신경망의 이해

퍼셉트론

1 | 가중치, 가중합, 바이어스, 활성화 함수

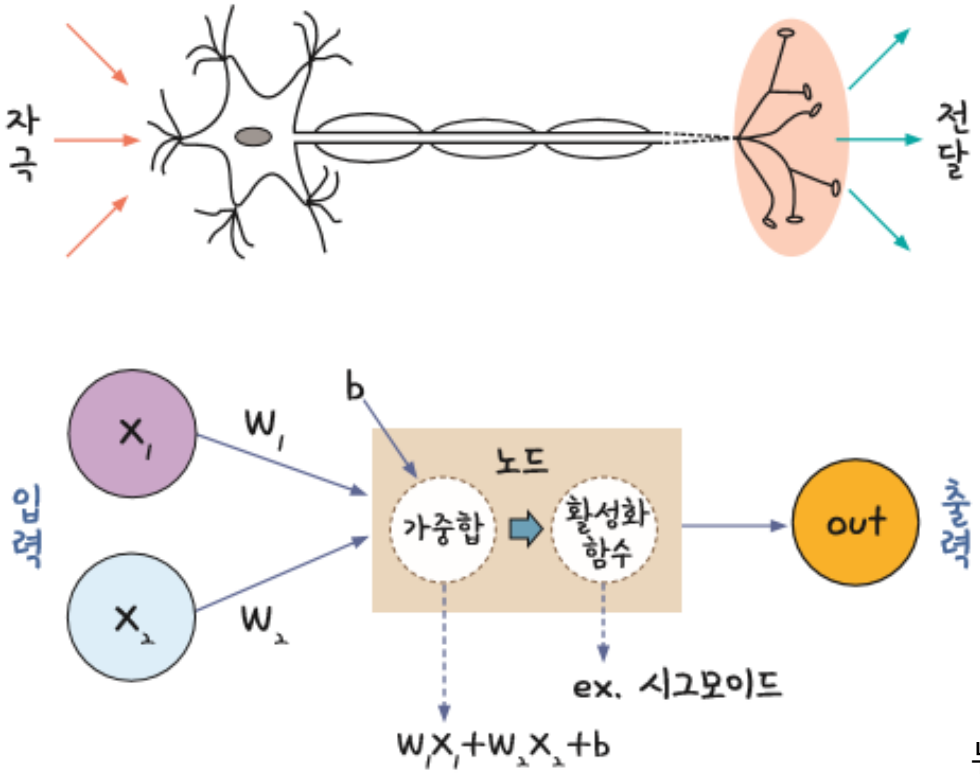
2 | 퍼셉트론의 과제

3 | XOR 문제

- 인간의 뇌는 치밀하게 연결된 약 1000억 개의 뉴런으로 이루어져 있음

- 뉴런과 뉴런 사이에는 시냅스라는 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 → **퍼셉트론의 개념과 유사!**



뉴런과 퍼셉트론의 비교

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

1 | 가중치, 가중합, 바이어스, 활성화 함수

- 용어를 정리해 보자.
- 기울기 a나 y 절편 b와 같은 용어를 퍼셉트론의 개념에 맞춰 좀 더 '딥러닝답게' 표현해 보면 다음과 같음

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

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

- 먼저 기울기 a는 퍼셉트론에서는 **가중치**를 의미하는 w(weight)로 표기됨
- y 절편 b는 똑같이 b라고 씀, 하지만 y = ax + b의 b가 아니라 편향, 선입견이라는 뜻인 **바이어스**(bias)에서 따온 b

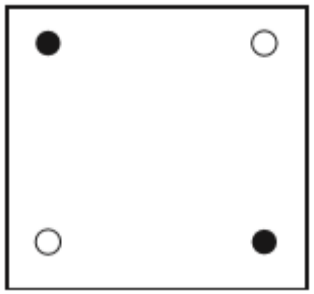
1 | 가중치, 가중합, 바이어스, 활성화 함수

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

- 가중합의 결과를 놓고 1 또는 0을 출력해서 다음으로 보냄

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

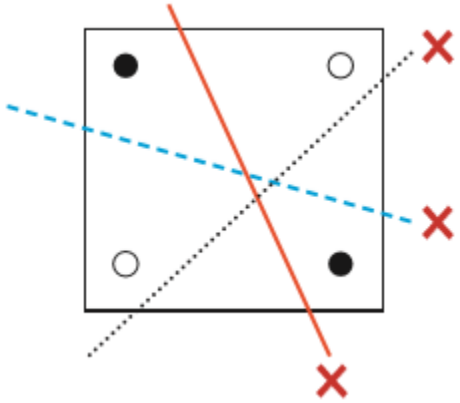
2 | 퍼셉트론의 과제



사각형 종이에 놓인 검은점 두 개와 흰점 두개

- 사각형 종이에 검은점 두 개와 흰점 두 개가 놓여 있음
- 이 네 점 사이에 직선을 하나 긋는다고 하자
- 이때 직선의 한쪽 편에는 검은점만 있고, 다른 한쪽에는 흰점만 있게끔 선을 그을 수 있을까?

2 | 퍼셉트론의 과제



선으로는 같은 색끼리 나눌 수 없다: 퍼셉트론의 한계

- 여러 개의 선을 아무리 그어보아도 하나의 직선으로는 흰점과 검은점을 구분할 수 없음

2 | 퍼셉트론의 과제

- 선형 회귀와 로지스틱 회귀를 통해 머신러닝이 결국 선이나 2차원 평면을 그리는 작업임을 배웠다.
- 따라서 이와 같은 개념인 퍼셉트론 역시 선을 긋는 작업이라고 할 수 있다
- 그런데 이 예시처럼 경우에 따라선 선을 아무리 그어도 해결되지 않는 상황이 있다

3 | XOR 문제

- 이것이 퍼셉트론의 한계를 설명할 때 등장하는 XOR(exclusive OR) 문제
- XOR 문제는 논리 회로에 등장하는 개념

- 컴퓨터는 두 가지의 디지털 값, 즉 0과 1을 입력해 하나의 값을 출력하는 회로가 모여 만들어지는데, 이 회로를 '게이트(gate)'라고 부름

- AND 게이트는 x_1와 x_2 둘 다 1일 때 결과값이 1로 출력됨
- OR 게이트는 둘 중 하나라도 1이면 결과값이 1로 출력됨
- XOR 게이트는 둘 중 하나만 1일 때 1이 출력됨

3 | XOR 문제

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

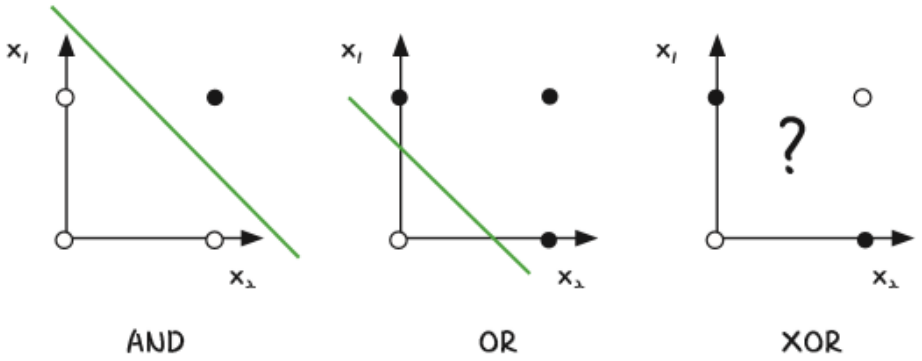
XOR 진리표

x_1	x_2	결괏값
0	0	0
0	1	1
1	0	1
1	1	0

AND, OR, XOR 게이트에 대한 진리표

3 | XOR 문제

- 표 6-1을 각각 그래프로 좌표 평면에 나타내 보자
- 결과값이 0이면 흰점으로, 1이면 검은점으로 나타낸 후 조금 전처럼 직선을 그어 위 조건을 만족할 수 있는지 보자



AND, OR, XOR 진리표대로 좌표 평면에 표현한 뒤 선을 그어 색이 같은 점끼리 나누기(XOR는 불가능)

- AND와 OR 게이트는 직선을 그어 결과값이 1인 값(검은점)을 구별할 수 있음
- 그러나 XOR의 경우 선을 그어 구분할 수 없음

3 | XOR 문제

- 이는 인공지능 분야의 선구자였던 MIT의 마빈 민스키(Marvin Minsky) 교수가 1969년에 발표한 <퍼셉트론즈(Perceptrons)>라는 논문에 나오는 내용

- 10여 년이 지난 후에야 이 문제가 해결되는데, 이를 해결한 개념이 바로 **다층 퍼셉트론**(multilayer perceptron)

다층 퍼셉트론

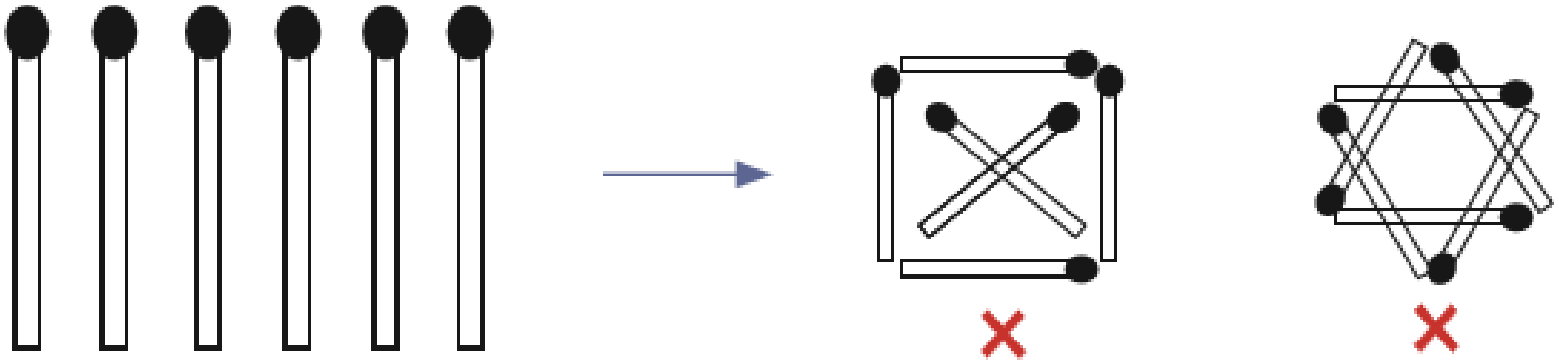
1 | 다층 퍼셉트론의 설계

2 | XOR 문제의 해결

3 | 코딩으로 XOR 문제 해결하기

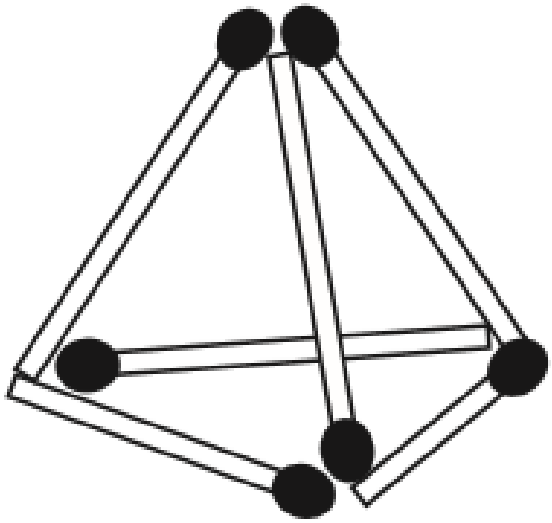
- 앞서 종이 위에 각각 엇갈려 놓인 검은점 두 개와 흰점 두 개를 하나의 선으로는 구별할 수 없다는 것을 살펴보았음

- 이 문제를 해결하려면 새로운 접근이 필요함
- EX) '성냥개비 여섯 개로 정삼각형 네 개를 만들 수 있는가'



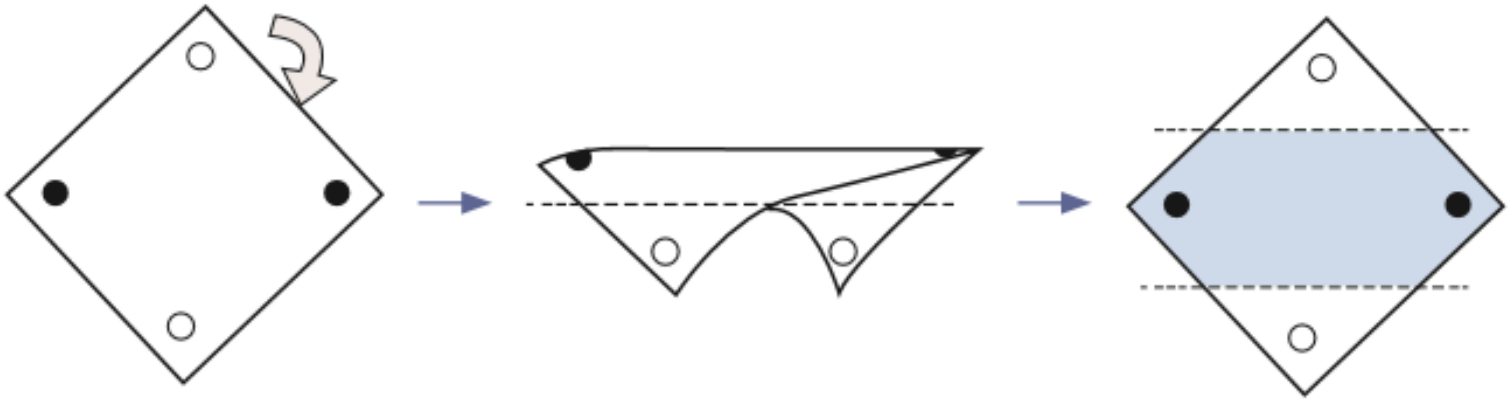
성냥개비 여섯 개로 정삼각형 네 개를?

- 2차원 평면에서만 해결하려는 고정관념을 깨고 피라미드 모양으로 성냥개비를 쌓아 올리면 해결



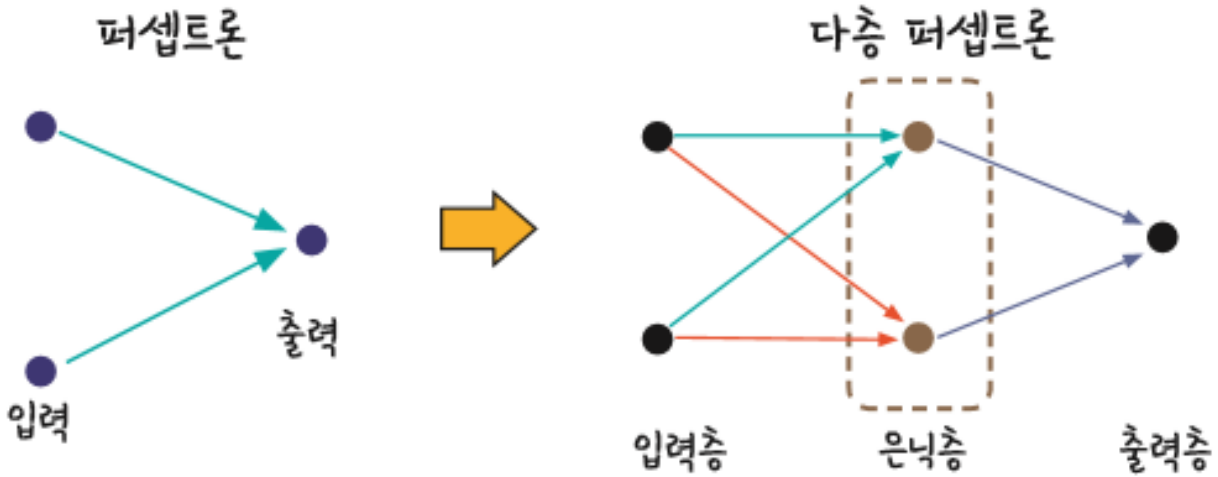
차원을 달리하니 쉽게 완성!

- XOR 문제를 극복은 평면을 휘어주는것! 즉, 좌표 평면 자체에 변화를 주는 것



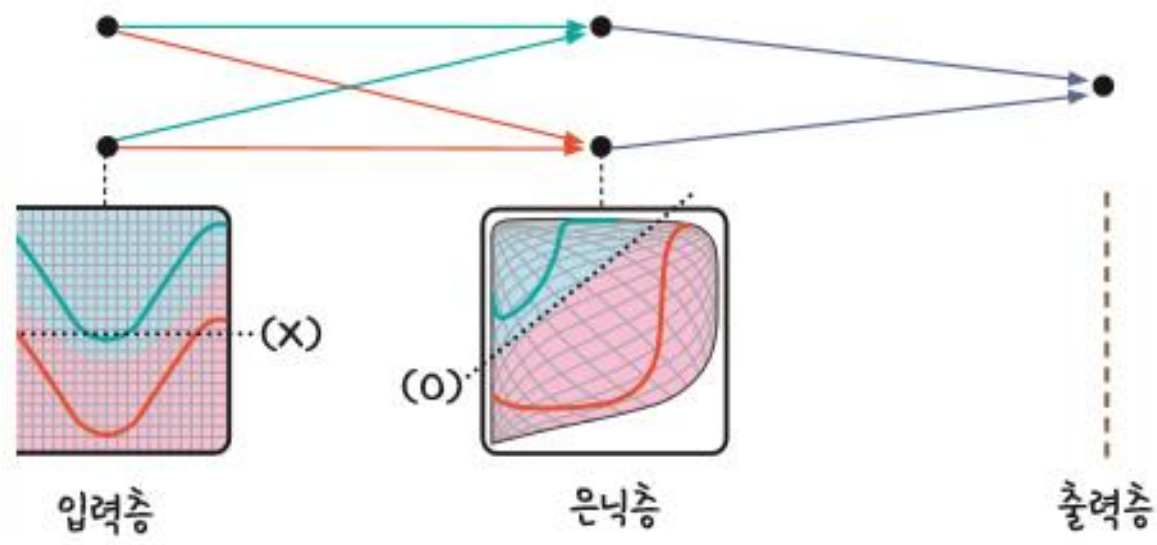
XOR 문제의 해결

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



퍼셉트론에서 다층 퍼셉트론으로

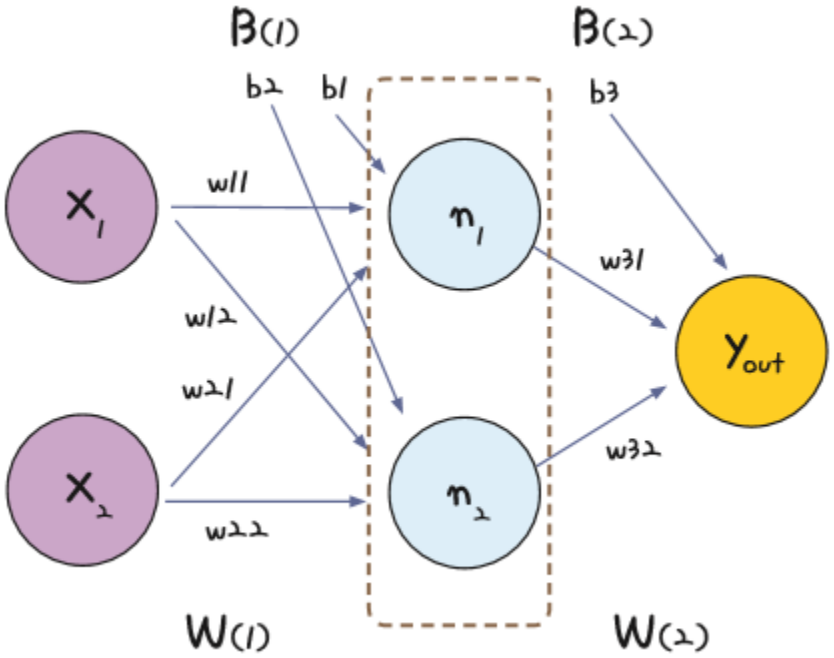
- 입력층과 은닉층의 그래프를 집어넣어 보면 그림과 같음
- 은닉층이 좌표 평면을 왜곡시키는 결과를 가져옴 → 두 영역을 가로지르는 선이 직선으로 바뀜



은닉층의 공간 왜곡(https://goo.gl/8qEGHD 참조)

1 | 다층 퍼셉트론의 설계

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



다중 퍼셉트론의 내부

1 | 다층 퍼셉트론의 설계

- 가운데 숨어있는 은닉층으로 퍼셉트론이 각각 자신의 가중치(w)와 바이어스(b) 값을 보내고, 이 은닉층에서 모인 값이 한 번 더 시그모이드 함수(기호로 σ라고 표시)를 이용해 최종 값으로 결과를 보냄
- 은닉층에 모이는 중간 정거장을 노드(node)라고 하며, 여기서는 n_1과 n_2로 표현

1 | 다층 퍼셉트론의 설계

- n_1과 n_2의 값은 각각 단일 퍼셉트론의 값과 같음

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

- 위 두 식의 결과값이 출력층으로 보내짐
- 출력층에서는 역시 시그모이드 함수를 통해 y 값이 정해짐

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

1 | 다층 퍼셉트론의 설계

- 이제 각각의 가중치(x)와 바이어스(b)의 값을 정할 차례
- 2차원 배열로 늘어놓으면 다음과 같이 표시할 수 있음
- 은닉층을 포함해 가중치 6개와 바이어스 3개가 필요함

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

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

2 | XOR 문제의 해결

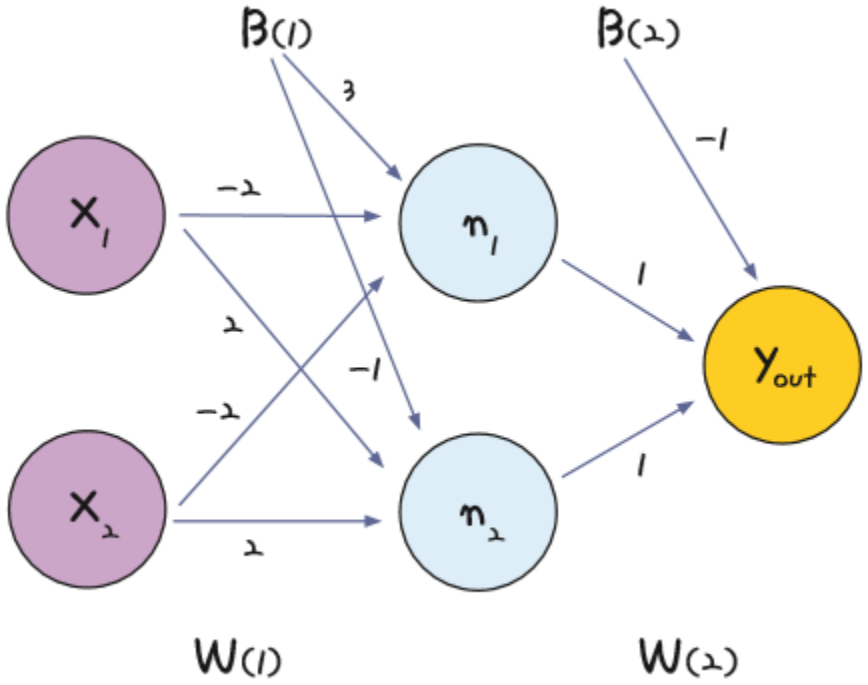
- 앞서 우리에게 어떤 가중치와 바이어스가 필요한지를 알아보았음
- 이를 만족하는 가중치와 바이어스의 조합은 무수히 많음
- 지금은 먼저 각 변숫값을 정하고 이를 이용해 XOR 문제를 해결하는 과정을 알아보자

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

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

2 | XOR 문제의 해결

- 이것을 도식에 대입하면 다음과 같음



다중 퍼셉트론의 내부에 변수를 채워보자.

2 | XOR 문제의 해결

- 이제 x_1의 값과 x_2의 값을 각각 입력해 y 값이 우리가 원하는 값으로 나오는지를 점검해 보자

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

XOR 다층 문제 해결

- 표에서 볼 수 있듯이 n_1, n_2, y를 구하는 공식에 차례로 대입하니 우리가 원하는 결과를 구할 수 있었음
- 숨어있는 두 개의 노드를 둔 다층 퍼셉트론을 통해 XOR 문제가 해결된 것

3 | 코딩으로 XOR 문제 해결하기

- 이제 주어진 가중치와 바이어스를 이용해 XOR 문제를 해결하는 파이썬 코드를 작성해 보자

- 표에서 n_1의 값을 잘 보면 입력 값 x_1, x_2가 모두 1일 때 0을 출력하고 하나라도 0이 아니면 1을 출력하게 되어 있음
- 이는 표에서 배운 AND 게이트의 정 반대 값을 출력하는 방식 (이를 NAND 게이트라고 부름)
- n_2의 값을 잘 보면 x_1, x_2에 대한 OR 게이트에 대한 답
- NAND 게이트와 OR 게이트, 이 두 가지를 내재한 각각의 퍼셉트론이 다중 레이어 안에서 각각 작동하고, 이 두 가지의 값에 대해 AND 게이트를 수행한 값이 바로 우리가 구하고자 하는 Y_{out}임을 알 수 있음

3 | 코딩으로 XOR 문제 해결하기

- 정해진 가중치와 바이러스를 넘파이 라이브러리를 사용해 다음과 같이 선언

```
import numpy as np

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

3 | 코딩으로 XOR 문제 해결하기

- 이제 퍼셉트론 함수를 만들어 보자
- 0과 1 중에서 값을 출력하게 설정

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

3 | 코딩으로 XOR 문제 해결하기

- 각 게이트의 정의에 따라 NAND 게이트, OR 게이트, AND 게이트, XOR 게이트 함수를 만들어 보자

```
# 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))
```

3 | 코딩으로 XOR 문제 해결하기

- 이제 x_1과 x_2 값을 번갈아 대입해 가며 최종 값을 출력해 보자

```
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))
```

3 | 코딩으로 XOR 문제 해결하기

다층 퍼셉트론으로 XOR 문제 해결하기

- XOR.py

```
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
```

3 | 코딩으로 XOR 문제 해결하기

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

# 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)
```

3 | 코딩으로 XOR 문제 해결하기

```
# 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))

# 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))
```

3 | 코딩으로 XOR 문제 해결하기

- 실행 결과

```
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
```

- 우리가 원하는 XOR 문제의 정답이 도출됨
- 이렇게 퍼셉트론 하나로 해결되지 않던 문제를 은닉층을 만들어 해결
- 은닉층을 여러 개 쌓아올려 복잡한 문제를 해결하는 과정이 뉴런이 복잡한 과정을 거쳐 사고를 낳는 사람의 신경망을 닮음
- 그래서 이 방법을 인공 신경망이라 부르기 시작했고, 이를 간단히 줄여서 **신경망**이라고 통칭

오차 역전파

1 | 오차 역전파의 개념

2 | 코딩으로 확인하는 오차 역전파

오차 역전파의 개념

- 퍼셉트론으로 해결되지 않던 문제를 신경망을 이용해 해결했음
- 신경망 내부의 가중치는 오차 역전파 방법을 사용해 수정
- 오차 역전파는 경사 하강법의 확장 개념
- 이 장에서는 오차 역전파의 기본 개념과 반드시 알아야 할 점을 짚어보자

1 | 오차 역전파의 개념

- 지금까지 입력 값과 출력 값을 알고 있는 상태에서 중간에 은닉층을 두는 다층 퍼셉트론의 개념에 대해서 공부했음
- 우리가 구해야 할 가중치(w)와 바이어스(b)가 무엇인지도 알아보았음

- 그런데 우리는 앞서 XOR 문제를 해결할 때 정답에 해당하는 가중치와 바이어스를 미리 알아본 후 이를 집어넣었음
- 실제 프로젝트에서는 **경사 하강법**을 이용한다!

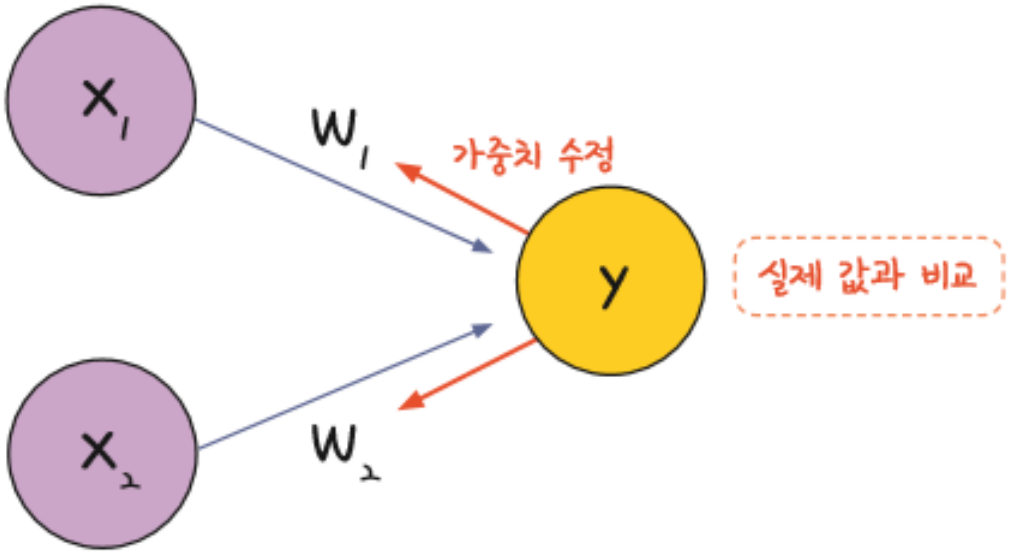
1 | 오차 역전파의 개념

- 앞서 배운 경사 하강법은 입력과 출력이 하나일 때, 즉 '단일 퍼셉트론'일 경우였음

- 그런데 이번에는 숨어 있는 층이 하나 더 생김

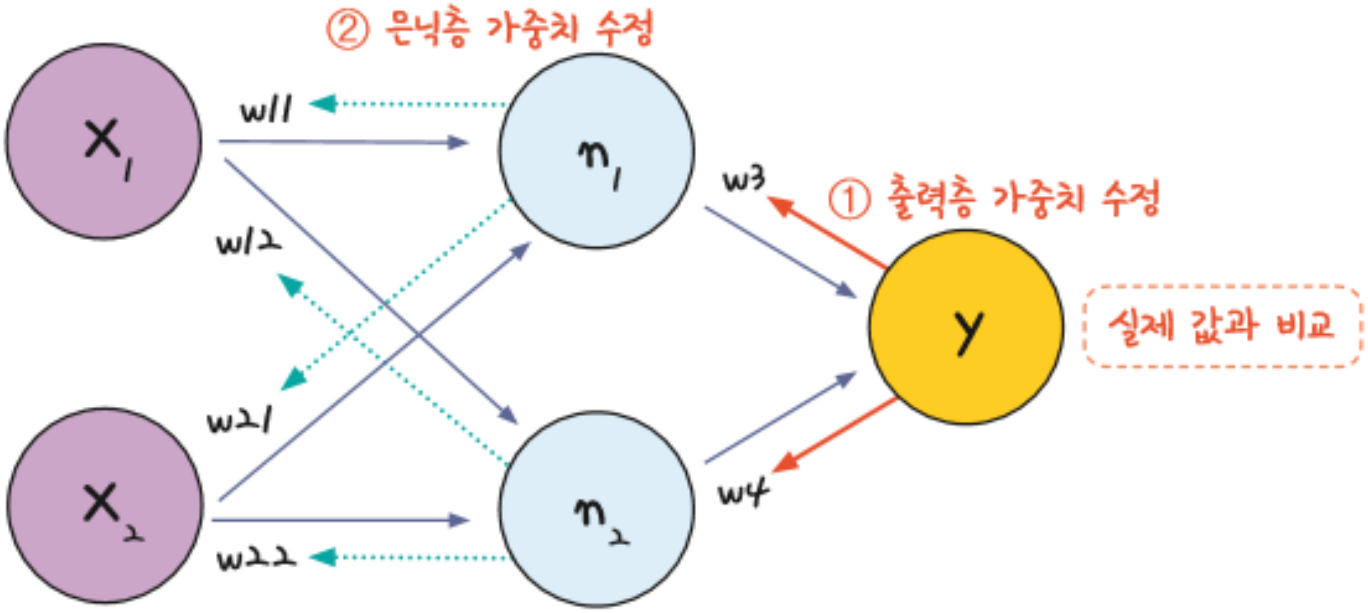
- 단일 퍼셉트론에서 결과값을 얻으면 오차를 구해 이를 토대로 앞 단계에서 정한 가중치를 조정하는 것과 마찬가지로
- 다층 퍼셉트론 역시 결과값의 오차를 구해 이를 토대로 하나 앞선 가중치를 차례로 거슬러 올라가며 조정해 감

1 | 오차 역전파의 개념



단일 퍼셉트론에서의 오차 수정

1 | 오차 역전파의 개념



다층 퍼셉트론에서의 오차 수정

1 | 오차 역전파의 개념

- 그러다 보니 최적화의 계산 방향이 출력층에서 시작해 앞(뒤어서 앞으로)으로 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.

1 | 오차 역전파의 개념

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

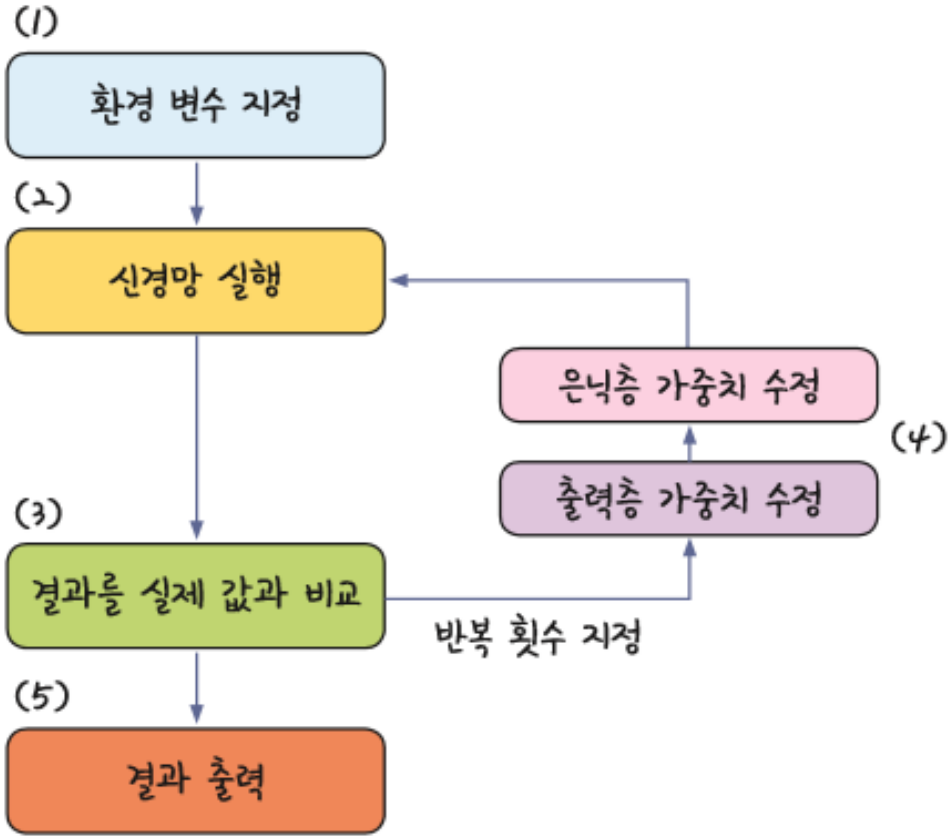
- 즉, '기울기가 0이 되는 방향'으로 나아가야 하는데, 이 말은 가중치에서 기울기를 뺐을 때 가중치의 변화가 전혀 없는 상태를 말함

- 따라서 오차 역전파를 다른 방식으로 표현하면 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



신경망의 구현 과정

2 | 코딩으로 확인하는 오차 역전파

- 각각을 조금 더 자세히 설명하면 다음과 같음

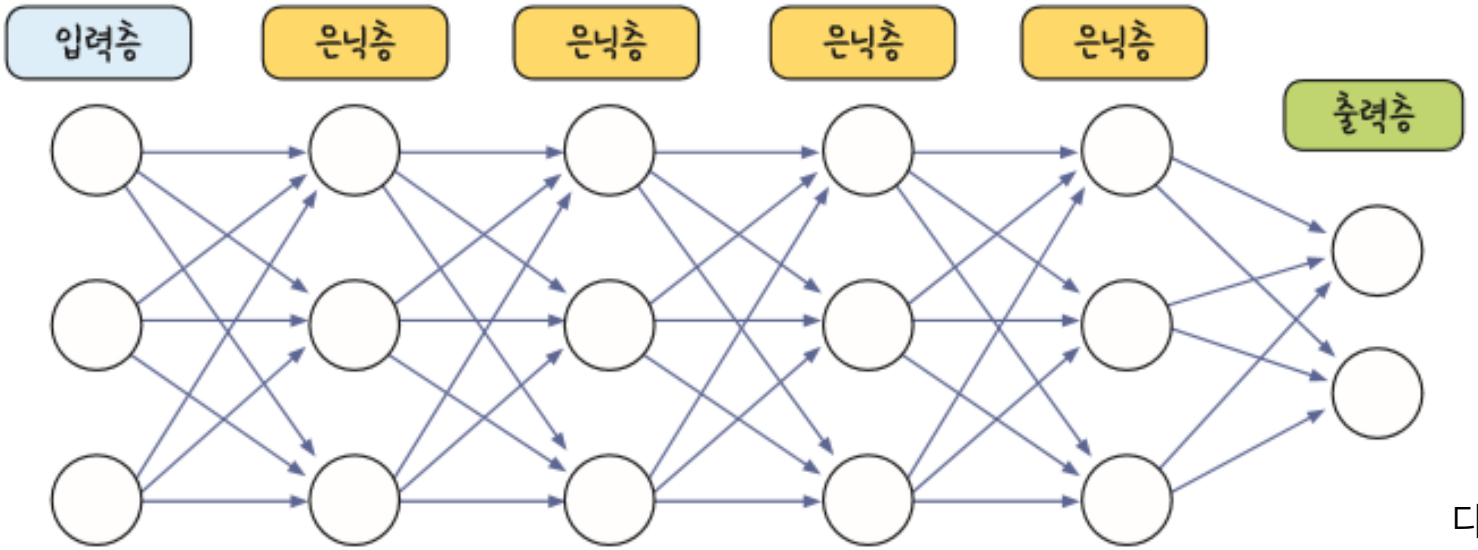
1) **환경 변수 지정**: 환경 변수에는 입력 값과 타깃 결과값이 포함된 데이터셋, 학습률 등이 포함됩니다. 또한, 활성화 함수와 가중치 등도 선언되어야 합니다.
2) **신경망 실행**: 초기값을 입력하여 활성화 함수와 가중치를 거쳐 결과값이 나오게 합니다.
3) **결과를 실제 값과 비교**: 오차를 측정합니다.
4) **역전파 실행**: 출력층과 은닉층의 가중치를 수정합니다.
5) **결과 출력**

신경망에서 딥러닝으로

1 | 기울기 소실 문제와 활성화 함수

2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 다층 퍼셉트론이 오차 역전파를 만나 신경망이 되었고, 신경망은 XOR 문제를 가볍게 해결
- 하지만 기대만큼 결과가 좋아지지 않았음
- 이유가 무엇일까?



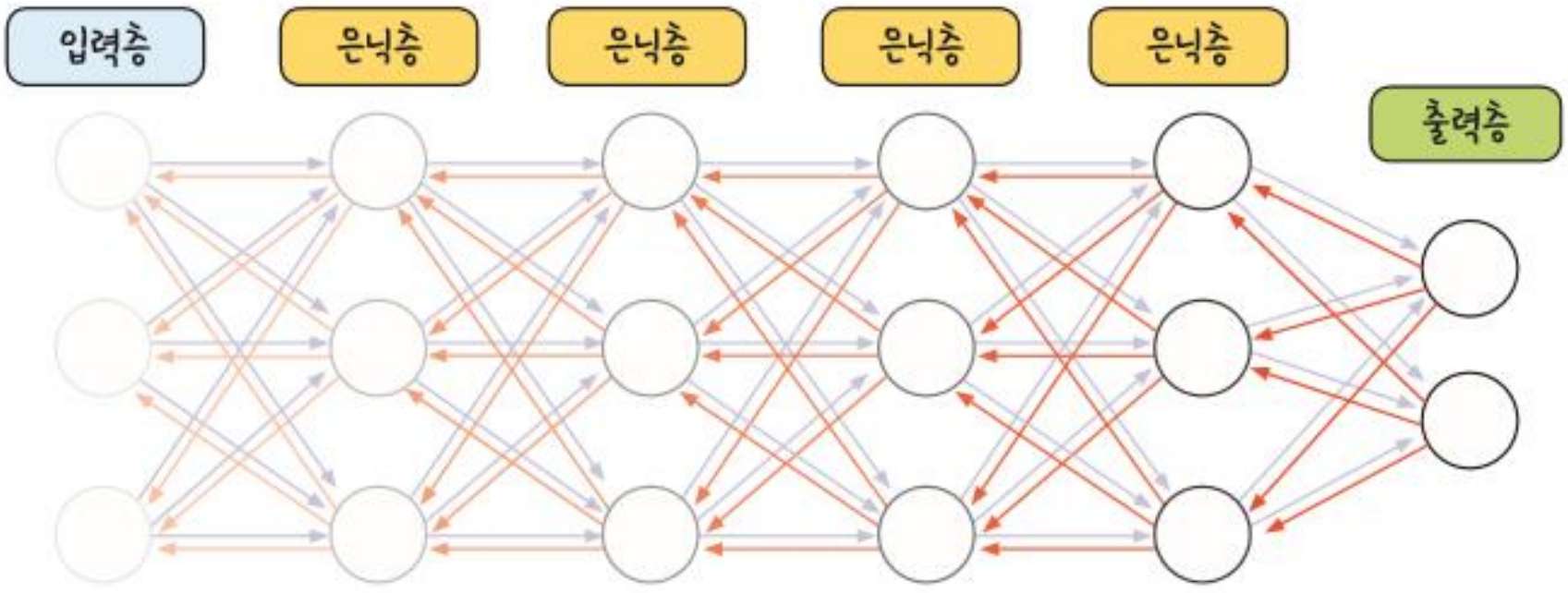
다층 확장

1 | 기울기 소실 문제와 활성화 함수

- 기울기 소실 문제!

- 오차 역전파는 출력층으로부터 하나씩 앞으로 되돌아가며 각 층의 가중치를 수정하는 방법
- 가중치를 수정하려면 미분 값, 즉 기울기가 필요하다고 배움
- 그런데 층이 늘어나면서 기울기가 중간에 0이 되어버리는 **기울기 소실**(vanishing gradient) 문제가 발생하기 시작

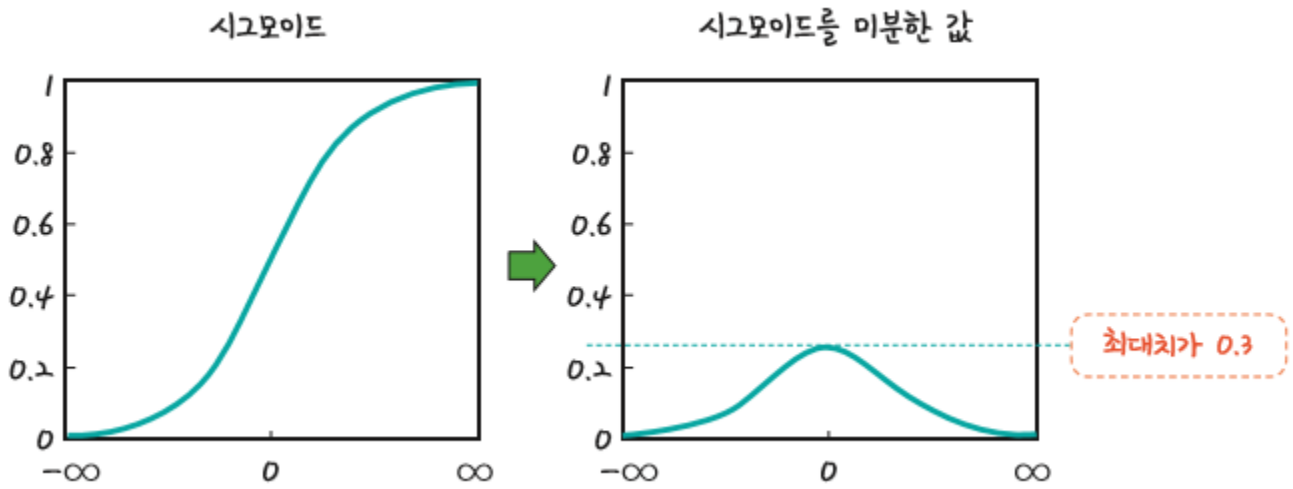
1 | 기울기 소실 문제와 활성화 함수



기울기 소실 문제 발생

1 | 기울기 소실 문제와 활성화 함수

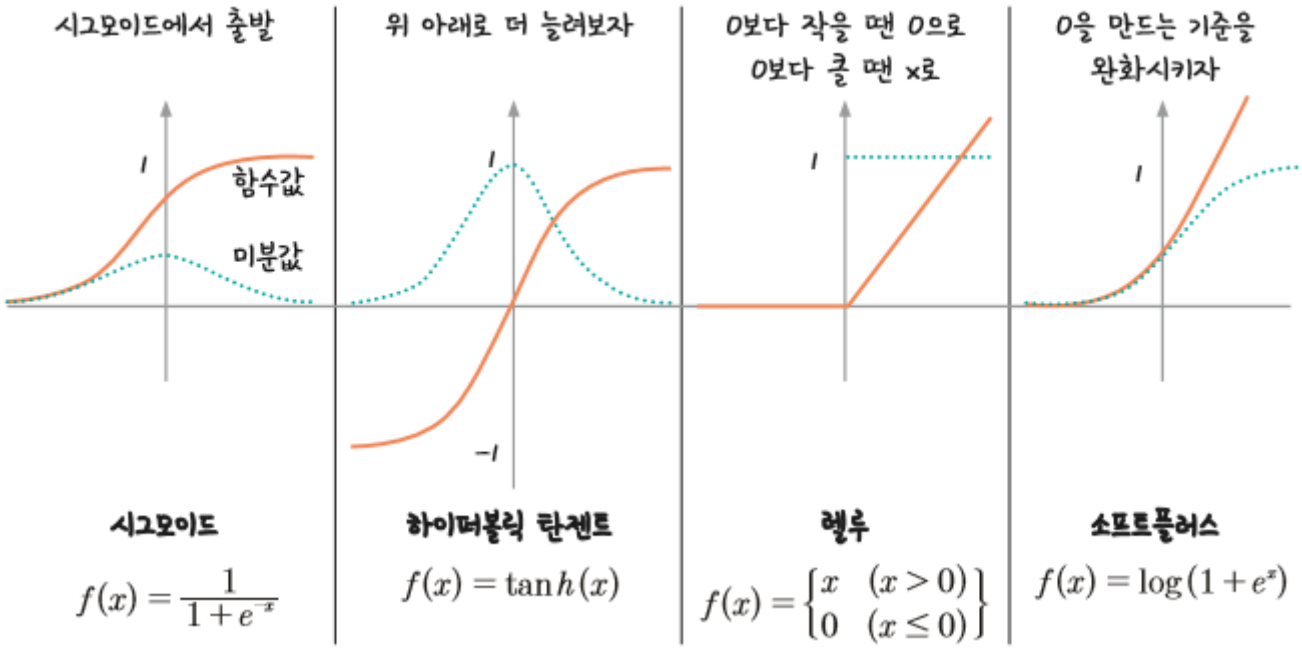
- 이는 활성화 함수로 사용된 시그모이드 함수의 특성 때문임
- 그림에서처럼 시그모이드를 미분하면 최대치가 0.3
- 1보다 작으므로 계속 곱하다 보면 0에 가까워짐
- 따라서 층을 거쳐 갈수록 기울기가 사라져 가중치를 수정하기가 어려워지는 것



시그모이드의 미분

1 | 기울기 소실 문제와 활성화 함수

- 이를 해결하고자 활성화 함수를 시그모이드가 아닌 여러 함수로 대체하기 시작



여러 활성화 함수의 도입

1 | 기울기 소실 문제와 활성화 함수

- 시그모이드 함수의 범위를 -1에서 1로 확장한 개념인 하이퍼볼릭 탄젠트(tanh)는 미분한 값의 범위가 함께 확장되는 효과를 가져왔음
- 하지만 여전히 1보다 작은 값이 존재하므로 기울기 소실 문제는 사라지지 않음

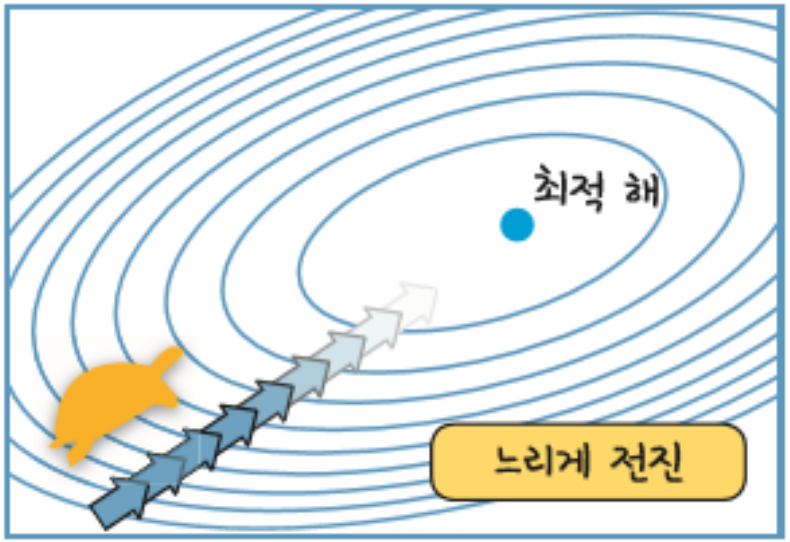
1 | 기울기 소실 문제와 활성화 함수

- 토론토대학교의 제프리 힌튼 교수가 제안한 렐루(ReLU)는 시그모이드의 대안으로 떠오르며 현재 가장 많이 사용되는 활성화 함수
- 렐루는 x가 0보다 작을 때는 모든 값을 0으로 처리하고, 0보다 큰 값은 x를 그대로 사용하는 방법. 이 방법을 쓰면 x가 0보다 크기만 하면 미분 값이 1이 됨
- 따라서 여러 은닉층을 거치며 곱해지더라도 맨 처음 층까지 사라지지 않고 남아 있을 수 있음: 딥러닝의 발전에 속도가 붙게 됨

2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 속도와 정확도 문제!

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

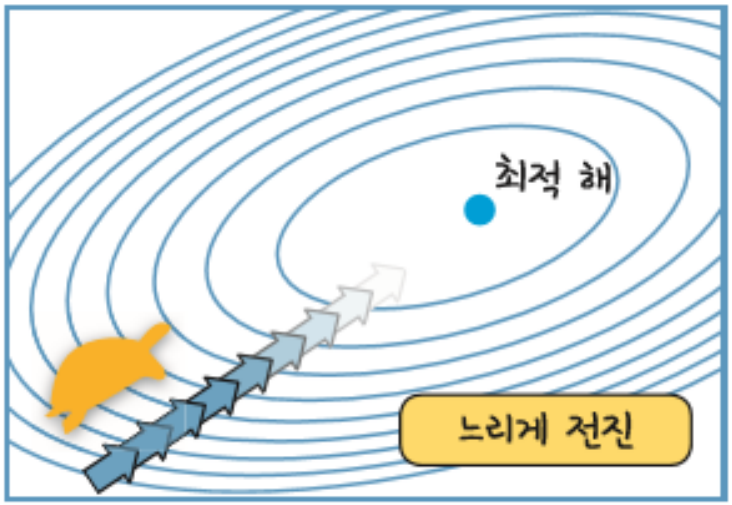


경사 하강법

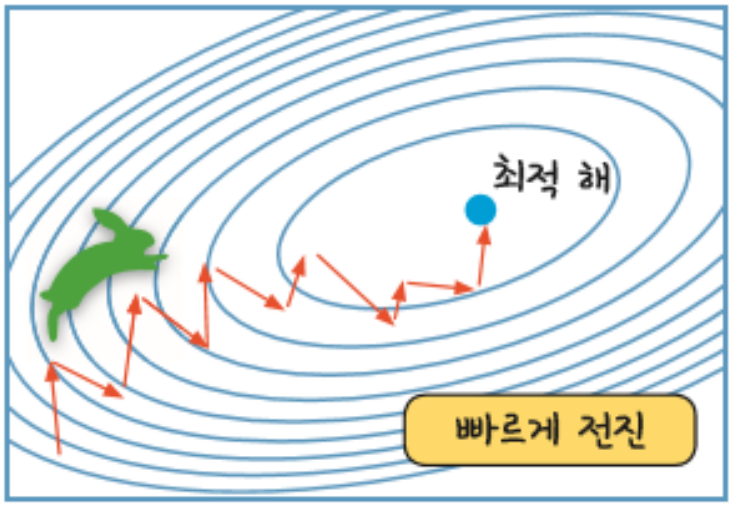
2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

확률적 경사 하강법(SGD)

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



경사 하강법

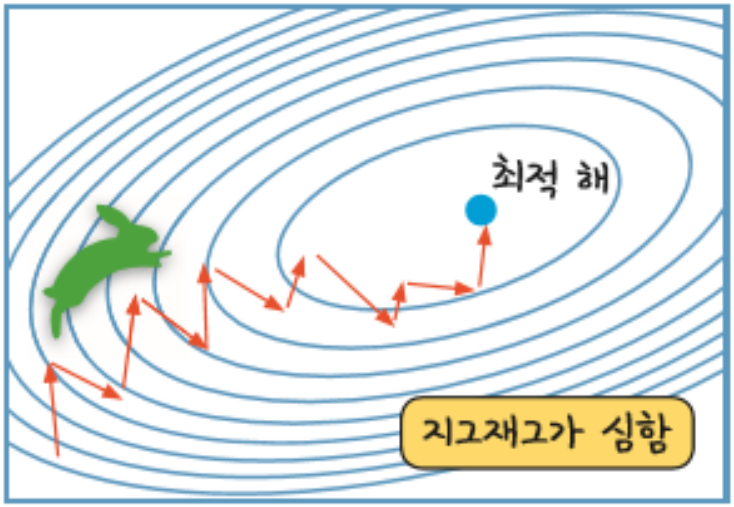


확률적 경사 하강법

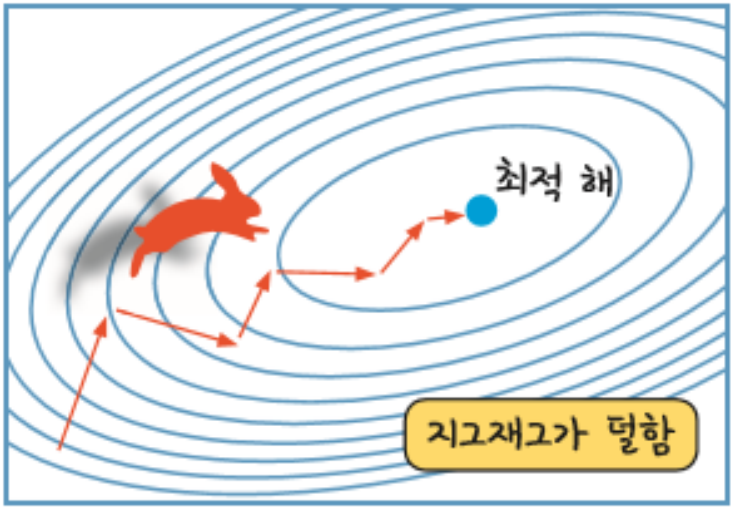
2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

모멘텀

- 경사 하강법과 마찬가지로 매번 기울기를 구하지만, 이를 통해 오차를 수정하기 전 바로 앞 수정 값과 방향(+, -)을 참고하여 같은 방향으로 일정한 비율만 수정되게 하는 방법 (이도에 탄력을 더한다)



확률적 경사 하강법



모멘텀을 적용한 확률적 경사 하강법

2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

- 이밖에 딥러닝 구동에 필요한 고급 경사 하강법과 케라스 내부에서의 활용법 정리

고급 경사 하강법	개요	효과	케라스 사용법
1. 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.
2. 모멘텀 (Momentum)	관성의 방향을 고려해 진동과 폭을 줄이는 효과	정확도 개선	keras.optimizers.SGD(lr = 0.1, momentum = 0.9) 모멘텀 계수를 추가합니다.
3. 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.

2 | 속도와 정확도 문제를 해결하는 고급 경사 하강법

4. 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.
5. 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.
6. 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.

딥러닝 구동에 사용되는 고급 경사 하강법 개요 및 활용법