퍼셉트론(perceptron)

입력 값을 놓고 활성화 함수에 의해 일정한 수준을 넘으면 참을, 그렇지 않으면 거짓을 내보내는 이 간단한 회로가 하는 일이 뉴런과 비슷하다.
-> 인공 신경망(Artificial Neural Network)의 연구

신경망의 기본 구조 - 여러 층의 퍼셉트론을 서로 연결시키고 복잡하게 조합하여 주어진 입력 값에 대한 판단을 하게 하는 것

신경망을 이루는 가장 중요한 기본 단위 - **퍼셉트론**
퍼셉트론은 입력 값과 활성화 함수를 사용해 출력 값을 다음으로 넘기는 가장 작은 신경망 단위

1. 가중치, 가중합, 바이어스, 활성화 함수

$y = ax + b$ (a는 기울기, b는 y 절편)
-> $y = wx + b$ (w는 가중치, b는 바이어스)

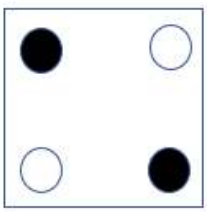
기울기 a -> **가중치(weight)**

y 절편 b -> **바이어스(bias)**

입력 값(x)과 가중치(w)의 곱을 모두 더한 다음 거기에 바이어스(b)를 더한 값 (예: $wx + b$)
-> **가중합(weghted sum)**

가중합의 결과를 놓고 1 또는 0을 판단하는 함수 -> **활성화 함수(activation function)**

2. 퍼셉트론의 과제



사각형 종이에 검은점 두 개와 흰점 두 개가 놓여 있다고 하자.
직선의 한쪽 편에는 검은점만 있고, 다른 한쪽에는 흰점만 있게끔 선을 그을 수 없음.
선형 회귀, 로지스틱 회귀, 퍼셉트론 -> 선이나 평면을 그리는 작업!
그런데 위 예시처럼 선을 그어도 해결되지 않는 상황이 있음.....

3. XOR 문제

컴퓨터는 두 가지의 디지털 값, 0과 1을 입력해 하나의 값을 출력하는 회로가 모여 만들어짐
-> 이 회로가 '**게이트(gate)**'

(AND 진리표)

x_1	x_2	결괏값
0	0	0
0	1	0
1	0	0
1	1	1

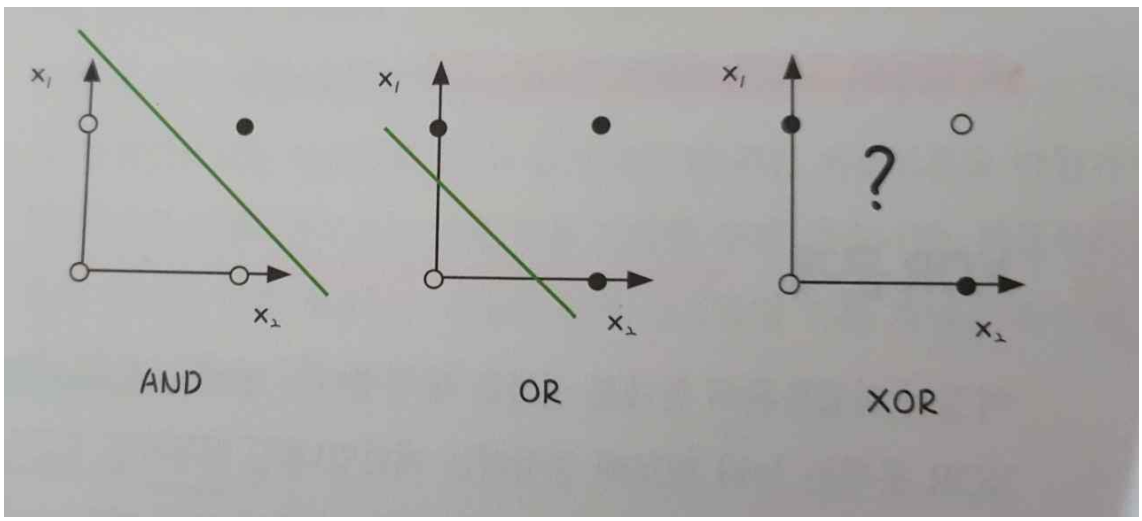
(OR 진리표)

x_1	x_2	결괏값
0	0	0
0	1	1
1	0	1
1	1	1

(XOR 진리표)

x_1	x_2	결괏값
0	0	0
0	1	1
1	0	1
1	1	0

각각 그래프를 좌표 평면에 나타내면 다음과 같음



AND, OR 게이트는 직선을 그어 결괏값이 1인 값(검은점)을 구별할 수 있지만 XOR의 경우

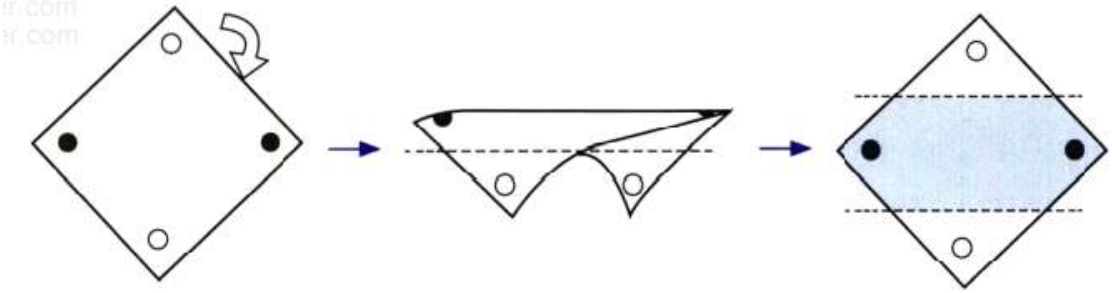
선을 그어 구분할 수 없음!!

-> **다층 퍼셉트론(multilayer perceptron)**을 사용하여 이를 해결!!

다층 퍼셉트론(multilayer perceptron)

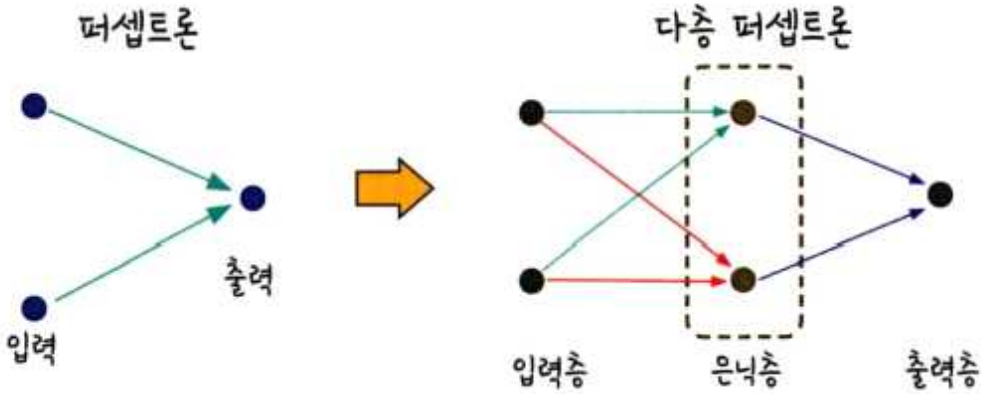
인공지능 학자들은 인공 신경망을 개발하기 위해서 반드시 XOR 문제를 극복해야만 했다.

좌표 평면 자체에 변화를 주면 XOR 문제 해결 가능!!

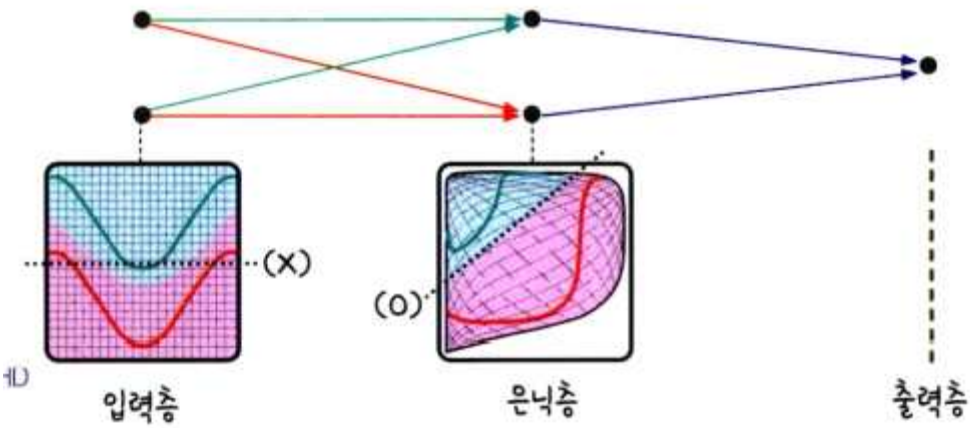


XOR 문제를 해결하기 위해서는 두 개의 퍼셉트론을 한 번에 계산할 수 있어야 함.

이를 가능하게 하려면 숨어있는 층, 즉 **은닉층(hidden layer)**가 필요!

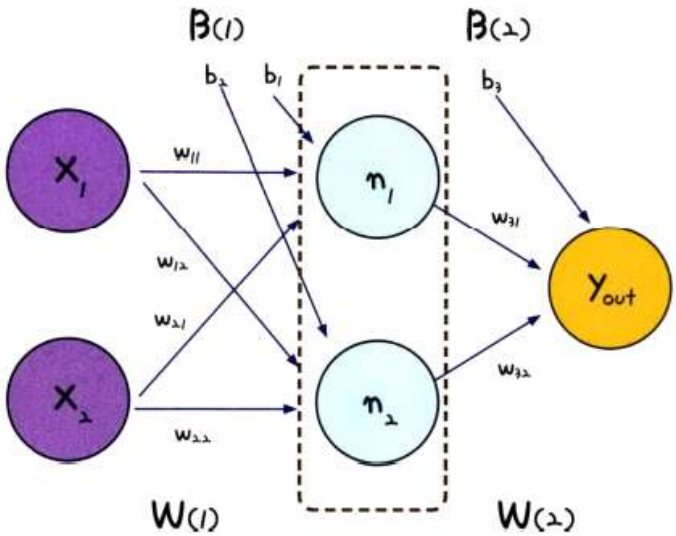


은닉층이 좌표 평면을 왜곡시키는 결과를 가져옴



왼쪽 그림을 보면 어떤 직선으로도 이를 해결할 수 없음. 하지만 은닉층을 만들어 공간을 왜곡하면 두 영역을 가로지르는 선이 직선으로 바뀜!

1. 다층 퍼셉트론의 설계



은닉층으로 퍼셉트론이 각각 자신의 가중치(w)와 바이어스(b) 값을 보내고, 이 은닉층에서 모인 값이 한 번 더 시그모이드 함수(기호로 σ라고 표시함)를 이용해 최종 값으로 결과를 보낸다.

은닉층에 모이는 중간 정거장을 노드(node)라고 하며 여기서는 n_1, n_2로 표현하였다.

$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$

$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$

위 두 식의 결괏값이 출력층으로 보내지고 출력층에서는 역시 시그모이드 함수를 통해 y 값이 정해진다. 이 값을 y_{out}이라 하면 다음과 같다.

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

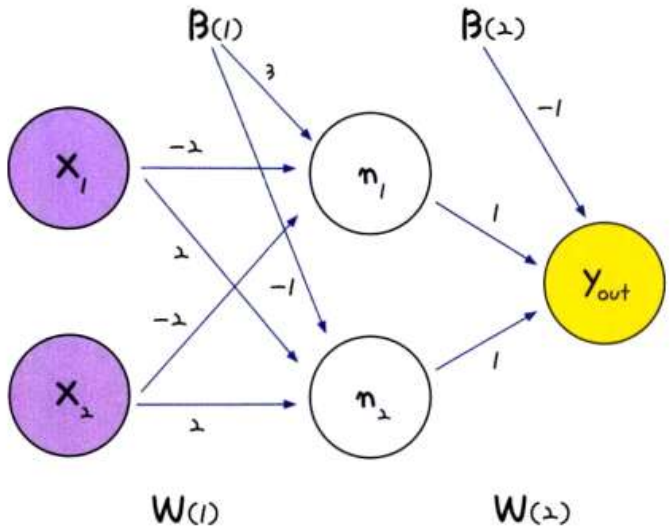
각각의 가중치와 바이어스는 2차원 배열로 다음과 같이 나타낼 수 있음

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

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

2. XOR 문제의 해결

만족하는 가중치와 바이어스의 조합은 무수히 많음



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

3. 코딩으로 XOR 문제 해결하기

(실습코드는 github 참고)

은닉층을 여러 개 쌓아올려 복잡한 문제를 해결하는 과정 - 뉴런이 복잡한 과정을 거쳐 사고를 낳는 사람의 신경망을 닮았음 -> 인공 신경망

if __name__ == "__main__" 명령어에 대해서는 다음 참고
https://madplay.github.io/post/python-main-function

오차 역전파

오차 역전파는 4장에서 배운 경사 하강법의 확장 개념
가중치와 바이어스를 실제 프로젝트에서는 어떻게 구할 수 있을까?

가중치를 구하는 방법 - 경사 하강법을 그대로 적용, 임의의 가중치를 선언하고 결괏값을 이용해 오차를 구한 뒤 이 오차가 최소인 지점으로 계속해서 조금씩 이동시킨다. 이 오차가 최

소가 되는 점(미분했을 때 기울기가 0이 되는 지점)을 찾으면 그것이 바로 알고자 하는 답이다!!

기존의 '단일 퍼셉트론'과 다르게 은닉층이 있는 **'다층 퍼셉트론'**도 원리는 크게 다르지 않음. 다층 퍼셉트론 역시 결괏값의 오차를 구해 이를 토대로 하나 앞선 가중치를 차례로 거슬러 올라가며 조정해 간다.

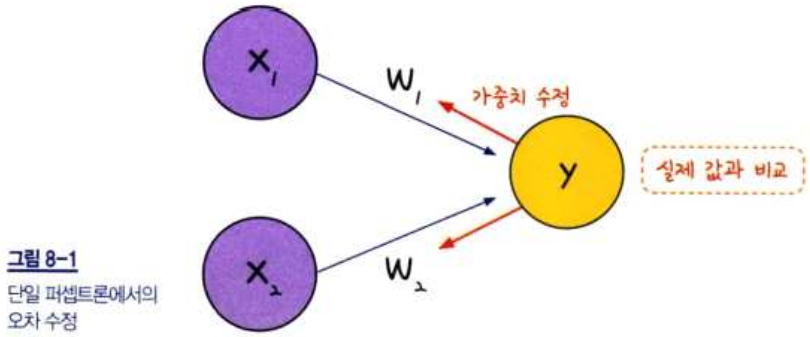


그림 8-1
단일 퍼셉트론에서의 오차 수정

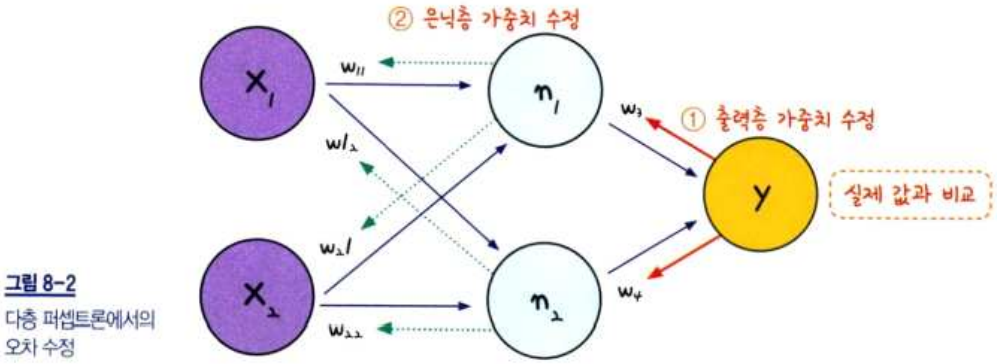


그림 8-2
다층 퍼셉트론에서의 오차 수정

최적화 계산 방향 - 출력층에서 시작해 앞으로 진행됨 -> 다층 퍼셉트론에서의 최적화 과정을 **오차 역전파(back propagation)**라고 부른다.

1. 임의의 초기 가중치(W)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 방향(미분 값이 0에 가까워지는 방향)으로 업데이트한다.
4. 위 과정을 더 이상 오차가 줄어들지 않을 때까지 반복한다.

기울기가 0이 되는 방향 -> 가중치에서 기울기를 뺐을 때 가중치의 변화가 전혀 없는 상태
오차 역전파 - 가중치에서 기울기를 빼도 값의 변화가 없을 때까지 계속해서 가중치 수정 작업을 반복하는 것!

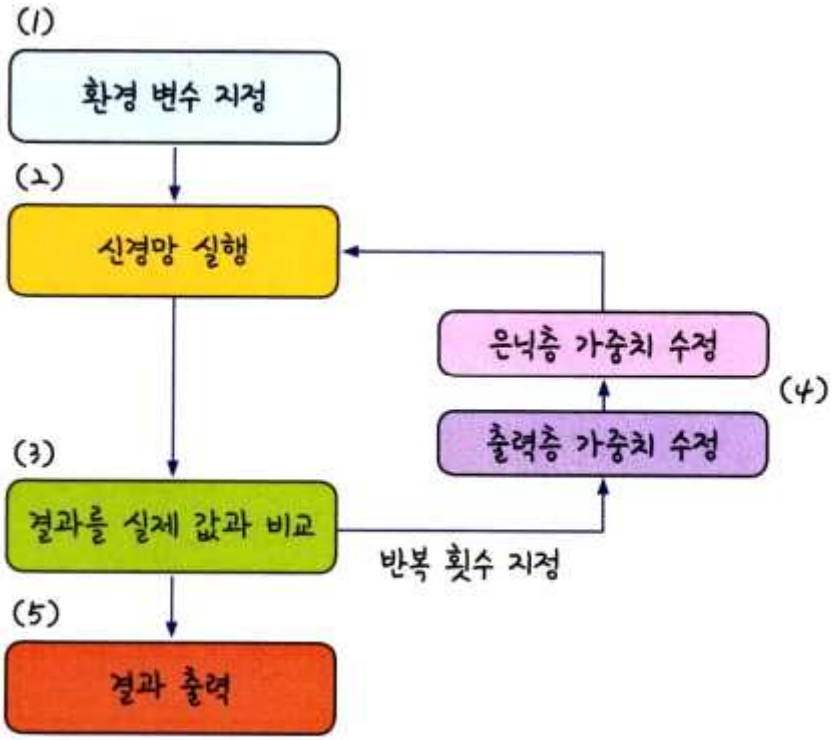
새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값!

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

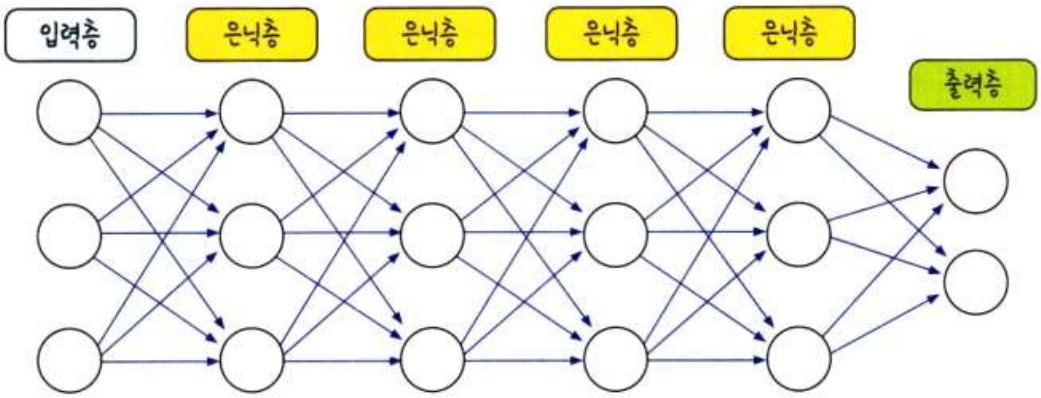
2. 코딩으로 확인하는 오차 역전파

오차 역전파 - 가중치를 몰라도 문제를 해결하기 위해 개발된 방법

입력된 실제 값과 다층 퍼셉트론의 계산 결과를 비교하여 가중치를 역전파 방식으로 수정해 가는 코딩은 다음과 같은 순서로 구현!

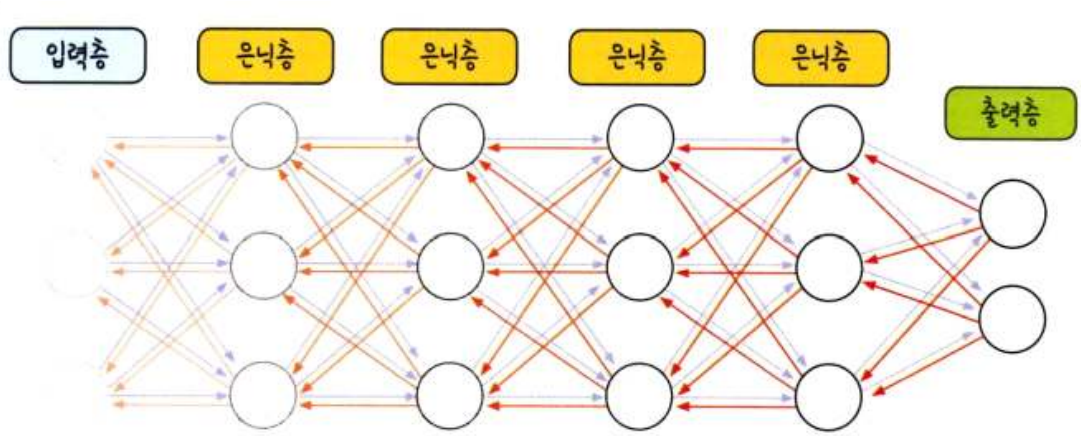


신경망에서 딥러닝으로

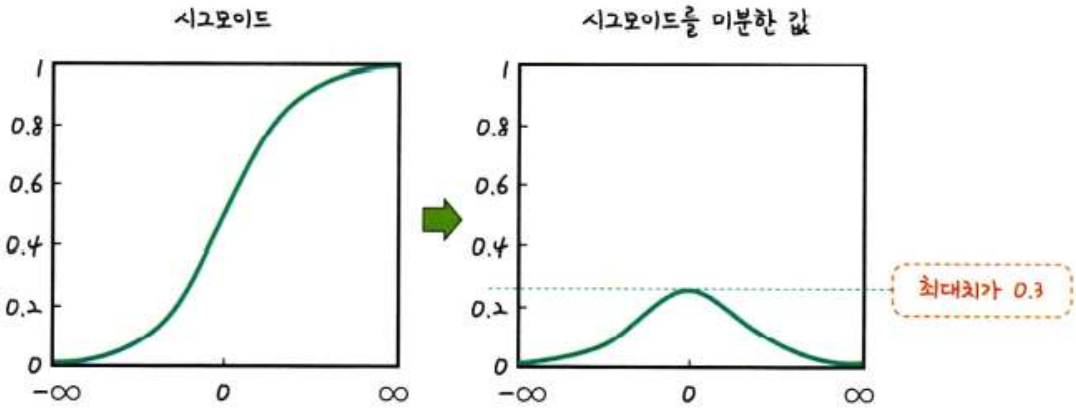


1. 기울기 소실 문제와 활성화 함수

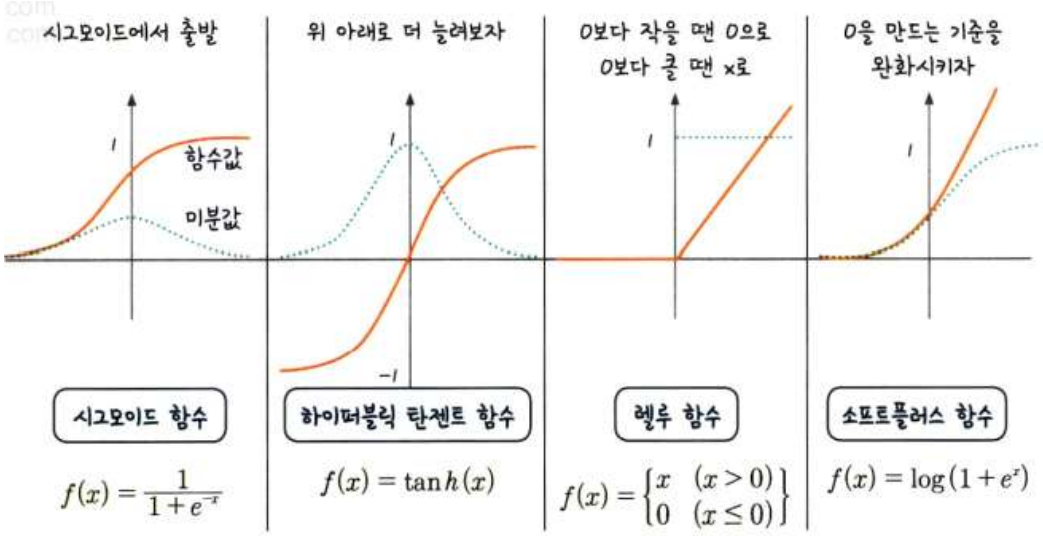
오차 역전파 - 출력층으로부터 하나씩 앞으로 되돌아가면 각 층의 가중치르 수정하는 방법
가중치 수정에는 기울기(미분 값)이 필요한데 층이 늘어나면서 역전파를 통해 전달되는 기울기의 값이 점점 작아져 맨 처음 층까지 전달되지 않는 기울기 소실(vanishing gradient) 문제가 발생함,



활성화 함수로 사용된 시그모이드 함수의 특성 때문에 기울기 소실 문제가 발생함. 시그모이드 함수를 미분하면 최대치가 0.3 이다. 1보다 작으므로 계속 곱하다 보면 0에 가까워짐.
-> 여러 층을 거칠수록 기울기가 사라져 가중치 수정이 어려워짐



-> 활성화 함수를 다른 여러 함수로 대체하기 시작



하이퍼볼릭 탄젠트 함수 - 시그모이드 함수의 범위를 -1에서 1로 확장한 개념, 여전히 1보다 작은 값이 존재하므로 기울기 소실 문제 사라지지 않음

렐루 함수 - 현재 가장 많이 사용되는 활성화 함수, 0보다 크면 x를 그대로 사용하고 그 이외의 값은 0으로 처리. 다만, x가 0보다 크기만 하면 미분값이 1이 되므로 여러 은닉층을 거쳐 곱해도 맨 처음 층까지 사라지지 않고 남아있을 수 있음.

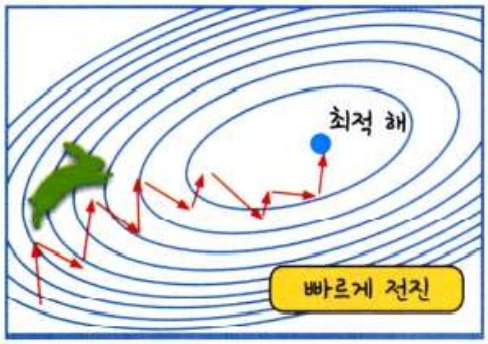
소프트플러스 함수 - 렐루 함수의 0이 되는 순간을 완화시킨 버전

2. 속도와 정확도 문제를 해결하는 고급 경사 하강법

경사 하강법은 정확하게 가중치를 찾아가지만, 한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있음. -> **고급 경사 하강법**

* **확률적 경사 하강법(Stochastic Gradient Descent, SGD)** - 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용하는 방법 (더 빨리 그리고 자주 업데이트를 하는 것이 가능함)

경사 하강법



확률적 경사 하강법

확률적 경사 하강법은 랜덤한 일부 데이터를 사용하는 만큼 중간 결과의 진폭이 크고 불안정해 보일 수 있으나 속도가 확연히 빠르면서도 최적 해에 근사한 값을 찾아낸다는 장점이 있음.

* **모멘텀(momentum)** - 모멘텀 SGD란 경사 하강법에 탄력을 더해 주는 것. 경사 하강법처럼 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법 (지그재그 줄어듬, 관성 효과 발생)