밑바닥 부터 시작하는 딥러닝1

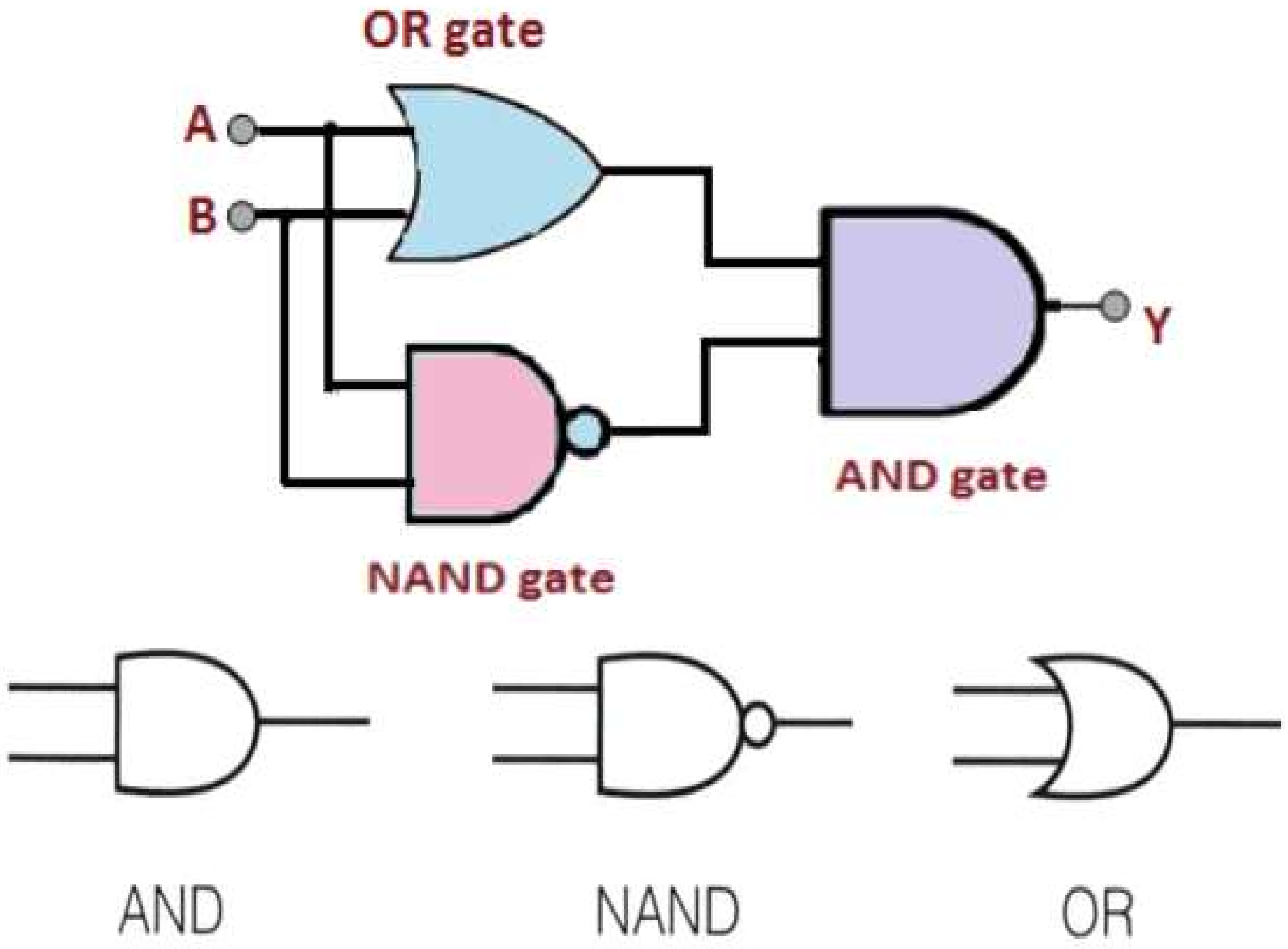
3장 신경망

학습자: 이관형

목차

2장 퍼셉트론 문제 해결

x_1	x_2	NAND s_1	OR s_2	AND y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0



경우의 수: 3 * 2 * 1(비복원 추출)

but 은닉층을 담당하는 게이트 두개는 순서 바뀌어도 상관 없음. 따라서 3가지 경우의 수만 발생(중복 제거)

경우의 수(OR과 NAND의 순서를 바꾸는건 됨)

1. (AND, NAND) - > (OR)일때
2. (AND, OR) -> (NAND)일때
3. (OR, NAND) -> (AND) -> 이건 XOR구현의 기본 원리

따라서 3가지 중 2가지만 보면 됨.

2장 퍼셉트론 문제 해결

```
for i in np.array([[0, 0],
                   [1, 0],
                   [0, 1],
                   [1, 1]]):
    y=XOR(i[0], i[1])
    print(str(i), '->', str(y))
```

```
[0 0] -> 1
[1 0] -> 1
[0 1] -> 1
[1 1] -> 1
```

경우의 수(OR과 NAND의 순서를 바꾸는건 됨)
1. (AND, NAND) - > (OR)일때
2. (AND, OR) -> (NAND)일때
3. (OR, NAND) -> (AND) -> 이건 XOR구현의 기본 원리
따라서 3가지 중 2가지만 보면 됨.

```
for i in np.array([[0, 0],
                   [1, 0],
                   [0, 1],
                   [1, 1]]):
    y=XOR(i[0], i[1])
    print(str(i), '->', str(y))
```

```
[0 0] -> 1
[1 0] -> 1
[0 1] -> 1
[1 1] -> 0
```

아니면 실제로 3게이트를
만족하는 가중치 편향이 모두
다름 -> 바꿔도 될 수가 없음

활성화 함수

h(x)의 이해

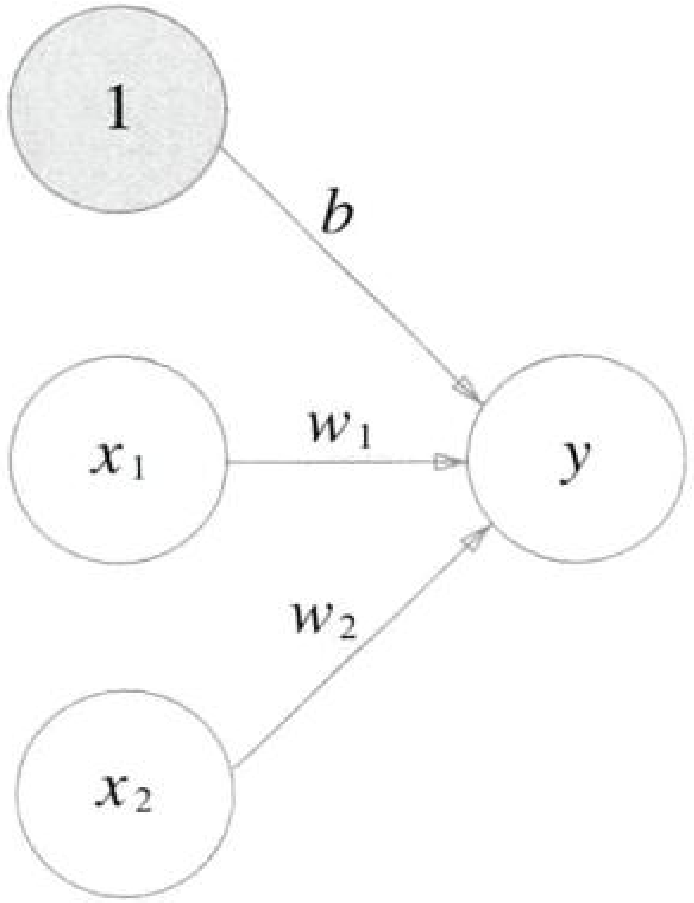
계단 함수

시그모이드 함수

비선형 그래프 사용 이유

ReLU 함수

편 향 을 명 시 하 는 퍼 셉 트 론



$$y = h(b + w_1x_1 + w_2x_2)$$

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

편향을 명시
편향도 가중치 관점에서

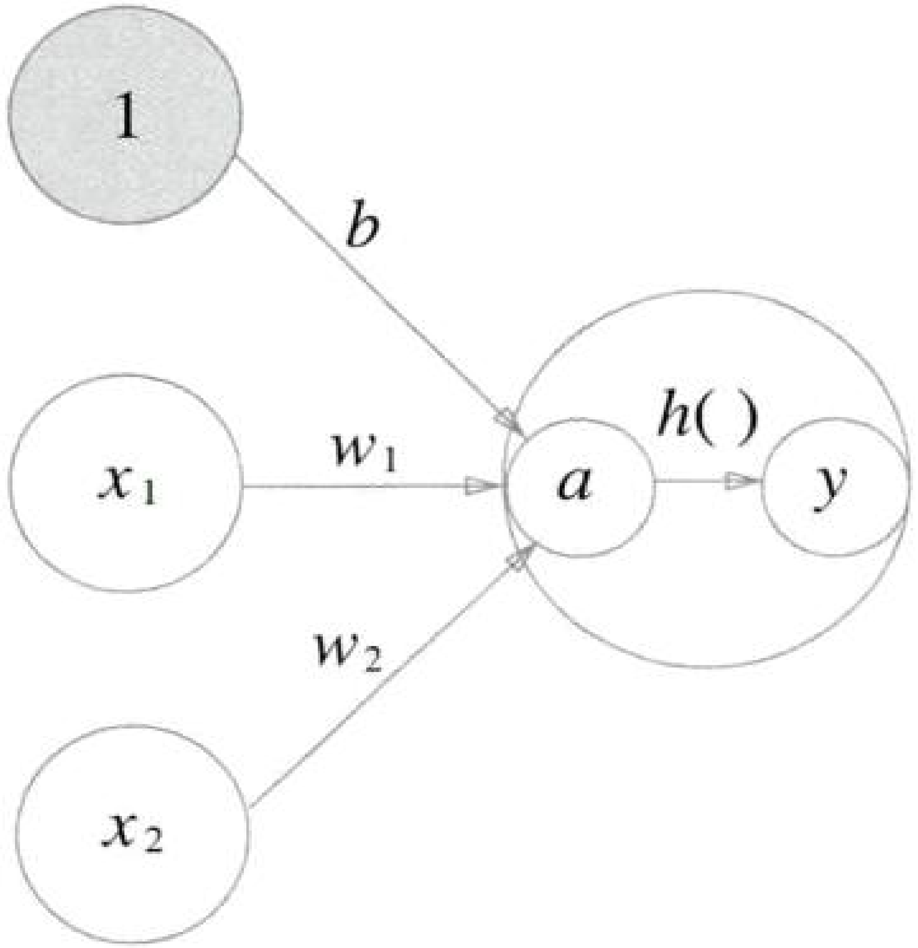
편향의 입력신호는 항상 1이다.
(1*b)

계단 함수(Heaviside): 극단적 (모 아니면 도)

앞에서 구한 가중치의 합과 **편향이 h(x) (계단함수)의 새로운 변수 x값이 됨**

x (b+ w1x1 + w2x2) > 0 이상일 시 출력 1
x (b+ w1x1 + w2x2) <= 0 이상일 시 출력 0

활성화 함수



왜 사용할까?

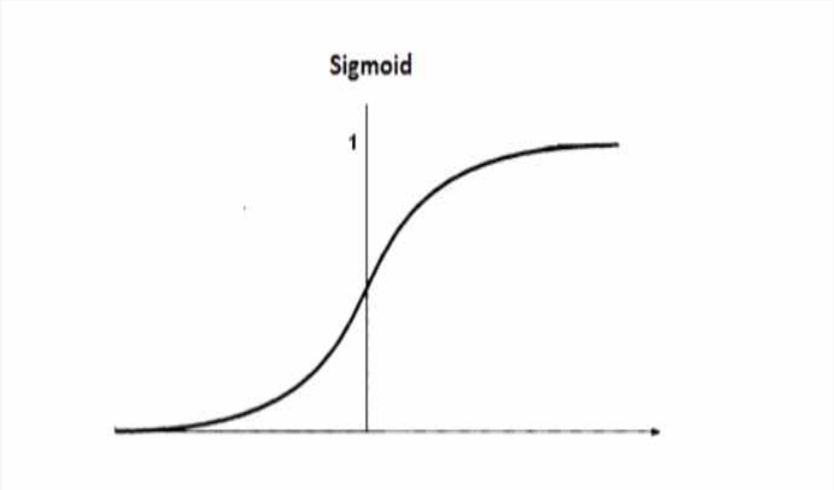
입력 신호에 따른 총합의 가능한 경우의 수
실수 ∞ (음수 ~ 양수)

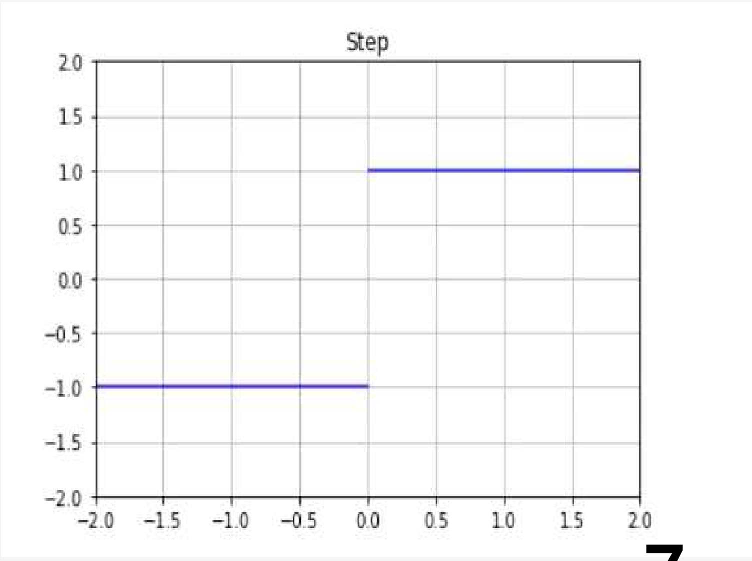
기본 적인 인공 신경망 로직은 분류 (0 OR 1)
함수를 사용하여 값들을 적절 값으로
매핑하기 위해로 이해

신경망 기본 로직

1. 가중치가 곱해진 입력신호의 총합 계산(a)
2. 총합을 활성화 함수에 입력 (h())
3. 결과 도출 h(a)

기본 종류





계 단 함 수(Heaviside)

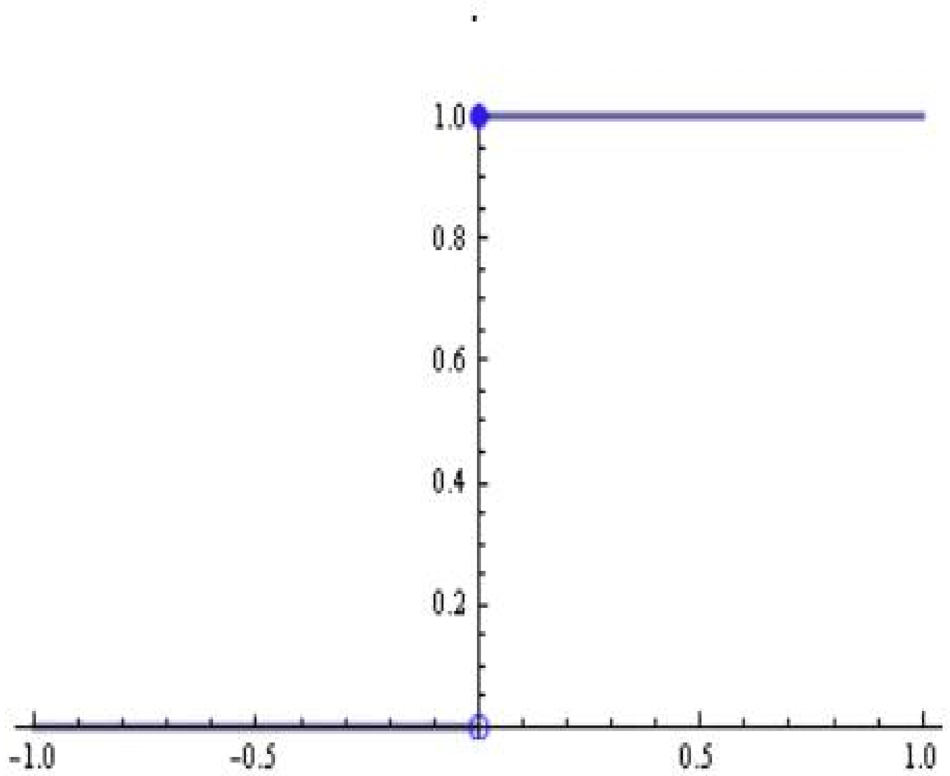


Figure 1: The Heaviside step function

$$y = h(b + w_1x_1 + w_2x_2)$$

$$h(x) = \begin{cases} 0 \ (x \le 0) \\ 1 \ (x > 0) \end{cases}$$

다층 퍼셉트론에서는 잘 안쓰임(->시그모이드)

갑자기 특정 부분에서 툭 끊기는

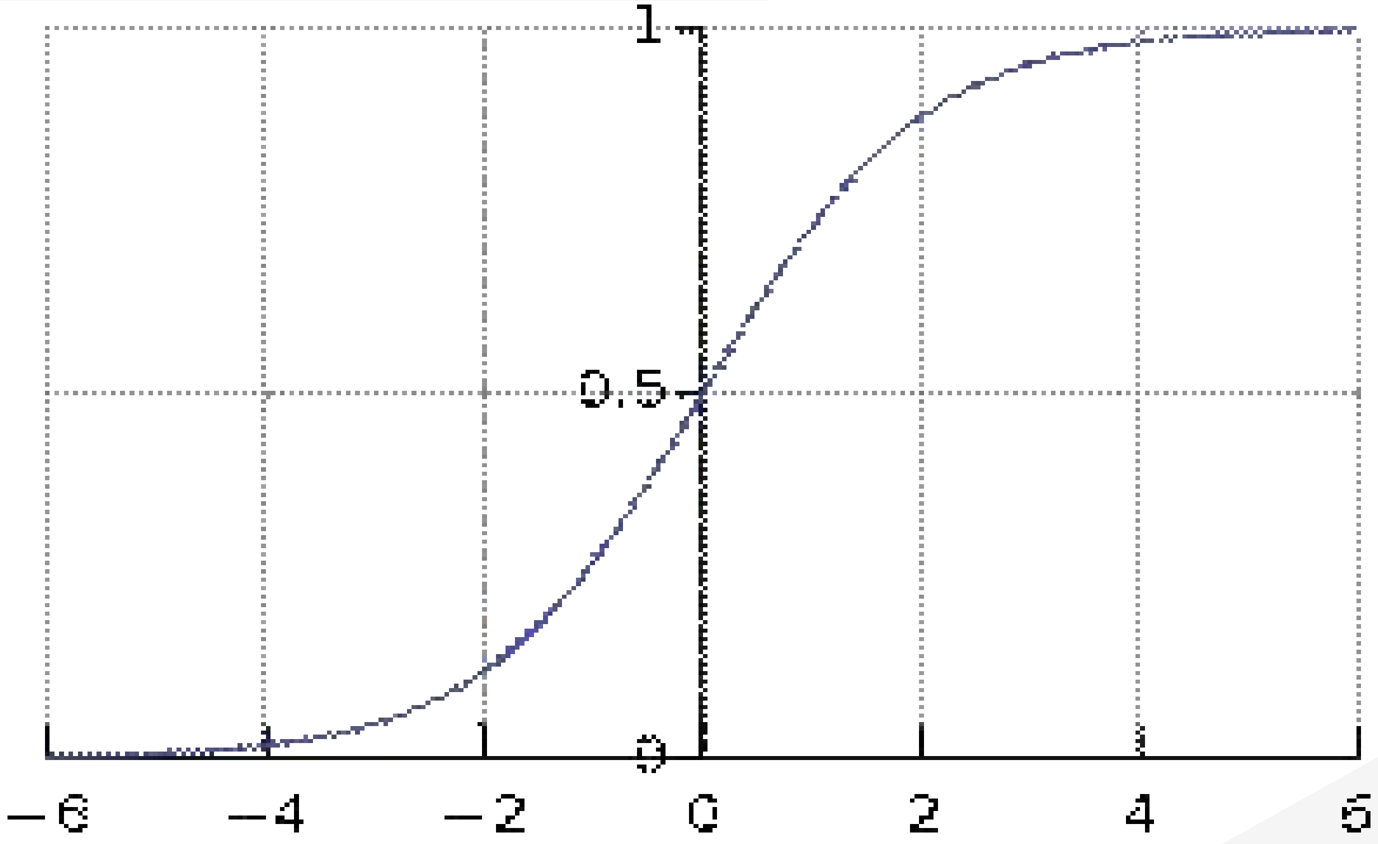
0보다 작으면 스위치 off, 0보다 크면 스위치 on

예시: *시시오도시

if else 구조

if x인자 음수 -> 0
if x인자 양수 -> 1

시그모이드 함수(sigmoid)



h(x) = 1/ 1 + exp(-x)

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

e : 자연 상수(2.7182)
계단 함수에서 끊어진 부분을 부드럽게 연결

왜 사용할까?
학습 시 미분 사용 계단 함수는 부드럽지 않아 미분이 0

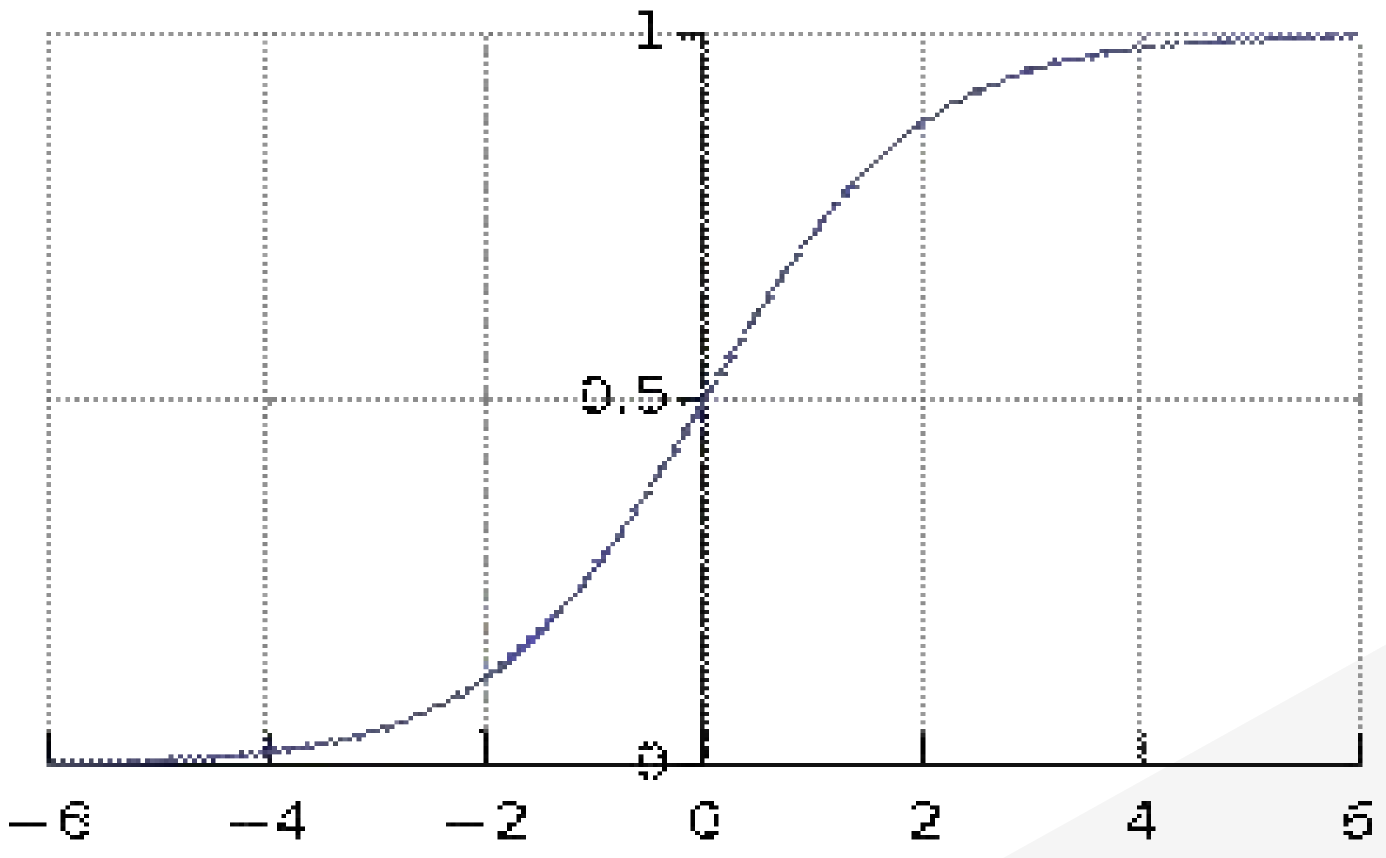
계단 함수보다 정보 보존(0 ~ 1값 빈값이 극단적 변화 x)

활성화 함수
다층 퍼셉트론: 시그모이드
단층 퍼셉트론: 계단 함수

Q. 소프트 맥스는 왜 사용할까?(이미 0~1 확률 값)

예시: 물레방아

시그모이드 그래프 그리기



종합

1. y절편 구하기
2. 점근선 판단
3. 볼록 판단
4. 점근선 판단

활성화 함수 코드 구현(계단 함수)

넘파이 트릭 이용하기

행렬이 0보다 크다 작다?
말이 안됨 how?해결(옆에 코드는 실수일때 사용 가능)

```
# 계단 함수 구현하기
def step_function(x):
    if x > 0:
        return 1
    else:
        return 0
```

```
print(step_function(1)) #실수일때 작동 가능
```

```
1
```

```
def step_function(x):
    y = x > 0
    return y.astype(np.int)
```

True: 1, False: 0 변환
astype으로 bool ->int로 변환

pyplot의 구현 방식

heaviside 수학적 그래프

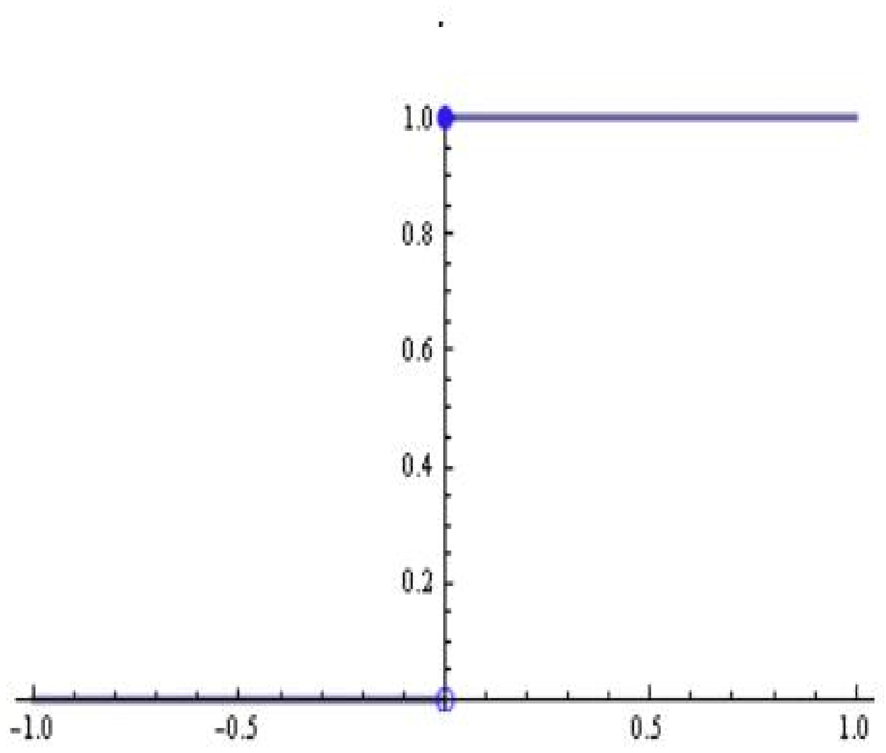
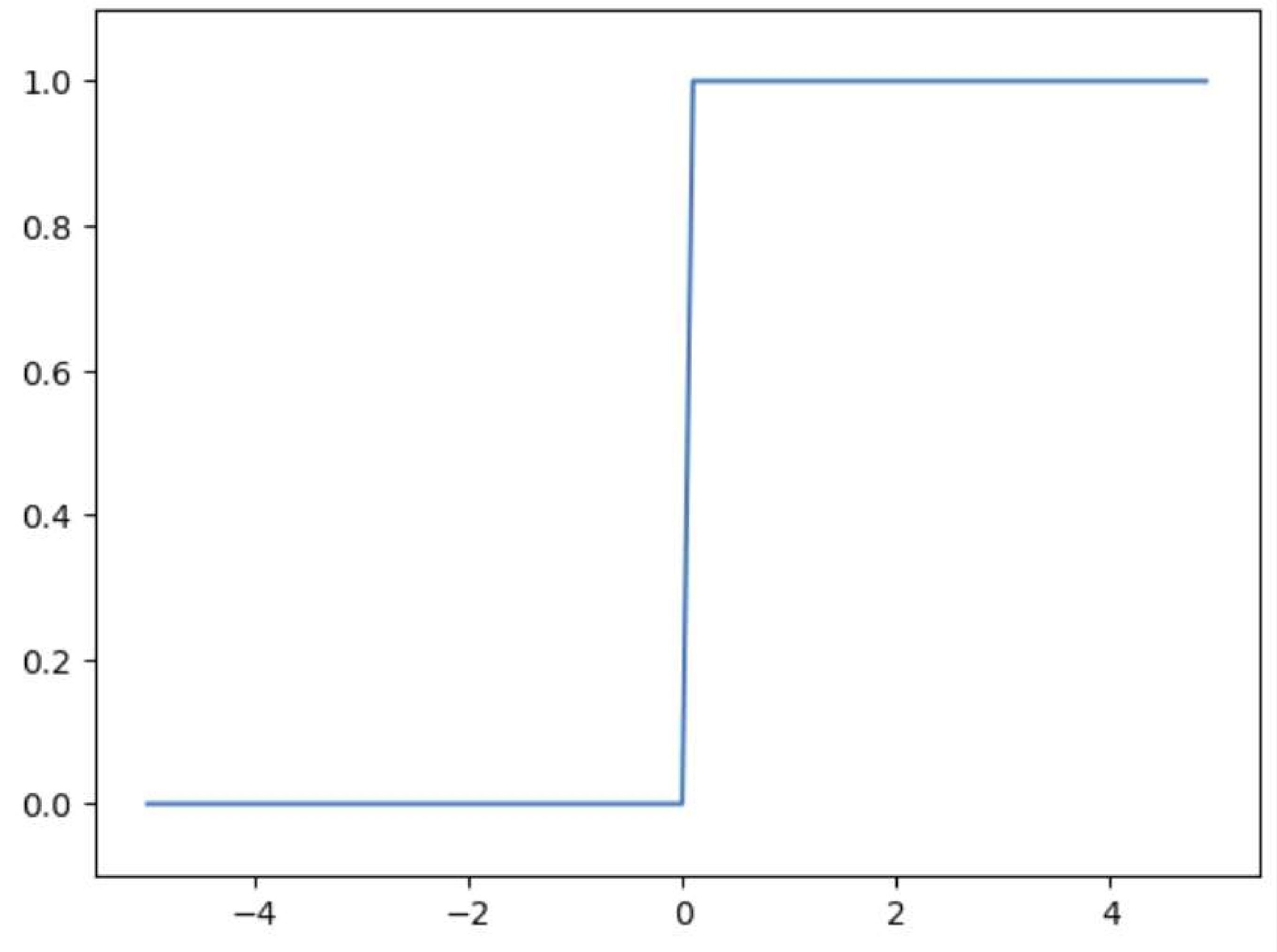


Figure 1: The Heaviside step function

python 시각화



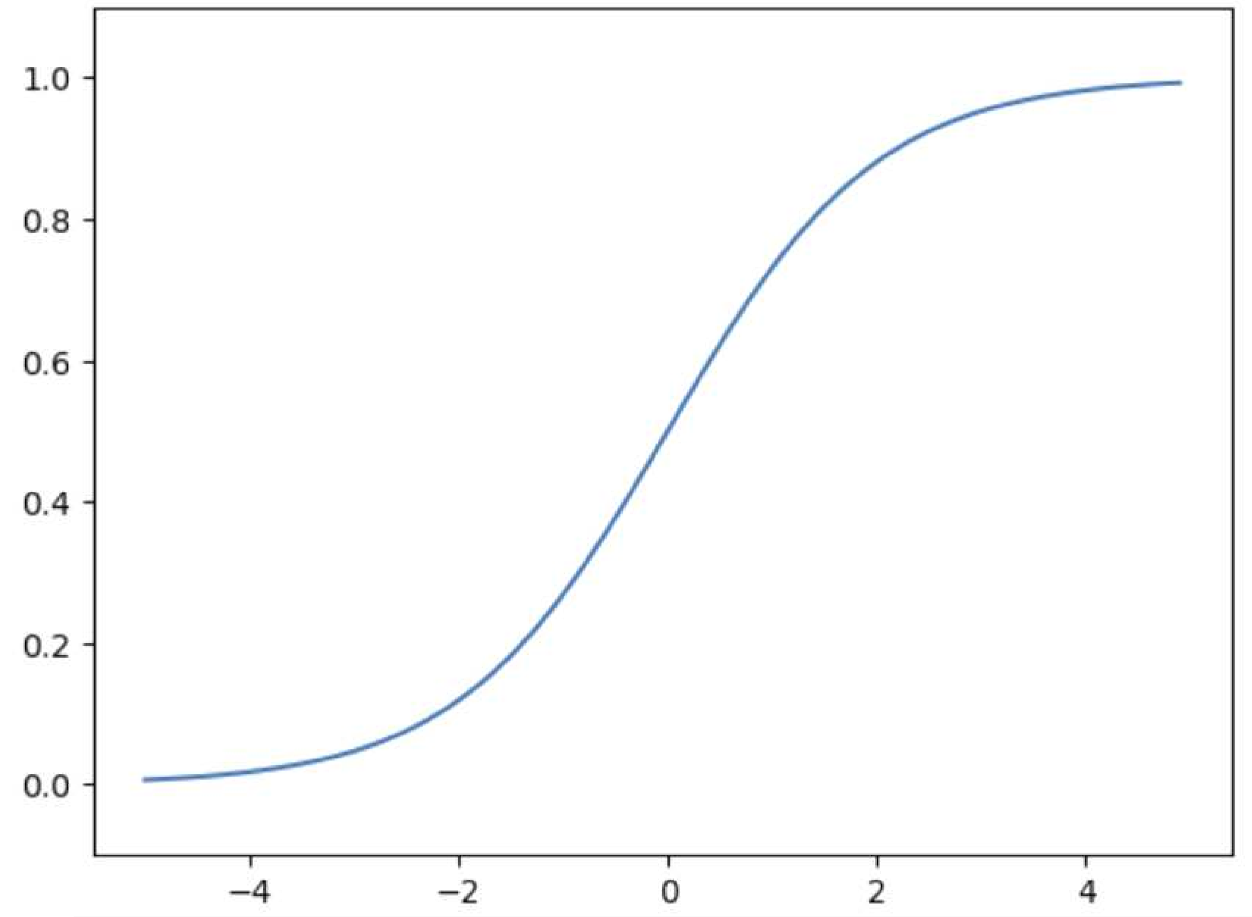
x가 0일때 파이썬 시각화 그림은
선이 연결 돼 있음 왜 그럴까?

python 그래프 그릴때
1. x, y의 값에 따른 점을 찍는다.
2. 그 점을 연결하는 직선을 그린다.

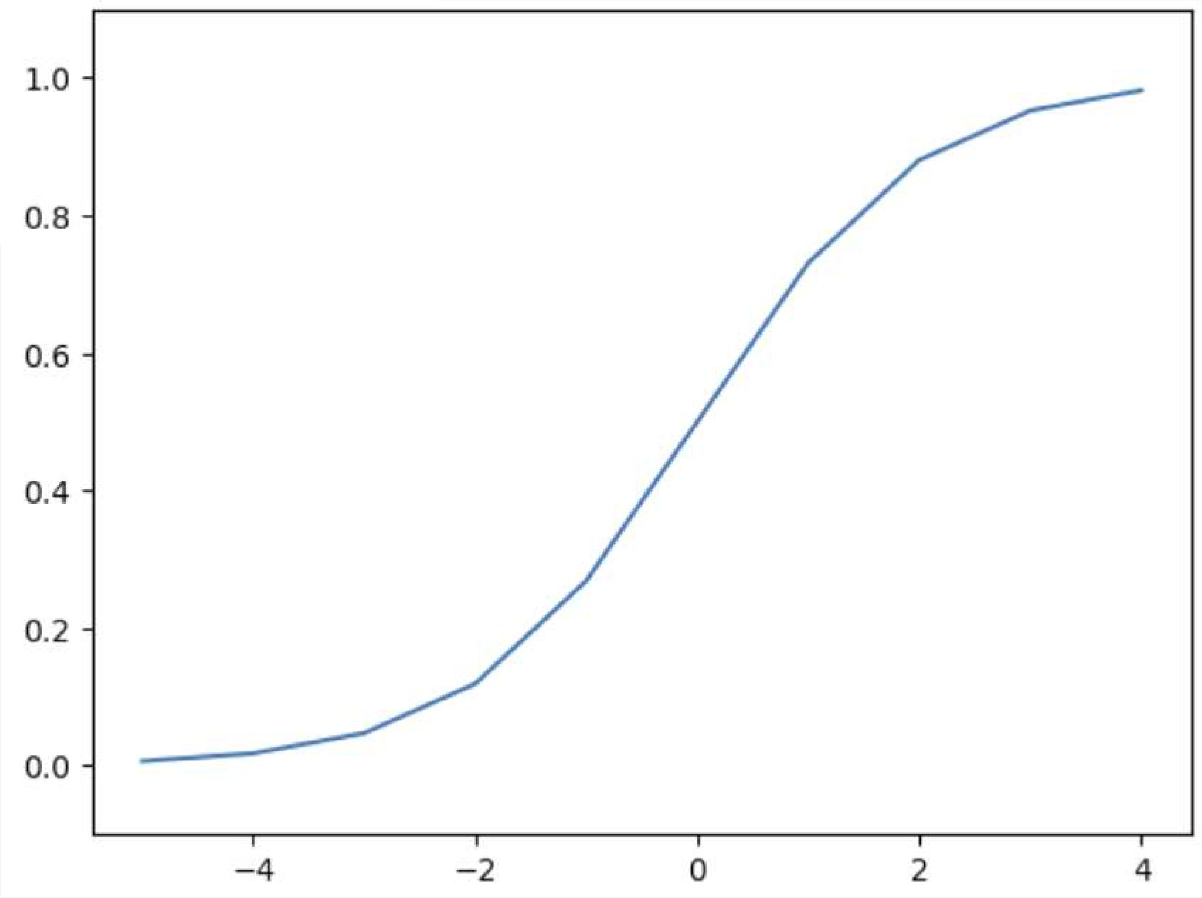
pyplot의 구현 방식

python x의 원소 수에 따른 변화(시그모이드)

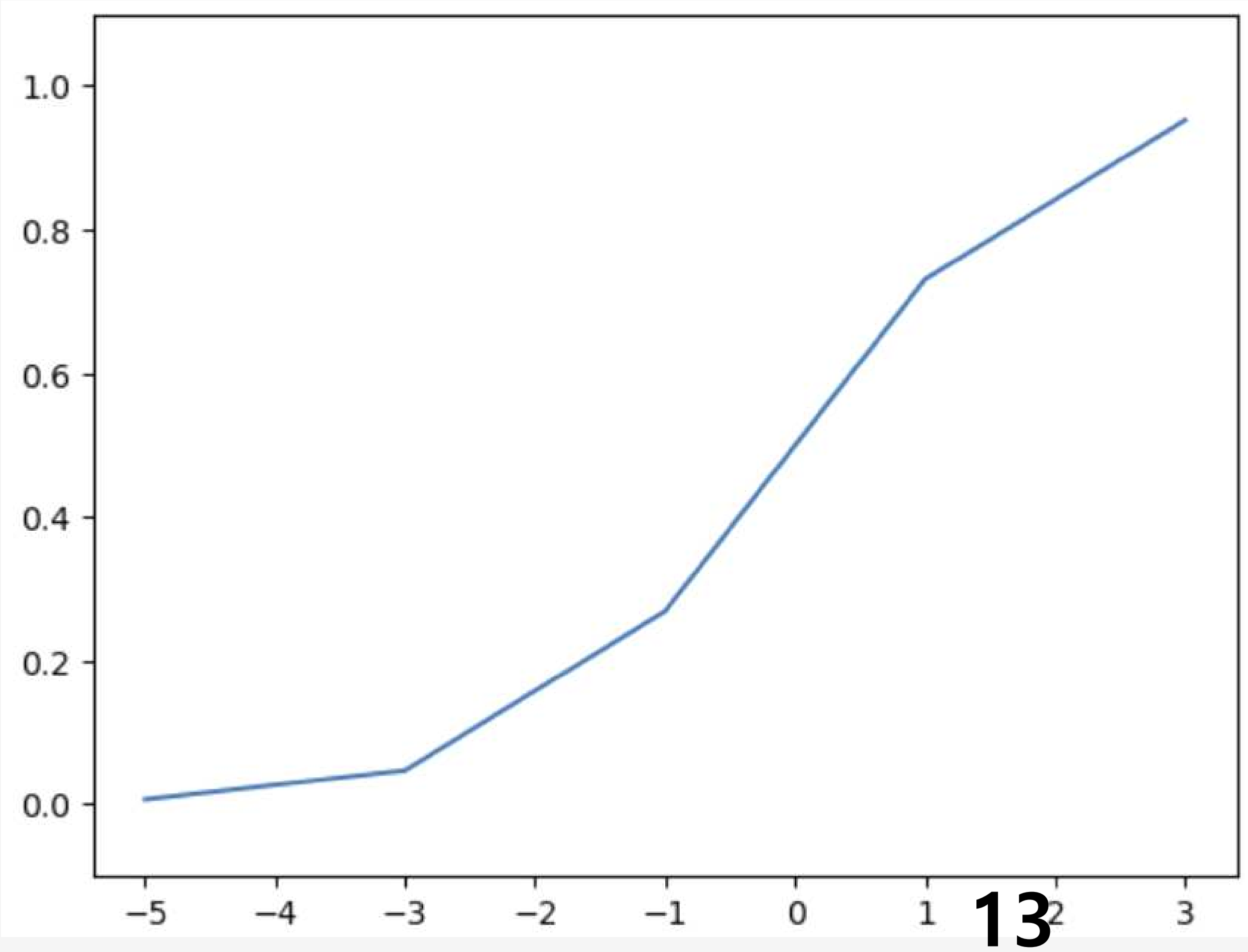
```
x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)
```



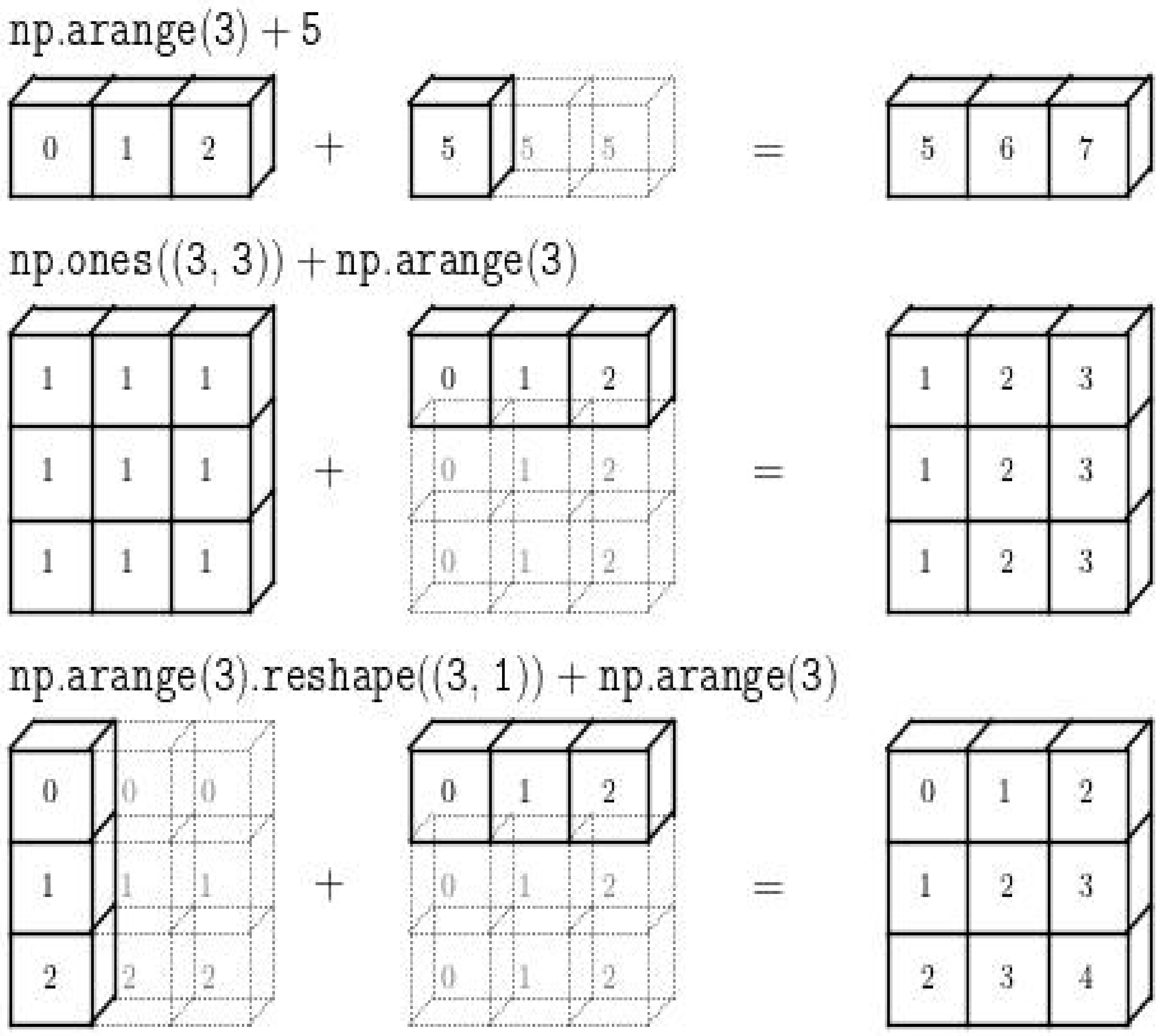

```
x = np.arange(-5.0, 5.0, 1)
y = sigmoid(x)
```




```
x = np.arange(-5.0, 5.0, 2)
y = sigmoid(x)
```



python 브로드 캐스트



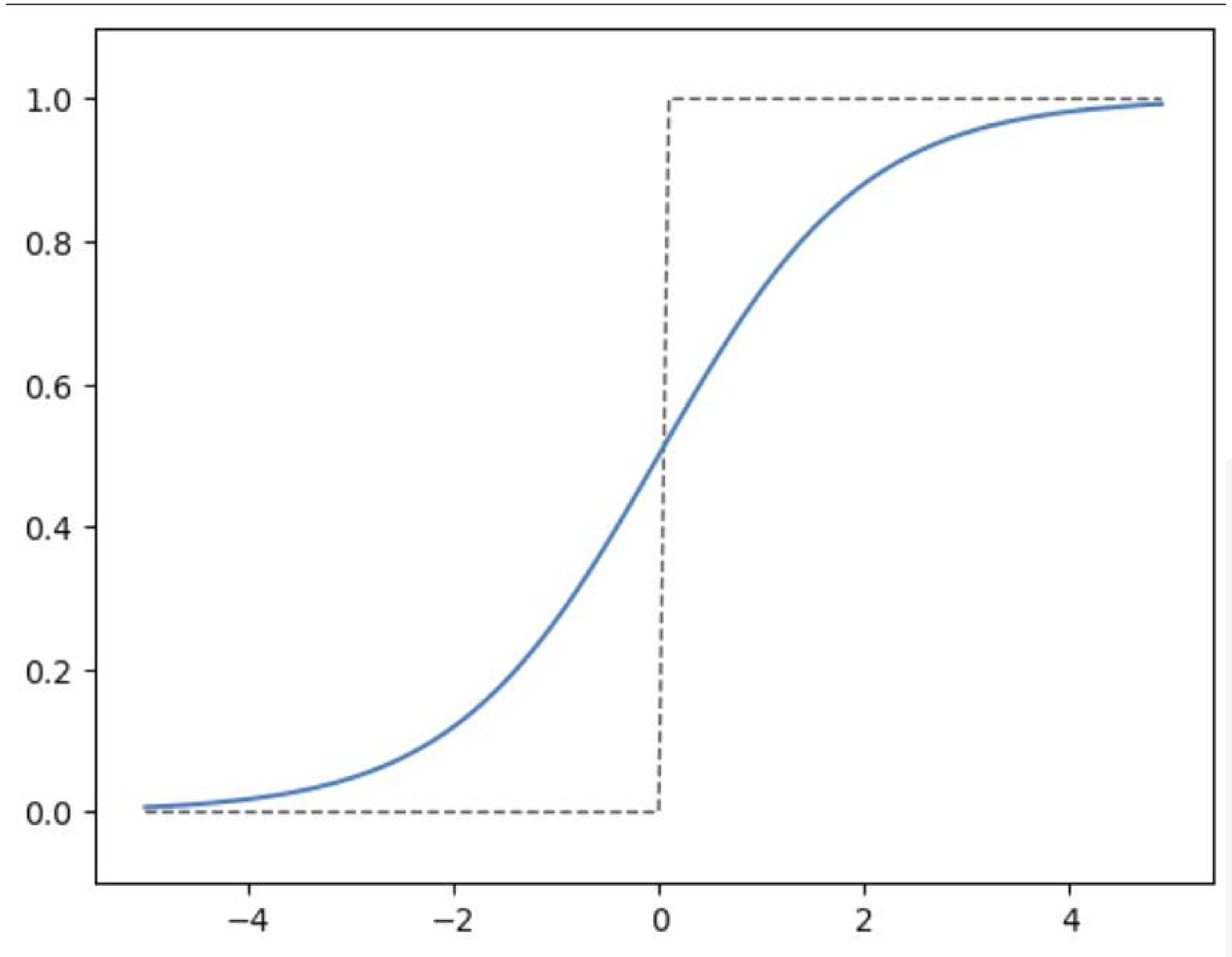
x = np.arange(-5.0, 5.0, 2)
y = sigmoid(x)

>각 행렬을 시그모이드 인자로? 수학에서는 이렇게 안됨.
but 파이썬 브로드캐스트 -> 구현

ex) [1.0, 2.0, 3.0]행렬
1/exp(-x) 각 원소마다 적용

[1.0, 2.0, 3.0]
1/exp(-x), 1/exp(-x), 1/exp(-x)이런식으로
브로드캐스트 가능

비선형 함수



계단, 시그모이드 둘 다 비선형 함수

비선형 함수?

선형이 아닌 함수

선형 함수 : 입력했을 때 출력이 **상수배** 만큼 변하는 함수
ex) y = c^3 * x
즉, 상수배로 표현할 수 없는 함수

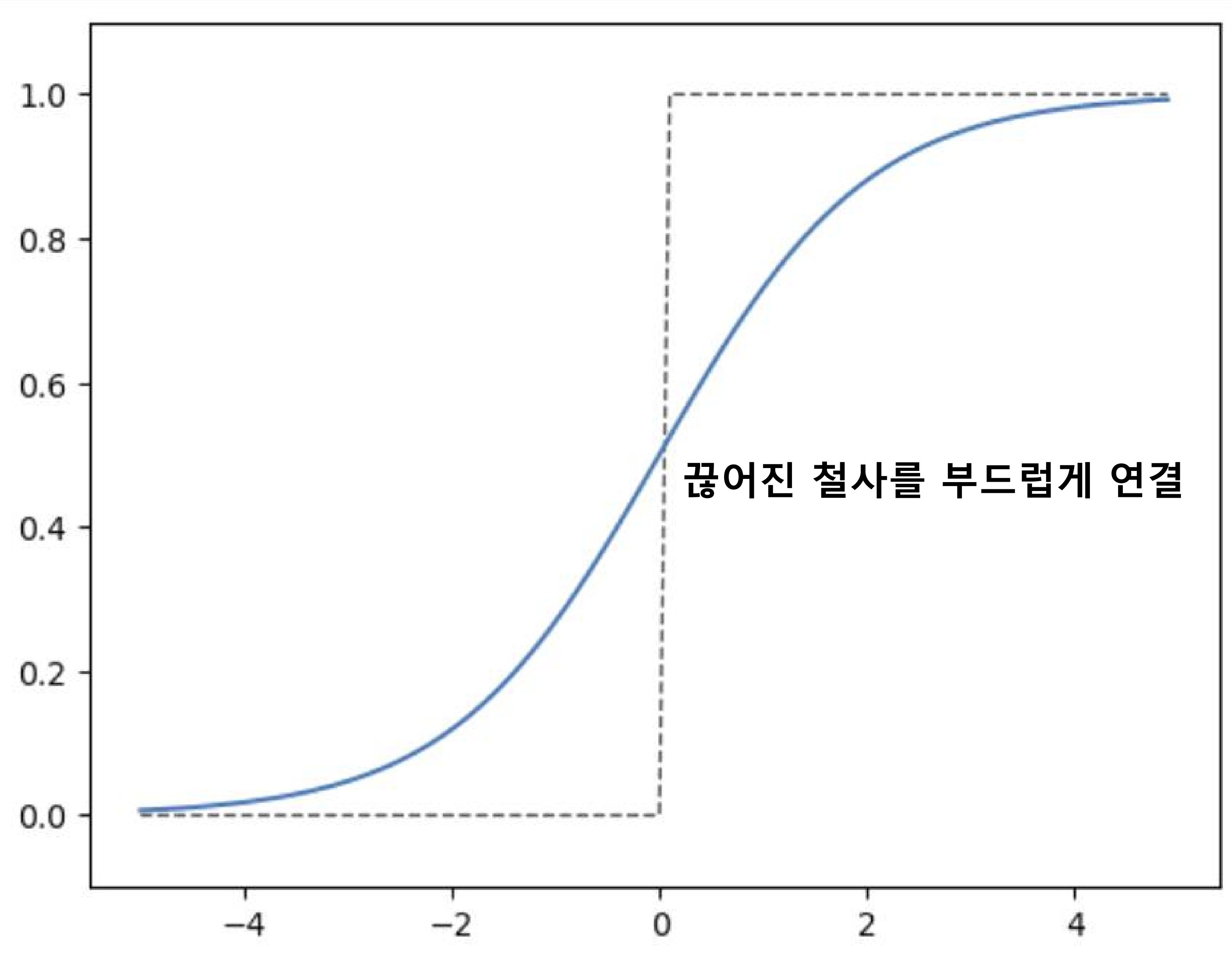
WHY?
비선형 함수 사용 이유: **신경망의 층을 깊게 하는 의미가 없어짐. -> 선형 함수를 사용해서는 안됨.**

층을 쌓지 않아도 직선으로 표현되면, 층을 쌓는 것이 비효율적 연산 속도만 잡는다.(층을 아무리 쌓아도 은닉층이 없는 네트워크와 동일하다.)

y = ax + b의 식으로 표현 가능
ex) y = c^3 * x 이것도 결국 y= ax + b로 표현가능
h(h(h(x))) = c * c * c * x처럼 곱 세번 따지고 보면
a = c^3으로 표현 가능

Y = aX^이런 것도 비선형임..
몰랐음

시그모이드 함수와 계단함수 비교



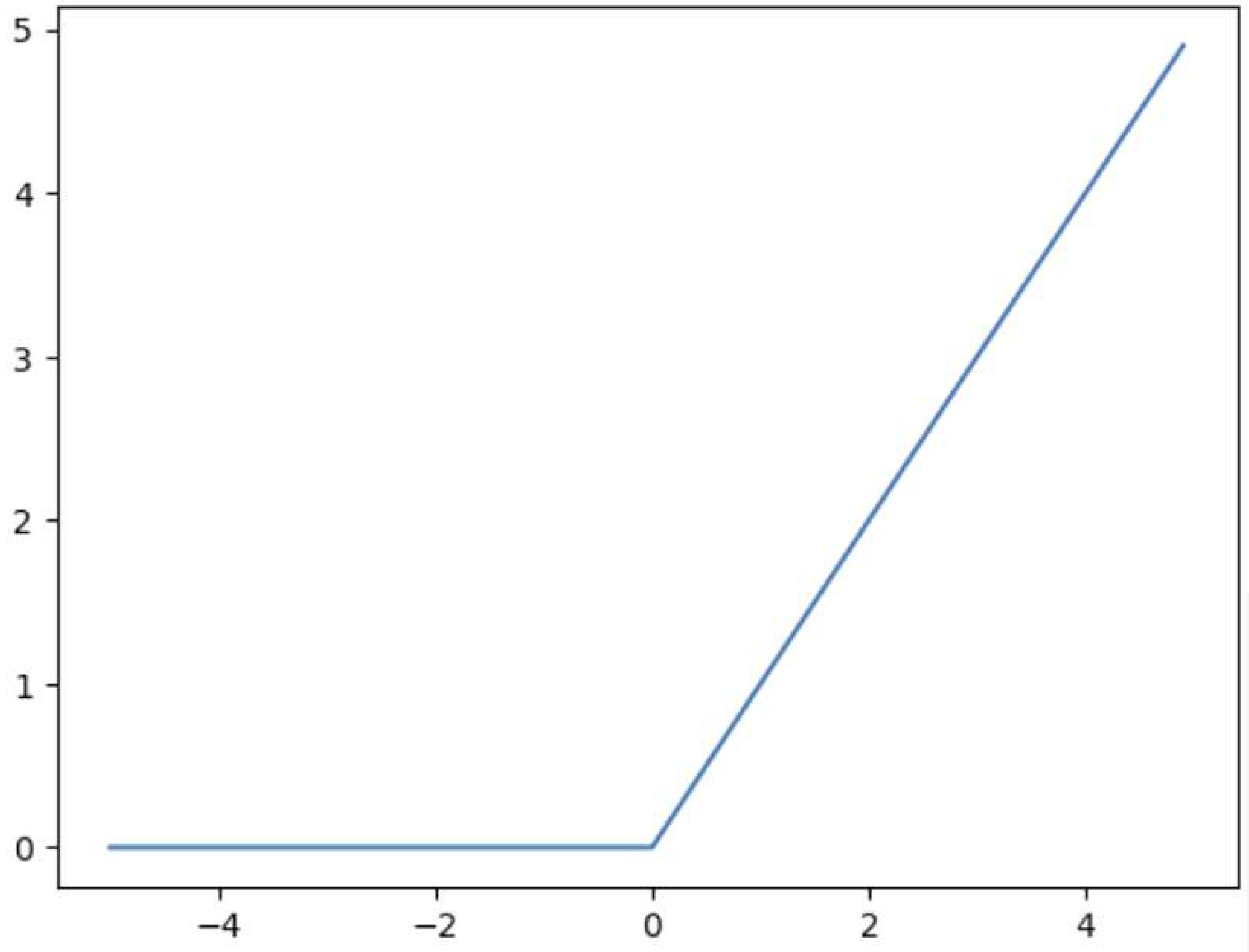
차이점: 매끄러움

계단 함수 0 or 1 반환
시그모이드 함수 : 0~1사이 연속적인 실수 값 반환

공통점: 입력이 작을 때 -> 출력 0에 가까움
입력이 클 때 -> 출력 1에 가까움

즉, 중요도에 따라 출력하는 값 0~1

ReLU함수의 등장



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

시그모이드 기울기 소실 문제 해결 방안

입력이 0 이상 -> 항등 함수

입력이 0 이하 -> 0반환

신경망에서 행렬

행렬의 이해

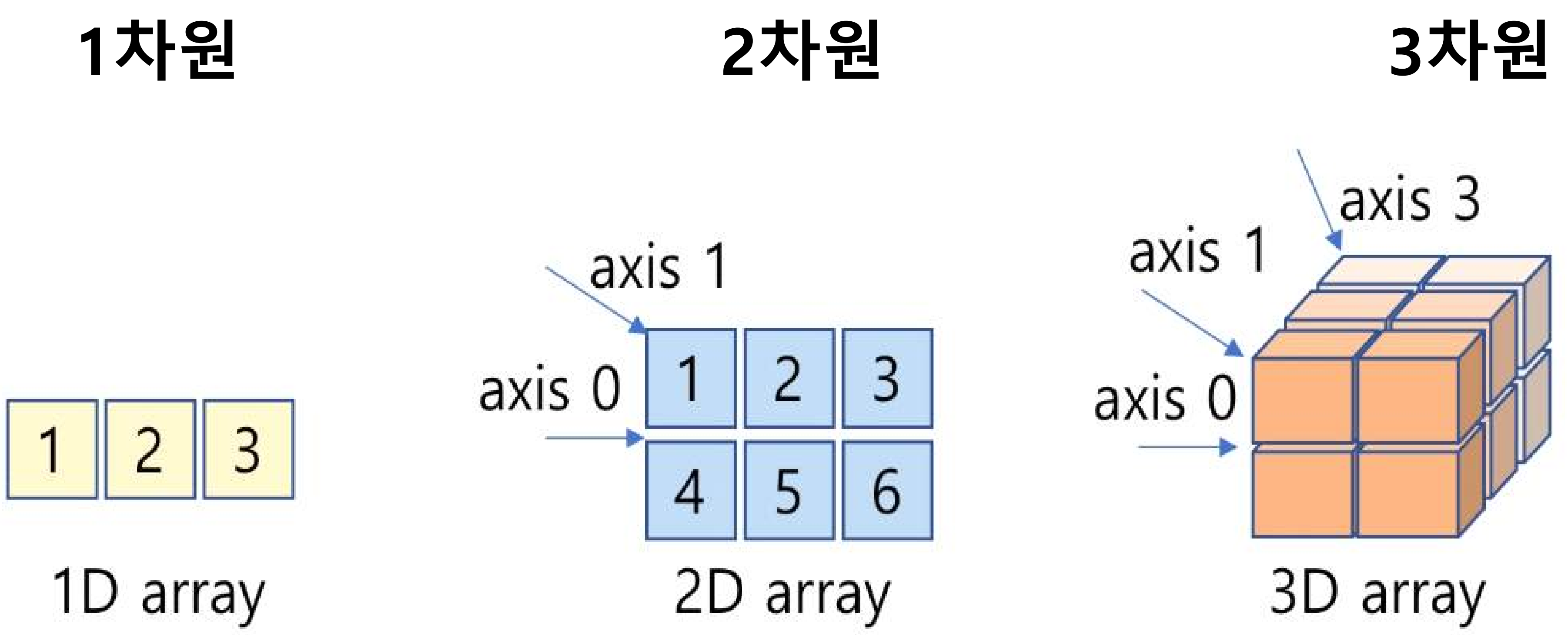
신경망에서 행렬

3층 신경망?

신경망에서 신호 전달 과정

행 렬 의 이 해

numpy 관점




```
np.array([1, 2, 3, 4, 5]) #1차원 선
np.array([
    [1, 2, 3, 4, 5],
    [1, 2, 3, 4, 5],
    [1, 2, 3, 4, 5]
])
#2차원 면
```

이해 point

0차원: **원소**(점)
1차원 : **선**(수많은 점)
2차원: **면**(수많은 선) -> **행렬**
3차원: **입체모형**(수많은 면)

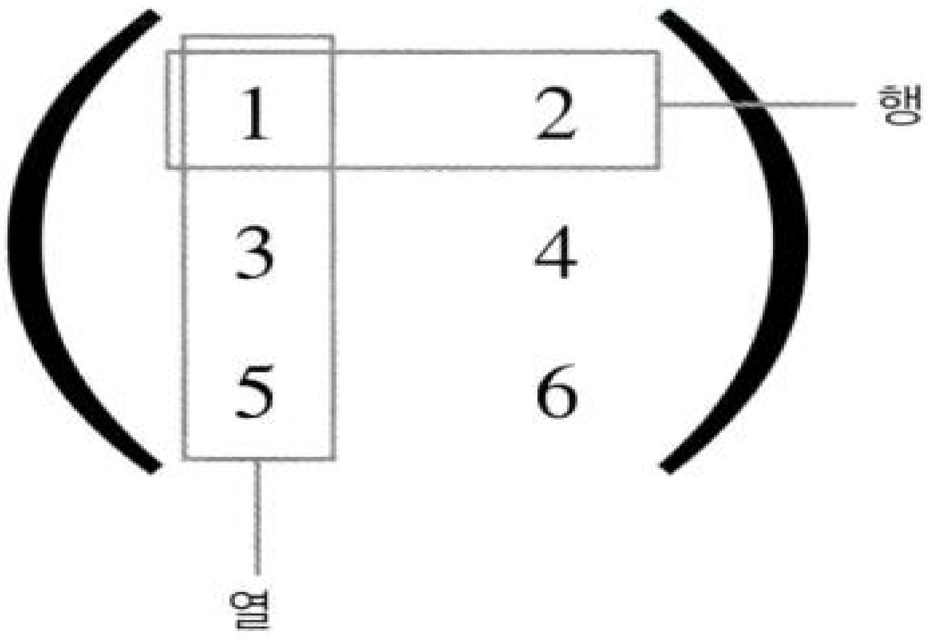
하나의 리스트 선으로 보면
리스트 안 여러개의 리스트 면

흥미롭다..

행 렬 의 이 해

행렬

그림 3-10 2차원 배열(행렬)의 행(가로)과 열(세로)



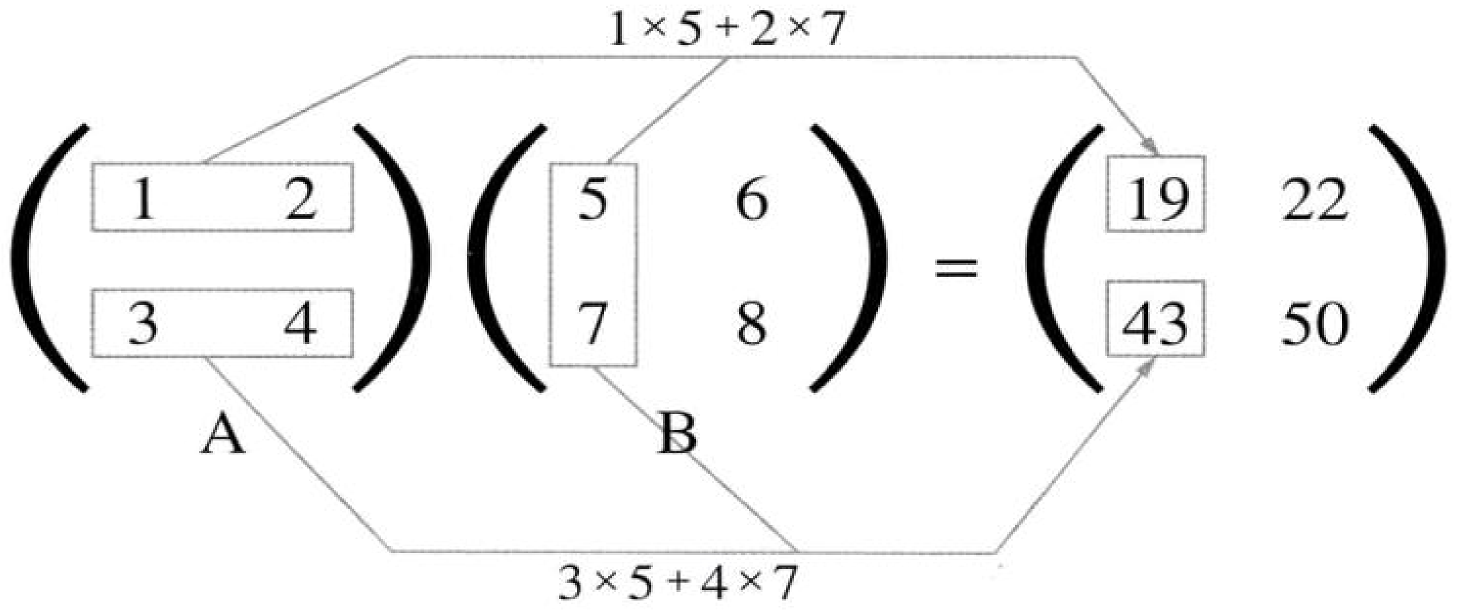
이해 point

데이터 프레임이라고 생각
index와 column이 없는
데이터 프레임

넘파이도 이런식으로 이해

행렬의 연산(합, 상수배, 곱)

그림 3-11 행렬의 곱 계산 방법



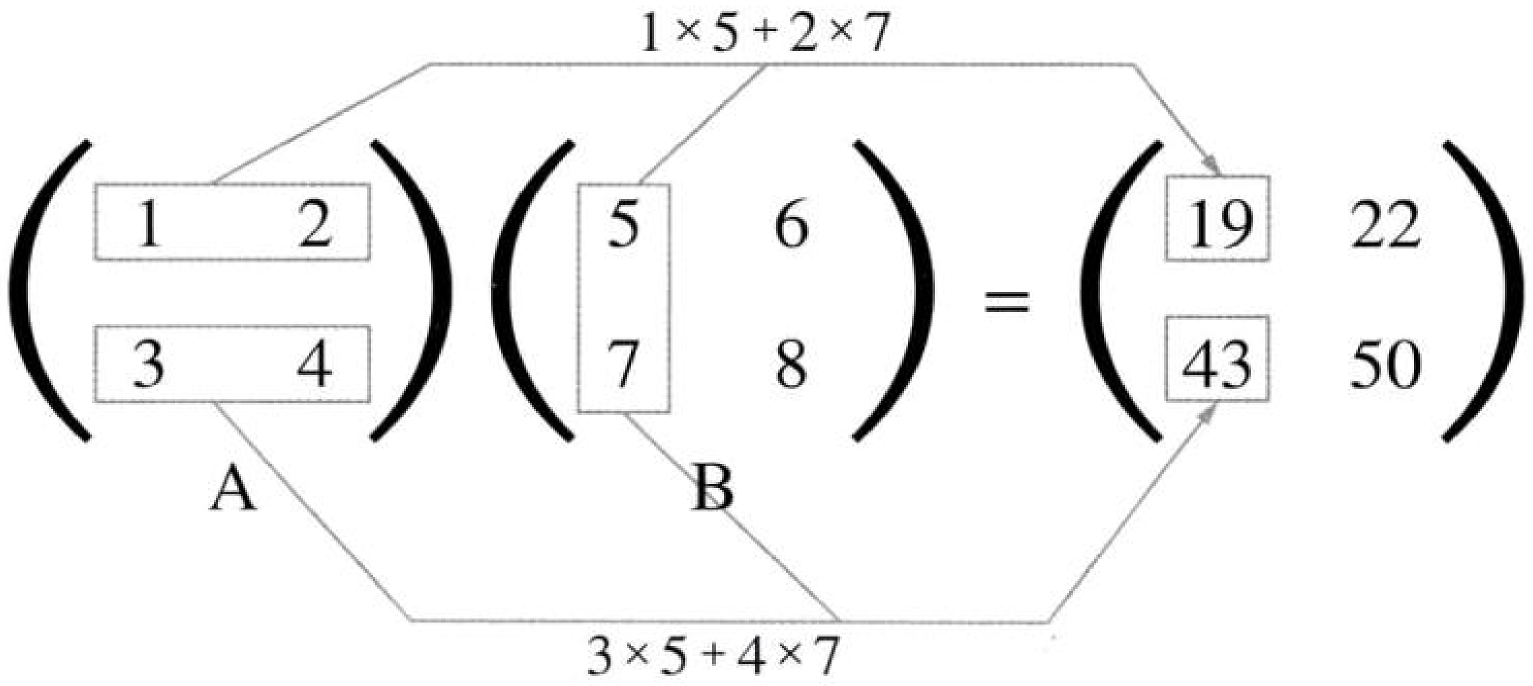
합
[1, 2]   [5, 6] -> [1+5, 2+6]
[3, 4]   [7, 8]   [3+7, 4+8]

상수배
2(1 2 -> 2 4
3 4)   6 8

행렬의 곱

행렬의 곱

그림 3-11 행렬의 곱 계산 방법



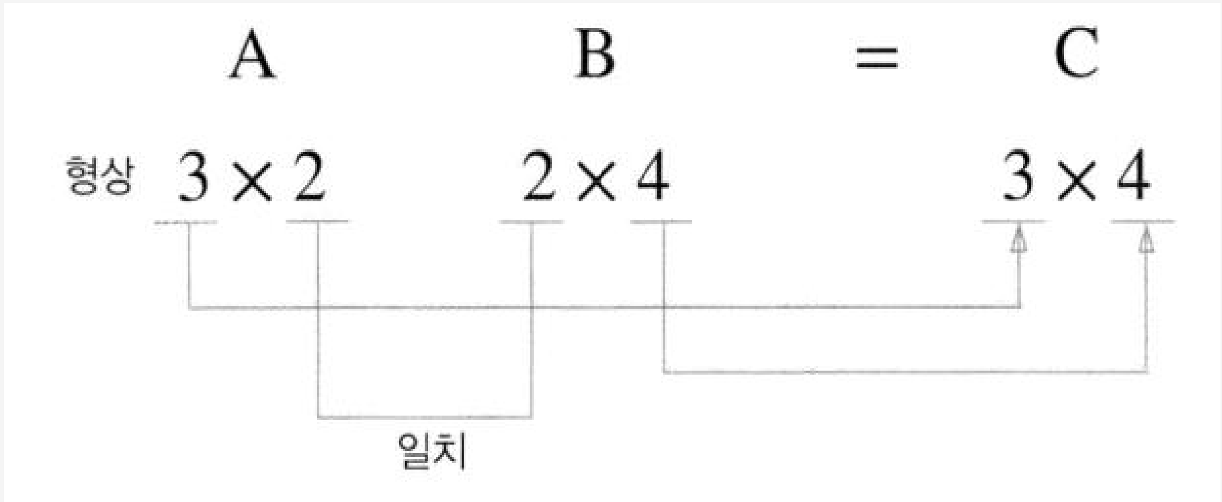
2 * 2 2 * 2 -> 2 * 2

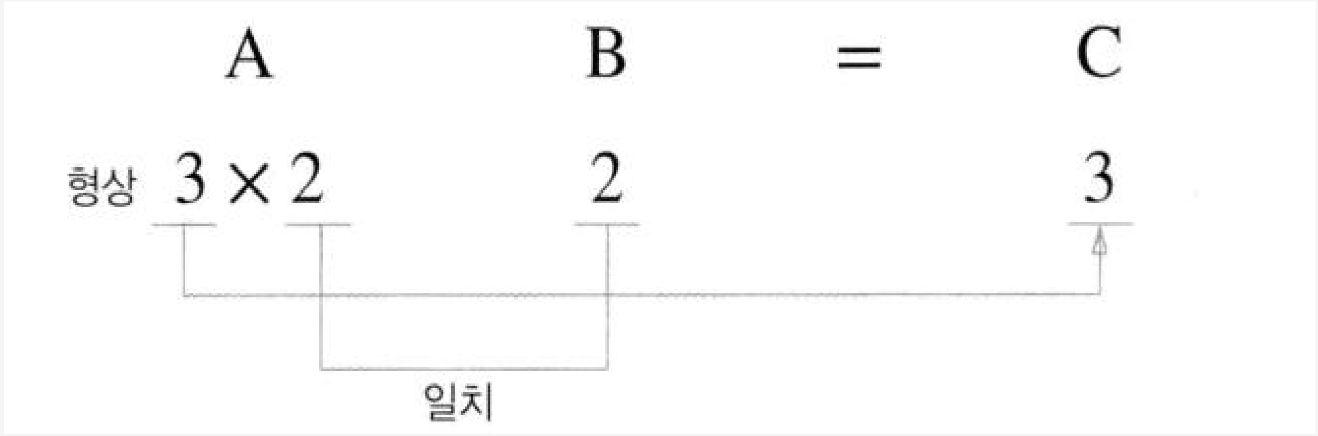
A행렬 행 * B행렬 열 방식

(앞 행렬의 행의 수)
×
(뒤 행렬의 열의 수)

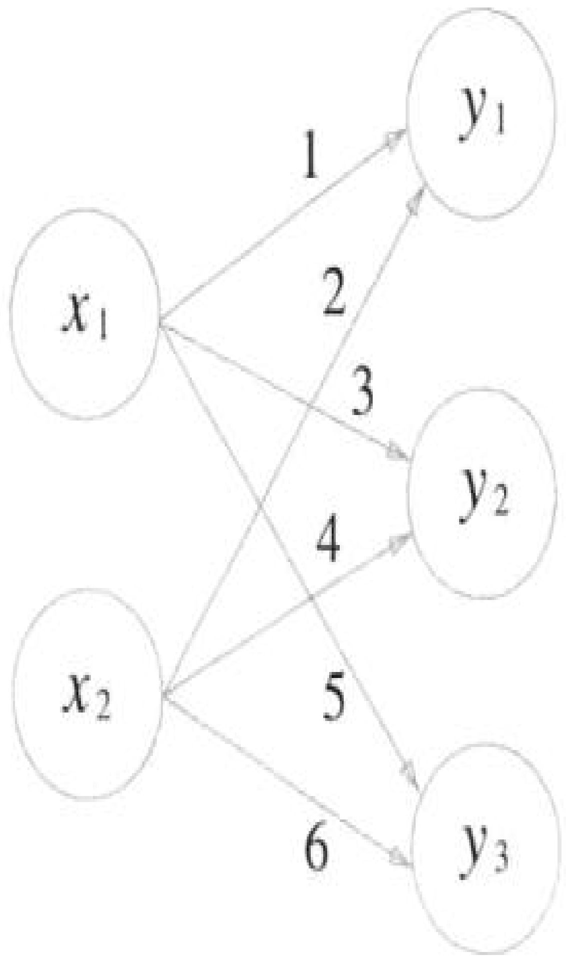
단, 교환법칙은 성립하지 않는다.

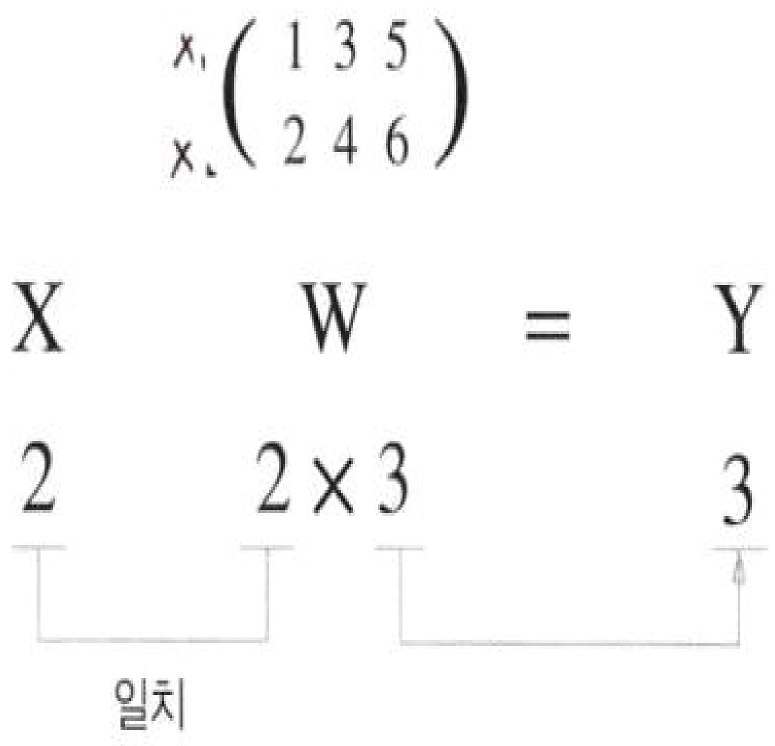
AB =! BA





신경망에서 행렬





cf. w행렬이 2 * 3이 나올 수 밖에 없는 이유

입력하는 뉴런의 수 2개
출력 받는 뉴런의 수 3개

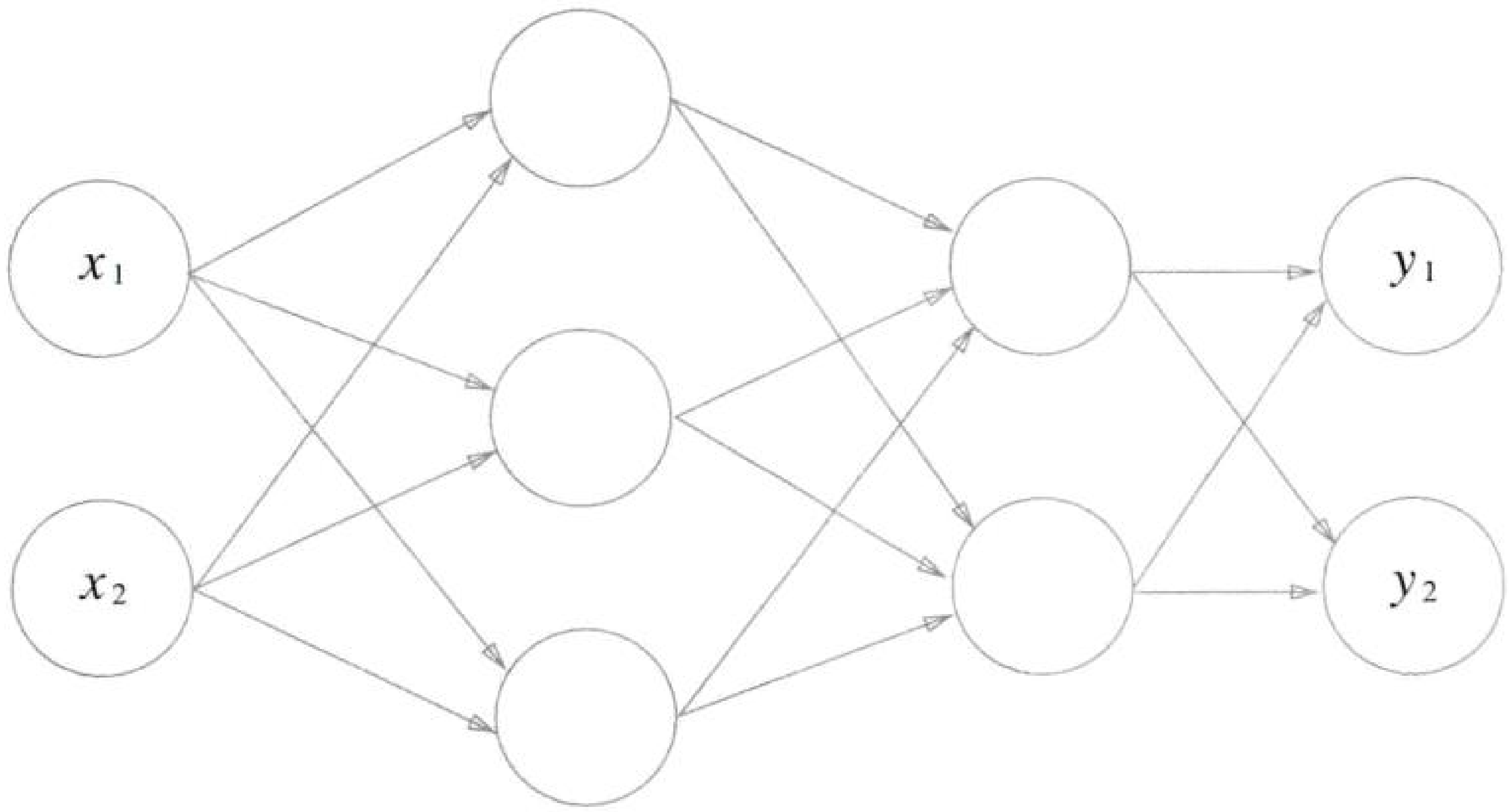
x1, x2라는 뉴런에서 각각 신호가 3개가 나오니,
2행이고 각각 발생하는 가중치가 x1 3가지 x2 3가지 이니
2 * 3 행렬이 될 수밖에 없다

X 0차원 : 원소 2개

W 2차원 : 2 * 3
행렬

3층 신경망?

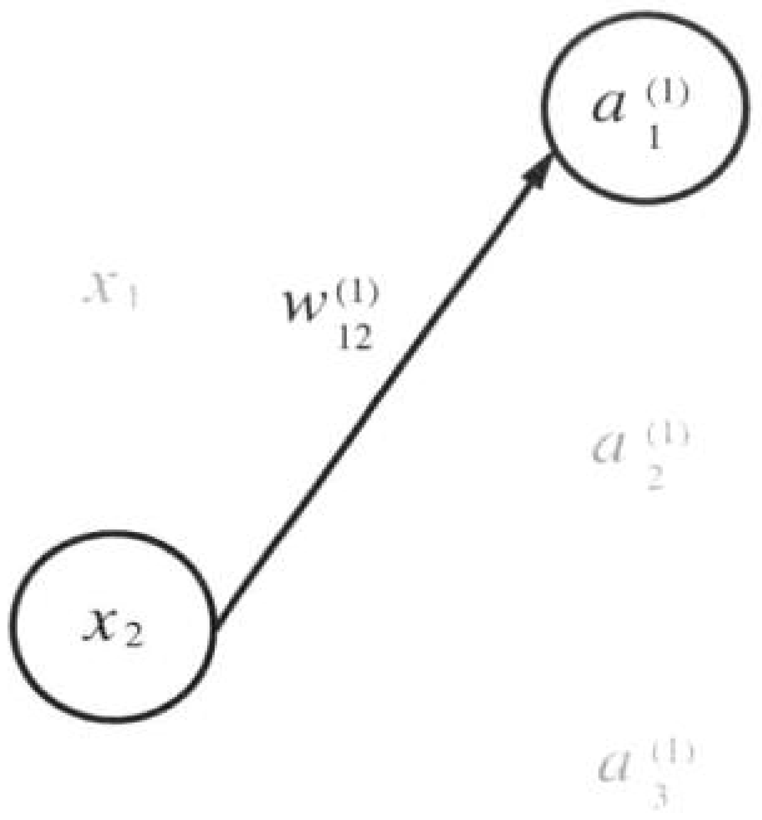
그림 3-15 3층 신경망 : 입력층(0층)은 2개, 첫 번째 은닉층(1층)은 3개, 두 번째 은닉층(2층)은 2개, 출력층(3층)은 2개의 뉴런으로 구성된다.

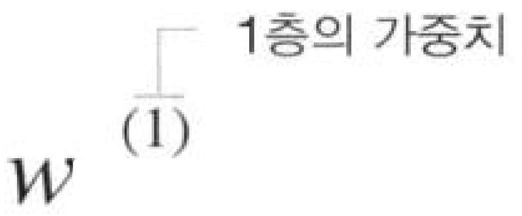


입출력 층에
은닉층 2개

표 기 법 설 명

그림 3-16 중요한 표기



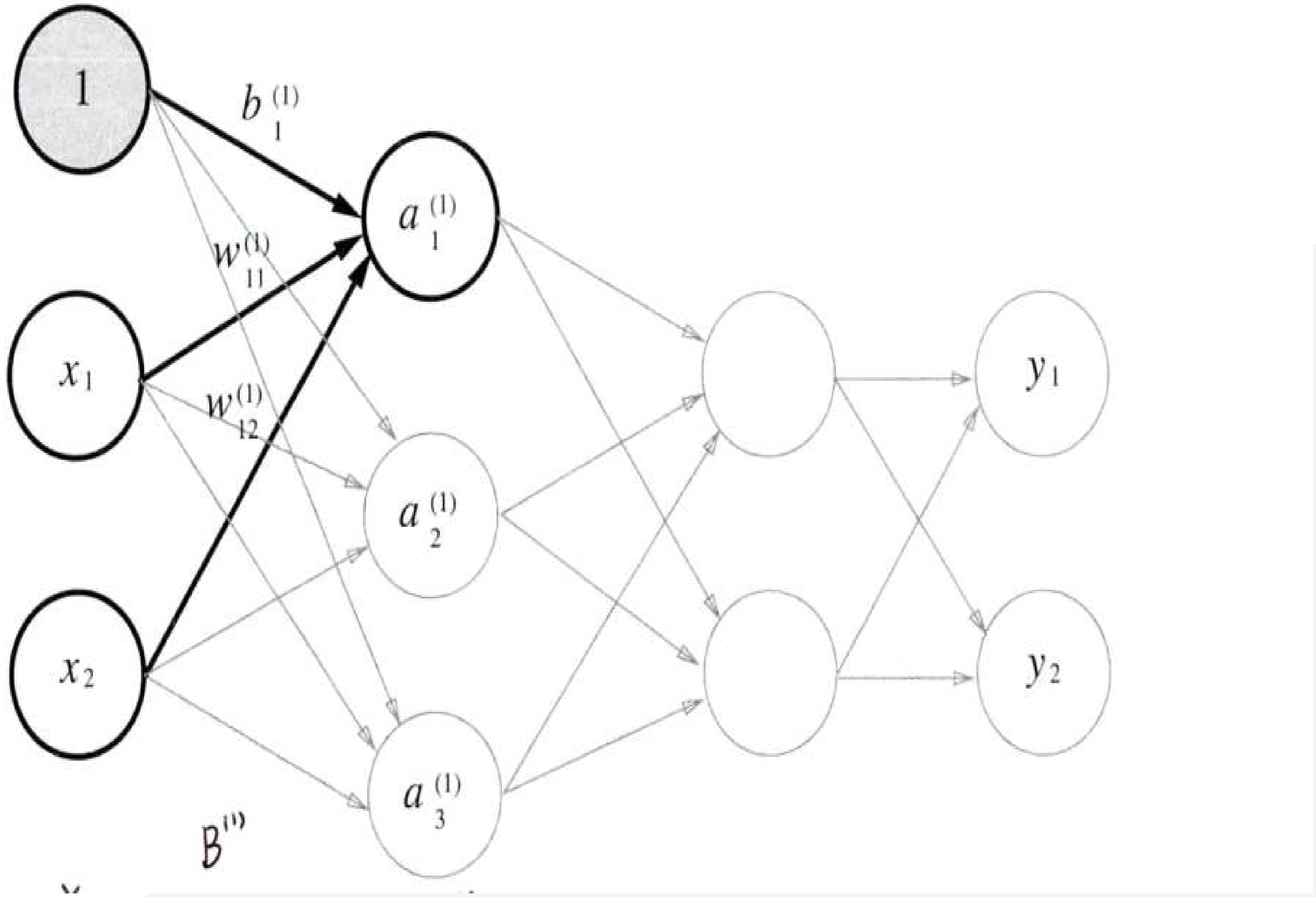


2 1

2 -> 앞 층의 2번째 뉴런

1 -> 다음 층의 1번째 뉴런

1층 신호 전달 과정



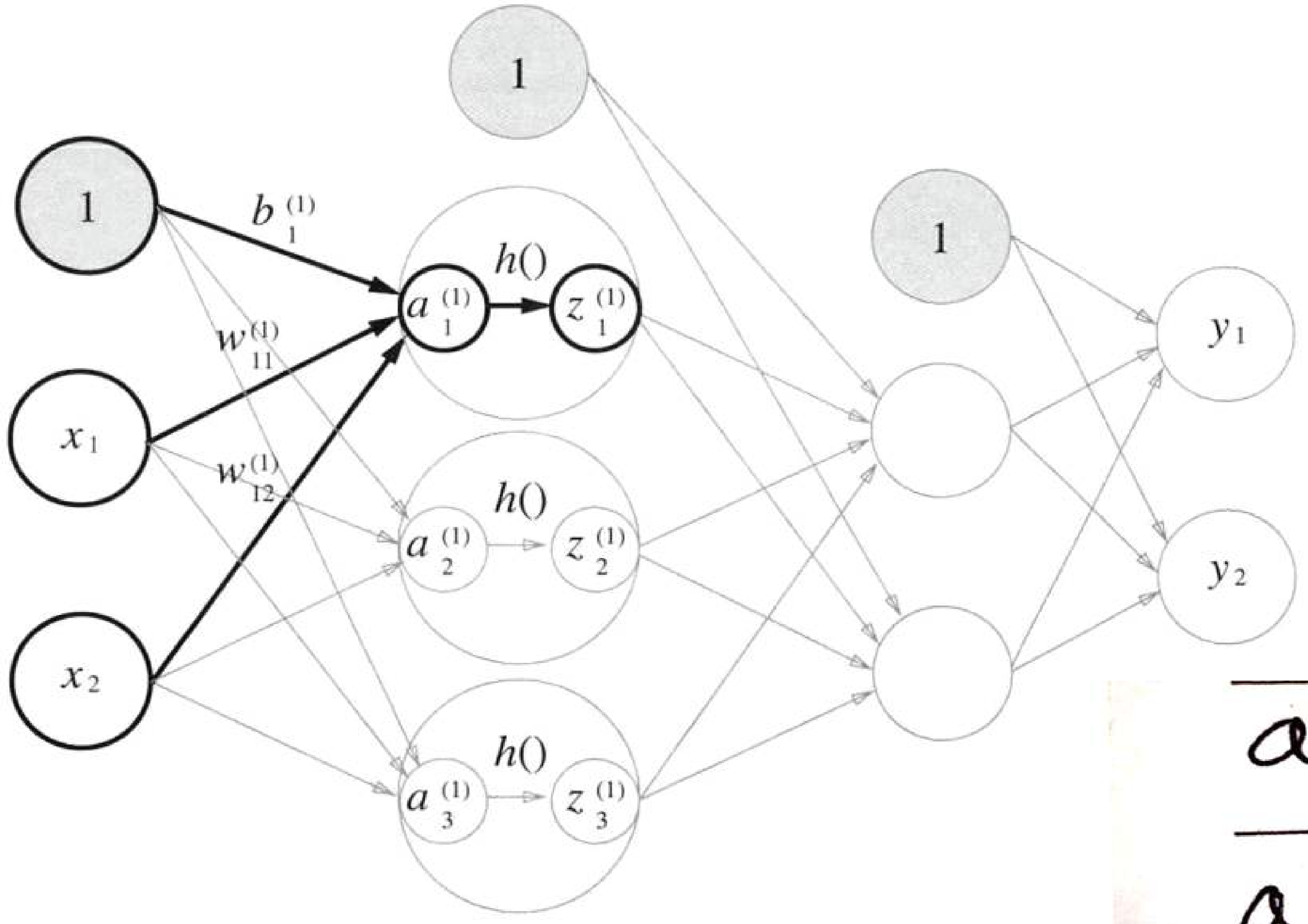
수정: 헷갈림 수정함

$$\mathbf{A}^{(1)} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{B}^{(1)}$$

행렬의 곱 이용

1층 신호 전달 과정

그림 3-18 입력층에서 1층으로의 신호 전달



수정

$W^{(층)}$

입력뉴런, 출력뉴런

ex) $W^{(1)}_{(12)}$ → 1층 1번 노드 → 2번 노드

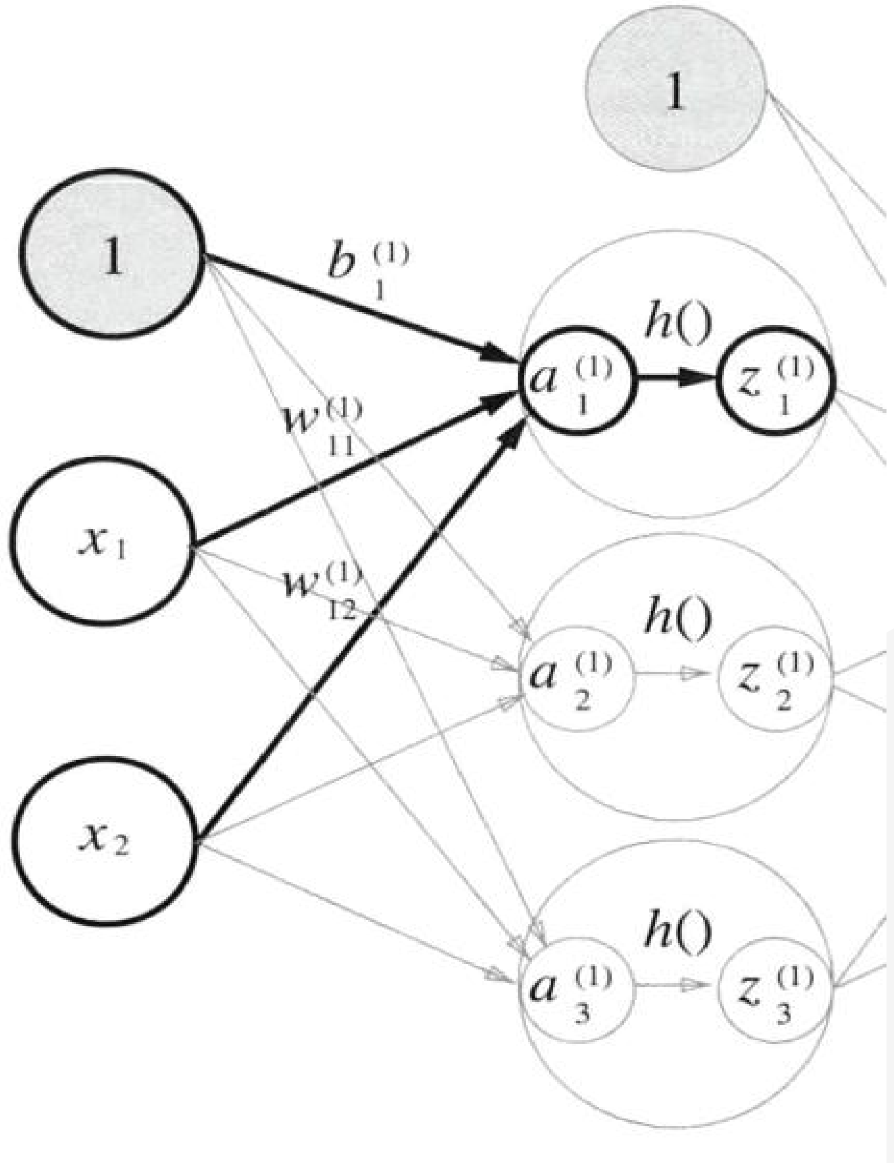
$$a_1^{(k)} = w_{11}^{(k)} \times x_1 + w_{21}^{(k)} \times x_2 + b_1^{(k)}$$

$$a_2^{(k)} = w_{12}^{(k)} \times x_1 + w_{22} \times x_2 + b_2^{(k)}$$

$$a_3^{(k)} = w_{13} \times x_1 + w_{23} \times x_2 + b_3^{(k)}$$

1층 신호 전달 과정

그림 3-18 입력층에서 1층으로의 신호 전달



행렬의 곱으로 표현 가능

$$\begin{pmatrix} a_1^{(k)} & a_2^{(k)} & a_3^{(k)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} W_{11}^{(k)} & W_{12}^{(k)} & W_{13}^{(k)} \\ W_{21}^{(k)} & W_{22}^{(k)} & W_{23}^{(k)} \end{pmatrix} + \begin{pmatrix} b_1^{(k)} & b_2^{(k)} & b_3^{(k)} \end{pmatrix}$$

2 * 2*3 -> 3

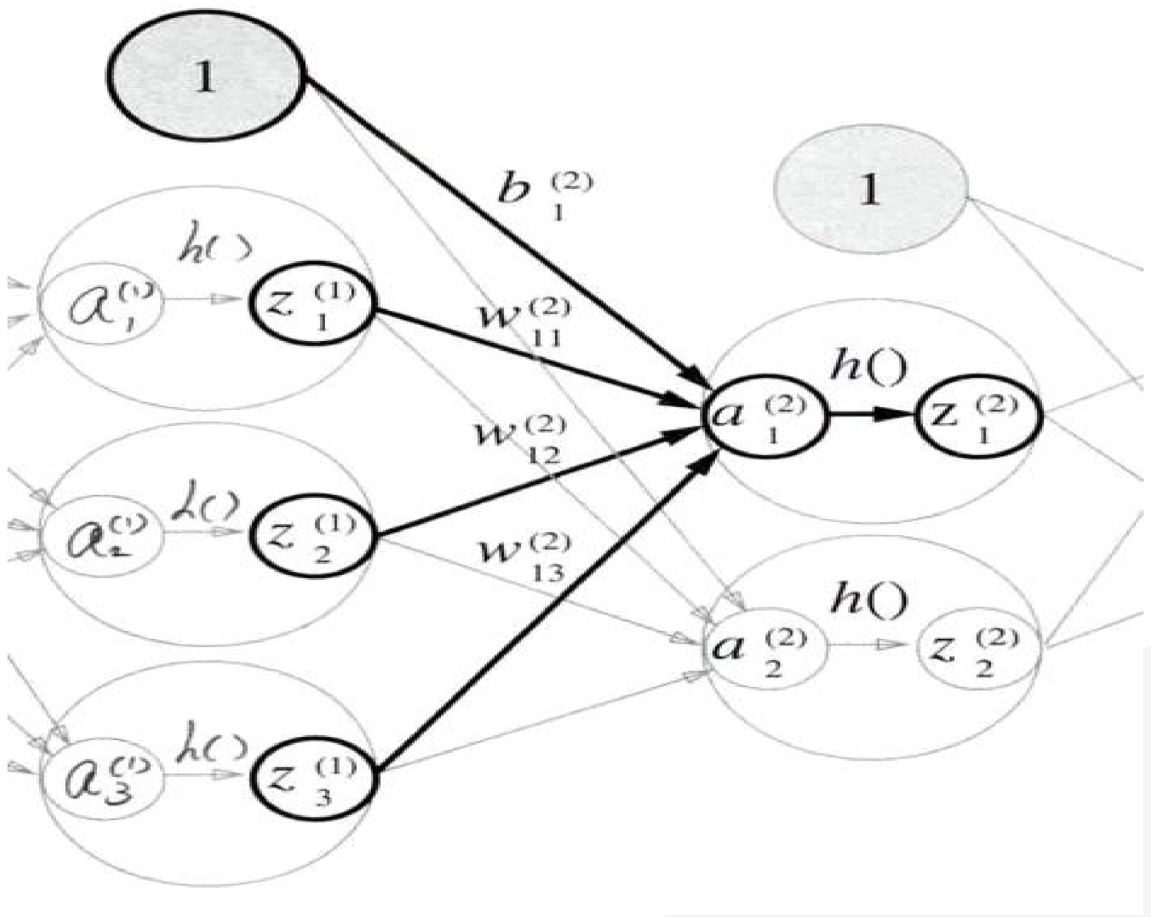
3 + 3행렬 -> 3(각각 원소 마다 덧셈)

tip !
그림 이해 후 행렬 보면 쉽다

행이 두개인 이유 : x1, x2에서 나오는 종류가 2가지
열이 3가지인 이유: x1, x2에서 나오는 가중치 각 3개

a1 = x1에서 나오는 가중치 한가지와 x2에서 나오는 가중치의 가중합 -> 빨간색 가중치 사용

2층 신호 전달 과정



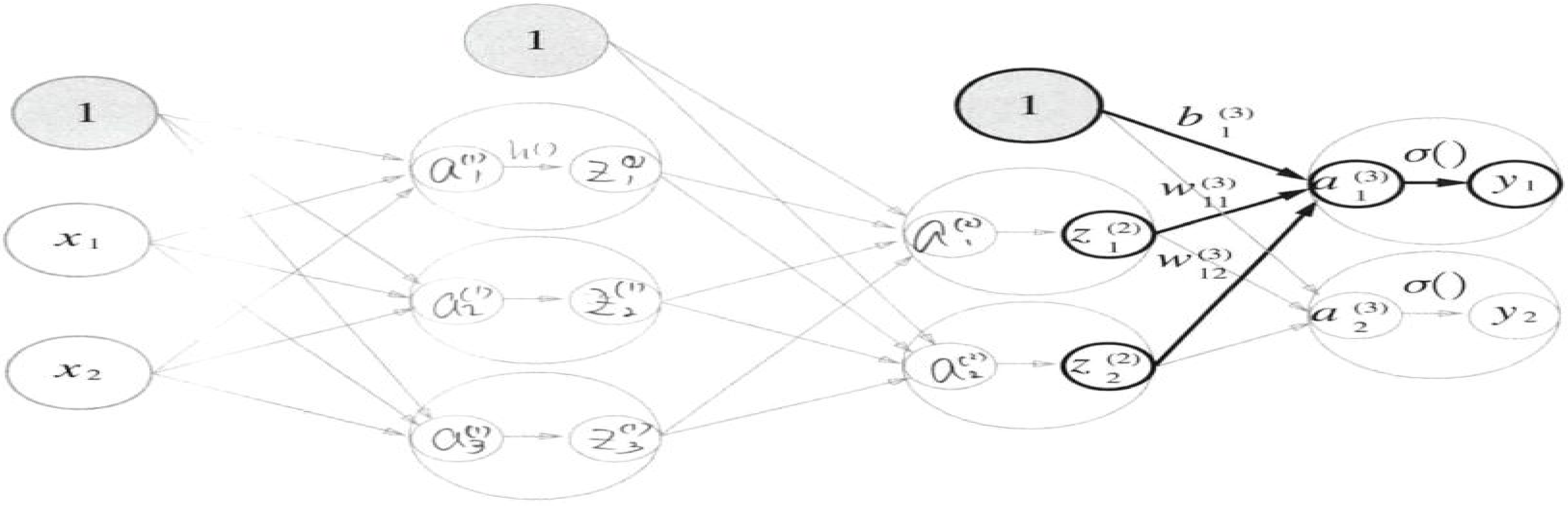
$$\begin{pmatrix} a_1^{(2)} & a_2^{(2)} \end{pmatrix} = \begin{pmatrix} z_1^{(1)} & z_2^{(1)} & z_3^{(1)} \end{pmatrix} \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \\ w_{31}^{(2)} & w_{32}^{(2)} \end{pmatrix} + \begin{pmatrix} b_1^{(2)} & b_2^{(2)} \end{pmatrix}$$

tip !
그림 이해 후 행렬 보면 쉽다

1층의 출력이 입력이 됨(z)
행이 3개인 이유 : z1, z2, z3에서 나오는 종류가 3가지(입력 뉴런의 개수)
열이 2가지인 이유: z1, z2, z3에서 나오는 가중치 각 3개(다음 뉴런의 개수)

3층 신호 전달 과정

그림 3-20 2층에서 출력층으로의 신호 전달



$$\left(a_1^{(3)} \quad a_2^{(3)}\right) = \left(z_1^{(2)} \quad z_2^{(2)}\right)\begin{pmatrix} w_{21}^{(3)} & w_{22}^{(3)} \\ w_{31}^{(3)} & w_{32}^{(3)} \end{pmatrix} + \left(b_1^{(3)} \quad b_2^{(3)}\right)$$

출력층 설계

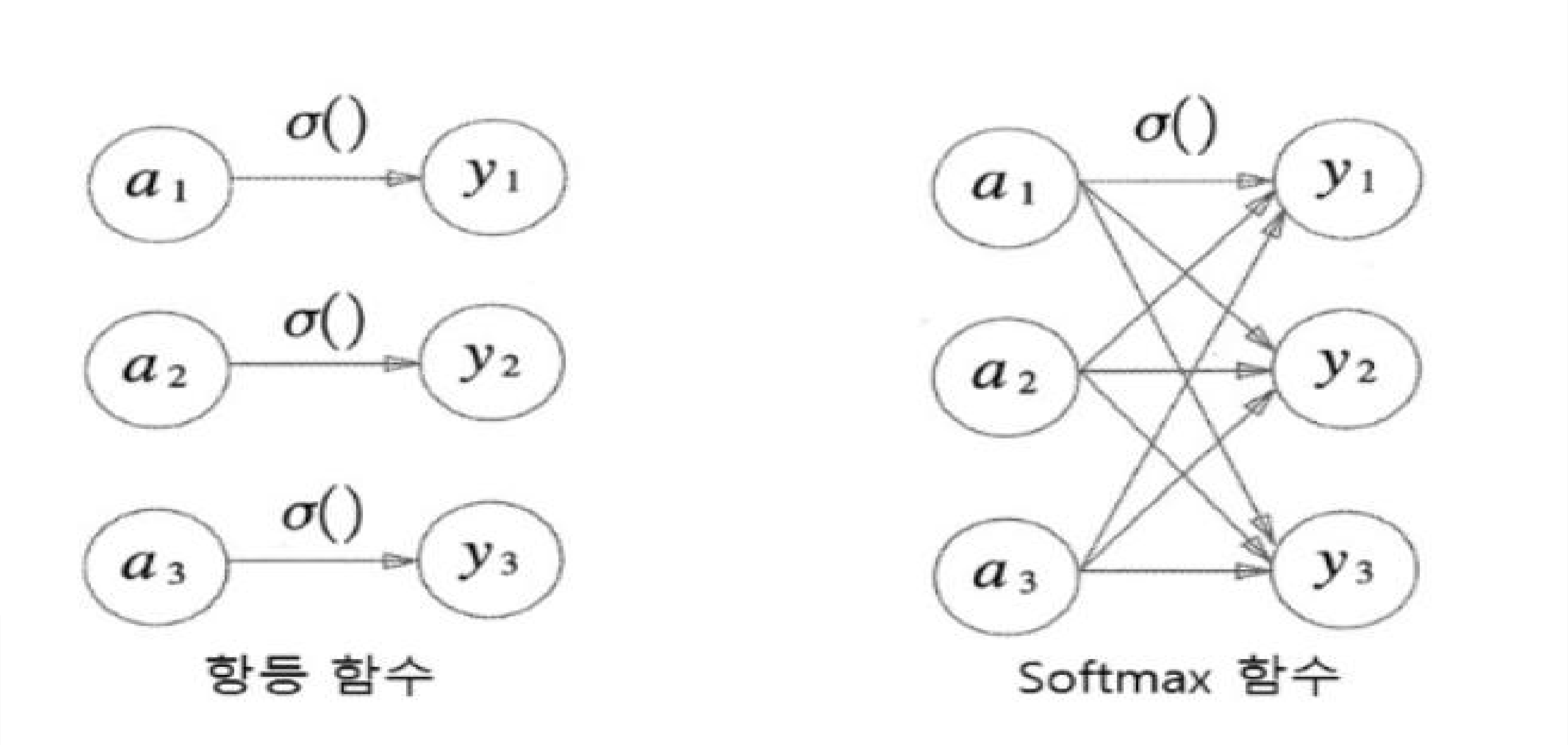
확률 벡터

소프트 맥스

소프트 맥스 특징

확률 벡터

출력층에서 사용하는 함수



회귀

분류

입력층에서는 시그모이드 출력층에선 소프트맥스
-> 확률 값으로 변환하기 위해
소프트 맥스: (일반 벡터) -> (확률 벡터)로 변환시켜줌

확률 벡터?

확률 변수에 대해 일어날 수 있는 사건의 확률을 벡터로 표현

ex) 동전 (1/2, 1/2)

특징

1. 원소는 모두 0 이상
2. 총 합이 1이다

소프트 맥스

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

일반 함수

1. exp(x) -> 지수 함수 -> 모두 다 0이상으로 변환

2. normalize(스케일링) -> 합을 1로 만드려고(확률값으로) (표준화 <-> 정규화)
 (X - MIN) / (MAX-MIN)

머신러닝 스케일링 방식

$$x' = \frac{x - x_{min}}{x_{max} - x_{min}}$$

정규화

$$z = \frac{x - \mu}{\sigma}$$

표준화

피처의 영향력이 극대화 되는 것 방지

소프트 맥스의 특징

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

overflow

python으로 실행 시 overflow문제를 발생할 수 있음.
지수함수의 특성

값이 커지면 정확한 수치를 내놓기 보다 inf로 반환

```
>>> a = np.array([1010, 1000, 990])
>>> np.exp(a) / np.sum(np.exp(a)) # 소프트맥스 함수의 계산
array([ nan,  nan,  nan])          # 제대로 계산되지 않는다.
>>>
```

해결 방식

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)} = \frac{C\exp(a_k)}{C\sum_{i=1}^{n} \exp(a_i)}$$
$$= \frac{\exp(a_k + \log C)}{\sum_{i=1}^{n} \exp(a_i + \log C)}$$
$$= \frac{\exp(a_k + C')}{\sum_{i=1}^{n} \exp(a_i + C')}$$

```
>>> c = np.max(a)              # c = 1010 (최댓값)
>>> a - c
array([  0, -10, -20])
>>>
```

지수 함수안 인자에 c값을 뺌으로 해결가능-> 큰값을 작게할 수 있다. why?

소프트 맥스 수학적 증명

지수 함수의 특성 이용

① $(a_1, a_2, a_3) \to (a_1+c, a_2+c, a_3+c)$

동일

e^x

$e^x \to (e^{a_1+c}, e^{a_2+c}, e^{a_3+c})$

nomalize

$$\to \left(\frac{e^{a_1+c}}{e^{a_1+c}+e^{a_2+c}+e^{a_3+c}}, \frac{e^{a_2+c}}{e^{a_1+c}+e^{a_2+c}+e^{a_3+c}}, \dots \right)$$

$$\to \left(\frac{\cancel{e^{c}} \times e^{a_1}}{\cancel{e^{c}}(e^{a_1}+e^{a_2}+e^{a_3})}, \frac{\cancel{e^{c}} \times e^{a_2}}{\cancel{e^{c}} \times (e^{a_1}+e^{a_2}+e^{a_3})} \dots \right)$$

결국은 동일한 값 출력 할 수 있음

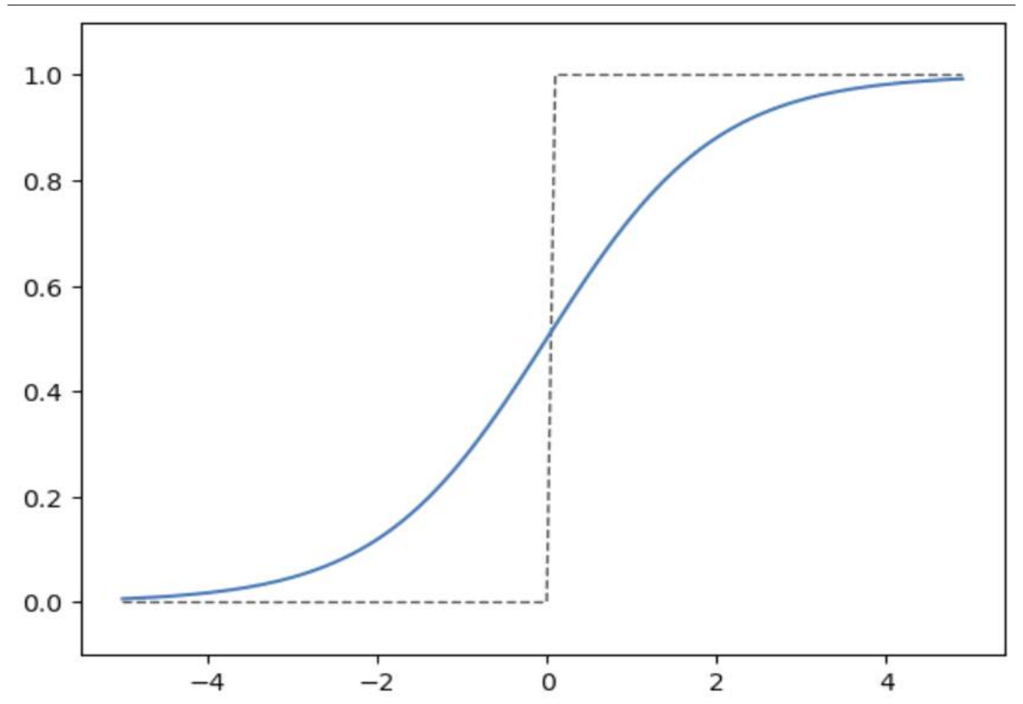
-> 0~1사이 값으로 확률값을 반환시켜준다

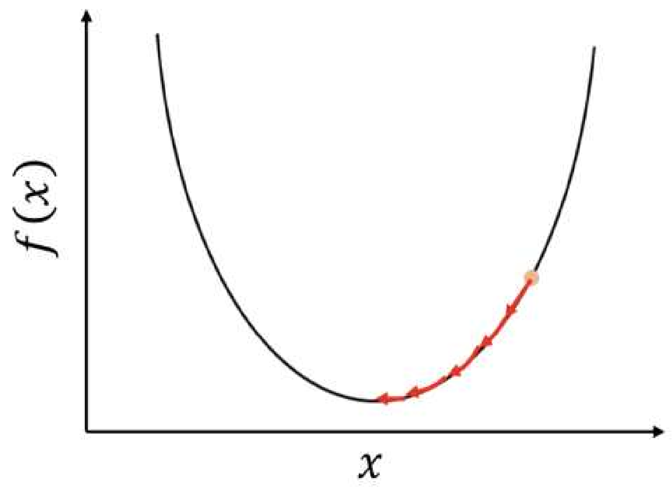
마 무 리

Q & A

마 무 리 하 며

Q & A





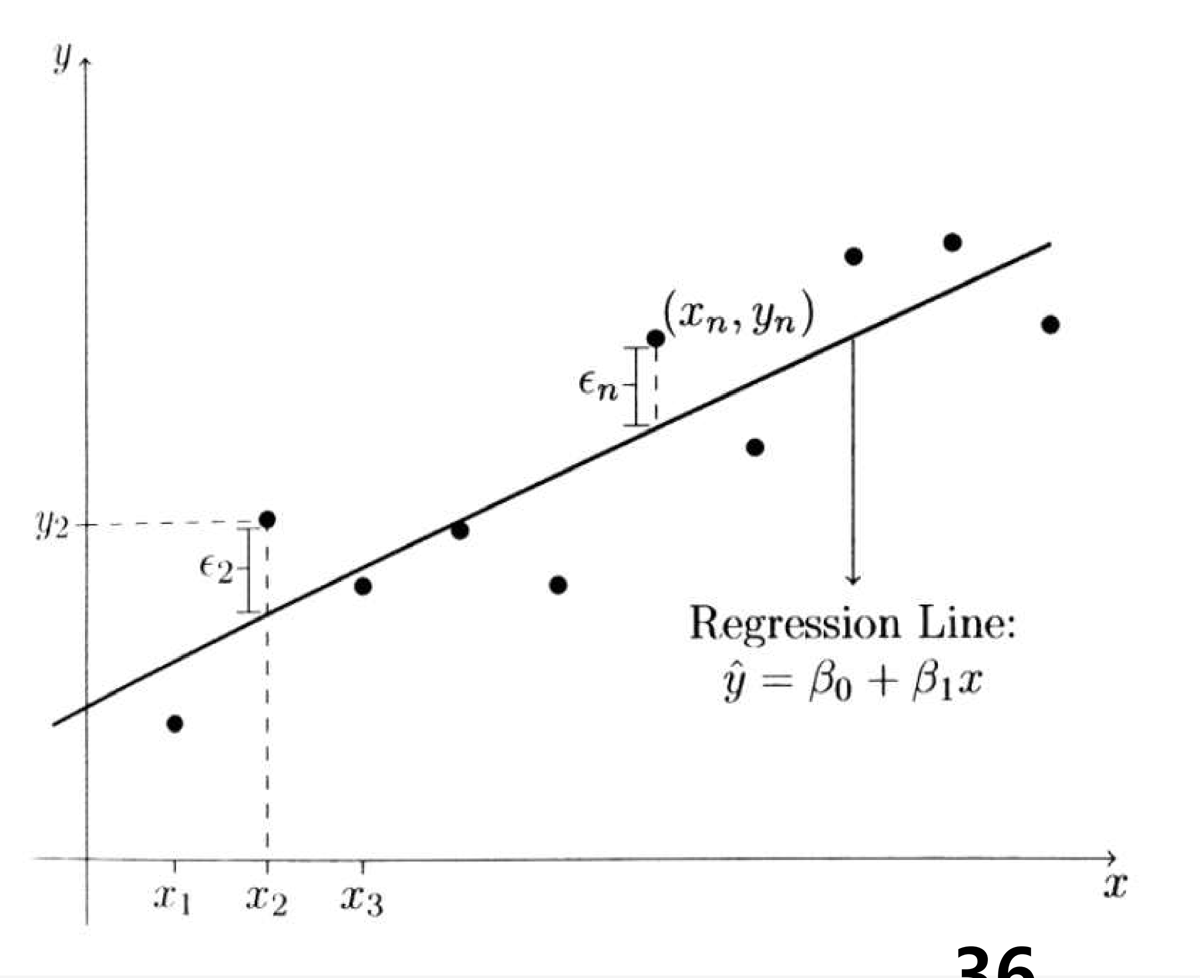
질문

1. 학습할때 미분을 통해 학습
-> 경사하강법? -> 오차를 줄이는 방향으로 학습 그렇다면 학습할때 사용하는 분포표는 오차의 분포 아님? 그렇게 되면 계단함수도 미분가능한 거 아닌가?

2. 그리고 출력층에서 왜 소프트 맥스를 사용해야 하는가
이미 시그모이드 함수를 통해 0~1값사이로 반환되는 데 입력값의 가중합이 활성화 함수를 만나서(이미 양수이고 정규화 되어있는데?)

3. 행렬의 이해
3-1. T했을 때 이해가 잘 안됨 어떤식으로 변환되는지
3-2. 행렬은 열벡터?로 작성한다고 하는데 무슨 의미인지

4. 소프트 맥스 overflow가 발생할 일이 있나?
차피 시그모이드로 0~1값으로 만들어주는데 큰 값이 들어올 일이 어떻게 생김?



마 무 리 하 며

질문 외부
1. 학점.. -> 대학원?
2. 우리과 프로그래밍 다른 It분야?

깨달은 것
1. 행렬의 이해, 행렬을 왜 사용하는지(계산 간편)
2. 여러 신경망에서 활성화 함수를 왜쓰는지 이해
3. 수학적 개념이 대거 등장. 나올때마다 개념을 잡는 방식으로 (구글링) 공부

앞으로 공부해보고 싶은 것
1. 데이터를 보고 판단해서 분류 모델링을 만들어보고 싶음
2. 나의 학습 방법이 맞는지..
3. 실력 증진의 팁 ! 정을이형만의 공부 방식은?