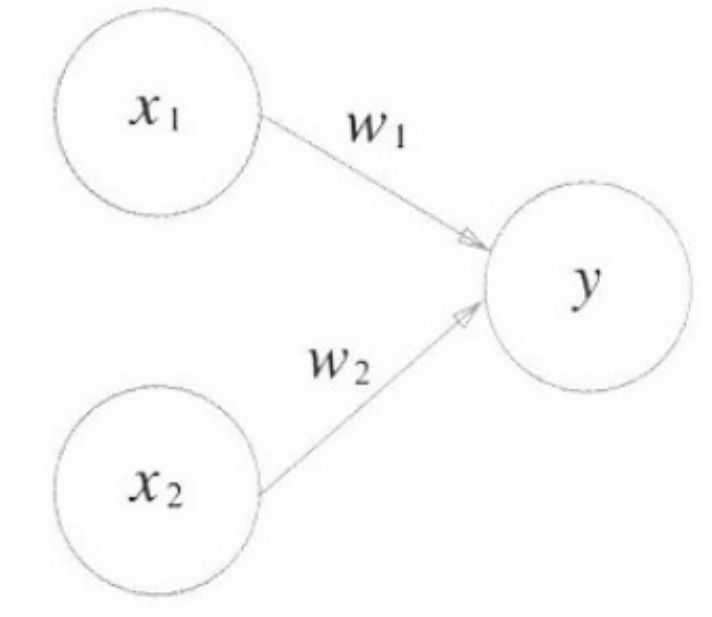
밑바닥부터 시작하는 딥러닝

퍼셉트론

퍼셉트론이란?

다수의 신호를 입력으로 받아 하나의 신호(0 또는 1)를 출력하는 알고리즘



가중치

입력신호마다 각각의 고유한 가중치가 곱해짐
가중치가 크다는 것은 해당 입력 신호가 중요함을 뜻함

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

출력

신호의 총합이 정해진 한계(θ)를 넘으면 1 출력

가중치와 편향

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

가중치
입력 신호가 결과의 주는 영향력을 조절

편향
출력 결과로 1을 얼마나 쉽게 출력할지 조정

편향이 -0.1

편향이 -20.0

논리회로 -> 퍼셉트론으로 구현

AND 게이트

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

```
def AND(x1, x2):
x = np.array([x1, x2])
w = np.array([0.5, 0.5])
b = -0.7

tmp = np.sum(x*w) + b
if tmp <= 0:
return 0
else:
return 1
```

OR 게이트

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

```
def OR(x1, x2):
x = np.array([x1, x2])
w = np.array([0.5, 0.5])
b = -0.2

tmp = np.sum(x*w) + b
if tmp <= 0:
return 0
else:
return 1
```

NAND 게이트

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

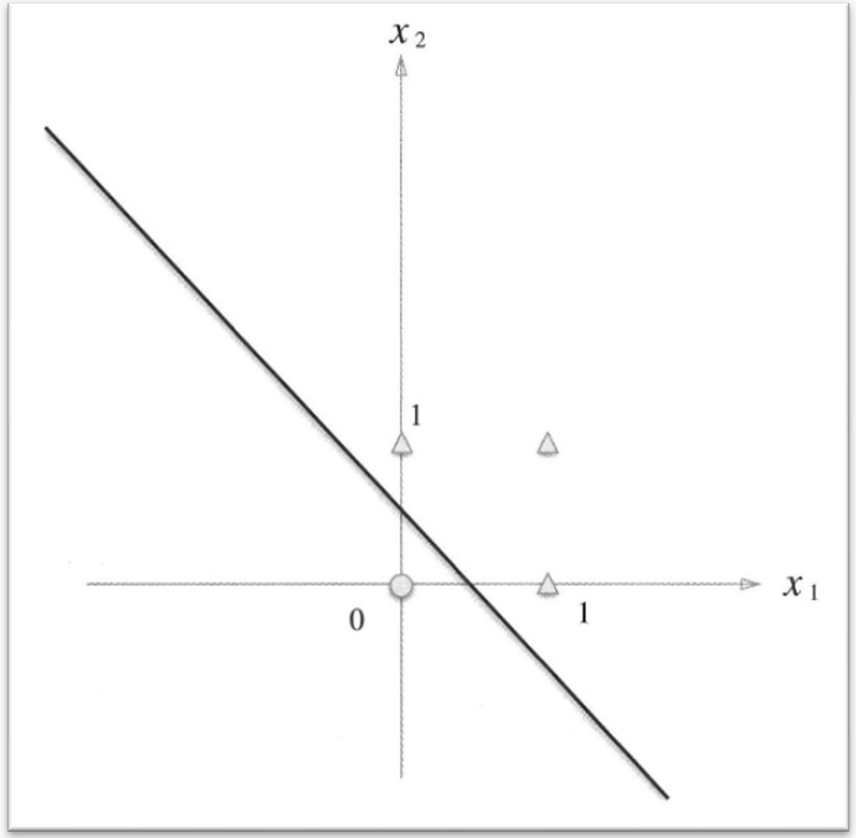
```
def NAND(x1, x2):
x = np.array([x1, x2])
w = np.array([-0.5, -0.5])
b = 0.7

tmp = np.sum(x*w) + b
if tmp <= 0:
return 0
else:
return 1
```

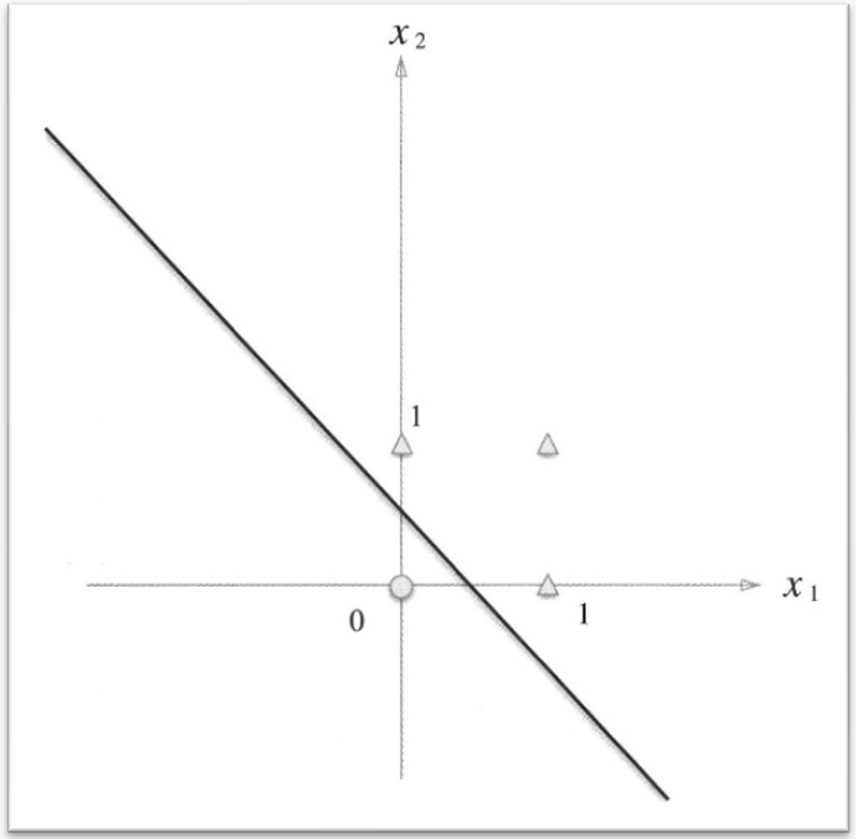
입력 2개와 출력 1개의 퍼셉트론

OR게이트의 경우

OR 게이트

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

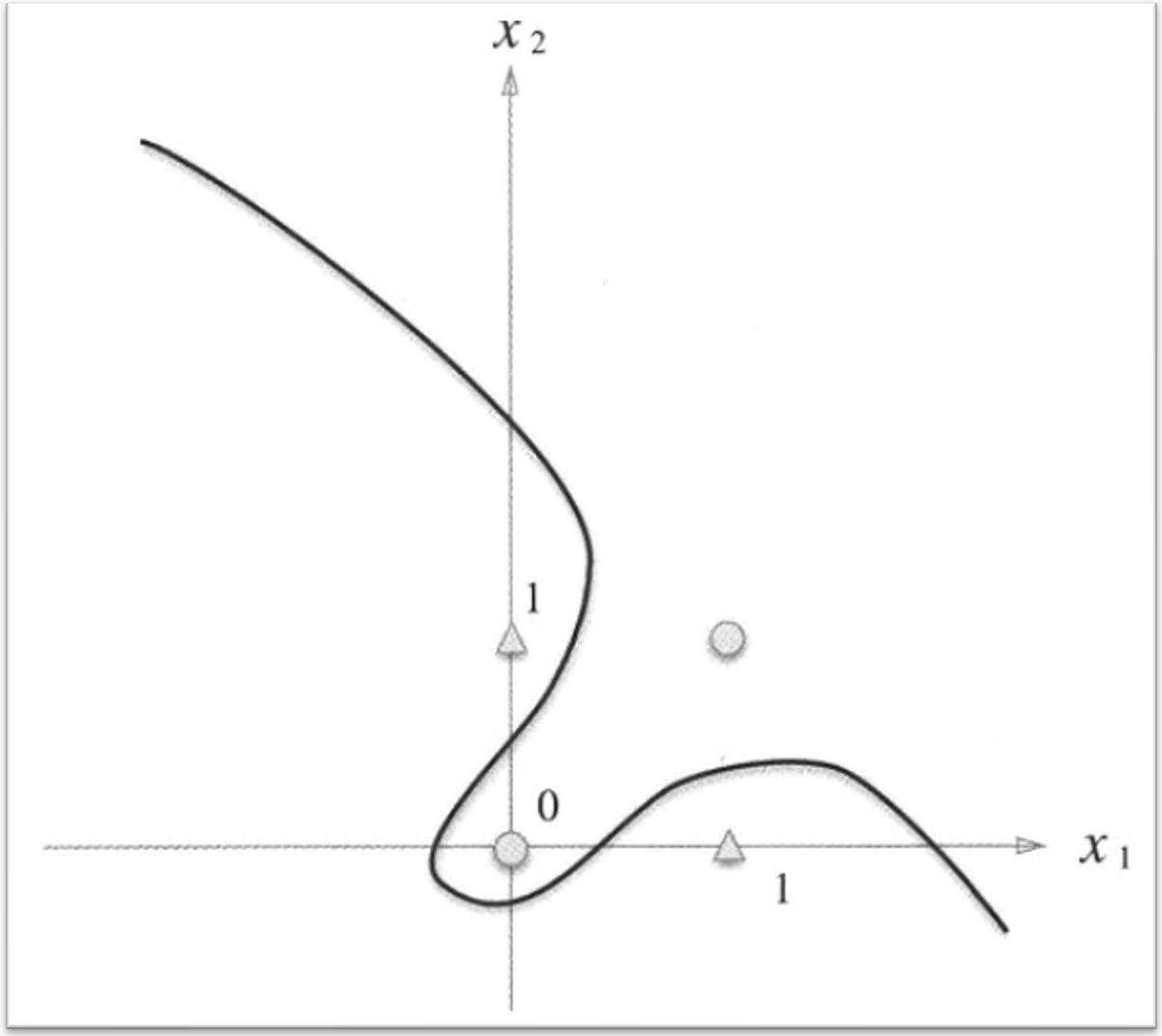




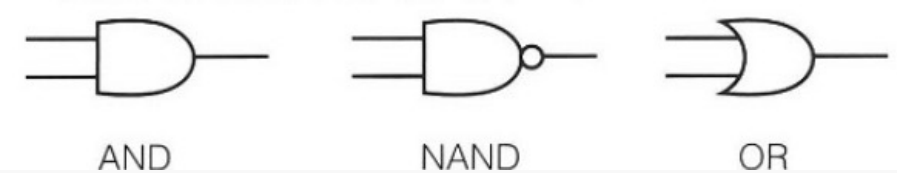
$$y = \begin{cases} 0\ (-0.5 + x_1 + x_2 \le 0) \\ 1\ (-0.5 + x_1 + x_2 > 0) \end{cases}$$

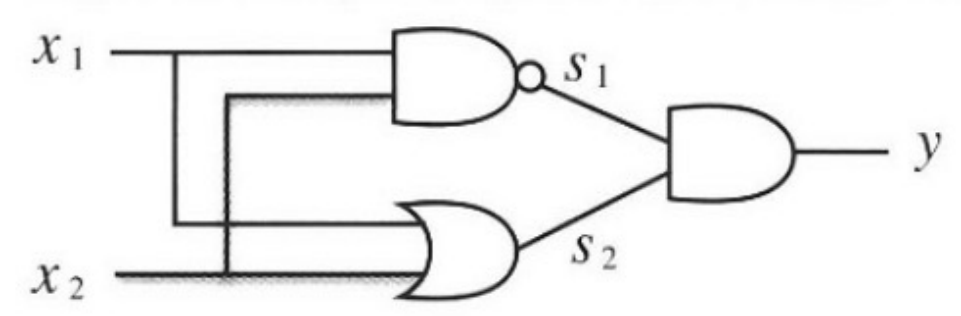
퍼셉트론의 한계
XOR게이트

XOR 게이트		
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



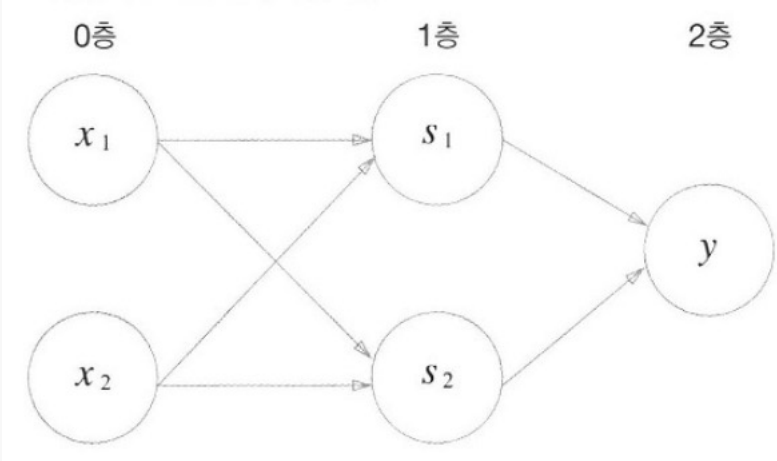
다층 퍼셉트론





XOR 게이트

x_1	x_2	s_1	s_2	y
0	0	1	0	0
0	1	1	1	1
1	0	1	1	1
1	1	0	1	0
		NAND	OR	AND




```
def XOR(x1, x2):
    s1 = NAND(x1, x2)
    s2 = OR(x1, x2)
    y = AND(s1, s2)
    return y
```

0층: 입력 신호를 받아
신호의 총합을 계산하여 1층 뉴런으로 신호 보냄

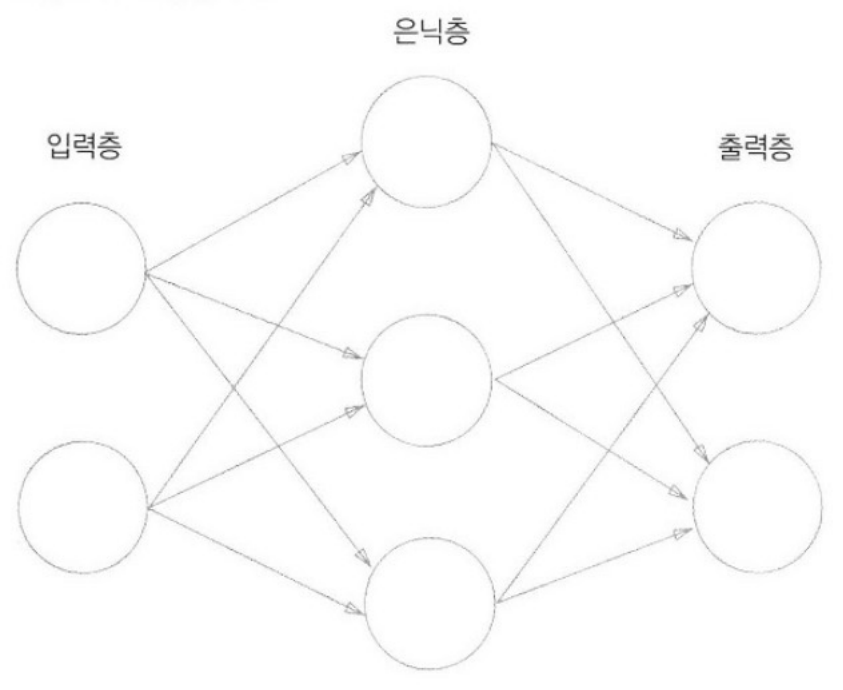
1층: 두 뉴런이 신호의 총합을 계산

2층: 뉴런은 y를 출력

신경망

신경망

데이터로부터 자동으로 학습



퍼셉트론과 활성화 함수를 통해
가중치 매개변수의 적절한 값을
데이터로 부터 자동으로 학습하는 구조

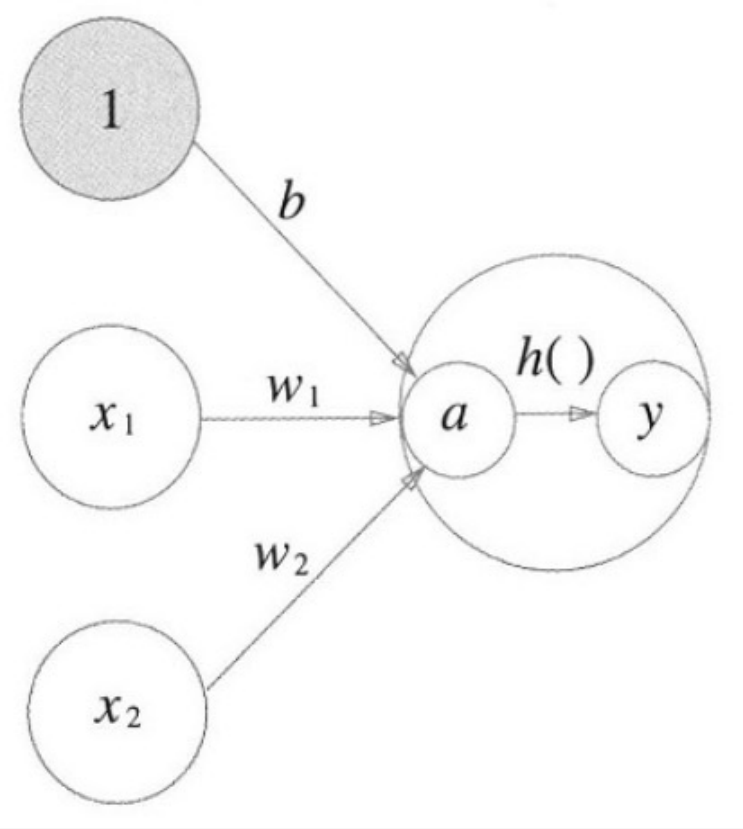
입력층 은닉층 출력층으로 나뉨

입력층은 데이터 입력
은닉층은 적절한 매개변수
출력층은 결과를 분류 또는 회귀에 활용

단층으로 해결할 수 없는 문제를
여러 층을 이용해 해결

활성화 함수

입력 신호의 총합을 출력 신호로 변환하는 함수

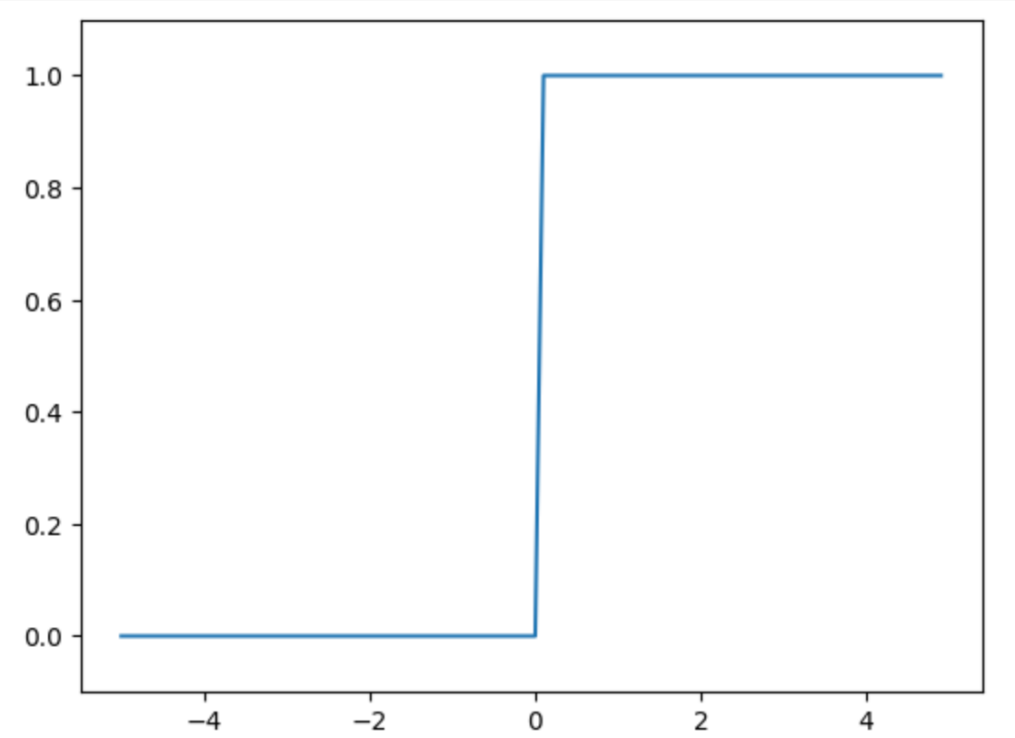


$$y = h(b + w_1x_1 + w_2x_2)$$

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

활성화 함수

계단 함수(step function)



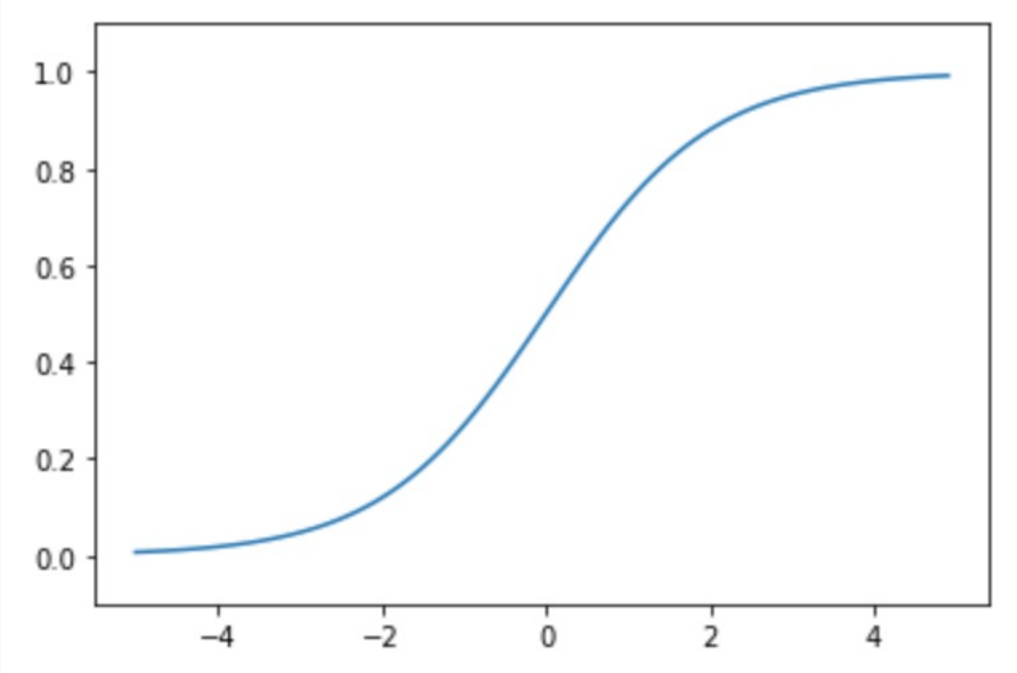
$$y = h(b + w_1 x_1 + w_2 x_2)$$

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

```
def step_function(x):
    return np.array(x > 0, dtype=np.int)

X = np.arange(-5.0, 5.0, 0.1)
Y = step_function(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```

활성화 함수

시그모이드 함수(sigmoid function)



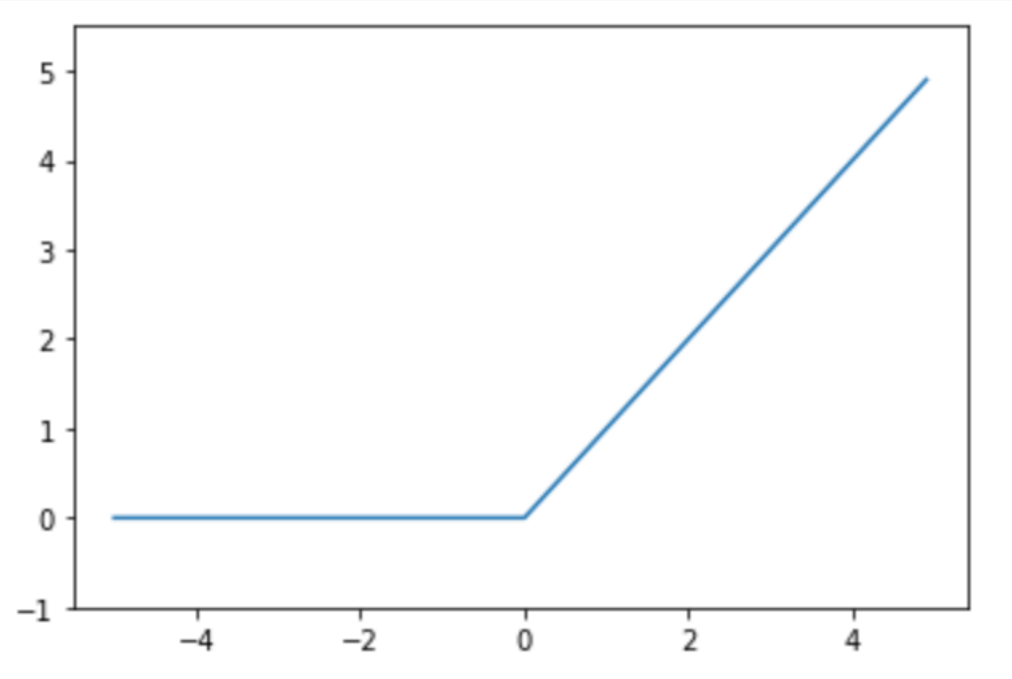

```
X = np.arange(-5.0, 5.0, 0.1)
Y = sigmoid(X)
plt.plot(X, Y)
plt.ylim(-0.1, 1.1)
plt.show()
```

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

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

활성화 함수

ReLU 함수(relu function)



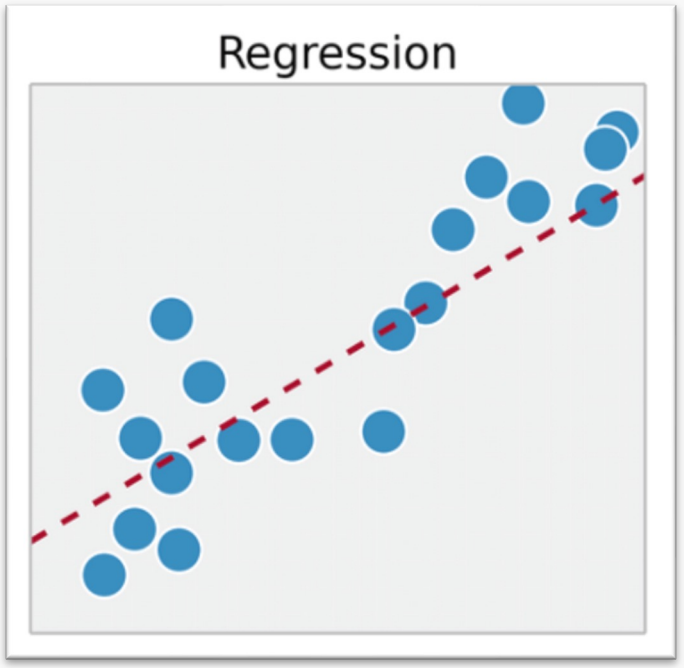

```
x = np.arange(-5.0, 5.0, 0.1)
y = relu(x)
plt.plot(x, y)
plt.ylim(-1.0, 5.5)
plt.show()
```

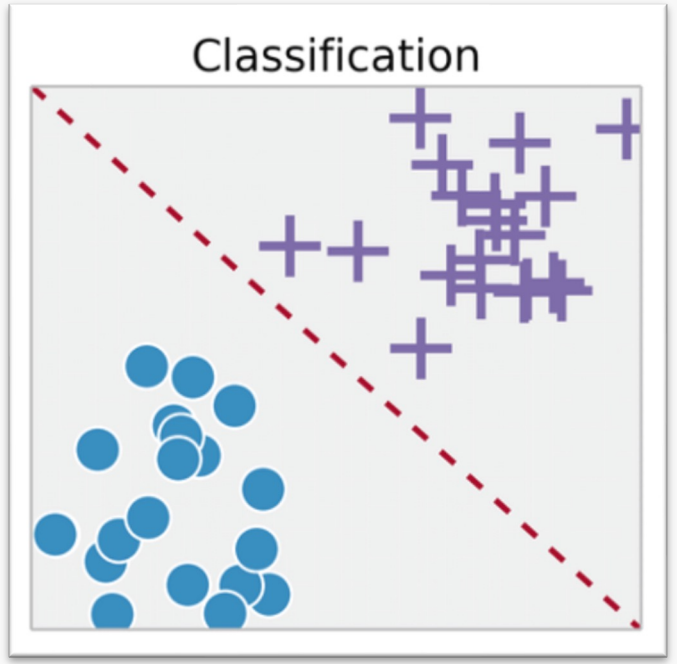
```
def relu(x):
    return np.maximum(0, x)
```

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

출력층 설계

회귀와 분류





입력 데이터에 수치를 예측하는 문제

항등 함수 사용

데이터가 어느 클래스에 속하느냐 문제

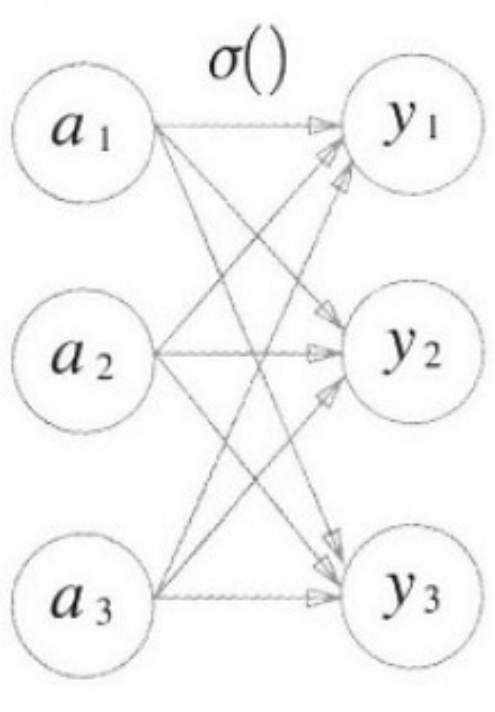
소프트맥스 함수 사용

출력층

소프트맥스 함수(softmax function)

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

$$= \frac{\text{특정 입력 신호}(a)\text{의 지수 함수}}{\text{모든 입력 신호의 지수 함수의 합}}$$



n은 출력층의 뉴런 수

y_k는 출력

a_k는 입력

소프트맥스 함수 출력의 총합은 1

각 원소의 대소 관계는 변하지 않음.
a의 원소들의 대소 관계 = y의 원소들의 대소 관계

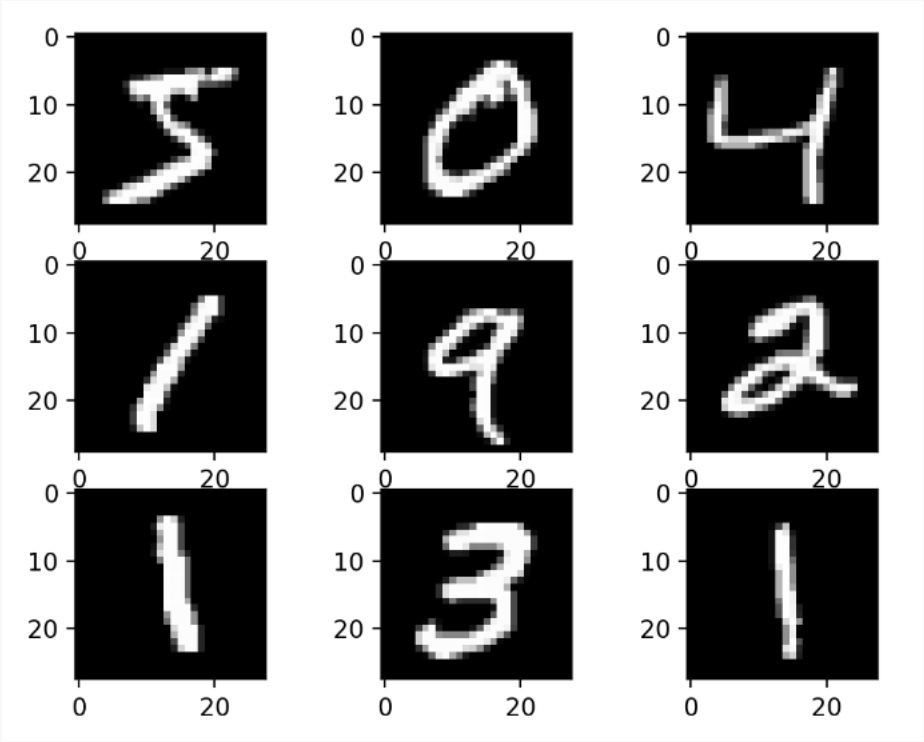
MNIST

훈련 이미지 60,000장
시험 이미지 10,000장
Grayscale 28 * 28 사이즈

```
[[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 3 18 18 18 126 136 175 26 166 255 247 127 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 30 36 94 154 170 253 253 253 253 253 225 172 253 242 195 64 0 0 0 0]
 [ 0 0 0 0 0 0 0 49 238 253 253 253 253 253 253 253 253 251 93 82 82 56 39 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 18 219 253 253 253 253 253 198 182 247 241 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 80 156 107 253 253 205 11 0 43 154 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 14 1 154 253 90 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 139 253 190 2 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 11 190 253 70 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 35 241 225 160 108 1 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 81 240 253 253 119 25 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 45 186 253 253 150 27 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 16 93 252 253 187 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 249 253 249 64 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 46 130 183 253 253 207 2 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 39 148 229 253 253 253 250 182 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 24 114 221 253 253 253 253 201 78 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 23 66 213 253 253 253 253 198 81 2 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 18 171 219 253 253 253 253 195 80 9 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 55 172 226 253 253 253 253 244 133 11 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 136 253 253 253 212 135 132 16 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
 [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

03 신경망

각 층의 뉴런들이 다음 층의 뉴런으로 신호 전달



```
(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True,
                                                  normalize=False)

print(x_train.shape)  # (60000, 784)
print(t_train.shape)  # (60000,)
print(x_test.shape)  # (10000, 784)
print(t_test.shape)  # (10000,)
```

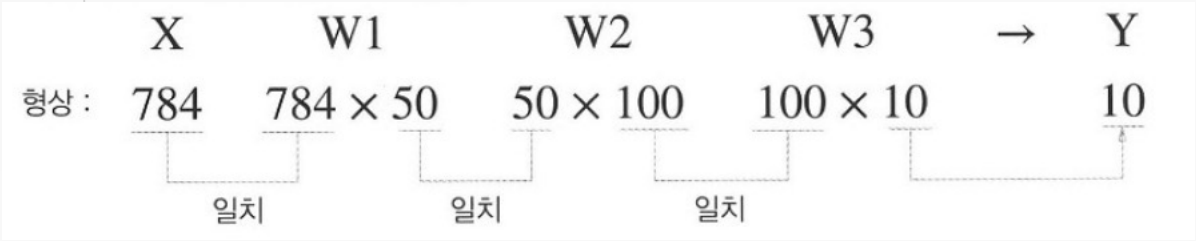
flatten : 입력 이미지를 평탄화(1차원 배열로)
normalize : 입력 이미지를 정규화(0.0~1.0 사이의 값으로)
one_hot_label : 레이블을 원-핫 인코딩 형태로 저장

```
def img_show(img):
    pil_img = Image.fromarray(np.uint8(img))
    pil_img.show()

(x_train, t_train), (x_test, t_test) = load_mnist(flatten=True, normalize=False)

img = x_train[0]
label = t_train[0]
print(label)  # 5

print(img.shape)  # (784,)
img = img.reshape(28, 28)
print(img.shape)  # (28, 28)

img_show(img)
```

03 신경망

각 층의 뉴런들이 다음 층의 뉴런으로 신호 전달

```python
def get_data():
    (x_train, t_train), (x_test, t_test)= load_mnist(normalize=True, flatten=True, one_hot_label=False)
    return x_test, t_test

def init_network():
    with open("sample_weight.pkl","rb") as f: #학습된 가중치 매개변수
        network=pickle.load(f)
    return network


def predict(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 = softmax(a3) #소프트맥스 사용

    return y
```



입력층 뉴런 : 784개
출력층 뉴런 : 10개
은닉층은 총 두개, 첫 번째 은닉층 : 50개의 뉴런, 두 번째 은닉층 : 100개의 뉴런

```
x, t=get_data()
network=init_network()

accuracy_cnt=0
for i in range(len(x)):
    y=predict(network, x[i])
    p=np.argmax(y) #확률이 가장 높은 원소의 인덱스를 반환
    if p==t[i]:
        accuracy_cnt += 1

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

```
Accuracy:0.9352
```

```
x, t=get_data()
network=init_network()

batch_size=100 #배치크기
accuracy_cnt=0

for i in range(0, len(x), batch_size):
    x_batch=x[i:i+batch_size]
    y_batch=predict(network, x_batch)
    p=np.argmax(y_batch, axis=1)
    accuracy_cnt += np.sum(p==t[i:i+batch_size])

print("Accuracy:" + str(float(accuracy_cnt) / len(x)))
```

```
Accuracy:0.9352
```

Q & A