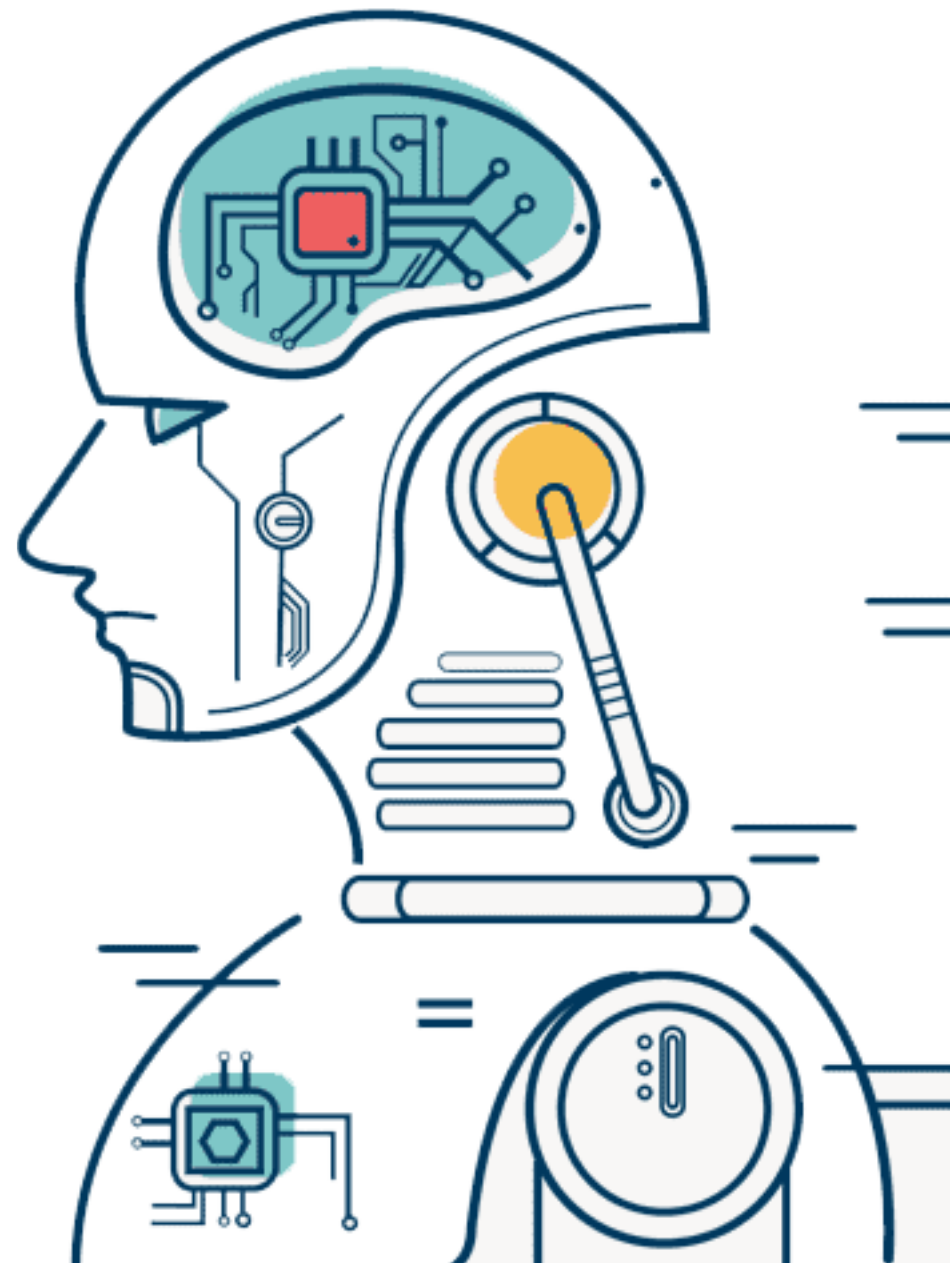
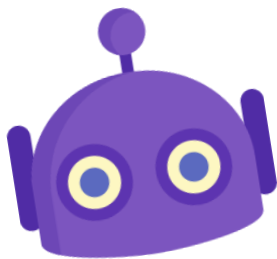


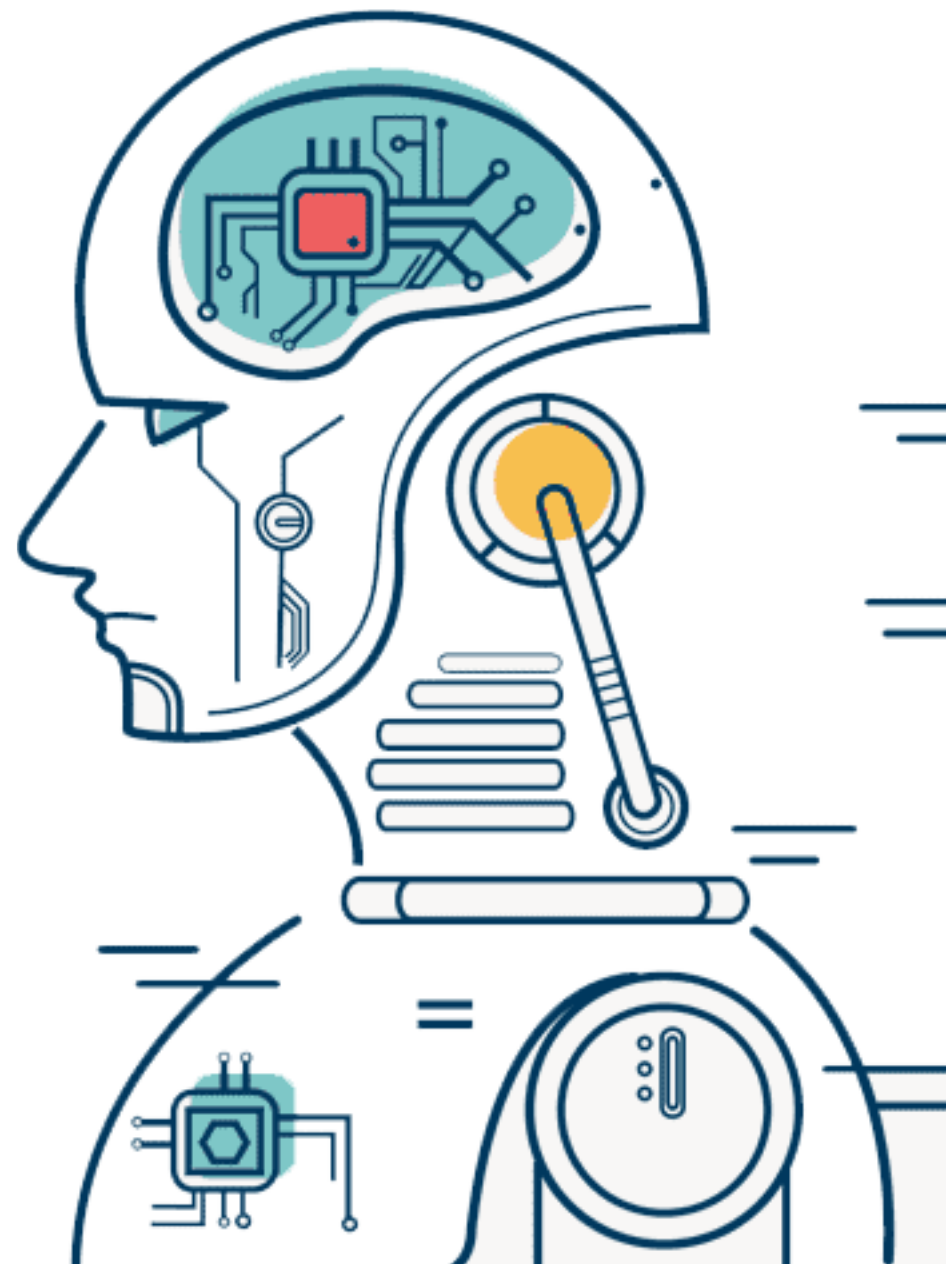
Deep Learning

Chapter 2 딥러닝 기초 (퍼셉트론, 다층 퍼셉트론)



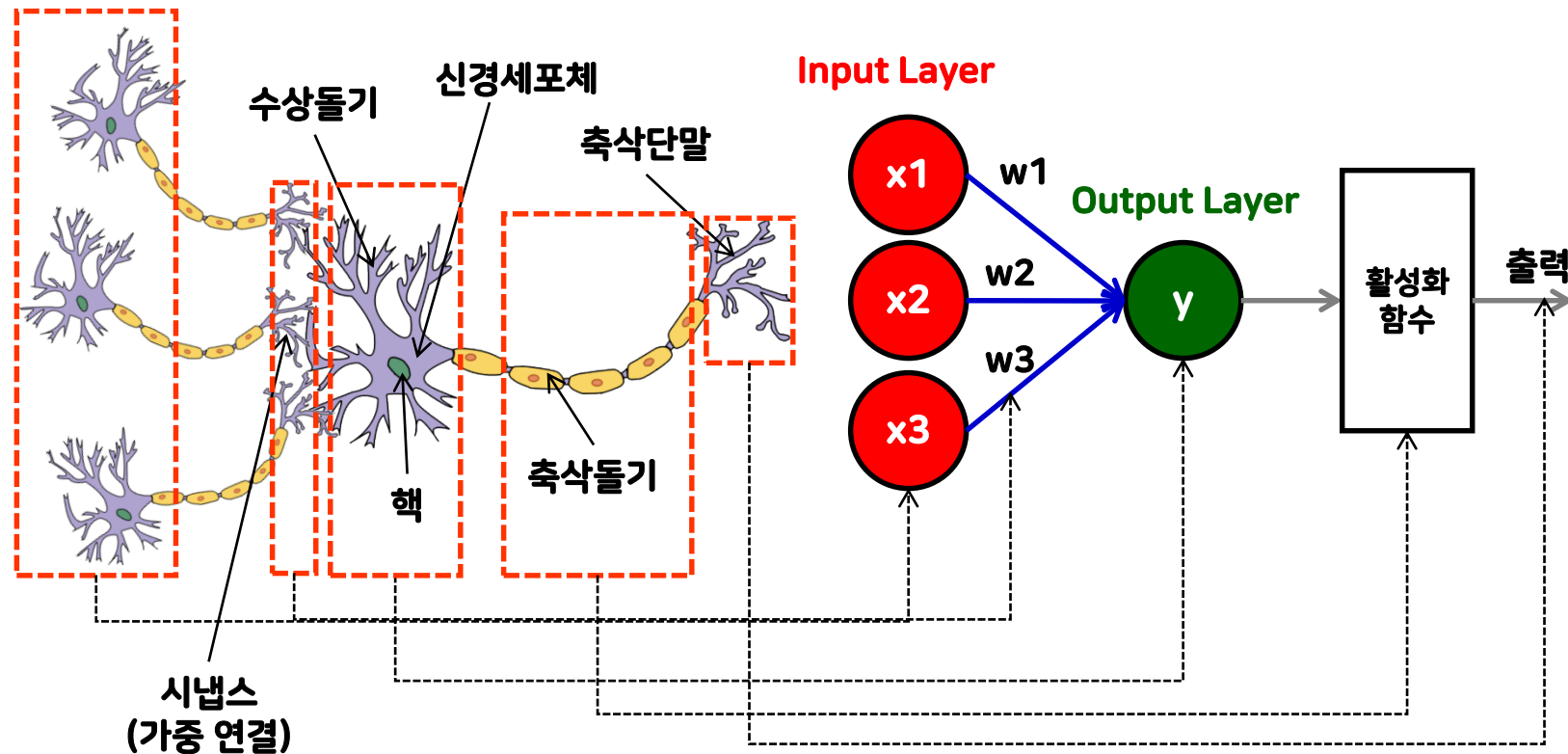


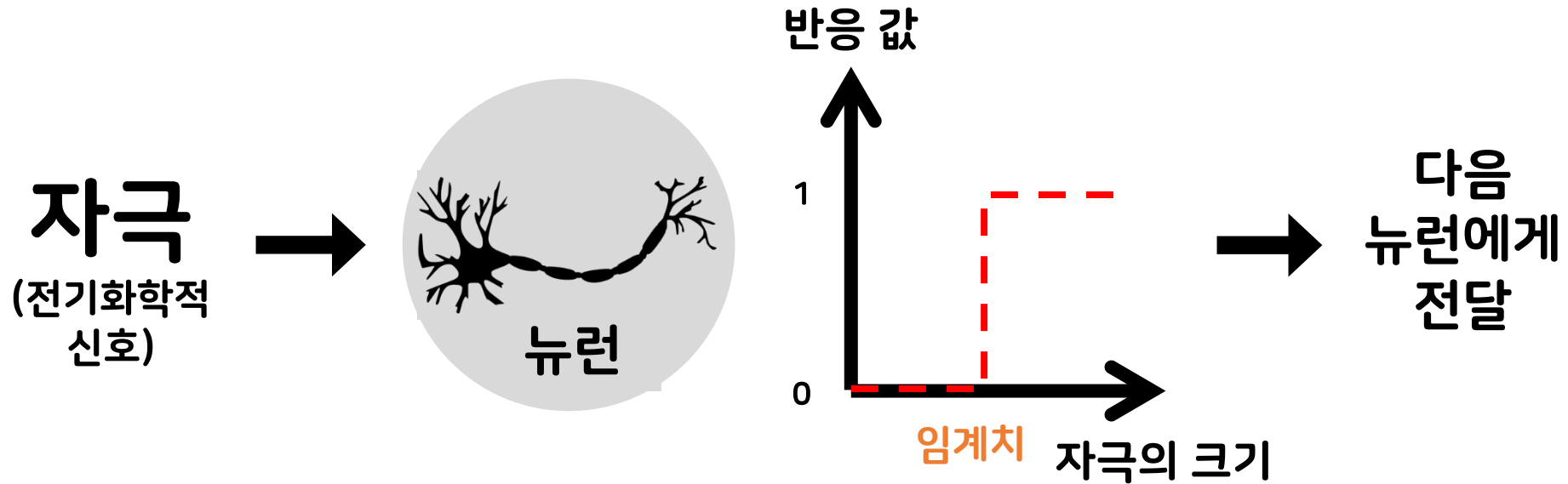
퍼셉트론



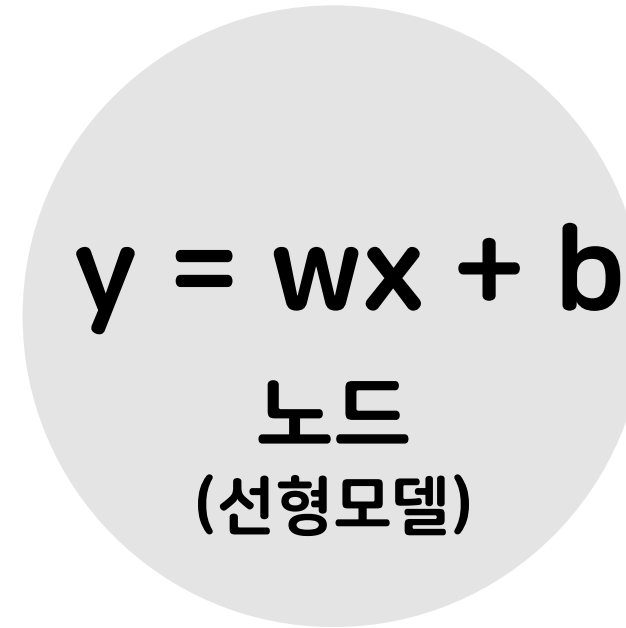
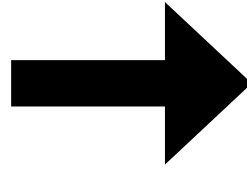
- 퍼셉트론의 개념을 이해하고, 구현 할 수 있다.
- 다층 퍼셉트론의 개념을 이해하고, 구현 할 수 있다.
- 행렬 곱셈, 행렬 덧셈을 이해 할 수 있다.

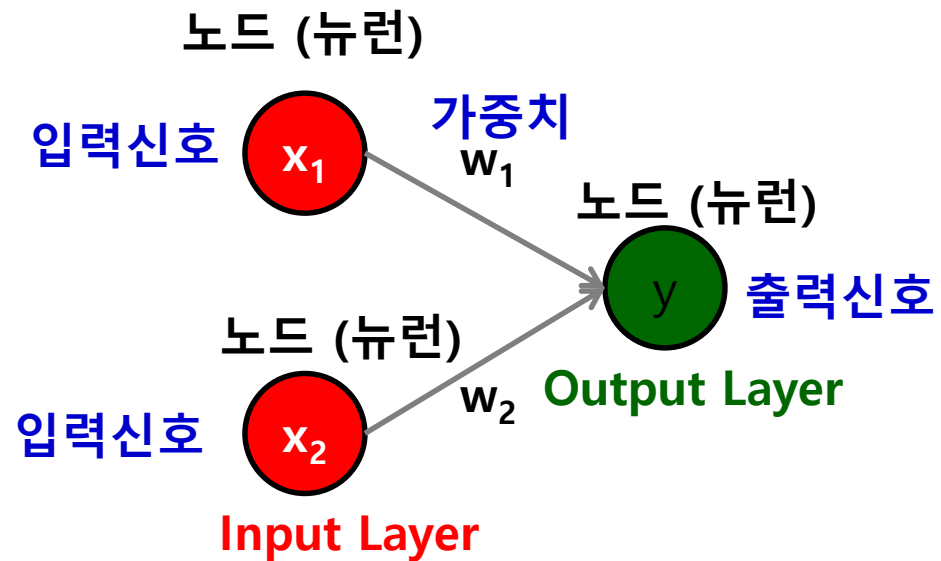
- **추상화** : 현실 세계를 가상 세계로 모델링하는 작업
- 신경망의 추상화 : 인간의 뉴런을 하나를 **노드** (인공 뉴런)으로 가상화하고 각 **노드의 특성** (가중치)를 **다르게 설정**하여 동일한 입력에 대해 다양한 반응을 발생하도록 하게 함.





신경의 흥분이 전달되기 위해서는 뉴런에 전달되는 자극의 크기가 임계치 이상이 되어야 함





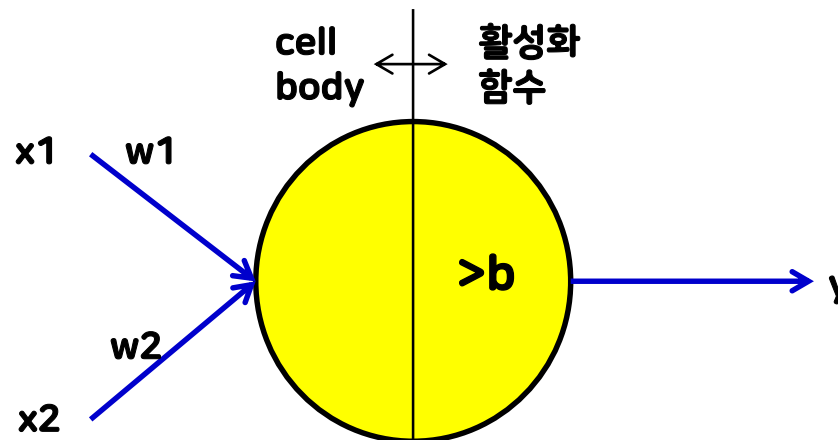
$$y = W_1 X_1 + W_2 X_2$$

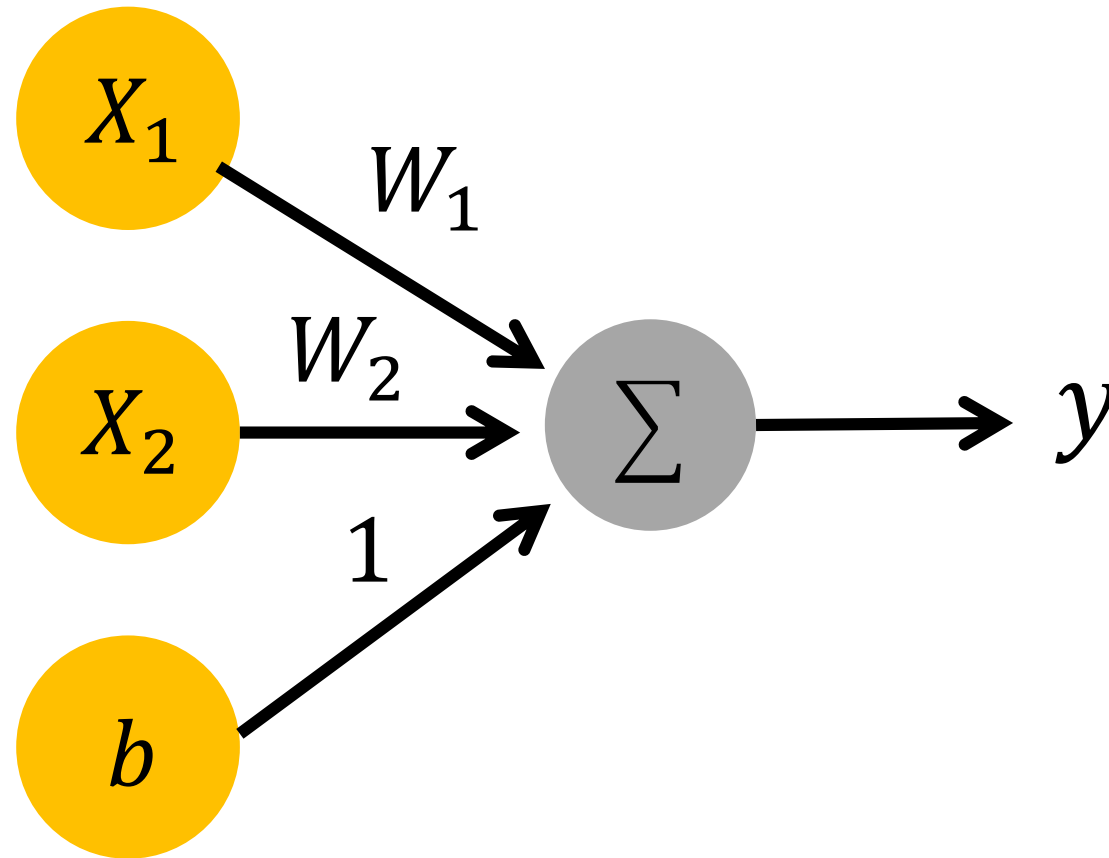
$$y = \sum_{i=0}^n W_i X_i$$

- 뉴런에서 보내온 신호의 총합이 정해진 임계값(b)을 넘어설 때만 1을 출력 (뉴런 활성화)

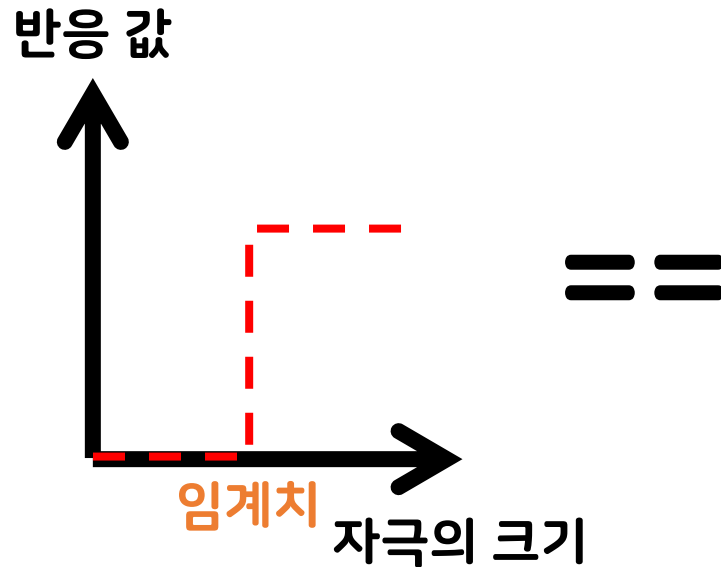
$$y = \begin{cases} 0 & (w_1x_1 + w_2x_2 \leq b) \\ 1 & (w_1x_1 + w_2x_2 > b) \end{cases}$$

- 가중치가 클수록 해당 신호가 결과에 영향을 크게 준다는 것을 의미





$$y = W_1X_1 + W_2X_2 + b$$

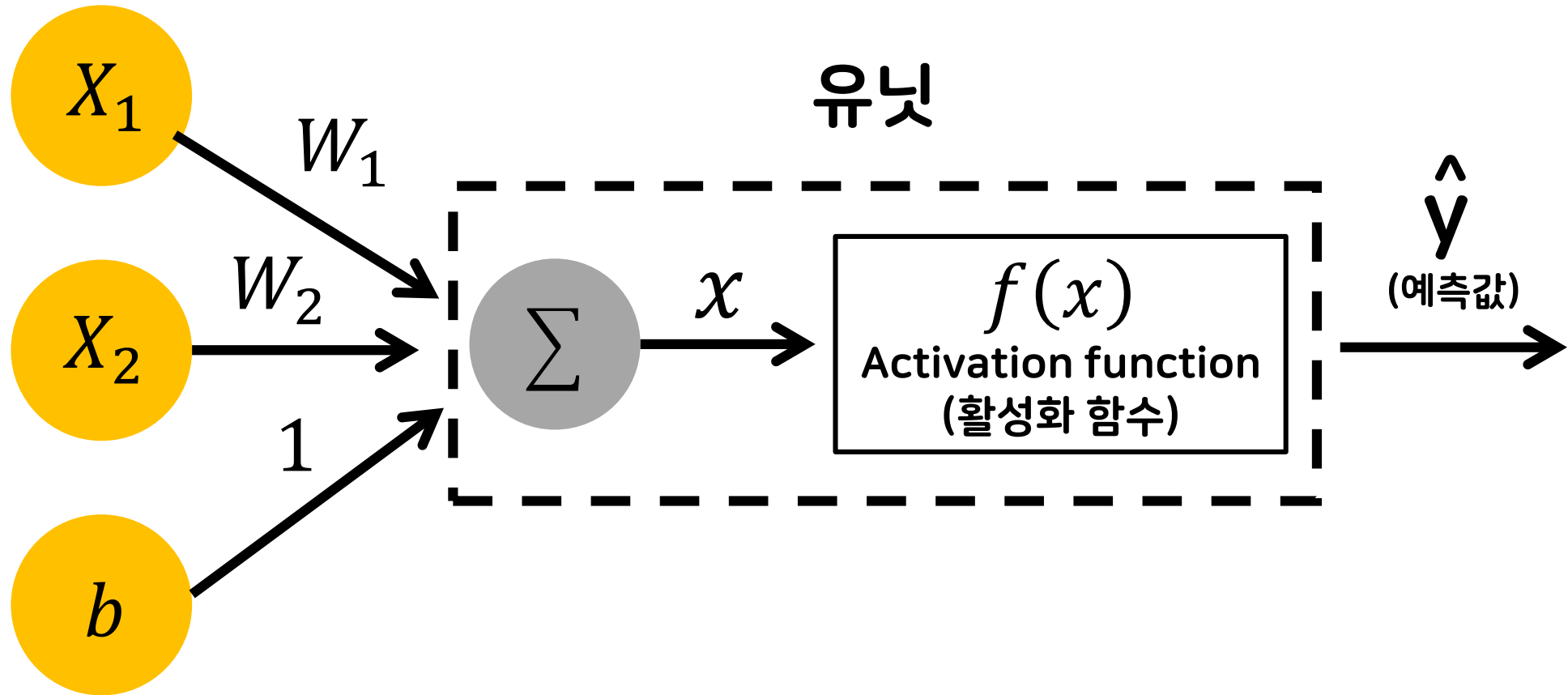


$f(x)$
Activation function
(활성화 함수)

$$y = \begin{cases} 0, & (W_1X_1 + W_2X_2 + b \leq 0) \\ 1, & (W_1X_1 + W_2X_2 + b > 0) \end{cases}$$

W_1, W_2 : 가중치 (weight) - 각 입력 신호가 결과에 주는 영향력을 조절하는 매개변수

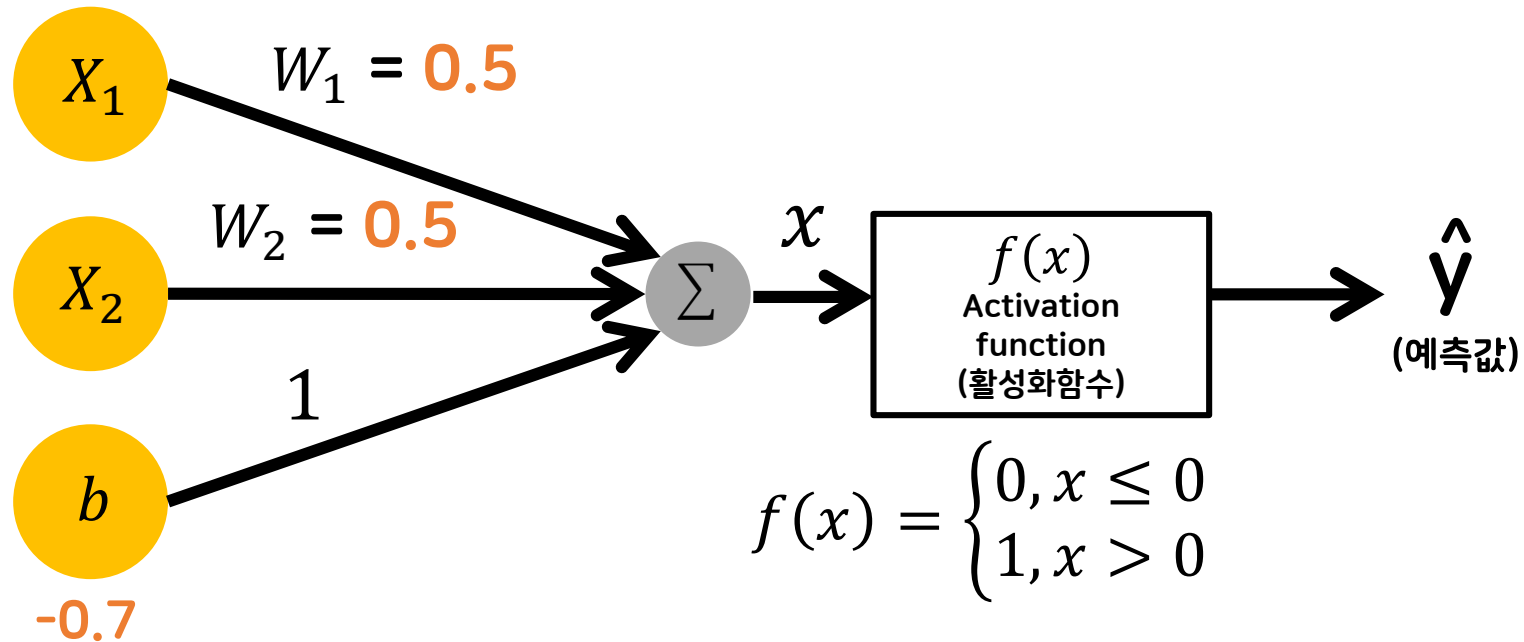
b : 편향 (bias) - 뉴런이 얼마나 쉽게 활성화하느냐를 조절하는 매개변수



AND 게이트 퍼셉트론 만들기

x1	x2	AND
0	0	0
0	1	0
1	0	0
1	1	1

인공신경망 - 퍼셉트론(Perceptron)

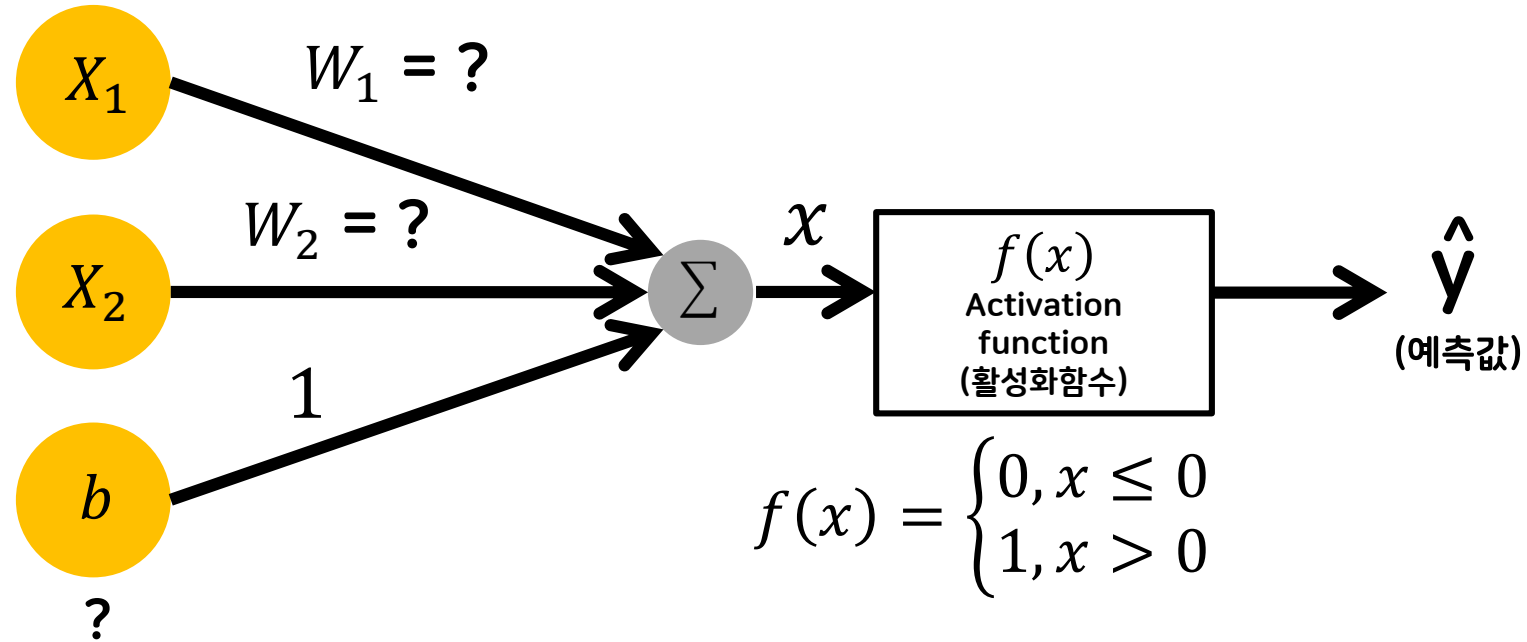


X1	X2	AND	x	y
0	0	0	$(0 \times 0.5) + (0 \times 0.5) + (1 \times -0.7) = -0.7$	class 0
0	1	0	$(0 \times 0.5) + (1 \times 0.5) + (1 \times -0.7) = -0.2$	class 0
1	0	0	$(1 \times 0.5) + (0 \times 0.5) + (1 \times -0.7) = -0.2$	class 0
1	1	1	$(1 \times 0.5) + (1 \times 0.5) + (1 \times -0.7) = 0.3$	class 1

OR 게이트 퍼셉트론 만들기

x1	x2	OR
0	0	0
0	1	1
1	0	1
1	1	1

인공신경망 - 퍼셉트론(Perceptron)

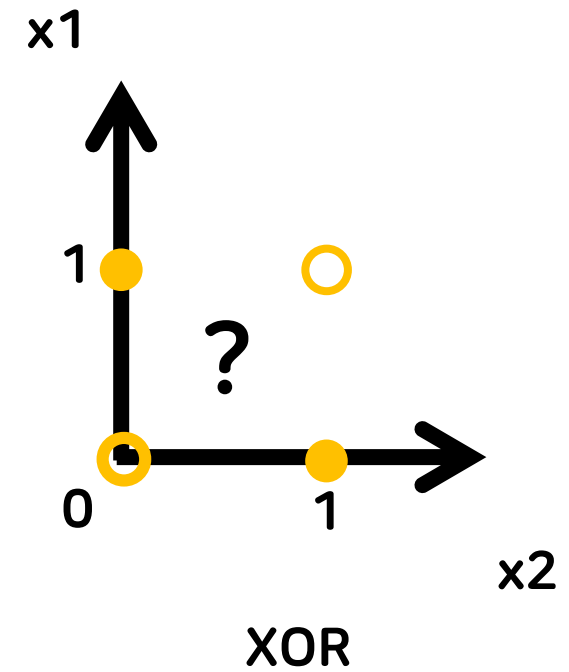
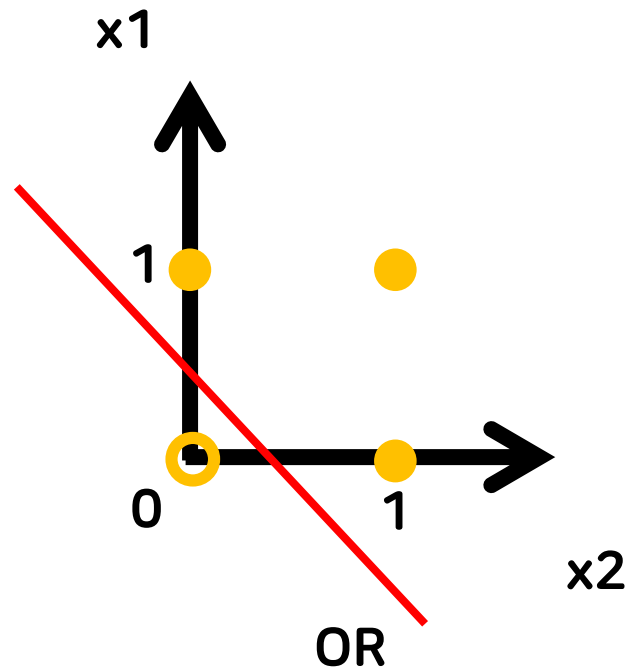
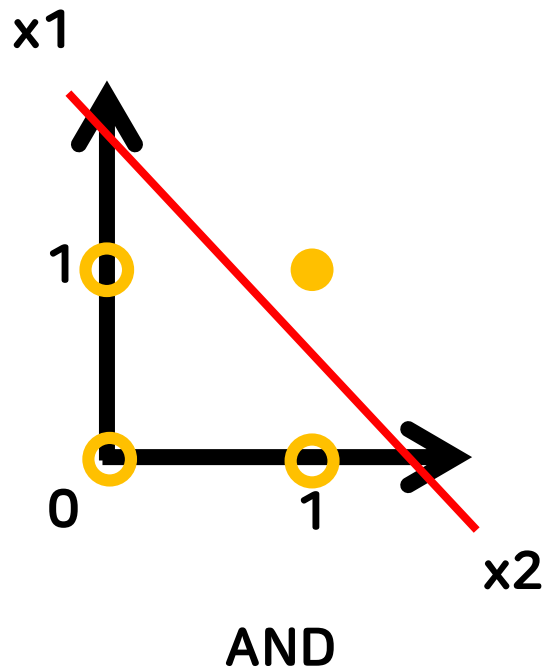


X1	X2	OR	x	y
0	0	0	?	class 0
0	1	1	?	class 1
1	0	1	?	class 1
1	1	1	?	class 1

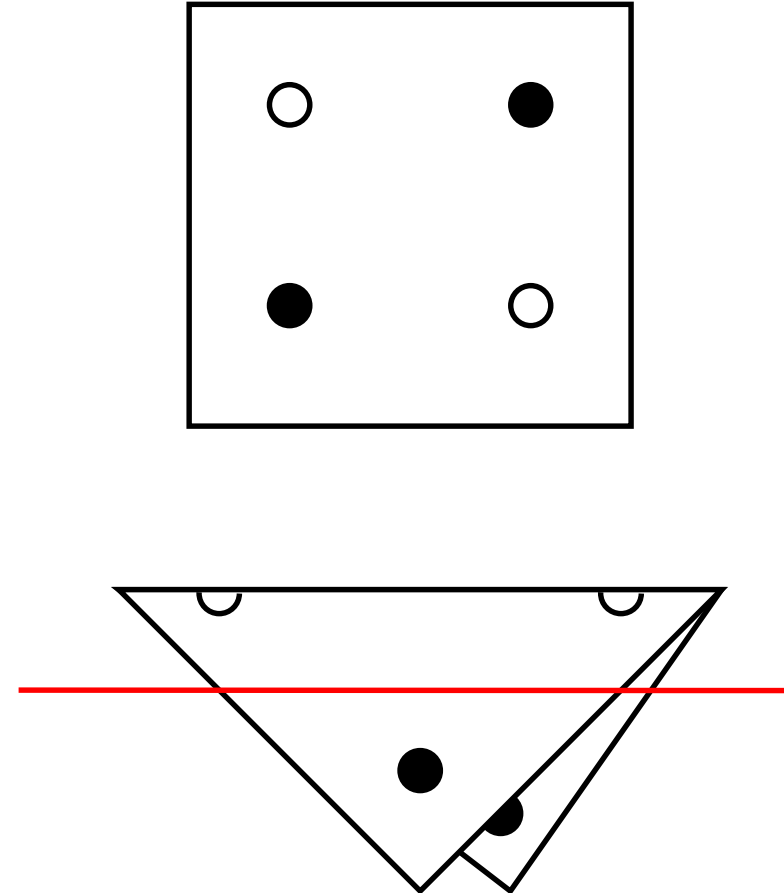
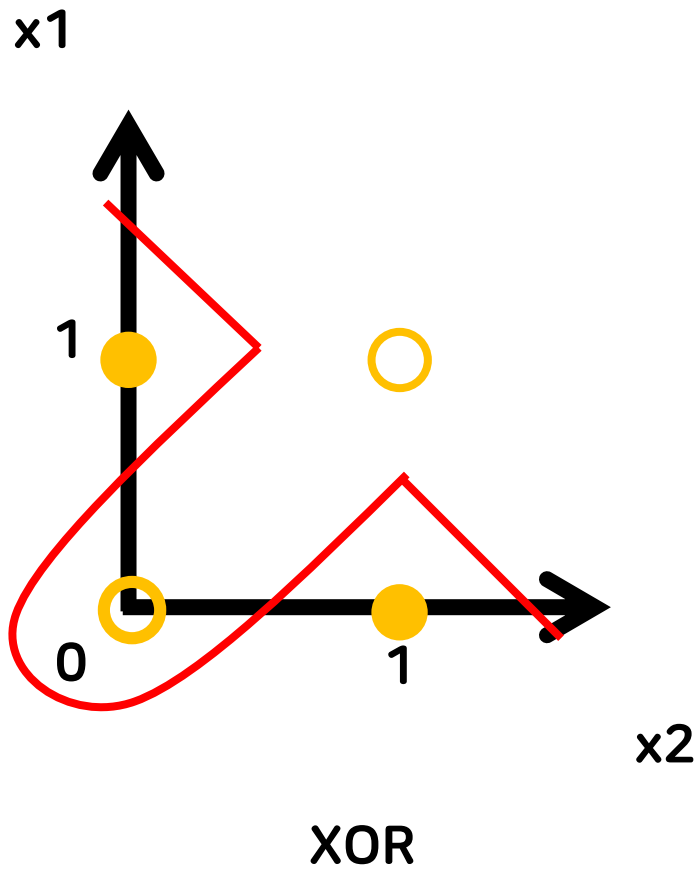
AND,OR는 해결이 가능하지만
간단한 XOR 문제를 해결 할 수 없다.

OR 게이트 퍼셉트론 만들기

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



한 선으로 분류하려면 ?

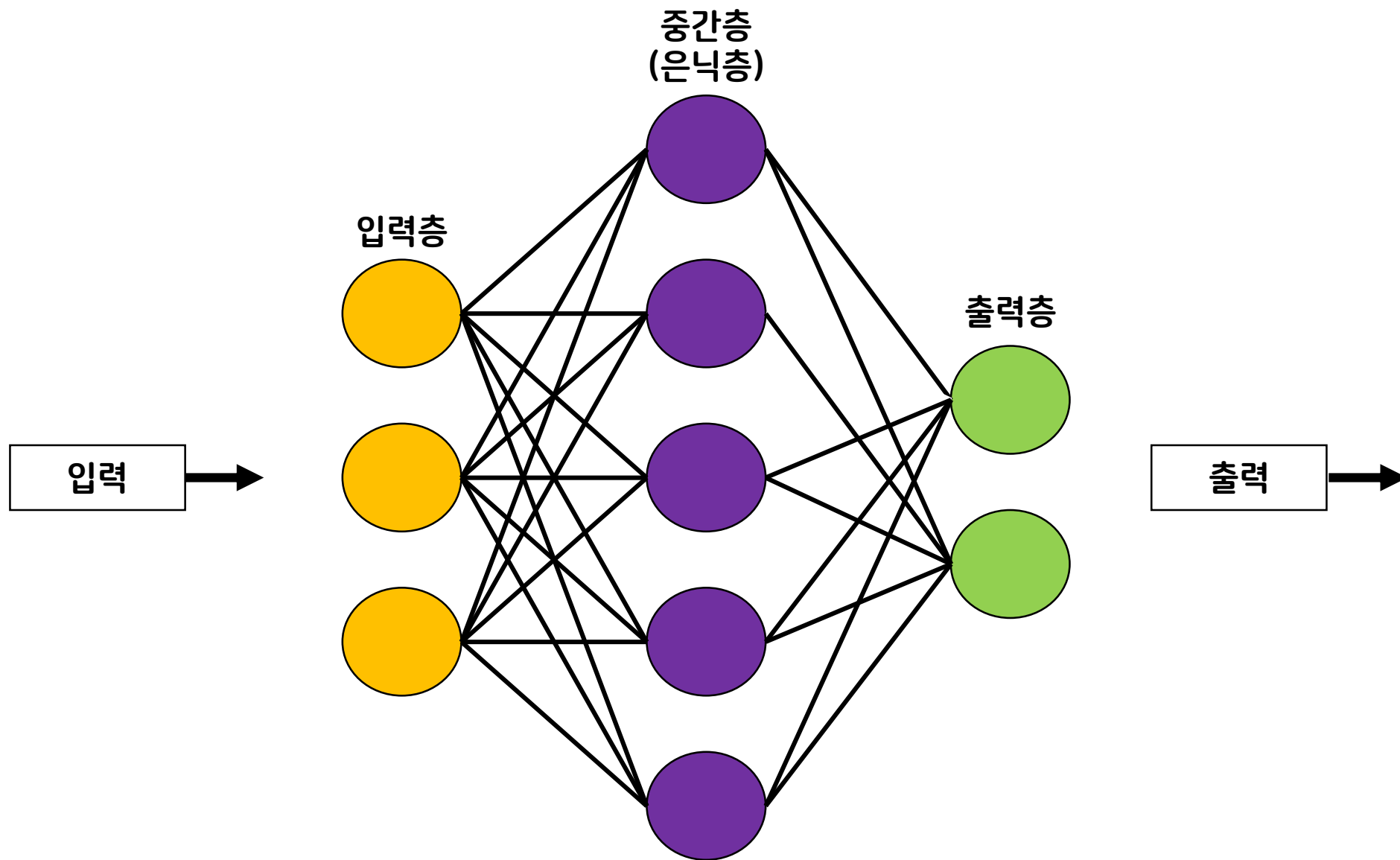


다층 퍼셉트론(Multilayer Perceptron)


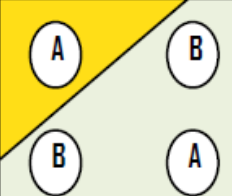
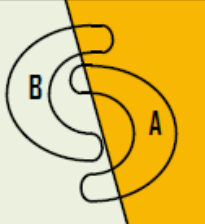
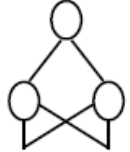
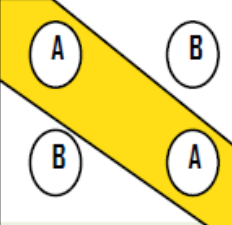
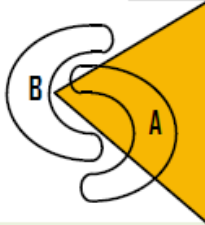
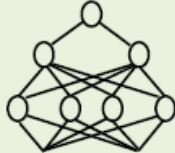
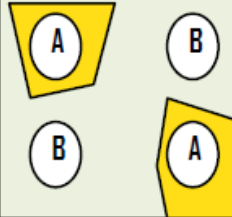

퍼셉트론을 여러 개의 층으로 구성하여 만든 신경망

- 비선형 데이터를 분리 할 수 있다.
- 학습시간이 오래 걸린다.
- 가중치 파라미터가 많아 과적합되기 쉽다.
- 가중치 초기 값에 민감하며 지역 최적점에 빠지기 쉽다.

다층 퍼셉트론(Multilayer Perceptron)



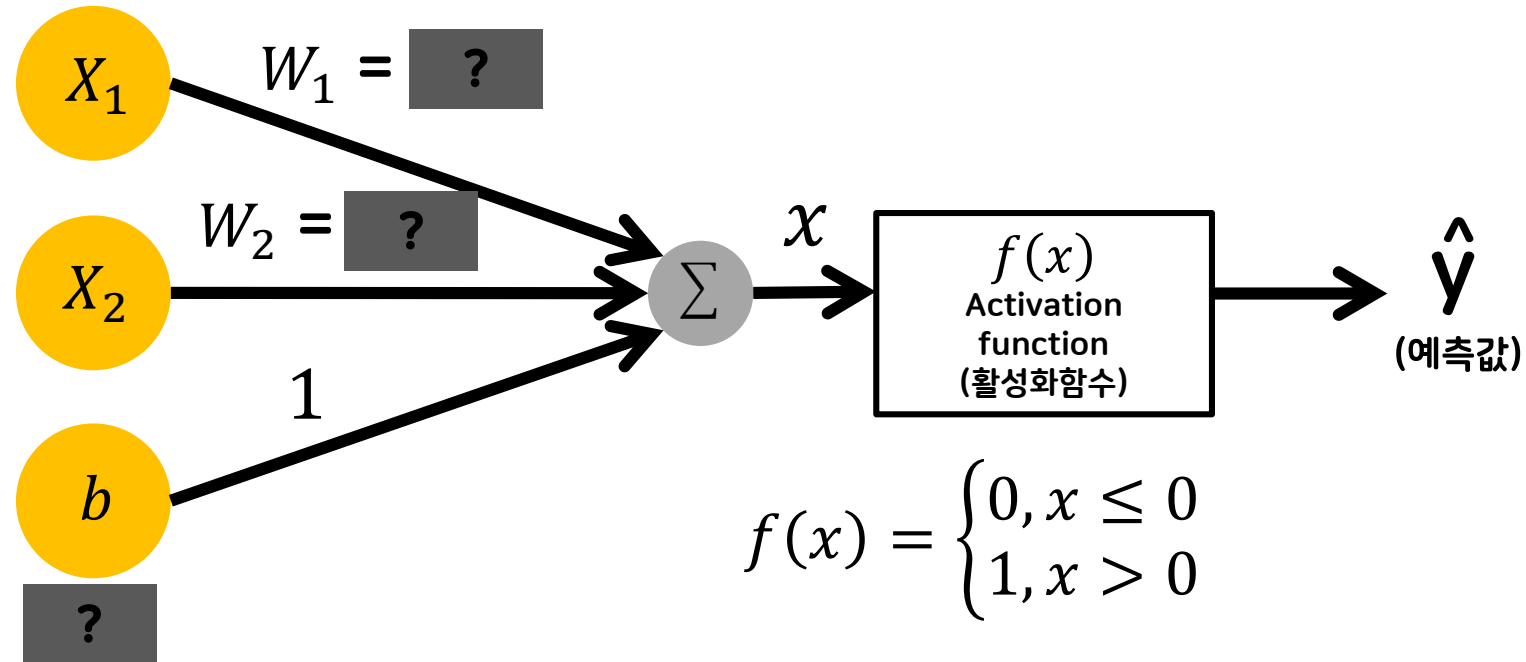
다층 퍼셉트론(Multilayer Perceptron)

구조	결정 영역 형태	Exclusive-OR 문제	얹힌 결정 영역을 갖는 클래스들
<p>단층</p> 	<p>초평면에 의해 나뉘어지는 반평면 (Hyper plane)</p>		
<p>두 개층</p> 	<p>불록한 모양 또는 닫힌 영역</p>		
<p>세 개층</p> 	<p>임의의 형태 (노드들의 개수에 따라 복잡도가 결정됨)</p>		

NAND 게이트 퍼셉트론 만들기

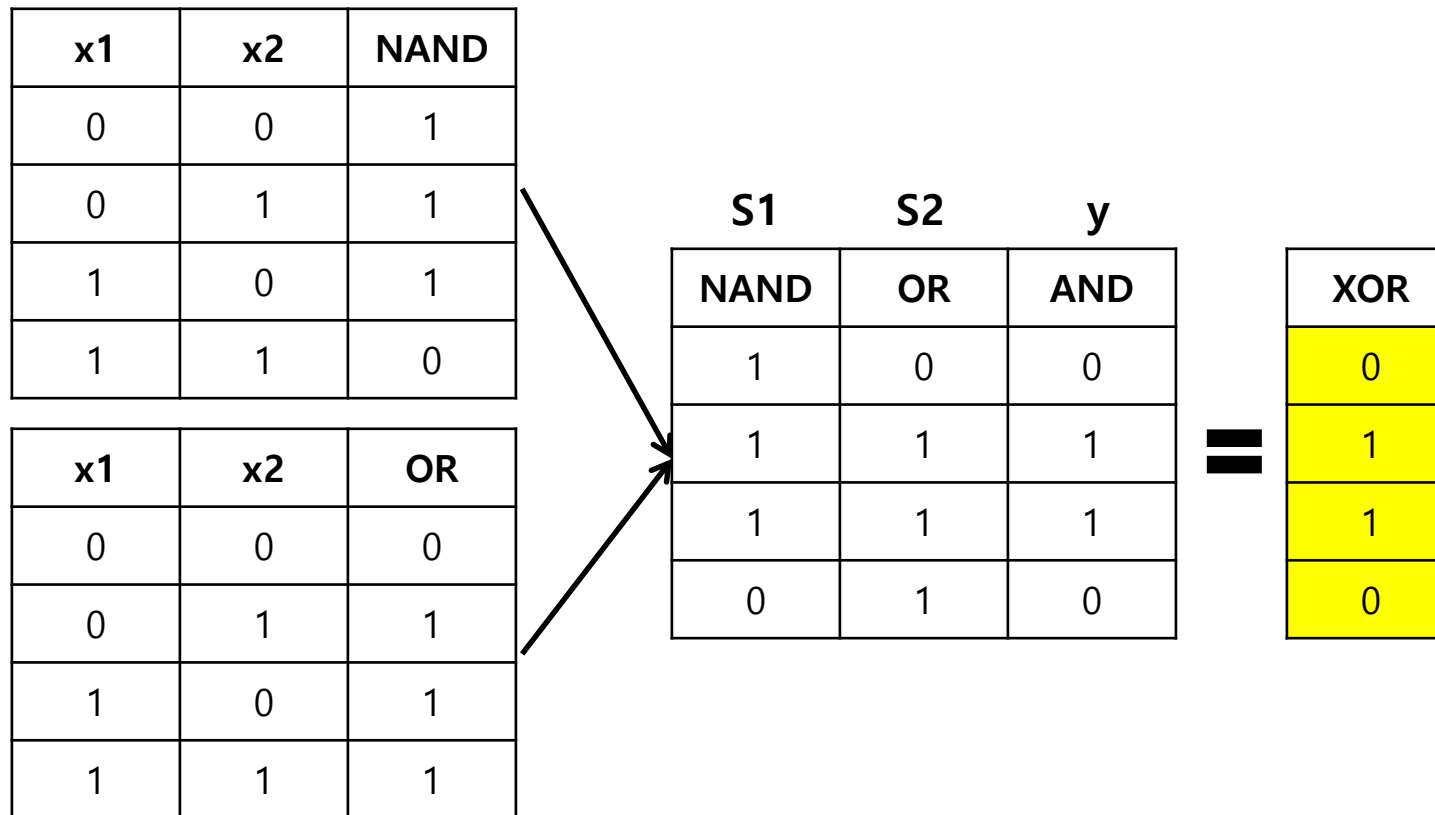
x1	x2	NAND
0	0	1
0	1	1
1	0	1
1	1	0

다층 퍼셉트론(Multilayer Perceptron)



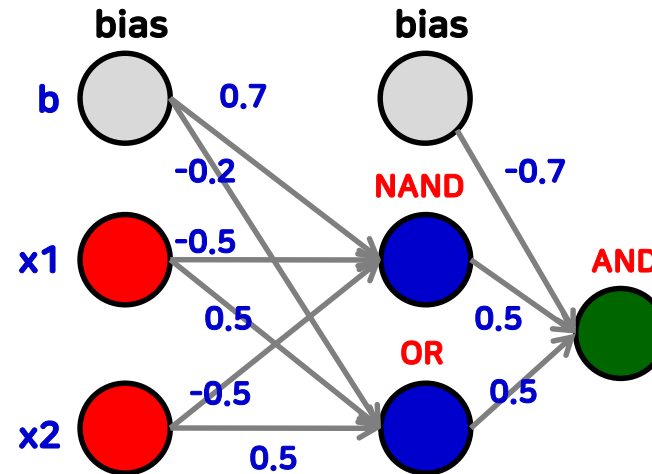
X1	X2	NAND	x	y
0	0	1	?	class 1
0	1	1	?	class 1
1	0	1	?	class 1
1	1	0	?	class 0

다층 퍼셉트론 활용 XOR 문제 풀기

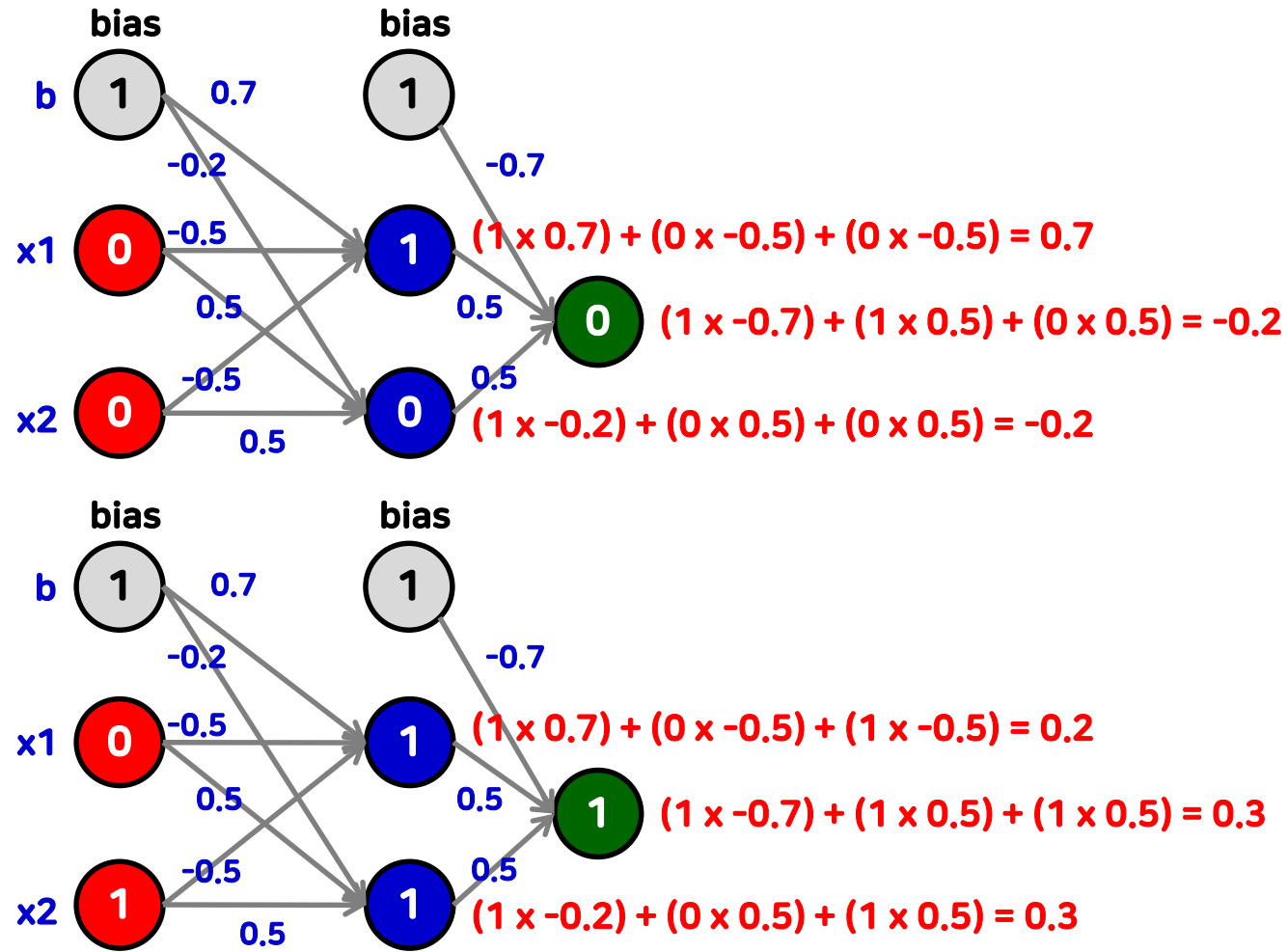


다층 퍼셉트론(Multilayer Perceptron)

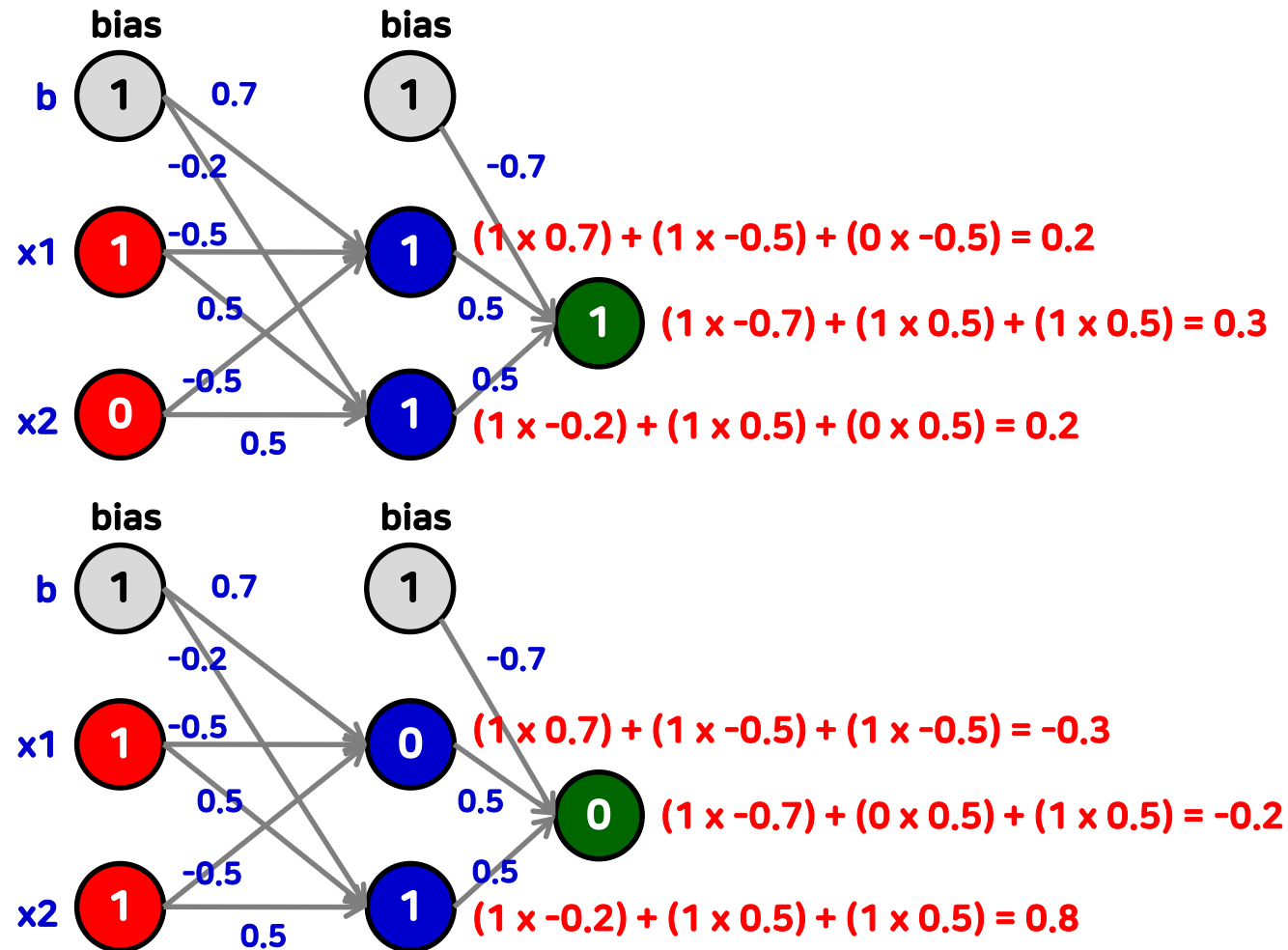
x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0



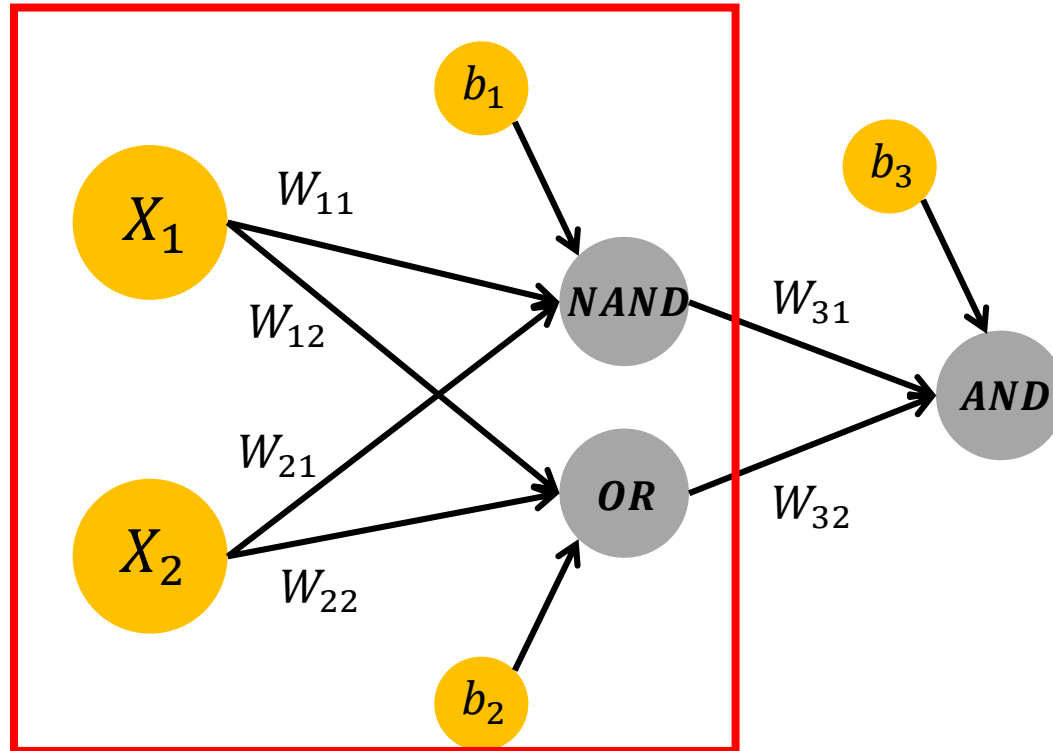
다층 퍼셉트론(Multilayer Perceptron)



다층 퍼셉트론(Multilayer Perceptron)

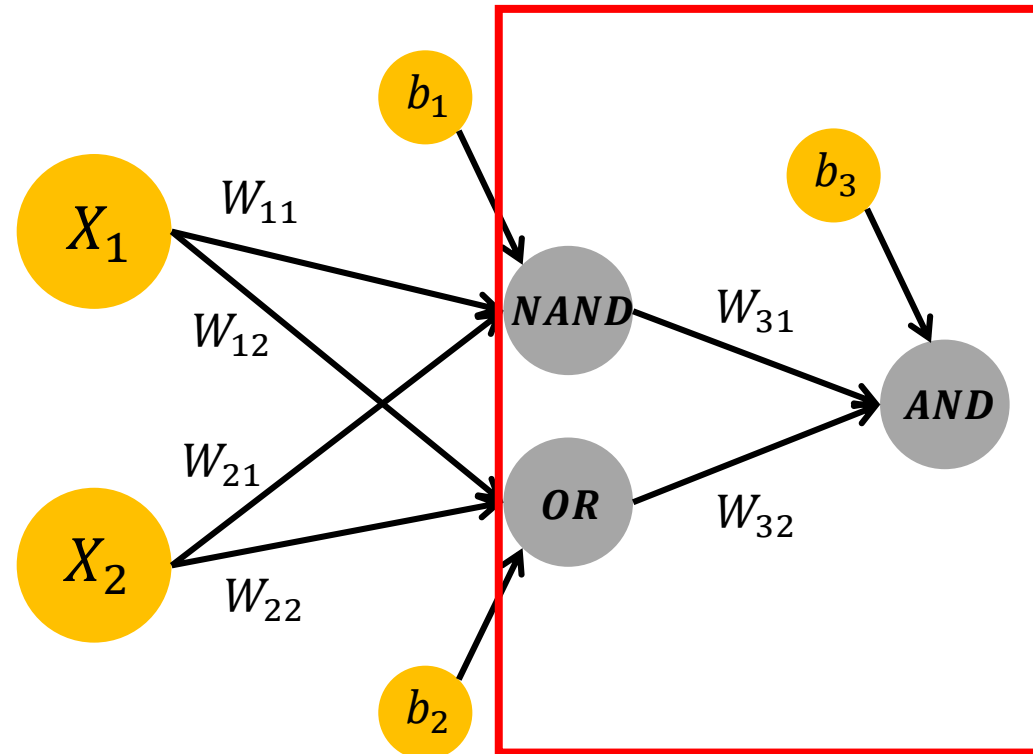


다층 퍼셉트론 활용 XOR 문제 풀기(행렬곱)



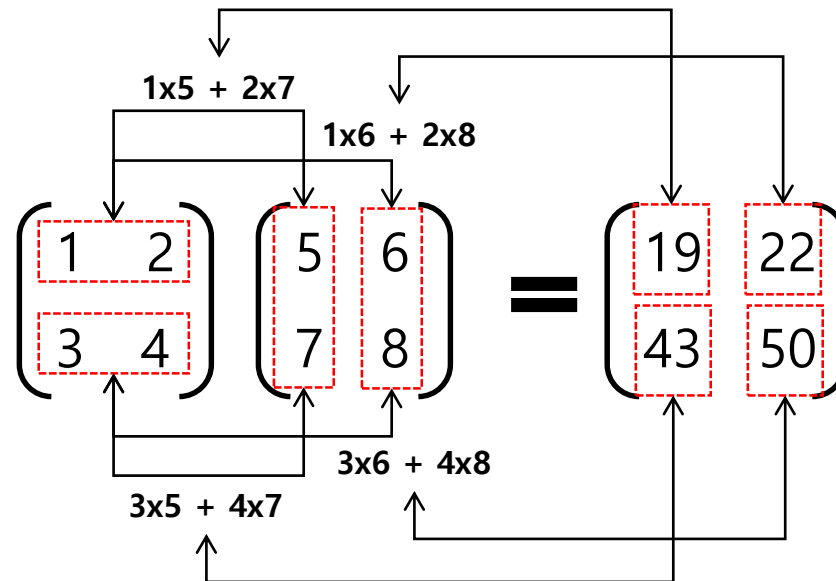
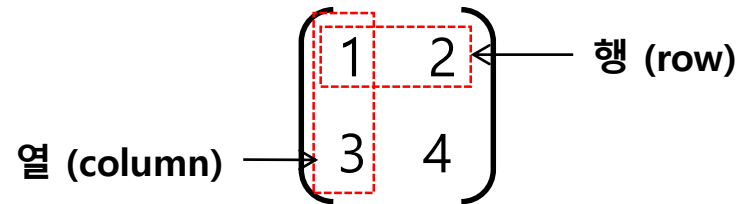
$$\begin{bmatrix} X_1 & X_2 \end{bmatrix} \times \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix} = \begin{bmatrix} S_1 & S_2 \end{bmatrix}$$

다층 퍼셉트론 활용 XOR 문제 풀기(행렬곱)



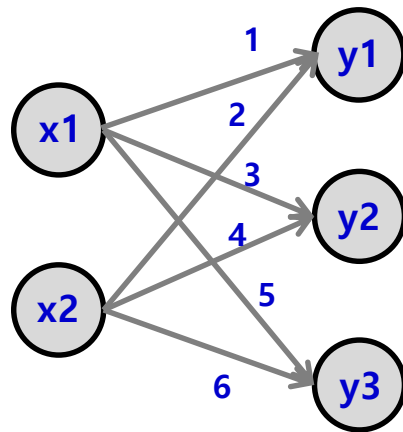
$$[S_1 \quad S_2] \times \begin{bmatrix} W_{31} \\ W_{32} \end{bmatrix} + [b_3] = Y$$

다차원 배열 연산

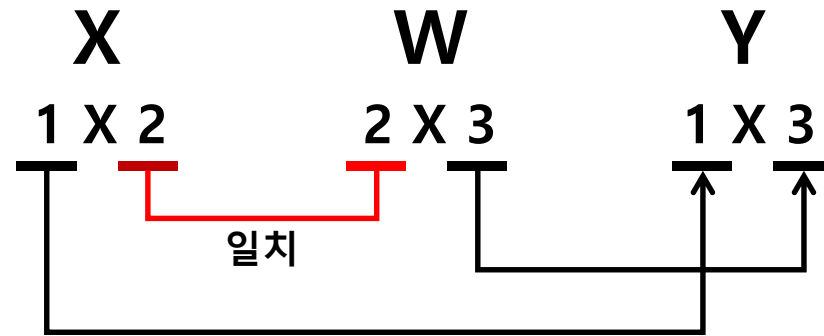


다층 퍼셉트론(Multilayer Perceptron)

신경망에서 행렬의 곱



$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$$

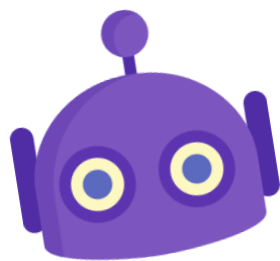


$$\begin{bmatrix} 1 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix} = \begin{bmatrix} 13 & 16 \end{bmatrix}$$

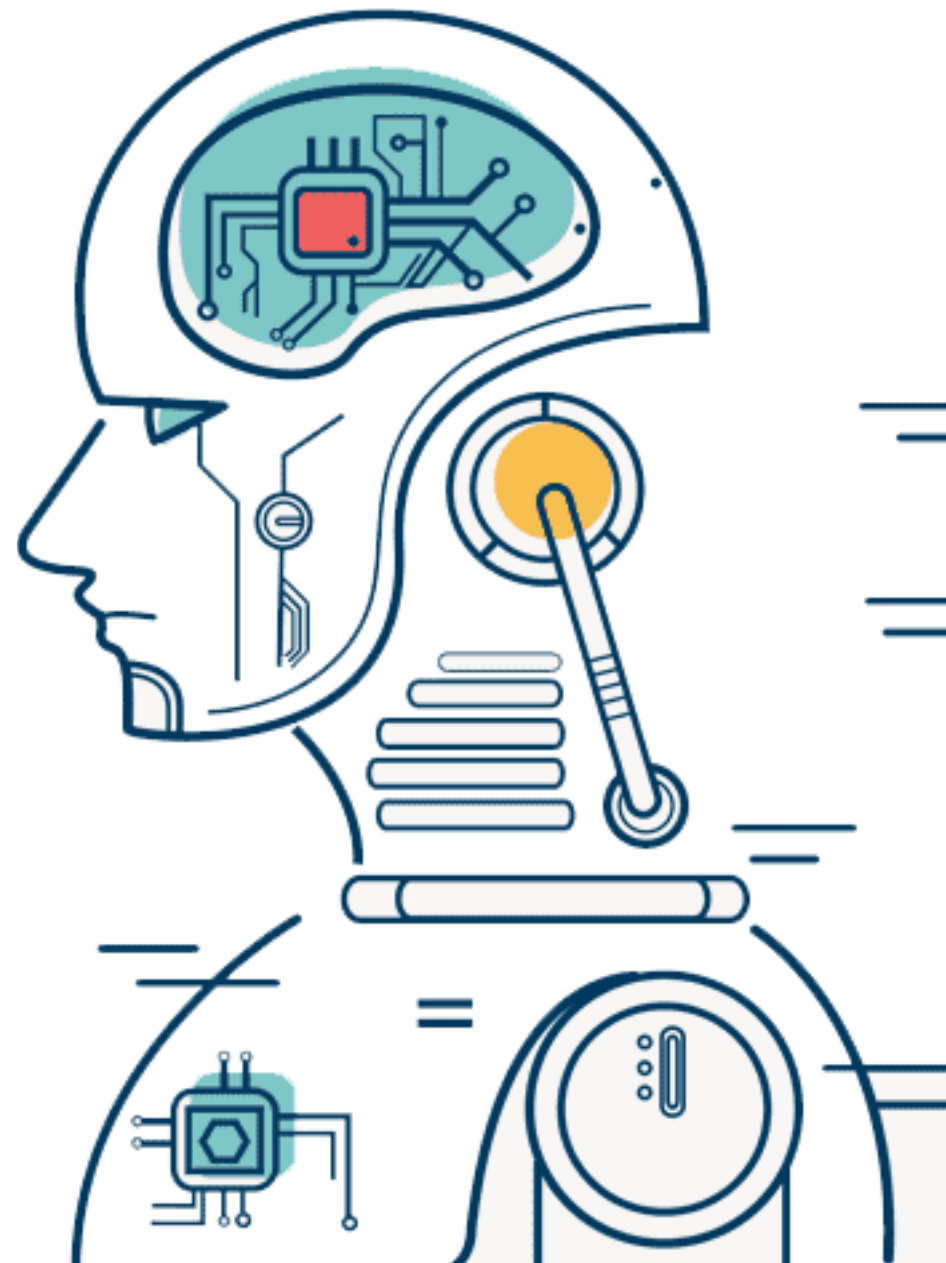
$$\begin{bmatrix} 13 & 16 \end{bmatrix} + \begin{bmatrix} 3 & 4 \end{bmatrix} = \begin{bmatrix} 16 & 20 \end{bmatrix}$$

다층 퍼셉트론의 동작





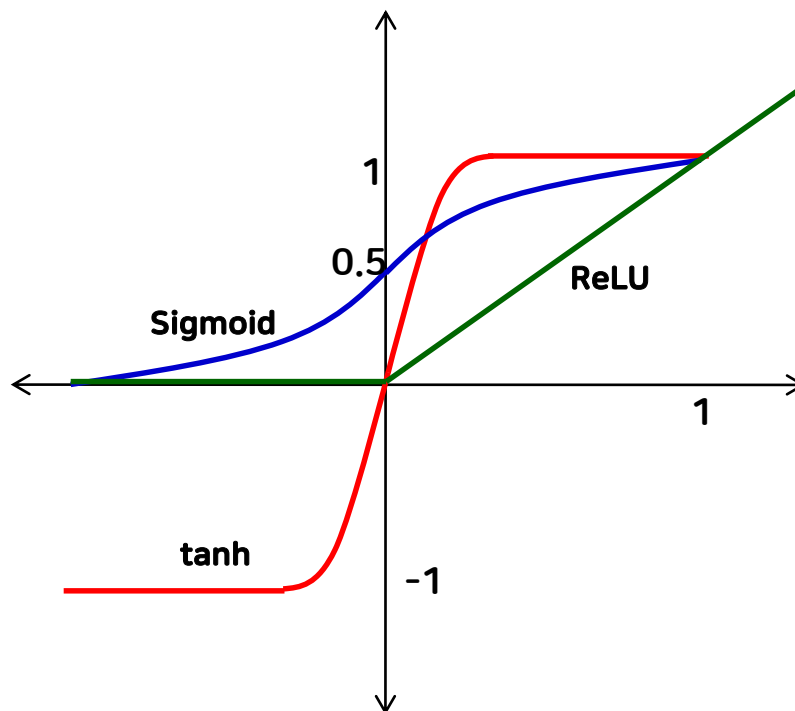
활성화 함수, 오차



- **활성화 함수를 이해 할 수 있다.**
- **오차 계산 방법을 이해할 수 있다.**

- 신경망은 (선형회귀와 달리) 한 계층의 신호를 다음 계층으로 그대로 전달하지 않고 비선형적인 활성화 함수를 거친 후에 전달
- 이렇게 하는 이유는 생물학적인 신경망을 모방한 것
 - 약한 신호는 전달하지 않고 어느 이상의 신호도 전달하지 않는 "S"자 형 곡선과 같이 "비선형적"인 반응을 한다고 생각
- 실제로 비선형의 활성화 함수를 도입한 신경망이 잘 동작

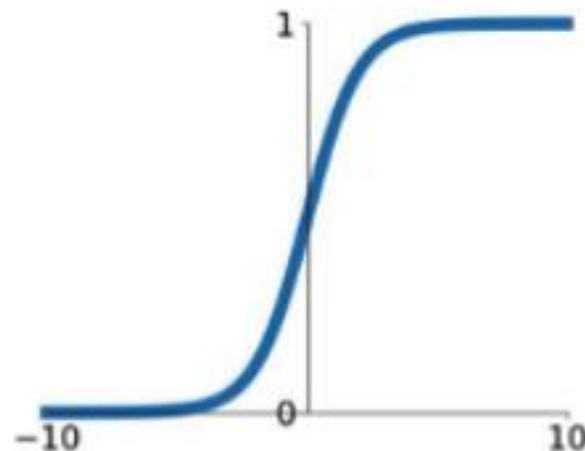
- **활성화 함수 (Activation Function)** : 뉴런에서 입력 신호가 일정 크기 이상(임계값)일 때만 신호를 전달하는 메커니즘을 모방한 함수 → 계단함수, 시그모이드 (sigmoid), tanh, ReLU (Rectified Unit)



Sigmoid 함수

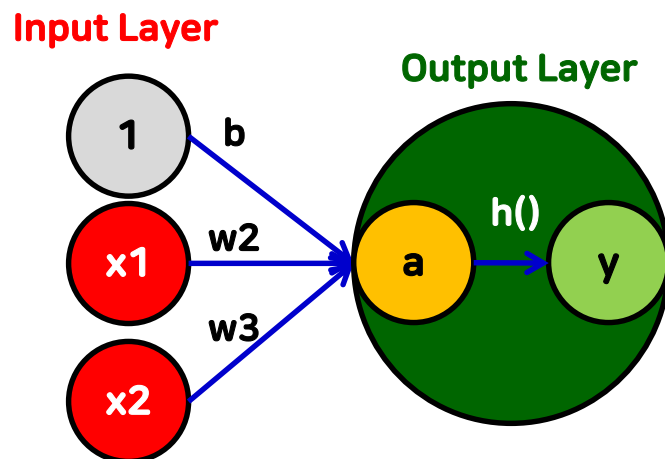
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



sigmoid 함수를 구현해보자

- 활성화 과정 : 가중치 신호를 조합한 결과가 a라는 노드가 되고 활성화 함수 h()를 통과하여 y라는 노드로 변환되는 과정



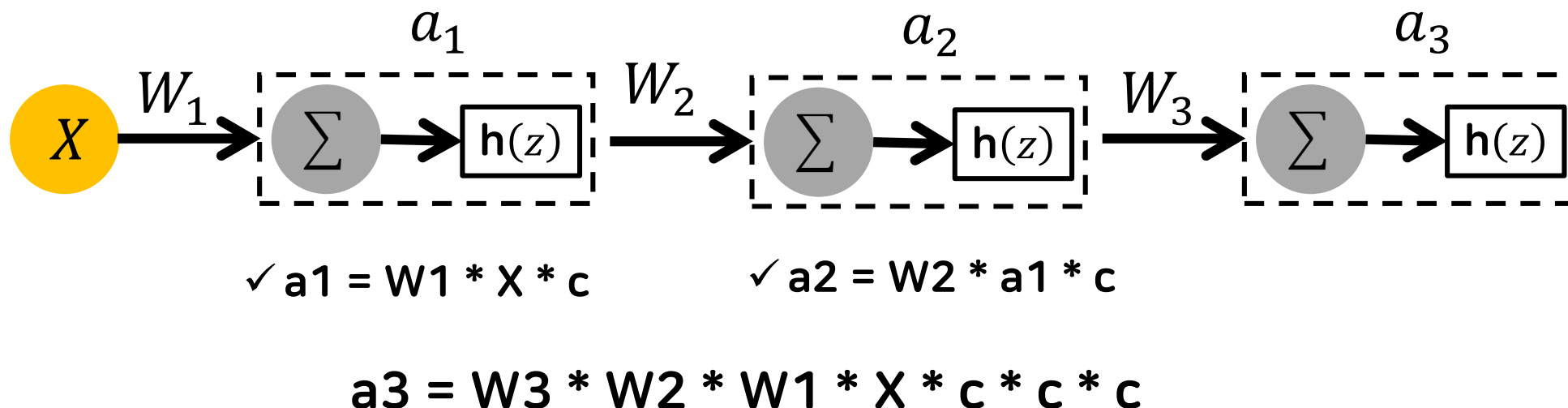
$$y = h(w_1x_1 + w_2x_2 + b)$$

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

- **편향은 뉴런이 활성화되는 기준값**이라고 할 수 있음 → **활성화 함수**
 - 만약 편향이 -0.1이라면 입력신호의 가중치를 곱한 값들의 합이 0.1을 초과할 때만 뉴런이 활성화
 - 편향이 큰 음수라면 그만큼 뉴런이 활성화가 되기 어려운 환경이 됨

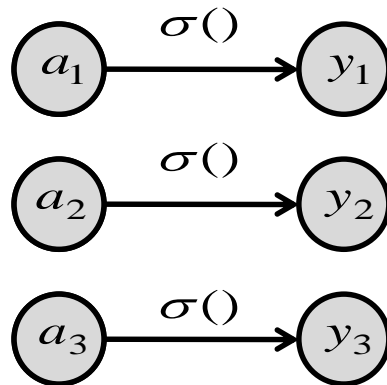
활성화 함수로 비선형 함수를 사용하는 이유

- 계단 함수(step), 시그모이드 함수(sigmoid) 등 → 비선형 함수
- 활성화 함수로 선형함수 $h(z) = cz$ 를 사용하면 중간층(은닉층)을 여러 개 구성한 효과를 살릴 수 없다.

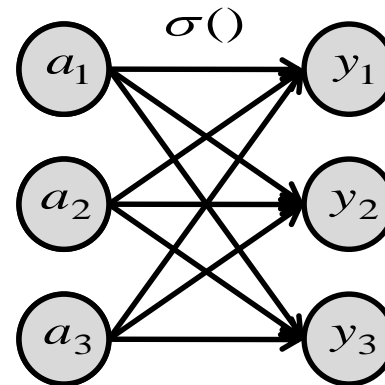


출력층의 활성화 함수

- 일반적으로 출력층의 활성화 함수로는 회귀에는 **항등 함수**를 분류에는 **시그모이드 / 소프트맥스 함수**를 사용
- **항등함수 (identify function)** : 입력을 그대로 출력하는 함수
- **softmax function** : 출력층의 각 뉴런이 모든 입력 신호의 영향을 받는 함수



항등함수



소프트맥스 함수

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

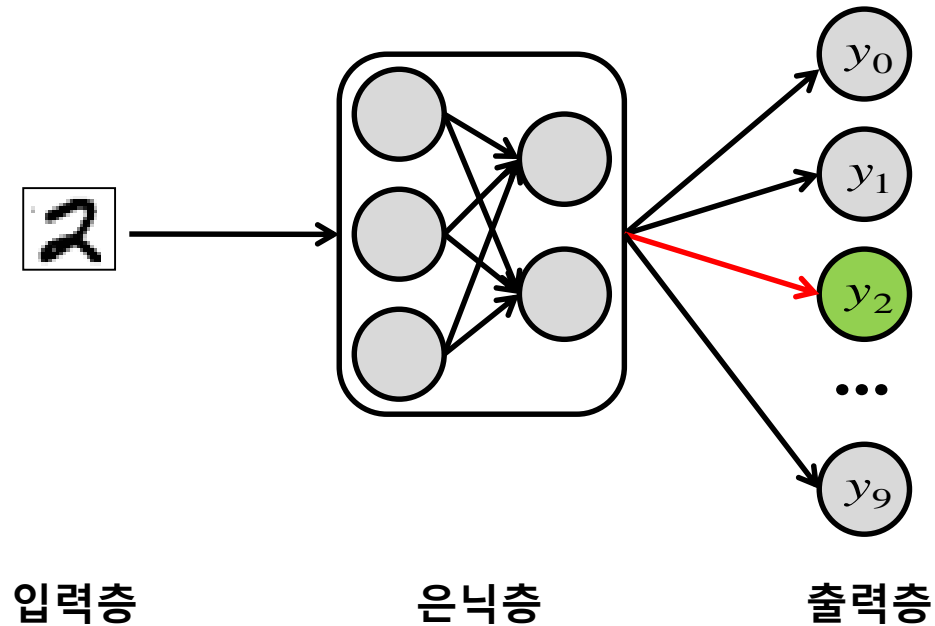
소프트맥스(softmax) 함수

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

softmax 함수를 구현해보자

출력층의 뉴런 수 설정

- 출력층의 뉴런 수는 **풀려는 문제에 맞게** 적절하게 설정해야 함
- 숫자 0부터 9까지를 분류하고 싶다면 출력층의 개수는 10개로 설정



출력층의 활성화 함수

유형	출력층 활성화 함수	오차함수
회귀	항등 함수	제곱오차
이진 분류	로지스틱 함수	교차 엔트로피
다클래스 분류	소프트맥스 함수	교차 엔트로피

Keras

linear	mean_squared_error
sigmoid	binary_crossentropy
softmax	categorical_crossentropy

Keras 활용 XOR 문제풀기

iris 데이터 신경망으로 풀기 (분류)

- 손실함수 (Loss function) : 최적의 매개변수 값을 탐색하기 위한 기준 지표 → **평균제곱오차 (MSE)**와 **교차 엔트로피 오차 (CEE)**를 주로 사용
- 정확도를 사용하지 않고 손실함수를 사용하는 이유 → **정확도의 미분값이 대부분 0이 되기때문**
- 신경망 학습에서 가중치, 편향을 탐색 → 손실 함수를 최대한 작게 하는 매개변수를 찾음 → 매개변수의 기울기 (미분)을 계산하고 기울기를 이용하여 매개변수를 갱신
- **손실함수의 미분** : 가중치 매개변수의 값을 아주 조금 변화시켰을 때 손실 함수가 어떻게 변하는지 의미
 - 미분 값이 음수이면 가중치 매개변수를 양의 방향으로 변화시켜 손실 함수의 값을 감소
 - 미분 값이 양수이면 가중치 매개변수를 음의 방향으로 변화시켜 손실 함수의 값을 감소
 - 미분 값이 0이면 가중치 매개변수를 어느쪽으로 움직여도 손실 함수의 값은 변하지 않음
→ 계단함수를 손실함수로 사용하지 않는 이유

평균 제곱 계열	mean_squared_error	평균제곱오차 $\text{mean}(\text{square}(y_t - y_0))$
	mean_absolute_error	평균절대 오차 $\text{mean}(\text{abs}(y_t - y_0))$
	mean_absolute_percentage_error	평균 절대 백분율 오차 $\text{mean}(\text{abs}(y_t - y_0) / \text{abs}(y_t))$
	mean_squared_logarithmic_error	평균제곱 로그 오차 $\text{mean}(\text{square}(\log(y_0)+1) - (\log(y_t)+1)))$
교차 엔트로피 계열	categorical_crossentropy	범주형 교차 엔트로피
	binary_crossentropy	이항 교차 엔트로피

평균 제곱 오차

- 평균제곱오차 (Mean Squared Error) → 회귀에서 주로 사용

$$E = \frac{1}{2} \sum_{k=1}^n (y_k - t_k)^2$$

y_k : 신경망의 출력 (신경망이 추정한 값)

t_k : 정답 레이블

k : 데이터의 차원 수

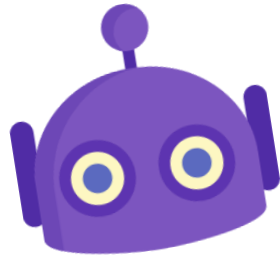
교차 엔트로피 오차

- **교차 엔트로피 오차 (Cross Entropy Error : CEE)** → 분류에서 주로 사용
- 레이블이 one-hot 인코딩인 경우에만 사용 가능

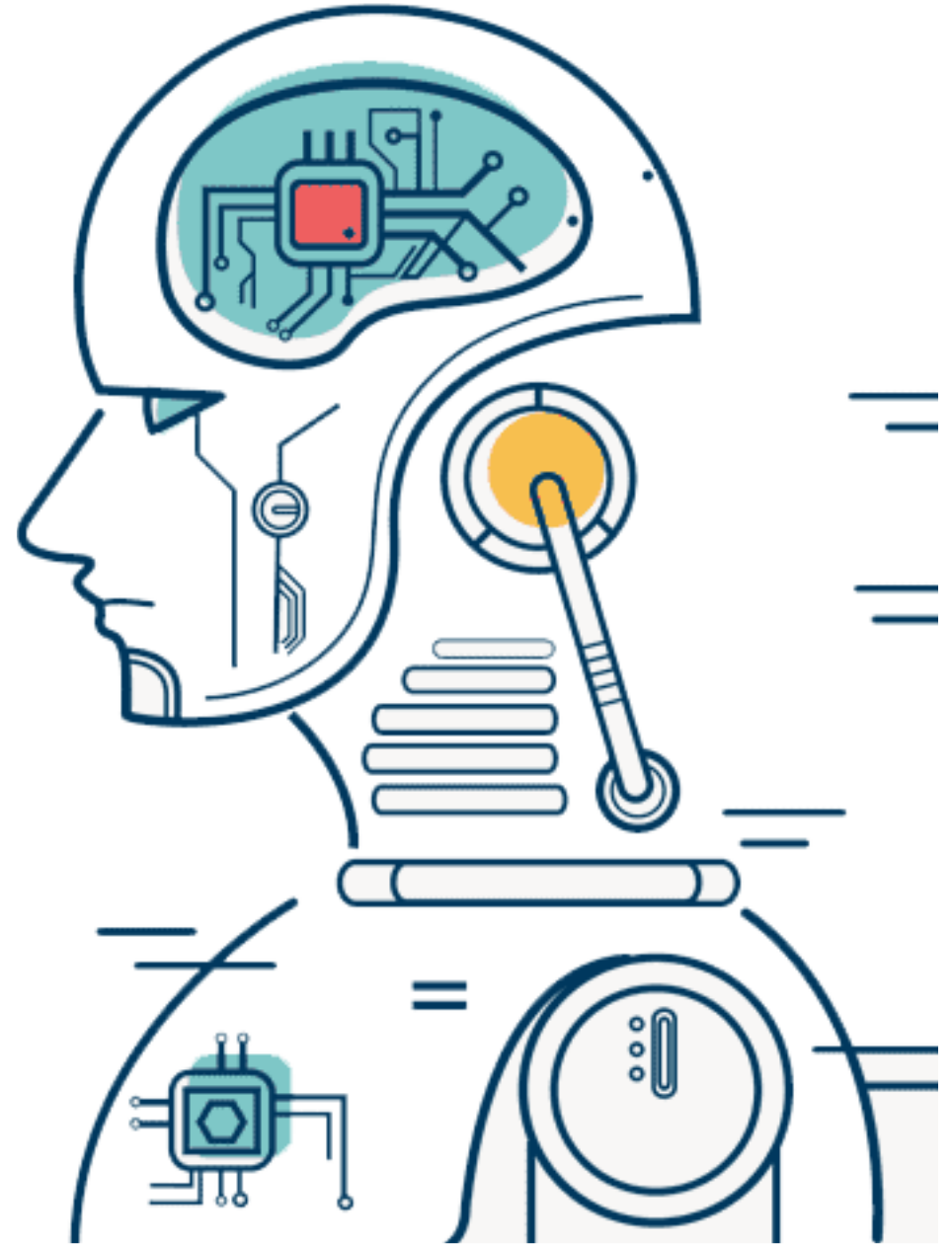
$$E = - \sum_{k=1}^n t_k \log y_k$$

y_k : 신경망의 출력 (신경망이 추정한 값)
 t_k : 정답 레이블
 k : 데이터의 차원 수

- t_k 가 1일 때 (정답일 때)만 자연로그의 합을 계산하는 식이 됨.

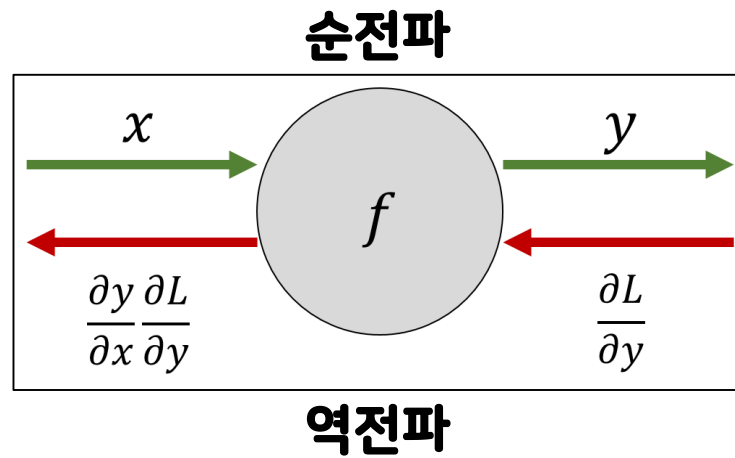


오차역전파



- 오차역전파의 개념을 이해 할 수 있다.

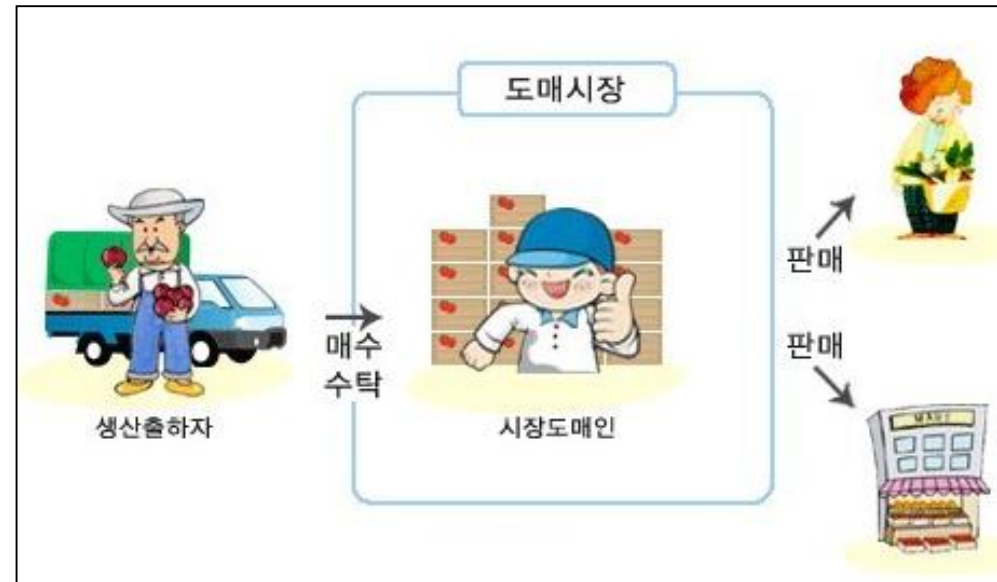
- **순전파** : 입력 데이터를 입력층에서부터 출력층까지 전파시키면서 출력값을 찾아가는 과정 → **추론**
- **역전파** : 에러를 출력층에서 입력층 쪽으로 전파시키면서 최적의 학습 결과를 찾아가는 것 → **학습**



오차 역전파 (Error Backpropagation)



오차 역전파 (Error Backpropagation)



유통 시스템의 문제점을 찾으려면 어떤 것이 나올까요 ?

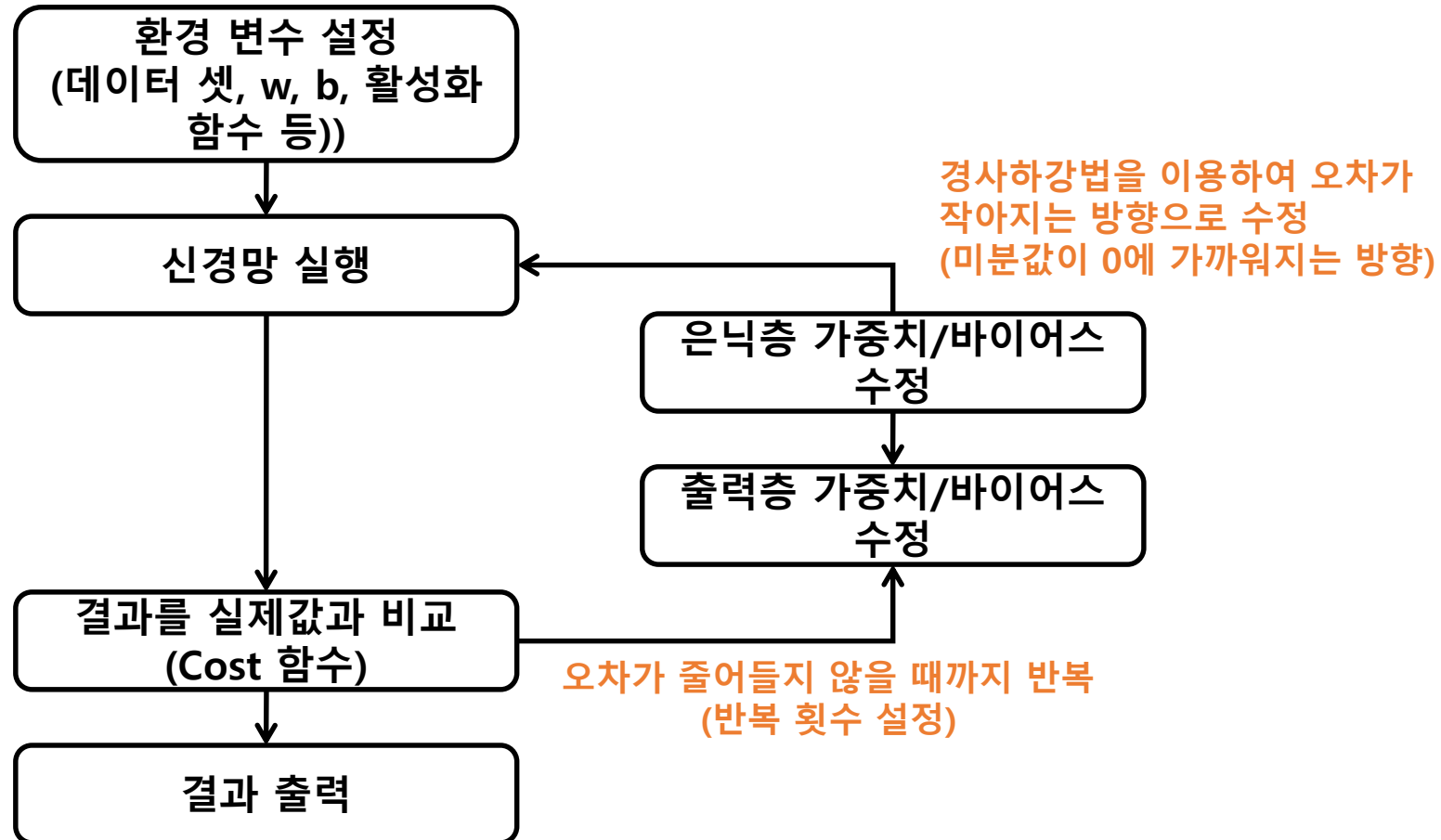
진행방향

진행반대방향

```

graph TD
    A[데이터 로드] --> B[데이터 세트  
(입력, 레이블)]
    A --> C[평가 데이터 세트  
(입력, 레이블)]
    B -- "feed forward" --> D[네트워크 모델]
    D --> E[손실 함수]
    E -- "모델의 출력값과 실제 레이블 간의 차이 계산" --> F[손실]
    F -- "back propagation  
가중치 업데이트" --> D
    D --> G[평가하기]
    G --> H[평가 정확도]
    D --> I[훈련된 네트워크 모델]
  
```

오차역전파 과정



오차 역전파 (Error Backpropagation)

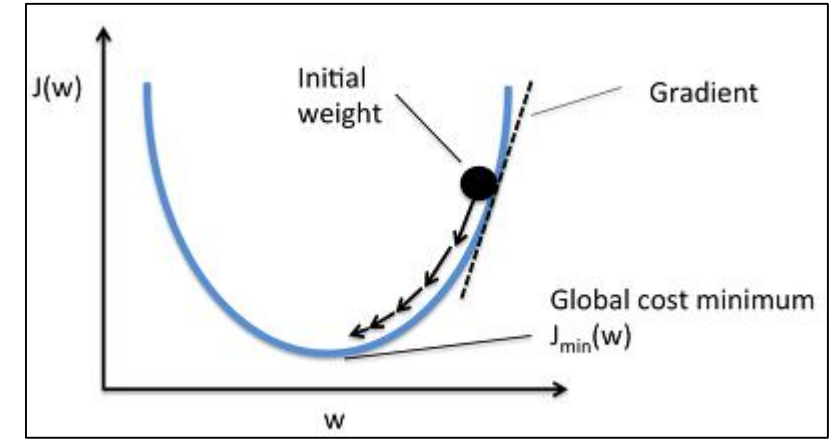
- 가중치 수정

$$\underbrace{W(t+1)}_{\text{새 가중치}} = \underbrace{Wt}_{\text{이전 가중치}} - \underbrace{\eta}_{\text{오차를 감소 하는 방향}} \underbrace{\frac{\partial E}{\partial W}}_{\text{이전 가중치에 대한 오차의 기울기 (미분값)}}$$

오차를 감소 하는 방향

학습률

이전 가중치에 대한 오차의 기울기 (미분값)



- 편미분 (∂) : 여러 개의 변수 중 하나에 대해서만 미분하고 다른 변수 값은 상수로 취급

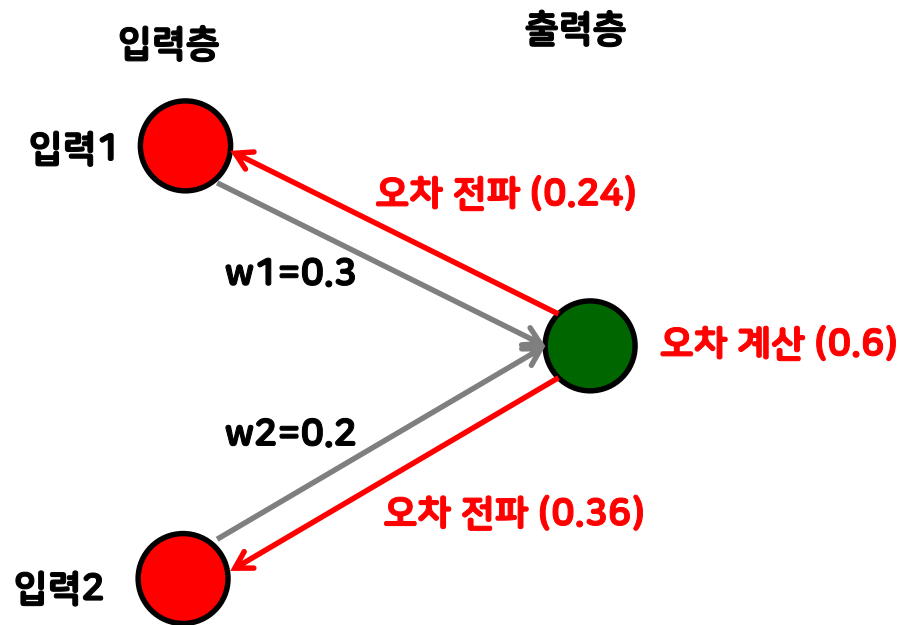
$$y = 2a + 3b$$

$$\frac{\partial y}{\partial a} = 2$$

$$\frac{\partial y}{\partial b} = 3$$

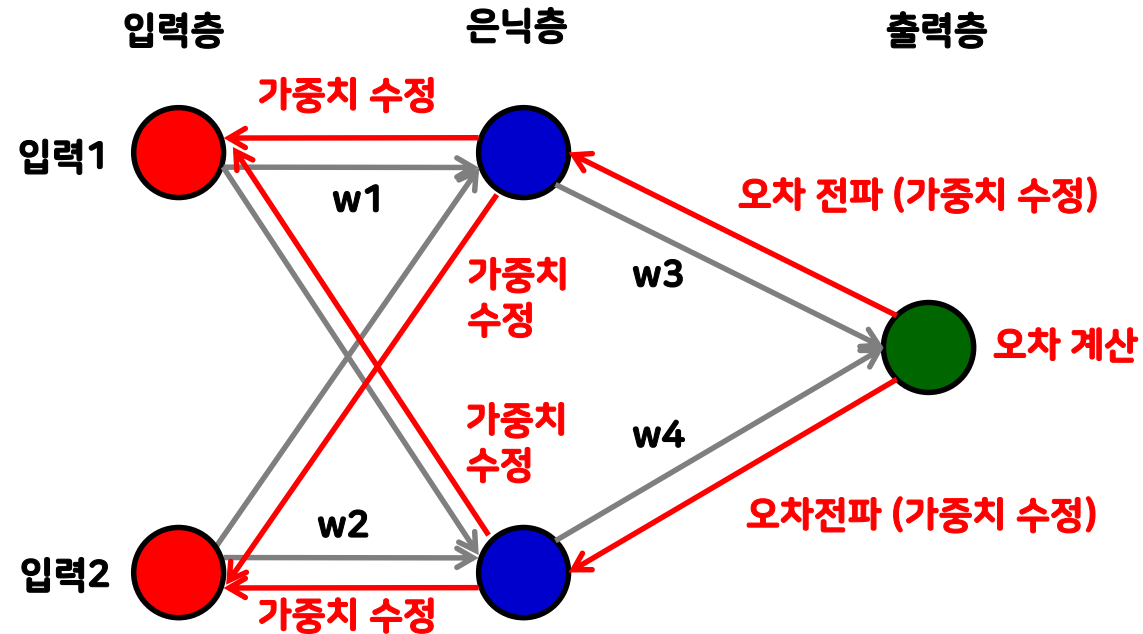
오차 역전파 (Error Backpropagation)

- 출력층에서 오류가 0.6이 발생하였다면 입력1 노드로 0.36을 입력2 노드로 0.24의 에러를 전파시켜서 각각이 가중치를 갱신 → 오차 역전파

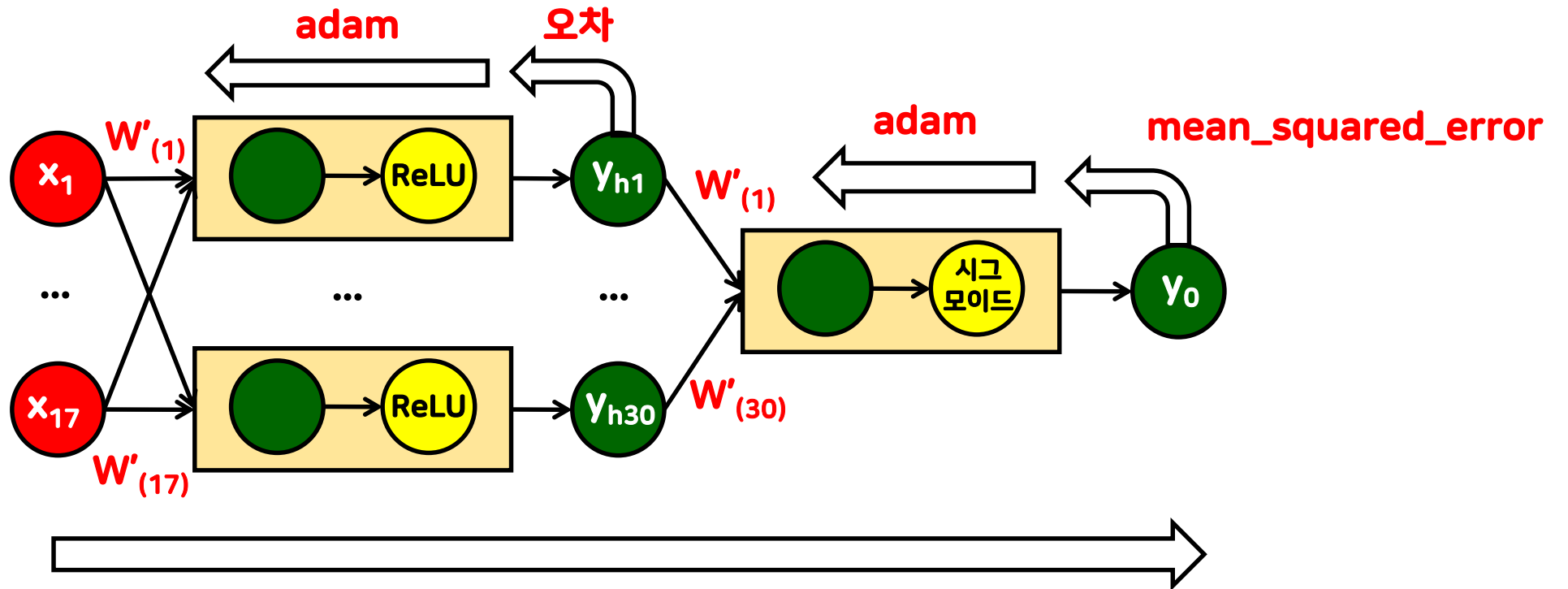


오차 역전파 (Error Backpropagation)

- MLP에서도 출력층의 오차를 은닉층으로 전파시켜 가중치나 바이어스를 갱신하고 다시 입력층으로 전파하여 가중치나 바이어스를 갱신

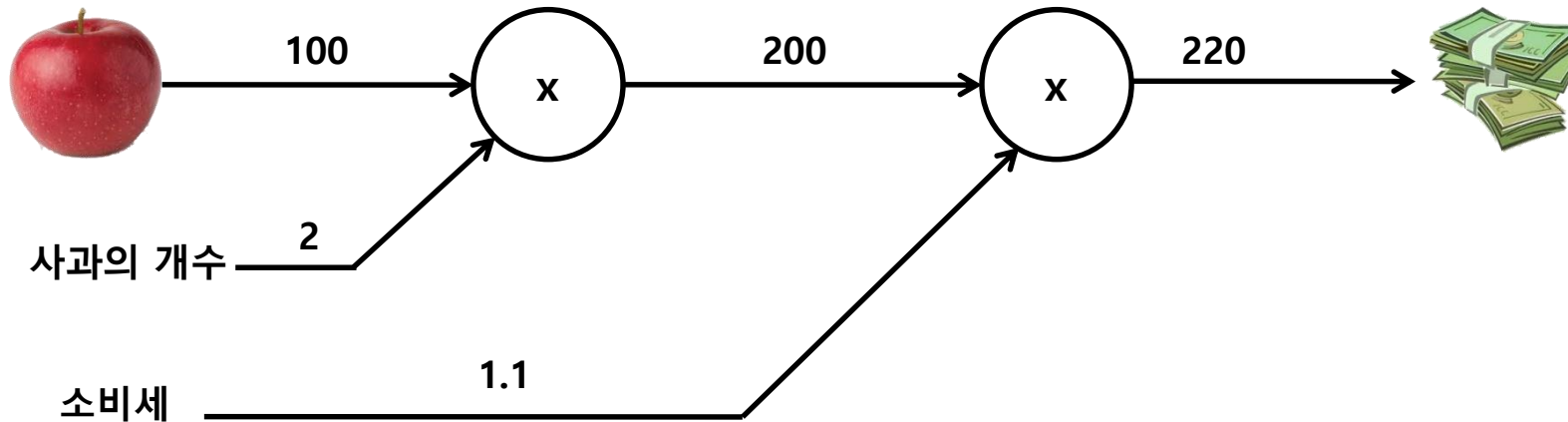
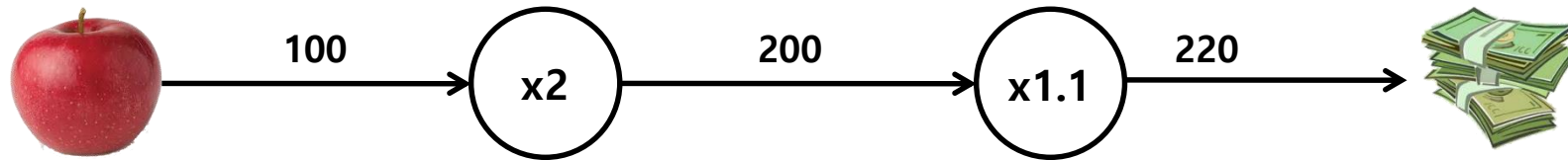


오차 역전파 (Error Backpropagation)



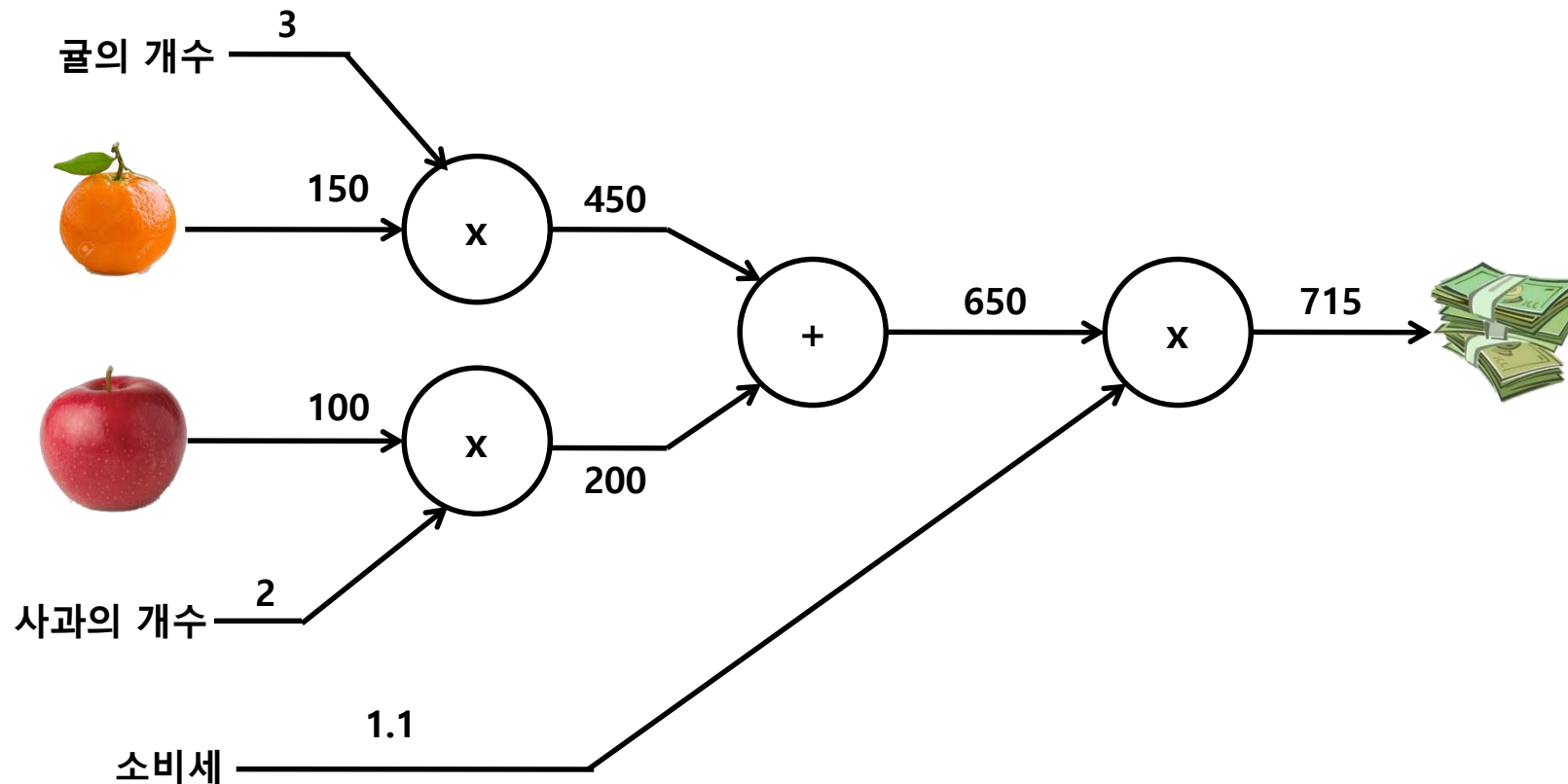
오차 역전파 (Error Backpropagation)

- 슈퍼에서 1개에 100원인 사과를 2개 샀다면 지불 금액은 ? (단, 소비세가 10% 부과됨)



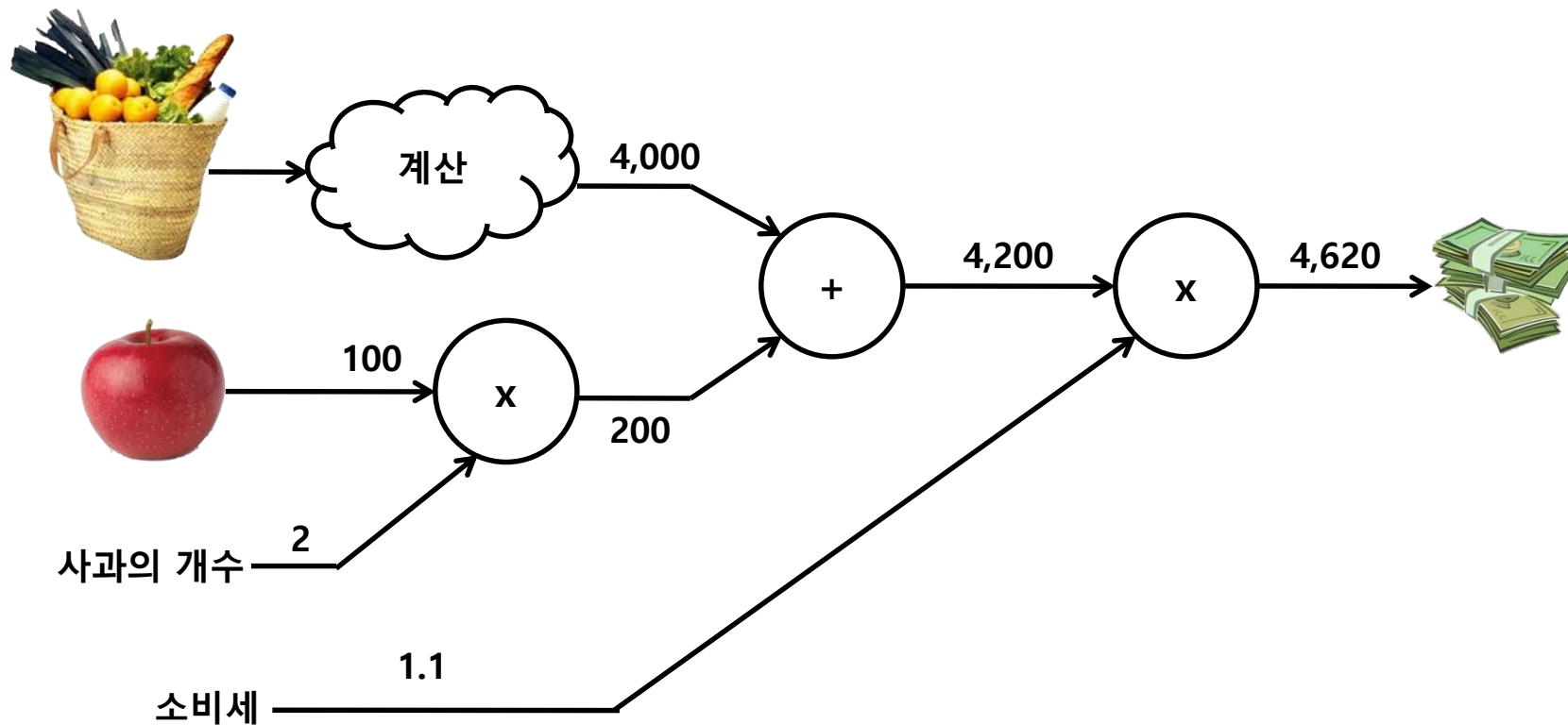
오차 역전파 (Error Backpropagation)

- 슈퍼에서 100원 짜리 사과 2개, 150원 짜리 귤 3개를 샀다면 지불 금액은 ?



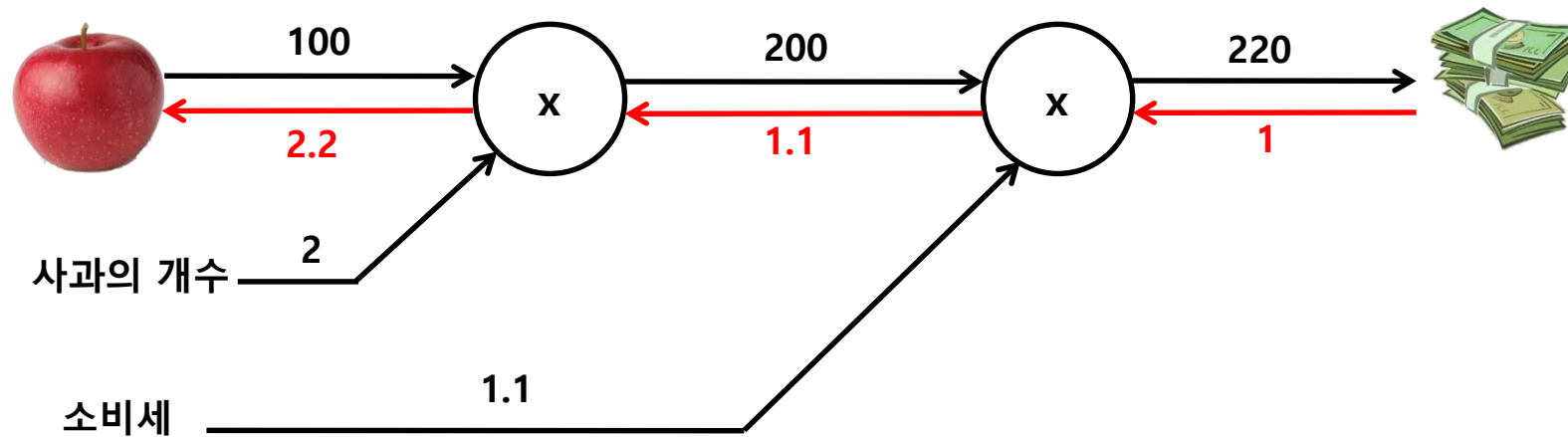
오차 역전파 (Error Backpropagation)

- 슈퍼에서 과일바구니와 사과 2개, 150원 짜리 굴 3개를 샀다면 지불 금액은 ?

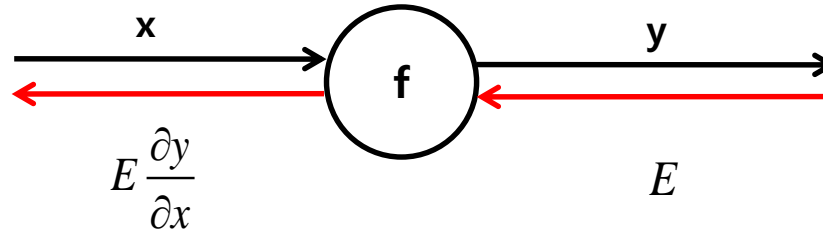


오차 역전파 (Error Backpropagation)

- 사과 가격에 대한 지불 금액의 미분값은 2.2 \rightarrow 사과 1원 오르면 최종 금액은 2.2원 오른다는 의미



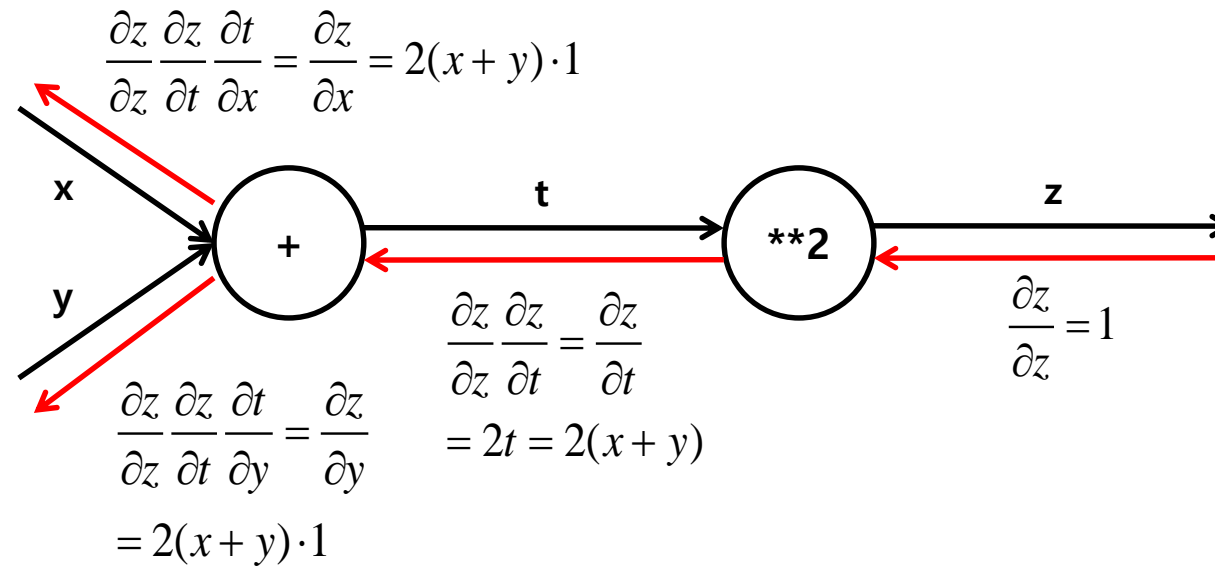
- 역전파는 국소적 미분을 오른쪽에서 왼쪽으로 전달



$$\frac{\partial y}{\partial x} = \frac{\text{출력}}{\text{입력}} = \text{입력에 대한 출력의 비}$$

연쇄법칙

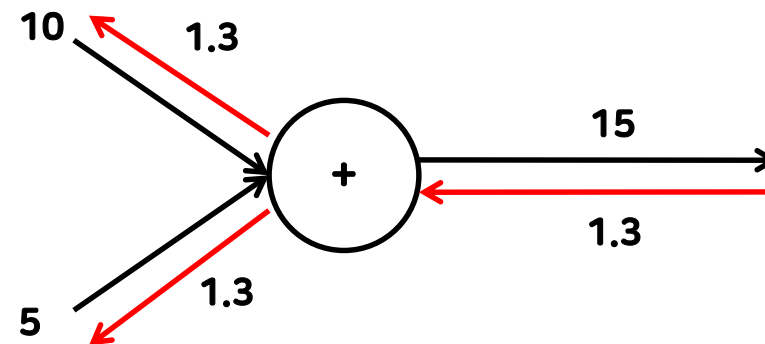
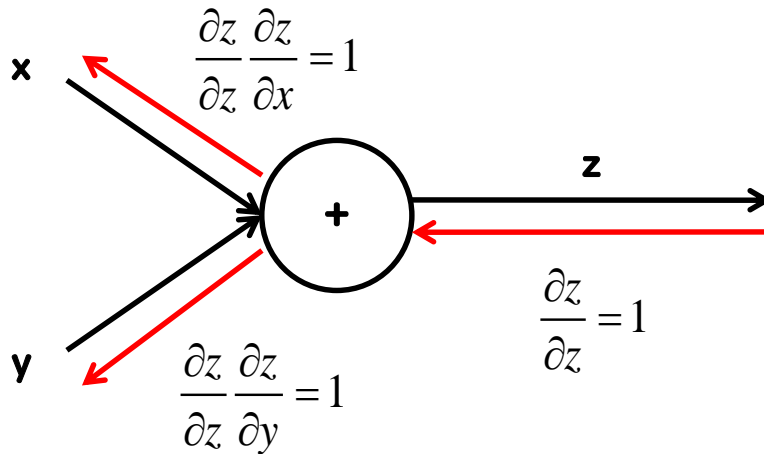
$$z = t^2 \quad t = x + y$$



연쇄법칙

- 덧셈 노드의 역전파는 입력 값을 그대로 전파

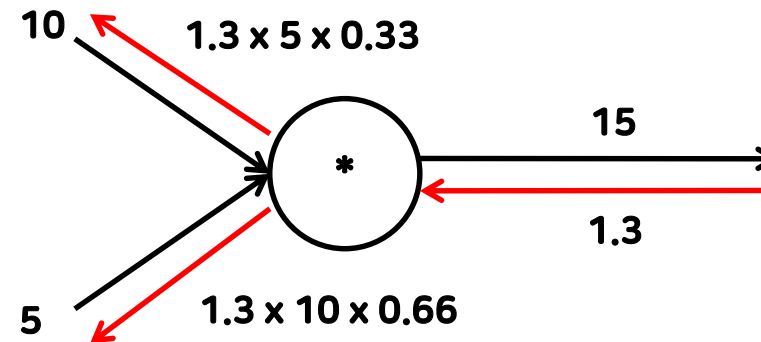
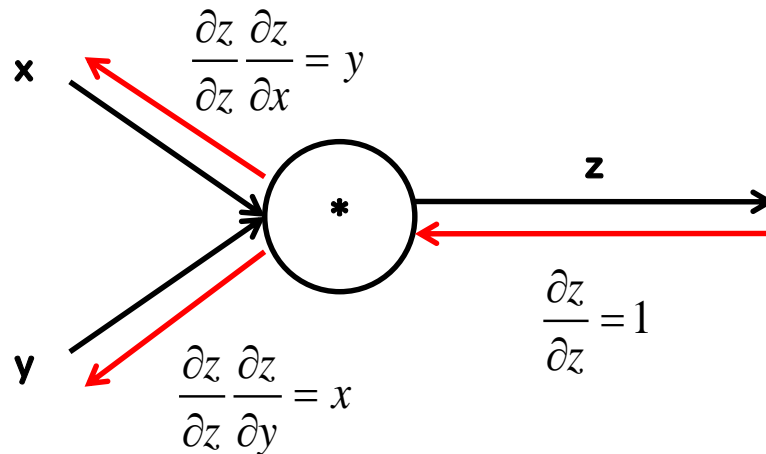
$$z = x + y$$



연쇄법칙

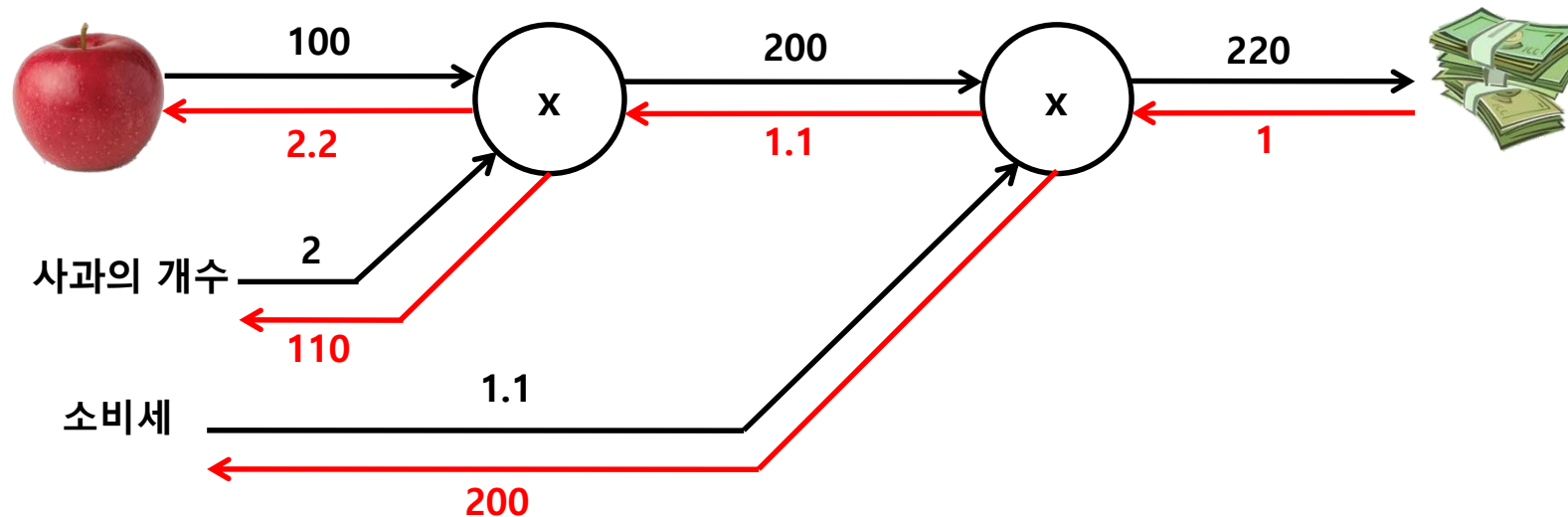
- 곱셈 노드의 역전파는 상류 값에 순전파 때의 입력 신호들을 서로 바꾼 값을 곱해서 하류로 보냄

$$z = xy$$



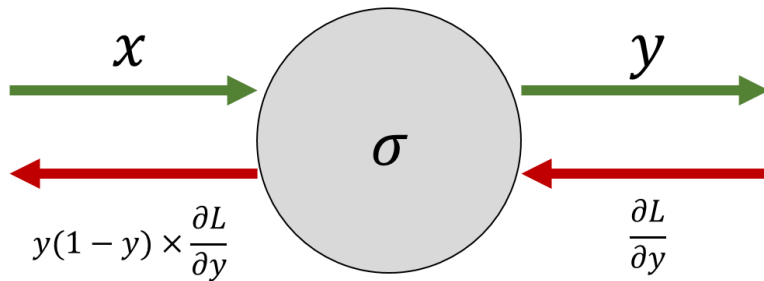
연쇄법칙

- 소비세와 사과 가격이 같은 양만큼 오르면 최종 금액에는 소비세가 200의 크기로 사과 가격이 2.2 크기로 영향을 준다는 의미

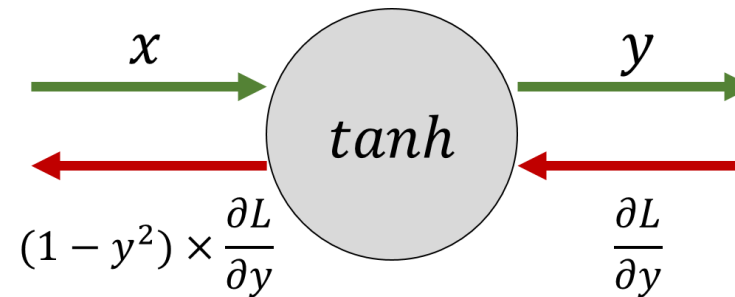


활성화 함수의 역전파

$$y = \frac{1}{1 + e^{-x}}$$
$$\frac{\partial y}{\partial x} = y(1 - y)$$



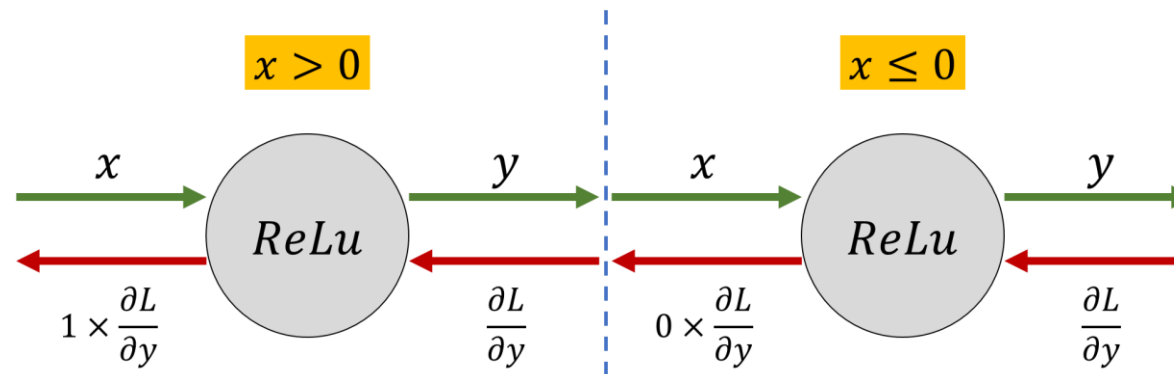
$$y = \tanh(x)$$
$$\frac{\partial y}{\partial x} = 1 - y^2$$



활성화 함수의 역전파

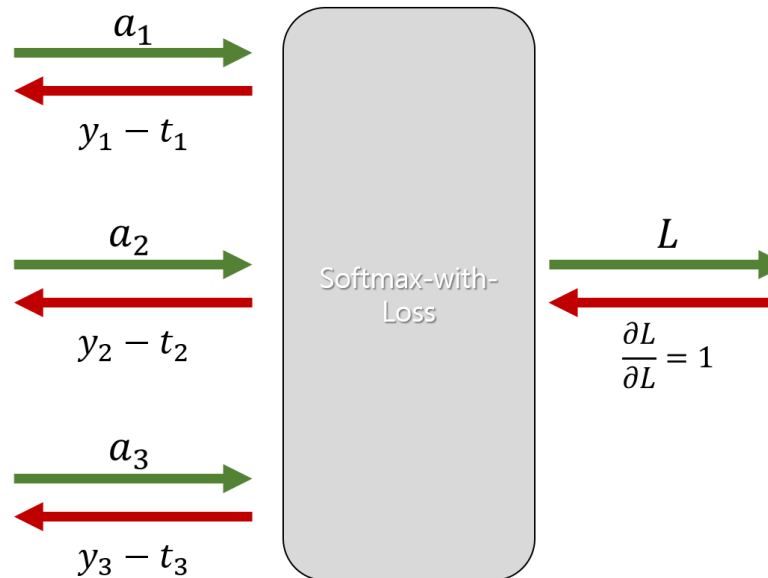
$$\begin{cases} y = x & (x > 0) \\ y = 0 & (x \leq 0) \end{cases}$$

$$\begin{cases} \frac{\partial y}{\partial x} = 1 & (x > 0) \\ \frac{\partial y}{\partial x} = 0 & (x \leq 0) \end{cases}$$



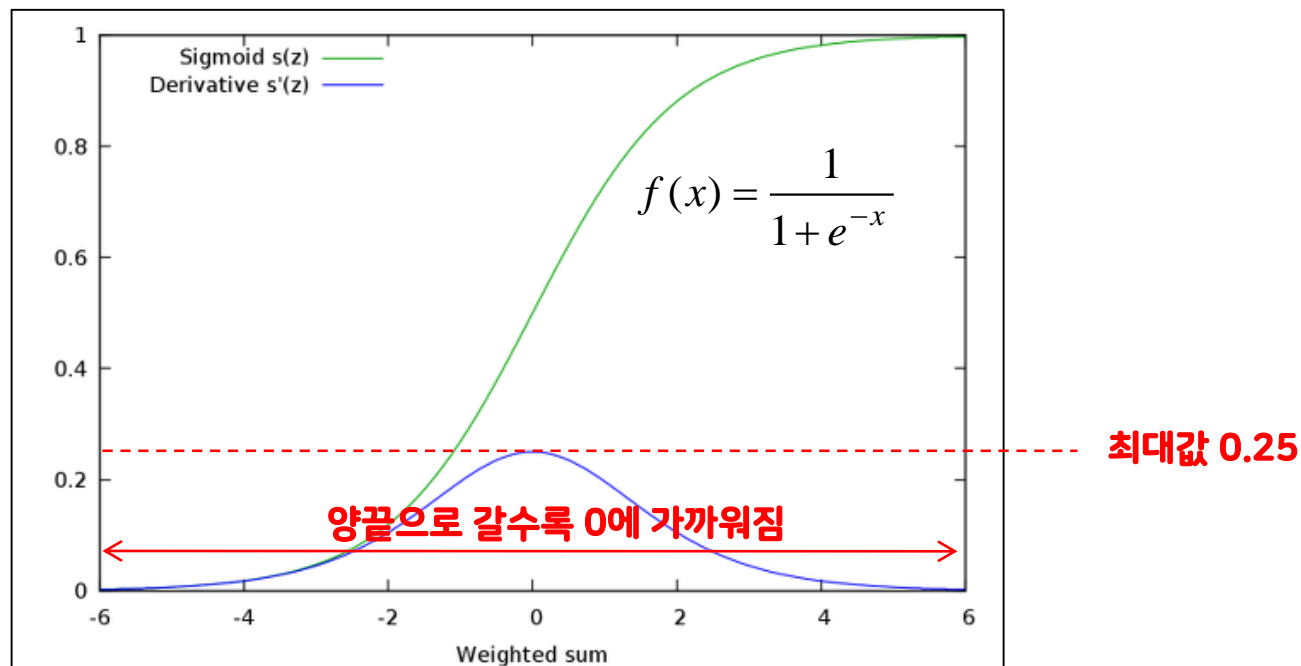
Softmax 함수의 역전파

- Softmax-with-Loss 노드는 a 을 입력받아 L 을 출력하고 정답인 노드만 1을 빼줌
- 만약 정답이 t_3 이라면 $y_3 - 1$ 으로 역전파하고 나머지는 y_1, y_2 로 역전파



Sigmoid 함수의 문제점

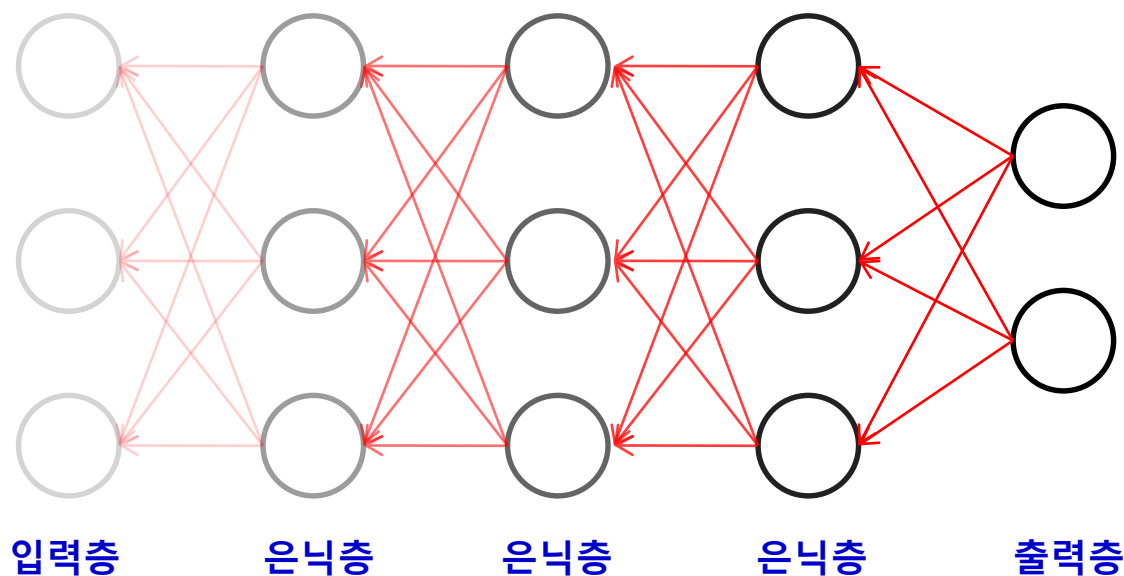
- **Gradient vanishing** 문제 발생 : 극단의 미분값(gradient)이 0이 곱해지면 전파되지 않음
- 활성화 함수 결과 값의 중심이 0이 아닌 0.5 → 모두 양수이거나 모두 음수일 가능성
- 지수이므로 계산이 복잡



시그모이드 함수를 미분

Sigmoid 함수의 문제점

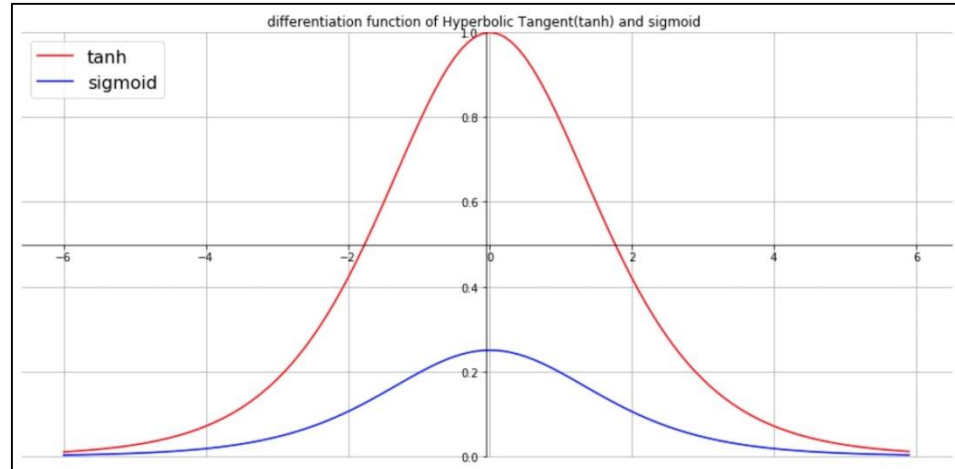
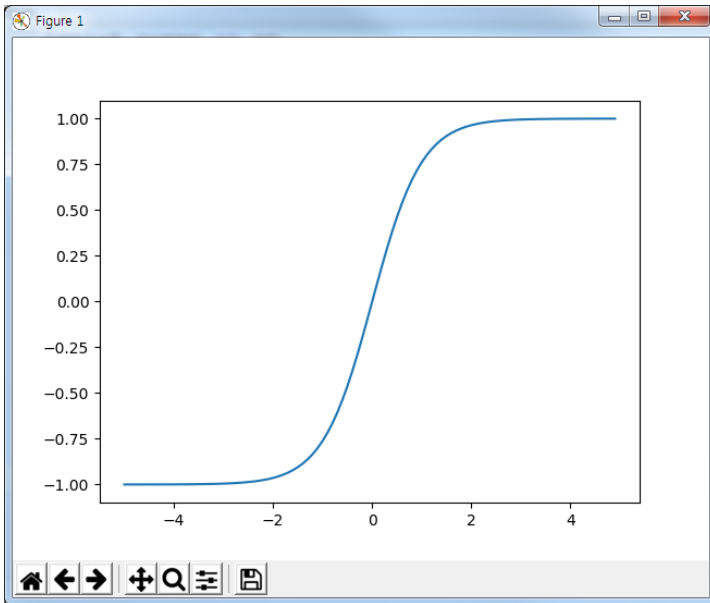
- MLP이 오차 역전파를 만나 신경망이 되었고 XOR 문제를 해결하였음.
- 하지만 오차 역전파는 출력층부터 입력층까지 하나씩 앞으로 돌아가면서 각 층의 가중치를 수정하는데 이때 미분 (기울기)을 하기 때문에 중간에 기울기가 0이 되는 경우가 발생 → **Gradient Vanishing**



sigmoid 함수와 tanh 함수를 미분해보자

Sigmoid 함수의 문제점

- Gradient vanishing 문제가 Sigmoid 함수보다는 tanh 함수가 덜 발생 → 결과 값이 $[-1, 1]$ 사이로 제한되고 중심값이 0이므로
- 여전히 vanishing gradient 문제 발생

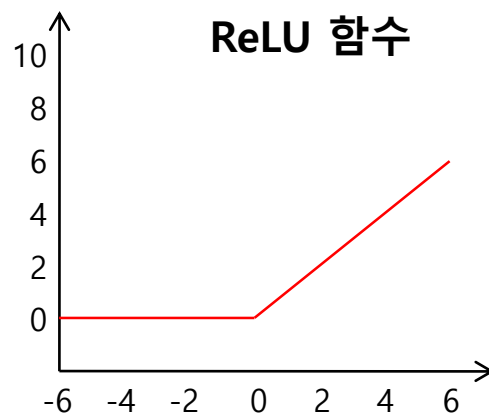


시그모이드와 tanh의 미분 그래프

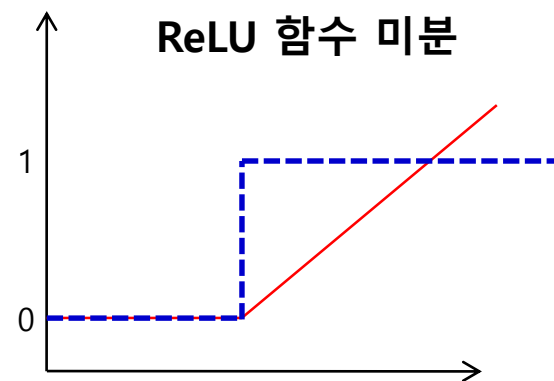
$$\tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$

ReLU 함수

- **Sparse activation** : 0이하의 입력에 대해 0을 출력함으로써 부분적으로 활성화시킬 수 있음
- **Efficient gradient propagation** : **gradient의 vanishing이 없음** → 양극단 값이 포화되지 않음
(gradient가 exploding되지 않음)
- **Efficient computation** : **선형함수이므로 미분 계산이 간단** (6배 정도 빠름)
- **Scale-invariant** : 스케일 조정에 따라 값이 변하지 않음



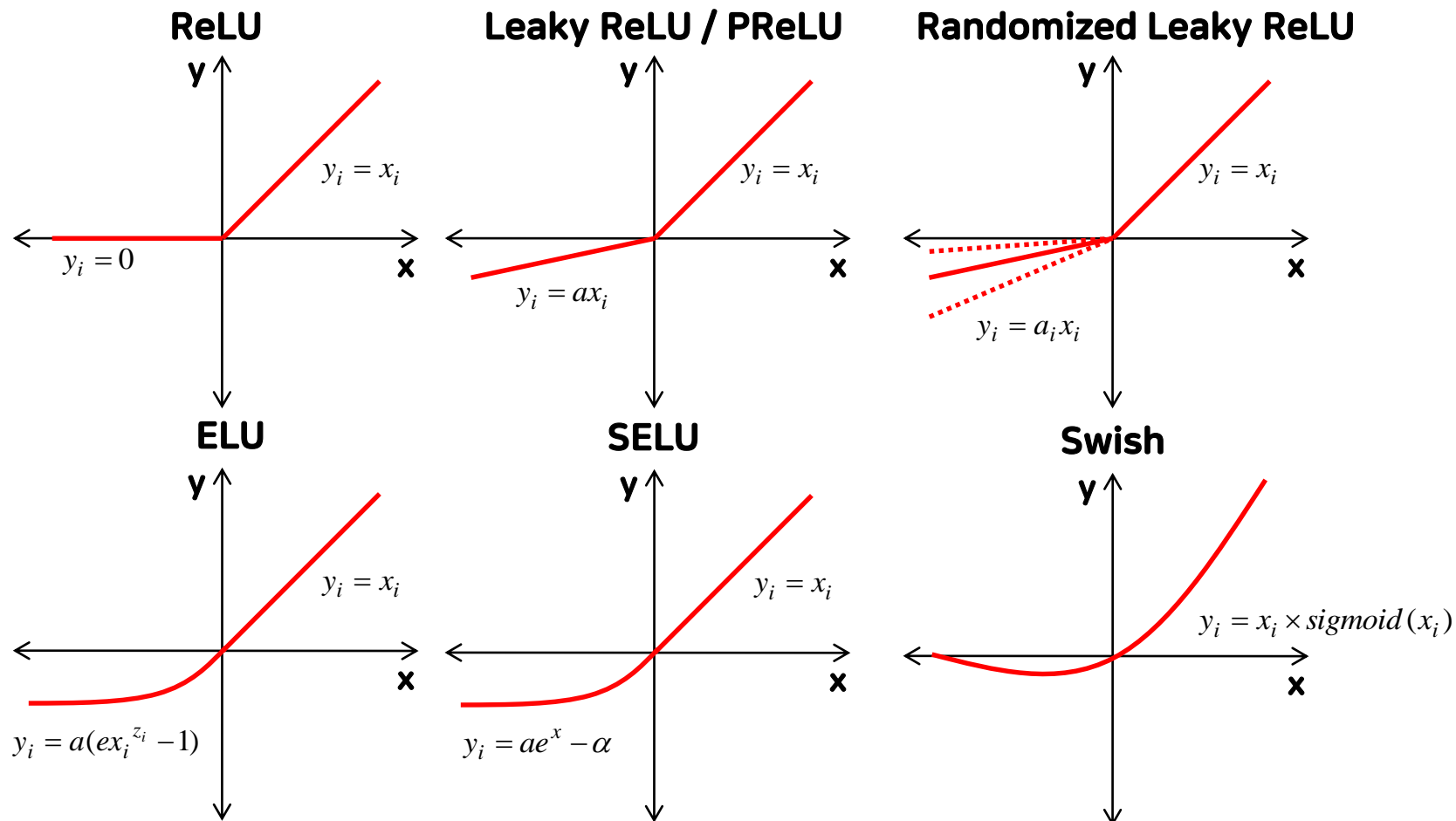
$$f(x) = \begin{cases} 0 & x < 0 \\ x & x \geq 0 \end{cases}$$



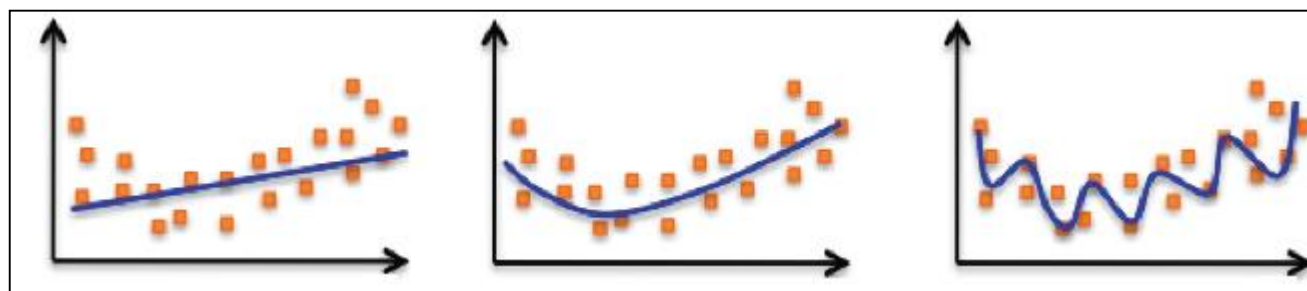
ReLU 함수

- 문제점 : 음수를 모두 0으로 처리하기 때문에 한번 음수가 나오면 그 노드는 학습되지 않는다는 문제
→ 좋지 않는 성능 → leaky ReLU나 다른 ReLU 사용
- **Leaky ReLU** : 음수의 기울기 값을 0이 아닌 작은 값(0.1, 0.01 등)으로 설정
- **Parametric ReLU (PReLU)** : 음수의 기울기 값이 변경
- **ELU (Exponential Linear Unit)** : 음수의 기울기 값을 지수 형태로 설정 → PReLU가 비슷한 성능
- **SELU (Scaled ELU)** : 2개의 파라미터를 이용하여 기울기를 변경 → 일정한 분산 → PReLU와 비슷
- **Swish** : 구글에서 나온 함수 → 성능 우수
- **maxout** : 두 개의 w와 b 중에서 큰 값을 선택 → 성능 우수
- 활성화 함수로 ReLU로 사용하더라도 **마지막 Layer는 Sigmoid나 tanh 함수를 사용** → 0~1 사이의 값을 나타내야 정확히 분류 하는데 좋기 때문

ReLU 함수



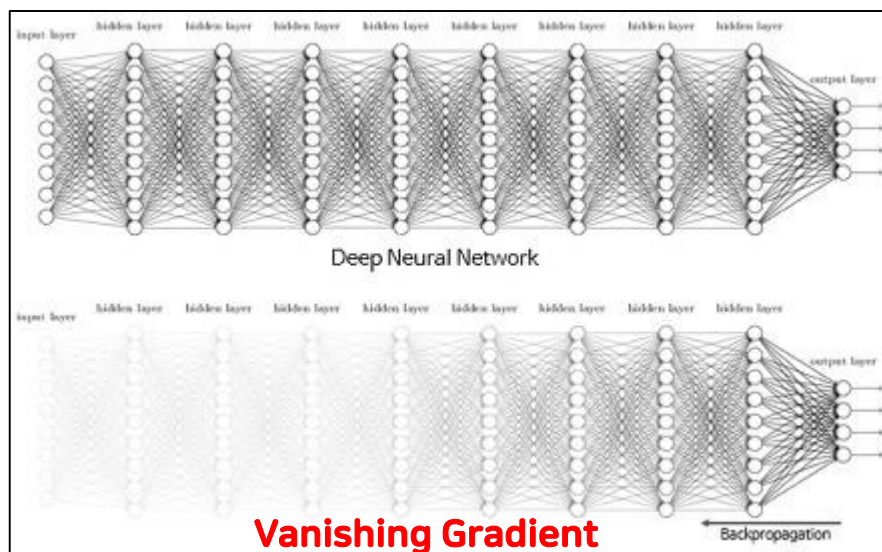
MLP의 문제점



Underfitting

Generalization

Overfitting



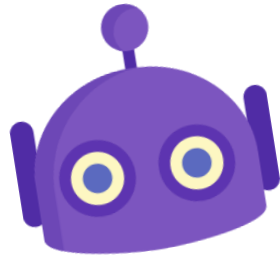
Vanishing Gradient

역전파를 사용할 때 sigmoid 함수 사용

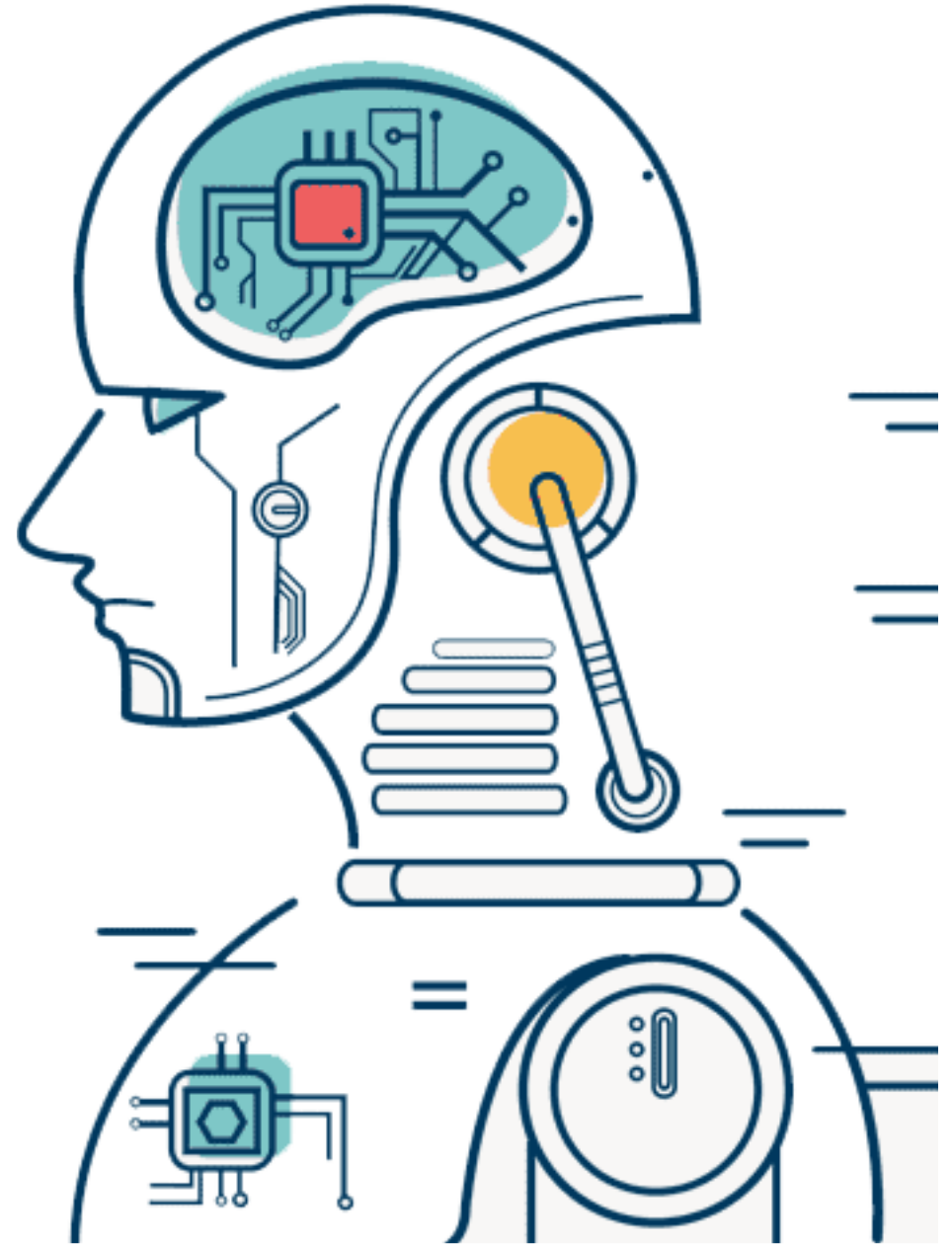
→ 0과 1값이 심하게 변형되어 감소됨

→ 층이 깊어지면 0에 가까운 값이 됨

→ Vanishing (Exploding)

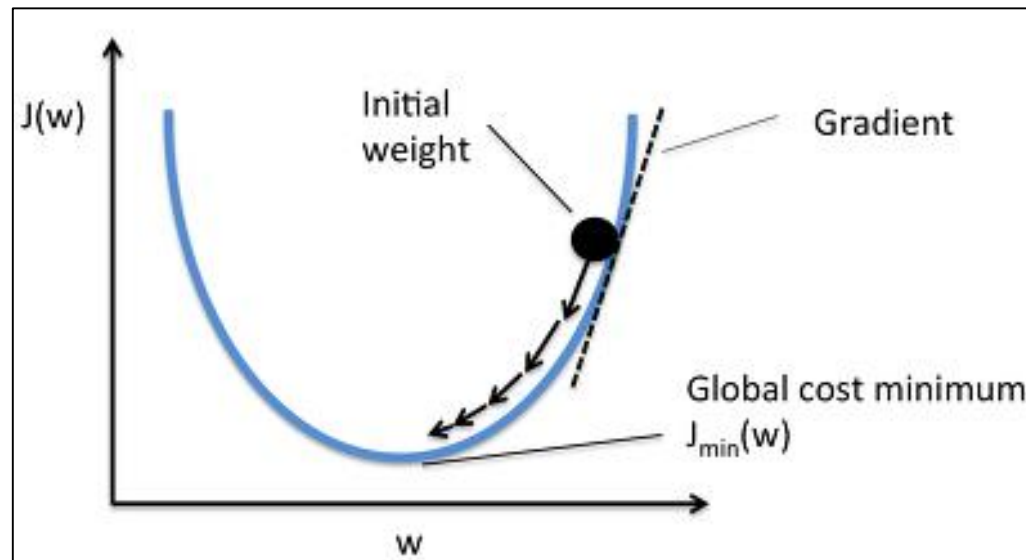


경사하강법

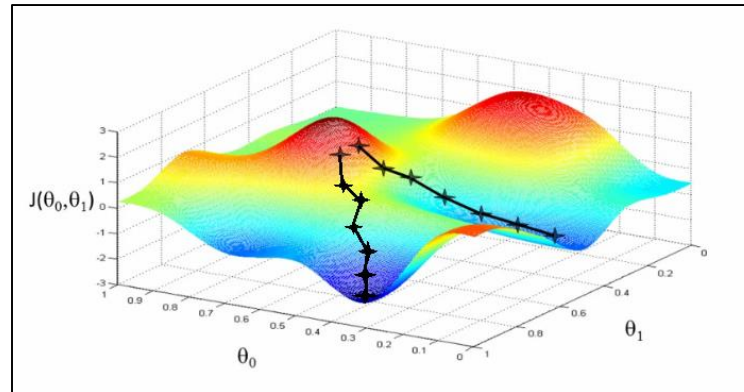


- **다양한 경사하강법 종류를 알 수 있다.**
- **Keras를 활용해 다양한 경사하강법을 적용 할 수 있다.**

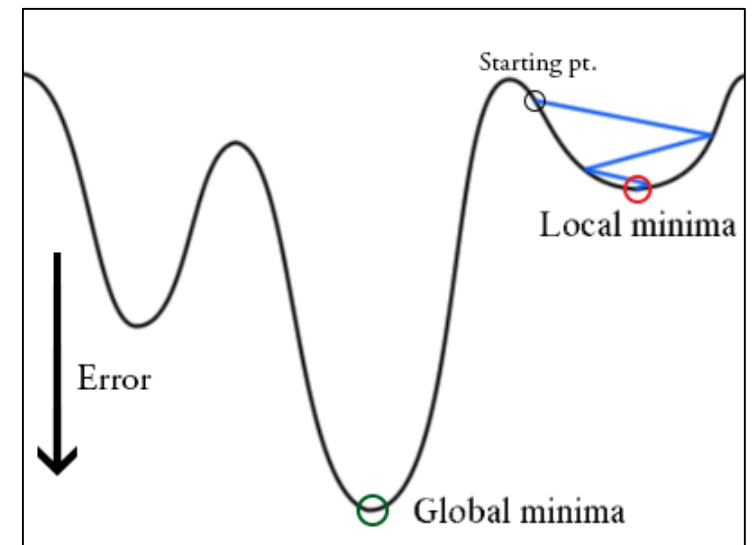
- 경사 하강법은 주어진 데이터 세트에 대해 손실함수 $E(w)$ 가 최소가 되는 가중치 w 를 찾는 작업
- 현재 지점의 w 에서 기울기가 가장 가파르게 하강하는 곳을 따라 조금씩 이동
- 현재 지점의 w 에서 $E(w)$ 의 w 에 대한 음의 미분값이 가장 높은 방향(기울기가 작은, 더 낮은 방향)으로 w 를 조금씩 이동
- 문제는 이동하는 정도가 **크면 최저 지점을 지나갈 수도 있고**, 너무 **작으면 이동 횟수가 많아져서** 최저 지점을 찾지 못할 수도 있음



- 한 발 내려간 시점에서 등산객이 자기가 산을 제대로 내려가고 있는지 주변 경사(기울기)를 보면서 판단하듯이 기울기가 음수인지? 양수인지? 확인하면서 최저지점으로 내려감.



- **지역 최소값 (local minimum)** : 학습 중에 손실함수의 전역 최소값 (global minimum)을 찾지 못하고 지역 최소값에 빠져 나오지 못하는 상황

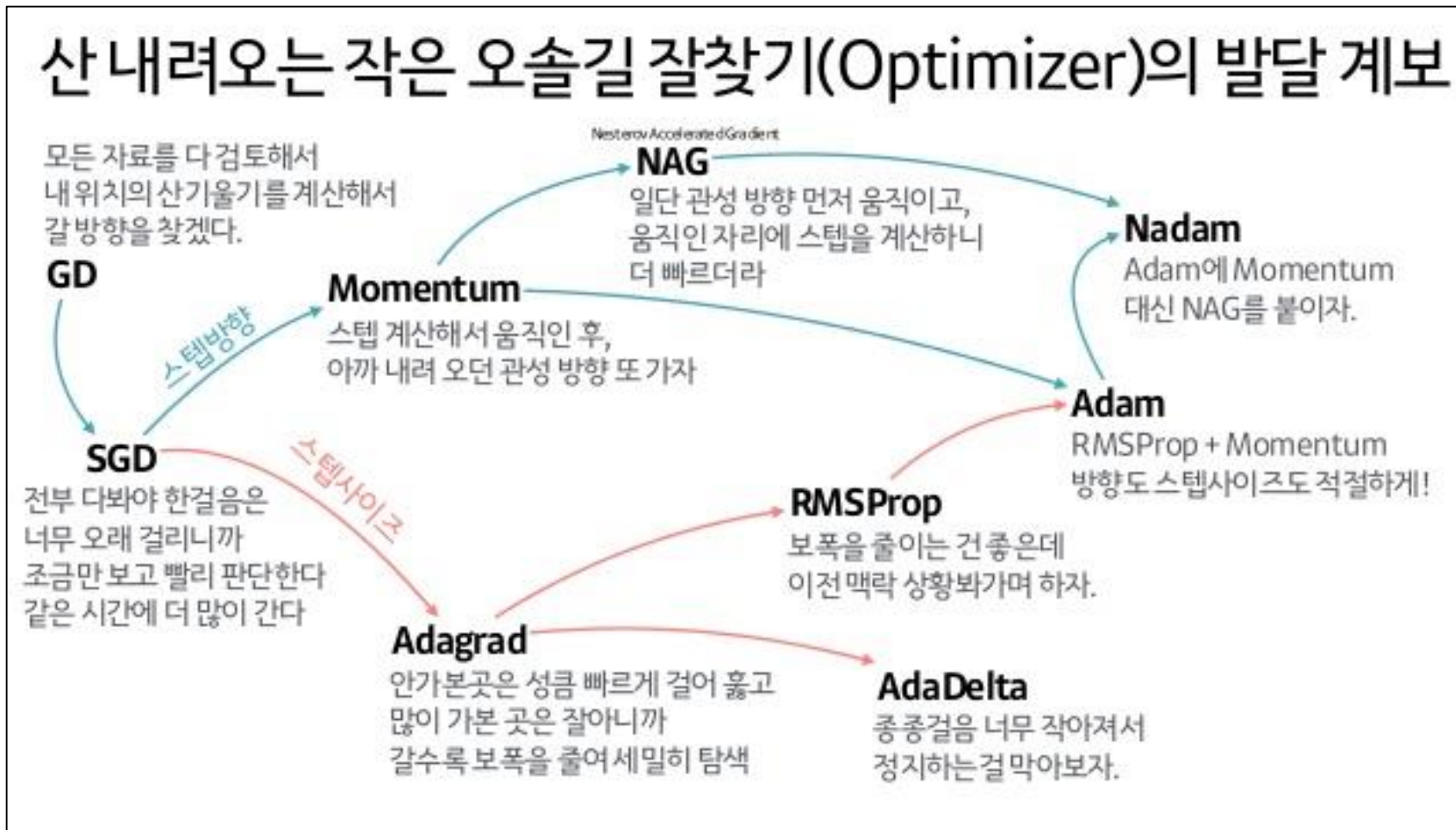


경사하강법 종류

- lr(learning rate) 값만 적절히 변경하고 epsilon, rho, decay는 그대로 사용하는 것을 권장

종류	개요	효과	케라스 사용법
SGD	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트 하게 하는 것	속도개선	<code>keras.optimizers.SGD(lr=0.1)</code>
모멘텀	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도개선	<code>keras.optimizers.SGD(lr=0.1, momentum=0.9)</code>
NAG	모멘텀이 이동시킬 방향으로 미리 이동해서 경사를 계산, 불필요한 이동을 줄이는 효과	정확도개선	<code>keras.optimizers.SGD(lr=0.1, momentum=0.9, nesterov=true)</code>
Adagrad	변수의 업데이트가 잦으면 학습률을 적게하여 이동 보폭을 조절하는 방법	보폭 크기 개선	<code>keras.optimizers.Adagrad(lr=0.01, epsilon=1e-6)</code>
RMSProp	Adagrad의 보폭 민감도를 보완한 방법	보폭 크기 개선	<code>keras.optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)</code>
Adam	모멘텀과 RMSProp를 합친 방법	정확도, 보폭크기개선	<code>keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-08, decay=0.0)</code>

경사하강법 종류

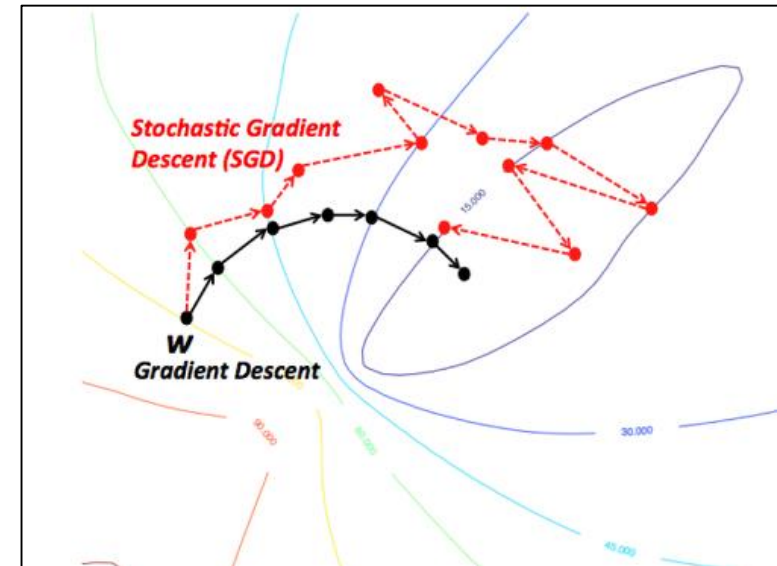


확률적 경사하강법 (SGD)

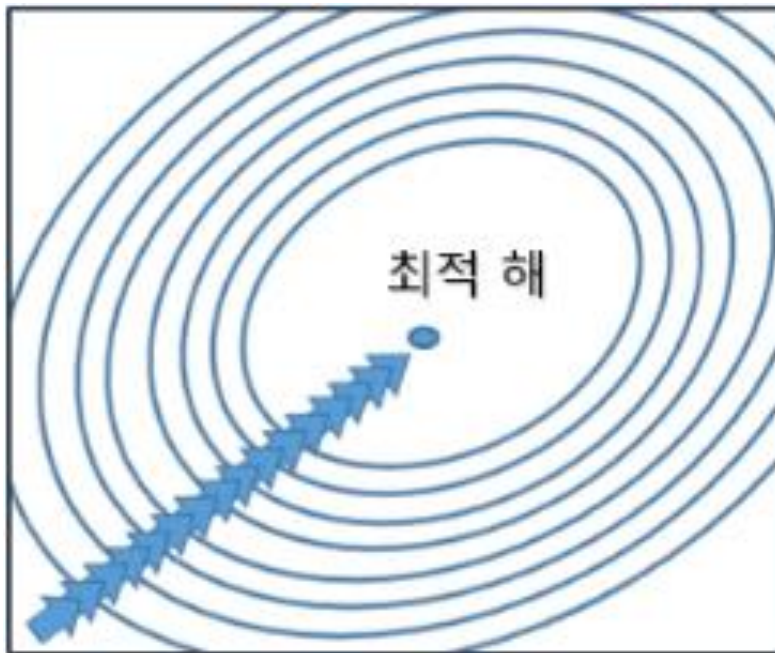
- 경사하강법의 많은 계산량(batch)은 속도를 느리게 하고 최적해를 찾기 전에 멈출 수도 있음.
- **SGD (Stochastic Gradient Descent)** : 전체 데이터가 아닌 랜덤하게 추출한 일부 데이터 (Mini-batch)를 사용하는 방법 → 더 빠르고 자주 업데이트 가능
- 중간 결과의 진폭이 크고 불안정해 보일 수 있지만 빠르고 최적해에 근사한 값을 찾아내는 장점

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

- W : 업데이트할 가중치
- η : 학습률 (일반적으로 0.01이나 0.001을 사용)
- $\frac{\partial L}{\partial W}$: W 에 대한 손실함수의 기울기



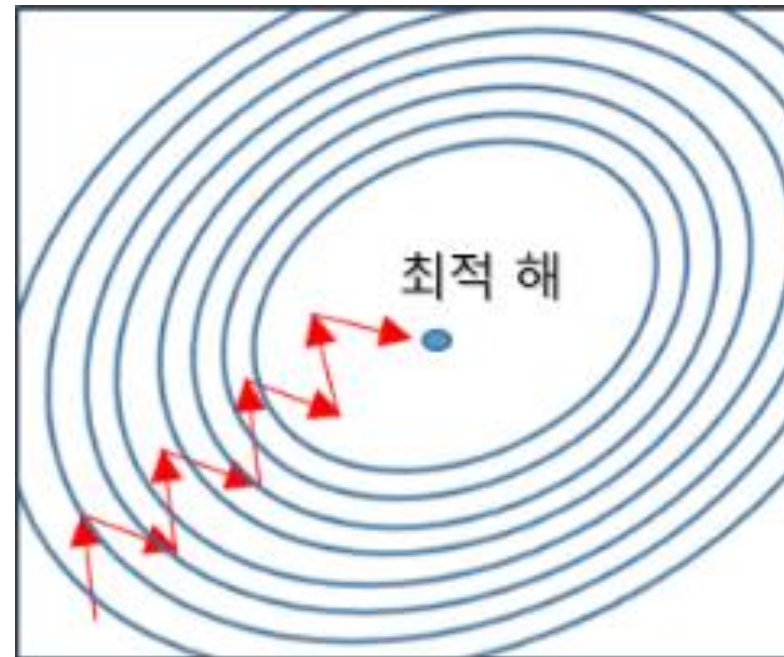
확률적 경사하강법 (SGD)



경사하강법

(Gradient Descent)

전체 데이터를 이용해 업데이트



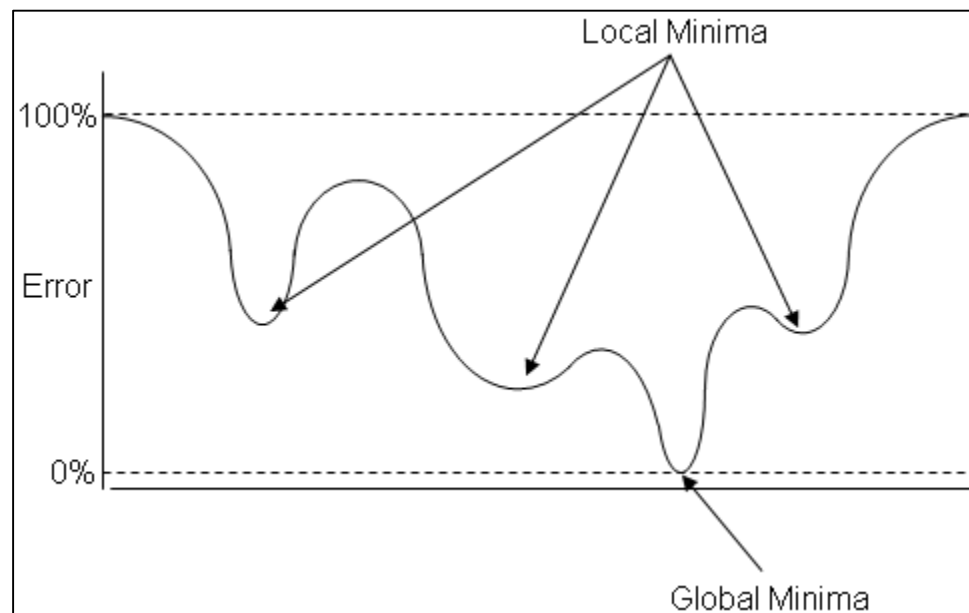
확률적경사하강법

(Stochastic Gradient Descent)

확률적으로 선택된 일부 데이터를
이용해 업데이트

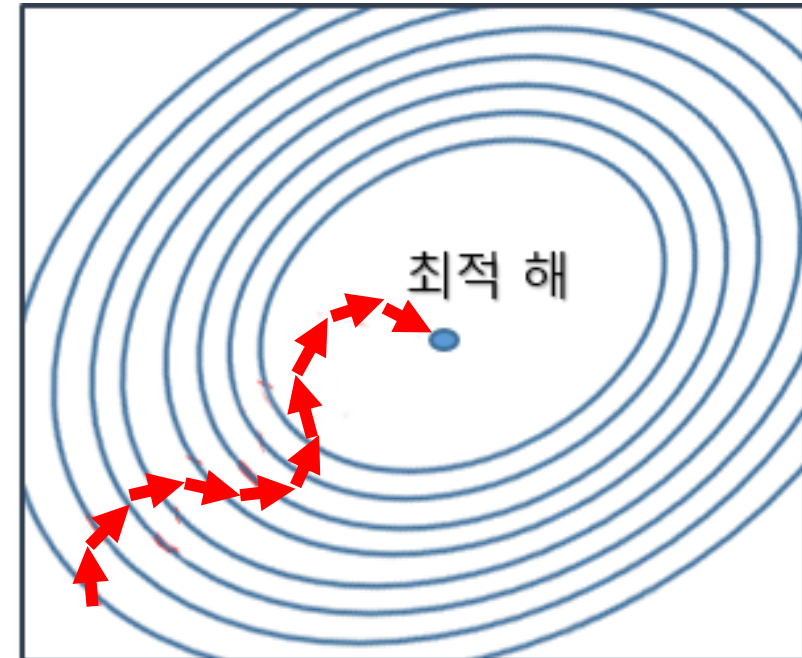
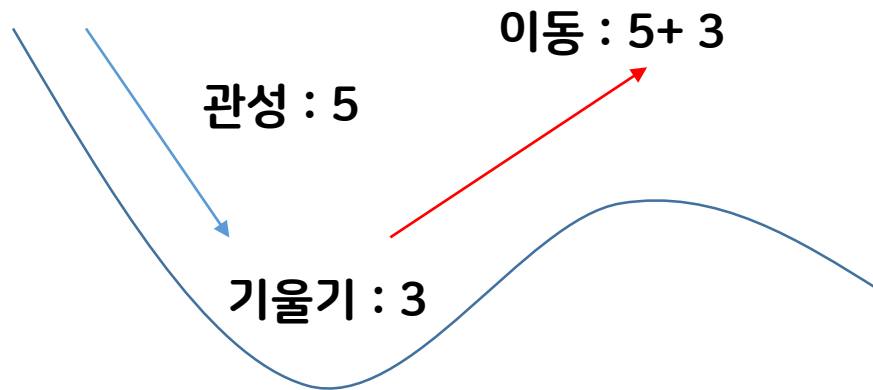
모멘텀

- 확률적 경사하강법은 방향에 따라서 기울기 값이 달라지는 경우에 적합하지 않음 → 최소 기울기 방향으로 움직이므로 (Local Minima에 빠질 수 있음)



모멘텀

관성하고 기울기를 모두
고려하여 한번에 이동



모멘텀

(Momentum)

경사 하강법에 관성을 적용해 업데이트

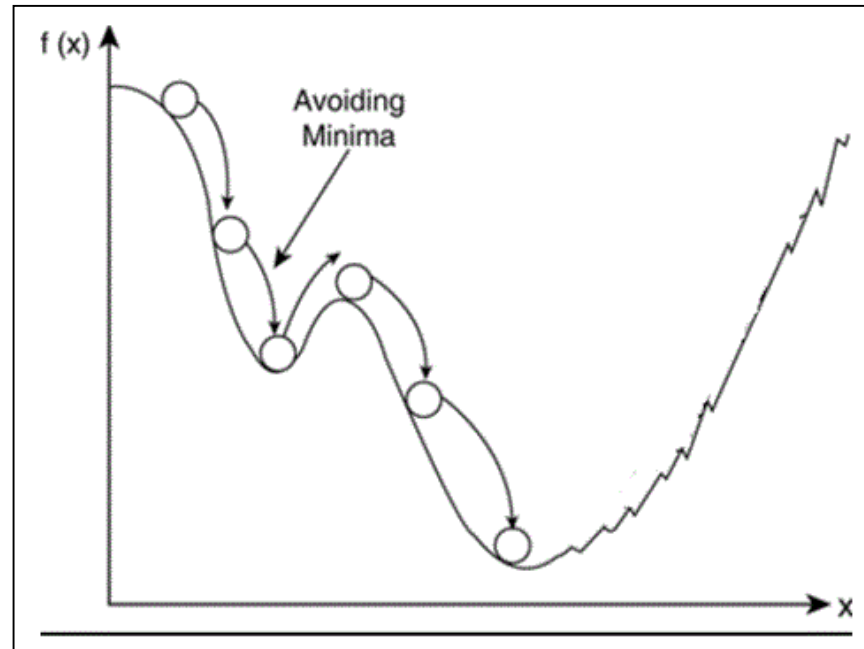
모멘텀

- 관성을 부여(업데이트에 사용했던 기울기의 일정 %를 남겨서 현재 기울기에 더하여 업데이트)하는 방법

속도만큼 더해서 더 크게 변하게 함

$$v \leftarrow av - \eta \frac{\partial L}{\partial W}$$
$$W \leftarrow W + v$$

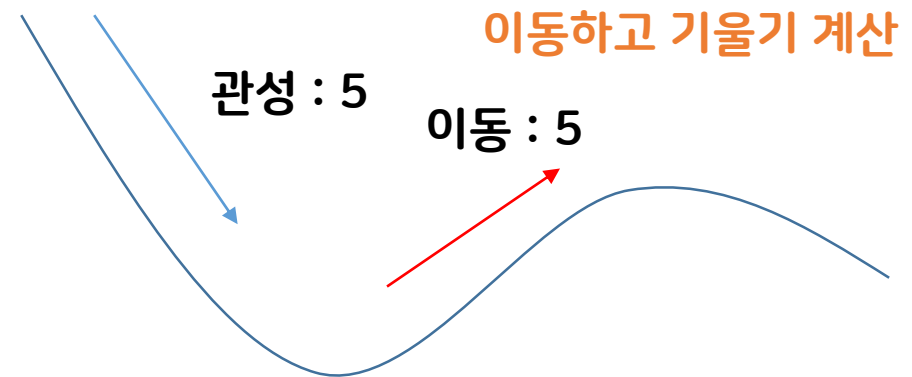
- v : 속도
- a : 모멘텀 계수 (보통 0.9)



NAG (Nesterov Accelerated Gradient)

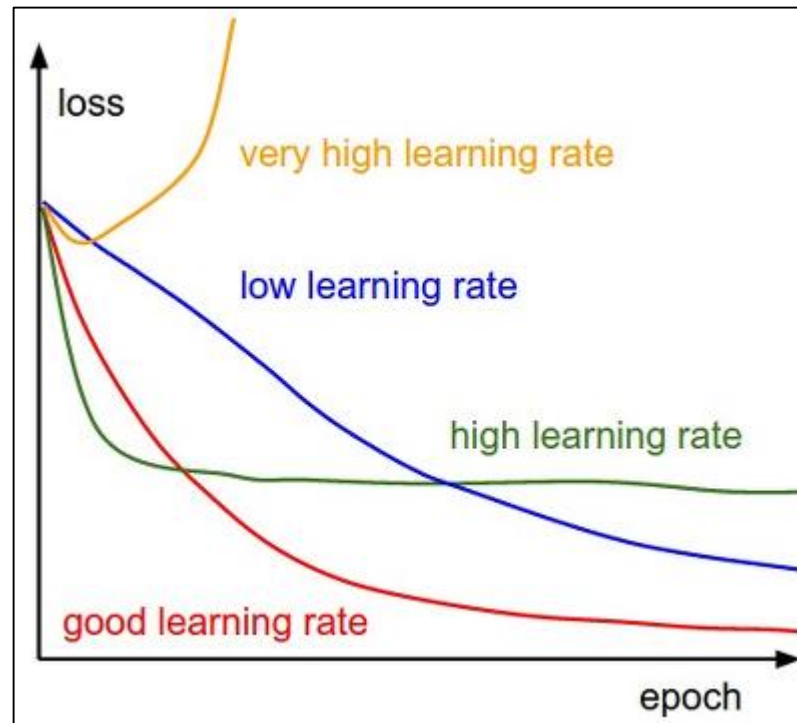
- 업데이트 시 모멘텀 방식으로 먼저 더한 다음 계산.
- 미리 해당 방향으로 이동한 뒤 그래디언트를 계산하는 효과.
- 불필요한 이동을 줄일 수 있다.

관성만큼 이동하고
이후에 기울기를 계산하여 이동



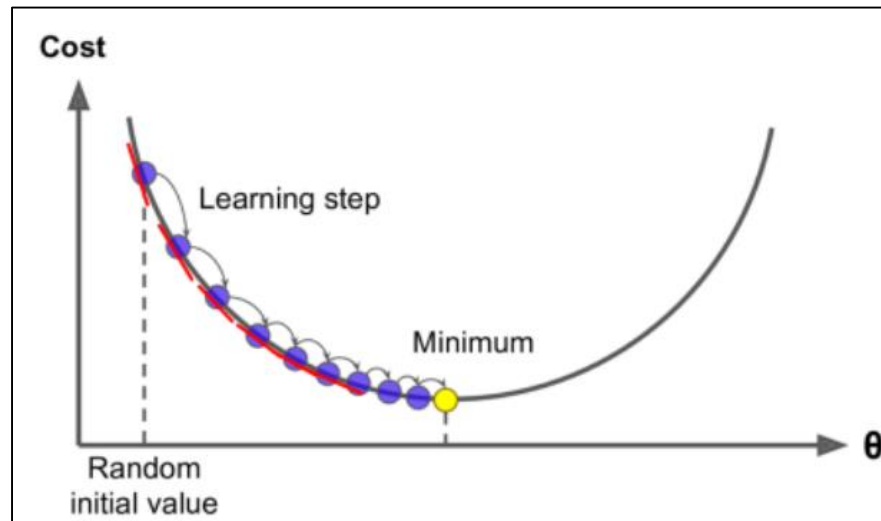
Adagrad

- 학습률 (Learning rate)는 스텝 사이즈로 작으면 학습이 오래 걸리고 크면 최적 값을 찾지 못하므로 적절한 학습률 적용이 필요



Adagrad

- 신경망에서 학습률은 이동 보폭으로 생각할 수 있는데 한번 갱신하는 가중치의 값의 크기를 결정
- 학습률을 작게하면 학습 시간이 느리고 크게하면 최적 점을 지나칠 수 있음.
- 학습 과정에서 점차 기울기가 감소하므로 학습률을 줄여가는데 (**Learning rate decay**) 보통 **과거의 기울기 값을 제공해서 더하는** 하는 방식을 사용 → SD, SGD, 모멘텀처럼 동일하게 줄이지 않고 **이전 기울기를 참조하는 적응형 방법**

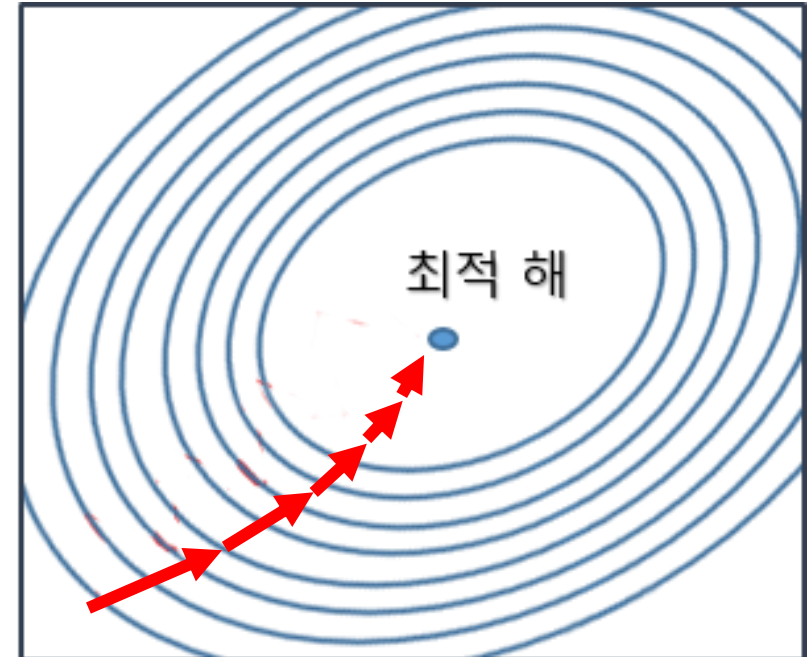


Adagrad

- 매개변수 원소 중에서 많이 움직인 (크게 갱신된) 원소는 학습률이 낮아지는 방식으로 학습률이 매개변수에 따라 변경되도록 하는 방식

$$h \leftarrow h + \frac{\partial L}{\partial W} \otimes \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

- h : 기존 기울기들의 제곱의 합



아다그리드

(Adaptive Gradient)

학습률 감소 방법을 적용해 업데이트

RMSprop

- AdaGrad는 제곱을 하여 학습률을 줄이기 때문에 빠르게 0에 가까워져 학습이 멈추는 문제가 발생
- RMSprop는 과거의 모든 기울기를 균등하게 더하지 않고 새로운 기울기의 정보만 반영하도록 해서 학습률이 크게 떨어져 0에 가까워지는 것을 방지하는 방법
- 이전에 제공되어 누적된 값과 새롭게 제공되는 부분의 반영비율을 decay 상수로 조절

decay 상수

$$h \leftarrow \gamma h + (1 - \gamma) \frac{\partial L}{\partial W} \otimes \frac{\partial L}{\partial W}$$
$$W \leftarrow W - \eta \frac{1}{\sqrt{h}} \frac{\partial L}{\partial W}$$

Adam

- 모멘텀과 RMSprop 방식을 섞은 방식으로 관성 계수를 사용하여 Local Minima에 빠지지 않고 학습률에 대한 계수를 적응형을 변경하는 방식

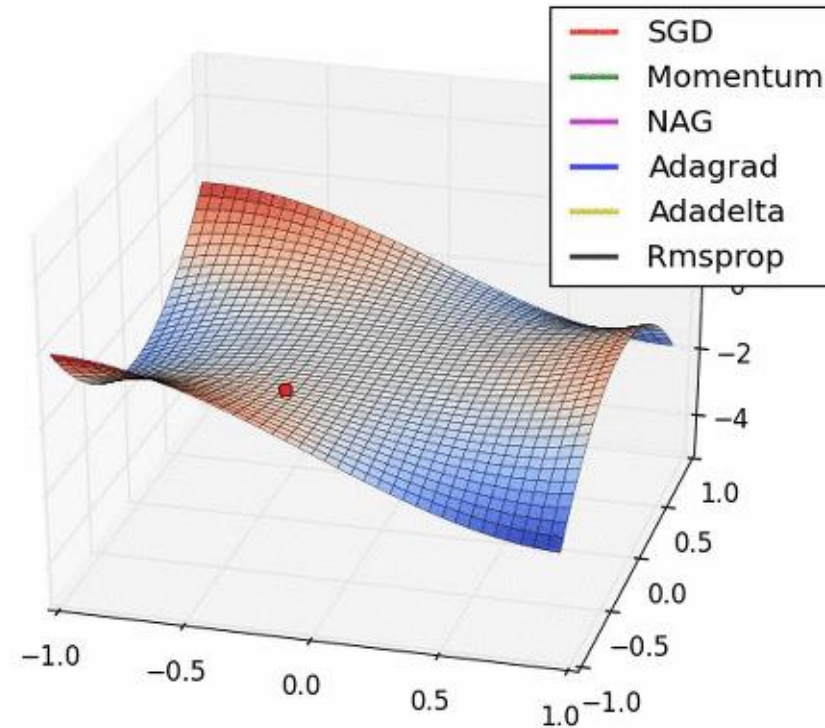
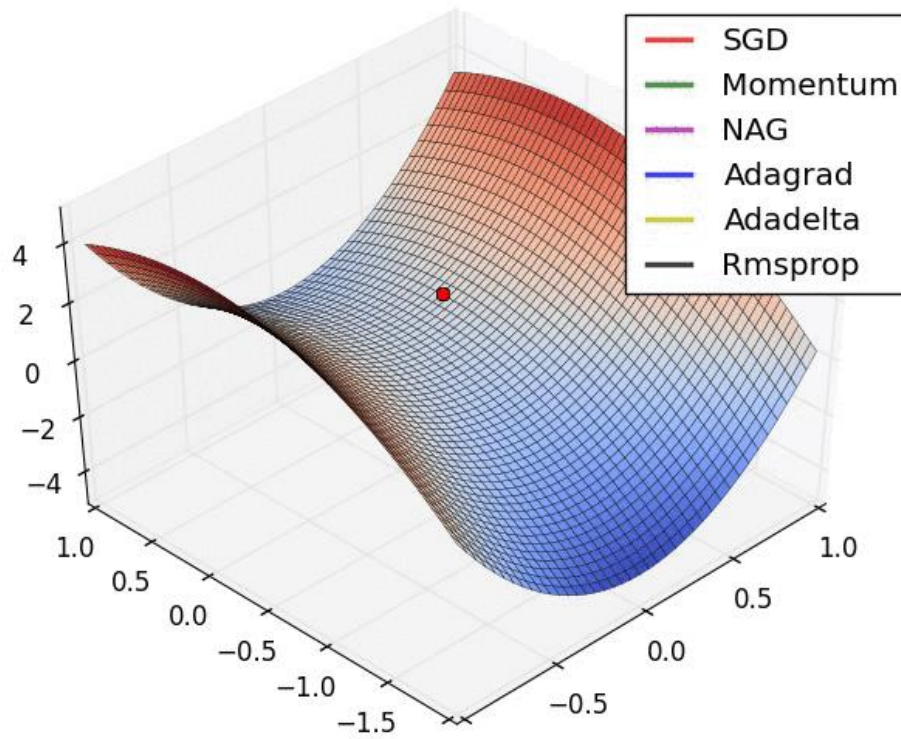
- (1) 현재의 기울기를 계산
- (2) 이전 모멘텀에 현재 기울기를 더해서 누적 \rightarrow beta1를 곱해서 누적량과 현재량의 반영비율 조절
- (3) 이전 기울기에 현재 기울기를 제곱해서 누적 \rightarrow beta2를 곱해서 누적량과 현재량의 반영비율 조절
- (4) 두 값을 더해서 제곱근을 구한 값을 나눔

$$M \leftarrow \beta_1 M + (1 - \beta_1) \frac{\partial L}{\partial W}$$

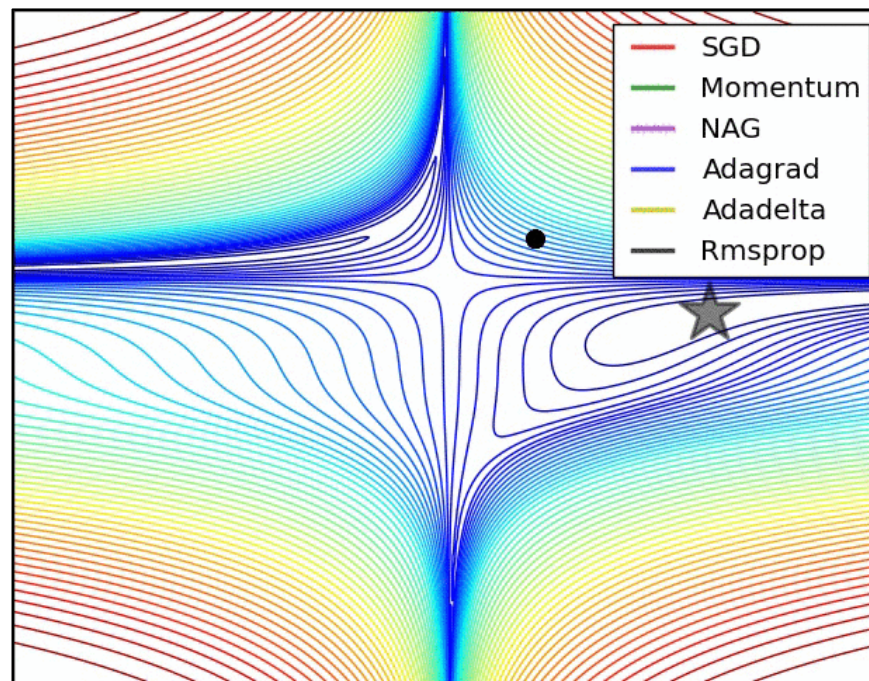
$$V \leftarrow \beta_2 V + (1 - \beta_2) \frac{\partial L}{\partial W} \times \frac{\partial L}{\partial W}$$

$$W \leftarrow W - \eta \frac{M}{\sqrt{V}}$$

시뮬레이션



시뮬레이션



Keras로 MNIST 손글씨 학습하기 +가중치, 활성화함수, 옵티마이저, 규제적용

**Keras활용 보스턴 주택 값 예측 신경망 실습
+모델검증, 학습중단, 모델저장, 모델로드**

<http://playground.tensorflow.org>

