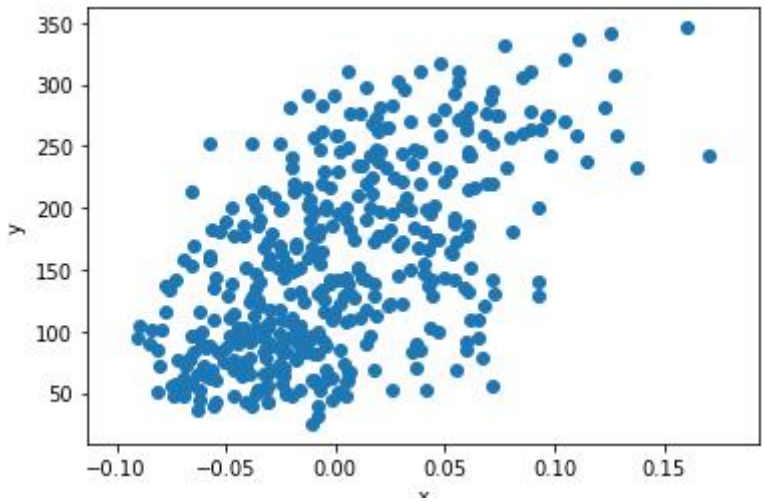
디지털 영상처리 연구실 연구보고서

정지우

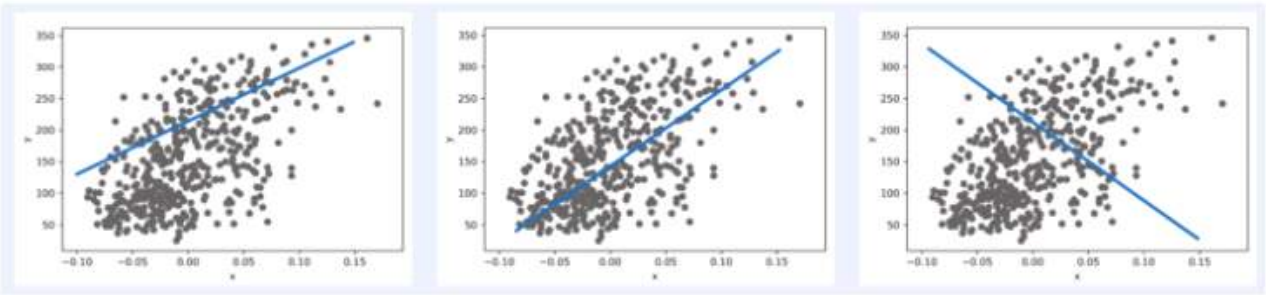
선형 회귀란?

- 선형 회귀는 알려진 다른 데이터 값을 활용하여 알 수 없는 데이터의 값을 예측하는 데이터 분석 기법



이러한 데이터가 있을 때, 가장 잘 나타낸 1차 함수를 찾는 것

=> 경사 하강법



만일 특성이 3차원~10차원 까지 나온다면, 특성의 개수를 여러개로 늘려 여러 차원을 관통하는 직선을 그리는 것이 아니라 특성의 개수를 2~3개로 줄여서 2차원 또는 3차원으로 나타낸다.

1. 선형 회귀에 대해 알아보고 데이터 준비

```
[ ]  from sklearn.datasets import load_diabetes
     diabetes = load_diabetes()
```

diabetes 에 당뇨병 환자의 데이터세트를 딕셔너리 처럼 저장함.

```
[ ]  print(diabetes.data.shape, diabetes.target.shape)
     # data는 442x10의 2차원 배열이고, target은 442개의 1차원 배열이다.
```

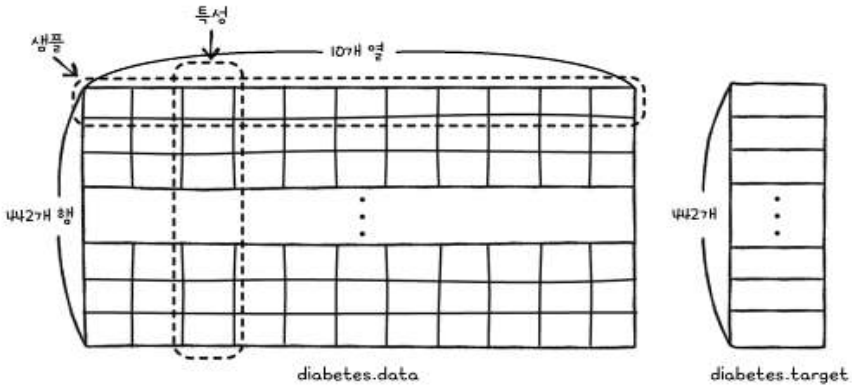
```
(442, 10) (442,)
```

```
[ ]  # 입력 데이터 자세히 보기
     diabetes.data[0:3]
```

```
array([[ 0.03807591,  0.05068012,  0.06169621,  0.02187239, -0.0442235 ,
        -0.03482076, -0.04340085, -0.00259226,  0.01990749, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, -0.02632753, -0.00844872,
        -0.01916334,  0.07441156, -0.03949338, -0.06833155, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, -0.00567042, -0.04559945,
        -0.03419447, -0.03235593, -0.00259226,  0.00286131, -0.02593034]])
```

```
[ ]  # 타겟 데이터 자세히 보기
     diabetes.target[:3]
```

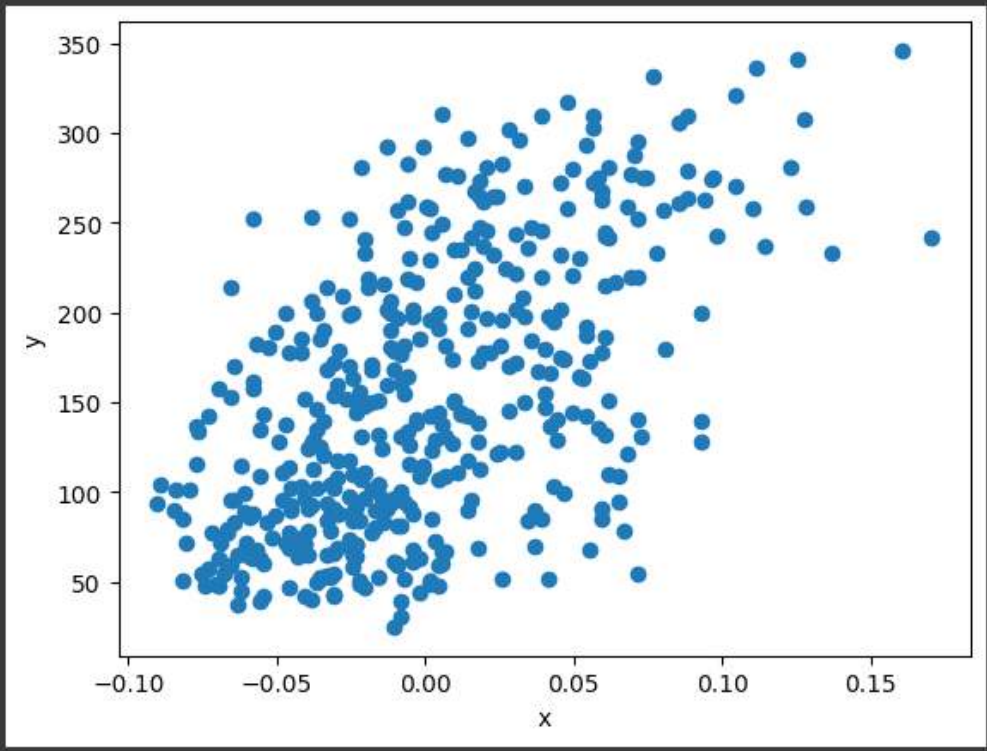
```
array([151.,  75., 141.])
```



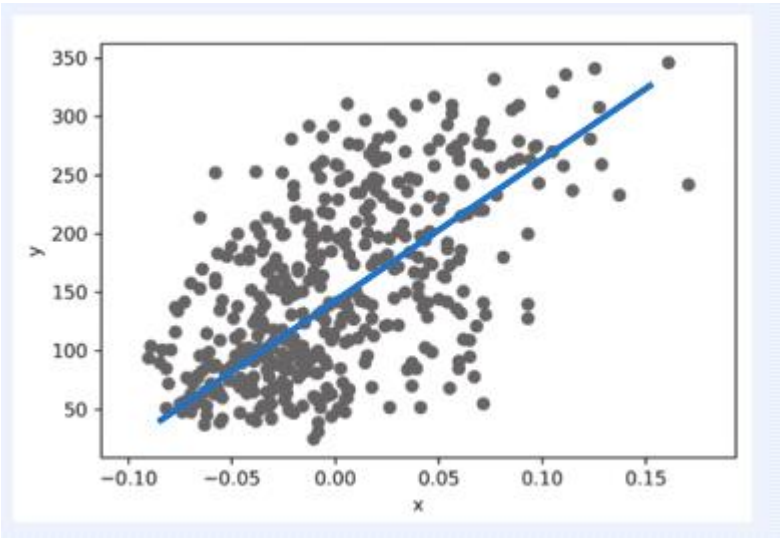
첫 번째 샘플(환자)　　네 번째 특성

```
array([[ 0.03807591,  0.05068012,  0.06169621,  0.02187235, -0.0442235 ,
        -0.03482076, -0.04340085, -0.00259226,  0.01990842, -0.01764613],
       [-0.00188202, -0.04464164, -0.05147406, -0.02632783, -0.00844872,
        -0.01916334,  0.07441156, -0.03949338, -0.06832974, -0.09220405],
       [ 0.08529891,  0.05068012,  0.04445121, -0.00567061, -0.04559945,
        -0.03419447, -0.03235593, -0.00259226,  0.00286377, -0.02593034]])
```

```
# 맷플롯립으로 산점도 그리기
plt.scatter(diabetes.data[:,2], diabetes.target)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



경사 하강법



만일 위 그림과 같은 직선이 있을 때,
x = 0.15 정도면, y = 300 정도를 예측할 것이다.

컴퓨터가 이러한 직선을 예측하는 방법은,

1. 무작위로 w와 b를 초기화 합니다.
2. x에서 샘플 하나를 선택하여 $\hat{y}$를 계산합니다.
3. 예측값 $\hat{y}$과 실제값 y를 비교합니다.
4. 예측값과 실제값의 오차가 줄어들도록 w와 b를 조정합니다.

2. 경사 하강법으로 학습하는 방법을 알아보자

경사 하강법은 모델이 데이터를 잘 표현할 수 있도록 기울기를 사용하여 모델을 조금씩 조정하는 최적화 알고리즘이다.

```
[ ] # w(가중치), b(절편)을 무작위로 초기화
    w = 1.0
    b = 1.0
```

```
[ ] # 훈련 데이터의 첫 샘플 데이터로 y얻기
    y_hat = x[0]*w + b
    print(y_hat)
```

```
1.0616962065186832
```

```
[ ] # 타겟과 예측 데이터 비교하기
    print(y[0])
```

```
151.0
```

```
[ ] # w값 조절해 예측값 바꾸기
    w_inc = w + 0.1
    y_hat_inc = x[0]*w_inc + b
    print(y_hat_inc)
```

```
1.0678658271705517
```

```
[ ] # w값 조정한 후 예측값 증가정도 확인
    w_rate = (y_hat_inc - y_hat) / (w_inc - w)
    print(w_rate)
```

```
0.06169620651868429
```

$$w\_rate = \frac{y\_hat\_inc - y\_hat}{w\_inc - w} = \frac{(x[0]*w\_inc + b) - (x[0]*w + b)}{w\_inc - w}$$

$$= \frac{x[0]*((w+0.1) - w)}{(w+0.1) - w} = x[0]$$

w값의 변화율에 관한 예측y값의 변화율은 초기 데이터 x이다.

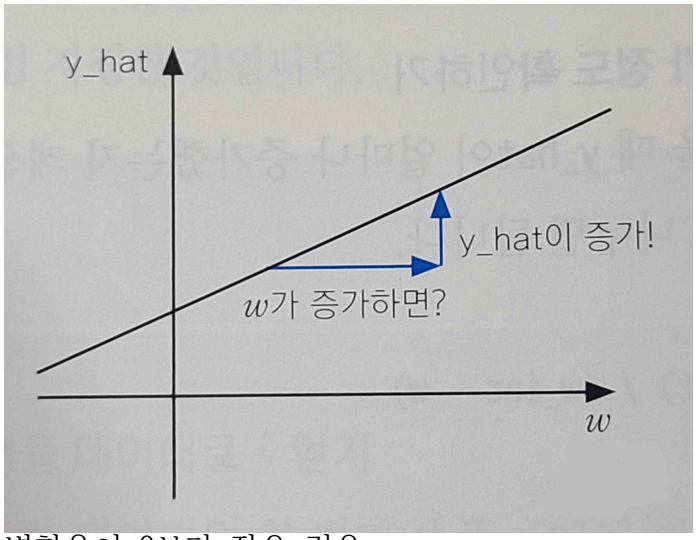
정리하면, 임의의 1차함수에 관한 예측값인 y_hat(1.061~)의 값이 실제값인 y(151) 보다 작아서 y_hat의 값을 증가시켜야합니다.

이때, 변화율(w_rate)은 양수이므로 w값을 증가시키면, y_hat의 값을 증가할 수 있습니다.

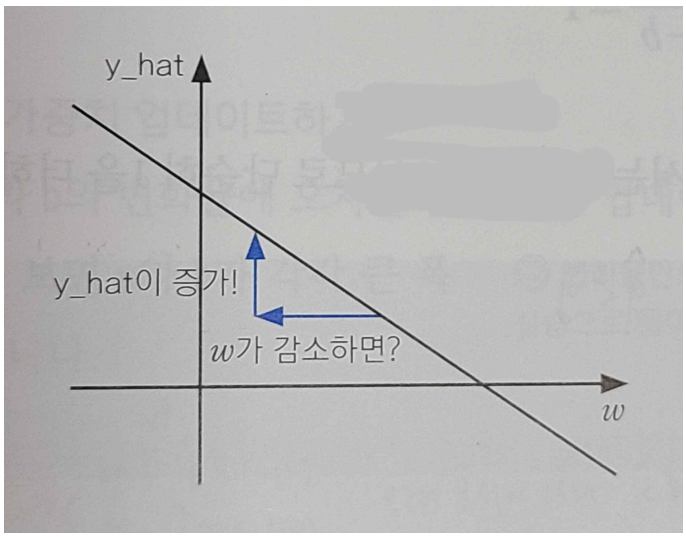
만일 변화율이 음수일 때, y_hat의 값을 증가시켜야한다면, w 값을 감소시키면 됩니다.

=> 변화율이 양수이든 음수이든 y_hat의 값을 증가시킬 수 있는 방법은

변화율이 0보다 큰 경우,



변화율이 0보다 작은 경우,



즉, 변화율(w_rate)를 w에다가 더하면 음수든 양수든 무조건 y_hat은 증가한다.

```
[ ]  # 변화율이 음수든 양수든 더해주면 y_hat 값이 증가
     w_new = w + w_rate
     w_new

     1.0616962065186843
```

즉, w_new = w + x[i] 인 것이다.

이번에는 b(절편)을 업데이트

```
# b 절편에 대한 변화율을 구한다음 업데이트
b_inc = b +0.1
y_hat_inc = x[0]*w+b_inc
print(y_hat_inc)
```

```
1.1616962065186833
```

```
[233] b_rate = (y_hat_inc - y_hat) / (b_inc - b)
      print(b_rate)
      # => 즉, b가 1만큼 증가하면 y_hat 도 1만큼 증가한다.
```

```
1.0
```

```
[234] b_new = b+1
      print(b_new)
```

```
2.0
```

$$b\_rate = \frac{y\_hat\_inc - y\_hat}{b\_inc - b} = \frac{(x[0]*w + b\_inc) - (x[0]*w + b)}{b\_inc - b}$$

$$= \frac{(b+0.1) - b}{(b+0.1) - b} = 1$$

b_rate 또한 무조건 1이 되는걸 알 수 있다.

하지만, 이러한 w,b의 업데이트 방법은 수동적이다.

1. y_hat이 y에 한참 못미치는 경우 큰폭으로 수정이 안됨
2. y_hat이 y보다 커지면 줄이지를 못함

=> 그래서 오차 역전파로 가중치와 절편을 더 적절히 업데이트

오차 역전파란, y와 y_hat의 차이가 크거나 역전되어서 감소시켜야 할 때,
오차(y - y_hat)를 변화율(w_rate = x, b_rate = 1)에 곱하여서 업데이트한다.

즉, 가중치와 절편을 업데이트 할 때

w = w + w_rate(x) => w = w + w_rate * 오차(err)

b = b + b_rate(1) => b = b + b_rate * 오차(err)

오차로 인해 차이가 크면 더 많이 더할 수 도 있고, y_hat이 y를 지나치면 방향도 바꿀 수 있음

```
# 1. 오차와 변화율을 곱하여 가중치 업데이트
err = y[0] - y_hat
w_new = w + w_rate * err
b_new = b + 1 * err
print(w_new, b_new)
```

```
10.250624555903848 150.9383037934813
```

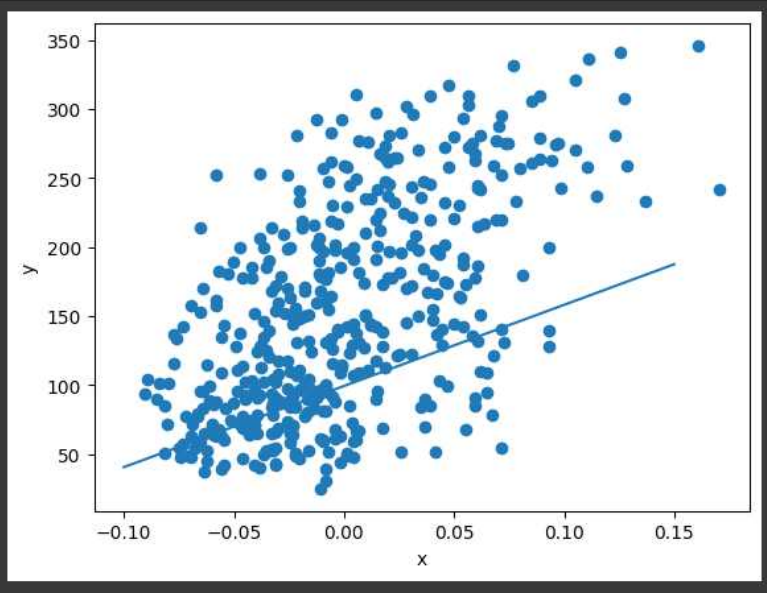
```
# 2. 두번째 샘플인 x[1]을 사용하여 오차를 구하고 새로운 w와 b를 구해본다.
y_hat = x[1] * w_new +b_new
err = y[1] - y_hat
w_rate = x[1] # 변화율은 샘플값과 같음
w_new = w_new + w_rate * err
b_new = b_new + 1 * err
print(w_new, b_new)
```

```
14.132317616380695 75.52764127612656
```

```
# 3. 전체 샘플을 반복하기
for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_i - y_hat
    w_rate = x_i
    w = w + w_rate * err
    b = b + 1 * err
print(w, b)
```

```
587.8654539985616 99.4093556453094
```

```
# 4. 산점도로 확인
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

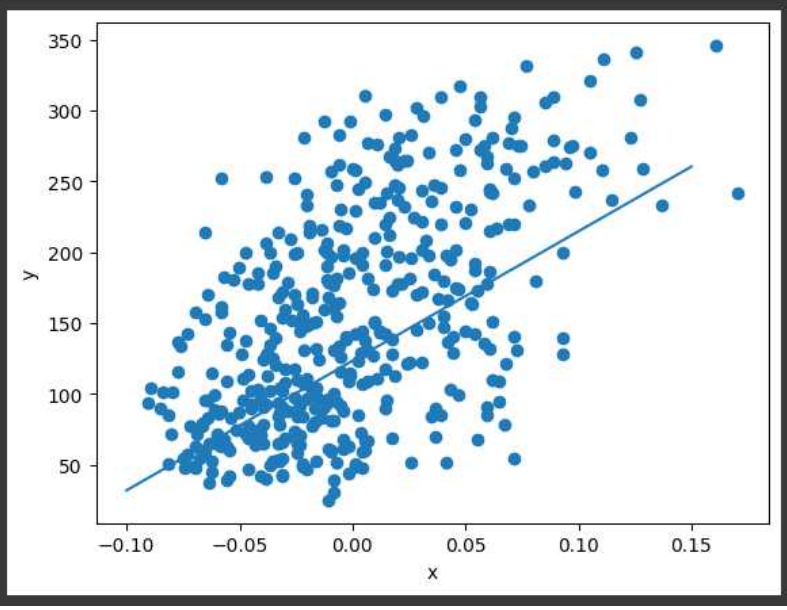


```python
# 5. 여러 에포크 반복하기
# 에포크: 주어진 훈련 데이터로 학습을 한번하는것을 에포크라고 한다.
# 100번 반복할것이다.

for i in range(1,100):
  for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_i - y_hat
    w_rate = x_i
    w = w + w_rate * err
    b = b + 1 * err
print(w, b)
```

```
913.5973364346786 123.39414383177173
```

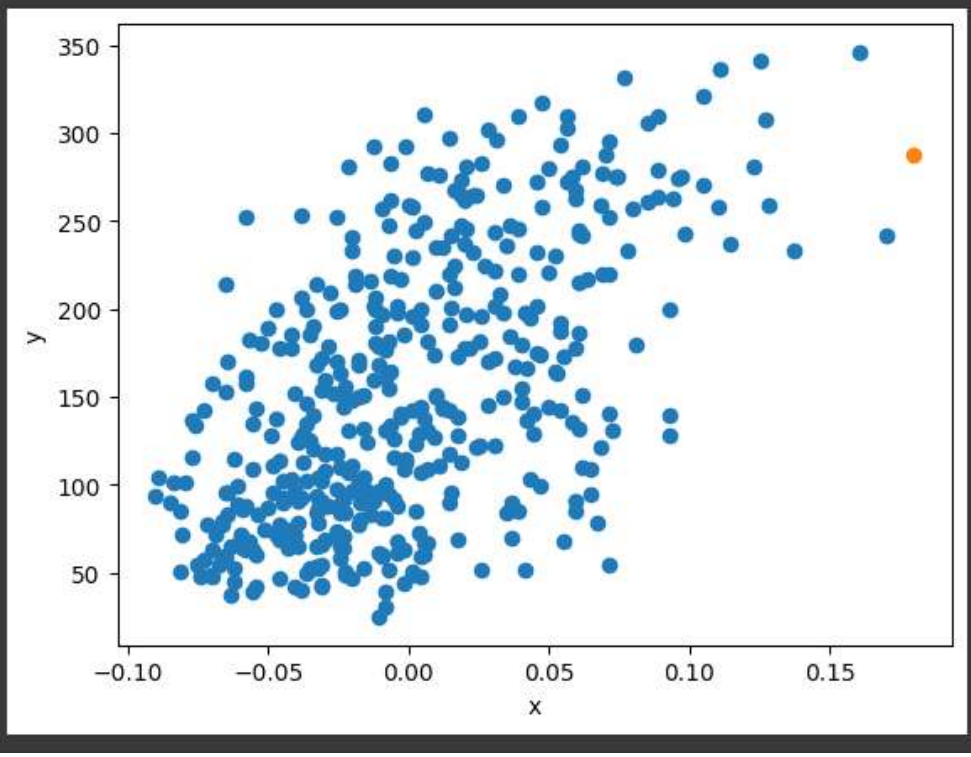
```python
plt.scatter(x, y)
pt1 = (-0.1, -0.1 * w + b)
pt2 = (0.15, 0.15 * w + b)
plt.plot([pt1[0], pt2[0]], [pt1[1], pt2[1]])
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



```
# 6. 모델로 예측하기
x_new = 0.18
y_pred = x_new * w + b
print(y_pred)
```

```
287.8416643900139
```

```
plt.scatter(x, y)
plt.scatter(x_new, y_pred)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```

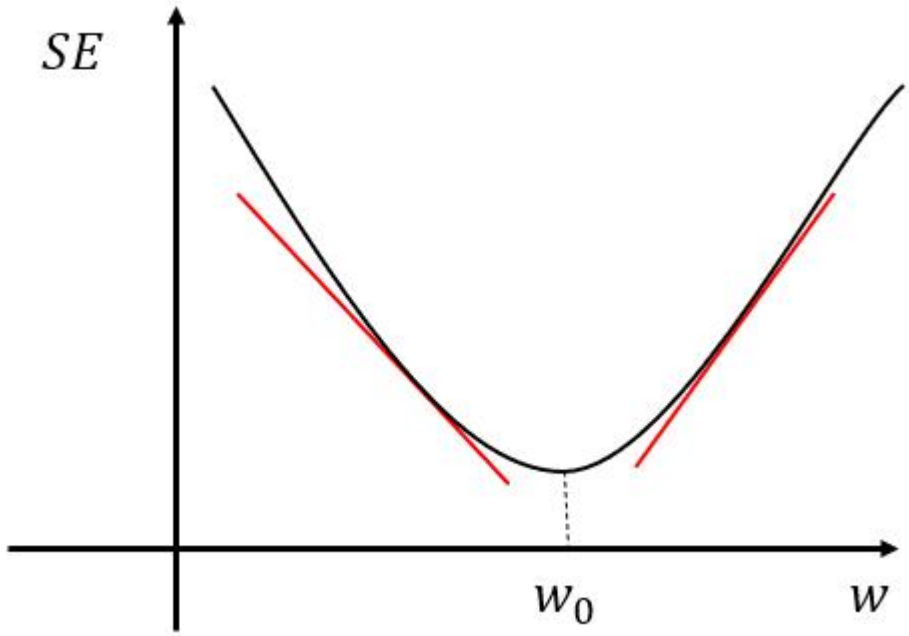


경사 하강법을 기술적인 말로 하면,
"어떤 손실 함수(loss function)가 정의되었을 때, 손실 함수의 값이 최소가 되는 지점을 찾아가는 방법“

위에서 사용한 손실함수는 "제곱 오차"라는 손실 함수를 사용하였습니다.

$$SE = (y - \hat{y})^2$$

단순히, 실제값에서 예측값을 뺀 것을 제곱한 형태입니다.



예상값(y_hat)과 실제값(y) 사이의 차이가 작아져서 오차(err)의 값이 안정이 되고, 이때의 직선을 찾고 싶은거기 때문에

즉, 제곱오차 라는 손실함수의 기울기가 0이 되는 지점을 찾아야 합니다.

$$\frac{\partial SE}{\partial w} = \frac{\partial}{\partial w}(y - \hat{y})^2 = 2(y - \hat{y})(-\frac{\partial}{\partial w}\hat{y}) = 2(y - \hat{y})(-x) = -2(y - \hat{y})x$$

보통 제곱오차를 미분하면 앞에 2라는 상수항 때문에 일반적으로

$\frac{1}{2}(y - \hat{y})$

위와 같이 제곱오차를 정의합니다.

즉, 아래 식과 같이 나오게 되고,

$$w = w - \frac{\partial SE}{\partial w} = w + (y - \hat{y})x$$

(이전에 사용한 수식) (오차 = y - y_hat)

즉, 가중치와 절편을 업데이트 할 때

w = w + w_rate(x) => w = w + w_rate * 오차(err)

b = b + b_rate(1) => b = b + b_rate * 오차(err)

••오차로 인해 차이가 크면 더 많이 더할 수 도 있고, y_hat이 y를 지나치면 방향도 바꿀 수 있음••

이렇게 오차 역전파를 사용한 코드와 같다는 것을 통해 수학적으로 증명됬음을 알 수 있다.

또한 절편도

$$\frac{\partial SE}{\partial b} = \frac{\partial}{\partial b}\frac{1}{2}(y - \hat{y})^2 = 2(y - \hat{y})(-\frac{\partial}{\partial b}\hat{y}) = (y - \hat{y})(-1) = -(y - \hat{y})$$

=> b = b - $\frac{\partial SE}{\partial b}$ = b + (y - y_hat)