsed : {}'.format(time.time()-start_time))

#평가
loss,mse=model.evaluate(test_x, test_y)
print('loss= ', loss)
print('mse= ', mse)

display(Image('base_result.png')) #ADDED

#예측
y_=model.predict(test_x)

#출력
plt.scatter(test_x, test_y)
plt.scatter(test_x, y_, color='r')
plt.show()

```

```

# 함수정의 보기
# def model.fit(x=None, y=None,
batch_size=None, epochs=1, verbose=1,
callbacks=None, validation_split=0.0,
validation_data=None, shuffle=True,
class_weight=None, sample_weight=None,
initial_epoch=0, steps_per_epoch=None,
validation_steps=None, validation_freq=1,
max_queue_size=10, workers=1,
use_multiprocessing=False, **kwargs)

```

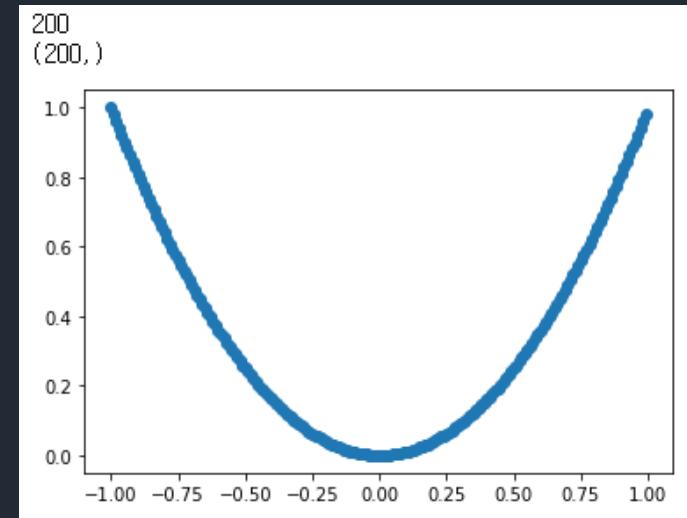
# Data Type I

## Data 준비

```
import numpy as np  
x=np.arange(-1, 1, 0.01)  
y=x**2  
plt.scatter(x, y)
```

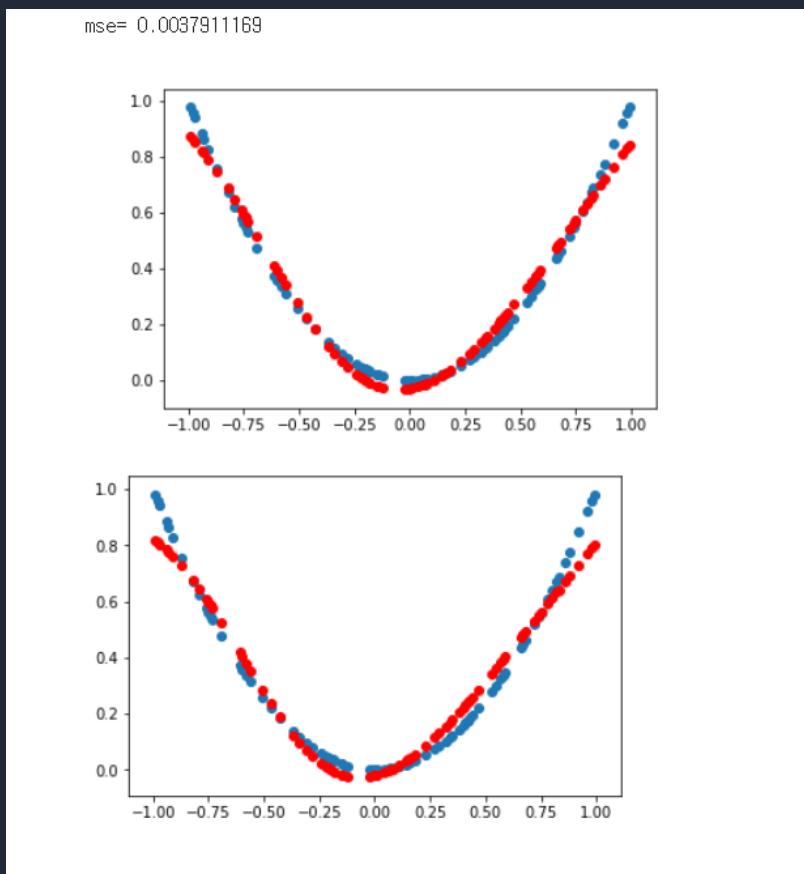
```
print(len(x))  
print(x.shape)
```

## 출력 모습

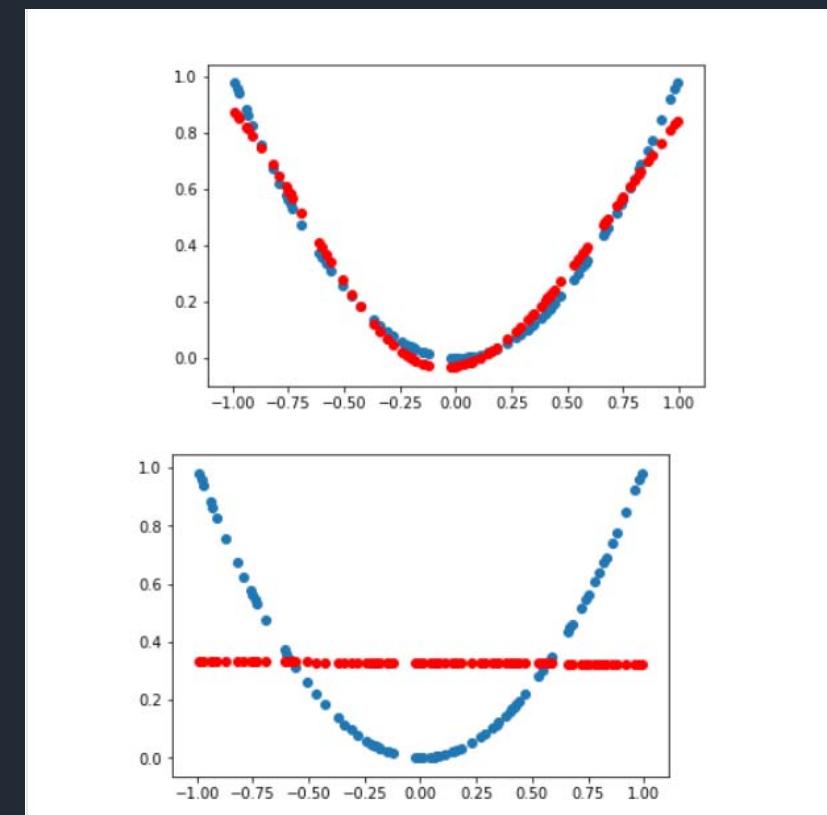


# Hidden Layer 개수 조절

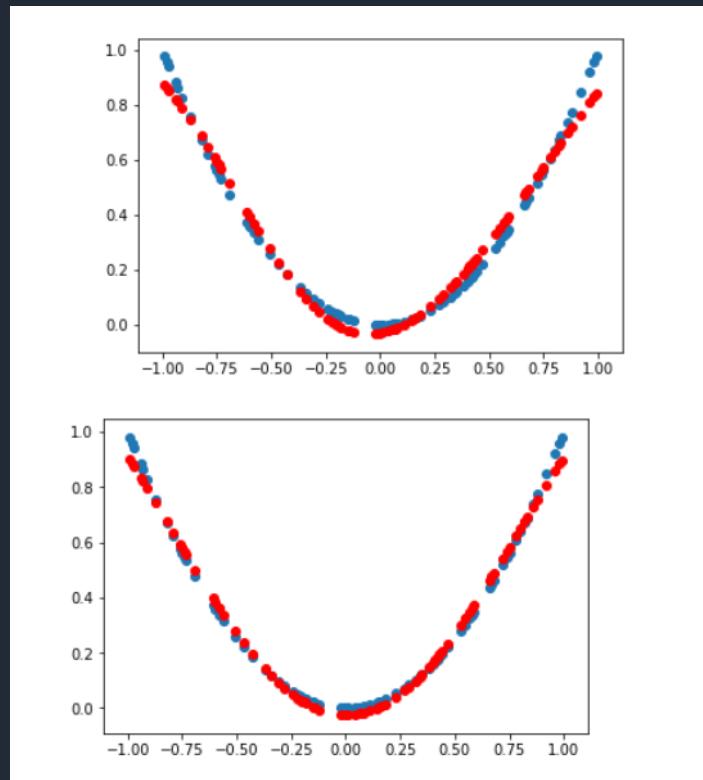
Hidden layer 1개



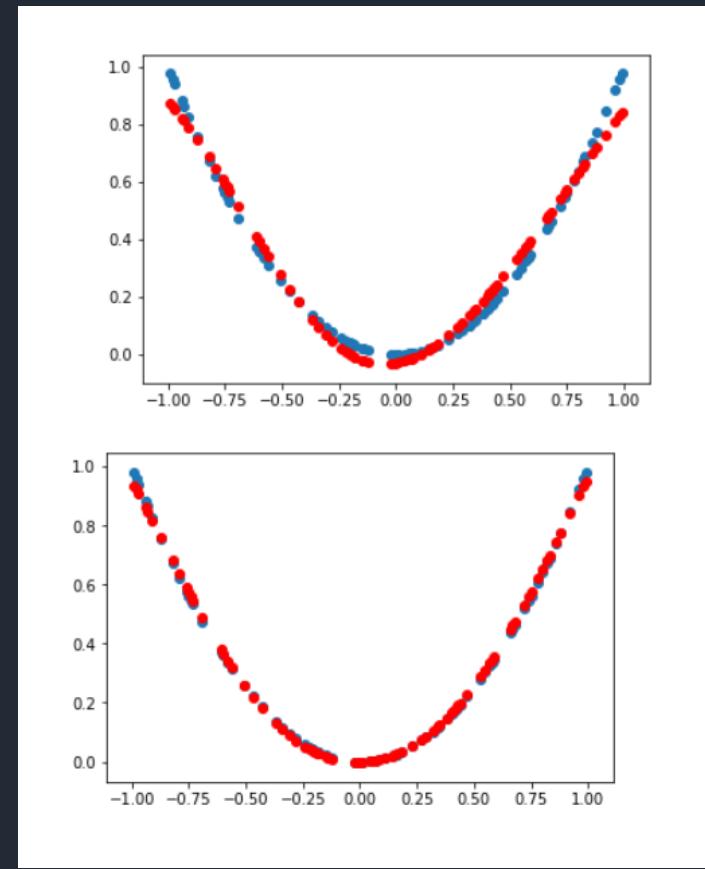
Hidden layer 없으면



## Hidden layer 5개

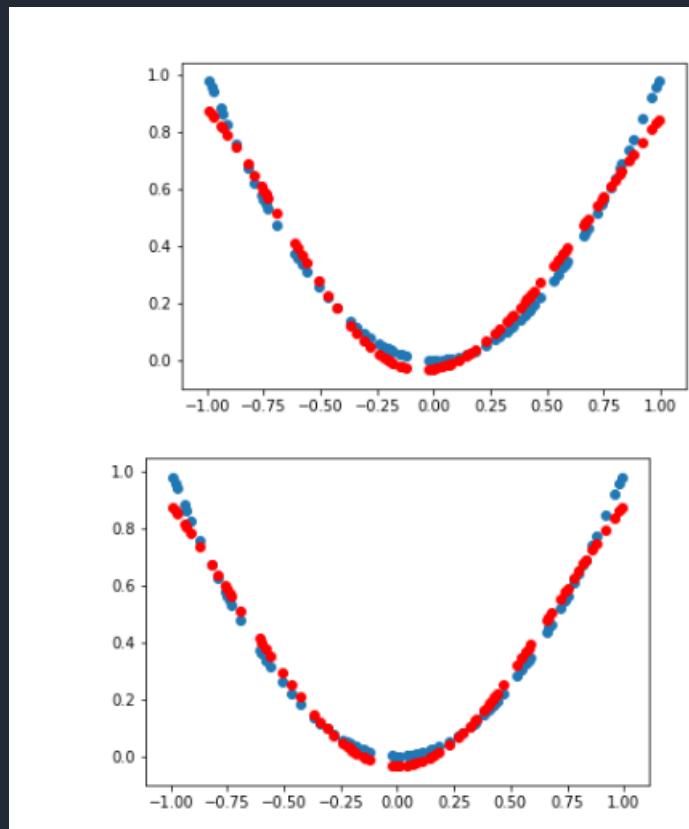


## Hidden layer 10개

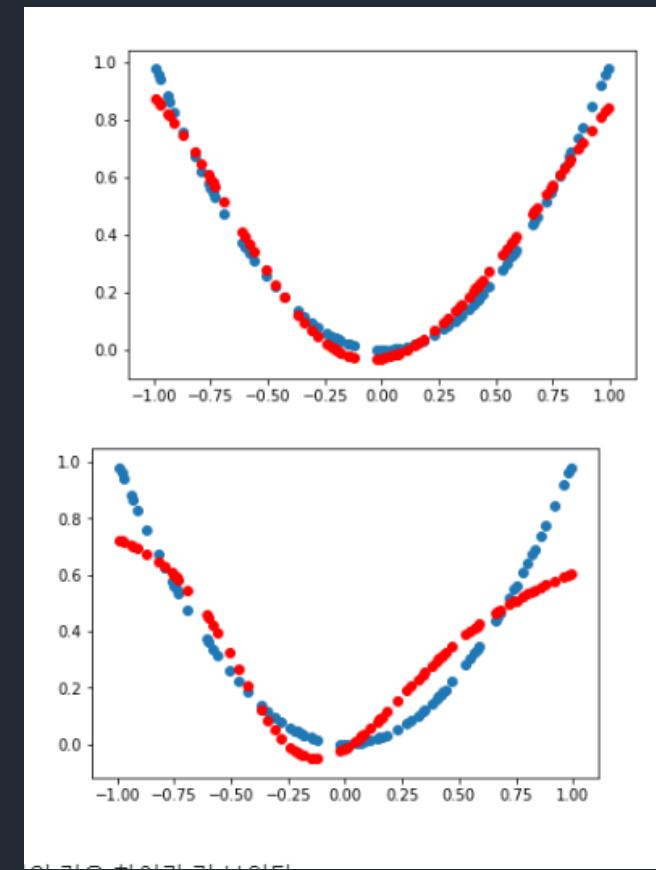


# Node 수 조절

Node 5개

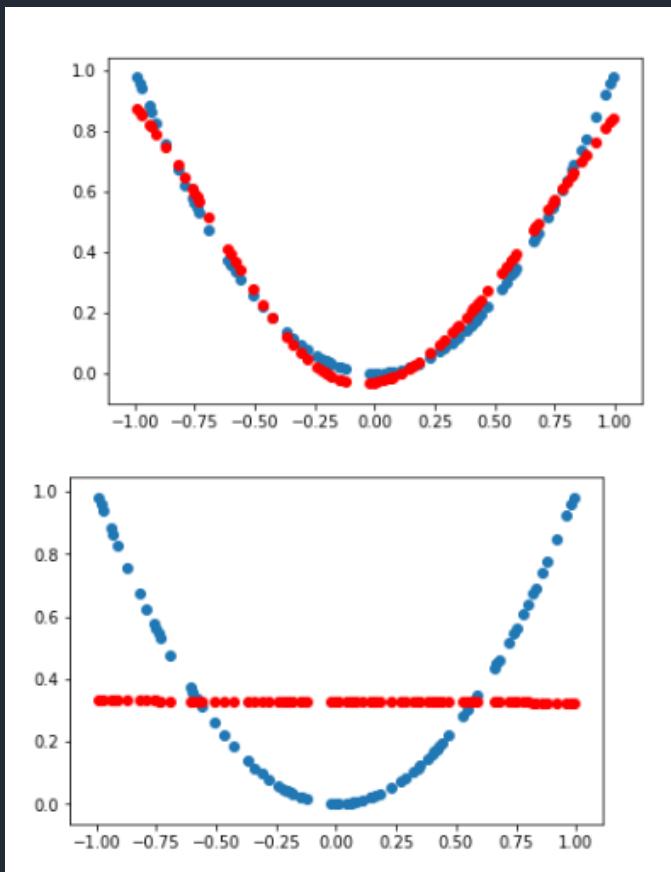


Node 2개



AI 기초 - 학습률 조절

## Node 1개



## Summary

- Node 1개로는 학습이 거의 되지 않음
- Layer, node 수 많을수록 학습 정도가 높아지나, 시간이 오래 걸림
- 문제의 complexity에 따라 model의 적합도가 다를 수 있음: 모델을 적절하게 수정해서 적용할 필요가 있음

# Data Type II

## Data 준비

```
#Data 준비를 위한 함수 정의
def get_sin_data(start=0, end=10, step=0.1):
    x = np.arange(start,end,step)
    np.random.shuffle(x)
    y = np.sin(x)

    split_index = int(x.shape[0]*0.6)

    train_x, test_x = x[:split_index], x[split_index:]
    train_y, test_y = y[:split_index], y[split_index:]

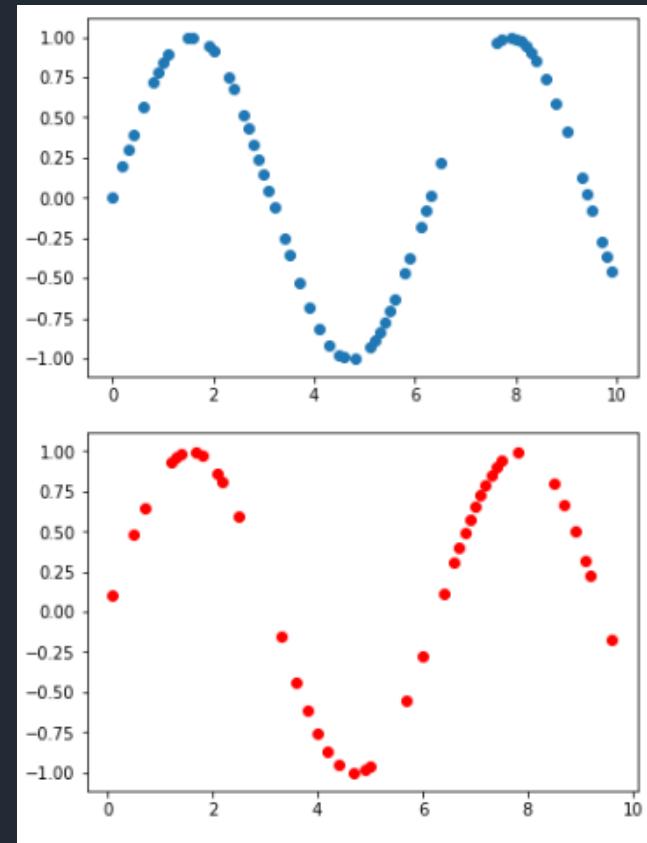
    return (train_x, train_y), (test_x, test_y)

#Data 준비
(train_x, train_y), (test_x, test_y) = get_sin_data(start=0, end=10, step=0.1)

plt.scatter(train_x,train_y)
plt.show()

plt.scatter(test_x,test_y,color="r")
plt.show()
```

## 출력모습



## Batch size - GPU와 관련된 옵션

### 한번에 GPU에 보내는 데이터의 수를 의미

- batch\_size가 1일 경우 1개를 보내고, 1개의 결과를 받고, 1번 웨이트를 업데이트 한다.
- batch\_size가 10일 경우 10개를 보내고, 10개의 결과를 한 번에 받고, 1번 웨이트를 업데이트 한다.

GPU는 보통 수천개의 코어를 가지고 있다. 동시에 꽤 많은 연산을 처리할 수 있으나, 데이터가 적으면 대부분은 사용하지 못하고 일부만 연산에 사용된다.

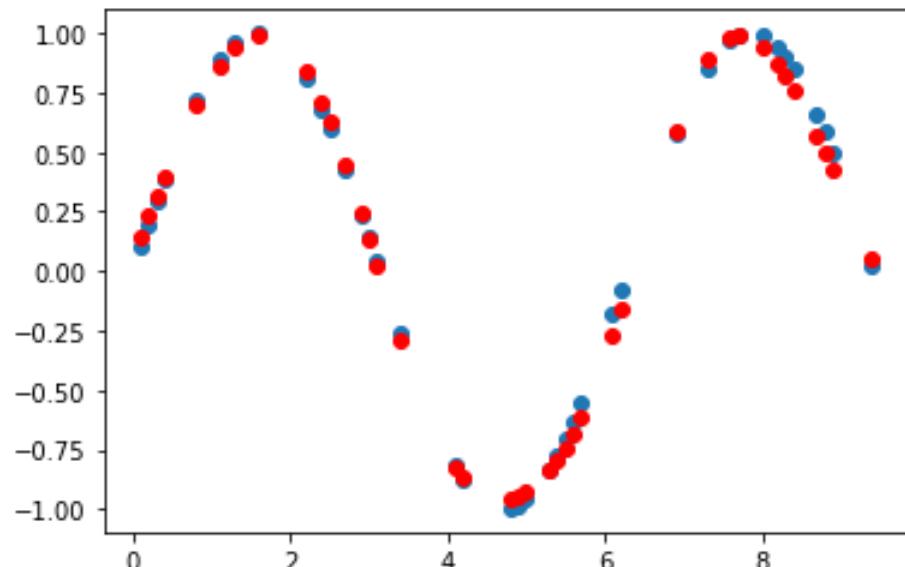
복수의 데이터를 한번에 보내어 한번에 연산을 할 수 있고, 그 결과를 반환할 수 있다. 이런 방법으로 연산 시간을 줄일 수 있다.

하지만, 복수의 데이터를 한번에 보내는 경우 한번에 보낸 결과가 한번에 오고 1번 업데이트 되면서 업데이트 되는 양상이 optimize되지 않는 단점이 있다.

# 학습 횟수에 따른 변화

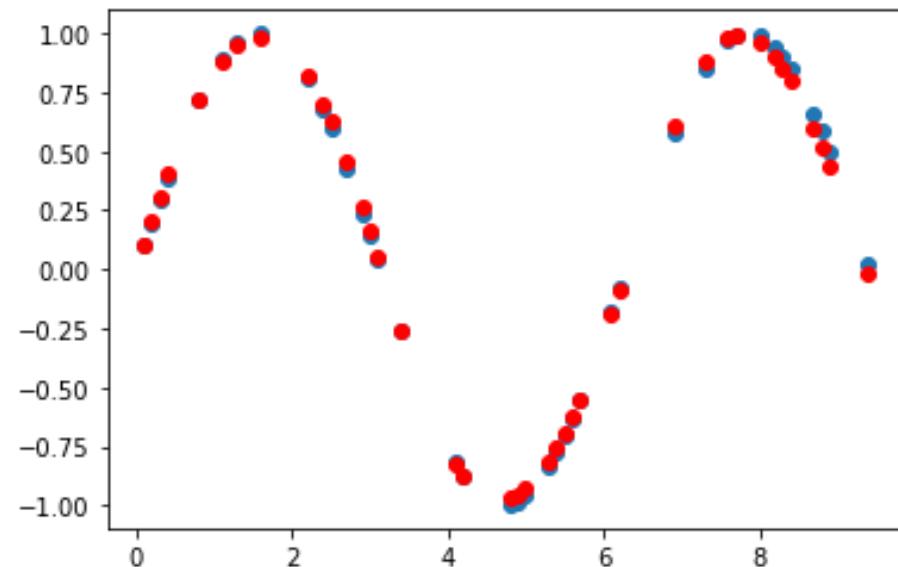
4 times fitting

elapsed : 3.806091070175171



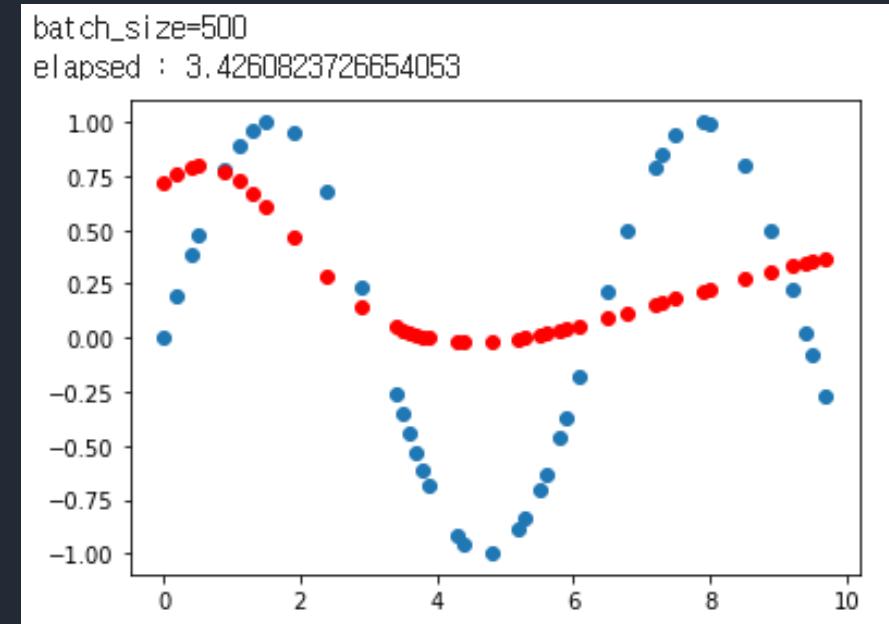
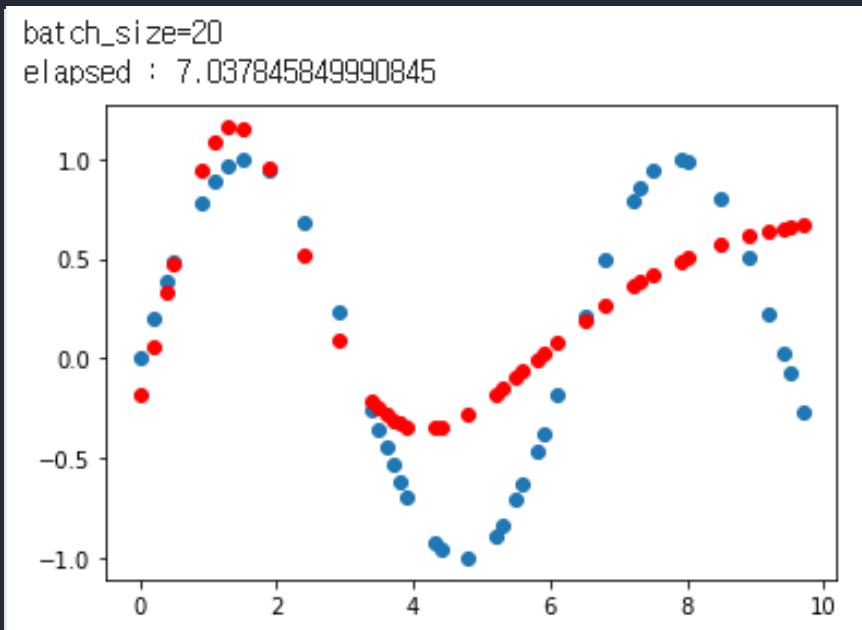
9 times fitting

elapsed : 3.809114694595337



# Batch Size 옵션 변화

Batch size 적용 [1,2,5,10,20,50,100,200,500]



Batch size 커질수록 비례해서 소요시간 감소

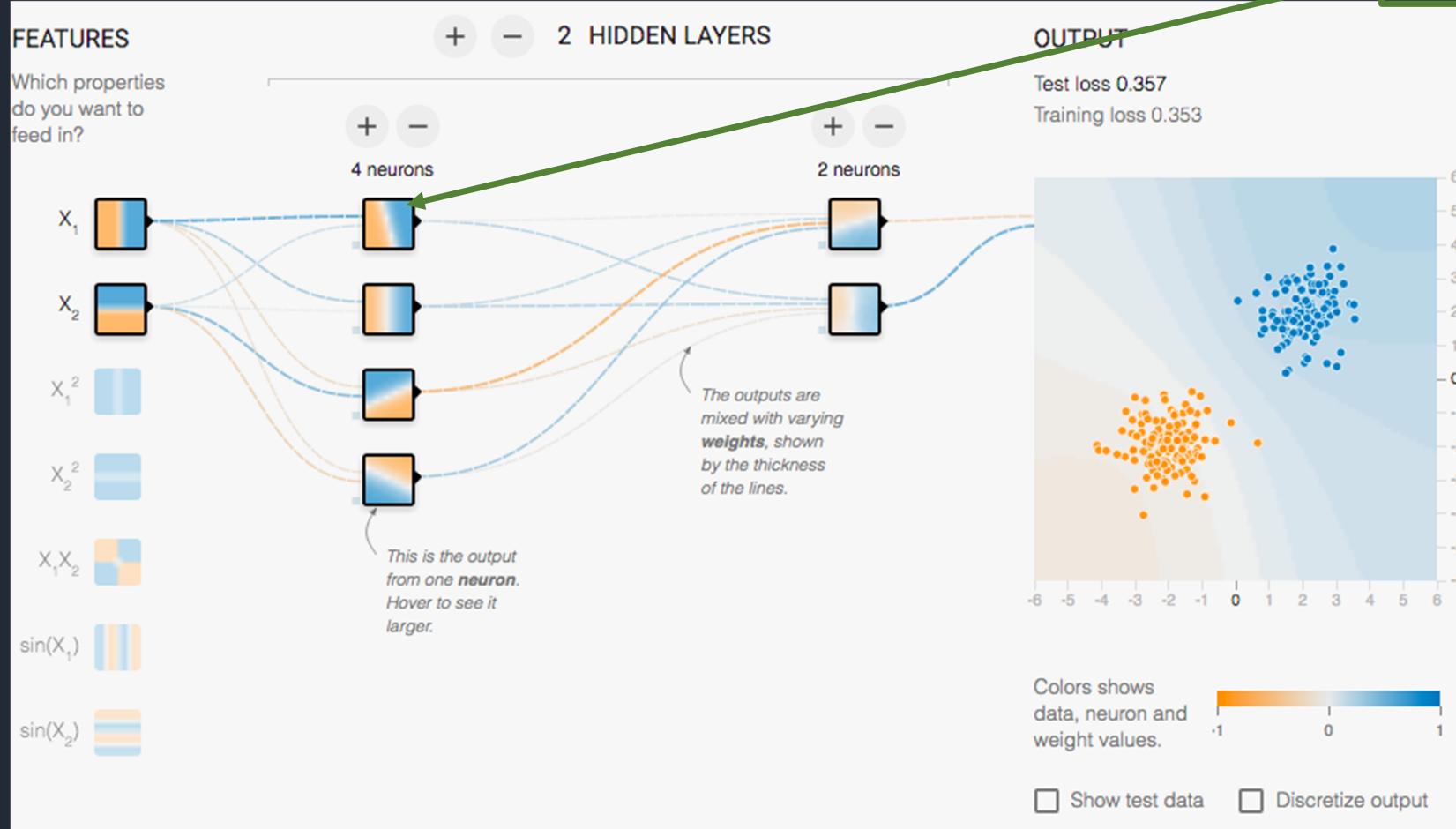
결과는 안좋아짐

적절한 batch size는 모델에 따라 경험적으로 찾아야

# 온라인 시뮬레이션

<http://playground.tensorflow.org>

네모 하나하나가 퍼셉트론



# History의 Loss 값 추이를 통한 Overfitting 확인

## 코드

```
# model.fit(train_x, train_y, epochs=1000,
verbose=0, batch_size=20)

history = model.fit(train_x, train_y,
epochs=1000, verbose=0, batch_size=20)

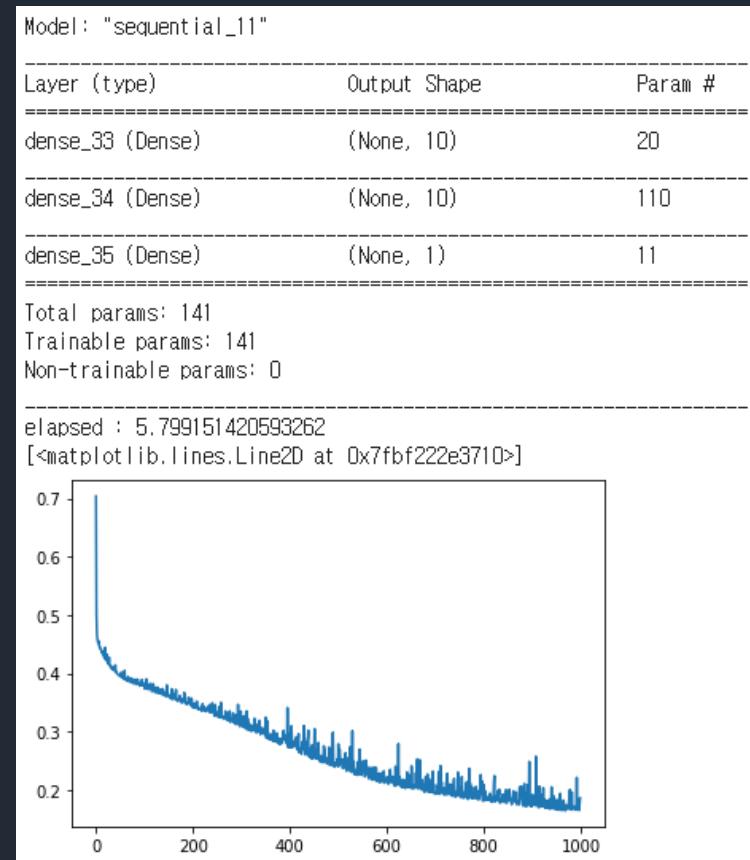
print("elapsed : {}".format(time.time() -
start_time))

plt.plot(history.history['loss'])
```

### 옵션

- Verbose=0 아무 표시 없음
- Verbose=1 Progress bar
- Verbose=2 1 line per each epoch

## 출력



# Data Shuffle의 영향

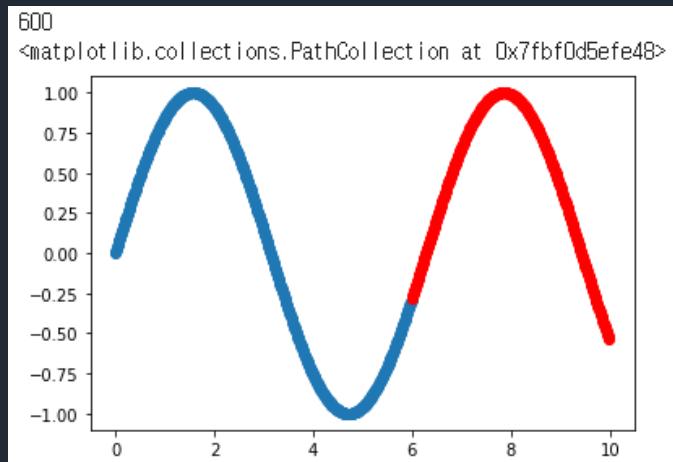
## Data 준비

```
x = np.arange(0,10,0.1)
# np.random.shuffle(x) # COMMENTED
y = np.sin(x)

split_index = int(x.shape[0]*0.6)
print(split_index)

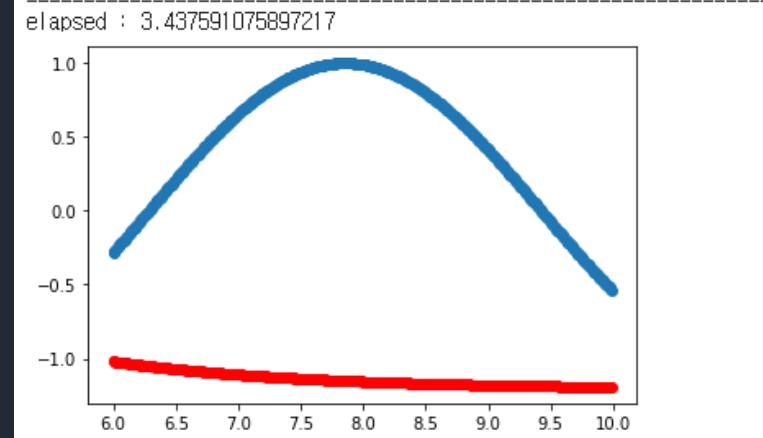
train_x, test_x = x[:split_index], x[split_index:]
train_y, test_y = y[:split_index], y[split_index:]

plt.scatter(train_x, train_y)
plt.scatter(test_x, test_y, color="r")
```



## 학습 결과

```
Model: "sequential_25"
-----
Layer (type)          Output Shape       Param #
dense_75 (Dense)     (None, 10)           20
-----
dense_76 (Dense)     (None, 10)           110
-----
dense_77 (Dense)     (None, 1)            11
-----
Total params: 141
Trainable params: 141
Non-trainable params: 0
```



## Data 준비

```

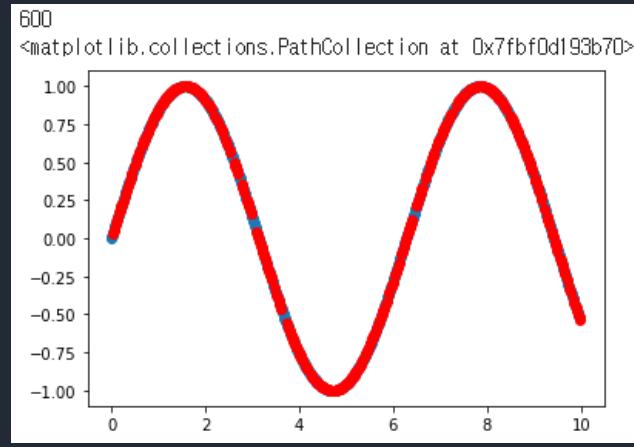
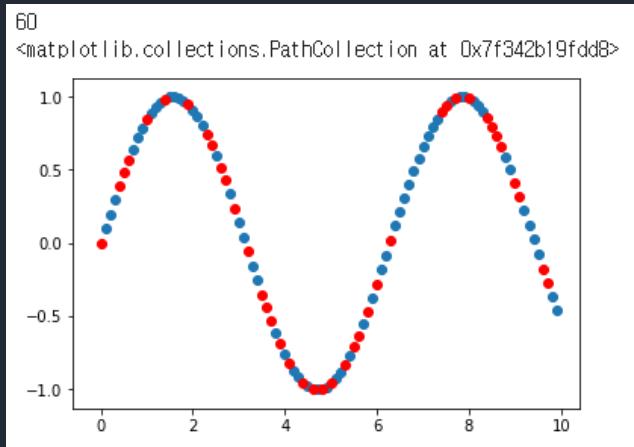
x = np.arange(0,10,0.1)
np.random.shuffle(x) # UNCOMMENT
y = np.sin(x)

split_index = int(x.shape[0]*0.6)
print(split_index)

train_x, test_x = x[:split_index], x[split_index:]
train_y, test_y = y[:split_index], y[split_index:]

plt.scatter(train_x, train_y)
plt.scatter(test_x, test_y, color="r")

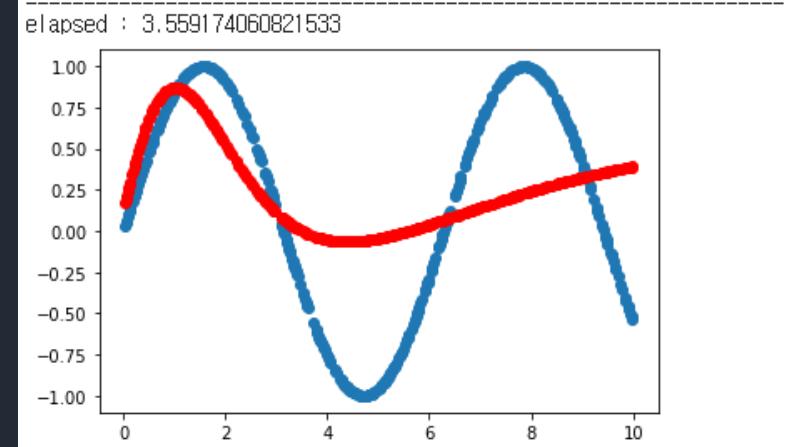
```



## 학습결과

Layer (type)	Output Shape	Param #
dense_81 (Dense)	(None, 10)	20
dense_82 (Dense)	(None, 10)	110
dense_83 (Dense)	(None, 1)	11

Total params: 141  
Trainable params: 141  
Non-trainable params: 0



# Data의 중요성 - Train with Random Data

## Data 준비

```
X=np.arange(0, 10, 0.1)
```

```
np.random.shuffle(x)
```

```
#y=np.sin(x)
```

```
y=np.random.random_sample(s.shpa[0])
```

```
split_index=int(x.shape[0]*0.6)
```

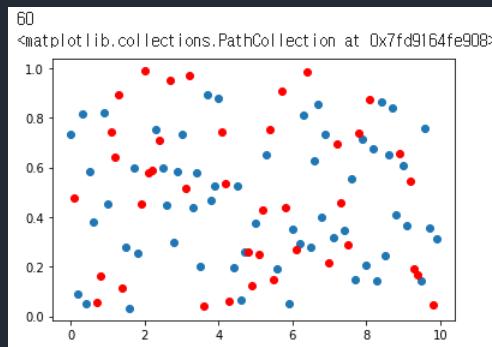
```
print(split_index)
```

```
train_x, test_x = x[:split_index], x[split_index:]
```

```
train_y, test_y = y[:split_index], y[split_index:]
```

```
plt.scatter(train_x, train_y)
```

```
plt.scatter(test_x, test_y, color='r')
```



## 학습결과

```
WARNING:tensorflow:From /tensorflow-1.15.2/python3.6/tensorflow_core/pyt
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
Model: "sequential"
```

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	20
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 1)	11

```
Total params: 141
```

```
Trainable params: 141
```

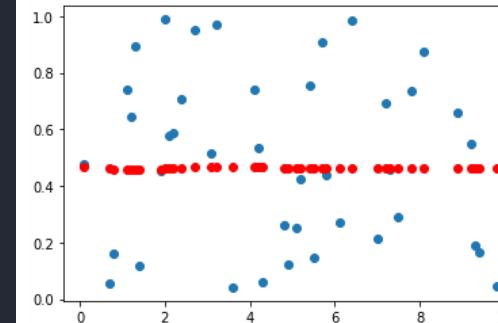
```
Non-trainable params: 0
```

```
elapsed : 4.945517539978027
```

```
40/40 [=====] - 0s 507us/sample - loss: 0.0907
```

```
loss= 0.09074879586696624
```

```
mse= 0.0907488
```



# Garbage in, garbage out.

---

랜덤함수의 경우 x와 y간 관계가 없으므로 학습할 수 있는 패턴이 존재하지 않음

입출력 간 관계가 없는 함수는 학습되지 않음

그러므로 실제 data로 학습이 안되면 data 점검이 필요함

- 정말 중요한 값이 포함되어 있는지
- label이 정확한지

# Overfitting

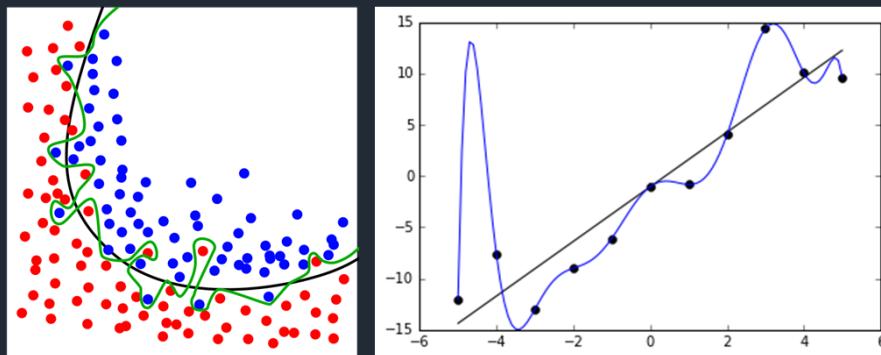
Data에 나타나는 다양성을 심하게 받아들여 나타나는 현상

Data가 충분치 못해 발생- data가 부족하면 일찍 발생함

알고리즘 상으로 완화시키는 기업으로 처리 (거의 필수적)

- Dropout – 값의 추이를 둔화시키는 방법
- Batch Normalization (가장 좋음) – 0-1 값으로 변형시켜주는 방법
- Regularization

근본적으로 data가 적어서 나타나는 현상이므로 data augmentation으로 해결하기도 함



→ Overfitting된 graph:  
오히려 성능이 떨어짐

# DropOut Code

```
(train_x, train_y), (test_x, test_y) = get_sin_data (start=0, end=10, step=0.1)
```

```
model=keras.Sequential()  
model.add(Dense(10, activation='tanh', input_shape=(1,))  
model.add(Dropout(0.1)) #Dropout added  
model.add(Dense(10, activation='tanh'))  
model.add(Dense(1))
```

```
model.compile (optimizer='SGD', loss='mse', metrics=['mse'])  
model.summary()
```

```
model.fit(train_x, train_y, epochs=1000, verbose=0, batchsize=20)
```

```
y_=model.predict(test_x) # 예측데이터
```

# Batch Normalization Code

```
from tensorflow.keras.layers import BatchNormalization
```

```
(train_x, train_y), (test_x, test_y)=get_sin_data(start=0, end=10, step=0.1)
```

```
model=keras.Sequential()
```

```
model.add(Dense(10, activation='tanh', input_shape=(1,)))
```

```
model.add(BatchNormalization()) #Added
```

```
model.add(Dense(10, activation='tanh'))
```

```
model.add(Dense(1))
```

레이어 사이에 삽입해서 출력값 숫자가  
극단적으로 치우치는 것을 방지하는 레이어  
(입력값을 0~1 사이의 값으로 변형하여 넣음)

## Normalization

- 입력이 여러 차원일 때 각 차원의 입력을 동일한 스케일로 맞춰 주면 학습이 빠르게 진행됨

Before Normalization

After Normalization

단순한 모델의 데이터로는 효과를 확인할 수 없음

# Regularization

## 방법

Dense() 생성시에 kernel-regularization, bias\_regularization으로 설정

→ weight 값을 조절함

- l1()
- l1\_l2()
- l2()

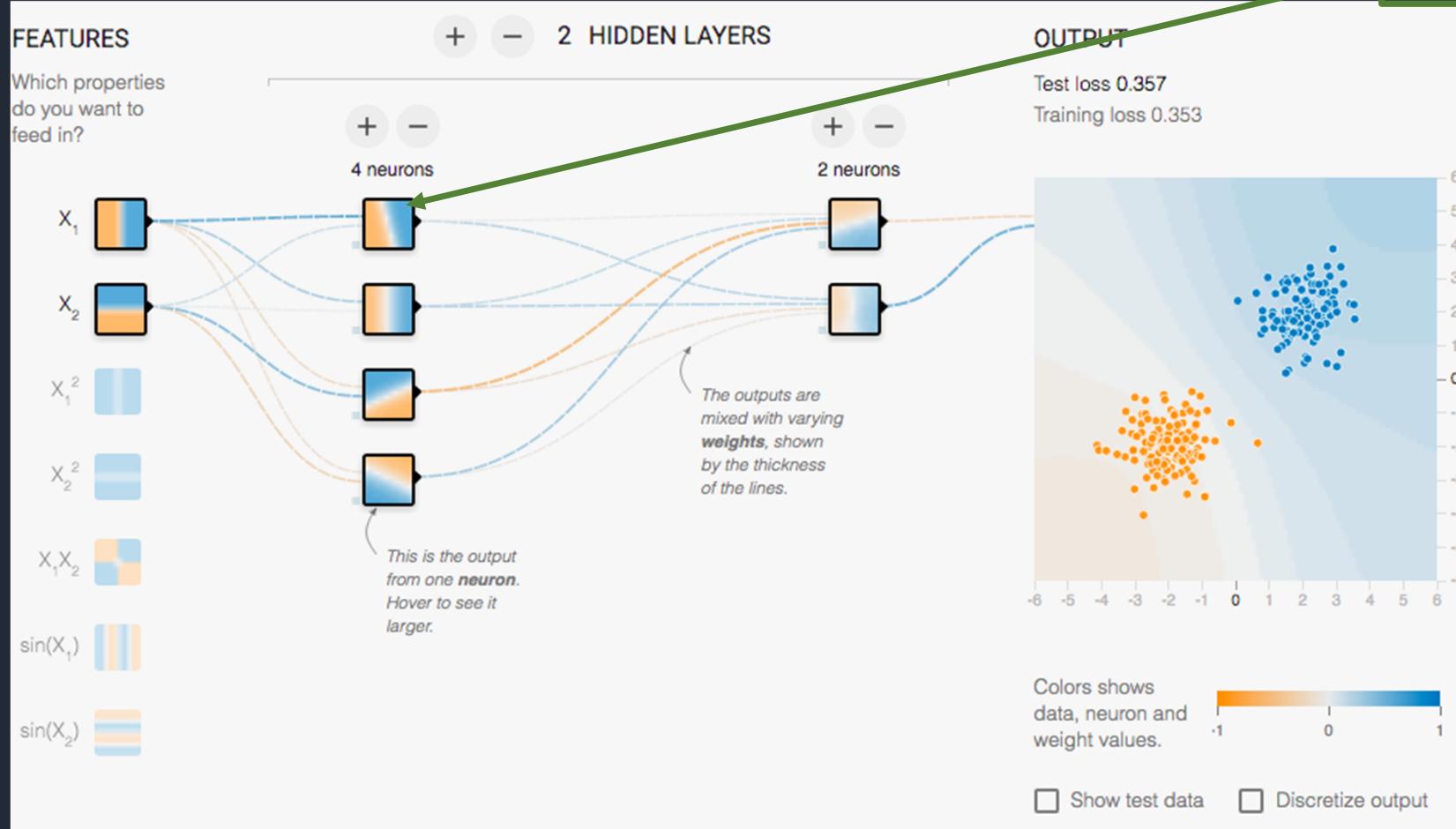
## Code

```
from tensorflow.keras.regularizers import l1, l2  
  
(train_x, train_y), (test_x, test_y) = get_sin_data(start=0,  
end=10, step=0.1)  
  
model=keras.Sequential()  
model.add(Dense(10, activation='tanh', input_shape=(1,),  
kernel_regularizer=l2(0.001)))  
model.add(Dense(10, activation='tanh',  
kernel_regularizer=l2(0.001)))  
model.add(Dense (1))  
  
model.compile(optimizer='SGD', loss='mse',  
metrics=['mse'])  
model.summary()
```

# 온라인 시뮬레이션

<http://playground.tensorflow.org>

네모 하나하나가 퍼셉트론



# CallBack



# CallBack

학습 도중 로그 출력이나 모델 저장 등 다양한 옵션을 취할 수 있음

## CallBack 적용 예시

- 모델 저장
- Loss 출력
- Early Stopping
- 학습율 조정
- 모두한번에

### Understanding Callbacks

We start by creating our own *Callback objects*. In these objects, we decide what we want to do at a certain point in training. For example, there is a callback in fastai called *SaveModelCallback()*. This callback checks if we have a *better model* at the end of every epoch and *saves the model* if we do. This way we don't just end up with the last parameters but the best ones.

```
class SaveModelCallback(TrackerCallback):
    "A `TrackerCallback` that saves the model when monitored quantity is best."
    def __init__(self, learn:Learner, monitor:str='valid_loss', mode:str='auto', every:str='improvement', name:str='bestmodel')

    def on_epoch_end(self, epoch:int, **kwargs:Any) -> None:
        "Compare the value monitored to its best score and maybe save the model."
        if self.every == "epoch": self.learn.save(f'{self.name}_{epoch}')
        else: #every="improvement"
            current = self.get_monitor_value()
            if current is not None and self.operator(current, self.best):
                print(f'Better model found at epoch {epoch} with {self.monitor} value: {current}.')
                self.best = current
                self.learn.save(f'{self.name}')
```

출처: <https://towardsdatascience.com/customize-your-training-loop-with-callbacks-9d93b415a602>

## 모델저장 - 모든 epoch에서 체크. overfitting 발생 직전 최적의 결과도 확보 가능

### 함수정의

```
def train_with_callbacks(callbacks):

    # 모델 정의
    model = keras.Sequential()
    model.add(Dense(10, activation='tanh', input_shape=(1,)))
    model.add(Dense(10, activation='tanh'))
    model.add(Dense(1))

    # 모델 컴파일
    model.compile(optimizer="SGD", loss="mse",
                  metrics=["mse"])
    model.summary()

    # 학습
    start_time = time.time()
    model.fit(train_x, train_y, epochs=1000, verbose=0,
              batch_size=20, validation_split=0.1, callbacks=callbacks)
    print("elapsed : {}".format(time.time() - start_time))
```

### 실행 코드

```
from tensorflow.keras.callbacks import
    ModelCheckpoint

    model_check_point =
        ModelCheckpoint('best_model.h5',
                        monitor='val_loss', mode='min',
                        save_best_only=True)

    # monitor='val_loss': Overfitting 발생하는 지점
    # 확인

    train_with_callbacks([model_check_point])
```

# Early Stopping

## 코드

```
from tensorflow.keras.callbacks import  
EarlyStopping  
  
early_stopping =  
EarlyStopping(monitor='val_loss', # 모니터링  
대상  
    mode='auto',      # 학습 방향 자동탐지  
    patience=50)     # 중지까지의 여유분  
  
train_with_callbacks([early_stopping])
```

## 실행결과

Model: "sequential_4"		
Layer (type)	Output Shape	Param #
dense_12 (Dense)	(None, 10)	20
dense_13 (Dense)	(None, 10)	110
dense_14 (Dense)	(None, 1)	11
Total params: 141		
Trainable params: 141		
Non-trainable params: 0		
elapsed : 7.130956649780273		

## 학습율 조정 - 학습과정을 모니터링하면서 진척되지 않으면 학습율을 조정함

### 코드

```
from tensorflow.keras.callbacks import
ReduceLROnPlateau

reduce_lr =
ReduceLROnPlateau(monitor='val_loss', # 모
니터링 대상
    factor=0.2,      # 줄이는 양
    patience=5,      # 대상 기간동안 유지
    min_lr=0.001)    # 최소 학습율
```

```
train_with_callbacks([reduce_lr])
```

### 실행결과

Model: "sequential_5"		
Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 10)	20
dense_16 (Dense)	(None, 10)	110
dense_17 (Dense)	(None, 1)	11

Total params:	141
Trainable params:	141
Non-trainable params:	0
elapsed : 10.464774131774902	

# Loss 출력

## 코드

```
# copy from https://gist.github.com/stared/dfb4dfa6d9a8501cd1cc8b8cb806d2e
from IPython.display import clear_output

class PlotLosses(Callback):

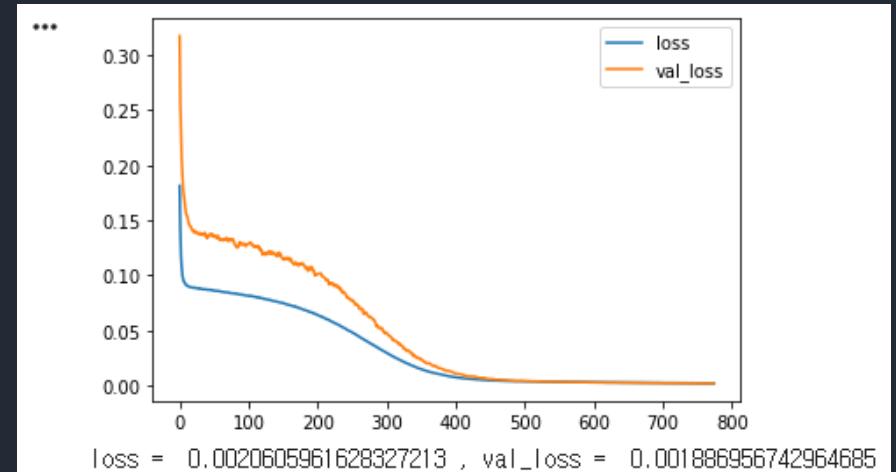
    def on_train_begin(self, logs={}):
        self.i = 0
        self.x = []
        self.losses = []
        self.val_losses = []

        self.fig = plt.figure()
        self.logs = []

    def on_epoch_end(self, epoch, logs={}):
        self.logs.append(logs)
        self.x.append(self.i)
        self.losses.append(logs.get('loss'))
        self.val_losses.append(logs.get('val_loss'))
        self.i += 1

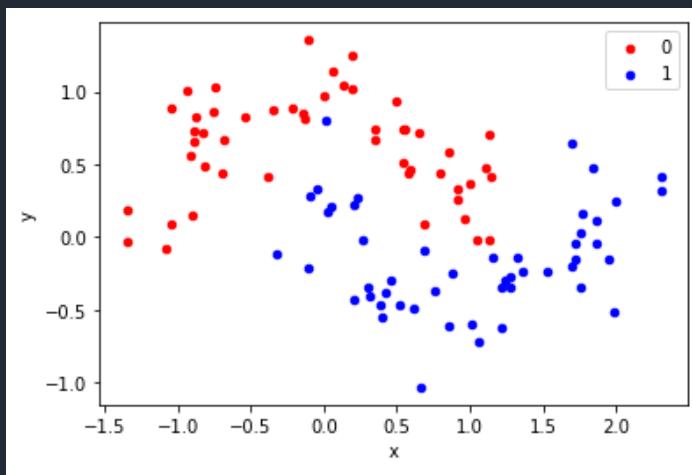
        clear_output(wait=True)
        plt.plot(self.x, self.losses, label="loss")
        plt.plot(self.x, self.val_losses, label="val_loss")
        plt.legend()
        plt.show();
        print("loss = ", self.losses[-1], ", val_loss = ", self.val_losses[-1])
```

## 출력



# Callback 적용 이전 Data 형태

Data 형태



학습 결과



# ModelCheckpoint, EarlyStopping, ReduceLROnPlateau

## Code

```

from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLROnPlateau

model = Sequential()
model.add(Dense(500, input_shape=(2,), activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

model_check_point = ModelCheckpoint('best_model.h5', monitor='val_loss', mode='min',
save_best_only=True)
plot_losses = PlotLosses()
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=50)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=5, min_lr=0.001)

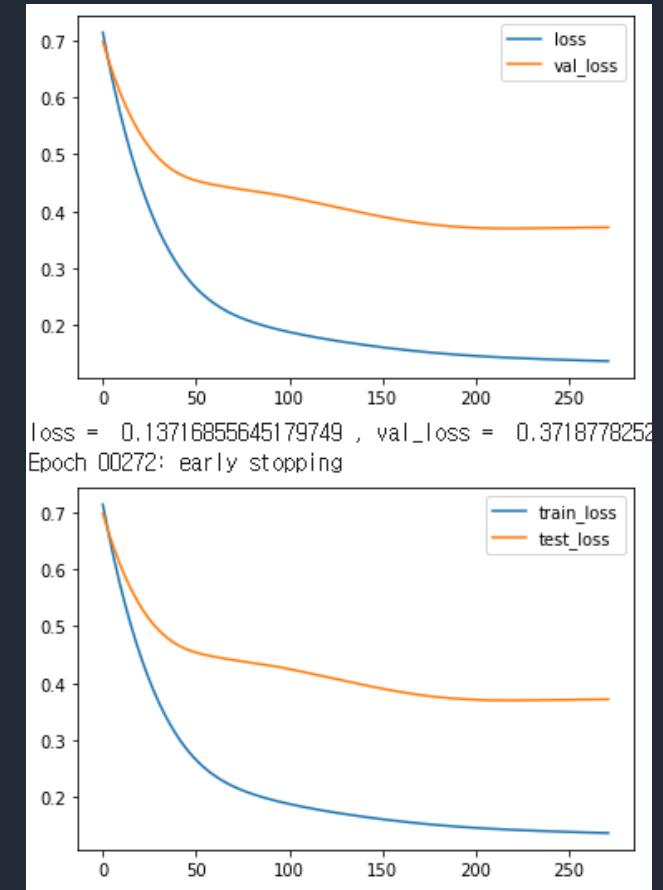
callbacks = [model_check_point, plot_losses, early_stopping, reduce_lr]

# history = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=4000,
verbose=0)
history = model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=4000,
verbose=0, callbacks=callbacks) #overfitting 생기기 전에 중단되도록 설정

pyplot.plot(history.history['loss'], label='train_loss')
pyplot.plot(history.history['val_loss'], label='test_loss')
pyplot.legend()
pyplot.show()

```

## 실행 결과



# References

---

1. AI 마인드 (마틴 포드 2019. 07. 01 터닝포인트)  
<http://aidev.co.kr/book/7875>
2. <https://www.tiobe.com/tiobe-index/>
3. 하용호@kakao  
<https://www.slideshare.net/yongho/ss-79607172>
4. 인공지능 이해 및 활용사례 온라인 강의 -2020 OSSW 개발자대회 온라인 개론
5. 임도형@(주)엑셈
6. [https://www.di.ens.fr/~lelarge/dldiy/slides/lecture\\_5/index.html#213](https://www.di.ens.fr/~lelarge/dldiy/slides/lecture_5/index.html#213)
7. <https://cs231n.github.io/neural-networks-3/>
8. [https://www.wikipendium.no/TDT4137\\_Cognitive\\_Architectures](https://www.wikipendium.no/TDT4137_Cognitive_Architectures)
9. labellImg SW  
<https://github.com/tzutalin/labellImg>
10. 예제로 배우는 파이썬 프로그래밍  
<http://pythonstudy.xyz/>