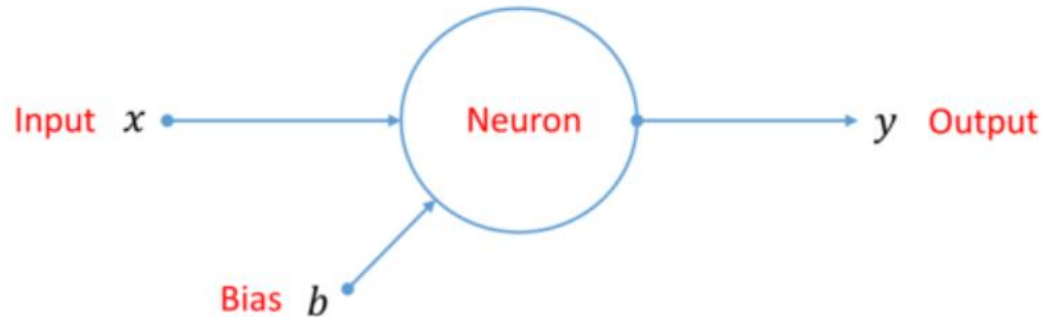


# 인공 신경망 퍼셉트론의 이해

# 인공 신경 세포(Artificial Neuron)

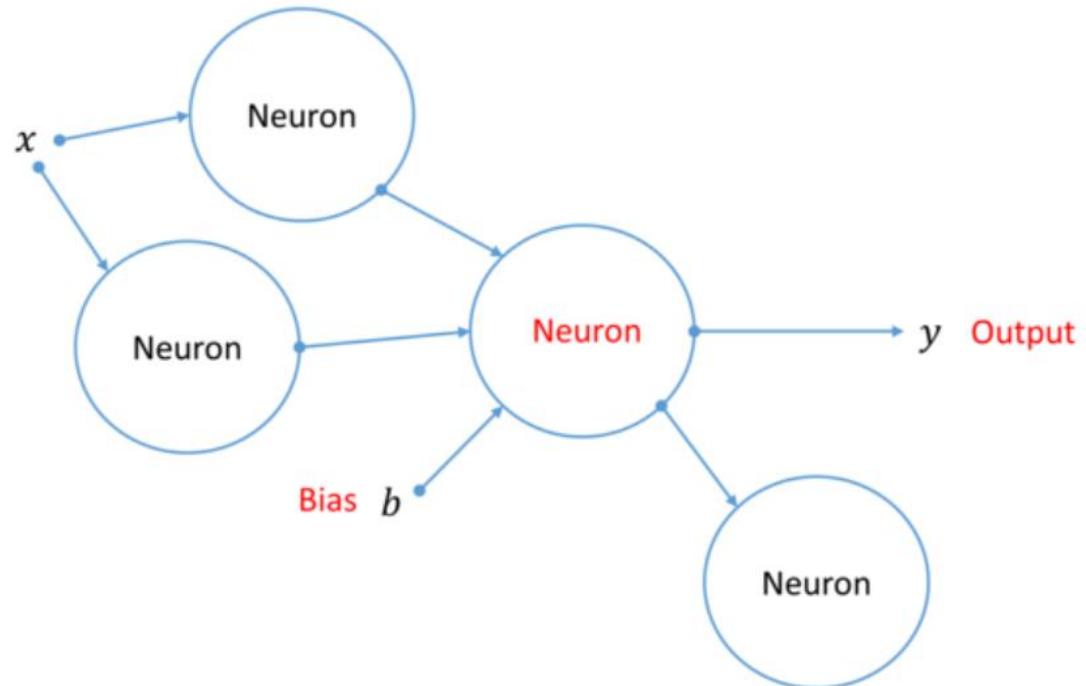
- 뉴런

- 입력
- 편향(bias)



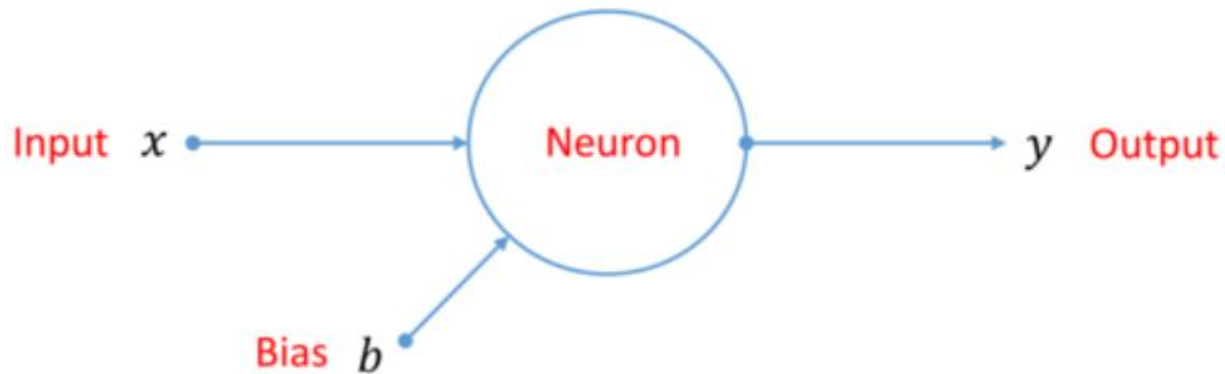
- 신경망(network)

- 뉴런의 연결



# 입력과 출력

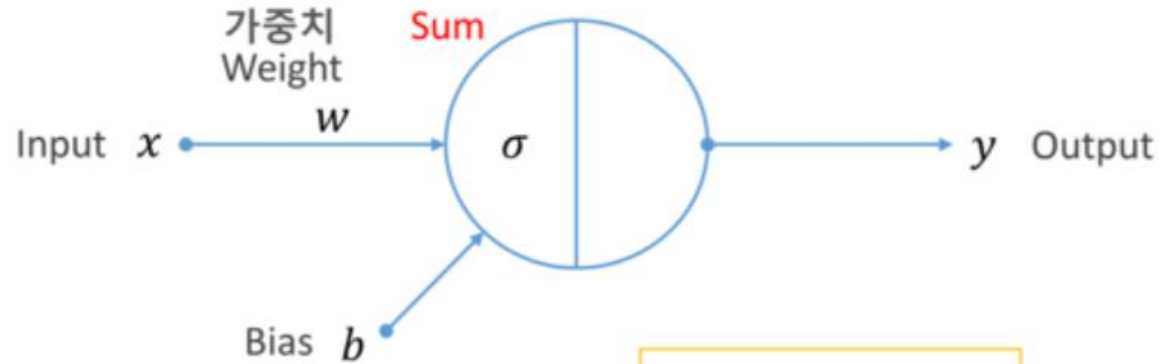
- 편향(bias)
  - 편향을 조정해 출력을 맞춤



Input $x$	Output $y$
Size of house	Price
Time spent for studying	Score in exam

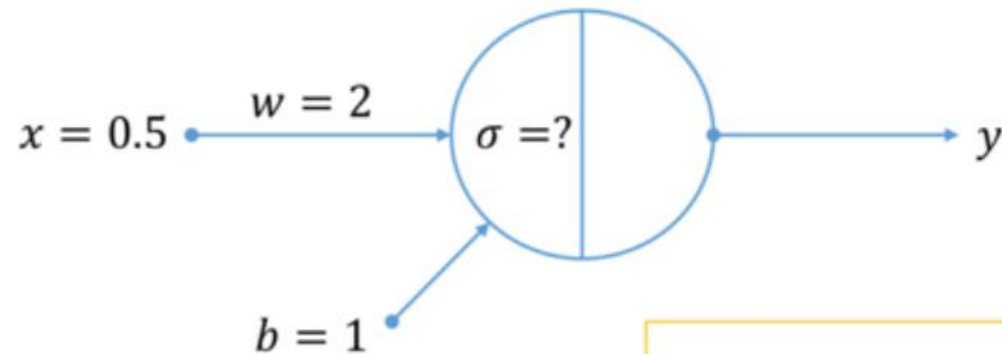
# 뉴런 연산

- 뉴런 식



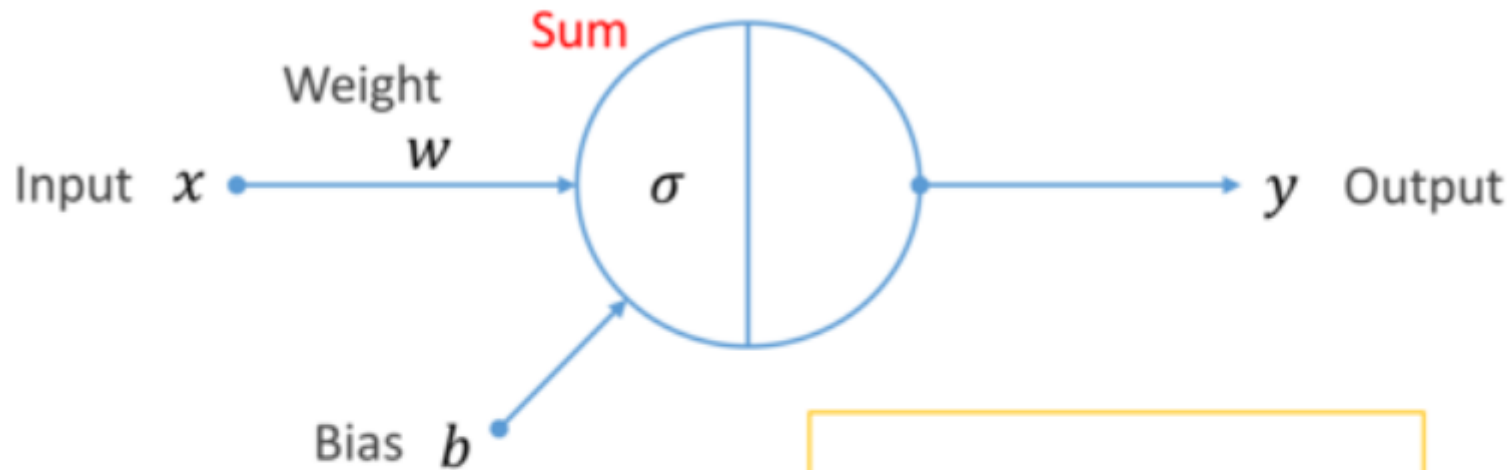
$$\sigma = w \cdot x + b$$

- 가중치와 편향



$$\begin{aligned}\sigma &= w \cdot x + b \\ &= 2 \cdot 0.5 + 1 \\ &= 2\end{aligned}$$

# 행렬 곱 연산



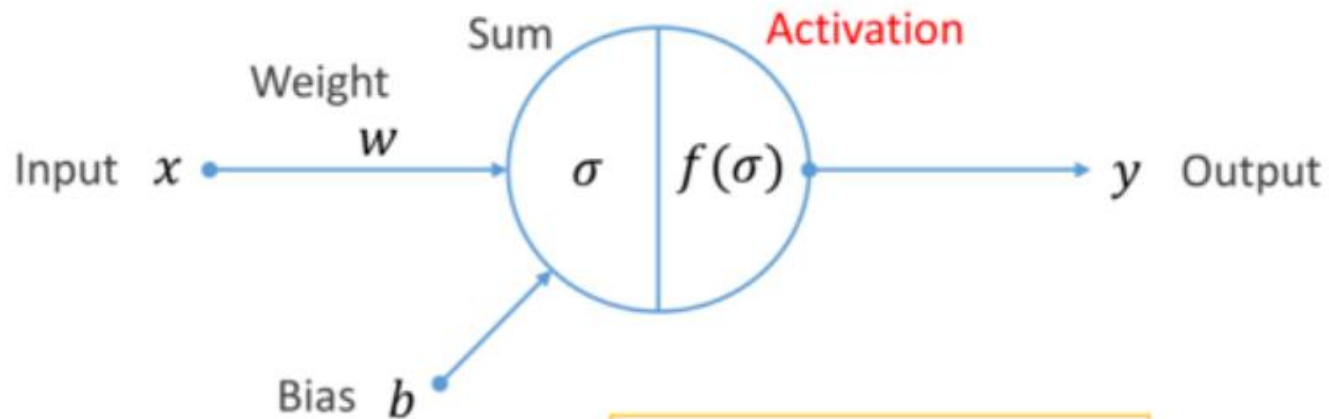
$$\begin{aligned}\sigma &= w \cdot x + b \\ &= [w \ b] \begin{bmatrix} x \\ 1 \end{bmatrix}\end{aligned}$$

행렬의 곱

# 활성화

- 활성화 함수

- 뉴런의 출력 값을 정하는 함수



$$\sigma = w \cdot x + b$$

$$f(\sigma) = f(w \cdot x + b)$$

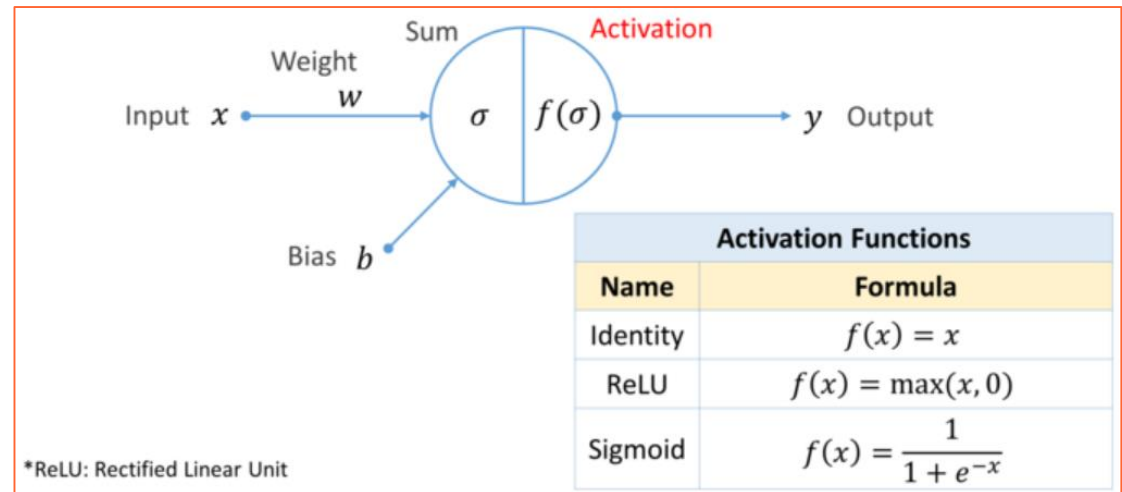
# 활성화 함수 ReLU, sigmoid

## • ReLU(교재 p43)

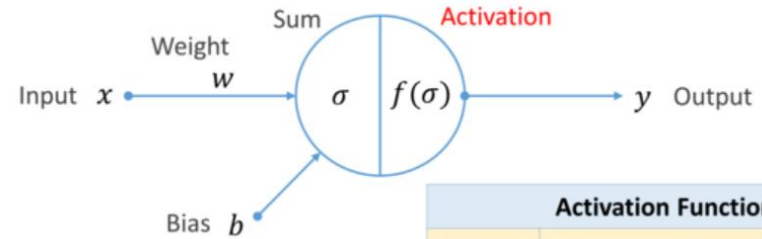
- Rectified(정류된) Linear Unit(선형 함수,  $y=x$ 를 의미)
  - 선형 함수를 정류하여 0 이하는 모두 0으로 한 함수
  - $\max(x, 0)$ 
    - 양수만 사용
- 2010년 이후
  - 층이 깊어질수록(deep) 많이 활용
    - 양수를 그대로 반환하므로 값의 왜곡이 적어지는 효과
  - 토론토 대학 힌트 교수

## • Sigmoid

- s자 형태의 곡선이라는 의미
  - 예전에 많이 사용



# 다양한 활성화 함수 종류



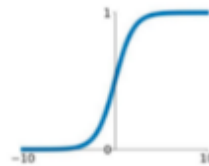
Activation Functions	
Name	Formula
Identity	$f(x) = x$
ReLU	$f(x) = \max(x, 0)$
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$

\*ReLU: Rectified Linear Unit

## Activation Functions

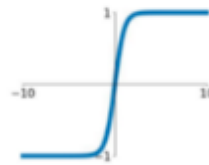
### Sigmoid

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



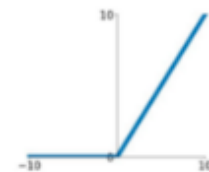
### tanh

$$\tanh(x)$$



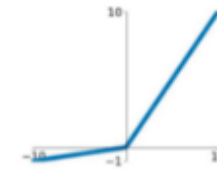
### ReLU

$$\max(0, x)$$



### Leaky ReLU

$$\max(0.1x, x)$$

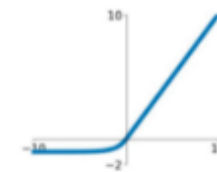


### Maxout

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

### ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Different Activation Functions and their Graphs



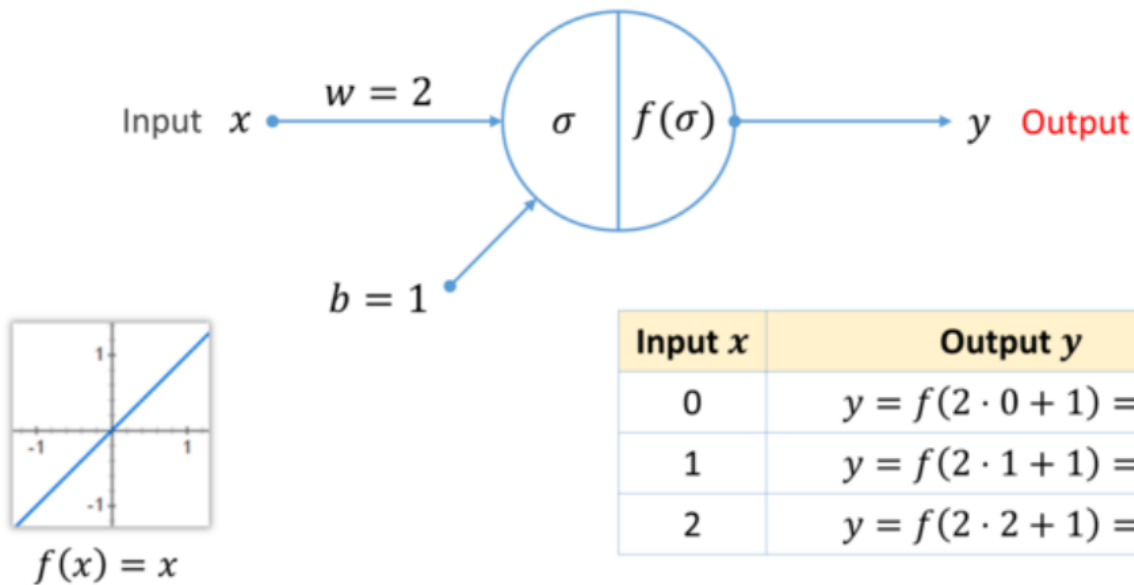
# 입출력의 예

## 출력 함수로

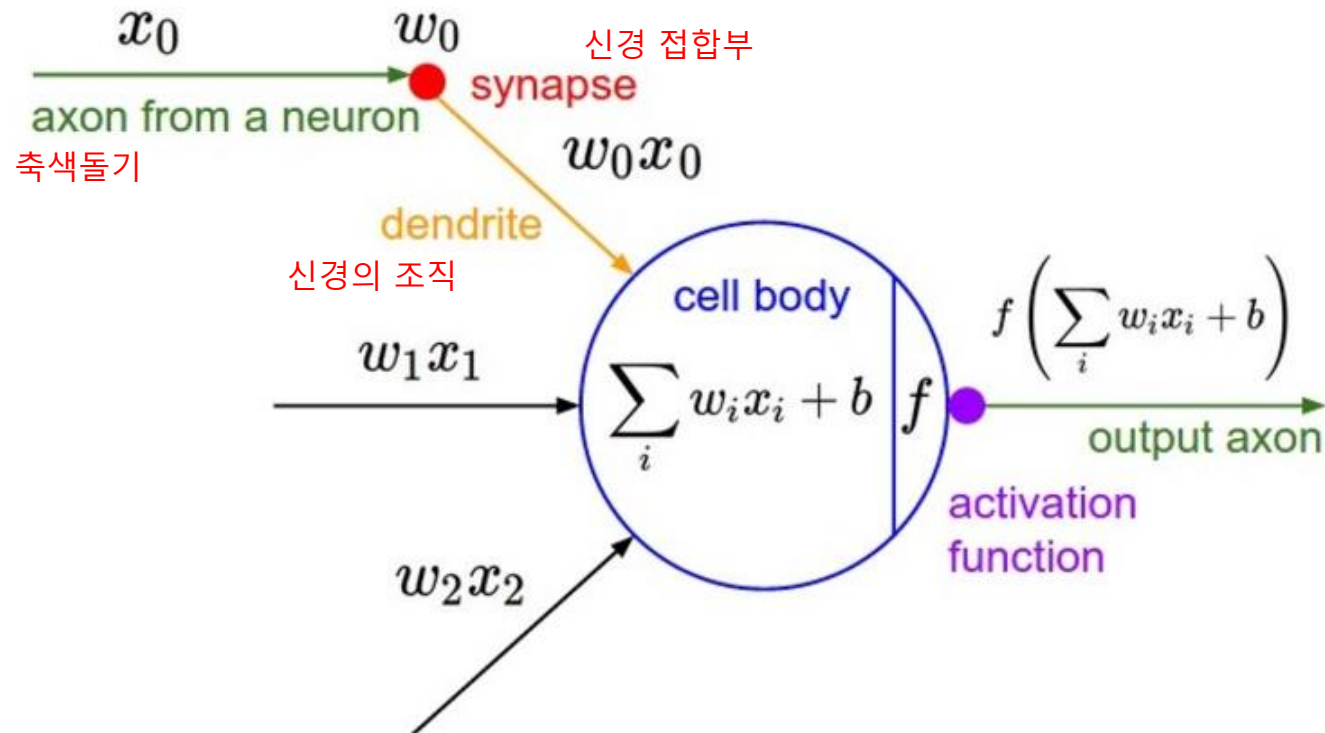
- 동일(identity) 함수(또는 리니어 함수)를 적용

$$y = f(\sigma) = f(w \cdot x + b) = w \cdot x + b$$

with **identity** (or **linear**) activation functions



# 일반화된 인공신경망



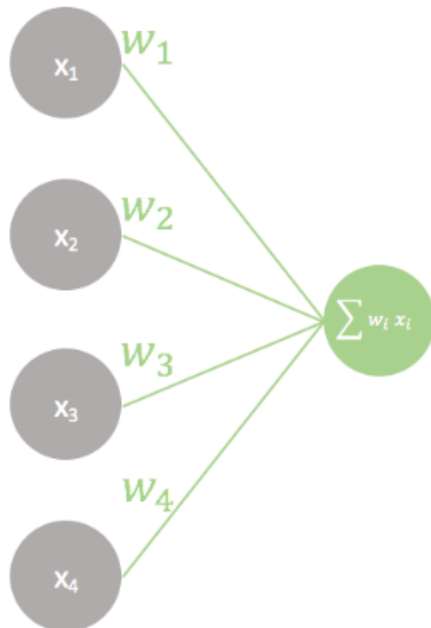
# 활성화 함수와 편향

## • 결과 값이 임계 값 역할

- 결과가 임계 값 이상이면 활성화
- 결과가 임계 값 미만이면 비활성화

Input layer

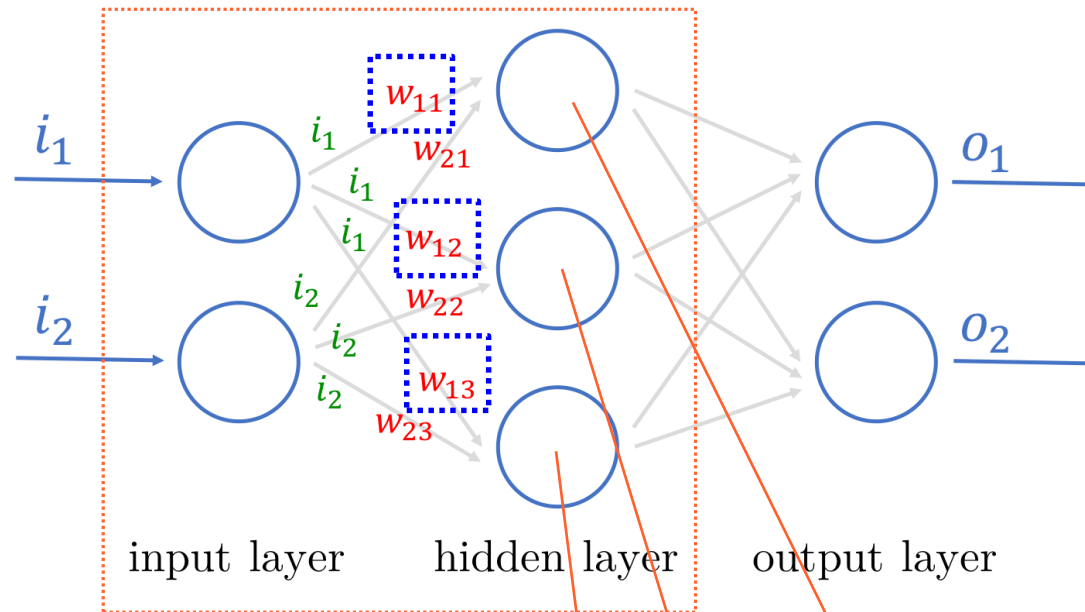
Output layer



Perceptron Unit

$$f_w(x) = \left\{ \begin{array}{l} \sum w_i x_i \geq \theta \rightarrow \text{neuron fires} \\ \sum w_i x_i < \theta \rightarrow \text{neuron doesn't fire} \end{array} \right\}$$

# 입력 2개, 출력 3개인 신경망 연산

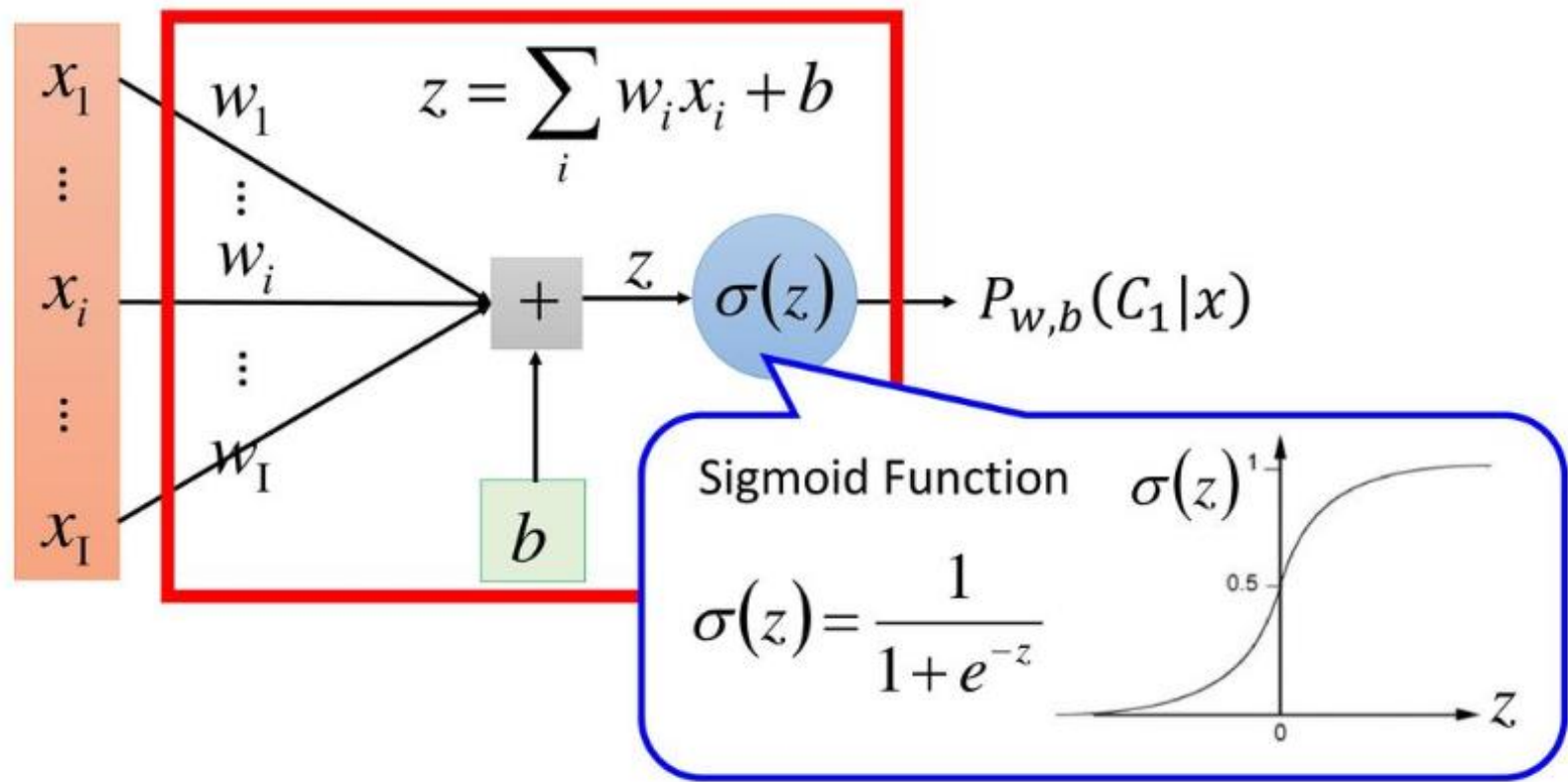


$$\begin{bmatrix} w_{11} & w_{21} \\ w_{12} & w_{22} \\ w_{13} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = \begin{bmatrix} (w_{11} \times i_1) + (w_{21} \times i_2) \\ (w_{12} \times i_1) + (w_{22} \times i_2) \\ (w_{13} \times i_1) + (w_{23} \times i_2) \end{bmatrix}$$

# 인공신경망의 시그모이드 함수

## • 활성화 함수의 예

- 시그모이드 함수
  - 출력 값이 (0~1)



# 가중치

- $3 \times 3$ 의 가중치 실수

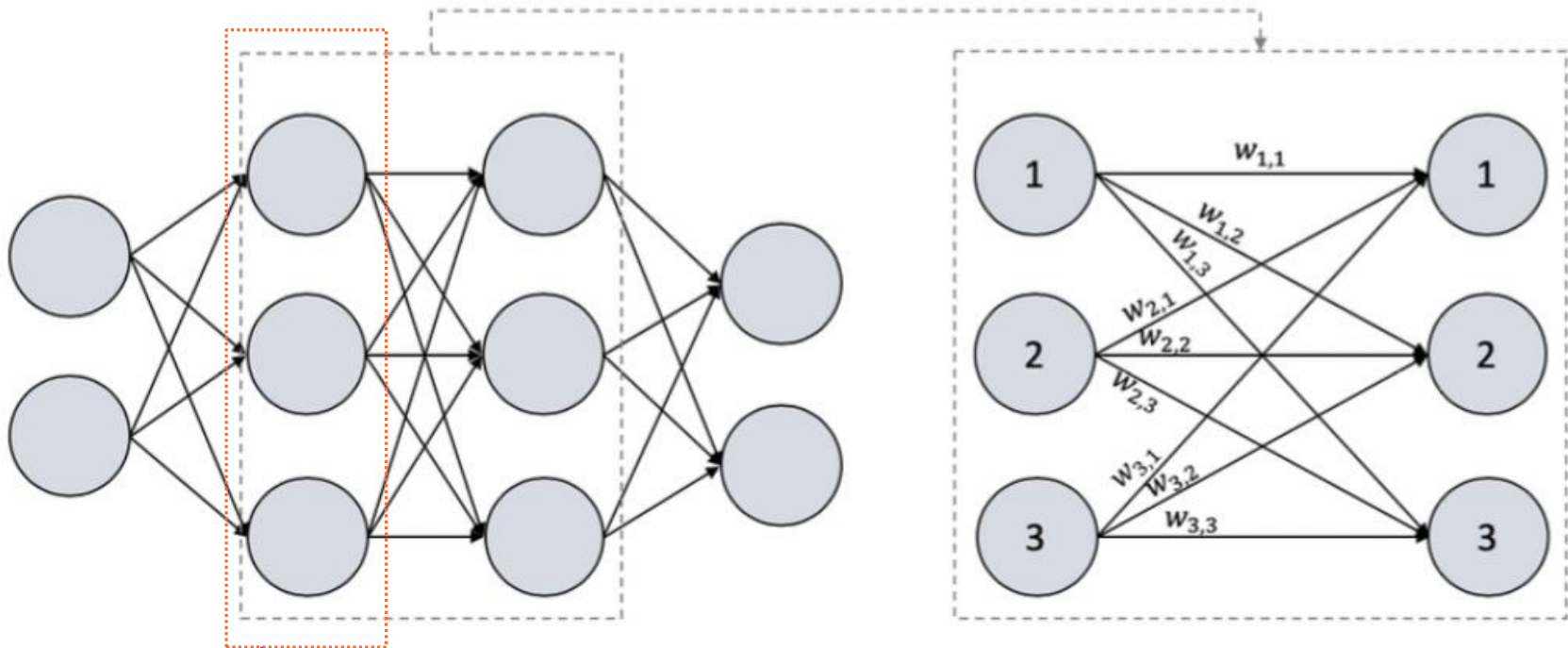


Figure 3. Connections of neurons between 2 layers

층, layer라고 부름

# 인공 신경망 행렬 연산

# 입력의 특징( $x_1 x_2 x_3 \dots x_i$ )과 입력의 자료 수

- 특징  $n$ 개가 있는 뉴런 신경망에서 하나의 출력 계산

✓ 샘플 수 1개에 대한 계산

Geektail/

$$x = [x_1 \quad x_2 \quad \dots \quad x_n] \quad w^T = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

$(1 \times n)$   $(n \times 1)$

$$xw^T = w_1x_1 + w_2x_2 + \dots + w_nx_n$$

$$= z$$

$(1 \times 1)$  ← 스칼라

$n$ : MNIST 손글씨에서 손글씨 이미지 하나의 픽셀 수인 786(28 x 28)을 의미

✓ 샘플 수  $s$ 개에 대한 계산

$$X = \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{s1} & \dots & x_{sn} \end{bmatrix} \quad w^T = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

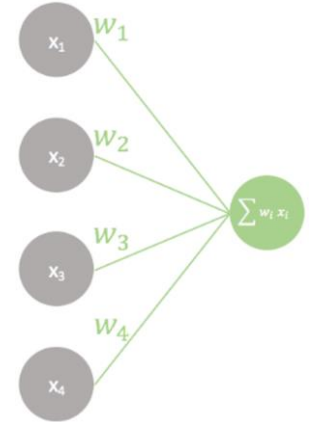
$(s \times n)$   $(n \times 1)$

$$Xw^T = \begin{bmatrix} w_1x_{11} + w_2x_{12} + \dots + w_nx_{1n} \\ w_1x_{21} + w_2x_{22} + \dots + w_nx_{2n} \\ \vdots \\ w_1x_{s1} + w_2x_{s2} + \dots + w_nx_{sn} \end{bmatrix}$$

$$= \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_s \end{bmatrix} = z$$

$(s \times 1)$  ← 벡터

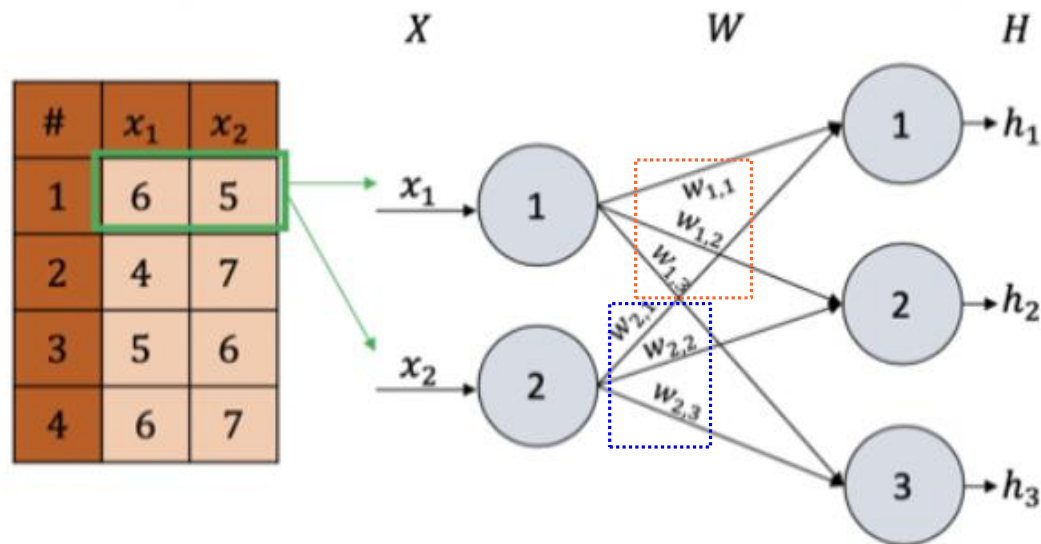
$s$ : MNIST 손글씨에서 훈련 데이터 6만개에서 6만을 의미





# 신경망 행렬 계산

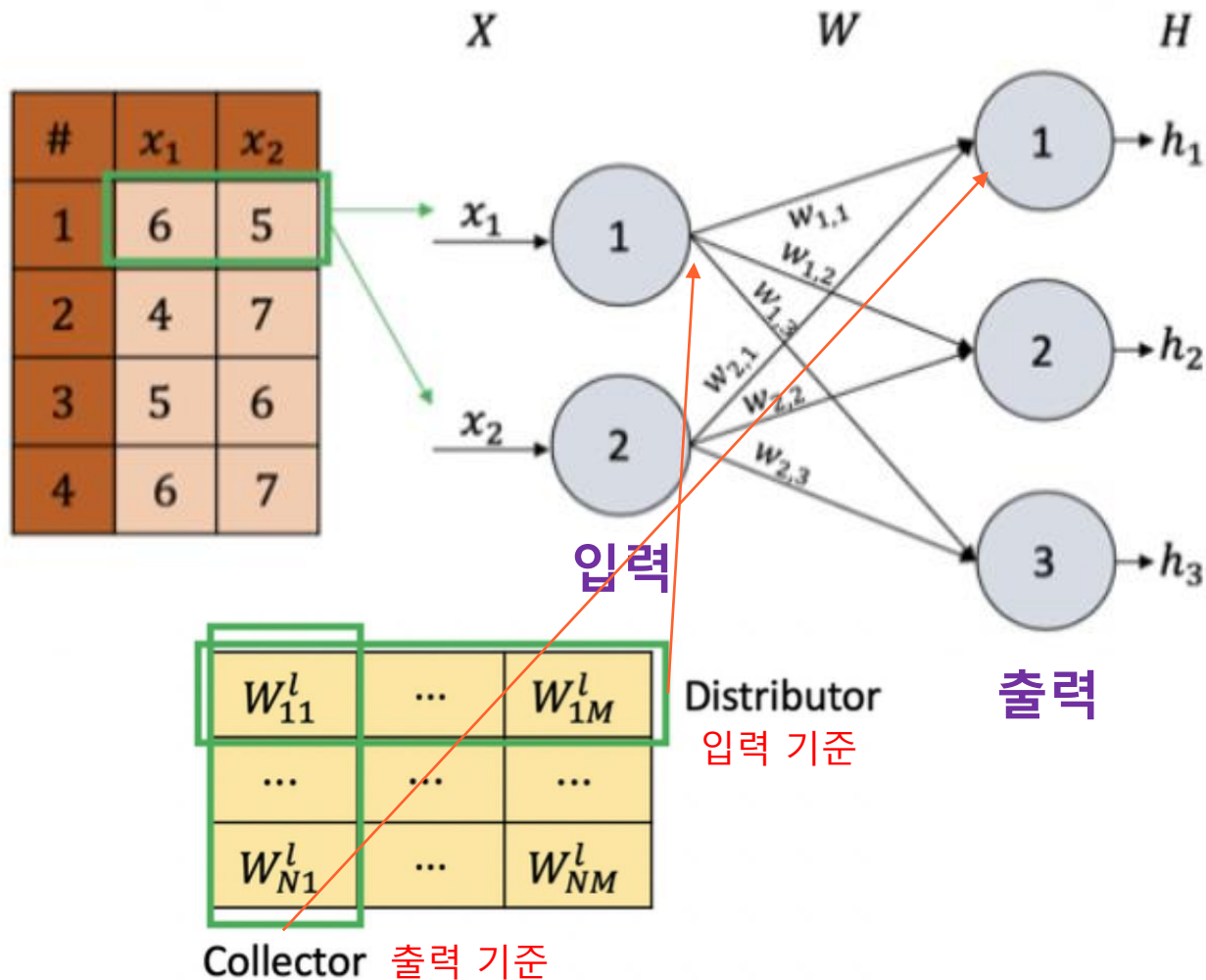
- 특징 2개
  - 샘플 수 4



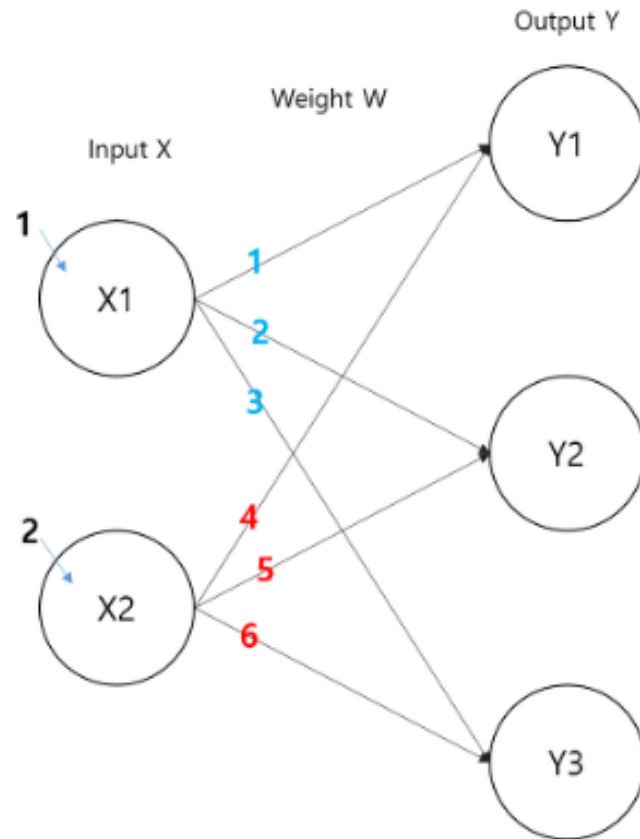
$$\begin{matrix} (1 \times 2) \\ \begin{bmatrix} x_1 & x_2 \end{bmatrix} \end{matrix} \cdot \begin{matrix} (2 \times 3) \\ \begin{bmatrix} W_{11}^1 & W_{12}^1 & W_{13}^1 \\ W_{21}^1 & W_{22}^1 & W_{23}^1 \end{bmatrix} \end{matrix} = \begin{matrix} (1 \times 3) \\ \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} \end{matrix}$$

$X \cdot W = H$

# 뉴런 계산



## 계산 사례



$$\begin{array}{rclcl}
 \mathbf{X} & * & \mathbf{W} & = & \mathbf{Y} \\
 1 \times 2 & * & 2 \times 3 & = & 1 \times 3 \\
 \\ 
 (1 \ 2) & & \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} & & (9 \ 12 \ 15)
 \end{array}$$

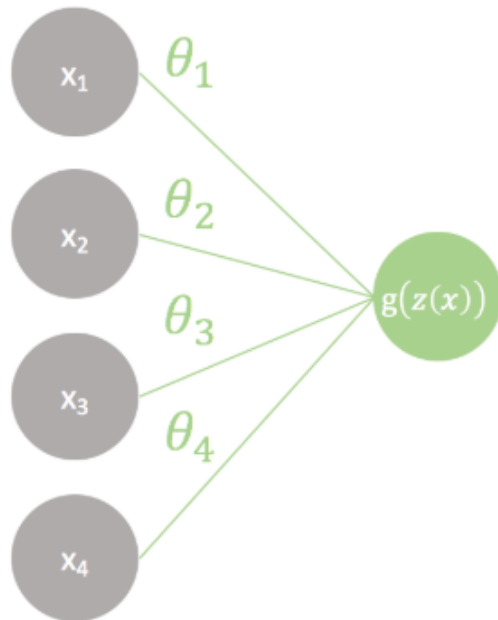
© sacko

# 하나의 출력 뉴런 연산

- 활성화 함수로 시그모이드 함수 적용

Input layer

Output layer



$$z(x) = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

# 층과 가중치

- 뉴런 층과 가중치 층

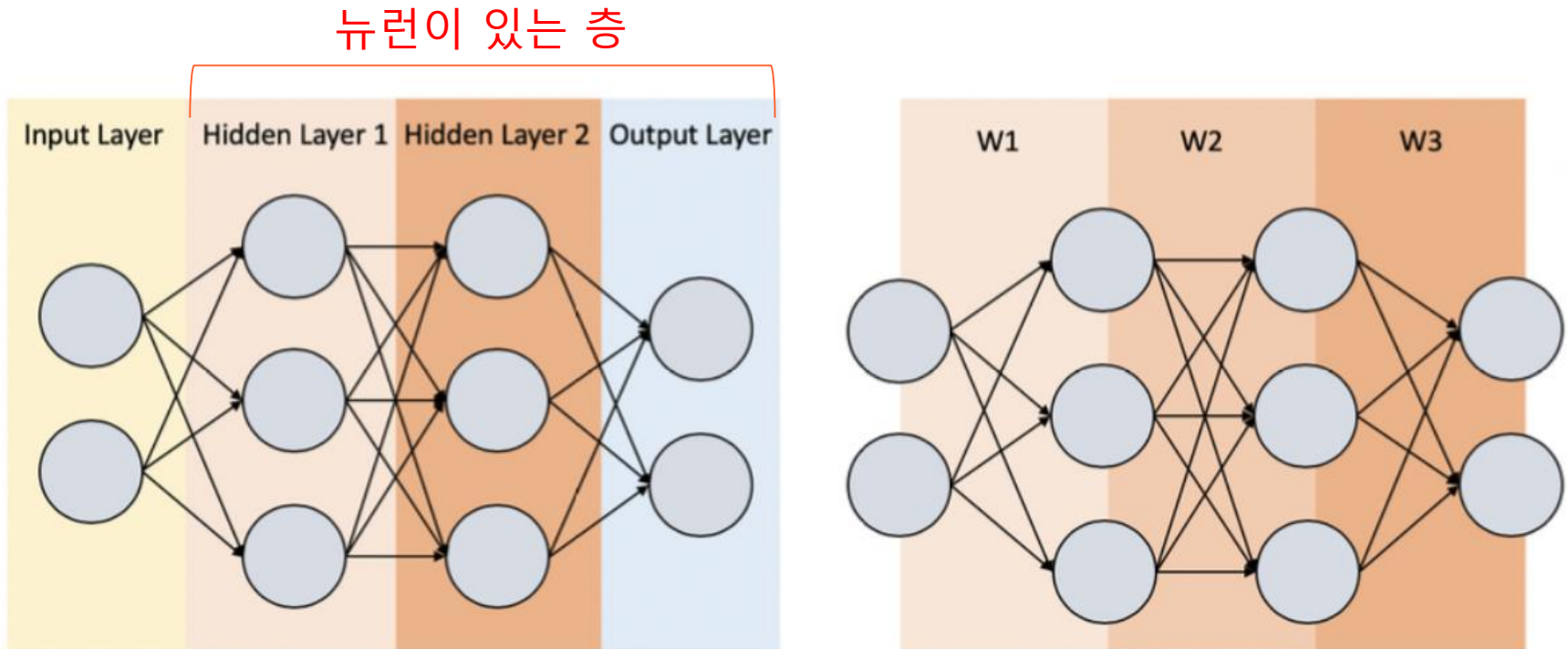
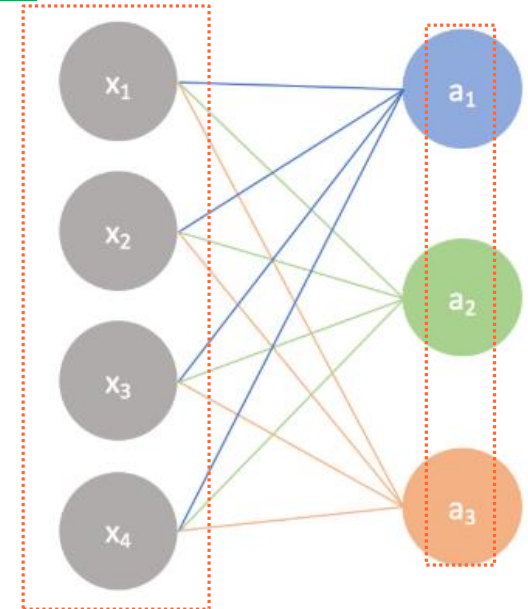


Figure 7. Layers of neuron vs Layers of weights

# 행렬의 다른 표현

- 입력을 오른쪽 행렬에 배치
- 가중치는 왼쪽 행렬에 배치
- 곱의 순서도 변환

Using multiple observations



$$\begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \begin{bmatrix} \text{Observation 1} & \text{Observation 2} & \text{Observation 3} & \text{Observation 4} \\ x_1 & x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 & x_2 \\ x_3 & x_3 & x_3 & x_3 \\ x_4 & x_4 & x_4 & x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} \xrightarrow{\text{activation}} \begin{bmatrix} \text{Observation 1} & \text{Observation 2} & \text{Observation 3} & \text{Observation 4} \\ a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \end{bmatrix}$$

# 활성화 함수 그리기

# 실습 파일

- 파일 생성
  - 06-neuron-activation-dense.ipynb



# 자연수와 자연수의 지수 승

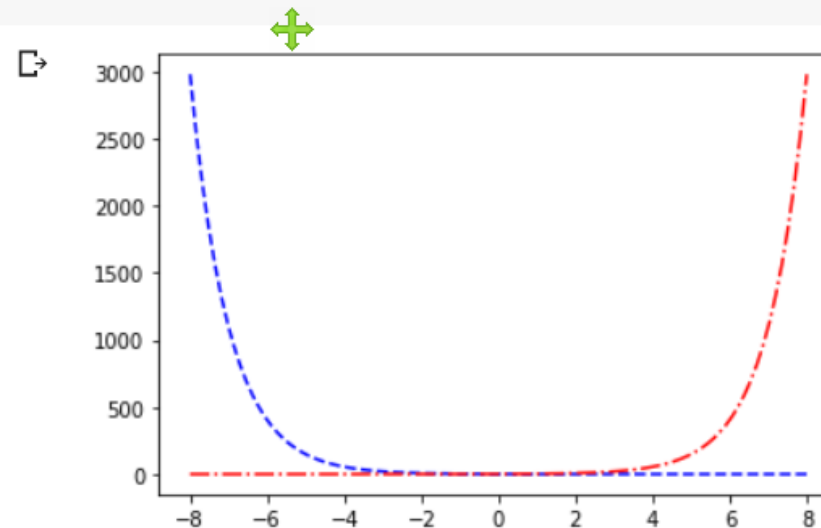
- $e$ 
  - 자연수, 오일러 수
  - 2.71828
- $y = e^{-x}$
- $y = e^x$

```
[76] import numpy as np
      np.e
```

```
↳ 2.718281828459045
```

```
[77] import numpy as np
      import matplotlib.pyplot as plt

      plt.figure(figsize=(6, 4))
      x = np.linspace(-8, 8, 100)
      plt.plot(x, np.exp(-x), 'b--')
      _ = plt.plot(x, np.exp(x), 'r-.')
```



# 시그모이드 함수

## • S자 곡선

- (0, 1) 사이의 값

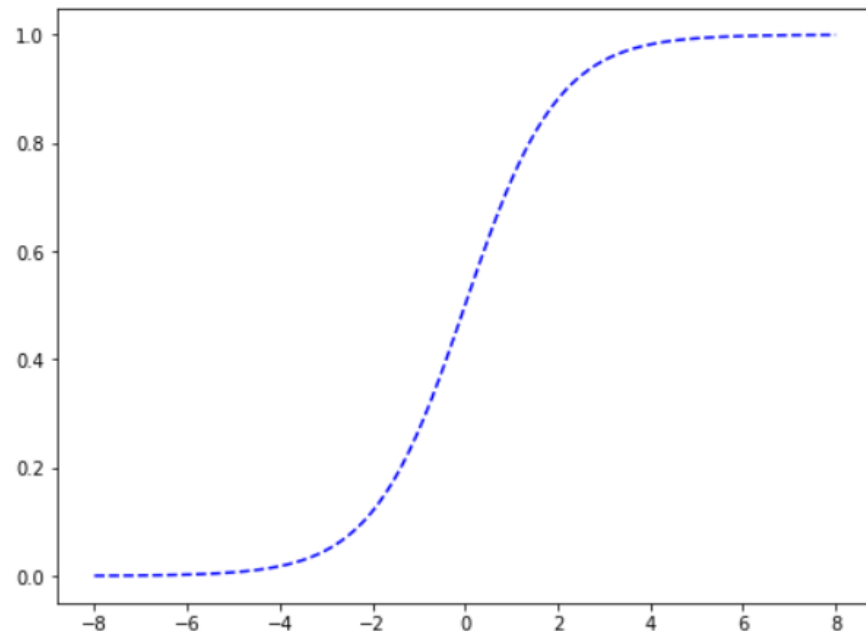
$$h(x) = \frac{1}{1 + e^{-x}}$$

```
[5] np.e
```

```
↳ 2.718281828459045
```

```
[44] 1 import numpy as np
      2 import matplotlib.pyplot as plt
      3
      4 def sigm_func(x): # sigmoid 함수
      5     return 1 / (1 + np.exp(-x))
      6
      7 # 시그모이드 함수 그리기
      8 plt.figure(figsize=(8, 6))
      9 x = np.linspace(-8, 8, 100)
     10 plt.plot(x, sigm_func(x), 'b--')
```

```
↳ [<matplotlib.lines.Line2D at 0x7f93b4130cc0>]
```



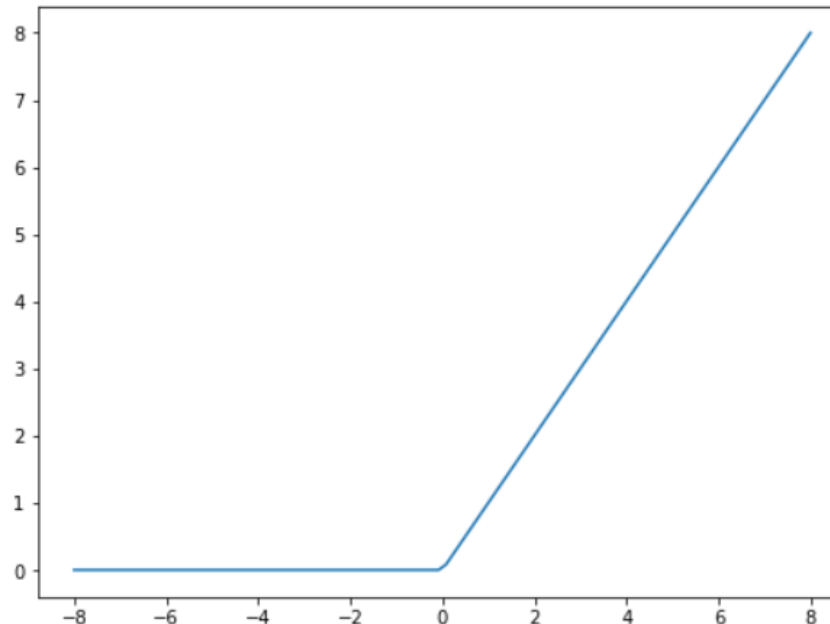
# ReLU 함수

- **x**
  - 0, 음수면 0
  - 양수면 x

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

```
[45] 1 import numpy as np
      2 import matplotlib.pyplot as plt
      3
      4 def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수
      5     return np.maximum(0, x)
      6     #return (x>0)*x # same
      7
      8 # ReLU 함수 그리기
      9 plt.figure(figsize=(8, 6))
     10 x = np.linspace(-8, 8, 100)
     11 plt.plot(x, relu_func(x))
```

☞ [<matplotlib.lines.Line2D at 0x7f93b409b748>]



# 시그모이드 ReLU 함께 그리기

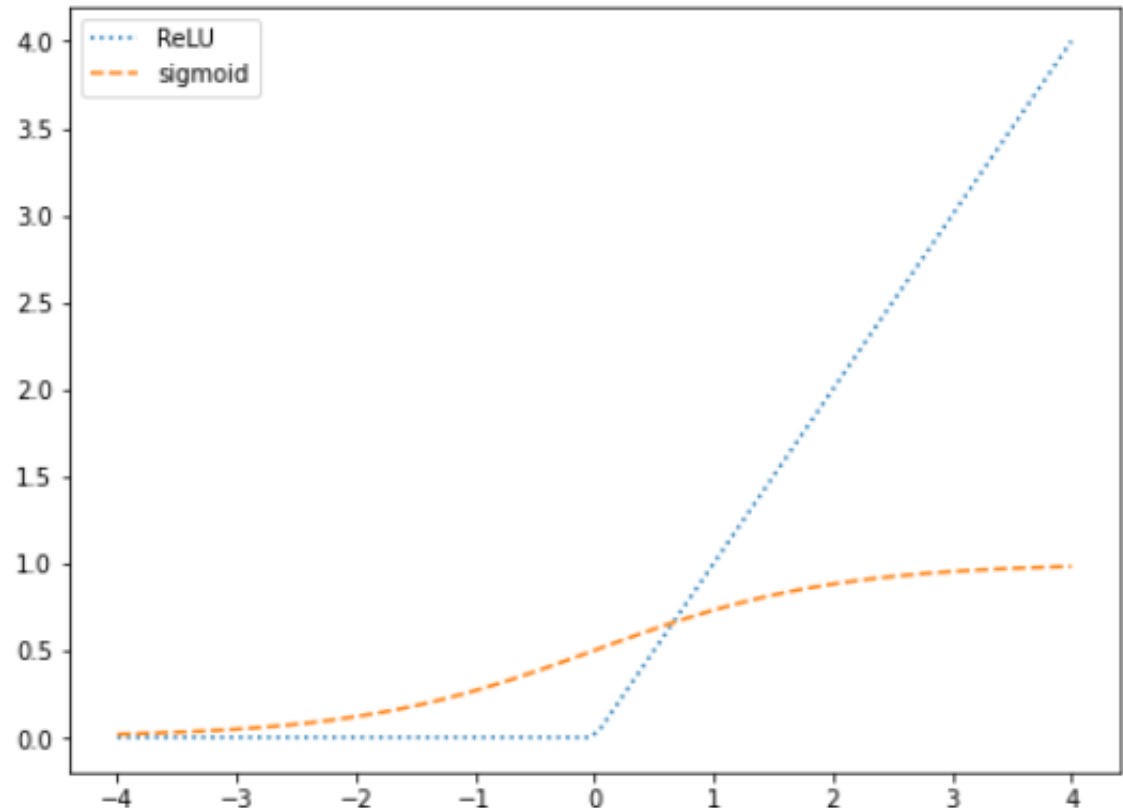
```
import numpy as np
import matplotlib.pyplot as plt

# ReLU(Rectified Linear Unit
# (정류된 선형 유닛) 함수
def relu_func(x):
    return np.maximum(0, x)
    #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수
    return 1 / (1 + np.exp(-x))

# 그래프 그리기
plt.figure(figsize=(8, 6))
x = np.linspace(-4, 4, 100)
y = np.linspace(-0.2, 2, 100)

plt.plot(x, relu_func(x), linestyle=':', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```



# 다양한 활성화 함수

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

def identity_func(x): # 항등함수
    return x

def linear_func(x): # 1차함수
    return 1.5 * x + 1 # a기울기(1.5), y절편b(1) 조정가능

def tanh_func(x): # TanH 함수
    return np.tanh(x)

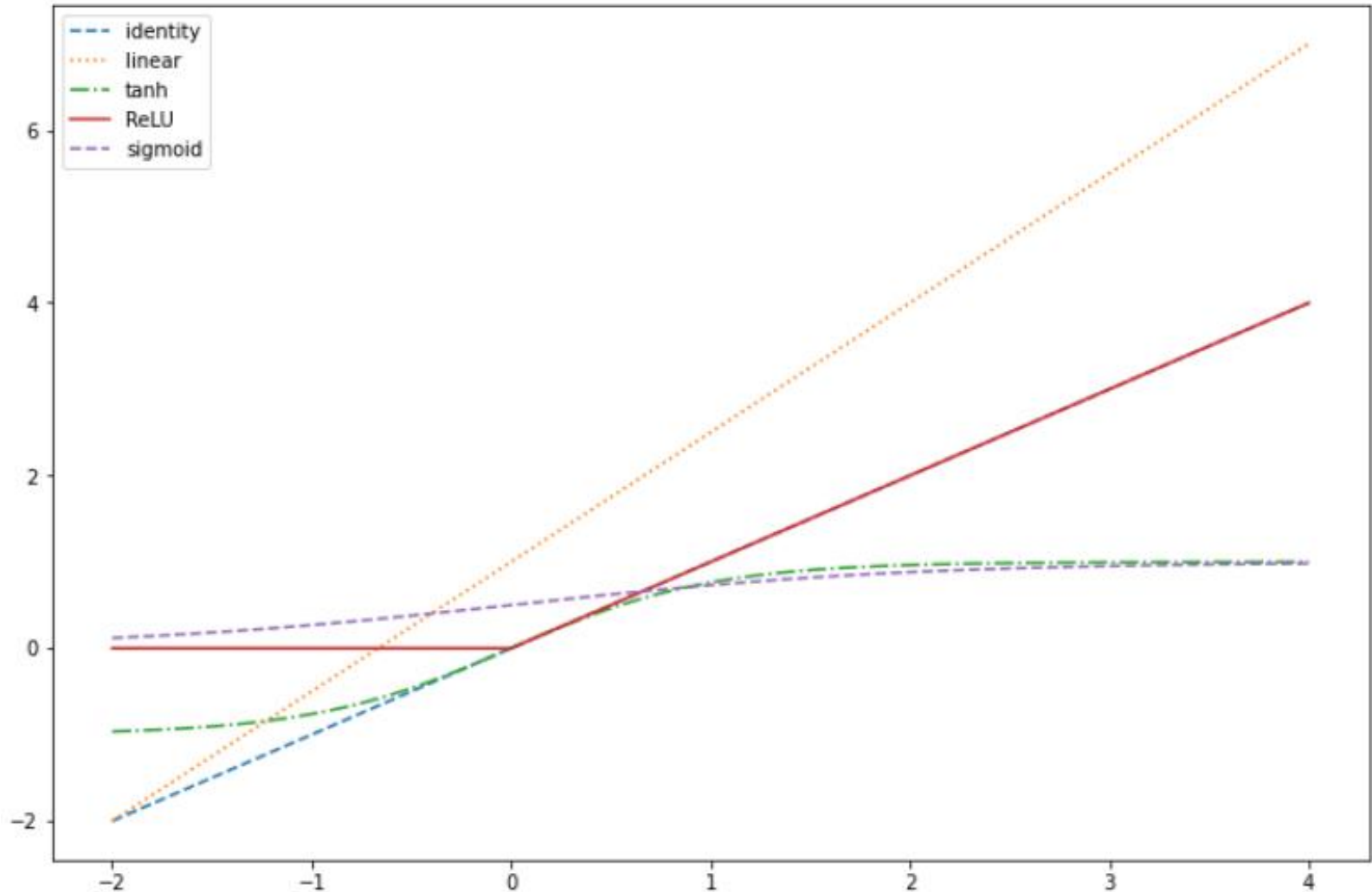
def relu_func(x): # ReLU(Rectified Linear Unit, 정류된 선형 유닛) 함수
    return np.maximum(0, x)
    #return (x>0)*x # same

def sigm_func(x): # sigmoid 함수
    return 1 / (1 + np.exp(-x))

# 그래프 그리기
plt.figure(figsize=(12, 8))
x = np.linspace(-2, 4, 100)

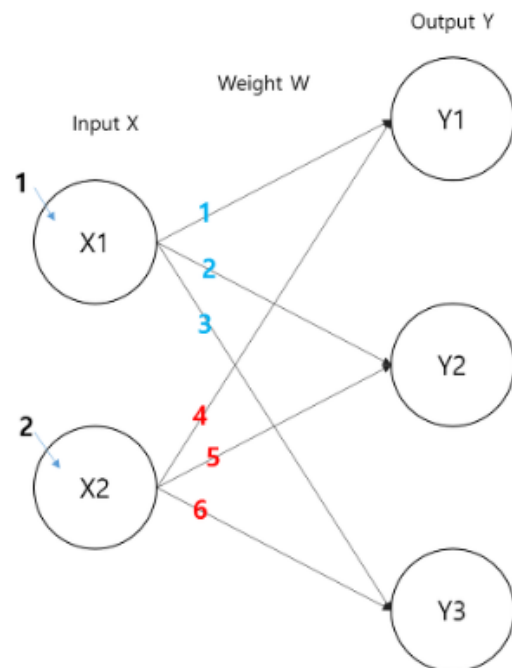
plt.plot(x, identity_func(x), linestyle='--', label="identity")
plt.plot(x, linear_func(x), linestyle=':', label="linear")
plt.plot(x, tanh_func(x), linestyle='-.', label="tanh")
plt.plot(x, relu_func(x), linestyle='-', label="ReLU")
plt.plot(x, sigm_func(x), linestyle='--', label="sigmoid")
plt.legend(loc='upper left')
```

# 활성화 함수 결과



# 인공 신경망 행렬 연산 코드

# 계산 사례



$$\begin{matrix} \mathbf{X} & * & \mathbf{W} & = & \mathbf{Y} \\ 1 \times 2 & * & 2 \times 3 & = & 1 \times 3 \end{matrix}$$

$$\begin{pmatrix} 1 & 2 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} = \begin{pmatrix} 9 & 12 & 15 \end{pmatrix}$$

## ▼ 뉴런의 행렬 연산

```
[14] x = [[1, 2]]
      w = [[1, 2, 3], [4, 5, 6]]

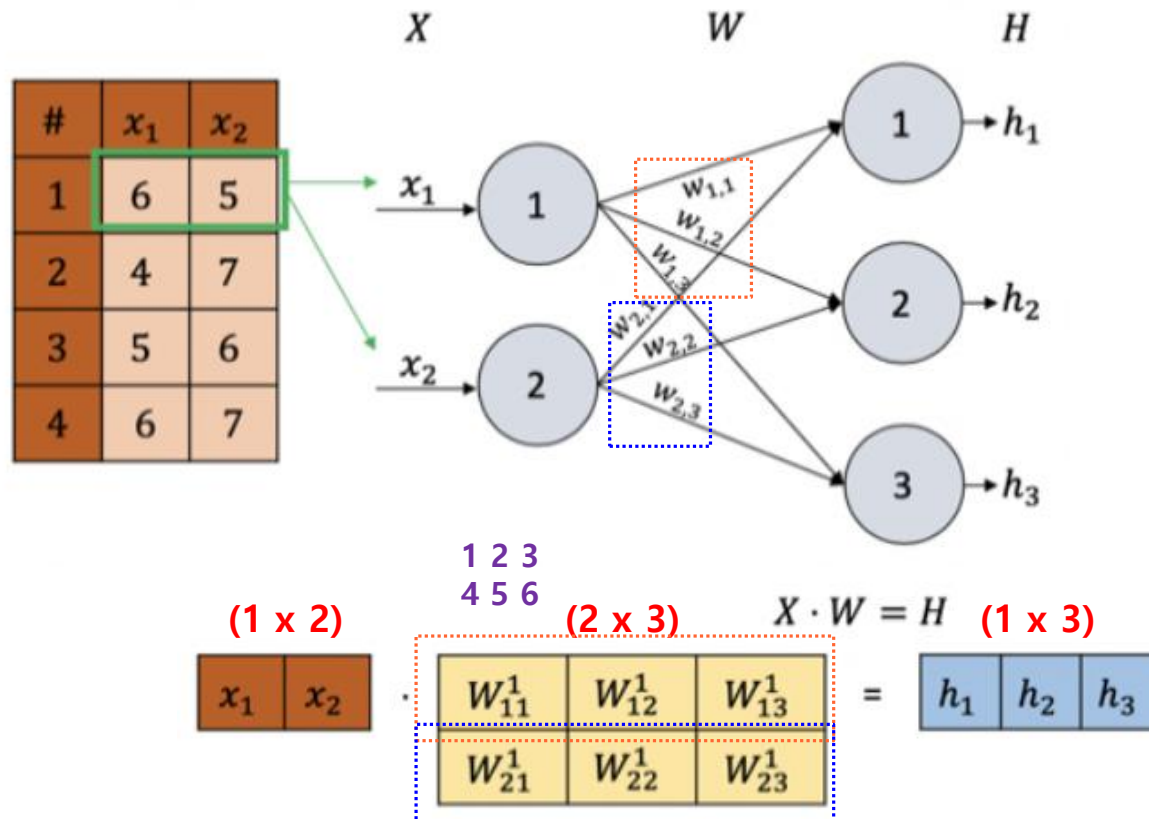
      y = tf.matmul(x, w)
      y.numpy()
```

```
↳ array([[ 9, 12, 15]], dtype=int32)
```



# 신경망 행렬 계산

- 특징 2개
  - 샘플 수 4



## 특징 2, 샘플 수 4개의 행렬 연산

```
[15] x = [[6, 5]]  
     w = [[1, 2, 3], [4, 5, 6]]  
  
     y = tf.matmul(x, w)  
     y.numpy()
```

```
↳ array([[26, 37, 48]], dtype=int32)
```

```
[16] x = [[6, 5], [4, 7], [5, 6], [6, 7]]  
     w = [[1, 2, 3], [4, 5, 6]]  
  
     y = tf.matmul(x, w)  
     y.numpy()
```

```
↳ array([[26, 37, 48],  
         [32, 43, 54],  
         [29, 40, 51],  
         [34, 47, 60]], dtype=int32)
```

# 행렬의 순서를 바꾼 계산

- 가중치와 입력 값의 순서도 수정

```
[78] #x = [[6, 5], [4, 7], [5, 6], [6, 7]]
      #w = [[1, 2, 3], [4, 5, 6]]
```

```
w = [[1, 4], [2, 5], [3, 6]]
x = [[6, 4, 5, 6], [5, 7, 6, 7]]
```

```
y = tf.matmul(w, x)
y.numpy()
```

```
↳ array([[26, 32, 29, 34],
          [37, 43, 40, 47],
          [48, 54, 51, 60]], dtype=int32)
```

Using multiple observations

$$\begin{array}{c} \begin{bmatrix} w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \\ w_1 & w_2 & w_3 & w_4 \end{bmatrix} \end{array} \begin{array}{c} \text{Observation 1} \\ \text{Observation 2} \\ \text{Observation 3} \\ \text{Observation 4} \end{array} \begin{bmatrix} x_1 & x_1 & x_1 & x_1 \\ x_2 & x_2 & x_2 & x_2 \\ x_3 & x_3 & x_3 & x_3 \\ x_4 & x_4 & x_4 & x_4 \end{bmatrix} + \begin{bmatrix} b \\ b \\ b \end{bmatrix} \xrightarrow{\text{activation}} \begin{array}{c} \begin{bmatrix} a_1 & a_1 & a_1 & a_1 \\ a_2 & a_2 & a_2 & a_2 \\ a_3 & a_3 & a_3 & a_3 \end{bmatrix} \end{array} \begin{array}{c} \text{Observation 1} \\ \text{Observation 2} \\ \text{Observation 3} \\ \text{Observation 4} \end{array}$$

Python

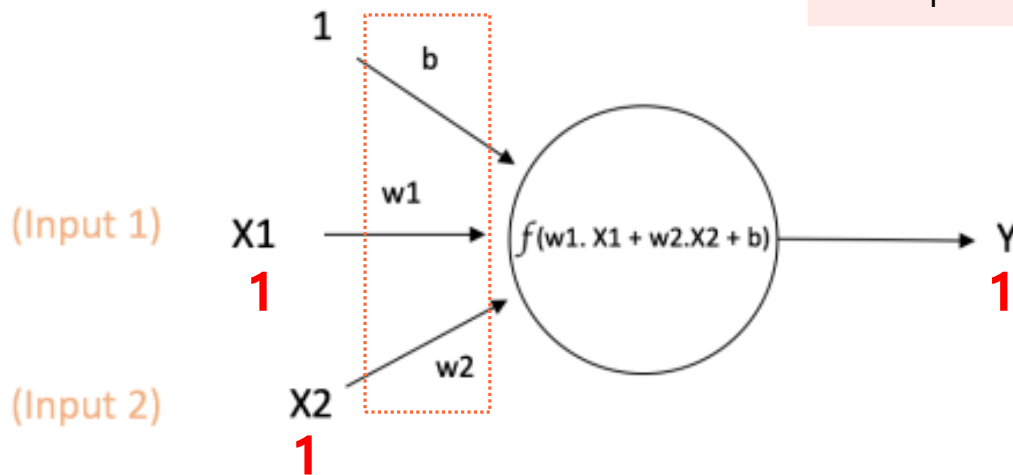
# 논리 게이트 AND OR XOR 신경망 구현

# AND 게이트 구현

## • 뉴런 구조

- 입력 2개, 편향, 출력 1
- 구할 값
  - 가중치 2개와 편향 1개

x1	x2	y
0	0	0
0	1	0
1	0	0
1	1	1



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

(Output)

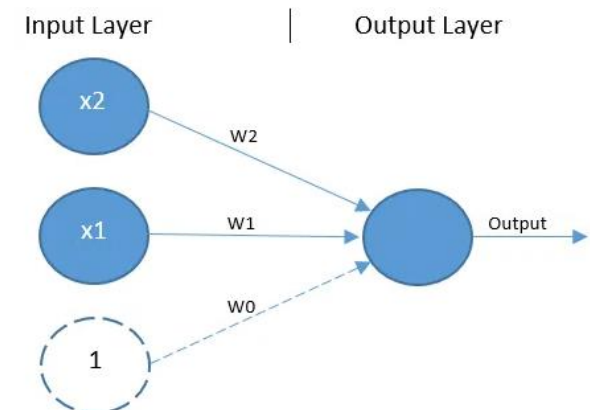


Figure 2: Single Layer Perceptron Network

```

▶ # tf.keras 를 이용한 AND 네트워크 계산
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[1], [0], [0], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, activation='sigmoid', input_shape=(2,)),
])

model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()

```

Model: "sequential\_4"

Layer (type) ➕	Output Shape	Param #
dense_6 (Dense)	(None, 1)	3
Total params: 3		
Trainable params: 3		
Non-trainable params: 0		

```

▶ history = model.fit(x, y, epochs=400, batch_size=1)
4/4 [=====] - 0s 1ms/step - loss: 0.0145
Epoch 372/400
4/4 [=====] - 0s 2ms/step - loss: 0.0144
Epoch 373/400
4/4 [=====] - 0s 2ms/step - loss: 0.0144
Epoch 374/400

```

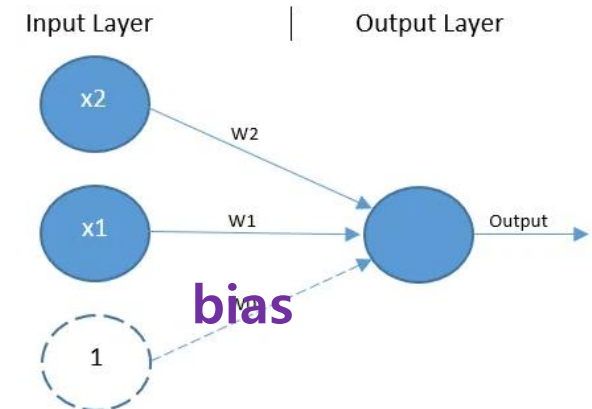
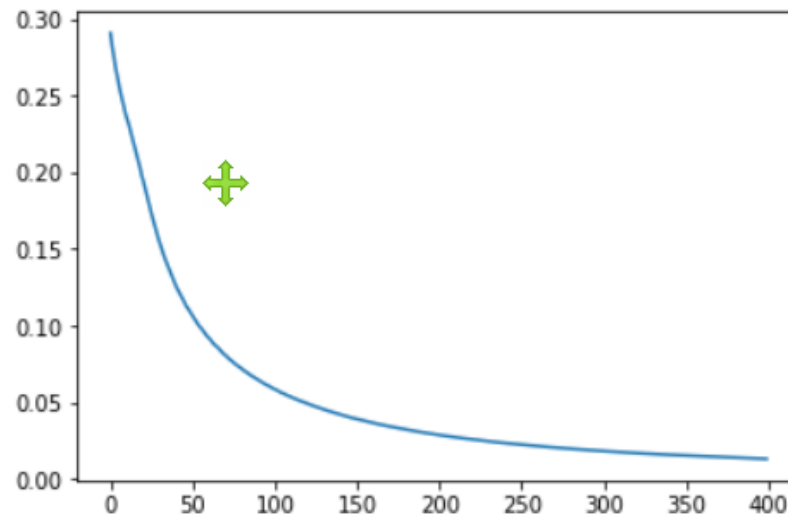


Figure 2: Single Layer Perceptron Network

# 손실 값 그래프와 결과 예측

```
[54] # 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시
import matplotlib.pyplot as plt
plt.plot(history.history['loss'])
```

↳ [<matplotlib.lines.Line2D at 0x7efe712955f8>]



```
[59] model.predict(x)
```

↳ array([[0.8535386 ],  
[0.12342612],  
[0.12364785],  
[0.00339741]], dtype=float32)

# 가중치와 편향 값 알아 보기

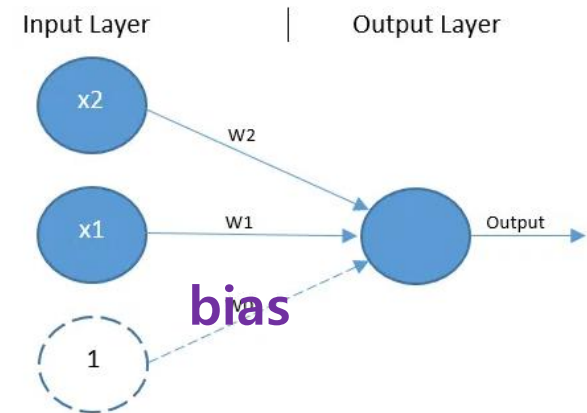


Figure 2: Single Layer Perceptron Network

```
[60] for weight in model.weights:
      print(weight)
```

```
>>> <tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>
<tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>
```

```
[61] model.weights[0]
```

```
>>> <tf.Variable 'dense_6/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[3.7209592],
       [3.723007 ]], dtype=float32)>
```

```
[62] model.weights[1]
```

```
>>> <tf.Variable 'dense_6/bias:0' shape=(1,) dtype=float32, numpy=array([-5.6813374], dtype=float32)>
```

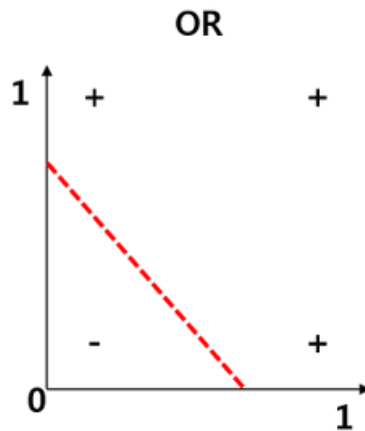


# OR 게이트 구현

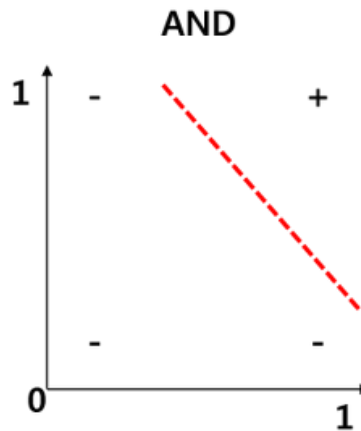
- 여러분이 직접 해 보세요.

# XOR 문제

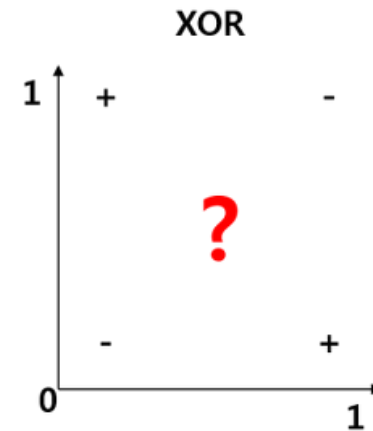
- 하나의 퍼셉트론으로는 XOR 게이트는 불가능
  - 마빈 민스키와 시모어 페퍼트가 증명
  - 첫 AI 겨울의 계기



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



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



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

# XOR 해결

- 뉴런 3 개의 2층으로 가능
  - 모델이 구해야 할 총 매개변수(가중치와 편향)
    - $3 * 2 + 3 * 1 = 9$ 개

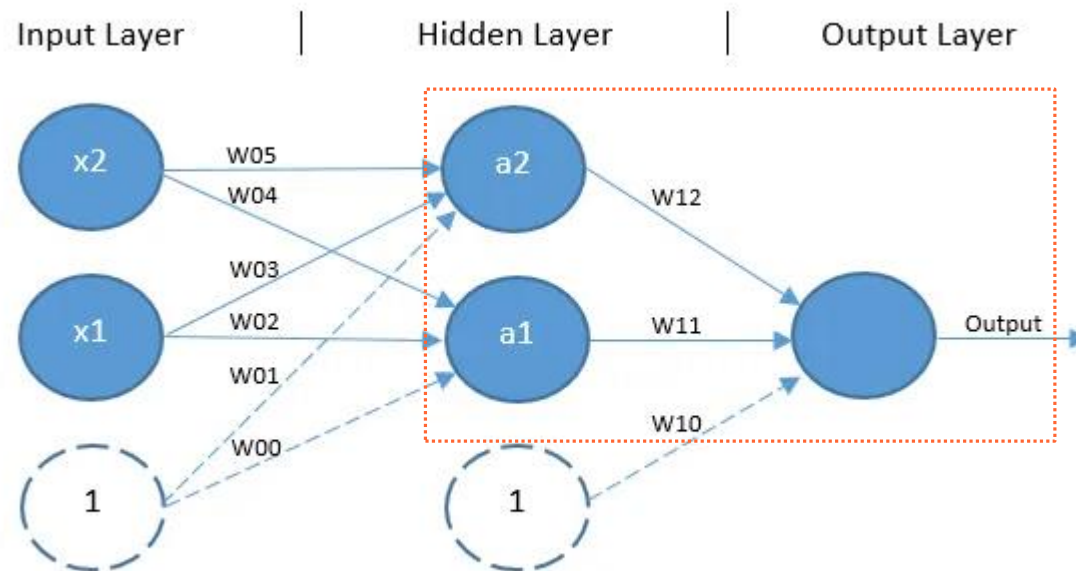


Figure 4: Multilayer Perceptron Architecture for XOR

# Sequential 모델

## • Dense 층

- 가장 기본적인 층
- 인자 units, activation
  - 뉴런 수와 활성화 함수
- 인자 input\_shape
  - 첫 번째 층에서만 정의
  - 입력의 차원을 명시
    - (2,)
    - 2개의 입력을 받는 1차원

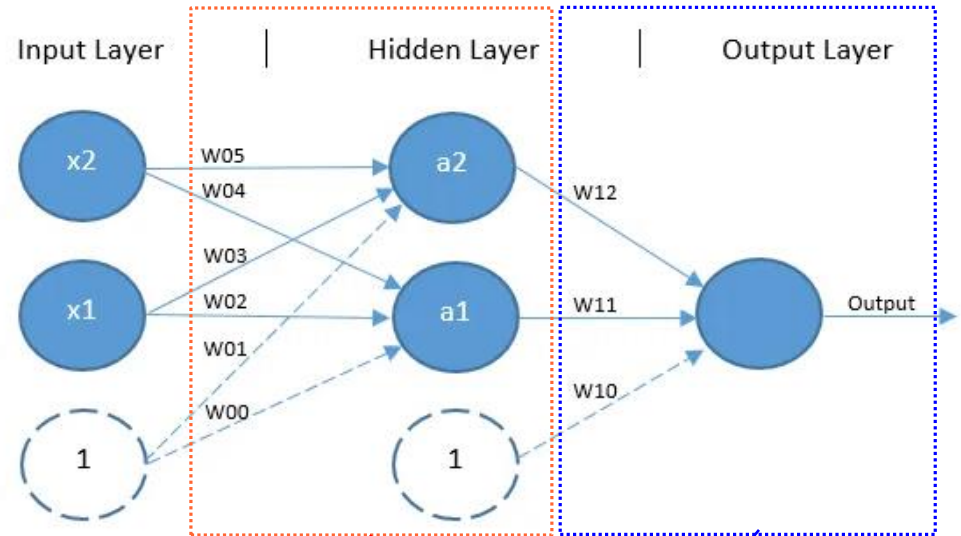


Figure 4: Multilayer Perceptron Architecture for XOR

```
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

# Sequential 모델과 딥러닝 구조

## • 입력, 은닉, 출력 층

### – 패러미터 수

#### • (입력층 뉴런 수 + 1) \* (출력층 뉴런 수)

```
x = np.array([[1, 1], [1, 0], [0, 1], [0, 0]])
y = np.array([[0], [1], [1], [0]])
```

```
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
dense_2 (Dense)	(None, 2)	6
dense_3 (Dense)	(None, 1)	3
Total params:	9	
Trainable params:	9	
Non-trainable params:	0	

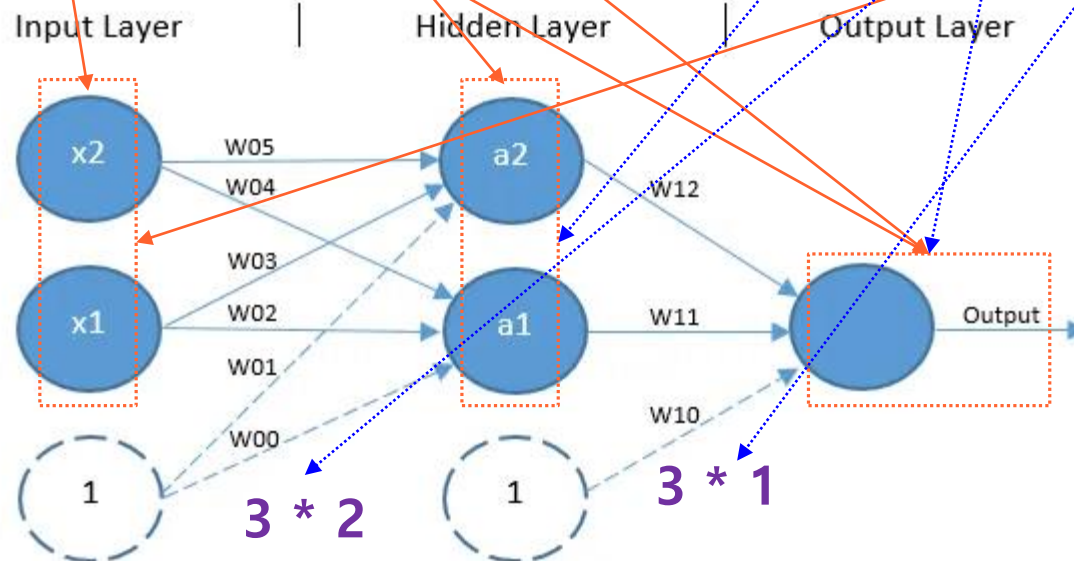


Figure 4: Multilayer Perceptron Architecture for XOR

# XOR 게이트 구현 소스

# 3.27 tf.keras 를 이용한 XOR 네트워크 계산

```
import tensorflow as tf
import numpy as np
x = np.array([[1,1], [1,0], [0,1], [0,0]])
y = np.array([[0], [1], [1], [0]])

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=2, activation='sigmoid', input_shape=(2,)),
    tf.keras.layers.Dense(units=1, activation='sigmoid')
])
```

```
model.compile(optimizer=tf.keras.optimizers.SGD(lr=0.3), loss='mse')
model.summary()
```

# 3.28 tf.keras 를 이용한 XOR 네트워크 학습

```
history = model.fit(x, y, epochs=2000, batch_size=1)
```

# 3.29 tf.keras 를 이용한 XOR 네트워크 평가

```
print(model.predict(x))
```

# 3.30 XOR 네트워크의 가중치와 편향 확인

```
for weight in model.weights:
    print(weight)
```

```
Epoch 1999/2000
4/4 [=====] - 0s 2ms/step - loss: 0.0017
Epoch 2000/2000
4/4 [=====] - 0s 1ms/step - loss: 0.0017
[[0.04060324]
 [0.9609237 ]
 [0.96031225]
 [0.04571233]]
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

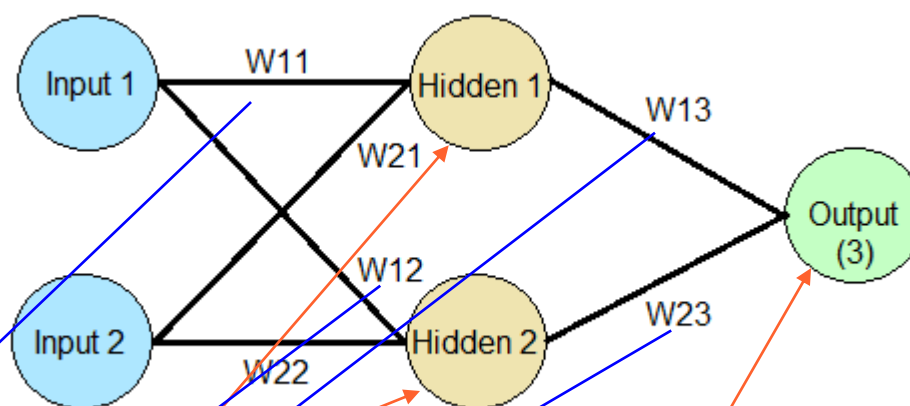
# 가중치와 model.weights

## 가중치 결과

- 층/kernel

## 편향 결과

- 층/bias



```
<tf.Variable 'dense_2/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[ 5.378626 , -5.299518 ],
       [-5.518024 ,  5.0764313]], dtype=float32)>
```

```
<tf.Variable 'dense_2/bias:0' shape=(2,) dtype=float32, numpy=array([-3.0511274, -2.8576972], dtype=float32)>
```

```
<tf.Variable 'dense_3/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[7.6903753],
       [7.7518315]], dtype=float32)>
```

```
<tf.Variable 'dense_3/bias:0' shape=(1,) dtype=float32, numpy=array([-3.8067687], dtype=float32)>
```

[b0, b1]

[b]

# XOR 모델의 학습 과정 시각화

- 손실(loss) 또는 오류 값의 변화
  - 가로는 에폭의 수
    - 학습 횟수가 증가하면서 계속 손실은 작아짐

```
# 3.34 2-레이어 XOR 네트워크의 loss 변화를 선 그래프로 표시  
import matplotlib.pyplot as plt
```

```
plt.plot(history.history['loss'])
```

