Week 2. Perceptron & forward propagation

Kisoo Kim

Kwangwoon University, Information Convergence

Kisooofficial@naver.com

목차

- 퍼셉트론
  - Introduction
  - 적용 분야 – 논리 회로(AND, OR, NAND)
  - 한계점 – XOR 문제 풀기

- 신경망
  - 신경망의 구조 및 계산 방법
  - 여러 가지 활성화 함수
  - 인공 신경망에서 행렬을 이용하여 계산하는 방법

퍼셉트론(Perceptron)

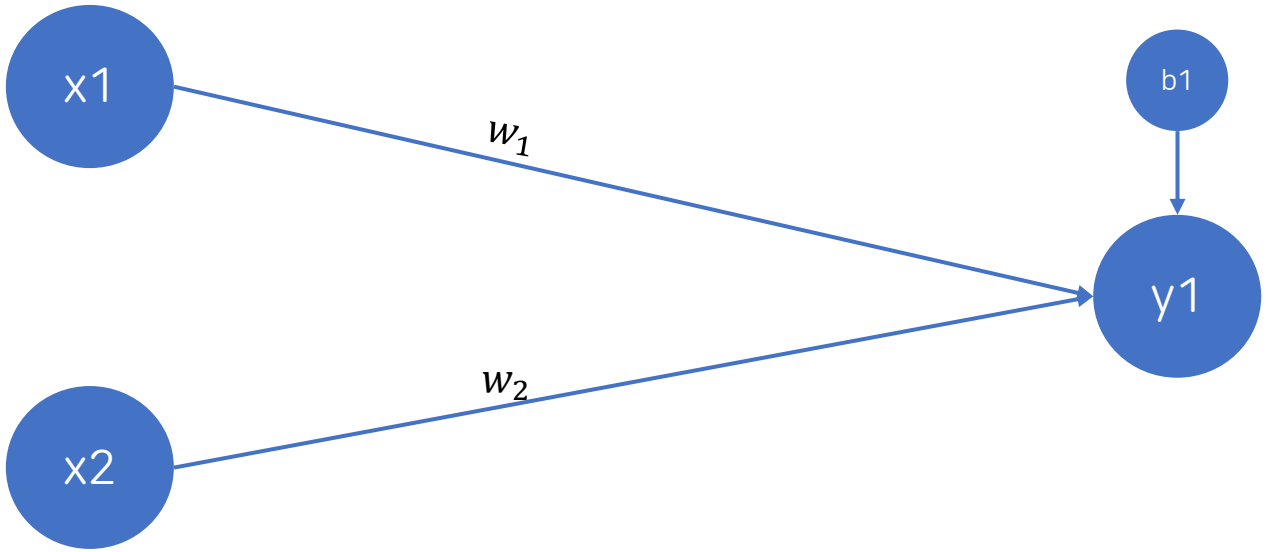
- 퍼셉트론
  - 다수의 신호를 입력으로 받아 하나의 신호를 출력
    - 신호 : 전류와 같은 흐름, 퍼셉트론에서는 흐른다/안 흐른다(1/0)으로 구분 가능
  - 입력이 2개인 퍼셉트론
    - x_1, x_2는 입력신호, w_1, w_2는 가중치 신호
    - 뉴런에 전해진 입력 신호의 총합이 정해진 한계를 넘어설 때만 1을 출력

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

퍼셉트론(Perceptron)

- 입력이 2개인 퍼셉트론
  - x_1, x_2는 입력신호, w_1, w_2는 가중치 신호
  - 뉴런에 전해진 입력 신호의 총합이 정해진 한계를 넘어설 때만 1을 출력

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



퍼셉트론(Perceptron)

- 논리 회로 – AND, NAND, OR, XOR
  - 진리표

입력(Input)		출력(Output)			
X1	X2	AND(X1, X2)	NAND(X1, X2)	OR(X1, X2)	XOR(X1, X2)
0	0	0	1	0	0
0	1	0	1	1	1
1	0	0	1	1	1
1	1	1	0	1	0

퍼셉트론(Perceptron)

- 논리 회로 – AND, NAND, OR
  - Python Implementation

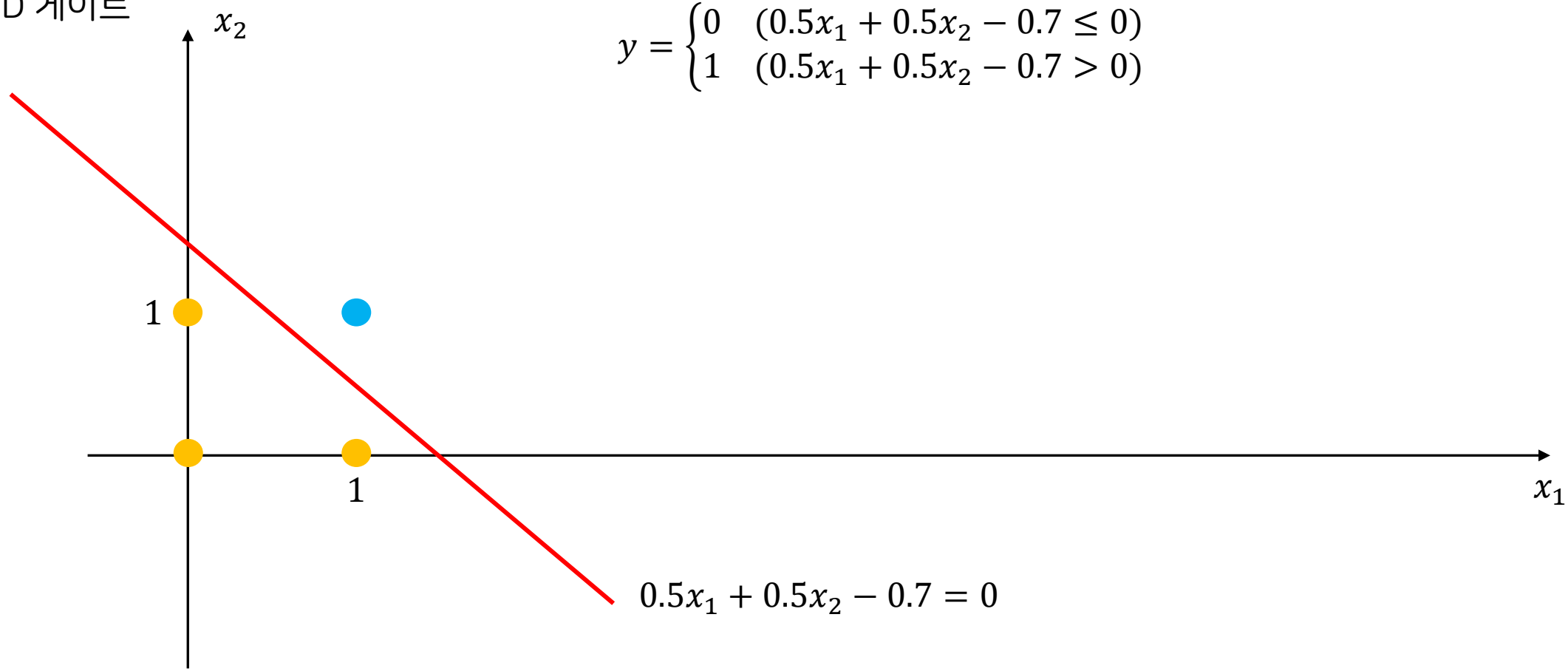
```python
# 2.3.3 가중치와 편향 구현하기
def AND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([0.5, 0.5])
    b = -0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1


def NAND(x1, x2):
    x = np.array([x1, x2])
    w = np.array([-0.5, -0.5])  # AND와는 가중치(w, b)만 다르다
    b = 0.7
    tmp = np.sum(w * x) + b
    if tmp <= 0:
        return 0
    else:
        return 1


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


print("AND")
print(AND(0, 0))  # 0
print(AND(0, 1))  # 0
print(AND(1, 0))  # 0
print(AND(1, 1))  # 1

print("NAND")
print(NAND(0, 0))  # 1
print(NAND(0, 1))  # 1
print(NAND(1, 0))  # 1
print(NAND(1, 1))  # 0

print("OR")
print(OR(0, 0))  # 0
print(OR(0, 1))  # 1
print(OR(1, 0))  # 1
print(OR(1, 1))  # 1
```

[1] 밑바닥부터 시작하는 딥러닝 github

퍼셉트론(Perceptron)

- 퍼셉트론의 시각화
  - AND 게이트

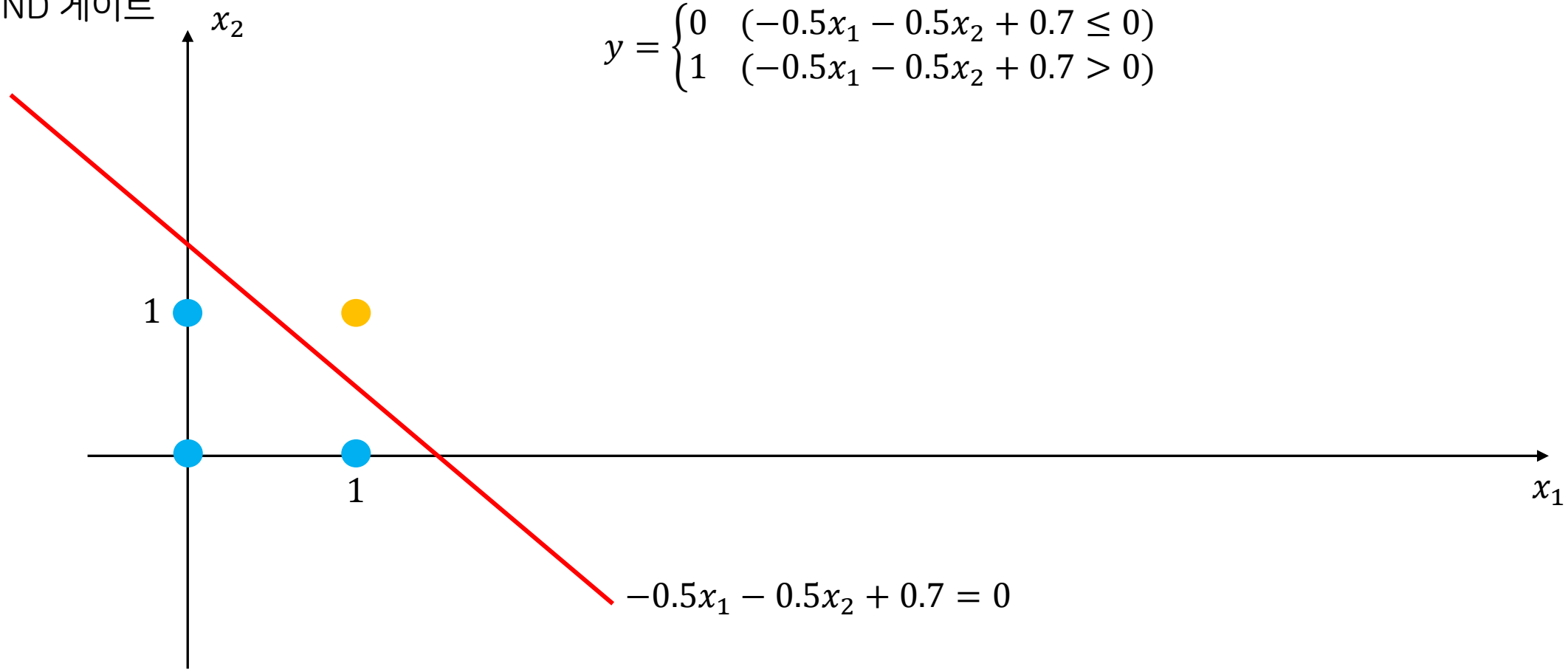
$$y = \begin{cases} 0 & (0.5x_1 + 0.5x_2 - 0.7 \leq 0) \\ 1 & (0.5x_1 + 0.5x_2 - 0.7 > 0) \end{cases}$$



퍼셉트론(Perceptron)

- 퍼셉트론의 시각화
  - NAND 게이트

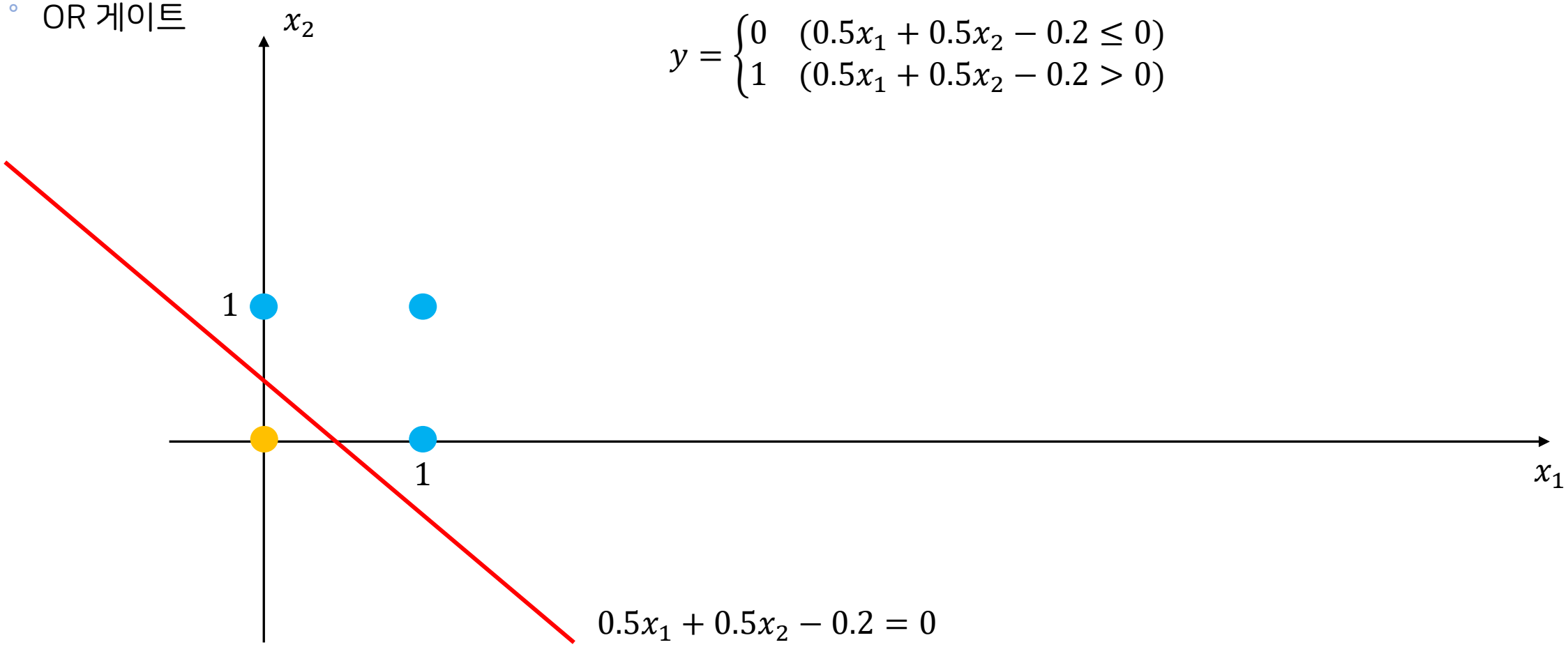
$$y = \begin{cases} 0 & (-0.5x_1 - 0.5x_2 + 0.7 \leq 0) \\ 1 & (-0.5x_1 - 0.5x_2 + 0.7 > 0) \end{cases}$$



퍼셉트론(Perceptron)

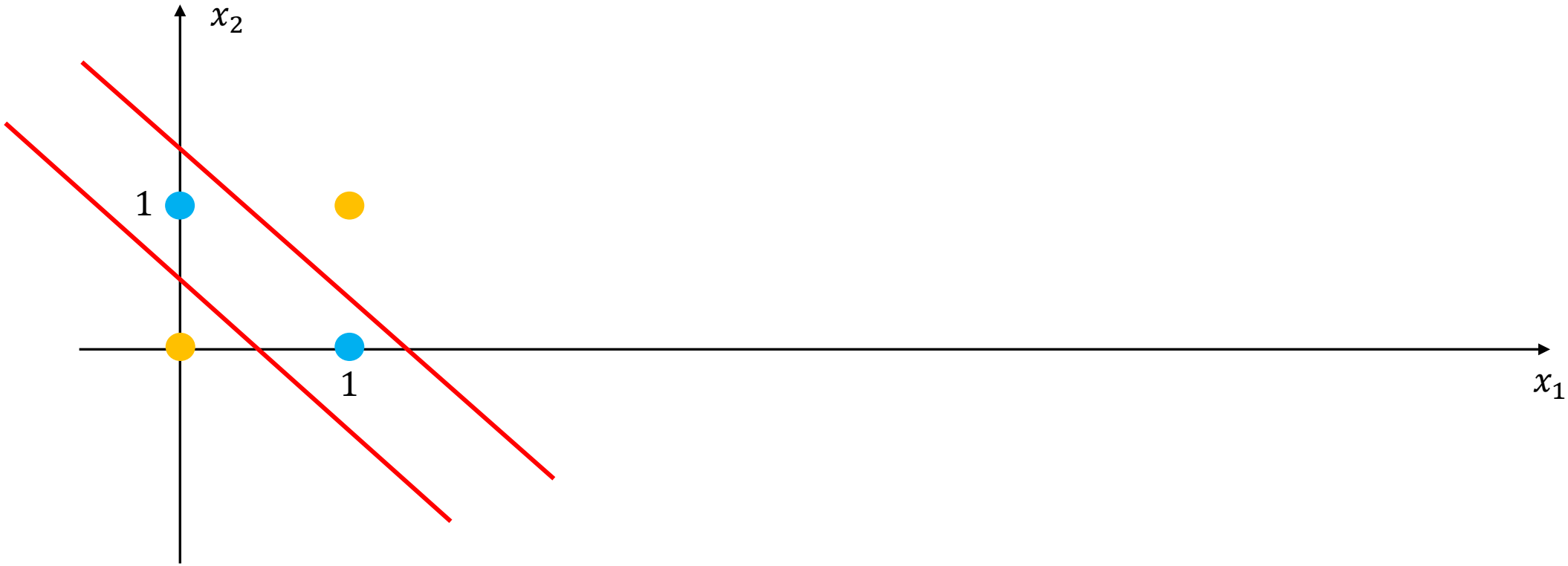
- 퍼셉트론의 시각화
  - OR 게이트

$$y = \begin{cases} 0 & (0.5x_1 + 0.5x_2 - 0.2 \leq 0) \\ 1 & (0.5x_1 + 0.5x_2 - 0.2 > 0) \end{cases}$$



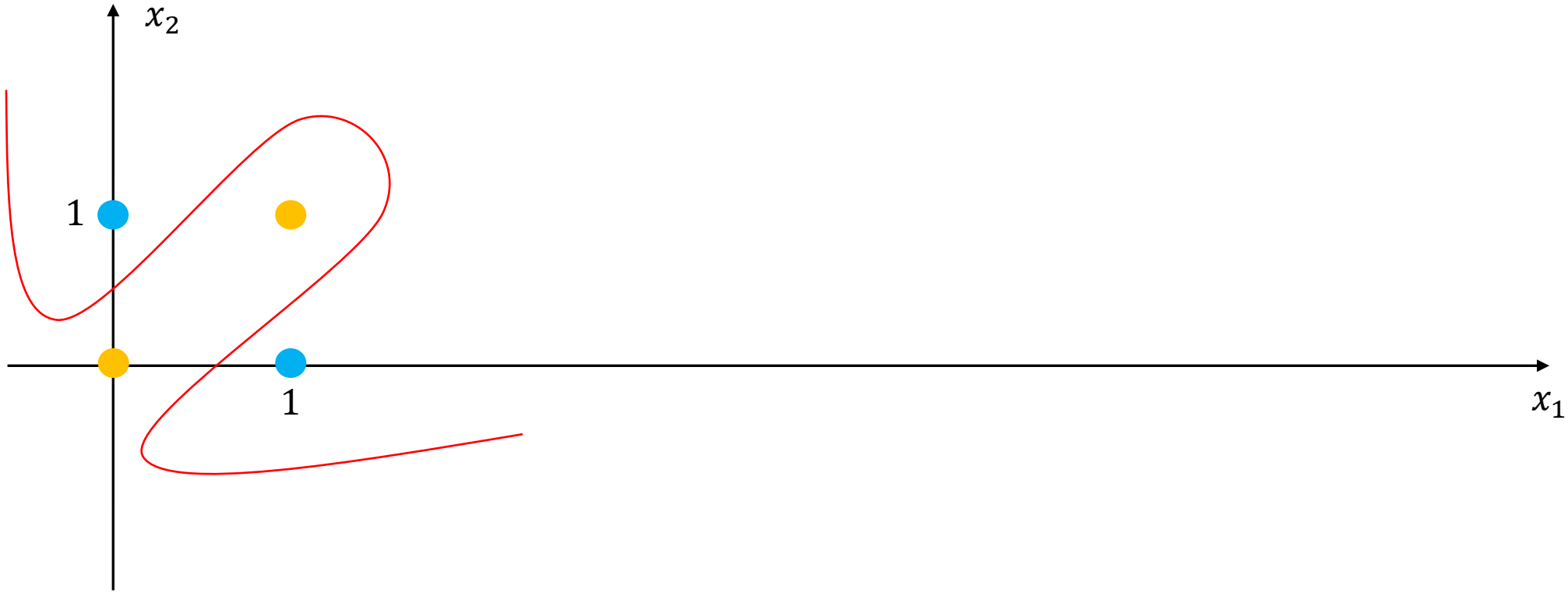
퍼셉트론(Perceptron)

- 퍼셉트론의 시각화
  - XOR 게이트
    - 두 개의 선형 분류기를 만들거나 or 선형이 아닌 비선형으로 만들기



퍼셉트론(Perceptron)

- 퍼셉트론의 시각화
  - XOR 게이트
    - 두 개의 선형 분류기를 만들거나 or 선형이 아닌 비선형으로 만들기



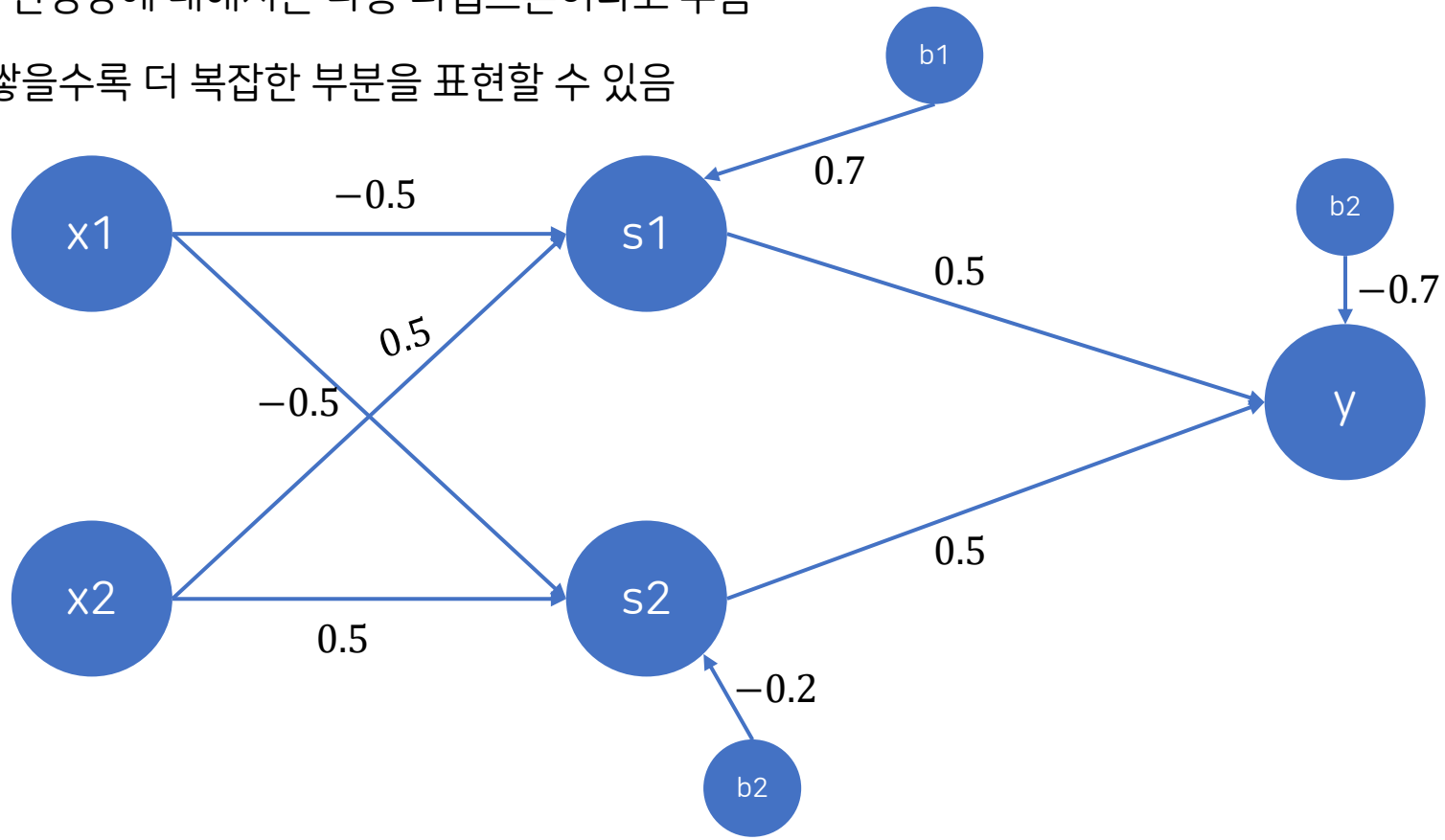
퍼셉트론(Perceptron)

- 한계점
  - 단층 퍼셉트론(Single Layer Perceptron)으로 XOR 게이트를 구현할 수 없음
  - 해결 방법 : 다층 퍼셉트론(Multi Layer Perceptron)의 등장
    - 기존의 조합하였던 AND, NAND, OR을 적절하게 조합하여 XOR 게이트로 만듦
- XOR 게이트의 진리표
  - x_1, x_2는 입력신호, w_1, w_2는 가중치 신호, s_1, s_2는 각각 NAND와 OR 게이트를 거치고 난 후의 결과
  - 최종 결과는 s_1, s_2를 AND 게이트를 거치게 한 후의 결과

입력(Input)		출력(Output)		
X1	X2	S1	S2	Y1
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0

퍼셉트론(Perceptron)

- XOR 게이트의 퍼셉트론
  - 아래와 같은 퍼셉트론을 2층 신경망이라고 표현
    - 2층 이상의 신경망에 대해서는 다층 퍼셉트론이라고 부름
    - 층을 깊게 쌓을수록 더 복잡한 부분을 표현할 수 있음



퍼셉트론(Perceptron)

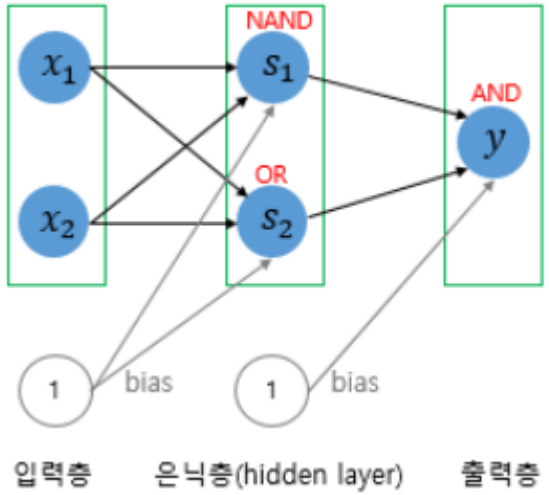
- XOR
  - Python Implementation

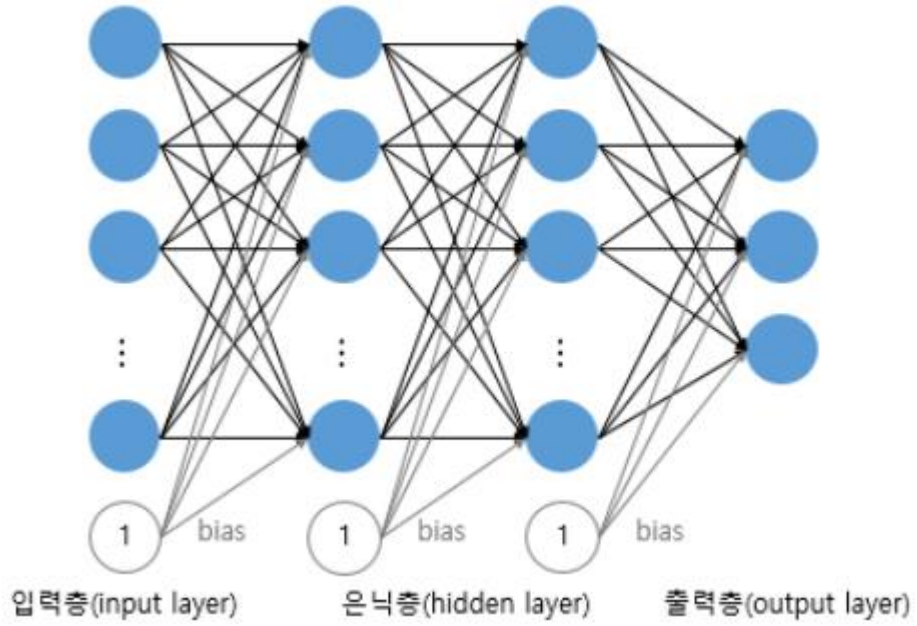
```
# 2.5.2 XOR 게이트 구현하기
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y


print("XOR")
print(XOR(0, 0))  # 0
print(XOR(0, 1))  # 1
print(XOR(1, 0))  # 1
print(XOR(1, 1))  # 0
```

[2] 밑바닥부터 시작하는 딥러닝 github

신경망(Neural Network)

- 인공 신경망(Artificial Neural Network)
  - 입력층(Input Layer), 은닉층(Hidden Layer), 출력층(Output Layer)로 구성



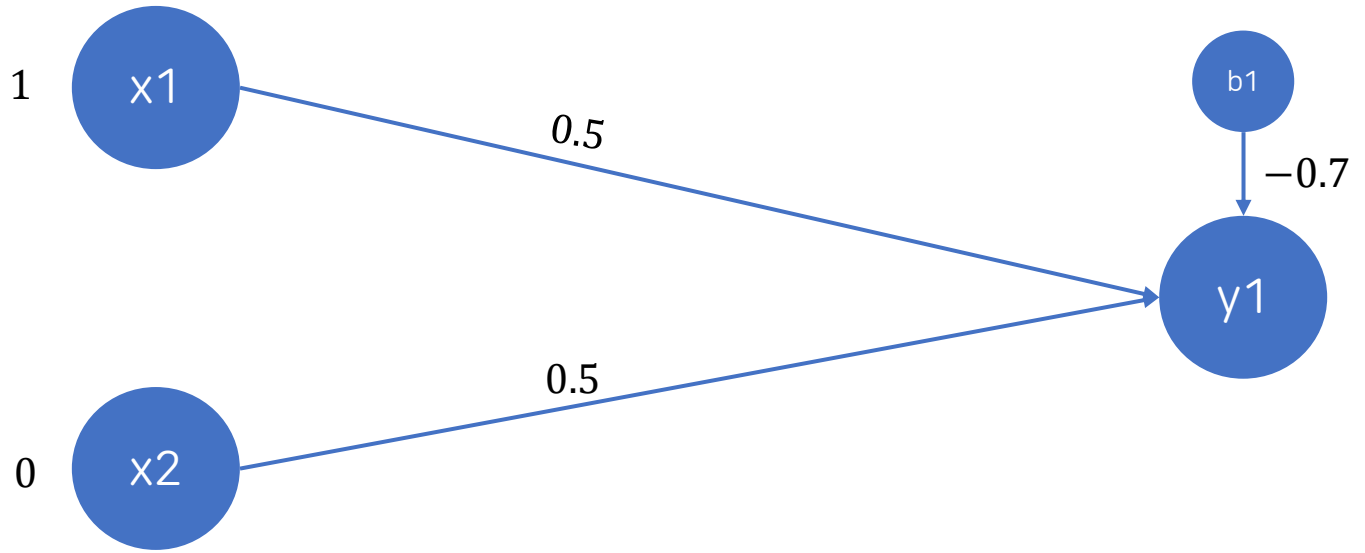


[3] https://wikidocs.net/24958

신경망(Neural Network)

- 인공 신경망(Artificial Neural Network)
  - 여러 개의 퍼셉트론으로 구성
  - 각 퍼셉트론 간의 연산이 완료되면 활성화 함수(activation function)을 통해 최종 연산이 진행됨.

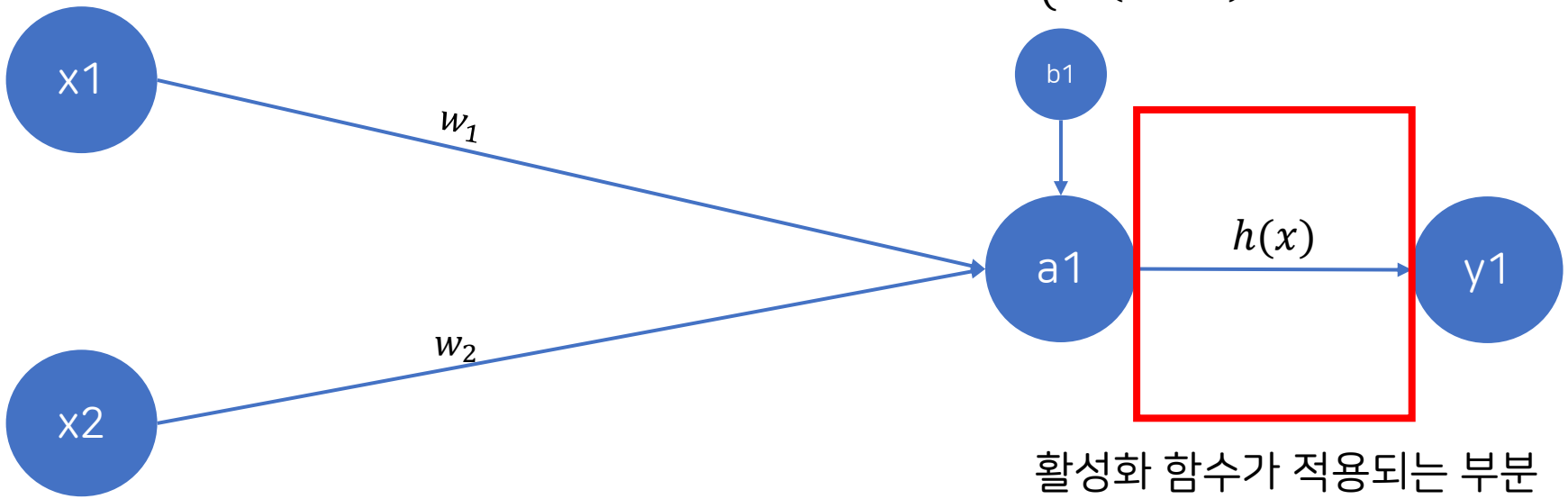
$$y = \begin{cases} 0 & (0.5x_1 + 0.5x_2 - 0.7 \leq 0) \\ 1 & (0.5x_1 + 0.5x_2 - 0.7 > 0) \end{cases}$$



신경망(Neural Network)

- 인공 신경망(Artificial Neural Network)
  - 여러 개의 퍼셉트론으로 구성
  - 각 퍼셉트론 간의 연산이 완료되면 활성화 함수(activation function)을 통해 최종 연산이 진행됨.

$$y = h(w_1 x_1 + w_2 x_2 + b) \qquad h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$

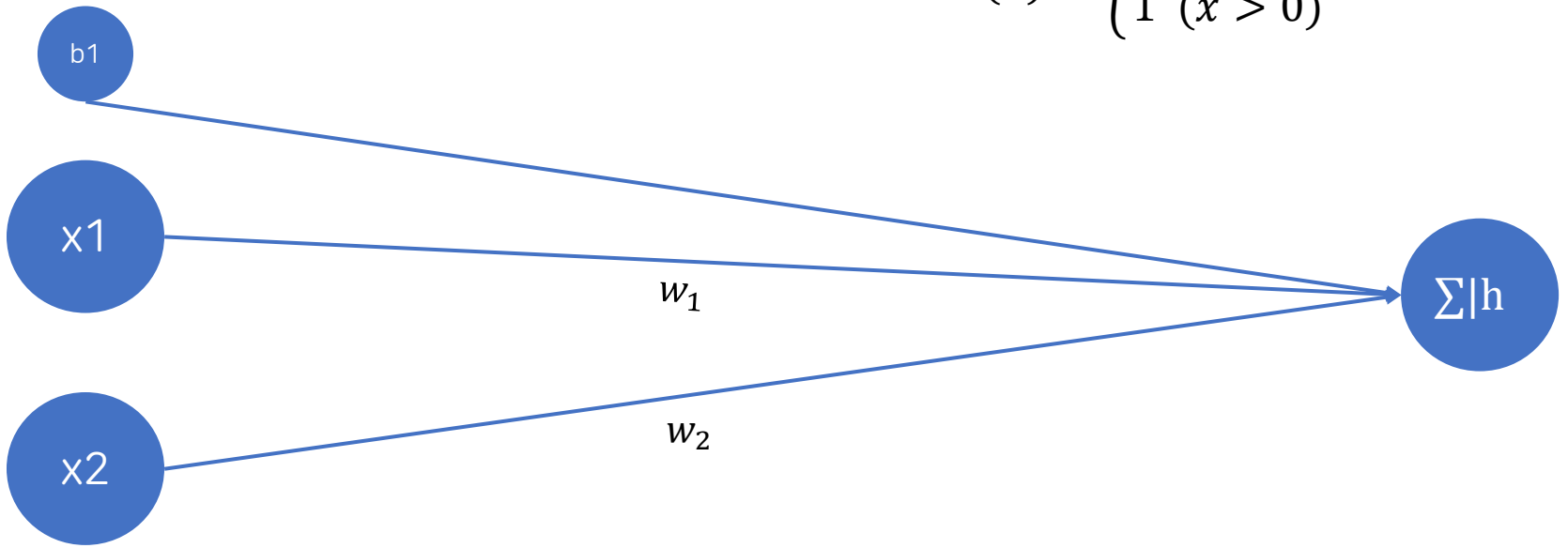


활성화 함수가 적용되는 부분
(입력 신호의 총합을 출력 신호로 변환함)

신경망(Neural Network)

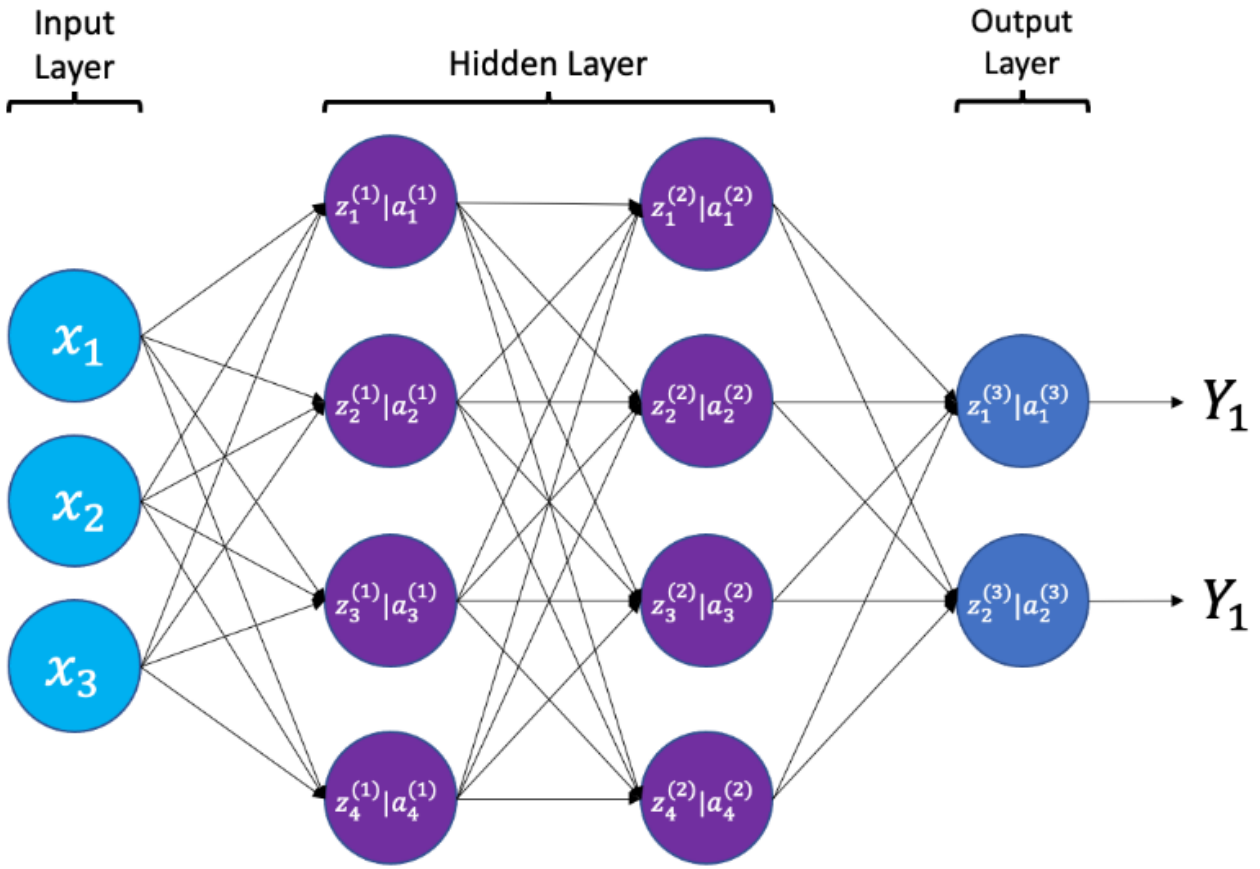
- 인공 신경망(Artificial Neural Network)
  - 여러 개의 퍼셉트론으로 구성
  - 각 퍼셉트론 간의 연산(선형 결합)이 완료되면 활성화 함수(activation function)을 통해 최종 연산이 진행됨.

$$y = h(w_1 x_1 + w_2 x_2 + b) \qquad h(x) = \begin{cases} 0 & (x \leq 0) \\ 1 & (x > 0) \end{cases}$$



신경망(Neural Network)

- 인공 신경망(Artifical Neural Network)의 구조



[4] https://carpentries-incubator.github.io/machine-learning-neural-python/04-create_net/index.html

활성화 함수(Activation Function)

- 활성화 함수(Activation Function)
  - 입력 신호의 총합을 출력 신호로 변환해주는 함수
  - 활성화 함수는 비선형 함수이어야만 함.

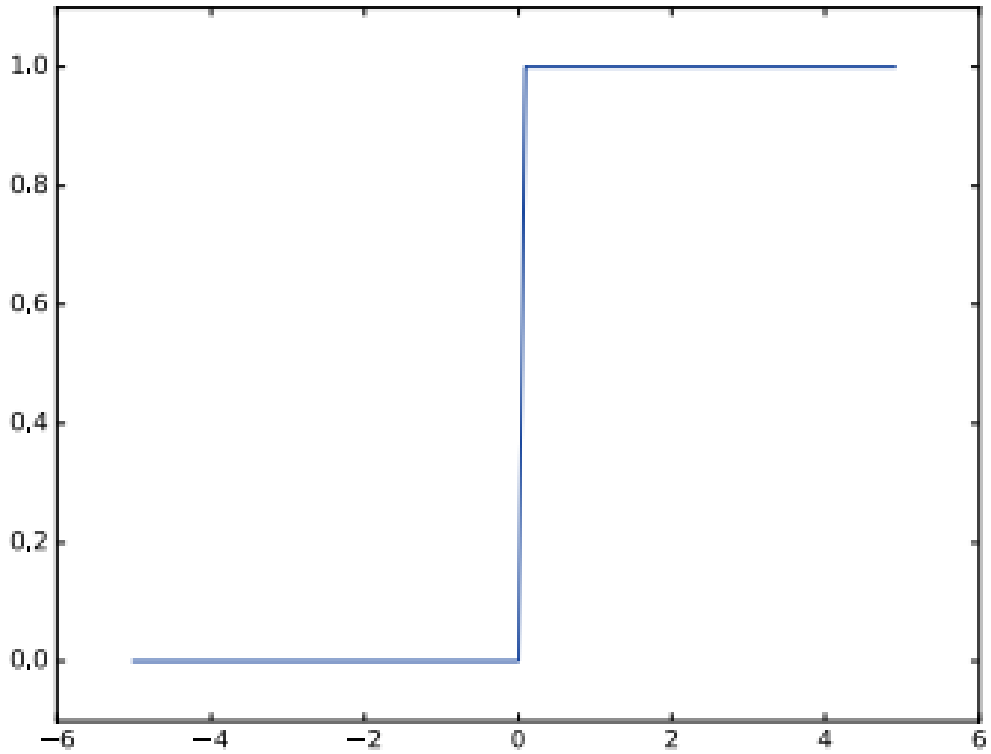
- 선형함수가 되면 안되는 이유
  - $f(x) = w_1x_1 + w_2x_2 + w_3x_3 + b$가 출력값의 결과이고 활성화함수가 $\mathrm{g}(x) = ax$라고 한다면?
    - 결과는 $\mathrm{g}(f(x)) = aw_1x_1 + aw_2x_2 + aw_3x_3 + b$
    - 가중치만 달라질 뿐 전체적인 함수의 변화가 있는 것이 아님 -> 굳이 은닉층을 써야 할까?

활성화 함수(Activation Function)

- 계단 함수(Step function)

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

```python
# numpy 배열을 위한 구현
def step_function(x):
    # y = x > 0
    # return y.astype(np.int)
    return np.array(x > 0, dtype=np.int)


# 3.2.3 계단 함수의 그래프
x = np.arange(-5.0, 5.0, 0.1)
y = step_function(x)

plt.plot(x, y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
# plt.show()
```



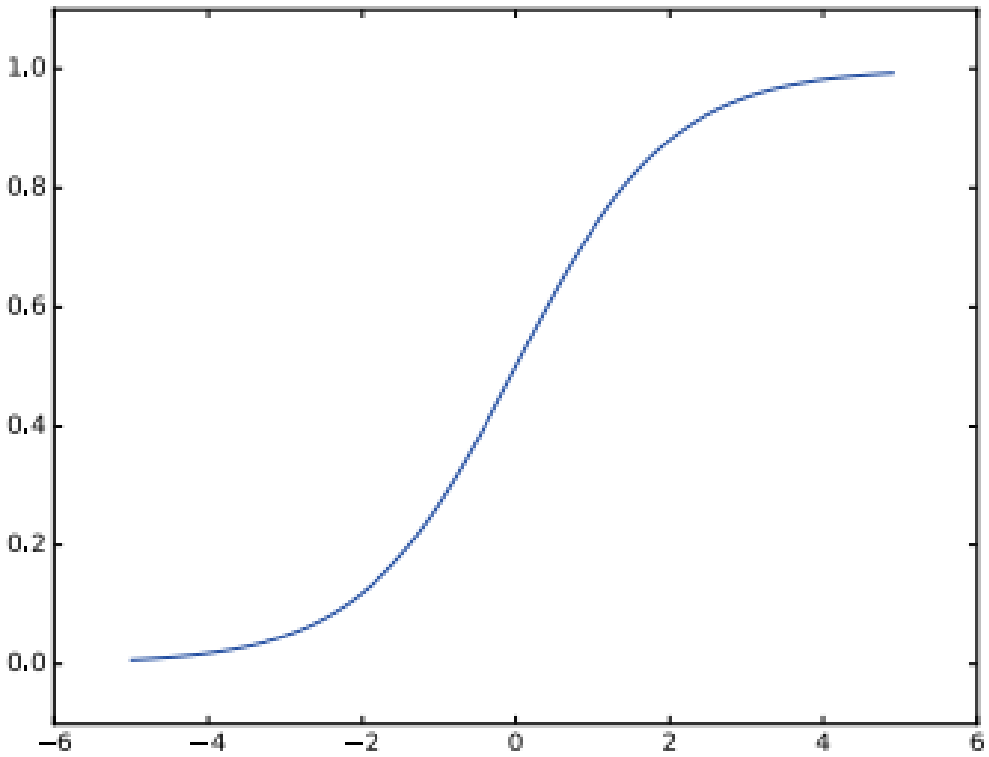
[6] 밑바닥부터 시작하는 딥러닝 github

활성화 함수(Activation Function)

- 시그모이드 함수(Sigmoid function)

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

```
# 3.2.4 시그모이드 함수 구현하기
def sigmoid(x):
    return 1 / (1 + np.exp(-x))


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

plt.plot(x, y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
plt.show()
```



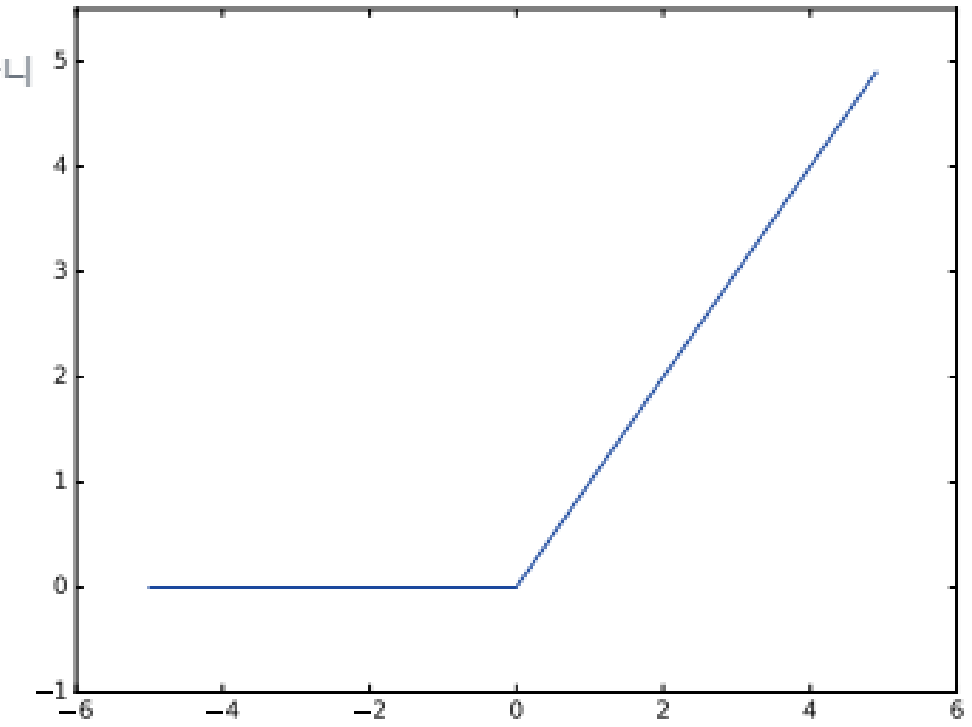
[7] 밑바닥부터 시작하는 딥러닝 github

활성화 함수(Activation Function)

- 렐루 함수(ReLU function)

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

```
# 3.2.7 ReLU 함수
# Rectified Linear Unit - 입력이 0을 넘으면 입력 그대로, 아니
def relu(x):
    return np.maximum(0, x)
x = np.arange(-5.0, 5.0, 0.1)
y = sigmoid(x)

plt.plot(x, y)
plt.ylim(-0.1, 1.1)  # y축의 범위 지정
plt.show()
```

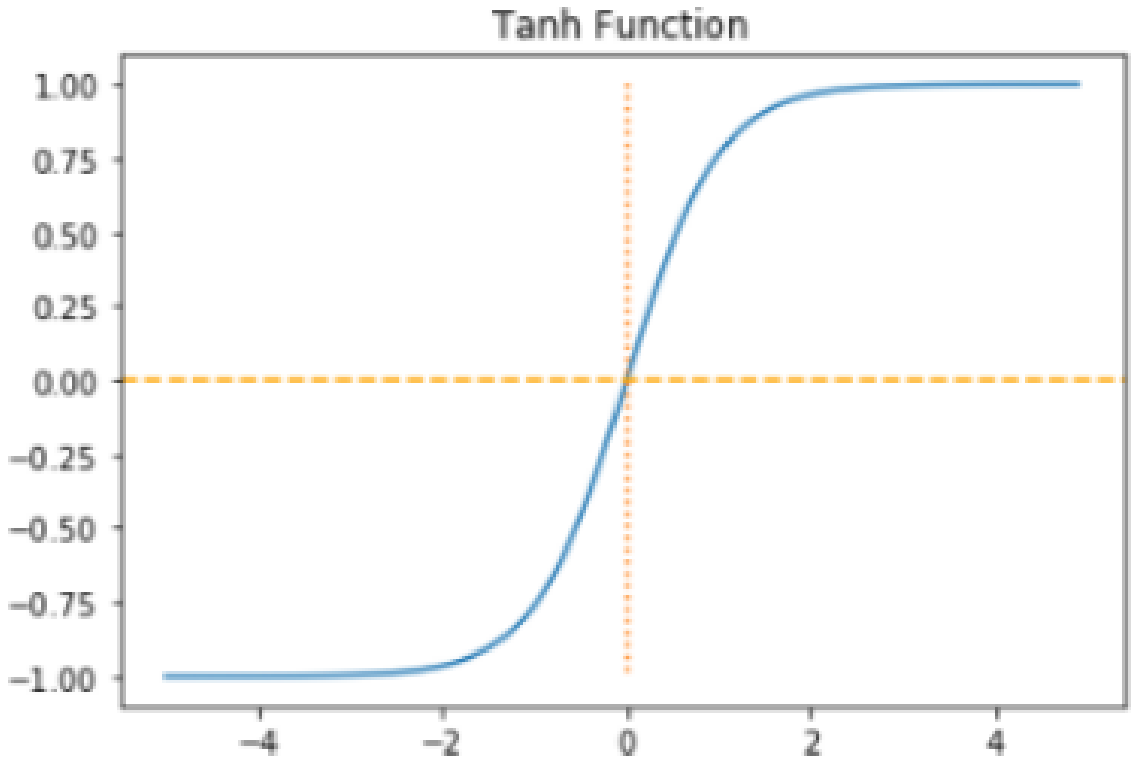


[8] 밑바닥부터 시작하는 딥러닝 github

활성화 함수(Activation Function)

- 하이퍼볼릭탄젠트 함수(Hyperbolic tangent function)

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



[9] https://wikidocs.net/24987

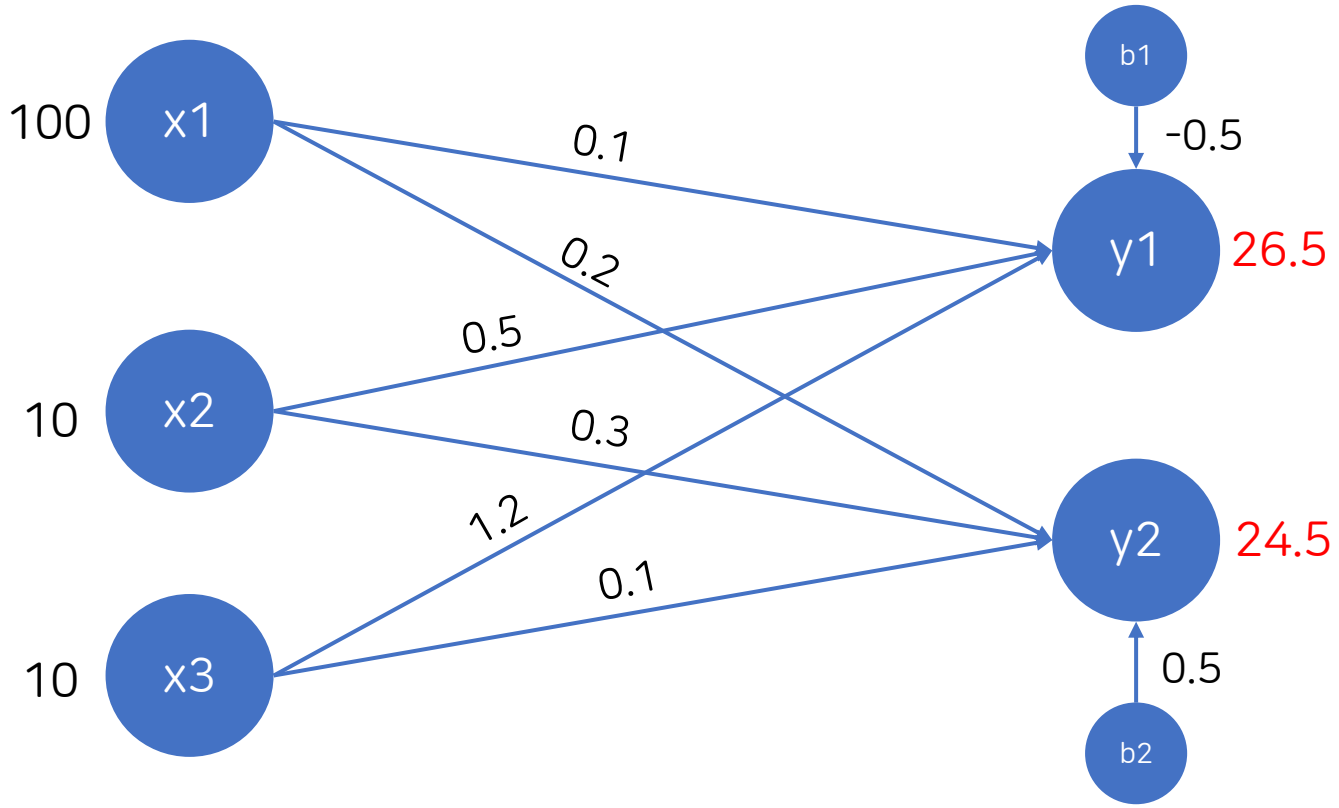
인공 신경망의 계산 방법

- 다차원 배열의 계산
  - 신경망에서의 행렬 곱
    - 데이터를 1개씩만 처리하는 것이 아닌, 여러 개를 가지고 연산을 진행하기 때문에 행렬이 필요
    - 신경망의 input과 가중치의 곱을 행렬곱으로 표현

Index	계획형(x1)	감성형(x2)	외향성(x3)
1	50	70	90
2	35	60	25
3	40	100	60
4	100	10	10
5	70	70	70

인공 신경망의 계산 방법

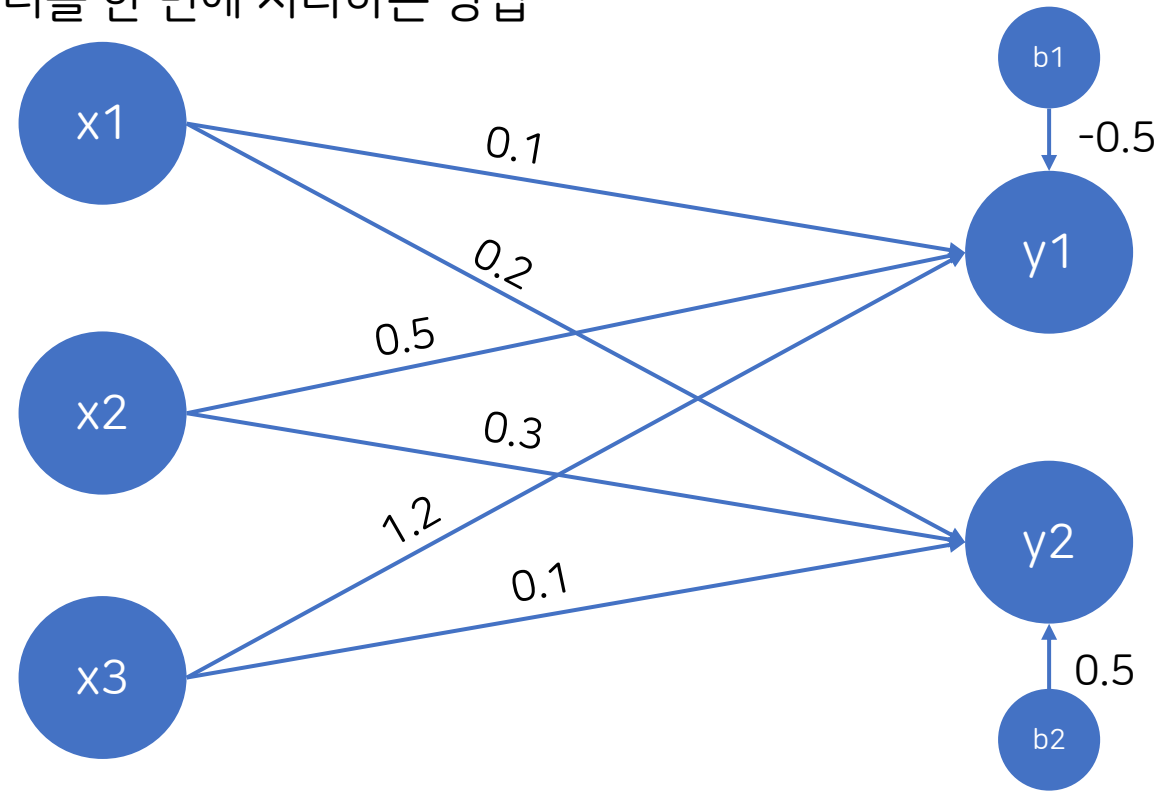
- 인공 신경망의 연산 과정 예시
  - 1가지 데이터에 대해서만 연산을 진행하는 경우



$$[100 \;\; 10 \;\; 10] \begin{bmatrix} 0.1 & 0.2 \\ 0.5 & 0.3 \\ 1.2 & 0.1 \end{bmatrix} + \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} = [26.5 \;\; 24.5]$$

인공 신경망의 계산 방법

- 인공 신경망의 연산 과정 예시
  - 2가지 이상의 데이터를 한 번에 처리하는 방법



$$\begin{bmatrix} 50 & 70 & 90 \\ 35 & 60 & 25 \\ 40 & 100 & 60 \\ 100 & 10 & 10 \\ 70 & 70 & 70 \end{bmatrix} \begin{bmatrix} 0.1 & 0.2 \\ 0.5 & 0.3 \\ 1.2 & 0.1 \end{bmatrix} + \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} = \begin{bmatrix} 147.5 & 40.5 \\ 63.0 & 28.0 \\ 125.5 & 44.5 \\ 26.5 & 24.5 \\ 125.5 & 42.5 \end{bmatrix}$$

Next Study

- 다층 신경망 구현 및 MNIST 데이터를 활용하여 처리하기

Any Questions?