3장 가장 훌륭한 예측선 긋기

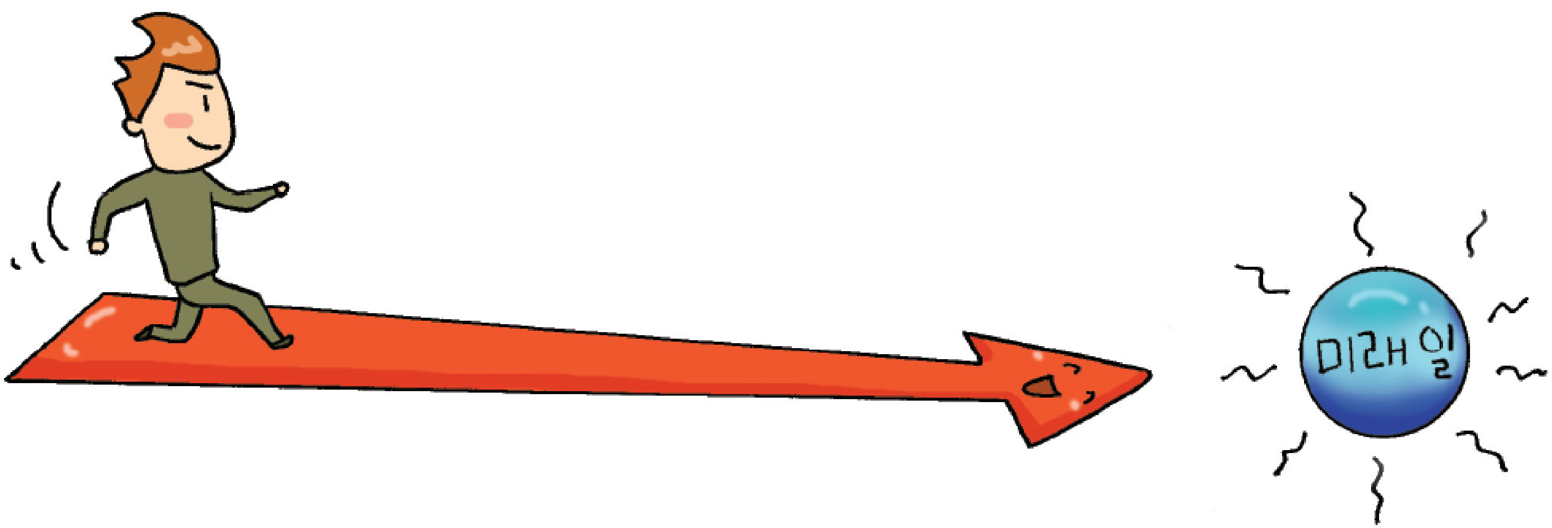
-선형 회귀

목차

선형회귀

- 딥러닝을 이해하려면 딥러닝의 가장 말단에서 이루어지는 가장 기본적인 두 가지 계산 원리를 알아야 함

  → 바로 선형 회귀와 로지스틱 회귀임



1 선형 회귀의 정의

- 독립 변수 :

  ‘x값이 변함에 따라 y값도 변한다’는 이 정의 안에서, 독립적으로 변할 수 있는 x값

- 종속 변수 :

  독립 변수에 따라 종속적으로 변하는 값

- 선형 회귀 :

  독립 변수 x를 사용해 종속 변수 y의 움직임을 예측하고 설명하는 작업을 말함

1 선형 회귀의 정의

- 단순 선형 회귀(simple linear regression) :

  하나의 x값 만으로도 y값을 설명할 수 있을 때

- 다중 선형 회귀(multiple linear regression) :

  x값이 여러 개 필요할 때

2 가장 훌륭한 예측선이란?

- 우선 독립 변수가 하나뿐인 단순 선형 회귀의 예를 공부해 보자

표 3-1 공부한 시간과 중간고사 성적 데이터

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

- 여기서 공부한 시간을 x라 하고 성적을 y라 할 때 집합 X와 집합 Y를 다음과 같이 표현할 수 있음

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

2 가장 훌륭한 예측선이란?

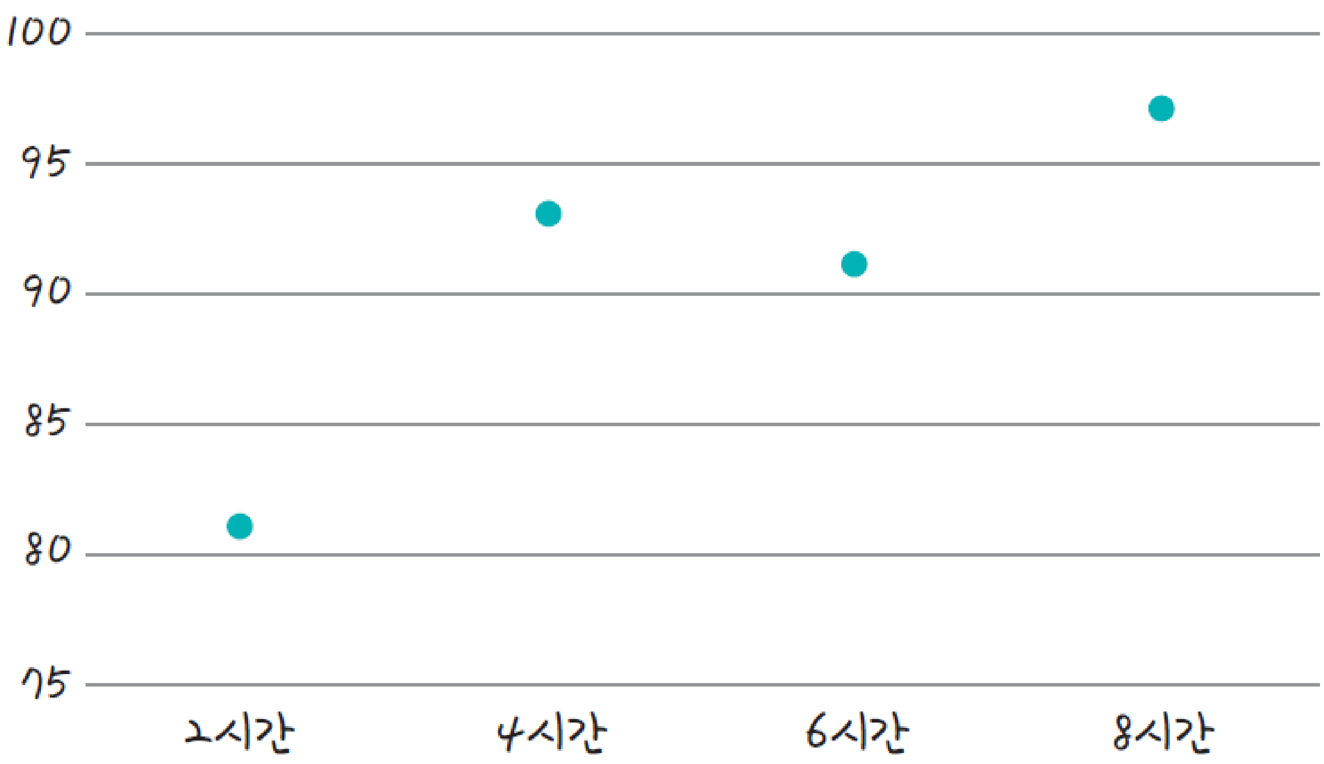


그림 3-1 공부한 시간과 성적을 좌표로 표현

2 가장 훌륭한 예측선이란?

- 선형 회귀를 공부하는 과정은 이 점들의 특징을 가장 잘 나타내는 선을 그리는 과정과 일치함
- 여기에서 선은 직선이므로 곧 일차 함수 그래프임

$$y = ax + b$$

- 여기서 x 값은 독립 변수이고 y값은 종속 변수임
- 즉, x 값에 따라 y 값은 반드시 달라짐
- 다만, 정확하게 계산하려면 상수 a 와 b의 값을 알아야 함
- 이 직선을 훌륭하게 그으려면 직선의 기울기 a 값과 y 절편 b값을 정확히 예측해 내야 함

3 최소 제곱법

- 최소 제곱법(method of least squares)이라는 공식을 알고 적용한다면, 이를 통해 일차 함수의 기울기 a와 y절편 b를 바로 구할 수 있음
- 지금 가진 정보가 x값(입력 값, 여기서는 ‘공부한 시간’)과 y값(출력 값, 여기서는 ‘성적’)일 때 이를 이용해 기울기 a를 구하는 방법은 다음과 같음

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

→ 이것이 바로 최소 제곱법임

3 최소 제곱법

- x의 편차(각 값과 평균과의 차이)를 제곱해서 합한 값을 분모로 놓고, x 와 y의 편차를 곱해서 합한 값을 분자로 놓으면 기울기가 나온다는 뜻임
  - 공부한 시간(x) 평균: $(2 + 4 + 6 + 8) \div 4 = 5$
  - 성적(y) 평균: $(81+ 93 + 91 + 97) \div 4 = 90.5$
- 이를 위 식에 대입하면 다음과 같음

$$a = \frac{(2-5)(81-90.5)+(4-5)(93-90.5)+(6-5)(91-90.5)+(8-5)(97-90.5)}{(2-5)^2+(4-5)^2+(6-5)^2+(8-5)^2}$$

$$= \frac{46}{20}$$

$$= 2.3$$

3 최소 제곱법

- 다음은 y절편인 b를 구하는 공식임

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

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

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

- 이제 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성됨

$$y = 2.3x + 79$$

3 최소 제곱법

- 예측 값 :

  x를 대입했을 때 나오는 y값

표 3-1 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**

3 최소 제곱법

- 좌표 평면에 이 예측 값을 찍어 보자

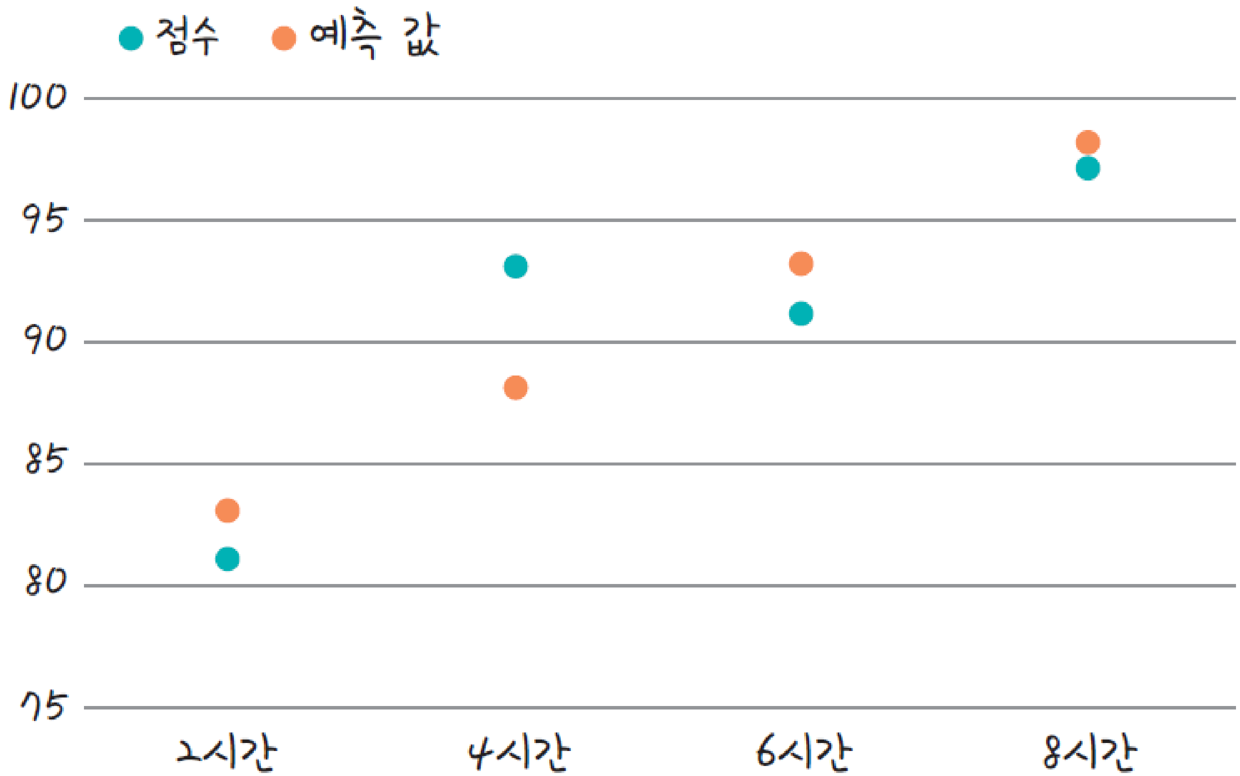


그림 3-2 공부한 시간, 성적, 예측 값을 좌표로 표현

3 최소 제곱법

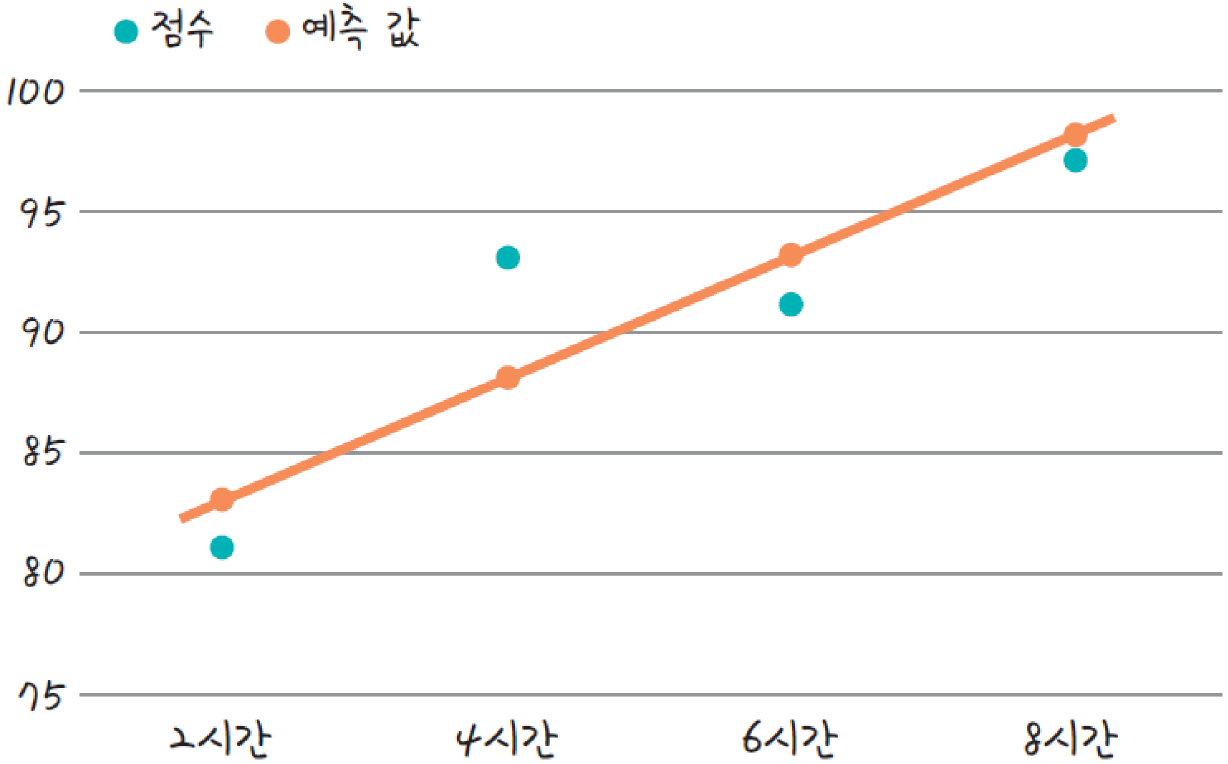


그림 3-3 오차가 최저가 되는 직선의 완성

3 최소 제곱법

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

4 코딩으로 확인하는 최소 제곱

- 넘파이 라이브러리를 불러옴
- 앞서 나온 데이터 값을 '리스트' 형식으로 다음과 같이 x와 y로 정의함

```
import numpy as np

x = [2, 4, 6, 8]
y = [81, 93, 91, 97]
```

TIP

파이썬 리스트를 만들려면 다음과 같이 리스트 이름을 정한 후 대괄호([ ])로 감싼 요소들을 쉼표(,)로 구분해 대입하면 됩니다.

리스트 이름 = [요소 1, 요소 2, 요소 3, ...]

4 코딩으로 확인하는 최소 제곱

- 이제 최소 제곱근 공식으로 기울기 a 와 y 절편 b 의 값을 구해보자
- x 의 모든 원소의 평균을 구하는 넘파이 함수는 mean( )임
- mx 라는 변수에 x 원소들의 평균값을, my에 y 원소들의 평균값을 입력

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

4 코딩으로 확인하는 최소 제곱

- 최소 제곱근 공식 중 분모의 값, 즉 ‘ x의 각 원소와 x의 평균값들의 차를 제곱하라’는 파이썬 명령을 만들 차례임
- 다음과 같이 divisor라는 변수를 만들어 구현할 수 있음

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

TIP

- sum()은 Σ에 해당하는 함수입니다
- **2는 제곱을 구하라는 의미입니다.
- for i in x는 x의 각 원소를 한 번씩 i 자리에 대입하라는 의미입니다.

4 코딩으로 확인하는 최소 제곱

- 이제 분자에 해당하는 부분을 구함
- x와 y의 편차를 곱해서 합한 값을 구하면 됨
- 다음과 같이 새로운 함수를 정의하여 dividend 변수에 분자의 값을 저장함

```
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)
```

4 코딩으로 확인하는 최소 제곱

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

TIP

def는 함수를 만들 때 사용하는 예약어입니다. 여기서는 top()이라는 함수를 새롭게 만들었고, 그 안에 최소 제곱법의 분자식을 그대로 가져와 구현하였습니다.

4 코딩으로 확인하는 최소 제곱

- 이제 위에서 구한 분모와 분자를 계산하여 기울기 a를 구함

```
a = dividend / divisor
```

- a를 구하고 나면   y절편을 구하는 공식을 이용해 b를 구할 수 있음

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

4 코딩으로 확인하는 최소 제곱

코드 3-1 선형 회귀 실습

- 예제 소스: deeplearning_class/01_Linear_Square_Method.ipynb

```
import numpy as np


# x 값과 y 값
x=[2, 4, 6, 8]
y=[81, 93, 91, 97]
```

4 코딩으로 확인하는 최소 제곱

```
# x와 y의 평균값
mx = np.mean(x)
my = np.mean(y)
print("x의 평균값:", mx)
print("y의 평균값:", my)

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

4 코딩으로 확인하는 최소 제곱

```
# 기울기 공식의 분자
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)

print("분모:", divisor)
print("분자:", dividend)
```

4 코딩으로 확인하는 최소 제곱

```
# 기울기와 y 절편 구하기
a = dividend / divisor
b = my - (mx*a)


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

4 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
```

5 평균 제곱 오차

- 여러 개의 입력 값을 계산할 때는 임의의 선을 그리고 난 후, 이 선이 얼마나 잘 그려졌는지를 평가하여 조금씩 수정해 가는 방법을 사용함
- 이를 위해 주어진 선의 오차를 평가하는 오차 평가 알고리즘이 필요함

6 잘못 그은 선 바로잡기

- 모든 딥러닝 프로젝트는 여러 개의 입력 변수를 다룸
- 가장 많이 사용하는 방법은 ‘일단 그리고 조금씩 수정해 나가기’ 방식임
- 가설을 하나 세운 뒤 이 값이 주어진 요건을 충족하는지 판단하여 조금씩 변화를 줌
- 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법
- 이는 딥러닝을 가능하게 해 주는 가장 중요한 원리 중 하나임

6 잘못 그은 선 바로잡기

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

6 잘못 그은 선 바로잡기

- 지금부터 오차를 계산하는 방법을 알아보자

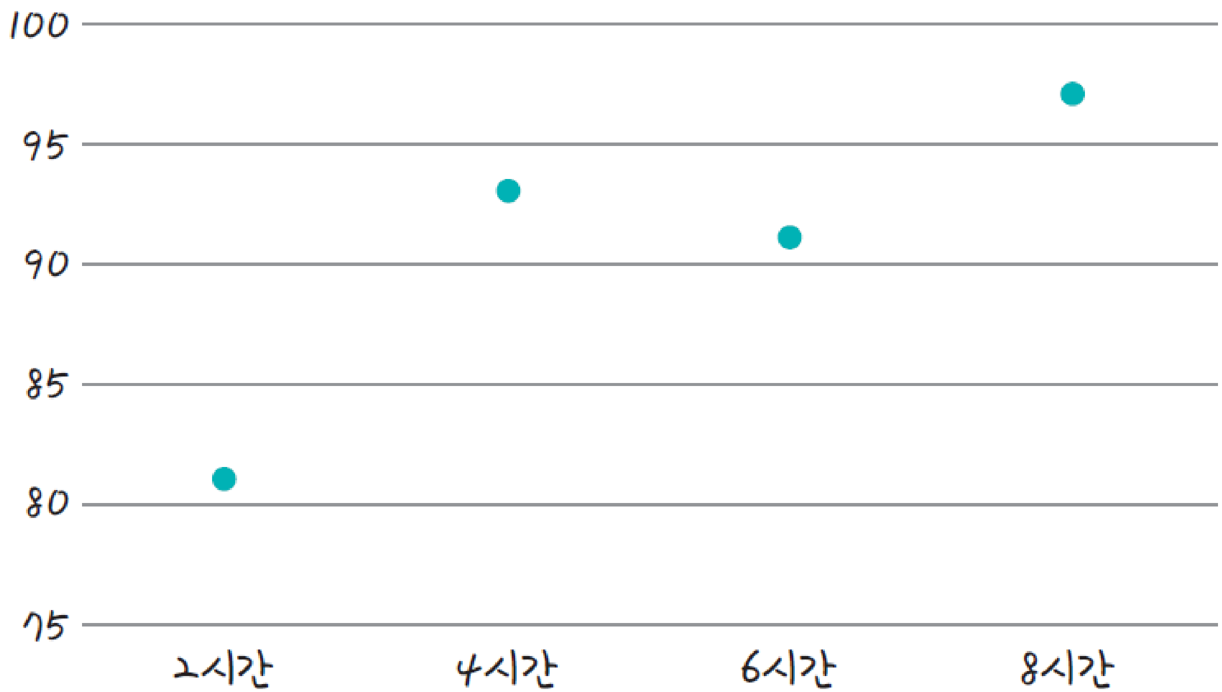


그림 3-4 공부한 시간과 성적의 관계도

6 잘못 그은 선 바로잡기

- 임의의 값을 대입한 뒤 오차를 구하고 이 오차를 최소화하는 방식을 사용해서 최종 a와 최종 b의 값을 구해 보자
- 대강 선을 그어보기 위해서 기울기 a와 y 절편 b 를 임의의 수 3과 76 이라고 가정해 보자

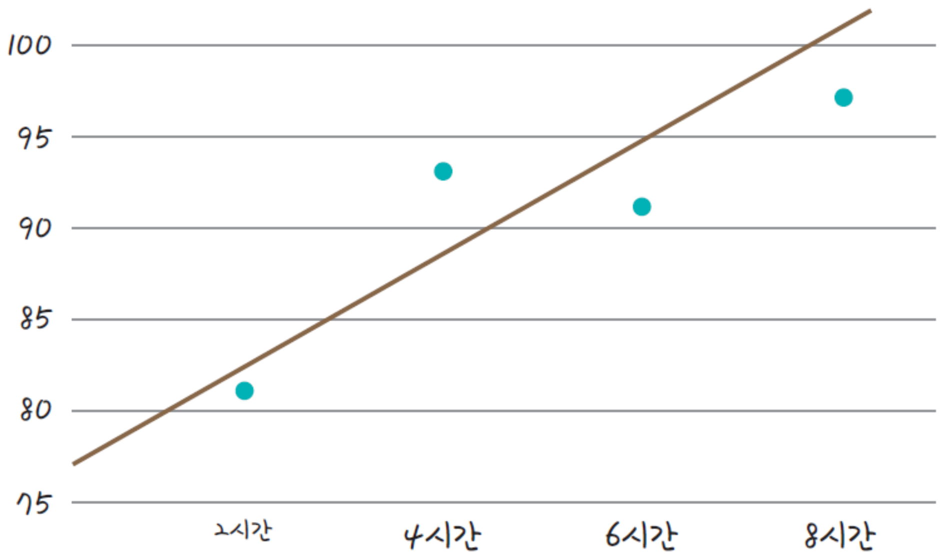


그림 3-5 임의의 직선 그려보기

6 잘못 그은 선 바로잡기

- 그림 3-6과 같은 임의의 직선이 어느 정도의 오차가 있는지를 확인하려면 각 점과 그래프 사이의 거리를 재면 됨

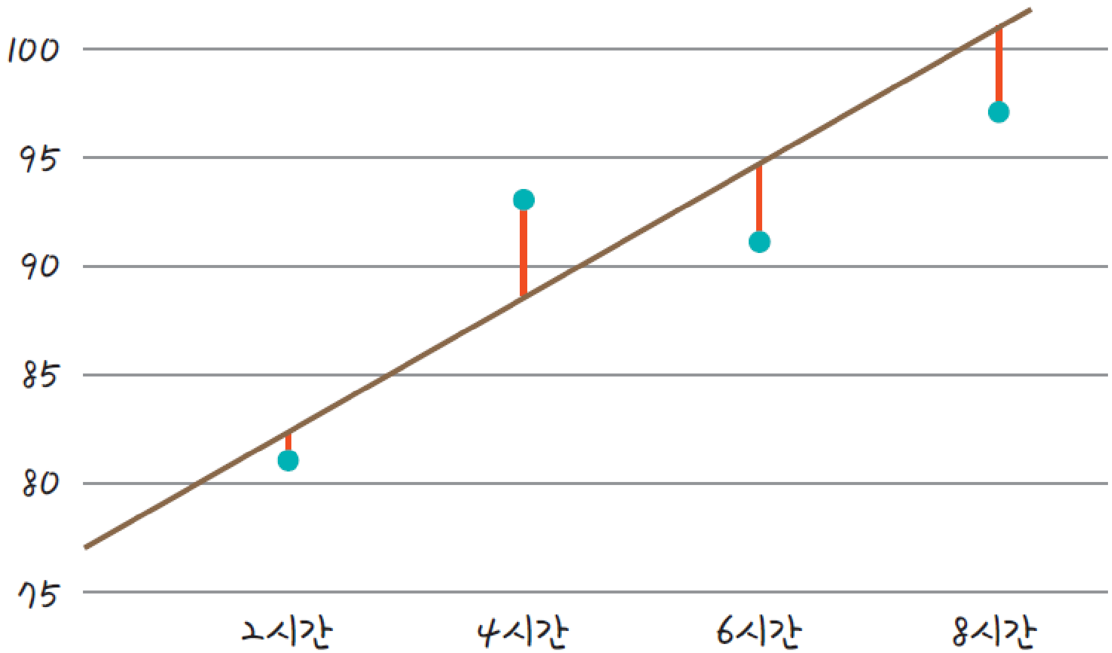


그림 3-6 임의의 직선과 실제 값 사이의 거리

6 잘못 그은 선 바로잡기

- 그림 3-6에서 볼 수 있는 빨간색 선은 직선이 잘 그어졌는지를 나타냄
- 이 직선들의 합이 작을수록 잘 그어진 직선이고, 이 직선들의 합이 클수록 잘못 그어진 직선이 됨

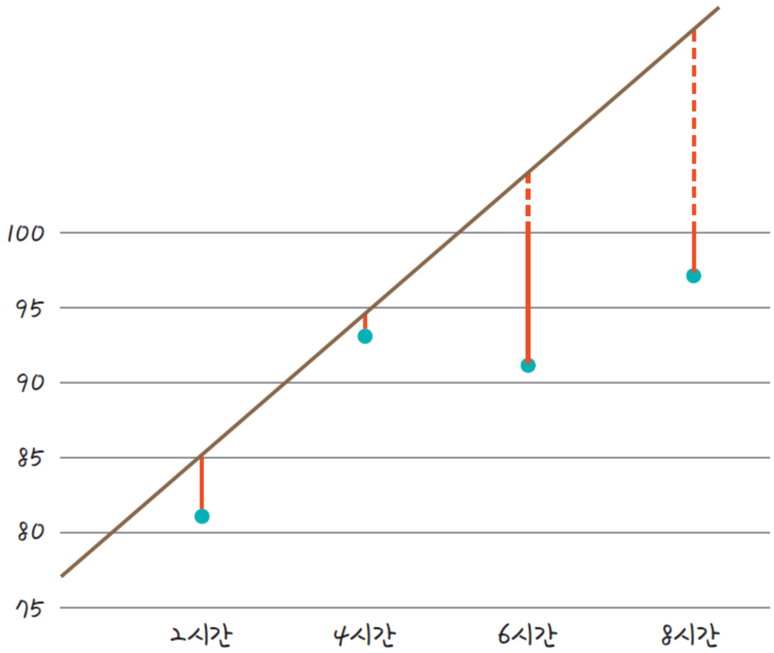


그림 3-7 기울기를 너무 크게 잡았을 때의 오차

6 잘못 그은 선 바로잡기

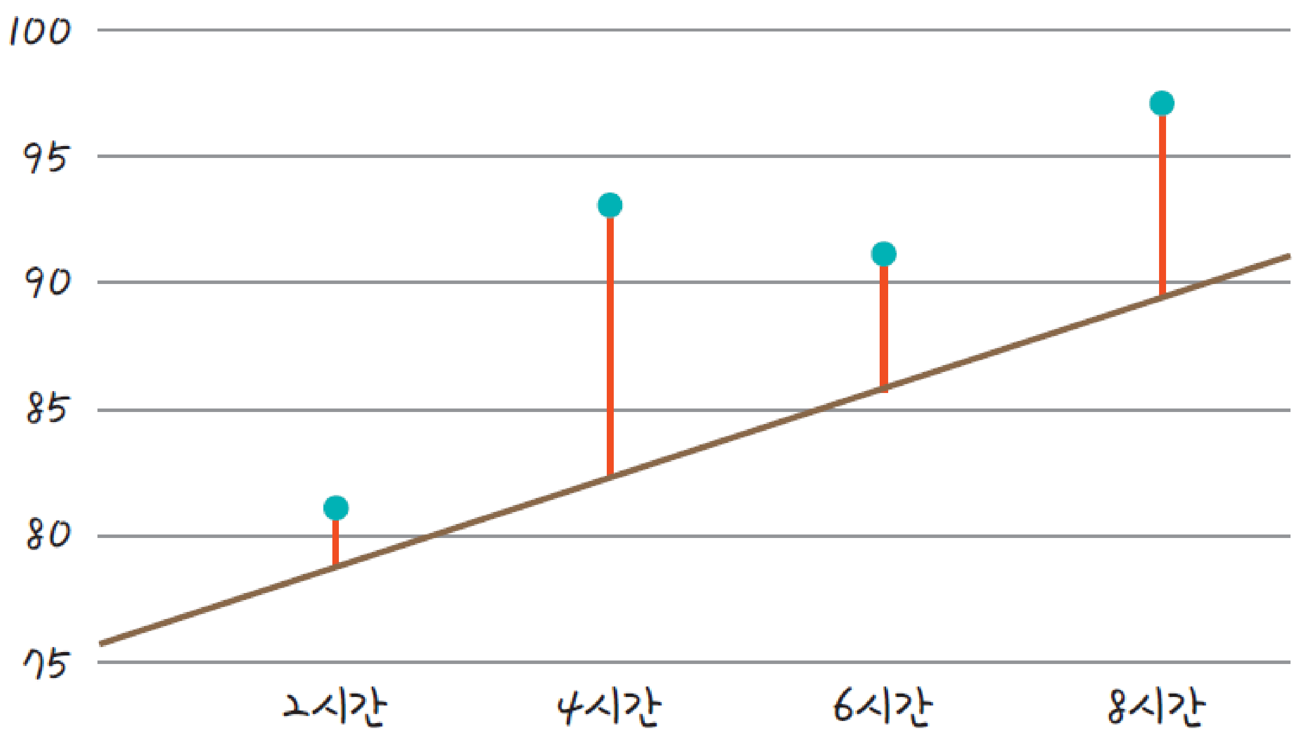


그림 3-8 기울기를 너무 작게 잡았을 때의 오차

6 잘못 그은 선 바로잡기

- 그래프의 기울기가 잘못 되었을수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
- 만약 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있는 것을 알 수 있음
- 거리는 입력 데이터에 나와 있는 y의 '실제 값'과 x를 식에 대입해서 나오는 '예측 값'과의 차이를 통해 구할 수 있음

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

6 잘못 그은 선 바로잡기

표 3-3 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

- 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 있어서 오차를 단순히 더해 버리면 합이 0이 될 수도 있기 때문임
- 부호를 없애야 정확한 오차를 구할 수 있음

6 잘못 그은 선 바로잡기

- 오차의 합을 구할 때는 각 오차의 값을 제곱해 줌

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

- 여기서 i 는 x 가 나오는 순서를, n 은 x 원소의 총 개수를 의미
- $\hat{y}_i$ 는 x_i 에 대응하는 '실제 값'이고 y_i 는 x_i 가 대입되었을 때 직선의 방정식(여기서는 $p = 3x + 76$)이 만드는 '예측 값'임
- 이 식으로 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44임

6 잘못 그은 선 바로잡기

- 평균 제곱 오차(Mean Squared Error, MSE) :

  오차의 합에 이어 각 x 값의 평균 오차를 이용함

  위에서 구한 값을 n으로 나누면 오차 합의 평균을 구할 수 있음

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

- 선형 회귀란 :

  임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고, 이 값을 가장 작게 만들어 주는 a와 b 값을 찾아가는 작업임

7 코딩으로 확인하는 평균 제곱 오차

- 이제 앞서 알아본 평균 제곱 오차를 파이썬으로 구현해 보자

```
fake_a_b = [3, 76]
```

7 코딩으로 확인하는 평균 제곱 오차

- 이번에는 data라는 리스트를 만들어 공부한 시간과 이에 따른 성적을 각각 짝을지어 저장함
- x 리스트와 y 리스트를 만들어 첫 번째 값을 x 리스트에 저장하고 두 번째 값을 y 리스트에 저장함

```
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]
```

TIP

파이썬에서 i[0]은 i 값 중 첫 번째를, i[1]은 두 번째 값을 의미합니다.

7 코딩으로 확인하는 평균 제곱 오차

- 다음은 내부 함수를 만들 차례임
- predict( )라는 함수를 사용해 일차 방정식 y = ax + b를 구현함

```
def predict(x):
    return fake_a_b[0]*x + fake_a_b[1]
```

7 코딩으로 확인하는 평균 제곱 오차

- 평균 제곱근 공식을 그대로 파이썬 함수로 옮기면 다음과 같음

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

```
def mse(y_hat, y):
    return ((y_hat-y) ** 2).mean())
```

- 여기서 **2는 제곱을 구하라는 뜻이고, mean( )은 평균값을 구하라는 뜻
- 예측 값과 실제 값을 각각 mse( )라는 함수의 y_hat와 y 자리에 입력해서 평균 제곱을 구함

7 코딩으로 확인하는 평균 제곱 오차

- 이제 mse( ) 함수에 데이터를 대입하여 최종값을 구하는 함수 mse_val( )

```
def mse_val(predict_result, y):
    return mse(np.array(predict_result), np.array(y))
```

- predict_result에는 앞서 만든 일차 방정식 함수 predict( )의 결괏값이 들어감
- 이 값과 y 값이 각각 예측 값과 실제 값으로 mse( ) 함수 안에 들어가게 됨

7 코딩으로 확인하는 평균 제곱 오차

- 이제 모든 x값을 predict( ) 함수에 대입하여 예측 값을 구함
- 이 예측 값과 실제 값을 통해 최종값을 출력하는 코드를 다음과 같이 작성함

```
# 예측 값이 들어갈 빈 리스트
predict_result = []

# 모든 x 값을 한 번씩 대입하여
for i in range(len(x)):
    # 그 결과에 해당하는 predict_result 리스트를 완성
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제 점수=%.f, 예측 점수=%.f" % (x[i], y[i], 
    predict(x[i])))
```

7 코딩으로 확인하는 평균 제곱 오차

코드 3-2 선형 회귀 실습 2

- 예제 소스: deeplearning_class/02_Mean_Squared_Error.ipynb

```
import numpy as np


# 기울기 a와 y 절편 b
fake_a_b = [3, 76]


# x, y의 데이터 값
data = [[2, 81], [4, 93], [6, 91], [8, 97]]
x = [i[0] for i in data]
y = [i[1] for i in data]
```

7 코딩으로 확인하는 평균 제곱 오차

```
# y = ax + b에 a와 b 값을 대입하여 결과를 출력하는 함수
def predict(x):
    return fake_a_b[0]*x + fake_a_b[1]

# MSE 함수
def mse(y_hat, y):
    return ((y_hat, y) ** 2).mean( ))

# MSE 함수를 각 y 값에 대입하여 최종 값을 구하는 함수
def mse_val(predict_result,y):
    return mse(np.array(predict_result), np.array(y))
```

7 코딩으로 확인하는 평균 제곱 오차

```
# 예측 값이 들어갈 빈 리스트
predict_result = []

# 모든 x 값을 한 번씩 대입하여
for i in range(len(x)):
    # predict_result 리스트를 완성
    predict_result.append(predict(x[i]))
    print("공부한 시간=%.f, 실제 점수=%.f, 예측 점수=%.f" % (x[i], y[i],
    predict(x[i])))

# 최종 MSE 출력
print("mse 최종값: " + str(mse_val(predict_result,y)))
```

7 코딩으로 확인하는 평균 제곱 오차

실행 결과

```
공부한 시간=2, 실제 점수=81, 예측 점수=82
공부한 시간=4, 실제 점수=93, 예측 점수=88
공부한 시간=6, 실제 점수=91, 예측 점수=94
공부한 시간=8, 실제 점수=97, 예측 점수=100
mse 최종값: 11.0
```