



# Day57; 20221125

📅 날짜	@2022년 11월 25일
👤 유형	@2022년 11월 25일
☰ 태그	

GitHub - u8yes/AI

You can't perform that action at this time. You signed in with another tab or window. You signed out in another tab or window. Reload to refresh your session. Reload to refresh your session.

<http://github.com/u8yes/AI>

u8yes/AI



A 1 Contributor 0 Issues 0 Stars 0 Forks

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4391fb71-f00b-4022-99d5-be38f42a0c71/05\\_%ED%8D%BC%EC%85%89%ED%8A%B8%EB%A1%A0.pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/4391fb71-f00b-4022-99d5-be38f42a0c71/05_%ED%8D%BC%EC%85%89%ED%8A%B8%EB%A1%A0.pdf)

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9c31d61a-00a4-42c7-8278-225a663605dc/06\\_%EC%8B%A0%EA%B2%BD%EB%A7%9D\(%EB%94%A5%EB%9F%AC%EB%8B%9D\).pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/9c31d61a-00a4-42c7-8278-225a663605dc/06_%EC%8B%A0%EA%B2%BD%EB%A7%9D(%EB%94%A5%EB%9F%AC%EB%8B%9D).pdf)

## 혼·하·파이썬)

append() 함수와 insert() 함수는 리스트에 요소 하나를 추가합니다. 한 번에 여러 요소를 추가하고 싶을 때는 **extend()** 함수를 사용합니다. extend() 함수는 매개변수로 리스트를 입력하는데, 원래 리스트 뒤에 새로운 리스트의 요소를 모두 추가해 줍니다.

```
>>> list_a = [1, 2, 3]
>>> list_a.extend([4, 5, 6])
>>> print(list_a)          ← 매개변수를 리스트 형태로 넣습니다.
[1, 2, 3, 4, 5, 6]
```

extend() 함수는 마치 append() 함수를 세 번 반복 실행한 효과를 가져옵니다.

리스트 연결 연산자는 연산 결과로 [1, 2, 3, 4, 5, 6]을 출력하고 있으며, 원본에 어떠한 변화도 없다는 것을 확인할 수 있습니다. 이어서 extend() 함수를 사용해 보겠습니다.

```
>>> list_a = [1, 2, 3]
>>> list_b = [4, 5, 6]
>>> list_a.extend(list_b)  → 실제로 결과로 아무 것도 출력하지 않았습니다.
>>> list_a
```

[1, 2, 3, 4, 5, 6] → 앞에 입력했던 list\_a 자체에 직접적인 변화가 있습니다(파고적 처리).  
>>> list\_b  
[4, 5, 6]

## 인덱스로 제거하기: del 키워드, pop()

인덱스로 제거한다는 것은 ‘리스트의 2번째 요소를 제거해 주세요’처럼 요소의 위치를 기반으로 요소를 제거하는 것입니다. **del 키워드** 또는 **pop()** 함수를 사용합니다. **del** 키워드는 다음과 같은 구문을 사용하여 리스트의 특정 인덱스에 있는 요소를 제거합니다.

**del** 리스트명[인덱스]

**pop()** 함수 또한 제거할 위치에 있는 요소를 제거하는데, 매개변수를 입력하지 않으면 -1이 들어가는 것으로 취급해서 마지막 요소를 제거합니다.

리스트명.pop(인덱스)

직접 해보는 손코딩

리스트 요소 하나 제거하기 [소스 코드 list03.py](#)

```
01 list_a = [0, 1, 2, 3, 4, 5]
02 print("# 리스트의 요소 하나 제거하기")
03
04 # 제거 방법[1] – del 키워드
05 del list_a[1]
06 print("del list_a[1]:", list_a)
07
08 # 제거 방법[2] – pop()
09 list_a.pop(2)
10 print("pop(2):", list_a)
```

실행 결과  
# 리스트의 요소 하나 제거하기  
del list\_a[1]: [0, 2, 3, 4, 5]  
pop(2): [0, 2, 4, 5]

```
>>> list_d = [0, 1, 2, 3, 4, 5, 6]
>>> del list_d[3:]
>>> list_d
[0, 1, 2]
```

간단하게 사용해 볼까요?

```
>>> list_c = [1, 2, 1, 2]      # 리스트 선언하기  
>>> list_c.remove(2)          # 리스트의 요소를 값으로 제거하기  
>>> list_c  
[1, 1, 2]
```

remove() 함수로 지정한 값이 리스트 내부에 여러 개 있어도 가장 먼저 발견되는 하나만 제거합니다. 위의 예제를 보면 리스트에 2가 두 개 있는데 앞쪽에 있는 2 하나만 제거되는 것을 알 수 있습니다.

만약 리스트에 중복된 여러 개의 값을 모두 제거하려면 반복문과 조합해서 사용해야 합니다.

note 리스트에 중복된 여러 개의 값을 제거하는 방법은 while 반복문을 살펴볼 때 알아보겠습니다(240쪽).

### 리스트.clear()

간단한 함수이므로 쉽게 이해할 수 있습니다.

```
>>> list_d = [0, 1, 2, 3, 4, 5]  
>>> list_d.clear()  
>>> list_d  
[] → 요소가 모두 제거되었습니다.
```

```
>>> list_e = [52, 273, 103, 32, 275, 1, 7]  
>>> list_e.sort() # 오름차순 정렬  
>>> list_e  
[1, 7, 32, 52, 103, 273, 275]  
>>> list_e.sort(reverse=True) # 내림차순 정렬 → 05-1 항수 만들기: 키워드 매개변수 "에서  
                                            자세하게 다루는 문법입니다.  
>>> list_e  
[275, 273, 103, 52, 32, 7, 1]
```

## + 여기서 잠깐

## for 반복문과 문자열

for 반복문은 문자열을 함께 사용할 수도 있습니다. 문자열을 for 반복문의 뒤에 넣으면 글자 하나하나에 반복이 적용됩니다. 실행 결과를 보면 어떤 식으로 실행되는지 쉽게 이해할 수 있을 것입니다.

```
for character in "안녕하세요":  
    print("-", character)
```

실행 결과

- 안
- 녕
- 하
- 세
- 요

## 05\_linear\_regression\_without\_min\_max

```
[]: import tensorflow as tf  
import numpy as np  
  
# open, higher, volume, lower, close  
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],  
              [823.02002, 828.070007, 1828100, 821.655029, 828.070007],  
              [819.929993, 824.400024, 1438100, 818.97998, 824.159973],  
              [816, 820.958984, 1008100, 815.48999, 819.23999],  
              [819.359985, 823, 1188100, 818.469971, 818.97998],  
              [819, 823, 1198100, 816, 820.450012],  
              [811.700012, 815.25, 1098100, 809.780029, 813.669983],  
              [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])  
  
[]: x_data = xy[:, :-1]  
y_data = xy[:, [-1]]
```

```
: tf.model = tf.keras.Sequential()  
# 모델 틀을 먼저 만들어놓고 나중에 구성하려는 신경망의 정보나 추가적인 요소 링들을 추가적으로 구현을 해주면 되겠다~~~~~라는 겁니다.  
tf.model.add(tf.keras.layers.Dense(units=1, input_dim=x_data.shape[1])) # keras 스킴 # Dense는 layers를 합싼다.  
tf.model.add(tf.keras.layers.Activation('linear'))
```

```
] : tf.model.compile(loss='mse', optimizer=tf.keras.optimizers.SGD(learning_rate=1e-5))  
tf.model.summary()
```

Model: "sequential\_2"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 1)	5
activation (Activation)	(None, 1)	0
<hr/>		
Total params: 5		
Trainable params: 5		
Non-trainable params: 0		
<hr/>		

Param #
5
0

input data = 8행 4열

w1

w2

w3

w4

output data = 8행 1열

param의 갯수의 이유: w의 갯수(가중치,기울기) + b의 갯수(편향) = 5

$x_1 \cdot w_1 + x_2 \cdot w_2 + x_3 \cdot w_3 + x_4 \cdot w_4 \rightarrow y_1$

mse는 손실함수 -  $H(w,b)$

a(w,b)편미분을 w랑 마이너스 해주면서 계속 옮겨줌

```
[0]: loss = tf.model.fit(x_data, y_data, epochs=1000)
[0]: 1/1 [=====] - 0s 4ms/step - loss: nan
Epoch 992/1000
1/1 [=====] - 0s 11ms/step - loss: nan
Epoch 993/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 994/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 995/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 996/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 997/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 998/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 999/1000
1/1 [=====] - 0s 3ms/step - loss: nan
Epoch 1000/1000
1/1 [=====] - 0s 3ms/step - loss: nan
```

결국 발산해버림.

## without\_min\_max

---

### 06\_linear\_regression\_min\_max

정규화, 표준화를 사용.

---

정규화 식

$$x - \min / \max - \min$$

각 열 별로 기준 min max를 찾음.

---

```

]: import tensorflow as tf
import numpy as np

# open, higher, volume, lower, close
xy = np.array([[828.659973, 833.450012, 908100, 828.349976, 831.659973],
               [823.02002, 828.070007, 1828100, 821.655029, 828.070007],
               [819.929993, 824.400024, 1438100, 818.97998, 824.159973],
               [816, 820.958984, 1008100, 815.48999, 819.23999],
               [819.359985, 823, 1188100, 818.469971, 818.97998],
               [819, 823, 1198100, 816, 820.450012],
               [811.700012, 815.25, 1098100, 809.780029, 813.669983],
               [809.51001, 816.659973, 1398100, 804.539978, 809.559998]])

]: def min_max_scaler(data):
    numerator = data - np.min(data, axis=0)
    denominator = np.max(data, axis=0) - np.min(data, axis=0)
    return numerator / (denominator + 1e-7)

]: xy_scaler = min_max_scaler(xy)
xy_scaler

]: array([[0.99999999, 0.99999999, 0., 1., 1., 1.],
           [0.70548491, 0.70439552, 1., 0.71881782, 0.83755791],
           [0.54412549, 0.50274824, 0.57608696, 0.606468, 0.6606331],
           [0.33890353, 0.31368023, 0.10869565, 0.45989134, 0.43800918],
           [0.51436, 0.42582389, 0.30434783, 0.58504805, 0.42624401],
           [0.49556179, 0.42582389, 0.31521739, 0.48131134, 0.49276137],
           [0.11436064, 0., 0.20652174, 0.22007776, 0.18597238],
           [0., 0.07747099, 0.5326087, 0., 0.]]))

]: x_scaler = xy_scaler[:, 0:-1]
y_scaler = xy_scaler[:, [-1]]

]: tf.model = tf.keras.Sequential()
# 모델 를을 먼저 만들어놓고 나중에 구성하려는 신경망의 정보나 추가적인 요인 를 추가적으로 구현을 해주면 되겠다~~~~~라는 겁니다.
tf.model.add(tf.keras.layers.Dense(units=1, input_dim=x_data.shape[1])) # keras 스크립트 Dense는 layers를 감싼다.
tf.model.add(tf.keras.layers.Activation('linear'))

```

```
: tf.model.compile(loss='mse', optimizer=tf.keras.optimizers.SGD(learning_rate=1e-5))
tf.model.summary()
```

Model: "sequential\_9"

Layer (type)	Output Shape	Param #
<hr/>		
dense_9 (Dense)	(None, 1)	5
activation_9 (Activation)	(None, 1)	0
<hr/>		
Total params: 5		
Trainable params: 5		
Non-trainable params: 0		
<hr/>		

```
] loss = tf.model.fit(x_scaler, y_scaler, epochs=1000)
1/1 [=====] - 0s 3ms/step - loss: 0.0656
Epoch 992/1000
1/1 [=====] - 0s 2ms/step - loss: 0.0655
Epoch 993/1000
1/1 [=====] - 0s 3ms/step - loss: 0.0655
Epoch 994/1000
1/1 [=====] - 0s 4ms/step - loss: 0.0655
Epoch 995/1000
1/1 [=====] - 0s 3ms/step - loss: 0.0655
Epoch 996/1000
1/1 [=====] - 0s 3ms/step - loss: 0.0655
Epoch 997/1000
1/1 [=====] - 0s 3ms/step - loss: 0.0655
Epoch 998/1000
1/1 [=====] - 0s 4ms/step - loss: 0.0655
Epoch 999/1000
1/1 [=====] - 0s 3ms/step - loss: 0.0655
Epoch 1000/1000
1/1 [=====] - 0s 2ms/step - loss: 0.0655
```

[Numpy] 최소값(min) 최대값(max) 구하기 (부록: apply\_along\_axis)

Numpy는 다차원 데이터를 다루는데 유용합니다. 다차원 데이터들의 최소값과 최대값을 구하는 방법에 대해...

 <https://m.blog.naver.com/wideeyed/221615469366>

[ 0, 0, 1,  
 5, 6, 7,

```
] : # data에서 axis=0이 되는지 설명 예제  
data = np.array([[1,6,7,12],  
                 [2,5,8,11],  
                 [3,4,9,10]])  
  
print(np.min(data, axis=1))  
[1 2 3]  
  
] : print(np.min(data, axis=0))  
[ 1  4  7 10]
```

---

## 퍼셉트론

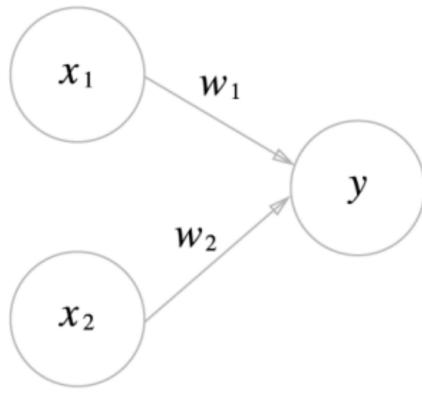
---

- 퍼셉트론(perceptron) 알고리즘을 설명
  - 프랑크 로젠틀라트(Frank Rosenblatt)가 1957년에 고안한 알고리즘
  - 신경망(딥러닝)의 기원이 되는 알고리즘
  - 퍼셉트론의 구조를 배우는 것은 신경망과 딥러닝으로 나아가는 데 중요한 아이디어를 배우는 일
- 

동그라미 모양을 뉴런이라고 표현.

## 퍼셉트론이란?

---



▶ 입력으로 2개의 신호를 받은 퍼셉트론의 예

- $x_1$ 과  $x_2$ 는 입력 신호
- $y$ 는 출력 신호
- $w_1$ 과  $w_2$ 는 가중치(weight) : 각 신호가 결과에 주는 영향력을 조절하는 요소로 작용
- 뉴런(혹은 노드) : 그림의 원
- $\theta(\text{theta})$  : 임계값(뉴런의 활성화 기준값)

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

세타(뉴런의 활성화 기준값)

$\theta(\text{theta})$

---

선형성을 뛴다는 것은 1차함수( $X^*W$ )

$Y = X^*W + \text{bias}$

---

**reversed()**

두 번째는 **reversed()** 함수를 사용하는 방법입니다.

직접 해보는 손코딩

반대로 반복하기(2) [소스 코드 reversed\\_for02.py](#)

```
01 # 역반복문  
02 for i in reversed(range(5)):  
03     # 출력합니다.  
04     print("현재 반복 변수: {}".format(i))
```

실행 결과	
현재 반복 변수: 4	
현재 반복 변수: 3	
현재 반복 변수: 2	
현재 반복 변수: 1	
현재 반복 변수: 0	

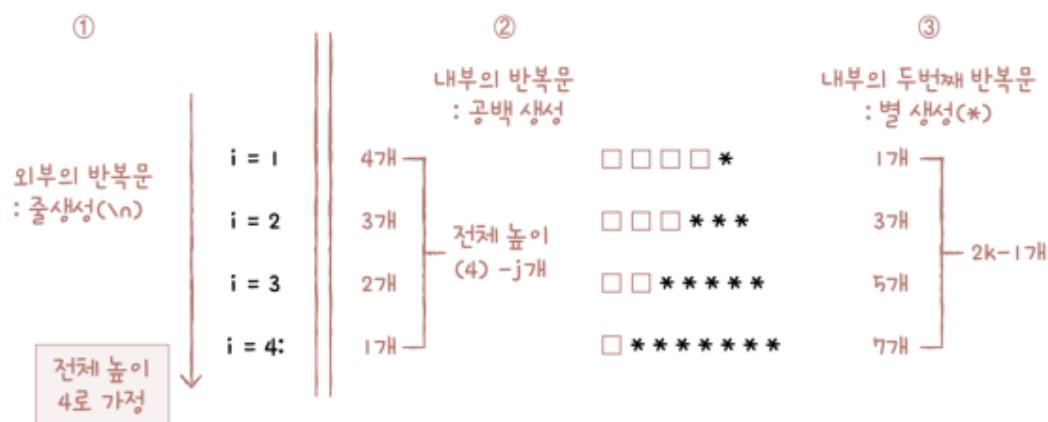
reversed() 함수를 적용하면 [0, 1, 2, 3, 4]라는 형태의 범위가 [4, 3, 2, 1, 0]으로 뒤집어집니다. 따라서 9부터 0까지 반대로 반복문을 돌릴 수 있습니다.

```
*  
***  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****  
*****
```

## 반복문으로 피라미드 만들기(2)

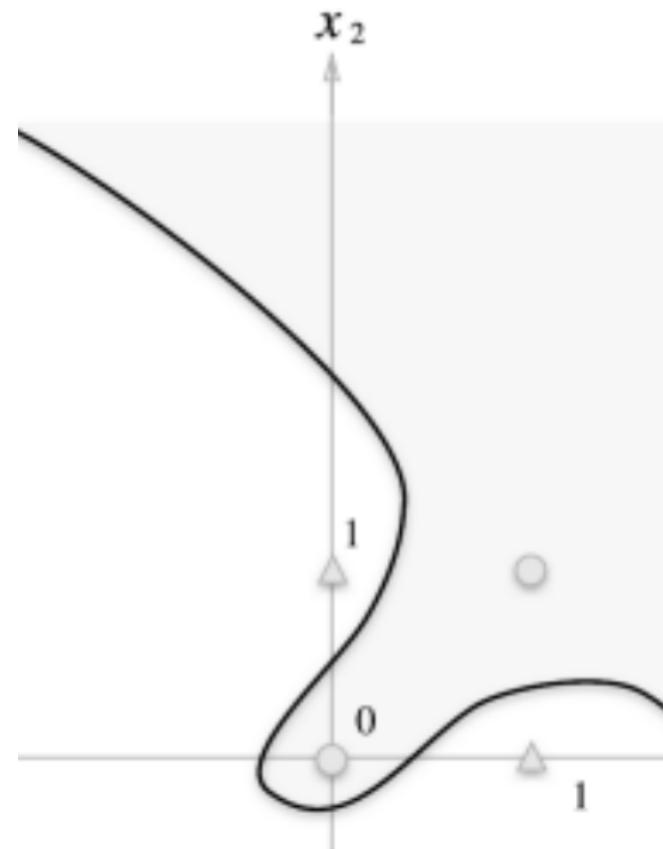
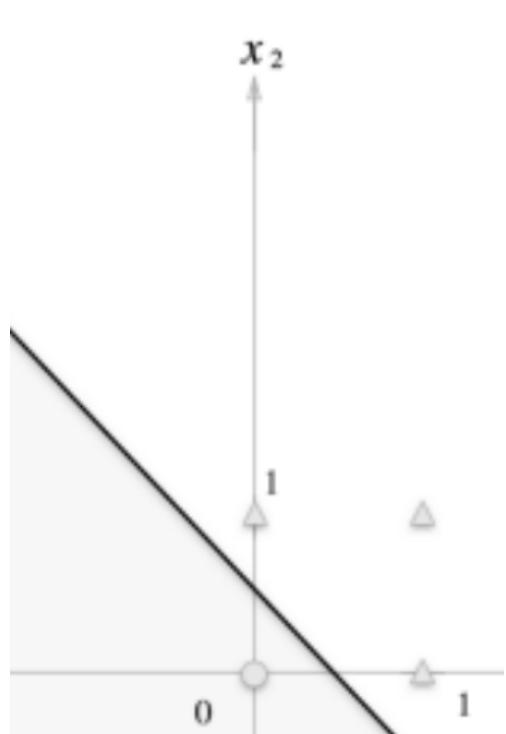
소스 코드 for\_pyramid02.py

```
01 output = ""  
02  
03 for i in range(1, 15):  
04     for j in range(14, i, -1):  
05         output += ' '  
06     for k in range(0, 2 * i - 1):  
07         output += '*'  
08     output += '\n'  
09  
10 print(output)
```



## OR 게이트

## - XOR 게이트



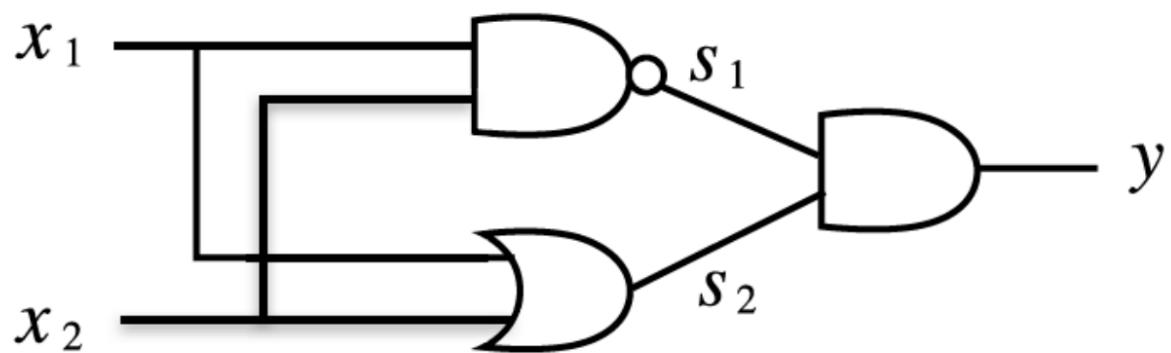
AND



NAND



OR



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

0은 신호를 안내보내는 거

1은 신호를 내보내는 거.

## $\Theta(\text{theta})$ : 임계값(뉴런의 활성화 기준값)

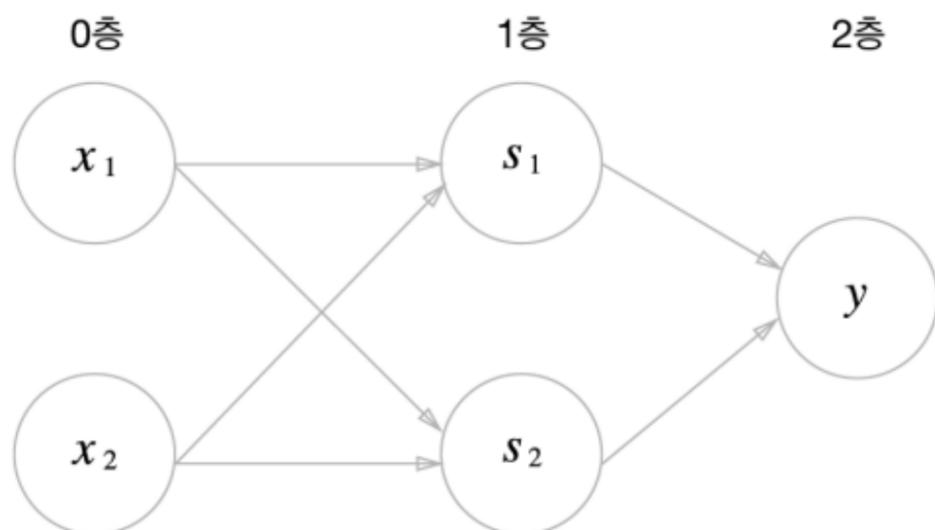
세타를 bias로 표현한 것.

0층은 입력층

1층은 Layer(Dense) - units를 담은 것이 Dense이고 공식적인 명칭은 Layer라고 부른다.

2층도 Layer이고 출력으로 나감.

## XOR 게이트 구현하기



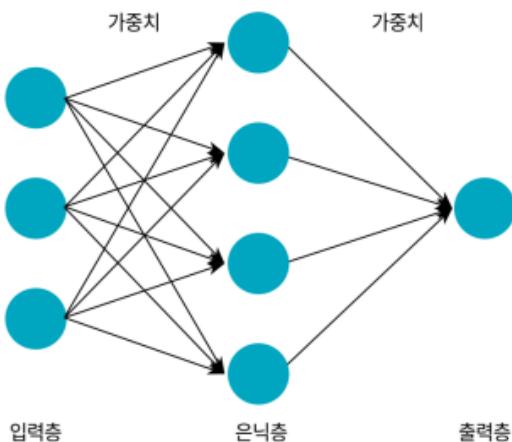
## XOR 게이트 구현하기

### ➤ 다층 퍼셉트론의 동작 설명

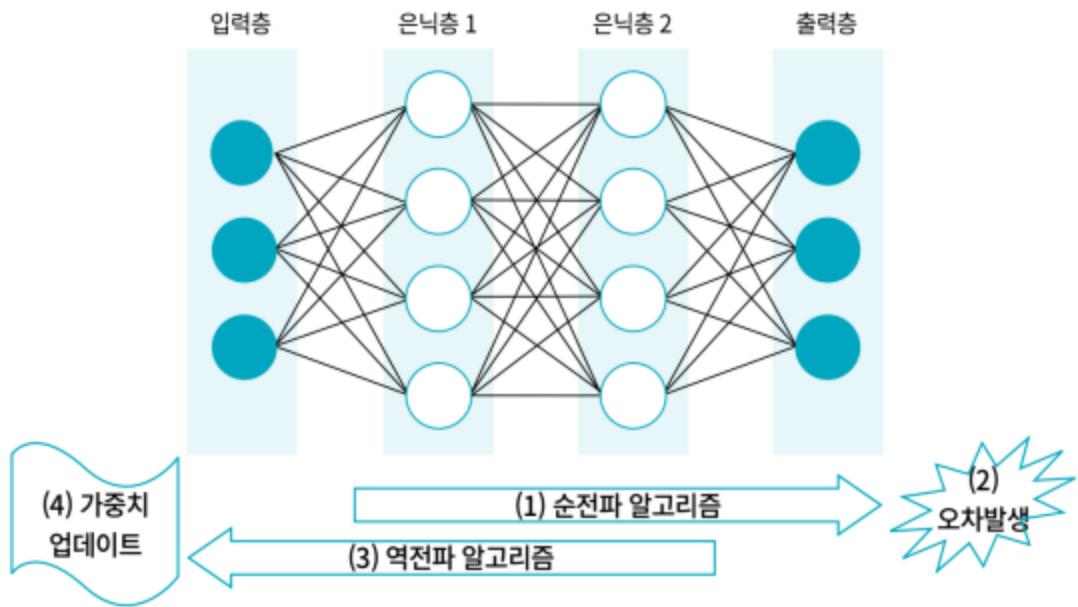
1. 0층의 두 뉴런이 입력 신호를 받아 1층의 뉴런으로 신호를 보낸다.
2. 1층의 뉴런이 2층의 뉴런으로 신호를 보내고, 2층의 뉴런은 이 입력신호를 바탕으로  $y$ 를 출력한다.
3. 단층 퍼셉트론으로는 표현하지 못한 것을 층을 하나 늘려 구현 할 수 있었다.
  - 퍼셉트론은 층을 쌓아(깊게 하여) 더 다양한 것을 표현할 수 있다.

### ② 인공신경망의 계층 구조 \*\*\*

- 하나의 인공신경망은 데이터를 입력하는 입력층, 데이터를 출력하는 출력층을 갖고 있는 단층신경망과 입력층과 출력층 사이에 보이지 않는 다수의 은닉층을 가지고 있을 수 있는 다층신경망으로 구분할 수 있다. 은닉층이 존재하지 않는 단층신경망은 한계점이 있기에 일반적인 인공신경망은 다층신경망을 의미한다.
- 입력층은 데이터를 입력받아 시스템으로 전송하는 역할을 한다. 은닉층은 신경망 외부에서는 은닉층의 노드에 직접 접근할 수 없도록 숨겨진, 말 그대로 은닉한 층이다. 은닉층은 입력층으로부터 값을 전달받아 가중치를 계산한 후 활성함수에 적용하여 결과를 산출하고 이를 출력층으로 보낸다. 출력층은 학습된 데이터가 포함된 층으로, 활성함수의 결과를 담고 있는 노드로 구성된다. 출력층의 노드 수는 출력 범주의 수로 결정된다. 분류 문제일 경우 출력층의 노드는 각 라벨의 확률을 포함한다.

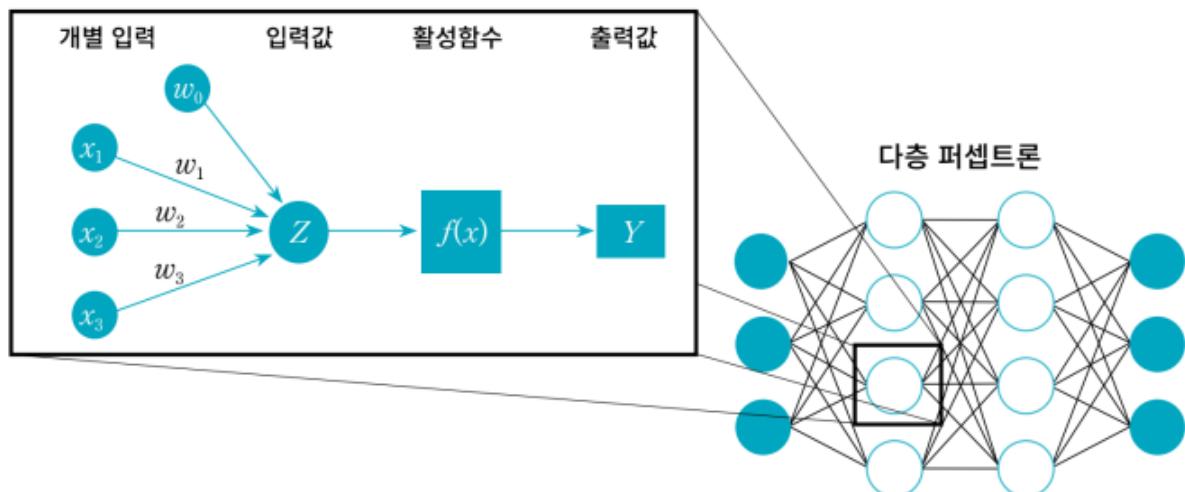


### 【 인공신경망의 학습 과정 】



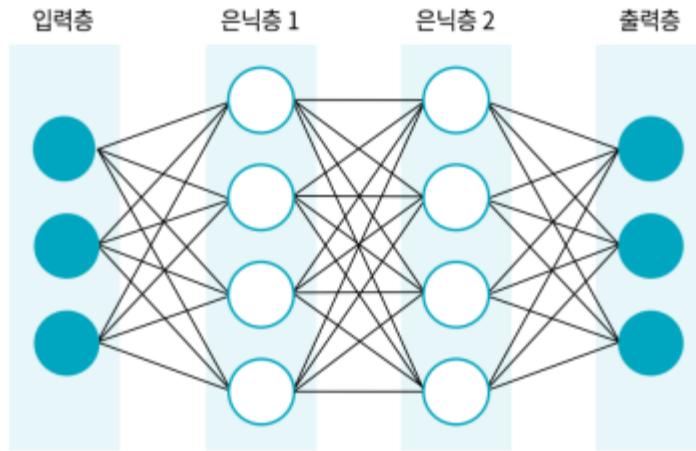
### 【 퍼셉트론 】

#### 단층 퍼셉트론



( $x_i$ : 개별 입력값,  $w_i$ : 개별 가중치,  $w_0$ : 상수항(편차),  $Z = w_0 + \sum x_i w_i$ : 입력값)

## 【 다층 퍼셉트론의 구조 】



딥러닝 = (전체) 신경망

## 01\_xor

```
: import tensorflow as tf
import numpy as np
x_data = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=np.float32)
y_data = np.array([[0],[1],[1],[0]], dtype=np.float32)

: tf.model = tf.keras.Sequential()
tf.model.add(tf.keras.layers.Dense(units=1, input_dim=2, activation='sigmoid'))
tf.model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.01), metrics=['accuracy'])
tf.model.summary()

Model: "sequential_4"
-----  

Layer (type)          Output Shape         Param #
-----  

dense_2 (Dense)        (None, 1)            3  

-----  

Total params: 3
Trainable params: 3
Non-trainable params: 0
```

```

: tf.model.fit(x_data,y_data,epochs=4000)
Epoch 3992/4000
1/1 [=====] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3993/4000
1/1 [=====] - 0s 3ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3994/4000
1/1 [=====] - 0s 3ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3995/4000
1/1 [=====] - 0s 3ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3996/4000
1/1 [=====] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3997/4000
1/1 [=====] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3998/4000
1/1 [=====] - 0s 4ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 3999/4000
1/1 [=====] - 0s 3ms/step - loss: 0.6932 - accuracy: 0.5000
Epoch 4000/4000
1/1 [=====] - 0s 2ms/step - loss: 0.6932 - accuracy: 0.5000
<keras.callbacks.History at 0x24de2273308>

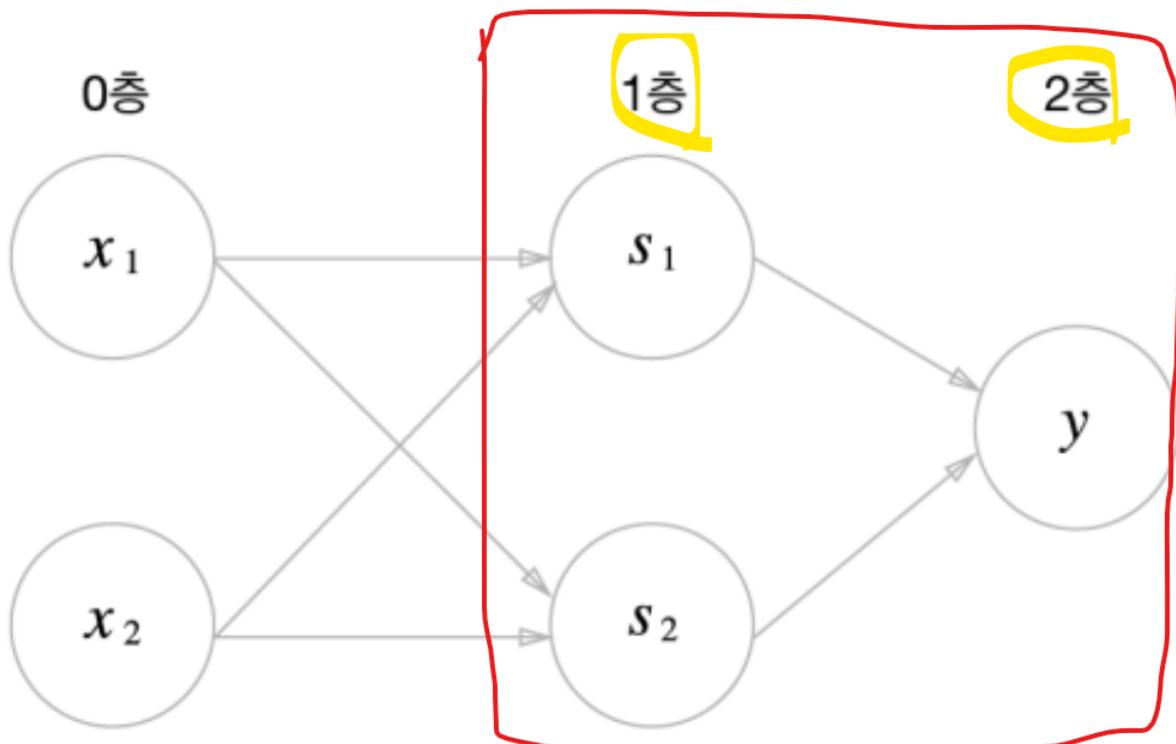
```

## 02\_xor\_NeuralNetwork

층(Layer)을 2개로

뉴런은 2개

0층인 입력은 2개.



참고

## 다양한 인공신경망 구조



- RNN: 순환 신경망으로 입력층의 데이터는 은닉층을 통해 출력층으로 가지만 은닉층의 결과값이 다음 입력 데이터가 입력될 때 자기 자신에게 영향을 주는 신경망이다. 언어 모델링, 음성 인식 등에 활용될 수 있다. 그러나 시간적으로 오래된 데이터에 대한 문맥 처리가 어려운 것이 단점이다.
- CNN: 합성곱 신경망으로 이미지 분류 및 다중객체탐지(Multi Object Detection)에 뛰어난 성능을 보인다. 합성곱과 풀링의 두 개의 작업으로 구성되어 있다.
- LSTM: 장단기 메모리라고 하며, RNN의 한계를 극복하기 위해 나온 인공신경망 모형이다. 오래된 데이터는 버리고, 가장 최근의 데이터를 더욱 오래 보존하려는 것이 특징이다.
- YOLO: 이미지 속에서 물체를 탐지하는 알고리즘으로, CNN과 목적은 유사하지만 다른 알고리즘을 갖는다. 한 장의 이미지를 수십 개의 박스로 나누어 각 박스에 대해 가장 확률이 높은 객체를 탐지한다. You Only Look Once의 약자로, 이미지 전체를 한 번만 보기 때문에 빠른 속도를 보여준다.
- GAN: 생산적 적대 신경망으로, 위조지폐범의 위조지폐 생산과 경찰(분류모형)의 지폐 판별 과정을 반복하면서 학습할 경우 위조지폐범이 진짜에 가까운 위조지폐를 생산할 수 있는 것처럼 분류모형으로부터 최적의 결과를 얻을 수 있도록 유도하는 학습이다. 대표적인 사례로 페이스북의 딥 페이스가 있다.

```
| : import tensorflow as tf
| : import numpy as np
x_data = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=np.float32)
y_data = np.array([[0],[1],[1],[0]], dtype=np.float32)
```

```
| : neurun_1 = 2 # layer 2#
| : neurun_2 = 1 # layer 1#
| :
| : tf.model = tf.keras.Sequential()
| : tf.model.add(tf.keras.layers.Dense(units=neurun_1, input_dim=x_data.shape[1], activation='sigmoid'))
# tf.model.add(tf.keras.layers.Activation('sigmoid'))
| :
| : tf.model.add(tf.keras.layers.Dense(units=neurun_2, activation='sigmoid'))
# tf.model.add(tf.keras.layers.Activation('sigmoid'))
```

```
tf.model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.1), metrics=['accuracy'])
tf.model.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 2)	6
dense_2 (Dense)	(None, 1)	3
<hr/>		
Total params: 9		
Trainable params: 9		
Non-trainable params: 0		

```
-----  
Param #  
-----  
6  
3
```

w값 4개, bias값 2개 = Param 6

w값 2개, bias값 1개 = Param 3

```
] : tf.model.fit(x_data, y_data, epochs=5000)  
Epoch 4992/5000  
1/1 [=====] - 0s 3ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4993/5000  
1/1 [=====] - 0s 3ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4994/5000  
1/1 [=====] - 0s 4ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4995/5000  
1/1 [=====] - 0s 3ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4996/5000  
1/1 [=====] - 0s 2ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4997/5000  
1/1 [=====] - 0s 4ms/step - loss: 0.0221 - accuracy: 1.0000  
Epoch 4998/5000  
1/1 [=====] - 0s 3ms/step - loss: 0.0220 - accuracy: 1.0000  
Epoch 4999/5000  
1/1 [=====] - 0s 3ms/step - loss: 0.0220 - accuracy: 1.0000  
Epoch 5000/5000  
1/1 [=====] - 0s 4ms/step - loss: 0.0220 - accuracy: 1.0000  
] : <keras.callbacks.History at 0x2b456128848>
```

```
predictions = tf.model.predict(x_data)  
predictions
```

```
1/1 [=====] - 0s 71ms/step  
array([[0.01715828],  
       [0.9762696 ],  
       [0.9762953 ],  
       [0.02256795]], dtype=float32)
```

## 03\_xor\_DeepLearning

```
: import tensorflow as tf  
import numpy as np  
  
x_data = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=np.float32)  
y_data = np.array([[0],[1],[1],[0]], dtype=np.float32)
```

```

: neuron_1 = 10
neuron_2 = 1

tf.model = tf.keras.Sequential()
tf.model.add(tf.keras.layers.Dense(units=neuron_1, input_dim=x_data.shape[1], activation='sigmoid')) # Layer 1
tf.model.add(tf.keras.layers.Dense(units=neuron_1, activation='sigmoid')) # Layer 2
tf.model.add(tf.keras.layers.Dense(units=neuron_1, activation='sigmoid')) # Layer 3 # 보통 3개 이상이면 deep learning이라고 한다.
tf.model.add(tf.keras.layers.Dense(units=neuron_1, activation='sigmoid')) # Layer 4
tf.model.add(tf.keras.layers.Dense(units=neuron_2, activation='sigmoid')) # Layer 5

```

```

: tf.model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.1), metrics=['accuracy'])
tf.model.summary()

```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 10)	30
dense_1 (Dense)	(None, 10)	110
dense_2 (Dense)	(None, 10)	110
dense_3 (Dense)	(None, 10)	110
dense_4 (Dense)	(None, 1)	11

Total params:
371
Trainable params: 371
Non-trainable params: 0

```

[4]: tf.model.fit(x_data, y_data, epochs=5000)
Epoch 4952/5000
1/1 [=====] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4993/5000
1/1 [=====] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4994/5000
1/1 [=====] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4995/5000
1/1 [=====] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4996/5000
1/1 [=====] - 0s 2ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4997/5000
1/1 [=====] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4998/5000
1/1 [=====] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 4999/5000
1/1 [=====] - 0s 3ms/step - loss: 0.6931 - accuracy: 0.5000
Epoch 5000/5000
1/1 [=====] - 0s 4ms/step - loss: 0.6931 - accuracy: 0.5000

```

```
[4]: <keras.callbacks.History at 0x1f082ce9588>
```

```

[5]: predictions = tf.model.predict(x_data)
predictions

```

```
1/1 [=====] - 0s 77ms/step
```

```

[5]: array([[0.49992523],
           [0.49992898],
           [0.50005597],
           [0.50001496]], dtype=float32)

```

---

## openCV - 영상 처리쪽에 더 주력을 두는 Computer Vision

---

딥러닝이 모든 분야에서 만능이 아니다.

맞는 분야에 적용을 했을 때 라야 맞다.

---

## 04\_xor\_tensorboard

```
import datetime
import numpy as np
import os
import tensorflow as tf

x_data = np.array([[0,0],[0,1],[1,0],[1,1]], dtype=np.float32)
y_data = np.array([0,1,1,0], dtype=np.float32)

neuron_1 = 10
neuron_2 = 1

tf.model = tf.keras.Sequential()
tf.model.add(tf.keras.layers.Dense(units=neuron_1, input_dim=x_data.shape[1], activation='sigmoid')) # Layer 1
tf.model.add(tf.keras.layers.Dense(units=neuron_1, activation='sigmoid')) # Layer 2
tf.model.add(tf.keras.layers.Dense(units=neuron_2, activation='sigmoid'))
```

```
: tf.model.compile(loss='binary_crossentropy', optimizer=tf.optimizers.SGD(learning_rate=0.1), metrics=['accuracy'])
tf.model.summary()

Model: "sequential_1"
=====
Layer (type)      Output Shape       Param #
=====
dense_3 (Dense)   (None, 10)        30
dense_4 (Dense)   (None, 10)        110
dense_5 (Dense)   (None, 1)         11
=====
Total params: 151
Trainable params: 151
Non-trainable params: 0
=====
```

---

```

for i in temp:
    print("첫 번째 반복문: {}".format(i))

for i in temp:
    print("두 번째 반복문: {}".format(i))

```

코드를 실행하면 "첫 번째 반복문" 부분만 실행되고 "두 번째 반복문" 부분은 전혀 출력되지 않습니다. 왜 그럴까요?

```

첫 번째 반복문: 6
첫 번째 반복문: 5
첫 번째 반복문: 4
첫 번째 반복문: 3
첫 번째 반복문: 2
첫 번째 반복문: 1

```

이는 reversed() 함수의 결과가 **제너레이터**이기 때문입니다. 제너레이터는 파이썬의 특별한 기능으로, 5장에서 살펴보고 일단 reversed() 함수와 반복문을 조합할 때는 함수의 결과를 여러 번 활용하지 않고 다음과 같이 for 구문 내부에 reversed() 함수를 곧바로 넣어서 사용한다고 기억해 주세요.

```

numbers = [1, 2, 3, 4, 5, 6]

for i in reversed(numbers):
    print("첫 번째 반복문: {}".format(i))

for i in reversed(numbers):
    print("두 번째 반복문: {}".format(i))

```

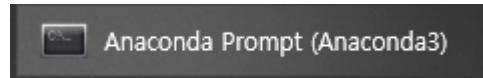
→ 필요한 시점에 reversed() 함수를 사용합니다.

```

7]: log_dir = os.path.join(".", "logs", "fit", datetime.datetime.now().strftime("%Y%m%d-%H%M%S")) # 템포를 만들라고 지시함.
tesorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

8]: loss = tf.model.fit(x_data, y_data, epochs=10000, callbacks=[tesorboard_callback])
1/1 [=====] - 0s 78ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9992/10000
1/1 [=====] - 0s 105ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9993/10000
1/1 [=====] - 0s 57ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9994/10000
1/1 [=====] - 0s 87ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9995/10000
1/1 [=====] - 0s 62ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9996/10000
1/1 [=====] - 0s 47ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9997/10000
1/1 [=====] - 0s 62ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9998/10000
1/1 [=====] - 0s 94ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 9999/10000
1/1 [=====] - 0s 62ms/step - loss: 0.0050 - accuracy: 1.0000
Epoch 10000/10000
1/1 [=====] - 0s 47ms/step - loss: 0.0050 - accuracy: 1.0000

```



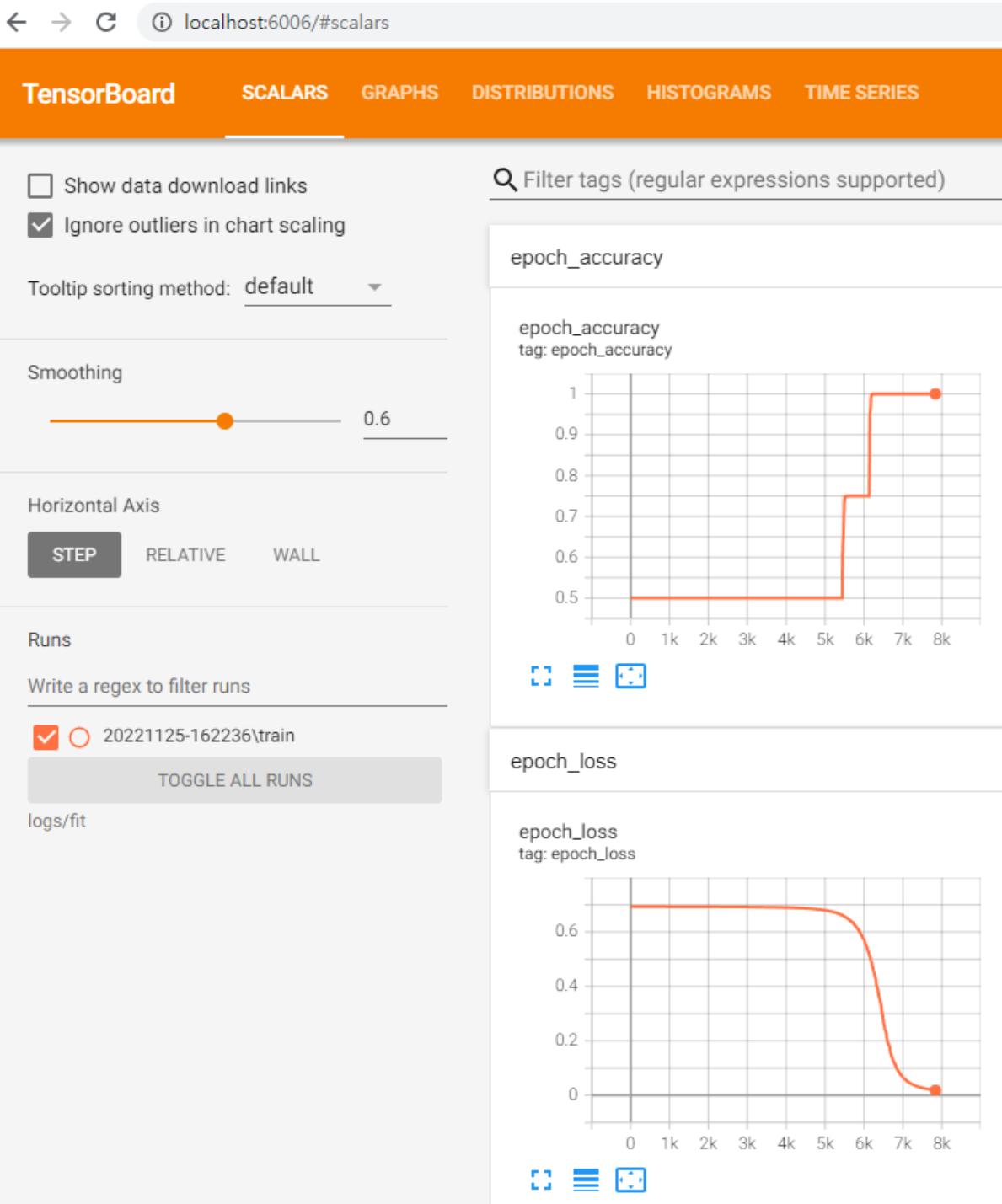
<http://localhost:6006/>

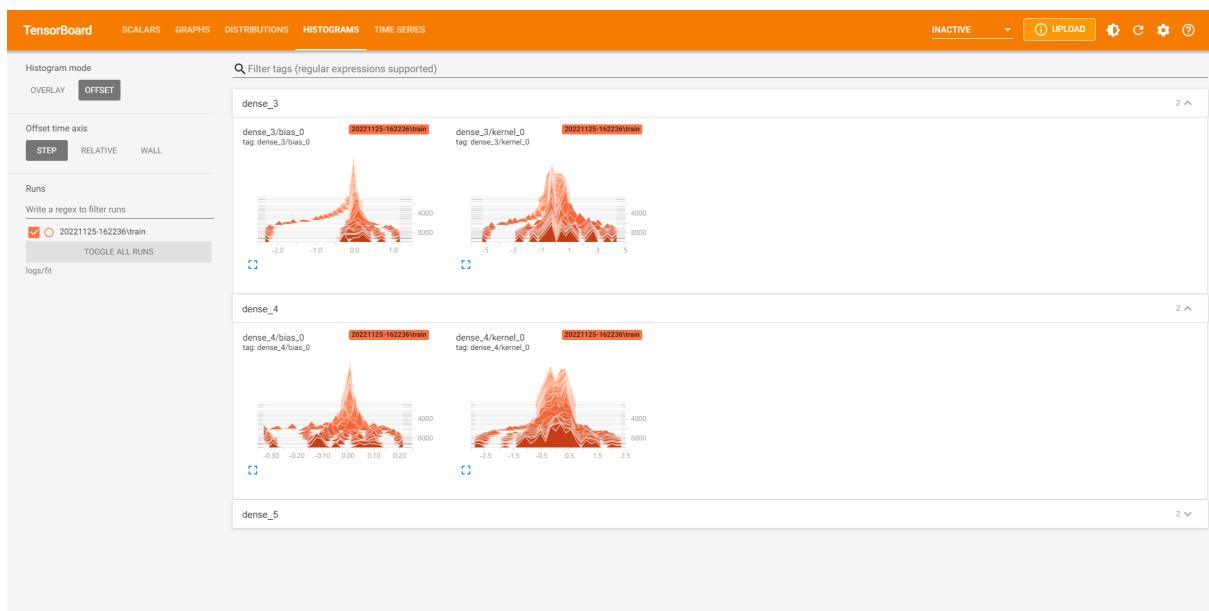
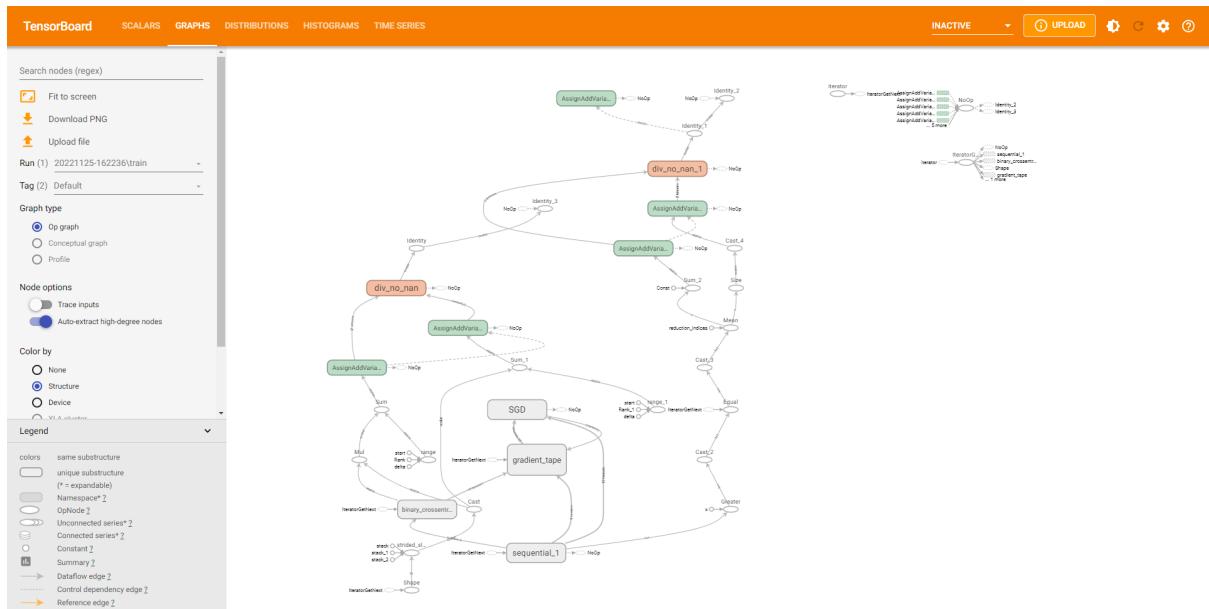
관리자: Anaconda Prompt (Anaconda3) - tensorboard --logdir logs/fit

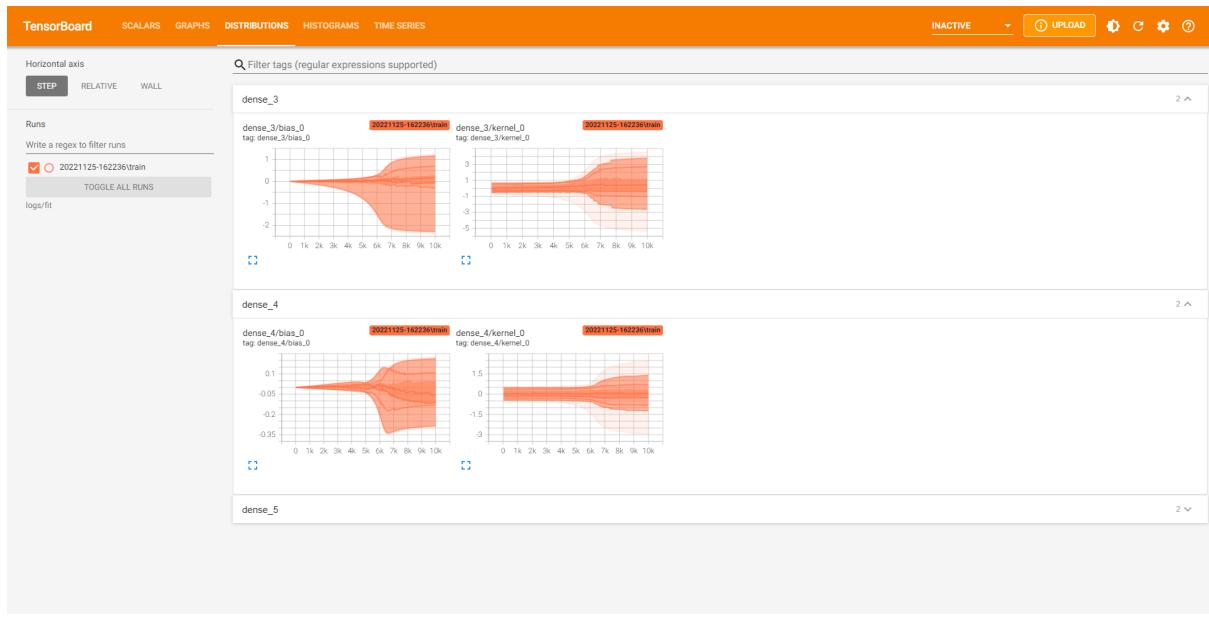
```

(base) C:\WINDOWS\system32>conda activate tf_cpu
(tf_cpu) C:\WINDOWS\system32>cd D:\heaven_dev\workspaces\AI\src\chap05
(tf_cpu) C:\WINDOWS\system32>d:
(tf_cpu) D:\heaven_dev\workspaces\AI\src\chap05>tensorboard --logdir logs/fit
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.9.0 at http://localhost:6006/ (Press CTRL+C to quit)
forrtl: error (200): program aborting due to control-C event
Image          PC          Routine         Line      Source
libifcoremd.dll    00007FFD62F5DF54 Unknown           Unknown Unknown
KERNELBASE.dll     00007FFDFC48AB73 Unknown           Unknown Unknown
KERNEL32.dll       00007FFDFDBC74B4 Unknown           Unknown Unknown
ntdll.dll        00007FFDFE6E26A1 Unknown           Unknown Unknown
(tf_cpu) D:\heaven_dev\workspaces\AI\src\chap05>tensorboard --logdir logs/fit
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.9.0 at http://localhost:6006/ (Press CTRL+C to quit)

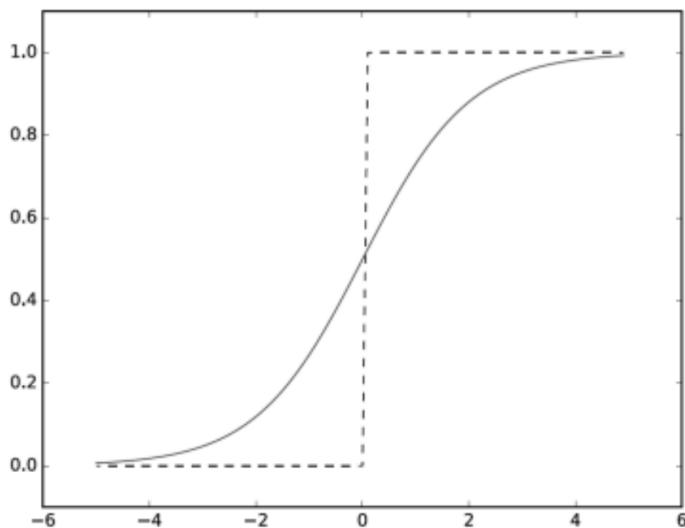
```







## 시그모이드 함수와 계단 함수 비교



\* 퍼셉트론과 신경망의 주된 차이는  
활성화 함수의 차이

- 계단 함수가 0과 1 중 하나의 값만 돌려주는 반면 시그모이드 함수는 실수를 돌려준다.
- 즉, 퍼셉트론에서는 뉴런 사이에 0 혹은 1이 흘렀다면, 신경망에서는 연속적인 실수가 흐른다.
- 두 함수 모두 비선형 함수.
- 신경망에서는 활성화 함수로 비선형 함수를 사용해야 함.

**rectified**

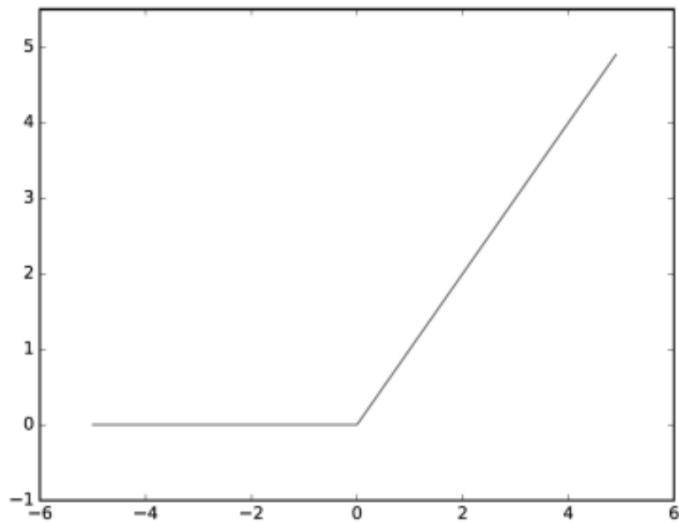
(형용사)

1 정류한

2 정류한

영어사전 다른 뜻 1

## ReLU 함수(Rectified Linear Unit Function)

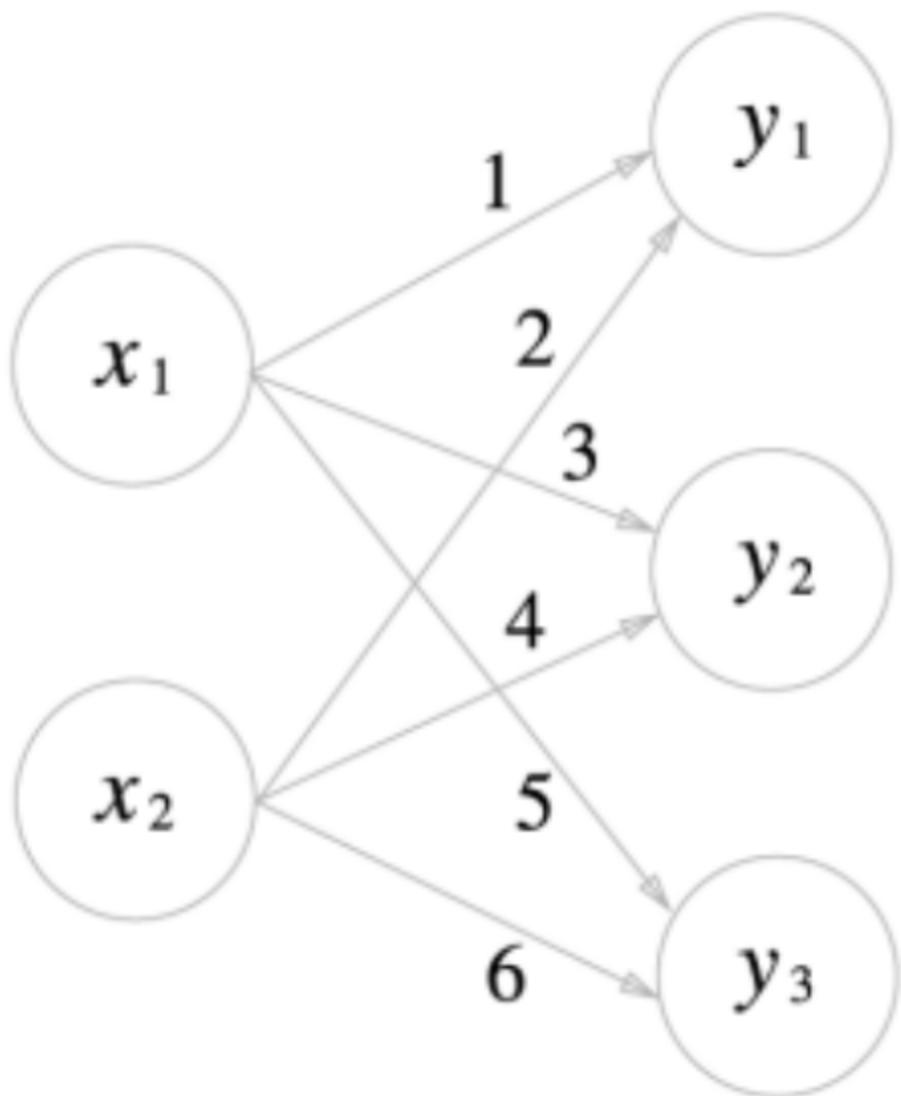


$$h(x) = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

- 최근 신경망 분야에서 이용.
- 입력이 0을 넘으면 그대로 출력하고, 0 이하이면 0을 출력하는 함수.

# 신경망의 내적

---



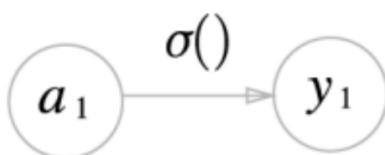
$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

$$X \quad W = Y$$

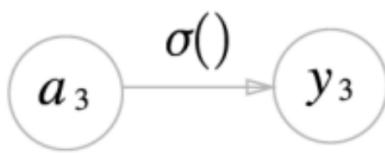
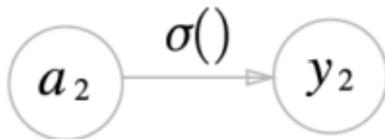
2                   $2 \times 3$                   3  
 ↓                  ↓                  ↓                  ↓  
 일치

선형 회귀는 byPass 해주는 항등함수를 활성화함수에서 적용해줘서 그저 넘겨준다.

### 출력층 설계 (1) – 항등함수



- 항등 함수(identity function) : 회귀
  - 입력을 그대로 출력



43페이지까지 훑어보면 이해됨.

43 / 133

[https://s3-us-west-2.amazonaws.com/secure.notion-static.com/58ba2de4-9382-4be0-b3bf-b3cbff76dfcc/06\\_%EC%8B%A0%EA%B2%BD%EB%A7%9D\(%EB%94%A5%EB%9F%AC%EB%8B%9D\).pdf](https://s3-us-west-2.amazonaws.com/secure.notion-static.com/58ba2de4-9382-4be0-b3bf-b3cbff76dfcc/06_%EC%8B%A0%EA%B2%BD%EB%A7%9D(%EB%94%A5%EB%9F%AC%EB%8B%9D).pdf)