머신러닝 지도 학습

박경미

목차

❖ Machin Learning Type

❖ Supervised Learning

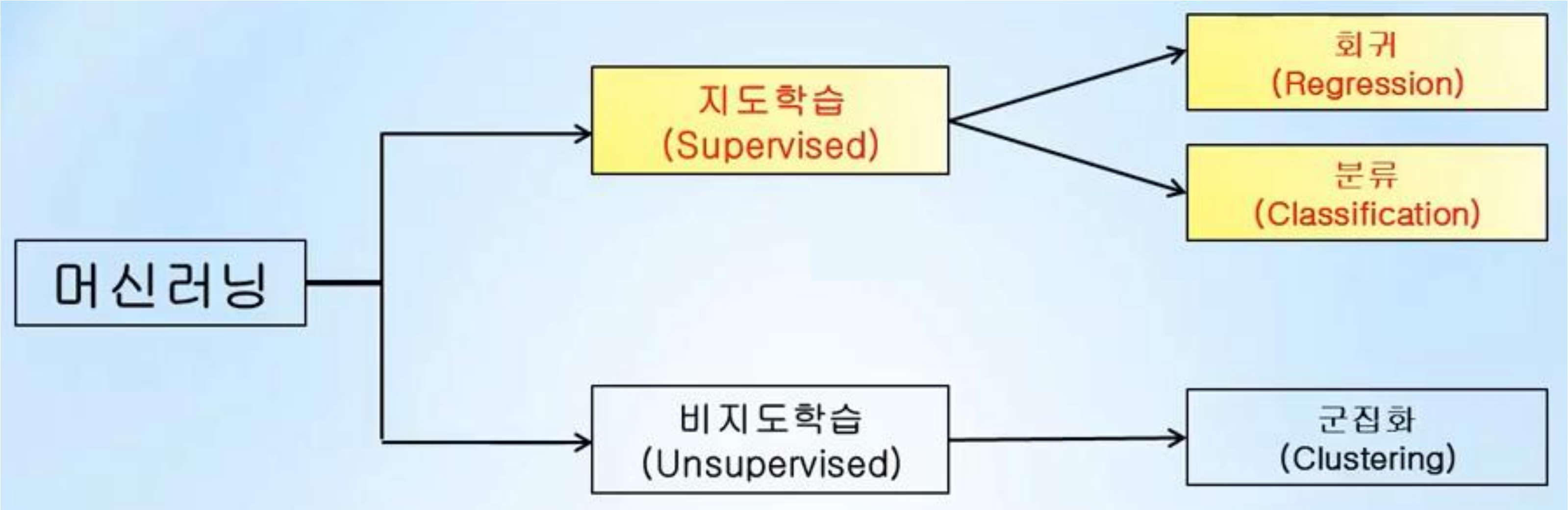
❖ Unsupervised Learning

❖ Linear Regression

Machin Learning Type

❖ **지도학습(Supervised Learning)**

❖ **비지도학습(Unsupervised Leanrning)**



Supervised Learning

❖ 지도학습

- **입력 값(x)**과 **정답(t, label)**을 포함하는 **Training Data**를 이용하여 **학습**을 하고, 그 학습된 결과를 바탕으로 미지의 데이터(Test Data)에 대해 미래 값을 예측(predict) 하는 방법 ->대부분의 머신러닝 문제는 지도학습에 해당됨
- 예1 : 시험공부 시간(입력)과 Pass/Fail(정답)을 이용하여 당락 여부를 예측
- 예2 : 집 평수(입력)와 가격 데이터(정답)을 이용하여 임의의 평수 가격 예측

Supervised Learning

❖ Regression, Classification

▪ 회귀(Regression)

- Training Data를 이용하여 연속적인(숫자) 값을 예측하는 것을 의미, 집 평수와 가격 관계, 공부 시간과 시험 성적 등의 관계

▪ 분류(Classification)

- Training Data를 이용하여 주어진 입력 값이 어떤 종류 값인지 구별하는 것

Regression

공부시간 (x)	시험성적 (t)
9	74
14	81
21	86
27	90
32	88
37	92

집평수 (x)	가격 (t)
20	98
25	119
30	131
40	133
50	140
55	196

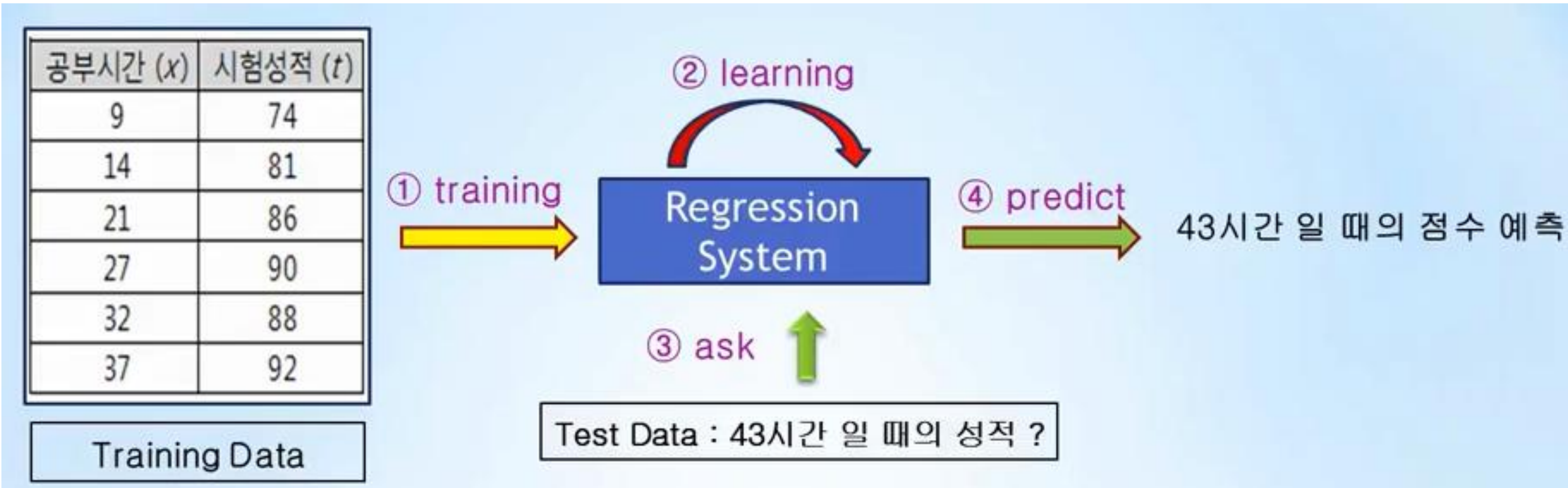
Classification

공부시간 (x)	시험성적 (t)
9	Fail
14	Fail
21	Pass
27	Pass
32	Pass
37	Pass

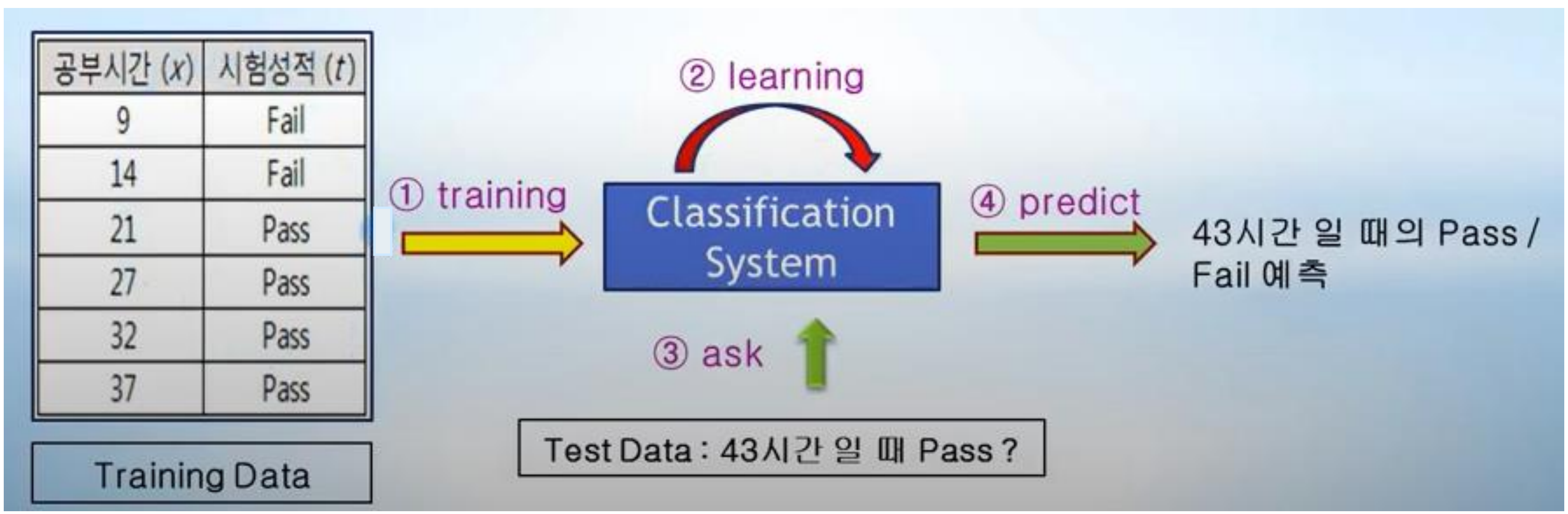
집평수 (x)	가격 (t)
20	Low
25	Low
30	Medium
40	Medium
50	Medium
55	High

Supervised Learning

❖ Regression, Classification



공부시간 (x)	시험성적 (t)
9	74
14	81
21	86
27	90
32	88
37	92

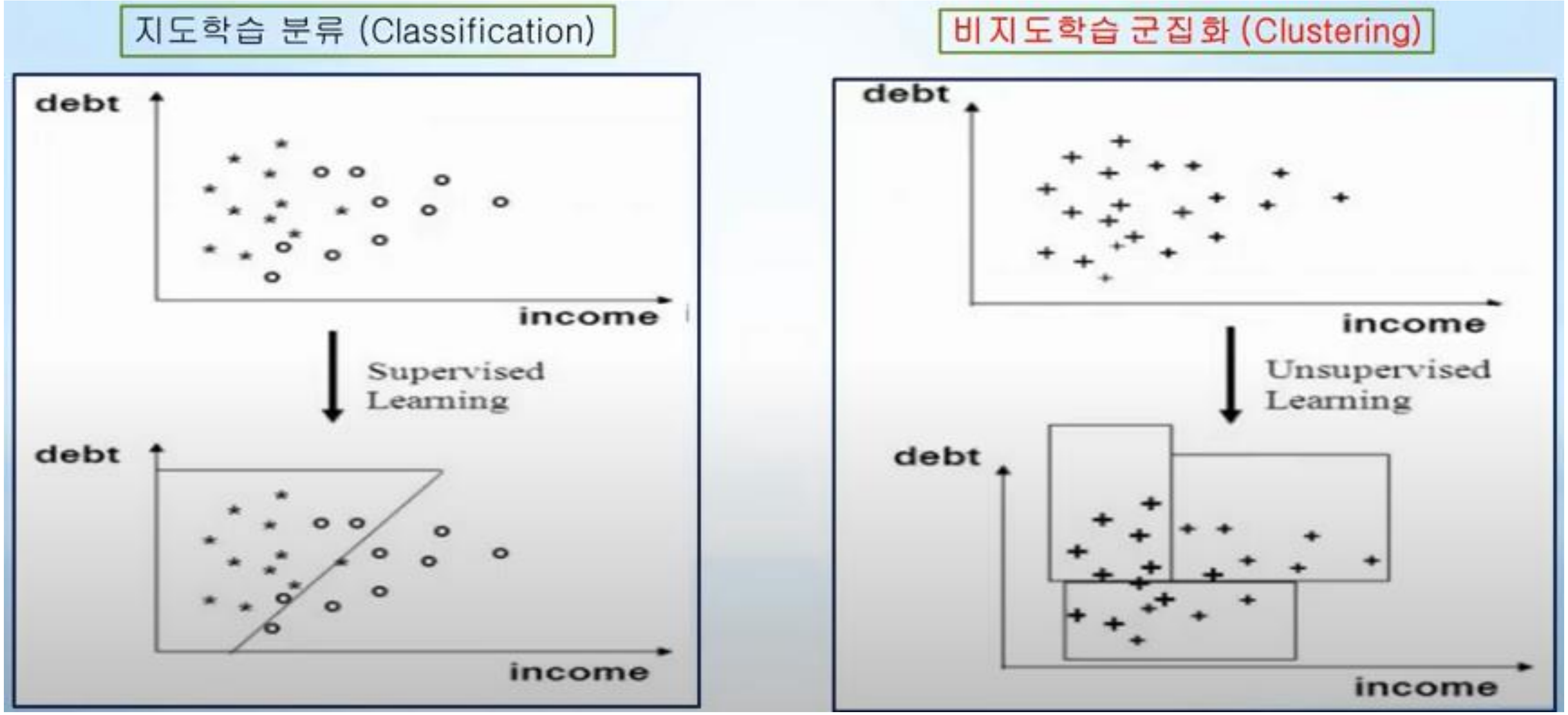


공부시간 (x)	시험성적 (t)
9	Fail
14	Fail
21	Pass
27	Pass
32	Pass
37	Pass

Unsupervised Learning

❖ 비지도 학습

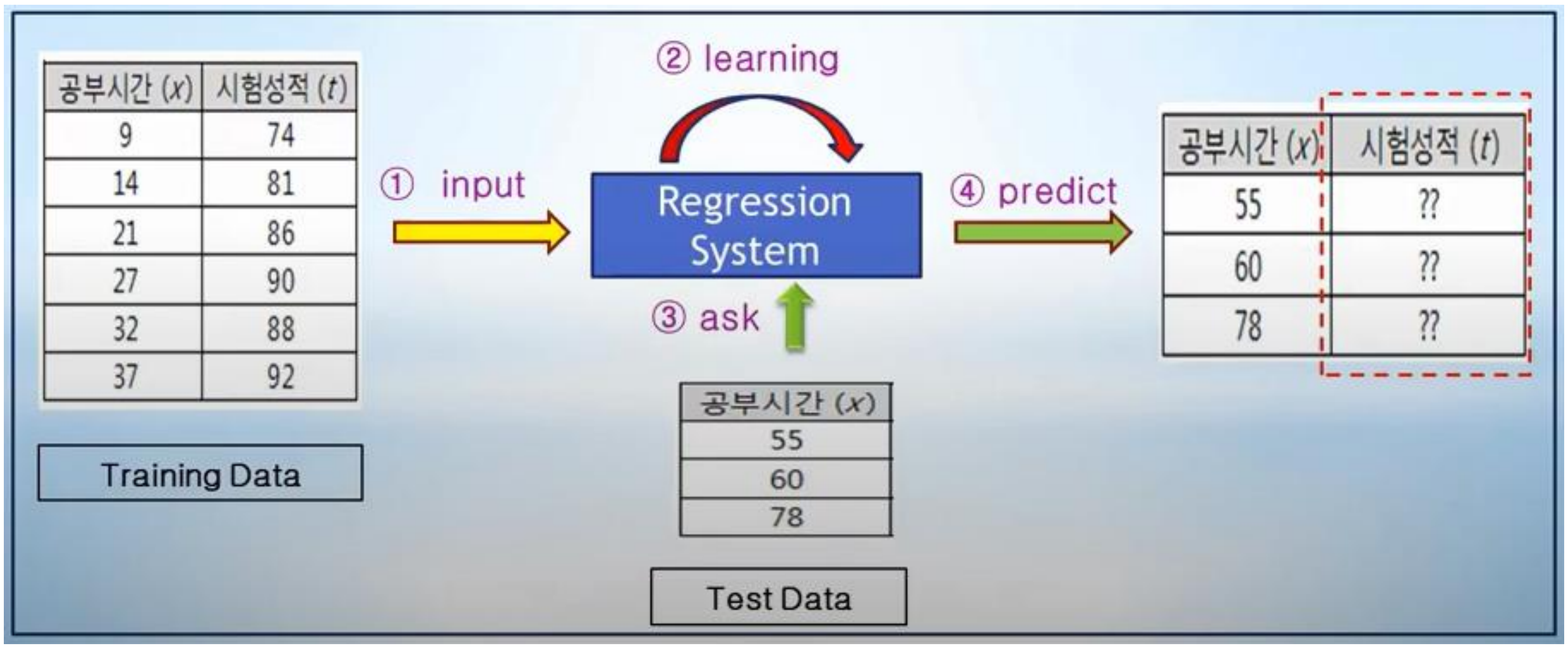
- 트레이닝 데이터에 정답은 없고 입력 데이터만 있기 때문에, 입력에 대한 정답을 찾는 것이 아닌 입력데이터의 패턴, 특성 등을 학습을 통해 발견하는 방법
- 예 : 군집화(Clustering) 알고리즘을 이용한 뉴스 그룹핑, 백화점의 상품 추천 시스템 등



Linear Regression

❖ 회귀(Regression)

- **Training Data를 이용하여 데이터의 특성과 상관관계 등을 파악하고, 그 결과를 바탕으로 Training Data에 없는 미지의 데이터가 주어졌을 경우에, 그 결과를 연속적인 (숫자) 값으로 예측하는 것**
- **예: 공부 시간과 시험성적 관계, 집 평수와 집 가격 관계 등**



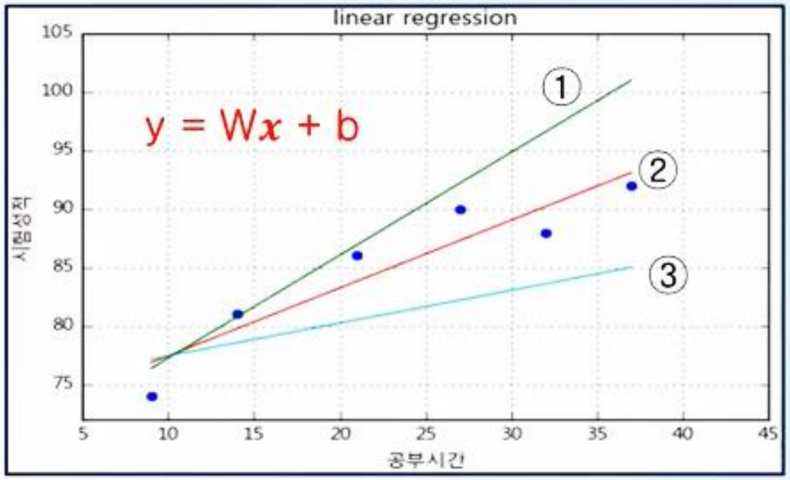
❖ Regression-학습(learning) 개념

[step1] analyze training data

공부시간 (x)	시험성적 (t)
9	74
14	81
21	86
27	90
32	88
37	92

- 학습데이터(training data)는 입력(x)인 공부시간에 비례해서 출력(y)인 시험성적도 증가하는 경향이 있음
- 즉, 입력(x)과 출력(y) 은 $y = Wx + b$ 형태로 나타낼 수 있음

[step2] find W and b



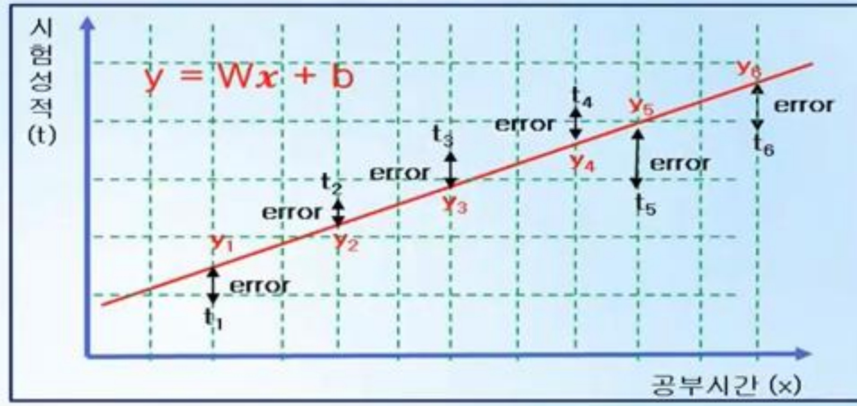
- ①, ②, ③,… 등의 다양한 $y = Wx + b$ 직선 中,
- training data의 특성을 가장 잘 표현할 수 있는 가중치 W (기울기), 바이어스 b (y절편)를 찾는 것이 학습(Learning) 개념임

> ※ 머신러닝에서는, 기울기 W는 가중치(weight), y 절편 b 는 바이어스(bias) 라고 함

Regression – 오차(error), 가중치(weight)W, 바이어스(bias) b

공부시간 (x)	시험성적 (t)
9	74
14	81
21	86
27	90
32	88
37	92

오차(error) = t - y



- training data의 정답(t)과 직선 y = Wx+b 값의 차이인 오차(error)는,

  오차(error) = t − y = t − (Wx+b) 으로 계산되며,

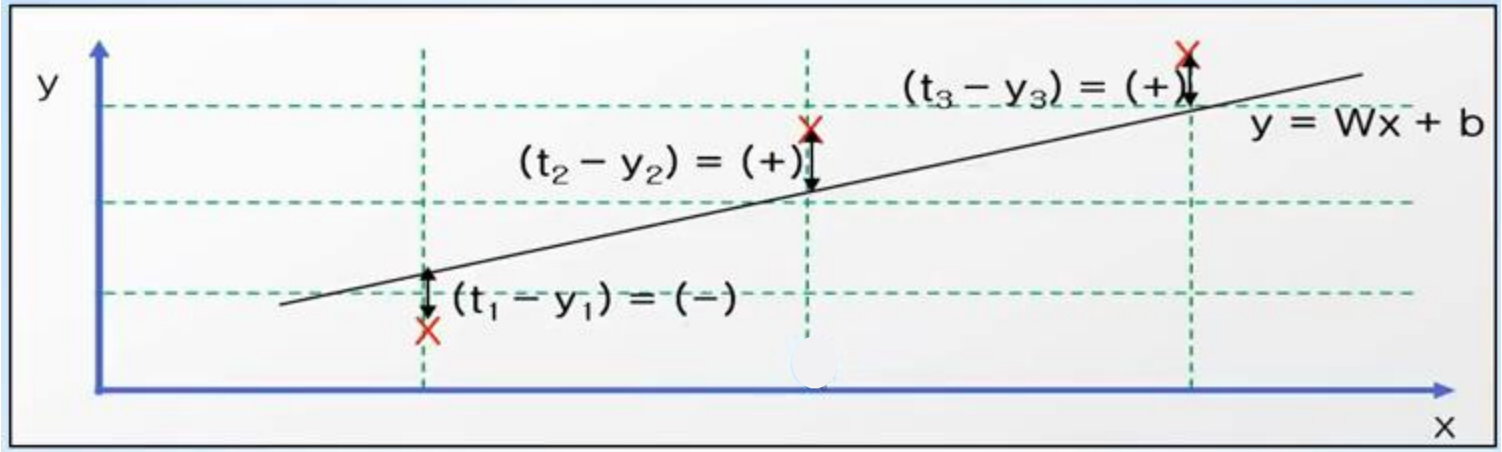
  오차가 크다면, 우리가 임의로 설정한 직선의 가중치와 바이어스 값이 잘못된 것이고, 오차가 작다면 직선의 가중치와 바이어스 값이 잘 된 것이기 때문에 미래 값 예측도 정확할 수 있다고 예상할 수 있음

- 머신러닝의 regression 시스템은,

  모든 데이터의 오차(error) = t − y = t − (Wx+b) 의 합이 최소가 되서,
  미래 값을 잘 예측할 수 있는 가중치 W 와 바이어스 b 값을 찾아야 함

❖ 손실함수(loss function, cost function)

- Training data의 정답(t)과 입력(x)에 대한 계산 값 y의 차이를 모두 더해 수식으로 나타낸 것



➢ 각각의 오차인 (t-y) 를 모두 더해서 손실함수(loss function)을 구하면 각각의 오차가 (+), (-) 등이 동시에 존재하기 때문에 오차의 합이 0 이 나올 수도 있음. 즉, 0 이라는 것이 최소 오차 값인지 아닌지를 판별하는 것이 어려움

➢ 손실함수에서 오차(error)를 계산할 때는 $(t-y)^2 = (t-[Wx+b])^2$ 을 사용함.
즉 오차는 언제나 양수이며, 제곱을 하기때문에 정답과 계산값 차이가 크다면, 제곱에 의해 오차는 더 큰 값을 가지게 되어 머신러닝 학습에 있어 장점을 가짐

Regression-손실함수(loss function)

$$\text{loss function} = \frac{(t_1-y_1)^2 + (t_2-y_2)^2 + \cdots + (t_n-y_n)^2}{n}$$

$$= \frac{[t_1-(Wx_1+b)]^2 + [t_2-(Wx_2+b)]^2 + \cdots + [t_n-(Wx_n+b)]^2}{n}$$

$$= \frac{1}{n} \sum_{i=1}^{n} [t_i-(Wx_i+b)]^2$$

Regression-손실함수(loss function)

$$y = Wx + b$$

$$\text{loss function} = E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - y_i]^2 = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

- x 와 t 는 training data 에서 주어지는 값이므로, 손실함수(loss function)인 *E(W,b)* 는 결국 W 와 b 에 영향을 받는 함수임.

  - *E(W,b)* 값이 작다는 것은 정답(t, target)과 y = Wx+b 에 의해 계산된 값의 평균 오차가 작다는 의미이며,

  - 평균 오차가 작다는 것은 미지의 데이터 x 가 주어질 경우, 확률적으로 미래의 결과값도 오차가 작을 것이라고 추측할 수 있음

  - 이처럼 training data를 바탕으로 손실 함수 *E(W,b)* 가 최소값을 갖도록 (W, b) 를 구하는 것이 (linear) regression model 의 최종 목적임

gradient decent algorithm-review loss function E(W,b)

$$y = Wx + b$$

$$\text{loss function} = E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - y_i]^2 = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

- 손실함수는 오차의 평균값을 나타내기 때문에, 손실함수가 최소값을 갖는다는 것은 실제 정답과 계산 값의 차이인 오차가 최소가 되어, 미지의 데이터에 대해서 결과를 더 잘 예측 할 수 있다는 것을 의미함.

- 이러한 손실함수는 W, b 에 영향을 받기 때문에, 손실함수가 최소가 되는 가중치 W 와 바이어스 b 를 찾는 것이 regression 을 구현하는 최종 목표임

↓

경사하강법
(gradient decent algorithm)

gradient decent algorithm-손실함수(loss function) 계산

- **계산을 쉽게 하고 손실함수의 모양 파악을 위해 E(W,b)에서 b=0으로 가정**

[예] 다음과 같은 Training Data 에서, W 값에 대한 손실함수 E(W,b) 계산

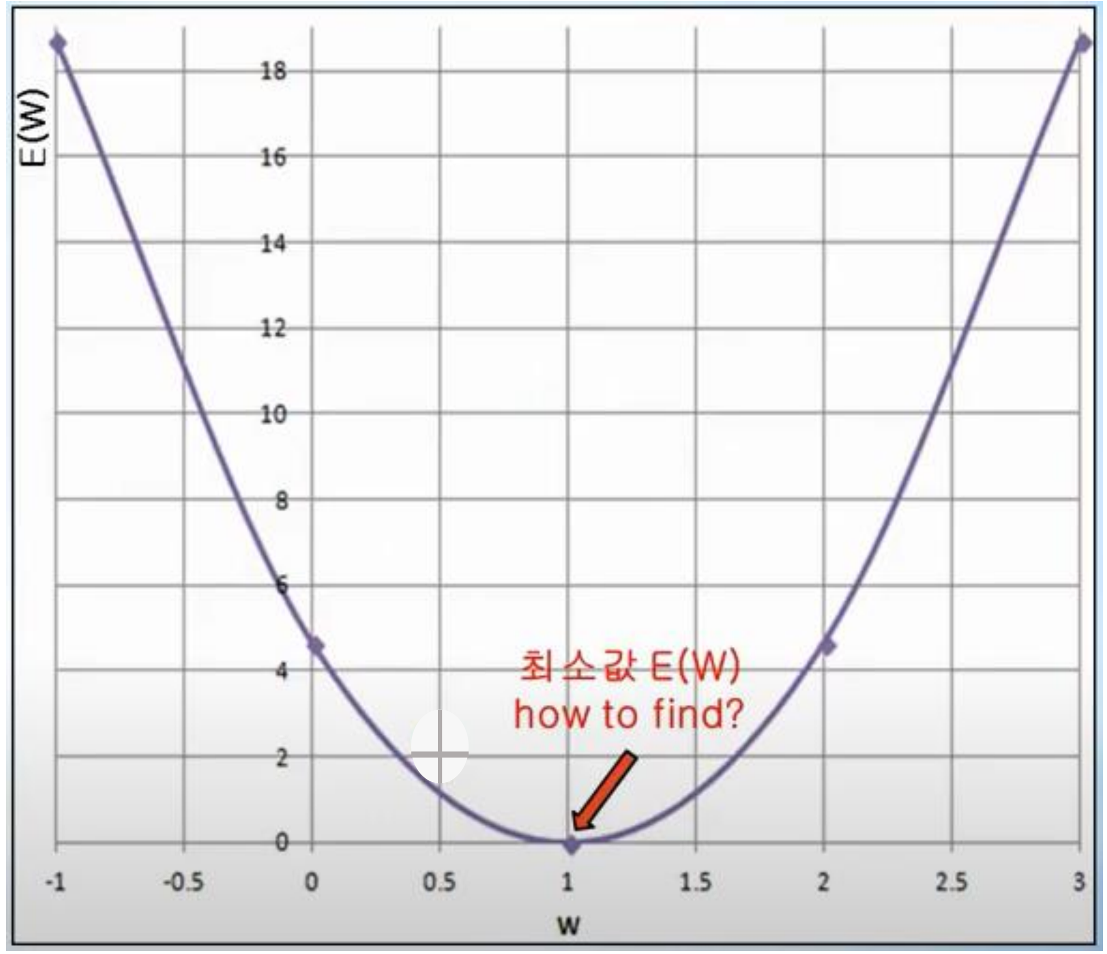
training data

x	t
1	1
2	2
3	3

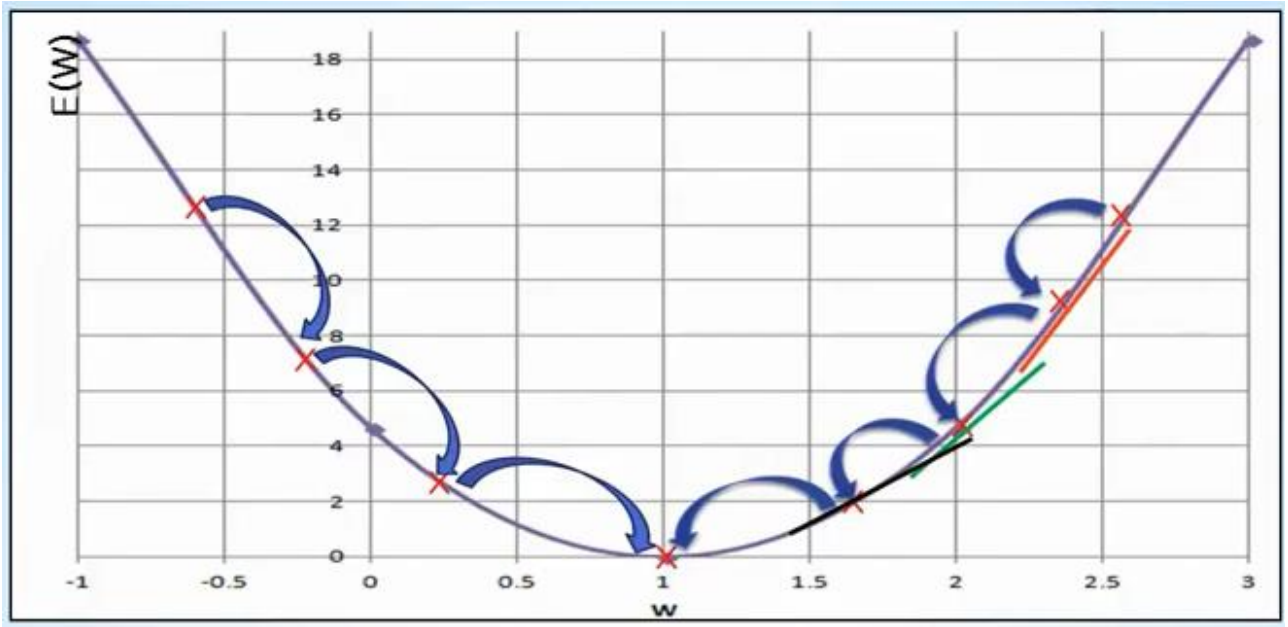
W = -1	$E(-1,0) = \frac{1}{3}\sum_{i=1}^{3}[t_i-(-1\cdot x_i+0)]^2 = \frac{[1-(-1\cdot 1+0)]^2 + [2-(-1\cdot 2+0)]^2 + [3-(-1\cdot 3+0)]^2}{3} = 18.7$
W = 0	$E(0,0) = \frac{1}{3}\sum_{i=1}^{3}[t_i-(0\cdot x_i+0)]^2 = \frac{[1-(0\cdot 1+0)]^2 + [2-(0\cdot 2+0)]^2 + [3-(0\cdot 3+0)]^2}{3} = 4.67$
W = 1	$E(1,0) = \frac{1}{3}\sum_{i=1}^{3}[t_i-(1\cdot x_i+0)]^2 = \frac{[1-(1\cdot 1+0)]^2 + [2-(1\cdot 2+0)]^2 + [3-(1\cdot 3+0)]^2}{3} = 0$
W = 2	$E(2,0) = \frac{1}{3}\sum_{i=1}^{3}[t_i-(2\cdot x_i+0)]^2 = \frac{[1-(2\cdot 1+0)]^2 + [2-(2\cdot 2+0)]^2 + [3-(2\cdot 3+0)]^2}{3} = 4.67$
W = 3	$E(3,0) = \frac{1}{3}\sum_{i=1}^{3}[t_i-(3\cdot x_i+0)]^2 = \frac{[1-(3\cdot 1+0)]^2 + [2-(3\cdot 2+0)]^2 + [3-(3\cdot 3+0)]^2}{3} = 18.7$

gradient decent algorithm-손실함수(loss function) 형태

W	E(W)
-1	18.7
0	4.67
1	0
2	4.67
3	18.7



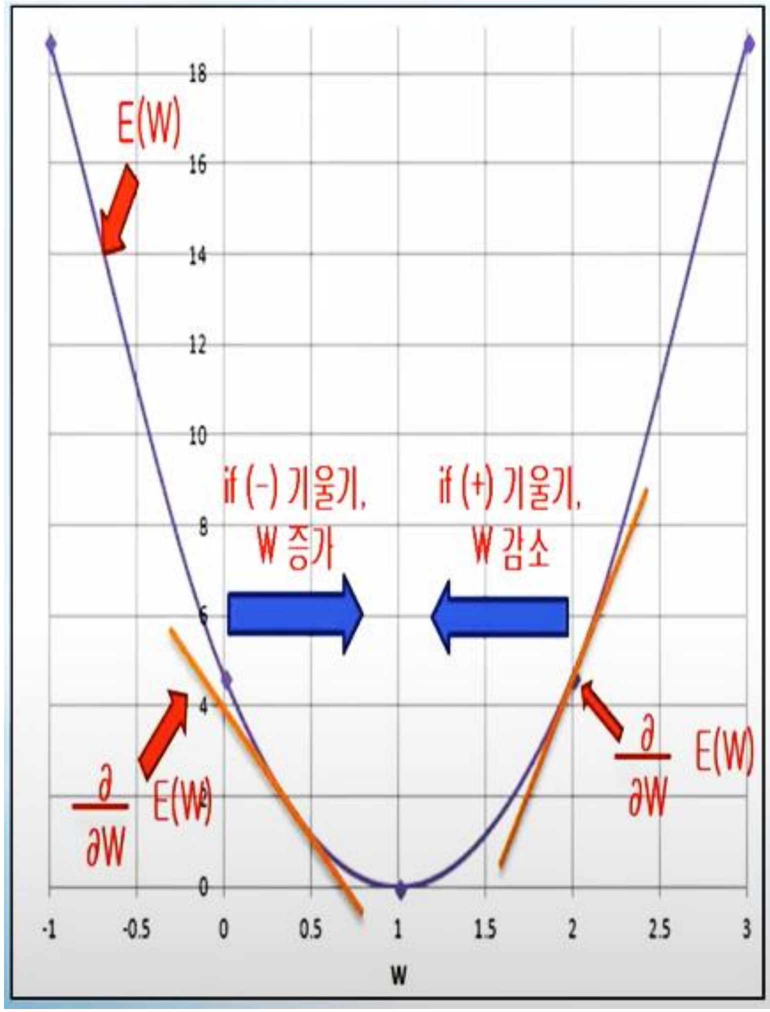
gradient decent algorithm-경사하강법 원리



- ① 임의의 가중치 W 선택 ② 그 W 에서의 직선의 기울기를 나타내는 미분 값 (해당 W 에서의 미분, ∂E(W)/∂W) 을 구함 ③ 그 미분 값이 작아지는 방향으로 W 감소(또는 증가) 시켜 나가면 ④ 최종적으로 기울기가 더 이상 작아지지 않는 곳을 찾을 수 있는데, 그 곳이 손실함수 E(W) 최소값임을 알 수 있음.

- 이처럼, W에서의 직선의 기울기인 미분 값을 이용하여, 그 값이 작아지는 방향으로 진행하여 손실함수 최소값을 찾는 방법을 경사하강법(gradient decent algorithm) 이라고 함

gradient decent algorithm-W 값 구하기



W 에서의 편미분 $\partial E(W)/\partial W$
해당 W 에서 기울기(slope)를 나타냄

⇒ $\partial E(W)/\partial W$ 양수 (+) 값을 갖는다면,
W 는 왼쪽으로 이동시켜야만(감소),
손실함수 E(W) 최소값 찾음

⇒ $\partial E(W)/\partial W$ 음수 (−) 값을 갖는다면,
W 는 오른쪽으로 이동시켜야만(증가),
손실함수 E(W) 최소값 찾음

$$W = W - \alpha \frac{\partial E(W,b)}{\partial W}$$

α 는 학습율(learning rate) 이라고 부르며,
W 값의 감소 또는 증가 되는 비율을 나타냄

gradient decent algorithm-손실함수 E(W,b) 최소 값이 되는 W,b

❖ Linear regression 목표

- **Training data 특성/분포를 가장 잘 나타내는 임의의 직선**

  $y = Wx + b$ 에서의 [W, b] 를 구하는 것

$$y = Wx + b$$

$$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - y_i]^2 = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

손실함수 E(W,b) 최소값을 갖는 W

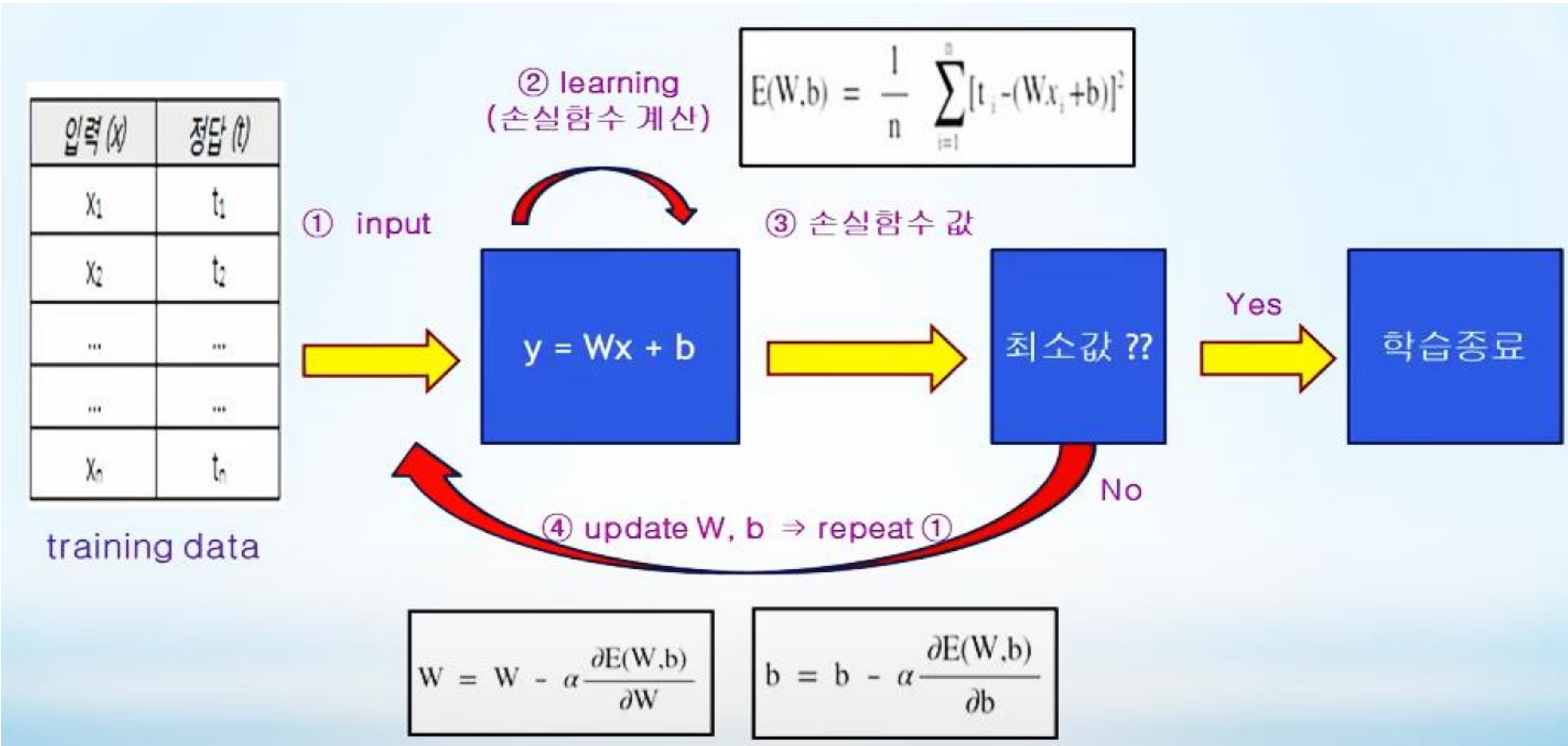
손실함수 E(W,b) 최소값을 갖는 b

$$W = W - \alpha\frac{\partial E(W,b)}{\partial W}$$

$$b = b - \alpha\frac{\partial E(W,b)}{\partial b}$$

※ α 는 학습율(learning rate) 이라고 부르며, W 값의 감소 또는 증가 되는 비율을 나타냄

gradient decent algorithm- 최적의 [W,b]계산 프로세스



Linear regression implementation using python

순서	데이터 및 수식	Python 구현
1	입력 (x) | 정답 (t) x_1 | t_1 x_2 | t_2 … | … … | … x_n | t_n	슬라이싱(slicing) 또는 list comprehension 등을 이용하여 입력 x 와 정답 t 를 numpy 데이터형으로 분리
2	$y = Wx + b$	`W = numpy.random.rand(…)` `b = numpy.random.rand(…)`

3	regression 손실함수
	$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i-(Wx_i+b)]^2$
		`def loss_func(…):`
		X, W, t, y 모두 numpy 행렬
		`y = numpy.dot(X, W) + b`
		`return ( numpy.sum( (t-y)**2) ) / ( len(x) )`

4	학습률 α	`learning_rate = 1e-3 or 1e-4 or 1e-5 …`

5	가중치 W, 바이어스 b
	$W = W - \alpha\frac{\partial E(W,b)}{\partial W}$
	$b = b - \alpha\frac{\partial E(W,b)}{\partial b}$
		`f = lambda x : loss_func(…)`
		`for step in range(6000): # 6000 은 임의값`
		`W -= learning_rate * numerical_derivative(f, W)`
		`b -= learning_rate * numerical_derivative(f, b)`
		numerical_derivative 는 머신러닝 강의 10] 수치미분 참조

Simple regression-concept

training data

입력 (x)	정답 (t)
1	2
2	3
3	4
4	5
5	6

training data를 보면,
y = x + 1 임을 알 수 있음.
머신러닝 regression으로
W = 1, b = 1 얻을수 있는지 확인

$$x_i \xrightarrow{W} \boxed{x_i W + b} \longrightarrow y_i$$

$x_1 W + b = y_1$

$x_2 W + b = y_2$

$x_3 W + b = y_3$

$x_4 W + b = y_4$

$x_5 W + b = y_5$

행렬변환 →

$$X \cdot W + b = Y$$

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} \cdot (W) + b = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \end{pmatrix}$$

(5 X 1) • (1 X 1) = (5 X 1)

- 오차를 계산하기 위해서는 training data의 모든 입력 x 에 대해 각각의 y = Wx+b 계산 해야 함 ⇒ 이때 입력 x, 정답 t, 가중치 W 모두를 행렬로 나타낸 후에, 행렬 곱(dot product)을 이용하면 계산 값 y 또한 행렬로 표시되어 모든 입력 데이터에 대해 한번에 쉽게 계산되는 것을 알 수 있음

Simple regression-concept

[1] 학습데이터(Training Data) 준비

입력 (x)	정답 (t)
1	2
2	3
3	4
4	5
5	6

```
import numpy as np

x_data = np.array([1, 2, 3, 4, 5]).reshape(5,1)
t_data = np.array([2, 3, 4, 5, 6]).reshape(5,1)

# raw_data = [ [1, 2], [2, 3], [3, 4], [4, 5], [5, 6] ]
```

[2] 임의의 직선 y = Wx + b 정의 (임의의 값으로 가중치 W, 바이어스 b 초기화)

$$y = Wx + b$$

```
W = np.random.rand(1,1)
b = np.random.rand(1)
print("W = ", W, ", W.shape = ", W.shape, ", b = ", b, ", b.shape = ", b.shape)
```

```
W = [[0.3468086]] , W.shape = (1, 1) , b = [0.11730069] , b.shape = (1,)
```

[3] 손실함수 E(W,b) 정의

$$E(W,b) = \frac{1}{n} \sum_{i=1}^{n} [t_i - (Wx_i + b)]^2$$

```
def loss_func(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )
```

Simple regression-concept

[4] 수치미분 numerical_derivative 및 utility 함수 정의

```
def numerical_derivative(f, x):
    delta_x = 1e-4 # 0.0001
    grad = np.zeros_like(x)

    it = np.nditer(x, flags=['multi_index'], op_flags=['readwrite'])

    while not it.finished:
        idx = it.multi_index
        tmp_val = x[idx]
        x[idx] = float(tmp_val) + delta_x
        fx1 = f(x) # f(x+delta_x)

        x[idx] = tmp_val - delta_x
        fx2 = f(x) # f(x-delta_x)
        grad[idx] = (fx1 - fx2) / (2*delta_x)

        x[idx] = tmp_val
        it.iternext()

    return grad
```

```
# 손실함수 값 계산 함수
# 입력변수 x, t : numpy type
def error_val(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )

# 학습을 마친 후, 임의의 데이터에 대해 미래 값 예측 함수
# 입력변수 x : numpy type
def predict(x):
    y = np.dot(x,W) + b

    return y
```

[5] 학습율 (learning rate) 초기화 및 손실함수가 최소가 될 때까지 W, b 업데이트

$$W = W - \alpha \frac{\partial E(W,b)}{\partial W}$$

$$b = b - \alpha \frac{\partial E(W,b)}{\partial b}$$

```
learning_rate = 1e-2  # 발산하는 경우, 1e-3 ~ 1e-6 등으로 바꾸어서 실행

f = lambda x : loss_func(x_data,t_data)      # f(x) = loss_func(x_data, t_data)

print("Initial error value = ", error_val(x_data, t_data), "Initial W = ", W, "\n", ", b = ", b )

for step in  range(8001):

    W -= learning_rate * numerical_derivative(f, W)

    b -= learning_rate * numerical_derivative(f, b)

    if (step % 400 == 0):
        print("step = ", step, "error value = ", error_val(x_data, t_data), "W = ", W, ", b = ",b )
```

Simple regression-concept

[6] 학습결과 및 입력 43에 대한 미래 값 예측

```
Initial error value =  8.93183665205504 Initial W =  [[0.3468086]]
 , b =  [0.11730069]
step =  0 error value =  5.288723739645144 W =  [[0.54347267]] , b =  [0.16234632]
step =  400 error value =  0.005646375874345945 W =  [[1.04879571]] , b =  [0.82387513]
step =  800 error value =  0.0003602721620039822 W =  [[1.01232571]] , b =  [0.95551115]
step =  1200 error value =  2.298749385508783e-05 W =  [[1.00311345]] , b =  [0.98876219]
step =  1600 error value =  1.466738009393298e-06 W =  [[1.00078645]] , b =  [0.99716135]
step =  2000 error value =  9.358655631456343e-08 W =  [[1.00019866]] , b =  [0.99928296]
step =  2400 error value =  5.971375574060026e-09 W =  [[1.00005018]] , b =  [0.99981888]
step =  2800 error value =  3.810090642396003e-10 W =  [[1.00001268]] , b =  [0.99995425]
step =  3200 error value =  2.431063080936452e-11 W =  [[1.0000032]] , b =  [0.99998844]
step =  3600 error value =  1.551161970081539e-12 W =  [[1.00000081]] , b =  [0.99999708]
step =  4000 error value =  9.897330421093632e-14 W =  [[1.0000002]] , b =  [0.99999926]
step =  4400 error value =  6.315081987274994e-15 W =  [[1.00000005]] , b =  [0.99999981]
step =  4800 error value =  4.029395698053509e-16 W =  [[1.00000001]] , b =  [0.99999995]
step =  5200 error value =  2.570992767160701e-17 W =  [[1.]] , b =  [0.99999999]
step =  5600 error value =  1.640445289574914e-18 W =  [[1.]] , b =  [1.]
step =  6000 error value =  1.046700699526440e-19 W =  [[1.]] , b =  [1.]
step =  6400 error value =  6.678591076979926e-21 W =  [[1.]] , b =  [1.]
step =  6800 error value =  4.261382945763941e-22 W =  [[1.]] , b =  [1.]
step =  7200 error value =  2.719574443959016e-23 W =  [[1.]] , b =  [1.]
step =  7600 error value =  1.736704892975740e-24 W =  [[1.]] , b =  [1.]
step =  8000 error value =  1.112104549744373e-25 W =  [[1.]] , b =  [1.]
```

손실함수 값,
W, b 확인

```
predict(43)
```

43 의 미래값 예측

```
array([[44.]])
```

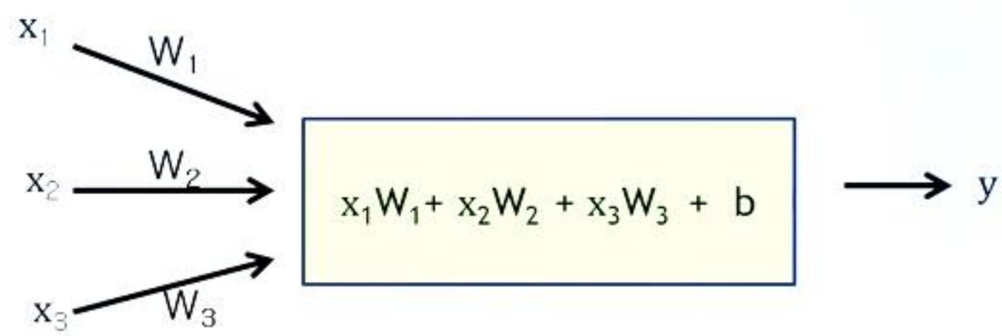
Multi-variable regression -concept

training data

25행

x_1	x_2	x_3	t
73	80	75	152
93	88	93	185
89	91	90	180
96	98	100	196
73	66	70	142
53	46	55	101
69	74	77	149
47	56	60	115
87	79	90	175
79	70	88	164
69	70	73	141
70	65	74	141
93	95	91	184
79	80	73	152
70	73	78	148
93	89	96	192
78	75	68	147
81	90	93	183
88	92	86	177
78	83	77	159
82	86	90	177
86	82	89	175
78	83	85	175
76	83	71	149
96	93	95	192

(25 X 4)



$$X \cdot W + b = Y$$

$$\begin{pmatrix} 73 & 80 & 75 \\ 93 & 88 & 93 \\ & \cdots & \\ & \cdots & \\ 76 & 83 & 71 \\ 96 & 93 & 95 \end{pmatrix} \cdot \begin{pmatrix} W_1 \\ W_2 \\ W_3 \end{pmatrix} + b = \begin{pmatrix} 152 \\ 185 \\ \cdots \\ \cdots \\ 149 \\ 192 \end{pmatrix}$$

(25 X 3) • (3 X 1) = (25 X 1)

Multi-variable regression -example

[1] 학습데이터(Training Data) 준비

x_1	x_2	x_3	t
73	80	75	152
93	88	93	185
...	...	...	...
...	...	...	...
76	83	71	149
96	93	95	192

```python
import numpy as np

loaded_data = np.loadtxt('./data-01-test-score.csv', delimiter=',', dtype=np.float32)

x_data = loaded_data[ :, 0:-1]
t_data = loaded_data[ :, [-1]]
```

[2] 임의의 직선 $y = W_1x_1 + W_2x_2 + W_3x_3 + b$ 정의

$$y = W_1x_1 + W_2x_2 + W_3x_3 + b$$

```python
W = np.random.rand(3,1)  # 3X1 행렬
b = np.random.rand(1)
print("W = ", W, ", W.shape = ", W.shape, ", b = ", b, ", b.shape = ", b.shape)
```

```
W =  [[0.04946736]
 [0.00916638]
 [0.56439521]] , W.shape =  (3, 1) , b =  [0.34662569] , b.shape =  (1,)
```

[3] 손실함수 E(W,b) 정의

$$E(W,b) = \frac{1}{n}\sum_{i=1}^{n}[t_i - (Wx_i + b)]^2$$

```python
def loss_func(x, t):
    y = np.dot(x,W) + b

    return ( np.sum( (t - y)**2 ) ) / ( len(x) )
```

Multi-variable regression -example

[6] 학습결과

```
Initial error value =  12612.552489005942 Initial W =  [[0.04946736]
 [0.00916638]
 [0.56439521]]
 , b =  [0.34662569]
step =  0 error value =  4667.485325027743 W =  [[0.22884158]
 [0.18957711]
 [0.74902839]] , b =  [0.34797584]
step =  400 error value =  6.516038270751993 W =  [[0.49272251]
 [0.47382228]
 [1.04522003]] , b =  [0.34945336]
step =  800 error value =  6.450168440656085 W =  [[0.47964981]
 [0.47799457]
 [1.05385525]] , b =  [0.34882266]
step =  1200 error value =  6.396649123184944 W =  [[0.46783145]
 [0.48195151]
 [1.061482  ]] , b =  [0.34818207]
step =  1600 error value =  6.353111065623369 W =  [[0.45714635]
 [0.48568404]
 [1.06822679]] , b =  [0.34753274]
step =  2000 error value =  6.317653653969927 W =  [[0.44748521]
 [0.48918877]
 [1.07419911]] , b =  [0.34687568]
step =  2400 error value =  6.288748220628605 W =  [[0.43874941]
 [0.49246669]
 [1.07949388]] , b =  [0.34621178]
step =  2800 error value =  6.265162397701323 W =  [[0.43084988]
 [0.49552202]
 [1.08419348]] , b =  [0.34554182]
step =  3200 error value =  6.245900647226594 W =  [[0.42370621]
 [0.49836145]
 [1.08836955]] , b =  [0.34486649]
step =  3600 error value =  6.230157297812982 W =  [[0.41724578]
 [0.50099336]
 [1.09208444]] , b =  [0.34418639]
```

Multi-variable regression -example

[7] 학습 결과 및 입력[100,98,81] 미래 값 예측

```
step =  5200 error value =  6.191014364108381 W =  [[0.39701586]
 [0.50964661]
 [1.10332228]] , b =  [0.34142807]
step =  5600 error value =  6.185199078254 65 W =  [[0.39310548]
 [0.51139482]
 [1.10542264]] , b =  [0.34073099]
step =  6000 error value =  6.180420062497554 W =  [[0.38956834]
 [0.5129984 ]
 [1.10730164]] , b =  [0.34003155]
step =  6400 error value =  6.176487722933351 W =  [[0.38636872]
 [0.5144676 ]
 [1.10898395]] , b =  [0.33933002]
step =  6800 error value =  6.173247368989568 W =  [[0.38347434]
 [0.51581229]
 [1.11049132]] , b =  [0.33862661]
step =  7200 error value =  6.170572724377186 W =  [[0.38085603]
 [0.51704184]
 [1.11184291]] , b =  [0.33792154]
step =  7600 error value =  6.168360663162613 W =  [[0.37848739]
 [0.51816518]
 [1.11305567]] , b =  [0.33721499]
step =  8000 error value =  6.166526934201 77 W =  [[0.37634459]
 [0.51919067]
 [1.11414458]] , b =  [0.33650712]
step =  8400 error value =  6.165002684499958 W =  [[0.37440604]
 [0.52012619]
 [1.11512291]] , b =  [0.33579807]
step =  8800 error value =  6.163731629484813 W =  [[0.37265225]
 [0.52097911]
 [1.11600242]] , b =  [0.33508799]
step =  9200 error value =  6.162667747895536 W =  [[0.3710656 ]
 [0.52175625]
 [1.11679356]] , b =  [0.33437698]
step =  9600 error value =  6.161773402669903 W =  [[0.36963011]
 [0.522464  ]
 [1.1175056 ]] , b =  [0.33366515]
step =  10000 error value =  6.161017808148 6735 W =  [[0.36833139]
 [0.52310825]
 [1.11814679]] , b =  [0.33295259]
```

```
test_data = np.array([100, 98, 81])

predict(test_data)
```

⬅ [100, 98, 81] 의 미래값 예측

```
array([179.00059026])
```