딥러닝

머신러닝/딥러닝

임 경 태

Week	Chapter	Contents
1	1, 2장	강의 소개, 파이썬 복습
2	1, 3장	파이썬 복습, Numpy, Pandas
3	1, 4장	딥러닝을 위한 미분
4	5장	회귀
5	5장	분류
6	6장	XOR문제
7	7장	딥러닝
8	1~7장	중간고사
9	8장	MNIST 필기체 구현 (팀 프로젝트)
10	9장	오차역전파
11	11장	합성곱 신경망(CNN)
12	12장	순환 신경망(RNN)
13	10장	자율주행 (Collision Avoidanve, Transfer Learning)
14	11장	자율주행 (Load Following)
15	8~12장	기말고사 (or 프로젝트 발표)

CONTENTS

목적 : ANN, DNN에 대한 이해와 XOR게이트 학습

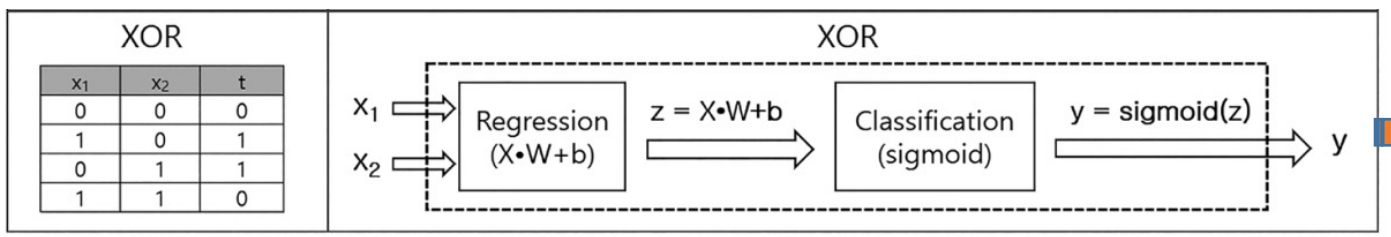
목표 : FFNN을 이용하여 XOR게이트 학습

내용 : 신경망, FFNN, 딥러닝을 이용한 XOR문제 해결

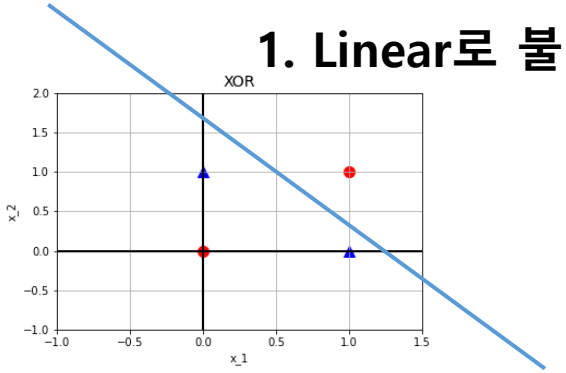
CONTENTS

XOR 예제로 알 수 있는 점

- 한 개의 분류 시스템 혹은 perceptron으로는 XOR게이트를 표현할 수 없음을 알았다.



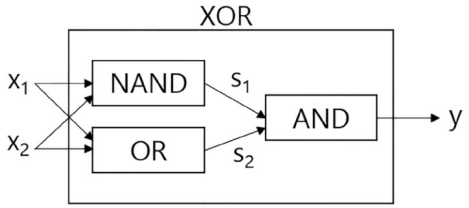
1. Linear로 불가



XOR 예제로 알 수 있는 점

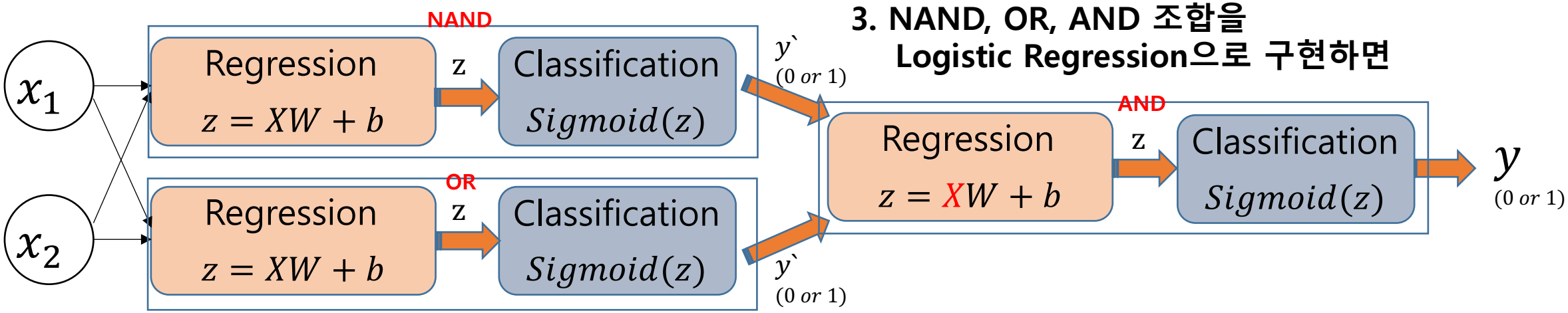
→ NAND, OR, AND게이트의 조합으로 XOR문제를 해결했다.

2. NAND, OR, AND 조합 가능 (Nonlinear)



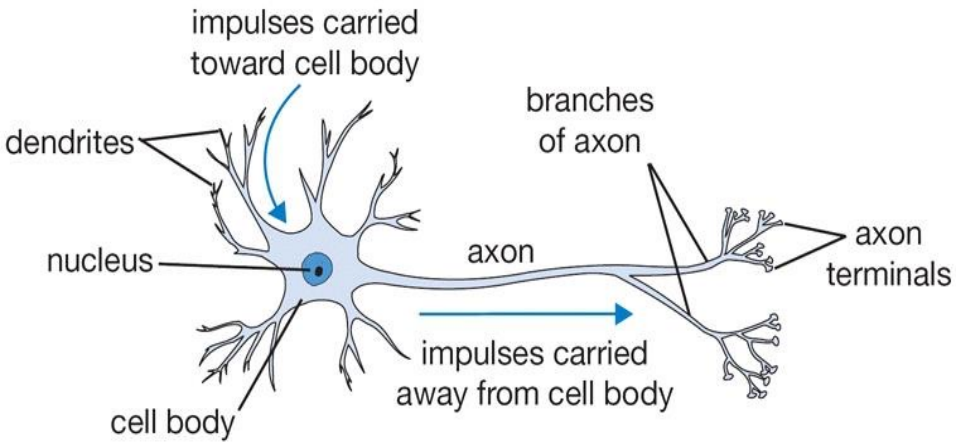

```
def XOR(x_1, x_2):
    s1 = NAND(x_1, x_2)
    s2 = OR(x_1, x_2)
    y = AND(s1, s2)
    return y
```

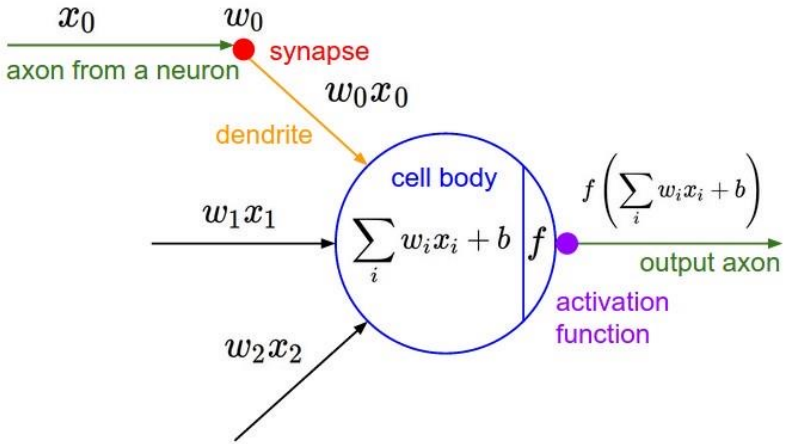
3. NAND, OR, AND 조합을 Logistic Regression으로 구현하면



XOR 예제로 알 수 있는 점

- **퍼셉트론의 층**을 쌓았더니 **비선형 관계가 표현**이 되었다.
- 복잡한 관계가 표현이 되었다.
- 퍼셉트론 뭉치 = 신경망



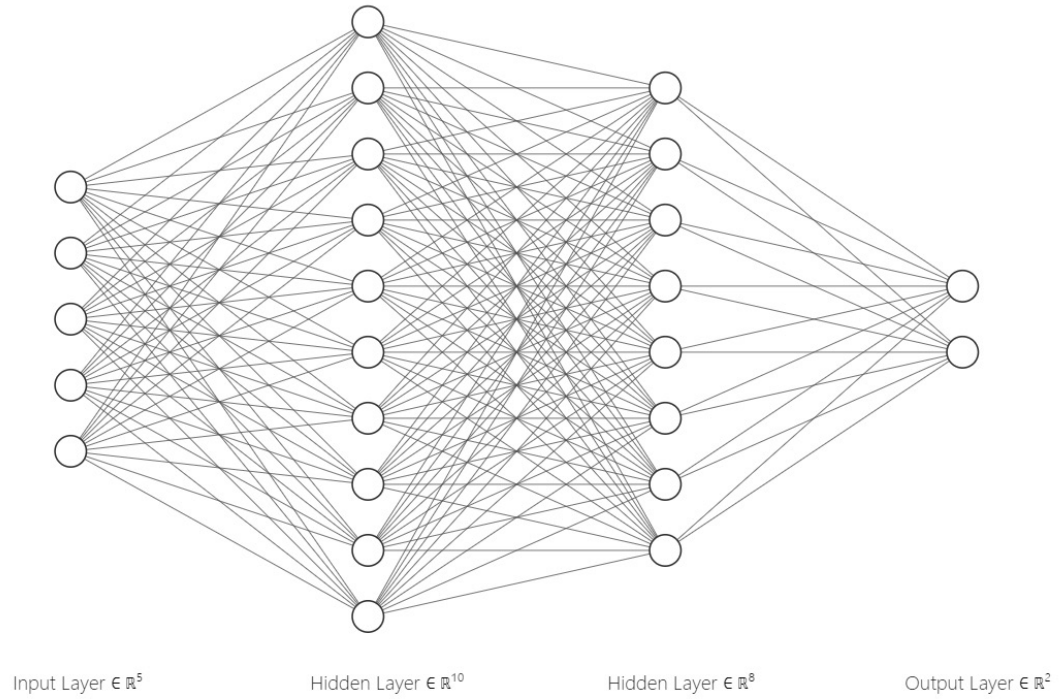


그림출처: Stanford CS231n_lecture2

CONTENTS

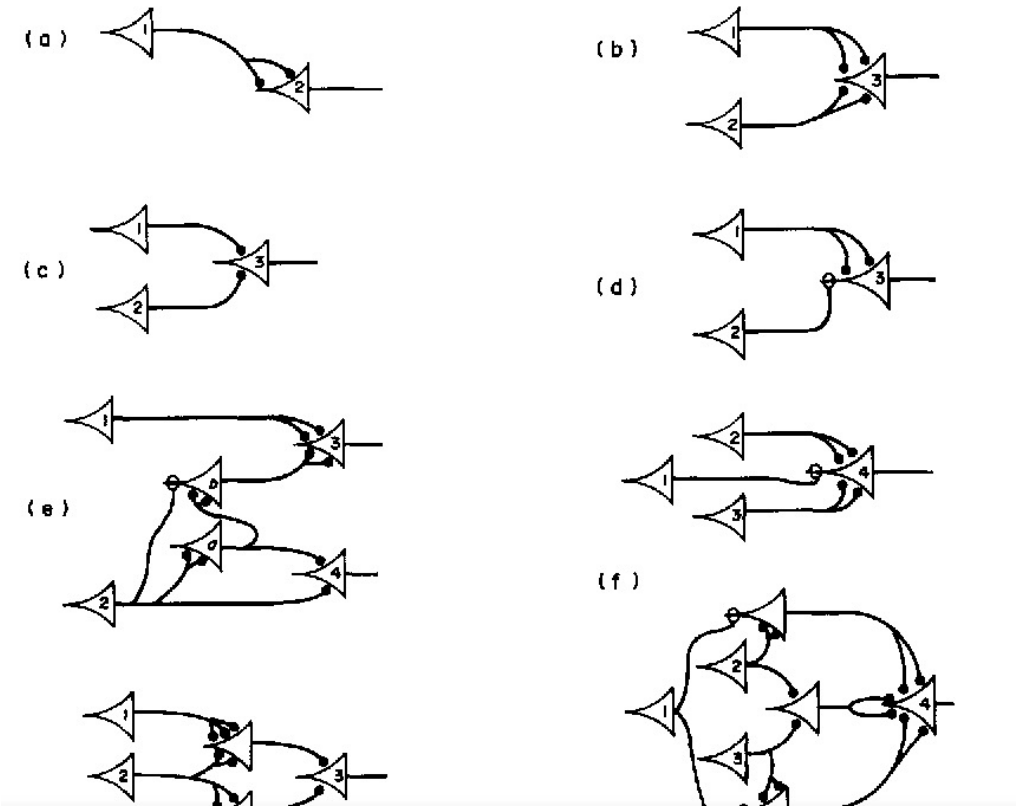
인공 신경망이란?

- **인공신경망**(artificial neural network, **ANN**)은 기계학습과 인지과학에서 생물학의 신경망에서 영감을 얻은 통계학적 학습 알고리즘.
- 시냅스의 결합으로 네트워크를 형성한 인공 뉴런이 학습을 통하여 시냅스의 **결합 세기(가중치)**를 변화시켜 문제해결능력을 가지는 비선형 모델



인공 신경망 역사

- **1943**년 워런 매컬러, 월터 피츠 – 명제 논리를 사용해 생물학적 뉴런의 네트워크를 **기계적으로 모델링**, 최초의 인공 신경망

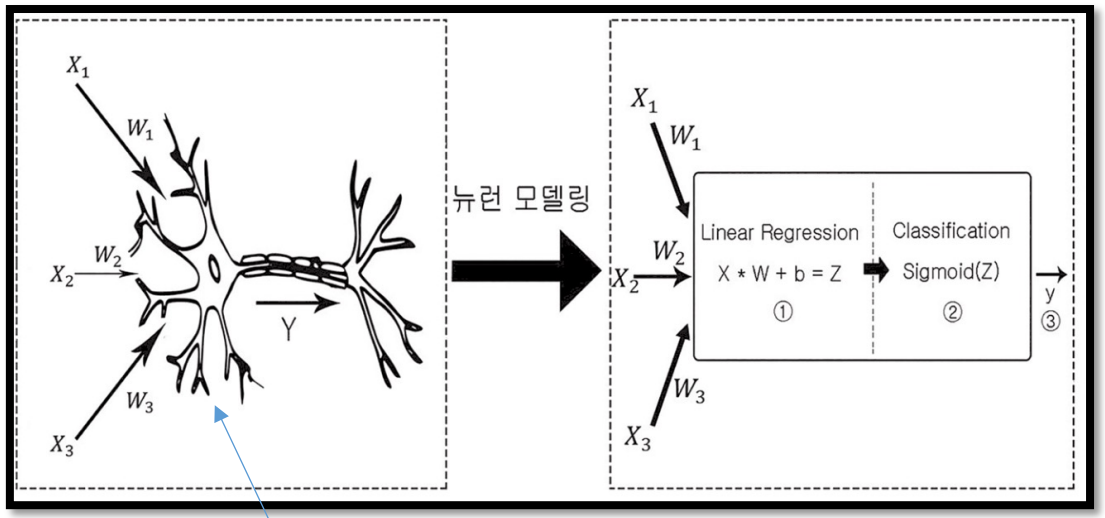


이미지 출처 : https://www.semanticscholar.org/paper/A-logical-calculus-of-the-ideas-immanent-in-nervous-McCulloch-Pitts/090c5a5df345ab60c41d6de02b3e366e1a27cf43

인공 신경망, 왜 이제야 뜰까?

1. 훈련을 위한 **데이터**가 많아짐
2. 하드웨어(특히 **GPU**)의 발전, 클라우드 컴퓨팅 → 연산 속도↑
3. **훈련 알고리즘**의 향상
4. 인공지능에 대한 투자, 관심 향상

신경망 → 인공 신경망 모델링



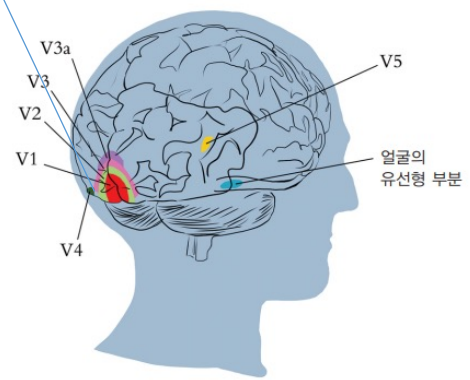
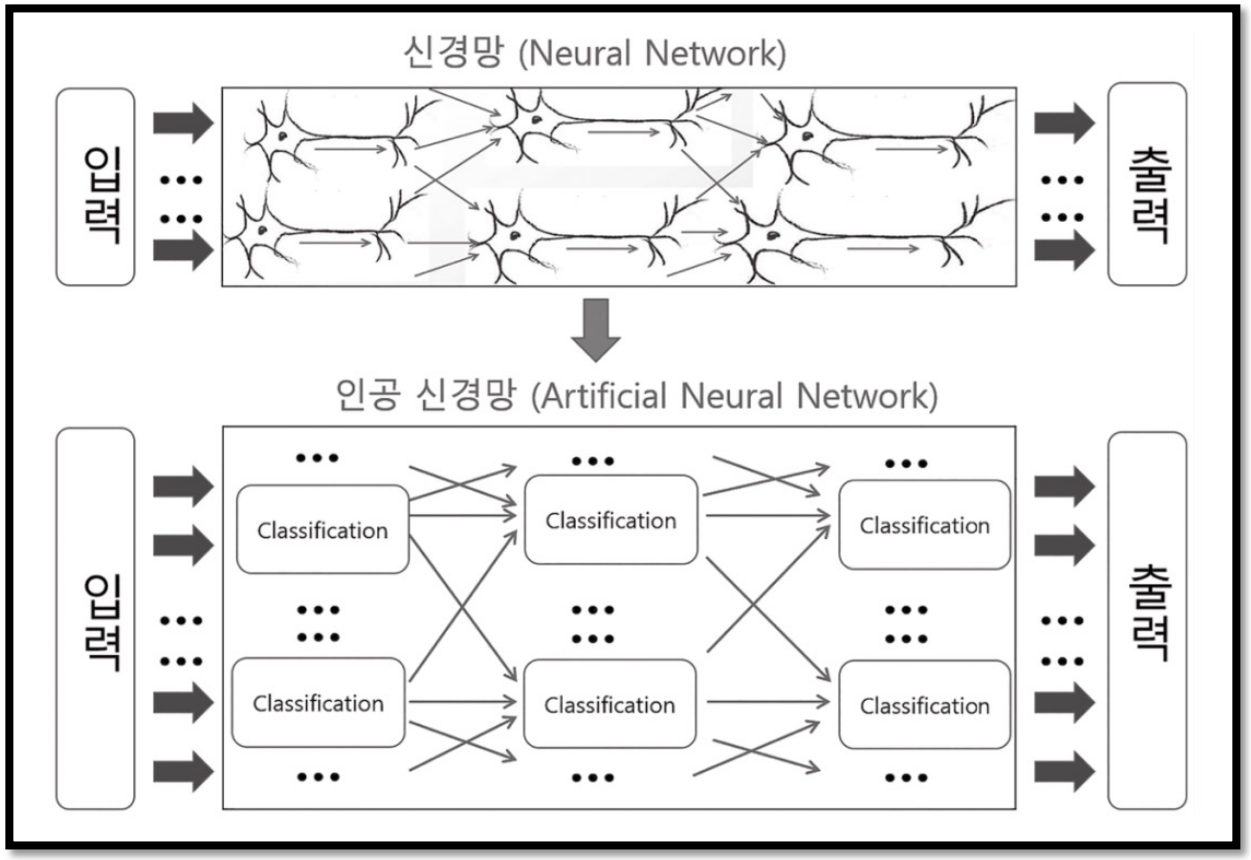


그림 1.7 대뇌피질의 시각 영역. V1 영역이 눈에서 입력을 받고 모서리 방향을 감지하는 단순한 셀을 가지고 있습니다. (V2, V3, V3a 영역을 포함해) 무수히 많은 연속적인 뉴런 층을 통해 정보를 재조합하여 추상적인 시각 자극을 표현합니다. (여기에 보이는) 사람의 뇌에는 색(V4), 동작(V5), 사람의 얼굴(얼굴의 유선형 부분)을 감지하는 데 집중적으로 특화된 뉴런을 가진 영역이 있습니다.

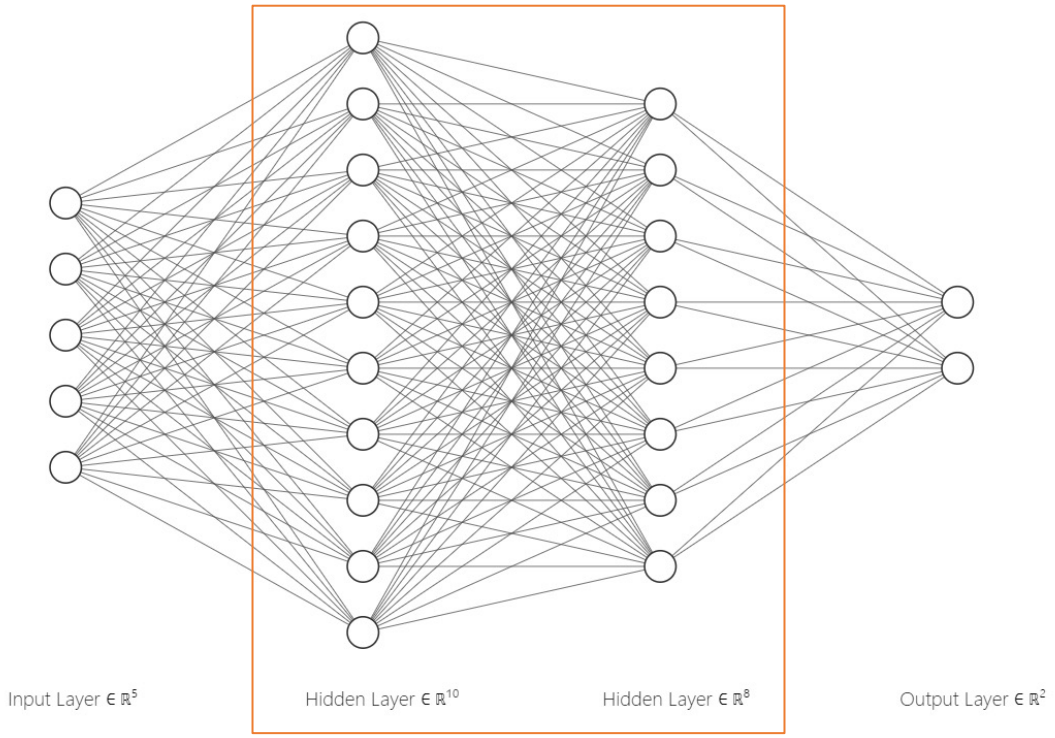
이미지출처: **Deep Learning Illustrated**



이미지 출처 : 머신러닝을 위한 파이썬 한 조각

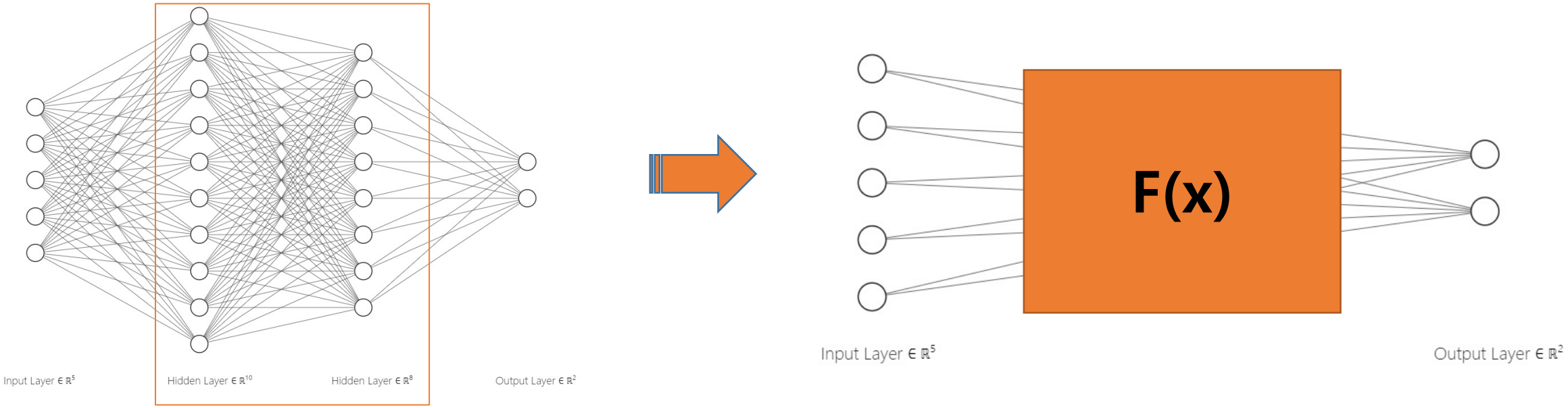
DNN – Deep Neural Network란?

- 인공신경망 중 은닉층(hidden layer)이 1개 이상 있을 시 Deep Neural Network(DNN)이라고 하며 층이 많을수록 deep하다고 한다



은닉층, Black Box

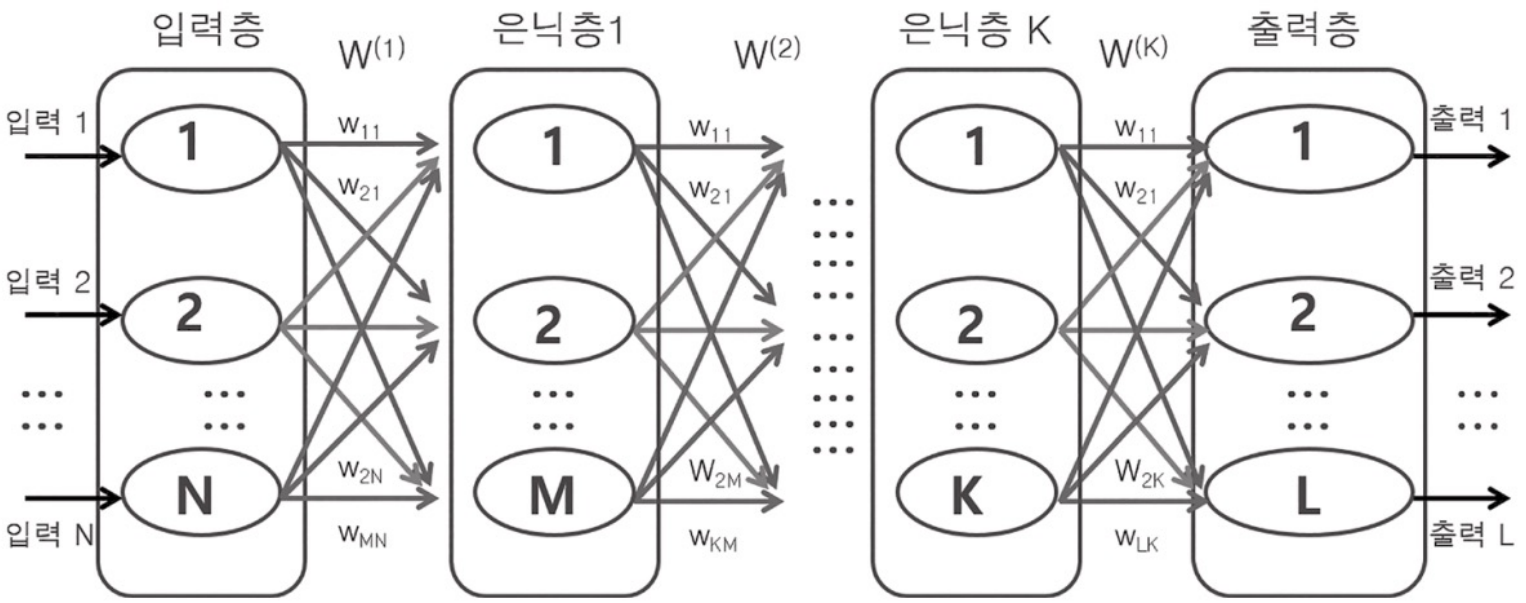
- 은닉층에서 수행되는 연산은 알기가 어려워서 은닉층을 "블랙박스"라고 도 한다.
- 블랙박스를 완벽하게 해석할 수 있으면? → XAI



DNN 뜯어보기

- 은닉층이 증가할수록 정확도가 높아지지만 연산량 증가(학습 시간 증가)

→ **trade-off** between **Accuracy** and **Training time**

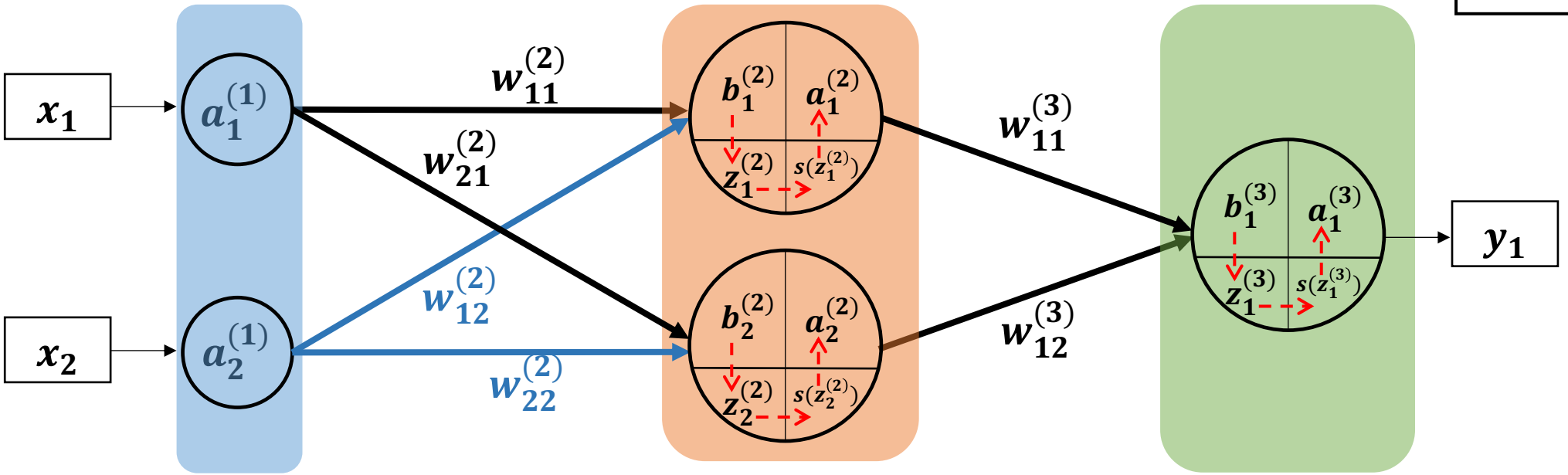


CONTENTS

Feed Forward Neural Network (FFNN)

FFNN: 입력값이 출력까지 Forward 방향으로 흘러가는 NN

$w_{21}^{(2)}$
- 2층으로 전달되는 가중치
- =노드1에서 노드2로 전달되는 신호를 얼마나 강/약하게 함?
- 앞 층의 1번째 뉴런
- 다음 층의 2번째 뉴런



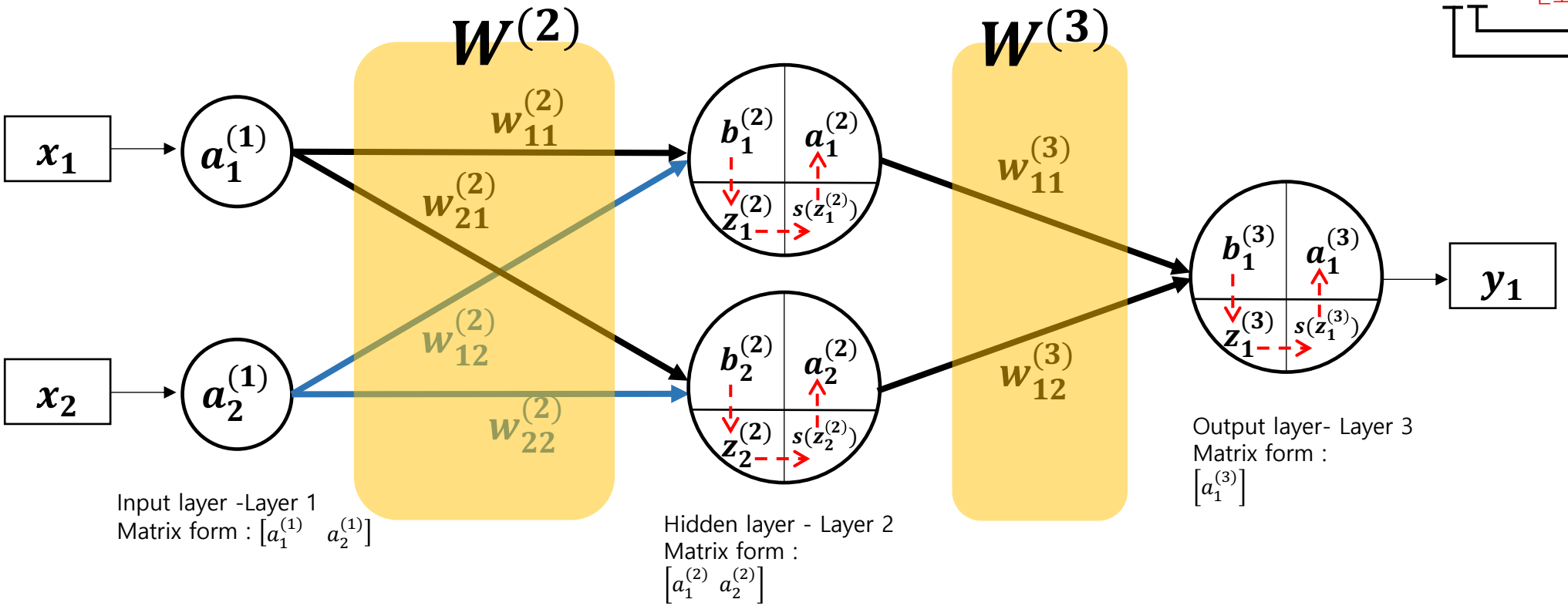
Input layer -Layer 1
Matrix form :
$\begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix}$

Hidden layer - Layer 2
Matrix form :
$\begin{bmatrix} a_1^{(2)} & a_2^{(2)} \end{bmatrix}$

Output layer- Layer 3
Matrix form :
$\begin{bmatrix} a_1^{(3)} \end{bmatrix}$

Feed Forward Neural Network (FFNN)

FFNN: 입력값이 출력까지 Forward 방향으로 흘러가는 NN



$w_{21}^{(2)}$
2층으로 전달되는 가중치
=노드1에서 노드2로 전달되는 신호를 얼마나 강/약하게 함?
앞 층의 1번째 뉴런
다음 층의 2번째 뉴런

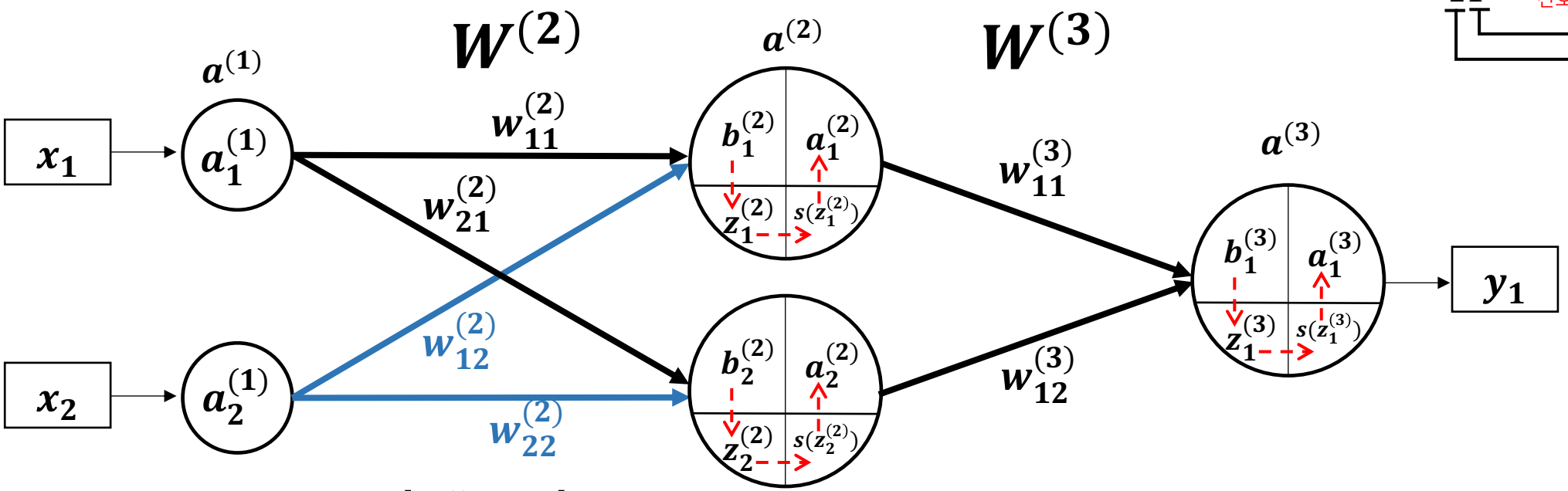
$$W^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix}$$

$$W^{(3)} = \begin{bmatrix} w_{11}^{(3)} & w_{12}^{(3)} \end{bmatrix}$$

Feed Forward Neural Network (FFNN)

FFNN: 입력값이 출력까지 Forward 방향으로 흘러가는 NN

$w_{21}^{(2)}$
- 2층으로 전달되는 가중치
- =노드1에서 노드2로 전달되는 신호를 얼마나 강/약하게 함?
- 앞 층의 1번째 뉴런
- 다음 층의 2번째 뉴런



$[a_1^{(1)} \quad a_2^{(1)}]$ $\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix}$ $[a_1^{(2)} \quad a_2^{(2)}]$ $[w_{11}^{(3)} \quad w_{12}^{(3)}]$ $[a_1^{(3)}]$

$$a^{(1)} \cdot W^{(2)} + b^{(1)} = z^{(2)}$$

$$[a_1^{(1)} \quad a_2^{(1)}] \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{bmatrix} + [b_1^{(1)} \quad b_2^{(1)}] = [z_1^{(2)} \quad z_2^{(2)}]$$

Linear Regression

$$sigmoid(z^{(2)}) = a^{(2)}$$

$$[s(z_1^{(2)}) \quad s(z_2^{(2)})] = [a_1^{(2)} \quad a_2^{(2)}]$$

Logistic Regression

$$a^{(2)} \cdot W^{(3)} + b^{(3)} = z^{(3)}$$

$$[a_1^{(2)} \quad a_2^{(2)}] \begin{bmatrix} w_{11}^{(3)} \\ w_{12}^{(3)} \end{bmatrix} + [b_1^{(3)}] = [z_1^{(3)}]$$

Linear Regression

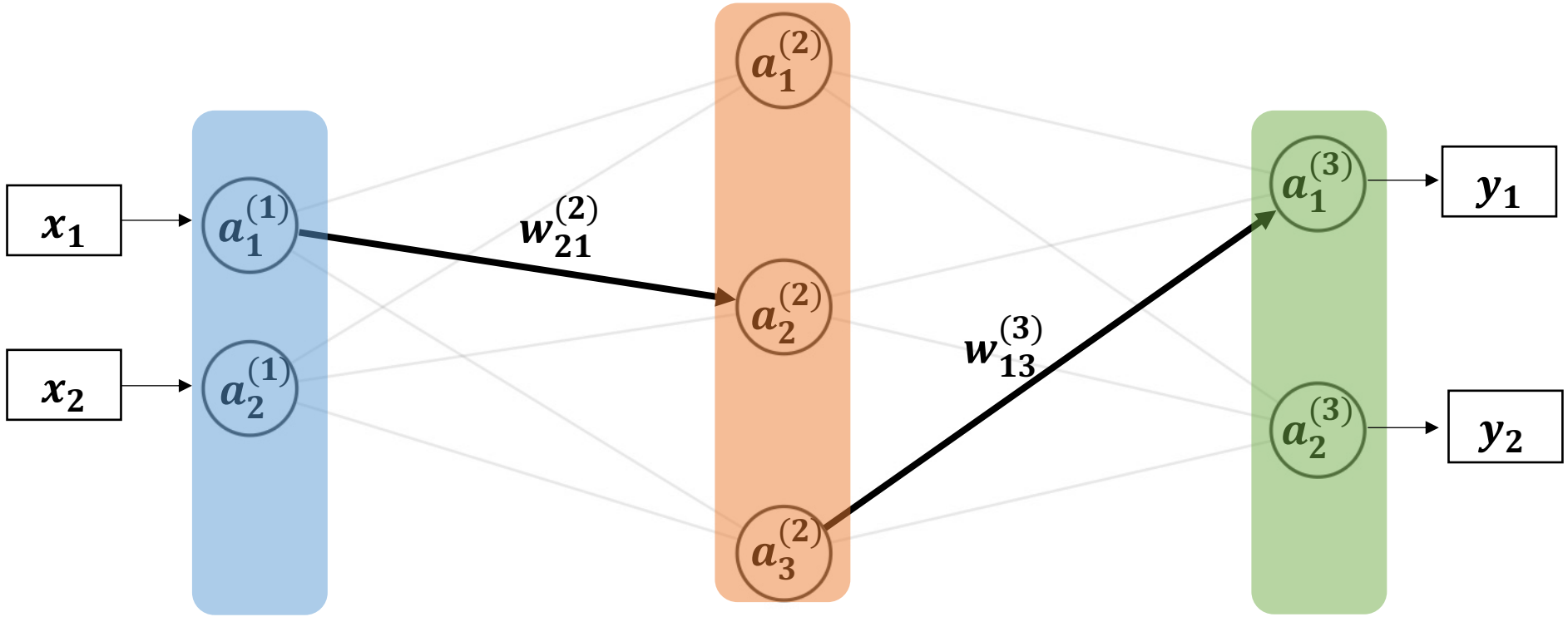
$$[s(z_1^{(3)})] = [a_1^{(3)}] = y_1$$

Logistic Regression

약간 더 복잡한 FFNN구조

- Layer 수, Layer의 노드 수, Output의 수는 Hyperparameter로 사용자가 지정

$w_{21}^{(2)}$
2층으로 전달되는 가중치
=노드1에서 노드2로 전달되는 신호를 얼마나 강/약하게 함?
앞 층의 1번째 뉴런
다음 층의 2번째 뉴런

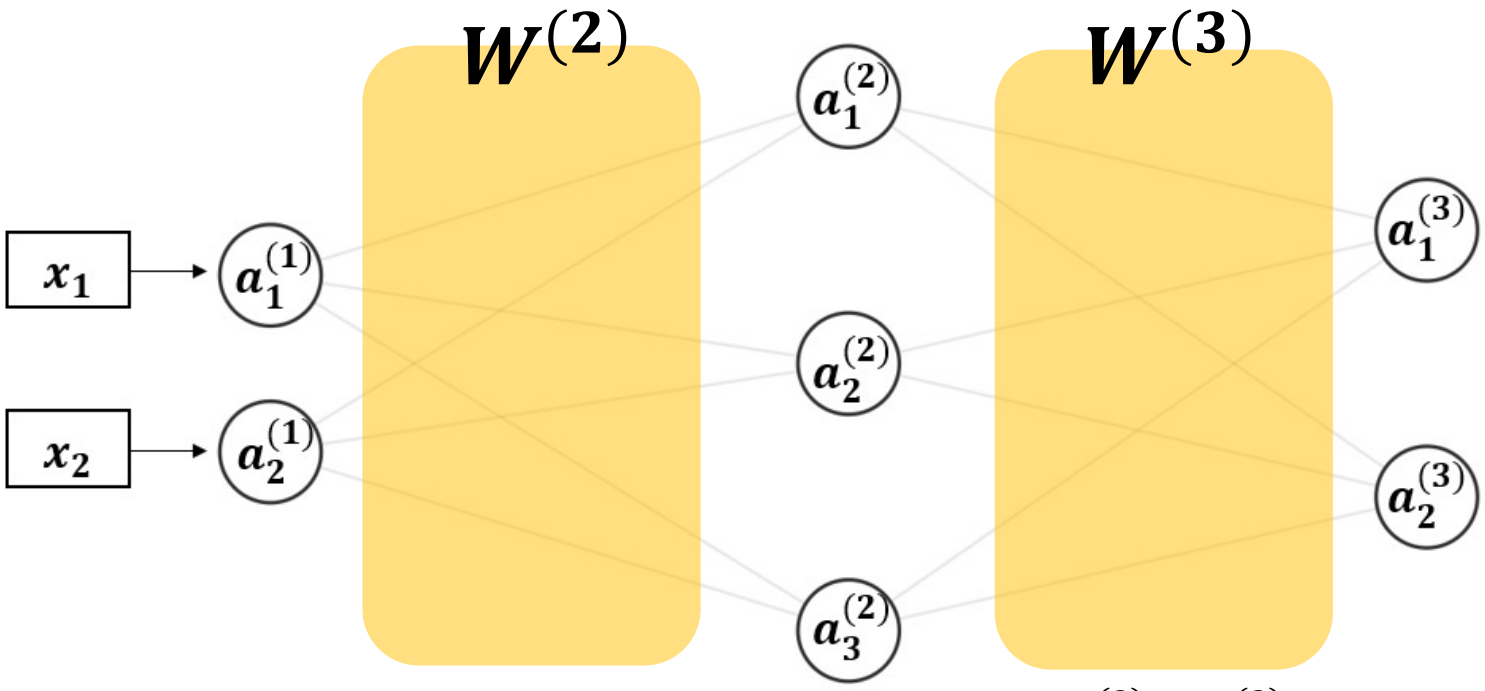


Input layer -Layer 1
Matrix form :
$[a_1^{(1)} \quad a_2^{(1)}]$

Hidden layer - Layer 2
Matrix form :
$[a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}]$

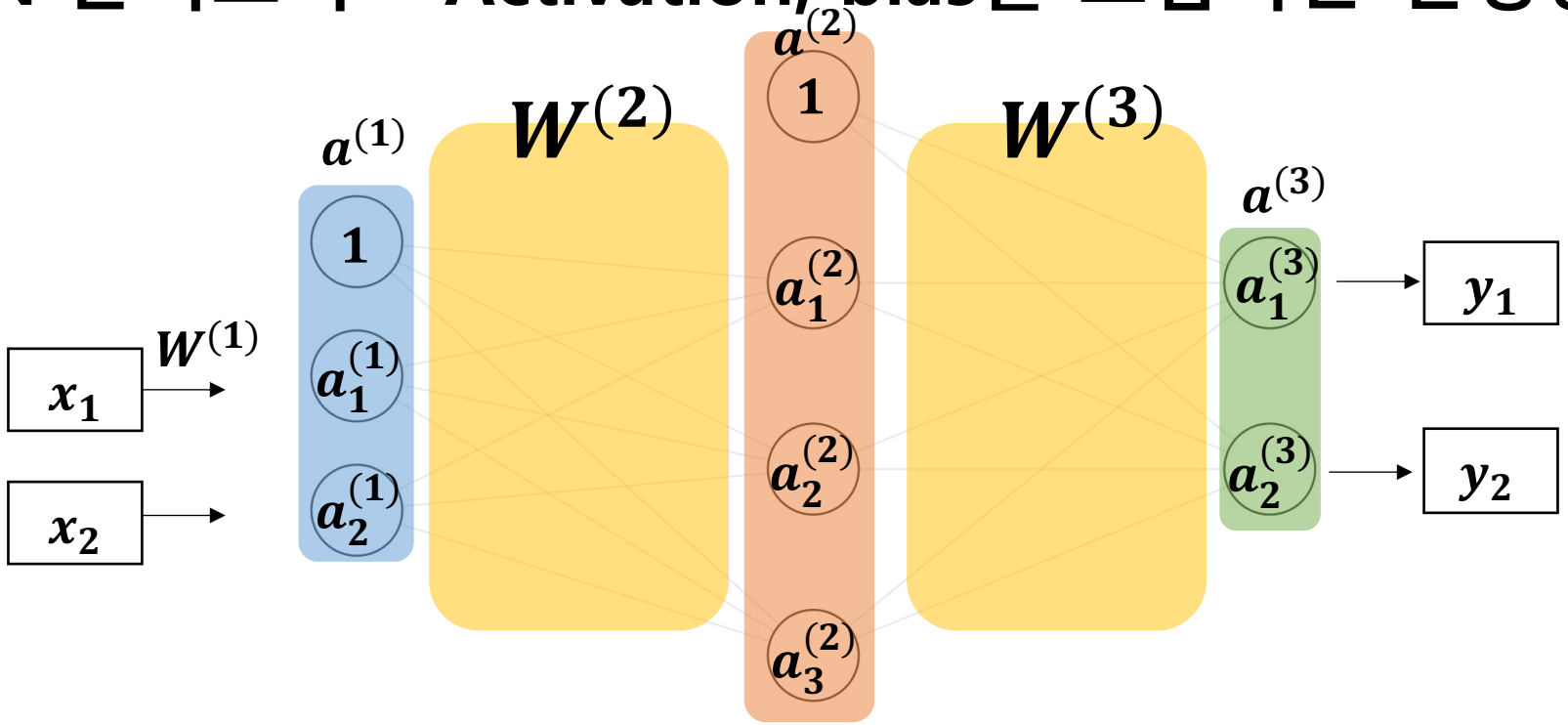
Output layer- Layer 3
Matrix form :
$[a_1^{(3)} \quad a_2^{(3)}]$

FFNN 뜯어보기 – 가중치를 행렬로 표기하면?



$$W^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \end{bmatrix} \qquad W^{(3)} = \begin{bmatrix} w_{11}^{(3)} & w_{21}^{(3)} \\ w_{12}^{(3)} & w_{22}^{(3)} \\ w_{13}^{(3)} & w_{23}^{(3)} \end{bmatrix}$$

FFNN 뜯어보기 – Activation, bias를 포함하는 신경망 구조



Input

x_1, x_2

$$z^{(2)} = \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} W^{(2)} + b^{(2)}$$

$$= \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} & b_2^{(2)} & b_3^{(2)} \end{bmatrix}$$

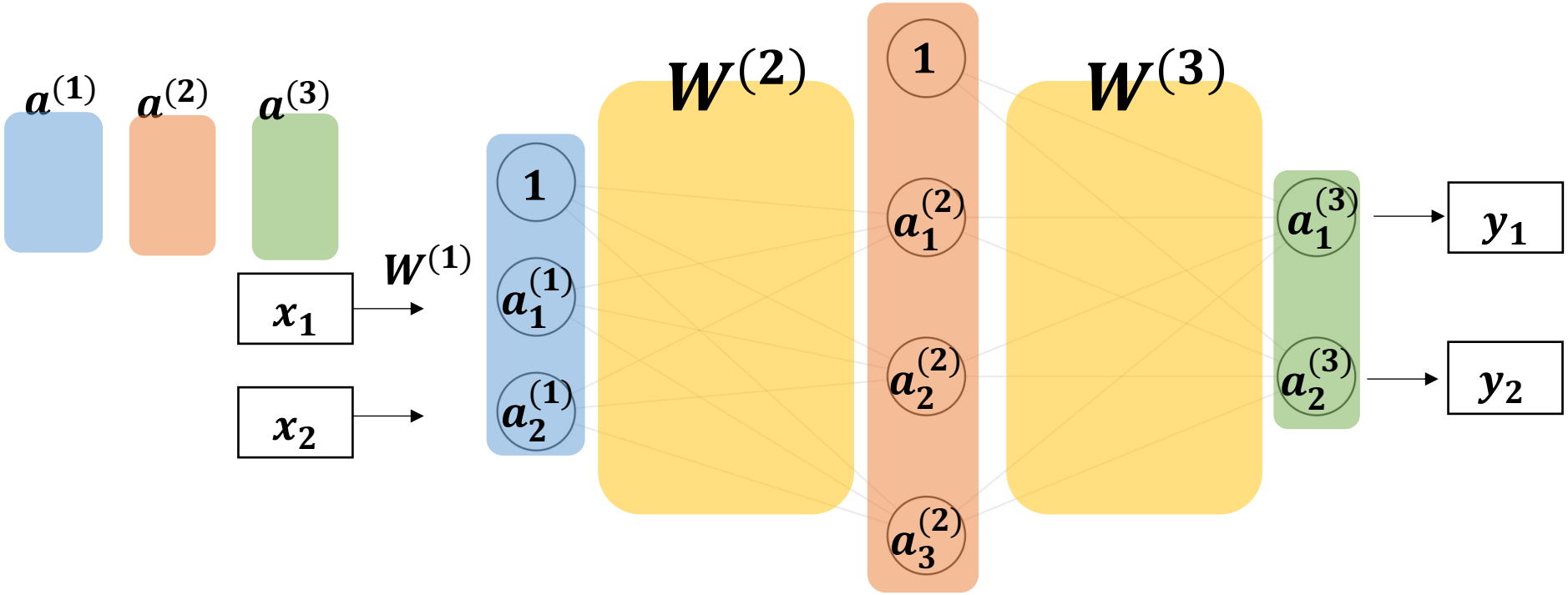
$$= \begin{bmatrix} z_1^{(2)} & z_2^{(2)} & z_3^{(2)} \end{bmatrix}$$

$$a^{(2)} = S\left(z^{(2)}\right)$$

$$= \begin{bmatrix} S(z_1^{(2)}) & S(z_2^{(2)}) & S(z_3^{(2)}) \end{bmatrix}$$

$$= \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix}$$

FFNN 뜯어보기 – Activation, bias를 포함하는 신경망 구조



$$z^{(3)} = a^{(2)}W^{(3)} + b^{(3)} = \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix} \boldsymbol{W^{(3)}} + \boldsymbol{b^{(3)}}$$

$$= \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix} \begin{bmatrix} w_{11}^{(3)} & w_{21}^{(3)} \\ w_{12}^{(3)} & w_{22}^{(3)} \\ w_{13}^{(3)} & w_{23}^{(3)} \end{bmatrix} + \begin{bmatrix} b_1^{(3)} & b_2^{(3)} \end{bmatrix}$$

$$= \begin{bmatrix} z_1^{(3)} & z_2^{(3)} \end{bmatrix}$$

$(1\times3)(3\times2) + (1\times2) \rightarrow (1\times2)$

$$a^{(3)} = S\left(z^{(3)}\right) = \begin{bmatrix} S(z_1^{(3)}) & S(z_2^{(3)}) \end{bmatrix} = \begin{bmatrix} a_1^{(3)} & a_2^{(3)} \end{bmatrix}$$

$(1\times2) \rightarrow (1\times2)$

$$\begin{bmatrix} y_1 & y_2 \end{bmatrix} = \begin{bmatrix} a_1^{(3)} & a_2^{(3)} \end{bmatrix}$$

FFNN 뜯어보기 - 정리

$$a^{(1)} = [a_1 \quad a_2] = [x_1 \quad x_2]W^{(1)} = [x_1 \quad x_2]\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = [x_1 \quad x_2]$$

Input

$\boldsymbol{x_1, x_2}$

$$z^{(2)} = \left[a_1^{(1)} \quad a_2^{(1)}\right]\boldsymbol{W^{(2)}} + \boldsymbol{b^{(2)}}$$
$$= \left[a_1^{(1)} \quad a_2^{(1)}\right]\begin{bmatrix}w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)}\\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)}\end{bmatrix} + \left[b_1^{(2)} \quad b_2^{(2)} \quad b_3^{(2)}\right]$$
$$= \left[z_1^{(2)} \quad z_2^{(2)} \quad z_3^{(2)}\right]$$

$(1\times2)(2\times3) + (1\times3) \rightarrow (1\times3)$

$$a^{(2)} = S\left(z^{(2)}\right) = \left[S(z_1^{(2)}) \quad S(z_2^{(2)}) \quad S(z_3^{(2)})\right] = \left[a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}\right]$$

$(1\times3) \rightarrow (1\times3)$

$$z^{(3)} = \left[a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}\right]\boldsymbol{W^{(3)}} + \boldsymbol{b^{(3)}}$$
$$= \left[a_1^{(2)} \quad a_2^{(2)} \quad a_3^{(2)}\right]\begin{bmatrix}w_{11}^{(3)} & w_{21}^{(3)}\\ w_{12}^{(3)} & w_{22}^{(3)}\\ w_{13}^{(3)} & w_{23}^{(3)}\end{bmatrix} + \left[b_1^{(3)} \quad b_2^{(3)}\right]$$
$$= \left[z_1^{(3)} \quad z_2^{(3)}\right]$$

$(1\times3)(3\times2) + (1\times2) \rightarrow (1\times2)$

$$a^{(3)} = S\left(z^{(3)}\right) = \left[S(z_1^{(3)}) \quad S(z_2^{(3)})\right] = \left[a_1^{(3)} \quad a_2^{(3)}\right]$$

$(1\times2) \rightarrow (1\times2)$

Output

$$[y_1 \quad y_2] = \left[a_1^{(3)} \quad a_2^{(3)}\right]$$

CONTENTS

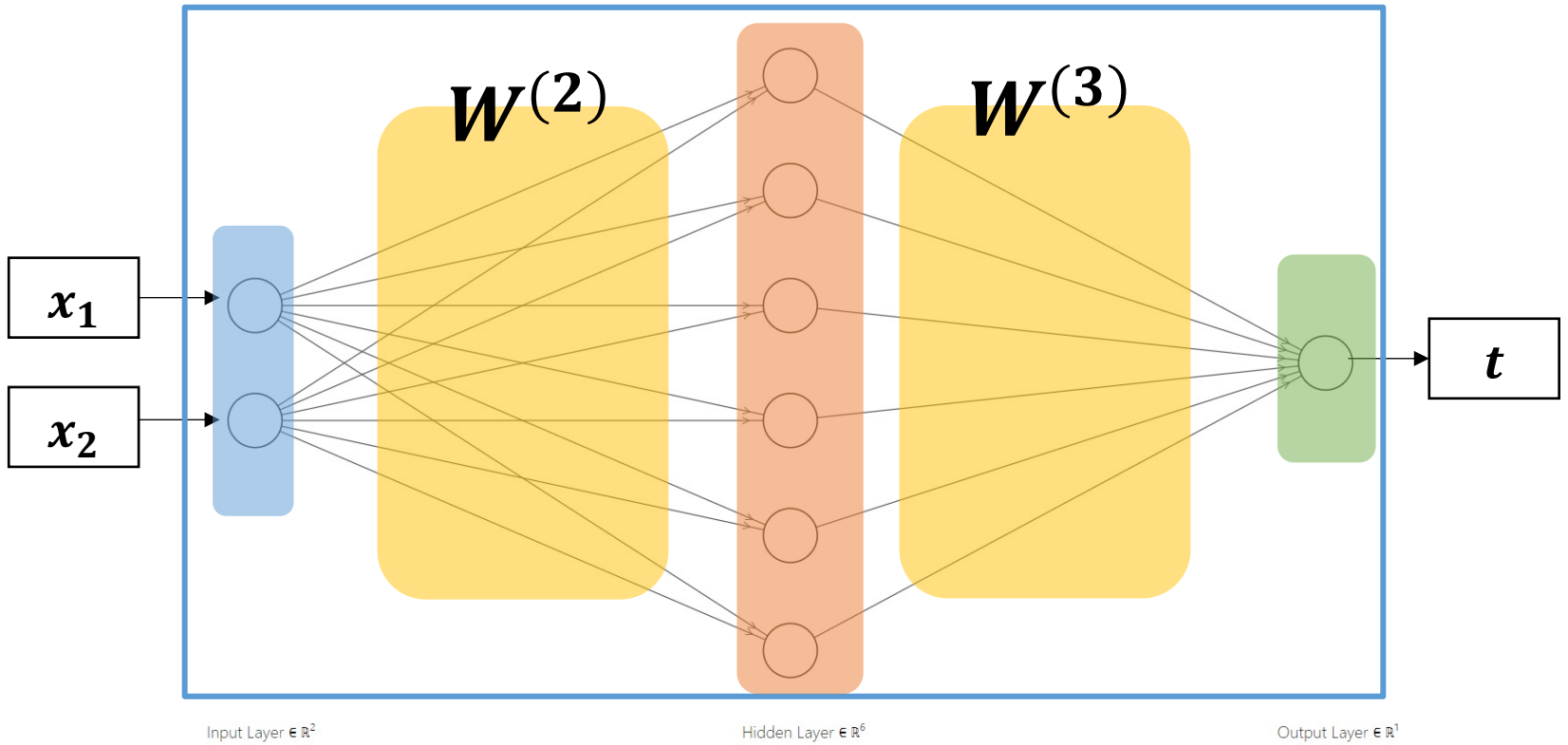
XOR 문제 - 딥러닝으로 해결

NAND, OR, AND같은 조합을 이용하지 않고 딥러닝 구조로 XOR 구현

- 딥러닝의 은닉층이 NAND, OR, AND 조합의 역할을 해주면 가능!

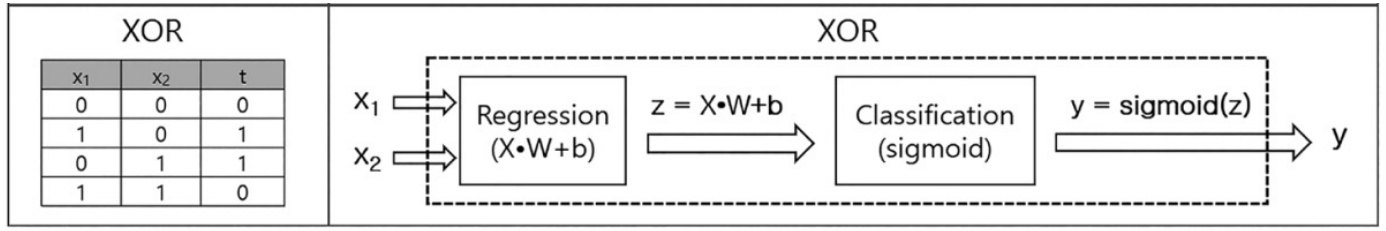
XOR

X_1	X_2	t
0	0	0
0	1	1
1	0	1
1	1	0

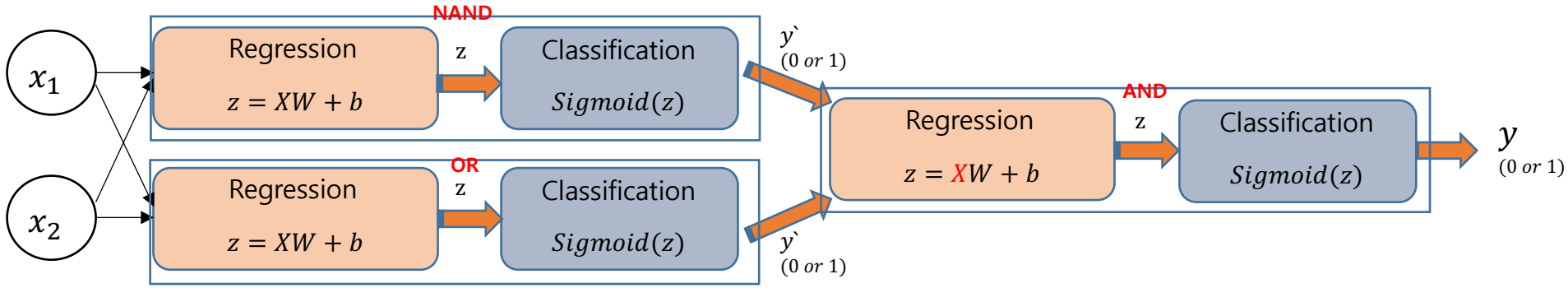


XOR 문제 - 방법 비교

1. Logistic Regression으로 구현



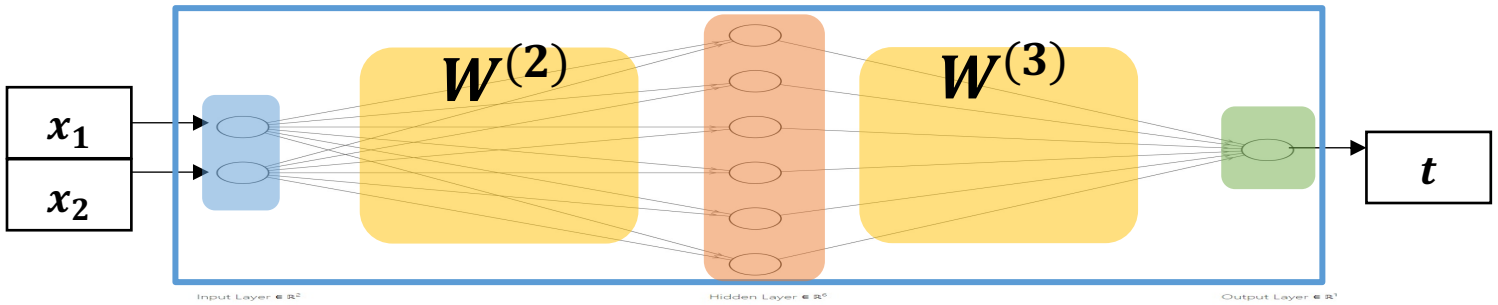
2. NAND, OR, AND 조합을 이용해 Logistic Regression으로 구현



3. 딥러닝으로 구현

XOR

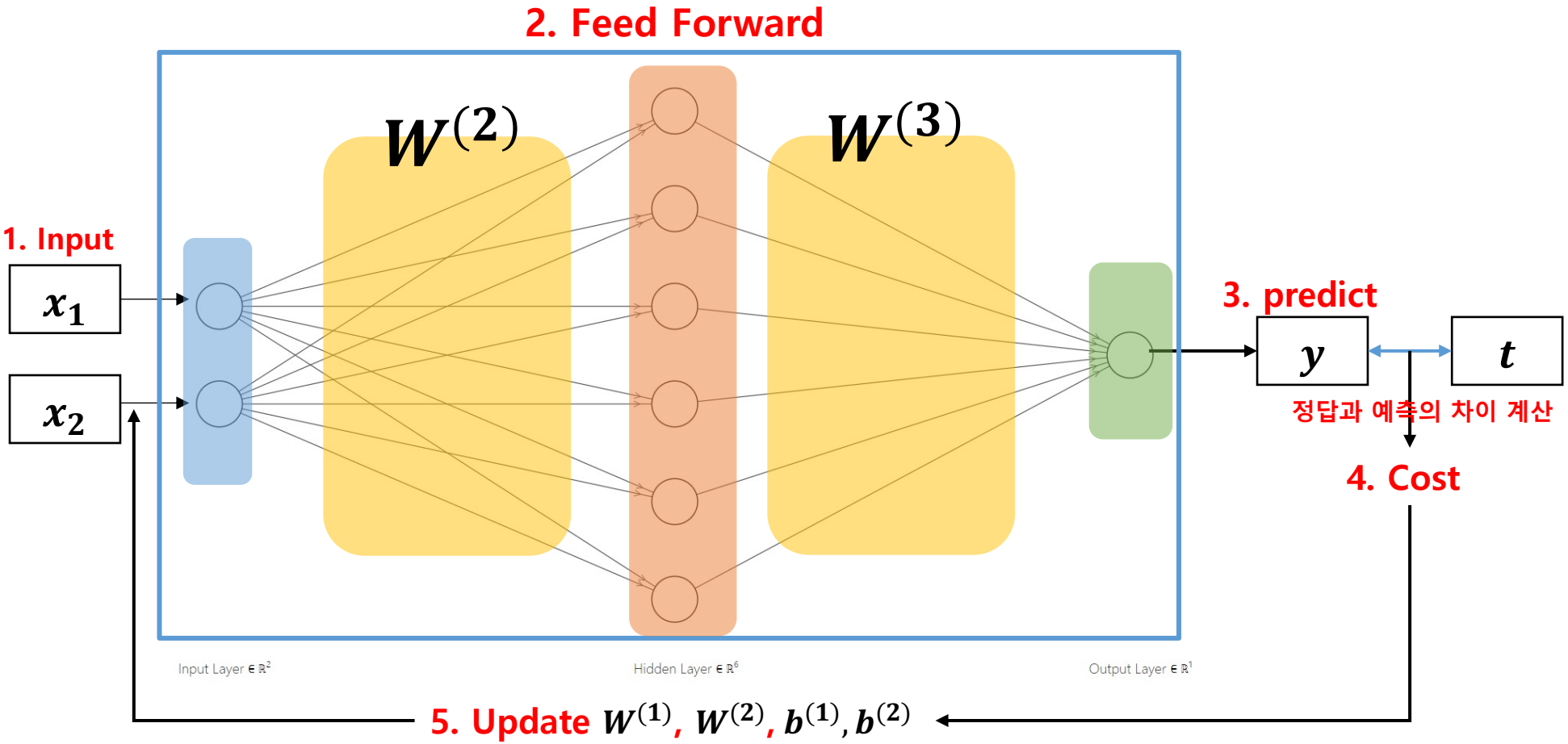
X_1	X_2	t
0	0	0
0	1	1
1	0	1
1	1	0



XOR 문제 - 딥러닝 아키텍처

XOR

X_1	X_2	t
0	0	0
0	1	1
1	0	1
1	1	0



- $Cost(W,b) = \frac{1}{m}\sum_{i=1}^{m} -ylog\big(h_\theta(x)\big) - (1-y)\log(1-h_\theta(x))$

$$\underset{W,b}{\text{minimize}}\, cost(W,b)$$

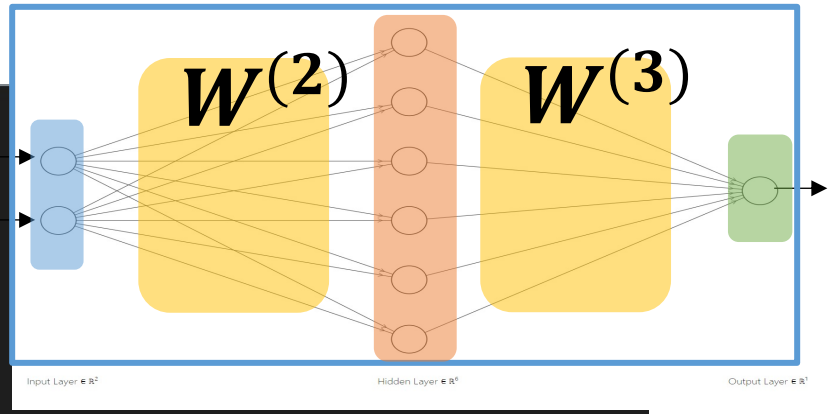
- $W^{(2)} = W^{(2)} - \alpha\frac{\partial Cost(W,b)}{\partial W^{(2)}}$
- $W^{(3)} = W^{(3)} - \alpha\frac{\partial Cost(W,b)}{\partial W^{(3)}}$
- $b^{(2)} = b^{(2)} - \alpha\frac{\partial Cost(W,b)}{\partial b^{(2)}}$
- $b^{(3)} = b^{(3)} - \alpha\frac{\partial Cost(W,b)}{\partial b^{(3)}}$

XOR 문제 - 딥러닝으로 해결 Code : define Sigmoid, derivative

```python
import numpy as np

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

# 수치미분 함수
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

       x[idx] = tmp_val
        it.iternext()

    return grad
```

XOR 문제 - 딥러닝으로 해결 Code

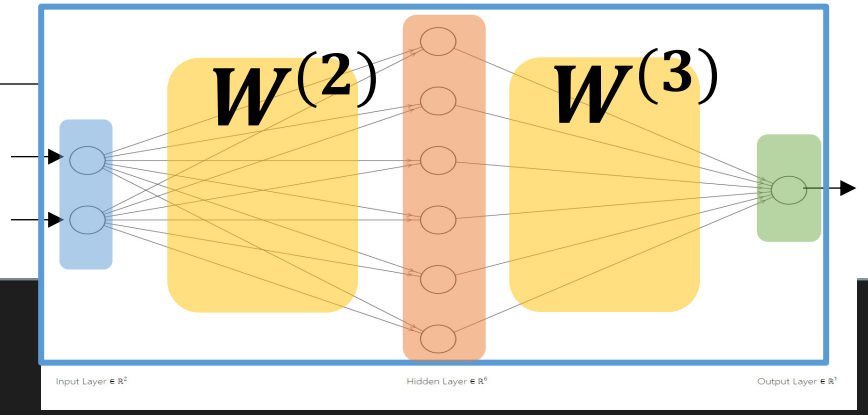
LogicGate calss




```python
class LogicGate:

    def __init__(self, gate_name, xdata, tdata):

        self.name = gate_name

        # 입력 데이터, 정답 데이터 초기화
        self.__xdata = xdata.reshape(4,2)  # 4개의 입력데이터 x1, x2 에 대하여 batch 처리 행렬
        self.__tdata = tdata.reshape(4,1)  # 4개의 입력데이터 x1, x2 에 대한 각각의 계산 값 행렬

        # 2층 hidden layer unit : 6개 가정,  가중치 W2, 바이어스 b2 초기화
        self.__W2 = np.random.rand(2,6)  # weight, 2 X 6 matrix
        self.__b2 = np.random.rand(6)

        # 3층 output layer unit : 1 개 , 가중치 W3, 바이어스 b3 초기화
        self.__W3 = np.random.rand(6,1)
        self.__b3 = np.random.rand(1)

        # 학습률 learning rate 초기화
        self.__learning_rate = 1e-2
        print(self.name + " object is created")
```

XOR 문제 - 딥러닝으로 해결 Code

LogicGate calss – **feed_forward**




```python
def feed_forward(self):        # feed forward 를 통하여 손실함수(cross-entropy) 값 계산

    delta = 1e-7    # log 무한대 발산 방지

    z2 = np.dot(self.__xdata, self.__W2) + self.__b2  # 은닉층의 선형회귀 값
    a2 = sigmoid(z2)                                  # 은닉층의 출력

    z3 = np.dot(a2, self.__W3) + self.__b3            # 출력층의 선형회귀 값
    y = a3 = sigmoid(z3)                              # 출력층의 출력

    # cross-entropy
    return  -np.sum( self.__tdata*np.log(y + delta) + (1-self.__tdata)*np.log((1 - y)+delta ) )
```

- $Cost(W,b) = \frac{1}{m}\sum_{i=1}^{m} -ylog\big(h_\theta(x)\big) - (1-y)\log(1-h_\theta(x))$

XOR 문제 - 딥러닝으로 해결 Code

LogicGate calss – **loss_val**

```
    def loss_val(self):          # 외부 출력을 위한 손실함수(cross-entropy) 값 계산

        delta = 1e-7    # log 무한대 발산 방지

        z2 = np.dot(self.__xdata, self.__W2) + self.__b2  # 은닉층의 선형회귀 값
        a2 = sigmoid(z2)                                  # 은닉층의 출력

        z3 = np.dot(a2, self.__W3) + self.__b3            # 출력층의 선형회귀 값
        y = a3 = sigmoid(z3)                              # 출력층의 출력

        # cross-entropy
        return  -np.sum( self.__tdata*np.log(y + delta) + (1-self.__tdata)*np.log((1 - y)+delta ) )
```

XOR 문제 - 딥러닝으로 해결 Code

LogicGate calss - **train**

```
    # 수치미분을 이용하여 손실함수가 최소가 될때 까지 학습하는 함수
    def train(self):

        f = lambda x : self.feed_forward()

        print("Initial loss value = ", self.loss_val())

        for step in  range(10001):

            self.__W2 -= self.__learning_rate * numerical_derivative(f, self.__W2)

            self.__b2 -= self.__learning_rate * numerical_derivative(f, self.__b2)

            self.__W3 -= self.__learning_rate * numerical_derivative(f, self.__W3)

            self.__b3 -= self.__learning_rate * numerical_derivative(f, self.__b3)

            if (step % 400 == 0):
                print("step = ", step, "  , loss value = ", self.loss_val())
```

XOR 문제 - 딥러닝으로 해결 Code

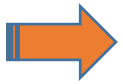
LogicGate calss - **predict**

```
    # query, 즉 미래 값 예측 함수
    def predict(self, xdata):

        z2 = np.dot(xdata, self.__W2) + self.__b2          # 은닉층의 선형회귀 값
        a2 = sigmoid(z2)                                   # 은닉층의 출력

        z3 = np.dot(a2, self.__W3) + self.__b3             # 출력층의 선형회귀 값
        y = a3 = sigmoid(z3)                               # 출력층의 출력

        if y > 0.5:
            result = 1  # True
        else:
            result = 0  # False

        return y, result
```

XOR 문제 - 딥러닝으로 해결 Code

train

```python
# XOR Gate 객체 생성, train

xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
tdata = np.array([0, 1, 1, 0])


xor_obj = LogicGate("XOR", xdata, tdata)

xor_obj.train()
```

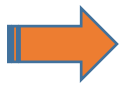


```
XOR object is created
Initial loss value =  4.735661084753321
step =  0    , loss value =  4.634876112146108
step =  400    , loss value =  2.775446295400008
step =  800    , loss value =  2.773263452948909
step =  1200    , loss value =  2.770917880220277
step =  1600    , loss value =  2.768128698254033
step =  2000    , loss value =  2.7645063286566858
step =  2400    , loss value =  2.7594693342625365
step =  2800    , loss value =  2.7521284652590445
step =  3200    , loss value =  2.741147600024205
step =  3600    , loss value =  2.7246269916546204
step =  4000    , loss value =  2.7000762411013386
step =  4400    , loss value =  2.6644912889238883
step =  4800    , loss value =  2.6145288972361405
step =  5200    , loss value =  2.5470986640337303
step =  5600    , loss value =  2.460951291588692
step =  6000    , loss value =  2.358547548152943
step =  6400    , loss value =  2.2456247222985573
step =  6800    , loss value =  2.127871832523778
step =  7200    , loss value =  2.007800434630348
step =  7600    , loss value =  1.8839207937411317
step =  8000    , loss value =  1.7518375989798356
step =  8400    , loss value =  1.6071395629609737
step =  8800    , loss value =  1.44991085325998
step =  9200    , loss value =  1.2871190506030352
step =  9600    , loss value =  1.1292858728978399
step =  10000    , loss value =  0.9849821204749392
```

XOR 문제 - 딥러닝으로 해결 Code

predict

```
1 test_data = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])
2
3 for data in test_data:
4     print(xor_obj.predict(data)
```



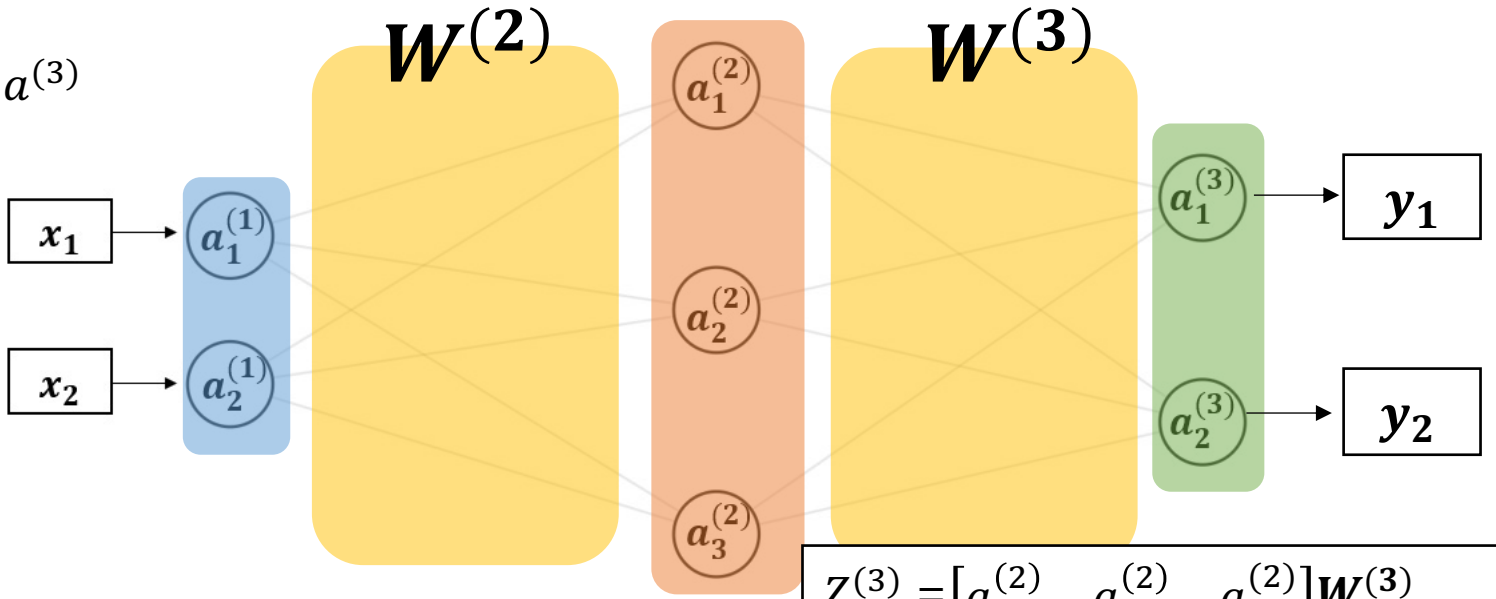
```
(array([0.0833791]), 0)
(array([0.81125073]), 1)
(array([0.7467601]), 1)
(array([0.32748571]), 0)
```


감사합니다.

Appendix

DNN 뜯어보기 – 선형 회귀 계산 식 (activation : 항등 함수, 편향이 없을경우)

항등함수
→ $Z^{(2)} = a^{(2)}, Z^{(3)} = a^{(3)}$



Input

x_1, x_2

$$Z^{(2)} = \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} W^{(2)}$$
$$= \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \end{bmatrix}$$
$$= \begin{bmatrix} z_1^{(2)} & z_2^{(2)} & z_3^{(2)} \end{bmatrix}$$
$$= \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix}$$

$$Z^{(3)} = \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix} W^{(3)}$$
$$= \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix} \begin{bmatrix} w_{11}^{(3)} & w_{21}^{(3)} \\ w_{12}^{(3)} & w_{22}^{(3)} \\ w_{13}^{(3)} & w_{23}^{(3)} \end{bmatrix}$$
$$= \begin{bmatrix} z_1^{(3)} & z_2^{(3)} \end{bmatrix}$$
$$= \begin{bmatrix} a_1^{(3)} & a_2^{(3)} \end{bmatrix}$$

Output

y_1, y_2

편향이란?

- 하나의 뉴런으로 입력된 모든 값을 다 더한 다음 이 값에 더해주는 상수

→최종적으로 출력되는 값을 조절

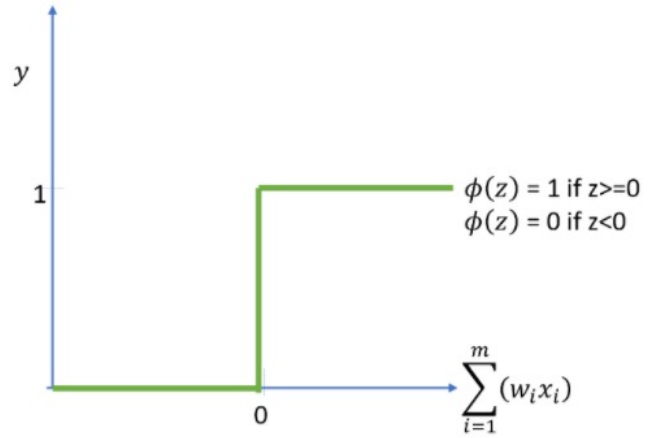
→과적합(overfitting)을 방지하는데 중요

$$t = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq \theta) \\ 1 & (w_1x_1 + w_2x_2 > \theta) \end{cases}$$ Perceptron 식에서 θ를 이항하면?

$$t = \begin{cases} 0 & (w_1x_1 + w_2x_2 - \theta \leq 0) \\ 1 & (w_1x_1 + w_2x_2 - \theta > 0) \end{cases}$$ bias == $-\theta$임을 알 수 있다!

편향이란?

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



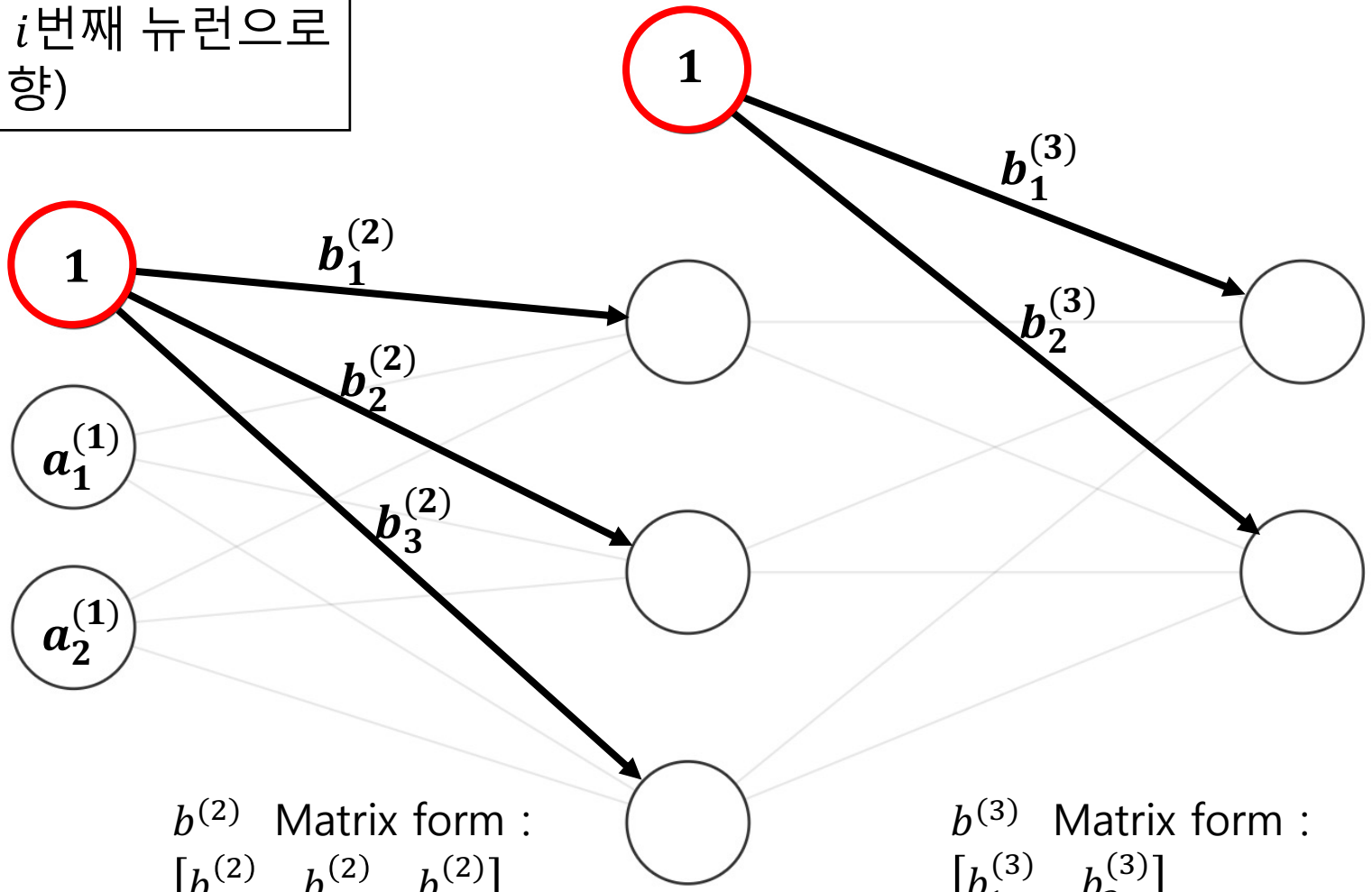
위 식과 계단함수(Step function)을 생각해보자.

신호에 특정 상수(θ)를 빼서 신호의 세기를 증가/감소 시킨다.

→ 신호에 편향을 만든다.

DNN 뜯어보기 – Activation, bias를 포함하는 신경망 구조

$b_i^{(n)}$: n층의 i번째 뉴런으로 가는 bias(편향)

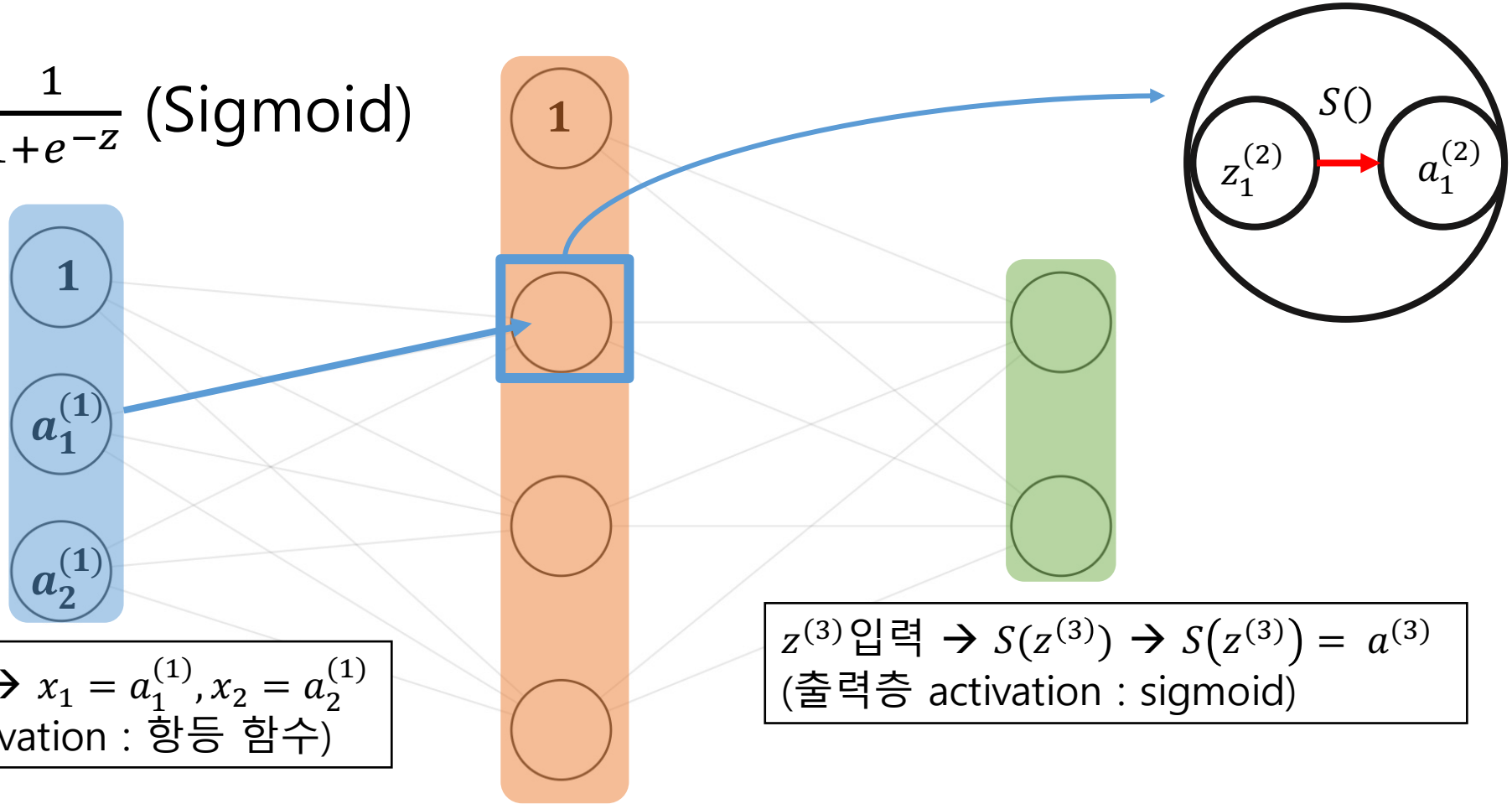


$b^{(2)}$ Matrix form : $[b_1^{(2)} \quad b_2^{(2)} \quad b_3^{(2)}]$

$b^{(3)}$ Matrix form : $[b_1^{(3)} \quad b_2^{(3)}]$

DNN 뜯어보기 – Activation, bias를 포함하는 신경망 구조

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

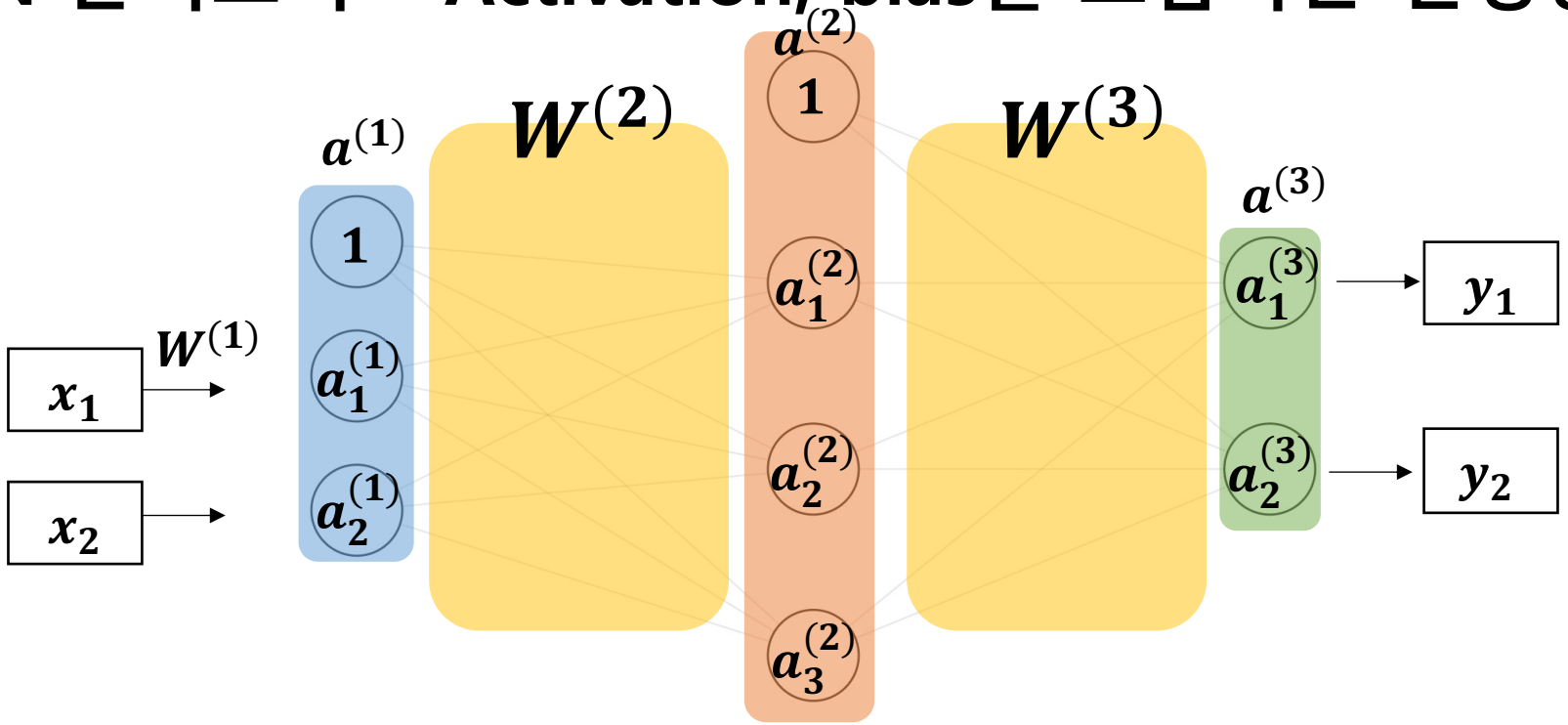


x_1, x_2 입력 → $x_1 = a_1^{(1)}, x_2 = a_2^{(1)}$
(입력층 activation : 항등 함수)

$z^{(3)}$입력 → $S(z^{(3)})$ → $S(z^{(3)}) = a^{(3)}$
(출력층 activation : sigmoid)

$a^{(2)}$입력 → $S(z^{(2)})$ → $a^{(2)} = S(z^{(2)})$
(은닉층 activation : sigmoid)

FFNN 뜯어보기 – Activation, bias를 포함하는 신경망 구조



Input

x_1, x_2

$$z^{(2)} = \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} W^{(2)} + b^{(2)}$$

$$= \begin{bmatrix} a_1^{(1)} & a_2^{(1)} \end{bmatrix} \begin{bmatrix} w_{11}^{(2)} & w_{21}^{(2)} & w_{31}^{(2)} \\ w_{12}^{(2)} & w_{22}^{(2)} & w_{32}^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(2)} & b_2^{(2)} & b_3^{(2)} \end{bmatrix}$$

$$= \begin{bmatrix} z_1^{(2)} & z_2^{(2)} & z_3^{(2)} \end{bmatrix}$$

$$a^{(2)} = S\left(z^{(2)}\right)$$

$$= \begin{bmatrix} S(z_1^{(2)}) & S(z_2^{(2)}) & S(z_3^{(2)}) \end{bmatrix}$$

$$= \begin{bmatrix} a_1^{(2)} & a_2^{(2)} & a_3^{(2)} \end{bmatrix}$$