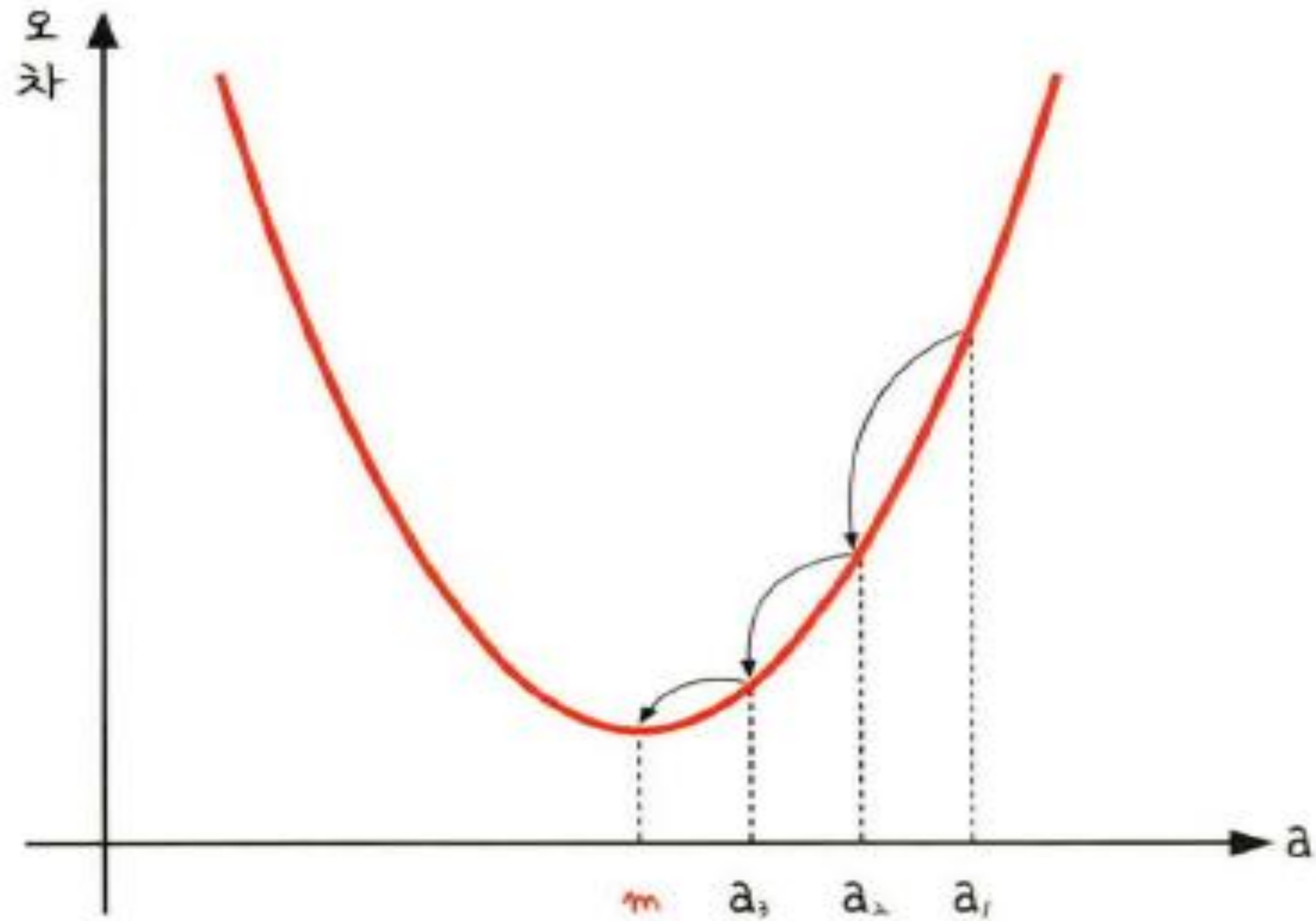
5. 오차 수정하기: 경사 하강법

서론

- 우리는 앞서 기울기 a를 너무 크게 잡으면 오차가 커지는 것을 확인했습니다.
- 기울기를 너무 작게 잡아도 오차가 커집니다.
- 기울기 a와 오차 사이에는 이렇게 상관관계가 있음을 알 수 있습니다.
- a를 무한대로 키우면 오차도 무한대로 커지고 a를 무한대로 작게 해도 역시 오차도 무한대로 작아지는 이러한 관계는 이차 함수 그래프로 표현할 수 있습니다.

서론

- 그림 은 기울기 a와 오차와의 관계를 이차 함수 그래프로 그려본 것입니다.



서론

- 오차가 가장 작은 점은 어디일까요?
- 그림의 그래프에서 오차가 가장 작을 때는 x가 그래프의 가장 아래쪽의 볼록한 부분에 이르렀을 때입니다.
- 즉, 기울기 a가 m 위치에 있을 때입니다.

서론

- 컴퓨터를 이용해 m의 값을 구하려면 임의의 한 점 (a)을 찍고 이 점을 m에 가까운 쪽으로 점점 이동($a_1 \rightarrow a_2 \rightarrow a_3$)시키는 과정이 필요합니다
- 그러려면 a_1 의 값보다 a_2의 값이 m에 더 가깝고 a_2 의 값보다 a_3 가 m에 더 가깝다는 것을 컴퓨터가 알아야 하겠지요.
- 이렇게 그래프에서 오차를 비교하여 가장 작은 방향으로 이동시키는 방법이 있습니다.
- 바로 미분 기울기를 이용하는 경사 하강법 (gradient decent) 입니다.

미분의 개념

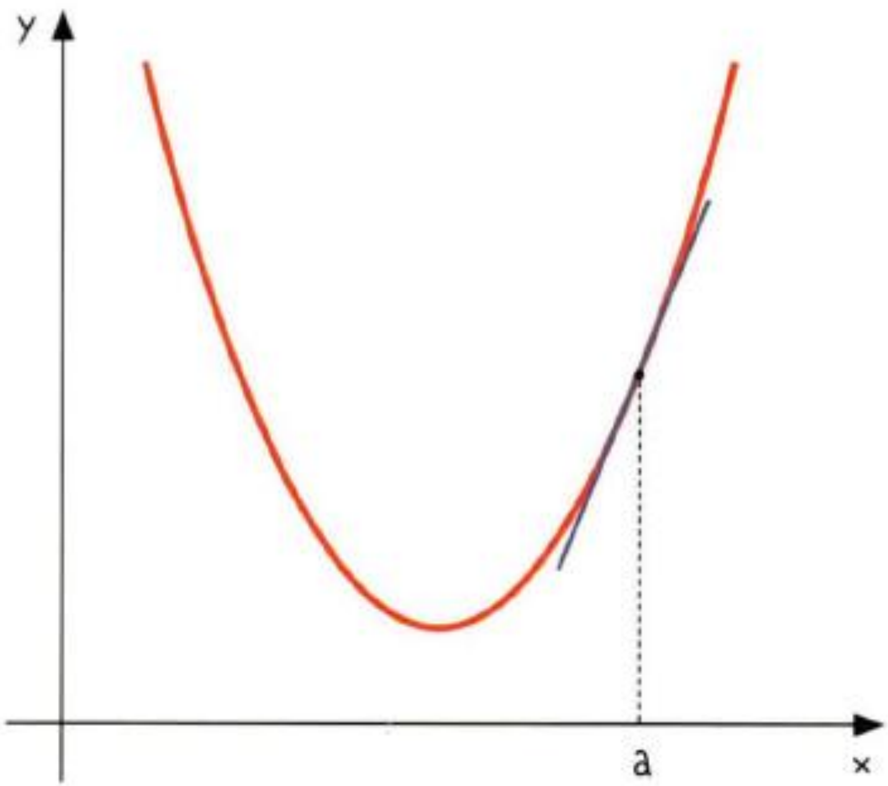
- 우리는 어느 지점이 우리가 원하는 m 값인지 모릅니다.
- 이를 알아내고자 새로운 아이디어가 필요할 때, 미분이 이 문제를 해결해 준다는 것을 발견했습니다.
- 여기서 잠시 미분과 기울기의 개념을 정리해 볼까요?
- $y = x^2$이 라는 그래프가 있다고 해 봅시다.
- x축에 한 점 a가 있을 때 이 값의 y 값은 $y = a^2$ 입니다.
- a가 아주 미세하게 오른쪽이나 왼쪽으로 이동하면 종속 변수인 y 값도 그에 따라 아주 미세하게 변화하겠지요.

미분의 개념

- 만약 a가 변화량이 0에 가까울 만큼 아주 미세하게 변화했다고 합시다.
- 그럼 y 값의 변화 역시 아주 미세해서 0에 가깝겠지요.
- 변화가 있긴 하지만, 그 움직임이 너무 미세하면 어느 쪽으로 '움직이려고 시도했다는 정도의 느낌만 있을 뿐입니다.
- 이 느낌을 수학적으로 이름 붙인 것이 바로 '순간 변화율' 입니다.

미분의 개념

- 순간 변화율은 '어느 쪽'이라는 방향성을 지니고 있으므로 이 방향에 맞추어 직선을 쭈욱 그릴 수가 있습니다.
- 이 선이 바로 이 점 에서의 '기울기'라고 불리는 접선입니다.



미분의 개념

- 곧 미분은 x 값이 아주 미세하게 움직일 때의 y 변화량을 구한 뒤, 이를 x의 변화량으로 나누는 과정 입니다
- "함수 f(x)를 미분하라"는 $\frac{d}{dx}f(x)$라고 표기하는데, 이를 식으로 표현하면 다음과 같습니다.

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

③ y 변화량의 차이를

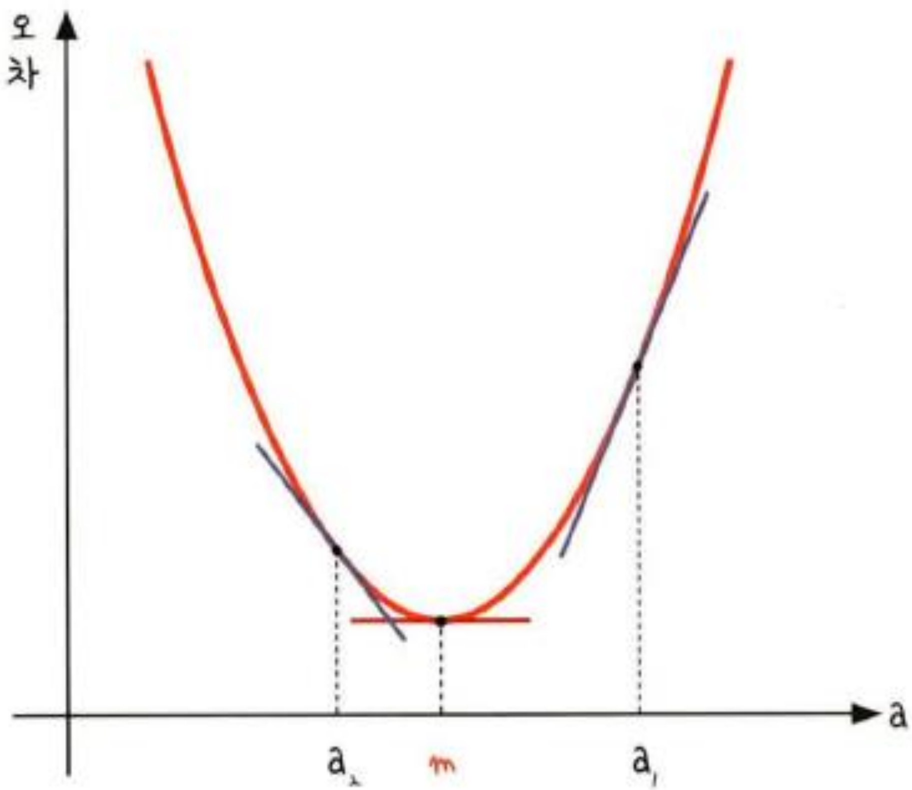
④ x 변화량으로 나눈 값(= 순간 변화율)을 구하라는 뜻!

② x의 변화량이 0에 가까울 만큼 작을 때

① 함수 f(x)를 x로 미분하라는 것은

경사 하강법의 개요

- 미분은 한 점에서의 순간 기울기라고 했습니다.
- 즉, $y = x^2$ 그래프에서 x에 다음과 같이 a_1, a_2 그리 고 m을 대입하여 그 자리에서 미분하면 그림처럼 각 점에서의 순간 기울기가 그려집니다.



경사 하강법의 개요

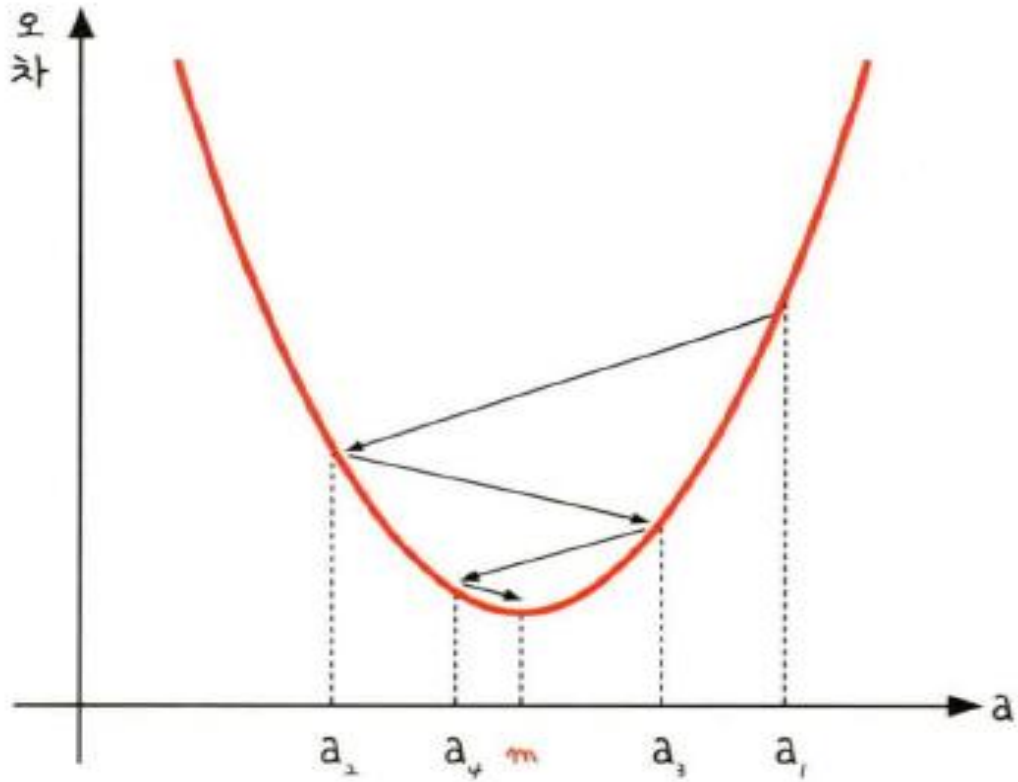
- 여기서 눈여겨 봐야할 것은 우리가 찾는 최솟값 m에서의 순간 기울기입니다.
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x축과 평행한 선이 됩니다.
- 즉, 기울기가 0입니다.
- 따라서 우리가 할 일은 '미분 값이 0인 지점'을 찾는 것이 됩니다.

경사 하강법의 개요

- 이를 위해서 다음과 같은 과정을 거칩니다.
  - 1. a_1에서 미분을 구한다.
  - 2. 구해진 기울기의 반대 방향(그래프에서 왼쪽 방향)으로 얼마간 이동시킨 a_2에서 미분을 구한다.
  - 3. a_3에서 미분을 구한다.
  - 4. 3의 값이 0이 아니면 a_2 에서 2 ~3 번 과정을 반복한다.

경사 하강법의 개요

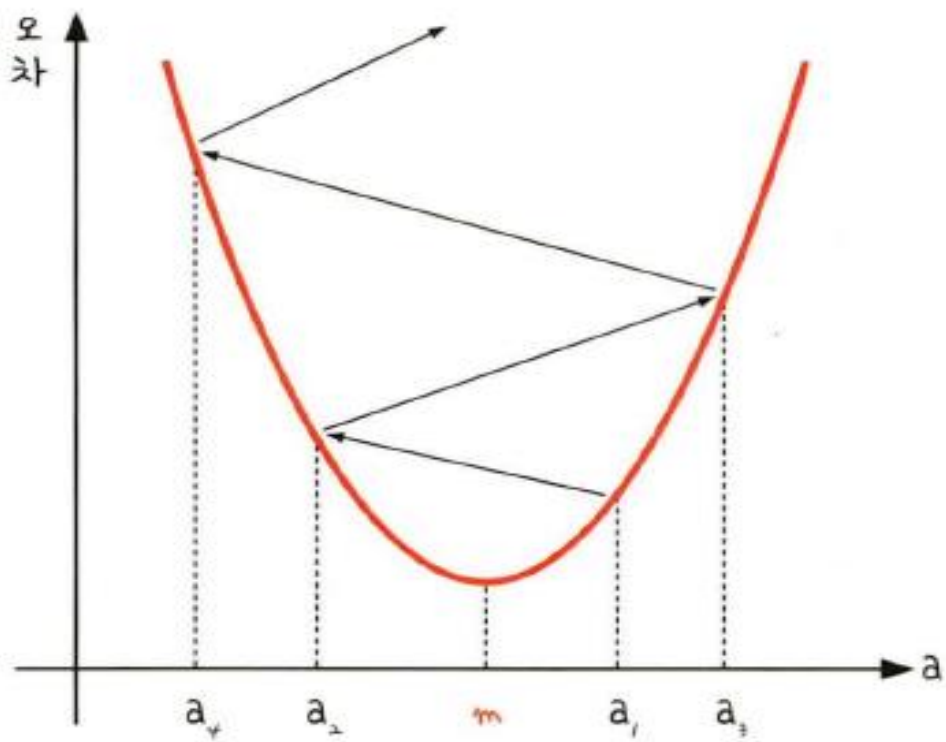
- 그러면 그림처럼 이동 결과가 한 점으로 수렴합니다.



- 경사 하강법은 이렇게 반복적으로 기울기 a를 변화시켜서 m의 값을 찾아내는 방법을 말합니다.

학습률

- 여기서 우리는 학습률(Learning rate)이라는 개념 을 알 수 있습니다.
- 기울기의 부호를 바꿔 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면 a값이 한 점으로 모이지 않고 그림처럼 위로 치솟아버립니다.



학습률

- 따라서 어느 만큼 이동시킬지를 신중히 결정해야 하는데, 이때 이동 거리를 정해주는 것이 바로 학습률입니다.
- 딥러닝에서 학습률의 값을 적절히 바꾸면서 최적의 학습률을 찾는 것은 중요한 최적화 과정 중 하나입니다.
- 참고로 케라스는 학습률을 자동으로 조절해 줍니다.
- 하지만 딥러닝을 배우려면 학습률의 개념을 알아두는 것이 중요합니다.

학습률

- 다시 말해서, 경사 하강법은 오차의 변화에 따라 이차 함수 그래프를 만들고 적절한 학습률을 설정해 미분 값이 0인 지점을 구하는 것 입니다.
- y 절편 b의 값도 이와 같은 성질을 가지고 있습니다.
- b 값이 너무 크면 오차도 함께 커지고 너무 작아도 오차가 커집니다. 그래서 최적의 b 값을 구할 때 역시 경사 하강법을 사용합니다.

코딩으로 확인하는 경사하강법

- 이제 코딩을 통해 경사 하강법을 실제로 적용하여 a와 b의 값을 구해 보겠습니다.
- 여기서는 텐서플로 라이브러리를 불러와 사용해 보겠습니다.

```
import tensorflow as tf
```

- 텐서플로는 구글이 오픈 소스 라이선스로 공개한 머신러닝 및 딥러닝 전문 라이브러리입니다.
- 우리는 케라스를 텐서플로 기반으로 구동시킬 것이므로 실습하는 컴퓨터에 텐서플로가 설치돼 있어야 합니다.

코딩으로 확인하는 경사하강법

- 데이터 입력과 X,Y를 지정하는 방법은 3장에서 다루었던 코드와 같으며, 여기에 학습률이 추가됩니다.
- 데이터를 입력하는 방법과 x 리스트, y 리스트를 만드는 방법은 앞서 다룬 코드의 방식과 같습니다.

```
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 경사하강법

- 이제 기울기 a와 y 절편 b의 값을 임의로 정하겠습니다.
- 다만, 기울기가 너무 커지거나 작아지면 실행 시간이 불필요하게 늘어나므로 기울기는 0 ~ 10 사이에서, y 절편은 0 ~ 100 사이에서 임의의 값을 얻게끔 합니다.

```
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 경사하강법

- Tensorflow 라이브러리를 tf 라는 약어로 불러오고 변수의 값을 정할 때는 Variable() 함수를 이용합니다.
- random_uniform() 은 임의의 수를 생성해 주는 함수로, 여기에 몇 개의 값을 뽑아낼지와 최솟값 및 최댓값을 적어줍니다.
- 예를 들어 random_uniform([1], 0, 10, ...)은 0에서 10사이에서 임의의 수 1개를 만들라는 뜻입니다.
- 데이터 형식은 실수형 (float64)으로 지정하고, 실행 시 같은 값이 나올 수 있게 seed 값을 설정해 주었습니다.

코딩으로 확인하는 경사하강법

- 이제 일차 방정식 ax + b의 식을 구현해 보겠습니다.

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

- 이어서 평균 제곱근 오차의 식을 구현해 보겠습니다.
- 텐서플로를 이용해 평균 제곱근 오차를 다음과 같이 구현할 수 있습니다.

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

코딩으로 확인하는 경사하강법

- 이제 경사하강법을 실행할 순서입니다.
- 텐서플로는 딥러닝에 최적화된 라이브러리입니다.
- 딥러닝에 반드시 필요한 경사 하강법을 미리 함수로 만들어 놓았습니다.
- 텐서플로의 GradientDescentOptimizer( ) 함수를 이용해 다음과 같이 경사 하강법의 결과를 gradient_decent 에 할당시킬 수 있습니다.

```
Gradient_decent = tf.train.GradientDescentOptimizer(learning_rate).minimize(rmse)
```

- 앞서 지정한 learning_rate와 평균 제곱근 오차를 통해 구한 rmse가 포함된 것을 볼 수 있습니다.

코딩으로 확인하는 경사하강법

- 이제 텐서플로를 실행시키고 결과값을 출력하는 부분을 다음과 같이 작성합니다.

```
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 = %.4f, 기울기 a = %.4f, y 절편 b = %.4f” % (step, sess.run(rmse),
sess.run(a), sess.run(b)))
```

- 텐서플로는 Session 함수를 이용해 구동에 필요한 리소스를 컴퓨터에 할당하고 이를 실행시킬 준비를 합니다.

코딩으로 확인하는 경사하강법

- Session을 통해 구현될 함수를 텐서플로에서는 '그래프'라고 부르며, Session 이 할당되면 Session.run('그래프명')의 형식으로 해당 함수를 구동시킵니다.
- global_variables_initializer() 는 변수를 초기화하는 함수입니다.
- 앞서 만든 gradient_decent를 총 필요한 수만큼 반복하여 실행합니다.
- 그리고 100번마다 RMSE, 기울기, y절편을 출력하게 하였습니다.

코딩으로 확인하는 경사하강법

- 이를 모두 정리하면 다음 코드와 같습니다.

```
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]

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)
```

코딩으로 확인하는 경사하강법

- 이를 모두 정리하면 다음 코드와 같습니다.

```
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)))
```

  - 여기서 에포크(Epoch)는 입력 값에 대해 몇 번이나 반복하여 실험했는지를 나타냅니다.
  - 우리가 설정한 실험을 반복하고 100 번마다 결과를 내놓습니다.

코딩으로 확인하는 경사하강법

- 평균 제곱근 오차(RMSE) 변화와 기울기 a가 2.3에 수렴하는 것 그리고 y 절편 b가 79에 수렴하는 과정을 볼 수 있습니다.
- 기울기 2.3과 y 절편 79는 우리가 앞서 최소제곱법을 통해 미리 확인한 정답값과 같습니다.
- 이렇게 하면 최소 제곱법을 쓰지 않고 평균 제곱근 오차를 구하고, 경사 하강법을 통해 기울기 a나 절편 b 값을 구할 수 있습니다.
- 이와 똑같은 방식을 x가 여러 개인 다중 선형 회귀 에서도 사용합니다.

다중 선형 회귀란?

- 앞서 학생들이 공부한 시간에 따른 예측 직선을 그리고자 기울기 a와 y 절편 b를 구했습니다.
- 그런데 이 예측 직선을 이용해도 실제 성적 사이에는 약간의 오차가 있었습니다
- 4시간 공부한 친구는 88점을 예측했는데 이보다 좋은 93점을 받았고, 6시간 공부한 친구는 93점을 받을 것으로 예측했지만 91 점을 받았습니다.
- 이러한 차이가 생기는 이유는 공부한 시간 이외의 다른 요소가 성적에 영향을 끼쳤기 때문입니다.

다중 선형 회귀란?

- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며, 정보를 추가해 새로운 예측 값을 구하려면 변수의 개수를 늘려 '다중선형 회귀'를 만들어 주어야 합니다
- 예를 들어, 일주일 동안 받는 과외 수업 횟수를 조사해서 이를 기록해 보았습니다.

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

- 그럼 지금부터 두 개의 독립 변수 x_1과 x_2가 생긴 것입니다.

다중 선형 회귀란?

- 이를 사용해 종속 변수 y를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옵니다.

$$y = a_1 x_1 + a_2 x_2 + b$$

- 그러면 두 기울기 a_1과 a_2는 각각 어떻게 구할 수 있을까요?
- 앞에서 배운 경사 하강법을 그대로 적용하면 됩니다.
- 바로 코딩을 통해 확인해 보겠습니다.

코드로 확인해 보는 다중 선형 회귀

- 지금까지 배운 내용을 토대로 다중 선형 회귀를 작성해 보겠습니다.
- 텐서플로를 불러온 뒤 z와 y의 값을 지정하는 과정은 동일합니다.
- 다만, 이번에는 x_1과 x_2 라는 두 개의 독립 변수 리스트를 만들어 줍니다.

```
import tensorflow as tf

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[0] for x_row2 in data]          # 새로 추가되는 값
y_data = [y_row[2] for y_row in data]
```

코드로 확인해 보는 다중 선형 회귀

- 이제 앞서 기울기의 값을 구하는 방식 그대로 또 하나의 기울기 a2를 구합니다.

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

- 이제 새로운 방정식 $y = a_1 * x_1 + a_2 * x_2 + b$ 에 맞춰 다음과 같이 식을 세웁니다.

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

코드로 확인해 보는 다중 선형 회귀

- 나머지 라인은 앞서 배운 선형 회귀와 같습니다.
- 결과를 출력하는 부분만 기울기가 두 개 나올 수 있게 수정합니다.

```
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)))
```

코드로 확인해 보는 다중 선형 회귀

- 이를 정리하면 다음 코드와 같습니다.

```
import tensorflow as tf

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)
```

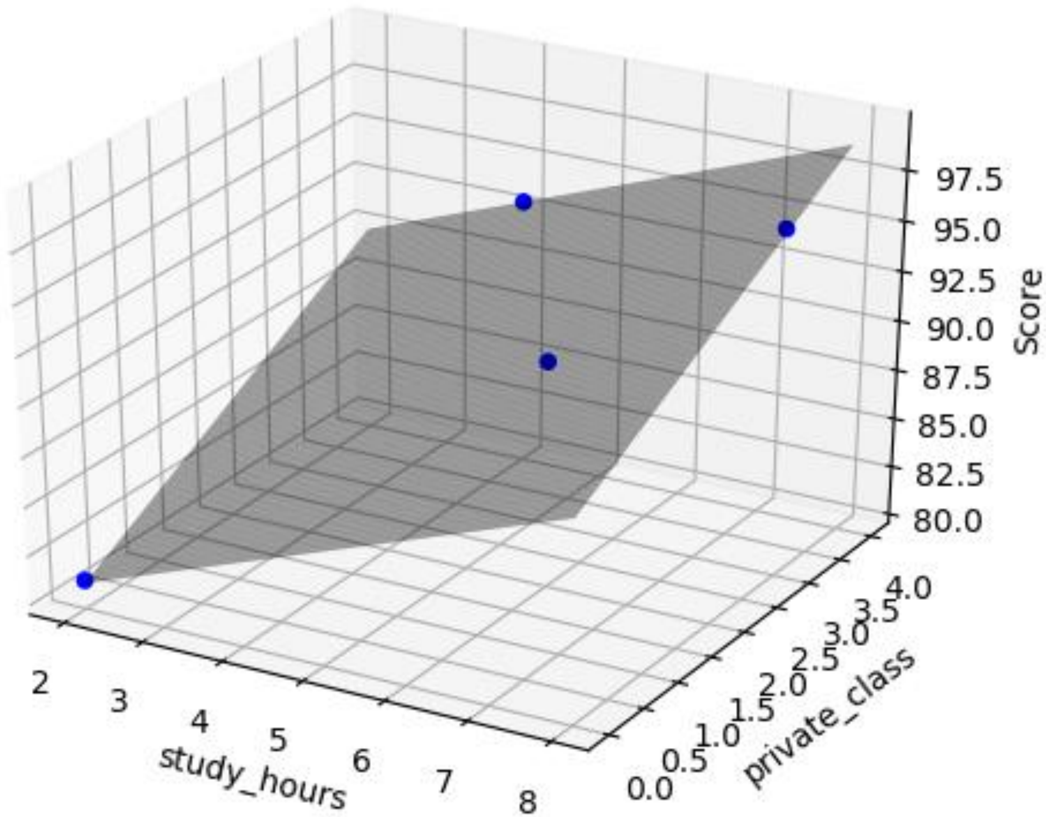
코드로 확인해 보는 다중 선형 회귀

- 이를 정리하면 다음 코드와 같습니다.

```
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)))
```

코드로 확인해 보는 다중 선형 회귀

- 다중 선형 회귀 문제에서의 기울기 a와 절편 b의 값을 구했습니다.
- 참고로 이를 그래프로 표현하면 그림과 같습니다.



코드로 확인해 보는 다중 선형 회귀

- 1차원 예측 직선이 3차원 '예측 평면'으로 바뀌었습니다.
- 과외 수업 횟수라는 새로운 변수가 추가되면서 1차원 직선에서만 움직이던 예측 결과가 더 넓은 평면 범위 안에서 움직이게 되었고, 이로 인해 좀 더 정밀한 예측을 할 수 있게 된 것입니다.

Q&A