4장. 오차 수정하기: 경사 하강법

1. 미분의 개념

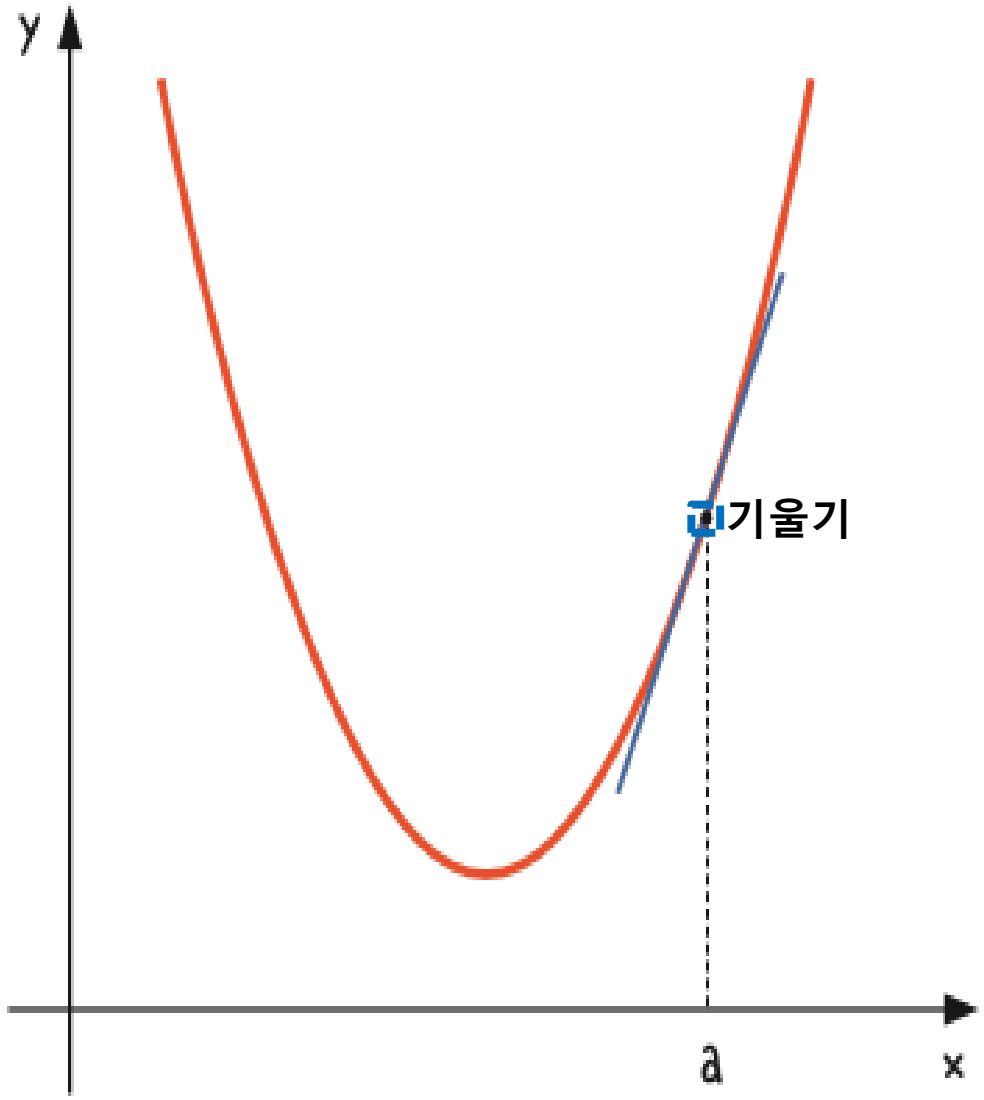
- **순간 변화율**의 의미

  → a: 미세한 변화 -> y: 미세한 변화

  변화가 있긴 하지만, 그 움직임이 너무 미세하면?

  → **'순간 변화율'**

- 순간 변화율은 '어느 쪽'이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선을 그릴 수가 있음



1. 미분의 개념

- 미분이란?
  - → x 값이 아주 미세하게 움직일 때의 y 변화량을 구한 뒤,
  - → 이를 x의 변화량으로 나누는 과정
  - → 한 점에서의 순간 기울기

- "함수 f(x)를 미분하라" $\frac{d}{dx}f(x)$ 라고 표기함.

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

① 함수 f(x)를 x로 미분하라는 것은

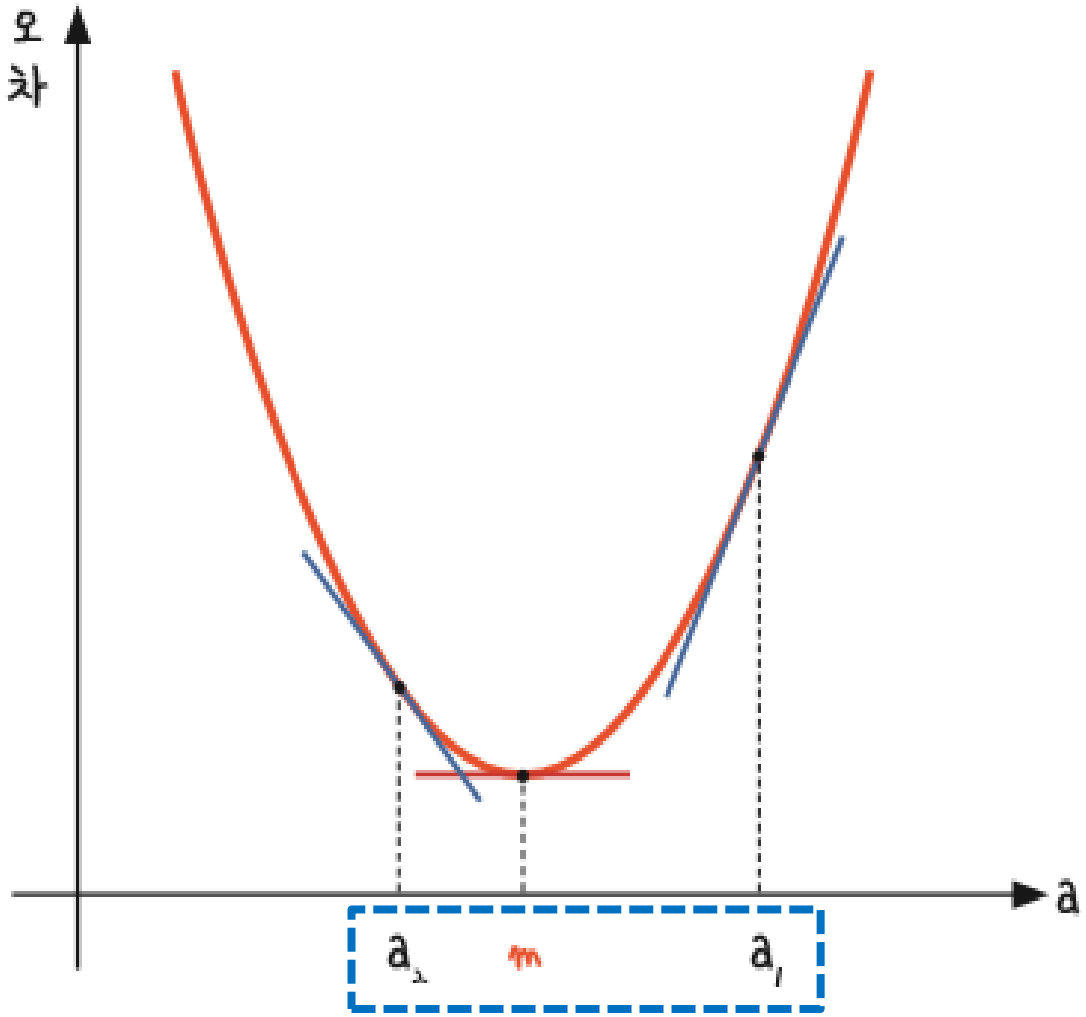
② x의 변화량이 0에 가까울 만큼 작을 때

③ y 변화량의 차이를

④ x 변화량으로 나눈 값(= 순간 변화율)을 구하라는 뜻!

2. 경사 하강법의 개요

- $y = x^2$ -> x에 a_1, a_2, m대입 후 미분
  → 각 점에서의 순간 기울기가 그려짐
- 꼭지점의 기울기
  → x축과 평행한 선이 됨 (기울기: 0)
- '미분 값이 0인 지점'을 찾는 것

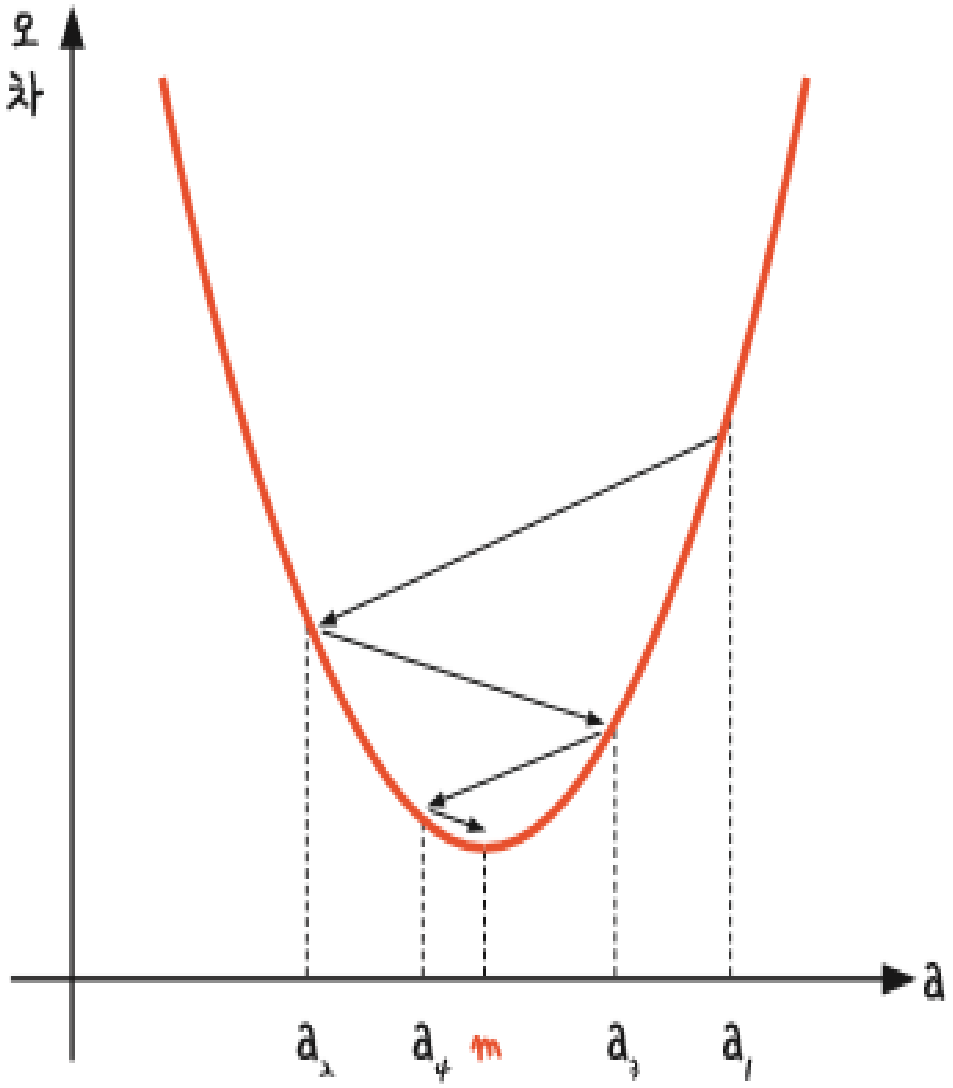


2. 경사 하강법의 개요

- 이를 위해서 다음과 같은 과정을 거침

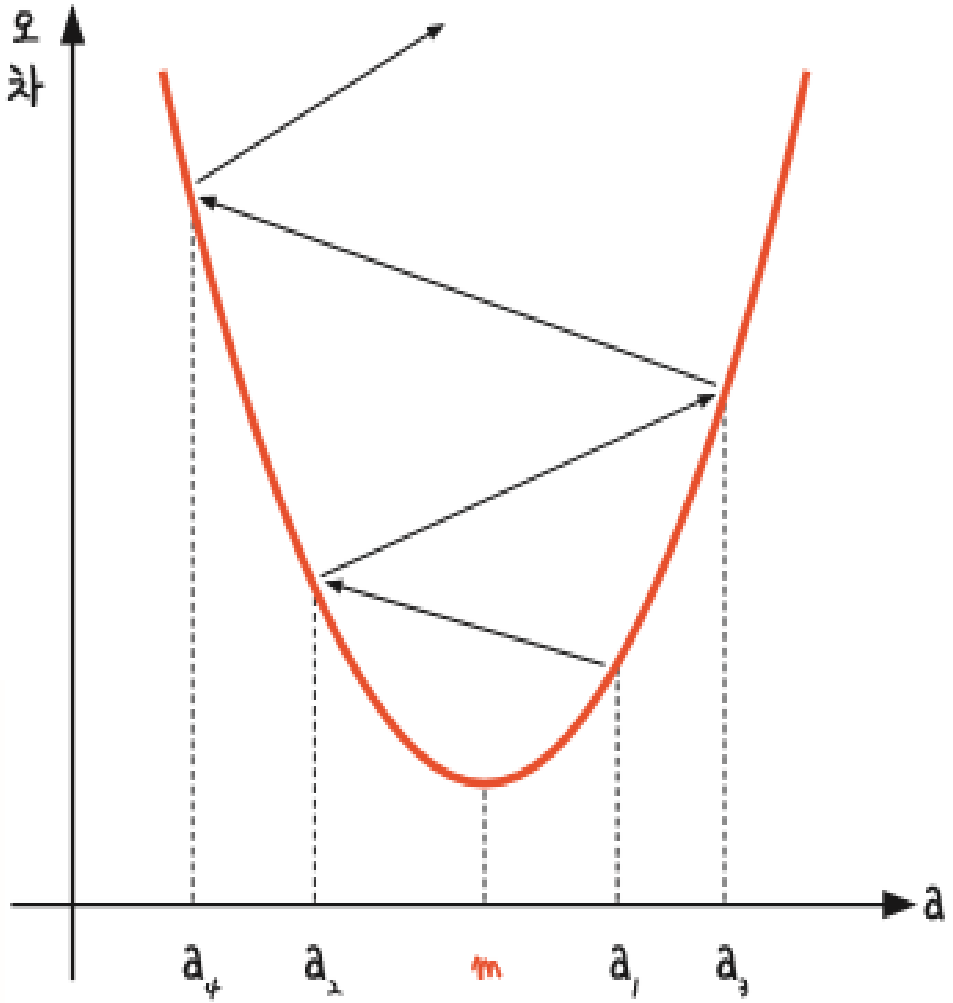
1) a_1에서 미분을 구한다.
2) 구해진 기울기의 반대 방향 얼마간 이동시킨 a_2에서 미분을 구한다.
3) a_3에서 미분을 구한다.
4) 3의 값이 0이 아니면 a_2에서 2~3번 과정을 반복한다.

- **경사 하강법은** 이렇게 반복적으로 기울기 a를 변화시켜서 m의 값을 찾아내는 방법



3. 학습률

- 기울기의 부호를 바꿔 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면 a 값이 한 점으로 모이지 않고 위로 치솟아 버림
- 어느 만큼 이동시킬지를 정해주는 것 : **학습률** (learning rate)



4. 코딩으로 확인하는 경사 하강법

```
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x_data = [x_row[0] for x_row in data]
y_data = [y_row[1] for y_row in data]

a = tf.Variable(tf.random_uniform([1], 0, 10, dtype = tf.float64, seed = 0))
b = tf.Variable(tf.random_uniform([1], 0, 100, dtype = tf.float64, seed = 0))

y = a * x_data + b

rmse = tf.sqrt(tf.reduce_mean(tf.square( y - y_data )))

learning_rate = 0.1

gradient_decent = tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(2001):
    sess.run(gradient_decent)
    if step % 100 == 0:
            print("Epoch: %.f, RMSE = %.04f, 기울기 a = %.4f, y 절편 b = %.4f" %
                    (step,sess.run(rmse),sess.run(a),sess.run(b)))
```

임의의 수를 생성

학습률

최소 평균 제곱근 오차

4. 코딩으로 확인하는 경사 하강법

- **실행 결과**

```
Epoch: 0, RMSE = 28.6853, 기울기 a = 7.2507, y 절편 b = 80.5525
Epoch: 100, RMSE = 2.8838, 기울기 a = 2.2473, y 절편 b = 79.3146
Epoch: 200, RMSE = 2.8815, 기울기 a = 2.2774, y 절편 b = 79.1348
Epoch: 300, RMSE = 2.8811, 기울기 a = 2.2903, y 절편 b = 79.0578
Epoch: 400, RMSE = 2.8810, 기울기 a = 2.2959, y 절편 b = 79.0247
Epoch: 500, RMSE = 2.8810, 기울기 a = 2.2982, y 절편 b = 79.0106
Epoch: 600, RMSE = 2.8810, 기울기 a = 2.2992, y 절편 b = 79.0045
Epoch: 700, RMSE = 2.8810, 기울기 a = 2.2997, y 절편 b = 79.0019
Epoch: 800, RMSE = 2.8810, 기울기 a = 2.2999, y 절편 b = 79.0008
Epoch: 900, RMSE = 2.8810, 기울기 a = 2.2999, y 절편 b = 79.0004
Epoch: 1000, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0002
Epoch: 1100, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0001
Epoch: 1200, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1300, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1400, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1500, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1600, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1700, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1800, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 1900, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
Epoch: 2000, RMSE = 2.8810, 기울기 a = 2.3000, y 절편 b = 79.0000
```

5. 다중 선형 회귀

- 4시간 공부한 친구는 88점을 예측했는데 이보다 좋은 93점을 받았고, 6시간 공부한 친구는 93점을 받을 것으로 예측했지만 91점을 받았음

  → **<u>예측과 실제값에 차이가 있다.</u>**

- 차이가 생기는 이유는 공부한 시간 이외의 **다른 요소가** 성적에 영향을 끼쳤기 때문
- 더 정확한 예측을 하려면 **추가 정보**를 입력해야 함
- 정보를 추가해 새로운 예측 값을 구하려면 변수의 개수를 늘려 **'다중 선형 회귀'**를 만들어 주어야 한다

공부한 시간(x_1)	2	4	6	8
과외 수업 횟수(x_2)	**0**	**4**	**2**	**3**
성적(y)	81	93	91	97

→ 두 개의 독립 변수 x_1과 x_2가 생긴 것

$$y = ax + b \quad \rightarrow \quad y = a_1x_1 + a_2x_2 + b$$

6. 코딩으로 확인하는 다중 선형 회귀

```
data = [[2, 0, 81], [4, 4, 93], [6, 2, 91], [8, 3, 97]]
x1 = [x_row1[0] for x_row1 in data]
x2 = [x_row2[1] for x_row2 in data]
y_data = [y_row[2] for y_row in data]

a1 = tf.Variable(tf.random_uniform([1], 0, 10, dtype=tf.float64, seed=0))
a2 = tf.Variable(tf.random_uniform([1], 0, 10, dtype=tf.float64, seed=0))
b = tf.Variable(tf.random_uniform([1], 0, 100, dtype=tf.float64, seed=0))

y = a1 * x1 + a2 * x2+ b

rmse = tf.sqrt(tf.reduce_mean(tf.square( y - y_data )))

learning_rate = 0.1

gradient_decent = tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for step in range(2001):
    sess.run(gradient_decent)
    if step % 100 == 0:
            print("Epoch: %.f, RMSE = %.04f, 기울기 a1 = %.4f, 기울기 a2 = %.4f, y 절편 b = %.4f" %
                  (step,sess.run(rmse),sess.run(a1), sess.run(a2),sess.run(b)))
```

임의의 수를 생성

학습률

최소 평균 제곱근 오차

6. 코딩으로 확인하는 다중 선형 회귀

- **실행 결과**

```
Epoch: 0, RMSE = 49.1842, 기울기 a1 = 7.5270, 기울기 a2 = 7.8160, y 절편 b = 80.5980
Epoch: 100, RMSE = 1.8368, 기울기 a1 = 1.1306, 기울기 a2 = 2.1316, y 절편 b = 78.5119
Epoch: 200, RMSE = 1.8370, 기울기 a1 = 1.1879, 기울기 a2 = 2.1487, y 절편 b = 78.1057
Epoch: 300, RMSE = 1.8370, 기울기 a1 = 1.2122, 기울기 a2 = 2.1571, y 절편 b = 77.9352
Epoch: 400, RMSE = 1.8370, 기울기 a1 = 1.2226, 기울기 a2 = 2.1607, y 절편 b = 77.8636
Epoch: 500, RMSE = 1.8370, 기울기 a1 = 1.2269, 기울기 a2 = 2.1622, y 절편 b = 77.8335
Epoch: 600, RMSE = 1.8370, 기울기 a1 = 1.2288, 기울기 a2 = 2.1628, y 절편 b = 77.8208
Epoch: 700, RMSE = 1.8370, 기울기 a1 = 1.2295, 기울기 a2 = 2.1631, y 절편 b = 77.8155
Epoch: 800, RMSE = 1.8370, 기울기 a1 = 1.2299, 기울기 a2 = 2.1632, y 절편 b = 77.8133
Epoch: 900, RMSE = 1.8370, 기울기 a1 = 1.2300, 기울기 a2 = 2.1632, y 절편 b = 77.8124

(중략)

Epoch: 2000, RMSE = 1.8370, 기울기 a1 = 1.2301, 기울기 a2 = 2.1633, y 절편 b = 77.8117
```