

ch6.

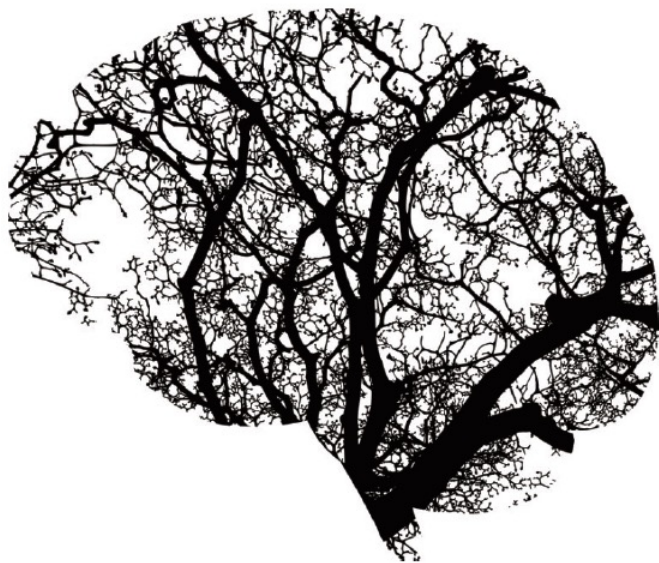
신경망의 이해

6장 목차

- 1 | 가중치, 가중합, 바이어스, 활성화 함수
- 2 | 퍼셉트론의 과제
- 3 | XOR 문제

퍼셉트론(1958)

- 인간의 뇌는 약 1000억개의 뉴런으로 이루어져 있고, 뉴런과 뉴런 사이에는 시냅스라는 연결 부위 존재

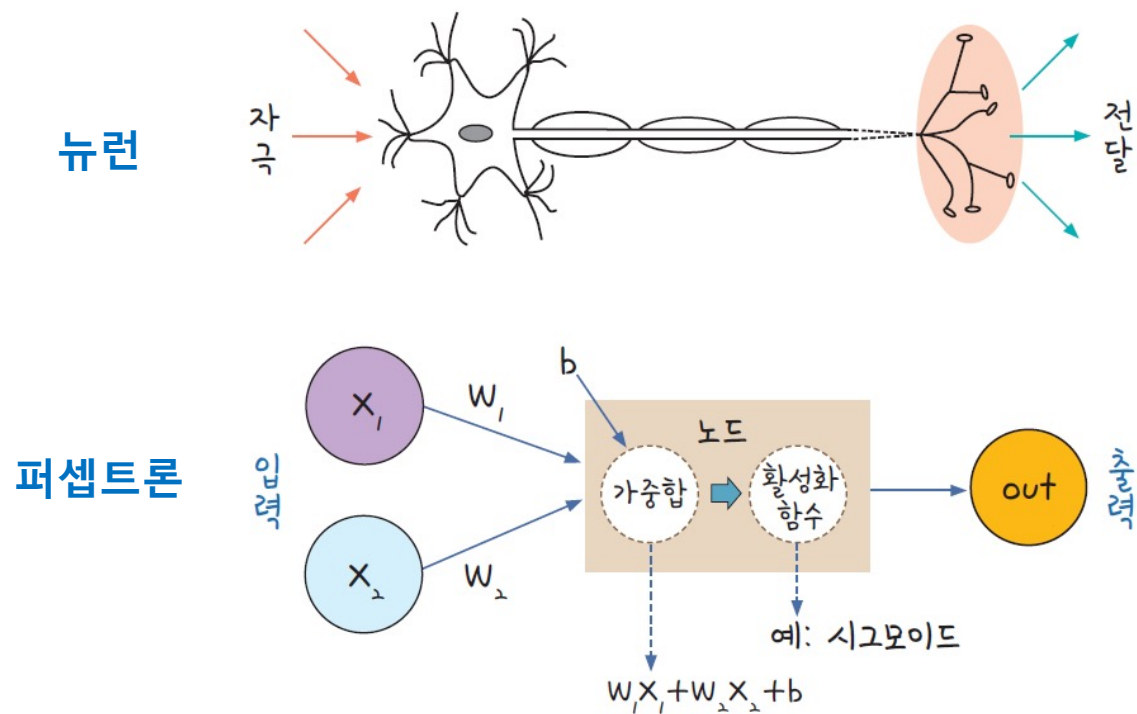


- 신경 말단에서 자극을 받으면 시냅스에서 화학 물질이 나와 임계값 넘으면 전위 변화를 일으키는데, 이 매커니즘이 로지스틱 회귀와 많이 닮음
(입력 값을 놓고 활성화 함수에 의해 일정 수준이 넘으면 참, 그렇지 않으면 거짓을 내보내는 회로)
-> **인공신경망** : 뉴런과 뉴런의 연결처럼 퍼셉트론의 연결을 통해 입력 값에 대한 판단을 하게 하는 것

퍼셉트론(1958)

- 여러 층의 퍼셉트론을 서로 연결시키고 복잡하게 조합하여 주어진 입력 값에 대한 판단을 하게 하는 것, 그것이 바로 신경망의 기본 구조임
- 퍼셉트론은 입력 값과 활성화 함수를 사용해 출력 값을 다음으로 넘기는 가장 작은 **신경망 기본 단위**

그림 6-1 뉴런과 퍼셉트론의 비교



가중치, 가중합, 바이어스, 활성화 함수

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

$$y = ax + b \text{ (} a \text{는 기울기, } b \text{는 } y \text{ 절편)}$$

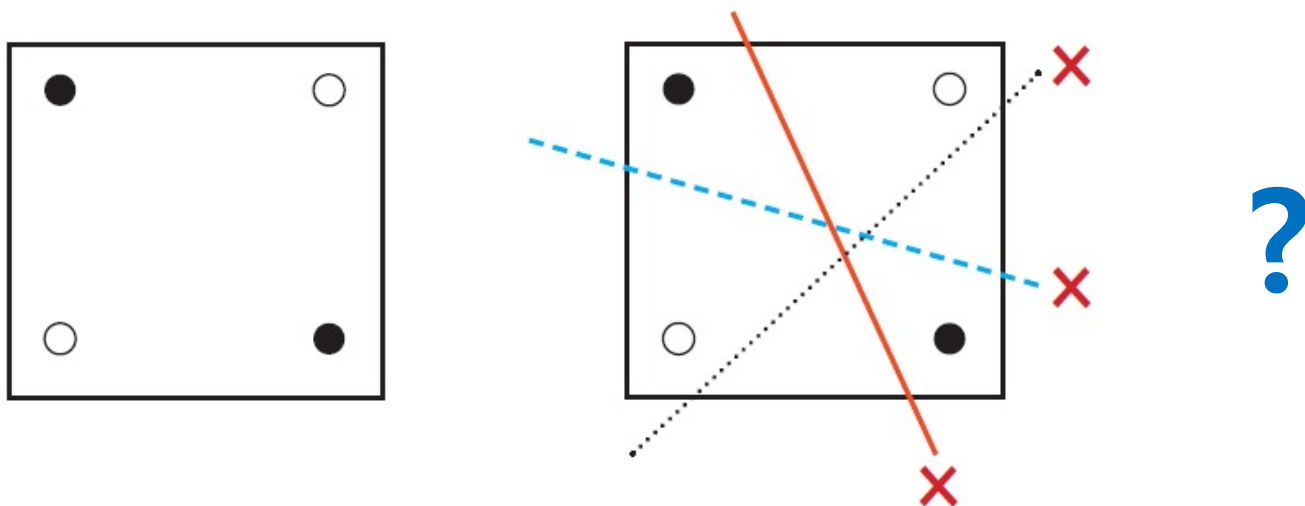
$$\rightarrow y = wx + b \text{ (} w \text{는 가중치, } b \text{는 바이어스)}$$

- 기울기 a 는 퍼셉트론에서는 가중치를 의미하는 $w(\text{weight})$ 로 표기됨
- y 절편 b 는 똑같이 $b(\text{bias})$ 라고 씀
- **가중 합(weighted sum)** : 입력 값 x 과 가중치 w 의 곱을 모두 더한 다음 거기에 바이어스 b 를 더한 값
- **활성화 함수(activation function)** : 가중합의 결과를 놓고 (시그모이드)1 또는 0을 출력해서 다음으로 보내는데, 여기서 0과 1을 판단하는 함수(sigmoid함수, relu함수, softmax함수 등)

XOR문제(1969) : 퍼셉트론의 과제

- 단 하나의 퍼셉트론으로는 많은 것을 기대할 수가 없음
- 지금부터는 퍼셉트론의 한계와 이를 해결하는 과정을 보며 신경망의 기본 개념을 확립해 보자
- 이 네 점 사이에 직선을 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
- 퍼셉트론 역시 선을 긋는 작업이라고 할 수 있음

이 예시처럼 경우에 따라서는 **선을 아무리 그어도 해결되지 않는 상황**이 있음

XOR문제(1969) : 퍼셉트론의 과제

- **XOR 문제** : 논리 회로에 등장하는 개념
- 컴퓨터는 두 가지 디지털 값(0과 1)을 입력해 하나의 값을 출력하는 회로가 모여 만들어지는데, 이 회로를 **게이트(gate)**라고 한다.
 - AND게이트 : x_1, x_2 둘 다 1일 때만 결과값이 1로 출력되는 게이트
 - OR게이트 : x_1, x_2 둘 중 하나라도 1이면 결과값이 1로 출력되는 게이트
 - XOR게이트 : 둘 중 하나만 1일 때만 결과값이 1로 출력되는 게이트

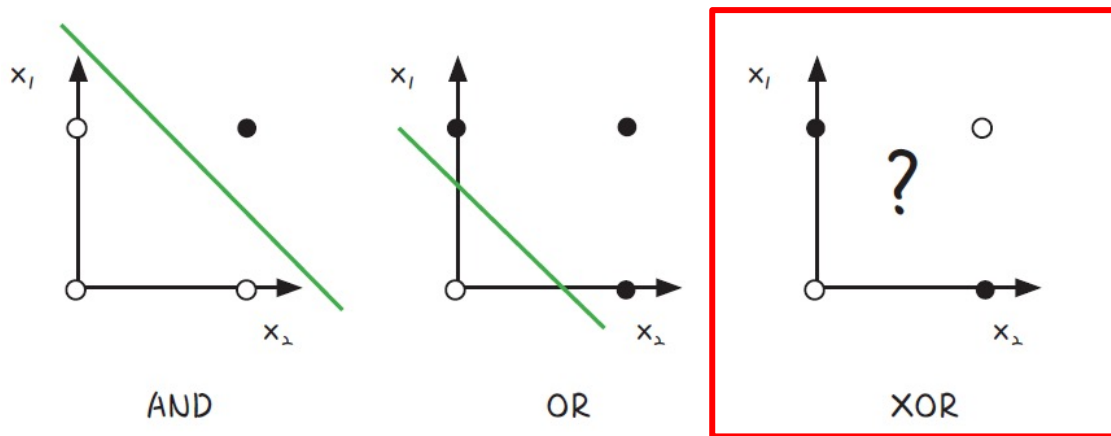
표 6-1 AND, OR, XOR 게이트에 대한 진리표

AND 진리표			OR 진리표			XOR 진리표		
x_1	x_2	결과값	x_1	x_2	결과값	x_1	x_2	결과값
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

XOR문제(1969) : 퍼셉트론의 과제

- 결괏값이 0이면 흰점으로, 1이면 검은점으로 나타낸 후 직선을 그어 위 조건을 만족할 수 있는지 보자

그림 6-4 AND, OR, XOR 진리표대로 좌표 평면에 표현한 뒤 선을 그어 색이 같은 점끼리 나누기(XOR은 불가능)



- AND와 OR 게이트는 직선을 그어 결괏값이 1인 값(검은점)을 구별할 수 있음
- XOR의 경우 선을 그어 구분할 수 없음
- MIT의 마빈 민스키(Marvin Minsky) 교수가 1969년에 발표한 <퍼셉트론즈(Perceptrons)>(1969)
: '뉴런 → 신경망 → 지능' 컨셉으로 '퍼셉트론 → 인공 신경망 → 인공지능' 구현이 생각처럼 쉽지 않다!
간단하다고 생각했던 XOR문제도 해결할 수 없었던 것 -> 이 후 10년간 인공지능의 침체기
-> 10년 후 XOR문제를 해결한 것이 다층 퍼셉트론

ch7.

다층 퍼셉트론
(1980)

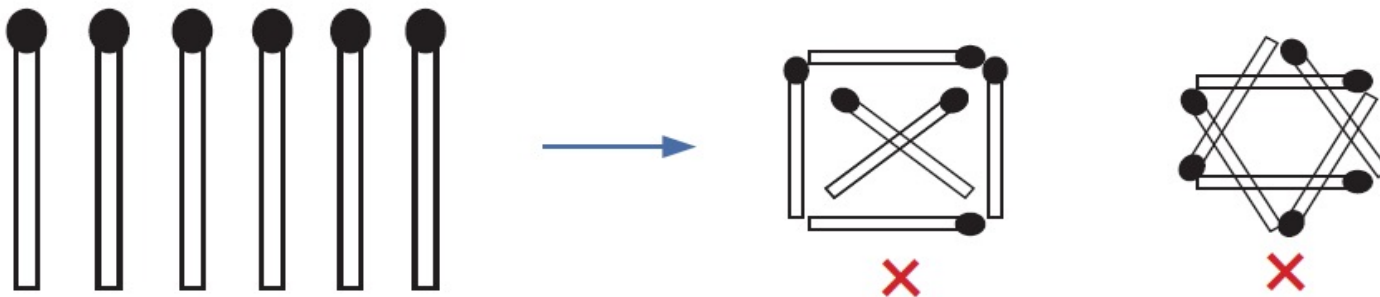
7장 다층 퍼셉트론

- 1 | 다층 퍼셉트론의 설계
- 2 | XOR 문제의 해결
- 3 | 코딩으로 XOR 문제 해결하기

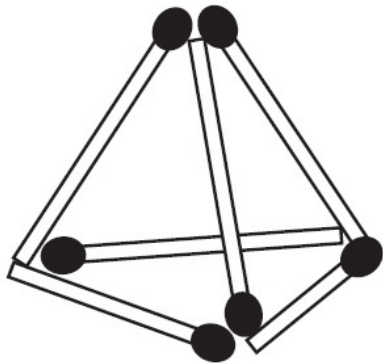
다층 퍼셉트론

- 단일 퍼셉트론으로는 XOR문제를 해결할 수 없다.
- 언뜻 보기에 해답이 없어 보이는 이 문제를 해결하려면 새로운 접근이 필요함

Q. 성냥개비 여섯 개로 정삼각형 네 개를 만들 수 있는가?



- **차원의 변화** : 이 문제는 2차원 평면에서만 해결하려는 고정관념을 깨고 3차원 피라미드 모양으로 성냥개비를 쌓아 올리니 해결됨



다층 퍼셉트론

- 인공지능 학자들은 인공 신경망을 개발하기 위해서 반드시 XOR 문제를 극복해야만 했음
- 이 문제 역시 고정관념을 깬 기발한 아이디어에서 해결점이 보였음

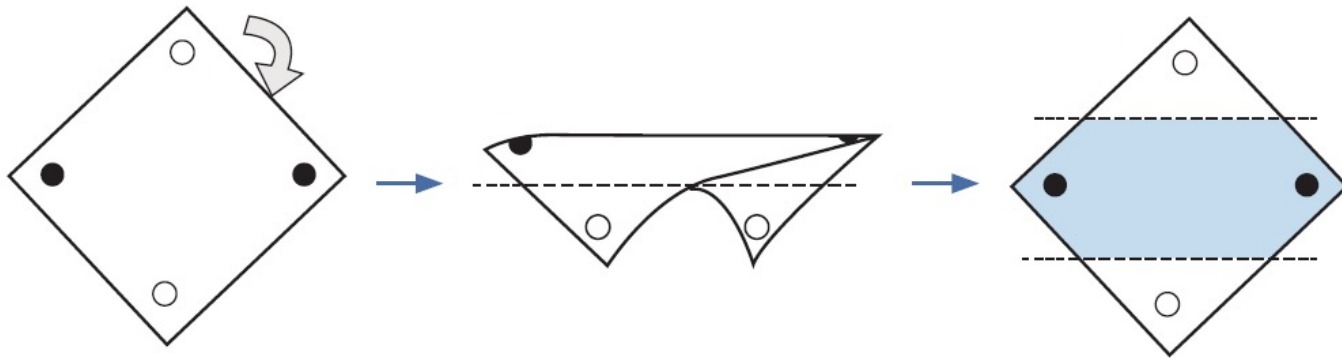
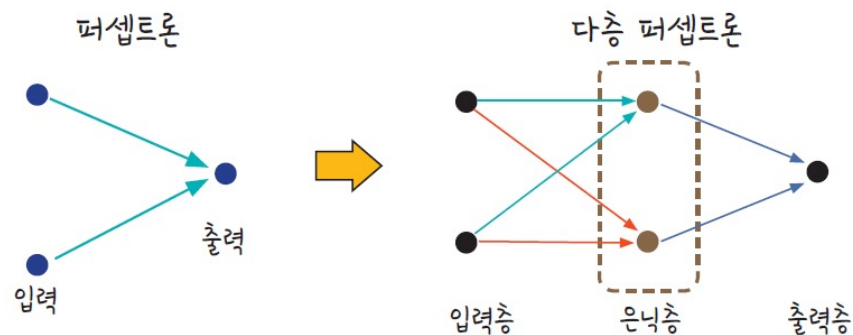


그림 7-3 XOR 문제의 해결은 평면을 휘어주는 것!

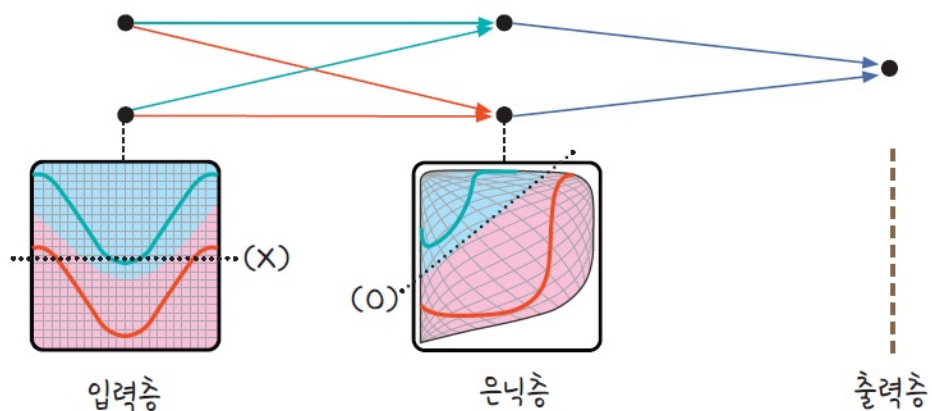
다층 퍼셉트론

좌표 평면 자체에 차원 변화를 주는 것으로 XOR 문제를 해결

- 은닉층(hidden layer)을 만들면 우리는 두 개의 퍼셉트론을 한 번에 계산할 수 있게 됨



- 아래 그림에서 파란색과 빨간색 영역을 구분하고자 할 때, 입력층만으로는 어떤 직선도 불가능하지만, 은닉층을 만들어 공간을 왜곡하면 두 영역을 구분하는 직선을 구할 수 있다!



다층 퍼셉트론

좌표 평면 자체에 차원 변화를 주는 것으로 XOR 문제를 해결

- 다층 퍼셉트론이 입력층과 출력층 사이에 숨어있는 은닉층을 만드는 것을 도식으로 나타내면

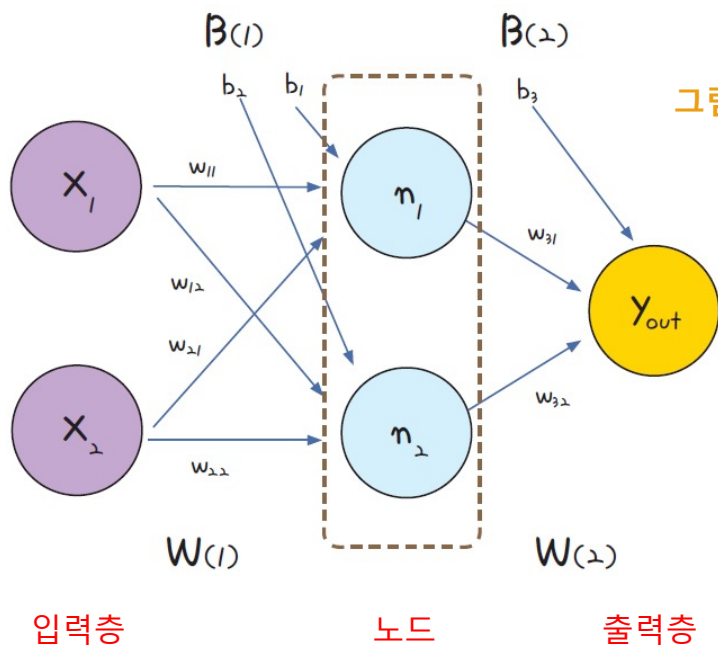


그림 7-6 다층 퍼셉트론의 내부

- 가운데 숨어있는 은닉층으로 퍼셉트론이 각각 자신의 **가중치 w** 와 **바이어스 b** 값을 보냄
- 이 은닉층에서 모인 값이 **시그모이드 함수(σ)**를 이용해 최종 값으로 결과를 보냄
- 노드(node)** : 은닉층에 모이는 중간 정거장. 여기서는 n_1 과 n_2 로 표현함

다층 퍼셉트론의 설계

- n_1 과 n_2 의 값은 각각 단일 퍼셉트론의 값과 같음

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

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

- n_1, n_2 결과값(입력값)이 출력층으로 보내짐
- 출력층에서는 역시 시그모이드 함수를 통해 y 값이 정해짐
- 이 값을 y_{out} 이라 할 때 식으로 표현하면 다음과 같음

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

- 이제 각각의 **가중치 w** 와 **바이어스 b** 의 값을 정할 차례임
- 2차원 배열로 늘어놓으면 다음과 같이 표시할 수 있음. **은닉층을 포함해 가중치 6개와 바이어스 3개**

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

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

XOR 문제의 해결

- 각 변숫값을 정하고 이를 이용해 XOR 문제를 해결하는 과정을 알아보자

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

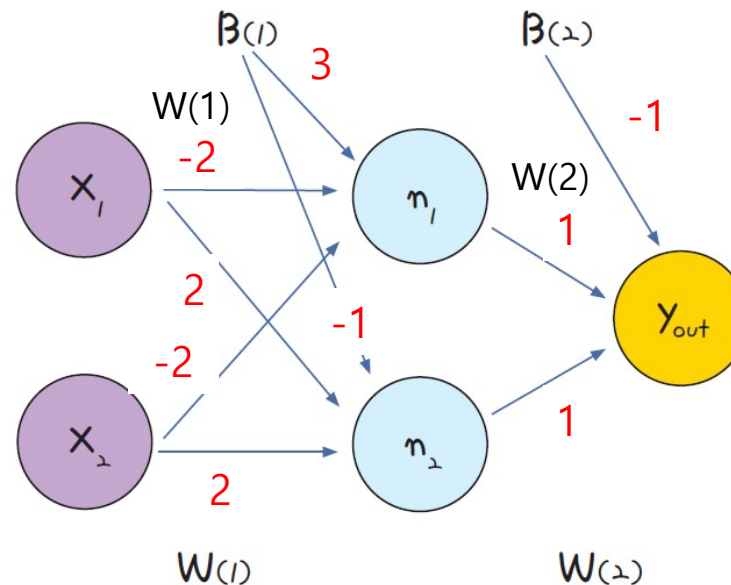
$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$

- 이것을 도식에 대입하면 다음과 같음

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

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

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



- 이제 x_1 의 값과 x_2 의 값을 각각 입력해 y_{out} 값이 우리가 원하는 값으로 나오는지를 점검해 보자

x_1	x_2	NAND 게이트 n_1	OR 게이트 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

-> 숨어있는 2개의 노드를 둔 다층 퍼셉트론을 구성해 결국 XOR 문제를 해결하였다!

코딩으로 XOR 문제 해결하기

- 위 표에서 입력 값 x_1, x_2 가 모두 0 또는 1일 때 0을 출력하고, 하나라도 0이 아니면 1을 출력한다.
 - > **NAND(Negative And) 게이트** : AND 게이트의 정반대 값을 출력한다.
 - AND게이트 : x_1, x_2 둘 다 1일 때만 결과값이 1로 출력되는 게이트
 - OR게이트 : x_1, x_2 둘 중 하나라도 1이면 결과값이 1로 출력되는 게이트
 - XOR게이트 : 둘 중 하나만 1일 때만 결과값이 1로 출력되는 게이트
- n_2 의 값을 잘 보면 x_1, x_2 에 대한 **OR 게이트**에 대한 답이다.
- **NAND 게이트**와 **OR 게이트**, 이 두 가지를 내재한 각각의 퍼셉트론이 다중 레이어 안에서 각각 작동하고, 이 두 가지 값에 대해 **AND 게이트**를 수행한 값이 바로 우리가 구하고자 하는 Y_{out} 임을 알 수 있다.

코딩으로 XOR 문제 해결하기

예제 소스 : deeplearning_class/06_XOR.ipynb

```
import numpy as np
```

```
# 가중치와 바이어스
```

```
w11 = np.array([-2, -2])  
w12 = np.array([2, 2])  
w2 = np.array([1, 1])  
b1 = 3  
b2 = -1  
b3 = -1
```

w, b 값들 정의
- 가중치 w11, w12, w2
- 바이어스 b1, b2, b3

```
# 퍼셉트론
```

```
def MLP(x, w, b):  
    y = np.sum(w * x) + b  
    if y <= 0:  
        return 0  
    else:  
        return 1
```

퍼셉트론 함수 정의
: 0 또는 1을 출력

```
# NAND 게이트
```

```
def NAND(x1, x2):  
    return MLP(np.array([x1, x2]), w11, b1)
```

```
# OR 게이트
```

```
def OR(x1, x2):  
    return MLP(np.array([x1, x2]), w12, b2)
```

```
# AND 게이트
```

```
def AND(x1, x2):  
    return MLP(np.array([x1, x2]), w2, b3)
```

```
# XOR 게이트
```

```
def XOR(x1, x2):  
    return AND(NAND(x1, x2), OR(x1, x2))
```

게이트들(NAND, OR, AND, XOR) 정의

```
# x1, x2 값을 번갈아 대입해 가며 최종값 출력
```

```
if __name__ == '__main__':  
    for x in [(0, 0), (1, 0), (0, 1), (1, 1)]:  
        y = XOR(x[0], x[1])  
        print("입력 값: " + str(x) + " 출력 값: " + str(y))
```

x1, x2값을 번갈아 대입해가며 최종값 출력

입력 값: (0, 0)	출력 값: 0
입력 값: (1, 0)	출력 값: 1
입력 값: (0, 1)	출력 값: 1
입력 값: (1, 1)	출력 값: 0

$$W(1) = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \quad B(1) = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$
$$W(2) = \begin{bmatrix} w_{31} \\ w_{32} \end{bmatrix} \quad B(2) = [b_3]$$
$$W(1) = \begin{bmatrix} -2 & 2 \\ -2 & 2 \end{bmatrix} \quad B(1) = \begin{bmatrix} 3 \\ -1 \end{bmatrix}$$
$$W(2) = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad B(2) = [-1]$$

우리가 원하는 XOR문제 정답 도출됨
-> 은닉층을 만들어 XOR문제 해결!
-> ANN(Artificial Neural Net, 인공신경망)

ch8.

back-propagation
(오차 역전파)

8장 목차

- 1 | 오차 역전파의 개념
- 2 | 코딩으로 확인하는 오차 역전파

back-propagation(오차 역전파)

- 신경망 내부의 가중치는 오차 역전파 방법을 사용해 수정함
- 오차 역전파는 경사 하강법의 확장 개념임
- **가중치를 구하는 방법은 경사 하강법을 그대로 이용하면 됨**
- **임의의 가중치**를 선언하고 결과값을 이용해 오차를 구한 뒤 오차가 최소인 지점으로 계속해서 조금씩 이동시킴(최적화)
-> **오차 역전파(back propagation)**
- 이 오차가 최소가 되는 점(미분했을 때 기울기가 0이 되는 지점)을 찾으면 그것이 바로 정답

“임의의 가중치를 선언하고 결과값의 오차를 구해
이를 토대로 하나 앞선 가중치를 차례로 거슬러 올라가며 조정해 나간다”

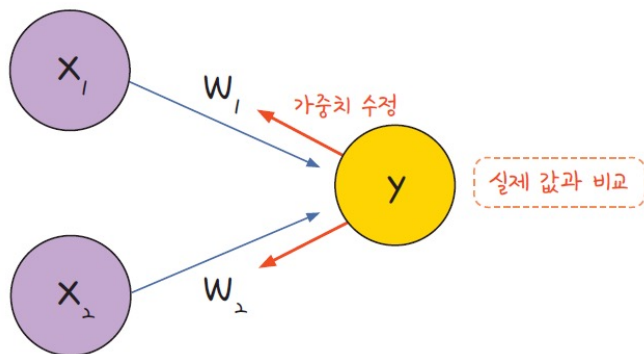


그림 8-1 단일 퍼셉트론에서의 오차 수정

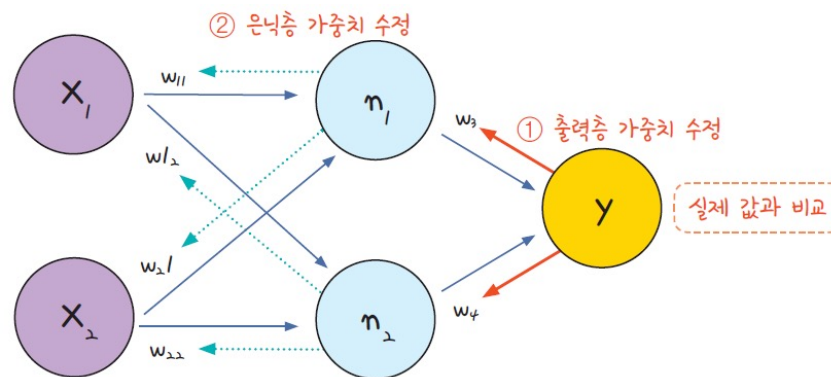



그림 8-2 다층 퍼셉트론에서의 오차 수정

back-propagation(오차 역전파)

- 오차 역전파(back propagation) : 다층 퍼셉트론에서의 최적화 과정
- **오차 역전파 구동 방식**은 다음과 같이 정리할 수 있음
 - 1 | 임의의 초기 가중치 w 를 준 뒤 결과 y_{out} 를 계산함
 - 2 | 계산 결과와 우리가 원하는 값 사이의 오차를 구함
 - 3 | 경사 하강법을 이용해 바로 앞 가중치를 오차가 작아지는 방향으로 가중치 업데이트함
 - 4 | 위 과정을 더이상 오차가 줄어들지 않을 때까지 반복함
- '오차가 작아지는 방향으로 업데이트'는 의미는 미분 값이 0에 가까워지는 방향으로 나아간다는 말임
- 즉, '기울기가 0이 되는 방향'으로 나아가야 하는데, 이 말은 가중치에서 기울기를 뺐을 때 가중치의 변화가 전혀 없는 상태를 말함
- **오차 역전파** : 가중치에서 기울기를 빼도 값의 변화가 없을 때까지 계속 가중치 수정 작업을 반복하는 것.

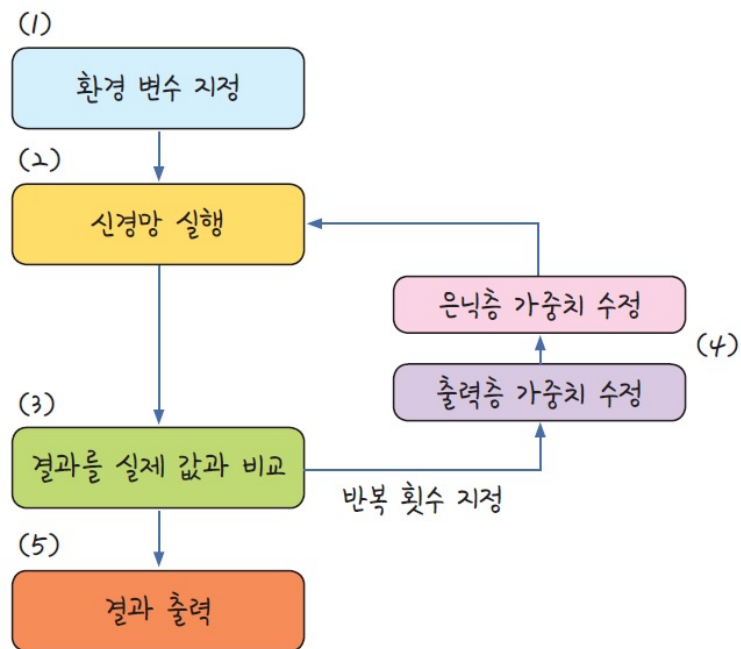
이를 수식으로 표현하면,

새 가중치는 현 가중치에서 '가중치에 대한 기울기'를 뺀 값


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

back-propagation(오차 역전파)

- 입력된 실제 값과 다층 퍼셉트론의 계산 결과를 비교하여 가중치를 역전파 방식으로 수정해 가는 코딩



- 각각을 조금 더 자세히 설명하면 다음과 같음

- 1 | **환경 변수** : 입력 값과 타겟 결괏값이 포함된 데이터셋, 학습률, 활성화 함수, 가중치 등
- 2 | **신경망 실행** : 초깃값을 입력하여 활성화 함수와 가중치를 거쳐 결괏값이 나오게 함
- 3 | **결과를 실제 값과 비교** : 오차를 측정함
- 4 | **역전파 실행** : 출력층과 은닉층의 가중치를 수정함
- 5 | **결과 출력**

ch9.

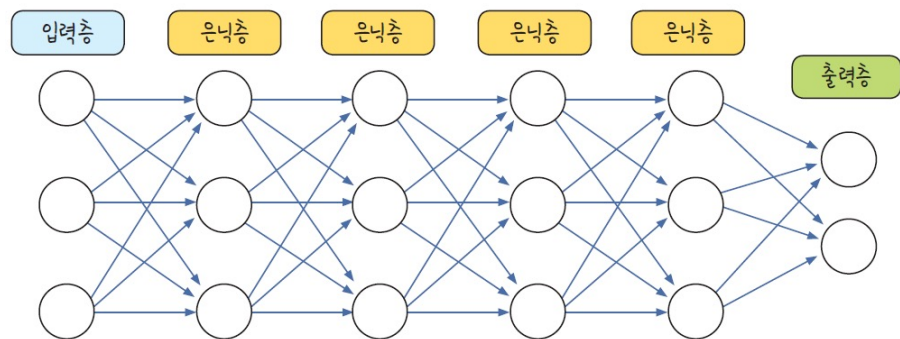
신경망에서
딥러닝으로

9장 목차

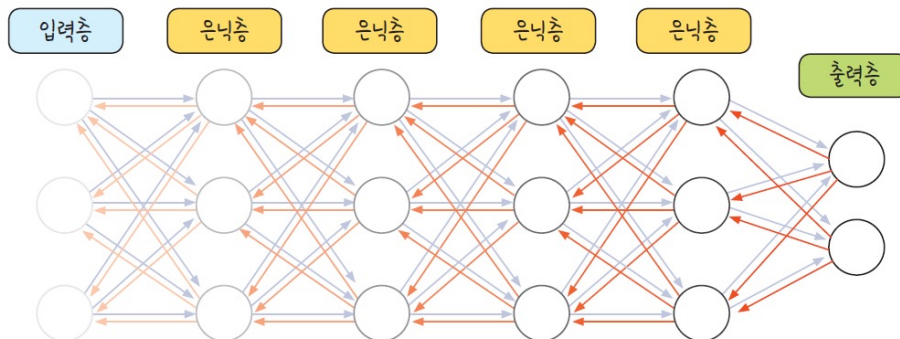
- 1 | 기울기 소실 문제와 활성화 함수
- 2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

기울기 소실 문제(vanishing gradient)

- 다층 퍼셉트론이 오차 역전파를 만나 신경망이 되었고, 신경망은 XOR문제를 가볍게 해결함
- 이제 신경망을 차곡차곡 쌓아올리면 마치 사람처럼 생각하고 판단하는 인공지능이 금방이라도 완성될 것처럼 보였지만 실제로는 결과가 좋지 않음. 그 원인은 **기울기 소실(vanishing gradient) 문제**

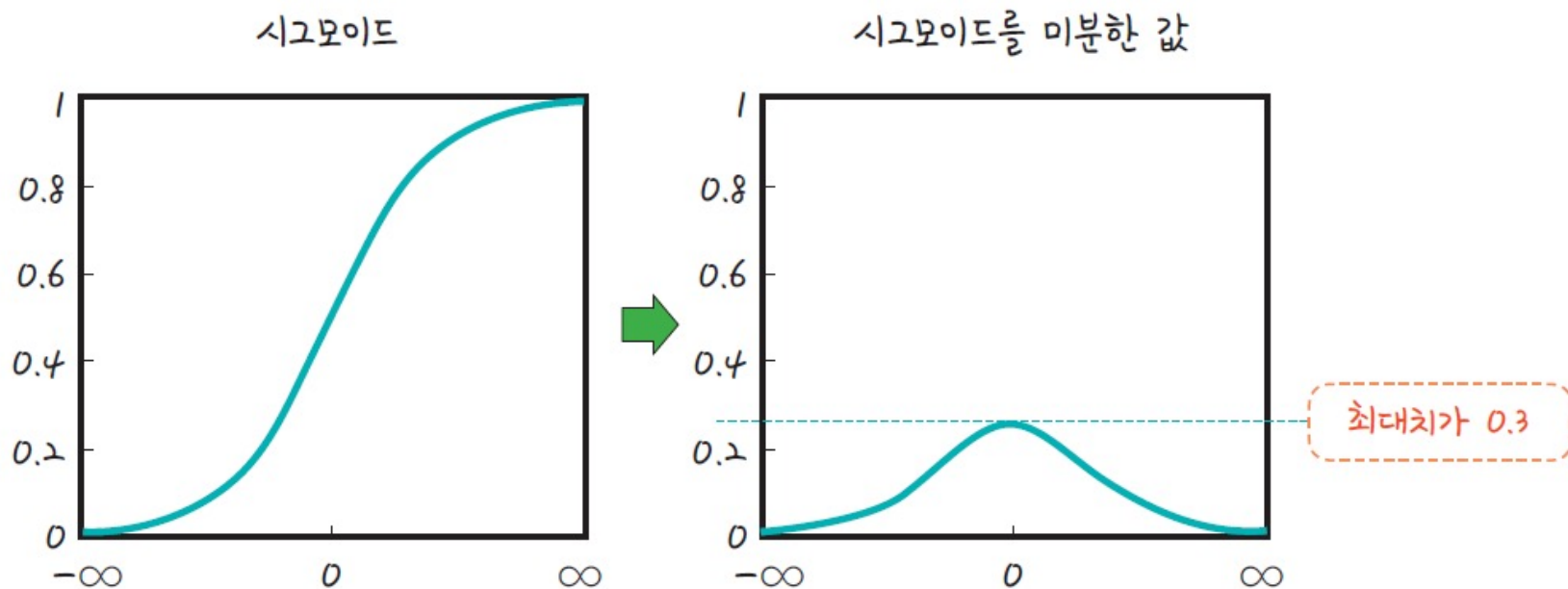


- 가중치를 수정하려면 미분 값, 즉 기울기가 필요하다고 배웠는데, 층이 늘어나면서 기울기 값이 점점 작아져 맨 처음 층까지 전달되지 않는 **기울기 소실 문제** 발생



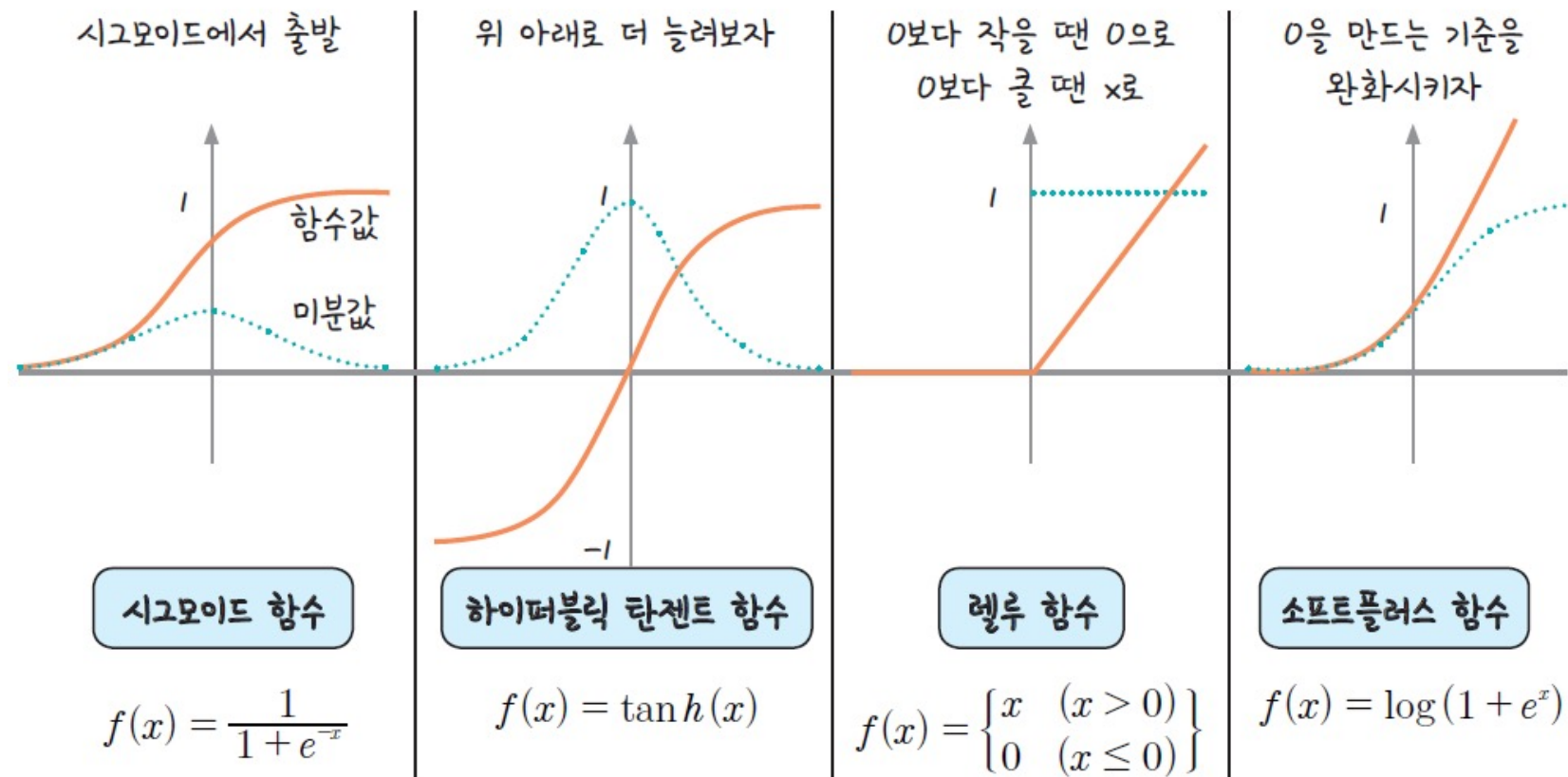
기울기 소실 문제와 활성화 함수

- 기울기 소실 문제가 발생하기 시작한 것은 활성화 함수로 사용된 **시그모이드 함수의 특성** 때문임



- 시그모이드 함수를 미분하면 최대치가 0.3임. 1보다 작으므로 계속 곱하다 보면 0에 가까워짐.
따라서 **여러 층을 거칠수록 기울기가 사라져 가중치를 수정하기가 어려워지는 것임**
- 이를 대체하기 위해 **활성화 함수를 시그모이드가 아닌 다른 함수들로 대체하기 시작**

그림 9-4 여러 활성화 함수의 도입



기울기 소실 문제와 활성화 함수

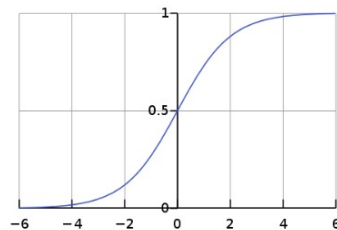
- 하이퍼볼릭 탄젠트(tanh) 함수
 - 미분한 값의 범위가 함께 확장되는 효과를 가져옴
 - 다양한 여전히 1보다 작은 값이 존재하므로 기울기 소실 문제는 사라지지 않음
- 렐루(ReLU) 함수
 - 시그모이드 함수의 대안으로 떠오르며 현재 **가장 많이 사용되는 활성화 함수**
 - 여러 은닉층을 거치며 곱해지더라도 맨 처음 층까지 사라지지 않고 남아있을 수 있음
 - 이 간단한 방법이 여러 층을 쌓을 수 있게 했고, 이로써 딥러닝의 발전에 속도가 붙게 됨
- 소프트플러스 (softplus) 함수
 - 이후 렐루의 0이 되는 순간을 완화

Activation function 종류

Sigmoid에서 시작된 활성화 함수는 ReLU를 비롯해 다양한 종류가 있다.

활성화 함수

sigmoid



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

설명

0~1의 값이
출력됨

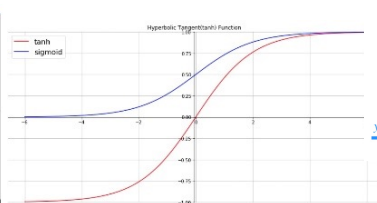
출력값

0 or 1

사용처

이진 분류 출력층

tanh
(hyperbolic
tangent)

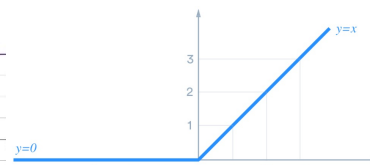


$$\tanh(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

위 아래로 늘리기

-1 ~ 1

relu



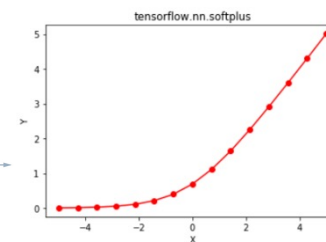
$$f(x) = \max(0, x)$$

0보다 작을 땐 0
0보다 클 땐 그대로

0 or X

Vanishing 해결
은닉층에 사용.

softplus

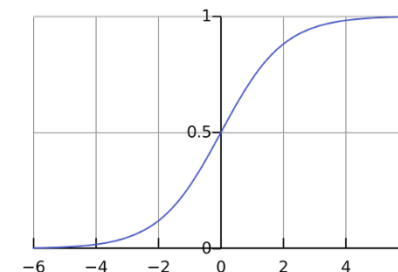


$$f(x) = \ln(1 + e^x)$$

0을 만드는 기준을
완화시키기

0 or X

softmax



$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^K e^{z_j}}$$

0과 1 사이 값이
여러 개 출력됨

0 ~ 1

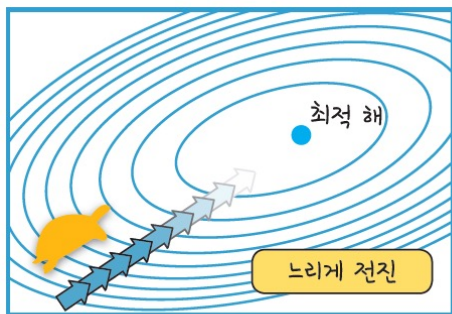
다중 분류 출력층

고급 경사 하강법 - 확률적 경사 하강법(SGD)

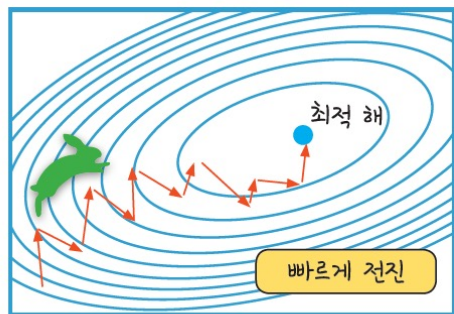
- 가중치를 업데이트하는 방법으로 우리는 경사 하강법을 배웠음
- 경사 하강법은 정확하게 가중치를 찾아가지만,
한 번 업데이트할 때마다 전체 데이터를 미분해야 하므로 계산량이 매우 많다는 단점이 있음
- 경사 하강법은 불필요하게 많은 계산량은 속도를 느리게 할 뿐 아니라, 최적 해를 찾기 전 최적화 과정이 멈출 수도 있음
- 이러한 점을 보완한 고급 경사 하강법이 등장하면서 딥러닝의 발전 속도는 더 빨라짐

확률적 경사 하강법(Stochastic Gradient Descent, SGD)

- 전체 데이터를 사용하는 것이 아니라, 랜덤하게 추출한 일부 데이터를 사용함
- 일부 데이터를 사용하므로 더 빨리 그리고 자주 업데이트를 하는 것이 가능해짐



경사 하강법



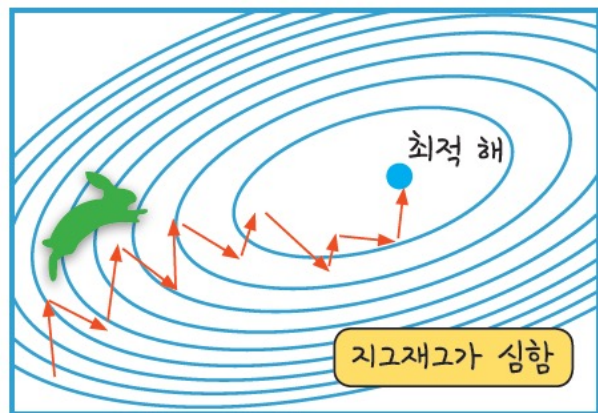
확률적 경사 하강법

- 속도가 빠르고 최적 해에 근사한 값을 찾아낸다는 장점 덕분에 경사 하강법의 대안으로 사용됨

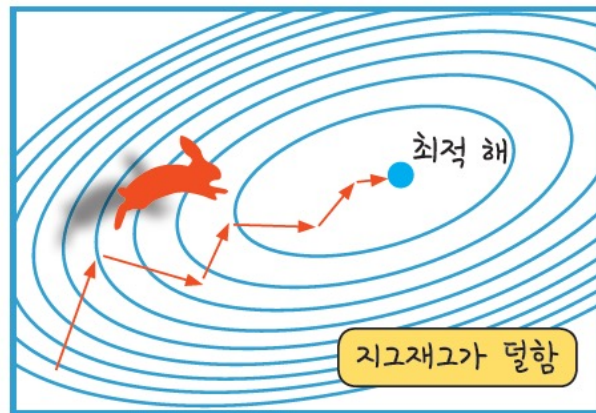
고급 경사 하강법 - 모멘텀(momentum)

모멘텀

- 모멘텀(momentum)이란 단어는 '관성, 탄력, 가속도'라는 뜻
- 모멘텀 SGD란 말 그대로 **경사 하강법에 탄력을 더해 주는 것**
- 다시 말해서, 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 **바로 앞 수정 값과 방향 (+, -)을 참고**하여 같은 방향으로 일정한 비율만 수정되게 하는 방법
- 수정 방향이 양수(+) 방향으로 한 번, 음수(-) 방향으로 한 번 지그재그로 일어나는 현상이 줄어들고, 이전 이동 값을 고려하여 일정 비율만큼만 다음 값을 결정하므로 **관성의 효과**를 낼 수 있음



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

고급 경사 하강법 정리

표 9-1 딥러닝 구동에 사용되는 고급 경사 하강법 개요 및 케라스 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	<code>keras.optimizers.SGD(lr = 0.1)</code> 케라스 최적화 함수를 이용합니다.
모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	<code>keras.optimizers.SGD(lr = 0.1, momentum = 0.9)</code> 모멘텀 계수를 추가합니다.
네스테로프 모멘텀 (NAG)	모멘텀이 이동시킬 방향으로 미리 이동해서 그레이디언트를 계산. 불필요한 이동을 줄이는 효과	정확도 개선	<code>keras.optimizers.SGD(lr = 0.1, momentum = 0.9, nesterov = True)</code> 네스테로프 옵션을 추가합니다.
아다그라드 (Adagrad)	변수의 업데이트가 잦으면 학습률을 적게 하여 이동 보폭을 조절하는 방법	보폭 크기 개선	<code>keras.optimizers.Adagrad(lr = 0.01, epsilon = 1e - 6)</code> 아다그라드 함수를 사용합니다. ※ 참고: 여기서 <code>epsilon</code> , <code>rho</code> , <code>decay</code> 같은 파라미터는 바꾸지 않고 그대로 사용하기를 권장하고 있습니다. 따라서 <code>lr</code> , 즉 <code>learning rate</code> (학습률) 값만 적절히 조절하면 됩니다.
알엠에스프롭 (RMSProp)	아다그라드의 보폭 민감도를 보완한 방법	보폭 크기 개선	<code>keras.optimizers.RMSprop(lr = 0.001, rho = 0.9, epsilon = 1e - 08, decay = 0.0)</code> 알엠에스프롭 함수를 사용합니다.
아담(Adam)	모멘텀과 알엠에스프롭 방법을 합친 방법	정확도와 보폭 크기 개선	<code>keras.optimizers.Adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999, epsilon = 1e - 08, decay = 0.0)</code> 아담 함수를 사용합니다.

현재 가장 많이 사용되는
고급 경사 하강법