ROKEY BOOT CAMP

AI(Computer Vision)개론

Chapter 3. 딥러닝을 위한 기초 수학

OOO 강사

학습 목차

딥러닝을 위한 기초 수학

- ‘딥러닝을 배운다’는 말에는 딥러닝의 실행법을 익히는 것뿐 아니라, 딥러닝의 수학 원리를 공부한다는 의미도 담겨 있음
- 원리를 알아야 정확히 실행할 수 있기 때문에 딥러닝의 원리를 이해하는 것은 좋은 코드를 만드는 것 이상으로 중요함
- 딥러닝의 수학 원리를 이해하기 위해서는 당연히 기본적인 수학 지식이 필요함
- 어떤 원리로 입력 값의 패턴을 분석하고 학습하는지 이해하려면 그 배경이 되는 수학 연산을 살펴보아야 하고, 여기에 사용되는 함수들을 알아야 하기 때문임

딥러닝을 위한 기초 수학

- 좋은 소식은 딥러닝 뒤에 있는 수학적 배경이 다른 머신 러닝과 비교했을 때 그다지 어렵지 않다는 것
- 딥러닝은 고등학교 수준의 수학만으로도 원리와 배경을 파악할 수 있음
- 조금 더 깊이 공부하더라도 대학교 교양 강좌 수준을 넘지 않는 범위에서 딥러닝의 원리를 이해할 수 있음
- 이 장에서는 딥러닝을 이해하는 데 꼭 필요한 기초 수학을 먼저 공부하겠음
- 각 수학 공식이 딥러닝의 어느 부분에 활용되는지 참고하면서, 수학에 대한 두려움을 없애고
  딥러닝 공부를 시작할 수 있길 바람

01
일차 함수,
기울기와 y 절편

일차 함수, 기울기와 y 절편

- 함수란 두 집합 사이의 관계를 설명하는 수학 개념
- 변수 x와 y가 있을 때, x가 변하면 이에 따라 y는 어떤 규칙으로 변하는지 나타냄
- 보통 함수를 나타낼 때는 function의 f와 변수 x를 사용해 y =f(x)라고 표시

일차 함수, 기울기와 y 절편

- 일차 함수는 y가 x에 관한 일차식으로 표현된 경우를 의미
- 예를 들어 다음과 같은 함수식으로 나타낼 수 있음

$$y = ax + b \ (a \neq 0)$$

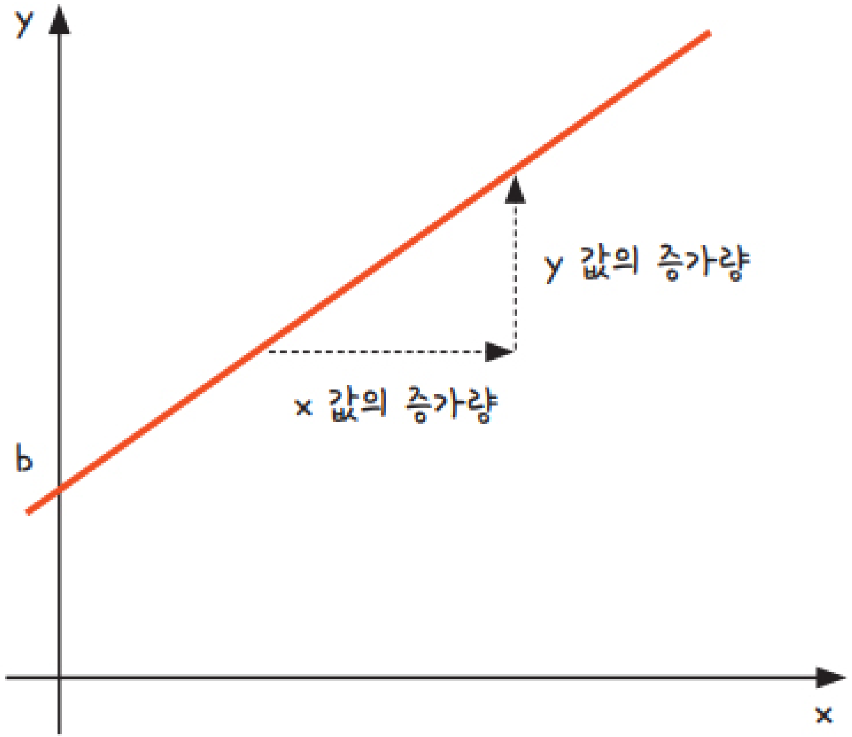
- x가 일차인 형태이며 x가 일차로 남으려면 a는 0이 아니어야 함

일차 함수, 기울기와 y 절편

- 일차 함수식 $y = ax + b$에서 a는 기울기, b는 절편이라고 함
- 기울기는 기울어진 정도를 의미하는데, 그림 3-1에서 x 값이 증가할 때 y 값이 어느 정도 증가하는지에 따라 그래프의 기울기 a가 정해짐
- 절편은 그래프가 축과 만나는 지점을 의미
- 그림 3-1에서 y축과 만나는 y 절편이 바로 b

일차 함수, 기울기와 y 절편

그림 3-1 | 일차 함수 그래프 ▶



일차 함수, 기울기와 y 절편

- 딥러닝의 수학 원리를 배울 때 초반부터 이 식이 등장
- x가 주어지고 원하는 y 값이 있을 때 적절한 a와 b를 찾는 것, 이것이 바로 딥러닝을 설명하는 가장 간단한 표현

02
이차 함수와 최솟값

이차 함수와 최솟값

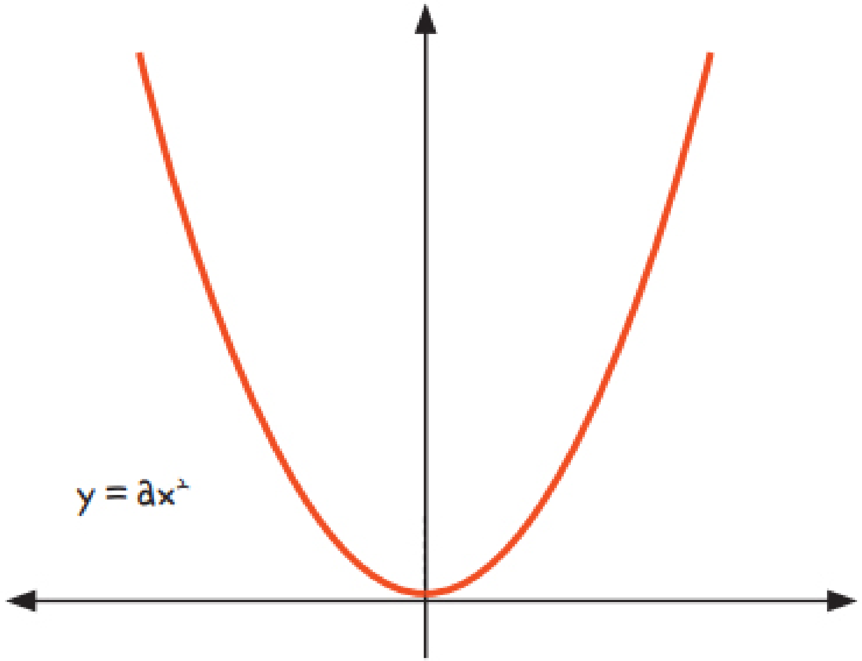
- 이차 함수란 y가 x에 관한 이차식으로 표현되는 경우를 의미
- 다음과 같은 함수식으로 표현할 수 있음

$$y = ax^2 \,(a \neq 0)$$

이차 함수와 최솟값

- 이차 함수의 그래프는 그림 3-2와 같이 포물선 모양
- a > 0이면 아래로 볼록한 그래프가 됨

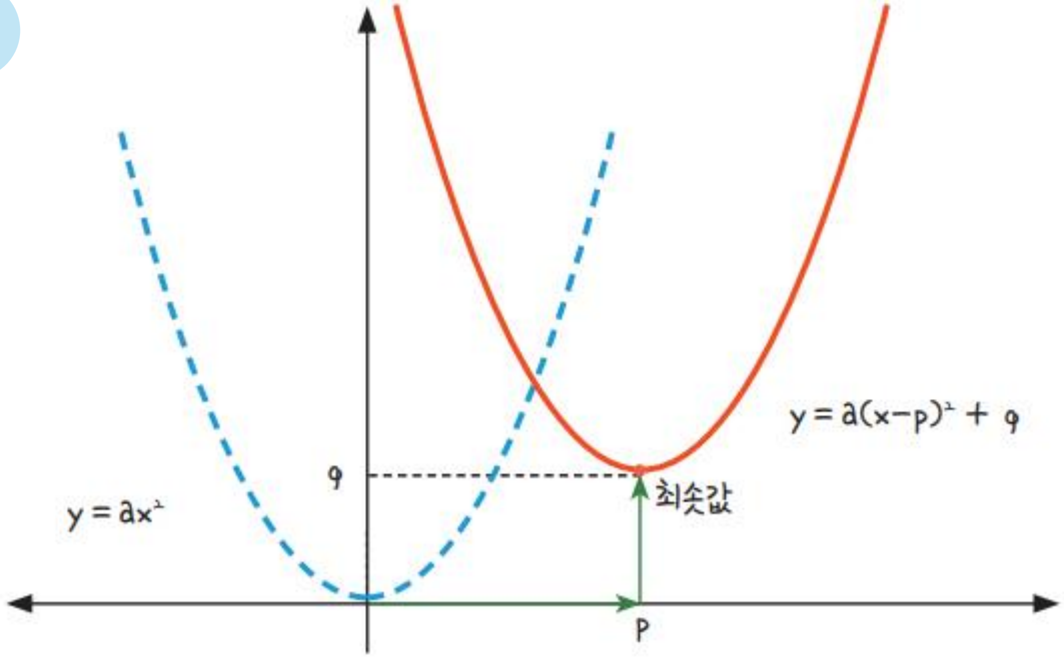
그림 3-2 | 이차 함수 그래프 ▶



이차 함수와 최솟값

- y = ax2의 그래프를 x축 방향으로 p만큼, y축 방향으로 q만큼 평행 이동시키면 그림 3-3과 같이 움직임
- 점 p와 q를 꼭짓점으로 하는 포물선이 됨
- 이때 포물선의 맨 아래에 위치한 지점이 최솟값이 되는데, 딥러닝을 실행할 때는 이 최솟값을 찾아내는 과정이 매우 중요함

그림 3-3 | 이차 함수 그래프의 평행 이동과 최솟값



이차 함수와 최솟값

- 이 최솟값은 4장에 소개할 '최소 제곱법' 공식으로 쉽게 알아낼 수 있음
- 딥러닝을 실제로 실행할 때 만나는 문제에서는 대부분 최소 제곱법을 활용할 수가 없음
- 그 이유는 최소 제곱법을 계산하기 위해 꼭 필요한 조건들을 알 수 없기 때문임
- 미분과 기울기를 이용해야 함

03

미분, 순간 변화율과 기울기

미분, 순간 변화율과 기울기

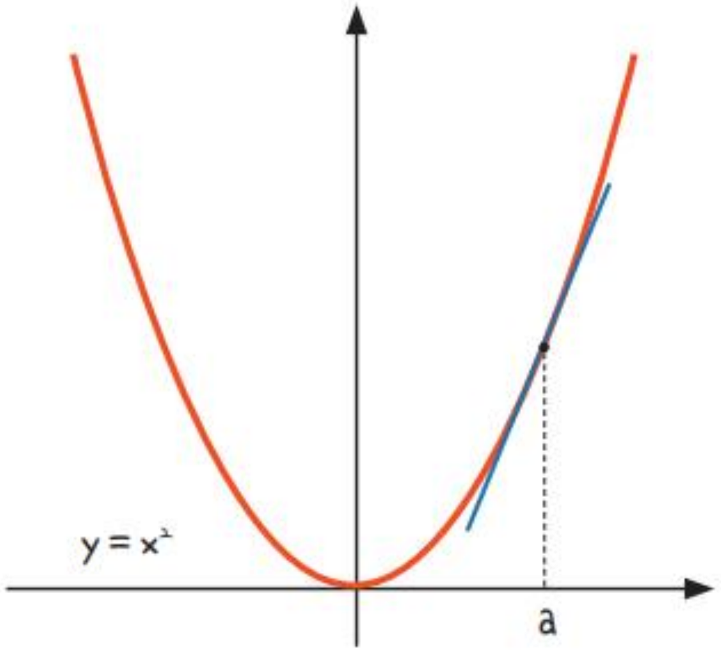
- 딥러닝을 이해하는 데 가장 중요한 수학 원리는 미분이라고 할 수 있음
- 조금 전 딥러닝은 결국 일차 함수의 a와 b 값을 구하는 것인데, a와 b 값은 이차 함수 포물선의 최솟값을 구하는 것
- 이 최솟값을 미분으로 구하기 때문에 미분이 딥러닝에서 중요한 것
- 미분과 기울기의 개념을 먼저 알아보자

미분, 순간 변화율과 기울기

미분, 순간 변화율과 기울기

- 그림 3-4와 같이 y = x2이라는 그래프가 있다고 해 보자
- x축에 있는 한 점 a에 대응하는 y의 값은 a2
- 이때 a가 오른쪽이나 왼쪽으로 조금씩 이동한다고 상상해 보자
- 이에 따라 y도 조금씩 변화할 것

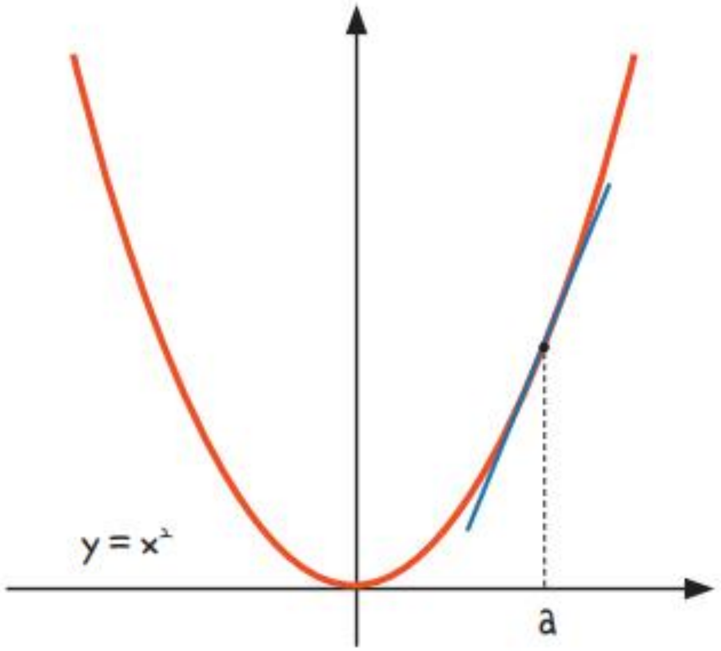
▼ 그림 3-4 | a에서의 순간 변화율은 곧 기울기다!



미분, 순간 변화율과 기울기

- 상상력을 조금 더 발휘해 이번에는 a가 미세하게 '0에 가까울 만큼' 움직였다고 하자
- y 값 역시 매우 미세하게 변화를 할 텐데, 이번에는 너무 미세해서 실제로 움직이는 것이 아니라 방향만 드러내는 정도의 순간적인 변화만 있을 것
- 이 순간의 변화를 놓고 순간 변화율이라는 이름을 붙였음
- 순간 변화율은 어느 쪽을 향하는 방향성을 지니고 있으므로, 이 방향을 따라 직선을 길게 그려 주면 그래프와 맞닿는 접선이 그려짐
- 이 선이 바로 이 점에서의 기울기가 됨

▼ 그림 3-4 | a에서의 순간 변화율은 곧 기울기다!



미분, 순간 변화율과 기울기

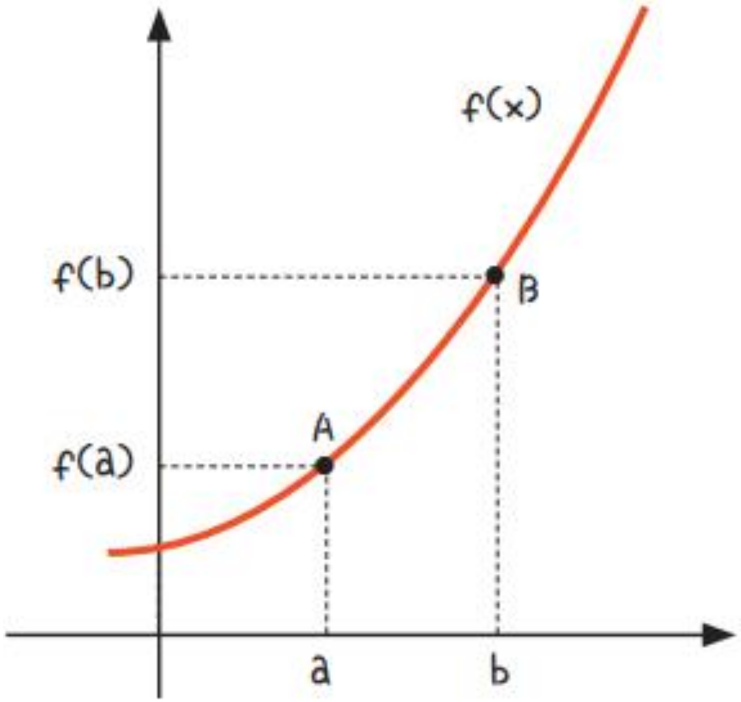
미분, 순간 변화율과 기울기

- 미분을 한다는 것은 쉽게 말해 이 '순간 변화율'을 구한다는 것
- 어느 순간에 어떤 변화가 일어나고 있는지 숫자로 나타낸 것을 미분 계수라고 하며,
  이 미분 계수는 곧 그래프에서의 기울기를 의미
- 이 기울기가 중요한 것은 기울기가 0일 때, 즉 x축과 평행한 직선으로 그어질 때가 바로
  그래프에서 최솟값인 지점이 되기 때문임

미분, 순간 변화율과 기울기

- 이제 순간 변화율을 구하는 방법을 알아보자
- 어떤 함수 f(x)가 그림 3-5와 같이 주어졌다고 하자
- 이 함수에 x축 위의 두 실수 a와 b를 대입하면 두 점 A, B는 그림과 같이 각각 A(a, f(a)), B(b, f(b))에 해당하는 곳에 표시

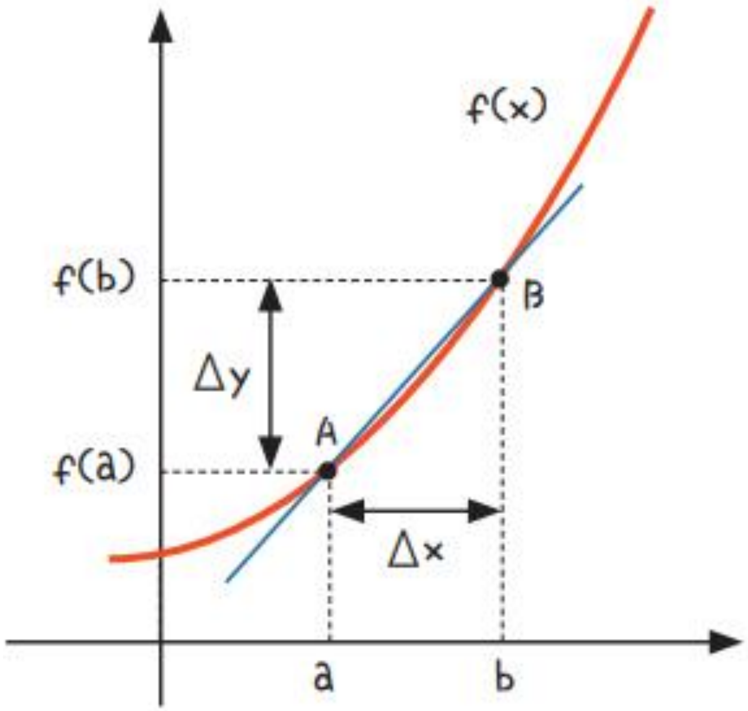
▼ 그림 3-5 | 함수 f(x)의 x축 위에 두 실수 a와 b를 대입



미분, 순간 변화율과 기울기

- 이때 두 점 A와 B를 이어 직선을 만들면 그림 3-6과 같이 두 점 A와 B를 지나는 직선의 기울기가 그려짐
- 여기서 Δ(델타)는 변화량을 나타내는 기호

▼ 그림 3-6 | A와 B를 지나는 직선은 이 두 점 간의 기울기, 곧 평균 변화율을 의미



미분, 순간 변화율과 기울기

- 이 그래프에서 x 값의 증가량은 b -a이고, y 값의 증가량은 f(b) - f(a)
- 이를 Δ를 써서 표현하면 x 값의 증가량은 Δx로, y 값의 증가량은 f(a + Δx) - f(a)로 나타낼 수 있음
- 직선의 기울기는 $\frac{y\text{ 값의 증가량}}{x\text{ 값의 증가량}}$ 이라고 했음
- A와 B를 지나는 직선의 기울기는 다음과 같이 표현할 수 있음

$$\text{직선 AB의 기울기} = \frac{y\text{ 값의 증가량}}{x\text{ 값의 증가량}} = \frac{f(b)-f(a)}{b-a} = \frac{f(a+\Delta x)-f(a)}{\Delta x}$$

미분, 순간 변화율과 기울기

- 이때 직선 AB의 기울기를 A와 B 사이의 '평균 변화율'이라고도 함
- 미분을 배우고 있는 우리에게 필요한 것은 순간 변화율
- 순간 변화율은 x의 증가량(Δx)이 0에 가까울 만큼 아주 작을 때의 순간적인 기울기를 의미하므로, 극한(limit) 기호를 사용해 다음과 같이 나타냄

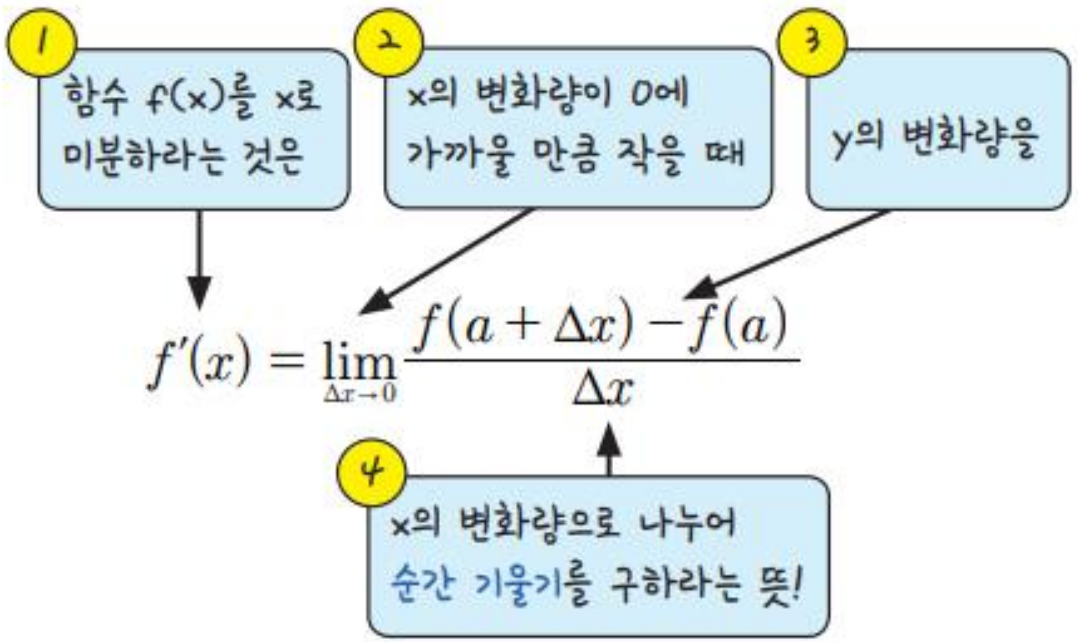
$$\lim_{\Delta x \to 0} \frac{f(a+\Delta x) - f(a)}{\Delta x}$$

미분, 순간 변화율과 기울기

- 여기서 $\lim\limits_{\Delta x \to 0}$ 는 ‘x의 증가량이 0에 가까울 만큼 작을 때’라는 뜻
- 기울기는 $\dfrac{y\text{ 값의 증가량}}{x\text{ 값의 증가량}}$ 이므로 순간 기 $\lim\limits_{\Delta x \to 0}\dfrac{y\text{ 값의 증가량}}{x\text{ 값의 증가량}}$ 으로 표현 $\lim\limits_{\Delta x \to 0}\dfrac{\Delta y}{\Delta x} = \dfrac{dy}{dx}$ 은 라고도 쓸 수 있음

미분, 순간 변화율과 기울기

- "함수 f(x)를 미분하라"는 것을 $f'(x)$ 또 $\frac{d}{dx}f(x)$ 로 표기하는데, 함수 f(x)를 미분하는 공식을 알기 쉽게 정리하면 다음과 같음



$$f'(x) = \lim_{\Delta x \to 0} \frac{f(a+\Delta x) - f(a)}{\Delta x}$$

미분, 순간 변화율과 기울기

- 다음은 딥러닝을 공부하는 과정 중에 자주 만나게 되는 중요한 다섯 가지 미분의 기본 공식

미분의 기본 공식

1 | $f(x) = x$일 때 $f'(x) = 1$

2 | $f(x) = a$에서 a가 상수일 때 $f'(x) = 0$

3 | $f(x) = ax$에서 a가 상수일 때 $f'(x) = a$

4 | $f(x) = x^a$에서 a가 자연수일 때 $f'(x) = ax^{a-1}$

5 | $f(g(x))$에서 $f(x)$와 $g(x)$가 미분 가능할 때 $\{f(g(x))\}' = f'(g(x)) \times g'(x)$

04
편미분

편미분

- 미분과 더불어 딥러닝을 공부할 때 가장 자주 접하게 되는 또 다른 수학 개념은 바로 편미분
- 미분과 편미분 모두 '미분하라'는 의미에서는 다를 바가 없음
- 여러 가지 변수가 식 안에 있을 때, 모든 변수를 미분하는 것이 아니라 우리가 원하는 한 가지 변수만 미분하고 그 외에는 모두 상수로 취급하는 것이 바로 편미분
- 예를 들어 f(x) = x와 같은 식을 미분할 때는 변수가 x 하나뿐이어서 미분하라는 의미에 혼란이 없음
- 다음 식을 보자

$$f(x, y) = x^2 + yx + a \ (a\text{는 상수})$$

편미분

- 여기에는 변수가 x와 y, 이렇게 두 개 있음
- 이 중 어떤 변수로 미분해야 하는지 정해야 하므로 편미분을 사용하는 것
- 만일 이 식처럼 여러 변수 중에서 x에 관해서만 미분하고 싶다면, 함수 f를 'x에 관해 편미분하라'고 하며 다음과 같이 식을 씀

$$\frac{\partial f}{\partial x}$$

편미분

- 앞에 나온 함수 f(x, y) = x2 + yx + a를 x에 관해 편미분하는 과정은 어떻게 될까?
- 먼저 바로 앞에서 배운 미분의 성질 4에 따라 x2항은 2x가 됨
- 미분법의 기본 공식 3에 따라 yx는 y가 됨
- 마지막 항 a는 미분의 성질 1에 따라 0이 됨
- 이를 정리하면 다음과 같이 나타낼 수 있음

$f(x, y) = x^2 + yx + a$일 때

$$\frac{\partial f}{\partial x} = 2x + y$$

05
지수와 지수 함수

지수와 지수 함수

- 지수란 다음과 같은 형태를 의미

$$a^{\square}$$

지수와 지수 함수

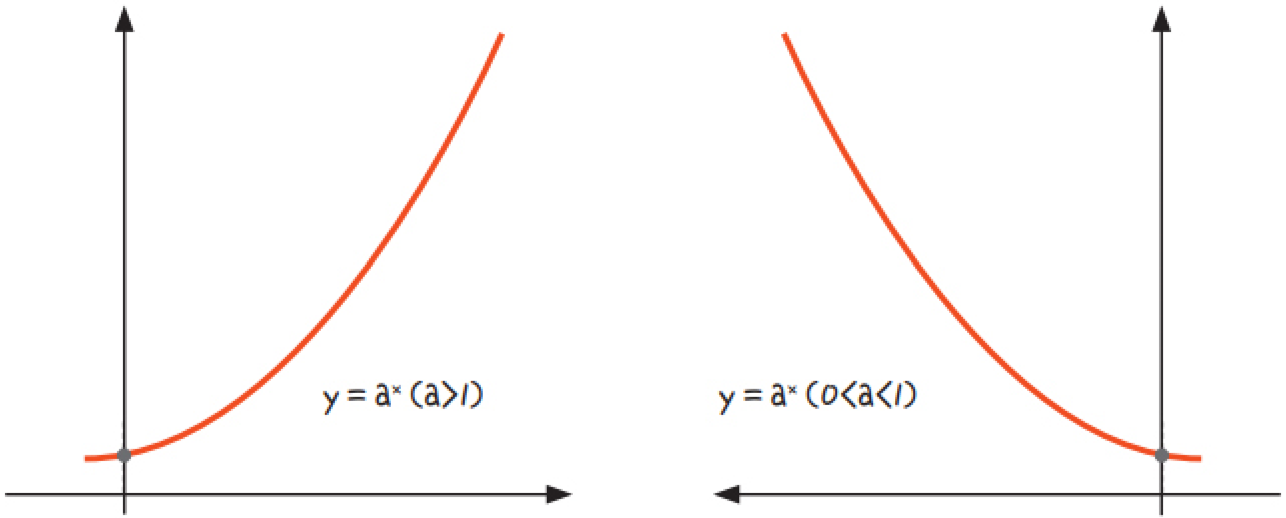
- 여기서 a를 '밑'이라 하고 를 '지수'라고 함
- a를 만큼 반복해서 곱한다는 뜻
- 지수 함수란 변수 x가 지수 자리에 있는 경우를 의미
- 식으로 나타내면 다음과 같은 형태

$$y = a^x \ (a \neq 1,\ a > 0)$$

지수와 지수 함수

- 지수 함수에서는 밑(a) 값이 무엇인지가 중요함
- 이 값이 1이면 함수가 아님
- 또 0보다 작으면 허수를 포함하게 되므로 안 됨
- 밑의 값은 a > 1이거나 0 < a < 1, 둘 중 하나가 도
- 이 두 가지 경우의 그래프는 각각 그림 3-7과 같음

▼ 그림 3-7 | a > 1일 때와 0 < a < 1일 때의 지수 함수



06
시그모이드 함수

시그모이드 함수

시그모이드 함수

- 딥러닝의 내부를 보면 입력받은 신호를 얼마나 출력할지를 계산하는 과정이 무수히 반복
- 이때 출력 값으로 얼마나 내보낼지를 계산하는 함수를 활성화 함수라고 함
- 활성화 함수는 딥러닝이 발전함에 따라 여러 가지 형태로 개발되어 왔는데, 그중 가장 먼저 배우는 중요한 함수가 바로 시그모이드 함수
- 시그모이드 함수는 지수 함수에서 밑 값이 자연 상수 e인 함수를 의미
- 자연 상수 e는 '자연 로그의 밑', '오일러의 수' 등 여러 이름으로 불리는데, 파이(π)처럼 수학에서 중요하게 사용되는 무리수이며 그 값은 대략 2.718281828...

시그모이드 함수

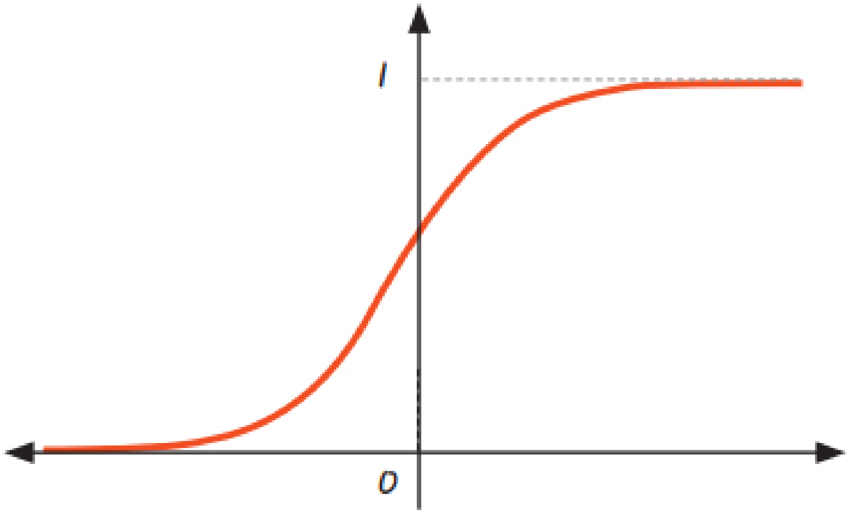
- 자연 상수 e가 지수 함수에 포함되어 분모에 들어가면 시그모이드 함수가 되는데, 이를 식으로 나타내면 다음과 같음

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

시그모이드 함수

- 시그모이드 함수를 그래프로 그려 보면 그림 3-8과 같이 S자 형태로 나타남

▼ 그림 3-8 | 시그모이드 함수의 그래프



시그모이드 함수

- x가 큰 값을 가지면 f(x)는 1에 가까워지고, x가 작은 값을 가지면 f(x)는 0에 가까워짐
- S자 형태로 그려지는 이 함수의 속성은 0 또는 1, 두 개의 값 중 하나를 고를 때 유용하게 쓰임

07
로그와 로그 함수

로그와 로그 함수

- 로그를 이해하려면 먼저 지수부터 이해해야 함
- a를 x만큼 거듭제곱한 값이 b라고 할 때, 이를 식으로 나타내면 다음과 같음

$$\boxed{a^x = b}$$

로그와 로그 함수

- 이때 a와 b를 알고 있는데 x를 모른다고 해 보자
- x는 과연 어떻게 구할 수 있을까?
- 이 x를 구하기 위해 사용하는 방법이 로그
- 영어로 Logarithm이라고 하는데 앞 세 글자 log를 사용해서 표시하며,
  지수식에서 a와 b의 위치를 다음과 같이 바꾸어 쓰면 됨

$$a^x = b$$

$$\log_a b = x$$

- 로그가 지수와 이렇게 밀접한 관계가 있듯이 로그 함수 역시 지수 함수와 밀접한 관계에 있음
- 바로 역함수의 관계
- 역함수는 x와 y를 서로 바꾸어 가지는 함수

로그와 로그 함수

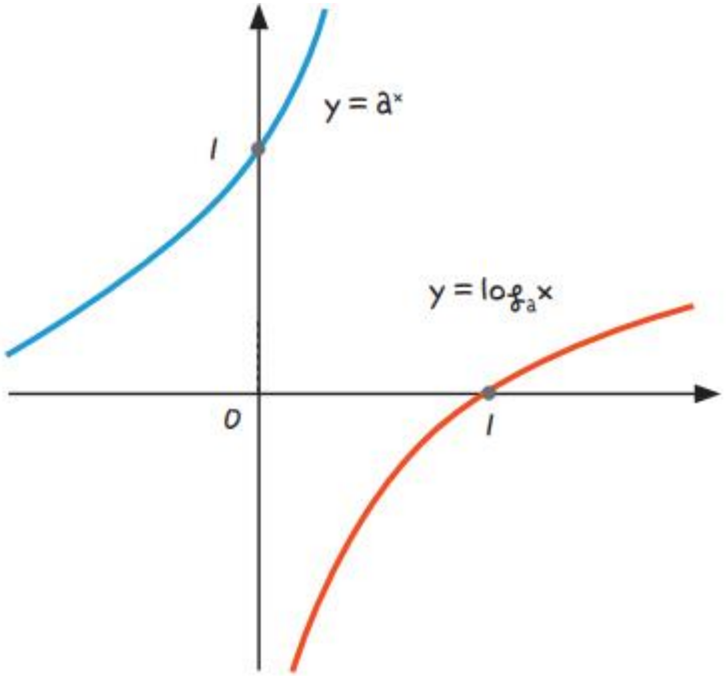
로그와 로그 함수

- 지수 함수 $y = ax(a \neq 1, a > 0)$는 로그 정의를 따라 $x = logay$로 바꿀 수 있음
- 역함수를 만들기 위해 x와 y를 서로 바꾸어 주면 됨
- 다음 식이 바로 로그 함수의 형태

로그와 로그 함수

- 역함수의 그래프는 y = x에 대해 대칭인 선으로 나타남
- 그림 3-9는 지수 함수 y = ax의 그래프를 y = x에 대칭으로 이동시킨 로그 함수 y = logax의 그래프를 보여 줌

▼ 그림 3-9 | 지수 함수 y = ax와 로그 함수 y = logax



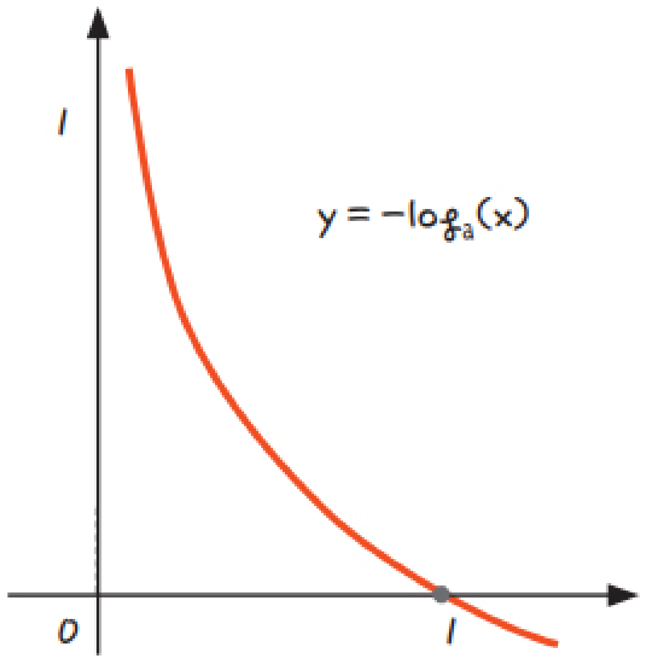
로그와 로그 함수

- 6장에서 로지스틱 회귀를 배울 때, 우리는 x가 1에 가까워지거나 0에 가까워질수록 오차가 커지는 그래프가 필요함
- 이러한 그래프를 만들기 위해 y =logax를 x축 또는 y축으로 대칭 이동하거나 알맞게 평행 이동하면 다음과 같음

로그와 로그 함수

1. x축에 대해 대칭 이동

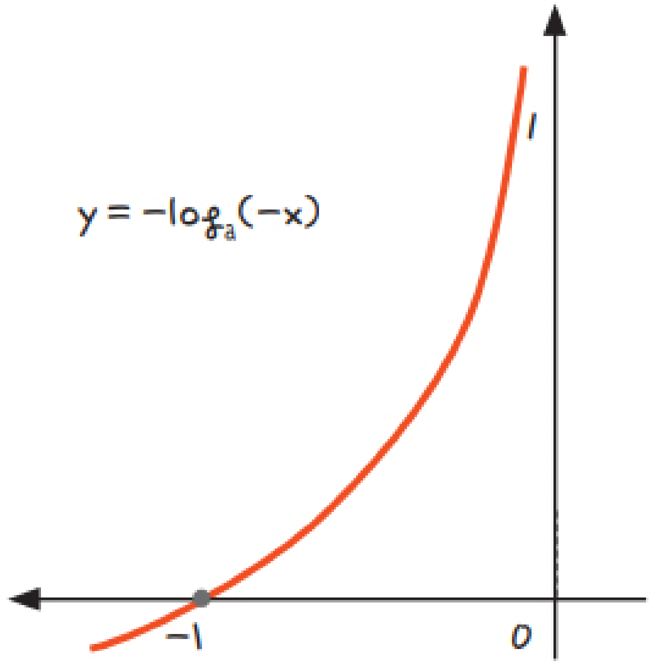
그림 3-10 | y = -loga (x) 그래프 ▶



로그와 로그 함수

2. x축과 y축에 대해 대칭 이동

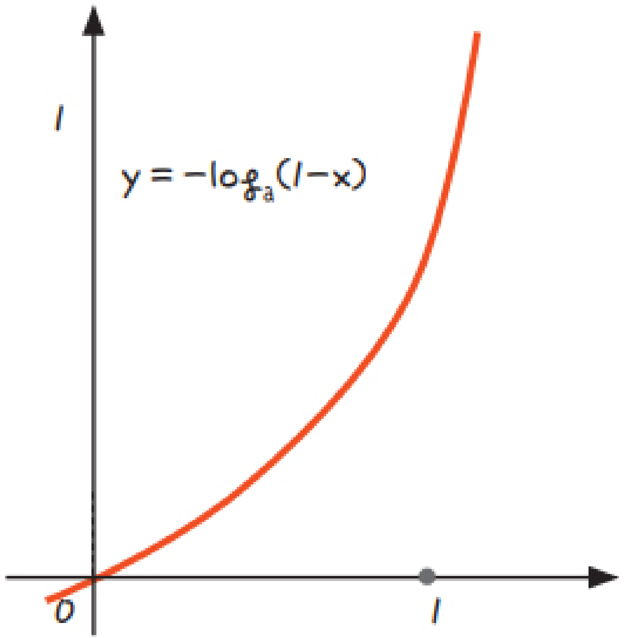
그림 3-11 | y = -loga (-x) 그래프 ▶



로그와 로그 함수

3. 2의 그래프를 x축 오른쪽 방향으로 1만큼 평행 이동

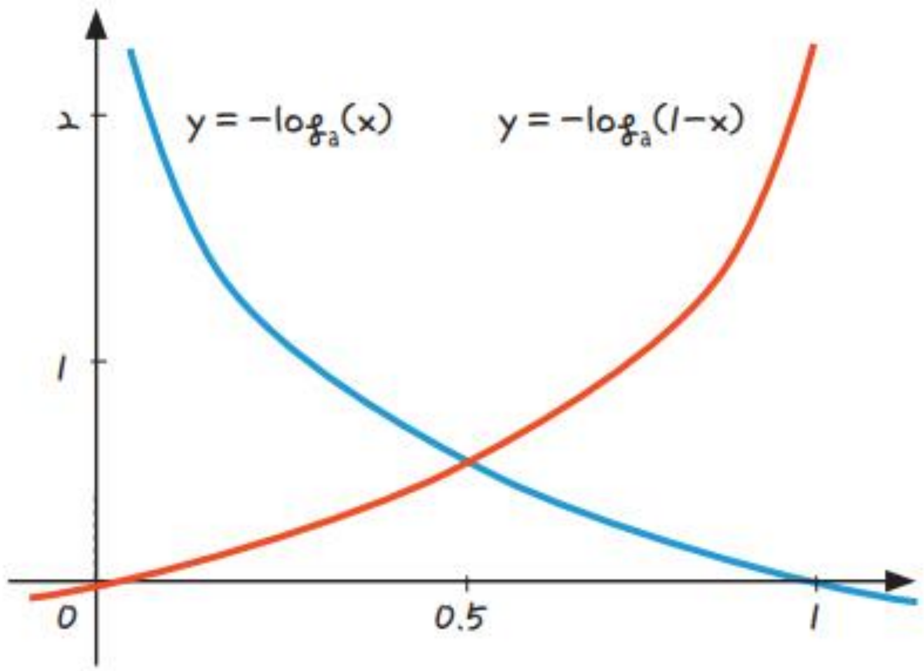
그림 3-12 | y = -loga (1-x) 그래프 ▶



로그와 로그 함수

4. 1과 3을 함께 나타낸 그래프

그림 3-13 | y = -loga (x)와 y = -loga (1-x) 그래프 ▶



로그와 로그 함수

- 지금까지 설명한 일차 함수, 이차 함수, 미분, 편미분, 지수 함수, 시그모이드 함수 그리고 로그 함수 이렇게 일곱 가지를 알고 있으면 4~22장 내용을 모두 이해할 수 있음
- 여기에 합을 표현하기 위해 만들어 $\sum$ (시그마) 기호가 종종 나옴
- $\sum_{i=1}^{n} F(i)$ 라고 하면 i를 1부터 n까지 F(i)에 대입해 더하라는 뜻
- 즉, $F(1)+F(2)+F(3)+\cdots+F(n)$ 이 됨
- 나머지 어려운 증명이나 체인 룰이 등장하는 수식의 계산은 딥러닝 활용 편을 모두 마치고 나서 이어지는 심화 학습 편에서 다룸
- 심화 학습 편을 공부하지 않아도 이 책에 나오는 모든 딥러닝 예제를 이해하고 실행하는 데는 문제없음

ROKEY BOOT CAMP

AI(Computer Vision)개론

Chapter 4. 가장 훌륭한 예측선

OOO 강사

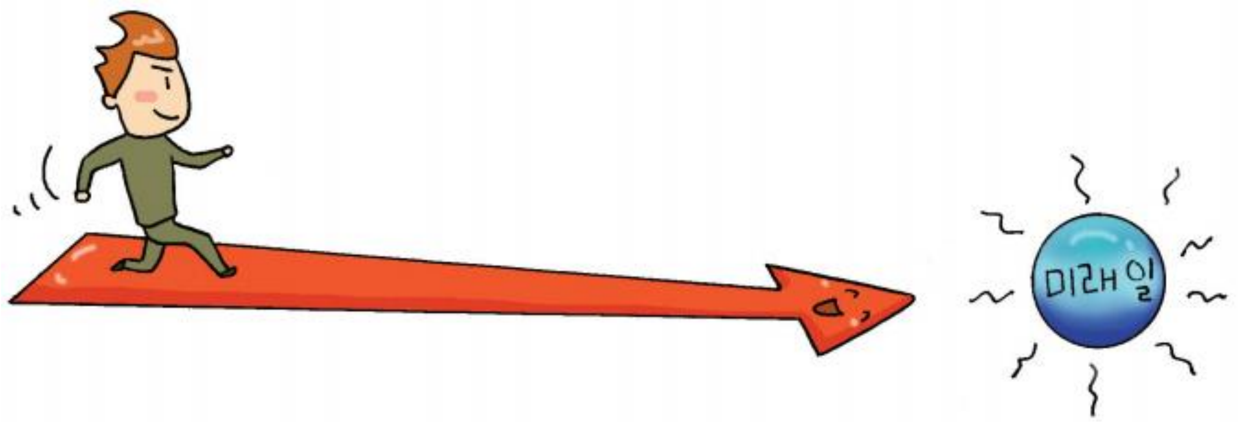
학습 목차

가장 훌륭한 예측선

- 딥러닝은 자그마한 통계의 결과들이 무수히 얽히고설켜 이루어지는 복잡한 연산의 결정체
- 우리 몸을 이해하려면 몸을 구성하는 기본 단위인 세포의 역할을 알아야 하듯, 딥러닝을 이해하려면 딥러닝의 가장 말단에서 이루어지는 기본적인 두 가지 계산 원리를 알아야 함
- 바로 선형 회귀와 로지스틱 회귀

가장 훌륭한 예측선

- 이 두 가지 개념을 중·고등학교 수준에서 공부하기란 쉽지 않음
- 대학에서 통계를 전공하지 않았다면 익숙하지 않을 주제
- 그러다 보니 여기서부터 시작하는 머신 러닝이 쉽지 않아 보이는 것이 무리는 아님
- 이 두 개념을 이해하기 위해 반드시 어려운 공식이나 수학, 통계학 개념에 통달해야 하는 것은 아님
- 어렵지 않은 수학 용어와 중·고등학교 수준으로도 딥러닝의 밑그림이 되는 개념을 충분히 이해할 수 있음
- 이를 알고 나면 딥러닝을 구동시키는 원리에 한 걸음 다가설 수 있음



가장 훌륭한 예측선

- 이 장의 제목인 '가장 훌륭한 예측선'이라는 표현은 '선형 회귀(linear regression) 분석을 이용한 모델'의 의미를 쉽게 풀어서 표현한 것
- 머신 러닝은 제대로 된 선을 긋는 작업부터 시작
- 선의 방향을 잘 정하면 그 선을 따라가는 것만으로도 지금은 보이지 않는 미래의 것을 예측할 수 있기 때문임
- 첫 단추가 많은 것을 결정
- 진입 장벽을 허물고 딥러닝의 세계로 들어오기 바람

01
선형 회귀의 정의

선형 회귀의 정의

"학생들의 중간고사 성적이 다 다르다."

네, 다르겠죠.

그런데 위 문장이 나타낼 수 있는 정보는 너무 제한적입니다. 학급의 학생마다 제각각 성적이 다르다는 당연한 사실 외에는 알 수 있는 것이 없습니다. 이번에는 다음 문장을 보겠습니다.

"학생들의 중간고사 성적이 [ ]에 따라 다 다르다."

선형 회귀의 정의

- 이 문장은 정보가 담길 여지를 열어 놓고 있음
- [ ] 부분에 시험 성적을 좌우할 만한 여러 가지 것이 들어간다면 좀 더 많은 사실을 전달할 수 있음
- 예를 들어 공부한 시간, 시험 당일의 컨디션, 사교육비 지출액 등이 들어갈 수 있음
- 무엇이 들어가든지 해당 성적의 이유를 나름대로 타당하게 설명할 수 있음
- 앞의 문장보다는 이 문장이 중간고사 성적의 차이와 이유를 나타낼 때 더욱 효과적

선형 회귀의 정의

- 여기서 [ ]에 들어갈 내용을 '정보'라고 함
- 머신 러닝과 딥러닝은 이 정보가 필요함
- 정보를 정확히 준비해 놓기만 하면 성적을 예측하는 방정식을 만들 수도 있음
- 이 단순한 정의를 이번에는 좀 더 수학적인 언어로 표현해 보자
- 성적을 변하게 하는 '정보' 요소를 x라고 하고, 이 x 값에 따라 변하는 '성적'을 y라고 하자
- 이를 정의하면 'x 값이 변함에 따라 y 값도 변한다'가 됨
- 이 정의 안에서 독립적으로 변할 수 있는 값 x를 독립 변수라고 함
- 또한, 이 독립 변수에 따라 종속적으로 변하는 y를 종속 변수라고 함

선형 회귀의 정의

- 독립 변수가 x 하나뿐이어서 이것만으로 정확히 설명할 수 없을 때는 x1, x2, x3 등 x 값을 여러 개 준비해 놓을 수도 있음
- 하나의 x 값만으로도 y 값을 설명할 수 있다면 단순 선형 회귀(simple linear regression)라고 함
- 또한, x 값이 여러 개 필요하다면 다중 선형 회귀(multiple linear regression)라고 함

02

가장 훌륭한 예측선이란?

가장 훌륭한 예측선이란?

- 우선 독립 변수가 하나뿐인 단순 선형 회귀의 예를 공부해 보자
- 성적을 결정하는 여러 요소 중에 '공부한 시간' 한 가지만 놓고 생각해 보자
- 중간고사를 본 4명의 학생에게 각각 공부한 시간을 물어보고 이들의 중간고사 성적을 표 4-1과 같이 정리했다고 하자

▼ 표 1-1 | 딥러닝 프로그래밍 툴의 장단점

공부한 시간	2시간	4시간	6시간	8시간
성적	81점	93점	91점	97점

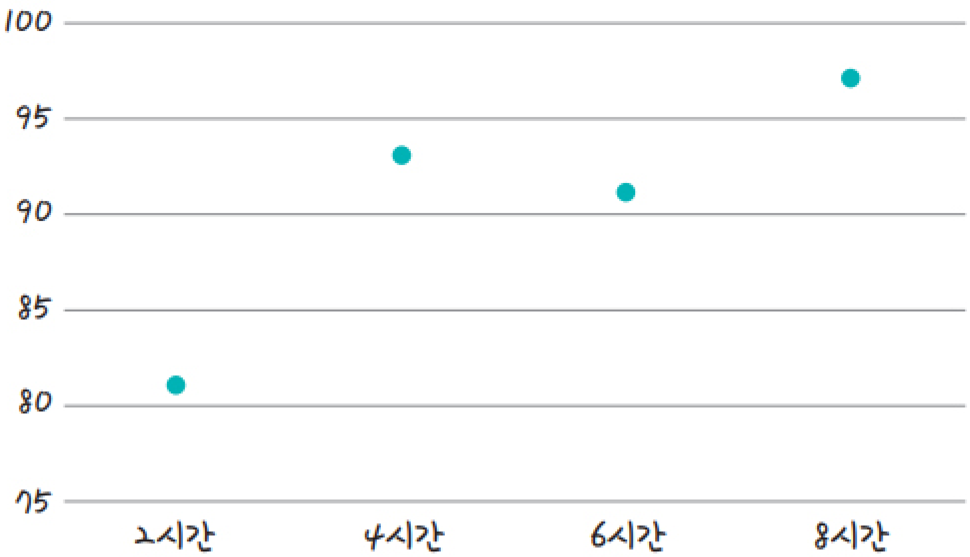
가장 훌륭한 예측선이란?

가장 훌륭한 예측선이란?

- 여기서 공부한 시간을 x라고 하고 성적을 y라고 할 때 집합 x와 집합 y를 다음과 같이 표현할 수 있음

$$X = \{2,\ 4,\ 6,\ 8\}$$
$$Y = \{81,\ 93,\ 91,\ 97\}$$

▼ 그림 4-1 | 공부한 시간과 성적을 좌표로 표현



가장 훌륭한 예측선이란?

- 좌표 평면에 나타내 놓고 보니, 왼쪽이 아래로 향하고 오른쪽이 위를 향하는 일종의 '선형(선으로 표시될 만한 형태)'을 보임
- 선형 회귀를 공부하는 과정은 이 점들의 특징을 가장 잘 나타내는 선을 그리는 과정과 일치

가장 훌륭한 예측선이란?

- 이 데이터에서 주어진 점들의 특징을 담은 선은 직선이므로 곧 일차 함수 그래프
- 일차 함수 그래프는 다음과 같은 식으로 표현할 수 있음

$$y = ax + b$$

가장 훌륭한 예측선이란?

- 여기서 x 값은 독립 변수이고 y 값은 종속 변수
- 즉, x 값에 따라 y 값은 반드시 달라짐
- 다만, 정확하게 계산하려면 상수 a와 b의 값을 알아야 함
- 이 직선을 훌륭하게 그으려면 직선의 기울기 a 값과 y 절편 b 값을 정확히 예측해 내야 함
- 앞서 선형 회귀는 곧 정확한 선을 그려 내는 과정이라고 했음
- 지금 주어진 데이터에서의 선형 회귀는 결국 최적의 a 값과 b 값을 찾아내는 작업이라고 할 수 있음

가장 훌륭한 예측선이란?

- 선을 잘 긋는 것이 어째서 중요할까?
- 잘 그어진 선을 통해 우리는 표 4-1의 공부한 시간과 중간고사 성적 데이터에 들어 있지 않은 여러 가지 내용을 유추할 수 있기 때문임
- 예를 들어 표 4-1에 나와 있지 않은 또 다른 학생의 성적을 예측하고 싶다고 하자
- 이때 정확한 직선을 그어 놓았다면 이 학생이 몇 시간을 공부했는지만 물어보면 됨
- 정확한 a값과 b 값을 따라 움직이는 직선에 학생이 공부한 시간인 x 값을 대입하면 예측 성적인 y 값을 구할 수 있는 것

가장 훌륭한 예측선이란?

- 딥러닝을 포함한 머신 러닝의 예측은 결국 이러한 기본 접근 방식과 크게 다르지 않음
- 기존 데이터(정보)를 가지고 어떤 선이 그려질지 예측한 후, 아직 답이 나오지 않은 그 무언가를 그 선에 대입해 보는 것
- 선형 회귀의 개념을 이해하는 것은 딥러닝을 이해하는 데 중요한 첫걸음

03
최소 제곱법

최소 제곱법

- 이제 우리 목표는 가장 정확한 선을 긋는 것
- 더 구체적으로는 정확한 기울기 a와 정확한 y 절편 b를 알아내면 된다고 했음
- 만일 우리가 최소 제곱법(method of least squares)이라는 공식을 알고 적용한다면, 이를 통해 일차 함수의 기울기 a와 y 절편 b를 바로 구할 수 있음

최소 제곱법

- 지금 가진 정보가 x 값(입력 값, 여기서는 ‘공부한 시간’)과 y 값(출력 값, 여기서는 ‘성적’)일 때 이를 이용해 기울기 a를 구하는 방법은 다음과 같음

$$a = \frac{(x - x\ \text{평균})(y - y\ \text{평균})\text{의 합}}{(x - x\ \text{평균})^2\text{의 합}} \qquad \text{(식 4.1)}$$

최소 제곱법

- 이것이 바로 최소 제곱법 공식
- 쉽게 풀어서 다시 쓰면 x의 편차(각 값과 평균과의 차이)를 제곱해서 합한 값을 분모로 놓고, x와 y의 편차를 곱해서 합한 값을 분자로 놓으면 기울기가 나온다는 의미
- 실제로 우리가 가진 y(성적) 값과 x(공부한 시간) 값을 이 식에 대입해 보자
- 먼저 x 값의 평균과 y 값의 평균을 구해 보면 다음과 같음

공부한 시간(x) 평균: (2 + 4 + 6 + 8) ÷ 4 = 5

성적(y) 평균: (81+ 93 + 91 + 97) ÷ 4 = 90.5

최소 제곱법

- 이를 식 4.1에 대입하면 다음과 같음

$$
\begin{aligned}
a &= \frac{(2-5)(81-90.5)+(4-5)(93-90.5)+(6-5)(91-90.5)+(8-5)(97-90.5)}{(2-5)^2+(4-5)^2+(6-5)^2+(8-5)^2} \\
&= \frac{46}{20} \\
&= 2.3
\end{aligned}
$$

최소 제곱법

- 기울기 a는 2.3이 나옴!
- 다음은 y 절편인 b를 구하는 공식

$$b = y\text{의 평균} - (x\text{의 평균} \times \text{기울기 } a) \qquad \text{(식 4.2)}$$

- 즉, y의 평균에서 x의 평균과 기울기의 곱을 빼면 b 값이 나온다는 의미

최소 제곱법

- 우리는 이미 y 평균, x 평균, 그리고 조금 전 구한 기울기 x까지 이 식을 풀기 위해 필요한 모든 변수를 알고 있음
- 이를 식에 대입해 보자

$$
\begin{aligned}
b &= 90.5 - (2.3 \times 5) \\
&= 79
\end{aligned}
$$

최소 제곱법

- y 절편 b는 79가 나왔음
- 이제 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성

$$y = 2.3x + 79$$

최소 제곱법

- 이 식에 우리가 가진 데이터를 대입해 보자
- x를 대입했을 때 나오는 y 값을 '예측 값'이라고 하겠음

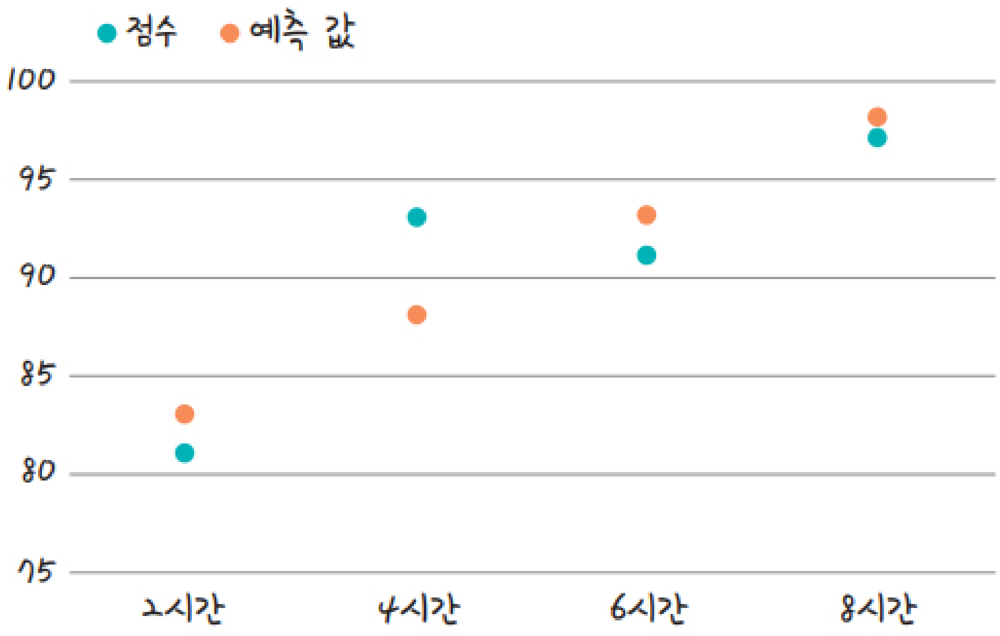
▼ 표 4-2 | 최소 제곱법 공식으로 구한 성적 예측 값

공부한 시간	2	4	6	8
성적	81	93	91	97
예측 값	**83.6**	**88.2**	**92.8**	**97.4**

최소 제곱법

- 좌표 평면에 이 예측 값을 찍어 보자

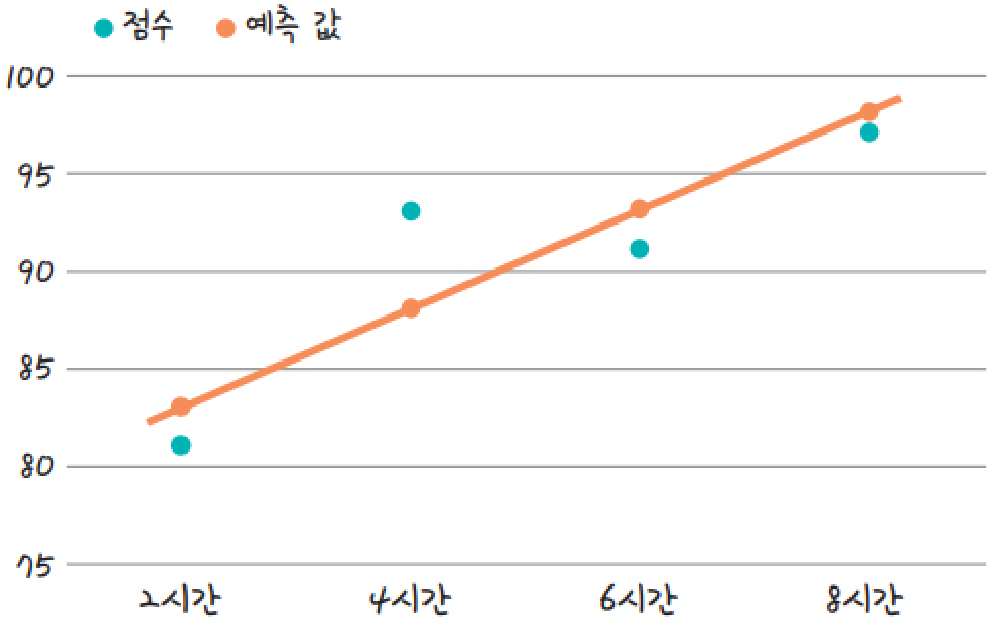
▼ 그림 4-2 | 공부한 시간, 성적, 예측 값을 좌표로 표현



최소 제곱법

- 예측한 점들을 연결해 직선을 그으면 그림 4-3과 같음

▼ 그림 4-3 | 오차가 최저가 되는 직선의 완성



최소 제곱법

- 이것이 바로 오차가 가장 적은 주어진 좌표의 특성을 가장 잘 나타내는 직선
- 우리가 원하는 예측 직선
- 이 직선에 우리는 다른 x 값(공부한 시간)을 집어넣어서 '공부량에 따른 성적을 예측'할 수 있음

04

파이썬 코딩으로 확인하는 최소 제곱

☑ 파이썬 코딩으로 확인하는 최소 제곱

- 우리가 이론을 배우는 목적은 딥러닝을 구현하기 위해서임
- 이 책에서 설명하는 모든 이론을 자유롭게 코드로 변환할 수 있어야 진정한 의미가 있음
- 지금까지 공부한 내용을 코딩으로 구현해 보자

파이썬 코딩으로 확인하는 최소 제곱

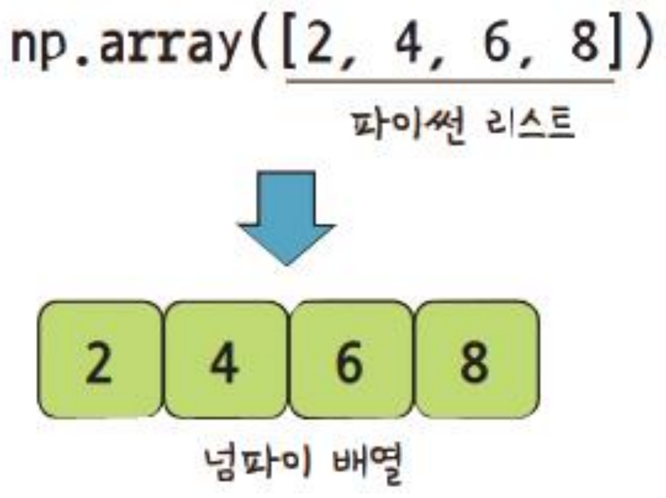
- 먼저 넘파이 라이브러리를 불러옴
- 넘파이는 파이썬에서 수학 연산과 분석을 하게 도와주는 라이브러리
- 공부한 시간을 리스트로 만들어 x라는 이름의 넘파이 배열로 저장
- 또 그때의 점수를 y라는 이름의 넘파이 배열로 저장

```
# 공부한 시간과 점수를 각각 x, y라는 이름의 넘파이 배열로 만듭니다.
x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])
```

파이썬 코딩으로 확인하는 최소 제곱

- 파이썬에서 리스트는 대괄호([ ])로 감싼 요소들을 쉼표(,)로 구분해 대입하여 만듦
- np.array() 함수를 사용하면 파이썬 리스트를 넘파이 배열로 바꾸어 여러 가지 계산을 수행할 수 있음

▼ 그림 4-4 | 파이썬 리스트의 기본 형태



파이썬 코딩으로 확인하는 최소 제곱

- 이제 최소 제곱근 공식으로 기울기 a의 값과 y 절편 b의 값을 구해 보자
- x의 모든 원소 평균을 구하는 넘파이 함수는 mean()
- mx 변수에는 x 원소들의 평균값을, my 변수에는 y 원소들의 평균값을 넣음

```
mx = np.mean(x)
my = np.mean(y)
```

파이썬 코딩으로 확인하는 최소 제곱

- 이제 앞서 살펴본 최소 제곱근 공식 중 분모 값, 즉 'x의 각 원소와 x의 평균값들의 차를 제곱하라'는 파이썬 명령을 만들 차례
- 다음과 같이 divisor라는 변수를 만들어 구현할 수 있음

```
divisor = sum([(i - mx)**2 for i in x])
```

sum()은 Σ에 해당하는 함수입니다.

x의 각 원소를 한 번씩 i 자리에 대입하라는 의미입니다.

제곱을 구하라는 의미입니다.

파이썬 코딩으로 확인하는 최소 제곱

- 이제 분자에 해당하는 부분을 구하겠음
- x와 y의 편차를 곱해서 합한 값을 구하면 됨
- 다음과 같이 새로운 함수를 정의해서 dividend 변수에 분자 값을 저장

```
def top(x, mx, y, my):
    d = 0
    for i in range(len(x)):
        d += (x[i] - mx) * (y[i] - my)
    return d
dividend = top(x, mx, y, my)
```

파이썬 코딩으로 확인하는 최소 제곱

- 임의의 변수 d의 초깃값을 0으로 설정한 후 x의 개수만큼 실행
- d에 x의 각 원소와 평균의 차, y의 각 원소와 평균의 차를 곱해서 차례로 더하는 최소 제곱법을 그대로 구현

파이썬 코딩으로 확인하는 최소 제곱

- def는 함수를 만들 때 사용하는 예약어
- 여기서는 top() 함수를 새롭게 만들었고, 그 안에 최소 제곱법의 분자식을 그대로 가져와 구현
- len(리스트)은 리스트 안에 들어 있는 원소 개수를 알려 줌
- x 리스트의 원소가 네 개이므로 len(x)는 4가 됨
- range()는 0부터 괄호 안의 숫자 바로 전까지 연속적인 숫자 객체를 만들어 줌
- 즉, range(4)는 0, 1, 2, 3의 숫자를 생성하게 됨

파이썬 코딩으로 확인하는 최소 제곱

- 이제 앞에서 구한 분모와 분자를 계산해 기울기 a를 구하겠음

```
a = dividend / divisor
```

- a를 구하고 나면 y 절편을 구하는 공식을 이용해 b를 구할 수 있음

```
b = my - (mx*a)
```

파이썬 코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같음

실습 파이썬 코딩으로 구하는 최소 제곱

```
import numpy as np

# 공부한 시간과 점수를 각각 x, y라는 이름의 넘파이 배열로 만듭니다.
x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])

# x의 평균값을 구합니다.
mx = np.mean(x)
```

파이썬 코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같음

실습 파이썬 코딩으로 구하는 최소 제곱

```
# y의 평균값을 구합니다.
my = np.mean(y)

# 출력으로 확인합니다.
print("x의 평균값: ", mx)
print("y의 평균값: ", my)

# 기울기 공식의 분모 부분입니다.
divisor = sum([(i - mx)**2 for i in x])
```

파이썬 코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같음

실습 파이썬 코딩으로 구하는 최소 제곱

```python
# 기울기 공식의 분자 부분입니다.
def top(x, mx, y, my):
    d = 0
    for i in range(len(x)):
        d += (x[i] - mx) * (y[i] - my)
    return d
dividend = top(x, mx, y, my)

# 출력으로 확인합니다.
print("분모: ", divisor)
print("분자: ", dividend)
```

파이썬 코딩으로 확인하는 최소 제곱

- 이를 하나의 파일로 정리해 보면 다음과 같음

실습 파이썬 코딩으로 구하는 최소 제곱

```
# 기울기 a를 구하는 공식입니다.
a = dividend / divisor

# y 절편 b를 구하는 공식입니다.
b = my - (mx*a)

# 출력으로 확인합니다.
print("기울기 a = ", a)
print("y 절편 b = ", b)
```

파이썬 코딩으로 확인하는 최소 제곱

실행 결과

```
x의 평균값: 5.0
y의 평균값: 90.5

분모: 20.0
분자: 46.0

기울기 a = 2.3
y 절편 b = 79.0
```

- 파이썬으로 최소 제곱법을 구현해 기울기 a의 값과 y 절편 b의 값이 각각 2.3과 79임을 구할 수 있었음

05
평균 제곱 오차

평균 제곱 오차

평균 제곱 오차

- 최소 제곱법을 이용해 기울기 a와 y 절편을 편리하게 구했지만, 이 공식만으로 앞으로 만나게 될 모든 상황을 해결하기는 어려움
- 여러 개의 입력을 처리하기에는 무리가 있기 때문임
- 예를 들어 앞서 살펴본 예에서는 변수가 '공부한 시간' 하나뿐이지만, 2장에서 살펴본 폐암 수술 환자의 생존율 데이터를 보면 입력 데이터의 종류가 17개나 됨
- 딥러닝은 대부분 입력 값이 여러 개인 상황에서 이를 해결하기 위해 실행되기 때문에 기울기 a와 y 절편 b를 찾아내는 다른 방법이 필요함

평균 제곱 오차

평균 제곱 오차

- 가장 많이 사용하는 방법은 '일단 그리고 조금씩 수정해 나가기' 방식
- 가설을 하나 세운 후 이 값이 주어진 요건을 충족하는지 판단해서 조금씩 변화를 주고, 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법
- 이는 딥러닝을 가능하게 하는 가장 중요한 원리 중 하나

평균 제곱 오차

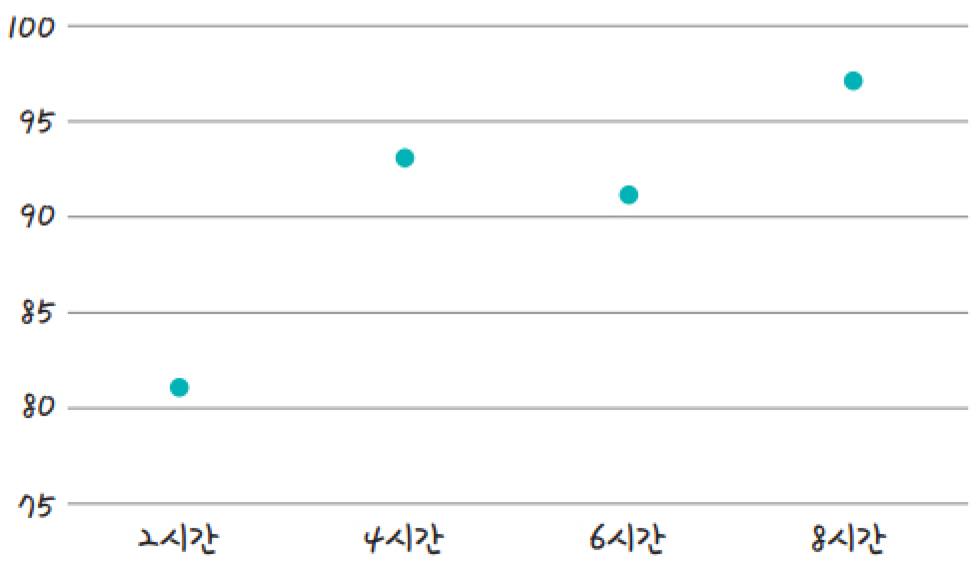
- 선을 긋고 나서 수정하는 과정에서 빠지면 안 되는 것이 있음
- 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하는 방법
- 즉, 각 선의 오차를 계산할 수 있어야 하고, 오차가 작은 쪽으로 바꾸는 알고리즘이 필요한 것

평균 제곱 오차

- 이를 위해 주어진 선의 오차를 평가하는 방법이 필요함
- 오차를 구할 때 가장 많이 사용되는 방법이 평균 제곱 오차(Mean Square Error, MSE)
- 지금부터 평균 제곱 오차를 구하는 방법을 알아보자
- 앞서 나온 공부한 시간과 성적의 관계도를 다시 한 번 볼까?

평균 제곱 오차

▼ 그림 4-5 | 공부한 시간과 성적의 관계도



평균 제곱 오차

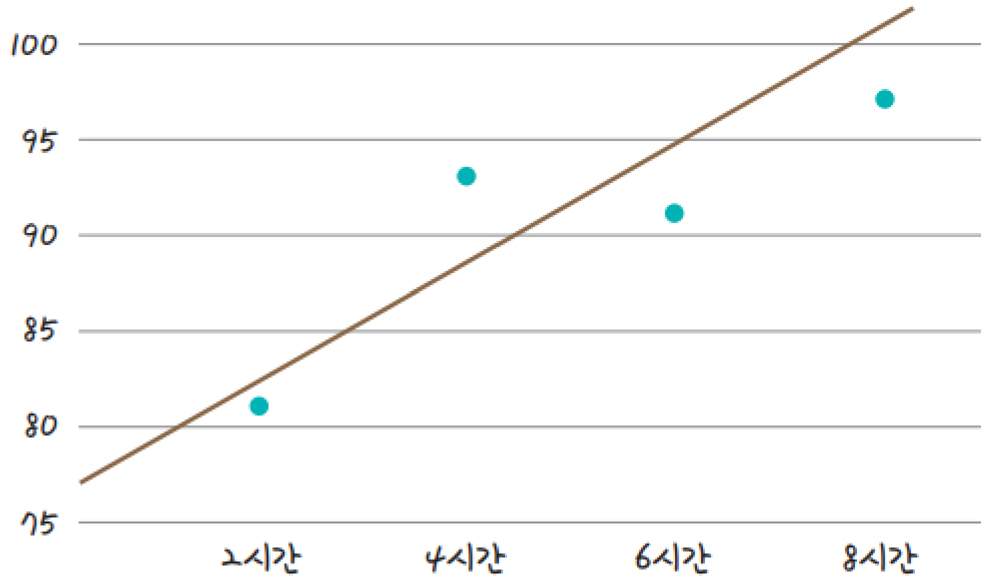
- 우리는 조금 전 최소 제곱법을 이용해 점들의 특성을 가장 잘 나타내는 최적의 직선이 y = 2.3x + 79임을 구했지만, 이번에는 최소 제곱법을 사용하지 않고 아무 값이나 a와 b에 대입해 보자
- 임의의 값을 대입한 후 오차를 구하고 이 오차를 최소화하는 방식을 사용해서 최종 a 값과 b 값을 구해 보자

평균 제곱 오차

평균 제곱 오차

- 먼저 대강 선을 그어 보기 위해 기울기 a와 y 절편 b를 임의의 수 3과 76이라고 가정해 보자
- Y = 3x + 76인 선을 그려 보면 그림 4-6과 같음

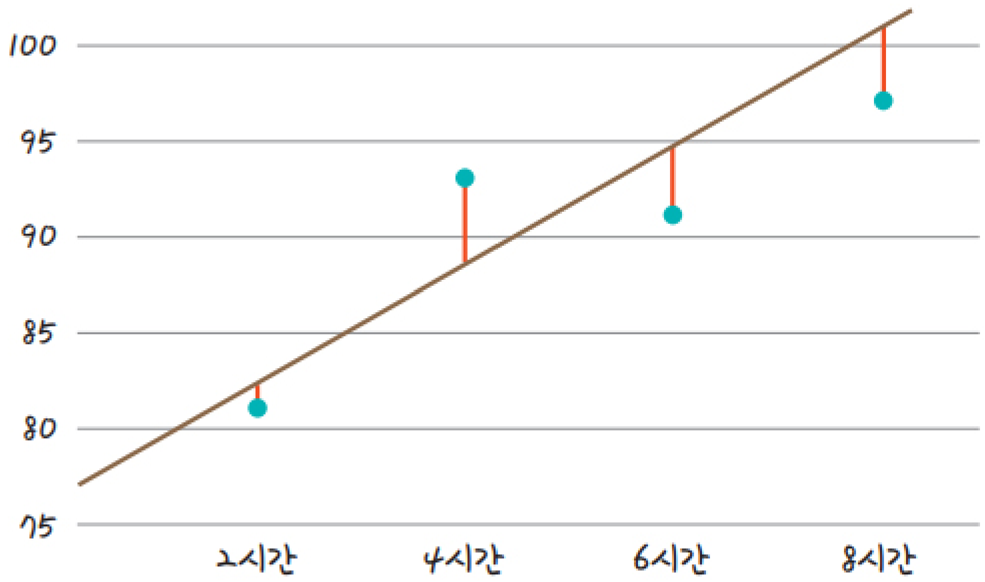
그림 4-6 | 임의의 직선 그려 보기



평균 제곱 오차

- 그림 4-6과 같은 임의의 직선이 어느 정도의 오차가 있는지 확인하려면 각 점과 그래프 사이의 거리를 재면 됨

그림 4-7 | 임의의 직선과 실제 값 사이의 거리 ▶

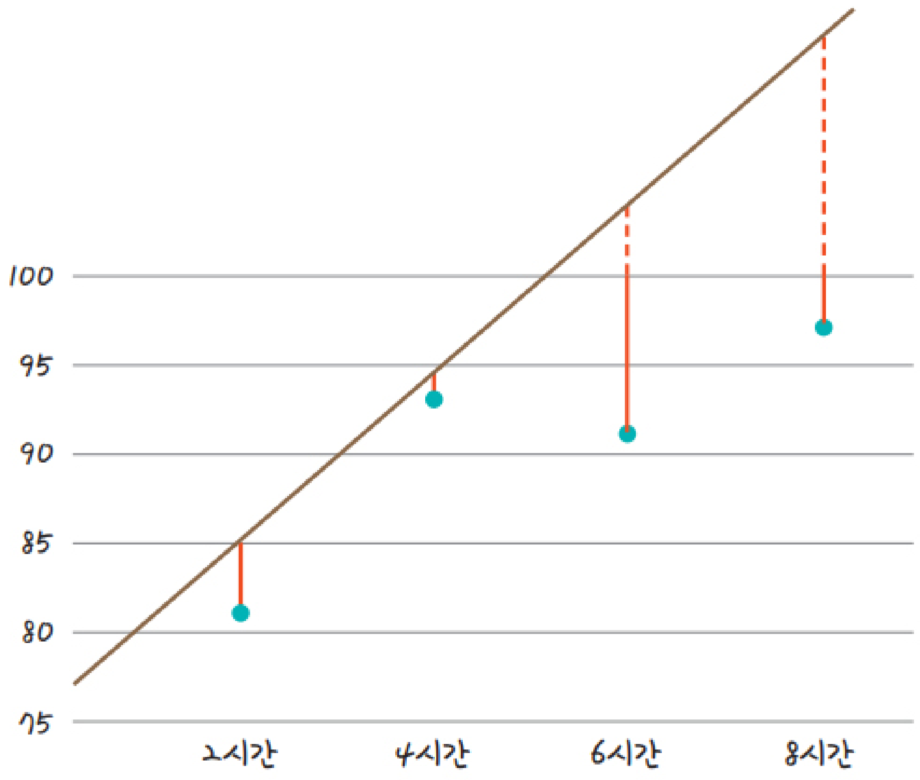


평균 제곱 오차

- 그림 4-7에서 볼 수 있는 빨간색 선은 직선이 잘 그어졌는지 나타냄
- 이 직선들의 합이 작을수록 잘 그어진 직선이고, 이 직선들의 합이 클수록 잘못 그어진 직선이 됨
- 예를 들어 기울기 값을 각각 다르게 설정한 그림 4-8과 그림 4-9의 그래프를 볼까?

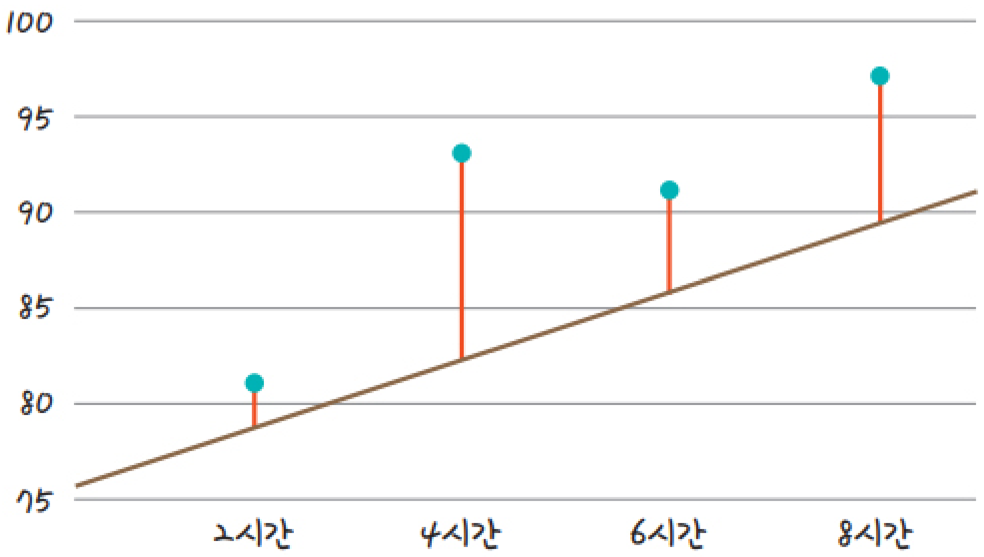
평균 제곱 오차

▼ 그림 4-8 | 기울기를 너무 크게 잡았을 때 오차



평균 제곱 오차

▼ 그림 4-9 | 기울기를 너무 작게 잡았을 때 오차



평균 제곱 오차

- 그래프의 기울기가 잘못되었을수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
- 만일 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있는 것을 알 수 있음
- 빨간색 선의 거리의 합을 실제로 계산해 보자
- 거리는 입력 데이터에 나와 있는 y의 '실제 값'과 x를 y = 3x + 76 식에 대입해서 나오는 '예측 값'의 차이를 이용해 구할 수 있음
- 예를 들어 2시간을 공부했을 때 실제 나온 점수(81점)와 그래프 y = 3x + 76 식에 x = 2를 대입했을 때(82점)의 차이가 곧 오차
- 오차를 구하는 방정식은 다음과 같음

$$\text{오차} = \text{실제 값} - \text{예측 값}$$

평균 제곱 오차

- 이 식에 주어진 데이터를 대입해 얻을 수 있는 모든 오차 값을 정리하면 표 4-3과 같음

▼ 표 4-3 | 주어진 데이터에서 오차 구하기

공부한 시간(x)	2	4	6	8
성적(실제 값, y)	81	93	91	97
예측 값	82	88	94	100
오차	1	–5	3	3

평균 제곱 오차

- 이렇게 해서 구한 오차를 모두 더하면 1 + (-5) + 3 + 3 = 2가 됨
- 이 값은 오차가 실제로 얼마나 큰지를 가늠하기에는 적합하지 않음
- 오차에 양수와 음수가 섞여 있어 오차를 단순히 더해 버리면 합이 0이 될 수도 있기 때문임
- 부호를 없애야 정확한 오차를 구할 수 있음
- 오차의 합을 구할 때는 각 오차 값을 제곱해 줌
- 이를 식으로 표현하면 다음과 같음

$$\text{오차의 합} = \sum_{i}^{n} (y_i - \hat{y}_i)^2$$

평균 제곱 오차

- 여기서 i는 x가 나오는 순서를, n은 x 원소의 총 개수를 의미
- yi는 xi에 대응하는 '실제 값'이고 는 xi가 대입되었을 때 직선의 방정식(여기서는 y = 3x + 76)이 만드는 '예측 값'
- 이 식으로 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44
- 우리가 구하고자 하는 평균 제곱 오차는 위에서 구한 오차의 합을 n으로 나눈 것

$$\text{평균 제곱 오차(MSE)} = \frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

평균 제곱 오차

- 이 식은 앞으로 머신 러닝과 딥러닝을 공부할 때 자주 등장할 중요한 식
- 앞서 구한 오차의 합(=44)과 x 원소의 총 개수(=4)를 이 식에 대입하면 ¼ × 44 = 11이란 값이 나옴
- 이로써 우리가 그은 임의의 직선이 11이라는 평균 제곱 오차를 갖는 직선이었다는 것을 알 수 있음
- 이제 우리의 작업은 11보다 작은 평균 제곱 오차를 가지게 만드는 a 값과 b 값을 찾는 것이 되었음
- 이렇듯 선형 회귀란 임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고,
  이 값을 가장 작게 만들어 주는 a 값과 b 값을 찾아가는 작업

06
파이썬 코딩으로 확인하는 평균 제곱 오차

파이썬 코딩으로 확인하는 평균 제곱 오차

- 이제 앞서 알아본 평균 제곱 오차를 파이썬으로 구현해 보자
- 임의로 정한 기울기 a와 y 절편 b의 값이 각각 3과 76이라고 할 때, 가상의 기울기가 fake_a, 가상의 y 절편이 fake_b인 함수식 predict()를 다음과 같이 정의할 수 있음

```python
fake_a = 3
fake_b = 76

def predict(x):
    return fake_a * x + fake_b
```

파이썬 코딩으로 확인하는 평균 제곱 오차

- 위 코드의 결괏값이 들어갈 빈 리스트를 만듦

```
predict_result = []
```

파이썬 코딩으로 확인하는 평균 제곱 오차

파이썬 코딩으로 확인하는 평균 제곱 오차

- 이제 모든 x 값을 predict() 함수에 한 번씩 대입해 예측 값 리스트를 채우는 코드를 다음과 같이 작성

```
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict
(x[i])))
```

파이썬 코딩으로 확인하는 평균 제곱 오차

- 다음으로 평균 제곱 오차를 구하는 함수를 만들 차례
- 평균 제곱 오차 공식을 그대로 파이썬 함수로 옮기면 다음과 같음

$$\frac{1}{n}\sum(y_i - \hat{y}_i)^2$$

```
n = len(x)
def mse(y, y_pred):
    return (1/n) * sum((y - y_pred)**2)
```

파이썬 코딩으로 확인하는 평균 제곱 오차

- 여기서 **2는 제곱을 구하라는 것이고, sum()은 합을 구하라는 것
- 실제 값과 예측 값을 각각 mse() 함수의 y와 y_pred 자리에 넣어서 평균 제곱을 구함

실습 파이썬 코딩으로 구하는 평균 제곱 오차

```
import numpy as np

# 가상의 기울기 a와 y 절편 b를 정합니다.
fake_a = 3
fake_b = 76

# 공부 시간 x와 성적 y의 넘파이 배열을 만듭니다.
x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])
```

파이썬 코딩으로 확인하는 평균 제곱 오차

☑ 파이썬 코딩으로 확인하는 평균 제곱 오차

```
# y = ax + b에 가상의 a 값과 b 값을 대입한 결과를 출력하는 함수입니다.
def predict(x):
    return fake_a * x + fake_b

# 예측 값이 들어갈 빈 리스트를 만듭니다.
predict_result = []

# 모든 x 값을 한 번씩 대입해 predict_result 리스트를 완성합니다.
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict
(x[i])))
```

☑ 파이썬 코딩으로 확인하는 평균 제곱 오차

```python
# 평균 제곱 오차 함수를 각 y 값에 대입해 최종 값을 구하는 함수입니다.
n = len(x)
def mse(y, y_pred):
    return (1/n) * sum((y - y_pred)**2)

# 평균 제곱 오차 값을 출력합니다.
print("평균 제곱 오차: " + str(mse(y, predict_result)))
```

파이썬 코딩으로 확인하는 평균 제곱 오차

실행 결과

```
공부시간=2, 실제점수=81, 예측점수=82
공부시간=4, 실제점수=93, 예측점수=88
공부시간=6, 실제점수=91, 예측점수=94
공부시간=8, 실제점수=97, 예측점수=100
평균 제곱 오차: 11.0
```

파이썬 코딩으로 확인하는 평균 제곱 오차

- 이를 통해 우리가 처음 가정한 a = 3, b = 76은 오차가 약 11.0이라는 것을 알게 됨
- 이제 남은 것은 이 오차를 줄이면서 새로운 선을 긋는 것
- 이를 위해서는 a 값과 b 값을 적절히 조절하면서 오차의 변화를 살펴보고, 그 오차가 최소화되는 a 값과 b 값을 구해야 함