Introduction to Deep learning and Simple applications

- This presentation is heavily based on:
- http://cs.nyu.edu/~fergus/pmwiki/pmwiki.php
- http://deeplearning.net/reading-list/tutorials/
- http://deeplearning.net/tutorial/lenet.html
- http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial
- _CS224 NLP with DL course at Stanford
- courses.cs.tau.ac.il/.../Lecture%201%20CNN%20introduction.ppt..
- Introduction to Deep Learning by Ismini Lourentzou
- www.iis.sinica.edu.tw/Workshop/dl2015/LEE.pptx
- #모두를위한딥러닝시즌2 #deeplearningzerotoall #PyTorch
- Github: https://github.com/deeplearningzeroto...
- YouTube: http://bit.ly/2UVk3kn - Slide: http://bit.ly/2VrZcWM

- ... and many other

HCI Lab
Inha University
Human-Computer Interaction Lab

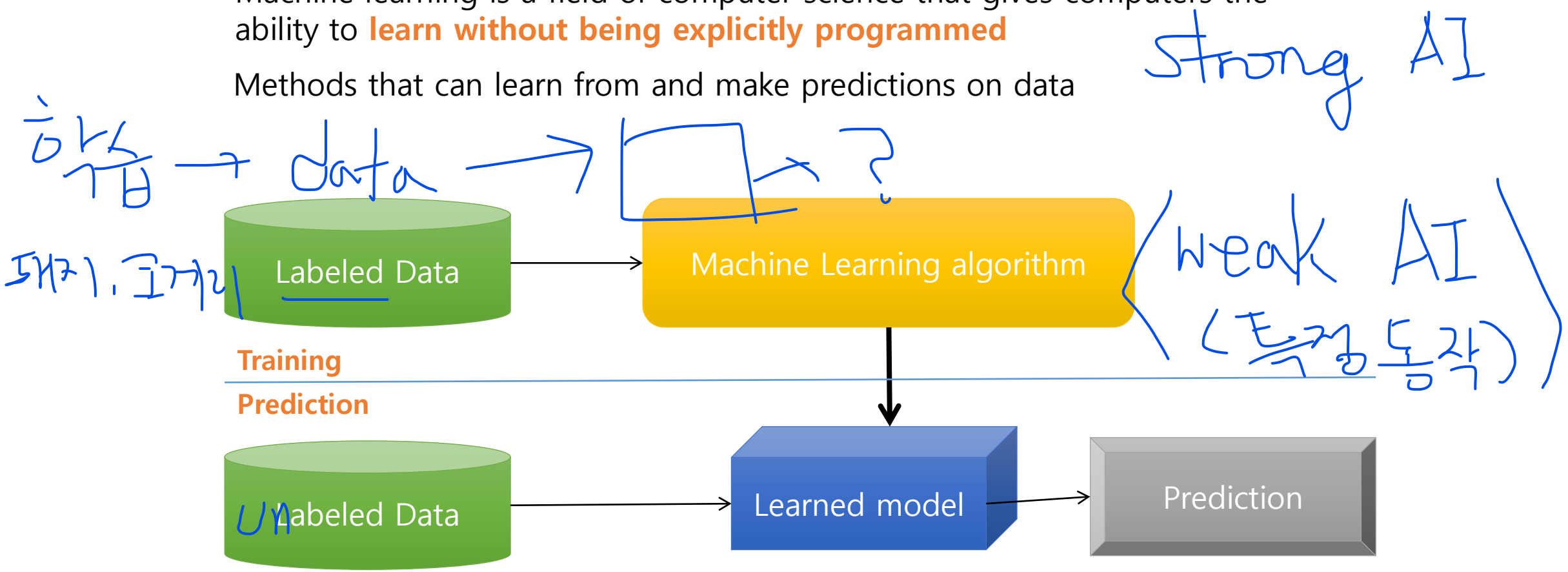
Contents

1. Machine Learning basics
2. What is Deep Learning
3. Neural network introduction
4. Setup Environment
5. Introduction to Convolutional

1. Machine Learning Basics

Machine learning is a field of computer science that gives computers the ability to **learn without being explicitly programmed**

Methods that can learn from and make predictions on data

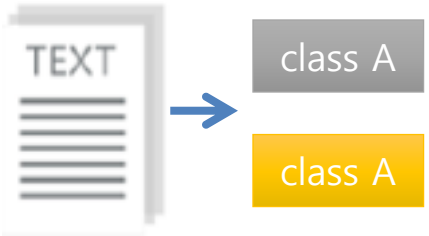


Types of Learning

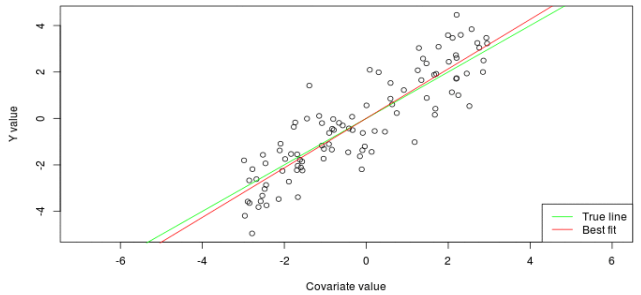
Supervised: Learning with a **labeled training** set
Example: email *classification* with already labeled emails

Unsupervised: Discover **patterns** in **unlabeled** data
Example: *cluster* similar documents based on text

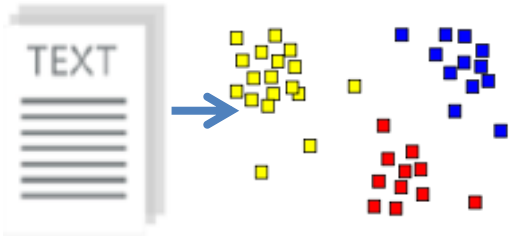
Reinforcement learning: learn to **act** based on **feedback/reward**
Example: learn to play Go, reward: *win or lose*



Classification



Regression



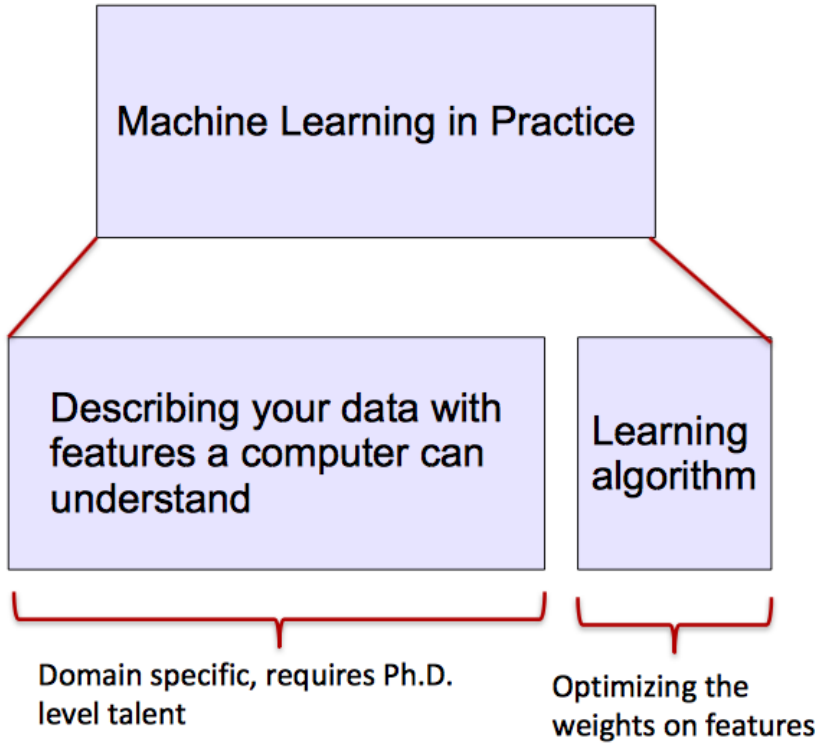
Clustering

Anomaly Detection
Sequence labeling
…

http://mbjoseph.github.io/2013/11/27/measure.html

ML vs. Deep Learning

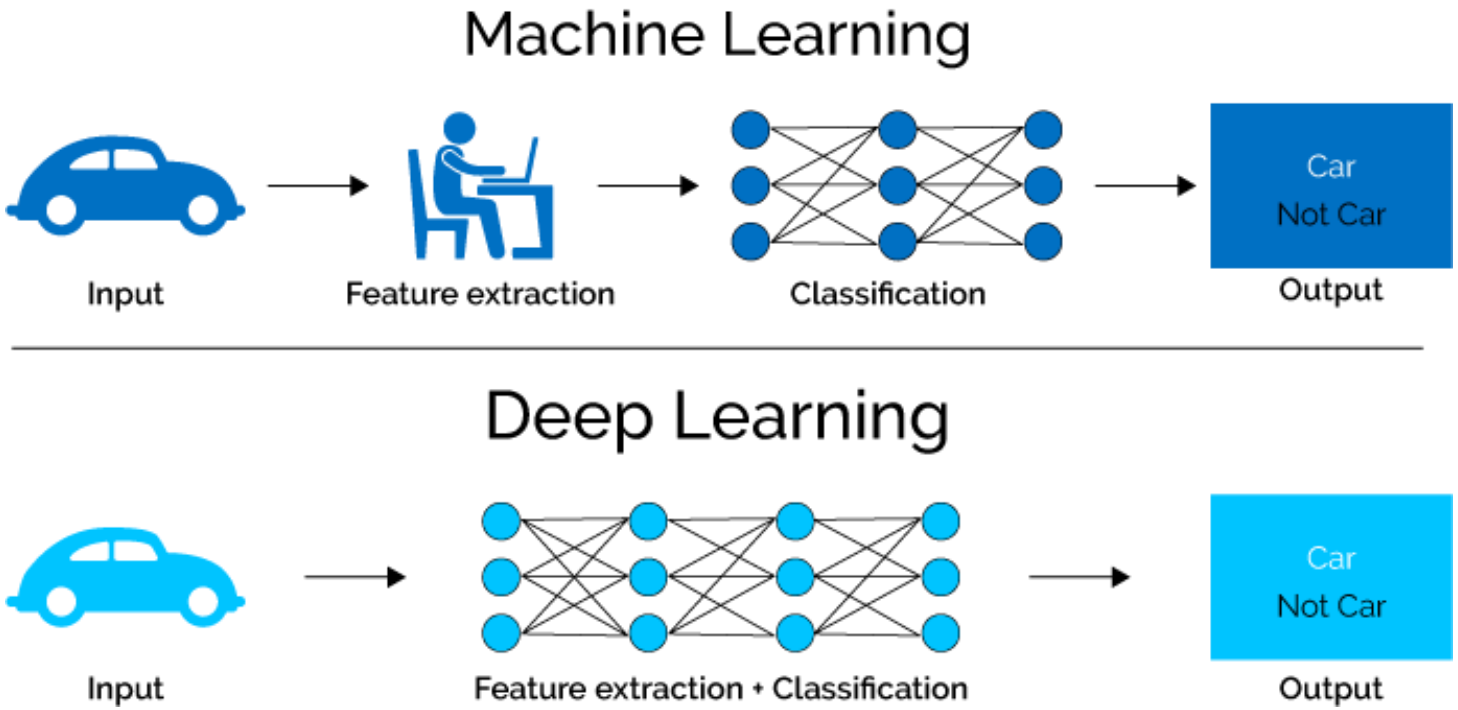
- Most machine learning methods work well because of **human-designed representations** and **input features**
- ML becomes just **optimizing weights** to best make a final prediction



Feature	NER
Current Word	✓
Previous Word	✓
Next Word	✓
Current Word Character n-gram	all
Current POS Tag	✓
Surrounding POS Tag Sequence	✓
Current Word Shape	✓
Surrounding Word Shape Sequence	✓
Presence of Word in Left Window	size 4
Presence of Word in Right Window	size 4

2. What is Deep Learning (DL) ?

- A machine learning subfield of learning **representations** of data.
- Exceptional effective at **learning patterns**.
- Deep learning algorithms attempt to learn (multiple levels of) representation by using a **hierarchy of multiple layers**
- If you provide the system **tons of information**, it begins to understand it and respond in useful wa ys.

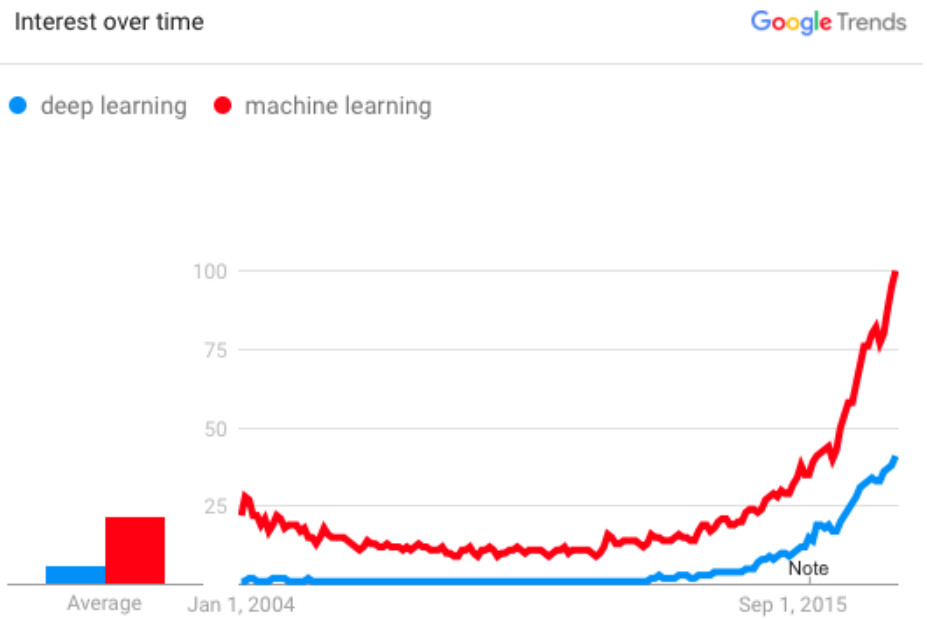


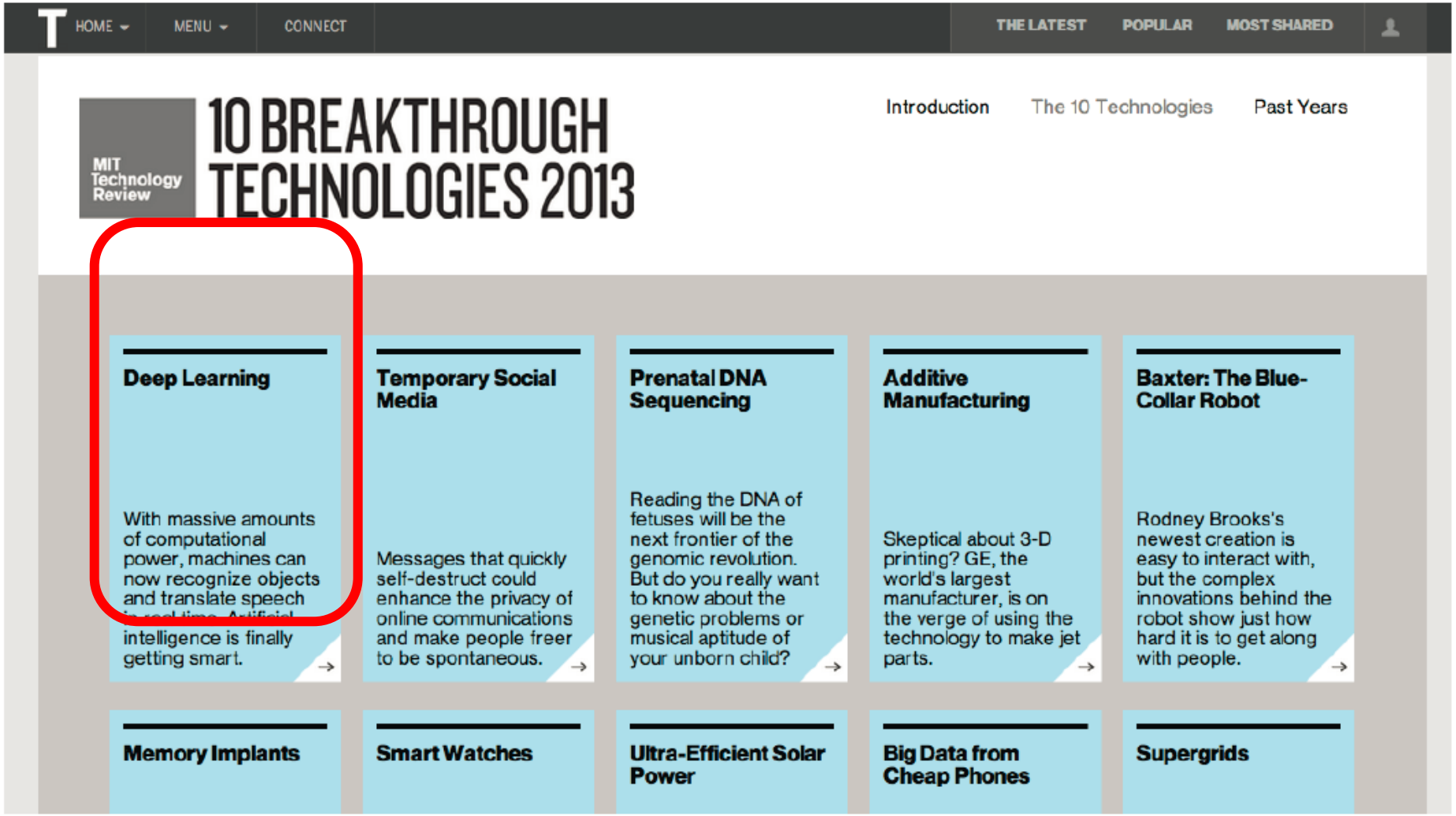
https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Why is DL useful?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing w orld, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

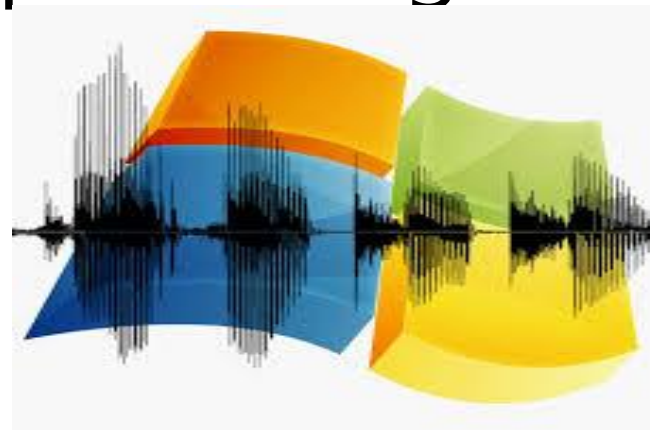
In ~2010 DL started outperforming o ther ML techniques
first in speech and vision, then NLP

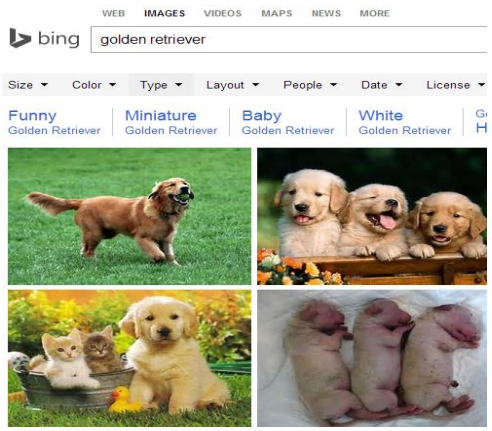


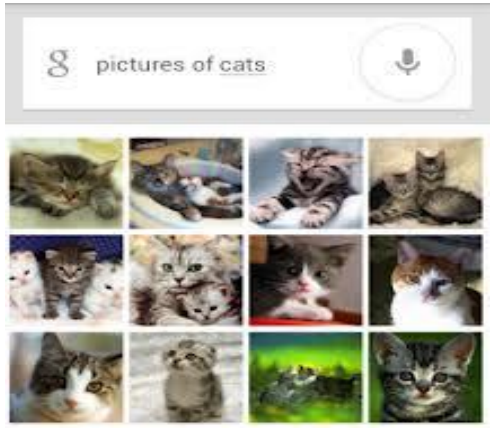


MIT Technology Review, April 23rd, 2013

Deep Learning – from Research to Technology



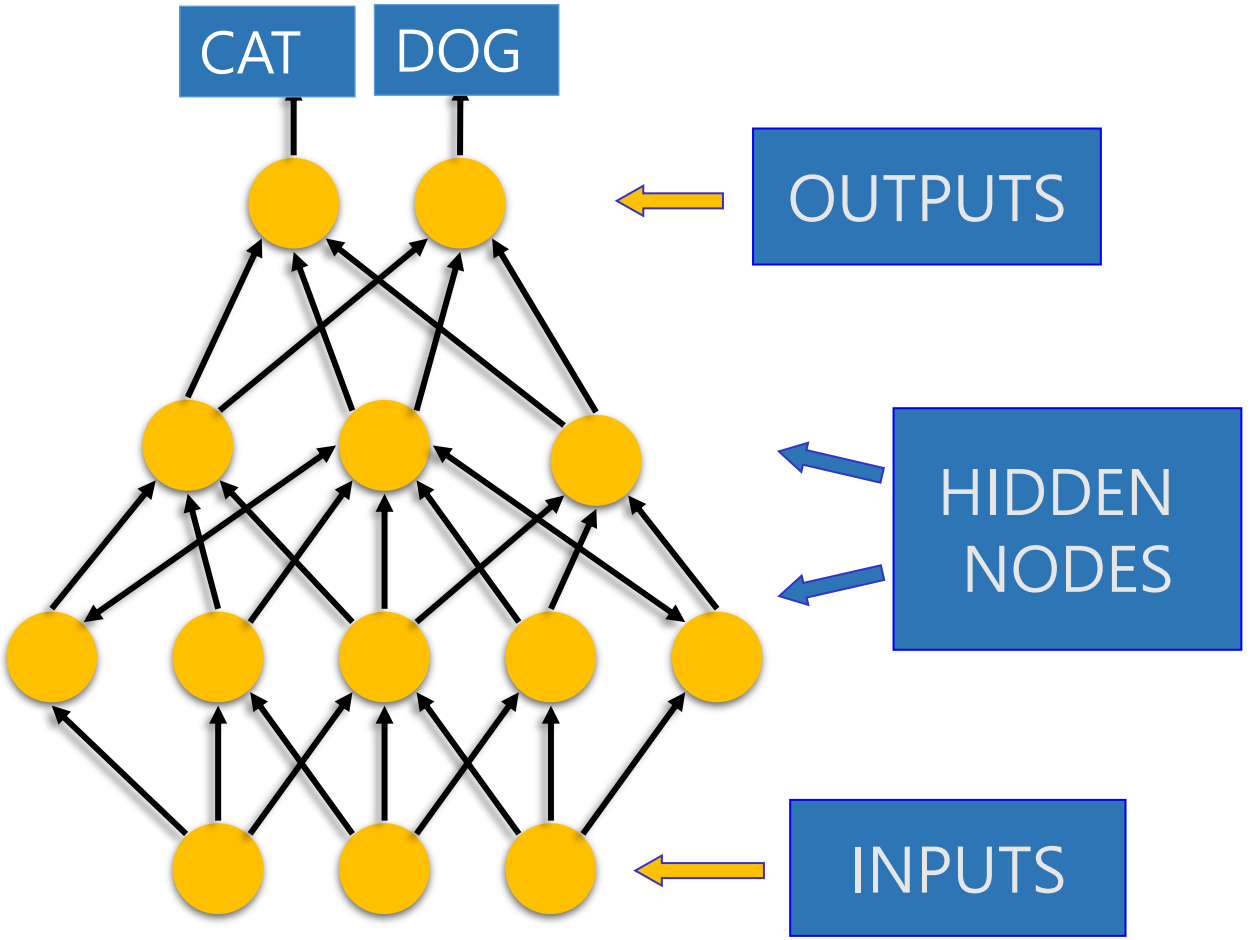




Deep Learning - breakthrough in visual a nd speech recognition

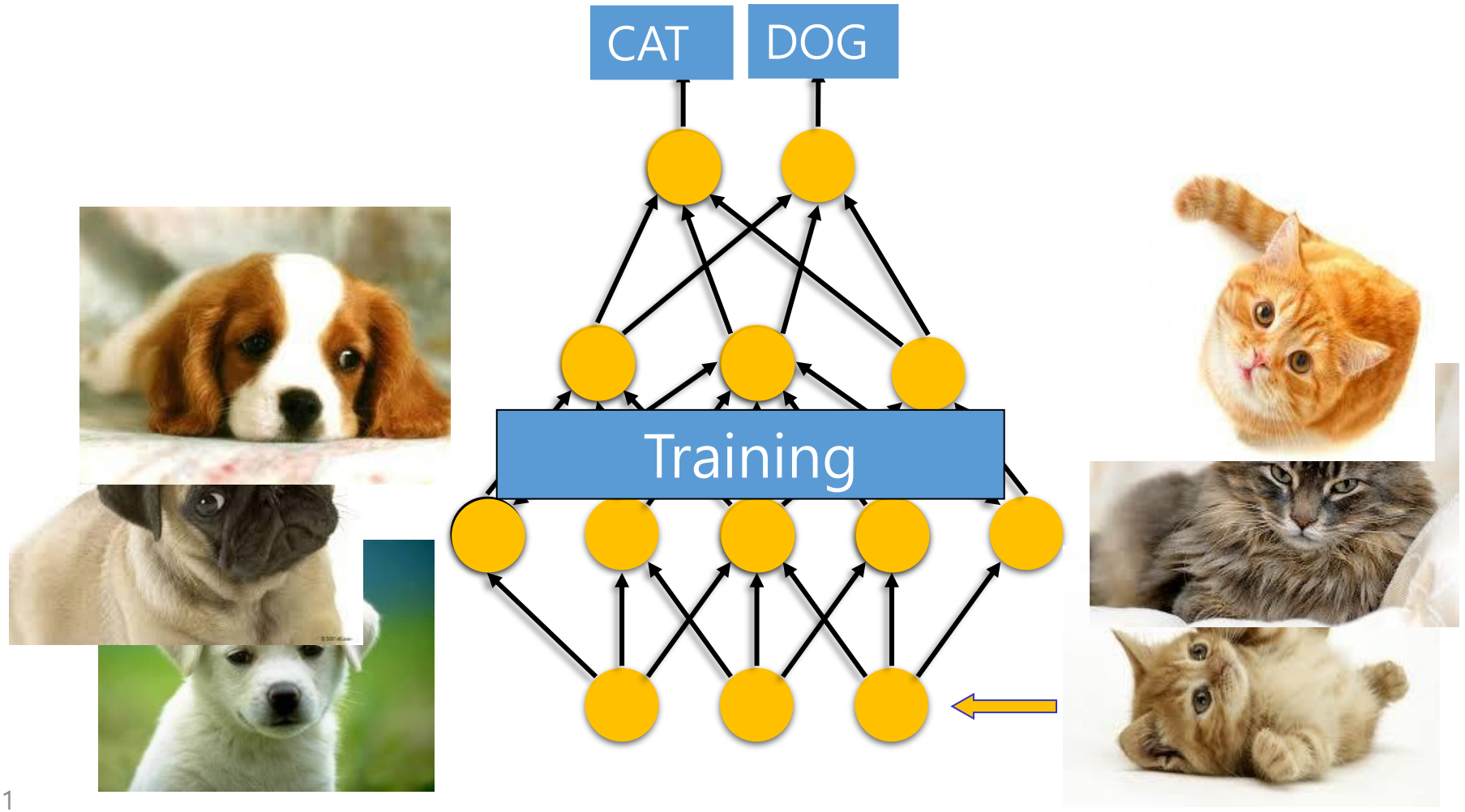
Deep Learning Basics

Deep Learning – is a set of machine learning algorithms based on multi-layer networks



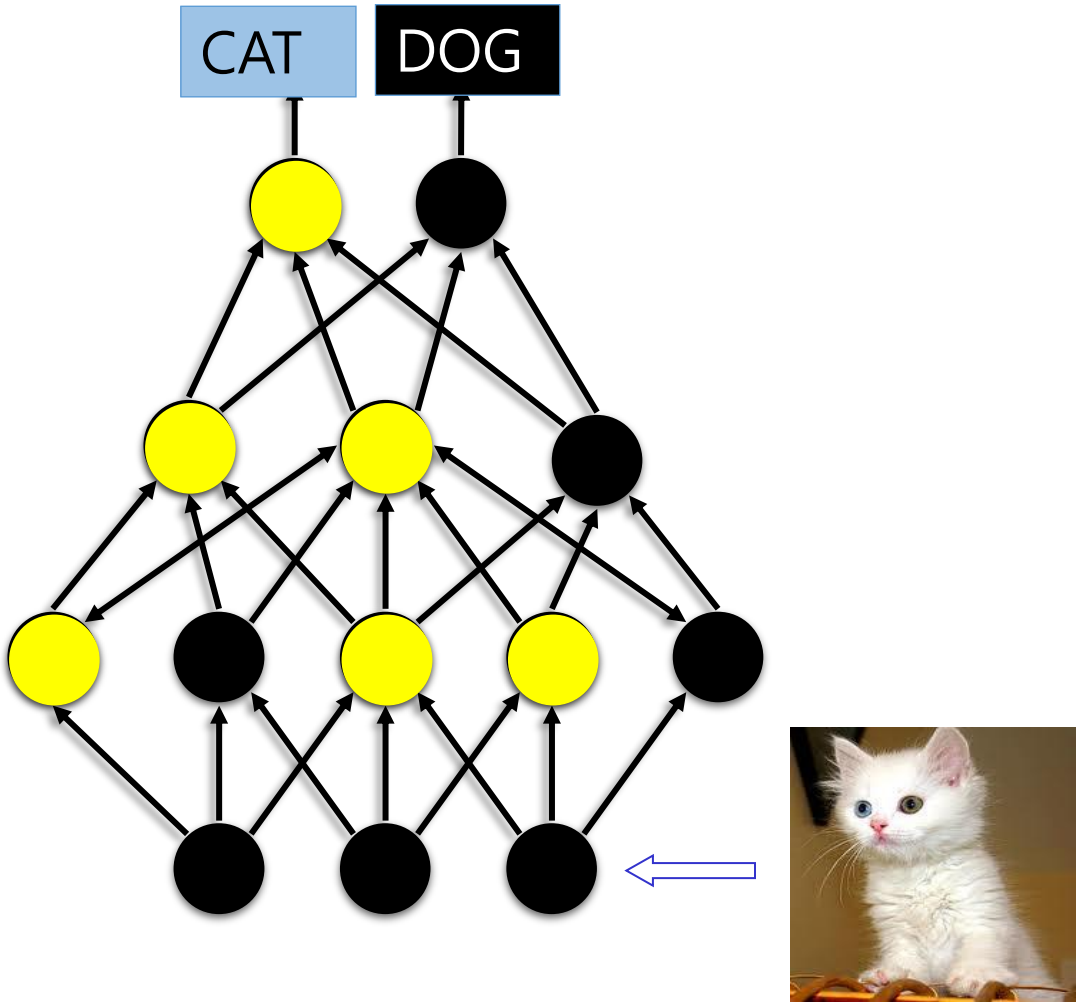
Deep Learning Basics

Deep Learning – is a set of machine learning algorithms based on multi-layer networks



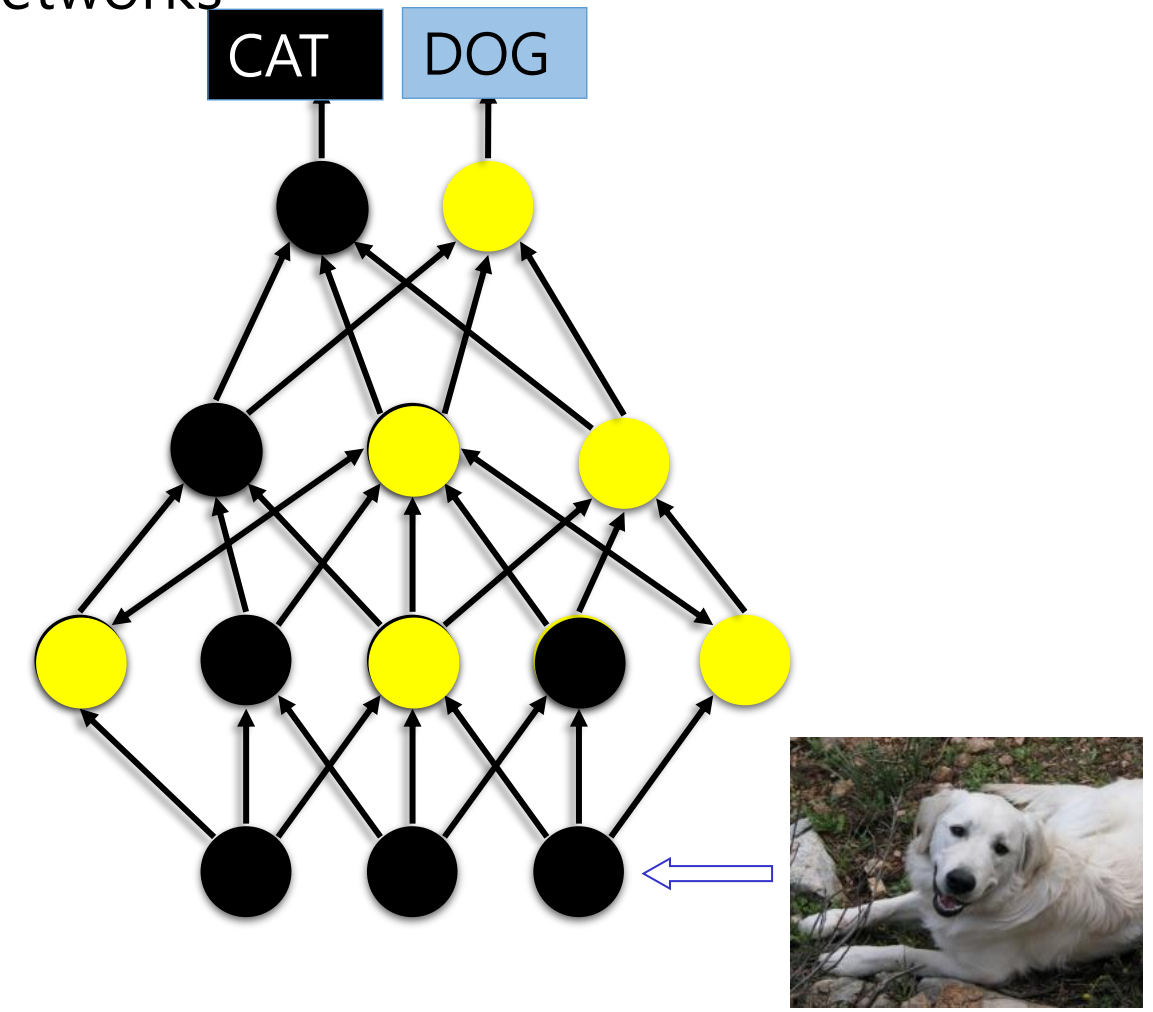
Deep Learning Basics

Deep Learning – is a set of machine learning algorithms based on multi-layer networks



Deep Learning Basics

Deep Learning – is a set of machine learning algorithms based on multi-layer networks



Deep Learning Taxonomy

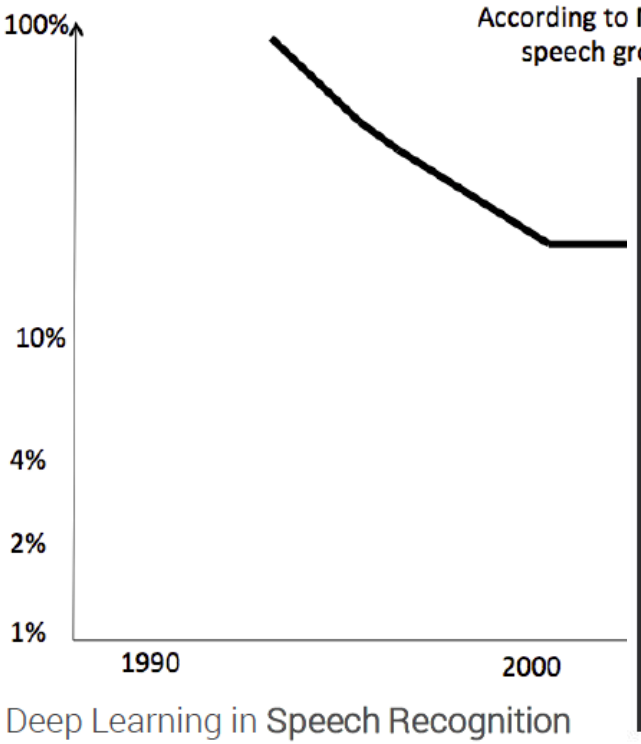
Supervised:

- Convolutional NN (LeCun)
- Recurrent Neural nets (Schmidhuber)

Unsupervised

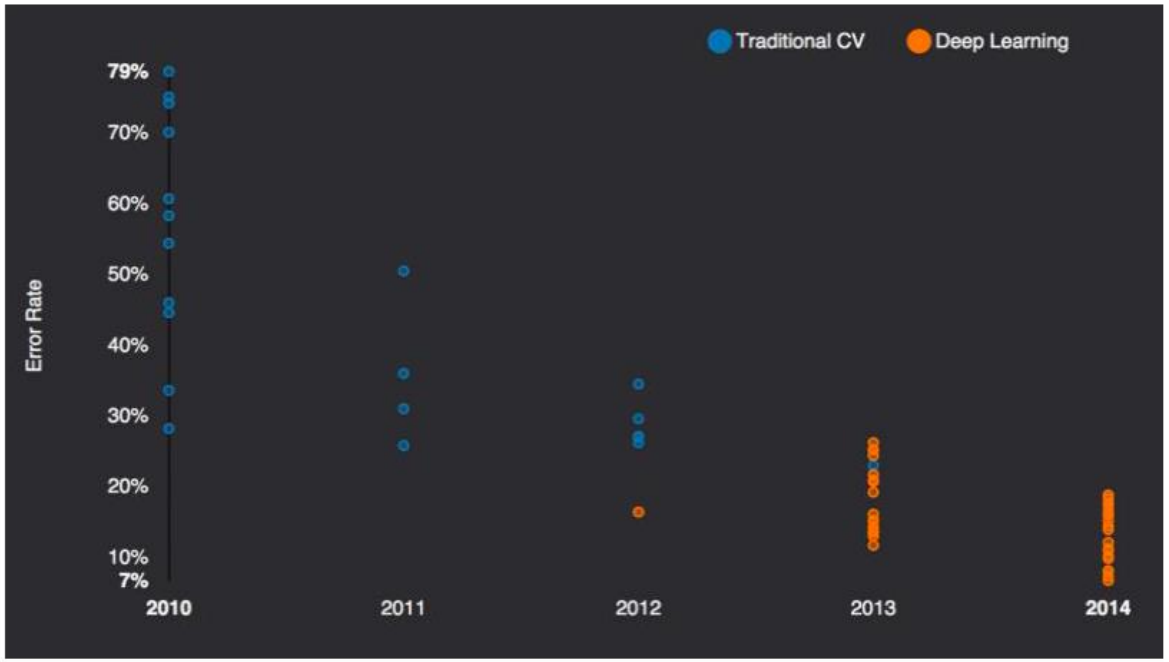
- Deep Belief Nets / Stacked RBMs (Hinton)
- Stacked denoising autoencoders (Bengio)
- Sparse AutoEncoders (LeCun, A. Ng,)

State of the art in …



According to Microsoft's speech group:

Deep Learning in Speech Recognition



ImageNet: The "computer vision World Cup"

Several big improvements in recent years in NLP

- ✓ Machine Translation
- ✓ Sentiment Analysis
- ✓ Dialogue Agents
- ✓ Question Answering
- ✓ Text Classification …

Leverage different levels of representation

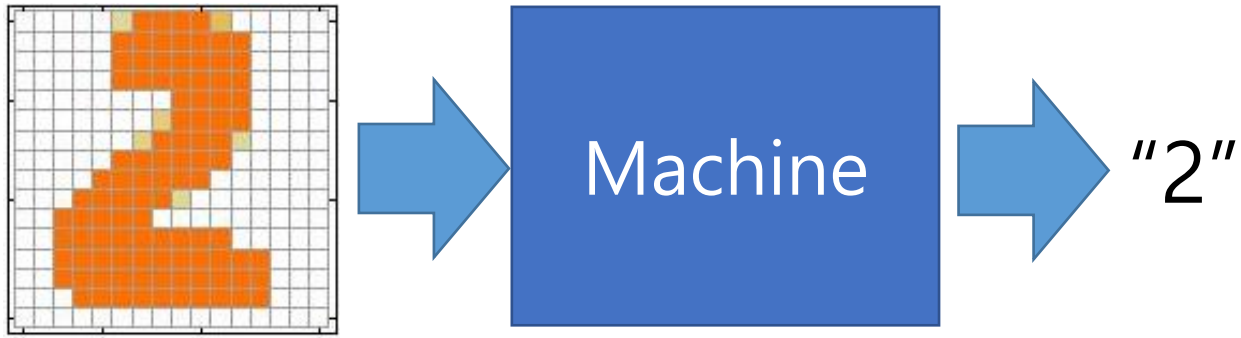
- words & characters
- syntax & semantics

3. Neural Network Introduction

What people already knew in 1980s

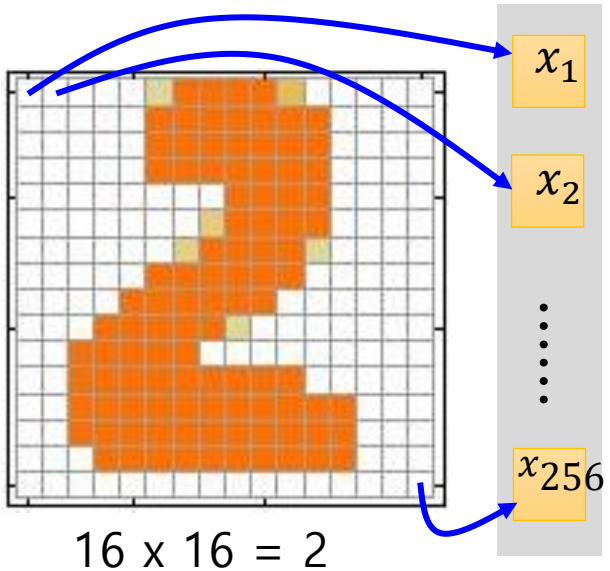
Example Application

- Handwriting Digit Recognition



Handwriting Digit Recognition

Input

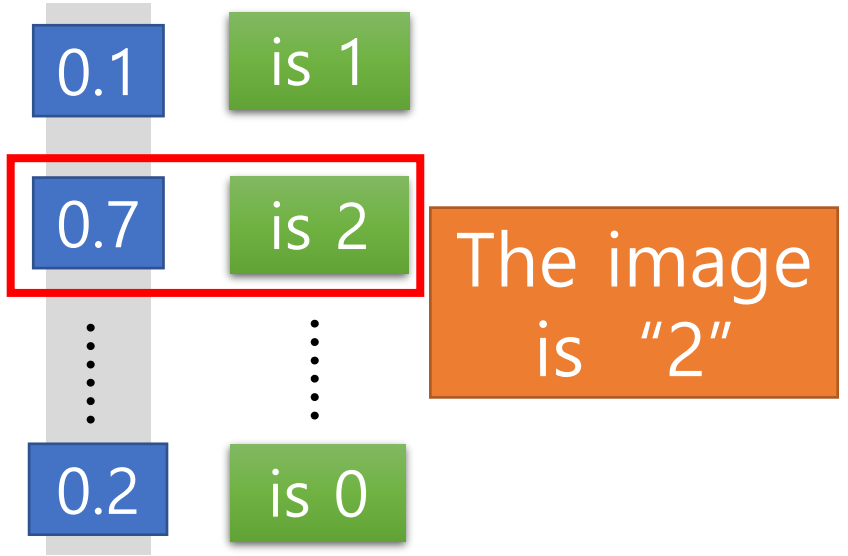


16 x 16 = 256

Ink → 1

No ink → 0

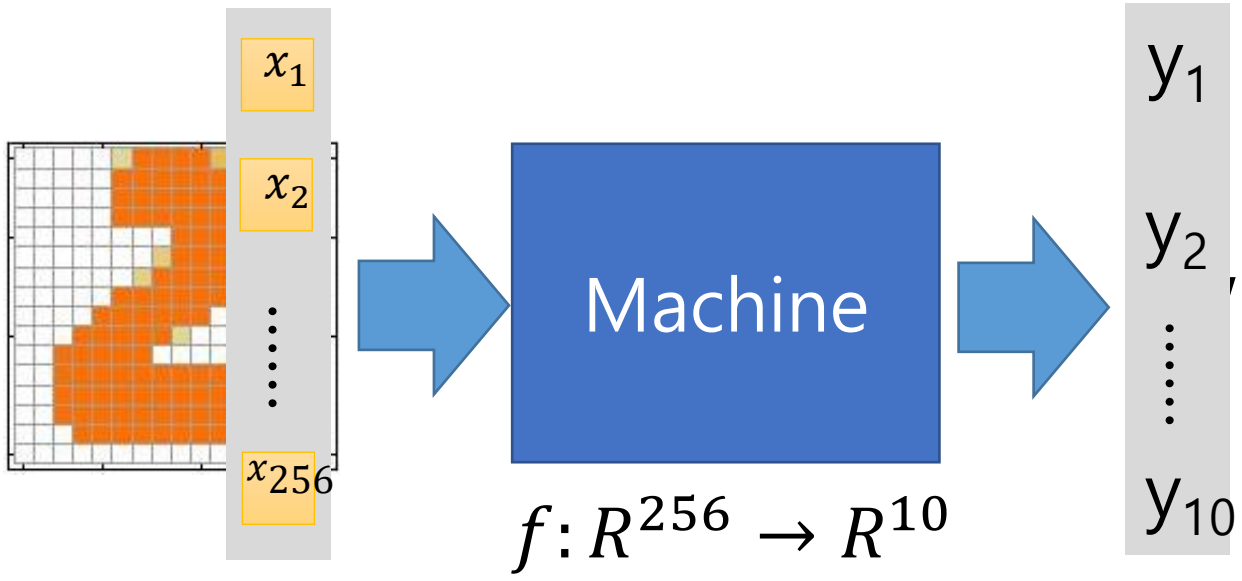
Output



Each dimension represents the confidence of a digit.

Example Application

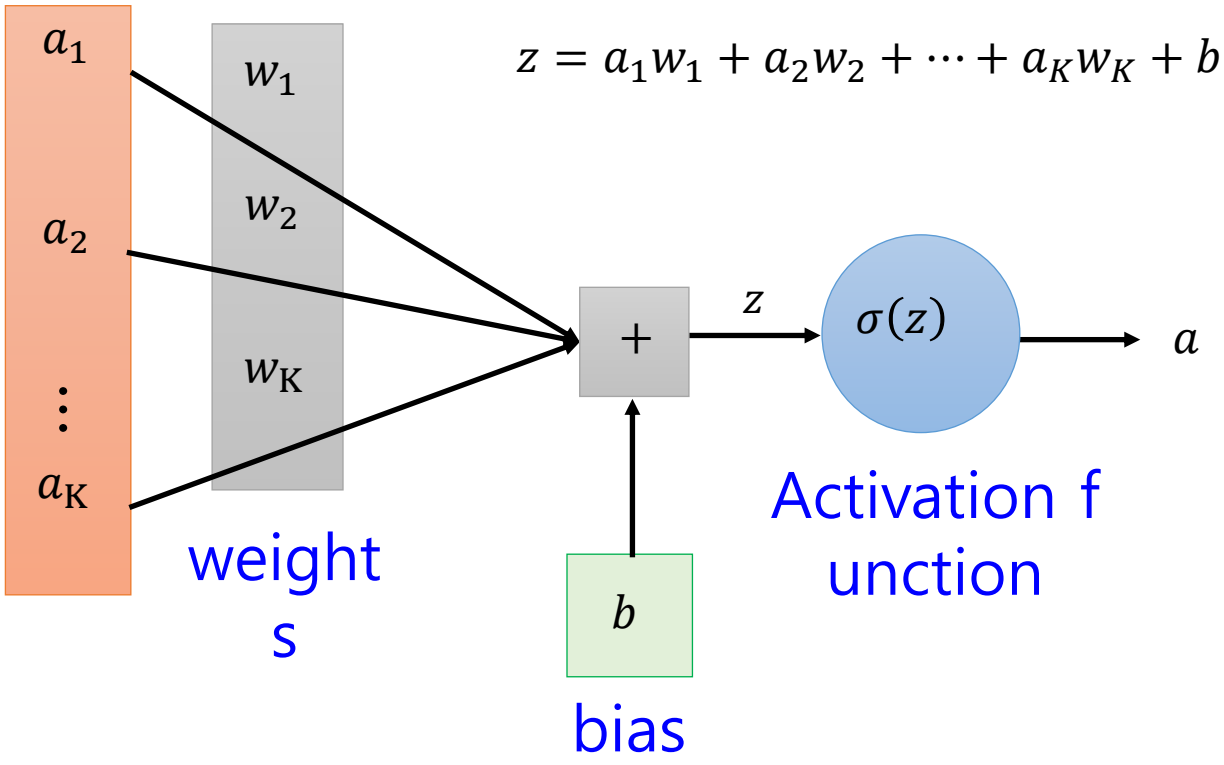
- Handwriting Digit Recognition



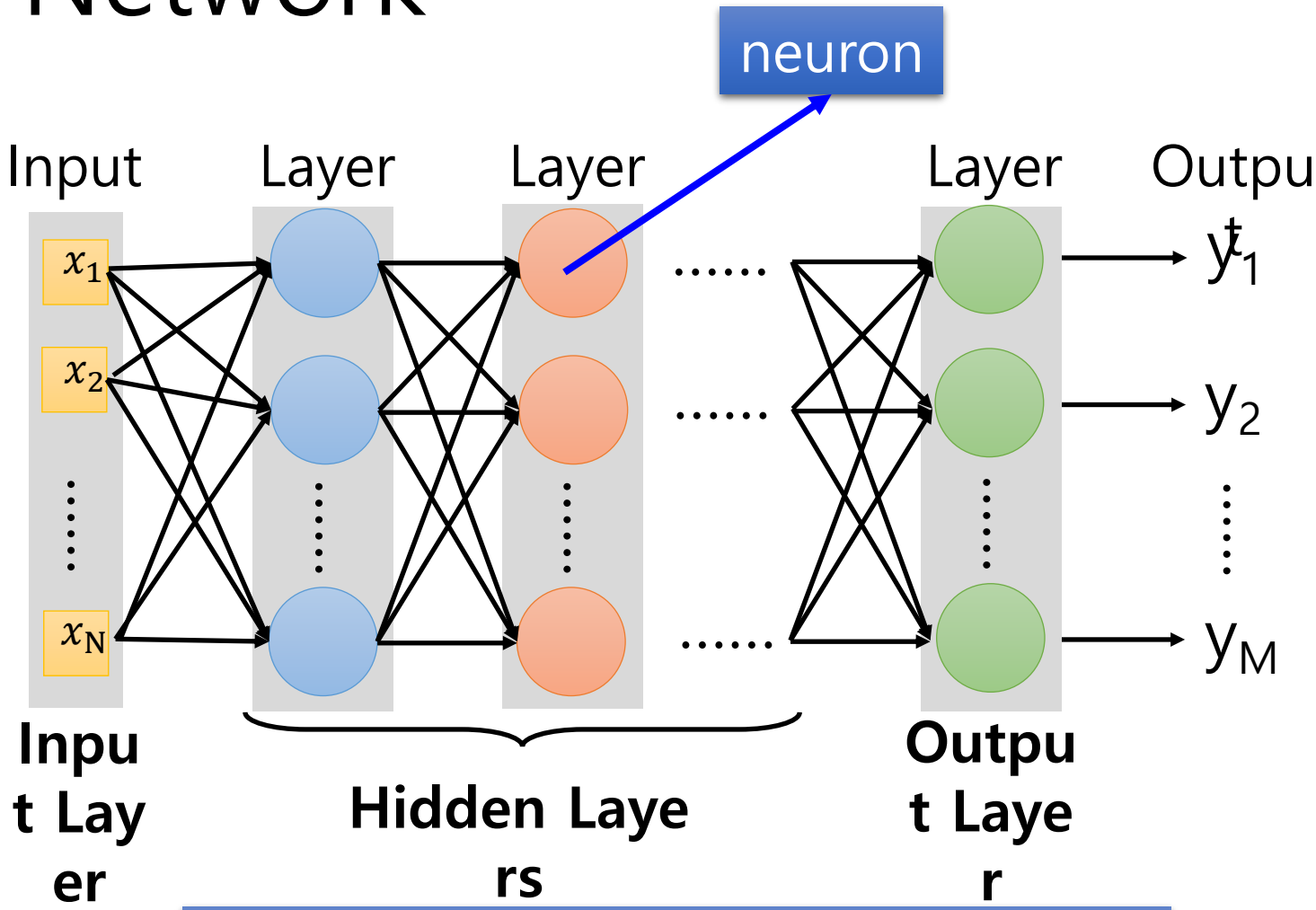
In deep learning, the function f is represented by neural network

Element of Neural Network

Neuron $f: R^K \rightarrow R$

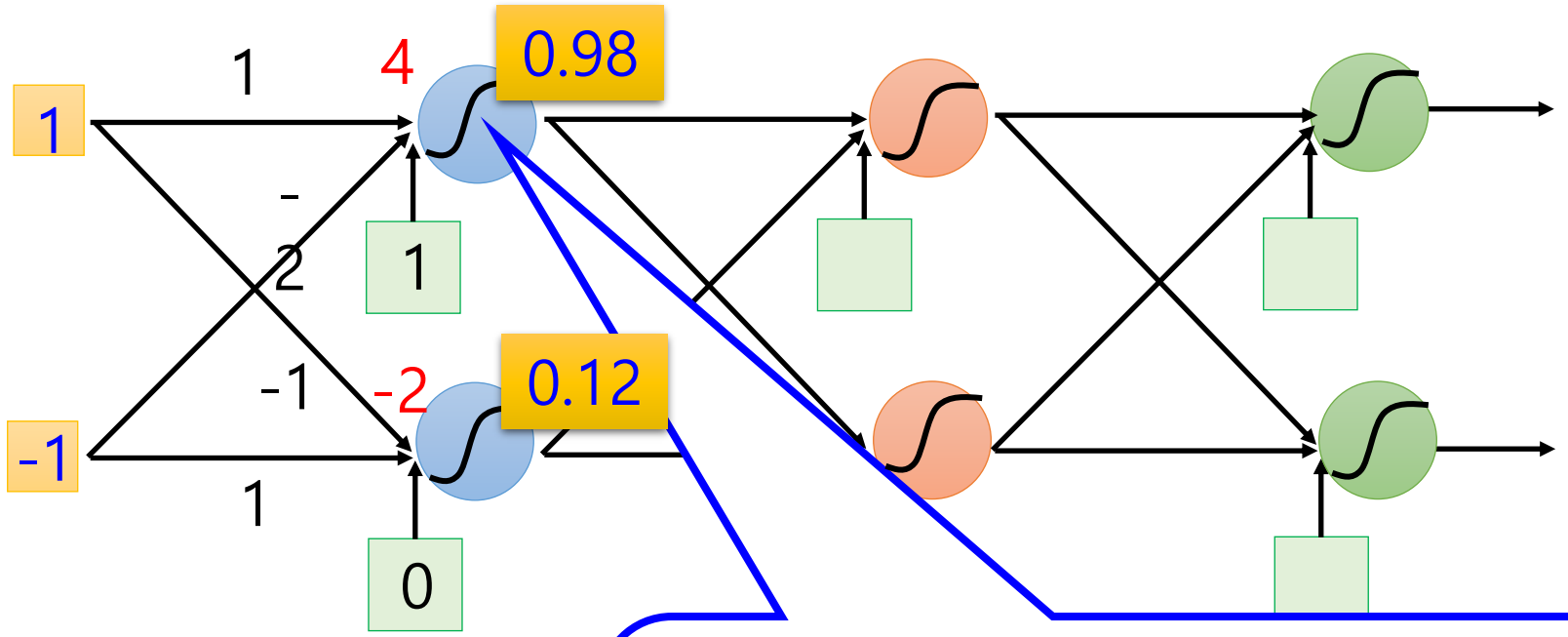


Neural Network



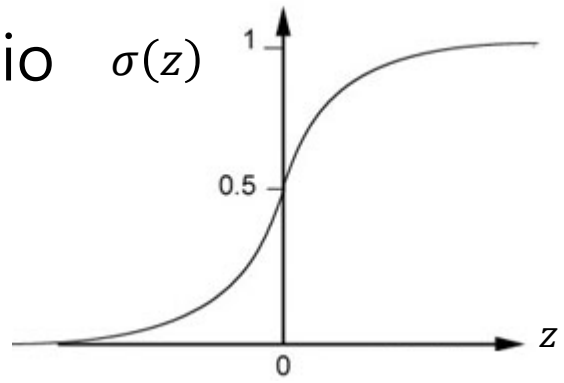
Deep means many hidden layers

Example of Neural Network

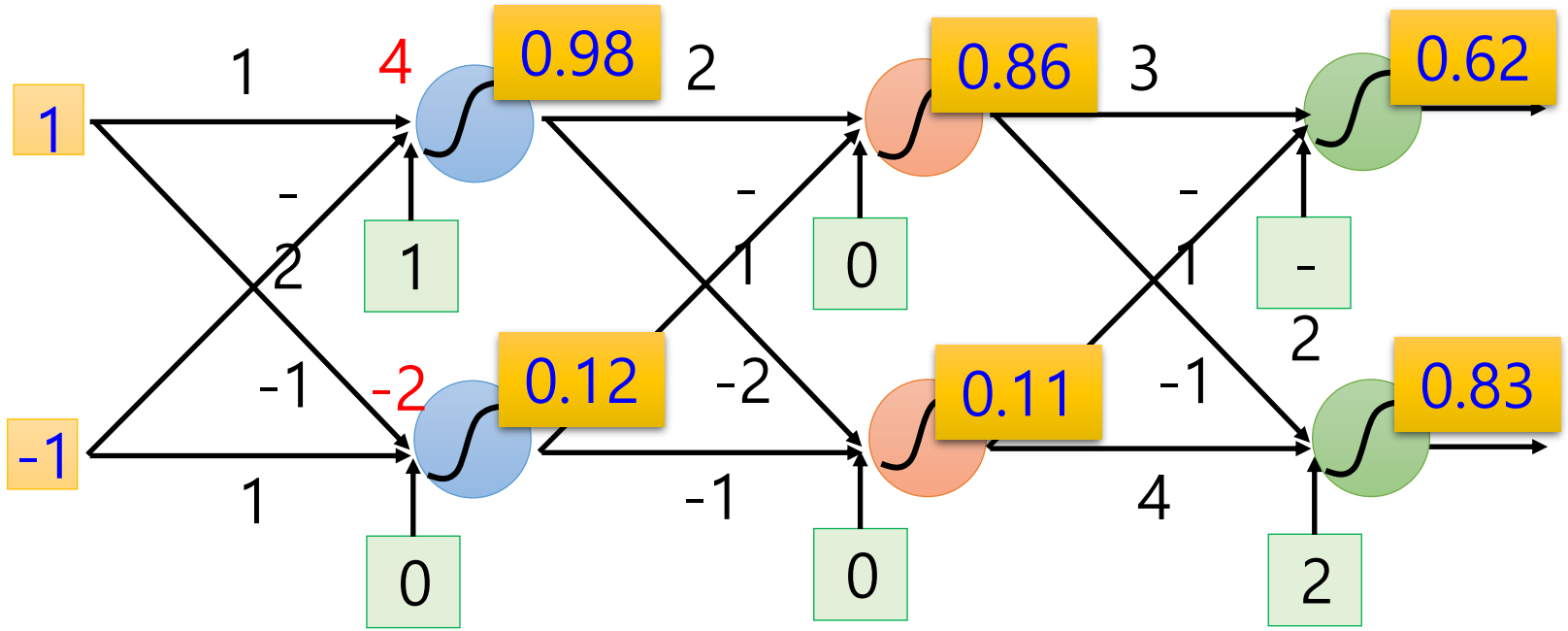


Sigmoid Functio n

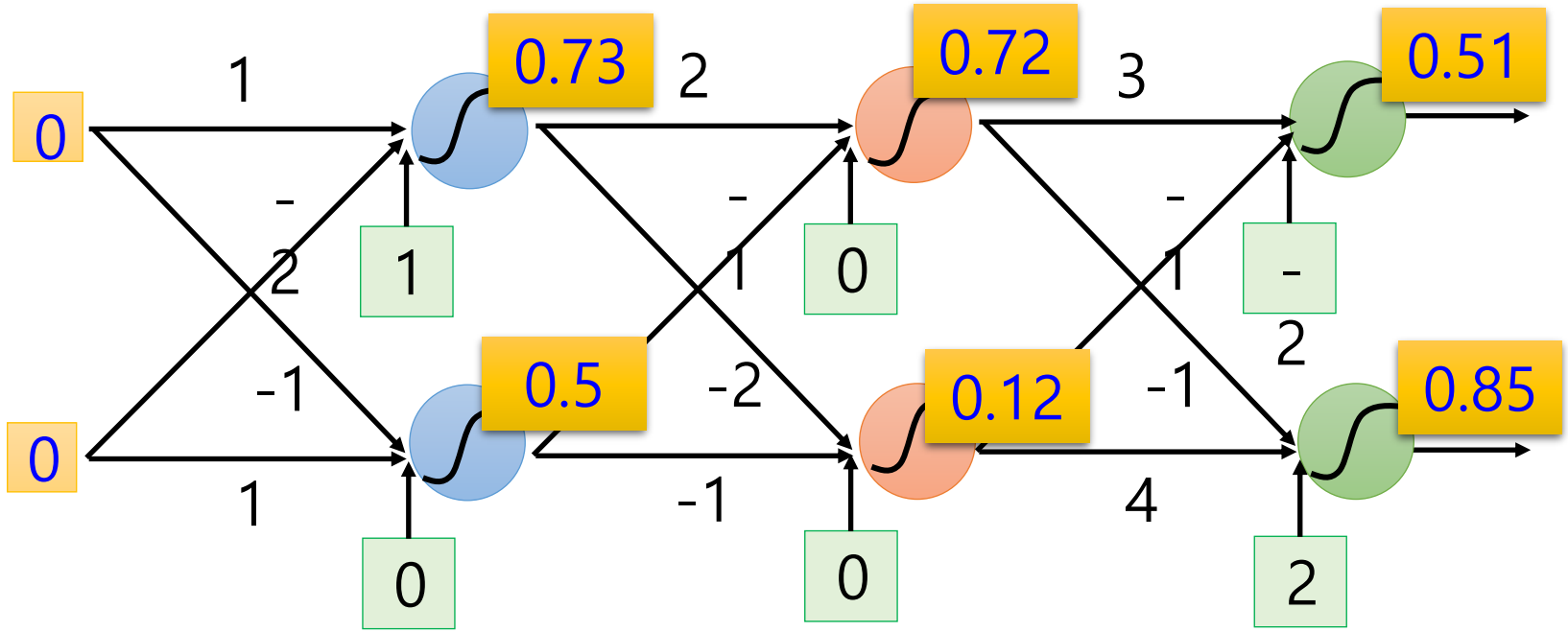
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



Example of Neural Network



Example of Neural Network

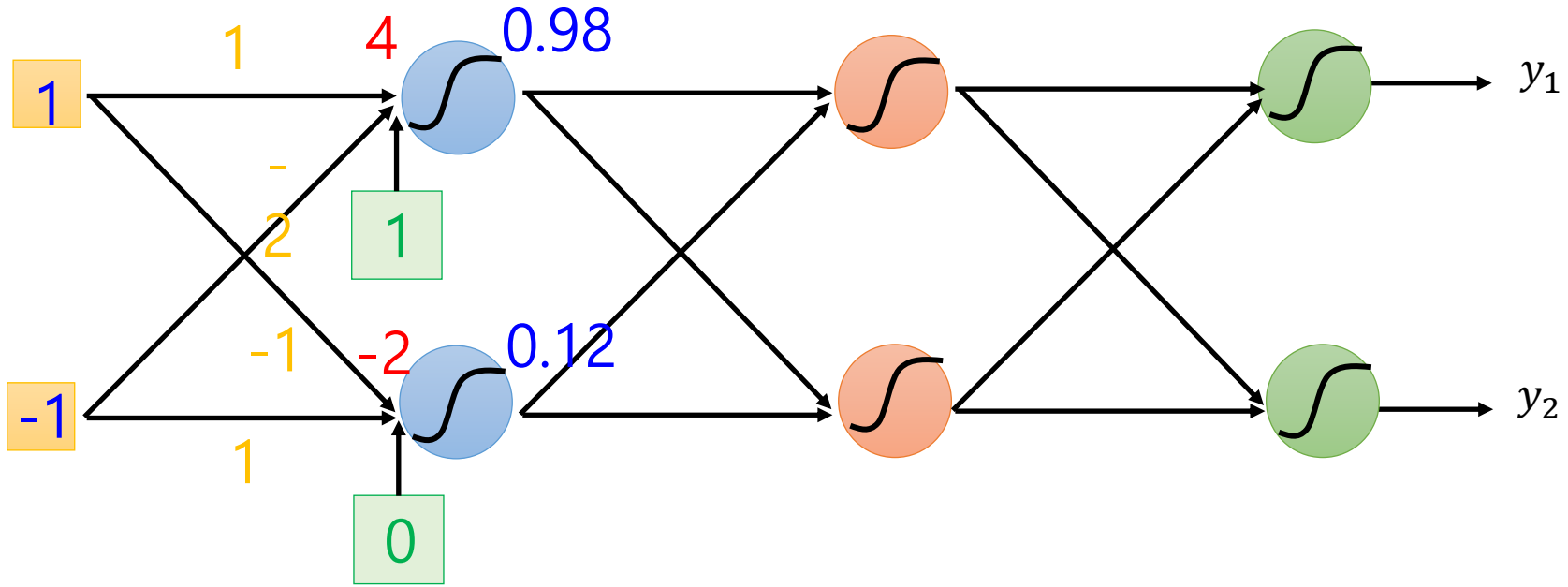


$f: R^2 \rightarrow R^2$

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right) = \begin{bmatrix}0.62\\0.83\end{bmatrix} \quad f\left(\begin{bmatrix}0\\0\end{bmatrix}\right) = \begin{bmatrix}0.51\\0.85\end{bmatrix}$$

Different parameters define different functio n

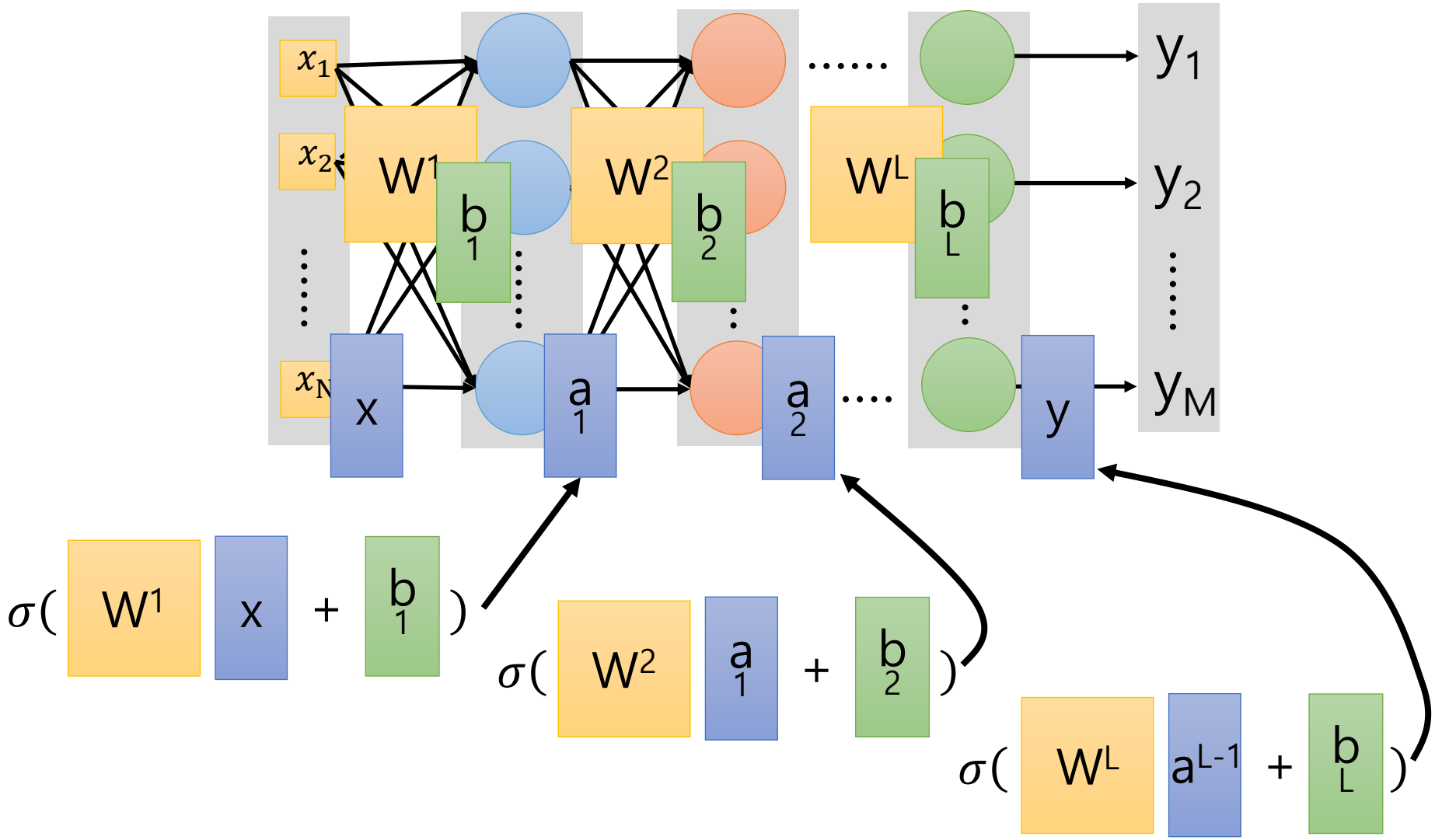
Matrix Operation



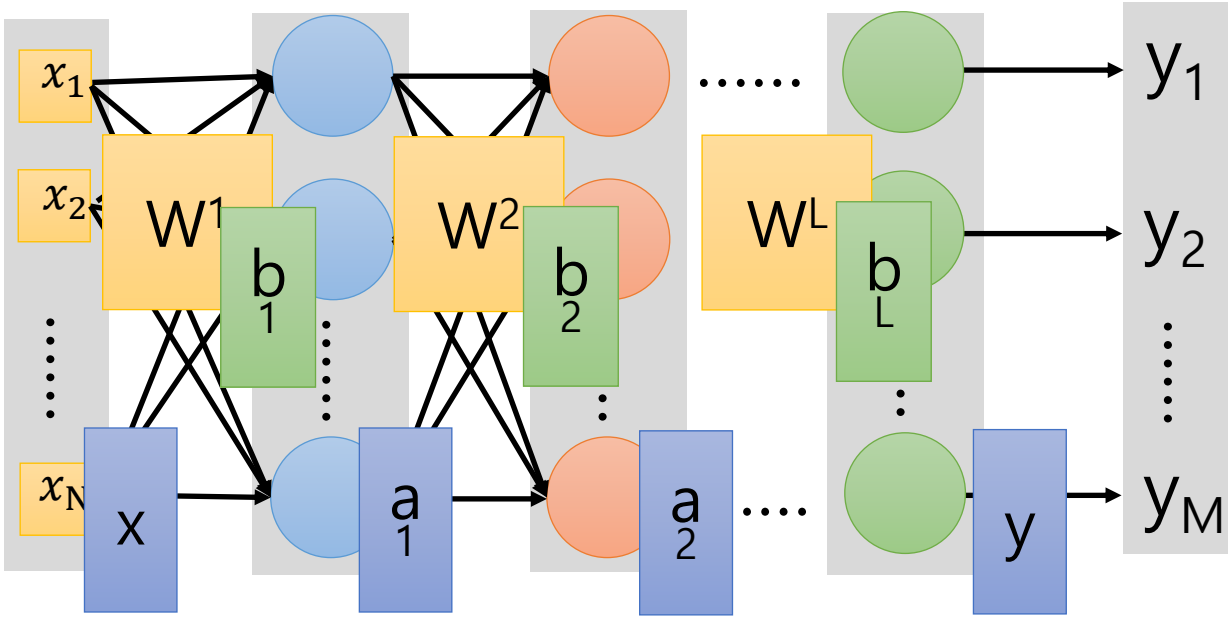
$$\sigma\left(\begin{bmatrix} 1 & -2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 0.98 \\ 0.12 \end{bmatrix}$$

$$\begin{bmatrix} 4 \\ -2 \end{bmatrix}$$

Neural Network



Neural Network



$$y = f(x)$$

Using parallel computing techniques to speed up matrix operation

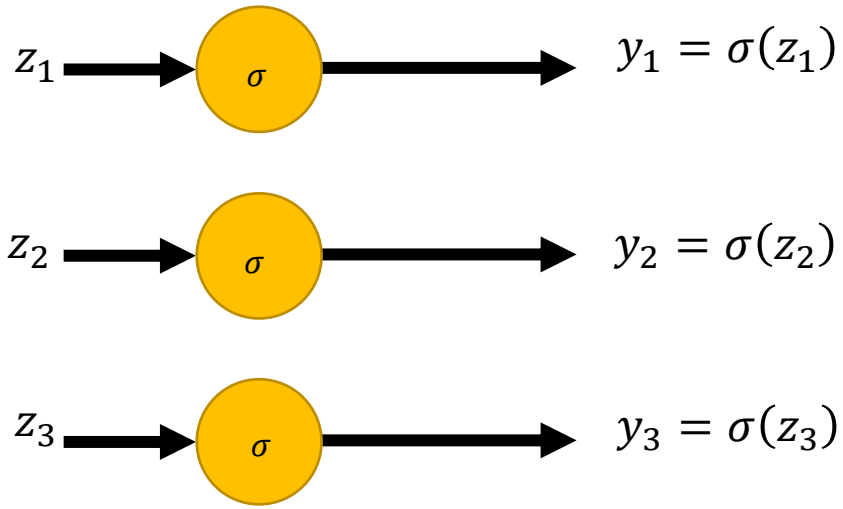
$$= \sigma(W^L \cdots \sigma(W^2 \sigma(W^1 x + b_1) + b_2) \cdots + b_L)$$

Softmax

- Softmax layer as the output layer

0.7
0.2
0.1
⇒ 1.0

Ordinary Layer



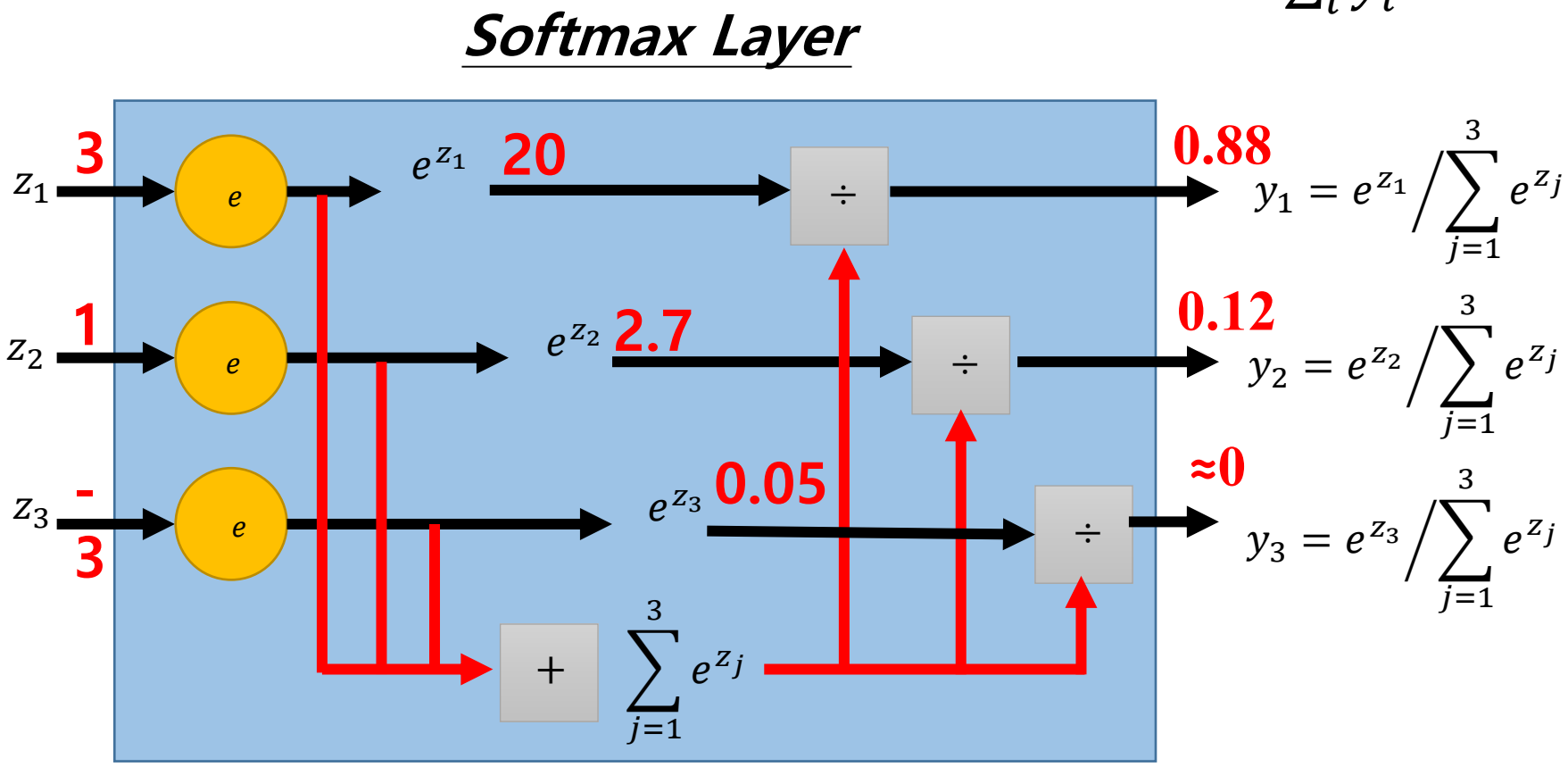
In general, the output of network can be any value.
May not be easy to inter pret

Softmax

- Softmax layer as the output layer

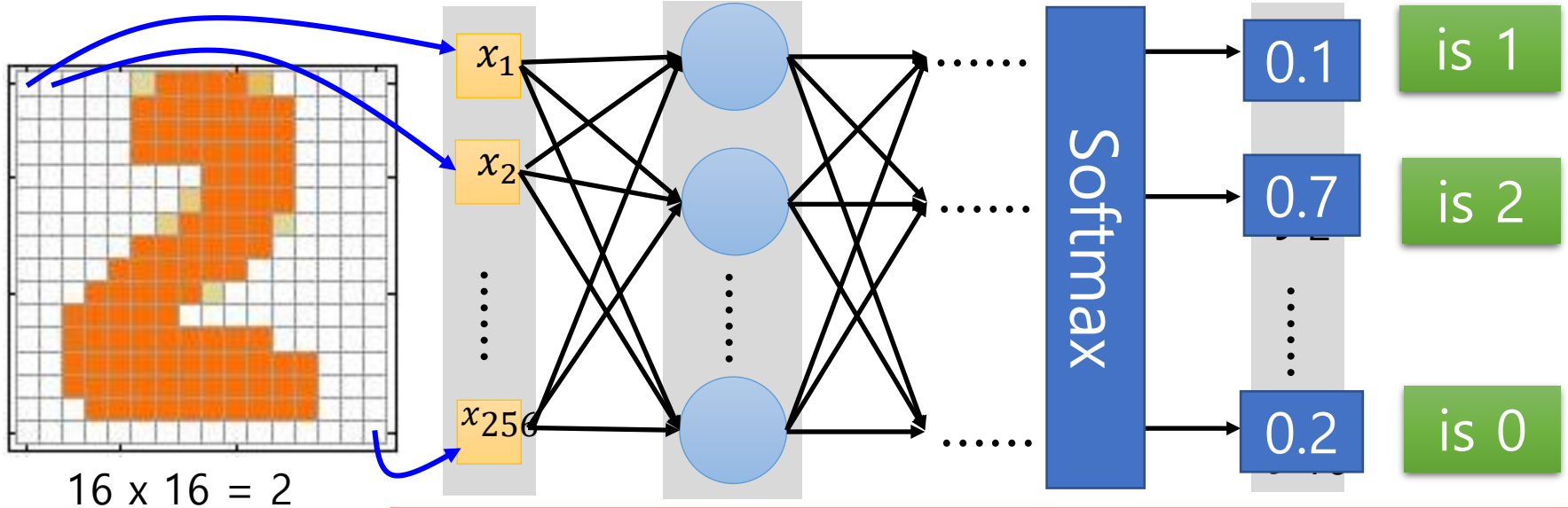
Probability.

- $1 > y_i > 0$
- $\sum_i y_i = 1$



How to set network parameters

$$\theta = \{W^1, b^1, W^2, b^2, \cdots W^L, b^L\}$$



16 x 16 = 256

Ink → 1

No ink → 0

Set the network parameters θ such that ……

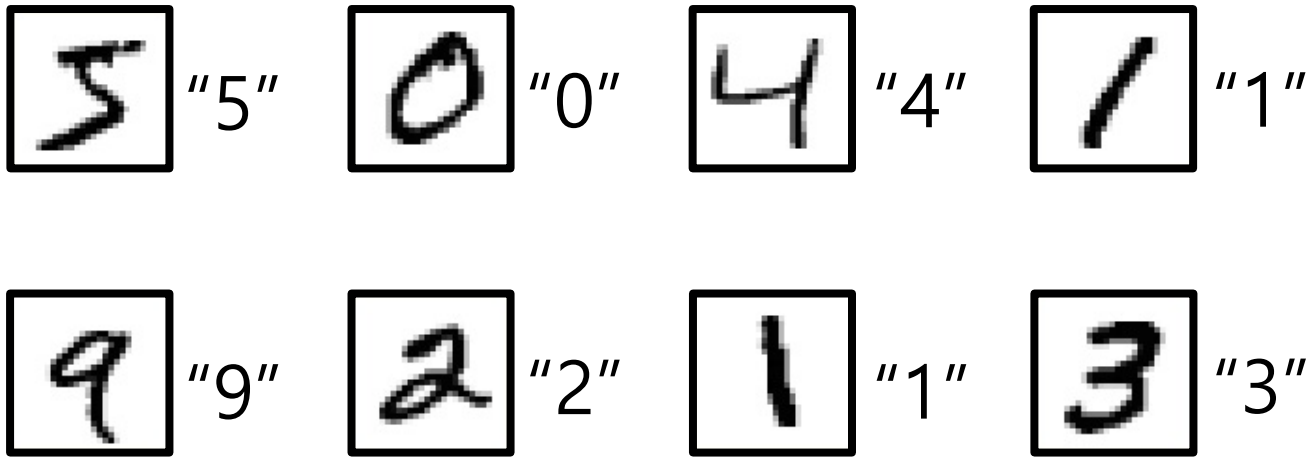
Input: … maximum value

Input: … y_2 has the maximum value

How to let the neural network achieve this

Training Data

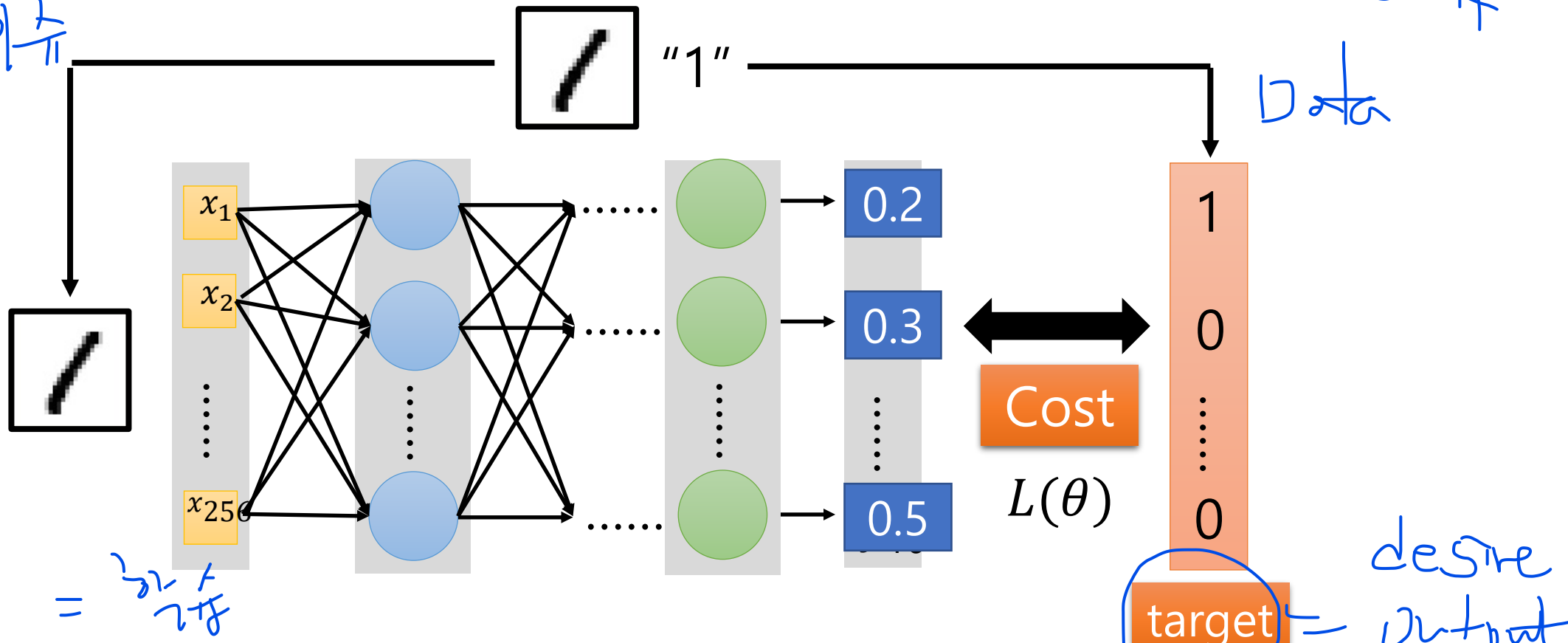
• Preparing training data: images and their labels



Using the training data to find the network parameters.

Cost

Given a set of network parameters θ, each example has a cost value.

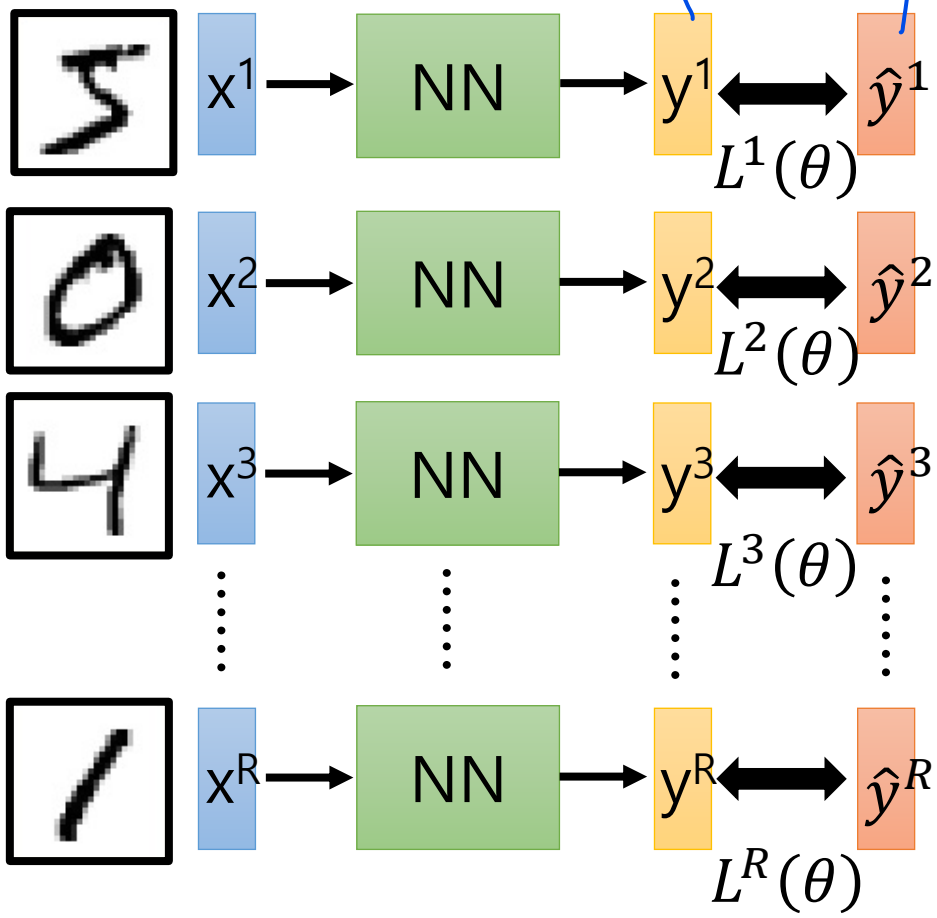


Cost can be Euclidean distance or cross entropy of the network output and target

Total Cost

target = desired = labeled output

For all training data



Total Cost:

$$C(\theta) = \sum_{r=1}^{R} L^r(\theta)$$

How bad the network parameters θ is on this task

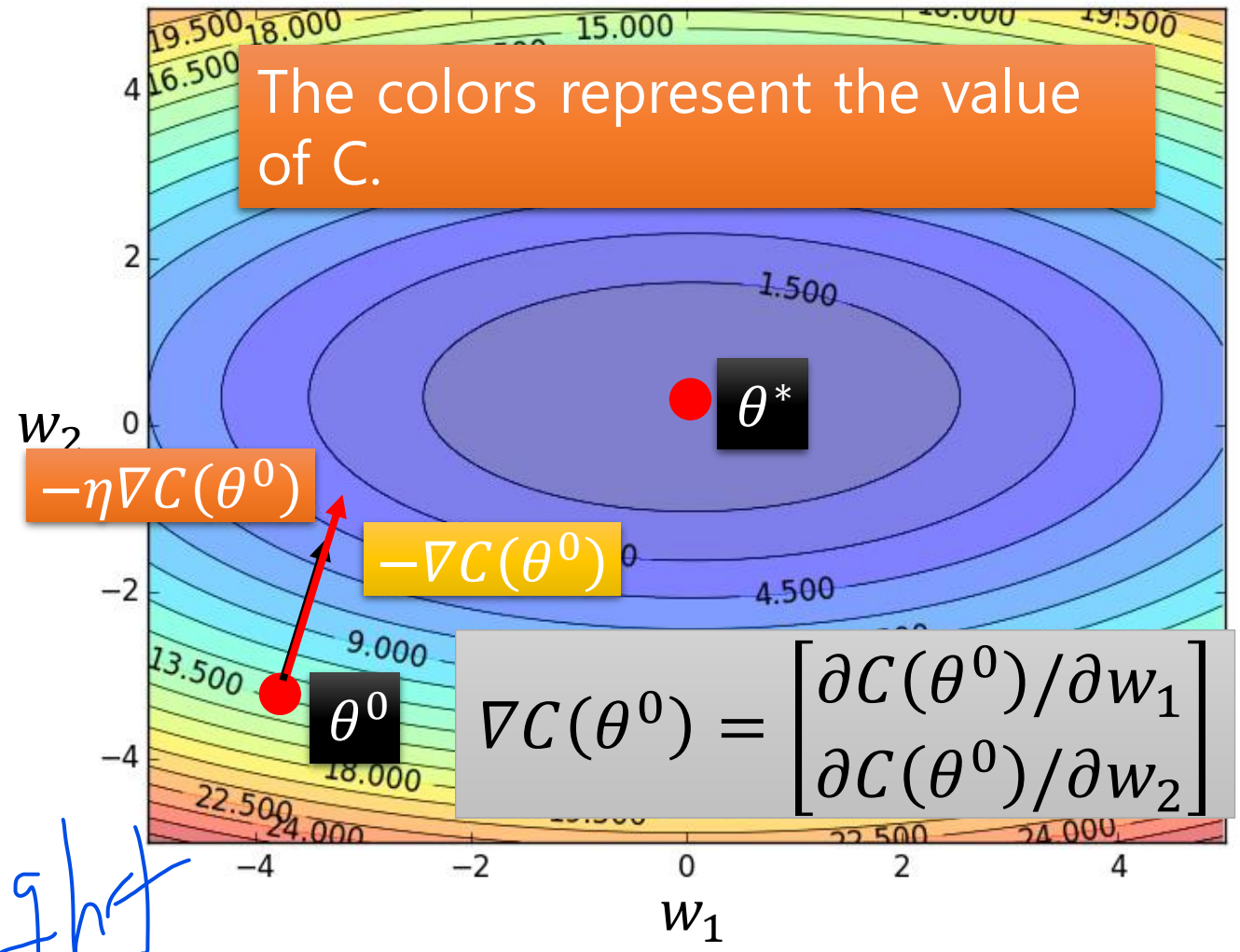
Find the network parameters θ^* that minimize this value

Gradient Descent

Assume there are only two parameters w_1 and w_2 in a network.

$$\theta = \{w_1, w_2\}$$

Error Surface



Randomly pick a starting point θ^0

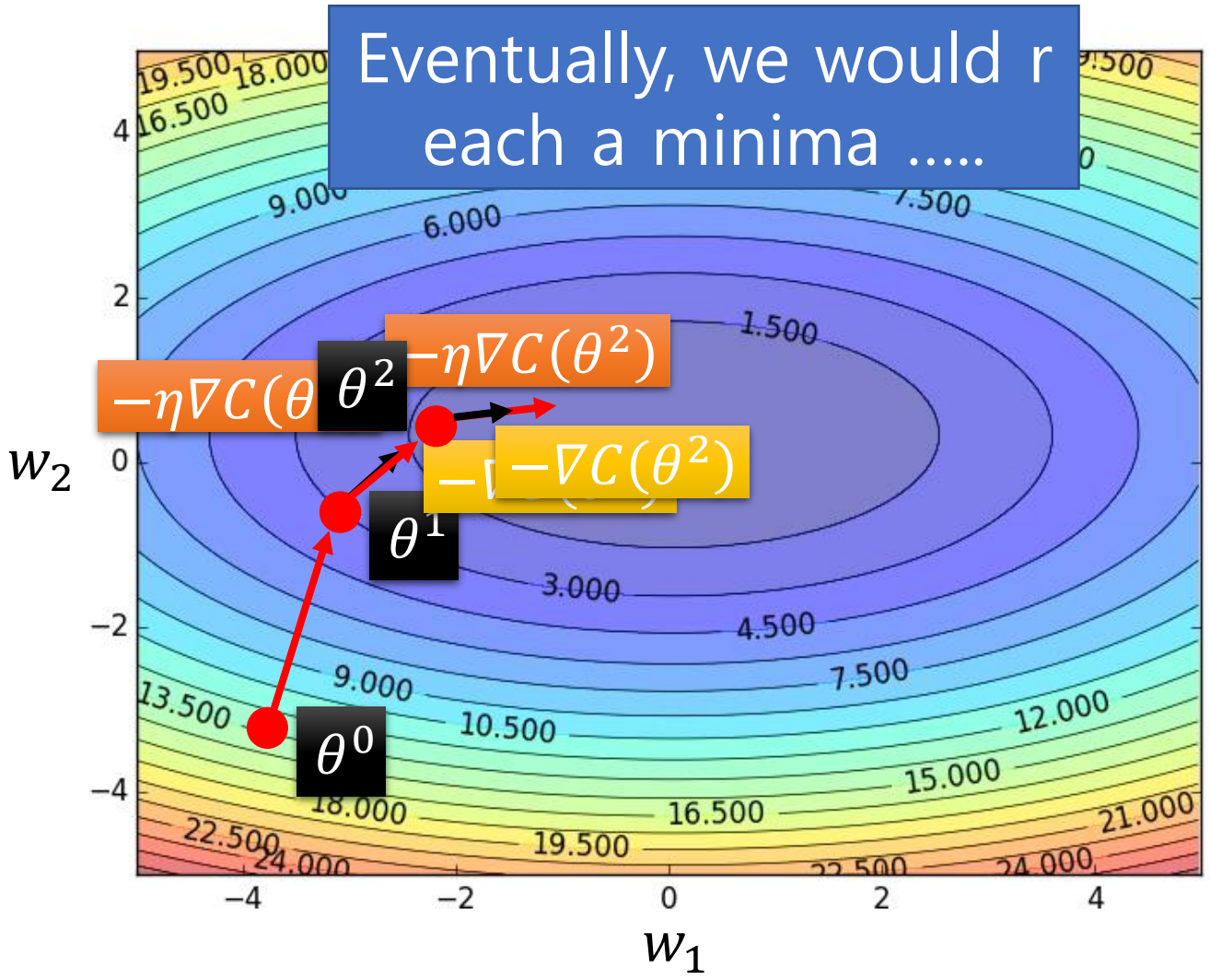
Compute the negative gradient at θ^0

$\Rightarrow -\nabla C(\theta^0)$

Times the learning rate η

$\Rightarrow -\eta\nabla C(\theta^0)$

Gradient Descent



Randomly pick a starting point θ^0

Compute the ne gative gradient at θ^0

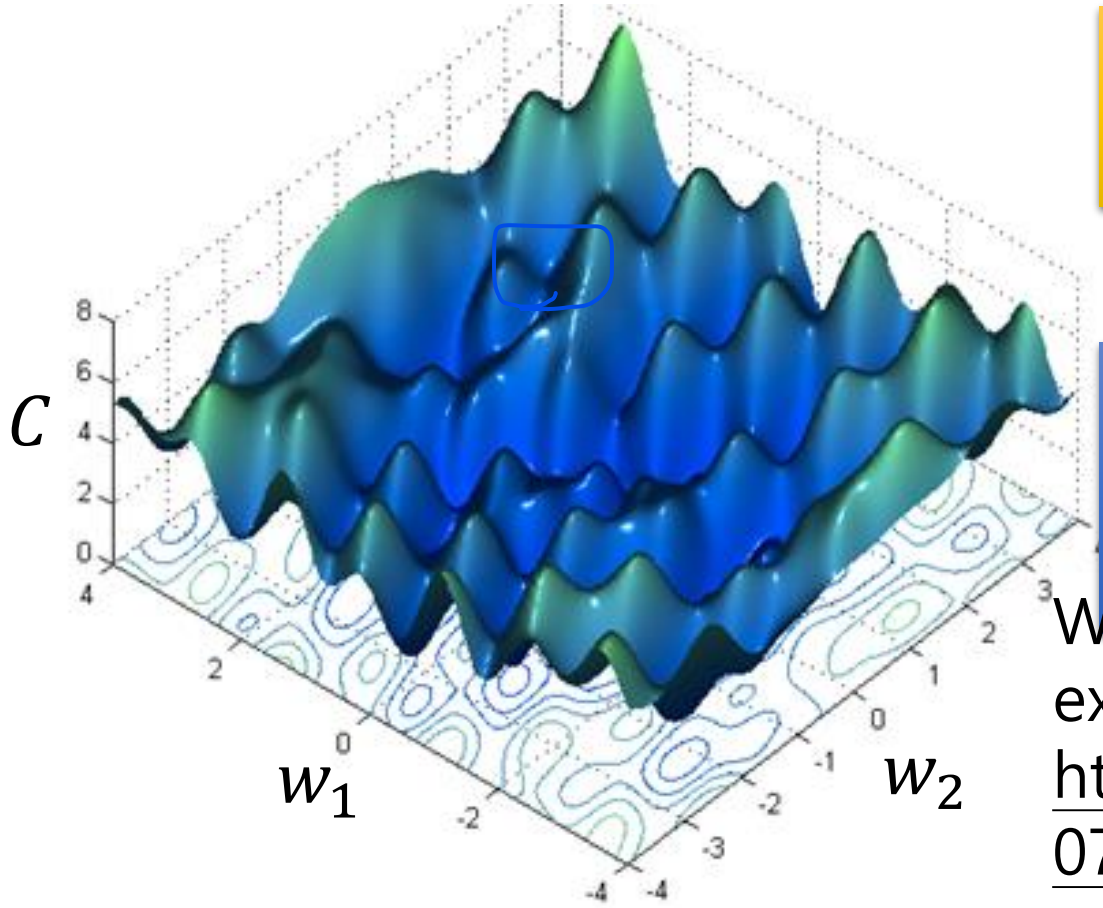
→ $-\nabla C(\theta^0)$

Times the learni ng rate η

→ $-\eta\nabla C(\theta^0)$

Local Minima

- Gradient descent never guarantee global minima

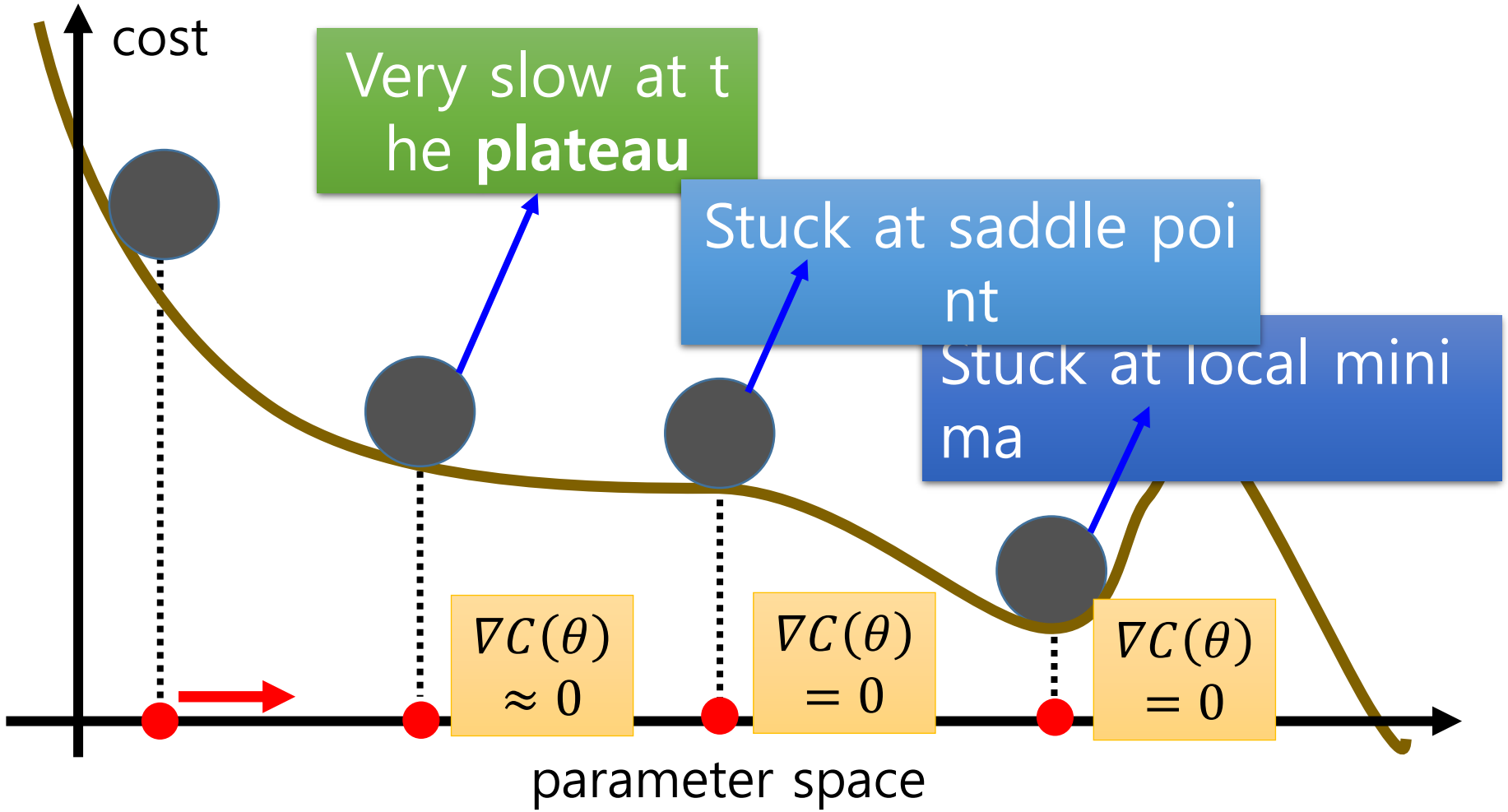


Different initial point θ^0

Reach different minima, so different results

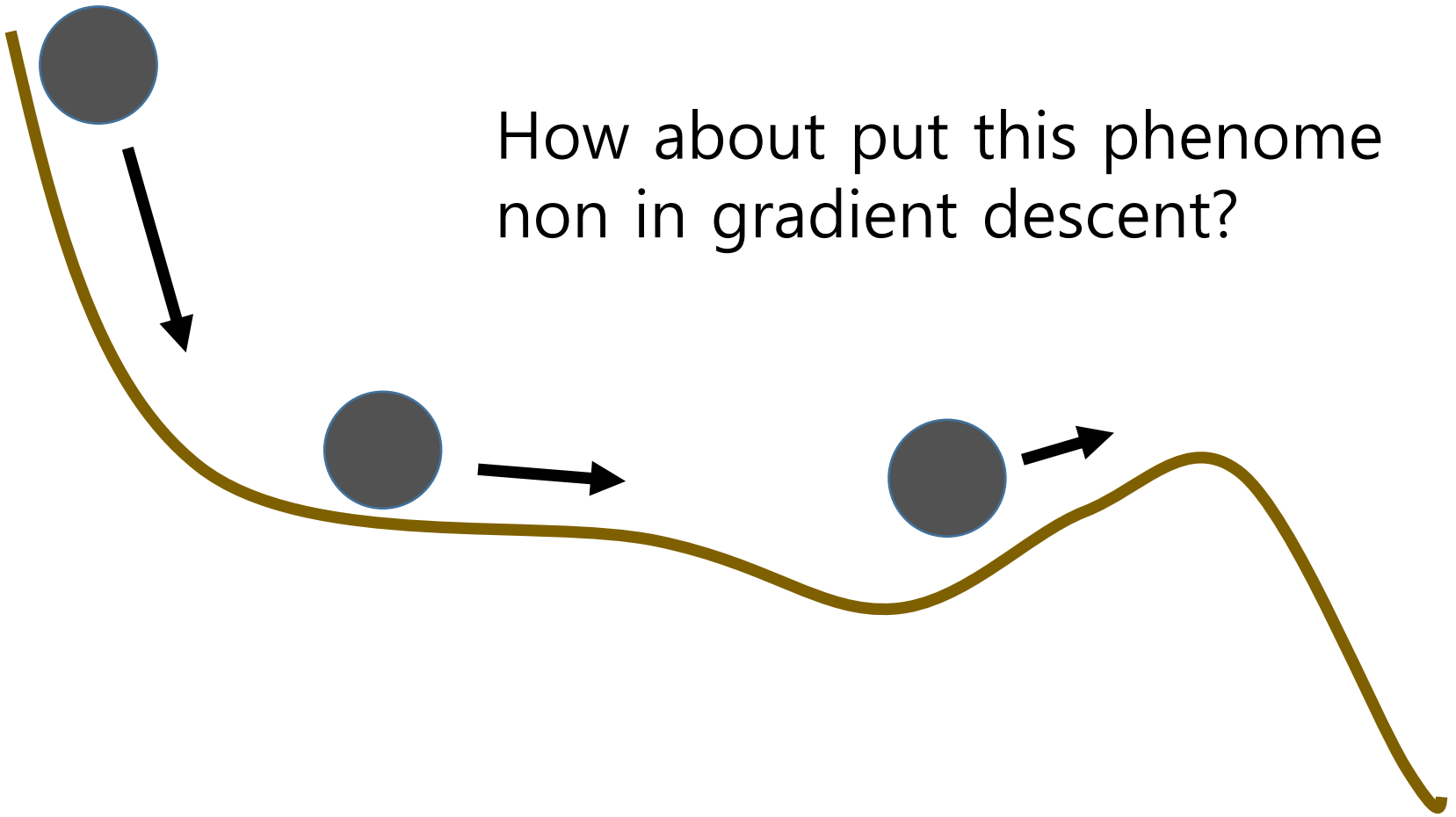
Who is Afraid of Non-Convex Loss Functions?
http://videolectures.net/eml07_lecun_wia/

Besides local minima ……



In physical world ……

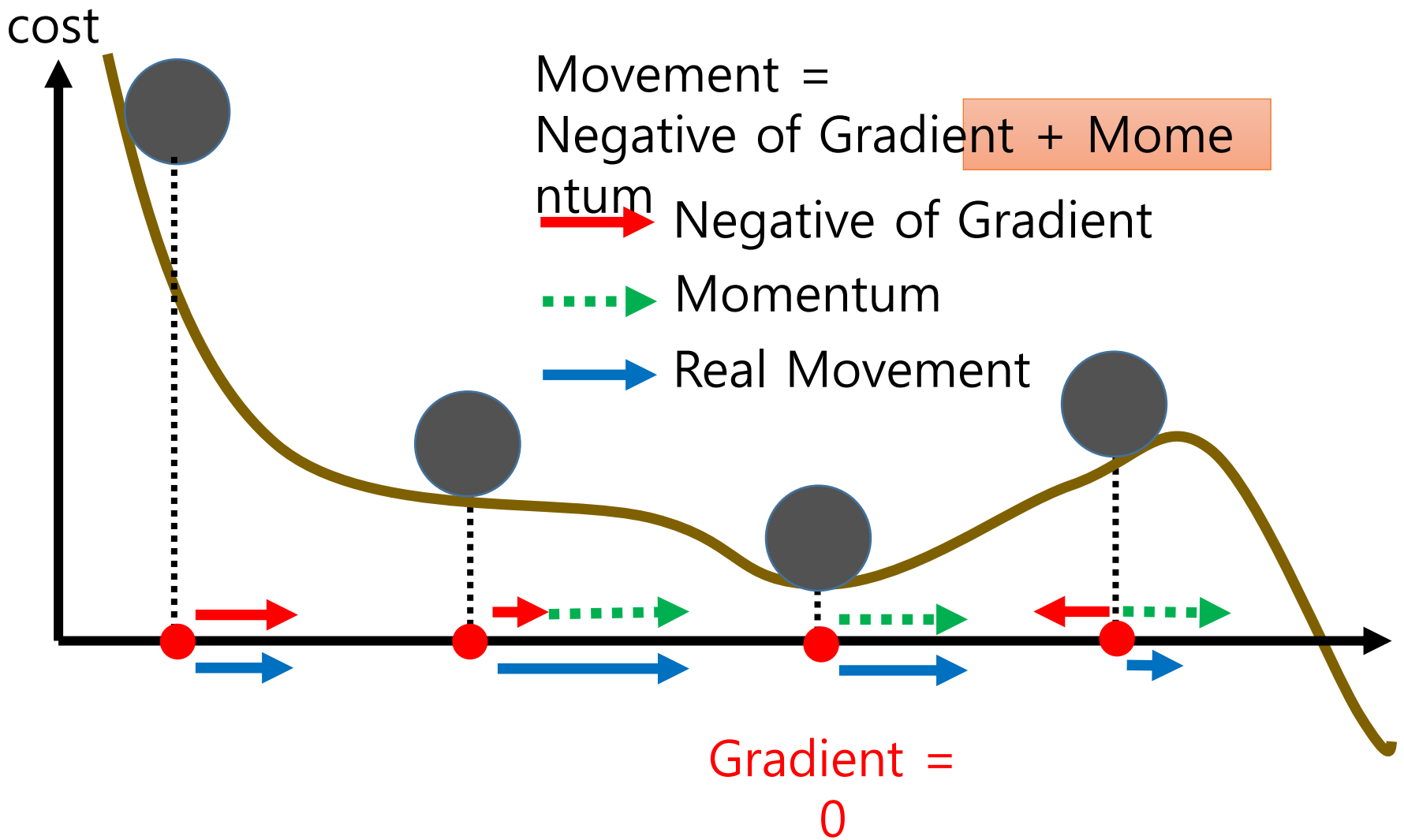
- Momentum



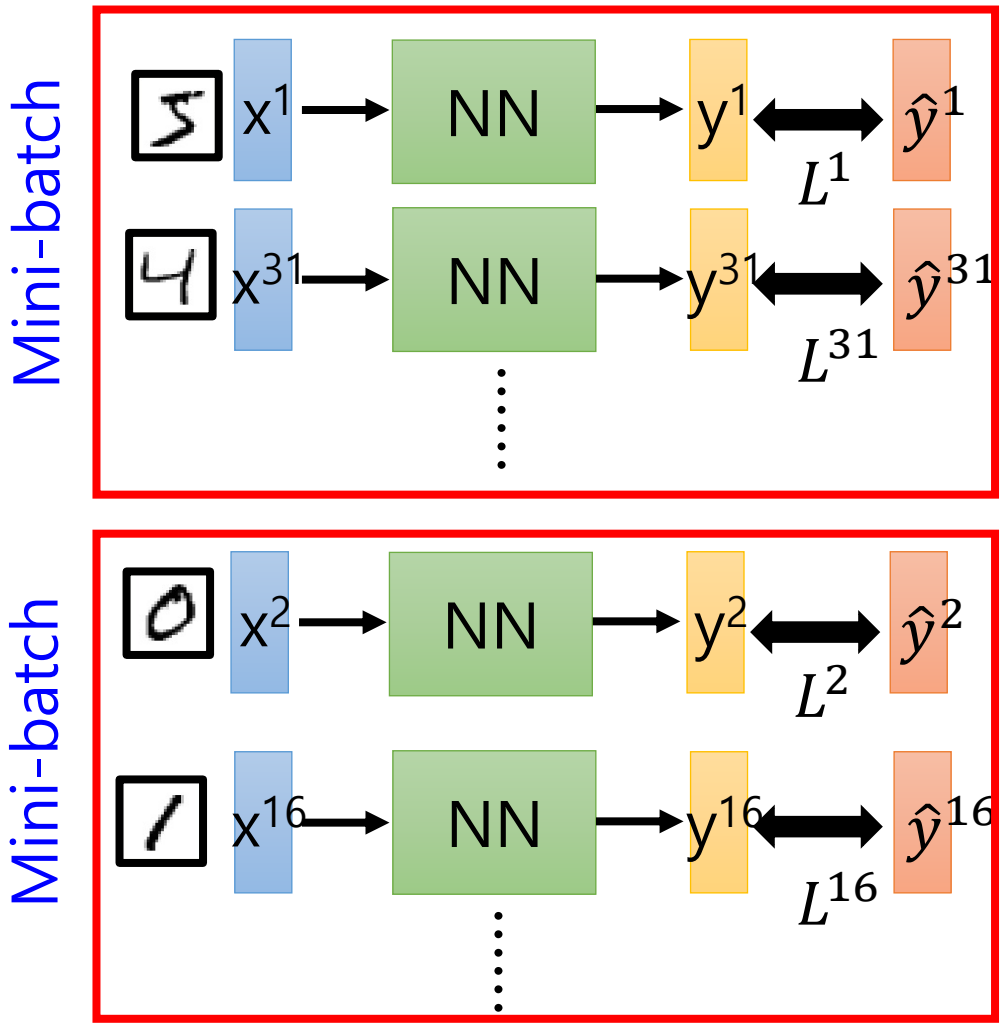
How about put this phenome non in gradient descent?

Momentum

Still not guarantee reaching global minima, but give some hope ……



Mini-batch



- Randomly initialize θ^0
- Pick the 1st batch

$$C = L^1 + L^{31} + \cdots$$

$$\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$$

- Pick the 2nd batch

$$C = L^2 + L^{16} + \cdots$$

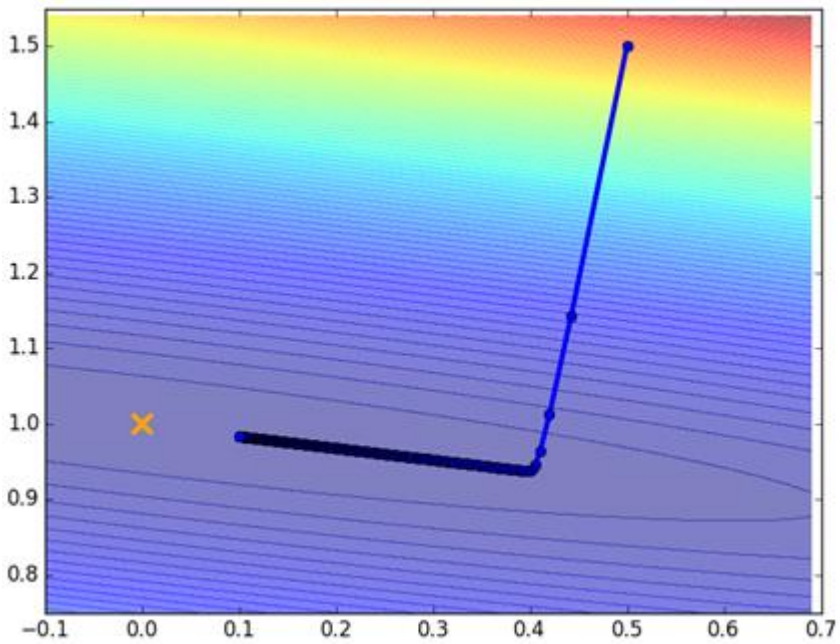
$$\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$$

⋮

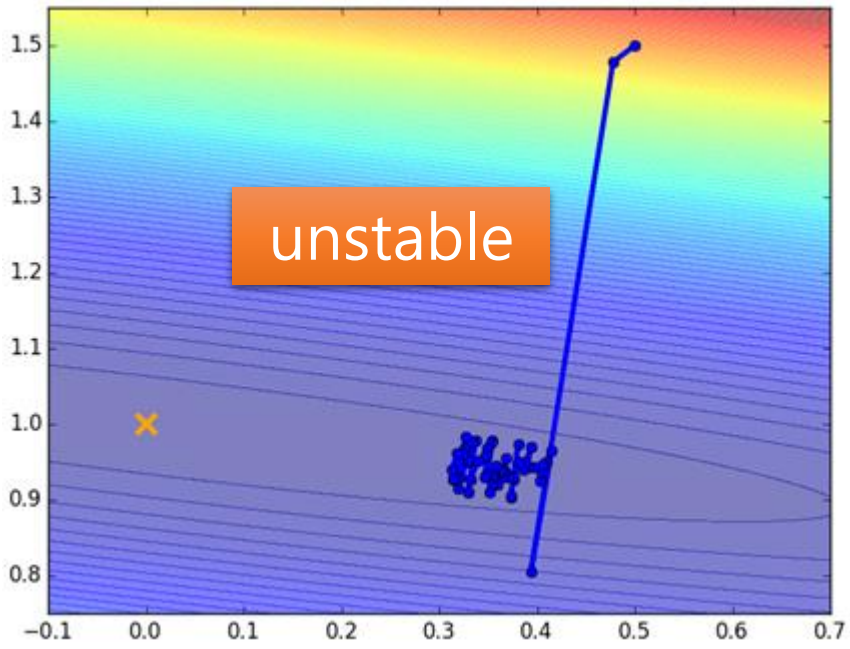
C is different each time when we update parameters!

Mini-batch

Original Gradient Desc



With Mini-batch

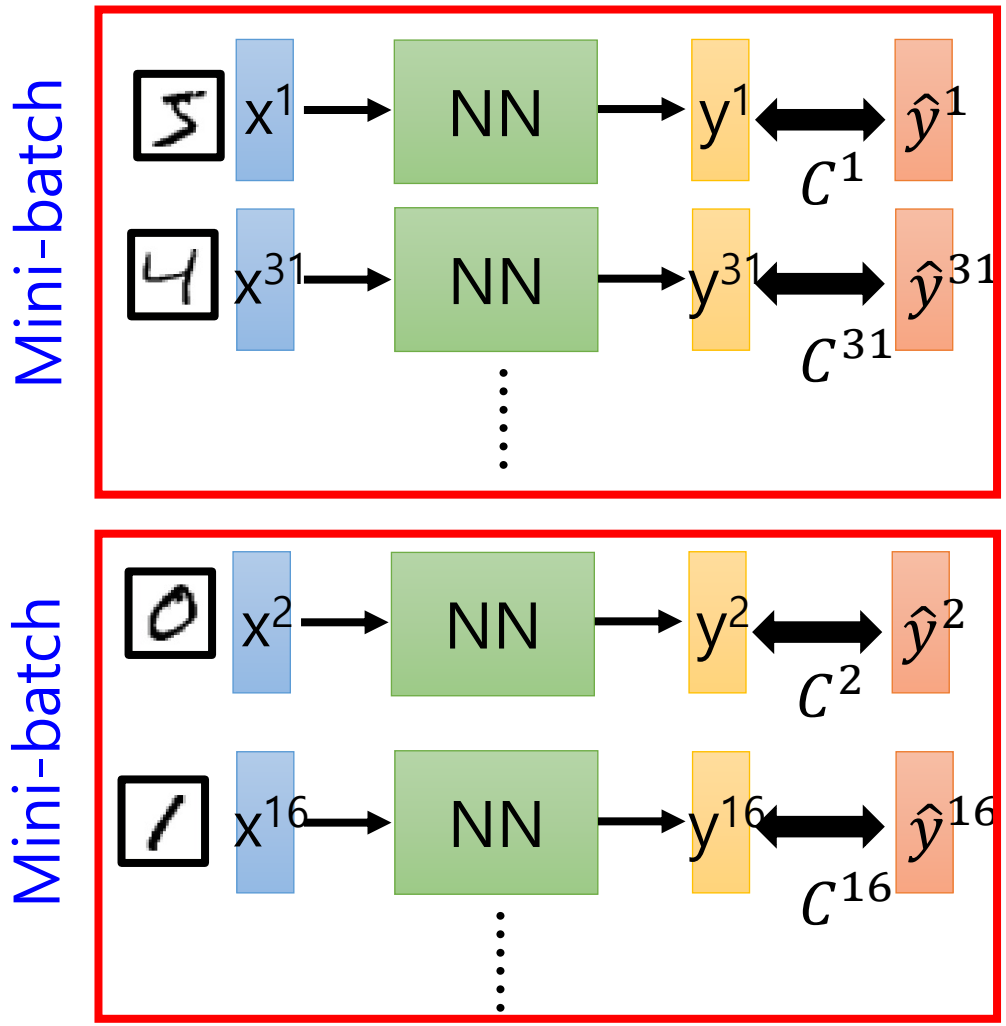


The colors represent the total C on all trainin g data.

Mini-batch

Faster

Better!



- Randomly initialize θ^0
- Pick the 1st batch

  $C = C^1 + C^{31} + \cdots$

  $\theta^1 \leftarrow \theta^0 - \eta \nabla C(\theta^0)$
- Pick the 2nd batch

  $C = C^2 + C^{16} + \cdots$

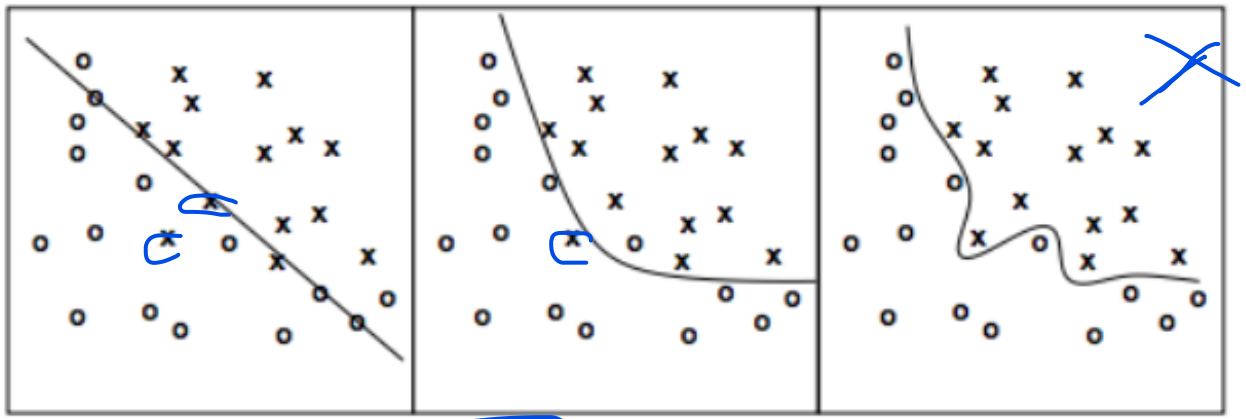
  $\theta^2 \leftarrow \theta^1 - \eta \nabla C(\theta^1)$

  ⋮
- Until all mini-batches have been picked

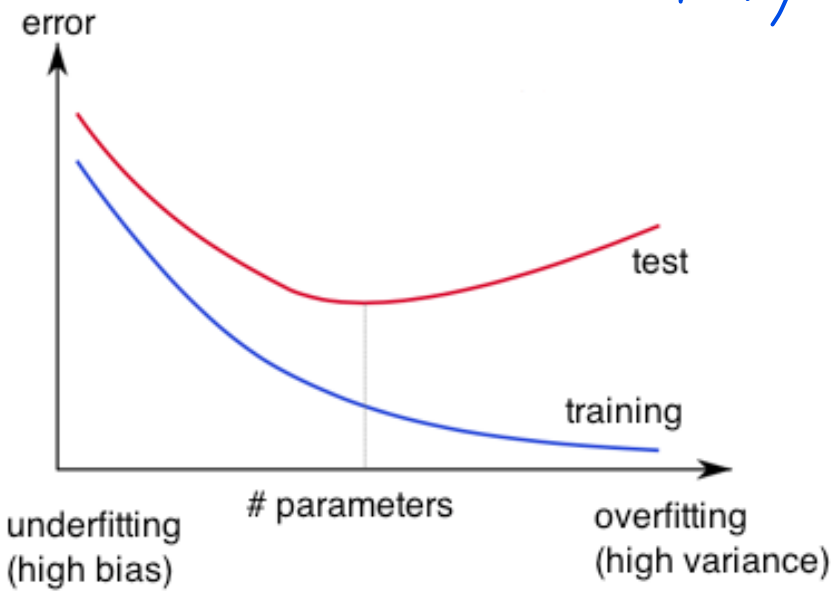
one epoch

Repeat the above process

Overfitting



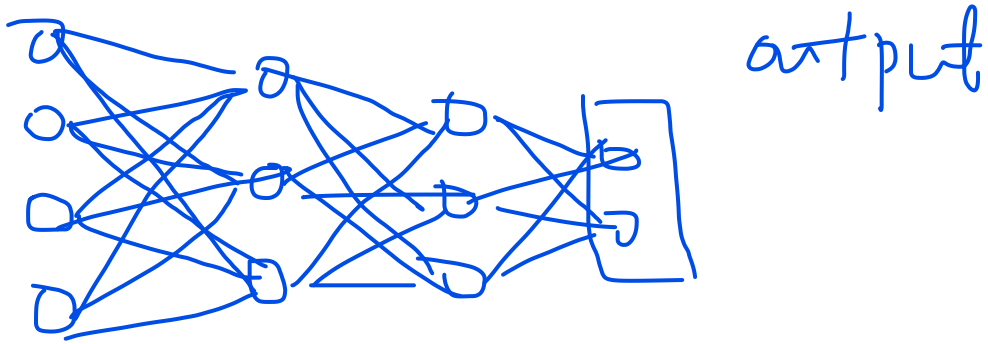
http://wiki.bethanycrane.com/overfitting-of-data



https://www.neuraldesigner.com/images/learning/selection_error.svg

Learned hypothesis may **fit** the training data very well, even out liers (**noise**) but fail to **generaliz e** to new examples (test data)

Backpropagation



- A network can have millions of parameters.
  - Backpropagation is the way to compute the gradients efficiently (not today)
  - Ref: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/DNN%20backprop.ecm.mp4/index.html
- Many toolkits can compute the gradients automatically

theano

TensorFlow

Ref: http://speech.ee.ntu.edu.tw/~tlkagk/courses/MLDS_2015_2/Lecture/Theano%20DNN.ecm.mp4/index.html

Deep N.N → 1st : 학습 data

2nd : 검증/

3rd : 학습 → 완료 → weight

고정

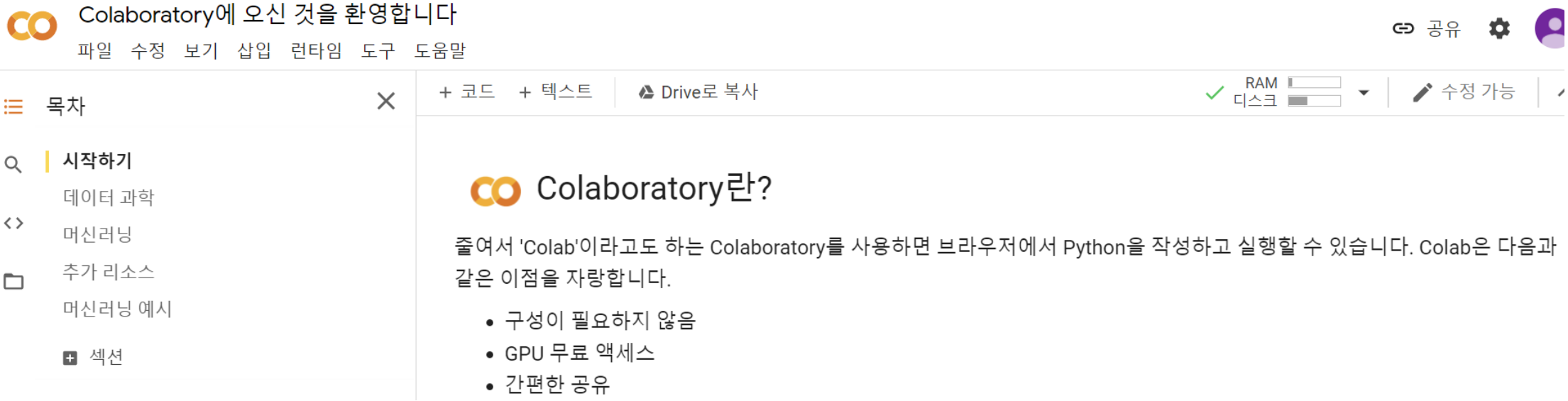
4. Environment Setup

컴퓨터 w → □ => test

- If you have a GPU → Install CUDA
- Without GPU: If the data becomes large, training is not practical

- Most popular development environments
  - GPU
  - Pytorch, Tensorflow, Keras

- Simple environment with no setup required
  - Google → colab.
  - https://colab.research.google.com/

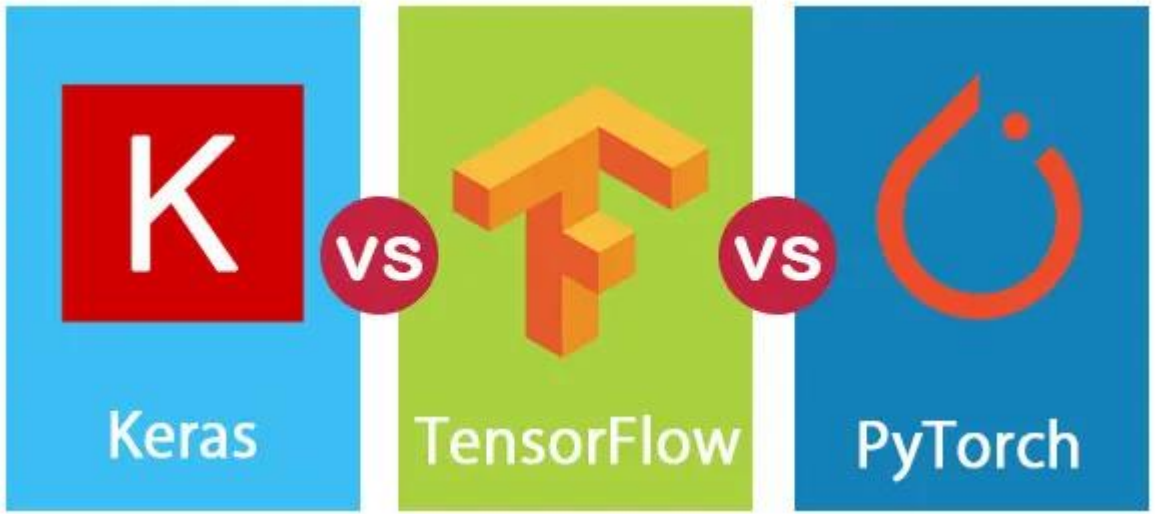

Colaboratory에 오신 것을 환영합니다
파일 수정 보기 삽입 런타임 도구 도움말
공유
목차
시작하기
데이터 과학
머신러닝
추가 리소스
머신러닝 예시
섹션
+ 코드 + 텍스트
Drive로 복사
RAM
디스크
수정 가능
Colaboratory란?
줄여서 'Colab'이라고도 하는 Colaboratory를 사용하면 브라우저에서 Python을 작성하고 실행할 수 있습니다. Colab은 다음과 같은 이점을 자랑합니다.
• 구성이 필요하지 않음
• GPU 무료 액세스
• 간편한 공유

Keras Vs Tensorflow

- Tensorflow is a well-known machine learning framework developed by Google and released as an open source.
- Keras is also a framework, but Keras is a framework that runs on top of Tensorflow.

- Tensorflow?
  - If there is a high possibility of using a new method rather than just using an existing function, it is recommended to use Tensorflow.
  - So, if you are observing neural networks closely and need to do research and development, Tensorflow is much better.
  - Keras is said to have been integrated since Tensorflow 1.2
  - So you can use Keras inside TensorFlow with tf.keras.

Difference Between Keras vs TensorFlow vs PyTorch

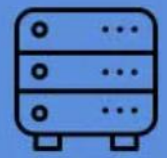


https://www.educba.com/keras-vs-tensorflow-vs-pytorch/

#1. Definition

Keras	TensorFlow	PyTorch
		
The Neural network library is available as an open-source.	TensorFlow is available as an open-source and a free software library.	It is a machine learning library that is available as an open-source.

#2. Coding Language

Keras	TensorFlow	PyTorch
		
It is available as a coding. All the codes are scripted in a single line.	The library is compact with C, C++, Java, and other coding languages. The accuracy is increased by programming it with small codes.	It is scripted only with python. The codes of PyTorch is scripted with larger lines.

Keras : A high-level API written in the Python language. A library that runs on top of Tensorflow. Supported by Google. Relatively easy. Not available for Pytorch.
Good for beginner.

#3. Applications

Keras

It is designed to perform robust experiments in neural networks.

TensorFlow

It is employed to teach the machine about multiple computational techniques.

PyTorch

It is used to build natural language processing and neural networks.

#4. Level of API's

Keras

It can execute on Theano and CNTK as it has high-level API.

TensorFlow

It comprises of both low level and high-level API's.

PyTorch

PyTorch focuses only on array expression because of its low-level API.

#5. Architecture

Keras

It has an understandable syntax and can be easily interpretable.

TensorFlow

It is popular because of its rapid computation ability in the various platform but has little complex architecture which is difficult to interpret

PyTorch

The beginners feel complicated at PyTorch's architecture but they are interested in its deep learning application and also used for various academic purposes.

#6. Speed

Keras

It operates at the minimum speed only.

TensorFlow

It works on maximum speed which in turns provides high performance.

PyTorch

The performance and speed of PyTorch are similar to TensorFlow.

#7. Dataset

Keras

It operates effectively in the smaller dataset as the speed of execution is low.

TensorFlow

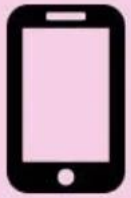
It is highly capable to manage large dataset as it has a maximum speed of execution.

PyTorch

It can manage a high-performance task in a higher dimensional dataset.

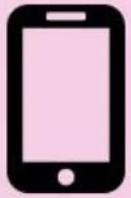
#8. Debugging

Keras



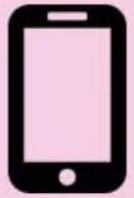
The administrator need not require any frequent process of debugging.

TensorFlow



It is challenging to perform debugging.

PyTorch



The abilities to debug is better than Keras and TensorFlow.

#9. Popularity

Keras

It is widely used in neural networks and supports convolutional and utility layers.

TensorFlow

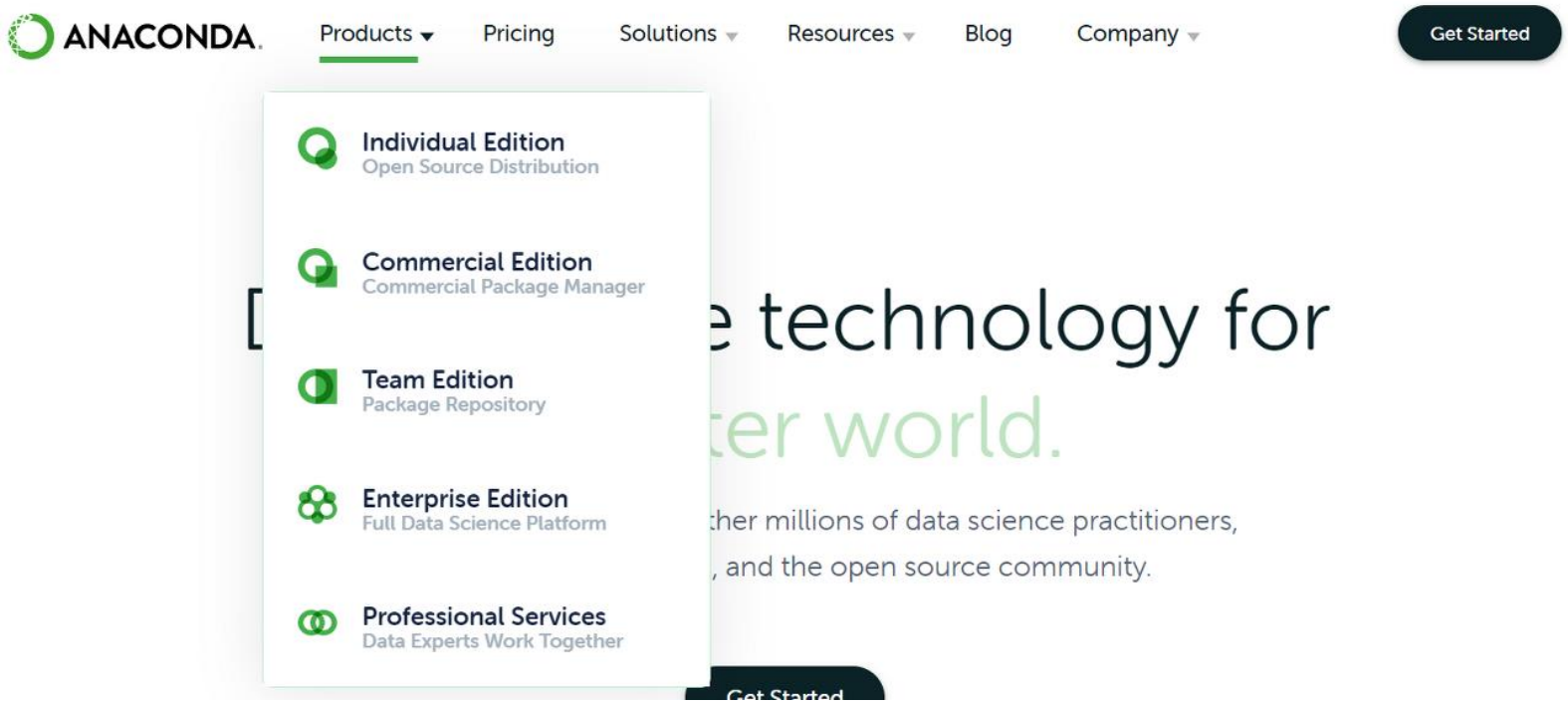
It is famous for its automated image capturing software and its internal use of google.

PyTorch

It is popular because of its automatic differentiation on deep learning networks and supports high power GPU applications with the NN module, optimum module, and autograd module.

Install Conda

1. Install Conda
2. www.anaconda.com/



Individual Edition

Your data science toolkit

With over 20 million users worldwide, the open-source Individual Edition (Distribution) is the easiest way to perform Python/R data science and machine learning on a single machine. Developed for solo practitioners, it is the toolkit that equips you to work with thousands of open-source packages and libraries.

Download

Anaconda Installers

Windows

Python 3.8

64-Bit Graphical Installer (466 MB)

32-Bit Graphical Installer (397 MB)

MacOS

Python 3.8

64-Bit Graphical Installer (462 MB)

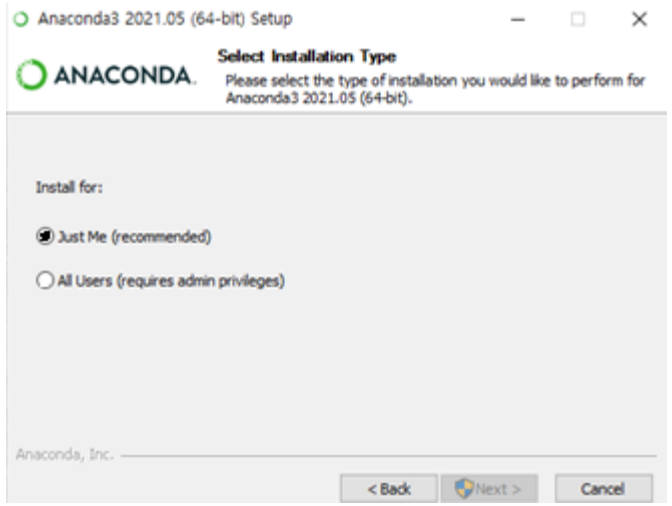
64-Bit Command Line Installer (454 MB)

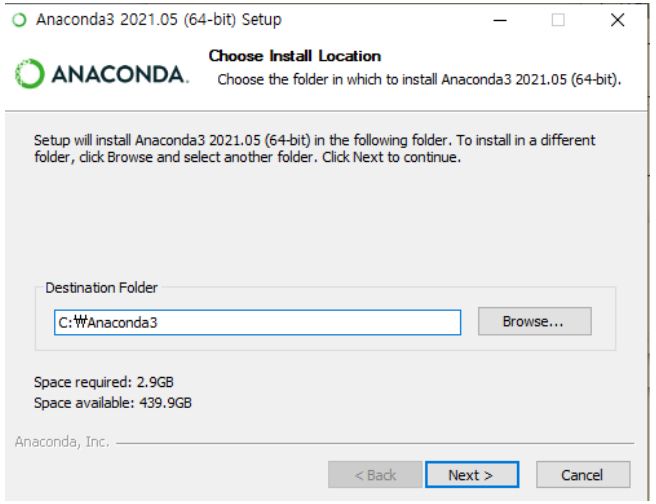
Linux

Python 3.8

64-Bit (x86) Installer (550 MB)

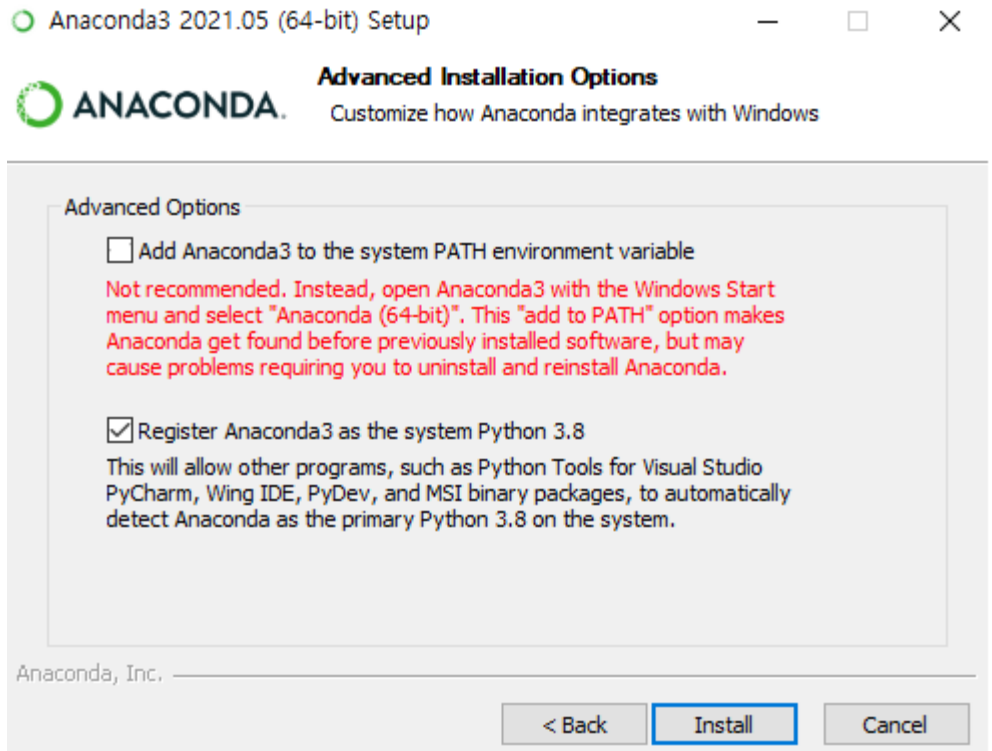
64-Bit (Power8 and Power9) Installer (290 MB)

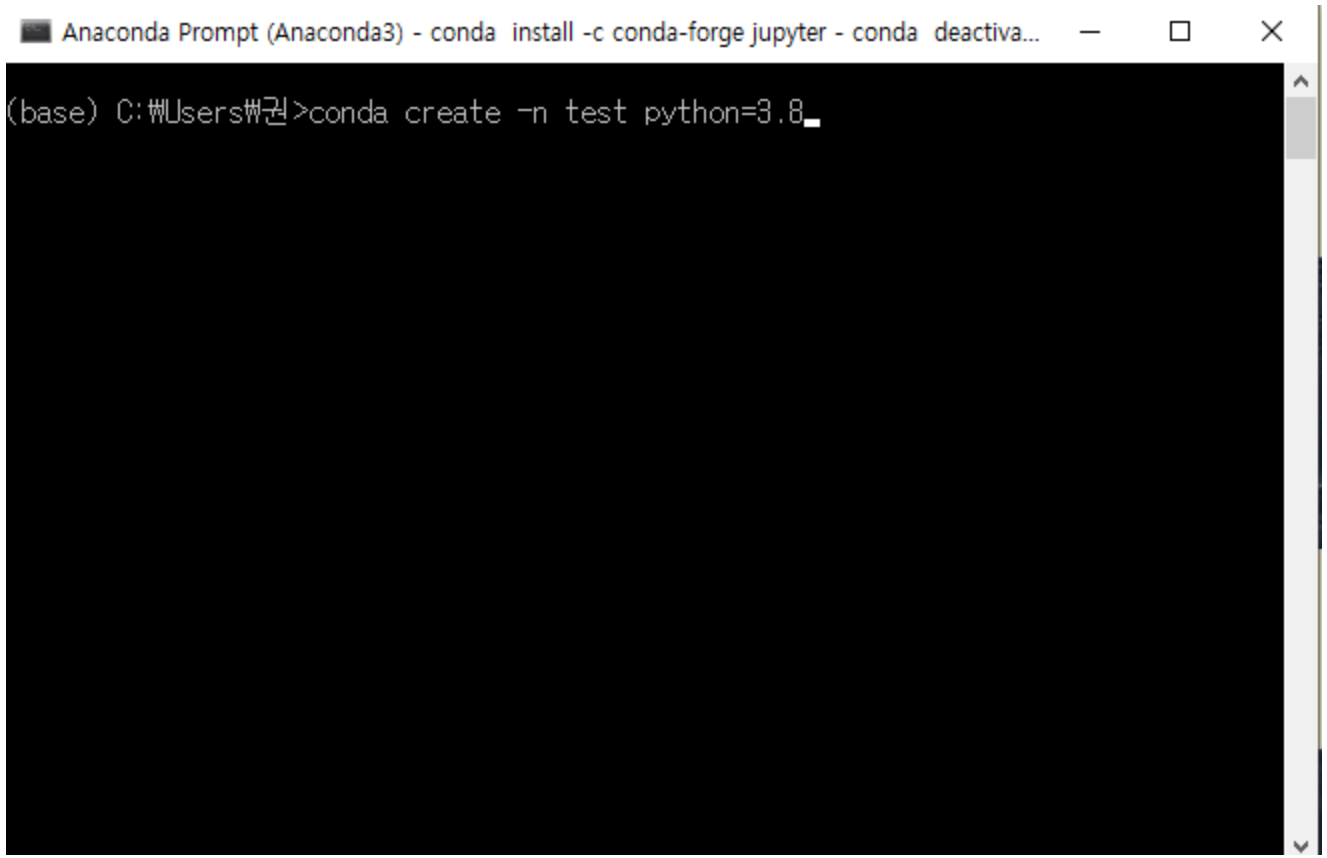


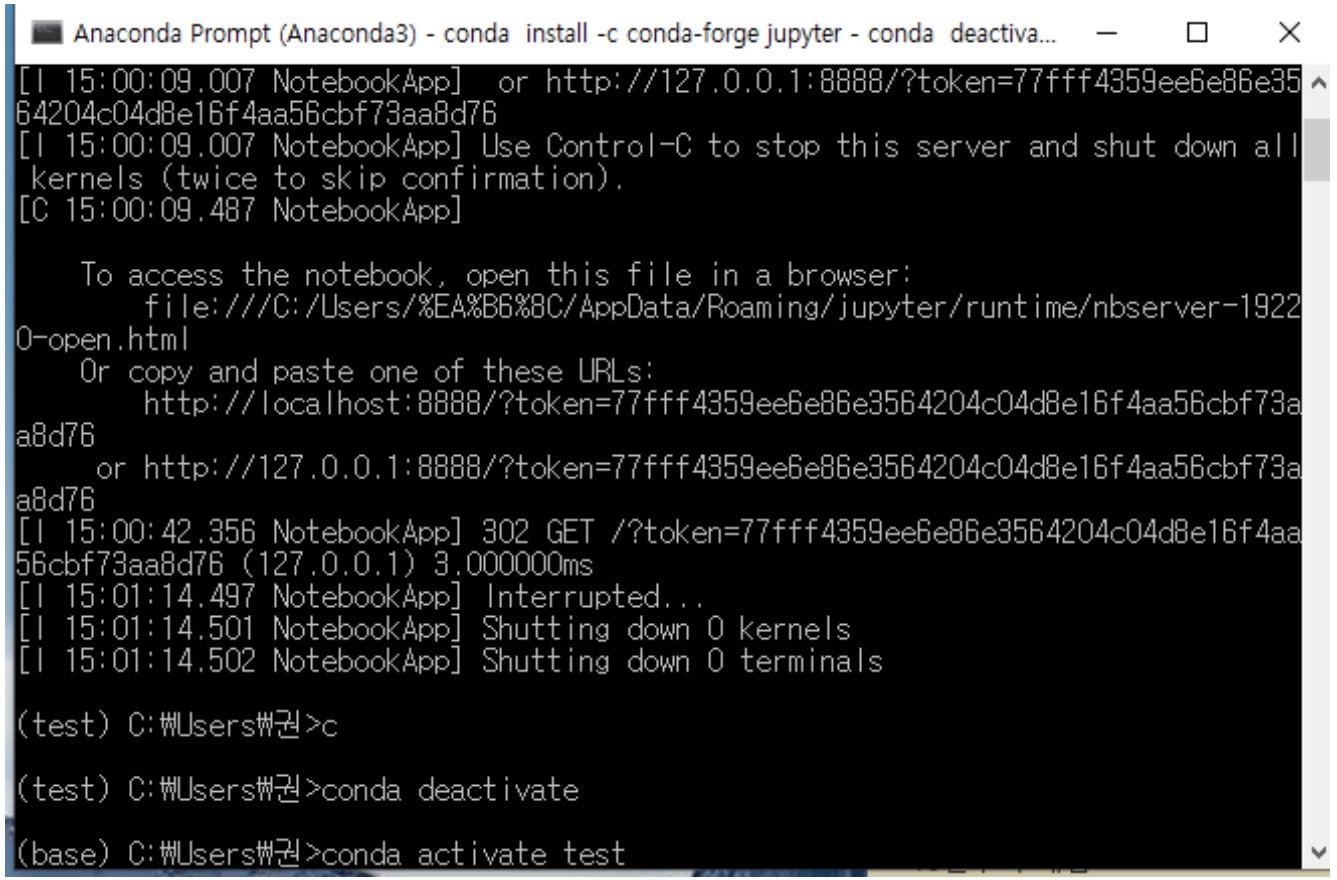


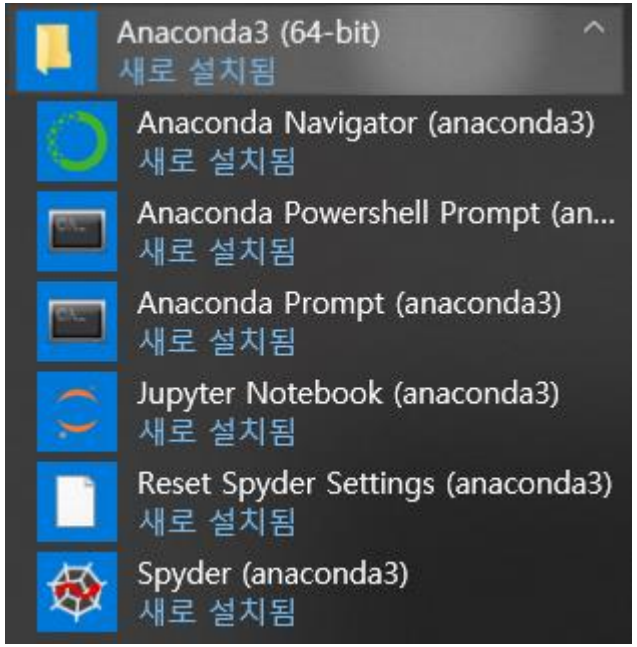
- 사용자 이름은 반드시 영어야 한다!
- Windows 사용자 이름이 한글이면 administrator 계정을 하나 더 만들어 사용하거나 사용자 이름을 영어로 변경 후 설치
- 변경 방법은 google srearch


Anaconda3 2020.07 (64-bit) Setup
ANACONDA.
Welcome to Anaconda3 2020.07 (64-bit) Setup
Setup will guide you through the installation of Anaconda3 2020.07 (64-bit).
It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.
Click Next to continue.
Next >
Cancel

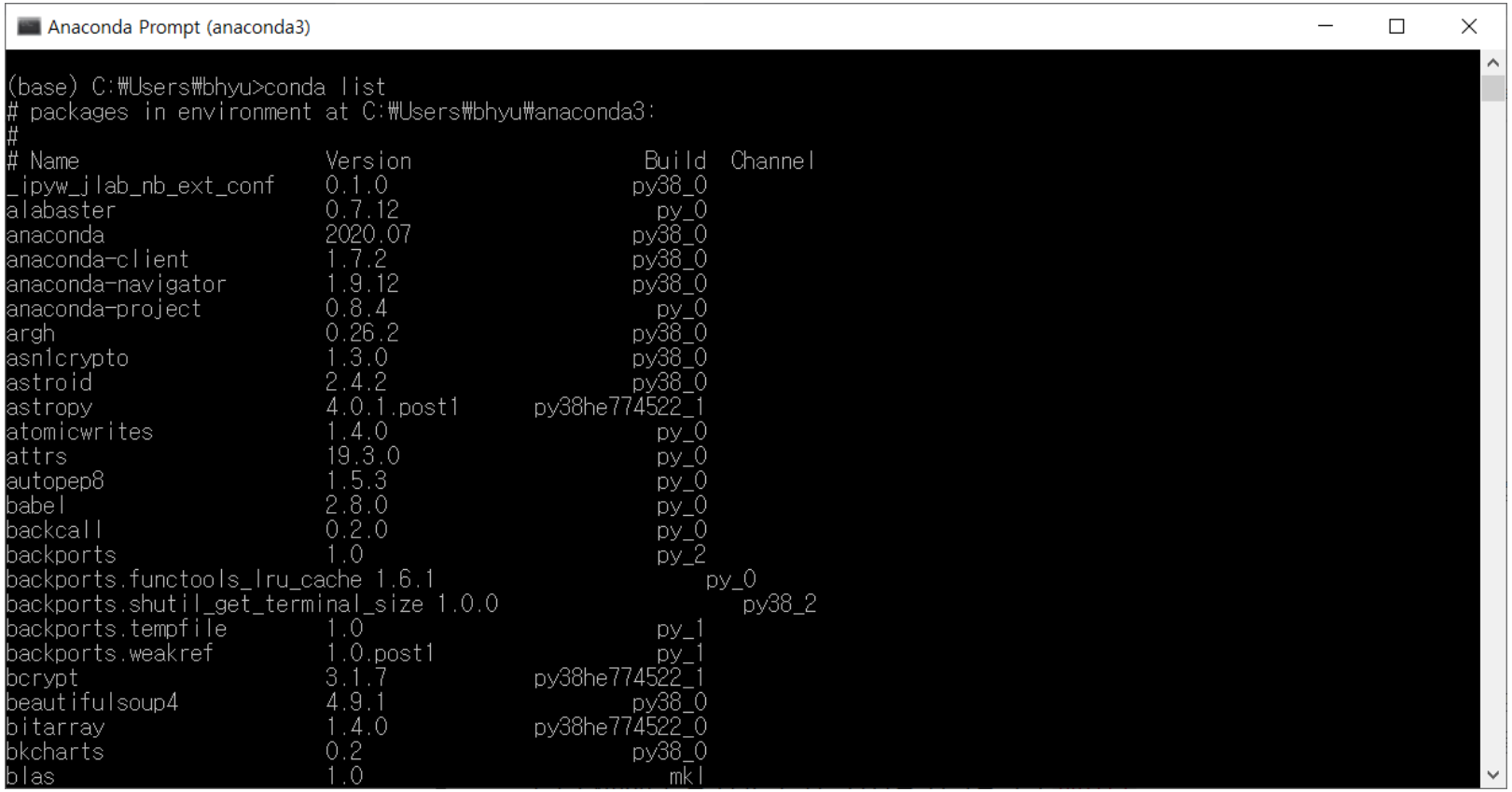

Anaconda3 2021.05 (64-bit) Setup
ANACONDA.
Advanced Installation Options
Customize how Anaconda integrates with Windows
Advanced Options
Add Anaconda3 to the system PATH environment variable
Not recommended. Instead, open Anaconda3 with the Windows Start menu and select "Anaconda (64-bit)". This "add to PATH" option makes Anaconda get found before previously installed software, but may cause problems requiring you to uninstall and reinstall Anaconda.
Register Anaconda3 as the system Python 3.8
This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 3.8 on the system.
Anaconda, Inc.
< Back
Install
Cancel


Anaconda Prompt (Anaconda3) - conda install -c conda-forge jupyter - conda deactiva...
(base) C:₩Users₩권>conda create -n test python=3.8


Anaconda Prompt (Anaconda3) - conda install -c conda-forge jupyter - conda deactiva...
[I 15:00:09.007 NotebookApp] or http://127.0.0.1:8888/?token=77fff4359ee6e86e3564204c04d8e16f4aa56cbf73aa8d76
[I 15:00:09.007 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 15:00:09.487 NotebookApp]
To access the notebook, open this file in a browser:
file:///C:/Users/%EA%B6%8C/AppData/Roaming/jupyter/runtime/nbserver-19220-open.html
Or copy and paste one of these URLs:
http://localhost:8888/?token=77fff4359ee6e86e3564204c04d8e16f4aa56cbf73aa8d76
or http://127.0.0.1:8888/?token=77fff4359ee6e86e3564204c04d8e16f4aa56cbf73aa8d76
[I 15:00:42.356 NotebookApp] 302 GET /?token=77fff4359ee6e86e3564204c04d8e16f4aa56cbf73aa8d76 (127.0.0.1) 3.000000ms
[I 15:01:14.497 NotebookApp] Interrupted...
[I 15:01:14.501 NotebookApp] Shutting down 0 kernels
[I 15:01:14.502 NotebookApp] Shutting down 0 terminals
(test) C:₩Users₩권>c
(test) C:₩Users₩권>conda deactivate
(base) C:₩Users₩권>conda activate test

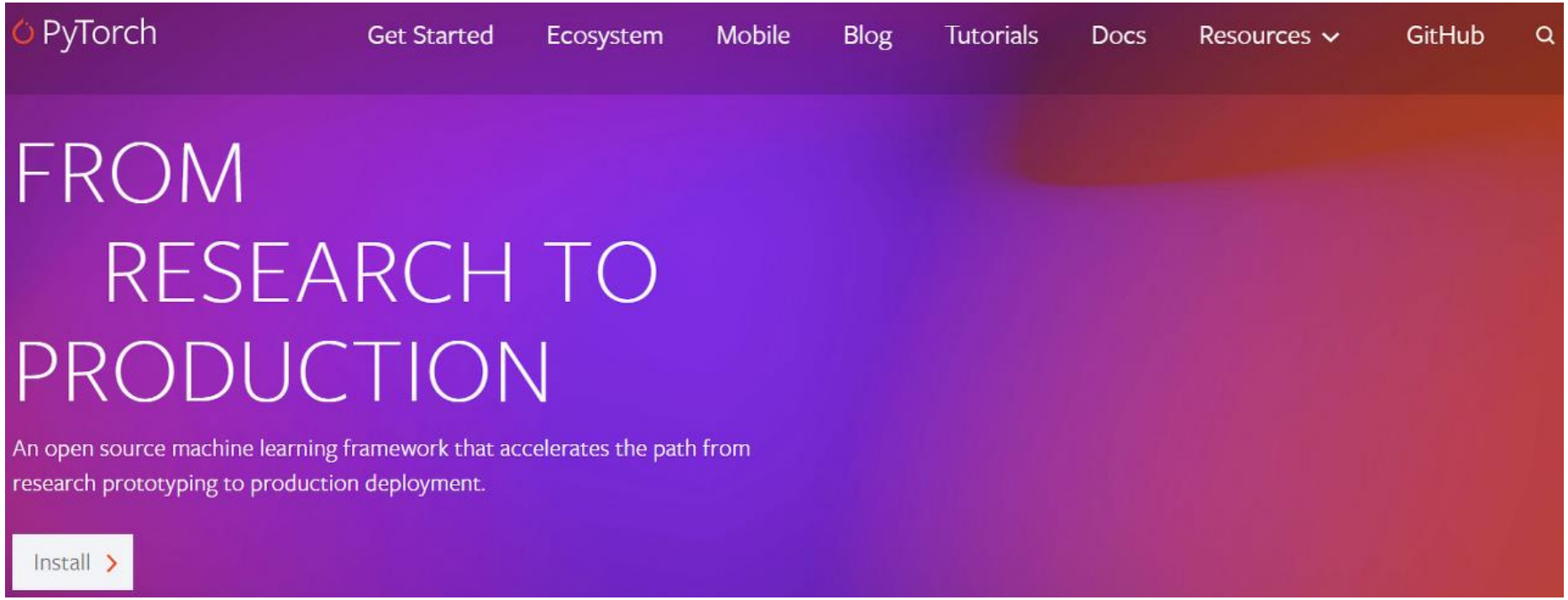


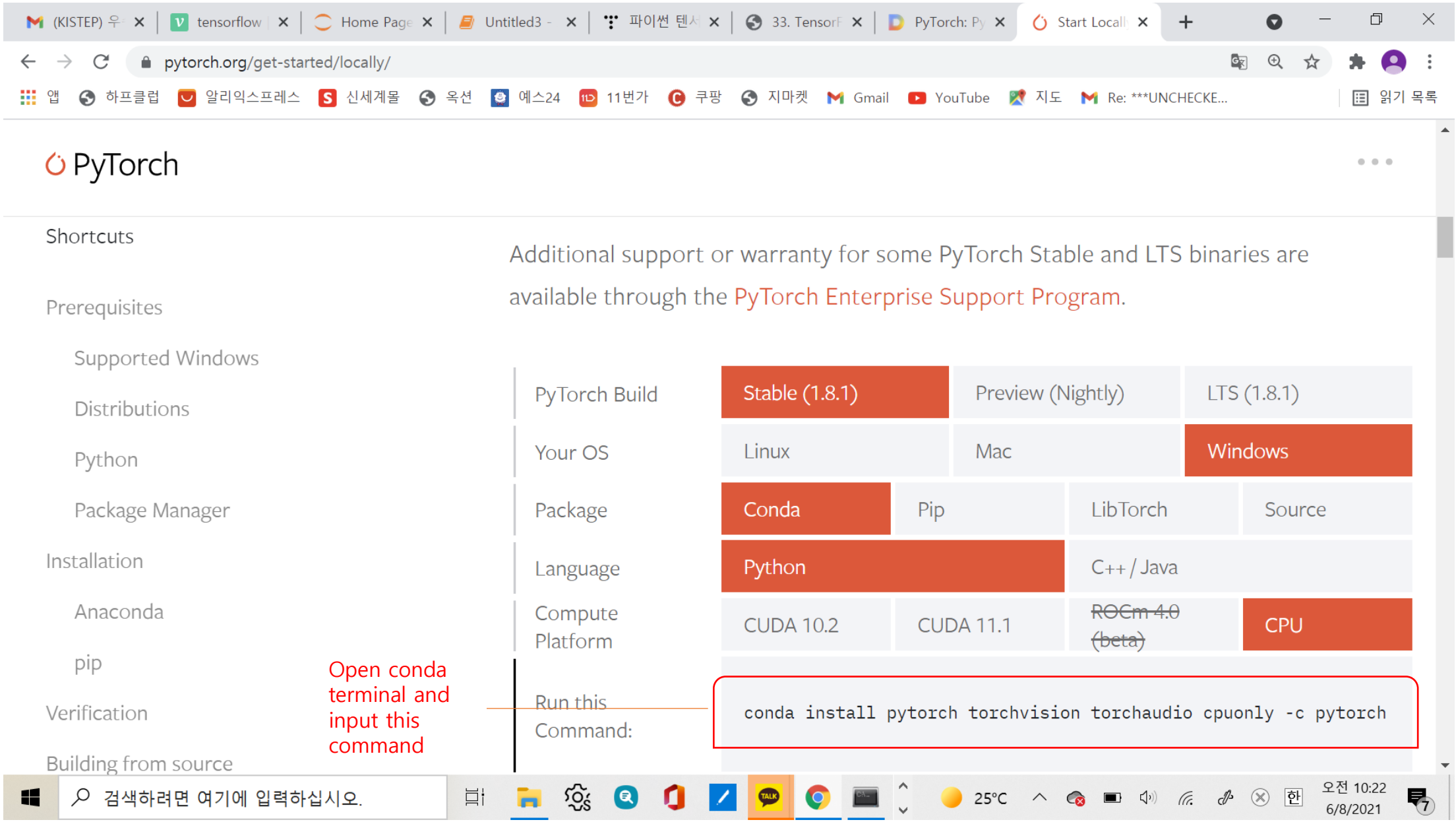
- Python version 확인

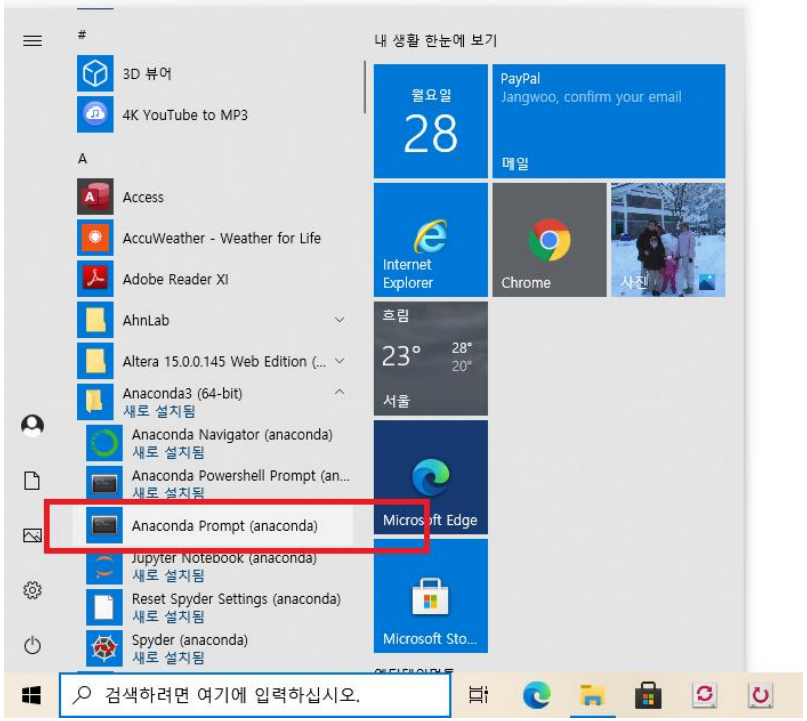


Pytorch Install

1. Install pytorch
2. -https://pytorch.org

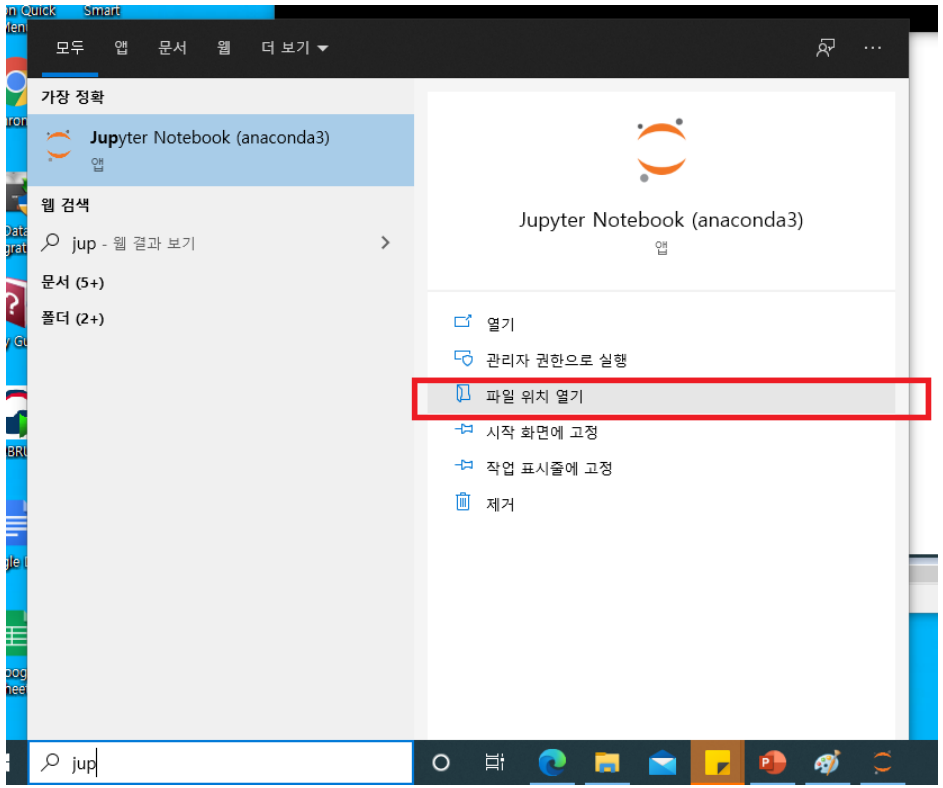


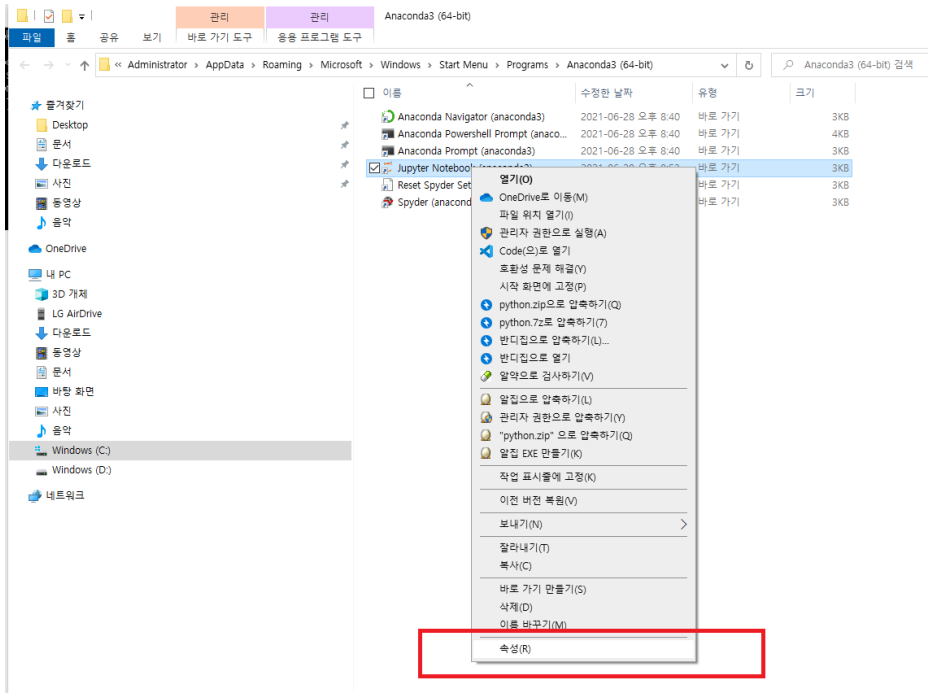

(KISTEP) 우
tensorflow
Home Page
Untitled3 -
파이썬 텐서
33. TensorF
PyTorch: Py
Start Locall
pytorch.org/get-started/locally/
앱
하프클럽
알리익스프레스
신세계몰
옥션
예스24
11번가
쿠팡
지마켓
Gmail
YouTube
지도
Re: ***UNCHECKE...
읽기 목록
PyTorch
Shortcuts
Prerequisites
Supported Windows
Distributions
Python
Package Manager
Installation
Anaconda
pip
Verification
Building from source
Additional support or warranty for some PyTorch Stable and LTS binaries are available through the PyTorch Enterprise Support Program.
PyTorch Build
Stable (1.8.1)
Preview (Nightly)
LTS (1.8.1)
Your OS
Linux
Mac
Windows
Package
Conda
Pip
LibTorch
Source
Language
Python
C++ / Java
Compute Platform
CUDA 10.2
CUDA 11.1
ROCm 4.0 (beta)
CPU
Run this Command:
conda install pytorch torchvision torchaudio cpuonly -c pytorch
Open conda terminal and input this command
검색하려면 여기에 입력하십시오.
25°C
오전 10:22
6/8/2021


#
3D 뷰어
4K YouTube to MP3
A
Access
AccuWeather - Weather for Life
Adobe Reader XI
AhnLab
Altera 15.0.0.145 Web Edition (...
Anaconda3 (64-bit)
새로 설치됨
Anaconda Navigator (anaconda)
새로 설치됨
Anaconda Powershell Prompt (an...
새로 설치됨
Anaconda Prompt (anaconda)
Jupyter Notebook (anaconda)
새로 설치됨
Reset Spyder Settings (anaconda)
새로 설치됨
Spyder (anaconda)
새로 설치됨
내 생활 한눈에 보기
월요일
28
PayPal
Jangwoo, confirm your email
메일
Internet Explorer
Chrome
사진
흐림
23°
28°
20°
서울
Microsoft Edge
Microsoft Sto...
검색하려면 여기에 입력하십시오.

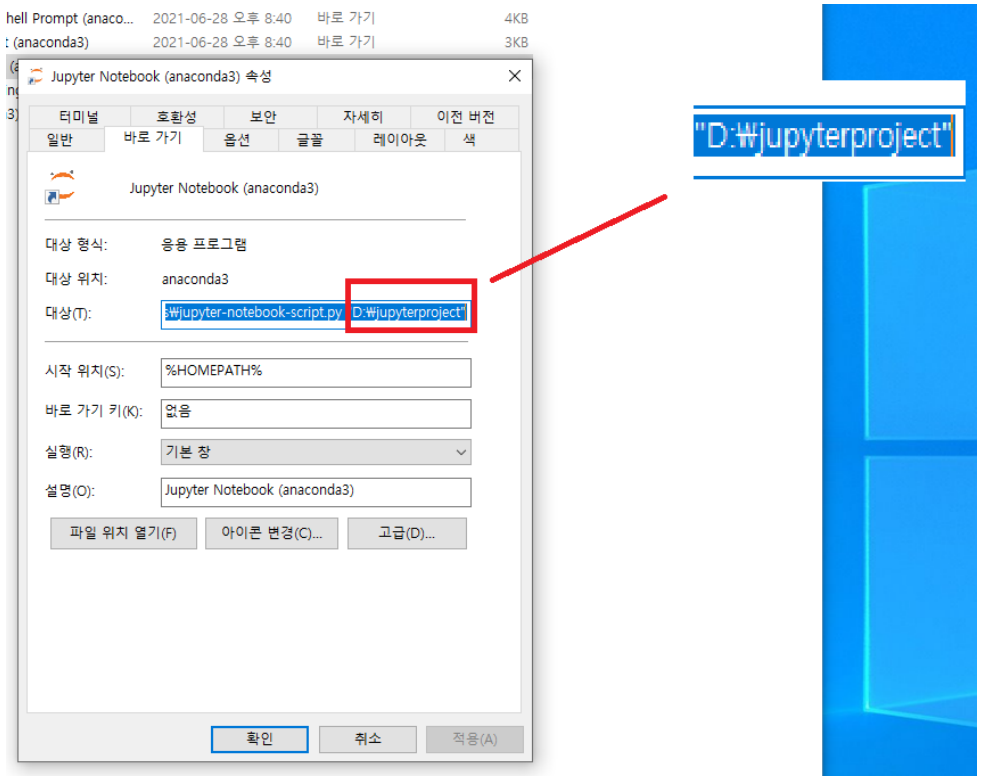

Anaconda Prompt (anaconda) - conda install pytorch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0 -c pytorch
(base) C:₩Users₩권>conda install pytorch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0 -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done
Package Plan
environment location: C:₩anaconda
added / updated specs:
- pytorch==1.8.0
- torchaudio==0.8.0
- torchvision==0.9.0
The following packages will be downloaded:
package | build
cudatoolkit-10.2.89 h74a9793_1 317.2 MB
libuv-1.40.0 he774522_0 265 KB
ninja-1.10.2 hbd14046_1 250 KB
pytorch-1.8.0 py3.8_cuda10.2_cudnn7_0 858.0 MB pytorch
torchaudio-0.8.0 py38 2.7 MB pytorch
torchvision-0.9.0 py38_cu102 7.3 MB pytorch
Total: 1.16 GB
The following NEW packages will be INSTALLED:
cudatoolkit pkgs/main/win-64::cudatoolkit-10.2.89-h74a9793_1
libuv pkgs/main/win-64::libuv-1.40.0-he774522_0
ninja pkgs/main/win-64::ninja-1.10.2-hbd14046_1
pytorch pytorch/win-64::pytorch-1.8.0-py3.8_cuda10.2_cudnn7_0
torchaudio pytorch/win-64::torchaudio-0.8.0-py38
torchvision pytorch/win-64::torchvision-0.9.0-py38_cu102
Proceed ([y]/n)? y

Jupyter Notebook working folder change





- Working folder change



Teachable machine

- Teachable Machine은 누구나 머신러닝 모델을 쉽고 빠르고 간단하게 만들 수 있도록 제작된 웹 기반 도구

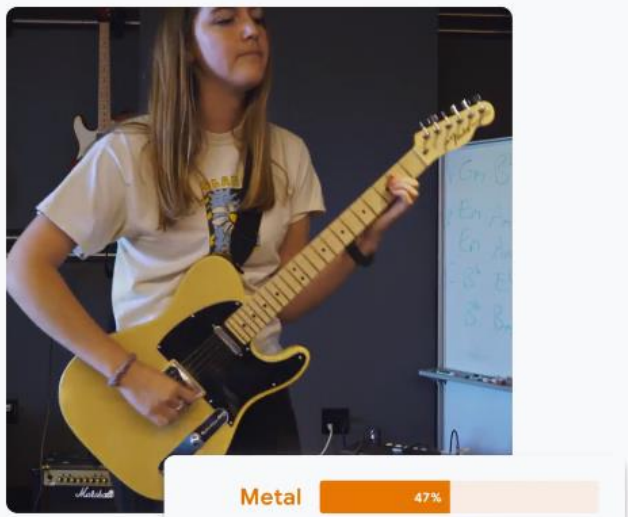
https://teachablemachine.withgoogle.com/

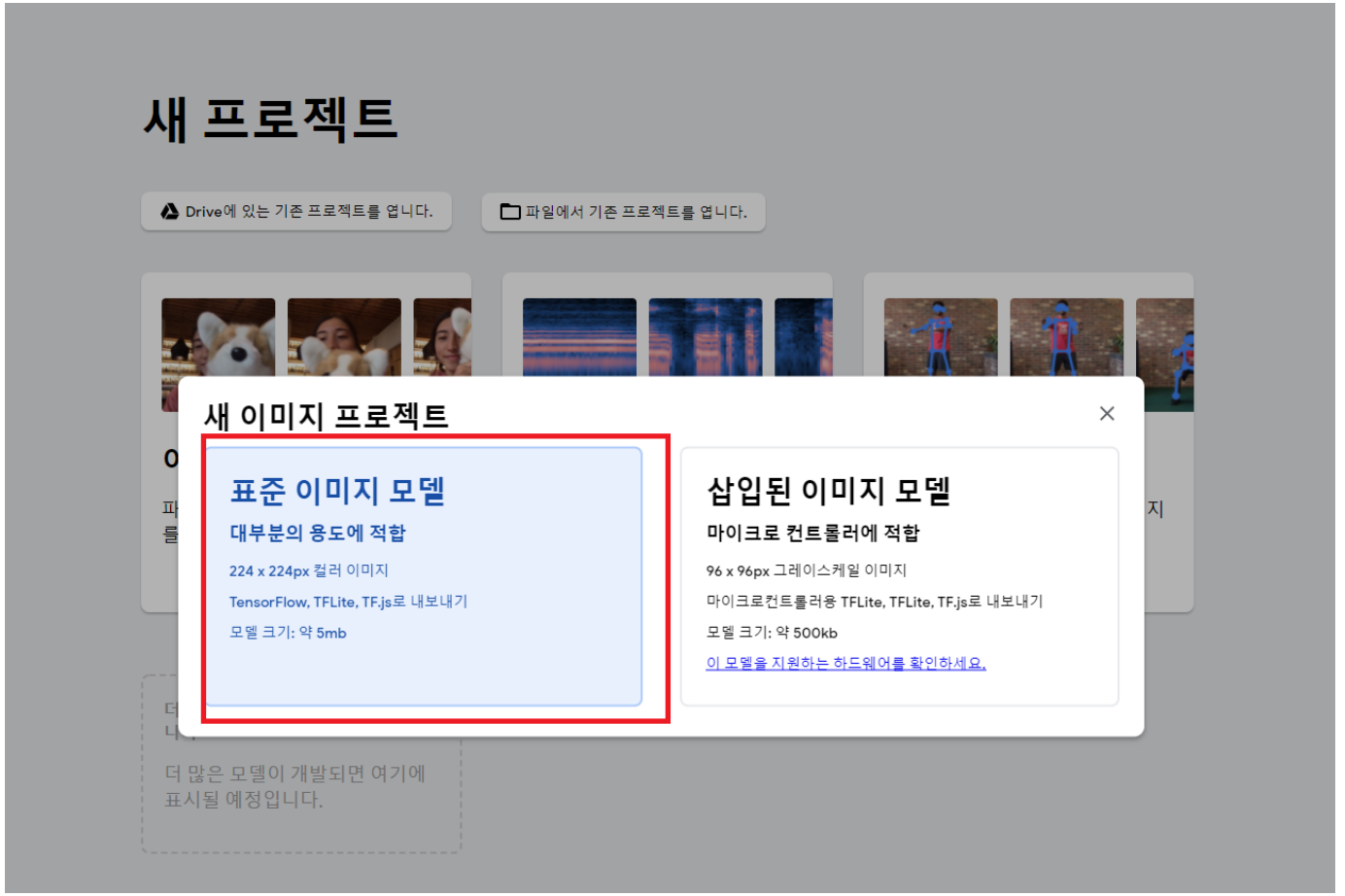
Teachable Machine

이미지, 사운드, 자세를 인식하도록 컴퓨터를 학습시키세요.

사이트, 앱 등에 사용할 수 있는 머신러닝 모델을 쉽고 빠르게 만들어 보세요. 전문지식이나 코딩 능력이 필요하지 않습니다.

시작하기

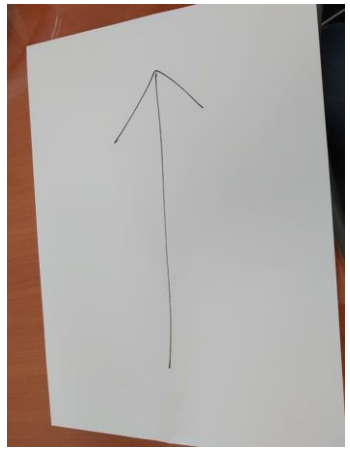



새 프로젝트
Drive에 있는 기존 프로젝트를 엽니다.
파일에서 기존 프로젝트를 엽니다.
새 이미지 프로젝트
표준 이미지 모델
대부분의 용도에 적합
224 x 224px 컬러 이미지
TensorFlow, TFLite, TF.js로 내보내기
모델 크기: 약 5mb
삽입된 이미지 모델
마이크로 컨트롤러에 적합
96 x 96px 그레이스케일 이미지
마이크로컨트롤러용 TFLite, TFLite, TF.js로 내보내기
모델 크기: 약 500kb
이 모델을 지원하는 하드웨어를 확인하세요.
더 많은 모델이 개발되면 여기에 표시될 예정입니다.

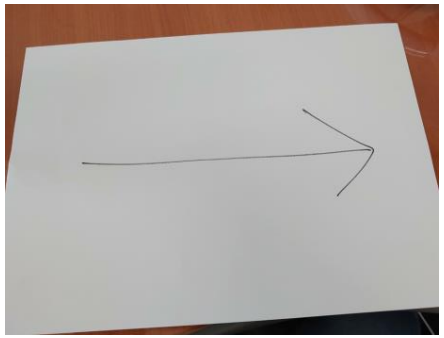

Class 1
이미지 샘플 추가:
웹캠
업로드
Class 2
이미지 샘플 추가:
웹캠
업로드
Class 3
이미지 샘플 추가:
웹캠
업로드
Class 4
이미지 샘플 추가:
웹캠
업로드
클래스 추가
학습
모델 학습시키기
고급
미리 보기
모델 내보내기
여기에서 모델을 미리 확인하려면 먼저 왼쪽에서 모델을 학습시켜야 합니다.
Class를 4개로

데이터 학습하기

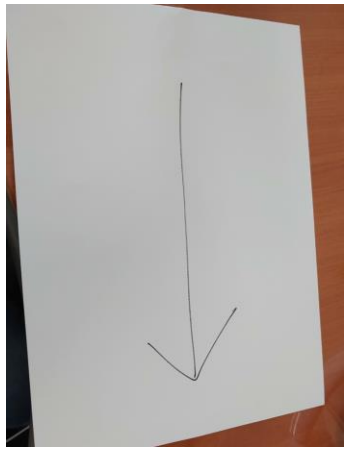
Class 1



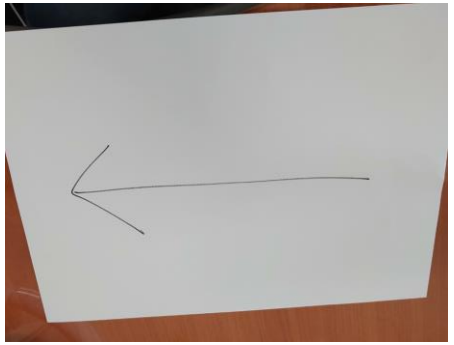
Class 2

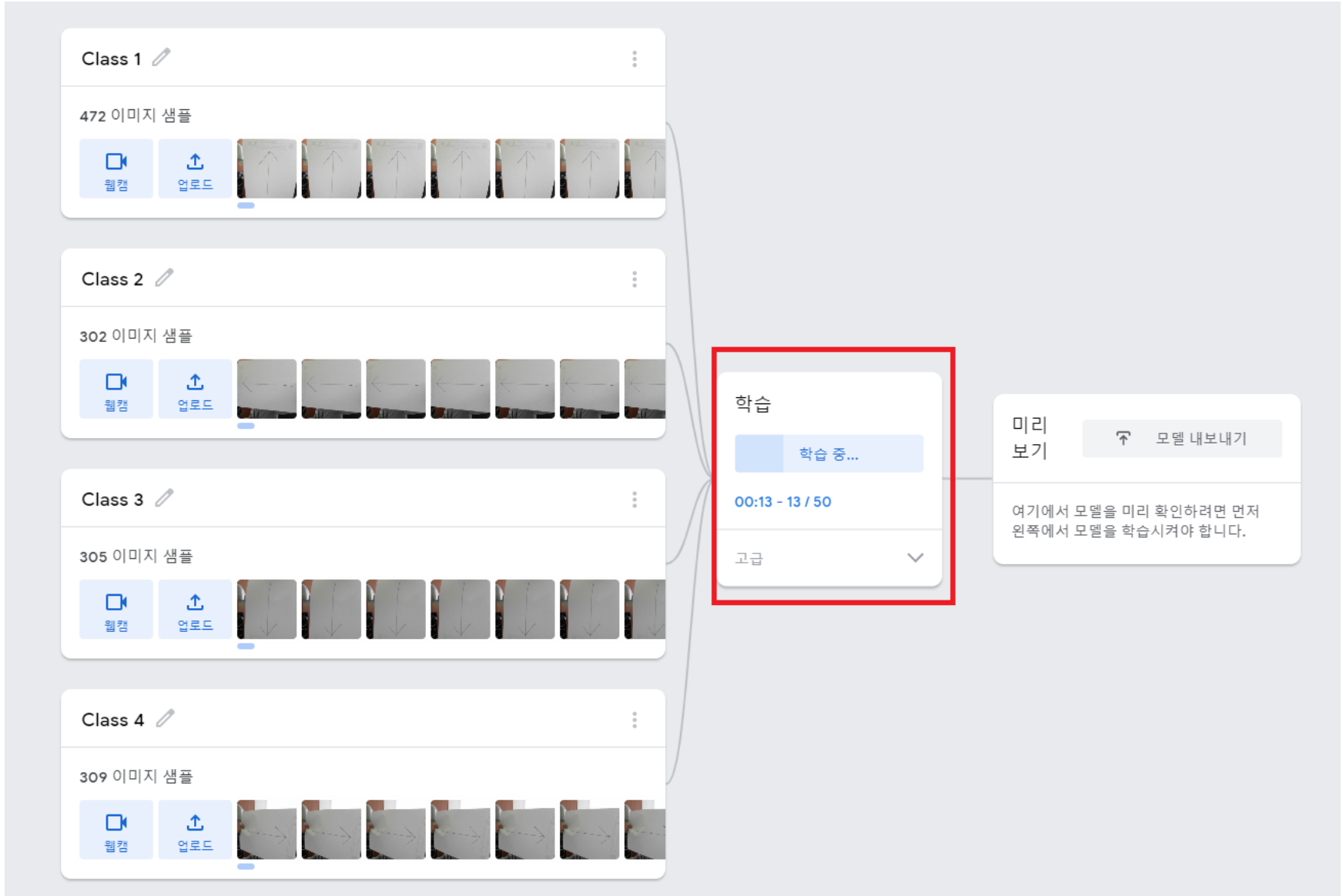


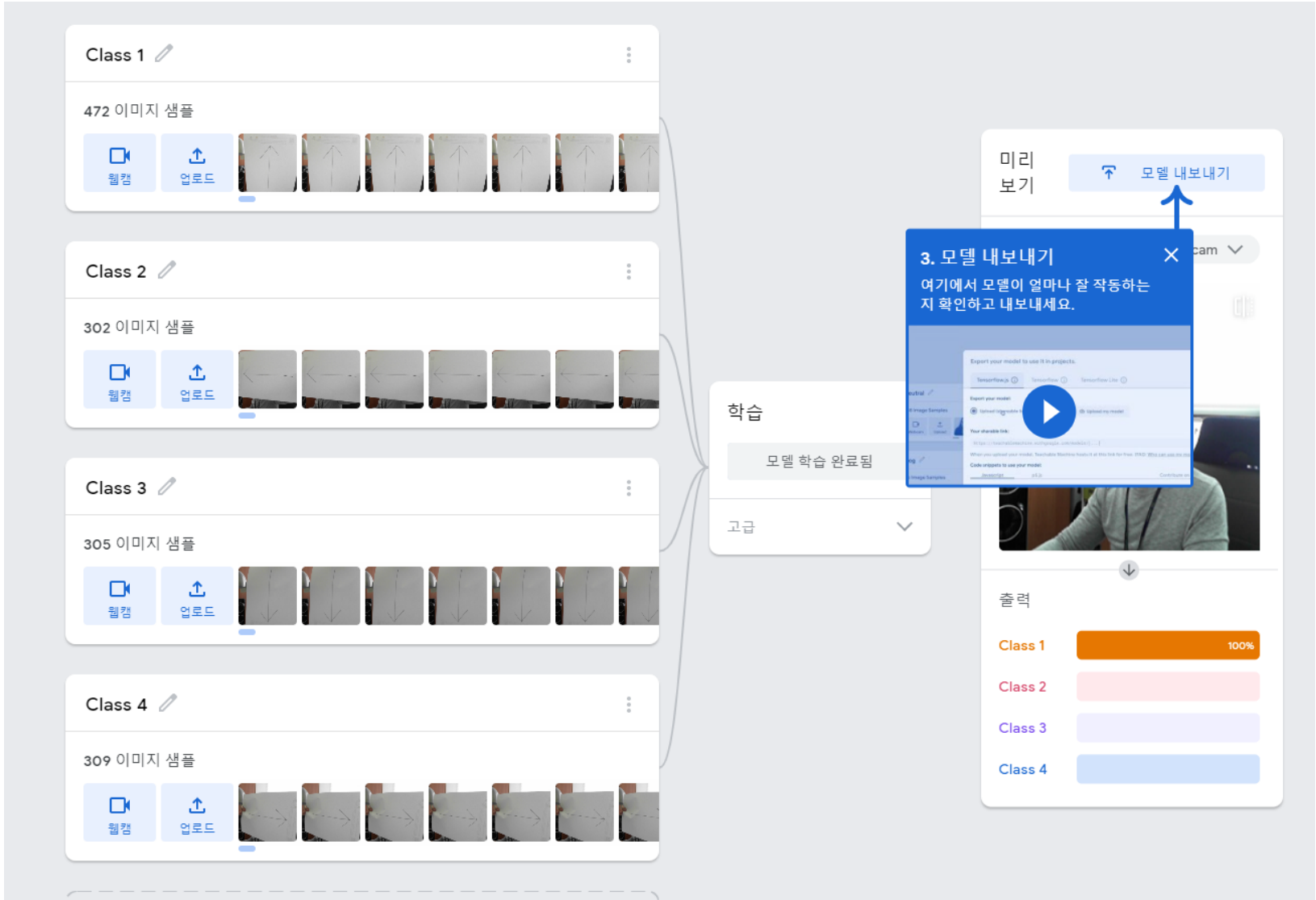
Class 3

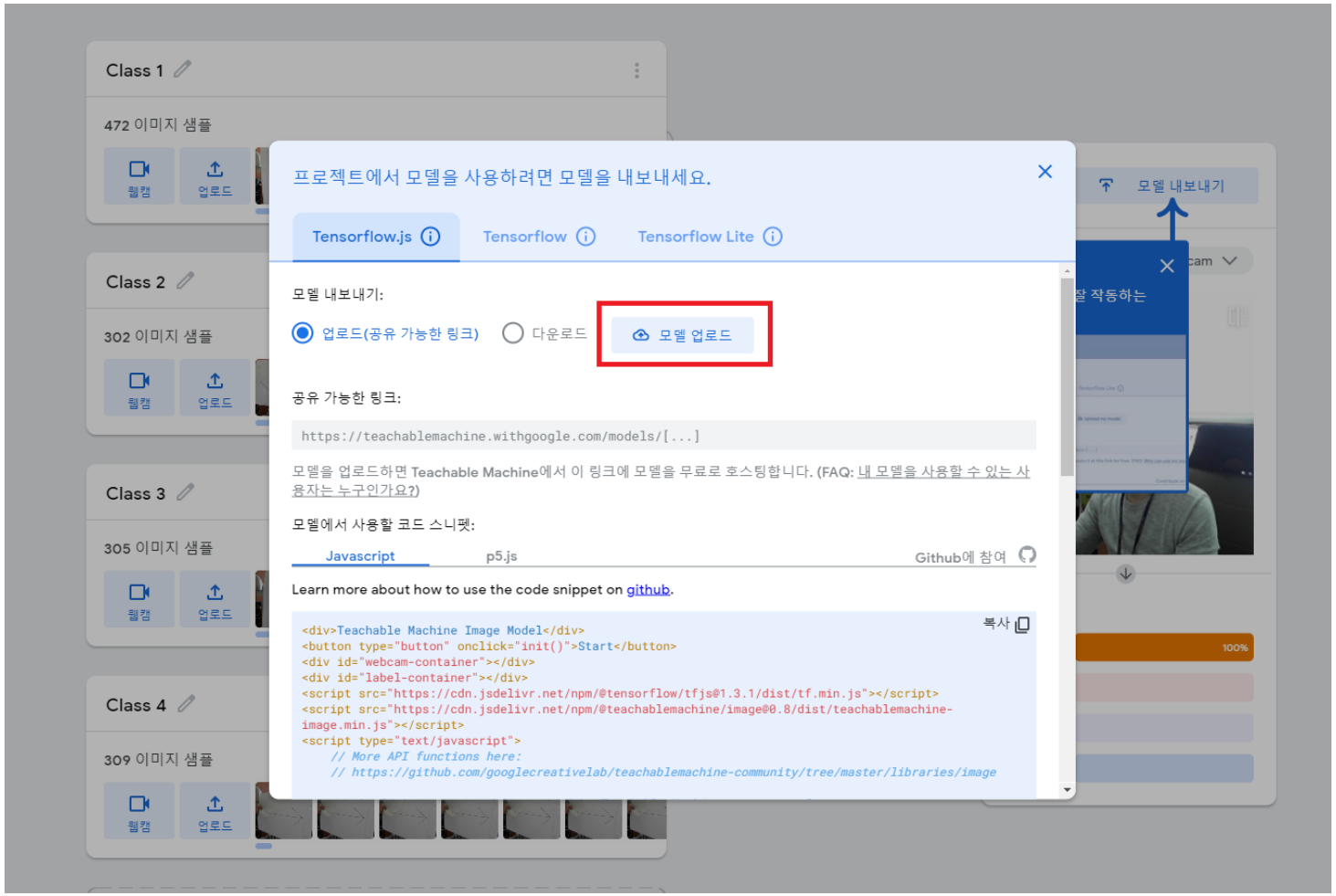


Class 4




Class 1
472 이미지 샘플
웹캠
업로드
Class 2
302 이미지 샘플
웹캠
업로드
Class 3
305 이미지 샘플
웹캠
업로드
Class 4
309 이미지 샘플
웹캠
업로드
학습
학습 중...
00:13 - 13 / 50
고급
미리 보기
모델 내보내기
여기에서 모델을 미리 확인하려면 먼저 왼쪽에서 모델을 학습시켜야 합니다.

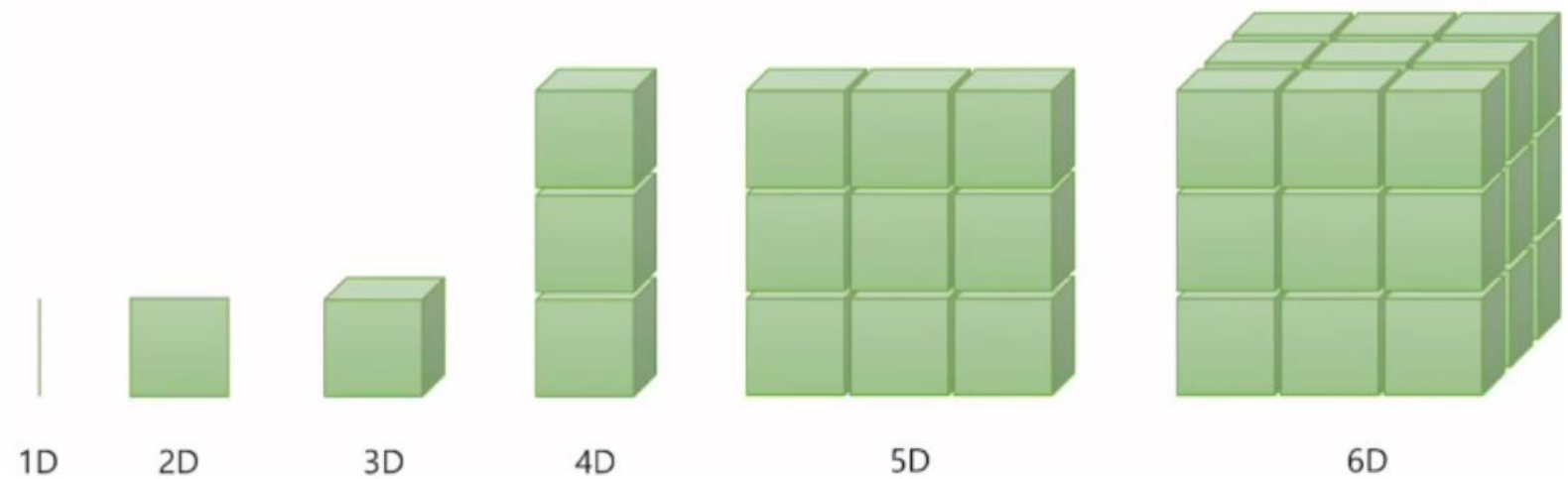

Class 1
472 이미지 샘플
웹캠
업로드
Class 2
302 이미지 샘플
웹캠
업로드
Class 3
305 이미지 샘플
웹캠
업로드
Class 4
309 이미지 샘플
웹캠
업로드
학습
모델 학습 완료됨
고급
미리 보기
모델 내보내기
3. 모델 내보내기
여기에서 모델이 얼마나 잘 작동하는 지 확인하고 내보내세요.
출력
Class 1
100%
Class 2
Class 3
Class 4


Class 1
472 이미지 샘플
웹캠
업로드
Class 2
302 이미지 샘플
웹캠
업로드
Class 3
305 이미지 샘플
웹캠
업로드
Class 4
309 이미지 샘플
웹캠
업로드
프로젝트에서 모델을 사용하려면 모델을 내보내세요.
Tensorflow.js
Tensorflow
Tensorflow Lite
모델 내보내기:
업로드(공유 가능한 링크)
다운로드
모델 업로드
공유 가능한 링크:
https://teachablemachine.withgoogle.com/models/[...]
모델을 업로드하면 Teachable Machine에서 이 링크에 모델을 무료로 호스팅합니다. (FAQ: 내 모델을 사용할 수 있는 사용자는 누구인가요?)
모델에서 사용할 코드 스니펫:
Javascript
p5.js
Github에 참여
Learn more about how to use the code snippet on github.
복사
Teachable Machine Image Model
<button type="button" onclick="init()">Start</button>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/@teachablemachine/image@0.8/dist/teachablemachine-image.min.js"></script>
<script type="text/javascript">
// More API functions here:
// https://github.com/googlecreativelab/teachablemachine-community/tree/master/libraries/image
모델 내보내기
100%

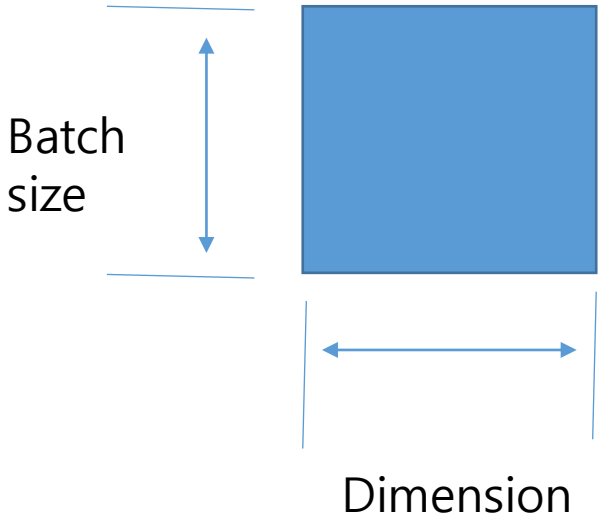
Basic concept

Tensor dimension



- 2D tensor

  |t| = (Batch size, dim)



Ex: 256개

x1 = [3, 1, 5,… 7]

……

x64 = [5, 2, 7,… 2]

……

x3000 = [2, 8, 9,… 9]

Batch size=64

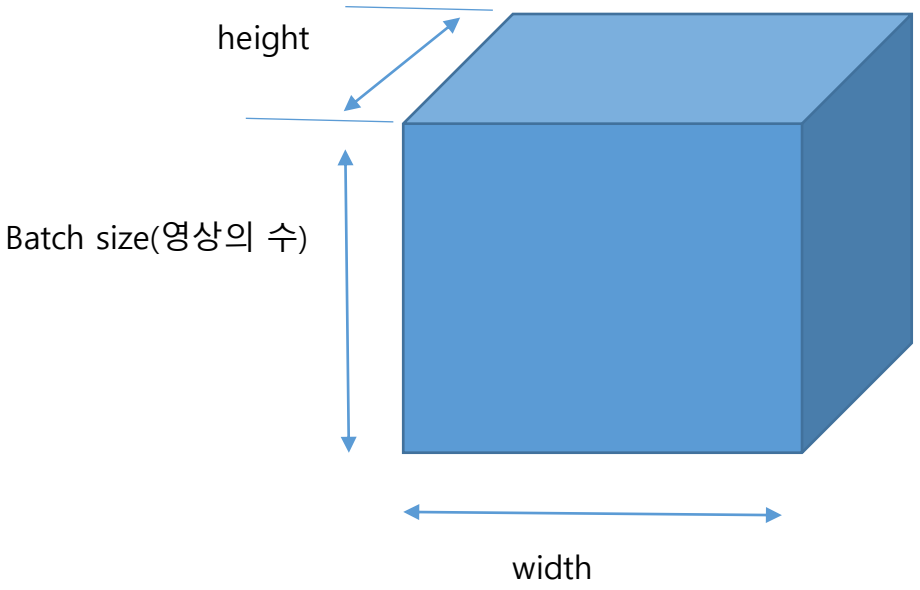
전체 훈련 데이터 = 3000x256

|t| = (64, 256)

3D tensor

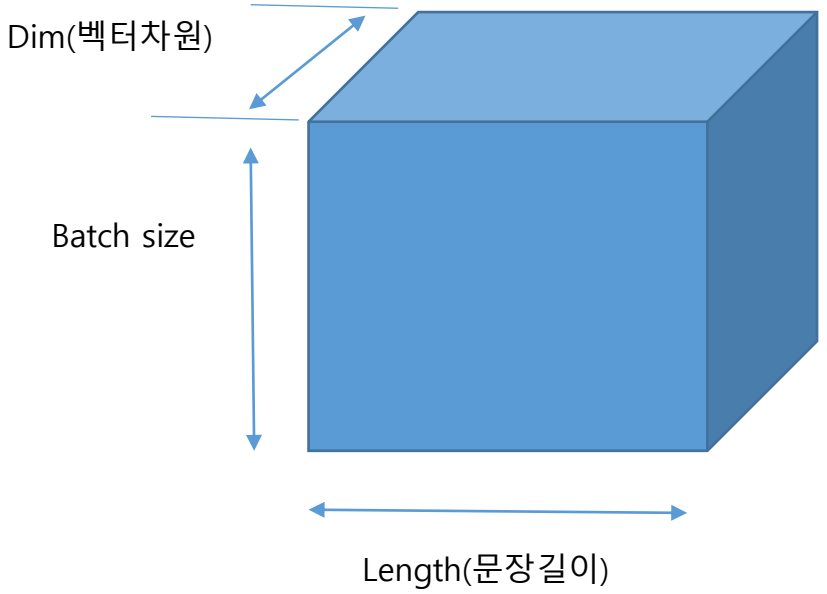
3D tensor in vision

|t| = (Batch size, width, height)



3D tensor in Natural Language Processing

|t| = (Batch size, length, dim)



- Training data

  [[나는 사과를 좋아해], [나는 바나나를 좋아해],[나는 사과를 싫어해],[나는 바나나를 싫어해]]

입력으로 사용하기 위해 단어별로 분리 |t|=[4, 3]

[['나는', '사과를', '좋아해'], ['나는', '바나나를', '좋아해'],['나는', '사과를', '싫어해'],['나는', '바나나를', '싫어해']]

각 단어를 3차원 벡터로 변환

```
'나는' = [0.1, 0.2, 0.9]
'사과를' = [0.3, 0.5, 0.1]
'바나나를' = [0.3, 0.5, 0.2]
'좋아해' = [0.7, 0.6, 0.5]
'싫어해' = [0.5, 0.6, 0.7]
```

- 훈련 데이터 재구성

```
[[[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.7, 0.6, 0.5]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.7, 0.6, 0.5]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.5, 0.6, 0.7]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.5, 0.6, 0.7]]]
```

- 훈련 데이터 |t|=[4, 3, 3]이고 batch size= 2이면

```
첫번째 배치 #1
[[[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.7, 0.6, 0.5]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.7, 0.6, 0.5]]]

두번째 배치 #2
[[[0.1, 0.2, 0.9], [0.3, 0.5, 0.1], [0.5, 0.6, 0.7]],
 [[0.1, 0.2, 0.9], [0.3, 0.5, 0.2], [0.5, 0.6, 0.7]]]
```

- 각 Batch 에서의 tensor |t|=[2,3,3]

Pytorch Basic

1D array

0번 index

-1번 index(맨뒤에서 시작하는 index)

```
import numpy as np
import torch
t = np.array([0., 1., 2., 3., 4., 5., 6.])
print(t)
```

```
print('Rank  of t: ', t.ndim)          → 차원 : 1
print('Shape of t: ', t.shape)         → 크기 :7  즉  (1x7) 벡터
```

```
Rank  of t:  1
Shape of t:  (7,)
```

[시작번호 : 끝번호]

```
print('t[0] t[1] t[-1] = ', t[0], t[1], t[-1]) # Element  t[0] t[1] t[-1] =  0.0 1.0 6.0

print('t[2:5] t[4:-1]  = ', t[2:5], t[4:-1])   # Slicing t[2:5] t[4:-1]  =  [ 2.  3.  4.] [ 4.  5.]

print('t[:2] t[3:]     = ', t[:2], t[3:])      # Slicing  t[:2] t[3:]     =  [ 0.  1.] [ 3.  4.  5.  6.]
```

[시작번호 : 끝까지]

* Slicing은 끝번호에 해당하는 것은 포함 하지 않음 !

2D array

2D Array with PyTorch

```
t = torch.FloatTensor([[1., 2., 3.],
                       [4., 5., 6.],
                       [7., 8., 9.],
                       [10., 11., 12.]
                      ])
print(t)
```

```
tensor([[ 1.,  2.,  3.],
        [ 4.,  5.,  6.],
        [ 7.,  8.,  9.],
        [10., 11., 12.]])
```

```
print(t.dim())  # rank
print(t.size()) # shape
print(t[:, 1])
print(t[:, 1].size())
print(t[:, :-1])
```

```
2
torch.Size([4, 3])
tensor([ 2.,  5.,  8., 11.])
torch.Size([4])
tensor([[ 1.,  2.],
        [ 4.,  5.],
        [ 7.,  8.],
        [10., 11.]])
```

* Slicing은 끝번호에 해당하는 것은 포함 하지 않음 !

- Broadcasting

  브로드캐스팅 기능은 서로 크기가 다른 행렬들이 사칙 연산을 수행할 수 있도록 자동으로 크기를 맞춰서 연산을 수행.

```
# Same shape
m1 = torch.FloatTensor([[3, 3]])
m2 = torch.FloatTensor([[2, 2]])
print(m1 + m2)
```

```
tensor([[5., 5.]])
```

```
# Vector + scalar
m1 = torch.FloatTensor([[1, 2]])
m2 = torch.FloatTensor([3]) # 3 -> [[3, 3]]
print(m1 + m2)
```

```
tensor([[4., 5.]])
```

```
# 2 x 1 Vector + 1 x 2 Vector
m1 = torch.FloatTensor([[1, 2]])
m2 = torch.FloatTensor([[3], [4]])
print(m1 + m2)
```

```
tensor([[4., 5.],
        [5., 6.]])
```

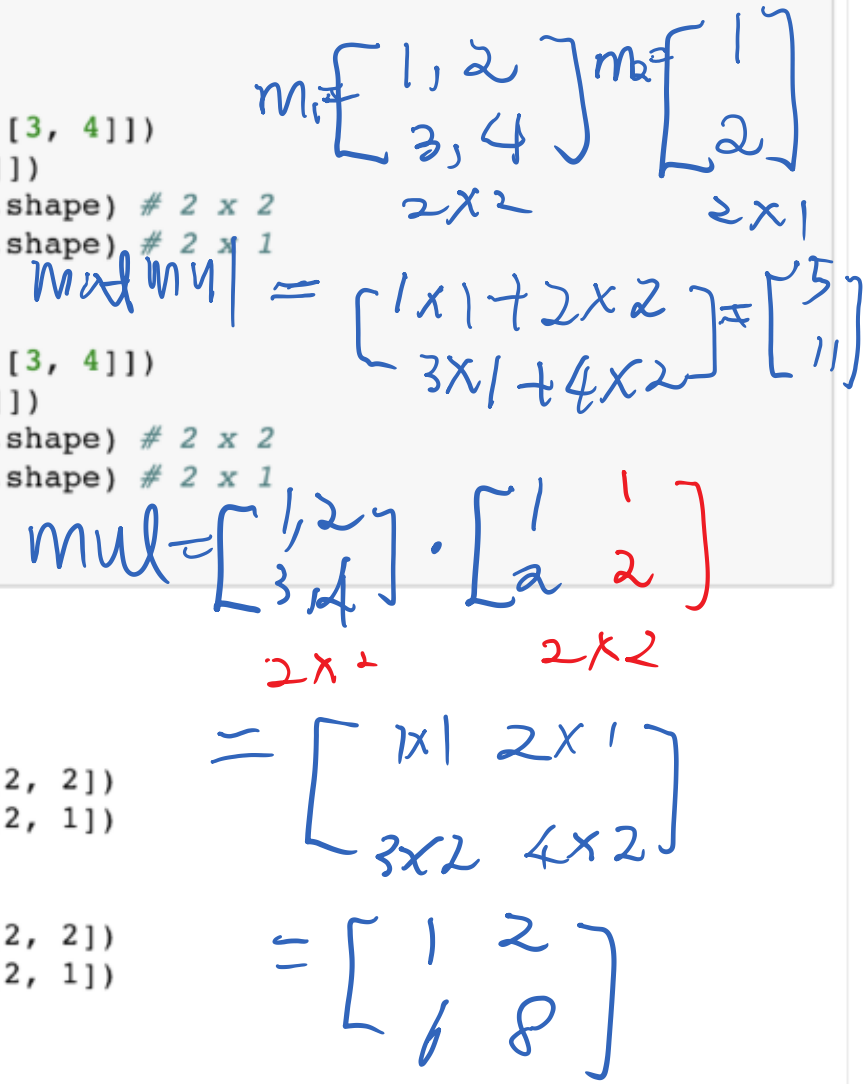
|m1|=(1,2) ➔ (2,2)
|m2|=(2,1)) ➔ (2,2

[1, 2] + [3 = [1,2] [3, 3]
4]v [1,2] + [4, 4]

- Mutliplication & Matrix Multiplication
  -Element-wise 곱셈에선 서로 다른 크기의 행렬이 브로드캐스팅 된 후에 곱셈이 수행.
  이는 동일한 크기의 행렬이 동일한 위치에 있는 원소끼리 곱하는 것과 같은 결과를 얻게 됨.

```
print()
print('-------------')
print('Mul vs Matmul')
print('-------------')
m1 = torch.FloatTensor([[1, 2], [3, 4]])
m2 = torch.FloatTensor([[1], [2]])
print('Shape of Matrix 1: ', m1.shape) # 2 x 2
print('Shape of Matrix 2: ', m2.shape) # 2 x 1
print(m1.matmul(m2)) # 2 x 1

m1 = torch.FloatTensor([[1, 2], [3, 4]])
m2 = torch.FloatTensor([[1], [2]])
print('Shape of Matrix 1: ', m1.shape) # 2 x 2
print('Shape of Matrix 2: ', m2.shape) # 2 x 1
print(m1 * m2) # 2 x 2
print(m1.mul(m2))
```

```
-------------
Mul vs Matmul
-------------
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[ 5.],
        [11.]])
Shape of Matrix 1:  torch.Size([2, 2])
Shape of Matrix 2:  torch.Size([2, 1])
tensor([[1., 2.],
        [6., 8.]])
tensor([[1., 2.],
        [6., 8.]])
```



Mean

- .mean()에서 dim을 인자로 주게 되면 어느 차원을 기준으로 평균을 구할지 설정가능 .
- 아무 것도 설정하지 않으면 전체 평균을 계산
- dim=0은 첫번째 차원 즉 행을 기준으로 평균을 계산.
- dim=0이라는 것은 첫번째 차원을 의미. 행렬에서 첫번째 차원은 '행'을 의미합니다. 그리고 인자로 dim을 준다면 해당 차원을 제거한다는 의미. 다시 말해 행렬에서 '열'만을 남기겠다는 의미
- dim=1은 두번째 차원 즉 열을 기준으로 평균을 계산
- dim=-1은 마지막 차원으로 평균을 계산하는데 위의 예제에선 마지막 차원 = 열이므로 t2.mean(dim=1)과 출력값이 동일.

```
t = torch.FloatTensor([1, 2])
print(t.mean())
```

```
tensor(1.5000)
```

```
# Can't use mean() on integers
t = torch.LongTensor([1, 2])
try:
    print(t.mean())
except Exception as exc:
    print(exc)
```

```
Can only calculate the mean of floating types. Got Long instead.
```

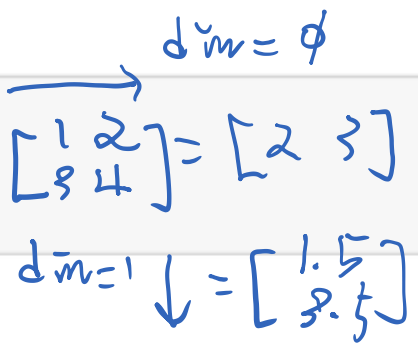
You can also use `t.mean` for higher rank tensors to get mean of all elements, or mean by particular dimension.

```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)
```

```
tensor([[1., 2.],
        [3., 4.]])
```

```
print(t.mean())
print(t.mean(dim=0))
print(t.mean(dim=1))
print(t.mean(dim=-1))
```

```
tensor(2.5000)
tensor([2., 3.])
tensor([1.5000, 3.5000])
tensor([1.5000, 3.5000])
```



Sum

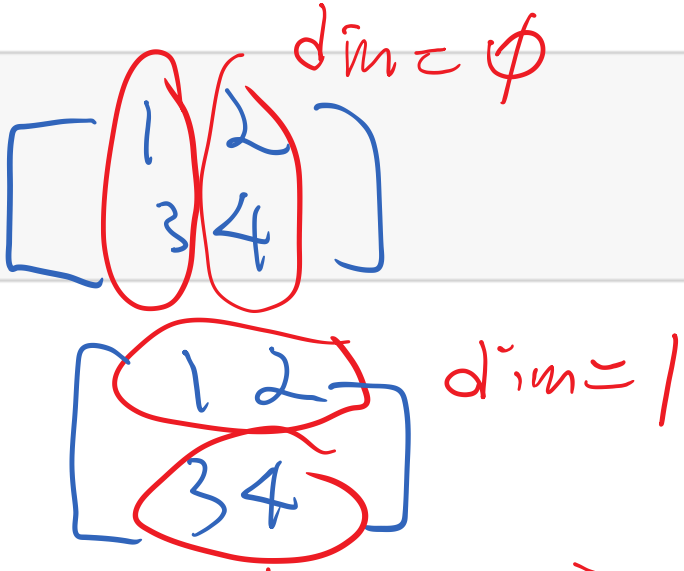
- 덧셈은 평균과 사용 방법이 완전히 동일.

```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)
```

```
tensor([[1., 2.],
        [3., 4.]])
```

```
print(t.sum())
print(t.sum(dim=0))
print(t.sum(dim=1))
print(t.sum(dim=-1))
```

```
tensor(10.)
tensor([4., 6.])
tensor([3., 7.])
tensor([3., 7.])
```



dim=-1은 마지막차원으로 이경우는 여기서 dim=1과 동일

Max & Argmax

- Max는 최대값을 반환하고 Argmax는 최대값을 가진 인덱스를 반환.
- 차원을 지정하면 두개의 데이터를 뽑을 수 있음.
- t.max(dim=0)[0]이 최대값(max)를 나타내고,
- t.max(dim=0)[1]이 최대값을 갖는 데이터 인덱스를 나타냄.

```
t = torch.FloatTensor([[1, 2], [3, 4]])
print(t)
```

```
tensor([[1., 2.],
        [3., 4.]])
```

첫번째 열에서 0번 index 1, 1번 index 3
두번째 열에서 0번 index 2, 1번 index 4

The `max` operator returns one value if it is called without an argument.

```
print(t.max()) # Returns one value: max
```

원소중 최대값 4 retrurn

```
tensor(4.)
```

The `max` operator returns 2 values when called with dimension specified. The first value is the maximum value, and the second value is the argmax: the index of the element with maximum value.

Dim=0, 첫번째 차원제거

```
print(t.max(dim=0)) # Returns two values: max and argmax
print('Max: ', t.max(dim=0)[0])
print('Argmax: ', t.max(dim=0)[1])
```

[0], max 값만
[1], argmax 값만 return

```
(tensor([3., 4.]), tensor([1, 1]))
Max:  tensor([3., 4.])
Argmax:  tensor([1, 1])
```

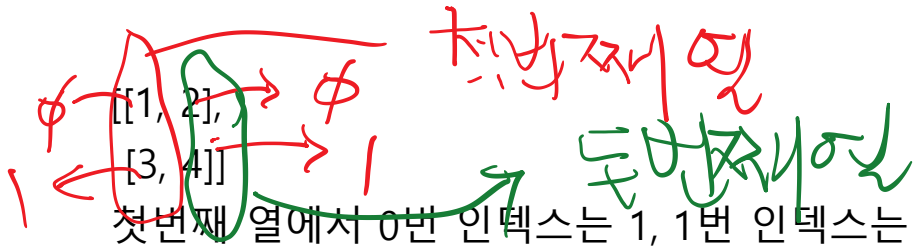
3, 4의 인덱스 return

```
print(t.max(dim=1))
print(t.max(dim=-1))
```

```
(tensor([2., 4.]), tensor([1, 1]))
(tensor([2., 4.]), tensor([1, 1]))
```

- print(t.max(dim=0)) # Returns two values: max and argmax
- (tensor([3., 4.]), tensor([1, 1]))
- 행의 차원을 제거한다는 의미이므로 (1, 2) 텐서를 만듬. 결과는 [3, 4].

- Argmax → [1, 1]. max에 dim 인자를 주면 argmax도 함께 리턴
- 첫번째 열에서 3의 인덱스는 1, 두번째 열에서 4의 인덱스는 1. 그러므로 [1, 1]이 리턴

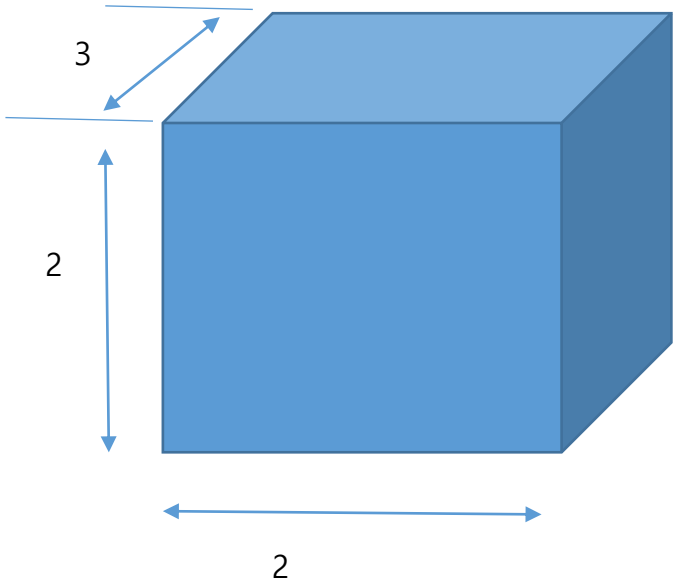


[[1, 2],
[3, 4]]

첫번째 열에서 0번 인덱스는 1, 1번 인덱스는 3.
두번째 열에서 0번 인덱스는 2, 1번 인덱스는 4.
다시 말해 3과 4의 인덱스는 [1, 1].

VIEW

- 원소의 수를 유지하면서 텐서의 크기 변경.



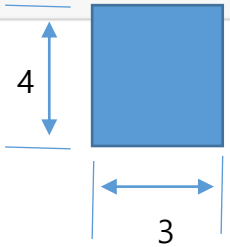

```
t = np.array([[[0, 1, 2],
               [3, 4, 5]],

              [[6, 7, 8],
               [9, 10, 11]]])
ft = torch.FloatTensor(t)
print(ft.shape)
```

```
torch.Size([2, 2, 3])
```

```
print(ft.view([-1, 3]))
print(ft.view([-1, 3]).shape)
```

-1:pytorch가 알아서.. 3: 차원의 길이 3으로

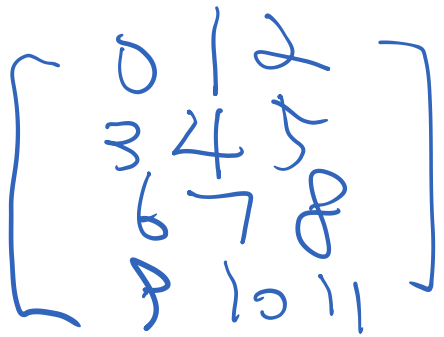
```
tensor([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.],
        [ 6.,  7.,  8.],
        [ 9., 10., 11.]])
torch.Size([4, 3])
```




```
print(ft.view([-1, 1, 3]))
print(ft.view([-1, 1, 3]).shape)
```

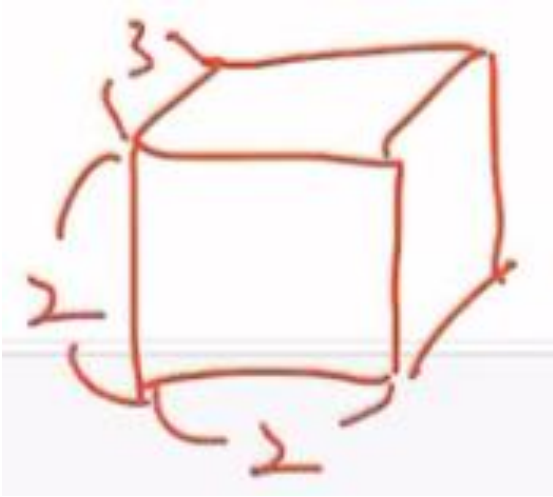
(? X 1 X 3)으로 변경

```
tensor([[[ 0.,  1.,  2.]],

        [[ 3.,  4.,  5.]],

        [[ 6.,  7.,  8.]],

        [[ 9., 10., 11.]]])
torch.Size([4, 1, 3])
```



- 파이토치 텐서의 뷰(View)는 넘파이에서의 Reshape와 같이 텐서의 크기(Shape)를 변경해주는 역할을 함.

```
t = np.array([[[0, 1, 2], [3, 4, 5]],
              [[6, 7, 8], [9, 10, 11]]])
ft = torch.FloatTensor(t)

print(ft.shape)
torch.Size([2, 2, 3])
```

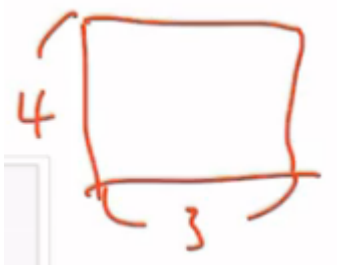


- 현재 위 텐서의 크기는 (2, 2, 3).

- view([-1, 3])→. -1은 첫번째 차원은 사용자가 잘 모르겠으니 파이토치에 맡기겠다는 의미, 3은 두번째 차원의 길이는 3
- → 현재 3차원 텐서를 2차원 텐서로 변경하되 (?, 3)의 크기로 변경하라는 의미. 결과적으로 (4, 3)의 크기를 가지는 텐서를 얻음.
- 내부적으로 크기 변환은 다음과 같이 이루어 짐. (2, 2, 3) -> (2 × 2, 3) -> (4, 3)

```
print(ft.view([-1, 3])) # ft라는 텐서를 (?, 3)의 크기로 변경
print(ft.view([-1, 3]).shape)
```

```
tensor([[ 0.,  1.,  2.],
        [ 3.,  4.,  5.],
        [ 6.,  7.,  8.],
        [ 9., 10., 11.]])
torch.Size([4, 3])
```



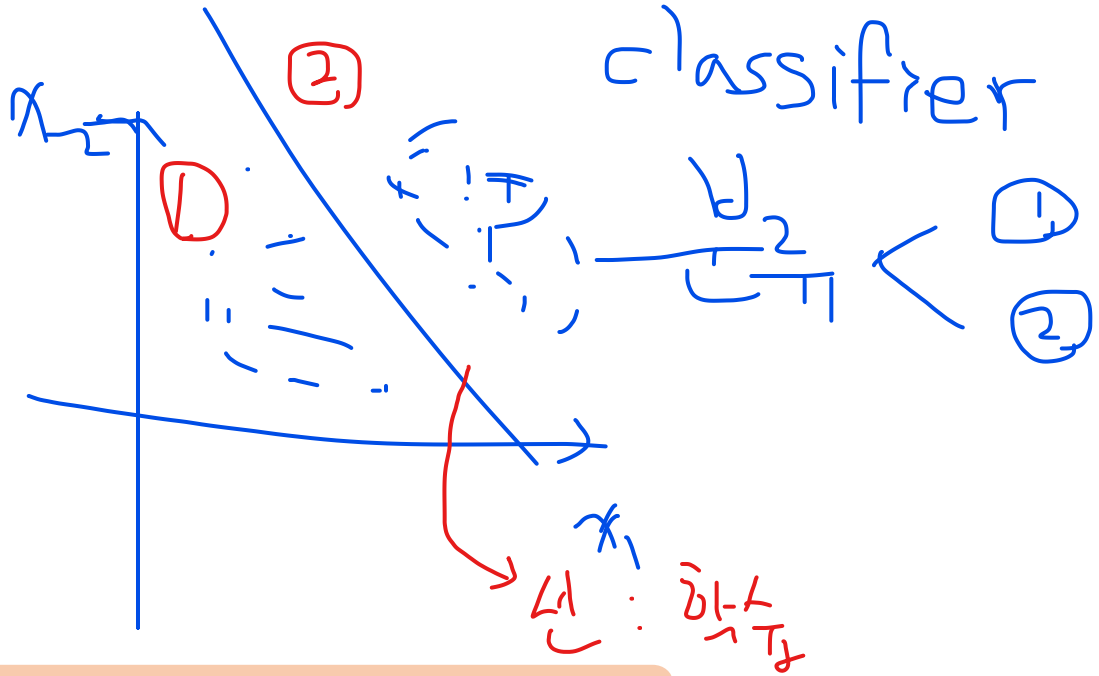
Linear regression

본 자료는 김성훈교수의 모두의 딥러닝 시즌1, 이승재씨의 ML/DL for Everyone season2 자료를 활용하여 제작함.

https://github.com/deeplearningzerotoall/PyTorch

Supervised Learning

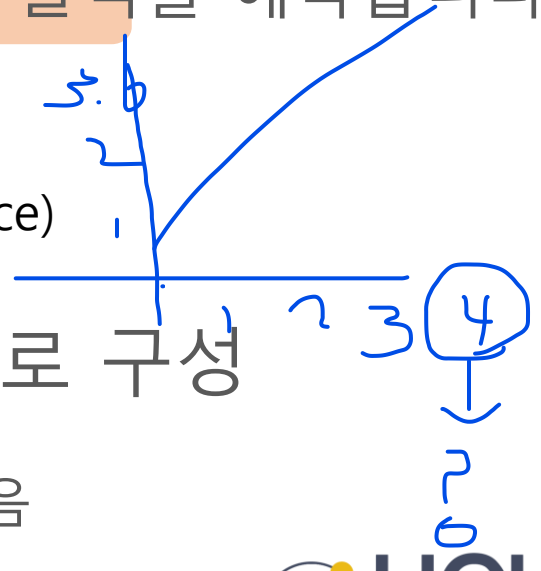
지도 학습 supervised learning



훈련 (training, learning)

입력과 출력의 샘플 데이터가 있을 때 새로운 입력에 대한 출력을 예측합니다 .

예측 (predict, Inference)



지도학습은 분류(classification) 와 회귀(regression) 로 구성

각 모델은 선형모델(Linear-Model)과
비-선형 모델(Non-Linear Model)로 구분할 수 있음

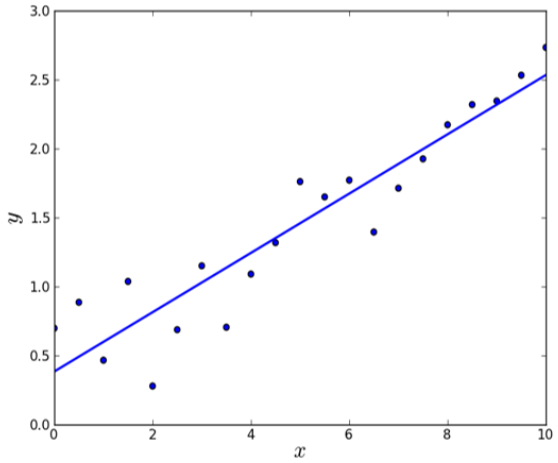
회귀(regression)

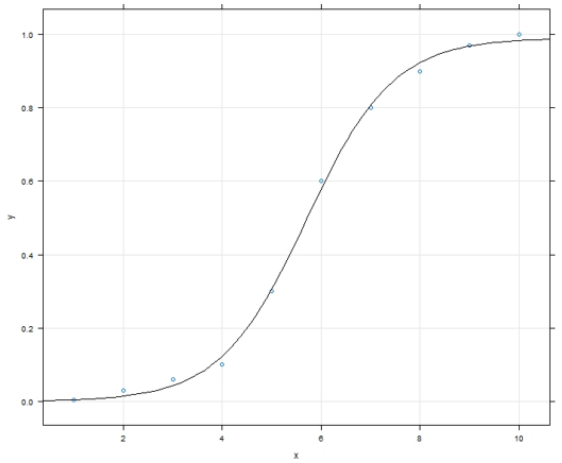
- 회귀는 연속적인 숫자 (실수) 를 예측
  - ➢ 회귀의 예
    - ✓ 교육 , 나이 , 주거지를 바탕으로 연간 소득 예측
    - ✓ 전년도 수확량 , 날씨 , 고용자수로 올해 수확량 예측

- 회귀는 예측 값에 미묘한 차이가 크게 중요하지 않는것이 큰 특징입니다.
  - ✓ 연봉 예측의 경우 40,000,001원 이나 39,999,999 원이 문제가 되지 않음 .

선형 모델(Linear Model)

- 추정할 대상의 파라미터의 모형이 선형식으로 표현이 가능한 회귀 모델(Linearizable Regression Model)이면 선형 모델이라 부름

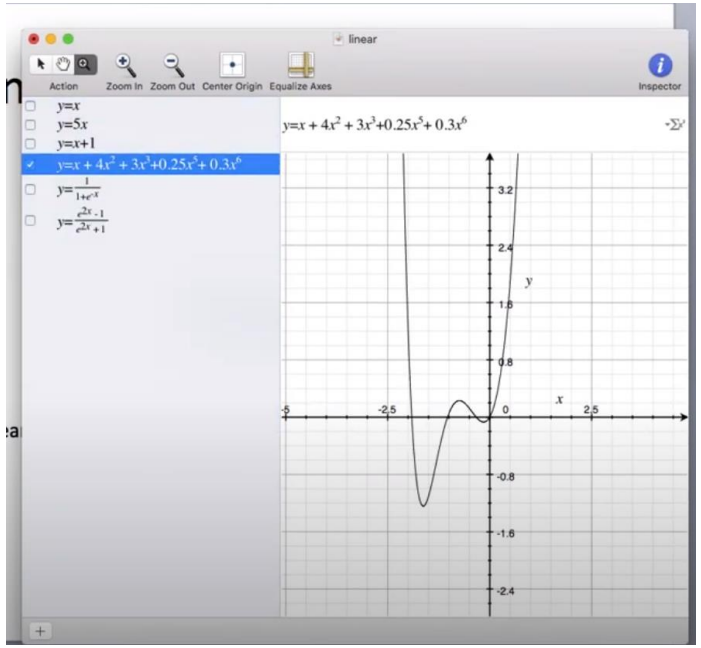


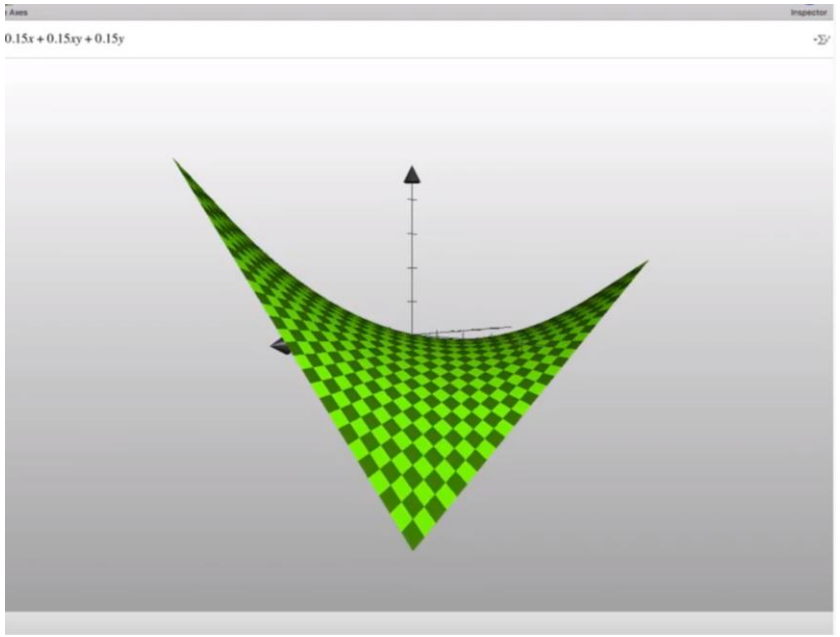


- $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3 \Rightarrow \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$
- $y = \beta_0 x^{\beta_1} \Rightarrow log(y) = log(\beta_0 x^{\beta_1}) \Rightarrow log\beta_0 + \beta_1 \log(x) \Rightarrow y^* = \beta_0^* + \beta_1 x^*$
- $y = \frac{e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}}{1 + e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3}} \Rightarrow \frac{y}{1-y} = e^{\beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3} \Rightarrow \log\left(\frac{y}{1-y}\right) = y^* = \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3$

- 독립변수(x)와 종속변수(y)의 관계의 일차식 표현이다 -> 잘못된 말
- 선형모델은 유연성이 떨어진다는 단점 존재
- EX) Linear Discreminant Classifier, Naïve Bayes, Logistic Regression, Perceptron, SVM(with linear kernel)

선형 모델 예





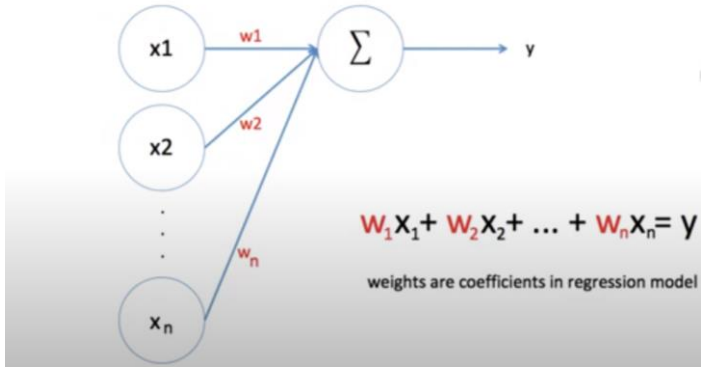
비선형 모델(Non-Linear Model)

- 독립변수(x)와 종속변수(y)의 관계의 선형 결합식으로 표현할 수 없는 모델을 비선형 모델이라 함
- 회귀 모델의 목적이 해석이 아니라 예측을 한다면 비선형 모델을 선택
- 복잡한 패턴을 갖는 데이터에 대해서 모델링이 가능(XOR Problem)
  - (Ex: Deep Learning, MLP(Mulitple Layer Perceptron), Quadratic Discriminant Classifier, Decision Trees, Random Forest, KNN, SVM(with Gaussian/RBF Kernel)

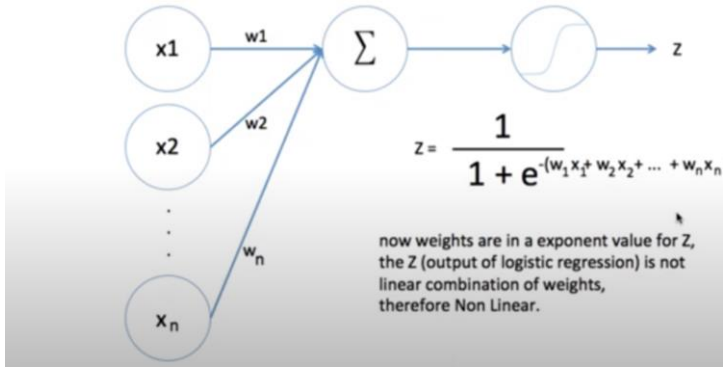
$$y = \frac{\beta_1 x}{\beta_2 + x}$$

- 참고 동영상: https://youtu.be/umiqnfQxlac

Linear regression



logistic regression is non linear model



Linear regression case 1

What would be the grade if I study 4 hours?

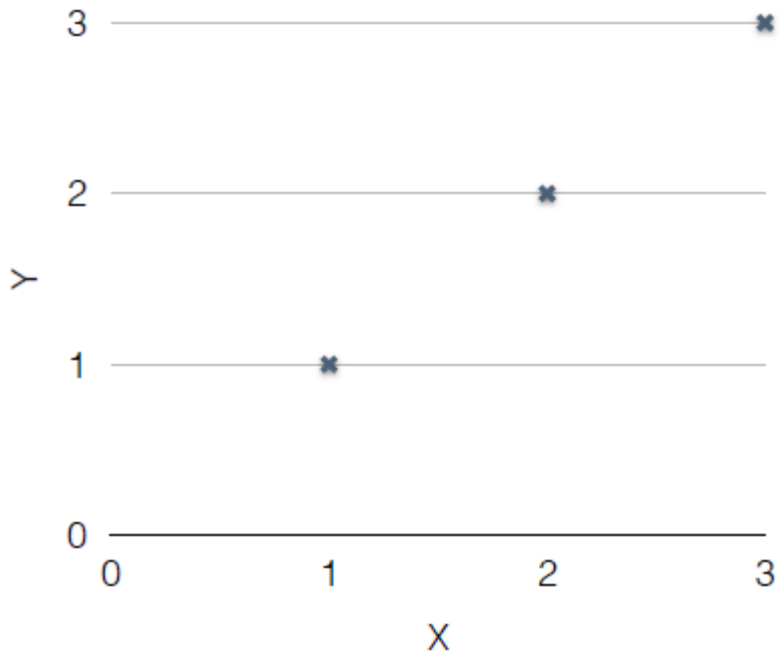
Hours (x)	Points (y)
1	2
2	4
3	6
4	?

Training dataset

Test dataset

Regression

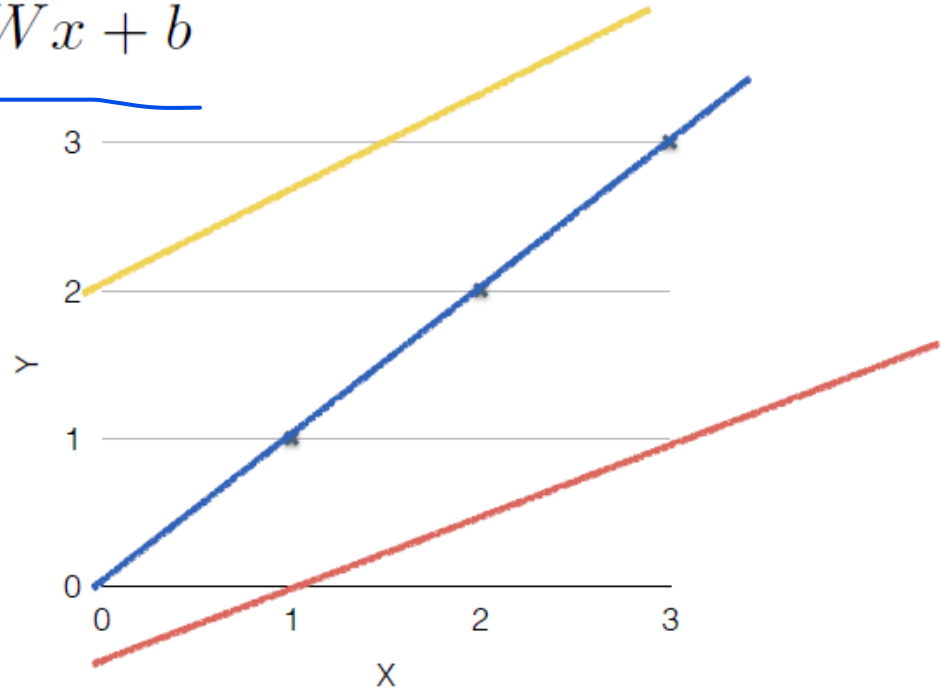
X	Y
1	1
2	2
3	3



Linear(Hypothesis)

- Which hypothesis is better?

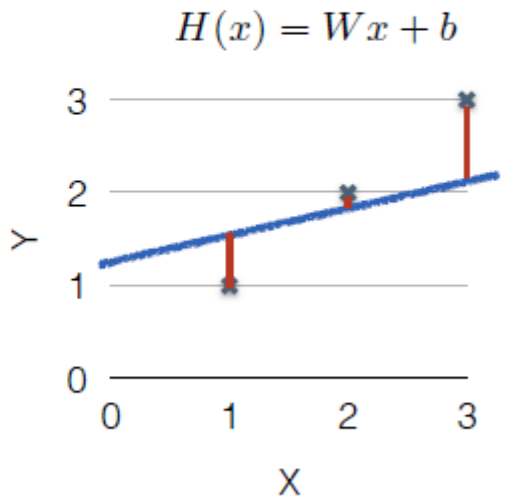
$$H(x) = Wx + b$$



Cost function

How fit the line to our (training) data

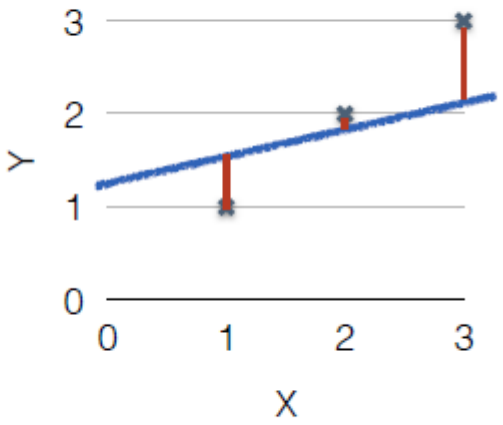
$$H(x) - y$$



$$\frac{(H(x^{(1)}) - y^{(1)})^2 + (H(x^{(2)}) - y^{(2)})^2 + (H(x^{(3)}) - y^{(3)})^2}{3}$$

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

$H(x) = Wx + b$



Cost function

$$cost = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x^{(i)}) - y^{(i)})^2$$

Goal(Minimize cost)

$$\underset{W,b}{\text{minimize}}\, cost(W, b)$$

How to minimize the cost?

- Hypothesis and Cost

$$H(x) = Wx + b$$

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} (H(x^{(i)}) - y^{(i)})^2$$

Simplified cost

$$H(x) = Wx$$

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

What cost(w) looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$

x	Y
1	1
2	2
3	3

- W=1, cost(W)=0

$$\frac{1}{3}((1*1-1)^2 + (1*2-2)^2 + (1*3-3)^2)$$

- W=0, cost(W)=4.67

$$\frac{1}{3}((0*1-1)^2 + (0*2-2)^2 + (0*3-3)^2)$$

- W=2, cost(W)=?

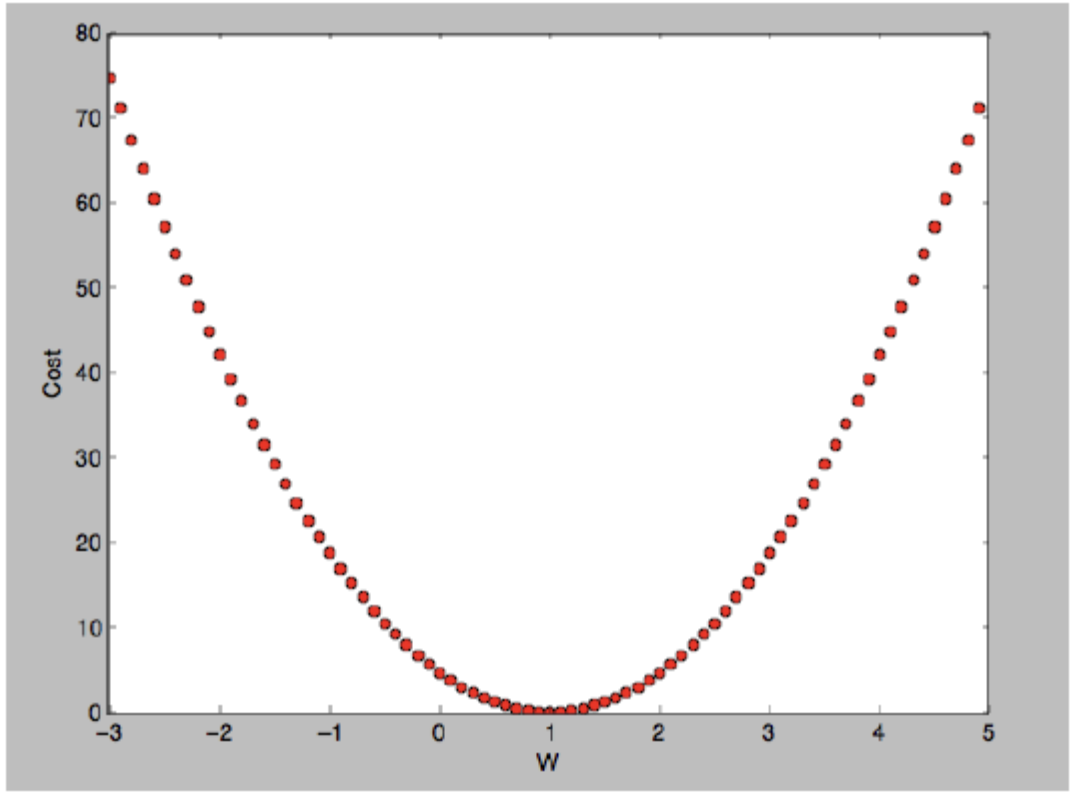
W=1, cost(W)=0

W=0, cost(W)=4.67

W=2, cost(W)=4.67

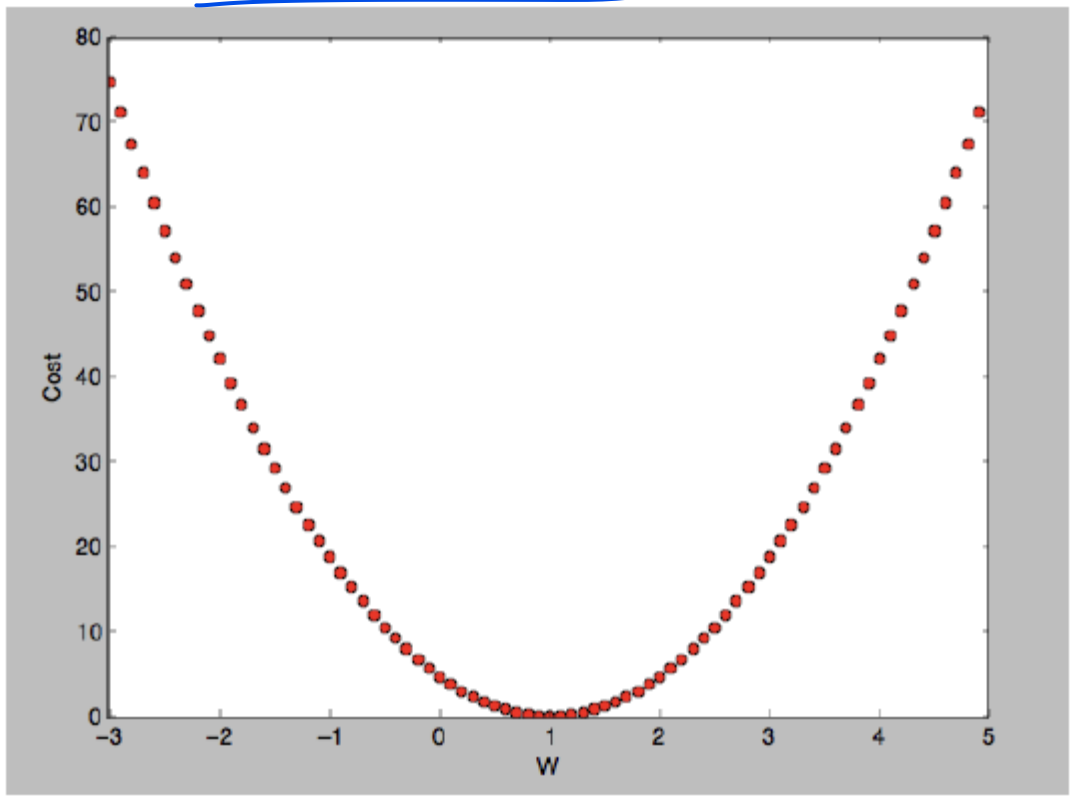
What cost(w) looks like?

$$cost(W) = \frac{1}{m}\sum_{i=1}(Wx^{(i)} - y^{(i)})^2$$



How to minimize cost?

$$cost(W) = \frac{1}{m}\sum_{i=1}(Wx^{(i)} - y^{(i)})^2$$



Gradient descent algorithm

- Minimize cost function
- Gradient descent is used many minimization problems
- For a given cost function, *cost (W, b)*, it will find *W, b* to minimize cost
- It can be applied to more general function: *cost (w1, w2, …)*

Gradient descent algorithm

- How it works?
- How would you find the lowest point?

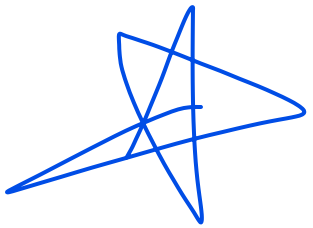
How it works?

- Start with initial guesses
- Start at 0,0 (or any other value)
- Keeping changing W and b a little bit to try and reduce cost(W, b)
- Each time you change the parameters, you select the gradient which reduces cost(W, b) the most possible
- Repeat
- Do so until you converge to a local minimum
- Has an interesting property
- Where you start can determine which minimum you end up

http://www.holehouse.org/mlclass/01_02_Introduction_regression_analysis_and_gr.html

Formal definition

$$cost(W) = \frac{1}{2m}\sum_{i=1}^{m}(Wx^{(i)} - y^{(i)})^2$$



$$W := W - \alpha\frac{\partial}{\partial W}cost(W)$$

α : 학습률

$$W := W - \alpha \frac{\partial}{\partial W} \frac{1}{2m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})^2$$

$$W := W - \alpha \frac{1}{2m} \sum_{i=1}^{m} 2(Wx^{(i)} - y^{(i)})x^{(i)}$$

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (Wx^{(i)} - y^{(i)})x^{(i)}$$

Gradient descent algorithm

$$W := W - \alpha \frac{1}{m} \sum_{i=1}^{m} (W x^{(i)} - y^{(i)}) x^{(i)}$$

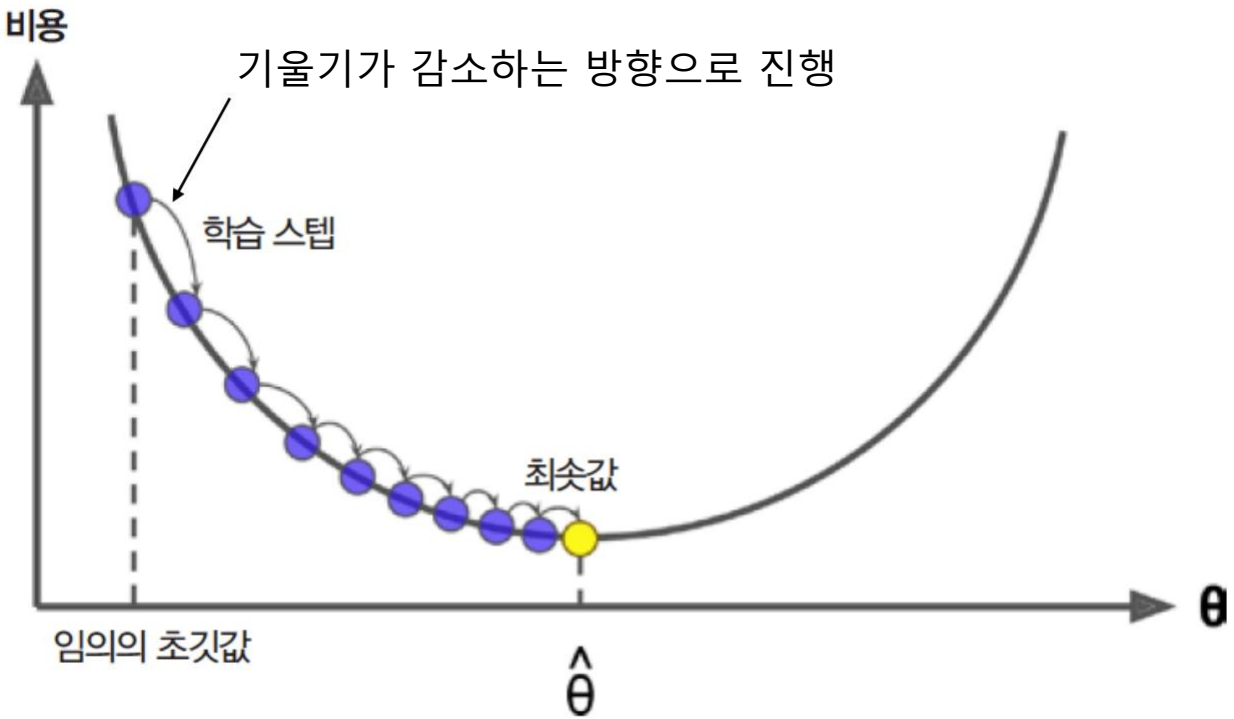
경사 하강법

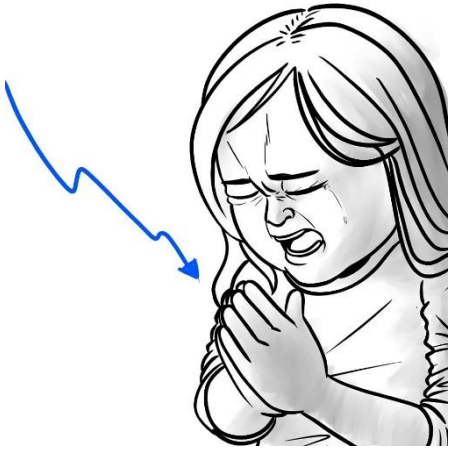
- 기계학습 방법은 경사하강법(Gradient Descent)를 이용합니다.
- Gradient Descent(GD): 모델 파라미터를 조금씩 수정하면서 비용 함수의 최소값을찾는 방법(≒학습속도)입니다 .
- 경사하강법은 미분을 다수의 Epoch을 통하여 진행됩니다.

학습률(학습속도, Learning Rate)

$$\theta^{\{next\ step\}} = \theta - \eta\nabla_\theta$$

그래디언트 : 세타에 대한 비용함수의 미분

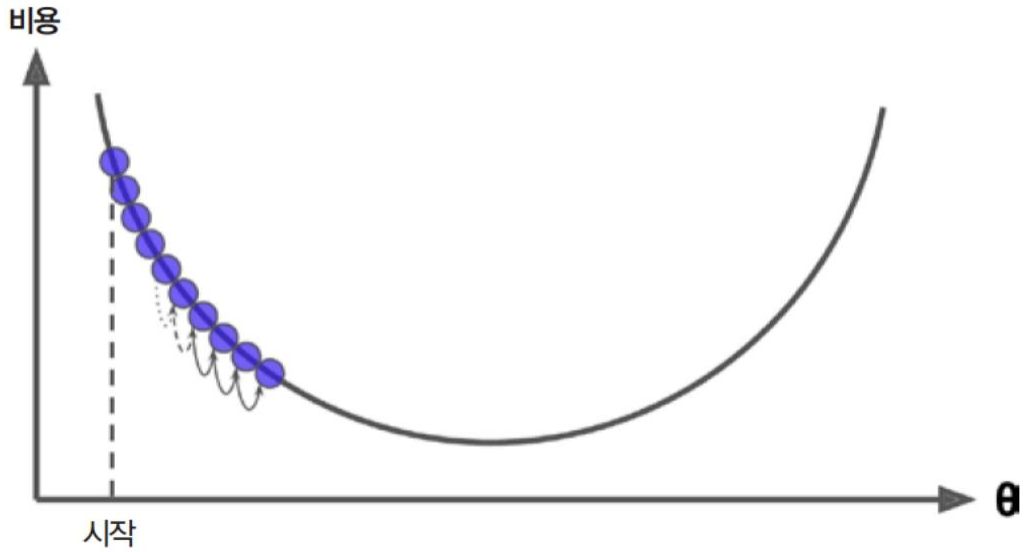




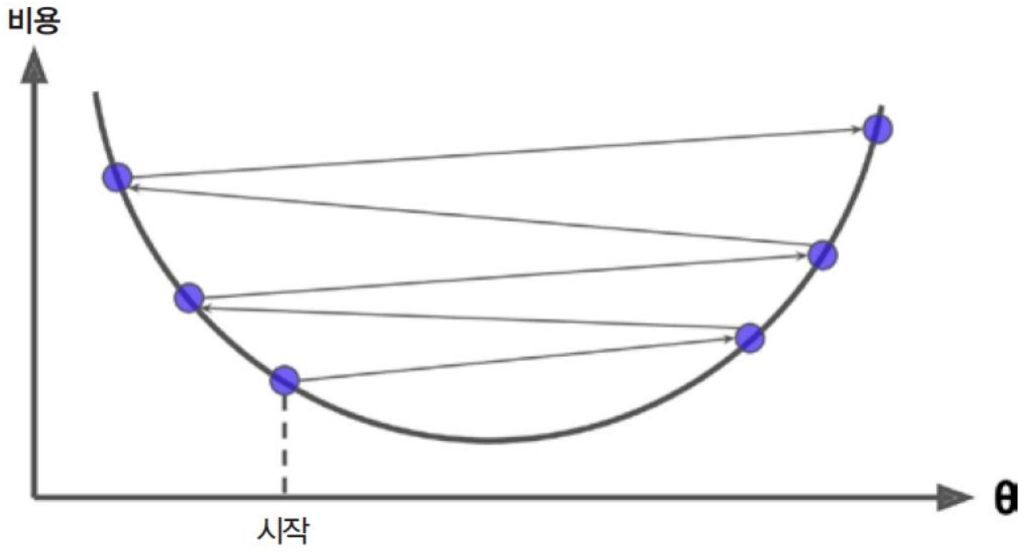
HCI Lab
Inha University
Human-Computer Interaction Lab

학습률

- 학습률은 그래디언트 적용량을 조정하는 하이퍼파라미터입니다.
  - 하이퍼 파라미터는 모델링할 때 사용자가 직접 설정해주는 값
  - 파라미터는 모델 내부에서 데이터로부터 결정되는 결정되는 변수입니다.



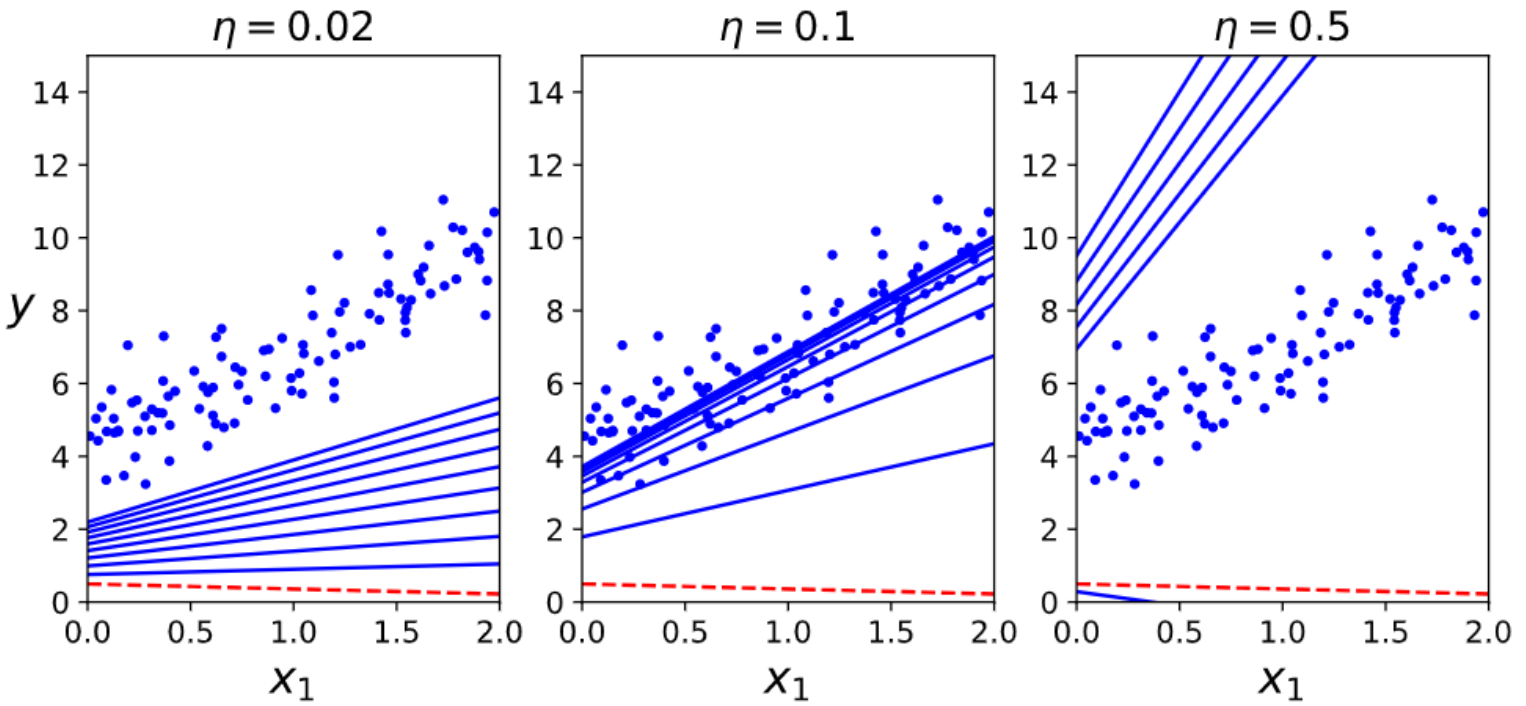
학습률이 너무 작을 때



학습률이 너무 클 때

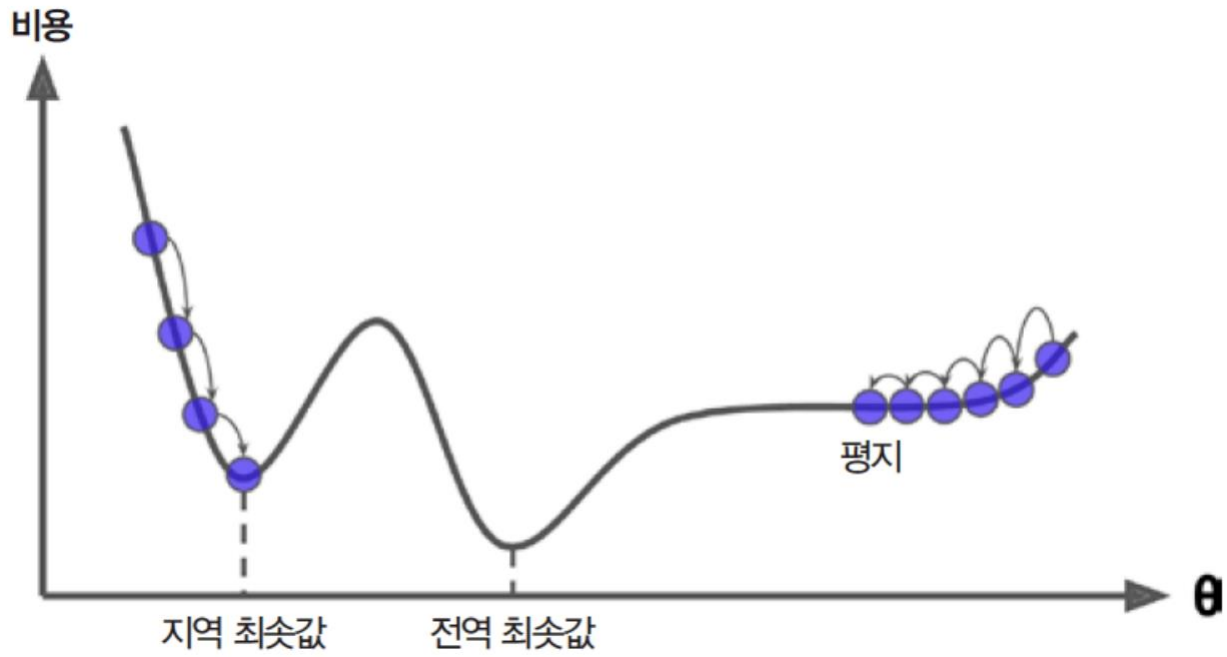
학습률에 따른 곡선

```
X_b = np.c_[np.ones((100, 1)), X]  # 모든 샘플에 x0 = 1을 추가합니다.

eta = 0.1
n_iterations = 1000
m = 100
theta = np.random.randn(2,1)

for iteration in range(n_iterations):
    gradients = 2/m * X_b.T.dot(X_b.dot(theta) - y)
    theta = theta - eta * gradients
```



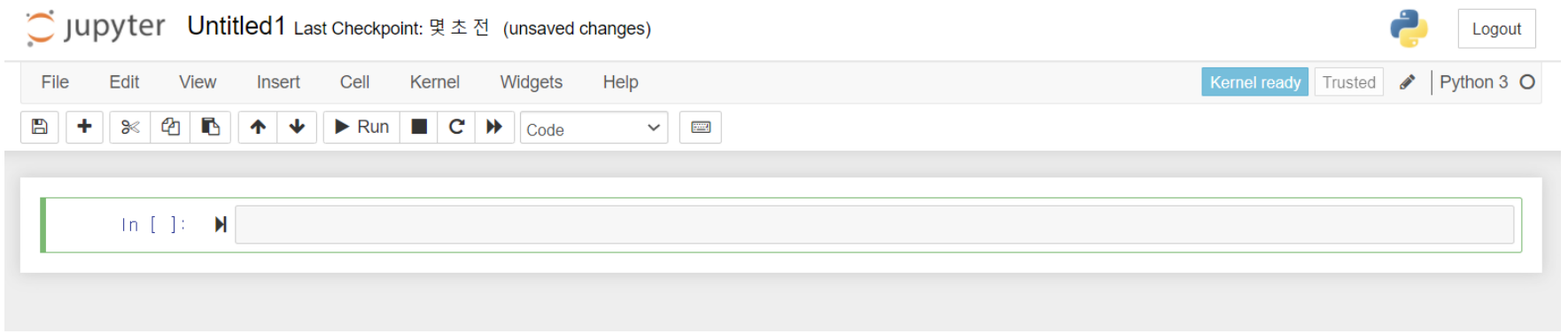
지역(Local) 최솟값, 전역(Global) 최솟값

- 평균제곱오차(MSE)는 볼록 함수이므로 지역 최솟값이 없고 학습 반복 횟수에 따라서 기울기가 변합니다 .



Lab 1. Linear regression

- Open Jupyter



- import torch
- import torch.nn as nn
- import torch.nn.functional as F
- import torch.optim as optim

- # For reproducibility
- torch.manual_seed(1) → Random value reneration

Imports

In [1]:
```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

In [2]:
```
# For reproducibility
torch.manual_seed(1)
```

Data definition

- 입력 x_train
- 출력 y_train

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])
```

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[1], [2], [3]])
```

```
print(x_train)
print(x_train.shape)
```

```
tensor([[1.],
        [2.],
        [3.]])
torch.Size([3, 1])
```

```
print(y_train)
print(y_train.shape)
```

$$X_{\text{train}} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \qquad Y_{\text{train}} = \begin{pmatrix} 2 \\ 4 \\ 6 \end{pmatrix}$$

Hours (x)	Points (y)
1	2
2	4
3	6

- Hypothesis
- W와 b 0 으로 초기화
  - 항상 출력 을 예측
- requires_grad=True
  - 학습할 것이라고 명시

$$y = Wx + b$$

Weight Bias

```
W = torch.zeros(1, requires_grad=True)
print(W)
```

```
tensor([0.], requires_grad=True)
```

```
b = torch.zeros(1, requires_grad=True)
print(b)
```

```
tensor([0.], requires_grad=True)
```

$$H(x) = Wx + b$$

```
hypothesis = x_train * W + b
print(hypothesis)
```

```
tensor([[0.],
        [0.],
        [0.]], grad_fn=<AddBackward0>)
```

Compute loss

Mean Squared Error (MSE)

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} \left(H(x^{(i)}) - y^{(i)} \right)^2$$

Mean Prediction Target

$$cost(W, b) = \frac{1}{m} \sum_{i=1}^{m} \left(H(x^{(i)}) - y^{(i)} \right)^2$$

```
print(hypothesis)
```

```
tensor([[0.],
        [0.],
        [0.]], grad_fn=<AddBackward0>)
```

```
print(y_train)
```

```
tensor([[1.],
        [2.],
        [3.]])
```

```
print(hypothesis - y_train)
```

```
tensor([[-1.],
        [-2.],
        [-3.]], grad_fn=<SubBackward0>)
```

```
print((hypothesis - y_train) ** 2)
```

```
tensor([[1.],
        [4.],
        [9.]], grad_fn=<PowBackward0>)
```

```
cost = torch.mean((hypothesis - y_train) ** 2)
print(cost)
```

```
tensor(4.6667, grad_fn=<MeanBackward1>)
```

$$\frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)})-y^{(i)}\right)^2$$

평균계산

Gradient descent

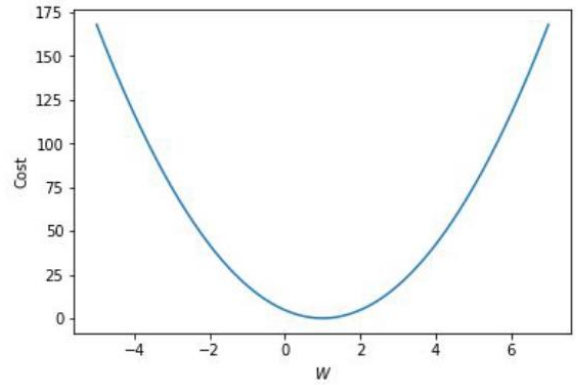
$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}\left(W x^{(i)} - y^{(i)}\right)^2$$

$$\nabla W = \frac{\partial\, cost}{\partial\, W} = \frac{2}{m}\sum_{i=1}^{m}\left(W x^{(i)} - y^{(i)}\right)x^{(i)}$$

$$W := W - \alpha \nabla W$$

Learning rate

Gradient



$$\frac{\partial cost}{\partial W} = \nabla W$$

Code

$$\nabla W = \frac{\partial cost}{\partial W} = \frac{2}{m}\sum_{i=1}^{m}\left(W x^{(i)} - y^{(i)}\right)x^{(i)}$$

$$W := W - \alpha \nabla W$$

```
gradient = 2 * torch.mean((W * x_train - y_train) * x_train)
lr = 0.1
W -= lr * gradient
```

Gradient descent with torch.optim

- torch.optim 라이브러리 사용
  - [W, b] 는 학습할 tensor들
  - lr=0.01 은 learning rate

- zero_grad() 로 gradient 초기화
- backward()로 gradient 계산
- step() 으로 gradient 개선

```
optimizer = optim.SGD([W, b], lr=0.01)
```

```
optimizer.zero_grad()
cost.backward()
optimizer.step()
```

```
print(W)
print(b)
```

```
tensor([0.0933], requires_grad=True)
tensor([0.0400], requires_grad=True)
```

Full training code

- 1회 정의
  - 데이터 정의
  - Hypothesis 초기화
  - Optimizer 정의

- 반복
  - Hypothesis 예측
  - Cost 계산
  - Optimizer 로 학습

```
x_train = torch.FloatTensor([[1], [2], [3]])
y_train = torch.FloatTensor([[2], [4], [6]])

W = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)

optimizer = optim.SGD([W, b], lr=0.01)

nb_epochs = 1000
for epoch in range(1, nb_epochs + 1):
    hypothesis = x_train * W + b
    cost = torch.mean((hypothesis - y_train) ** 2)

    optimizer.zero_grad()
    cost.backward()
    optimizer.step()
```

- Output

```
Epoch    0/1000 W: 0.093, b: 0.040 Cost: 4.666667
Epoch  100/1000 W: 0.873, b: 0.289 Cost: 0.012043
Epoch  200/1000 W: 0.900, b: 0.227 Cost: 0.007442
Epoch  300/1000 W: 0.921, b: 0.179 Cost: 0.004598
Epoch  400/1000 W: 0.938, b: 0.140 Cost: 0.002842
Epoch  500/1000 W: 0.951, b: 0.110 Cost: 0.001756
Epoch  600/1000 W: 0.962, b: 0.087 Cost: 0.001085
Epoch  700/1000 W: 0.970, b: 0.068 Cost: 0.000670
Epoch  800/1000 W: 0.976, b: 0.054 Cost: 0.000414
Epoch  900/1000 W: 0.981, b: 0.042 Cost: 0.000256
Epoch 1000/1000 W: 0.985, b: 0.033 Cost: 0.000158
```

Multivariable linear regression

Predicting exam score:

regression using three inputs (x1, x2, x3)

multi-variable/feature

x_1 (quiz 1)	x_2 (quiz 2)	x_3 (midterm 1)	Y (final)
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Hypothesis

$$H(x) = Wx + b$$

Hypothesis

$$H(x) = Wx + b$$

$$H(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

Cost function

$$H(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

$$cost(W, b) = \frac{1}{m} \sum_{I=1}^{m} (H(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}) - y^{(i)})^2$$

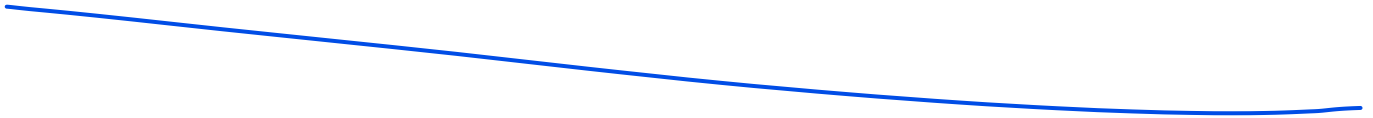
Multi-variable

$$H(x_1, x_2, x_3) = w_1 x_1 + w_2 x_2 + w_3 x_3 + b$$

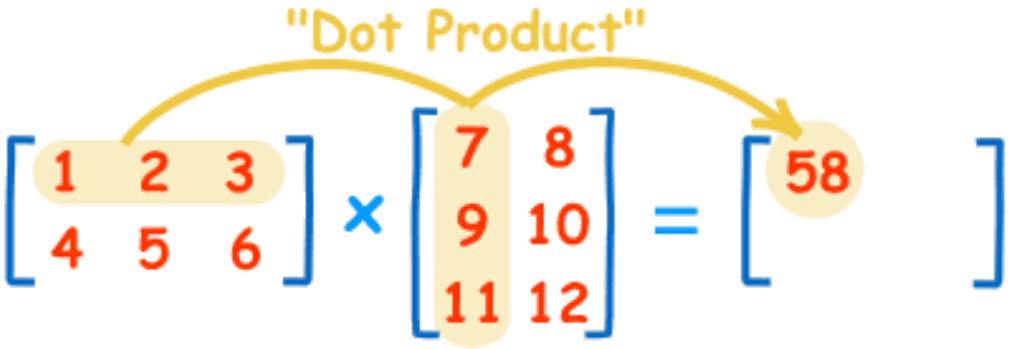
$$H(x_1, x_2, x_3, ..., x_n) = w_1 x_1 + w_2 x_2 + w_3 x_3 + ... + w_n x_n + b$$

Matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$



Matrix multiplication



https://www.mathsisfun.com/algebra/matrix-multiplying.html

Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

```
# H(x) 계산
hypothesis = x_train.matmul(W)     # or .mm or @
```

Hypothesis using matrix

$$H(x_1, x_2, x_3) = x_1 w_1 + x_2 w_2 + x_3 w_3$$

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

https://www.symbolab.com/solver/matrix-calculator

Hypothesis using matrix

$$H(x_1, x_2, x_3) = x_1w_1 + x_2w_2 + x_3w_3$$

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Many x instances

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Test Scores for General Psychology

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \end{pmatrix}$$

http://college.cengage.com/mathematics/brase/understandable_statistics/7e/students/datasets/mlr/frames/frame.htr

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

Hypothesis using matrix

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Hypothesis using matrix

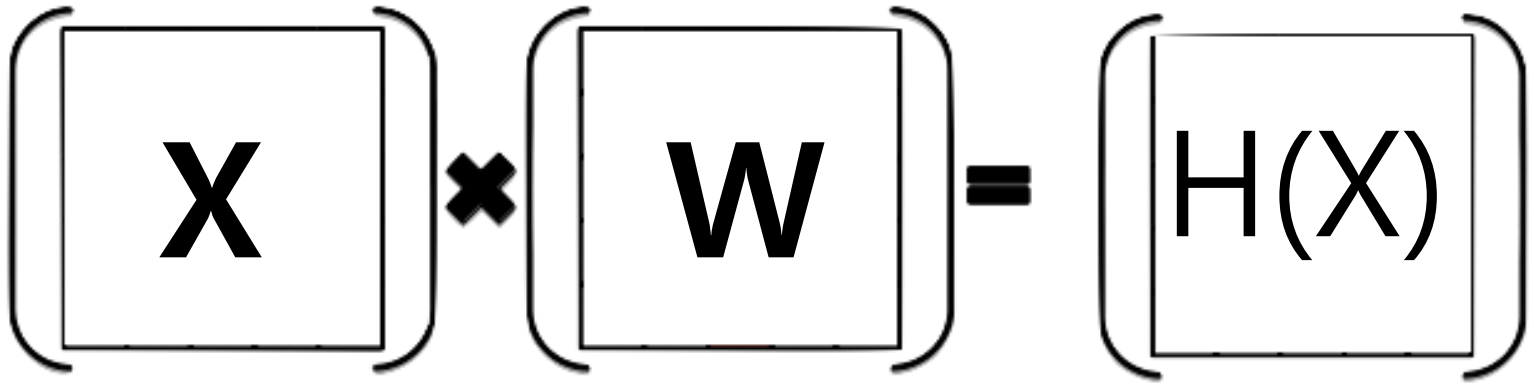
$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

`[5, 3]` `[3, 1]` `[5, 1]`

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Hypothesis using matrix



[5, 3] [?, ?] [5, 1]

$$H(X) = XW$$

Hypothesis using matrix

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

`[n, 3]` `[3, 1]` `[n, 1]`

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Hypothesis using matrix (n output)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot ? = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

`[n, 3]` `[?, ?]` `[n, 2]`

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Hypothesis using matrix (n output)

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

`[n, 3]` `[3, 2]` `[n, 2]`

$$H(X) = XW$$

https://www.symbolab.com/solver/matrix-calculator

Cost function : MSE

- Single variable linear regression 과 동일

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}\left(H(x^{(i)}) - y^{(i)}\right)^2$$

Mean Prediction Target

```
cost = torch.mean((hypothesis - y_train) ** 2)
```

Gradient descent with torch.optim

$$\nabla W = \frac{\partial cost}{\partial W} = \frac{2}{m}\sum_{i=1}^{m}\left(W x^{(i)} - y^{(i)}\right)x^{(i)}$$

$$W := W - \alpha \nabla W$$

```
# optimizer 설정
optimizer = optim.SGD([W, b], lr=1e-5)

# optimizer 사용법
optimizer.zero_grad()
cost.backward()
optimizer.step()
```

Full code

```
# 데이터
x_train = torch.FloatTensor([[73, 80, 75],
                             [93, 88, 93],
                             [89, 91, 90],
                             [96, 98, 100],
                             [73, 66, 70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])

# 모델 초기화
W = torch.zeros((3, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# optimizer 설정
optimizer = optim.SGD([W, b], lr=1e-5)
```

```
nb_epochs = 20
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x_train.matmul(W) + b # or .mm or @

    # cost 계산
    cost = torch.mean((hypothesis - y_train) ** 2)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()

    print('Epoch {:4d}/{} hypothesis: {} Cost: {:.6f}'.format(
        epoch, nb_epochs, hypothesis.squeeze().detach(),
        cost.item()
    ))
```

```
Epoch    0/20 hypothesis: tensor([0., 0., 0., 0., 0.]) Cost: 29661.800781
Epoch    1/20 hypothesis: tensor([67.2578, 80.8397, 79.6523, 86.7394, 61.6605]) Cost: 9298.520508
Epoch    2/20 hypothesis: tensor([104.9128, 126.0990, 124.2466, 135.3015,  96.1821]) Cost: 2915.713135
Epoch    3/20 hypothesis: tensor([125.9942, 151.4381, 149.2133, 162.4896, 115.5097]) Cost: 915.040527
Epoch    4/20 hypothesis: tensor([137.7968, 165.6247, 163.1911, 177.7112, 126.3307]) Cost: 287.936005
Epoch    5/20 hypothesis: tensor([144.4044, 173.5674, 171.0168, 186.2332, 132.3891]) Cost: 91.371017
Epoch    6/20 hypothesis: tensor([148.1035, 178.0144, 175.3980, 191.0042, 135.7812]) Cost: 29.758139
Epoch    7/20 hypothesis: tensor([150.1744, 180.5042, 177.8508, 193.6753, 137.6805]) Cost: 10.445305
Epoch    8/20 hypothesis: tensor([151.3336, 181.8983, 179.2240, 195.1707, 138.7440]) Cost: 4.391228
Epoch    9/20 hypothesis: tensor([151.9824, 182.6789, 179.9928, 196.0079, 139.3396]) Cost: 2.493135
Epoch   10/20 hypothesis: tensor([152.3454, 183.1161, 180.4231, 196.4765, 139.6732]) Cost: 1.897688
Epoch   11/20 hypothesis: tensor([152.5485, 183.3610, 180.6640, 196.7389, 139.8602]) Cost: 1.710541
Epoch   12/20 hypothesis: tensor([152.6620, 183.4982, 180.7988, 196.8857, 139.9651]) Cost: 1.651413
Epoch   13/20 hypothesis: tensor([152.7253, 183.5752, 180.8742, 196.9678, 140.0240]) Cost: 1.632387
Epoch   14/20 hypothesis: tensor([152.7606, 183.6184, 180.9164, 197.0138, 140.0571]) Cost: 1.625923
Epoch   15/20 hypothesis: tensor([152.7802, 183.6427, 180.9399, 197.0395, 140.0759]) Cost: 1.623412
Epoch   16/20 hypothesis: tensor([152.7909, 183.6565, 180.9530, 197.0538, 140.0865]) Cost: 1.622141
Epoch   17/20 hypothesis: tensor([152.7968, 183.6643, 180.9603, 197.0618, 140.0927]) Cost: 1.621253
Epoch   18/20 hypothesis: tensor([152.7999, 183.6688, 180.9644, 197.0662, 140.0963]) Cost: 1.620500
Epoch   19/20 hypothesis: tensor([152.8014, 183.6715, 180.9666, 197.0686, 140.0985]) Cost: 1.619770
Epoch   20/20 hypothesis: tensor([152.8020, 183.6731, 180.9677, 197.0699, 140.1000]) Cost: 1.619033
```

Final (y)
152
185
180
196
142

Pytorch에서 nn.module 상속으로 coding

- Nn.module 상속하고 제공되는 function 사용하면 bug 줄이고 편리하게 coding 가능

Imports

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
```

```
# For reproducibility
torch.manual_seed(1)
```

```python
# 모델 초기화
W = torch.zeros((3, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)

# H(x) 계산
hypothesis = x_train.matmul(W) + b # or .mm or @
```

```python
import torch.nn as nn

class MultivariateLinearRegressionModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear = nn.Linear(3, 1)

    def forward(self, x):
        return self.linear(x)

hypothesis = model(x_train)
```

nn.Module 을 상속해서 모델 생성

- nn.Linear(3, 1)
  - ◯ 입력 차원: 3
  - ◯ 출력 차원: 1
- Hypothesis 계산은 forward() 에서
  - Gradient 계산은 backward()

F.mse.loss 사용

```
# cost 계산
cost = torch.mean((hypothesis - y_train) ** 2)
```

```
import torch.nn.functional as F



# cost 계산
cost = F.mse_loss(prediction, y_train)
```

- torch.nn.functional 의 loss function 사용으로 다른 loss ftn 사용시 편리(l1_loss, smooth_l1_loss 등...)

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

```
# 데이터
x1_train = torch.FloatTensor([[73], [93], [89], [96], [73]])
x2_train = torch.FloatTensor([[80], [88], [91], [98], [66]])
x3_train = torch.FloatTensor([[75], [93], [90], [100], [70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])
```

```
# 모델 초기화
w1 = torch.zeros(1, requires_grad=True)
w2 = torch.zeros(1, requires_grad=True)
w3 = torch.zeros(1, requires_grad=True)
b = torch.zeros(1, requires_grad=True)
```

Gradient descent →

```
# optimizer 설정
optimizer = optim.SGD([w1, w2, w3, b], lr=1e-5)
```

```
nb_epochs = 1000
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x1_train * w1 + x2_train * w2 + x3_train * w3 + b

    # cost 계산
    cost = torch.mean((hypothesis - y_train) ** 2)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()
```

By Matrix function

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1 w_1 + x_2 w_2 + x_3 w_3 \end{pmatrix}$$

```
x_train = torch.FloatTensor([[73, 80, 75],
                             [93, 88, 93],
                             [89, 91, 90],
                             [96, 98, 100],
                             [73, 66, 70]])
y_train = torch.FloatTensor([[152], [185], [180], [196], [142]])
```

```
print(x_train.shape)
print(y_train.shape)
```

```
torch.Size([5, 3])
torch.Size([5, 1])
```

```
# 모델 초기화
W = torch.zeros((3, 1), requires_grad=True)
b = torch.zeros(1, requires_grad=True)
```

```
# optimizer 설정
optimizer = optim.SGD([W, b], lr=1e-5)

nb_epochs = 20
for epoch in range(nb_epochs + 1):

    # H(x) 계산
    hypothesis = x_train.matmul(W) + b # or .mm or @

    # cost 계산
    cost = torch.mean((hypothesis - y_train) ** 2)

    # cost로 H(x) 개선
    optimizer.zero_grad()
    cost.backward()
    optimizer.step()
```

Print output

```
    # 100번마다 로그 출력
    if epoch % 100 == 0:
        print('Epoch {:4d}/{} w1: {:.3f} w2: {:.3f} w3: {:.3f} b: {:.3f} Cost: {:.6f}'.format(
            epoch, nb_epochs, w1.item(), w3.item(), w3.item(), b.item(), cost.item()
        ))
```

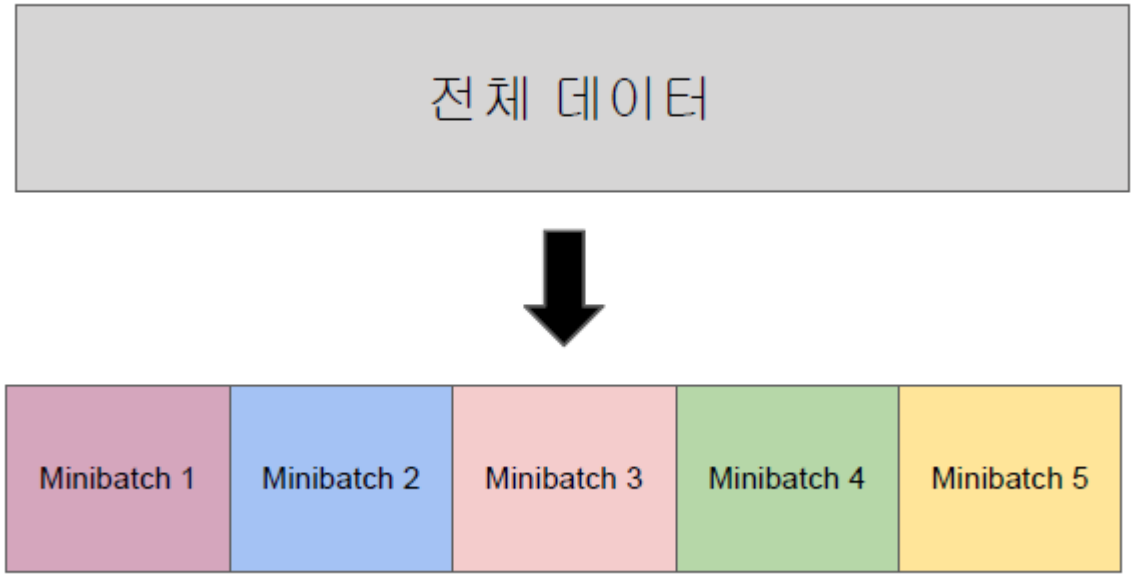
```
Epoch    0/1000 w1: 0.294 w2: 0.297 w3: 0.297 b: 0.003 Cost: 29661.800781
Epoch  100/1000 w1: 0.674 w2: 0.676 w3: 0.676 b: 0.008 Cost: 1.563634
Epoch  200/1000 w1: 0.679 w2: 0.677 w3: 0.677 b: 0.008 Cost: 1.497607
Epoch  300/1000 w1: 0.684 w2: 0.677 w3: 0.677 b: 0.008 Cost: 1.435026
Epoch  400/1000 w1: 0.689 w2: 0.678 w3: 0.678 b: 0.008 Cost: 1.375730
Epoch  500/1000 w1: 0.694 w2: 0.678 w3: 0.678 b: 0.009 Cost: 1.319511
Epoch  600/1000 w1: 0.699 w2: 0.679 w3: 0.679 b: 0.009 Cost: 1.266222
```

Loading Data

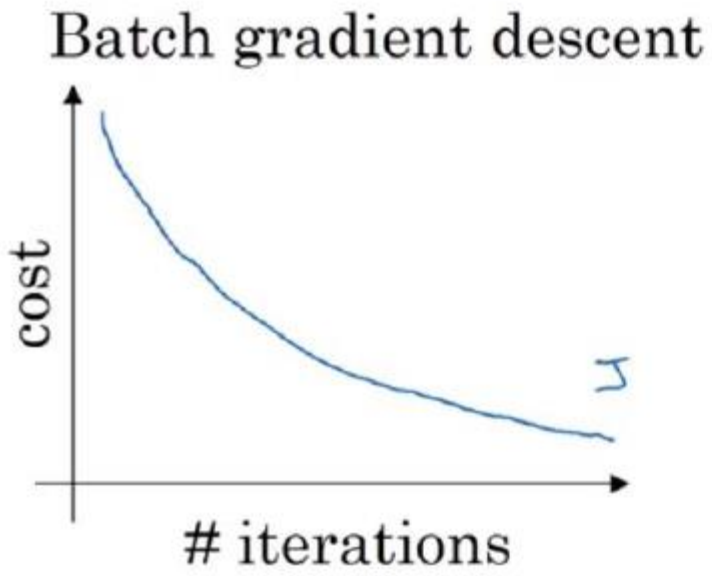
- Big size data load
- Issue
  - Complex DNN needs huge data et to learn
  - Not possible train the network on big size data at once.
  - Big size data make your computer slow!
  - Sometimes not possible calculate gradient descent values.

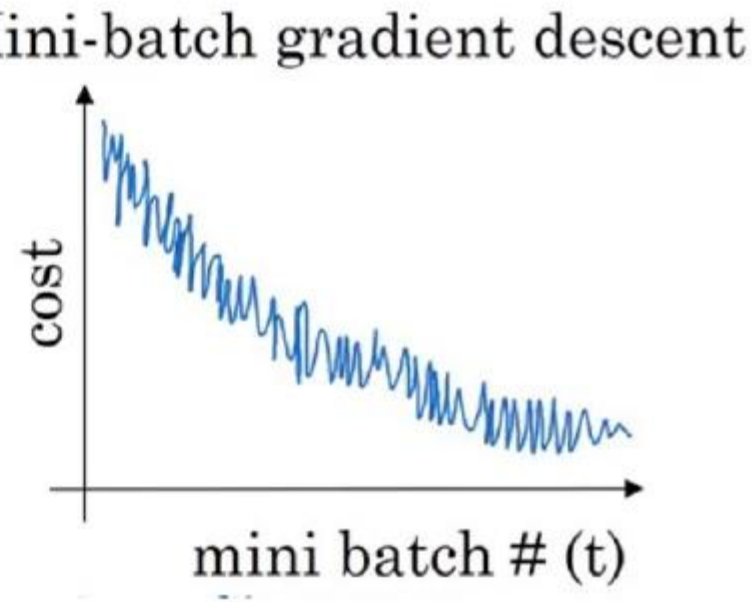
Minibatch Gradient Descent

- 전체 데이터를 균일하게 나눠서 학습



- 업데이트를 좀 더 빠르게 할 수 있음.
- 전체 데이터를 쓰지 않아서 잘못된 방향으로 업데이트를 할 수도 있음





Pytorch Data set

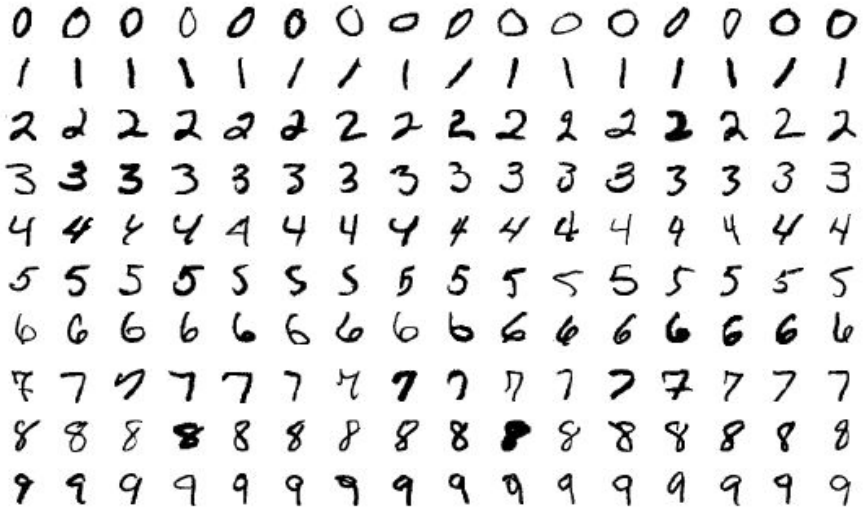
```
from torch.utils.data import Dataset


class CustomDataset(Dataset):
    def __init__(self):
        self.x_data = [[73, 80, 75],
                       [93, 88, 93],
                       [89, 91, 90],
                       [96, 98, 100],
                       [73, 66, 70]]
        self.y_data = [[152], [185], [180], [196], [142]]

    def __len__(self):
        return len(self.x_data)

    def __getitem__(self, idx):
        x = torch.FloatTensor(self.x_data[idx])
        y = torch.FloatTensor(self.y_data[idx])

        return x, y


dataset = CustomDataset()
```

- `torch.utils.data.Dataset` 상속

- `__len__()`
  - 이 데이터셋의 총 데이터 수
- `__getitem__()`
  - 어떠한 인덱스 **idx** 를 받았을 때, 그에 상응하는 입출력 데이터 반환

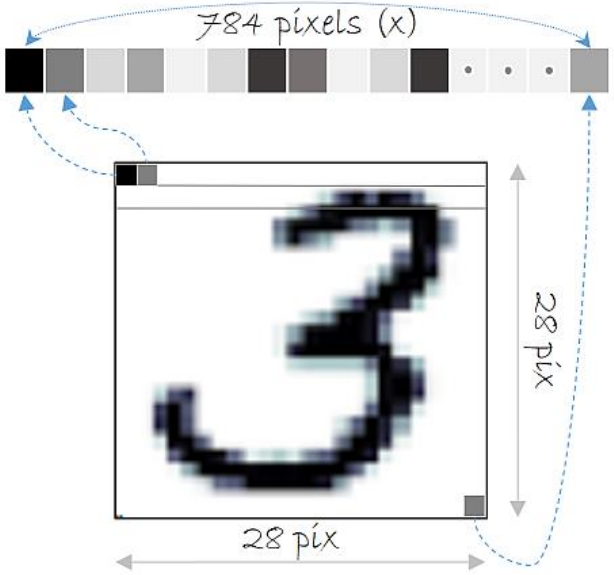
MNIST data 이용한 필기체 숫자인식

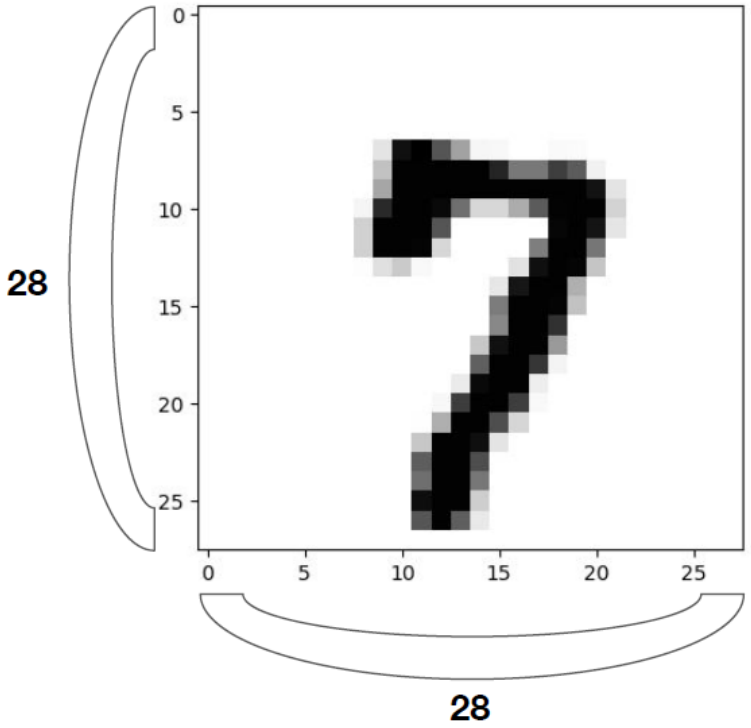
MNIST: handwritten digits dataset



train-images-idx3-ubyte.gz: training set images (9912422 bytes; 60,000 samples)
train-labels-idx1-ubyte.gz: training set labels (28881 bytes)
T10k-images-idx3-ubyte.gz : test set images (1648877 bytes; 10,000 samples)
T10k-labels-idx1-ubyte.gz : test set labels (4542 bytes)

- 28*28=784 개 data
- X.view 함수가 28*28 data를 784개 입력 data로 변환 시킴





- **28 x 28 image**
- **1 channel gray image**
- **0 ~ 9 digits**

```
for X, Y in data_loader:
    # reshape input image into [batch_size by 784]
    # label is not one-hot encoded
    X = X.view(-1, 28 * 28)
```

torchvision

The torchvision package consists of popular datasets, model architectures, and common image transformations for computer vision.

torchvision.datasets
- MNIST
- Fashion-MNIST
- EMNIST
- COCO
- LSUN
- ImageFolder
- DatasetFolder
- Imagenet-12
- CIFAR
- STL10
- SVHN
- PhotoTour
- SBU
- Flickr
- VOC

torchvision.models
- Alexnet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3

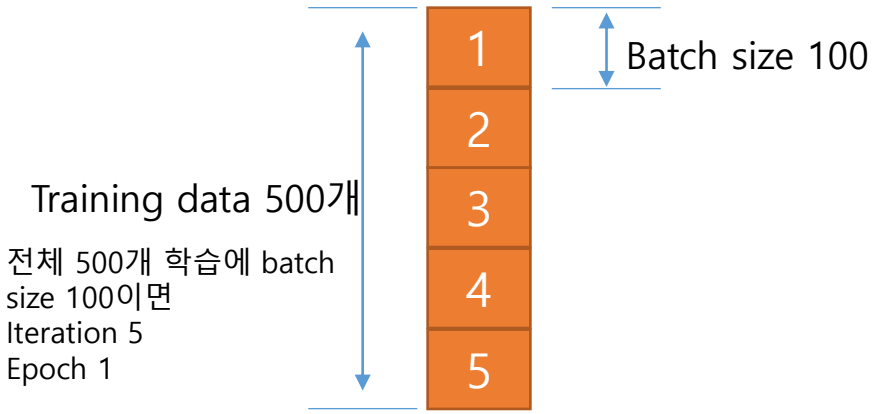
torchvision.transforms
- Transforms on PIL Image
- Transforms on torch.*Tensor
- Conversion Transforms
- Generic Transforms
- Functional Transforms

torchvision.utils

https://pytorch.org/docs/stable/torchvision/index.html

- Batch size, iteration , epoch

Batch size : 전체 training data set을 여러 작은 group으로 나누었을때 하나의 소그룹에 속하는 데이터 수



Torchvision package에 있는 dataset 이용 MNIST 불러오기

```
# Lab 7 Learning rate and Evaluation
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import random
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
random.seed(777)
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

실험결과를 재현하거나 개선되었는지 확인하기 위해 동일한 랜덤 수 사용 필요

Torchvision package에 있는 dataset 이용 MNIST 불러오기

```
# parameters
training_epochs = 15
batch_size = 100
```

MNIST data 위치

MNIST 이미지 불러올때 어떤 transform 적용해 불러올것인지
Pytorch에서 image는 0~1 사이값과 Ch, Height, width 형태
일반 image는 0~255 H, W, C 형태를 갖음

```
# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)
```

true이면 trainset load

학습용 data

tested data

Root 에 MNIST data 없으면 down 받겠다는것

Data 불러오기

Load 할 Data 종류

```
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

Batch 사이즈로 잘라 사용시 맨 마지막 데이터 수가
batchsize 보다 작으면 버리기

```
# MNIST data image of shape 28 * 28 = 784
linear = torch.nn.Linear(784, 10, bias=True).to(device)
```

```
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.SGD(linear.parameters(), lr=0.1)
```

Weight, bias

Learning rate=0.1

Stochastic gradient descent

MNIST image
Label

```
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = len(data_loader)

    for X, Y in data_loader:
        # reshape input image into [batch_size by 784]
        # label is not one-hot encoded
        X = X.view(-1, 28 * 28).to(device)
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = linear(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning finished')
```

```
Epoch: 0001 cost = 0.535468459
Epoch: 0002 cost = 0.359274179
Epoch: 0003 cost = 0.331187546
Epoch: 0004 cost = 0.316578031
Epoch: 0005 cost = 0.307158172
Epoch: 0006 cost = 0.300180733
Epoch: 0007 cost = 0.295130193
Epoch: 0008 cost = 0.290851533
Epoch: 0009 cost = 0.287417084
Epoch: 0010 cost = 0.284379542
Epoch: 0011 cost = 0.281825215
Epoch: 0012 cost = 0.279800713
Epoch: 0013 cost = 0.277808994
Epoch: 0014 cost = 0.276154280
Epoch: 0015 cost = 0.274440825
Learning finished
```

With 아래 구간에서 gradient 계산하지 않겠다 는 것

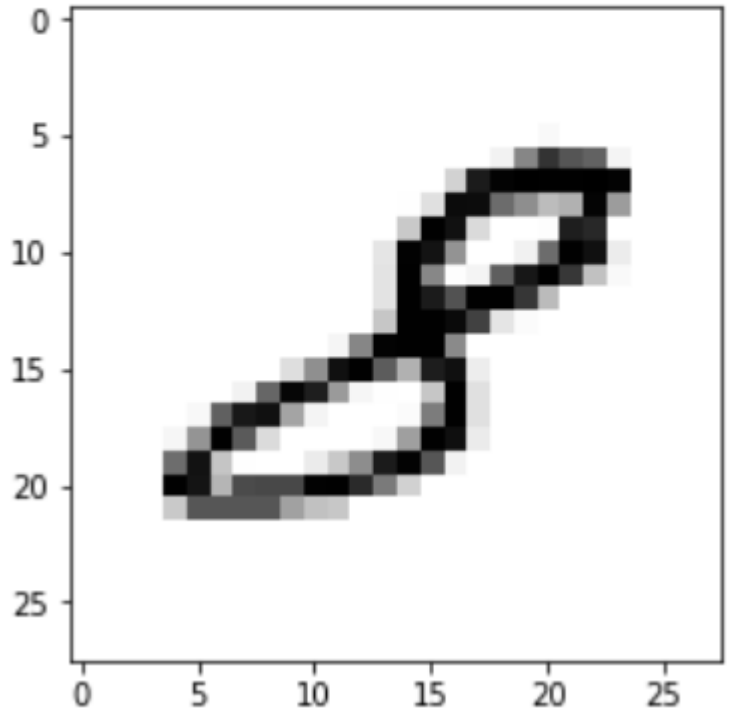
to() 함수는 연산을 어디서 수행할지를 정합니다. to() 함수는 모델의 매개변수를 지정한 장치의 메모리로 보냅니다. CPU를 사용할 경우에는 필요가 없지만, GPU를 사용하려면 to('cuda'

X는 텐서를 (?, 28*28)의 크기로 변경

```
# Test the model using test sets
with torch.no_grad():
    X_test = mnist_test.test_data.view(-1, 28 * 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = linear(X_test)
    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())

    # Get one and predict
    r = random.randint(0, len(mnist_test) - 1)
    X_single_data = mnist_test.test_data[r:r + 1].view(-1, 28 * 28).float().to(device)
    Y_single_data = mnist_test.test_labels[r:r + 1].to(device)

    print('Label: ', Y_single_data.item())
    single_prediction = linear(X_single_data)
    print('Prediction: ', torch.argmax(single_prediction, 1).item())

    plt.imshow(mnist_test.test_data[r:r + 1].view(28, 28), cmap='Greys', interpolation='nearest')
    plt.show()
```

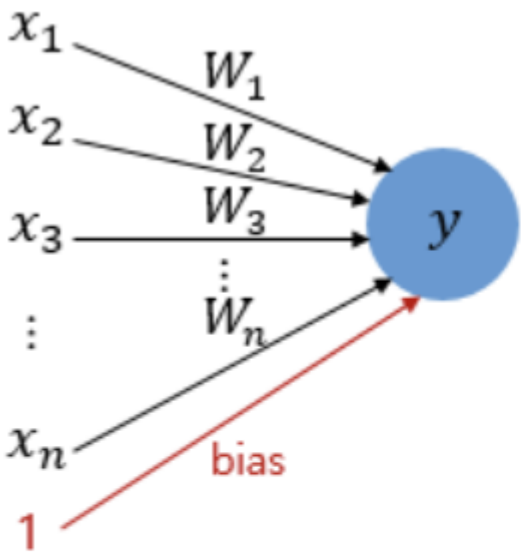
- lab-07_2_mnist_introduction.ipynb

```
Accuracy: 0.8862999677658081
Label:  8
Prediction:  3
```



Perceptron XOR

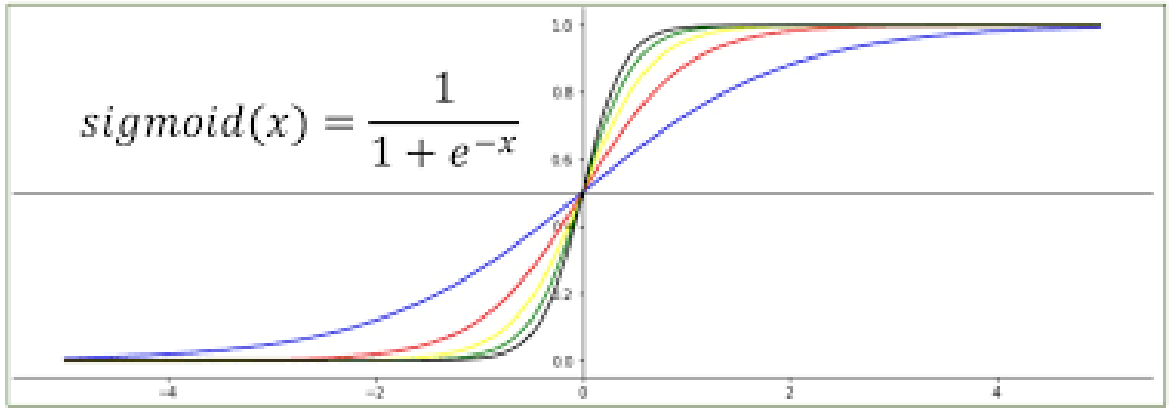
퍼셉트론



Linear repression | MSE

Binary classfication | B.C

Multi variable | ∩ C



Synaps

$x_1 w_1$

$x_2 w_2$

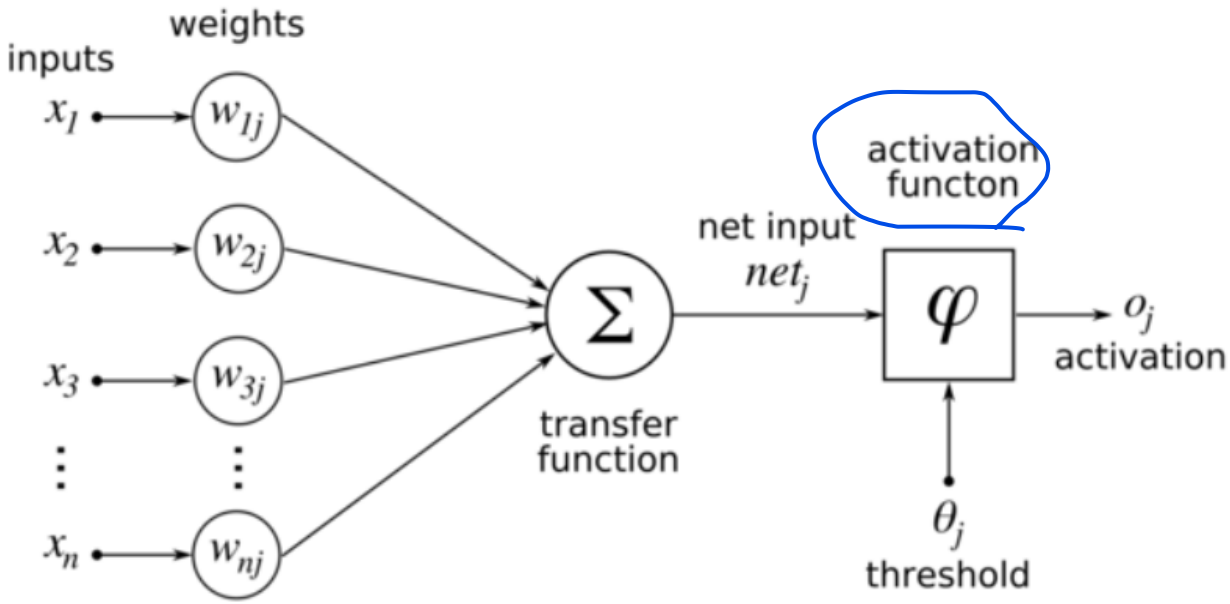
Σ

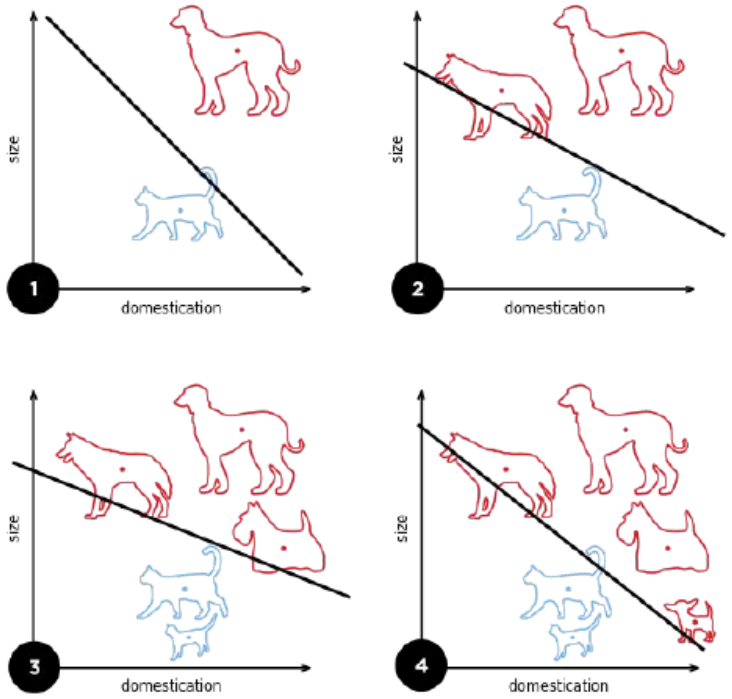
$$if \sum_i^n W_i x_i + b \geq 0 \rightarrow y = 1$$

$$if \sum_i^n W_i x_i + b < 0 \rightarrow y = 0$$

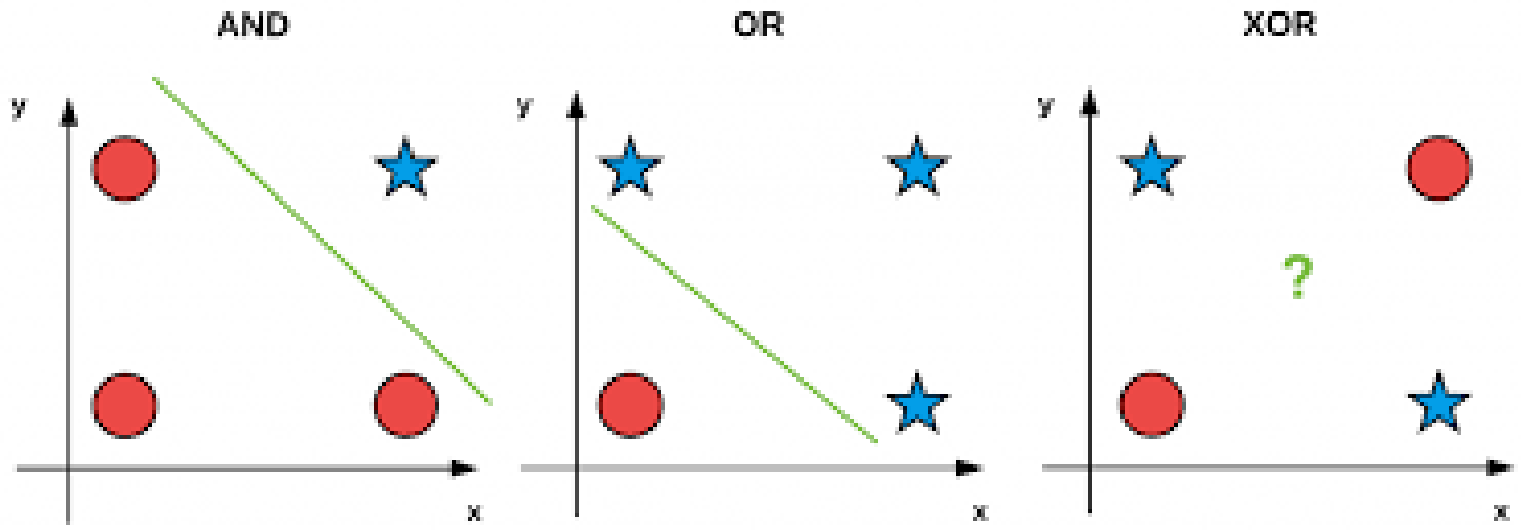
A	B	OR	AND	XOR
0	0	0	0	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	0

Perceptron XOR





- **Linear classifier**


AND
OR
XOR
y
x
y
x
y
?
x

Cross entropy

- 딥러닝에서는 분류문제에 대한 Cost Function으로 Cross Entropy를 사용
  - 분류의 대상이 참/거짓 처럼 2개인 경우 binary cross entropy를 사용
  - 이미지넷과 같이 수많은 종류의 대상을 분류하는 경우에는 multi cross entropy를 사용

$$H(P,Q) = \sum_{i=1}^{k} log_2 \frac{1}{Q_i} * P_i$$

Qi (예측확률), *Pi* (실제확률)

1.예측값과 실제값이 완전히 다른 경우, Cross Entropy의 값은 무한대.
2.예측값과 실제값이 완전히 일치하면, Cross Entropy는 Entropy와 동일.
3.머신러닝에서 학습이 잘 진행되면, Cross Entropy는 Entropy와 근접.

Implementation

```
import torch
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)
```

XOR 입력, 출력

```
linear = nn.Linear(2, 1, bias=True)
sigmoid = nn.Sigmoid()
model = nn.Sequential(linear, sigmoid).to(device)
```

입력 2., 출력 1 linear model
시그모이드 함수

가져온 레이어들을 연결해서 퍼셉트론 모델을 완성.

```
# 비용 함수와 옵티마이저 정의
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)
```

```
#10,001번의 에포크 수행. 0번 에포크부터 10,000번 에포크까지.
for step in range(10001):
    optimizer.zero_grad()
    hypothesis = model(X)

    # 비용 함수
    cost = criterion(hypothesis, Y)
    cost.backward()
    optimizer.step()

    if step % 100 == 0: # 100번째 에포크마다 비용 출력
        print(step, cost.item())
```

- 이진 분류를 목적으로 하므로 Binary Cross Entropy Loss를 사용
- 최적화 방법으로는 변함없이 경사하강법(SGD)를 사용

- 학습 10000번 반복하고, 100의 배수번째 학습때마다 cost를 출력

```
0 0.7273974418640137
100 0.6931475400924683
200 0.6931471824645996
300 0.6931471824645996
...
9900 0.6931471824645996
10000 0.6931471824645996
```

Multi Layer Perceptron

LR | B.C | Sigmoid
M | Softmax

- XOR

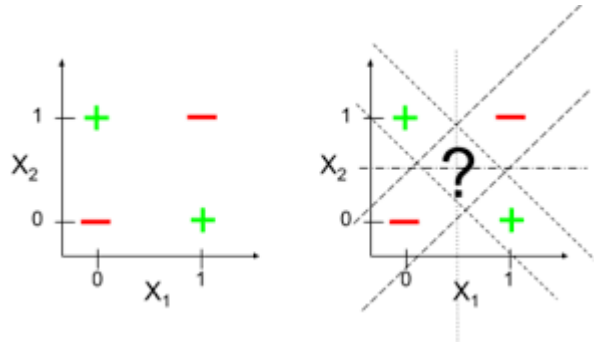
a)

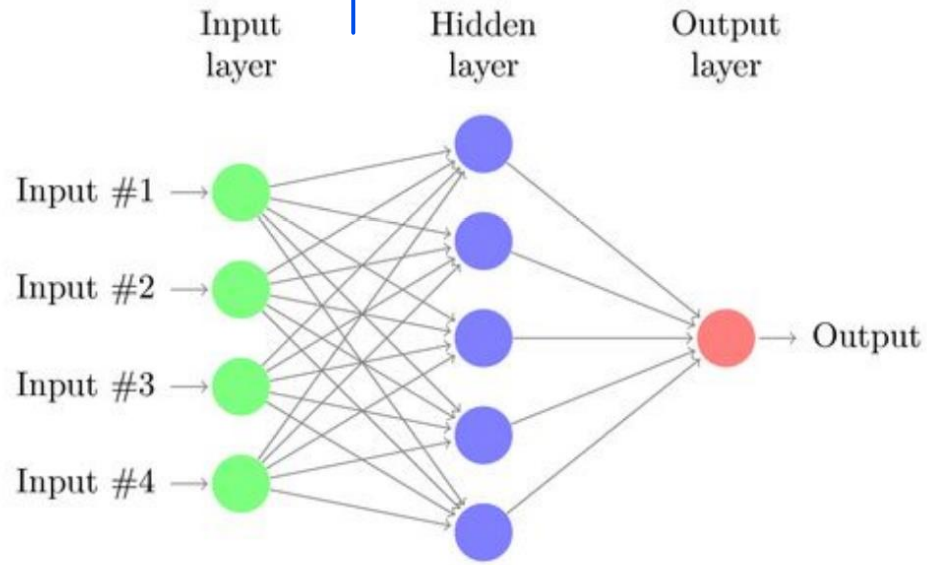
XOR Gate

b)

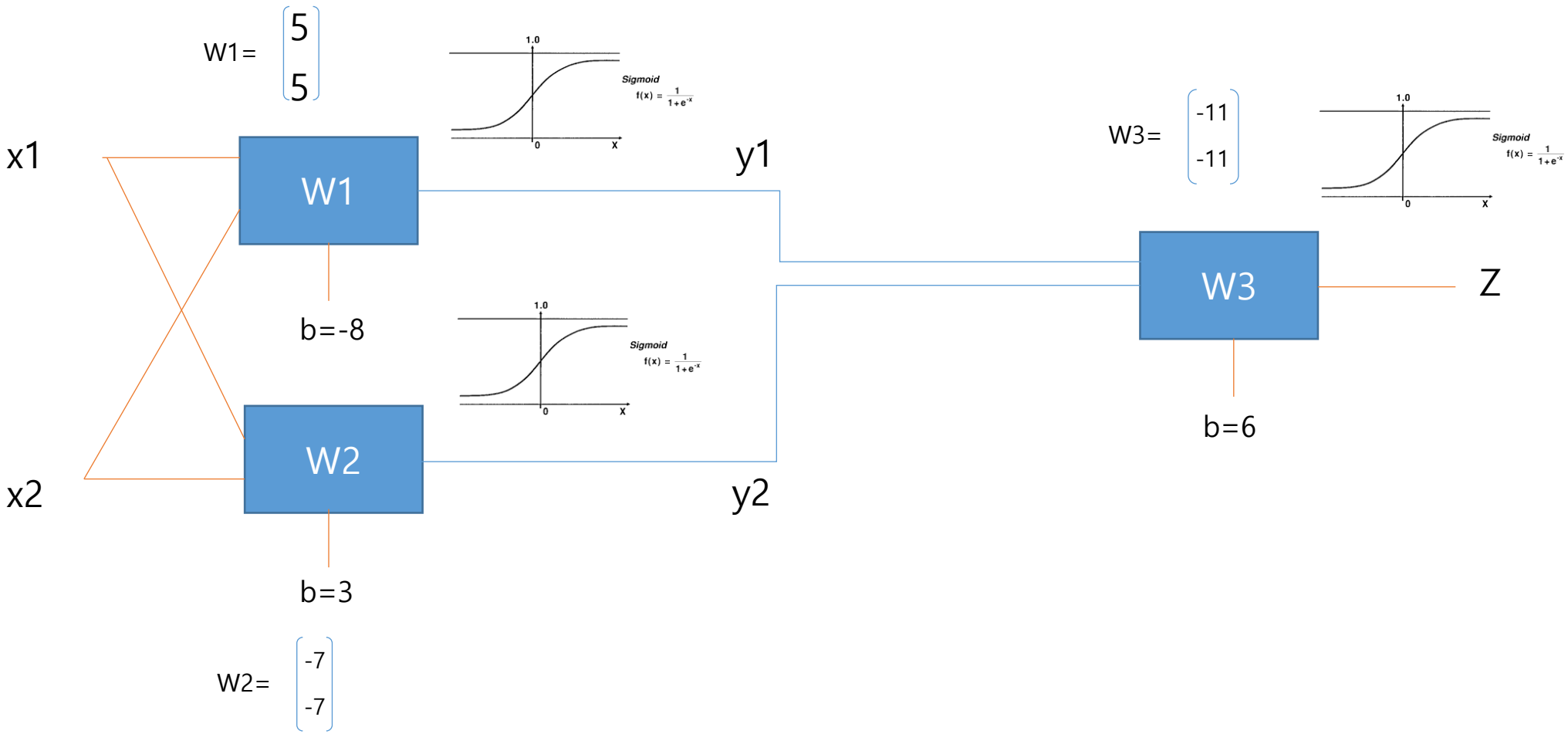
A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

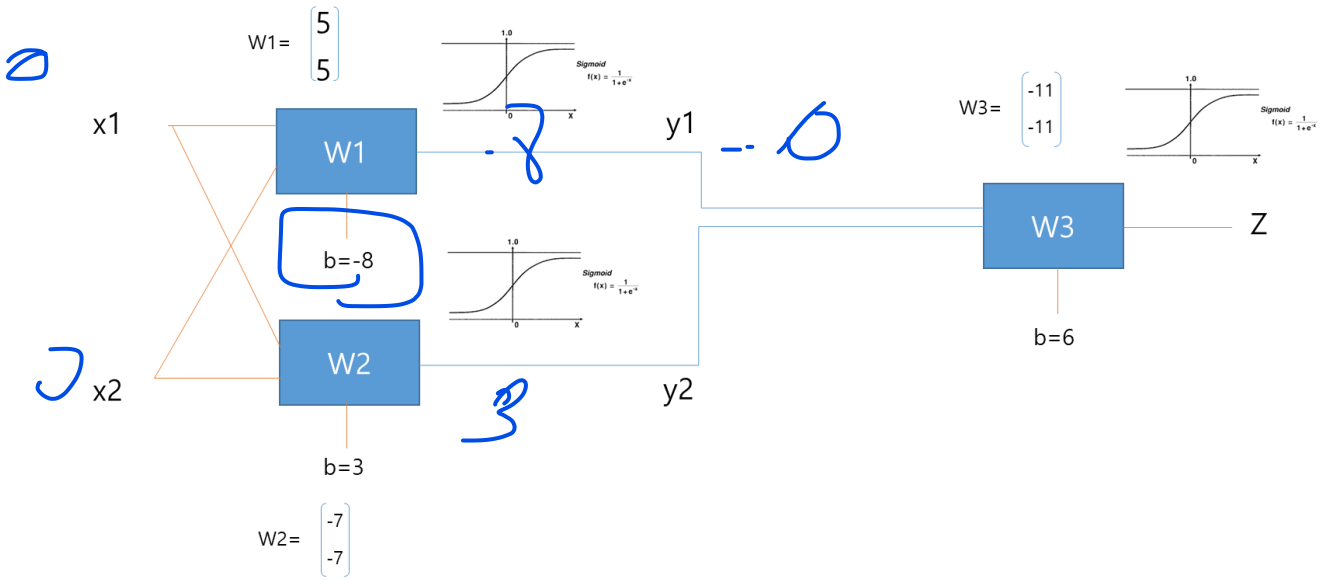
XOR accuracy table





- We need to use MLP, multilayer perceptrons (multilayer neural nets)
- No one on earth had found a viable way to train MLPs good enough to learn such simple functions (by Marvin Minsky, founder of the MIT AI Lab, 1969)


W1=
5
5
1.0
Sigmoid
f(x) = 1/(1+e^-x)
0
X
x1
W1
y1
b=-8
W3=
-11
-11
1.0
Sigmoid
f(x) = 1/(1+e^-x)
0
X
W3
Z
1.0
Sigmoid
f(x) = 1/(1+e^-x)
0
X
b=6
W2
x2
y2
b=3
W2=
-7
-7



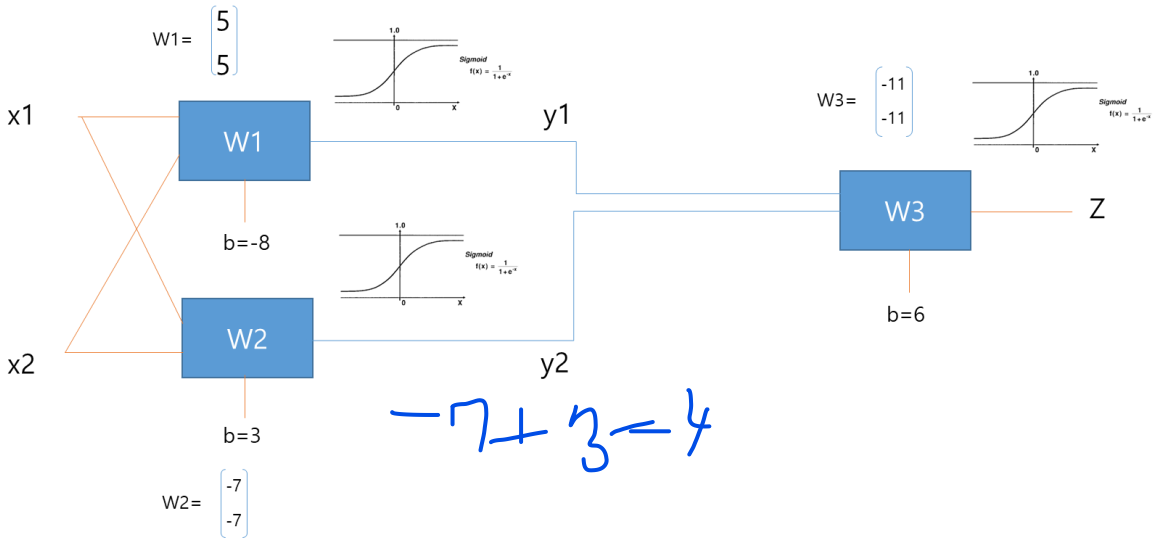
$[0\ 0]\begin{bmatrix}5\\5\end{bmatrix}-8=-8,\ y_1=\text{sigmoid}(-8)=0$

$[0\ 0]\begin{bmatrix}7\\-7\end{bmatrix}+3=3,\ y_2=\text{sigmoid}(3)=1$

$[0\ 1]\begin{bmatrix}-11\\-11\end{bmatrix}+6=-5\quad z=\text{sigmoid}(-5)=0$

x1	x2	y1	y2	z	XOR
0	0				0
0	1				1
1	0				1
1	1				0

[0.7]



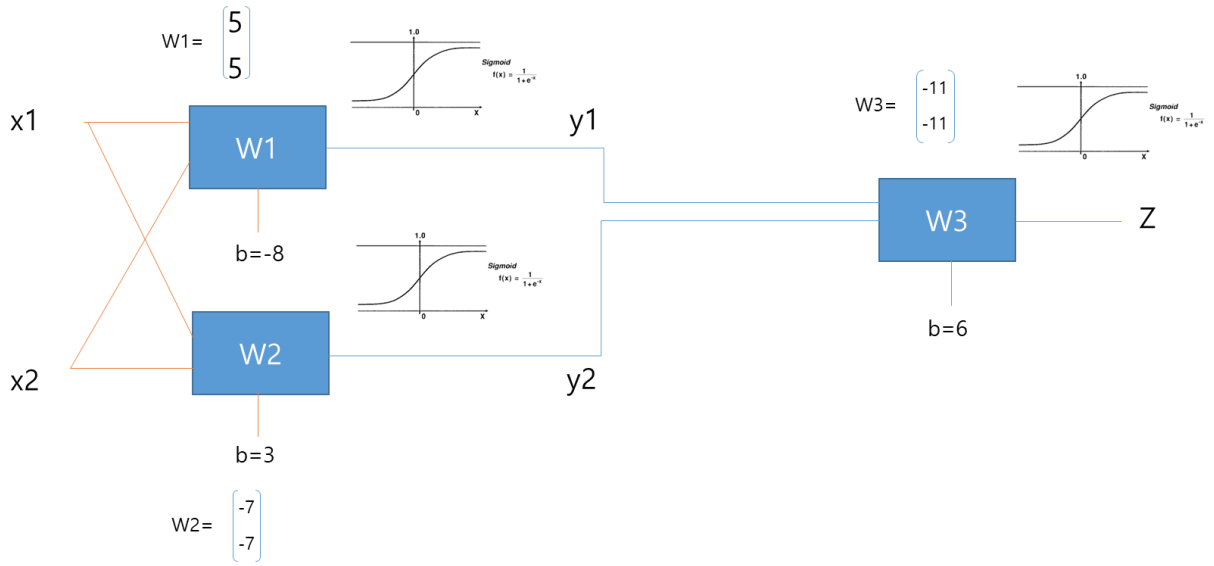
$-7+3=4$

$[0\ 1]\begin{bmatrix}5\\5\end{bmatrix}-8=-8,\ y_1=\text{sigmoid}(-8)=0$

$[0\ 1]\begin{bmatrix}-7\\-7\end{bmatrix}+3=-4\quad y_2=\text{sigmoid}(-4)=0$

$[0\ 0]\begin{bmatrix}-11\\-11\end{bmatrix}+6=6\quad z=\text{sigmoid}(6)=1$

x1	x2	y1	y2	z	XOR
0	0	0	0	0	0
0	1	0	0	1	1
1	0				1
1	1				0

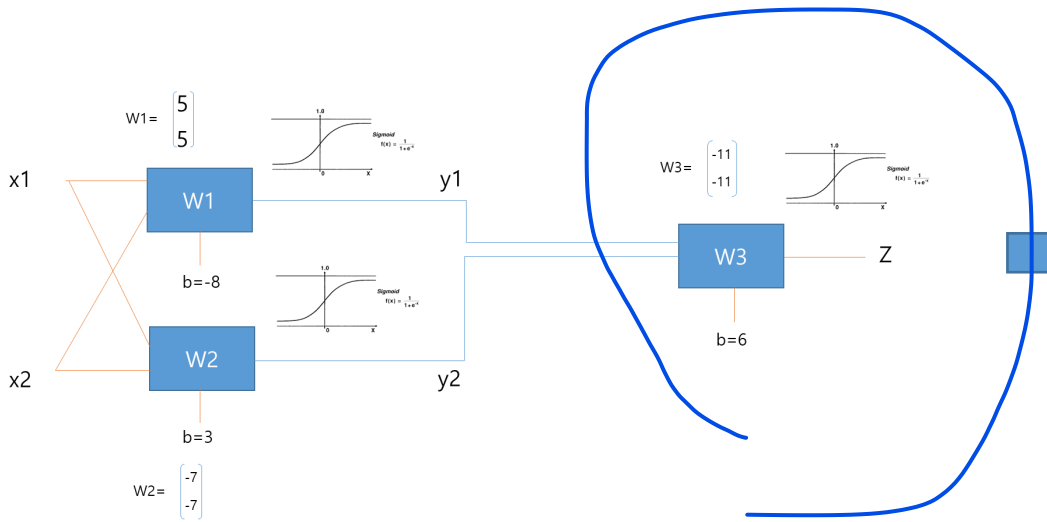


[1 1] $\begin{bmatrix}5\\5\end{bmatrix}$ −8 = 2, y_1 = sigmoid(2) = 1

[1 1] $\begin{bmatrix}-7\\-7\end{bmatrix}$ +3 = −11 y_2 = sigmoid(−11) = 0

[1 0] $\begin{bmatrix}-11\\-11\end{bmatrix}$ +6 = −5 z = sigmoid(−5) = 0

x1	x2	y1	y2	z	XOR
0	0	0	1	0	0
0	1	0	0	1	1
1	0	0	0	1	1
1	1	1	0	0	0



X

0, 0

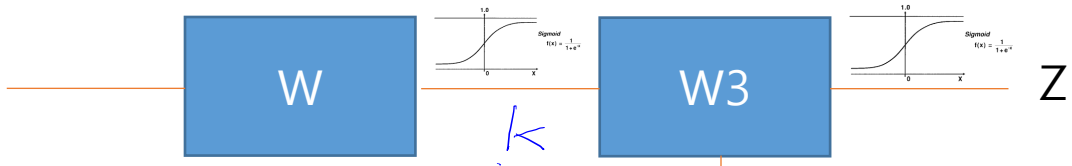
0, 1

1, 0

1, 1

W= $\begin{pmatrix} 5 & -7 \\ 5 & -7 \end{pmatrix}$

W3= $\begin{pmatrix} -11 \\ -11 \end{pmatrix}$



B= $\begin{pmatrix} -8, & 3 \end{pmatrix}$

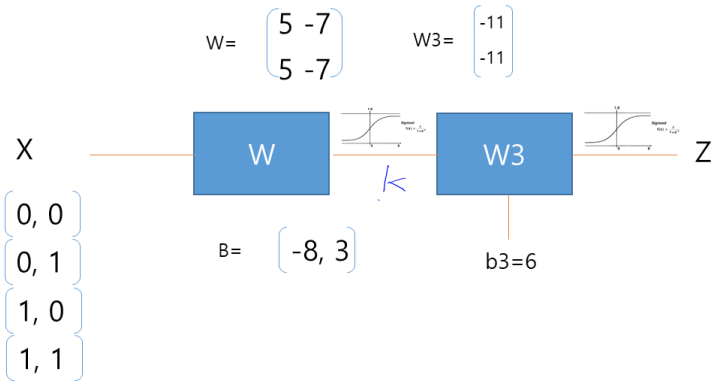
b3=6

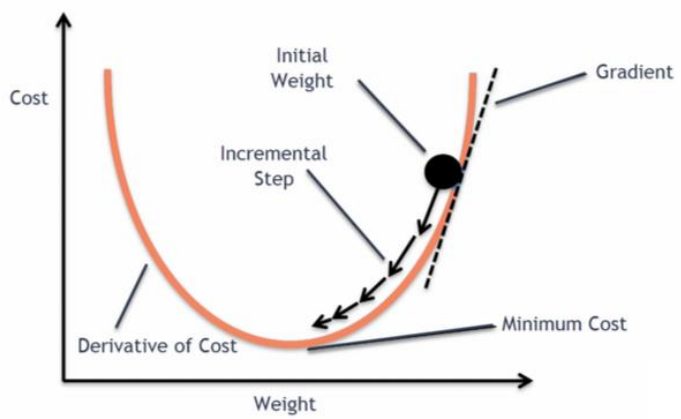
$K(x) = \text{Sigmoid}(xW_1 + B_1)$

$Z = H(x) = \text{Sigmoid}(K(x)W_2 + b_2)$

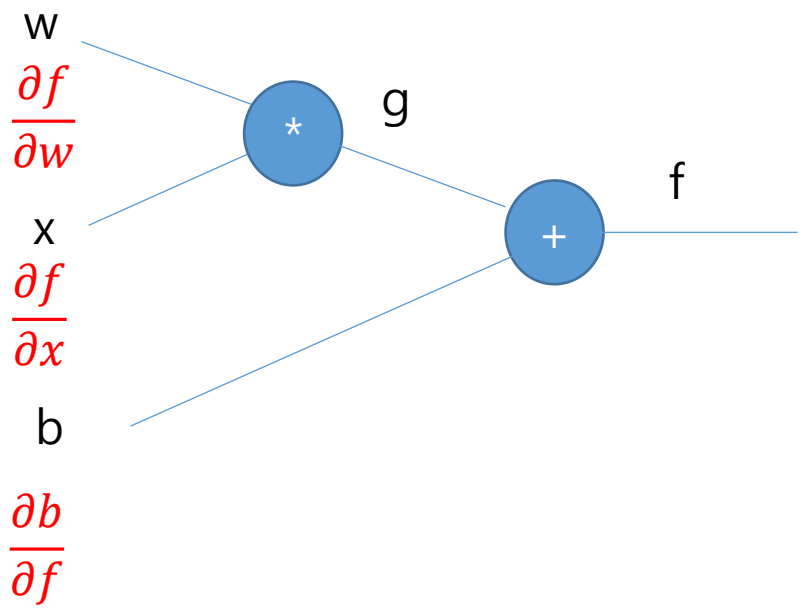
Back propagation

- Update W, w3, B, b3 from training data





- f=wx+b, g=wx
- f=g+b
- W, x, b 가 f에 미치는 영향을 미분함수로 구함



1. Forward(w=-2, x=5, b=3)
2. backward

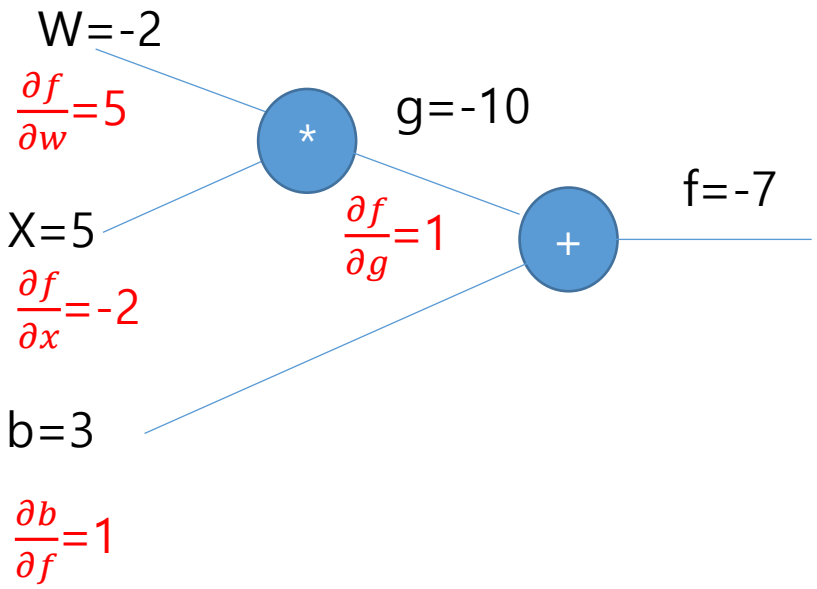
f=wx+b, g=wx, f=g+b

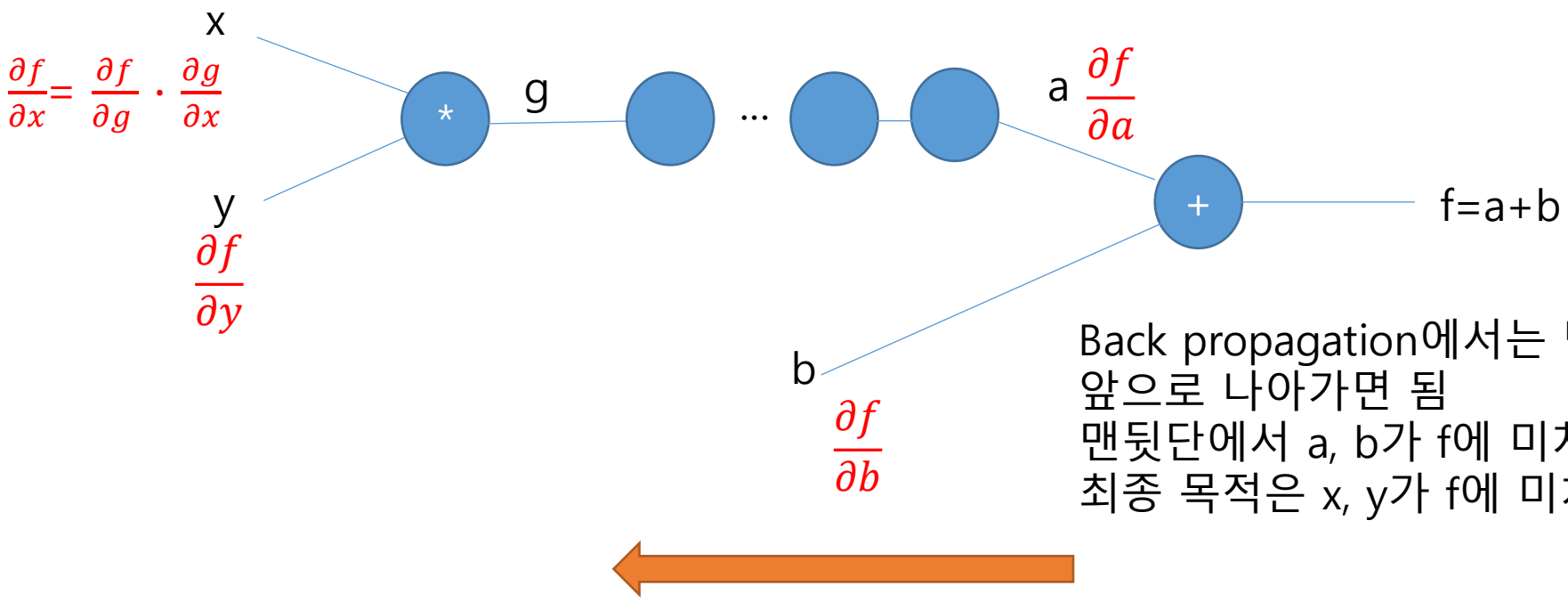
$\frac{\partial g}{\partial w}$=x $\frac{\partial g}{\partial x}$=w

$\frac{\partial f}{\partial g}$=1 $\frac{\partial f}{\partial b}$=1

$\frac{\partial f}{\partial w}=\frac{\partial f}{\partial g}\cdot\frac{\partial g}{\partial w}$= 1*5=5

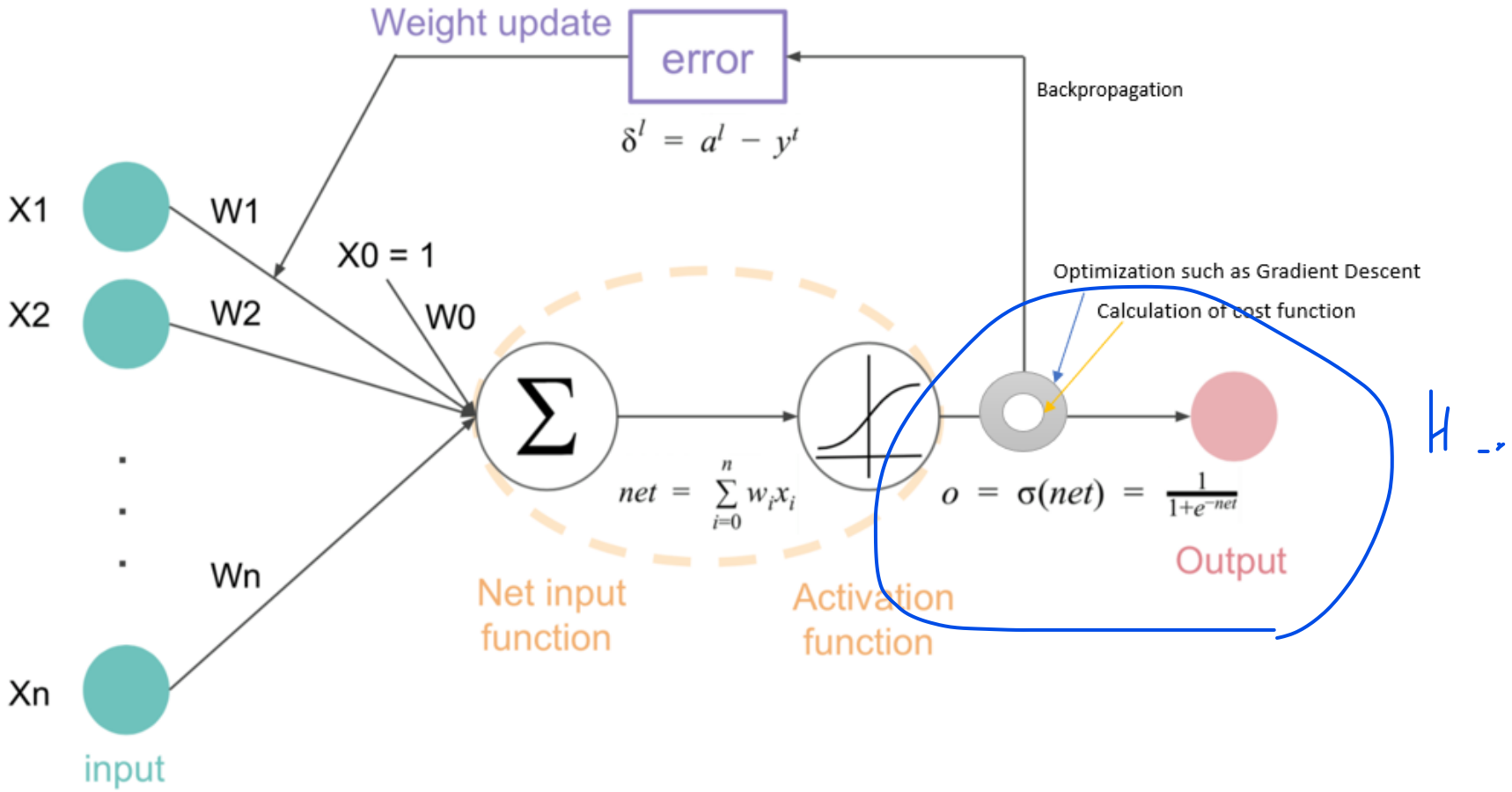
$\frac{\partial f}{\partial x}=\frac{\partial f}{\partial g}\cdot\frac{\partial g}{\partial x}$= 1*-2=-2

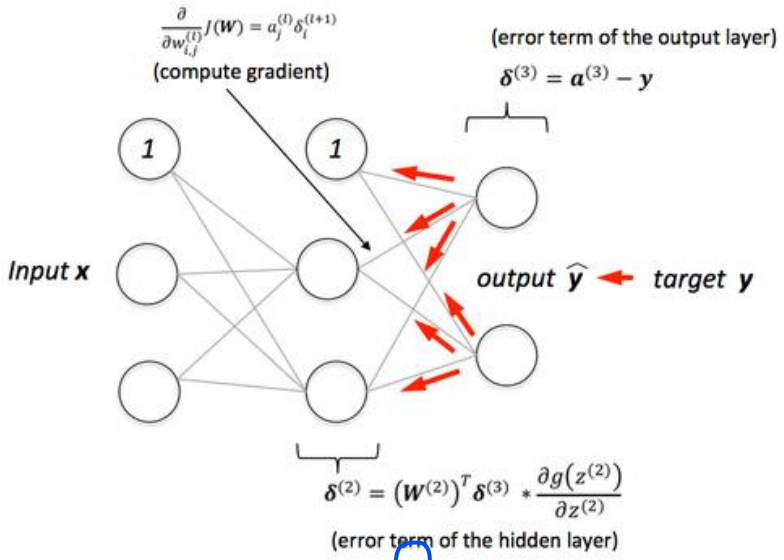


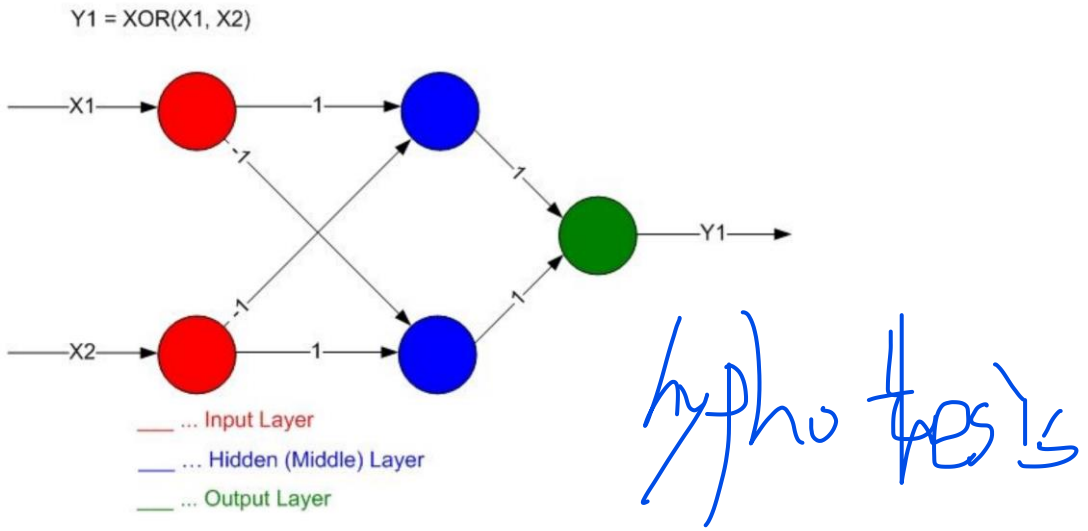


Back propagation에서는 맨 뒷단 부터 계산해 앞으로 나아가면 됨
맨뒷단에서 a, b가 f에 미치는 영향 계산. $\frac{\partial f}{\partial a}$, $\frac{\partial f}{\partial b}$
최종 목적은 x, y가 f에 미치는 영향 계산 하는 것

Back propagation







$\frac{\partial f(Loss)}{\partial W}$

$Loss = Output - target$

$\frac{\partial f(Loss)}{\partial w}$ 로 loss 값을 최소화 할 수 있은 W를 뒷 layer 부터 찾아 나감

Back propagation Code

```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)

# nn layers
w1 = torch.Tensor(2, 2).to(device)
b1 = torch.Tensor(2).to(device)
w2 = torch.Tensor(2, 1).to(device)
b2 = torch.Tensor(1).to(device)

def sigmoid(x):
 # sigmoid function
 return 1.0 / (1.0 + torch.exp(-x))
 # return torch.div(torch.tensor(1), torch.add(torch.tensor(1.0), torch.exp(-x)))

def sigmoid_prime(x):
 # derivative of the sigmoid function
 return sigmoid(x) * (1 - sigmoid(x))
```

XOR 학습데이터와 target

Weight와 bias 선언

a)

XOR Gate

b)

A	B	OUT
0	0	0
0	1	1
1	0	1
1	1	0

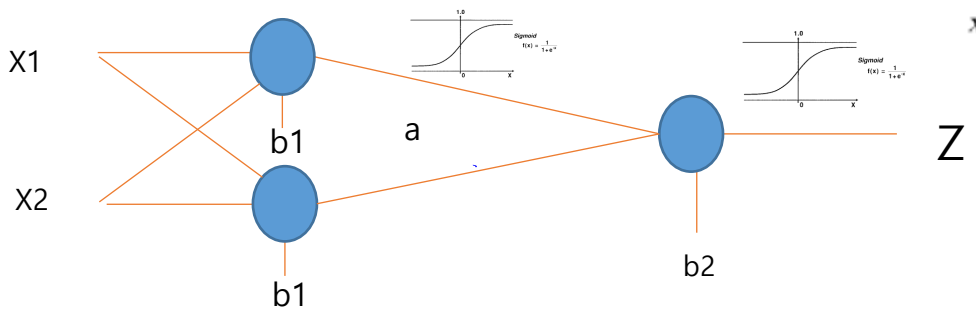
XOR accuracy table

2 X 2

W1 = $\begin{pmatrix} W11 & w12 \\ W21 & w22 \end{pmatrix}$

2 X 1

W2 = $\begin{pmatrix} W11 \\ w21 \end{pmatrix}$



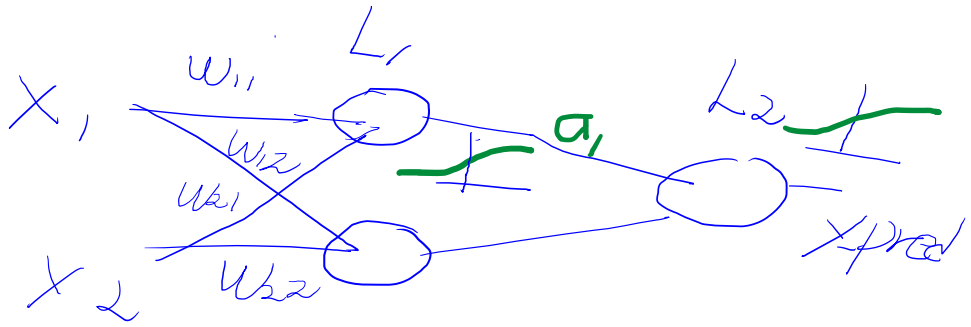
Forward part

```
for step in range(10001):
    # forward
    l1 = torch.add(torch.matmul(X, w1), b1)
    a1 = sigmoid(l1)
    l2 = torch.add(torch.matmul(a1, w2), b2)
    Y_pred = sigmoid(l2)

    cost = -torch.mean(Y * torch.log(Y_pred) + (1 - Y) * torch.log(1 - Y_pred))
```

Binary cross entropy loss

$$-\frac{1}{N}\sum_{i=1}^{N} y_i \cdot log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$$

Binary Cross-Entropy / Log Loss



Back propagation part

```
for step in range(10001):
    # forward
    ...
    # Back prop (chain rule)
    # Loss derivative
    d_Y_pred = (Y_pred - Y) / (Y_pred * (1.0 - Y_pred) + 1e-7)

    # Layer 2
    d_l2 = d_Y_pred * sigmoid_prime(l2)
    d_b2 = d_l2
    d_w2 = torch.matmul(torch.transpose(a1, 0, 1), d_b2)

    # Layer 1
    d_a1 = torch.matmul(d_b2, torch.transpose(w2, 0, 1))
    d_l1 = d_a1 * sigmoid_prime(l1)
    d_b1 = d_l1
    d_w1 = torch.matmul(torch.transpose(X, 0, 1), d_b1)
```

Derivative of Binary cross entropy

0으로 나누는 것을 방지하기 위해 추가한 항

Derivative of sigmoid ftn

Derivative of bias

Derivative of weight

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial a} \cdot \frac{\partial a}{\partial w} =$$

Weight update

```
for step in range(10001):
    # forward
    ...
    # Back prop (chain rule)
    ...

    # Weight update
    w1 = w1 - learning_rate * d_w1
    b1 = b1 - learning_rate * torch.mean(d_b1, 0)
    w2 = w2 - learning_rate * d_w2
    b2 = b2 - learning_rate * torch.mean(d_b2, 0)

    if step % 100 == 0:
        print(step, cost.item())
```

Back propagation with Pytorch

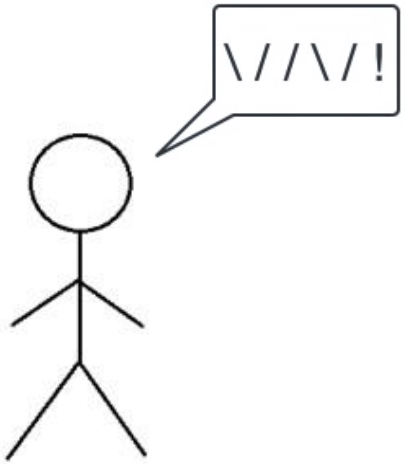
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)

# nn layers
linear1 = torch.nn.Linear(2, 2, bias=True)
linear2 = torch.nn.Linear(2, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
model = torch.nn.Sequential(linear1, sigmoid, linear2, sigmoid).to(device)

# define cost/loss & optimizer
criterion = torch.nn.BCELoss().to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=1)
for step in range(10001):
        optimizer.zero_grad()
        hypothesis = model(X)

        # cost/loss function
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()
        if step % 100 == 0:
                print(step, cost.item())
```

Xor-nn-wide-deep

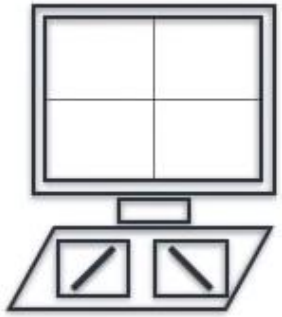
```
X = torch.FloatTensor([[0, 0], [0, 1], [1, 0], [1, 1]]).to(device)
Y = torch.FloatTensor([[0], [1], [1], [0]]).to(device)

# nn layers. 4 layer
linear1 = torch.nn.Linear(2, 10, bias=True)
linear2 = torch.nn.Linear(10, 10, bias=True)
linear3 = torch.nn.Linear(10, 10, bias=True)
linear4 = torch.nn.Linear(10, 1, bias=True)
sigmoid = torch.nn.Sigmoid()
...
```

Convolution Neural Network

Convolutional Neural Networks



Person



Computer

Alphabet

Understanding CNN

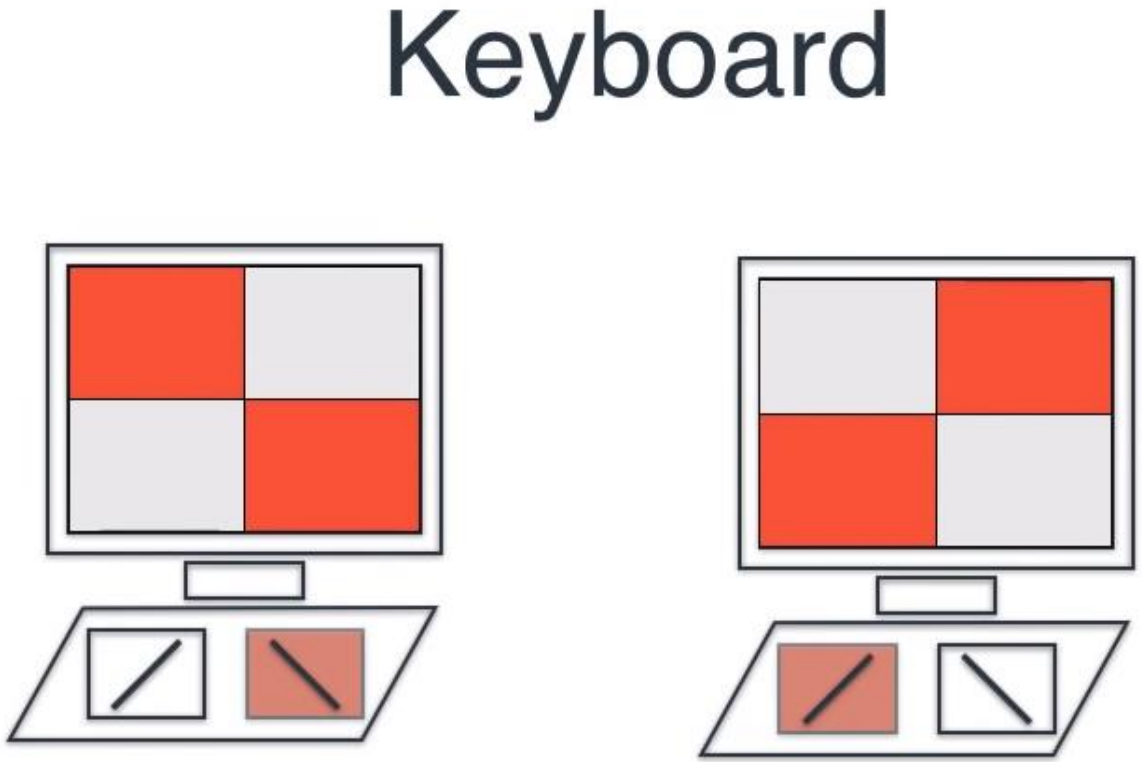


Image recognition software

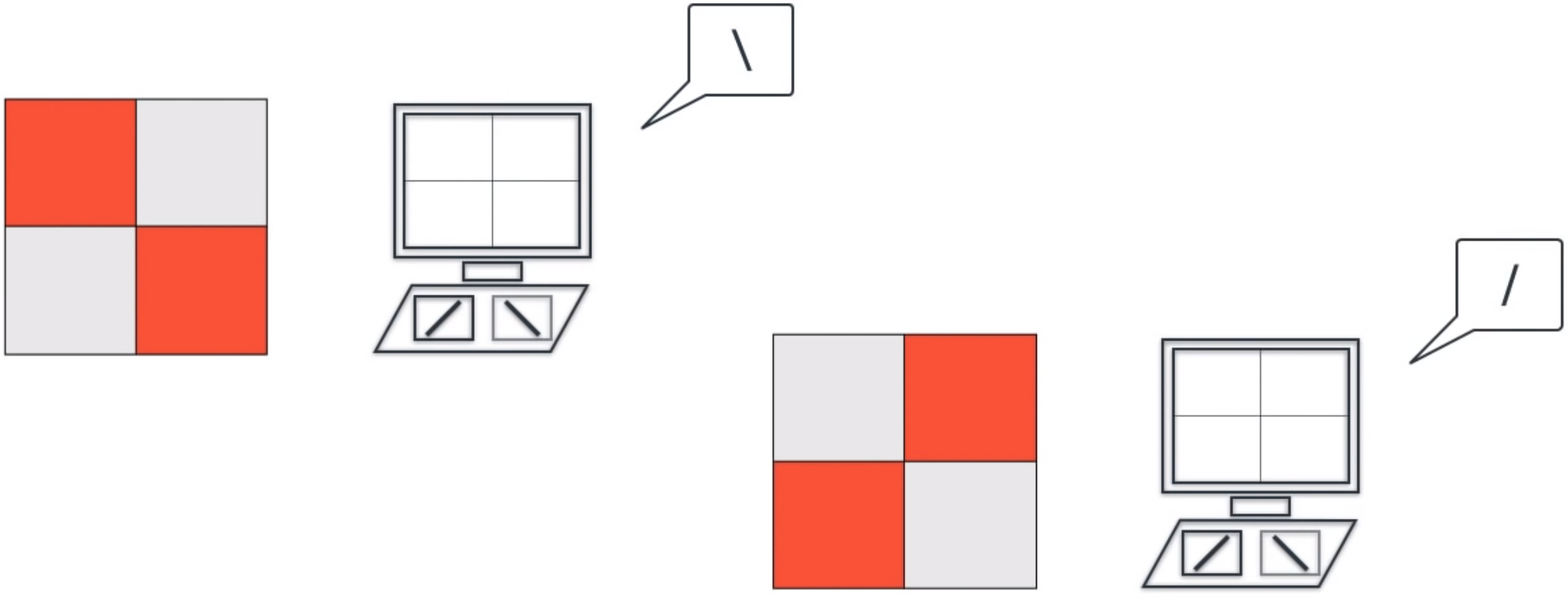
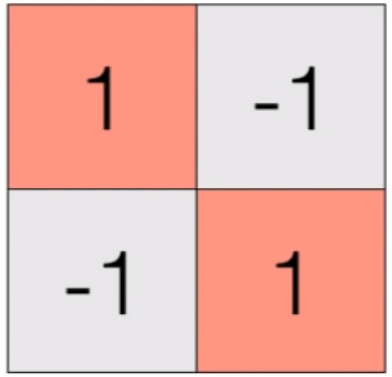


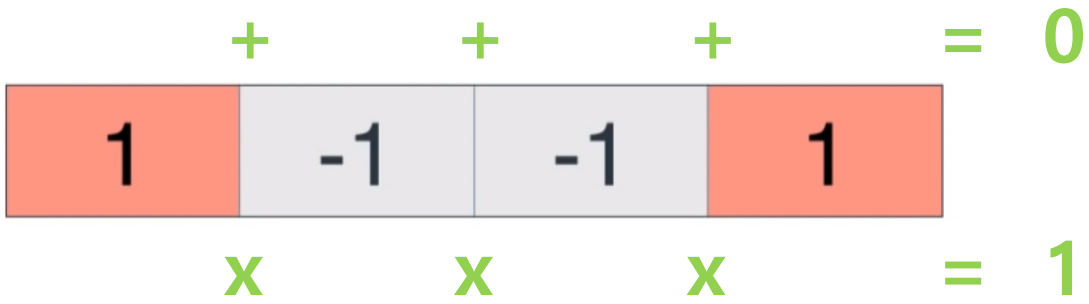
Image recognition software

We have to do some math ematical operations!

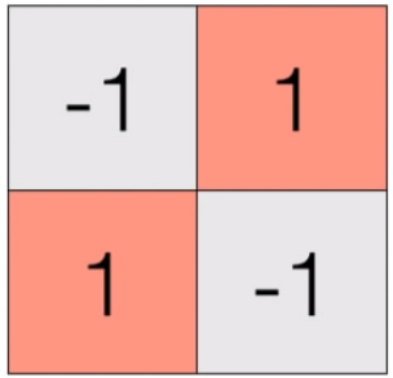
But how?

\





/



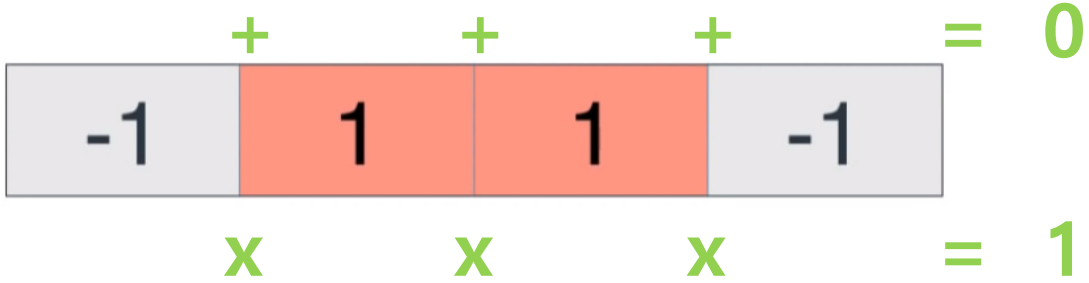


Image recognition software

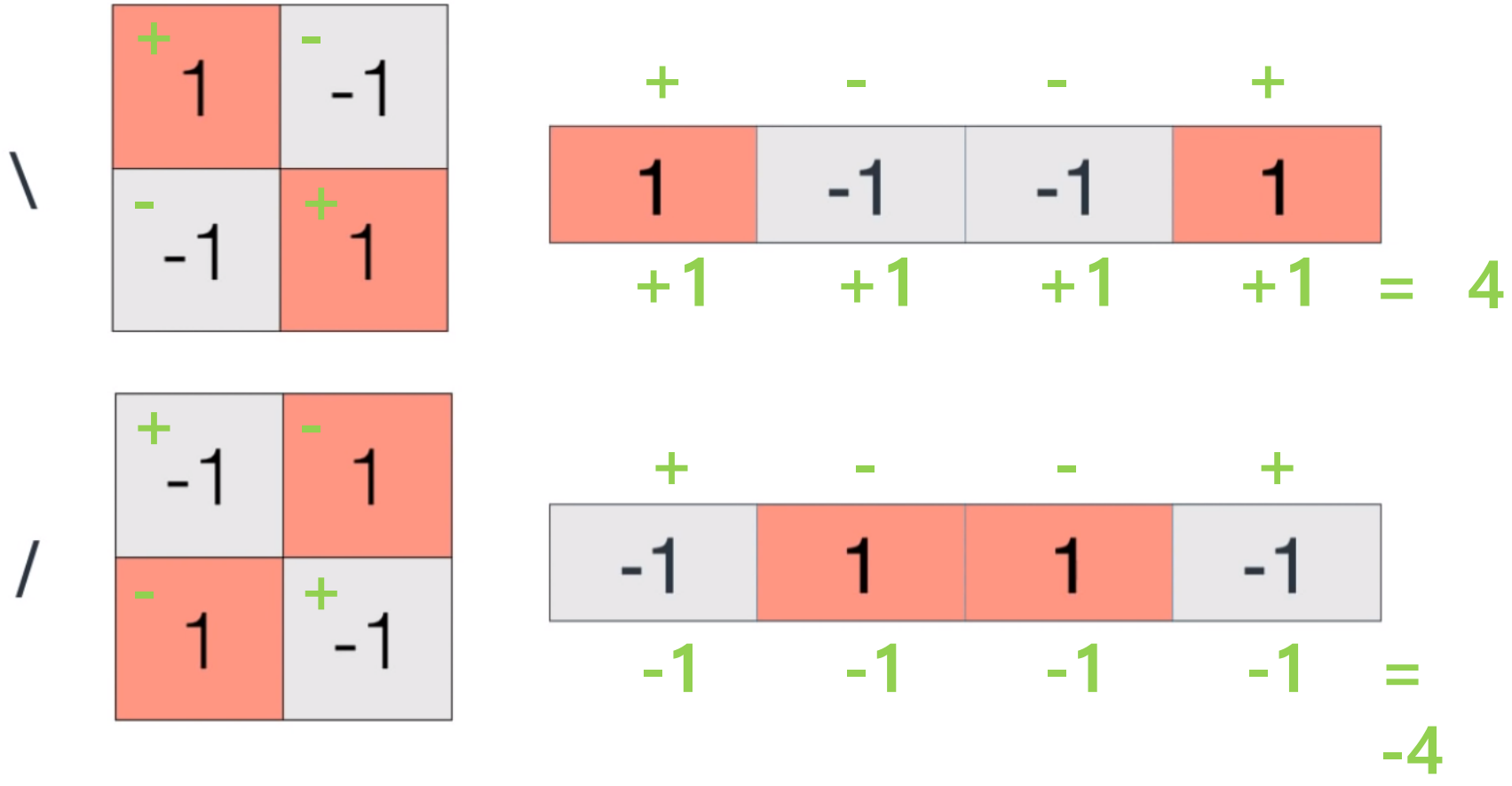
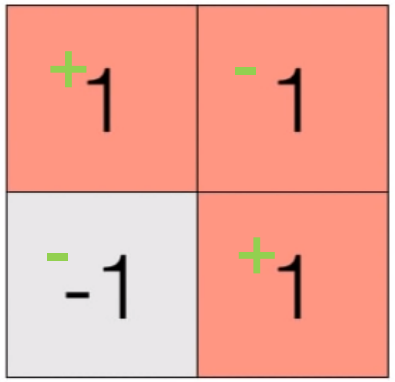
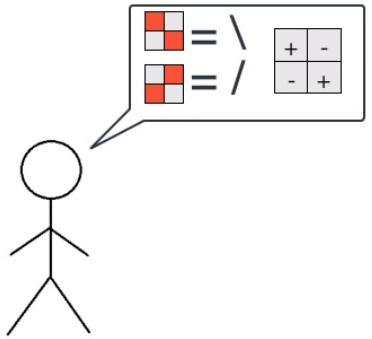


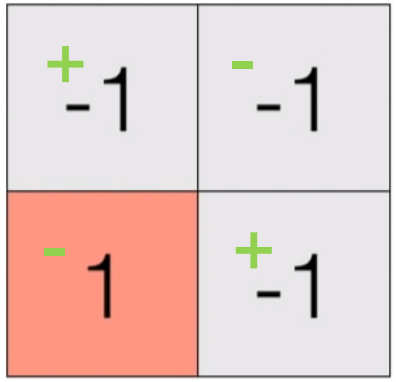
Image recognition software

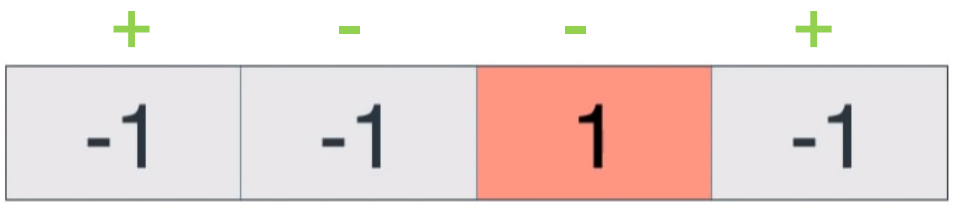
\





/





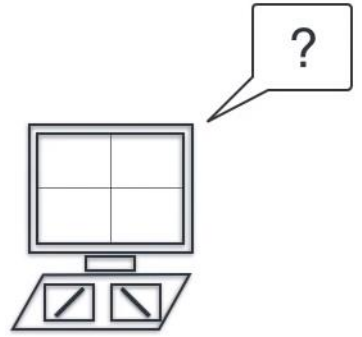
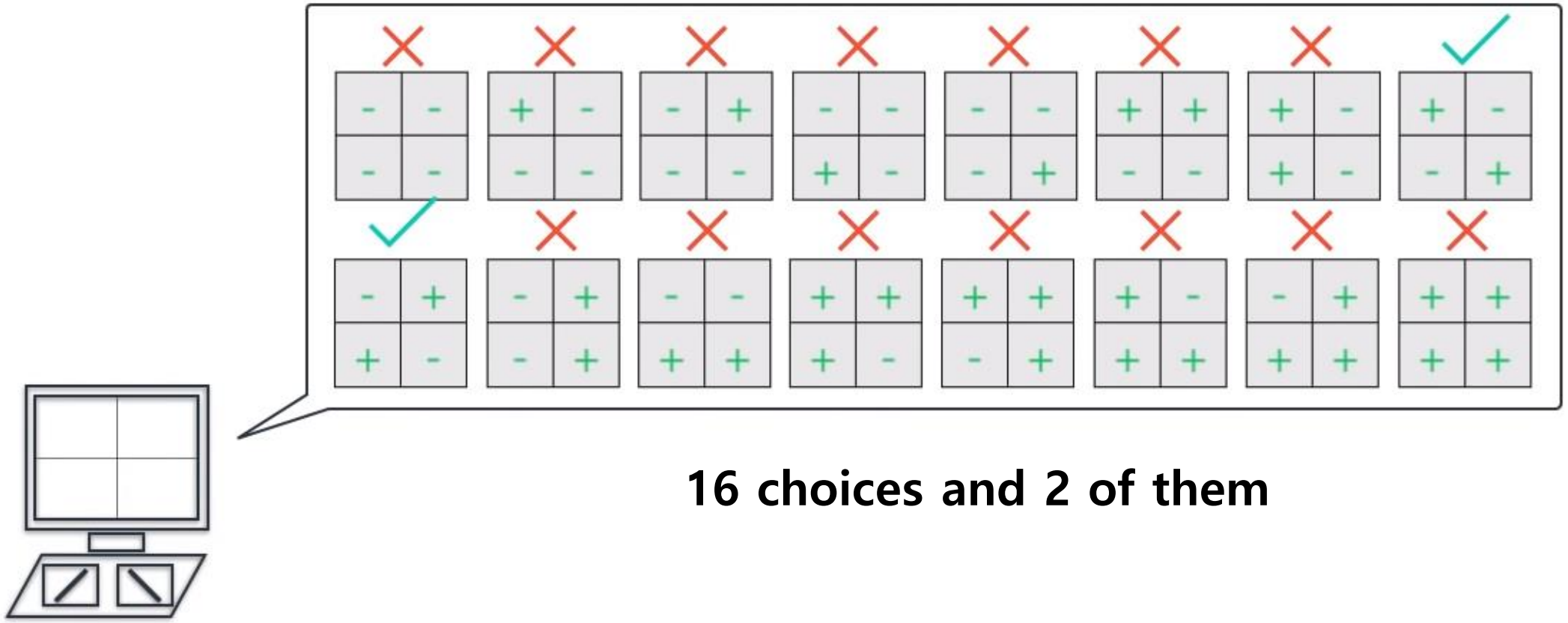


Image recognition software



16 choices and 2 of them

Image recognition software

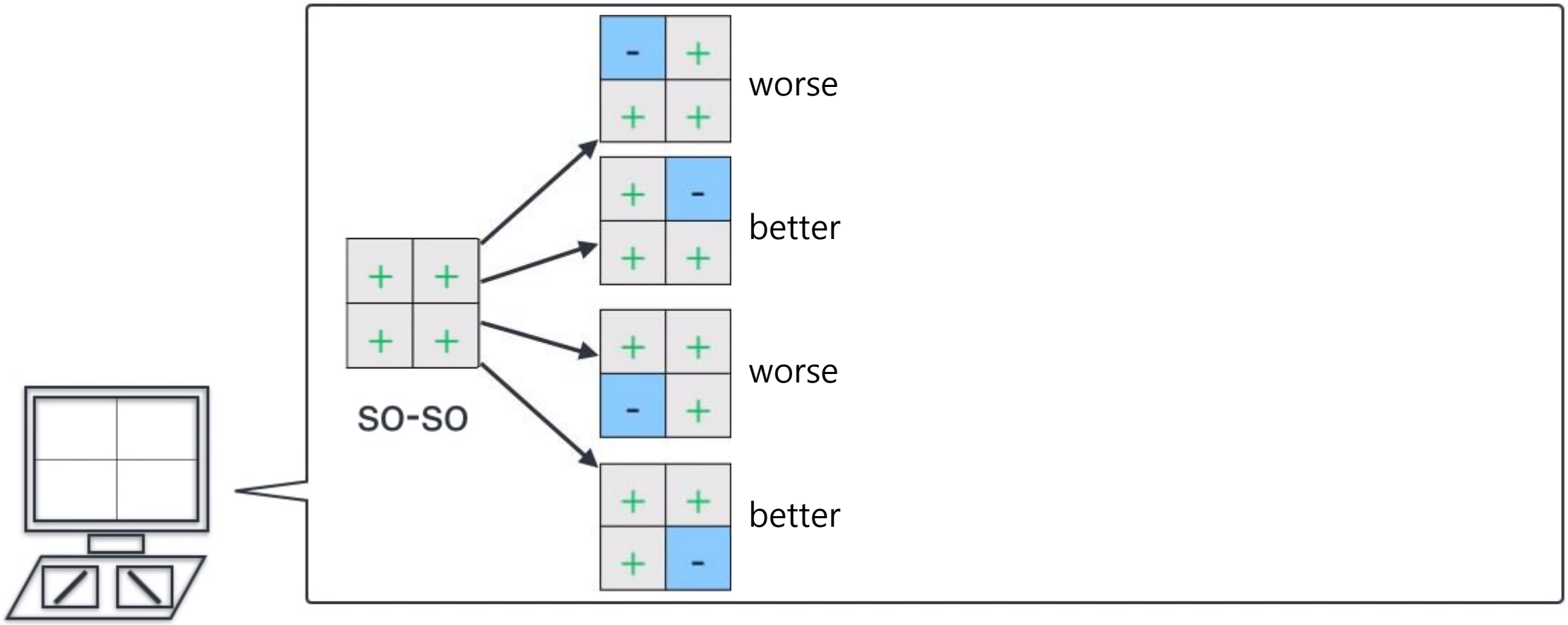
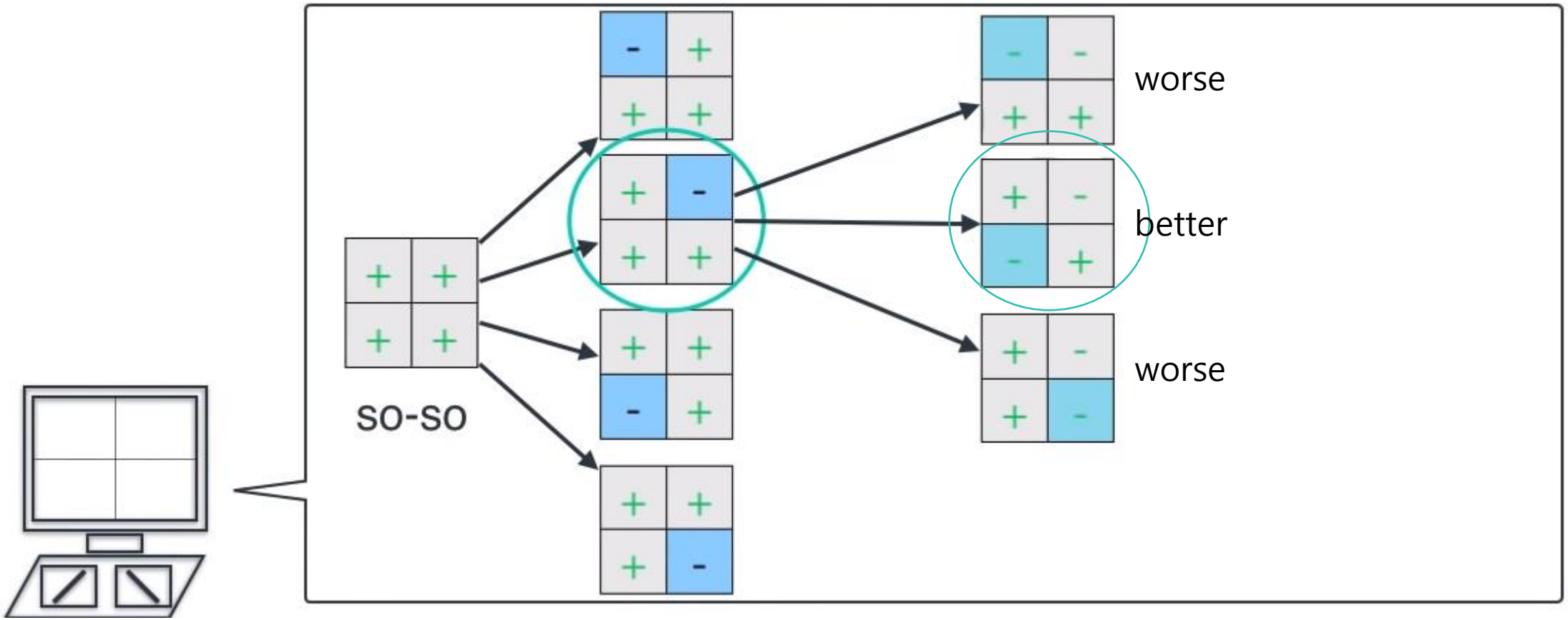
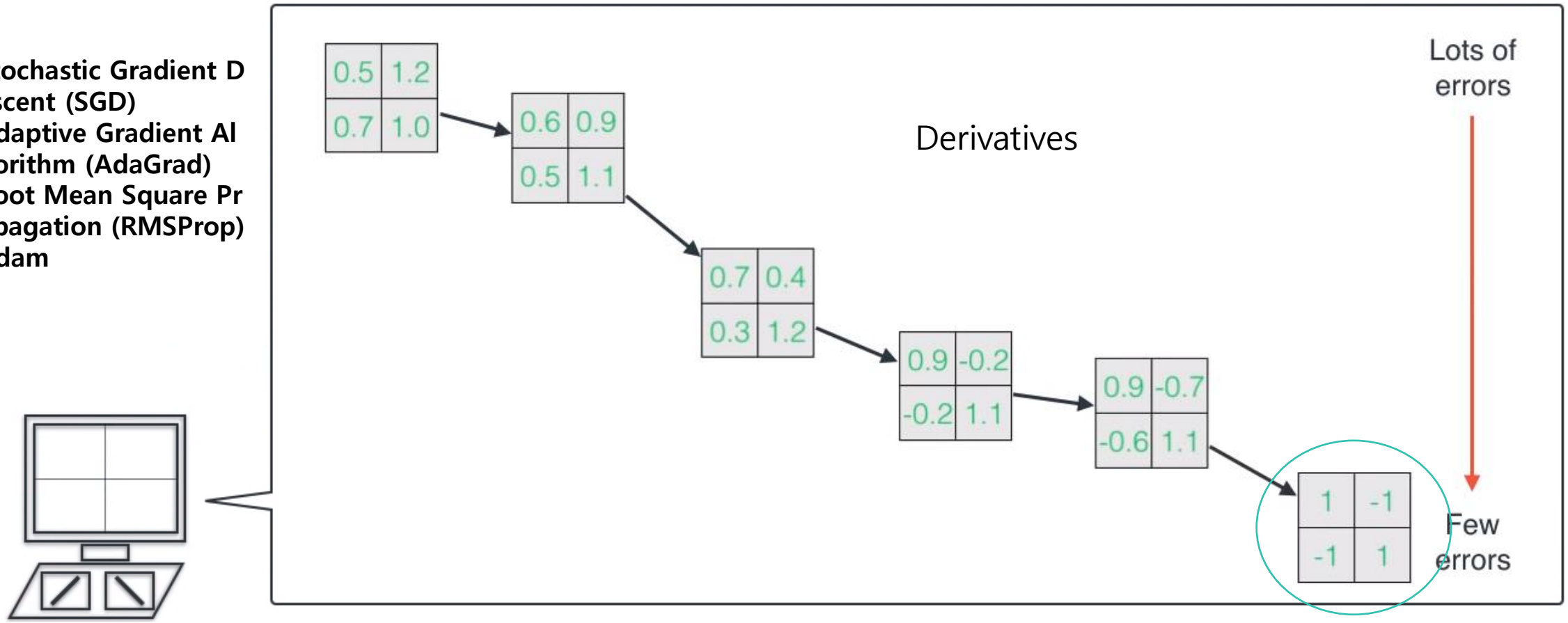


Image recognition software



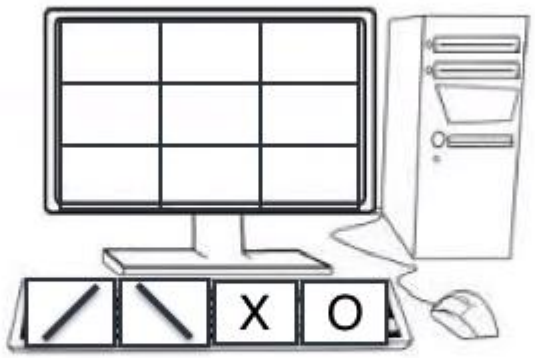
Gradient Descent algorithm

- **Stochastic Gradient D escent (SGD)**
- **Adaptive Gradient Al gorithm (AdaGrad)**
- **Root Mean Square Pr opagation (RMSProp)**
- **Adam**

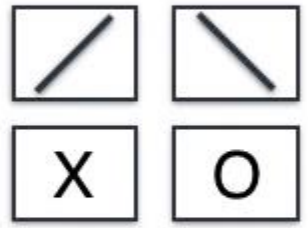


Some mission

Person

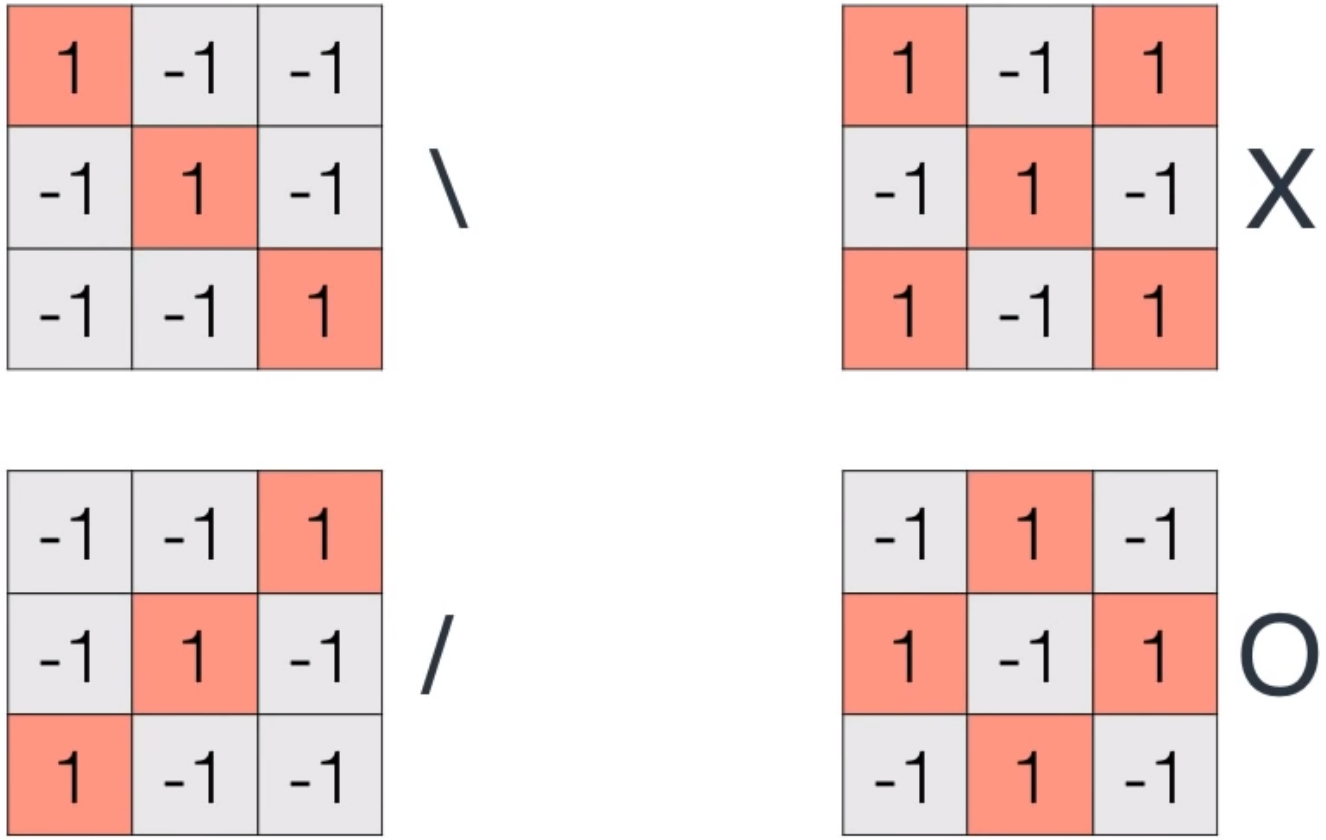


Computer

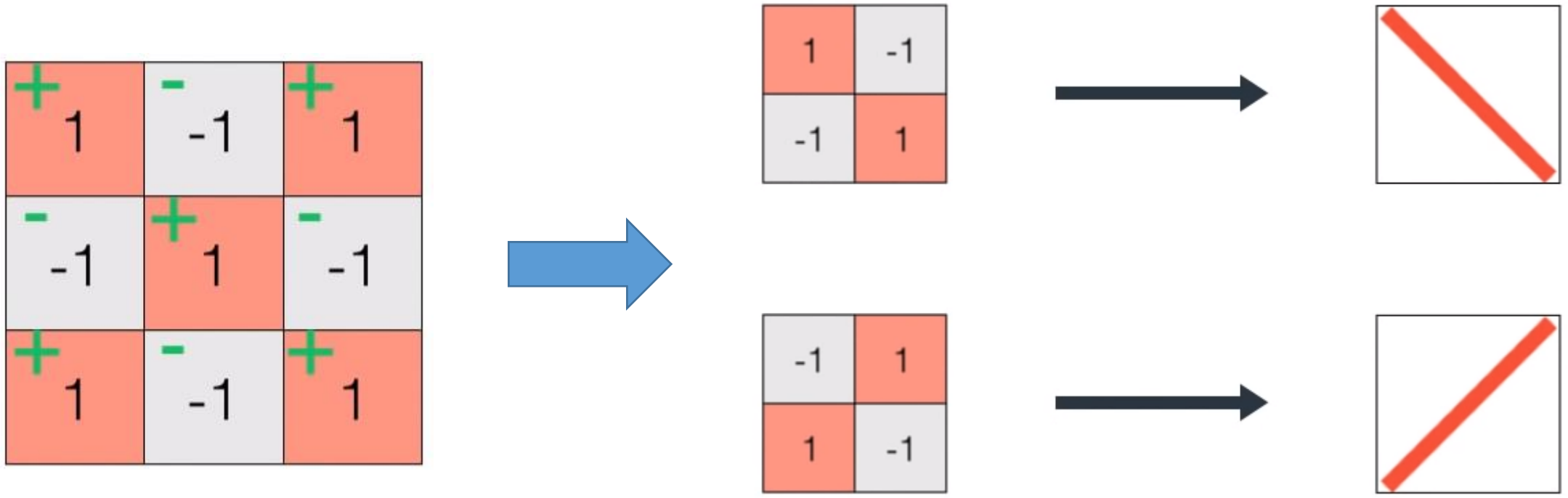


Alphabet

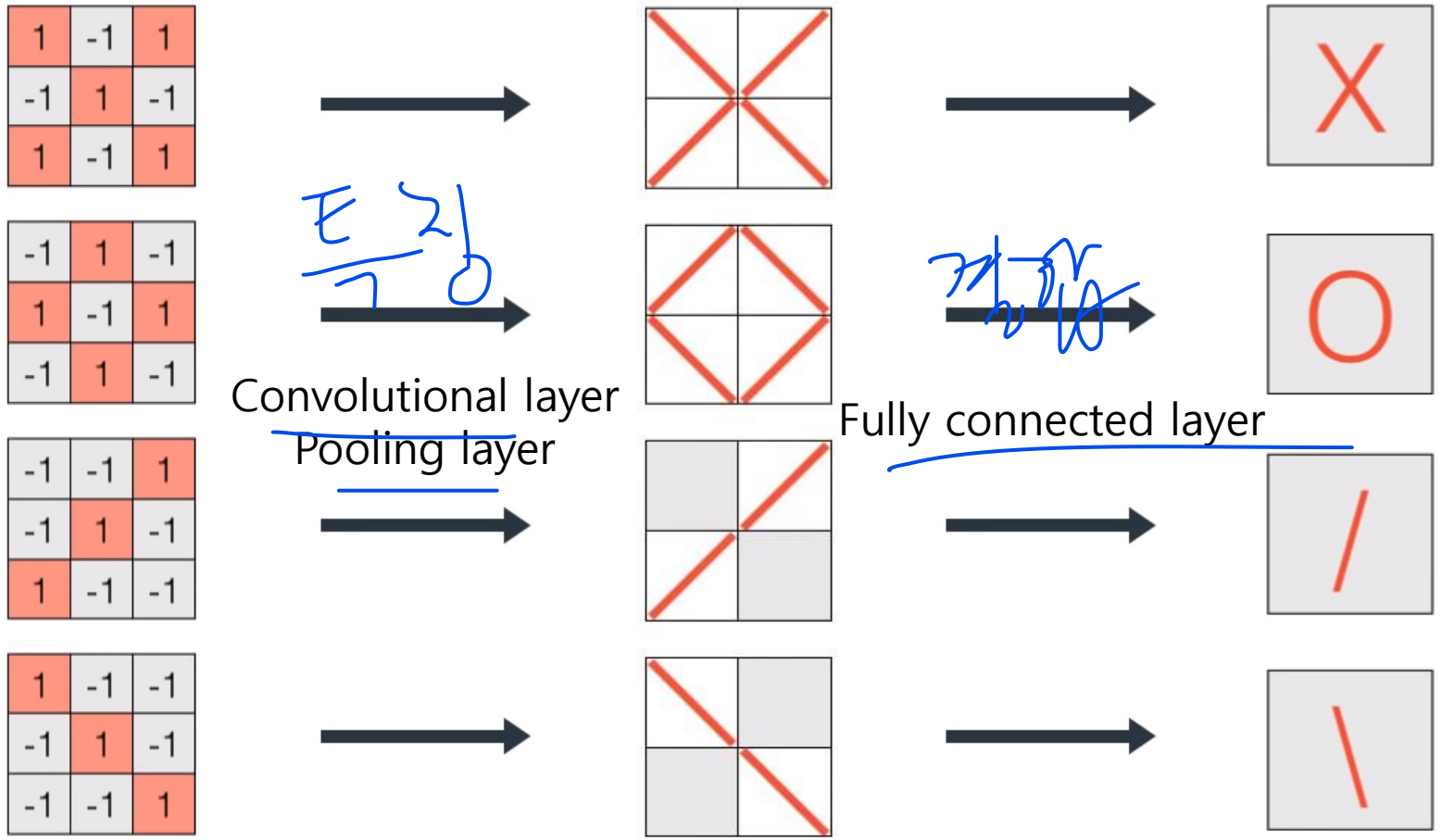
Computer vision



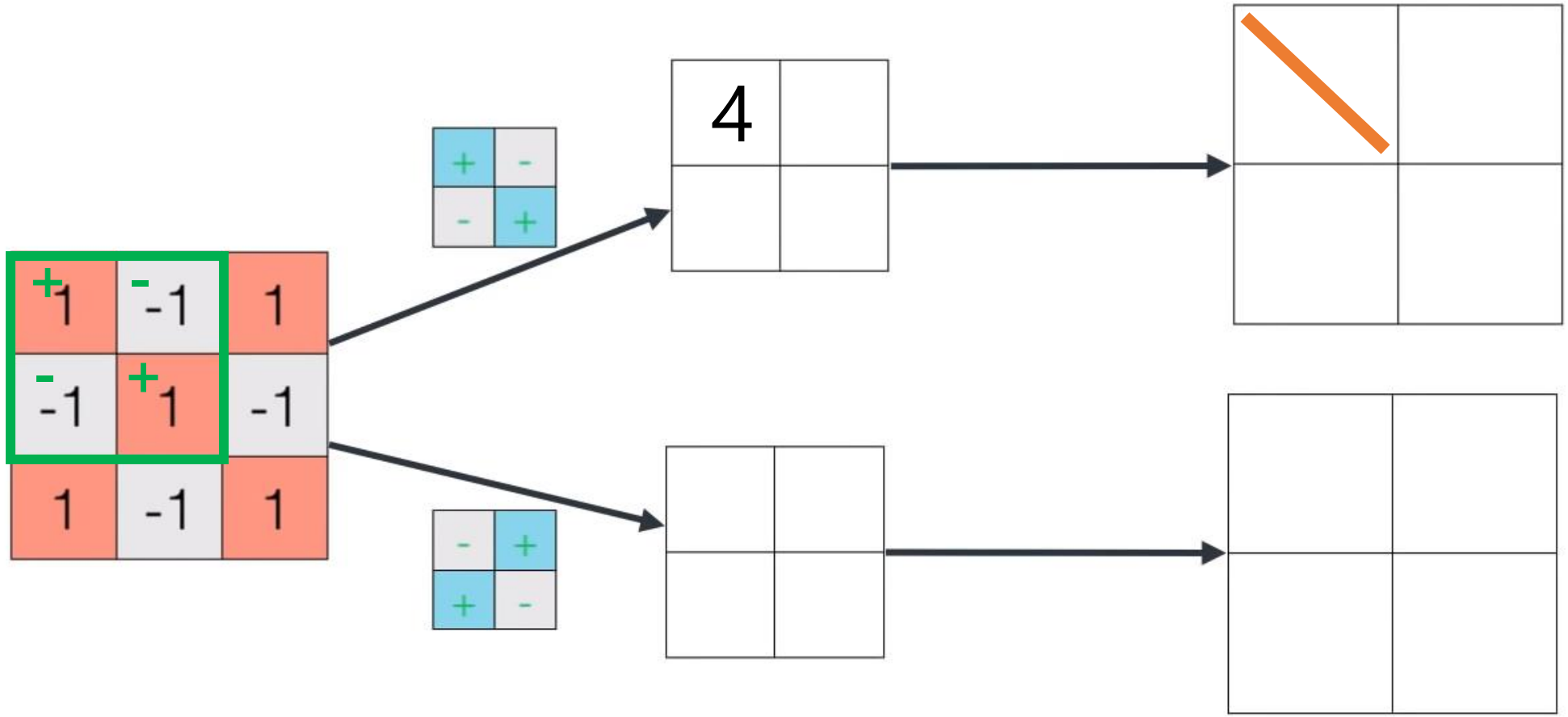
Using previous knowledge



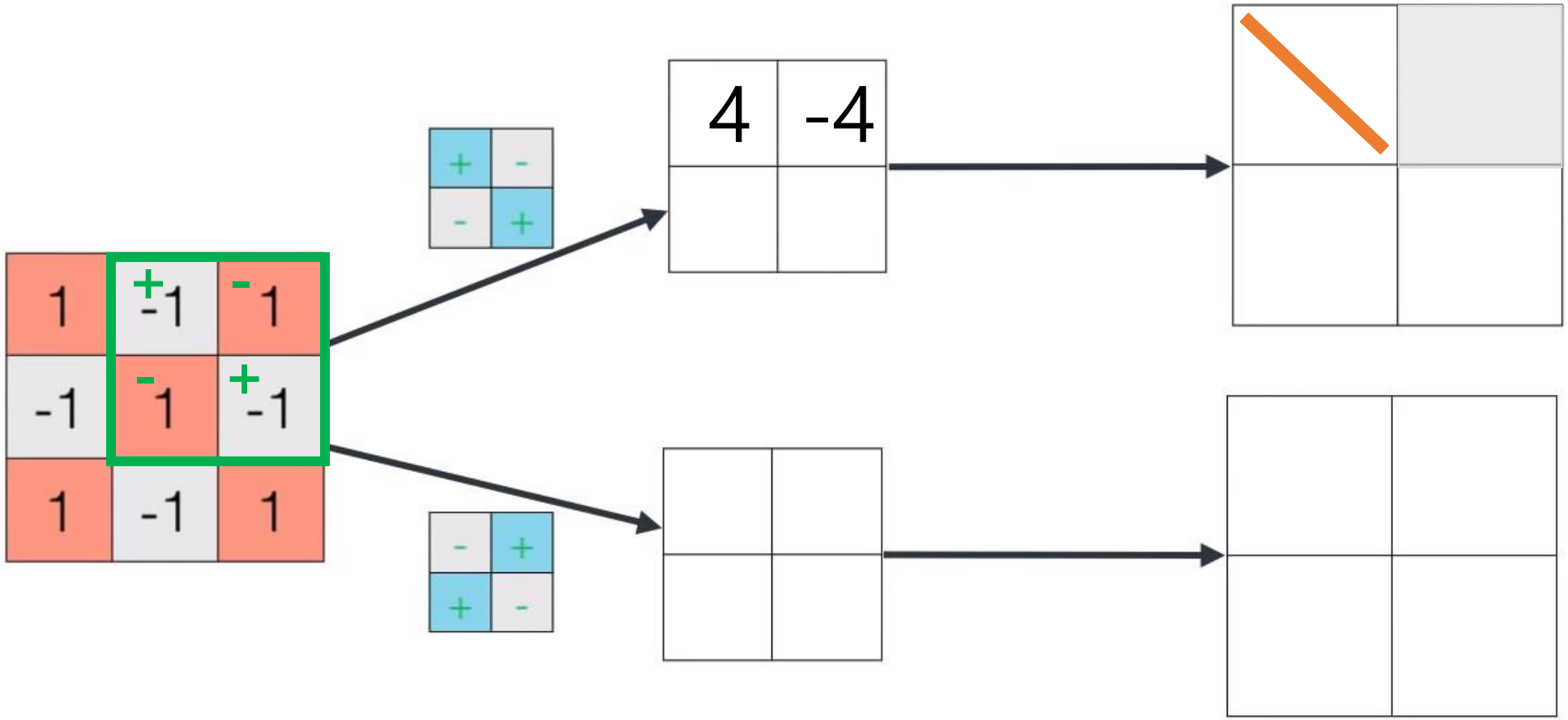
Convolutional Neural Network



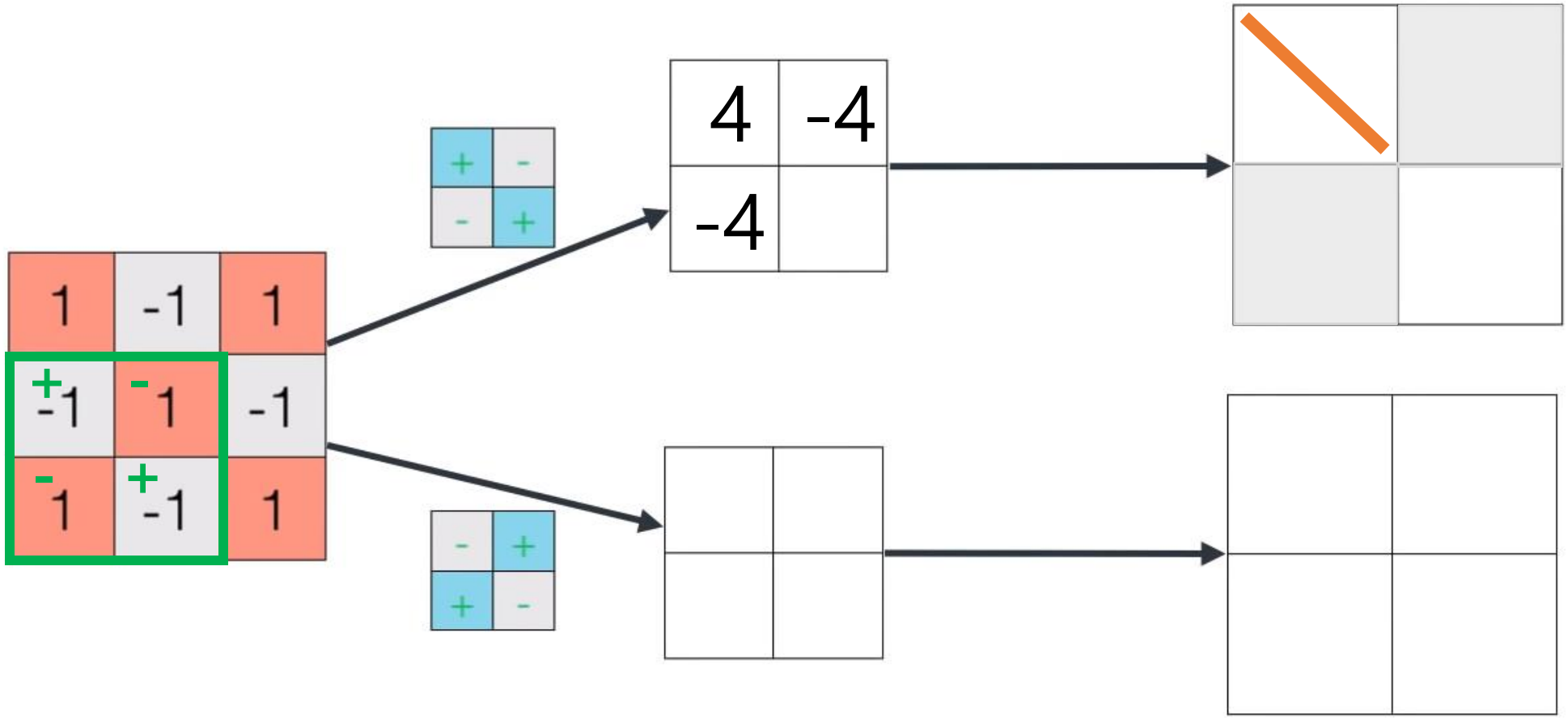
Convolutional Neural Network



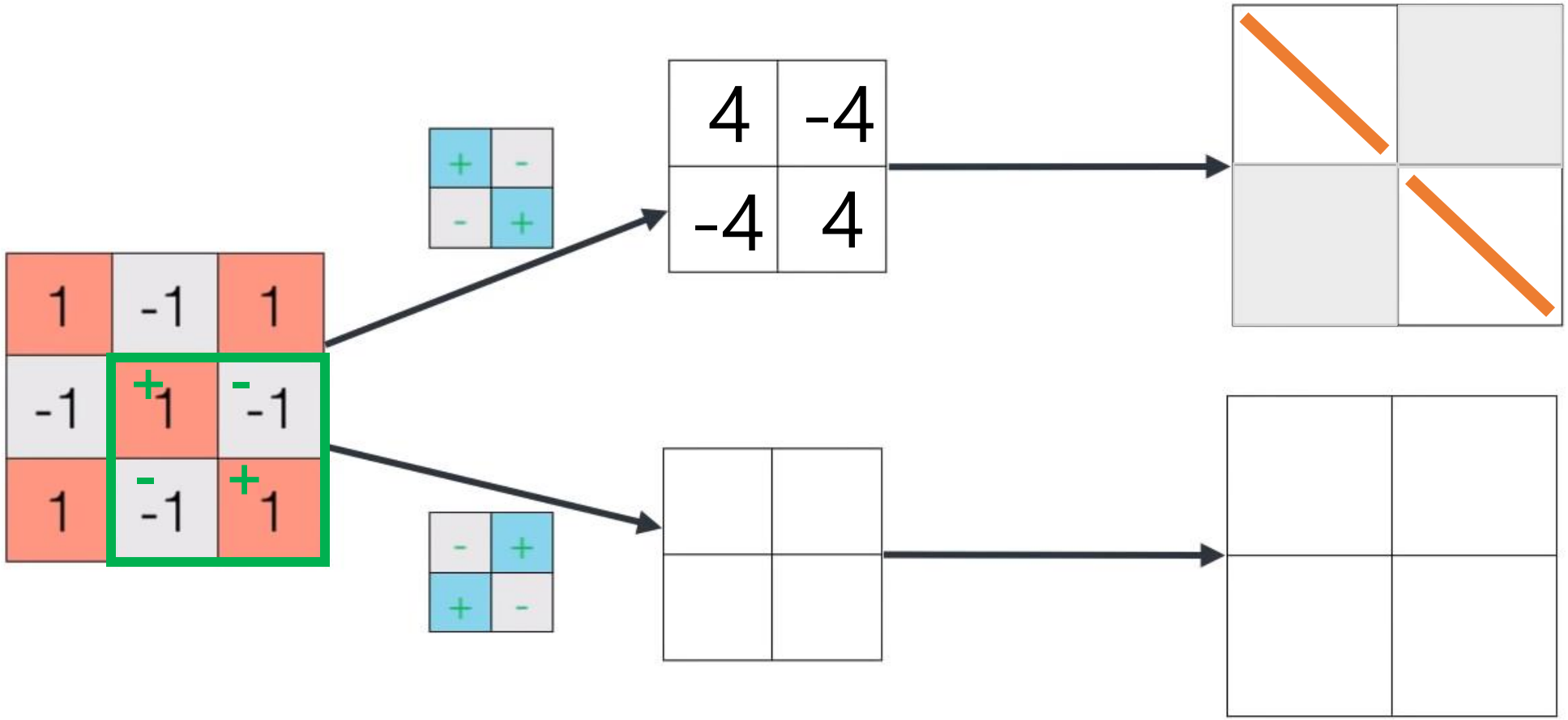
Convolutional Neural Network



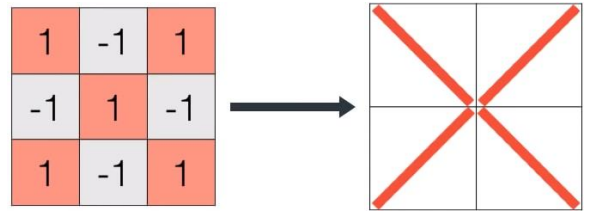
Convolutional Neural Network

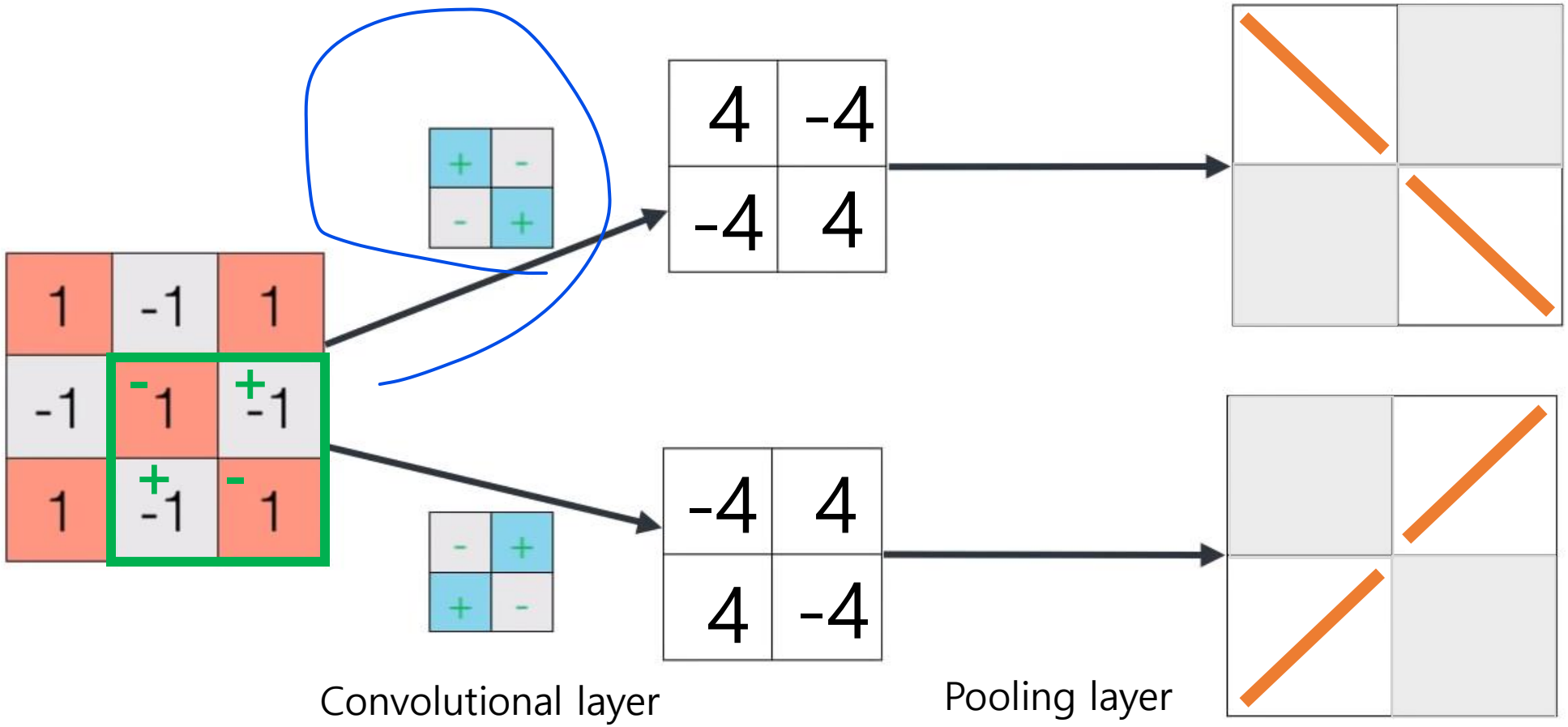


Convolutional Neural Network

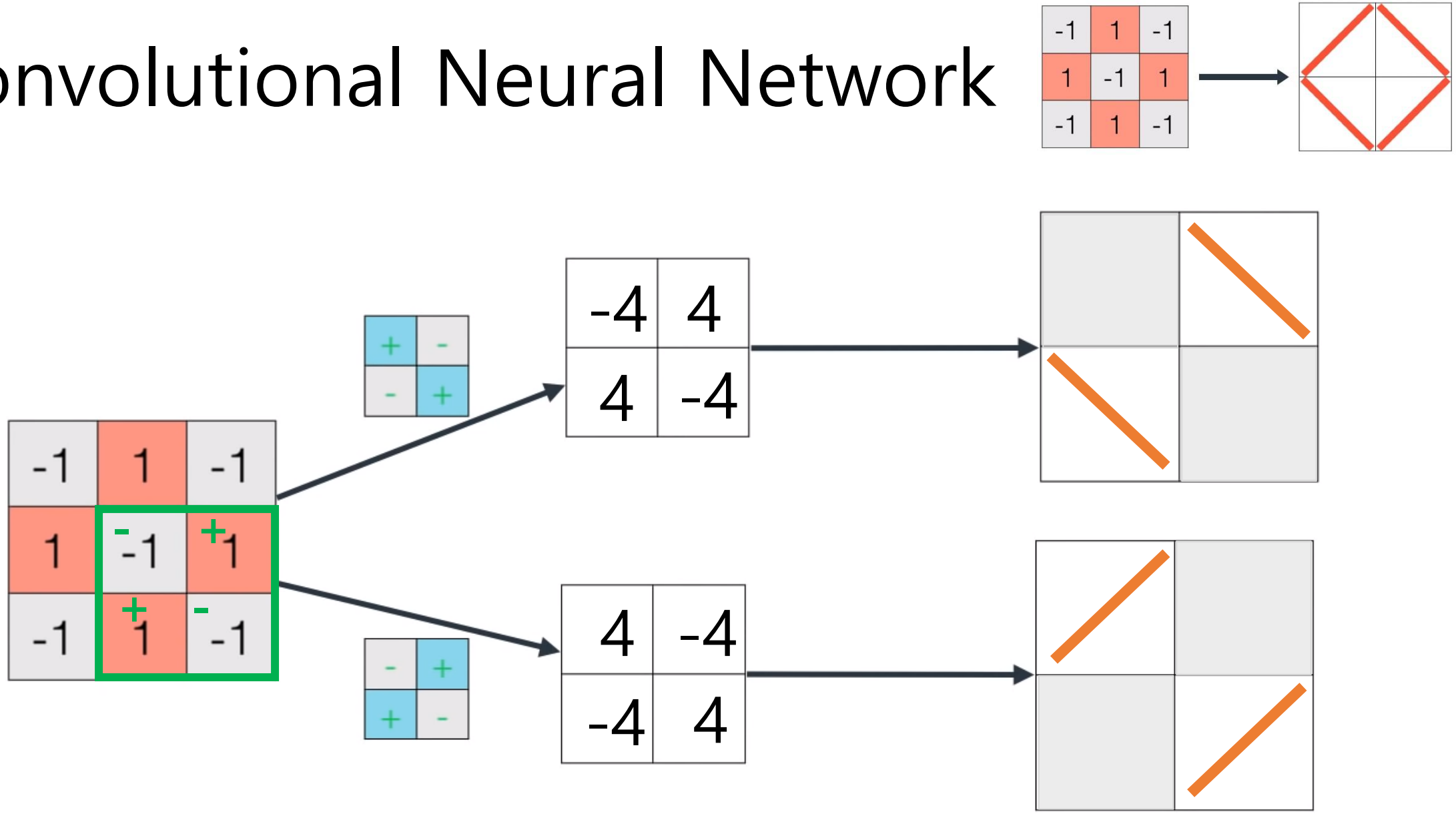


Convolutional Neural Network

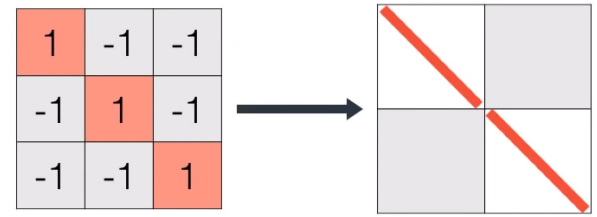


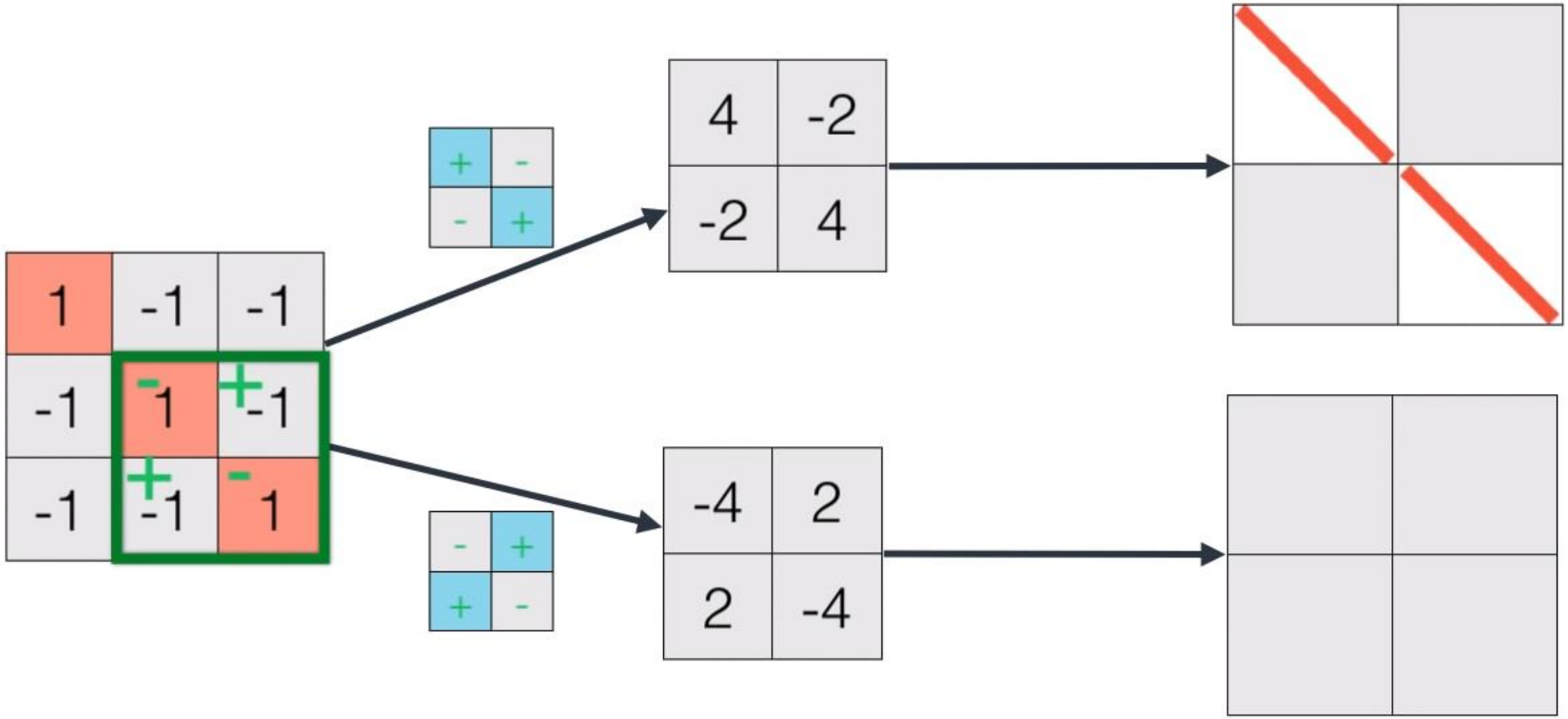


Convolutional Neural Network

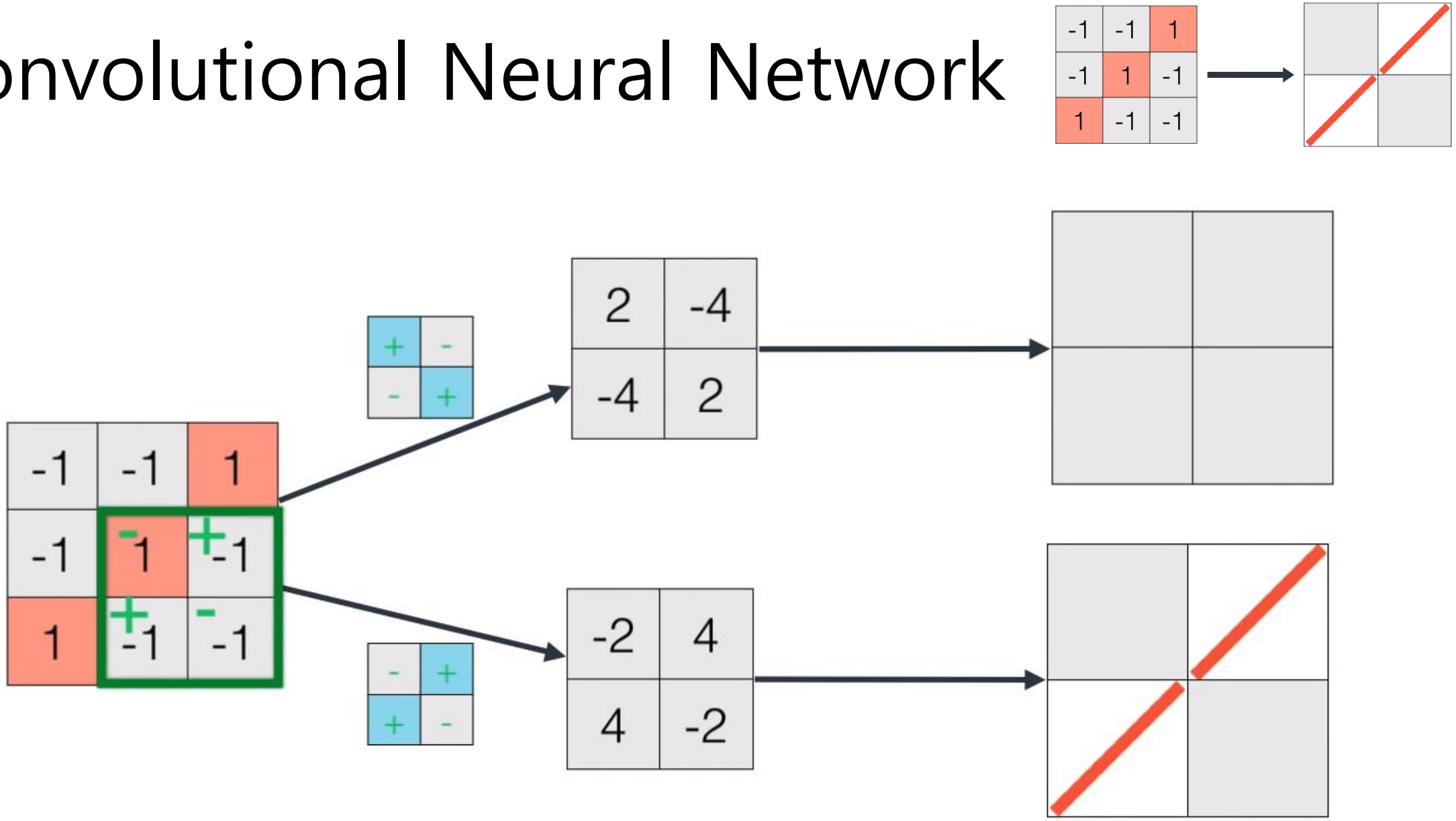


Convolutional Neural Network

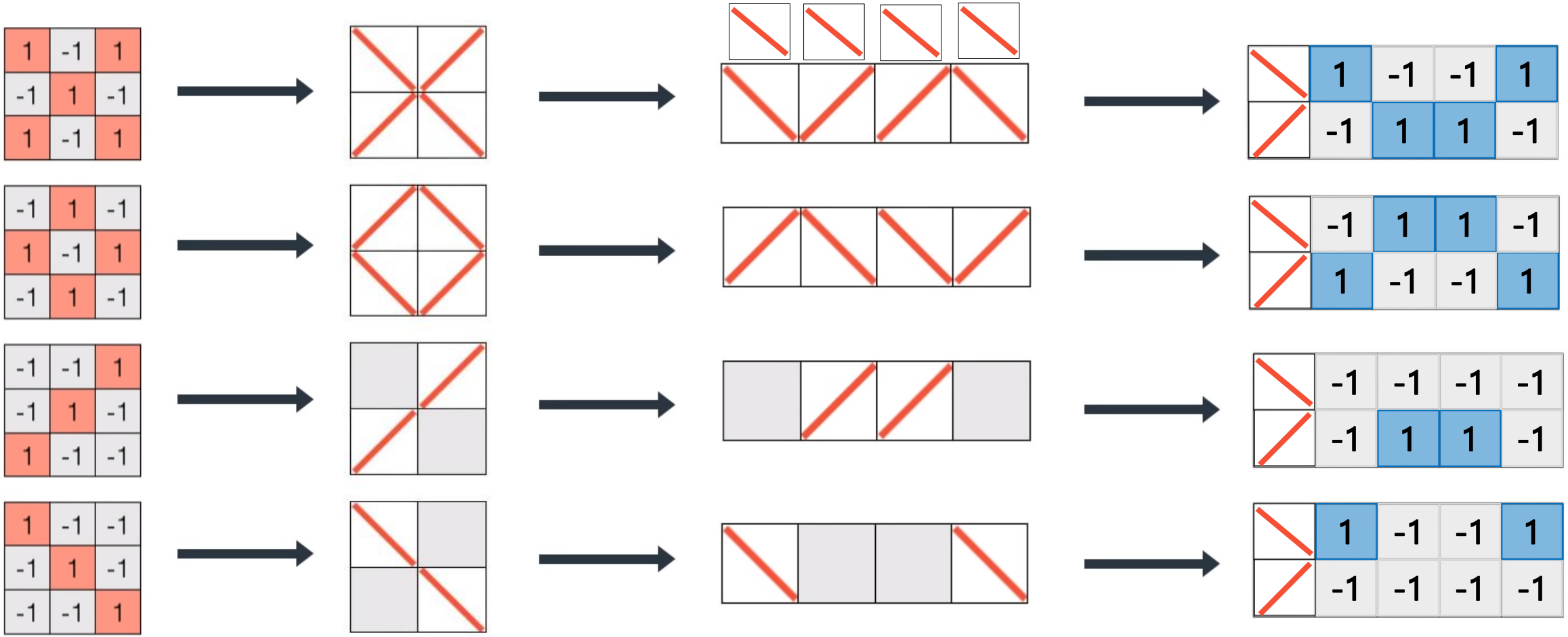




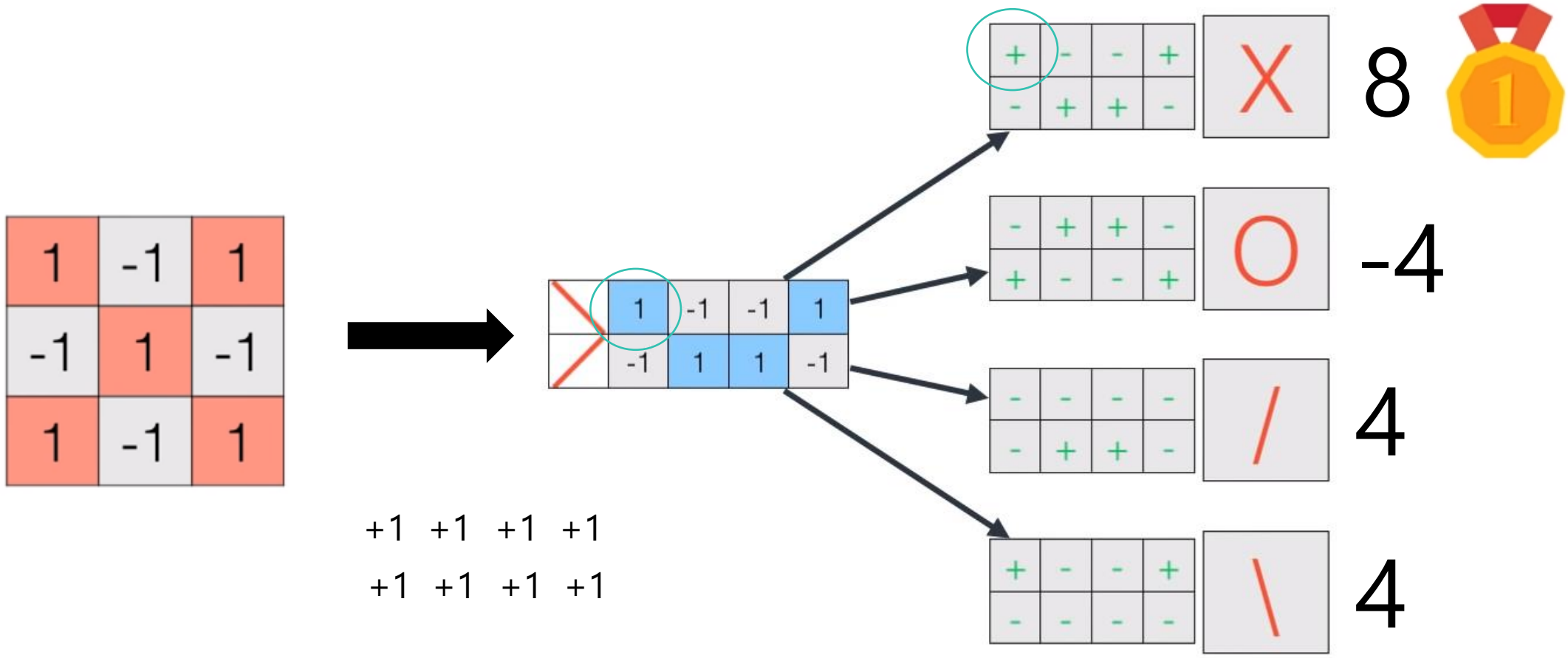
Convolutional Neural Network



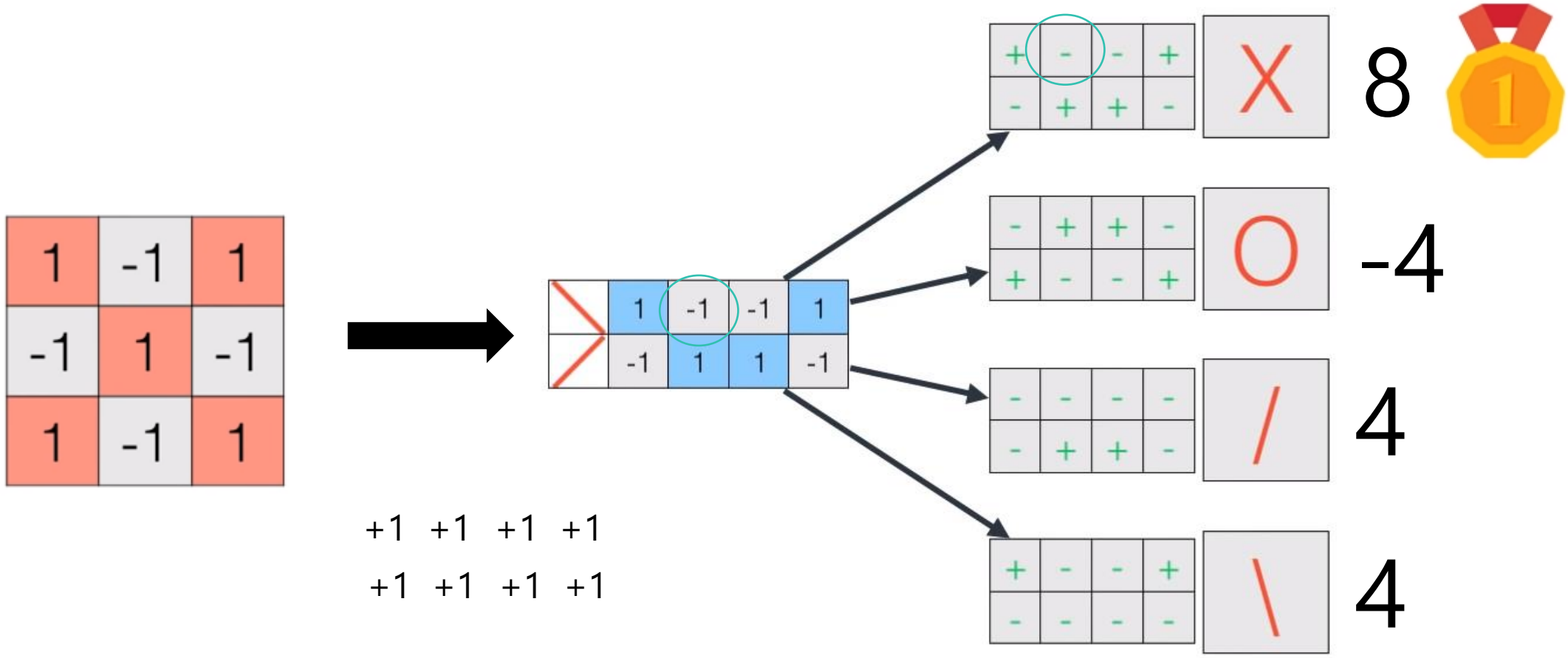
Summaries our results



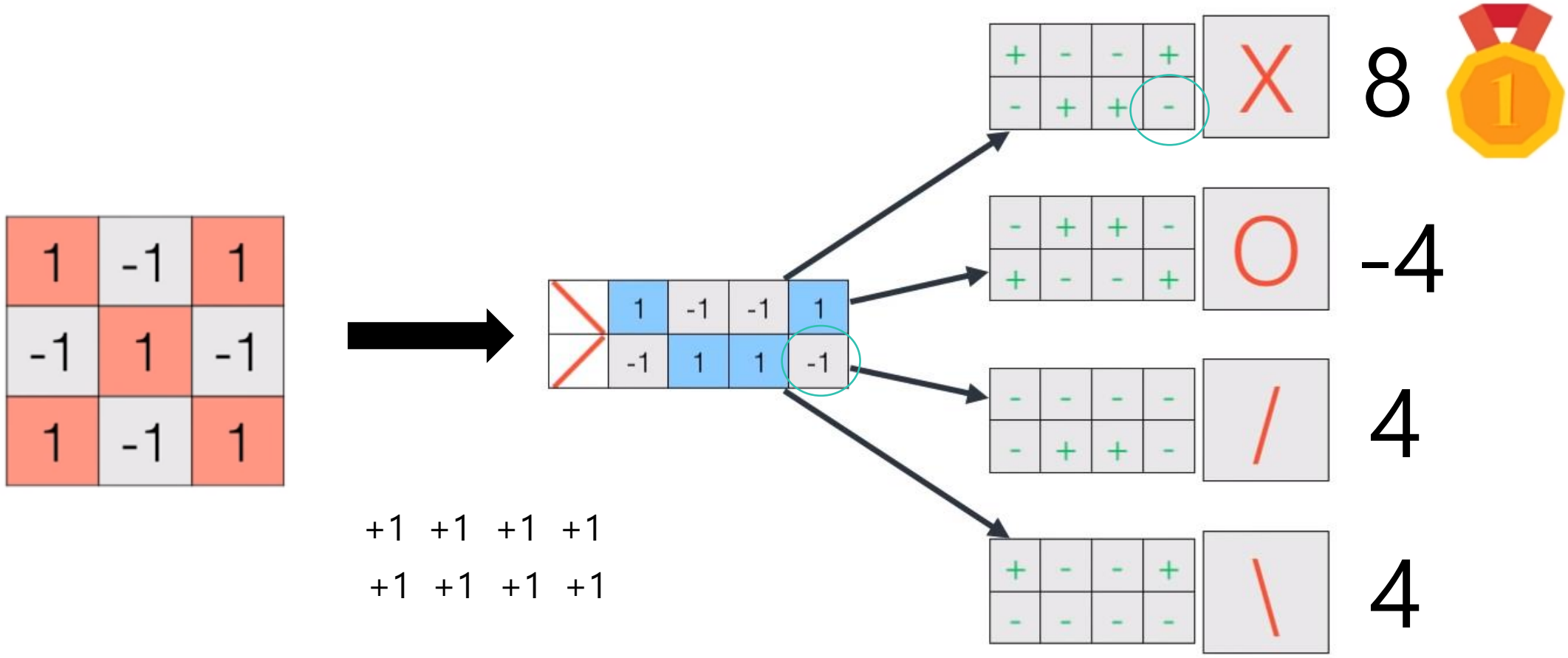
Prediction



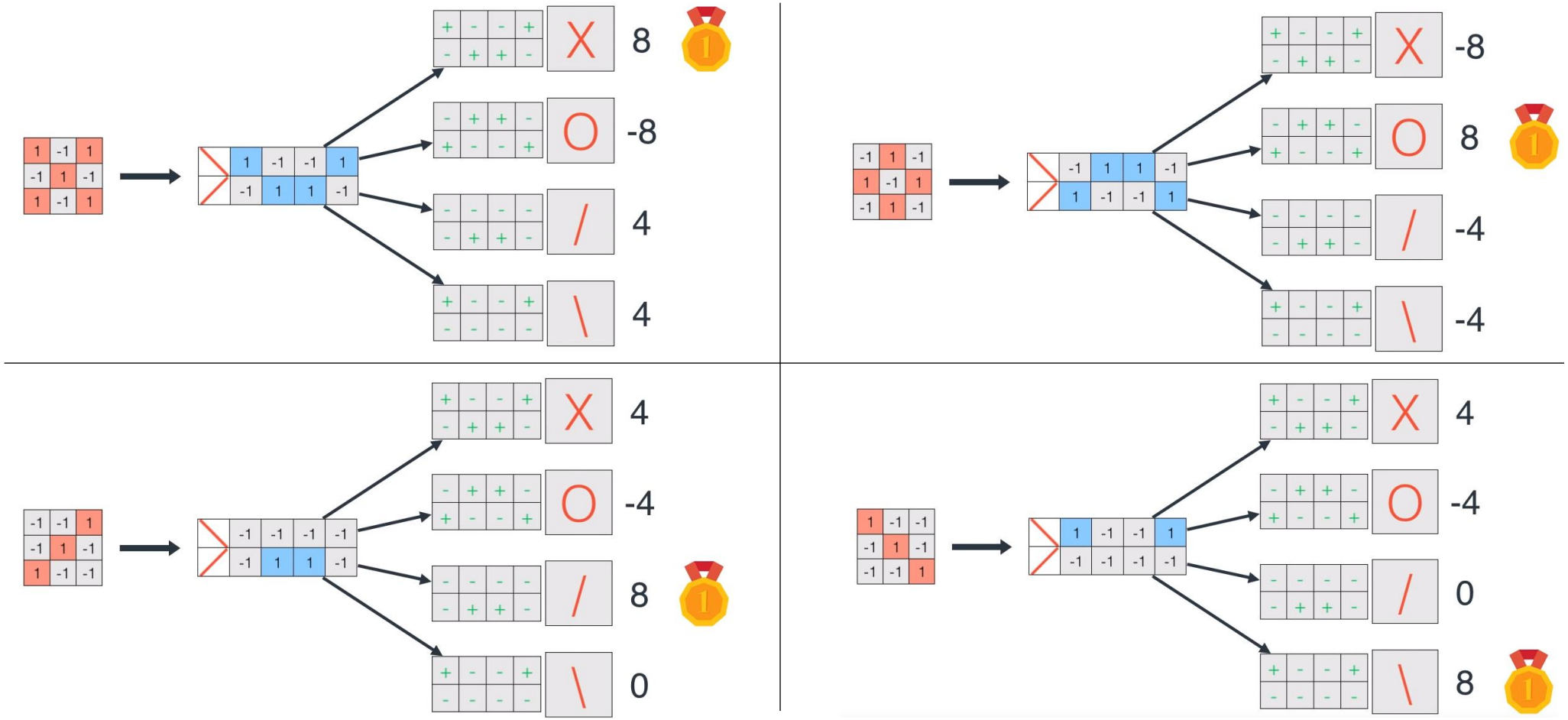
Prediction

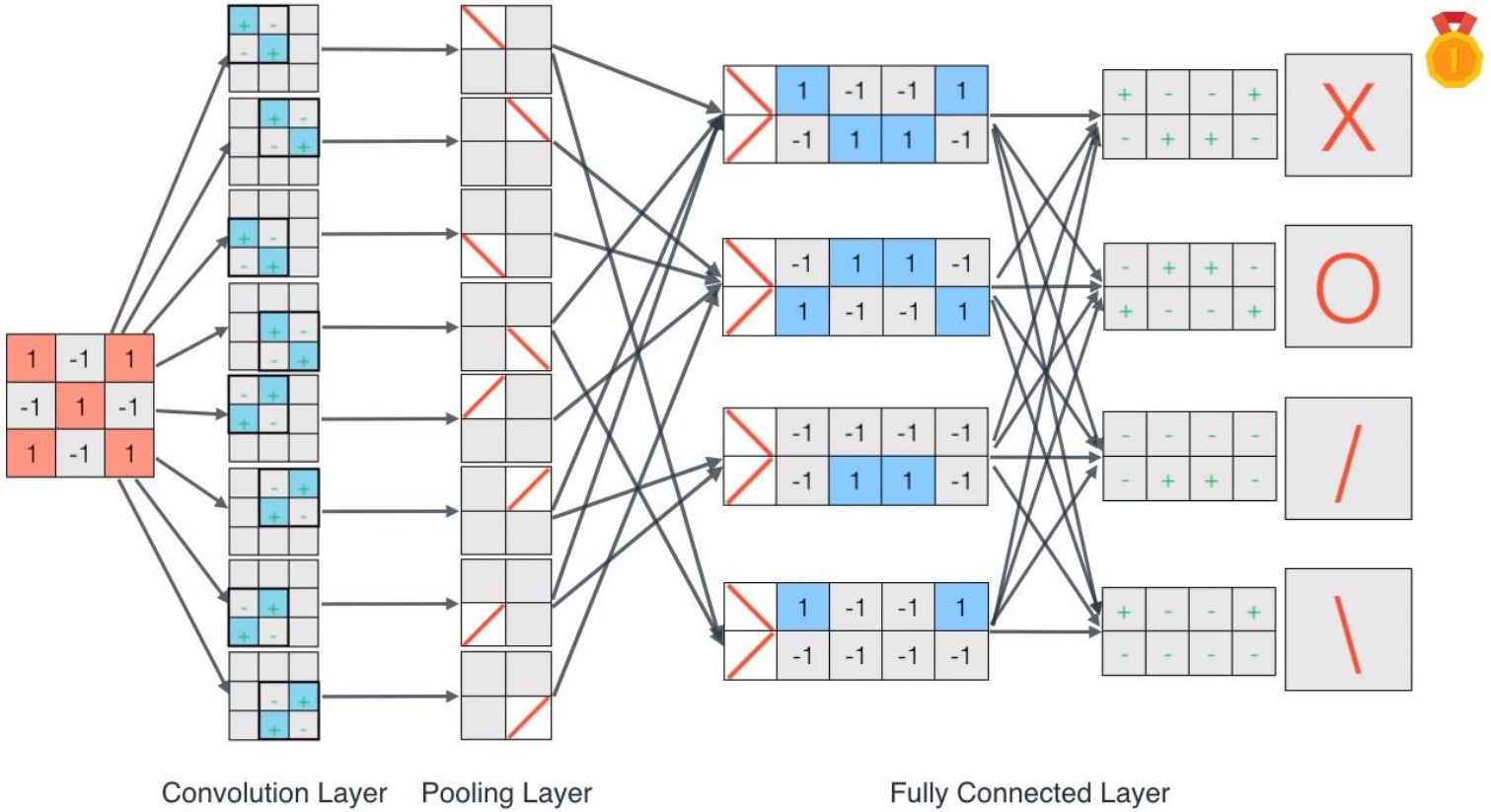


Prediction



Prediction





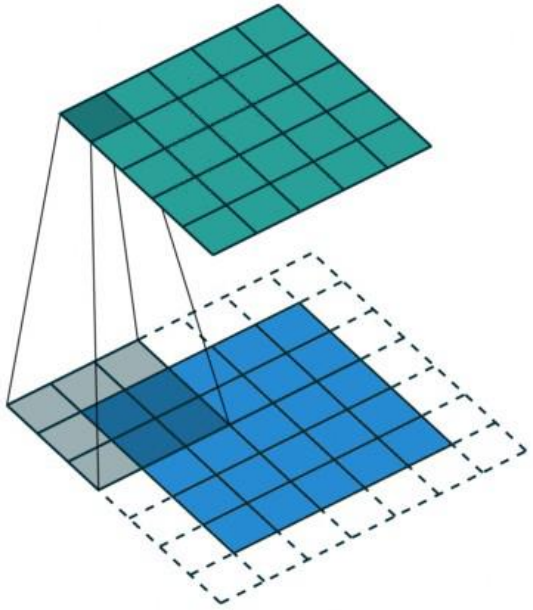
Conclusion

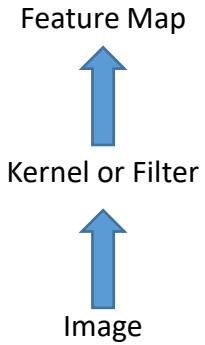
- Full Convolutional Neural Network:
  - Convolutional layer
  - Pooling layer
  - Fully connected layer

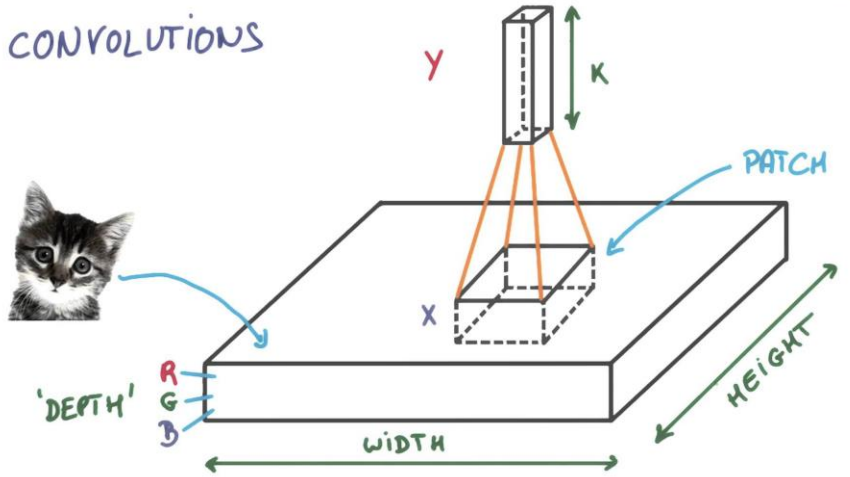
* https://www.youtube.com/channel/UCgBncpylJ1kiVaPyP-PZauQ

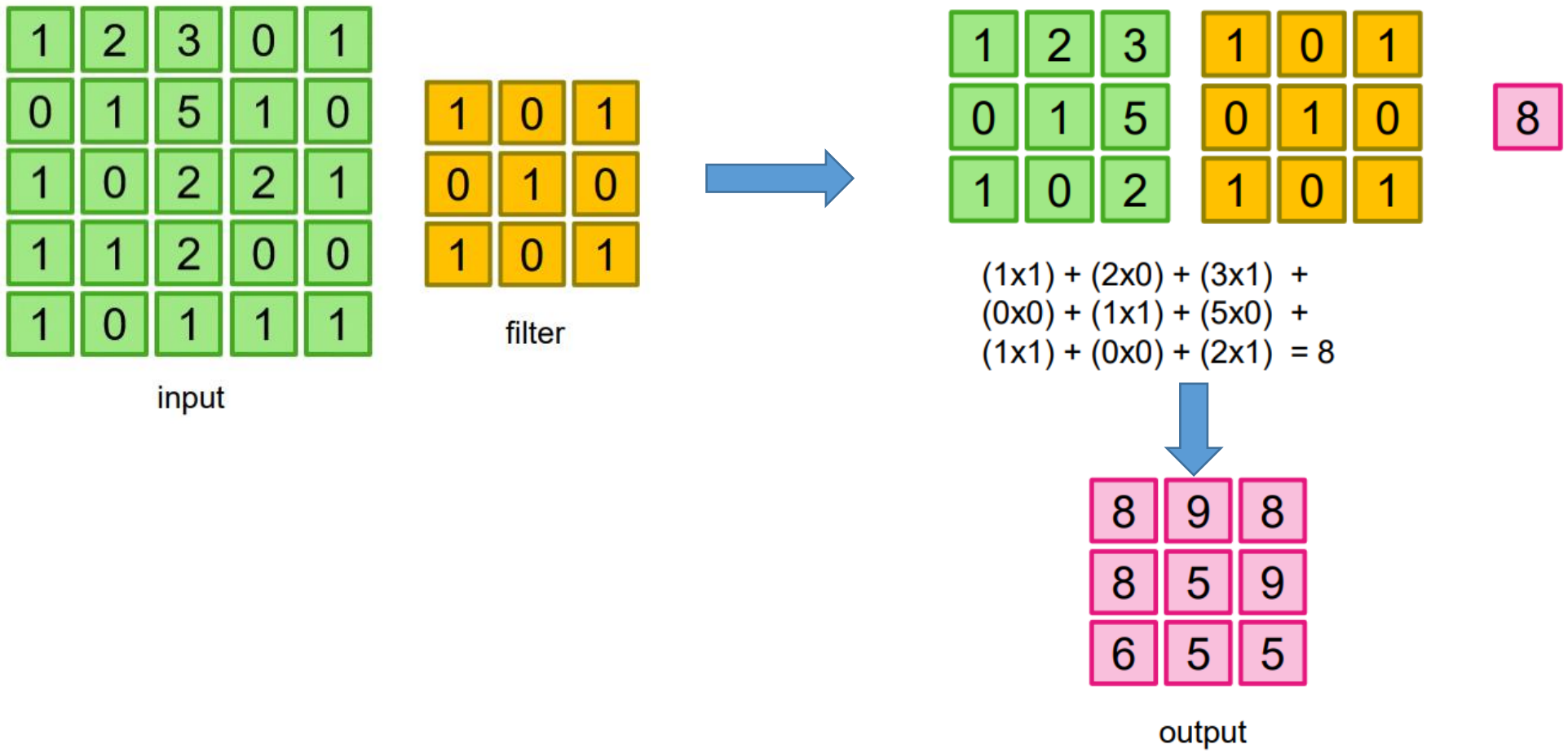
What is Convolution?

- 이미지 위에서 stride 값 만큼 filter(kernel)을 이동시키면서 겹쳐지는 부분의 각 원소의 값을 곱해서 모두 더한 값을 출력으로 하는 연산



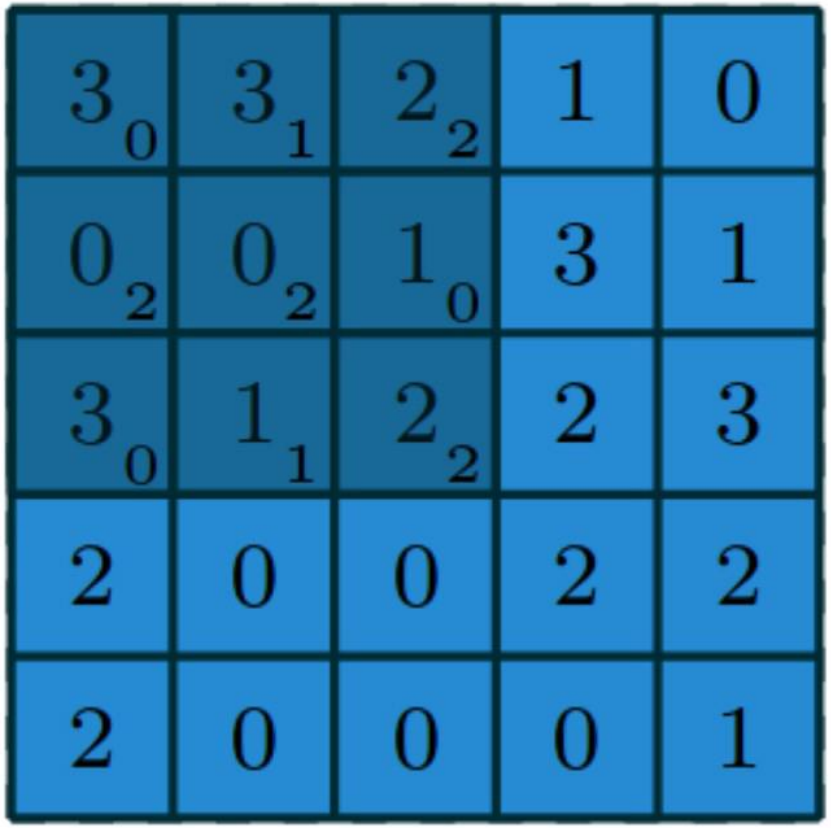





1 2 3 0 1
0 1 5 1 0
1 0 2 2 1
1 1 2 0 0
1 0 1 1 1
input
1 0 1
0 1 0
1 0 1
filter
1 2 3
0 1 5
1 0 2
1 0 1
0 1 0
1 0 1
8
(1x1) + (2x0) + (3x1) +
(0x0) + (1x1) + (5x0) +
(1x1) + (0x0) + (2x1) = 8
8 9 8
8 5 9
6 5 5
output

What is Convolution?

Image

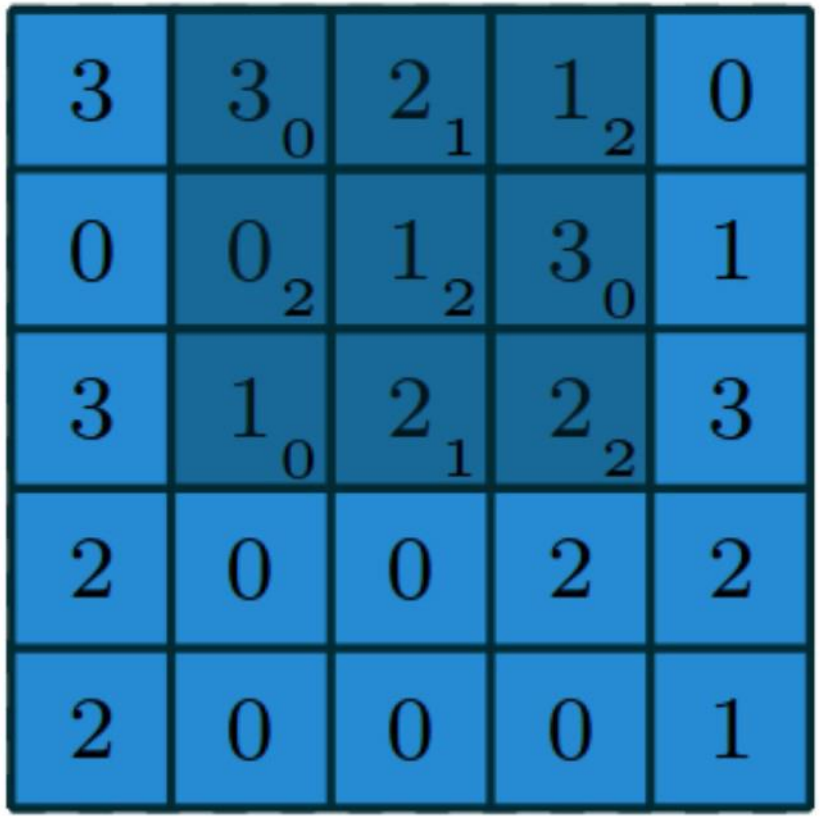


Feature map

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

What is Convolution?

Image

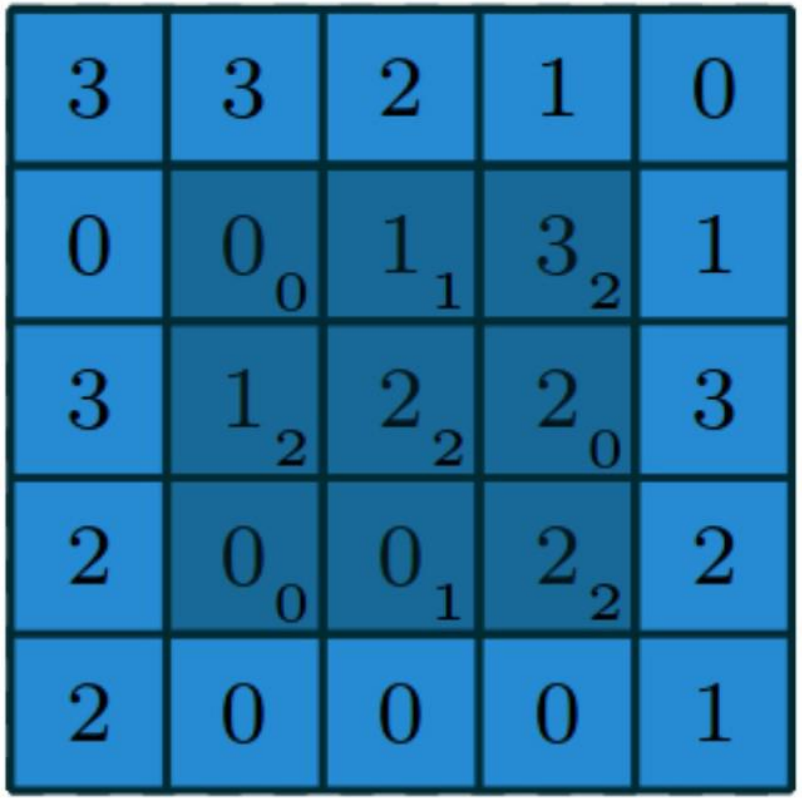


Feature map

12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

What is Convolution?

Image

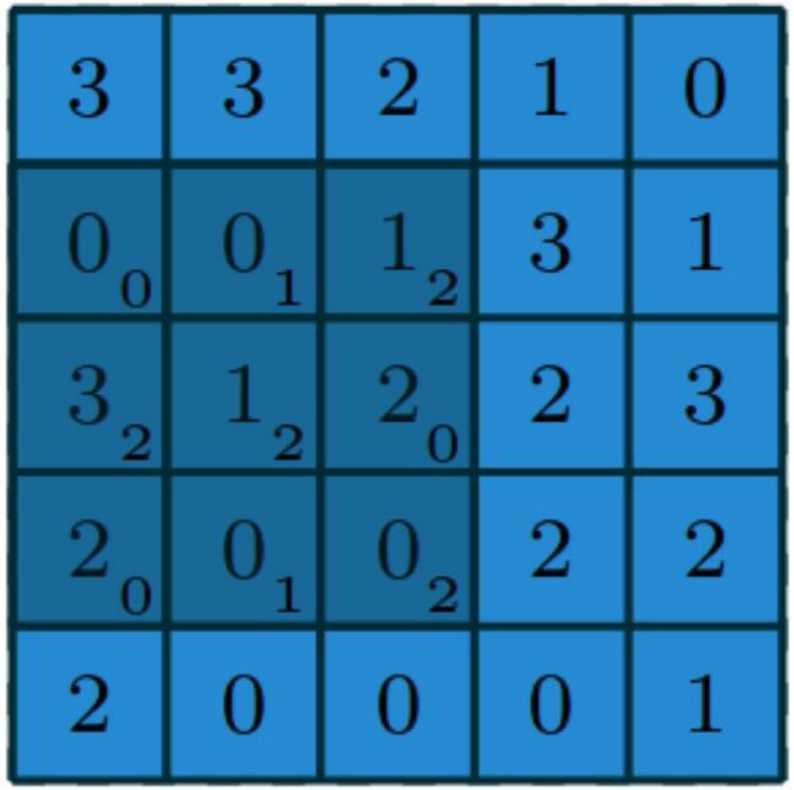


Feature map

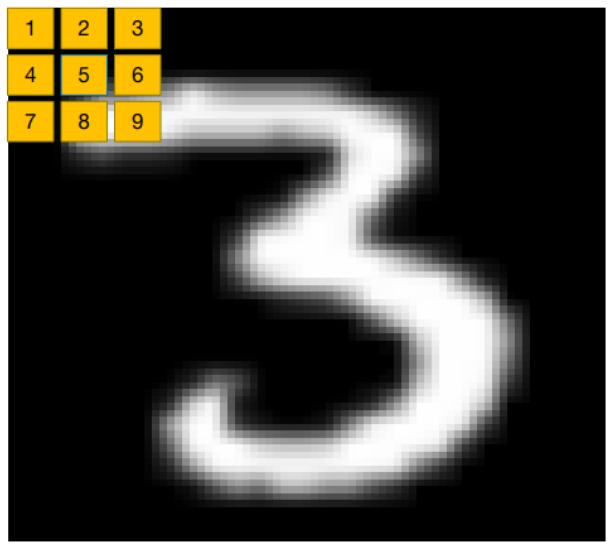
12.0	12.0	17.0
10.0	17.0	19.0
9.0	6.0	14.0

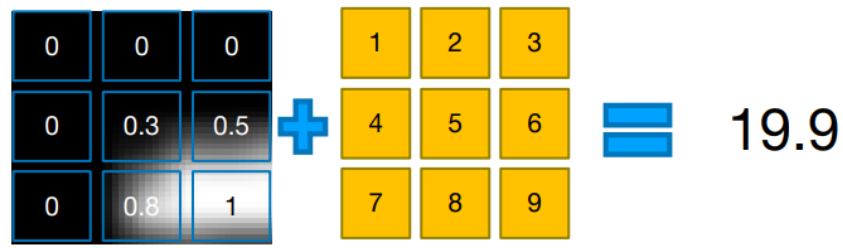
What is Convolution?

Image



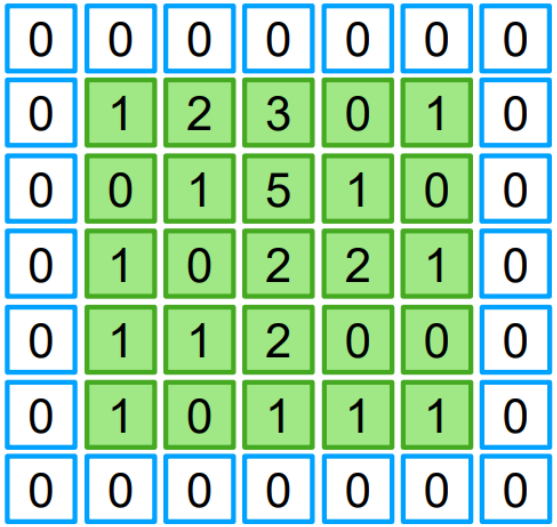
Feature map





(0 * 1) + (0 * 2) + (0 * 3) +
(0 * 4) + (0.3 * 5) + (0.5 * 6) +
(0 * 7) + (0.8 * 8) + (1 * 9) = 19.9

- Stride and Padding
  - stride : filter를 한번에 얼마나 이동 할 것인가
  - Padding : zero-padding



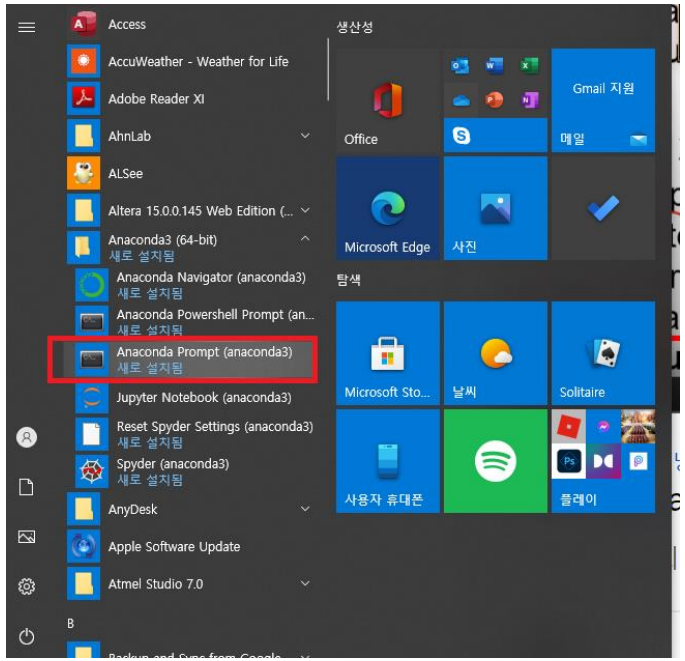
input + padding

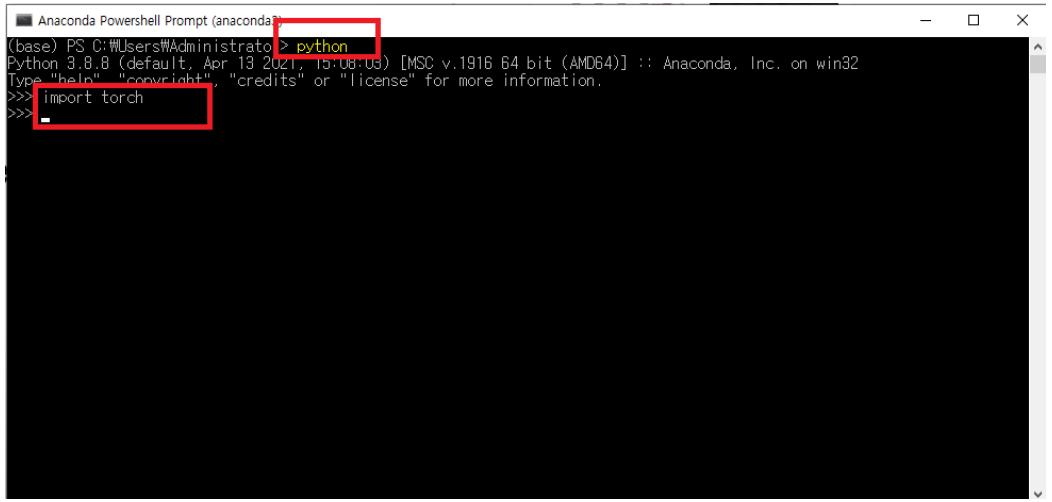
Output size of convolution

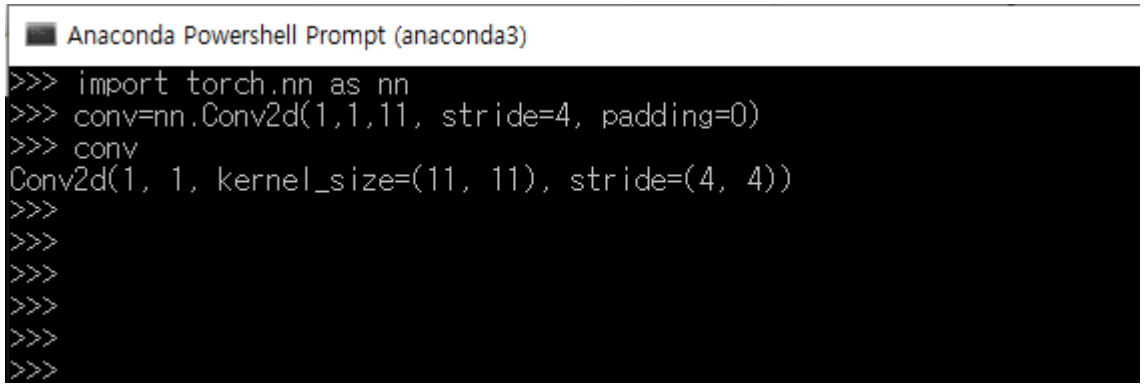
$$Output\ size = \frac{input\ size - filter\ size + (2 * padding)}{Stride} + 1$$

- Case 1: Input image size : 227 x 227 filter size = 11x11, stride = 4, padding = 0
  - output Image size =((227-11, 227-11)+(2*0))/4+1 = 55 x 55
- Case 2: input image size : 64 x 64 filter size = 7x7, stride = 2, padding = 0
  - output image size =((64-7, 64-7)+(2*0))/2+1 = 57/2+1 =28.5+1=29
- Case 3 : input image size : 32 x 64 filter size = 5x5, stride = 1, padding = 0
  - output image size =((32-5, 64-5)+(2*0))/1+1 = (27, 59)+1 = 28x 60

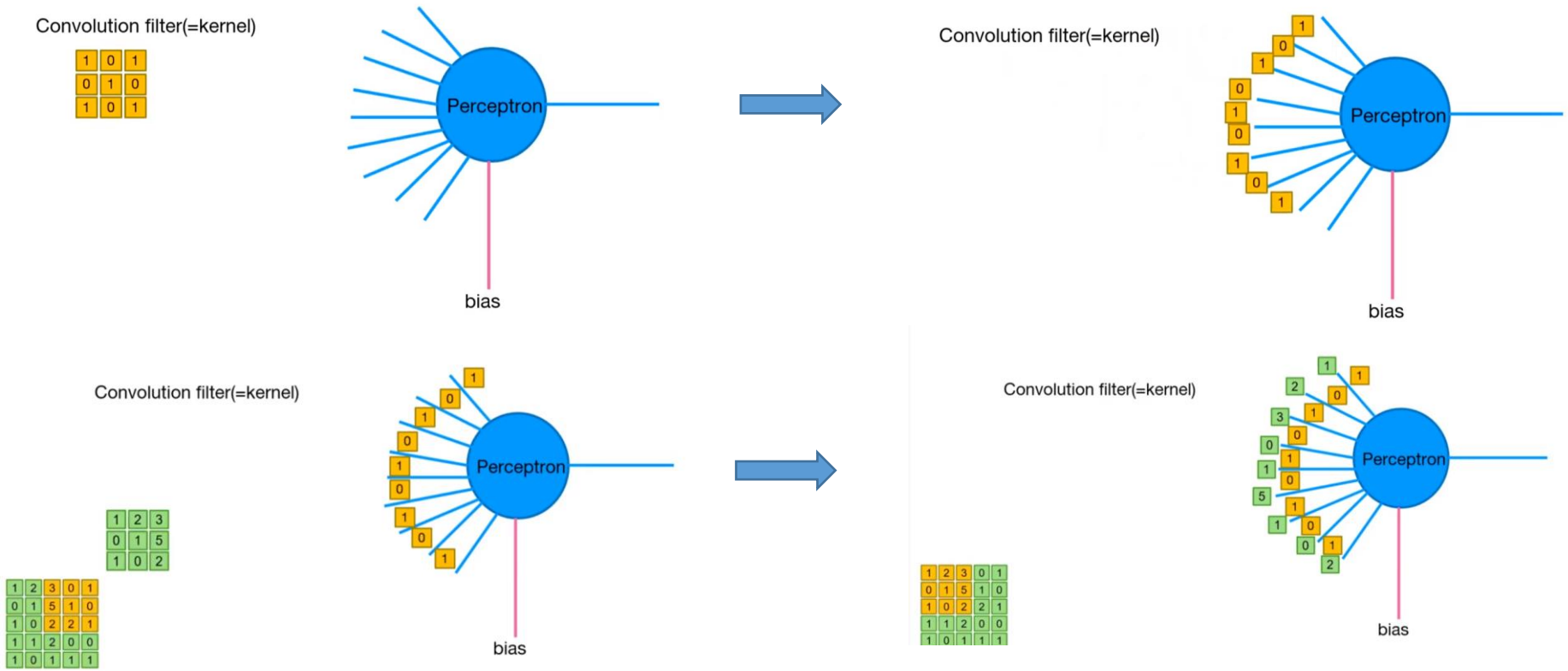
```
>import torch
>import torch.nn as nn
>conv=nn.Conv2d(1,1,11, stride=4, padding= 0)  # input size=1, batch size =1
>conv
>inputs=torch.Tensor(1,1,227,227)
>inputs.shape
>out=conv(inputs)
>out.shape
```





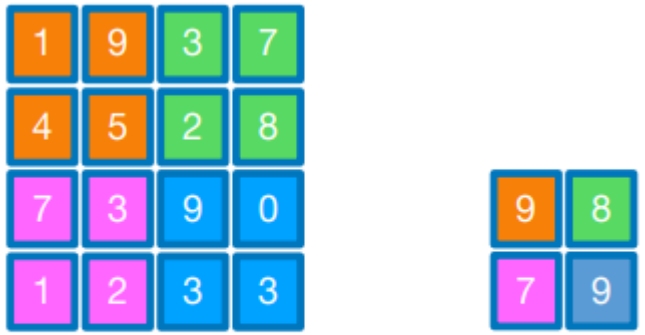


Perceptron & Convolution



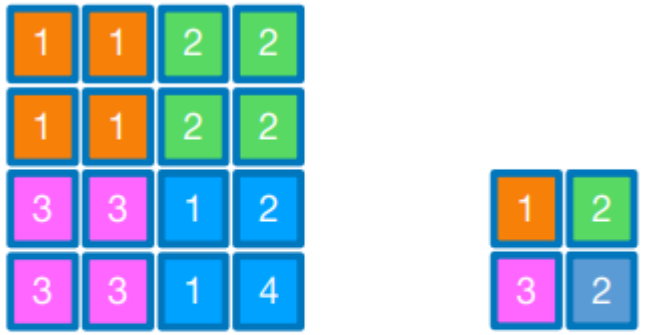
Pooling

- Max Pooling



max pooling

- Average Pooling



Average pooling

MaxPool2d

MaxPool2d

CLASS `torch.nn.MaxPool2d`(*kernel_size, stride=None, padding=0, dilation=1, return_indices=False, ceil_mode=False*) [SOURCE]

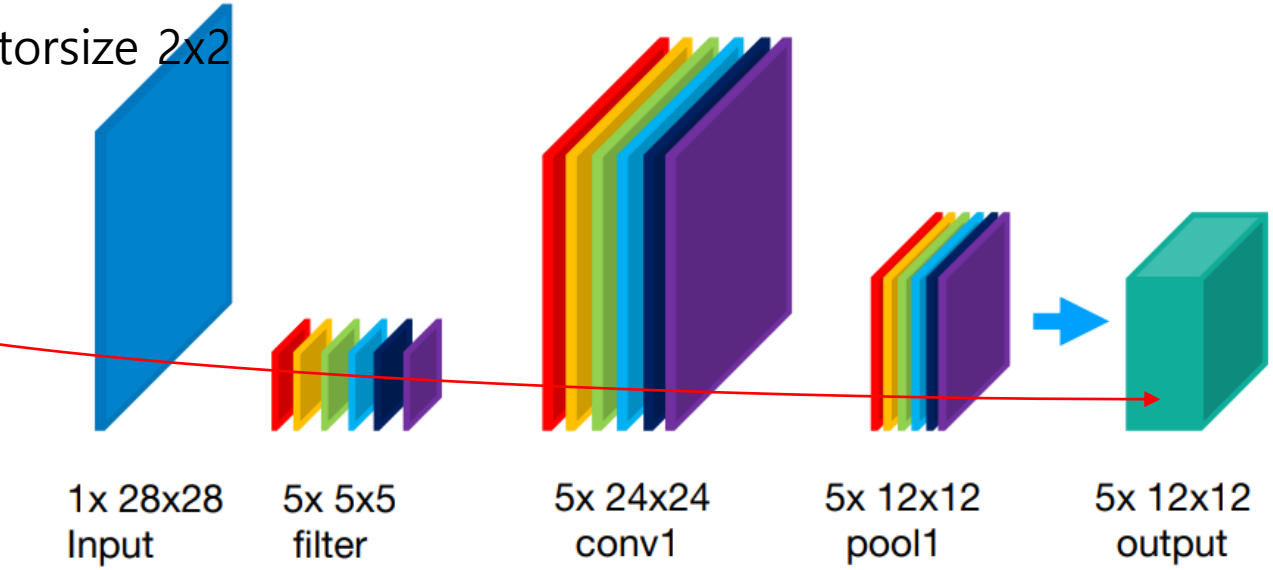
Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

CNN implementation

```
>input = torch.Tensor(1,1,28,28)
>conv1=nn.Conv2d(1,5,5)  #input channel 1, output channel 5, filter size 5
>pool = nn.MaxPool2d(2)  # Pooling vectorsize 2x2
>out = conv1(input)
>out2 =pool(out)
>out.size()
>out2.size()
```



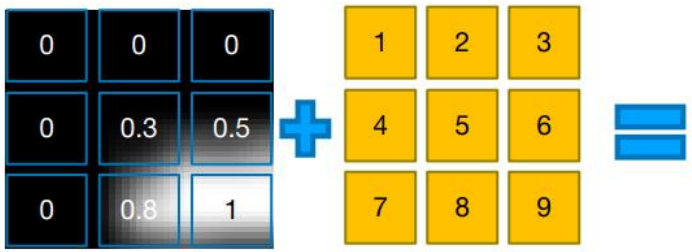
Pytorch nn.Conv2d

CLASS `torch.nn.Conv2d(`*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True*`)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$



???

ex) 입력 채널 1 /출력채널 1 / 커널 크기3x3

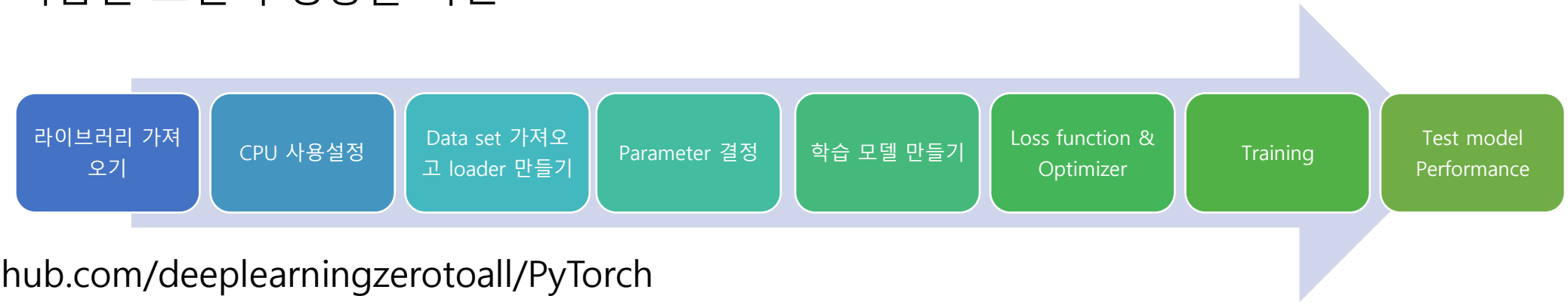
conv = nn.Conv2d(1,1,3)

- 입력 형태
  - input type : torch.Tensor
  - input shape : (N x C x H x W)
    (batch_size, channel, height, width)

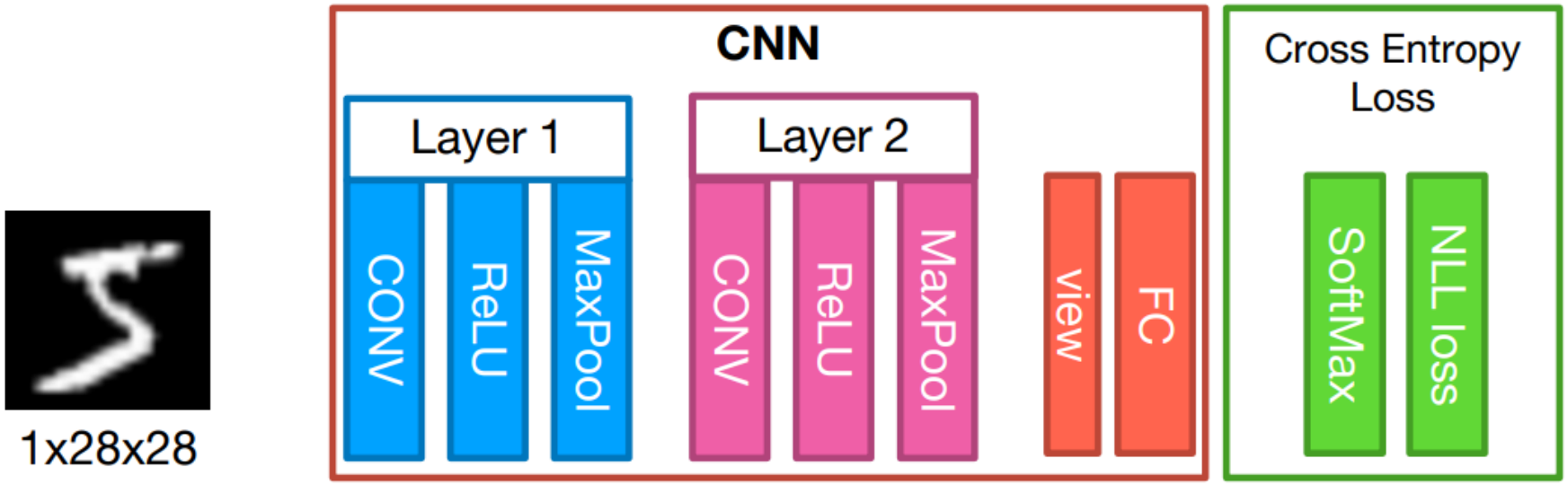
- Import torch
- Import torch.nn as nn
- Conv=nn.Conv2d(1,1,11,stride=4, padding=0)
- Conv

Handwritten number recognition(MNIST data set) with CNN

1. 라이브러리 가져오고 (torch, torchvision, matplotlib 등)
2. GPU 사용 설정 하고 random value를 위한 seed 설정
3. 학습에 사용되는 parameter 설정(learning_rate, training_epochs, batch_size, etc)
4. Dataset load (학습에 쓰기 편하게) & loader 만들기
5. 학습 모델 만들기(class CNN(torch.nn.Module))
6. Loss function (Criterion)을 선택하고 최적화 도구 선택(optimizer)
7. 모델 학습 및 loss check(Criterion의 output)
8. 학습된 모델의 성능을 확인



https://github.com/deeplearningzerotoall/PyTorch



(Layer 1) Convolution layer = (in_c=1, out_c=32,kernel_size =3, stride=1,padding=1)

(Layer 1) MaxPool layer = (kernel_size=2, stride =2)

(Layer 2) Convolution layer = (in_c=32, out_c=64, kernel_size =3, stride=1,padding=1)

(Layer 2) MaxPool layer = (kernel_size=2, stride =2)

view => (batch_size x [7,7,64] => batch_size x [3136])

Fully_Connect layer => (input=3136, output = 10)

In [1]:
```
# Lab 11 MNIST and Convolutional Neural Network
import torch
import torchvision.datasets as dsets
import torchvision.transforms as transforms
import torch.nn.init
```

In [2]:
```
device = 'cuda' if torch.cuda.is_available() else 'cpu'

# for reproducibility
torch.manual_seed(777)
if device == 'cuda':
    torch.cuda.manual_seed_all(777)
```

In [3]:
```
# parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

In [4]:
```
# MNIST dataset
mnist_train = dsets.MNIST(root='MNIST_data/',
                          train=True,
                          transform=transforms.ToTensor(),
                          download=True)

mnist_test = dsets.MNIST(root='MNIST_data/',
                         train=False,
                         transform=transforms.ToTensor(),
                         download=True)
```

In [5]:
```
# dataset loader
data_loader = torch.utils.data.DataLoader(dataset=mnist_train,
                                          batch_size=batch_size,
                                          shuffle=True,
                                          drop_last=True)
```

In [6]:

```python
# CNN Model (2 conv layers)
class CNN(torch.nn.Module):

    def __init__(self):
        super(CNN, self).__init__()
        # L1 ImgIn shape=(?, 28, 28, 1)
        #    Conv     -> (?, 28, 28, 32)
        #    Pool     -> (?, 14, 14, 32)
        self.layer1 = torch.nn.Sequential(
            torch.nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # L2 ImgIn shape=(?, 14, 14, 32)
        #    Conv      ->(?, 14, 14, 64)
        #    Pool      ->(?, 7, 7, 64)
        self.layer2 = torch.nn.Sequential(
            torch.nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            torch.nn.ReLU(),
            torch.nn.MaxPool2d(kernel_size=2, stride=2))
        # Final FC 7x7x64 inputs -> 10 outputs
        self.fc = torch.nn.Linear(7 * 7 * 64, 10, bias=True)
        torch.nn.init.xavier_uniform_(self.fc.weight)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)   # Flatten them for FC
        out = self.fc(out)
        return out
```

In [7]:
```
# instantiate CNN model
model = CNN().to(device)
```

In [8]:
```
# define cost/loss & optimizer
criterion = torch.nn.CrossEntropyLoss().to(device)    # Softmax is internally computed.
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

In [*]:
```
# train my model
total_batch = len(data_loader)
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0

    for X, Y in data_loader:
        # image is already size of (28x28), no reshape
        # label is not one-hot encoded
        X = X.to(device)
        Y = Y.to(device)

        optimizer.zero_grad()
        hypothesis = model(X)
        cost = criterion(hypothesis, Y)
        cost.backward()
        optimizer.step()

        avg_cost += cost / total_batch

    print('[Epoch: {:>4}] cost = {:>.9}'.format(epoch + 1, avg_cost))

print('Learning Finished!')
```

In [*]:
```
# Test model and check accuracy
with torch.no_grad():
    X_test = mnist_test.test_data.view(len(mnist_test), 1, 28, 28).float().to(device)
    Y_test = mnist_test.test_labels.to(device)

    prediction = model(X_test)

    correct_prediction = torch.argmax(prediction, 1) == Y_test
    accuracy = correct_prediction.float().mean()
    print('Accuracy:', accuracy.item())
```