

신영

2.1 퍼셉트론이란?

Perceptron 동작 원리

2.2 퍼셉트론을 이용한 단순 논리 회로

AND 게이트 진리표

NAND 게이트 진리표

OR 게이트 진리표

전체 논리 회로 진리표

2.3 퍼셉트론의 한계

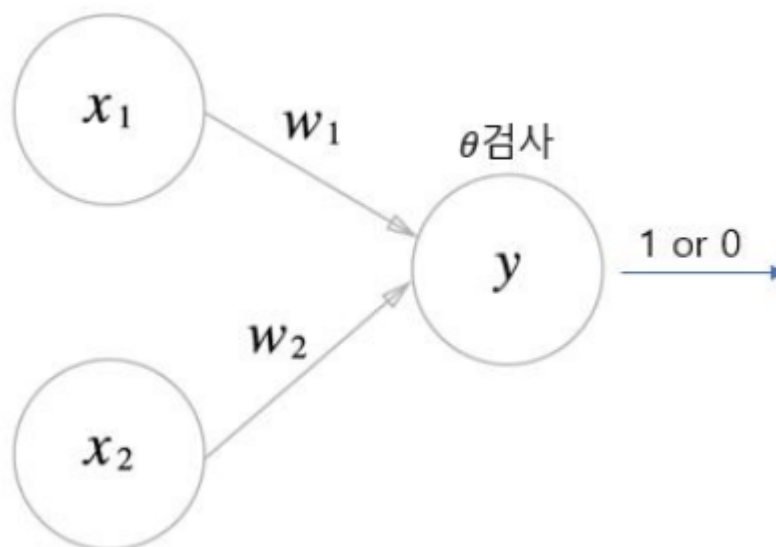
XOR 게이트 진리표

2.4 다층 퍼셉트론

2.1 퍼셉트론이란?

Perceptron

- 신경망(딥러닝)의 기원이 되는 알고리즘
- 다수의 신호를 입력으로 받아 하나의 신호를 출력



x1과 x2는 입력 신호

y는 출력 신호

w1과 w2는 가중치

| 그림의 원을 뉴런, 노드라고 부른다.

Perceptron 동작 원리

입력신호가 뉴런에 보내질 때는 각각 고유의 가중치가 곱해진다. ($w_1 \times 1 + w_2 \times 2$)

뉴런에서 보내온 신호의 총합이 **정해진 한계**를 넘어설 때만 1을 출력

그 한계는 **임계값**이라고 하며, θ 기호(theta; 세타)로 나타낸다.

$$\begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

- 가중치(w)란, 각 신호가 결과에 영향을 주는 정도를 의미
- 가중치가 클수록 그만큼 그 신호가 중요하다는 것을 알려줌

2.2 퍼셉트론을 이용한 단순 논리 회로

퍼셉트론을 사용하여 여러 논리회로를 구현 가능

AND, OR, NAND, XOR 게이트를 알아보고 파이썬으로 구현

$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases} \xrightarrow[\substack{-\theta = b \text{로 치환} \\ (b = \text{편향(bias)})}]{\theta \text{를 좌변으로 이동}} y = \begin{cases} 0 & (b + w_1x_1 + w_2x_2 \leq 0) \\ 1 & (b + w_1x_1 + w_2x_2 > 0) \end{cases}$$

| $w_1, w_2 \rightarrow$ 각 입력 신호가 결과에 주는 **영향력(중요도)**

| $b(\text{편향}) \rightarrow$ 뉴런이 **얼마나 쉽게 활성화** 하느냐를 조정하는 매개변수

AND 게이트 진리표

| 모두 참일 때만 참을 반환

x_1	x_2	y

0	0	0
1	0	0
0	1	0
1	1	1

AND gate를 퍼셉트론으로 표현해줄

- w_1, w_2, θ 의 값 정해야 함.
- ex) $x_1 = 0.5, x_2 = 0.5, b = -0.8$
- 이밖에도 진리 표를 만족하는 가중치의 조합은 다양하다.
ex) (0.5, 0.5, 0.7) (0.5, 0.5, 0.8) (1.0, 1.0, 1.0) 등등.

```
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([1, 1])
    b = -1.4
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

```
for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
    y = AND(xs[0], xs[1])
    print(str(xs) + " -> " + str(y))
```

```
(0, 0) -> 0
(1, 0) -> 0
(0, 1) -> 0
(1, 1) -> 1
```

NAND게이트 진리표

NAND gate는 **Not AND**를 의미한다.

NAND는 두 입력이 모두 참일 때 거짓을 반환하고, 그 외에는 참을 반환

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

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

```
# numpy 사용여부
# scalar 연산, vector 연산, ma
def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])
    b = 0.7 # nand origin
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

```
for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
    y = NAND(xs[0], xs[1])
    print(str(xs) + " -> " + str(y))
```

```
(0, 0) -> 1
(1, 0) -> 1
(0, 1) -> 1
(1, 1) -> 0
```

OR 게이트 진리표

OR는 두 입력 중 하나라도 참이면 참을 반환

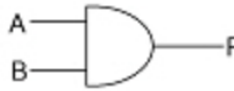

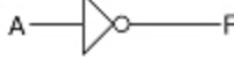




x ₁	x ₂	y
0	0	0
1	0	1
0	1	1
1	1	1

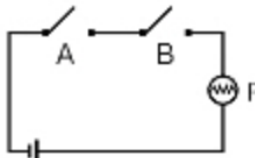
```
def OR(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.2
    tmp = np.sum(w*x) + b
    if tmp <= 0:
        return 0
    else:
        return 1
```

```
for xs in [(0, 0), (1, 0), (0, 1), (1, 1)]:
    y = OR(xs[0], xs[1])
    print(str(xs) + " -> " + str(y))
```

```
(0, 0) -> 0
(1, 0) -> 1
(0, 1) -> 1
(1, 1) -> 1
```

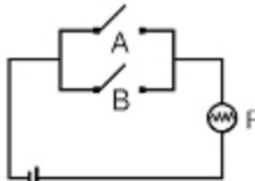
전체 논리 회로 진리표

게이트	논리기호	논리식	진리표															
AND		$F = A \cdot B$ $= AB$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	0	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = A + B$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	1
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
NOT (=Inverter)		$F = \overline{A}$	<table border="1"> <thead> <tr> <th>A</th> <th>F</th> </tr> </thead> <tbody> <tr><td>0</td><td>1</td></tr> <tr><td>1</td><td>0</td></tr> </tbody> </table>	A	F	0	1	1	0									
A	F																	
0	1																	
1	0																	
NAND (Not AND)		$F = \overline{AB}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	1	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR (Not OR)		$F = \overline{A + B}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	0
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
XOR (Exclusive OR)		$F = A \oplus B$ $= \overline{A}B + A\overline{B}$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	0	0	1	1	1	0	1	1	1	0
A	B	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
XNOR (Exclusive NOR)		$F = A \odot B$ $= \overline{A}B + AB$	<table border="1"> <thead> <tr> <th>A</th> <th>B</th> <th>F</th> </tr> </thead> <tbody> <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> </tbody> </table>	A	B	F	0	0	1	0	1	0	1	0	0	1	1	1
A	B	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																



AND Gate

A, B 단자 모두가 On(1)이어야 F에 신호 도달



OR Gate

A, B 단자 하나만 On(1)이어도 F에 신호 도달

2.3 퍼셉트론의 한계

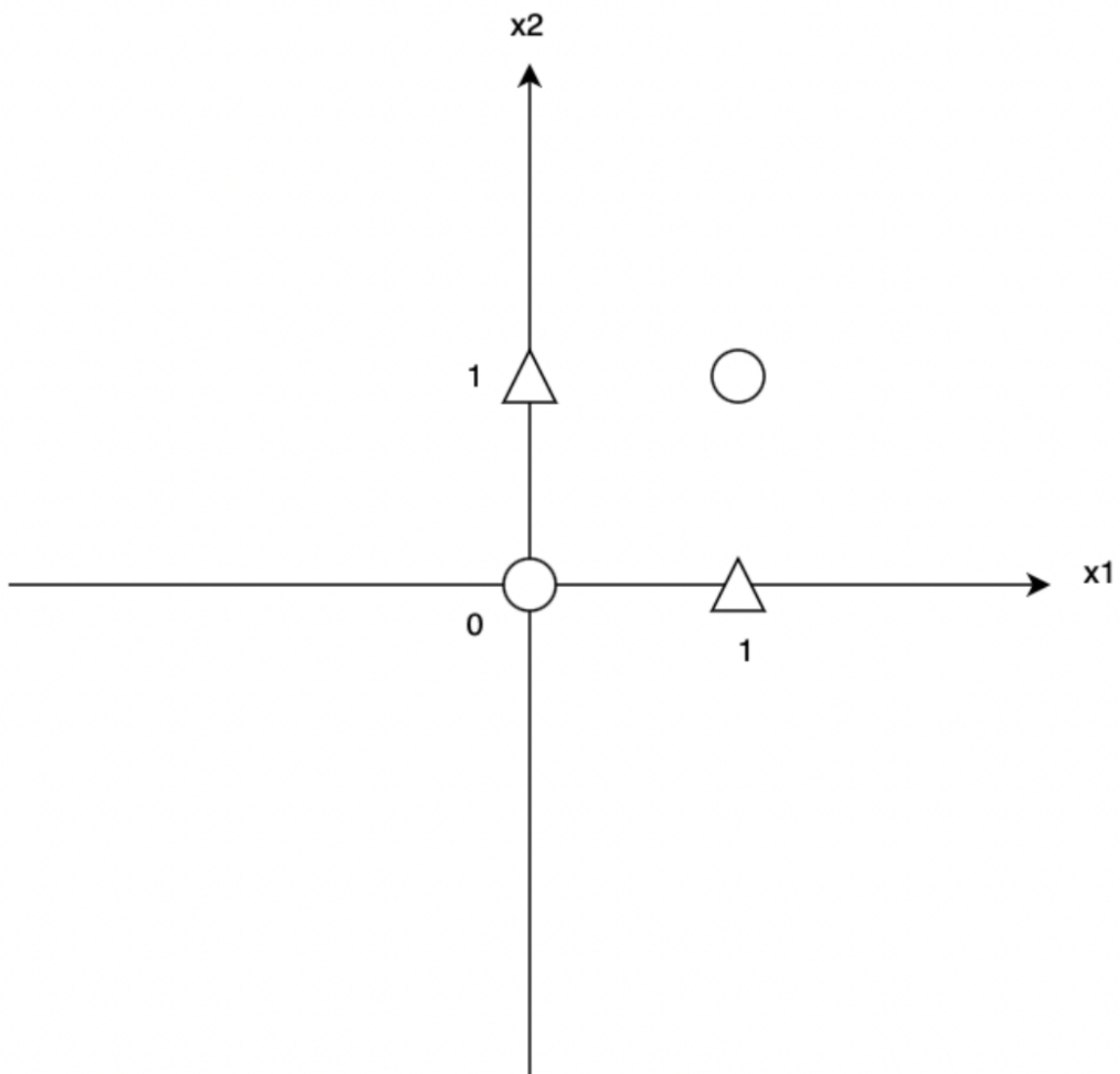
XOR 게이트 진리표

배타적 논리합, Exclusive OR

두 입력이 다를 때 참을 반환

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

이제까지의 방식으로는 위 진리표를 만족시키는 가중치를 정할 수 없다.
그 이유는 다음의 그림을 통해 알 수 있다.



위의 그림은 XOR 게이트의 출력을 나타낸 것이다.

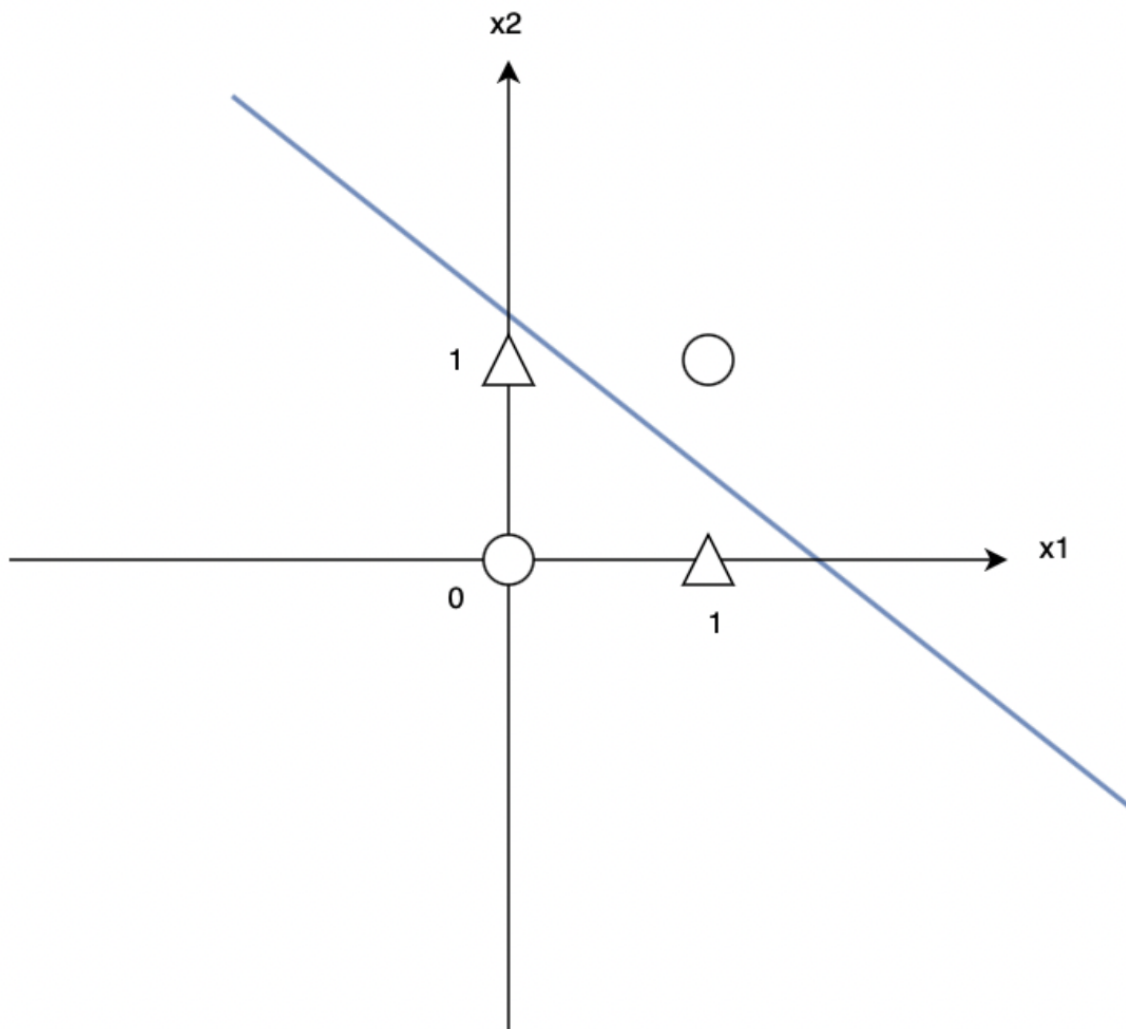
Perceptron은 기본적으로 수식의 형태가 **직선(선형)**이다.

$$\begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$

위의 식을 살펴보면 상수 w_1, w_2, θ 에 어떤 값을 대입했을 때,

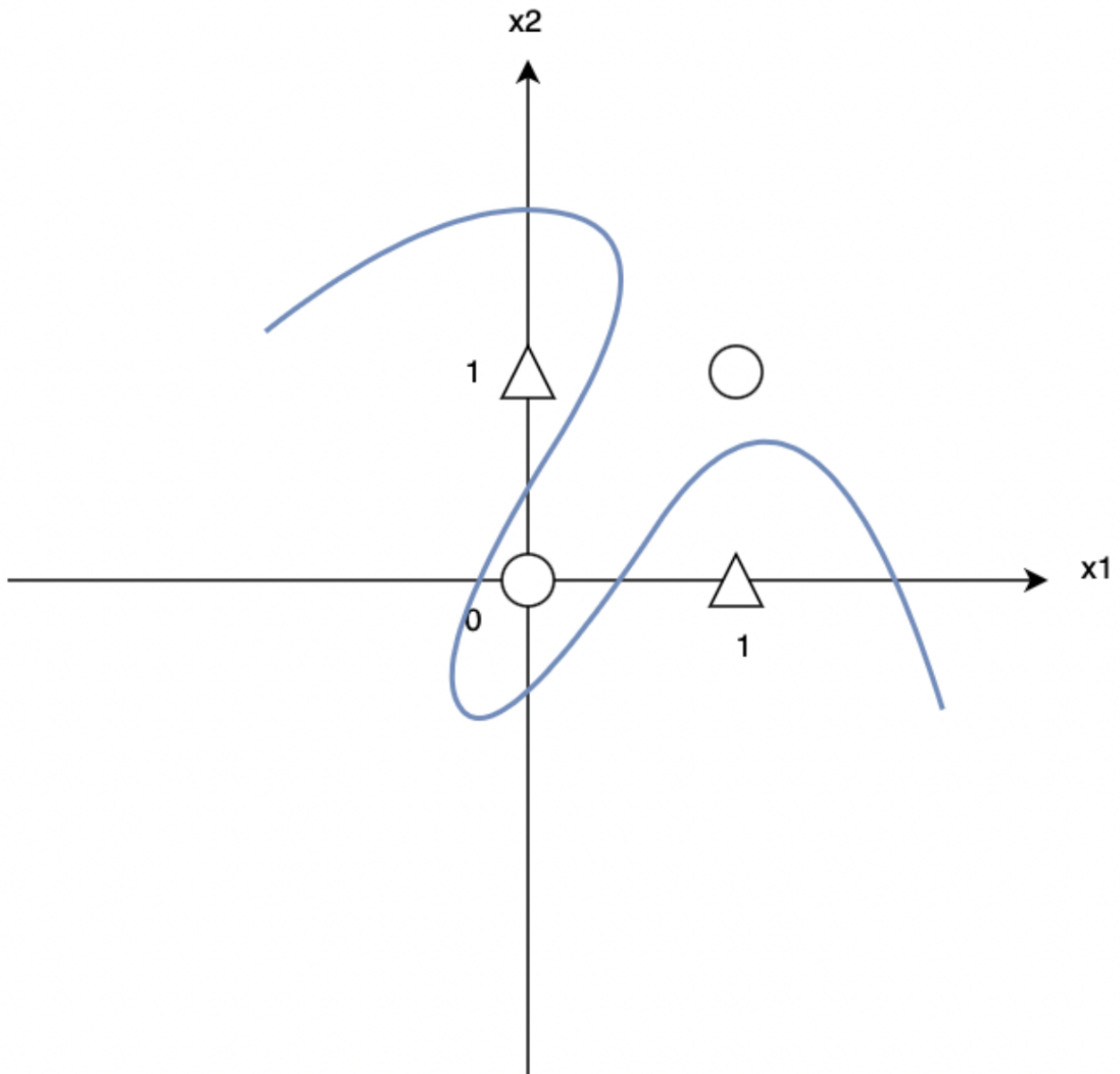
x_1 과 x_2 를 축으로 하는 좌표에서 **직선을 기준으로 영역이 나뉠**을 알 수 있다.

따라서 XOR gate를 퍼셉트론으로 표현하기 위해서는 위의 좌표에서의 결과 값을 직선으로 나눌 수 있어야 한다.



위의 그림에서 알 수 있듯이 ●과 ▲를 직선 하나로는 나누는 방법은 불가능하다.

하지만 '직선'이라는 제약을 없앤다면 가능하다.



위와 같은 곡선의 영역을 **비선형** 영역, 직선의 영역을 **선형** 영역이라고 한다.

단층 퍼셉트론으로 XOR을 구현할 수 없는 이유 정리



XOR 게이트의 진리표를 만족하는 퍼셉트론의 매개변수를 찾을 수가 없다.

$y = b + w_1x_1 + w_2x_2$ 이 식을 보면, 입력(x)와 출력(y)는

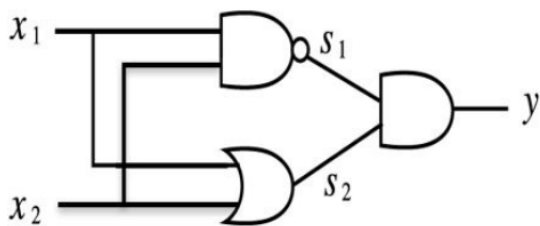
선형관계 이므로,

x가 커지면 y도 커져야 하는데, y는 감소할 수 없다.

따라서 단층 퍼셉트론으로는 XOR 게이트를 구현할 수 없으며, 다층 퍼셉트론으로 구현할 수 있다.

2.4 다층 퍼셉트론

- 단층 퍼셉트론을 쌓아 만든, 여러 층의 퍼셉트론
- 단층 퍼셉트론으로는 XOR게이트를 구현할 수 없었으나, 다층 퍼셉트론 사용시 XOR 구현 가능



x_1	x_2	s_1	s_2	y
0	0	1	0	0
1	0	1	1	1
0	1	1	1	1
1	1	0	1	0

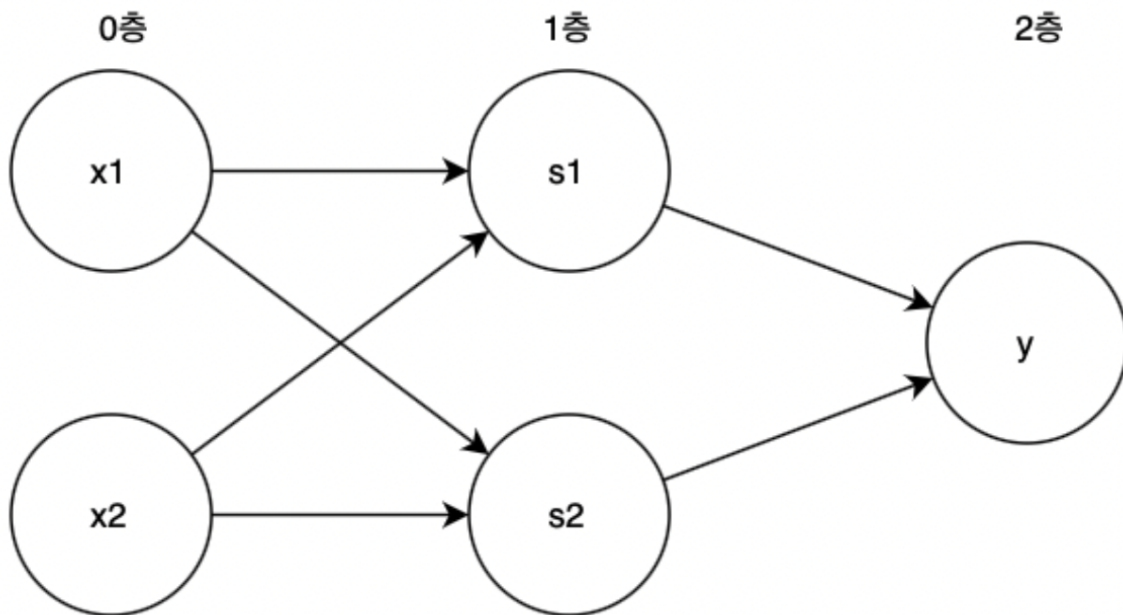
XOR gate는 퍼셉트론에 '층을 쌓은' 다층 퍼셉트론으로 구현이 가능하다.

기본적으로 XOR gate는 AND, NAND, OR gate의 선형표현을 조합하여 비선형 표현을 만들 수 있다.

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

```
y = AND(s1, s2)
return y
```

- 먼저 NAND로 두 입력의 부정된 AND 결과를 얻고
- 그 다음 OR로 두 입력 중 하나라도 참인지 확인
- 마지막으로 AND로 이 두 결과를 결합하여 최종적으로 XOR 결과를 얻음.



NAND와 OR은 독립적으로 수행되고, 마지막에 AND로 결합되기 때문에
둘 중 뭘 먼저해도 XOR은 구현이 됨.
다만 마지막에 AND로 결합하는 과정을 바꾸면 XOR이 제대로 구현되지 않음.

