

딤러닝을 함께 공부해보세~  
영차영차 2일차

### 3. 신경망

### 3-1. 퍼셉트론에서 신경망으로..

### 3-1. 퍼셉트론에서 신경망으로..

입력층, 은닉층, 출력층.. (그냥 용어만 추가)

그림 2-13 XOR의 퍼셉트론

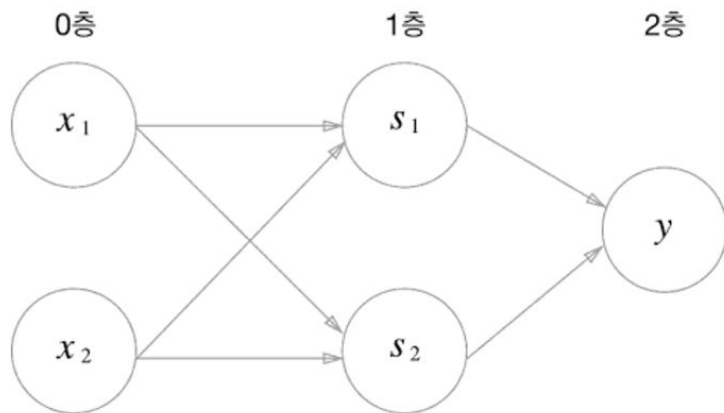
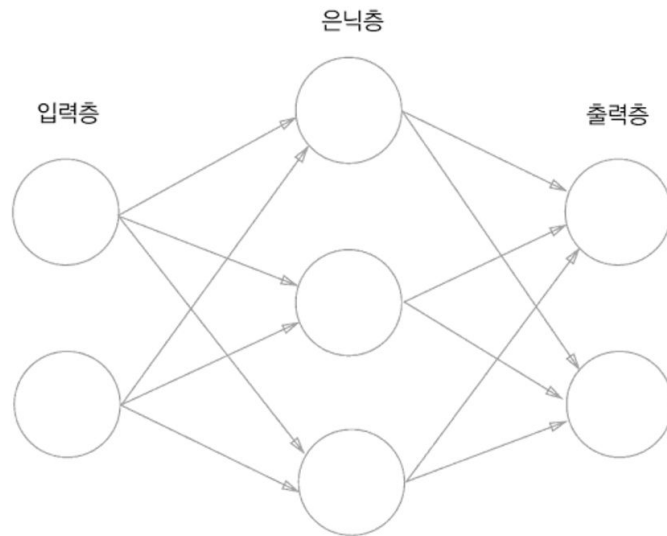
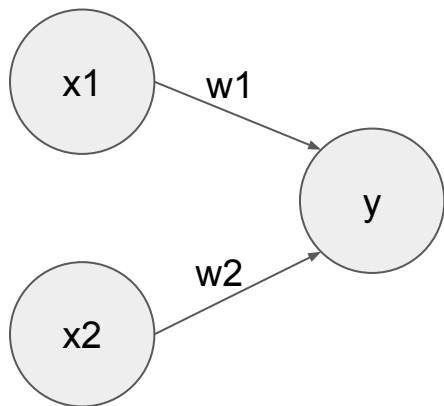


그림 3-1 신경망의 예

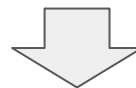


### 3-1. 퍼셉트론에서 신경망으로..

퍼셉트론 복습



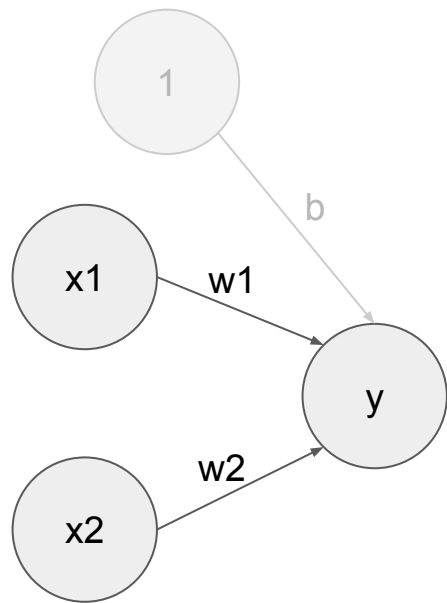
$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$



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

### 3-1. 퍼셉트론에서 신경망으로..

퍼셉트론 복습



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

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

### 3-1. 퍼셉트론에서 신경망으로..

자세하게 들여다 보면 입력과 가중치 신호를 조합한 결과 그리고.. 그 값을 가지고 출력값으로 변환하는 계산

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

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

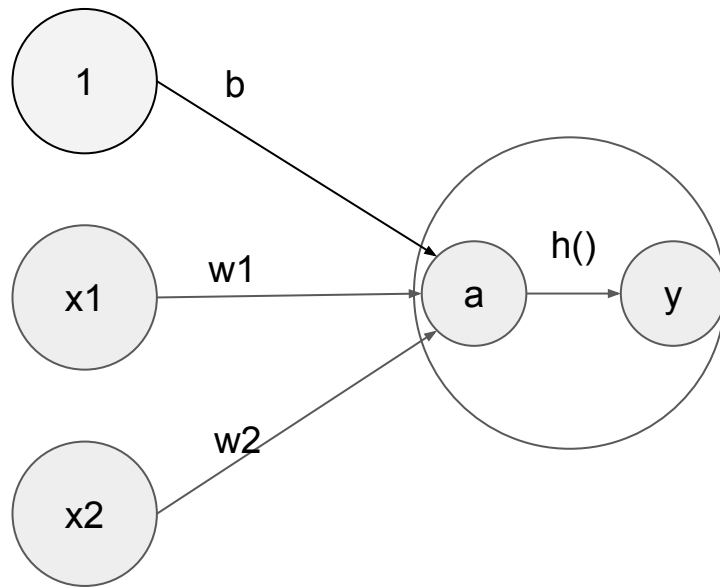
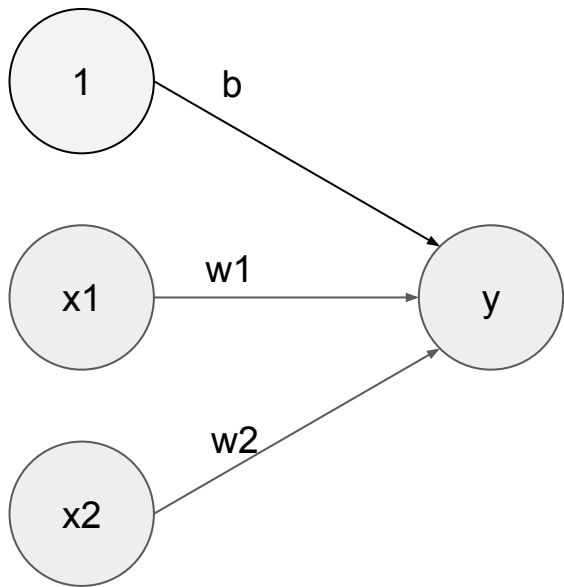
## 3-2. 활성화 함수(Activation Function)

출력 값으로 변환하는 계산! 사실 이미 사용하던 것....!



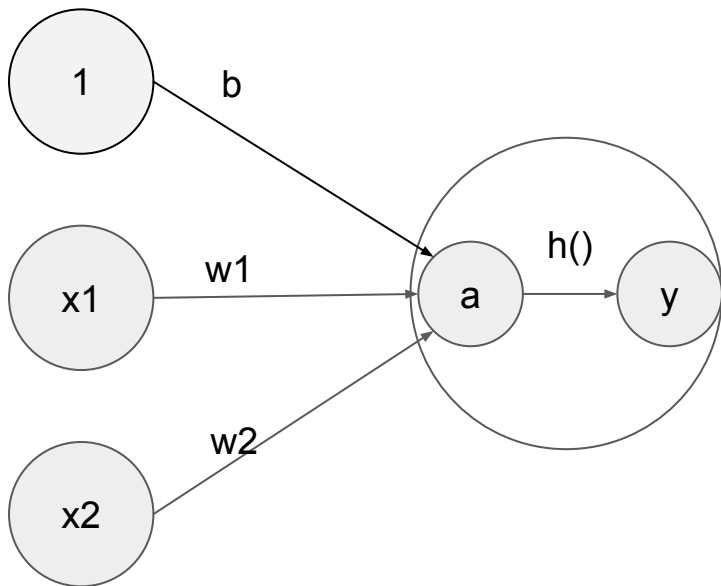
## 3-2. 활성화 함수(Activation Function)

활성화 함수(Activation Function)의 등장!



## 3-2. 활성화 함수(Activation Function)

활성화 함수(Activation Function)의 등장! (사실 이미 알던 것)

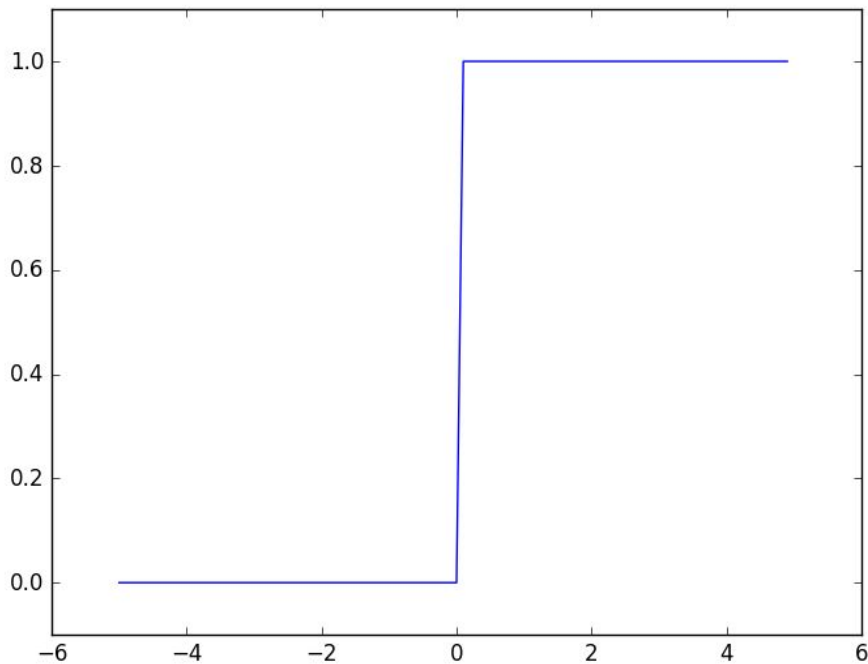


a: 입력과 가중치 신호를 조합한 결과(입력신호의 총합) **노드 == 뉴런**

y: 활성화 함수 **h()**를 통과하여 변환된 결과 **노드 == 뉴런**

## 3-2. 활성화 함수(Activation Function)

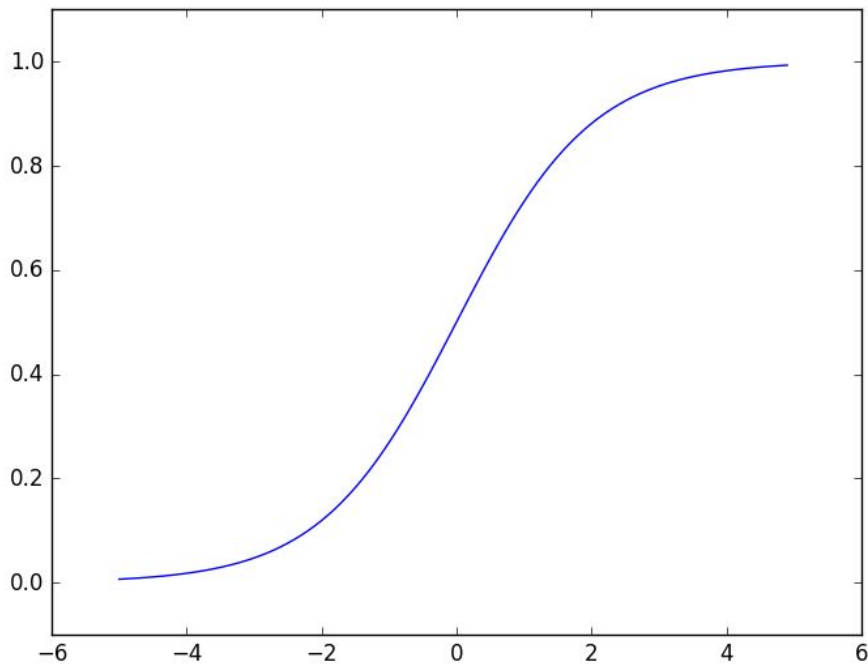
계단 함수



```
def step_function(x):  
    if x > 0:  
        return 1  
    else:  
        return 0
```

## 3-2. 활성화 함수(Activation Function)

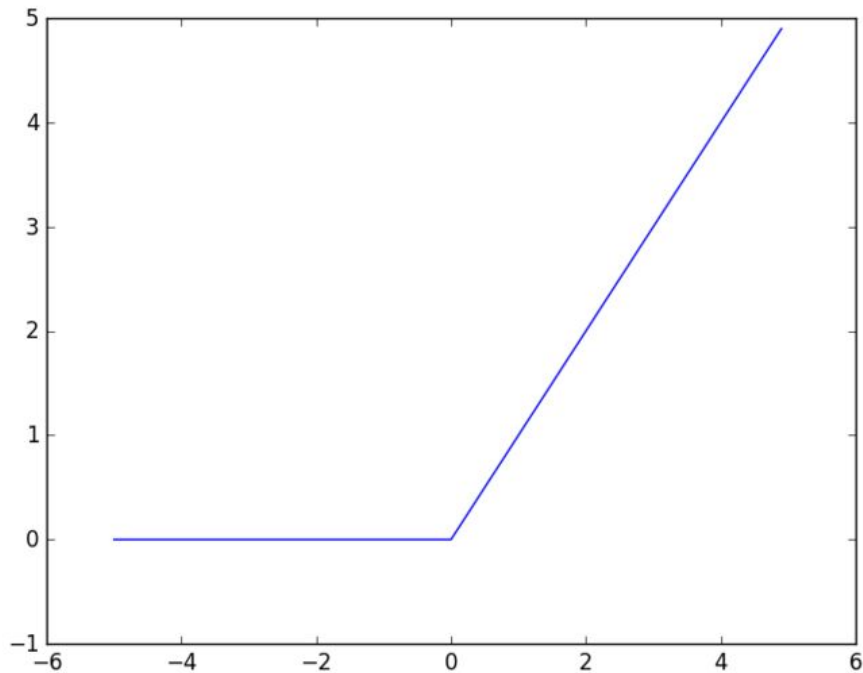
시그모이드(sigmoid) 함수



```
def sigmoid(x):  
    return 1 / (1 + np.exp(-x))
```

## 3-2. 활성화 함수(Activation Function)

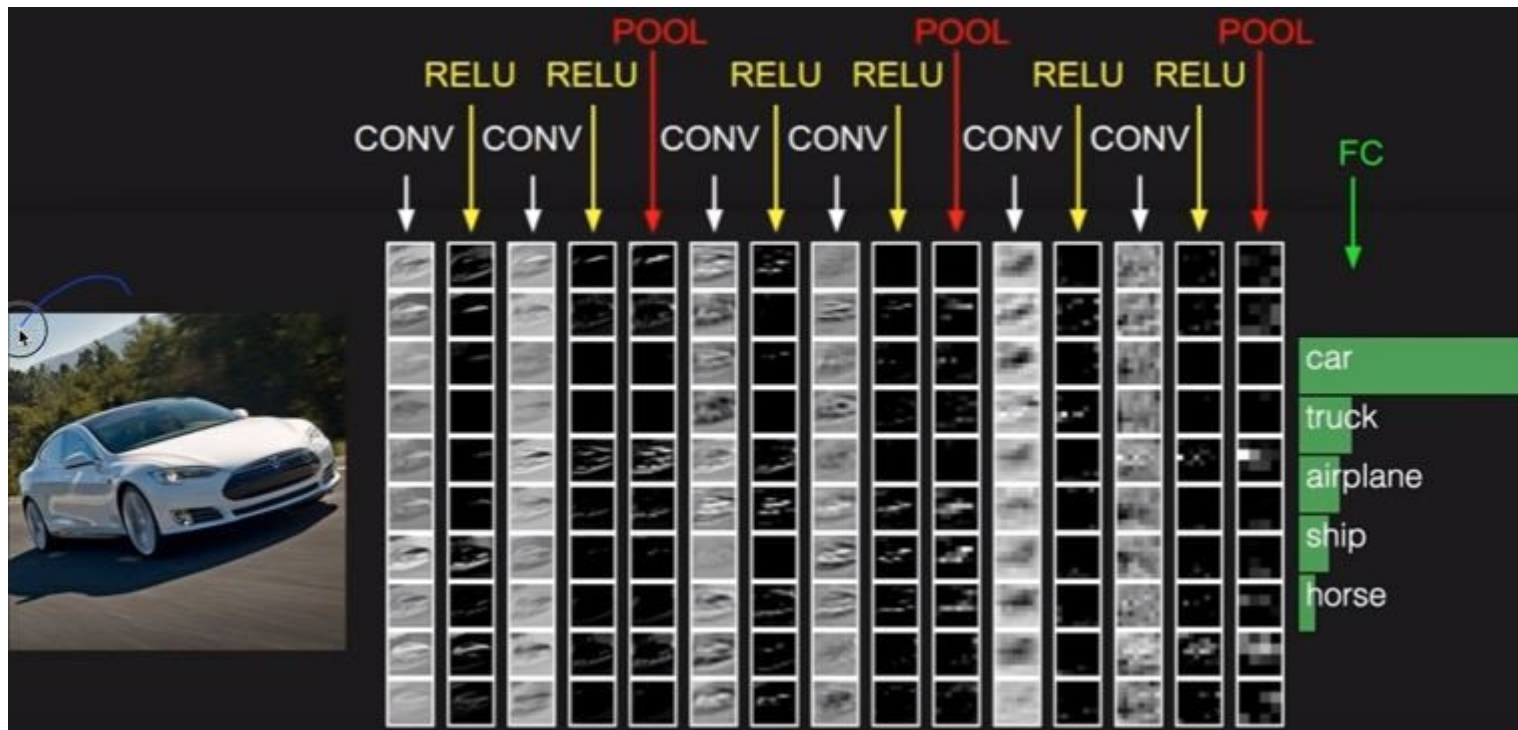
### ReLU 함수



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

## 3-2. 활성화 함수(Activation Function)

각자 다른 쓰임새가 있으나 요즘에는 ReLU를 많이 사용한다.

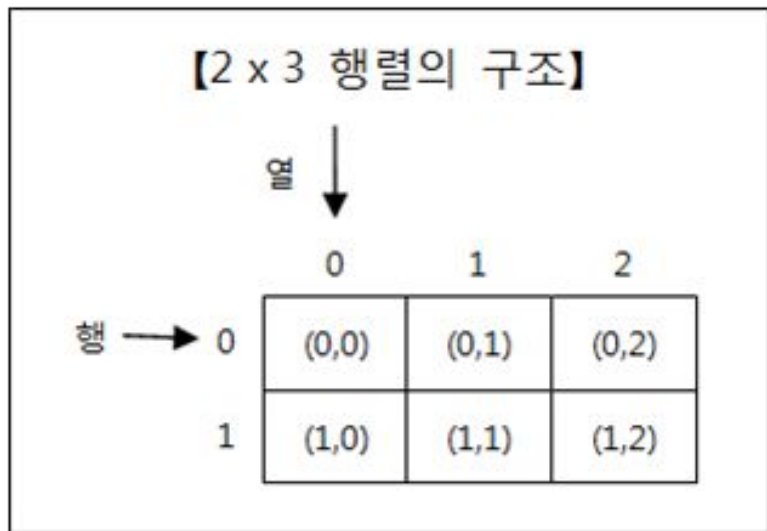


### 3-3. 다차원 배열의 계산

신경망을 효율적으로 구현하기 위하여.. 잠시 다차원 배열 설명을 하겠습니다.

### 3-3. 다차원 배열의 계산

다차원 배열

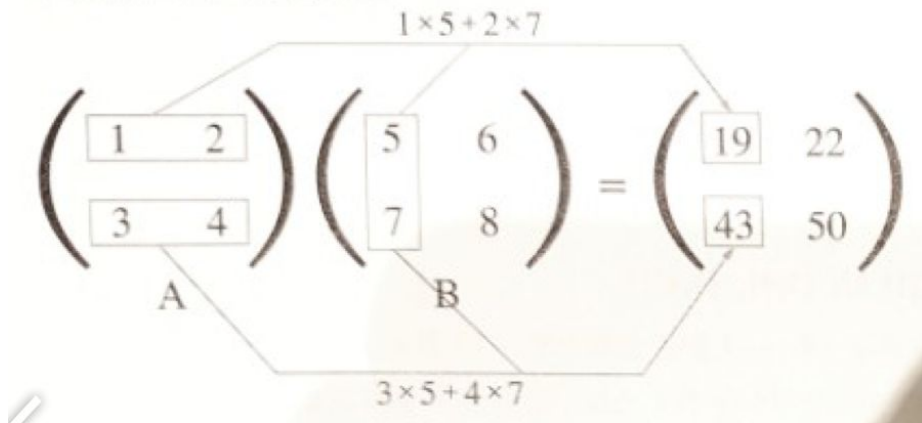




### 3-3. 다차원 배열의 계산

행렬의 내적 (행렬 곱) - 연산 방법

그림 3-11 행렬의 내적 계산 방법



```
import numpy as np

A = np.array([[1, 2],
              [3, 4]])

B = np.array([[5, 6],
              [7, 8]])

r = np.dot(A, B)

print(r)
```

```
[[19 22]
 [43 50]]
```

### 3-3. 다차원 배열의 계산

행렬의 내적 (행렬 곱) - 대응하는 차원의 원소 수가 같아야 곱이 가능하다

그림 3-12 행렬의 곱에서는 대응하는 차원의 원소 수를 일치시켜라.

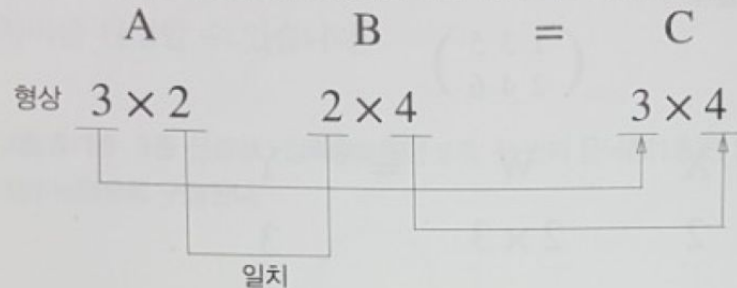
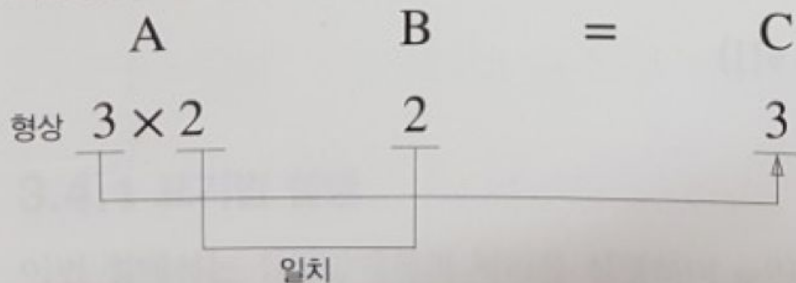
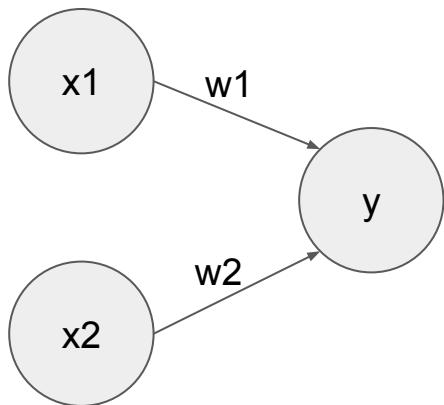


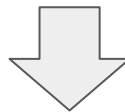
그림 3-13 A가 2차원 행렬, B가 1차원 배열일 때도 대응하는 차원의 원소 수를 일치시켜라.



### 3-3. 다차원 배열의 계산

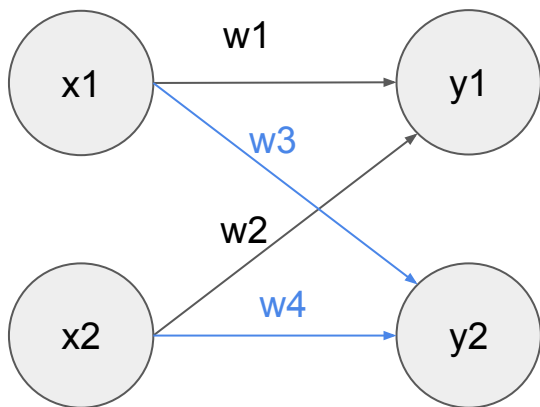


$$x_1 w_1 + x_2 w_2 = y$$

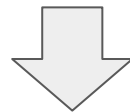


$$(x_1 \ x_2) \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = (y)$$

### 3-3. 다차원 배열의 계산

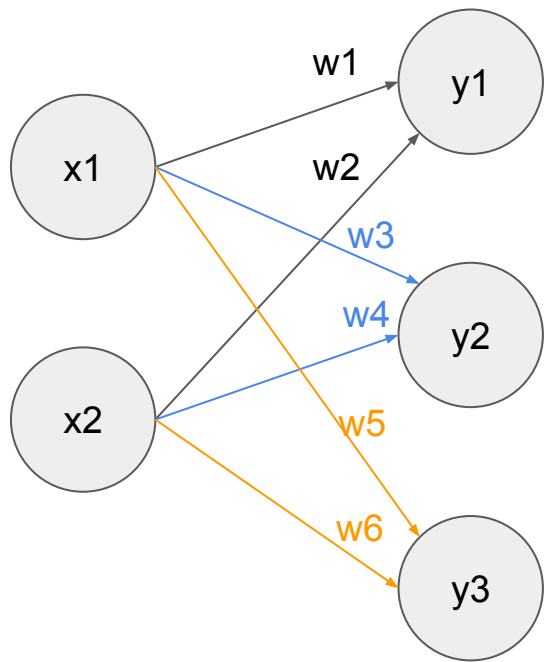


$$\begin{aligned}x_1 w_1 + x_2 w_2 &= y_1 \\x_1 w_3 + x_2 w_4 &= y_2\end{aligned}$$

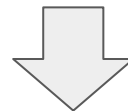


$$(x_1 \ x_2) \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = (y_1, y_2)$$

### 3-3. 다차원 배열의 계산

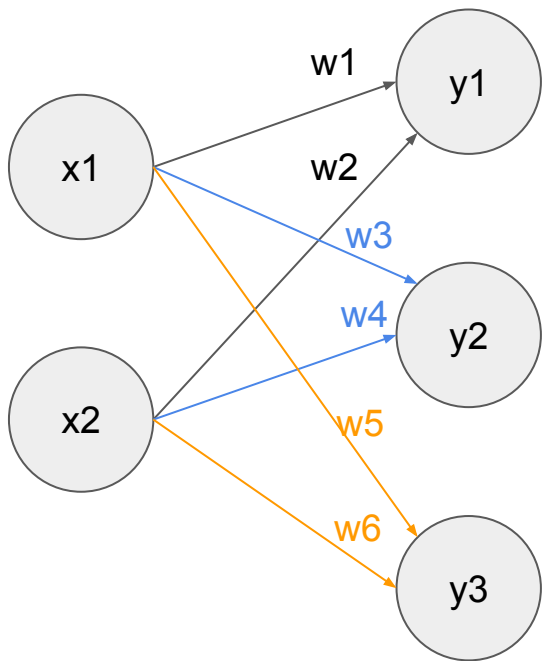


$$\begin{aligned}x_1 w_1 + x_2 w_2 &= y_1 \\x_1 w_3 + x_2 w_4 &= y_2 \\x_1 w_5 + x_2 w_5 &= y_2\end{aligned}$$



$$(x_1 \ x_2) \begin{bmatrix} w_1 & w_3 & w_5 \\ w_2 & w_4 & w_6 \end{bmatrix} = (y_1, y_2, y_3)$$

### 3-3. 다차원 배열의 계산



$$(x_1 \ x_2) \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix} = (y_1, y_2, y_3)$$

```
X = np.array([1, 2])  
W = np.array([[1, 3, 5],  
              [2, 4, 6]])  
Y = np.dot(X, W)  
print(Y)  
[ 5 11 17]
```

### 3-3. 다차원 배열의 계산

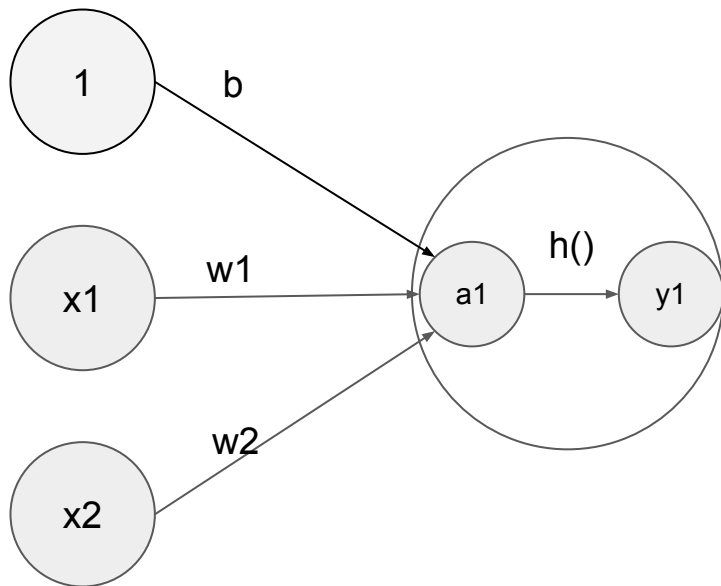
신경망을 효율적으로 구현할 수 있고,

입력값 수 출력값 수 그리고 필요한 가중치( $W$ ) 등을 유연하게 조정할 수 있다.

-> 다중 레이어를 갖는 **Deep Learning** 에서 매우 유용!

### 3-4. 3층 신경망 구현하기

다시 돌아와서..





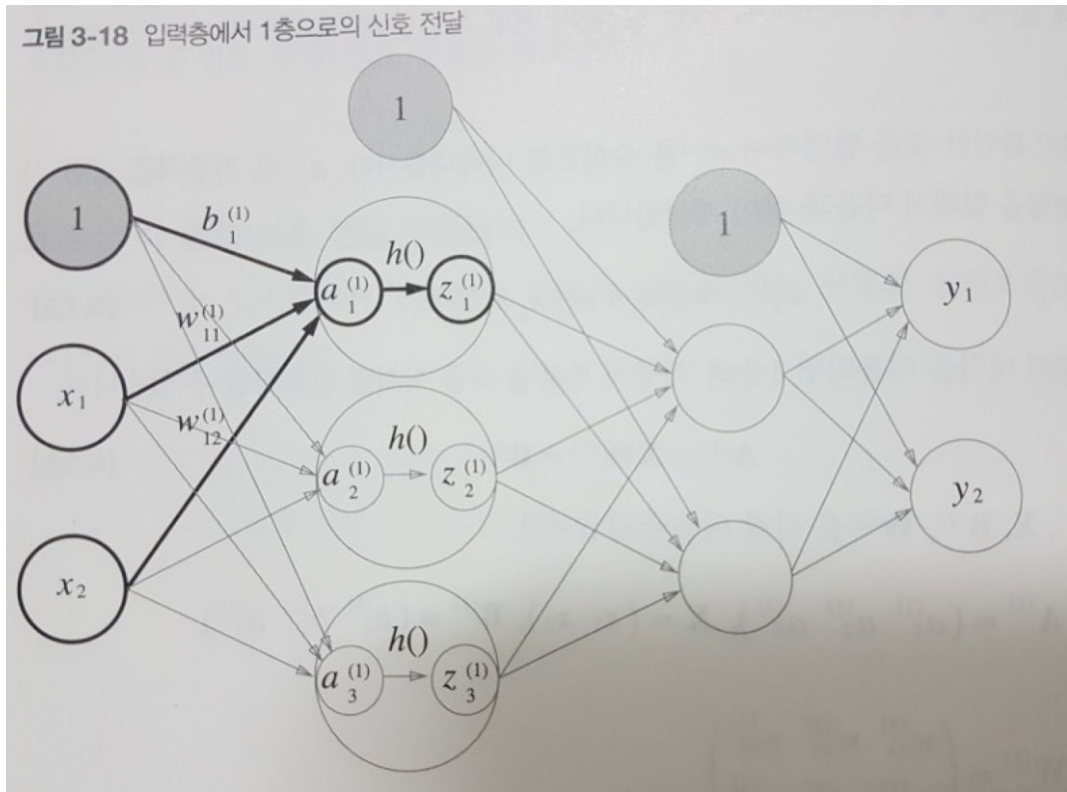
## 3-4. 3층 신경망 구현하기

헛갈릴 수도 있다! 앞으로 구현할 신경망에서..

- 입력( $X$ )은 우리가 제공할 값
- 출력( $Y$ )은 신경망에서 나올 결과 값
- 가중치( $W$ )와 편향( $B$ )는 이미 학습되어 주어진 값
  - > 자동으로 학습하는 방법은 나중에!

### 3-4. 3층 신경망 구현하기

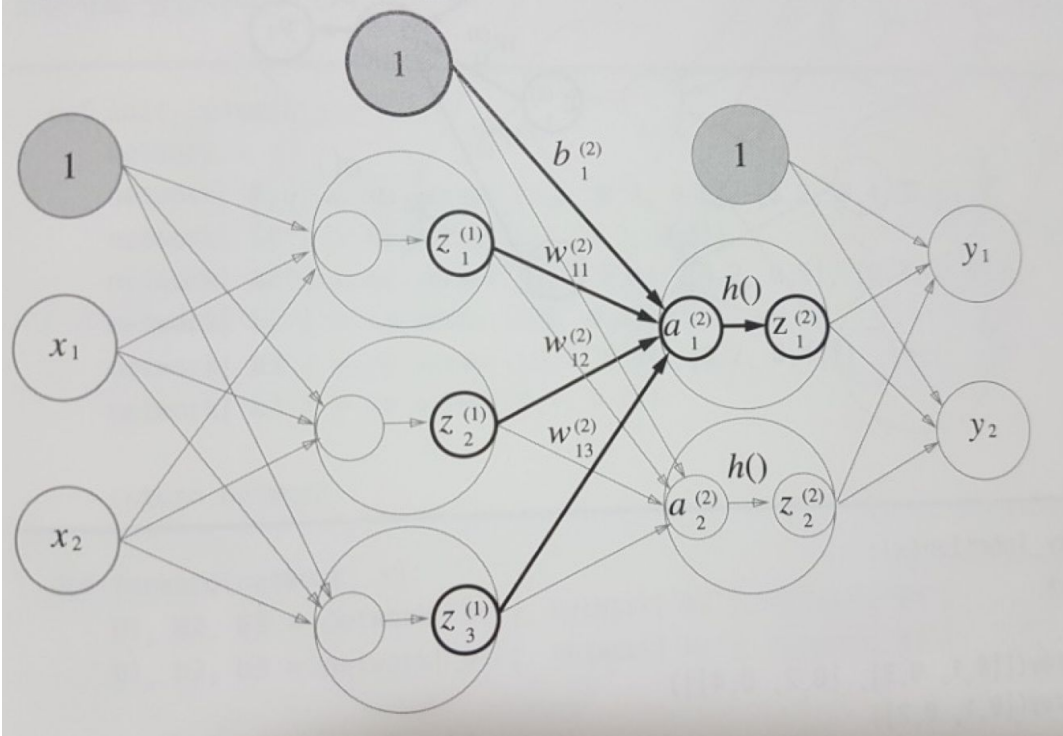
입력층에서 1층으로 신호 전달



### 3-4. 3층 신경망 구현하기

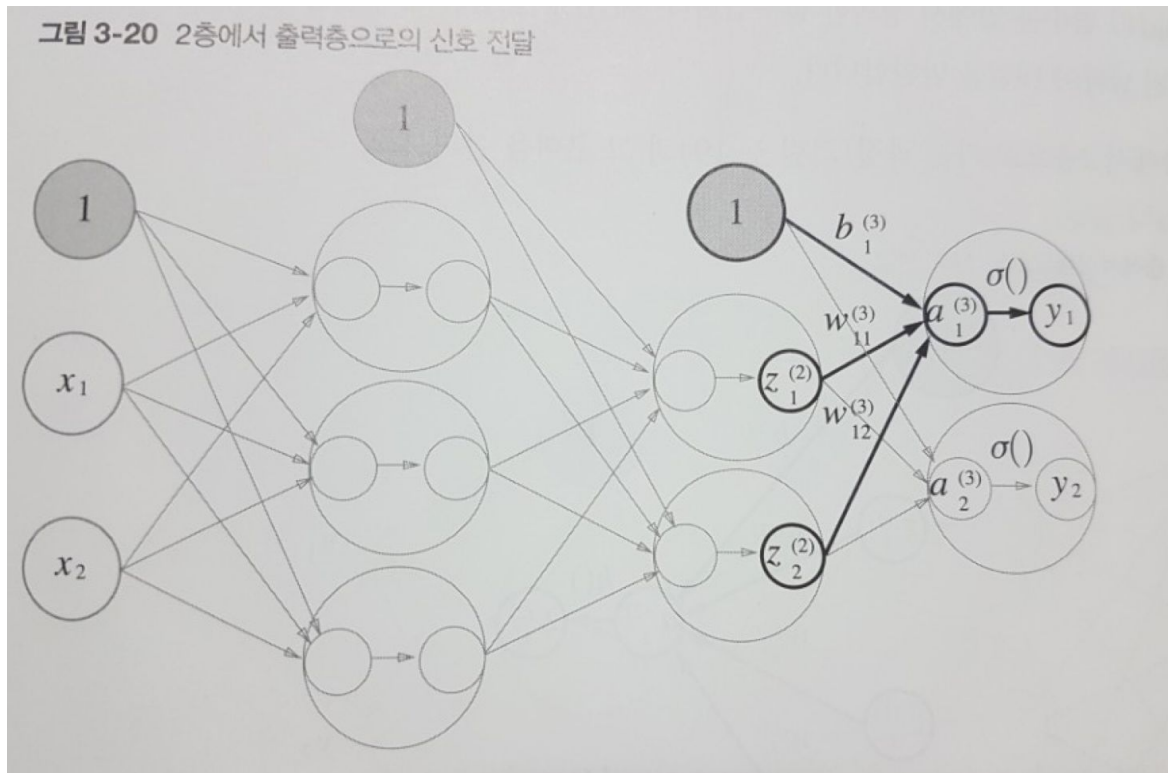
1층에서 2층으로의 신호 전달

그림 3-19 1층에서 2층으로의 신호 전달



### 3-4. 3층 신경망 구현하기

2층에서 출력층으로의 신호 전달



## 3-4. 3층 신경망 구현하기

순수 코코코딩 으로 3층짜리 신경망을 구현해보자

```
import numpy as np

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

def identity_function(x):
    return x

def init_network():
    network = {}
    network['W1'] = np.array([[0.1, 0.3, 0.5], [0.2, 0.4, 0.6]])
    network['b1'] = np.array([0.1, 0.2, 0.3])
    network['W2'] = np.array([[0.1, 0.4], [0.2, 0.5], [0.3, 0.6]])
    network['b2'] = np.array([0.1, 0.2])
    network['W3'] = np.array([[0.1, 0.3], [0.2, 0.4]])
    network['b3'] = np.array([0.1, 0.2])

    return network
```

```
def forward(network, x):
    W1, W2, W3 = network['W1'], network['W2'], network['W3']
    b1, b2, b3 = network['b1'], network['b2'], network['b3']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    z2 = sigmoid(a2)
    a3 = np.dot(z2, W3) + b3
    y = identity_function(a3)
```

```
    return y
```

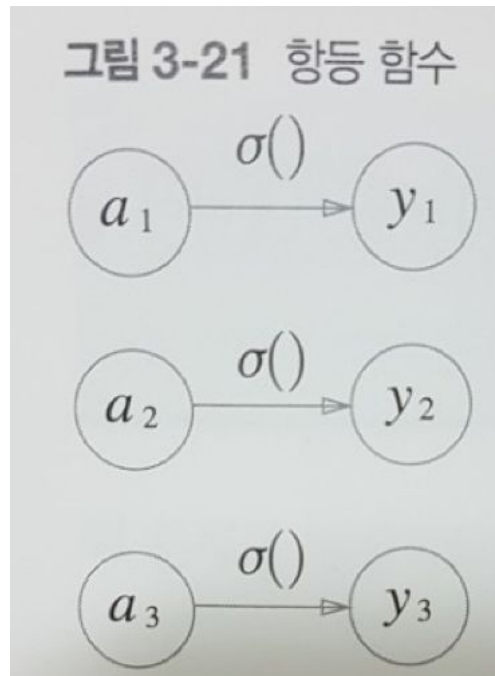
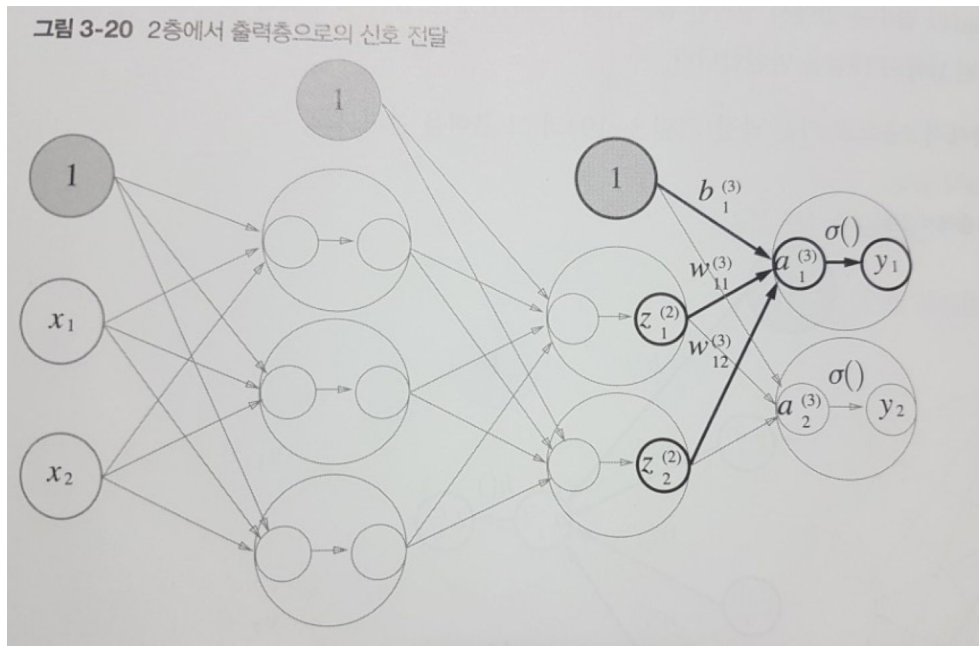
```
network = init_network()
x = np.array([1.0, 0.5])
y = forward(network, x)
```

```
print(y)
```

[ 0.31682708 0.69627909]

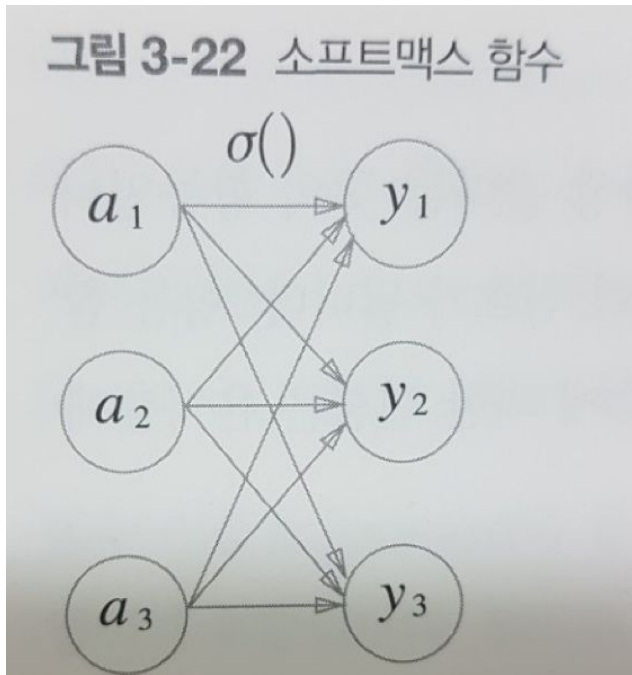
### 3-5. 그렇다면 마지막 출력층은?

값을 그대로 내보내는 항등함수!



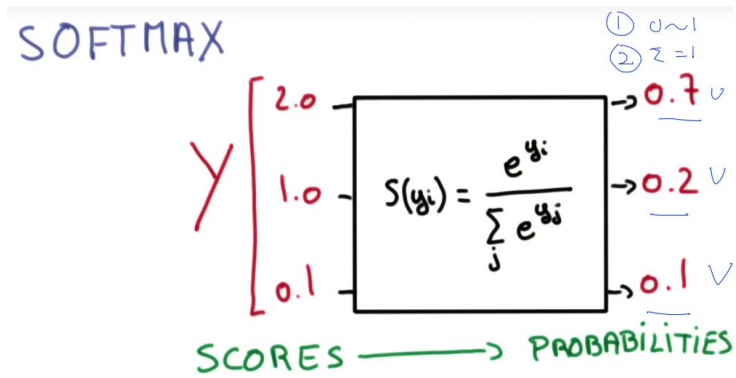
### 3-5. 그렇다면 마지막 출력층은?

모든 출력의 합을 1로 만드는!! 소프트맥스(softmax)!



### 3-5. 그렇다면 마지막 출력층은?

모든 출력의 합을 1로 만드는!! 소프트맥스(softmax)!



```
import numpy as np

def softmax(a):
    c = np.max(a)
    exp_a = np.exp(a - c)
    sum_exp_a = np.sum(exp_a)
    y = exp_a / sum_exp_a

    return y

a = np.array([0.3, 2.9, 4.0])
y = softmax(a)

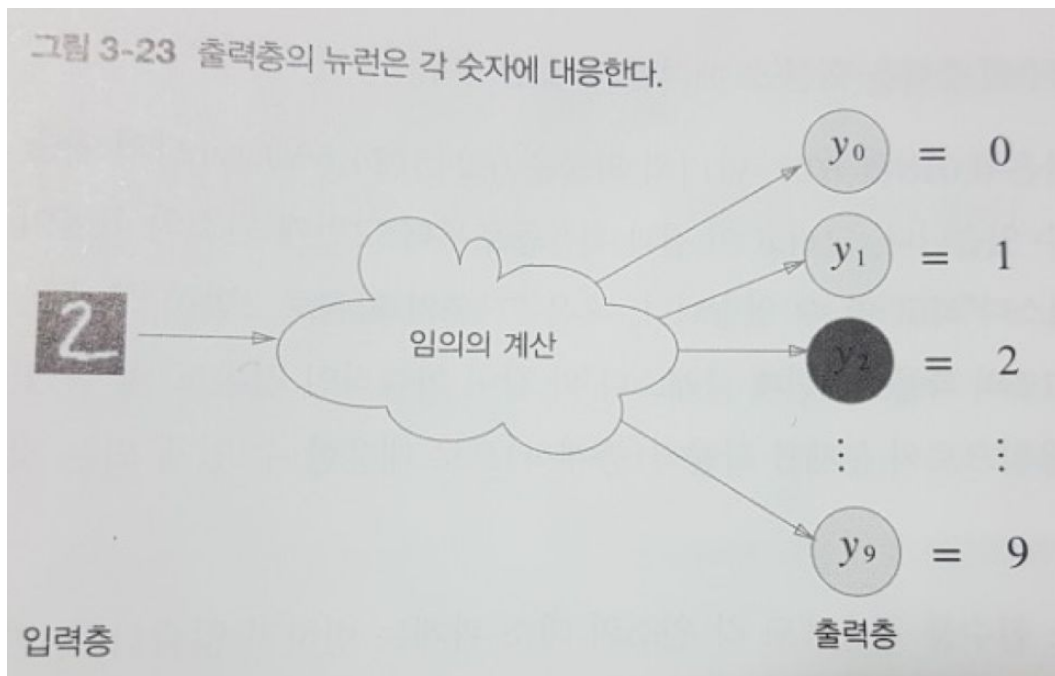
print(y)
print(np.sum(y))
```

```
[ 0.01821127  0.24519181  0.73659691]
1.0
```



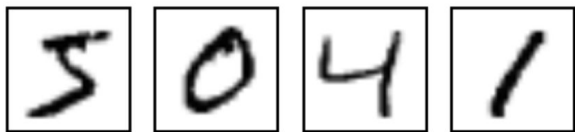
### 3-5. 그렇다면 마지막 출력층은?

- + 출력층의 뉴런 수는 어떻게 정해지는가? **classification(분류)** 과 관련

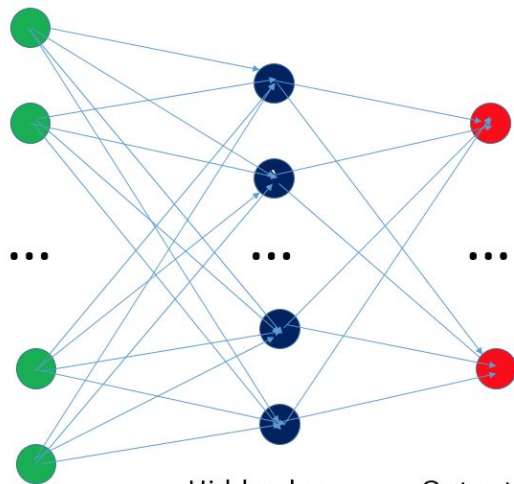


### 3-5. 그렇다면 마지막 출력층은?

손글씨를 인식해보자 : MNIST



Input image:  
28x28 pixels



Input layer:  
784 neurons,  
one per pixel

Hidden layer:  
100 neurons

Output layer:  
10 neurons



7

Output:  
predicted  
digit value