

AI 프로그래밍

- 5주차



이하 공전 컴퓨터 정보과
민 정혜

- 다중 선형 회귀 복습
- Numpy, Matplotlib 실습
- 퍼셉트론

다중 선형 회귀

- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며, 정보를 추가해 새로운 예측값을 구하려면 변수의 개수를 늘려 다중 선형 회귀를 만들어 주어야 함

| | | | | |
|-------------------|----|----|----|----|
| 공부한 시간(x_1) | 2 | 4 | 6 | 8 |
| 과외 수업 횟수(x_2) | 0 | 4 | 2 | 3 |
| 성적(y) | 81 | 93 | 91 | 97 |

표 4-1 공부한 시간, 과외 수업 횟수에 따른 성적 데이터

다중 선형 회귀

- 그럼 지금부터 두 개의 독립 변수 x_1 과 x_2 가 생긴 것임
- 이를 사용해 종속 변수 y 를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴

$$y = a_1x_1 + a_2x_2 + b$$

- 이번에는 x 의 값이 두 개이므로 다음과 같이 data 리스트를 만들고 x_1 과 x_2 라는 두 개의 독립 변수 리스트를 만들어 줌

```
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]  
x1 = [i[0] for i in data]  
x2 = [i[1] for i in data]  
y = [i[2] for i in data]
```

다중 선형 회귀

- data가 그래프로 어떻게 보이는지를 확인해 보자
- 먼저 x, y 두 개의 축이던 이전과 달리 x_1, x_2, y 이렇게 세 개의 축이 필요함
- 3D 그래프를 그려주는 라이브러리를 아래와 같이 불러옴

```
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d # 3D 그래프 그리는 라이브러리 가져오기

ax = plt.axes(projection='3d') # 그래프 유형 정하기
ax.set_xlabel('study_hours')
ax.set_ylabel('private_class')
ax.set_zlabel('Score')
ax.scatter(x1, x2, y)
plt.show()
```

다중 선형 회귀

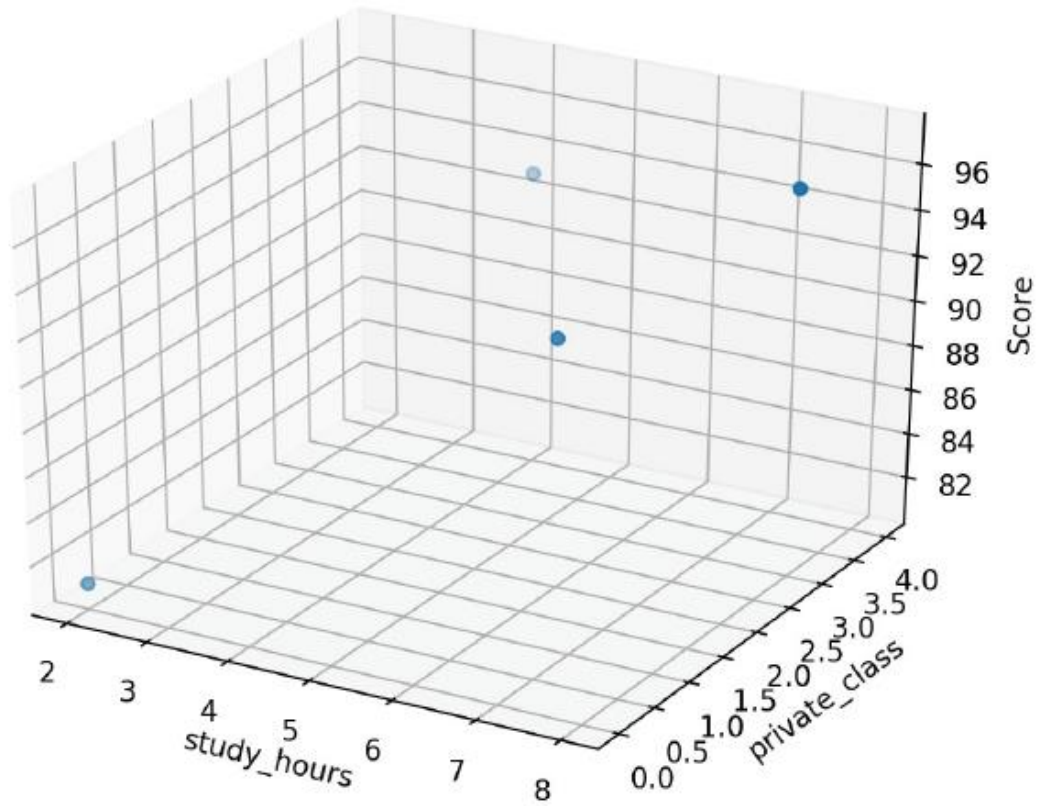


그림 4-6 축이 하나 더 늘어 3D로 배치된 모습

다중 선형 회귀

- 이제 x 가 두 개가 되었으므로 x 과 x_2 두 가지의 변수를 지정함
- 각각의 값에 기울기 a 값이 다르므로 기울기도 a_1 과 a_2 이렇게 두 가지를 만들
- 각각 앞서 했던 방법과 같은 방법으로 경사 하강법을 적용하고 학습률을 곱해 기존의 값을 업데이트 함

```
y_pred = a1 * x1_data + a2 * x2_data + b # y를 구하는 식을 세우기
error = y_data - y_pred # 오차를 구하는 식
a1_diff = -(1/len(x1_data)) * sum(x1_data * (error)) # 오차 함수를 a1로 미분한 값
a2_diff = -(1/len(x2_data)) * sum(x2_data * (error)) # 오차 함수를 a2로 미분한 값

b_new = -(1/len(x1_data)) * sum(y_data - y_pred) # 오차 함수를 b로 미분한 값
a1 = a1 - lr * a1_diff # 학습률을 곱해 기존의 a1 값 업데이트
a2 = a2 - lr * a2_diff # 학습률을 곱해 기존의 a2 값 업데이트
b = b - lr * b_diff # 학습률을 곱해 기존의 b 값 업데이트
```

다중 선형 회귀

- 다중 선형 회귀 문제에서의 기울기 a_1 , a_2 와 절편 b 의 값을 찾아 확인할 수 있음

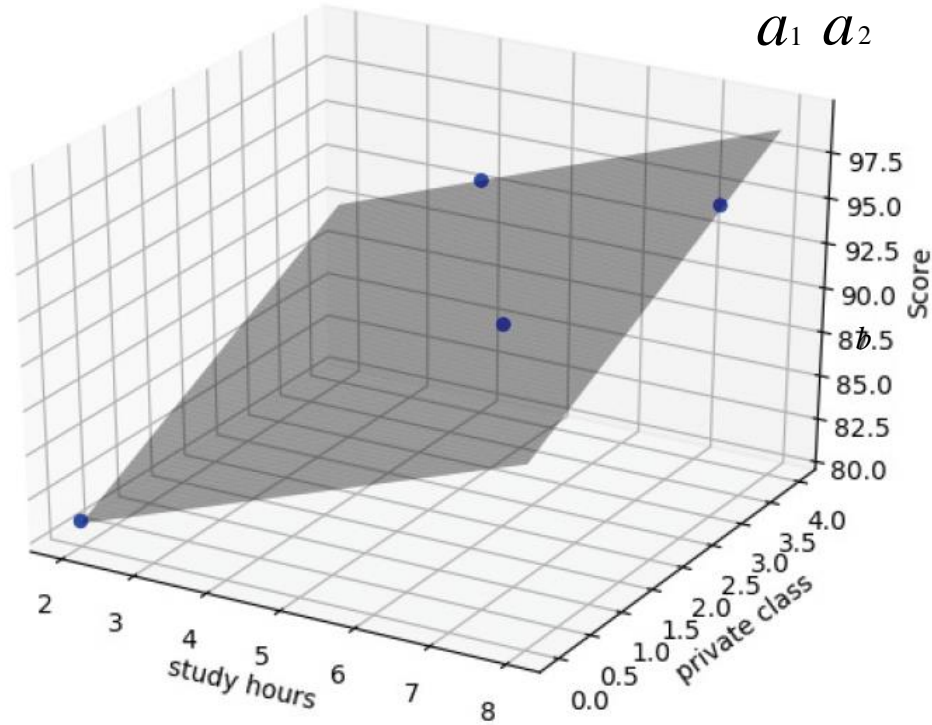


그림 4-7 다중 선형 회귀의 그래프: 차원이 하나 더 늘어난 모습

자료실의 colab_04_Multi_Linear_Regression_loss.ipynb을 실행 한 후, 첫번째와 두번째 loss 확인.

다중 선형 회귀

- 1차원 예측 직선이 3차원 '예측 평면'으로 바뀜
- 과외 수업 횟수(privateclass)라는 새로운 변수가 추가됨
- 1차원 직선에서만 움직이던 예측 결과가 더 넓은 평면 범위 안에서 움직이게 됨
- 이로 인해 좀 더 정밀한 예측을 할 수 있게 된 것임

Numpy

cheat sheet : 문법 등을
요약해 놓은 것

Python For Data Science Cheat Sheet

NumPy Basics

Learn Python for Data Science interactively at www.datacamp.com



NumPy

The NumPy library is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

Use the following import convention:

```
>>> import numpy as np
```



NumPy Arrays

1D array

```
[1 2 3]
```

2D array

axis 1
axis 0

```
[[1.5 2. 3.]  
 [4. 5. 6.]]
```

3D array

axis 2
axis 1
axis 0

```
[[[1.5 2. 3.]  
 [4. 5. 6.]]  
 [[1.5 2. 3.]  
 [4. 5. 6.]]  
 [[1.5 2. 3.]  
 [4. 5. 6.]]]
```

Creating Arrays

```
>>> a = np.array([1,2,3])  
>>> b = np.array([(1.5,2,3), (4,5,6)], dtype = float)  
>>> c = np.array([(1.5,2,3), (4,5,6)], [(3,2,1), (4,5,6)],  
                 dtype = float)
```

Initial Placeholders

```
>>> np.zeros((3,4))  
>>> np.ones((2,3,4),dtype=np.int16)  
>>> d = np.arange(10,25,5)  
  
>>> np.linspace(0,2,9)  
  
>>> e = np.full((2,2),7)  
>>> f = np.eye(2)  
>>> np.random.random((2,2))  
>>> np.empty((3,2))
```

Create an array of zeros
Create an array of ones
Create an array of evenly spaced values (step value)
Create an array of evenly spaced values (number of samples)
Create a constant array
Create a 2X2 identity matrix
Create an array with random values
Create an empty array

I/O

Saving & Loading On Disk

```
>>> np.save('my_array', a)  
>>> np.savez('array.npz', a, b)  
>>> np.load('my_array.npy')
```

Saving & Loading Text Files

```
>>> np.loadtxt('myfile.txt')  
>>> np.genfromtxt('my_file.csv', delimiter=',')  
>>> np.savetxt('myarray.txt', a, delimiter=' ')
```

Data Types

```
>>> np.int64  
>>> np.float32  
>>> np.complex  
>>> np.bool  
>>> np.object  
>>> np.string_  
>>> np.unicode_
```

Signed 64-bit integer types
Standard double-precision floating point
Complex numbers represented by 128 floats
Boolean type storing TRUE and FALSE values
Python object type
Fixed-length string type
Fixed-length unicode type

Inspecting Your Array

```
>>> a.shape  
>>> len(a)  
>>> b.ndim  
>>> a.size  
>>> b.dtype  
>>> b.dtype.name  
>>> b.astype(int)
```

Array dimensions
Length of array
Number of array dimensions
Number of array elements
Data type of array elements
Name of data type
Convert an array to a different type

Asking For Help

```
>>> np.info(np.ndarray.dtype)
```

Array Mathematics

Arithmetic Operations

```
>>> g = a - b  
array([[ -0.5,  0. ,  0. ],  
       [ -3. , -3. , -3. ]])  
>>> np.subtract(a,b)  
>>> b + a  
array([[ 2.5,  4. ,  6. ],  
       [ 5. ,  7. ,  9. ]])  
>>> np.add(b,a)  
>>> a / b  
array([[ 0.66666667,  1. ,  1. ],  
       [ 0.25 ,  0.4 ,  0.5 ]])  
>>> np.divide(a,b)  
>>> a * b  
array([[ 1.5,  4. ,  9. ],  
       [ 4. , 10. , 18. ]])  
>>> np.multiply(a,b)  
>>> np.exp(b)  
>>> np.sqrt(b)  
>>> np.sin(a)  
>>> np.cos(b)  
>>> np.log(a)  
>>> e.dot(f)  
array([[ 7. ,  7.]])
```

Subtraction
Subtraction
Addition
Addition
Division
Division
Multiplication
Multiplication
Exponentiation
Square root
Print sines of an array
Element-wise cosine
Element-wise natural logarithm
Dot product

Comparison

```
>>> a == b  
array([[False,  True,  True],  
       [False, False, False]], dtype=bool)  
>>> a < 2  
array([ True, False, False], dtype=bool)  
>>> np.array_equal(a, b)
```

Element-wise comparison
Element-wise comparison
Array-wise comparison

Aggregate Functions

```
>>> a.sum()  
>>> a.min()  
>>> b.max(axis=0)  
>>> b.cumsum(axis=1)  
>>> a.mean()  
>>> b.median()  
>>> a.corrcoef()  
>>> np.std(b)
```

Array-wise sum
Array-wise minimum value
Maximum value of an array row
Cumulative sum of the elements
Mean
Median
Correlation coefficient
Standard deviation

Copying Arrays

```
>>> h = a.view()  
>>> np.copy(a)  
>>> h = a.copy()
```

Create a view of the array with the same data
Create a copy of the array
Create a deep copy of the array

Sorting Arrays

```
>>> a.sort()  
>>> c.sort(axis=0)
```

Sort an array
Sort the elements of an array's axis

Subsetting, Slicing, Indexing

Also see Lists

Subsetting

```
>>> a[2]  
3  
>>> b[1,2]  
6.0
```

Select the element at the 2nd index
Select the element at row 1 column 2 (equivalent to b[1][2])

Slicing

```
>>> a[0:2]  
array([1, 2])  
>>> b[0:2,1]  
array([ 2.,  5.])  
>>> b[:1]  
array([[1.5, 2., 3.]])  
>>> c[1,...]  
array([[ 3.,  2.,  1.],  
       [ 4.,  5.,  6.]])
```

Select items at index 0 and 1
Select items at rows 0 and 1 in column 1
Select all items at row 0 (equivalent to b[0:1, :])
Same as [1, :, :]
Reversed array a

Boolean Indexing

```
>>> a[a<2]  
array([1])  
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]  
array([ 4. ,  2. ,  6. ,  1.5])
```

Select elements from a less than 2

Fancy Indexing

```
>>> b[[1, 0, 1, 0], [0, 1, 2, 0]]  
array([ 4. ,  2. ,  6. ,  1.5])  
>>> b[[1, 0, 1, 0]][:, [0,1,2,0]]  
array([[ 1.5,  2. ,  3. ,  4. ],  
       [ 4. ,  5. ,  6. ,  4. ],  
       [ 1.5,  2. ,  3. ,  1.5]])
```

Select elements (1,0), (0,1), (2,2) and (0,0)
Select a subset of the matrix's rows and columns

Array Manipulation

Transposing Array

```
>>> i = np.transpose(b)  
>>> i.T
```

Permute array dimensions
Permute array dimensions

Changing Array Shape

```
>>> b.ravel()  
>>> g.reshape(3,-2)
```

Flatten the array
Reshape, but don't change data

Adding/Removing Elements

```
>>> h.resize((2,6))  
>>> np.append(h,g)  
>>> np.insert(a, 1, 5)  
>>> np.delete(a, [1])
```

Return a new array with shape (2,6)
Append items to an array
Insert items in an array
Delete items from an array

Combining Arrays

```
>>> np.concatenate((a,d),axis=0)  
array([ 1,  2,  3, 10, 15, 20])  
>>> np.vstack((a,b))  
array([[ 1. ,  2. ,  3. ],  
       [ 1.5,  2. ,  3. ],  
       [ 4. ,  5. ,  6. ]])  
>>> np.r_[e,f]  
>>> np.hstack((e,f))  
array([[ 7. ,  7. ,  1. ,  0. ],  
       [ 7. ,  7. ,  0. ,  1.]])  
>>> np.column_stack((a,d))  
array([[ 1, 10],  
       [ 2, 15],  
       [ 3, 20]])  
>>> np.c_[a,d]
```

Concatenate arrays
Stack arrays vertically (row-wise)
Stack arrays vertically (row-wise)
Stack arrays horizontally (column-wise)
Create stacked column-wise arrays
Create stacked column-wise arrays

Splitting Arrays

```
>>> np.hsplit(a,3)  
[array([1]), array([2]), array([3])]  
>>> np.vsplit(a,2)  
[array([[ 1.5,  2. ,  1. ],  
       [ 4. ,  5. ,  6. ]]),  
 array([[ 3. ,  2. ,  3. ],  
       [ 4. ,  5. ,  6. ]])]
```

Split the array horizontally at the 3rd index
Split the array vertically at the 2nd index



Matplot lib

Matplotlib for beginners

Matplotlib is a library for making 2D plots in Python. It is designed with the philosophy that you should be able to create simple plots with just a few commands:

1 Initialize

```
import numpy as np
import matplotlib.pyplot as plt
```

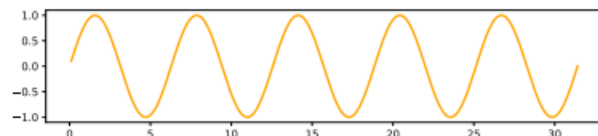
2 Prepare

```
X = np.linspace(0, 4*np.pi, 1000)
Y = np.sin(X)
```

3 Render

```
fig, ax = plt.subplots()
ax.plot(X, Y)
fig.show()
```

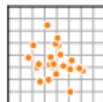
4 Observe



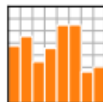
Choose

Matplotlib offers several kind of plots (see Gallery):

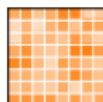
```
X = np.random.uniform(0, 1, 100)
Y = np.random.uniform(0, 1, 100)
ax.scatter(X, Y)
```



```
X = np.arange(10)
Y = np.random.uniform(1, 10, 10)
ax.bar(X, Y)
```



```
Z = np.random.uniform(0, 1, (8,8))
ax.imshow(Z)
```



```
Z = np.random.uniform(0, 1, (8,8))
ax.contourf(Z)
```



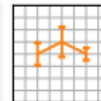
```
Z = np.random.uniform(0, 1, 4)
ax.pie(Z)
```



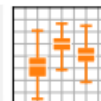
```
Z = np.random.normal(0, 1, 100)
ax.hist(Z)
```



```
X = np.arange(5)
Y = np.random.uniform(0, 1, 5)
ax.errorbar(X, Y, Y/4)
```



```
Z = np.random.normal(0, 1, (100,3))
ax.boxplot(Z)
```



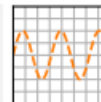
Tweak

You can modify pretty much anything in a plot, including limits, colors, markers, line width and styles, ticks and ticks labels, titles, etc.

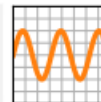
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, color="black")
```



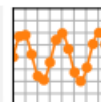
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linestyle="--")
```



```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, linewidth=5)
```



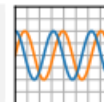
```
X = np.linspace(0, 10, 100)
Y = np.sin(X)
ax.plot(X, Y, marker="o")
```



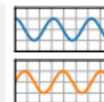
Organize

You can plot several data on the the same figure, but you can also split a figure in several subplots (named Axes):

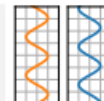
```
X = np.linspace(0, 10, 100)
Y1, Y2 = np.sin(X), np.cos(X)
ax.plot(X, Y1, X, Y2)
```



```
fig, (ax1, ax2) = plt.subplots((2,1))
ax1.plot(X, Y1, color="C1")
ax2.plot(X, Y2, color="C0")
```

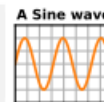


```
fig, (ax1, ax2) = plt.subplots((1,2))
ax1.plot(Y1, X, color="C1")
ax2.plot(Y2, X, color="C0")
```

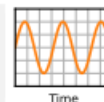


Label (everything)

```
ax.plot(X, Y)
fig.suptitle(None)
ax.set_title("A Sine wave")
```



```
ax.plot(X, Y)
ax.set_ylabel(None)
ax.set_xlabel("Time")
```

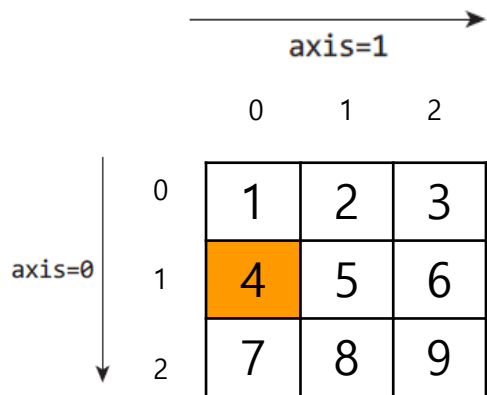


Explore

Figures are shown with a graphical user interface that allows to zoom and pan the figure, to navigate between the different views and to show the value under the mouse.

Save (bitmap or vector format)

```
fig.savefig("my-first-figure.png", dpi=300)
fig.savefig("my-first-figure.pdf")
```



| | | axis=1 → | | |
|----------|---|----------|---|---|
| | | 0 | 1 | 2 |
| axis=0 ↓ | 0 | 1 | 2 | 3 |
| | 1 | 4 | 5 | 6 |
| | 2 | 7 | 8 | 9 |

```
import numpy as np
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
a1=a[1:2,0:1]
a2=a[1:2,0]
a3=a[1,0:1]
a4=a[1,0]
print('a1',a1,a1.shape)
print('a2',a2,a2.shape)
print('a3',a3,a3.shape)
print('a4',a4,a4.shape)
```

numpy

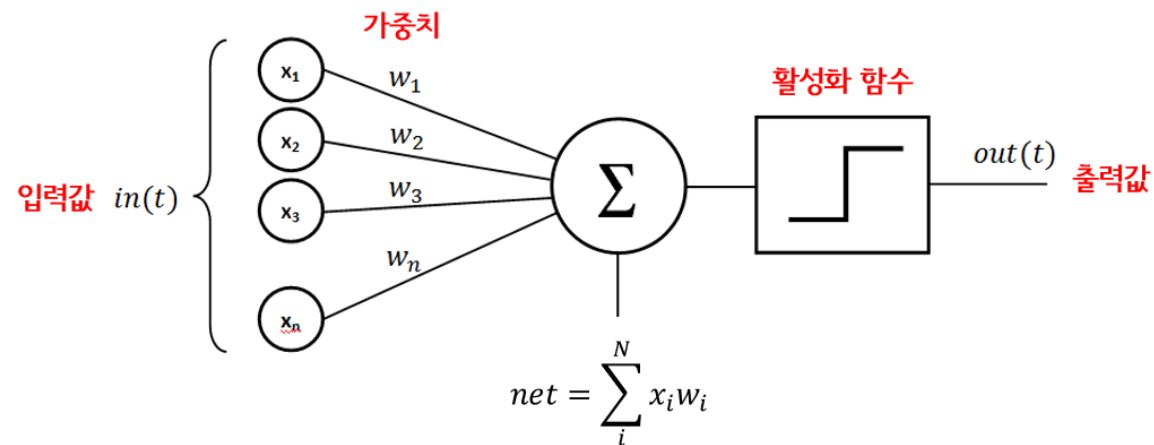
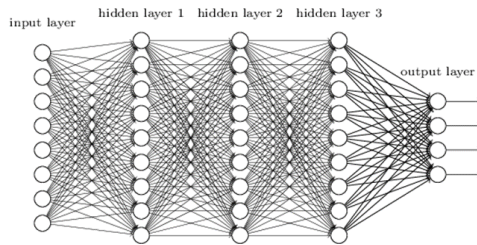
```
import numpy as np
a = np.array([1, 2, 3])
a1 = a[np.newaxis, :]
a2 = a[:, np.newaxis]
print('a', a, a.shape)
print('a1', a1, a1.shape)
print('a2', a2, a2.shape)
```

```
a   [1  2  3]           (3, )
a1  [[1  2  3]]        (1, 3)
a2  [[1]
      [2]
      [3]] (3, 1)
```

머신 러닝/ 딥러닝

머신 러닝

딥 러닝



퍼셉트론

신경망 (Neural Network)

신경계의 기본 단위인 뉴런을 모델화한 것이다. 하나의 인공 뉴런(퍼셉트론)에서는 다수의 입력 신호를 받아서 하나의 신호를 출력한다.

퍼셉트론 (Perceptron)

퍼셉트론(perceptron): 인공 뉴런 은 1957년에 로젠블라트(Frank Rosenblatt)가 고안한 인공 신경망이다.

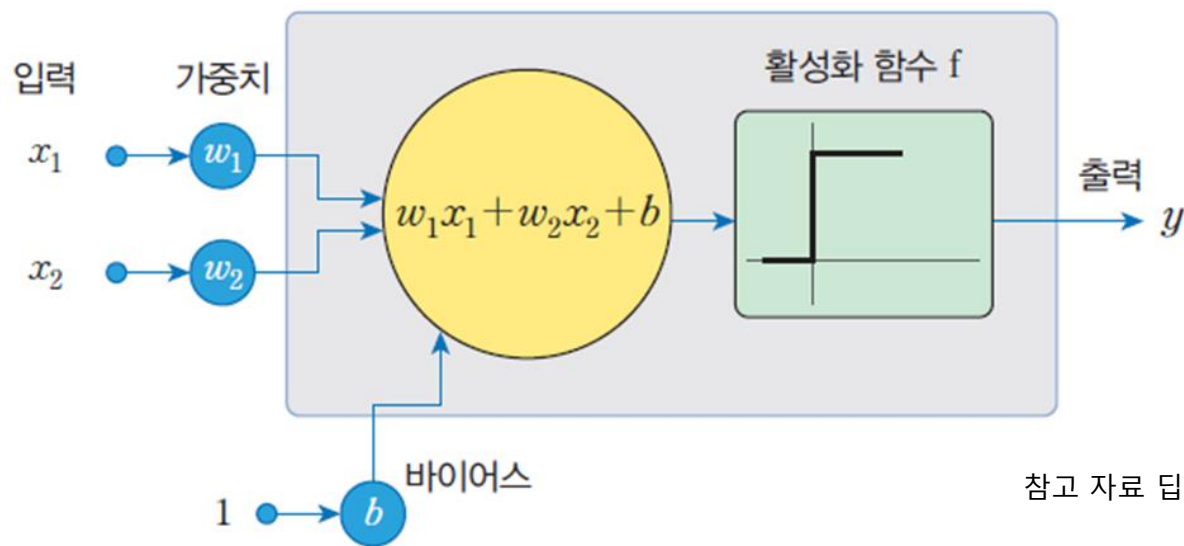


그림 5-5 퍼셉트론에서의 뉴런의 모델

- 다수의 입력 신호를 받아서 하나의 신호를 출력한다.
- 가중치는 입력 신호가 출력에 미치는 **중요도**를 조절
- 바이어스는 뉴런이 얼마나 쉽게 활성화 되는지 결정
- 가중합 : $w_1x_1 + w_2x_2 + b$
- 활성화 함수는 가중합의 결과를 놓고 0, 1을 판단

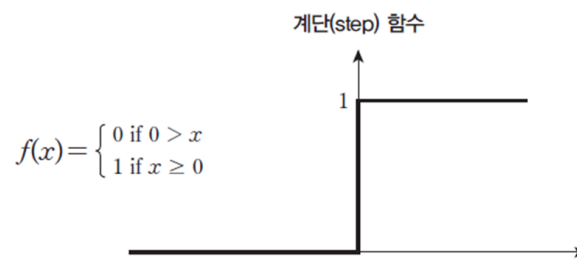


그림 5-7 퍼셉트론에서의 활성화 함수

참고 자료 딥러닝 express

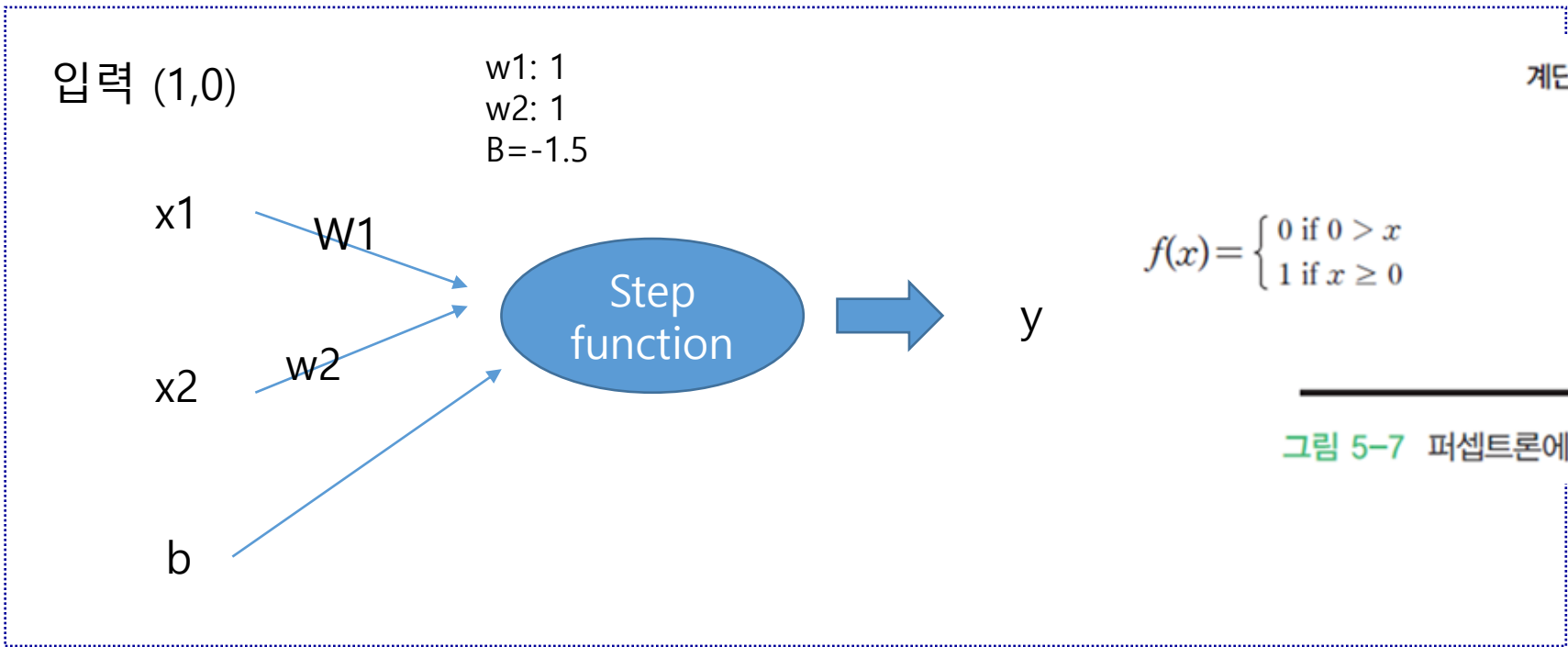
퍼셉트론 (Perceptron)

- 뉴런에서는 입력 신호의 가중치 합이 어떤 임계값을 넘는 경우에만 뉴런이 활성화되어서 1을 출력한다. 그렇지 않으면 0을 출력한다.

$$y = \begin{cases} 1 & \text{if } (w_1x_1 + w_2x_2 + b \geq 0) \\ 0 & \text{otherwise} \end{cases}$$

참고 자료 딥러닝 express

퍼셉트론 (Perceptron)



$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

계단(step) 함수

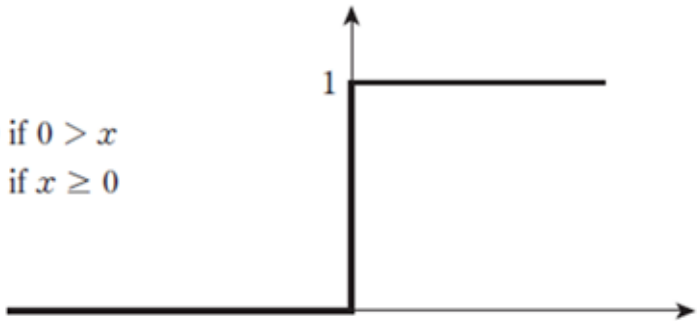
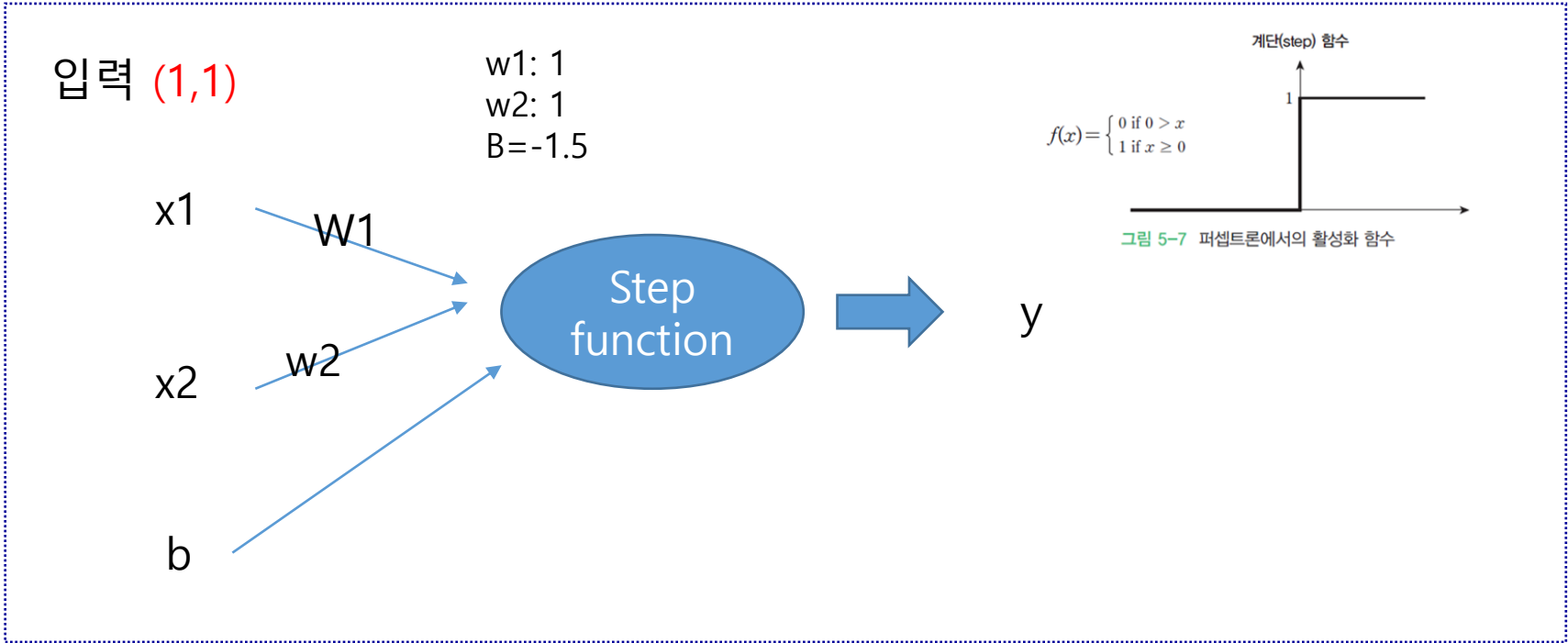


그림 5-7 퍼셉트론에서의 활성화 함수

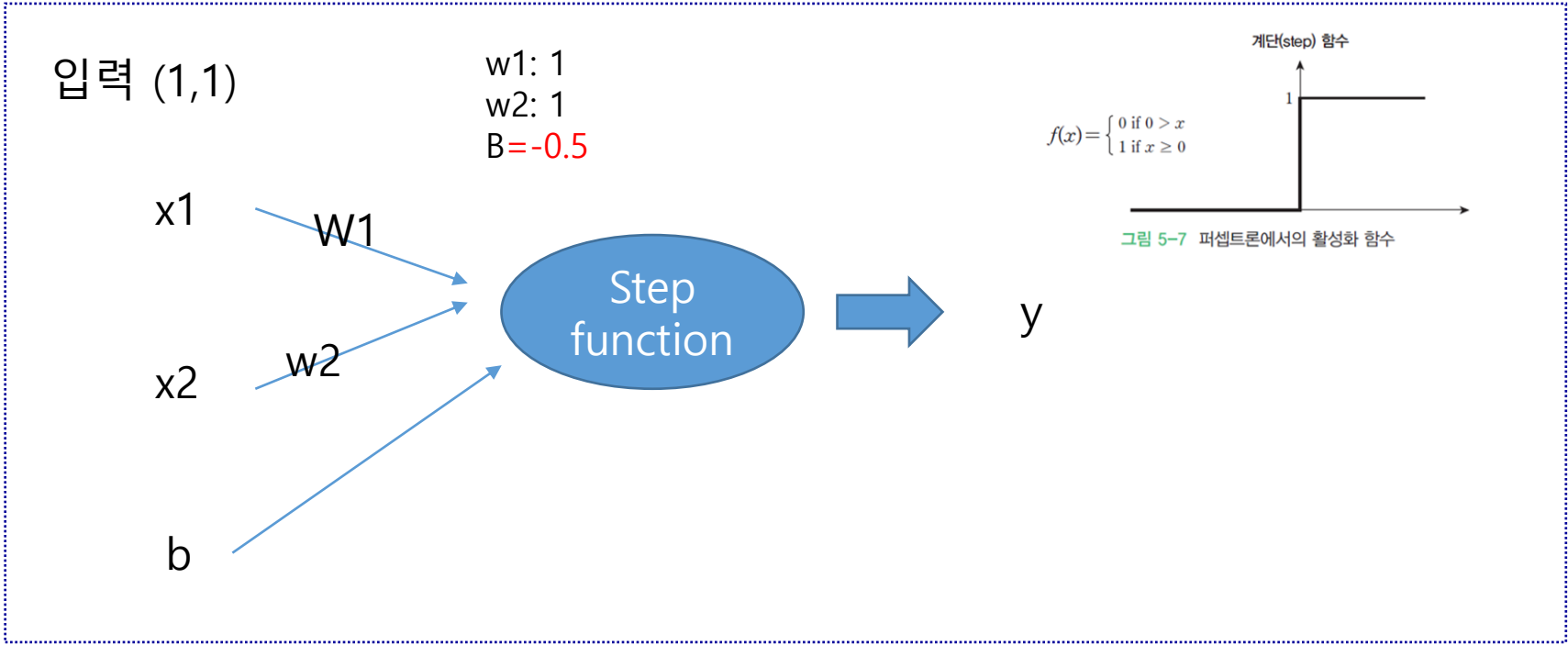
출력값 y 는?

퍼셉트론 (Perceptron)



출력값 y 는?

퍼셉트론 (Perceptron)

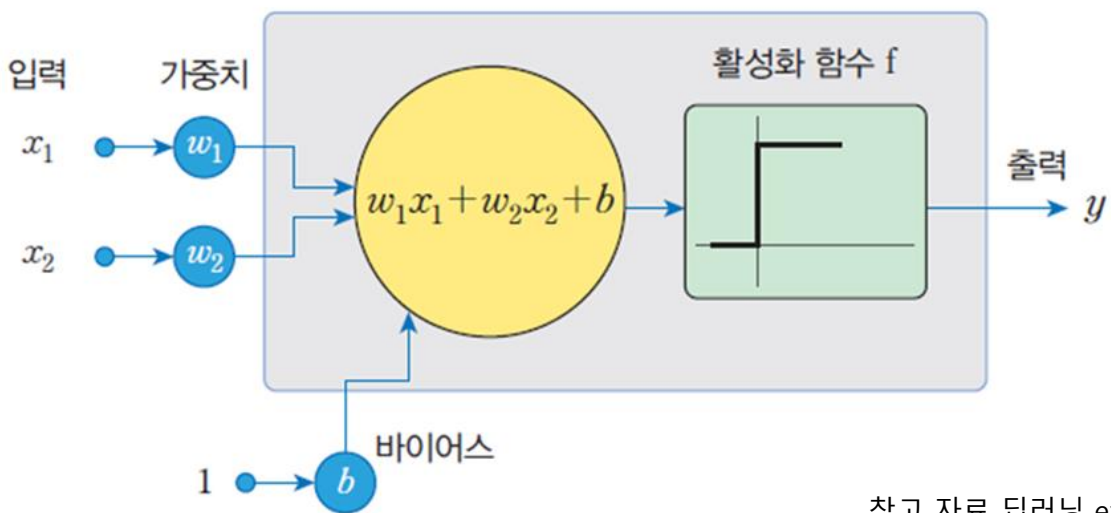


출력값 y 는

퍼셉트론 (Perceptron) – 논리 연산 수행

AND 연산

| x_1 | x_2 | y |
|-------|-------|-----|
| 0 | 0 | 0 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 1 | 1 |



참고 자료 딥러닝 express

그림 5-5 퍼셉트론에서의 뉴런의 모델

만족시키는 w_1, w_2, b 찾기

$w_1=1, w_2=1, b=-1.5$ 를 테스트 해보기

퍼셉트론 (Perceptron) - 논리 연산

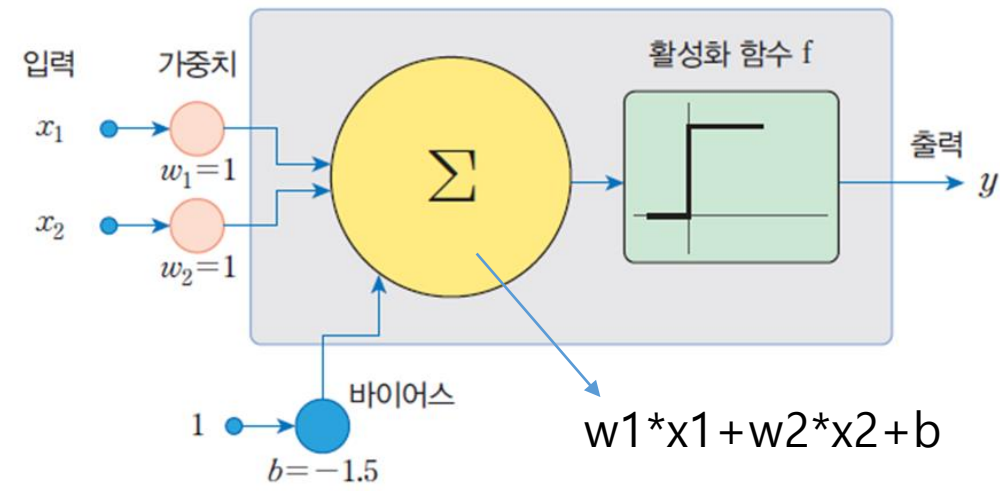


그림 5-6 논리 연산을 하는 퍼셉트론

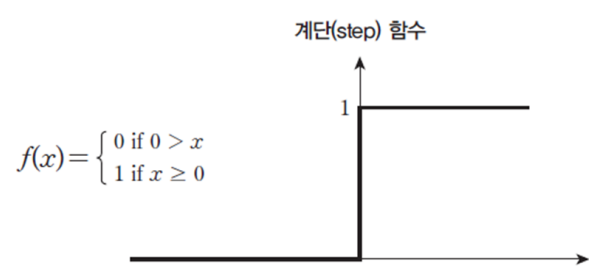


그림 5-7 퍼셉트론에서의 활성화 함수

| x_1 | x_2 |
|-------|-------|
| 0 | 0 |
| 1 | 0 |
| 0 | 1 |
| 1 | 1 |



| 가중치의 합 |
|--------|
| |
| |
| |
| |

| Y |
|---|
| |
| |
| |
| |

참고 자료 딥러닝 express

퍼셉트론 (Perceptron) 구현

```
epsilon = 0.0000001
```

```
def perceptron(x1, x2):
```

```
    w1, w2, b = 1.0, 1.0, -1.5
```

```
    sum = x1*w1+x2*w2+b
```

```
    if sum > epsilon :
```

부동소수점 오차를 방지하기 위하여

```
        return 1
```

```
    else :
```

```
        return 0
```

```
print(perceptron(0, 0))
```

```
print(perceptron(1, 0))
```

```
print(perceptron(0, 1))
```

```
print(perceptron(1, 1))
```

참고 자료 딥러닝 express

w1=1.0, w2=1.0, b=-0.5 일때 출력 구하기

- 1) 입력이 (0,0) 일 때
- 2) 입력이 (1,0) 일 때
- 3) 입력이 (0,1) 일 때
- 4) 입력이 (1,1) 일 때

퍼셉트론 (Perceptron) 학습

- 학습이라고 부르려면 신경망이 스스로 가중치를 자동으로 설정해주는 알고리즘이 필요하다. 퍼셉트론에서도 학습 알고리즘이 존재한다.

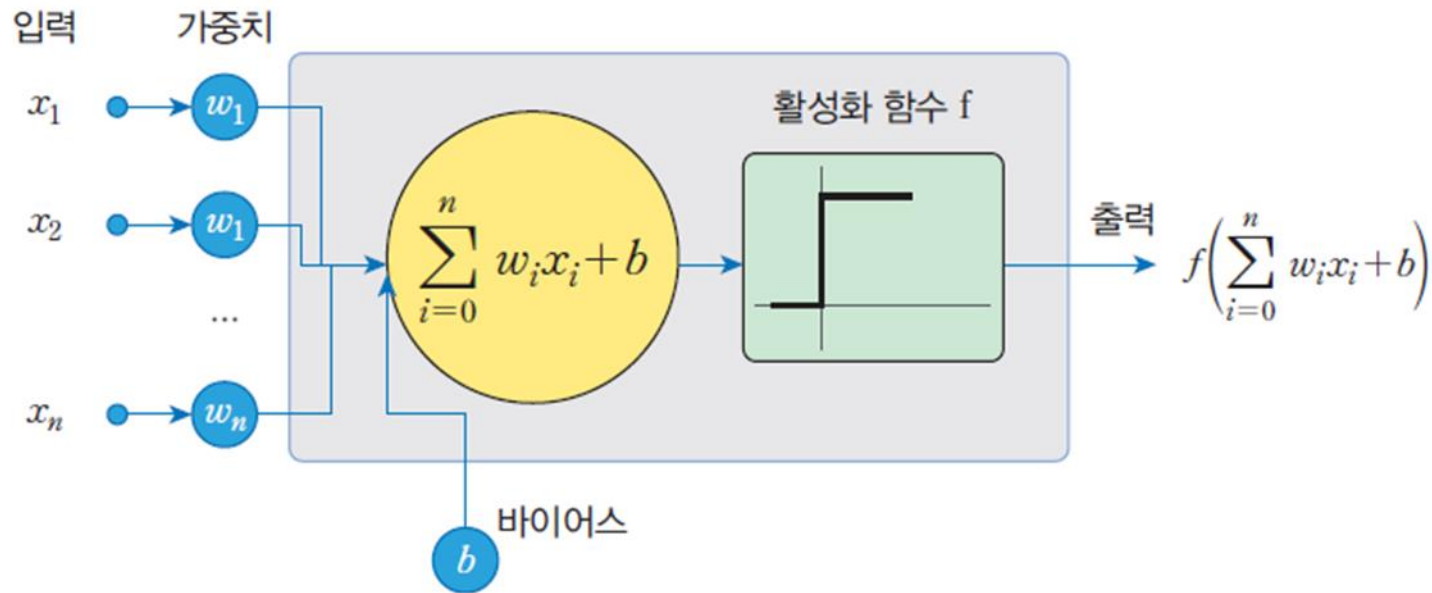


그림 5-8 퍼셉트론

퍼셉트론 (Perceptron) 학습 -scikit learn

- 학습이라고 부르려면 신경망이 스스로 가중치를 자동으로 설정해주는 알고리즘이 필요하다. 퍼셉트론에서도 학습 알고리즘이 존재한다.

```
from sklearn.linear_model import Perceptron

# 샘플과 레이블이다.
X = [[0,0],[0,1],[1,0],[1,1]]
y = [0, 0, 0, 1]

# 퍼셉트론을 생성한다. tol는 종료 조건이다. random_state는 난수의 시드이다.
clf = Perceptron(tol=1e-3, random_state=0)

# 학습을 수행한다.
clf.fit(X, y)

# 테스트를 수행한다.
print(clf.predict(X))
```

참고 자료 딥러닝 express

퍼셉트론 (Perceptron) 학습 OR 연산

| x_1 | x_2 | Y |
|-------|-------|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

OR

SCIKIT LEARN 학습

퍼셉트론 (Perceptron) 학습 OR 연산

| x_1 | x_2 | Y |
|-------|-------|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 1 |

OR

퍼셉트론 (Perceptron) 학습 XOR 연산

| x1 | x2 | y |
|----|----|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

XOR 연산을 수행하는 퍼셉트론 학습

퍼셉트론 (Perceptron) 학습 XOR 연산

| x1 | x2 | y |
|----|----|---|
| 0 | 0 | 0 |
| 1 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 1 | 0 |

XOR 연산을 수행하는 퍼셉트론 학습

퍼셉트론 (Perceptron)

- XOR 연산은 퍼셉트론으로 학습이 불가능 하다
- 퍼셉트론도 $w \cdot x + b$ 의 형태. 직선을 이용한 분류.

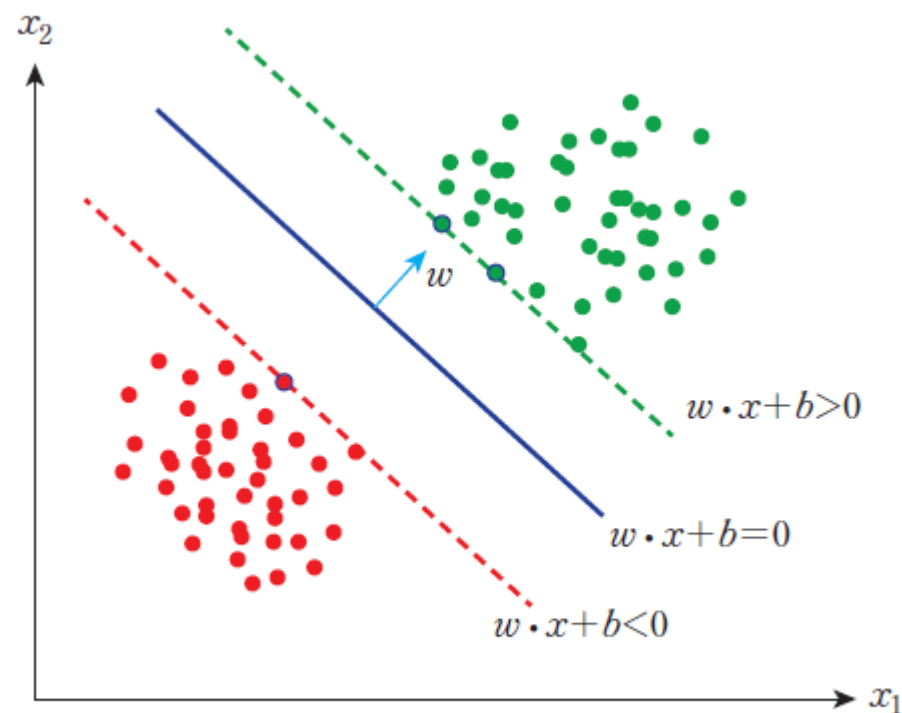


그림 5-10 선형 분류자

퍼셉트론 (Perceptron)

AND 진리표

| x_1 | x_2 | 결괏값 |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

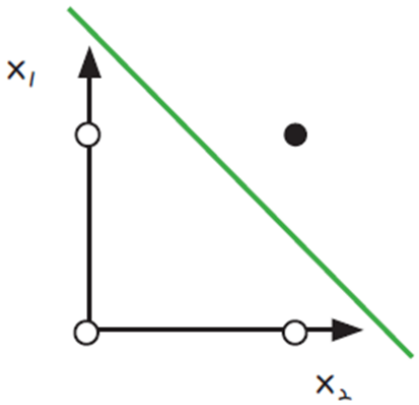
OR 진리표

| x_1 | x_2 | 결괏값 |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

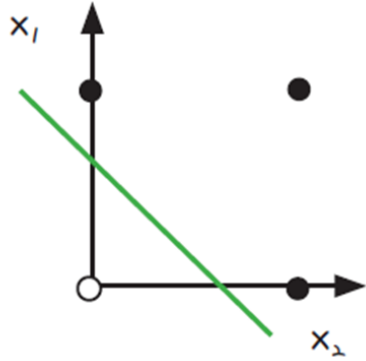
XOR 진리표

| x_1 | x_2 | 결괏값 |
|-------|-------|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

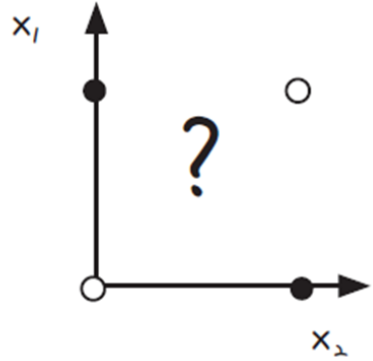
- 결괏값이 0이면 흰점으로, 1이면 검은점으로 나타낸 후 조금 전처럼 직선을 그어 위 조건을 만족할 수 있는지 보자



AND



OR



XOR

그림 6-4 AND, OR, XOR 진리표대로 좌표 평면에 표현한 뒤 선을 그어 색이 같은 점끼리 나누기(XOR은 불가능)

퍼셉트론 (Perceptron)

- AND와 OR 게이트는 직선을 그어 결괏값이 1인 값(검은점)을 구별할 수 있음
- XOR의 경우 선을 그어 구분할 수 없음
- 이는 인공지능 분야의 선구자였던 MIT의 마빈 민스키(Marvin Minsky) 교수가 1969년에 발표한 <퍼셉트론즈(Perceptrons)>라는 논문에서 나오는 내용
 - ➔ '뉴런 → 신경망 → 지능'이라는 도식을 따라 '퍼셉트론 → 인공 신경망 → 인공지능'이 가능하리라 꿈꾸던 당시 사람들은 이것이 생각처럼 쉽지 않다는 사실을 깨닫게 됨
- 알고 보니 간단한 XOR 문제조차 해결할 수 없었던 것임
- 이 논문 이후 인공지능 연구가 한동안 침체기를 겪게 됨
- 10여 년이 지난 후에야 이 문제가 해결되는데, 이를 해결한 개념이 바로 **다층 퍼셉트론**(multilayer perceptron)

행렬 곱셈

$$W=[w_1, w_2], \quad X=[x_1, x_2]$$

$$WX=w_1*x_1+w_2*x_2$$

$$W= \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} \quad X= \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$WX= \begin{bmatrix} w_1x_1 + & w_2x_2 \\ w_3x_1 + & w_4x_2 \end{bmatrix}$$

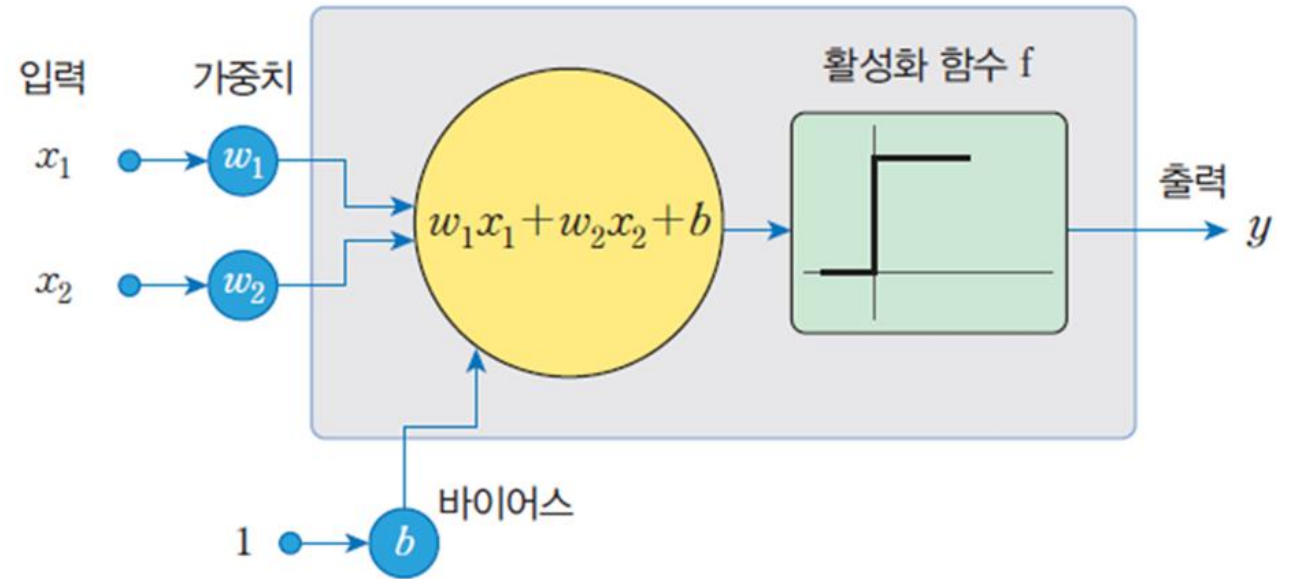


그림 5-5 퍼셉트론에서의 뉴런의 모델

$$w_1*x_1+w_2*x_2+b \Rightarrow WX+b$$

활성화 함수

- 활성화 함수(activation function)은 입력의 총합을 받아서 출력값을 계산하는 함수이다
- MLP에서는 다양한 활성화 함수를 사용한다.

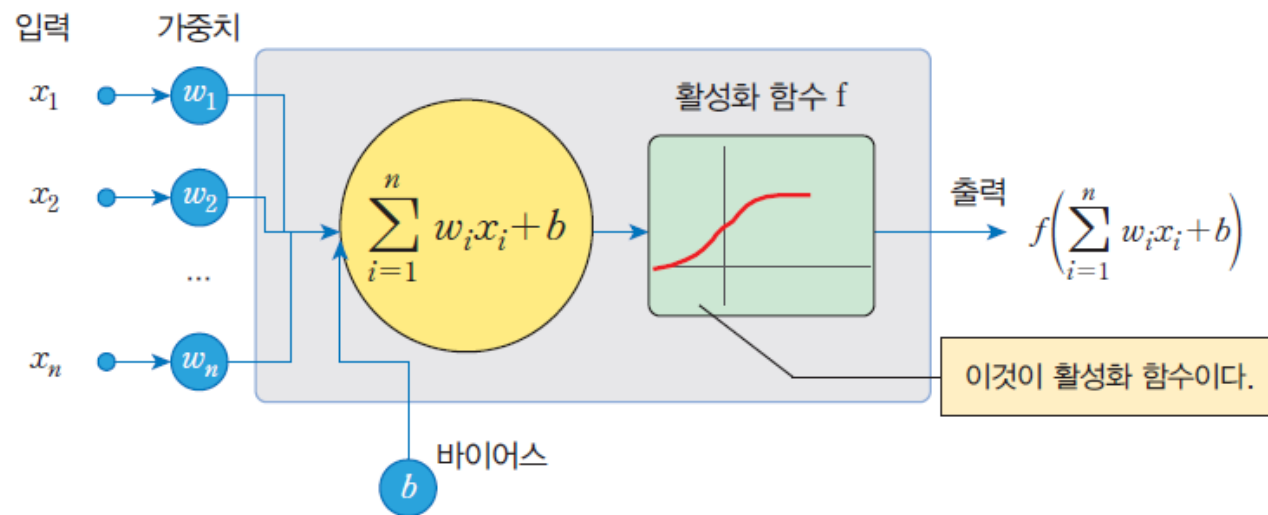


그림 6-2 활성화 함수

활성화 함수

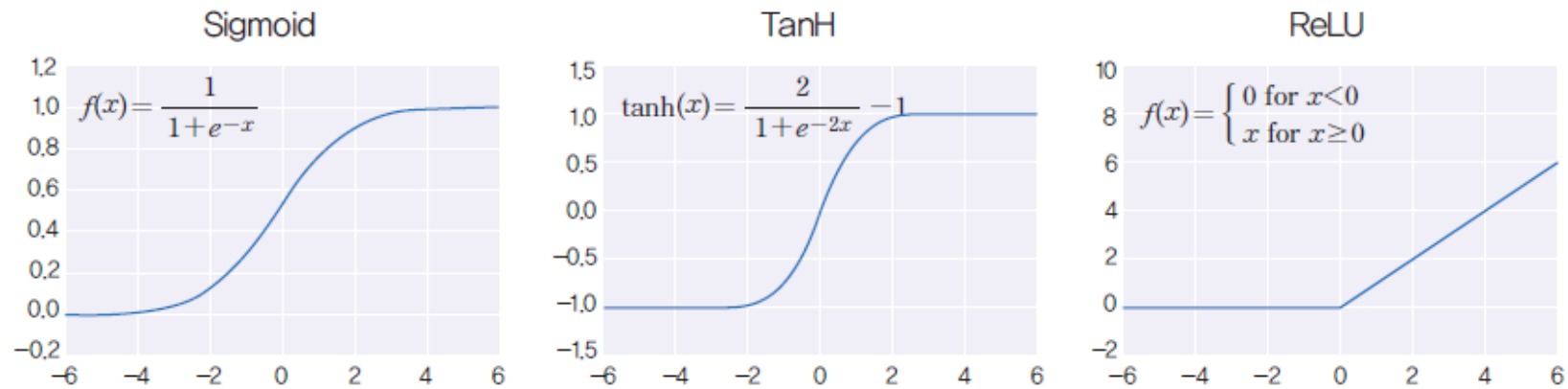


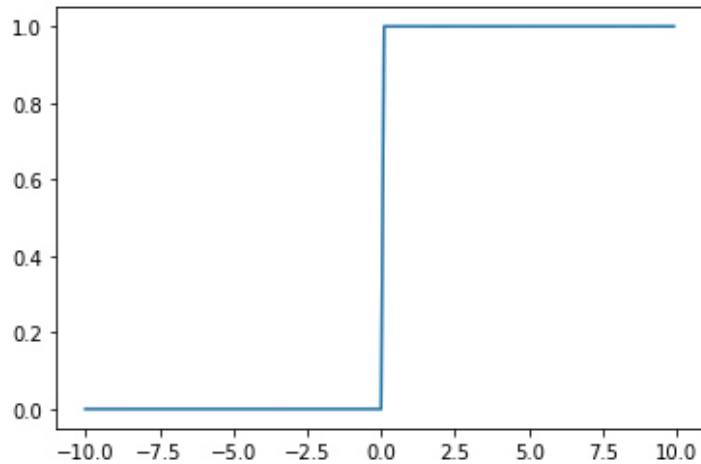
그림 6-3 많이 사용되는 활성화 함수

참고 자료 딥러닝 express

활성화 함수 (step)

- 계단 함수는 입력 신호의 총합이 0을 넘으면 1을 출력하고, 그렇지 않으면 0을 출력하는 함수이다.

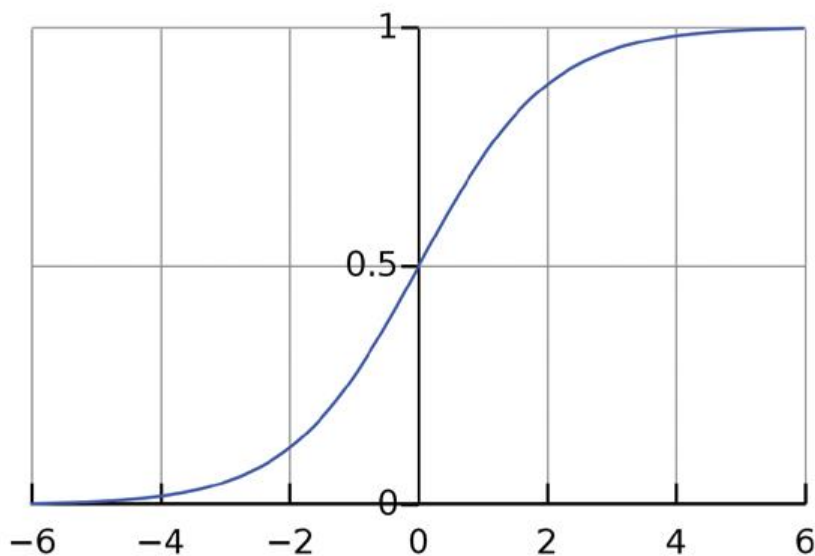
$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



활성화 함수 (시그모이드)

- 1980년대부터 사용돼온 전통적인 활성화 함수이다. 시그모이드는 다음과 같이 s자와 같은 형태를 가진다.

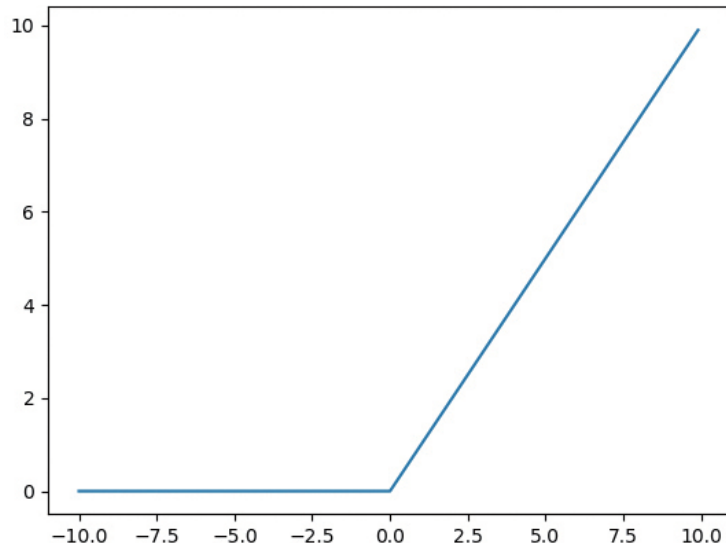
$$f(x) = \frac{1}{1 + e^{-x}}$$



활성화 함수 (Rectified Linear Unit function)

- ReLU 함수는 최근에 가장 인기 있는 활성화 함수이다. ReLU 함수는 입력이 0을 넘으면 그대로 출력하고, 입력이 0보다 적으면 출력은 0이 된다.

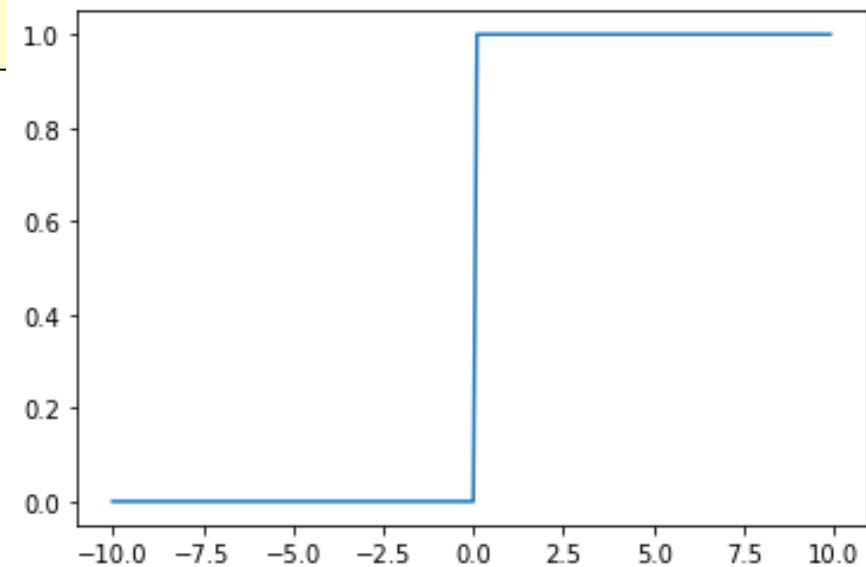
$$f(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$



Step 함수

```
def step(x):  
    result = x > 0.000001          # True 또는 False  
    return result.astype(np.int)    # 정수로 반환
```

```
import numpy as np  
import matplotlib.pyplot as plt  
  
x = np.arange(-10.0, 10.0, 0.1)  
y = step(x)  
plt.plot(x, y)  
plt.show()
```

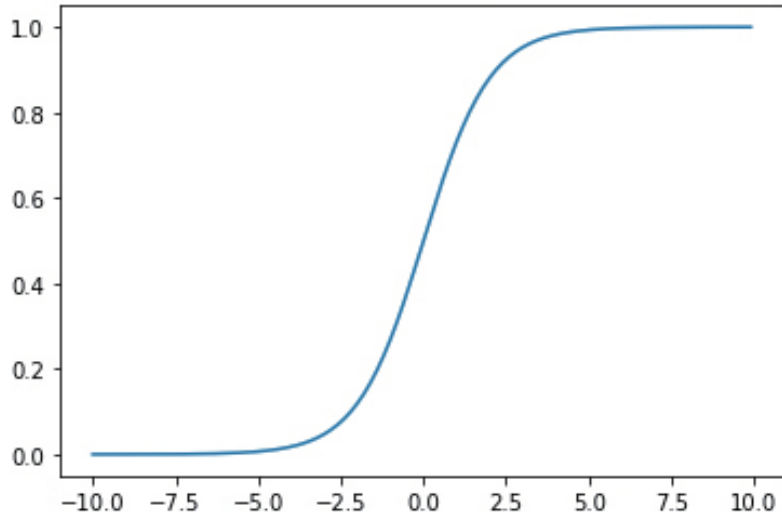


시그모이드 함수

```
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1.0 / (1.0 + np.exp(-x))

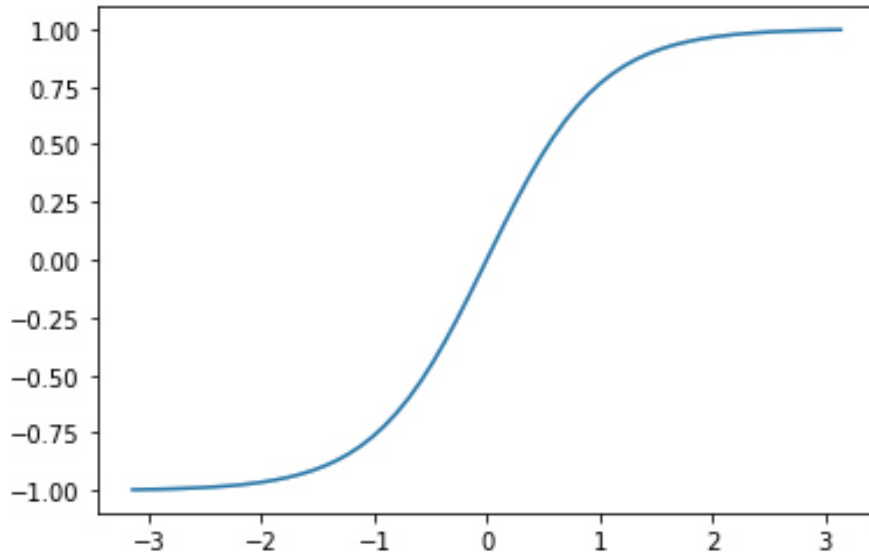
x = np.arange(-10.0, 10.0, 0.1)
y = sigmoid(x)
plt.plot(x, y)
plt.show()
```



Tanh 함수

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-np.pi, np.pi, 60)
y = np.tanh(x)
plt.plot(x, y)
plt.show()
```

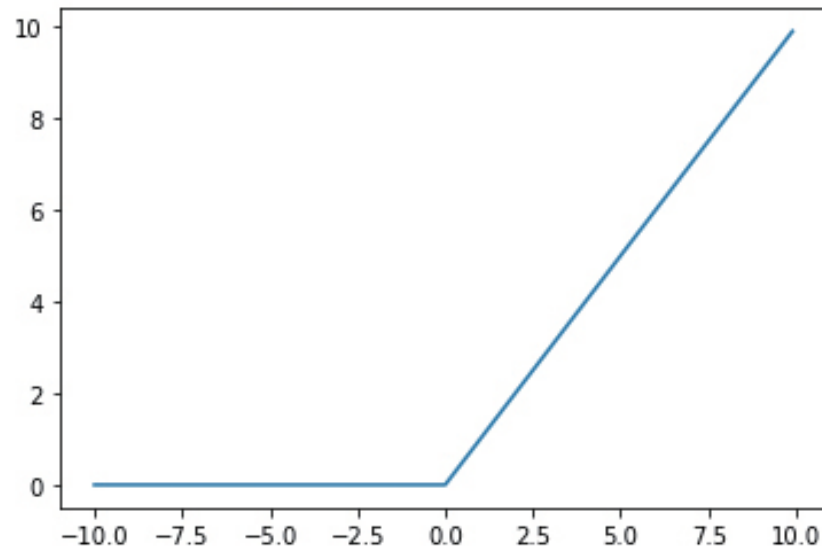


relu 함수

```
import numpy as np
import matplotlib.pyplot as plt

def relu(x):
    return np.maximum(x, 0)

x = np.arange(-10.0, 10.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.show()
```



다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 종이 위에 각각 엇갈려 놓인 검은점 두 개와 흰점 두 개를 하나의 선으로는 구별할 수 없다는 것을 살펴봄
- 언뜻 보기에 해답이 없어 보이는 이 문제를 해결하려면 새로운 접근이 필요함

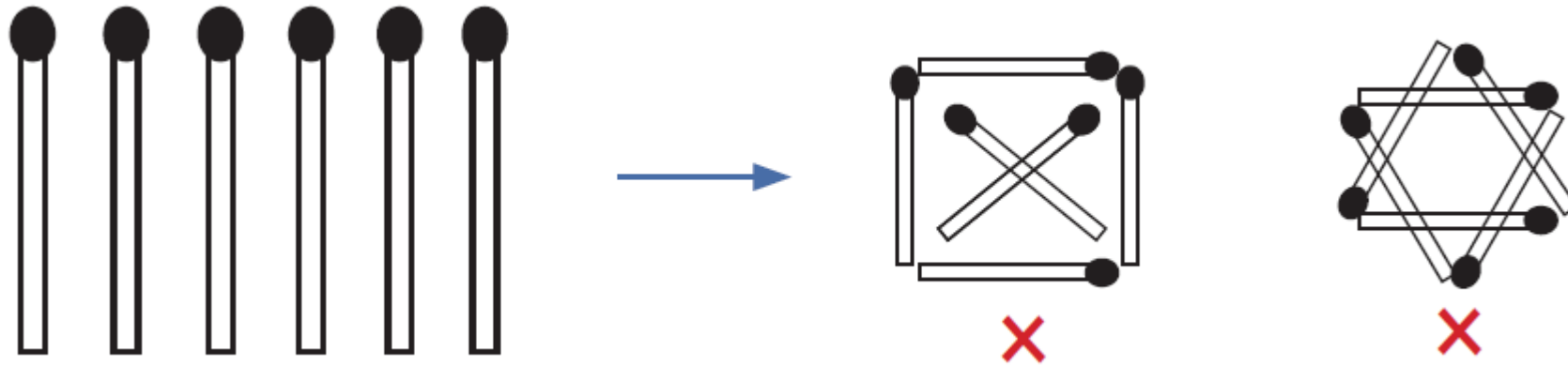


그림 7-1 성냥개비 여섯 개로 정삼각형 네 개를?

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 골똥히 연구해도 답을 찾지 못했던 이 문제는 2차원 평면에서만 해결하려는 고정 관념을 깨고 피라미드 모양으로 성냥개비를 쌓아 올리니 해결됨

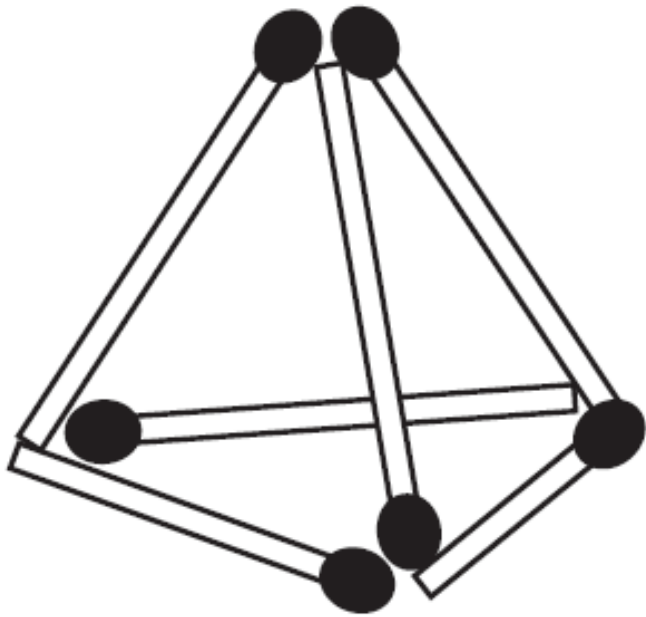


그림 7-2 차원을 달리하니 쉽게 완성!

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 인공지능 학자들은 인공 신경망을 개발하기 위해서 반드시 XOR 문제를 극복해야만 했음
- 이 문제 역시 고정관념을 깬 기발한 아이디어에서 해결점이 보였음

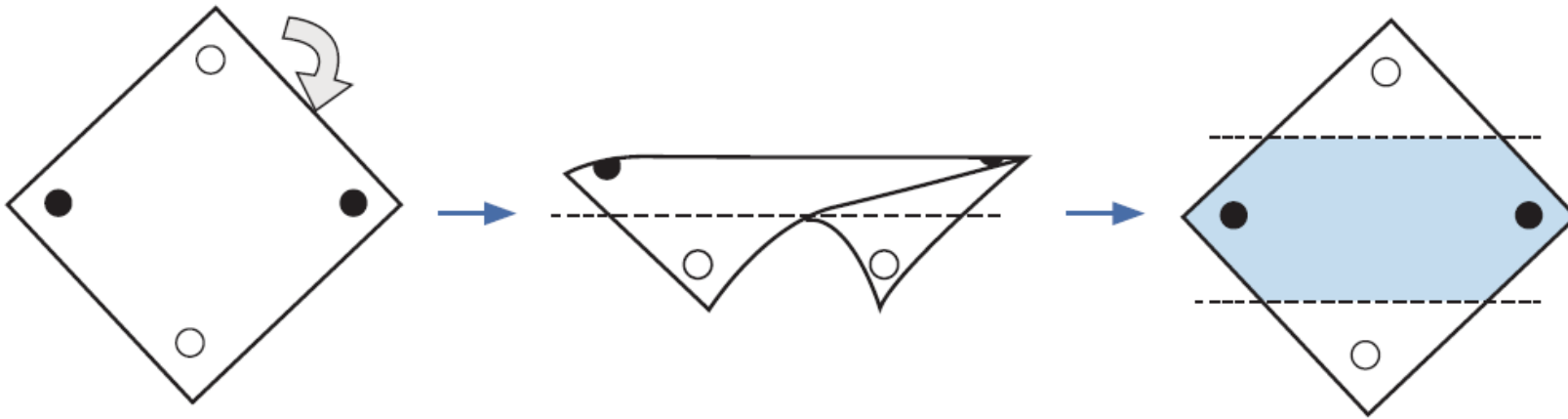


그림 7-3 XOR 문제의 해결은 평면을 휘어주는 것!

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 좌표 평면 자체에 변화를 주는 것
- XOR 문제를 해결하기 위해서 우리는 두 개의 퍼셉트론을 한 번에 계산할 수 있어야 함
- 이를 가능하게 하려면 숨어있는 층, 즉 은닉층(hidden layer)을 만들면 됨

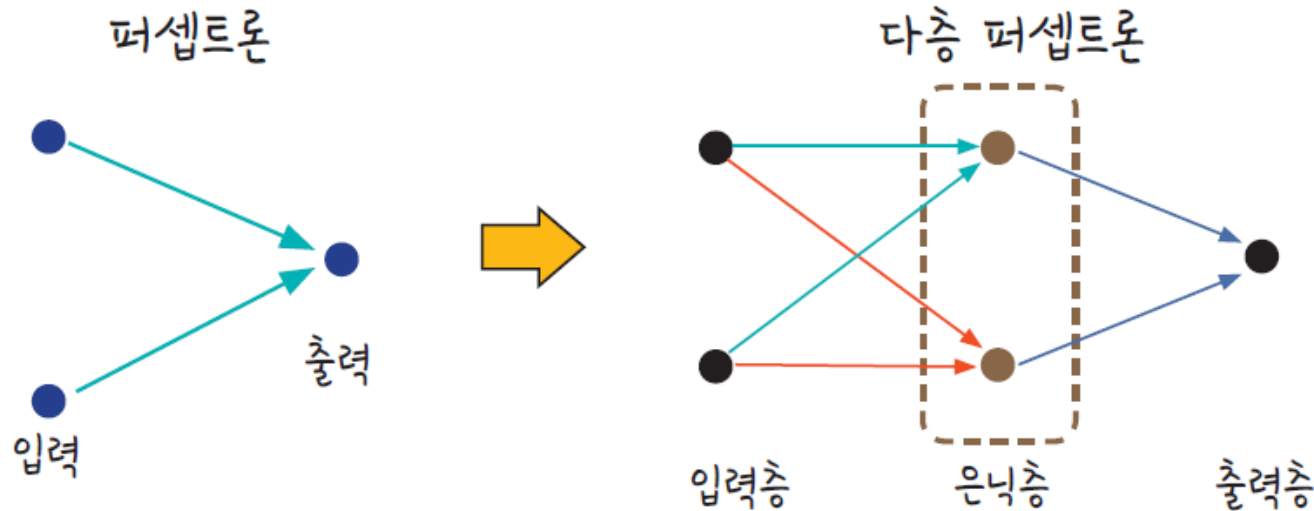


그림 7-4 퍼셉트론에서 다중 퍼셉트론으로

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

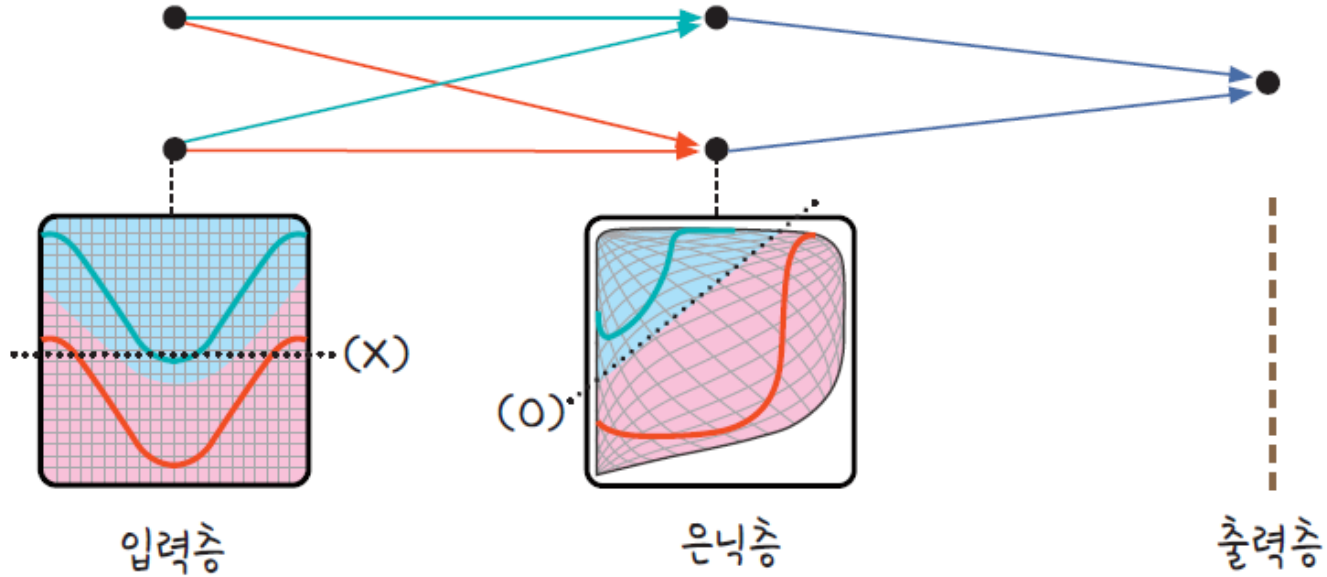


그림 7-5 은닉층의 공간 왜곡(<https://goo.gl/8qEGHD> 참조)

- 입력 값(input)을 놓고 파란색과 빨간색의 영역을 구분한다고 할 때, 그림 7-5의 왼쪽 그림을 보면 어떤 직선으로도 이를 해결할 수 없음
- 은닉층을 만들어 공간을 왜곡하면 두 영역을 가로지르는 선이 직선으로 바뀜

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 다중 퍼셉트론이 입력층과 출력층 사이에 숨어있는 은닉층을 만드는 것을
도식으로 나타내면 그림 7-6과 같음

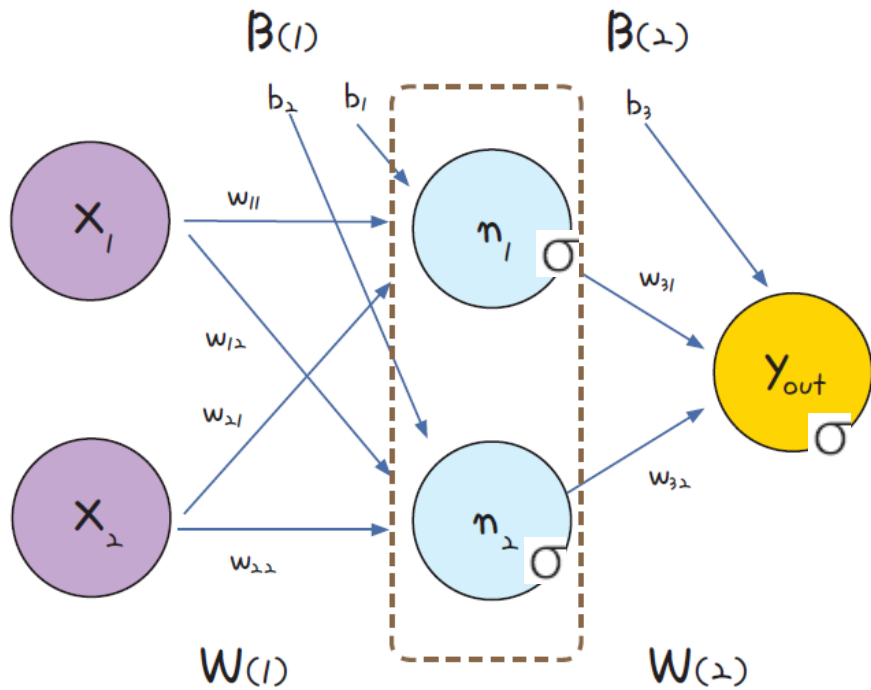


그림 7-6 다중 퍼셉트론의 내부

다중 퍼셉트론 설계

- 가운데 숨어있는 은닉층으로 퍼셉트론이 각각 자신의 가중치(w)와 바이어스(b) 값을 보냄
- 이 은닉층에서 모인 값이 한 번 더 시그모이드 함수(기호로 σ 라고 표시함)를 이용해 최종 값으로 결과를 보냄
- 노드(node) :
은닉층에 모이는 중간 정거장
여기서는 n_1 과 n_2 로 표현함

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- n_1 과 n_2 의 값은 각각 단일 퍼셉트론의 값과 같음

$$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$$

$$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$$

- 위 두 식의 결과값이 출력층으로 보내짐
- 출력층에서는 역시 시그모이드 함수를 통해 y 값이 정해짐
- 이 값을 y_{out} 이라 할 때 식으로 표현하면 다음과 같음

$$y_{\text{out}} = \sigma(n_1 w_{31} + n_2 w_{32} + b_3)$$

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 각각의 가중치(w)와 바이어스(b)의 값을 정할 차례임
- 2차원 배열로 늘어놓으면 다음과 같이 표시할 수 있음
- 은닉층을 포함해 가중치 6개와 바이어스 3개가 필요함

$$W(1) = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad B(1) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \quad B(2) = [b_3]$$

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

2 | XOR 문제의 해결

- 각 변수값을 정하고 이를 이용해 XOR 문제를 해결하는 과정을 알아보자

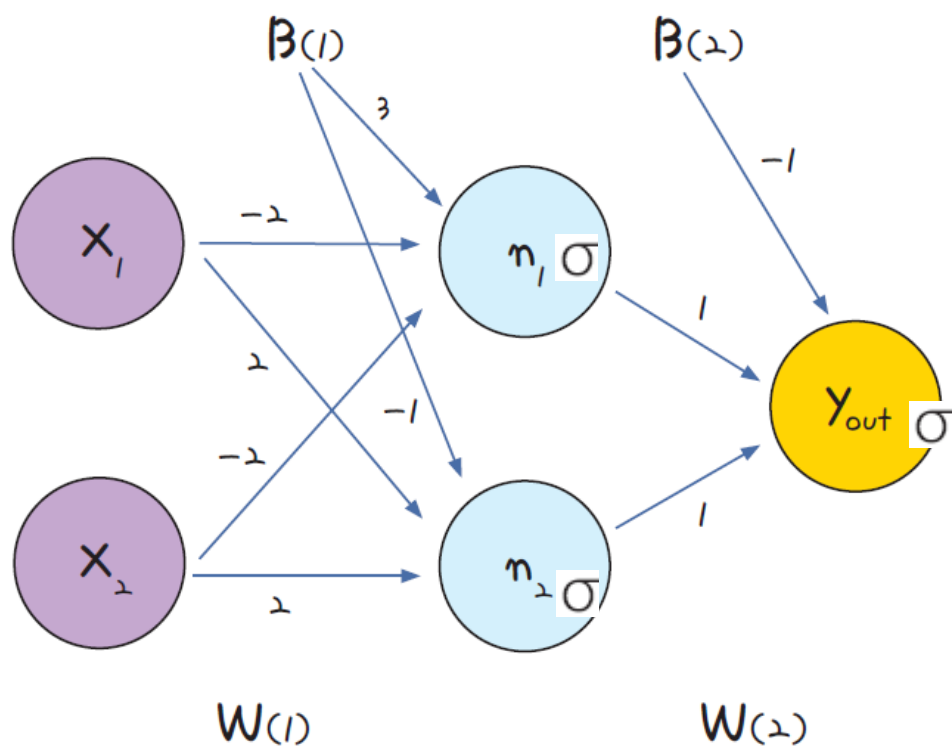
$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

2 | XOR 문제의 해결

- 이것을 도식에 대입하면 다음과 같음



$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$
$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$

그림 7-7 다중 퍼셉트론의 내부에 변수를 채워보자.

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

2 | XOR 문제의 해결

- 이제 X_1 의 값과 X_2 의 값을 각각 입력해 y 값이 우리가 원하는 값으로 나오는지를 점검해 보자

| x_1 | x_2 | n_1 | n_2 | y_{out} | 우리가 원하는 값 |
|-------|-------|---|---------------------------------------|---------------------------------------|-----------|
| 0 | 0 | $\sigma(0 * (-2) + 0 * (-2) + 3) \approx 1$ | $\sigma(0 * 2 + 0 * 2 - 1) \approx 0$ | $\sigma(1 * 1 + 0 * 1 - 1) \approx 0$ | 0 |
| 0 | 1 | $\sigma(0 * (-2) + 1 * (-2) + 3) \approx 1$ | $\sigma(0 * 2 + 1 * 2 - 1) \approx 1$ | $\sigma(1 * 1 + 1 * 1 - 1) \approx 1$ | 1 |
| 1 | 0 | $\sigma(1 * (-2) + 0 * (-2) + 3) \approx 1$ | $\sigma(1 * 2 + 0 * 2 - 1) \approx 1$ | $\sigma(1 * 1 + 1 * 1 - 1) \approx 1$ | 1 |
| 1 | 1 | $\sigma(1 * (-2) + 1 * (-2) + 3) \approx 0$ | $\sigma(1 * 2 + 1 * 2 - 1) \approx 1$ | $\sigma(0 * 1 + 1 * 1 - 1) \approx 0$ | 0 |

표 7-1 XOR 다층 문제 해결



\approx 기호는 '거의 같다'를 의미하는 것으로 이해하면 됩니다.

다중 퍼셉트론 (Multi Layer Perceptron: MLP)

| x_1 | x_2 | n_1 | n_2 | y_{out} | 우리가 원하는 값 |
|-------|-------|---|---------------------------------------|---------------------------------------|-----------|
| 0 | 0 | $\sigma(0 * (-2) + 0 * (-2) + 3) \approx 1$ | $\sigma(0 * 2 + 0 * 2 - 1) \approx 0$ | $\sigma(1 * 1 + 0 * 1 - 1) \approx 0$ | 0 |
| 0 | 1 | $\sigma(0 * (-2) + 1 * (-2) + 3) \approx 1$ | $\sigma(0 * 2 + 1 * 2 - 1) \approx 1$ | $\sigma(1 * 1 + 1 * 1 - 1) \approx 1$ | 1 |
| 1 | 0 | $\sigma(1 * (-2) + 0 * (-2) + 3) \approx 1$ | $\sigma(1 * 2 + 0 * 2 - 1) \approx 1$ | $\sigma(1 * 1 + 1 * 1 - 1) \approx 1$ | 1 |
| 1 | 1 | $\sigma(1 * (-2) + 1 * (-2) + 3) \approx 0$ | $\sigma(1 * 2 + 1 * 2 - 1) \approx 1$ | $\sigma(0 * 1 + 1 * 1 - 1) \approx 0$ | 0 |

x_1 **NAND** $x_2 \Rightarrow n_1$

x_1 **OR** $x_2 \Rightarrow n_2$

n_1 **AND** $n_2 \Rightarrow y_{out}$

NAND

| X | Y | F |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

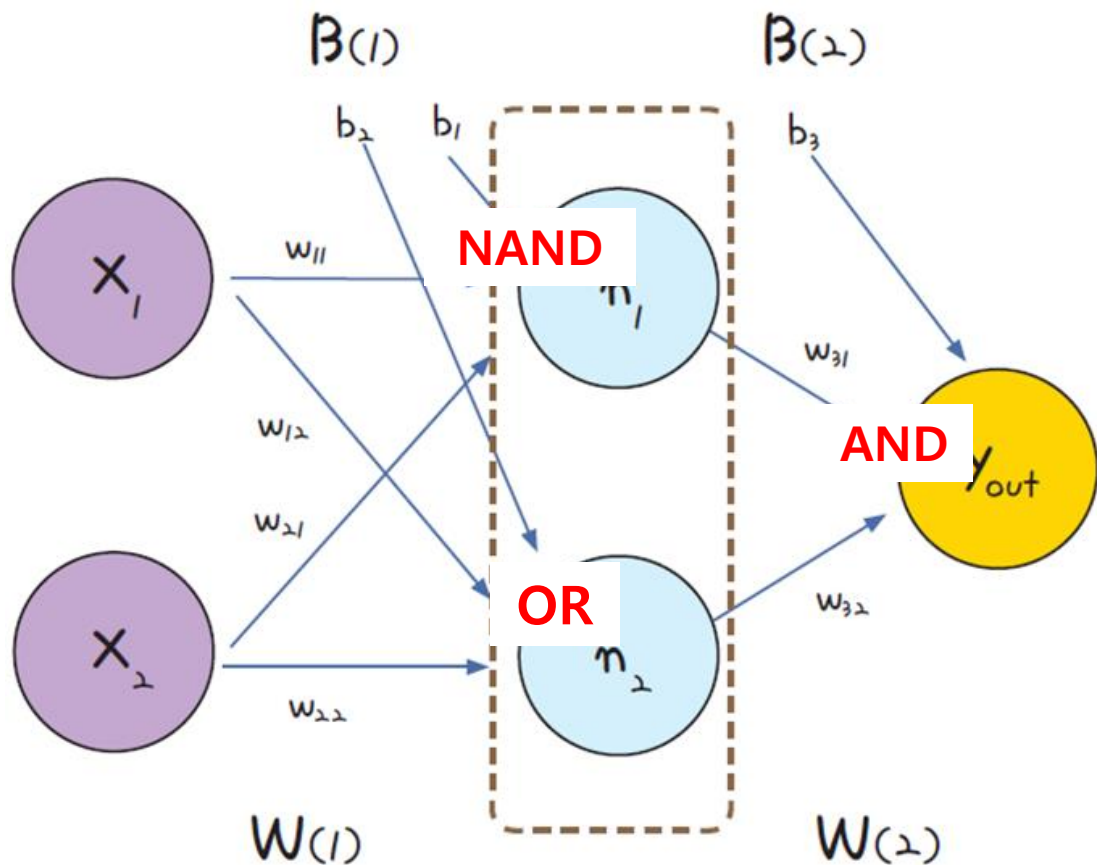
OR

| X | Y | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

AND

| X | Y | F |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

다중 퍼셉트론 (Multi Layer Perceptron: MLP)



다중 퍼셉트론 (Multi Layer Perceptron: MLP)

다중 퍼셉트론으로 XOR 문제 해결

- 정해진 가중치와 바이어스를 numpy라이브러리를 사용해 다음과 같이 선언함

```
import numpy as np
```

```
w11 = np.array([-2, -2])
```

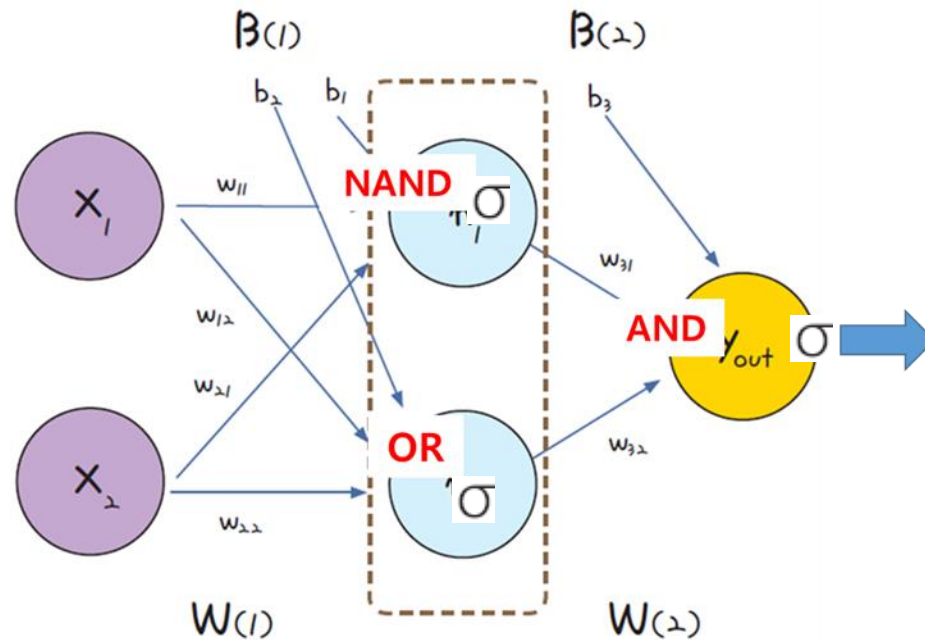
```
w12 = np.array([2, 2])
```

```
w2 = np.array([1, 1])
```

```
b1 = 3
```

```
b2 = -1
```

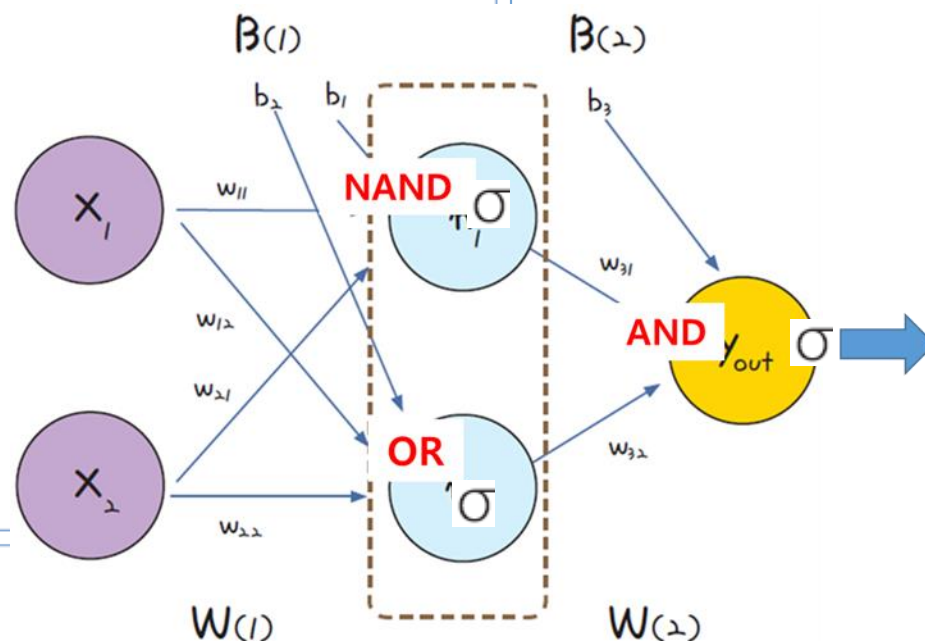
```
b3 = -1
```



다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 이제 퍼셉트론 함수를 만들어 줌
- 0과 1 중에서 값을 출력하게 설정함

```
def MLP(x, w, b):  
    y = np.sum(w * x) + b  
    if y <= 0:  
        return 0  
    else:  
        return 1
```



다중 퍼셉트론 (Multi Layer Perceptron: MLP)

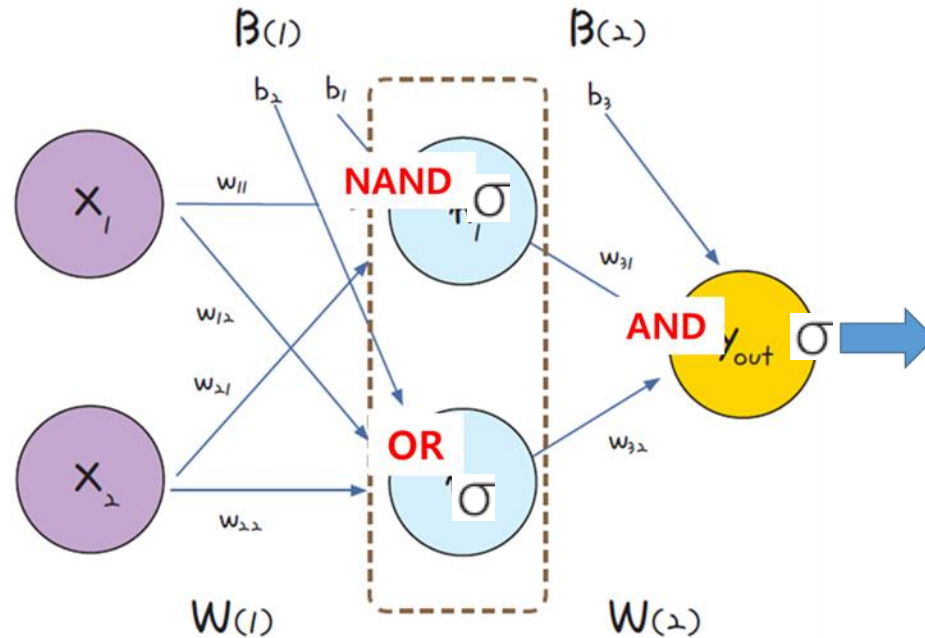
- 각 게이트의 정의에 따라 NAND 게이트, OR 게이트, AND 게이트, XOR 게이트 함수를 만들어 줌

NAND 게이트

```
def NAND(x1, x2):  
    return MLP(np.array([x1, x2]), w11, b1)
```

OR 게이트

```
def OR(x1, x2):  
    return MLP(np.array([x1, x2]), w12, b2)
```



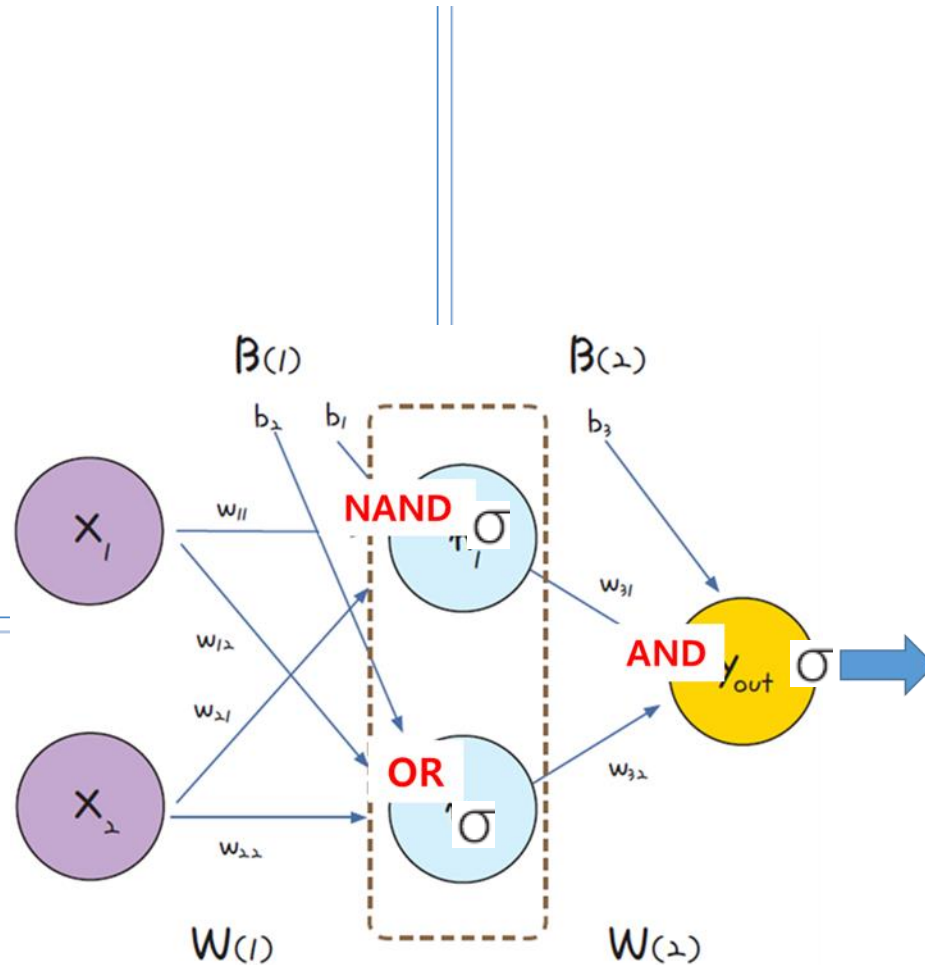
다중 퍼셉트론 (Multi Layer Perceptron: MLP)

AND 게이트

```
def AND(x1, x2):  
    return MLP(np.array([x1, x2]), w2, b3)
```

XOR 게이트

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



다중 퍼셉트론 (Multi Layer Perceptron: MLP)

- 이제 x_1 과 x_2 값을 번갈아 대입해 가며 최종 값을 출력해 보자

```
if __name__ == '__main__':  
    for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:  
        y = XOR(x[0], x[1])  
        print("입력 값: " + str(x) + " 출력 값: " + str(y))
```

NAND,OR,AND 기능을 하는
퍼셉트론이 연결되어

XOR 기능을 수행

실행
결과



입력 값: (0, 0) 출력 값: 0

입력 값: (1, 0) 출력 값: 1

입력 값: (0, 1) 출력 값: 1

입력 값: (1, 1) 출력 값: 0

수고 하셨습니다



jhmin@inhatec.ac.kr