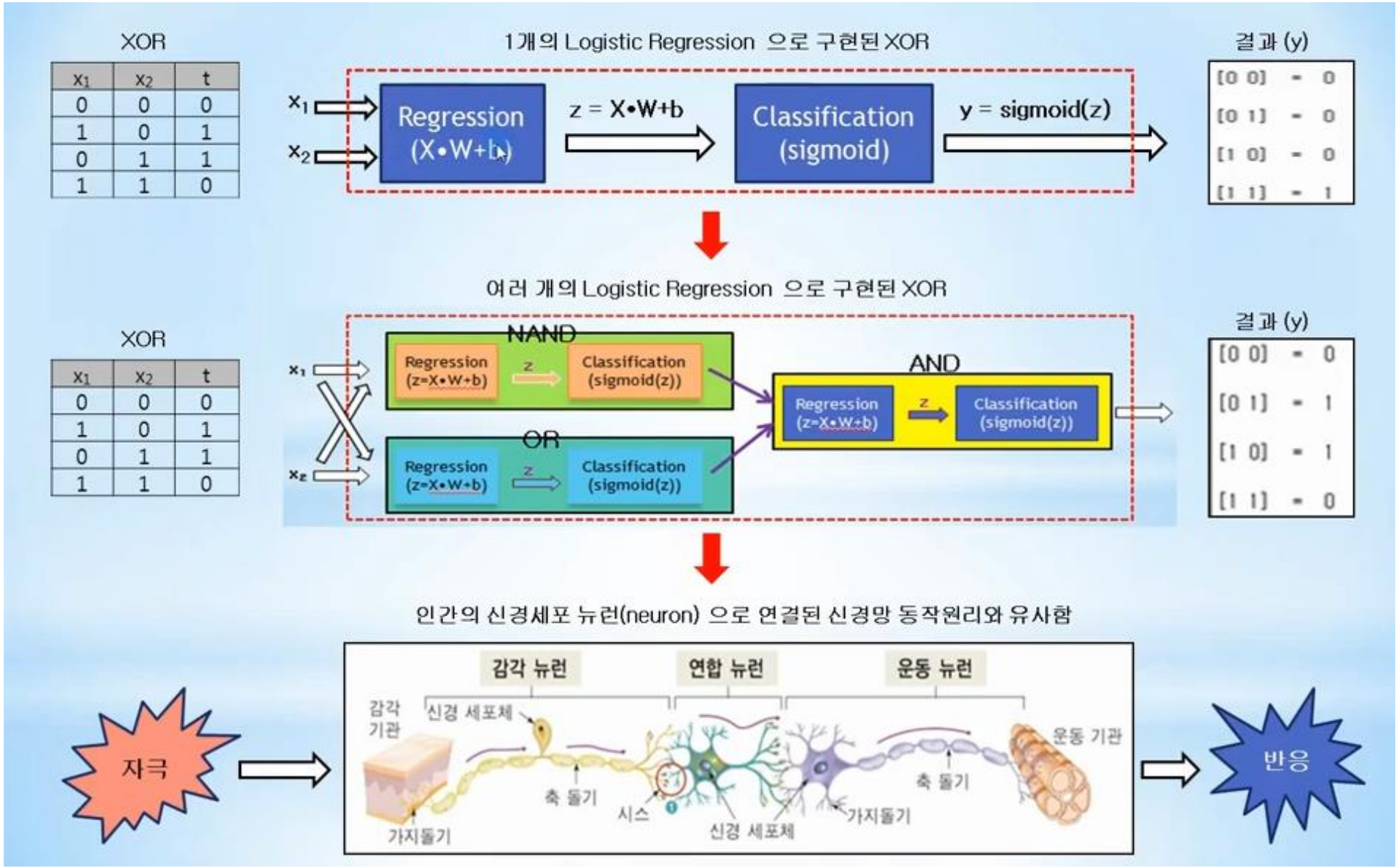
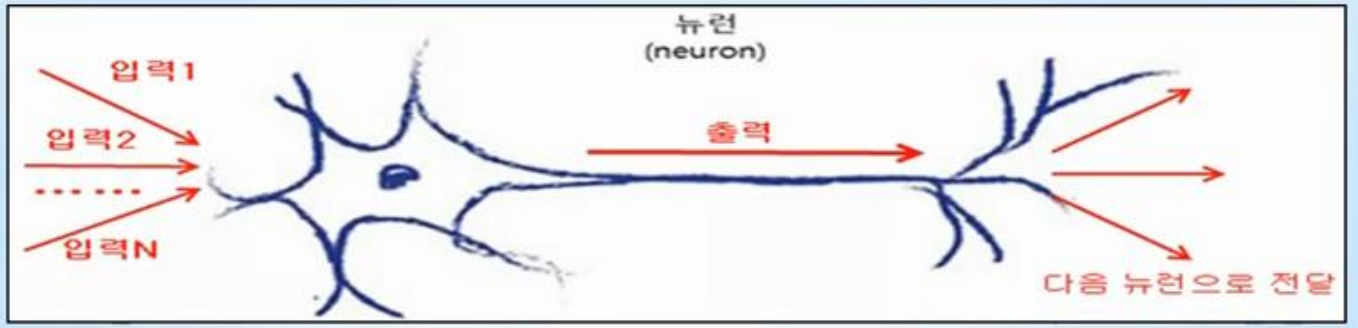
5. 딥러닝

박경미

XOR 문제


XOR
x1 x2 t
0 0 0
1 0 1
0 1 1
1 1 0
1개의 Logistic Regression 으로 구현된 XOR
x1
x2
Regression (X•W+b)
z = X•W+b
Classification (sigmoid)
y = sigmoid(z)
결과 (y)
[0 0] - 0
[0 1] - 0
[1 0] - 0
[1 1] - 1
여러 개의 Logistic Regression 으로 구현된 XOR
XOR
x1 x2 t
0 0 0
1 0 1
0 1 1
1 1 0
NAND
Regression (z=X•W+b)
z
Classification (sigmoid(z))
OR
Regression (z=X•W+b)
z
Classification (sigmoid(z))
AND
Regression (z=X•W+b)
z
Classification (sigmoid(z))
결과 (y)
[0 0] - 0
[0 1] - 1
[1 0] - 1
[1 1] - 0
인간의 신경세포 뉴런(neuron) 으로 연결된 신경망 동작원리와 유사함
자극
감각 뉴런
연합 뉴런
운동 뉴런
감각 기관
신경 세포체
축 돌기
시스
신경 세포체
가지돌기
축 돌기
운동 기관
가지돌기
반응

신경망(Neural Network)-concept(1)

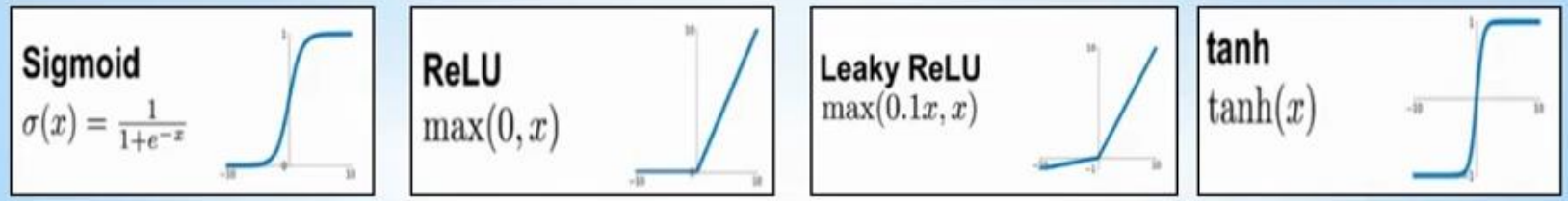


⇒ 신경 세포 뉴런(neuron)은 이전 뉴런으로부터 입력신호를 받아 또 다른 신호를 발생시킴

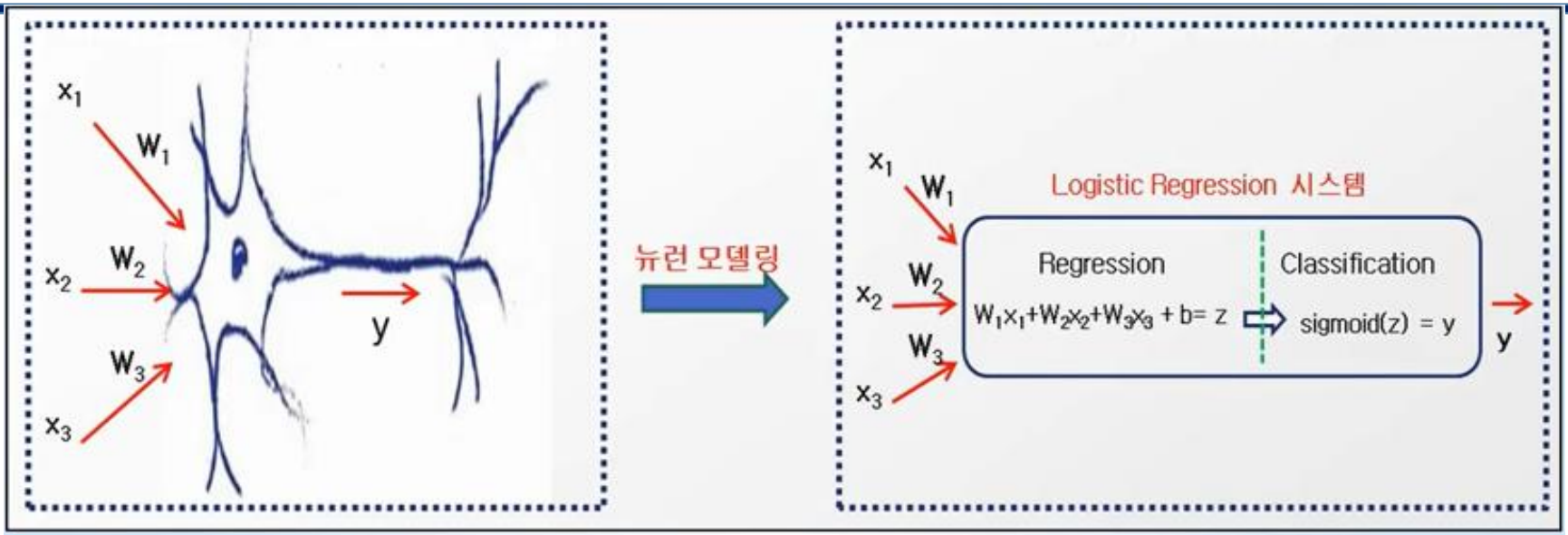
⇒ 그러나 입력에 비례해서 출력을 내는 형태 (y=WX) 가 아니라, 입력 값들의 모든 합이 어느 임계점(threshold)에 도달해야만 출력 신호를 발생시킴.

⇒ 이처럼 입력신호를 받아 특정 값의 임계점을 넘어서는 경우에, 출력을 생성해주는 함수를 활성화 함수(activation function)라고 하는데, 지금까지 사용해왔던 Logistic Regression 시스템의 sigmoid 함수가 대표적인 활성화 함수*임. 즉, sigmoid 에서의 임계점은 0.5 로서, 입력값 합이 0.5 보다 크다면 1을 출력으로 내보내고, 0.5 보다 값이 작으면 출력을 내보내지 않는다고 볼 수 있음 (0 은 출력이 없는 상태)

※ 활성화함수 : sigmoid, ReLU, Leaky ReLU, hyperbolic tangent(tanh) …



신경망(Neural Network)-concept(1)



➢ 신경 세포인 뉴런 동작원리를 머신러닝에 적용하기 위해서는,

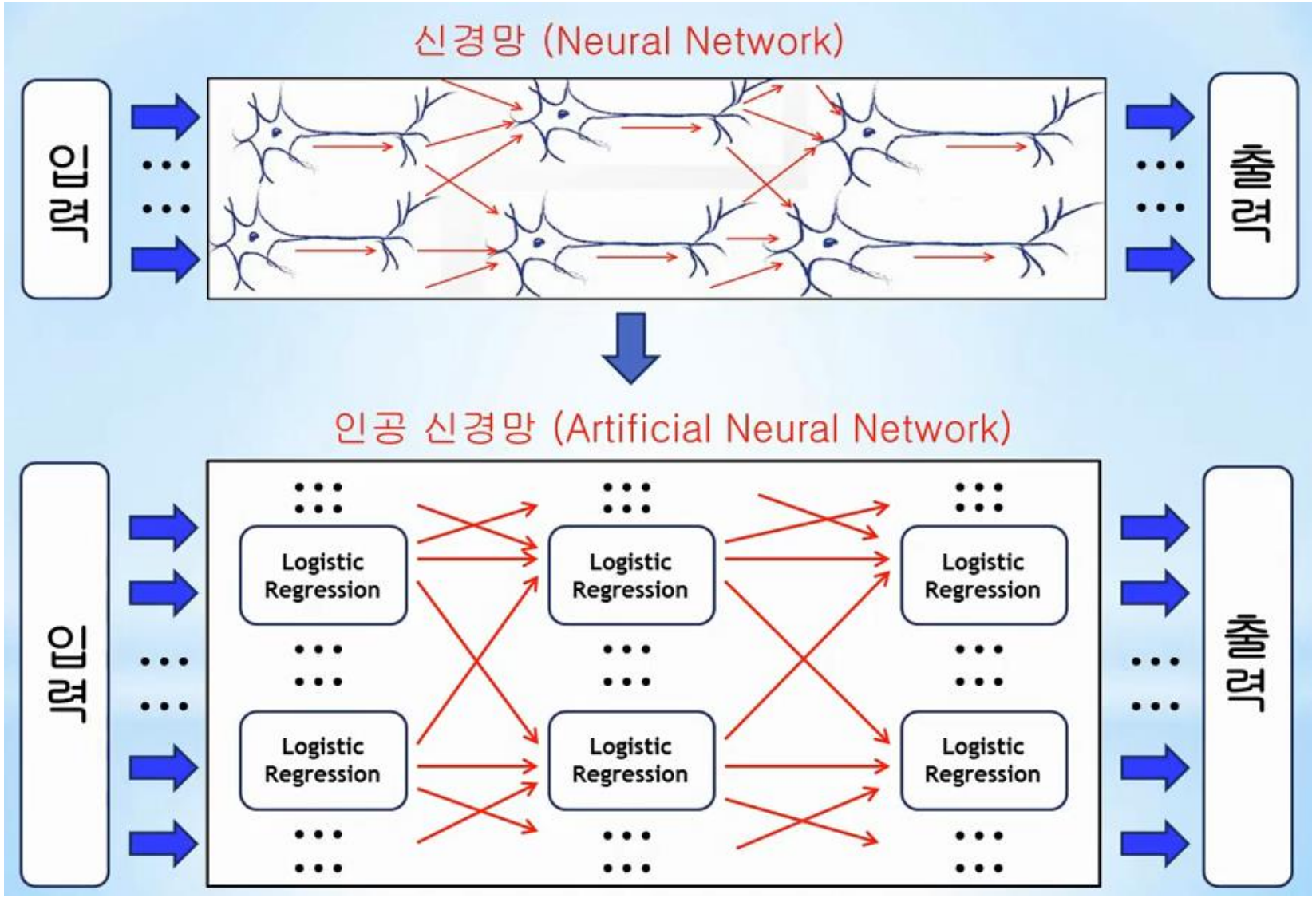
① 입력 신호와 가중치를 곱하고 적당한 바이어스를 더한 후 (Linear Regression)

② 그 값을 활성화 함수(sigmoid) 입력으로 전달 (Classification)해서 sigmoid 함수 임계점 0.5 를 넘으면 1을, 그렇지 않으면 0 을 다음의 뉴런으로 전달해주는

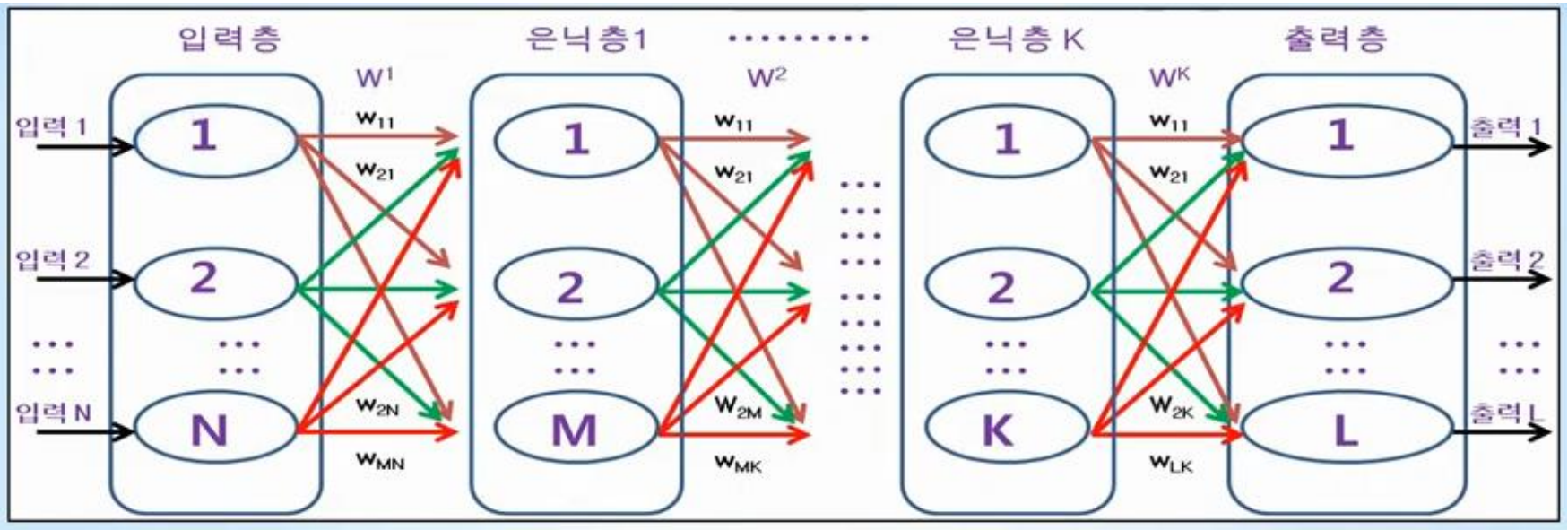
⬇

multi-variable Logistic Regression 시스템 구축

신경망(Neural Network)-architecture



딥러닝(Deep Learnning)-concept



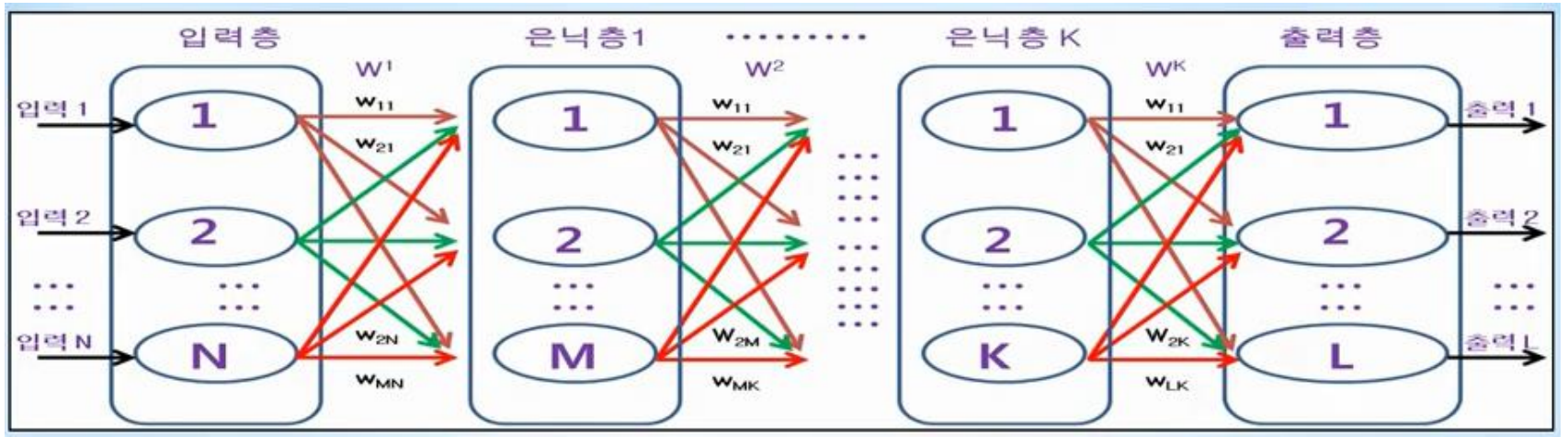
➢ 딥러닝 (Deep Learning)

노드(node) : 1개의 logistic regression을 나타냄

- 노드가 서로 연결되어 있는 신경망 구조를 바탕으로 입력층(Input Layer), 1개 이상의 은닉층(Hidden Layer), 출력층(Output Layer)을 구축하고, 출력층(Output Layer)에서의 오차를 기반으로 각 노드(뉴런)의 가중치(Weight)를 학습하는 머신러닝 한 분야

- [참고] 딥러닝 구조에서 1개 이상의 은닉층(hidden layer)을 이용하여 학습시키면 정확도가 높은 결과를 얻을 수 있다고 알려져 있음. 즉 은닉층을 깊게(Deep) 할수록 정확도가 높아진다고 해서 딥(Deep)러닝이라는 용어가 사용 되고 있음

딥러닝(Deep Learnning)-concept



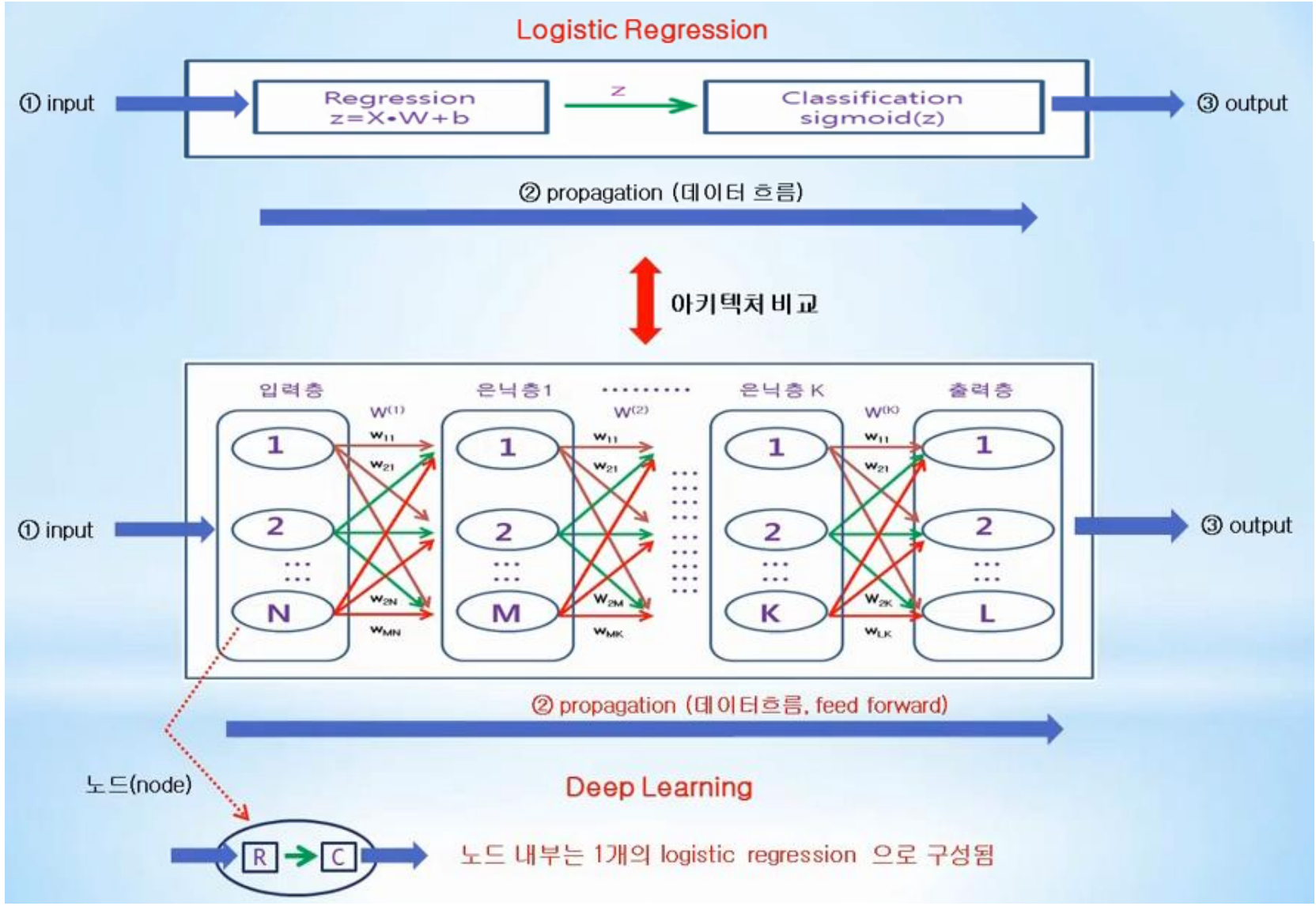
Propagation

가중치 w_{21} ⇒ 특정 계층의 노드 1에서 다음 계층의 노드 2로 전달되는 신호를 강화 또는 약화 시키는 가중치 (즉, 다음계층의 노드 번호가 먼저 나옴)

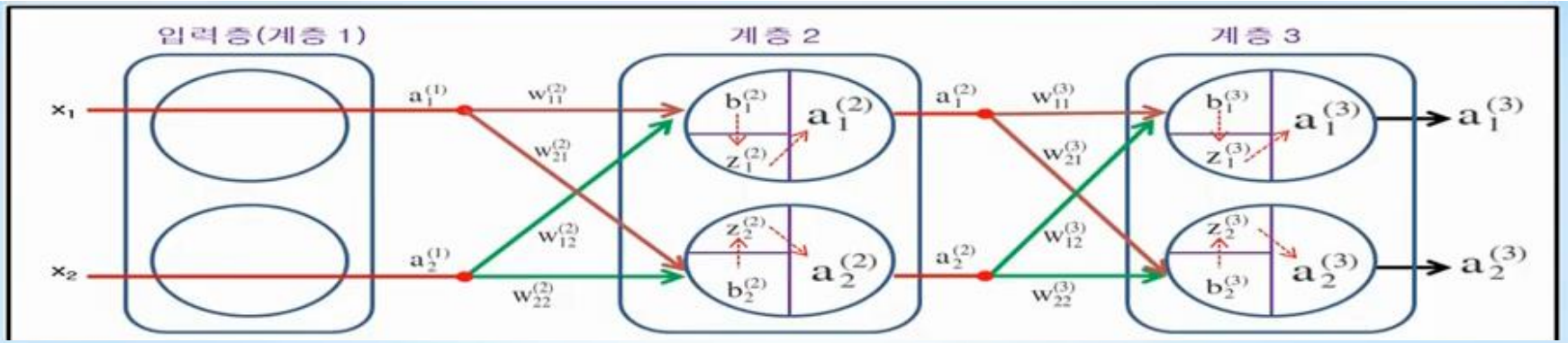
가중치 w_{2N} ⇒ 특정 계층의 노드 N에서 다음 계층의 노드 2로 전달되는 신호를 강화 또는 약화 시키는 가중치 (즉, 다음계층의 노드 번호가 먼저 나옴)

이러한 가중치 값들은, 층과 층사이의 모든 노드에 초기화 되어 있으며, 데이터가 입력층에서 출력층으로 전파(propagation) 될 때, 각 층에 있는 노드의 모든 가중치(w_{11}, w_{12}, … w_{MK}, w_{LK} 등)는 신호를 약화시키거나(낮은 가중치) 또는 신호를 강화(높은 가중치) 시키며, 최종적으로는 오차가 최소 값이 될 때 최적의 값을 가지게 됨

아키텍처 비교 –logistic regressoion vs. deep learining



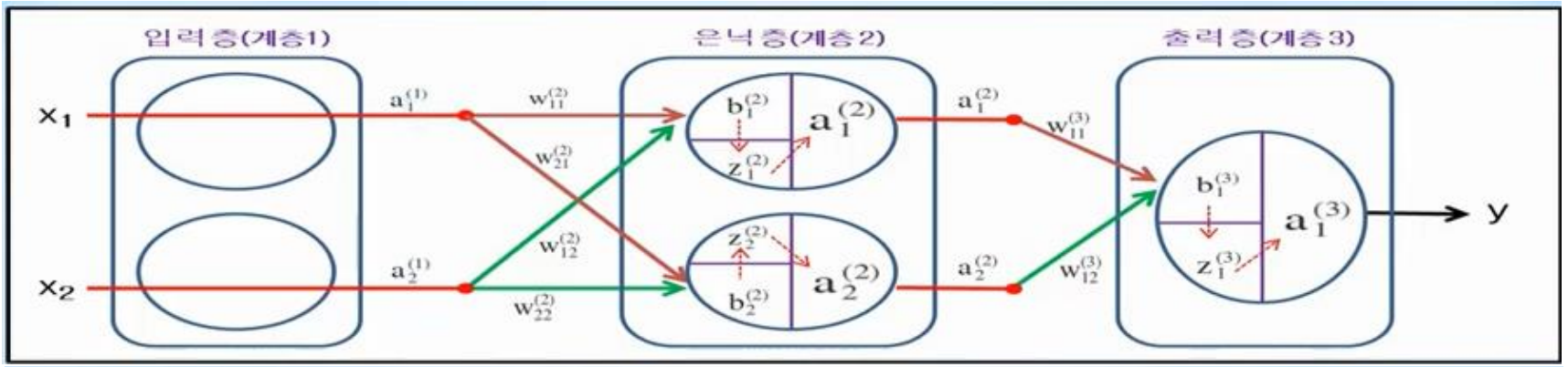
피드포워드(feed forward –표기법(notation)



- 계층간 가중치 표기법 (weight notation)
  - 가중치 $w_{21}^{(2)}$ ⇒ 계층 2의 노드에 적용되는 가중치로서, 1 계층의 노드 1에서 2계층의 노드 2로 전달되는 신호를 강화 또는 약화시키는 가중치 (즉, 가중치에서의 아래숫자는 다음계층의 노드 번호가 먼저 나옴)
- 노드의 바이어스 표기법 (bias notation)
  - 바이어스 $b_1^{(2)}$ ⇒ 계층 2에 있는 첫번째 노드 (node 1) 에 적용되는 바이어스
- 노드의 선형회귀 계산 값 표기법 (linear regression notation)
  - 선형회귀 계산 값 $z_2^{(2)}$ ⇒ 계층 2의 두번째 노드 (node 2) 선형회귀 계산 값 ($z_2^{(2)} = x_1 w_{21}^{(2)} + x_2 w_{22}^{(2)} + b_2^{(2)}$)
- 노드의 출력 표기법 (node output notation)
  - 노드의 출력 값 $a_2^{(2)}$ ⇒ 계층 2의 두번째 노드 (node 2) 출력값으로서, logistic regression 계산 값.

    활성화함수(activation function)로서 sigmoid를 사용한다면 $a_2^{(2)} = \mathrm{sigmoid}(z_2^{(2)})$

피드포워드(feed forwar)- 동작방식(1)



➢ 피드포워드 (feed forward)

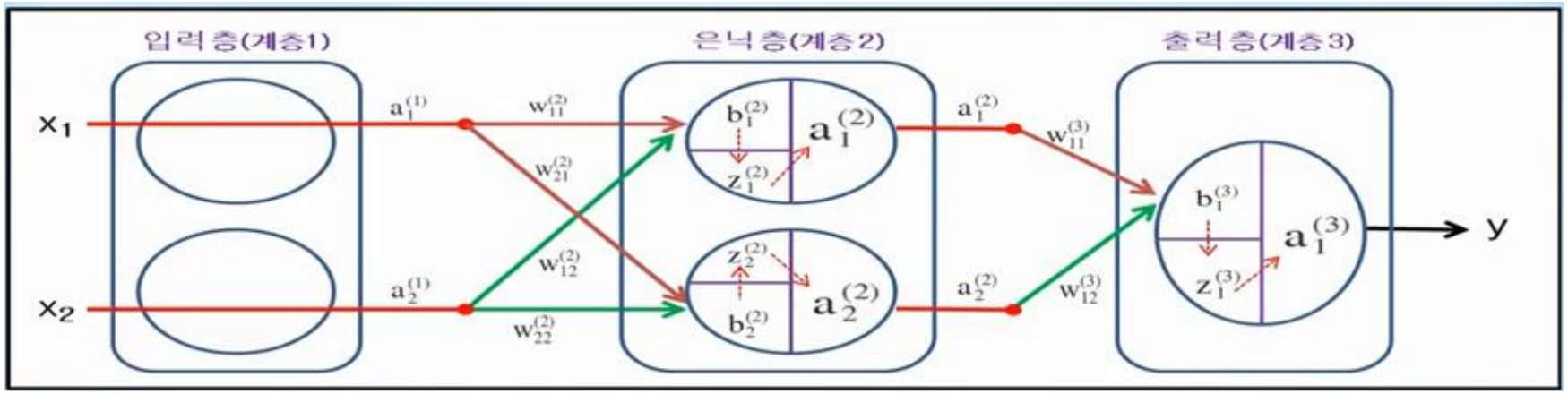
- 입력 층(input layer)으로 데이터가 입력되고, 1개 이상으로 구성되는 은닉 층(hidden layer)을 거쳐서 마지막에 있는 출력 층(output layer)으로 출력 값을 내보내는 과정
- 딥러닝 에서는 이전 층(previous layer)에서 나온 출력 값 ⇒ 층과 층 사이에 적용되는 가중치(weight) 영향을 받은 다음 ⇒ 다음 층(next layer)의 입력 값 으로 들어 가는 것을 의미함

➢ 입력 층(input layer) 출력

- 딥러닝 입력 층에서는 활성화 함수인 sigmoid 를 적용하지 않고, 입력 값 그대로 출력으로 내보내는 것이 관례화 되어 있음.

일반 식	행렬 식
$a_1^{(1)} = x_1 \quad a_2^{(1)} = x_2$	$\begin{pmatrix} a_1^{(1)} & a_2^{(1)} \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix}$

피드포워드(feed forwar)- 동작방식(2)



➢ 은닉 층(hidden layer) 선형회귀 값

입력층과 은닉층 사이의 가중치

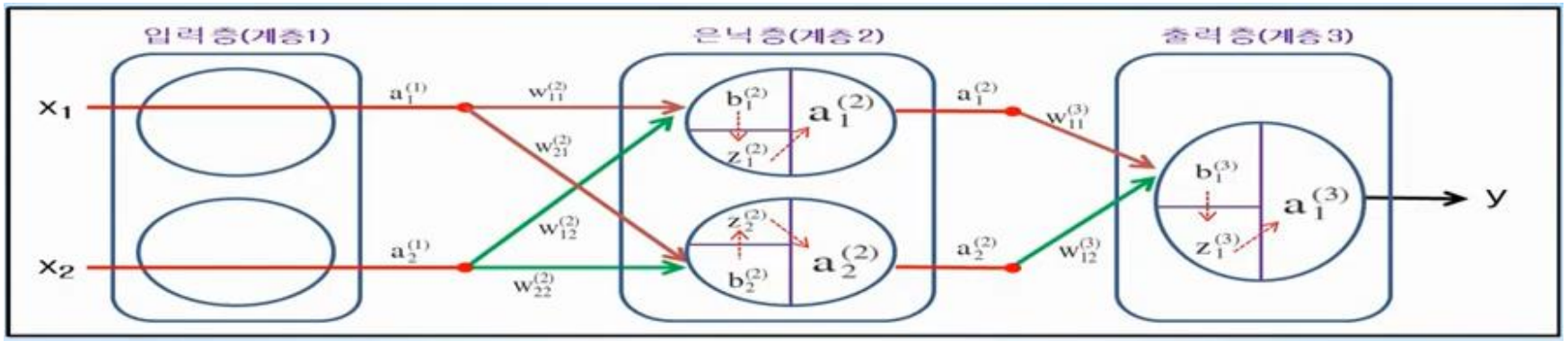
일반 식	행렬 식
$z_1^{(2)} = a_1^{(1)}w_{11}^{(2)} + a_2^{(1)}w_{12}^{(2)} + b_1^{(2)}$ $z_2^{(2)} = a_1^{(1)}w_{21}^{(2)} + a_2^{(1)}w_{22}^{(2)} + b_2^{(2)}$	은닉 층 선형회귀 값 / 입력 층 출력 / 은닉 층 바이어스 $\begin{pmatrix} z_1^{(2)} & z_2^{(2)} \end{pmatrix} = \begin{pmatrix} a_1^{(1)} & a_2^{(1)} \end{pmatrix} \bullet \begin{pmatrix} w_{11}^{(2)} & w_{12}^{(2)} \\ w_{21}^{(2)} & w_{22}^{(2)} \end{pmatrix} + \begin{pmatrix} b_1^{(2)} & b_2^{(2)} \end{pmatrix}$

➢ 은닉 층(hidden layer) 출력

$$a_1^{(2)} = \text{sigmoid}(z_1^{(2)}) \qquad a_2^{(2)} = \text{sigmoid}(z_2^{(2)})$$

- 활성화함수(activation function)는 sigmoid 이므로 출력 값의 범위는 0~1 사이의 값만을 가짐

피드포워드(feed forwar)- 동작방식(3)



➢ 출력 층(output layer) 선형회귀 값

은닉층과 출력층 사이의 가중치

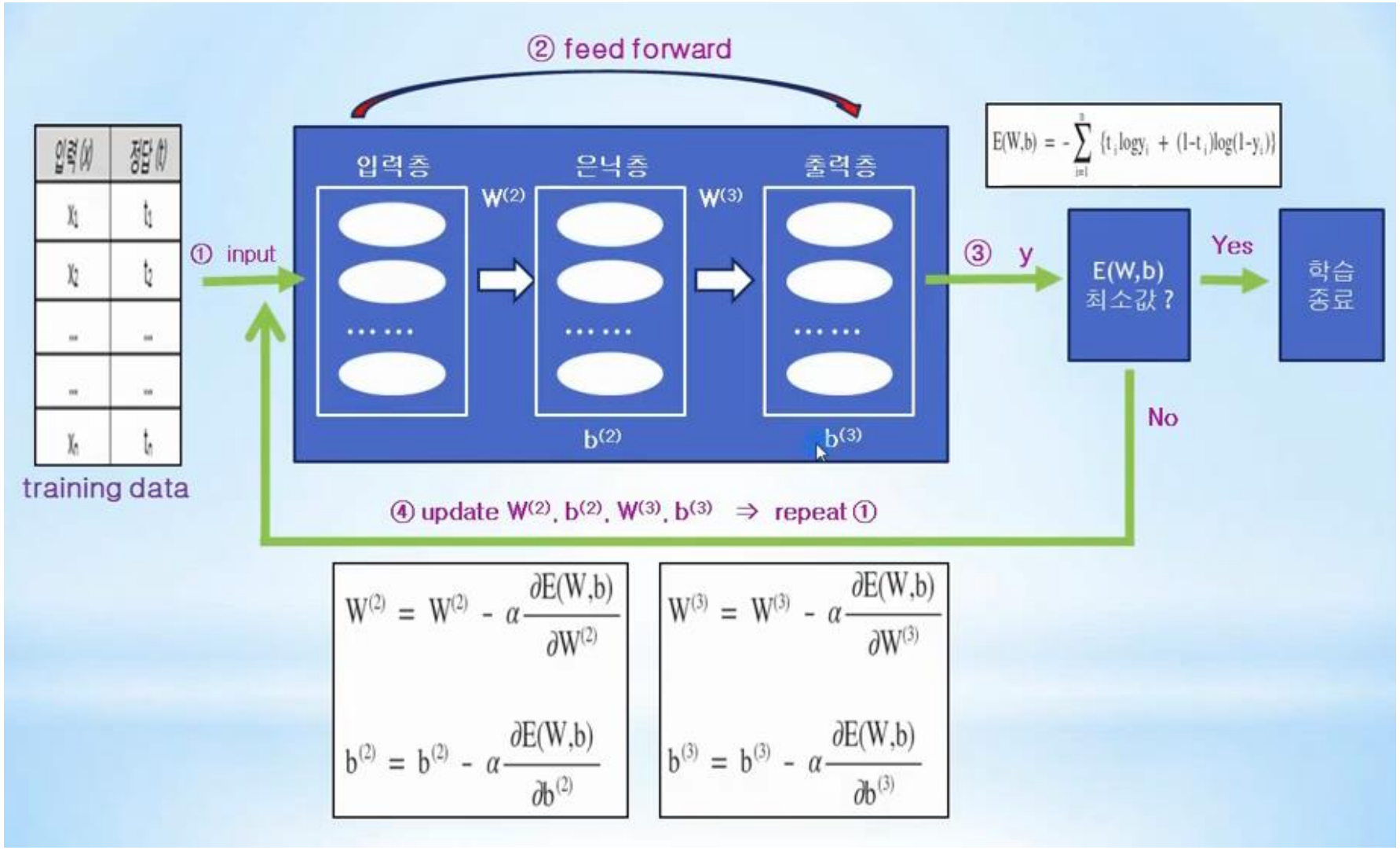
일반 식	행렬 식
$z_1^{(3)} = a_1^{(2)}w_{11}^{(3)} + a_2^{(2)}w_{12}^{(3)} + b_1^{(3)}$	출력 층 선형회귀 값 / 은닉 층 출력 / 출력 층 바이어스 $(z_1^{(3)}) = (a_1^{(2)} \; a_2^{(2)}) \cdot \begin{pmatrix} w_{11}^{(3)} \\ w_{12}^{(3)} \end{pmatrix} + (b_1^{(3)})$

➢ 출력 층(output layer) 출력

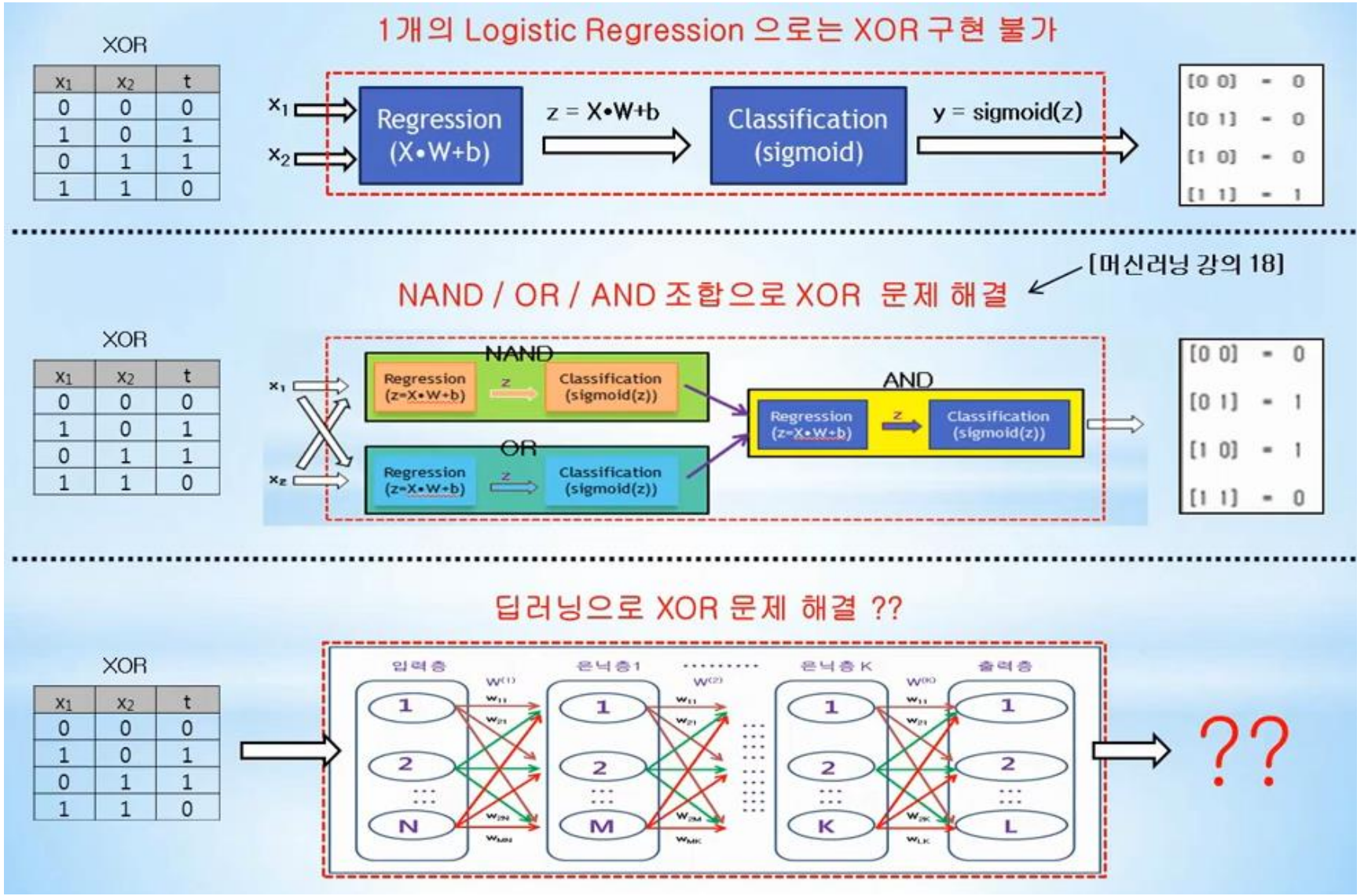
$$y = a_1^{(3)} = \text{sigmoid}(z_1^{(3)})$$

- 출력 값 a1 은, 입력 데이터에 대해 최종적으로 계산해야 하는 y 값 이며, 이러한 y 값과 정답 t 와의 차이인 오차(loss)를 통해서 가중치와 바이어스를 학습해야 하는 것을 의미함
- 즉, 딥러닝 에서는 출력 층(output layer)에서의 출력 값(y)과 정답(t)과의 차이를 이용하여, 오차가 최소가 되도록 각 층에 있는 가중치와 바이어스를 최적화 해야 함

딥러닝에서의 [W,b]계산 프로세스

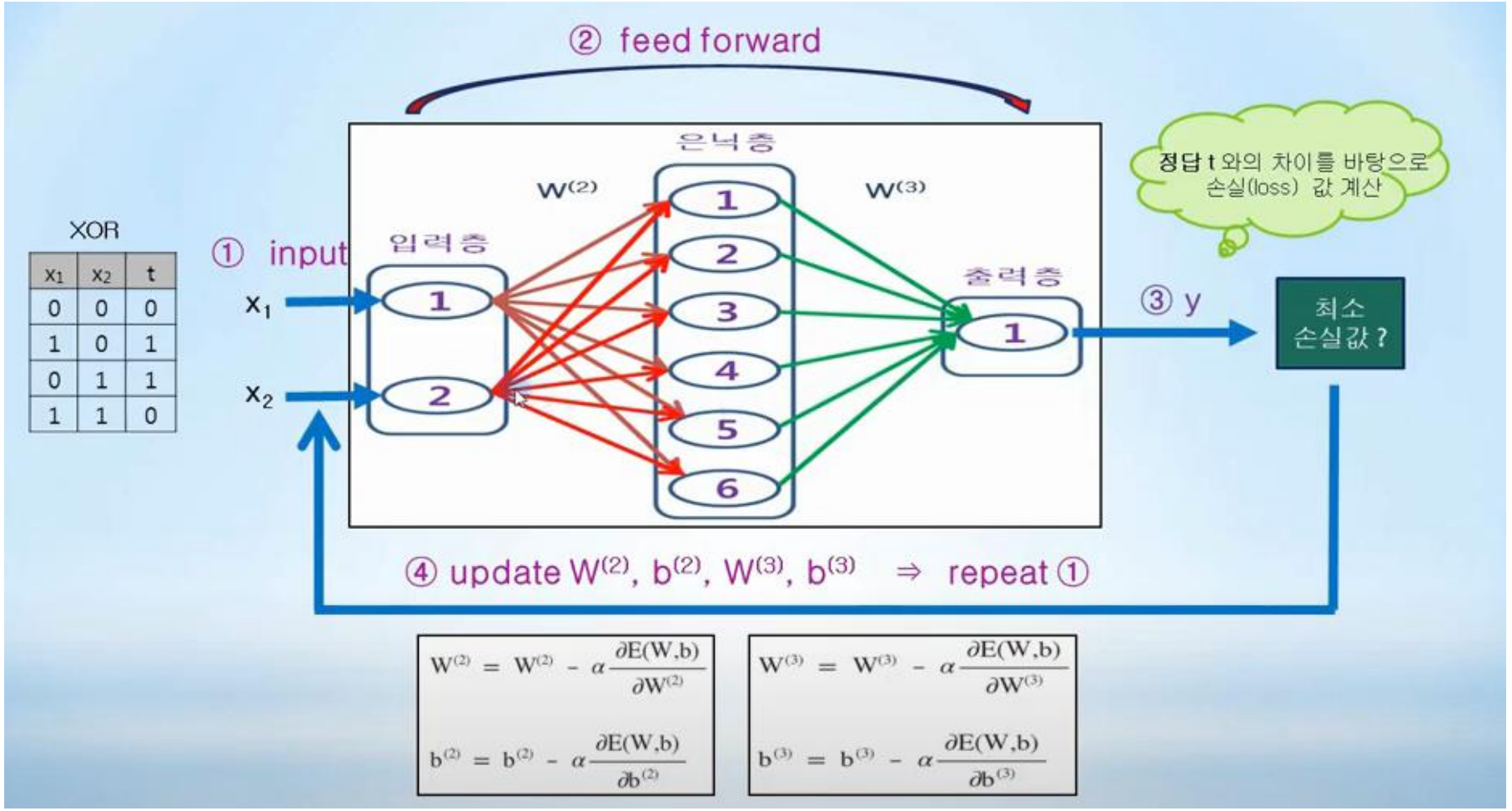


딥러닝으로 XOR문제 해결



XOR문제 – 딥러닝 아키텍처

- 딥러닝은 **1개 이상의 은닉층(hidden layer)**을 만들 수 있고, 각 **은닉 층(hidden layer)에 존재하는 노드(node) 개수** 또한 **임의의 개수**를 만들 수 있음. 은닉층과 노드 수가 많아 지면 학습 속도가 느려지므로 적절한 개수의 은닉층과 노드 수를 고려하여 구현하는 것이 필요함



LogicGate class – 딥러닝 버전

external function

```
def  sigmoid(x):                          # 0 또는 1 을 출력하기 위한 sigmoid 함수

def  numerical_derivative(f, x):    #  수치미분함수 (소스코드는 [머신러닝 강의 10] 참조)
```

LogicGate class

```
class  LogicGate:
    def    __init__(self, gate_name, xdata, tdata)   # 입력/정답 데이터/가중치/바이어스 초기화
    def    feed_forward(self)                        # feed forward 이용하여 손실함수 값 계산
    def    loss_val(self)                            # 손실함수 값 계산 (외부 출력을 위해 사용됨)
    def    train(self)                               # 수치미분을 이용하여 손실함수 최소값 찾는 method
    def    predict(self, xdata)                      # 미래 결과 값 예측 method
```

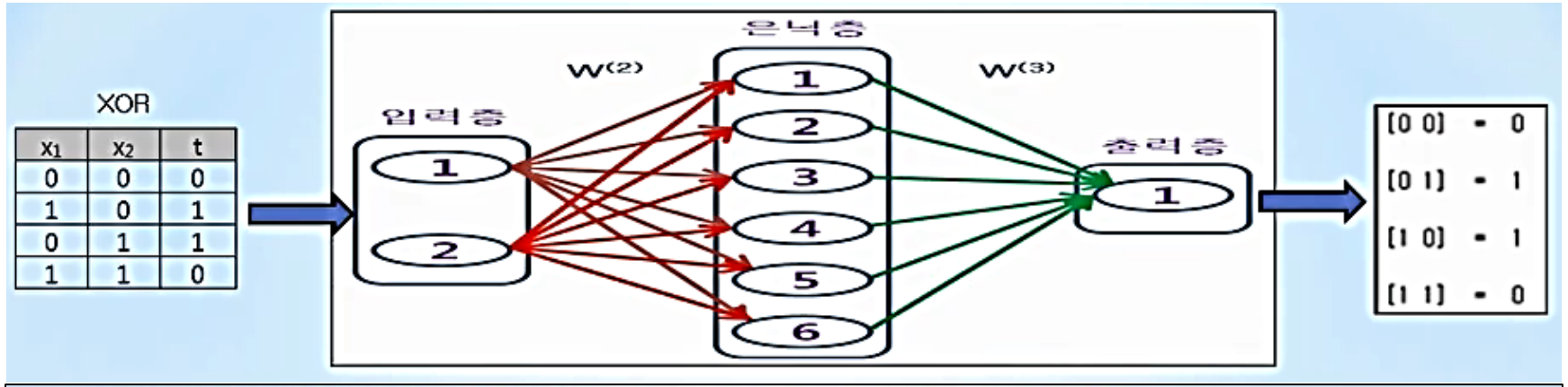
usage

```
xdata = np.array([ [0, 0], [0, 1], [1, 0], [1, 1] ])      # 입력 데이터 생성
tdata  = np.array([0, 1, 1, 0])                                  # 정답 데이터 생성 (XOR 예시)

XOR_obj = LogicGate("XOR_GATE", xdata, tdata)      # LogicGate 객체생성
XOR_obj.train()                                                  # 손실함수 최소값 갖도록 학습

XOR_obj.predict(…)                                          # 임의 데이터에 대해 결과 예측
```

XOR Gate 검증- 딥러닝 버전



➢ 신경망 기반의 딥러닝을 구현하여 XOR 문제 해결 !!

① NAND / OR / AND 조합을 이용하지 않고,

② 입력층 / 은닉층 / 출력층으로 구성된 딥러닝 아키텍처 (Neural Network) 이용

입력 층, 1개 이상의 은닉 층, 출력 층을 가지는 딥러닝을 설계한다면 , 이런 딥러닝을 이용해서 **XOR**보다 더 복잡한 **필기체 손글씨 인식**, **이미지 인식** 등의 문제도 해결 할 수 있다는 insight를 얻을 수 있음