9주차

오차 수정하기: 경사하강법

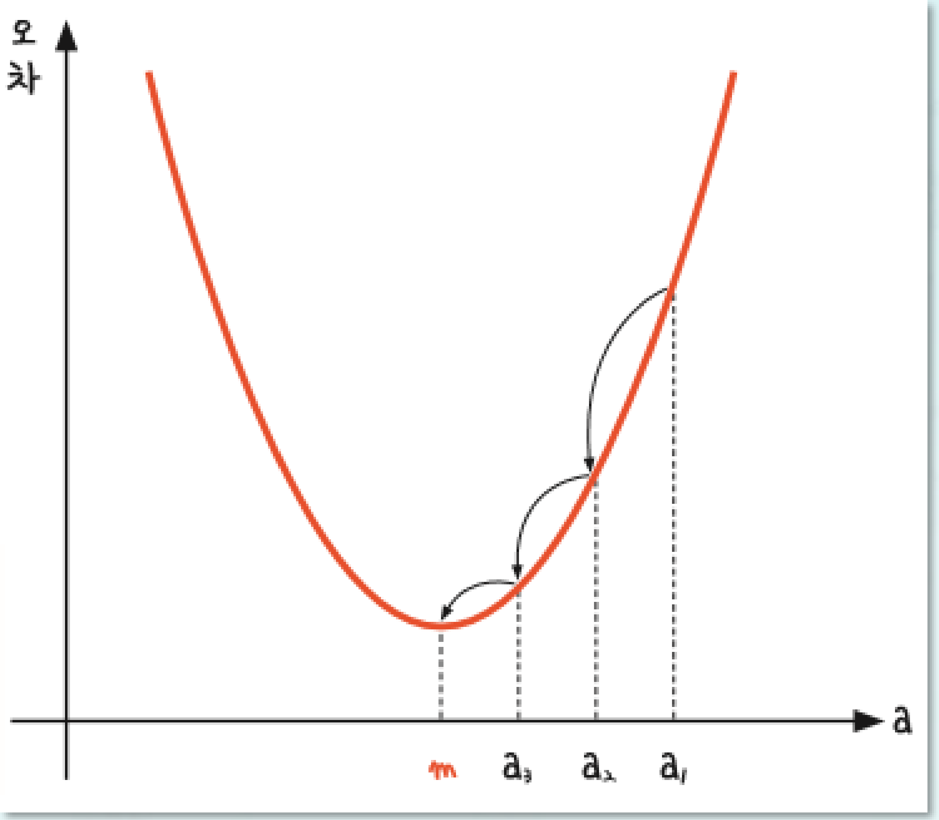
모두를 위한 인공지능의 활용

한동대학교
김영섭 교수

미분의 개념

- 기울기 a 무한대 → 오차 무한대
- a를 무한대로 감소 → 오차도 무한대로 감소
- 위와 같은 관계는 “이차 함수 그래프”로 표현
- 이차 함수 그래프에서 오차 최소는 기울기 a가 m에 있을 때
- 오차가 가장 작은 점을 찾는것!!

  → 경사 하강법

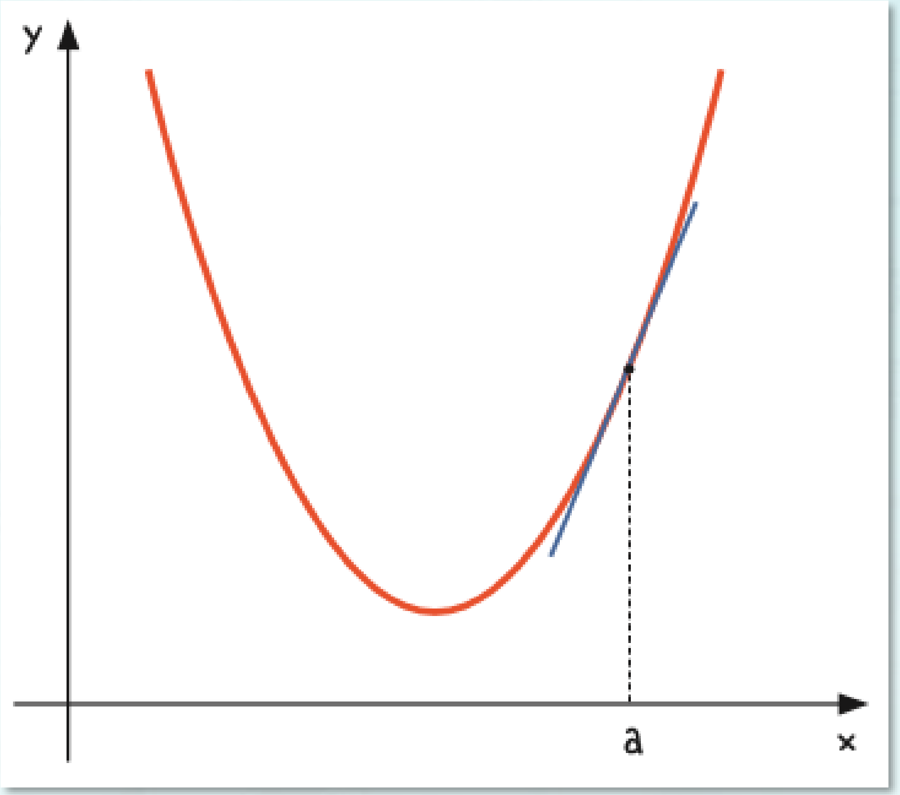


미분의 개념

- 순간 변화율의 의미
  - **a**가 변화량이 **0**에 가까울 만큼 아주 미세하게 변화했다면,
  - **y** 값의 변화 역시 아주 미세해서 **0**에 가까울 것
- 변화가 있긴 하지만**,** 그 움직임이 너무 미세하면**?**
  - 어느 쪽으로 '움직이려고 시도했다'는 정도의 느낌만 있을 뿐.
  - 이 느낌을 수학적으로 이름 붙인 것이 바로 '순간 변화율'

미분의 개념

- 순간 변화율은 ‘어느 쪽’이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선을 그릴 수가 있음
- 이 선이 바로 이 점에서의 ‘기울기’라고 불리는 접선



미분의 개념

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

$$\frac{d}{dx}f(x)$$

- "함수 **f(x)**를 미분하라"는 라고 표기함

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

③ y 변화량의 차이를

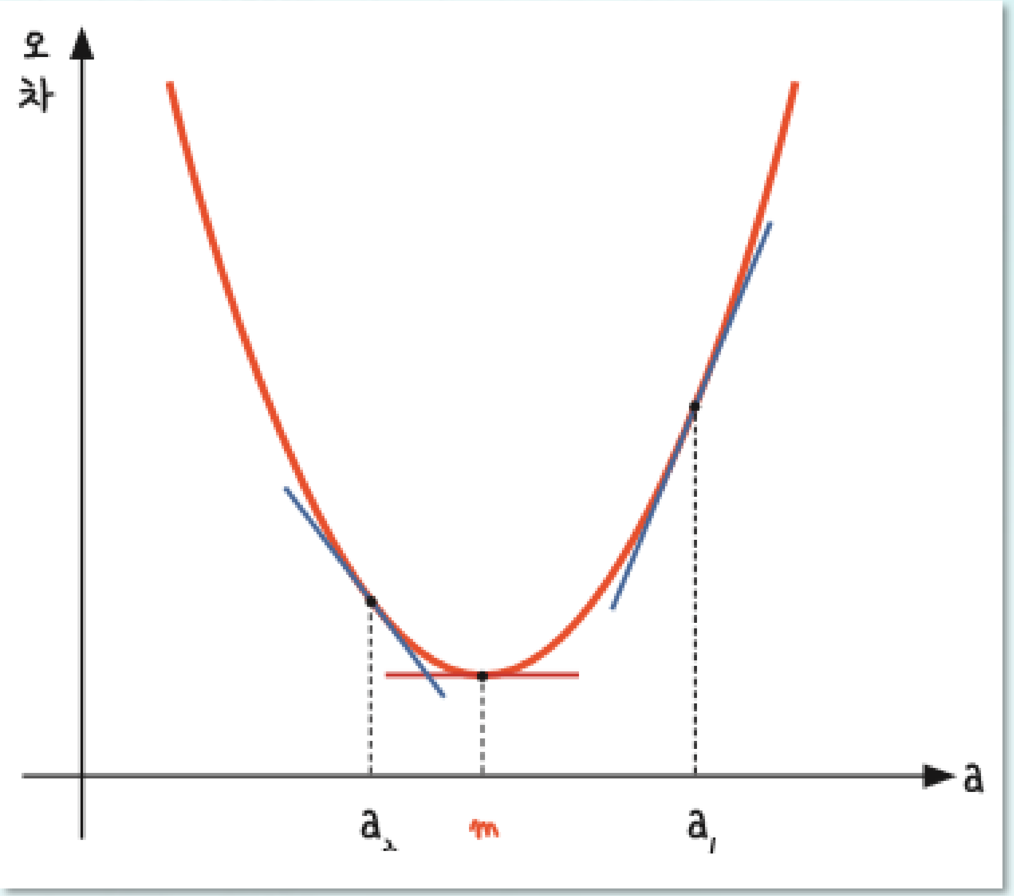
④ x 변화량으로 나눈 값(= 순간 변화율)을 구하라는 뜻!

② x의 변화량이 0에 가까울 만큼 작을 때

① 함수 f(x)를 x로 미분하라는 것은

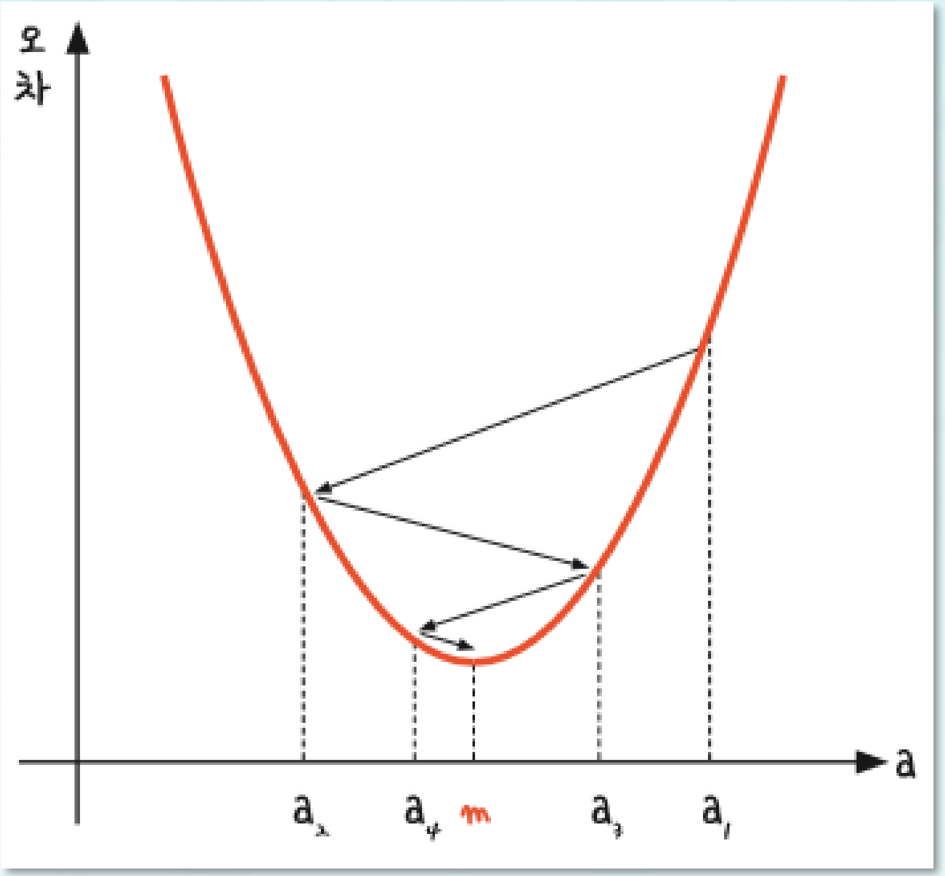
미분의 개념

- y= x^2 그래프에서 x에 a_1, a_2 그리고 m을 대입하여 그 자리에서 미분하면 각 점에서의 순간 기울기가 그려짐
- 알고 싶은 것 : 최솟값 m에서의 순간 기울기 → 미분 값이 0인 지점



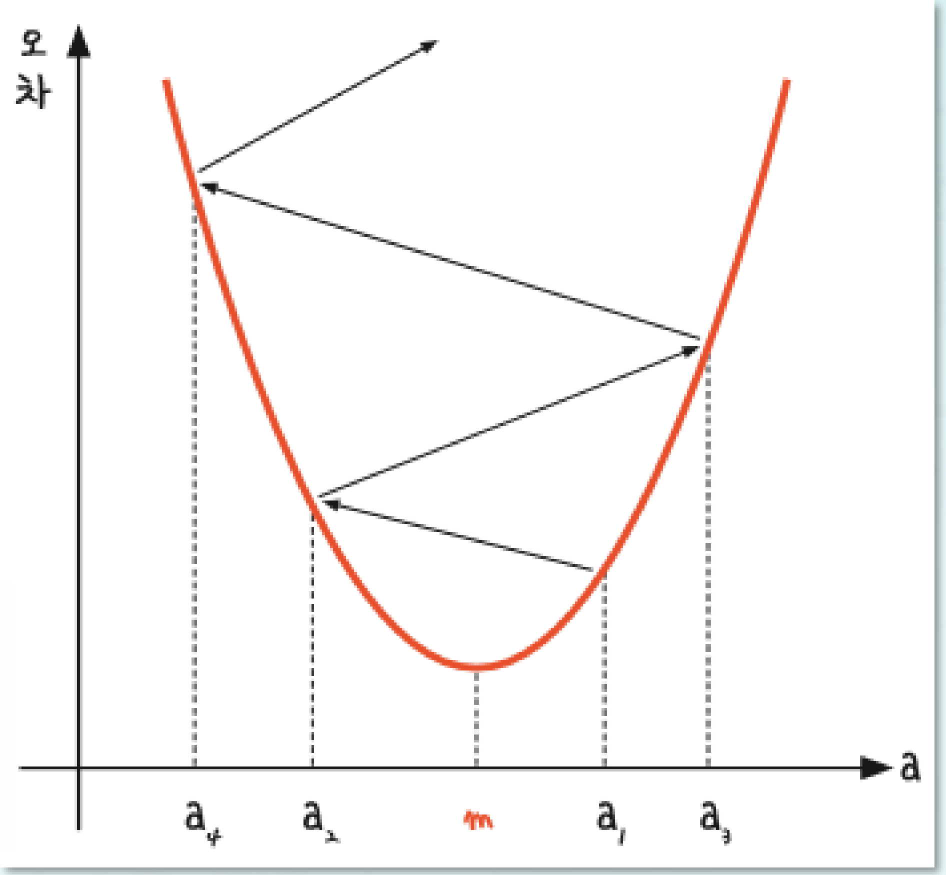
미분의 개념

1. a1에서 미분을 구한다.
2. 구해진 기울기의 반대 방향 얼마간 이동시킨 a2에서 미분을 구한다.
3. a3에서 미분을 구한다.
4. 3의 값이 0이 아니면 a2에서 2~3번 과정을 반복한다.



학습률

- 기울기의 부호를 바꿔 이동시킬 때 너무 멀리 이동시키면 **a** 값이 위로 치솟음
- 학습률을 너무 크게 잡으면 발산
- 어느 만큼 이동시킬지를 정해주는 것 → 학습률
- **y**절편 **b** 또한 **b** 값이 크면 오차도 함께 커지고 너무 작아도 오차가 커짐. 최적의 **b**값을 구할 때도 경사 하강법 사용



코딩으로 확인하는 경사 하강법

- 텐서플로 : 구글이 오픈 소스 라이선스로 공개한 딥러닝 전문 라이브러리
- **3**장에서 배운 데이터 입력과 **x, y**를 지정하는 방법에 학습률이 추가

```python
import tensorflow as tf

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]

learning_rate = 0.1
```

코딩으로 확인하는 경사 하강법

- 임의의 기울 **a**와 **y** 절편 **b** 설정
- 기울기는 **0~10** 사이
  y 절편은 **0~100** 사이 임의의 값
- **Tensorflow** 라이브러리를 **tf** 약어
- 변수의 값을 정할 때 **Variable()**
- **random_uniform()** 임의의 수를 생성해 주는 함수
- 예**. random_uniform([1], 0, 10,…)** 은 **0**에서 **10** 사이에서 임의의 수 **1**개 생성
- 데이터 형식은 실수형**(float64)** 실행 시 같은 값이 나올 수 있게 **seed** 값을 설정

```
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))
```

코딩으로 확인하는 경사 하강법

- 일차 방정식 **ax + b**의 식을 구현

```
y = a * x_data + b
```

- 평균 제곱근 오차의 식을 구현

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

코딩으로 확인하는 경사 하강법

- 텐서플로의 **GradientDescentOptimizer()** 함수를 이용하여 경사 하강법의 결과를 **gradient_decent**에 할당
- 앞서 지정한 **learning_rate**와 평균 제곱근 오차를 통해 구한 **rmse**를 사용

```
gradient_decent =
tf.train.GradientDescentOptimizer(learning
_rate). minimize(rmse)
```

코딩으로 확인하는 경사 하강법

- 텐서플로를 실행 및 결괏값 출력

```
with tf.Session() as sess:
    # 변수 초기화
    sess.run(tf.global_variables_initializer())
    # 2001번 실행(0번째를 포함하므로)
    for step in range(2001):
        sess.run(gradient_decent)
        # 100번마다 결과 출력
        if step % 100 == 0:
            print("Epoch: %.f, RMSE = %.04f, 기
울기 a = %.4f, y 절편 b = %.4f" %

(step,sess.run(rmse),sess.run(a),sess.run(b)))
```

코딩으로 확인하는 경사 하강법

```
import tensorflow as tf

# x, y의 데이터 값
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]

# 기울기 a와 y 절편 b의 값을 임의로 정한다.
# 단, 기울기의 범위는 0 ~ 10 사이이며, y 절편은 0 ~
100 사이에서 변하게 한다.
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))

# y에 대한 일차 방정식 ax+b의 식을 세운다.
y = a * x_data + b
```

```
# 텐서플로 RMSE 함수
rmse = tf.sqrt(tf.reduce_mean(tf.square( y - y_data )))

# 학습률 값
learning_rate = 0.1

# RMSE 값을 최소로 하는 값 찾기
gradient_decent =
tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)

# 텐서플로를 이용한 학습
with tf.Session() as sess:
    # 변수 초기화
    sess.run(tf.global_variables_initializer())
    # 2001번 실행(0번째를 포함하므로)
    for step in range(2001):
        sess.run(gradient_decent)
        # 100번마다 결과 출력
        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)))
```

코딩으로 확인하는 경사 하강법

- 에포크(**Epoch**) : 입력 값에 대해 몇 번이나 반복하여 실험했는지
- 평균 제곱근 오차(**RMSE**)의 변화, 기울기 **a**가 **2.3**에 수렴, **y** 절편
  b가 **79**에 수렴하는 과정
- 기울기 **2.3**과 **y** 절편 **79**는 최소 제곱법을 통해 계산한 값과 같다.
- 최소 제곱법을 쓰지 않고 경사 하강법을 통해 기울기와 절편의 값을 구할 수 있음
- 다중 선형회귀에서도 사용

```
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
```

다중 선형 회귀

- **4**시간 공부한 친구는 **88**점을 예측했는데 이보다 좋은 **93**점을 받았고**, 6**시간 공부한 친구는 **93**점을 받을 것으로 예측했지만 **91**점을 받음
  예측과 실제값에 차이**!**
- 차이가 생기는 이유는 공부한 시간 이외의 다른 요소가 성적에 영향을 끼쳤기 때문
- 정보를 추가해 새로운 예측 값을 구하려면 변수의 개수를 늘려 '다중 선형 회귀'를 만들어 주어야 한다

다중 선형 회귀

- 일주일 동안 받는 과외 수업 횟수를 조사해서 이를 기록해 보면,
  → 두 개의 독립 변수 x_1과 x_2가 생긴 것
- 종속 변수 y를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴
- 두 기울기 a_1 과 a_2 는 경사 하강법을 그대로 적용

$$y = a_1x_1 + a_2x_2 + b$$

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

코딩으로 확인하는 다중 선형 회귀

- 지금까지 배운 내용을 토대로 다중 선형 회귀를 작성해 보자
- 텐서플로를 불러온 뒤 **x**와 **y**의 값을 지정하는 과정은 동일함
- 다만, 이번에는 **x1**과 **x2**라는 두 개의 독립 변수 리스트를 만들어 줌

```
import tensorflow as tf

# x1, x2, y의 데이터 값

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]   # 새로 추가되는 값
y_data = [y_row[2] for y_row in data]
```

코딩으로 확인하는 다중 선형 회귀

- 이제 앞서 기울기의 값을 구하는 방식 그대로 또 하나의 기울기 a_2를 구함

```
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)) # 새로 추가되는 값
b = tf.Variable(tf.random_uniform([1], 0, 100,
dtype=tf.float64, seed=0))
```

- 이제 새로운 방정식 $y = a_1x_1 + a_2x_2 + b$ 에 맞춰 다음과 같이 식을 세움

```
y = a1 * x1 + a2 * x2+ b
```

코딩으로 확인하는 다중 선형 회귀

- 나머지 라인은 앞서 배운 선형 회귀와 같음
- 결과를 출력하는 부분만 기울기가 두 개 나올 수 있게 수정

```
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)))
```

코딩으로 확인하는 다중 선형 회귀

```python
import tensorflow as tf

# x1, x2, y의 데이터 값

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]   # 새로 추가되는 값
y_data = [y_row[2] for y_row in data]

# 기울기 a와 y 절편 b의 값을 임의로 정한다.
# 단, 기울기의 범위는 0 ~ 10 사이이며, y 절편은 0 ~ 100 사이에서 변하게
한다.
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))
# 새로 추가되는 값
b = tf.Variable(tf.random_uniform([1], 0, 100, dtype=tf.float64, seed=0))
```

```python
# 새로운 방정식
y = a1 * x1 + a2 * x2+ b

# 텐서플로 RMSE 함수
rmse = tf.sqrt(tf.reduce_mean(tf.square( y - y_data )))

# 학습률 값
learning_rate = 0.1

# RMSE 값을 최소로 하는 값 찾기
gradient_decent =
tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)

# 학습이 진행되는 부분
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)))
```

코딩으로 확인하는 다중 선형 회귀

- 과외 수업 횟수라는 새로운 변수가 추가되면서 **1**차원 직선에서만 움직이던 예측 결과가 더 넓은 **3**차원 평면 범위 안에서 좀 더 정밀한 예측을 할 수 있게 됨

```
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
```