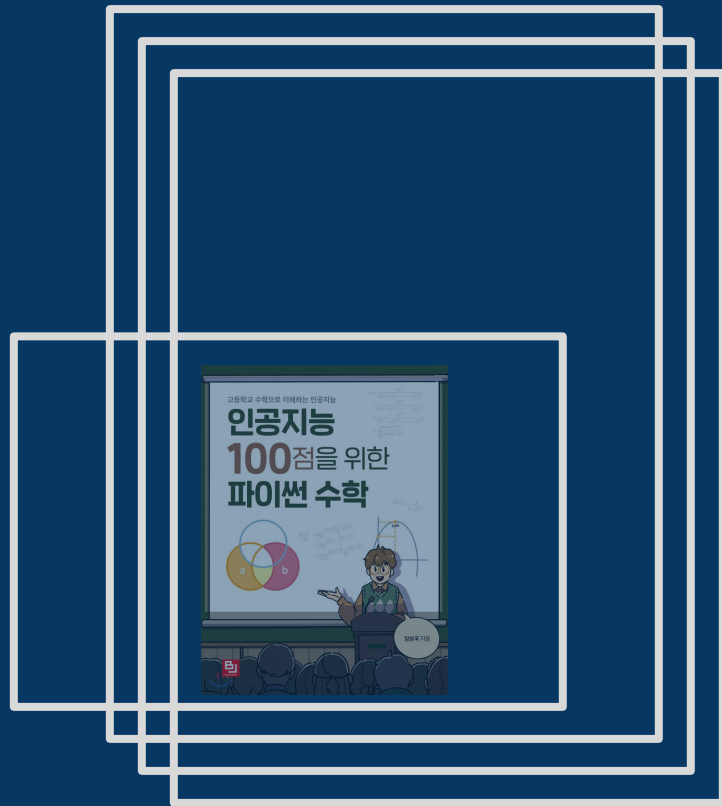
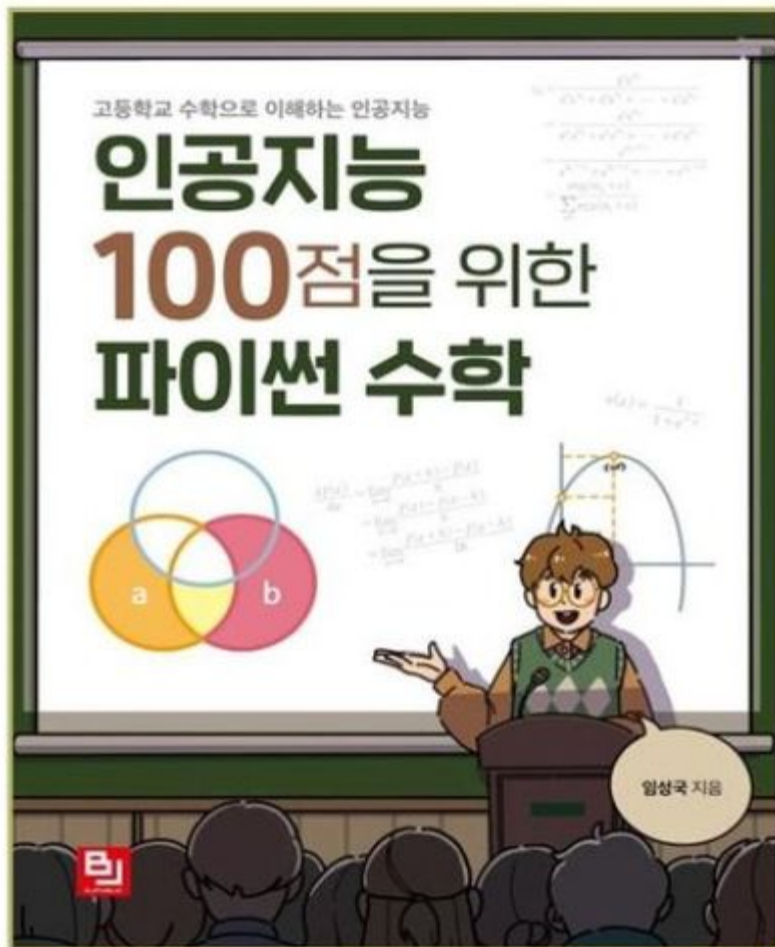


# 파이썬으로 구현해보는 딥러닝

임성국

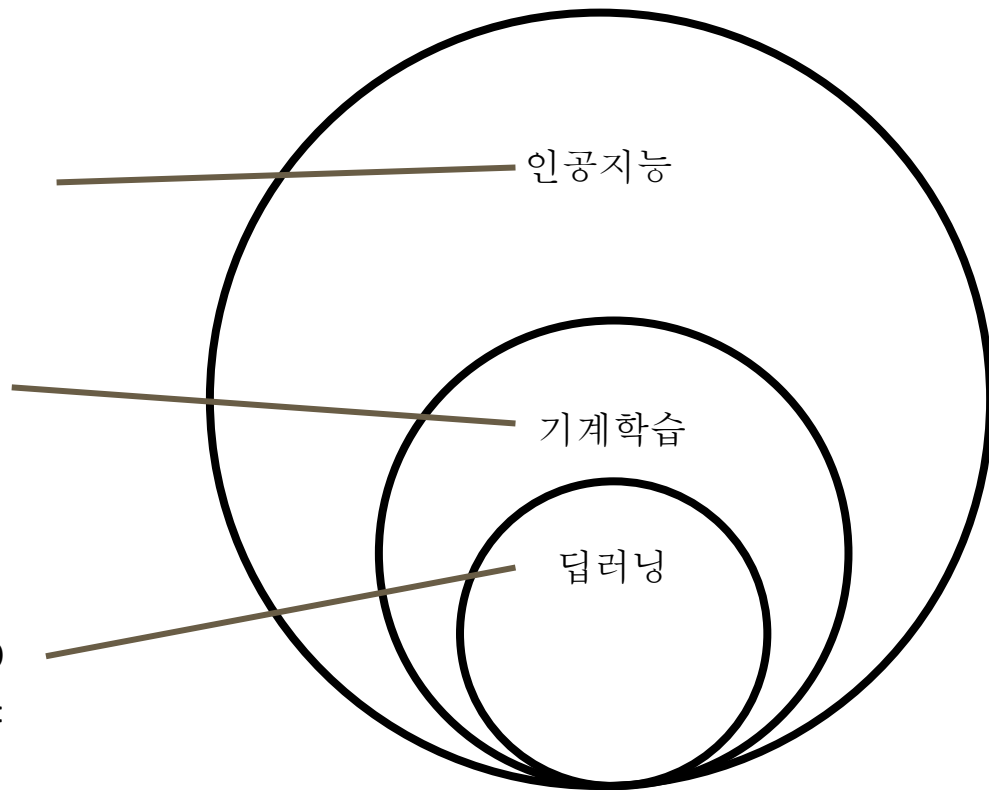




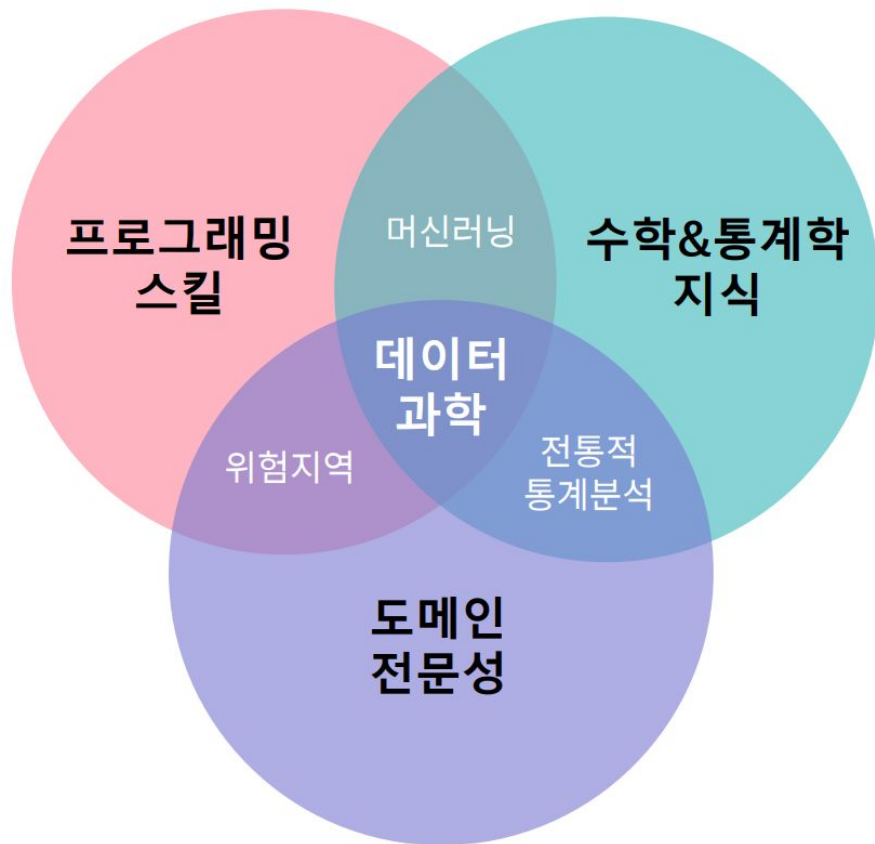
# 인공지능, 머신러닝, 딥러닝

## ● 인공지능, 기계학습, 딥러닝 구별

- 인공지능 : 기계가 인간의 행동을 모방, 기존 프로그래밍 포함
- 기계학습 : 인공지능 중 코드로 명시하지않은 동작을 **데이터**를 이용해 **스스로 학습**
- 딥러닝 : 기계학습 중 DNN(Deep Neural Network)를 이용한 분야



## 🕒 데이터과학



# 인공지능, 머신러닝, 딥러닝

## ● 학습의 종류

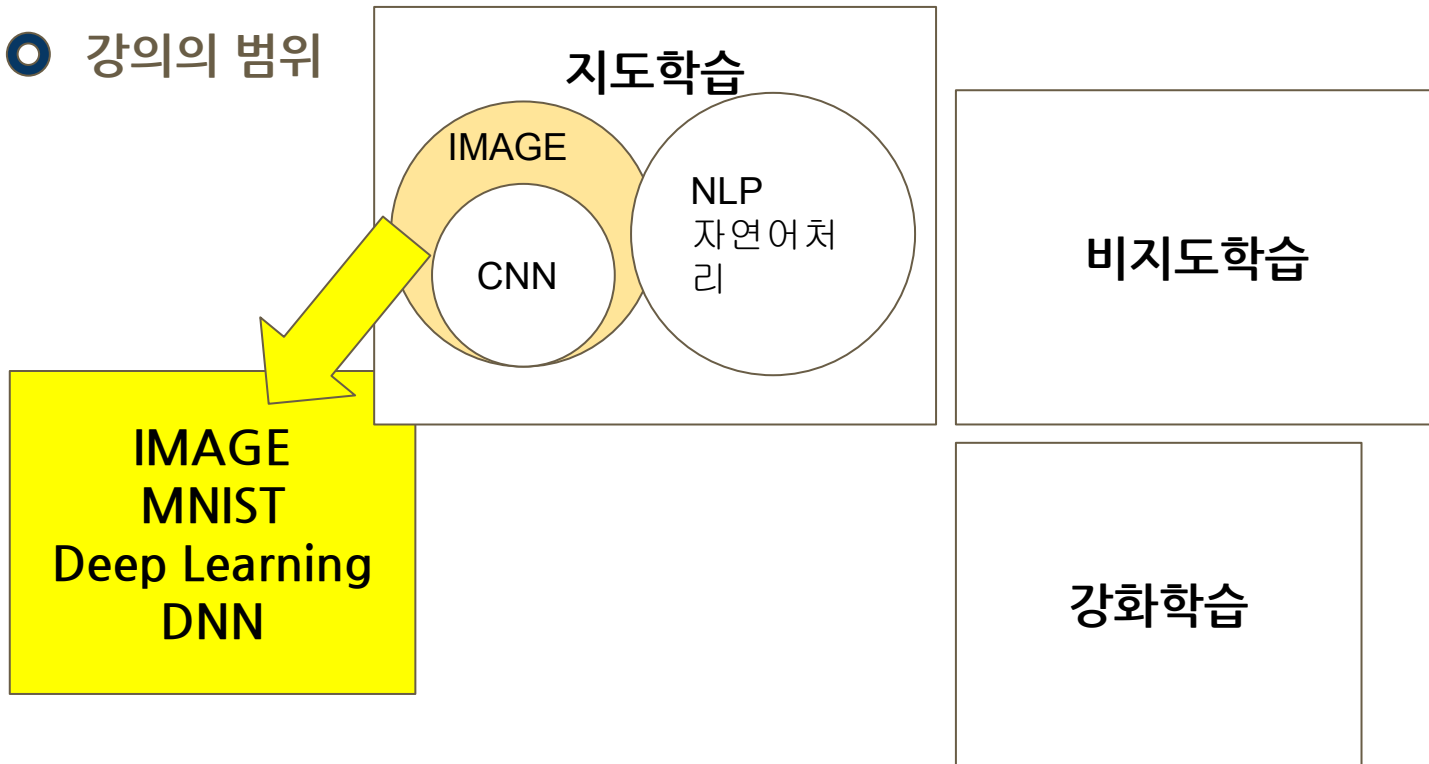
**지도학습** : 데이터마다 답(레이블)이 있는 경우

**비지도학습** : 데이터가 답을 갖지 않은 경우

**강화학습** : 시간이 지난 다음에 답이 나오는 경우, 보상

# 학습범위

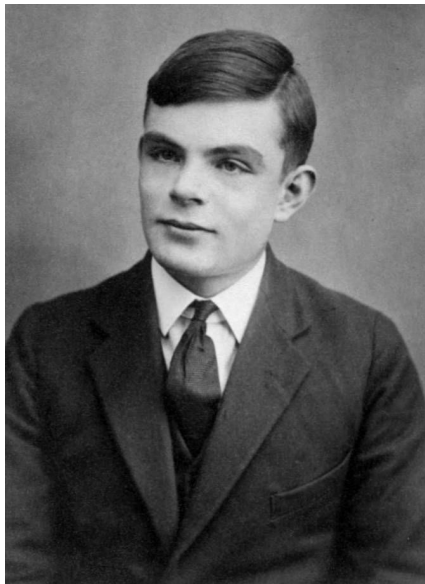
## 강의의 범위



# 인공지능의 역사

## 연대표

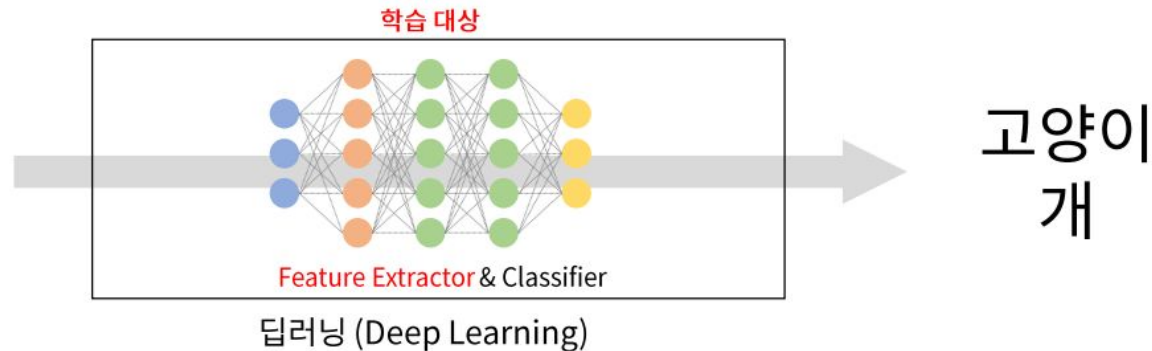
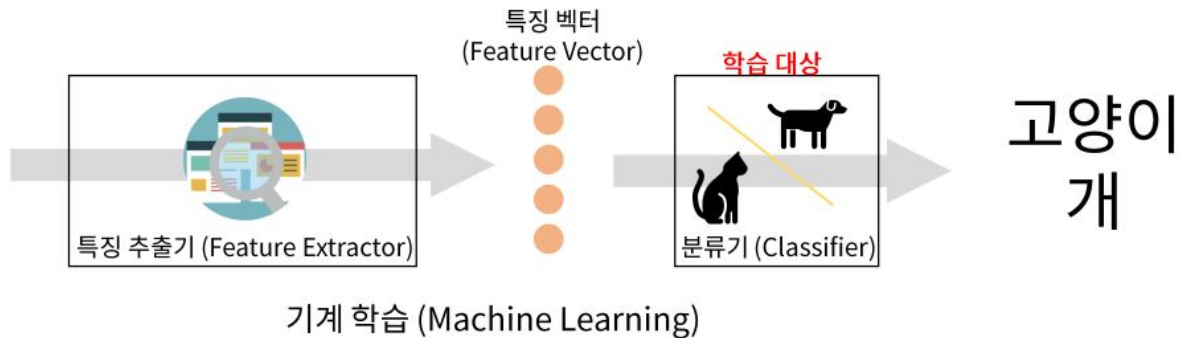
1950	알란 튜링 (1912-1954) - 생각하는 기계의 구현 가능성, 튜링테스트
1956	낙관의 시대  프랭크 로젠블라트의 퍼셉트론
1969	마빈 민스키 '퍼셉트론' 무용론  암흑기 시작  전문가 시스템
1982	역전파 이론으로 신경망의 개선과 보완, 재조명
1997	딥블루 (체스)
2005	DARPA
2011	왓슨
2016	알파고
현재	인공지능, 머신러닝의 전성기





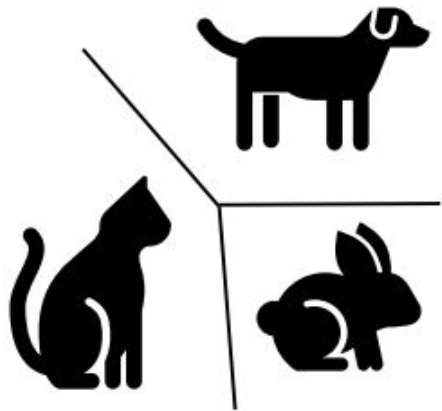
# 머신러닝의 분야

## ● 딥러닝

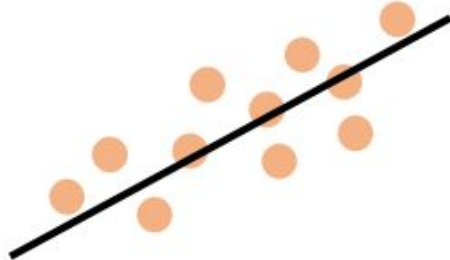


# 머신러닝의 분야

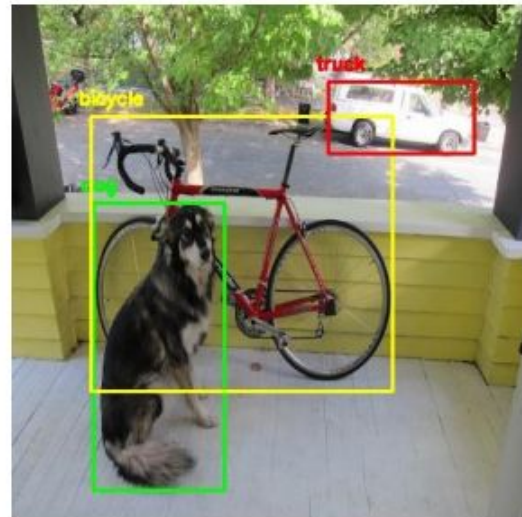
## ● 딥러닝으로 할수 있는 일



분류 (Classification)



회귀 (Regression)



물체 검출  
(Object Detection)

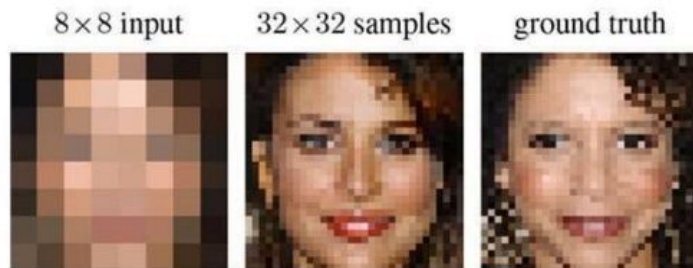
# 머신러닝의 분야

## ○ 딥러닝으로 할수 있는 일



sky tree road grass water bldg mntn fg obj.

영상 분할  
(Image Segmentation)

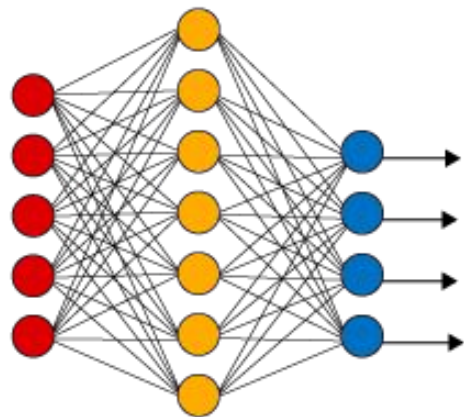


영상 초해상도  
(Image Super Resolution)

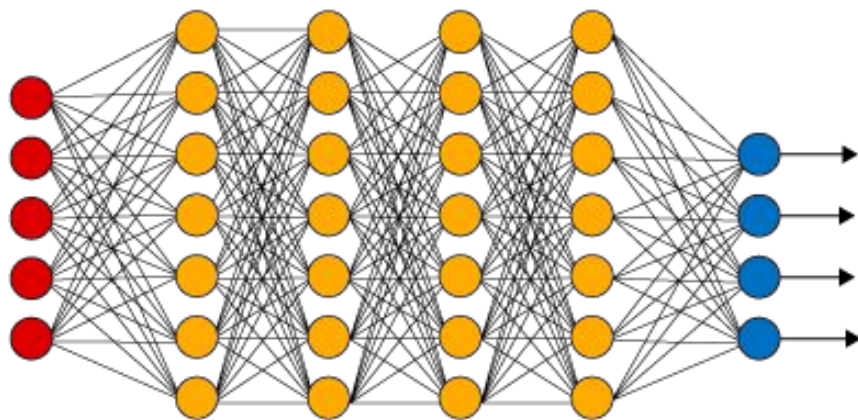
# 01. 입문자를 위한 인공지능

## ○ SNN, DNN

Simple Neural Network



Deep Learning Neural Network



● Input Layer

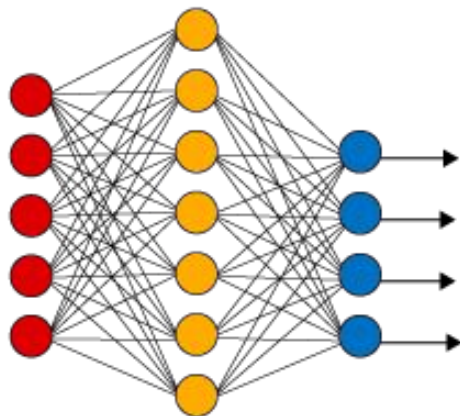
● Hidden Layer

● Output Layer

# 01. 입문자를 위한 인공지능

## ○ SNN, DNN

Simple Neural Network



계산량

입력 5

히든 7

출력 4

1단계 필요변수 :  $5 \times 7 + 7$

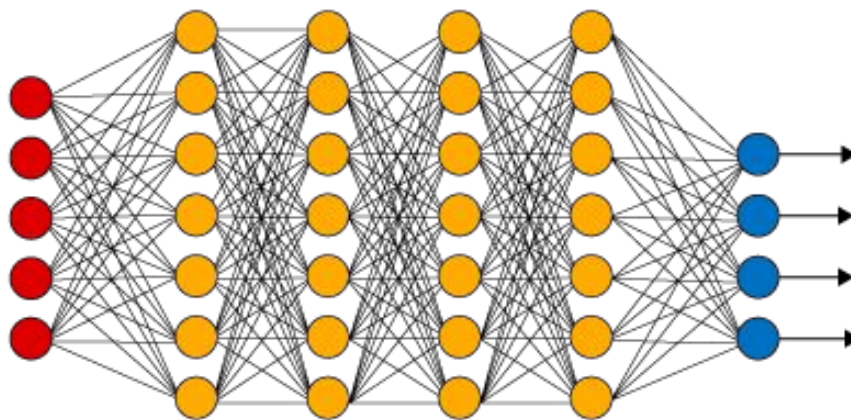
2단계 필요변수 :  $7 \times 4 + 4$

SUM = 74

# 01. 입문자를 위한 인공지능

## ○ SNN, DNN

Deep Learning Neural Network



계산량

입력 5  
히든 7, 7, 7, 7  
출력 4

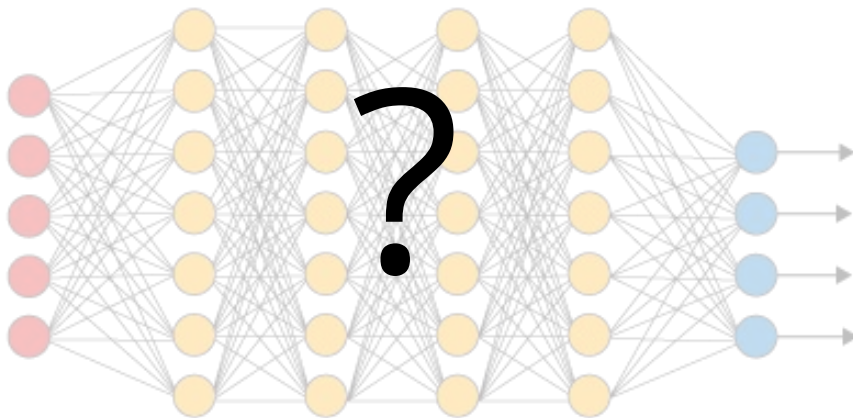
1단계 필요변수 :  $5 \times 7 + 7$   
2단계 필요변수 :  $7 \times 7 + 7$   
3단계 필요변수 :  $7 \times 7 + 7$   
4단계 필요변수 :  $7 \times 7 + 7$   
5단계 필요변수 :  $7 \times 4 + 4$

SUM = 242

# 01. 입문자를 위한 인공지능

## ● SNN, DNN

Deep Learning Neural Network



### 계산량

입력 784 (=28\*28)

히든 256, 256, 256, 256

출력 10

1단계 필요변수 :  $784 \times 256 + 256$

2단계 필요변수 :  $256 \times 256 + 256$

3단계 필요변수 :  $256 \times 256 + 256$

4단계 필요변수 :  $256 \times 256 + 256$

5단계 필요변수 :  $256 \times 10 + 10$

SUM = 400,906

# 준비



# 고교수학기초

## ● 수학적 기반

행렬, 벡터, 확률과 통계, 미분 등의 고등학교 수학과정에 나오는 지식이 필요합니다. 인공지능, 머신러닝, 딥러닝은 대학과정에 있는 선형대수를 기반으로 하기 때문에 고등학교 수준의 기본적인 수학의 이해는 선행되어야 합니다.

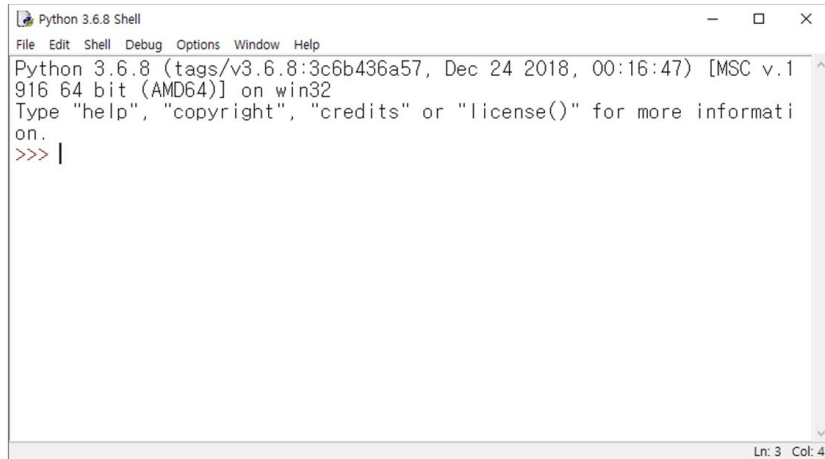
- 행렬계산
- 미분 & 수치미분

# 파이썬

## ● 인공지능에 사용되는 언어

R 과 파이썬은 인공지능에 주로 사용되는 언어입니다. 처음 프로그래밍에 접하는 이들도 쉽게 시작할 수 있는 언어입니다.

이중 파이썬은 범용언어이기 때문에 다양한 곳에 사용되고 있습니다.



```
Python 3.6.8 Shell
File Edit Shell Debug Options Window Help
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1
916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more informati
on.
>>> |
```

# 컴파일러와 인터프리터

## ● 2가지 해석 방식

컴파일러 : C, C++

- 프로그램 전체 번역
- 빠른 수행속도
- 컴파일시 상당 시간 소요
- 일부코드 수정시 전체 컴파일

인터프리터 : Python, Java

..

- 프로그램 라인단위 번역
- 상대적으로 느린 수행속도
- 컴파일 필요없음
- 컴파일 필요없음

# 파이썬 기초

## ○ 파이썬 설치

2.7 --- X

3.6 ~ 3.8, 64비트 --- O

Python **3.8.10** - May 3, 2021

3.10 is not yet stable ...

화성탐사선 큐리오시티 (2011.11.26) :

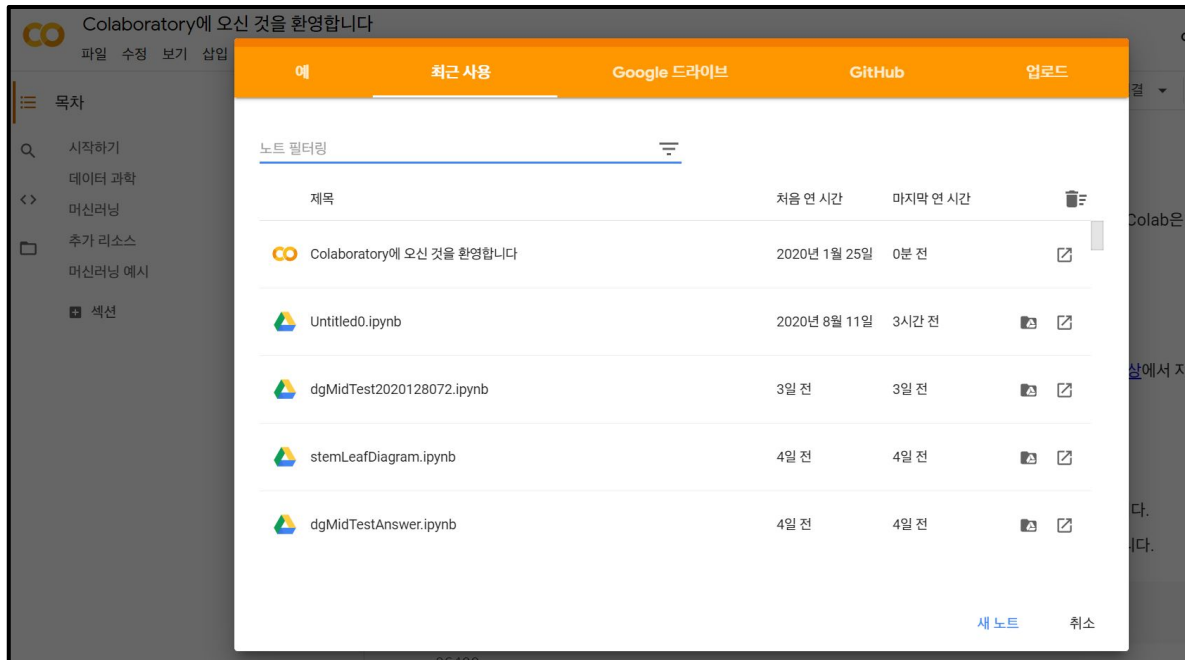
RAD750 (2001출시, 200MHz, Power PC 750, 256KB EEPROM, 2GB DRAM)

```
C
+---python
    +---python36
    +---python37
    +---python38
```

# 파이썬 기초



Google Colab  
= 온라인 파이썬  
주피터노트북



# 주피터노트북과 Colab

## ○ 설치

```
pip install scipy matplotlib
```

```
pip install jupyter
```

Google Colab 은 주피터노트북을 온라인상에서 사용할 수 있는 구글의 서비스다. 머신러닝관련 패키지들이 대부분 설치되어 있으며, 무료로 사용가능하다. 일부지만 GPU 를 이용한 학습까지 할 수 있다.

## 🔴 설치

**colab**

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

# 파이썬 기초문법



## ○ 주석이란?

프로그래밍 내부에 삽입된 메모  
프로그래밍 내부에 삽입된 사람을 위한 메모  
# 뒤의 내용은 python 인터프리터가 읽지 않는다.

### CODE

```
# 아래 줄에 대한 설명  
print('Hello World')      # 옆 줄에 대한 설명
```

# 변수

## ● 변수만들기

1. 알파벳으로 시작,
2. 중간은 알파벳, 숫자, 특수문자 중 \_
3. 대문자, 소문자 구분
4. 예약어 사용 불가

# 변수

## ○ 변수와 상수

변수 - 변할 수 있는 수(값)

특정 값을 수정하기 위해 저장할 수 있는 메모리상의 공간

상수 - 변할 수 없는 수(값)

일단 만들어진 후 수정될 수 없는 값, 읽기만 가능

# 변수

## ○ 변수사용예

### CODE

```
# 변수
x = 10
print(x)

x = 100
print(x)

y = 3.14
print(x*y)
print(type(x*y))
```

# 산술연산

## ○ 기본연산

파이썬의 산술연산으로는 기초적인 사칙연산, 정수나누기, 나머지연산, 거듭제곱등이 있다.

$+$ ,  $-$ ,  $*$ ,  $/$ ,  $//$ ,  $\%$ ,  $**$

# 자료구조

## ○ 리스트

리스트는 여러 요소를 담을 수 있고, 수정, 삭제가 가능합니다.  
데이터들은 [ ] 안에 담게 됩니다.

`a = [1, 2, 3, 4, 5, 6, 7, 8]`

리스트의 각 요소는 앞에서부터 인덱스 번호가 0부터 1씩 증가하면서 붙여집니다. 즉 `a[0]`은 첫 번째 요소인 1입니다. 세 번째 요소인 3은 `a[2]`입니다. 0부터 시작하기 때문에 n번째 요소의 인덱스번호는 `n-1`이 됩니다.

# 자료구조

## ● 인덱스, 슬라이싱

**a = [1, 2, 3, 4, 5, 6, 7, 8]**

인덱스 번호를 이용해 특정 위치값을 읽거나 수정할 수 있습니다.

**a[0]      # 1**

인덱스 번호의 시작과 끝을 지정하면 사이의 값을 읽어옵니다.

**a[0:3]      # [1,2,3]    - 0번부터 시작(이상) 3번 전까지(미만)**

# 자료구조

## ○ 튜플

**b = (1, 2, 3, 4, 5, 6, 7, 8)**

튜플은 인덱스와 달리 수정할 수 없습니다. 한번 만들어진 튜플은 상수처럼 읽기만 가능합니다.

인덱스, 슬라이싱 모두 리스트와 같은 방식으로 사용됩니다.



# 자료구조

## ○ 딕셔너리

딕셔너리는 하나의 요소가 **key**와 **value**로 구성되어 저장, 요소는 그 요소를 가리키는 **key**와 설명에 해당하는 **value**로 구성

```
a = {'name': 'Joy', 'phone': '010-0000-0123'}
```

딕셔너리는 { }로 묶이고, **key**와 **value** 사이에 :으로 구분됩니다.  
키(**key**)를 사용해서 값(**value**)을 읽어옵니다.

# 조건문

## ○ if 조건문

조건이 '참'인 경우에 실행되는 구조

```
# 1.3.7 if문
hungry = True
hungry = not hungry
if hungry:
    print("i'm hungry")
else :
    print("i'm not hungry")
    print("i'm sleepy")
```

# 반복문

## ○ for

조건이 만족하는 동안 혹은 정해진 횟수가 될 때까지 실행되는 구조

# 1.3.8 for문

```
for myNum in [1,20,3]:  
    print(myNum)
```

```
for myIter in range(10):  
    print(myIter)
```

# 함수

## 🕒 def 함수 작성 예

# 함수1

```
def hello():  
    print("hello world")
```

hello()

# 함수2

```
def hello(name):  
    print("Hello " + name + "!")
```

hello("cat")

# 클래스와 객체

## ○ 함수와 클래스 차이

함수 = ‘기능’ 모음

클래스 = ‘정보’ + ‘기능’ 모음

# 클래스와 객체

## ○ 구조

```
class 클래스이름:
    def __init__(self, 인수, ...):
        ...
    def 메소드이름1(self, 인수, ...):
        ...
    def 메소드이름2(self, 인수, ...):
        ...
```

# numpy

## ● numpy란?

- Numpy는 C언어로 구현된 파이썬 라이브러리
- Numerical Python 줄임말
- 고성능의 수치계산을 위해 제작
- 벡터 및 행렬 연산에 효율적

# numpy

## ⦿ 배열 만들기

### CODE

```
# 3.9 배열 (array)

import numpy as np

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



# numpy

## 산술연산

### 리스트와 numpy 배열의 연산 비교

#### CODE

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

#### CODE

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

# numpy

## 산술연산

### CODE

```
# 3.9.2. 산술연산
import numpy as np
a = np.array([1,2,3,4])
b = np.array([.4, .4, .3, .2])
print(a+b)
print(a-b)
print(a*b)
print(a/b)
```

배열의 사칙연산  
VS  
배열요소의 사칙연산

.4 는 0.4 와 동일

# numpy

## 2차원 배열과 행렬

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

# numpy

## 2차원 배열과 행렬

### CODE

```
# 3.9.3. 2차원 배열과 행렬
import numpy as np
A = np.array([[1,2],[3,4]])
B = np.array([[5,6],[7,8]])
print(A+B)
print(A*B)
```

배열의 사칙연산 -- X  
VS  
배열요소의 사칙연산 -- O

# numpy

● 행렬의 곱: `np.dot(A,B)`  $\neq$  행렬요소의 곱 ( $A*B$ )

$$A \cdot B = C$$

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}$$

$$c_{11} = a_1b_1 + a_2b_3$$

$$c_{12} = a_1b_2 + a_2b_4$$

$$c_{21} = a_3b_1 + a_4b_3$$

$$c_{22} = a_3b_2 + a_4b_4$$

# numpy

## 3.9.4 행렬의 곱

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \cdot \begin{pmatrix} b_1 & b_2 \\ b_3 & b_4 \end{pmatrix} = \begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix}$$

$C(1,1) =$  앞 행렬의 1행과 뒤행렬의 1열을 곱한 다음 더한 것  
 $C(1,2) =$  앞 행렬의 1행과 뒤행렬의 2열을 곱한 다음 더한 것  
 $C(2,1) =$  앞 행렬의 2행과 뒤행렬의 1열을 곱한 다음 더한 것  
 $C(2,2) =$  앞 행렬의 2행과 뒤행렬의 2열을 곱한 다음 더한 것

앞 행렬의 열의 크기, 뒤 행렬의 행의 크기가 같아야 행렬곱이 가능

$(2,2) \times (2,2)$  -- OK  $[2,2]$

$(2,3) \times (3,2)$  -- OK  $[2,2]$

$(1,10) \times (10,1)$  -- OK  $[1,1]$

$$C_{11} = c_1 = a_1b_1 + a_2b_3$$

$$C_{12} = c_2 = a_1b_2 + a_2b_4$$

$$C_{21} = c_3 = a_3b_1 + a_4b_3$$

$$C_{22} = c_4 = a_3b_2 + a_4b_4$$

# numpy

## 3.9.4 행렬의 곱

### CODE

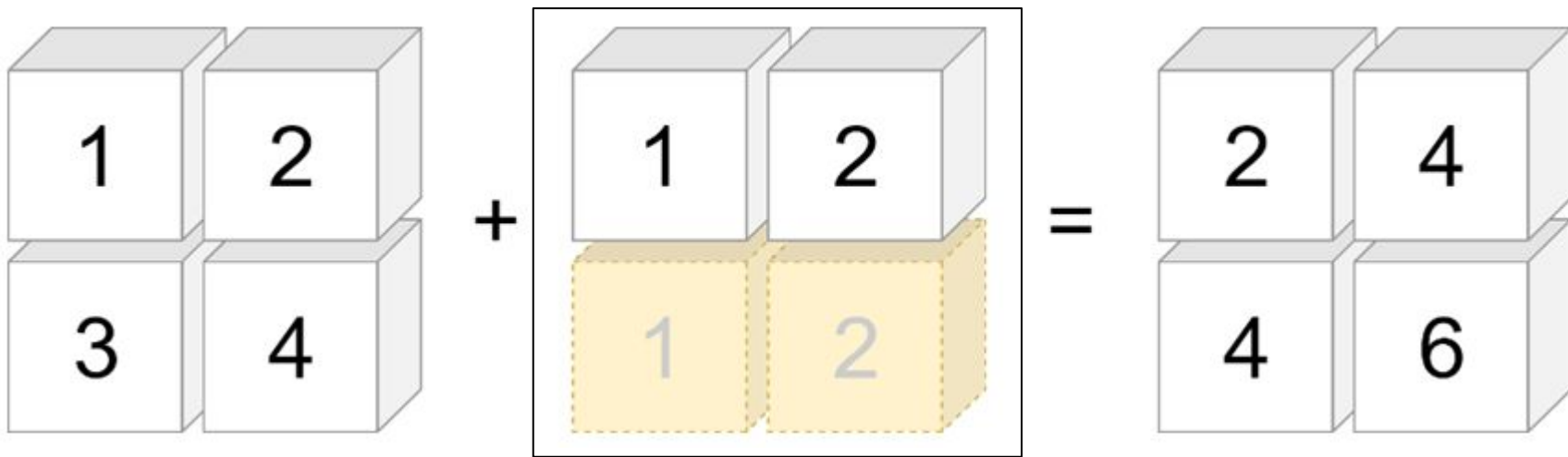
```
# 3.9.4. 행렬의 곱

import numpy as np

A = np.array([[1,2],[3,4]])
B = np.array([[1,2],[3,4]])
print(np.dot(A,B))      # A*B
```

# numpy

## ● 브로드캐스트

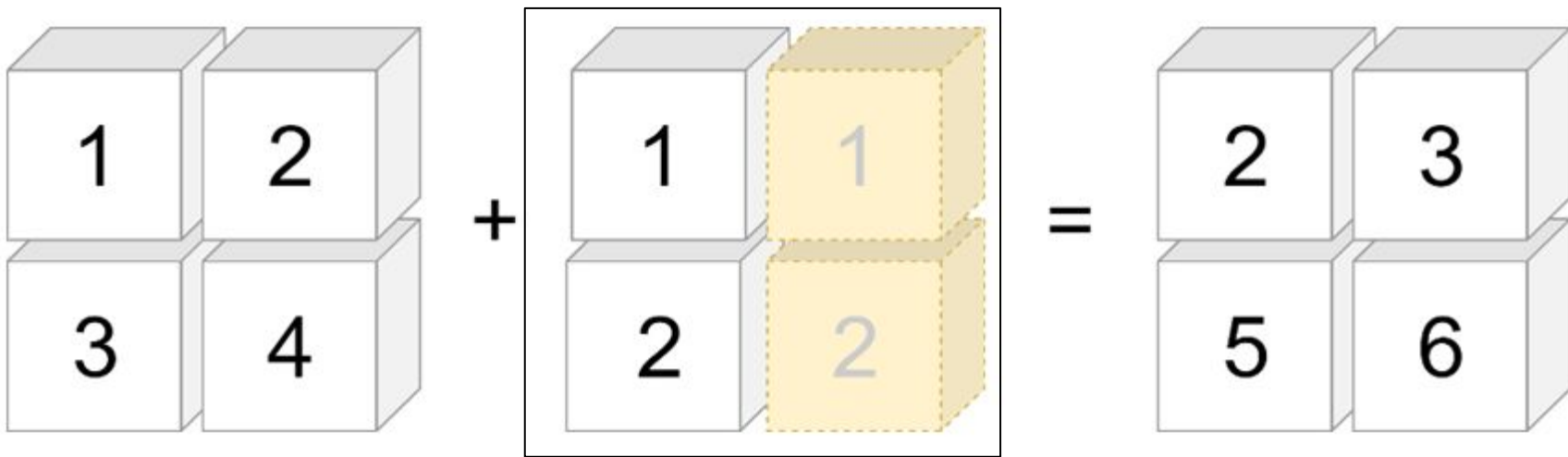


행 확장



# numpy

## ● 브로드캐스트

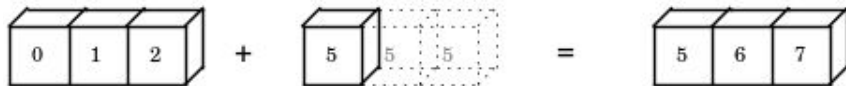


영 확장

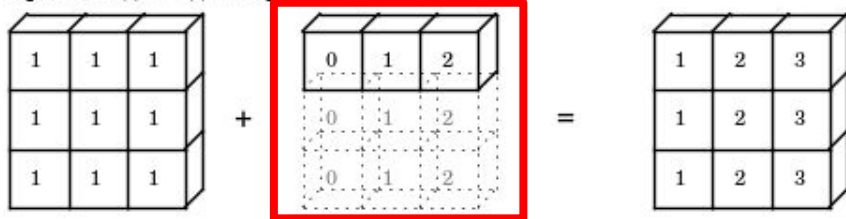
# numpy

## ● 브로드캐스트

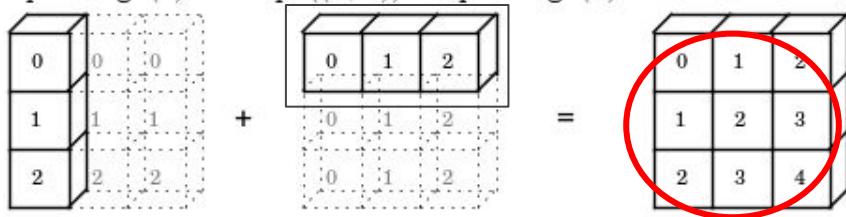
`np.arange(3) + 5`



`np.ones((3, 3)) + np.arange(3)`



`np.arange(3).reshape((3, 1)) + np.arange(3)`



### CODE

```
A = np.arange(3) + 5
B = np.ones((3, 3)) + np.arange(3)
C = np.arange(6).reshape((2, 3)) + np.arange(3)
```

```
[0,0,0] + [0,1,2] = [0,1,2]
[1,1,1]   [0,1,2]   [1,2,3]
[2,2,2]   [0,1,2]   [2,3,4]
```

```
np.arange(3) = [0,1,2]
```

# numpy

## ● 브로드 캐스트

### CODE

```
# 3.9.4. 행렬의 곱
```

```
import numpy as np
```

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

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

```
print(np.dot(A,B))
```

# numpy

## ● 리스트 중첩

### CODE

```
# 리스트 중첩으로 3 보다 큰 요소를 리스트로 추출
```

```
originArray = [1,2,3,4,5]
```

```
[i for i in originArray if i > 3]
```

[결과]

[4, 5]

# numpy

## 🕒 Numpy 에서 불리언 색인 추출

CODE

```
import numpy as np

originArray2 = np.array([1,2,3,4,5])
print(originArray2 > 3)
print(originArray2[originArray2 > 3])
```

결과

```
[False False False  True  True]
[4 5]
```

# matplotlib 를 이용한 그래프

# Matplotlib

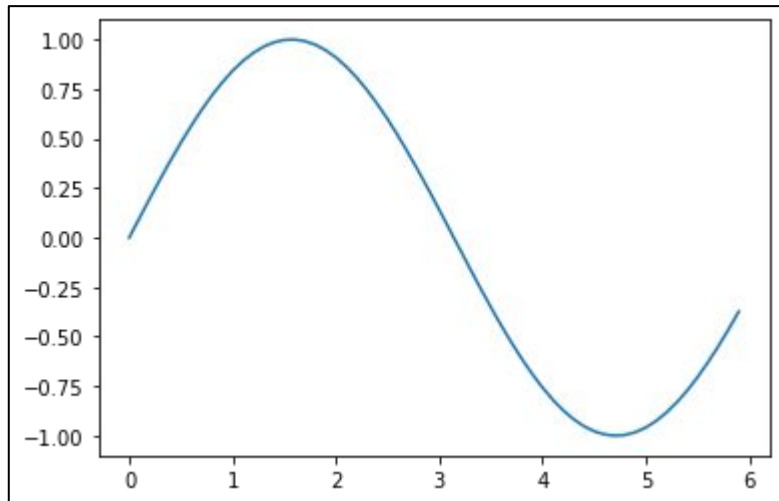
## sin 그래프

### CODE

```
# 3.10.1. sin 그래프 그리기

import numpy as np
import matplotlib.pyplot as plt

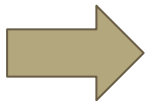
x = np.arange(0, 6, 0.1)
y = np.sin(x)
plt.plot(x,y)
plt.show()
```



# Matplotlib

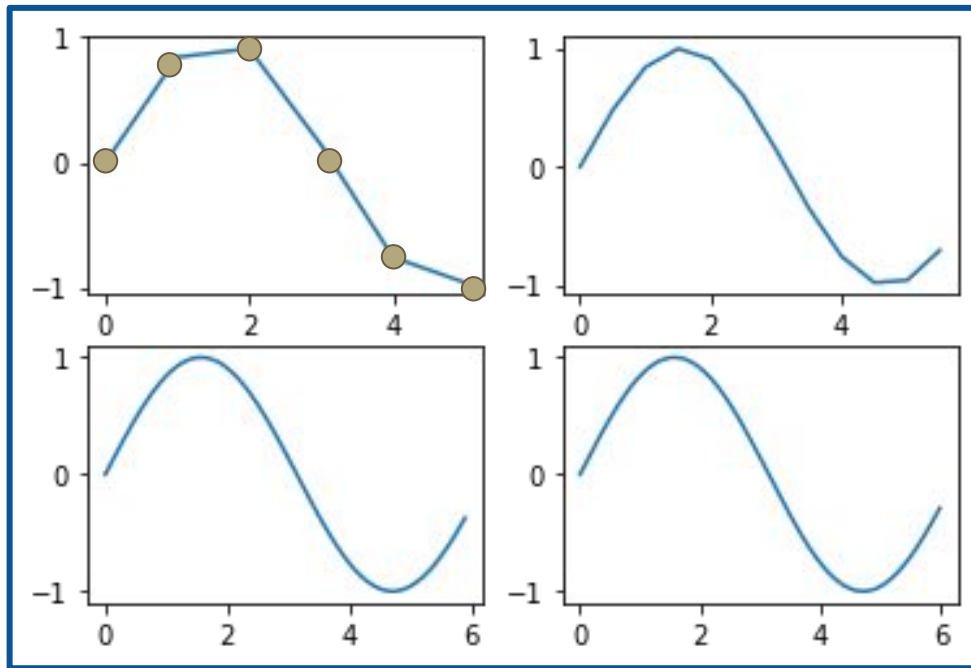
## ● 그래프를 그리기 위한 데이터

6개, 12개, 60개, 600개



`rate = 60`

`np.arange(0,6,6/rate)`





# Matplotlib

## 🕒 이미지파일 출력

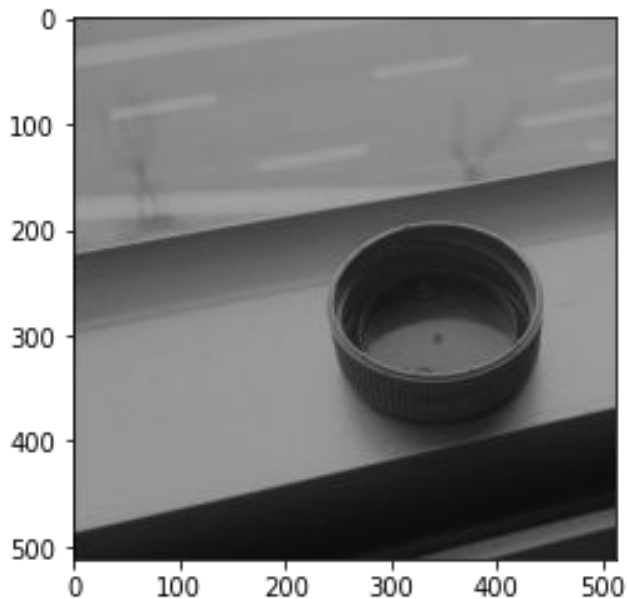
# 3.10.3. CoLab 을 이용한 이미지 파일 화면 출력

```
import matplotlib.pyplot as plt
import matplotlib.image as pmg

from google.colab import files
upload = files.upload()

filename_key = upload.keys()
filename_key_list = list(filename_key)
filename_key_list[0]

img = pmg.imread(filename_key_list[0])
plt.imshow(img)
plt.show()
```



# MNIST

# MNIST 특징

## ● 연구원과 고등학생의 혼합 손글씨

훈련용으로 연구원들이 만든 데이터 SD-3를 사용했고, 검증용으로 고등학생들이 만든 데이터인 SD-1을 사용했습니다. SD-3는 SD-1보다 깨끗하고 인식하기 쉽습니다. 이런 문제점 때문에 SD-1과 SD-3를 혼합한 MNIST 데이터셋을 만들었습니다. 그래서 훈련용 데이터 60,000개는 SD-1과 SD-3가 각각 절반씩 차지하고, 검증용 데이터 10,000개도 SD-1과 SD-3가 절반씩 차지합니다.

# MNIST DataSet 구성

## ○ 구성

### train

60,000 개

28×28 (=784) 픽셀

x\_train

훈련 데이터

y\_train

정답 레이블

### test

10,000 개

28x28 (=784) 픽셀

x\_test

검증 데이터

y\_test

정답 레이블

# MNIST DataSet 구성

## 읽어오기

### MNIST 불러오기

CODE

```
# 4.1 MNIST DataSet 구성

from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

## ○ numpy란?

[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	3	18	18	18	126	136	175	26	166	255	247	127	0	0	0	0	0]
[	0	0	0	0	0	0	0	0	30	36	94	154	170	253	253	253	253	253	225	172	253	242	195	64	0	0	0	0	0]	
[	0	0	0	0	0	0	0	49	238	253	253	253	253	253	253	253	253	251	93	82	82	56	39	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	18	219	253	253	253	253	253	198	182	247	241	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	80	156	107	253	253	205	11	0	43	154	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	14	1	154	253	90	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	139	253	190	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	11	190	253	70	0	0	0	0	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	35	241	225	160	108	1	0	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	81	240	253	253	119	25	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	186	253	253	150	27	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	16	93	252	253	187	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	249	253	249	64	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	130	183	253	253	207	2	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	0	39	148	229	253	253	253	250	182	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0	0	0	0	0	0	0	0	24	114	221	253	253	253	253	201	78	0	0	0	0	0	0	0	0	0]	
[	0	0	0	0																										

```
print(x_train[0])
```

# MNIST 에서 사진 가져오기

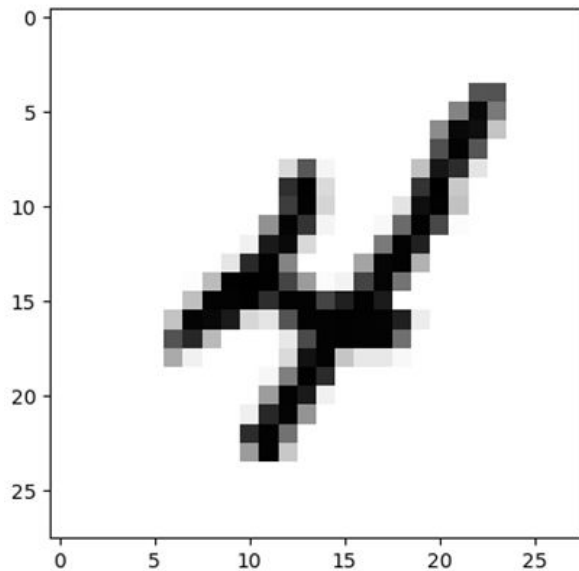
## 🕒 이미지로 출력

### CODE

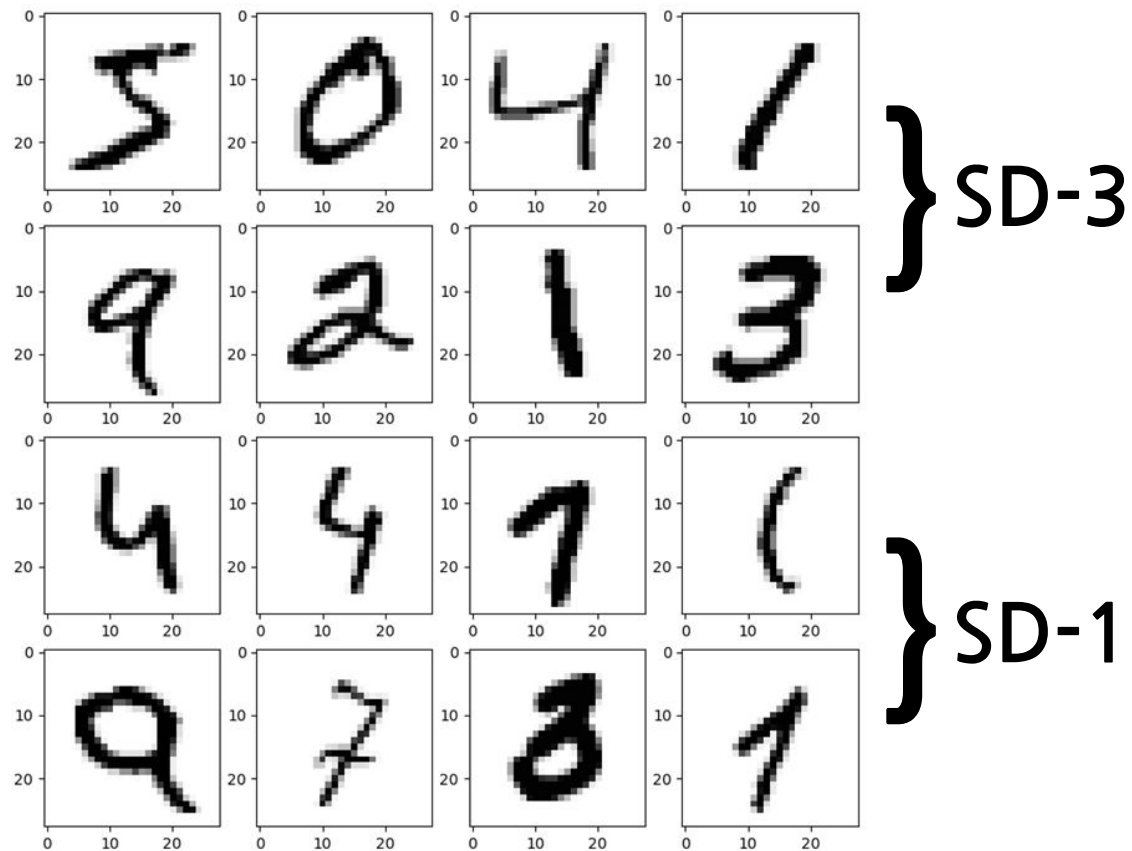
```
import matplotlib.pyplot as plt
from keras.datasets import mnist

..

image = x_train[9]
plt.imshow(image, cmap='Greys')
plt.show()
```



# MNIST 특징

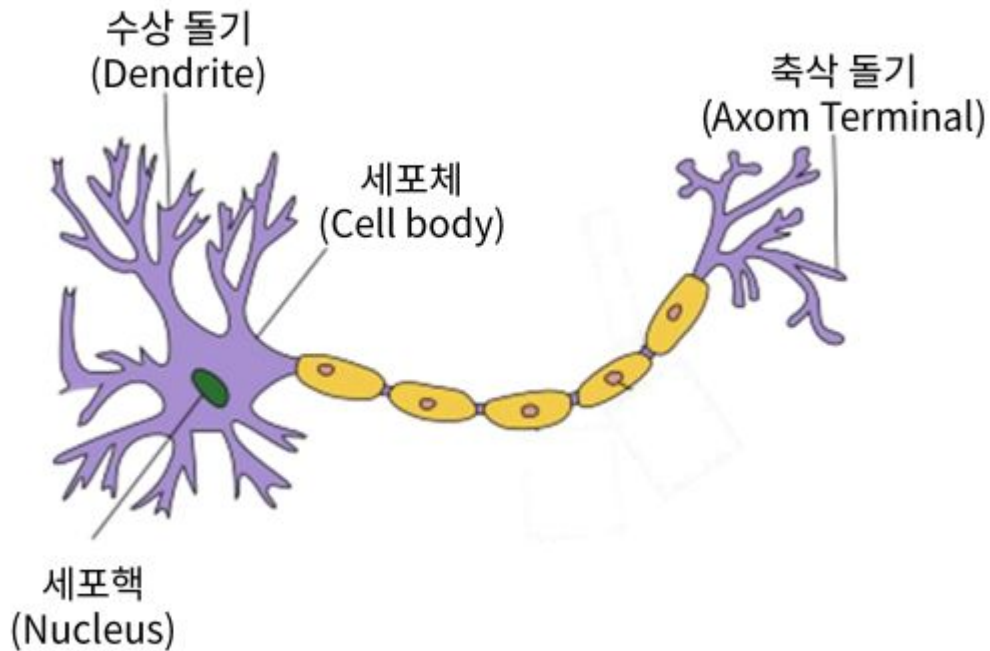




# 퍼셉트론 VS XOR

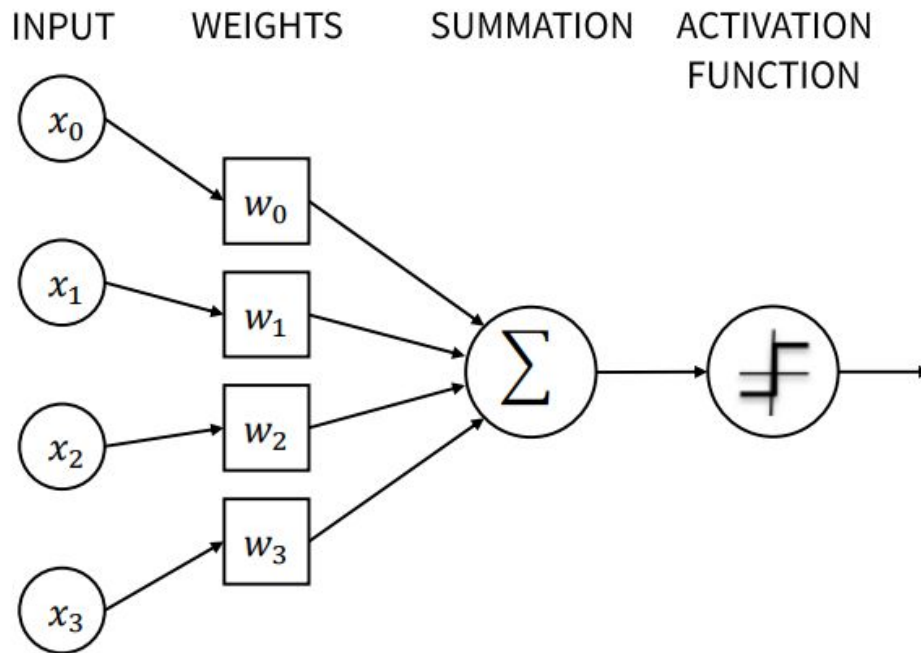
# 퍼셉트론 & 뉴런

## ○ 뉴런



# 퍼셉트론 & 뉴런

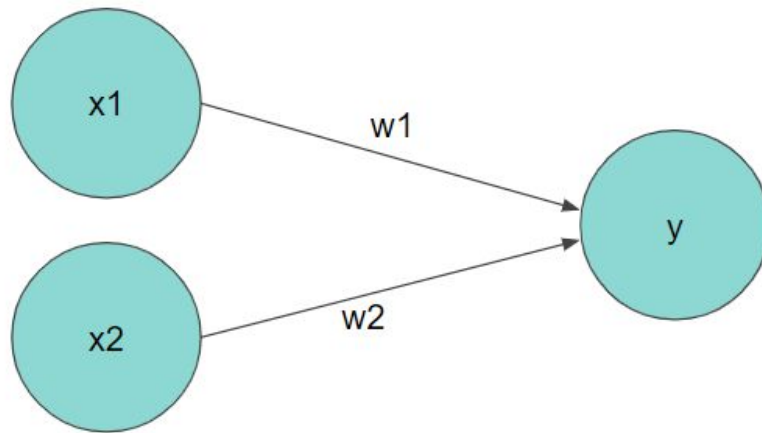
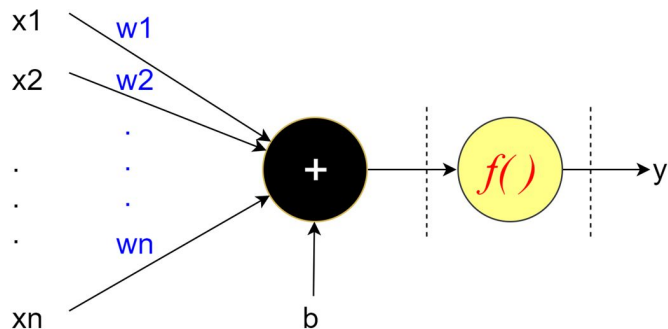
## ○ 퍼셉트론



Rosenblatt의 퍼셉트론 구조  
(Rosenblatt, 1958)

# 퍼셉트론 & 뉴런

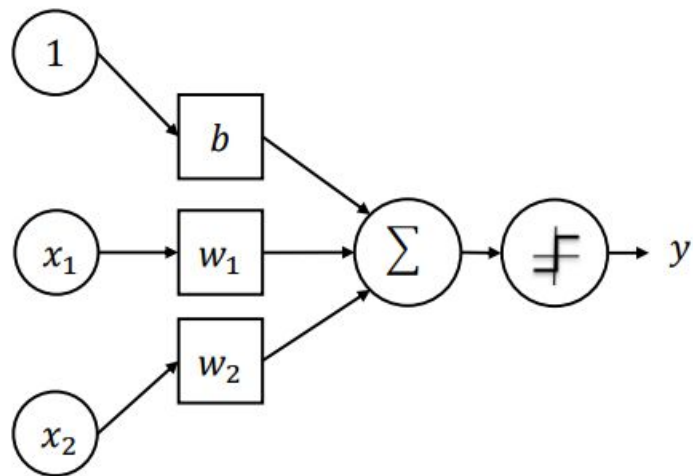
## ○ 퍼셉트론



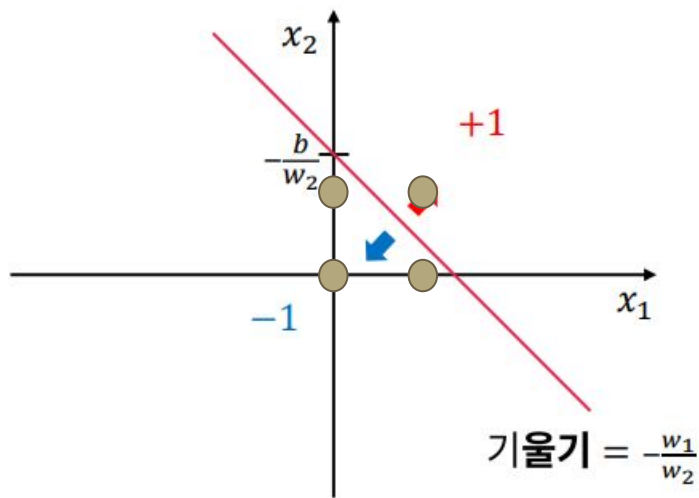
$$y = \begin{cases} 0 & (w_1 \times x_1 + w_2 \times x_2 \leq \theta) \\ 1 & (w_1 \times x_1 + w_2 \times x_2 > \theta) \end{cases}$$

# 퍼셉트론 & 뉴런

## ○ 퍼셉트론



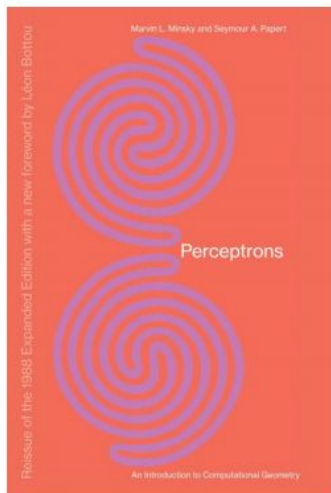
$$y = \begin{cases} +1, & b + w_1x_1 + w_2x_2 \geq 0 \\ -1, & b + w_1x_1 + w_2x_2 < 0 \end{cases}$$



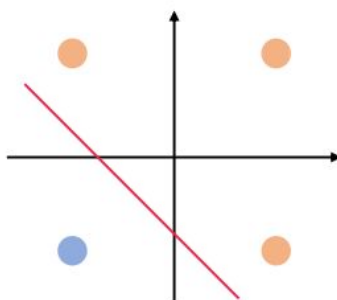
$$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2}$$

# 퍼셉트론 & 뉴런

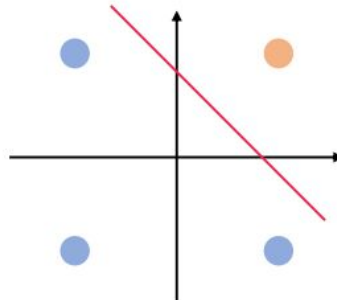
## ○ XOR 문제



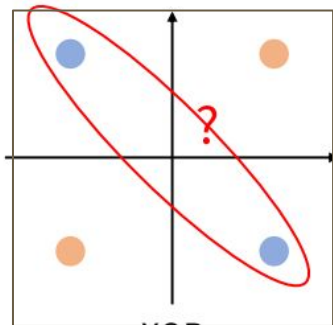
Perceptrons  
(Minsky and Papert, 1969)



OR



AND



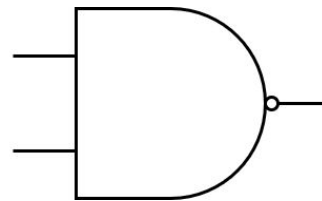
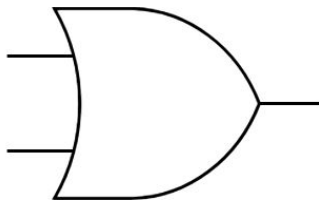
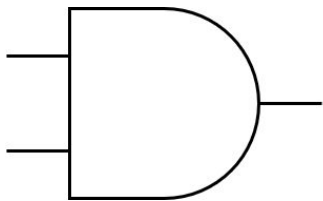
XOR



# 논리연산자

# 논리연산자 만들기

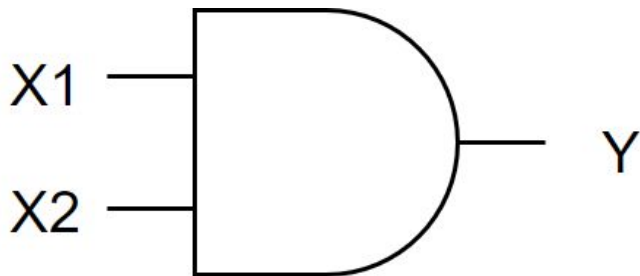
● AND --- OR --- NAND





# 논리연산자 만들기

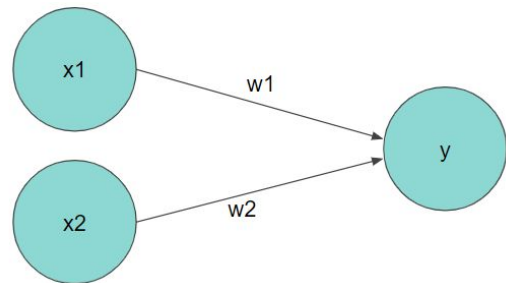
## ● AND



X1(입력1)	X2(입력2)	Y (출력)
0	0	0
1	0	0
0	1	0
1	1	1

# 논리연산자 만들기

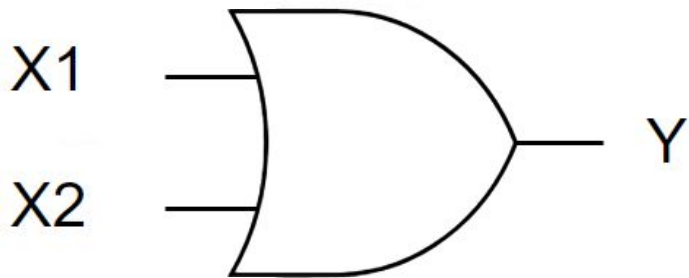
## AND



입력		결과	결과 - <u>임계값</u> (#2)	출력
x1	x2	$w_1x_1 + w_2x_2$	$w_1x_1 + w_2x_2 - q$	y
0	0	0	$-q$ ----- (#3)	0
0	1	$w_2$	$w_2 - q$ ----- (#4)	0
1	0	$w_1$ ---- (#1)	$w_1 - q$ ----- (#5)	0
1	1	$w_1 + w_2$	$w_1 + w_2 - q$ ---- (#6)	1

# 논리연산자 만들기

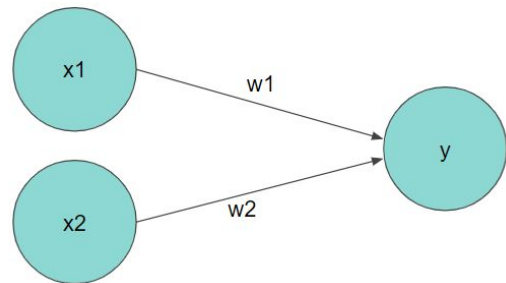
## OR



X1(입력1)	X2(입력2)	Y (출력)
0	0	0
1	0	1
0	1	1
1	1	1

# 논리연산자 만들기

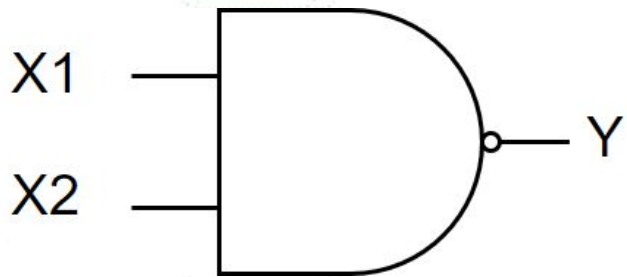
OR



입력		결과	결과 - <u>임계값</u>	출력
x1	x2	$w1x1+w2x2$	$w1x1+w2x2-q$	y
0	0	0	$-q \leq 0$	0
0	1	$w2$	$w2-q > 0$	1
1	0	$w1$	$w1-q > 0$	1
1	1	$w1+w2$	$w1+w2-q > 0$	1

# 논리연산자 만들기

## ● NAND

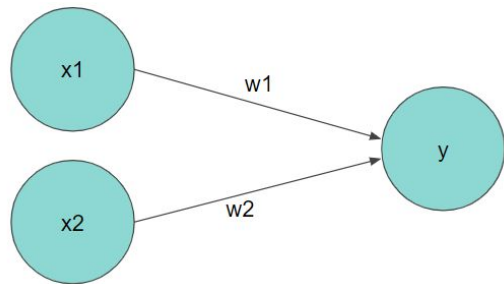


X1(입력1)	X2(입력2)	Y (출력)
0	0	1
1	0	1
0	1	1
1	1	0

# 논리연산자 만들기

## ● NAND

입력		결과	결과 - 임계값		출력
x1	x2	$w1x1+w2x2$	$w1x1+w2x2-q$		y
0	0	0	$-q$	$> 0$	(1) $q < 0$
0	1	$w2$	$w2-q$	$> 0$	(2) $q < w2$
1	0	$w1$	$w1-q$	$> 0$	(3) $q < w1$
1	1	$w1+w2$	$w1+w2-q$	$\leq 0$	(4) $q \geq w1 + w2$



# 논리연산자 만들기

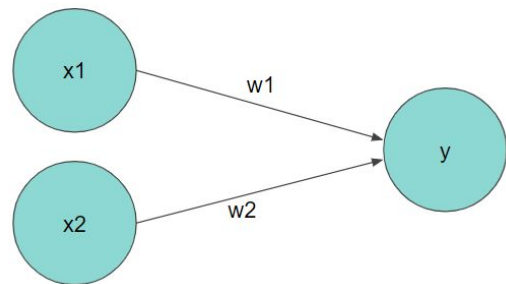
## ● XOR



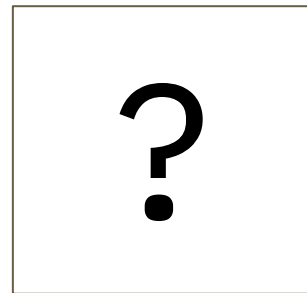
x1(입력 1)	x2(입력 2)	x1 XOR x2
0	0	0
0	1	1
1	0	1
1	1	0

# 논리연산자 만들기

## XOR



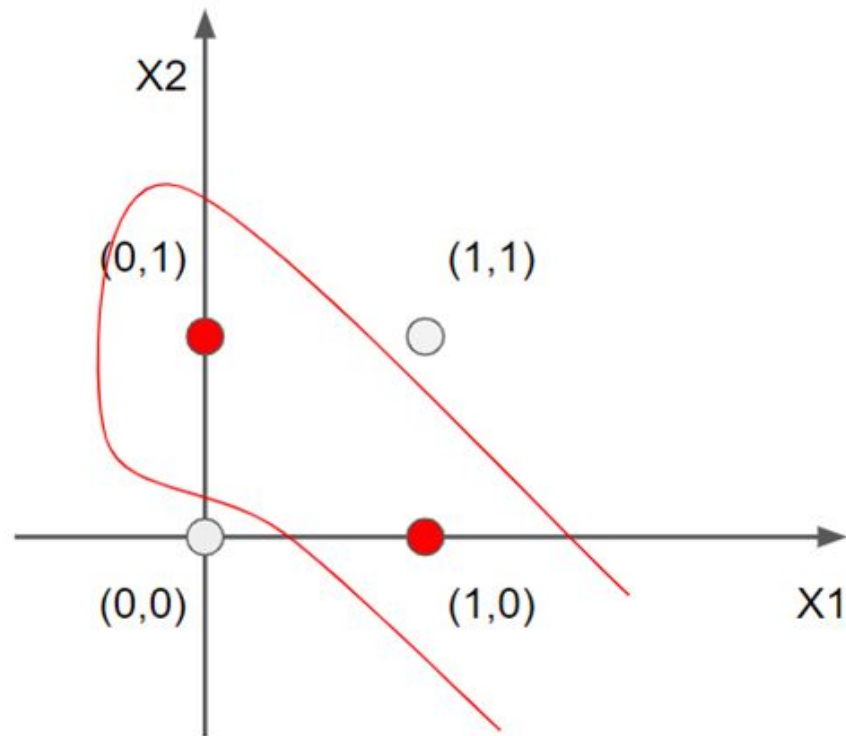
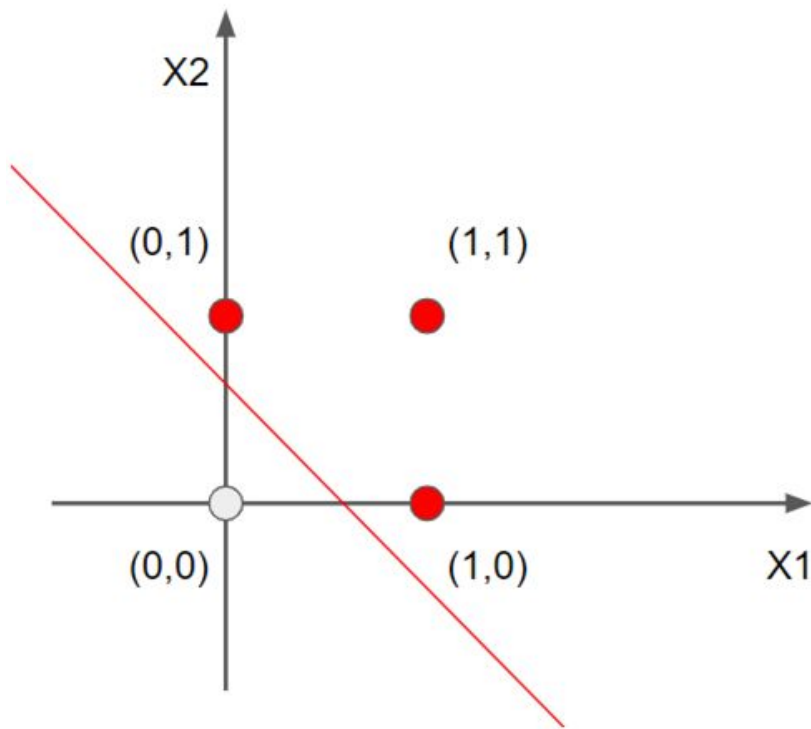
입력		결과	결과 - 임계값		출력
X1	X2	$w_1x_1 + w_2x_2$	$w_1x_1 + w_2x_2 - q$		Y
0	0	0	$-q < 0$	$q > 0$	0
0	1	$w_2$	$w_2 - q > 0$	$q < w_2$	1
1	0	$w_1$	$w_1 - q > 0$	$q < w_1$	1
1	1	$w_1 + w_2$	$w_1 + w_2 - q < 0$	$q > w_1 + w_2$	0





# 논리연산자 만들기

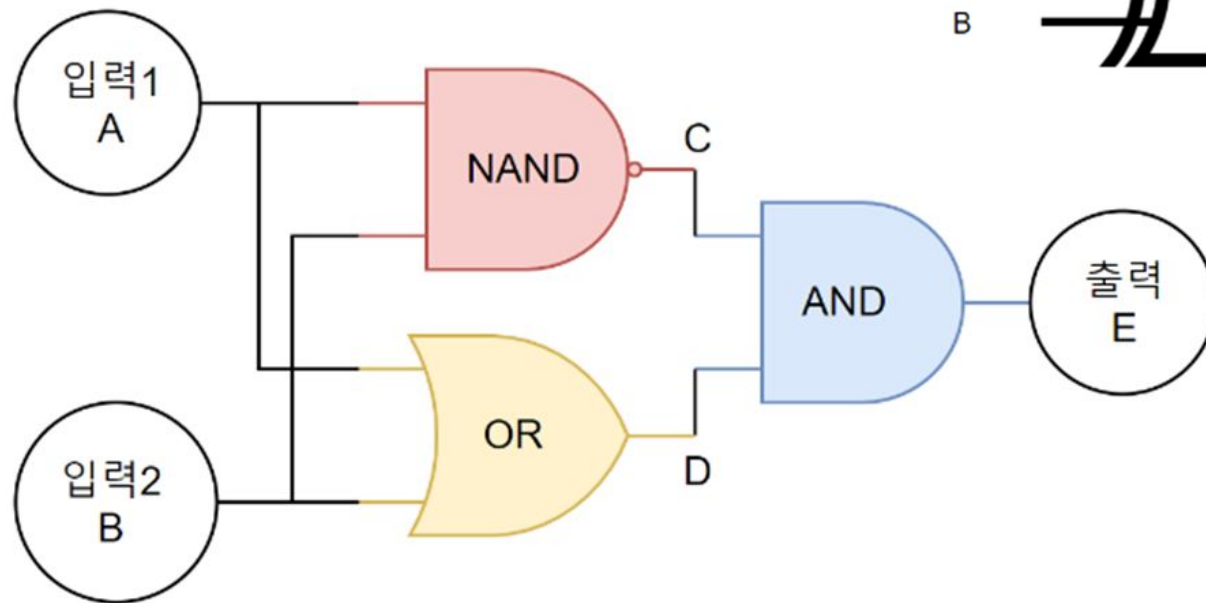
## OR VS XOR



# 다층퍼셉트론

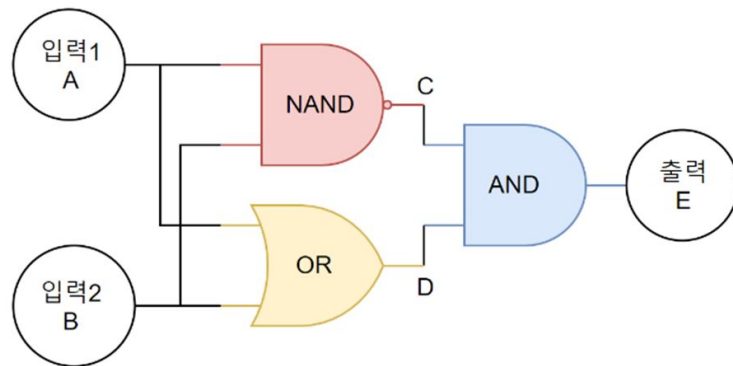
# 다층퍼셉트론 (MLP)

## XOR



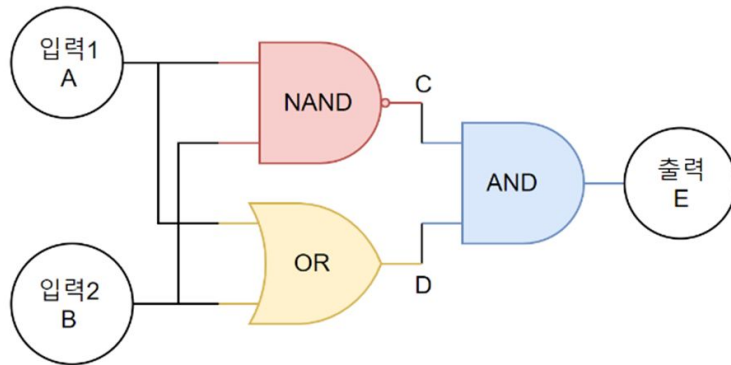
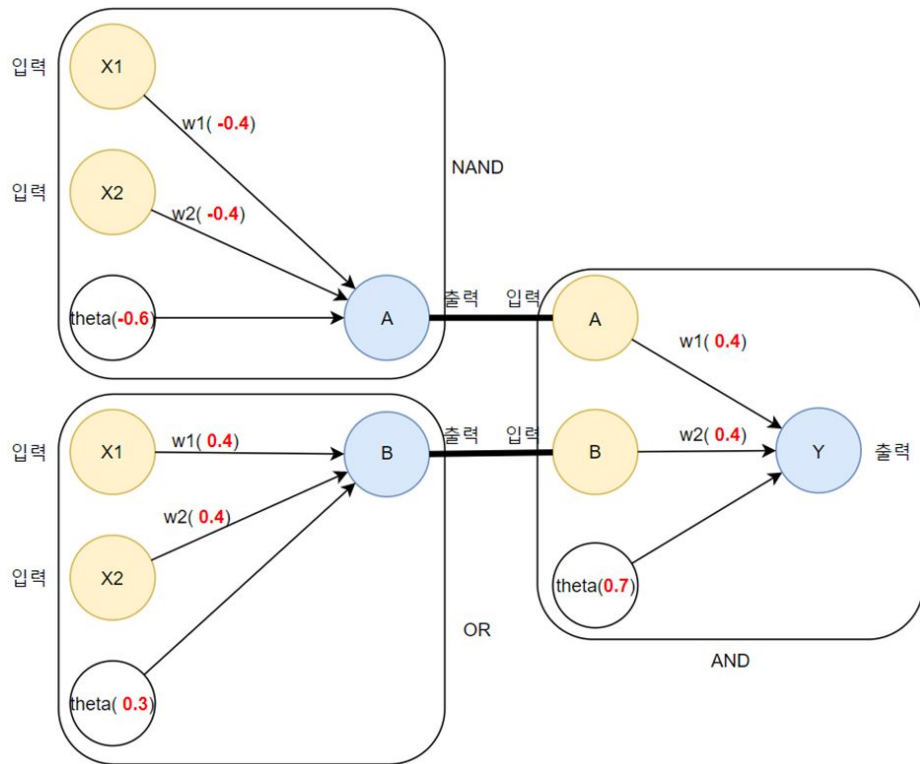
# 다층퍼셉트론

## XOR



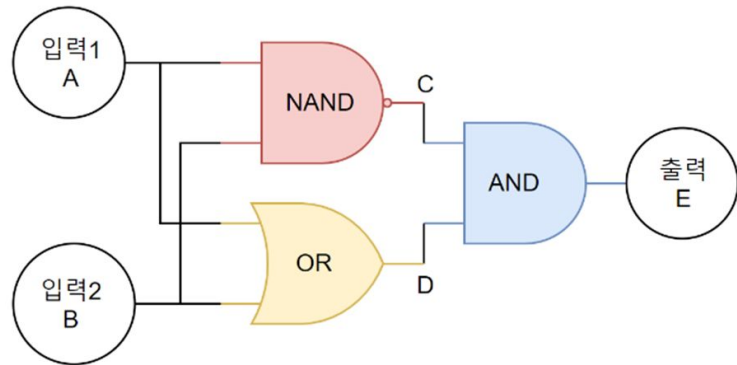
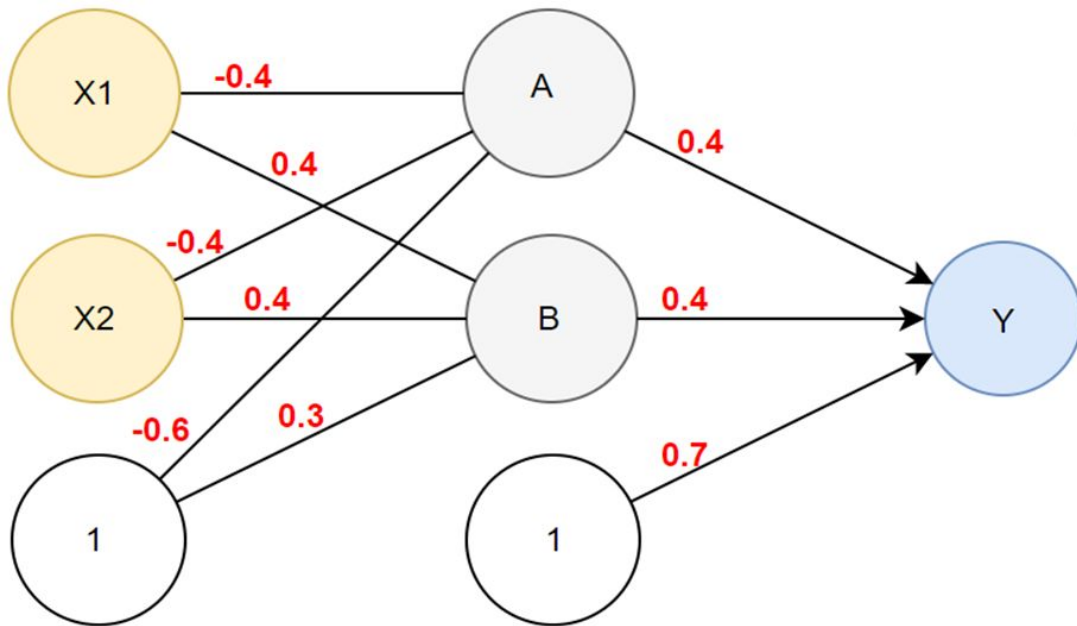
A 입력	B 입력	C NAND(A,B)	D OR(A,B)	E AND(C,D)	XOR
0	0	1	0	0	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	0	1	0	0

## XOR



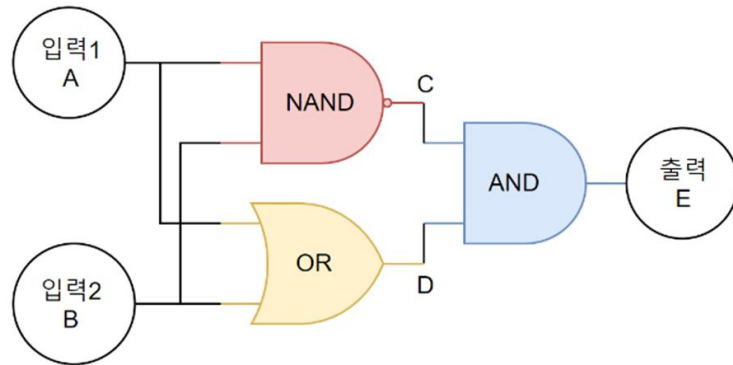
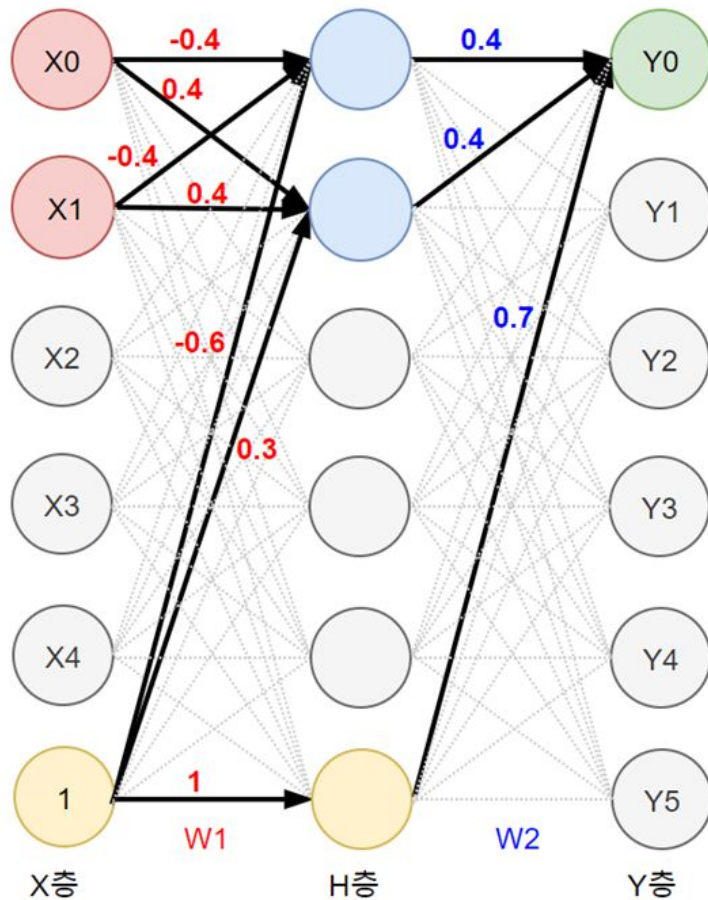
# 다층퍼셉트론

## ● XOR



# 다층퍼셉트론

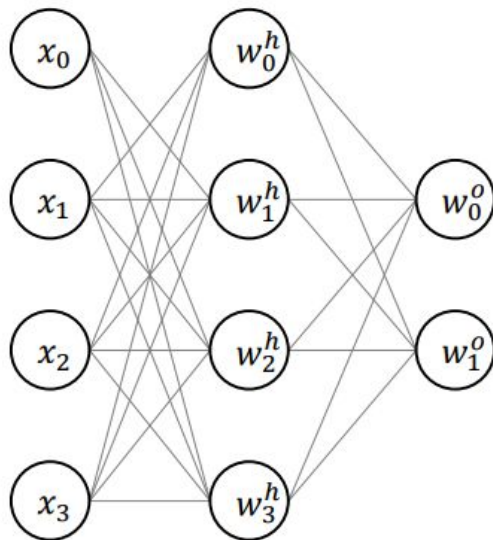
● XOR



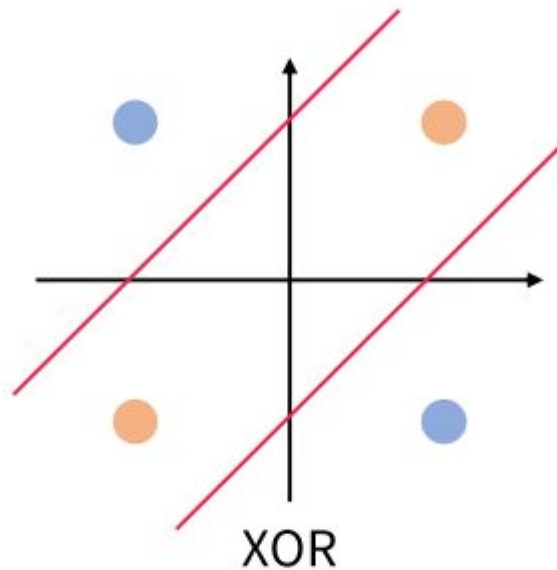
# 다층퍼셉트론

## 구조

첫번째 계층 (입력 계층)    두번째 계층 (은닉 계층)    세번째 계층 (출력 계층)



다층 퍼셉트론 (1986)  
(Multi-Layered Perceptrons; MLP)



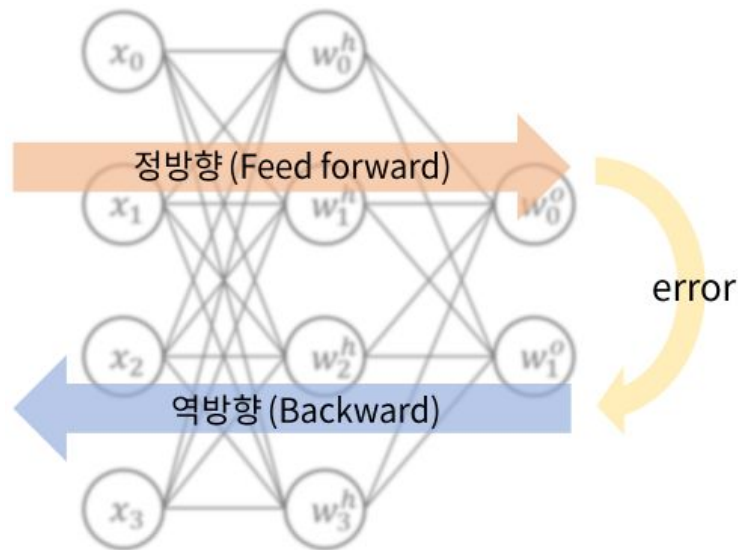
MLP로 XOR 문제를 해결한 예



# 다층퍼셉트론

## ● 역전파 알고리즘

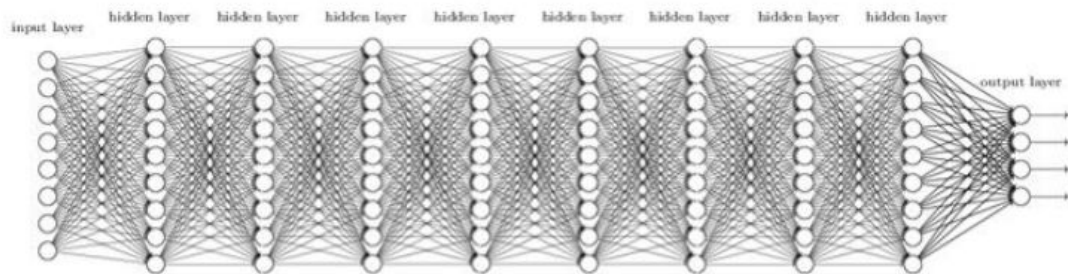
정방향으로 이루어지는 계산 이후에  
역방향으로 간단하게 기울기를 찾는  
방법이 고안됨



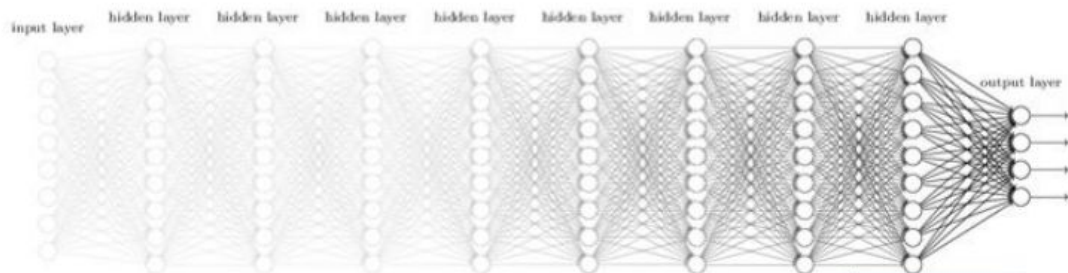
오류 역전파 알고리즘  
(Backpropagation Algorithm; BP)

# 다층퍼셉트론

## ● 기울기 소실



Deep Neural Network



Vanishing Gradient

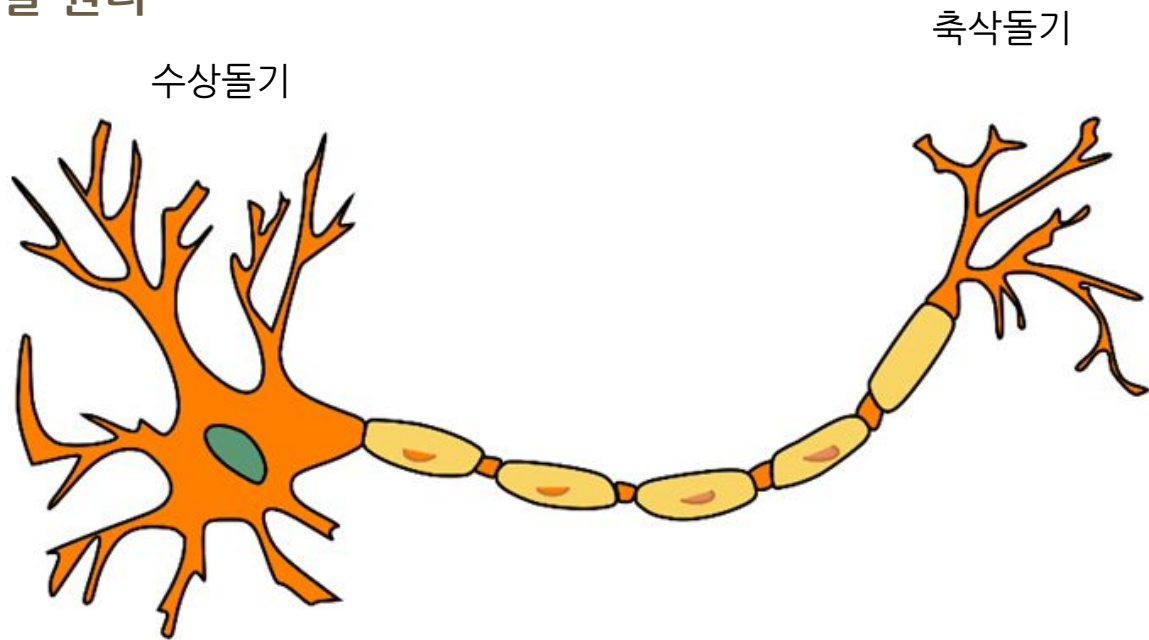
Backpropagation

계층이 깊어질 수록 학습이 어려워지는 이유는  
기울기 소실(Vanishing Gradient)이 발생하기 때문

# 신경망

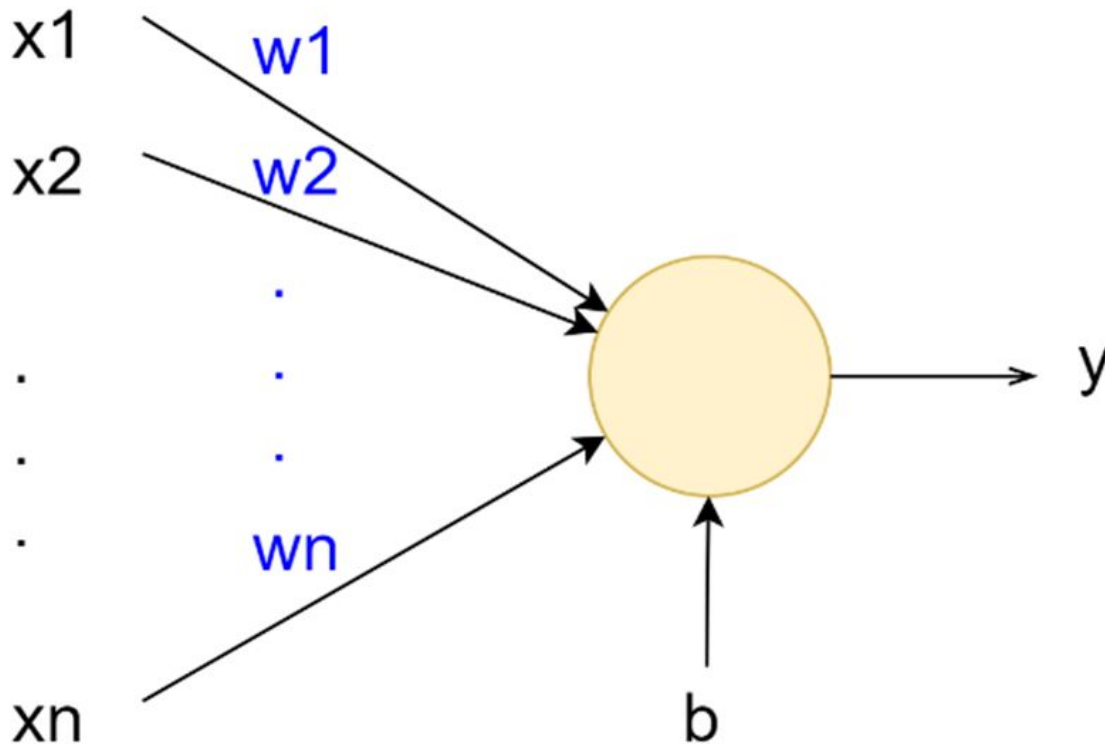
# 인간의 신경세포, 뉴런

## ● 뉴런의 신호 전달 원리



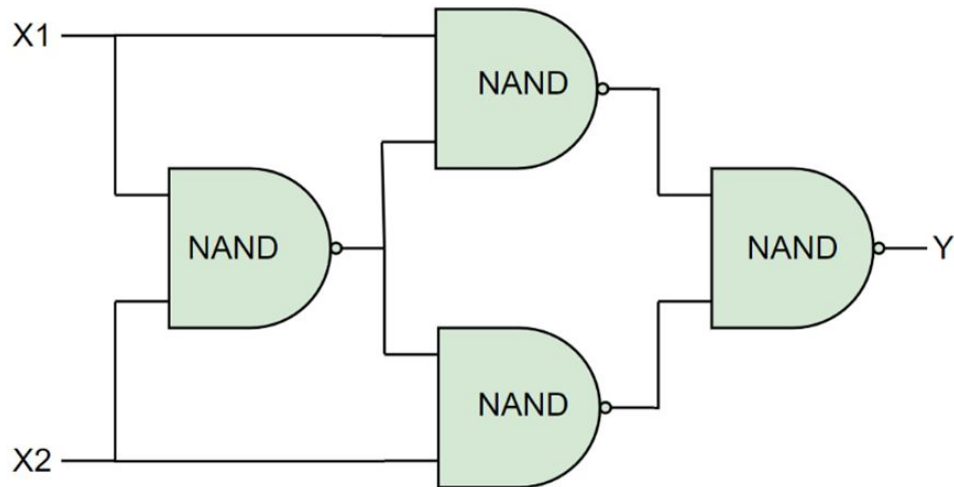
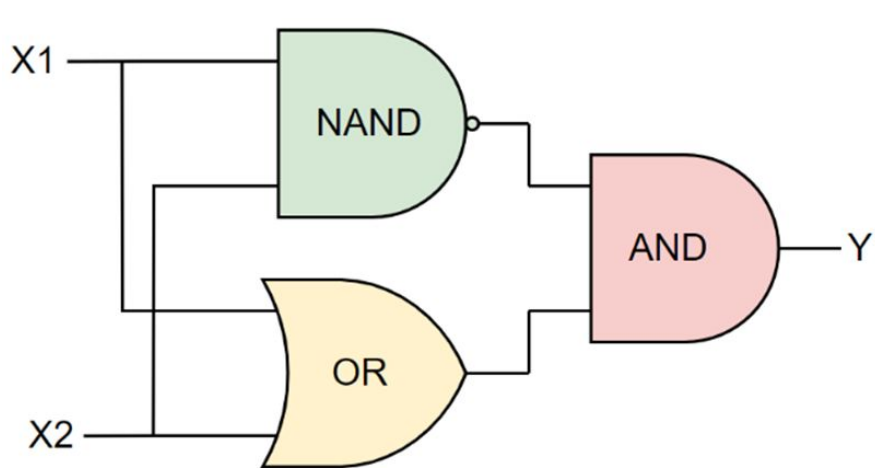
# 인공신경망(ANN)

## ● 뉴런을 닮은 퍼셉트론



# 퍼셉트론과 논리게이트

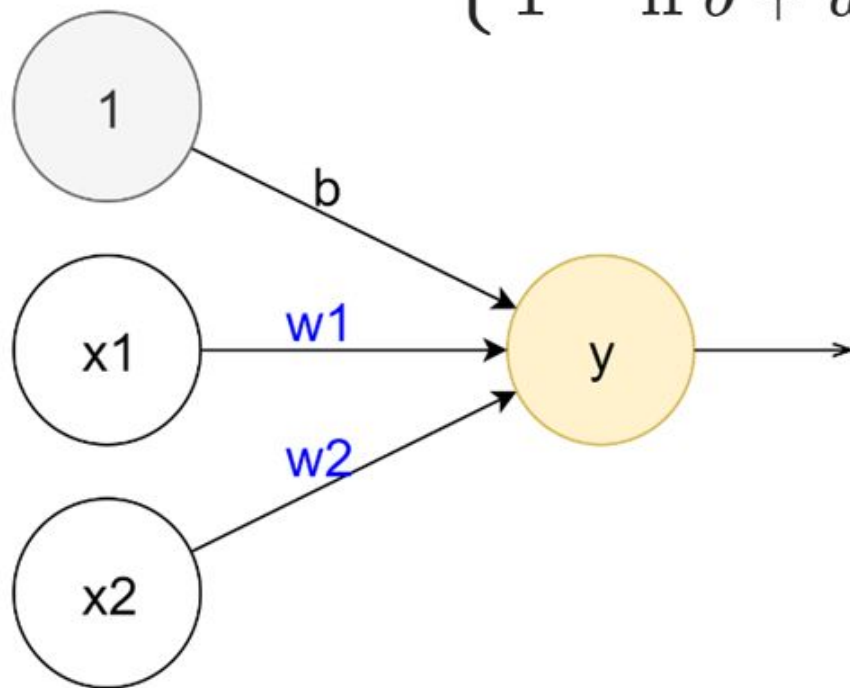
## ● 결과가 동일한 두 가지 신호 전달 원리



# 신경망의 수학적 이해

## ● 수학적 이해

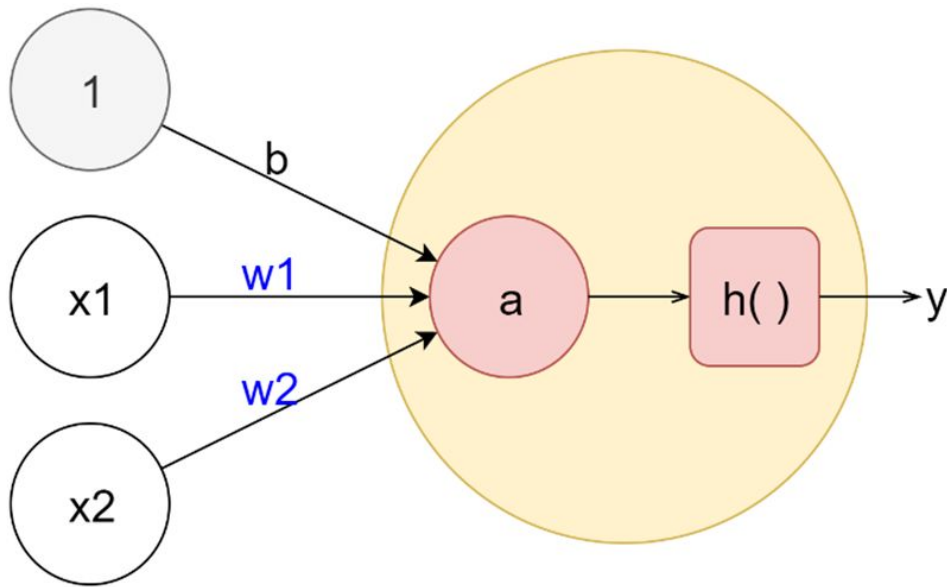
$$y = \begin{cases} 0 & \text{if } b + w_1x_1 + w_2x_2 \leq 0 \\ 1 & \text{if } b + w_1x_1 + w_2x_2 > 0 \end{cases}$$



# 활성함수

## ● 활성함수를 포함한 퍼셉트론

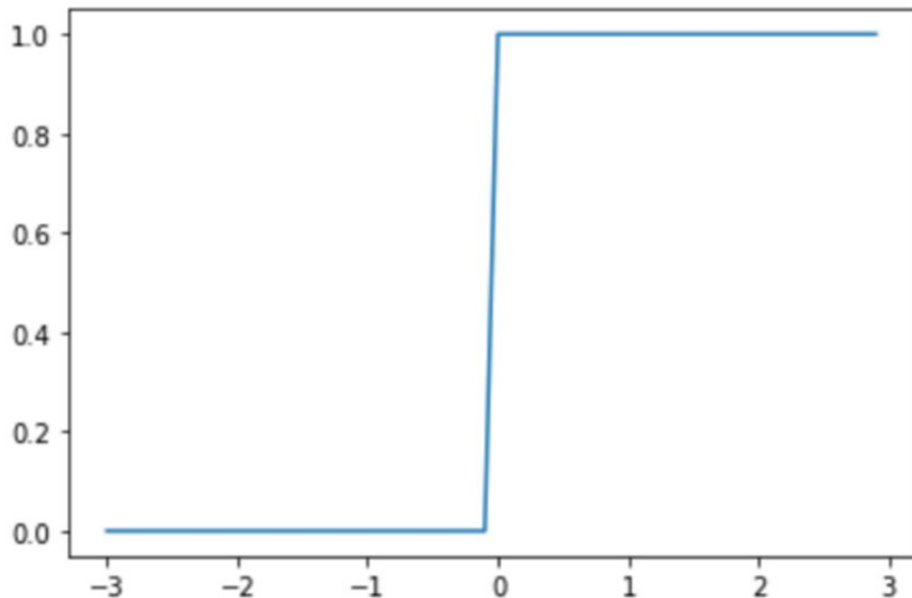
$$y = \begin{cases} 0 & \text{if } b + w_1x_1 + w_2x_2 \leq 0 \\ 1 & \text{if } b + w_1x_1 + w_2x_2 > 0 \end{cases}$$





# 활성함수

## 5.1 계단 함수

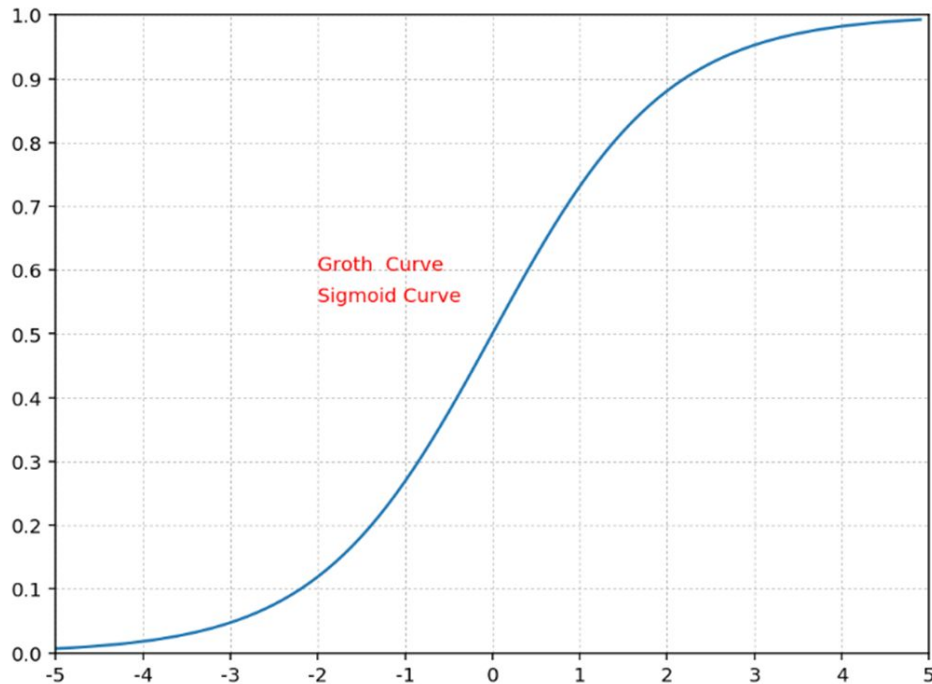


$$h(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 & \text{if } x > 0 \end{cases}$$

$$\frac{f(x) - f(x-h)}{h}$$

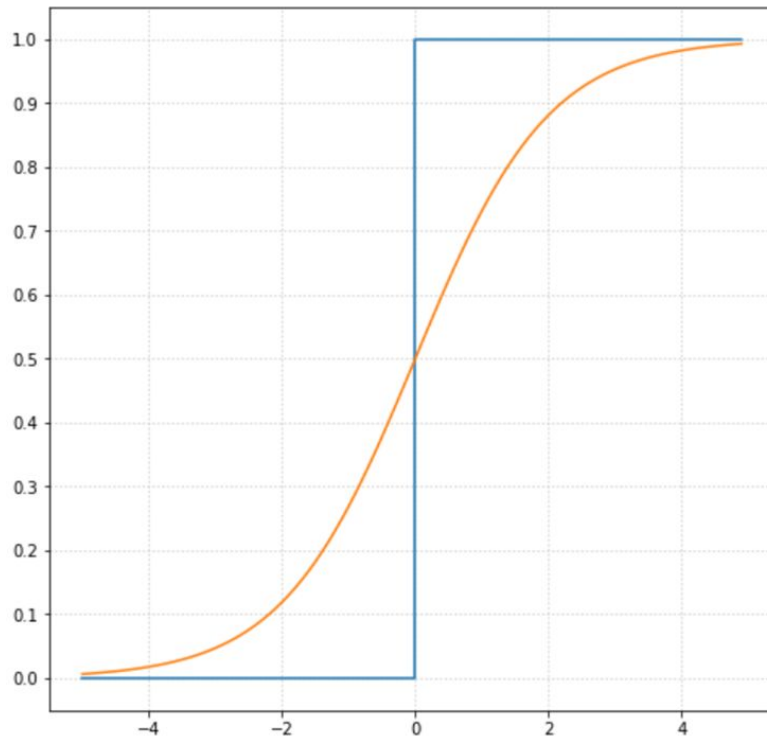
# 활성함수

## ● 시그모이드 함수



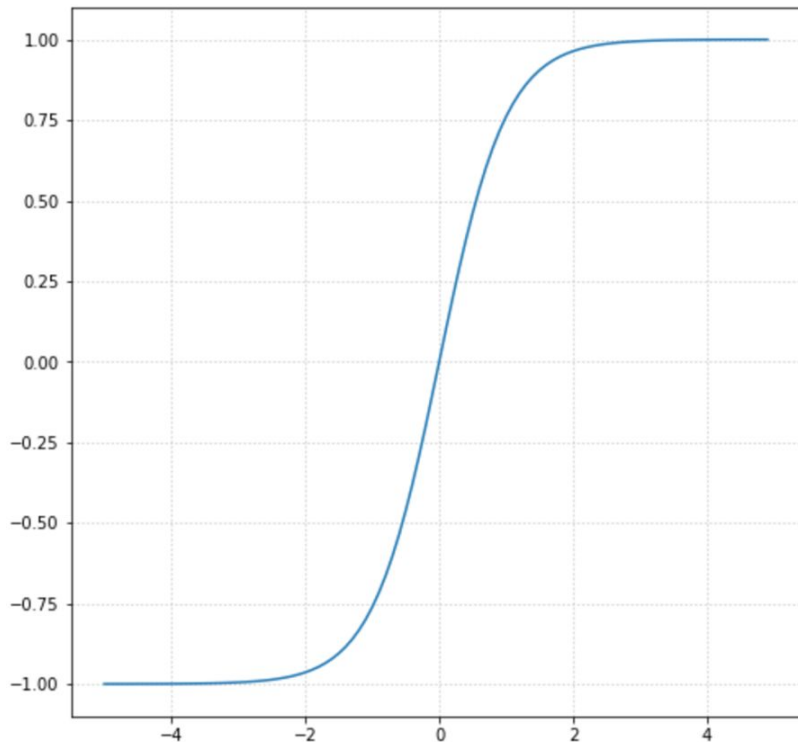
$$s(x) = \frac{1}{1 + e^{-x}}$$

## ● 시그모이드와 계단함수 비교



# 활성함수

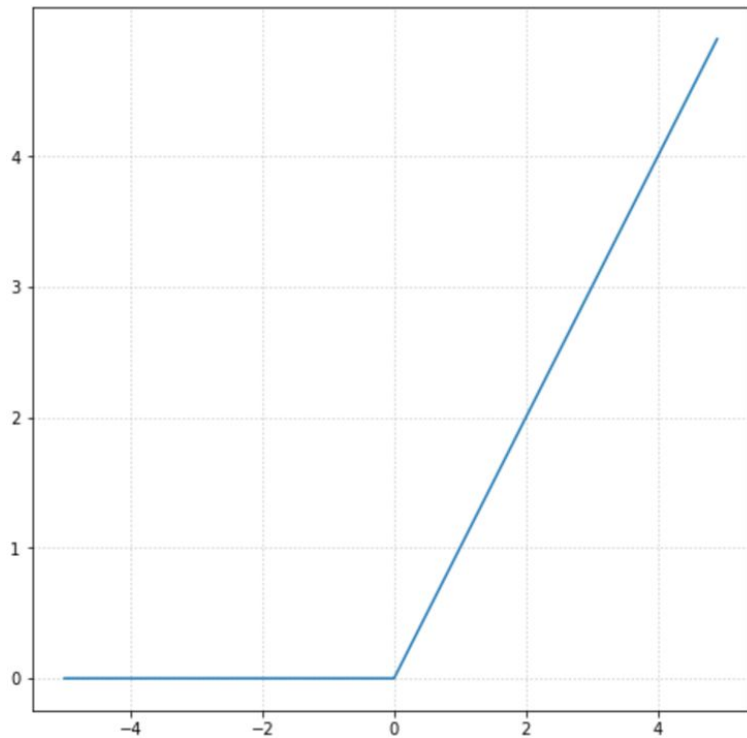
## ○ 하이퍼볼릭탄젠트(tanh)



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

# 활성함수

## ● ReLu



$$r(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$