인공신경망과딥러닝심화

Lecture 05. 선형 회귀 모델 – 먼저 긋고 수정하기

동덕여자대학교
데이터사이언스 전공
권 범

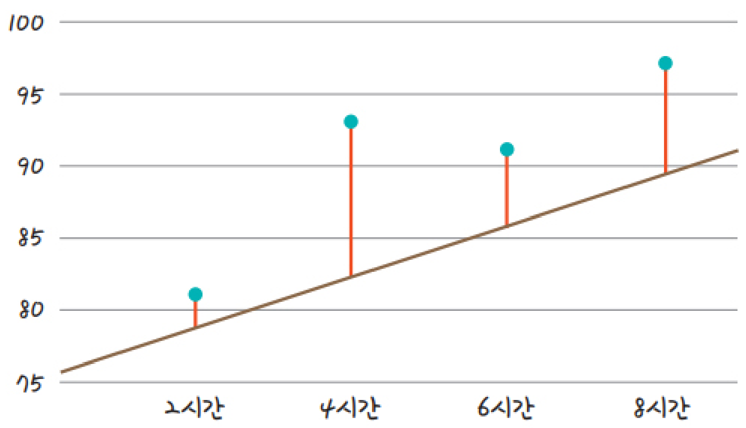
목차

시작하기 전에

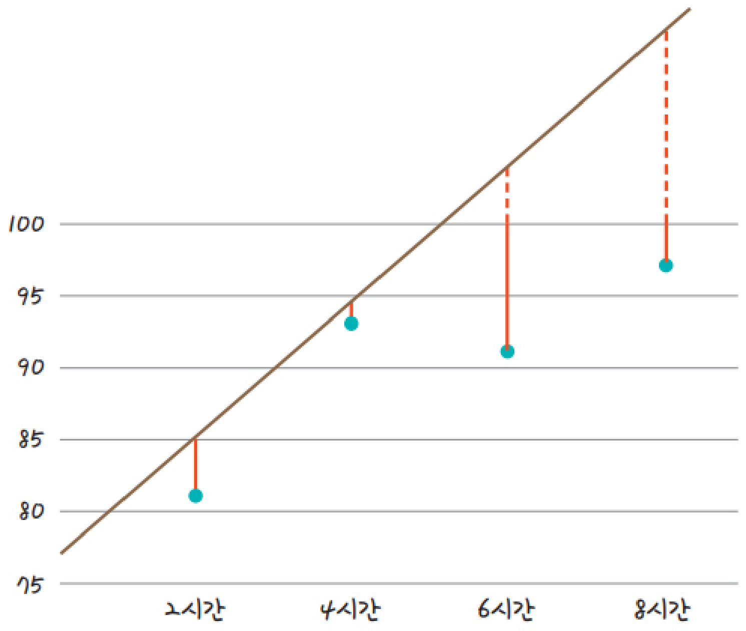
❖ 기울기와 오차의 관계 (1/3)

- 우리는 앞서 기울기 a를 너무 크게 잡거나 작게 잡으면 오차가 커지는 것을 확인
- 기울기 a와 오차 사이에는 이렇게 상관관계가 있음
- 이때 기울기가 무한대로 커지거나 무한대로 작아지면 그래프는 y축과 나란한 직선이 됨
- 오차도 함께 무한대로 커짐

기울기를 너무 작게 잡았을 때 오차



기울기를 너무 크게 잡았을 때 오차

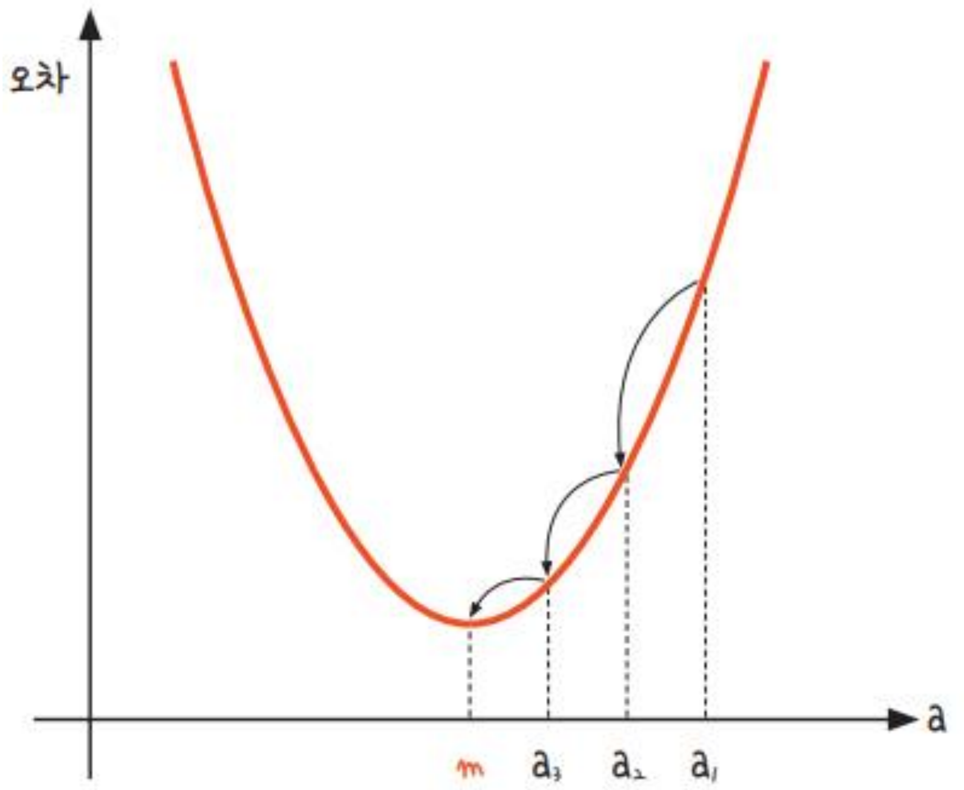


[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

시작하기 전에

❖ 기울기와 오차의 관계 (2/3)

- 이를 다시 표현하면 **기울기 a와 오차 사이에는** 아래 그림의 빨간색 그래프와 같은 **이차 함수의 관계가 있다는 의미**



[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

시작하기 전에

❖ 기울기와 오차의 관계 (3/3)

- 이 그래프상에서 오차가 가장 작을 때는 언제일까?
- 그래프의 가장 아래쪽 볼록한 부분에 이르렀을 때 즉 기울기 a가 m의 위치에 있을 때, 오차가 가장 작음

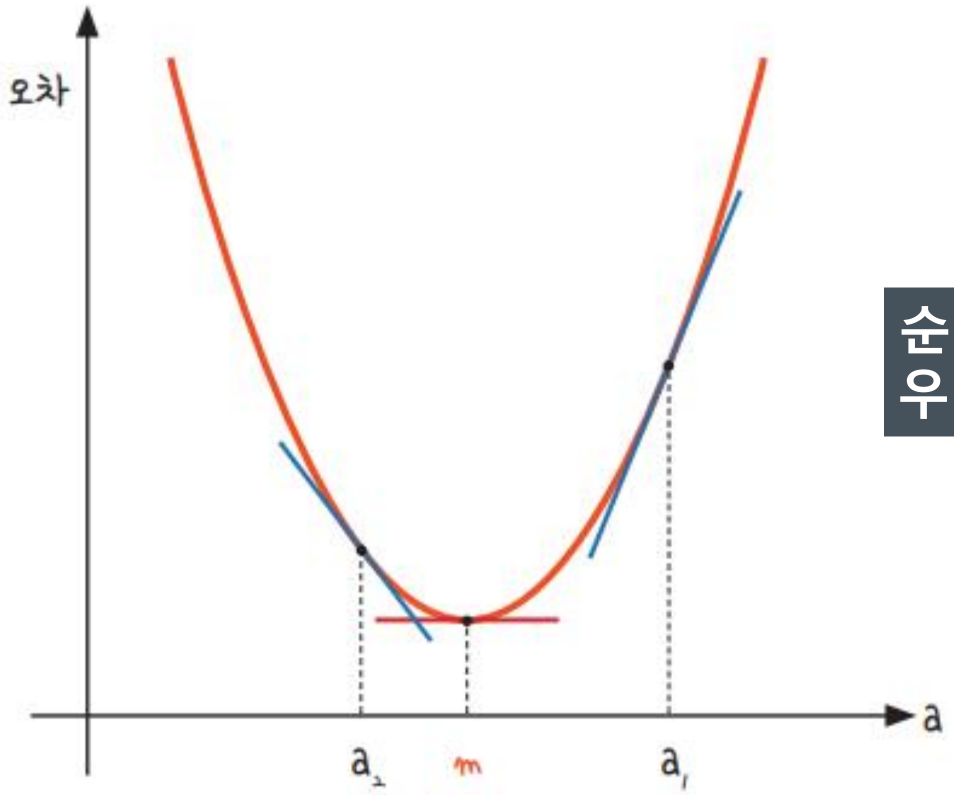
- 우리는 앞선 수업에서 임의의 기울기를 집어넣어 평균 제곱 오차를 구해 보았음
- 그때의 기울기를 a_1이라고 한다면, 기울기를 적절히 바꾸어 a_2, a_3으로 이동시키다 결국 m에 이르게 하면 최적의 기울기를 찾게 되는 것
- 이 작업을 위해 a_1 값보다 a_2 값이 m에 더 가깝고, a_3 값이 a_2 값보다 m에 더 가깝다는 것을 컴퓨터가 판단
- 이러한 판단을 하게 하는 방법이 바로 미분 기울기를 이용하는 **경사 하강법(Gradient Decent)**

01. 경사 하강법의 개요

01. 경사 하강법의 개요

❖ 경사 하강법에 대해 알아보기 (1/5)

- Lecture 03에서 미분은 한 점에서의 순간 기울기라고 배웠음
- $y = x^2$ 그래프에서 x에 다음과 같이 a_1, a_2 그리고 m을 대입해 그 자리에서 미분하면 아래 그림과 같이 각 점에서의 순간 기울기가 그려짐



순간 기울기가 0인 점이 곧 우리가 찾는 최솟값 m이다!

[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

01. 경사 하강법의 개요

❖ 경사 하강법에 대해 알아보기 (2/5)

- 여기서 눈여겨보아야 할 것은 우리가 찾는 최솟값 m에서의 순간 기울기
- 그래프가 이차 함수 포물선이므로 꼭짓점의 기울기는 x축과 평행한 선이 됨 즉, **기울기**가 **0**

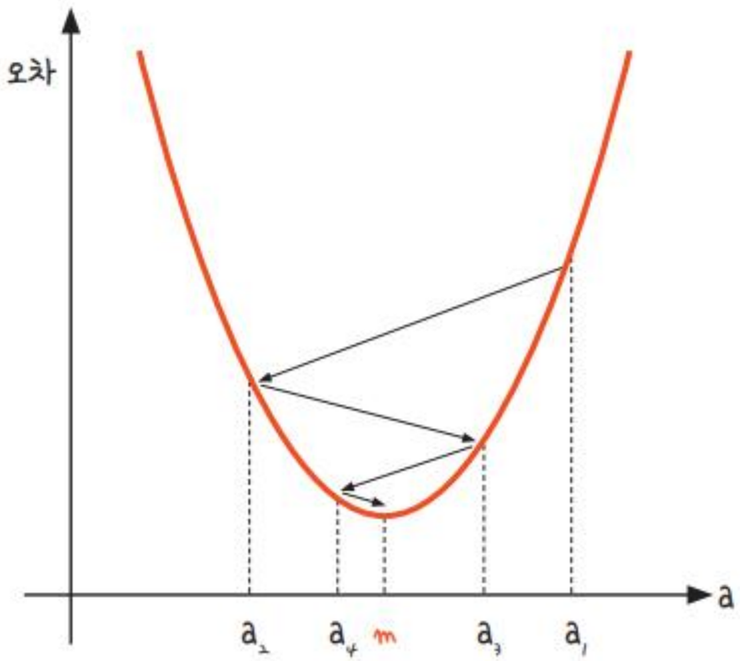
우리가 할 일은 **미분 값이 0인 지점**을 찾는 것이 됨

01. 경사 하강법의 개요

❖ 경사 하강법에 대해 알아보기 (3/5)

- 이를 위해 다음 과정을 거침

① a_1에서 미분을 구함
② 구한 기울기의 반대 방향(기울기가 +면 음의 방향, -면 양의 방향)으로 얼마간 이동시킴(a_2라고 가정)
③ a_2에서 미분을 구함
④ 앞에서 구한 미분 값이 0이 아니라면 ①~③의 과정을 반복



최솟점 m을 찾아가는 과정

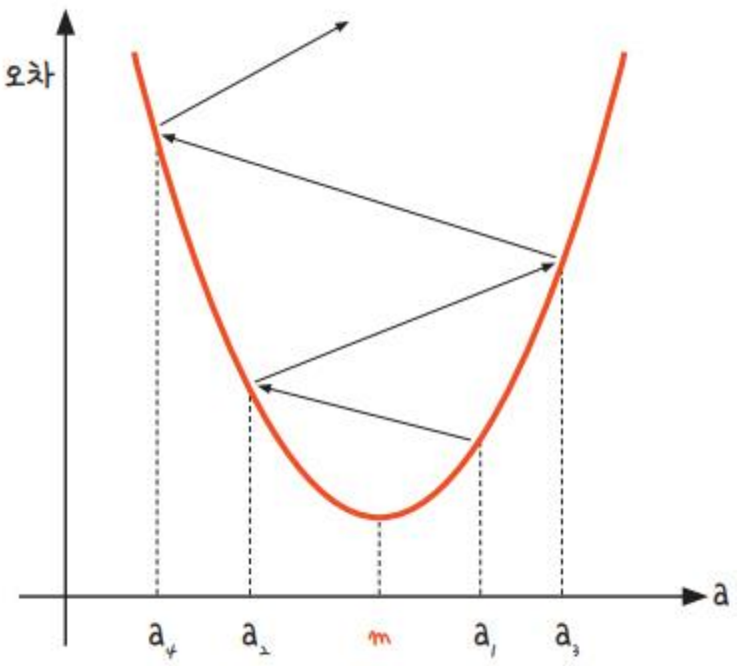
왼쪽 그림과 같이 기울기가 0인 한 점(m)으로 수렴

[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

01. 경사 하강법의 개요

❖ 경사 하강법에 대해 알아보기 (4/5)

- 경사 하강법은 이렇게 반복적으로 기울기 a를 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 아래 그림과 같이 위로 치솟아 버림



학습률을 너무 크게 잡으면 한 점으로 수렴하지 않고 발산

[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

01. 경사 하강법의 개요

❖ 경사 하강법에 대해 알아보기 (5/5)

- 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 그래프를 만들고,
적절한 학습률을 설정해 **미분 값이 0인 지점을 구하는 것**

- y-절편 b의 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

02. 파이썬 코딩으로 확인하는 선형 회귀

❖ 파이썬으로 선형 회귀 구현하기 (1/10)

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

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

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

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

02. 파이썬 코딩으로 확인하는 선형 회귀

❖ 파이썬으로 선형 회귀 구현하기 (2/10)

- 이 값을 미분할 때 우리가 궁금한 것은 a와 b라는 것을 기억해야 함
- 식 전체를 미분하는 것이 아니라 필요한 값을 중심으로 미분해야 하기 때문임
- 이렇게 특정한 값, 예를 들어 a와 b를 중심으로 미분할 때
  이를 a와 b로 '편미분한다'고 하는 것을 'Lecture 03. 딥러닝을 위한 기초 수학'에서 배웠음
- a와 b로 각각 편미분한 결과를 옮겨 보면 다음과 같음

a로 편미분한 결과

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

b로 편미분한 결과

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

❖ 파이썬으로 선형 회귀 구현하기 (3/10)

- 이를 각각 파이썬 코드로 바꾸면 다음과 같음

```
y_pred = a * x + b        # 예측 값을 구하는 식입니다.
error = y - y_pred        # 실제 값과 비교한 오차를 변수 error에 저장합니다.

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

❖ 파이썬으로 선형 회귀 구현하기 (4/10)

- 여기에 학습률을 곱해 기존의 a 값과 b 값을 업데이트

```
# 학습률(Learning Rate)을 정합니다.
lr = 0.03

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

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

❖ 파이썬으로 선형 회귀 구현하기 (5/10)

- 나머지는 앞서 공부한 바와 같으며, 그래프로 표현하는 코드를 넣어 정리하면 다음과 같이 코드가 완성

```
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.figure()
plt.scatter(x, y, marker=‘o’, c=‘k’, s=100)
plt.xlabel(“Study Time(x)”)
plt.ylabel(“Score(y)”)
plt.grid(True)
plt.show()
```

❖ 파이썬으로 선형 회귀 구현하기 (6/10)

```
# 기울기 a의 값과 y-절편 b의 값을 초기화합니다.
a = 0
b = 0

# 학습률(Learning Rate)을 정합니다.
lr = 0.03

# 몇 번 반복시킬지 횟수를 설정합니다.
epoch = 2001

# x의 값이 총 몇 개인지 셉니다.
n = len(x)
```

❖ 파이썬으로 선형 회귀 구현하기 (7/10)

```
# 경사 하강법(Gradient Descent)을 시작합니다.
for j in range(epoch):          # epoch 수 만큼 반복
    y_pred = a * x + b          # 예측 값을 구하는 식입니다.
    error = y - y_pred          # 실제 값과 비교한 오차를 변수 error에 저장합니다.

    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.

    if (j % 100) == 0:  # j가 100의 배수가 될 때마다 현재의 a 값, b 값을 출력합니다.
        print("epoch=%.f, 기울기=%.4f, y-절편=%.4f" % (j, a, b))
```

02. 파이썬 코딩으로 확인하는 선형 회귀

❖ 파이썬으로 선형 회귀 구현하기 (8/10)

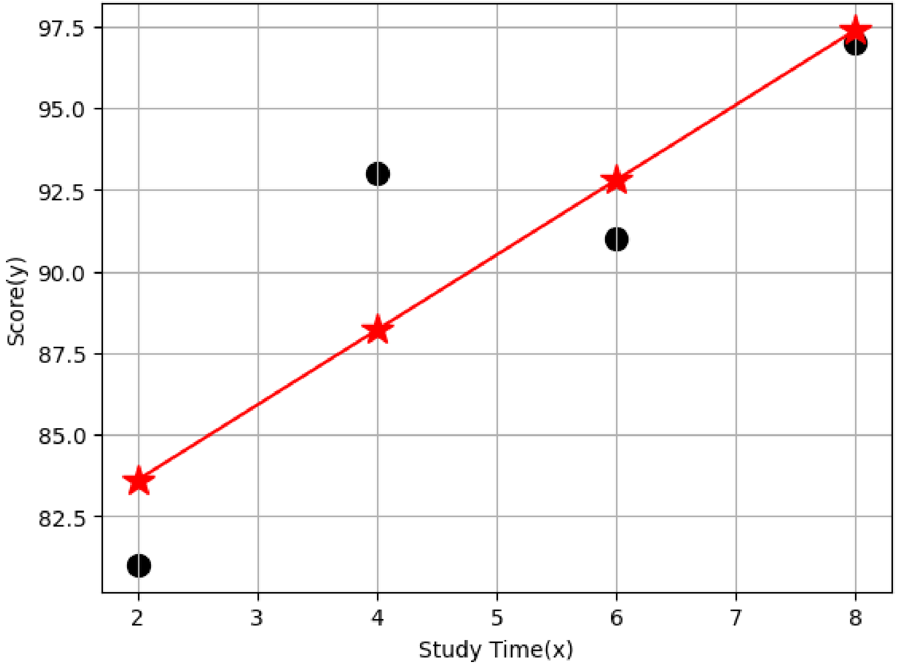
```
# 앞서 구한 최종 a 값을 기울기, b 값을 y-절편에 대입해 y_pred 값을 계산합니다.
y_pred = a * x + b

# 그래프를 그리고 출력합니다.
plt.figure()
plt.scatter(x, y, marker=‘o’, c=‘k’, s=100)
plt.plot(x, y_pred, marker=‘*’, c=‘r’, markersize=15)
plt.xlabel(“Study Time(x)”)
plt.ylabel(“Score (y)”)
plt.grid(True)
plt.show()
```

❖ 파이썬으로 선형 회귀 구현하기 (9/10)

실행결과

```
epoch=0, 기울기=27.8400, y-절편=5.4300
epoch=100, 기울기=7.0739, y-절편=50.5117
… (중략) …
epoch=1900, 기울기=2.3000, y-절편=79.0000
epoch=2000, 기울기=2.3000, y-절편=79.0000
```

- ✓ 여기서 에포크(Epoch)는 입력 값에 대해 몇 번이나 반복해서 실험했는지 나타냄
- ✓ j가 100의 배수가 될 때마다 결과를 출력함



그래프로 표현한 모습

02. 파이썬 코딩으로 확인하는 선형 회귀

❖ 파이썬으로 선형 회귀 구현하기 (10/10)

- 기울기 a의 값이 2.3에 수렴하는 것과 y-절편 b의 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 있음
- 이와 똑같은 방식을 x가 여러 개인 다중 선형 회귀에서도 사용

03. 다중 선형 회귀의 개요

03. 다중 선형 회귀의 개요

❖ 다중 선형 회귀(Multiple Linear Regression)에 대해 알아보기 (1/3)

- 앞서 학생들이 공부한 시간에 따른 예측 직선을 그리고자 기울기 a와 y-절편 b를 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 때문**임

- 더 정확한 예측을 하려면 추가 정보를 입력해야 하며,
  정보를 추가해 새로운 예측 값을 구하려면 **변수 개수를 늘려**
  다중 선형 회귀(Multiple/Multivariable Linear Regression) 만들어 주어야 함

❖ 다중 선형 회귀(Multiple Linear Regression)에 대해 알아보기 (2/3)

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

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

[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

03. 다중 선형 회귀의 개요

❖ 다중 선형 회귀(Multiple Linear Regression)에 대해 알아보기 (3/3)

- 그럼 지금부터 독립 변수 x_1과 x_2가 두 개 생긴 것
- 이를 사용해 종속 변수 y를 만들 경우,
  기울기를 두 개 구해야 하므로 다음과 같은 식이 나옴
  $y = a_1x_1 + a_2x_2 + b$

- 두 기울기 a_1과 a_2는 각각 어떻게 구할 수 있을까?
- 앞서 배운 경사 하강법을 그대로 적용

바로 파이썬 코드로 확인해 보겠음

04. 파이썬 코딩으로 확인하는 다중 선형 회귀

01. 경사 하강법의 개요
02. 파이썬 코딩으로 확인하는 선형 회귀
03. 다중 선형 회귀의 개요
05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 파이썬으로 다중 선형 회귀 구현하기 (1/13)

- 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 y의 넘파이 배열을 만듦

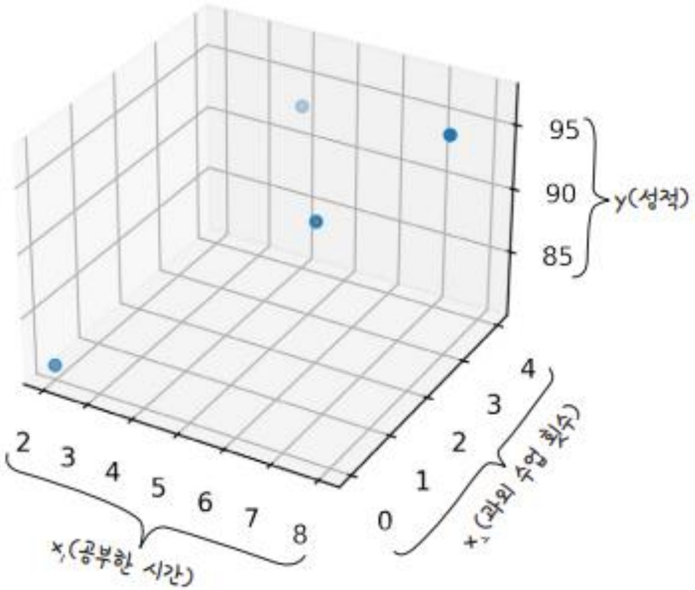
```
x1 = np.array([2, 4, 6, 8])
x2 = np.array([0, 4, 2, 3])
y = np.array([81, 93, 91, 97])
```

04. 파이썬 코딩으로 확인하는 다중 선형 회귀

❖ 파이썬으로 다중 선형 회귀 구현하기 (2/13)

- 데이터의 분포를 그래프로 표현해 보면 다음과 같음

```
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, projection="3d")
ax.scatter3D(x1, x2, y, s=200)
ax.set_xlabel("Study Time(x1)")
ax.set_ylabel("Private Lesson Time(x2)")
ax.set_zlabel("Score(y)")
plt.show()
```



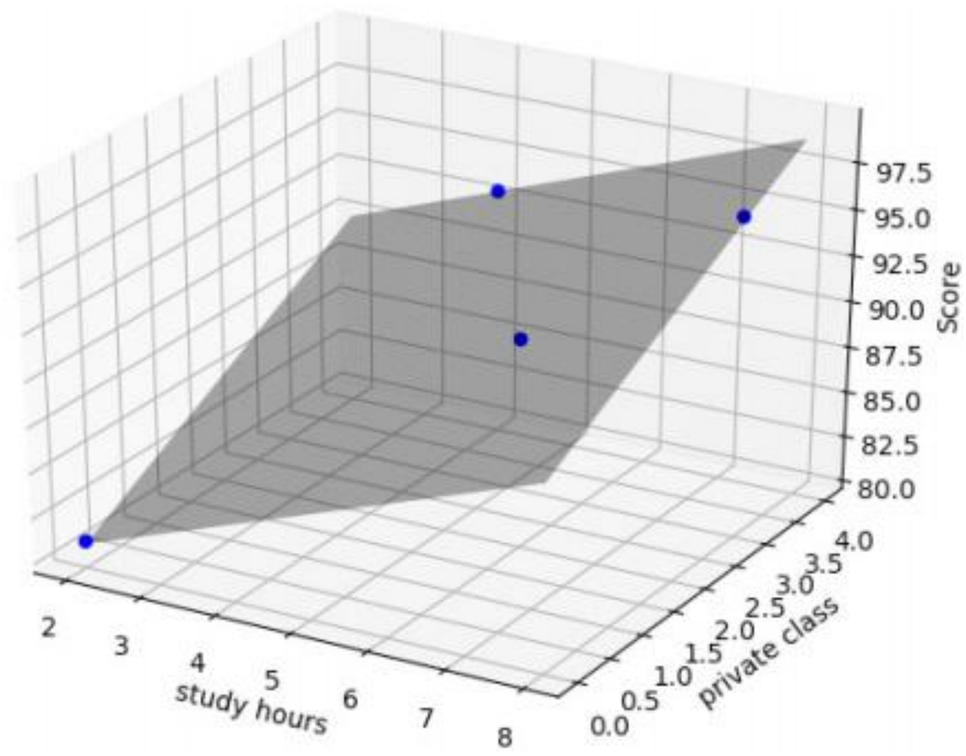
[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

축이 하나 더 늘어 3D로 배치된 모습

✓ 앞서 x와 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

❖ 파이썬으로 다중 선형 회귀 구현하기 (3/13)

- 앞서 선형 회귀는 선을 긋는 작업이라고 했음
- 그러면 다중 선형 회귀는 어떨까?
- 최적의 결과를 찾은 후 이를 그래프로 표현하면 아래 그림과 같이 **평면**으로 표시됨



다중 선형 회귀

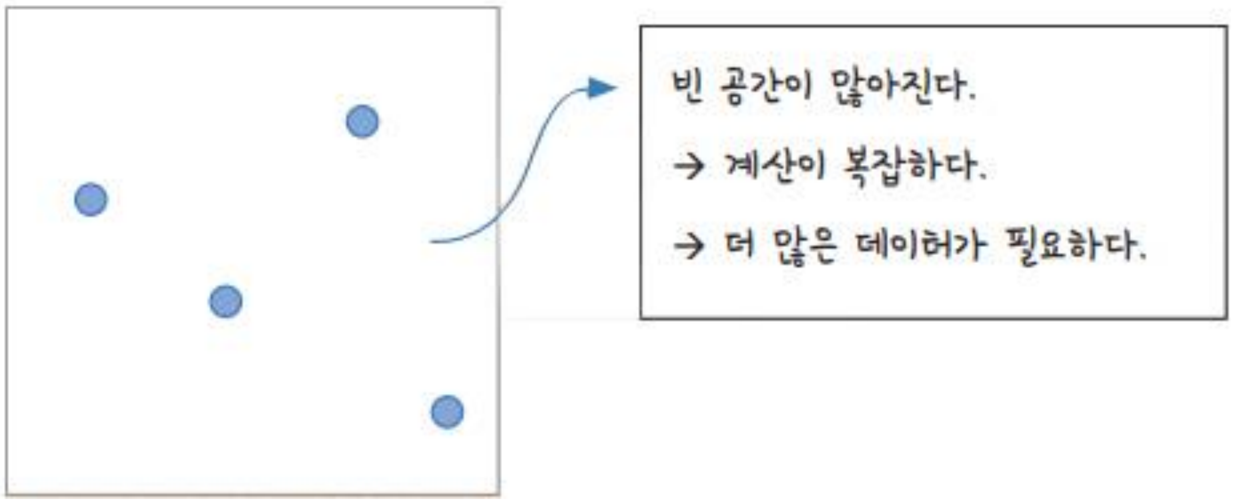
[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

❖ 파이썬으로 다중 선형 회귀 구현하기 (4/13)

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

단순 선형 회귀

다중 선형 회귀



[사진출처] 모두의 딥러닝 (출판사: 길벗, 저자: 조태호)

❖ 파이썬으로 다중 선형 회귀 구현하기 (5/13)

- 코드의 형태는 크게 다르지 않음
- 다만 x가 두 개가 되었으므로 x_1, x_2 두 변수를 만들고, 기울기도 a_1과 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    # 예측 값을 구하는 식을 세웁니다.
error = y - y_pred                # 실제 값과 비교한 오차를 변수 error에 저장합니다.
```

❖ 파이썬으로 다중 선형 회귀 구현하기 (6/13)

- 오차 함수를 a_1, a_2, b로 각각 편미분한 값을 a1_diff, a2_diff, b_diff라고 할 때, 이를 구하는 식은 다음과 같음

```
# x 값이 총 몇개인지 셉니다. x1과 x2의 수가 같으므로 x1만 세겠습니다.
n = len(x1)
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.
```

❖ 파이썬으로 다중 선형 회귀 구현하기 (7/13)

- 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.
```

04. 파이썬 코딩으로 확인하는 다중 선형 회귀

❖ 파이썬으로 다중 선형 회귀 구현하기 (8/13)

- 이제 실제 점수와 예측된 점수를 출력해서 예측이 잘되는지 확인

```
print(“실제 점수: ”, y)
print(“예측 점수: ”, y_pred)
```

04. 파이썬 코딩으로 확인하는 다중 선형 회귀

❖ 파이썬으로 다중 선형 회귀 구현하기 (9/13)

- 지금까지 코드를 정리하면 다음과 같음

```
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])

# 데이터의 분포를 그래프로 나타냅니다.
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(1, 1, 1, projection=“3d”)
ax.scatter3D(x1, x2, y, s=200)
ax.set_xlabel(“Study Time(x1)”)
ax.set_ylabel(“Private Lesson Time(x2)”)
ax.set_zlabel(“Score(y)”)
plt.show()

```

❖ 파이썬으로 다중 선형 회귀 구현하기 (10/13)

```python
# 기울기 a1, a2의 값과 y-절편 b의 값을 초기화합니다.
a1 = 0
a2 = 0
b = 0

# 학습률(Learning Rate)을 정합니다.
lr = 0.01  # 0.03으로 설정하면 발산합니다.

# 몇 번 반복시킬지 횟수를 설정합니다.
epoch = 2001

# x 값이 총 몇개인지 셉니다. x1과 x2의 수가 같으므로 x1만 세겠습니다.
n = len(x1)
```

❖ 파이썬으로 다중 선형 회귀 구현하기 (11/13)

```
# 경사 하강법(Gradient Descent)을 시작합니다.
for j in range(epoch):                    # epoch 수만큼 반복합니다.
    y_pred = a1 * x1 + a2 * x2 + b        # 예측 값을 구하는 식을 세웁니다.
    error = y - y_pred                    # 실제 값과 비교한 오차를 변수 error에 저장합니다.

    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 (j % 100) == 0:  # j가 100의 배수가 될 때마다 현재의 a1, a2, b의 값을 출력합니다.
      print("epoch=%.f, 기울기1=%.4f, 기울기2=%.4f, y-절편=%.4f" % (j, a1, a2, b))
```

04. 파이썬 코딩으로 확인하는 다중 선형 회귀

❖ 파이썬으로 다중 선형 회귀 구현하기 (12/13)

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

❖ 파이썬으로 다중 선형 회귀 구현하기 (13/13)

실행결과

```
epoch=0, 기울기1=9.2800, 기울기2=4.2250, y-절편=1.8100
epoch=100, 기울기1=9.5110, 기울기2=5.0270, y-절편=22.9205
epoch=200, 기울기1=7.3238, 기울기2=4.2950, y-절편=37.8751
… (중략) …
epoch=1800, 기울기1=1.5361, 기울기2=2.2982, y-절편=77.6095
epoch=1900, 기울기1=1.5263, 기울기2=2.2948, y-절편=77.6769
epoch=2000, 기울기1=1.5191, 기울기2=2.2923, y-절편=77.7260

실제 점수: [81 93 91 97]
예측 점수: [80.76387645 92.97153922 91.42520875 96.7558749 ]
```

반복을 통해 최적의 기울기 a_1과 a_2 및 절편을 찾아가며
실제 점수에 가까운 예측 값을 만들어 내고 있음을 알 수 있음

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

01. 경사 하강법의 개요
02. 파이썬 코딩으로 확인하는 선형 회귀
03. 다중 선형 회귀의 개요
04. 파이선 코딩으로 확인하는 다중 선형 회귀

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (1/12)

- 우리는 머신 러닝의 기본인 선형 회귀에 대해 배우고 있음

  우리의 목표는 **딥러닝**

- 앞으로 딥러닝을 실행하기 위해서 **텐서플로라는 라이브러리의 케라스 API를 불러와 사용**할 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

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (2/12)

- 선형 회귀는 현상을 분석하는 방법의 하나
- 머신러닝은 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
  머신러닝에서는 가설 함수(Hypothesis)라고 하며 $H(x)$라고 표기
- 또 기울기 a는 변수 x에 어느 정도의 가중치를 곱하는지 결정하므로,
  가중치(Weight)라고 하며, w로 표시
- 절편 b는 데이터의 특성에 따라 따로 부여되는 값이므로 **편향(Bias)**이라고 하며, b로 표시
- 우리가 앞서 배운 $y = ax + b$는 머신러닝에서 다음과 같이 표기

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

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (3/12)

- 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)

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (4/12)

- 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)

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (5/12)

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

```
from tensorflow import keras
from keras import Sequential, Input
from keras.layers import Dense
```

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (6/12)

- Sequential( ) 함수와 Dense( ) 함수는 Lecture 02에서 이미 소개한 바 있음
- 이 함수를 불러와 선형 회귀를 실행하는 코드는 다음과 같음

```
model = Sequential()
model.add(Input(shape=(1,)))
model.add(Dense(1, activation=“linear”))  # y = w * x + b

model.compile(optimizer=“sgd”, loss=“mse”)

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

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (7/12)

- 단 세 줄의 코드에 앞서 공부한 모든 것이 담겨 있음
- 어떻게 설정하는지 살펴보자

① 먼저 가설 함수는 $H(x) = wx + b$

✓ 입력될 변수(=학습 시간)는 하나뿐이므로 Input( ) 함수의 shape에 (1,)이라고 설정
✓ 이때 출력되는 값(=성적)도 하나이므로 Dense( ) 함수의 첫 번째 인자에 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 회귀를 다루고 있으므로 "linear"라고 적어 주면 됨
✓ 딥러닝 목적에 따라 다른 활성화 함수를 넣을 수 있는데, 예를 들어 다음 수업에서 배울 시그모이드 함수가 필요하다면 "sigmoid"라고 넣어 주는 식

② 경사 하강법

✓ 앞서 배운 경사 하강법을 실행하려면 옵티마이저에 sgd라고 설정
✓ 손실 함수는 평균 제곱 오차를 사용할 것이므로 mse라고 설정

③ 모델 학습

✓ 끝으로 앞서 따로 적어 주었던 epochs 숫자를 model.fit( ) 함수에 적음

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (8/12)

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

```
y_pred = model.predict(x)

# 예측 결과를 그래프로 나타냅니다.
plt.figure()
plt.scatter(x, y, marker='o', c='k', s=100)
plt.plot(x, y_pred, marker='*', c='r', markersize=15)
plt.xlabel("Study Time(x)")
plt.ylabel("Score(y)")
plt.grid(True)
plt.show()
```

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (9/12)

- 이제 모든 코드를 모아 보면 다음과 같음

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

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

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

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (10/12)

```
model = Sequential()

# 출력 값, 입력 값, 분석 방법에 맞게끔 모델을 설정합니다.
model.add(Input(shape=(1,)))
model.add(Dense(1, activation=“linear”))  # y = w * x + b

# 오차 수정을 위해 경사 하강법(Stochastic Gradient Descent, SGD)을,
# 오차의 정도를 판단하기 위해 평균 제곱 오차(MSE)를 사용합니다.
model.compile(optimizer=“sgd”, loss=“mse”)

# 오차를 최소화 하는 과정을 2000번 반복합니다.
model.fit(x, y, epochs=2000)
```

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (11/12)

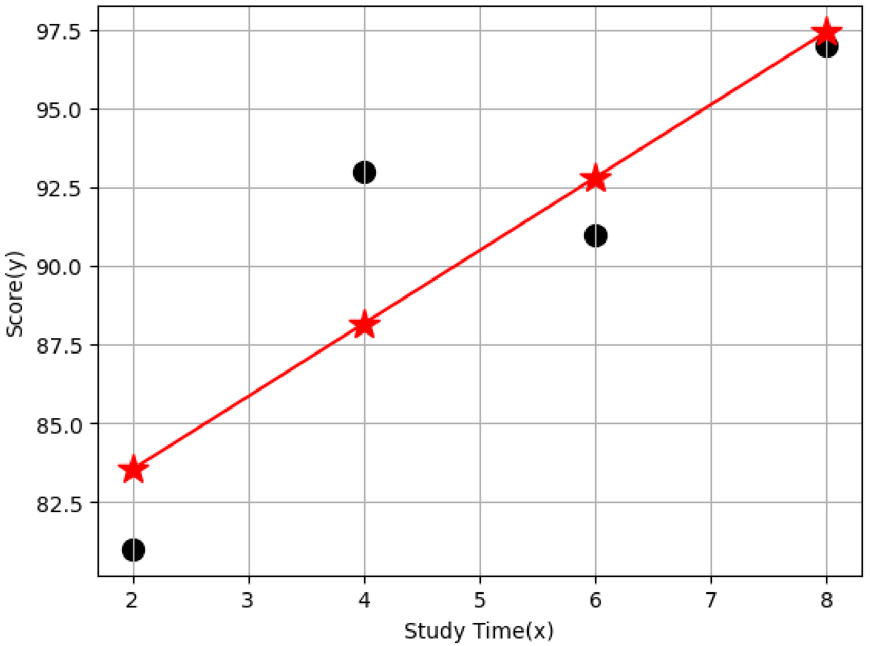
```
y_pred = model.predict(x)

# 예측 결과를 그래프로 나타냅니다.
plt.figure()
plt.scatter(x, y, marker=‘o’, c=‘k’, s=100)
plt.plot(x, y_pred, marker=‘*’, c=‘r’, markersize=15)
plt.xlabel(“Study Time(x)”)
plt.ylabel(“Score(y)”)
plt.grid(True)
plt.show()

# 임의의 시간을 입력하여 학습 완료된 모델이 예측한 시험 점수를 확인합니다.
hour = 7
prediction = model.predict([hour])

print(“%.f시간을 공부할 경우의 예상 점수는 %.2f점입니다.” % (hour, prediction))
```

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 선형 회귀 모델 구현하기 (12/12)

실행결과

```
Epoch 1/2000 1/1 [==============================] - 0s 311ms/step - loss: 7661.8359
… (중략) …
Epoch 2000/2000 1/1 [==============================] - 0s 7ms/step - loss: 8.3022
```




```
7시간을 공부할 경우의 예상 점수는 95.12점입니다.
```

텐서플로로 실행한 선형 회귀 분석 결과

- ✓ 앞서 구한 선형 회귀 결과와 같은 그래프를 구했음
- ✓ 임의의 시간을 넣었을 때 예상되는 점수를 보여 줌

❖ 텐서플로를 활용하여 다중 선형 회귀 모델 구현하기 (1/5)

- 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 함수의 shape에 (2,)로 설정해 줌

```
model.add(Input(shape=(2,)))
```

❖ 텐서플로를 활용하여 다중 선형 회귀 모델 구현하기 (2/5)

- 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 = 3
result = model.predict([[hour, private_class]])

print("%.f시간을 공부를 하고 %.f시간의 과외를 받을 경우, 예상 점수는 %.f입니다."
      % (hour, private_class, result))
```

❖ 텐서플로를 활용하여 다중 선형 회귀 모델 구현하기 (3/5)

- 모든 코드를 정리하면 다음과 같음

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

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

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

05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

❖ 텐서플로를 활용하여 다중 선형 회귀 모델 구현하기 (4/5)

```
model = Sequential()

# 입력 변수가 두 개(학습 시간, 과외 시간)이므로 shape을 (2,)로 설정합니다.
model.add(Input(shape=(2,)))
model.add(Dense(1, activation=“linear”))  # y = w1 * x1 + w2 * x2 + b

model.compile(loss=“mse”, optimizer=“sgd”)

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

# 임의의 학습 시간과 과외 시간을 입력하여 학습 완료된 모델이 예측한 시험 점수를 확인합니다.
hour = 7
private_class = 3
result = model.predict([[hour, private_class]])

print(“%.f시간을 공부를 하고 %.f시간의 과외를 받을 경우, 예상 점수는 %.f입니다.”
      % (hour, private_class, result))
```

❖ 텐서플로를 활용하여 다중 선형 회귀 모델 구현하기 (5/5)

실행결과

```
Epoch 1/2000 1/1 [==============================] - 1s 823ms/step - loss: 9241.6133
Epoch 2/2000 1/1 [==============================] - 0s 20ms/step - loss: 1507.2310
… (중략) …
Epoch 1999/2000 1/1 [==============================] - 0s 6ms/step - loss: 0.0743
Epoch 2000/2000 1/1 [==============================] - 0s 13ms/step - loss: 0.0742

7시간을 공부를 하고 3시간의 과외를 받을 경우, 예상 점수는 95입니다.
```

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

끝맺음

❖ 01. 경사 하강법의 개요

❖ 02. 파이썬 코딩으로 확인하는 선형 회귀

❖ 03. 다중 선형 회귀의 개요

❖ 04. 파이선 코딩으로 확인하는 다중 선형 회귀

❖ 05. 텐서플로에서 실행하는 선형 회귀, 다중 선형 회귀 모델

THANK YOU!
Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: bkwon@dongduk.ac.kr