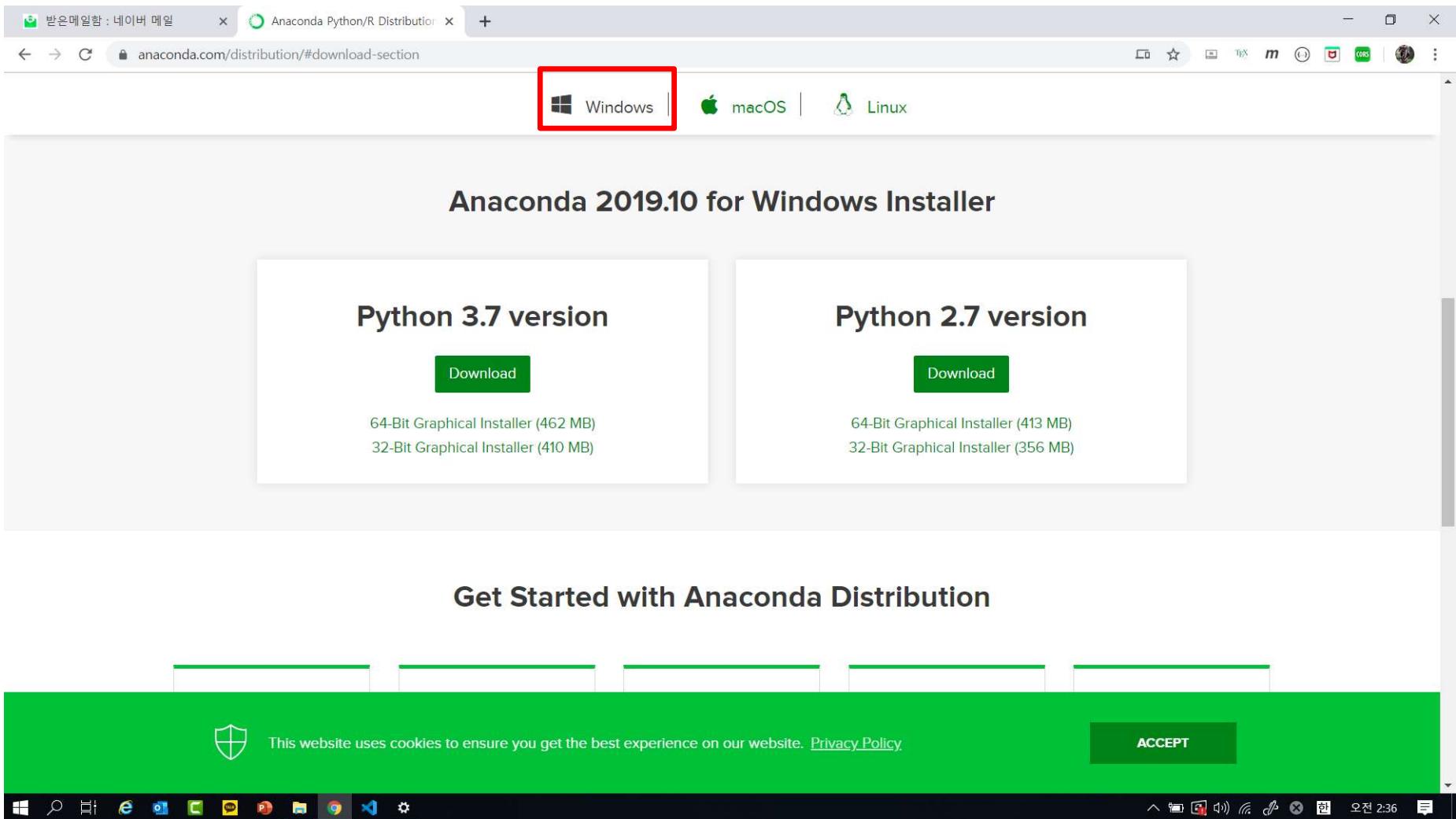


# Python 을 활용한 머신러닝 입문

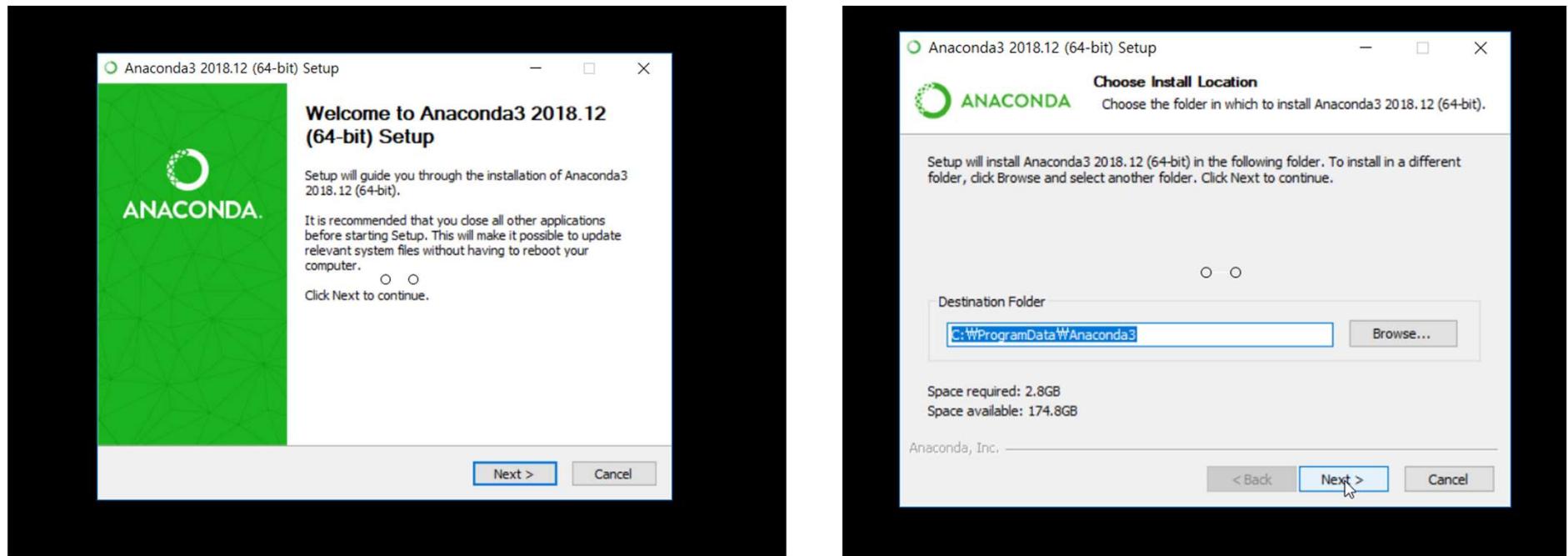
2020. 2

# Anaconda Installation

- <https://www.anaconda.com/distribution/#download-section>



# Anaconda 설치



- 개인별 작업 directory 생성
- 패키지 설치 : pip install <package name>
- 설치된 Python package 확인 : pip list

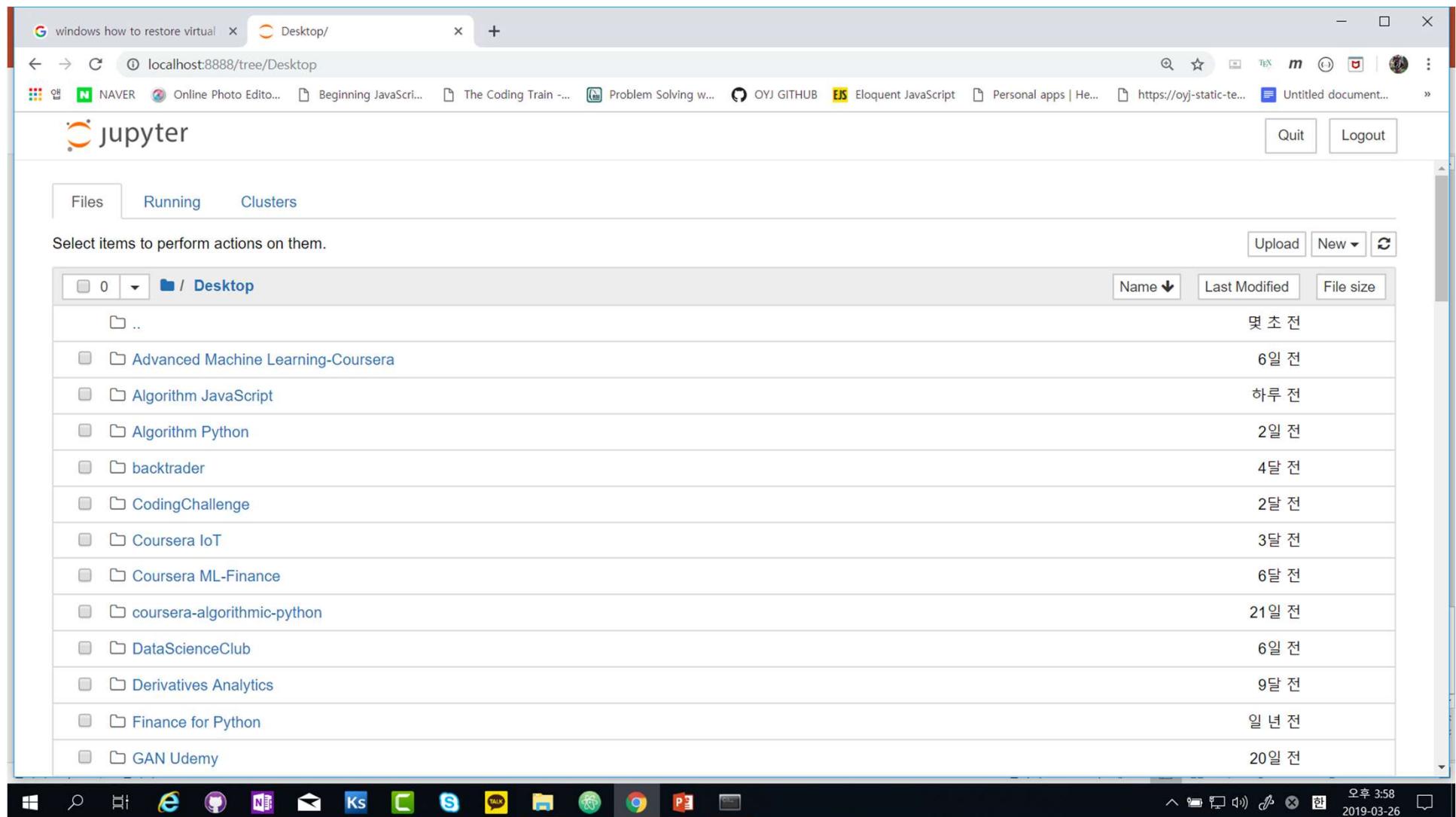


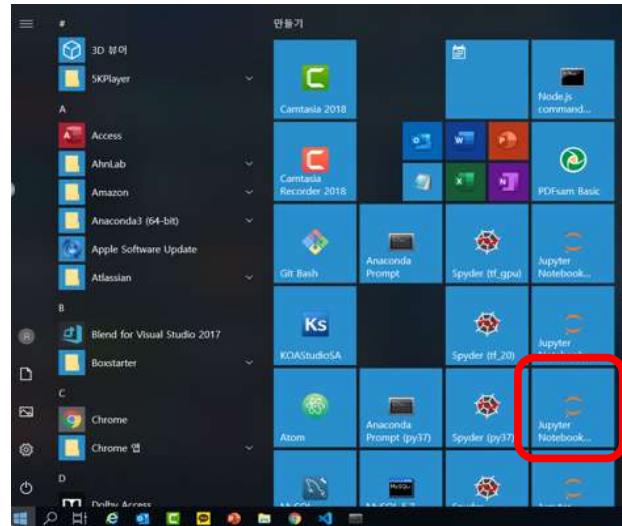
```
Anaconda Prompt

(MLwithPython) C:\Users\Recording>python
Python 3.7.1 (default, Dec 10 2018, 22:54:23) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

(MLwithPython) C:\Users\Recording>pip freeze
alabaster==0.7.12
anaconda-client==1.7.2
anaconda-navigator==1.9.6
anaconda-project==0.8.2
asn1crypto==0.24.0
astroid==2.1.0
astropy==3.1
atomicwrites==1.2.1
attrs==18.2.0
Babel==2.6.0
backcall==0.1.0
backports.os==0.1.1
backports.shutil-get-terminal-size==1.0.0
beautifulsoup4==4.6.3
bitarray==0.8.3
bkcharts==0.2
blaze==0.11.3
bleach==3.0.2
bokeh==1.0.2
boto==2.49.0
Bottleneck==1.2.1
certifi==2018.11.29
cffi==1.11.5
```

# >jupyter notebook





## Anaconda로 설치한 경우 시작 icon 자동 생성

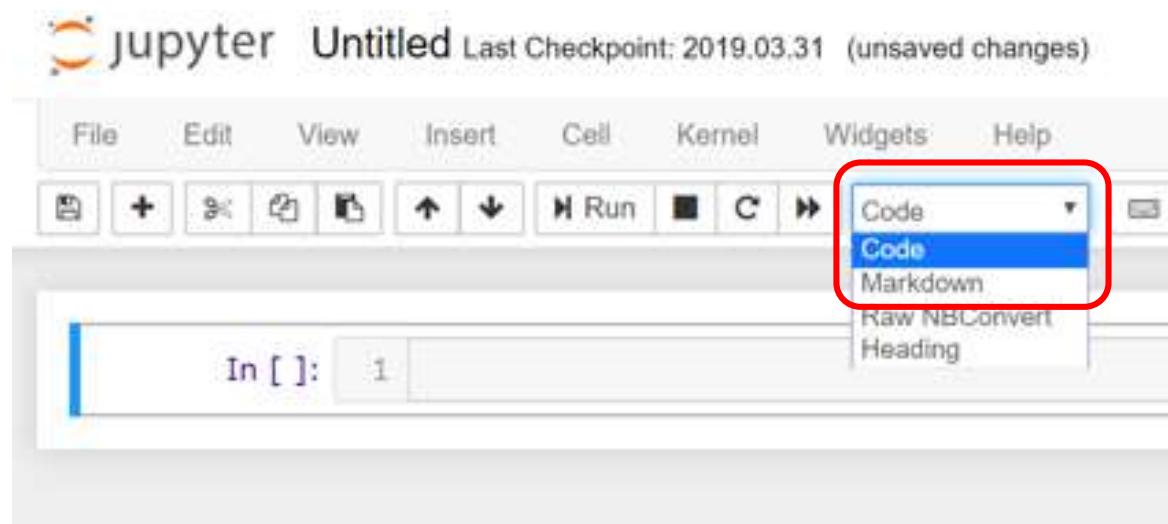
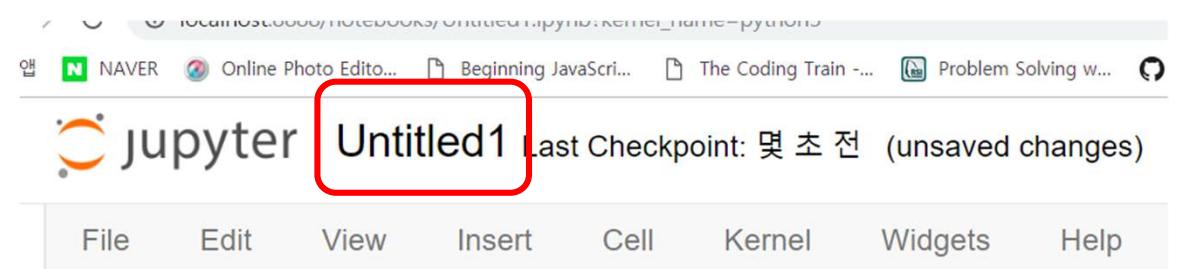
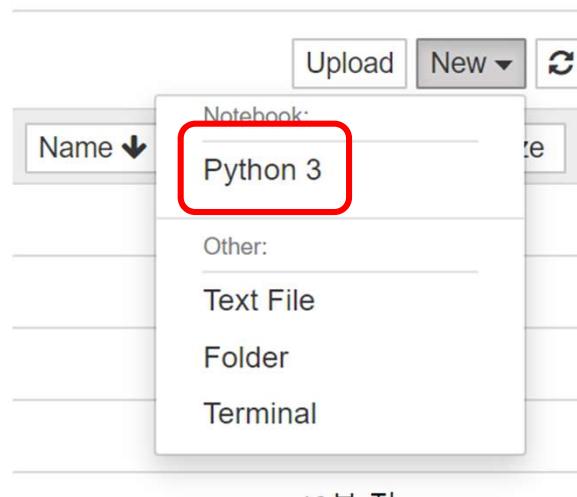
### 명령 prompt > jupyter notebook

```
(py37) C:\Users\trimu>jupyter notebook
[1 09:19:03.913 NotebookApp] JupyterLab extension loaded from C:\Users\trimu\Miniconda3\envs\py37\lib\site-packages\jupyterlab
[1 09:19:03.913 NotebookApp] JupyterLab application directory is C:\Users\trimu\Miniconda3\envs\py37\share\jupyter\lab
[1 09:19:03.915 NotebookApp] Serving notebooks from local directory: C:\Users\trimu
[1 09:19:03.916 NotebookApp] The Jupyter Notebook is running at:
[1 09:19:03.916 NotebookApp] http://localhost:8888/?token=cde0891905a6a5459965881382c153b4f9f5593f1cab1221
[1 09:19:03.916 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 09:19:04.085 NotebookApp]

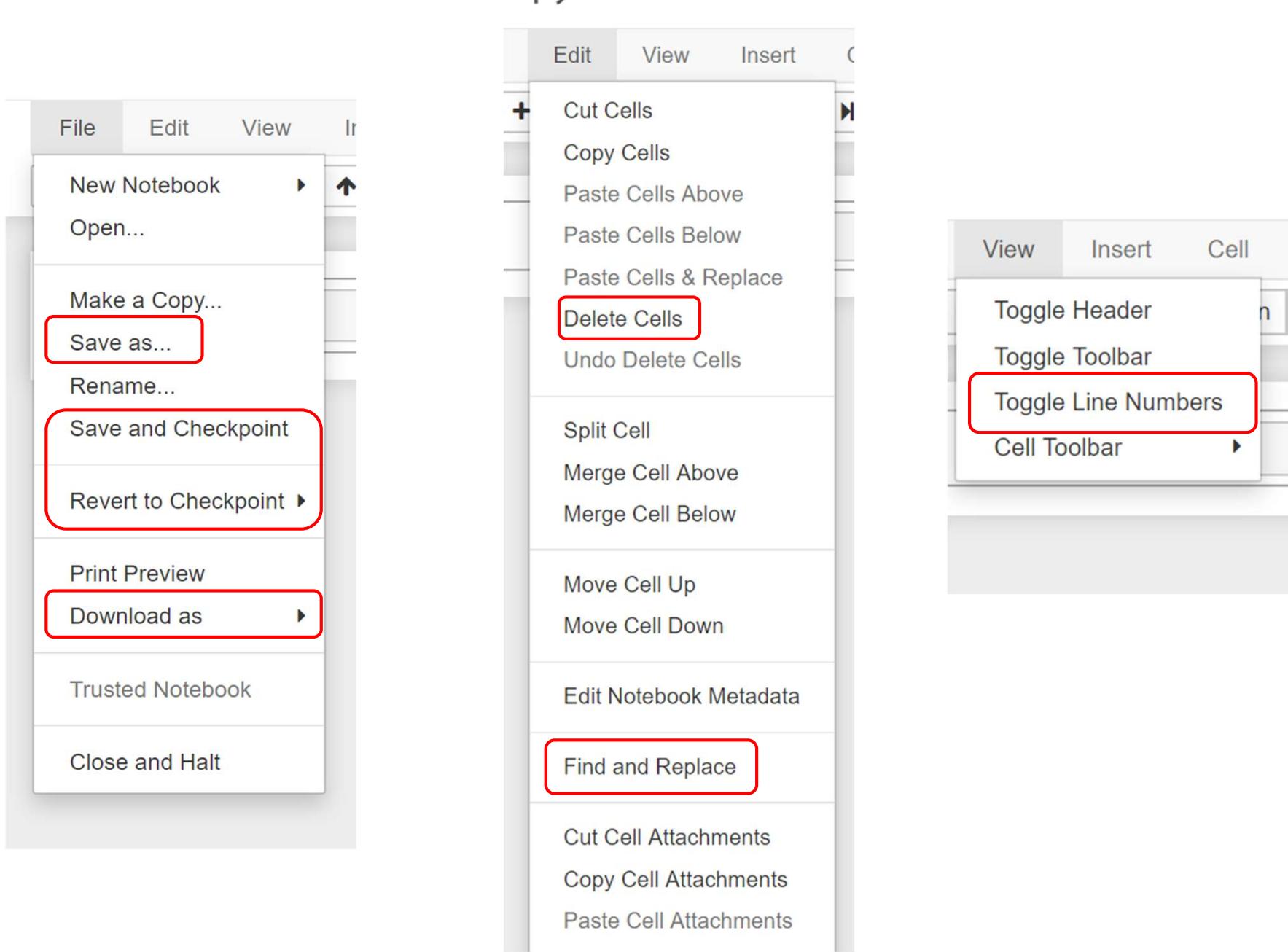
To access the notebook, open this file in a browser:
  file:///C:/Users/trimu/AppData/Roaming/jupyter/runtime/nbserver-27684-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=cde0891905a6a5459965881382c153b4f9f5593f1cab1221
```

```
C:\Users\whoami>jupyter notebook list
Currently running servers:
http://localhost:8888/?token=9381e1ec86f4ace207780ac683ab440fde7d7a4ecf65af0c ::
```

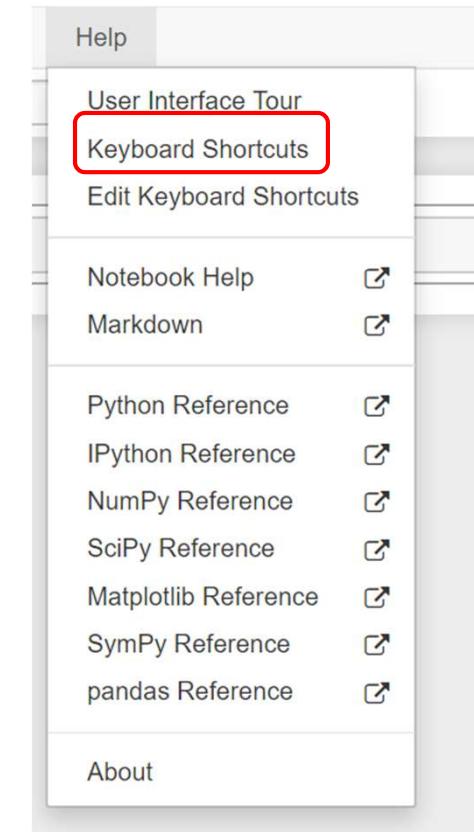
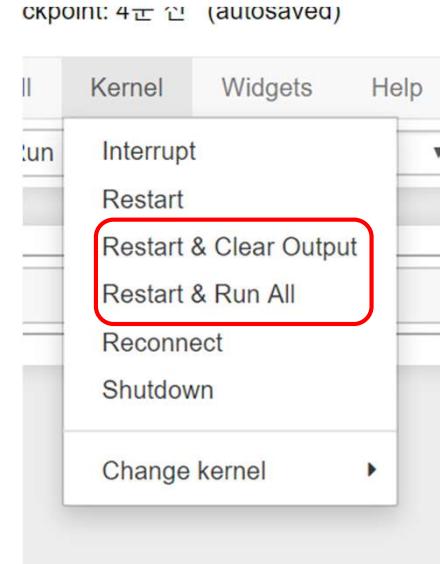
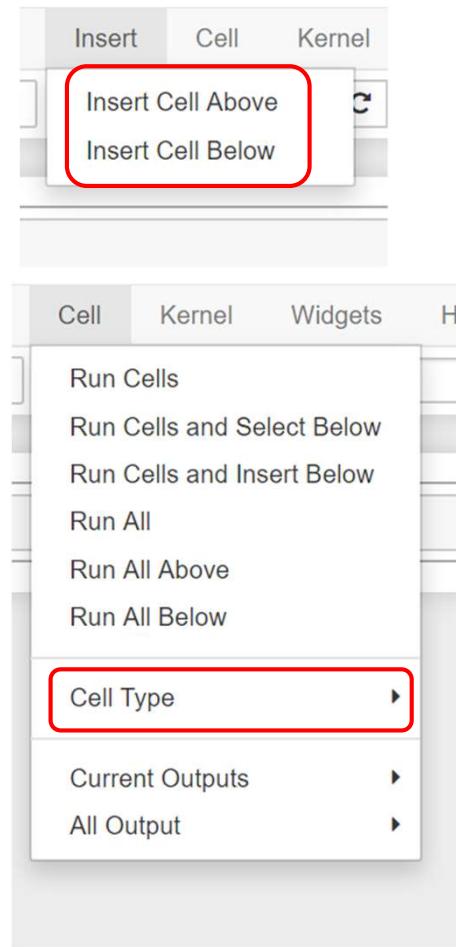
# Jupyter Notebook 사용방법



# Jupyter Notebook 사용방법



# Jupyter Notebook 사용방법



\* Login 비밀번호 요구할 경우 사용자 계정의 `~/.jupyter/jupyter_notebook_config.py` 삭제

# git Installation

<https://git-scm.com/download/win>

The screenshot shows the official Git website at <https://git-scm.com>. The main navigation menu includes links for About, Documentation, Downloads (selected), and Community. A sidebar on the left promotes the "Pro Git book". The central content area is titled "Downloading Git". It displays a large downward arrow icon and the message "Your download is starting...". Below this, it states: "You are downloading the latest (2.21.0) 64-bit version of Git for Windows. This is the most recent maintained build. It was released 2 months ago, on 2019-02-26." It also provides links for other download options like "Git for Windows Setup", "32-bit Git for Windows Setup", and "64-bit Git for Windows Setup". The "64-bit Git for Windows Setup" link is highlighted with a red border. At the bottom, it says: "The current source code release is version 2.21.0. If you want the newer version, you can build it from the [source code](#)".

# Github Repository

<https://github.com/ironmanciti/machineLearningBasic>

The screenshot shows the GitHub repository page for 'ironmanciti / machineLearningBasic'. The page includes the repository name, statistics (1 commit, 1 branch, 0 releases, 1 contributor), and options to clone or download the repository. The repository contains a single file, README.md, which has been updated once. The file content is visible in the main area.

ironmanciti / machineLearningBasic

Watch 0 | Star 0 | Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

파이썬을 활용한 머신러닝 입문 과정

Edit

Manage topics

1 commit 1 branch 0 releases 1 contributor

Branch: master ▾ New pull request Create new file Upload files Find File Clone or download ▾

ironmanciti Initial commit

README.md Initial commit

README.md

Clone with HTTPS Use SSH  
Use Git or checkout with SVN using the web URL.  
<https://github.com/ironmanciti/machineLe>

Open in Desktop Download ZIP

## machineLearningBasic

파이썬을 활용한 머신러닝 입문 과정

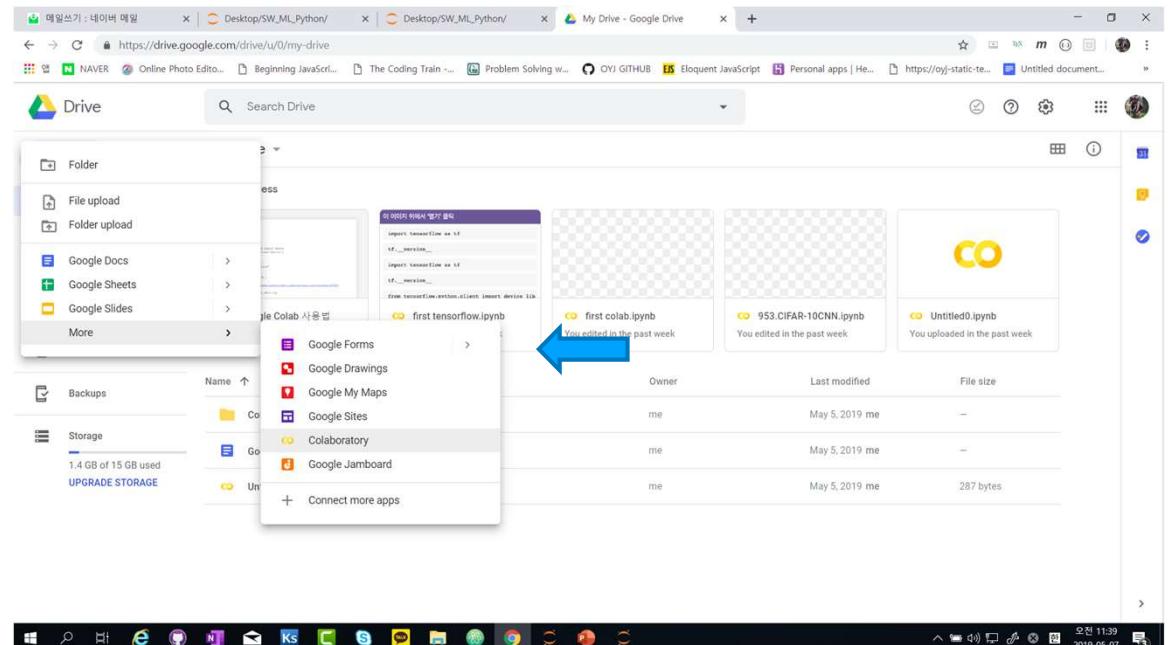
# Google Colaboratory

- Free GPU 제공
- Google Drive 와 연동
- Jupyter Notebook 환경
- Deep Learning beginner 를 위한 최적의 환경
- 각종 snippet 제공

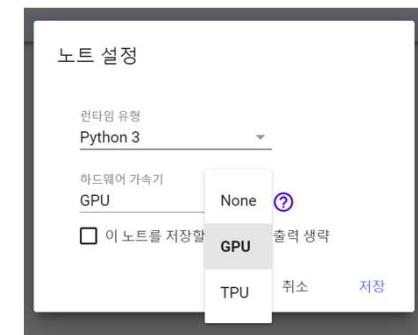
# Google Colaboratory 사용하기

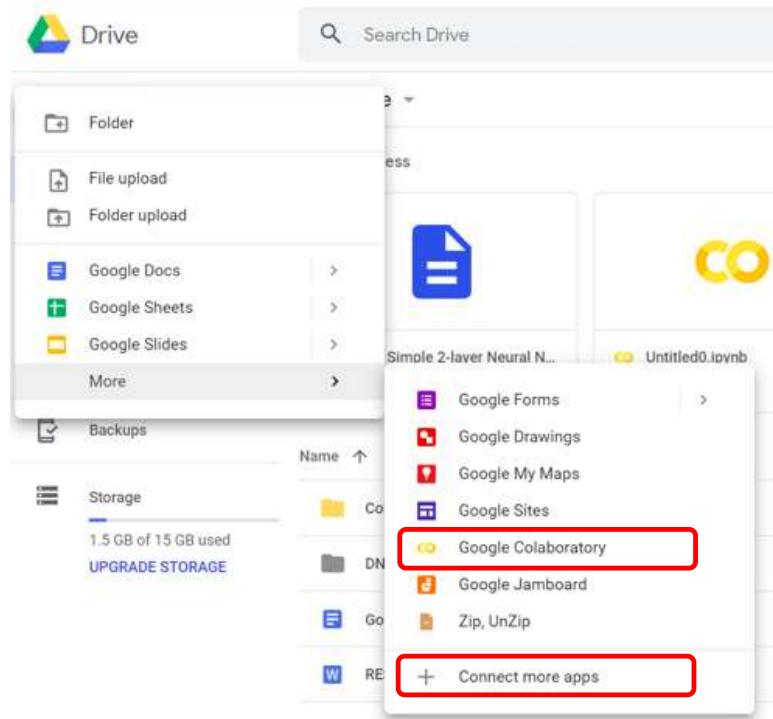
- Google drive

→ Colaboratory 연결



- 런타임 → 런타임 유형변경 → GPU 선택





보이지 않을 경우 Colab 추가

- gpu check

```
[1] from tensorflow.python.client import device_lib  
device_lib.list_local_devices()
```

↳ The default version of TensorFlow in Colab will soon switch to TensorFlow 2.x.  
We recommend you [upgrade](#) now or ensure your notebook will continue to use TensorFlow 1.x via the  
`%tensorflow_version 1.x` magic: [more info](#).

```
[name: "/device:CPU:0"  
device_type: "CPU"  
memory_limit: 268435456  
locality {  
}  
incarnation: 11925876377786261971, name: "/device:XLA_CPU:0"  
device_type: "XLA_CPU"  
memory_limit: 17179869184  
locality {  
}  
incarnation: 5168198067976361036  
physical_device_desc: "device: XLA_CPU device"]
```

CPU

```
incarnation: 1135439848827926988  
physical_device_desc: "device: XLA_GPU device", name: "/device:GPU:0"  
device_type: "GPU"  
memory_limit: 11330115994  
locality {  
bus_id: 1  
links {  
}  
}  
incarnation: 18206703667281236574  
physical_device_desc: "device: 0, name: Tesla K80, pci bus id: 0000:00:04.0, compute capability: 3.7"]
```

GPU

- Google Drive & Colab 연결

## 1) from google.colab import auth

```
from google.colab import drive  
drive.mount('/content/gdrive')
```

... Go to this URL in a browser: [https://accounts.google.com/o/oauth2/auth?client\\_id=947318989803-6bn6qk](https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk)

Enter your authorization code:

```
[5] !cd gdrive/My\ Drive; ls -al;
```

```
total 20961  
-rw----- 1 root root 21433196 Oct 29 15:11 _2016_01_BS1_20170524.csv  
drwx----- 2 root root 4096 Oct 29 14:44 FinancialSheet  
-rw----- 1 root root 151 Sep 21 18:57 'Google Colab 사용법.gdoc'  
-rw----- 1 root root 22593 Aug 26 20:53 RESUME.docx  
-rw----- 1 root root 26 Oct 29 20:16 'Sample upload.txt'  
-rw----- 1 root root 616 Oct 29 14:24 test.ipynb  
-rw----- 1 root root 269 Oct 29 14:49 Untitled  
-rw----- 1 root root 297 Oct 30 03:10 Untitled0.ipynb  
-rw----- 1 root root 269 Oct 30 03:16 'Untitled (1)'
```

## 2) Read csv file

```
[8] import pandas as pd  
pd.read_csv("/content/gdrive/My Drive/FinancialSheet/2015_4Q_FS/_2015_04_CF2_20160601.csv")
```

- github repository clone (small file < 25MB)

```
[3] !git clone https://github.com/ironmanciti/machineLearningBasic.git
```

```
↳ Cloning into 'machineLearningBasic'...
remote: Enumerating objects: 1142, done.
remote: Counting objects: 100% (1142/1142), done.
remote: Compressing objects: 100% (1095/1095), done.
remote: Total 1142 (delta 59), reused 1118 (delta 43), pack-reused 0
Receiving objects: 100% (1142/1142), 31.53 MiB | 10.27 MiB/s, done.
Resolving deltas: 100% (59/59), done.
```

```
[4] ls
```

```
↳ machineLearningBasic/ sample_data/
```

```
[5] cd machineLearningBasic/
```

```
↳ /content/machineLearningBasic
```

```
[6] ls
```

```
↳ 93.Mnist_Dense.ipynb __pycache__/ README.md
```

```
[7] cat README.md
```

```
↳ # machineLearningBasic
## 파이썬을 활용한 머신러닝 입문 과정 - 2019
```

## • CSV file read to pandas (Local file / web path)

```
[13] from google.colab import files  
  
uploaded = files.upload()  
  
↳ 파일 선택 clean_df.csv  
• clean_df.csv(application/vnd.ms-excel) - 42240 bytes, last modified: 2018. 12. 23. - 100% done  
Saving clean_df.csv to clean_df (1).csv
```

```
[15] uploaded  
  
↳ {'clean_df.csv': b'symboling,normalized-losses,make,num-of-doors,body-style,drive-wheels,engine-location,wl'
```

```
[17] import pandas as pd  
  
pd.read_csv("clean_df.csv")  
  
↳ Unnamed:  
      0   symboling   normalized-  
      losses     make    num-  
           of-  
           doors   body-  
           style   drive-  
           wheels  engine-  
           location  wheel-  
           base    length
```

```
[18] !wget https://storage.googleapis.com/applied-dl/heart.csv  
  
↳ --2019-10-29 07:46:14-- https://storage.googleapis.com/applied-dl/heart.csv  
Resolving storage.googleapis.com (storage.googleapis.com)... 108.177.97.128, 2404:6800:4008:c01::80  
Connecting to storage.googleapis.com (storage.googleapis.com)|108.177.97.128|:443... connected.  
HTTP request sent, awaiting response... 200 OK  
Length: 13273 (13K) [application/octet-stream]  
Saving to: 'heart.csv'  
  
heart.csv      100%[=====] 12.96K --.-KB/s  in 0s  
  
2019-10-29 07:46:14 (86.0 MB/s) - 'heart.csv' saved [13273/13273]
```

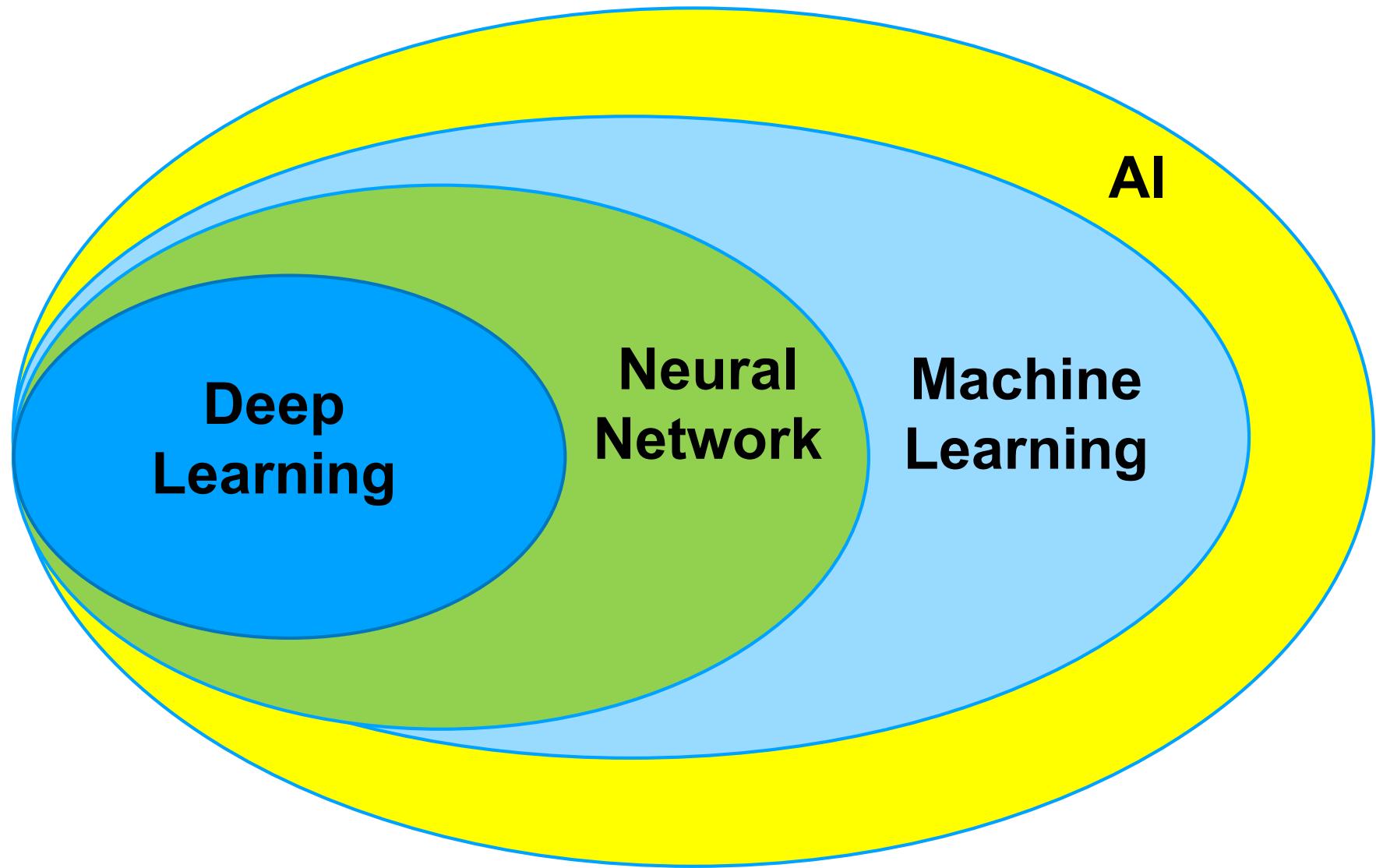
Local에서 read

Web에서 read

# Machine Learning

## 개요

# AI vs Machine Learning vs Deep Learning



# History of Machine Learning

- **탄생 [1950년대]**

1958년 코넬대 심리학자 프랭크 로센블래트가 인간의 뇌신경을 본떠 Perceptron 고안. 신경망 기반 인공지능 연구의 부흥기 시작

- **AI의 첫번째 암흑기 [1970년대]**

Marvin Minsky 가 Perceptron 은 XOR 문제를 해결할 수 없음을 수학적으로 증명. 인공지능에 대한 대규모 연구 지원 중단

- **중흥기 [1980년대]**

산업계에 전문가 시스템(Expert System)이 도입되며 본격적으로 확산

- **AI 의 두번째 암흑기 [1987 – 1993]**

투자대비 효용성의 한계가 노출. 슈퍼컴퓨터와 시뮬레이션 분야로  
연구방향을 전환

\* Jeffery Hinton (Toronto 대학) back-propagation algorithm 개발  
→ 민스키가 틀렸다는 것을 증명

- **IBM Deep Blue 가 Garry Kasparov 에 승리 - 1996**

- **Google Brain 이 최초로 인간 얼굴 인식 - 2012**

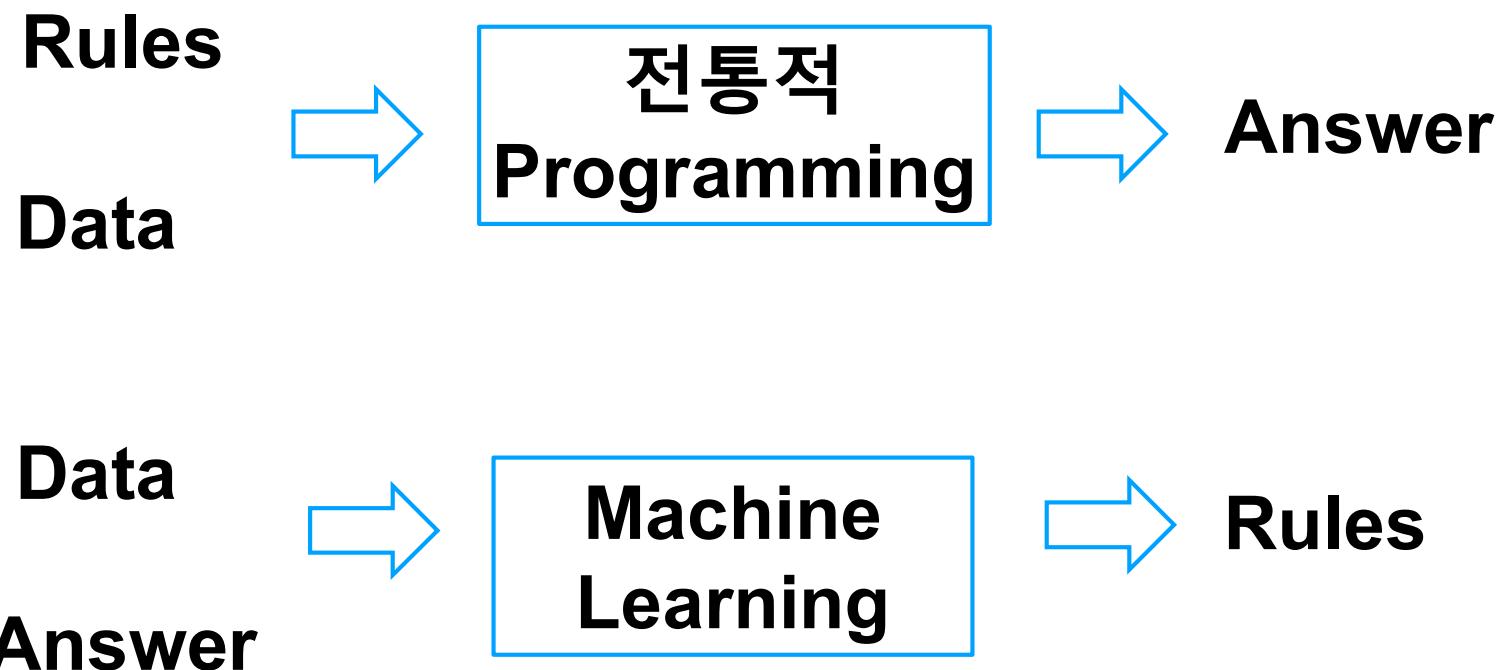
- **~ 상업적 대 폭발기**

- **Alphago 이세돌에 승리 – 2016**

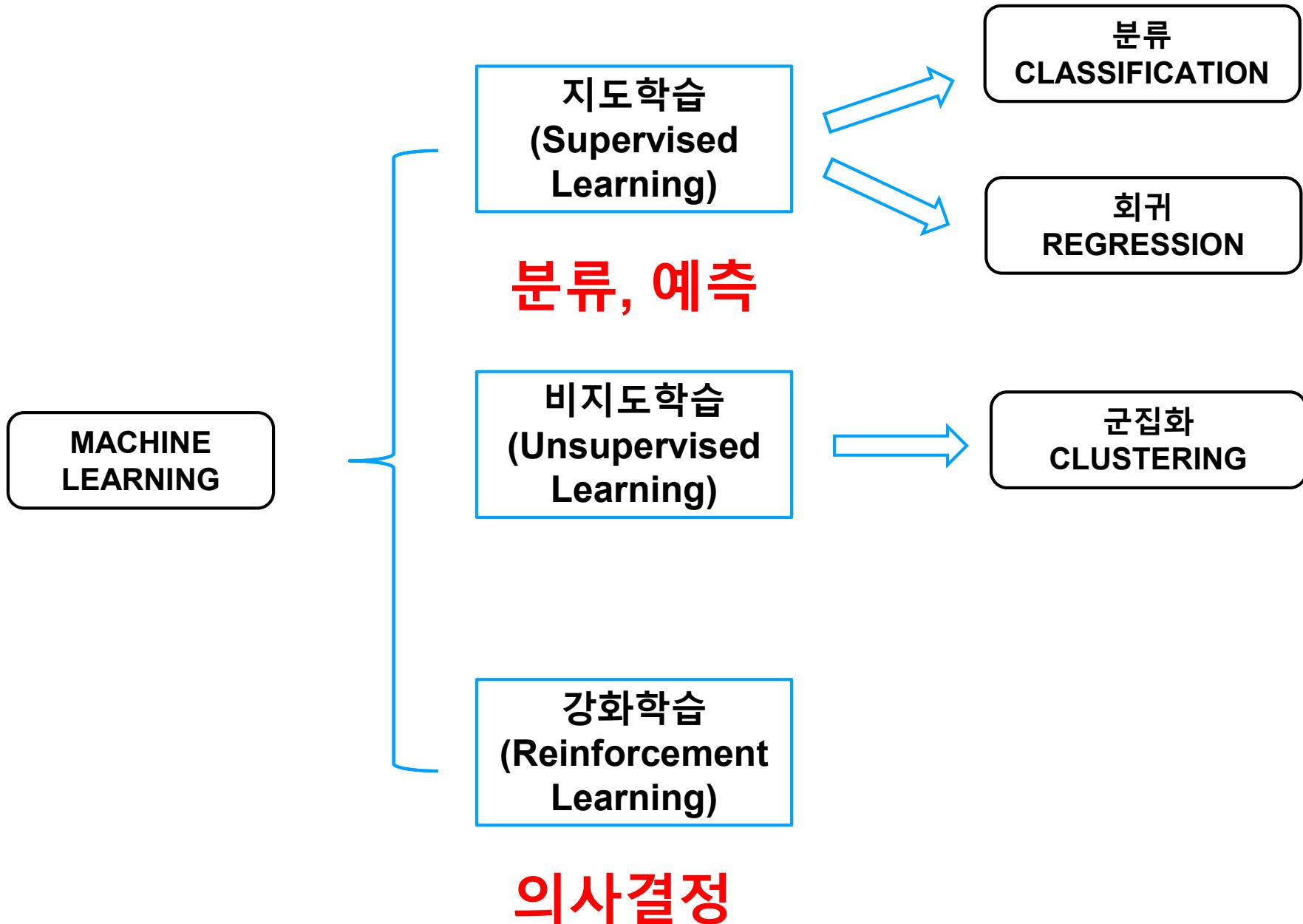
# 전통적 Programming

vs

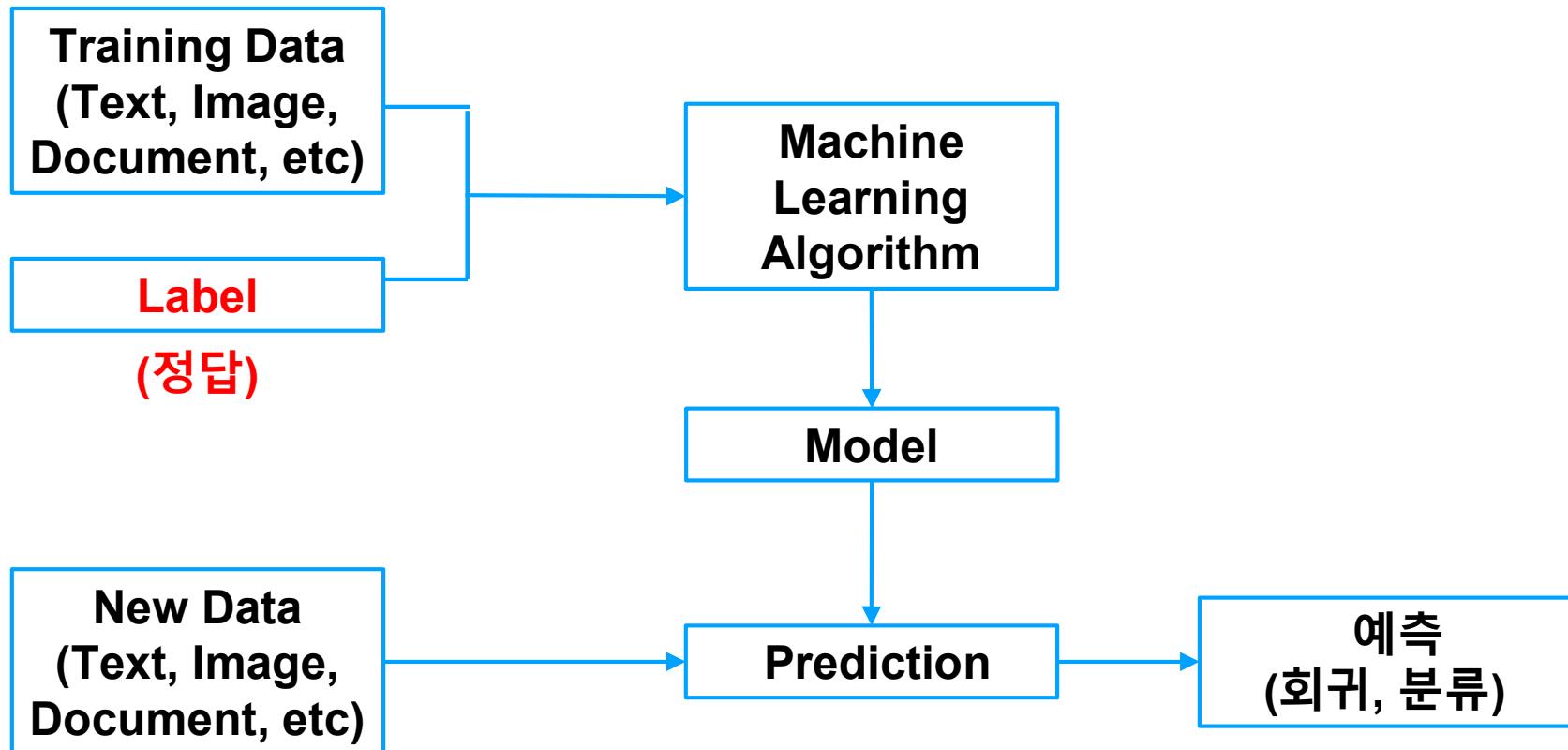
# Machine Learning



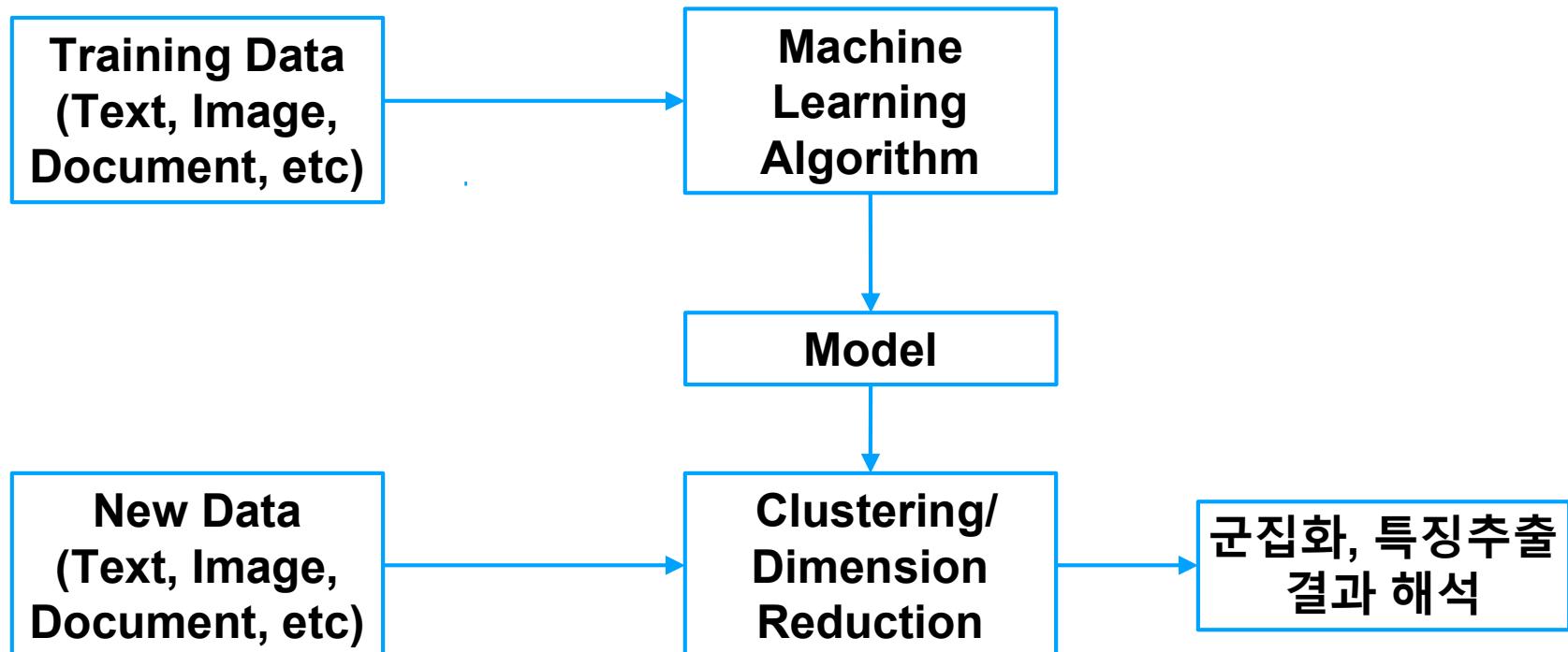
# Machine Learning 의 종류



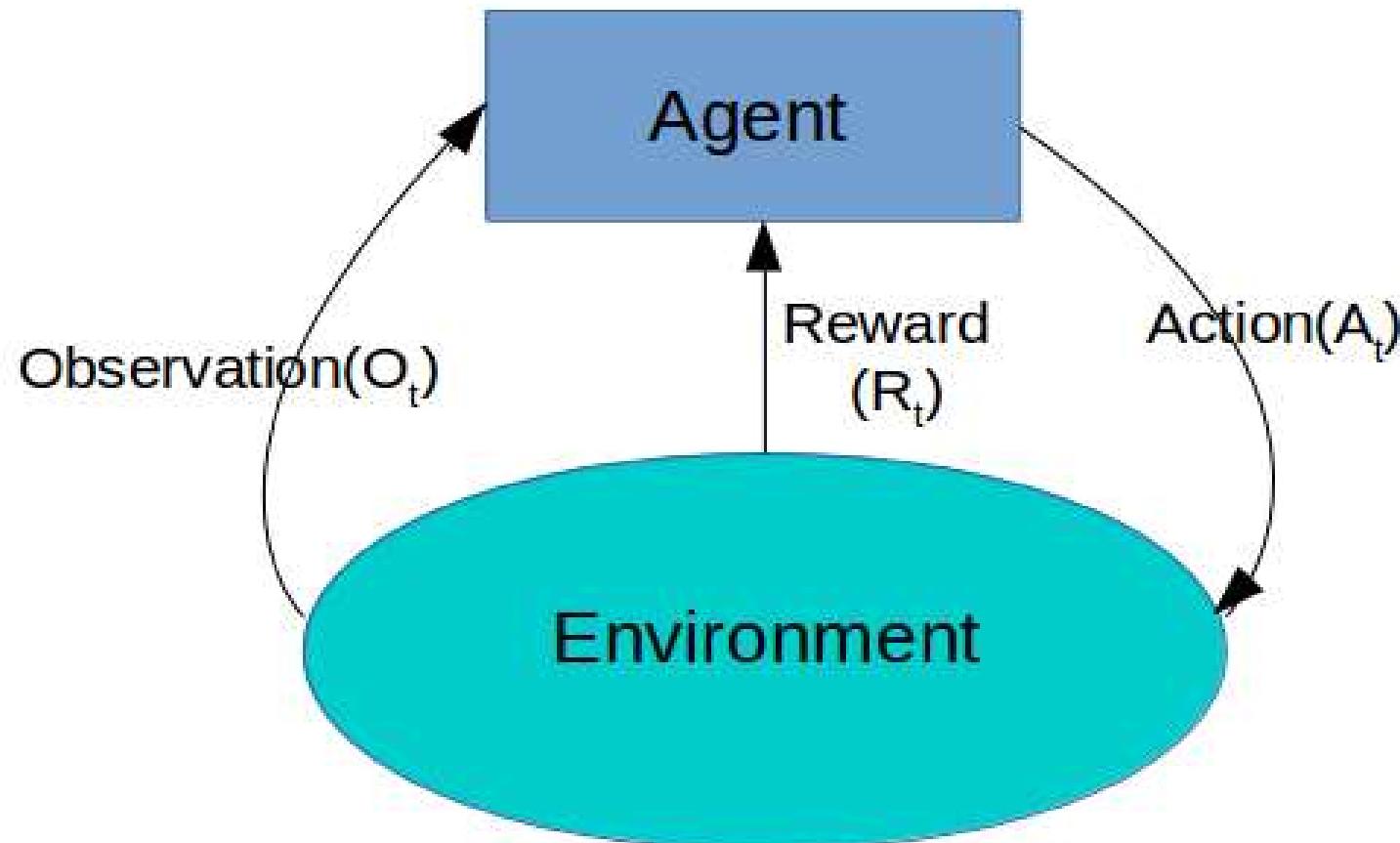
# Supervised Learning (지도학습)



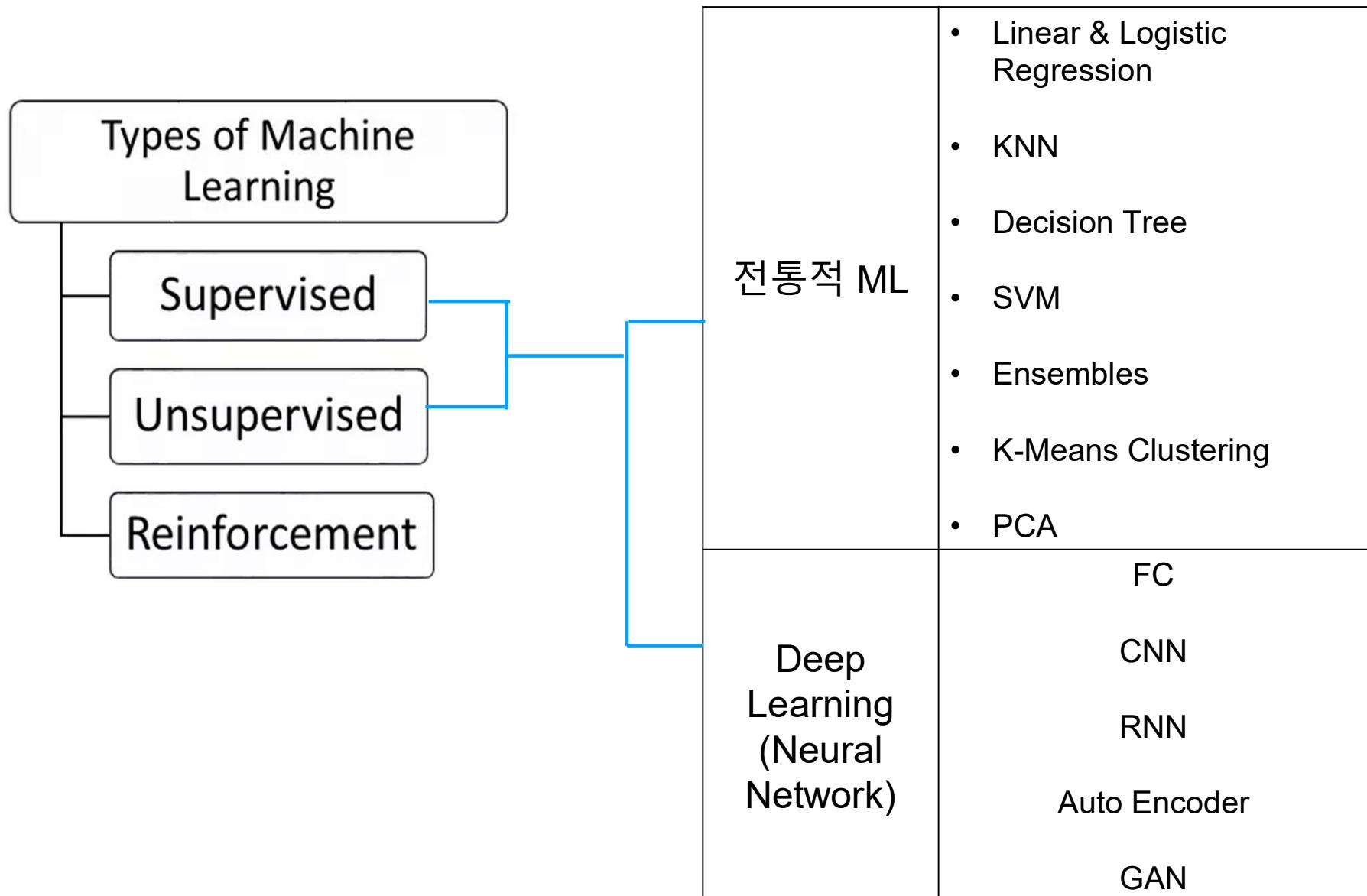
# Unsupervised Learning (비지도학습)



# Reinforcement Learning (강화학습)



# Machine Learning 기법의 종류



# 전통적 ML의 기법

종류	용도
Linear / Logistic Regression	선형회귀/분류
KNN (K-Nearest Neighbor)	분류/회귀
Decision Tree (결정나무)	분류
SVM (Support Vector Machine)	분류, 회귀
Ensemble (Random Forest, XGBoost, etc)	분류, 회귀
K-Means Clustering (K-평균 군집화)	군집화
PCA (Principal Component Analysis)	차원 축소

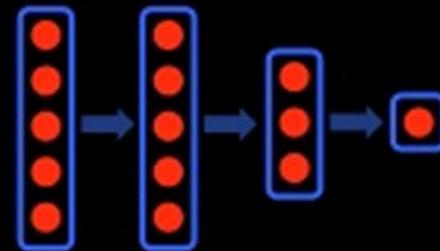
# Neural Network 의 종류

종류	용도
FC (Fully Connected Neural Network)	분류, 회귀
CNN (Convolutional Neural Network)	Image 인식
RNN (Recurrent Neural Network)	시계열 인식, 자동번역, 감성분석 등
AE (Auto Encoder)	비지도 학습, 차원축소
GAN (Generative Adversarial Nets)	적대적 생성모델, 이미지 위조

## Why is Deep Learning taking off?



Engine



Large neural networks



Labeled data  
( $x, y$  pairs)

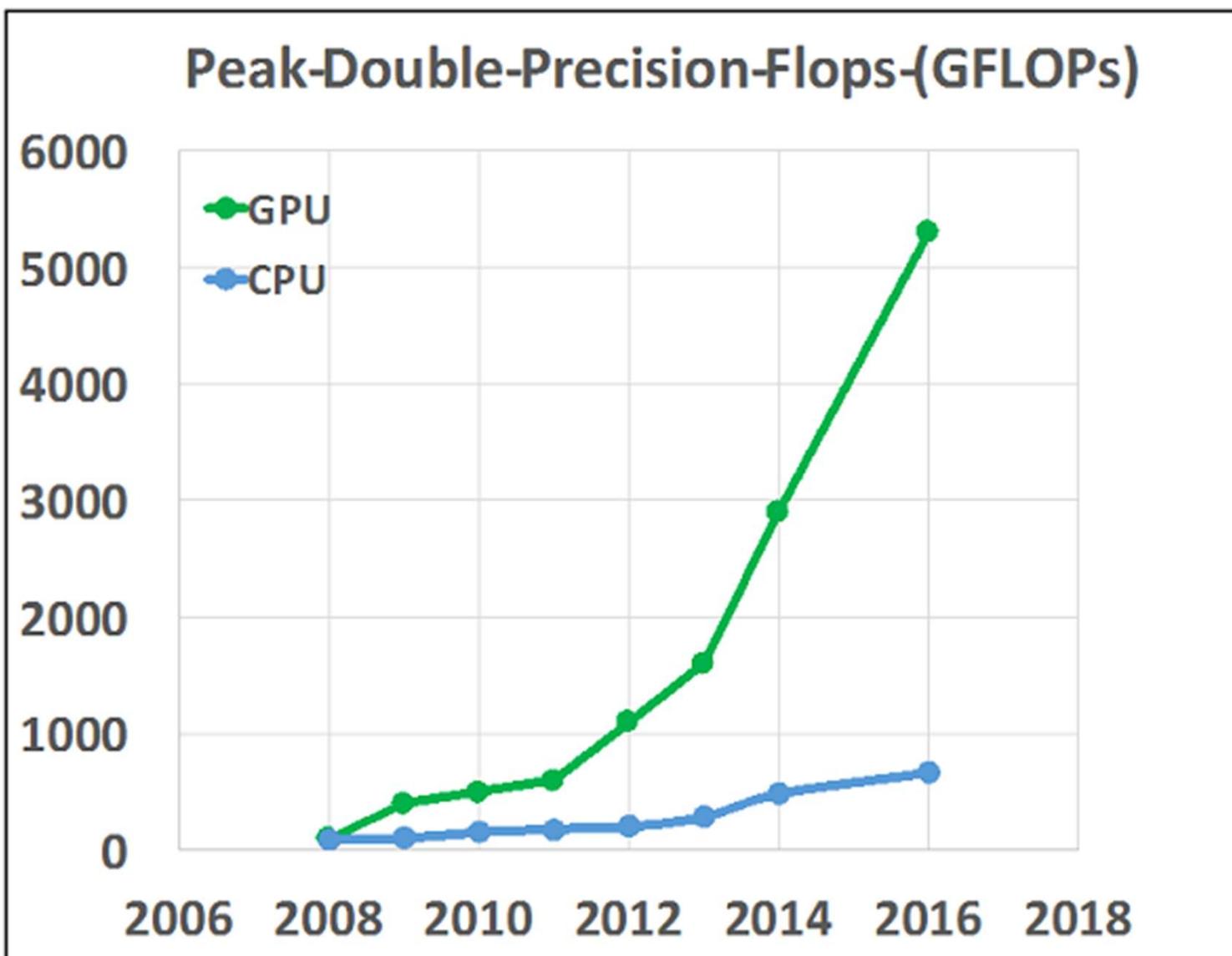
Baidu Research

Andrew Ng

**Rocket Engine : NVIDIA + Deep Learning Algorithm**

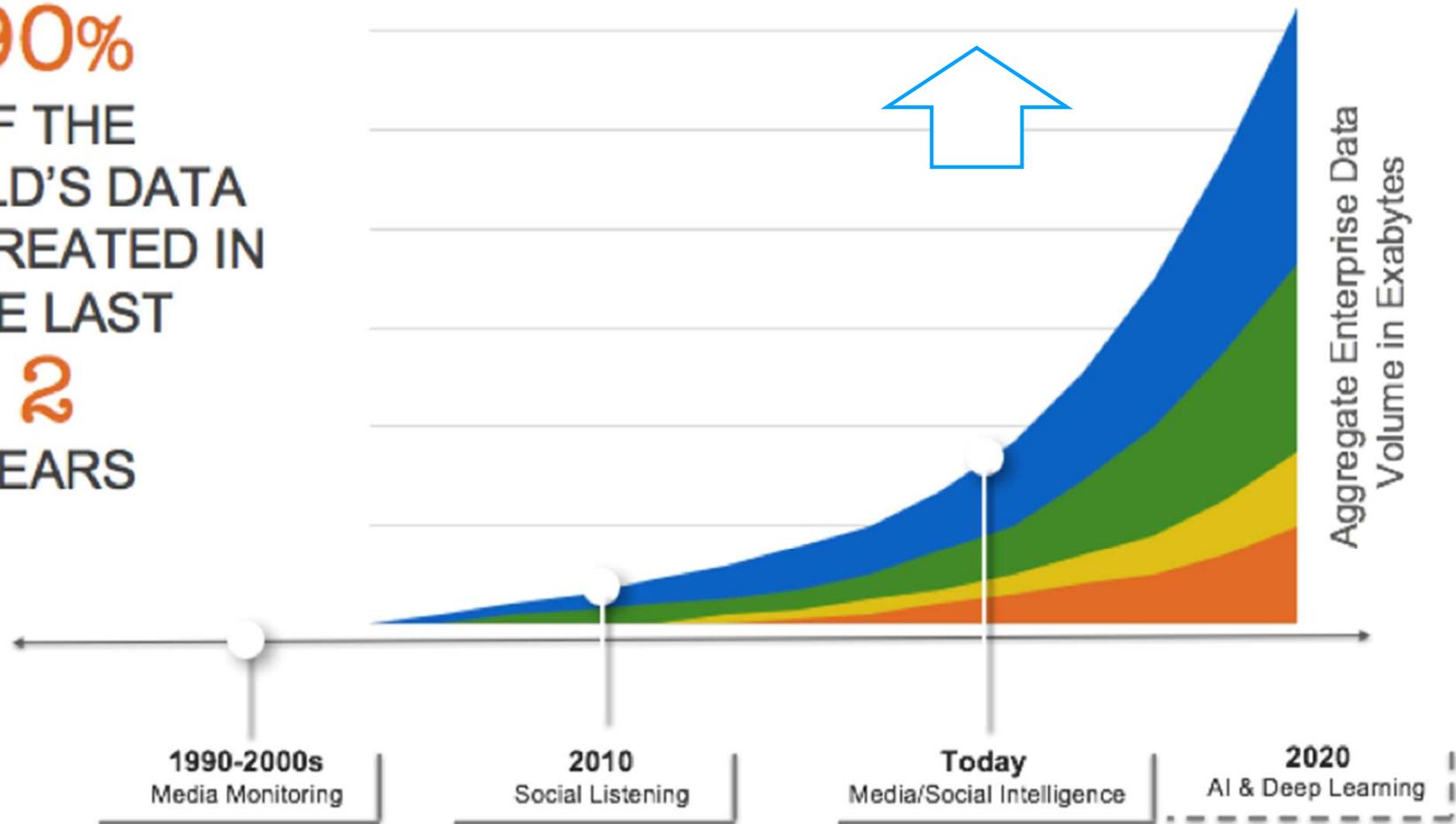
**Fuel : Data (25,000 pictures for cat)**

# CPU vs GPU Performance

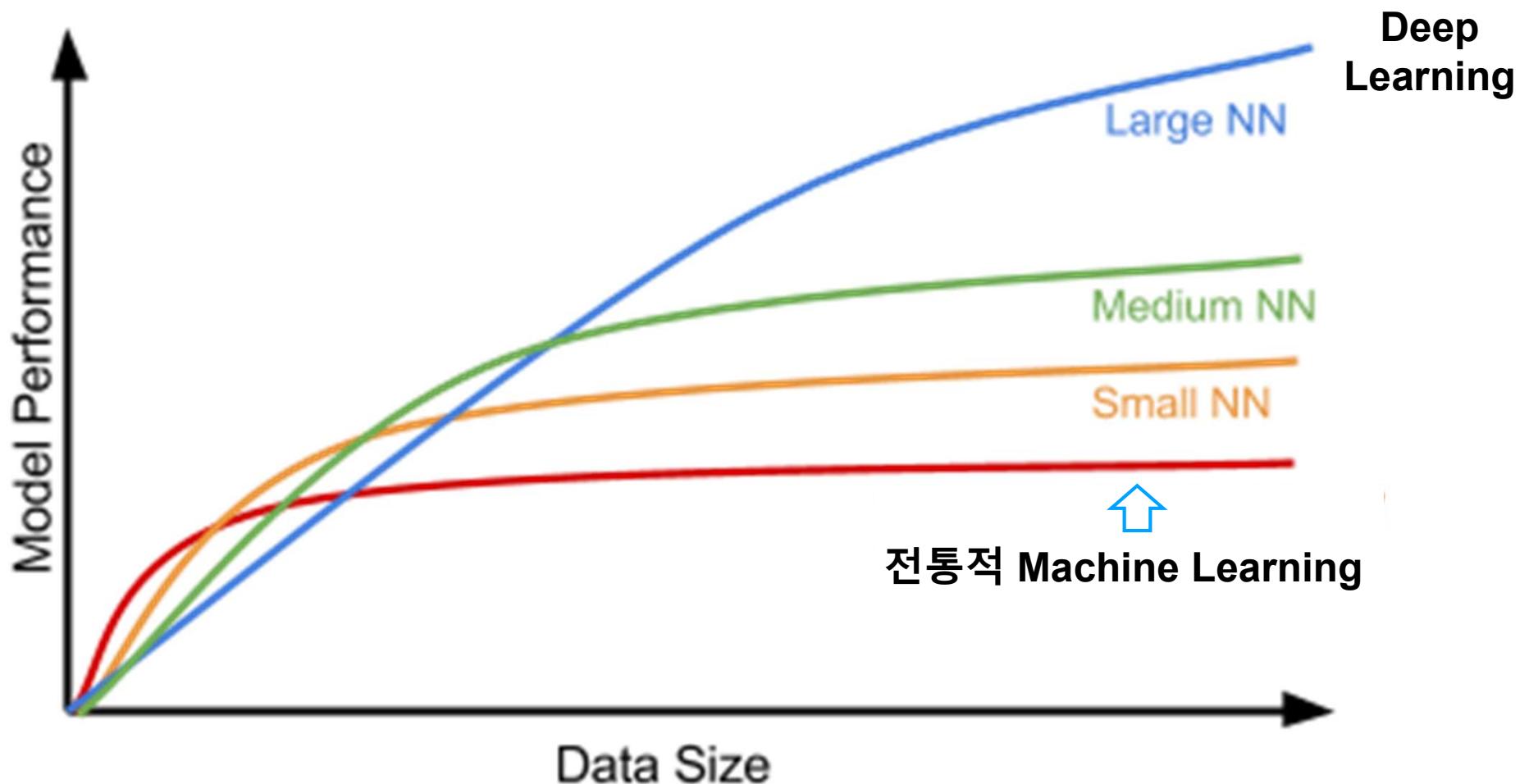


## 지도학습 모델의 획기적 성공 이유

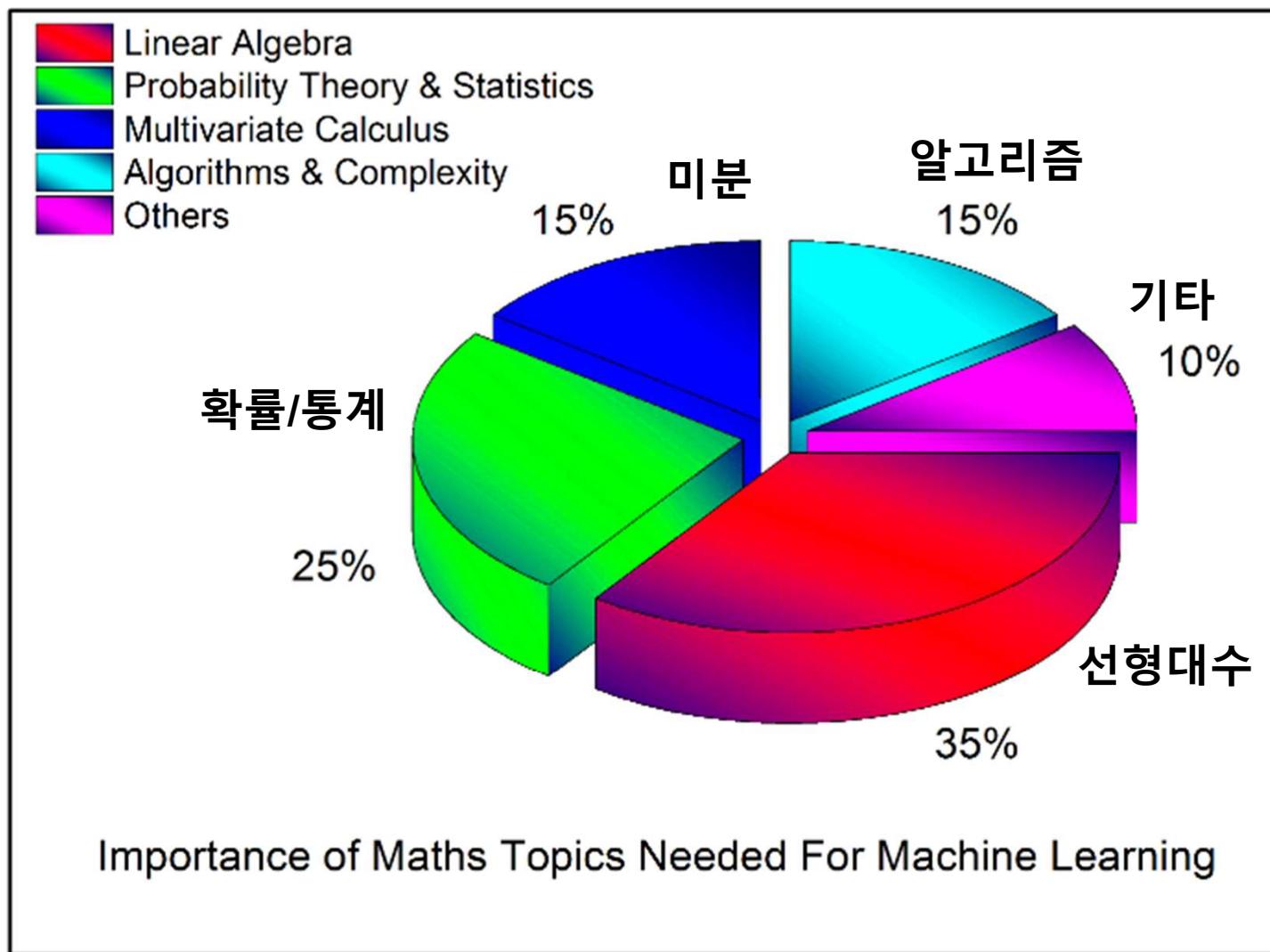
90%  
OF THE  
WORLD'S DATA  
WAS CREATED IN  
THE LAST  
2  
YEARS



# 전통적 Machine Learning vs. Deep Neural Network



# Machine Learning 학습에 필요한 수학 지식



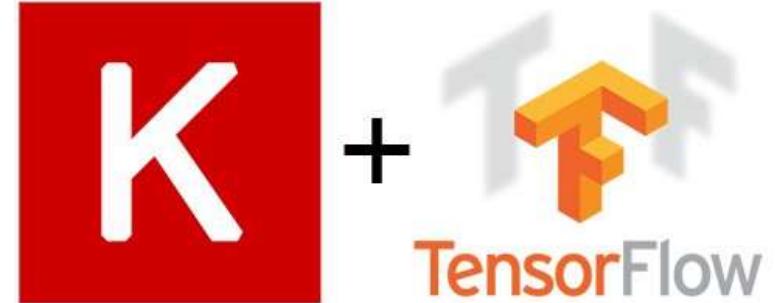
# Machine Learning



전통적 Machine Learning Tool:

- 벤치마크용 데이터셋 예제
- 데이터 전처리(preprocessing)
- 지도 학습(Supervised learning)
- 비지도 학습(Unsupervised learning)
- 모형 평가 및 선택 (evaluation and selection)

# Deep Learning



Deep Learning Tool:

- Pre-train model, Sample Dataset 제공
- Deep Learning 에 필요한 각종 함수 제공
- GPU support
- 각종 language 지원 API 제공

# 과정 SCOPE

- Crash Course – Numpy, Pandas, Matplotlib
- Linear Regression (선형회귀)
- KNN (K-Nearest Neighbor)
- Decision Tree
- Logistic Regression
- SVM (Support Vector Machine)
- Random Forest

- K-Means Clustering
- Tensorflow/Keras
- Simple Neural Network
- CNN
- RNN
- GAN(Optional)
- 실습문제

# Crash Course

1. Numpy & Linear Algebra
2. Pandas
3. Matplotlib
4. Feature Scaling

# Numpy

- numpy array
  - 1 차원 - vector, 2 차원 – matrix, 3 차원 이상 - tensor
  - rank – array 의 dimension (차원)
  - shape – 각 dimension 의 size
  - dtype – tensor 의 data type

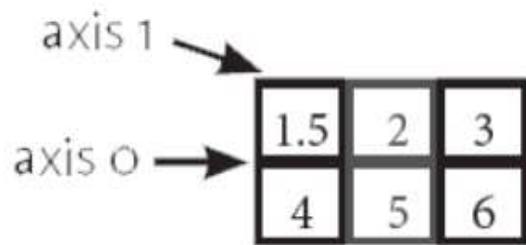
	Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} & \begin{bmatrix} 3 & 2 \\ 5 & 4 \end{bmatrix} \end{bmatrix}$

shape : (1,)    (2, 1)    (2, 2)    (2, 2, 2)

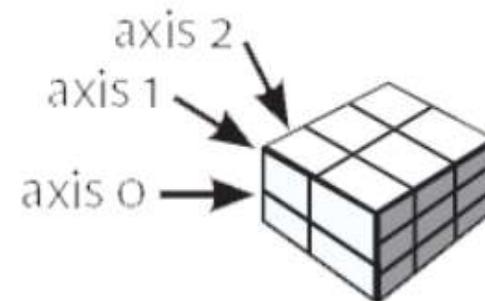
## 1D array



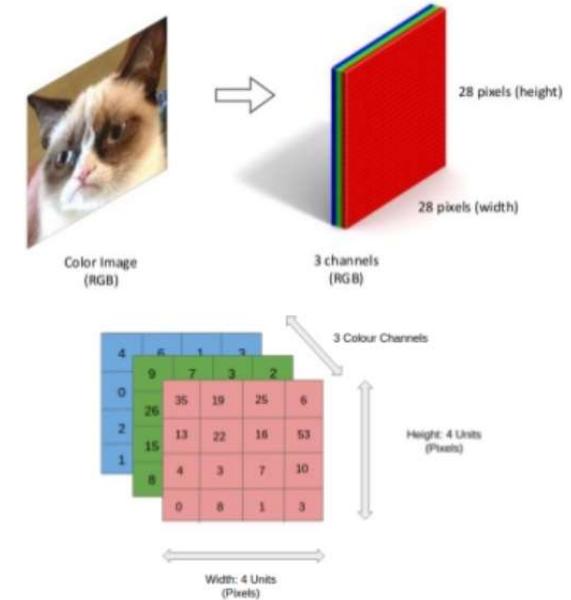
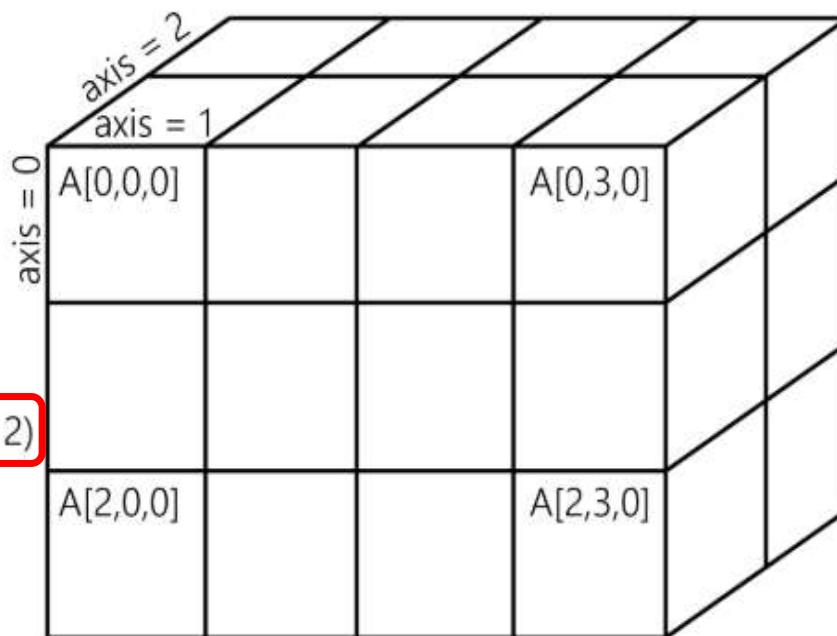
## 2D array



## 3D array



ndarray  
ndim = 3  
shape = (3, 4, 2)



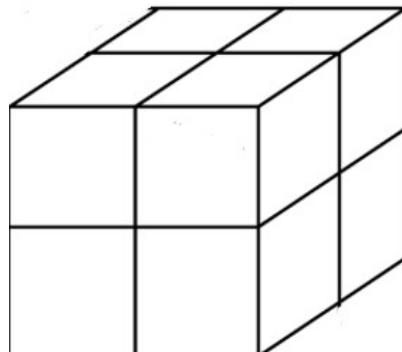
# Matrix 의 numpy 표현

8	5	3
1	2	9

`np.array([[8, 5, 3], [1, 2, 9]])`



`[[8 5 3]  
[1 2 9]]`



`np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])`

`[[[1 2]  
[3 4]]`

`[[[5 6]  
[7 8]]]`

# Joining & Slicing

- concatenate

```
[[1 , 2 ],  
 [10, 20]] + [[3 , 4 ],  
 [30, 40]]
```

```
[[1 , 2 ],  
 [10, 20],  
 [3 , 4 ],  
 [30, 40]]
```

**axis=0**

```
[[1 , 2 , 3 , 4 ],  
 [10, 20, 30, 40]]
```

**axis=1**

- slicing

```
a = np.array([[1,2], [3,4]])  
a[1,:]
```

```
[[1 2]  
 [3 4]]
```

```
[[3, 4], ]
```

# Broadcasting - 차원의 크기가 서로 다른 배열 간 산술연산

$$\begin{array}{c} \text{shape (4,)} \\ \boxed{1} \quad \boxed{2} \quad \boxed{3} \quad \boxed{4} \end{array} + \begin{array}{c} \text{scalar} \\ \boxed{1} \quad \boxed{1} \quad \boxed{1} \quad \boxed{1} \end{array} = \begin{array}{c} \text{shape (4,)} \\ \boxed{2} \quad \boxed{3} \quad \boxed{4} \quad \boxed{5} \end{array}$$

Broadcasting

$$\begin{array}{c} \text{shape (4,3)} \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline 9 & 10 & 11 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{shape (3,)} \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{shape (4,3)} \\ \begin{array}{|c|c|c|} \hline 0 & 2 & 4 \\ \hline 3 & 5 & 7 \\ \hline 6 & 8 & 10 \\ \hline 9 & 11 & 13 \\ \hline \end{array} \end{array}$$

Broadcasting

$$\begin{array}{c} \text{shape (4,3)} \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline 9 & 10 & 11 \\ \hline \end{array} \end{array} + \begin{array}{c} \text{shape (4,1)} \\ \begin{array}{|c|} \hline 0 \\ \hline 1 \\ \hline 2 \\ \hline 3 \\ \hline \end{array} \end{array} = \begin{array}{c} \text{shape (4,3)} \\ \begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 4 & 5 & 6 \\ \hline 8 & 9 & 10 \\ \hline 12 & 13 & 14 \\ \hline \end{array} \end{array}$$

Broadcasting

# numpy array 간의 연산

- `np.add(a, b)` – 두개의 tensor 를 element-wise 더함

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

- `np.subtract(a, b)`, `np.multiply(a, b)`, `np.divide(a, b)`
- `np.matMul(a, b)` - dot product of two matrices

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \times \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = [1 * 1 + 2 * 3 \quad 1 * 2 + 2 * 4] = [7 \quad 10]$$

shape :  $(1, 2) \times (2, 2) \rightarrow (1, 2)$

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

# Transpose, Reshape

- np.transpose() – row 와 column 을 전치

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}. \text{transpose}() \rightarrow \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

- np.reshape([m, n])

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}. \text{reshape}([3, 2]) \rightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

# Pandas

- Dataframe - R 을 모방하여 구현
- Tabular 형식의 Data 처리 (Excel 대체 가능)

	A	B	C	D	E	F	G	H	I
1	Order Date	OrderID	Salesperson	UK Units	UK Order Amt	USA Units	USA Order Amt	Total Units	Total Order Amt
2	1/01/2011	10392	Fuller			13	1440	13	1440
3	2/01/2011	10397	Gloucester	17	716.72			17	716.72
4	2/01/2011	10771	Bromley	18	344			18	344
5	3/01/2011	10393	Finchley			16	2556.95	16	2556.95
6	3/01/2011	10394	Finchley			10	442	10	442
7	3/01/2011	10395	Gillingham	9	2122.92			9	2122.92
8	6/01/2011	10396	Finchley			7	1903.8	7	1903.8
9	8/01/2011	10399	Callahan			17	1765.6	17	1765.6
10	8/01/2011	10404	Fuller			7	1591.25	7	1591.25
11	9/01/2011	10398	Fuller			11	2505.6	11	2505.6
12	9/01/2011	10403	Coghill	18	855.01			18	855.01
13	10/01/2011	10401	Finchley			7	3868.6	7	3868.6

# Matplotlib



Version 3.1.1

Fork me on GitHub

[home](#) | [examples](#) | [tutorials](#) | [API](#) | [contents](#) » [User's Guide](#) »

[previous](#) | [next](#) | [modules](#) | [index](#)

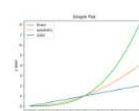
## Tutorials

This page contains more in-depth guides for using Matplotlib. It is broken up into beginner, intermediate, and advanced sections, as well as sections covering specific topics.

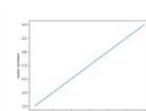
For shorter examples, see our [examples page](#). You can also find [external resources](#) and a [FAQ](#) in our [user guide](#).

### Introductory

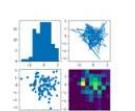
These tutorials cover the basics of creating visualizations with Matplotlib, as well as some best-practices in using the package effectively.



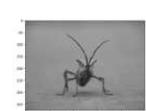
[Usage Guide](#)



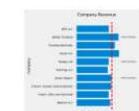
[Pyplot tutorial](#)



[Sample plots in Matplotlib](#)



[Image tutorial](#)



[The Lifecycle of a Plot](#)



[Customizing Matplotlib with style sheets and rcParams](#)

### Quick search

Go

### Table of Contents

#### Tutorials

- [Introductory](#)
- [Intermediate](#)
- [Advanced](#)
- [Colors](#)
- [Text](#)
- [Toolkits](#)

#### Related Topics

##### Documentation overview

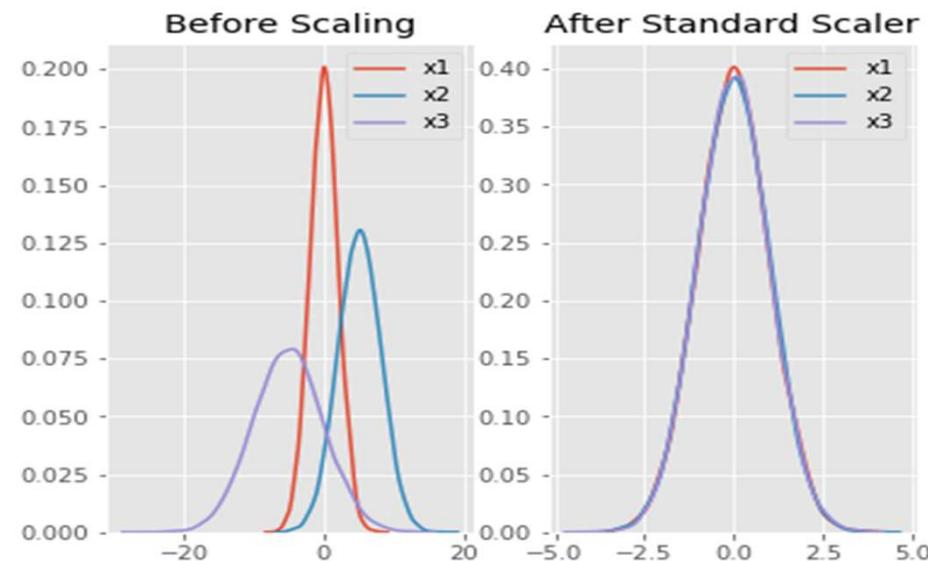
- [User's Guide](#)
  - Previous: [Installing](#)
  - Next: [Usage Guide](#)

[Show Page Source](#)

# Feature Scaling

- Raw data 를 전 처리하여 input data 의 구간을 표준화
- Standard Scaling :  
$$z = (x - \mu) / s \quad (\mu : 평균, s : 표준편차)$$
- Minmax Scaling :

$$x_{\text{new}} = \frac{x_i - \min(X)}{\max(x) - \min(X)}$$



# scikit-learn API 의 구조

- 모든 object 의 interface 가 일관성 있게 구성
  - fit() – dataset 의 data 를 기반으로 model parameter 추정
  - transform() – dataset 을 변환
  - fit\_transform() – fit() + transform()
  - predict() – 주어진 data 에 대한 예측
  - XXXX\_score() – 예측의 quality 측정

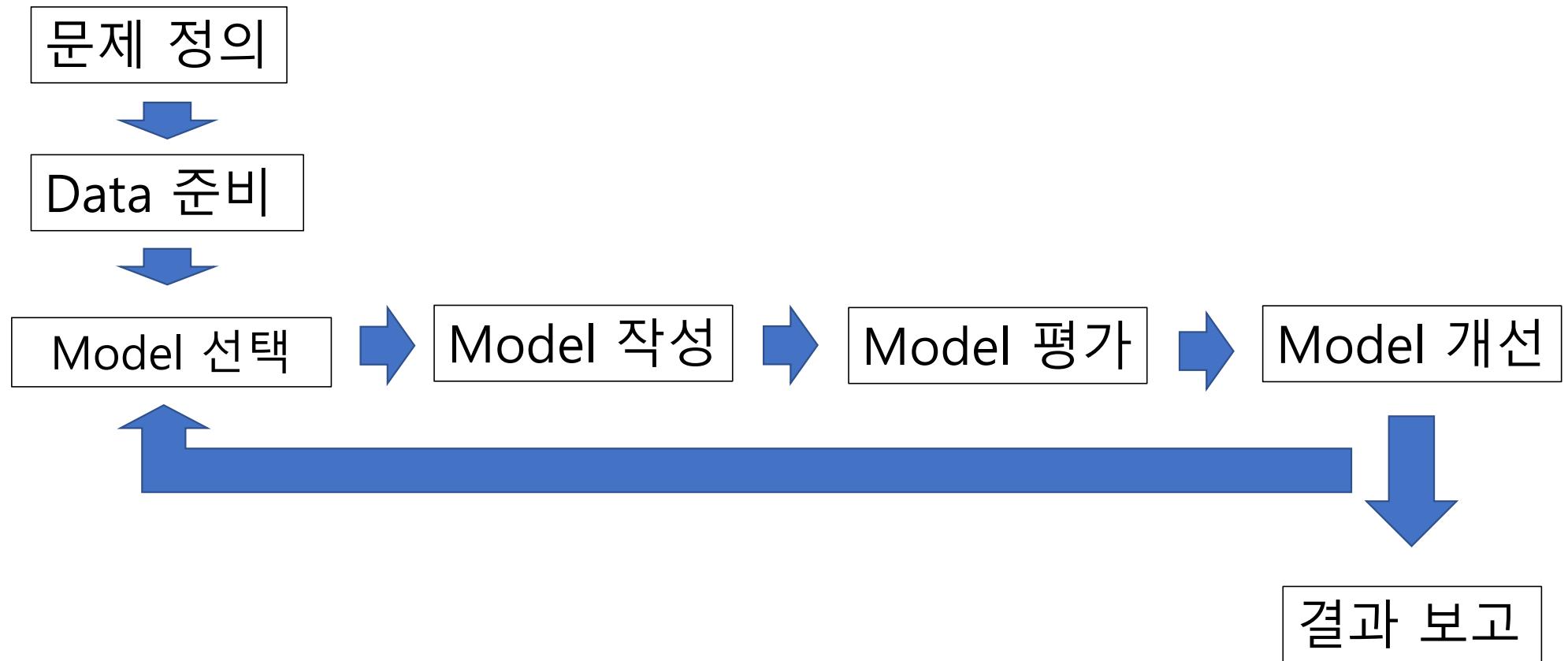
# 실습: sklearn 을 이용한 feature scaling (normalization)

- Simple Feature Scaling
- Standard Scaling
- MinMax Scaling

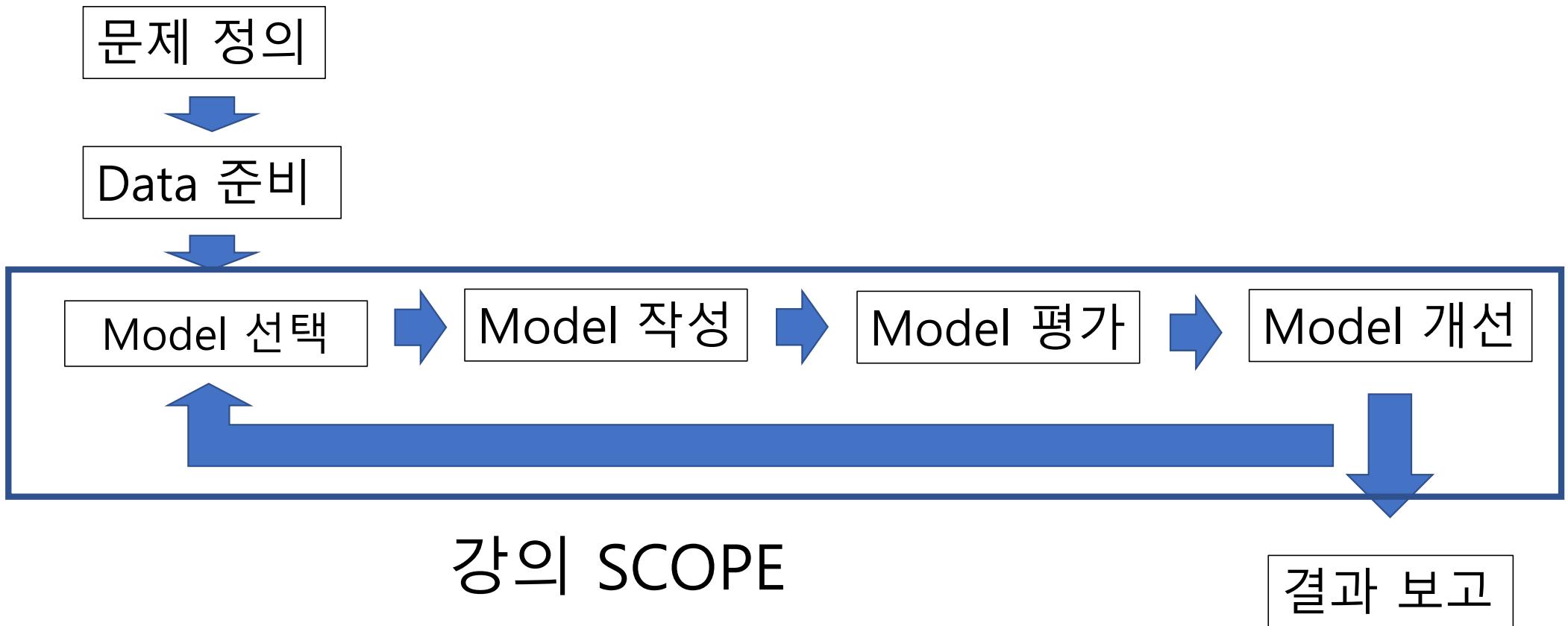
# Machine Learning

## End-to-End Process

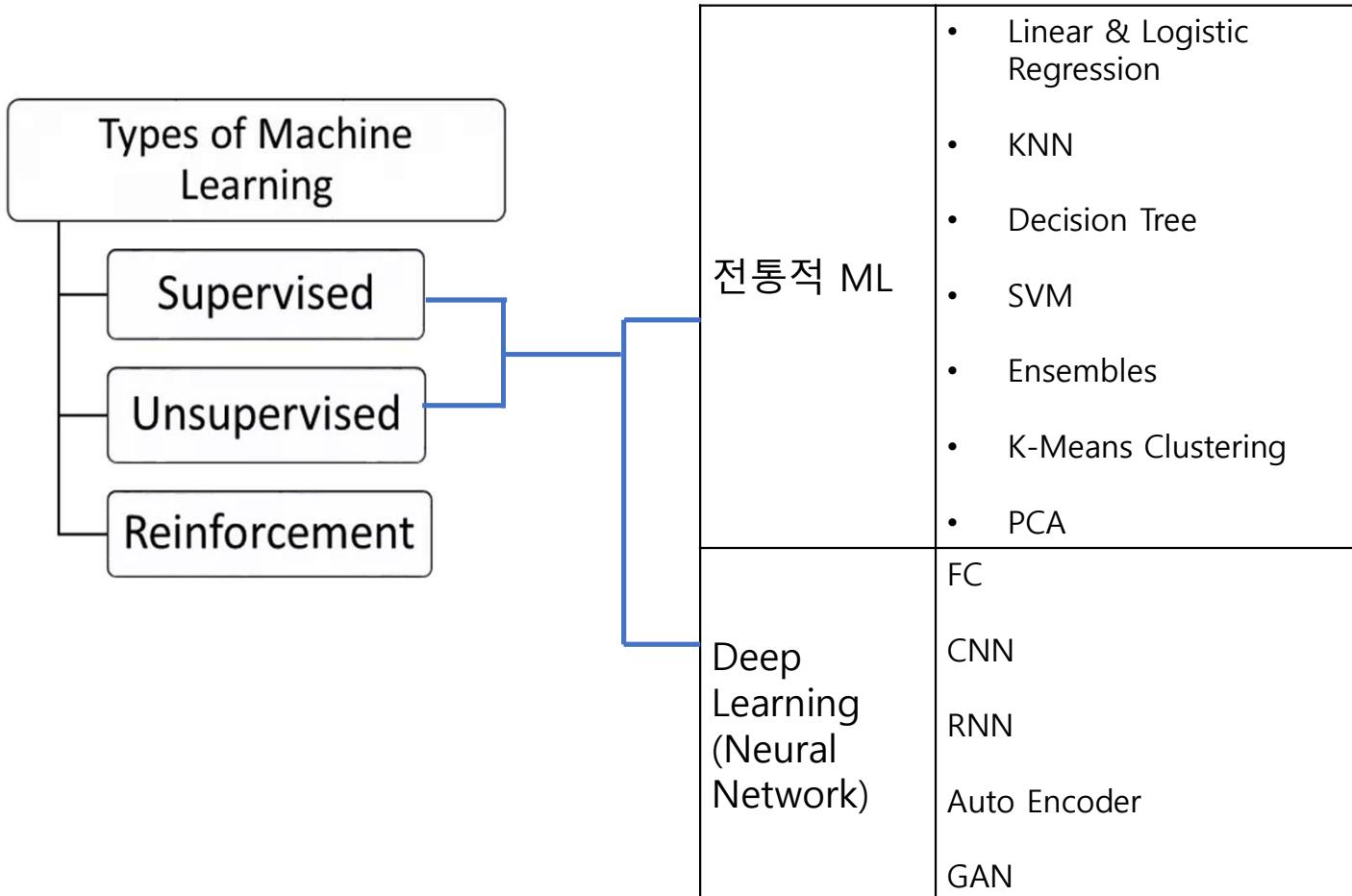
# End-to-End of Machine Learning



# End-to-End of Machine Learning



# Model 선택



## Model 작성 순서

Import Libraries : sklearn, numpy, pandas, matplotlib, etc



Data Load : csv, sklearn.datasets, etc

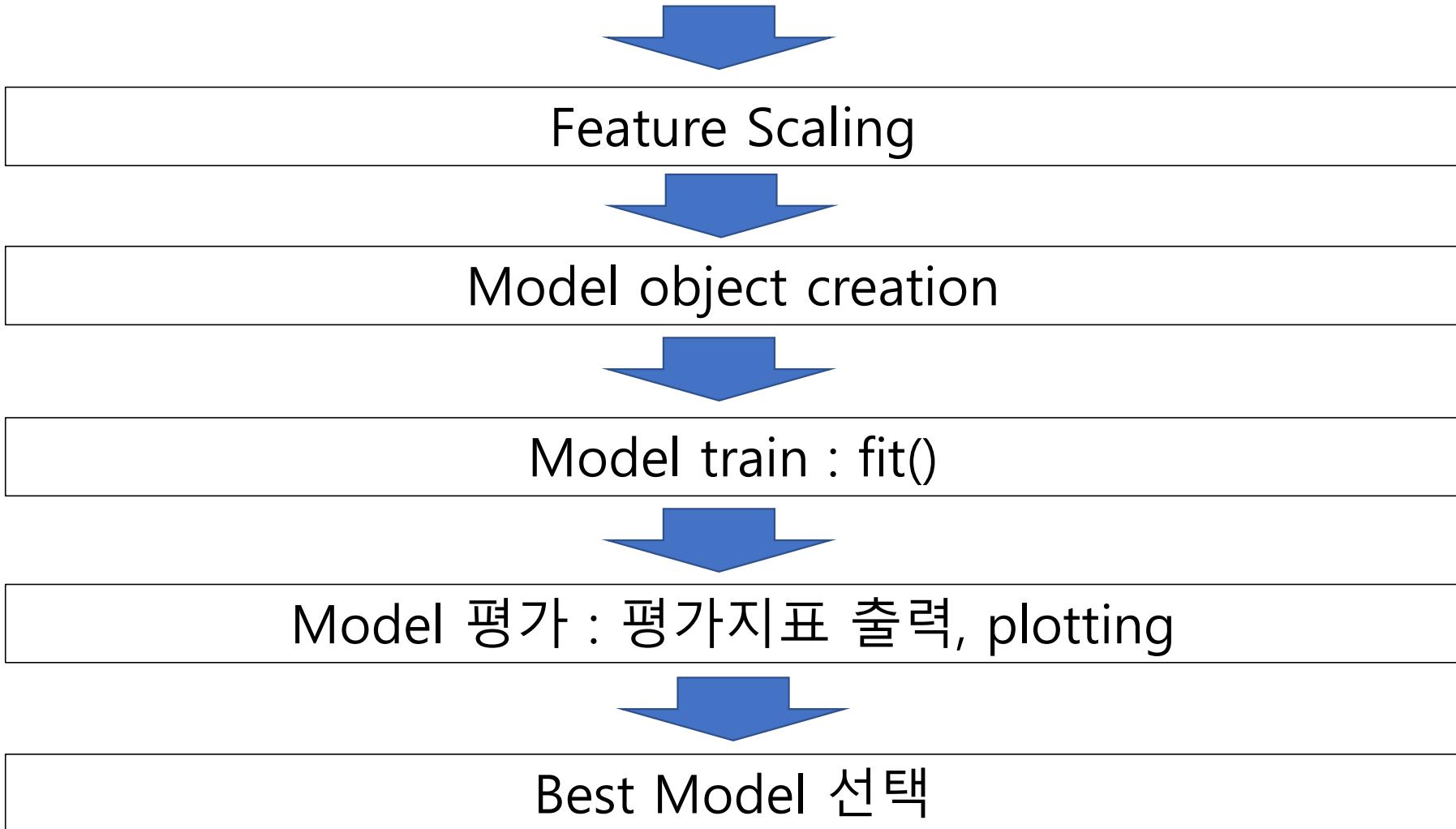


Data 내용 파악 : shape, statistics, visualize



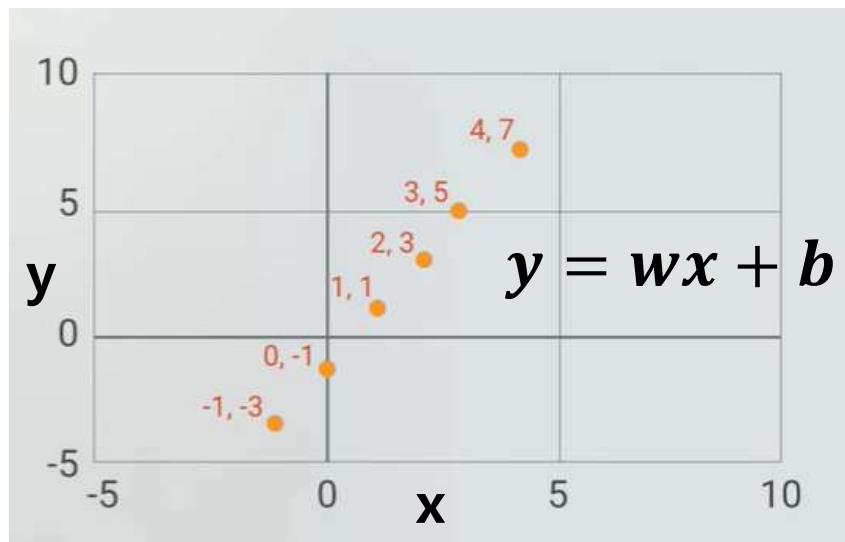
Train / test dataset 분할 : sklearn, manual





# Linear Regression

1. Univariate Linear Regression → 변수가 한 개 (ex. 혈압)
2. Multivariate Linear Regression  
→ 변수가 여러 개 (ex. 혈압, 체질량, 몸무게)
3. Polynomial Regression → 변수의 차수(degree) 증가



- x, y 가 주어지고 w, b 가 미지수
- ↓
- w, b 를 infer (추정)

# 1. Univariate Linear Regression (단변수선형회귀)

$$y = a + bx \rightarrow \text{Hypothesis (가설)}$$



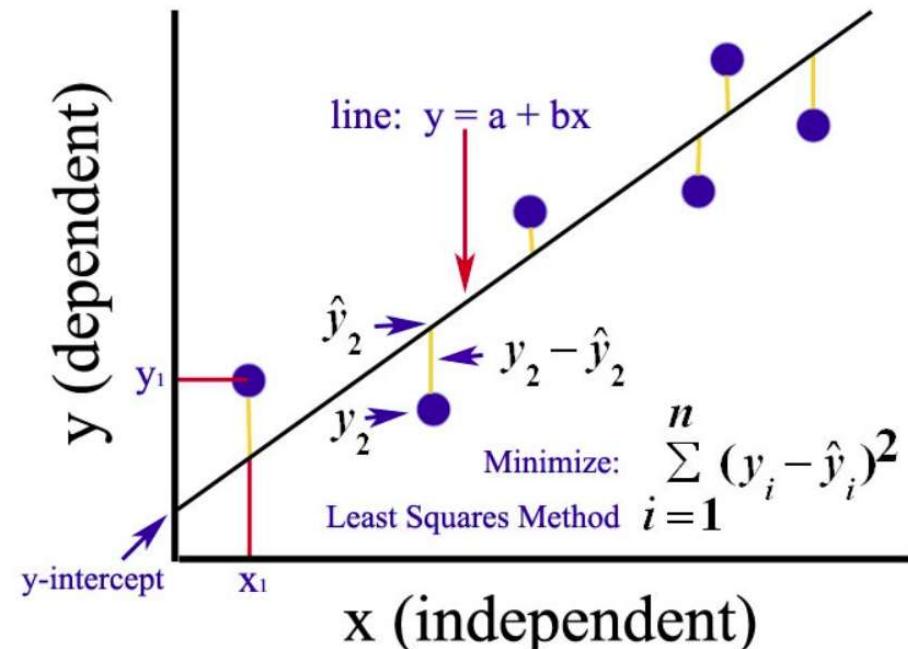
- OLS (Ordinary Least Squares, 최소자승법)

$$\text{Minimize } \sum_{i=1}^n (\text{true} - \text{prediction})^2$$



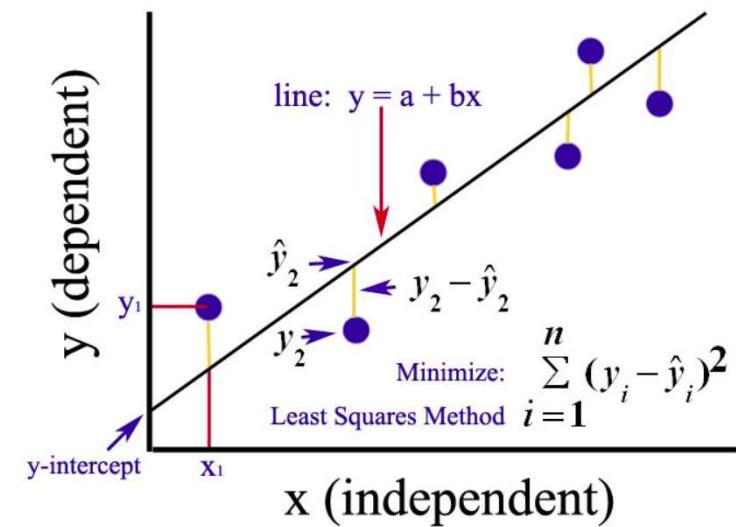
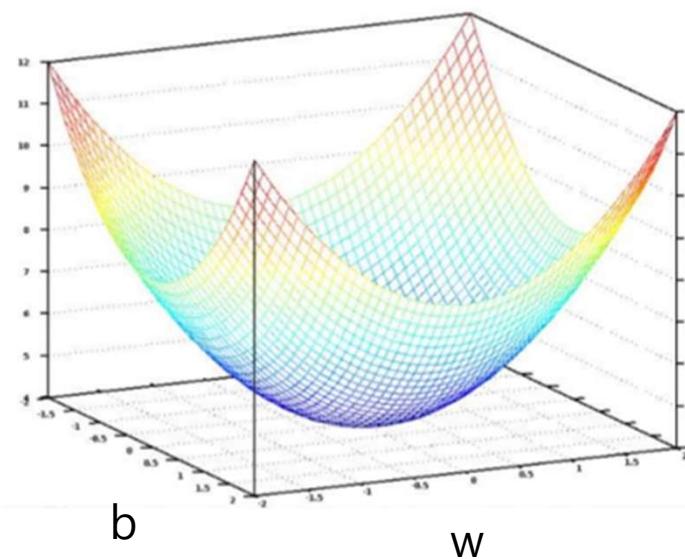
**Cost Function (비용함수)**

- 가설이 얼마나 틀렸는지 측정



# Cost Function - Linear Regression

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



$$y = \theta x + b$$

MSE 를 최소화 하는  
θ 와 b 를 optimize

# How to solve Cost Function of $\theta$ ?

- 선형대수의 Normal Equation (정규방정식) 이용
  - $\hat{\theta} = (X^T \cdot X)^{-1} \cdot X^T \cdot y$  ( $\hat{\theta}$ : 기울기/절편, X: training data, y: target data)
  - 계산 복잡도가  $O(n^3)$  Data 특성의 수(n)가 많으면 사용할 수 없음 (시간, memory)
- **Gradient Descent**: 미분을 이용한 점근적 방법 → Machine Learning 의 기본

$$MSE = \frac{1}{m} \sum_{i=1}^m (\hat{y}^{(i)} - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^m (\theta X^{(i)} - y^{(i)})^2$$



$$\frac{\partial f}{\partial \theta} = \frac{2}{m} \sum_{i=1}^m (\theta X^{(i)} - y^{(i)}) \cdot X^{(i)}$$

- 평가기준 2 : R2 score (결정계수)

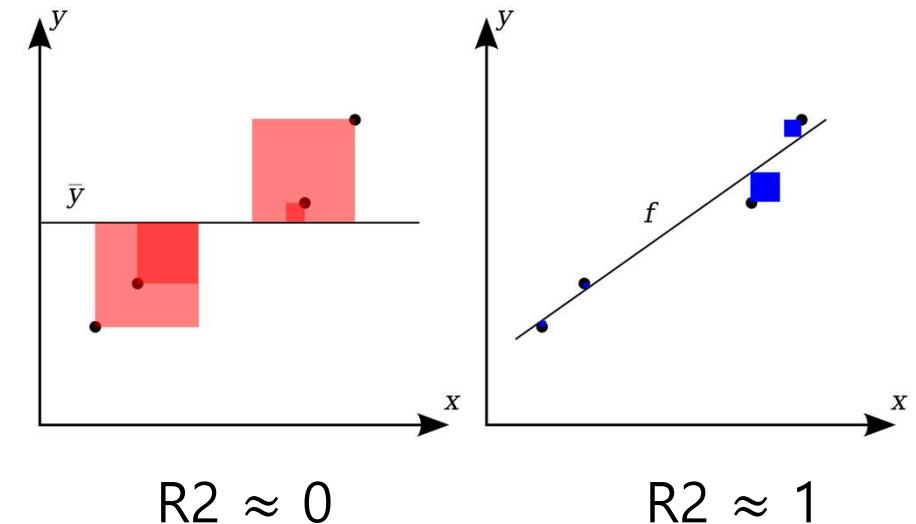
`sklearn.metrics.r2_score`

- \* 결정계수(R2) – 회귀식의 정확도 측정

$$0 \leq R^2 \leq 1$$

$$R^2 = 1 - \frac{\text{SSE}}{\text{SST}}$$

$$\left( \frac{\text{Sum of Square Error}}{\text{Sum of Square Total}} \right)$$



- 체질량지수(bmi) 하나만으로 univariate linear regression
- 체질량지수(bmi) 와 혈압(bp) 두 가지 변수로 multivariate linear regression

## 2. Multivariate Linear Regression (다변수선형회귀)

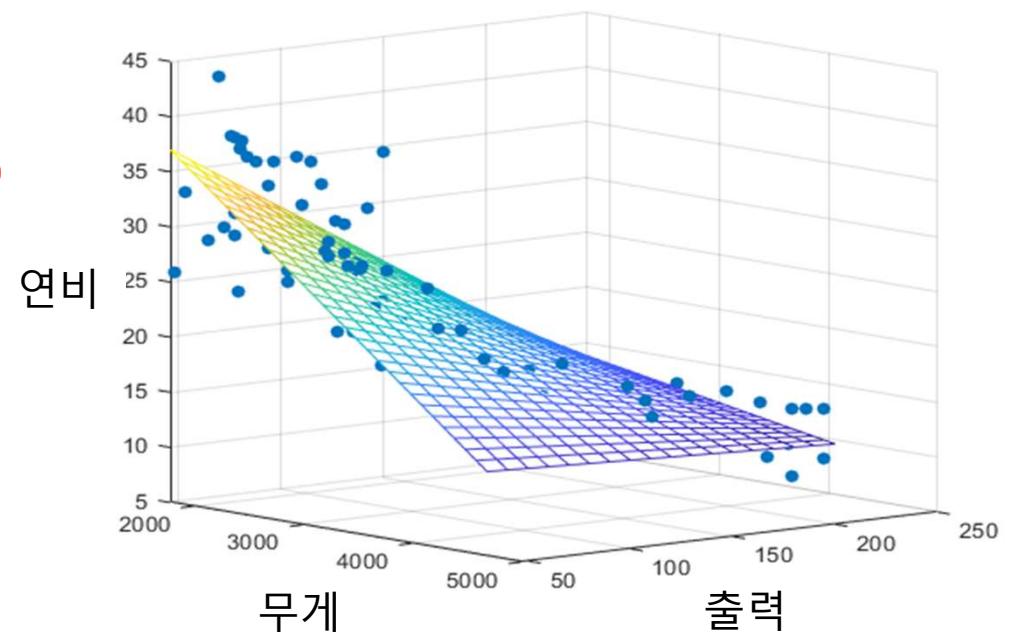
$$\hat{Y} = \theta_0 + \theta_1 X_1 + \theta_2 X_2 + \theta_3 X_3 + \dots + \theta_n X_n$$

$$\hat{Y} = \theta \mathbf{X}$$

`sklearn.linear_model.LinearRegression()`

$\theta = \text{coef}_-$

$\theta_0 = \text{intercept}_-$



# 실습: 당뇨병 data 를 이용한 선형회귀

- Dataset : `sklearn.datasets.load_diabetes()`
- Feature : 나이, 성별, 체질량지수, 혈압, 6가지 혈청 수치 → scaling 되어 있음
- Target : 1년 뒤 측정한 당뇨병의 진행률
- Model : OLS (Ordinary Least Squares)
- 평가기준 1 : RMSE (Root Mean Squared Error)  
`sklearn.metrics.mean_squared_error`

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

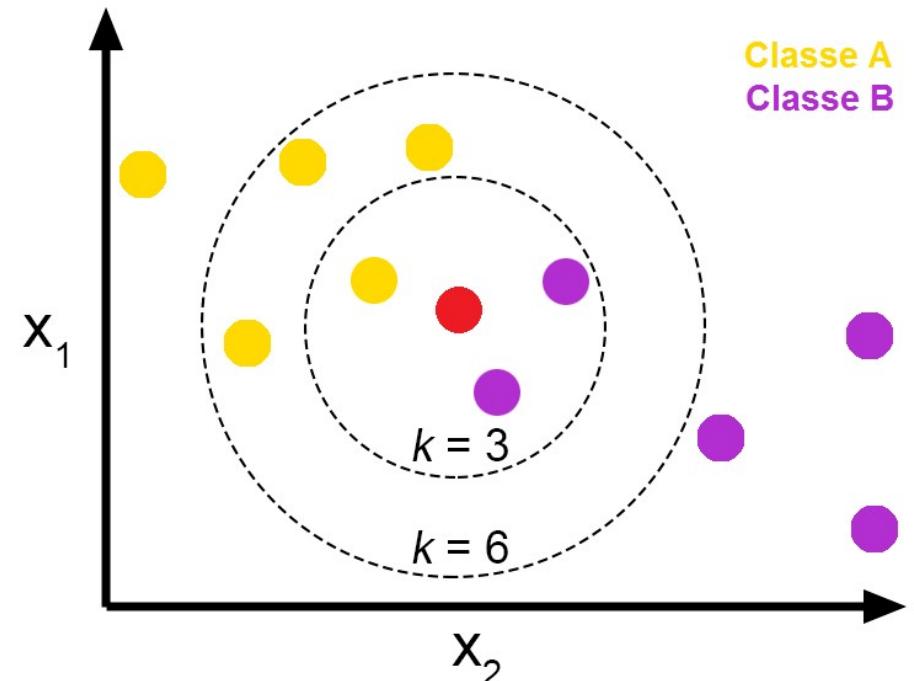
n : sample 수

# Classification

# K-Nearest Neighbors

# KNN (K-Nearest Neighbors, K 최근접 이웃)

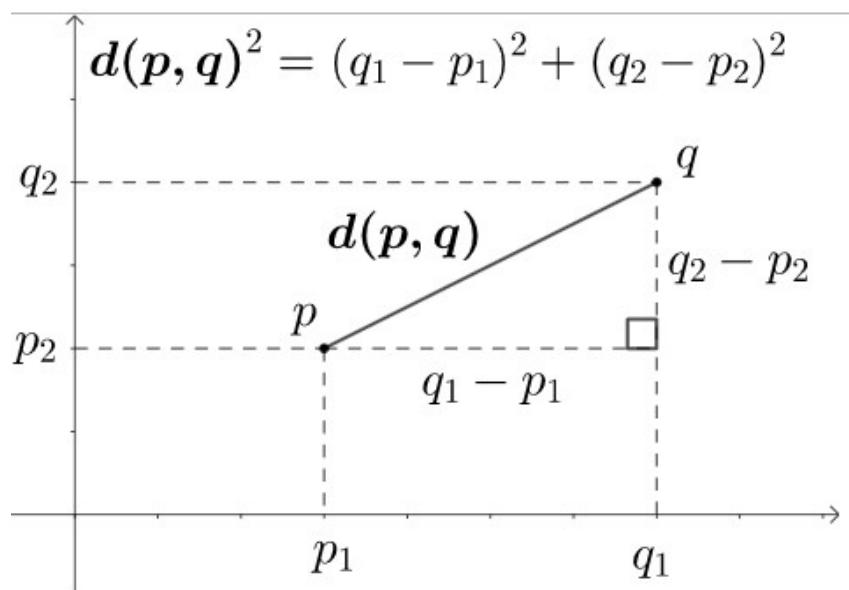
- 다른 observation (관측치, X data) 과의 유사성에 따라 분류 (classify)
- 서로 가까이 있는 data 들을 “이웃” (neighbor)이라고 부른다.
- 가까이 있는 이웃의 label 들 중 가장 많은 것을 unknown case 의 prediction 으로 응답한다.
- 장점 : simple and easy to implement
- 단점 : dataset 이 커지면 slow.  
outlier/missing value 의 영향이 크다.



# KNN 알고리즘

1. K 값을 선택한다.

2. Unknown case 와 모든 data point 간의 거리를 계산한다.



$$\begin{aligned} d(\mathbf{p}, \mathbf{q})^2 &= (q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2 \\ d(\mathbf{p}, \mathbf{q}) &= d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} \\ &= \sqrt{\sum_{i=1}^n (q_i - p_i)^2}. \end{aligned}$$

3. Training dataset에서 unknown data point와 가장 가까이 있는 K개의 관측치(observation)을 선택한다.
4. K개의 nearest neighbors의 label 중 가장 많은 것을 unknown data point의 **class**로 **분류**한다 ( $\rightarrow$  classification)

K개의 nearest neighbors의 label value의 평균을 predicted **value**로 **계산** 한다 ( $\rightarrow$  regression)

## 실습 :

1. sklearn 에서 제공하는 iris (붓꽃) 분류 dataset 사용 :

꽃잎의 각 부분의 너비와 길이 등을 측정한 데이터이며  
150개의 레코드로 구성

2. Dataset 의 형식:

data – 독립변수 (ndarray)

target – 종속변수 (ndarray)

feature\_names – 독립변수 이름 list

target\_names : 종속변수 이름 list

DESCR – 자료에 대한 설명

### 3. Data 의 내용 :

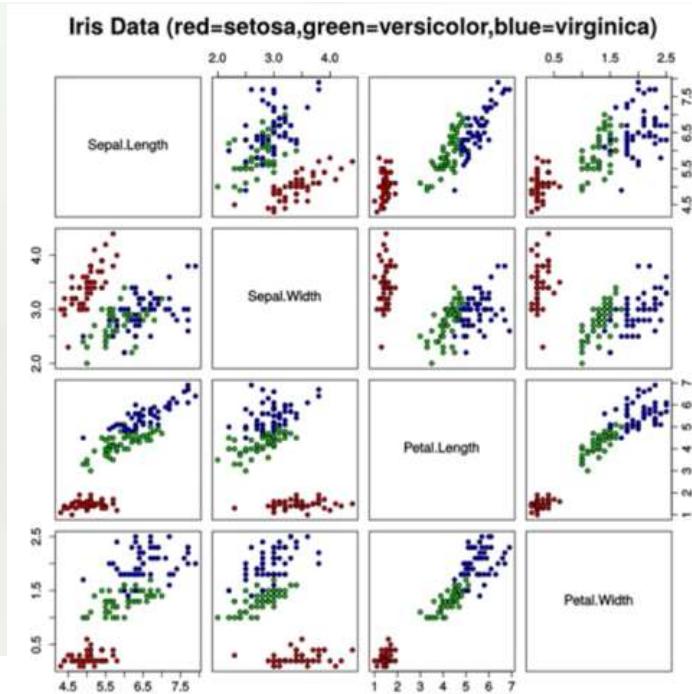
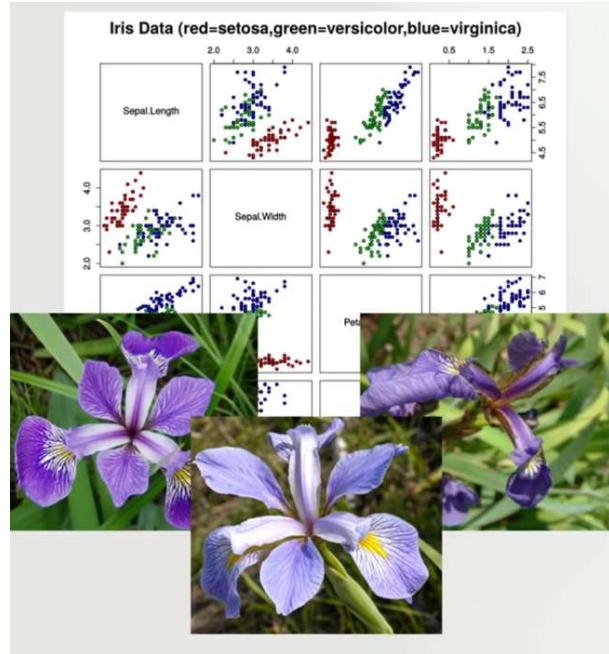
Sepal Length : 꽃받침 길이  
Sepal Width : 꽃받침 너비  
Petal Length : 꽃잎 길이  
Petal Width : 꽃잎 너비

### 4. Neighbor 개수 ( $K = 15$ )

### 5. 거리에 따른 가중치 parameter

uniform – neighbor 간의 거리 무시

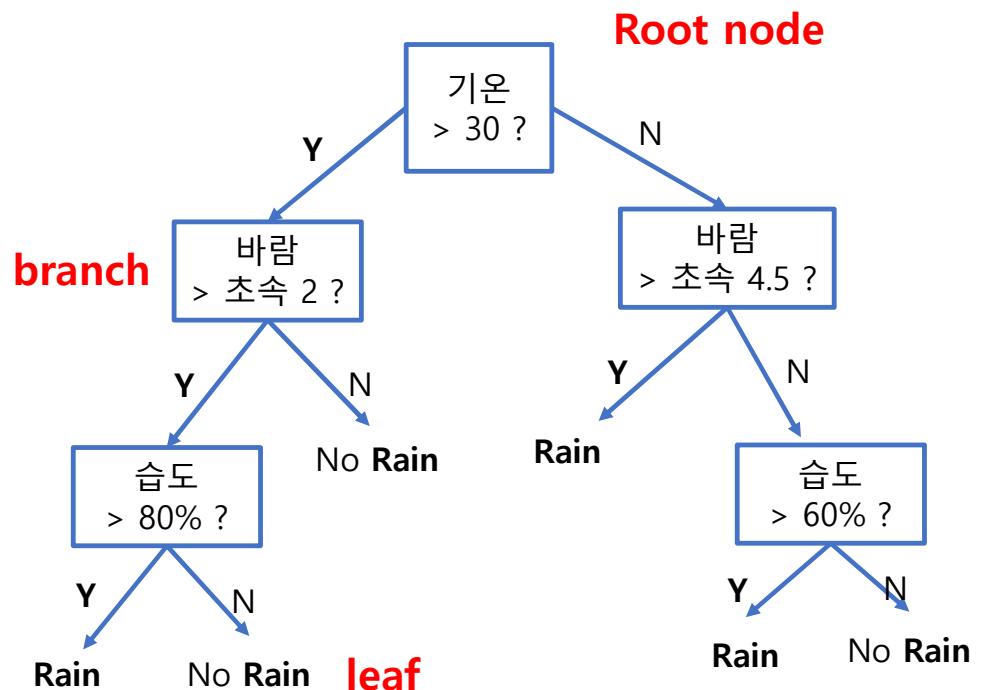
distance – 가까운 neighbor 에 더 높은 가중치 부여



# Decision Tree

# Decision Tree (결정나무)

- 모든 가능한 결정 경로(Decision Path)를 tree 형태로 구성
- 각 node 는 test 를 의미
- 각 branch 는 test 의 결과에 해당
- 각 leaf node 는 classification 에 해당
- 장점 : **white-box model**  
**data preprocessing** 불필요
- 단점 : **overfitting** 되기 쉽다.  
훈련 데이터의 작은 변화에도 매우 민감



# Decision Tree 알고리즘의 종류

1. ID3 – 기본적 알고리즘. 정보이득(Information Gain) 을 이용한 트리 구성
2. CART (Classification and Regression Tree)
  - Gini 불순도에 기반한 트리 구성
3. C4.5, C5.0 – ID3 개선
4. 기타 – CHAID, MARS

# 엔트로피 (Entropy), 정보 이득(information gain)

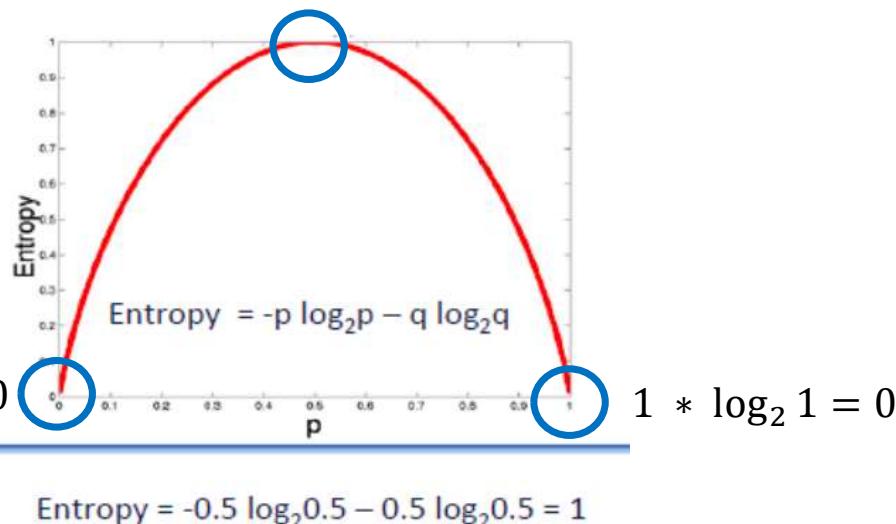
- 엔트로피(Entropy) - 주어진 데이터 집합의 혼잡도. 즉, 우리가 가지고 있지 않은 정보의 양을 의미.
- 주어진 데이터 집합에서 서로 다른 종류의 레코드들이 섞여 있으면 엔트로피가 높고, 같은 종류의 레코드들이 섞여 있으면 엔트로피가 낮음.
- 우리가 시스템에 대해 알게 될수록 시스템의 엔트로피는 감소. 예를 들어 데이터의 통계를 알게 되면 엔트로피는 감소. 이것을 정보 이득(information gain)이라고 한다.
- 엔트로피 값은 0에서 1사이의 값. 가장 혼합도가 높은 상태의 값이 1이고, 반대는 0

- Decision Tree 에서는 엔트로피가 높은 상태에서 낮은 상태가 되도록 데이터를 특정 조건을 찾아 나무 모양으로 구분해 나감.

$$\text{Entropy} = - \sum_{i=1}^m p_i \log_2(p_i) , \quad p_i = \frac{\text{freq}(C_i, S)}{|S|}$$

(S: 주어진 데이터들의 집합, C: 레코드(클래스) 값들의 집합, freq(Ci,S): S에서 Ci에 속하는 레코드의 수, |S|: 주어진 데이터들의 집합의 데이터 개수)

$$p = 0.5, \text{ Entropy} = 1$$



$$\log_2 0.5 = \log_2 \frac{1}{2} = -\log_2 2 = -1$$

$$\log_a 1 = 0$$

$$\log_a a = 1$$

$$\log a^{MN} = \log a^M + \log a^N$$

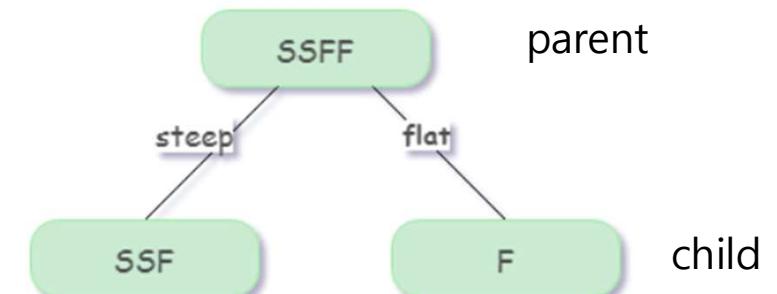
$$\log a^{M/N} = \log a^M - \log a^N$$

$$\log a^{M^p} = p \log a^M$$

- Information Gain = Entropy(Parent) – (weight) \* Entropy(Child)

ex)

Feature			Label
경사도	노면상태	속도제한	속도
steep	bumpy	Yes	slow
steep	smooth	Yes	slow
flat	bumpy	No	fast
steep	smooth	No	fast



- “경사도”의 information gain 계산:

$$\text{Entropy}(\text{Parent}) = -\{0.5 \log_2(0.5) + 0.5 \log_2(0.5)\} = 1$$

$$\text{Entropy}(\text{Child/steep}) = -\{0.667 \log_2(0.667) + 0.334 \log_2(0.334)\} = 0.918$$

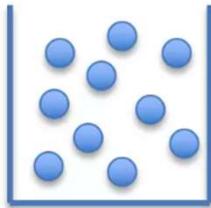
$$\text{Entropy}(\text{Child/flat}) = -\{0 + 1 \log_2(1)\} = 0$$

$$\text{Entropy}(\text{가중평균}) = \frac{3}{4} \cdot 0.918 + \frac{1}{4} \cdot 0 = 0.688$$

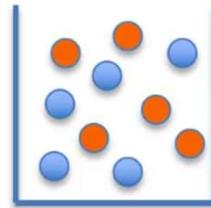
- Information Gain =  $1 - 0.688 = 0.312$

- Gini Impurity (지니불순도)

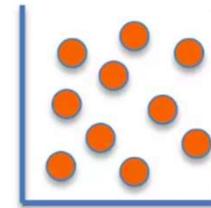
- CART 알고리즘에서 사용



항아리 1.



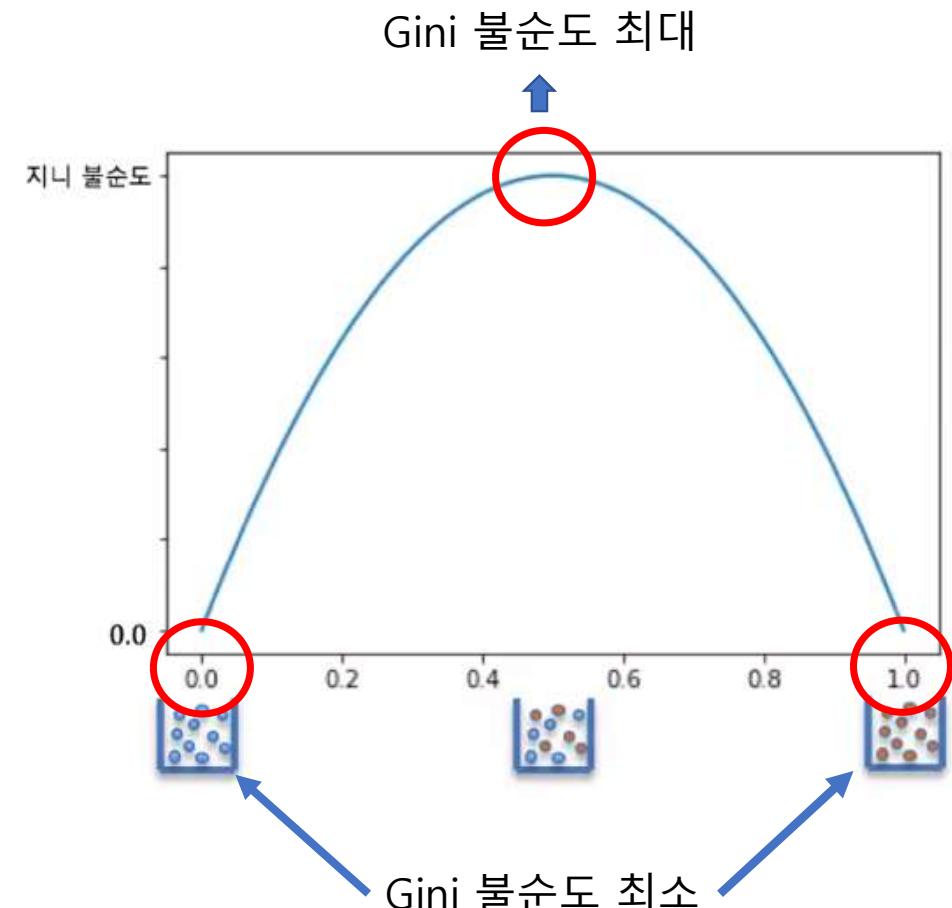
항아리 2.



항아리 3.

지니불순도를 최소화 하는 방향으로 Tree 를 구성

$$\sum_{j=1}^J p_j(1 - p_j)$$



# Decision Tree 알고리즘 (ID3)

1. Initial open node 를 생성하고 모든 instance 를 open node 에 넣는다.
2. Open node 가 없어질 때까지 loop
  - 분할할 open node 선택
  - information gain 이 최대인 attribute(feature) 선택
  - 선택된 attribute 의 class (Y, N) 별로 instance sort
  - sort 된 item 으로 새로운 branch 생성
  - sort 된 item 이 모두 하나의 class 인 경우 leaf node close

# 실습 : Decision Tree 작성 및 시각화

1. KNN에서 사용하였던 Iris data 사용
2. Tree의 max\_depth = 2, None으로 변경하여 test/비교
3. graphviz를 이용한 visualization

# Graphviz 설치 방법 (or Google Collab 사용)

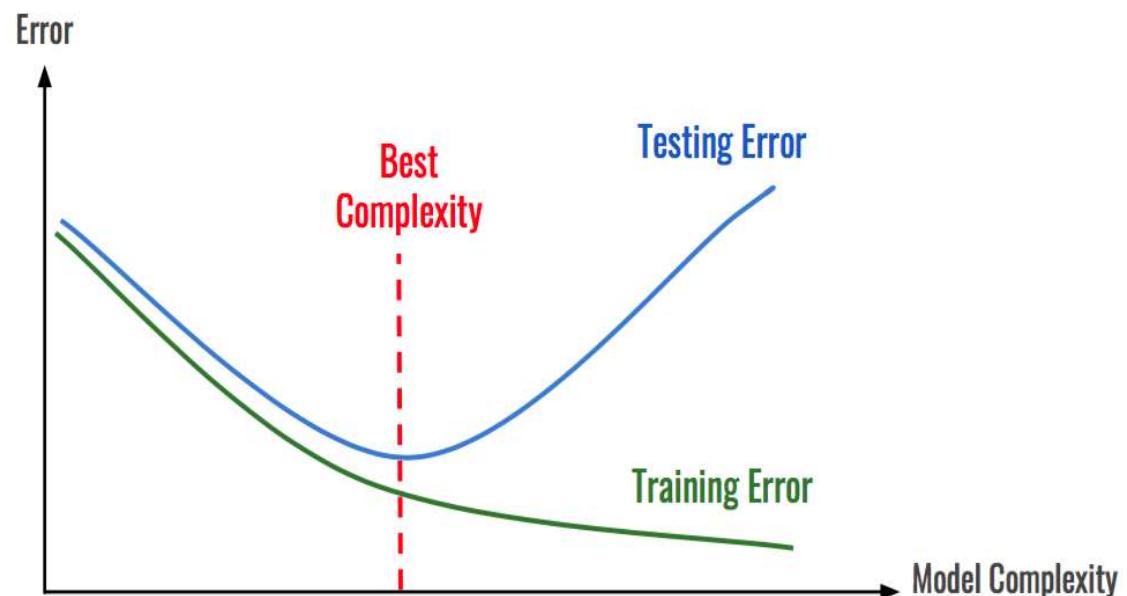
1. Install graphviz windows (<https://graphviz.gitlab.io/download/>)  
Stable 2.38 Windows install packages → graphviz-2.38.msi 설치
2. 환경변수 setting  
C:\Program Files (x86)\Graphviz2.38\bin 경로 copy  
내PC - 속성 - 고급시스템설정 - 환경변수 - Path - 편집  
- 새로만들기 - 경로 paste
3. anaconda prompt에서 pydotplus install

```
>conda install python-graphviz  
>pip install pydotplus
```

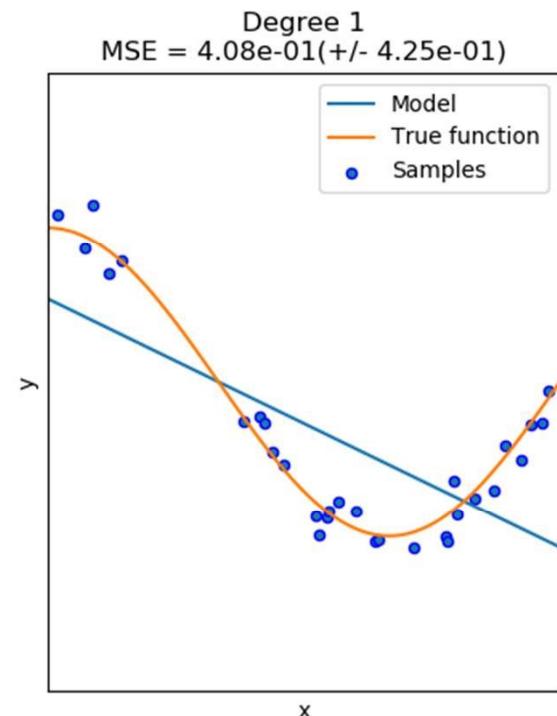
# Training/Test/Evaluation

# Overfitting(과적합) 과 Underfitting(과소적합)

- Training Data 에 비해 Test Data 의 ERROR 율이 높게 나타나는 경우 이를 과적합 (Overfitting) 이라고 한다.
- 반대로 모델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못하는 경우 과소적합 (Underfitting) 이라고 한다.



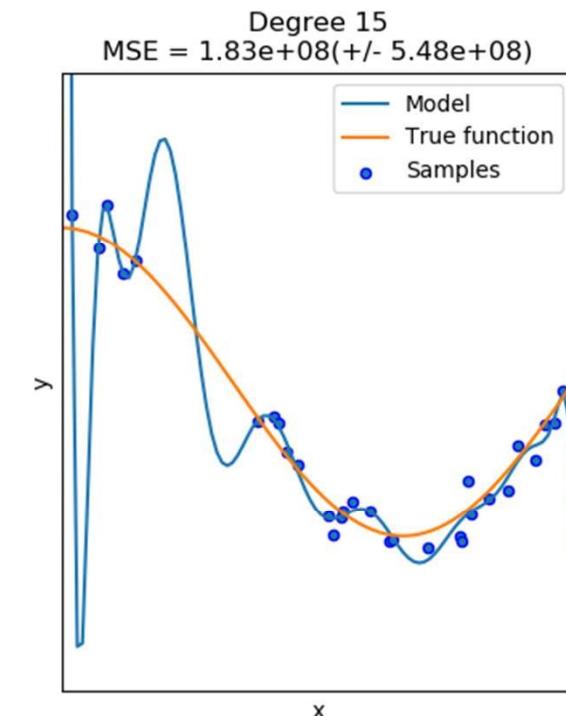
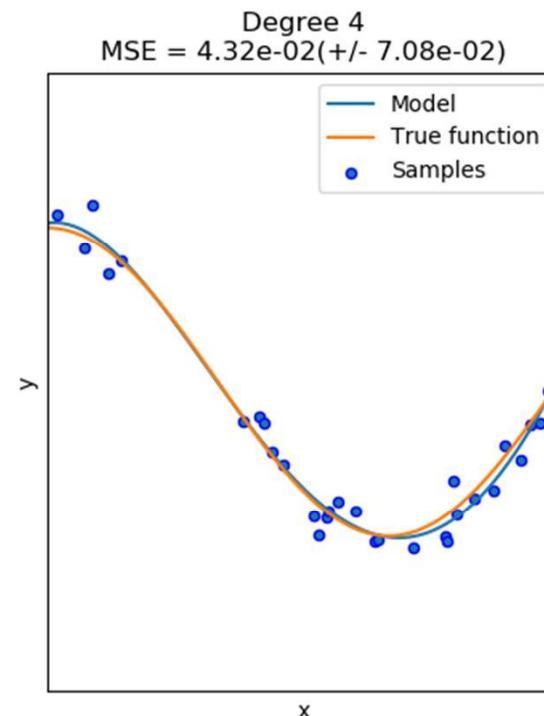
## Underfitting (과소적합)



모델이 너무 단순  
(data의 중요 부분을 놓침)  
High Bias Model



## Overfitting (과대적합)

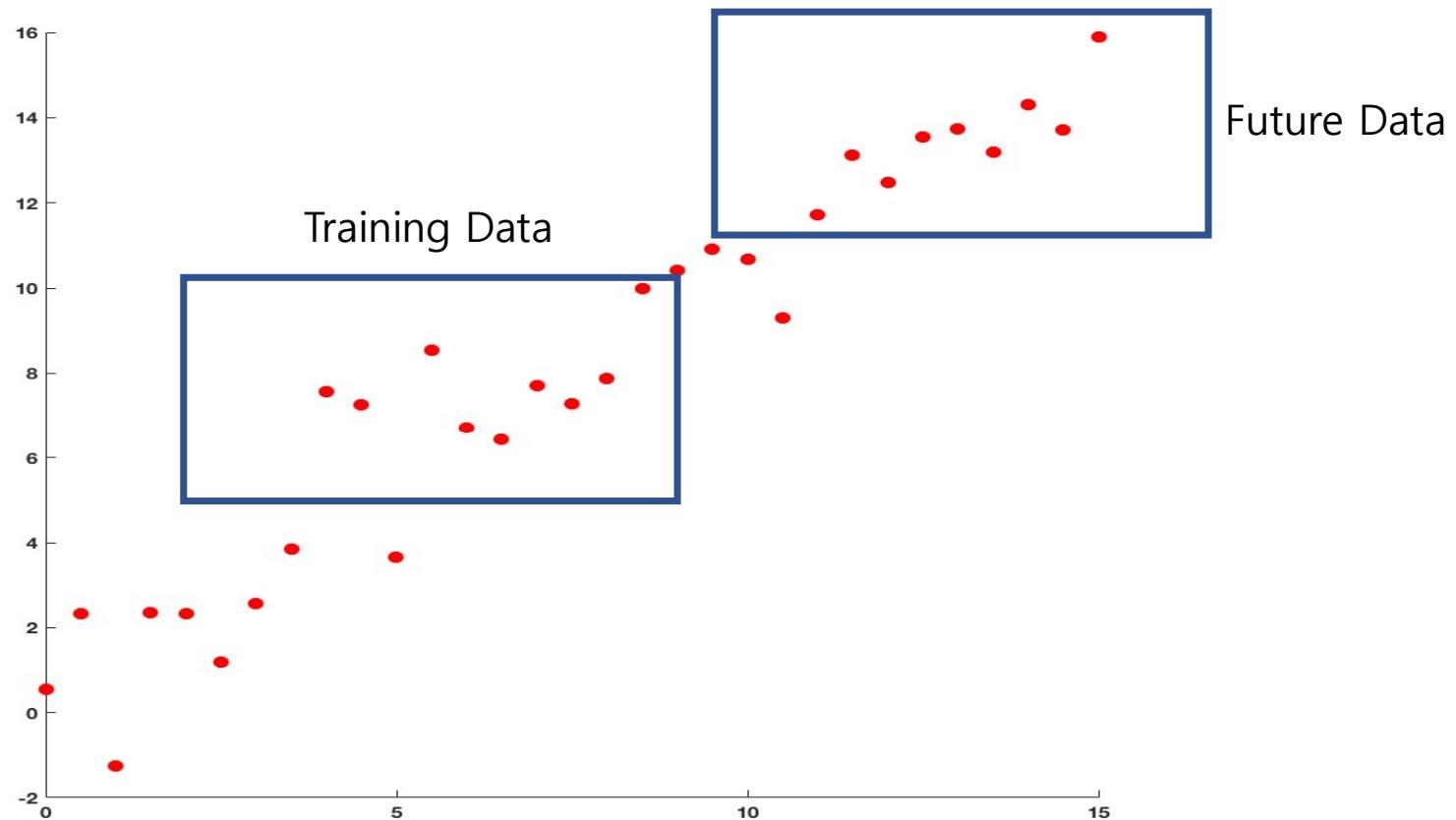


모델이 너무 복잡  
(실제와 무관한 noise 까지 학습)  
High Variance Model

# Machine Learning 의 Source of Error

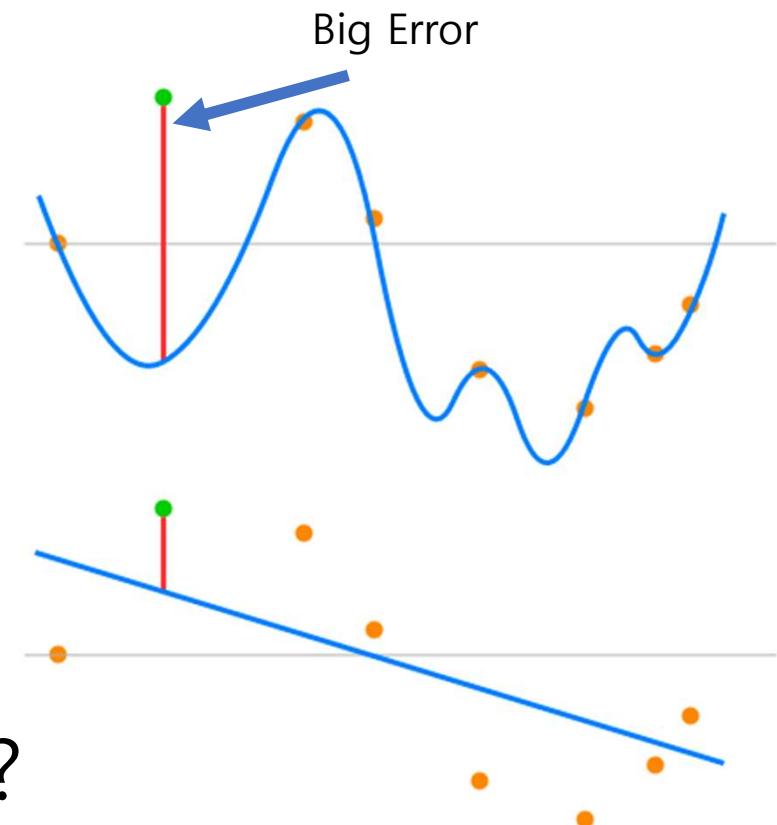
- 세가지의 Error Source
  1. 학습 Data 와 실제 data 분포의 차이에 의한 error → Variance
  2. Approximation Model 과 True Function 의 차이에 의한 error → Bias
  3. Noise 에 의한 error → 제거할 수 없음
- Variance 를 줄이려면 Dataset 의 크기를 늘이고, Bias 를 줄이려면 모델의 Complexity 를 올린다.  
→ Bias-Variance Dilemma (Bias-Variance Trade-off)

# Variance – Training Data vs. Future Data



# Bias – True & Inference Function 차이

- True function 은 sine 함수
- Model 1 – polynomial regression
- Model 2 – Linear regression



Which one is better model ?

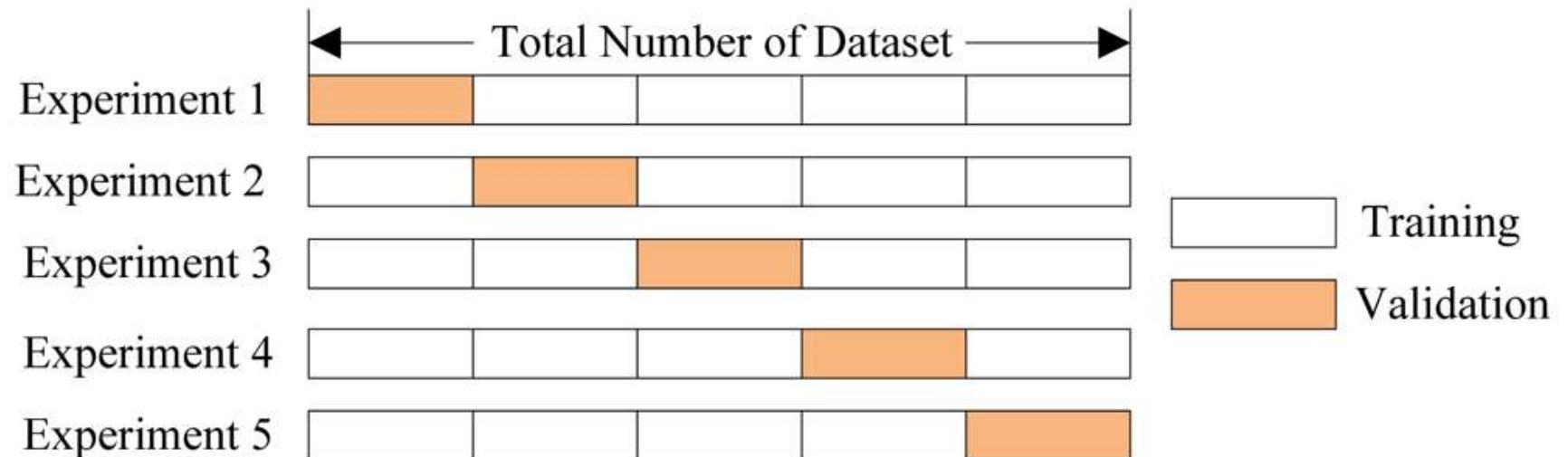
- Variance – infinite data sampling 으로 해결 가능
- Bias – true function 을 알면 해결 가능
- **BUT**, 현실에서는 infinite data sampling 도 할 수 없고 true function 도 알 수 없으므로 간접적 방법을 사용
  1. Cross Validation
  2. Precision / Recall / F1-Score

# Training & Testing & Cross-Validation Set

- Training Set
  - parameter 를 inference 하는 procedure 에 사용하는 data
  - 보지 못한 Data 의 분포가 Training set 과 상이할 경우 문제
  - Training set 내에서 cross-validation set 을 구성 (Data 가 충분한 경우)
- Testing Set
  - 학습한 Machine Learning Model 을 Test 하기 위해 사용하는 data
  - 미래의 instance 인 것처럼 간주
- Training set 과 Testing set 은 섞이면 안되고 동일한 분포를 유지 해야함.

# Cross Validation (교차검증)

- 훈련세트를 여러 개의 sub-set 으로 나누고 각 모델을 이 sub-set 의 조합으로 훈련시키고 나머지 부분으로 검증
- Data 의 수가 적은 경우 사용



# 실습: Train/Test split and K-Fold Cross-Validation

## 1. Training Set 과 Test set 으로 구분

- 당뇨병 dataset 을 이용하여 실습
- sklearn.model\_selection 의 train\_test\_split method 사용
- Test set (true value) 를 x 축으로, predicted value 를 y 축으로 시각화
  - True value vs. predicted value 산점도(scatter plot) 표시
  - True value 와 predicted value 가 일치할 경우를 line 으로 표시

## 2. k-Fold Cross-Validation

- sklearn.model\_selection 의 cross\_val\_score 를 이용하여 model 평가
- 데이터가 소량인 경우 혹은 여러 model 의 적합성 비교에 편리
- computing cost high

# Model Performance for Biased Data

# Confusion Matrix (혼동 행렬)

		True condition	
		Condition positive	Condition negative
Predicted Condition	Total population	Condition positive	Condition negative
	Predicted condition positive	TP True positive	FP False positive
	Predicted condition negative	FN False negative	TN True negative

- Classification (분류) 성능의 정확성 측정

TP – 1 을 1 로 제대로 분류, FP – 0 을 1 로 잘못 분류  
FN – 0 을 1 로 잘못 분류, TN – 0 을 0 으로 제대로 분류

- Classification rate (Accuracy) =  $(TP + TN) / (TP+TN+FP+FN)$   
→ 단순 정확성. 전체 데이터 중에서, 제대로 분류된 데이터의 비율

# Confusion Matrix 를 이용한 분류 모델 성능 평가

- Precision = TP / (TP + FP)
  - 정밀성. Positive로 예측한 내용 중에, 실제 Positive의 비율  
→ Model 이 sample 을 True 로 분류했을 때 얼마나 자주 맞추었는가 ?
  - positive 분류의 정확성 측정 (1 에 가까울 수록 좋음)

		True condition	
		Condition positive	Condition negative
Total population		Condition positive	Condition negative
Predicted Condition	Predicted condition positive	True positive	False positive
	Predicted condition negative	False negative	True negative

ex) 포르노 영상 검출기 – 포르노로 분류했을 때 실제 포르노인 비율  
security check 영상 탐지기 – 통과 승인된 사람 중 실제 직원 비율

# Confusion Matrix 를 이용한 분류 모델 성능 평가

- Recall (Sensitivity/ True positive Rate) =  $TP / (TP + FN)$ 
  - 민감도. 전체 Positive 데이터 중에서 Positive로 분류한 비율 (1에 가까울수록 좋음)
  - Positive case 를 놓치고 싶지 않은 경우의 성능 측정

		True condition	
		Condition positive	Condition negative
Total population		Condition positive	Condition negative
Predicted Condition	Predicted condition positive	True positive	False positive
	Predicted condition negative	False negative	True negative

→ Type I error

→ Type II error

ex) 포르노 영상 검출기 – 전체 포르노 중 포르노로 분류된 비율  
security check 영상 탐지기 – 전체 직원 중에서 통과로 분류된 비율

# Precision / Recall 활용 방법

- Confidence 수준을 올리고 싶으면 Precision 을 높이고 Recall 을 낮추도록 Threshold 조정. 너무 많은 case 를 놓치고 싶지 않은 경우 Recall 을 높이고, Precision 을 낮춘다.
- 전체적 성능 측정에 활용 (조화평균)

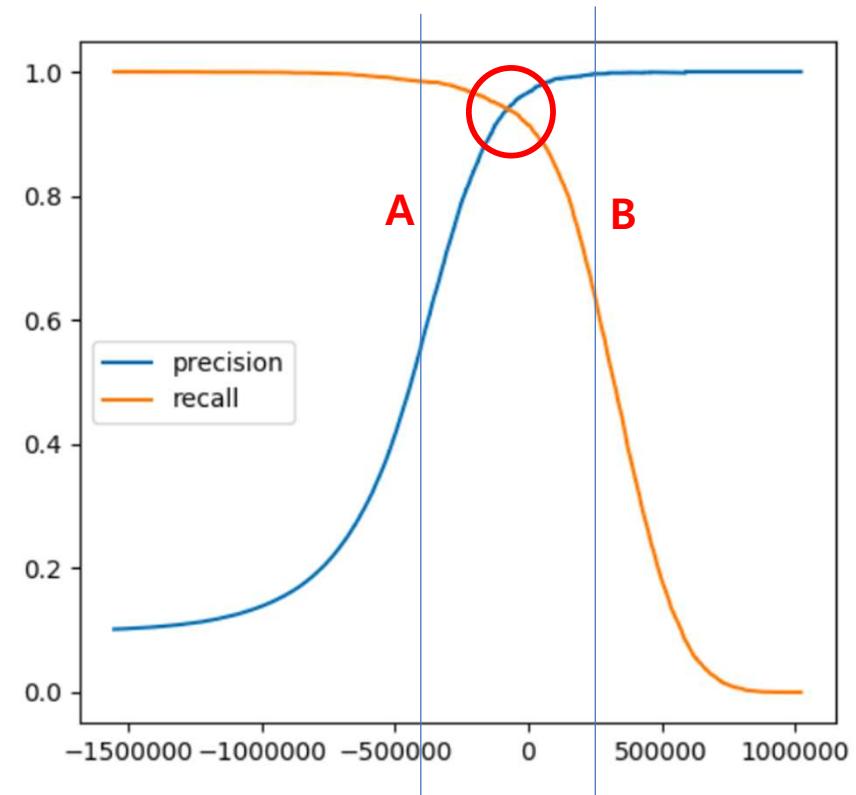
$$F1\text{-Score} = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1-Score = 0 ---> Poor (P=0 or R=0)

F1-Score = 1 ---> Perfect (P=1 and R=1)

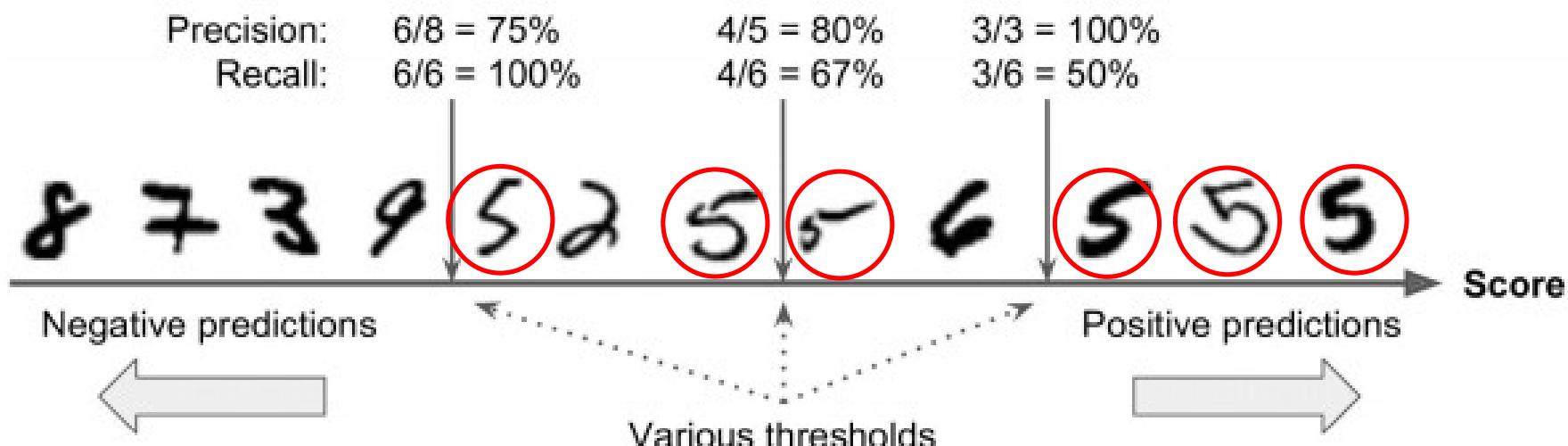
A – High Recall / Low Precision

B – High Precision / Low Recall



# Precision/Recall Trade-off

- 숫자 중 5 를 검출

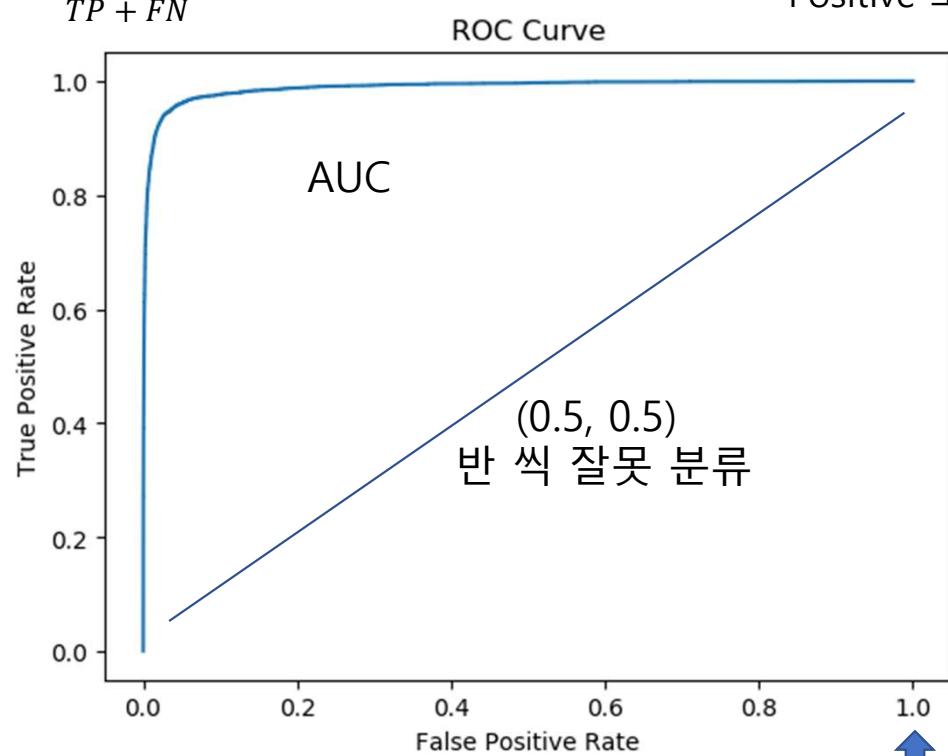


5 가 검출에서 누락되지 않기 원하는 경우

5 가 정확히 검출되기 원하는 경우

# ROC Curve (수신자 조작 특성 곡선)

$$TPR = \frac{TP}{TP + FN}$$



모든 Positive 를  
Positive 로 정확히 분류



- ROC (Receiver Operating Characteristic) curve 는 radar 상의 적기 탐지를 위해 개발되었던 분석 기법
- ROC\_AUC (Area Under the Receiver Operating Characteristic Curve) 라고도 함
- roc\_auc\_score 를 이용하여 **분류기(classifier) 간의 성능 비교**를 할 수 있다.

$$FPR = \frac{FP}{FP + TN}$$

모든 Negative 가  
Positive 로 잘못 분류



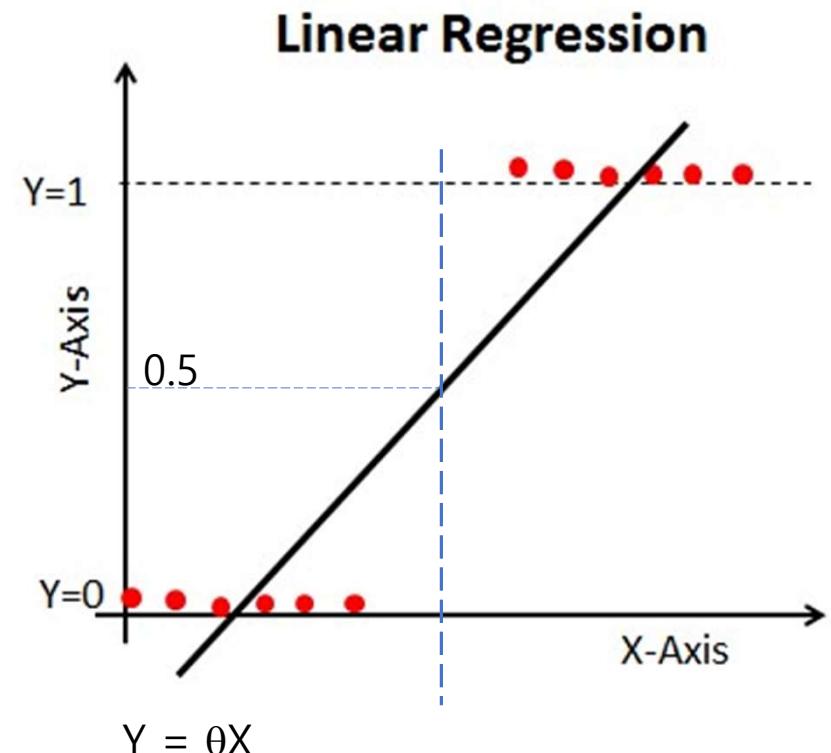
# Logistic Regression

# Logistic Regression (로지스틱 회귀)

- 선형회귀를 분류 문제에 적용 가능 ?

$$\hat{y} = \begin{cases} 0 & \text{if } \theta X < 0.5 \\ 1 & \text{if } \theta X \geq 0.5 \end{cases}$$

- 0 과 1 로 구성된 분류 문제에 적합한 함수 ?
- 0 과 1 에 속할 확률값을 return
- 미분가능한 성질



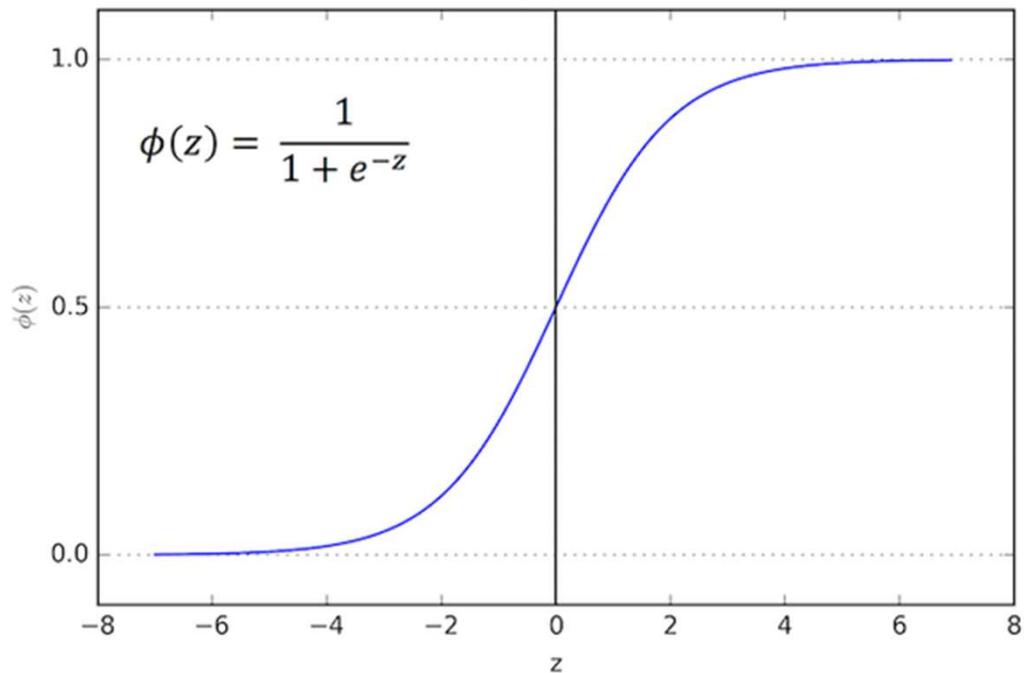
# Sigmoid 함수

- S curve 형성
- Logistic 함수

$$f(z) = \frac{1}{1+e^{-z}} \quad (z = \theta X)$$

$$\hat{y} = \begin{cases} 0 & \text{if } f(z) < 0.5 \\ 1 & \text{if } f(z) \geq 0.5 \end{cases}$$

- $[0, 1]$  로 bound 되어 있음
- 미분가능
- 0.5 부근에서 급격히 변화



# Logistic Regression (로지스틱 회귀) classifier

1. 가장 단순한 분류기 → binary-class
2. 구현이 간단하고 모든 classification problem 의 기초
3. Logistic Regression 의 기본 concept 은 Deep Learning 에도 적용
4. 독립변수와 종속변수 간의 관계를 찾아내고 평가함
5. `sklearn.linear_model.LogisticRegression(solver='lbfgs')`  
\* solver – optimization algorithm

# 실습: Logistic Regression 을 이용한 Binary Classification

1. 특정 사용자가 구매를 할지 여부를 예측 (구매: 1, 구매 않음: 0)
2. Dataset 구성  
사용자 id, 성별, 연령, 추정급여, 구매여부  
이중 연령, 추정급여 두가지 feature 를 이용하여 구매여부 예측
3. Train / Test dataset 은 8 : 2 로 분리
4. Feature Scaling 실시 (standard scaling)
5. Model evaluation by Confusion Matrix, f1-score
6. Visualization – ROC curve

# Support Vector Machine

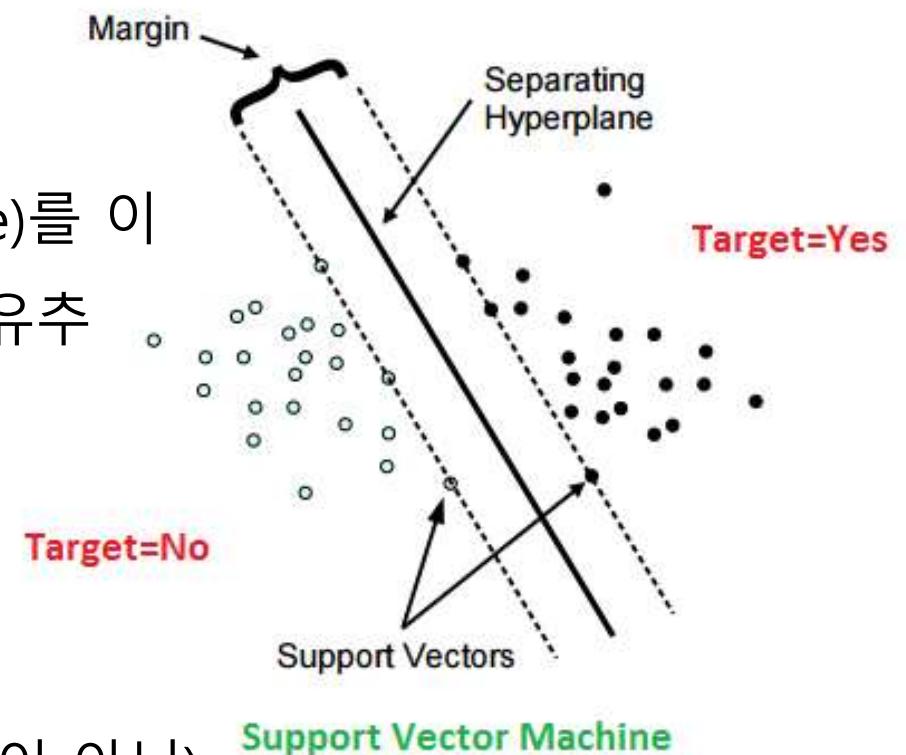
# SVM (Support Vector Machine)

- SVM 은 data 를 positive (+) 와 negative (-) 두개의 class 로 구분하는 선형 분류기 (Linear Classifier) 임
- 복잡한 분류에 잘 맞고 중간 크기 이하의 dataset 에 적합
- Training data 가  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$  의 vector 로 표시되고,  $y_i = \{-1, +1\}$  인 경우 SVM 은 다음 선형방정식을 만족하는 w (weight vector) 를 찾음

$$f(x_i) = w \cdot x_i + b$$

*if*  $f(x_i) \geq 0$ ,  $y_i = +1$ ; *else*  $y_i = -1$

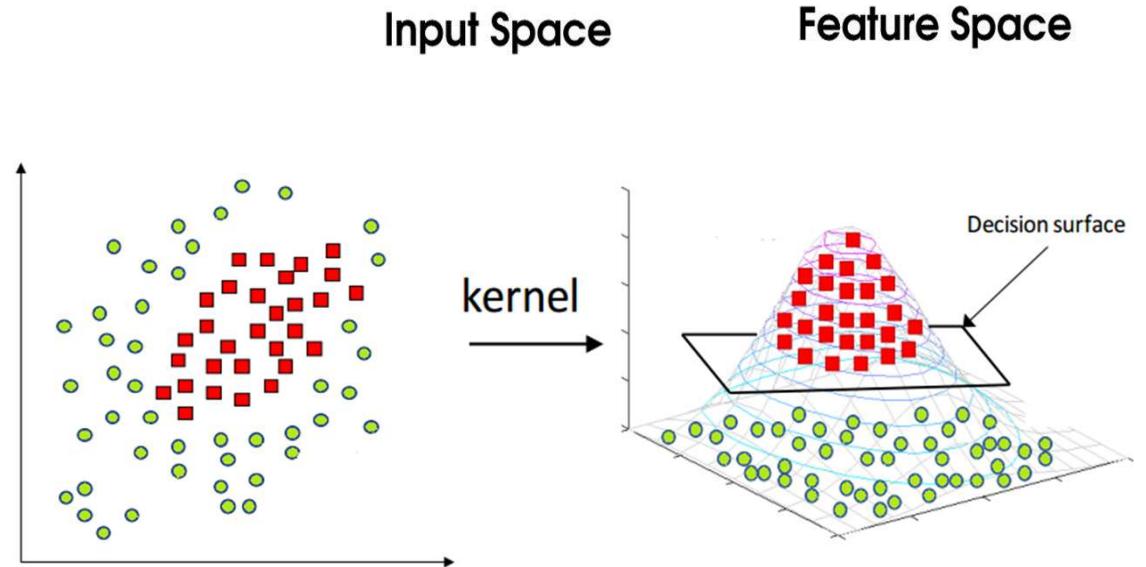
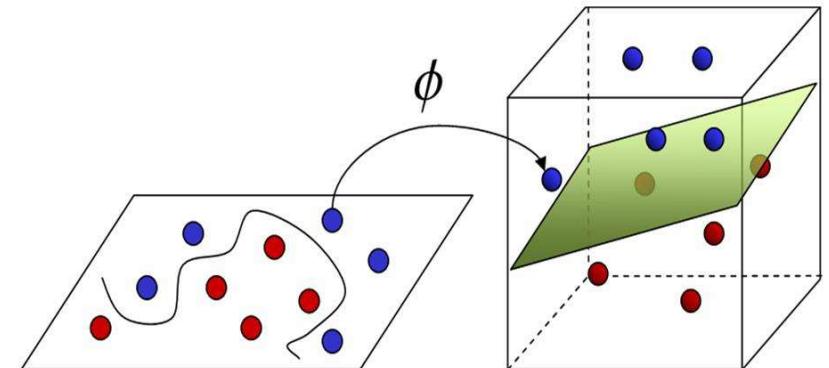
- Data 를 High Dimension feature space 에 mapping 하여 separate
- Support Vector (도록 경계에 위치한 sample)를 이용하여 Data 사이를 분리하는 hyper-plane 유추
- 장점 – High Dimensional Space 에서 정확
- 단점 – Overfitting 되기 쉽다.  
No Probability Estimation (확률모델이 아님)



# Kernel Function

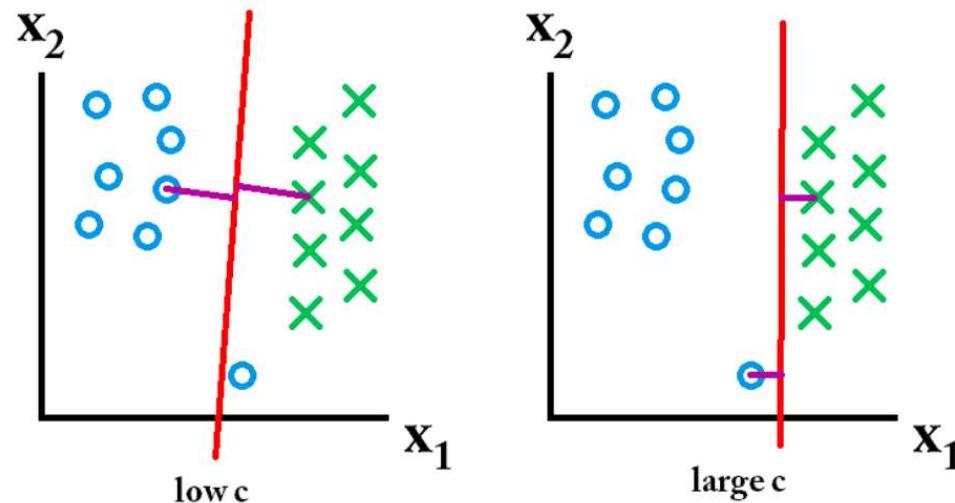
- Non-Linearly separable dataset  
→ kernel trick 이용하여 차원 변경
- Kernel 의 종류  
다항식  
가우시안 RBF  
Hyperbolic Tanget, etc

Principle of Support Vector Machines (SVM)



# Regularization parameter C

- Larger C – less regularization (data 를 가능한 정확히 fitting, hard margin 분류)
- Lower C – more regularization (generalization 을 위해 error 허용, soft margin 분류)



## 실습: Logistic Regression 과 동일한 Dataset 사용

1. 특정 사용자가 구매를 할지 여부를 예측 (구매: 1, 구매않음: 0)
2. Dataset 구성  
사용자 id, 성별, 연령, 추정급여, 구매여부  
이중 연령, 추정급여 두가지 feature 를 이용하여 구매여부 예측
3. Train / Test dataset 은 8 : 2 로 분리
4. Feature Scaling 실시 (standard scaling)
5. Model evaluation by Confusion Matrix, f1-score

# Ensemble Learning

## Random Forest/Gradient Boosting

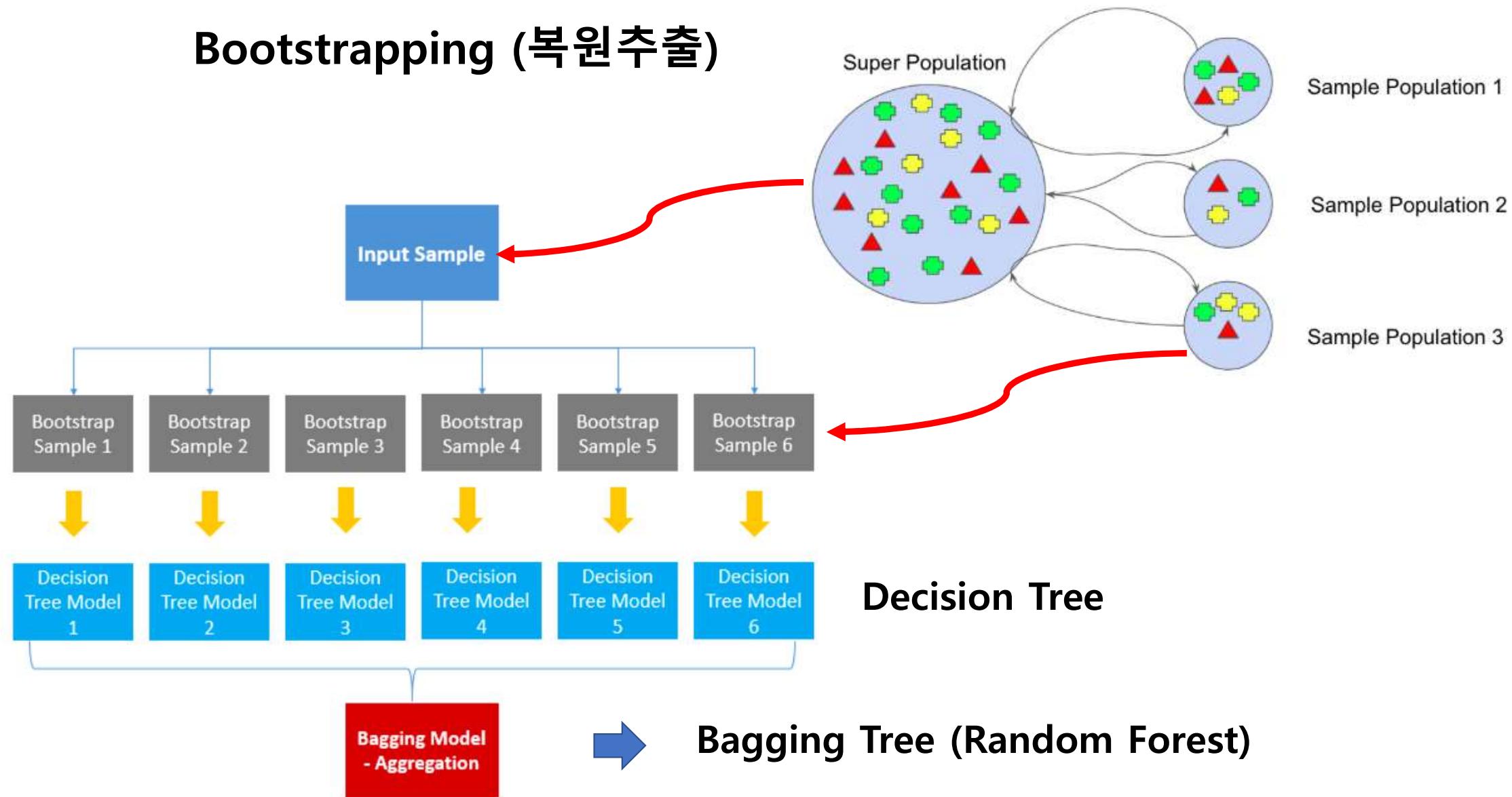
# What is Ensemble Learning ?

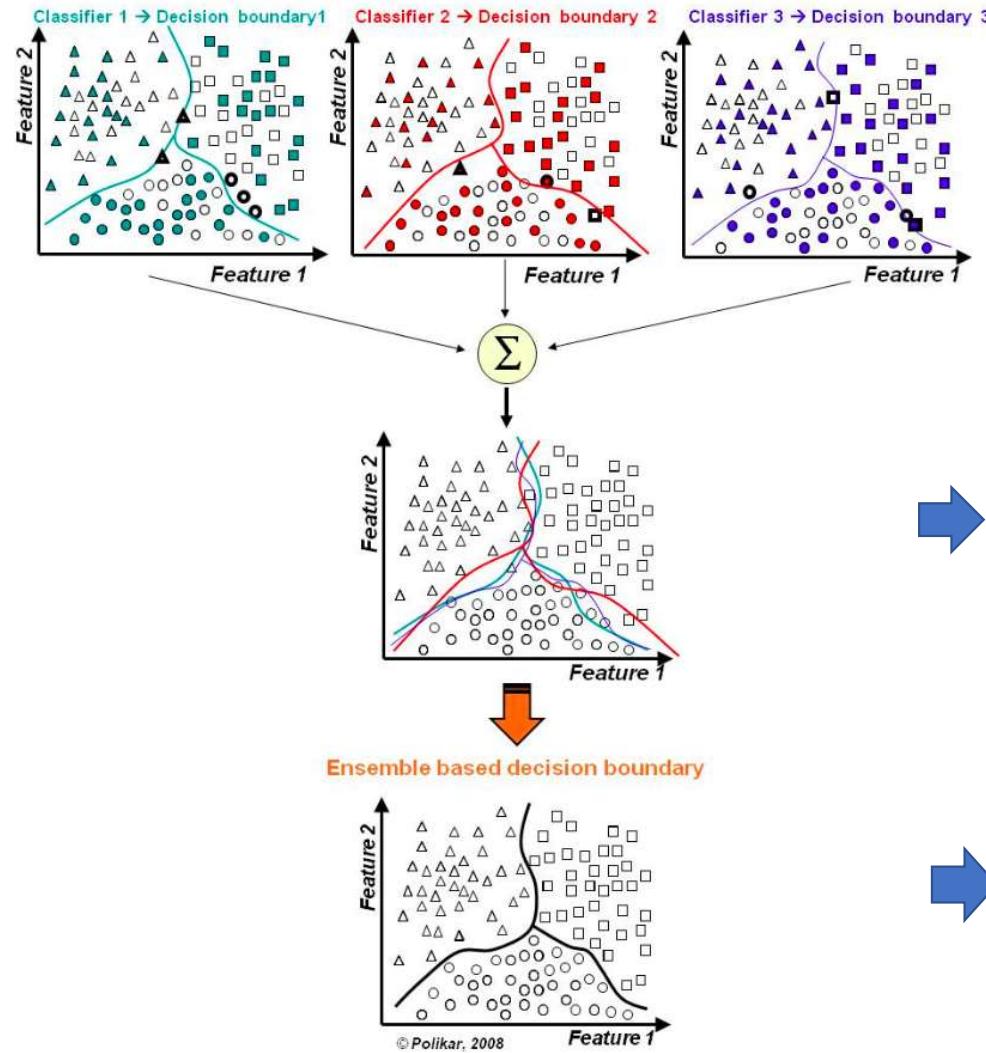
- 다수의 약한 학습기 (weak learner) 를 조합(Ensemble) 하여 더 높은 성능  
추출
- Decrease Variance by **Bagging** (Bootstrap Aggregating)
- Decrease Bias by **Boosting**

# What is Bagging (Bootstrap Aggregating) ?

- Training sample 의 sub-set 을 무작위로 추출하여 classifier 훈련
- Bootstrap – 중복을 허용하는 random sampling 방법 (통계학)  
→ b 개의 independent training set 을 평균하면,  
 $\text{variance} \rightarrow \frac{\sigma^2}{b}$  로 감소, mean → 동일
- Aggregating – voting for classification
- **Random Forest** 가 대표적인 method

# Bootstrapping (복원추출)





## Decision Trees

## Majority Voting

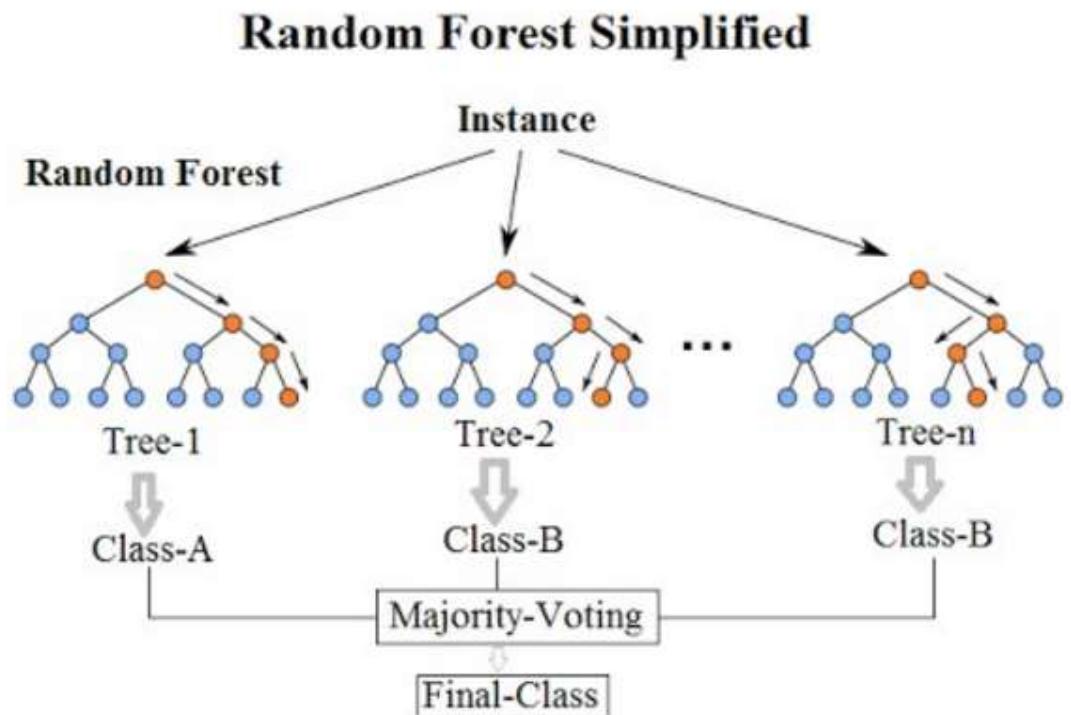
## Random Forest

# Bagging Tree 알고리즘

1. training data에서 random sampling하여  $X_b$ ,  $y_b$ 를 고름  
(ex. 1,000 개의 data 중 100 개 sampling)
2.  $X_b$ ,  $y_b$ 를 이용하여 **ID3 알고리즘**으로 **decision tree** 구성
3.  $B$  개의 tree가 만들어질 때까지 1, 2 반복
4.  $B$  개의 모든 tree를 이용하여 classification한 후 majority vote로 결정
5. Bias를 유지하며 Variance를 줄인다.  
→ bagging은 random sampling에 의해 model을 fit 하므로 **bias (model complexity)**가 커지지 않으면서 **variance**를 줄일 수 있는 특징이 있다.

# Random Forest

- Bagging Tree  
→ Random Forest로 발전
- **Randomly Choose Attributes**
- bootstrapping에 의한 복원추출  
+  
**attribute의 random 선택**에 따른  
independent trees 구성



# Random Forest Algorithm

1. Decision Tree 에 포함될 attribute 들을 random 하게 선정

→ 모든 attribute 를 가지고 Tree 를 만들 경우 매우 강한 attribute 가 모든 tree 에 항상 포함되는 것을 막기 위한 방법

(ex. 30 개 attribute 중 10 개만 random selection)

→ 좀 더 Random 하고 독립적인 classifier (Tree) 들을 생성시킬 수 있으므로 Random Forest 로 명명

2. Tree-based model 이므로 white box 특징을 유지

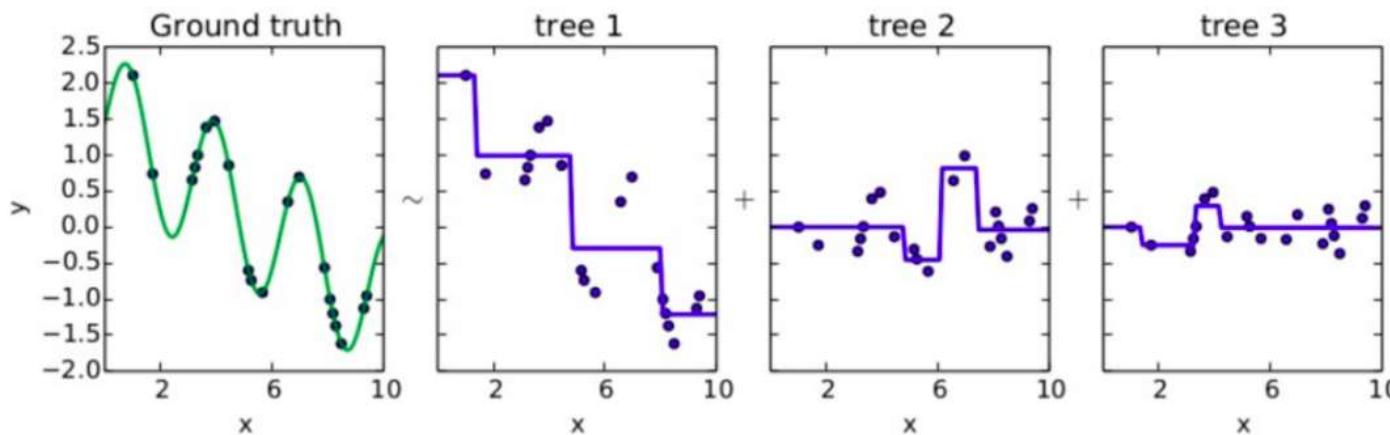
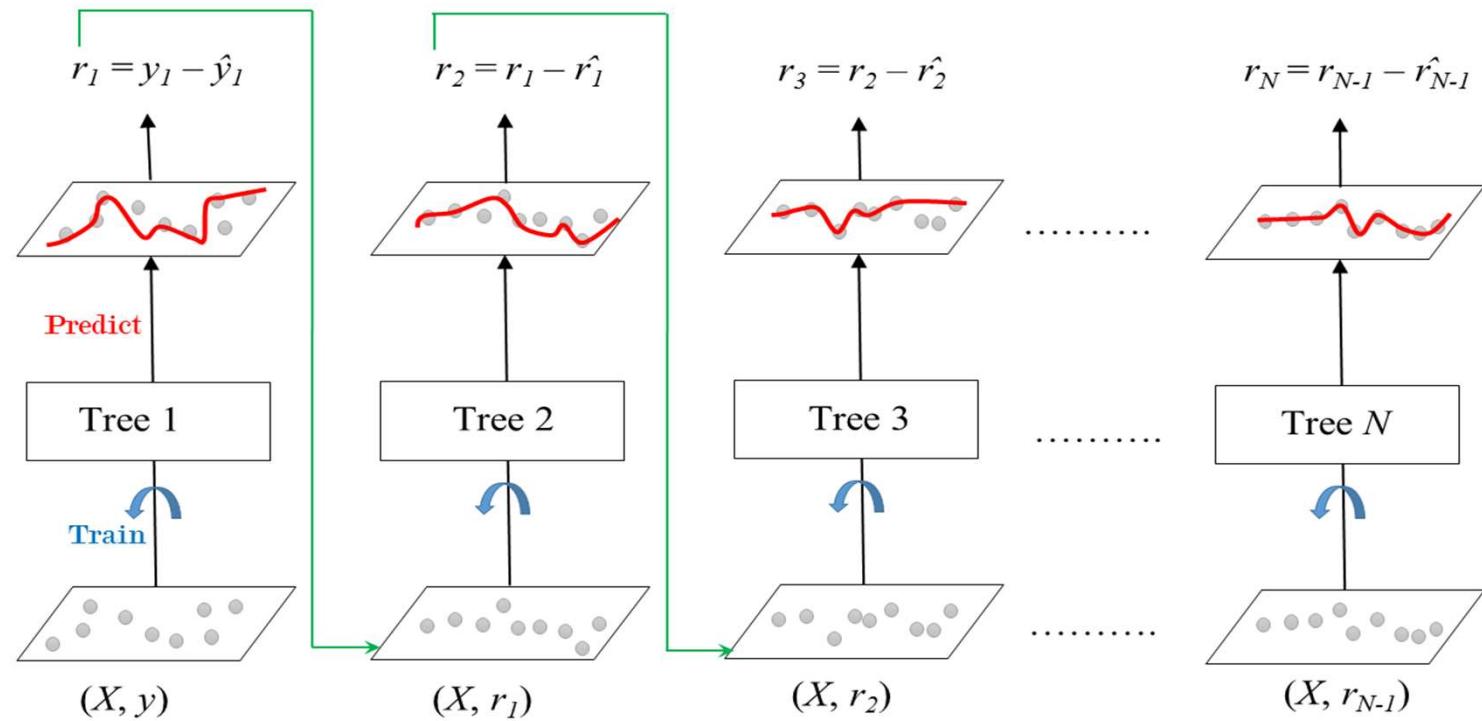
3. High prediction accuracy
4. 병렬적으로 생성 가능하므로 속도가 빠르다
5. 각 tree 는 매우 deep 하게 생성된다.  
→ 인위적인 prune 을 하지 않으나 Ensemble 을 하면 low bias,  
low variance 가 된다.

# What is Boosting ?

- Misclassified data 에 더 높은 weight 를 부여하여 다음 번 model 의 sampling 에 포함될 확률을 높이는 것
- 다수의 weak learner (small decision tree) 를 훈련 시켜 majority voting 하는 것은 Bagging 과 동일
- **AdaBoost, Gradient Boost (XGBoost)** 가 대표적인 method

# Gradient Boost 알고리즘

- Weak learner - random choice 보다 약간 더 나은 성능의 모델  
→ Decision Tree 사용
- 기존의 weak learner 가 생성한 residual error 를 감소시키는 추가 tree 를 더 이상의 감소 효과가 없을 때까지 생성
- Gradient Descent (경사하강법)에 의해 loss function (ex. MSE) 을 optimize



- 이전 tree에서 발생한 잔차 (residual error)를 next tree에서 보정

- 실습: Social\_Network\_Ads data 를 이용

1. Random Forest 를 이용한 분류
2. Gradient Boosting 을 이용한 분류

# Overfitting 방지 기법 (Regularization)

- Linear Regression (선형회귀)

$$J(W) = MSE_{train}(W) + \lambda \Omega(W)$$

손실함수              Train loss

$\lambda$  : regularization 강도  
 $\Omega$  : regularizer

대표적 regularizer

1. L2 regularization :  $\Omega(W) = \|W\|_2$  (ridge regularization)
2. L1 regularization :  $\Omega(W) = \|W\|_1$  (lasso regularization)
3. Elastic net regularization : L1 + L2

# Overfitting 방지 기법 (Regularization)

- Logistic Regression (이진 분류)

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m \left[ y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \lambda \frac{1}{2m} \sum_{j=0}^m \theta_j^2$$

↓  
regularizer

Binary cross-entropy 함수

$\lambda$  : regularization 강도

요점 – weight( $\theta$ ) 를 의도적으로 더해주면 optimization 은 더해준 만큼의 weight( $\theta$ ) 영향을 줄여주는 방향으로 진행

# Generalization

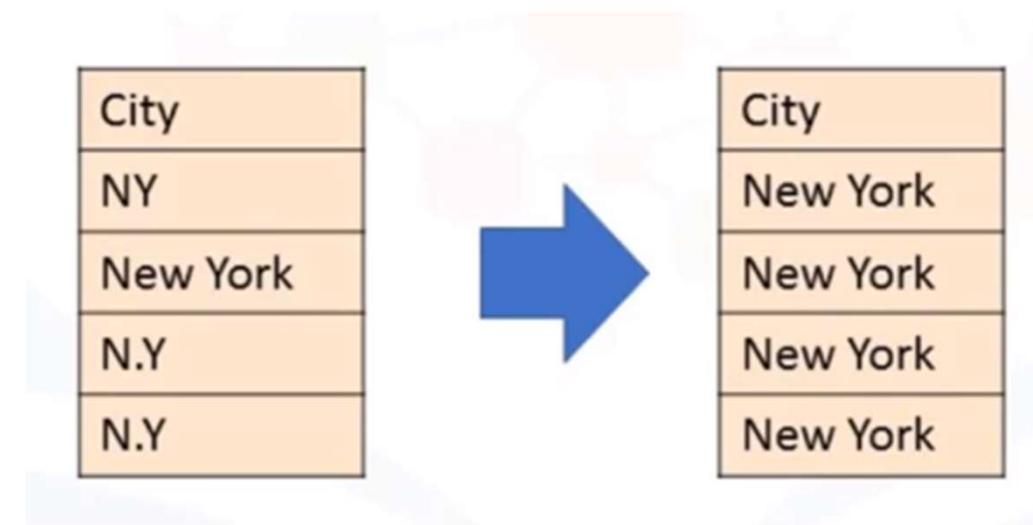
- No Free Lunch Theorem
  - 모든 문제를 하나의 model로 해결할 수 없다. 즉, 하나의 문제에 잘 맞는 모델이 다른 문제에도 잘 맞는 것은 아님. 따라서, 다양한 model을 try하여 가장 잘 맞는 모델을 선정.
- Speed, accuracy, complexity의 trade-off로 model과 알고리즘 선택
  - 모든 data를 구할 수 없고, True Function을 아는 것은 불가능하므로 machine learning은 단지 확률과 통계에 기반하여 approximate하는 것
- 간단한 model → 복잡한 model 순으로 Try
  - Bias-variance Trade-off 원칙을 잊지 말 것

# Feature Engineering

# 머신러닝을 위한 Feature Engineering

- ✓ Missing Values 처리
- ✓ Data Formatting
- ✓ 편향(skewed data) 처리
- ✓ Data Normalization
- ✓ Binning
- ✓ categorical 변수의 수치화

## Data Formatting



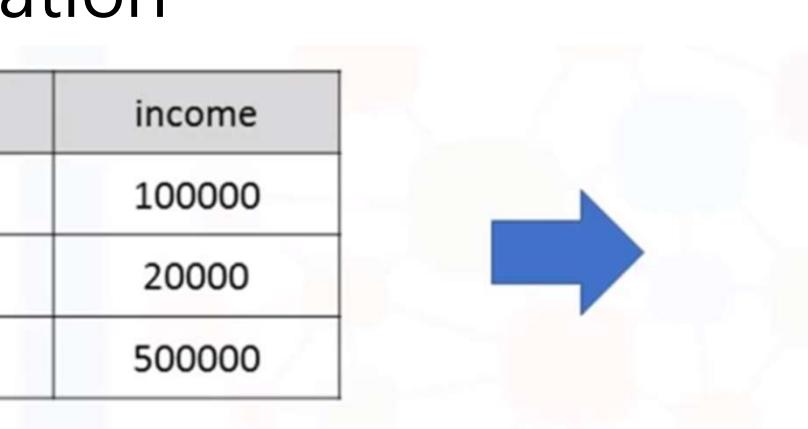
The diagram illustrates a transformation process. On the left, there is a table with a single column labeled "City". It contains five rows with the following values: "NY", "New York", "N.Y", and "N.Y". A large blue arrow points to the right, indicating a transformation or mapping. On the right, there is another table with a single column labeled "City". This table has five rows, all of which contain the value "New York".

City
NY
New York
N.Y
N.Y

City
New York
New York
New York
New York

## Data Normalization



The diagram illustrates a transformation process. On the left, there is a table with two columns: "age" and "income". It contains three rows with the following data: (age 20, income 100000), (age 30, income 20000), and (age 40, income 500000). A large blue arrow points to the right, indicating a transformation or mapping. On the right, there is another table with two columns: "age" and "income". The "age" column has values 0.2, 0.3, and 0.4. The "income" column has values 0.2, 0.04, and 1 respectively, corresponding to the normalized age values.

age	income
20	100000
30	20000
40	500000

age	income
0.2	0.2
0.3	0.04
0.4	1

Binning

price
13495
16500
18920
41315
5151
6295
...



price	price-binned
13495	Low
16500	Low
18920	Medium
41315	High
5151	Low
6295	Low
...	...

Categorical 변수의 수치화

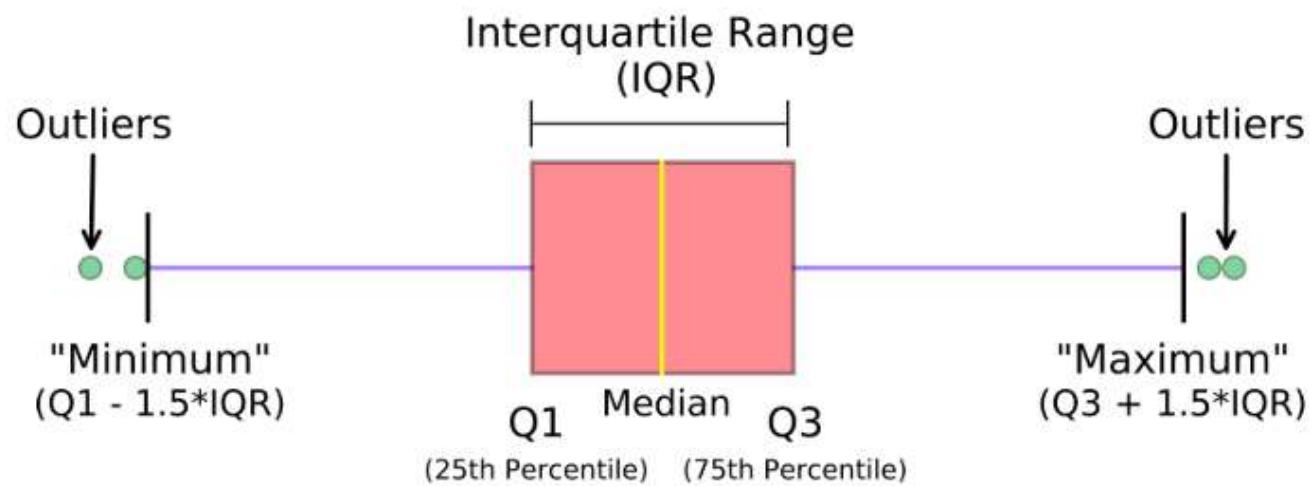
fuel
gas
diesel
gas
gas



gas	diesel
1	0
0	1
1	0
1	0

# Boxplot

- 4 분위수



# 실습: Titanic 호 data 를 이용한 Feature Engineering 과 Modeling

- Survival 예측 model 작성
- Feature Engineering
- Grid Search 를 통한 최적 model 및 parameter tuning

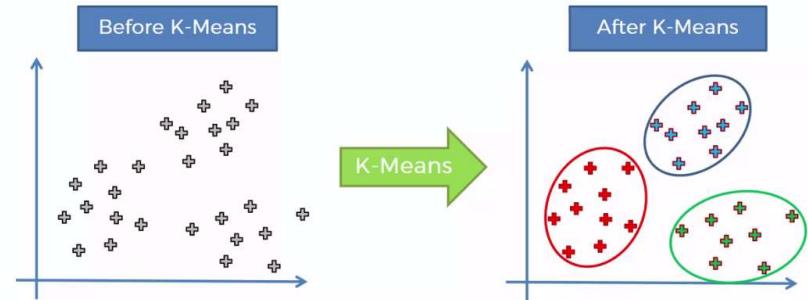
# Clustering

# Clustering 이란 ?

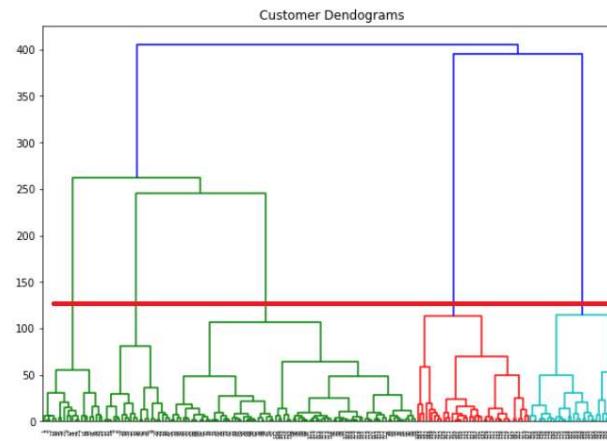
- 비슷한 object 들끼리 모으는 것
- label data 가 없는 것이 classification 과 차이  
    → unsupervised machine learning
- 적용 사례
  - 고객의 구매 형태별 분류
  - 고객의 취향에 맞는 책, 동영상 등의 추천
  - 신용카드 사용의 fraud detection
  - 뉴스 자동 분류 및 추천
  - 유전자 분석 등

# Clustering 알고리즘의 종류

- K-Means Clustering

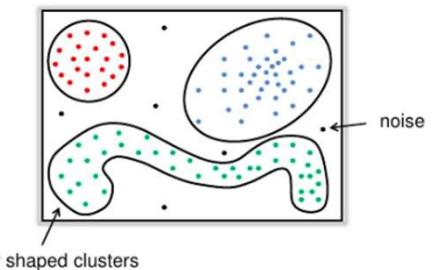


- Hierarchical Clustering (dendrogram)



- Density-based Clustering (DBSCAN)

**DBSCAN**  
Density based spatial clustering of applications with noise



# K-Means Clustering 알고리즘 – Distance 계산

Distance = Euclidean Distance

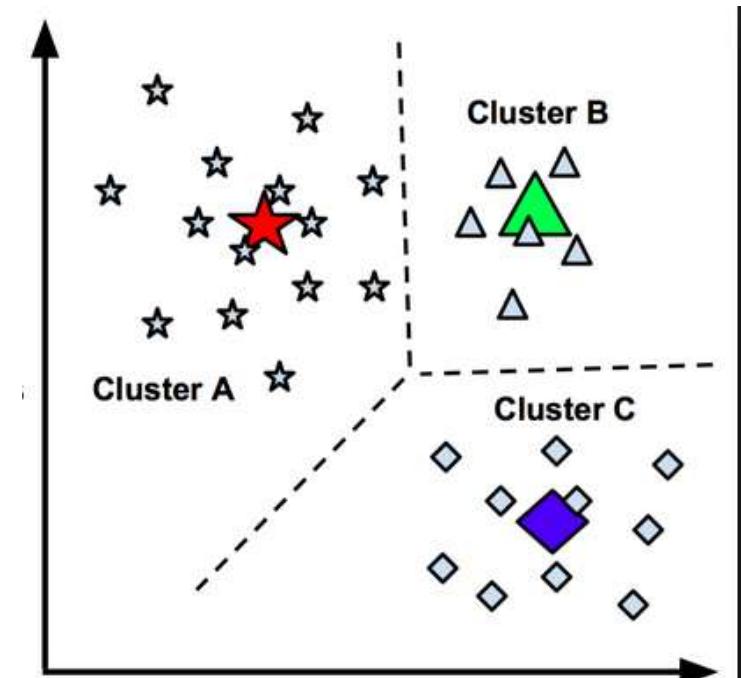
$$= \sqrt{\sum_{i=0}^n (x_{1i} - x_{2i})^2}$$

Ex)

고객	나이	수입	교육
1	54	190	3
2	50	200	8

$\rightarrow x_1$   
 $\rightarrow x_2$

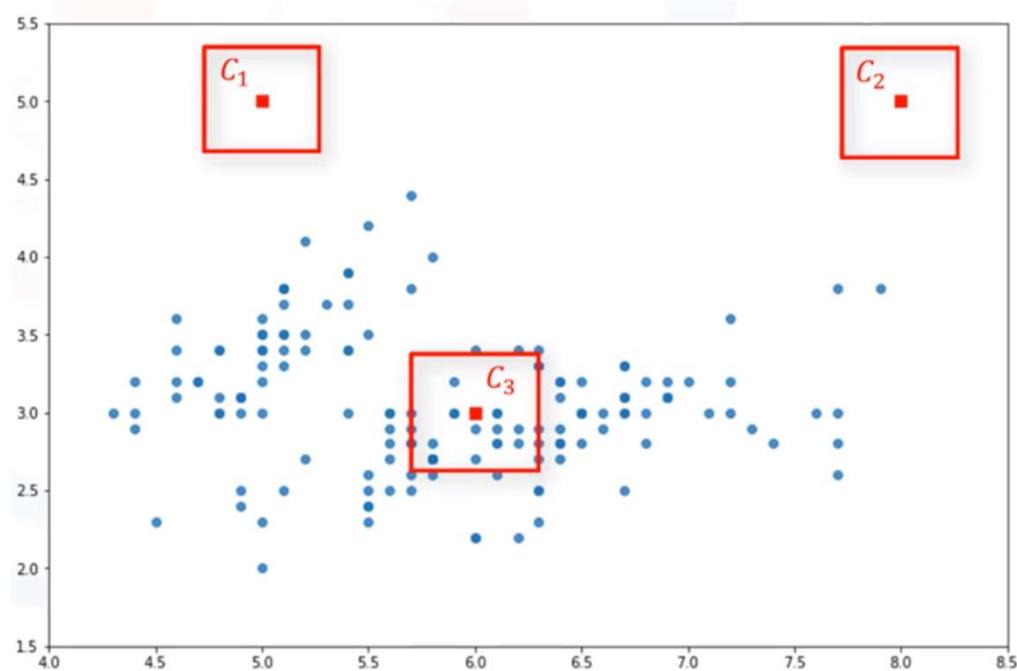
$$\text{Distance}(x_1, x_2) = \sqrt{(54 - 50)^2 + (190 - 200)^2 + (3 - 8)^2} = 11.87$$



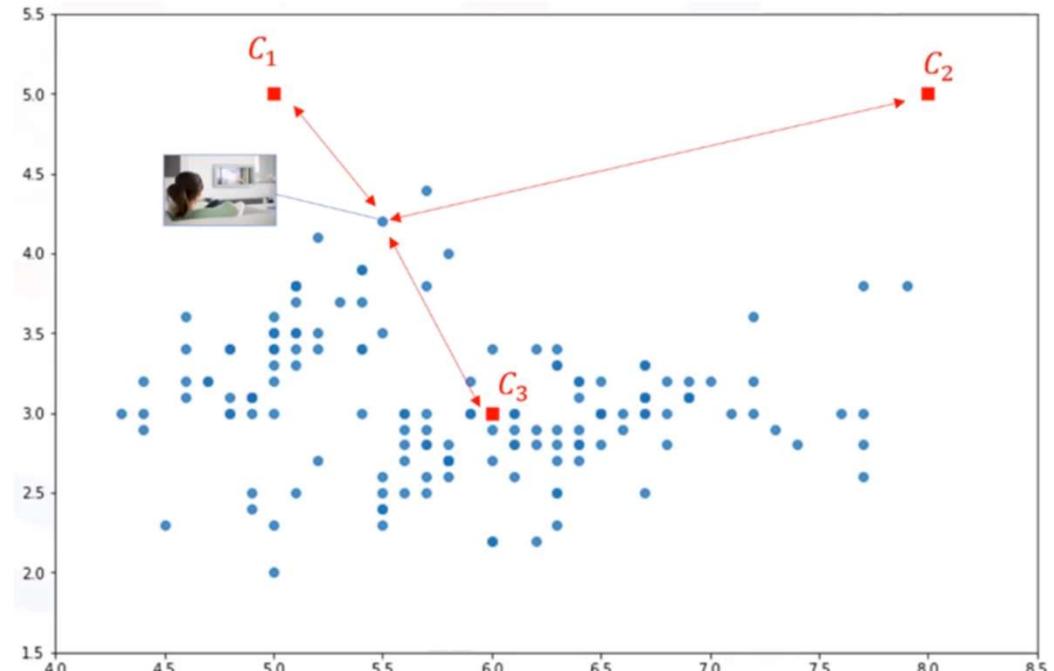
# K-Means Clustering 알고리즘

1. Random하게 k개의 centroid(중심점)를 정한다.
2. 각 centroid로부터 각 data point 까지의 거리를 계산.
3. 각 data point를 가장 가까운 centroid에 할당하여 cluster를 생성.
4. K centroid의 위치를 다시 계산
5. centroid가 더 이상 움직이지 않을 때까지 2-4 단계를 반복

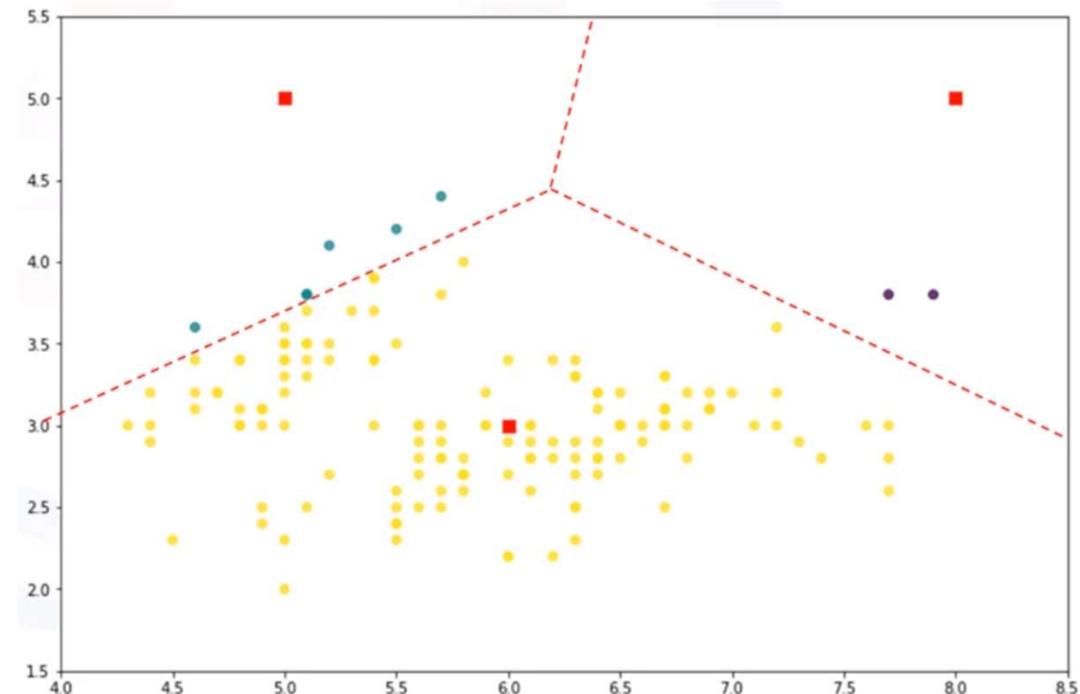
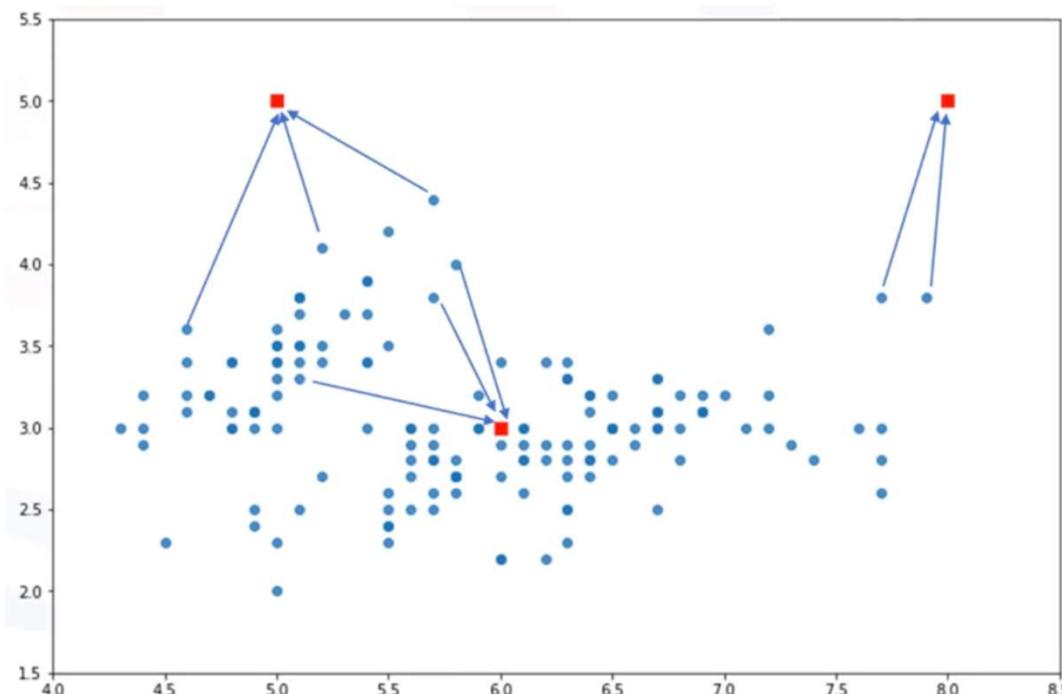
## 1. 임의의 centroid 선정 : k=3



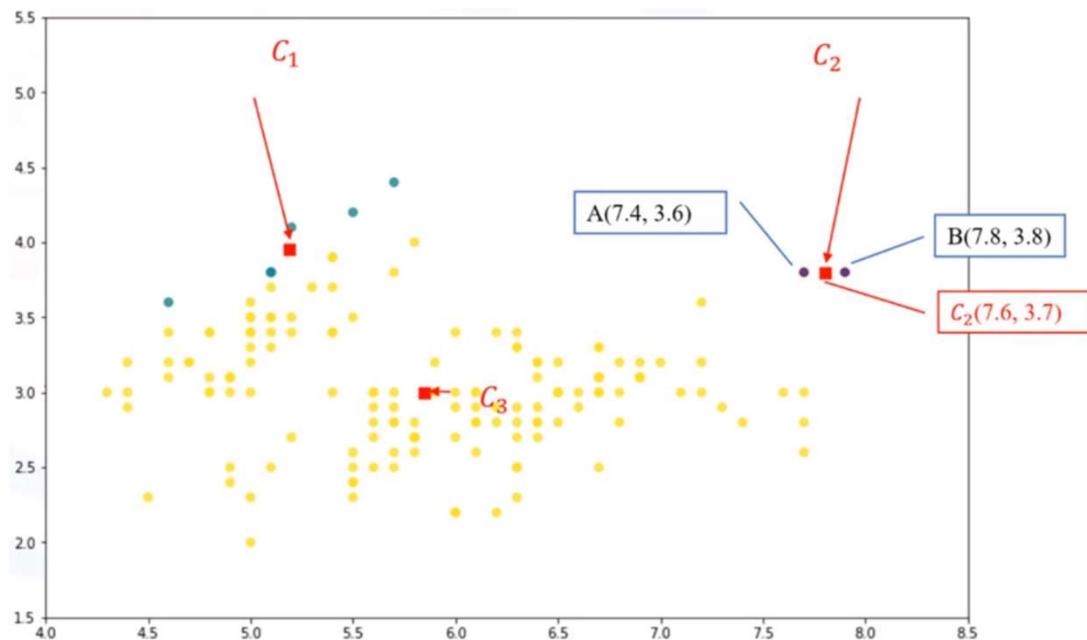
## 2. 거리 계산 for each data point



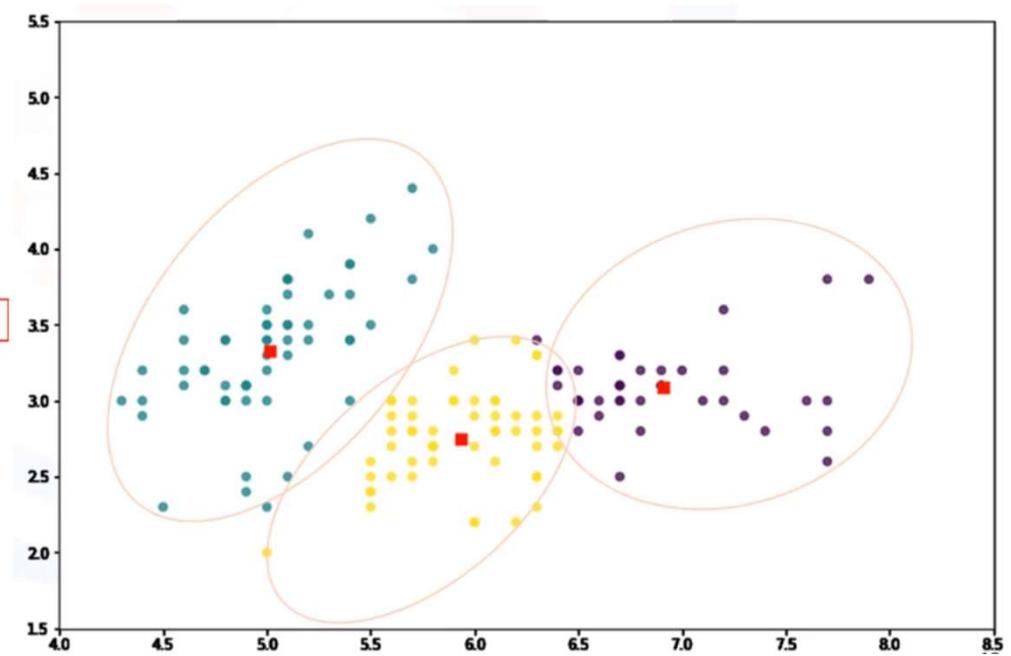
### 3. 가장 가까운 centroid 로 각 data point 할당



#### 4. 각 cluster 의 new centroid 계산



#### 5. Centroid 변화 없을 때까지 반복



# Choosing k



$K$  를 잘 정하는 것이 중요하다.

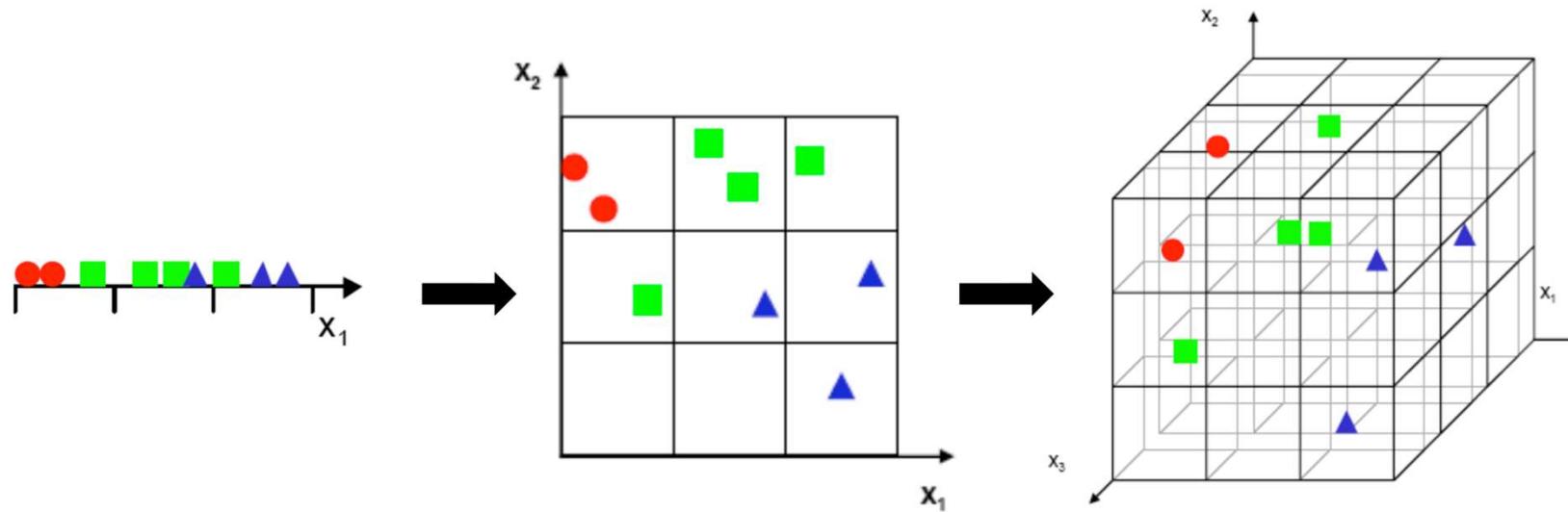
# 실습: K-Means Clustering

1. `sklearn.cluster.Kmeans`
2. Dataset : `sklearn.dataset.samples_generator.make_blobs` 사용
3. `Matplotlib` 을 이용하여 시각화

차원축소

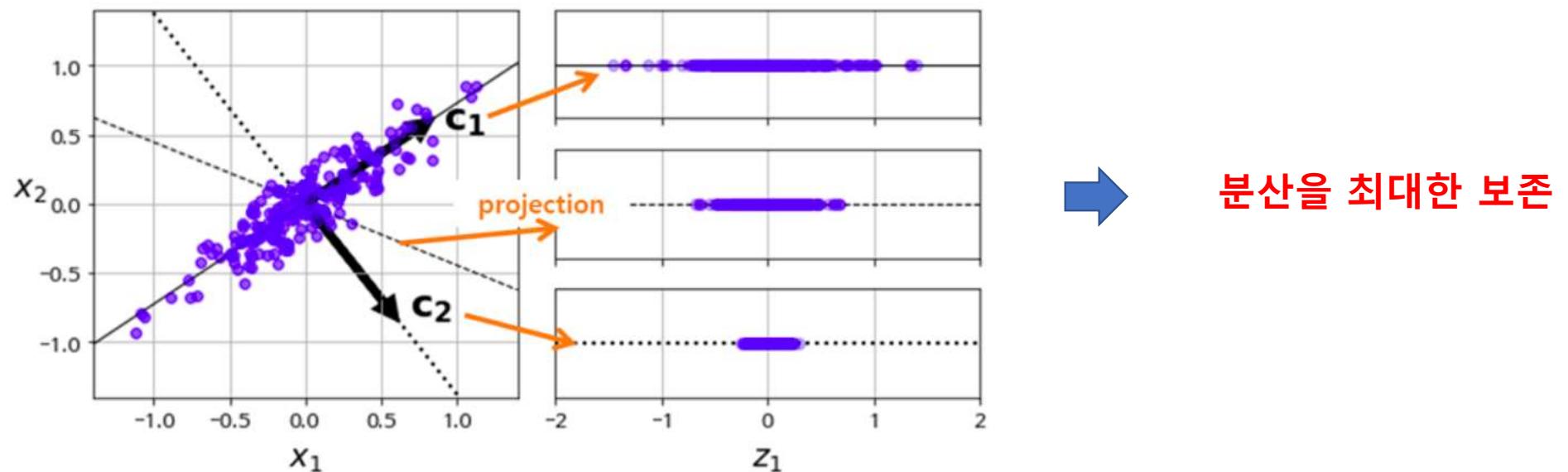
# 차원의 저주 (Curse of Dimensionality)

- 차원이 증가함에 따라 vector 공간내의 space 도 증가하는데 데이터의 양이 적으면 빈공간이 많이 발생하여 예측의 정확도가 떨어진다.
- 유사한 성격의 feature (예, 키, 신장, 앉은키, 기온, 수도관 동파, 빙판길 미끄러짐 사고 등) 는 하나의 새로운 feature로 성분을 합칠 수 있음



# PCA (Principal Component Analysis) - 주성분 분석

- 선형대수학의 SVD 를 이용하여 분산이 최대인 축을 찾음
- 데이터의 **분산(variance)**을 최대한 보존하면서 서로 직교하는 새 축을 찾아, 고차원 공간의 표본들을 선형 연관성이 없는 저차원 공간으로 변환



## • 특이값 분해 (Singular Value Decomposition, SVD)

- 모든  $m \times n$  행렬에 대해 적용 가능 (고유값 분해를 모든 행렬에 일반화)
- $m \times n$  행렬  $A$  는 다음과 같은 형태의 세가지 행렬의 곱으로 분해할 수 있다.

$$A_{m \times n} = U_{m \times m} \Sigma_{m \times n} V^T_{n \times n}$$



$U$  :  $AA^T$ 를 고유값 분해해서 얻어진  $m \times m$  직교행렬(orthogonal matrix)

$V$  :  $A^TA$ 를 고유값 분해해서 얻어진  $n \times n$  직교행렬(orthogonal matrix) - 주성분 column 들로 구성

$\Sigma$  :  $U, V$  를 고유값 분해해서 나오는 고유값(eigenvalue)들의 square root( $\sqrt{\lambda_i}$ )를 대각원소로 하는  $m \times n$  직사각 대각행렬로 그 대각원소들을  $A$ 의 특이값(singular value)이라 부른다

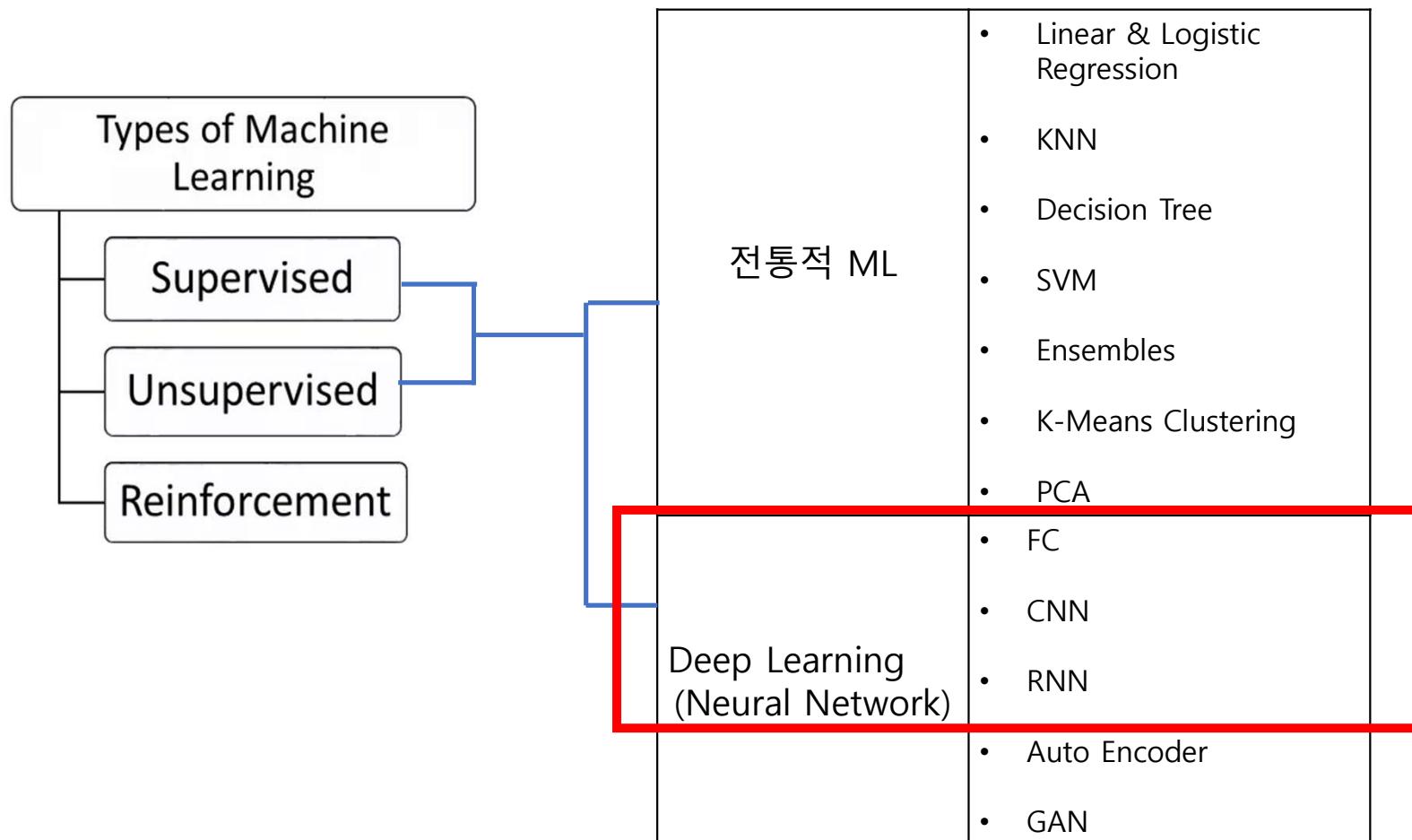
**u1σ1v1 : A 의 정보를 가장 많이 보여주는 성분 → Principal Component Analysis**

## 실습: PCA

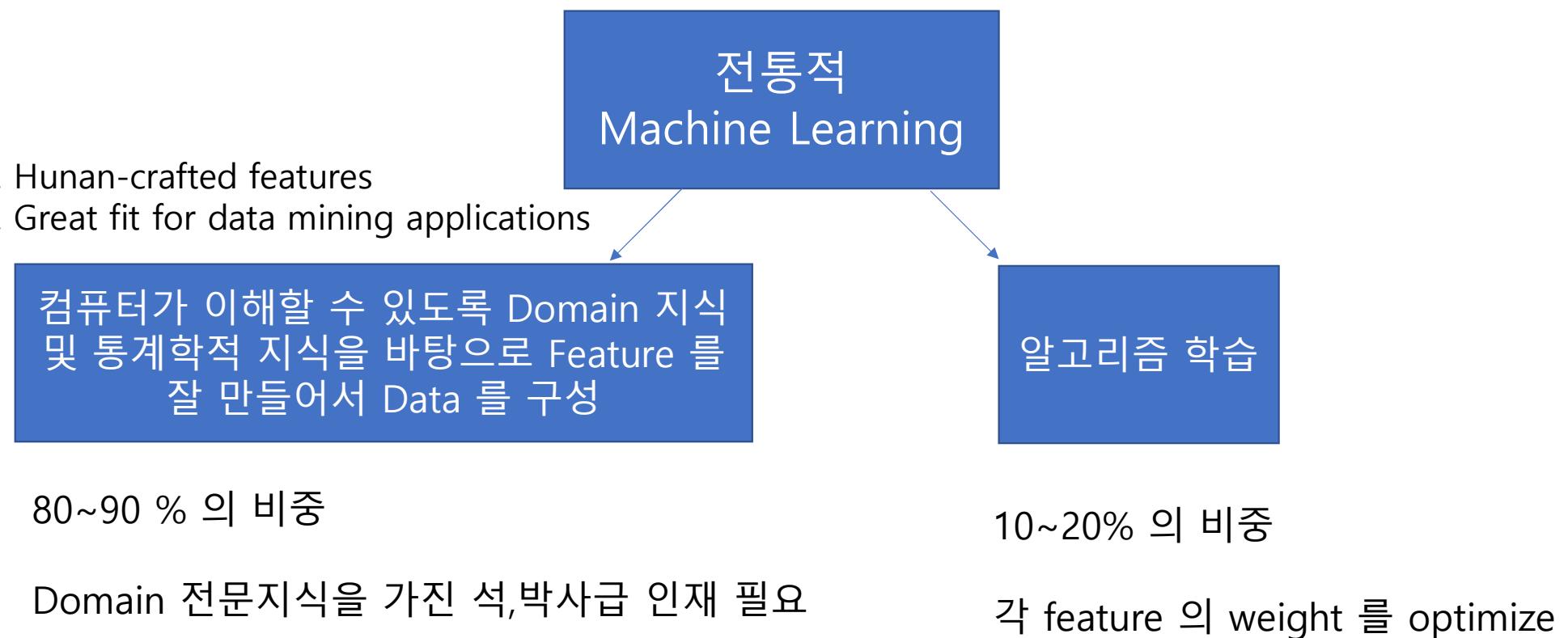
1. sklearn.decomposition 의 PCA 를 이용하여 차원 축소
2. Dataset : 통신회사 고객 이탈 (Churn) data 이용
3. 27 개의 feature 를 2 개로 reduce 한 후 2 개의 feature 를 이용하여 Logistic Regression
4. Matplotlib 을 이용하여 시각화

# Neural Network and Deep Learning

# Machine Learning 모델의 종류



# 전통적 Machine Learning 의 학습



# Deep Learning 의 학습

- 중요한 Feature 를 스스로 구분하여 weight 를 부여
  - 사람이 manually 정해준 feature 는 over-specified, incomplete 위험성 있고 작성에 많은 시간 소요
- 여러 층에 걸친 내부 parameter 를 스스로 학습
  - 적용하기 쉽고 빠르다.
- Raw data 를 거의 그대로 사용 – computer vision, 언어처리 등 (ex, image, sound, characters, words)
- Unsupervised, supervised learning 모두 가능
- 이미지 인식, 대화/언어 문제에 탁월한 성능

# Artificial Neuron (Perceptron)

구성요소:

**Pre-Activation :**

$$a(x) = b + \sum_i w_i x_i = b + w^T X$$

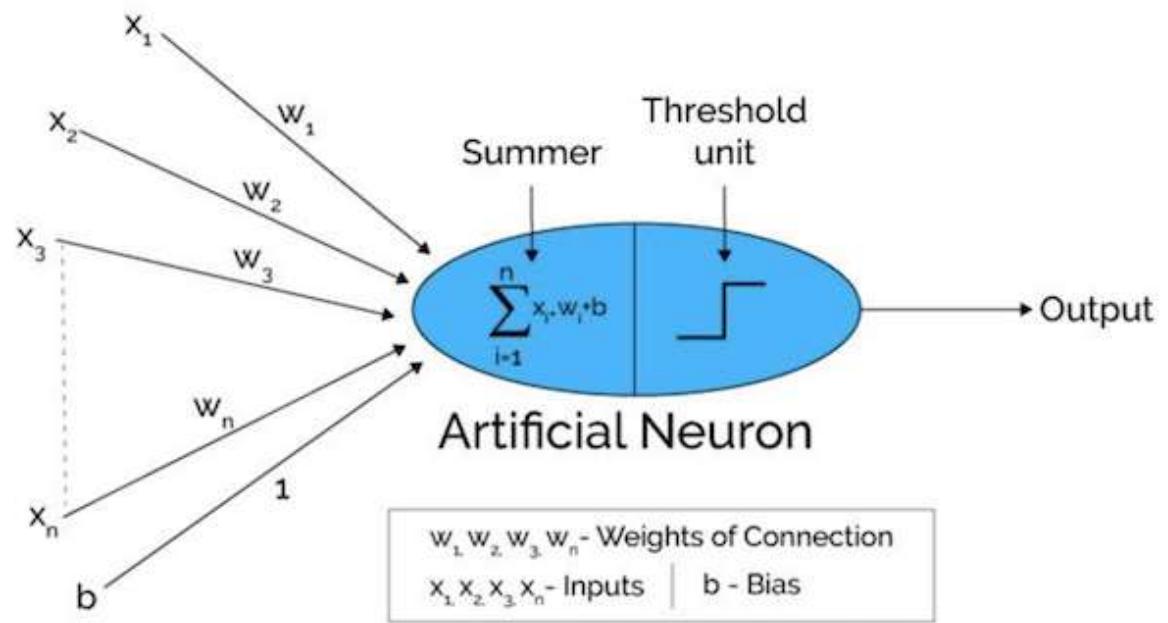
**Activation :**

$$h(x) = g(a(x)) = g(b + \sum_i w_i x_i)$$

w : connection weights

b : bias

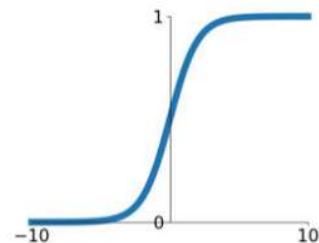
g : activation function



# Activation Functions

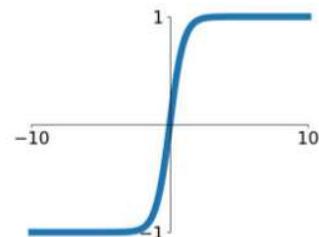
**Sigmoid**

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



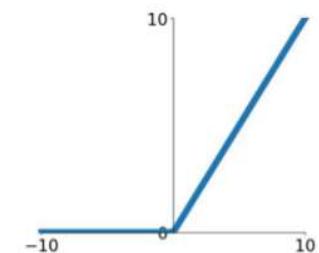
**tanh**

$$\tanh(x)$$



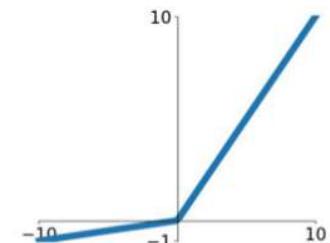
**ReLU**

$$\max(0, x)$$



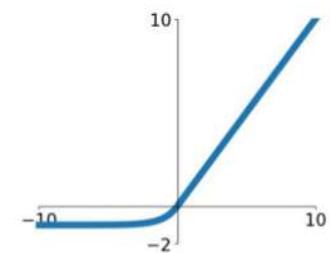
**Leaky ReLU**

$$\max(0.1x, x)$$



**ELU**

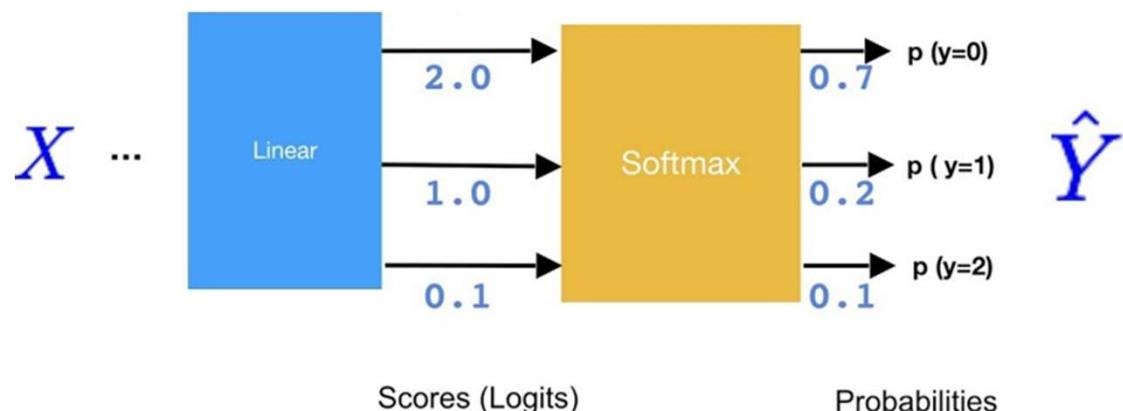
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



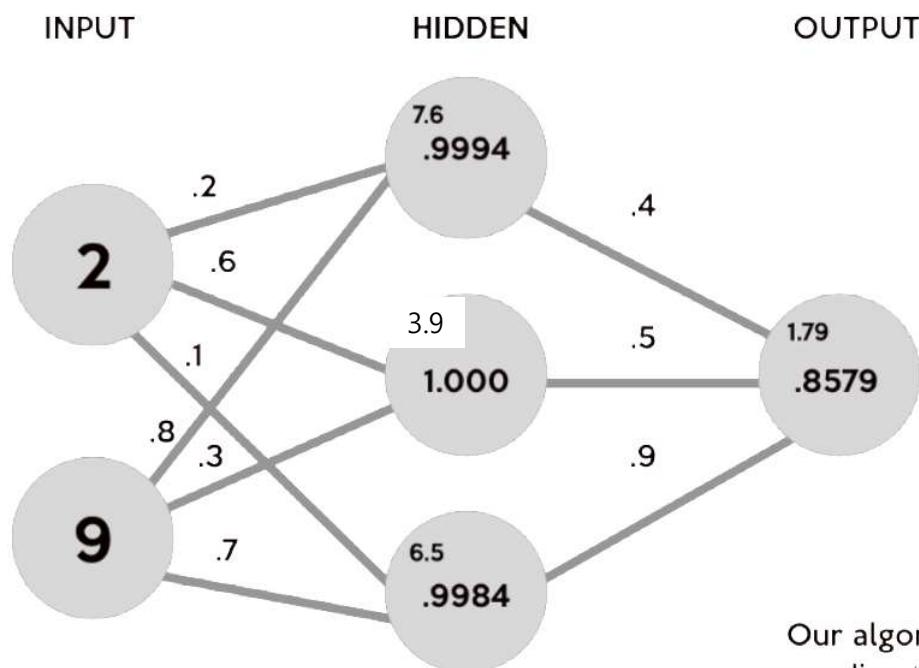
# Softmax Activation Function

- 출력값의 class 분류를 위하여 출력값에 대해 정규화 → 확률 분포 출력

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

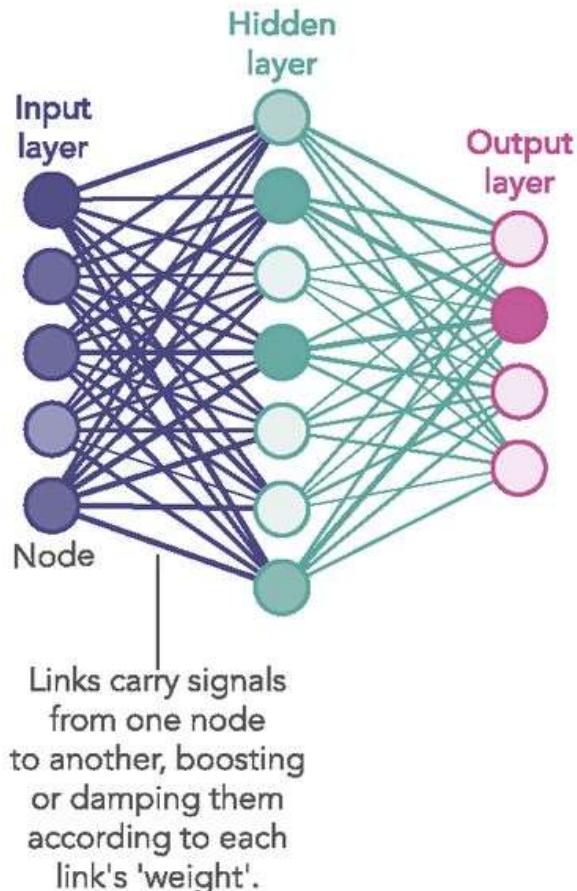


# Neural Network 의 작동 원리

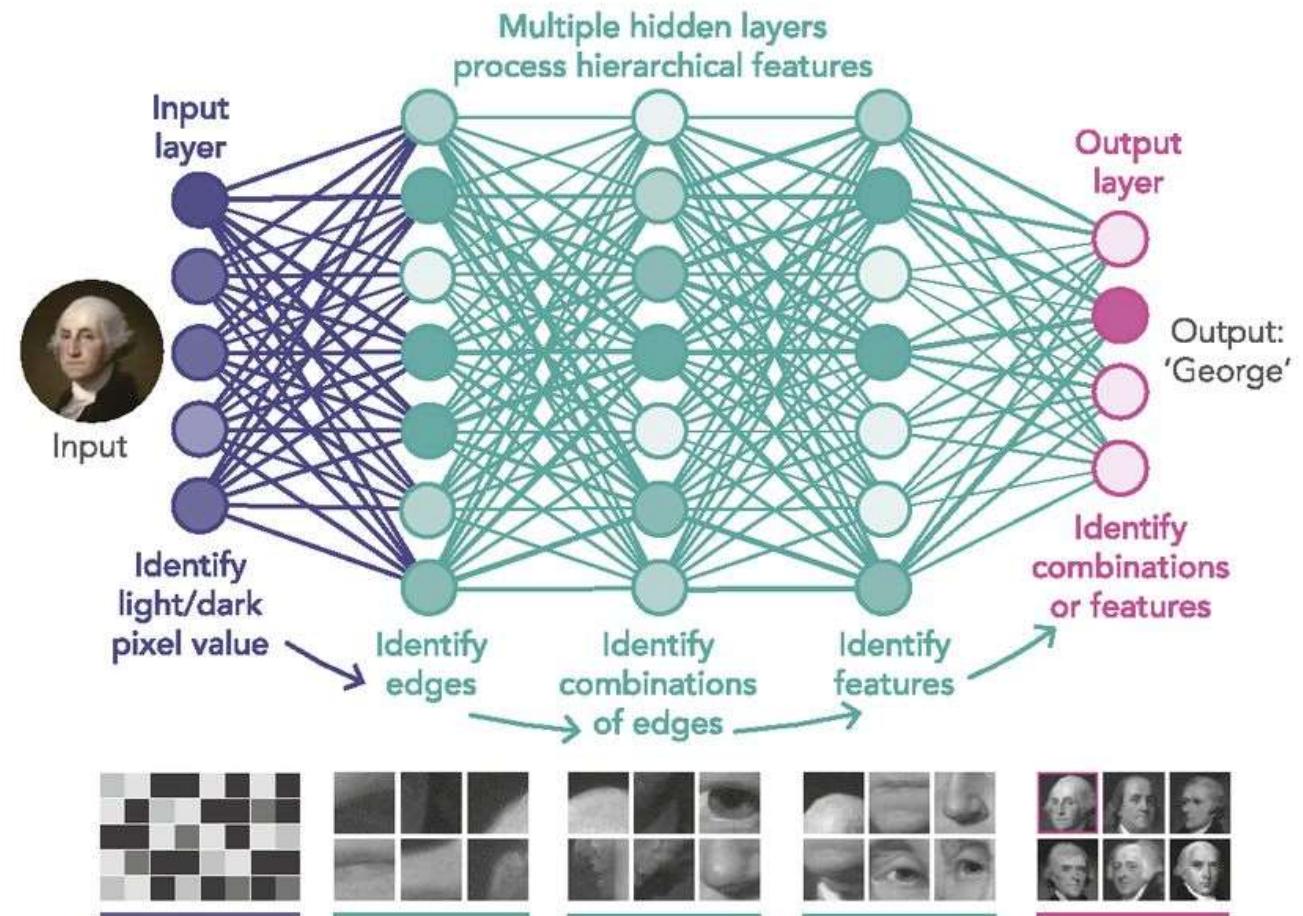


Our algorithm,  
according to the  
untrained (random)  
weights, produced .85  
as our test score  
result.

### 1980S-ERA NEURAL NETWORK



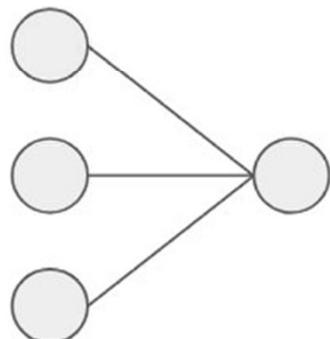
### DEEP LEARNING NEURAL NETWORK



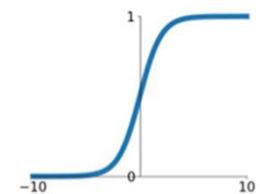
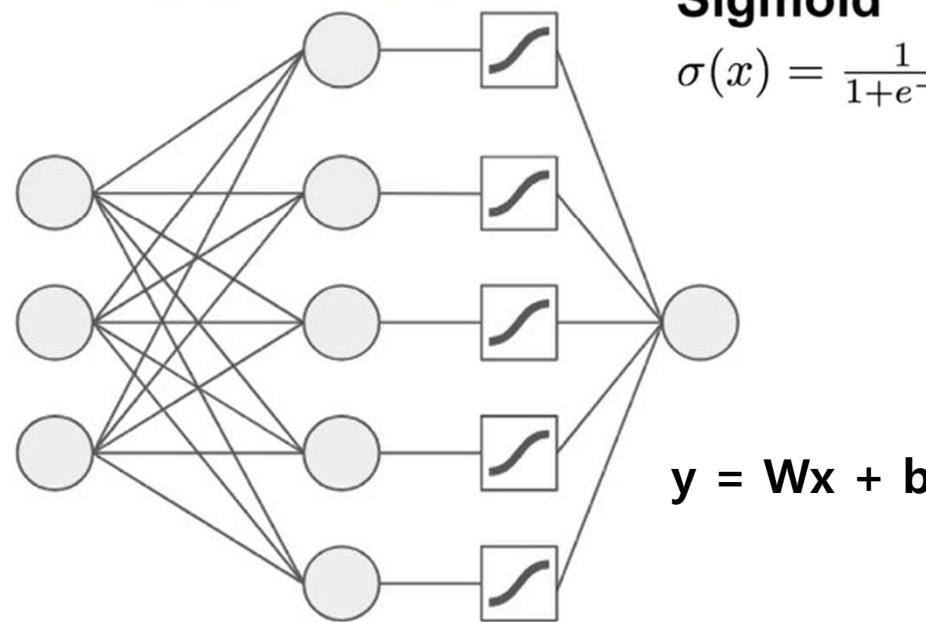
# Regression (Linear / non-Linear)

A. Linear-regression model

$$y = Wx + b$$

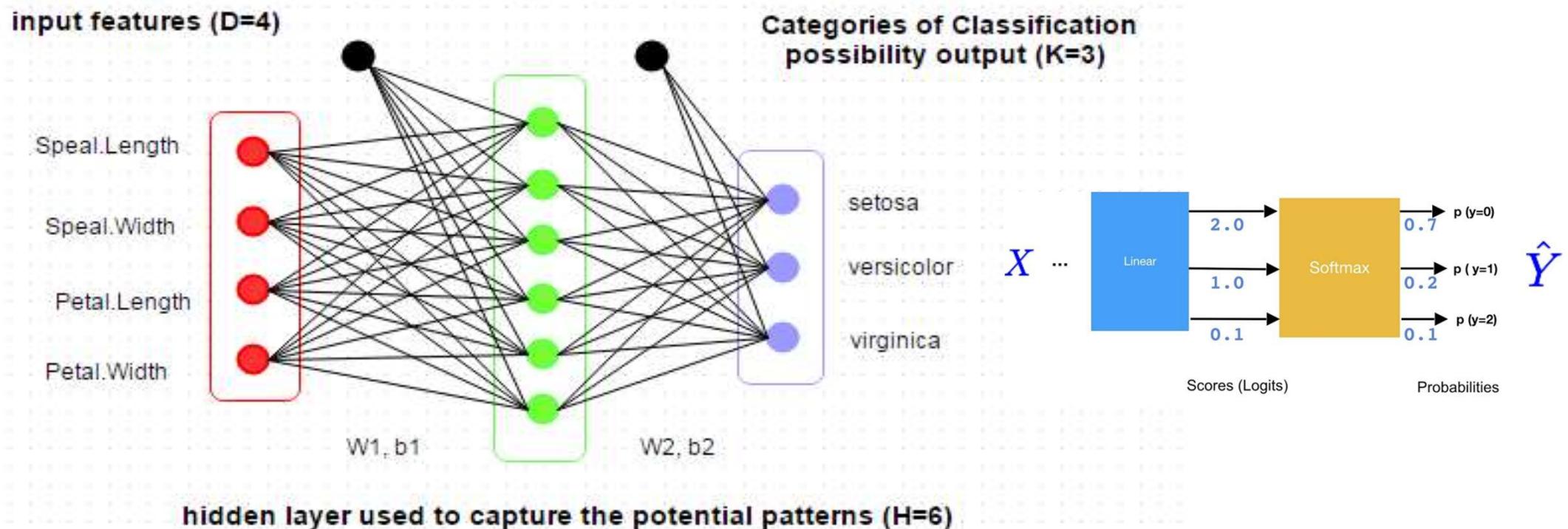


B. Two-layer neural network with nonlinear internal activation

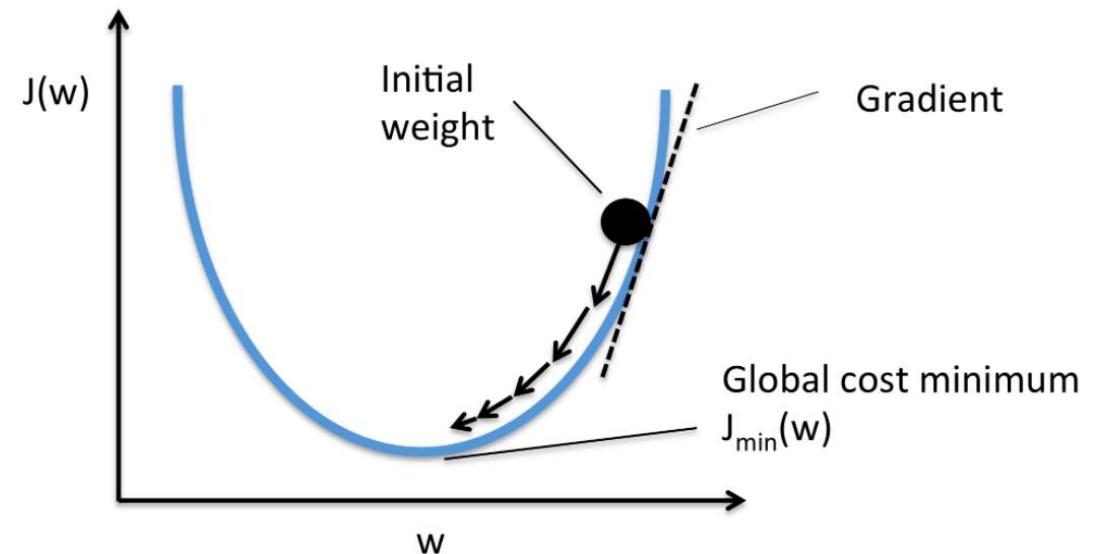
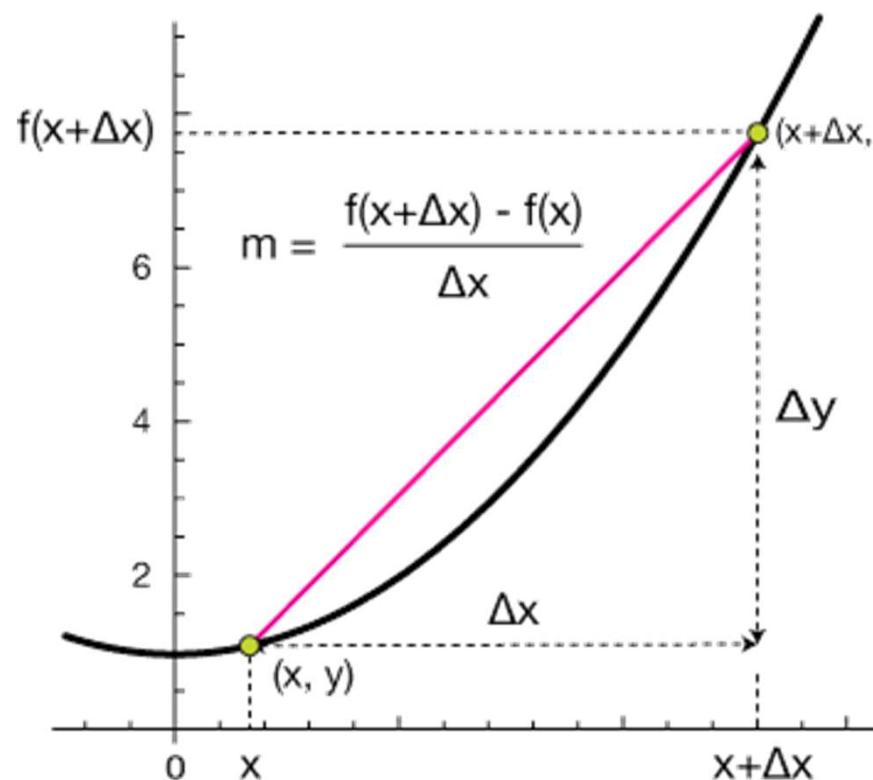


# Multi-Class Classification (Softmax)

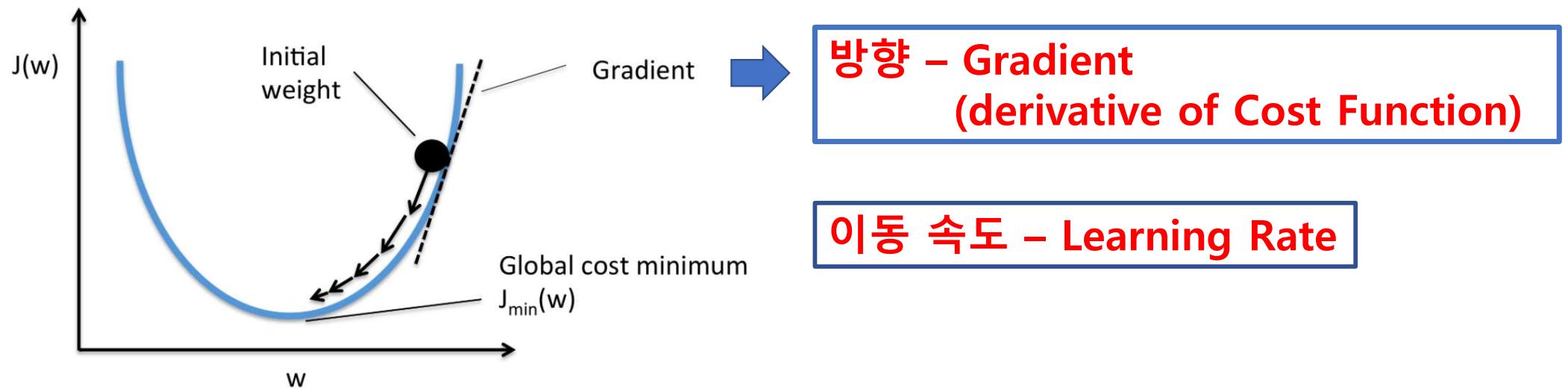
## Classification Example for IRIS data by DNN



# Derivative (도함수, 미분, 접선의 기울기)

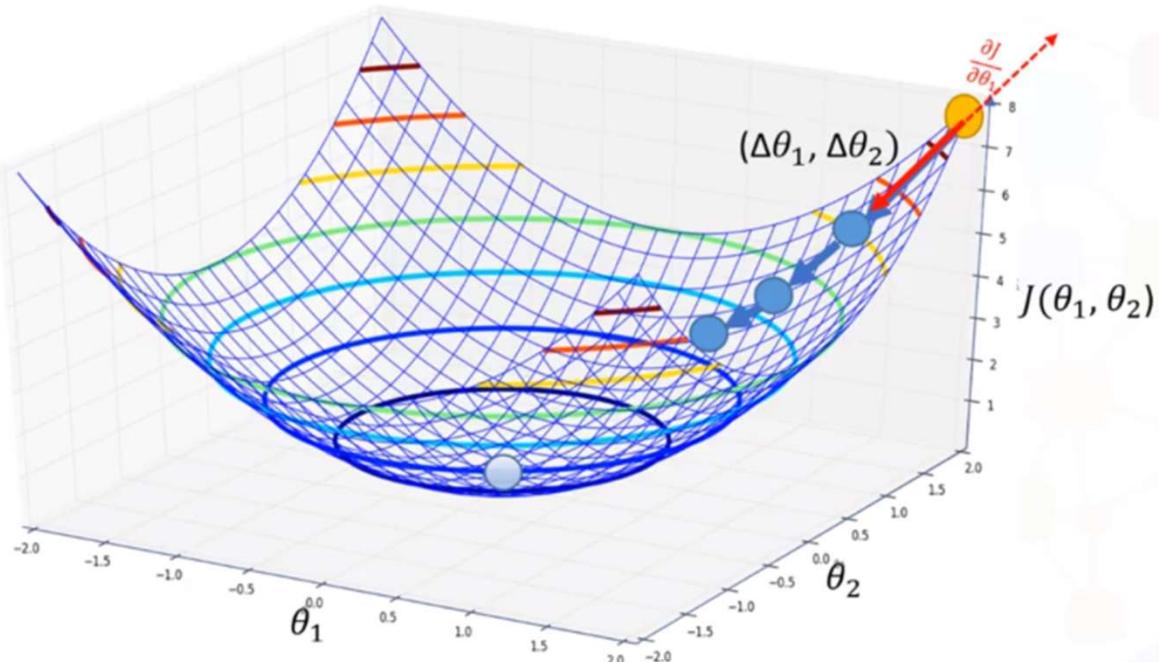


# Gradient Descent (경사하강법) Optimization



$$\text{New } W = \text{old } W - (\text{Learning Rate}) * (\text{Gradient})$$

Goal : Minimize  $J(\theta_1, \theta_2)$ ,  $y = \theta_1 X_1 + \theta_2 X_2$



$$\hat{y} = \sigma(\theta_1 x_1 + \theta_2 x_2)$$

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m y^i \log(\hat{y}^i) + (1 - y^i) \log(1 - \hat{y}^i)$$

1. Loss Function (손실함수) 의 derivative(slope) 계산
2. Step size (Learning Rate 계산)  
slope \* learning rate
3. Update the parameter (batch)
4. Repeat until slope = 0

# Gradient Descent for Linear Regression

- Hypothesis : 
$$h_{\theta}(X) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n \\ = \theta^T X$$

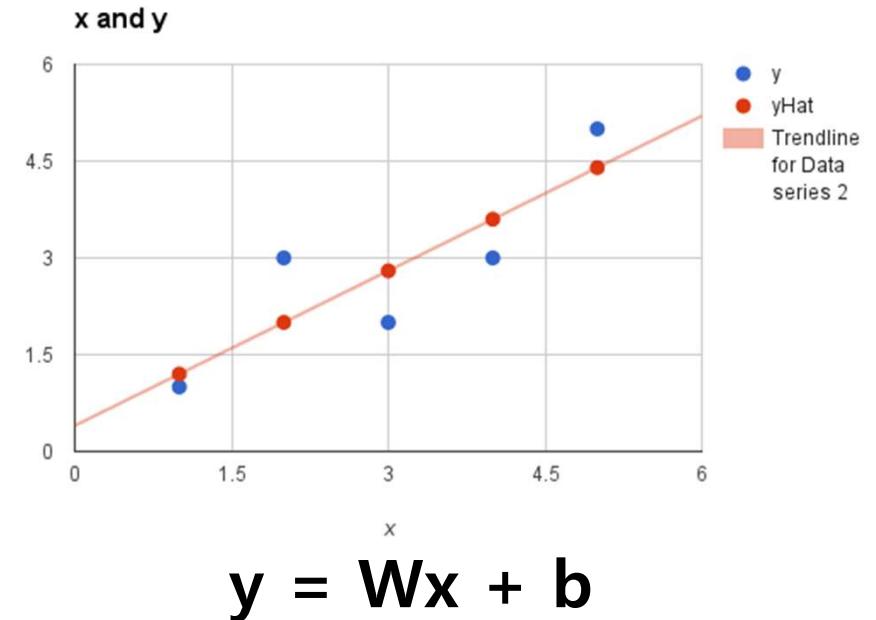
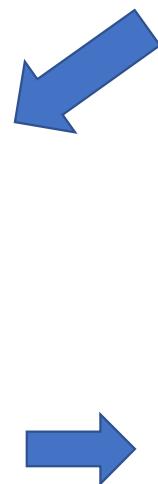
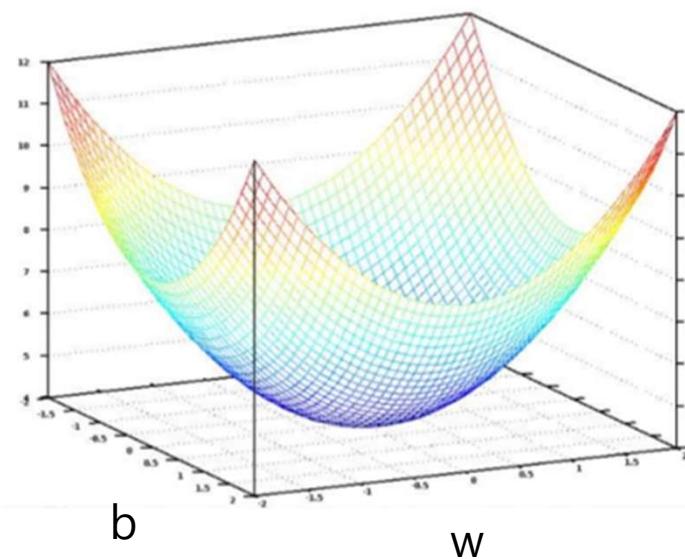
$$\theta^T = [\theta_0, \theta_1, \dots, \theta_n]$$
$$X = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- Cost Function :  $J(\theta) = \frac{1}{2m} \sum_{i=0}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

→ 미분 가능 / convex  
 $\frac{1}{2}$  은 수학적 trick (계산의 간편성)

# Cost Function - Linear Regression

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



MSE(Mean Squared Error) 를  
최소화 하는  $W$  와  $b$  를 optimize

# Loss Function 의 미분과 parameter update

- $y = Wx + b = \theta_0 + \theta_1 x$
- Loss Function  $L(\theta_0, \theta_1) = \frac{1}{N} \sum_{i=0}^m (y_i - (\theta_0 + \theta_1 x_i))^2$
- Gradient  $\frac{\partial L(\theta_0, \theta_1)}{\partial \theta_0} = -2 \frac{1}{N} \sum_{i=0}^m (y_i - (\theta_0 + \theta_1 x_i))$   
 $\frac{\partial L(\theta_0, \theta_1)}{\partial \theta_1} = -2 \frac{1}{N} \sum_{i=0}^m x_i (y_i - (\theta_0 + \theta_1 x_i))$
- Update  $\theta_0 := \theta_0 + \alpha \frac{\partial L(\theta_0, \theta_1)}{\partial \theta_0}$   
 $\theta_1 := \theta_1 + \alpha \frac{\partial L(\theta_0, \theta_1)}{\partial \theta_1}$

# Gradient Descent for Logistic Regression (이진분류)

- Hypothesis (model) :  $\sigma(\theta^T X) = \frac{1}{1+e^{-\theta^T X}}$

$$\begin{aligned}\theta^T &= [\theta_0, \theta_1, \dots, \theta_n] \\ X &= \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} & y &= \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}\end{aligned}$$

- Cost Function :  $J(\theta) = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}))$

$$\text{if } y = 1 : J(\theta) = -\log(\hat{y}^{(i)})$$

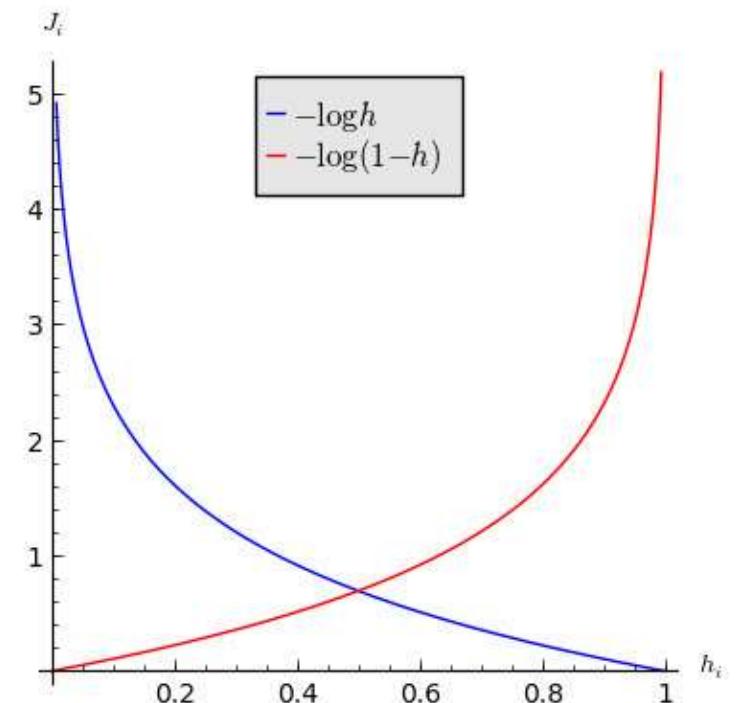
$$y = 0 : J(\theta) = -\log(1 - \hat{y}^{(i)})$$

# Cost Function - Logistic Regression (Binary Cross-entropy)

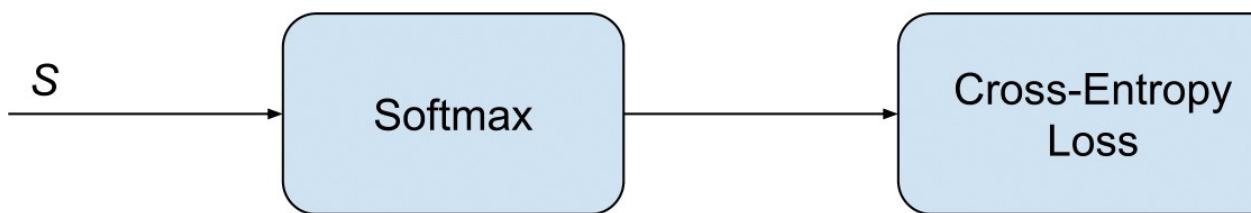
$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) + y^{(i)} \log h_\theta(x^{(i)}) \right]$$

If  $y^{(i)} = 1$  :  $J(\theta) = -\log h_\theta(x^{(i)})$   
where  $h_\theta(x^{(i)})$  should be close to 1

If  $y^{(i)} = 0$  :  $J(\theta) = -\log(1 - h_\theta(x^{(i)}))$   
where  $h_\theta(x^{(i)})$  should be close to 0



# Cost Function - Categorical Crossentropy (Softmax Loss)



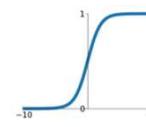
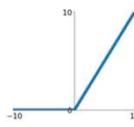
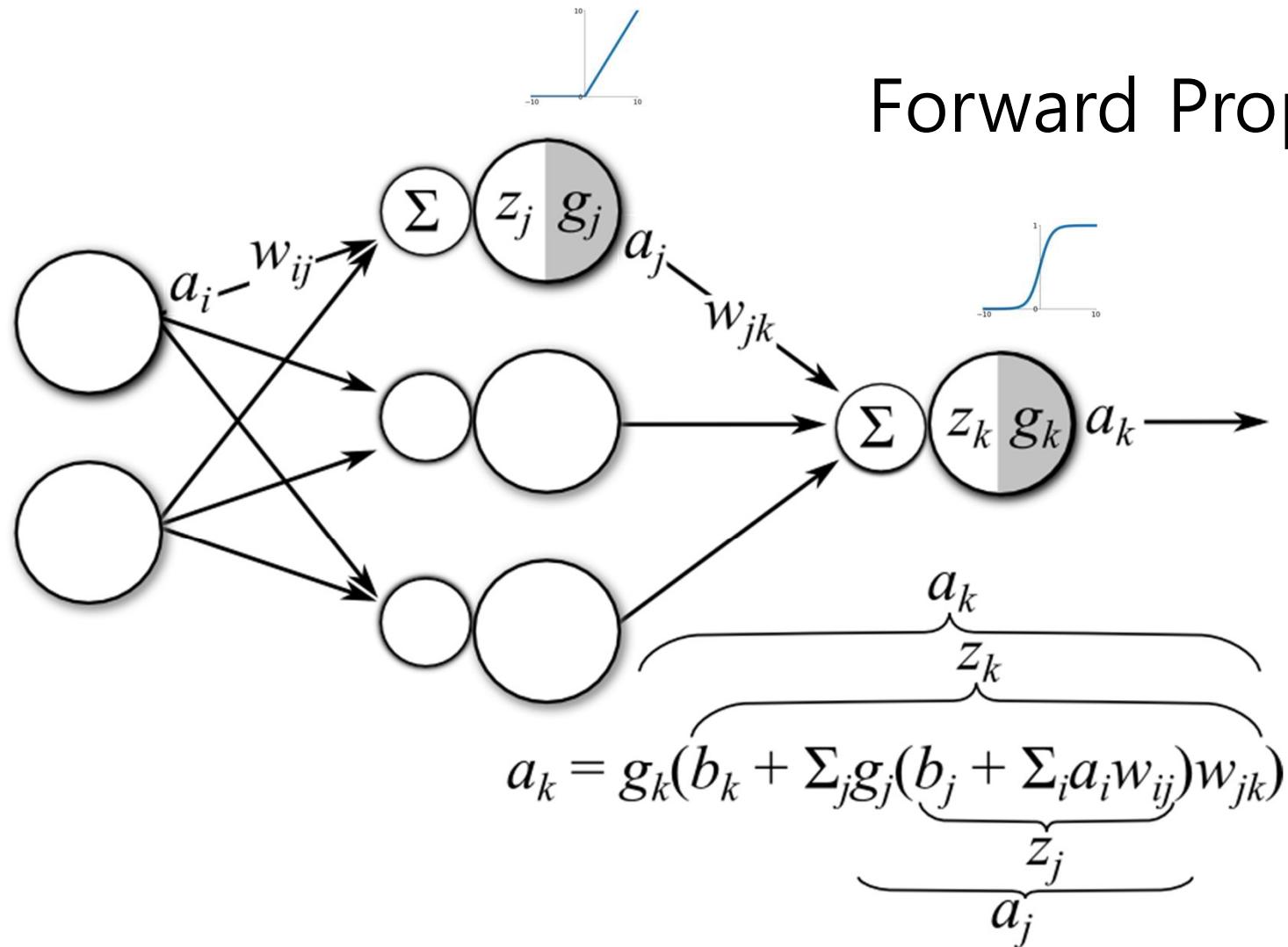
$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = - \sum_i^C t_i \log(f(s)_i)$$

$t_i$  : 0 이 아닌 target  
(one-hot encoded  
되어 있으므로  
multi-class 중 오직  
1 개만 1)

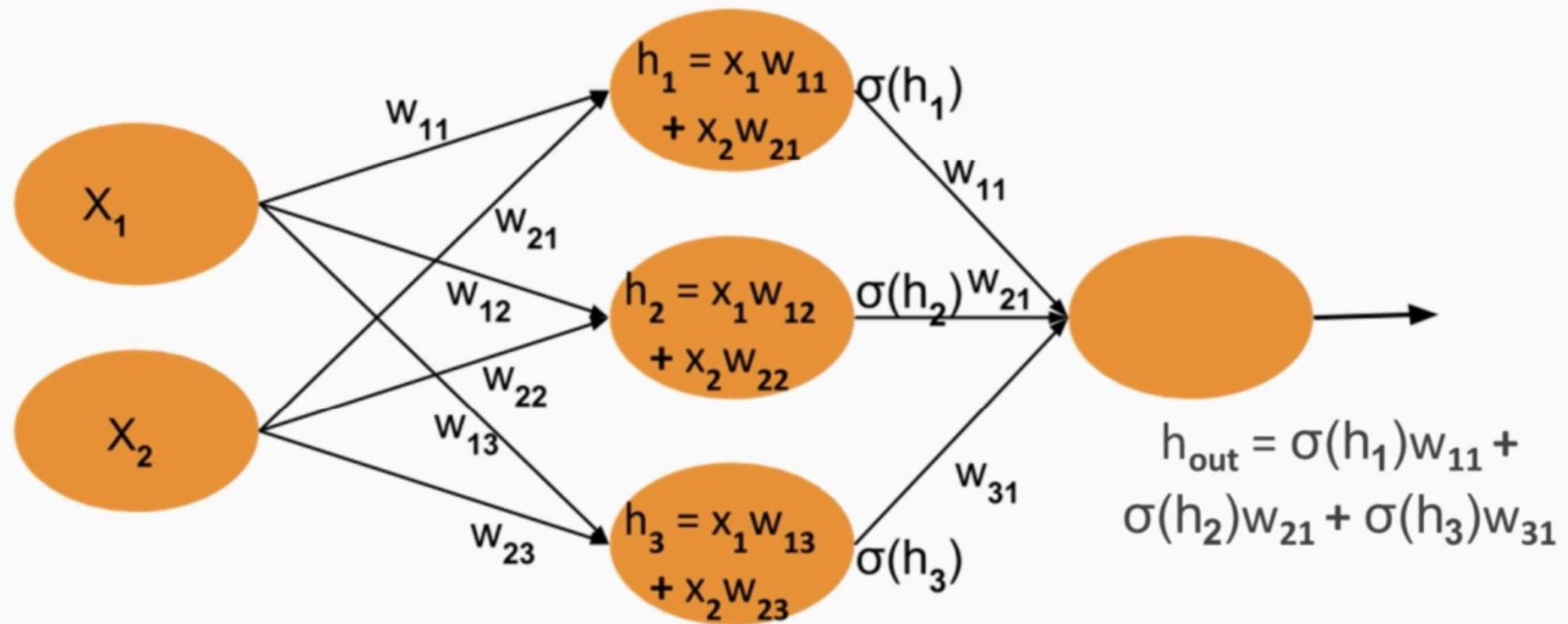
C : multi-classes

Index	0	1	2	3	4	5	6	7	8	9
<b>True Label</b>	0	0	0	0	0	0	0	<b>1</b>	0	0
<b>Prediction</b>	0.1	0.01	0.01	0.01	0.20	0.01	0.01	<b>0.60</b>	0.03	0.02

# Forward Propagation



# Forward Propagation



# Backward Propagation 기초 공식

- 기본 함수의 도함수 (derivative) :  $\frac{dx^2}{dx} = 2x$ ,  $\frac{de^x}{dx} = e^x$ ,  $\frac{d\ln(x)}{dx} = \frac{1}{x}$
- Chain Rule :

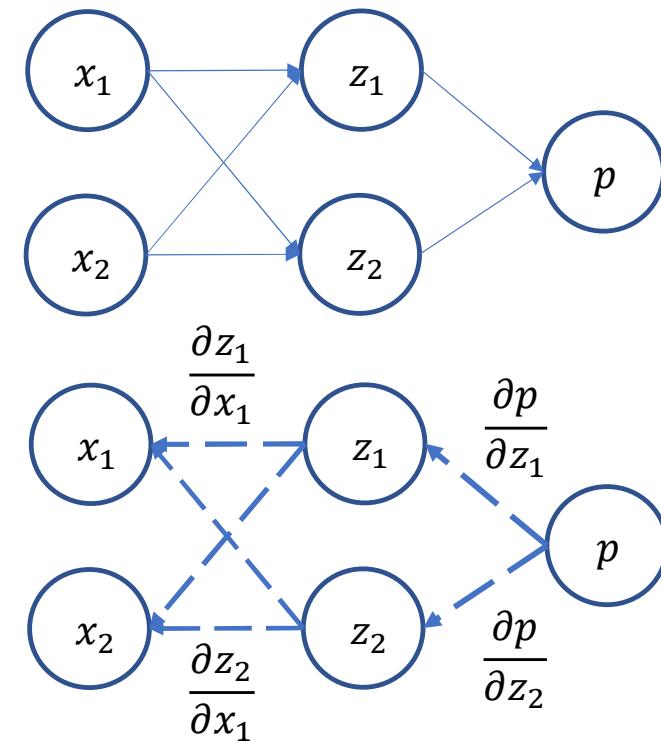
$$p = f(z_1, z_2)$$

$$z_1 = f(x_1, x_2)$$

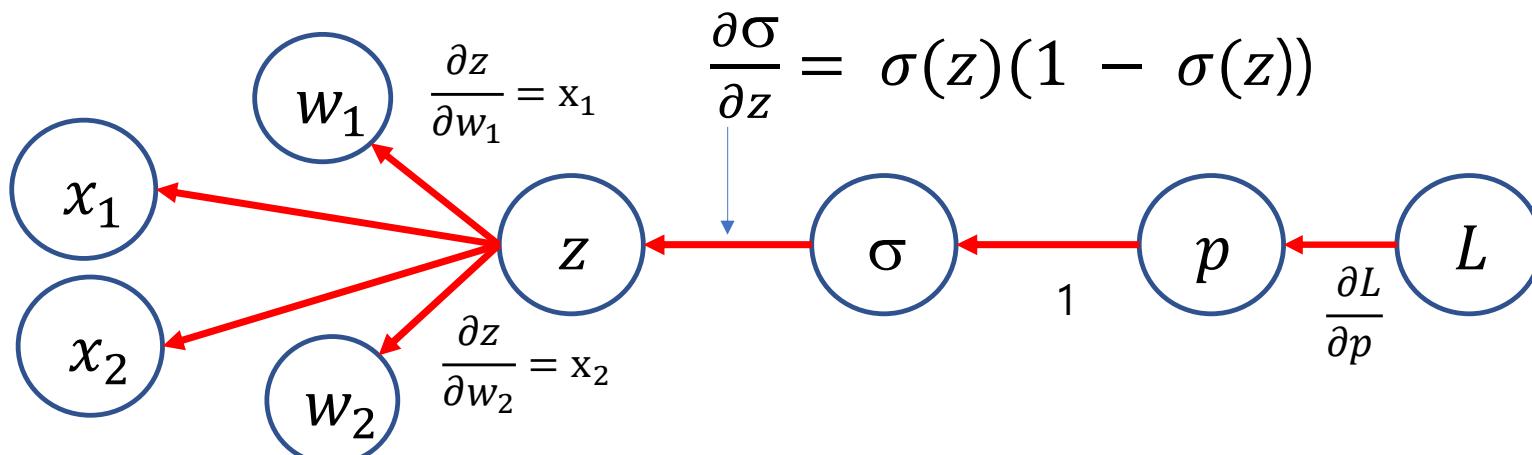
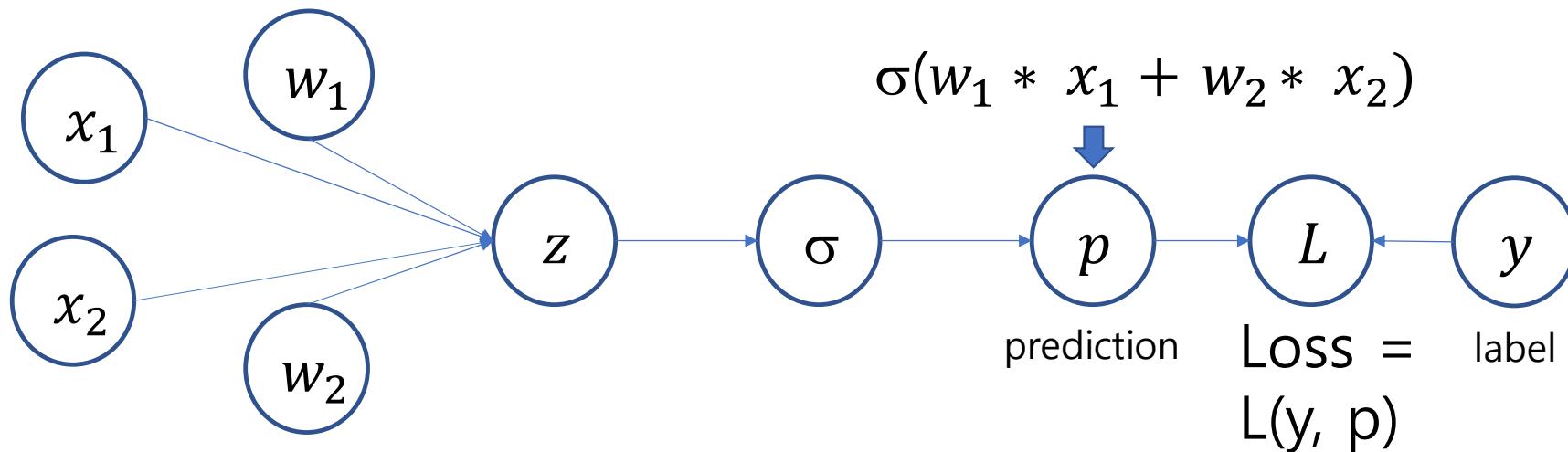
$$z_2 = f(x_1, x_2)$$

$$\frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial z_2} \frac{\partial z_2}{\partial x_1}$$

$$\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial z_2} \frac{\partial z_2}{\partial x_2}$$



Example : 2 개의 feature 를 가진 1 layer + sigmoid activation



경사하강법 적용  
을 위해  $\frac{\partial L}{\partial w_1}$  과  
 $\frac{\partial L}{\partial w_2}$  필요

## Sigmoid 함수의 미분

$$\begin{aligned}\frac{d}{dx} \sigma(x) &= \frac{d}{dx} \left[ \frac{1}{1 + e^{-x}} \right] \\&= \frac{d}{dx} (1 + e^{-x})^{-1} \\&= -(1 + e^{-x})^{-2} (-e^{-x}) \\&= \frac{e^{-x}}{(1 + e^{-x})^2} \\&= \frac{1}{1 + e^{-x}} \cdot \frac{e^{-x}}{1 + e^{-x}} \\&= \frac{1}{1 + e^{-x}} \cdot \frac{(1 + e^{-x}) - 1}{1 + e^{-x}} \\&= \frac{1}{1 + e^{-x}} \cdot \left( \frac{1 + e^{-x}}{1 + e^{-x}} - \frac{1}{1 + e^{-x}} \right) \\&= \frac{1}{1 + e^{-x}} \cdot \left( 1 - \frac{1}{1 + e^{-x}} \right) \\&= \boxed{\sigma(x) \cdot (1 - \sigma(x))}\end{aligned}$$

# Backpropagation (Chain Rule 적용)

$$3: \frac{\partial p}{\partial h_1} \quad \frac{\partial p}{\partial h_2}$$

We will need these for GD

$$2: \frac{\partial p}{\partial z_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1}$$

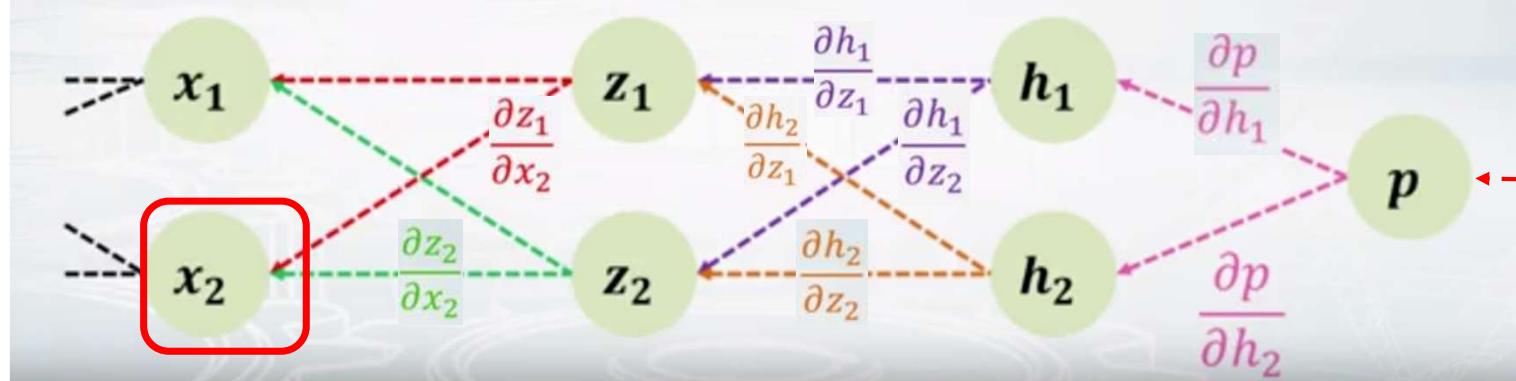
$$\frac{\partial p}{\partial z_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2}$$

$$1: \frac{\partial p}{\partial x_1} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_1} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_1} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_1}$$

$$1: \boxed{\frac{\partial p}{\partial x_2} = \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_1} \frac{\partial z_1}{\partial x_2} + \frac{\partial p}{\partial h_1} \frac{\partial h_1}{\partial z_2} \frac{\partial z_2}{\partial x_2} + \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial z_2} \frac{\partial z_2}{\partial x_2}}$$

\* 같은 색으로 표시된 부분은 한번 계산하면 여러 번 reuse

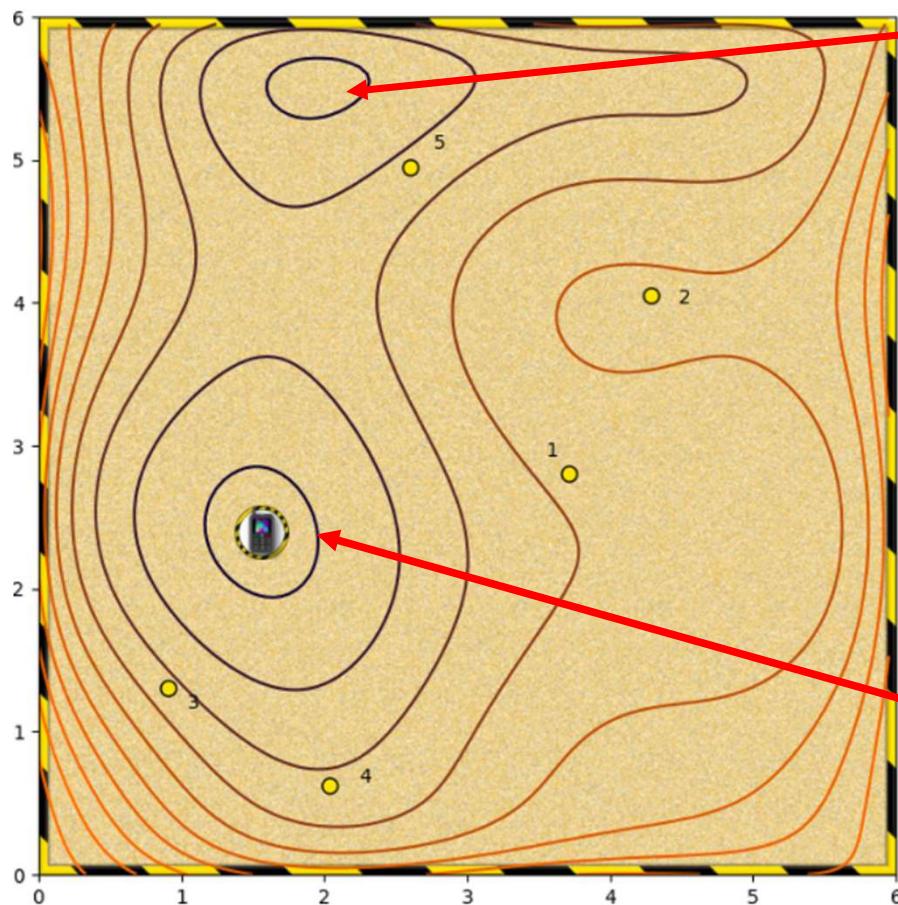
$$\begin{aligned} \frac{\partial L}{\partial w_1} &= \frac{\partial L}{\partial p} \frac{\partial p}{\partial w_1} \\ \frac{\partial L}{\partial p} &= \frac{\partial p}{\partial h_2} \frac{\partial h_2}{\partial w_1} \\ &= \frac{\partial p}{\partial p} \frac{\partial h_2}{\partial h_2} \frac{\partial h_2}{\partial w_1} \end{aligned}$$



# Backpropagation 요약

- 각각의 input data 에 대하여,
- 각 layer 별로 forward pass output 값을 계산
- Output layer 의 cost function 값을 계산
- Backpropagation 을 통해 cost function 의 derivative 를 전단계의 layer 로 전달
- Error term 의 값에 따라 각 layer 의 weight 를 update

# Global Minima / Local Minima



Local Minimum

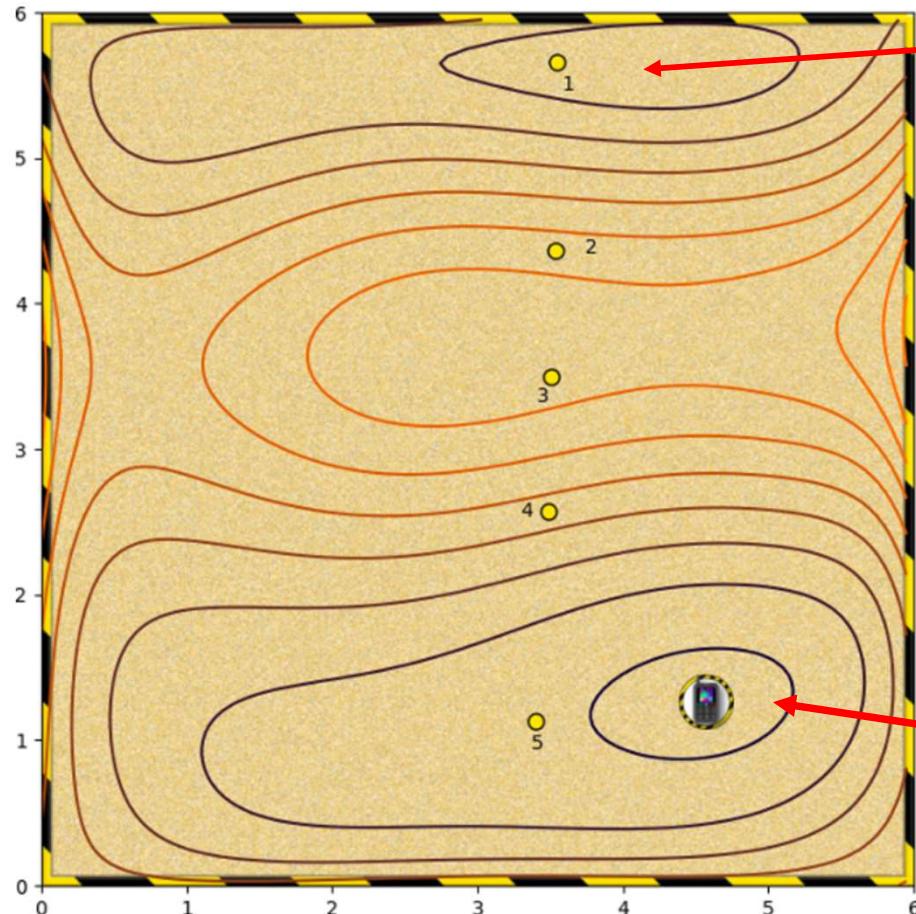
단순히 가장 가파른 경사만을 따라갈 경우,

1, 3, 4 – Global Minima 도달 가능

2, 5 – Local Minima 도달 가능

Global Minimum

# Global Minima / Local Minima



Local Minimum

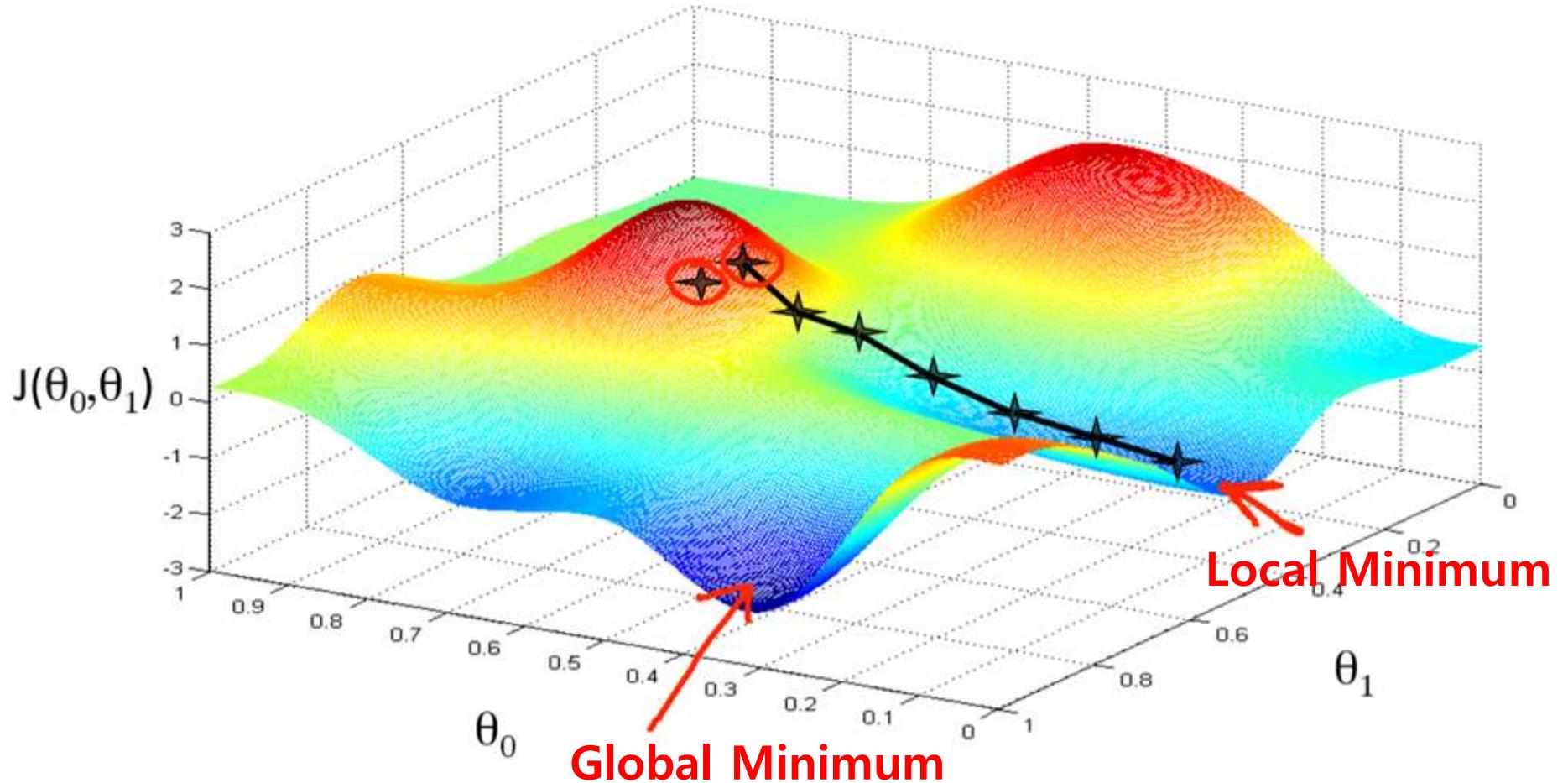
단순히 가장 가파른 경사만을 따라갈 경우,

3, 4, 5 – Global Minima 도달 가능

1, 2 – Local Minima 도달 가능

Global Minimum

# Global Minima / Local Minima



# Learning Rate ( $\alpha$ )

- Step size
- Range :  $1e-6 \sim 1.0$  (default 0.01)
  - High learning rate – fast learning, may overshoot the target
  - Low learning rate – slow learning, may take long time
- Adaptive Learning Rates  
초기값을 크게 주고 학습 진행에 따라 slow down

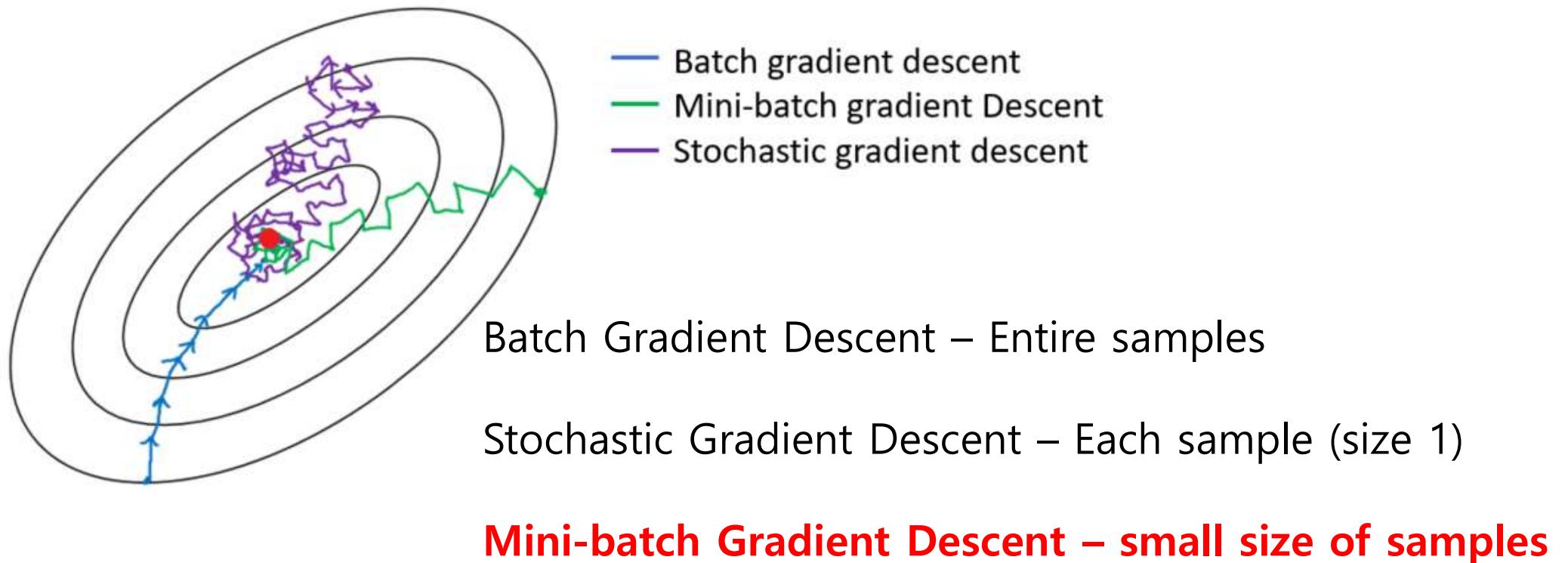
# Optimizers

- Stochastic Gradient Descent Optimizer
- RMSProp Optimzer
- Adagrad Optimizer
- Adam Optimizer, etc

[http://ruder.io/content/images/2016/09/contours\\_evaluation\\_optimizers.gif](http://ruder.io/content/images/2016/09/contours_evaluation_optimizers.gif)

[http://ruder.io/content/images/2016/09/saddle\\_point\\_evaluation\\_optimizers.gif](http://ruder.io/content/images/2016/09/saddle_point_evaluation_optimizers.gif)

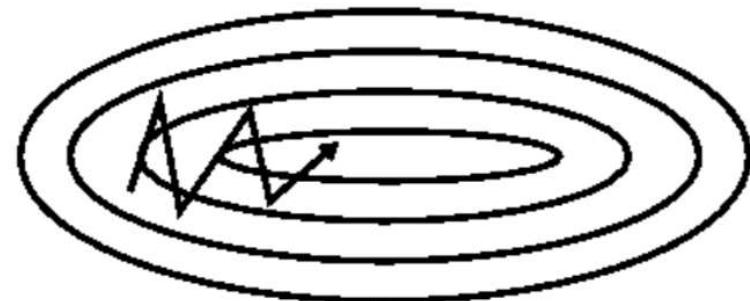
# Stochastic Gradient Descent (확률적 경사하강법)



# Momentum : 방향성을 유지하며 가속



(a) SGD without momentum

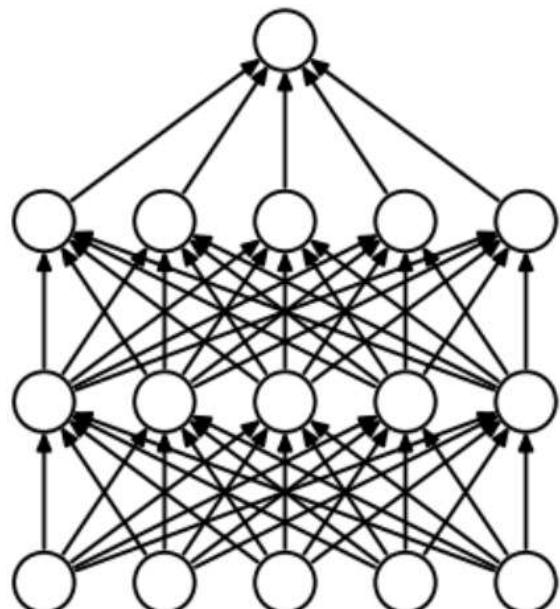


(b) SGD with momentum

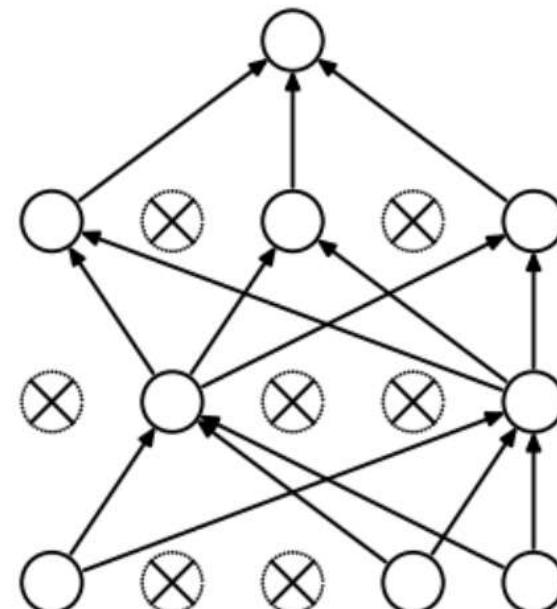
Global minimum 에 빨리 도달하기 위해 vertically 는 변화가 적고  
horizontally 는 변화가 크도록 parameter 조절

# Dropout regularization

Random 한 drop out 을 통한 과적합 방지 (특정 feature 의존 방지)



(a) Standard Neural Net



(b) After applying dropout.

# epoch

- 정의 – 전체 dataset 이 neural network 을 통해 한번 처리된 것
- Epoch 은 model 의 training 시에 hyperparameter 로 횟수 지정
- 하나의 epoch 은 한번에 처리하기 어려운 size 이므로 여러 개의 batch 로 나누어 처리
- Parameter training 을 위해서는 여러 번 epoch 을 반복해야 한다.
- One epoch 내에서의 iteration 횟수는 total sample size / batch size
- Ex) 1 epoch = 4 iterations = 2000 training example / 500 batches

# Hyper-parameters

- $\alpha$  - Learning Rate
- $\beta$  - momentum term
- # of layers
- Dropout rate
- # of epochs
- Batch size

# Network Layer 와 Neuron 의 개수는 어떻게 결정하는가 ?

- 정해진 rule 이 없음 : Empirical Try and See

→ Too few : 과소적합, Too many : 과대적합

- Input 및 output node 고려
- Training data 의 volume 고려
- Function 의 복잡도 고려
- Training algorithm 고려

# Hyper-parameter 값은 어떻게 정하는가 ?

- 정해진 RULE 이 없음
- 유사한 model 참조
- 경험에 의한 guessing
- Grid search – computationally expensive

# Open Source Libraries for Deep Learning

- **Scikit-Learn** – 2007, Python Library based on Matplotlib, NumPy, SciPy
- Theano - 2007, Open Source Python Library
- **Tensorflow** – 2015, Google. Open Source Machine Learning Framework
- **Keras** – 2015, Open Source Python Library
  - (working on top of Tensorflow, Theano, CNTK)
- Microsoft Cognitive Tool – 2016, CNTK
- Caffe – 2017, Berkeley AI Research
- **Pytorch** – 2016, Facebook
- H2O – 2011, Open Source Big Data platform on Apache Hadoop

# Tensorflow 2.0

# What is Tensorflow ?

- <https://www.tensorflow.org/overview/?hl=ko>

The screenshot shows the TensorFlow Core website's homepage in Korean. At the top, there is a navigation bar with links for 'Install', 'Learn' (which is currently selected), 'API', 'Resources', 'Community', 'Why TensorFlow', and a search bar. A language dropdown is also present. Below the navigation bar, the page title is 'TensorFlow Core'. There are tabs for '개요' (Overview), 'Tutorials', 'Guide', and 'TF 1'. A banner at the top of the main content area says 'Last chance to register for TensorFlow World, Oct 28-31. Use code TF20 for 20% off select passes.' with a 'Register now' button. The main content features a large orange graphic on the left and text in Korean. The text reads: 'TensorFlow는 머신러닝을 위한 엔드 투 엔드 오픈소스 플랫폼입니다.' and 'TensorFlow를 사용하면 초보자와 전문가 모두 머신러닝 모델을 쉽게 만들 수 있습니다. 시작하려면 아래의 섹션을 참조하세요.' Below this, there are two orange buttons labeled '가이드 보기' (View Guide). To the right, there is a diagram illustrating a machine learning model architecture with multiple layers and data flow.

TensorFlow는 머신러닝을 위한 엔드 투 엔드 오픈소스 플랫폼입니다.

TensorFlow를 사용하면 초보자와 전문가 모두 머신러닝 모델을 쉽게 만들 수 있습니다. 시작하려면 아래의 섹션을 참조하세요.

가이드 보기

가이드 보기

가이드에서는 완벽한 엔드 투 엔드 예제와 함께 TensorFlow를 사용하는 방법을 보여줍니다.

가이드는 TensorFlow의 개념과 구성요소에 대해 설명합니다.

The diagram illustrates a neural network architecture. It shows an input layer represented by a laptop screen, which feeds into a hidden layer represented by a stack of three rectangular nodes. This is followed by an output layer represented by a single rectangular node. Various lines with arrows indicate the flow of data between these layers, with one specific connection highlighted by a dashed orange circle and a red dot, representing the flow of information through the network.

# Tensorflow Installation

- pip install --upgrade tensorflow
- import tensorflow as tf
- tf.\_\_version\_\_

## 일반용 – Sequential API

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

## 전문가용 – Subclassing API

```
class MyModel(tf.keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.conv1 = Conv2D(32, 3, activation='relu')
        self.flatten = Flatten()
        self.d1 = Dense(128, activation='relu')
        self.d2 = Dense(10, activation='softmax')

    def call(self, x):
        x = self.conv1(x)
        x = self.flatten(x)
        x = self.d1(x)
        return self.d2(x)

model = MyModel()

with tf.GradientTape() as tape:
    logits = model(images)
    loss_value = loss(logits, labels)
grads = tape.gradient(loss_value, model.trainable_variables)
optimizer.apply_gradients(zip(grads, model.trainable_variables))
```

# Sequential API

```
import tensorflow as tf  
mnist = tf.keras.datasets.mnist
```

```
(x_train, y_train), (x_test, y_test) = mnist.load_data() → Data Loading  
x_train, x_test = x_train / 255.0, x_test / 255.0 → Data Normalization
```

```
model = tf.keras.models.Sequential([  
    tf.keras.layers.Flatten(input_shape=(28, 28)),  
    tf.keras.layers.Dense(128, activation='relu'),  
    tf.keras.layers.Dropout(0.2),  
    tf.keras.layers.Dense(10, activation='softmax')  
])
```

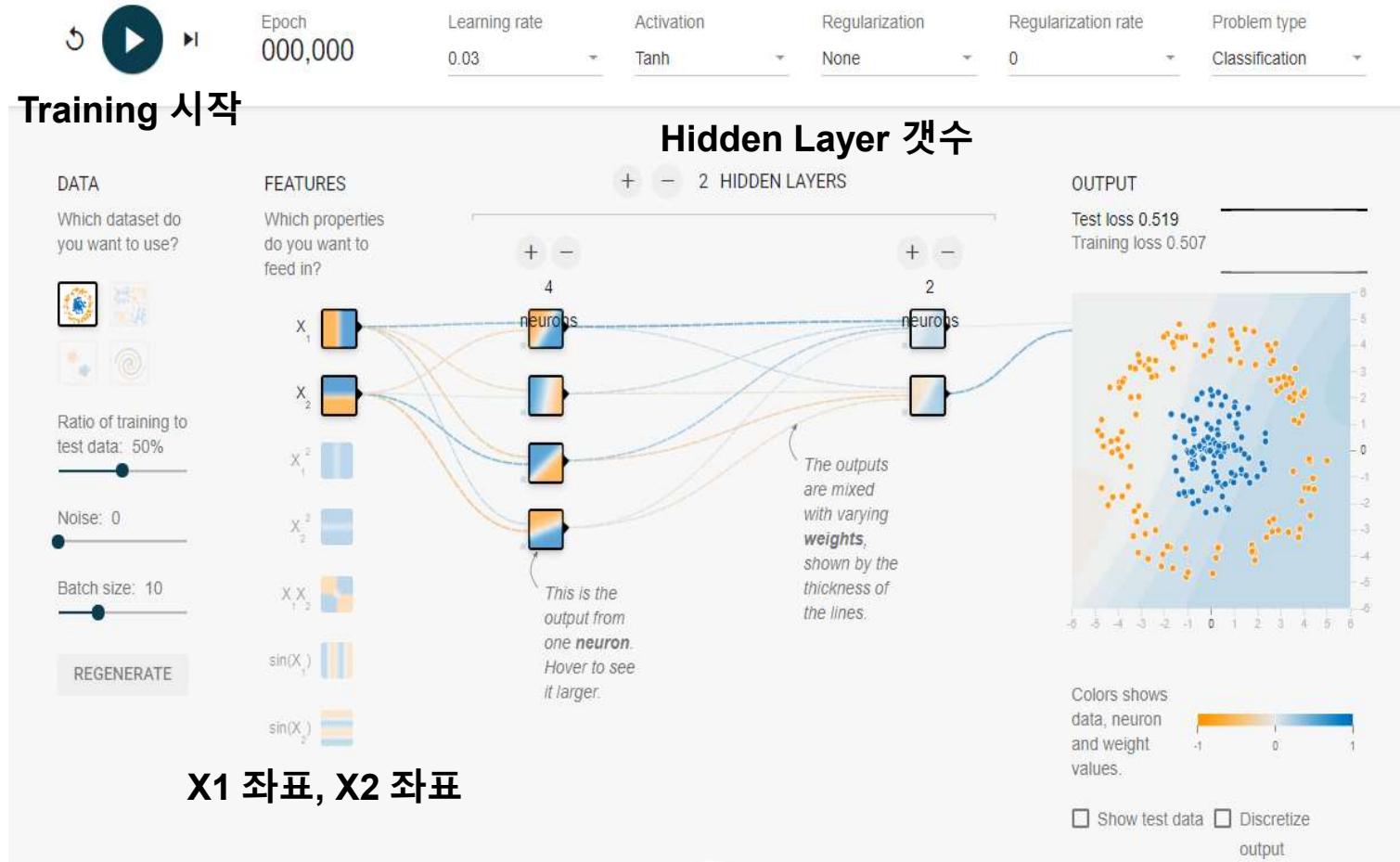
```
model.compile(optimizer='adam',  
              loss='sparse_categorical_crossentropy', → Model Compile  
              metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=5) → Model Train  
model.evaluate(x_test, y_test) → Model 평가
```

# Deep Learning

## Models

# Tensorflow Playground



# Basic Neural Network

# 실습 : Boston 주택가격 Regreesion

1. Boston House Price Dataset
  - `sklearn.datasets.load_boston` 이용
2. 보스턴 시의 주택 가격에 대한 데이터
  - 주택의 여러가진 요건들과 주택의 가격 정보가 포함.
  - 주택의 가격에 영향을 미치는 요소를 이용하여 회귀분석
3. 13 개의 종속변수와 1 개의 독립변수 (주택가격 중앙값) 으로 구성

## - Feature 설명

CRIM 자치시(town) 별 1인당 범죄율,

ZN 25,000 평방피트를 초과하는 거주지역의 비율

INDUS 비소매상업지역이 점유하고 있는 토지의 비율

CHAS 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)

NOX 10ppm 당 농축 일산화질소

RM 주택 1가구당 평균 방의 개수

AGE 1940년 이전에 건축된 소유주택의 비율

DIS 5개의 보스턴 직업센터까지의 접근성 지수

RAD 방사형 도로까지의 접근성 지수

TAX 10,000 달러 당 재산세율

PTRATIO 자치시(town)별 학생/교사 비율

B  $1000(Bk - 0.63)^2$ , 여기서 Bk는 자치시별 흑인의 비율을 말함

LSTAT 모집단의 하위계층의 비율(%)

MEDV 본인 소유의 주택가격(중앙값) (단위: \$1,000)

# Deep Neural Network

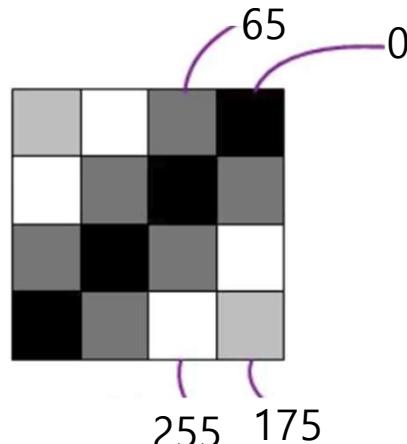
# Mnist Dataset 소개

- Mixed National Institute of Standards and Technology
- 0 ~ 9 의 10 개 숫자 손글씨 image dataset
- 28 x 28 pixel 의 gray scale image
- 각 image 마다 0 to 9 의 label 로 쌍을 이루고 있음
- Train set 60,000 / Test set 10,000
- Machine Learning 의 Hello World 에 해당

## Pixel 의 구성

0 – black

255 – white



label = 5



label = 0



label = 4



label = 1



label = 9



label = 2



label = 1



label = 3



label = 1



label = 4



label = 3



label = 5



label = 3



label = 6



label = 1



label = 7



label = 2



label = 8



label = 6

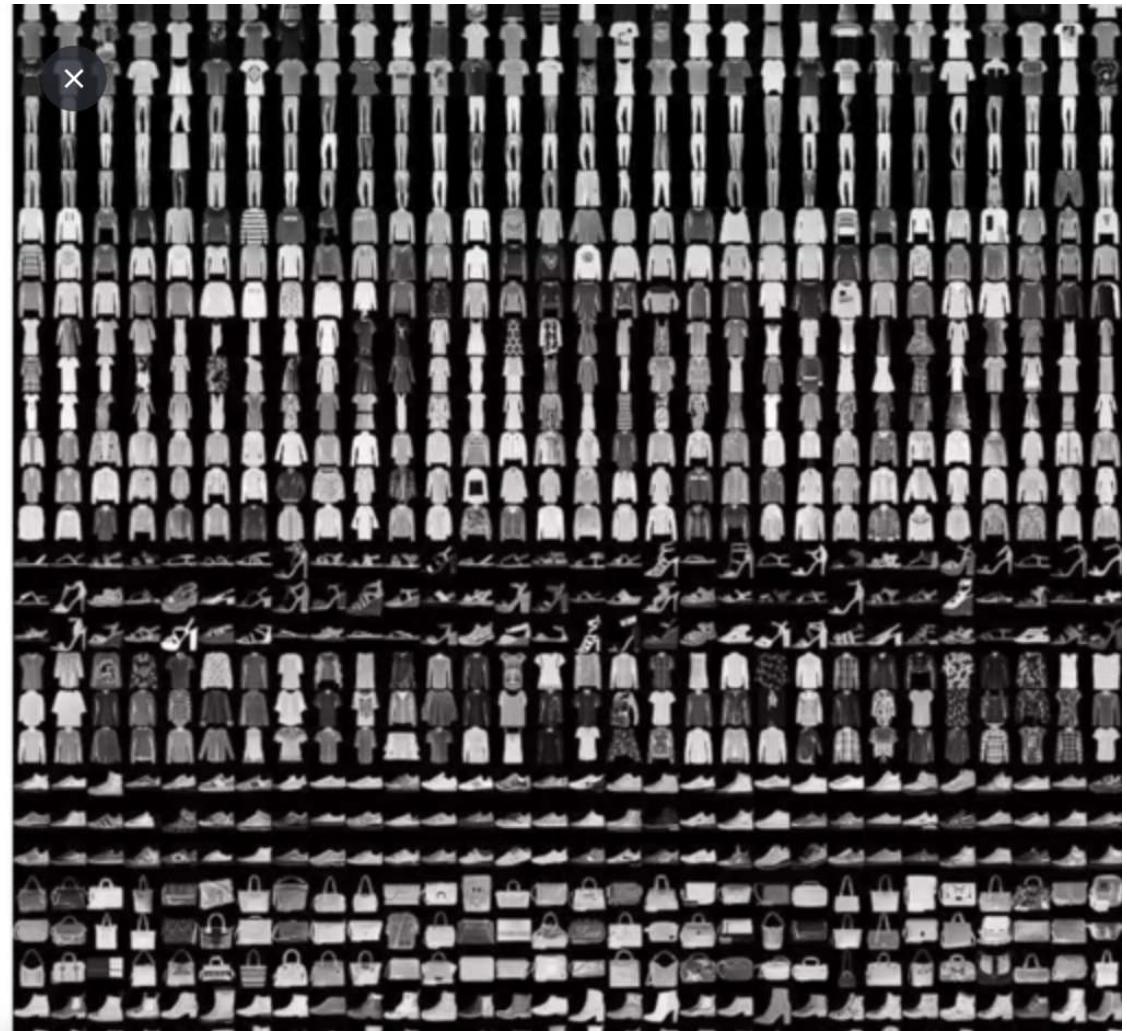
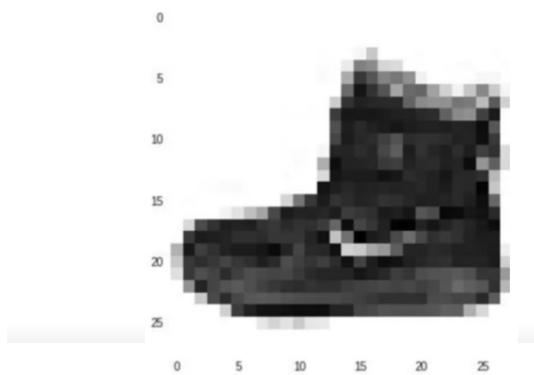


label = 9

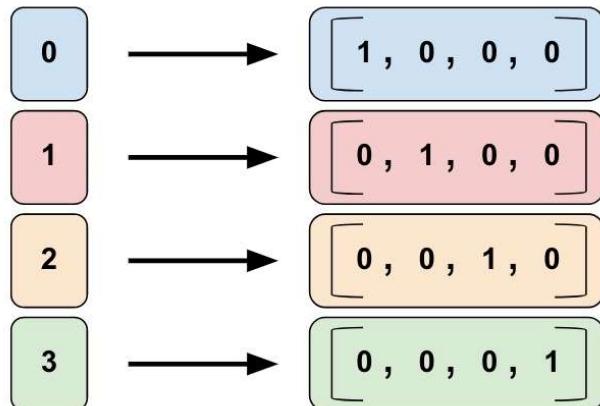


## Fashion MNIST

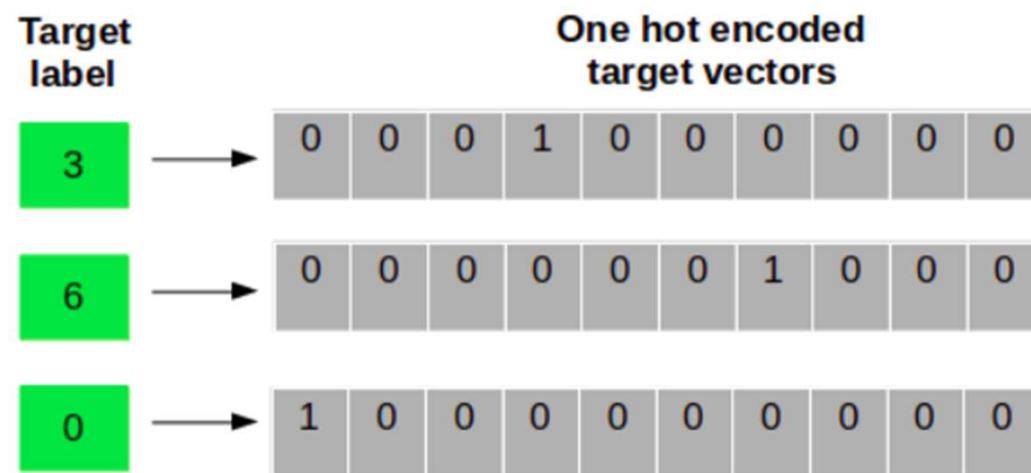
- 70k Images
- 10 Categories
- Images are 28x28
- Can train a neural net!



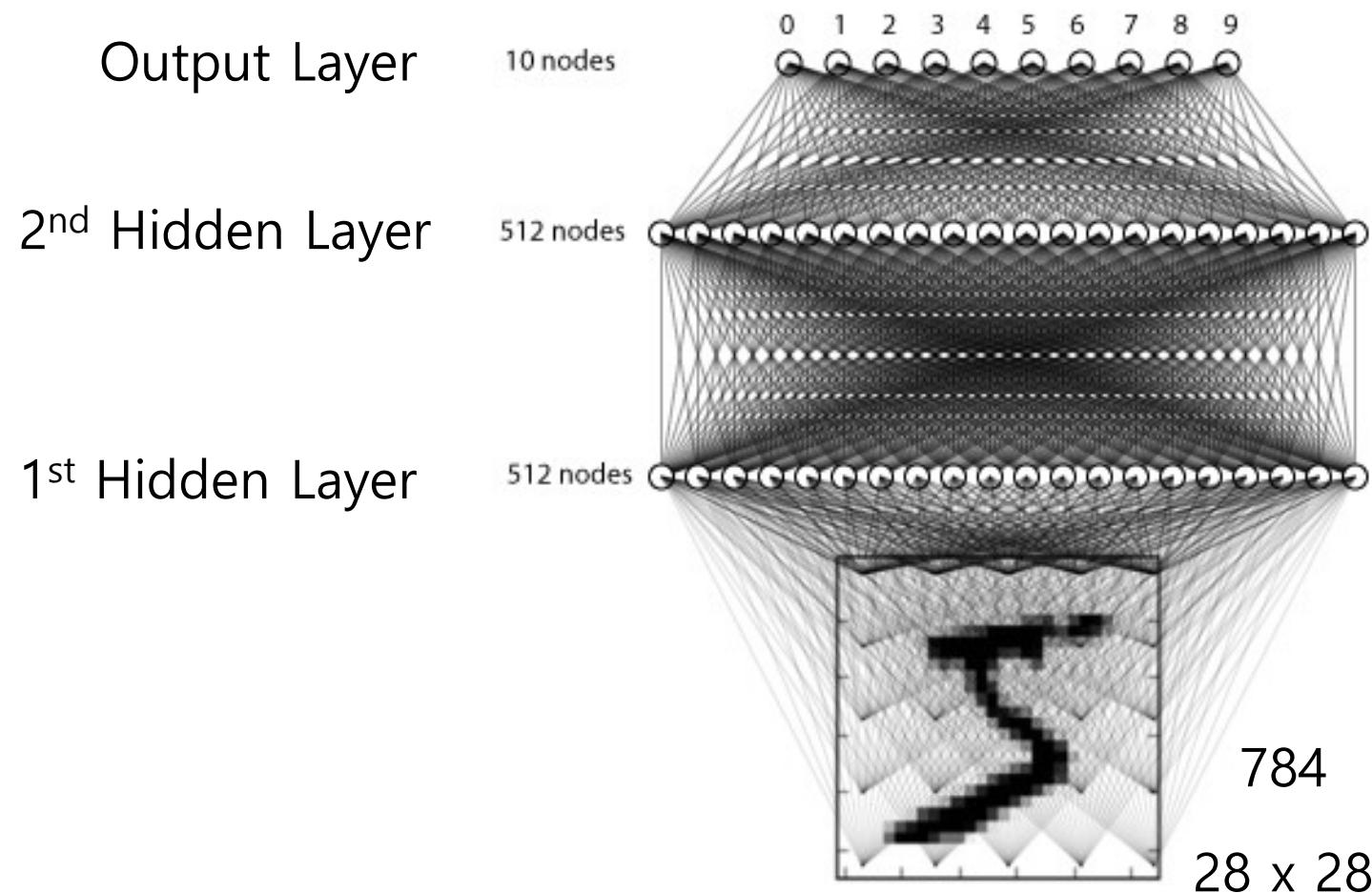
# One-Hot encoding



color	color_red	color_blue	color_green
red	1	0	0
green	0	0	1
blue	0	1	0
red	1	0	0



# Output Layer - softmax



# 실습 : Mnist set 을 이용한 손글씨 인식

1. 2-Layer 이상의 Fully Connected(Dense) Neural Network
2. Input reshaping and scaling
3. One-hot encoding
4. Neural Network Model 구성 및 Compile
5. Model Train
6. Performance Evaluation

# 실습 : Hyper-parameter Tuning 을 이용한 손글씨 인식 성능 개선

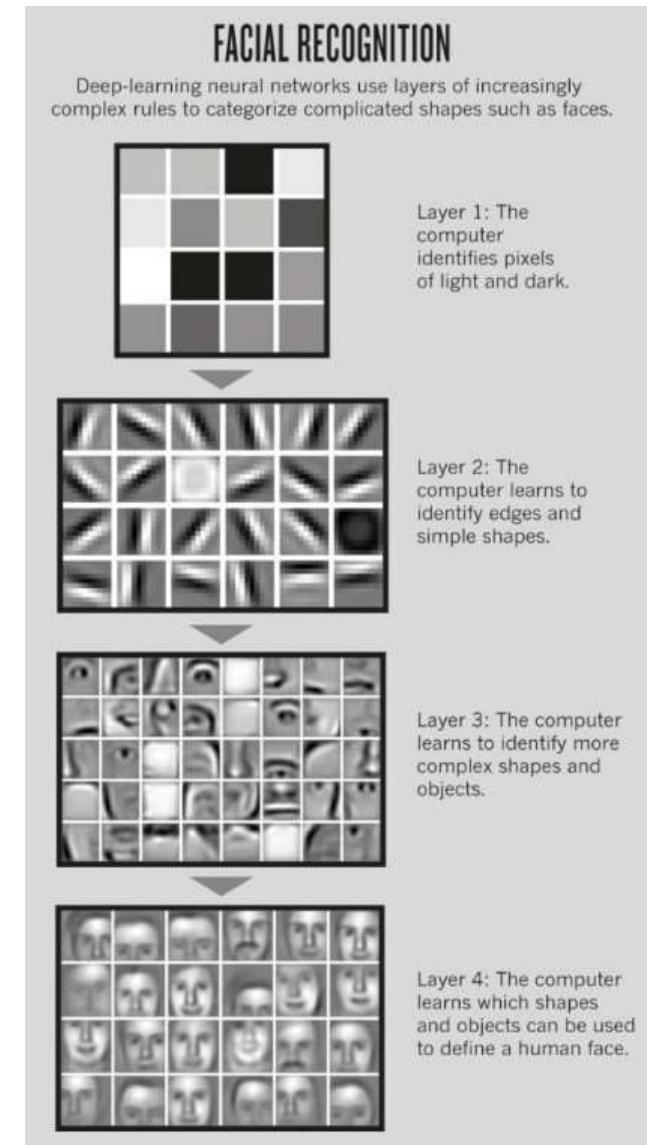
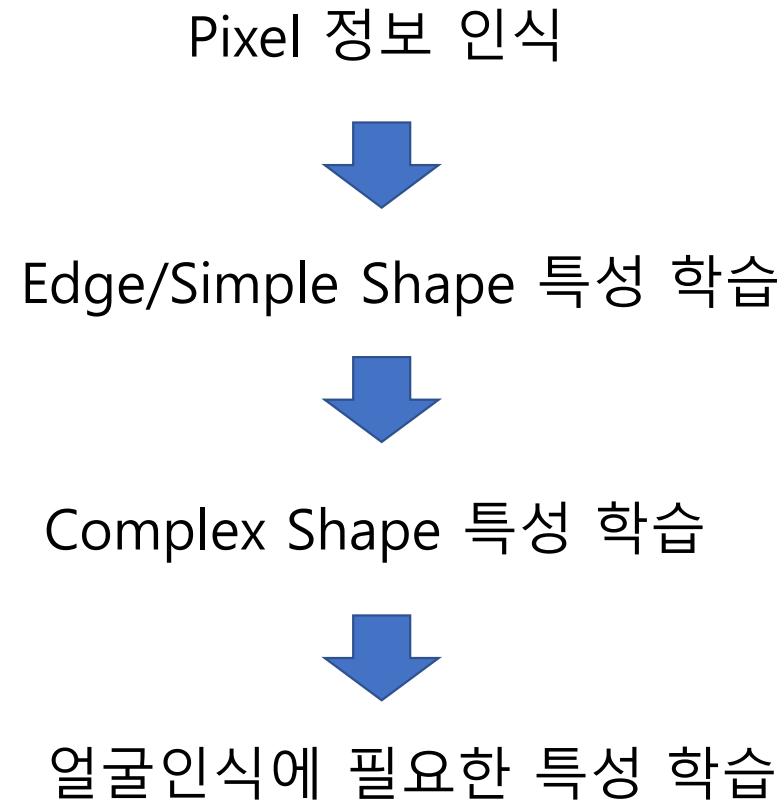
	Model 1	Model 2	Model 3	Model 4	Model 5
# of Hidden Layers	0	2	2	2	3
# of Hidden neurons	128	128	128	512	?+?+?
# of epochs	10	10	10	10	10/15/20
Dropout	0	0	0.2	0.2	0.2/0.3
Batch size	128	128	512	512	256/512
accuracy	92.6	97.4	98.01	98.06	98.40

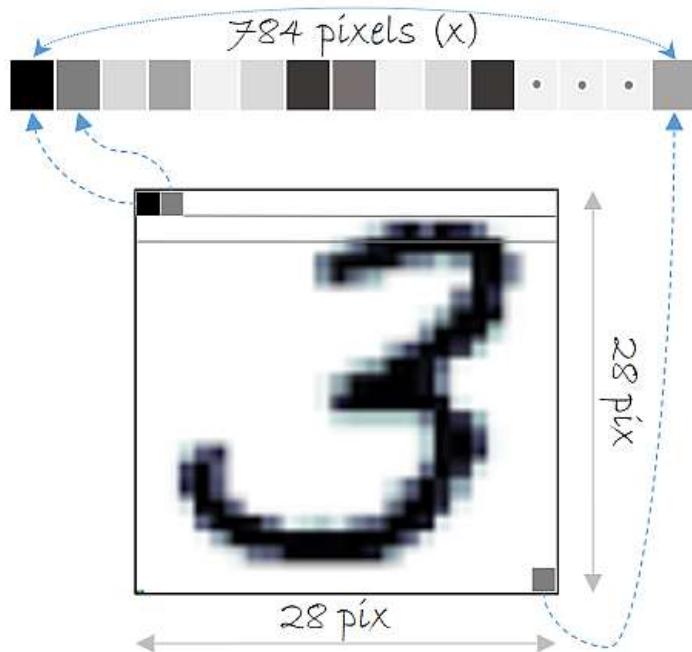
→ Hyperparameter tuning 을 통해 98.4 % 의 정확도 달성

→ Enough ? → No ! → CNN

# CNN (Convolutional Neural Network)

# Multiple Levels of Abstraction





이전 실습 모델에서의 input image 처리

$60000 \times 28 \times 28 \rightarrow \text{reshape}(60000, 784)$

Image Data 의  
공간적, 지역적 특성 상실

If 1 mega pixel  $\rightarrow 1,000 \times 1,000 \times 3 = 3 \text{ million features !!!}$

$\rightarrow 300 \text{ 만 차원} \times \text{Layer 수} \times \text{각 Layer 의 Neuron 수}$

$\rightarrow$  계산량 급증

$\rightarrow$  Image Data 를 처리하기 위한 특별한 구조의 Neural Network 필요

# CNN 의 특별한 Layers

- **Convolutional Layer (합성곱층)**

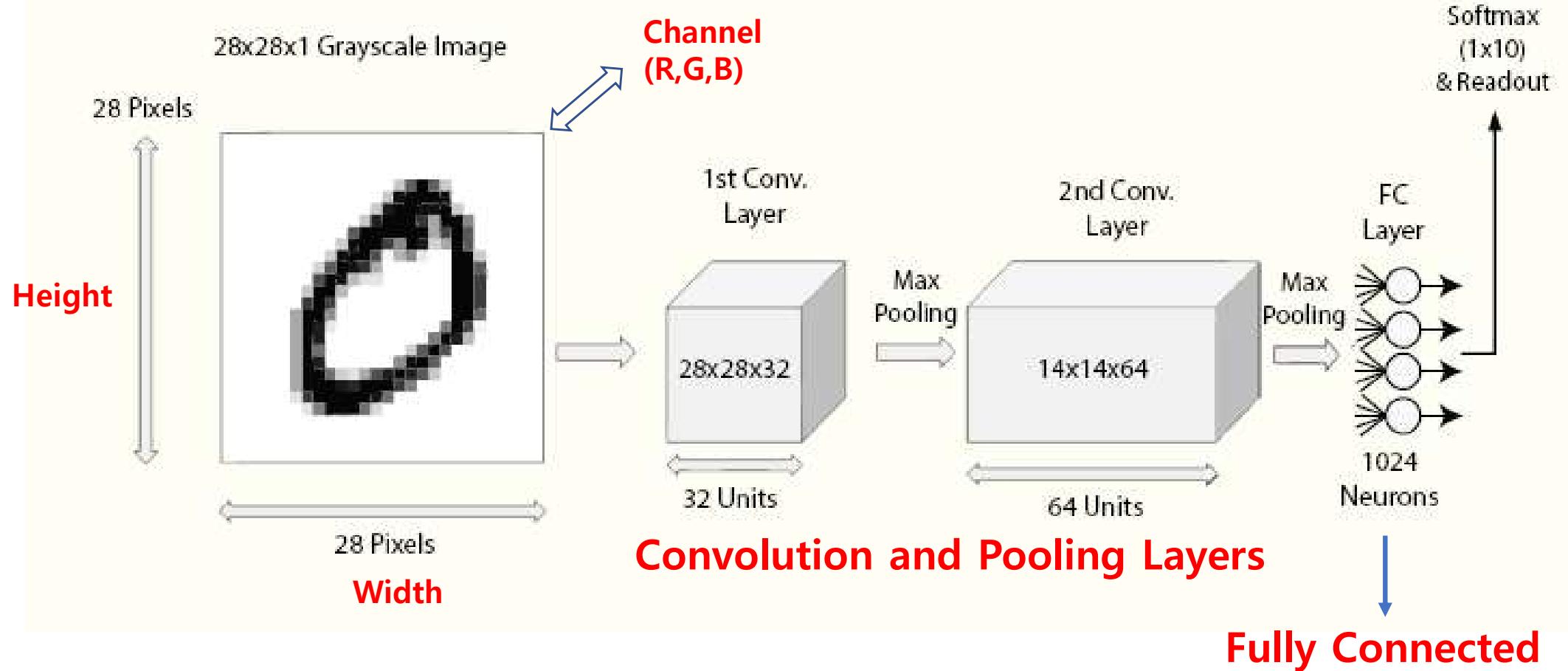
- Image 정보의 공간적 지역 특성 보존
- Filter (Kernel) 을 이용한 이미지 특성 추출

- **Pooling Layer (풀링층)**

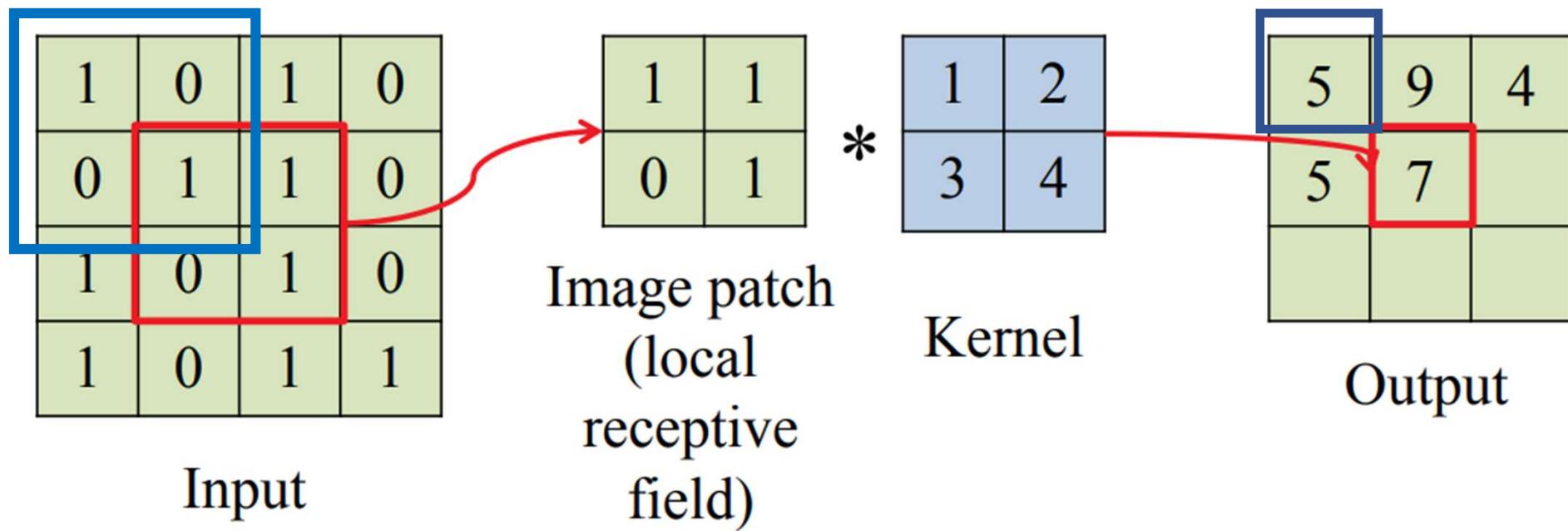
- Image data 의 정보 손실 없는 압축

→ 계산량 및 메모리 사용량 축소, 파라미터의 수 감소 (과적합 방지)

## 2. Dimensions of Layers



# How Convolution works ?

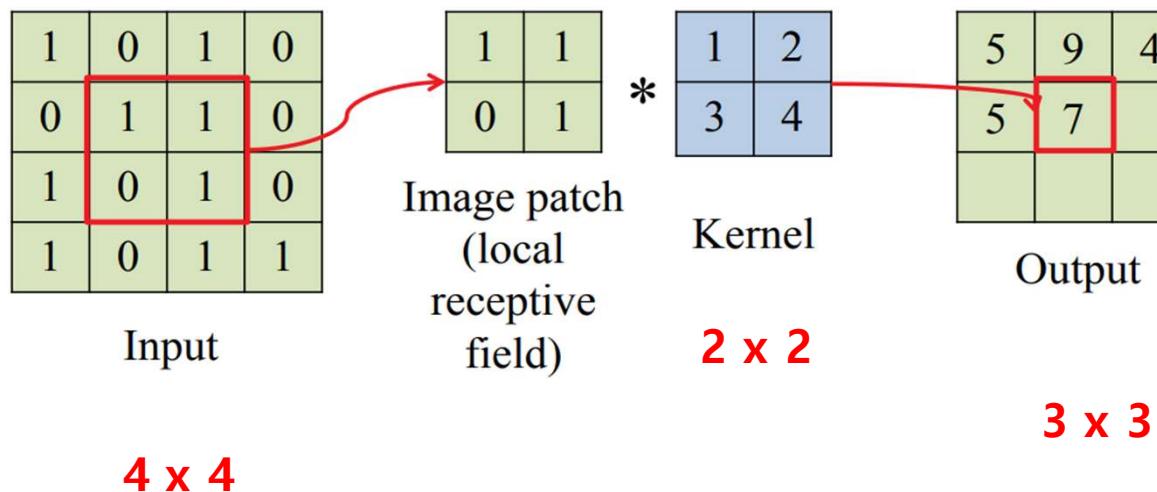


Kernel\_size=(2, 2), stride=(1, 1), No padding

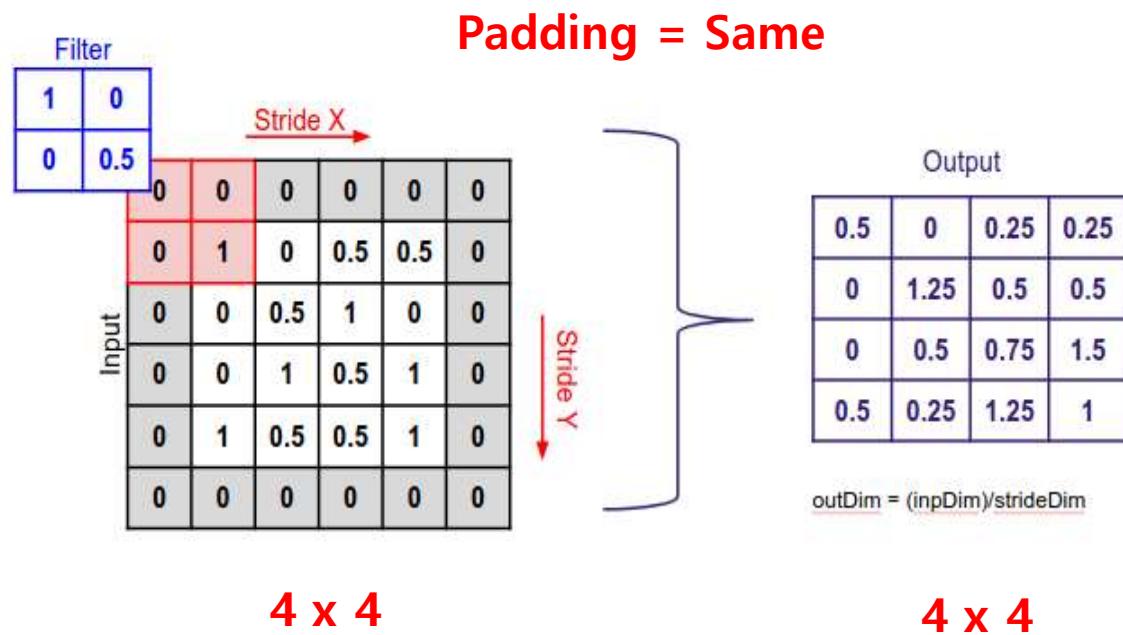
# Padding

- Convolution 때마다 image의 edge 정보가 소실

Padding = **Valid (No Padding)**

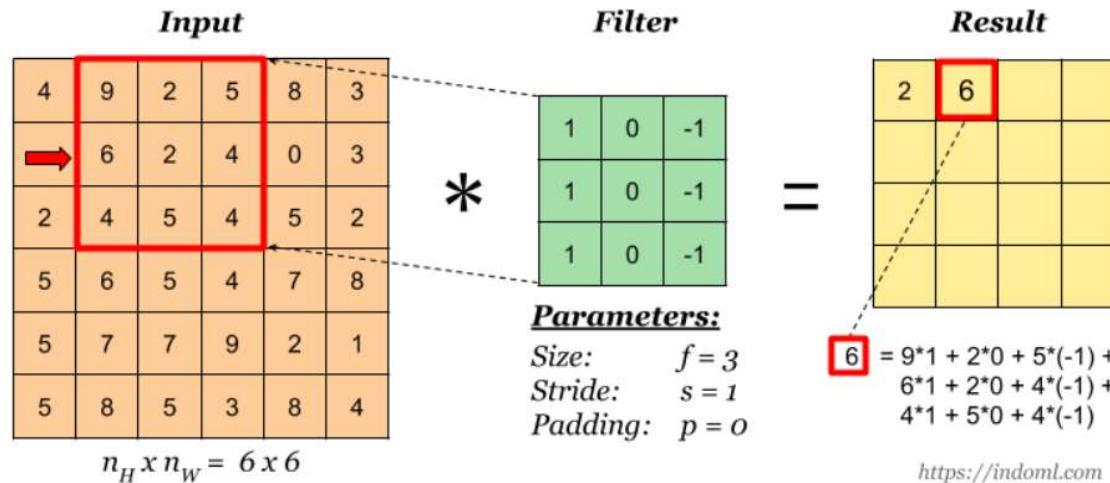


- Input size = output size ( $\rightarrow$  Input 의 주위에 0 pixel padding)
- Padding = "same" convolution



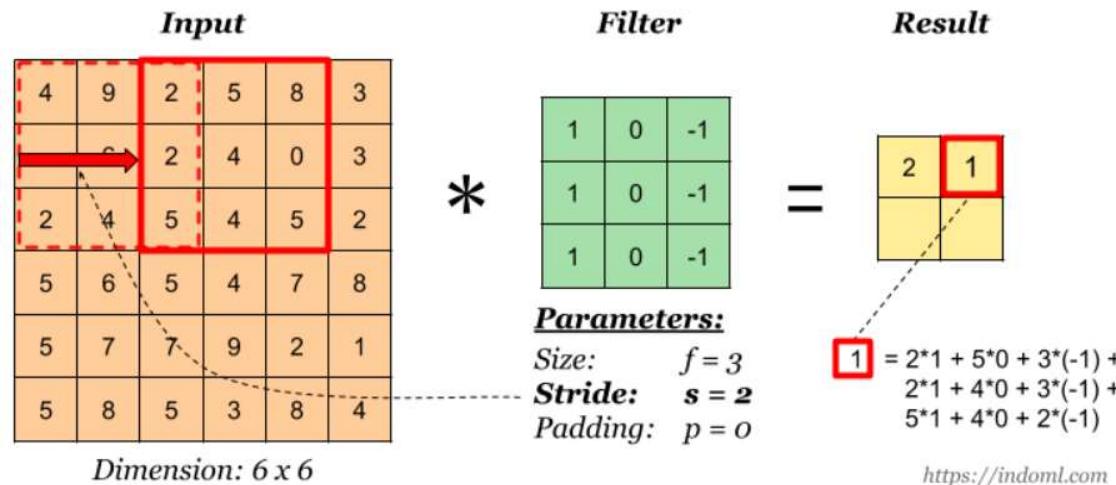
# Striding

Stride = 1



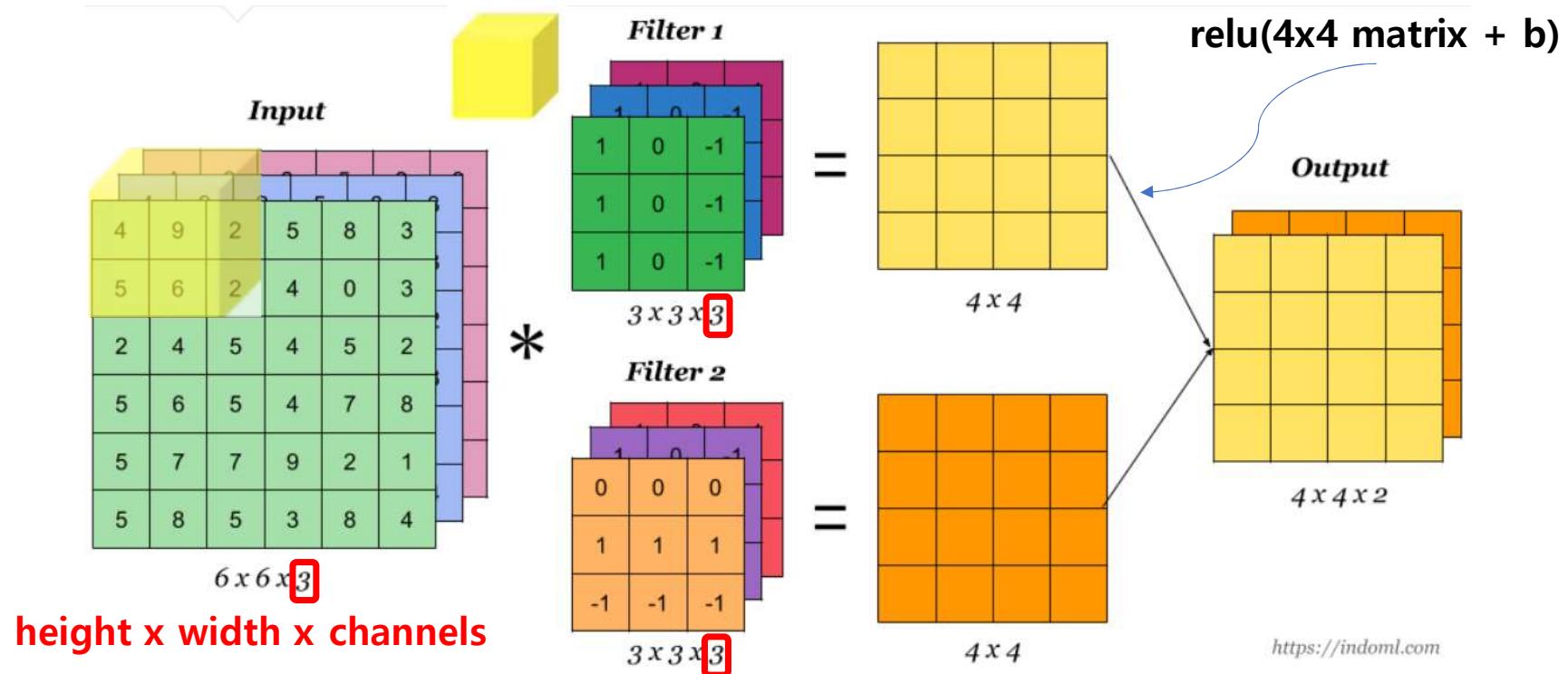
$4 \times 4$

Stride = 2



$2 \times 2$

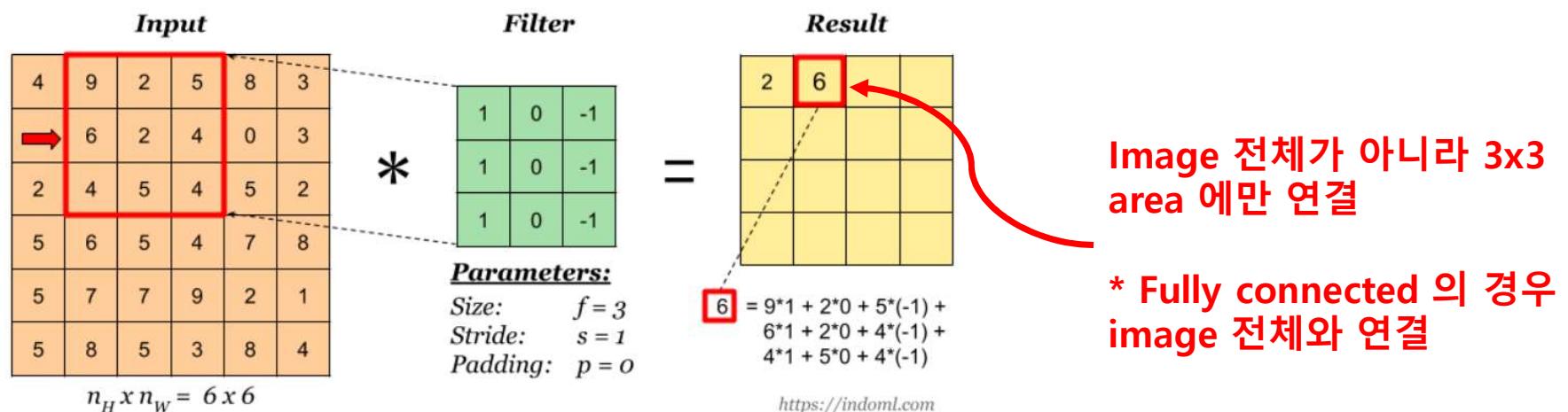
# Convolutions over Volumes (RGB images)



# Convolution Layer 의 2 가지 특성

## 1. Locality

- kernel size 만큼의 작은 구역 (patch) 의 인접한 pixel 들에 대한 correlation 관계를 비선형 필터를 적용하여 추출
- 이러한 필터를 여러 개 적용하면 다양한 local 특징을 추출 가능



## 2. Parameter Sharing

- input 상의 모든 patch 들은 동일한 kernel 을 적용하여 next layer 의 output 을 출력한다.  
ex) vertical edge detector – image 전체에 동일한 kernel 적용
- Fully connected layer 를 image data 에 사용할 경우에 비해 parameter 의 수를 획기적으로 줄임

# Kernel(Filter) 의 특성 추출 예

convolution

Original image

Kernel

$\begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$

$*$

$=$





Edge detection

$\begin{array}{|c|c|c|} \hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline \end{array}$

$*$

$=$



Sharpening

$\begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$

$*$   $\frac{1}{9}$

$=$



Blurring

- CNN 이전에는 edge detection filter 를 computer vision 전문가들이 모두 manually 만들어 줌  
ex) 수직 / 수평 / 45 도 / 명암 구분 filter 등
- Neural Network 은 훨씬 더 다양한 특성의 filter 들을 back-propagation 을 이용하여 자동으로 학습하고 스스로 만들어 냄

# Ex) Edge Detector

밝은 부분	어두운 부분
10 10 10 0 0 0	
10 10 10 0 0 0	
10 10 10 0 0 0	
10 10 10 0 0 0	
10 10 10 0 0 0	
10 10 10 0 0 0	
	

\*

1	0	-1
1	0	-1
1	0	-1
		

=

0	10	10	0
0	10	10	0
0	10	10	0
			

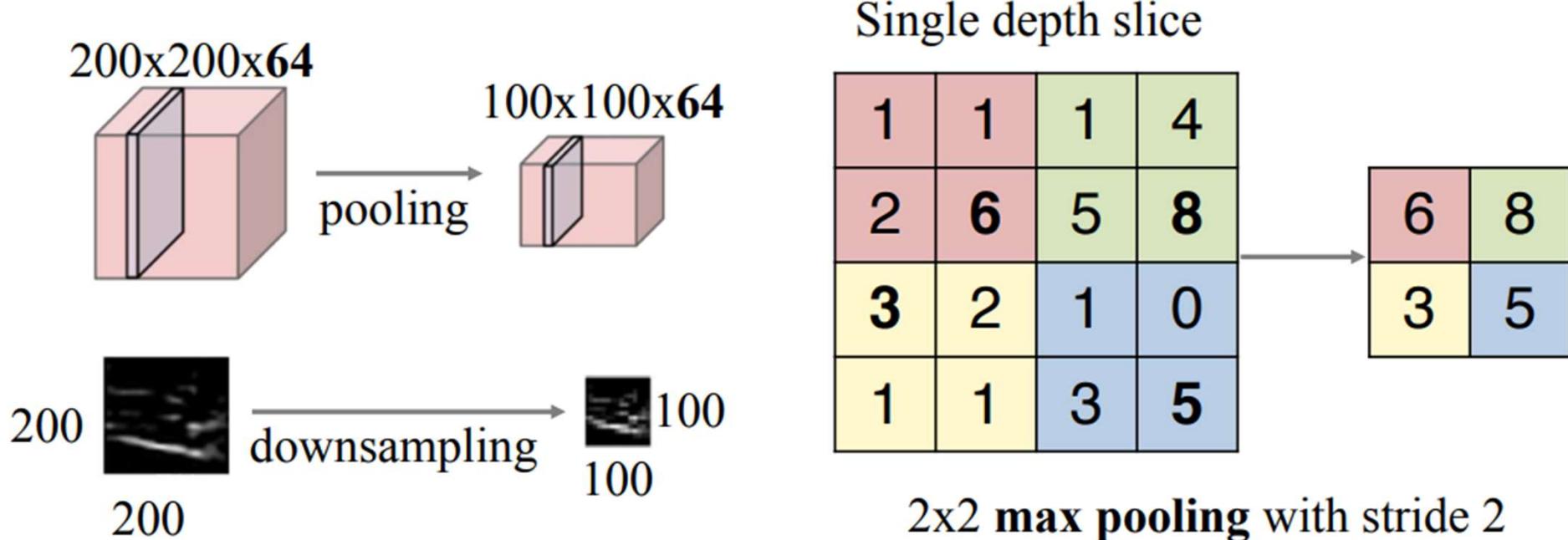
**Edge**

**Backpropagation**에  
의한 자동 학습

$w_1$	$w_2$	$w_3$
$w_4$	$w_5$	$w_6$
$w_7$	$w_8$	$w_9$

# How Pooling works ?

- Pooling 의 뉴런은 가중치가 없음
- 최대, 평균을 이용한 이미지 subsampling (부표본 작성)



# Pooling Layer 의 2 가지 특성

## 1. Positional Invariance

- 특정 pixel 의 정확한 position 에 less sensitive
- 여러 번의 pooling 을 거치면 넓은 영역에 걸쳐 같은 효과 발생  
(See Wider !)

## 2. Size 축소

- 계산량을 크게 줄임
- 과적합 방지

# What is Flattening ?

1	1	0
4	2	1
0	2	1

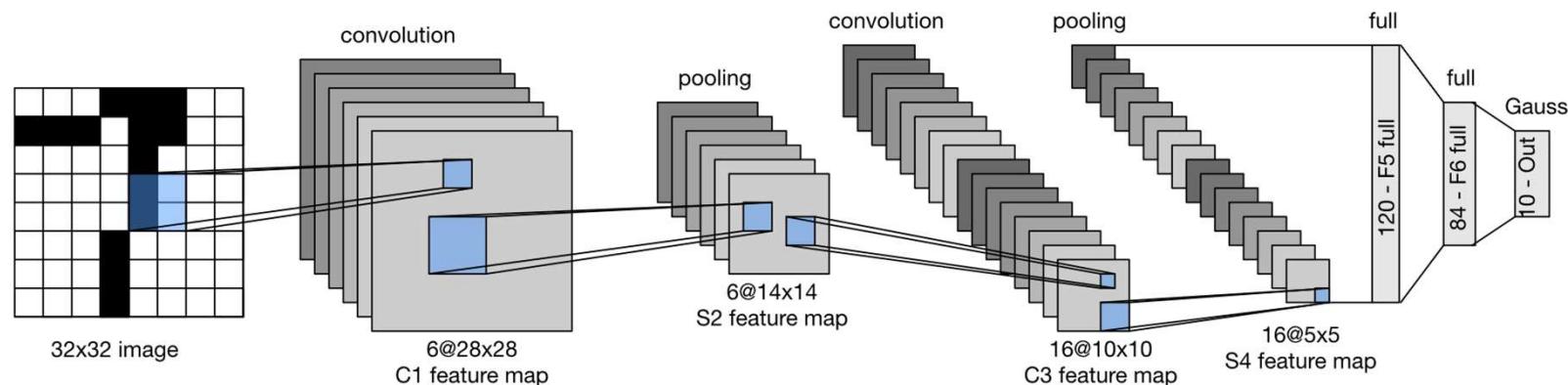
Pooled Feature  
Map



1
1
0
4
2
1
0
2
1

# CNN (Convolutional Neural Network, 합성곱 신경망)

- LeNet : 5 개층 Yan Le Cunn - 1998



입력

합성곱

1	1	3	4
3	6	2	8
3	9	1	0
1	3	3	4

풀링

6	8
9	4

합성곱

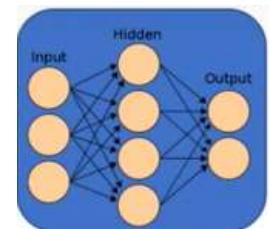
1	1	3	4
3	6	2	8
3	9	1	0
1	3	3	4

풀링

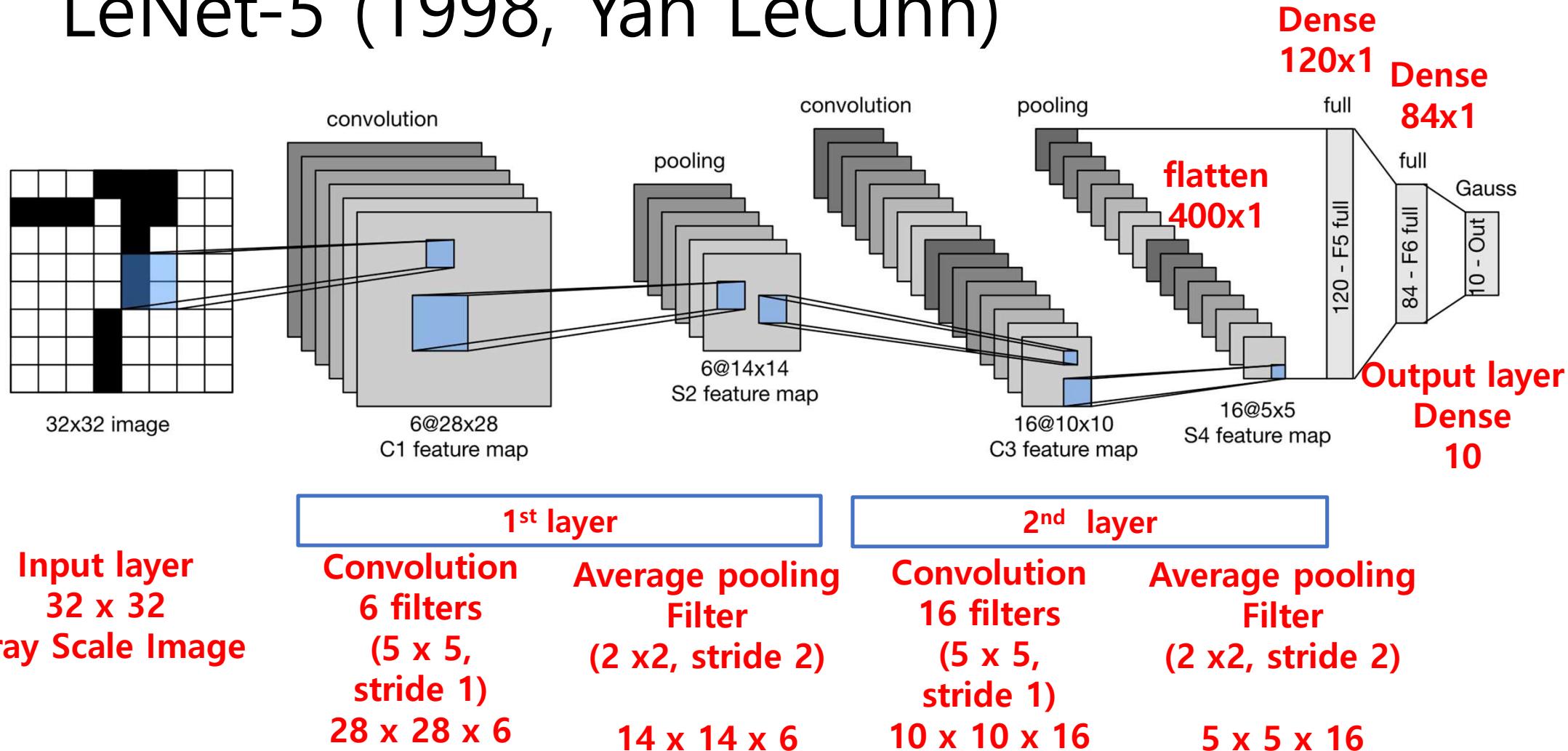
6	8
9	4

완전연결

6
8
9
4



# LeNet-5 (1998, Yan LeCunn)



# Famous CNN models

- Alex Net – 2012 년 ILSVRC(ImageNet Large Scale Visual Recognition Competition) 대회 우승
- GoogleLeNet(Inception Net) – 2014 년 ILSVRC 대회 우승
- ResNet – 2015 년 ILSVRC 대회 우승 (152 개 층)
- MobileNet – mobile device 용 pre-trained model (ImageNet 20,000 개 classes)
- VGG-16 : Keras built-in pre-trained model (2014 년, 16 layers)

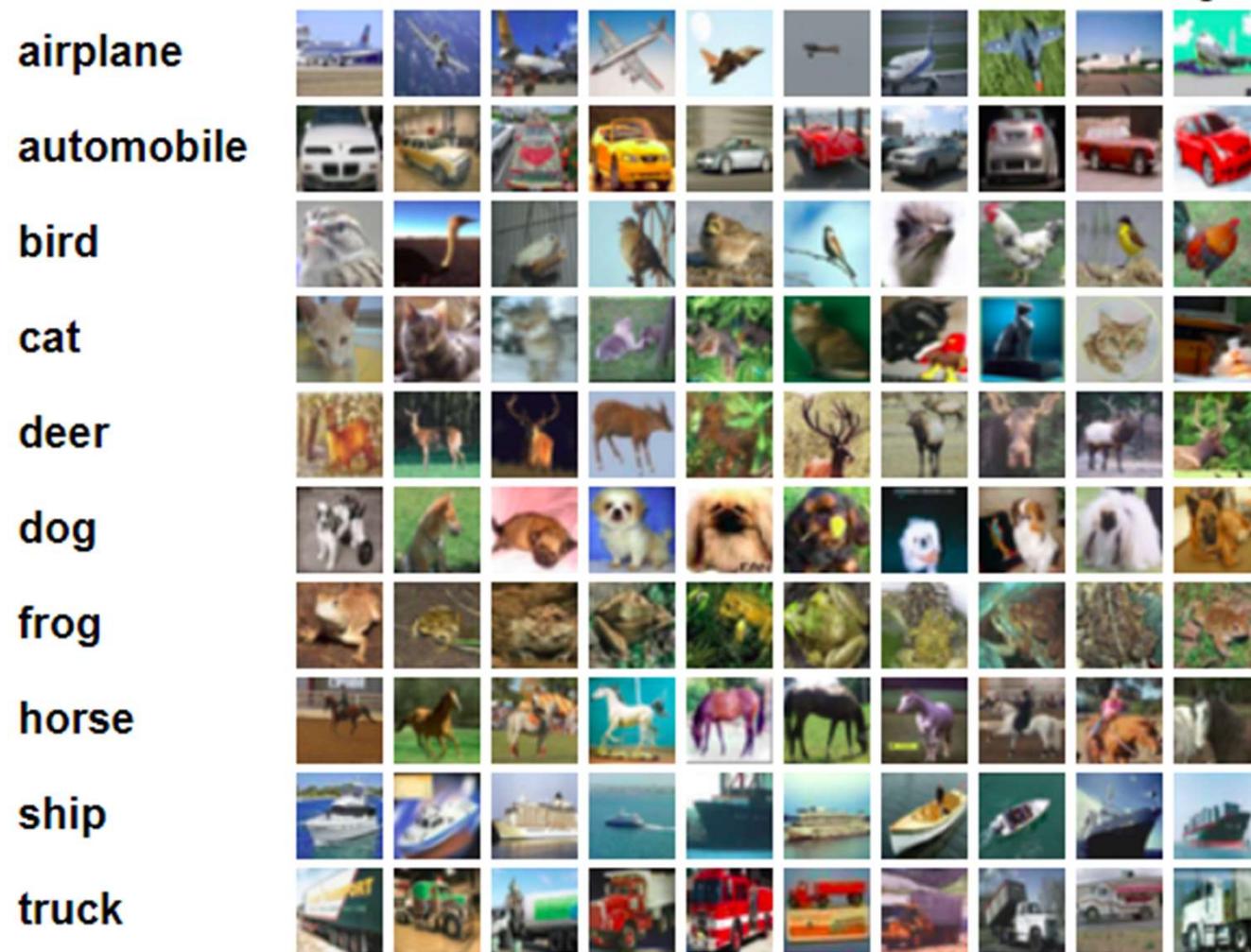
# 실습 : Le Net 을 이용한 손글씨 인식

1. Keras 를 이용한 Le Net 구축
2. 구축한 Le Net model 을 이용하여 Mnist 손글씨 분류

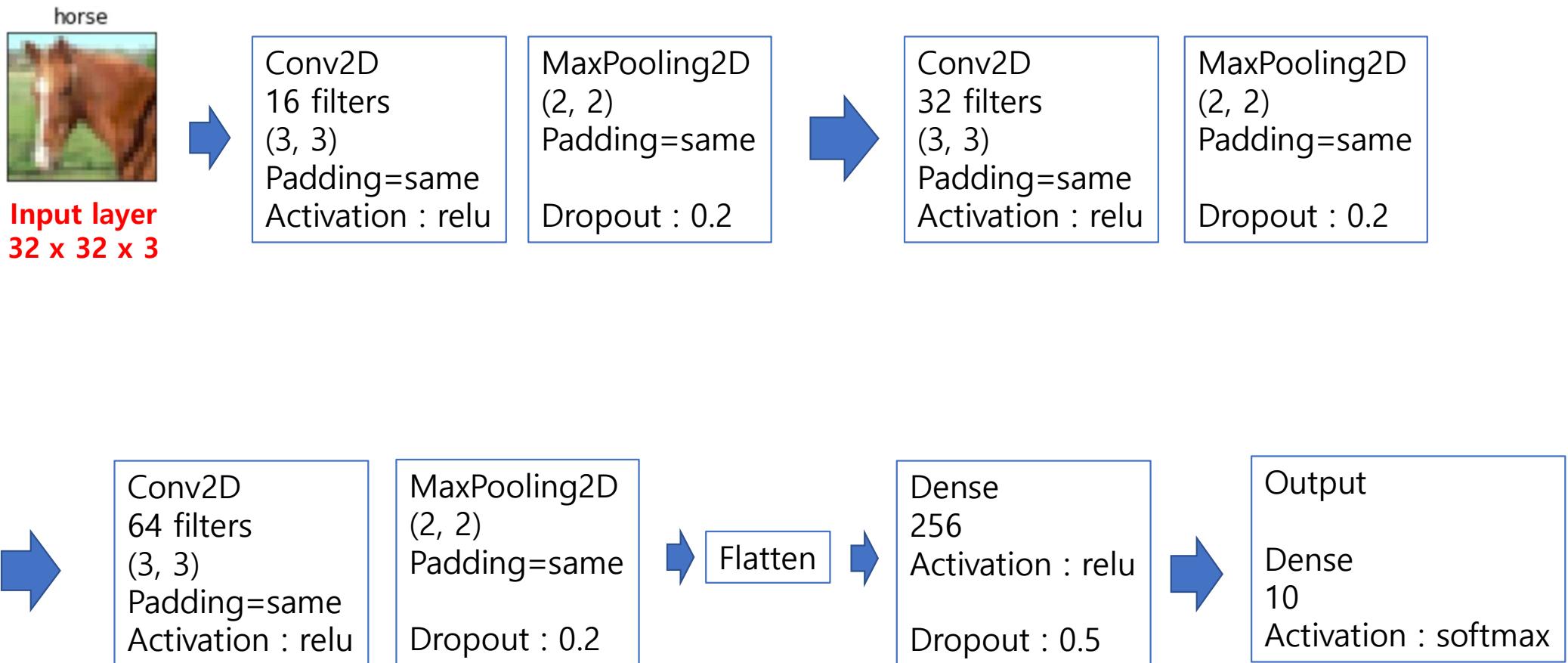
## 실습 : Deeper CNN 을 이용한 CIFAR-10 분류

1. CIFAR-10 dataset 은 32x32 color image 를 가진 10개의 class (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck)
2. 각 class 별 6,000 개씩 total 60,000 개 image
3. Image 가 blur 하여 난이도 높음  
(최근 성적 : <https://en.wikipedia.org/wiki/CIFAR-10>)
4. Google Colab GPU 환경 이용

# CIFAR-10 image dataset



# Neural Network Architecture



# What is JSON / YAML ?

- JSON (**JavaScript Object Notation**)

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

- YAML (YAML Ain't Markup Language)

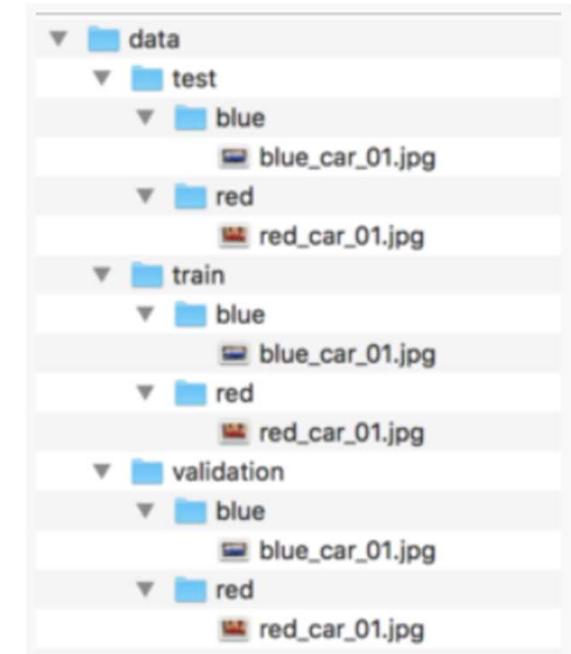
```
invoice: 34843  
  date : 2001-01-23  
  bill-to: &id001  
  given : Chris  
  family : Dumars  
  address: lines: | 458 Walkman Dr. Suite #292  
  city : Royal Oak  
  state : MI  
  postal : 48046
```

실습 : 구축한 model 의 weight 와  
architecture 저장 및 loading

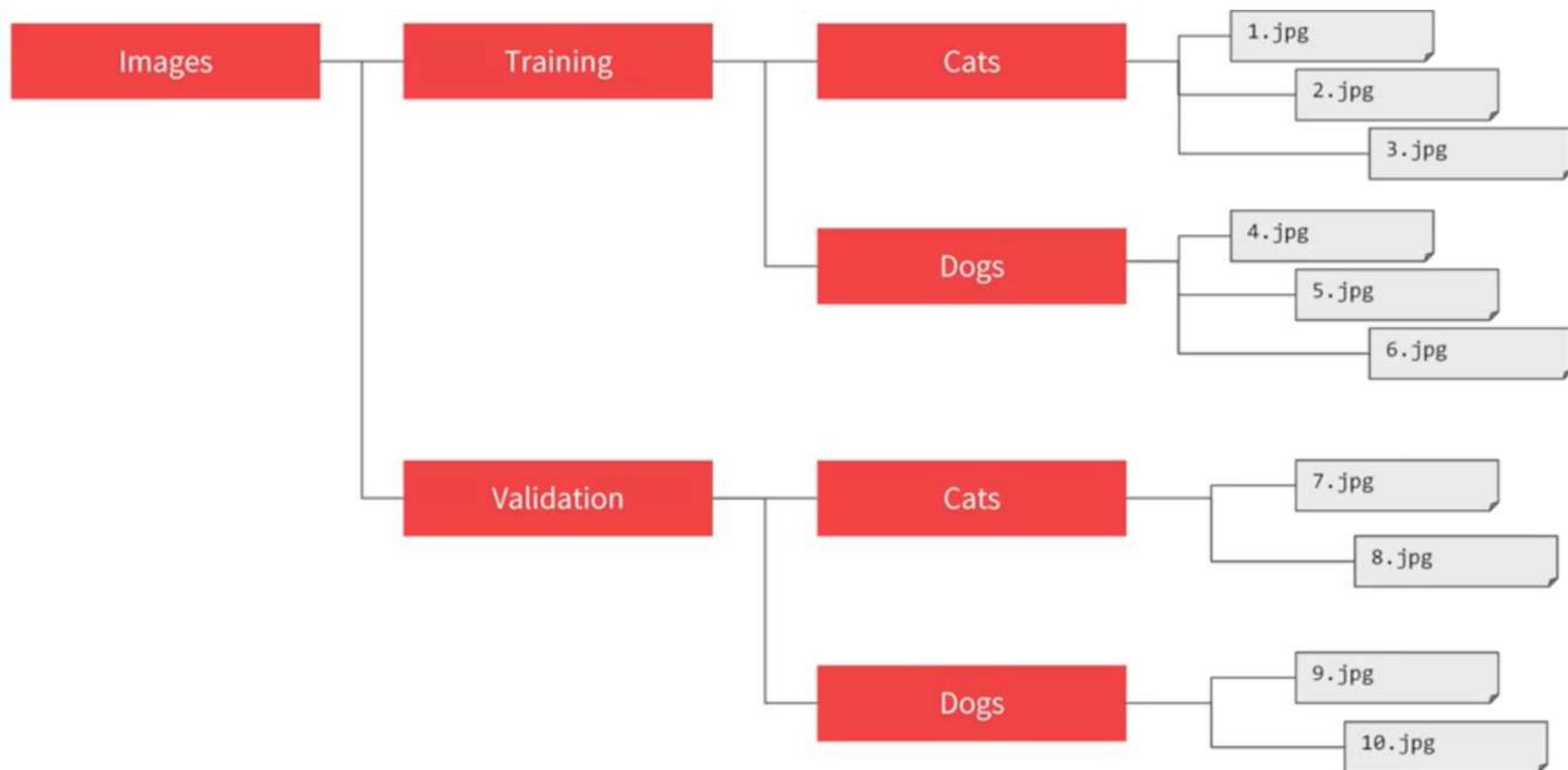
1. JSON / YAML file 로 model architecture 저장
2. JSON / YAML file 로 부터 model 복원
3. HDF5 file 로 model 의 parameter 저장 및 복원

# ImageDataGenerator methods

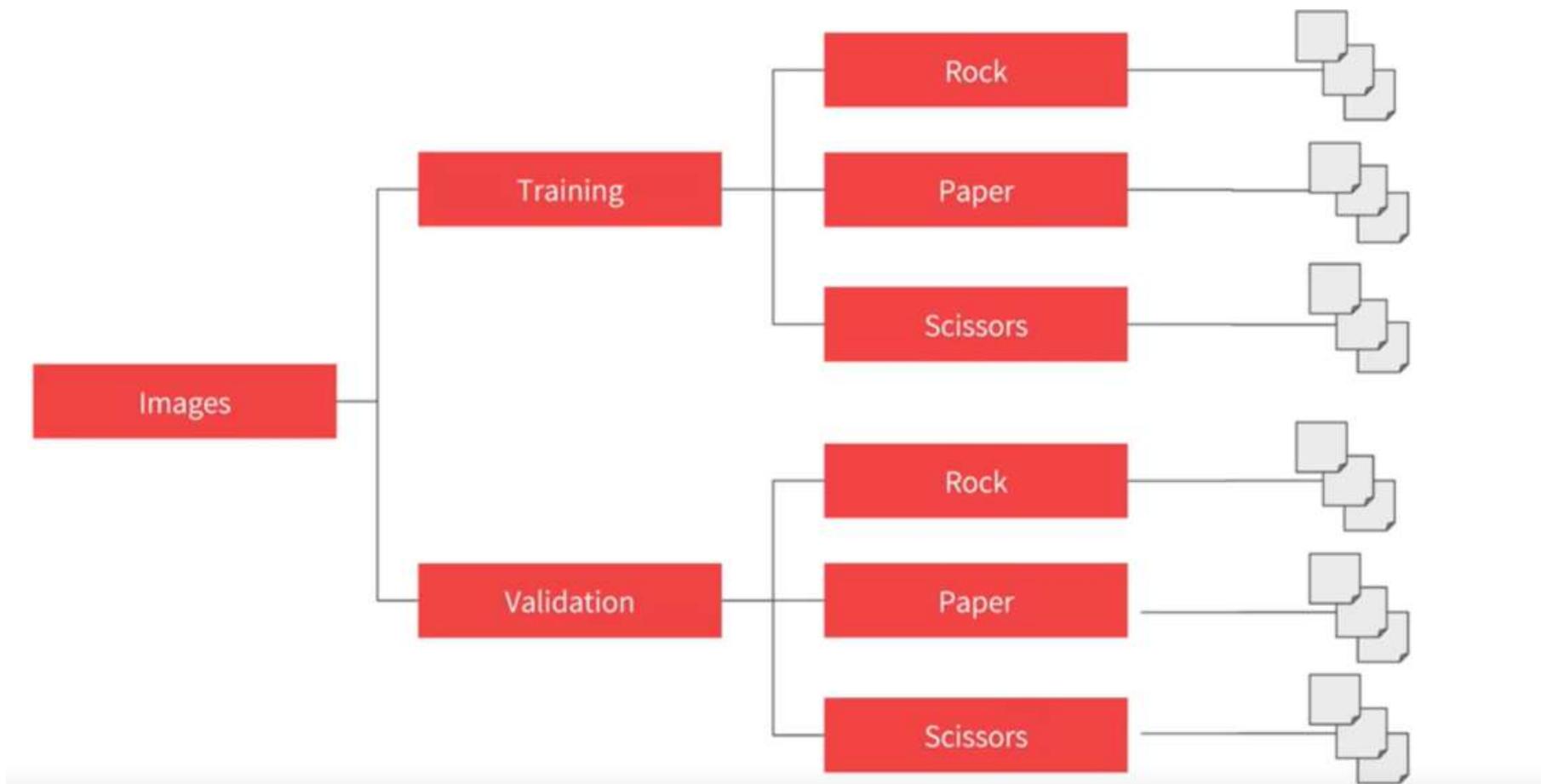
- .flow (X\_train, y\_train, batch\_size) : augmented data 반환
- .flow\_from\_directory
  - 대용량 data 를 directory 에서 직접 load  
(with data augmentation)
  - directory 구조에 의해 자동으로 label 인식
- .flow\_from\_dataframe
  - 모든 image 가 하나의 folder 에 있는 경우



# ImageDataGenerator folder structure



# ImageDataGenerator folder structure



# flow\_from\_directory 사용법

- from tensorflow.keras.preprocessing.image import ImageDataGenerator

- Instance 생성 :

```
train_data_gen = ImageDataGenerator(rescale=1/255.)
```

- flow\_from\_directory method 호출 :

```
train_generator = train_data_gen.flow_from_directory(  
    train_dir,  
    target_size(150, 150), → image size 통일  
    batch_size=20,  
    class_mode='binary' or 'categorical')
```

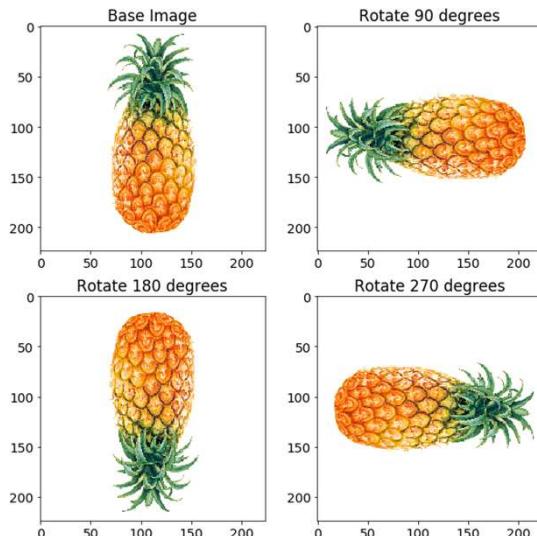
# flow\_from\_directory 사용법

- history = model.fit\_generator(  
    train\_generator,  
    step\_per\_epoch=100, → total image 개수 / batch\_size  
    epochs=15,  
    validation\_data=validation\_generator,  
    validation\_steps=50,  
    verbose=2)

# Data Augmentation

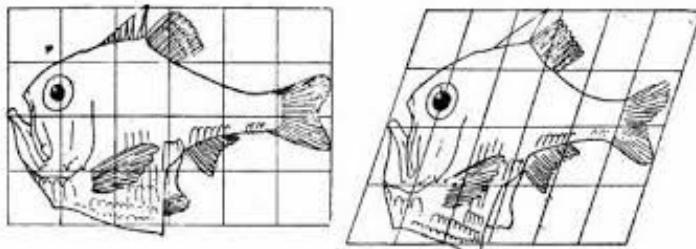
- 목적
  1. 부족한 training data increase
  2. overfitting 방지
- Keras.preprocessing.image 의 ImageDataGenerator 클래스 이용
- ImageDataGenerator
  - flow, flow\_from\_directory, flow\_from\_dataframe method 사용
- model.fit\_generator 를 이용하여 train
- Train, Validation dataset 에 모두 적용.

# Data Augmentation

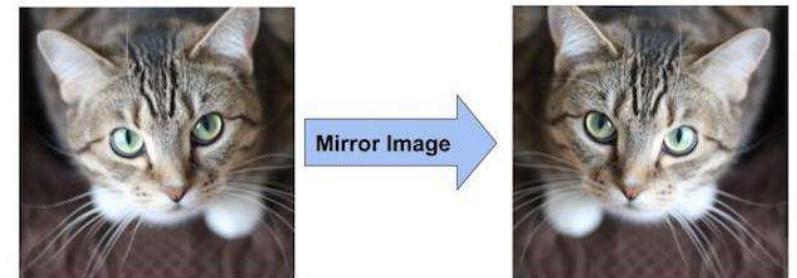


Rotation

Shearing



Mirroring



Cropping

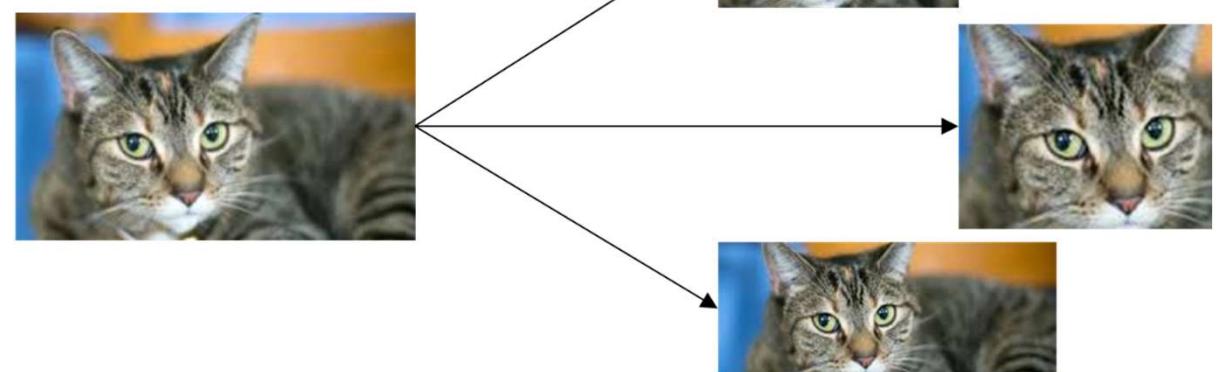


Fig. 517. *Argyropelecus Olfersii*.

Fig. 518. *Sternopyx diaphana*.

# Data Augmentation

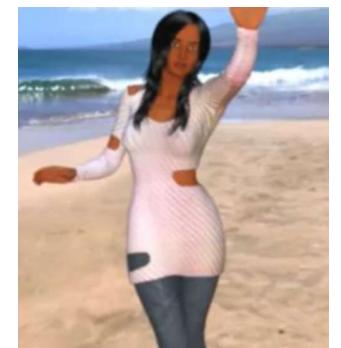
Color shifting



zoom



Horizontal Flip



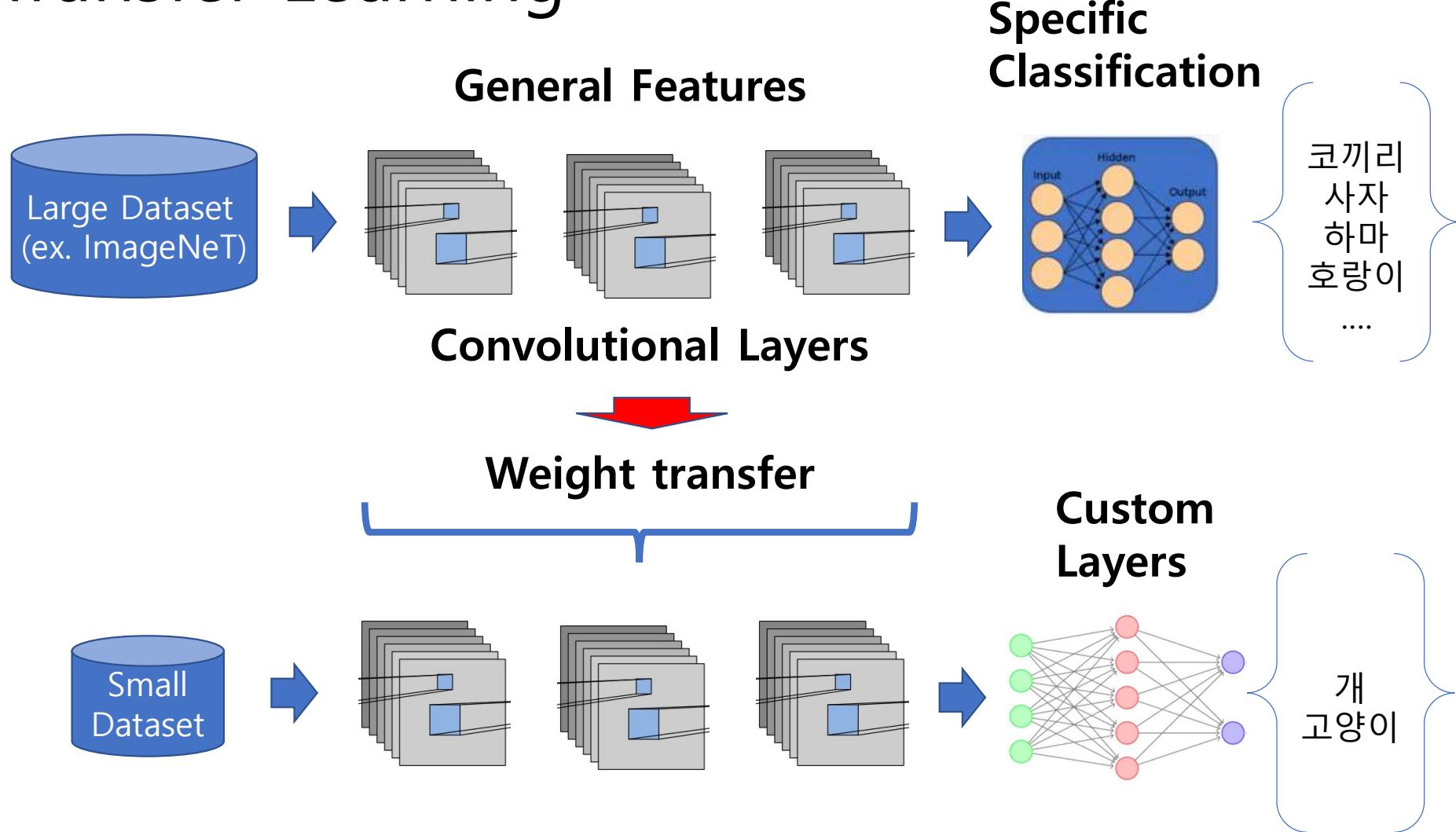
# ImageDataGenerator parameters

- train\_datagenerator = ImageDataGenerator(  
    rescale=1./255,  
    rotation\_range=40,   → 0~40 도 사이에서 random 회전  
    width\_shift\_range=0.2,   → 사진 중심 20% 이동  
    height\_shift\_range=0.2,  
    shear\_range=0.2   → 20% shear  
    zoom\_range=0.2   → 20% 까지 randomly zoom  
    horizontal\_flip=True,  
    fill\_mode='nearest')   → 생성된 공간은 인접 pixel로 채움

# 실습: ImageDataGenerator method

- from\_flow\_directory 및 ImageDataGenerator 를 이용한 data feed 구조화 및 Data Augmentation
- Kaggle 의 ["Dogs vs. Cats" dataset](#) 의 25,000 image 중 2,000 매 발췌.
- 소량의 image data (2000 장) 를 이용하여 model 학습  
→ overfitting 유도
- Data Augmentation 을 이용한 overfitting 해결

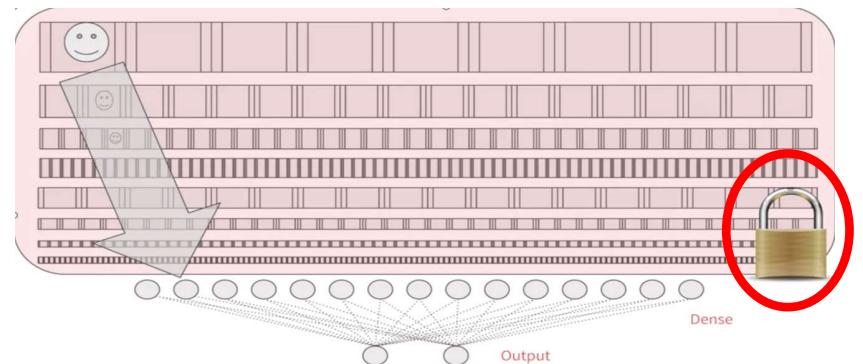
# Transfer Learning



# Transfer Learning Strategy

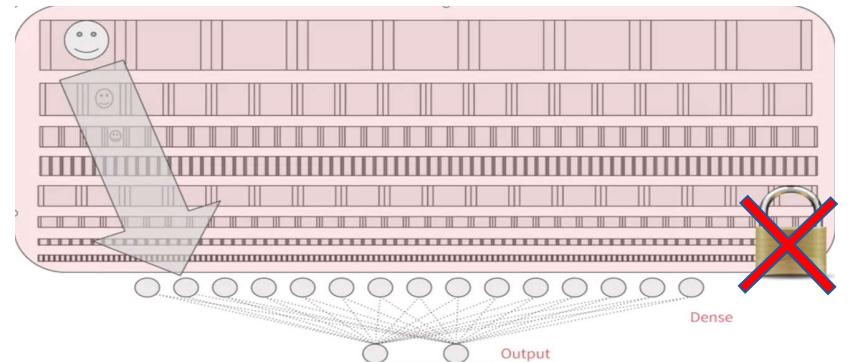
- Strategy 1

CNN layer 는 freeze 하고,  
추가한 완전연결층만 새로이 train



- Strategy 2

전체 Layer 를 매우 작은 learning rate 로 re-train



# Transfer Learning 고려사항

- 목적에 맞는 **dataset** 선택

ex) Cat & Dog 구분 → ImageNet 에 포함  
Cancer cell 구분 → ImageNet 에 없음

- 보유 Data 의 **Volume** 고려

1. 모든 weight 새로이 training (Large Data 보유)
2. Weight 의 일부만 training
3. 마지막 layer 만 Fine-tuning (Small Data 보유)

# 실습 : Pre-trained model 을 이용한 image 분류

- Keras 에 내장된 ResNet50 pre-trained model 을 이용
- ResNet50 pre-trained model 을 include\_top = False/True 의 두가지로 fine tuning 하여 비교

```
base_model = tf.keras.applications.ResNet50(weights = 'imagenet', include_top = False)
full_model = tf.keras.applications.ResNet50(weights='imagenet', include_top = True)
```

**include\_top=False**

conv5\_block3\_2\_relu  
conv5\_block3\_3\_conv  
conv5\_block3\_3\_bn  
conv5\_block3\_add  
conv5\_block3\_out

conv5\_block3\_2\_relu  
conv5\_block3\_3\_conv  
conv5\_block3\_3\_bn  
conv5\_block3\_add  
conv5\_block3\_out  
avg\_pool  
probs

**include\_top=True**

- 임의의 image 를 ResNet50 의 입력 spec 에 맞도록 resize

```
tf.keras.applications.resnet50.preprocess_input(Sample_Image)
```

- decode\_predictions 를 이용한 결과값 비교

```
tf.keras.applications.resnet50.decode_predictions(predictions, top = 5)[0]
```

- base model 의 마지막에 5 개 layer 추가

```
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)

x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(512, activation = 'relu')(x)
preds = tf.keras.layers.Dense(2, activation = 'softmax')(x)
```

- 마지막 50 개 layer 만 fine tuning

```
for layer in model.layers[:-50]:  
    layer.trainable = False
```

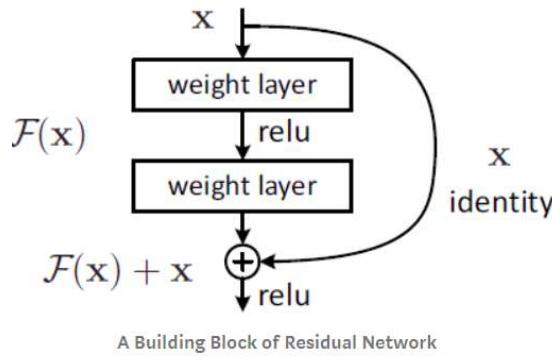
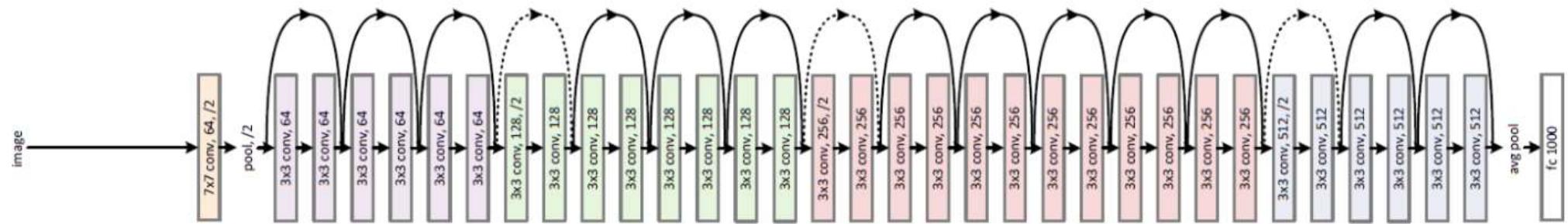
```
for layer in model.layers[-50:]:  
    layer.trainable = True
```

## ImageNet

- Open Source Image Repository <http://www.image-net.org/>
- 1000 classes over 1.5 MM images

# ResNet Structure

34-layer residual



- vanishing/exploding gradients 문제 해결을 위해 skip / shortcut connection 을 추가
- input  $x$  를 몇 개 layer 의 output에 추가

# Deep Learning

# Sequence Model

# What is sequence data ?

- Speech recognition : 파동의 연속 → 단어의 연속으로 변환
- Music generation : 연속된 음표 출력
- Sentiment classification : Text → 평점, 부정/긍정 판단
- DNA 분석 : 염기서열 → 질병유무, 단백질 종류 등
- 자동 번역 : 한국어 → 영어
- Video activity recognition : 연속된 장면 → 행동 판단
- Financial Data : 시계열자료 → 주가, 환율 예측 등

# Problem of Standard Neural Network

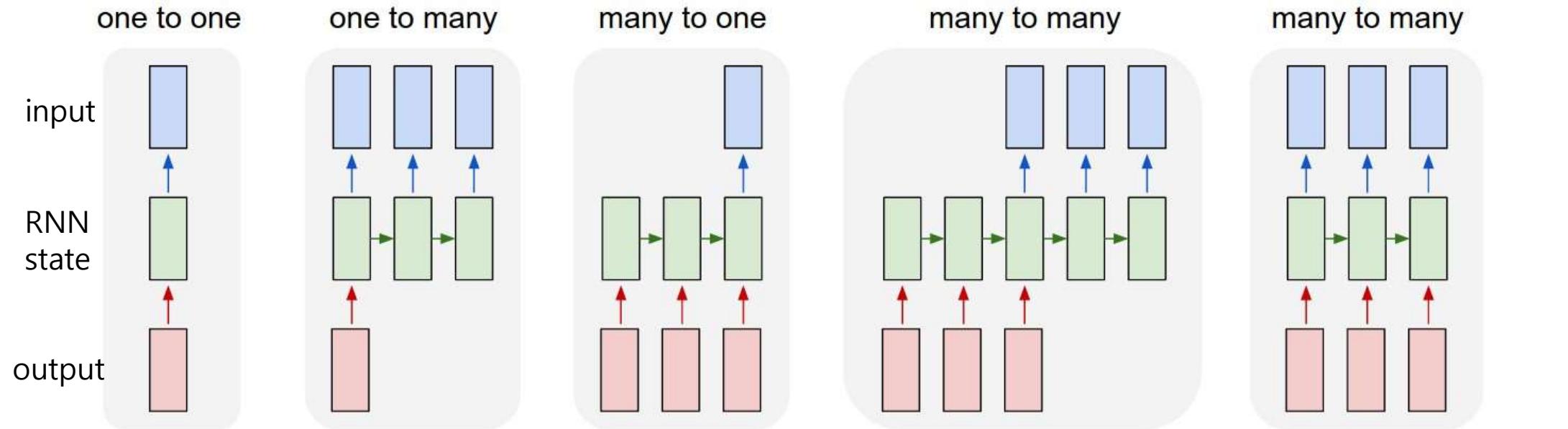
- 입력의 길이와 출력의 길이가 고정되어 있음
  - 따라서 standard NN 에서는 maximum input length 정하고 초과하면 truncate, 모자라면 padding 을 해 주어야 한다.
- 입력 데이터의 순서를 무시 (모든 observation 은 독립적임)

# RNN (Recurrent Neural Network)

# RNN (Recurrent Neural Network)

- 시퀀스 데이터에 특화
- '기억' 능력을 갖고 있음
  - \* 네트워크의 기억 - 지금까지의 입력 데이터를 요약한 정보  
(새로운 입력이 들어올때마다 네트워크는 자신의 기억을 조금씩 수정)
- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보  
(사람의 시퀀스 정보 처리 방식과 비슷, 기억을 바탕으로 새로운 단어 이해)
- 이 과정은 새로운 단어마다 계속해서 반복 → Recurrent (순환적)

# Different Types of RNN



예) 이미지 분류

- 이미지로 부터 문장 생성  
(Image Captioning)
- 작곡, 작시
- 감성 분석  
(sentiment Analysis → positive/negative)
- Spam detection
- 기계 번역 (영어 → 한국어 문장)
- Chatbot
- Question Answering
- video 각 frame에 label 생성
- 품사 tagging
- 개체명 인식
- Character 단위 문장 생성 등

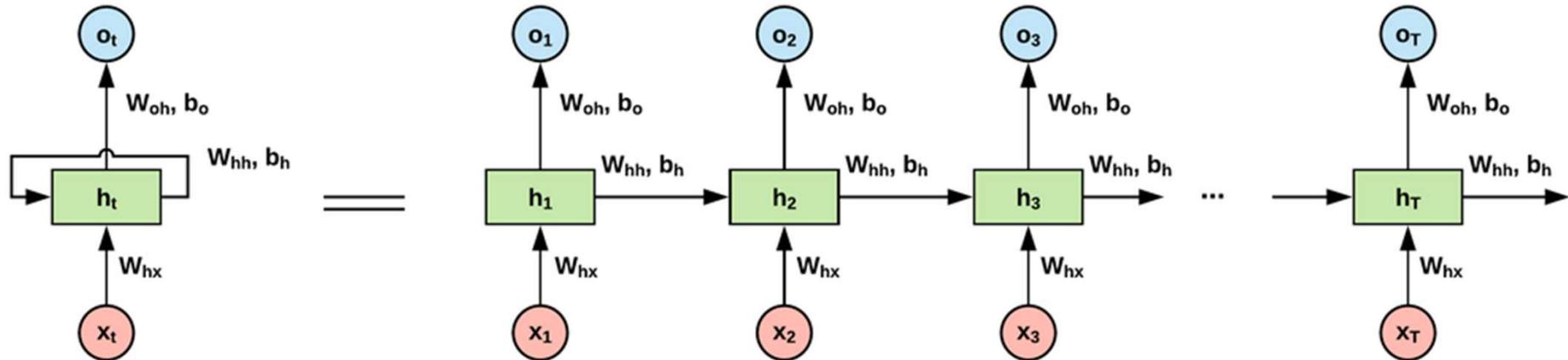
# RNN (Unfold 표시)

Internal State :

$$h_t = \tanh(W_h h_{t-1} + W_x x_t)$$

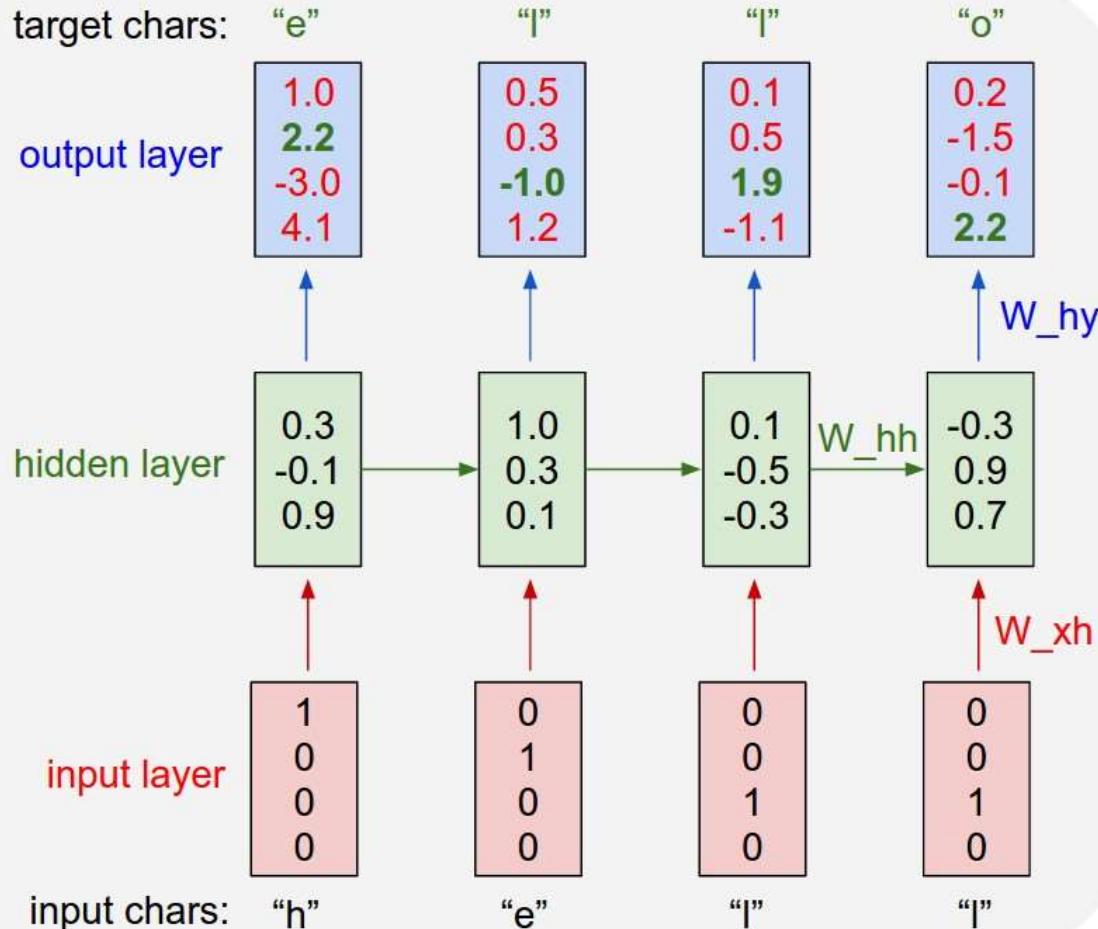
Output :

$$o_t = \text{softmax}(W_o h_t)$$



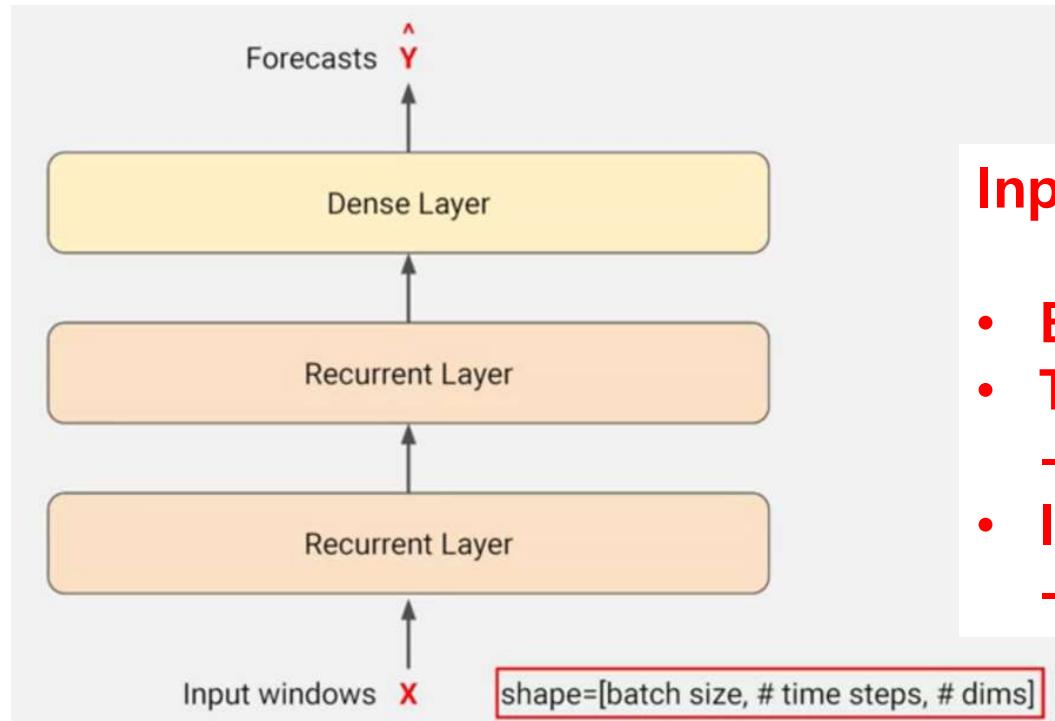
- RNN 을 순서대로 펼쳐 놓으면 weight 를 공유하는 매우 deep 한 neural network 이 된다.
- BPTT (Backpropagation Through Time) 으로 parameter 학습

# How RNN is trained ?



- "h", "e", "l", "o" 4 개 문자만 있다고 가정
- 각 문자를 One-hot encoding
- "h" 를 시작 문자로 주면 "hello" 가 출력 되도록 훈련
- 각 time step 의 target character 는 "hello" 내의 next character
- Gradient descent 와 backpropagation 을 통해 output layer 의 target score 가 증가되도록 함 (green color)
- "l" 다음의 character 는 현재의 "l" 만으로 판단할 수 없음 (history 필요)

# Recurrent Neural Network Overview



## Input shape – 3 차원 표시

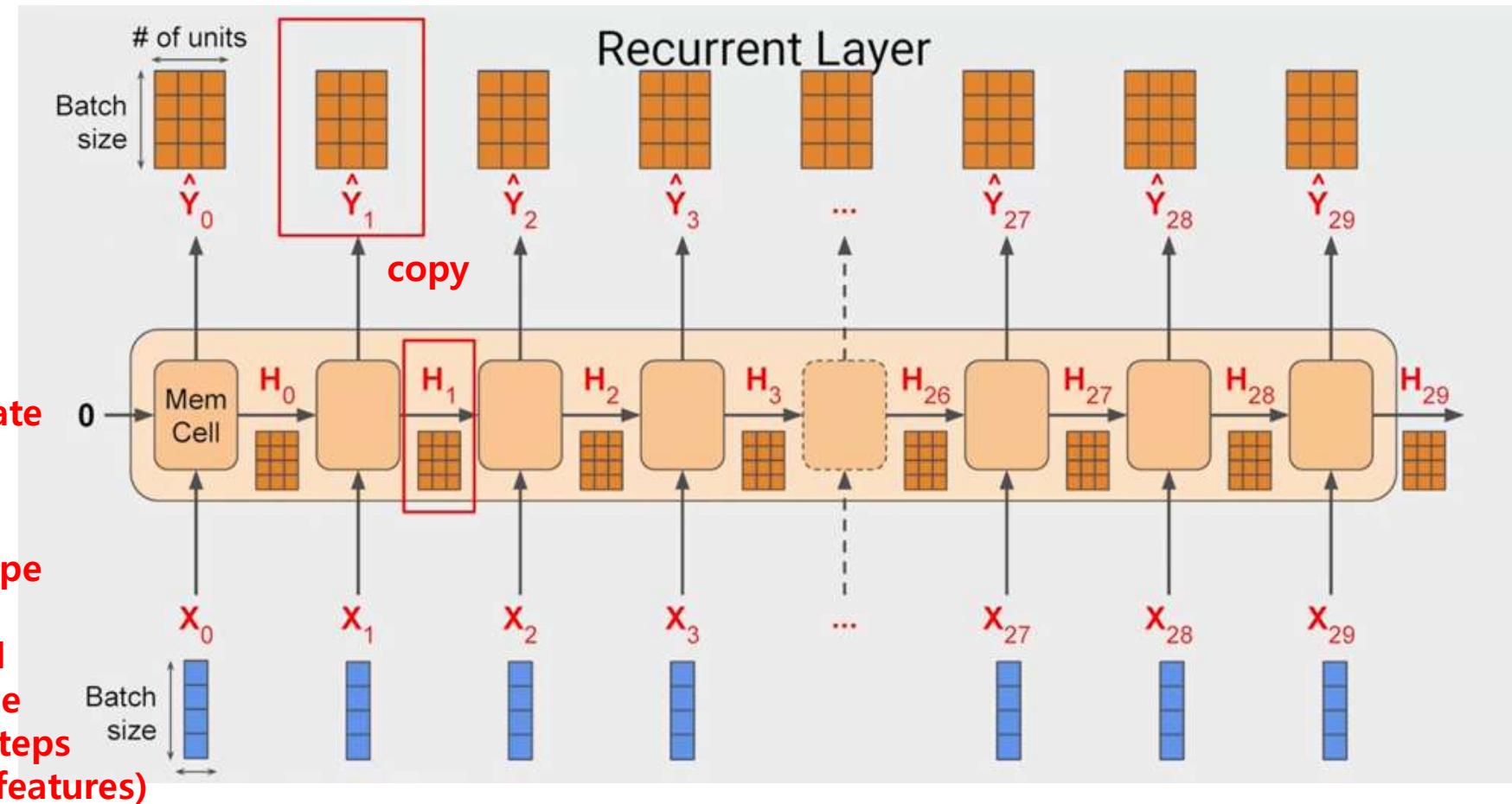
- **Batch size**
- **Time step**
  - sequence 변환된 text 최대 길이
- **Input features**
  - word embedding feature 수

```
sentences = [  
    'I love my dog',  
    'I love my cat',  
    'You love my dog!',  
    'I was born in Korea and graduaged University in USA.'  
]
```

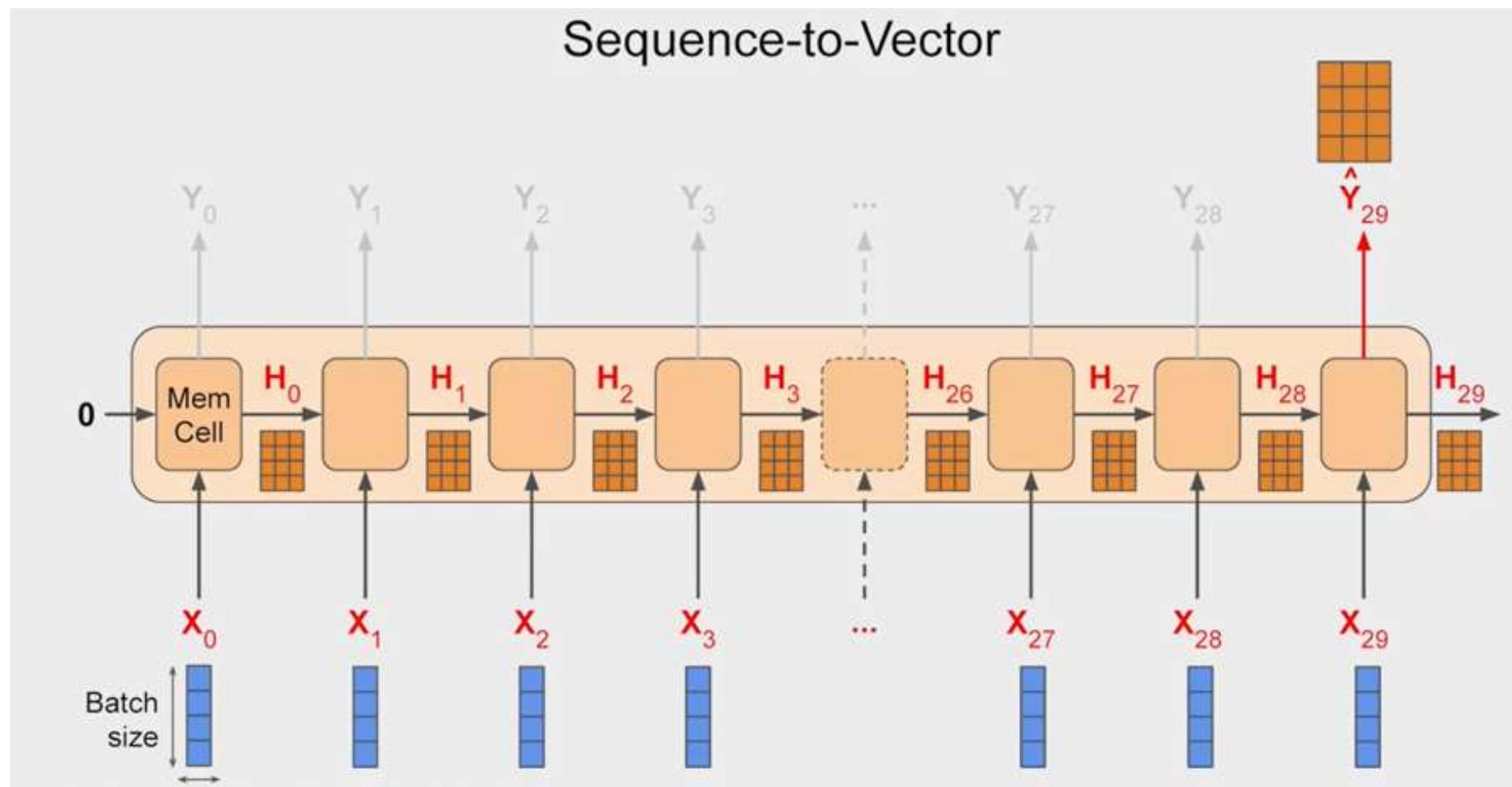


```
shape=[None, 10, 16]  
array([[ 0,  0,  0,  0,  0,  0,  2,  3,  4,  5],  
      [ 0,  0,  0,  0,  0,  0,  2,  3,  4,  7],  
      [ 0,  0,  0,  0,  0,  0,  8,  3,  4,  5],  
      [ 2,  9, 10,  6, 11, 12, 13, 14,  6, 15]])
```

# Simple RNN 의 input shape (feature=1 인 경우) 과 Time Step



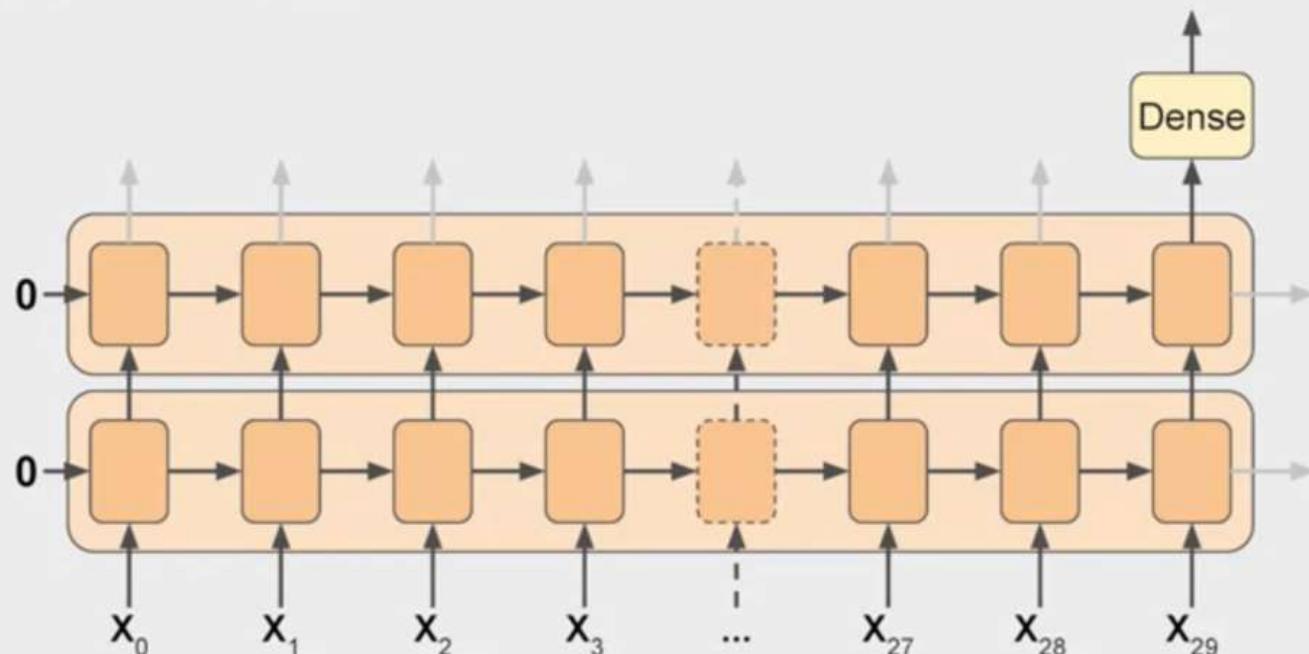
# Sequence-to-Vector (default of RNN)



# Sequence-to-Vector (Stacked RNN layers)

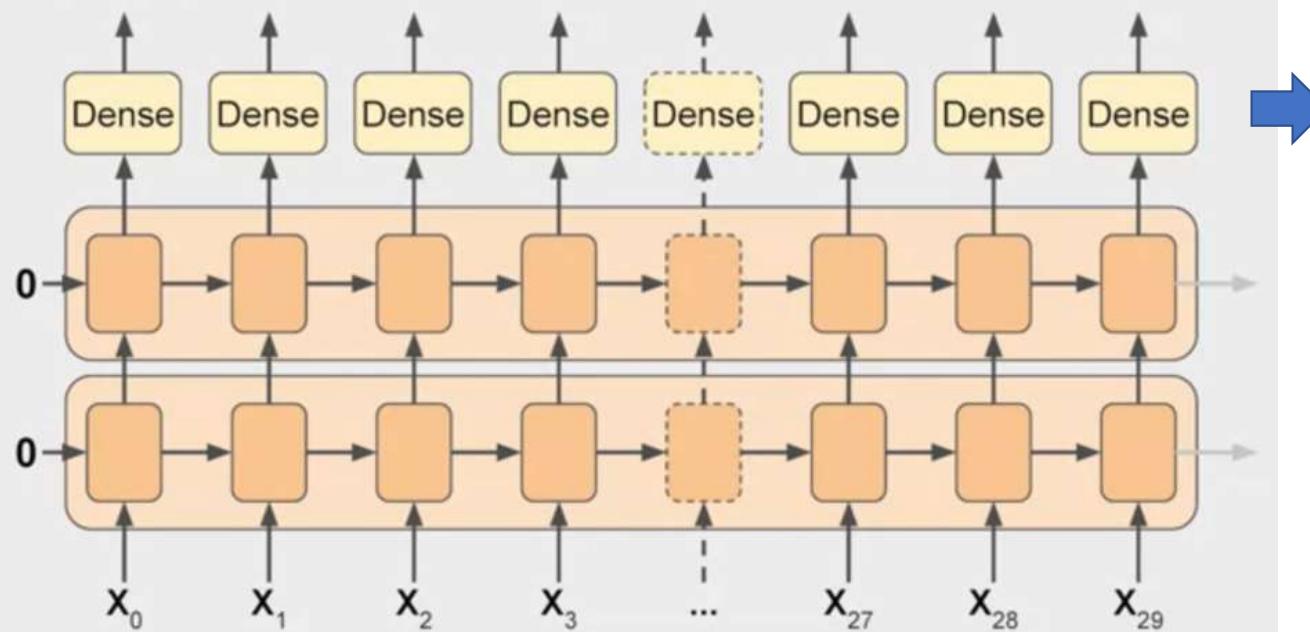
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True,
                           input_shape=[None, 1]),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1)
])
```

1<sup>st</sup> dimension : batch size 생략  
2<sup>nd</sup> dimension : None → any time step  
3<sup>rd</sup> dimension : feature 개수



# Sequence-to-Sequence RNN

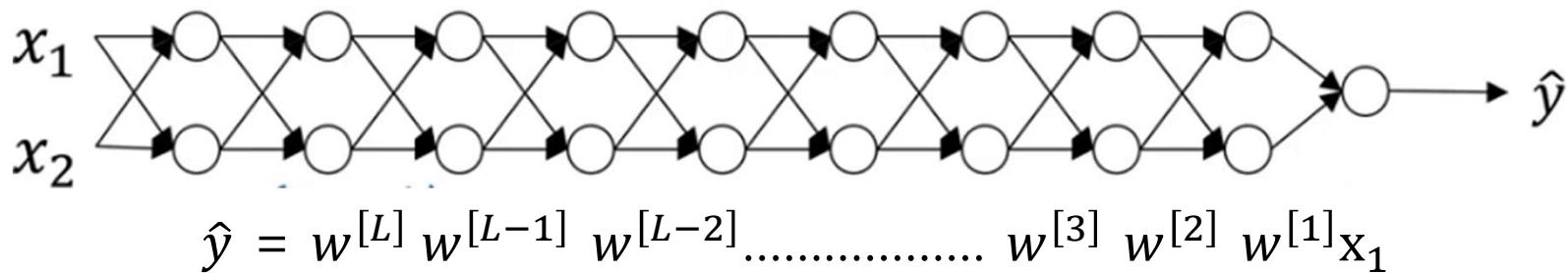
```
model = keras.models.Sequential([
    keras.layers.SimpleRNN(20, return_sequences=True,
                           input_shape=[None, 1]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.Dense(1)
])
```



각 time step마다 같은 Dense layer를 이용하여 output 출력

## Simple RNN 의 문제점 : Vanishing & Exploding Gradient

- 매우 deep 한 network 을 훈련시킬 경우 앞부분 Layer weight 의 미분값 크기가 매우 작아지거나 커지는 현상



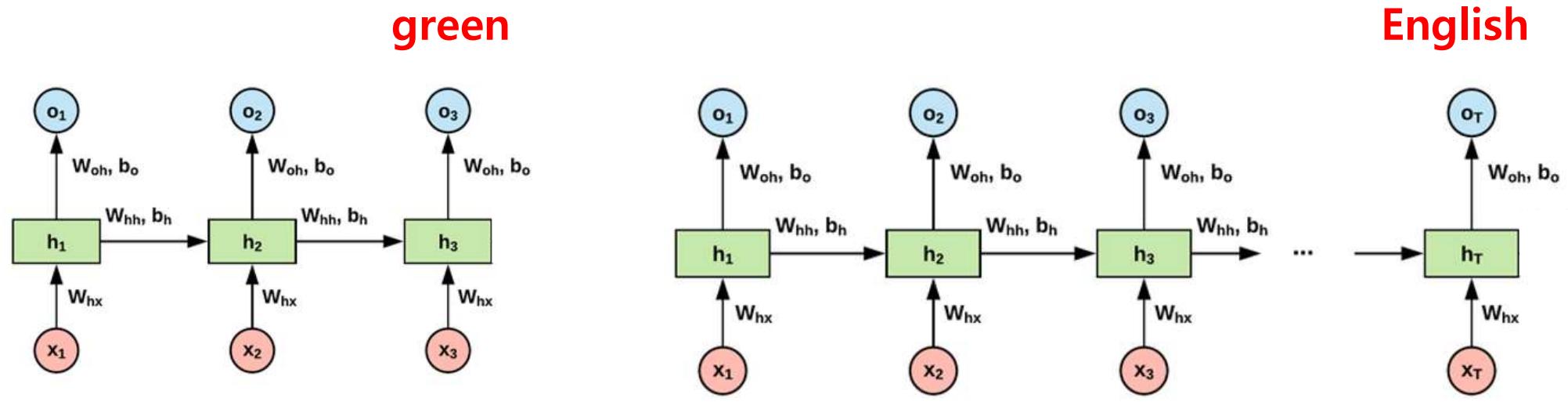
if  $w^{[1]} = 1.5 \rightarrow 1.5^L$  (Explode)  
if  $w^{[1]} = 0.5 \rightarrow 0.5^L$  (Vanish)

- 문제점 – Exploding – 훈련 자체가 안됨  
Vanishing – 경사하강법이 매우 느리게 진행

# Solutions of Vanishing Gradient

- Relu 사용
- Weight 의 신중한 초기화
  - Ex) Xavier (Glorot) Initializer with Tanh, He Initializer with Relu
- LSTM (Long Short Term Memory) /  
GRU (Gated Recurrent Unit) 사용
  - ➔ Vanishing Gradient + memory

# LSTM (Long Short-Term Memory)



Tree color is



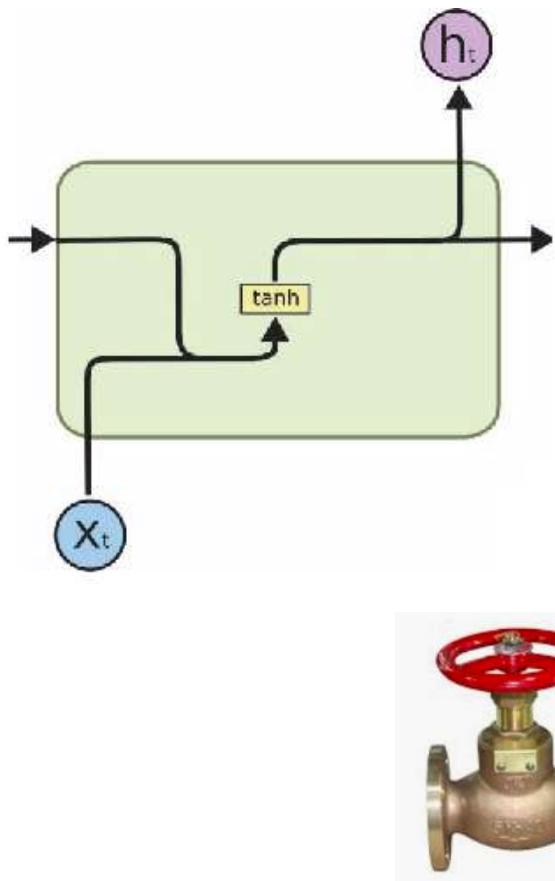
Simple RNN 으로 기억 가능

He was born in **England** and moved to Asia and his mother language is

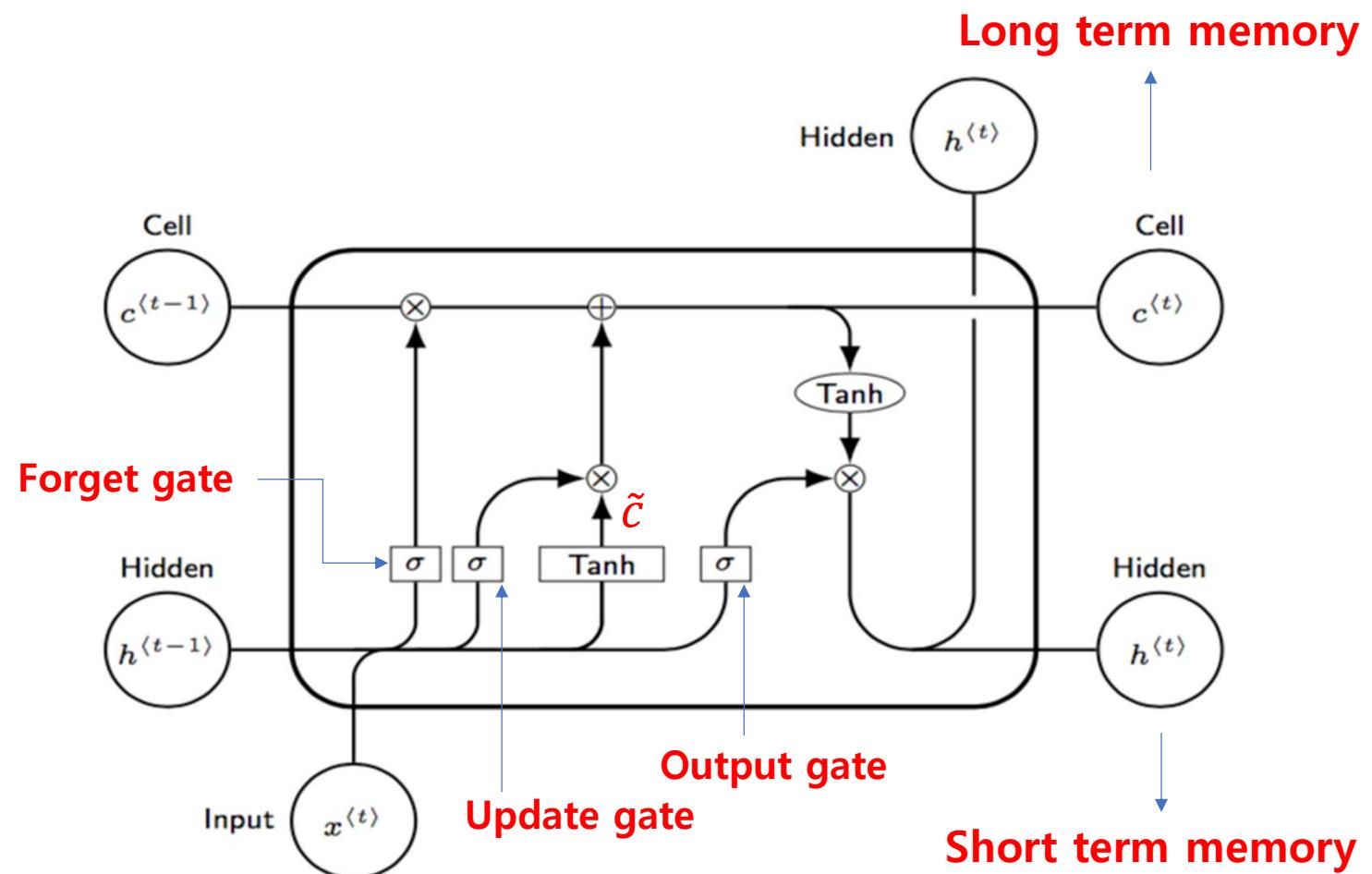


**Long-Term Memory 필요**

# SimpleRNN



# LSTM (Long Short-Term Memory)



# LSTM 내부 구조

- Input – 이전 step 의 output + new data

$$\tilde{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c) \rightarrow \text{새로운 cell status 후보}$$

- Update gate – 새로운 input 을 어느정도 받아들일지 결정 (0-무시, 1-전체)

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$

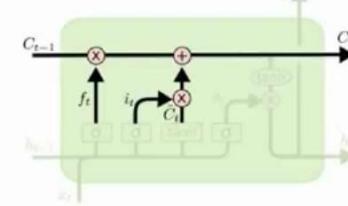
- Forget gate – 내부 state 를 어느정도 기억할지 결정 (0-forget, 1-전체 기억)

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$

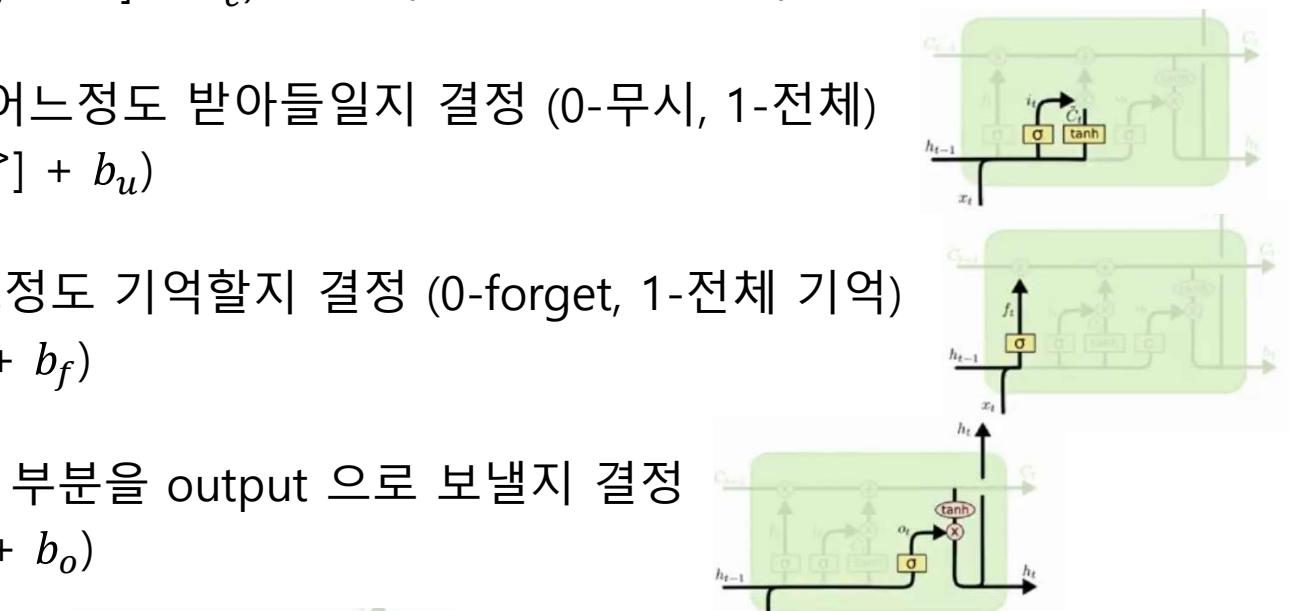
- Output gate – cell state 의 어느 부분을 output 으로 보낼지 결정

$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$

$$C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \Gamma_f * C^{<t-1>}$$



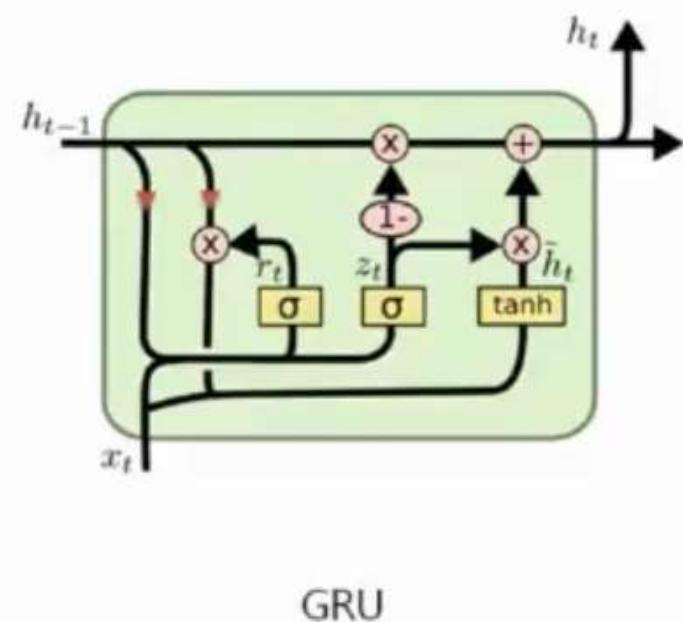
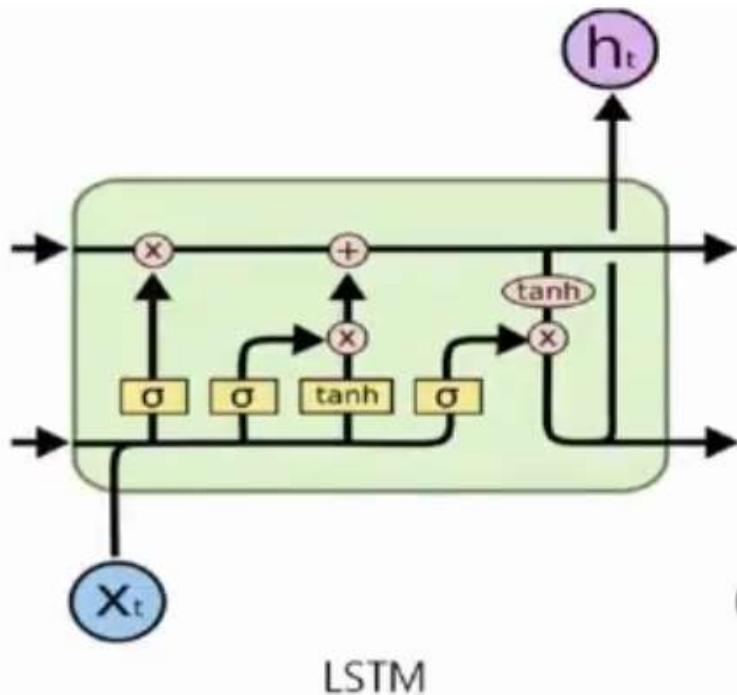
$$a^{<t>} = \Gamma_o * \tanh C^{<t>}$$



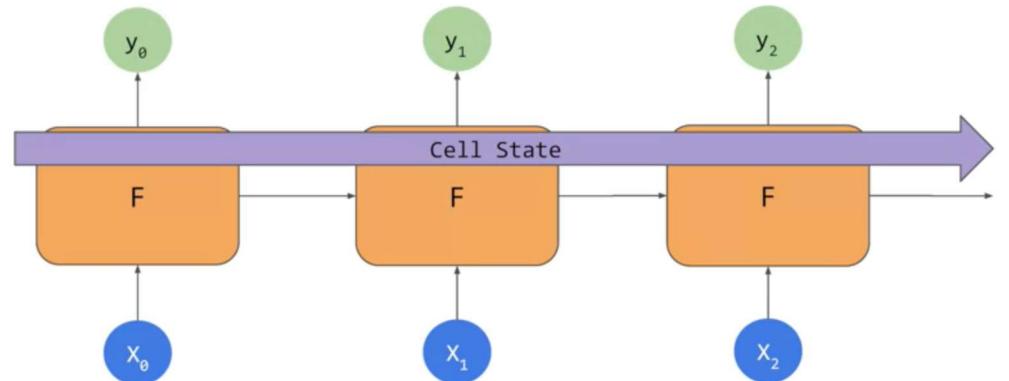
Past cell status \* forget gate + Current cell status candidate \* update gate

# GRU (Gated Recurrent Unit)

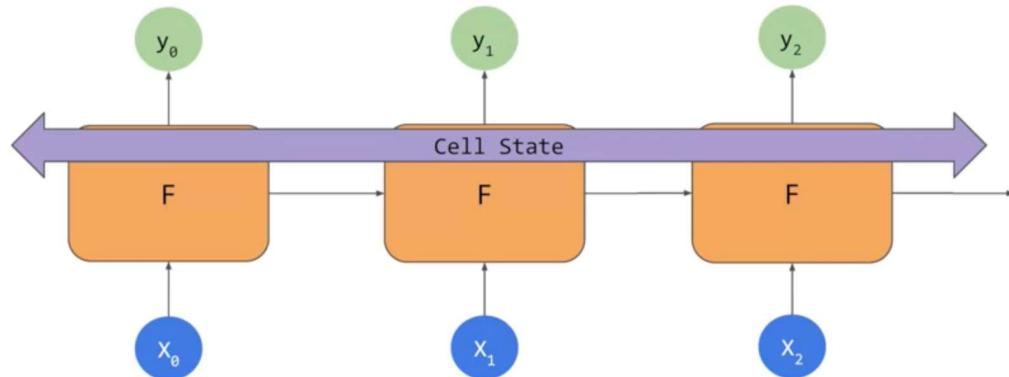
- LSTM 의 장점을 유지하면서 일부 Gate 를 생략하여 계산의 복잡성을 낮춤



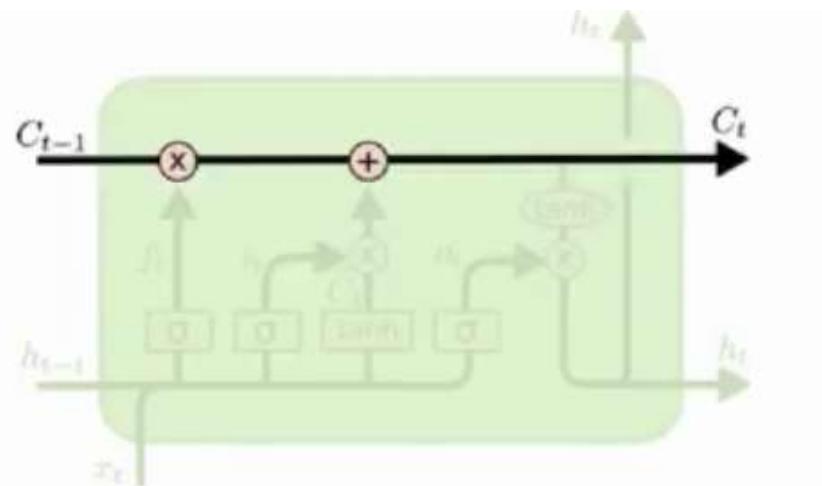
# LSTM Cell State



Uni-directional



Bi-directional



## When use / not to use **Bi-directional RNN**

- 미래를 예측할 경우 (내일의 주가, 경기 결과 등) – **not to use**
- **Language Model** 의 품사 tagging 등에 유용 - **use**

# 자연어 처리(NLP)의 RNN 적용

# 단어(word) 와 문장(sentence)의 표현 방법

- Text 전 처리 – 특수문자 제거, 불용어 제거 등
- Token 분리 – 단어(형태소) 단위 분리
- One-hot-encoding
- Word Embedding

# 단어의 Vector 표현

- One-Hot-Encoding

Vocabulary 사전

a -1  
aaron – 2  
. .  
apple – 456  
. .  
king – 4914  
. .  
orange – 6257  
. .  
queen – 7157  
. .  
zebra – 9,999  
<UNK> - 10,000

→ **One-Hot encoding** →

King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

10,000

# Word Embedding

- 단순 One-Hot-Encoding 의 문제점 – 단어 간의 유사도가 표시되지 않음
- 수자화된 단어의 나열 → sentiment 추출
- 연관성 있는 단어들을 군집화하여 multi-dimension 공간에 vector 로 표시
- 예를 들어, 호감(positive), 비호감(negative) 두 가지 label 에 따라 관련 단어들을 두개의 category 로 군집화  
ex) IMDB : boring, bad, unfunny → negative  
              funny, good, interesting → positive

# Word Embedding (Feature 화 표시)

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
size	⋮	⋮	⋮	⋮	⋮	⋮
cost	⋮	⋮	⋮	⋮	⋮	⋮
alike	⋮	⋮	⋮	⋮	⋮	⋮
verb	⋮	⋮	⋮	⋮	⋮	⋮

↑

I like a glass of orange \_\_\_\_\_  
I like a glass of apple \_\_\_\_\_

Man (5931) 의 300 dimension vector 표시

# Embedding matrix (example)

학습된 features

words

	0	1	2	3	4	5	6	7	8	9	...	290	291	292	
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800	0.101
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200	0.354
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826	-0.721
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310	0.120
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200	-0.541
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080	-0.371
beans	-0.423290	-0.264500	0.200870	0.082187	0.066944	1.027600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260	-0.361
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970	-0.271
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730	-0.035
breakfast	0.073378	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931	-0.031
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167	0.051
today	-0.156570	0.594890	-0.031445	-0.077586	0.278630	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	-0.261
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250	-0.311
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300	-0.461
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.528600	...	-0.470090	0.063743	-0.545210	-0.191
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878	0.001
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980	-0.291
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.418680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670	-0.241
love	0.139490	0.534530	-0.252470	-0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.099469	0.001
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543	0.151

20 rows x 300 columns

# Embedding Layer 를 이용한 vector 표현

- Projection Matrix

$$\begin{bmatrix} 0 & 0 & 0 & \textcolor{green}{1} & 0 \end{bmatrix} \times \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \\ 4 & 6 & 13 \\ \textcolor{green}{10} & \textcolor{green}{12} & \textcolor{green}{19} \\ 11 & 18 & 25 \end{bmatrix} = \boxed{\begin{bmatrix} 10 & 12 & 19 \end{bmatrix}} \quad \xleftarrow{\text{Blue curved arrow}} \quad \textcolor{red}{x_k^{New}}$$

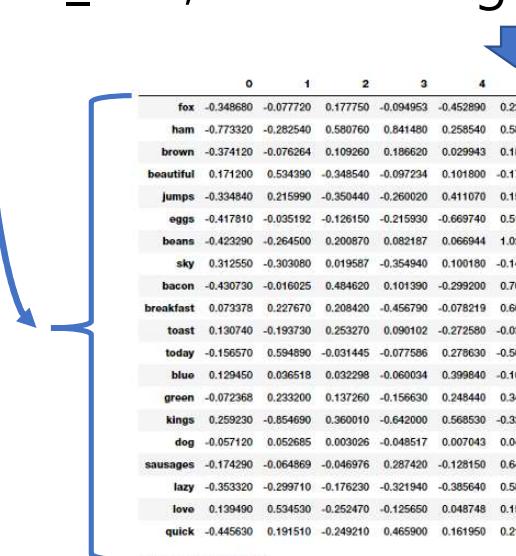
$x_k$                        $W_{V \times N}$

k 번째 단어 (One-Hot-Encoding)

- One-Hot-Vector 에 Projection Matrix(Embedding Layer) 를 곱해 새로운 vector 생성  $\rightarrow$  계산의 간편성
- Projection Matrix 의 k 번째 row 가 k 번째 단어에 대응하는 weight 임

# Neural Network 의 word embedding Layer

- 입력 data 에 대해 numpy matrix 연산을 manually 하는 것은 비 효율적이므로, tf.keras.layers.Embedding() 이용
- Tensorflow NLP model 의 1<sup>st</sup> layer 는 Embedding layer 로 시작
  - tf.keras.layers.Embedding(vocab\_size, embedding dimension)
  - One-hot-encoding
  - indexing

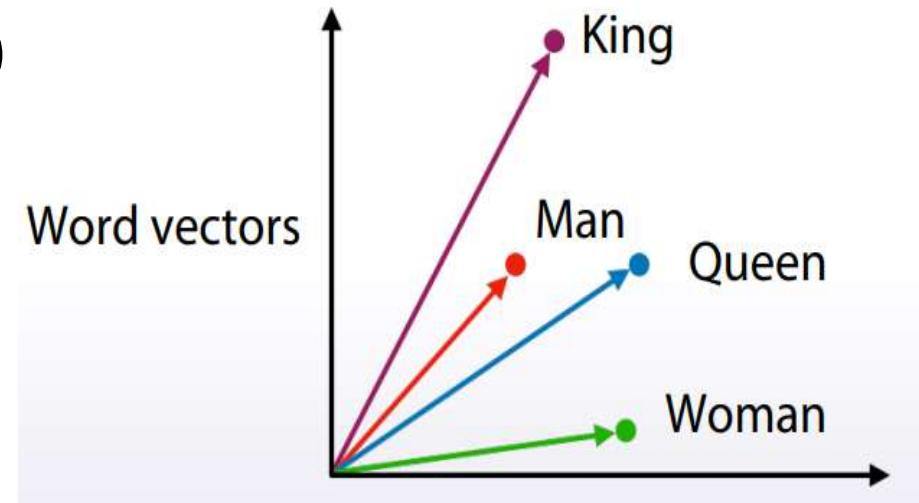


	0	1	2	3	4	5	6	7	8	9	...	290	291	292
fox	-0.348680	-0.077720	0.177750	-0.094953	-0.452890	0.237790	0.209440	0.037886	0.035064	0.899010	...	-0.283050	0.270240	-0.654800
ham	-0.773320	-0.282540	0.580760	0.841480	0.258540	0.585210	-0.021890	-0.463680	0.139070	0.658720	...	0.464470	0.481400	-0.829200
brown	-0.374120	-0.076264	0.109260	0.186620	0.029943	0.182700	-0.631980	0.133060	-0.128980	0.603430	...	-0.015404	0.392890	-0.034826
beautiful	0.171200	0.534390	-0.348540	-0.097234	0.101800	-0.170860	0.295650	-0.041816	-0.516550	2.117200	...	-0.285540	0.104670	0.126310
jumps	-0.334840	0.215990	-0.350440	-0.260020	0.411070	0.154010	-0.386110	0.206380	0.386700	1.460500	...	-0.107030	-0.279480	-0.186200
eggs	-0.417810	-0.035192	-0.126150	-0.215930	-0.669740	0.513250	-0.797090	-0.068611	0.634660	1.256300	...	-0.232860	-0.139740	-0.681080
beans	0.422390	-0.264500	0.200670	0.082187	0.066944	0.127600	-0.989140	-0.259950	0.145960	0.766450	...	0.048760	0.351680	-0.786260
sky	0.312550	-0.303080	0.019587	-0.354940	0.100180	-0.141530	-0.514270	0.886110	-0.530540	1.556600	...	-0.667050	0.279110	0.500970
bacon	-0.430730	-0.016025	0.484620	0.101390	-0.299200	0.761820	-0.353130	-0.325290	0.156730	0.873210	...	0.304240	0.413440	-0.540730
breakfast	0.073978	0.227670	0.208420	-0.456790	-0.078219	0.601960	-0.024494	-0.467980	0.054627	2.283700	...	0.647710	0.373820	0.019931
toast	0.130740	-0.193730	0.253270	0.090102	-0.272580	-0.030571	0.096945	-0.115060	0.484000	0.848380	...	0.142080	0.481910	0.045167
today	-0.156570	0.594890	-0.031445	-0.077586	-0.509210	-0.066350	-0.081890	-0.047986	2.803600	...	-0.326580	-0.413380	0.367910	
blue	0.129450	0.036518	0.032298	-0.060034	0.399840	-0.103020	-0.507880	0.076630	-0.422920	0.815730	...	-0.501280	0.169010	0.548250
green	-0.072368	0.233200	0.137260	-0.156630	0.248440	0.349870	-0.241700	-0.091426	-0.530150	1.341300	...	-0.405170	0.243570	0.437300
kings	0.259230	-0.854690	0.360010	-0.642000	0.568530	-0.321420	0.173250	0.133030	-0.089720	1.526600	...	-0.470090	0.063743	-0.545210
dog	-0.057120	0.052685	0.003026	-0.048517	0.007043	0.041856	-0.024704	-0.039783	0.009614	0.308416	...	0.003257	-0.036864	-0.043878
sausages	-0.174290	-0.064869	-0.046976	0.287420	-0.128150	0.647630	0.056315	-0.240440	-0.025094	0.502220	...	0.302240	0.195470	-0.653980
lazy	-0.353320	-0.299710	-0.176230	-0.321940	-0.385640	0.586110	0.411160	-0.416680	0.073093	1.486500	...	0.402310	-0.038554	-0.288670
love	0.139490	0.534530	-0.252470	0.125650	0.048748	0.152440	0.199060	-0.065970	0.128830	2.055900	...	-0.124380	0.178440	-0.009469
quick	-0.445630	0.191510	-0.249210	0.465900	0.161950	0.212780	-0.046480	0.021170	0.417660	1.686900	...	-0.329460	0.421860	-0.039543

20 rows x 300 columns

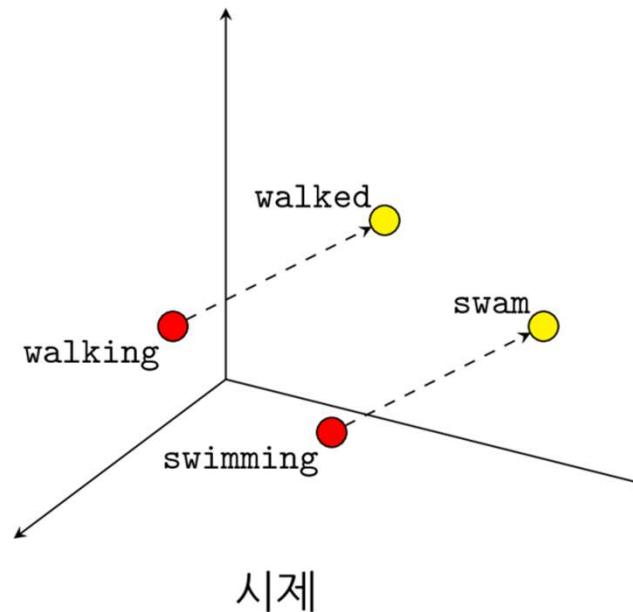
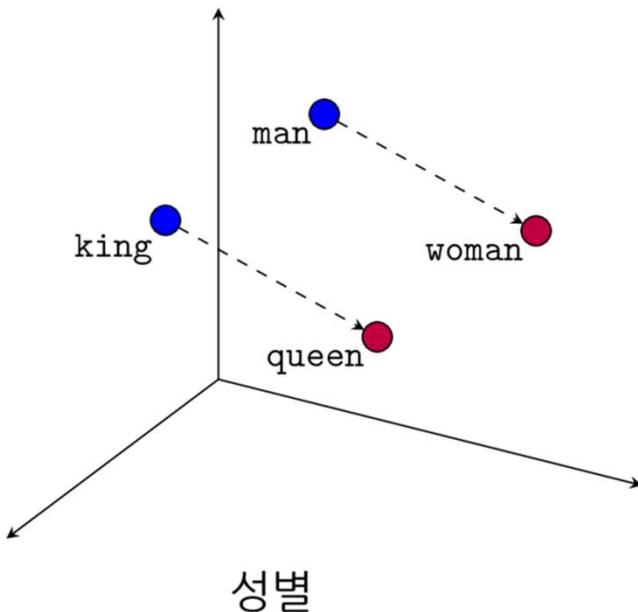
# Word2Vec

- 2013년 구글에서 개발
- 단어를 vector화 (Word Embedding) 하여 단어의 의미 표현
- 매우 큰 Corpus (ex, 10억, 100 억 단어)  
에서 word embedding 자동 학습



# Word2Vec Vectorized (Word Embedding)

king – queen  $\approx$  man - woman



Spain	Madrid
Italy	Rome
Germany	Berlin
Turkey	Ankara
Russia	Moscow
Canada	Ottawa
Japan	Tokyo
Vietnam	Honoi
China	Beiging

국가-수도

<https://ronxin.github.io/wevi/>

# GloVe (글로브)

- 미국 스탠포드대학에서 2014년 개발한 워드 임베딩 방법론
- Word2Vec 은 사용자가 지정한 윈도우(주변 단어 몇개) 내에서만 학습/분석이 이뤄지기 때문에 말뭉치 전체의 co-occurrence(통계적 동시 등장 확률) 가 반영되기 어려운 단점 보완
- Training data - glove.6B.100d.txt training data
  - Wikipedia data with 6 billion tokens and a 400,000 word vocabulary
- 100개의 차원을 가지는 임베딩 벡터로 제공

# Transfer Learning of GloVe

- GloVe 의 100 dimension version download

```
curl -o glove.6B.100d.txt https://storage.googleapis.com/laurencemoroney-blog.appspot.com/glove.6B.100d.txt
```

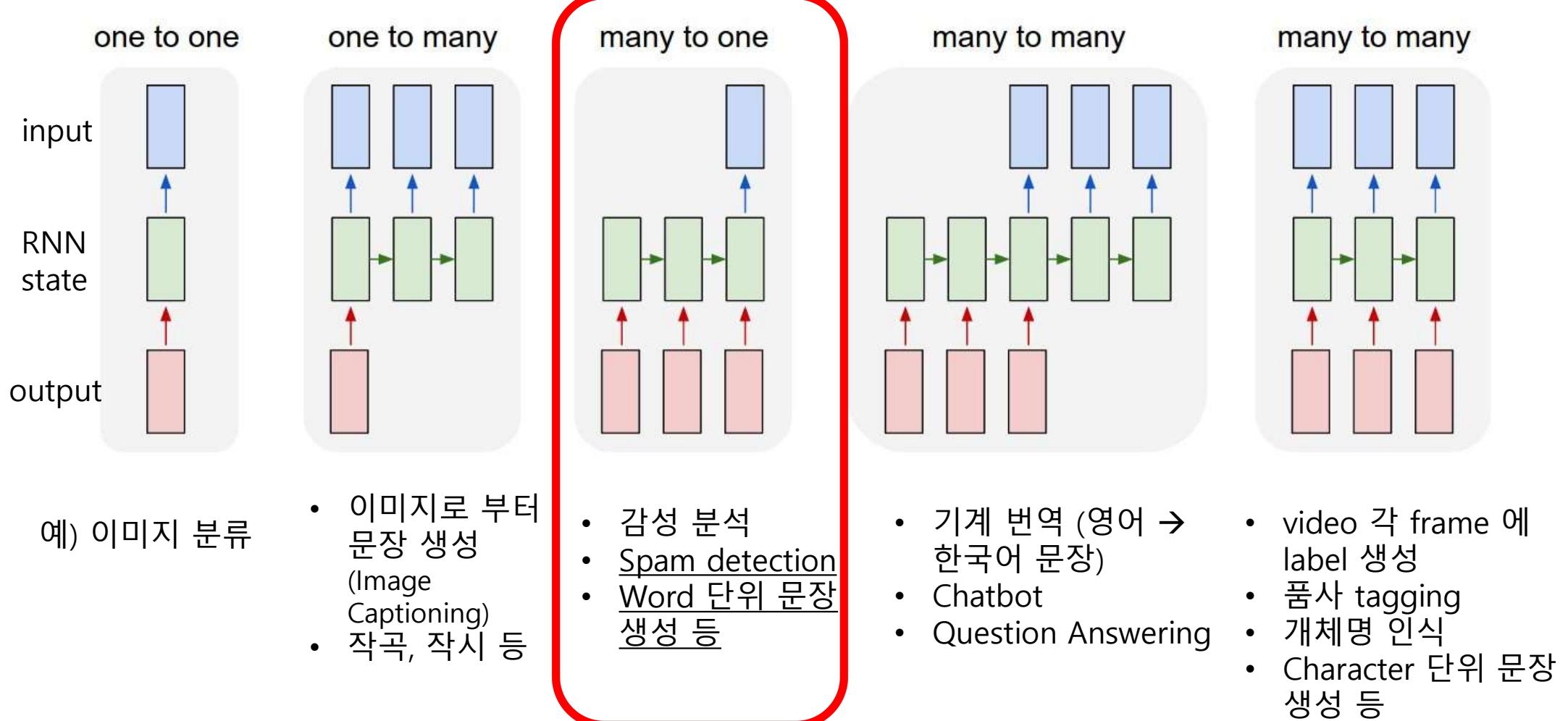
- Embedding layer 의 weights 에 glove.6B.100d weight load
- trainable=False 로 weight 고정

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size+1, embedding_dim, input_length=max_length,
                             weights=[embeddings_matrix], trainable=False),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Conv1D(64, 5, activation='relu'),
    tf.keras.layers.MaxPooling1D(pool_size=4),
    tf.keras.layers.LSTM(64),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

# 실습: keras word encoding

- keras 제공 pad\_sequences method
- sentence 의 token화

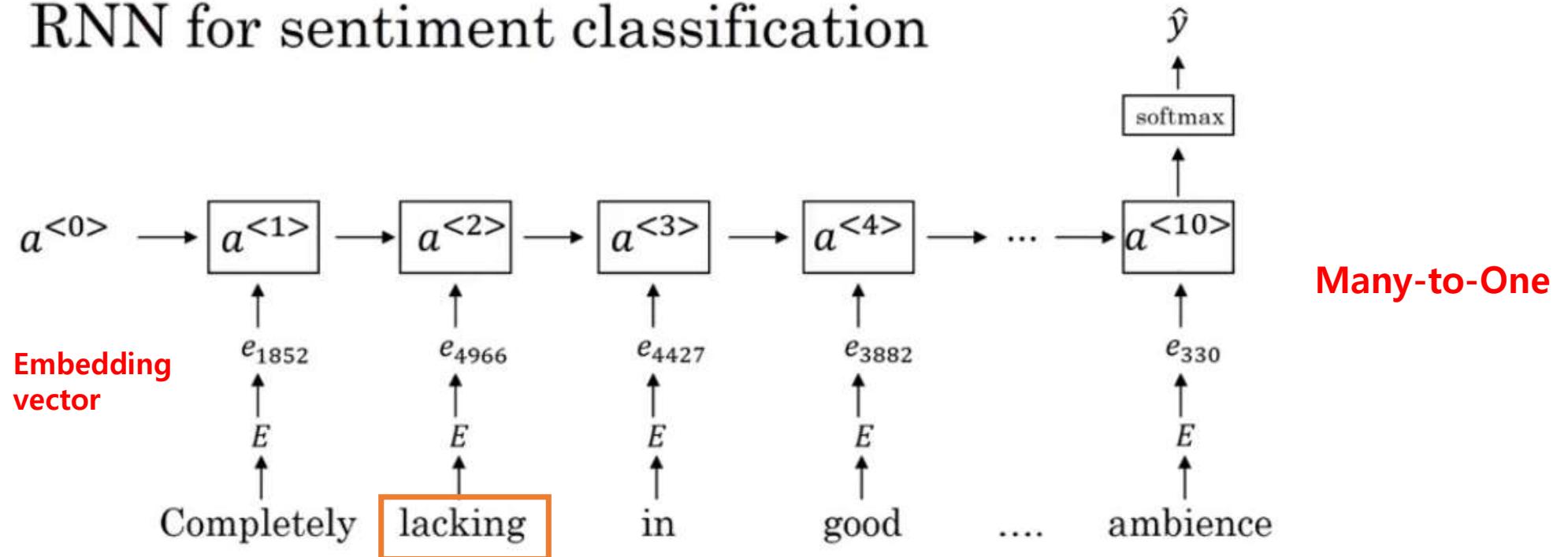
# RNN 을 이용한 Sentiment Analysis



# 감성분석 (Sentiment Analysis)

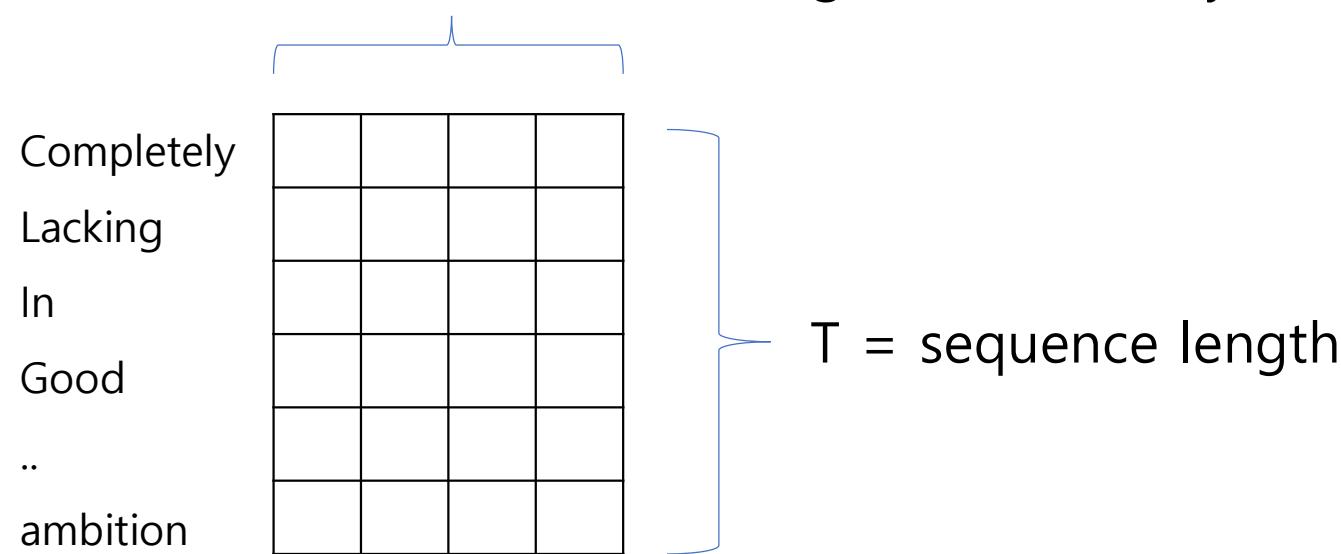
Traditional NLP : 단순히 단어의 출현 빈도를 count 하여 positive / negative 분류  
→ 단어의 순서 무시 (Bag of Words)

RNN for sentiment classification

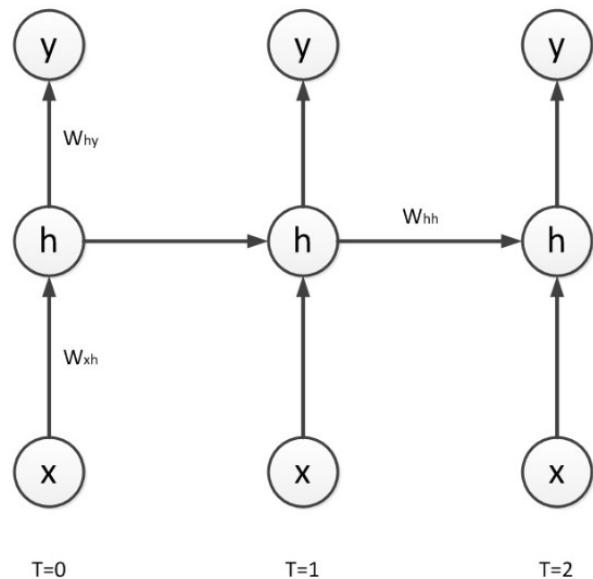


# RNN input ( $T \times D$ )

D = embedding dimensionality

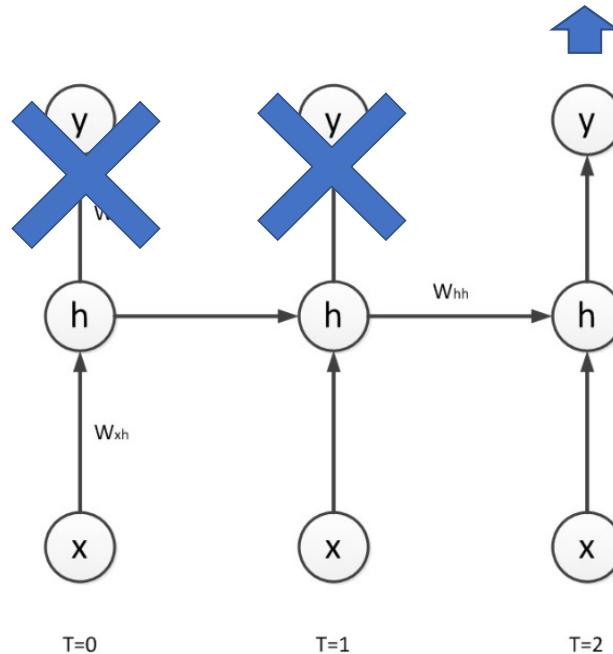


# RNN output



`return_sequences = True`

Sentiment analysis 는 last y 만 필요



`return_sequences = False`

# 실습 : 영화 관람평 분류 (sentiment 분석)

- Keras 의 built-in IMDB (Internet Movie Database) 이용
  - 각 25,000 개 training/testing 영화관람평
- Movie reviews sentiment classification
  - Label : positive, negative
- Preprocessing 되어 있고 모든 review 는 word indexes (integers) 로 표시  
(인덱스 순서는 빈번히 나타나는 단어 순서. ex. 3 : 3 번째로 빈번히 나타나는 단어)
- LSTM 을 이용한 Many-to-One type 의 RNN 으로 구현

# Tensorflow-datasets (tfds)

```
audio                      "fashion_mnist"          text
                            "horses_or_humans"
                            "image_label_folder"
                            "imagenet2012"
                            "imagenet2012_corrupted"
                            "kmnist"
                            "lsun"
                            "mnist"
                            "omniglot"
                            "open_images_v4"
                            "oxford_iit_pet"
                            "quickdraw_bitmap"
                            "rock_paper_scissors"
                            "shapes3d"
                            "smallnorb"
                            "sun397"
                            "svhn_cropped"
                            "tf_flowers"
structured
                            "higgs"
                            "iris"
                            "titanic"
"nsynth"
image
                            "abstract_reasoning"
                            "caltech101"
                            "cats_vs_dogs"
                            "celeb_a"
                            "celeb_a_hq"
                            "cifar10"
                            "cifar100"
                            "cifar10_corrupted"
                            "coco2014"
                            "colorectal_histology"
                            "cycle_gan"
                            "diabetic_retinopathy..."
                            "dsprites"
                            "dtd"
                            "emnist"
                            "mnist"
                            "omniglot"
                            "open_images_v4"
                            "oxford_iit_pet"
                            "quickdraw_bitmap"
                            "rock_paper_scissors"
                            "shapes3d"
                            "smallnorb"
                            "sun397"
                            "svhn_cropped"
                            "tf_flowers"
text
                            "cnn_dailymail"
                            "glue"
                            "imdb"
                            "multi_nli"
                            "squad"
                            "wikipedia"
                            "xnli"
translate
                            "flores"
                            "para_crawl"
                            "ted_hrlr_translate"
                            "ted_multi_translate"
                            "wmt15_translate"
                            "wmt16_translate"
                            "wmt17_translate"
                            "wmt18_translate"
                            "wmt19_translate"
```

# 실습 : 이상한 나라의 엘리스 문장 생성기

- LSTM, keras API 이용

- 161793 글자로 이루어진 text 를 '.' 과 ';' 단위로 statement 분리
  - Corpus = re.split('[.,]', text)

```
["project gutenberg's alice's adventures in wonderland",
 ' by lewis carroll this ebook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever',
 ' you may copy it',
 ' give it away or re-use it under the terms of the project gutenberg license included with this ebook or online at www',
 'gutenberg',
 "org title: alice's adventures in wonderland author: lewis carroll posting date: june 25",
```

- Keras API 를 이용하여 tokenize 및 pad\_sequence
- LSTM 이용하여 model 작성
  - Next word 예측

# 실습 : 이상한 나라의 엘리스 문장 생성기

- LSTM, keras API 이용

- 161793 글자로 이루어진 text 를 '.' 과 ';' 단위로 statement 분리
  - Corpus = re.split('[.,]', text)

```
["project gutenberg's alice's adventures in wonderland",
 ' by lewis carroll this ebook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever',
 ' you may copy it',
 ' give it away or re-use it under the terms of the project gutenberg license included with this ebook or online at www',
 'gutenberg',
 "org title: alice's adventures in wonderland author: lewis carroll posting date: june 25",
```

- Keras API 를 이용하여 tokenize 및 pad\_sequence
- LSTM 이용하여 model 작성
  - Next word 예측

# Sentence 생성 (이상한 나라의 Alice)

```
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1

# create input sequences using list of tokens
input_sequences = []
for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)

# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:, :-1], input_sequences[:, -1]

label = ku.to_categorical(label, num_classes=total_words)    # one-hot-encoding
```

Line:	Input Sequences:
[4 2 66 8 67 68 69 70]	[4 2]
	[4 2 66]
	[4 2 66 8]
	[4 2 66 8 67]
	[4 2 66 8 67 68]
	[4 2 66 8 67 68 69]
	[4 2 66 8 67 68 69 70]

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])
input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

# create predictors and label
predictors, label = input_sequences[:, :-1], input_sequences[:, -1]
```

[4 2 66 8 67 68 69 70]



Padded Input Sequences:

Input (X)	Label (Y)
[0 0 0 0 0 0 0 0 0 0 4]	2
[0 0 0 0 0 0 0 0 0 4 2 66]	
[0 0 0 0 0 0 0 0 4 2 66 8]	
[0 0 0 0 0 0 0 4 2 66 8 67]	
[0 0 0 0 0 0 4 2 66 8 67 68]	
[0 0 0 0 0 4 2 66 8 67 68 69]	
[0 0 0 0 4 2 66 8 67 68 69 70]	

## Sentence / Label 구성

Sentence: [ 0 0 0 0 4 2 66 8 67 68 69 70 ]

```
x:[0 0 0 0 4 2 66 8 67 68 69]
```

Label: [ 70 ]

# 실습 : 한글 어린왕자 문장 생성기

- 이상한 나라의 Alice 문장 생성기를 한글 어린왕자 문장 생성기로  
응용

# Keras Processing Tips

# Early Stopping

- Epoch 이 반복되어도 더 이상 성능 향상이 이루어지지 않을 경우 조기에 training 종료

# Callback

- Training 중에 keras 의 behavior 를 변경할 수 있는 customization 기능

# Callbacks

```
class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('loss')<0.4):
            print("\nLoss is low so cancelling training!")
            self.model.stop_training = True
```

```
callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images/255.0
test_images=test_images/255.0
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

# 실습 : 자동차 연비 예측 Regression

- UCI Machine Learning Data 이용
- 의도적으로 과적합을 만들고, early stopping 기능 test
- Early stopping 전, 후의 loss plot 하여 비교
- Checkpoint 추가 및 model save / reload

# GAN

# GAN (Generative Adversarial Network)

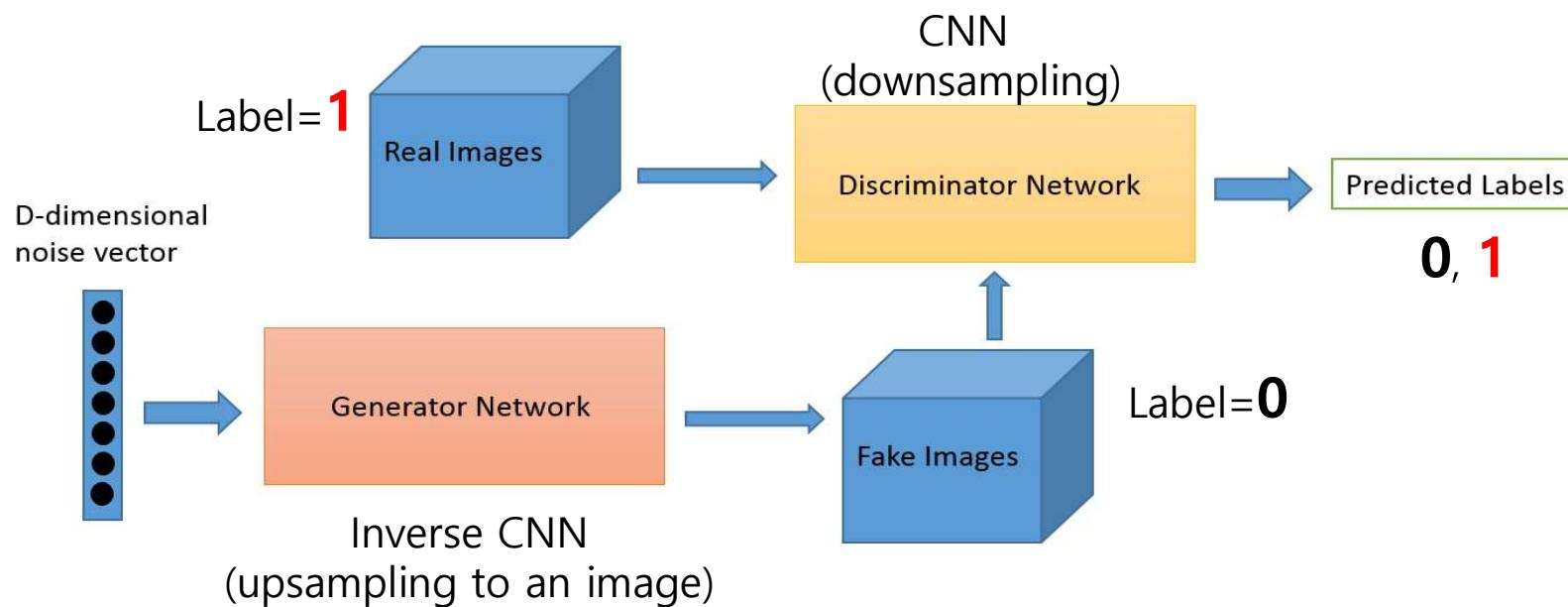
- “the most interesting idea in the last 10 years in ML” – Yann LeCun
- 2014 Ian Goodfellow 가 최초 제안
- WaveNet – 2016 Google DeepMind 가 제안한 GAN model
- Computer 가 이미지, 인간의 목소리, 악기소리, 소설, 기사 등을 실제와 같이 생성
- 위조를 담당하는 Generator(위조범) 와 위조를 판별하는 Discriminator (경관)의 두개 Deep Neural Network 으로 구성

# Generative vs. Discriminative Algorithms

- Discriminative Algorithm
  - input data 의 feature 를 기준으로 label 예측 (ex. Spam 분류)
  - $p(y | X)$  → “the probability of y given X”
- Generative Algorithm
  - 주어진 label 을 기준으로 feature 예측
    - (feature extraction (x) → feature filling(o))
  - $p(X | y)$  → “the probability of features given y”

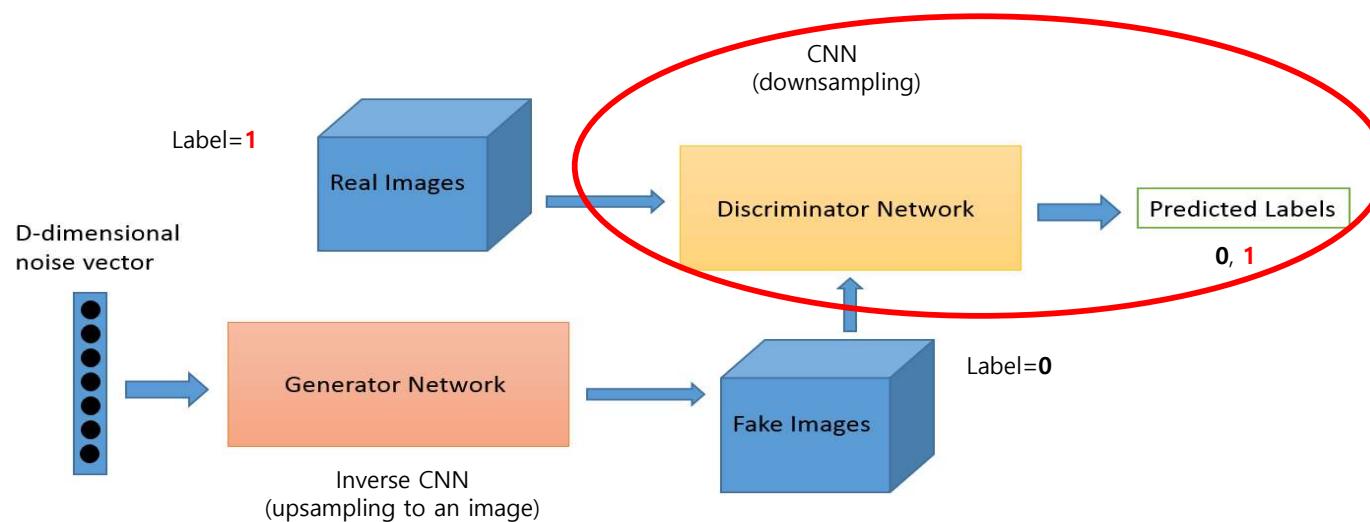
# GAN process

1. Generator 가 random number 를 취하여 random image 생성
2. 생성된 image 를 actual dataset 에서 받은 image 와 함께 discriminator 에게 공급
3. Discriminator 는 real 과 fake image 를 가지고 0~1 사이의 확률을 반환 (1 – real, 0 – fake)



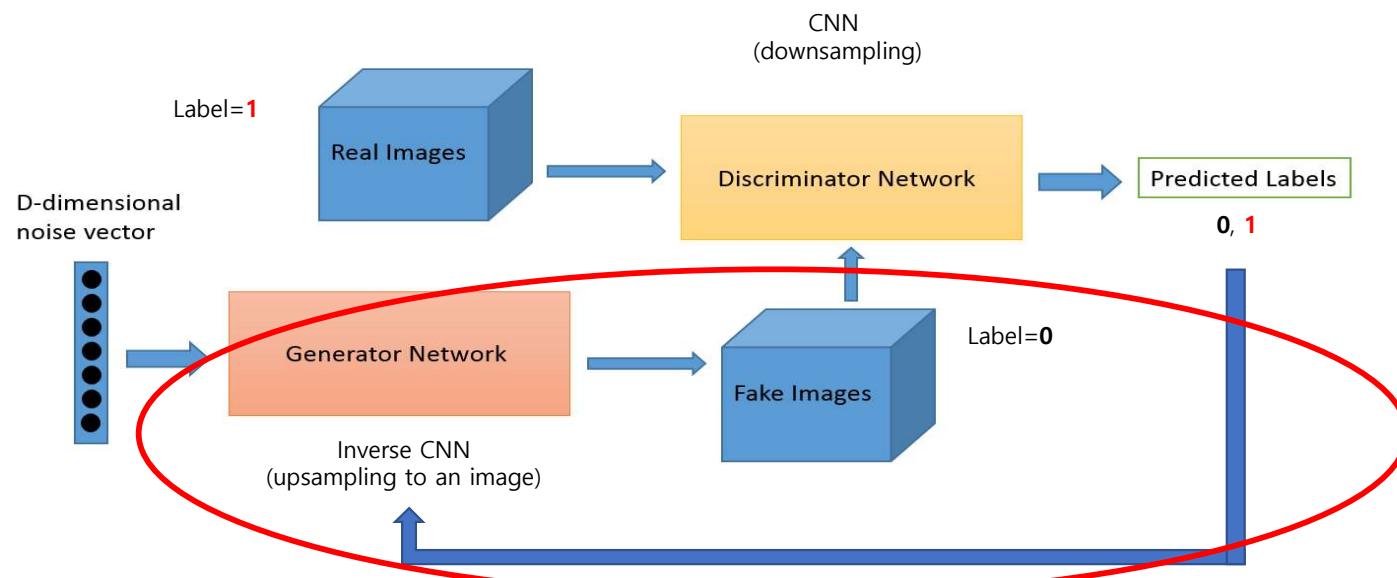
# Training of Discriminator

1. Generator 가 random number 를 취하여 random image 생성
2. 생성된 image 를 actual dataset 에서 받은 image 와 함께 discriminator 에게 공급
3. Discriminator 는 real 과 fake image 를 가지고 0~1 사이의 확률을 반환 (1 – real, 0 – fake)  
→ Binary classification problem 으로 discriminator train



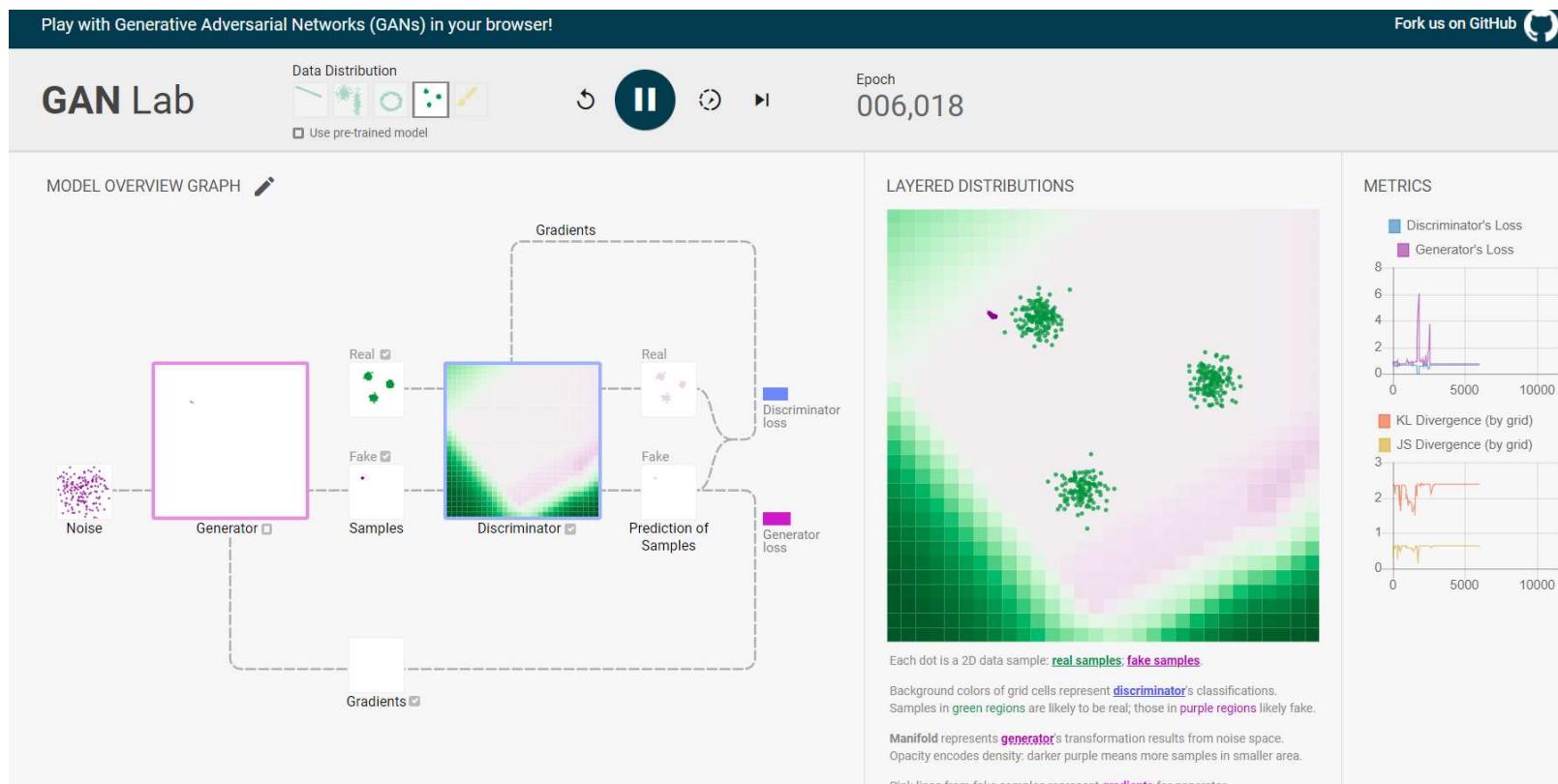
# Training of Generator

1. Generator 가 random number 를 취하여 random image (fake image) 생성
2. 생성된 fake image 를 discriminator 에게 공급하면 Discriminator 는 0~1 사이의 확률을 predict (1 – real, 0 – fake)
3. Discriminator 의 확률을 1 과 비교하여 차이분을 backpropagation 으로 보정

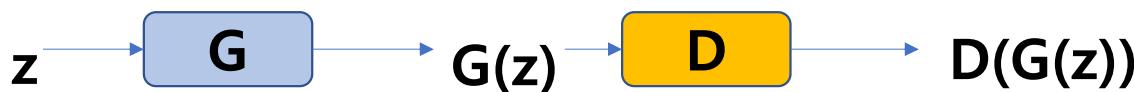
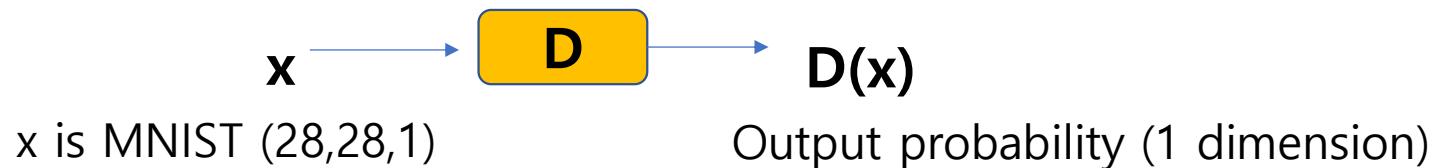


# GAN Lab

- <https://poloclub.github.io/ganlab/>



# GAN implementation



Discriminator

Input size : (28, 28, 1)

Output size: 1

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(64, (5, 5),
                      strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

model.add(layers.Conv2D(512, (5, 5), strides=(2, 2), padding='same'))
model.add(layers.LeakyReLU())
model.add(layers.Dropout(0.3))

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

# GAN implementation

```
model = tf.keras.Sequential()
model.add(layers.Dense(7*7*256, use_bias=False,
                      input_shape=(100,))) # seed 를 입력으로 받음, 출력 12544
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Reshape((7, 7, 256)))
assert model.output_shape == (None, 7, 7, 256)      # 배치사이즈로 None 추가

model.add(layers.Conv2DTranspose(128, (5, 5),
                               strides=(1, 1), padding='same', use_bias=False))
assert model.output_shape == (None, 7, 7, 128)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(64, (5, 5),
                               strides=(2, 2), padding='same', use_bias=False))
assert model.output_shape == (None, 14, 14, 64)
model.add(layers.BatchNormalization())
model.add(layers.LeakyReLU())

model.add(layers.Conv2DTranspose(1, (5, 5),
                               strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
assert model.output_shape == (None, 28, 28, 1)
```

Generator

Input size : 100

Output size: (28, 28, 1)

# GAN object function

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

The diagram illustrates the GAN objective function with the following annotations:

- $\mathbb{E}_{x \sim p_{data}(x)}$  [Expectation]: An arrow points from this term to the text "x 는 real data 로 부터 표본추출".
- $D(x)$  [D(real) 확률]: An arrow points from this term to the text "x 는 real data 로 부터 표본추출".
- $\mathbb{E}_{z \sim p_z(z)}$  [Expectation]: An arrow points from this term to the text "z 는 N(0,1)로 부터 표본추출".
- $D(G(z))$  [D(fake) 확률]: An arrow points from this term to the text "z 는 N(0,1)로 부터 표본추출".
- $1 - D(G(z))$ : An arrow points from this term to the word "fake".

- $D(x)$  classifier (Discriminator) 는 real 이면 1, fake 면 0 을 return 하도록 훈련.  
따라서,  $D(G(z))$  은 G (Generator) 가 만들어낸  $G(z)$  가 real 이라고 판단되면 1, fake 라고 판단되면 0 return
- $\max_D V(D, G)$  가 되려면  $D(x)$  와  $1-D(G(z))$  이 모두 1 이 되어야 한다.
- $\min_G V(D, G)$  가 되려면  $1-D(G(z))$  이 0 가 되어야 한다. ( $D(x)$  는 G 와 무관하므로 무시)

# Loss function (손실함수) 와 Optimizer 정의

- Discriminator

```
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
real_loss = cross_entropy(tf.ones_like(real_output), real_output)
fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
total_loss = real_loss + fake_loss
```

- Generator

```
cross_entropy(tf.ones_like(fake_output), fake_output)
```

- Optimizer

```
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

# GAN special layers

- Generator 는 tf.keras.layers.Conv2DTranspose 를 사용하여 upsampling
- Discriminator 는 Conv2D 를 이용한 전형적인 CNN 이고 output layer 에 sigmoid activation 사용

# 실습 : CNN 을 이용한 mnist dataset 위조

- Discriminator 의 Goal
  - mnist dataset 을 “진짜” 로 인식하고, Generator 에서 공급되는 image 를 fake 로 구분
- Generator 의 Goal
  - discriminator 가 “진짜” 로 인식할 fake image 생성  
(Gaussian random noise 로 부터 image 생성)
- GAN 의 training 은 시간이 오래 걸린다 → GPU 필요

# 1. Numpy and Linear Algebra (선형대수)

- 수치 data 를 다루는데 효율적이고 높은 성능 제공
- 각종 수학적 함수 제공
- Python scientific library 들이 Numpy 기반으로 구축

## ndarray

- n-dimensional array (다차원 배열 객체)로 구성

In [1]:

```
1 import sys
2 import numpy as np
3 print("python {}".format(sys.version))
4 print("numpy {}".format(np.__version__))
5 sys.version
6 np.__version__
```

python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]  
numpy 1.18.1

Out[1]:

'1.18.1'

In [2]:

```
1 # 스칼라
2 x = 6
3 x
```

Out[2]:

6

In [3]:

```
1 # 벡터
2 x = np.array([1,2,3])
3 x
```

Out[3]:

array([1, 2, 3])

In [4]:

```
1 print("vector dimension {}".format(x.shape))
2 print("vector size {}".format(x.size))
```

vector dimension (3,)
vector size 3

In [5]:

```
1 # matrix
2 y = np.array([[1,2,3], [4,5,6], [7,8,9]])
3 print(y)
```

```
[[1 2 3]
[4 5 6]
[7 8 9]]
```

In [6]:

```
1 print("array dimension {}".format(y.shape))
2 print("array size {}".format(y.size))
```

```
array dimension (3, 3)
array size 9
```

In [7]:

```
1 # 주어진 dimension 의 matrix 정의
2 x = np.ones((2,3))
3 print(x)
```

```
[[1. 1. 1.]
[1. 1. 1.]]
```

In [8]:

```
1 print("array dimension {}".format(x.shape))
2 print("array size {}".format(x.size))
```

```
array dimension (2, 3)
array size 6
```

In [9]:

```
1 x = np.ones((3,3,3))
2 print(x)
```

```
[[[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]]]
```

```
[[[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]]]
```

```
[[[1. 1. 1.]
[1. 1. 1.]
[1. 1. 1.]]]]
```

In [10]:

```
1 print("array dimension {}".format(x.shape))
2 print("array size {}".format(x.size))
3 print("array length {}".format(len(x)))
```

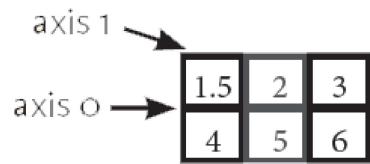
array dimension (3, 3, 3)  
array size 27  
array length 3

## Indexing / Slicing

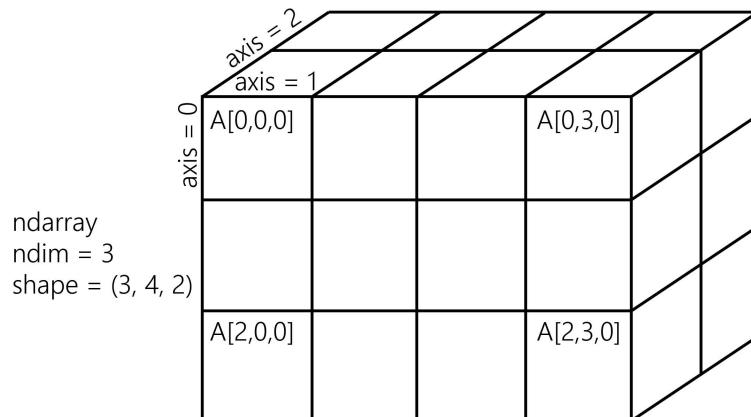
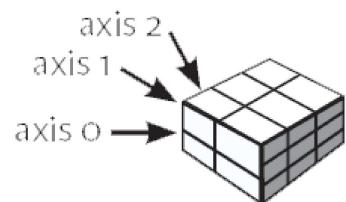
1D array



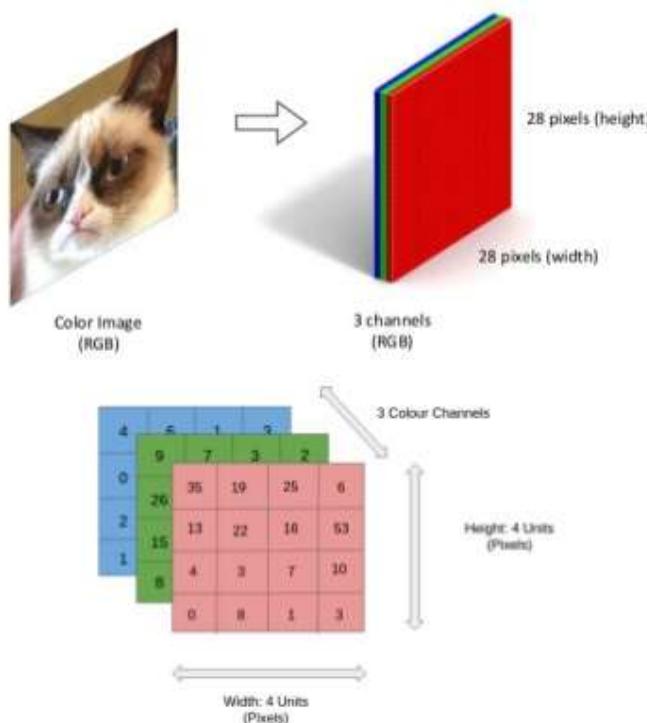
2D array



3D array



# color image is 3rd-order tensor



In [11]:

```
1 # indexing matrix
2 A = np.ones((5,5), dtype=np.int)
3 print(A)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

In [12]:

```
1 A[0]
```

Out[12]:

```
array([1, 1, 1, 1, 1])
```

In [13]:

```
1 A[0,1] = 2
2 print(A)
```

```
[[1 2 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]]
```

In [14]:

```
1 A[:, 0] = 3
2 print(A)
```

```
[[3 2 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]
 [3 1 1 1 1]]
```

In [15]:

```
1 A[:, :] = 5
2 print(A)
```

```
[[5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]
 [5 5 5 5 5]]
```

In [16]:

```
1 A = np.zeros((3,3,3), dtype=np.int)
2 A[:, :, 0] = 3
3 print(A)
```

```
[[[3 0 0]
 [3 0 0]
 [3 0 0]]]
```

```
[[[3 0 0]
 [3 0 0]
 [3 0 0]]]
```

```
[[[3 0 0]
 [3 0 0]
 [3 0 0]]]
```

In [17]:

```
1 A = np.zeros((3,3,3), dtype=np.int)
2 A[0, :, :] = 3
3 A[2,2,2] = 9
4 print(A)
```

```
[[[3 3 3]
 [3 3 3]
 [3 3 3]]]
```

```
[[[0 0 0]
 [0 0 0]
 [0 0 0]]]
```

```
[[[0 0 0]
 [0 0 0]
 [0 0 9]]]
```

In [18]:

```
1 A = np.zeros((3,3,3), dtype=np.int)
2 A[:, 0, :] = 3
3 print(A)
```

[[[3 3 3]  
[0 0 0]  
[0 0 0]]]

[[3 3 3]  
[0 0 0]  
[0 0 0]]]

[[3 3 3]  
[0 0 0]  
[0 0 0]]]

In [19]:

```
1 A = np.zeros((3,3,3), dtype=np.int)
2 A[:, 0, 0] = 3
3 print(A)
```

[[[3 0 0]  
[0 0 0]  
[0 0 0]]]

[[3 0 0]  
[0 0 0]  
[0 0 0]]]

[[3 0 0]  
[0 0 0]  
[0 0 0]]]

In [20]:

```
1 A[-1, -1, -1] = -1
2 A
```

Out[20]:

```
array([[[ 3,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]],

      [[ 3,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]],

      [[ 3,  0,  0],
       [ 0,  0,  0],
       [ 0,  0, -1]]])
```

In [21]:

```
1 A[-2, -2, -2] = -2
2 A
```

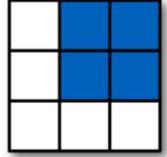
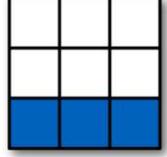
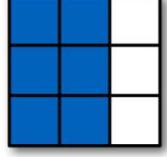
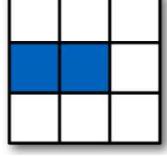
Out[21]:

```
array([[[ 3,  0,  0],
       [ 0,  0,  0],
       [ 0,  0,  0]],

      [[ 3,  0,  0],
       [ 0, -2,  0],
       [ 0,  0,  0]],

      [[ 3,  0,  0],
       [ 0,  0,  0],
       [ 0,  0, -1]]])
```

## Slicing

	Expression	Shape
	arr[:2, 1:]	(2, 2)
	arr[2] arr[2, :] arr[2:, :]	(3,) (3,) (1, 3)
	arr[:, :2]	(3, 2)
	arr[1, :2] arr[1:2, :2]	(2,) (1, 2)

In [22]:

```
1 arr = np.array([[1,2,3], [4,5,6], [7,8,9]])
2 arr
```

Out[22]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [23]:

```
1 arr[:2, 1:]
```

Out[23]:

```
array([[2, 3],  
       [5, 6]])
```

In [24]:

```
1 arr[2]
```

Out[24]:

```
array([7, 8, 9])
```

In [25]:

```
1 arr[2:, :]
```

Out[25]:

```
array([7, 8, 9])
```

In [26]:

```
1 arr[2:,:, :]
```

Out[26]:

```
array([[7, 8, 9]])
```

In [27]:

```
1 arr[:, :2]
```

Out[27]:

```
array([[1, 2],  
       [4, 5],  
       [7, 8]])
```

In [28]:

```
1 arr[1, :2]
```

Out[28]:

```
array([4, 5])
```

In [29]:

```
1 arr[1:2, :2]
```

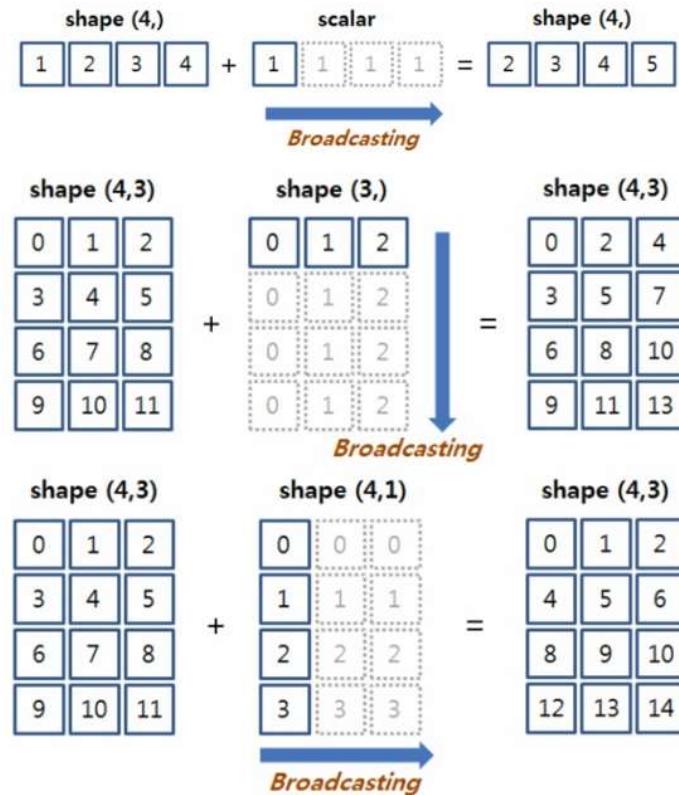
Out[29]:

```
array([[4, 5]])
```

## matrix 간의 연산 (Element-wise)

### broadcasting

- 서로 dimension 이 다른 두개의 array 를 연산할 때 더 큰 쪽의 axes 에 작은 쪽 array 를 반복시켜 match 시키는 것



In [30]:

```
1 a = np.array([1, 2, 3, 4])
2 b = 1
3 print(a + b)
```

[2 3 4 5]

In [31]:

```
1 a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
2 b = np.array([0, 1, 2])
3 print(a + b)
```

$\begin{bmatrix} 0 & 2 & 4 \\ 3 & 5 & 7 \\ 6 & 8 & 10 \\ 9 & 11 & 13 \end{bmatrix}$

In [32]:

```
1 a = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 10, 11]])
2 b = np.array([[0], [1], [2], [3]])
3
4 print(np.add(a, b))
```

```
[[ 0  1  2]
 [ 4  5  6]
 [ 8  9 10]
 [12 13 14]]
```

In [33]:

```
1 print(a * b)
```

```
[[ 0  0  0]
 [ 3  4  5]
 [12 14 16]
 [27 30 33]]
```

In [34]:

```
1 np.multiply(a, b)
```

Out[34]:

```
array([[ 0,  0,  0],
       [ 3,  4,  5],
       [12, 14, 16],
       [27, 30, 33]])
```

In [35]:

```
1 print(a - b)
2 print()
3 print(np.subtract(a, b))
```

```
[[0 1 2]
 [2 3 4]
 [4 5 6]
 [6 7 8]]
```

```
[[0 1 2]
 [2 3 4]
 [4 5 6]
 [6 7 8]]
```

In [36]:

```
1 c = np.array([[1, 2], [1, 2]])
2 print(c)
```

```
[[1 2]
 [1 2]]
```

In [37]:

```
1 print(a - c)
```

```
-----  
ValueError                                Traceback (most recent call last)  
<ipython-input-37-0afb6935dcf5> in <module>  
----> 1 print(a - c)
```

ValueError: operands could not be broadcast together with shapes (4,3) (2,2)

벡터는 두 가지 방법으로 곱할 수 있다. 가장 일반적인 방법은 내적(inner product)을 구하는 것이다. 내적이 되려면 두 벡터의 dimension이 같아야 한다. 내적은 각 element의 곱을 모두 더한 것이다.

inner product 은 dot product (점곱)이라고도 불린다.

In [38]:

```
1 a = np.array([2,5,1]).reshape(-1,1)  
2 a
```

Out[38]:

```
array([[2],  
       [5],  
       [1]])
```

In [39]:

```
1 b = np.array([4,3,5]).reshape(-1,1)  
2 b
```

Out[39]:

```
array([[4],  
       [3],  
       [5]])
```

$$\mathbf{a}^T \mathbf{b} = \begin{bmatrix} 2 & 5 & 1 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \\ 5 \end{bmatrix} = [2 * 4 + 5 * 3 + 1 * 5] = 28$$

In [40]:

```
1 # a, b 의 내적 ==> scalar  
2 np.dot(a.T, b)
```

Out[40]:

```
array([[28]])
```

## norm (노름, 벡터의 길이)

inner product 은 vector 의 길이를 구할 때에도 사용된다. 벡터  $a$  의 길이는  $\|a\|$  로 나타내며,

$$\|a\| = \sqrt{a^T a}$$

In [41]:

```
1 np.sqrt(np.dot(a.T, a))
```

Out[41]:

```
array([[5.47722558]])
```

In [42]:

```
1 np.linalg.norm(a)
```

Out[42]:

```
5.477225575051661
```

## matrix 곱셈 (행렬 곱셈, dot product)

두 행렬 A 와 B 는 A 의 열(column) 갯수가 B 의 행(row) 갯수와 같을 때 곱할 수 있다.  
결과 행렬 C 의 shape 은 A 의 row x B 의 column 이 된다.

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} e & f \\ g & h \end{bmatrix} = \begin{bmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 2 & 5 & 2 \\ 1 & 6 & 8 \end{bmatrix}$$

$2 \times 2$        $2 \times 3$        $2 \times 3$

$$A = \begin{bmatrix} 2 & 5 & 1 \\ 4 & 5 & 3 \end{bmatrix} \text{ and } B = \begin{bmatrix} 4 & 3 & 5 & 7 \\ 9 & 5 & 3 & 4 \\ 5 & 3 & 6 & 7 \end{bmatrix}$$

$$AB = \begin{bmatrix} 2*4 + 5*9 + 1*5 & 2*3 + 5*5 + 1*3 & 2*5 + 5*3 + 1*6 & 2*7 + 5*4 + 1*7 \\ 4*4 + 5*9 + 3*5 & 4*3 + 5*5 + 3*3 & 4*5 + 5*3 + 3*6 & 4*7 + 5*4 + 3*7 \end{bmatrix}$$

In [43]:

```
1 a = np.array([[2, 1], [1, 4]])
2 b = np.array([[1, 2, 0], [0, 1, 2]])
3 print(a.shape)
4 print(b.shape)
5 print(np.matmul(a, b))
```

```
(2, 2)
(2, 3)
[[2 5 2]
 [1 6 8]]
```

In [44]:

```
1 print(a@b)
```

```
[[2 5 2]
 [1 6 8]]
```

In [45]:

```
1 print(a.dot(b))
```

```
[[2 5 2]
 [1 6 8]]
```

## 전치행렬 (Transposed Matrix)

In [46]:

```
1 # matrix transpose
2 A = np.arange(9).reshape((3,3))
3 print(A)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

In [47]:

```
1 print(A.T)
```

```
[[0 3 6]
 [1 4 7]
 [2 5 8]]
```

In [48]:

```
1 A = np.array(range(10)).reshape((2,5))
2 print(A)
```

```
[[0 1 2 3 4]
 [5 6 7 8 9]]
```

In [49]:

```
1 B = A.T
2 print(B)
```

```
[[0 5]
 [1 6]
 [2 7]
 [3 8]
 [4 9]]
```

In [50]:

```
1 print(A.shape)
2 print(B.shape)
```

(2, 5)  
(5, 2)

## 다차원 배열(array)을 1차원 배열로 변환

In [55]:

```
1 print(A.ravel())
```

[1 2 3 2 3 5 4 5 6]

In [56]:

```
1 # tensors
2 A = np.ones((3,3,3,3,3,3,3,3,3))
3 print(A.shape)
4 print(len(A))
5 print(len(A.shape))
6 print(A.size)
```

(3, 3, 3, 3, 3, 3, 3, 3, 3)  
3  
10  
59049

## 2. Pandas Crash - Basic Pandas

Pandas 는 Series data type 과 DataFrame data type 으로 구성된다.

Series (1 차원) : numpy array 와 유사.

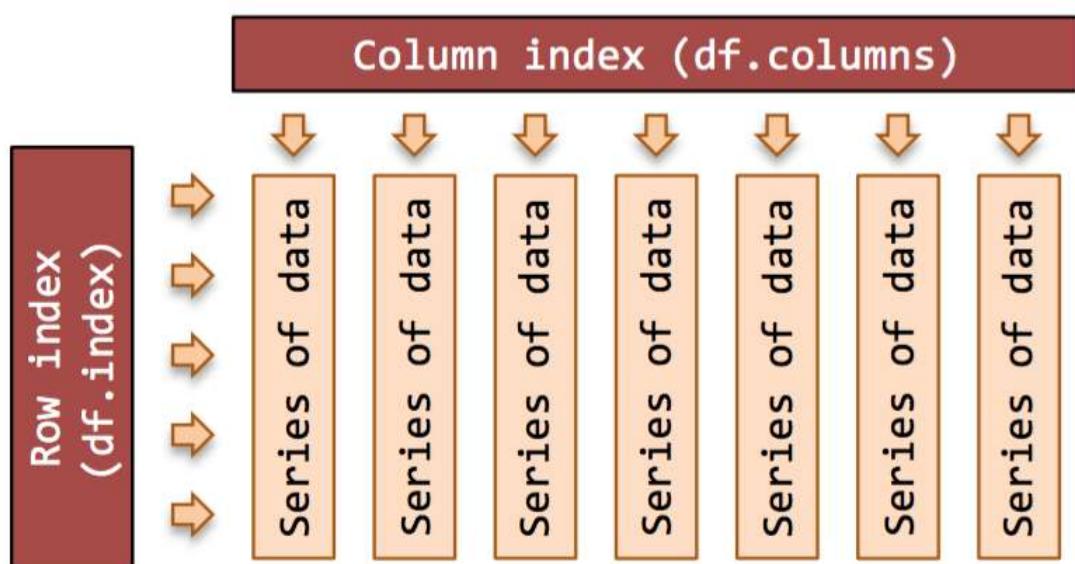
- 차이점 - numpy 와 달리 Series 는 axis (행, 열)에 label 을 부여할 수 있다. 즉, numpy 와 같이 숫자로만 indexing 하는 것이 아니라 label 명으로 indexing 을 할 수 있다. 또한 숫자 뿐 아니라 임의의 Python object 를 모두 element 로 가질 수 있다.

DataFrame (2차원, table)

- Python program 안의 Excel

Series vs DataFrame

Series		Series		DataFrame	
apples		oranges		apples	oranges
0	3	0	0	0	0
1	2	1	3	1	3
2	0	2	7	2	7
3	1	3	2	3	2



In [1]:

```
1 import numpy as np
2 import pandas as pd
```

## Series 생성

- list, numpy array, dictionary 를 모두 Series 로 변환할 수 있다.
- dictionary 의 경우 key 가 label, value 가 value 로 번환된다.

In [2]:

```
1 labels = ['a', 'b', 'c', 'd']
2 my_list = [10, 20, 30, 40]
3 arr = np.array([10, 20, 30, 40])
4 dict = {'a': 10, 'b': 20, 'c': 30, 'd': 40}
```

In [3]:

```
1 series1 = pd.Series(dict)    # dictionary
2 series1
```

Out[3]:

```
a    10
b    20
c    30
d    40
dtype: int64
```

In [4]:

```
1 series2 = pd.Series(data=my_list, index=labels)    # list
2 series2
```

Out[4]:

```
a    10
b    20
c    30
d    40
dtype: int64
```

In [5]:

```
1 series3 = pd.Series(arr)    # numpy array
2 series3
```

Out[5]:

```
0    10
1    20
2    30
3    40
dtype: int32
```

In [6]:

```
1 series4 = pd.Series(data=labels) # list
2 series4
```

Out[6]:

```
0    a
1    b
2    c
3    d
dtype: object
```

## Series 의 indexing

In [7]:

```
1 series1['b':'c']
```

Out[7]:

```
b    20
c    30
dtype: int64
```

In [8]:

```
1 series3[1:3]
```

Out[8]:

```
1    20
2    30
dtype: int32
```

## Series 간의 연산

In [9]:

```
1 series1 + series2
```

Out[9]:

```
a    20
b    40
c    60
d    80
dtype: int64
```

## DataFrame

DataFrame 은 여러개의 Series 를 같은 index 기준으로 모아 Table 을 만든 것이다.

In [10]:

```
1 import pandas as pd
2 import numpy as np
```

In [11]:

```
1 np.random.seed(101)
2 data = np.random.randn(5, 4)
```

In [12]:

```
1 df = pd.DataFrame(data, index=['A', 'B', 'C', 'D', 'E'], columns=['W', 'X', 'Y', 'Z'])
2 df
```

Out[12]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

## dataframe 의 column 명

In [13]:

```
1 df.columns
```

Out[13]:

```
Index(['W', 'X', 'Y', 'Z'], dtype='object')
```

## column 별 unique value 의 갯수

In [14]:

```
1 df.nunique()
```

Out[14]:

```
W    5
X    5
Y    5
Z    5
dtype: int64
```

In [15]:

```
1 df['W'].value_counts()
```

Out[15]:

```
0.651118    1  
0.190794    1  
2.706850    1  
0.188695    1  
-2.018168   1  
Name: W, dtype: int64
```

In [16]:

```
1 df.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
Index: 5 entries, A to E  
Data columns (total 4 columns):  
 #  Column  Non-Null Count Dtype  
---  
 0  W        5 non-null    float64  
 1  X        5 non-null    float64  
 2  Y        5 non-null    float64  
 3  Z        5 non-null    float64  
dtypes: float64(4)  
memory usage: 200.0+ bytes
```

In [17]:

```
1 df.describe()
```

Out[17]:

	W	X	Y	Z
<b>count</b>	5.000000	5.000000	5.000000	5.000000
<b>mean</b>	0.343858	0.453764	0.452287	0.431871
<b>std</b>	1.681131	1.061385	1.454516	0.594708
<b>min</b>	-2.018168	-0.758872	-0.933237	-0.589001
<b>25%</b>	0.188695	-0.319318	-0.848077	0.503826
<b>50%</b>	0.190794	0.628133	0.528813	0.605965
<b>75%</b>	0.651118	0.740122	0.907969	0.683509
<b>max</b>	2.706850	1.978757	2.605967	0.955057

## DataFrame indexing

In [18]:

```
1 df['W']
```

Out[18]:

```
A 2.706850  
B 0.651118  
C -2.018168  
D 0.188695  
E 0.190794  
Name: W, dtype: float64
```

In [19]:

```
1 type(df['W'])
```

Out[19]:

```
pandas.core.series.Series
```

In [20]:

```
1 df[['W', 'Z']]
```

Out[20]:

	W	Z
A	2.706850	0.503826
B	0.651118	0.605965
C	-2.018168	-0.589001
D	0.188695	0.955057
E	0.190794	0.683509

In [21]:

```
1 df.W
```

Out[21]:

```
A 2.706850  
B 0.651118  
C -2.018168  
D 0.188695  
E 0.190794  
Name: W, dtype: float64
```

**new column 추가/삭제**

In [22]:

```
1 df['new'] = df['W'] + df['X']
2 df
```

Out[22]:

	W	X	Y	Z	new
<b>A</b>	2.706850	0.628133	0.907969	0.503826	3.334983
<b>B</b>	0.651118	-0.319318	-0.848077	0.605965	0.331800
<b>C</b>	-2.018168	0.740122	0.528813	-0.589001	-1.278046
<b>D</b>	0.188695	-0.758872	-0.933237	0.955057	-0.570177
<b>E</b>	0.190794	1.978757	2.605967	0.683509	2.169552

In [23]:

```
1 df.drop('new', axis=1)
```

Out[23]:

	W	X	Y	Z
<b>A</b>	2.706850	0.628133	0.907969	0.503826
<b>B</b>	0.651118	-0.319318	-0.848077	0.605965
<b>C</b>	-2.018168	0.740122	0.528813	-0.589001
<b>D</b>	0.188695	-0.758872	-0.933237	0.955057
<b>E</b>	0.190794	1.978757	2.605967	0.683509

In [24]:

```
1 df
```

Out[24]:

	W	X	Y	Z	new
<b>A</b>	2.706850	0.628133	0.907969	0.503826	3.334983
<b>B</b>	0.651118	-0.319318	-0.848077	0.605965	0.331800
<b>C</b>	-2.018168	0.740122	0.528813	-0.589001	-1.278046
<b>D</b>	0.188695	-0.758872	-0.933237	0.955057	-0.570177
<b>E</b>	0.190794	1.978757	2.605967	0.683509	2.169552

In [25]:

```
1 df.drop('new', axis=1, inplace=True)
2 df
```

Out[25]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

In [26]:

```
1 df.drop('D', axis=0, inplace=True)
2 df
```

Out[26]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
E	0.190794	1.978757	2.605967	0.683509

In [27]:

```
1 df.shape
```

Out[27]:

(4, 4)

## Missing Data 처리

- missing data 가 있는 row 혹은 columns 을 완전히 삭제 : dropna()
- 임의의 data 로 대체 :fillna()

In [28]:

```
1 import pandas as pd
2 import numpy as np
```

In [29]:

```
1 df = pd.DataFrame({'A': [1, 2, np.nan],  
2                      'B': [5, np.nan, np.nan],  
3                      'C': [1, 2, 3]})
```

In [30]:

```
1 df
```

Out[30]:

	A	B	C
0	1.0	5.0	1
1	2.0	NaN	2
2	NaN	NaN	3

**missing value 를 포함하고 있는 모든 row 삭제 - default**

In [31]:

```
1 df.dropna()
```

Out[31]:

	A	B	C
0	1.0	5.0	1

**missing value 를 포함하고 있는 모든 column 삭제**

In [32]:

```
1 df.dropna(axis=1)
```

Out[32]:

	C
0	1
1	2
2	3

**missing value 대체**

In [33]:

```
1 df.fillna(value=0)
```

Out[33]:

	A	B	C
0	1.0	5.0	1
1	2.0	0.0	2
2	0.0	0.0	3

In [34]:

```
1 df['A'].fillna(value=df['A'].mean())
```

Out[34]:

0 1.0  
1 2.0  
2 1.5  
Name: A, dtype: float64

## excel file handling

In [35]:

```
1 df = pd.read_csv("winequality-red.csv", sep=";")  
2 df.head()
```

Out[35]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	1
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	1
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	1
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	1
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	1

◀ ▶

### 3. Matplotlib Crash - Basic Matplotlib

#### Basic Plotting with matplotlib

matplotlib 은 두가지 style 로 사용 가능

1. Functional Programming Style

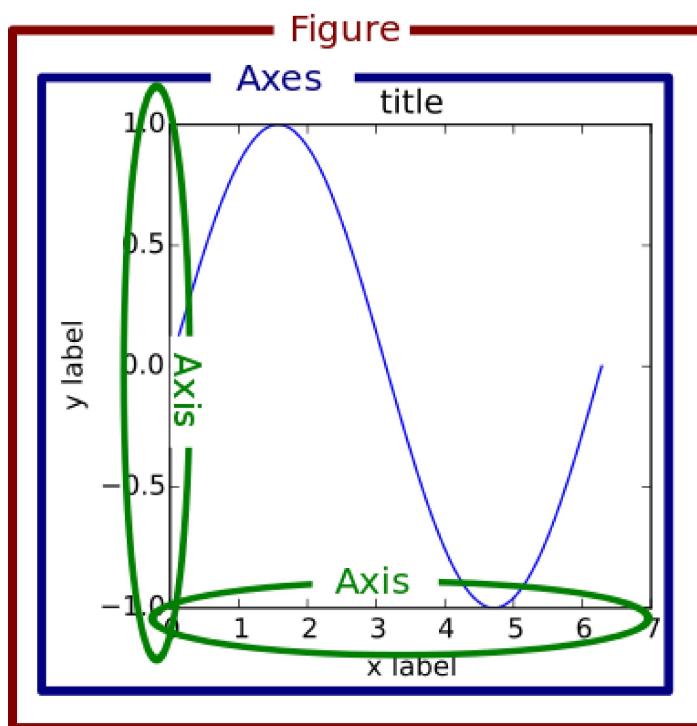
```
import matplotlib.pyplot as plt  
plt.plot(x, y)
```

2. OOP style

```
fig, ax = plt.subplots()  
ax.plot(x, y)
```

matlab style (pylab) 은 더이상 사용하지 않음

pyplot 의 object 구성



Functional Programming Style

In [1]:

```
1 import numpy as np
2 from matplotlib import pyplot as plt
3 %matplotlib inline
4 #한글 폰트 사용
5 from matplotlib import font_manager
6 import matplotlib
7 font_path = "C:/Windows/Fonts/H2GTRM.TTF"          #폰트 경로
8 font_name = font_manager.FontProperties(fname=font_path).get_name() #폰트 이름 알아오기
9 matplotlib.rc('font', family=font_name)             #font 지정
10 matplotlib.rcParams['axes.unicode_minus'] = False    #한글사용시 마이너스 사인 깨짐 방지
```

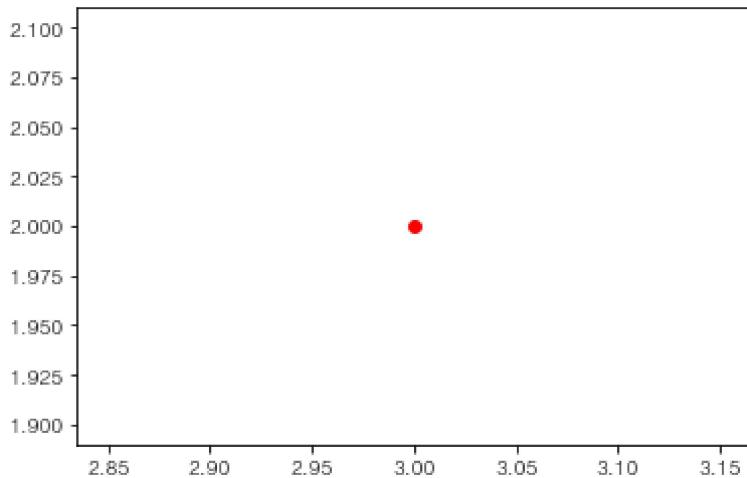
## 점찍기

In [2]:

```
1 plt.plot(3, 2, 'o', c='r')
```

Out[2]:

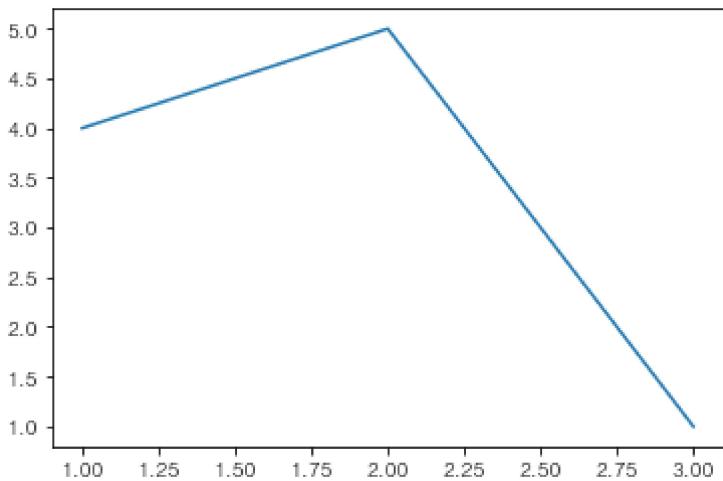
[<matplotlib.lines.Line2D at 0x1b3512c19c8>]



## 선긋기

In [3]:

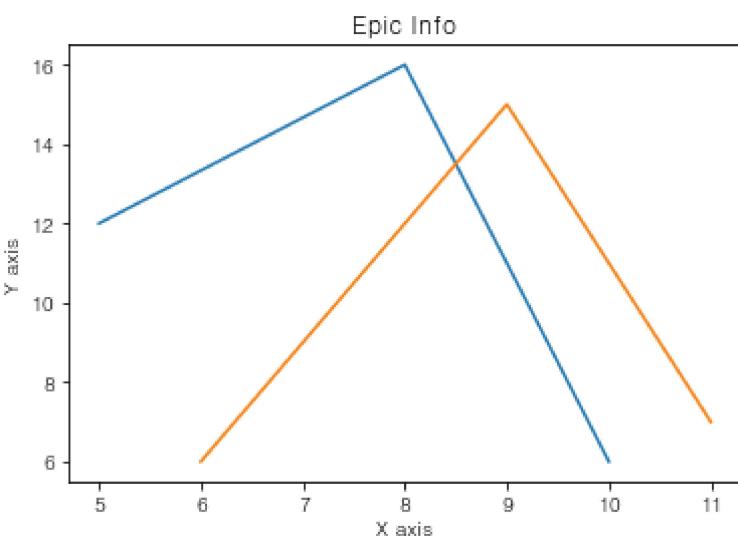
```
1 plt.plot([1,2,3],[4,5,1])
2
3 plt.show()
```



## title, label 추가

In [4]:

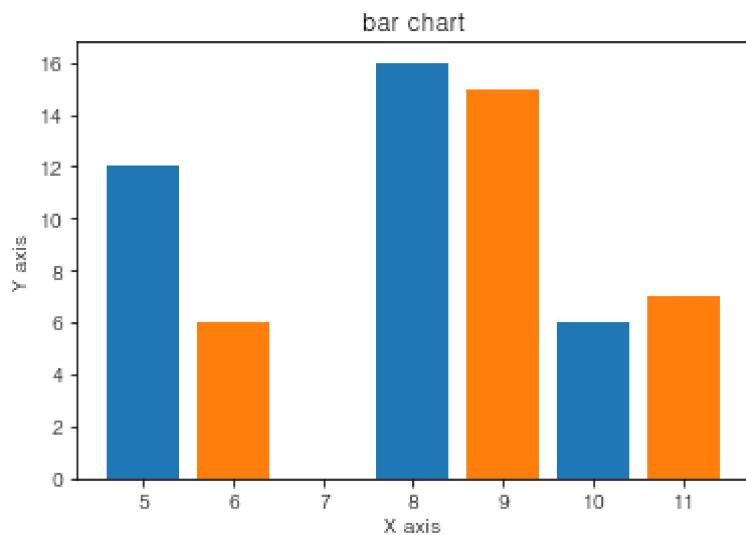
```
1 x = [5,8,10]
2 y = [12,16,6]
3 x2 = [6, 9, 11]
4 y2 = [6, 15, 7]
5
6 plt.plot(x,y)
7 plt.plot(x2,y2)
8
9 plt.title('Epic Info')
10 plt.ylabel('Y axis')
11 plt.xlabel('X axis')
12
13 plt.show()
```



## bar chart

In [5]:

```
1 plt.bar(x, y)
2 plt.bar(x2, y2)
3
4 plt.title('bar chart')
5 plt.ylabel('Y axis')
6 plt.xlabel('X axis')
7
8 plt.show()
```



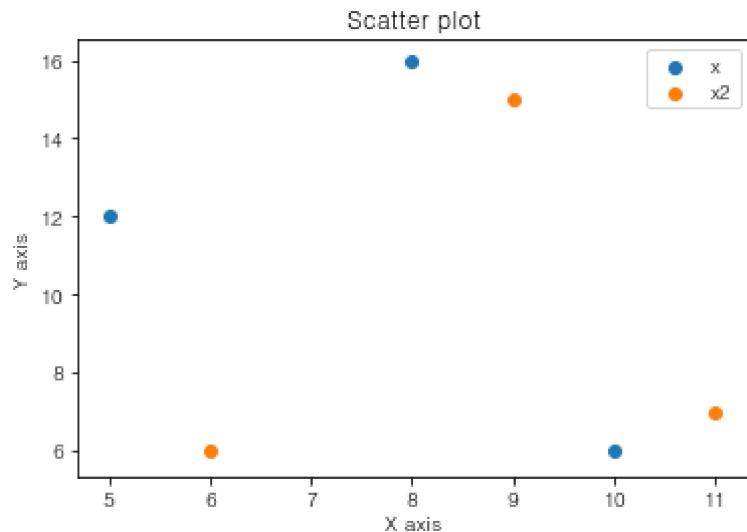
scatter plot (산점도)

In [6]:

```
1 plt.scatter(x, y, label='x')
2 plt.scatter(x2, y2, label='x2')
3
4 plt.title('Scatter plot')
5 plt.ylabel('Y axis')
6 plt.xlabel('X axis')
7 plt.legend()
```

Out[6]:

```
<matplotlib.legend.Legend at 0x1b3535716c8>
```



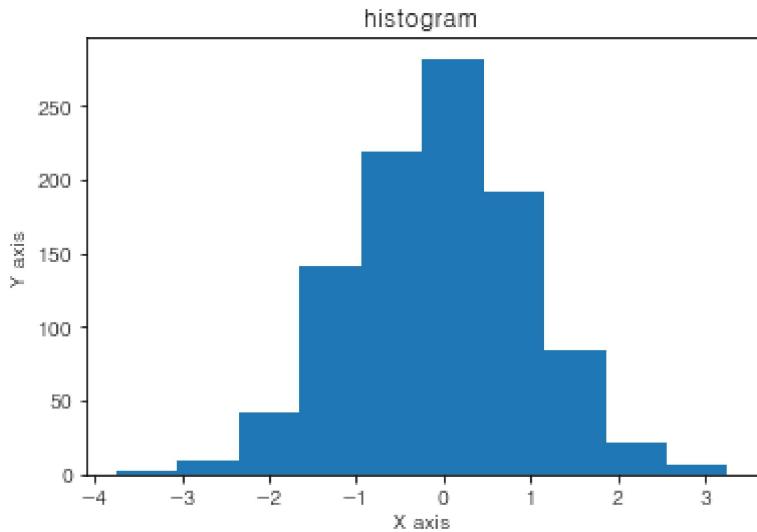
## histogram

In [7]:

```
1 x = np.random.randn(1000)
2
3 plt.hist(x, bins=10)
4
5 plt.title('histogram')
6 plt.ylabel('Y axis')
7 plt.xlabel('X axis')
```

Out[7]:

Text(0.5, 0, 'X axis')



## OOP style

### multiple subplots

$y = x + \text{noise}$  인  $(x, y)$  쌍의 toy data 생성

In [10]:

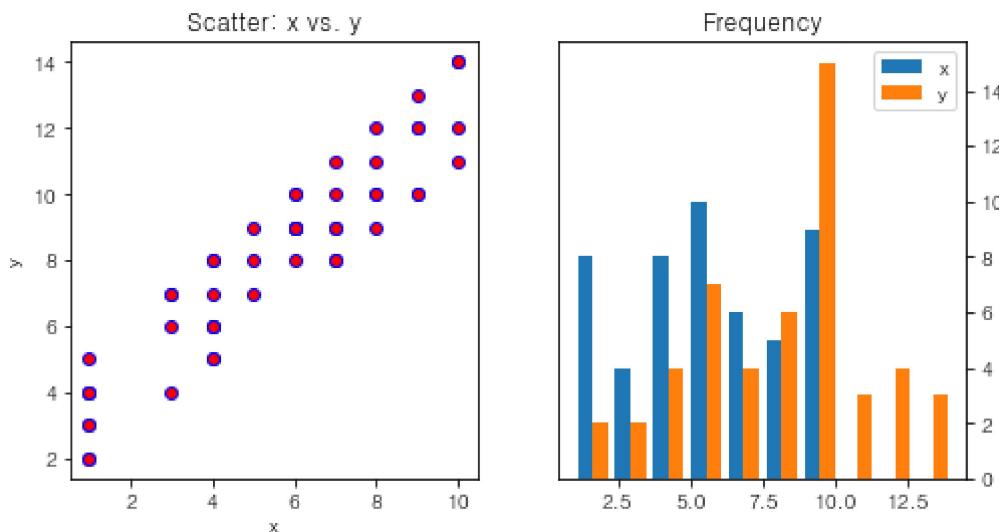
```
1 x = np.random.randint(low=1, high=11, size=50)
2 y = x + np.random.randint(1, 5, size=x.size)
3
4 data = np.column_stack((x, y))
5 data[:5]
```

Out[10]:

```
array([[ 6,  9],
       [ 4,  8],
       [ 5,  7],
       [ 1,  2],
       [ 9, 13]])
```

In [11]:

```
1 fig, (ax1, ax2) = plt.subplots(nrows=1, ncols=2, figsize=(8, 4))
2
3 ax1.scatter(x, y, marker='o', c='r', edgecolor='b')
4 ax1.set_title("Scatter: x vs. y")
5 ax1.set_xlabel('x')
6 ax1.set_ylabel('y')
7
8 ax2.hist(data, bins=10, label=('x', 'y'))
9 ax2.set_title("Frequency")
10 ax2.legend()
11 ax2.yaxis.tick_right()
```



## imshow

image data 처럼 행과 열을 가진 행렬 형태의 2차원 데이터는 imshow로 표시

In [12]:

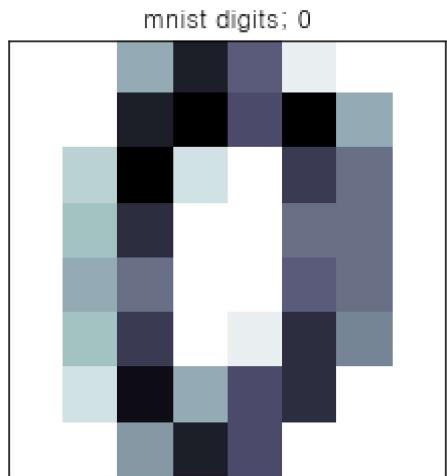
```
1 from sklearn.datasets import load_digits
2 digits = load_digits()
3 X = digits.images[0]
4 X
```

Out[12]:

```
array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

In [13]:

```
1 plt.title("mnist digits; 0")
2 plt.imshow(X, interpolation='nearest', cmap=plt.cm.bone_r)
3 plt.xticks([])
4 plt.yticks([])
5 plt.grid(False)
```



# Feature Scaling

- 특정 feature의 value가 다른 feature들 보다 훨씬 크면, 그 값이 목적함수를 지배하게 되므로 정확한 학습이 되지 않음
- sklearn의 preprocessing module은 scale, minmax\_scale 함수와 이에 대응하는 StandardScaler, MinMaxScaler class 및 fit(), transform() method를 제공하여 쉽게 scaling을 할 수 있도록 지원

## 1) Simple Feature Scaling

$$X_{new} = \frac{X_{old}}{X_{max}}$$

## 2) Min-Max Scaling

- 최대/최소값이 1, 0이 되도록 scaling
- x=min이면 y=0, x=max이면 y=1.

$$X_{new} = \frac{X_{old} - X_{min}}{X_{max} - X_{min}}$$

## 3) Standard Scaling (Z-score)

- 평균과 표준편차를 이용하여 scaling
- $\mu$ : 평균,  $\sigma$ : 표준편차

$$X_{new} = \frac{X_{old} - \mu}{\sigma}$$

In [1]:

```
1 import numpy as np
2 import pandas as pd
3 from sklearn.preprocessing import scale, minmax_scale
```

In [2]:

```
1 X = np.arange(-3, 6).astype('float32').reshape(-1, 1)
2 X = np.vstack([X, [20]]) # outlier 추가
3 df = pd.DataFrame(np.hstack([X, X/X.max(), scale(X), minmax_scale(X)]),
4                   columns=['X', 'simple scaling', 'standard scaling', 'minmax scaling'])
```

In [3]:

```
1 df
```

Out[3]:

	X	simple scaling	standard scaling	minmax scaling
0	-3.0	-0.15	-0.950995	0.000000
1	-2.0	-0.10	-0.789809	0.043478
2	-1.0	-0.05	-0.628624	0.086957
3	0.0	0.00	-0.467438	0.130435
4	1.0	0.05	-0.306253	0.173913
5	2.0	0.10	-0.145067	0.217391
6	3.0	0.15	0.016119	0.260870
7	4.0	0.20	0.177304	0.304348
8	5.0	0.25	0.338490	0.347826
9	20.0	1.00	2.756273	1.000000

## sklearn 의 MinMaxScaler class 사용

In [4]:

```
1 from sklearn.preprocessing import MinMaxScaler
2
3 scaler = MinMaxScaler()
4 after_norm = scaler.fit(X)
5 scaler.transform(X)
6
7 after_norm = scaler.fit_transform(X)
8 after_norm
```

Out[4]:

```
array([[0.        ],
       [0.04347826],
       [0.08695652],
       [0.13043478],
       [0.17391304],
       [0.2173913 ],
       [0.26086957],
       [0.30434783],
       [0.34782609],
       [1.        ]])
```

## sklearn 의 StandardScaler 사용

In [5]:

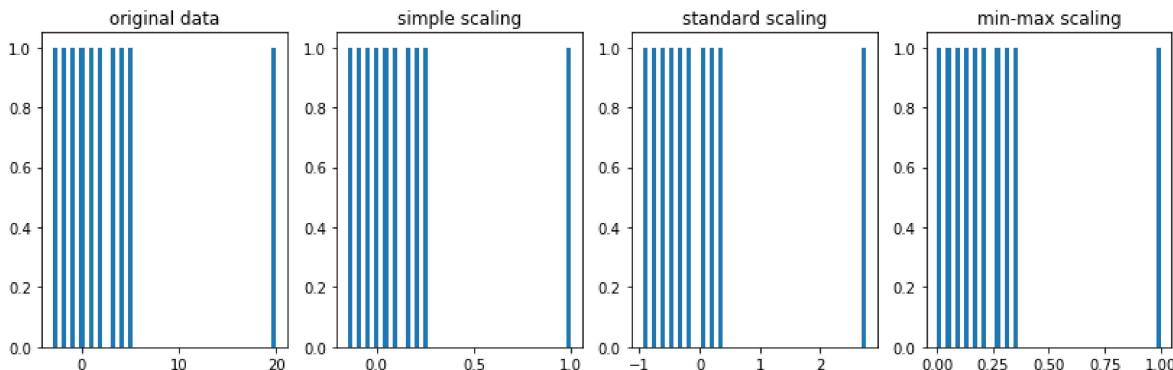
```
1 from sklearn.preprocessing import StandardScaler  
2  
3 scaler = StandardScaler()  
4 scaler.fit(X)  
5 scaler.transform(X)
```

Out[5]:

```
array([[-0.95099461],  
       [-0.78980908],  
       [-0.62862356],  
       [-0.46743803],  
       [-0.3062525 ],  
       [-0.14506697],  
       [ 0.01611855],  
       [ 0.17730408],  
       [ 0.33848961],  
       [ 2.75627252]])
```

In [6]:

```
1 import matplotlib.pyplot as plt  
2 %matplotlib inline  
3  
4 plt.figure(figsize=(14,4))  
5 plt.subplot(1,4,1)  
6 plt.hist(X, bins=50);  
7 plt.title('original data')  
8  
9 plt.subplot(1,4,2)  
10 plt.hist(X/X.max(), bins=50);  
11 plt.title('simple scaling')  
12  
13 plt.subplot(1,4,3)  
14 plt.hist(scale(X), bins=50);  
15 plt.title('standard scaling')  
16  
17 plt.subplot(1,4,4)  
18 plt.hist(minmax_scale(X), bins=50);  
19 plt.title('min-max scaling');
```





# 010.Simple Linear Regression

## Univariate Linear Regression (단변수 선형회귀)

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score
import pandas as pd
```

In [2]:

```
# Load the diabetes dataset
# 나이, 성별, 체질량지수, 혈압, 6가지 혈청 수치 --> already scaled
diabetes = datasets.load_diabetes()
diabetes.feature_names
```

Out[2]:

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [3]:

```
diabetes.data.shape
```

Out[3]:

```
(442, 10)
```

In [4]:

```
diabetes.data[:5]
```

Out[4]:

```
array([[ 0.03807591,  0.05068012,  0.06169621,  0.02187235, -0.0442235 ,
       -0.03482076, -0.04340085, -0.00259226,  0.01990842, -0.01764613],
      [-0.00188202, -0.04464164, -0.05147406, -0.02632783, -0.00844872,
       -0.01916334,  0.07441156, -0.03949338, -0.06832974, -0.09220405],
      [ 0.08529891,  0.05068012,  0.04445121, -0.00567061, -0.04559945,
       -0.03419447, -0.03235593, -0.00259226,  0.00286377, -0.02593034],
      [-0.08906294, -0.04464164, -0.01159501, -0.03665645,  0.01219057,
       0.02499059, -0.03603757,  0.03430886,  0.02269202, -0.00936191],
      [ 0.00538306, -0.04464164, -0.03638469,  0.02187235,  0.00393485,
       0.01559614,  0.00814208, -0.00259226, -0.03199144, -0.04664087]])
```

In [5]:

```
# 1년 뒤 측정한 당뇨병의 진행률 - target(label)
diabetes.target[:5]
```

Out[5]:

```
array([151.,  75., 141., 206., 135.])
```

In [6]:

```
df_diab = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
```

In [7]:

```
df_diab.head()
```

Out[7]:

	age	sex	bmi	bp	s1	s2	s3	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.0

◀ ▶

## univariate linear regression data 생성

In [8]:

```
# Use only one feature (단변수 선형회귀)
```

```
diabetes_X = df_diab['bmi'].values.reshape(-1, 1)
```

In [9]:

```
# training/testing data set 분할
```

```
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]
```

In [10]:

```
# targets data 를 training/testing set 으로 분리
```

```
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]
```

## sklearn model 이용

In [11]:

```
# regression object 생성
```

```
regr = linear_model.LinearRegression()
```

In [12]:

```
# training set 을 이용하여 model 훈련  
regr.fit(diabetes_X_train, diabetes_y_train)
```

Out[12]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [13]:

```
# The coefficients  
print('Coefficients: \n', regr.coef_)  
print('Intercept: \n', regr.intercept_)
```

```
Coefficients:  
[938.23786125]  
Intercept:  
152.91886182616167
```

In [14]:

```
print(regr.predict(np.array([[-0.050]])))  
print(regr.predict(np.array([[0.025]])))
```

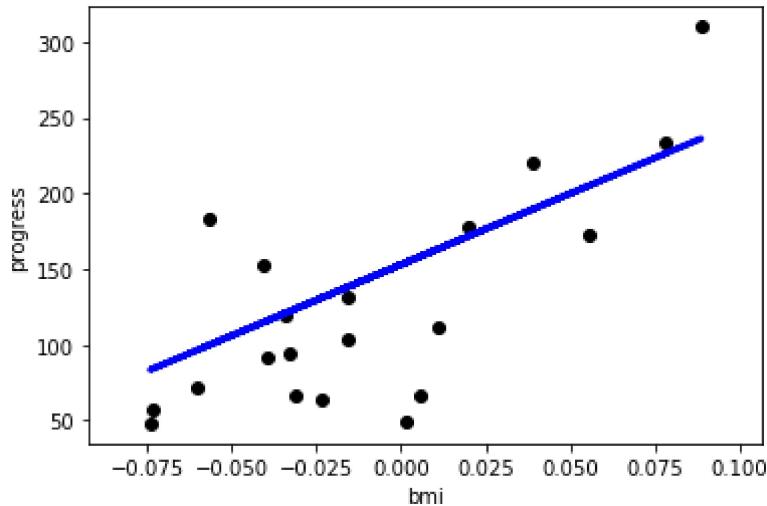
```
[106.00696876]  
[176.37480836]
```

In [15]:

```
y_pred = regr.predict(diabetes_X_test)
```

In [16]:

```
# visualization  
plt.scatter(diabetes_X_test, diabetes_y_test, color='black')  
plt.plot(diabetes_X_test, y_pred, color='blue', linewidth=3)  
  
plt.xlabel('bmi')  
plt.ylabel('progress')  
plt.show()
```



## R2 계산

In [17]:

```
print("결정계수 : {:.2f}".format(r2_score(diabetes_y_test, y_pred)))
```

결정계수 : 0.47

## Multivariate Linear Regression (다변수 선형회귀)

bmi, bp 두 가지 변수를 이용한 Linear Regression

In [18]:

```
df_diab.head()
```

Out[18]:

	age	sex	bmi	bp	s1	s2	s3	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.0

In [19]:

```
# 2 개의 feature 를 사용  
diabetes_X = df_diab[['bmi', 'bp']].values
```

In [20]:

```
diabetes_X.shape
```

Out[20]:

```
(442, 2)
```

In [21]:

```
# Split the data into training/testing sets  
diabetes_X_train = diabetes_X[:-20]  
diabetes_X_test = diabetes_X[-20:]
```

In [22]:

```
# Split the targets into training/testing sets  
diabetes_y_train = diabetes.target[:-20]  
diabetes_y_test = diabetes.target[-20:]
```

In [23]:

```
# Create linear regression object  
regr = linear_model.LinearRegression()
```

In [24]:

```
# Train the model using the training sets  
regr.fit(diabetes_X_train, diabetes_y_train)
```

Out[24]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [25]:

```
# Make predictions using the testing set  
diabetes_y_pred = regr.predict(diabetes_X_test)
```

In [26]:

```
# The coefficients  
print('Coefficients: \n', regr.coef_)  
print('Intercept: \n', regr.intercept_)
```

Coefficients:

```
[781.17008831 410.14503587]
```

Intercept:

```
152.86283685571757
```

In [27]:

```
regr.predict(np.array([[0.08, -0.075]]))
```

Out[27]:

```
array([184.59556623])
```

In [28]:

```
# R2 계산
print("결정계수: {:.2f}".format(r2_score(diabetes_y_test, diabetes_y_pred)))
```

결정계수: 0.47

In [29]:

```
# 3 차원 표현을 위한 meshgrid 작성
x1_min = diabetes_X_test[:, 0].min()
x1_max = diabetes_X_test[:, 0].max()
x2_min = diabetes_X_test[:, 1].min()
x2_max = diabetes_X_test[:, 1].max()

X1, X2 = np.meshgrid(np.linspace(x1_min, x1_max, 100), np.linspace(x2_min, x2_max, 100))
```

In [30]:

```
print(X1.shape)
print(X2.shape)
```

```
(100, 100)
(100, 100)
```

In [31]:

```
XX = np.column_stack([X1.ravel(), X2.ravel()])
```

In [32]:

```
Y = regr.predict(XX)
Y
```

Out[32]:

```
array([ 61.58034238,  62.86453569,  64.148729 , ..., 255.33883314,
       256.62302645, 257.90721976])
```

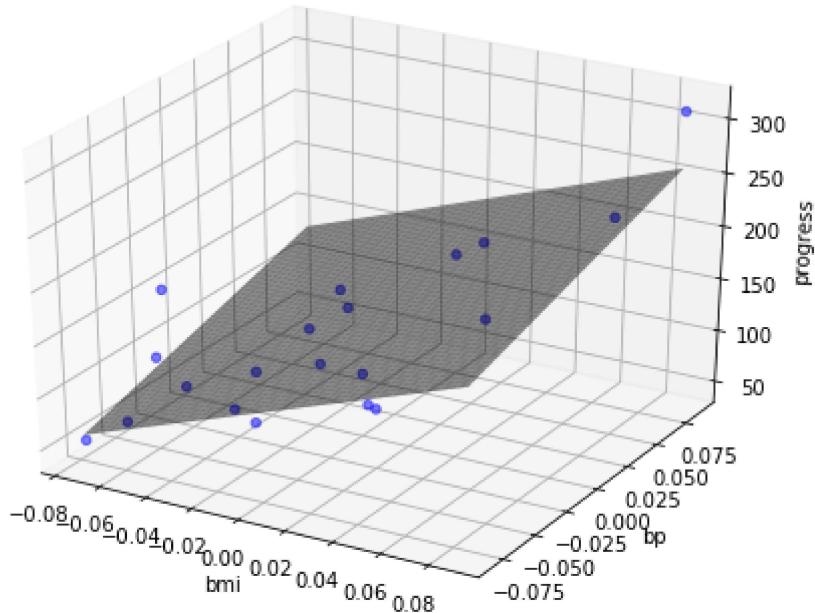
In [33]:

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(diabetes_X_test[:,0], diabetes_X_test[:,1], diabetes_y_test, c='blue', marker='o', alpha=0.5)
ax.plot_surface(X1, X2, Y.reshape(X1.shape), color='None', alpha=0.5)

ax.set_xlabel('bmi')
ax.set_ylabel('bp')
ax.set_zlabel('progress')
plt.show()
```



# 020.KNN (K-Nearest Neighbors, 최근접 이웃)

## iris dataset

iris.feature\_names :

Sepal Length : 꽃받침 길이

Sepal Width : 꽃받침 너비

Petal Length : 꽃잎 길이

Petal Width : 꽃잎 너비

Species (꽃의 종류) : setosa / versicolor / virginica 의 3종류로 구분된다.

꽃받침 길이, 너비 두가지 feature 를 가지고 KNN 알고리즘을 이용하여 꽃의 종류 분류

**neighbors.KNeighborsClassifier(n\_neighbors, weights=weights)**

- weights :
  - uniform : uniform weights. 모든 neighbor 의 가중치를 동일하게 취급
  - distance : neighbor 의 거리에 반비례하여 가중치 조정

In [1]:

```
import numpy as np
from sklearn import neighbors, datasets
```

In [2]:

```
iris = datasets.load_iris()
```

```
print(iris.data.shape)
print(iris.feature_names)
print(iris.target_names)
```

```
(150, 4)
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
['setosa' 'versicolor' 'virginica']
```

In [3]:

```
# 꽃받침 길이, 너비 두가지 feature 선택
X = iris.data[:, :2]
y = iris.target

print(X[:5])
print()
print(y[:5])
```

```
[[5.1 3.5]
[4.9 3. ]
[4.7 3.2]
[4.6 3.1]
[5.  3.6]]
```

```
[0 0 0 0 0]
```

In [4]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

## KNN object 생성 및 train

In [5]:

```
clf = neighbors.KNeighborsClassifier(n_neighbors=15, weights='uniform')
clf.fit(X_train, y_train)
```

Out[5]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                      metric_params=None, n_jobs=None, n_neighbors=15, p=2,
                      weights='uniform')
```

In [6]:

```
y_predict = clf.predict(X_test)
y_predict
```

Out[6]:

```
array([0, 2, 1, 1, 0, 1, 0, 1, 2, 2, 0, 0, 2, 2, 0, 2, 1, 1, 2, 1, 0, 2,
       1, 0, 0, 1, 0, 2, 2, 2, 0, 1, 2, 0, 2, 1, 1])
```

In [7]:

```
from sklearn.metrics import confusion_matrix

confusion_matrix(y_test, y_predict)
```

Out[7]:

```
array([[12,  0,  0],
       [ 0,  9,  5],
       [ 0,  3,  9]], dtype=int64)
```

## 예측의 정확도 평가

In [41]:

```
from sklearn.metrics import accuracy_score
```

In [42]:

```
accuracy_score(y_test, y_predict)
```

Out[42]:

```
0.7894736842105263
```

# 030.Decision Tree

## iris dataset

iris.feature\_names :

Sepal Length : 꽃받침 길이

Sepal Width : 꽃받침 너비

Petal Length : 꽃잎 길이

Petal Width : 꽃잎 너비

Species (꽃의 종류) : setosa / versicolor / virginica 의 3종류로 구분된다.

위 feature 를 모두 가지고 Decision Tree 알고리즘을 이용하여 꽃의 종류 분류

In [1]:

```
from sklearn.datasets import load_iris  
from sklearn import tree  
  
iris = load_iris()
```

In [2]:

```
iris.data.shape
```

Out[2]:

```
(150, 4)
```

In [3]:

```
iris.feature_names
```

Out[3]:

```
['sepal length (cm)',  
'sepal width (cm)',  
'petal length (cm)',  
'petal width (cm)']
```

In [4]:

```
clf = tree.DecisionTreeClassifier(max_depth=2)
```

In [5]:

```
clf.fit(iris.data, iris.target)
```

Out[5]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                      max_depth=2, max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

## graphviz 설치

- conda install python-graphviz
- pip install pydotplus

In [6]:

```
import graphviz

dot_data = tree.export_graphviz(clf, out_file=None, feature_names=iris.feature_names, class_
_names=iris.target_names,
                               filled=True, rounded=True, special_characters=True)
```

## gini - 지니 불순도

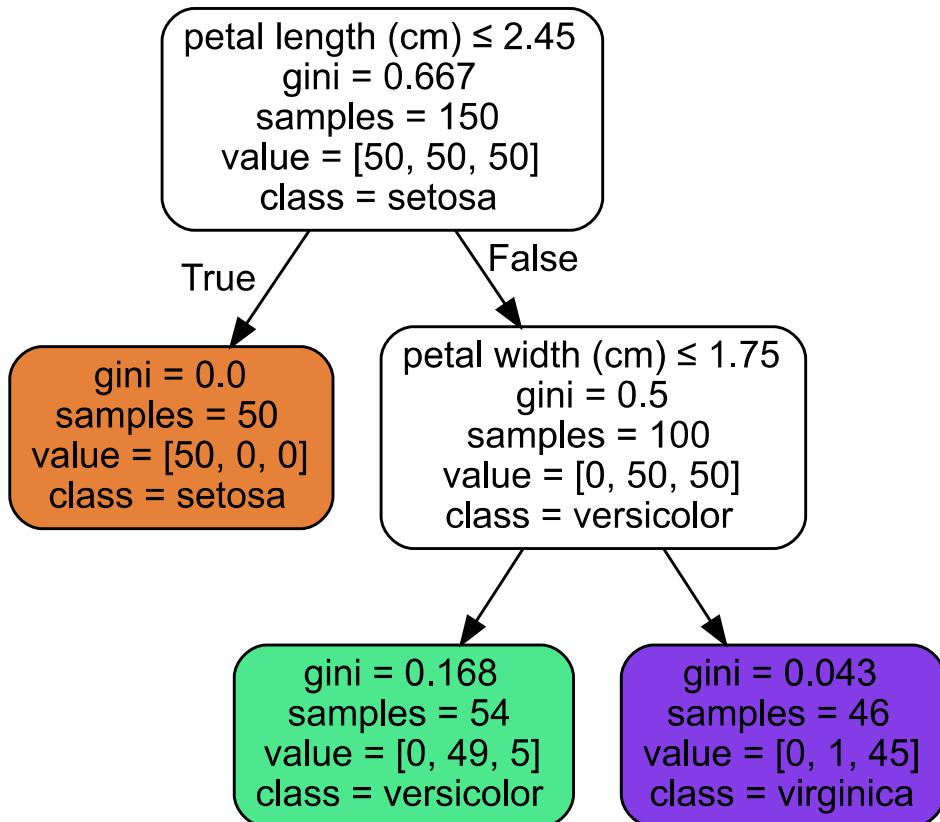
지니 불순도는 집합에 이질적인 것이 얼마나 섞였는지를 측정하는 지표이며 CART (Classification And Regression Tree) 알고리즘에서 사용한다.

집합에 있는 항목이 모두 같다면 지니 불순도는 최솟값(0)을 갖게 되며 이 집합은 완전히 순수하다고 할 수 있다.

In [7]:

```
graphviz.Source(dot_data)
```

Out[7]:

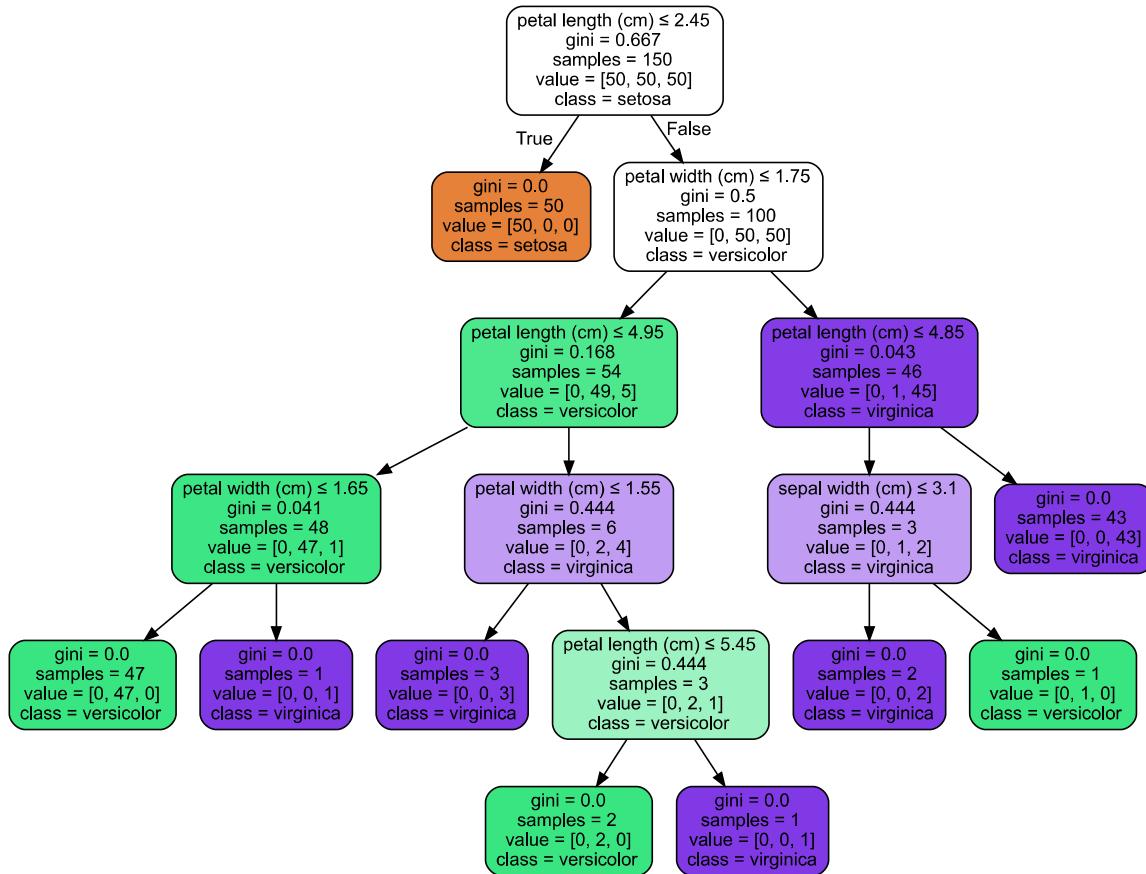


In [8]:

```
clf = tree.DecisionTreeClassifier(max_depth=None)
clf.fit(iris.data, iris.target)

dot_data = tree.export_graphviz(clf, out_file=None, feature_names=iris.feature_names, class_
_names=iris.target_names,
                               filled=True, rounded=True, special_characters=True)
graphviz.Source(dot_data)
```

Out[8]:



## train / test split & accuracy check

In [9]:

```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

In [10]:

```
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2)
```

In [11]:

```
clf = tree.DecisionTreeClassifier(max_depth=None)
clf.fit(X_train, y_train)
```

Out[11]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                      max_depth=None, max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=None, splitter='best')
```

In [12]:

```
y_pred = clf.predict(X_test)
```

In [13]:

```
accuracy_score(y_test, y_pred)
```

Out[13]:

```
0.9666666666666667
```

## 040.Simple Train / Test Split

- 일정 비율로 data 분할 (ex. 80 : 20)

In [1]:

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
```

In [2]:

```
# Load the diabetes dataset
# 나이, 성별, 체질량지수, 혈압, 6가지 혈청 수치
diabetes = datasets.load_diabetes()
diabetes.feature_names
```

Out[2]:

```
['age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6']
```

In [3]:

```
df_diab = pd.DataFrame(data=diabetes.data, columns=diabetes.feature_names)
```

In [4]:

```
df_diab.head()
```

Out[4]:

	age	sex	bmi	bp	s1	s2	s3	
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.0

◀ ▶

In [5]:

```
X = df_diab.values
X.shape
```

Out[5]:

```
(442, 10)
```

In [6]:

```
y = diabetes.target  
y.shape
```

Out[6]:

```
(442,)
```

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [8]:

```
print(X_train.shape)  
print(y_train.shape)  
print()  
print(X_test.shape)  
print(y_test.shape)
```

```
(353, 10)  
(353,)
```

```
(89, 10)  
(89,)
```

In [9]:

```
# regression object 생성  
lm = linear_model.LinearRegression()  
  
# training set 을 이용하여 model 훈련  
lm.fit(X_train, y_train)
```

Out[9]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [10]:

```
# testing set 을 이용한 예측  
y_pred = lm.predict(X_test)  
  
# MSE(mean squared error)  
print("Mean squared error: {:.2f}".format(mean_squared_error(y_test, y_pred)))  
# R2  
print("Variance score: {:.2f}".format(r2_score(y_test, y_pred)))
```

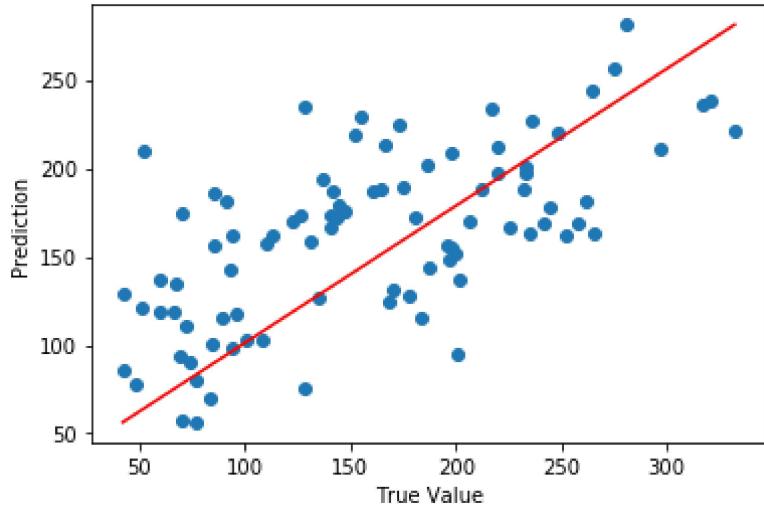
Mean squared error: 3179.57

Variance score: 0.41

In [11]:

```
# visualization
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_pred.min(), y_pred.max()], c='r')

plt.xlabel('True Value')
plt.ylabel('Prediction')
plt.show()
```



## K-Fold Validation

- data 가 적은 경우 사용
- data 가 충분하더라도 training 중 과적합을 방지하려는 경우 사용

In [12]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, cross_val_predict
```

In [13]:

```
lm = linear_model.LinearRegression()
lm.fit(X, y)

scores = cross_val_score(lm, X, y, cv=6)

print("cross validation scores : ", scores)
print("cross validation mean scores : {:.2f}".format(scores.mean()))
```

```
cross validation scores : [0.4554861  0.46138572 0.40094084 0.55220736 0.43
942775 0.56923406]
cross validation mean scores : 0.48
```

In [14]:

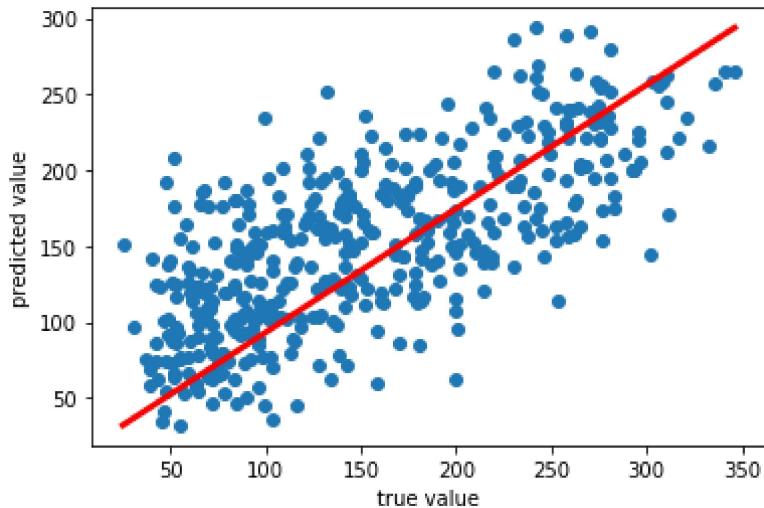
```
y_pred = cross_val_predict(lm, X, y, cv=6)

plt.scatter(y, y_pred)
plt.plot([y.min(), y.max()], [y_pred.min(), y_pred.max()], c='r', lw=3)

plt.xlabel('true value')
plt.ylabel('predicted value')
```

Out[14]:

Text(0, 0.5, 'predicted value')



In [15]:

```
# MSE(mean squared error)
print("Mean squared error: {:.2f}".format(mean_squared_error(y, y_pred)))
# R2
print("Variance score: {:.2f}".format(r2_score(y, y_pred)))
```

Mean squared error: 3019.46

Variance score: 0.49

# 050.Logistic Regression

data: 성별 소득 data에 따라 특정 구매자의 구매할지 여부를 예측

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

dataset = pd.read_csv('datasets/Social_Network_Ads.csv')
```

In [2]:

```
dataset.tail()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

In [3]:

```
dataset['Purchased'].value_counts()
```

Out[3]:

```
0    257
1    143
Name: Purchased, dtype: int64
```

**Age, EstimatedSalary** 를 변수로 선택

In [4]:

```
X = dataset.iloc[:, [2,3]].values.astype("float32")
y = dataset.iloc[:, 4].values.astype("float32")
```

dataset 을 Training 과 Test set 으로 분리

In [5]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

## Feature Scaling

train data 를 fit 하고, test data 는 train data 의 분포에 맞추어 transform

In [6]:

```
sc = StandardScaler()  
X_train = sc.fit_transform(X_train)  
X_test = sc.transform(X_test)
```

In [7]:

```
print(X_train.shape)
```

```
(320, 2)
```

Training set 에 대해 Logistic Regression model 을 fitting

In [8]:

```
lr_classifier = LogisticRegression(solver='lbfgs', random_state=0)  
lr_classifier.fit(X_train, y_train)
```

Out[8]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
intercept_scaling=1, l1_ratio=None, max_iter=100,  
multi_class='auto', n_jobs=None, penalty='l2',  
random_state=0, solver='lbfgs', tol=0.0001, verbose=0,  
warm_start=False)
```

## predict

- predict() - 예측된 class 반환 using X\_test
- predict\_proba() - class 당 probability 반환 using X\_test

In [9]:

```
y_pred = lr_classifier.predict(X_test)

print(y_pred)
print()
print("Test set true counts = ", sum(y_test))
print("predicted true counts = ", sum(y_pred))
print("accuracy = {:.2f}".format(sum(y_pred == y_test) / len(y_test)))
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0. 0. 0. 1.
 0. 1. 1. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 1. 0.
 0. 0. 0. 0. 1. 0. 1. 1.]
```

```
Test set true counts = 22.0
predicted true counts = 18.0
accuracy = 0.93
```

In [10]:

```
y_pred_proba = lr_classifier.predict_proba(X_test)

print(y_pred_proba[:5])
print()
print(y_pred_proba[:,1][:50])
```

```
[[0.87397564 0.12602436]
 [0.82308938 0.17691062]
 [0.7922792 0.2077208]
 [0.89908522 0.10091478]
 [0.89298557 0.10701443]]

[0.12602436 0.17691062 0.2077208 0.10091478 0.10701443 0.00976727
 0.01804698 0.73434492 0.00762765 0.49898276 0.04202461 0.03461863
 0.17142359 0.37820537 0.01954722 0.35259451 0.29180572 0.01565439
 0.98632223 0.04749635 0.09836247 0.95925398 0.28456724 0.87407643
 0.0057815 0.96935938 0.08898959 0.08702488 0.20483206 0.17101724
 0.0251403 0.30491539 0.92241231 0.1627098 0.01775628 0.00463549
 0.02454727 0.07340366 0.03224642 0.54144972 0.07660121 0.2909855
 0.06819766 0.04474186 0.79863516 0.03140396 0.31159272 0.91993059
 0.01098176 0.84966242]
```

In [11]:

```
y_pred_proba_1 = y_pred_proba[:, 1]

THRESHOLD = 0.5
print("Counts of y_predicted as 1 with threshold {} : ".format(THRESHOLD),
      sum(y_pred_proba_1 > THRESHOLD))

THRESHOLD = 0.6
print("Counts of y_predicted as 1 with threshold {} : ".format(THRESHOLD),
      sum(y_pred_proba_1 > THRESHOLD))
```

```
Counts of y_predicted as 1 with threshold 0.5 : 18
Counts of y_predicted as 1 with threshold 0.6 : 16
```

## confusion matrix 를 이용한 model 평가

In [12]:

```
from sklearn.metrics import confusion_matrix, f1_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, roc_auc_score
from sklearn.model_selection import cross_val_predict
```

In [13]:

```
print("confusion matrix\n", confusion_matrix(y_test, y_pred, labels=[1, 0]))
print()
print("f1 score\n", f1_score(y_test, y_pred))
print()
print("Accuracy: {:.2f}".format(accuracy_score(y_test, y_pred)))
print("Precision: {:.2f}".format(precision_score(y_test, y_pred, labels=[1, 0])))
print("Recall: {:.2f}".format(recall_score(y_test, y_pred, labels=[1, 0])))
```

confusion matrix

```
[[17  5]
 [ 1 57]]
```

f1 score

0.85

Accuracy: 0.93

Precision: 0.94

Recall: 0.77

## Visualization

```
roc_curve(y_true, y_score)
```

fpr - false positive rates, tpr - true positive rates

In [14]:

```
y_probas = lr_classifier.predict_proba(X_test)
y_scores = y_probas[:,1]

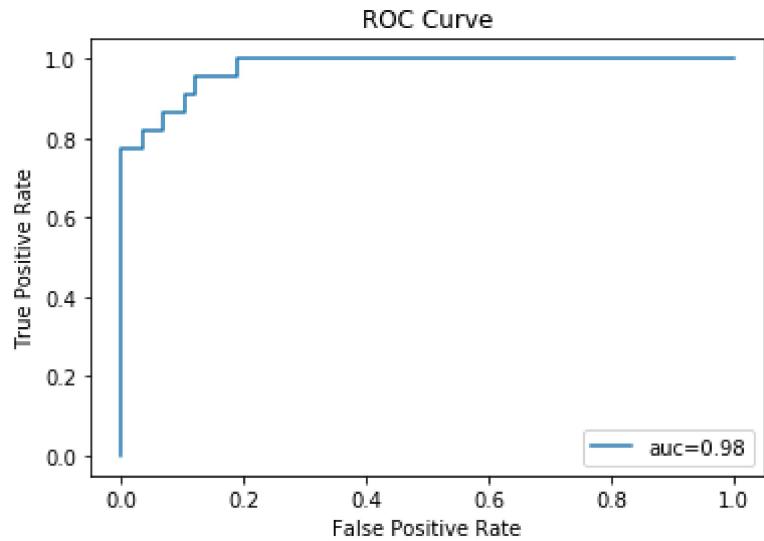
fpr, tpr, _ = roc_curve(y_test, y_scores)
auc = roc_auc_score(y_test, y_scores)
```

In [15]:

```
plt.plot(fpr, tpr, label="auc=" + "{:.2f}".format(auc))
plt.legend(loc=4)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
```

Out[15]:

Text(0.5, 1.0, 'ROC Curve')



# 060.SVM (Support Vector Machine)

data: 성별 소득 data에 따라 특정 구매자의 구매할지 여부를 예측

In [1]:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 from sklearn.model_selection import train_test_split
5 from sklearn.preprocessing import StandardScaler
6 from sklearn.svm import SVC
7
8 dataset = pd.read_csv('datasets/Social_Network_Ads.csv')
```

In [2]:

```
1 dataset.head()
```

Out[2]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [3]:

```
1 X = dataset.iloc[:, [2,3]].values.astype("float32")
2 y = dataset.iloc[:, 4].values.astype("float32")
```

In [4]:

```
1 # dataset 을 Training 과 Test set 으로 분리
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [5]:

```
1 # Feature Scaling
2 sc = StandardScaler()
3 X_train = sc.fit_transform(X_train)
4 X_test = sc.transform(X_test)
5 print(X_train.shape)
```

(320, 2)

Training set에 대해 Logistic Regression을 fitting

- rbf (Radial Basis Function) 사용

$$k(x_1, x_2) = \exp(-\gamma ||x_1 - x_2||^2)$$

In [6]:

```
1 classifier = SVC(kernel='rbf')
2
3 classifier.fit(X_train, y_train)
```

Out[6]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [7]:

```
1 y_pred = classifier.predict(X_test)
2
3 print(y_pred)
4 print()
5 print("Test set true counts = ", sum(y_test))
6 print("predicted true counts = ", sum(y_pred))
7 print("accuracy = {:.2f}".format(sum(y_pred == y_test) / len(y_test)))
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0. 0. 0. 1. 0. 1. 0. 1.
 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 1. 0. 0. 1.
 0. 1. 1. 0. 0. 1. 1. 1. 0. 0. 1. 0. 0. 1. 0. 1. 0. 1. 0. 0. 0. 0. 1. 0.
 0. 1. 0. 0. 0. 0. 1. 1.]
```

```
Test set true counts = 22.0
predicted true counts = 24.0
accuracy = 0.95
```

In [8]:

```
1 from sklearn.metrics import confusion_matrix, f1_score
2 from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_curve, roc_auc
3 from sklearn.model_selection import cross_val_predict
4
5 # making confusion matrix
6 print("confusion matrix\n", confusion_matrix(y_test, y_pred, labels=[1, 0]))
7 print()
8 print("f1 score\n", f1_score(y_test, y_pred))
9 print()
10 print("Accuracy: {:.2f}\n".format(accuracy_score(y_test, y_pred)))
11 print("Precision: {:.2f}\n".format(precision_score(y_test, y_pred, labels=[1, 0])))
12 print("Recall: {:.2f}\n".format(recall_score(y_test, y_pred, labels=[1, 0])))
```

confusion matrix

```
[[21  1]
 [ 3 55]]
```

f1 score

```
0.9130434782608695
```

Accuracy: 0.95

Precision: 0.88

Recall: 0.95

## 070.Random Forest

- Titanic 호의 생존 예측

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, f1_score, accuracy_score

dataset = pd.read_csv("datasets/titanic.csv")
```

In [2]:

```
dataset.head()
```

Out[2]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 2117
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17543
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O 310128
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450

In [3]:

```
dataset.drop(['PassengerId', 'Name', 'Cabin', 'Ticket'], axis=1, inplace=True)
```

In [4]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype  
 --- 
 0   Survived  891 non-null   int64  
 1   Pclass    891 non-null   int64  
 2   Sex       891 non-null   object  
 3   Age       714 non-null   float64 
 4   SibSp    891 non-null   int64  
 5   Parch    891 non-null   int64  
 6   Fare     891 non-null   float64 
 7   Embarked 889 non-null   object  
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

In [5]:

```
mean_age = dataset['Age'].mean()

dataset['Age'].fillna(mean_age, inplace=True)
```

In [6]:

```
dataset = pd.get_dummies(dataset)

dataset.head()
```

Out[6]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarkt
0	0	3	22.0	1	0	7.2500	0	1	
1	1	1	38.0	1	0	71.2833	1	0	
2	1	3	26.0	0	0	7.9250	1	0	
3	1	1	35.0	1	0	53.1000	1	0	
4	0	3	35.0	0	0	8.0500	0	1	

In [7]:

```
Y = dataset['Survived']
X = dataset.drop(['Survived'], axis=1)
```

In [8]:

```
# dataset 을 Training 과 Test set 으로 분리
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

In [9]:

```
# Feature Scaling
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
print(X_train.shape)
```

(712, 10)

In [10]:

```
# Training set of the Random Forest Classifier model is fitting
classifier = RandomForestClassifier(n_estimators=10, criterion='entropy', random_state=0)
classifier.fit(X_train, y_train)
```

Out[10]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                      criterion='entropy', max_depth=None, max_features='auto',
                      max_leaf_nodes=None, max_samples=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10,
                      n_jobs=None, oob_score=False, random_state=0, verbose=0,
                      warm_start=False)
```

In [11]:

```
y_pred = classifier.predict(X_test)

print(y_pred)
print()
print("Test set true counts = ", sum(y_test))
print("predicted true counts = ", sum(y_pred))
print("accuracy = {:.2f}".format(sum(y_pred == y_test) / len(y_test)))

[0 0 0 1 0 0 1 1 1 0 1 0 1 1 1 0 0 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 1 1 1 0 1 0
 0 0 1 1 0 1 0 1 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 1 0
 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 0 0 1
 1 0 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1]
```

Test set true counts = 69  
predicted true counts = 61  
accuracy = 0.81

In [12]:

```
# making confusion matrix
print("confusion matrix\n", confusion_matrix(y_test, y_pred, labels=[1, 0]))
print()
print("f1 score\n", f1_score(y_test, y_pred, labels=[1, 0]))
```

confusion matrix  
[[48 21]  
 [13 97]]

f1 score  
0.7384615384615385

# Gradient Boosting Classifier

- min\_samples\_split : node 분리에 필요한 최소 sample 수 => overfitting 방지
- max\_depth : tree 깊이 조절 => overfitting 방지
- learning\_rate : 각 tree 의 기여도 조정, n\_estimators 와 trade-off
- n\_estimators : number of sequential trees

In [13]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbclassifier = GradientBoostingClassifier(learning_rate=0.1, n_estimators=500, max_depth=5)
gbclassifier.fit(X_train, y_train)
```

Out[13]:

```
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=5,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=500,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
```

In [14]:

```
y_pred = gbclassifier.predict(X_test)

print(y_pred)
print()
print("Test set true counts = ", sum(y_test))
print("predicted true counts = ", sum(y_pred))
print("accuracy = {:.2f}".format(sum(y_pred == y_test) / len(y_test)))
```

```
[0 0 0 1 0 0 1 1 1 0 1 0 1 1 0 0 0 0 0 0 1 0 0 1 1 0 1 1 0 0 1 0 0 0 0
 0 0 0 0 1 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 1 0 1 0 1 1 1 0 0 0
 0 0 0 0 0 0 0 1 0 0 0 1 1 0 1 1 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 1 1 0 0 1 0
 1 1 0 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 1 1 0 1
 1 0 0 1 1 1 0 0 0 1 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 0 1 0 1]
```

```
Test set true counts =  69
predicted true counts =  60
accuracy = 0.82
```

In [15]:

```
# making confusion matrix
print("confusion matrix\n", confusion_matrix(y_test, y_pred, labels=[1, 0]))
print()
print("f1 score\n", f1_score(y_test, y_pred, labels=[1, 0]))
```

confusion matrix

```
[[48 21]
 [12 98]]
```

f1 score

```
0.7441860465116279
```

In [16]:

```
gbclassifier.feature_importances_      # Age, EstimatedSalary 의 중요도
```

Out[16]:

```
array([0.09770708, 0.21109568, 0.05260612, 0.02299356, 0.26469205,
       0.14813365, 0.17907038, 0.00735964, 0.00546825, 0.01087359])
```

In [17]:

```
feature_imp = pd.Series(gbclassifier.feature_importances_, X.columns).sort_values(ascending=False)
feature_imp
```

Out[17]:

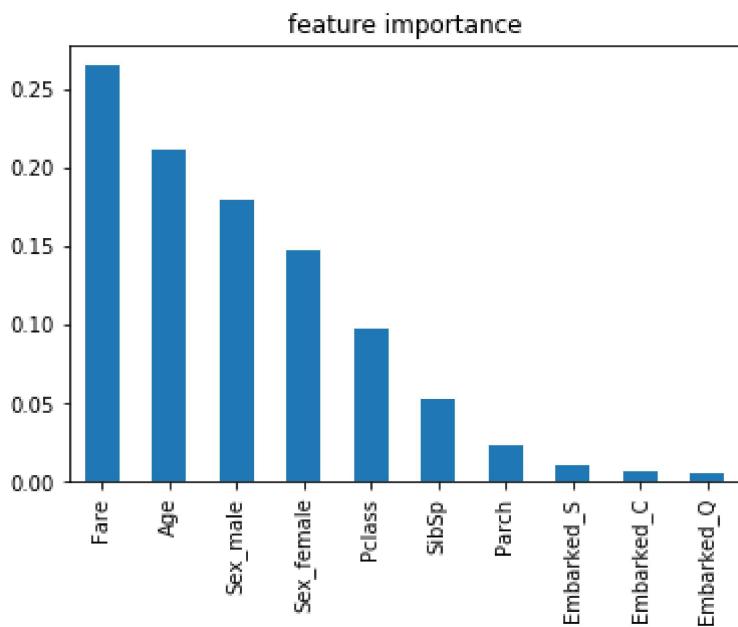
```
Fare      0.264692
Age       0.211096
Sex_male   0.179070
Sex_female  0.148134
Pclass     0.097707
SibSp      0.052606
Parch      0.022994
Embarked_S 0.010874
Embarked_C  0.007360
Embarked_Q  0.005468
dtype: float64
```

In [18]:

```
feature_imp.plot(kind='bar', title='feature importance')
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21648554288>
```



# 075. 실습 - Titanic 호 data 를 이용한 Feature Engineering 과 Modeling

Pclass - Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd) → 객실 등급

survival - Survival (0 = No; 1 = Yes) → 생존여부

name - Name

sex - Sex

age - Age

sibsp - Number of Siblings/Spouses Aboard → 함께 탑승한 형제 또는 배우자 수

parch - Number of Parents/Children Aboard → 함께 탑승한 부모 또는 자녀 수

ticket - Ticket Number

fare - Passenger Fare (British pound)

cabin - Cabin → 선실번호

embarked - Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton) → 탑승한 항구

In [1]:

```
1 import numpy as np  
2 import pandas as pd  
3 import matplotlib.pyplot as plt
```

In [2]:

```
1 df_titanic = pd.read_csv("datasets/titanic.csv")
```

In [3]:

```
1 df_titanic.shape
```

Out[3]:

(891, 12)

In [4]:

```
1 df_titanic.head()
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450

In [5]:

```
1 df_titanic.drop(['PassengerId', 'Name', 'Cabin', 'Ticket'], axis=1, inplace=True)
2 df_titanic.head()
```

Out[5]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	3	male	22.0	1	0	7.2500	S
1	1	1	female	38.0	1	0	71.2833	C
2	1	3	female	26.0	0	0	7.9250	S
3	1	1	female	35.0	1	0	53.1000	S
4	0	3	male	35.0	0	0	8.0500	S

In [6]:

```
1 df_titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 8 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   Survived  891 non-null   int64  
 1   Pclass    891 non-null   int64  
 2   Sex       891 non-null   object  
 3   Age       714 non-null   float64 
 4   SibSp    891 non-null   int64  
 5   Parch    891 non-null   int64  
 6   Fare     891 non-null   float64 
 7   Embarked 889 non-null   object  
dtypes: float64(2), int64(4), object(2)
memory usage: 55.8+ KB
```

In [7]:

```
1 df_titanic.isnull().sum()
```

Out[7]:

```
Survived      0
Pclass        0
Sex           0
Age          177
SibSp        0
Parch        0
Fare         0
Embarked     2
dtype: int64
```

In [8]:

```
1 df_titanic.describe()
```

Out[8]:

	Survived	Pclass	Age	SibSp	Parch	Fare
<b>count</b>	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
<b>mean</b>	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
<b>std</b>	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
<b>min</b>	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
<b>25%</b>	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
<b>50%</b>	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
<b>75%</b>	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
<b>max</b>	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

## outlier 검출

- outlier 가 prediction(특히 regression) 에 영향을 큰 영향을 줄 수 있으므로, 연속값으로 구성된 column 의 outlier 검출
- outlier 의 정의는 일반적으로 IQR(Inter-quartile range, Q3 - Q1) \* 1.5 보다 큰 경우로 한다.
- 세개 이상의 outlier 값을 가진 row 를 drop 시킨다.

In [9]:

```
1 THRESHOLD_OUTLIER = 3
2 outlier_indices = []
3
4 for col in ['Age', 'SibSp', 'Parch', 'Fare']:
5     Q1 = np.percentile(df_titanic[col], 25)
6     Q3 = np.percentile(df_titanic[col], 75)
7     IQR = Q3 - Q1
8     outlier_step = IQR * 1.5
9     outlier_list = df_titanic[(df_titanic[col] < Q1 - outlier_step) |
10                             (df_titanic[col] > Q3 + outlier_step)].index
11     outlier_indices.extend(outlier_list)
```

In [10]:

```
1 len(outlier_indices)
```

Out[10]:

375

In [11]:

```
1 outlier_indices[:10]
```

Out[11]:

[7, 16, 24, 27, 50, 59, 63, 68, 71, 85]

In [12]:

```
1 from collections import Counter
2
3 multiple_outliers = list(k for k, v in Counter(outlier_indices).items() if v >= THRESHOLD_OUT
4 multiple_outliers
```

Out[12]:

[27, 88, 159, 180, 201, 324, 341, 792, 846, 863]

In [13]:

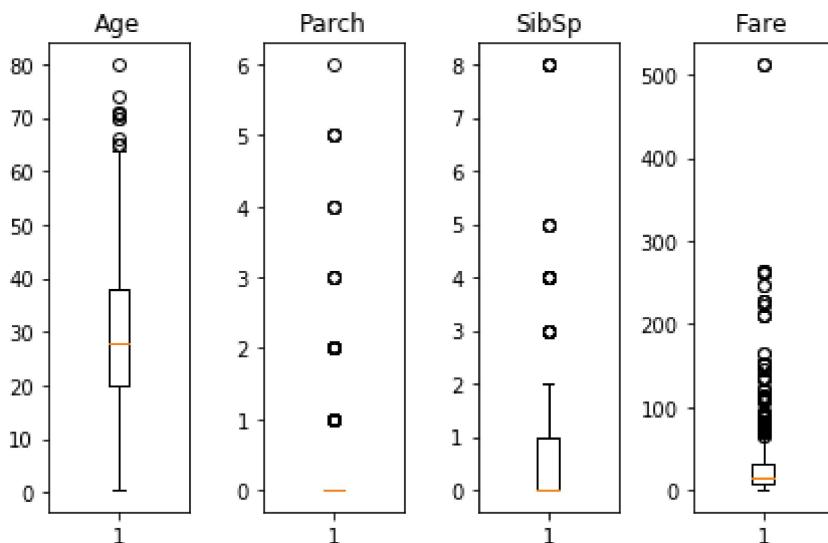
```
1 df_titanic.iloc[multiple_outliers]
```

Out[13]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
27	0	1	male	19.0	3	2	263.00	S
88	1	1	female	23.0	3	2	263.00	S
159	0	3	male	NaN	8	2	69.55	S
180	0	3	female	NaN	8	2	69.55	S
201	0	3	male	NaN	8	2	69.55	S
324	0	3	male	NaN	8	2	69.55	S
341	1	1	female	24.0	3	2	263.00	S
792	0	3	female	NaN	8	2	69.55	S
846	0	3	male	NaN	8	2	69.55	S
863	0	3	female	NaN	8	2	69.55	S

In [14]:

```
1 plt.subplot(141)
2 plt.boxplot(df_titanic[df_titanic['Age'] > 0]['Age'])
3 plt.title('Age')
4 plt.subplot(142)
5 plt.boxplot(df_titanic['Parch'])
6 plt.title('Parch')
7 plt.subplot(143)
8 plt.boxplot(df_titanic['SibSp'])
9 plt.title('SibSp')
10 plt.subplot(144)
11 plt.boxplot(df_titanic['Fare'])
12 plt.title('Fare')
13 plt.tight_layout()
```



In [15]:

```
1 df_titanic = df_titanic.drop(multiple_outliers).reset_index(drop=True)
```

In [16]:

```
1 df_titanic.tail()
```

Out[16]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
876	0	2	male	27.0	0	0	13.00	S
877	1	1	female	19.0	0	0	30.00	S
878	0	3	female	Nan	1	2	23.45	S
879	1	1	male	26.0	0	0	30.00	C
880	0	3	male	32.0	0	0	7.75	Q

## Feature Analysis

- feature 간의 correlation check
- Survived 와 각 Feature 간 상관관계 파악

In [18]:

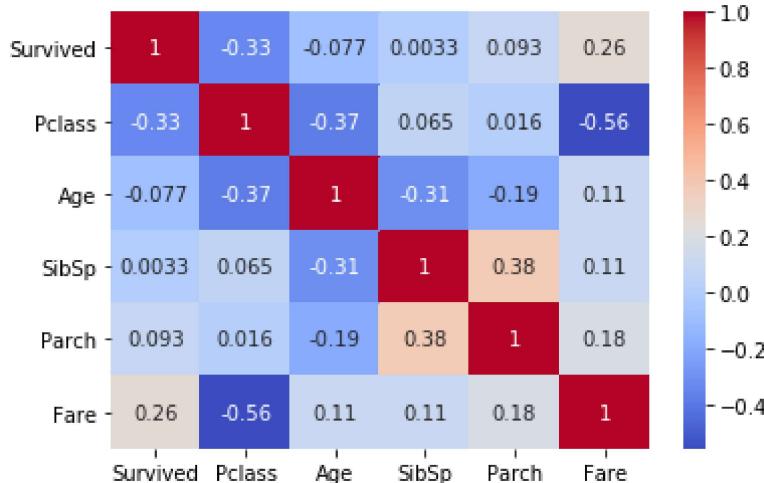
```
1 df_titanic.corr()
```

Out[18]:

	Survived	Pclass	Age	SibSp	Parch	Fare
Survived	1.000000	-0.334097	-0.076867	0.003330	0.092819	0.264613
Pclass	-0.334097	1.000000	-0.374495	0.065031	0.016440	-0.555558
Age	-0.076867	-0.374495	1.000000	-0.307129	-0.186457	0.110219
SibSp	0.003330	0.065031	-0.307129	1.000000	0.379281	0.110072
Parch	0.092819	0.016440	-0.186457	0.379281	1.000000	0.183801
Fare	0.264613	-0.555558	0.110219	0.110072	0.183801	1.000000

In [19]:

```
1 import seaborn as sns  
2  
3 g = sns.heatmap(df_titanic.corr(), annot=True, cmap="coolwarm")
```



가장 상관 관계가 높은 feature 는 Fare 이고 Pclass 는 가장 상관관계가 낮은 것으로 나타난다.

## Missing Value 처리

In [20]:

```
1 df_titanic.isnull().sum()
```

Out[20]:

```
Survived      0  
Pclass       0  
Sex          0  
Age         170  
SibSp        0  
Parch        0  
Fare         0  
Embarked     2  
dtype: int64
```

In [21]:

```
1 df_titanic.shape
```

Out[21]:

```
(881, 8)
```

881 개의 data 중 170 개를 drop 시키는 것은 너무 많은 양이므로 Age 는 median 으로 채워주고 Embarked 는 drop 시킨다.

In [22]:

```
1 df_titanic['Age'].fillna(df_titanic['Age'].median(), inplace=True)
2 df_titanic.isna().sum()
```

Out[22]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked   2
dtype: int64
```

In [23]:

```
1 df_titanic.dropna(inplace=True)
2 df_titanic.isna().sum()
```

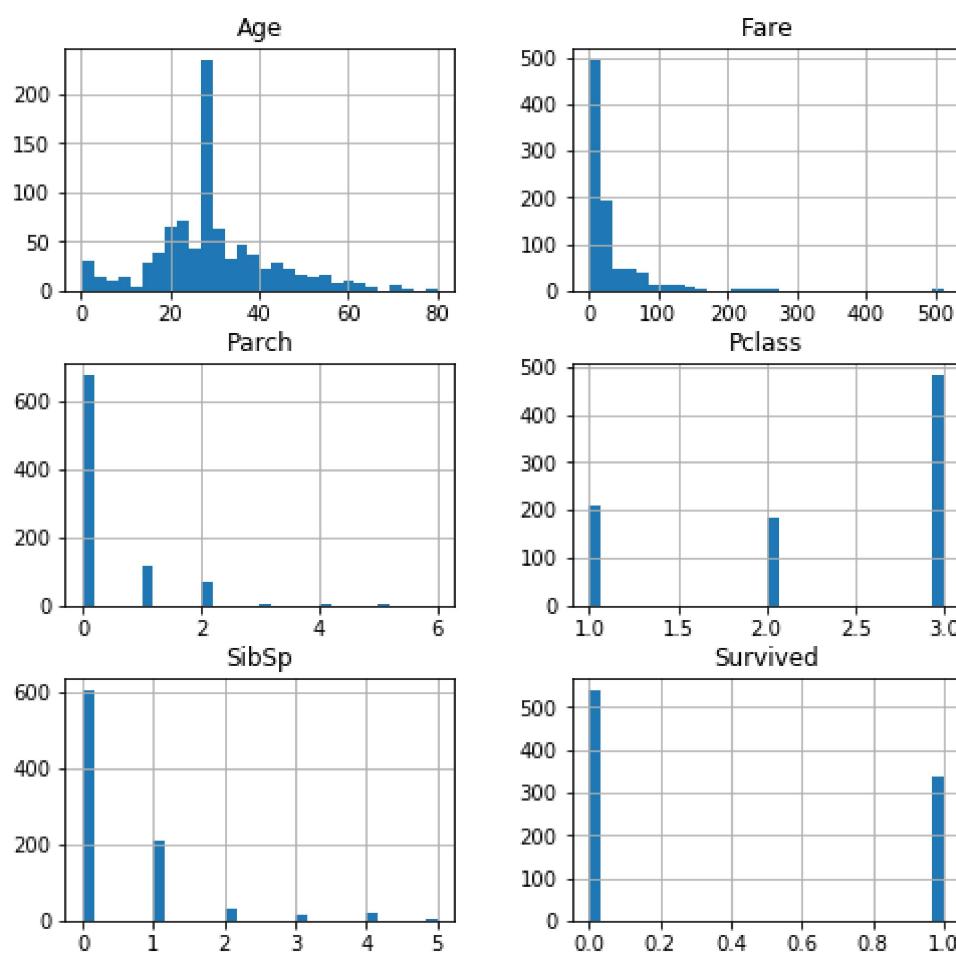
Out[23]:

```
Survived    0
Pclass      0
Sex         0
Age         0
SibSp       0
Parch       0
Fare        0
Embarked   0
dtype: int64
```

## Data 의 skewness check

In [24]:

```
1 df_titanic.hist(bins=30, figsize=(8, 8));
```



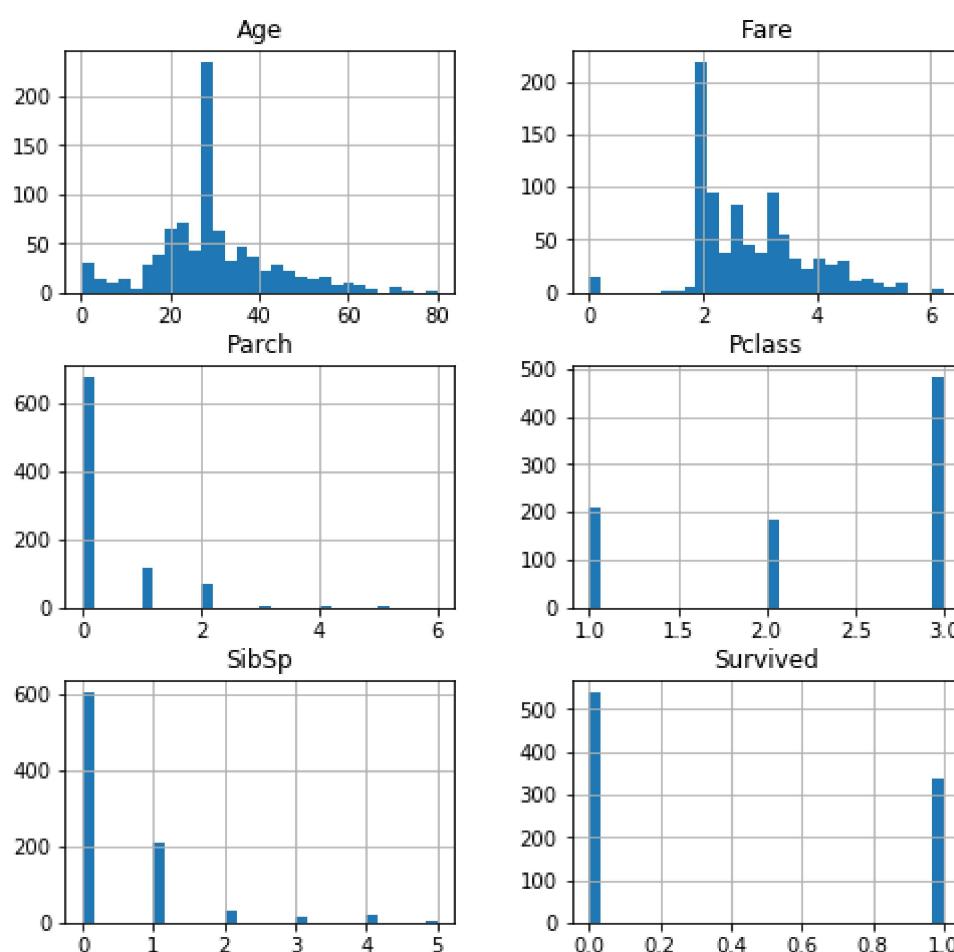
Fare 의 분포가 심하게 skew 되어 있으므로 log 값으로 바꾸어 skewness 를 완화 시킨다.

In [25]:

```
1 df_titanic['Fare'] = df_titanic['Fare'].map(lambda x: np.log(x) if x > 0 else 0)
```

In [26]:

```
1 df_titanic.hist(bins=30, figsize=(8, 8));
```



In [29]:

```
1 df_titanic.head()
```

Out[29]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked
0	0	3	22.0	1	0	1.981001	0	1	
1	1	1	38.0	1	0	4.266662	1	0	
2	1	3	26.0	0	0	2.070022	1	0	
3	1	1	35.0	1	0	3.972177	1	0	
4	0	3	35.0	0	0	2.085672	0	1	

## Category 변수 처리

Category column 들을 one-hot-encoding 으로 변환한다.

In [30]:

```
1 df_titanic = pd.get_dummies(df_titanic)
2 df_titanic.head()
```

Out[30]:

	Survived	Pclass	Age	SibSp	Parch	Fare	Sex_female	Sex_male	Embarked
0	0	3	22.0	1	0	1.981001	0	1	
1	1	1	38.0	1	0	4.266662	1	0	
2	1	3	26.0	0	0	2.070022	1	0	
3	1	1	35.0	1	0	3.972177	1	0	
4	0	3	35.0	0	0	2.085672	0	1	

## Train / Test dataset split

In [31]:

```
1 df_titanic.shape
```

Out[31]:

(879, 11)

In [32]:

```
1 X_train = df_titanic[:700]
2 X_test = df_titanic[700:]
```

In [33]:

```
1 y_train = X_train.pop('Survived')
2 y_test = X_test.pop('Survived')
```

## Standard Scaling

In [34]:

```
1 from sklearn.preprocessing import StandardScaler
```

In [35]:

```
1 sc = StandardScaler()
2 X_train_scaled = sc.fit_transform(X_train)
3 X_test_scaled = sc.transform(X_test)
```

## Modeling

- 어떤 algorithm 이 가장 적합한지 알지 못하므로 k-fold cross-validation 을 통하여 algorithm 비교. 일단은 default parameter 를 사용한다.

In [36]:

```
1 from sklearn.neighbors import KNeighborsClassifier
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.linear_model import LogisticRegression
4 from sklearn.svm import SVC
5 from sklearn.ensemble import RandomForestClassifier
6 from sklearn.ensemble import GradientBoostingClassifier
7 from sklearn.model_selection import KFold
8 from sklearn.model_selection import cross_val_score
9
10 n_fold = 10
```

In [37]:

```
1 models = []
2 models.append(('LR', LogisticRegression(solver='lbfgs', max_iter=300)))
3 models.append(('KNN', KNeighborsClassifier()))
4 models.append(('CART', DecisionTreeClassifier()))
5 models.append(('SVM', SVC(gamma='scale')))
6 models.append(('RMF', RandomForestClassifier(n_estimators=10)))
7 models.append(('GBM', GradientBoostingClassifier()))
```

In [38]:

```
1 results = []
2 names = []
3 for name, model in models:
4     kfold = KFold(n_splits=n_fold)
5     cv_results = cross_val_score(model, X_train_scaled, y_train, cv=kfold, scoring='accuracy')
6     results.append(cv_results)
7     names.append(name)
8     print("{}: {:.5f} ({:.5f})".format(name, cv_results.mean(), cv_results.std()))
```

LR: 0.78857 (0.02195)  
KNN: 0.80143 (0.03962)  
CART: 0.76714 (0.06702)  
SVM: 0.80429 (0.03384)  
RMF: 0.81143 (0.03429)  
GBM: 0.82571 (0.04324)

## Algorithm Parameter Tuning

- 가장 성능이 좋았던 KNN 과 SVC 및 GBM 에 대하여 GridSearchCV 를 이용하여 parameter tuning 을 한다.

In [39]:

```
1 from sklearn.model_selection import GridSearchCV
```

## Grid Search of KNN

In [41]:

```
1 neighbors = [1,3,5,7,9,11,13,15,17,19,21]
2 param_grid = dict(n_neighbors=neighbors)
3 print(param_grid)
4
5 grid = GridSearchCV(estimator = KNeighborsClassifier(),
6                      param_grid = param_grid, scoring='accuracy', cv=kfold)
7
8 grid_result = grid.fit(X_train_scaled, y_train)
9
10 print("Best param : {:.5f} using {}".format(grid_result.best_score_, grid_result.best_params_))
```

{'n\_neighbors': [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]}

Best param : 0.81571 using {'n\_neighbors': 11}

## Grid Search of SVC

In [42]:

```
1 c_values = [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0]
2 kernel_values = ['linear', 'poly', 'rbf', 'sigmoid']
3 param_grid = dict(C = c_values, kernel = kernel_values)
4 param_grid
```

Out[42]:

{'C': [0.1, 0.3, 0.5, 0.7, 0.9, 1.0, 1.3, 1.5, 1.7, 2.0],
'kernel': ['linear', 'poly', 'rbf', 'sigmoid']}

In [43]:

```
1 grid = GridSearchCV(estimator=SVC(gamma='scale'),
2                      param_grid=param_grid, scoring='accuracy', cv=kfold)
3
4 grid_result = grid.fit(X_train_scaled, y_train)
5
6 print("Best param : {:.5f} using {}".format(grid_result.best_score_, grid_result.best_params_))
```

Best param : 0.80571 using {'C': 0.5, 'kernel': 'rbf'}

## Grid Search of GBM

In [44]:

```
1 n_estimators = [100, 200, 300, 400, 500]
2 max_depth = [3, 5, 7, 9]
3 param_grid = dict(n_estimators = n_estimators, max_depth = max_depth)
4 param_grid
```

Out[44]:

```
{'n_estimators': [100, 200, 300, 400, 500], 'max_depth': [3, 5, 7, 9]}
```

In [45]:

```
1 grid = GridSearchCV(estimator=GradientBoostingClassifier(),
2                      param_grid=param_grid, scoring='accuracy', cv=kfold)
3
4 grid_result = grid.fit(X_train_scaled, y_train)
5
6 print("Best param : {:.5f} using {}".format(grid_result.best_score_, grid_result.best_params_))
```

Best param : 0.82571 using {'max\_depth': 3, 'n\_estimators': 200}

# 080.K-Means Clustering

Random하게 생성된 toy dataset으로 K-Means clustering test

make\_blobs :

Input :

n\_samples : cluster에 균등 분할될 total data point 숫자

centers : generate 할 centroid 숫자

cluster\_std : cluster의 standard deviation

Output :

X: 생성된 sample 들

y: 각 sample의 label

KMeans :

```
init : initialization method -> k-means++ (smart choosing of centroids)
n_clusters : k 값
n_init : 반복횟수
```

In [2]:

```
1 import random
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from sklearn.cluster import KMeans
5 from sklearn.datasets.samples_generator import make_blobs
6 %matplotlib inline
```

In [3]:

```
1 np.random.seed(0)
```

In [4]:

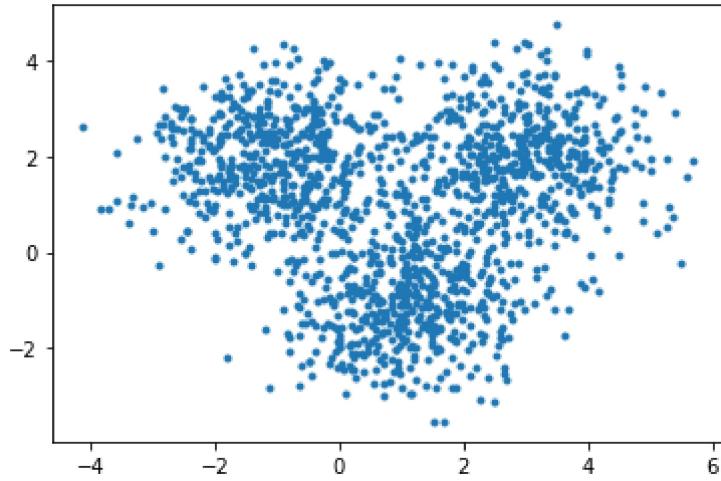
```
1 centroidLocation = [[3,2], [1,-1],[-1,2]]
2
3 X, _ = make_blobs(n_samples=1500, centers=centroidLocation)
```

In [5]:

```
1 plt.scatter(X[:,0], X[:,1], marker='.')  
In [5]:
```

Out[5]:

```
<matplotlib.collections.PathCollection at 0x27e9cb528c8>  
In [5]:
```



In [6]:

```
1 print(X.shape)  
2 print(X[:10])  
In [6]:
```

```
(1500, 2)  
[[ 2.71230522 -1.79211502]  
 [-0.17387429  1.94224344]  
 [ 1.4595849 -0.71994214]  
 [ 3.47043314  2.31144707]  
 [ 3.44819528  3.69618157]  
 [ 3.61407937  2.92220667]  
 [-1.14763741  3.13181196]  
 [ 3.06651722  2.3024719 ]  
 [ 4.8831507   0.65224094]  
 [-0.5033848   3.01213319]]  
In [6]:
```

In [7]:

```
1 nclusters = 3  
In [7]:
```

In [8]:

```
1 k_means = KMeans(n_clusters=nclusters)
```

In [9]:

```
1 k_means.fit(X)
```

Out[9]:

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,  
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',  
       random_state=None, tol=0.0001, verbose=0)
```

In [10]:

```
1 k_means.labels_
```

Out[10]:

```
array([2, 1, 2, ..., 0, 2, 0])
```

In [11]:

```
1 centers = k_means.cluster_centers_  
2 centers
```

Out[11]:

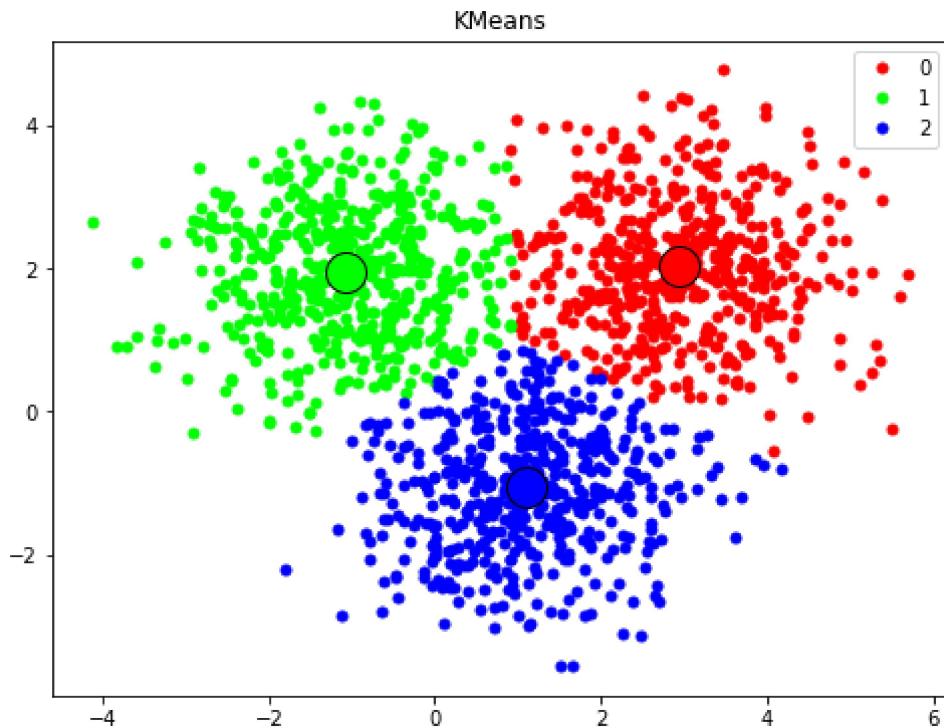
```
array([[ 2.94332137,  2.03105234],  
      [-1.07551635,  1.95077137],  
      [ 1.09377948, -1.06187746]])
```

In [12]:

```
1 from matplotlib.colors import ListedColormap
2
3 plt.figure(figsize=(8,6))
4
5 colors = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
6
7 for i in range(nclusters):
8     members = k_means.labels_ == i
9     plt.plot(X[members, 0], X[members, 1], '.', color=colors(i), markersize=10, label=i)
10    plt.plot(centers[i, 0], centers[i, 1], 'o', color=colors(i), markeredgecolor='k', markersize=2)
11
12 plt.title("KMeans")
13 plt.legend()
```

Out[12]:

<matplotlib.legend.Legend at 0x27e9cc2e6c8>



# 085.PCA (Principal Component Analysis)

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [2]:

```
churn_df = pd.read_csv("datasets/ChurnData.csv")
```

In [3]:

```
churn_df.head()
```

Out[3]:

	tenure	age	address	income	ed	employ	equip	callcard	wireless	longmor
0	11.0	33.0	7.0	136.0	5.0	5.0	0.0	1.0	1.0	4.40
1	33.0	33.0	12.0	33.0	2.0	0.0	0.0	0.0	0.0	9.44
2	23.0	30.0	9.0	30.0	1.0	2.0	0.0	0.0	0.0	6.30
3	38.0	35.0	5.0	76.0	2.0	10.0	1.0	1.0	1.0	6.04
4	7.0	35.0	14.0	80.0	2.0	15.0	0.0	1.0	0.0	7.10

5 rows × 28 columns



In [7]:

```
churn_df.cov()
```

Out[7]:

	tenure	age	address	income	ed	
<b>tenure</b>	468.331633	122.197663	100.318342	304.014422	-1.961432	8
<b>age</b>	122.197663	171.002789	99.173618	354.827688	-1.202136	7
<b>address</b>	100.318342	99.173618	103.193467	173.266834	-1.900754	4
<b>income</b>	304.014422	354.827688	173.266834	16494.385025	23.319347	39
<b>ed</b>	-1.961432	-1.202136	-1.900754	23.319347	1.652638	-
<b>employ</b>	86.408417	72.922487	47.400754	397.076131	-2.462940	8
<b>equip</b>	-1.255905	-0.462437	-0.750000	-0.683668	0.310930	-
<b>callcard</b>	4.220075	1.018769	0.971608	-1.172513	-0.041834	
<b>wireless</b>	-0.694925	-0.389799	-0.676884	-1.731357	0.156533	-
<b>longmon</b>	163.287617	48.297260	42.363304	53.088339	-0.924504	3
<b>tollmon</b>	33.927186	10.963970	-2.473367	-7.435879	-0.014322	1
<b>equipmon</b>	-18.314866	-7.775112	-24.875741	0.691636	12.394793	-2
<b>cardmon</b>	156.696062	39.277192	44.655088	-25.652399	-1.348964	2
<b>wiremon</b>	20.161975	-1.042518	-24.863618	-77.787397	8.025101	-
<b>longten</b>	12866.973639	3840.569926	3392.155289	4638.484977	-79.694340	257
<b>tollten</b>	7078.406746	2007.630917	1075.765704	-27.568819	-17.138128	176
<b>cardten</b>	13791.806338	4147.752996	3892.144284	2340.616796	-84.115672	260
<b>voice</b>	-0.501482	0.116910	-0.323367	5.021759	0.137312	
<b>pager</b>	0.182035	0.039824	-0.481156	3.275628	0.148869	
<b>internet</b>	-1.776080	-0.510151	-0.965829	6.570653	0.353769	-
<b>callwait</b>	-0.105302	0.130578	-0.101256	5.201859	-0.010427	
<b>confer</b>	0.872060	0.200101	-0.154774	-2.024925	-0.084925	
<b>ebill</b>	-1.067538	-0.314171	-0.870352	-2.645427	0.273367	-
<b>loglong</b>	13.679504	3.628259	3.040975	6.160576	-0.051312	
<b>logtoll</b>	1.885549	0.343964	0.052487	-5.648245	-0.002611	
<b>Ininc</b>	4.012093	3.083756	1.627789	65.752681	0.199988	
<b>custcat</b>	3.135804	0.579523	-0.184673	4.259548	0.018216	
<b>churn</b>	-3.710000	-1.711407	-1.204523	-5.304221	0.126382	-

28 rows × 28 columns

In [8]:

```
A = np.array(churn_df.values)
A.shape
```

Out[8]:

```
(200, 28)
```

In [9]:

```
columns = churn_df.columns[:-1]
columns.size
```

Out[9]:

```
27
```

In [10]:

```
X = np.asarray(churn_df.loc[:, columns], dtype=np.float32)
y = np.asarray(churn_df['churn'])
```

In [11]:

```
# train / test dataset split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

In [12]:

```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [13]:

```
print(X_train.shape)
print(X_test.shape)
```

```
(160, 27)
(40, 27)
```

## PCA 적용 전 Logistic Regression

- PCA 적용 후의 결과와 비교 목적

In [14]:

```
# Fitting Logistic Regression
clf = LogisticRegression(solver='lbfgs', random_state=0)
clf.fit(X_train, y_train)
```

Out[14]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [15]:

```
# predict test set
y_pred = clf.predict(X_test)
# Confusion matrix
accuracy_score(y_test, y_pred)
```

Out[15]:

0.775

## PCA 적용

- 27 개의 feature 를 2 개로 차원 축소
- components\_
  - array, shape (n\_components, n\_features)
  - n\_feature dimension 공간에서의 주성분 축
  - data 의 분산을 최대로 보존하는 방향
  - explainedvariance 에 순서대로 정렬되어 있음

In [16]:

```
# Apply kernel PCA
from sklearn.decomposition import PCA

pca = PCA(n_components=2)          # 2 개 component 로 차원 축소

X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
```

In [17]:

```
print("차원 축소된 X_train :", X_train_pca.shape)
print(X_train_pca[:5])
print()
print("차원 축소된 X_test :", X_test_pca.shape)
print(X_test_pca[:5])
```

차원 축소된 X\_train : (160, 2)  
[[ 6.1252427 -2.9101825 ]  
[-1.7907636 2.6611078 ]  
[-0.6123653 2.6992145 ]  
[ 0.11432906 -1.7805345 ]  
[ 3.1427593 4.7126474 ]]

차원 축소된 X\_test : (40, 2)  
[[ 0.517728 1.3013049 ]  
[-0.2538366 -2.9665475 ]  
[-4.058648 -0.6377735 ]  
[-0.5239167 3.185847 ]  
[ 0.64675385 1.1866788 ]]

In [18]:

```
print(pca.components_.shape)
print("첫번째 주성분 :")
print(pca.components_[0])
print('설명된 분산의 비율', pca.explained_variance_ratio_)
```

(2, 27)

첫번째 주성분 :

```
[0.18870449 0.09407663 0.06999469 0.02405294 0.08039848 0.10532499
 0.09130017 0.22012776 0.24022882 0.1703587 0.2678314 0.16389638
 0.25017655 0.28326163 0.1763821 0.2743571 0.22993164 0.24062625
 0.26112375 0.07475983 0.22729254 0.21587749 0.06959318 0.17457926
 0.17900118 0.07082605 0.29323992]
```

설명된 분산의 비율 [0.25193477 0.2176447 ]

In [21]:

```
# Fitting Logistic Regression
clf = LogisticRegression(solver='lbfgs', random_state=0)
clf.fit(X_train_pca, y_train)
```

Out[21]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=0, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [22]:

```
# predict test set
y_pred = clf.predict(X_test_pca)
y_pred
```

Out[22]:

```
array([0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 1., 0., 1., 1., 0., 1., 0., 0.,
       0., 0., 0., 0., 0., 0.])
```

In [23]:

```
# Confusion matrix
from sklearn.metrics import accuracy_score
accuracy_score(y_test, y_pred)
```

Out[23]:

0.725

In [24]:

```
sum(pca.explained_variance_ratio_)
```

Out[24]:

0.46957947313785553

## 차원 축소된 churn data 시각화

- 27 개의 feature 가 2 개의 PCA 로 차원 축소 되었으므로 평면상의 시각화 가능

In [25]:

```
X_train_pca.shape
```

Out[25]:

```
(160, 2)
```

In [26]:

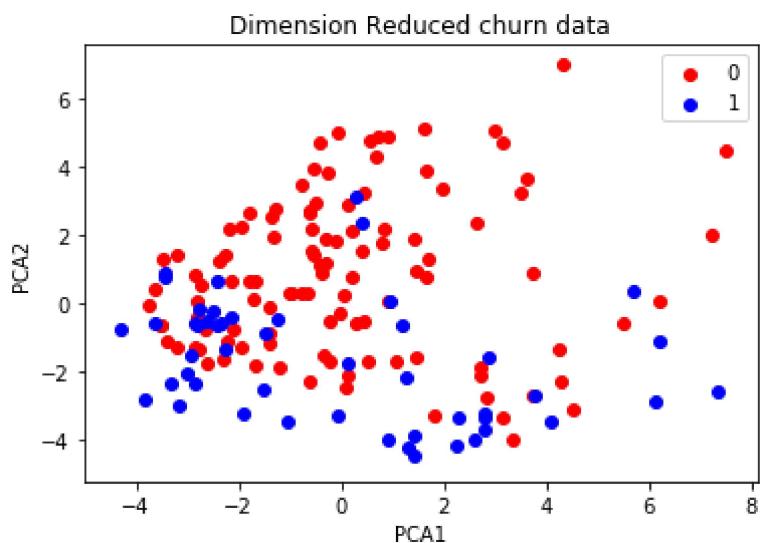
```
from matplotlib.colors import ListedColormap

cmap = ListedColormap(['r', 'b'])

X1, X2 = X_train_pca[y_train == 0, 0], X_train_pca[y_train == 0, 1]
plt.scatter(X1, X2, color=cmap(0), label=0)

X1, X2 = X_train_pca[y_train == 1, 0], X_train_pca[y_train == 1, 1]
plt.scatter(X1, X2, color=cmap(1), label=1)

plt.xlabel('PCA1')
plt.ylabel('PCA2')
plt.legend()
plt.title('Dimension Reduced churn data');
plt.show()
```



# 090.Introduction to Tensorflow 2.0 and Keras

In [1]:

```
import tensorflow as tf
import numpy as np
```

## version check

In [2]:

```
print(tf.__version__)
print(tf.version.VERSION)
print(tf.keras.__version__)
```

2.0.0  
2.0.0  
2.2.4-tf

## GPU check

In [3]:

```
from tensorflow.python.client import device_lib

print(device_lib.list_local_devices())
print('*****')
print("GPU Available: ", tf.test.is_gpu_available())
```

[name: "/device:CPU:0"
device\_type: "CPU"
memory\_limit: 268435456
locality {}
}
incarnation: 14491299742170962191
, name: "/device:GPU:0"
device\_type: "GPU"
memory\_limit: 3186409472
locality {}
bus\_id: 1
links {}
}
incarnation: 3506417215696966542
physical\_device\_desc: "device: 0, name: Quadro M1200, pci bus id: 0000:01:00.0, compute capability: 5.0"
]
\*\*\*\*\*
GPU Available: True

# Introduction to keras

## High Level API

In [4]:

```
from tensorflow.keras.layers import Dense, Activation

# load mnist data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train = X_train.reshape(-1, 784).astype("float32")
X_test = X_test.reshape(-1, 784).astype("float32")

X_train /= 255.
X_test /= 255.

Y_train = tf.keras.utils.to_categorical(y_train)
Y_test = tf.keras.utils.to_categorical(y_test)
```

In [5]:

```
EPOCHS = 2
BATCH_SIZE = 512

# model 1 : No Hidden Layer
model = tf.keras.models.Sequential()
model.add(Dense(120, input_shape=(784,), activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(10 , activation='softmax'))

model.summary()
```

Model: "sequential"

---

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 120)	94200
dense_1 (Dense)	(None, 64)	7744
dense_2 (Dense)	(None, 10)	650

---

Total params: 102,594  
Trainable params: 102,594  
Non-trainable params: 0

---

In [6]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
```

In [7]:

```
history = model.fit(X_train, Y_train, batch_size=BATCH_SIZE, epochs=EPOCHS, verbose=2, validation_data=(X_test, Y_test))

score = model.evaluate(X_test, Y_test, verbose=0)

print("Test score :", score[0])      # evaluation of loss function
print("Test accuracy :", score[1])
```

Train on 60000 samples, validate on 10000 samples  
Epoch 1/2  
60000/60000 - 2s - loss: 0.5718 - accuracy: 0.8431 - val\_loss: 0.2388 - val\_accuracy: 0.9346  
Epoch 2/2  
60000/60000 - 1s - loss: 0.2068 - accuracy: 0.9410 - val\_loss: 0.1811 - val\_accuracy: 0.9459  
Test score : 0.18114189576208592  
Test accuracy : 0.9459

# 100.Boston House Price Regression

13 개의 종속변수와 1 개의 독립변수 (주택가격 중앙값) 으로 구성

## 종속변수 (13 개)

CRIM 자치시(town) 별 1인당 범죄율  
ZN 25,000 평방피트를 초과하는 거주지역의 비율  
INDUS 비소매상업지역이 점유하고 있는 토지의 비율  
CHAS 찰스강에 대한 더미변수(강의 경계에 위치한 경우는 1, 아니면 0)  
NOX 10ppm 당 농축 일산화질소  
RM 주택 1가구당 평균 방의 개수  
AGE 1940년 이전에 건축된 소유주택의 비율  
DIS 5개의 보스턴 직업센터까지의 접근성 지수  
RAD 방사형 도로까지의 접근성 지수  
TAX 10,000 달러 당 재산세율  
PTRATIO 자치시(town)별 학생/교사 비율  
B 1000(Bk-0.63)^2, 여기서 Bk는 자치시별 흑인의 비율을 말함  
LSTAT 모집단의 하위계층의 비율(%)

## 독립변수 (1 개)

MEDV 본인 소유의 주택가격(중앙값) (단위: \$1,000)

In [1]:

```
from sklearn.datasets import load_boston
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [2]:

```
boston = load_boston()
df_boston = pd.DataFrame(data=boston.data, columns=boston.feature_names)
df_boston.head()
```

Out[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7

◀ ▶

In [3]:

```
X = boston.data
y = boston.target
```

In [4]:

```
y[:10]
```

Out[4]:

```
array([24. , 21.6, 34.7, 33.4, 36.2, 28.7, 22.9, 27.1, 16.5, 18.9])
```

In [5]:

```
print(X.shape)
print(y.shape)
```

```
(506, 13)
(506,)
```

In [6]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [7]:

```
sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

**model build**

In [8]:

```
model = Sequential()  
  
model.add(Dense(64, input_shape=(13,)))  
model.add(Activation('relu'))  
model.add(Dense(64))  
model.add(Activation('relu'))  
model.add(Dense(1))
```

In [9]:

```
model.compile(loss='mean_squared_error', optimizer='adam', metrics=['mae', 'mse'])
```

epoch 수와 batch\_size 변경하며 test

In [29]:

```
history = model.fit(X_train, y_train, batch_size=32, epochs=5, validation_data=(X_test, y_test), verbose=1)
```

Train on 379 samples, validate on 127 samples

Epoch 1/5

```
379/379 [=====] - 0s 211us/sample -  
loss: 8.9008 - mae: 2.0802 - mse: 8.9008 - val_loss: 8.2485 - val_mae: 2.0228  
- val_mse: 8.2485
```

Epoch 2/5

```
379/379 [=====] - 0s 206us/sample -  
loss: 8.7863 - mae: 2.0636 - mse: 8.7863 - val_loss: 8.4611 - val_mae: 2.0284  
- val_mse: 8.4611
```

Epoch 3/5

```
379/379 [=====] - 0s 206us/sample -  
loss: 8.7183 - mae: 2.0440 - mse: 8.7183 - val_loss: 8.8901 - val_mae: 2.0525  
- val_mse: 8.8901
```

Epoch 4/5

```
379/379 [=====] - 0s 195us/sample -  
loss: 8.8146 - mae: 2.0514 - mse: 8.8146 - val_loss: 8.2381 - val_mae: 2.0195  
- val_mse: 8.2381
```

Epoch 5/5

```
379/379 [=====] - 0s 169us/sample -  
loss: 8.7074 - mae: 2.0514 - mse: 8.7074 - val_loss: 8.2015 - val_mae: 2.0040  
- val_mse: 8.2015
```

In [30]:

```
model.evaluate(X_test, y_test, verbose=0)
```

Out[30]:

```
[8.201463751905546, 2.0040097, 8.201463]
```

In [31]:

```
y_pred = model.predict(X_test)
```

In [32]:

```
from sklearn.metrics import mean_squared_error, r2_score  
  
print(mean_squared_error(y_test, y_pred))  
print(r2_score(y_test, y_pred))
```

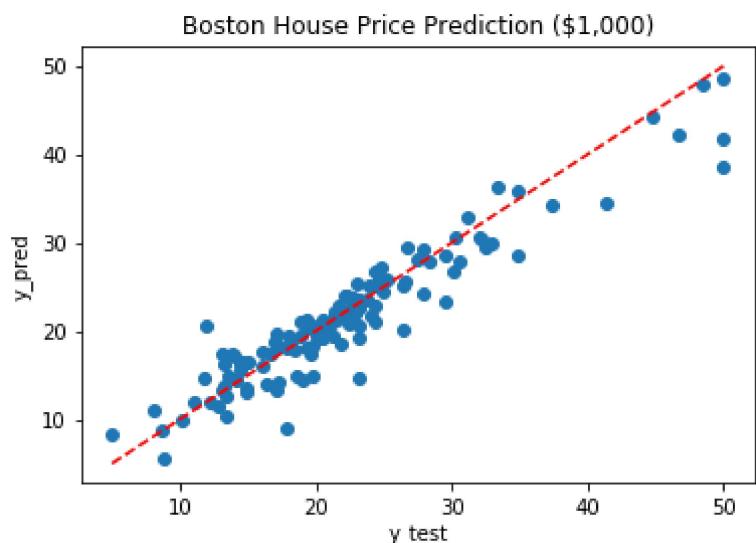
8.201463476911298  
0.8886062606599112

In [33]:

```
plt.scatter(y_test, y_pred)  
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', c='r')  
plt.xlabel('y_test')  
plt.ylabel('y_pred')  
plt.title('Boston House Price Prediction ($1,000)')
```

Out[33]:

Text(0.5, 1.0, 'Boston House Price Prediction (\$1,000)')

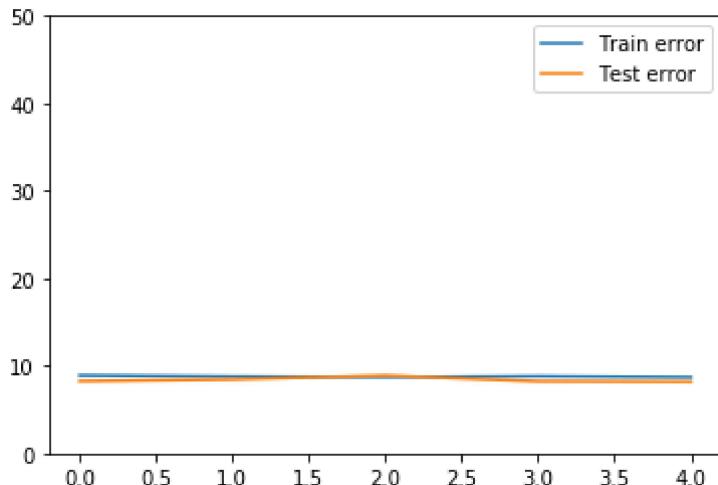


In [34]:

```
plt.plot(history.history['mse'], label='Train error')
plt.plot(history.history['val_mse'], label='Test error')
plt.ylim([0, 50])
plt.legend()
```

Out[34]:

<matplotlib.legend.Legend at 0x18d10ab2e48>



## sklearn LinearRegression 비교

In [35]:

```
from sklearn.linear_model import LinearRegression
regr = LinearRegression()

X_train, X_test, y_train, y_test = train_test_split(X, y)

sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [36]:

```
regr.fit(X_train, y_train)
```

Out[36]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

In [37]:

```
y_pred = regr.predict(X_test)
```

In [38]:

```
# The coefficients
print('Coefficients: \n', regr.coef_)
print('Intercept: \n', regr.intercept_)
```

Coefficients:

```
[ -9.17763837  5.1017426  1.74812926  2.82026969 -7.62046073
 20.03233904  1.03040482 -14.81706029  5.99193742 -5.07751023
 -9.13728095  4.30316225 -19.86059204]
```

Intercept:

```
25.639799558258552
```

In [39]:

```
# MSE(mean squared error) 계산
print("Mean squared error: {:.2f}".format(mean_squared_error(y_test, y_pred)))

# R2 계산
print("Variance score: {:.2f}".format(r2_score(y_test, y_pred)))
```

Mean squared error: 20.52

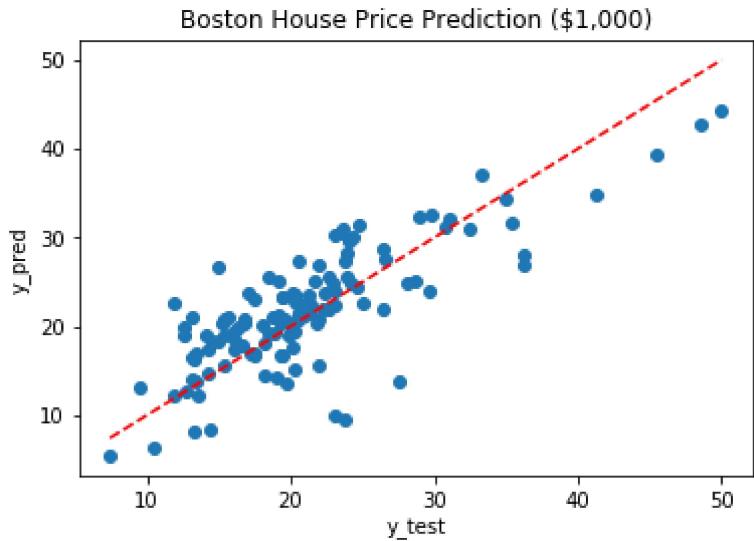
Variance score: 0.61

In [40]:

```
plt.scatter(y_test, y_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', c='r')
plt.xlabel('y_test')
plt.ylabel('y_pred')
plt.title('Boston House Price Prediction ($1,000)')
```

Out[40]:

Text(0.5, 1.0, 'Boston House Price Prediction (\$1,000)')



# 실습

## UCI Machine Learning Repository 의 Auto MPG dataset 을 사용하여 Regression 예측 model 작성

auto-mpg.data - data file

auto-mpg.names - data 설명 file

1. mpg: continuous
2. cylinders: multi-valued discrete
3. displacement: continuous (배기량)
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete
8. origin: multi-valued discrete, 1 - USA, 2 - Europe, 3 - Japan
9. car name: string (unique for each instance)

Missing Attribute Values: horsepower has 6 missing values ==> "?" 로 들어 있으므로 read\_csv 시 nan 으로 변환

In [23]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

## Data load 및 Preprocessing

In [24]:

```
data_path = keras.utils.get_file("auto-mpg.data",
                                 "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
                                 auto-mpg.data")

column_names = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight', 'acceleration', 'model
year', 'origin']

rawdata = pd.read_csv(data_path, names=column_names, na_values=?, comment=\t, sep=
= " ", skipinitialspace=True)

rawdata.dropna(inplace=True)

data = rawdata.copy()

data = pd.get_dummies(data, columns=['cylinders', 'origin'])

label = data.pop('mpg')

X_train, X_test, y_train, y_test = train_test_split(data.values, label.values)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

## Regression Model Build

**train**

**predict**

$r^2$  계산

# 110.mnist dataset 손글씨 인식 - Basic Fully Connected Layer

In [1]:

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

import tensorflow as tf
from tensorflow.keras import layers

np.random.seed(102)
```

In [2]:

```
# load mnist data
(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
```

In [3]:

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

In [4]:

```
X_train[0][15]
```

Out[4]:

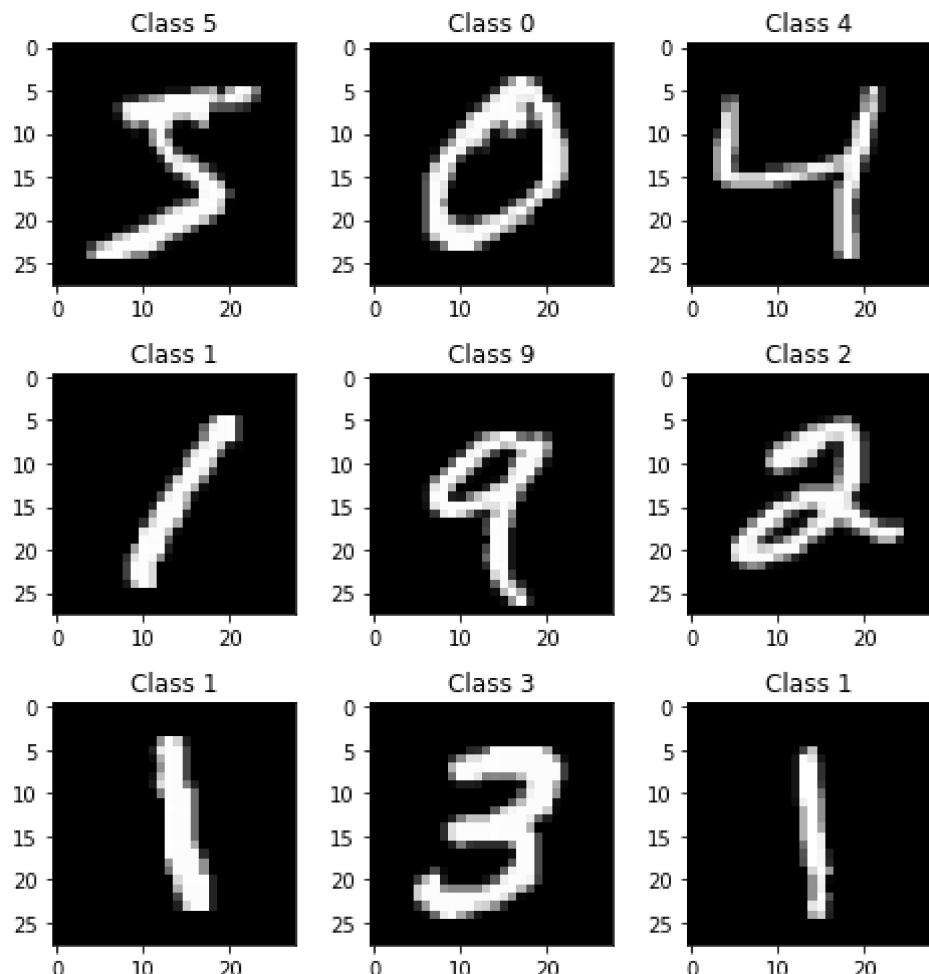
```
array([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
       0, 45, 186, 253, 253, 150, 27,  0,  0,  0,  0,  0,
       0,  0], dtype=uint8)
```

In [27]:

```
plt.figure(figsize=(7 ,7))

for i in range(9):
    plt.subplot(3, 3, i+1)
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.title("Class {}".format(y_train[i]))
    plt.tight_layout()

plt.show()
```



Dense layer 의 input 으로 만들기 위해  $28 \times 28$  을 784 로 flatten

In [6]:

```
X_train_scaled = X_train / 255.  
X_test_scaled = X_test / 255.  
  
X_train_scaled = X_train.reshape(-1, 784).astype("float32")  
X_test_scaled = X_test.reshape(-1, 784).astype("float32")  
  
print(X_train_scaled.shape)  
print(X_test_scaled.shape)  
print(y_train.shape)  
print(y_test.shape)
```

```
(60000, 784)  
(10000, 784)  
(60000,)  
(10000,)
```

## Target label 을 one-hot format, i.e.

```
0 -> [1, 0, 0, 0, 0, 0, 0, 0, 0]  
1 -> [0, 1, 0, 0, 0, 0, 0, 0, 0]  
2 -> [0, 0, 1, 0, 0, 0, 0, 0, 0]  
etc.
```

In [7]:

```
y_train_onehot = tf.keras.utils.to_categorical(y_train)  
y_test_onehot = tf.keras.utils.to_categorical(y_test)  
  
print(y_train_onehot.shape)  
print(y_test_onehot.shape)
```

```
(60000, 10)  
(10000, 10)
```

## tf.data 를 이용한 shuffling and batch 구성

In [8]:

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train_scaled, y_train_onehot)).shuffle(10000).batch(32)  
test_ds = tf.data.Dataset.from_tensor_slices((X_test_scaled, y_test_onehot)).batch(32)
```

In [9]:

```
for train, data in train_ds.take(1):  
    print(train.shape, data.shape)
```

```
(32, 784) (32, 10)
```

In [10]:

```
N_NEURONS = 128  
DROPOUT = 0.2  
EPOCHS = 5
```

In [11]:

```
# 2 : Hidden Layers  
model = tf.keras.Sequential()  
model.add(layers.Dense(N_NEURONS, input_shape=(784,)))  
model.add(layers.Activation('relu'))  
model.add(layers.Dropout(DROPOUT))  
  
model.add(layers.Dense(N_NEURONS))  
model.add(layers.Activation('relu'))  
model.add(layers.Dropout(DROPOUT))  
  
model.add(layers.Dense(10))  
model.add(layers.Activation('softmax'))
```

In [12]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	100480
activation (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16512
activation_1 (Activation)	(None, 128)	0
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290
activation_2 (Activation)	(None, 10)	0

Total params: 118,282  
Trainable params: 118,282  
Non-trainable params: 0

In [13]:

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=["accuracy"])
```

In [14]:

```
history = model.fit(train_ds, epochs=EPOCHS, verbose=1, validation_data=test_ds)
```

Epoch 1/5

```
1875/1875 [=====] - 12s 7ms/step - loss: 2.0403 - accuracy: 0.7061 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
```

Epoch 2/5

```
1875/1875 [=====] - 8s 4ms/step - loss: 0.6908 - accuracy: 0.8357 - val_loss: 0.3672 - val_accuracy: 0.9193
```

Epoch 3/5

```
1875/1875 [=====] - 9s 5ms/step - loss: 0.4749 - accuracy: 0.8834 - val_loss: 0.2844 - val_accuracy: 0.9357
```

Epoch 4/5

```
1875/1875 [=====] - 8s 4ms/step - loss: 0.3833 - accuracy: 0.9014 - val_loss: 0.2701 - val_accuracy: 0.9375
```

Epoch 5/5

```
1875/1875 [=====] - 8s 4ms/step - loss: 0.3455 - accuracy: 0.9101 - val_loss: 0.2556 - val_accuracy: 0.9375
```

In [15]:

```
score = model.evaluate(test_ds, verbose=0)
```

In [16]:

```
print(model.metrics_names)
print("Test score : {:.2f}".format(score[0]))      # evaluation of loss function
print("Test accuracy :", score[1])
```

['loss', 'accuracy']

Test score : 0.26

Test accuracy : 0.9375

In [17]:

```
# according to the trained classifier for each input example.
predicted_classes = model.predict_classes(X_test_scaled)
```

In [18]:

```
(predicted_classes == y_test).shape
```

Out[18]:

(10000,)

In [19]:

```
correct_indices = np.nonzero(predicted_classes == y_test)[0]
incorrect_indices = np.nonzero(predicted_classes != y_test)[0]
```

In [20]:

```
print(correct_indices.shape)
print(incorrect_indices.shape)
```

(9375,)
(625,)

In [21]:

```
correct_indices[:9]
```

Out[21]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 9], dtype=int64)
```

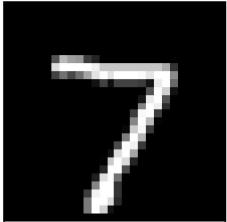
In [22]:

```
plt.figure(figsize=(6,6))

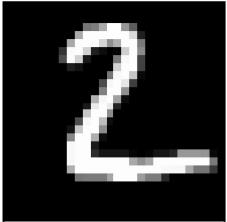
for i, idx in enumerate(correct_indices[:9]):
    plt.subplot(3, 3, i+1)
    plt.imshow(X_test[idx], cmap='gray')
    plt.title("Predicted {}, True {}".format(predicted_classes[idx], y_test[idx]))
    plt.xticks([])
    plt.yticks([])
    plt.tight_layout()

plt.figure(figsize=(6,6))
for i, idx in enumerate(incorrect_indices[:9]):
    plt.subplot(3,3,i+1)
    plt.imshow(X_test[idx], cmap='gray')
    plt.title("Predicted {}, True {}".format(predicted_classes[idx], y_test[idx]))
    plt.xticks([])
    plt.yticks([])
    plt.tight_layout()
```

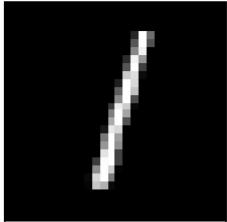
Predicted 7, True 7



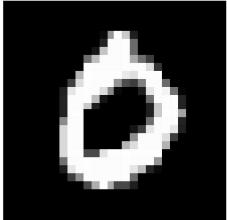
Predicted 2, True 2



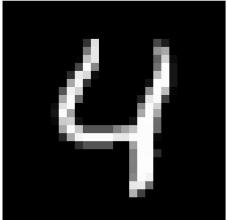
Predicted 1, True 1



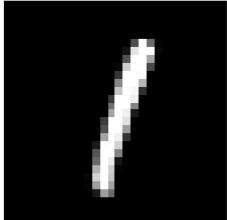
Predicted 0, True 0



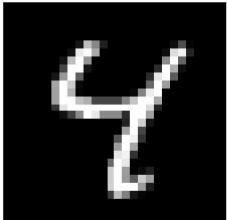
Predicted 4, True 4



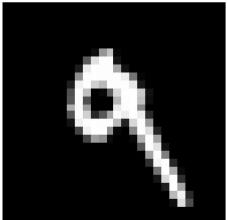
Predicted 1, True 1



Predicted 4, True 4

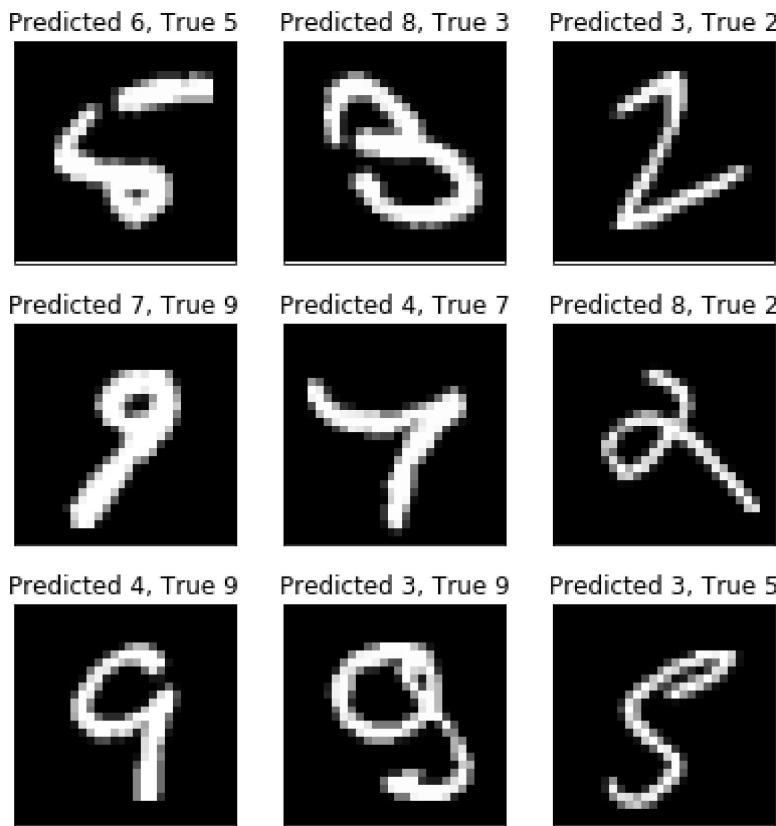


Predicted 9, True 9



Predicted 9, True 9

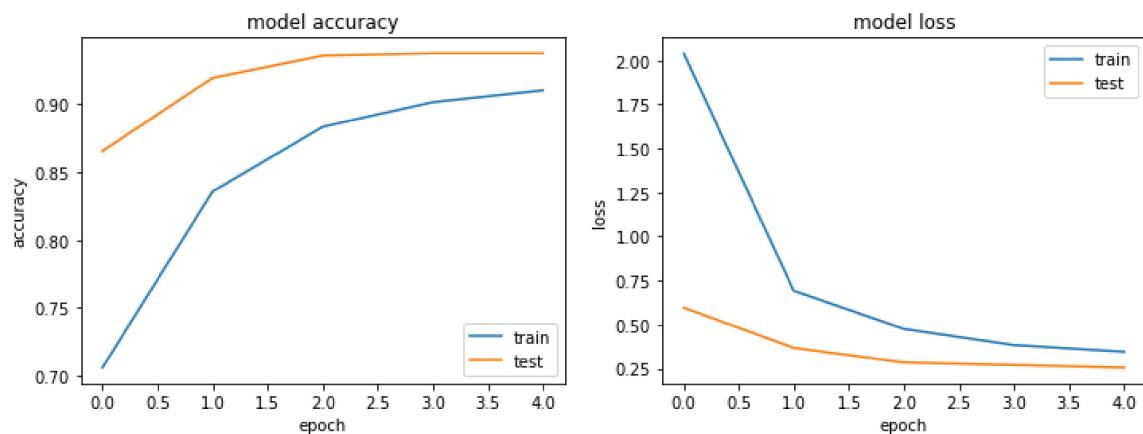




In [23]:

```
plt.figure(figsize=(12,4))
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train', 'test'])

plt.subplot(1,2,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test']);
```



In [24]:

```
np.argmax(model.predict(X_test[5].reshape((1,784))))
```

Out[24]:

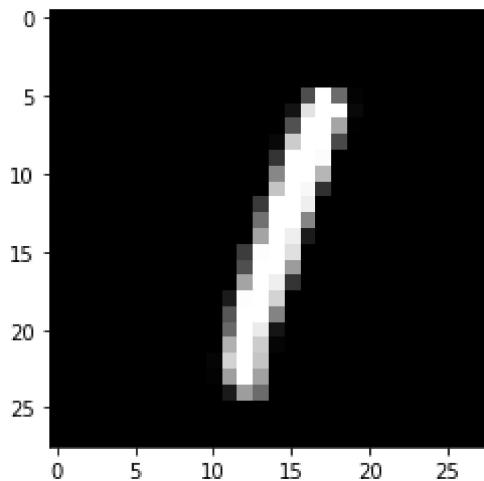
1

In [25]:

```
plt.imshow(X_test[5], cmap='gray')
```

Out[25]:

<matplotlib.image.AxesImage at 0x21f2262a188>



## 실습 : fashion MNIST 를 이용하여 위와 동일한 작업

Label Class

0 T-shirt/top

1 Trouser

2 Pullover

3 Dress

4 Coat

5 Sandal

6 Shirt

7 Sneaker

8 Bag

9 Ankle boot

In [26]:

```
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.fashion_mnist.load_data()  
  
print(train_images.shape)  
print(train_labels.shape)  
print(test_images.shape)  
print(test_labels.shape)
```

```
(60000, 28, 28)  
(60000,)  
(10000, 28, 28)  
(10000,)
```

In [27]:

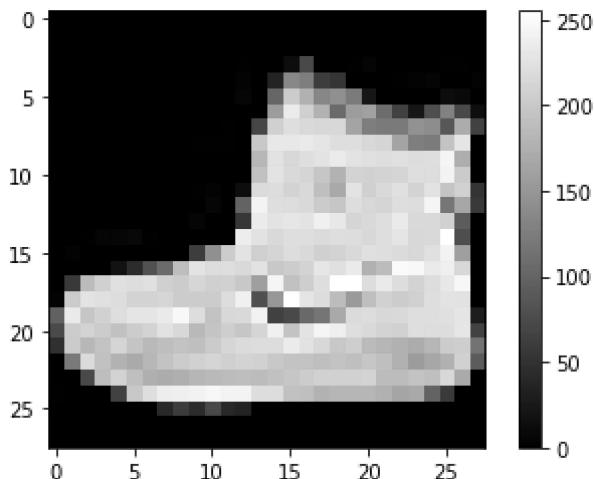
```
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat', 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
```

In [28]:

```
plt.imshow(train_images[0], cmap='gray')  
plt.colorbar()
```

Out[28]:

```
<matplotlib.colorbar.Colorbar at 0x238a0175448>
```



In [29]:

```
train_images = train_images / 255.0  
test_images = test_images / 255.0
```

In [30]:

```
plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i]])
    plt.gca().set_xticks([])
    plt.gca().set_yticks([])
```



In [ ]:

```
# Your code here
```

# 120.LeNet-5 (1998, Yan LeCunn)

In [2]:

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (7,7)
import tensorflow as tf

from tensorflow.keras.layers import Conv2D, AveragePooling2D
from tensorflow.keras.layers import Dense, Flatten, Activation

from tensorflow.keras.datasets import mnist
np.random.seed(101)
```

In [3]:

```
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 28, 28)
(60000,)
(10000, 28, 28)
(10000,)
```

In [4]:

```
X_train_scaled = X_train / 255.
X_test_scaled = X_test / 255.
```

In [5]:

```
X_train_scaled = np.expand_dims(X_train_scaled, axis=3)
X_test_scaled = np.expand_dims(X_test_scaled, axis=3)
```

In [6]:

```
X_train_scaled.shape
X_test_scaled.shape
```

Out[6]:

```
(10000, 28, 28, 1)
```

In [7]:

```
y_train_onehot = tf.keras.utils.to_categorical(y_train)
y_test_onehot = tf.keras.utils.to_categorical(y_test)
```

**tf.data 를 이용한 shuffling and batch 구성**

In [8]:

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train_scaled, y_train_onehot)).shuffle(10000).batch(128)
test_ds = tf.data.Dataset.from_tensor_slices((X_test_scaled, y_test_onehot)).batch(128)
```

## LeNet 구성

In [9]:

```
# LeNet
model = tf.keras.Sequential()

model.add(Conv2D(6, kernel_size=5, padding="same", input_shape=(28, 28, 1)))
model.add(Activation("relu"))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"))

model.add(Conv2D(16, kernel_size=5, padding="valid"))
model.add(Activation('relu'))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation("relu"))

model.add(Dense(84))
model.add(Activation("relu"))

model.add(Dense(10))
model.add(Activation("softmax"))
```

In [10]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 6)	156
activation (Activation)	(None, 28, 28, 6)	0
average_pooling2d (AveragePo	(None, 14, 14, 6)	0
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
activation_1 (Activation)	(None, 10, 10, 16)	0
average_pooling2d_1 (Average	(None, 5, 5, 16)	0
flatten (Flatten)	(None, 400)	0
dense (Dense)	(None, 120)	48120
activation_2 (Activation)	(None, 120)	0
dense_1 (Dense)	(None, 84)	10164
activation_3 (Activation)	(None, 84)	0
dense_2 (Dense)	(None, 10)	850
activation_4 (Activation)	(None, 10)	0

Total params: 61,706

Trainable params: 61,706

Non-trainable params: 0

In [11]:

```
model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy'])
```

In [12]:

```
history = model.fit(train_ds, epochs=5, validation_data=test_ds)
```

```
Epoch 1/5  
469/469 [=====] - 8s 16ms/step - loss: 0.4016 - accuracy: 0.8819 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00  
Epoch 2/5  
469/469 [=====] - 5s 10ms/step - loss: 0.1117 - accuracy: 0.9660 - val_loss: 0.0712 - val_accuracy: 0.9777  
Epoch 3/5  
469/469 [=====] - 5s 10ms/step - loss: 0.0756 - accuracy: 0.9764 - val_loss: 0.0785 - val_accuracy: 0.9745  
Epoch 4/5  
469/469 [=====] - 5s 10ms/step - loss: 0.0608 - accuracy: 0.9806 - val_loss: 0.0489 - val_accuracy: 0.9858  
Epoch 5/5  
469/469 [=====] - 5s 10ms/step - loss: 0.0497 - accuracy: 0.9849 - val_loss: 0.0483 - val_accuracy: 0.9842
```

In [13]:

```
score = model.evaluate(test_ds, verbose=0)
```

In [14]:

```
print("Test Score ", score[0])  
print("Test Accuracy ", score[1])
```

```
Test Score 0.048334635669274165  
Test Accuracy 0.9842
```

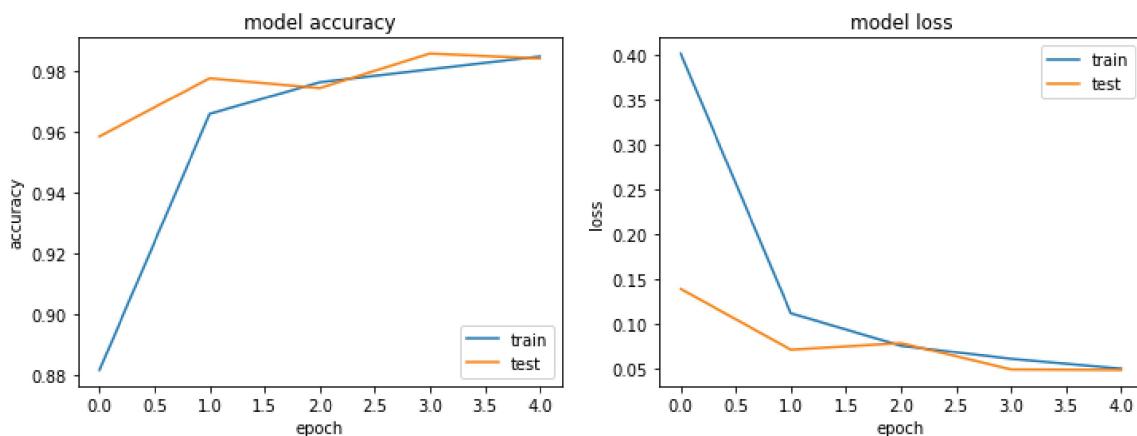
In [15]:

```
plt.figure(figsize=(12,4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.xlabel('epoch')
plt.ylabel('accuracy')
plt.legend(['train', 'test'])

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.xlabel('epoch')
plt.ylabel('loss')
plt.legend(['train', 'test'])
```

Out[15]:

<matplotlib.legend.Legend at 0x213a9047108>



In [16]:

```
y_pred = model.predict_classes(X_test_scaled)
y_pred
```

Out[16]:

array([7, 2, 1, ..., 4, 5, 6], dtype=int64)

In [17]:

```
y_test
```

Out[17]:

array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)

In [18]:

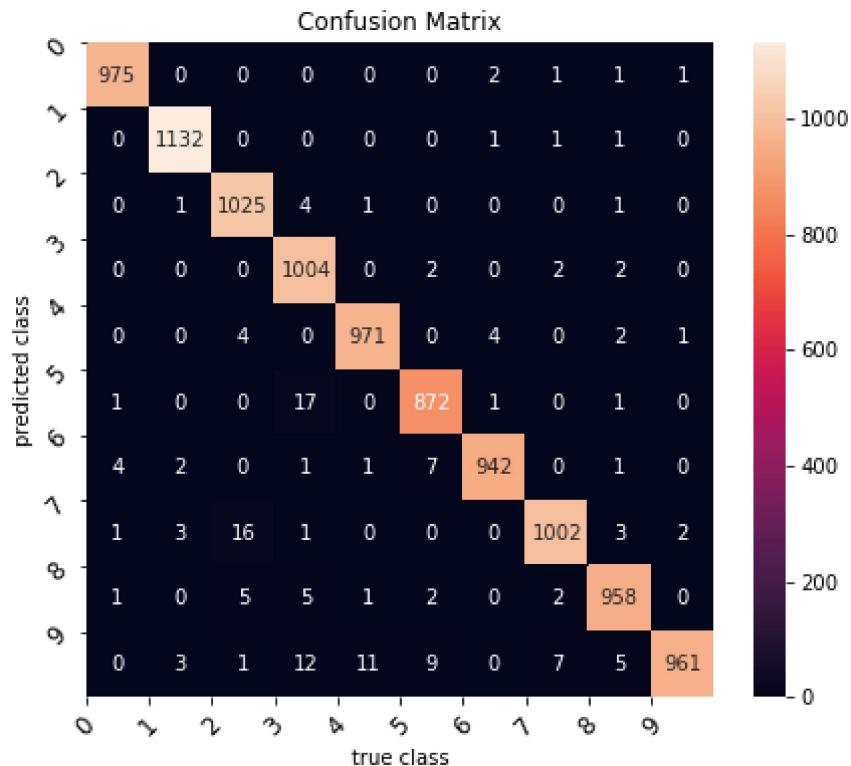
```
from sklearn.metrics import confusion_matrix, accuracy_score  
  
print(confusion_matrix(y_test, y_pred))
```

```
[[ 975  0  0  0  0  0  2  1  1  1]  
[ 0 1132  0  0  0  0  1  1  1  0]  
[ 0  1 1025  4  1  0  0  0  1  0]  
[ 0  0  0 1004  0  2  0  2  2  0]  
[ 0  0  4  0 971  0  4  0  2  1]  
[ 1  0  0  17  0 872  1  0  1  0]  
[ 4  2  0  1  1  7 942  0  1  0]  
[ 1  3  16  1  0  0  0 1002  3  2]  
[ 1  0  5  5  1  2  0  2 958  0]  
[ 0  3  1 12  11  9  0  7  5 961]]
```

In [22]:

```
import seaborn as sns  
  
plt.figure(figsize=(7,6))  
  
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')  
  
plt.xticks(np.arange(10), list(range(10)), rotation=45, fontsize=12)  
plt.yticks(np.arange(10), list(range(10)), rotation=45, fontsize=12)  
plt.xlabel("true class")  
plt.ylabel("predicted class")  
plt.title('Confusion Matrix')  
print('Test Accuracy :', accuracy_score(y_test, y_pred))
```

Test Accuracy : 0.9842



# 130.CIFAR-10 을 이용한 CNN 구축

In [1]:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras import utils
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
```

In [2]:

```
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
```

In [3]:

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(50000, 32, 32, 3)
(10000, 32, 32, 3)
(50000, 1)
(10000, 1)
```

In [4]:

```
cifa10_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

In [5]:

```
fig, axes = plt.subplots(2, 8, figsize=(15, 4))
axes = axes.ravel()
for i in range(16):
    idx = np.random.randint(0, len(y_train))
    axes[i].imshow(X_train[idx, :])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_title(cifa10_classes[y_train[idx, 0]])
```



In [6]:

```
# data normalization
X_train_scaled = X_train / 255.
X_test_scaled = X_test / 255.
```

In [7]:

```
# one-hot encoding of class labels
y_train_onehot = utils.to_categorical(y_train)
y_test_onehot = utils.to_categorical(y_test)
print(y_train_onehot.shape)
print(y_test_onehot.shape)
```

```
(50000, 10)
(10000, 10)
```

In [8]:

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train_scaled, y_train_onehot)).shuffle(10000).batch(64)
test_ds = tf.data.Dataset.from_tensor_slices((X_test_scaled, y_test_onehot)).batch(64)
```

In [9]:

```
# model build
model = Sequential()

model.add(Conv2D(16, (3, 3), padding='same', input_shape=(32, 32, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D((2,2)))
model.add(Dropout(0.2))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
```

In [10]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
conv2d (Conv2D)		
	(None, 32, 32, 16)	448
<hr/>		
activation (Activation)	(None, 32, 32, 16)	0
<hr/>		
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
<hr/>		
dropout (Dropout)	(None, 16, 16, 16)	0
<hr/>		
conv2d_1 (Conv2D)	(None, 16, 16, 32)	4640
<hr/>		
activation_1 (Activation)	(None, 16, 16, 32)	0
<hr/>		
max_pooling2d_1 (MaxPooling2	(None, 8, 8, 32)	0
<hr/>		
dropout_1 (Dropout)	(None, 8, 8, 32)	0
<hr/>		
conv2d_2 (Conv2D)	(None, 8, 8, 64)	18496
<hr/>		
activation_2 (Activation)	(None, 8, 8, 64)	0
<hr/>		
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 64)	0
<hr/>		
dropout_2 (Dropout)	(None, 4, 4, 64)	0
<hr/>		
flatten (Flatten)	(None, 1024)	0
<hr/>		
dense (Dense)	(None, 256)	262400
<hr/>		
dropout_3 (Dropout)	(None, 256)	0
<hr/>		
dense_1 (Dense)	(None, 10)	2570
<hr/>		
=====		
Total params: 288,554		
Trainable params: 288,554		
Non-trainable params: 0		
<hr/>		
-		

In [11]:

```
# model compile  
model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=['accuracy'])
```

In [12]:

```
model.fit(train_ds, epochs=5, validation_data=test_ds, verbose=1, shuffle=True)
```

Epoch 1/5

```
782/782 [=====] - 15s 19ms/step - loss: 1.7074 - accuracy: 0.3728 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
```

Epoch 2/5

```
782/782 [=====] - 11s 14ms/step - loss: 1.3905 - accuracy: 0.4965 - val_loss: 1.2322 - val_accuracy: 0.5655
```

Epoch 3/5

```
782/782 [=====] - 11s 14ms/step - loss: 1.2511 - accuracy: 0.5545 - val_loss: 1.0728 - val_accuracy: 0.6226
```

Epoch 4/5

```
782/782 [=====] - 11s 15ms/step - loss: 1.1668 - accuracy: 0.5848 - val_loss: 1.0379 - val_accuracy: 0.6353
```

Epoch 5/5

```
782/782 [=====] - 12s 15ms/step - loss: 1.1146 - accuracy: 0.6033 - val_loss: 0.9640 - val_accuracy: 0.6596
```

Out[12]:

```
<tensorflow.python.keras.callbacks.History at 0x1eed0f01f48>
```

In [13]:

```
model.evaluate(test_ds, verbose=0)
```

Out[13]:

```
[0.9639831325810426, 0.6596]
```

In [14]:

```
y_pred = model.predict_classes(X_test_scaled)  
print(y_pred.shape)  
print(y_pred)
```

```
(10000,)  
[3 8 8 ... 5 4 7]
```

In [15]:

```
y_true = y_test.ravel()  
y_true
```

Out[15]:

```
array([3, 8, 8, ..., 5, 1, 7], dtype=uint8)
```

In [16]:

```
fig, axes = plt.subplots(2, 8, figsize=(15, 4))
axes = axes.ravel()
for i in range(16):
    idx = np.random.randint(0, len(y_test))
    axes[i].imshow(X_test[idx, :])
    axes[i].set_xticks([])
    axes[i].set_yticks([])
    axes[i].set_title("true={} \npredicted={}".
                      format(cifar10_classes[y_true[idx]], cifar10_classes[y_pred[idx]])))
plt.tight_layout()
```



## accuracy 시각화

In [17]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
import seaborn as sns

cm = confusion_matrix(y_true, y_pred)
cm
```

Out[17]:

```
array([[677, 20, 73, 40, 14, 4, 18, 15, 105, 34],
       [16, 770, 4, 20, 8, 5, 21, 5, 46, 105],
       [65, 2, 457, 81, 131, 75, 123, 40, 20, 6],
       [13, 7, 59, 494, 67, 169, 119, 39, 18, 15],
       [23, 2, 61, 75, 562, 32, 120, 105, 18, 2],
       [5, 3, 58, 218, 55, 540, 53, 51, 10, 7],
       [4, 1, 23, 51, 28, 11, 869, 4, 7, 2],
       [11, 1, 28, 54, 69, 84, 17, 721, 3, 12],
       [54, 35, 10, 22, 7, 9, 11, 6, 827, 19],
       [33, 112, 9, 27, 8, 14, 32, 25, 61, 679]], dtype=int64)
```

## heatmap 작성

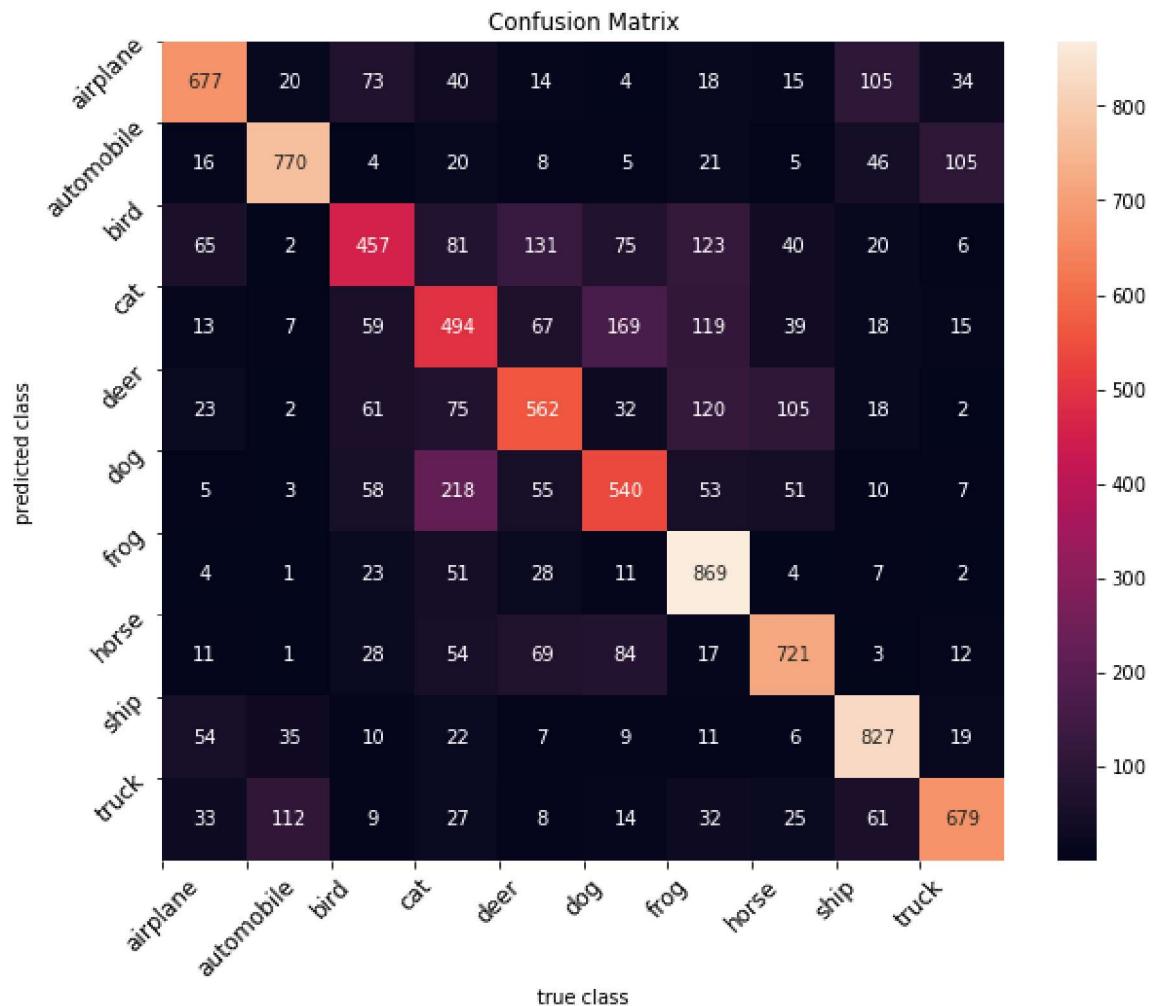
In [21]:

```
plt.figure(figsize=(10,8))

sns.heatmap(cm, annot=True, fmt='d')

plt.xticks(np.arange(10), cifar10_classes, rotation=45, fontsize=12)
plt.yticks(np.arange(10), cifar10_classes, rotation=45, fontsize=12)
plt.xlabel("true class")
plt.ylabel("predicted class")
plt.title('Confusion Matrix')
print('Test Accuracy :', accuracy_score(y_true, y_pred))
```

Test Accuracy : 0.6596



실습 :

Convolutional Layer 와 Pooling Layer 를 추가하고 Hyper-parameter 조정하여 성능 개선 혹은 epochs 늘려서 accuracy 향상 비교

In [ ]:

## 140.model save / load

### 생성한 모델을 disk에 저장하고 복원

- model의 architecture 저장 : json, yaml format
- model의 weight 저장 : h5 file format
- hdf5(Hierarchical Data Format version 5)는 대용량 데이터를 저장하기 위한 파일 포맷

In [1]:

```
import numpy as np
import tensorflow as tf

from tensorflow.keras.layers import Conv2D, AveragePooling2D
from tensorflow.keras.layers import Dense, Flatten, Activation
from tensorflow.keras.models import Sequential
from tensorflow.keras.datasets import mnist
from tensorflow.keras import utils

(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.astype("float32")
X_test = X_test.astype("float32")
X_train /= 255
X_test /= 255

X_train = X_train[:, :, :, np.newaxis]
X_test = X_test[:, :, :, np.newaxis]
y_train = utils.to_categorical(y_train)
y_test = utils.to_categorical(y_test)

# LeNet
model = tf.keras.Sequential()

model.add(Conv2D(6, kernel_size=5, padding="same", input_shape=(28, 28, 1)))
model.add(Activation("relu"))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padding="valid"))

model.add(Conv2D(16, kernel_size=5, padding="valid"))
model.add(Activation('relu'))

model.add(AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padding="valid"))

model.add(Flatten())

model.add(Dense(120))
model.add(Activation("relu"))

model.add(Dense(84))
model.add(Activation("relu"))

model.add(Dense(10))
model.add(Activation("softmax"))

model.compile(loss="categorical_crossentropy", optimizer='adam', metrics=['accuracy'])

history = model.fit(X_train, y_train, batch_size=128, epochs=1, validation_data=(X_test, y_test))

score = model.evaluate(X_test, y_test, verbose=0)
print(score[0])
print(score[1])
```

Train on 60000 samples, validate on 10000 samples

```
60000/60000 [=====] - 8s 138us/sample - loss: 0.2599 - accuracy: 0.9225 - val_loss: 0.0879 - val_accuracy: 0.9737
0.08788994872383774
0.9737
```

## Model Architecture Save

In [2]:

```
json_string = model.to_json()
```

In [3]:

```
with open("lenet.json", "w") as json_file:  
    json_file.write(json_string)
```

## weight 만 save

In [4]:

```
model.save_weights("lenet_weight.h5")
```

## 전체 model save

In [5]:

```
model.save("lenet_model.h5")
```

## Model Load

- architecture load + weight load + compile

In [6]:

```
from tensorflow.keras.models import model_from_json  
  
json_file = open("lenet.json", "r")  
read_model = json_file.read()  
json_file.close()  
  
loaded_model = model_from_json(read_model)  
loaded_model.load_weights("lenet_weight.h5")  
  
loaded_model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=['accuracy'])
```

In [7]:

```
score = loaded_model.evaluate(X_test, y_test, verbose=0)
```

In [8]:

```
from tensorflow.keras.models import load_model  
  
loaded_model = load_model('lenet_model.h5')
```

In [9]:

```
score = loaded_model.evaluate(X_test, y_test, verbose=0)
```

In [10]:

```
y_predict = model.predict_classes(X_test)  
y_predict
```

Out[10]:

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

In [11]:

```
np.argmax(y_test, axis=1)
```

Out[11]:

```
array([7, 2, 1, ..., 4, 5, 6], dtype=int64)
```

In [15]:

```
print("accuracy = {} %".format(np.sum(y_predict == np.argmax(y_test, axis=1)) / len(y_test) * 100))
```

```
accuracy = 97.37 %
```

In [16]:

```
loaded_model.predict_classes(X_test[:1,:])
```

Out[16]:

```
array([7], dtype=int64)
```

# 155.ImageDataGenerator 를 이용한 data feed 구조화 및 Data Augmentation

- [https://storage.googleapis.com/mledu-datasets/cats\\_and\\_dogs\\_filtered.zip](https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip)  
([https://storage.googleapis.com/mledu-datasets/cats\\_and\\_dogs\\_filtered.zip](https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip)) 에서 2,000 장의 JPG file 을 download 하여 압축 해제

**NOTE:** Kaggle 의 "Dogs vs. Cats" dataset (<https://www.kaggle.com/c/dogs-vs-cats/data>) 의 25,000 image 중 교육용으로 2,000 장 발췌.

- 소량의 image data (2000 장) 를 이용하여 model 학습  $\rightarrow$  overfitting 유도

In [1]:

```
try:  
    %tensorflow_version 2.x  
except:  
    pass
```

In [2]:

```
import tensorflow as tf  
tf.__version__
```

In [3]:

```
#Google Colab  
#!wget --no-check-certificate |  
#https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip -O /tmp/cats_and_dogs_filtered.zip  
  
#windows  
_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'  
path_to_zip = tf.keras.utils.get_file('/tmp/cats_and_dogs_filtered.zip', origin=_URL, extract=True)
```

In [4]:

```
import os  
import zipfile  
  
local_zip = '/tmp/cats_and_dogs_filtered.zip'  
  
zip_ref = zipfile.ZipFile(local_zip, 'r')  
  
zip_ref.extractall('/tmp')  
zip_ref.close()
```

/tmp/cats\_and\_dogs\_filtered directory 의 train 과 validation subdirectory 에 training 과 validation datasets 이 구분되어 있고 각 directory 는 cats and dogs subdirectory 로 구성  $\rightarrow$  ImageDataGenerator 가 자동으로 labeling.

In [5]:

```
base_dir = '/tmp/cats_and_dogs_filtered'

train_dir = os.path.join(base_dir, 'train')
validation_dir = os.path.join(base_dir, 'validation')

# Directory with our training cat/dog pictures
train_cats_dir = os.path.join(train_dir, 'cats')
train_dogs_dir = os.path.join(train_dir, 'dogs')

# Directory with our validation cat/dog pictures
validation_cats_dir = os.path.join(validation_dir, 'cats')
validation_dogs_dir = os.path.join(validation_dir, 'dogs')
```

In [6]:

```
train_cat_fnames = os.listdir( train_cats_dir )
train_dog_fnames = os.listdir( train_dogs_dir )

print(train_cat_fnames[:10])
print(train_dog_fnames[:10])

['cat.0.jpg', 'cat.1.jpg', 'cat.10.jpg', 'cat.100.jpg', 'cat.101.jpg', 'cat.102.jpg', 'ca
t.103.jpg', 'cat.104.jpg', 'cat.105.jpg', 'cat.106.jpg']
['dog.0.jpg', 'dog.1.jpg', 'dog.10.jpg', 'dog.100.jpg', 'dog.101.jpg', 'dog.102.jpg',
'dog.103.jpg', 'dog.104.jpg', 'dog.105.jpg', 'dog.106.jpg']
```

In [7]:

```
print('total training cat images :', len(os.listdir( train_cats_dir ) ))
print('total training dog images :', len(os.listdir( train_dogs_dir ) ))

print('total validation cat images :', len(os.listdir( validation_cats_dir ) ))
print('total validation dog images :', len(os.listdir( validation_dogs_dir ) ))

total training cat images : 1000
total training dog images : 1000
total validation cat images : 500
total validation dog images : 500
```

## matplotlib 을 이용하여 data 시각화

- `matplotlib.image.imread(fname)` : Read an image from a file into an array
- `matplotlib.pyplot.imshow` : Display an image

## 개와 고양이 사진 각 8 장씩 display

- `gcf()` - get current figure, current figure 없으면 new one 생성
- `fig.set_size_inches(w,h), (1in == 2.54cm)`

In [8]:

```
%matplotlib inline
import matplotlib.image as mpimg
import matplotlib.pyplot as plt

fig, ax = plt.subplots(4, 4, figsize=(12, 12))
cats = [os.path.join(train_cats_dir, fname) for fname in train_cat_fnames[:8]]
dogs = [os.path.join(train_dogs_dir, fname) for fname in train_dog_fnames[:8]]
ax = ax.ravel()

for i, img_path in enumerate(cats + dogs):
    img = mpimg.imread(img_path)
    ax[i].imshow(img)
    ax[i].axis('off')
```



## 약 72% 정확성을 목표로 small model 작성

이 사진들은 사진의 크기가 제각각이고 사람이나 다른 물건과 섞여 있는 image 들이므로 150x150 size 로 규격화

- 150x150x3 for color image
- output : 0 ('cats'), 1 ('dogs')
- convolution layers : feature map 의 size 축소
- pooling layer : 차원을 반으로 감축

In [9]:

```
import tensorflow as tf

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(16, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
```

In [10]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 16)	448
max_pooling2d (MaxPooling2D)	(None, 74, 74, 16)	0
conv2d_1 (Conv2D)	(None, 72, 72, 32)	4640
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 32)	0	
conv2d_2 (Conv2D)	(None, 34, 34, 64)	18496
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 64)	0	
flatten (Flatten)	(None, 18496)	0
dense (Dense)	(None, 512)	9470464
dense_1 (Dense)	(None, 1)	513
Total params: 9,494,561		
Trainable params: 9,494,561		
Non-trainable params: 0		

- optimizer 는 RMSprop, Adam, Adagrad 중 택일  $\rightarrow$  자동으로 learning rate 조절
  - [RMSprop optimization algorithm \(\[https://en.wikipedia.org/wiki/Stochastic\\\_gradient\\\_descent#RMSProp\]\(https://en.wikipedia.org/wiki/Stochastic\_gradient\_descent#RMSProp\)\)](https://en.wikipedia.org/wiki/Stochastic_gradient_descent#RMSProp)
  - [Adam \(\[https://en.wikipedia.org/wiki/Stochastic\\\_gradient\\\_descent#Adam\]\(https://en.wikipedia.org/wiki/Stochastic\_gradient\_descent#Adam\)\)](https://en.wikipedia.org/wiki/Stochastic_gradient_descent#Adam)
  - [Adagrad \(<https://developers.google.com/machine-learning/glossary/#AdaGrad>\)](https://developers.google.com/machine-learning/glossary/#AdaGrad)

In [11]:

```
from tensorflow.keras.optimizers import RMSprop
```

```
model.compile(optimizer=RMSprop(lr=0.001), loss='binary_crossentropy', metrics = ['acc'])
```

## Data 전처리

- image generator 는 source image 를 읽어 float32 tensor 로 변환하여 label 과 함께 network 에 공급
  - 150x150 size 의 20 개 image 를 1 batch 로 생성 + labels (binary)
- training image 용 image generator 와 validation 용 image generator 생성
- image pixel 값을  $[0, 255]$   $\rightarrow [0, 1]$  로 normalize
  - keras.preprocessing.image.ImageDataGenerator 의 rescale parameter 이용
- image generator 에서 생성된 data 는 model 의 다음 method 로 처리
  - fit\_generator
  - evaluate\_generator
  - predict\_generator

In [12]:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1.0/255.)
test_datagen = ImageDataGenerator(rescale = 1.0/255.)

# -----
# Flow training images in batches of 20 using train_datagen generator
# -----
train_generator = train_datagen.flow_from_directory(train_dir,
                                                    batch_size=20,
                                                    class_mode='binary',
                                                    target_size=(150, 150))

# -----
# Flow validation images in batches of 20 using test_datagen generator
# -----
validation_generator = test_datagen.flow_from_directory(validation_dir,
                                                       batch_size=20,
                                                       class_mode = 'binary',
                                                       target_size = (150, 150))
```

Found 2000 images belonging to 2 classes.

Found 1000 images belonging to 2 classes.

## Training

- 2000 개 training image 와 1000 개 validation image 를 15 epoch 수행
- steps\_per\_epoch = n\_samples / batch\_size (n\_samples : total number of training data)

In [13]:

```
history = model.fit_generator(train_generator,
                               validation_data=validation_generator,
                               epochs=15,
                               verbose=2)
```

```
Epoch 1/15
100/100 - 12s - loss: 1.3911 - acc: 0.5570 - val_loss: 0.6957 - val_acc: 0.5360
Epoch 2/15
100/100 - 10s - loss: 0.6359 - acc: 0.6545 - val_loss: 0.5994 - val_acc: 0.6540
Epoch 3/15
100/100 - 10s - loss: 0.5499 - acc: 0.7445 - val_loss: 0.7266 - val_acc: 0.6200
Epoch 4/15
100/100 - 10s - loss: 0.4567 - acc: 0.7810 - val_loss: 0.7250 - val_acc: 0.6560
Epoch 5/15
100/100 - 10s - loss: 0.3791 - acc: 0.8335 - val_loss: 0.5793 - val_acc: 0.7270
Epoch 6/15
100/100 - 10s - loss: 0.2756 - acc: 0.8860 - val_loss: 0.6617 - val_acc: 0.7350
Epoch 7/15
100/100 - 10s - loss: 0.1893 - acc: 0.9235 - val_loss: 0.7866 - val_acc: 0.7310
Epoch 8/15
100/100 - 10s - loss: 0.1276 - acc: 0.9570 - val_loss: 0.9915 - val_acc: 0.7110
Epoch 9/15
100/100 - 10s - loss: 0.0959 - acc: 0.9665 - val_loss: 1.0758 - val_acc: 0.7310
Epoch 10/15
100/100 - 10s - loss: 0.0694 - acc: 0.9795 - val_loss: 1.3617 - val_acc: 0.7190
Epoch 11/15
100/100 - 10s - loss: 0.0501 - acc: 0.9835 - val_loss: 1.9260 - val_acc: 0.6880
Epoch 12/15
100/100 - 10s - loss: 0.1115 - acc: 0.9815 - val_loss: 1.5389 - val_acc: 0.7120
Epoch 13/15
100/100 - 10s - loss: 0.0367 - acc: 0.9890 - val_loss: 1.5708 - val_acc: 0.7160
Epoch 14/15
100/100 - 10s - loss: 0.0753 - acc: 0.9905 - val_loss: 1.9837 - val_acc: 0.7320
Epoch 15/15
100/100 - 10s - loss: 0.1161 - acc: 0.9795 - val_loss: 1.7118 - val_acc: 0.7200
```

## Evaluating Accuracy and Loss for the Model

- train 과정의 training/validation accuracy 와 loss 를 시각화
- overfitting 시작점 파악

In [33]:

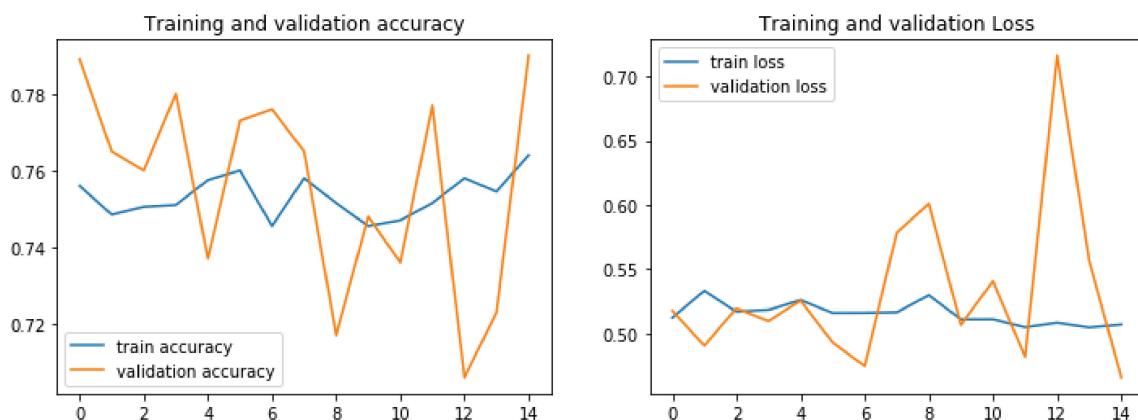
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(history.history['acc'], label='train accuracy')
ax1.plot(history.history['val_acc'], label='validation accuracy')
ax1.set_title('Training and validation accuracy')
ax1.legend()

ax2.plot(history.history['loss'], label='train loss')
ax2.plot(history.history['val_loss'], label='validation loss')
ax2.set_title('Training and validation Loss')
ax2.legend()
```

Out[33]:

```
<matplotlib.legend.Legend at 0x7fd272a9eb70>
```



# Data Augmentation 을 이용한 overfitting 해결

```
# Updated to do image augmentation
train_datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

* rotation_range : degrees (0-180) 사에이서 random 하게 회전
* width_shift and height_shift :range (전체 높이나 너비) 내에서 수직, 수평으로 random 하게 위치 이동
* shear_range : shearing transformation 을 random 하게 적용
* zoom_range : random 하게 zooming
* horizontal_flip : image 의 절반을 수직으로 random 하게 flipping
* fill_mode : 새롭게 생성된 공간은 인접 pixel 로 채움
```

- 각 epoch 마다 ImageDataGenerator 는 위 조건의 transformation 을 image 에 적용하여 transformed image 가 training 에 이용됨.
- generated image 는 original image 와 완전히 다른 것이 아니며 same image 의 서로 다른 version
- 각 epoch 마다, 각 training sample 은 단 한번 augment 되므로 결국 2000 개의 transformed image 로 매 epoch 마다 train

In [14]:

```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

test_datagen = ImageDataGenerator(rescale=1./255)
```

In [15]:

```
train_generator = train_datagen.flow_from_directory(  
    train_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')  
  
validation_generator = test_datagen.flow_from_directory(  
    validation_dir,  
    target_size=(150, 150),  
    batch_size=20,  
    class_mode='binary')
```

Found 2000 images belonging to 2 classes.

Found 1000 images belonging to 2 classes.

In [16]:

```
history = model.fit_generator(train_generator,  
                             validation_data=validation_generator,  
                             epochs=15,  
                             verbose=2)
```

Epoch 1/15

100/100 - 15s - loss: 0.7453 - acc: 0.6130 - val\_loss: 0.5736 - val\_acc: 0.7210

Epoch 2/15

100/100 - 14s - loss: 0.6424 - acc: 0.6675 - val\_loss: 0.5424 - val\_acc: 0.7320

Epoch 3/15

100/100 - 13s - loss: 0.6199 - acc: 0.6745 - val\_loss: 0.5237 - val\_acc: 0.7540

Epoch 4/15

100/100 - 13s - loss: 0.6214 - acc: 0.6855 - val\_loss: 0.5399 - val\_acc: 0.7220

Epoch 5/15

100/100 - 13s - loss: 0.5897 - acc: 0.7050 - val\_loss: 0.5561 - val\_acc: 0.7100

Epoch 6/15

100/100 - 13s - loss: 0.5817 - acc: 0.6995 - val\_loss: 0.5371 - val\_acc: 0.7420

Epoch 7/15

100/100 - 13s - loss: 0.5762 - acc: 0.7000 - val\_loss: 0.5071 - val\_acc: 0.7540

Epoch 8/15

100/100 - 14s - loss: 0.5786 - acc: 0.7005 - val\_loss: 0.5615 - val\_acc: 0.6880

Epoch 9/15

100/100 - 14s - loss: 0.5694 - acc: 0.7250 - val\_loss: 0.5315 - val\_acc: 0.7230

Epoch 10/15

100/100 - 13s - loss: 0.5707 - acc: 0.7125 - val\_loss: 0.5595 - val\_acc: 0.7400

Epoch 11/15

100/100 - 14s - loss: 0.5660 - acc: 0.7150 - val\_loss: 0.4822 - val\_acc: 0.7620

Epoch 12/15

100/100 - 13s - loss: 0.5511 - acc: 0.7150 - val\_loss: 0.6402 - val\_acc: 0.6820

Epoch 13/15

100/100 - 13s - loss: 0.5403 - acc: 0.7320 - val\_loss: 0.4774 - val\_acc: 0.7750

Epoch 14/15

100/100 - 13s - loss: 0.5615 - acc: 0.7115 - val\_loss: 0.4910 - val\_acc: 0.7740

Epoch 15/15

100/100 - 13s - loss: 0.5396 - acc: 0.7280 - val\_loss: 0.5274 - val\_acc: 0.7310

In [17]:

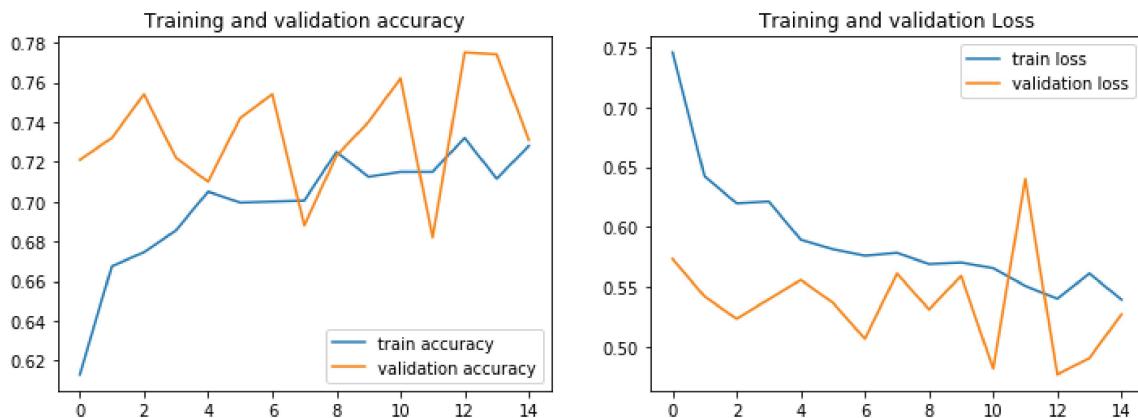
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(history.history['acc'], label='train accuracy')
ax1.plot(history.history['val_acc'], label='validation accuracy')
ax1.set_title('Training and validation accuracy')
ax1.legend()

ax2.plot(history.history['loss'], label='train loss')
ax2.plot(history.history['val_loss'], label='validation loss')
ax2.set_title('Training and validation Loss')
ax2.legend()
```

Out[17]:

```
<matplotlib.legend.Legend at 0x267e0a68a08>
```



## 훈련된 Model 을 이용한 prediction

- 임의의 image 를 선택하여 dog, cat 구분

In [18]:

```
import numpy as np

test_imgs, test_labels = next(validation_generator)
predictions = model.predict(test_imgs)
predictions = [1 if predict >= 0.5 else 0 for predict in predictions]
predictions
```

Out[18]:

```
[0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]
```

In [19]:

```
fig, ax = plt.subplots(1, 10, figsize=(16, 8))
ax = ax.ravel()

for i in range(10):
    img = (test_imgs[i] * 255).astype(np.uint8)
    label = (test_labels[i].astype(np.uint8))
    ax[i].imshow(img)
    ax[i].axis('off')
    ax[i].set_title("{:.0f} / {}".format(test_labels[i], str(predictions[i])))
```



# 160. Keras 내장 사전 훈련 모델을 이용한 Transfer Learning

- ResNet50 pre-trained model 사용
- google colab 사용
- My Drive에 training data upload

## STEP #1: IMPORT LIBRARIES

In [1]:

```
try:  
    %tensorflow_version 2.x  
except:  
    pass  
  
import tensorflow as tf  
  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import random
```

TensorFlow 2.x selected.

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.1.0'
```

## STEP #2: IMPORT MODEL WITH PRE-TRAINED WEIGHTS

In [0]:

```
model = tf.keras.applications.ResNet50(weights = 'imagenet')
```

## STEP #3: EVALUATE THE PRE-TRAINED MODEL

In [4]:

```
from google.colab import drive  
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True).

In [0]:

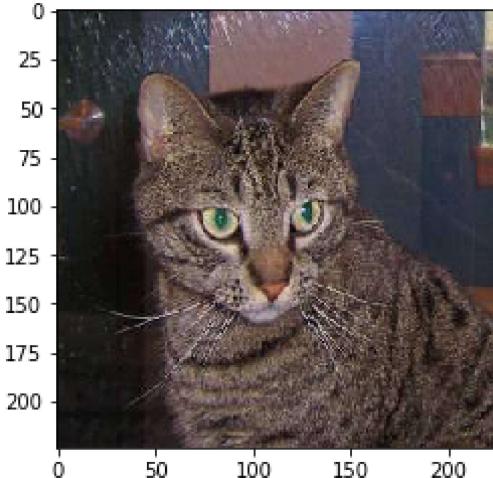
```
Sample_Image= tf.keras.preprocessing.image.load_img(r'/content/drive/My Drive/datasets/Transfer Learning Data/train/cats/cat.1.jpg', target_size = (224, 224))
```

In [6]:

```
plt.imshow(Sample_Image)
```

Out[6]:

```
<matplotlib.image.AxesImage at 0x7f8a80726f98>
```



In [0]:

```
Sample_Image = tf.keras.preprocessing.image.img_to_array(Sample_Image)
```

In [8]:

```
np.shape(Sample_Image)
```

Out[8]:

```
(224, 224, 3)
```

In [9]:

```
Sample_Image = np.expand_dims(Sample_Image, axis = 0)  
np.shape(Sample_Image)
```

Out[9]:

```
(1, 224, 224, 3)
```

**ResNet50 의 입력 spec 에 맞도록 Sample\_Image preprocessing**

In [0]:

```
Sample_Image = tf.keras.applications.resnet50.preprocess_input(Sample_Image)
```

In [11]:

```
predictions = model.predict(Sample_Image)
print(predictions.shape)
print(predictions[0,:10])
```

```
(1, 1000)
[3.2660496e-06 1.9151489e-06 5.0984380e-07 7.1587347e-06 2.6283299e-06
 9.8124992e-06 6.1671926e-07 1.3496027e-06 2.0055033e-06 4.6503956e-07]
```

## 확률 분포로 반환된 prediction 을 category name 으로 decode

In [12]:

```
print('predictions:', tf.keras.applications.resnet50.decode_predictions(predictions, top = 5)[0])
```

```
predictions: [('n02123045', 'tabby', 0.71038485), ('n02123159', 'tiger_cat', 0.17
 78177), ('n02124075', 'Egyptian_cat', 0.06622926), ('n02127052', 'lynx', 0.0063
 938326), ('n03958227', 'plastic_bag', 0.00395526)]
```

## ImageDataGenerator 이용

In [13]:

```
image_datagen = tf.keras.preprocessing.image.ImageDataGenerator(preprocessing_function=
tf.keras.applications.resnet50.preprocess_input)

train_generator = image_datagen.flow_from_directory('/content/drive/My Drive/datasets/Transfer Learning Data/train/',
                                                    target_size = (224, 224),
                                                    color_mode = 'rgb',
                                                    batch_size = 32,
                                                    class_mode = 'categorical',
                                                    shuffle = True)
```

Found 202 images belonging to 2 classes.

## STEP #4: TRANSFER LEARNING 적용 및 model RETRAIN

- ResNet50 의 top layer 제거 (include\_top = False)
- GlobalAaveragePooling2D + Dense() layer 5 개 추가
  - global\_average\_layer = keras.layers.GlobalAveragePooling2D()(base\_model.output)
  - prediction\_layer = keras.layers.Dense(units=2, activation='softmax')(global\_average\_layer)

In [0]:

```
base_model = tf.keras.applications.ResNet50(weights = 'imagenet', include_top = False)
full_model    = tf.keras.applications.ResNet50(weights= 'imagenet', include_top = True)
```

In [15]:

```
print(full_model.summary())
```

Model: "resnet50"

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	[None, 224, 224, 3]	0	
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	input_3[0][0]
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	conv1_pad[0]
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	conv1_conv[0][0]
conv1_relu (Activation)	(None, 112, 112, 64)	0	conv1_bn[0][0]
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	conv1_relu[0]
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	pool1_pad[0]
conv2_block1_1_conv (Conv2D)	(None, 56, 56, 64)	4160	pool1_pool[0][0]
conv2_block1_1_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_bloc
k1_1_conv[0][0]			
conv2_block1_1_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_
1_bn[0][0]			
conv2_block1_2_conv (Conv2D)	(None, 56, 56, 64)	36928	conv2_bloc
k1_1_relu[0][0]			
conv2_block1_2_bn (BatchNormali	(None, 56, 56, 64)	256	conv2_bloc
k1_2_conv[0][0]			
conv2_block1_2_relu (Activation	(None, 56, 56, 64)	0	conv2_block1_
2_bn[0][0]			
conv2_block1_0_conv (Conv2D)	(None, 56, 56, 256)	16640	pool1_pool
[0][0]			
conv2_block1_3_conv (Conv2D)	(None, 56, 56, 256)	16640	conv2_blo
ck1_2_relu[0][0]			

conv2_block1_0_bn (BatchNormali (None, 56, 56, 256) 1024	conv2_blo
ck1_0_conv[0][0]	
conv2_block1_3_bn (BatchNormali (None, 56, 56, 256) 1024	conv2_blo
ck1_3_conv[0][0]	
conv2_block1_add (Add) (None, 56, 56, 256) 0	conv2_block1_0
_bn[0][0]	conv2_block1_3_bn[0][0]
conv2_block1_out (Activation) (None, 56, 56, 256) 0	conv2_block1_
add[0][0]	
conv2_block2_1_conv (Conv2D) (None, 56, 56, 64) 16448	conv2_bloc
k1_out[0][0]	
conv2_block2_1_bn (BatchNormali (None, 56, 56, 64) 256	conv2_bloc
k2_1_conv[0][0]	
conv2_block2_1_relu (Activation (None, 56, 56, 64) 0	conv2_block2_
1_bn[0][0]	
conv2_block2_2_conv (Conv2D) (None, 56, 56, 64) 36928	conv2_bloc
k2_1_relu[0][0]	
conv2_block2_2_bn (BatchNormali (None, 56, 56, 64) 256	conv2_bloc
k2_2_conv[0][0]	
conv2_block2_2_relu (Activation (None, 56, 56, 64) 0	conv2_block2_
2_bn[0][0]	
conv2_block2_3_conv (Conv2D) (None, 56, 56, 256) 16640	conv2_blo
ck2_2_relu[0][0]	
conv2_block2_3_bn (BatchNormali (None, 56, 56, 256) 1024	conv2_blo
ck2_3_conv[0][0]	
conv2_block2_add (Add) (None, 56, 56, 256) 0	conv2_block1_o
ut[0][0]	conv2_block2_3_bn[0][0]
conv2_block2_out (Activation) (None, 56, 56, 256) 0	conv2_block2_
add[0][0]	
conv2_block3_1_conv (Conv2D) (None, 56, 56, 64) 16448	conv2_bloc

k2\_out[0][0]

---

conv2\_block3\_1\_bn (BatchNormali (None, 56, 56, 64) 256 conv2\_bloc  
k3\_1\_conv[0][0]

---

conv2\_block3\_1\_relu (Activation (None, 56, 56, 64) 0 conv2\_block3\_1\_bn[0][0]

---

conv2\_block3\_2\_conv (Conv2D) (None, 56, 56, 64) 36928 conv2\_bloc  
k3\_1\_relu[0][0]

---

conv2\_block3\_2\_bn (BatchNormali (None, 56, 56, 64) 256 conv2\_bloc  
k3\_2\_conv[0][0]

---

conv2\_block3\_2\_relu (Activation (None, 56, 56, 64) 0 conv2\_block3\_2\_bn[0][0]

---

conv2\_block3\_3\_conv (Conv2D) (None, 56, 56, 256) 16640 conv2\_bloc  
k3\_2\_relu[0][0]

---

conv2\_block3\_3\_bn (BatchNormali (None, 56, 56, 256) 1024 conv2\_bloc  
k3\_3\_conv[0][0]

---

conv2\_block3\_add (Add) (None, 56, 56, 256) 0 conv2\_block2\_o  
ut[0][0] conv2\_block3\_3\_bn[0][0]

---

conv2\_block3\_out (Activation) (None, 56, 56, 256) 0 conv2\_block3\_ add[0][0]

---

conv3\_block1\_1\_conv (Conv2D) (None, 28, 28, 128) 32896 conv2\_bloc  
k3\_out[0][0]

---

conv3\_block1\_1\_bn (BatchNormali (None, 28, 28, 128) 512 conv3\_bloc  
k1\_1\_conv[0][0]

---

conv3\_block1\_1\_relu (Activation (None, 28, 28, 128) 0 conv3\_block1\_1\_bn[0][0]

---

conv3\_block1\_2\_conv (Conv2D) (None, 28, 28, 128) 147584 conv3\_bloc  
k1\_1\_relu[0][0]

---

conv3\_block1\_2\_bn (BatchNormali (None, 28, 28, 128) 512 conv3\_bloc  
k1\_2\_conv[0][0]

---

conv3\_block1\_2\_relu (Activation (None, 28, 28, 128) 0 conv3\_block1\_

2\_bn[0][0]

---

conv3\_block1\_0\_conv (Conv2D) (None, 28, 28, 512) 131584 conv2\_bloc  
ck3\_out[0][0]

---

conv3\_block1\_3\_conv (Conv2D) (None, 28, 28, 512) 66048 conv3\_bloc  
ck1\_2\_relu[0][0]

---

conv3\_block1\_0\_bn (BatchNormali (None, 28, 28, 512) 2048 conv3\_bloc  
ck1\_0\_conv[0][0]

---

conv3\_block1\_3\_bn (BatchNormali (None, 28, 28, 512) 2048 conv3\_bloc  
ck1\_3\_conv[0][0]

---

conv3\_block1\_add (Add) (None, 28, 28, 512) 0 conv3\_block1\_0  
\_bn[0][0] conv3\_block1\_3\_bn[0][0]

---

conv3\_block1\_out (Activation) (None, 28, 28, 512) 0 conv3\_block1\_0  
add[0][0]

---

conv3\_block2\_1\_conv (Conv2D) (None, 28, 28, 128) 65664 conv3\_bloc  
ck1\_out[0][0]

---

conv3\_block2\_1\_bn (BatchNormali (None, 28, 28, 128) 512 conv3\_bloc  
k2\_1\_conv[0][0]

---

conv3\_block2\_1\_relu (Activation (None, 28, 28, 128) 0 conv3\_block2\_1  
\_bn[0][0]

---

conv3\_block2\_2\_conv (Conv2D) (None, 28, 28, 128) 147584 conv3\_bloc  
ck2\_1\_relu[0][0]

---

conv3\_block2\_2\_bn (BatchNormali (None, 28, 28, 128) 512 conv3\_bloc  
k2\_2\_conv[0][0]

---

conv3\_block2\_2\_relu (Activation (None, 28, 28, 128) 0 conv3\_block2\_2  
\_bn[0][0]

---

conv3\_block2\_3\_conv (Conv2D) (None, 28, 28, 512) 66048 conv3\_bloc  
ck2\_2\_relu[0][0]

---

conv3\_block2\_3\_bn (BatchNormali (None, 28, 28, 512) 2048 conv3\_bloc  
ck2\_3\_conv[0][0]

---

conv3\_block2\_add (Add) (None, 28, 28, 512) 0 conv3\_block1\_o

ut[0][0]		conv3_block2_3_bn[0][0]	
conv3_block2_out (Activation) (None, 28, 28, 512) 0		conv3_block2_add[0][0]	
conv3_block3_1_conv (Conv2D) (None, 28, 28, 128) 65664		conv3_bloc ck2_out[0][0]	
conv3_block3_1_bn (BatchNormali (None, 28, 28, 128) 512		conv3_bloc k3_1_conv[0][0]	
conv3_block3_1_relu (Activation (None, 28, 28, 128) 0		conv3_block3_1_bn[0][0]	
conv3_block3_2_conv (Conv2D) (None, 28, 28, 128) 147584		conv3_bloc ck3_1_relu[0][0]	
conv3_block3_2_bn (BatchNormali (None, 28, 28, 128) 512		conv3_bloc k3_2_conv[0][0]	
conv3_block3_2_relu (Activation (None, 28, 28, 128) 0		conv3_block3_2_bn[0][0]	
conv3_block3_3_conv (Conv2D) (None, 28, 28, 512) 66048		conv3_bloc ck3_2_relu[0][0]	
conv3_block3_3_bn (BatchNormali (None, 28, 28, 512) 2048		conv3_bloc k3_3_conv[0][0]	
conv3_block3_add (Add) (None, 28, 28, 512) 0		conv3_block2_o ut[0][0]	conv3_block3_3_bn[0][0]
conv3_block3_out (Activation) (None, 28, 28, 512) 0		conv3_block3_3_bn[0][0]	
conv3_block4_1_conv (Conv2D) (None, 28, 28, 128) 65664		conv3_bloc ck3_out[0][0]	
conv3_block4_1_bn (BatchNormali (None, 28, 28, 128) 512		conv3_bloc k4_1_conv[0][0]	
conv3_block4_1_relu (Activation (None, 28, 28, 128) 0		conv3_bloc k4_1_bn[0][0]	

conv3\_block4\_2\_conv (Conv2D) (None, 28, 28, 128) 147584 conv3\_blo  
ck4\_1\_relu[0][0]

---

conv3\_block4\_2\_bn (BatchNormali (None, 28, 28, 128) 512 conv3\_bloc  
k4\_2\_conv[0][0]

---

conv3\_block4\_2\_relu (Activation (None, 28, 28, 128) 0 conv3\_block4\_2  
\_bn[0][0]

---

conv3\_block4\_3\_conv (Conv2D) (None, 28, 28, 512) 66048 conv3\_blo  
ck4\_2\_relu[0][0]

---

conv3\_block4\_3\_bn (BatchNormali (None, 28, 28, 512) 2048 conv3\_blo  
ck4\_3\_conv[0][0]

---

conv3\_block4\_add (Add) (None, 28, 28, 512) 0 conv3\_block3\_o  
ut[0][0]  
conv3\_block4\_3\_bn[0][0]

---

conv3\_block4\_out (Activation) (None, 28, 28, 512) 0 conv3\_block4\_  
add[0][0]

---

conv4\_block1\_1\_conv (Conv2D) (None, 14, 14, 256) 131328 conv3\_blo  
ck4\_out[0][0]

---

conv4\_block1\_1\_bn (BatchNormali (None, 14, 14, 256) 1024 conv4\_blo  
ck1\_1\_conv[0][0]

---

conv4\_block1\_1\_relu (Activation (None, 14, 14, 256) 0 conv4\_block1\_1  
\_bn[0][0]

---

conv4\_block1\_2\_conv (Conv2D) (None, 14, 14, 256) 590080 conv4\_blo  
ck1\_1\_relu[0][0]

---

conv4\_block1\_2\_bn (BatchNormali (None, 14, 14, 256) 1024 conv4\_blo  
ck1\_2\_conv[0][0]

---

conv4\_block1\_2\_relu (Activation (None, 14, 14, 256) 0 conv4\_block1\_2  
\_bn[0][0]

---

conv4\_block1\_0\_conv (Conv2D) (None, 14, 14, 1024) 525312 conv3\_blo  
ck4\_out[0][0]

---

conv4\_block1\_3\_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4\_blo  
ck1\_2\_relu[0][0]

---

conv4\_block1\_0\_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4\_blo  
ck1\_0\_conv[0][0]

---

conv4\_block1\_3\_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4\_blo  
ck1\_3\_conv[0][0]

---

conv4\_block1\_add (Add) (None, 14, 14, 1024) 0 conv4\_block1\_  
0\_bn[0][0] conv4\_block1\_3\_bn[0][0]

---

conv4\_block1\_out (Activation) (None, 14, 14, 1024) 0 conv4\_block1\_  
add[0][0]

---

conv4\_block2\_1\_conv (Conv2D) (None, 14, 14, 256) 262400 conv4\_blo  
ck1\_out[0][0]

---

conv4\_block2\_1\_bn (BatchNormali (None, 14, 14, 256) 1024 conv4\_blo  
ck2\_1\_conv[0][0]

---

conv4\_block2\_1\_relu (Activation (None, 14, 14, 256) 0 conv4\_block2\_  
1\_bn[0][0]

---

conv4\_block2\_2\_conv (Conv2D) (None, 14, 14, 256) 590080 conv4\_blo  
ck2\_1\_relu[0][0]

---

conv4\_block2\_2\_bn (BatchNormali (None, 14, 14, 256) 1024 conv4\_blo  
ck2\_2\_conv[0][0]

---

conv4\_block2\_2\_relu (Activation (None, 14, 14, 256) 0 conv4\_block2\_  
2\_bn[0][0]

---

conv4\_block2\_3\_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4\_blo  
ck2\_2\_relu[0][0]

---

conv4\_block2\_3\_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4\_blo  
ck2\_3\_conv[0][0]

---

conv4\_block2\_add (Add) (None, 14, 14, 1024) 0 conv4\_block1\_  
out[0][0] conv4\_block2\_3\_bn[0][0]

---

conv4\_block2\_out (Activation) (None, 14, 14, 1024) 0 conv4\_block2\_  
add[0][0]

---

conv4\_block3\_1\_conv (Conv2D) (None, 14, 14, 256) 262400 conv4\_blo  
ck2\_out[0][0]

---

conv4_block3_1_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_blo
ck3_1_conv[0][0]	
conv4_block3_1_relu (Activation (None, 14, 14, 256) 0	conv4_block3_
1_bn[0][0]	
conv4_block3_2_conv (Conv2D) (None, 14, 14, 256) 590080	conv4_blo
ck3_1_relu[0][0]	
conv4_block3_2_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_blo
ck3_2_conv[0][0]	
conv4_block3_2_relu (Activation (None, 14, 14, 256) 0	conv4_block3_
2_bn[0][0]	
conv4_block3_3_conv (Conv2D) (None, 14, 14, 1024) 263168	conv4_blo
ck3_2_relu[0][0]	
conv4_block3_3_bn (BatchNormali (None, 14, 14, 1024) 4096	conv4_blo
ck3_3_conv[0][0]	
conv4_block3_add (Add) (None, 14, 14, 1024) 0	conv4_block2_
out[0][0]	conv4_block3_3_bn[0][0]
conv4_block3_out (Activation) (None, 14, 14, 1024) 0	conv4_block3_
add[0][0]	
conv4_block4_1_conv (Conv2D) (None, 14, 14, 256) 262400	conv4_blo
ck3_out[0][0]	
conv4_block4_1_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_blo
ck4_1_conv[0][0]	
conv4_block4_1_relu (Activation (None, 14, 14, 256) 0	conv4_block4_
1_bn[0][0]	
conv4_block4_2_conv (Conv2D) (None, 14, 14, 256) 590080	conv4_blo
ck4_1_relu[0][0]	
conv4_block4_2_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_blo
ck4_2_conv[0][0]	
conv4_block4_2_relu (Activation (None, 14, 14, 256) 0	conv4_block4_
2_bn[0][0]	

conv4_block4_3_conv (Conv2D) (None, 14, 14, 1024) 263168	conv4_block4_2_relu[0][0]
conv4_block4_3_bn (BatchNormali (None, 14, 14, 1024) 4096	conv4_block4_3_conv[0][0]
conv4_block4_add (Add) (None, 14, 14, 1024) 0	conv4_block3_out[0][0]
	conv4_block4_3_bn[0][0]
conv4_block4_out (Activation) (None, 14, 14, 1024) 0	conv4_block4_add[0][0]
conv4_block5_1_conv (Conv2D) (None, 14, 14, 256) 262400	conv4_block4_ck4_out[0][0]
conv4_block5_1_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_block5_1_conv[0][0]
conv4_block5_1_relu (Activation (None, 14, 14, 256) 0	conv4_block5_1_bn[0][0]
conv4_block5_2_conv (Conv2D) (None, 14, 14, 256) 590080	conv4_block5_1_relu[0][0]
conv4_block5_2_bn (BatchNormali (None, 14, 14, 256) 1024	conv4_block5_2_conv[0][0]
conv4_block5_2_relu (Activation (None, 14, 14, 256) 0	conv4_block5_2_bn[0][0]
conv4_block5_3_conv (Conv2D) (None, 14, 14, 1024) 263168	conv4_block5_2_relu[0][0]
conv4_block5_3_bn (BatchNormali (None, 14, 14, 1024) 4096	conv4_block5_3_conv[0][0]
conv4_block5_add (Add) (None, 14, 14, 1024) 0	conv4_block4_out[0][0]
	conv4_block5_3_bn[0][0]
conv4_block5_out (Activation) (None, 14, 14, 1024) 0	conv4_block5_add[0][0]
conv4_block6_1_conv (Conv2D) (None, 14, 14, 256) 262400	conv4_block5_ck5_out[0][0]

conv4_block6_1_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_blo ck6_1_conv[0][0]
conv4_block6_1_relu (Activation (None, 14, 14, 256) 0 conv4_block6_1_bn[0][0]
conv4_block6_2_conv (Conv2D) (None, 14, 14, 256) 590080 conv4_blo ck6_1_relu[0][0]
conv4_block6_2_bn (BatchNormali (None, 14, 14, 256) 1024 conv4_blo ck6_2_conv[0][0]
conv4_block6_2_relu (Activation (None, 14, 14, 256) 0 conv4_block6_2_bn[0][0]
conv4_block6_3_conv (Conv2D) (None, 14, 14, 1024) 263168 conv4_blo ck6_2_relu[0][0]
conv4_block6_3_bn (BatchNormali (None, 14, 14, 1024) 4096 conv4_blo ck6_3_conv[0][0]
conv4_block6_add (Add) (None, 14, 14, 1024) 0 conv4_block5_out[0][0] conv4_block6_3_bn[0][0]
conv4_block6_out (Activation) (None, 14, 14, 1024) 0 conv4_block6_add[0][0]
conv5_block1_1_conv (Conv2D) (None, 7, 7, 512) 524800 conv4_blo ck6_out[0][0]
conv5_block1_1_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_blo ck1_1_conv[0][0]
conv5_block1_1_relu (Activation (None, 7, 7, 512) 0 conv5_block1_1_bn[0][0]
conv5_block1_2_conv (Conv2D) (None, 7, 7, 512) 2359808 conv5_blo ck1_1_relu[0][0]
conv5_block1_2_bn (BatchNormali (None, 7, 7, 512) 2048 conv5_blo ck1_2_conv[0][0]
conv5_block1_2_relu (Activation (None, 7, 7, 512) 0 conv5_block1_2_bn[0][0]

conv5_block1_0_conv (Conv2D) (None, 7, 7, 2048)	2099200	conv4_blo ck6_out[0][0]
conv5_block1_3_conv (Conv2D) (None, 7, 7, 2048)	1050624	conv5_blo ck1_2_relu[0][0]
conv5_block1_0_bn (BatchNormali (None, 7, 7, 2048)	8192	conv5_bloc k1_0_conv[0][0]
conv5_block1_3_bn (BatchNormali (None, 7, 7, 2048)	8192	conv5_bloc k1_3_conv[0][0]
conv5_block1_add (Add) (None, 7, 7, 2048)	0	conv5_block1_0 _bn[0][0]
		conv5_block1_3_bn[0][0]
conv5_block1_out (Activation) (None, 7, 7, 2048)	0	conv5_block1_a dd[0][0]
conv5_block2_1_conv (Conv2D) (None, 7, 7, 512)	1049088	conv5_blo ck1_out[0][0]
conv5_block2_1_bn (BatchNormali (None, 7, 7, 512)	2048	conv5_bloc k2_1_conv[0][0]
conv5_block2_1_relu (Activation (None, 7, 7, 512)	0	conv5_block2_1 _bn[0][0]
conv5_block2_2_conv (Conv2D) (None, 7, 7, 512)	2359808	conv5_blo ck2_1_relu[0][0]
conv5_block2_2_bn (BatchNormali (None, 7, 7, 512)	2048	conv5_bloc k2_2_conv[0][0]
conv5_block2_2_relu (Activation (None, 7, 7, 512)	0	conv5_block2_2 _bn[0][0]
conv5_block2_3_conv (Conv2D) (None, 7, 7, 2048)	1050624	conv5_blo ck2_2_relu[0][0]
conv5_block2_3_bn (BatchNormali (None, 7, 7, 2048)	8192	conv5_bloc k2_3_conv[0][0]
conv5_block2_add (Add) (None, 7, 7, 2048)	0	conv5_block1_o ut[0][0]

conv5_block2_3_bn[0][0]			
conv5_block2_out (Activation)	(None, 7, 7, 2048)	0	conv5_block2_a dd[0][0]
conv5_block3_1_conv (Conv2D)	(None, 7, 7, 512)	1049088	conv5_blo ck2_out[0][0]
conv5_block3_1_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_bloc k3_1_conv[0][0]
conv5_block3_1_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_1 _bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2359808	conv5_blo ck3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_bloc k3_2_conv[0][0]
conv5_block3_2_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_2 _bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624	conv5_blo ck3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali	(None, 7, 7, 2048)	8192	conv5_blo ck3_3_conv[0][0]
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_o ut[0][0]
			conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_a dd[0][0]
avg_pool (GlobalAveragePooling2	(None, 2048)	0	conv5_block3_出 [0][0]
probs (Dense)	(None, 1000)	2049000	avg_pool[0][0]
=====			
=====			
=====			
Total params:	25,636,712		
Trainable params:	25,583,592		
Non-trainable params:	53,120		

None

In [16]:

```
for layer in base_model.layers[-5:]:
    print(layer.name)
print()
for layer in full_model.layers[-7:]:
    print(layer.name)
```

conv5\_block3\_2\_relu  
conv5\_block3\_3\_conv  
conv5\_block3\_3\_bn  
conv5\_block3\_add  
conv5\_block3\_out

conv5\_block3\_2\_relu  
conv5\_block3\_3\_conv  
conv5\_block3\_3\_bn  
conv5\_block3\_add  
conv5\_block3\_out  
avg\_pool  
probs

## base model 의 마지막에 5 개 layer 추가

In [0]:

```
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)

x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(1024, activation = 'relu')(x)
x = tf.keras.layers.Dense(512, activation = 'relu')(x)
preds = tf.keras.layers.Dense(2, activation = 'softmax')(x)
```

In [0]:

```
model = tf.keras.models.Model(inputs = base_model.input, outputs = preds)
```

In [19]:

```
for layer in model.layers[-10:]:
    print(layer.name)
```

conv5\_block3\_3\_conv  
conv5\_block3\_3\_bn  
conv5\_block3\_add  
conv5\_block3\_out  
global\_average\_pooling2d  
dense  
dense\_1  
dense\_2  
dense\_3  
dense\_4

## Fine tuning 하기 전에 새로이 add 한 layer 들을 1 차 training 하여 초기화

In [0]:

```
for layer in model.layers[:-5]:  
    layer.trainable = False
```

```
for layer in model.layers[-5:]:  
    layer.trainable = True
```

In [0]:

```
model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [22]:

```
model.fit_generator(generator = train_generator, epochs = 5)
```

WARNING:tensorflow:From <ipython-input-22-bc98654f7206>:1: Model.fit\_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.

Instructions for updating:

Please use Model.fit, which supports generators.

WARNING:tensorflow:sample\_weight modes were coerced from

```
...  
to  
['...']  
Train for 7 steps  
Epoch 1/5  
7/7 [=====] - 5s 665ms/step - loss: 2.  
3096 - accuracy: 0.5941  
Epoch 2/5  
7/7 [=====] - 1s 166ms/step - loss: 0.  
4176 - accuracy: 0.8317  
Epoch 3/5  
7/7 [=====] - 1s 165ms/step - loss: 0.  
4183 - accuracy: 0.8515  
Epoch 4/5  
7/7 [=====] - 1s 168ms/step - loss: 0.  
0791 - accuracy: 0.9851  
Epoch 5/5  
7/7 [=====] - 1s 170ms/step - loss: 0.  
0776 - accuracy: 0.9752
```

Out[22]:

```
<tensorflow.python.keras.callbacks.History at 0x7f8a6019eac8>
```

In [ ]:

```
model.summary()
```

## 마지막 50 개 layer 만 fine tuning

In [0]:

```
for layer in model.layers[:-50]:  
    layer.trainable = False
```

In [0]:

```
for layer in model.layers[-50:]:  
    layer.trainable = True
```

In [0]:

```
model.compile(optimizer = 'Adam', loss = 'categorical_crossentropy', metrics = ['accuracy'])
```

In [0]:

```
model.fit_generator?
```

- steps\_per\_epoch : Total number of steps (batches of samples) to yield from generator

In [28]:

```
history = model.fit_generator(generator = train_generator, steps_per_epoch=train_generator.  
n//train_generator.batch_size, epochs = 5)
```

WARNING:tensorflow:sample\_weight modes were coerced from

...  
to  
['...']

Train for 6 steps

Epoch 1/5

6/6 [=====] - 5s 840ms/step - loss: 0.  
4429 - accuracy: 0.8765

Epoch 2/5

6/6 [=====] - 1s 213ms/step - loss: 0.  
1284 - accuracy: 0.9896

Epoch 3/5

6/6 [=====] - 1s 192ms/step - loss: 0.  
0232 - accuracy: 0.9941

Epoch 4/5

6/6 [=====] - 1s 188ms/step - loss: 0.  
0015 - accuracy: 1.0000

Epoch 5/5

6/6 [=====] - 1s 190ms/step - loss: 3.  
2881e-05 - accuracy: 1.0000

## STEP #5: EVALUATE THE MODEL

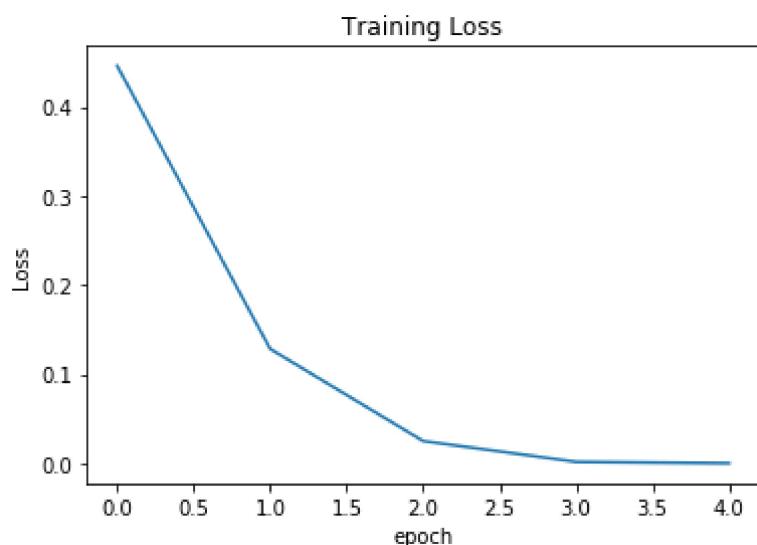
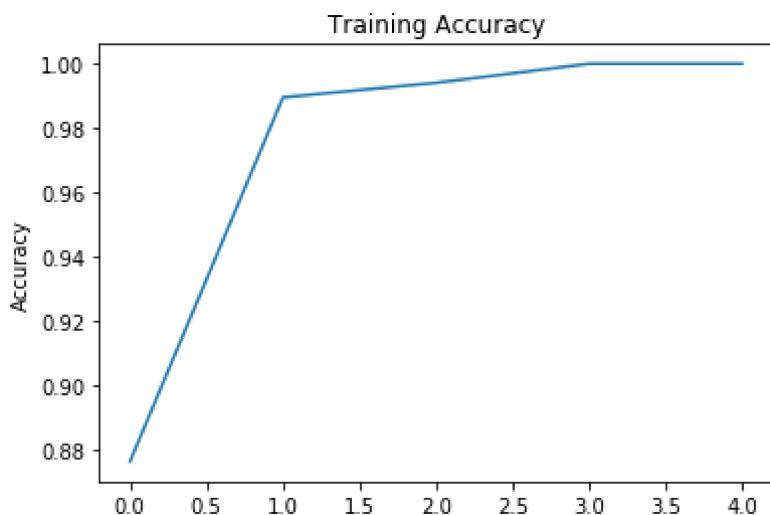
In [29]:

```
acc = history.history['accuracy']
loss = history.history['loss']

plt.figure()
plt.plot(acc, label='Training Accuracy')
plt.ylabel('Accuracy')
plt.title('Training Accuracy')

plt.figure()

plt.plot(loss, label='Training Loss')
plt.ylabel('Loss')
plt.title('Training Loss')
plt.xlabel('epoch')
plt.show()
```



In [0]:

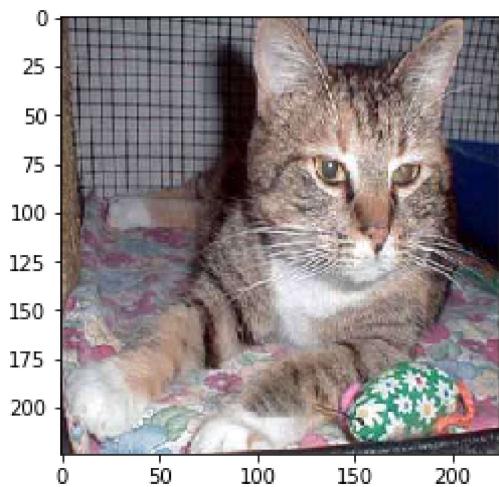
```
Sample_Image= tf.keras.preprocessing.image.load_img(r'/content/drive/My Drive/datasets/Transfer Learning Data/cat.282.jpg', target_size = (224, 224))
```

In [31]:

```
plt.imshow(Sample_Image)
```

Out[31]:

```
<matplotlib.image.AxesImage at 0x7f8a669d7160>
```



In [32]:

```
Sample_Image = tf.keras.preprocessing.image.img_to_array(Sample_Image)  
np.shape(Sample_Image)
```

Out[32]:

```
(224, 224, 3)
```

In [0]:

```
Sample_Image = np.expand_dims(Sample_Image, axis = 0)
```

In [34]:

```
Sample_Image = tf.keras.applications.resnet50.preprocess_input(Sample_Image)  
predictions = model.predict(Sample_Image)  
print('Predictions:', predictions)
```

```
Predictions: [[1. 0.]]
```

In [0]:

# 170. Keras Word Encoding

- Tokenizer, pad\_sequence 소개

In [1]:

```
import csv
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

In [2]:

```
tokenizer = Tokenizer(num_words=100, oov_token='<OOV>') # 빈도수 상위 100 개로 구성

sentences = [
    'I love my dog',
    'I love my cat',
    'You love my dog!',
    'I was born in Korea and graduated University in USA.'
]

tokenizer.fit_on_texts(sentences)
word_index = tokenizer.word_index
print(word_index)

{'<OOV>': 1, 'i': 2, 'love': 3, 'my': 4, 'dog': 5, 'in': 6, 'cat': 7, 'you': 8, 'was': 9,
 'born': 10, 'korea': 11, 'and': 12, 'graduated': 13, 'university': 14, 'usa': 15}
```

In [3]:

```
sequences = tokenizer.texts_to_sequences(sentences)
```

Out[3]:

```
[[2, 3, 4, 5],
 [2, 3, 4, 7],
 [8, 3, 4, 5],
 [2, 9, 10, 6, 11, 12, 13, 14, 6, 15]]
```

In [4]:

```
padded = pad_sequences(sequences)
```

Out[4]:

```
array([[ 0,  0,  0,  0,  0,  0,  2,  3,  4,  5],
       [ 0,  0,  0,  0,  0,  0,  2,  3,  4,  7],
       [ 0,  0,  0,  0,  0,  0,  8,  3,  4,  5],
       [ 2,  9, 10,  6, 11, 12, 13, 14,  6, 15]])
```

In [0]:

# 180. IMDB 영화평 Text 분류 - Sentiment Analysis

IMDB (Internet Movie Database, <https://www.imdb.com/>) Dataset

- IMDb site 의 50,000 개 영화평 text 로 구성
- 25,000 training set 과 25,000 testing set 으로 구성 (training set 과 test set 이 반반임)
- positive / negative review 가 동일한 숫자로 구성
- keras.imdb 사용. 이미 preprocessing 이 되어 있고 review 내의 각 단어는 integer 숫자로 변환되어 있음. (빈번히 사용되는 순서로 숫자 부여)
- num\_words=10000 - training set 에 가장 빈번히 등장하는 상위 10,000 개 단어 유지

In [1]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Dense, LSTM, Embedding, Dropout
from tensorflow.keras.models import Sequential
import tensorflow_datasets as tfds
import numpy as np
```

In [2]:

```
tf.__version__
```

Out[2]:

```
'2.0.0'
```

imdb\_reviews/subwords8k - 8000 개 단어로 이루어진 dataset

In [3]:

```
imdb, info = tfds.load("imdb_reviews/subwords8k", with_info=True, as_supervised=True)
```

WARNING: Logging before flag parsing goes to stderr.  
W1220 13:09:28.724305 2645904 dataset\_builder.py:439] Warning: Setting shuffle\_files=True because split=TRAIN and shuffle\_files=None. This behavior will be deprecated on 2019-08-06, at which point shuffle\_files=False will be the default for all splits.

In [4]:

```
imdb
```

Out[4]:

```
{'test': <_OptionsDataset shapes: ((None,), ()), types: (tf.int64, tf.int64)>,
 'train': <_OptionsDataset shapes: ((None,), ()), types: (tf.int64, tf.int64)>,
 'unsupervised': <_OptionsDataset shapes: ((None,), ()), types: (tf.int64, tf.int64)>}
```

In [5]:

```
info
```

Out[5]:

```
tfds.core.DatasetInfo(  
    name='imdb_reviews',  
    version=0.1.0,  
    description='Large Movie Review Dataset.  
This is a dataset for binary sentiment classification containing substantially more  
data than previous benchmark datasets. We provide a set of 25,000 highly polar  
movie reviews for training, and 25,000 for testing. There is additional unlabeled  
data for use as well.',  
    urls=['http://ai.stanford.edu/~amaas/data/sentiment/'],  
    features=FeaturesDict({  
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),  
        'text': Text(shape=(None,), dtype=tf.int64, encoder=<SubwordTextEncode  
r vocab_size=8185>),  
    }),  
    total_num_examples=100000,  
    splits={  
        'test': 25000,  
        'train': 25000,  
        'unsupervised': 50000,  
    },  
    supervised_keys=('text', 'label'),  
    citation="""@InProceedings{maas-EtAl:2011:ACL-HLT2011,  
        author = {Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. a  
nd Huang, Dan and Ng, Andrew Y. and Potts, Christopher},  
        title = {Learning Word Vectors for Sentiment Analysis},  
        booktitle = {Proceedings of the 49th Annual Meeting of the Association for C  
omputational Linguistics: Human Language Technologies},  
        month = {June},  
        year = {2011},  
        address = {Portland, Oregon, USA},  
        publisher = {Association for Computational Linguistics},  
        pages = {142--150},  
        url = {http://www.aclweb.org/anthology/P11-1015}  
    }""",  
    redistribution_info=  
)
```

## train, test split

- tf.data dataset

In [6]:

```
train_dataset, test_dataset = imdb['train'], imdb['test']
```

In [7]:

```
for stmt, label in train_dataset.take(1):
    print(stmt)
    print(label.numpy())
```

tf.Tensor(

[4027 12 2841 181 38 2177 453 54 2059 481 105 202 142 38  
19 3294 2041 7961 1746 1320 54 9 132 78 43 1 3322 7961  
6 6077 8 932 34 19 6824 7961 2273 5754 8044 3 147 315  
15 1915 1137 571 687 4657 539 20 5299 424 132 434 2894 1565  
610 2 72 37 1421 59 7860 7961 5667 37 96 3193 3660 7961  
59 20 4928 1281 1848 4109 4 48 1 724 2273 5754 7961 875  
38 3923 36 43 1 1586 428 345 7989 10 16 10 17 1293  
38 19 3294 2041 7961 1746 1320 54 394 5581 7961 265 7 430  
1 1325 49 5216 7961 168 5 2536 856 1555 1577 127 265 7  
430 2497 64 5216 3 1419 79 2273 5754 7961 5551 1022 4 1061  
6766 7961 969 1 7982 7980 424 1557 27 9 1325 49 1793 498  
2497 124 3 5581 7968 8 388 2 4 2308 2030 7850 34 3922  
8050 2 9 33 67 6163 2146 2 26 9 547 7 35 1  
74 1276 221 10 16 10 17 7368 34 4 3007 34 423 665  
222 3427 771 5581 7961 4340 76 1 423 665 222 2367 3677 44  
2574 344 5 379 1 1466 2427 502 42 70 31 2490 75 1  
4628 1940 4782 8 80 93 248 7974 1 108 5535 568 31 7  
7584 7961 23 4902 7961 4374 8 25 1 4148 628 28 217 78  
200 148 5581 7961 4908 8 1 2367 7397 297 158 684 7020 9  
479 23 5581 7968 8 6649 34 177 1 5535 568 5 176 139  
2567 1 1417 46 265 1 1053 221 10 16 10 17 4459 8  
92 7158 40 44 14 344 99 1 3624 28 4730 4559 6456 136  
58 5 30 1244 4783 7321 47 2048 8030 28 320 43 1233 688  
8049 7974 46 70 186 25 7 430 4702 3689 7961 6551 2046 7961  
25 5012 7968 8044 38 19 1625 2316 223 2326 86 5581 7961 4908  
7961 1 2367 3677 5 144 5838 1 4353 1493 6 1 502 5  
868 1 423 665 222 6249 8043 3 1325 49 5 2497 64 92  
886 220 8029 221 10 16 10 17 392 1325 49 5 2497 64  
1654 7 252 946 7 92 385 48 1 5758 5 53 1226 79  
4477 7 5373 5 6109 3 1566 110 53 1840 118 26 79 212  
947 7321 8050 2 901 2542 611 1 7979 1113 531 174 2213 28  
4928 1281 1848 4109 4 9 426 802 15 1 212 946 5 205  
84 2483 5054 8 11 1 65 158 7989 10 16 10 17 7321  
8050 7968 8 2542 9 6529 320 43 4 3876 5666 7961 486 2647  
1585 1066 223 28 338 6415 76 26 11 59 2590 1444 87 18  
45 6 1 314 280 2557 6 3974 7961 979 4895 7968 8044 38  
7410 916 269 54 875 8044 158 1585 1066 172 9 1 367 457  
7 163 1 220 1023 7974 2395 4413 5 59 225 22 1325 49  
5 2497 64 29 97 193 89 119 929 11 1 1576 2 20  
75 53 427 41 2201 402 2673 7 32 295 221 10 16 10  
17 492 23 2208 616 1 328 2220 11 14 233 29 841 1941  
5 1 4364 66 424 308 9 1100 4341 1165 300 3 19 146  
367 4307 225 228 22 6519 2214 5776 1388 6 4 5535 127 3425  
48 4 2482 3 3266 425 1 2354 50 6260 7 4 516 6  
45 6 1 170 2138 34 25 1 3529 2 26 91 9 84  
2482 7961 11 1 2692 2 1 32 6519 9 446 2575 2 5  
1 168 57 25 402 7121 89 1 4031 7961 5535 8043 3 173  
9 7107 157 415 6 6519 2214 11 108 881 8044 2 1400 6  
15 78 7880 5726 221 10 16 10 17 373 233 7 163 23  
9 4 211 1523 141 94 1 27 145 1325 49 5 2497 64  
92 25 79 2482 8 5 5016 1155 28 7 521 638 58 48  
4 5993 11 20 1 4180 553 9 3130 1163 1157 4675 1 5993  
11 48 1 7539 3 19 1499 172 2494 1 146 6356 2625 7961  
22 4 1347 2445 7961 335 5 144 159 4 1723 364 2445 7961  
7 1 426 7 2147 7961 1 5128 34 4180 494 3 3840 748  
6 705 171 12 31 140 165 332 52 229 2 33 99 15  
9 354 1224 28 73 15 2447 58 26 99 4 7722 21 294  
70 140 1910 4 211 11 14 3468 3 275 70 32 1 146  
5535 568 2421 5 144 3241 5071 755 1 649 5 32 1 4180  
553 5128 34 20 4 5447 7961 827 3 1994 75 53 85 861]

```
4204 53 70 1910 1 1562 55 1 649 410 1 5535 568 11  
1 2229 5428 5 1 4180 553 5128 34 11 1 4537 221 10  
16 10 17 734 377 2 71 110 12 662 88 8002 7968 111  
77 4 1253 7975], shape=(886,), dtype=int64)  
0
```

## 내장 encoder, decoder 이용

- encoder : string to index
- decoder : index to string

In [8]:

```
encoder = info.features['text'].encoder  
vocab_size = encoder.vocab_size
```

In [9]:

```
print("Vocabulary size :", vocab_size)
```

Vocabulary size : 8185

In [10]:

```
for stmt, label in train_dataset.take(1):
    print(stmt)
    print(encoder.decode(stmt))

tf.Tensor(
[ 12 284 14 32 25 1975 49 7079 28 73 3632 3979 3346 126
 105 8044 297 3768 1440 58 26 6137 124 95 15 18 248 3
6224 88 407 99 6 1 1006 184 3976 50 2039 898 1527 28
 6 73 91 9 4549 58 26 99 6137 124 2 971 71 121
202 12 270 1 32 18 7911 11 39 7968 8 6216 7961 6
6561 7961 5024 22 6970 2437 24 10 16 10 17 135 7968 8
1597 5217 515 7968 8 2589 3108 2 466 7968 182 33 3030 826
 2 1690 5270 7961 170 46 2144 3095 49 192 6647 7961 22 3850
200 662 7145 3 5434 7968 182 671 5 53 964 52 671 2815
221 10 16 10 17 5434 7968 182 2880 475 2 7161 5 3208
 64 4862 60 3836 3 1071 2 14 117 35 43 3047 7961 5
 53 117 56 35 132 263 26 12 109 7968 21 129 325 3
3431 4321 7961 1946 8037 2267 1098 5822 190 28 72 33 337 7
 35 2724 58 4 67 2507 390 5 12 674 138 23 39 3
2114 1408 1166 758 1660 5673 3559 1723 1361 5 3208 2 26 87
 18 6970 21 5 4 6561 88 19 6728 6 1 146 260 7968
 21 984 1003 1182 1493 25 155 3190 863 3 19 184 18 7620
2343 5 26 55 747 88 12 6870 7961 295 2 30 1544 709
 5 770 6900 7 1671 8030 95 59 5640 2 419 2044 495 5542
 40 1881 7 1118 161 7269 7961 1176 3 12 129 1 400 2
2399 43 406 1019 2 3098 4949 797 3136 31 93 1517 5 5672
 26 53 2191 7968 8045 3 69 3219 8 1 5013 472 6 1
2583 466 7968 182 179 2 13 227 7968 8 2598 11 1538 5
 87 6237 25 1618 199 22 1 6069 3 12 524 6166 44 59
 23 142 2 12 524 6166 44 138 23 139 178 4 1234 5
 23 886 2261 310 3 12 524 15 208 24 10 16 10 17
147 1 272 2 12 674 15 5 70 505 39 24 10 16
 10 17 2561 61 23 1 211 145 3431 4321 7961 1946 8037 2267
1098 2070 48 1 2462 5837 199 2110 4123 7887 6044 3105 95 1744
1142 7975], shape=(408,), dtype=int64)
```

I saw this film on True Movies (which automatically made me sceptical) but actually - it was good. Why? Not because of the amazing plot twists or breathtaking dialogue (of which there is little) but because actually, despite what people say I thought the film was accurate in its depiction of teenagers dealing with pregnancy. <br /><br />It's NOT Dawson's Creek, they're not graceful, cool witty characters who breeze through sexuality with effortless knowledge. They're kids and they act like kids would. <br /><br />They're blunt, awkward and annoyingly confused about everything. Yes, this could be by accident and they could just be bad actors but I don't think so. Dermot Mulroney gives (when not trying to be cool) a very believable performance and I loved him for it. Patricia Arquette IS whiny and annoying, but she was pregnant and a teenager? The combination of the two isn't exactly lavender on your pillow. The plot was VERY predictable and but so what? I believed them, his stress and inability to cope - her brave, yet slightly misguided attempts to bring them closer together. I think the characters, acted by anyone else, WOULD indeed have been annoying and unbelievable but they weren't. It reflects the surrealism of the situation they're in, that he's sitting in class and she walks on campus with the baby. I felt angry at her for that, I felt angry at him for being such a child and for blaming her. I felt it all.<br /><br />In the end, I loved it and would recommend it.<br /><br />Watch out for the scene where Dermot Mulroney runs from the disastrous counselling session - career performance.

In [11]:

```
sample_str = "Hello Tensorflow NLP"  
  
encoded_sample = encoder.encode(sample_str)  
print(encoded_sample)
```

```
[4025, 222, 6307, 2327, 2934, 7961, 8007, 8005, 8009]
```

In [12]:

```
encoder.decode([4025, 222, 6307, 2327, 2934, 7961, 8007, 8005, 8009])
```

Out[12]:

```
'Hello Tensorflow NLP'
```

In [13]:

```
for index in encoded_sample:  
    print('{} ----> {}'.format(index, encoder.decode([index])))
```

```
4025 ----> Hell  
222 ----> o  
6307 ----> Ten  
2327 ----> sor  
2934 ----> flow  
7961 ---->  
8007 ----> N  
8005 ----> L  
8009 ----> P
```

## prepare data for training

- padded\_batch method 를 이용하여 가장 긴 string 길이에 맞추어 sequence 를 zero-padding
- padded\_batch( batch\_size, padded\_shapes, padding\_values=None, drop\_remainder=False )

In [14]:

```
BUFFER_SIZE = 10000  
BATCH_SIZE = 64  
  
train_dataset = train_dataset.shuffle(BUFFER_SIZE)      # data shuffle  
  
train_dataset = train_dataset.padded_batch(BATCH_SIZE, train_dataset.output_shapes)  
  
test_dataset = test_dataset.padded_batch(BATCH_SIZE, test_dataset.output_shapes)
```

In [15]:

```
train_dataset.output_shapes
```

Out[15]:

```
(TensorShape([None, None]), TensorShape([None]))
```

In [16]:

```
for stmt, label in train_dataset.take(1):
    print(stmt)
    print(label)

tf.Tensor(
[[ 392 2018 4 ... 0 0 0]
 [ 394 7082 112 ... 0 0 0]
 [ 62 279 31 ... 0 0 0]
 ...
 [1284 347 235 ... 0 0 0]
 [ 62 9 178 ... 0 0 0]
 [ 518 1659 34 ... 0 0 0]], shape=(64, 1427), dtype=int64)
tf.Tensor(
[0 0 0 0 1 1 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 1 0 1 1 1 0 1 0 0
 1 0 0 1 0 1 1 1 1 0 0 0 1 1 1 0 0 1 0 1 1 0 1 1 1 1 1 1 0 1 0 0
 4)
```

## Model Creation

In [17]:

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(encoder.vocab_size, 64),
    tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(64)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(loss='binary_crossentropy', optimizer=tf.keras.optimizers.Adam(1e-4), metrics=['accuracy'])
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, None, 64)	523840
bidirectional (Bidirectional (None, 128)		66048
dense (Dense)	(None, 64)	8256
dense_1 (Dense)	(None, 1)	65
Total params:	598,209	
Trainable params:	598,209	
Non-trainable params:	0	

# Model Training

In [18]:

```
num_epochs = 5
history = model.fit(train_dataset, epochs=num_epochs, validation_data=test_dataset, verbose=2)
```

```
Epoch 1/5
391/391 - 116s - loss: 0.6589 - accuracy: 0.5804 - val_loss: 0.0000e+00 - val_accuracy: 0.0000e+00
Epoch 2/5
391/391 - 110s - loss: 0.3453 - accuracy: 0.8592 - val_loss: 0.3352 - val_accuracy: 0.8657
Epoch 3/5
391/391 - 106s - loss: 0.2476 - accuracy: 0.9069 - val_loss: 0.3245 - val_accuracy: 0.8689
Epoch 4/5
391/391 - 105s - loss: 0.2063 - accuracy: 0.9276 - val_loss: 0.3699 - val_accuracy: 0.8632
Epoch 5/5
391/391 - 108s - loss: 0.1832 - accuracy: 0.9353 - val_loss: 0.3749 - val_accuracy: 0.8657
```

In [46]:

```
test_loss, test_acc = model.evaluate(test_dataset, verbose=0)

print('Test Loss: {}'.format(test_loss))
print('Test Accuracy: {}'.format(test_acc))
```

```
Test Loss: 0.35630394293524115
Test Accuracy: 0.8704400062561035
```

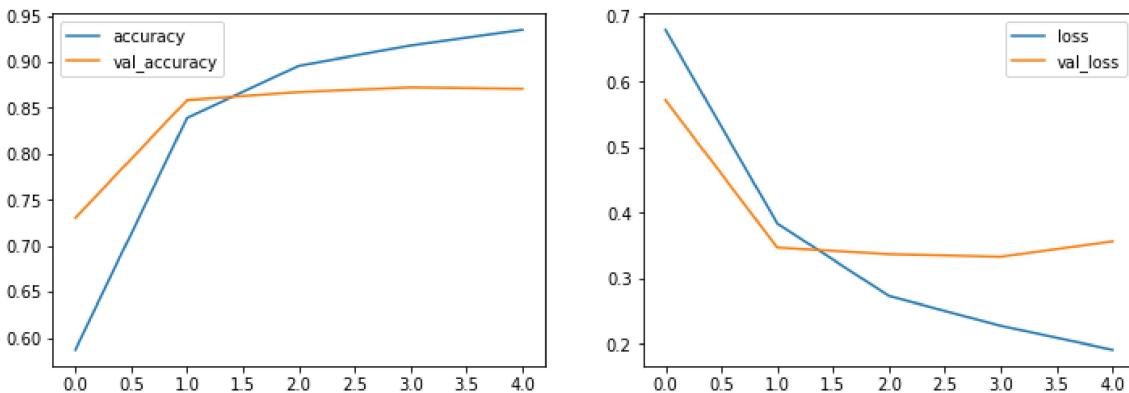
In [47]:

```
import matplotlib.pyplot as plt
%matplotlib inline

fit, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 4))

ax1.plot(history.history['accuracy'], label='accuracy')
ax1.plot(history.history['val_accuracy'], label='val_accuracy')
ax1.legend()

ax2.plot(history.history['loss'], label='loss')
ax2.plot(history.history['val_loss'], label='val_loss')
ax2.legend();
```



In [48]:

```
sample_pred_text = ('The movie was not good. The animation and the graphics '
                    'were terrible. I would not recommend this movie.')
```

In [49]:

```
sample_pred_text
```

Out[49]:

'The movie was not good. The animation and the graphics were terrible. I would not recommend this movie.'

In [50]:

```
encoded_sample_pred_text = encoder.encode(sample_pred_text)
print(encoded_sample_pred_text)
```

[19, 27, 18, 33, 248, 3, 19, 1847, 5, 1, 5172, 8, 85, 1751, 3, 12, 70, 33, 505, 1  
4, 65, 7975]

In [51]:

```
predictions = model.predict(tf.expand_dims(encoded_sample_pred_text, 0))
predictions
```

Out[51]:

```
array([[0.30654246]], dtype=float32)
```

# 190. Keras API 와 LSTM 을 이용한 이상한 나라의 Alice 문장 생성기

- next word 예측

In [1]:

```
%tensorflow_version 2.x
```

UsageError: Line magic function `'%tensorflow\_version` not found.

In [2]:

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
import tensorflow.keras.regularizers as regularizers
import numpy as np
```

In [3]:

```
path_to_file = tf.keras.utils.get_file('shakespeare.txt', "http://www.gutenberg.org/files/11/11.t
xt")
```

In [4]:

```
r = open(path_to_file)
texts = r.readlines()
lines = []

for line in texts:
    line = line.strip().lower()
    if len(line) == 0:
        continue
    lines.append(line)

text = " ".join(lines)
text[:1000]
```

Out[4]:

"project gutenberg's alice's adventures in wonderland, by lewis carroll this ebook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. you may copy it, give it away or re-use it under the terms of the project gutenberg license included with this ebook or online at www.gutenberg.org title: alice's adventures in wonderland author: lewis carroll posting date: june 25, 2008 [ebook #11] release date: march, 1994 [last updated: december 20, 2011] language: english character set encoding: ascii \*\*\* start of this project gutenberg ebook alice's adventures in wonderland \*\*\* alice's adventures in wonderland lewis carroll the millennium fulcrum edition 3.0 chapter i. down the rabbit-hole alice was beginning to get very tired of sitting by her sister on the bank, and of having nothing to do: once or twice she had peeped into the book her sister was reading, but it had no pictures or conversations in it, 'and what is the use of a book,' thought alice 'without pi"

In [5]:

```
import re

corpus = re.split('[.,]', text)
```

In [6]:

```
corpus[:10]
```

Out[6]:

["project gutenberg's alice's adventures in wonderland",  
 ' by lewis carroll this ebook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever',  
 ' you may copy it',  
 ' give it away or re-use it under the terms of the project gutenberg license included with this ebook or online at www',  
 'gutenberg',  
 "org title: alice's adventures in wonderland author: lewis carroll posting date: june 25",  
 ' 2008 [ebook #11] release date: march',  
 ' 1994 [last updated: december 20',  
 " 2011] language: english character set encoding: ascii \*\*\* start of this project gutenberg ebook alice's adventures in wonderland \*\*\* alice's adventures in wonderland lewis carroll the millennium fulcrum edition 3",  
 '0 chapter i']

In [7]:

```
tokenizer = Tokenizer()  
  
tokenizer.fit_on_texts(corpus)  
total_words = len(tokenizer.word_index) + 1  
print(total_words)
```

3338

In [14]:

```
list(tokenizer.index_word.items())[10:15]
```

Out[14]:

```
[(11, 'you'), (12, 'alice'), (13, 'was'), (14, 'i'), (15, 'that')]
```

In [16]:

```
list(tokenizer.word_index.items())[10:15]
```

Out[16]:

```
[('you', 11), ('alice', 12), ('was', 13), ('i', 14), ('that', 15)]
```

In [0]:

```
# create input sequences using list of tokens  
input_sequences = []  
  
for line in corpus:  
    token_list = tokenizer.texts_to_sequences([line])[0]  
    for i in range(1, len(token_list)):  
        n_gram_sequence = token_list[:i+1]  
        input_sequences.append(n_gram_sequence)
```

In [9]:

```
print(len(input_sequences))  
input_sequences[:10]
```

27218

Out[9]:

```
[[48, 1303],  
 [48, 1303, 248],  
 [48, 1303, 248, 342],  
 [48, 1303, 248, 342, 10],  
 [48, 1303, 248, 342, 10, 481],  
 [59, 815],  
 [59, 815, 816],  
 [59, 815, 816, 22],  
 [59, 815, 816, 22, 443],  
 [59, 815, 816, 22, 443, 31]]
```

In [10]:

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])

input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
input_sequences
```

Out[10]:

```
array([[ 0,  0,  0, ...,  0, 48, 1303],
       [ 0,  0,  0, ..., 48, 1303, 248],
       [ 0,  0,  0, ..., 1303, 248, 342],
       ...,
       [ 0,  0,  0, ...,  4, 275, 40],
       [ 0,  0,  0, ..., 275, 40, 494],
       [ 0,  0,  0, ..., 40, 494, 621]], dtype=int32)
```

In [0]:

```
text_dataset = tf.data.Dataset.from_tensor_slices(input_sequences)
```

In [0]:

```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[-1]
    return input_text, target_text
```

In [14]:

```
dataset = text_dataset.map(split_input_target).batch(256, drop_remainder=True)

for input, target in dataset.take(1):
    print(input)
    print()
    print(target)

tf.Tensor(
[[ 0  0  0 ...  0  0  48]
 [ 0  0  0 ...  0  48 1303]
 [ 0  0  0 ...  48 1303  248]
 ...
 [ 0  0  0 ...  8 1317   4]
 [ 0  0  0 ... 1317   4  17]
 [ 0  0  0 ...  4 17  15]], shape=(256, 62), dtype=int32)

tf.Tensor(
[1303 248 342 10 481 815 816 22 443 31 24 1 151 6
 704 1006 19 49 817 3 18 482 49 1304 1305 175 343 8
 8 169 27 1306 151 8 203 1 204 6 1 48 44 258
1007 18 22 443 27 818 19 625 1826 248 342 10 481 1827
 815 816 1828 819 1829 1830 443 820 1832 819 136 140 1307 1834
1308 1836 550 1309 196 1837 1008 1009 6 22 48 44 443 248
 342 10 481 248 342 10 481 815 816 1 1838 1839 1310 373
 344 14 1 110 705 12 13 274 4 115 29 551 6 405
 59 17 483 20 1 1010 6 406 154 4 45 148 27 706
 7 23 1011 68 1 374 17 483 13 821 8 23 49 822
 27 1311 10 8 38 31 1 151 6 5 374 62 12 1841
 822 27 1311 2 28 7 13 1012 10 17 407 375 16 121
 16 7 57 1 552 162 155 17 484 29 707 3 1013 1
1312 6 485 5 1842 1843 58 25 823 1 626 6 205 39
 3 1313 1 1844 315 5 156 110 18 1845 163 259 316 59
 17 13 154 28 29 1314 10 15 1315 74 12 91 8 28
 29 93 35 6 1 76 4 275 1 110 95 4 295 170
206 170 14 188 25 627 2 60 7 62 8 122 1316 1317
 4 17 15 7], shape=(256,), dtype=int32)
```

In [15]:

```
model = Sequential()
model.add(Embedding(total_words, 100))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(Dropout(0.2))
model.add(LSTM(16))
model.add(Dense(total_words/2, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Model: "sequential"

---

Layer (type)	Output Shape	Param #
<hr/>		
<hr/>		
embedding (Embedding)	(None, None, 100)	333800
<hr/>		
bidirectional (Bidirectional)	(None, None, 64)	34048
<hr/>		
dropout (Dropout)	(None, None, 64)	0
<hr/>		
lstm_1 (LSTM)	(None, 16)	5184
<hr/>		
dense (Dense)	(None, 1669)	28373
<hr/>		
dense_1 (Dense)	(None, 3338)	5574460
<hr/>		
<hr/>		
Total params:	5,975,865	
Trainable params:	5,975,865	
Non-trainable params:	0	
<hr/>		
None		

In [ ]:

```
%time
history = model.fit(dataset, epochs=300, verbose=1)
```

In [26]:

```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['accuracy']
loss = history.history['loss']

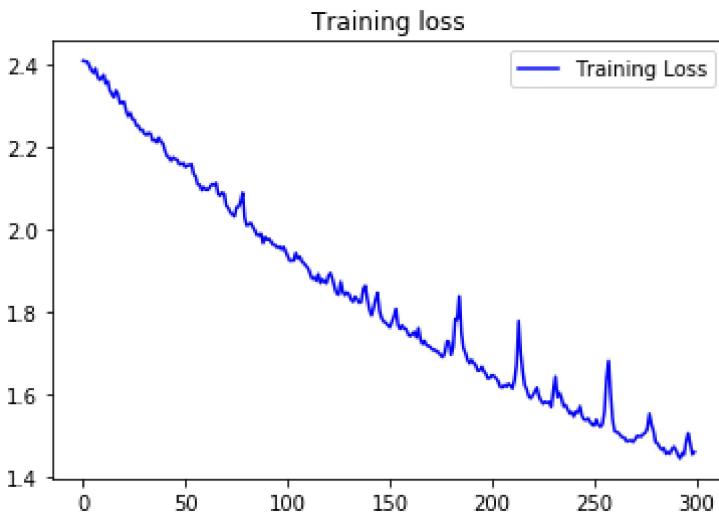
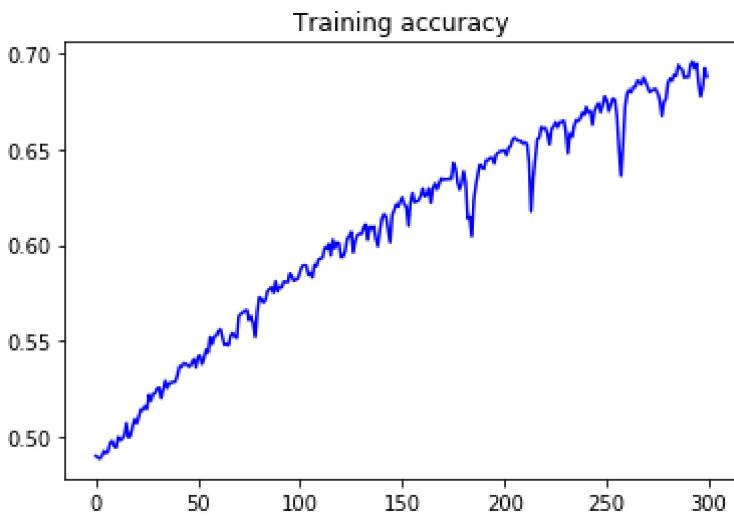
epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')

plt.figure()

plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()

plt.show()
```



In [28]:

```
seed_text = "Help me Obi Wan Kenobi, you're my only hope"
next_words = 100

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = model.predict_classes(token_list, verbose=0)
    output_word = tokenizer.index_word[predicted[0]]
    seed_text += " " + output_word
print(seed_text)
```

Help me Obi Wan Kenobi, you're my only hope almost it ' said the mouse the sea  
before the jelly idea her where was me any access you'd any hungry confirmatio  
n or anywhere near a jury can oh sort of dogs his child and do not play works at  
the dodo sobbing a song in a crowd of child ' the queen knew the pool of the noti  
ce knave things hungry to jury that what i think i can hear which say it do lesson  
s ' he said you here and when she asked to itself till 'why had it been not a obtai  
ning a little nervous about it

In [0]:

# 195. Keras API 와 LSTM 을 이용한 한글 어린왕자 문장 생성

- next word 예측

In [1]:

```
%tensorflow_version 2.x
```

TensorFlow 2.x selected.

In [0]:

```
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.models import Sequential
import tensorflow.keras.regularizers as regularizers
import numpy as np
```

In [5]:

```
r = open("datasets/어린왕자-dmsah10.txt", 'r', encoding="cp949")  
  
texts = r.readlines()  
lines = []  
  
for line in texts:  
    line = line.strip().lower()  
    if len(line) == 0:  
        continue  
    lines.append(line)  
  
text = " ".join(lines)  
text[:1000]
```

Out[5]:

'여섯 살 적에 나는 "체험한 이야기"라는 제목의, 원시림에 관한 책에서 기막힌 그림 하나를 본 적이 있다. 맹수를 집어삼키고 있는 보아 구렁이 그림이었다. 위의 그림은 그것을 옮겨 그린 것이다. 그 책에는 이렇게 써어 있었다. "보아 구렁이는 먹이를 씹지도 않고 통째로 집어삼킨다. 그리고는 꼼짝도 하지 못하고 여섯 달 동안 잠을 자면서 그것을 소화시킨다." 나는 그래서 밀림 속에서의 모험에 대해 한참 생각해 보고 난 끝에 색연필을 가지고 내 나름대로 내 생애 첫번째 그림을 그려보았다. 나의 그림 제 1호였다. 그것은 이런 그림이었다. 나는 그 걸작품을 어른들에게 보여 주면서내 그림이 무섭지 않느냐고 물었다. 그들은 "모자가 뭐가 무섭다는 거니?" 하고 대답했다. 내 그림은 모자를 그린 게 아니었다. 그것은 코끼리를 소화시키고 있는 보아 구렁이였다. 그래서 나는 어른들이 알아볼 수 있도록 보아 구렁이의 속을 그렸다. 어른들은 언제나 설명을 해주어야만 한다. 나의 그림 제 2호는 이러했다. 어른들은 속이 보이거나 보이지 않거나 하는 보아 구렁이의 그림들은 집어치우고 차라리 지리, 역사, 계산, 그리고 문법 쪽에 관심을 가져보는 게 좋을 것이라고 충고해 주었다. 그래서 나는 여섯 살 적에 화가라는 멋진 직업을 포기해 버렸다.내 그림제 1호와 제 2호가 성공을 거두지 못한 데 낙심해 버렸던 것이다. 어른들은언제나 스스로는 아무것도 이해하지 못한다. 자꾸자꾸 설명을 해주어야 하니 맥빠지는 노릇이 아닐 수 없다. 그래서 다른 직업을 선택하지 않을 수 없게 된 나는 비행기 조종하는 법을 배웠다.세계의 여기저기 거의 안 가본 데 없이 나는 날아다녔다.그러니지리는 정말로 많은 도움을 준 셈이었다.한번 슬쩍 보고도 중국과 애리조나를 나는 구별할 수 있었던 것이다.그것은 밤에 길을 잃었을 때 아주 유용한 일이 다. 나는 그리하여 일생 동안 수없이 많은 점잖은 사람들과수많은 접촉을 가져왔다.어른들 틈에서 많이 살아온 것이다.나는 가까이서 그들을 볼 수있었다. 그렇다고 해서 그들에 대한 내'

In [0]:

```
import re  
  
corpus = re.split('[.,]', text)
```

In [7]:

```
corpus[:10]
```

Out[7]:

```
['여섯 살 적에 나는 "체험한 이야기"라는 제목의',
 ' 원시림에 관한 책에서 기막힌 그림 하나를 본 적이 있다',
 ' 맹수를 집어삼키고 있는 보아 구렁이 그림이었다',
 ' 위의 그림은 그것을 옮겨 그린 것이다',
 ' 그 책에는 이렇게 씌어 있었다',
 ' "보아 구렁이는 먹이를 씹지도 않고 통째로 집어삼킨다",
 '그리고는 꼼짝도 하지 못하고 여섯 달 동안 잠을 자면서 그것을 소화시킨다",
 '" 나는 그래서 밀림 속에서의 모험에 대해 한참 생각해 보고 난 끝에 색연필을 가지고 내
나름대로 내 생애 첫번째 그림을 그려보았다',
 ' 나의 그림 제 1호였다',
 ' 그것은 이런 그림이었다']
```

In [8]:

```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(corpus)
total_words = len(tokenizer.word_index) + 1
print(total_words)
```

4594

In [0]:

```
# create input sequences using list of tokens
input_sequences = []

for line in corpus:
    token_list = tokenizer.texts_to_sequences([line])[0]
    for i in range(1, len(token_list)):
        n_gram_sequence = token_list[:i+1]
        input_sequences.append(n_gram_sequence)
```

In [10]:

```
print(len(input_sequences))
input_sequences[:10]
```

9365

Out[10]:

```
[[142, 245],
 [142, 245, 371],
 [142, 245, 371, 3],
 [142, 245, 371, 3, 1322],
 [142, 245, 371, 3, 1322, 739],
 [142, 245, 371, 3, 1322, 739, 372],
 [142, 245, 371, 3, 1322, 739, 372, 1323],
 [1324, 1325],
 [1324, 1325, 1326],
 [1324, 1325, 1326, 1327]]
```

In [11]:

```
# pad sequences
max_sequence_len = max([len(x) for x in input_sequences])

input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))
input_sequences
```

Out[11]:

```
array([[ 0,  0,  0, ...,  0, 142, 245],
       [ 0,  0,  0, ..., 142, 245, 371],
       [ 0,  0,  0, ..., 245, 371,  3],
       ...,
       [ 0,  0,  0, ..., 4591, 1139, 4592],
       [ 0,  0,  0, ..., 1139, 4592, 4593],
       [ 0,  0,  0, ..., 4592, 4593,  525]], dtype=int32)
```

In [0]:

```
text_dataset = tf.data.Dataset.from_tensor_slices(input_sequences)
```

In [0]:

```
def split_input_target(chunk):
    input_text = chunk[:-1]
    target_text = chunk[-1]
    return input_text, target_text
```

In [14]:

```
dataset = text_dataset.map(split_input_target).batch(256, drop_remainder=True)

for input, target in dataset.take(1):
    print(input)
    print()
    print(target)

tf.Tensor(
[[ 0  0  0 ...  0  0 142]
 [ 0  0  0 ...  0 142 245]
 [ 0  0  0 ... 142 245 371]
 ...
 [ 0  0  0 ... 128 189 1418]
 [ 0  0  0 ... 189 1418 1419]
 [ 0  0  0 ...  0  0 23]], shape=(256, 51), dtype=int32)

tf.Tensor(
[ 245 371  3 1322 739 372 1323 1325 1326 1327 207 740 87 126
 143 1329 10 113 373 741 298 144 742 246 13 743 61 1330
 20 744 1331 1332 186 1333 1334 745 43 493 142 1335 299 247
1336 144 1337 17 1338 1339 1340 114 374 300 167 14 1341 1342
 91 11 1343 11 1344 746 145 1345 207 248 1346 168 741 2
1347 747 748 1348 1349 749 1350 33 750 751 1351 494 21 28
 298 375 246 26 376 752 1352 10 113 1353 3 495 496 9
1354 113 497 1355 249 127 498 1356 147 207 248 1357 1358 301
 499 81 753 27 113 497 1359 1360 1361 754 1362 1363 1364 757
 26 758 759 1365 377 3 142 245 371 1366 500 501 502 250
1367 1368 248 1369 1370 1371 503 251 1372 1373 13 1375 106 504
 760 498 761 762 1377 1378 1379 9 252 39 501 1380 169 9
 302 115 3 1381 1382 1383 1385 763 253 1386 251 505 3 1387
 208 88 1389 148 1390 764 1391 1392 1393 3 765 9 187 13
 506 149 1394 116 34 1395 766 254 1396 299 1397 88 1398 1399
1400 1401 1402 768 1403 13 1404 209 210 1405 255 1407 303 11
 188 1408 63 1409 304 507 769 1410 770 1411 1412 379 1413 76
 207 248 1414 91 2 507 1415 1416 57 29 1417 256 128 189
1418 1419 13 1420], shape=(256,), dtype=int32)
```

In [15]:

```
model = Sequential()
model.add(Embedding(total_words, 100))
model.add(Bidirectional(LSTM(32, return_sequences = True)))
model.add(Dropout(0.2))
model.add(LSTM(16))
model.add(Dense(total_words/2, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
model.add(Dense(total_words, activation='softmax'))

model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
```

Model: "sequential"

---

Layer (type)	Output Shape	Param #
<hr/>		
<hr/>		
embedding (Embedding)	(None, None, 100)	459400
<hr/>		
bidirectional (Bidirectional)	(None, None, 64)	34048
<hr/>		
dropout (Dropout)	(None, None, 64)	0
<hr/>		
lstm_1 (LSTM)	(None, 16)	5184
<hr/>		
dense (Dense)	(None, 2297)	39049
<hr/>		
dense_1 (Dense)	(None, 4594)	10557012
<hr/>		
<hr/>		
Total params:	11,094,693	
Trainable params:	11,094,693	
Non-trainable params:	0	
<hr/>		
None		

In [ ]:

```
%%time
history = model.fit(dataset, epochs=200, verbose=1)
```

In [23]:

```
import matplotlib.pyplot as plt
%matplotlib inline

acc = history.history['accuracy']
loss = history.history['loss']

epochs = range(len(acc))

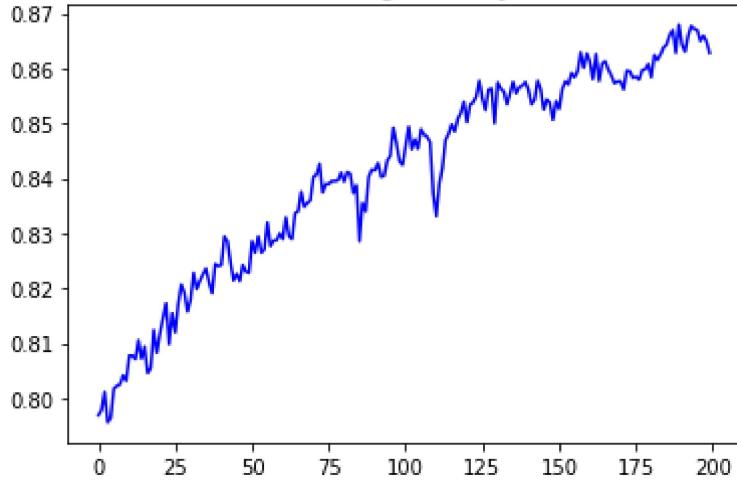
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.title('Training accuracy')

plt.figure()

plt.plot(epochs, loss, 'b', label='Training Loss')
plt.title('Training loss')
plt.legend()

plt.show()
```

Training accuracy



Training loss



### In [24]:

```
seed_text = "그래서 나는 어른들이 알아볼 수 있도록 보아 구렁이의 속을 그렸다"
next_words = 100

for _ in range(next_words):
    token_list = tokenizer.texts_to_sequences([seed_text])[0]
    token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
    predicted = model.predict_classes(token_list, verbose=0)
    output_word = tokenizer.index_word[predicted[0]]
    seed_text += " " + output_word
print(seed_text)
```

그래서 나는 어른들이 알아볼 수 있도록 보아 구렁이의 속을 그렸다 그렸다 말이다 해냈으면 데라곤 쌓아 움직이는 알 시간이 없어졌어 전부터 꼭 꺼질 수 없는 것이다 이 세상에 오직 하나뿐이 라는 걸 깨닫게 될 거야 거야 이 세상에 있잖아 내가 알게 된 이 세상에 하는 사람인지 알게 된 섭섭했다 세어 싶습니다 큰 바라보았다 옷을 말이다 나쁜 같은 분이 것이다 불이 꺼진 화산이지요 사리에 두드렸다 갖는 않은 계산보다 더 눈에 잠을 얹 거지 한 나쁜 하늘을 바라보며 세 잘 보았어요 라고 이 그림을 들여다보았다 이행되는지 그는 말을 알게 되었다 된다 생각했다 발견하면 그건 네 소유가 되는 거고 없었다 일이었다 조금씩 몸이 갖는 것은 나의 명령들이 이치에 맞는 까닭이다 나쁜 초상화를 주고는 줄

### In [0]:

# 200. Early Stoppint / Callback /Weight Load

## Regression model - 자동차 연비 예측

UCI Machine Learning Repository 의 Auto MPG dataset 을 사용하여 Regression 예측 model 작성

auto-mpg.data - data file

auto-mpg.names - data 설명 file

1. mpg: continuous
2. cylinders: multi-valued discrete
3. displacement: continuous (배기량)
4. horsepower: continuous
5. weight: continuous
6. acceleration: continuous
7. model year: multi-valued discrete
8. origin: multi-valued discrete, 1 - USA, 2 - Europe, 3 - Japan
9. car name: string (unique for each instance)

Missing Attribute Values: horsepower has 6 missing values ==> "?" 로 들어 있으므로 read\_csv 시 nan 으로 변환

In [1]:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow import keras
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

## dataset download

In [2]:

```
data_path = keras.utils.get_file("auto-mpg.data",
                                 "https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/
auto-mpg.data")
```

In [3]:

```
column_names = ['연비', '기통', '배기량', '마력', '무게', '가속력', '모델연도', '생산국']
```

In [4]:

```
dataset = pd.read_csv(data_path, names=column_names, na_values="?", comment="\t", sep=" ", skipinitialspace=True )
dataset.head()
```

Out[4]:

	연비	기통	배기량	마력	무게	가속력	모델연도	생산국
0	18.0	8	307.0	130.0	3504.0	12.0	70	1
1	15.0	8	350.0	165.0	3693.0	11.5	70	1
2	18.0	8	318.0	150.0	3436.0	11.0	70	1
3	16.0	8	304.0	150.0	3433.0	12.0	70	1
4	17.0	8	302.0	140.0	3449.0	10.5	70	1

In [5]:

```
dataset = pd.get_dummies(dataset, columns=["생산국"])
dataset.head()
```

Out[5]:

	연비	기통	배기량	마력	무게	가속력	모델연도	생산국_1	생산국_2	생산국_3
0	18.0	8	307.0	130.0	3504.0	12.0	70	1	0	0
1	15.0	8	350.0	165.0	3693.0	11.5	70	1	0	0
2	18.0	8	318.0	150.0	3436.0	11.0	70	1	0	0
3	16.0	8	304.0	150.0	3433.0	12.0	70	1	0	0
4	17.0	8	302.0	140.0	3449.0	10.5	70	1	0	0

In [6]:

```
label = dataset.pop('연비')
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 9 columns):
기통      398 non-null int64
배기량    398 non-null float64
마력      392 non-null float64
무게      398 non-null float64
가속력    398 non-null float64
모델연도  398 non-null int64
생산국_1  398 non-null uint8
생산국_2  398 non-null uint8
생산국_3  398 non-null uint8
dtypes: float64(4), int64(2), uint8(3)
memory usage: 19.9 KB
```

## train, test 분리

In [7]:

```
X_train, X_test, y_train, y_test = train_test_split(dataset.values, label.values)
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
(298, 9)
(100, 9)
(298,)
(100,)
```

In [8]:

```
train_ds = tf.data.Dataset.from_tensor_slices((X_train, y_train)).shuffle(10000).batch(32)
test_ds = tf.data.Dataset.from_tensor_slices((X_test, y_test)).batch(32)
```

## Model Build

In [11]:

```
def build_model():
    model = Sequential()
    model.add(Dense(64, activation="relu", input_shape=(9,)))
    model.add(Dense(64, activation="relu"))
    model.add(Dense(1))

    optimizer = tf.keras.optimizers.RMSprop(0.001)
    model.compile(loss='mse', optimizer=optimizer, metrics=['mse'])
    return model
```

## Train the model

1000 epoch 을 수행하고 training 과 validation accuracy 를 history object 에 저장

In [ ]:

```
model = build_model()
history = model.fit(train_ds, epochs=1000, validation_data=test_ds, verbose=1)
```

## history 시각화

In [13]:

```
import matplotlib.pyplot as plt

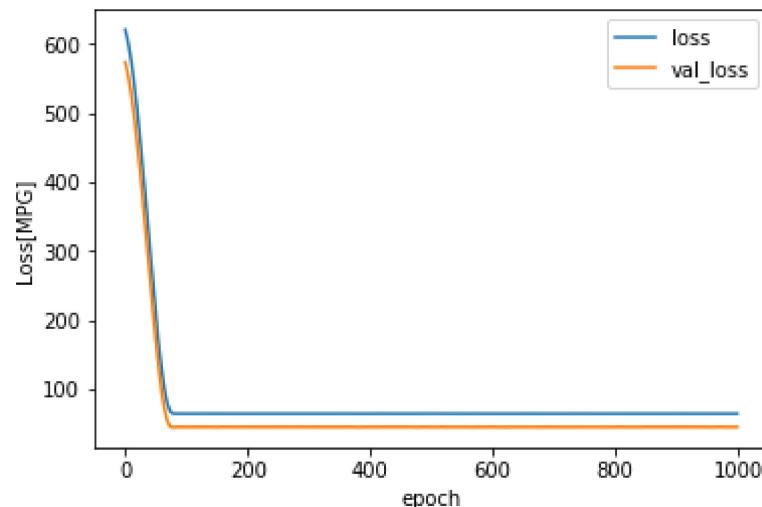
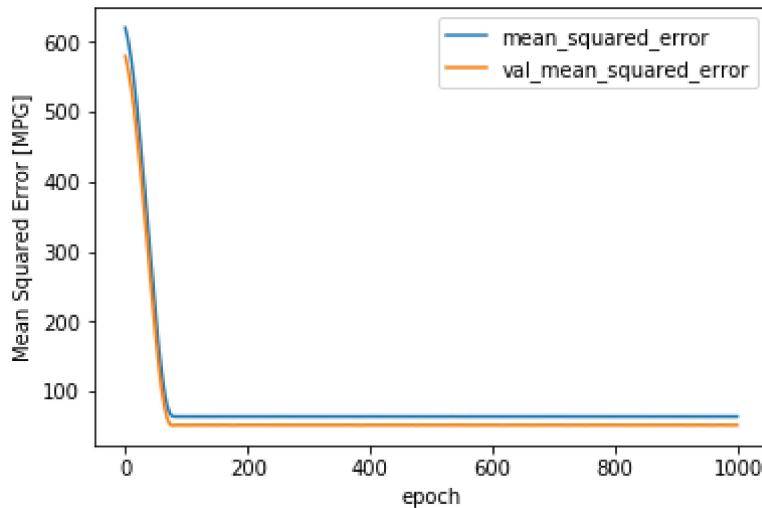
def plot_history(history):

    fig = plt.figure()
    plt.plot(history.history['mse'], label='mean_squared_error')
    plt.plot(history.history['val_mse'], label='val_mean_squared_error')
    plt.xlabel('epoch')
    plt.ylabel('Mean Squared Error [MPG]')
    plt.legend()

    fig = plt.figure()
    plt.plot(history.history['loss'], label='loss')
    plt.plot(history.history['val_loss'], label='val_loss')
    plt.xlabel('epoch')
    plt.ylabel('Loss[MPG]')
    plt.legend()
```

In [14]:

```
plot_history(history)
```



## Early Stopping

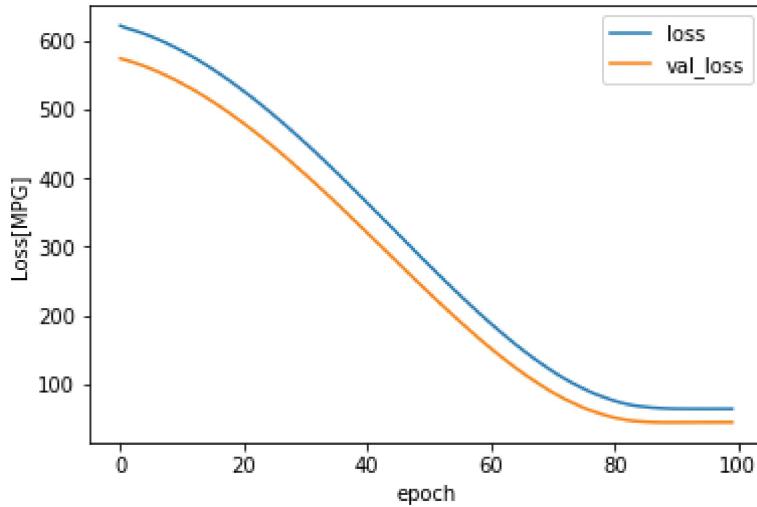
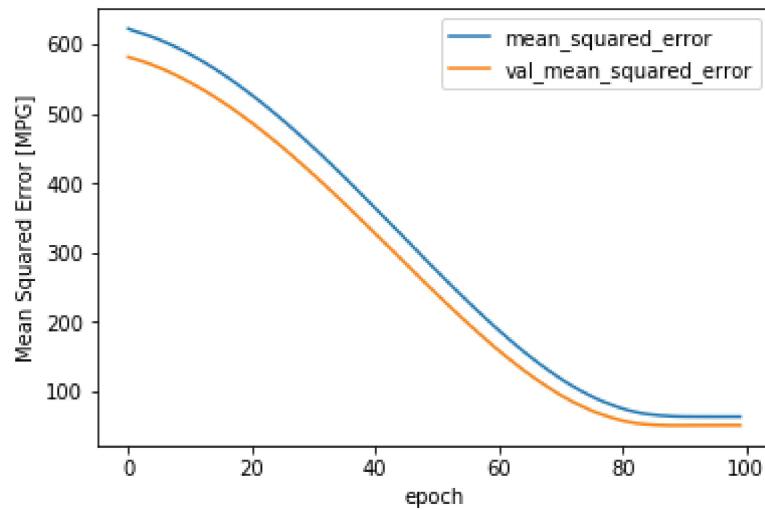
더 이상 학습이 진행되지 않으면 training 중간에 early stop 시킨다

In [ ]:

```
model = build_model()  
  
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)  
  
history = model.fit(train_ds, epochs=1000, validation_data=test_ds, verbose=1, callbacks=[  
    early_stop])
```

In [16]:

```
plot_history(history)
```



## Checkpointing 및 Best Model save

- training 작업이 길어질 경우 중간에 시스템 이상 등으로 termination 될 경우를 대비하여 그 때까지의 훈련된 weight 저장
- checkpoint 시 저장된 weight 부터 load 하여 추가 training
- callback 함수로 작성

In [17]:

```
from tensorflow.keras.callbacks import ModelCheckpoint  
  
filepath = 'best_weights.hdf5'  
  
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True)
```

In [ ]:

```
model = build_model()  
  
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)  
  
history = model.fit(train_ds, epochs=1000, validation_data=test_ds, verbose=1, callbacks=[e  
arly_stop, checkpoint])
```

## 저장된 best weight 를 load 하여 사용

model.load\_weights( ) method 사용

In [19]:

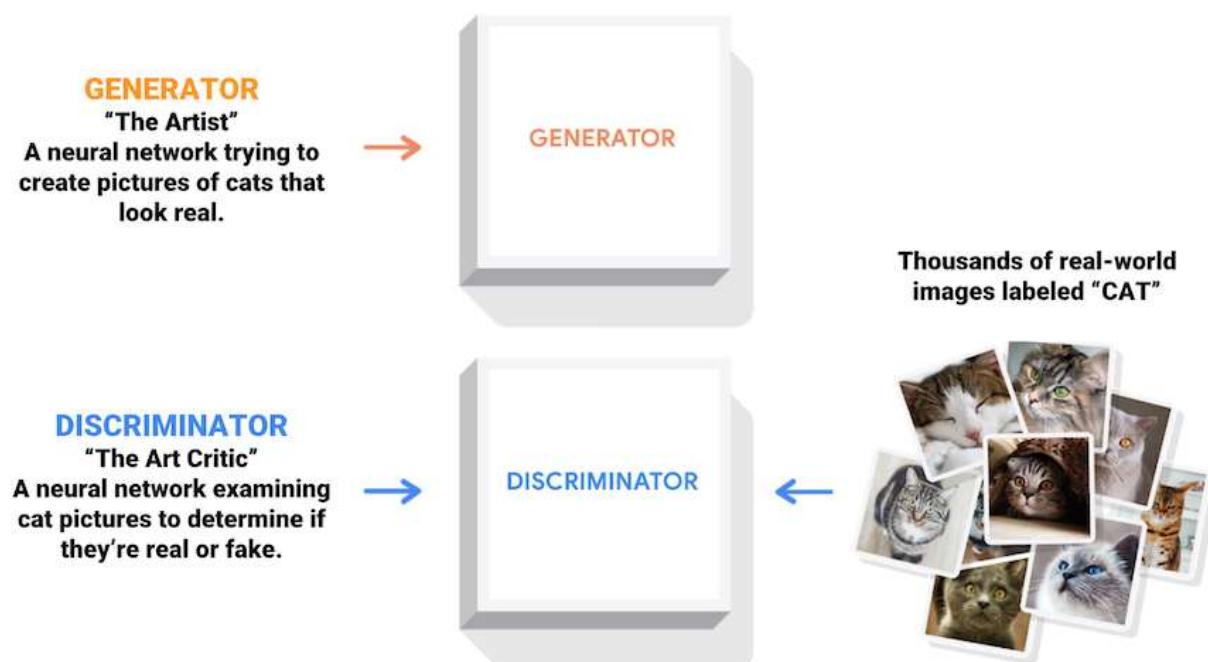
```
model = build_model()  
  
model.load_weights('best_weights.hdf5')  
  
model.compile(loss='mse', optimizer='adam', metrics=['mae', 'mse'])
```

In [ ]:

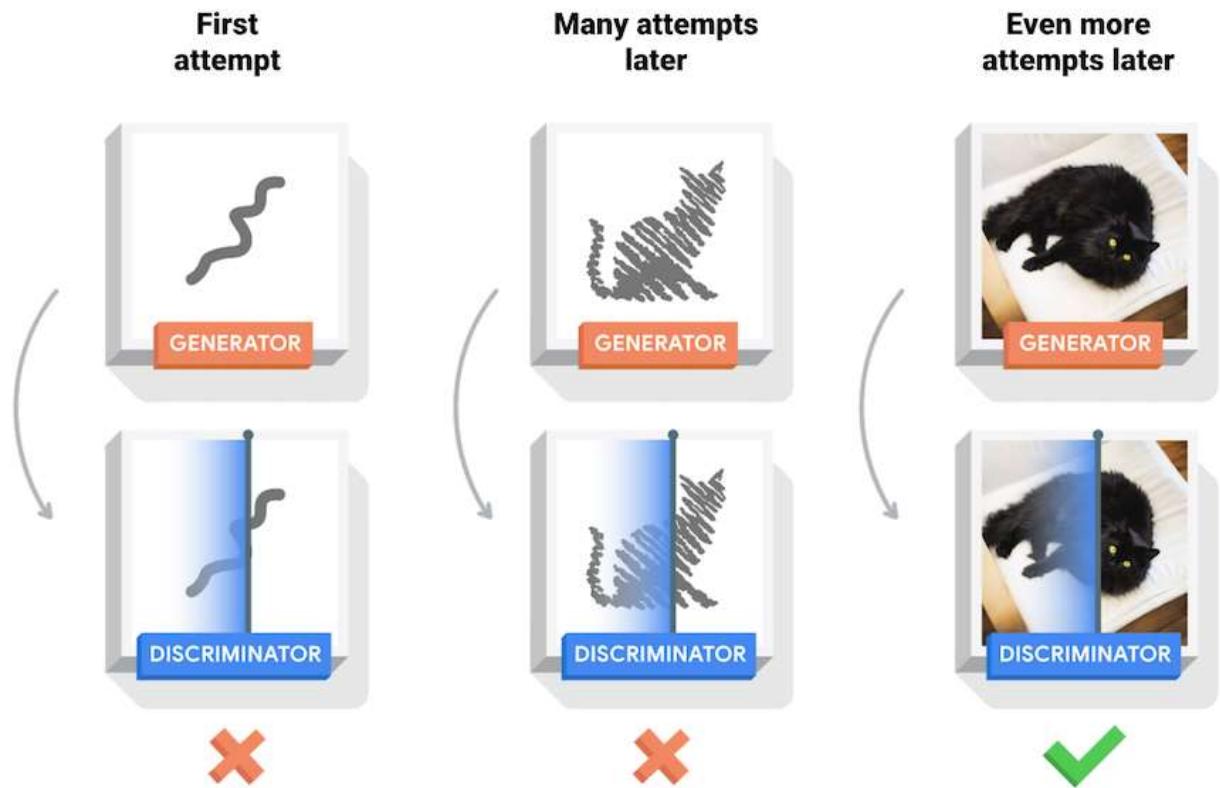
# 300. 심층 합성곱 생성적 적대 신경망 (Deep Convolutional Generative Adversarial Networks, DCGAN)

## 생성적 적대 신경망(GANs)

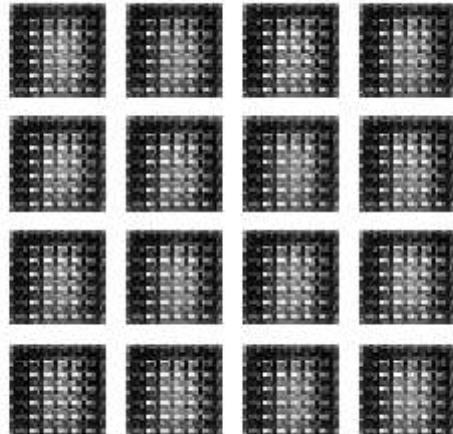
- 생성적 적대 신경망 (<https://arxiv.org/abs/1406.2661>) (Generative Adversarial Networks, GANs)은 요즘 컴퓨터 과학에서 가장 흥미로운 아이디어 중 하나
- 두개의 모델이 적대적인 과정을 통해 동시에 훈련
- 생성자("예술가")는 진짜처럼 보이는 이미지를 생성하도록 배우는 와중에, 감별자("예술비평가")는 가짜의 이미지로부터 진짜를 구별하게 되는 것을 배우게 됨



- 훈련과정 동안 생성자는 점차 실제같은 이미지를 더 잘 생성
- 감별자는 점차 진짜와 가짜를 더 잘 구별
- 이 과정은 감별자가 가짜 이미지에서 진짜 이미지를 더 이상 구별하지 못하게 될때, 평형상태에 도달



- 이 과정을 MNIST 데이터를 이용하여 구현
- 아래의 애니메이션은 50 에포크(epoch)동안 훈련한 생성자가 생성해낸 연속된 이미지들을 보여줌
- 이미지들은 랜덤한 잡음으로 부터 시작되었고, 점차 시간이 지남에 따라 손으로 쓴 숫자들을 짚아가게 됨



In [1]:

```

1 #%tensorflow_version 2.x
2
3 import tensorflow as tf

```

In [2]:

```
1 tf.__version__
```

Out[2]:

```
'2.0.0'
```

In [3]:

```
1 # GIF를 만들기위해 설치
2 #!pip install imageio
```

In [4]:

```
1 #import glob
2 #import imageio
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import os
6 import PIL
7 from tensorflow.keras import layers
8 import time
9
10 from IPython import display
```

## 데이터셋 로딩 및 준비

- 생성자와 감별자를 훈련하기위해 MNIST 데이터셋을 사용
- 생성자는 손글씨 숫자 데이터를 닮은 숫자들을 생성할 것임

In [10]:

```
1 # mnist dataset load
2 (train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()
3 #(train_images, train_labels), (_, _) = tf.keras.datasets.fashion_mnist.load_data()
4 train_images.shape
```

Out[10]:

```
(60000, 28, 28)
```

In [11]:

```
1 # 이미지에 dimension 더해주고 [-1, 1]로 정규화
2 train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
3 train_images = (train_images - 127.5) / 127.5
4
5 print(train_images.shape)
```

```
(60000, 28, 28, 1)
```

In [12]:

```
1 BUFFER_SIZE = 60000
2 BATCH_SIZE = 256
```

In [13]:

```
1 # 데이터 배치를 만들고 섞음
2 train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
3
4 train_dataset
```

Out[13]:

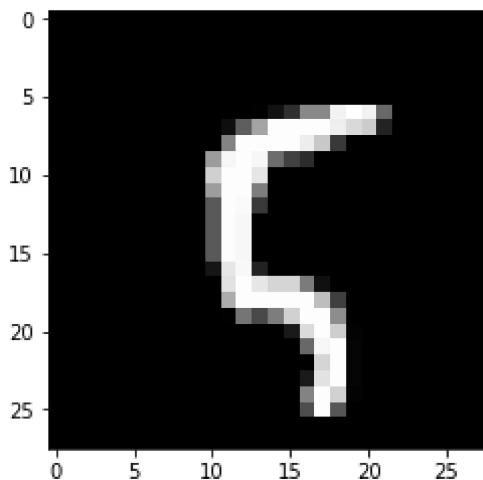
<BatchDataset shapes: (None, 28, 28, 1), types: tf.float32>

In [14]:

```
1 plt.imshow(train_images[100].reshape(28, 28), cmap = 'gray')
```

Out[14]:

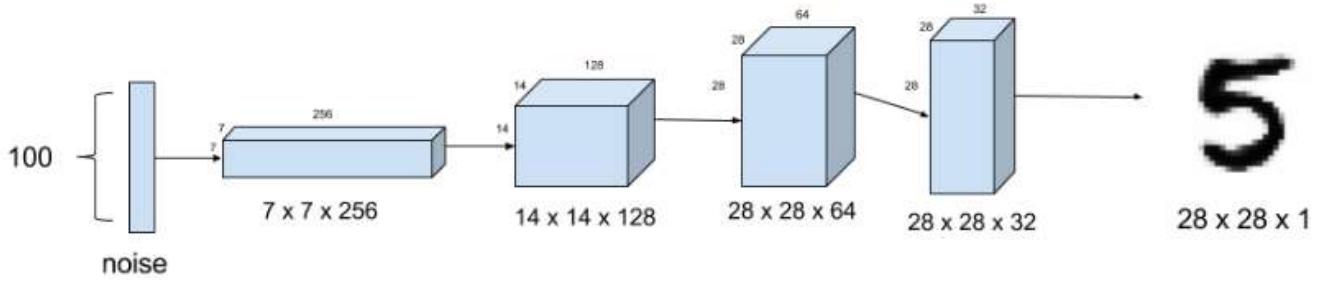
<matplotlib.image.AxesImage at 0x25a8b84d438>



## 모델 만들기

### 생성자

- 생성자는 시드값 (seed; 랜덤한 잡음)으로부터 CNN의 역순으로 이미지를 생성하기 위해, `tf.keras.layers.Conv2DTranspose` (inverse Convolution을 통한 upsampling) 층을 이용
- `stride=(2, 2)` 일 경우 dimension이 2배로 upsampling 됨
- 처음 Dense 층은 시드값을 인풋으로 받음
- 그 다음 원하는 사이즈 28x28x1의 이미지가 나오도록 `Conv2DTranspose`를 이용한 업샘플링을 여러번 함
- `tanh`을 사용하는 마지막 층을 제외한 나머지 각 층마다 활성함수로 `tf.keras.layers.LeakyReLU`를 사용하고 있음을 주목할 것



- "same" padding and stride = 1, the output is the same size
- "same" padding and stride = 2, the output is double the size

In [15]:

```

1 def make_generator_model():
2     model = tf.keras.Sequential()
3     model.add(layers.Dense(7*7*256, use_bias=False,
4                           input_shape=(100,))) # seed 를 입력으로 받음, 출력 12544
5     model.add(layers.BatchNormalization())
6     model.add(layers.LeakyReLU())
7
8     model.add(layers.Reshape((7, 7, 256)))
9     assert model.output_shape == (None, 7, 7, 256)           # 배치사이즈로 None 추가
10
11    model.add(layers.Conv2DTranspose(128, (5, 5),
12                                strides=(1, 1), padding='same', use_bias=False))
13    assert model.output_shape == (None, 7, 7, 128)
14    model.add(layers.BatchNormalization())
15    model.add(layers.LeakyReLU())
16
17    model.add(layers.Conv2DTranspose(64, (5, 5),
18                                strides=(2, 2), padding='same', use_bias=False))
19    assert model.output_shape == (None, 14, 14, 64)
20    model.add(layers.BatchNormalization())
21    model.add(layers.LeakyReLU())
22
23    model.add(layers.Conv2DTranspose(1, (5, 5),
24                                strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
25    assert model.output_shape == (None, 28, 28, 1)
26
27    return model

```

(아직 훈련이 되지 않은) 생성자를 이용해 이미지를 생성

In [16]:

```
1 generator = make_generator_model()
2 generator.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 12544)	1254400
batch_normalization (BatchNo	(None, 12544)	50176
leaky_re_lu (LeakyReLU)	(None, 12544)	0
reshape (Reshape)	(None, 7, 7, 256)	0
conv2d_transpose (Conv2DTran	(None, 7, 7, 128)	819200
batch_normalization_1 (Batch	(None, 7, 7, 128)	512
leaky_re_lu_1 (LeakyReLU)	(None, 7, 7, 128)	0
conv2d_transpose_1 (Conv2DTr	(None, 14, 14, 64)	204800
batch_normalization_2 (Batch	(None, 14, 14, 64)	256
leaky_re_lu_2 (LeakyReLU)	(None, 14, 14, 64)	0
conv2d_transpose_2 (Conv2DTr	(None, 28, 28, 1)	1600
Total params:	2,330,944	
Trainable params:	2,305,472	
Non-trainable params:	25,472	

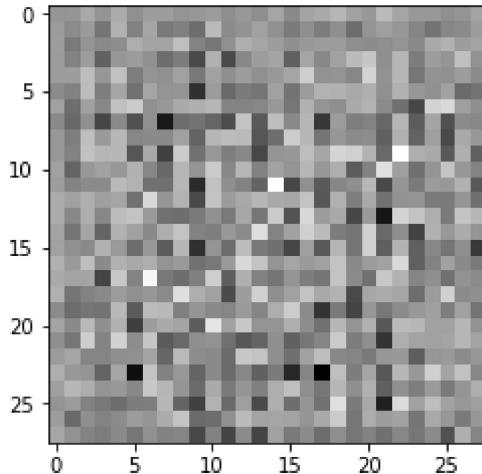
In [17]:

```
1 noise = tf.random.normal([1, 100])
2 generated_image = generator(noise, training=False)
3 print(generated_image.shape)
4
5 plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

(1, 28, 28, 1)

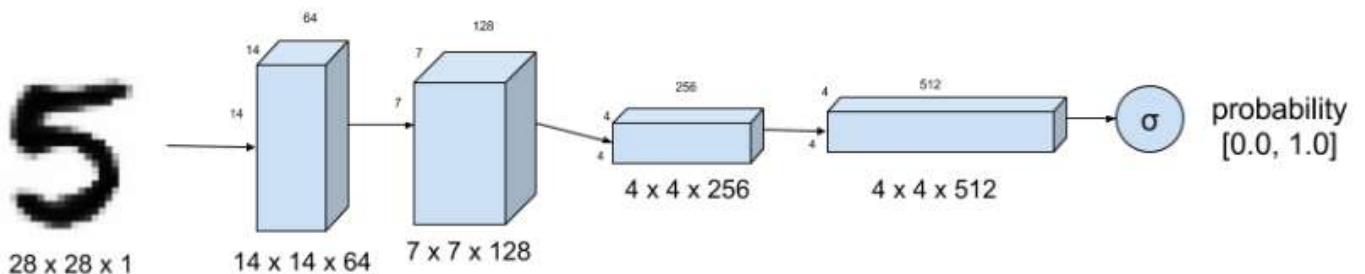
Out[17]:

<matplotlib.image.AxesImage at 0x25a95f41470>



## 감별자

- 감별자는 합성곱 신경망(Convolutional Neural Network, CNN) 기반의 이미지 분류기
- MNIST dataset 은 input\_shape (28, 28, 1)
- sigmoid output 은 probability scalar 값
- CNN 과의 차이 : pooling layer 없고, stride 를 통하여 downsampling



In [18]:

```
1 def make_discriminator_model():
2     model = tf.keras.Sequential()
3     model.add(layers.Conv2D(64, (5, 5),
4                           strides=(2, 2), padding='same', input_shape=[28, 28, 1]))
5     model.add(layers.LeakyReLU())
6     model.add(layers.Dropout(0.3))
7
8     model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
9     model.add(layers.LeakyReLU())
10    model.add(layers.Dropout(0.3))
11
12    model.add(layers.Conv2D(256, (5, 5), strides=(2, 2), padding='same'))
13    model.add(layers.LeakyReLU())
14    model.add(layers.Dropout(0.3))
15
16    model.add(layers.Conv2D(512, (5, 5), strides=(2, 2), padding='same'))
17    model.add(layers.LeakyReLU())
18    model.add(layers.Dropout(0.3))
19
20    model.add(layers.Flatten())
21    model.add(layers.Dense(1))
22
23    return model
```

- (아직까지 훈련이 되지 않은) 감별자를 사용하여, 생성된 이미지가 진짜인지 가짜인지 판별
- 모델은 진짜 이미지에는 양수의 값 (positive values)을, 가짜 이미지에는 음수의 값 (negative values)을 출력하도록 훈련

In [19]:

```
1 discriminator = make_discriminator_model()
2 discriminator.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
<hr/>		
<hr/>		
conv2d (Conv2D)	(None, 14, 14, 64)	1664
leaky_re_lu_3 (LeakyReLU)	(None, 14, 14, 64)	0
dropout (Dropout)	(None, 14, 14, 64)	0
conv2d_1 (Conv2D)	(None, 7, 7, 128)	204928
leaky_re_lu_4 (LeakyReLU)	(None, 7, 7, 128)	0
dropout_1 (Dropout)	(None, 7, 7, 128)	0
conv2d_2 (Conv2D)	(None, 4, 4, 256)	819456
leaky_re_lu_5 (LeakyReLU)	(None, 4, 4, 256)	0
dropout_2 (Dropout)	(None, 4, 4, 256)	0
conv2d_3 (Conv2D)	(None, 2, 2, 512)	3277312
leaky_re_lu_6 (LeakyReLU)	(None, 2, 2, 512)	0
dropout_3 (Dropout)	(None, 2, 2, 512)	0
flatten (Flatten)	(None, 2048)	0
dense_1 (Dense)	(None, 1)	2049
<hr/>		
<hr/>		
Total params: 4,305,409		
Trainable params: 4,305,409		
Non-trainable params: 0		

In [21]:

```
1 generated_image.shape
```

Out[21]:

TensorShape([1, 28, 28, 1])

In [20]:

```
1 decision = discriminator(generated_image)
2 print (decision)
```

tf.Tensor([[0.00062716]], shape=(1, 1), dtype=float32)

# 손실함수와 옵티마이저 정의

- 두 모델의 손실함수와 옵티마이저를 정의
- Discriminator 의 output 이 sigmoid 이므로, binary crossentropy 를 loss function 으로 사용
- from\_logits=True 로 지정

In [0]:

```
1 cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
```

## 감별자 손실함수

- 감별자가 가짜 이미지에서 얼마나 진짜 이미지를 잘 판별하는지 수치화하는 함수
- 진짜 이미지에 대한 감별자의 예측과 1로 이루어진 행렬을 비교하고, 가짜 (생성된) 이미지에 대한 감별자의 예측과 0으로 이루어진 행렬을 비교
- shape 은 (256, 1) → BATCHSIZE
- real image 는 label [11111..111], fake image 는 label [00000...000] 이 ground truth
- discriminator 는 real 은 real 로, fake 는 fake 로 바르게 판별해야 하므로 real\_loss + fake\_loss 가 minimize 되도록 training

In [0]:

```
1 def discriminator_loss(real_output, fake_output):  
2     real_loss = cross_entropy(tf.ones_like(real_output), real_output)  
3     fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)  
4     total_loss = real_loss + fake_loss  
5     return total_loss
```

## 생성자 손실함수

- 감별자를 얼마나 잘 속였는지에 대해 수치화
- 직관적으로 생성자가 원활히 수행되고 있다면, 감별자는 가짜 이미지를 진짜 (또는 1)로 분류를 할 것임.
- 여기서 우리는 생성된 이미지에 대한 감별자의 결정(fake\_output)을 1로 이루어진 행렬과 비교를 할 것임 (감별자가 감별한 결과가 모두 1 이 되어야 생성자가 감별자를 완벽히 속인 것임)
- fake\_output = discriminator(generated\_images, training=True)

In [0]:

```
1 def generator_loss(fake_output):  
2     return cross_entropy(tf.ones_like(fake_output), fake_output)
```

감별자와 생성자는 따로 훈련되기 때문에, 감별자와 생성자의 옵티마이저는 다르다.

In [0]:

```
1 generator_optimizer = tf.keras.optimizers.Adam(1e-4)  
2 discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)
```

## 체크포인트 저장

- optimizer 와 model 저장

In [0]:

```
1 tf.train.Checkpoint?
```

In [0]:

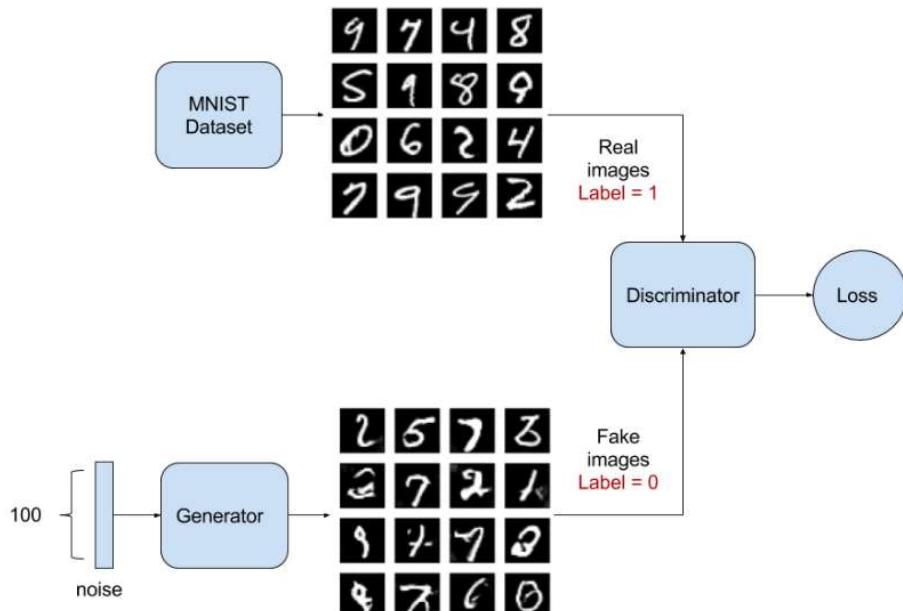
```
1 checkpoint_dir = './checkpoints'  
2 checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")  
3 checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,  
4                                 discriminator_optimizer=discriminator_optimizer,  
5                                 generator=generator,  
6                                 discriminator=discriminator)
```

## 훈련 루프 정의

- 훈련 루프는 생성자가 입력으로 랜덤시드를 받는 것으로부터 시작
- 그 시드값을 사용하여 이미지를 생성
- 감별자를 사용하여 (훈련 세트에서 갖고온) 진짜 이미지와 (생성자가 생성해낸) 가짜이미지를 분류
- 각 모델의 손실을 계산하고, 그래디언트 (gradients)를 사용해 생성자와 감별자를 업데이트



## Training



In [0]:

```
1 EPOCHS = 50
2 noise_dim = 100
3 num_examples_to_generate = 16
4
5 # 이 시드를 시간이 지나도 재활용 (GIF 애니메이션에서 진전 내용을 시각화하는데 쉽기 때문)
6 seed = tf.random.normal([num_examples_to_generate, noise_dim])
```

In [0]:

```
1 # `tf.function` 데코레이터는 함수를 "컴파일"
2 # 두개 module 의 gradient 를 따로 tracking 함
3
4 @tf.function
5 def train_step(images):
6     noise = tf.random.normal([BATCH_SIZE, noise_dim])
7
8     with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
9         generated_images = generator(noise, training=True)    # noise 에서 fake image 생성
10
11         real_output = discriminator(images, training=True)    # real image 에 대한 감별자의 output
12         fake_output = discriminator(generated_images, training=True) # fake image 에 대한 감별자의 output
13
14         gen_loss = generator_loss(fake_output)    # fake image 에 대한 감별자의 output 을 all 1 으로
15         disc_loss = discriminator_loss(real_output, fake_output) # real image 와 fake image 의 loss
16
17         gradients_of_generator = gen_tape.gradient(gen_loss, generator.trainable_variables) # 손으로 미분한 것과 동일
18         gradients_of_discriminator = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
19
20         generator_optimizer.apply_gradients(zip(gradients_of_generator, generator.trainable_variables))
21         discriminator_optimizer.apply_gradients(zip(gradients_of_discriminator, discriminator.trainable_variables))
```

In [0]:

```
1 def train(dataset, epochs):
2     for epoch in range(epochs):
3         start = time.time()
4
5         for image_batch in dataset:
6             train_step(image_batch)
7
8             # GIF를 위한 이미지를 바로 생성
9             display.clear_output(wait=True)
10            generate_and_save_images(generator, epoch + 1, seed)
11
12            # 15 에포크가 지날 때마다 모델을 저장
13            if (epoch + 1) % 15 == 0:
14                checkpoint.save(file_prefix = checkpoint_prefix)
15
16            print ('에포크 {} 에서 걸린 시간은 {} 초 입니다'.format(epoch +1, time.time()-start))
17
18            # 마지막 에포크가 끝난 후 생성
19            display.clear_output(wait=True)
20            generate_and_save_images(generator, epochs, seed)
```

이미지 생성 및 저장

- training 이 False로 맞춰진 것을 주목. 이렇게 하면 (배치정규화를 포함하여) 모든 층들이 추론 모드로 실행

In [0]:

```
1 def generate_and_save_images(model, epoch, test_input):
2
3     predictions = model(test_input, training=False)
4
5     fig = plt.figure(figsize=(4,4))
6
7     for i in range(predictions.shape[0]):
8         plt.subplot(4, 4, i+1)
9         plt.imshow(predictions[i, :, :, 0] * 127.5 + 127.5, cmap='gray')
10        plt.axis('off')
11
12    plt.savefig('image_at_epoch_{:04d}.png'.format(epoch))
13    plt.show()
```

## 모델 훈련

- 위에 정의된 train() 메서드를 생성자와 감별자를 동시에 훈련하기 위해 호출
- 생성적 적대 신경망을 학습하는 것은 매우 까다로울 수 있 dma. 생성자와 감별자가 서로를 제압하지 않는 것이 중요 (예를 들어 학습률이 비슷하면 한쪽이 우세해짐.)
- 훈련 초반부에는 생성된 이미지는 랜덤한 노이즈처럼 보이나 훈련이 진행될수록, 생성된 숫자는 점차 진짜처럼 보임
- 약 50 에포크가 지난 후, MNIST 숫자와 닮은 이미지가 생성
- 코랩에서 기본 설정으로 실행하면, 에포크마다 1분정도 소요

In [0]:

```
1 %%time
2 train(train_dataset, EPOCHS)
```

마지막 체크포인트를 복구합니다.

In [0]:

```
1 checkpoint.restore(tf.train.latest_checkpoint(checkpoint_dir))
```

Type *Markdown* and *LaTeX*:  $\alpha^2$

## GIF 생성

In [0]:

```
1 # 에포크 숫자를 사용하여 하나의 이미지를 보여준다
2 def display_image(epoch_no):
3     return PIL.Image.open('image_at_epoch_{:04d}.png'.format(epoch_no))
4
5 display_image(EPOCHS)
```

In [0]:

1	
---	--