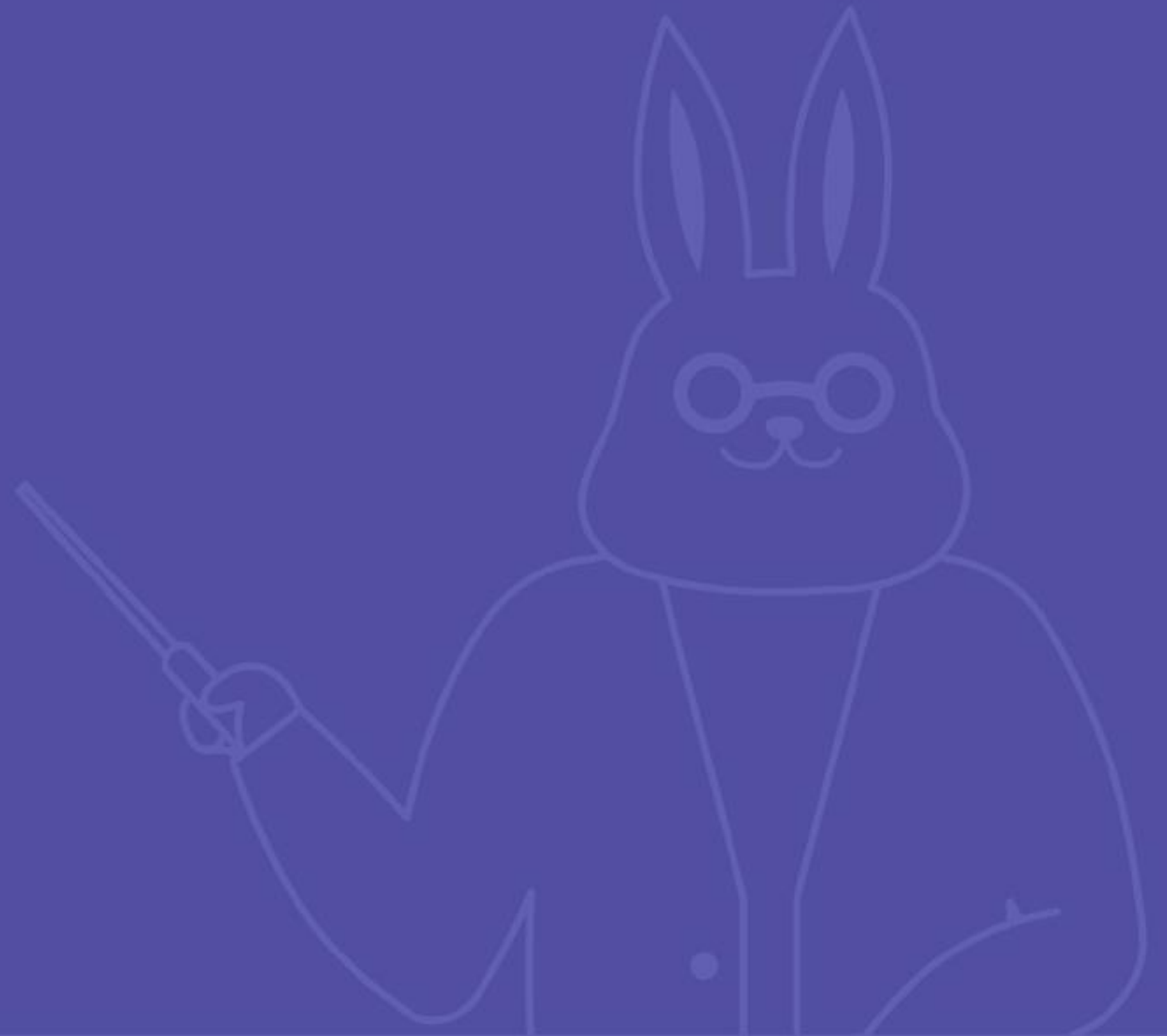


# 17주차 인공지능

인공지능



## 💡 강사소개



링크드인

## 한서우

안 물어요....!

모르는 거, 궁금한 거 아무거나 물어 봐주세요.

아는 거 답하는 것과 모르는 내용 찾는 것을 좋아합니다.

## 이력/경력

(현) 한국전자기술연구원 – 인공지능 개발자/연구자

(현) Webarter Inc. - CTO, Co-founder

(현) 국제인공지능&윤리협회 – 자문위원

전자공학과 석사 졸업

미국 카네기멜론대학 AI 교육프로그램 이수(전액 정부 지원금)

전자공학과 학사 졸업

# 수강목표

## 1. Python 복습

이전 수업에서 배운 Python을 까먹지는 않았는지 복습하는 시간을 가질거예요.

## 2. 행렬 개념 확인

인공지능에서 자주 등장하는 행렬에 대한 정의 및 기본 성질에 대해 배울거예요.

## 3. 인공지능 개념

인공지능이란 무엇인지 그리고 인공지능의 초기 모델에 대해 배울거예요.

# 목차

01. Python

02. Numpy

03. 행렬

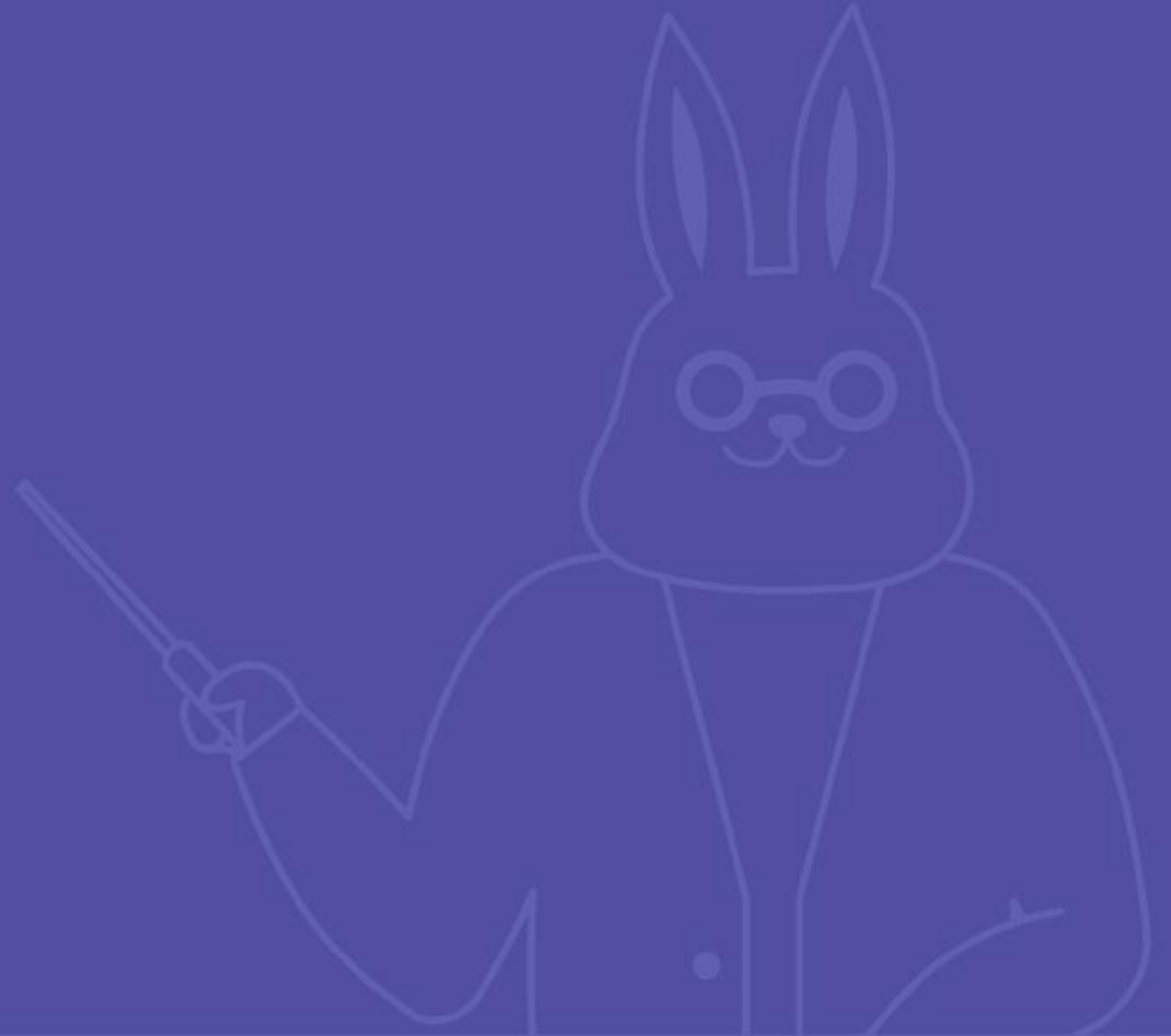
04. 인공지능

05. XOR과 MLP

06. Q&A

01

# Python



## ✓ Python 을 배우는 이유

1. 쉬운 문법

2. 높은 확장성 및 이식성

3. 활발한 생태계

✓ 쉬운 문법

Java 코드

```
public class Main{
    public static void main(String[] args){
        System.out.print("Hello World!");
    }
}
```

Python 코드

```
print("Hello World!")
```











C 코드

```
# include <stdio.h>

Int main(){
    print("Hello World!");
}
```

	메모리	시간
C	1116KB	0ms
Python	29076KB	64ms
Java	14168KB	132ms

✓ 활발한 생태계

Apr 2022	Apr 2021	Change	Programming Language		Ratings	Change
1	3	▲		Python	13.92%	+2.88%
2	1	▼		C	12.71%	-1.61%
3	2	▼		Java	10.82%	-0.41%
4	4			C++	8.28%	+1.14%
5	5			C#	6.82%	+1.91%
6	6			Visual Basic	5.40%	+0.85%
7	7			JavaScript	2.41%	-0.03%
8	8			Assembly language	2.35%	+0.03%
9	10	▲		SQL	2.28%	+0.45%
10	9	▼		PHP	1.64%	-0.19%

출처: [TIOBE](#)



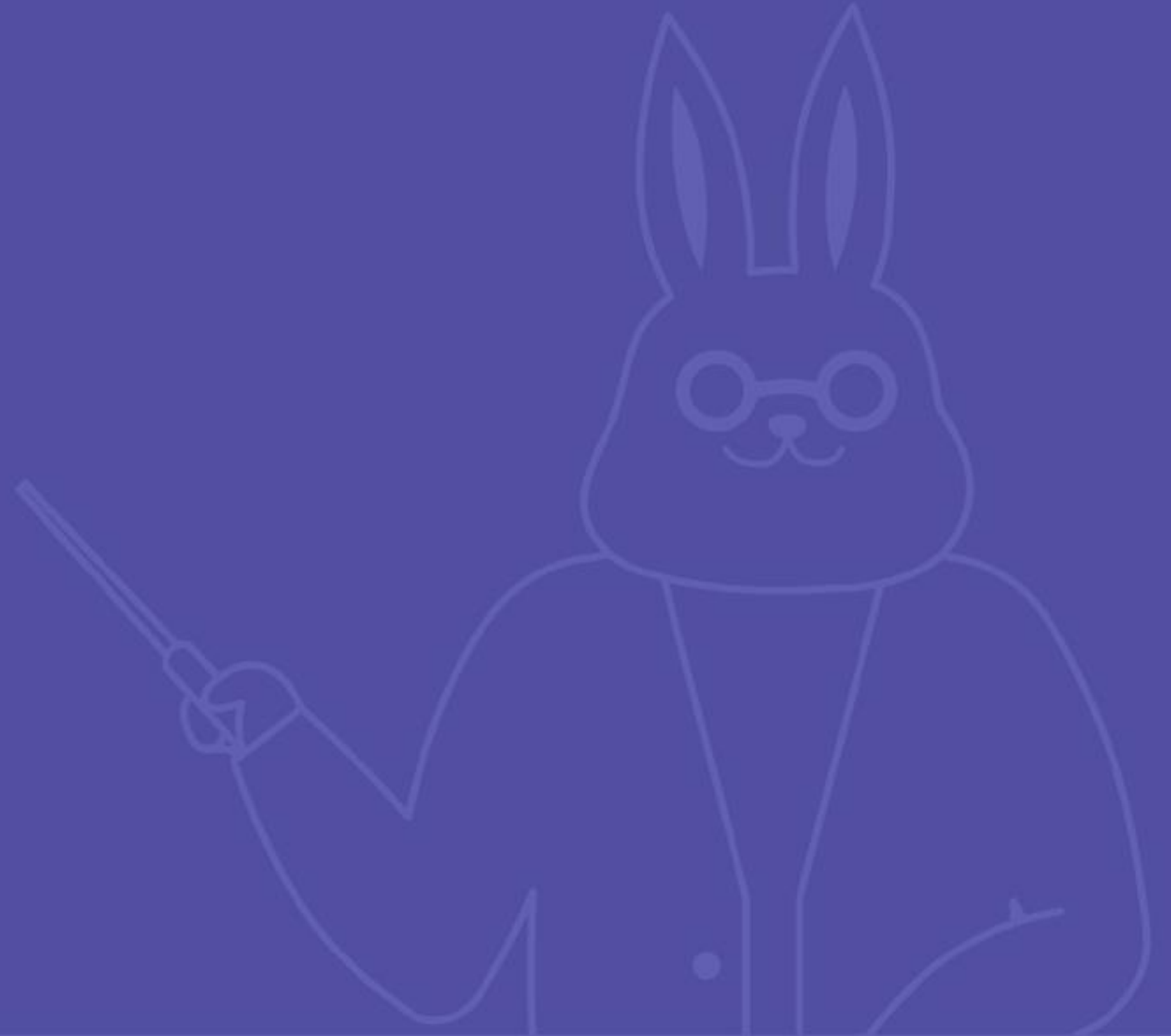
✓ Python 대표 자료형

	예시	특징 1	특징 2	특징 3	특징 4
숫자	123, 1.1	int, float ..	연산자를 통해 수학적 연산을 할 수 있음		immutable
불	False, True	Logical operator : and, or, not			immutable
문자열	'a', "ABC"	홀따옴표 또는 겹따옴표로 표현	순서가 있다	iterable	immutable
리스트	[1, '2', 3]	여러 자료를 저장할 수 있음	순서가 있다	iterable	mutable
딕셔너리	{ 'a' : 10, 'b' : 20 }	Key 와 value 의 순서쌍으로 이루어짐	Hashing 알고리즘을 통해 자료 탐색이 상당히 빠름	iterable	mutable
집합	{10, 20, 30}	중복을 허용하지 않음	자료의 중복을 제거할때 주로 사용	iterable	mutable

✓ Python 실습

02

# Numpy



## ✓ Python 모듈과 패키지



**모듈** : 함수나 변수 또는 클래스를 모아 놓은 파일

**패키지** : 연관된 여러 모듈의 묶음

**라이브러리** : 여러 모듈과 패키지를 묶어 부르는 말

## ✓ 모듈 불러오기

```
# my_module.py
def plus(a,b):
    c = a + b
    return c
```

```
# main.py
import my_module
print(my_module.plus(2,3))

import my_module as mm
print(mm.plus(2,3))
```

```
# main.py
from my_module import plus
print(plus(2,3))
```

`import` 라는 키워드를 이용해 다른 모듈의 함수, 변수, 클래스를 불러올 수 있음

이때, **모듈 이름**.(함수 or 변수 or 클래스) 로 사용할 수 있음

또는 `from` 이라는 키워드를 통해 특정 함수, 변수, 클래스를 불러올 수도 있음

## ✓ Numpy

- C언어로 구현된 Python 라이브러리
- 고성능 수치 계산을 위해 제작
- ndarray: 다차원 배열 클래스



### 코드

```
import numpy as np

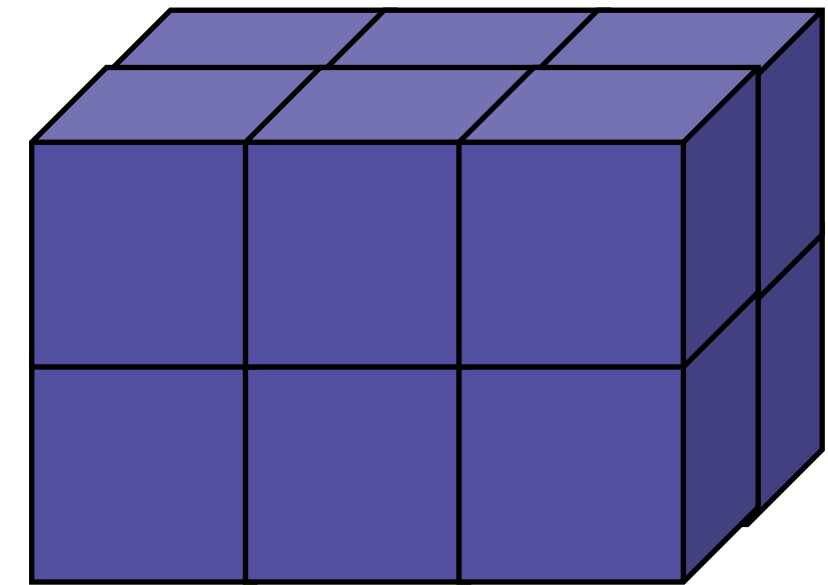
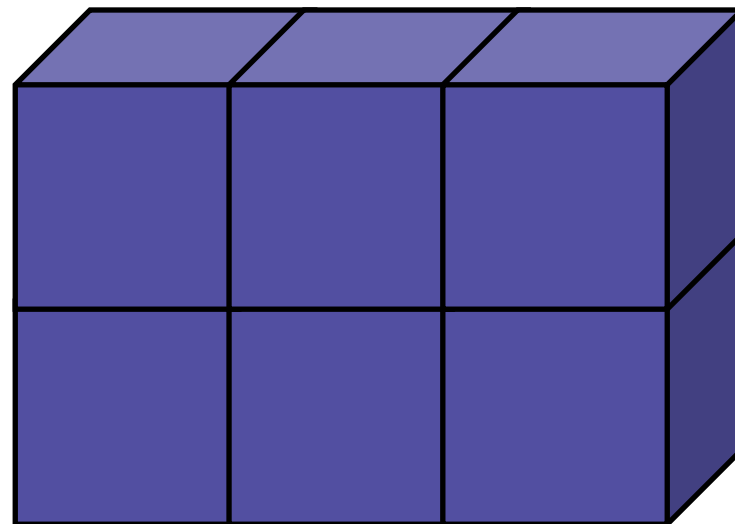
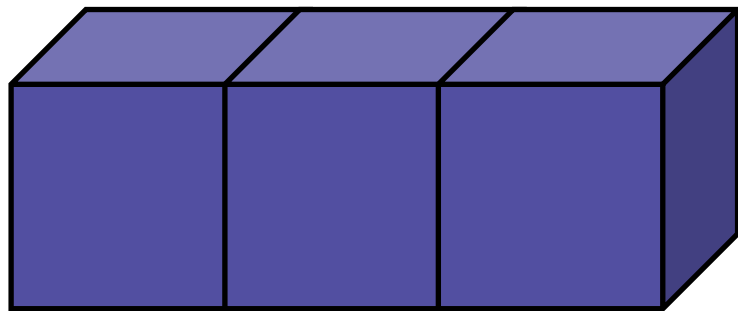
a = np.array([1, 2, 3]) # 넘파이 ndarray 객체의 생성
print(a)
print(a.shape) # a 객체의 형태(shape)
print(a.ndim) # a 객체의 차원
print(a.dtype) # a 객체 내부 자료 형
print(a.itemsize) # a 객체 내부 자료 형이 차지하는 메모리 크기(byte)
print(a.size) # a 객체의 전체 크기(항목의 수)
```

### 출력창

```
array([1, 2, 3])
(3, )
1
dtype('int32')
4
3
```

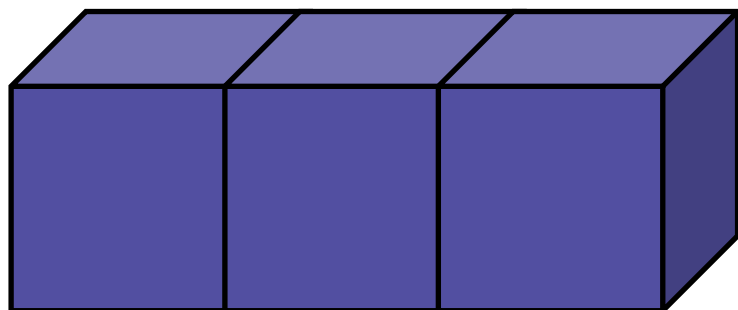
## ✓ 배열(array) vs. 행렬(matrix) vs. 벡터(vector)

- 배열(array): 컴퓨터에서 일반적으로 사용하는 개념으로 수를 포함하는 어떤 데이터의 묶음을 의미
- 벡터(vector): 1차원으로 묶은 수를 부르며 행만 구성된 것을 행벡터, 열만 구성된 것을 열벡터라 부름
- 행렬(matrix): 2차원으로 묶은 수를 의미, 우리가 알고 있는 수많은 행렬 계산식에 사용

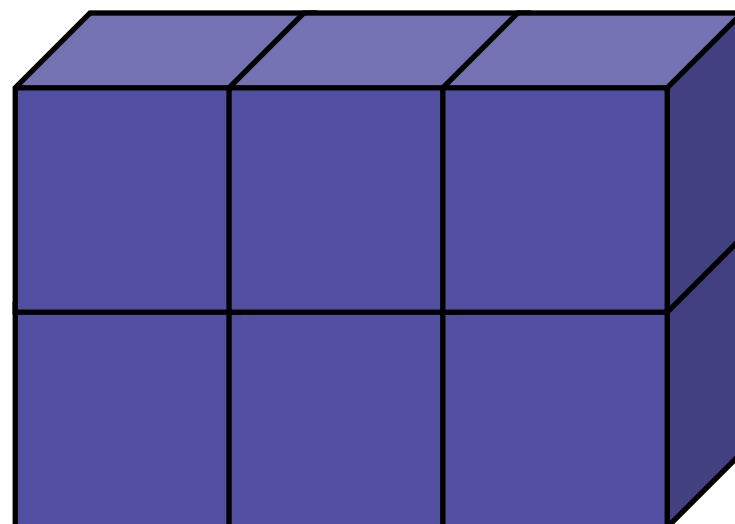


## ✓ 배열(array) vs. 행렬(matrix) vs. 벡터(vector)

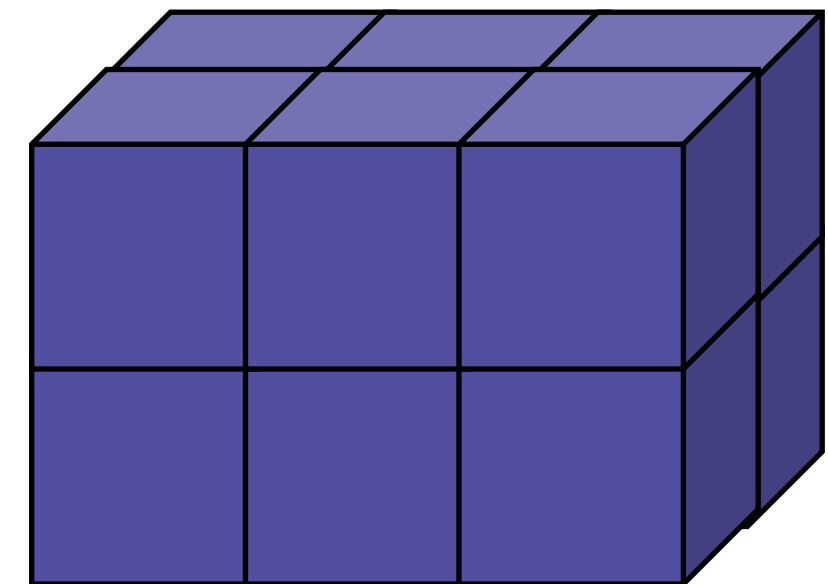
- 배열(array): 컴퓨터에서 일반적으로 사용하는 개념으로 수를 포함하는 어떤 데이터의 묶음을 의미
- 벡터(vector): 1차원으로 묶은 수를 부르며 행만 구성된 것을 행벡터, 열만 구성된 것을 열벡터라 부름
- 행렬(matrix): 2차원으로 묶은 수를 의미, 우리가 알고 있는 수많은 행렬 계산식에 사용



벡터



배열



행렬



03

# 행 령



### ✓ 행렬의 정의

- 행렬(matrix)은 숫자들을 직사각형 형태로 행과 열에 따라 나열한 것
- 예를 들어, 아래 행렬  $A$ 는 **행이 2개**, **열이 3개**로 구성

3 columns

↓ ↓ ↓

$$A = \begin{bmatrix} -2 & 5 & 6 \\ 5 & 2 & 7 \end{bmatrix}$$

← ← 2 rows

### ✓ 행렬의 차원

- 행렬의 차원은 크기를 나타냄 → 이는 행과 열 순서로 행과 열의 숫자를 나타냄
- 행렬  $A$ 는 행이 2개, 열이 3개 있으므로, 차원을  $2 \times 3$ 이라고 표기하며, "2 곱하기 3"이라고 읽음

3 columns

↓ ↓ ↓

$$A = \begin{bmatrix} -2 & 5 & 6 \\ 5 & 2 & 7 \end{bmatrix}$$

← ← 2 rows

### ✓ 행렬의 차원 퀴즈

- $B$  행의 개수?
- $B$  열의 개수?
- $\square \times \square$  행렬

$$B = \begin{bmatrix} -8 & -4 \\ 23 & 12 \\ 18 & 10 \end{bmatrix}$$

### ✓ 행렬의 차원 퀴즈1

- $B$  행의 개수?
- $B$  열의 개수?
- $3 \times 2$  행렬
- 행렬의 차원을 다룰 땐, **행  $\times$  열** 이라는 것을 기억!

$$B = \begin{bmatrix} -8 & -4 \\ 23 & 12 \\ 18 & 10 \end{bmatrix}$$

### ✓ 행렬의 차원 퀴즈2

- $F$  행의 개수?
- $F$  열의 개수?
- $\square \times \square$  행렬

$$F = \begin{bmatrix} -2 \\ 0 \\ 10 \end{bmatrix}$$

### ✓ 행렬의 차원 퀴즈2

- $F$  행의 개수?
- $F$  열의 개수?
- $3 \times 1$  행렬

$$F = \begin{bmatrix} -2 \\ 0 \\ 10 \end{bmatrix}$$

### ✓ 행렬의 요소

- 행렬의 요소란? 단순히 행렬에 있는 수
- 행렬 안에 있는 각 요소는 그것이 위치한 행과 열로 이름을 지음
- $g_{1,2} =$
- $g_{2,1} =$

$$G = \begin{bmatrix} 4 & 14 & -7 \\ 18 & 5 & 13 \\ -20 & 4 & 22 \end{bmatrix}$$



### ✓ 행렬의 요소

- 행렬의 요소란? 단순히 행렬에 있는 수
- 행렬 안에 있는 각 요소는 그것이 위치한 행과 열로 이름을 지음
- $g_{1,2} = 14$
- $g_{2,1} = 18$

$$G = \begin{bmatrix} 4 & 14 & -7 \\ 18 & 5 & 13 \\ -20 & 4 & 22 \end{bmatrix}$$

### ✓ 행렬의 요소

- 행렬의 요소란? 단순히 행렬에 있는 수
- 행렬 안에 있는 각 요소는 그것이 위치한 행과 열로 이름을 지음
- 일반적으로 행렬  $A$ 의  $i$ 번째 행과  $j$ 번째 열에 있는 요소는 \_\_\_\_\_로 나타냄

$$G = \begin{bmatrix} 4 & 14 & -7 \\ 18 & 5 & 13 \\ -20 & 4 & 22 \end{bmatrix}$$

### ✓ 행렬의 요소

- 행렬의 요소란? 단순히 행렬에 있는 수
- 행렬 안에 있는 각 요소는 그것이 위치한 행과 열로 이름을 지음
- 일반적으로 행렬  $A$ 의  $i$ 번째 행과  $j$ 번째 열에 있는 요소는  $a_{i,j}$ 로 나타냄

$$G = \begin{bmatrix} 4 & 14 & -7 \\ 18 & 5 & 13 \\ -20 & 4 & 22 \end{bmatrix}$$

## ✓ 행벡터와 열벡터

$$\mathbf{A} = [a_{ij}] = \begin{pmatrix} a_{11} & a_{12} & a_{1n} \\ a_{21} & a_{22} & a_{2n} \\ \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{mn} \end{pmatrix} = \begin{pmatrix} \mathbf{a}_1^T \\ \mathbf{a}_2^T \\ \vdots \\ \mathbf{a}_m^T \end{pmatrix} = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n)$$

$$\mathbf{a}_i = \begin{pmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{in} \end{pmatrix} \quad \mathbf{b}_j = \begin{pmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{pmatrix}$$

## ✓ 행렬과 스칼라배

성질

예제

곱셈의 결합법칙

$$(cd)A = c(dA)$$

분배법칙

$$c(A + B) = cA + cB$$

$$(c + d)A = cA + dA$$

곱셈법 단위원 법칙

$$1A = A$$

곱셈법 0의 법칙

$$0 \cdot A = O$$

$$c \cdot O = O$$

곱셈의 닫힘법칙

$cA$ 는  $A$ 와 같은 차원을 가진 행렬입니다.

### ✓ 행렬과 스칼라배

- 행렬을 다룰 때, 우리는 실수를 스칼라라고 부름
- 스칼라배란 실수와 행렬의 곱셈을 뜻함
- 스칼라배에서, 행렬의 각 요소는 주어진 스칼라에 곱해짐

$$2 \cdot \begin{bmatrix} 5 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 2 \cdot 5 & 2 \cdot 2 \\ 2 \cdot 3 & 2 \cdot 1 \end{bmatrix}$$

$$= \begin{bmatrix} 10 & 4 \\ 6 & 2 \end{bmatrix}$$

✓ 행렬과 스칼라배 다른 예시(판서)

성질	예제
곱셈의 결합법칙	$(cd)A = c(dA)$
분배법칙	$c(A + B) = cA + cB$
	$(c + d)A = cA + dA$
곱셈법 단위원 법칙	$1A = A$
곱셈법 0의 법칙	$0 \cdot A = O$
	$c \cdot O = O$
곱셈의 닫힘법칙	$cA$ 는 $A$ 와 같은 차원을 가진 행렬입니다.

## ✓ 행렬의 덧셈, 뺄셈 예시(Numpy)

### 코드

```
import numpy as np

A = np.array([10, 11, 12, 13, 14])
B = np.array([0, 1, 2, 3, 4])
print(A+B)
print(A-B)

C = np.array([[5, 6], [7, 8]])
D = np.array([[10, 20], [30, 40]])
print(C+D)
print(C-D)
```

### 출력창

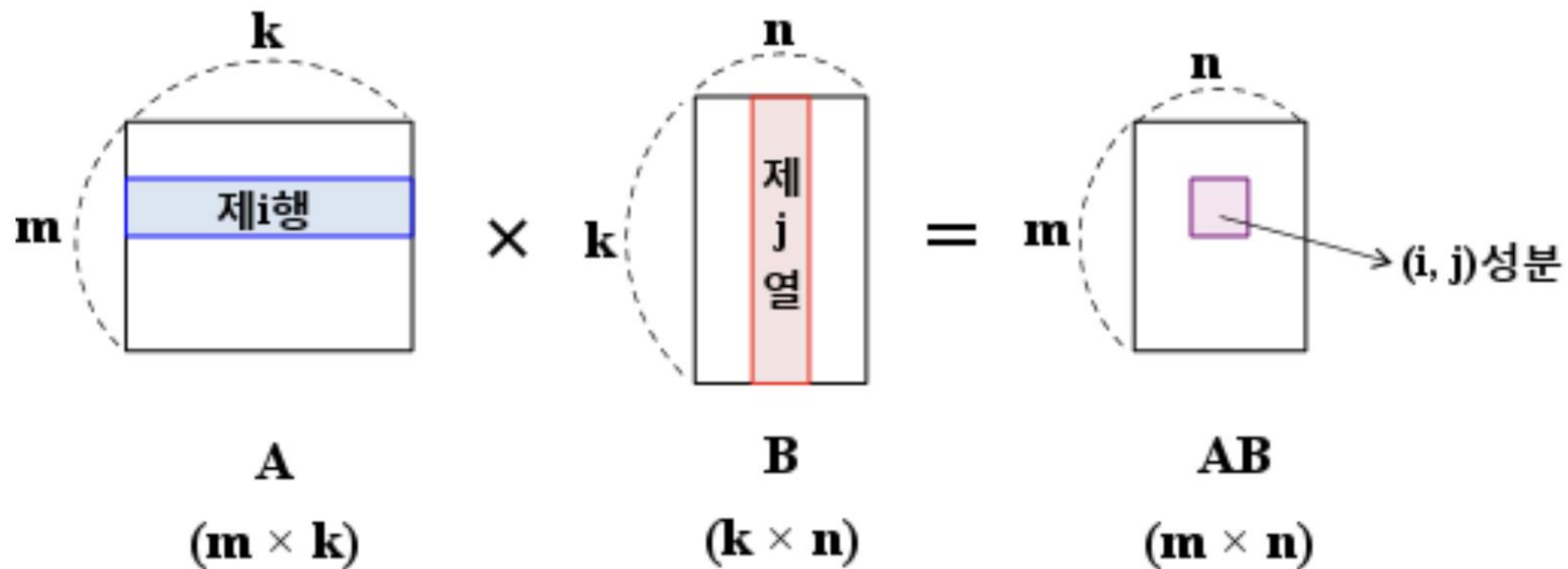
```
array([10,12,14,16,18])
array([10,10,10,10,10])

array([[15,26],
       [37,48]])
array([[ -5,-14],
       [-27,-32]])
```



### ✓ 행렬의 곱셈

- 두 행렬 A의 열의 개수와 행렬 B의 행의 개수가 같을 때, 행렬 A의 제 i 행의 각 성분과 행렬 B의 제 j 행의 각 성분을 그 순서대로 곱하여 더한 것을 (i, j) 성분으로 하는 행렬을 A와 B의 곱이라 함
- 기호로 AB라 나타냄



✓ 행렬의 곱셈 예시(판서)

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

$$B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

## ✓ 행렬의 곱셈 예시(Numpy)

### 코드

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
print(A.shape) # A 객체의 형태(shape)
B = np.array([[5, 6], [7, 8]])
print(B.shape) # B 객체의 형태(shape)
C = np.dot(A, B)
print(C) # C 객체
```

### 출력창

```
(2, 2)
(2, 2)
[ [19 22]
  [43 50] ]
```

### ✓ 행렬의 전치행렬

- 전치행렬(transposed matrix)이란? 어떤 행렬의 행과 열을 서로 맞바꾼 행렬

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

$$B = \begin{bmatrix} x & y \\ z & w \end{bmatrix} \Rightarrow B^T = \begin{bmatrix} x & z \\ y & w \end{bmatrix}$$

$$C = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -3 & 5 & -2 & 7 \end{bmatrix} \Rightarrow C^T = \begin{bmatrix} 1 & -3 \\ 1 & 5 \\ 1 & -2 \\ 1 & 7 \end{bmatrix}$$

## ✓ 행렬의 전치행렬 예시(Numpy)

### 코드

```
import numpy as np

A = np.array([[1, 2], [3, 4]])
print(A.transpose()) # A 객체의 전치행렬
print(np.transpose(A)) # A 객체의 전치행렬
```

### 출력창

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

### ✓ 행렬의 단위행렬

- 단위행렬 E란? 왼쪽 위에서 오른쪽 아래로 대각선 방향의 성분이 1이고 다른 성분이 모두 0인 n차 정사각형 행렬을 n차 단위 행렬이라고 함
- 숫자에서 1과 같은 역할

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

### ✓ 행렬의 역행렬

- 역행렬(inverse matrix)이란? 어떤 행렬  $A$ 와 곱했을 때, 곱셈에 대한 **항등원**인 **단위행렬  $E$** 가 나오게 하는 행렬을 행렬  $A$ 의 역행렬이라고 함
- 행렬  $A$ 의 역행렬은 기호로  $A^{-1}$ 라고 함

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \quad A^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

(단,  $ad - bc \neq 0$ )

✓ 행렬의 역행렬 예시(판서)



## ✓ 행렬의 전치행렬 예시(Numpy)

### 코드

```
import numpy as np

A = np.array([[0, 1], [2, 3]])
print(np.linalg.inv(A)) # A 객체의 역행렬
```

### 출력창

```
[[ -1.5,  0.5]
 [  1.,  0.]]
```

04

# 인공지능



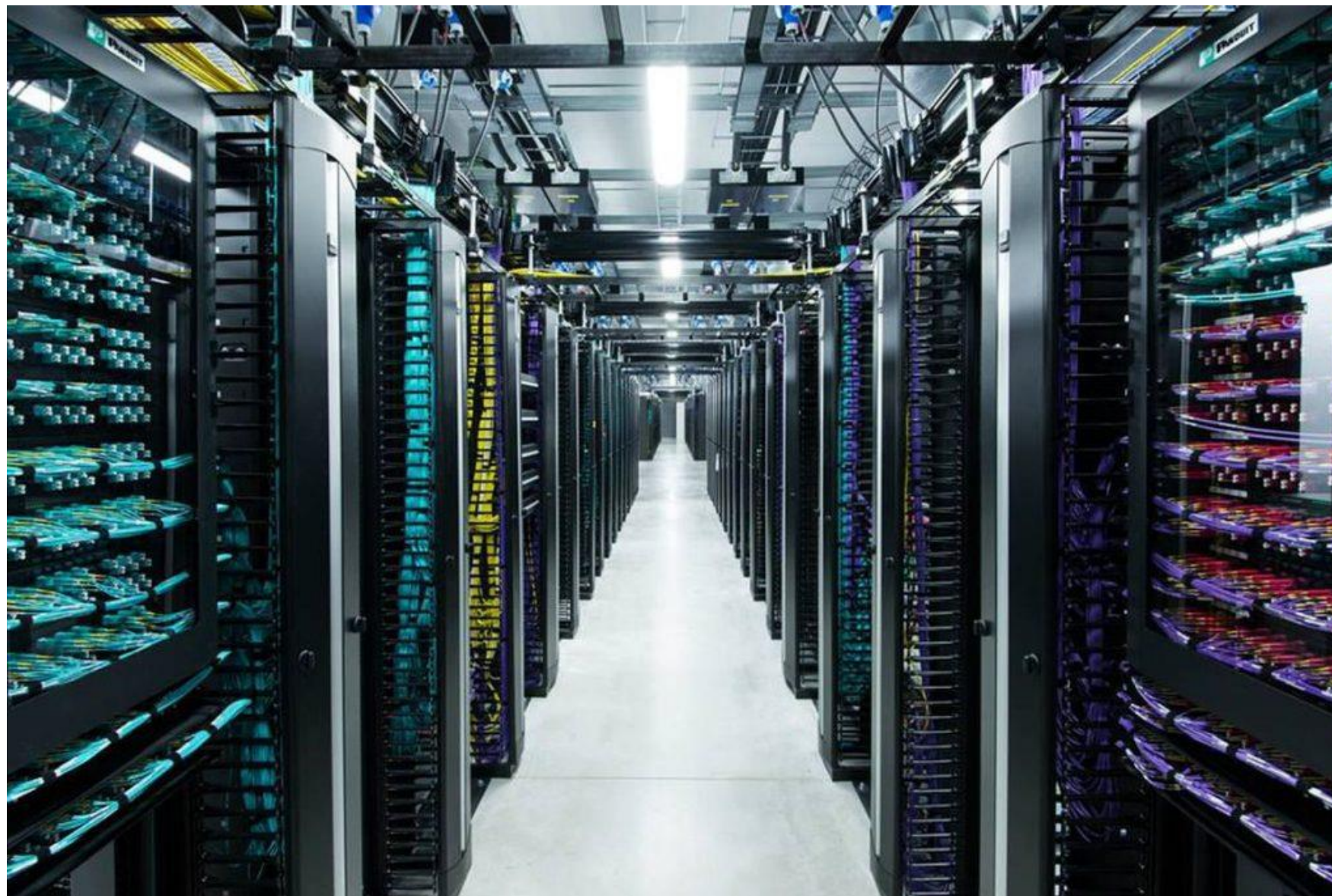
## ✓ 인공지능 강의 목표

- 인공지능 실시간 강의는 온라인 강의의 **예습이 핵심 목표** → 수식은 최소로, 알짜배기 개념 강의
- 실시간 강의 이해 기반, 온라인 강의에서 **심화 학습** 진행
- 실시간 강의의 **난이도가 너무 쉽다**고 생각되면, 언제든지 디스 코드를 이용한 피드백 요청
- 온라인 강의에서 **어려운 내용이** 있었다면, 언제든지 디스 코드를 이용한 **추가 설명** 요청



## ✓ 인공지능 설명

- 인공지능이란? 컴퓨터가 인간처럼 생각하고 학습하고 판단하여 스스로 행동하도록 만드는 기술



<https://www.menerga-adria.com/blog/2017/10/24/data-centers-cooling-and-ai/>

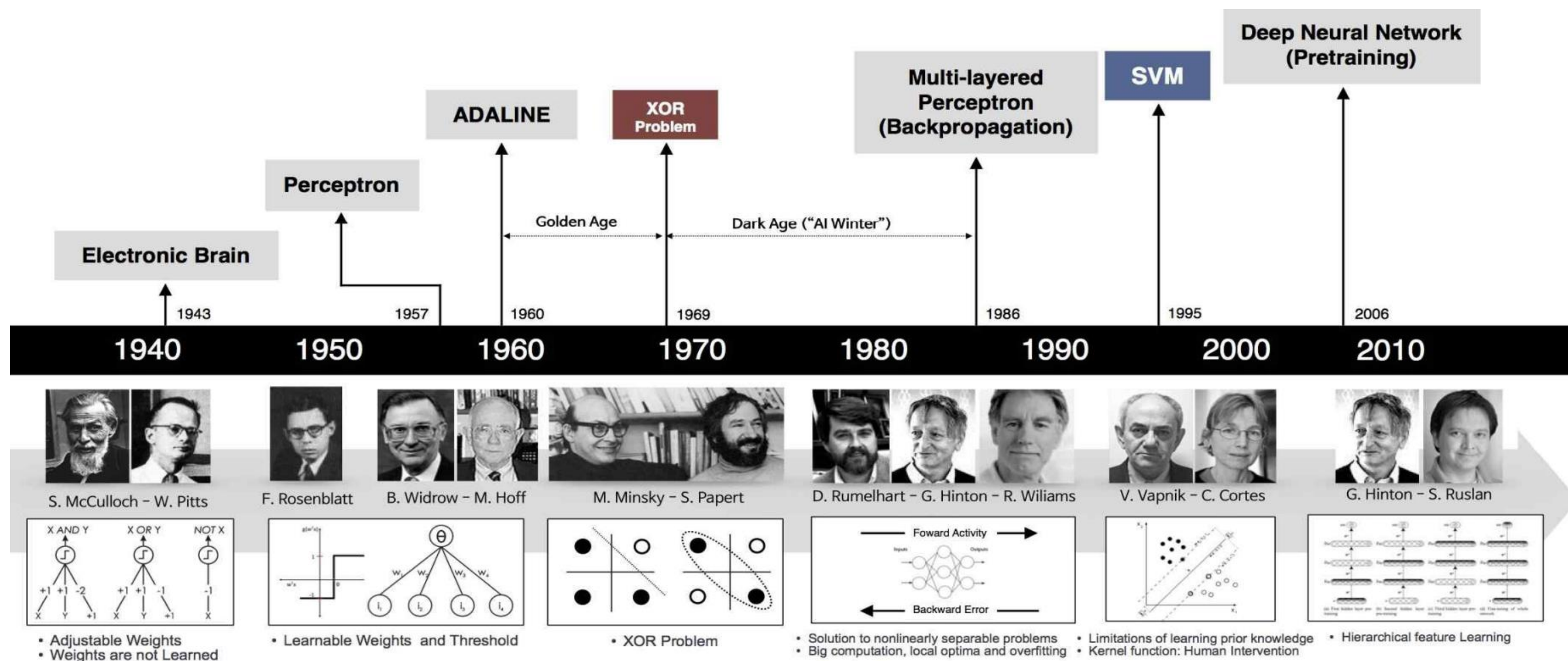


<https://www.sciencetimes.com/articles/29732/20210218/new-study-takes-closer-hologram-infused-food.htm>



## ✓ 인공지능 역사

- AI 발전을 이끄는 기술의 등장(뉴럴 네트워크 개발의 이정표)
- “글로벌 인공지능 연구의 4대 키워드와 시사점” – NIA 한국정보화진흥원



Andrew L. Beam(2017), "Deep Learning 101"

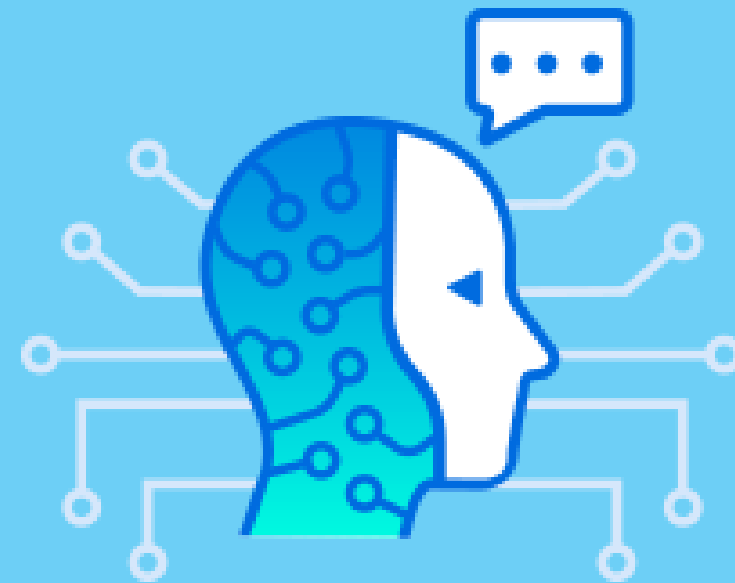
## ✓ 인공지능 유형

- **강인공지능 Strong AI (Artificial General Intelligence)**
  - 인공지능이 사람과 똑같이 스스로 학습하여 똑같이 행동한다는 것을 의미
  - 자의식이 있는 인공지능
  - AI가 스스로 데이터를 찾아서 학습이 가능한 상태를 의미
  - 강인공지능의 예) SF 영화 속에서 스스로 생각하고 알아서 행동하는 로봇
- **약인공지능 Weak AI (Artificial Narrow Intelligence)**
  - 강인공지능이 현실적으로 불가능한 상황
  - 현시점에서 활용되고 있는 대부분의 인공지능이 약인공지능임
  - 영상, 음성, 자연어 인식 등 특정 영역에만 활용이 가능
  - 알고리즘은 물론 기초 데이터와 규칙을 입력해야 함
  - 약인공지능의 예) 알파고(바둑), 인공지능 왓슨(암 진단)
- **초인공지능 (Artificial Super Intelligence)**
  - 인간의 지식을 1,000배 이상 초월하고 모든 면에서 월등한 인공지능
  - 초인공지능의 예) 영화 속의 가상인물인 터미네이터

## ✓ 약인공지능 분류

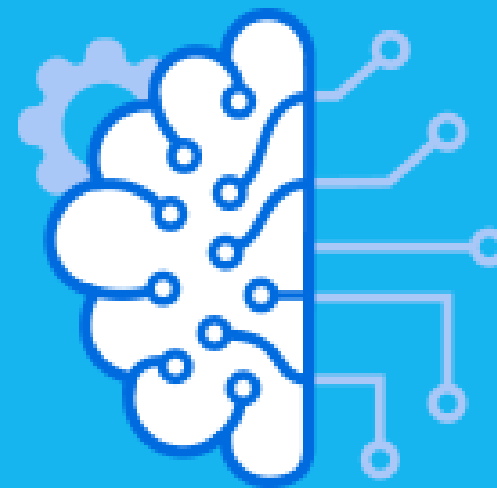
### 인공지능 Artificial Intelligence

사이버네틱스  
전문가 시스템  
...



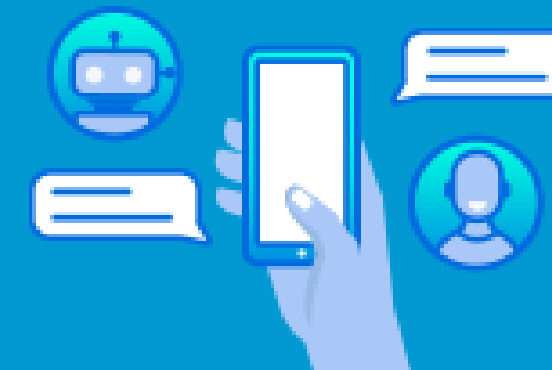
### 기계학습 Machine Learning

인공신경망  
결정 트리  
베이즈 네트워크  
서포트 벡터 머신  
...



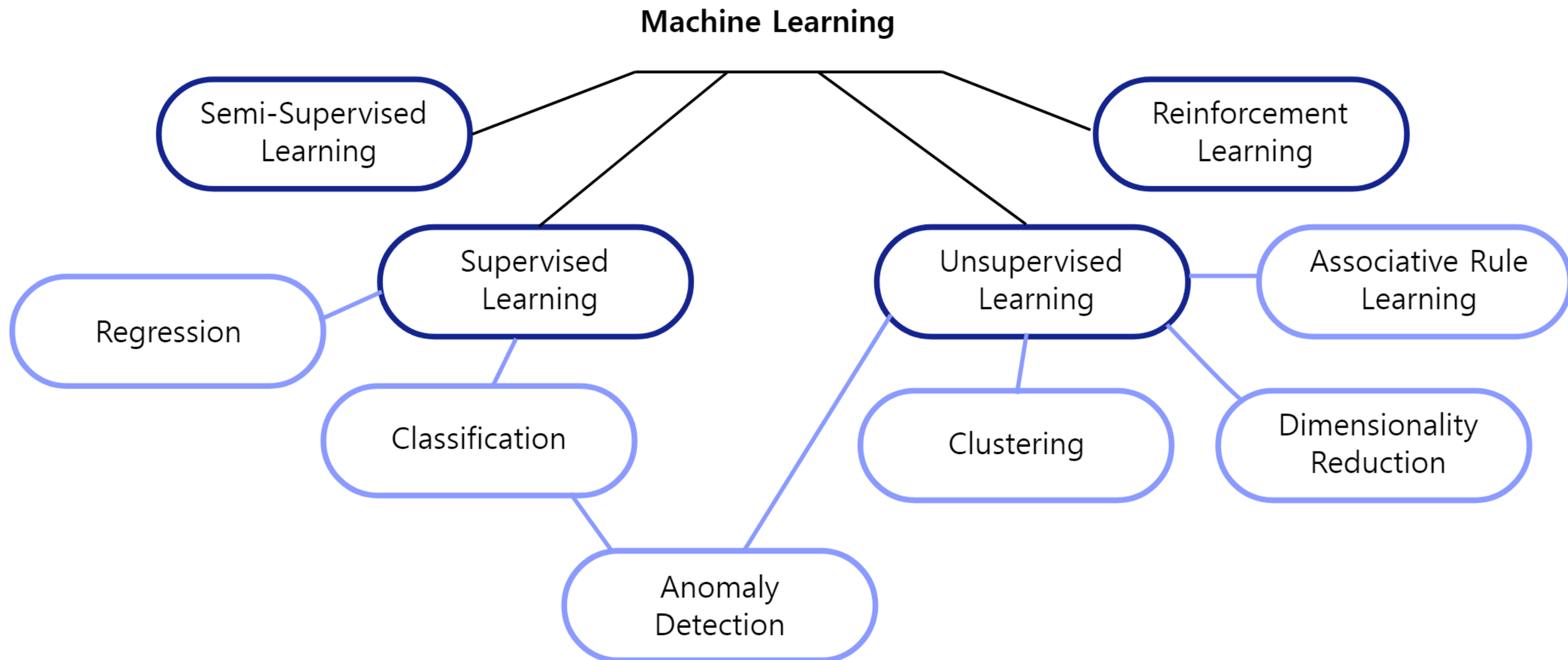
### 딥러닝 Deep Learning

CNN  
RNN  
RBM  
...



## ✓ 머신러닝 학습 방법

- 머신러닝 알고리즘은 아래와 같이 4종류의 학습 방법으로 나뉨
- 예전에는 3 종류였는데, Semi-supervised learning이 최근 추가됨





## ✓ 머신러닝 학습 방법

### • 지도학습 (supervised learning )

- 입력과 이에 대응하는 정답 데이터를 연관시키는 관계를 학습하는 방법
- 입력과 출력 쌍이 데이터로 주어지는 경우 그들 사이의 대응 관계를 학습하게 됨

### • 비지도 학습 (unsupervised learning)

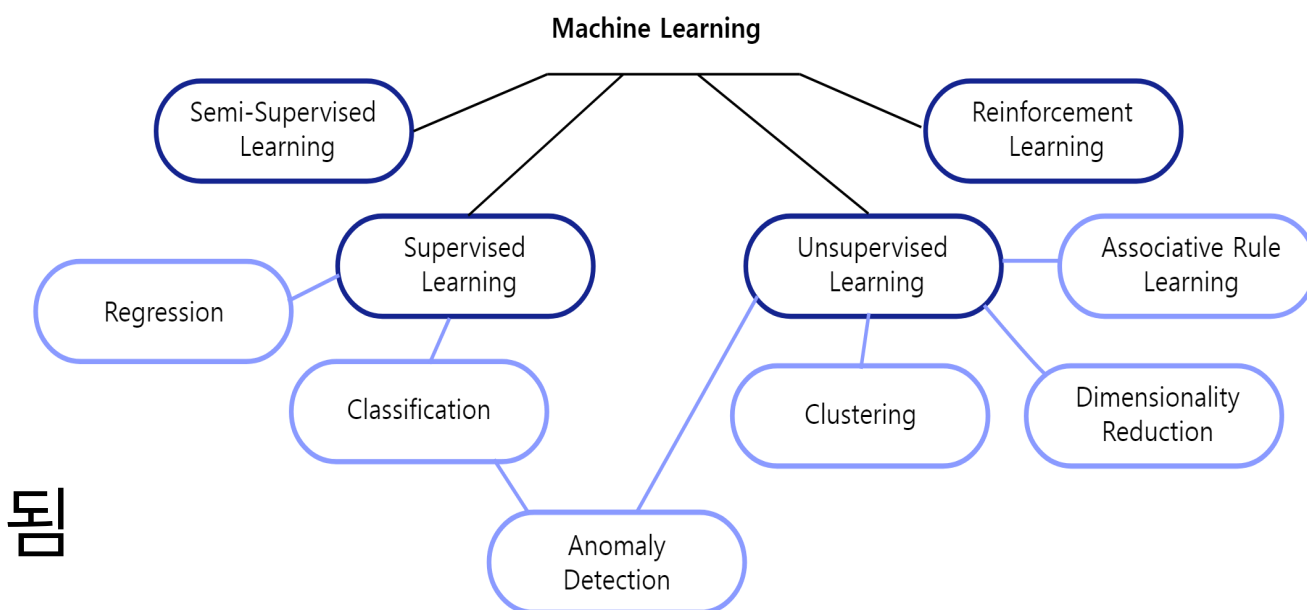
- 출력 없이 또는 출력 값을 알려주지 않고 주어진 입력만으로 스스로 모델을 구축하여 학습하는 방법.
- 입력만 있고 출력 즉 레이블(label)이 없는 경우에 적용하며, 입력 사이의 규칙성 등을 스스로 찾아내는 것이 학습의 주요 목표

### • 준지도학습 (semi-supervised learning )

- 적은 입력과 매칭하는 정답 데이터 쌍 그리고 정답이 없는 대량의 데이터의 관계를 학습하는 방법
- 소량의 라벨 데이터에는 지도학습을 적용하고, 대용량 라벨 없는 데이터에는 비지도 학습을 적용

### • 강화 학습 ( reinforcement learning)

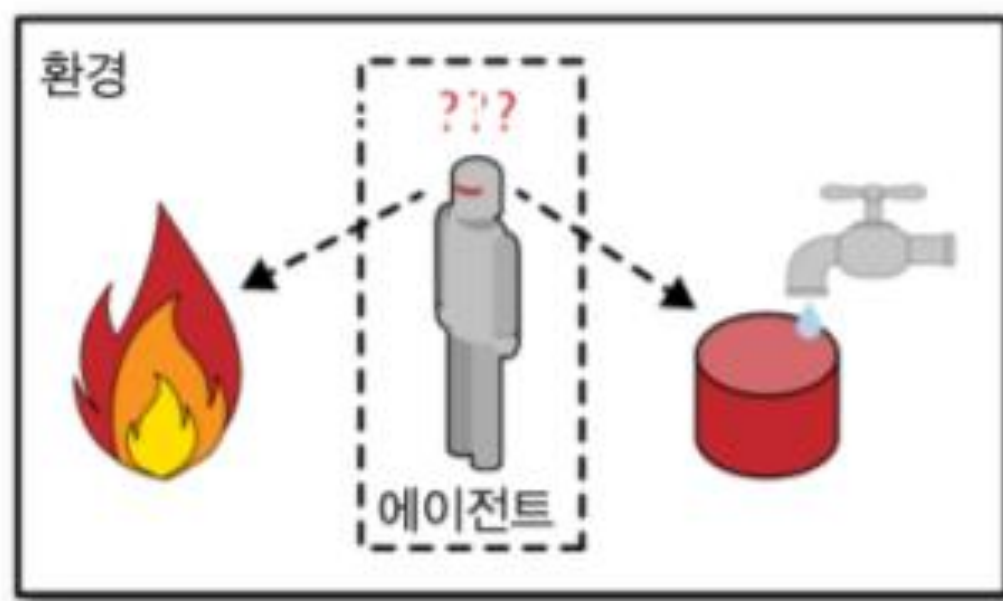
- 주어진 입력에 대응하는 행동을 취하는 시스템에 대해 보상이 주어지게 되며, 이러한 보상을 이용하여 학습하는 방법
- 지도 학습과 달리 주어진 입력에 대한 출력, 즉 정답 행동이 주어지지 않음
- 주요 응용 분야 : 로봇, 게임, 내비게이션 등



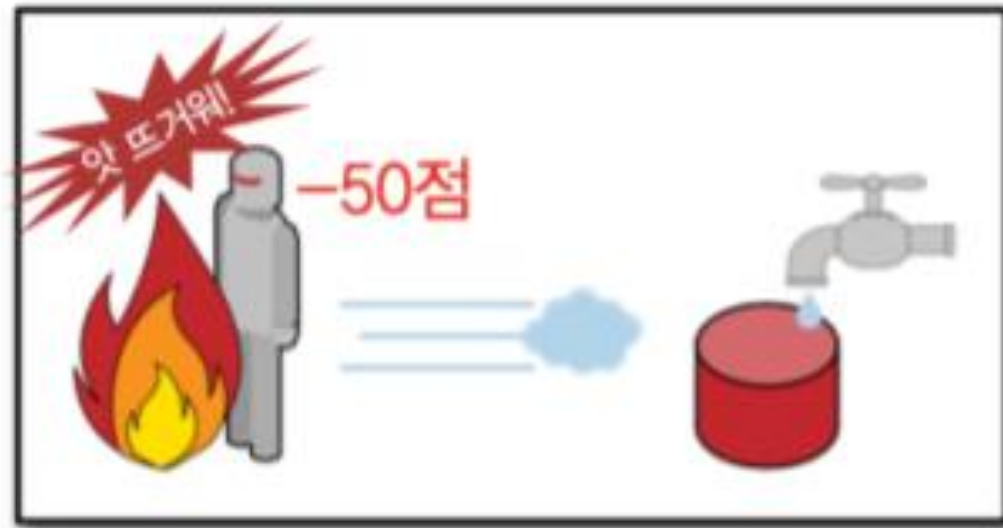
✔ 머신러닝 학습 방법 퀴즈1



✓ 머신러닝 학습 방법 퀴즈2



- 1 관찰
- 2 정책에 따라 행동을 선택

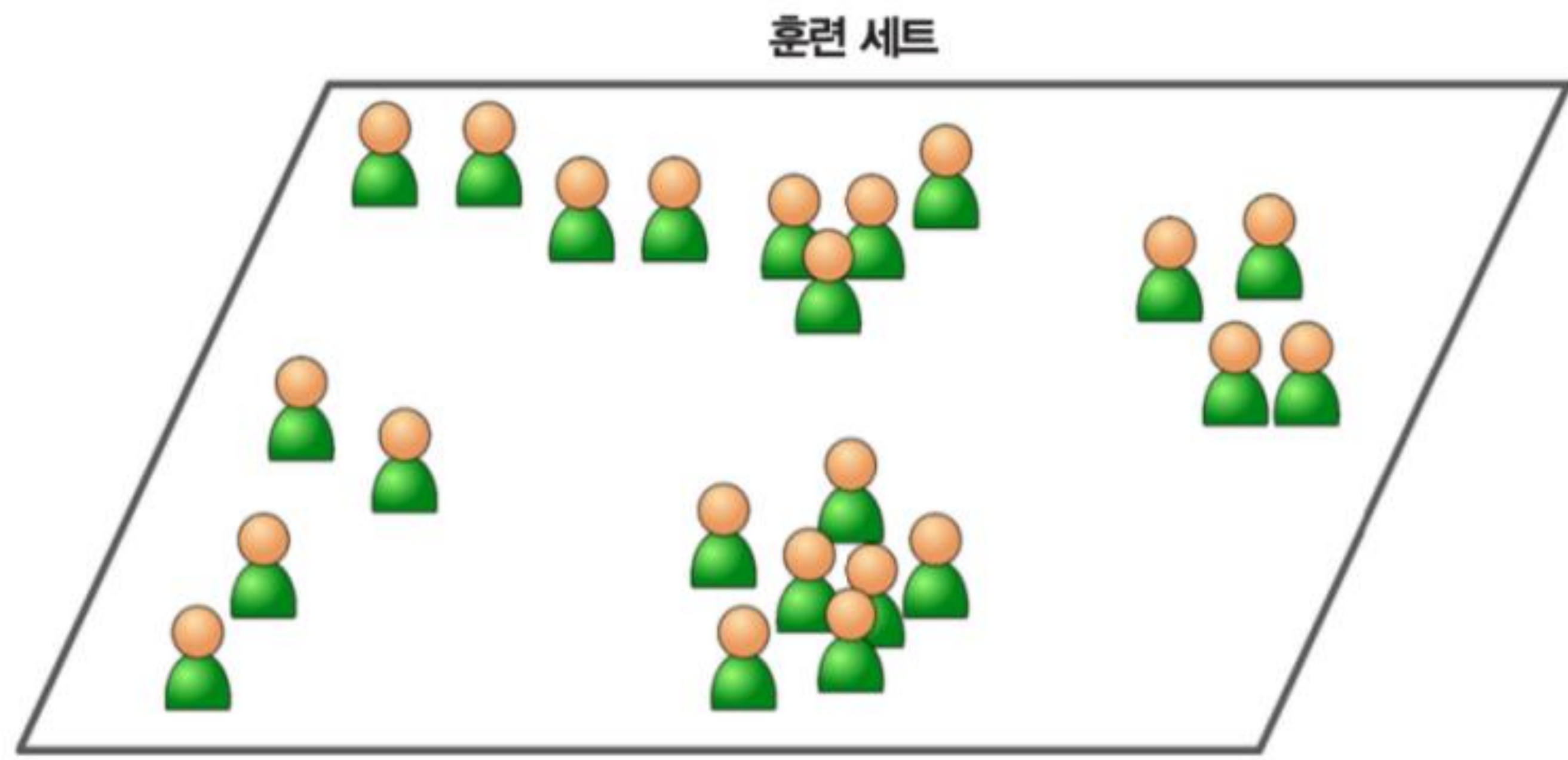


- 3 행동 실행!
- 4 보상이나 벌점을 받음

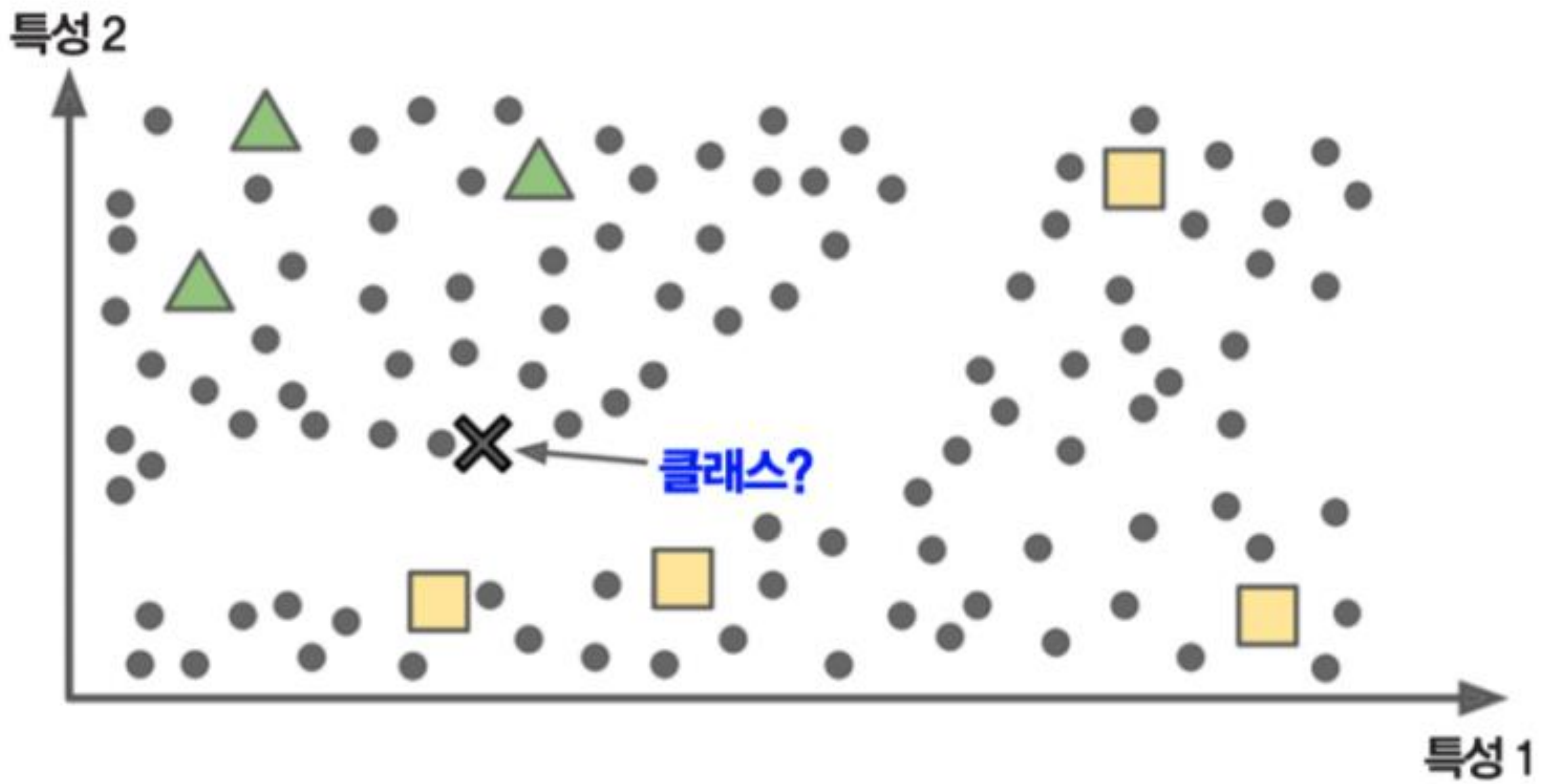


- 5 정책 수정(학습 단계)
- 6 최적의 정책을 찾을 때까지 반복

✔ 머신러닝 학습 방법 퀴즈3

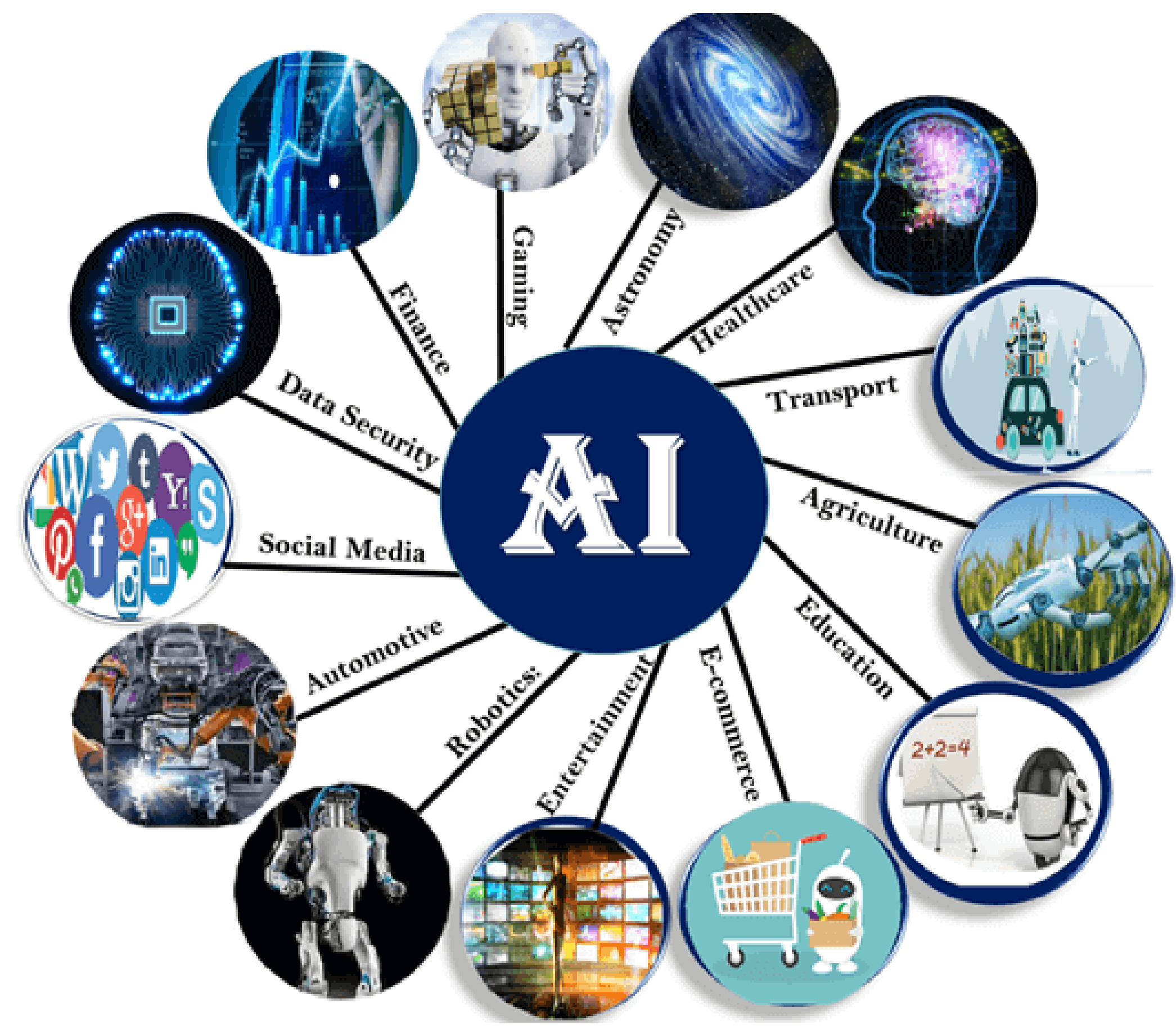


✔ 머신러닝 학습 방법 퀴즈4

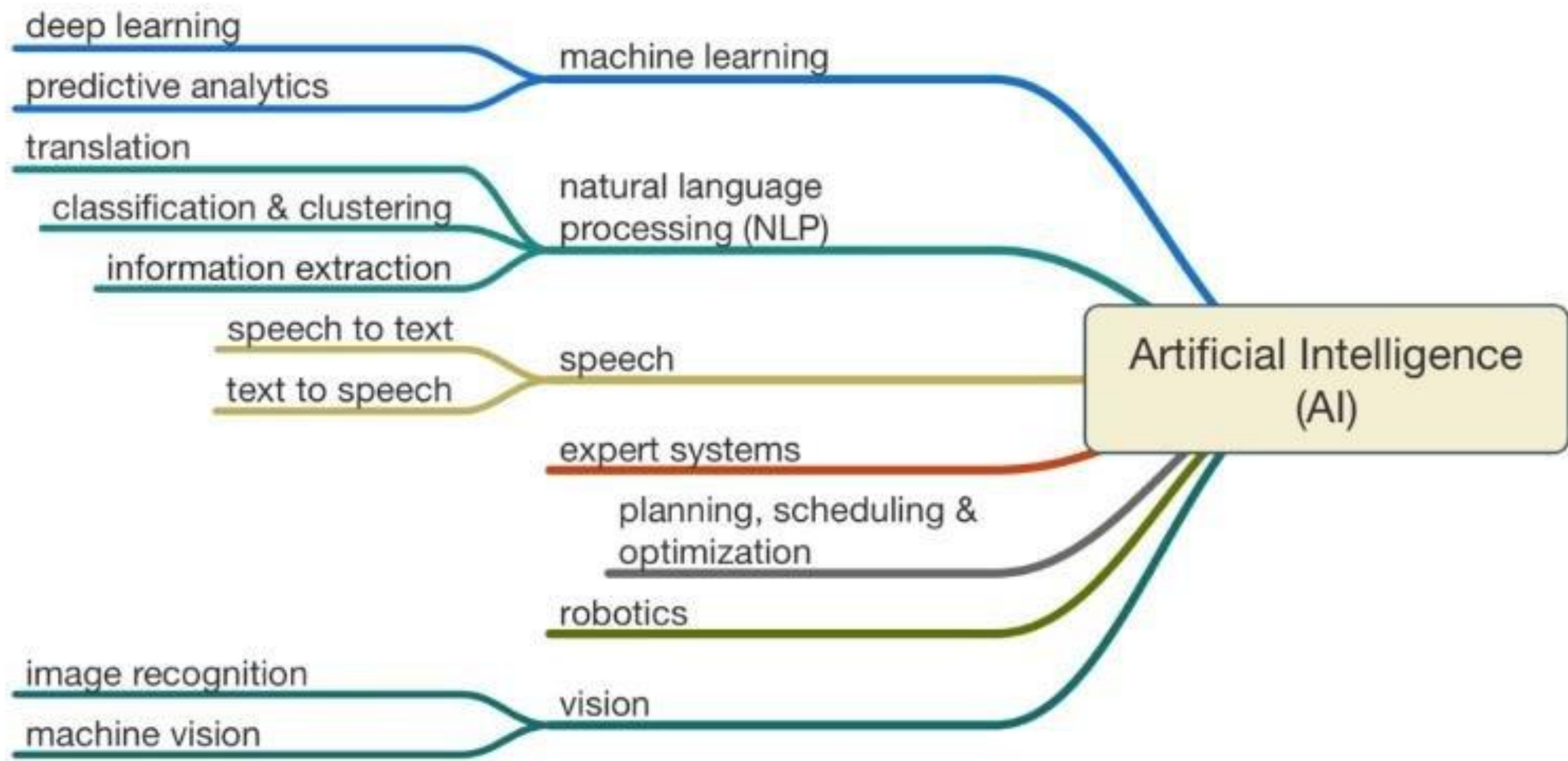




✓ 인공지능 적용범위

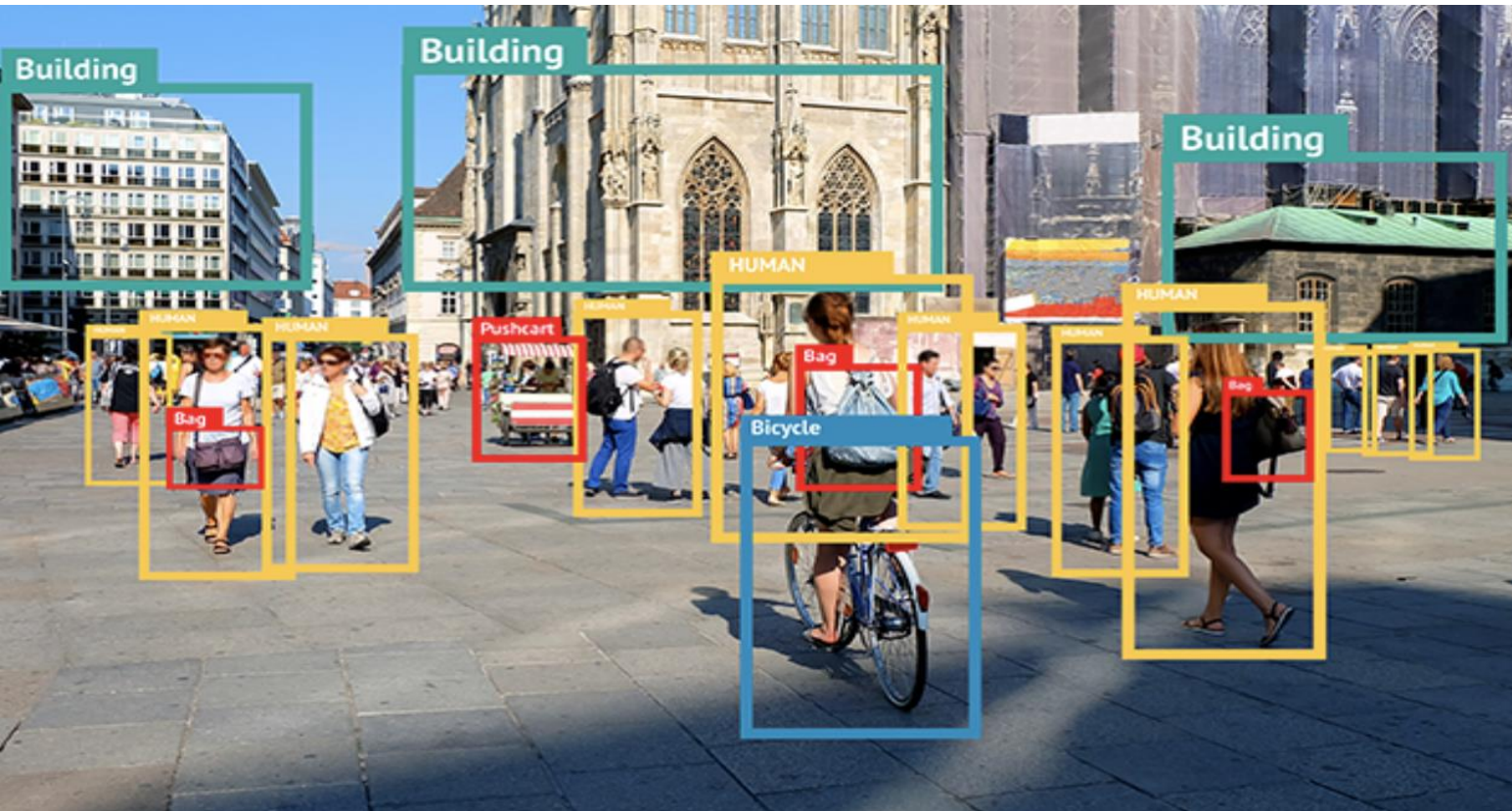


✓ 인공지능 연구/활용 범위





✓ 인공지능 연구/활용 범위 퀴즈



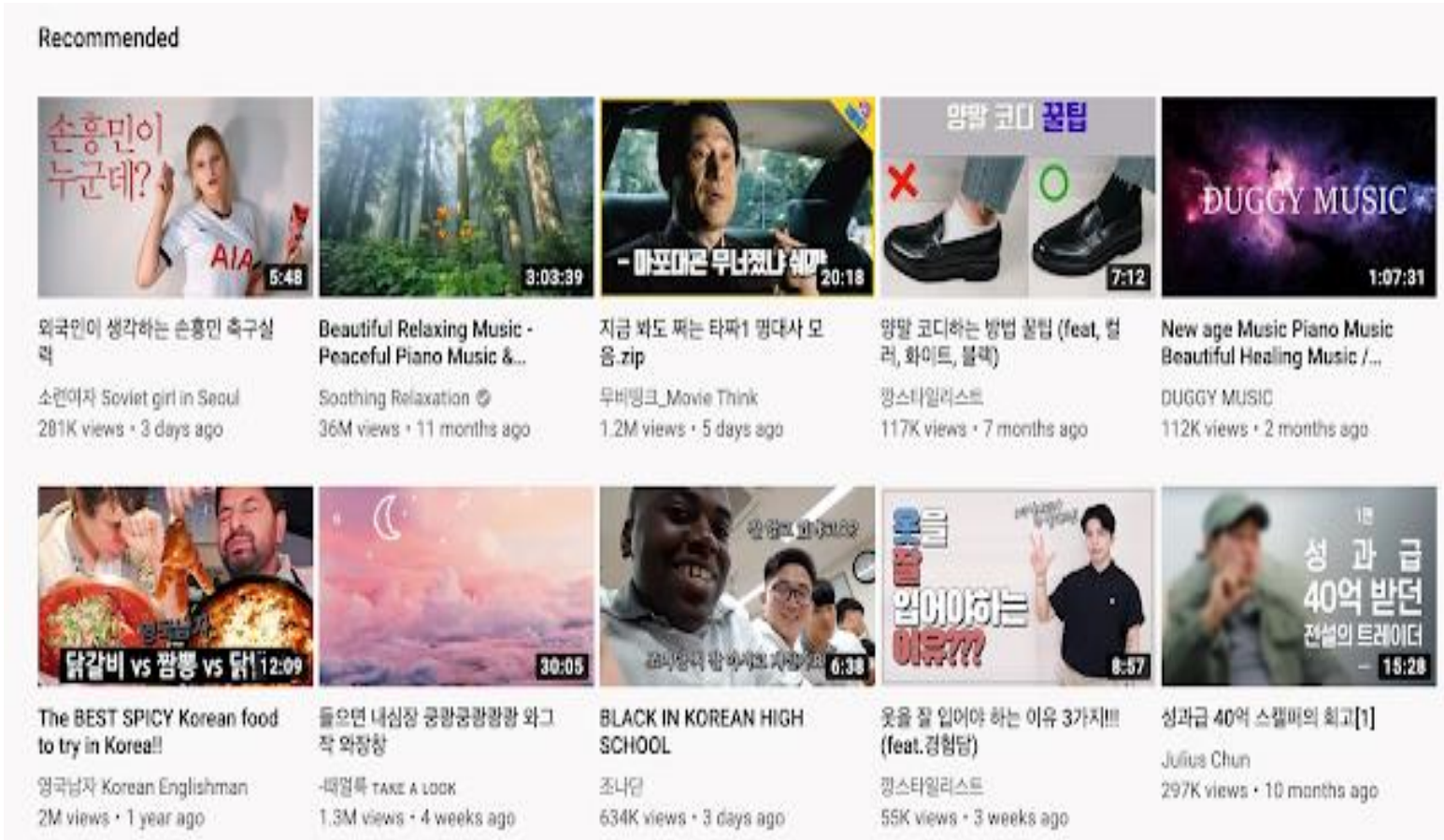
<https://blog.lgcns.com/2232>



<https://ppss.kr/archives/154439>



<https://www.news1.kr/articles/?3715761>



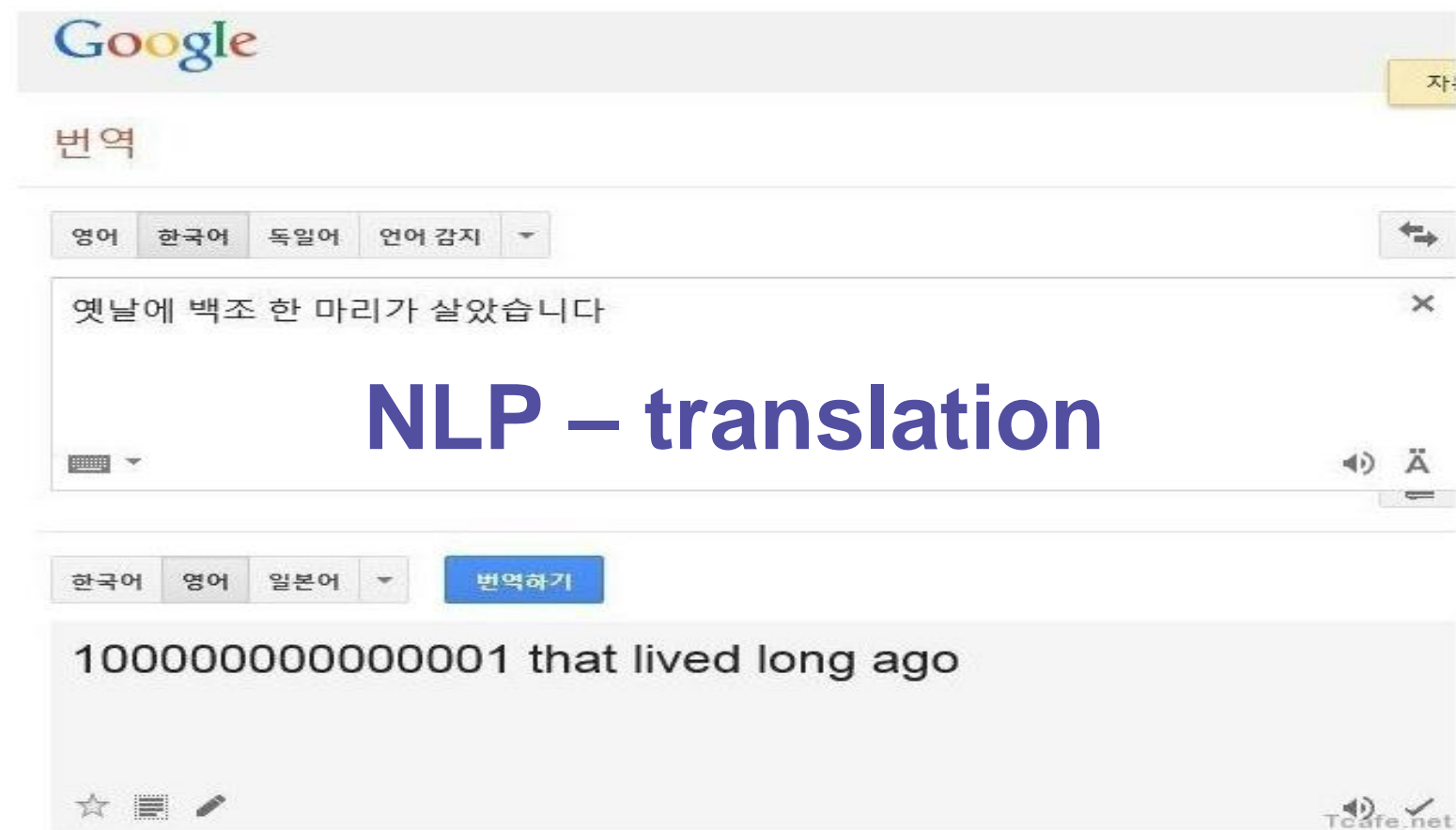
<http://yhs968.blogspot.com/2019/09/part-2-deep-neural-networks-for-youtube.html>



✓ 인공지능 연구/활용 범위 퀴즈



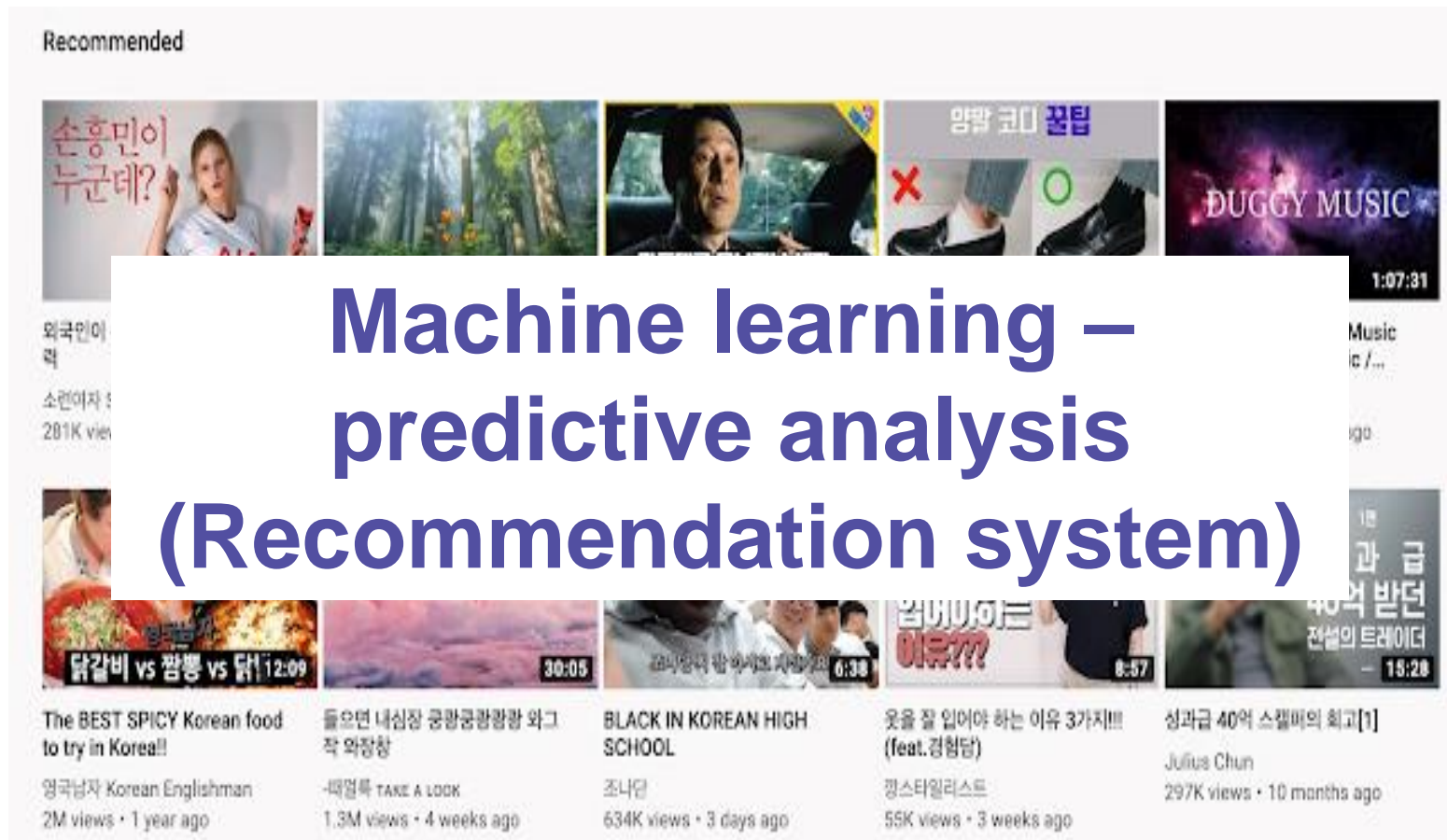
<https://blog.lgcns.com/2232>



<https://ppss.kr/archives/154439>

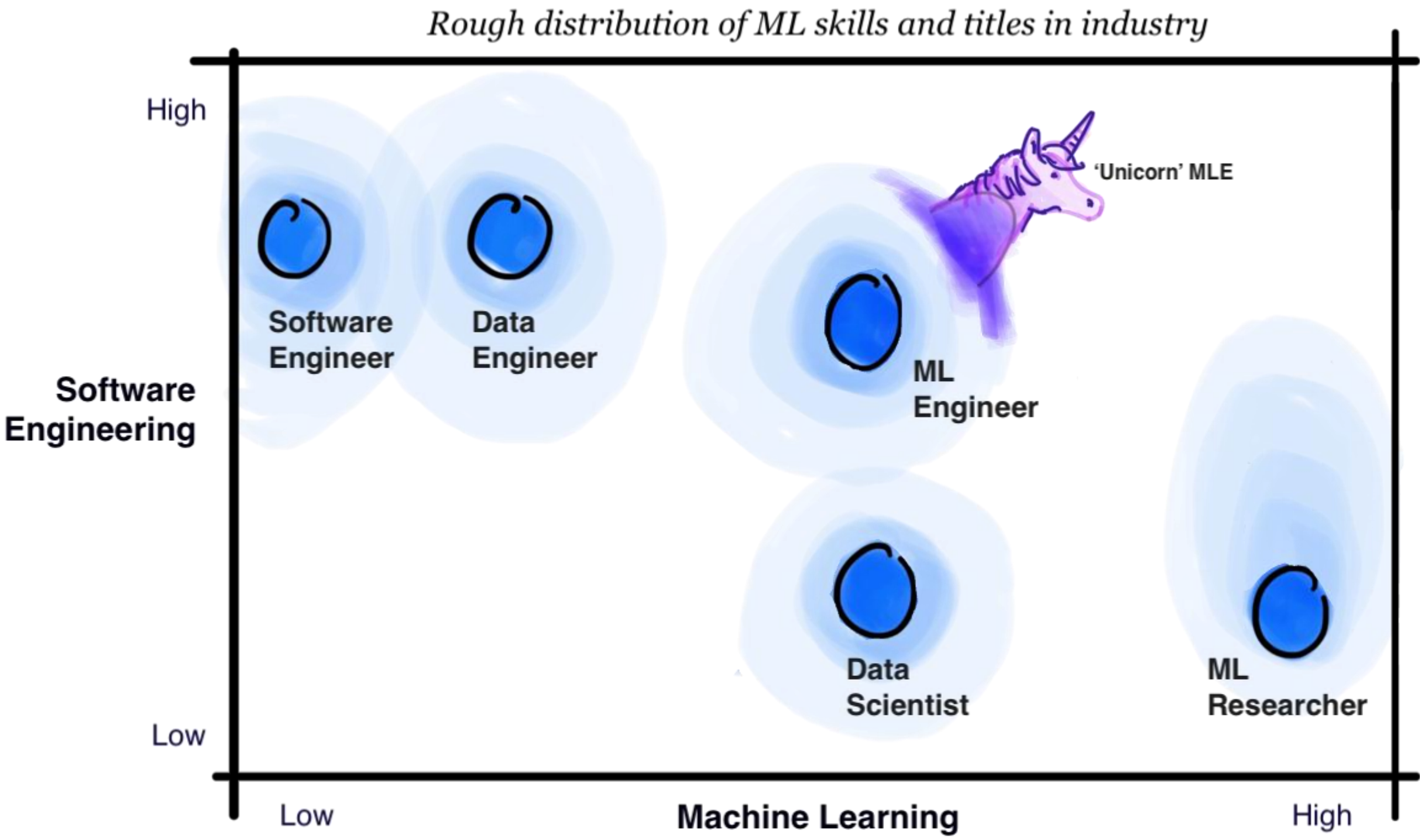


<https://www.news1.kr/articles/?3715761>



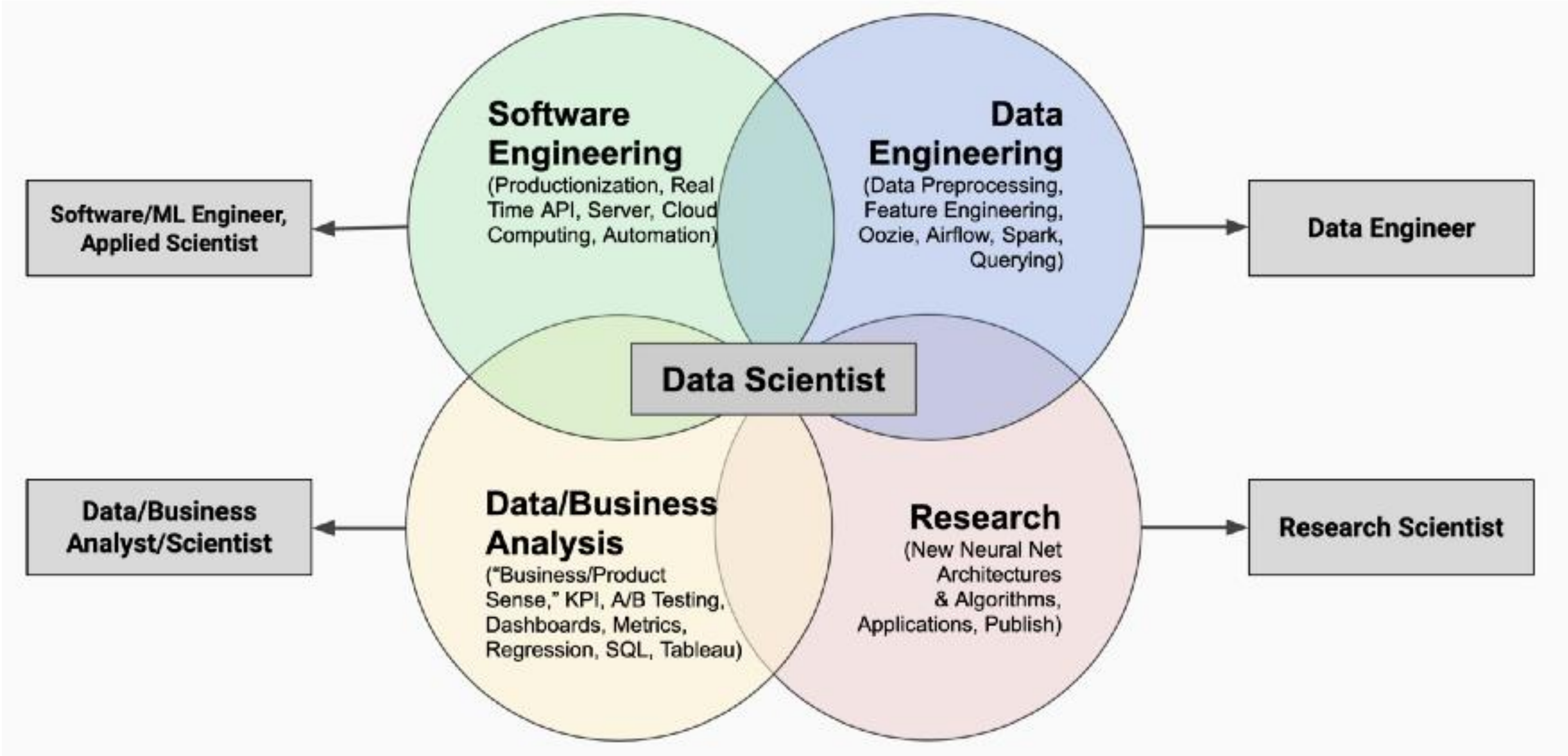
<http://yhs968.blogspot.com/2019/09/part-2-deep-neural-networks-for-youtube.html>

✓ 인공지능 직군





✓ 인공지능 직군



05

# XOR과 MLP



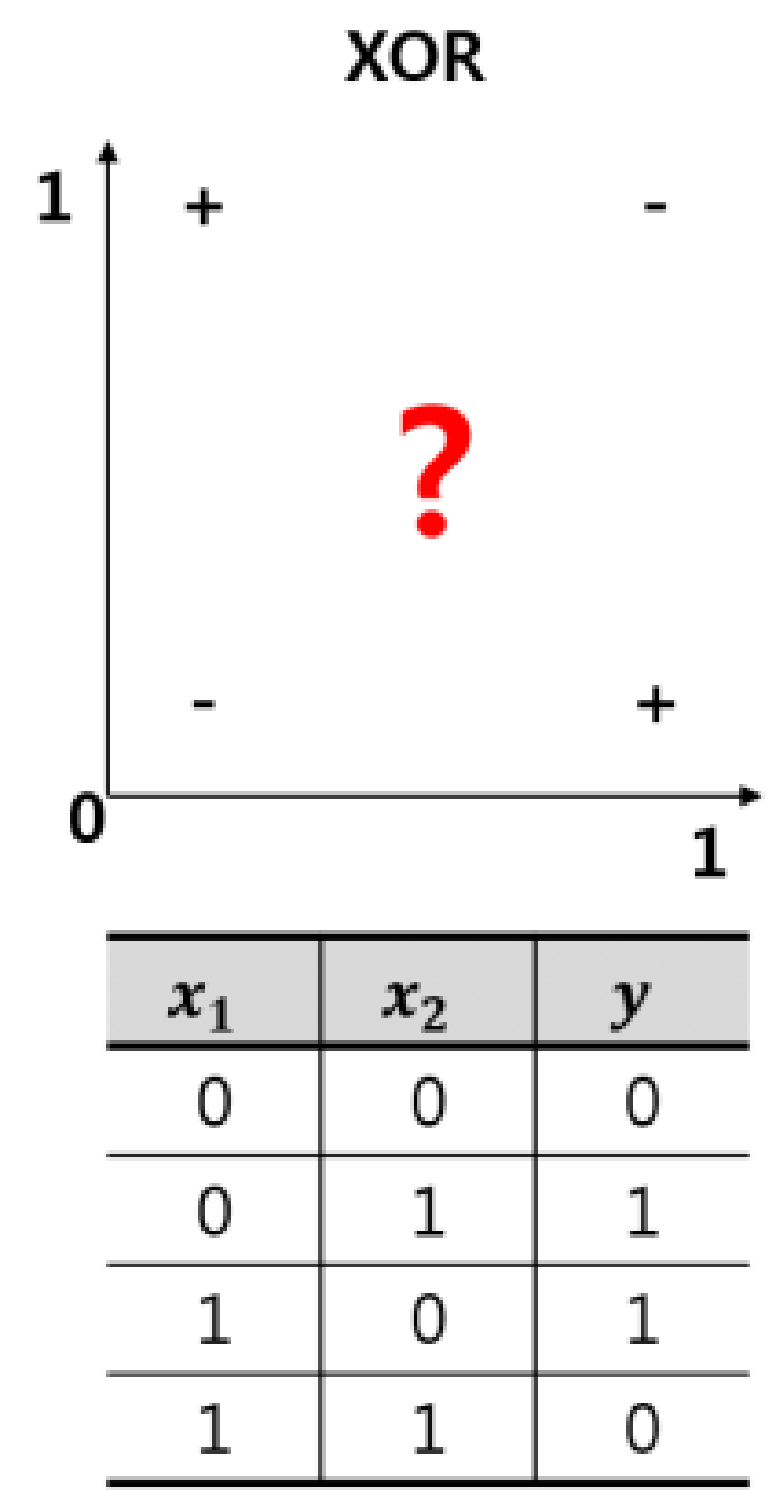
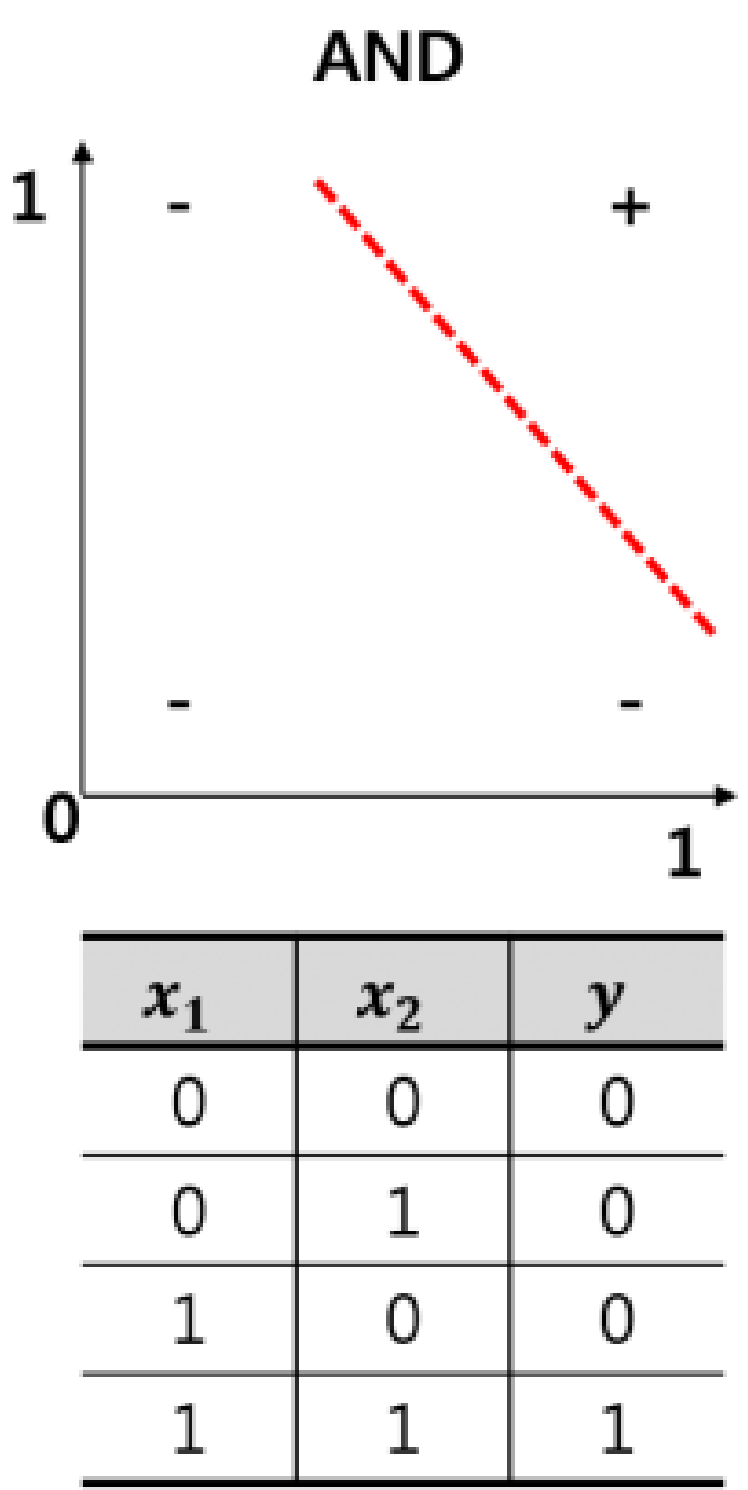
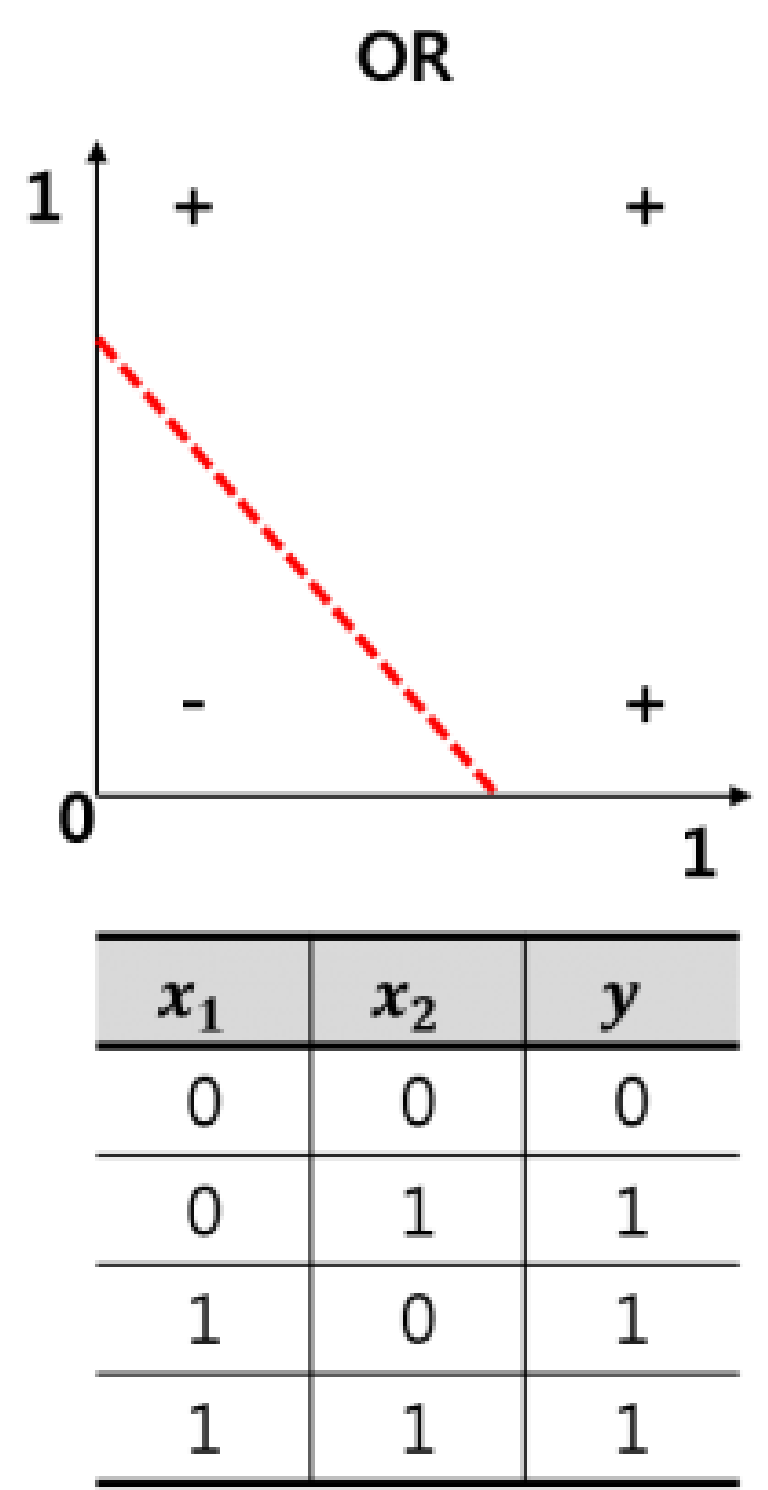
✓ 논리게이트

- 논리 게이트는 디지털 회로를 만드는데 있어 가장 기본적인 요소
- 대부분의 논리 게이트들은 두 개의 입력과 한 개의 출력을 가짐
- 주어진 어떤 순간에 모든 단자는 두 개의 조건 중의 하나인데, 이것을 서로 다른 전압으로 표현하면 전압이 높음(1)과 낮음(0)임
- 기본 논리 게이트에는 AND, OR, XOR, NOT, NAND, 그리고 NOR 등 모두 6개의 종류가 있음

게이트	기호	의미	진리표	논리식															
AND		입력신호가 모두 1일 때 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	0	1	0	0	1	1	1	$Y = A \cdot B$ $Y = AB$
A	B	Y																	
0	0	0																	
0	1	0																	
1	0	0																	
1	1	1																	
OR		입력신호 중 1개만 1이어도 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	1	$Y = A + B$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	1																	
NOT		입력된 정보를 반대로 변환하여 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	Y	0	1	1	0	$Y = A'$ $Y = \overline{A}$									
A	Y																		
0	1																		
1	0																		
BUFFER		입력된 정보를 그대로 출력	<table><tr><th>A</th><th>Y</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	A	Y	0	0	1	1	$Y = A$									
A	Y																		
0	0																		
1	1																		
NAND		NOT + AND, 즉 AND의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	1	1	0	1	1	1	0	$Y = \overline{A \cdot B}$ $Y = \overline{AB}$
A	B	Y																	
0	0	1																	
0	1	1																	
1	0	1																	
1	1	0																	
NOR		NOT + OR, 즉 OR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	0	$Y = \overline{A + B}$
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	0																	
XOR		입력신호가 모두 같으면 0, 한 개라도 틀리면 1출력	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	Y	0	0	0	0	1	1	1	0	1	1	1	0	$Y = A \oplus B$ $Y = AB + \overline{A}\overline{B}$
A	B	Y																	
0	0	0																	
0	1	1																	
1	0	1																	
1	1	0																	
XNOR		NOT + XOR, 즉 XOR의 부정	<table><tr><th>A</th><th>B</th><th>Y</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	Y	0	0	1	0	1	0	1	0	0	1	1	1	$Y = A \odot B$ $Y = \overline{A \oplus B}$ $Y = AB + \overline{A}\overline{B}$
A	B	Y																	
0	0	1																	
0	1	0																	
1	0	0																	
1	1	1																	

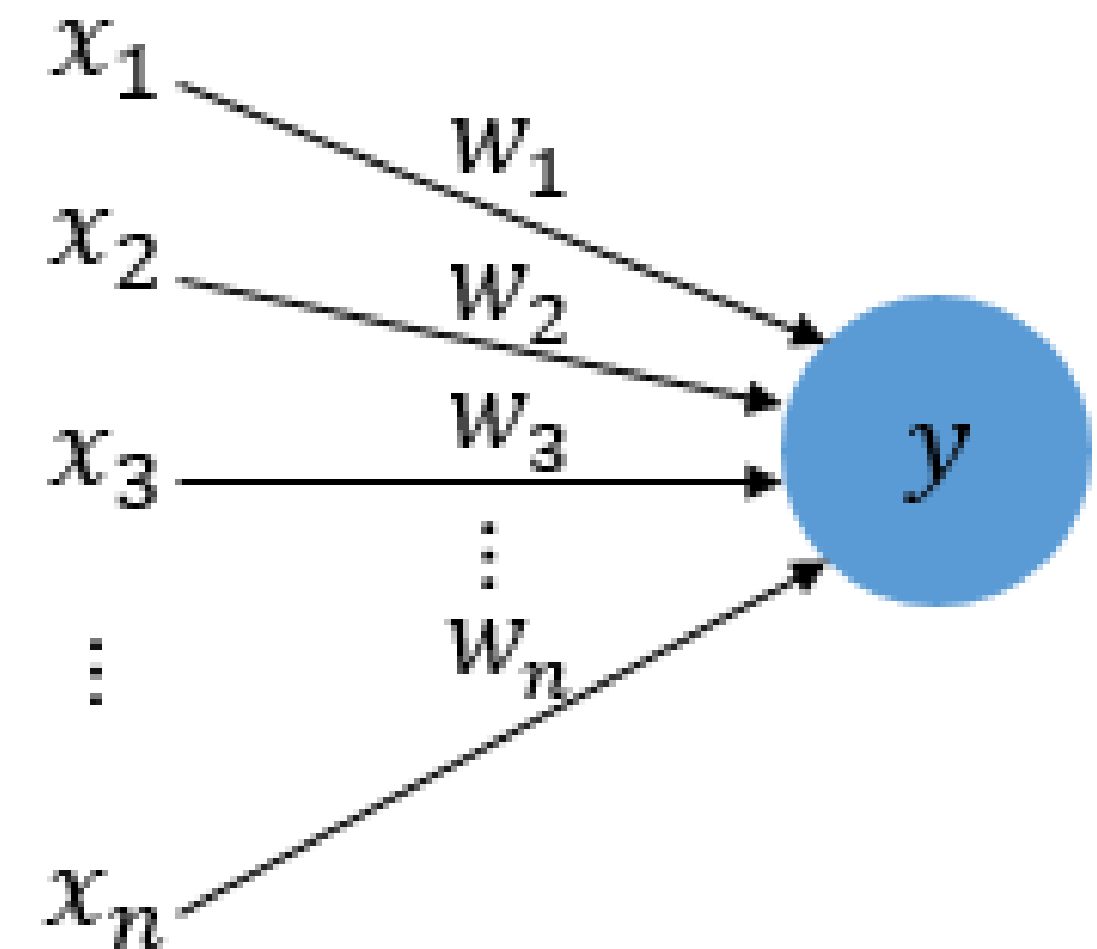
✓ XOR 문제

- 여러 게이트의 특성을 기반으로 +, -를 구분하는 선형 경계를 찾는 문제
- 하나의 퍼셉트론으로 해결하는 방법을 강구



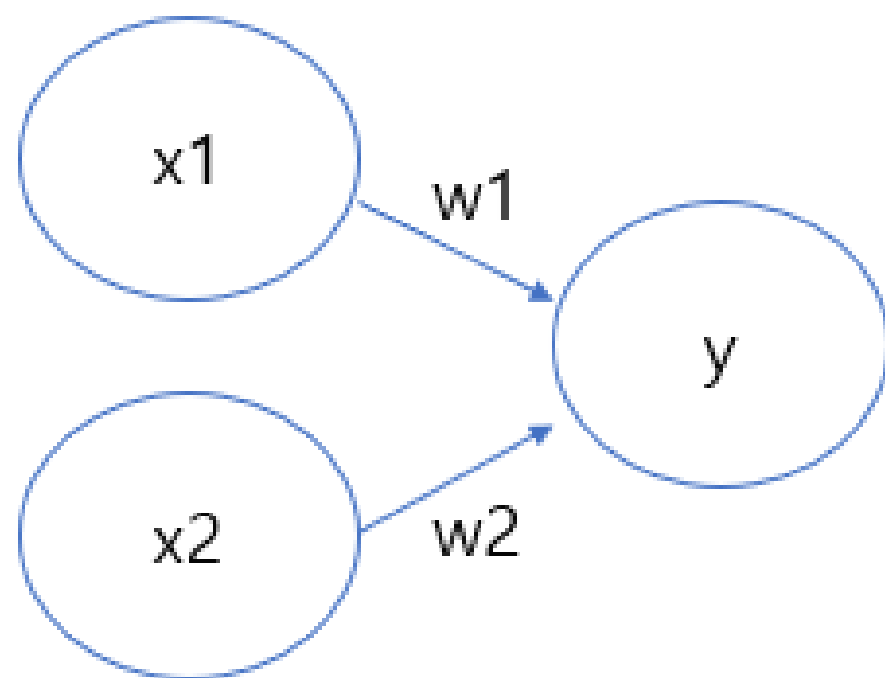
## ✓ 퍼셉트론

- 퍼셉트론(Perceptron)은 프랑크 로젠블라트(Frank Rosenblatt)가 1957년에 제안한 초기 형태의 인공 신경망
- 다수의 입력으로부터 하나의 결과를 내보내는 알고리즘
- 퍼셉트론은 실제 뇌를 구성하는 신경 세포 뉴런의 동작과 유사
- 신경 세포 뉴런은 가지돌기에서 신호를 받아들이고, 이 신호가 일정치 이상의 크기를 가지면 축삭 돌기를 통해서 신호를 전달



## ✓ 단층 퍼셉트론

- 아래 그림에서 원을 뉴런 혹은 노드라고 부름
- 입력 신호가 뉴런에 보내질 때는 각각 고유한 가중치가 곱해짐
- $(w_1, w_2)$  가중치는 각 신호가 결과에 주는 영향력을 조절하는 요소로 작용
- 즉, 가중치가 클 수록 해당 신호가 그만큼 더 중요함
- 그 신호를 받은 다음 뉴런은 이전 뉴런에서 보내온 신호의 총합이 정해진 한계를 넘어설 때만 1을 출력
- 그 한계를 보통 임계 값 (theta)이라 함
- 가중치를 갖는 층이 한 층이기 때문에 단층 퍼셉트론이라고 부름



$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases} \quad \xrightarrow{\theta \rightarrow b} \quad y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

$\theta \rightarrow b$   
치환



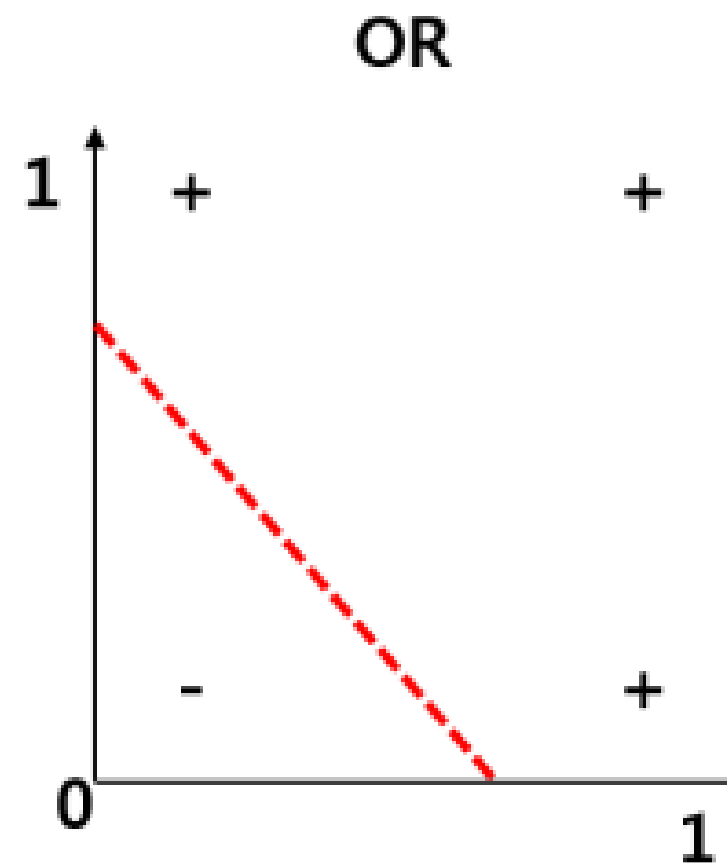
## ✓ 게이트 문제(코드)

- 단층 퍼셉트론(선형분류)로 OR 파이썬 코드로 모델링

### 코드

```
import numpy as np

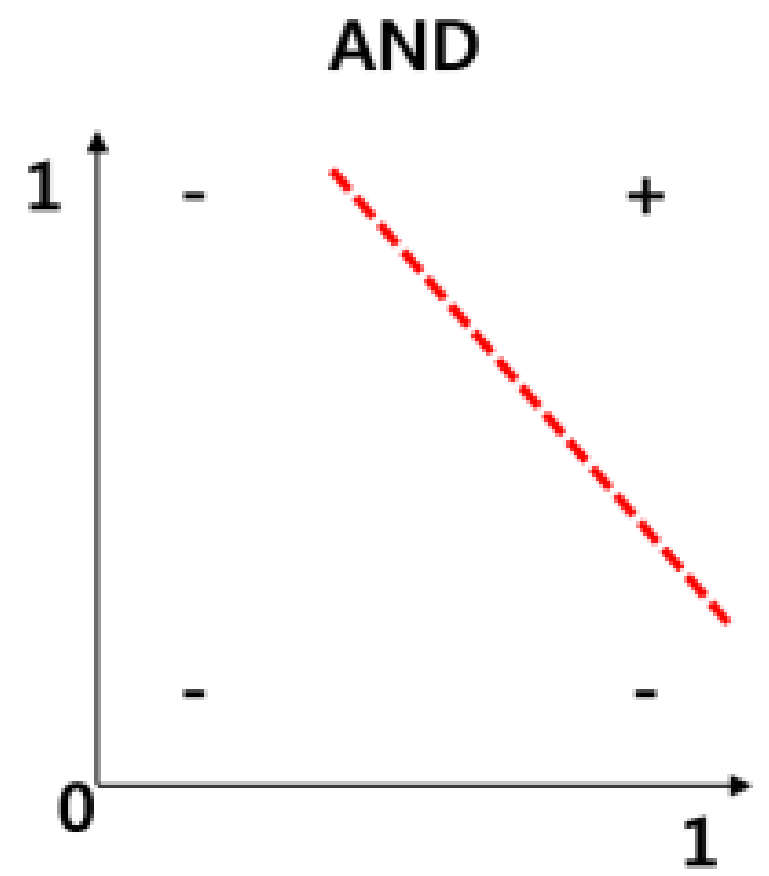
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    temp = np.sum(w*x)+b
    if temp <= 0:
        return 0
    else:
        return 1
```



$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	1

✔ 게이트 문제 퀴즈(코드)

- 단층 퍼셉트론(선형분류)로 AND 파이썬 코드로 모델링



$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

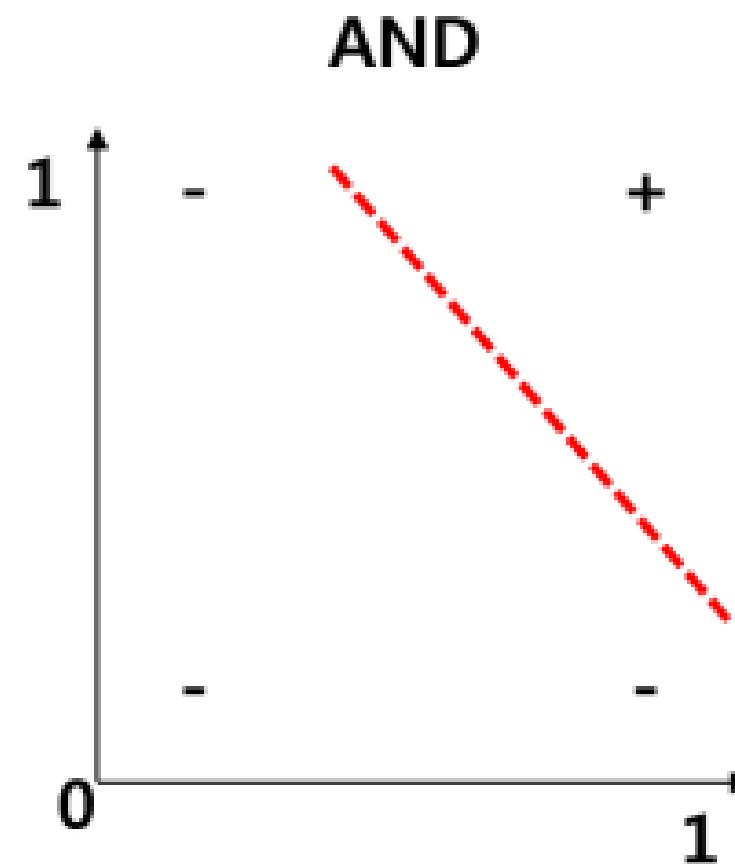
## ✓ 게이트 문제 퀴즈(코드)

- 단층 퍼셉트론(선형분류)로 AND 파이썬 코드로 모델링

### 코드

```
import numpy as np

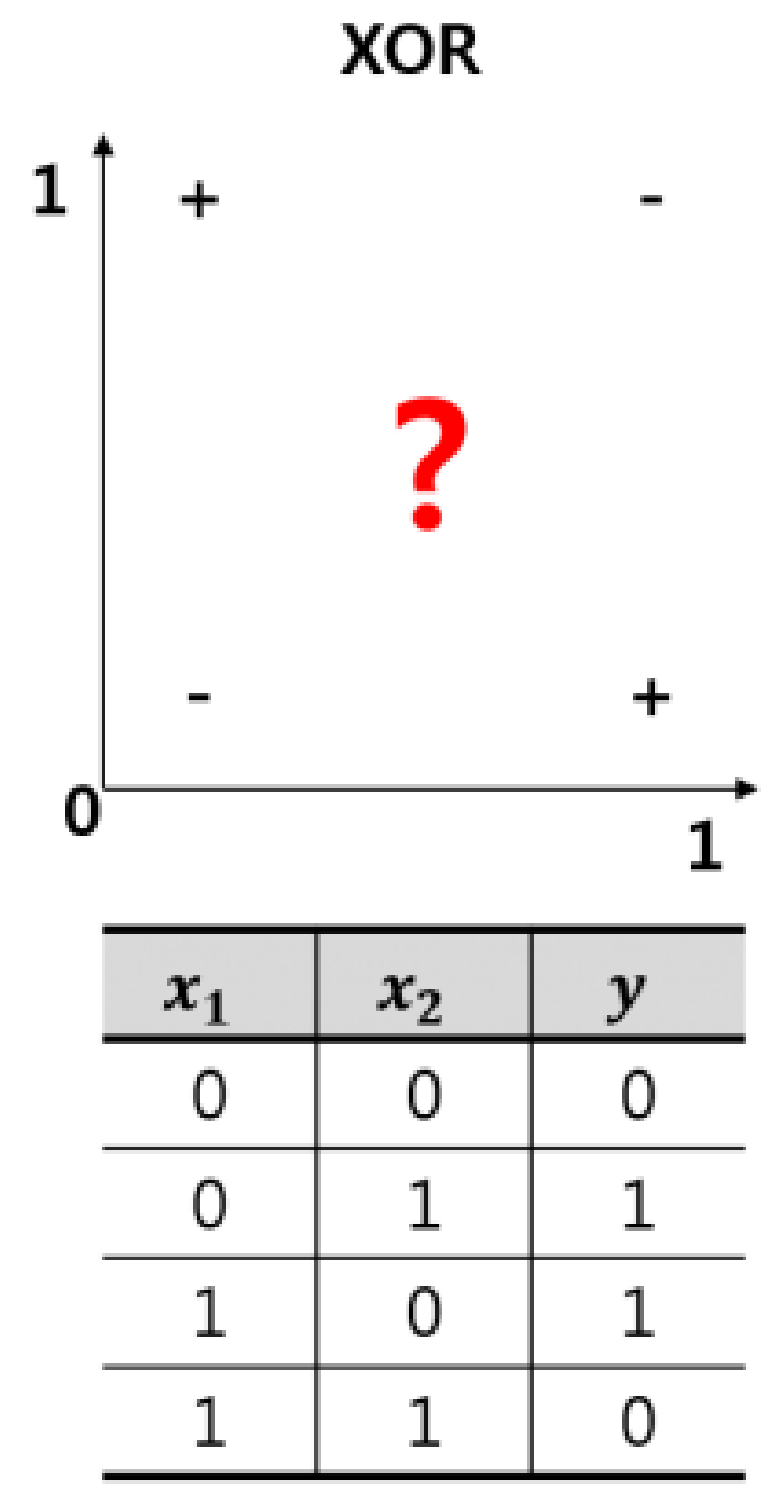
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = 0.7
    temp = np.sum(w*x)+b
    if temp <= 0:
        return 0
    else:
        return 1
```



$x_1$	$x_2$	$y$
0	0	0
0	1	0
1	0	0
1	1	1

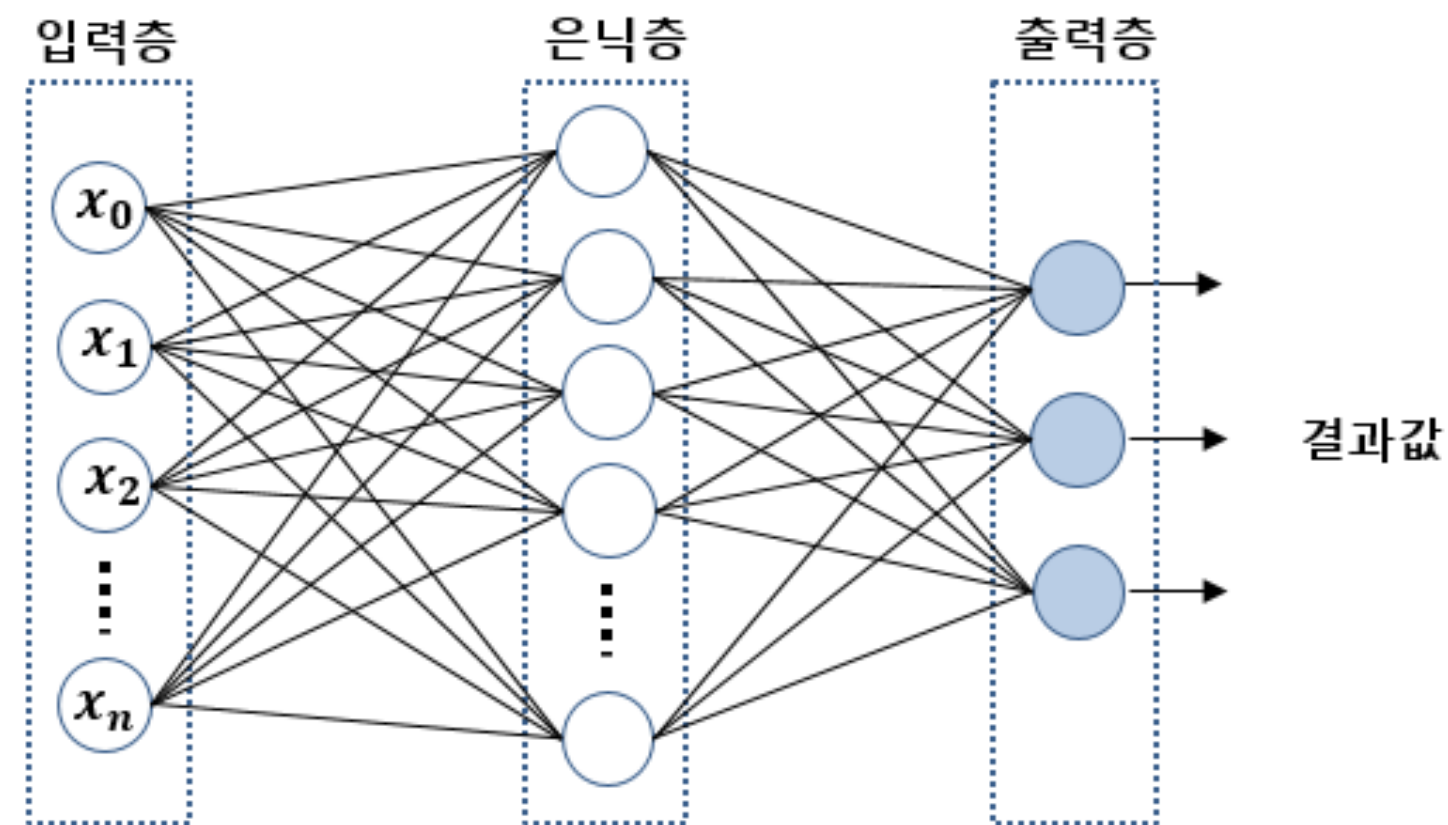
✓ 게이트 문제

- 단층 퍼셉트론(선형분류)로 XOR 파이썬 코드로 모델링



## ✓ 다층 퍼셉트론

- 단층 퍼셉트론은 입력층과 출력층만 존재하지만, 다층 퍼셉트론은 중간에 은닉층(hidden layer)라 불리는 층을 더 추가함
- 선형적으로만 풀던 게이트 문제를 여러 층을 추가함으로써 비선형적으로 풀 수 있게 됨
- 따라서, XOR 문제를 해결
- 은닉층은 2개일 수도, 수십, 수백개일 수도 있음
- 은닉층이 2개 이상인 신경망을 심층 신경망(Deep Neural Network, DNN)이라 함



## ✓ XOR 게이트 문제(코드)

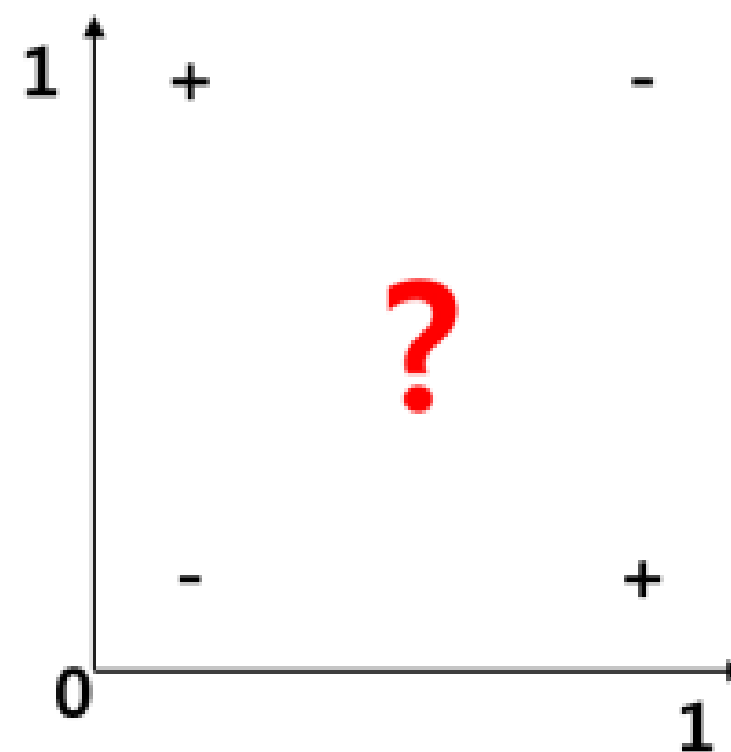
- 단층 퍼셉트론(선형분류)로 XOR 파이썬 코드로 모델링

코드

```
import numpy as np

def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7
    temp = np.sum(w*x)+b
    if temp <= 0:
        return 0
    else:
        return 1
```

XOR



$x_1$	$x_2$	$y$
0	0	0
0	1	1
1	0	1
1	1	0

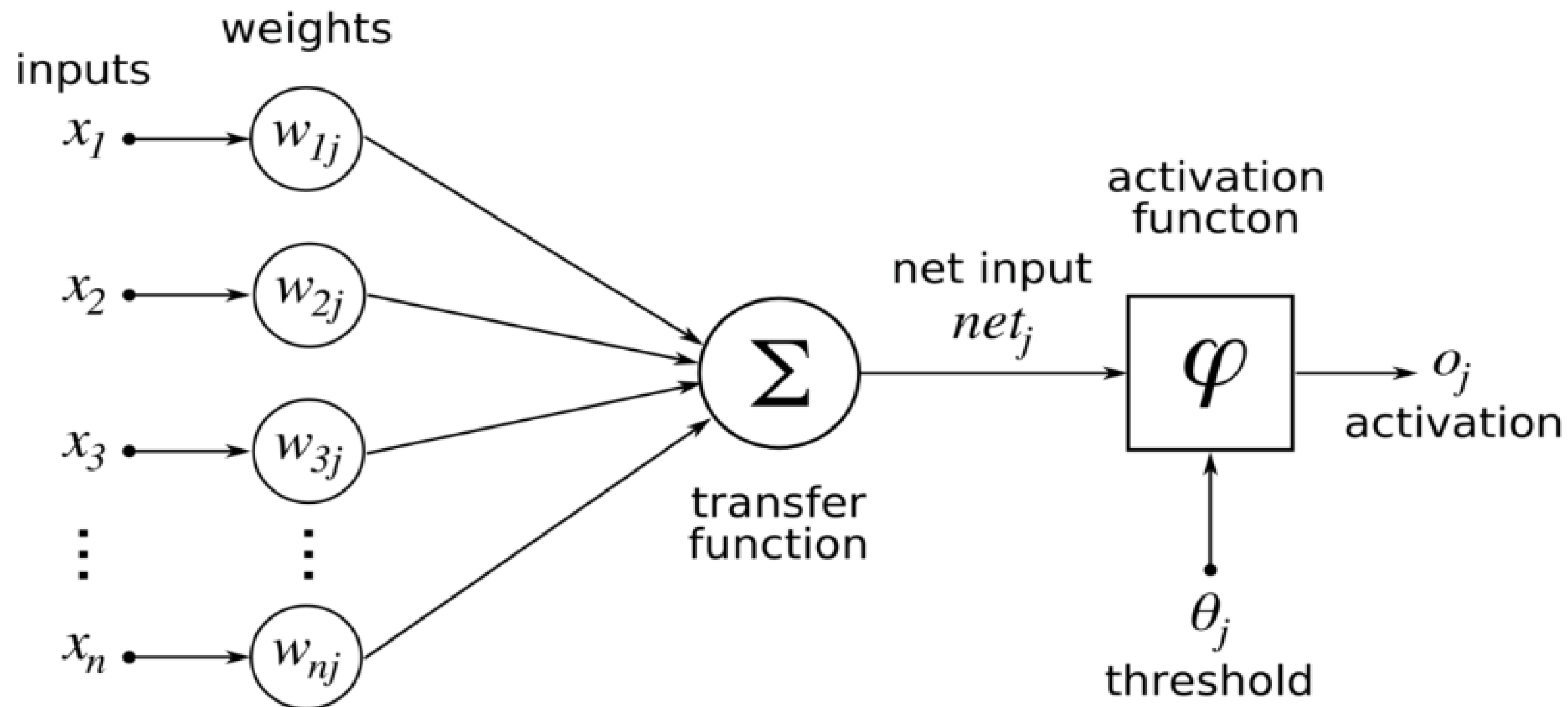
코드

```
import numpy as np

def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

## ✓ 신경망, 활성화 함수

- 입력, 출력층 외 은닉층이 추가된 아래 그림을 신경망이라고 부름
- 활성화 함수(Activation function)은 가중치가 곱해진 신호의 총합이 활성화를 일으키는지, 즉 임계값을 넘는지 판단함

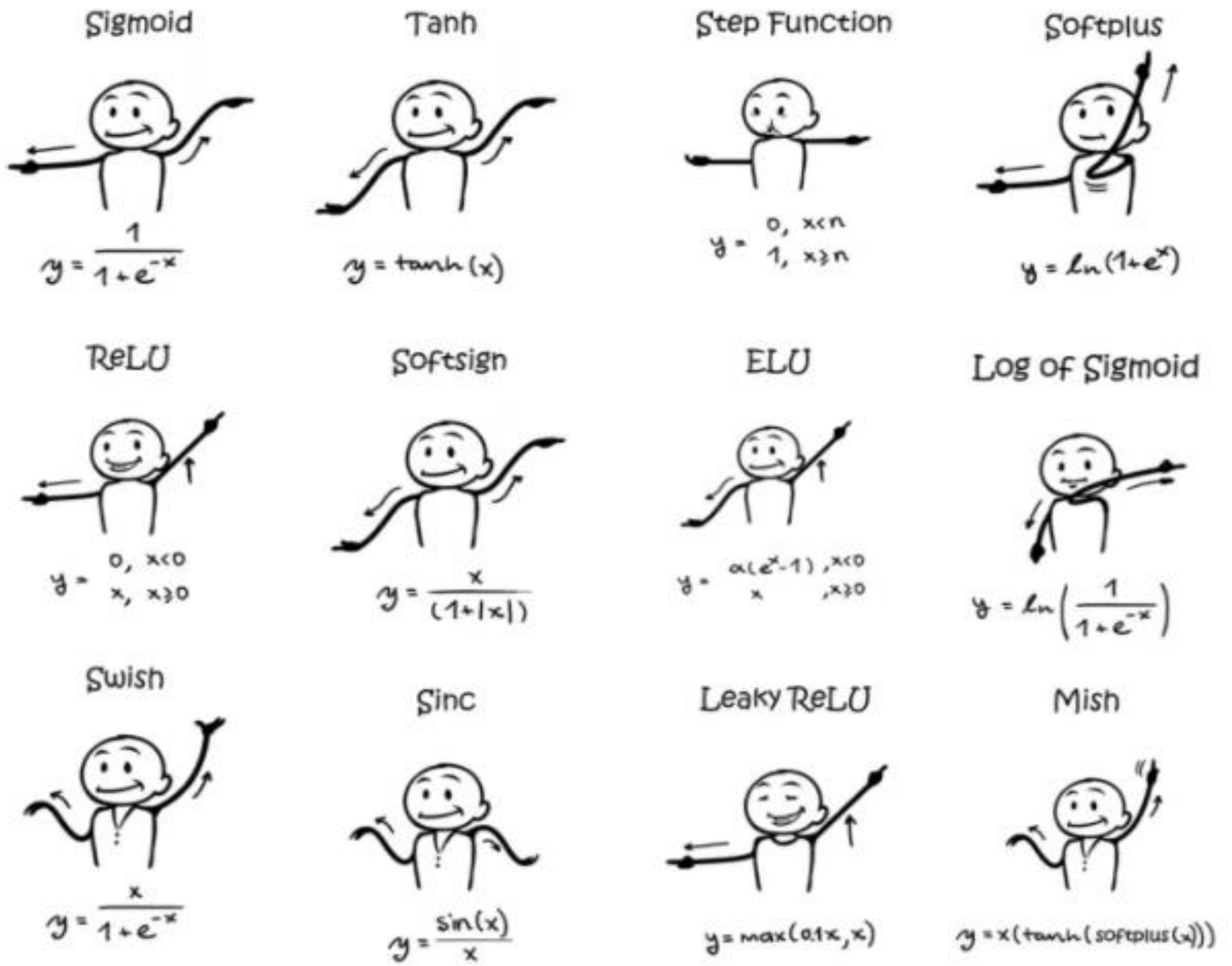


## ✓ 활성화 함수 종류

- 활성화 함수는 이전 층(layer)의 결과값을 변환하여 다른 층의 뉴런으로 신호를 전달하는 역할
- 활성화 함수가 필요한 이유는 모델의 복잡도를 올리기 위함
- 비선형 문제를 해결하기 위해 단층 퍼셉트론을 쌓는 방법을 이용했는데 은닉층(hidden layer)를 무작정 쌓기만 한다고 해서 비선형 문제를 해결할 수 있는 것은 아님
- 활성화 함수를 사용하면 입력값에 대한 출력값이 비선형(nonlinear)적으로 나오므로 선형분류기를 비선형분류기로 만들 수 있음



✓ 활성화함수 종류



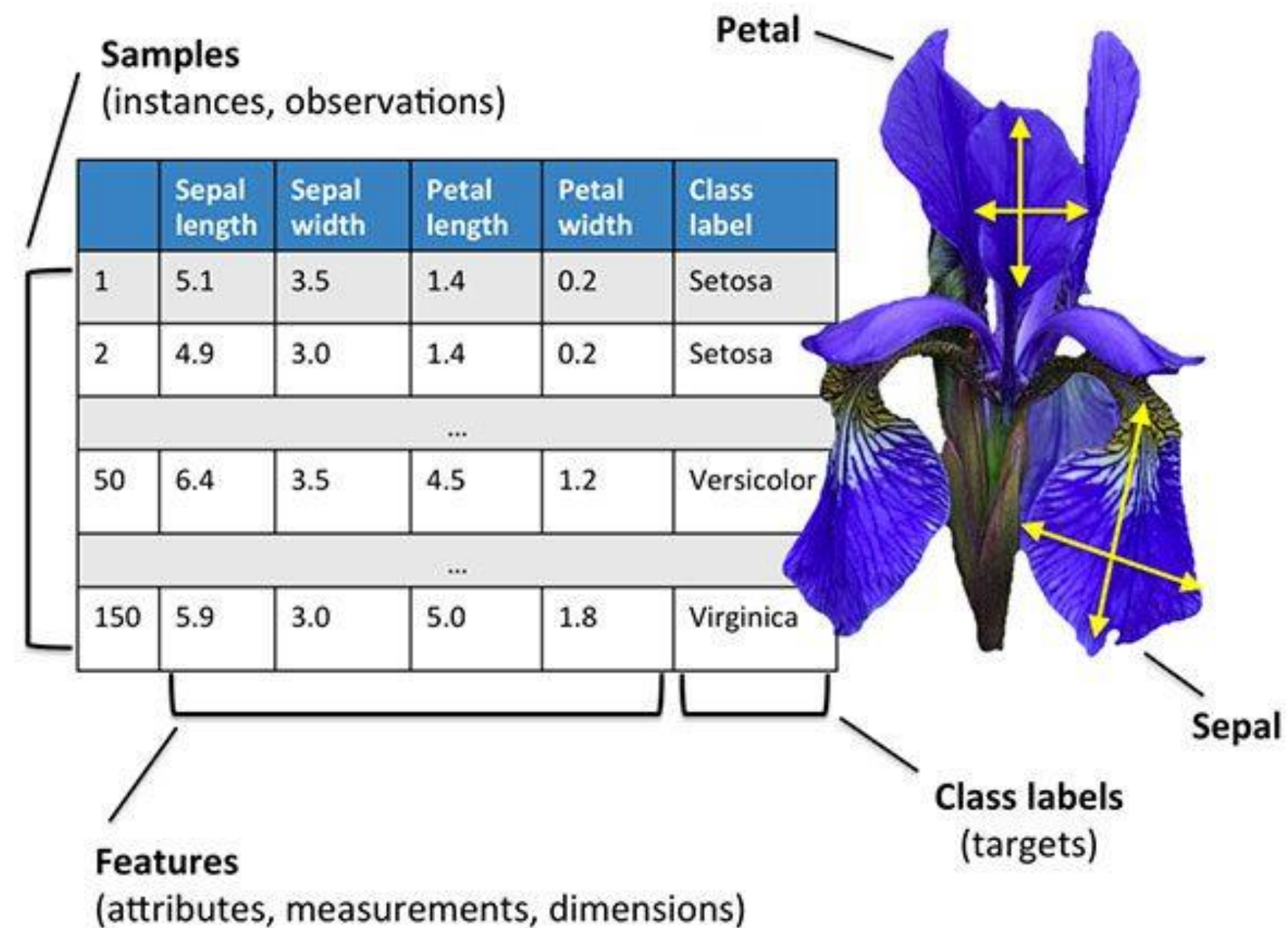
✓ 활성화함수 종류

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = \max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Copyright © Sebastian Raschka 2016  
(<http://sebastianraschka.com>)

## ✓ MLP Python 예제(코드)

- iris 데이터셋 MLP를 이용한 이진 분류 실습
- Python, numpy를 이용한 실습



## ✓ 손실함수

- 손실함수란 신경망을 학습시키는 과정에서 훈련용 샘플과 모델의 결과 값을 특정 함수를 통해 비교하며, 이 함수를 칭함
- 적절한 구조를 가진 신경망 모델의 가중치를 변경시켜서, 최종적으로 우리가 원하는 함수를 높은 정확도로 구현하고자 함
- 이때, 가중치를 변경시키는 과정이 바로 ‘학습’

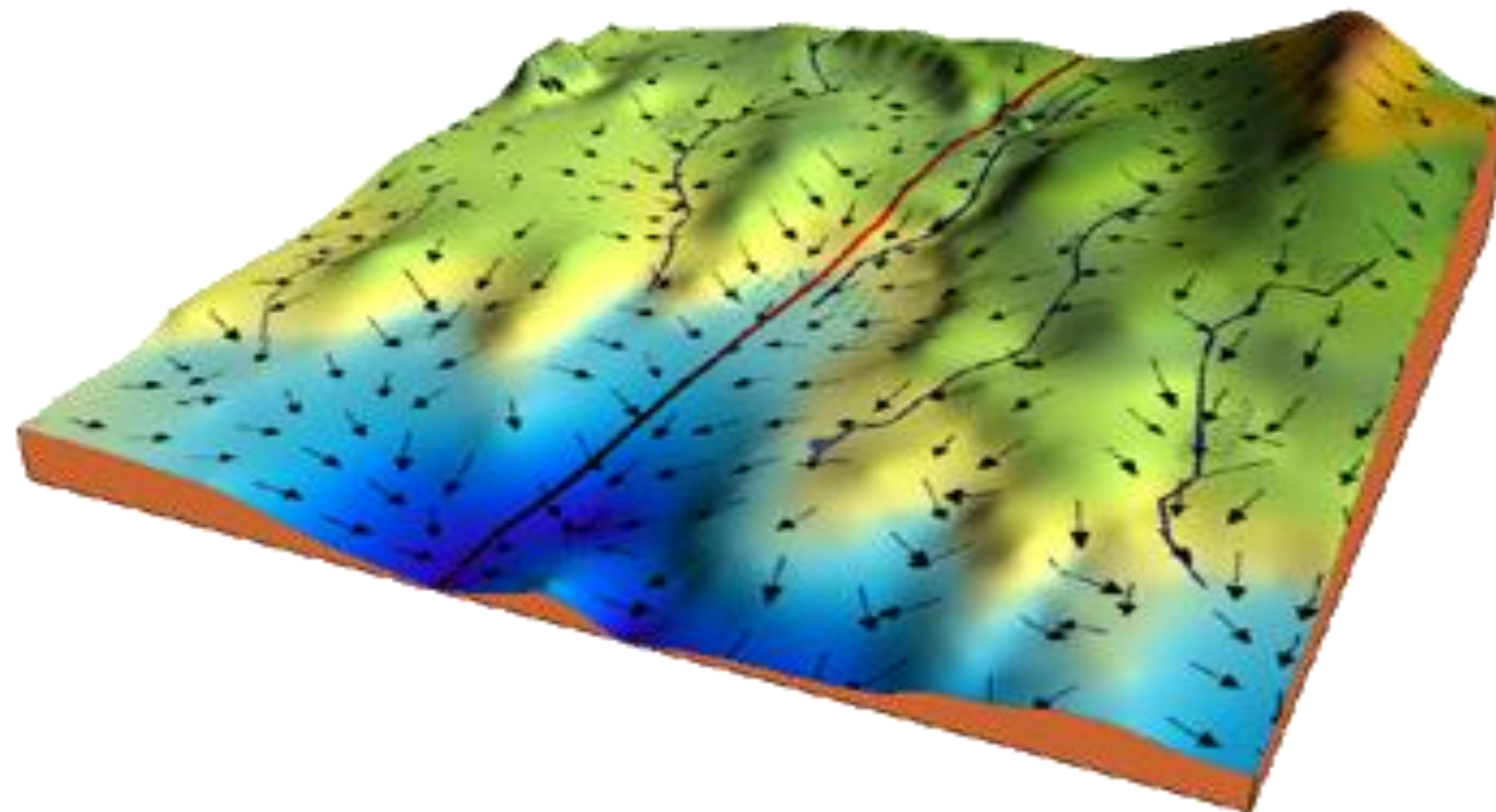
손실함수 종류

- ① MSE(Mean Squared Error)
- ② RMSE(Root Mean Squared Error)
- ③ Binary Cross Entropy
- ④ Categorical Cross Entropy
- ⑤ Sparse Category Cross Entropy
- ⑥ Focal Loss



## ✓ 경사하강법

- 경사하강법이란 1차 근삿값 발견용 최적화 알고리즘
- 기본 개념은 함수의 기울기(경사)를 구하고 경사의 절댓값이 낮은 쪽으로 계속 이동시켜 극값에 이를 때까지 반복시키는 것
- 보통 함수의 최솟값을 찾고자 할 때, 미분계수를 구함 → 컴퓨터에서는 미분을 안 이용함
- 실제 분석에서(특히, 딥러닝 알고리즘을 활용하는 경우) 보게 되는 함수들은 형태가 굉장히 복잡해서 미분계수와 그 근을 계산하기 어려운 경우가 많음
- 미분계수 계산 과정을 컴퓨터로 구현하는 것보다, 경사하강법을 구현하는 것이 훨씬 쉬움
- 데이터의 양이 매우 큰 경우 경사하강법과 같은 순차적인 방법이 계산량 측면에서 훨씬 효율적



06

# Q&A



# 크레딧

/\* elice \*/

코스 매니저

콘텐츠 제작자

한서우

강사

한서우

감수자

디자이너

한서우

# 연락처

TEL

070-4633-2015

WEB

<https://elice.io>

E-MAIL

[contact@elice.io](mailto:contact@elice.io)

