SW코딩과 AI활용 (Software Coding and AI Practical Use)

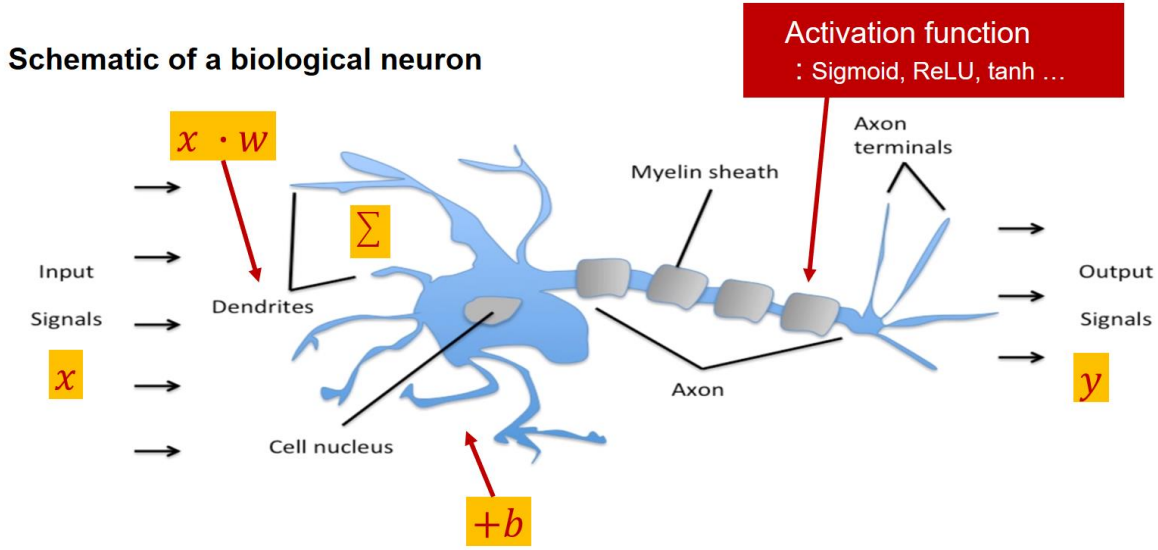
Machine Learning & Deep Learning

Lecture 18

- 퍼셉트론 (Perceptron)
- XOR 해결을 위한 다층 퍼셉트론 (Multilayer Perceptron)
- 인공 신경망 (Artificial Neural Network)
- 역전파 (Back Propagation)
- 기울기 소실 (vanishing gradient)
- 활성화 함수 (Activation Function)
- 고급 경사 하강법 (Advanced Gradient Descent)

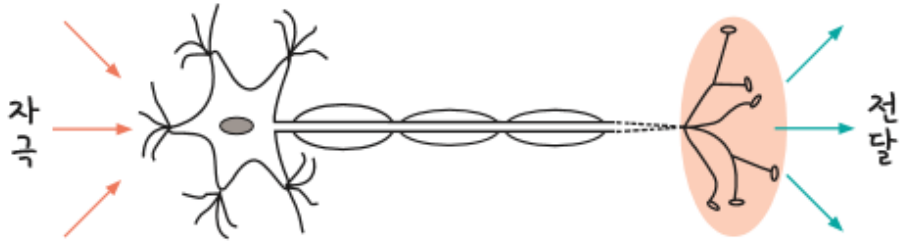
김희숙 (H.S.Kim)

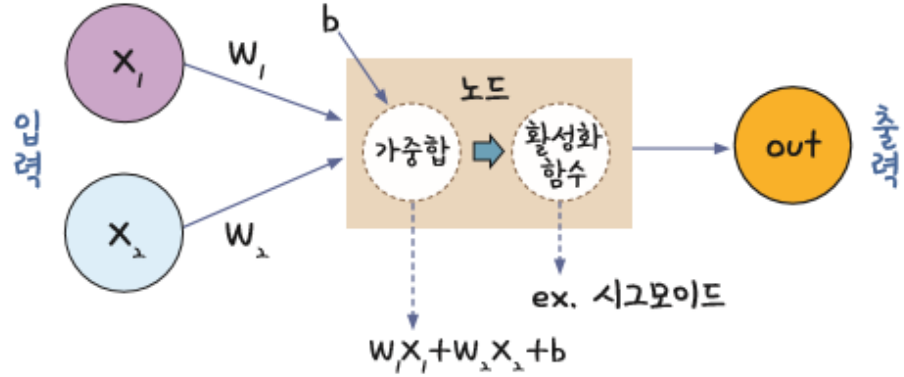
퍼셉트론(perceptron)



- ✓ 인간의 뇌는 치밀하게 연결된 약 1000억 개의 **뉴런(neuron)**으로 이루어져 있음
- ✓ 뉴런과 뉴런 사이에는 **시냅스(synapse)** 라는 연결 부위가 있는데, 신경 말단에서 자극을 받으면, 시냅스에서 화학 물질이 나와 전위 변화를 일으킴
- ✓ 전위가 임계 값을 넘으면 다음 뉴런으로 신호를 전달하고, 임계 값에 미치지 못하면 아무것도 하지 않음

→ '**퍼셉트론(perceptron)**'의 개념과 유사!





뉴런과 퍼셉트론의 비교

- **신경망을 이루는 가장 중요한 기본 단위는 퍼셉트론(perceptron)**
- **퍼셉트론**은 **입력 값**과 **활성화 함수**를 사용해 **출력 값**을 다음으로 넘기는 **가장 작은 신경망 단위**

가중치

✓ 중학교 수학 수준에 맞춰 설명했던 기울기 a 나 y 절편 b 와 같은 용어를 퍼셉트론의 개념에 맞춰 좀 더 '딥러닝답게' 표현해 보면 다음과 같음

$y = ax + b$ (a는 기울기, b는 y 절편)

→ $y = wx + b$ (w는 가중치, b는 바이어스)

✓ 먼저 기울기 a는 퍼셉트론에서는 **가중치**를 의미하는 **w(weight)**로 표기됨

바이어스

✓ y 절편 b는 똑같이 b라고 씀
하지만 y = ax + b의 b가 절편이 아니라
편향, 선입견이라는 뜻인 **바이어스(bias)**를 의미함

가중합

✓ **가중합(weighted sum)** : **입력 값(x)과 가중치(w)의 곱을 모두 더한 다음 바이어스(b)를 더한 값**

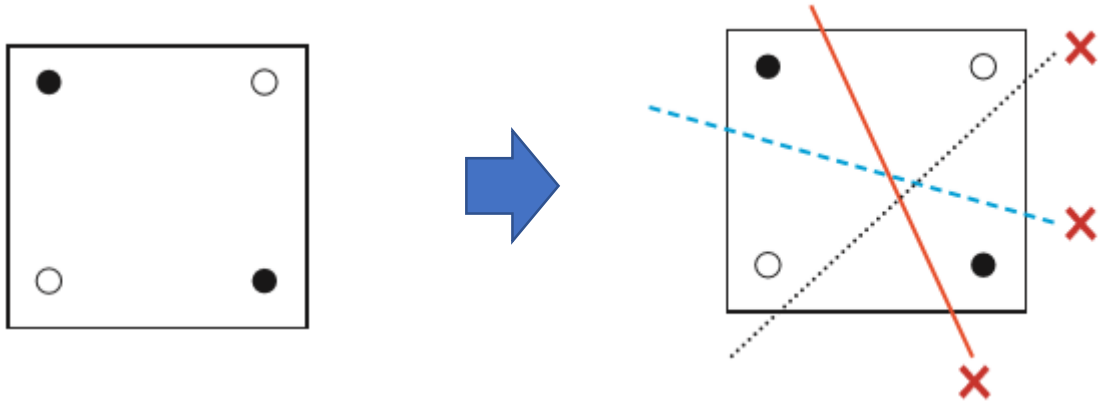
$$y = wx + b \ (w\text{는 가중치}, \ b\text{는 바이어스})$$

✓ 가중합의 결과를 놓고 1 또는 0 을 출력해서 다음으로 보냄

활성화 함수

✓ 여기서 0 과 1을 판단하는 함수가 있는데, 이를 **활성화 함수(activation function)** 라고 함
앞서 배웠던 **시그모이드 함수**가 바로 대표적인 활성화 함수

사각형 종이에 놓인 검은점 2 개와 흰점 2개



선으로는 같은 색끼리 나눌 수 없다 → **퍼셉트론의 한계**

- 사각형 종이에 검은점 2 개와 흰점 2 개가 놓여 있음
- 검은점과 흰점을 분류하기 위해서 4개의 점 사이에 **직선**을 하나 긋는다고 하자
- 이때 직선의 한쪽 편에는 검은점만 있고, 다른 한쪽에는 흰점만 있게끔 선을 그을 수 있을까?
  - ✓ 여러 개의 선을 아무리 그어보아도 하나의 직선으로는 흰점과 검은점을 구분할 수 없음

- 이것이 퍼셉트론의 한계를 설명할 때 등장하는 **XOR(exclusive OR)** 문제

AND 진리표

x_1	x_2	결과값
0	0	0
0	1	0
1	0	0
1	1	1

OR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	1

XOR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	0

- 결과값이 **1 이면 검은점, 0 이면 흰점으로**으로 나타낸 후 조금 전처럼 직선을 그어 조건을 만족할 수 있는지 보자.

AND 진리표

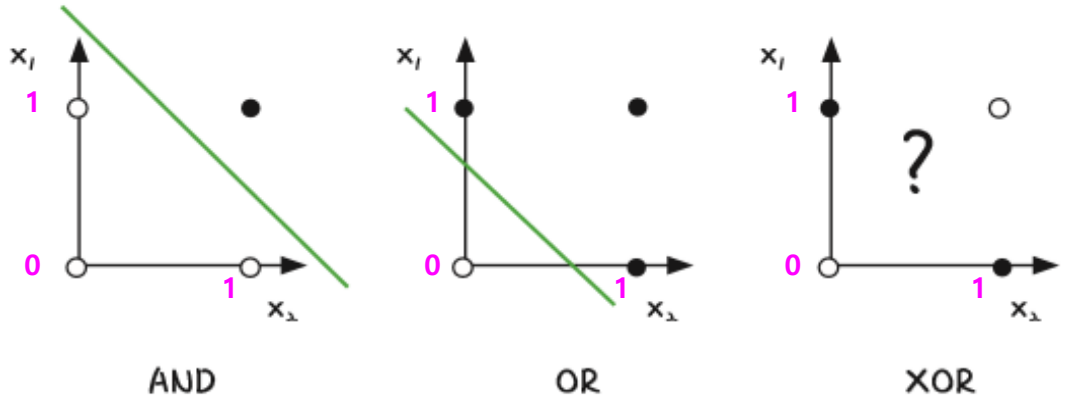
x_1	x_2	결과값
0	0	0
0	1	0
1	0	0
1	1	1

OR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	1

XOR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	0



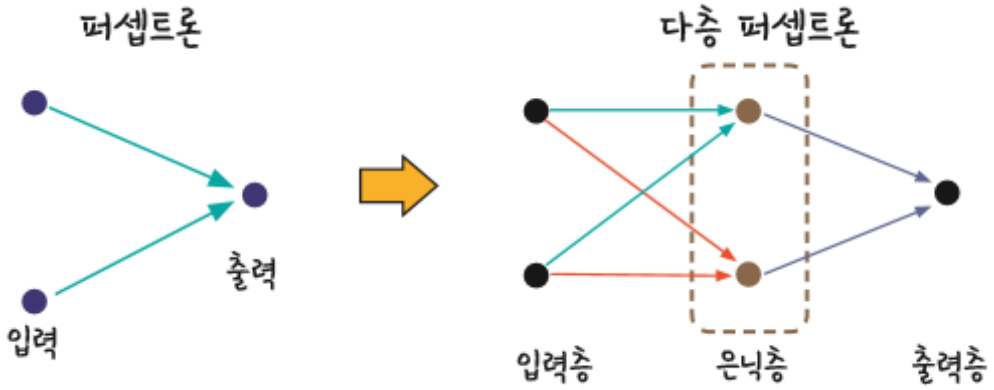
XOR는 선긋기가 불가능

- XOR의 경우, 선을 그어 분류할 수 없음
- 인공지능 분야의 선구자였던 MIT의 마빈 민스키(Marvin Minsky) 교수가 1969년에 발표한 <퍼셉트론즈(Perceptrons)>라는 논문에 나오는 내용
- 이 논문 이후 인공지능 연구는 침체기에 접어들었음

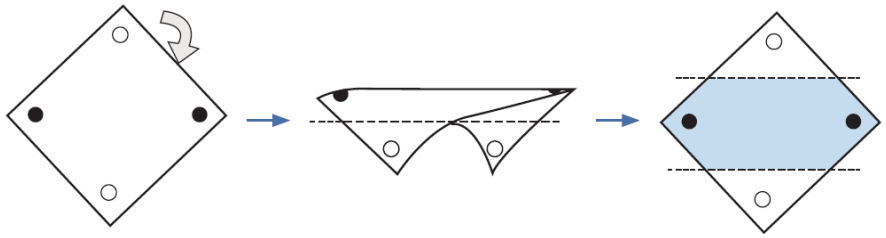
1927. 08. 09 ~ 2016. 01. 24

- 퍼셉트론(perceptron)은 인공신경망의 한 종류로서, 1957년에 코넬 항공 연구소의 프랑크 로젠블라트(Frank Rosenblatt)에 의해 고안되었다.
  이것은 가장 간단한 형태의 피드포워드(Feedforward) 네트워크로도 볼 수 있다.
- 퍼셉트론은 입력된 패턴을 한번에 하나씩 비교하면서 원하는 출력값이 나올 때까지 가중치를 조절하면서 학습하는 개념이다.
- 1960년대 후반에 당시 뇌 신경망의 일부를 모방한 형태인 인공신경망 퍼셉트론(perceptron)의 한계를 밝혔다. **단층 퍼셉트론**은 XOR 연산이 불가능하지만, **다층 퍼셉트론(multilayer perceptron)**으로는 XOR 연산이 가능한데 각 weight, bias를 찾을 방법이 없다는 것이다.
  퍼셉트론에 대해 낙관적으로 기대했지만 **마빈 민스키(Marvin Minsky) 교수**가 퍼셉트론이 학습할 수 있는 범위에 한계가 있음을 입증하면서 새로운 국면을 맞이하게 됐다.
- 이후, 1986년 **제프리 힌튼(Geoffrey Hinton)** 교수팀이 XOR 문제 해결 → **다층 퍼셉트론(MLP; MultiLayer Perceptron), 오차 역전파(Back Propagation)** 구현

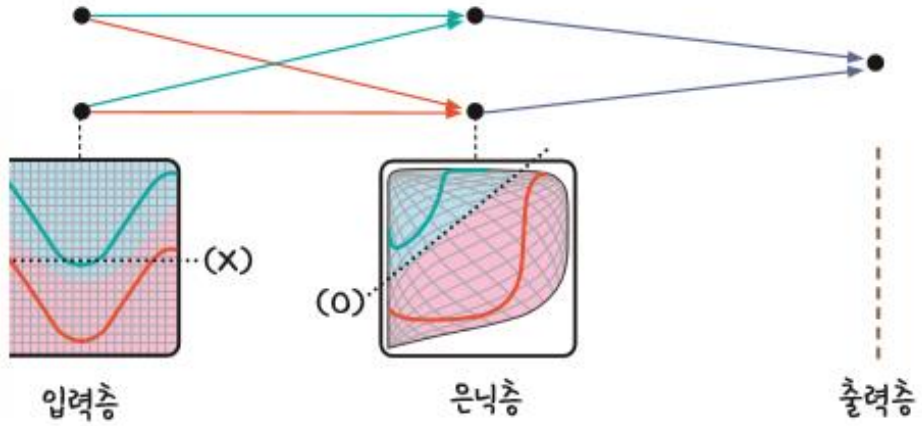
- XOR 문제를 해결하기 위해서 **2 개의 퍼셉트론을 한 번에 계산할 수 있어야 함**
- 이를 가능하게 하려면 숨어있는 층, 즉 **은닉층(hidden layer)**을 만들면 됨



퍼셉트론에서 **다층 퍼셉트론(MLP; MultiLayer Perceptron)**으로

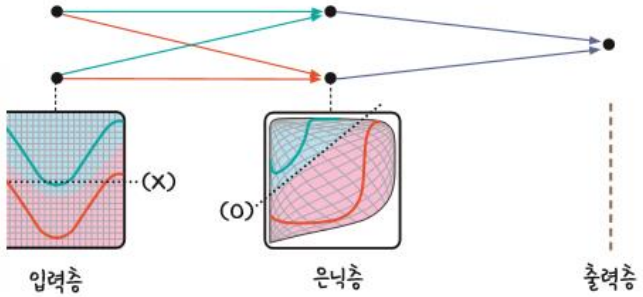


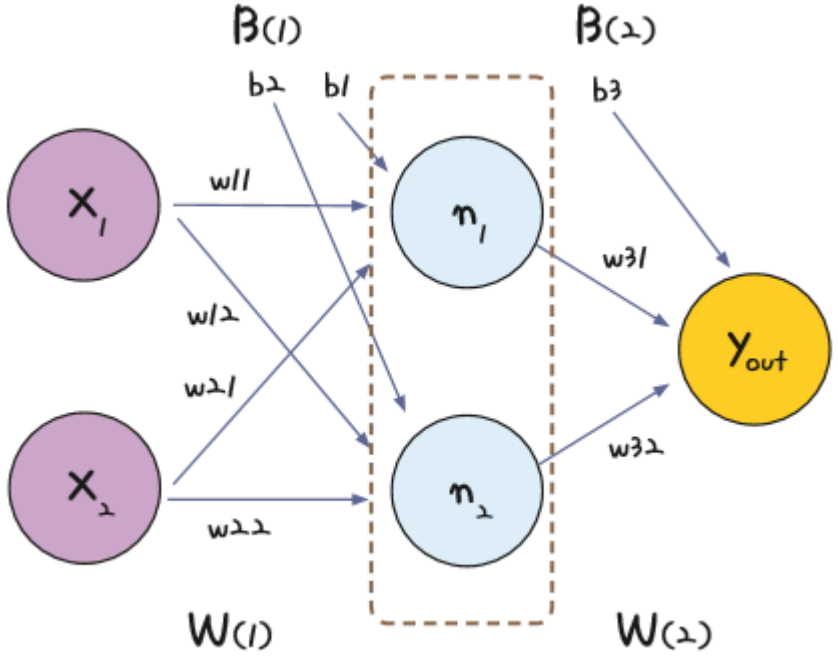
- 입력층과 은닉층의 그래프를 집어넣어 보면 아래 그림과 같음
- **은닉층이 좌표 평면을 왜곡**시키는 결과를 가져옴 → **두 영역을 가로지르는 선이 직선으로 바뀜**



은닉층의 공간 왜곡(https://goo.gl/8qEGHD 참조)

- **다층 퍼셉트론**이 입력층과 출력층 사이에 숨어있는 **은닉층**을 만드는 것을 도식으로 나타내면 아래 그림과 같음



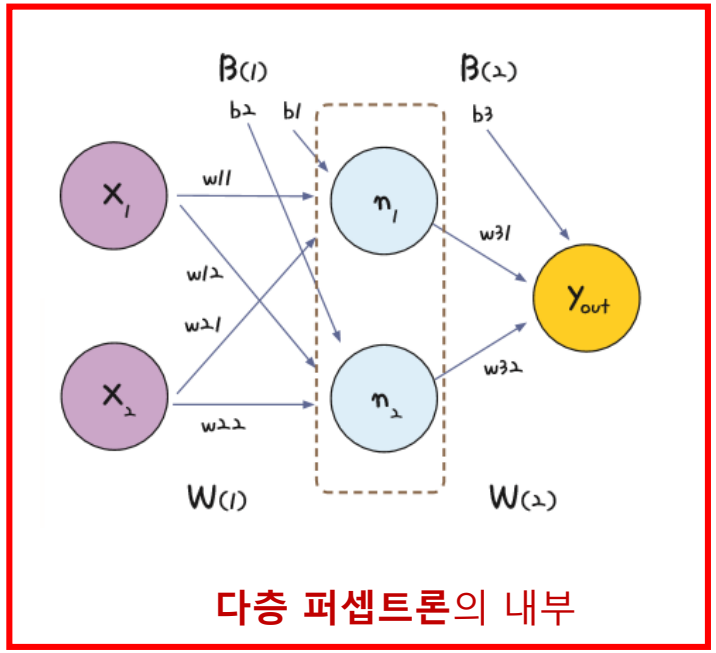


다층 퍼셉트론의 내부

- n_1 과 n_2 의 은닉층 값은 각각 단층 퍼셉트론의 값과 같음

시그모이드

$$n_1 = \sigma\left(x_1 w_{11} + x_2 w_{21} + b_1\right)$$
$$n_2 = \sigma\left(x_2 w_{21} + x_2 w_{22} + b_2\right)$$



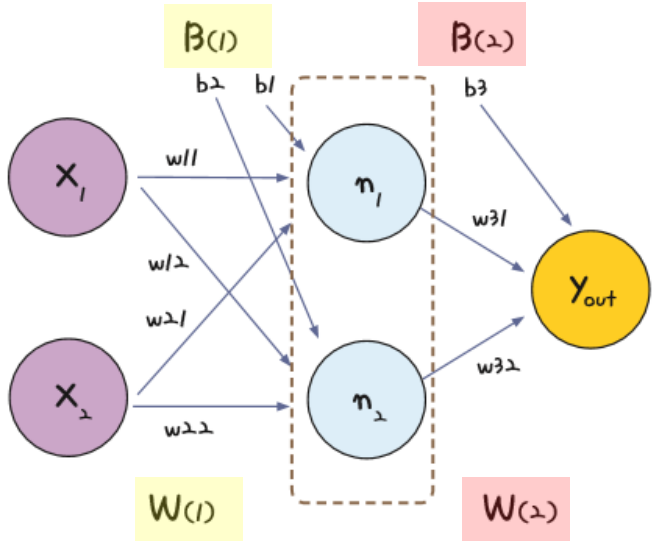
다층 퍼셉트론의 내부

- 은닉층의 결과값이 출력층으로 보내짐
- 출력층에서는 역시 시그모이드 함수를 통해 y 값이 정해짐

$$y_{\text{out}} = \sigma\left(n_1 w_{31} + n_2 w_{32} + b_3\right)$$

시그모이드

- 이제 각각의 **가중치(w)**와 **바이어스(b)**의 값을 정할 차례
- 은닉층을 포함해 **가중치(w) 6개**와 **바이어스(b) 3개**가 필요함



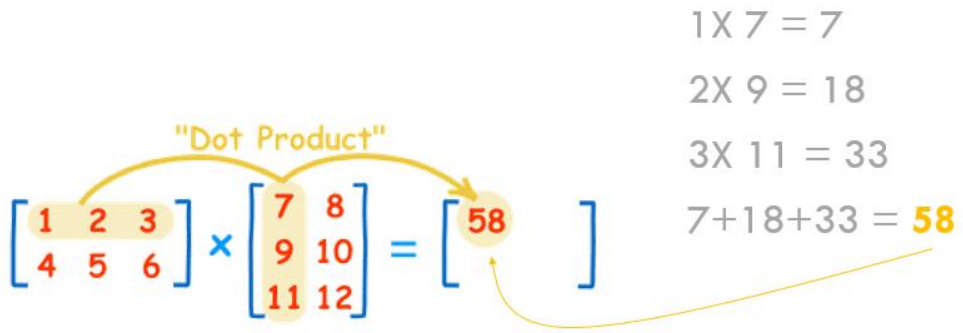
$$n_1 = \sigma\left(x_1 w_{11} + x_2 w_{21} + b_1\right)$$
$$n_2 = \sigma\left(x_2 w_{21} + x_2 w_{22} + b_2\right)$$

$$y_{\text{out}} = \sigma\left(n_1 w_{31} + n_2 w_{32} + b_3\right)$$

$$W(1) = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad B(1) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \quad B(2) = [b_3]$$

행렬의 곱 [참고]



$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

$$\begin{pmatrix} x_1 & x_2 & x_3 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_1w_1 + x_2w_2 + x_3w_3 \end{pmatrix}$$

$$H(x) = Wx + b \quad \rightarrow \quad H(X) = XW$$

행렬의 곱 [참고]

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$w_1x_1 + w_2x_2 + w_3x_3 + \ldots + w_nx_n$$

Instance

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

$$H(X) = XW$$

행렬의 곱 [참고]

x_1	x_2	x_3	Y
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[5, 3] [3, 1] [5, 1]

$$H(X) = XW$$

행렬의 곱 [참고]

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \\ x_{51}w_1 + x_{52}w_2 + x_{53}w_3 \end{pmatrix}$$

[n, 3] [3, 1] [n, 1]

None None

$$H(X) = XW$$

행렬의 곱 [참고]

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot ? = \begin{pmatrix} x_{11}w_{11}+x_{12}w_{21}+x_{13}w_{31} & x_{11}w_{12}+x_{12}w_{22}+x_{13}w_{32} \\ x_{21}w_{11}+x_{22}w_{21}+x_{23}w_{31} & x_{21}w_{12}+x_{22}w_{22}+x_{23}w_{32} \\ x_{31}w_{11}+x_{32}w_{21}+x_{33}w_{31} & x_{31}w_{12}+x_{32}w_{22}+x_{33}w_{32} \\ x_{41}w_{11}+x_{42}w_{21}+x_{43}w_{31} & x_{41}w_{12}+x_{42}w_{22}+x_{43}w_{32} \\ x_{51}w_{11}+x_{52}w_{21}+x_{53}w_{31} & x_{51}w_{12}+x_{52}w_{22}+x_{53}w_{32} \end{pmatrix}$$

[n, 3] [3, 2] [n, 2]

Output : 2

$$H(X) = XW$$

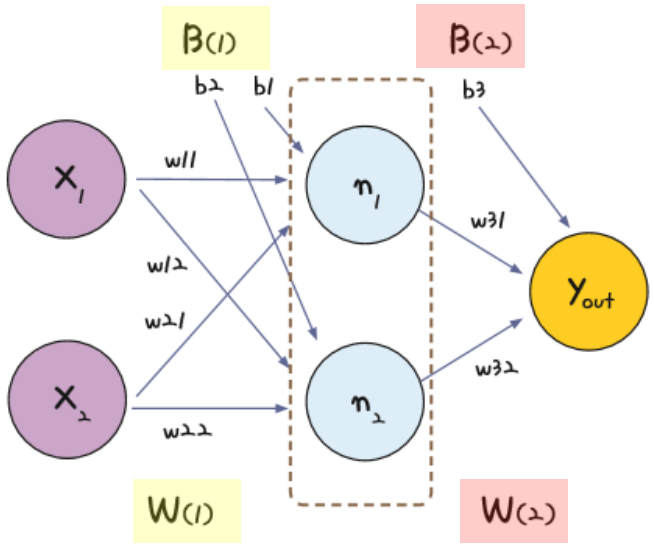
행렬의 곱 [참고]

$$\begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \\ x_{51} & x_{52} & x_{53} \end{pmatrix} \cdot \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} = \begin{pmatrix} x_{11}w_{11} + x_{12}w_{21} + x_{13}w_{31} & x_{11}w_{12} + x_{12}w_{22} + x_{13}w_{32} \\ x_{21}w_{11} + x_{22}w_{21} + x_{23}w_{31} & x_{21}w_{12} + x_{22}w_{22} + x_{23}w_{32} \\ x_{31}w_{11} + x_{32}w_{21} + x_{33}w_{31} & x_{31}w_{12} + x_{32}w_{22} + x_{33}w_{32} \\ x_{41}w_{11} + x_{42}w_{21} + x_{43}w_{31} & x_{41}w_{12} + x_{42}w_{22} + x_{43}w_{32} \\ x_{51}w_{11} + x_{52}w_{21} + x_{53}w_{31} & x_{51}w_{12} + x_{52}w_{22} + x_{53}w_{32} \end{pmatrix}$$

[n, 3] **[3, 2]** [n, 2]

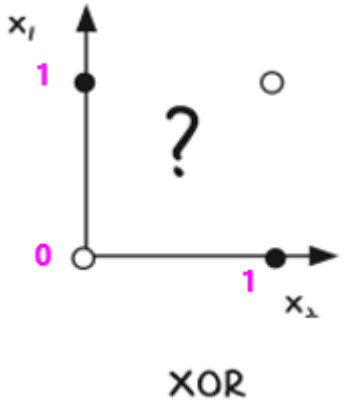
$$H(X) = XW$$

- **Example (1)**
  - ✓ XOR를 만족하는 **가중치(w) 6개와 바이어스(b) 3개의 조합은 무수히 많음**



XOR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	0



$$w(1) = \begin{bmatrix} 5, -7 \\ 5, -7 \end{bmatrix}, \quad b(1) = \begin{bmatrix} -8 \\ 3 \end{bmatrix}$$

$$w(2) = \begin{bmatrix} -11 \\ -11 \end{bmatrix}, \quad b(2) = 6$$

- **Example (1)**

XOR 진리표

x_1	x_2	결과값
0	0	0
0	1	1
1	0	1
1	1	0

$$w(1) = \begin{bmatrix} 5, -7 \\ 5, -7 \end{bmatrix}, \quad b(1) = \begin{bmatrix} -8 \\ 3 \end{bmatrix}$$

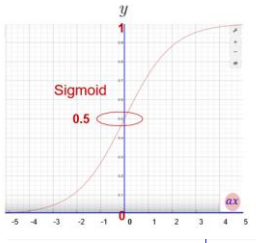
X11	X12	*	W11	W12	=	X11*W11 + X12*W21	X11*W12 + X12*W22
X21	X22		W21	W22		X21*W11 + X22*W21	X21*W12 + X22*W22
X31	X32			[2, 2]		X31*W11 + X32*W21	X31*W12 + X32*W22
X41	X42					X41*W11 + X42*W21	X41*W12 + X42*W22
[4, 2]						[4, 2]	

$$\left(\begin{bmatrix}0,0\\0,1\\1,0\\1,1\end{bmatrix} * \begin{bmatrix}5,-7\\5,-7\end{bmatrix}\right) + \begin{bmatrix}-8\\3\end{bmatrix}$$

XOR

$$\left(\begin{bmatrix}0,1\\0,0\\0,0\\1,0\end{bmatrix} * \begin{bmatrix}-11\\-11\end{bmatrix}\right) + 6$$

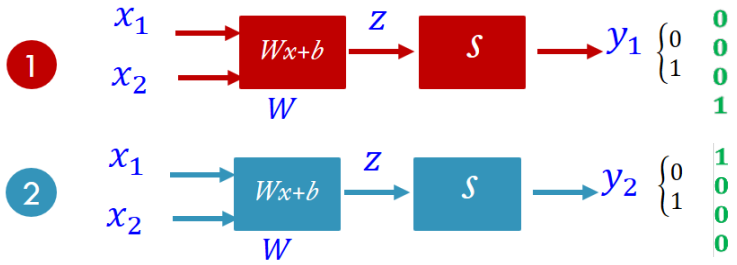
은닉층



$H(x) = Wx + b$

W=(5, 5), b=-8 일때
(0*5 + 0*5) + (-8) = -8 ==> sigmoid(-8) = 0
(0*5 + 1*5) + (-8) = -3 ==> sigmoid(-3) = 0
(1*5 + 0*5) + (-8) = -3 ==> sigmoid(-3) = 0
(1*5 + 1*5) + (-8) = 2 ==> sigmoid(2) = 1

$w = \begin{bmatrix}5\\5\end{bmatrix}, b = -8$

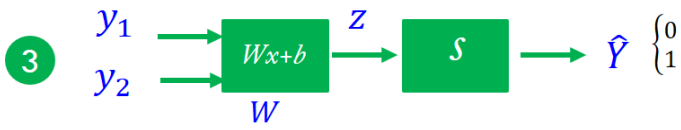


W=(-7, -7), b=3 일때,
(0*-7 + 0*-7) + 3 = 3 ==> sigmoid(3) = 1
(0*-7 + 1*-7) + 3 = -4 ==> sigmoid(-4) = 0
(1*-7 + 0*-7) + 3 = -4 ==> sigmoid(-4) = 0
(1*-7 + 1*-7) + 3 = -11 ==> sigmoid(-11) = 0

$w = \begin{bmatrix}-7\\-7\end{bmatrix}, b = 3$

$w = \begin{bmatrix}-11\\-11\end{bmatrix}, b = 6$

W=(-11, -11), b=6 일때
(0*-11 + 1*-11) + 6 = -5 ==> sigmoid(-5) = 0
(0*-11 + 0*-11) + 6 = 6 ==> sigmoid(6) = 1
(0*-11 + 0*-11) + 6 = 6 ==> sigmoid(6) = 1
(1*-11 + 0*-11) + 6 = -5 ==> sigmoid(-5) = 0



x_1	x_2	y1	y2	$\hat{y}$	XOR
0	0	0	1	0	0 (-)
0	1	0	0	1	1 (+)
1	0	0	0	1	1 (+)
1	1	1	0	0	0 (-)

- **Example (2)**
  - ✓ XOR를 만족하는 **가중치(w) 6개와 바이어스(b) 3개의 조합은 무수히 많음**
- 아래의 값을 가중치(w)와 바이어스(b)로 설정해서 XOR 결과값이 나오는지 점검해 보자.

1단계

$$w(1) = \begin{bmatrix} -2, 2 \\ -2, 2 \end{bmatrix}, \quad b(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

x_1	x_2	n_1	n_2	y_{out}	우리가 원하는 값
0	0	$\sigma(0*(-2)+0*(-2)+3) = 3 \rightarrow 1$	$\sigma(0*2+0*2-1) = -1 \rightarrow 0$	$\sigma(1*1+0*1-1) = 0$	0
0	1	$\sigma(0*(-2)+1*(-2)+3) = 1 \rightarrow 1$	$\sigma(0*2+1*2-1) = 1 \rightarrow 1$	$\sigma(1*1+1*1-1) = 1$	1
1	0	$\sigma(1*(-2)+0*(-2)+3) = 1 \rightarrow 1$	$\sigma(1*2+0*2-1) = 1 \rightarrow 1$	$\sigma(1*1+1*1-1) = 1$	1
1	1	$\sigma(1*(-2)+1*(-2)+3) = -1 \rightarrow 0$	$\sigma(1*2+1*2-1) = 3 \rightarrow 1$	$\sigma(0*1+1*1-1) = 0$	0

NAND OR AND XOR 최종 출력 값

2 단계

$$w(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b(2) = -1$$

가중치(weight)와 바이어스(bias)는 다음과 같이 가정한 후,

XOR 문제를 해결하는 다층 퍼셉트론(multilayer perceptron)을 코딩으로 구현해 보자.

1단계

$$w(1) = \begin{bmatrix} -2, 2 \\ -2, 2 \end{bmatrix}, \quad b(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

2 단계

$$w(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad b(2) = -1$$

18_(26 page) 강의용 _XOR_Multilayer_perceptron.ipynb

```
입력 값: (0, 0)
퍼셉트론 통과 :  3 ,   퍼셉트론 통과 :  -1 ,   퍼셉트론 통과 :  0 ,   --> 최종 출력 값: 0
입력 값: (0, 1)
퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   --> 최종 출력 값: 1
입력 값: (1, 0)
퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   --> 최종 출력 값: 1
입력 값: (1, 1)
퍼셉트론 통과 :  -1 ,   퍼셉트론 통과 :  3 ,   퍼셉트론 통과 :  0 ,   --> 최종 출력 값: 0
```

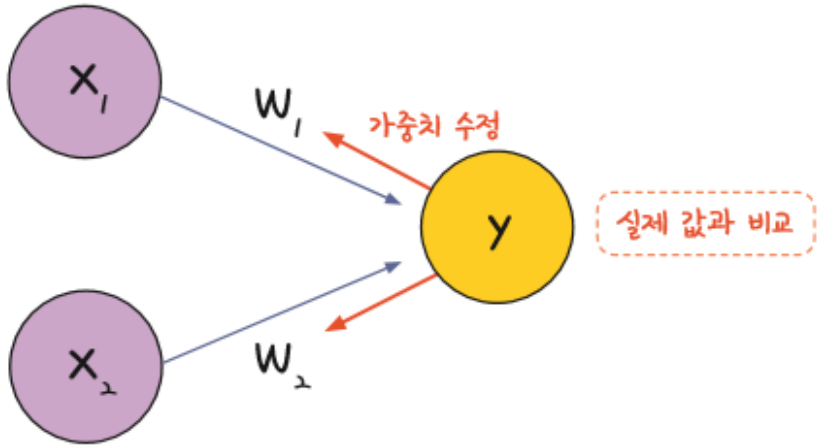
```
입력 값: (0, 0)
퍼셉트론 통과 :  3 ,   퍼셉트론 통과 :  -1 ,   퍼셉트론 통과 :  0 ,   --> 최종 출력 값: 0
입력 값: (0, 1)
퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   --> 최종 출력 값: 1
입력 값: (1, 0)
퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   퍼셉트론 통과 :  1 ,   --> 최종 출력 값: 1
입력 값: (1, 1)
퍼셉트론 통과 :  -1 ,   퍼셉트론 통과 :  3 ,   퍼셉트론 통과 :  0 ,   --> 최종 출력 값: 0
```

- 우리가 원하는 XOR 문제의 정답이 도출됨
- 퍼셉트론 하나로 해결되지 않던 문제를 **은닉층**을 만들어 해결
- **은닉층을 여러 개 쌓아 올려** 복잡한 문제를 해결하는 과정이 뉴런이 복잡한 과정을 거쳐 사고를 낳는 사람의 **신경망을 닮음**
- 그래서 이 방법을 **Artificial Neural Network (인공 신경망)**이라고 부르기 시작했고, 이를 간단히 줄여서 **Neural Network(신경망)**이라고 통칭

- 퍼셉트론으로 해결되지 않던 문제를 **신경망**을 이용해 해결했음
- 현재 신경망 내부의 **가중치(weight)**는 **오차 역전파(Back Propagation)** 방법을 사용해 최적의 값을 업데이트할 수 있음
- **오차 역전파**는 **경사 하강법**의 확장 개념
- 앞서 XOR 문제를 해결할 때 정답에 해당하는 가중치(weight)와 바이어스(bias)를 미리 알아본 후 구현하였음 → **실제 프로젝트에서는 경사 하강법을 이용**
- **임의의 가중치(weight)**를 선언하고 결과값을 이용해 **오차**를 구한 뒤 이 **오차가 최소인 지점으로** 계속해서 조금씩 **가중치(weight)를 이동**시킴
- 이 **오차가 최소가 되는 점**(미분했을 때 기울기가 0이 되는 지점)**을 찾으면** 그것이 바로 **우리가 알고자 하는 가중치(weight)**이다.

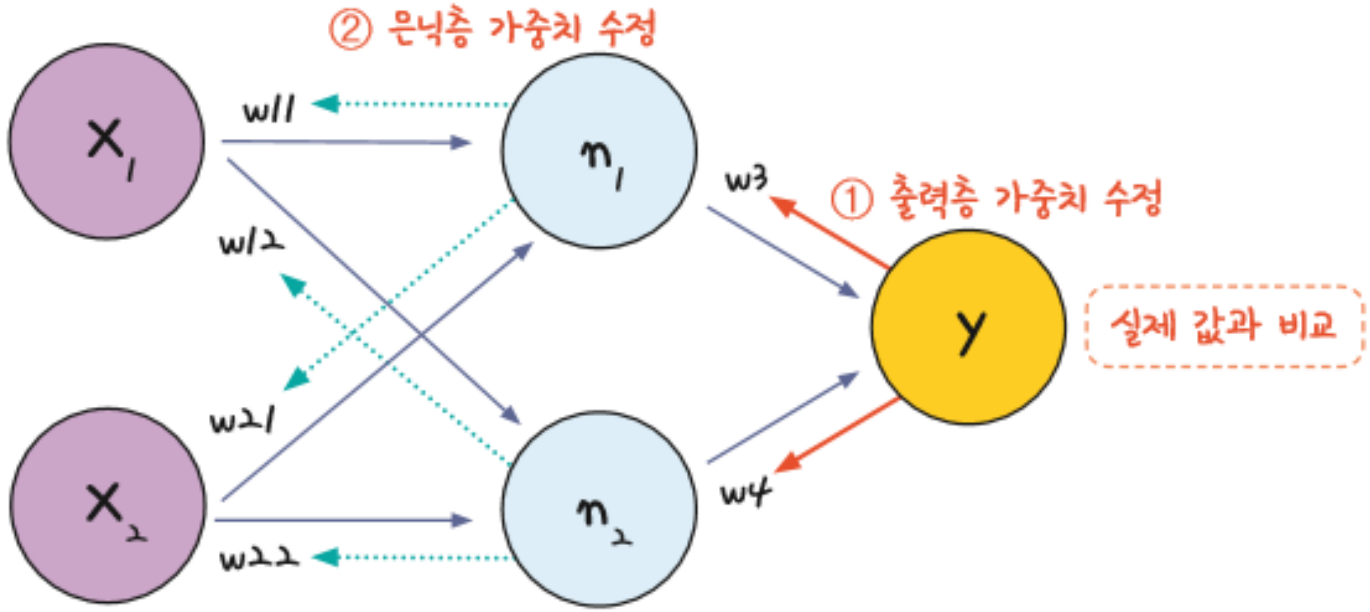
- 앞서 배운 **경사 하강법**은 **입력과 출력이 하나일 때**, 즉 **'단층 퍼셉트론'**일 경우였음
  그런데 이번에는 숨어 있는 은닉층이 하나 더 생김

- **단층 퍼셉트론**에서 결과값을 얻으면 **오차**를 구해 이를 기반으로 **앞 단계에서 정한 가중치를 조정**하는 것과 마찬가지로



단층 퍼셉트론에서의 오차 수정

- **다층 퍼셉트론** 역시 결과값의 오차를 구해 이를 토대로 하나 앞선 가중치를 차례로 거슬러 올라가며 조정해 감



다층 퍼셉트론에서의 오차 수정

- 그러다 보니 오차를 최적화하기 위한 계산 방향이 **출력층에서 시작해서 뒤에서 앞으로** 진행됨

  → **오차 역전파(Back Propagation)**라고 부름

- **오차 역전파 구동 방식의 정리**

  1) **임의의 초기 가중치($w_{(1)}$)를 준 뒤 결과(y_{out})를 계산한다.**
  2) **계산 결과와 우리가 원하는 값 사이의 오차를 구한다.**
  3) **경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 업데이트한다.**
  4) **1)~3) 과정을 더이상 오차가 줄어들지 않을 때까지 반복한다.**

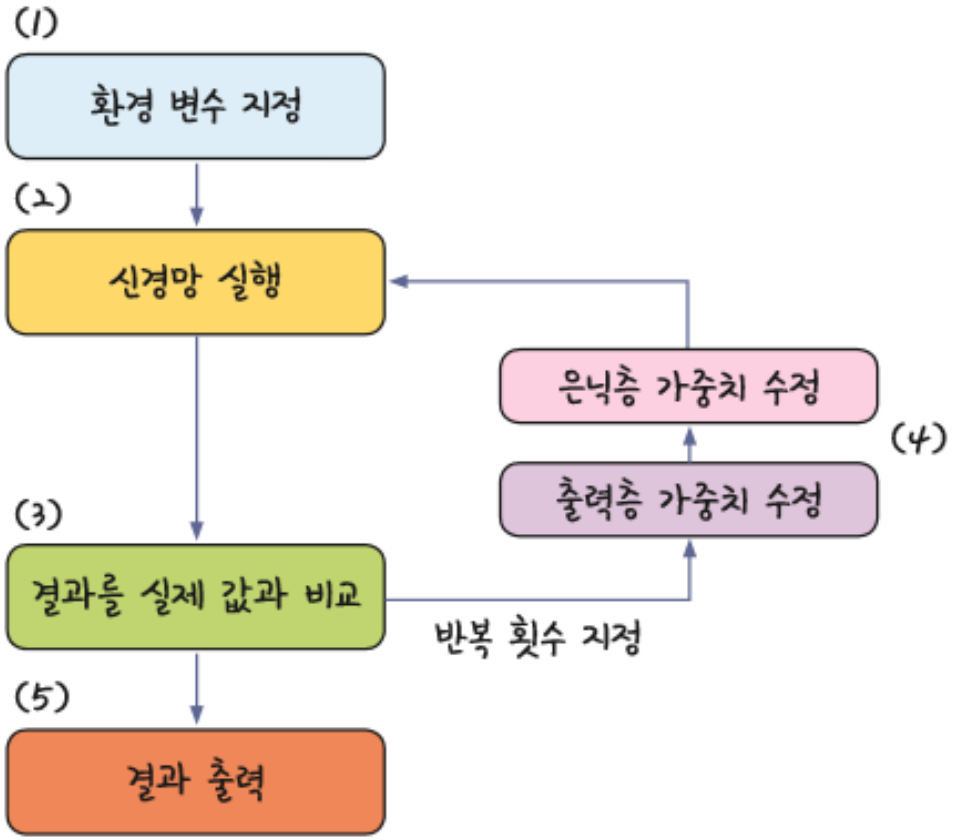
- 여기서 '오차가 작아지는 방향으로 업데이트한다'는 의미는
  미분 값이 0 에 가까워지는 방향으로 나아간다는 뜻

- 즉, '기울기가 0 이 되는 방향'으로 나아가야 하는데,
  이 말은 **가중치 w 에서 기울기(오차)를 뺐을 때 가중치 w의 변화가 전혀 없는 상태를 말함**

- 따라서 오차 역전파를 다른 방식으로 표현하면 **가중치에서 기울기를 빼도 가중치 값의 변화가 없을 때까지 계속해서 가중치 수정 작업을 반복**하는 것

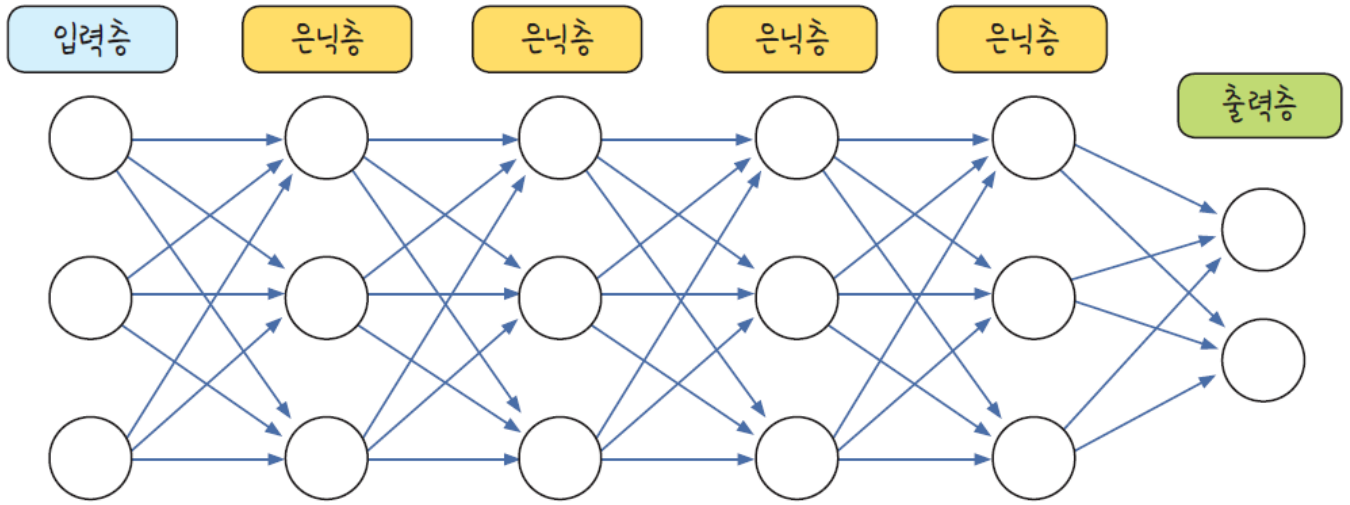
새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값!

$$W(t+1) = Wt - \frac{\partial 오차}{\partial W}$$



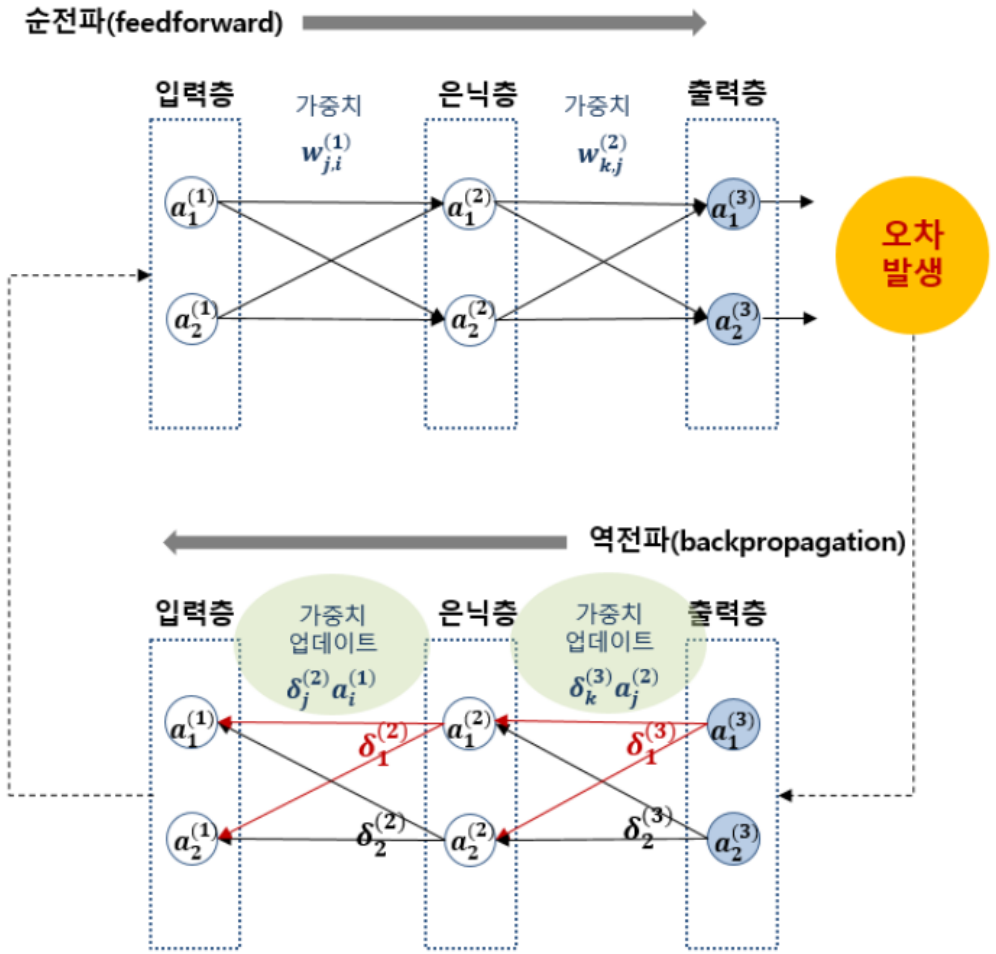
신경망의 구현 과정

- 다층 퍼셉트론이 **오차 역전파**를 만나 **신경망**이 되었고, 신경망은 **XOR문제를 가볍게 해결함**
- 이제 신경망을 차곡차곡 쌓아올리면 마치 사람처럼 생각하고 판단하는 인공지능이 금방이라도 완성될 것처럼 보임

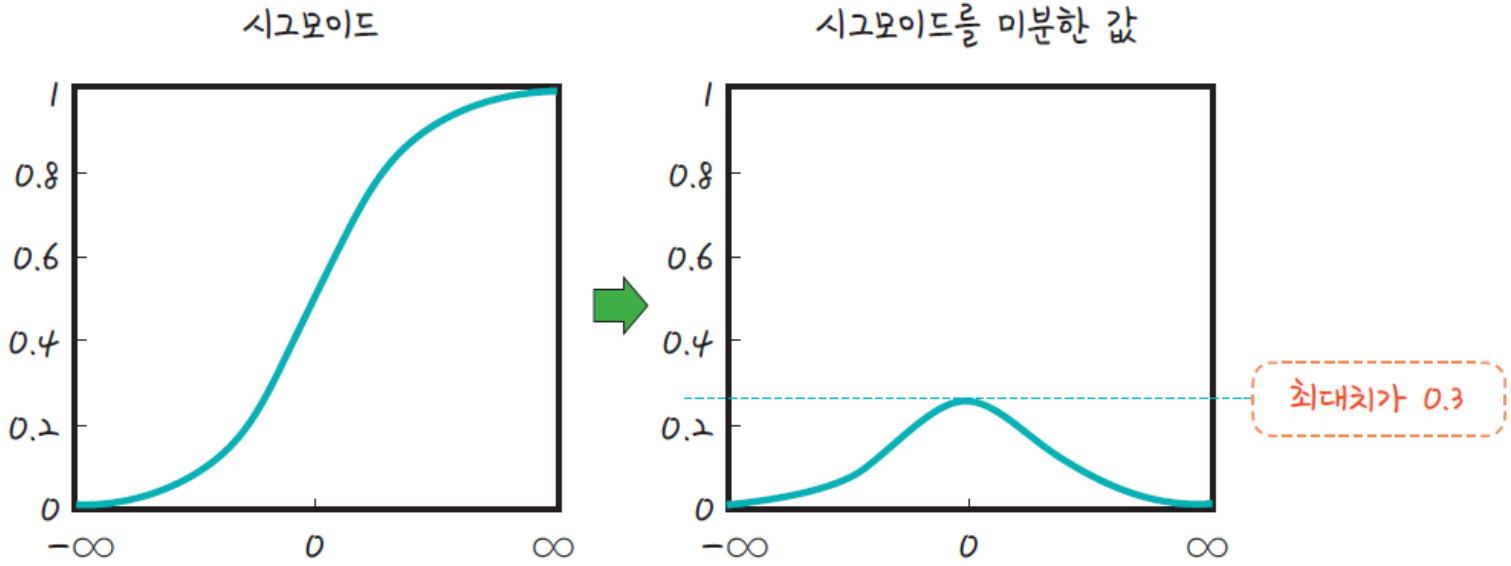


다층 확장

- **오차 역전파** : 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- **가중치를 수정**하려면 **미분** 값, 즉 **기울기**가 필요하다고 배움

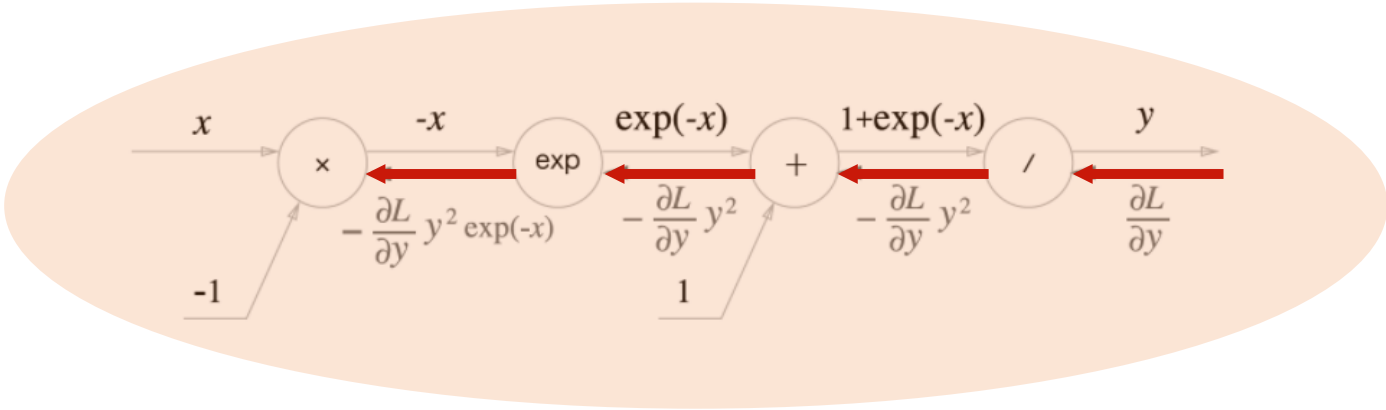


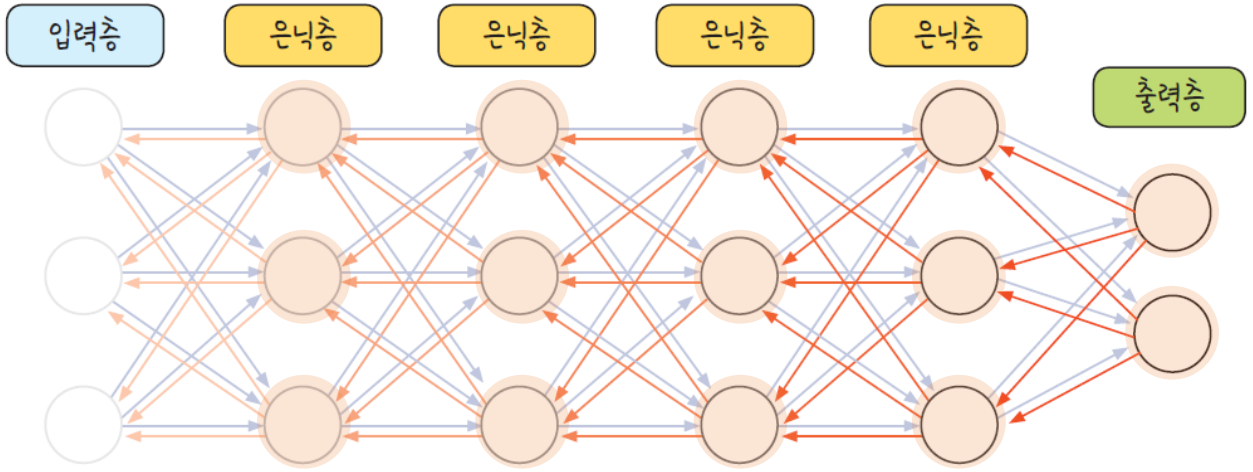
- **기울기 소실(vanishing gradient) 문제**가 발생하기 시작한 것은 활성화 함수로 사용된 **시그모이드 함수의 특성** 때문임
- 여러 층을 거칠수록 기울기가 사라져 가중치를 수정하기가 어려워지는 것임
- **1 보다 작으므로 계속 곱하다 보면 0 에 가까워짐**



시그모이드의 미분

- 오차 역전파 : 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- 가중치를 수정하려면 **미분** 값, 즉 **기울기**가 필요하다고 배움

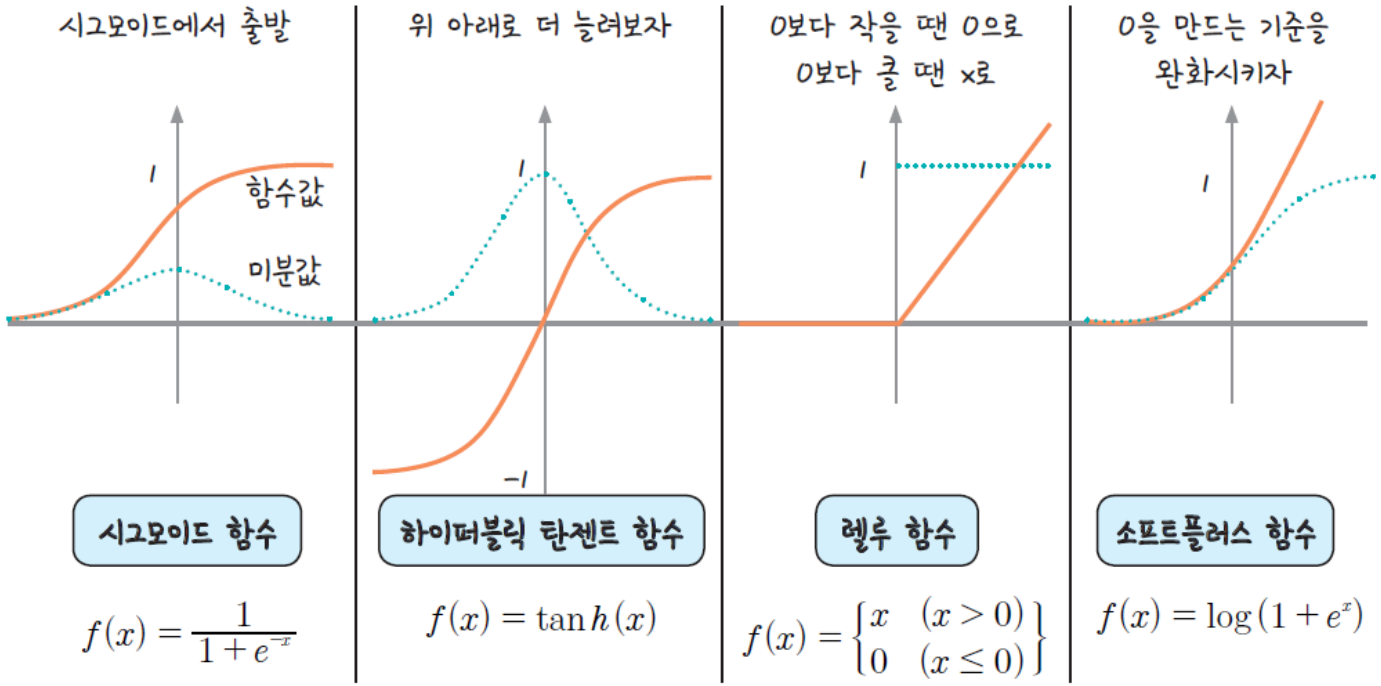




시그모이드 값은 1 보다 작고, 미분 값은 최대치가 0.3이므로 계속 곱하다 보면 0 에 가까워짐

- 이를 해결하고자 **활성화 함수(Activation function)**를 시그모이드가 아닌 여러 함수로 대체하기 시작함

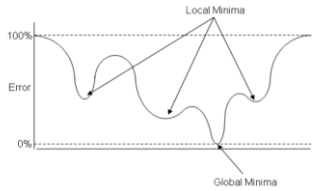
$$y = \frac{1}{1 + e^{(ax+b)}}$$

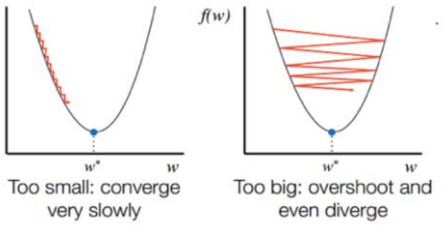


활성화 함수(Activation function)

- 경사 하강법에서는 학습시 스텝의 크기 (step size)가 중요하다.
  - ✓ 학습률(learning rate)이 너무 작을 경우
    - 알고리즘이 수렴하기 위해 반복해야 하는 값이 많으므로 학습 시간이 오래 걸린다.
    - 지역 최소값(local minimum)에 수렴할 수 있다.
  - ✓ 학습률 (learning rate) 이 너무 클 경우

    학습 시간은 적게 걸리나, 스텝이 너무 커서 전역 최소값(global minimum)을 가로질러 반대편으로 건너뛰어 최소값에서 멀어질 수 있다.

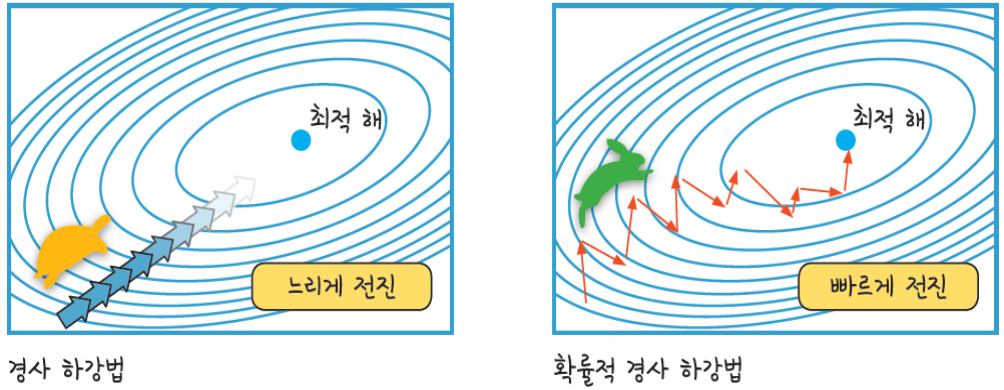




- 가중치를 업데이트하는 경사 하강법(Gradient descent)은 정확하게 가중치를 찾아가지만, 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있음
- 이러한 점을 보완한 고급 경사 하강법이 등장하면서 딥러닝의 발전 속도는 더 빨라짐

확률적 경사 하강법(SGD; Stochastic Gradient Descent)

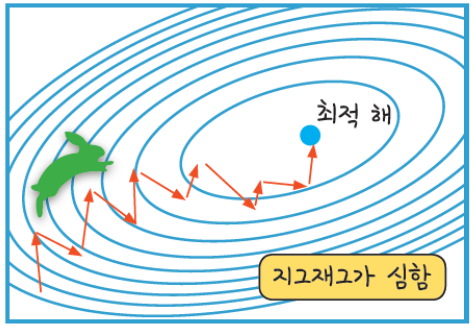
- 전체 데이터로 미분을 이용한 기울기를 계산하는 것이 아니라, **랜덤하게 추출한 일부 데이터**만 사용하여 기울기를 계산하여 경사하강법을 적용함
- **일부 데이터를 사용**하여 기울기를 계산하므로 **더 빨리, 더 자주 업데이트를** 하는 것이 가능해짐
- **랜덤한 일부 데이터를 사용**하는 만큼 **확률적 경사 하강법은 중간 결과의 진폭이 크고 불안정해 보일 수 있음**
- 속도가 확연히 빠르면서도 최적 해에 근사한 값을 찾아낸다는 장점 덕분에 **경사 하강법의 대안으로 사용**되고 있음



경사 하강법과 **확률적 경사 하강법**의 비교

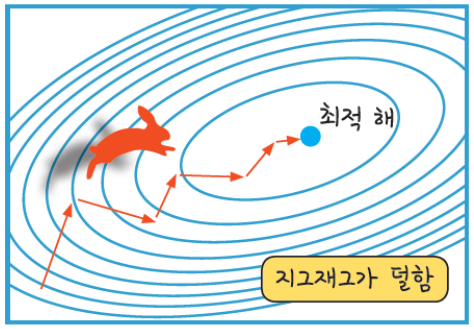
Momentum(모멘텀)

- **모멘텀(momentum)**이란 단어는 '관성, 탄력, 가속도'라는 뜻
- 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전에 **바로 앞 수정 값과 방향(+, -)**을 참고하여 **같은 방향으로, 일정한 비율만 수정**되게 하는 방법
- 경사 하강법은 기본적으로 learning rate의 크기만큼 gradient의 방향으로 이동하기 때문에 대부분의 경우에 SGD는 최적 해 방향으로 곧장 이동하는 것이 아니라, 그림과 같이 진동하며 이동한다. **모멘텀**은 수정 방향이 **양수(+) 방향으로 한 번, 음수(-) 방향으로 한 번 지그재그로 일어나는 현상이 줄어들고, 이전 이동 값을 고려하여 일정 비율만큼만 효과**를 낼 수 있음



확률적 경사 하강법

SGC

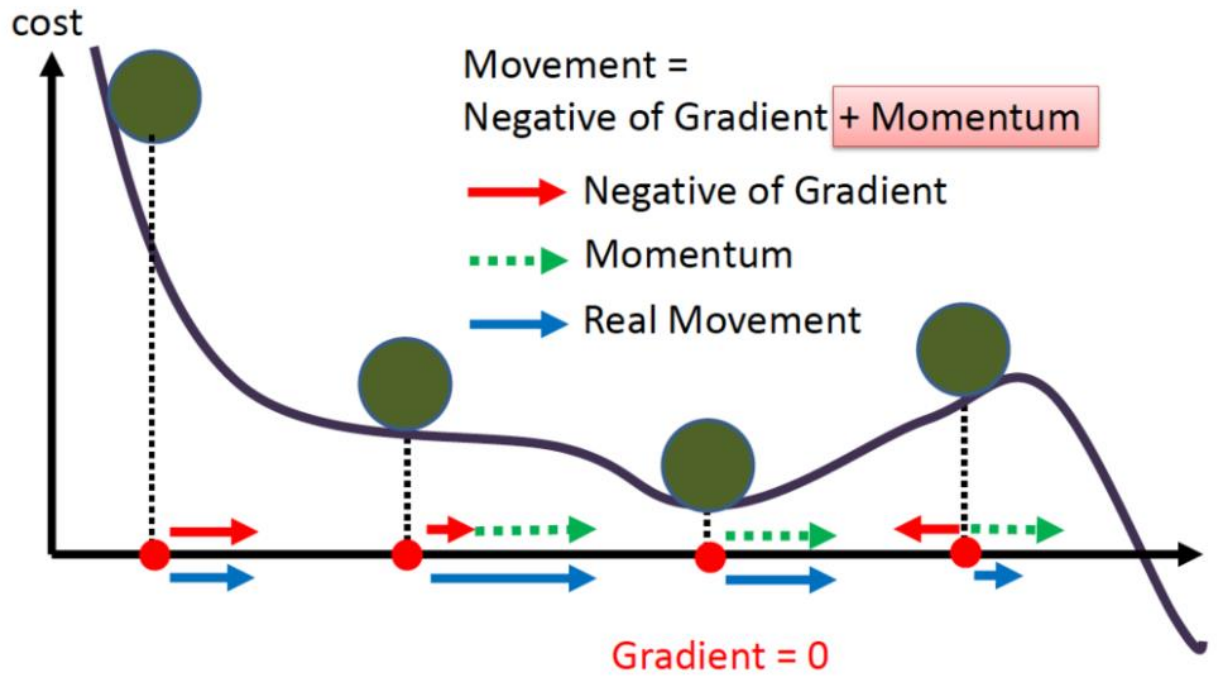


모멘텀을 적용한 확률적 경사 하강법

SGC + 모멘텀

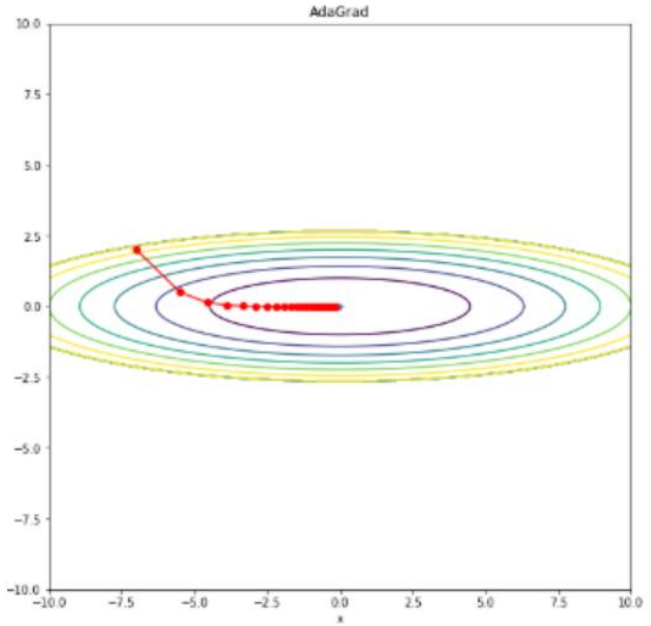
Momentum(모멘텀)

- 이와 같은 문제점을 해결하기 위해 사용하는 방법이 모멘텀이다.
- 쉽게 말해 기울기에 관성을 부과하여 작은 기울기는 쉽에 넘어갈 수 있도록 만든 것이다.
- 즉, 공을 예로 들면 언덕에서 공을 굴렸을 때, 낮은 언덕은 공의 관성을 이용하여 쉽게 넘어갈 수 있게 하여 지역 최소값을 탈출 할 수 있게 한다는 뜻이다.



AdaGrad(아다그라드)

- 아다그라드는 **w(가중치)의 업데이트 횟수에 따라 학습률(Learning rate)을 조절하는 옵션**이 추가된 최적화 방법
- **많이 변화한 w(가중치)들에 대해서는 학습률(learning rate)을 작게**하고,
  많이 변화하지 않은 w(가중치)들은 학습률(learning rate)을 크게하여 loss값을 줄인다.



RMSProp

- **AdaGrad 는 최소값에 도달하기 전에 학습률을 0에 수렴하게 만들 수도 있는 문제점**이 있다.
- **문제점을 해결하기 위해서 RMSProp이 제안됨**
- 정식 논문은 없지만 **Geoffrey Hinton이 코세라 강의에서 제안**

- Root Mean Square의 약자
  - ✓ AdaGrad의 수식 중에서 Root Square(제곱근)에, 가중평균 Mean이라는 개념을 추가

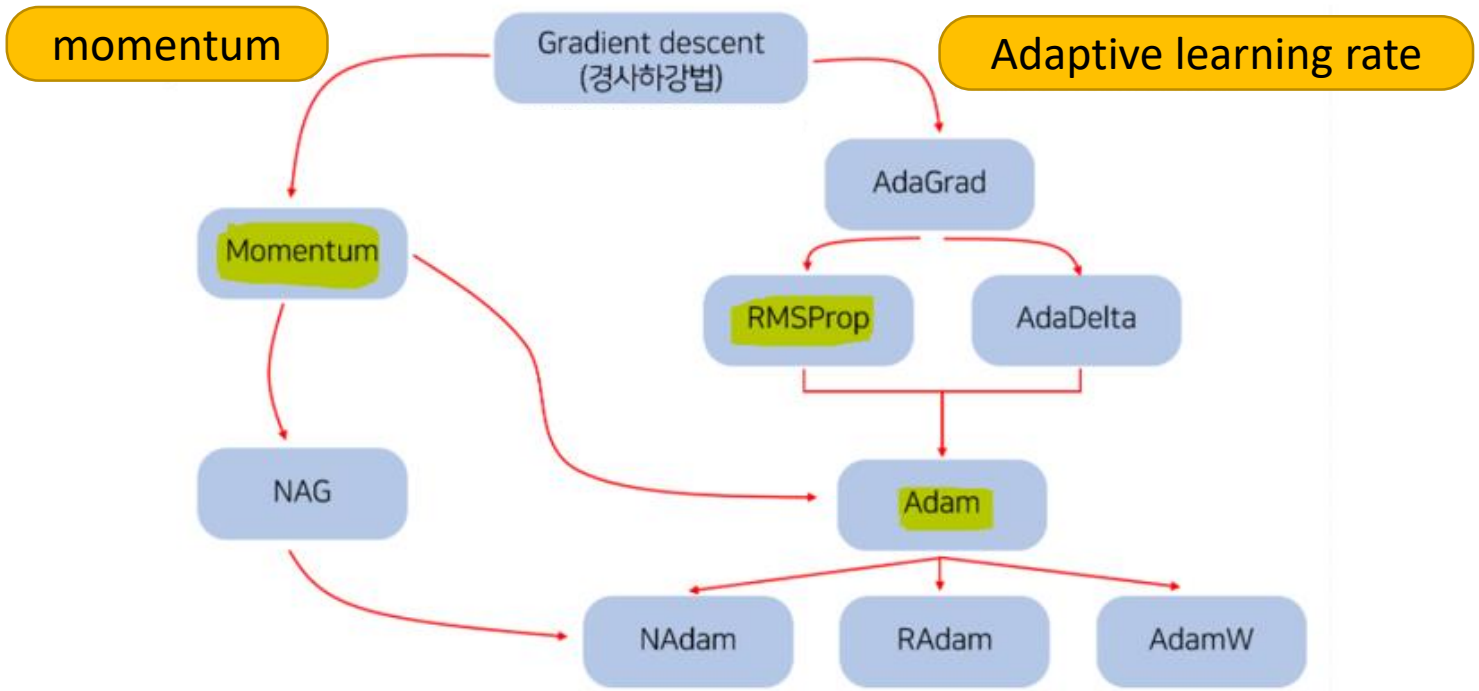
$$h \leftarrow \rho h + (1-\rho)\left(\frac{\partial E}{\partial w}\right)^2$$

$$w \leftarrow w - \eta \frac{1}{\sqrt{h}} \frac{\partial E}{\partial w}$$

h를 구하는 식에 ρ 가 존재함으로써 이전 시점의 h를 적당한 비율로 감소시킴.
이를 통해 아다그라드의 단점을 해결함.
Geoffrey Hinton은 ρ 값으로 0.9를 추천.

Adam(Adaptive Moment Estimation)

- 현재 주로 사용되는 고급 경사 하강법



딥러닝 구동에 사용되는 고급 경사 하강법 개요 및 활용법

고급 경사 하강법	개요	효과	케라스 사용법
확률적 경사 하강법 (SGD)	랜덤하게 추출한 일부 데이터를 사용해 더 빨리, 자주 업데이트를 하게 하는 것	속도 개선	keras.optimizers.SGD(lr = 0.1) 케라스 최적화 함수를 이용합니다.
모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True) 네스테로프 옵션을 추가합니다.
아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6) 아다그라드 함수를 사용합니다. ※ 참고: 여기서 epsilon, rho, decay 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 lr, 즉 learning rate(학습률) 값만 적절히 조절하면 됩니다.

딥러닝 구동에 사용되는 고급 경사 하강법 개요 및 활용법

고급 경사 하강법	개요	효과	케라스 사용법
알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0) 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0) 아담 함수를 사용합니다.

1. 퍼셉트론 (Perceptron)
2. XOR 해결을 위한 다층 퍼셉트론 (Multilayer Perceptron)
3. 인공 신경망 (Artificial Neural Network)
4. 역전파 (Back Propagation)
5. 기울기 소실 (vanishing gradient)
6. 활성화 함수 (Activation Function)
7. 고급 경사 하강법 (Advanced Gradient Descent)

<Optimizer의 종류>