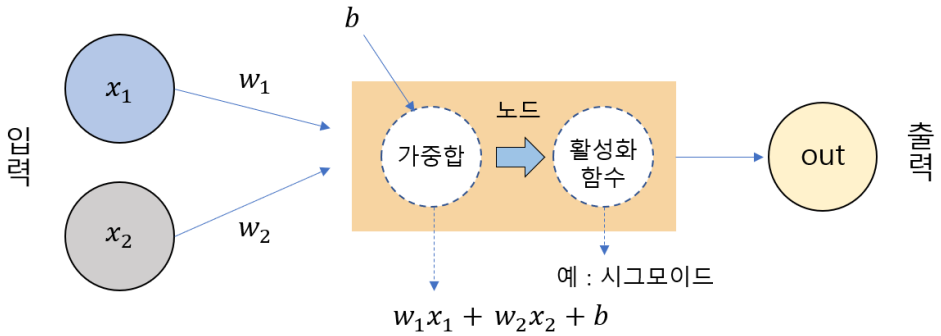
<6장. 퍼셉트론>



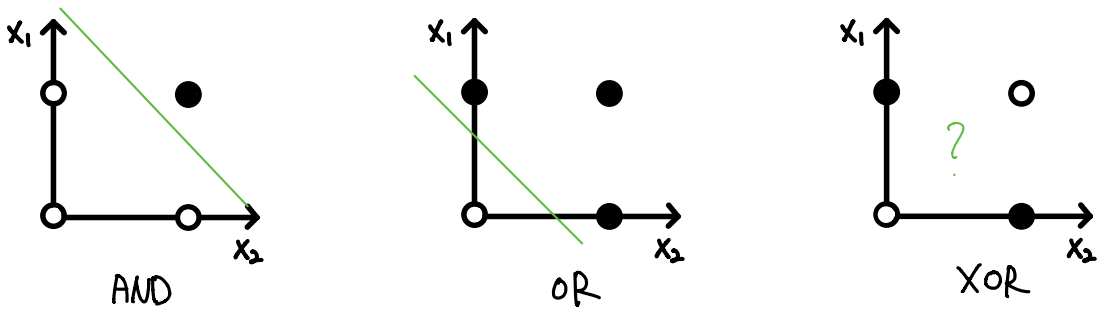
1. 가중치, 가중합, 바이어스, 활성화 함수

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

가중합 : 입력값(x)과 가중치(w)의 곱을 모두 더한 다음 바이어스(b)를 더한 값
가중합의 결과를 놓고 1 또는 0을 출력해서 다음으로 보냄

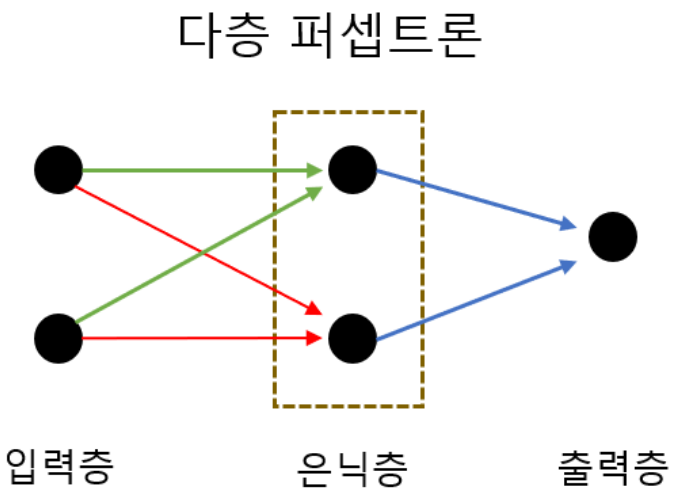
활성화 함수 : 0과 1을 판단하는 함수 ex) 시그모이드 함수

3. XOR 문제



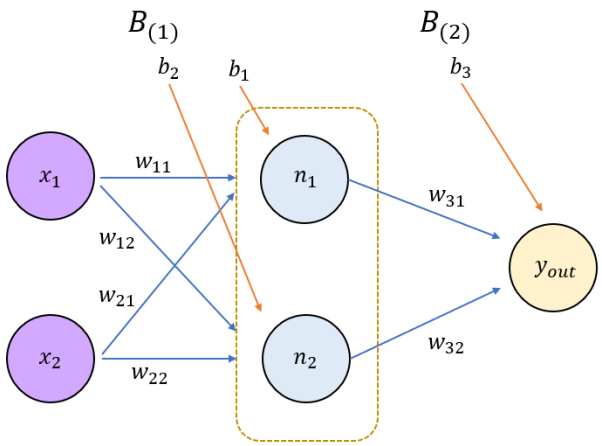
XOR문제를 어떻게 해결할 것인가? => 다층 퍼셉트론

<7장. 다층 퍼셉트론>



두 개의 퍼셉트론을 한 번에 계산할 수 있도록 숨어있는 층
즉, 은닉층을 만들어야함

1. 다층 퍼셉트론의 설계



1. 은닉층으로 퍼셉트론이 각각 자신의 가중치와 바이어스값을 보낸다.
2. 은닉층에서 모인 값이 활성화 함수를 이용해 최종 값으로 결과를 보낸다.

- 노드(n1, n2) : 은닉층에 모이는 중간 정거장
  - n1과 n2의 값은 각각 단일 퍼셉트론의 값과 같음
- 활성화 함수 => 시그모이드 함수(σ)

$$n_1 = \sigma(x_1 w_{11} + x_2 w_{21} + b_1)$$
$$n_2 = \sigma(x_1 w_{12} + x_2 w_{22} + b_2)$$
$$y_{out} = \sigma(n_1 w_{31} + n_2 w_{32} + b_3)$$

두 식의 결과값이 출력층으로 보내짐

출력값

- 가중치와 바이어스의 값 정하기

은닉층을 포함해 가중치 6개와 바이어스 3개가 필요

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

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

2. XOR 문제 해결

$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \qquad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$

$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \qquad B(2) = [-1]$$

공식 대입

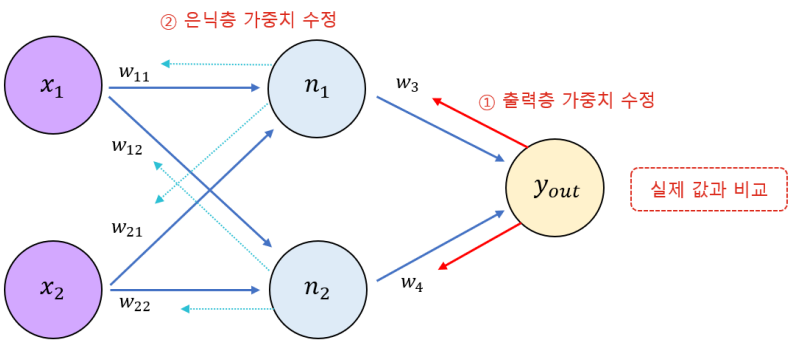
x_1	x_2	n_1	n_1	y_{out}	원하는 값
0	0	$\sigma(0\times(-2)+0\times(-2)+3)\approx 1$	$\sigma(0\times 2+0\times 2-1)\approx 0$	$\sigma(1\times 1+0\times 1-1)\approx 0$	0
0	1	$\sigma(0\times(-2)+1\times(-2)+3)\approx 1$	$\sigma(0\times 2+1\times 2-1)\approx 1$	$\sigma(1\times 1+1\times 1-1)\approx 1$	1
1	0	$\sigma(1\times(-2)+0\times(-2)+3)\approx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

<8장. 오차 역전파>

1. 오차 역전파의 개념

- XOR 문제는 가중치와 바이어스를 미리 알아본 후 대입

=> 실제 프로젝트에서는 가중치와 바이어스를 어떻게 알 수 있을까?



- 결과값의 오차를 구해 이를 토대로 하나 앞선 가중치를 차례로 거슬러 올라가며 조정

=> 최적화의 계산 방향이 출력층에서 앞으로 진행

=> 다층 퍼셉트론에서의 최적화 과정을 오차 역전파 (back propagation)이라고 부름

1) 임의의 초기 가중치(W)를 준 뒤 결과를 계산
2) 계산 결과와 우리가 원하는 값 사이의 오차를 구함
3) 경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 업데이트
4) 위 과정을 더이상 오차가 줄어들지 않을 때까지 반복

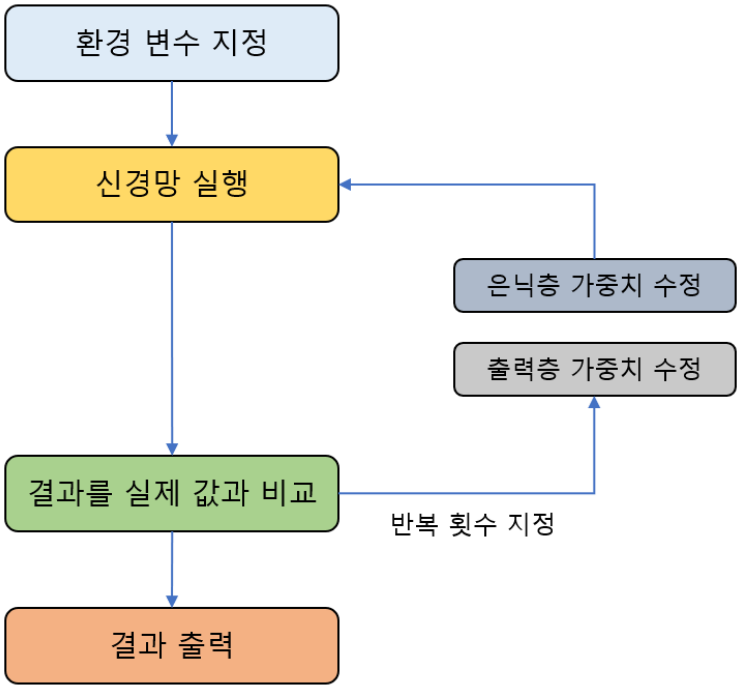
새 가중치는 ↓ 현 가중치에서 ↓ '가중치에 대한 기울기 ' 를 뺀 값!

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

⟹ 오차가 작아지는 방향으로 업데이트
= 기울기가 0이 되는 방향으로 나아감
= 미분 값이 0에 가까워 지는 방향으로 업데이트

2. 코딩으로 확인하는 오차 역전파

- 신경망 구현 과정



1) 환경 변수 지정
: 데이터셋(입력 값, 타깃 결과값), 학습률, 활성화 함수, 가중치 등을 지정

2) 신경망 실행
: 초기값을 입력하여 활서오하 함수와 가중치를 거쳐 결과값이 나오게 함

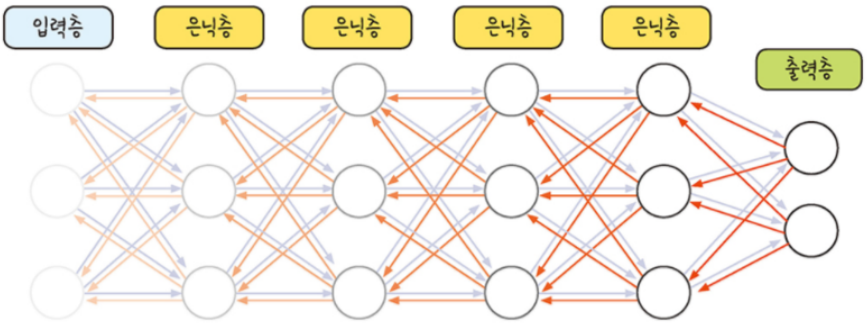
3) 결과를 실제 값과 비교
: 오차를 측정

4) 역전파 실행
: 출력층과 은닉층의 가중치 수정

5) 결과 출력

<9장. 신경망에서 딥러닝으로>

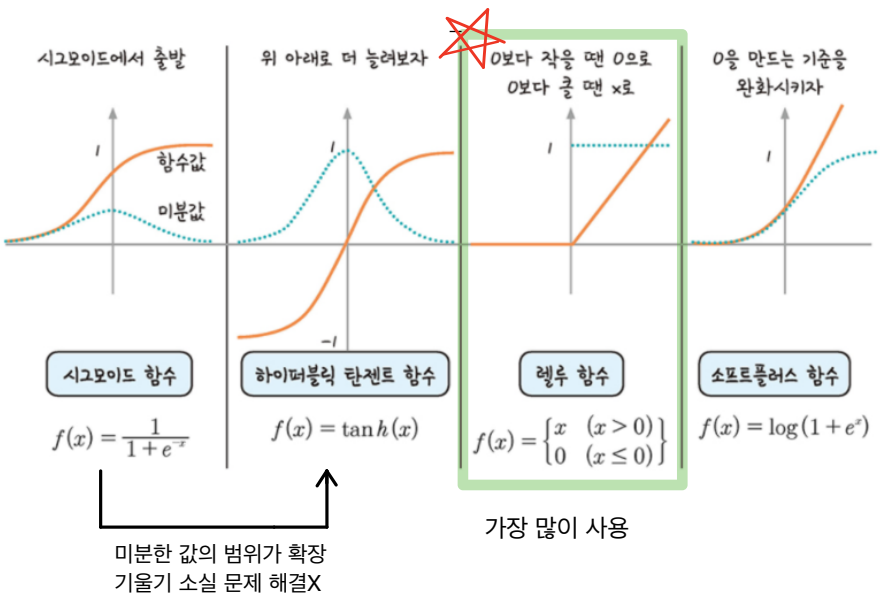
1. 기울기 소실 문제와 활성화 함수



- 층이 늘어나면서 역전파를 통해 전달되는 이 기울기의 값이 점점 작아져 맨 처음 층까지 전달되지 않는 기울기 소실(vanishing gradient) 문제가 발생

=> 이는 활성화 함수로 사용된 시그모이드 함수가 미분하면 최대치가 0.3인 특성 때문이다.
미분의 최대치가 1보다 작으므로 계속 곱하다 보면 0에 가까워 진다.
따라서 여러 층을 거칠 수록 기울기가 사라져 가중치를 수정하기 어려워지는 것이다.

=> 이를 해결하고자 활성화 함수를 다른 함수로 대체하기 시작



2. 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 확률적 경사 하강법

: 랜덤하게 추출한 일부 데이터를 사용 -> 속도 개선

- 모멘텀

: 오차를 수정하기 전 바로 앞 수전 값과 방향을 참고하여 같은 방향으로 일정한 비율만 수정
-> 정확도 개선

- 네스테로프 모멘텀

: 모멘텀이 이동시킬 방향으로 미리 이동해 그레디언트를 계산. 불필요한 이동을 줄이는 효과
-> 정확도 개선

- 아다그라드

: 변수의 업데이트가 잦으면 합습률을 적게 하여 이동 보폭을 조절하는 방법
-> 보폭 크기 개선

- 알엠에스프롭

: 아다그라드의 보폭 민감도를 보완한 방법 -> 보폭 크기 개선

- 아담

: 모멘텀과 알엠에스프롭 방법을 합친 방법 -> 정확도와 보폭 크기 개선