강사 윤영주
yj.youn1103@gmail.com

둘째마당
예측 모델의 기본 원리

4장 가장 훌륭한 예측선

1 선형 회귀의 정의

2 가장 훌륭한 예측선이란?

3 최소 제곱법

4 파이썬 코딩으로 확인하는 최소 제곱

5 평균 제곱 오차

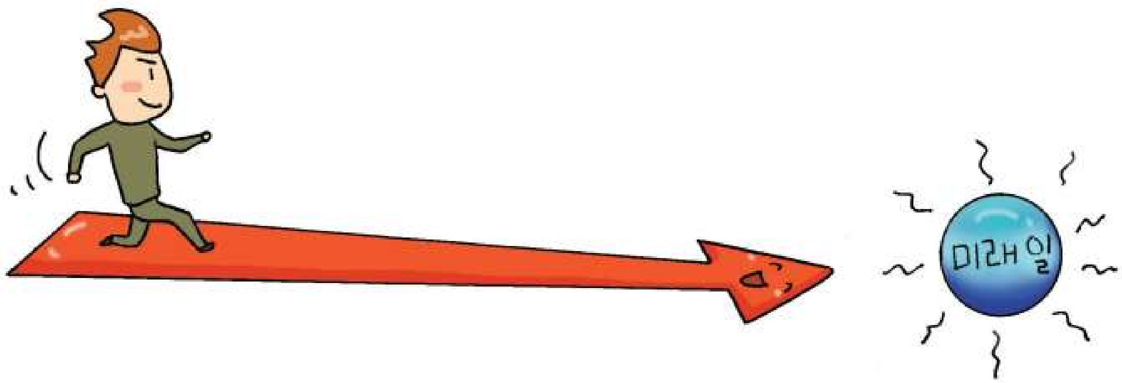
6 파이썬 코딩으로 확인하는 평균 제곱 오차

가장 훌륭한 예측선

- **가장 훌륭한 예측선**
  - 딥러닝은 자그마한 통계의 결과들이 무수히 얽히고설켜 이루어지는 복잡한 연산의 결정체
  - 우리 몸을 이해하려면 몸을 구성하는 기본 단위인 세포의 역할을 알아야 하듯, 딥러닝을 이해하려면 딥러닝의 가장 말단에서 이루어지는 기본적인 두 가지 계산 원리를 알아야 함
  - 바로 **선형 회귀**와 **로지스틱 회귀**

가장 훌륭한 예측선

- **가장 훌륭한 예측선**
  - 이 두 가지 개념을 중·고등학교 수준에서 공부하기란 쉽지 않음
  - 대학에서 통계를 전공하지 않았다면 익숙하지 않을 주제
  - 그러다 보니 여기서부터 시작하는 머신 러닝이 쉽지 않아 보이는 것이 무리는 아님
  - 이 두 개념을 이해하기 위해 반드시 어려운 공식이나 수학, 통계학 개념에 통달해야 하는 것은 아님
  - 어렵지 않은 수학 용어와 중·고등학교 수준으로도 딥러닝의 밑그림이 되는 개념을 충분히 이해할 수 있음
  - 이를 알고 나면 딥러닝을 구동시키는 원리에 한 걸음 다가설 수 있음



가장 훌륭한 예측선

- **가장 훌륭한 예측선**
  - 이 장의 제목인 '가장 훌륭한 예측선'이라는 표현은 '**선형 회귀**(linear regression) 분석을 이용한 모델'의 의미를 쉽게 풀어서 표현한 것
  - 머신 러닝은 제대로 된 선을 긋는 작업부터 시작
  - 선의 방향을 잘 정하면 그 선을 따라가는 것만으로도 지금은 보이지 않는 미래의 것을 예측할 수 있기 때문임
  - 첫 단추가 많은 것을 결정
  - 진입 장벽을 허물고 딥러닝의 세계로 들어오기 바람

1 선형 회귀의 정의

1 선형 회귀의 정의

● 선형 회귀의 정의

"학생들의 중간고사 성적이 다 다르다."

네, 다르겠죠.

그런데 위 문장이 나타낼 수 있는 정보는 너무 제한적입니다. 학급의 학생마다 제각각 성적이 다르다는 당연한 사실 외에는 알 수 있는 것이 없습니다. 이번에는 다음 문장을 보겠습니다.

"학생들의 중간고사 성적이 [ ]에 따라 다 다르다."

1 선형 회귀의 정의

- **선형 회귀의 정의**
  - 이 문장은 정보가 담길 여지를 열어 놓고 있음
  - [ ] 부분에 시험 성적을 좌우할 만한 여러 가지 것이 들어간다면 좀 더 많은 사실을 전달할 수 있음
  - 예를 들어 공부한 시간, 시험 당일의 컨디션, 사교육비 지출액 등이 들어갈 수 있음
  - 무엇이 들어가든지 해당 성적의 이유를 나름대로 타당하게 설명할 수 있음
  - 앞의 문장보다는 이 문장이 중간고사 성적의 차이와 이유를 나타낼 때 더욱 효과적

1 선형 회귀의 정의

- **선형 회귀의 정의**
  - 여기서 [ ]에 들어갈 내용을 '정보'라고 함
  - 머신 러닝과 딥러닝은 이 정보가 필요함
  - 정보를 정확히 준비해 놓기만 하면 성적을 예측하는 방정식을 만들 수도 있음
  - 이 단순한 정의를 이번에는 좀 더 수학적인 언어로 표현해 보자
  - 성적을 변하게 하는 '정보' 요소를 x라고 하고, 이 x 값에 따라 변하는 '성적'을 y라고 하자
  - 이를 정의하면 'x 값이 변함에 따라 y 값도 변한다'가 됨
  - 이 정의 안에서 독립적으로 변할 수 있는 값 x를 **독립 변수**라고 함
  - 또한, 이 독립 변수에 따라 종속적으로 변하는 y를 **종속 변수**라고 함

1 선형 회귀의 정의

- **선형 회귀의 정의**
  - 독립 변수가 x 하나뿐이어서 이것만으로 정확히 설명할 수 없을 때는 x_1, x_2, x_3 등 x 값을 여러 개 준비해 놓을 수도 있음
  - 하나의 x 값만으로도 y 값을 설명할 수 있다면 **단순 선형 회귀**(simple linear regression)라고 함
  - 또한, x 값이 여러 개 필요하다면 **다중 선형 회귀**(multiple linear regression)라고 함

2 가장 훌륭한 예측선이란?

2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 우선 독립 변수가 하나뿐인 단순 선형 회귀의 예를 공부해 보자
  - 성적을 결정하는 여러 요소 중에 '공부한 시간' 한 가지만 놓고 생각해 보자
  - 중간고사를 본 4명의 학생에게 각각 공부한 시간을 물어보고 이들의 중간고사 성적을 표 4-1과 같이 정리했다고 하자

▼ **표 4-1 | 공부한 시간과 중간고사 성적 데이터**

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

2 가장 훌륭한 예측선이란?

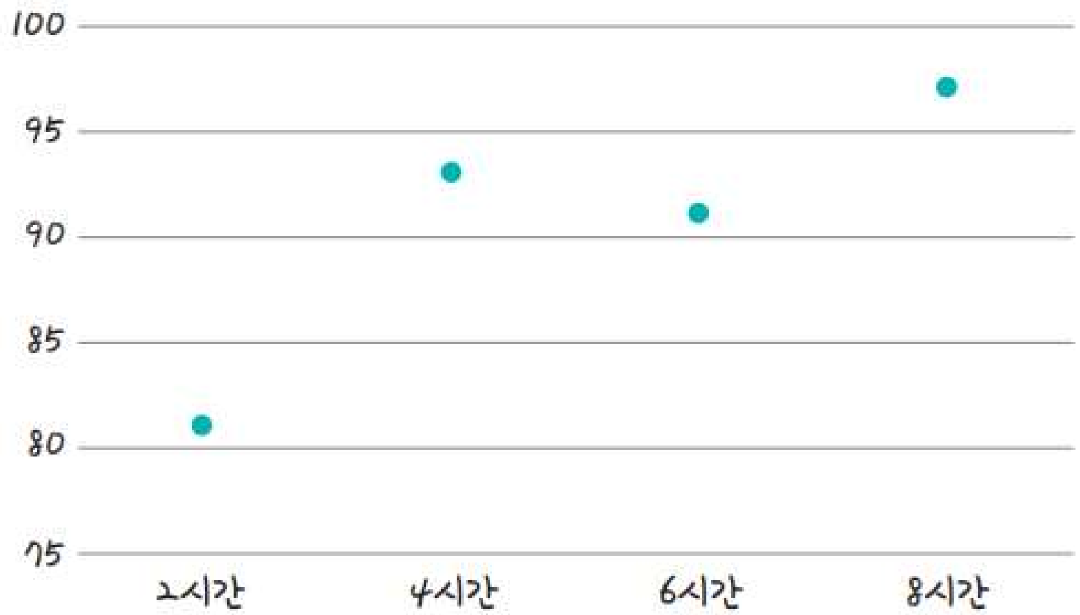
- **가장 훌륭한 예측선이란?**
  - 여기서 공부한 시간을 x라고 하고 성적을 y라고 할 때, 집합 x와 집합 y를 다음과 같이 표현할 수 있음

$$X = \{2,\ 4,\ 6,\ 8\}$$

$$Y = \{81,\ 93,\ 91,\ 97\}$$

2 가장 훌륭한 예측선이란?

▼ 그림 4-1 | 공부한 시간과 성적을 좌표로 표현



2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 좌표 평면에 나타내 놓고 보니, 왼쪽이 아래로 향하고 오른쪽이 위를 향하는 일종의 '선형(선으로 표시될 만한 형태)'을 보임
  - 선형 회귀를 공부하는 과정은 이 점들의 특징을 가장 잘 나타내는 선을 그리는 과정과 일치

2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 이 데이터에서 주어진 점들의 특징을 담은 선은 직선이므로 곧 일차 함수 그래프
  - 일차 함수 그래프는 다음과 같은 식으로 표현할 수 있음

$$y = ax + b$$

2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 여기서 x 값은 독립 변수이고 y 값은 종속 변수
  - 즉, x 값에 따라 y 값은 반드시 달라짐
  - 다만, 정확하게 계산하려면 상수 a와 b의 값을 알아야 함
  - 이 직선을 훌륭하게 그으려면 직선의 기울기 a 값과 y 절편 b 값을 정확히 예측해 내야 함
  - 앞서 선형 회귀는 곧 정확한 선을 그려 내는 과정이라고 했음
  - 지금 주어진 데이터에서의 선형 회귀는 결국 최적의 a 값과 b 값을 찾아내는 작업이라고 할 수 있음

2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 선을 잘 긋는 것이 어째서 중요할까?
  - 잘 그어진 선을 통해 우리는 표 4-1의 공부한 시간과 중간고사 성적 데이터에 들어 있지 않은 여러 가지 내용을 유추할 수 있기 때문임
  - 예를 들어 표 4-1에 나와 있지 않은 또 다른 학생의 성적을 예측하고 싶다고 하자
  - 이때 정확한 직선을 그어 놓았다면 이 학생이 몇 시간을 공부했는지만 물어보면 됨
  - 정확한 a값과 b 값을 따라 움직이는 직선에 학생이 공부한 시간인 x 값을 대입하면 예측 성적인 y 값을 구할 수 있는 것

2 가장 훌륭한 예측선이란?

- **가장 훌륭한 예측선이란?**
  - 딥러닝을 포함한 머신 러닝의 예측은 결국 이러한 기본 접근 방식과 크게 다르지 않음
  - 기존 데이터(정보)를 가지고 어떤 선이 그려질지 예측한 후, 아직 답이 나오지 않은 그 무언가를 그 선에 대입해 보는 것
  - 선형 회귀의 개념을 이해하는 것은 딥러닝을 이해하는 데 중요한 첫걸음

3 최소 제곱법

3 최소 제곱법

최소 제곱법

- 이제 우리 목표는 가장 정확한 선을 긋는 것
- 더 구체적으로는 정확한 기울기 a와 정확한 y 절편 b를 알아내면 된다고 했음
- 만일 우리가 **최소 제곱법**(method of least squares)이라는 공식을 알고 적용한다면, 이를 통해 일차 함수의 기울기 a와 y 절편 b를 바로 구할 수 있음

3 최소 제곱법

최소 제곱법

- 지금 가진 정보가 x 값(입력 값, 여기서는 '공부한 시간')과 y 값(출력 값, 여기서는 '성적')일 때 이를 이용해 기울기 a를 구하는 방법은 다음과 같음

$$a = \frac{(x - x\ \text{평균})(y - y\ \text{평균})\text{의 합}}{(x - x\ \text{평균})^2\text{의 합}} \qquad \text{(식 4.1)}$$

3 최소 제곱법

- **최소 제곱법**
  - 이것이 바로 최소 제곱법 공식
  - 쉽게 풀어서 다시 쓰면 x의 편차(각 값과 평균과의 차이)를 제곱해서 합한 값을 분모로 놓고, x와 y의 편차를 곱해서 합한 값을 분자로 놓으면 기울기가 나온다는 의미
  - 실제로 우리가 가진 y(성적) 값과 x(공부한 시간) 값을 이 식에 대입해 보자
  - 먼저 x 값의 평균과 y 값의 평균을 구해 보면 다음과 같음

    공부한 시간(x) 평균: (2 + 4 + 6 + 8) ÷ 4 = 5

    성적(y) 평균: (81+ 93 + 91 + 97) ÷ 4 = 90.5

3 최소 제곱법

- **최소 제곱법**
  - 이를 식 4.1에 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

- **최소 제곱법**
  - 기울기 a는 2.3이 나옴!
  - 다음은 y 절편인 b를 구하는 공식

$$b = y\text{의 평균} - (x\text{의 평균} \times \text{기울기 } a) \qquad \text{(식 4.2)}$$

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

3 최소 제곱법

- **최소 제곱법**
  - 우리는 이미 y 평균, x 평균, 그리고 조금 전 구한 기울기 x까지 이 식을 풀기 위해 필요한 모든 변수를 알고 있음
  - 이를 식에 대입해 보자

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

3 최소 제곱법

- **최소 제곱법**
  - y 절편 b는 79가 나왔음
  - 이제 다음과 같이 예측 값을 구하기 위한 직선의 방정식이 완성

$$y = 2.3x + 79$$

3 최소 제곱법

최소 제곱법

- 이 식에 우리가 가진 데이터를 대입해 보자
- x를 대입했을 때 나오는 y 값을 '예측 값'이라고 하겠음

▼ **표 4-2 | 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**

최소 제곱법

- 좌표 평면에 이 예측 값을 찍어 보자

▼ 그림 4-2 | 공부한 시간, 성적, 예측 값을 좌표로 표현

3 최소 제곱법

- **최소 제곱법**
  - 예측한 점들을 연결해 직선을 그으면 그림 4-3과 같음

▼ **그림 4-3 | 오차가 최저가 되는 직선의 완성**

3 최소 제곱법

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

4 파이썬 코딩으로 확인하는 최소 제곱

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 우리가 이론을 배우는 목적은 딥러닝을 구현하기 위해서임
  - 이 책에서 설명하는 모든 이론을 자유롭게 코드로 변환할 수 있어야 진정한 의미가 있음
  - 지금까지 공부한 내용을 코딩으로 구현해 보자

4 파이썬 코딩으로 확인하는 최소 제곱

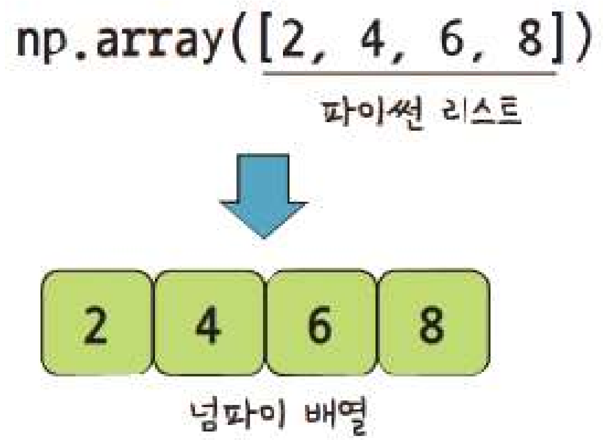
- **파이썬 코딩으로 확인하는 최소 제곱**
  - 먼저 넘파이 라이브러리를 불러옴
  - 넘파이는 파이썬에서 수학 연산과 분석을 하게 도와주는 라이브러리
  - 공부한 시간을 리스트로 만들어 x라는 이름의 넘파이 배열로 저장
  - 또 그때의 점수를 y라는 이름의 넘파이 배열로 저장

```
# 공부한 시간과 점수를 각각 x, y라는 이름의 넘파이 배열로 만듭니다.
x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])
```

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 파이썬에서 리스트는 대괄호([ ])로 감싼 요소들을 쉼표(,)로 구분해 대입하여 만듦
  - np.array() 함수를 사용하면 파이썬 리스트를 넘파이 배열로 바꾸어 여러 가지 계산을 수행할 수 있음

▼ **그림 4-4 | 파이썬 리스트의 기본 형태**



4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 이제 최소 제곱근 공식으로 기울기 a의 값과 y 절편 b의 값을 구해 보자
  - x의 모든 원소 평균을 구하는 넘파이 함수는 mean()
  - mx 변수에는 x 원소들의 평균값을, my 변수에는 y 원소들의 평균값을 넣음

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

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 이제 앞서 살펴본 최소 제곱근 공식 중 분모 값, 즉 'x의 각 원소와 x의 평균값들의 차를 제곱하라'는 파이썬 명령을 만들 차례
  - 다음과 같이 divisor라는 변수를 만들어 구현할 수 있음

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

sum( )은 Σ에 해당하는 함수입니다.

제곱을 구하라는 의미입니다.

x의 각 원소를 한 번씩 i 자리에 대입하라는 의미입니다.

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 이제 분자에 해당하는 부분을 구하겠음
  - x와 y의 편차를 곱해서 합한 값을 구하면 됨
  - 다음과 같이 새로운 함수를 정의해서 dividend 변수에 분자 값을 저장

```python
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 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 임의의 변수 d의 초깃값을 0으로 설정한 후 x의 개수만큼 실행
  - d에 x의 각 원소와 평균의 차, y의 각 원소와 평균의 차를 곱해서 차례로 더하는 최소 제곱법을 그대로 구현

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - def는 함수를 만들 때 사용하는 예약어
  - 여기서는 top() 함수를 새롭게 만들었고, 그 안에 최소 제곱법의 분자식을 그대로 가져와 구현
  - len(리스트)은 리스트 안에 들어 있는 원소 개수를 알려 줌
  - x 리스트의 원소가 네 개이므로 len(x)는 4가 됨
  - range()는 0부터 괄호 안의 숫자 바로 전까지 연속적인 숫자 객체를 만들어 줌
  - 즉, range(4)는 0, 1, 2, 3의 숫자를 생성하게 됨

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 이제 앞에서 구한 분모와 분자를 계산해 기울기 a를 구하겠음

```
a = dividend / divisor
```

  - a를 구하고 나면 y 절편을 구하는 공식을 이용해 b를 구할 수 있음

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

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**
  - 이를 하나의 파일로 정리해 보면 다음과 같음

실습 | 파이썬 코딩으로 구하는 최소 제곱

```
import numpy as np

# 공부한 시간과 점수를 각각 x, y라는 이름의 넘파이 배열로 만듭니다.
x = np.array([2, 4, 6, 8])
y = np.array([81, 93, 91, 97])

# x의 평균값을 구합니다.
mx = np.mean(x)
```

4 파이썬 코딩으로 확인하는 최소 제곱

파이썬 코딩으로 확인하는 최소 제곱

```
# y의 평균값을 구합니다.
my = np.mean(y)

# 출력으로 확인합니다.
print("x의 평균값: ", mx)
print("y의 평균값: ", my)

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

4 파이썬 코딩으로 확인하는 최소 제곱

- **파이썬 코딩으로 확인하는 최소 제곱**

```python
# 기울기 공식의 분자 부분입니다.
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)

# 출력으로 확인합니다.
print("분모: ", divisor)
print("분자: ", dividend)
```

4 파이썬 코딩으로 확인하는 최소 제곱

● **파이썬 코딩으로 확인하는 최소 제곱**

```
# 기울기 a를 구하는 공식입니다.
a = dividend / divisor

# y 절편 b를 구하는 공식입니다.
b = my - (mx*a)

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

4 파이썬 코딩으로 확인하는 최소 제곱

파이썬 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
```

- 파이썬으로 최소 제곱법을 구현해 기울기 a의 값과 y 절편 b의 값이 각각 2.3과 79임을 구할 수 있었음

모두의 딥러닝 개정 3판

5 평균 제곱 오차

5 평균 제곱 오차

- **평균 제곱 오차**
  - 최소 제곱법을 이용해 기울기 a와 y 절편을 편리하게 구했지만, 이 공식만으로 앞으로 만나게 될 모든 상황을 해결하기는 어려움
  - 여러 개의 입력을 처리하기에는 무리가 있기 때문임
  - 예를 들어 앞서 살펴본 예에서는 변수가 ‘공부한 시간’ 하나뿐이지만, 2장에서 살펴본 폐암 수술 환자의 생존율 데이터를 보면 입력 데이터의 종류가 17개나 됨
  - 딥러닝은 대부분 입력 값이 여러 개인 상황에서 이를 해결하기 위해 실행되기 때문에 기울기 a와 y 절편 b를 찾아내는 다른 방법이 필요함

5 평균 제곱 오차

- **평균 제곱 오차**
  - 가장 많이 사용하는 방법은 ‘일단 그리고 조금씩 수정해 나가기’ 방식
  - 가설을 하나 세운 후 이 값이 주어진 요건을 충족하는지 판단해서 조금씩 변화를 주고, 이 변화가 긍정적이면 오차가 최소가 될 때까지 이 과정을 계속 반복하는 방법
  - 이는 딥러닝을 가능하게 하는 가장 중요한 원리 중 하나

5 평균 제곱 오차

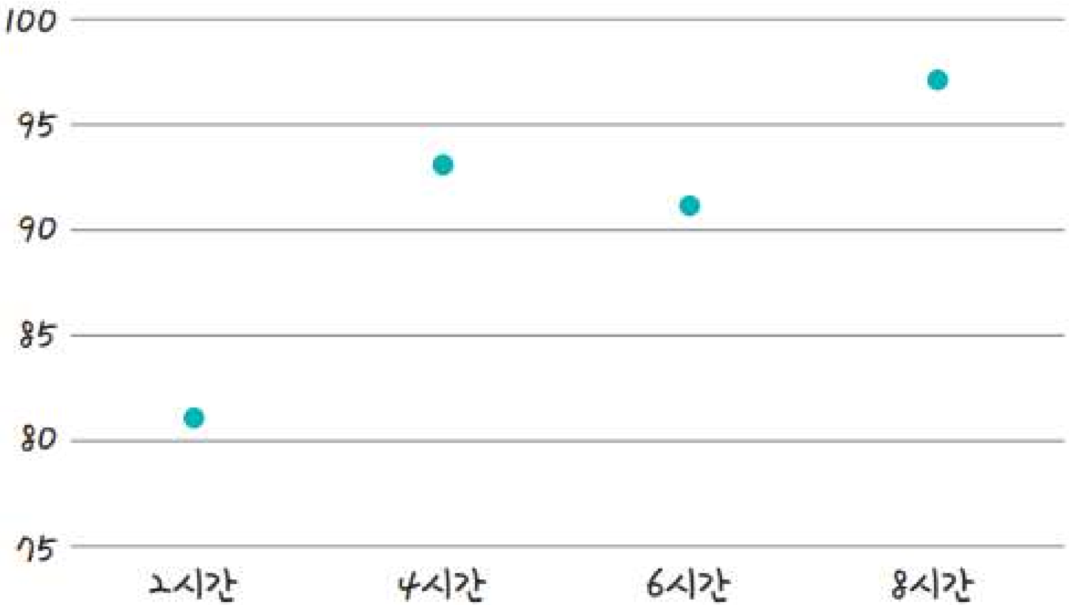
- **평균 제곱 오차**
  - 선을 긋고 나서 수정하는 과정에서 빠지면 안 되는 것이 있음
  - 나중에 그린 선이 먼저 그린 선보다 더 좋은지 나쁜지를 판단하는 방법
  - 즉, 각 선의 오차를 계산할 수 있어야 하고, 오차가 작은 쪽으로 바꾸는 알고리즘이 필요한 것

5 평균 제곱 오차

- **평균 제곱 오차**
  - 이를 위해 주어진 선의 오차를 평가하는 방법이 필요함
  - 오차를 구할 때 가장 많이 사용되는 방법이 **평균 제곱 오차**(Mean Square Error, MSE)
  - 지금부터 평균 제곱 오차를 구하는 방법을 알아보자
  - 앞서 나온 공부한 시간과 성적의 관계도를 다시 한 번 볼까?

5 평균 제곱 오차

▼ 그림 4-5 | 공부한 시간과 성적의 관계도



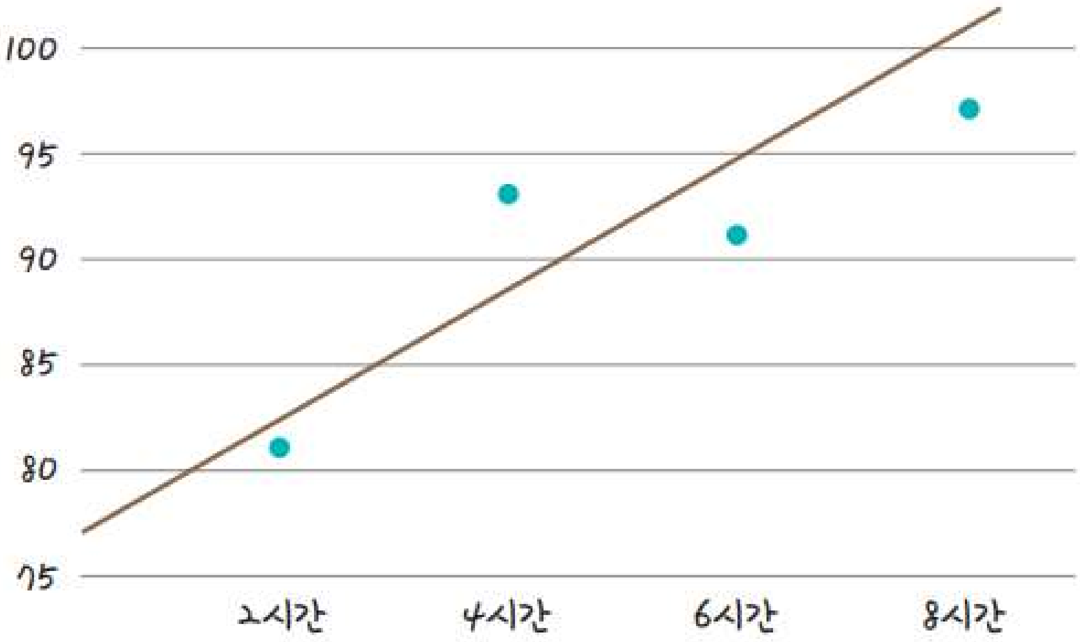
5 평균 제곱 오차

- **평균 제곱 오차**
  - 우리는 조금 전 최소 제곱법을 이용해 점들의 특성을 가장 잘 나타내는 최적의 직선이 $y = 2.3x + 79$임을 구했지만, 이번에는 최소 제곱법을 사용하지 않고 아무 값이나 a와 b에 대입해 보자
  - 임의의 값을 대입한 후 오차를 구하고 이 오차를 최소화하는 방식을 사용해서 최종 a 값과 b 값을 구해 보자

5 평균 제곱 오차

- **평균 제곱 오차**
  - 먼저 대강 선을 그어 보기 위해 기울기 a와 y 절편 b를 임의의 수 3과 76이라고 가정해 보자
  - Y = 3x + 76인 선을 그려 보면 그림 4-6과 같음

▼ **그림 4-6 | 임의의 직선 그려 보기**

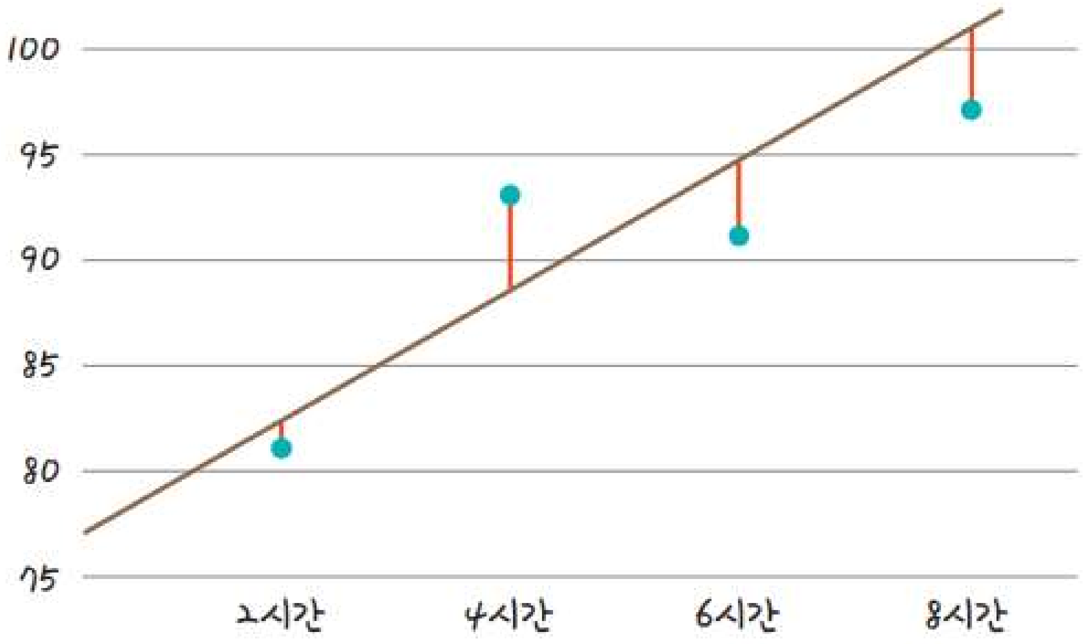


5 평균 제곱 오차

● 평균 제곱 오차

- 그림 4-6과 같은 임의의 직선이 어느 정도의 오차가 있는지 확인하려면 각 점과 그래프 사이의 거리를 재면 됨

▼ **그림 4-7 | 임의의 직선과 실제 값 사이의 거리**

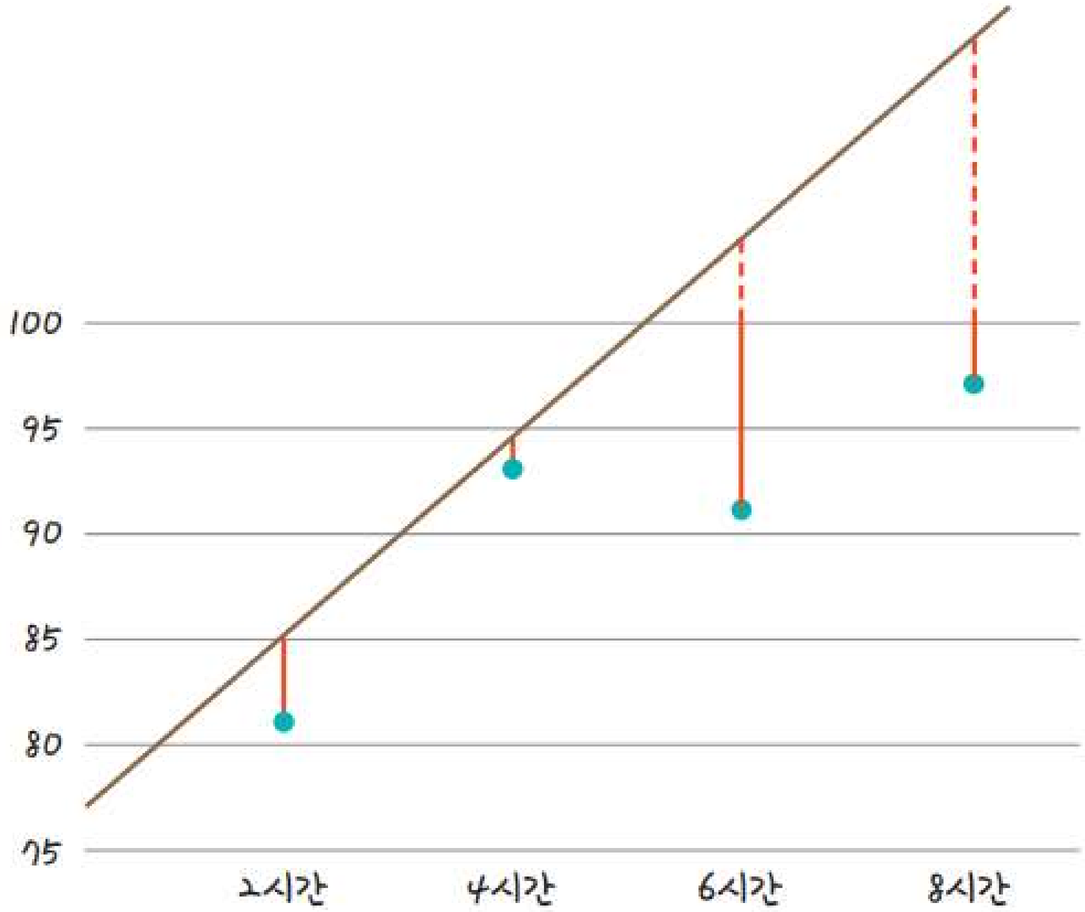


5 평균 제곱 오차

- **평균 제곱 오차**
    - 그림 4-7에서 볼 수 있는 빨간색 선은 직선이 잘 그어졌는지 나타냄
    - 이 직선들의 합이 작을수록 잘 그어진 직선이고, 이 직선들의 합이 클수록 잘못 그어진 직선이 됨
    - 예를 들어 기울기 값을 각각 다르게 설정한 그림 4-8과 그림 4-9의 그래프를 볼까?

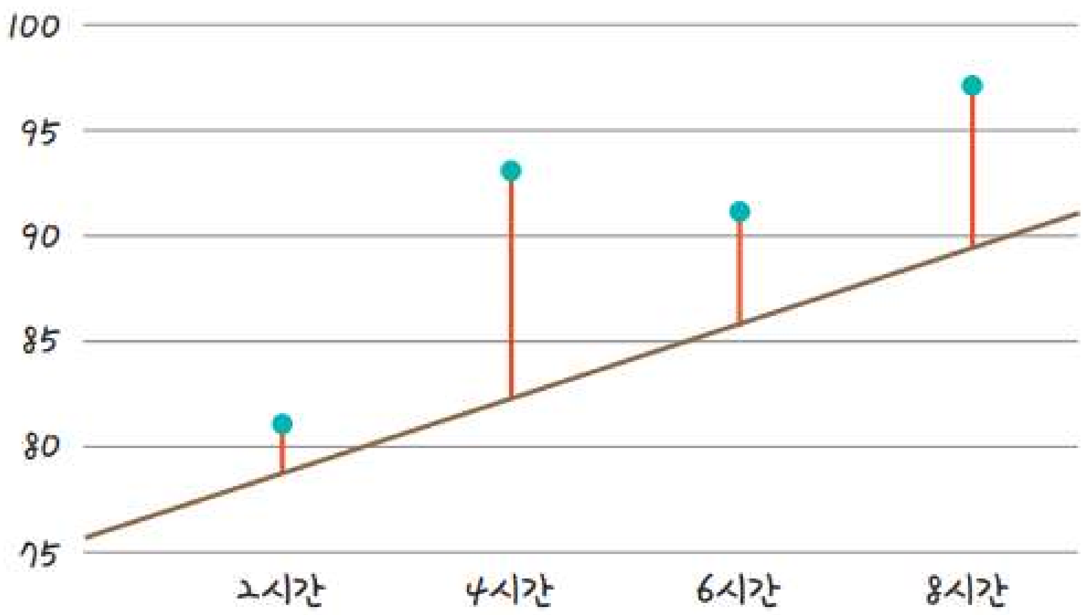
5 평균 제곱 오차

▼ 그림 4-8 | 기울기를 너무 크게 잡았을 때 오차



5 평균 제곱 오차

▼ 그림 4-9 | 기울기를 너무 작게 잡았을 때 오차



5 평균 제곱 오차

- **평균 제곱 오차**
  - 그래프의 기울기가 잘못되었을수록 빨간색 선의 거리의 합, 즉 오차의 합도 커짐
  - 만일 기울기가 무한대로 커지면 오차도 무한대로 커지는 상관관계가 있는 것을 알 수 있음
  - 빨간색 선의 거리의 합을 실제로 계산해 보자
  - 거리는 입력 데이터에 나와 있는 y의 '실제 값'과 x를 $y = 3x + 76$ 식에 대입해서 나오는 '예측 값'의 차이를 이용해 구할 수 있음
  - 예를 들어 2시간을 공부했을 때 실제 나온 점수(81점)와 그래프 $y = 3x + 76$ 식에 $x = 2$를 대입했을 때(82점)의 차이가 곧 오차
  - 오차를 구하는 방정식은 다음과 같음

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

5 평균 제곱 오차

- **평균 제곱 오차**
  - 이 식에 주어진 데이터를 대입해 얻을 수 있는 모든 오차 값을 정리하면 표 4-3과 같음

▼ **표 4-3 | 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

5 평균 제곱 오차

- **평균 제곱 오차**
  - 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 있어 오차를 단순히 더해 버리면 합이 0이 될 수도 있기 때문임
  - 부호를 없애야 정확한 오차를 구할 수 있음
  - 오차의 합을 구할 때는 각 오차 값을 제곱해 줌
  - 이를 식으로 표현하면 다음과 같음

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

5 평균 제곱 오차

- **평균 제곱 오차**
  - 여기서 i는 x가 나오는 순서를, n은 x 원소의 총 개수를 의미
  - y_i는 x_i에 대응하는 '실제 값'이고 $\hat{y}_i$는 x_i가 대입되었을 때 직선의 방정식(여기서는 y = 3x + 76)이 만드는 '예측 값'
  - 이 식으로 오차의 합을 다시 계산하면 1 + 25 + 9 + 9 = 44
  - 우리가 구하고자 하는 **평균 제곱 오차**는 위에서 구한 오차의 합을 n으로 나눈 것

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

5 평균 제곱 오차

- **평균 제곱 오차**
  - 이 식은 앞으로 머신 러닝과 딥러닝을 공부할 때 자주 등장할 중요한 식
  - 앞서 구한 오차의 합(=44)과 x 원소의 총 개수(=4)를 이 식에 대입하면 $\frac{1}{4} \times 44 = 11$이란 값이 나옴
  - 이로써 우리가 그은 임의의 직선이 11이라는 평균 제곱 오차를 갖는 직선이었다는 것을 알 수 있음
  - 이제 우리의 작업은 11보다 작은 평균 제곱 오차를 가지게 만드는 a 값과 b 값을 찾는 것이 되었음
  - 이렇듯 **선형 회귀**란 임의의 직선을 그어 이에 대한 평균 제곱 오차를 구하고, 이 값을 가장 작게 만들어 주는 a 값과 b 값을 찾아가는 작업

6 파이썬 코딩으로 확인하는 평균 제곱 오차

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 이제 앞서 알아본 평균 제곱 오차를 파이썬으로 구현해 보자
  - 임의로 정한 기울기 a와 y 절편 b의 값이 각각 3과 76이라고 할 때, 가상의 기울기가 fake_a, 가상의 y 절편이 fake_b인 함수식 predict()를 다음과 같이 정의할 수 있음

```python
fake_a = 3
fake_b = 76


def predict(x):
    return fake_a * x + fake_b
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 위 코드의 결괏값이 들어갈 빈 리스트를 만듦

```
predict_result = []
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 이제 모든 x 값을 predict() 함수에 한 번씩 대입해 예측 값 리스트를 채우는 코드를 다음과 같이 작성

```
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict
(x[i])))
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 다음으로 평균 제곱 오차를 구하는 함수를 만들 차례
  - 평균 제곱 오차 공식을 그대로 파이썬 함수로 옮기면 다음과 같음

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

```python
n = len(x)
def mse(y, y_pred):
    return (1/n) * sum((y - y_pred)**2)
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 여기서 **2는 제곱을 구하라는 것이고, sum()은 합을 구하라는 것
  - 실제 값과 예측 값을 각각 mse() 함수의 y와 y_pred 자리에 넣어서 평균 제곱을 구함

실습 | 파이썬 코딩으로 구하는 평균 제곱 오차

```
import numpy as np

# 가상의 기울기 a와 y 절편 b를 정합니다.
fake_a = 3
fake_b = 76

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

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**

```
# y = ax + b에 가상의 a 값과 b 값을 대입한 결과를 출력하는 함수입니다.
def predict(x):
    return fake_a * x + fake_b

# 예측 값이 들어갈 빈 리스트를 만듭니다.
predict_result = []

# 모든 x 값을 한 번씩 대입해 predict_result 리스트를 완성합니다.
for i in range(len(x)):
    predict_result.append(predict(x[i]))
    print("공부시간=%.f, 실제점수=%.f, 예측점수=%.f" % (x[i], y[i], predict
(x[i])))
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**

```python
# 평균 제곱 오차 함수를 각 y 값에 대입해 최종 값을 구하는 함수입니다.
n = len(x)
def mse(y, y_pred):
    return (1/n) * sum((y - y_pred)**2)

# 평균 제곱 오차 값을 출력합니다.
print("평균 제곱 오차: " + str(mse(y, predict_result)))
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**

실행 결과

```
공부시간=2, 실제점수=81, 예측점수=82
공부시간=4, 실제점수=93, 예측점수=88
공부시간=6, 실제점수=91, 예측점수=94
공부시간=8, 실제점수=97, 예측점수=100
평균 제곱 오차: 11.0
```

6 파이썬 코딩으로 확인하는 평균 제곱 오차

- **파이썬 코딩으로 확인하는 평균 제곱 오차**
  - 이를 통해 우리가 처음 가정한 a = 3, b = 76은 오차가 약 11.0이라는 것을 알게 됨
  - 이제 남은 것은 이 오차를 줄이면서 새로운 선을 긋는 것
  - 이를 위해서는 a 값과 b 값을 적절히 조절하면서 오차의 변화를 살펴보고, 그 오차가 최소화되는 a 값과 b 값을 구해야 함