



# 둘째마당

## 예측 모델의 기본 원리

# 5장 선형 회귀 모델: 먼저 굿고 수정하기

---

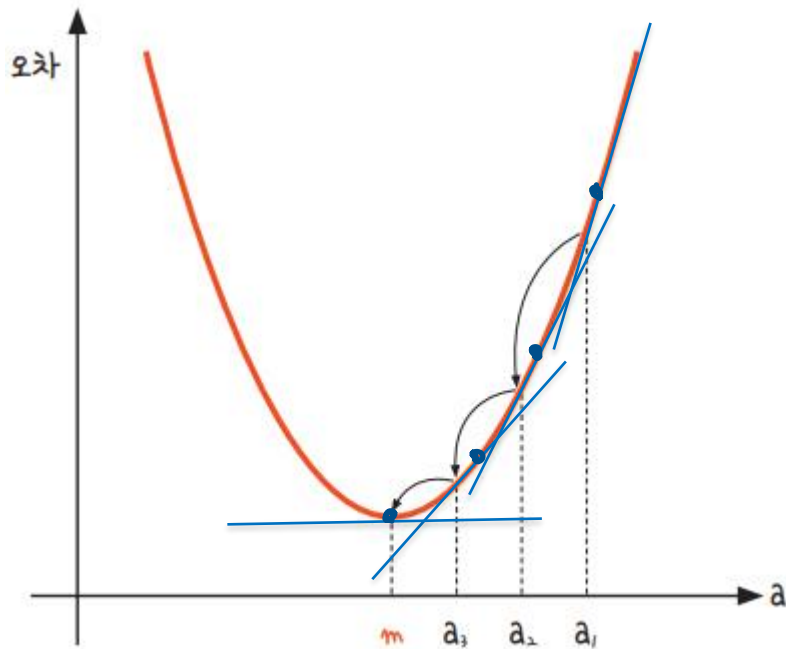
- 1 경사 하강법의 개요
- 2 파이썬 코딩으로 확인하는 선형 회귀
- 3 다중 선형 회귀의 개요
- 4 파이썬 코딩으로 확인하는 다중 선형 회귀
- 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

# 선형 회귀 모델: 먼저 굿고 수정하기

- 선형 회귀 모델: 먼저 굿고 수정하기
  - 우리는 앞서 기울기  $a$ 를 너무 크게 잡으면 오차가 커지는 것을 확인
  - 기울기를 너무 작게 잡아도 오차가 커짐
  - 기울기  $a$ 와 오차 사이에는 이렇게 상관관계가 있음
  - 이때 기울기가 무한대로 커지거나 무한대로 작아지면 그래프는  $y$ 축과 나란한 직선이 됨
  - 오차도 함께 무한대로 커짐
  - 이를 다시 표현하면 기울기  $a$ 와 오차 사이에는 그림 5-1의 빨간색 그래프와 같은 **이차 함수의 관계**가 있다는 의미

# 선형 회귀 모델: 먼저 굿고 수정하기

▼ 그림 5-1 | 기울기  $a$ 와 오차의 관계: 적절한 기울기를 찾았을 때 오차가 최소화된다



# 선형 회귀 모델: 먼저 굿고 수정하기

## ● 선형 회귀 모델: 먼저 굿고 수정하기

- 이 그래프상에서 오차가 가장 작을 때는 언제일까?
- 그래프의 가장 아래쪽 볼록한 부분에 이르렀을 때
- 즉, 기울기  $a$ 가  $m$ 의 위치에 있을 때
- 우리는 앞 장에서 임의의 기울기를 집어넣어 평균 제곱 오차를 구해 보았음
- 그때의 기울기를  $a_1$ 이라고 한다면, 기울기를 적절히 바꾸어  $a_2, a_3$ 으로 이동시키다 결국  $m$ 에 이르게 하면 최적의 기울기를 찾게 되는 것
- 이 작업을 위해  $a_1$  값보다  $a_2$  값이  $m$ 에 더 가깝고,  $a_3$  값이  $a_2$  값보다  $m$ 에 더 가깝다는 것을 컴퓨터가 판단
- 이러한 판단을 하게 하는 방법이 바로 미분 기울기를 이용하는 **경사 하강법(gradient decent)**



# 1 경사 하강법의 개요

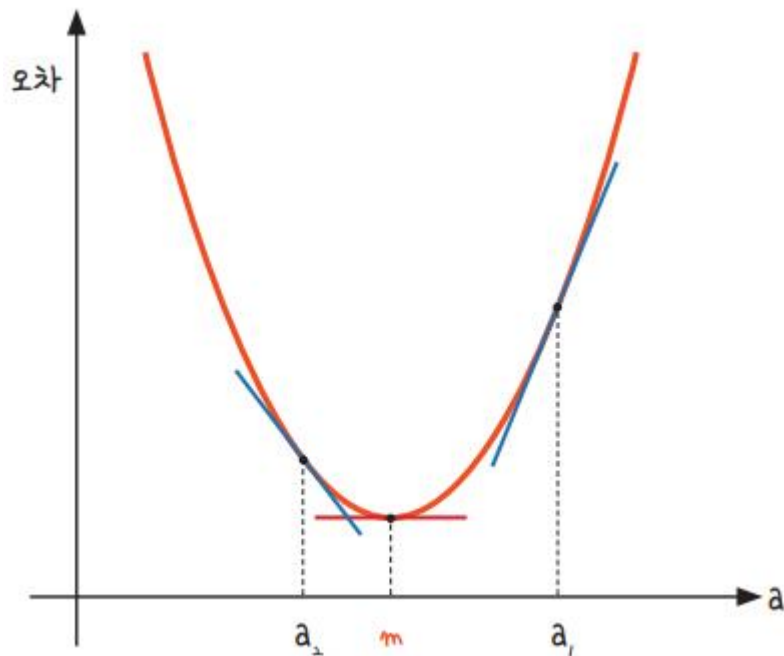
---

# 1 경사 하강법의 개요

## ● 경사 하강법의 개요

- '3장. 딥러닝을 위한 기초 수학'에서 미분은 한 점에서의 순간 기울기라고 배웠음
- $y = x^2$  그래프에서  $x$ 에 다음과 같이  $a_1, a_2$  그리고  $m$ 을 대입해 그 자리에서 미분하면 그림 5-2와 같이 각 점에서의 순간 기울기가 그려짐

▼ 그림 5-2 | 순간 기울기가 0인 점이 곧 우리가 찾는 최솟값  $m$ 이다



# 1 경사 하강법의 개요

- 경사 하강법의 개요

- 여기서 눈여겨보아야 할 것은 우리가 찾는 최솟값  $m$ 에서의 순간 기울기
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는  $x$ 축과 평행한 선이 됨
- 즉, 기울기가 0
- 우리가 할 일은 '미분 값이 0인 지점'을 찾는 것이 됨



# 1 경사 하강법의 개요

- 경사 하강법의 개요

- 이를 위해 다음 과정을 거침

1 |  $a_1$ 에서 미분을 구함

2 | 구한 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킨  $a_2$ 에서 미분을 구함(그림 5-3 참조)

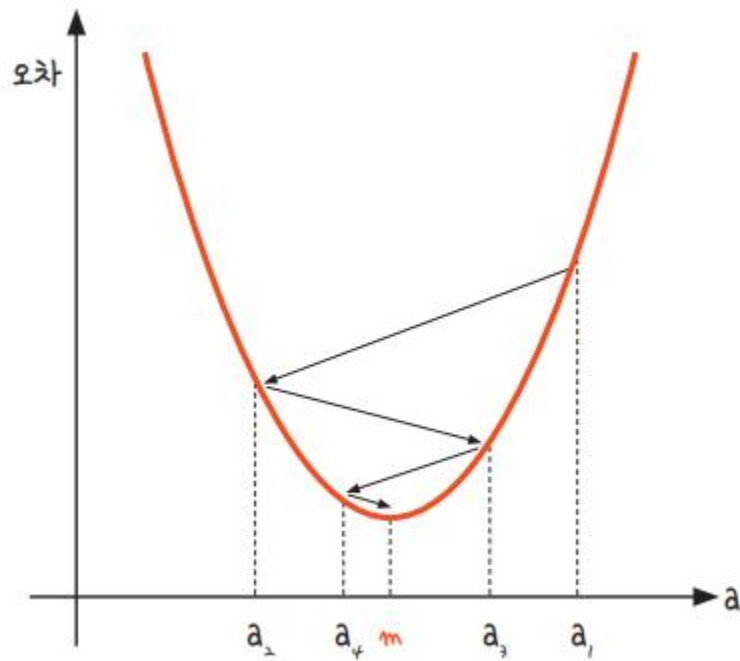
3 | 앞에서 구한 미분 값이 0이 아니면 1과 2 과정을 반복

# 1 경사 하강법의 개요

## ● 경사 하강법의 개요

- 그림 5-3과 같이 기울기가 0인 한 점( $m$ )으로 수렴

### ▼ 그림 5-3 | 최솟점 $m$ 을 찾아가는 과정



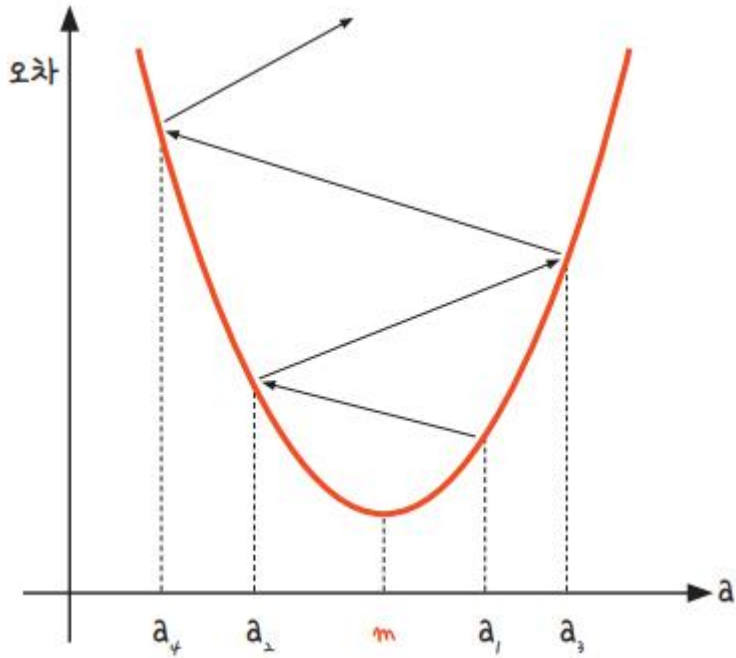
# 1 경사 하강법의 개요

## ● 경사 하강법의 개요

- 경사 하강법은 이렇게 반복적으로 기울기  $a$ 를 변화시켜서  $m$  값을 찾아내는 방법
- 여기서 우리는 **학습률**(learning rate)이라는 개념을 알 수 있음
- 기울기의 부호를 바꾸어 이동시킬 때 적절한 거리를 찾지 못해 너무 멀리 이동시키면  $a$  값이 한 점으로 모이지 않고 그림 5-4와 같이 위로 치솟아 버림

# 1 경사 하강법의 개요

▼ 그림 5-4 | 학습률을 너무 크게 잡으면 한 점으로 수렴하지 않고 발산



# 1 경사 하강법의 개요

## ● 경사 하강법의 개요

- 어느 만큼 이동시킬지 신중히 결정해야 하는데, 이때 이동 거리를 정해 주는 것이 바로 학습률
- 딥러닝에서 학습률의 값을 적절히 바꾸면서 최적의 학습률을 찾는 것은 중요한 최적화 과정 중 하나
- 다시 말해 **경사 하강법**은 오차의 변화에 따라 **이차 함수 그래프**를 만들고 적절한 **학습률**을 설정해 **미분 값이 0인 지점**을 구하는 것
- $y$  절편  $b$ 의 값도 이와 같은 성질을 가지고 있음
- $b$  값이 너무 크면 오차도 함께 커지고, 너무 작아도 오차가 커짐
- 최적의  $b$  값을 구할 때 역시 경사 하강법을 사용



## 2 파이썬 코딩으로 확인하는 선형 회귀

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀
  - 지금까지 내용을 파이썬 코드로 옮겨 볼 차례
  - 먼저 평균 제곱 오차의 식을 다시 가져와 보자

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

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

- 여기서  $\hat{y}_i$  는  $y = ax + b$ 의 식에  $x_i$ 를 집어넣었을 때 값이므로  $y_i = ax_i + b$ 를 대입하면 다음과 같이 바뀜

$$\frac{1}{n} \sum (y_i - (ax_i + b))^2$$



## 2 파이썬 코딩으로 확인하는 선형 회귀

### ● 파이썬 코딩으로 확인하는 선형 회귀

- 이 값을 미분할 때 우리가 궁금한 것은  $a$ 와  $b$ 라는 것을 기억해야 함
- 식 전체를 미분하는 것이 아니라 필요한 값을 중심으로 미분해야 하기 때문임
- 이렇게 특정한 값, 예를 들어  $a$ 와  $b$ 를 중심으로 미분할 때 이를  $a$ 와  $b$ 로 '편미분한다'고 하는 것을 '3장. 딥러닝을 위한 기초 수학'에서 배웠음

- $a$ 로 편미분한 결과  $= \frac{2}{n} \sum -x_i(y_i - (ax_i + b))$  면 다음과 같음

$$b \text{로 편미분한 결과} = \frac{2}{n} \sum -(y_i - (ax_i + b))$$

$$E = \frac{1}{n} \sum (y_i - (ax_i + b))^2$$

- '3장. 딥러닝을 위한 기초 수학'에서 설명한 '미분의 기본 공식 2~5'를 사용해 편미분했음

# 편미분으로 $a, b$ 구하기

## 미분의 기본 공식

1 |  $f(x) = x$ 일 때  $f'(x) = 1$

2 |  $f(x) = a$ 에서  $a$ 가 상수일 때  $f'(x) = 0$

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

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

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



$$* E = f(g(x)) = f(y_i - (ax_i + b)) = \frac{1}{n} \sum (y_i - (ax_i + b))^2$$

$$* \frac{\partial E}{\partial a} = \frac{1}{n} \sum 2(y_i - (ax_i + b)) \cdot (-x_i) = \frac{2}{n} \sum -x_i(y_i - (ax_i + b))$$

$$\frac{\partial E}{\partial b} = \frac{1}{n} \sum 2(y_i - (ax_i + b)) \cdot (-1) = \frac{2}{n} \sum -(y_i - (ax_i + b))$$

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀
  - 이를 각각 파이썬 코드로 바꾸면 다음과 같음

```
y_pred = a * x + b          # 예측 값을 구하는 식입니다.  
error = y - y_pred          # 실제 값과 비교한 오차를 error로 놓습니다.  
  
a_diff = (2/n) * sum(-x * (error)) # 오차 함수를 a로 편미분한 값입니다.  
b_diff = (2/n) * sum(-(error))    # 오차 함수를 b로 편미분한 값입니다.
```

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀
  - 여기에 학습률을 곱해 기존의 a 값과 b 값을 업데이트

```
lr = 0.03          # 학습률을 정합니다.  
a = a - lr * a_diff # 학습률을 곱해 기존의 a 값을 업데이트합니다.  
b = b - lr * b_diff # 학습률을 곱해 기존의 b 값을 업데이트합니다.
```



## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

학습률 0.03은 어떻게 정했나요?

- 여러 학습률을 적용해 보며 최적의 결과를 만드는 학습률을 찾아낸 것
- 최적의 학습률은 데이터와 딥러닝 모델에 따라 다르므로 그때그때 찾아내야 함
- 앞으로 배우게 될 딥러닝 프로젝트에서는 자동으로 최적의 학습률을 찾아 주는 최적화 알고리즘들을 사용

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀
  - 나머지는 앞서 공부한 바와 같음
  - 중간 과정을 그래프로 표현하는 코드를 넣어 모두 정리하면 다음과 같이 코드가 완성



## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

- 이 실습에는 그래프 관련 부분을 실행하기 위해 matplotlib 라이브러리가 필요함
- 코랩의 경우에는 기본으로 제공하지만 주피터 노트북을 이용해 실습 중이라면 다음 명령으로 라이브러리를 설치

!pip install matplotlib

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

### 실습 | 선형 회귀 모델 실습



```
import numpy as np
import matplotlib.pyplot as plt

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

# 데이터의 분포를 그래프로 나타냅니다.
plt.scatter(x, y)
plt.show()
```



## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

```
# 기울기 a의 값과 절편 b의 값을 초기화합니다.
```

```
a = 0
```

```
b = 0
```

```
# 학습률을 정합니다.
```

```
lr = 0.03
```

```
# 몇 번 반복될지 설정합니다.
```

```
epochs = 2001
```

```
# x 값이 총 몇 개인지 셉니다.
```

```
n = len(x)
```

## 2 파이썬 코딩으로 확인하는 선형 회귀

### ● 파이썬 코딩으로 확인하는 선형 회귀

```
# 경사 하강법을 시작합니다.
for i in range(epochs):
    y_pred = a * x + b
    error = y - y_pred

    # 에포크 수만큼 반복합니다.
    # 예측 값을 구하는 식입니다.
    # 실제 값과 비교한 오차를 error로 놓습니다.

    a_diff = (2/n) * sum(-x * (error)) # 오차 함수를 a로 편미분한 값입니다.
    b_diff = (2/n) * sum(-(error))     # 오차 함수를 b로 편미분한 값입니다.

    a = a - lr * a_diff # 학습률을 곱해 기존의 a 값을 업데이트합니다.
    b = b - lr * b_diff # 학습률을 곱해 기존의 b 값을 업데이트합니다.
```

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

```
if i % 100 == 0:      # 100번 반복될 때마다 현재의 a 값, b 값을 출력합니다.
    print("epoch=%.f, 기울기=%.04f, 절편=%.04f" % (i, a, b))

# 앞서 구한 최종 a 값을 기울기, b 값을 y 절편에 대입해 그래프를 그립니다.
y_pred = a * x + b

# 그래프를 출력합니다.
plt.scatter(x, y)
plt.plot(x, y_pred, 'r')
plt.show()
```

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

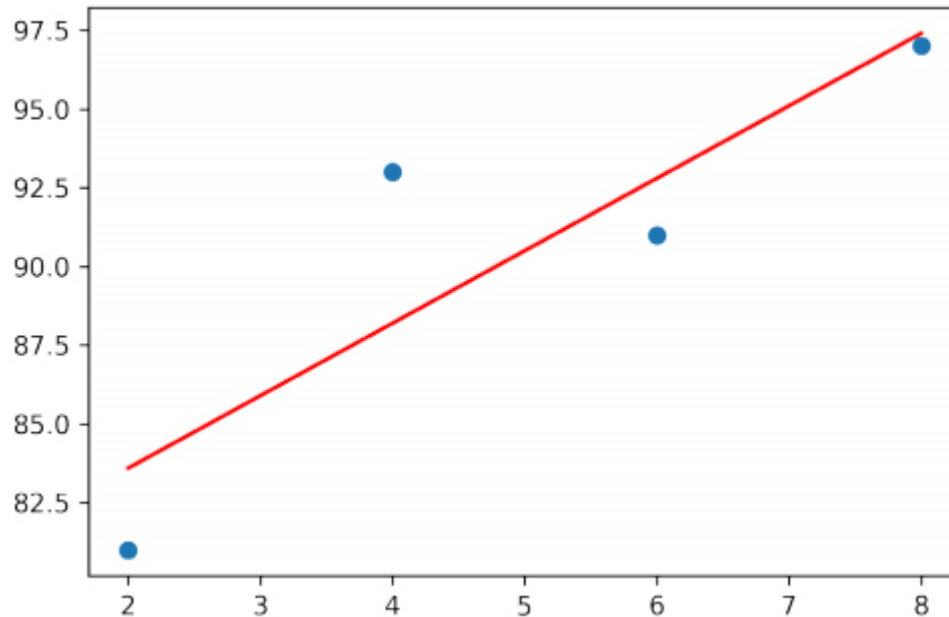
### 실행 결과

```
epoch=0, 기울기=27.8400, 절편=5.4300  
epoch=100, 기울기=7.0739, 절편=50.5117  
epoch=200, 기울기=4.0960, 절편=68.2822  
... (중략) ...  
epoch=1900, 기울기=2.3000, 절편=79.0000  
epoch=2000, 기울기=2.3000, 절편=79.0000
```

## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀

▼ 그림 5-5 | 그래프로 표현한 모습





## 2 파이썬 코딩으로 확인하는 선형 회귀

- 파이썬 코딩으로 확인하는 선형 회귀
  - 여기서 에포크(epoch)는 입력 값에 대해 몇 번이나 반복해서 실험했는지 나타냄
  - 우리가 설정한 실험을 반복하고 100번마다 결과를 내놓음

## 2 파이썬 코딩으로 확인하는 선형 회귀

### ● 파이썬 코딩으로 확인하는 선형 회귀

- 기울기  $a$ 의 값이 2.3에 수렴하는 것과  $y$  절편  $b$ 의 값이 79에 수렴하는 과정을 볼 수 있음
- 기울기 2.3과  $y$  절편 79는 앞서 우리가 최소 제곱법을 이용해 미리 확인한 값과 같음
- 이렇게 해서 최소 제곱법을 쓰지 않고 평균 제곱 오차와 경사 하강법을 이용해 원하는 값을 구할 수 있었음
- 이와 똑같은 방식을  $x$ 가 여러 개인 다중 선형 회귀에서도 사용



### 3 다중 선형 회귀의 개요

---

\* 특성(feature)이 여러개  
 $x_1, x_2, x_3, \dots$



### 3 다중 선형 회귀의 개요

#### ● 다중 선형 회귀의 개요

- 앞서 학생들이 공부한 시간에 따른 예측 직선을 그리고자 기울기  $a$ 와  $y$  절편  $b$ 를 구함
- 이 예측 직선을 이용해도 실제 성적 사이에는 약간의 오차가 있었음
- 4시간 공부한 친구는 88점을 예측했는데 이보다 좋은 93점을 받았고, 6시간 공부한 친구는 93점을 받을 것으로 예측했지만 91점을 받았음
- 이러한 차이가 생기는 이유는 공부한 시간 이외의 다른 요소가 성적에 영향을 끼쳤기 때문임
- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며, 정보를 추가해 새로운 예측 값을 구하려면 변수 개수를 늘려 **다중 선형 회귀**를 만들어 주어야 함

### 3 다중 선형 회귀의 개요

- 다중 선형 회귀의 개요

- 예를 들어 일주일 동안 받는 과외 수업 횟수를 조사해서 이를 기록해 보았음

▼ 표 5-1 | 공부한 시간, 과외 수업 횟수에 따른 성적 데이터

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

## 3 다중 선형 회귀의 개요

### ● 다중 선형 회귀의 개요

- 그럼 지금부터 독립 변수  $x_1$ 과  $x_2$ 가 두 개 생긴 것
- 이를 사용해 종속 변수  $y$ 를 만들 경우 기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴

$$y = a_1x_1 + a_2x_2 + b$$

- 두 기울기  $a_1$ 과  $a_2$ 는 각각 어떻게 구할 수 있을까?
- 앞서 배운 경사 하강법을 그대로 적용
- 바로 파이썬 코드로 확인해 보자



## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

- 지금까지 배운 내용을 토대로 다중 선형 회귀를 만들어 보자
- 이번에는  $x$  값이 두 개이므로 다음과 같이 공부 시간  $x_1$ , 과외 시간  $x_2$ , 성적  $y$ 의 넘파이 배열을 만듦

```
x1 = np.array([2, 4, 6, 8])  
x2 = np.array([0, 4, 2, 3])  
y = np.array([81, 93, 91, 97])
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

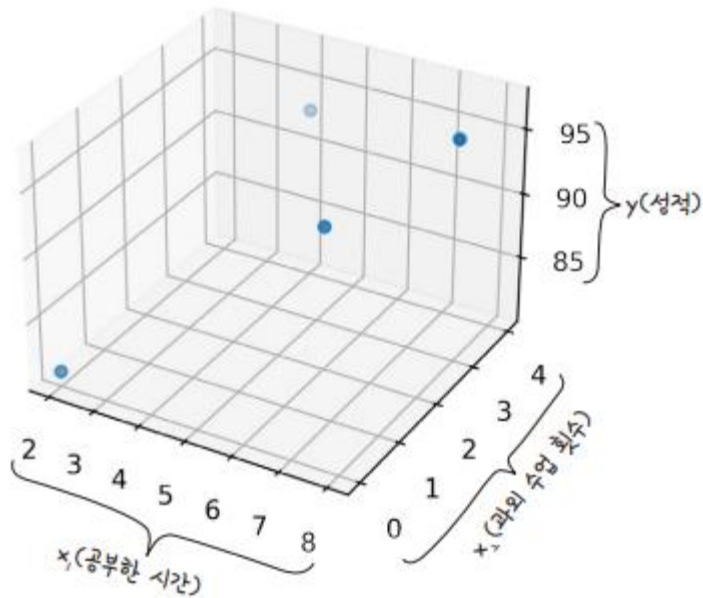
- 파이썬 코딩으로 확인하는 다중 선형 회귀
  - 데이터의 분포를 그래프로 표현해 보면 다음과 같음

```
fig = plt.figure()  
ax = fig.add_subplot(111, projection='3d')  
ax.scatter3D(x1, x2, y);  
plt.show()
```

row, col, index

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

▼ 그림 5-6 | 축이 하나 더 늘어 3D로 배치된 모습



## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

- 앞서  $x$ 와  $y$  두 개의 축이던 것과는 달리  $x_1, x_2, y$  이렇게 세 개의 축이 필요함
- 새로운 변수가 추가되면 차원이 하나씩 추가되면서 계산은 더욱 복잡해지는 것을 알 수 있음



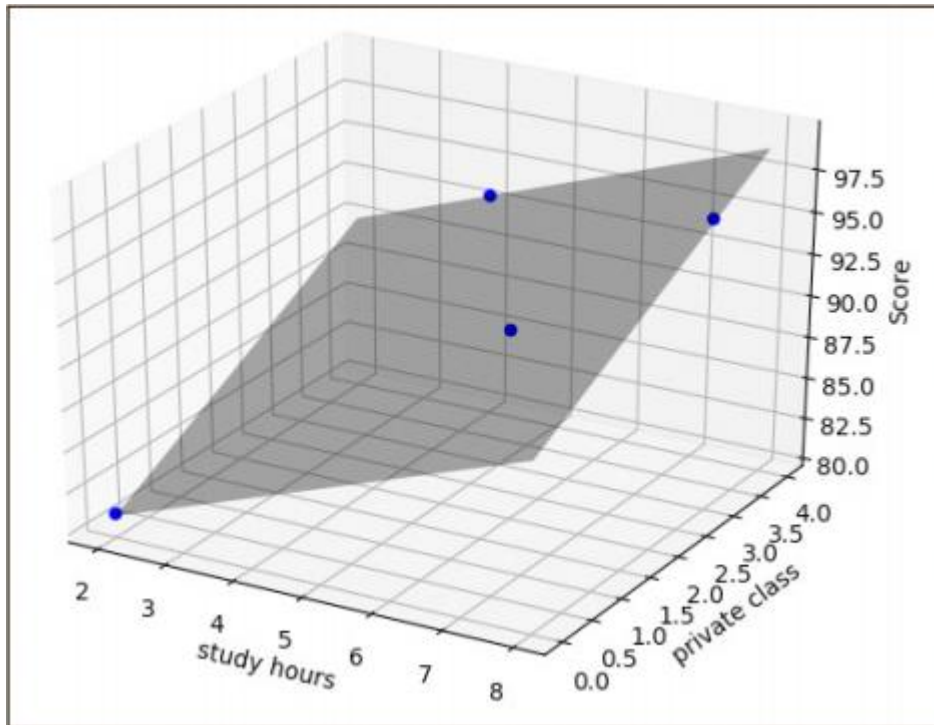
## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

- 앞서 선형 회귀는 선을 긋는 작업이라고 했음
- 그러면 다중 선형 회귀는 어떨까?
- 최적의 결과를 찾은 후 이를 그래프로 표현하면 그림 5-7과 같이 평면으로 표시

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ▼ 그림 5-7 | 다중 선형 회귀



## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

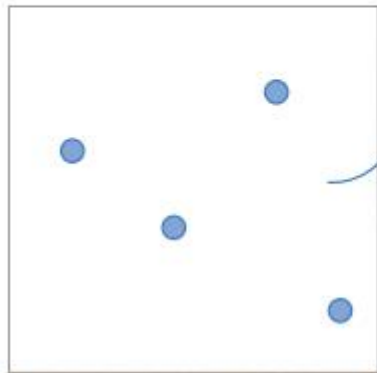
### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

- 직선상에서 예측하던 것이 평면으로 범위가 넓어지므로 계산이 복잡해지고 더 많은 데이터를 필요로 하게 됨

단순 선형 회귀



다중 선형 회귀



빈 공간이 많아진다.  
→ 계산이 복잡하다.  
→ 더 많은 데이터가 필요하다.

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

- 코드의 형태는 크게 다르지 않음
- 다만  $x$ 가 두 개가 되었으므로  $x_1, x_2$  두 변수를 만들고, 기울기도  $a_1$ 과  $a_2$  이렇게 두 개를 만듦
- 앞서 했던 방법대로 경사 하강법을 적용해 보자
- 먼저 예측 값을 구하는 식을 다음과 같이 세움

```
y_pred = a1 * x1 + a2 * x2 + b # 기울기와 절편 자리에 a1, a2, b를 각각 대입합니다.  
error = y - y_pred             # 실제 값과 비교한 오차를 error로 놓습니다.
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

- 오차 함수를  $a_1$ ,  $a_2$ ,  $b$ 로 각각 편미분한 값을  $a1\_diff$ ,  $a2\_diff$ ,  $b\_diff$ 라고 할 때

```
n = len(x1)                # 변수의 총 개수입니다.
a1_diff = (2/n) * sum(-x1 * (error)) # 오차 함수를 a1로 편미분한 값입니다.
a2_diff = (2/n) * sum(-x2 * (error)) # 오차 함수를 a2로 편미분한 값입니다.
b_diff = (2/n) * sum(-(error))      # 오차 함수를 b로 편미분한 값입니다.
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

- 학습률을 곱해 기존의 기울기와 절편을 업데이트한 값을 구함

```
a1 = a1 - lr * a1_diff    # 학습률을 곱해 기존의 a1 값을 업데이트합니다.  
a2 = a2 - lr * a2_diff    # 학습률을 곱해 기존의 a2 값을 업데이트합니다.  
b = b - lr * b_diff       # 학습률을 곱해 기존의 b 값을 업데이트합니다.
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

- 이제 실제 점수와 예측된 점수를 출력해서 예측이 잘되는지 확인

```
print("실제 점수: ", y)  
print("예측 점수: ", y_pred)
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀
  - 지금까지 코드를 정리하면 다음과 같음

### 실습 | 파이썬 코딩으로 확인하는 다중 선형 회귀



```
import numpy as np
import matplotlib.pyplot as plt

# 공부 시간 x1과 과외 시간 x2, 성적 y의 넘파이 배열을 만듭니다.
x1 = np.array([2, 4, 6, 8])
x2 = np.array([0, 4, 2, 3])
y = np.array([81, 93, 91, 97])
```



## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

```
# 데이터의 분포를 그래프로 나타냅니다.  
fig = plt.figure()  
ax = fig.add_subplot(111, projection='3d')  
ax.scatter3D(x1, x2, y);  
plt.show()  
  
# 기울기 a의 값과 절편 b의 값을 초기화합니다.  
a1 = 0  
a2 = 0  
b = 0  
  
# 학습률을 정합니다.  
lr = 0.01
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

```
# 몇 번 반복될지 설정합니다.  
epochs = 2001  
  
# x 값이 총 몇 개인지 셉니다. x1과 x2의 수가 같으므로 x1만 세겠습니다.  
n = len(x1)  
  
# 경사 하강법을 시작합니다.  
for i in range(epochs):  
    # 에포크 수만큼 반복합니다.  
  
    y_pred = a1 * x1 + a2 * x2 + b # 예측 값을 구하는 식을 세웁니다.  
    error = y - y_pred             # 실제 값과 비교한 오차를 error로 놓습니다.
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

```

a1_diff = (2/n) * sum(-x1 * (error)) # 오차 함수를 a1로 편미분한 값입니다.
a2_diff = (2/n) * sum(-x2 * (error)) # 오차 함수를 a2로 편미분한 값입니다.
b_diff = (2/n) * sum(-(error))      # 오차 함수를 b로 편미분한 값입니다.

a1 = a1 - lr * a1_diff               # 학습률을 곱해 기존의 a1 값을 업데이트합니다.
a2 = a2 - lr * a2_diff               # 학습률을 곱해 기존의 a2 값을 업데이트합니다.
b = b - lr * b_diff                  # 학습률을 곱해 기존의 b 값을 업데이트합니다.

if i % 100 == 0: # 100번 반복될 때마다 현재의 a1, a2, b의 값을 출력합니다.
    print("epoch=%.f, 기울기1=%.04f, 기울기2=%.04f, 절편=%.04f" % (i, a1,
a2, b))

```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

```
# 실제 점수와 예측된 점수를 출력합니다.  
print("실제 점수: ", y)  
print("예측 점수: ", y_pred)
```

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

### ● 파이썬 코딩으로 확인하는 다중 선형 회귀

#### 실행 결과

... (전략) ...

epoch=1700, 기울기1=1.5496, 기울기2=2.3028, 절편=77.5168

epoch=1800, 기울기1=1.5361, 기울기2=2.2982, 절편=77.6095

epoch=1900, 기울기1=1.5263, 기울기2=2.2948, 절편=77.6769

epoch=2000, 기울기1=1.5191, 기울기2=2.2923, 절편=77.7260

실제 점수: [81 93 91 97]

예측 점수: [80.76387645 92.97153922 91.42520875 96.7558749]

## 4 파이썬 코딩으로 확인하는 다중 선형 회귀

- 파이썬 코딩으로 확인하는 다중 선형 회귀

- 2,000번 반복했을 때 최적의 기울기  $a_1$ 과  $a_2$  및 절편을 찾아가며 실제 점수에 가까운 예측 값을 만들어 내고 있음을 알 수 있음



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

---



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 우리는 머신 러닝의 기본인 선형 회귀에 대해 배우고 있음
- 우리 목표는 딥러닝
- 2장에서 잠시 살펴보았지만, 앞으로 우리는 딥러닝을 실행하기 위해 텐서플로라는 라이브러리의 케라스 API를 불러와 사용할 것
- 지금까지 배운 선형 회귀의 개념과 딥러닝 라이브러리들이 어떻게 연결되는지 살펴볼 필요가 있음
- 이를 통해 텐서플로 및 케라스의 사용법을 익히는 것은 물론이고 딥러닝 자체에 대한 학습도 한걸음 더 나가게 될 것



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 선형 회귀는 현상을 분석하는 방법의 하나
- 머신 러닝은 이러한 분석 방법을 이용해 예측 모델을 만드는 것
- 두 분야에서 사용하는 용어가 약간 다름
- 예를 들어 함수  $y = ax + b$ 는 공부한 시간과 성적의 관계를 유추하기 위해 필요했던 식
- 이렇게 문제를 해결하기 위해 가정하는 식을 머신 러닝에서는 가설 함수(hypothesis)라고 하며  $H(x)$ 라고 표기
- 또 기울기  $a$ 는 변수  $x$ 에 어느 정도의 가중치를 곱하는지 결정하므로, **가중치**(weight)라고 하며, **w**로 표시
- 절편  $b$ 는 데이터의 특성에 따라 따로 부여되는 값이므로 **편향**(bias)이라고 하며, **b**로 표시
- 우리가 앞서 배운  $y = ax + b$ 는 머신 러닝에서 다음과 같이 표기

$$y = ax + b \rightarrow H(x) = wx + b$$



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 또한, 평균 제곱 오차처럼 실제 값과 예측 값 사이의 오차에 대한 식을  
손실 함수(**loss function**)라고 함

평균 제곱 오차 → 손실 함수(loss function)



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 최적의 기울기와 절편을 찾기 위해 앞서 경사 하강법을 배웠음
- 딥러닝에서는 이를 **옵티마이저(optimizer)**라고 함
- 우리가 사용했던 경사 하강법은 딥러닝에서 사용하는 여러 옵티마이저 중 하나였음

경사 하강법 → 옵티마이저(optimizer)



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 이제부터는 손실 함수, 옵티마이저라는 용어를 사용해 설명
- 먼저 텐서플로에 포함된 케라스 API 중 필요한 함수들을 다음과 같이 불러옴

```
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델
  - Sequential() 함수와 Dense() 함수는 2장에서 이미 소개한 바 있음
  - 이 함수를 불러와 선형 회귀를 실행하는 코드는 다음과 같음

```
model.add(Dense(1, input_dim=1, activation='linear')) ---- ①  
model.compile(optimizer='sgd', loss='mse') ---- ②  
model.fit(x, y, epochs=2000) ---- ③
```

## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 단 세 줄의 코드에 앞서 공부한 모든 것이 담겨 있음
- 어떻게 설정하는지 살펴보자
- ① 먼저 가설 함수는  $H(x) = wx + b$
- 이때 출력되는 값(=성적)이 하나씩이므로 Dense() 함수의 첫 번째 인자에 1이라고 설정
- 입력될 변수(=학습 시간)도 하나뿐이므로 input\_dim 역시 1이라고 설정
- 입력된 값을 다음 층으로 넘길 때 각 값을 어떻게 처리할지를 결정하는 함수를 **활성화 함수**라고 함
- activation은 활성화 함수를 정하는 옵션
- 여기에서는 **선형 회귀**를 다루고 있으므로 'linear'라고 적어 주면 됨
- 딥러닝 목적에 따라 다른 활성화 함수를 넣을 수 있는데, 예를 들어 다음 절에서 배울 **시그모이드 함수**가 필요하다면 'sigmoid'라고 넣어 주는 식



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델
  - ② 앞서 배운 경사 하강법을 실행하려면 옵티마이저에 `sgd`라고 설정
  - 손실 함수는 평균 제곱 오차를 사용할 것이므로 `mse`라고 설정
  - ③ 끝으로 앞서 따로 적어 주었던 `epochs` 숫자를 `model.fit()` 함수에 적음



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 학습 시간(x)이 입력되었을 때의 예측 점수는 `model.predict(x)`로 알 수 있음
- 예측 점수로 그래프를 그려 보면 다음과 같음

```
plt.scatter(x, y)
plt.plot(x, model.predict(x), 'r')    # 예측 결과를 그래프로 나타냅니다.
plt.show()
```





## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델
  - 이제 모든 코드를 모아 보면 다음과 같음

### 실습 | 텐서플로에서 실행하는 선형 회귀



```
import numpy as np
import matplotlib.pyplot as plt

# 텐서플로의 케라스 API에서 필요한 함수들을 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])

model = Sequential()
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

```
# 출력 값, 입력 변수, 분석 방법에 맞게끔 모델을 설정합니다.  
model.add(Dense(1, input_dim=1, activation='linear'))  
  
# 오차 수정을 위해 경사 하강법(sgd)을, 오차의 정도를 판단하기 위해  
# 평균 제곱 오차(mse)를 사용합니다.  
model.compile(optimizer='sgd', loss='mse')  
  
# 오차를 최소화하는 과정을 2000번 반복합니다.  
model.fit(x, y, epochs=2000)  
  
plt.scatter(x, y)  
plt.plot(x, model.predict(x), 'r')    # 예측 결과를 그래프로 나타냅니다.  
plt.show()
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

```
# 임의의 시간을 집어넣어 점수를 예측하는 모델을 테스트해 보겠습니다.  
hour = 7  
prediction = model.predict([hour])  
print("%.f시간을 공부할 경우의 예상 점수는 %.02f점입니다." % (hour, prediction))
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### 실행 결과

Epoch 1/2000

1/1 [=====] - 1s 114ms/step - loss: 9241.3984

... (중략) ...

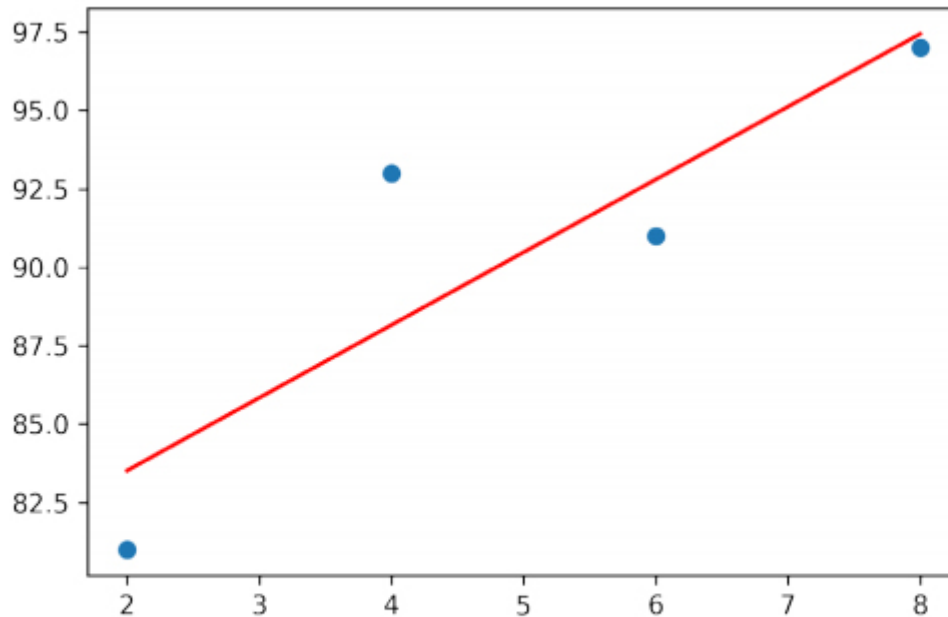
Epoch 2000/2000

1/1 [=====] - 0s 2ms/step - loss: 8.3022

## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

▼ 그림 5-8 | 텐서플로로 실행한 선형 회귀 분석 결과



7시간을 공부할 경우의 예상 점수는 95.12점입니다.



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 앞서 구한 선형 회귀 결과와 같은 그래프를 구했음
- 임의의 시간을 넣었을 때 예상되는 점수를 보여 줌
- 마찬가지로 다중 선형 회귀 역시 텐서플로를 이용해서 실행해 보자
- 앞서 실행했던 내용과 거의 유사함
- 다만, 입력해야 하는 변수가 한 개에서 두 개로 늘었음
- 이 부분을 적용하려면 input\_dim 부분을 2로 변경해 줌

```
model.add(Dense(1, input_dim=2, activation='linear'))
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### ● 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 변수가 두 개이므로, 모델의 테스트를 위해서도 변수를 두 개 입력해야 함
- 임의의 학습 시간과 과외 시간을 입력했을 때의 점수는 다음과 같이 설정해서 구함

```
hour = 7
private_class = 4
prediction = model.predict([[hour, private_class]])

print("%.f시간을 공부하고 %.f시간의 과외를 받을 경우, 예상 점수는 %.02f점입니다."
      % (hour, private_class, prediction))
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델
  - 모든 코드를 정리하면 다음과 같음

### 실습 I 텐서플로에서 실행하는 다중 선형 회귀



```
import numpy as np
import matplotlib.pyplot as plt

# 텐서플로의 케라스 API에서 필요한 함수들을 불러옵니다.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```





## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

```
x = np.array([[2, 0], [4, 4], [6, 2], [8, 3]])
```

```
y = np.array([81, 93, 91, 97])
```

```
model = Sequential()
```

```
# 입력 변수가 두 개(학습 시간, 과외 시간)이므로 input_dim에 2를 입력합니다.
```

```
model.add(Dense(1, input_dim=2, activation='linear'))
```

```
model.compile(optimizer='sgd', loss='mse')
```

```
model.fit(x, y, epochs=2000)
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

```
# 임의의 학습 시간과 과외 시간을 집어넣어 점수를 예측하는 모델을 테스트해 보겠습니다.  
hour = 7  
private_class = 4  
prediction = model.predict([[hour, private_class]])  
  
print("%.f시간을 공부하고 %.f시간의 과외를 받을 경우, 예상 점수는 %.02f점입니다."  
      % (hour, private_class, prediction))
```



## 5 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

- 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

### 실행 결과

Epoch 1/2000

1/1 [=====] - 0s 119ms/step - loss: 8184.9204

... (중략) ...

Epoch 2000/2000

1/1 [=====] - 0s 2ms/step - loss: 0.0743

7시간을 공부하고 4시간의 과외를 받을 경우, 예상 점수는 97.53점입니다.

- 다중 선형 회귀 모델을 통해 임의의 학습 시간과 과외 시간을 입력했을 때의 예상 점수를 확인할 수 있었음