07.15

# 인공신경망

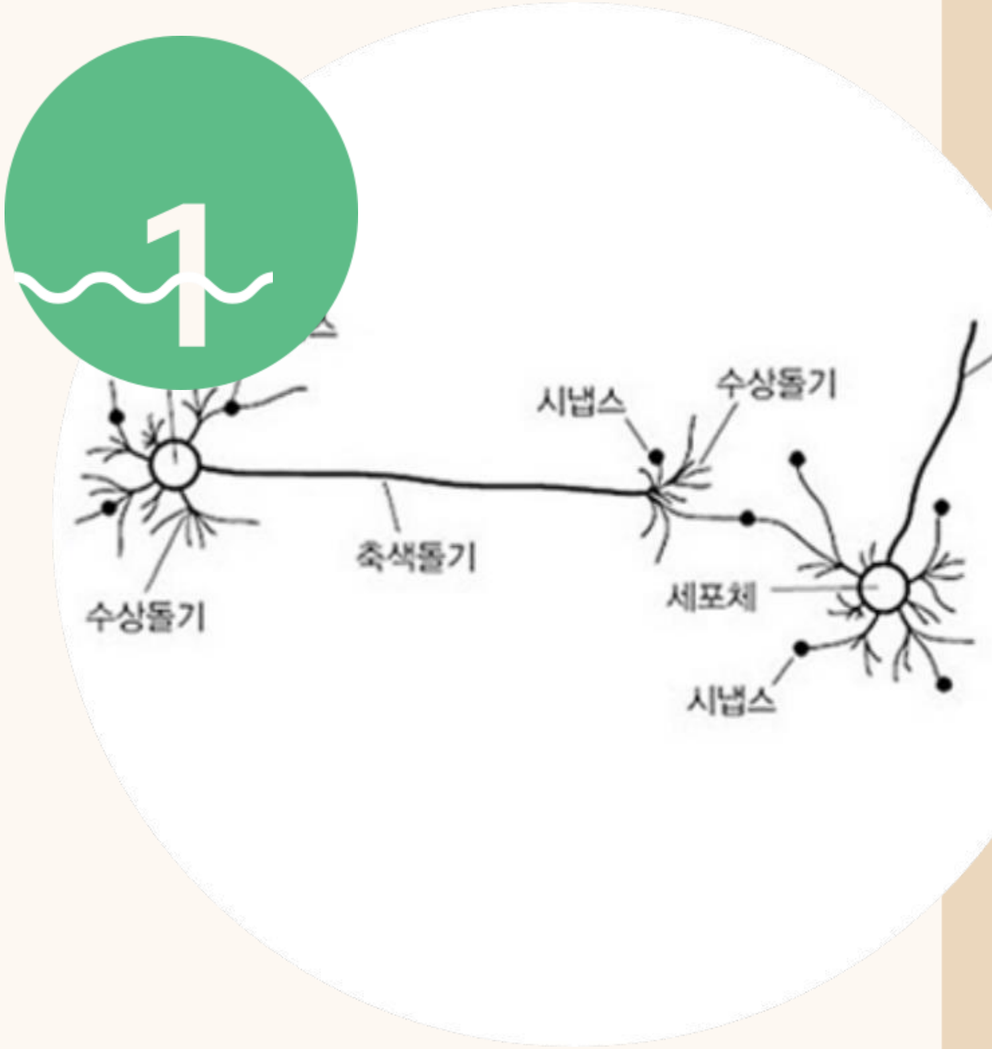
장혜선

# 목차

CHAPTER. 1

# 개념 설명

- 인공뉴런
- 인공신경망



# 인공뉴런

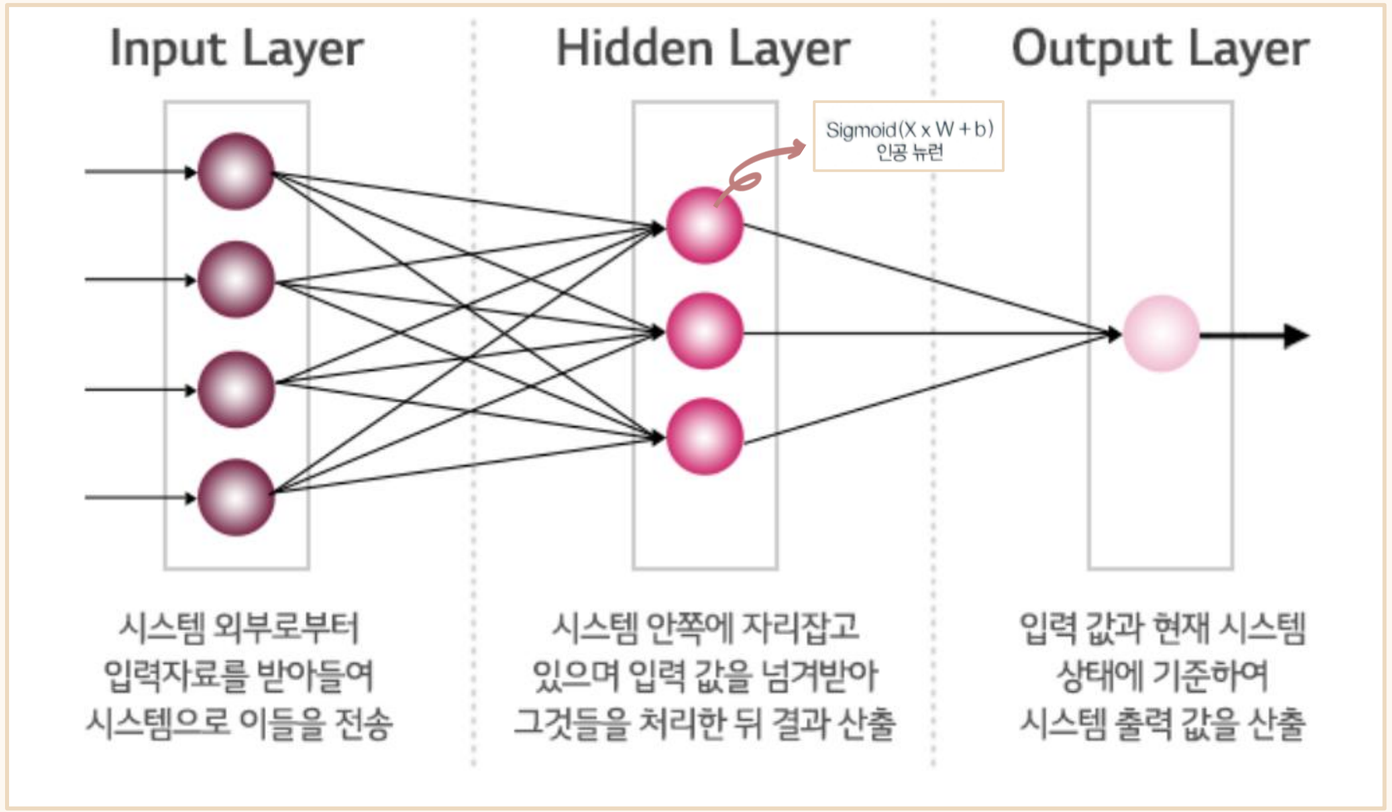
$$y = 활성화함수(x \times W + b)$$

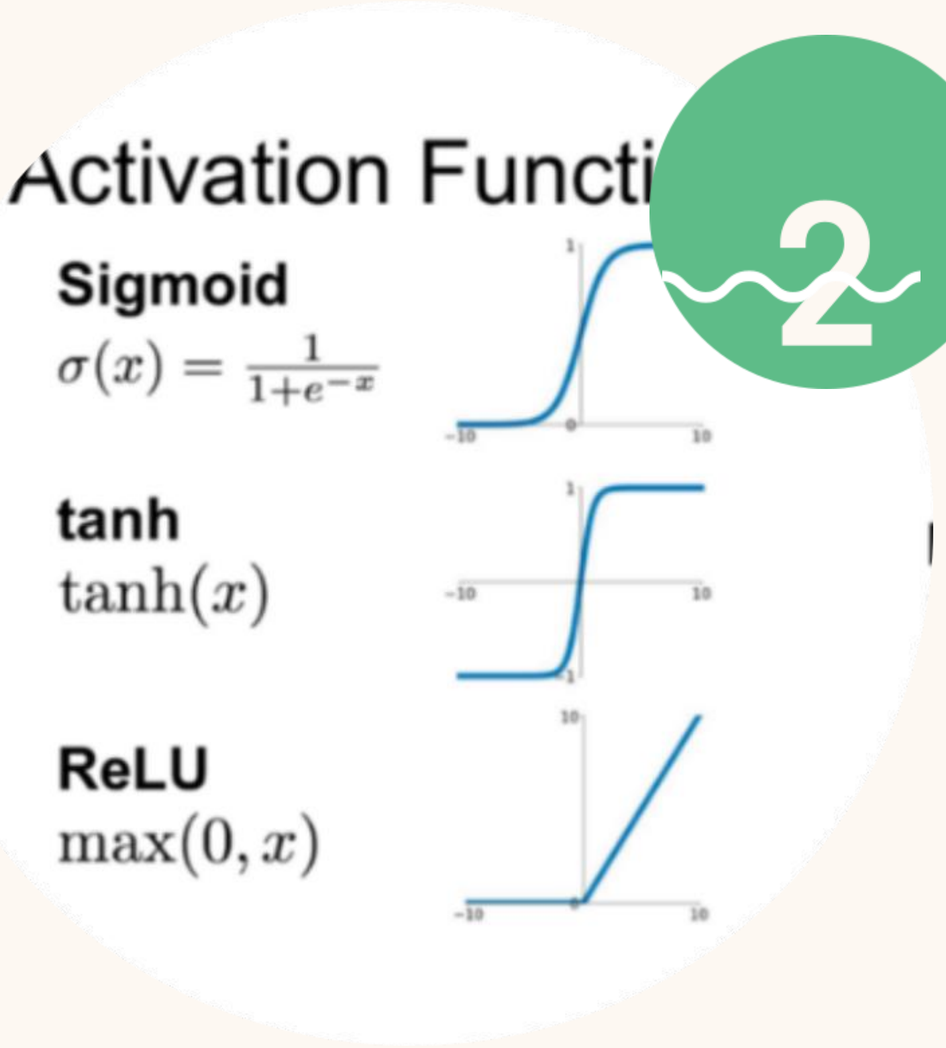
x : 입력값　　W : 가중치

y : 결과값　　b : 편향

학습 : 원하는 y 값을 만들어내기 위해 W와 b의 값을 변경해가면서
적절한 값을 찾아내는 최적화 과정

# 인공신경망


Input Layer
Hidden Layer
Output Layer
Sigmoid(X x W + b)
인공 뉴런
시스템 외부로부터
입력자료를 받아들여
시스템으로 이들을 전송
시스템 안쪽에 자리잡고
있으며 입력 값을 넘겨받아
그것들을 처리한 뒤 결과 산출
입력 값과 현재 시스템
상태에 기준하여
시스템 출력 값을 산출

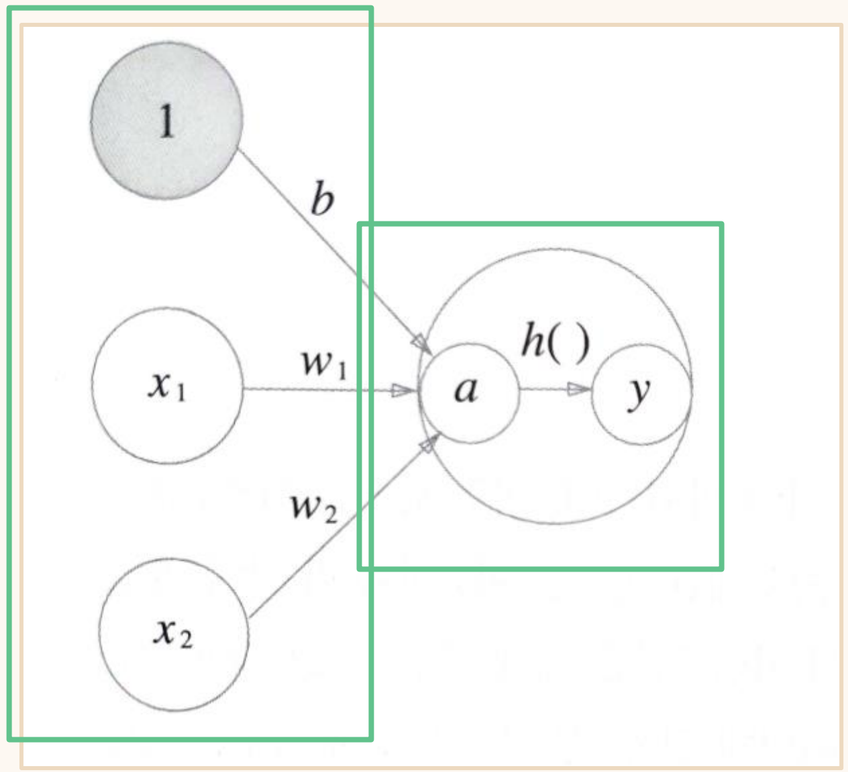



2

CHAPTER. 2

# 활성화함수

- 활성화 함수란?
- Sigmoid
- ReLU
- tanh




$$a = b + w_1x_1 + w_2x_2$$

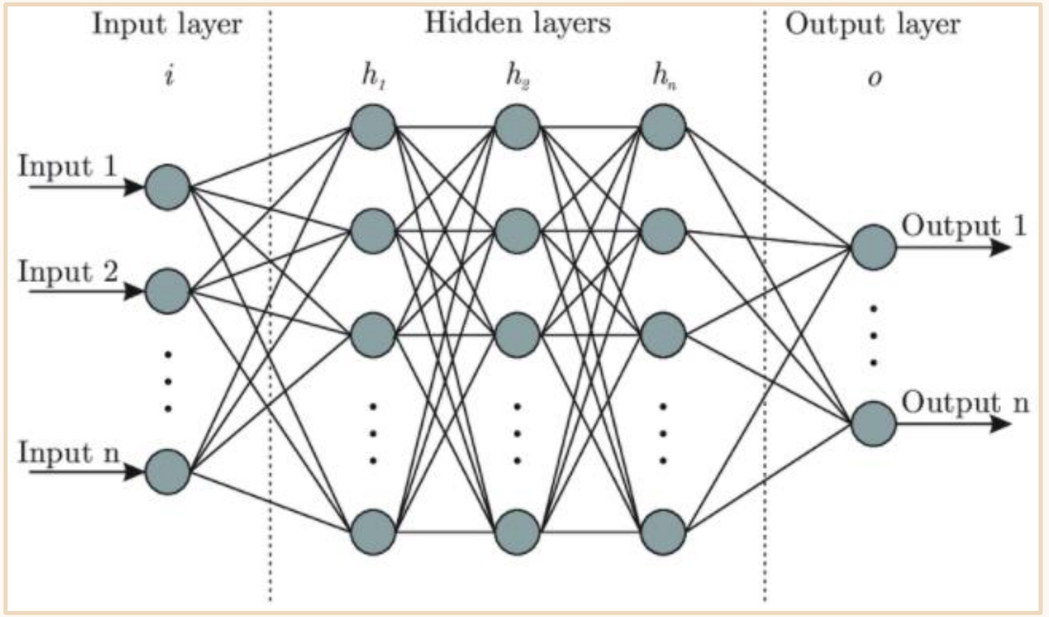
$$y = h(a)$$

## 활성화 함수

- 인공신경망을 통과해온 값을 최종적으로 어떤 값으로 만들지 결정

- 입력 신호의 총합을 출력 신호로 변환하는 함수

※ y = 활성화함수( x x W + b)




# 왜 비선형 함수를 써야할까?

선형 함수를 사용하면 신경망 층을 깊게 하는 의미가 사라짐

$$h(x) = cx$$

$$y(x) = h(h(h(x)))$$

$$y(x) = c * c * c * x$$

$$y(x) = ax \ (a = c^3)$$

선형 함수 : 1개의 직선

비선형 함수 : 직선 1개로 그릴 수 없음

# sigmoid

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

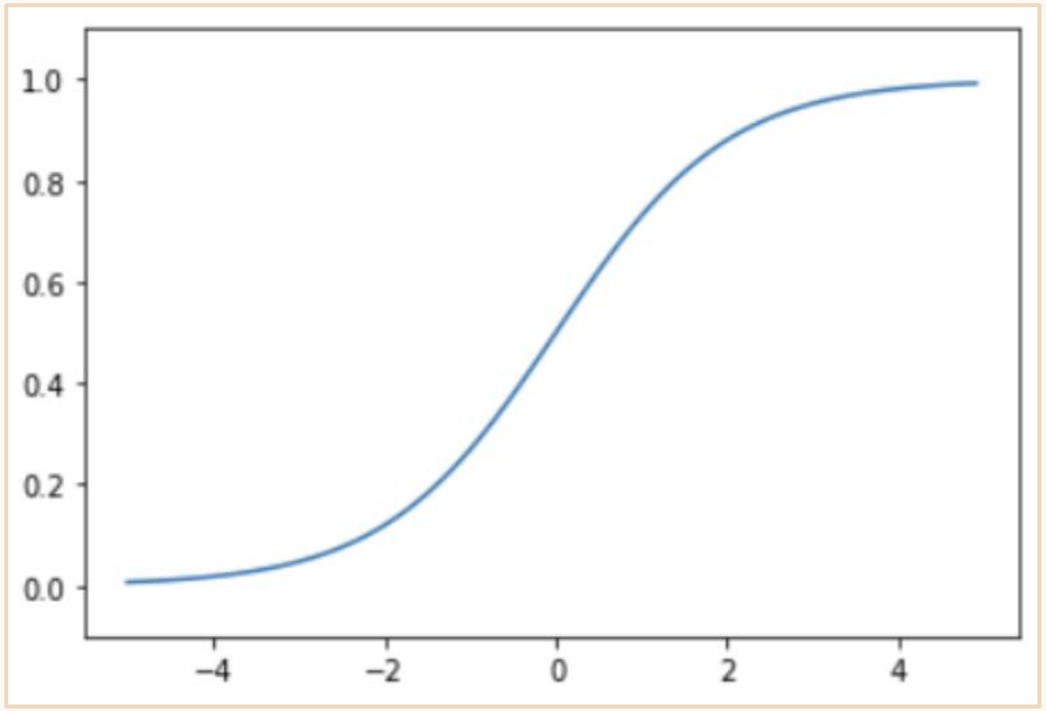
$$= e^{-x}$$

시그모이드 함수

```
import numpy as np

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

X = np.array([-1.0,1.0,2.0])
sigmoid(X)
```

```
array([0.26894142, 0.73105858, 0.88079708])
```

# sigmoid




```python
import numpy as np
import matplotlib.pylab as plt


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

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

matplotlib : 데이터를 시각화와 그래프를 그려줌

# sigmoid

## 장점

출력값이 0~1로 분류가 쉬움.

입력에 따라 값이 급격하게 변하지 않음.

매끄러운 곡선을 가져 기울기 폭주가 발생하지 않음.

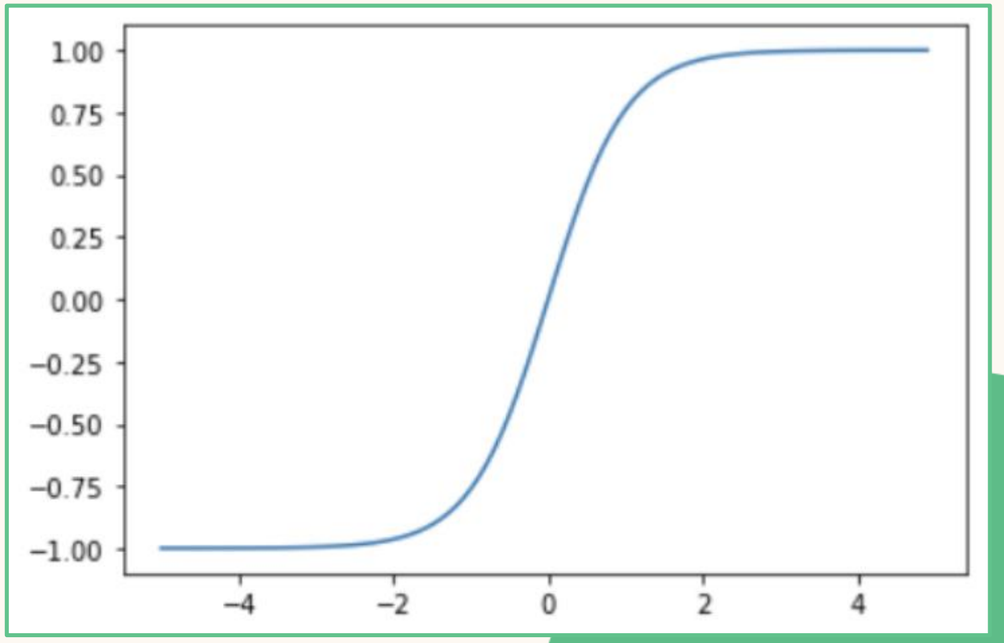
## 단점

기울기 소실이 발생할 수 있음.

함수값의 중심이 0이 아니라 학습이 느려질 수 있음.

# tanh

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




하이퍼볼릭 탄젠트 함수

# tanh

## 장점

함수의 중심값을 0으로 옮겨 학습 속도를 개선함.

## 단점

기울기 소실이 발생할 수 있음.

# ReLU

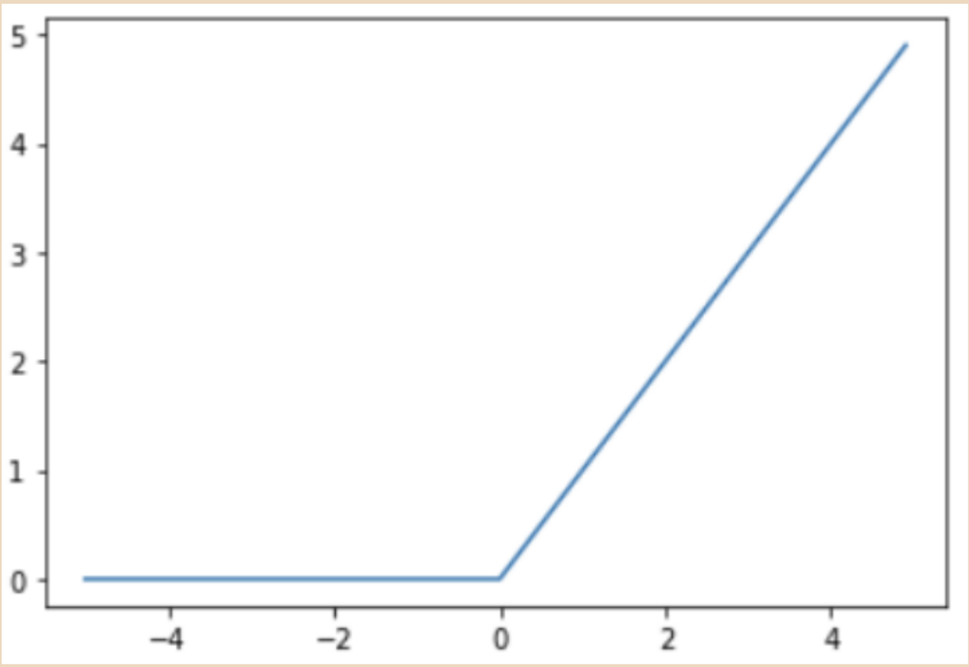
$$h(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

렐루 함수

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

X = np.array([-1.0,1.0,2.0])
ReLU(x)
```

```
array([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. , 0.1,
       0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2, 1.3, 1.4,
       1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7,
       2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, 3.9, 4. ,
       4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])
```

# ReLU




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

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

x = np.arange(-5.0, 5.0, 0.1)
y = ReLU(x)

plt.plot(x,y)
plt.show()
```

matplotlib : 데이터를 시각화와 그래프를 그려줌

# ReLU

## 장점

학습 속도가 비교적 빠름.
다른 활성화 함수에 비해 효율적인 결과가 나타남.
연산 비용이 크지않고, 구현이 매우 간단함.

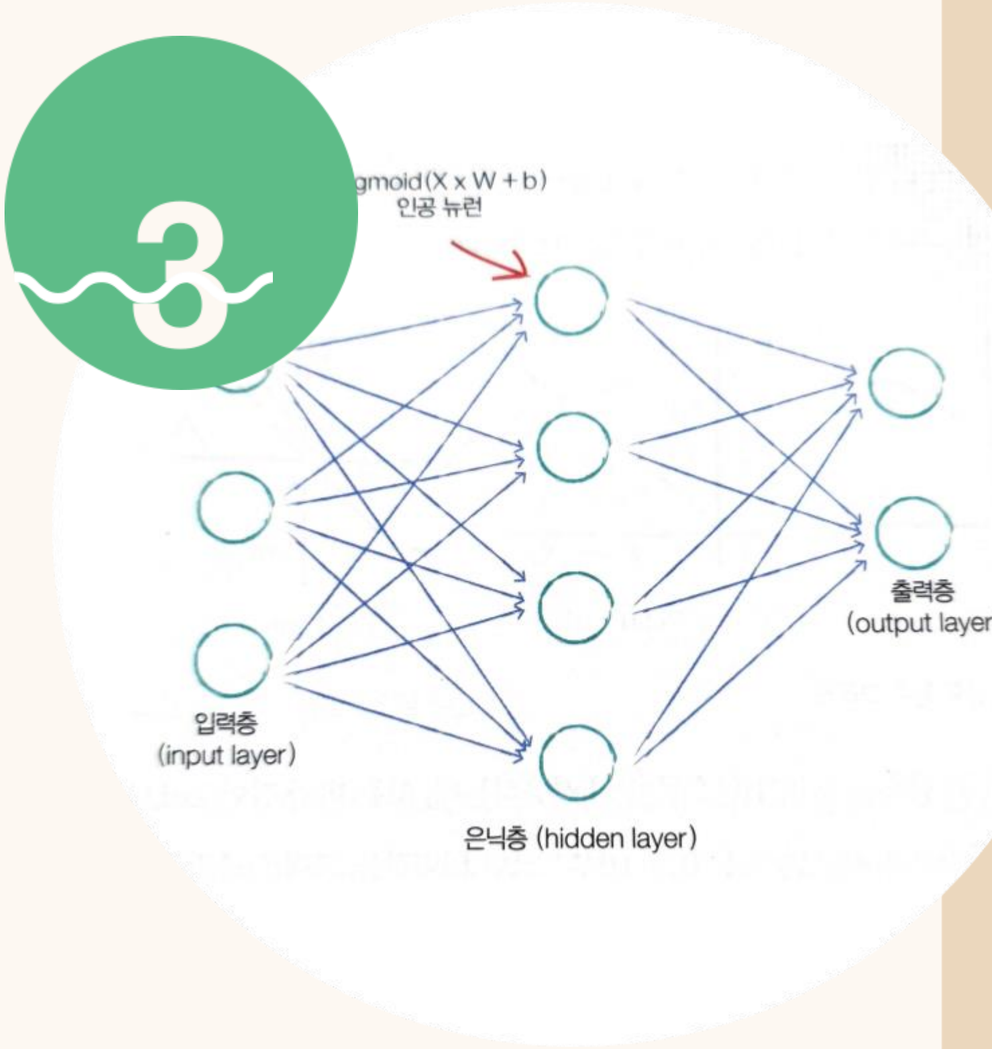
## 단점

값이 음수일 경우 학습하지 못한다.

CHAPTER. 3

# 모델 구현

- 신경망 구현
- 심층 신경망 구현



## 주제

털과 날개를 기준으로 포유류 조류 구분하는 모델

## 활성화 함수

ReLU

x : [털, 날개]  y : [기타, 포유류, 조류]

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

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

# 신경망 모델 구성

```
X = tf.placeholder(tf.float32)
Y = tf.placeholder(tf.float32)

W = tf.Variable(tf.random_uniform([2, 3], -1., 1.))

b = tf.Variable(tf.zeros([3]))

L = tf.add(tf.matmul(X, W), b)
L = tf.nn.relu(L)

model = tf.nn.softmax(L)
```

[8.04, 2.76, -6.52] -> [0.53 0.24 0.23]

# 신경망 모델 구성

```
cost = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(model), axis=1))
```

각 개별 결과에 대한 합 구하고 평균을 냄

axis =1


# 신경망 모델 학습

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)

for step in range(100):
    sess.run(train_op, feed_dict={X: x_data, Y: y_data})

    if (step + 1) % 10 == 0:
        print(step + 1, sess.run(cost, feed_dict={X: x_data, Y: y_data}))
```

```
10 1.2315385
20 1.2273331
30 1.2232035
40 1.2191473
50 1.2151623
60 1.2112455
70 1.2073952
80 1.2036089
90 1.1998845
100 1.1962202
```

# 신경망 모델 결과

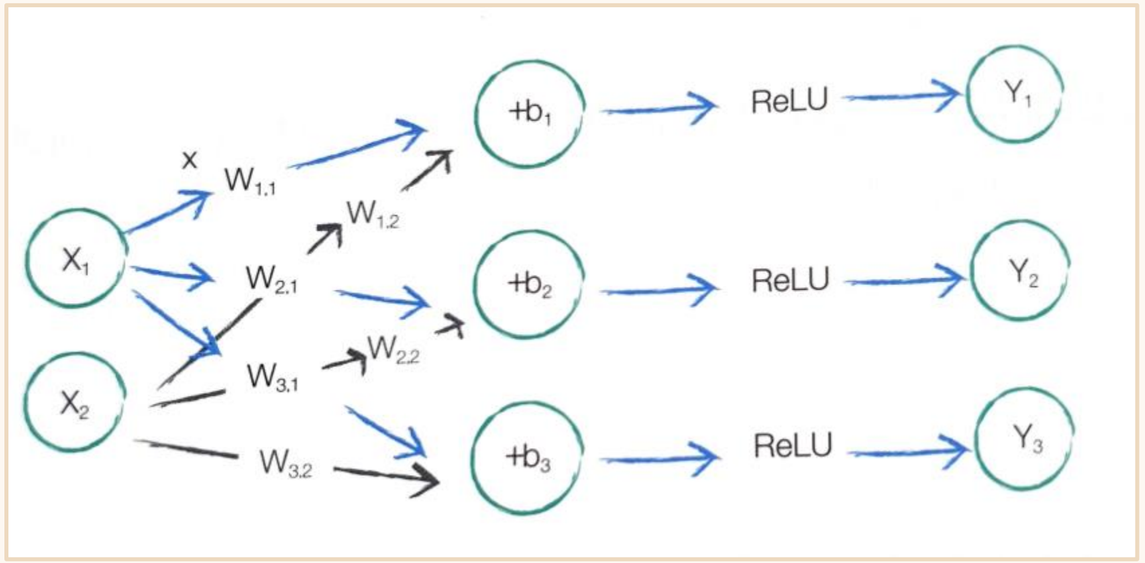
```
prediction = tf.argmax(model, 1)
target = tf.argmax(Y, 1)
print('예측값:', sess.run(prediction, feed_dict={X: x_data}))
print('실제값:', sess.run(target, feed_dict={Y: y_data}))

is_correct = tf.equal(prediction, target)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도: %.2f' % sess.run(accuracy * 100, feed_dict={X: x_data, Y: y_data}))
```

```
예측값: [0 1 1 0 0 0]
실제값: [0 1 2 0 0 2]
정확도: 66.67
```

=> 정확도가 떨어짐

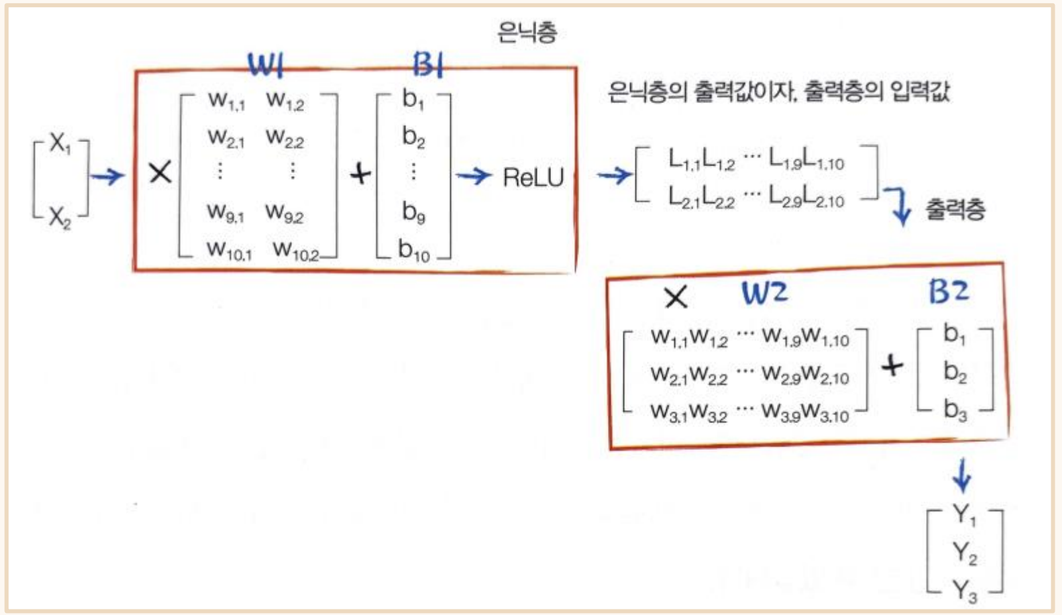
# 신경망 / 심층 신경망

## 2층 신경망




```
W = tf.Variable(tf.random_uniform([2, 3], -1., 1.))

b = tf.Variable(tf.zeros([3]))
```

## 3층 신경망




```
W1 = tf.Variable(tf.random_uniform([2, 10], -1., 1.))
W2 = tf.Variable(tf.random_uniform([10, 3], -1., 1.))

b1 = tf.Variable(tf.zeros([10]))
b2 = tf.Variable(tf.zeros([3]))
```

# 신경망 모델 구성

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

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

# 신경망 모델 구성

```
L1 = tf.add(tf.matmul(X, W1), b1)
L1 = tf.nn.relu(L1)

model = tf.add(tf.matmul(L1, W2), b2)
```

```
cost = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(labels=Y, logits=model))
```

각 개별 결과에 대한 합 구하고 평균을 냄

# 신경망 모델 결과

```
for step in range(100):
    sess.run(train_op, feed_dict={X: x_data, Y: y_data})

    if (step + 1) % 10 == 0:
        print(step + 1, sess.run(cost, feed_dict={X: x_data, Y: y_data}))

prediction = tf.argmax(model, 1)
target = tf.argmax(Y, 1)
print('예측값:', sess.run(prediction, feed_dict={X: x_data}))
print('실제값:', sess.run(target, feed_dict={Y: y_data}))

is_correct = tf.equal(prediction, target)
accuracy = tf.reduce_mean(tf.cast(is_correct, tf.float32))
print('정확도: %.2f' % sess.run(accuracy * 100, feed_dict={X: x_data, Y: y_data}))
```

```
10 0.96343166
20 0.772008
30 0.63483053
40 0.5201743
50 0.42044616
60 0.33820453
70 0.26924652
80 0.21293397
90 0.16839021
100 0.13388805
예측값: [0 1 2 0 0 2]
실제값: [0 1 2 0 0 2]
정확도: 100.00
```

# 감사합니다

장혜선