XOR 해결을 위한 다층 퍼셉트론

1단계 w1, b1 그리고 w2, b2 정의

2단계 w3, b3 정의

In [1]:

```
import numpy as np

#가중치 weight (1 단계)
w1 = np.array([-2, -2])
w2 = np.array([2, 2])

# 바이어스 bias (1 단계)
b1 = 3
b2 = -1
#-----------------------------------
#가중치 weight (2 단계)
w3 = np.array([1, 1])

# 바이어스 bias (2 단계)
b3 = -1
```

다층퍼셉트론(MLP; Multilayer Perceptron) 함수 정의

In [2]:

```
# y = wx+b
def MLP(x, w, b):
    y = np.sum(x * w) + b
    if y <= 0:
        return 0   #시그모이드 결과 0 출력을 의미
    else:
        return 1   #시그모이드 결과 1 출력을 의미
```

NAND 게이트 함수 정의 (MLP 함수 호출) : 1단계에 해당하는 w1, b1를 할당

In [3]:

```
def NAND(x1,x2):
    result = MLP(np.array([x1, x2]), w1, b1)
    print("NAND 게이트 출력 : ",result)
    return result  # w1 = np.array([-2, -2]), b1 = 3
```

OR 게이트 함수 정의 (MLP 함수 호출) : 1단계에 해당하는 w2, b2를 할당

In [4]:

```
# OR 게이트
def OR(x1,x2):
    result = MLP(np.array([x1, x2]), w2, b2)
    print("OR 게이트 출력 : ",result)
    return result  # w2 = np.array([2, 2]), b2 = -1
```

AND 게이트 함수 정의 (MLP 함수 호출) : 2단계에 해당하는 w3, b3 할당

In [5]:

```
# AND 게이트
def AND(hidden_x1, hidden_x2):
    result = MLP(np.array([hidden_x1, hidden_x2]), w3, b3)
    print("AND 게이트 출력 : ",result)
    return result  # w3 = np.array([1, 1]), b3 = -1
```

XOR 게이트 함수 정의 (AND 함수 호출)

In [6]:

```
# XOR 게이트
def XOR(x1,x2):
    result = AND(NAND(x1, x2),OR(x1,x2))
    print("XOR 게이트 출력(NAND와 OR를 기반으로 AND연산 결과) : ", result)
    return result #NAND, OR는 1단계에 해당,1단계 결과를 AND로 2단계 수행
```

XOR 게이트 함수 호출

In [7]:

```
# x1, x2 값을 번갈아 대입해 가며 최종값 출력
print('__name__ : ', __name__) #import하지 않고, 현재의 모듈에의해 실행되었다면 '__main__'출력됨

number = 1

if __name__ == '__main__':
    for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:
        print("Wn", "● ", number, "번째 for문" )
        print('2개의 x 값 출력 :', x)
        print('첫번째 x 값 : ', x[0], ', 두번째 x 값 : ', x[1])
        print('-'*80)
        y = XOR(x[0], x[1]) # XOR 함수 호출
        print('-'*80)
        print("출력 값: ", y)
        number += 1

```

```
__name__ :  __main__

 ●  1 번째 for문
2개의 x 값 출력 : (0, 0)
첫번째 x 값 :  0 , 두번째 x 값 :  0
--------------------------------------------------------------------------------
NAND 게이트 출력 :  1
OR 게이트 출력 :  0
AND 게이트 출력 :  0
XOR 게이트 출력(NAND와 OR를 기반으로 AND연산 결과) :  0
--------------------------------------------------------------------------------
출력 값:  0

 ●  2 번째 for문
2개의 x 값 출력 : (1, 0)
첫번째 x 값 :  1 , 두번째 x 값 :  0
--------------------------------------------------------------------------------
NAND 게이트 출력 :  1
OR 게이트 출력 :  1
AND 게이트 출력 :  1
XOR 게이트 출력(NAND와 OR를 기반으로 AND연산 결과) :  1
--------------------------------------------------------------------------------
출력 값:  1

 ●  3 번째 for문
2개의 x 값 출력 : (0, 1)
첫번째 x 값 :  0 , 두번째 x 값 :  1
--------------------------------------------------------------------------------
NAND 게이트 출력 :  1
OR 게이트 출력 :  1
AND 게이트 출력 :  1
XOR 게이트 출력(NAND와 OR를 기반으로 AND연산 결과) :  1
--------------------------------------------------------------------------------
출력 값:  1

 ●  4 번째 for문
2개의 x 값 출력 : (1, 1)
첫번째 x 값 :  1 , 두번째 x 값 :  1
--------------------------------------------------------------------------------
NAND 게이트 출력 :  0
OR 게이트 출력 :  1
AND 게이트 출력 :  0
```

```
XOR 게이트 출력(NAND와 OR를 기반으로 AND연산 결과) :  0
--------------------------------------------------------------------------------
출력 값:  0
```

XOR 해결을 위한 다층 퍼셉트론 (한 번 더 연습하기)

In [8]:

```python
import numpy as np

# 가중치와 바이어스
w1 = np.array([-2, -2])
w2 = np.array([2, 2])
w3 = np.array([1, 1])
b1 = 3
b2 = -1
b3 = -1

# 퍼셉트론
def MLP(x, w, b):
    y = np.sum(w * x) + b
    if y <= 0:
        return 0   #시그모이드 결과 0 출력을 의미
    else:
        return 1   #시그모이드 결과 1 출력을 의미

# NAND 게이트
def NAND(x1,x2):
    return MLP(np.array([x1, x2]), w1, b1)  # w1 = np.array([-2, -2]), b1 = 3

# OR 게이트
def OR(x1,x2):
    return MLP(np.array([x1, x2]), w2, b2)  # w2 = np.array([2, 2]), b2 = -1

# AND 게이트
def AND(x1,x2):
    return MLP(np.array([x1, x2]), w3, b3)  # w3 = np.array([1, 1]), b3 = -1

# XOR 게이트
def XOR(x1,x2):
    return AND(NAND(x1, x2),OR(x1,x2))


# x1, x2 값을 번갈아 대입해 가며 최종값 출력
print(__name__) #import하지 않고, 현재의 모듈에의해 실행되었다면 '__main__'출력됨
if __name__ == '__main__':
    for x in [(0, 0), (0, 1), (1, 0), (1, 1)]:
        print('-'*100)
        print("입력 값: " + str(x), end= "  ")
        y = XOR(x[0], x[1])
        print("--> 최종 출력 값: " + str(y))
```

```
__main__
----------------------------------------------------------------------------------------------------
입력 값: (0, 0)  --> 최종 출력 값: 0
----------------------------------------------------------------------------------------------------
입력 값: (0, 1)  --> 최종 출력 값: 1
----------------------------------------------------------------------------------------------------
입력 값: (1, 0)  --> 최종 출력 값: 1
----------------------------------------------------------------------------------------------------
입력 값: (1, 1)  --> 최종 출력 값: 0
```

In [9]:

```
#참고 : 현재의 모듈 이름
print(__name__)
```

```
__main__
```

```
if __name__ == '__main__':

__name__ 은 현재 모듈의 이름을 담고있는 내장 변수이다.
▶ 파이썬에서 직접 실행된 모듈의 경우 __main__이라는 값을 가지게 된다.
▶ 파이썬에서 직접 실행되지 않고, import된 모듈은 "모듈의 이름(파일명)"을 가지게 된다.

아래와 같이 if __name__ == '__main__': 코드를 사용하면
__name__ 변수의 값이'__main__' 인지 확인할 수 있다.
```

In [ ]:

```

```