Chapter05.

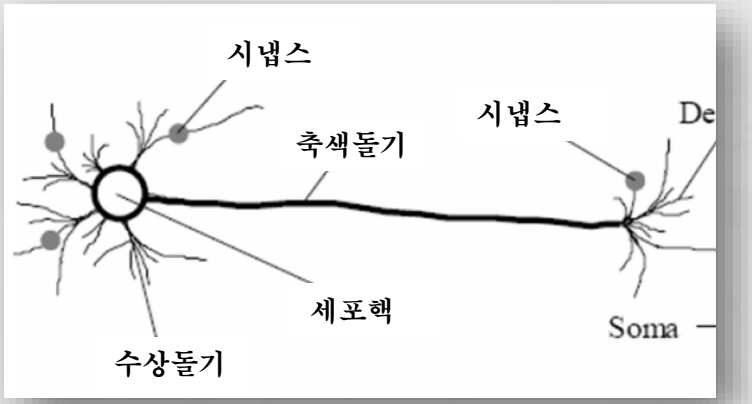
ANN & DNN

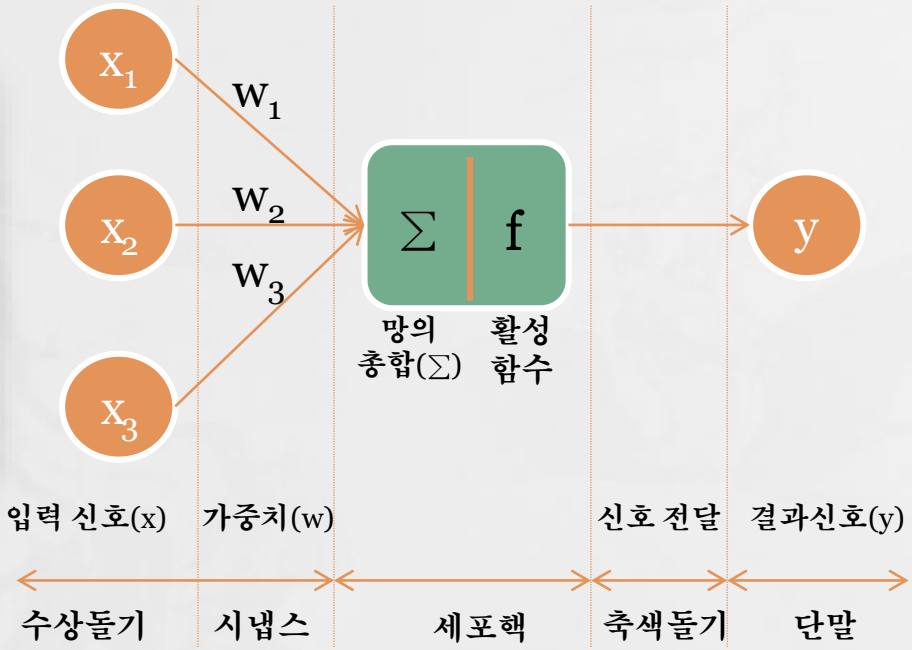
작성자 : 김진성

목차

1. 인공신경망(ANN) 개요

- 생물학적 Neural Network vs 인공 Neural Network

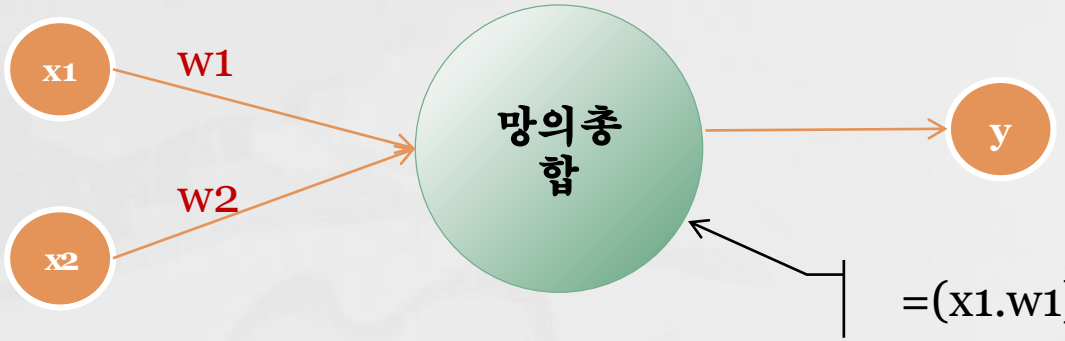




망의 총합(∑) : 입력신호(X)와 가중치(w) 곱의 합 $\sum_{i=1}^{n} W_i X_i$

활성함수(f) : 망의 총합을 받아서 다음 뉴런(y) 전달 출력 $y = f\left[\sum_{i=1}^{n} W_i X_i\right]$

● 가중치(weight)



➢입력(x1, x2) : 수상돌기에 해당하는 외부 신경 자극

➢가중치 (w1, w2) : 시냅스에서 신호 세기 결정

✓ 신호 세기 = 가중치(w1, w2) : 중요 변수에 따라서 가중치가 달라짐

✓ x변수가 y에 주는 영향력을 나타내는 값

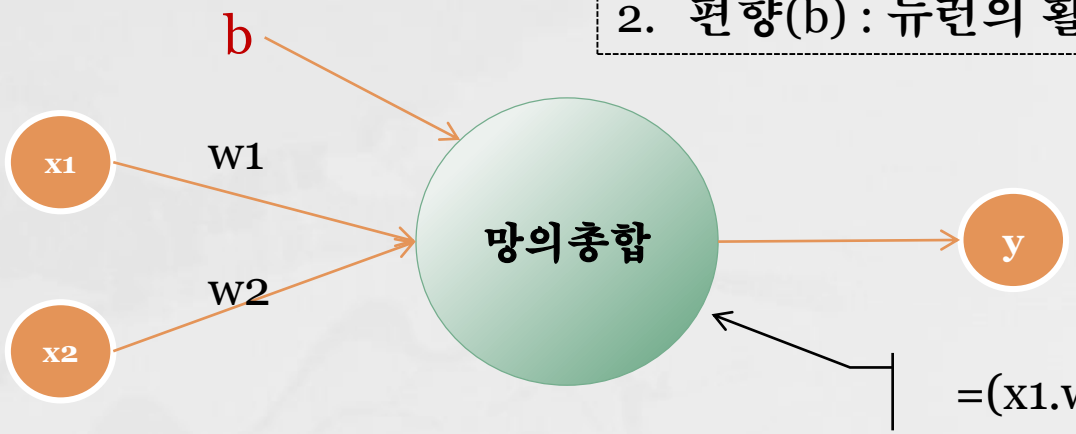
✓값이 클 수록 해당 변수가 더 중요하다는 의미(강한 신호를 y에 보낸다.)

➢출력(y) : 망의 총 합을 축색돌기로 통해서 받음

✓ y = (x1.w1) + (x2.w2)

● 편향(bias)

<<2개 조절변수>>
1. 가중치(w) : x 변수의 중요도 조절변수
2. 편향(b) : 뉴런의 활성화 조절 변수



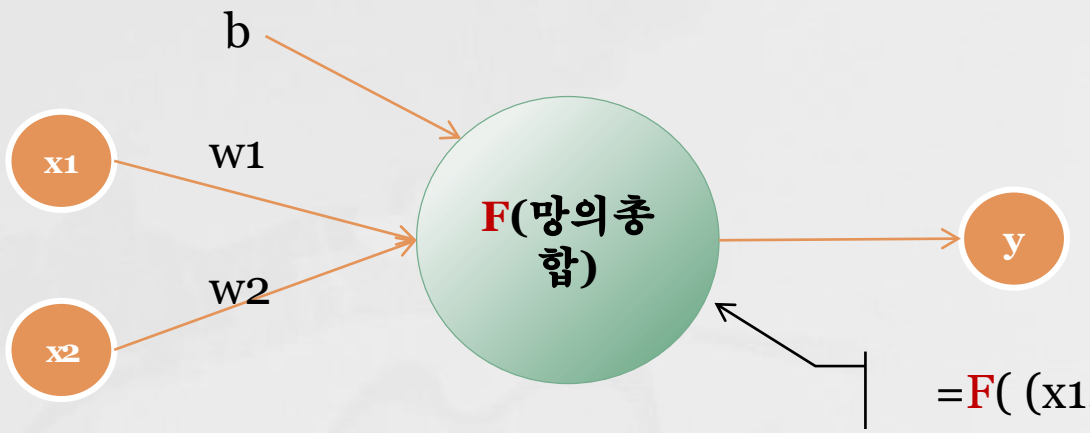
$$=(x1.w1 + x2.w2) + b$$

$$= \sum_{i=1}^{n} w_i x_i + b$$

➢편향(bias) : 뉴런의 활성화를 조절하는 변수

- ✓ 활성(1)/비활성(0)의 임계치
- ✓ 망의 총합을 다음 계층으로 넘길 때 기준이 되는 값
- ✓ y = (x1.w1 + x2.w2) + b
- ✓ cf) 선형회귀방정식과 동일 : y = (a1.x1 + a2.x2) + b : (b : 상수)

● 활성화 함수(Activation Function)



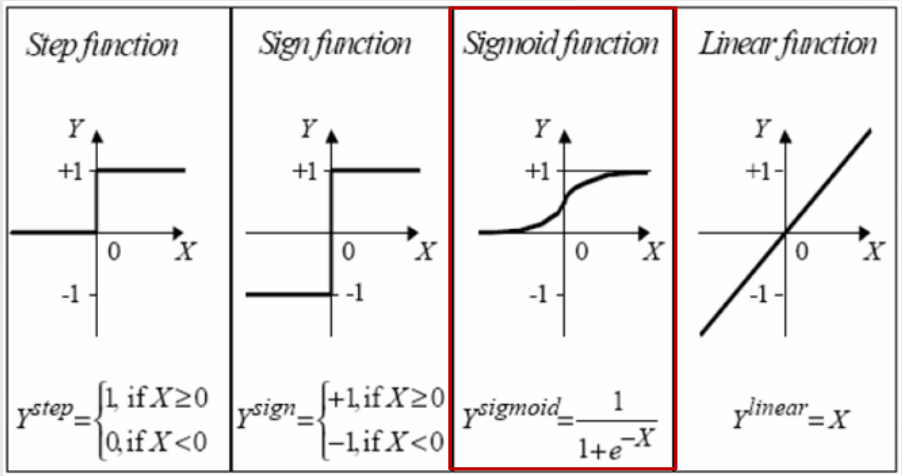
$$= F((x1.w1 + x2.w2) + b)$$

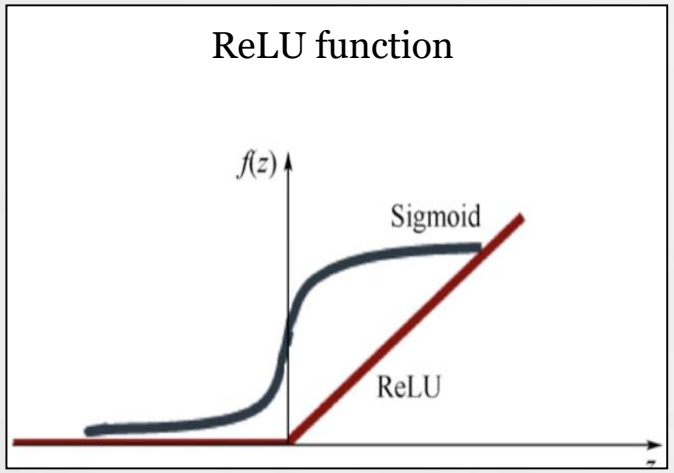
$$= F\left[\sum_{i=1}^{n} w_i x_i + b \right]$$

- 활성함수(Activation Function) : 망의 총합을 다른 뉴런으로 활성화 결정
  - ✓ 망의 총합이 bias(임계값) 보다 크면 활성화(1), 작으면 비활성화(0)
  - ✓ 활성함수의 유형에 따라서 출력 값 범위 결정 : **(0 ~ 1), (-1 ~ 1), (-inf ~ inf)**

● 활성화 함수 유형

➢ Step function, Sign function, Sigmoid function, Linear function, ReLU(Rectified Linear Unit)





➢ Sigmoid function

✓0~1 까지 연속적으로 변화하는 출력값을 갖기 때문에 가중치나 바이어스(bias) 변화 시 출력에 변화를 준다.(예: 0.5이상이면 1 연산, 현재 활성함수로 가장 많이 사용)

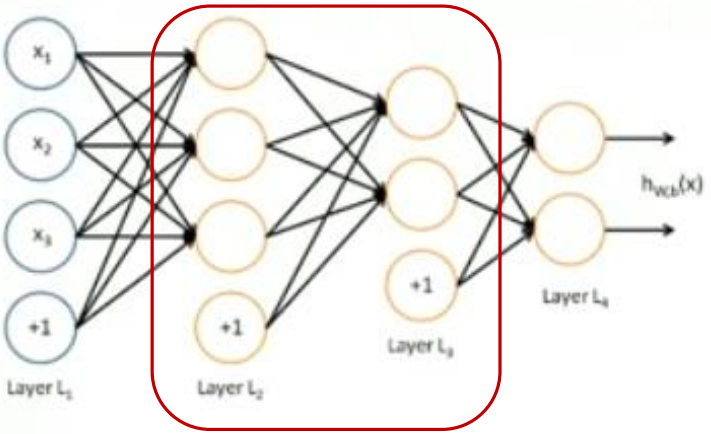
➢ **ReLU function**

✓ 0보다 작을 때는 0을 사용, 0보다 큰 값에 대해서는 해당 값을 그대로 사용하는 방법

2. Deep Learning 유형

A *deep learning* model

- A neural network
- add *multiple* hidden layers?
- Input → hidden layer → … → hidden layer → output

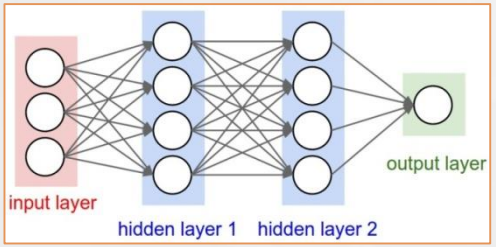


Difficulties

- Not enough data
- Local optima
- Diffusion of gradients

"Houston, we have a problem"

- **심층 신경망(DNN : Deep Neural Network)**
  - ✓ 입력층(input layer)과 출력층(output layer) 사이에 여러 개의 은닉층(hidden layer)
  - ✓ 중간층의 다층화로 뉴런 처리와 전달, 산출되는 특징 값이 늘어남 → 정확도 향상
  - ✓ 파라미터 수가 너무 많아짐 → 연산 많아짐, 과적합
  - ✓ 완전연결 계층(Fully Connected NN)



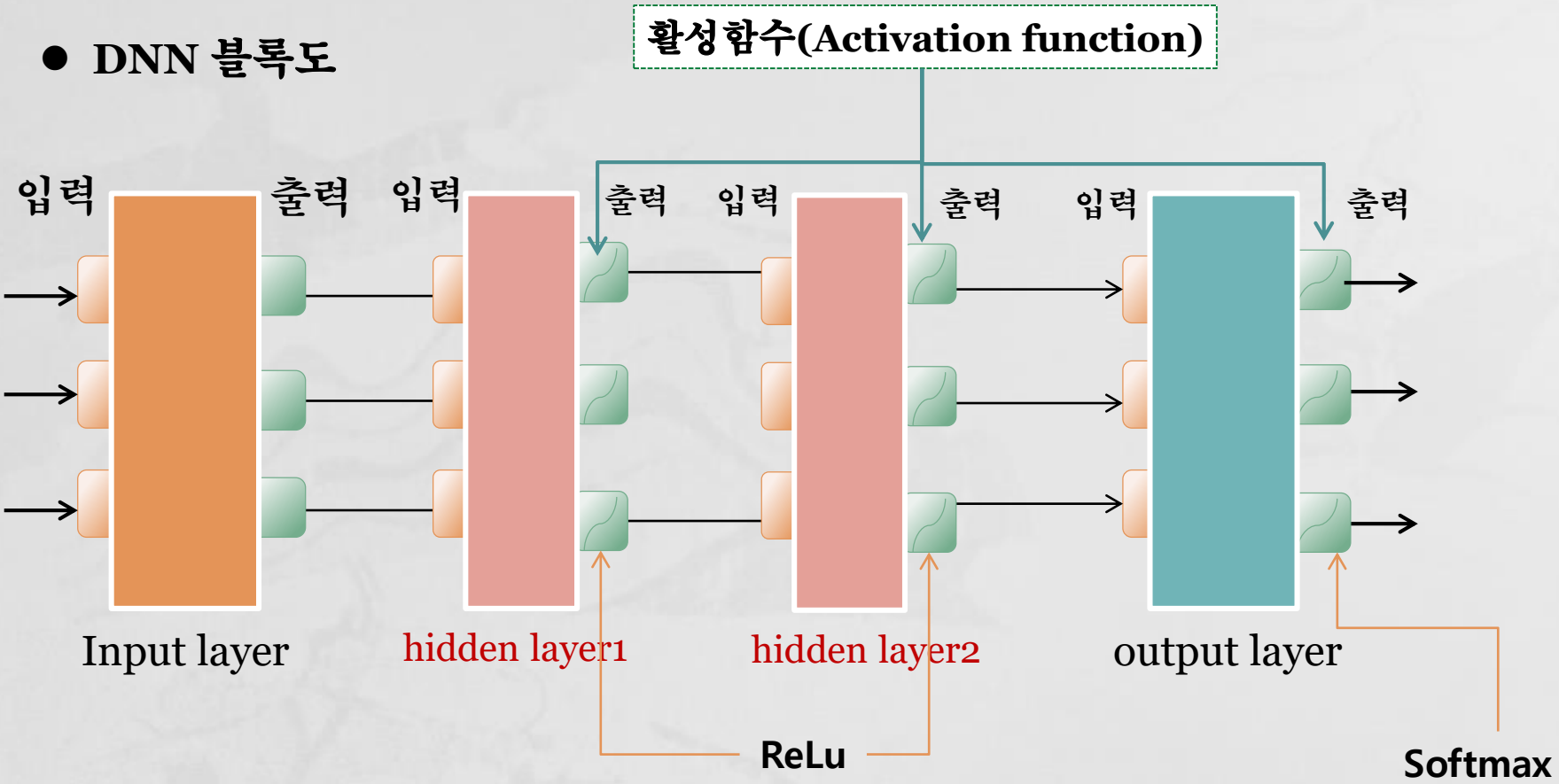
- **심층 신뢰 신경망 (DBN : Deep Belief Network)**
  - ✓ 다중계층으로 이루어진 심층 신경망
  - ✓ Dropout : 과적합 문제 해결을 위한 무작위 네트워크 삭제
  - ✓ 제한된 볼츠만 머신(Restricted Boltzmann Machine: RBM) : 가중치 갱신 알고리즘

- **합성곱 신경망 (Convolution Neural Network, CNN)**
  - ✓ 이미지 입력 → 어떤 이미지인지 판별하는 Classification모델
  - ✓ 기계학습에서 이미지의 정보를 뉴런에 전달할 때 이미지의 일부 범위로 좁혀서 분석하고 그 범위를 조금씩 잘라내며 분석을 반복하는 방식

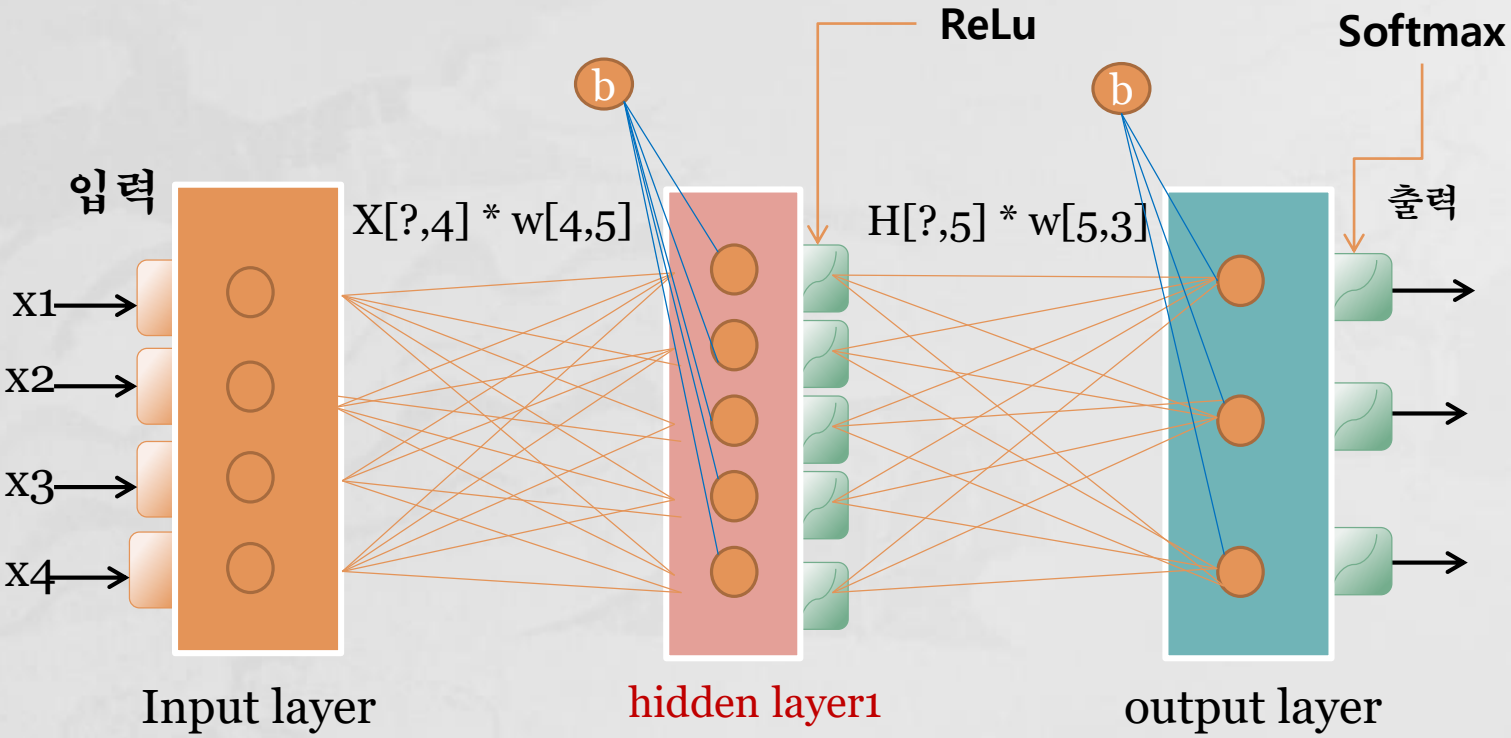
- **순환신경망 (Recurrent Neural Network, RNN)**
  - ✓ 시계열 분석을 가능하게 만들어 준 동적 데이터와 호환되는 딥러닝 모델
  - ✓ 동적 데이터 : 자연어 대화, 동영상, 음성, 시계열의 통계 데이터와 로그 데이터
  - ✓ 최근 자연어 대화 등의 분야에서 많이 이용

3. DNN(Deep Neural Network) layers

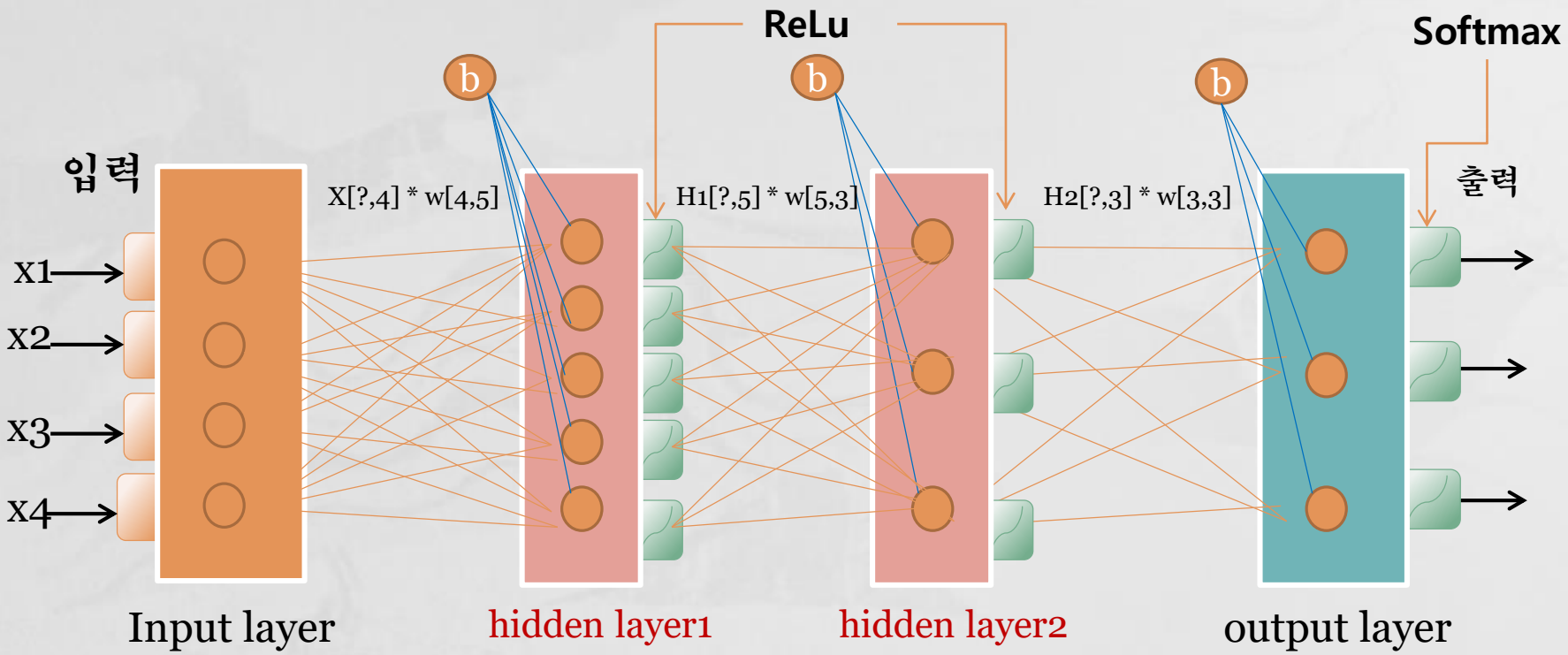
- DNN 블록도



- **Hidden layer1(node=5) 블록도**

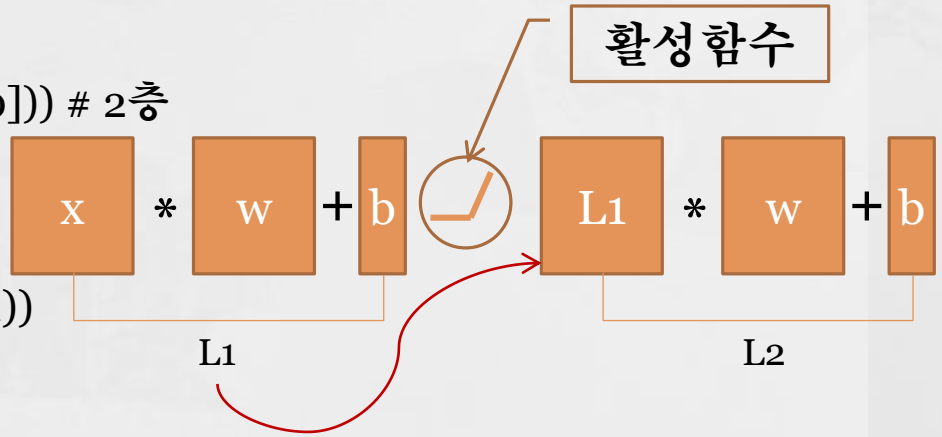


- **Hidden layer2(node=5, node=3) 블록도**



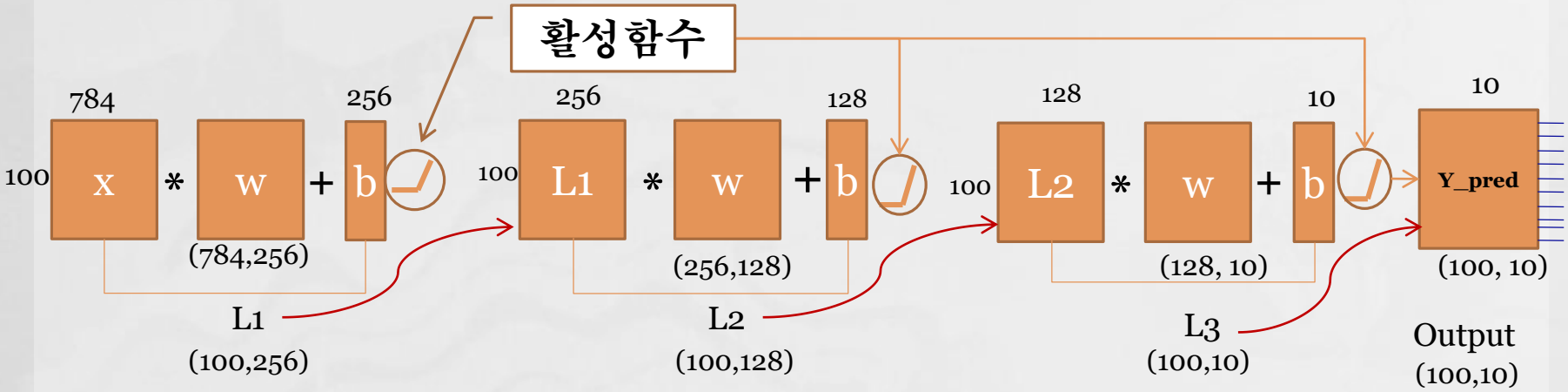
1) ReLu : Hidden layer 활성함수

```
# tf Graph Input
X = tf.placeholder(tf.float32, [None, 784]) # mnist data image of shape 28*28=784
Y = tf.placeholder(tf.float32, [None, 10])  # 0-9 digits recognition => 10 classes

# Set model weights
W1 = tf.Variable(tf.random_normal([784, 256])) # 1층
B1 = tf.Variable(tf.random_normal([256]))

W2 = tf.Variable(tf.random_normal([256, 10])) # 2층
B2 = tf.Variable(tf.random_normal([10]))

# Construct model
L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
y_pred = tf.add(tf.matmul(L1, W2), B2)
```



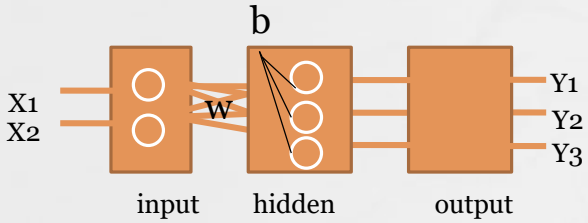
Sigmoid 함수를 deep learning에 적용시키면
계층이 증가할 수록 0에 가까은 값이 나타남
따라서 0보다 큰 값은 해당 값을 직접 나타내는
Relu 함수를 이용한다.

- MNIST dataset



2) Node : Hidden layer neuron

- **단층 ANN인 경우 : hidden = output**



➢ Weight 수 : 다차원

Weight 행 = input node

Weight 열 = output size=hidden node

```
X = tf.placeholder(tf.float32, [None, 2])
Y = tf.placeholder(tf.float32, [None, 3])
w = tf.Variable(tf.random_normal([2, 3]))
```

➢ Bias 수 : 1차원

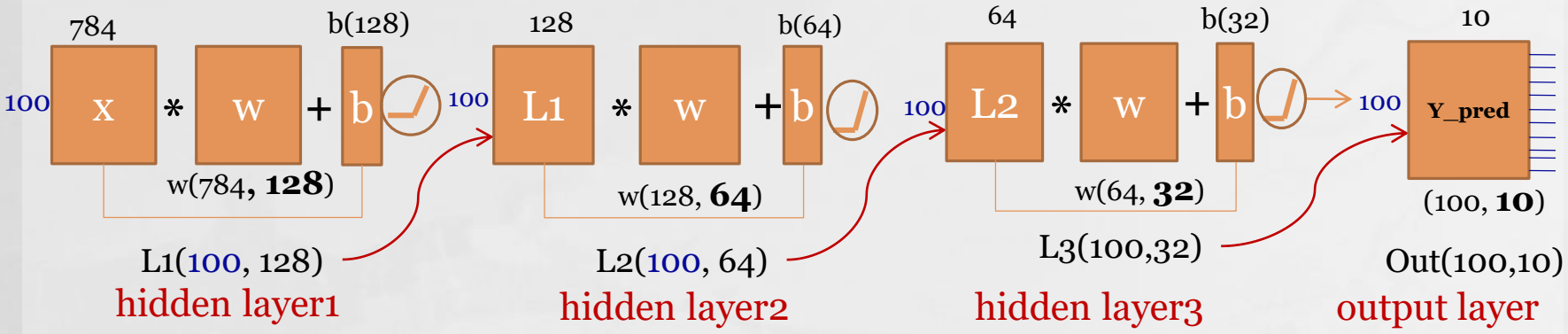
Bias size = output size=hidden node

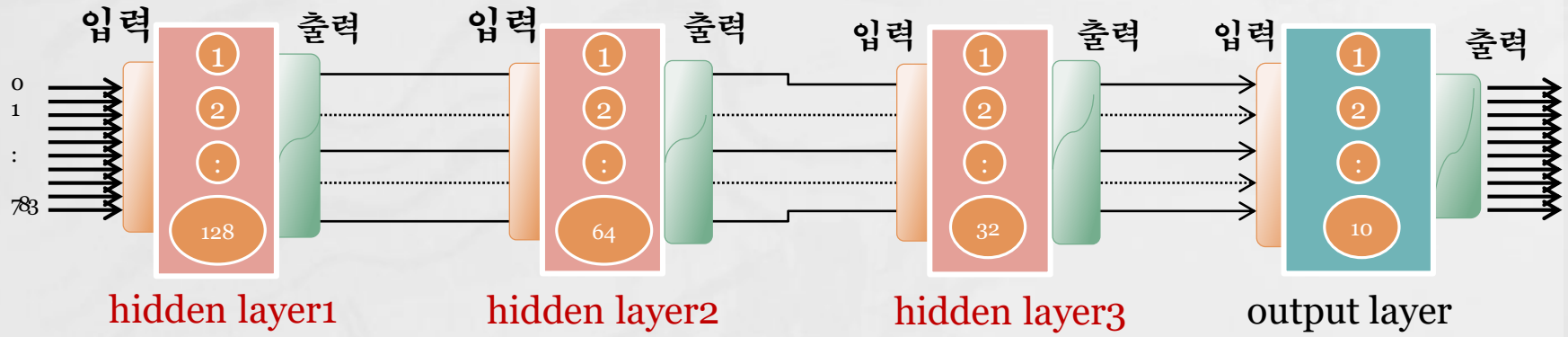
```
b = tf.Variable(tf.zeros([3]))
```

※ 단층 layer : hidden node 수 = 출력 수

● **다층 ANN인 경우 : hidden ≠ output**

※ Hyper parameter : hidden layer & node





➢ Weight 차원 : 2차원

Weight 행 = 입력 node(이전 layer 출력 수)

Weight 열 = 출력 node(다음 layer 입력 수)

➢ Bias 차원 : 1차원

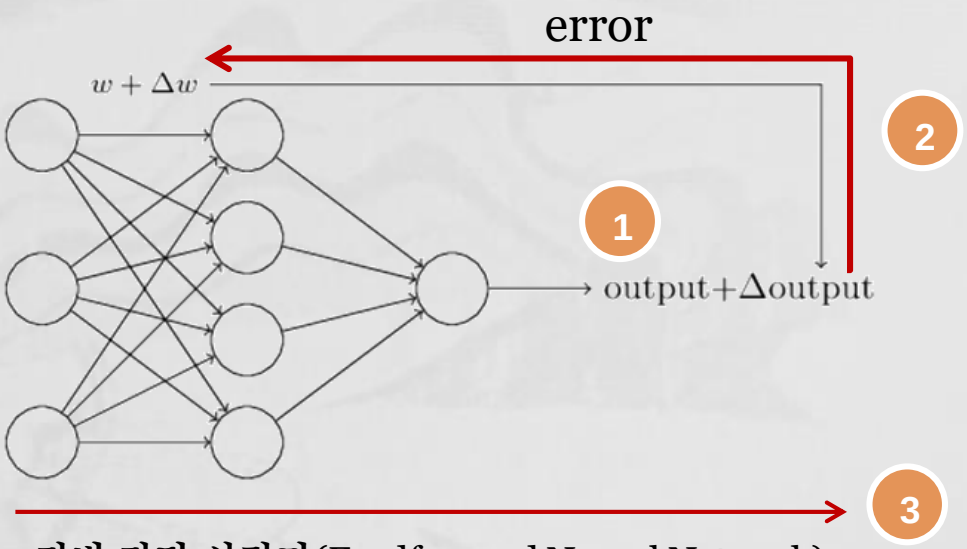
출력 node(다음 layer 입력)

※ layer node 수 : 입력층과 가까울 수록 많게, 출력층과 가까울 수록 적게 지정

3) 역전파(Backpropagation) 알고리즘

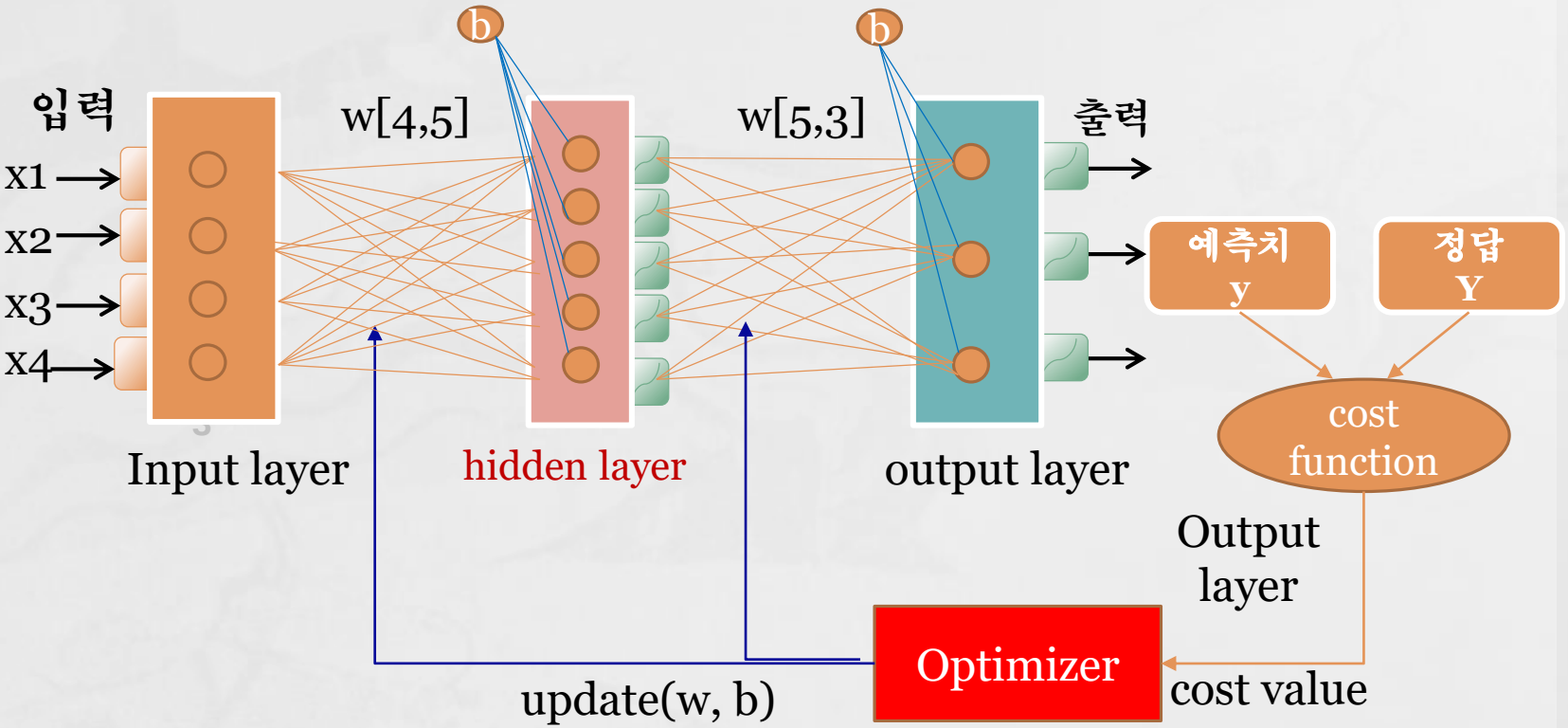
✓딥러닝 프레임워크 필수 라이브러리

✓출력에서 생긴 오차(error)를 입력 쪽(역방향)으로 전파시켜 순차적으로 편미분을 수행하여 훈련 데이터에 최적화된 weight값을 얻는 알고리즘



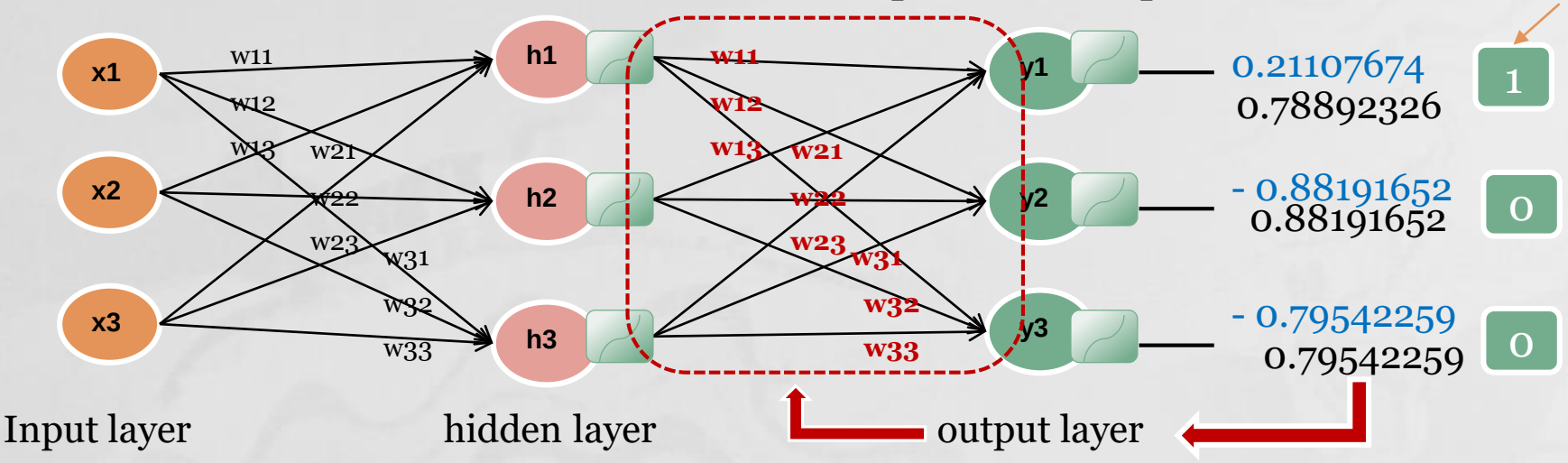
전방 전달 신경망(Feedforward Neural Network):

- DNN model 신경망
  - ✓ Hidden layer 1인 경우



[역전파 단계1] Hidden vs Output 가중치 수정

은닉층과 출력층 사이 가중치 수정 : output 오차, output, hidden 이용




```
wo += tf.matmul( (output_errors * output_outputs * (1 - output_outputs)),
                 tf.transpose(hidden_outputs) )
```

$$\begin{bmatrix} [[\ 0.21107674\ ] \\ [-0.88191652] \\ [-0.79542259]] \end{bmatrix} \times \begin{bmatrix} [[\ 0.78892326] \\ [\ 0.88191652] \\ [\ 0.79542259]] \end{bmatrix} \times \begin{bmatrix} [[\ 1-0.78892326] \\ [\ 1-0.88191652] \\ [\ 1-0.79542259]] \end{bmatrix} * \begin{bmatrix} [[\ 0.774682\ ] \\ [\ 0.82362533] \\ [\ 0.90239143]] \end{bmatrix}$$

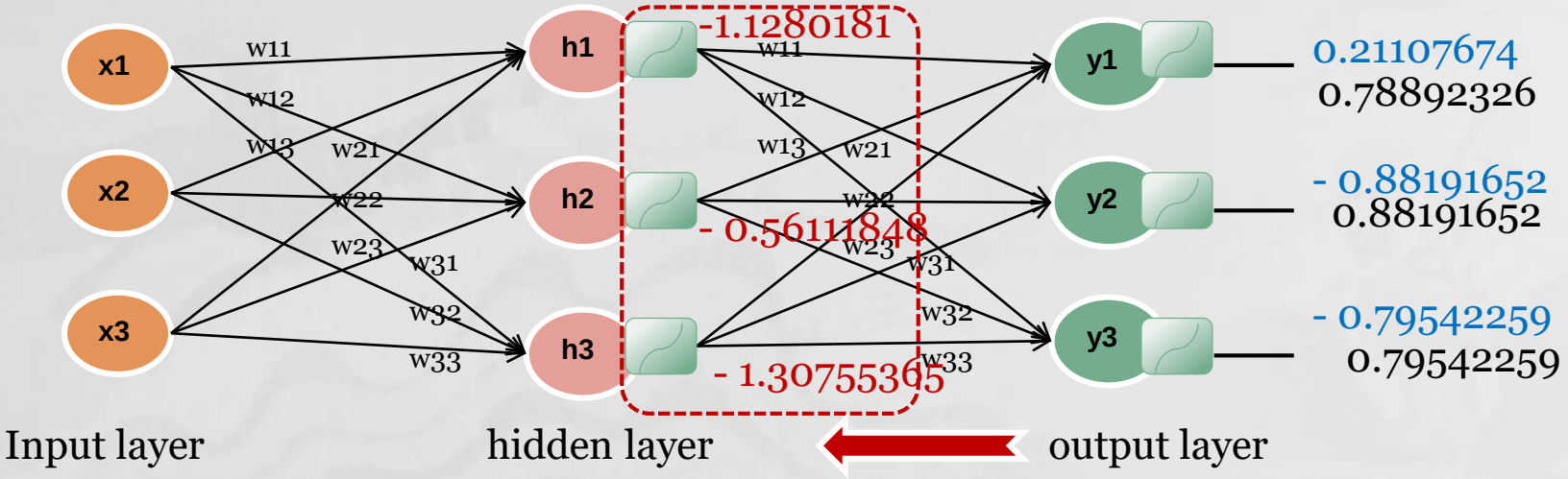
$wo(3x3) \leftarrow 3x1 * 1x3$

T

[[0.774682 0.82362533 0.90239143]]

[역전파 단계2] Input vs Hidden 가중치 수정

1) hidden 오차 = hidden 가중치(wo) 전치행렬 * output 오차



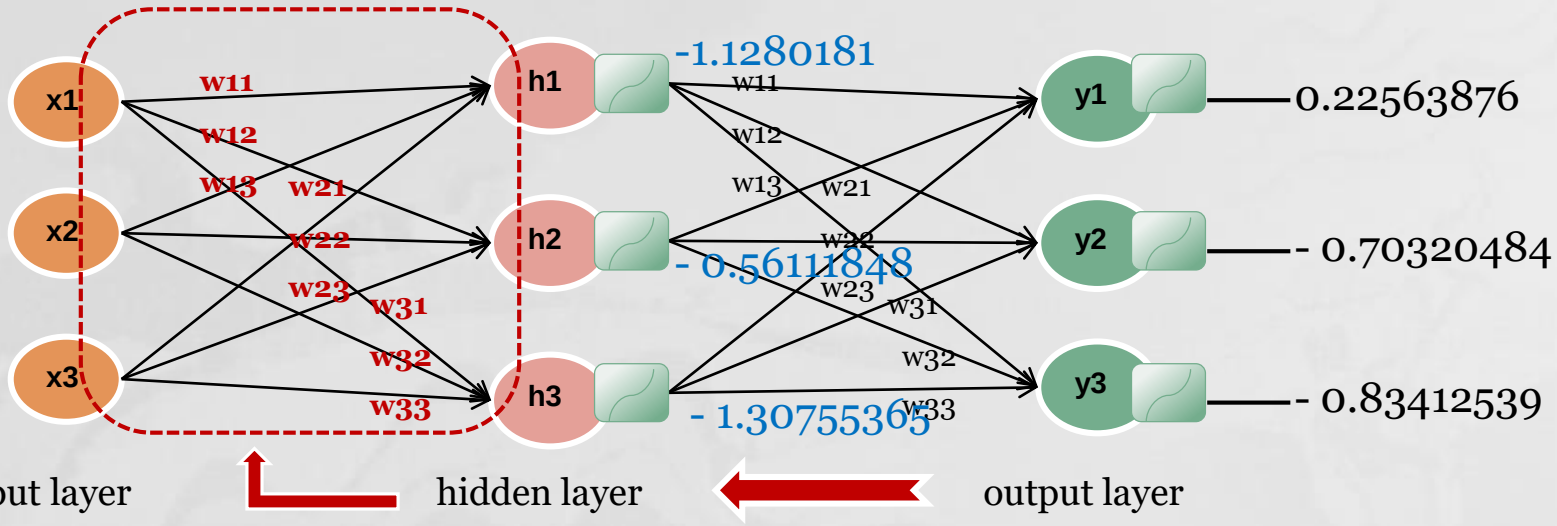
hidden_errors = tf.matmul(wo.T, output_errors) # 은닉층 오차

$$\begin{bmatrix}-1.1280181\\ -0.56111848\\ -1.30755365\end{bmatrix} = \begin{bmatrix}0.36751758 & 0.84878687 & 0.5313797\\ 0.43590182 & 0.3637141 & 0.39280269\\ 0.45776142 & 0.76221831 & 0.85123049\end{bmatrix} * \begin{bmatrix}0.21107674\\ -0.88191652\\ -0.79542259\end{bmatrix}$$

3x1 ← wo(3x3) 3x1

❖ 입력층과 은닉층 사이의 가중치를 수정하기 위해서 은닉층 오차 필요

2) Input vs Hidden 가중치 수정 : hidden 오차와 hidden output, input 이용



wi += tf.matmul((hidden_errors * hidden_outputs * (1 - hidden_outputs)), tf.transpose(inputs))

$$\begin{bmatrix} [-1.1280181] \\ [-0.56111848] \\ [-1.30755365] \end{bmatrix} \times \begin{bmatrix} [0.\ 0.774682] \\ [0.\ 0.8236253] \\ [0.\ 0.9023914] \end{bmatrix} \times \begin{bmatrix} [1-0.774682] \\ [1-0.8236253] \\ [1-0.9023914] \end{bmatrix} * \begin{bmatrix} [5.1] \\ [3.5] \\ [1.4] \end{bmatrix}$$

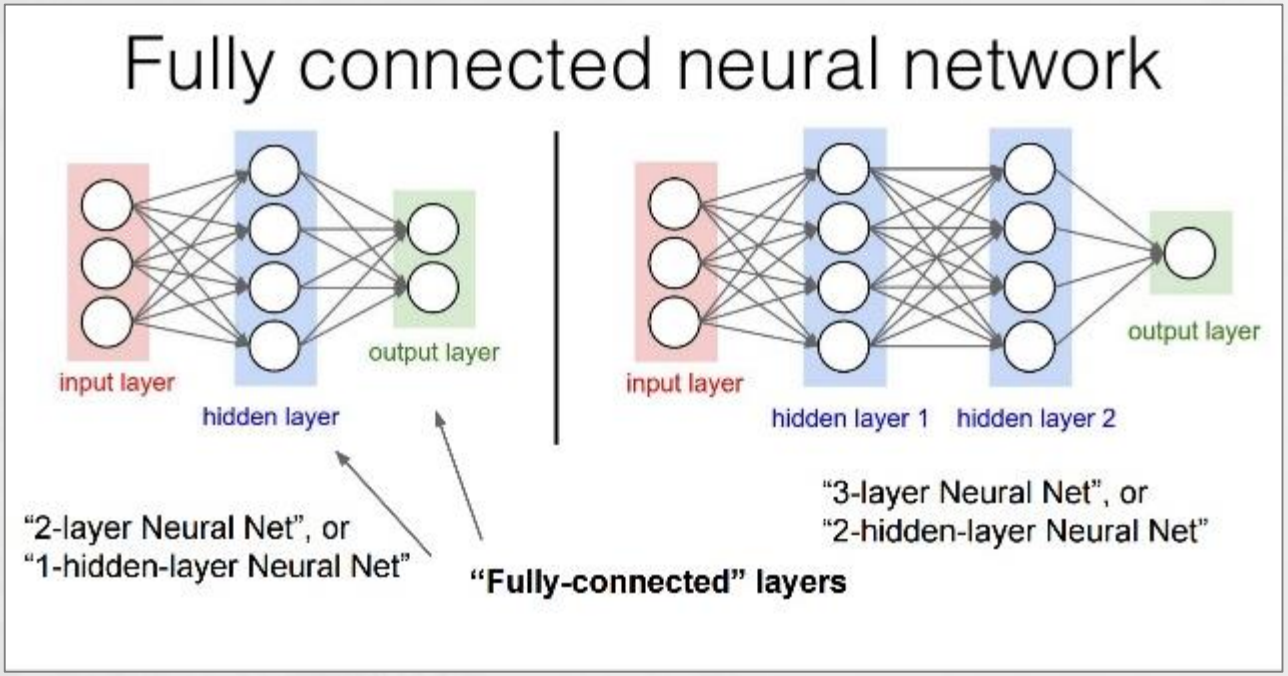
T

[[5.1 3.5 1.4]]

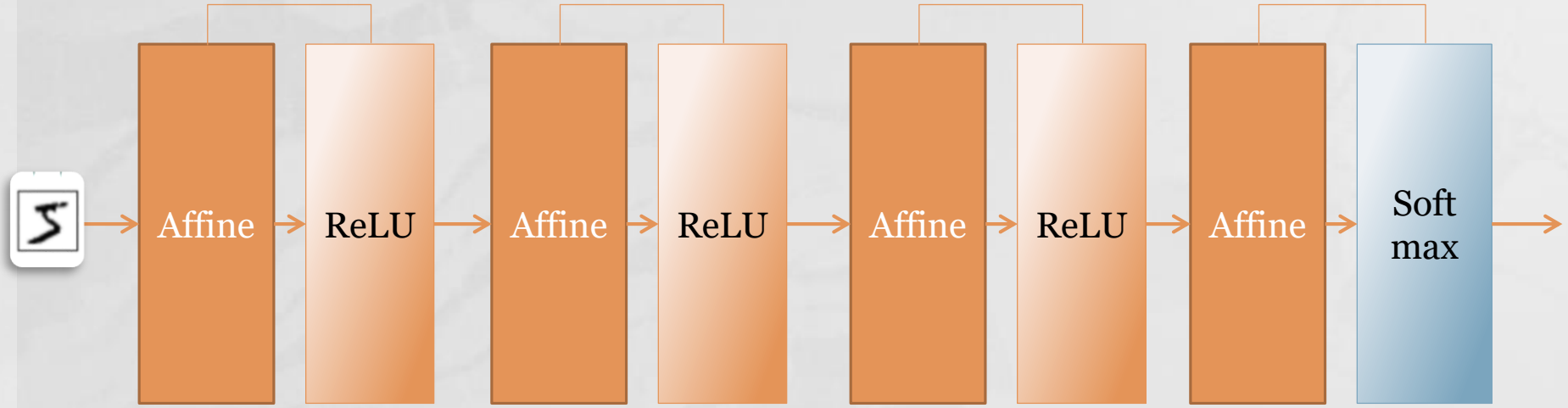
wi(3x3) ← 3x1 * 1x3

4) Fully Connected Layers

- 인접한 계층의 모든 뉴런과 결합
- Affine 계층 : 완전히 연결된 계층

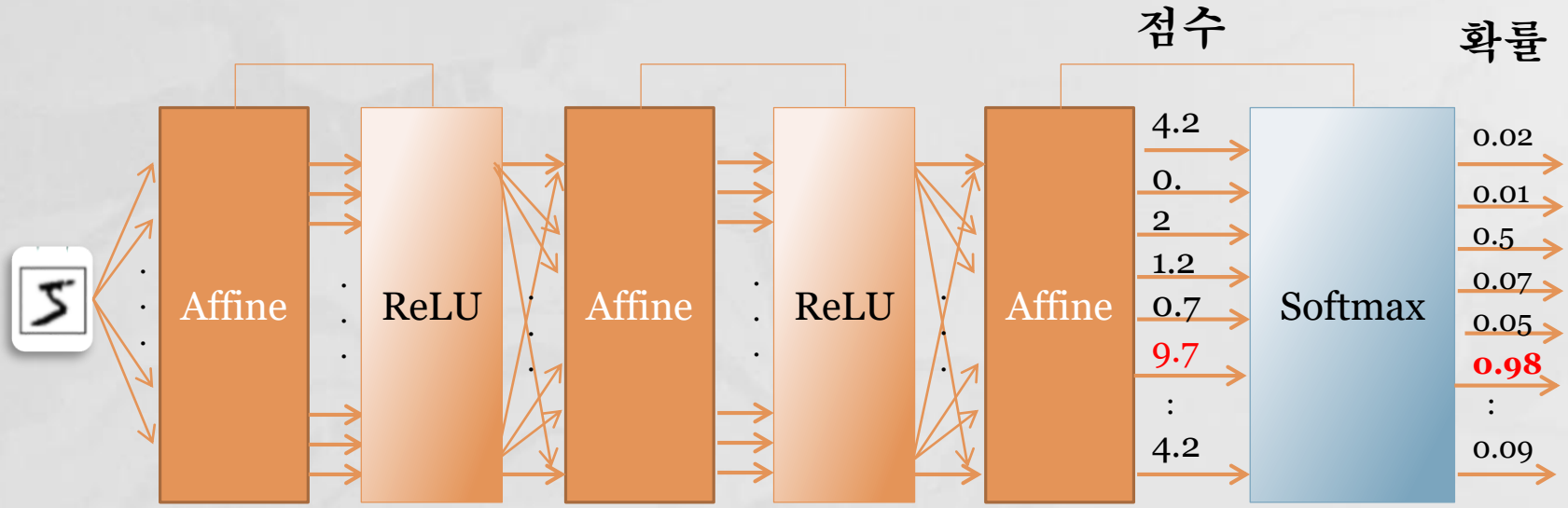


- Affine 계층 : 완전히 연결된 계층
  - ✓ 4개 완전연결 신경망 예



Affine 계층 뒤에 활성화 함수 ReLU 계층(or Sigmoid) 연결
마지막 4번째 계층은 Affine 계층과 Softmax 계층에서 확률값으로 최종 출력

- Affine 계층 : 완전히 연결된 계층
  - ✓ 3개 완전연결 신경망 예



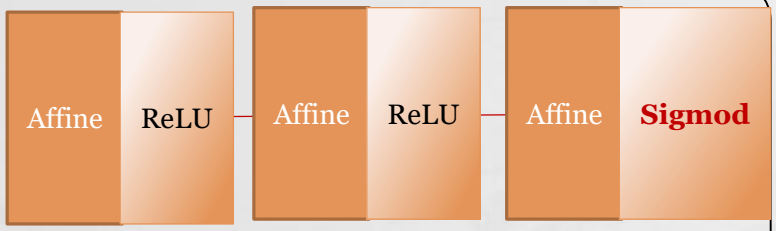
점수 : 정규화하지 않은 출력 결과(Affine 계층) -> 비율척도(회귀분석)

확률 : 입력값을 정규화(출력의 합 1)한 출력 결과(Softmax 계층) -> 0~1사이

4. DNN model(layer3)

- **Sigmoid 이항 분류기**

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# Deep learning(weights & bias for DNN layers)
W1 = tf.Variable(tf.random_normal([784, 256])) # 1층
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 128])) # 2층
b2 = tf.Variable(tf.random_normal([128]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([128, 10])) # 3층
b3 = tf.Variable(tf.random_normal([10]))
y_pred = tf.sigmoid(tf.matmul(L2, W3) + b3)
```



● Softmax 다항 분류기

```
# input place holders
X = tf.placeholder(tf.float32, [None, 784])
Y = tf.placeholder(tf.float32, [None, 10])

# Deep learning(weights & bias for DNN layers)
# - ReLu activation function apply
W1 = tf.Variable(tf.random_normal([784, 256])) # 1층
b1 = tf.Variable(tf.random_normal([256]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)

W2 = tf.Variable(tf.random_normal([256, 128])) # 2층
b2 = tf.Variable(tf.random_normal([128]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)

W3 = tf.Variable(tf.random_normal([128, 10])) # 3층
b3 = tf.Variable(tf.random_normal([10]))
y_pred = tf. softmax(tf.matmul(L2, W3) + b3)
```