

9주차(1/3)

아달라인 경사하강법 구현

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

아달라인 경사하강법 구현

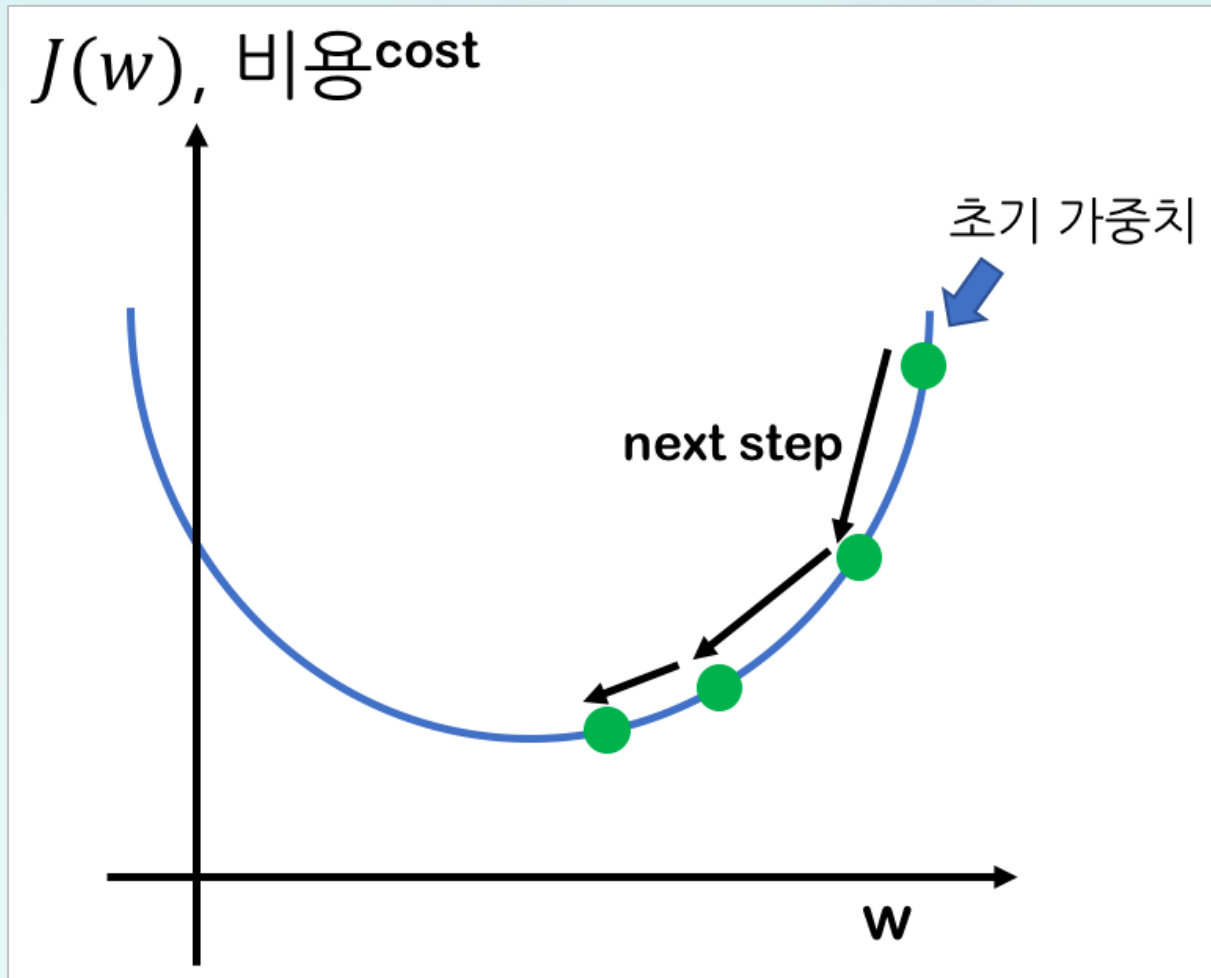
- 학습 목표
 - 아달라인 알고리즘이 무엇인지 학습한다.
 - 비용함수를 통해 오차가 최소화되는 방법을 학습한다.
 - 경사하강법을 통해 최저점을 찾는 방법을 학습한다.
- 학습 내용
 - 아달라인 알고리즘
 - 비용 함수
 - 경사하강법

1. 경사하강법: 비용 함수

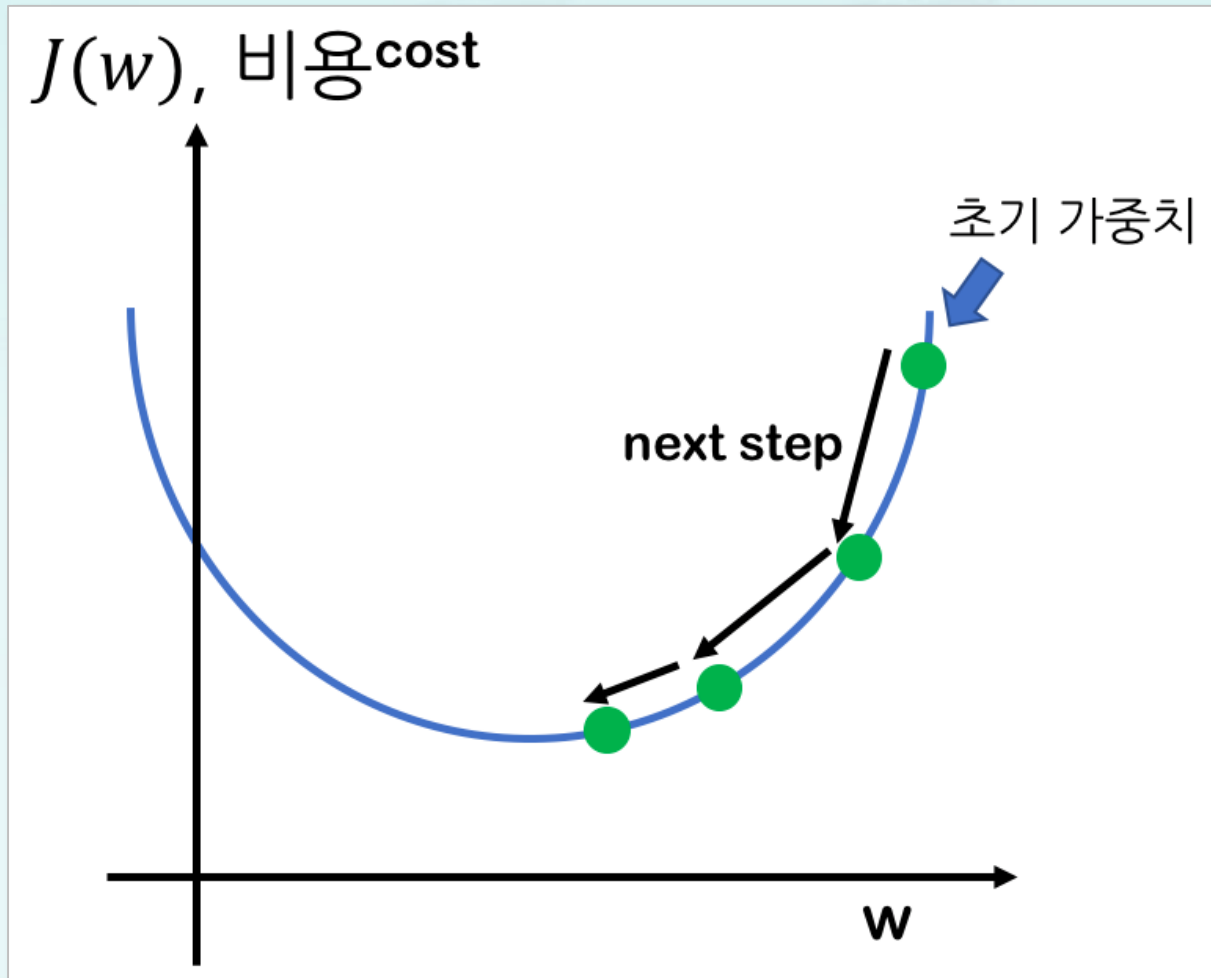
- 비용 함수
 - 최소 제곱법을 이용한 비용 함수 $J(w)$

$$J(w) = \frac{1}{2} \sum_{i=1}^m \left(y^i - \sum_{j=1}^n (w_j x_j^i) \right)^2$$

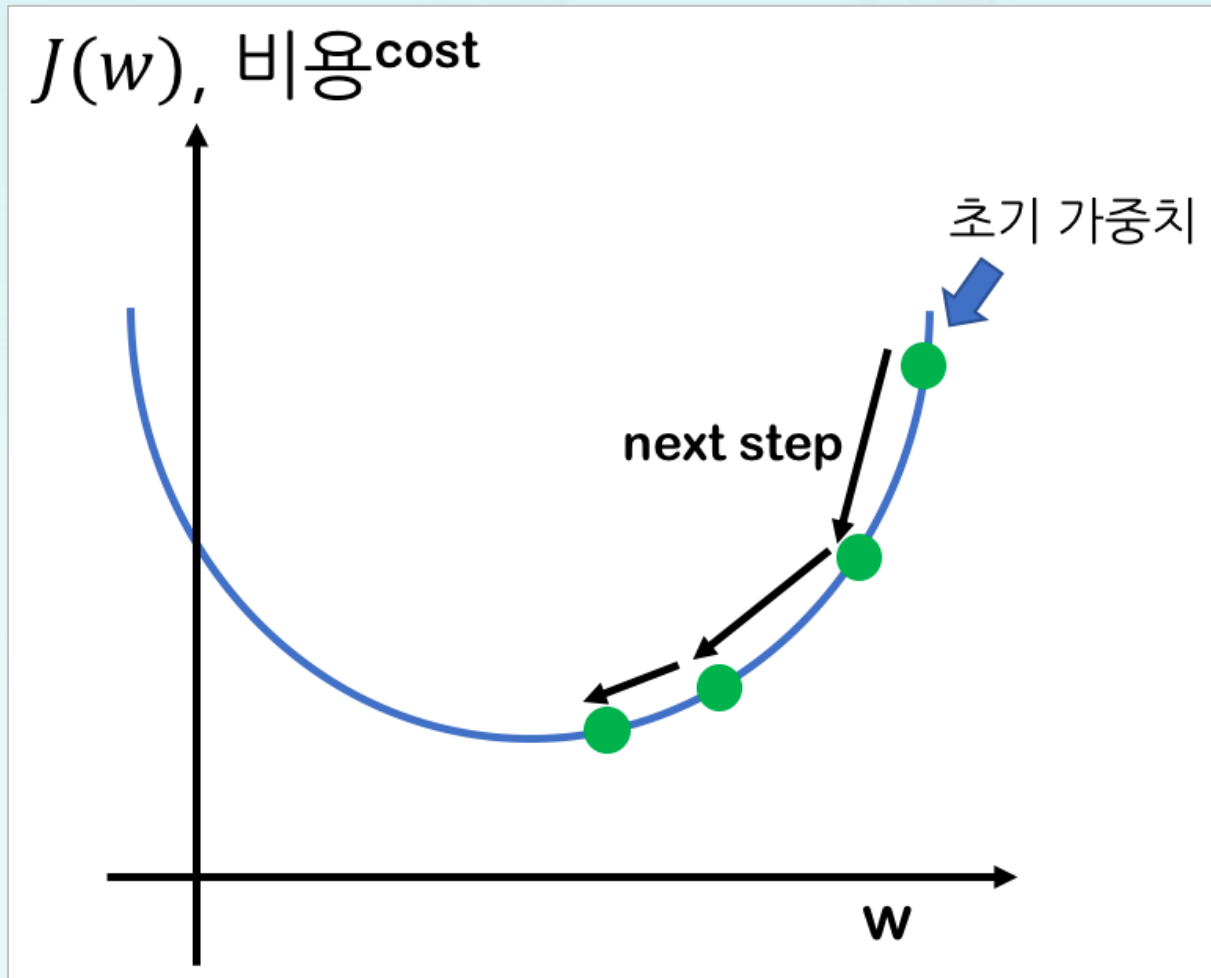
1. 경사하강법: 비용 함수



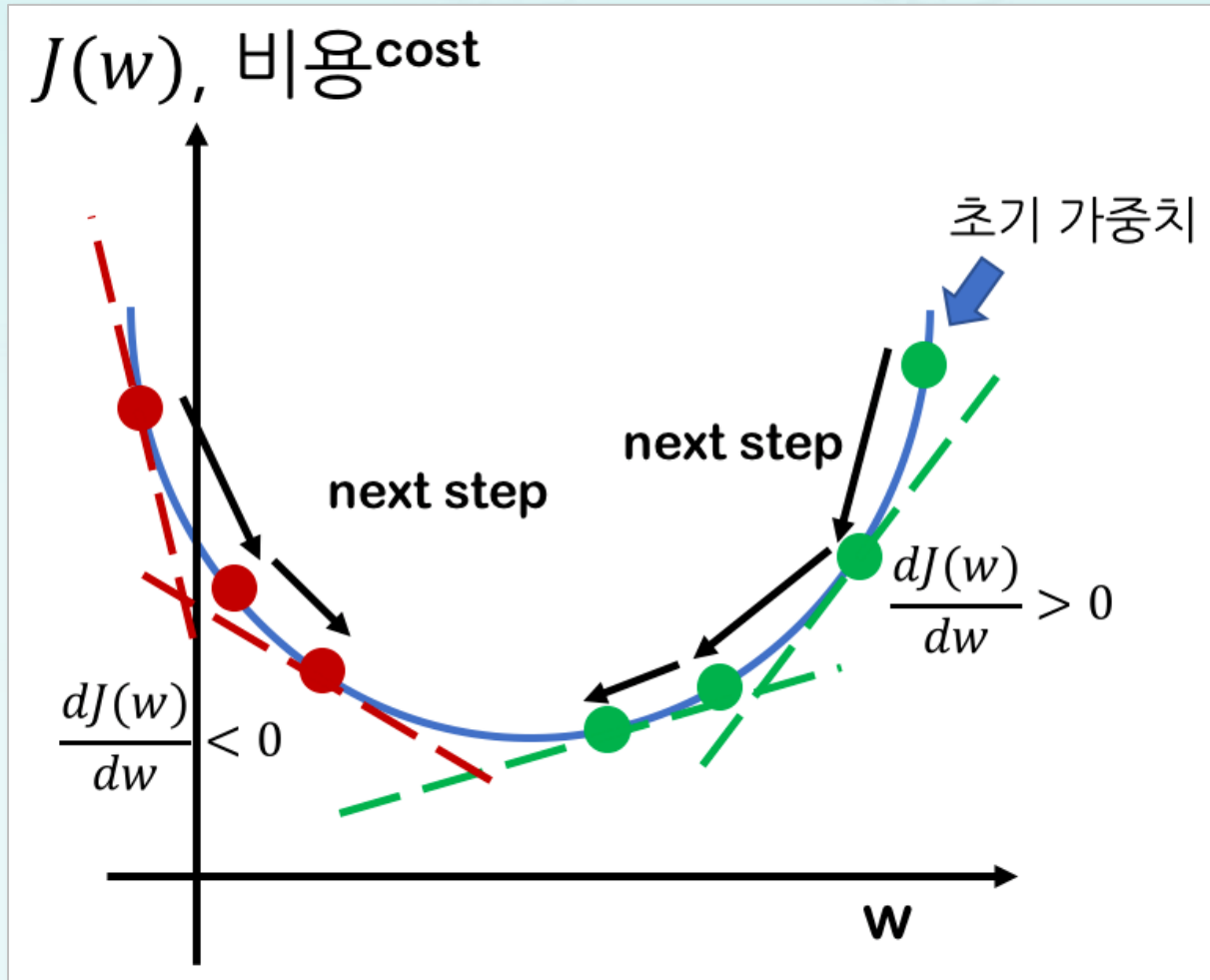
1. 경사하강법: 비용 함수



1. 경사하강법: 비용 함수

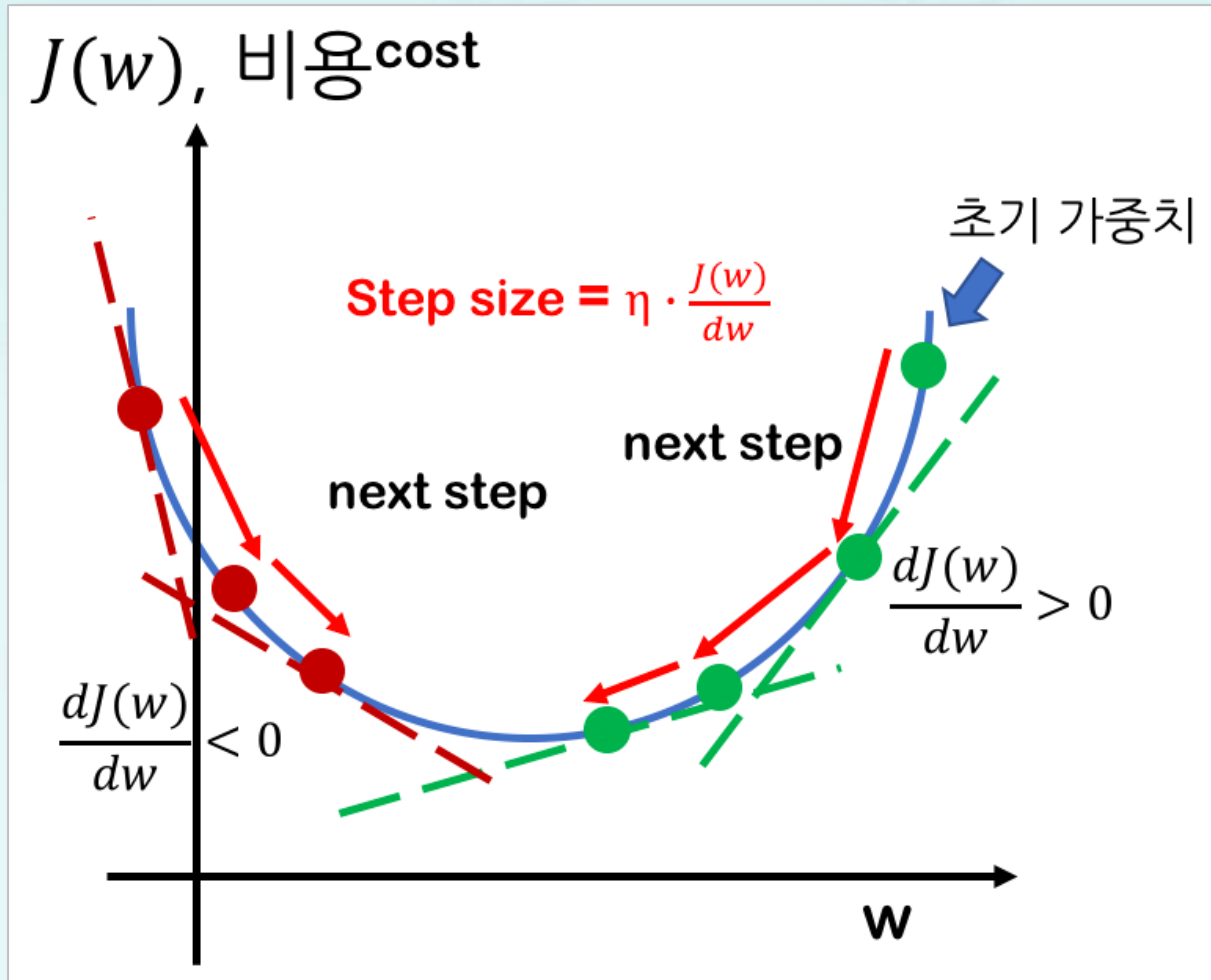


1. 경사하강법: 스텝의 방향과 크기



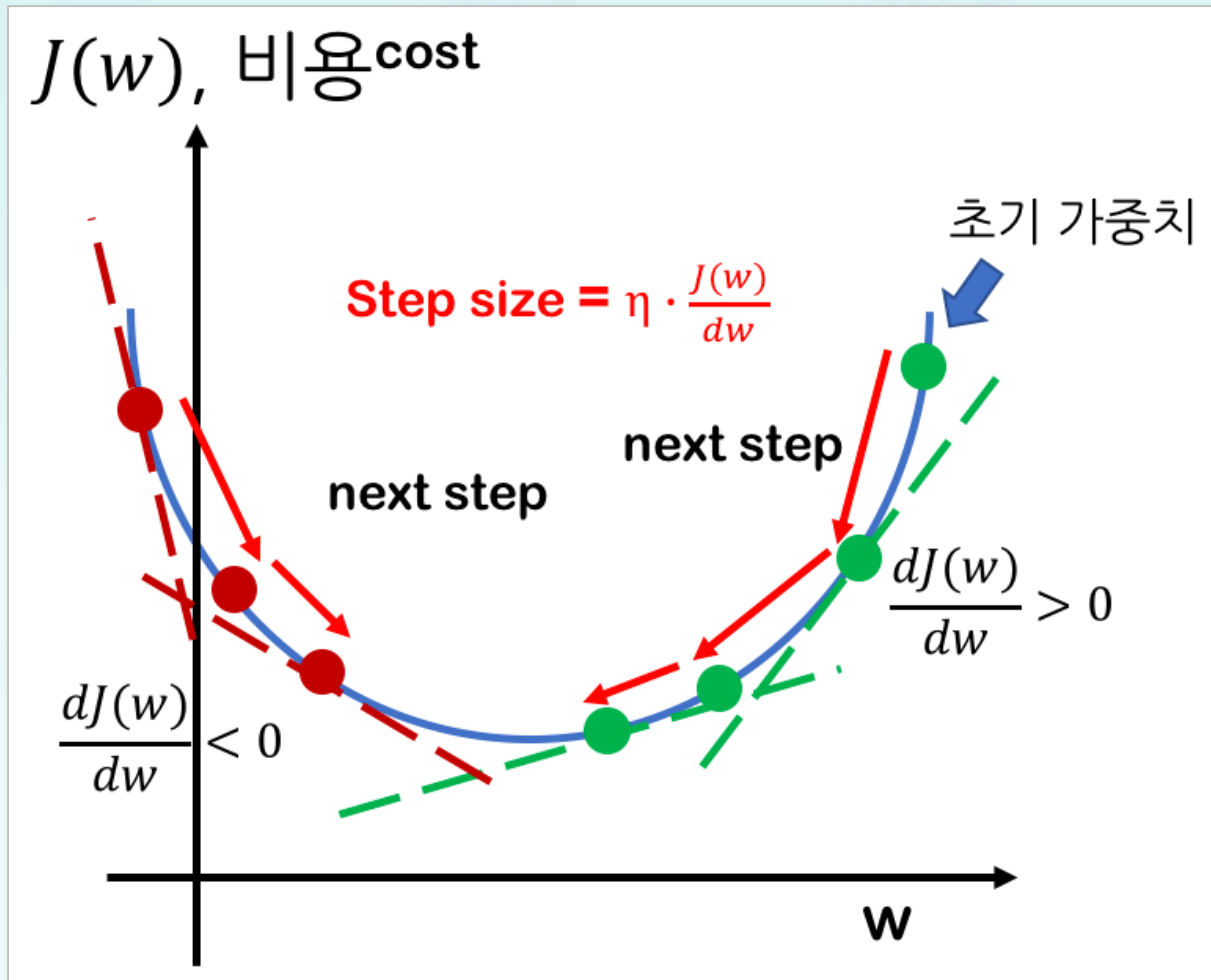
- Step 방향 : $-\frac{dJ(w)}{dw}$

1. 경사하강법: 스텝의 방향과 크기



- Step 방향 : $-\frac{dJ(w)}{dw}$
- Step 크기 : $-\eta \cdot \frac{dJ(w)}{dw} = \Delta w$

1. 경사하강법: 스텝의 방향과 크기

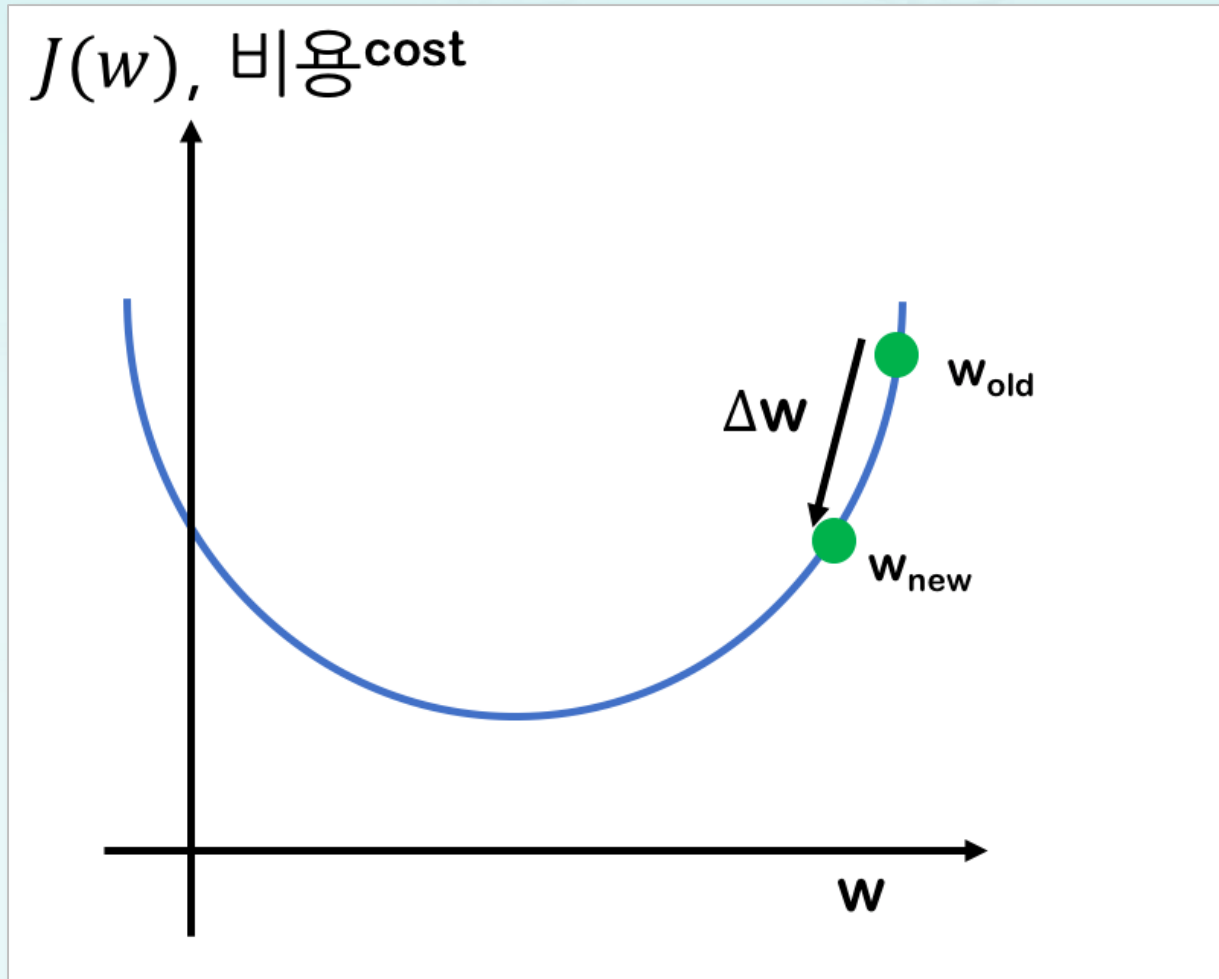


- Step 방향 : $-\frac{dJ(w)}{dw}$
- Step 크기 : $-\eta \cdot \frac{dJ(w)}{dw} = \Delta w$

1. 경사하강법: 가중치 조정

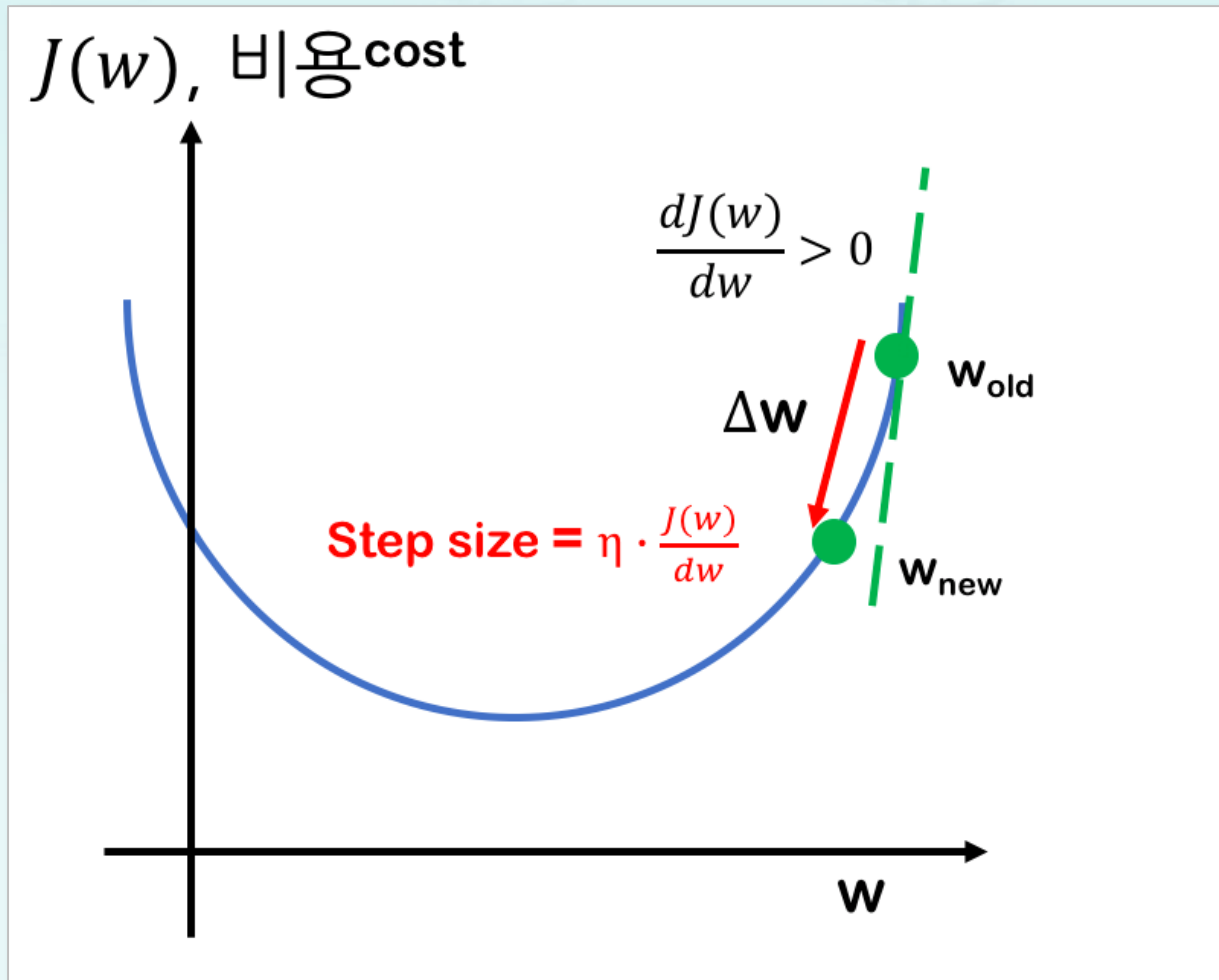
- 스텝의 크기 Δw

$$w_{new} = w_{old} + \Delta w$$



1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw



$$\begin{aligned} w_{new} &= w_{old} + \Delta w \\ &= w_j + \eta \frac{\partial J(w)}{\partial w_j} \end{aligned}$$

1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw

$$\frac{\partial J(w)}{\partial w_j} = \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - h(z^{(i)}) \right)^2$$

합성함수 미분법

$$f(g(x))' = f'(g(x))g'(x)$$

1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw

$$\begin{aligned}\frac{\partial J(w)}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - h(z^{(i)}) \right)^2 \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - h(z^{(i)}) \right)\end{aligned}$$

합성함수 미분법

$$f(g(x))' = f'(g(x))g'(x)$$

$h(\cdot)$ = identity function

$$h(z) = z, \quad z = \sum w x$$

1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw

$$\begin{aligned}\frac{\partial J(w)}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - h(z^{(i)}) \right)^2 \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - h(z^{(i)}) \right) \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_i w_j x_j^{(i)} \right)\end{aligned}$$

합성함수 미분법

$$f(g(x))' = f'(g(x))g'(x)$$

$h(\cdot) = \text{identity function}$

$$h(z) = z, \quad z = \sum w x$$

1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw

$$\begin{aligned}\frac{\partial J(w)}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - h(z^{(i)}) \right)^2 \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - h(z^{(i)}) \right) \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_i w_j x_j^{(i)} \right) \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \left(- \sum_i x_j^{(i)} \right)\end{aligned}$$

합성함수 미분법

$$f(g(x))' = f'(g(x))g'(x)$$

$h(\cdot) = \text{identity function}$

$$h(z) = z, z = \sum w x$$

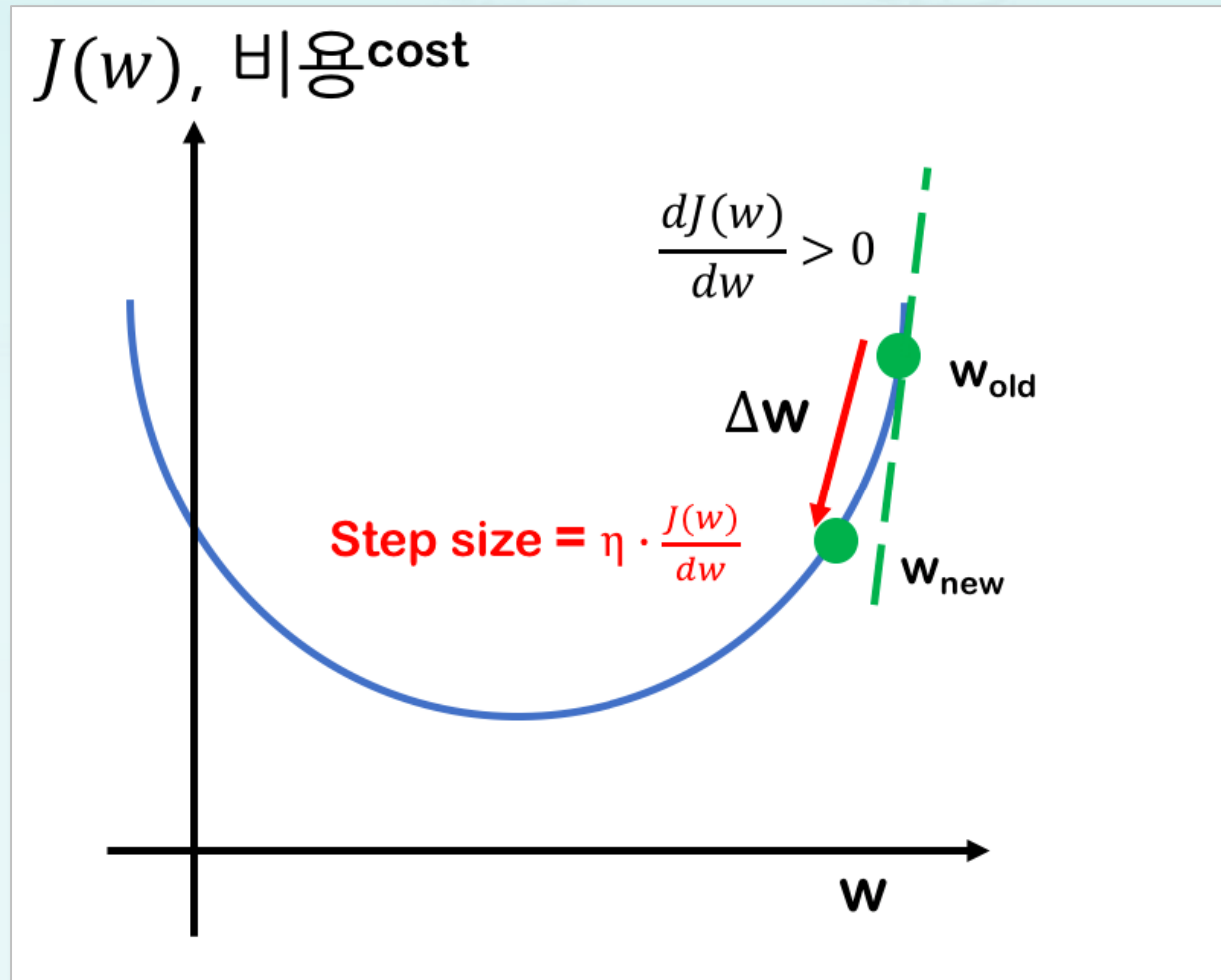
$y^{(i)}, x_j^{(i)}$ 상수 취급

1. 경사하강법: 가중치 조정

- 스텝의 크기 Δw

$$\begin{aligned}\frac{\partial J(w)}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i \left(y^{(i)} - h(z^{(i)}) \right)^2 \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - h(z^{(i)}) \right) \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \frac{\partial}{\partial w_j} \left(y^{(i)} - \sum_i w_j x_j^{(i)} \right) \\ &= \frac{1}{2} \sum_i 2 \left(y^{(i)} - h(z^{(i)}) \right) \left(- \sum_i x_j^{(i)} \right) \\ &= - \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)}\end{aligned}$$

1. 경사하강법: 가중치 조정

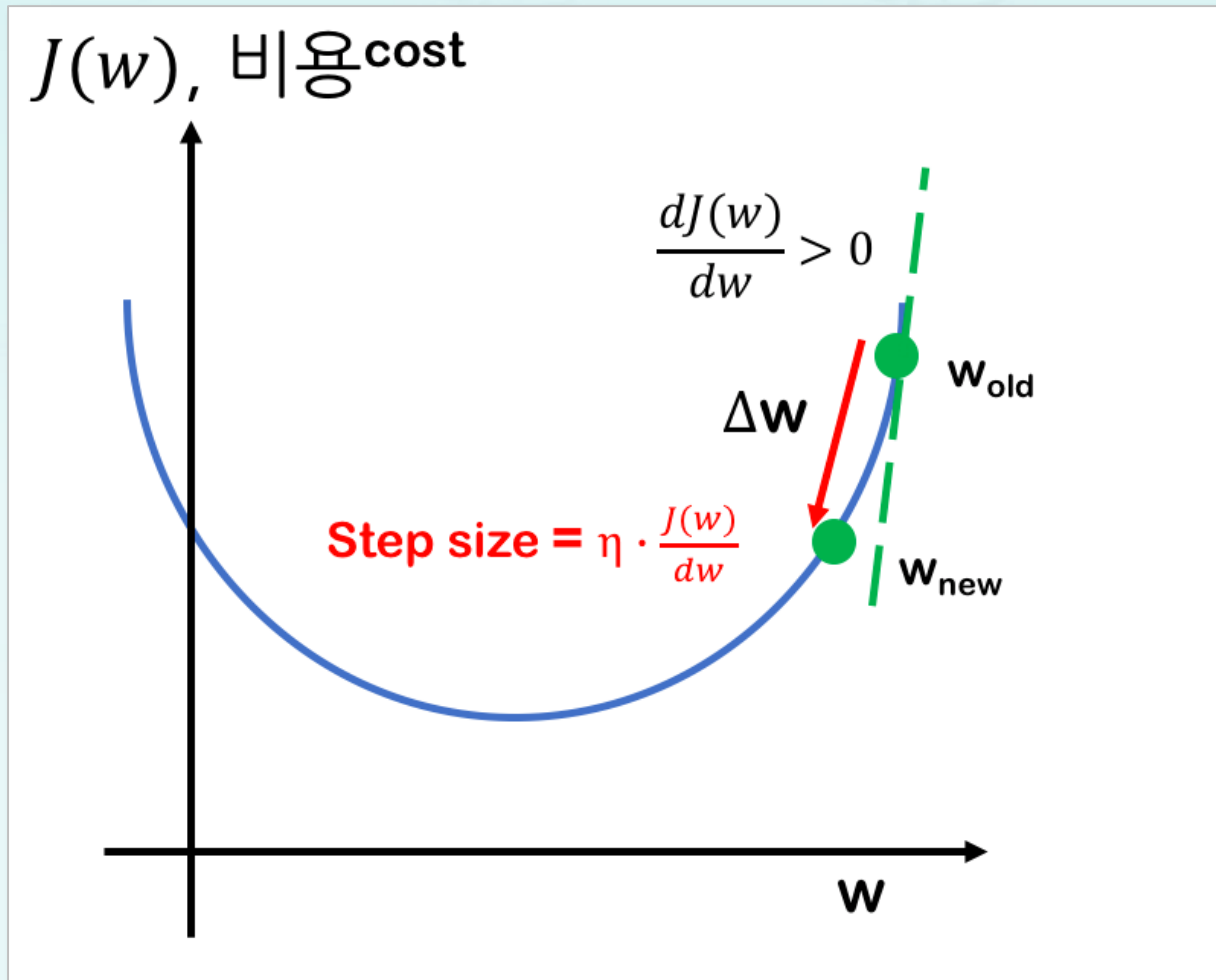


- 스텝의 크기 Δw

$$\begin{aligned}w_{new} &= w_{old} + \Delta w \\&= w_{old} + \eta \frac{\partial J(w)}{\partial w_j} \\&= w_{old} + \eta \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)} \\&= w_{old} + \eta \sum_i \left(y^{(i)} - \hat{y}^{(i)} \right) x_j^{(i)}\end{aligned}$$

1. 경사하강법: 가중치 조정

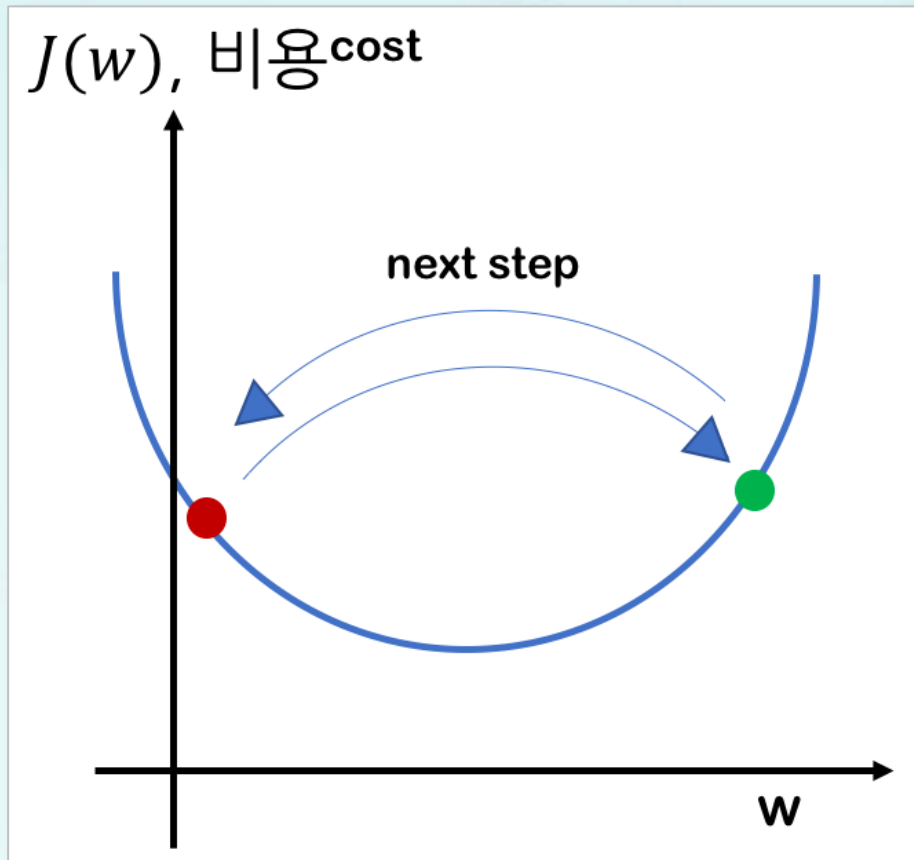
- 스텝의 크기 Δw



$$\begin{aligned} w_{new} &= w_{old} + \Delta w \\ &= w_{old} + \eta \frac{\partial J(w)}{\partial w_j} \\ &= w_{old} + \eta \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)} \end{aligned}$$

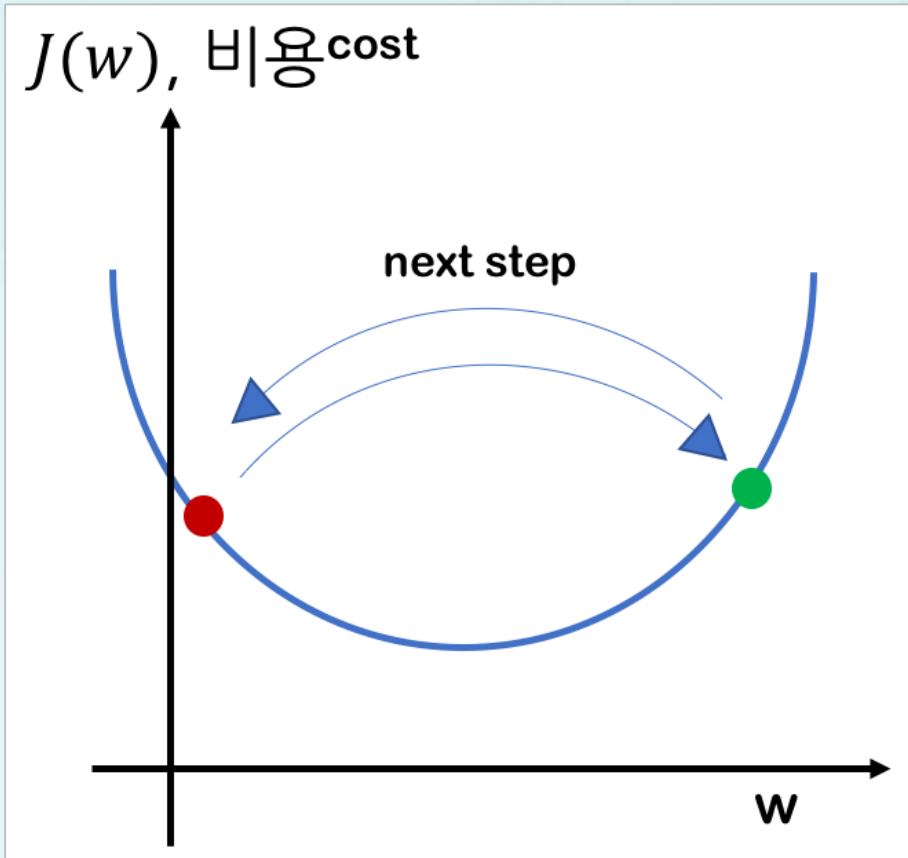
1. 경사하강법: 학습률

- 학습률 ($\eta : \uparrow$) 너무 클 경우
- 학습률 ($\eta : \downarrow$)

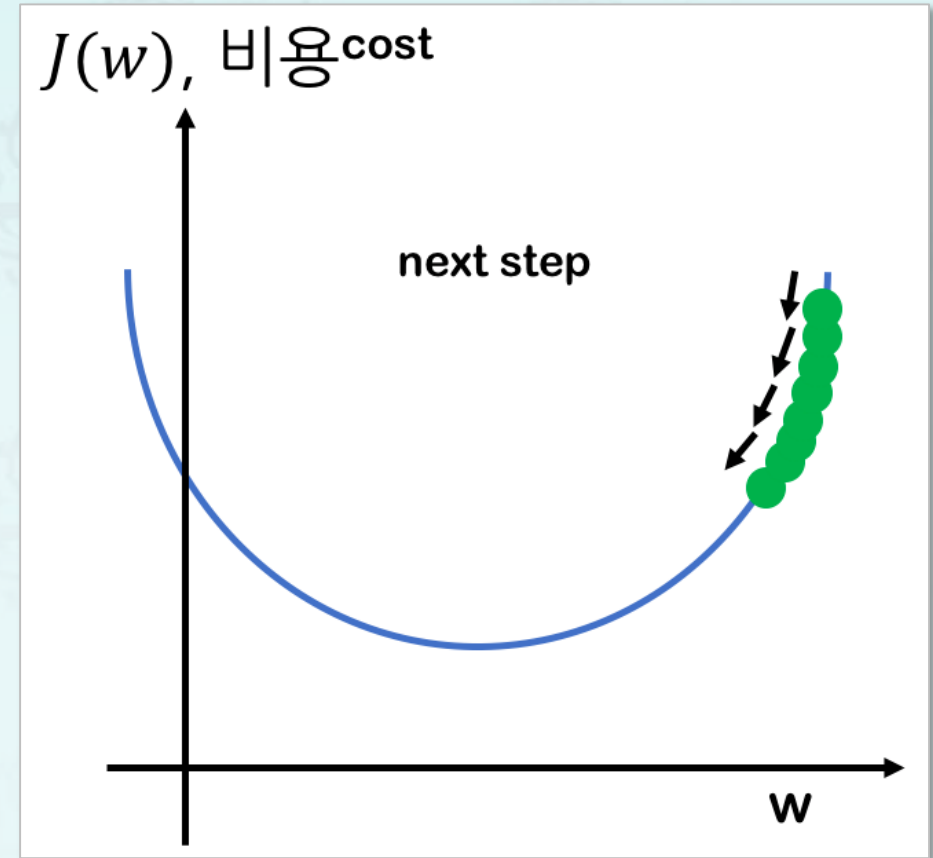


1. 경사하강법: 학습률

- 학습률 ($\eta : \uparrow$) 너무 클 경우

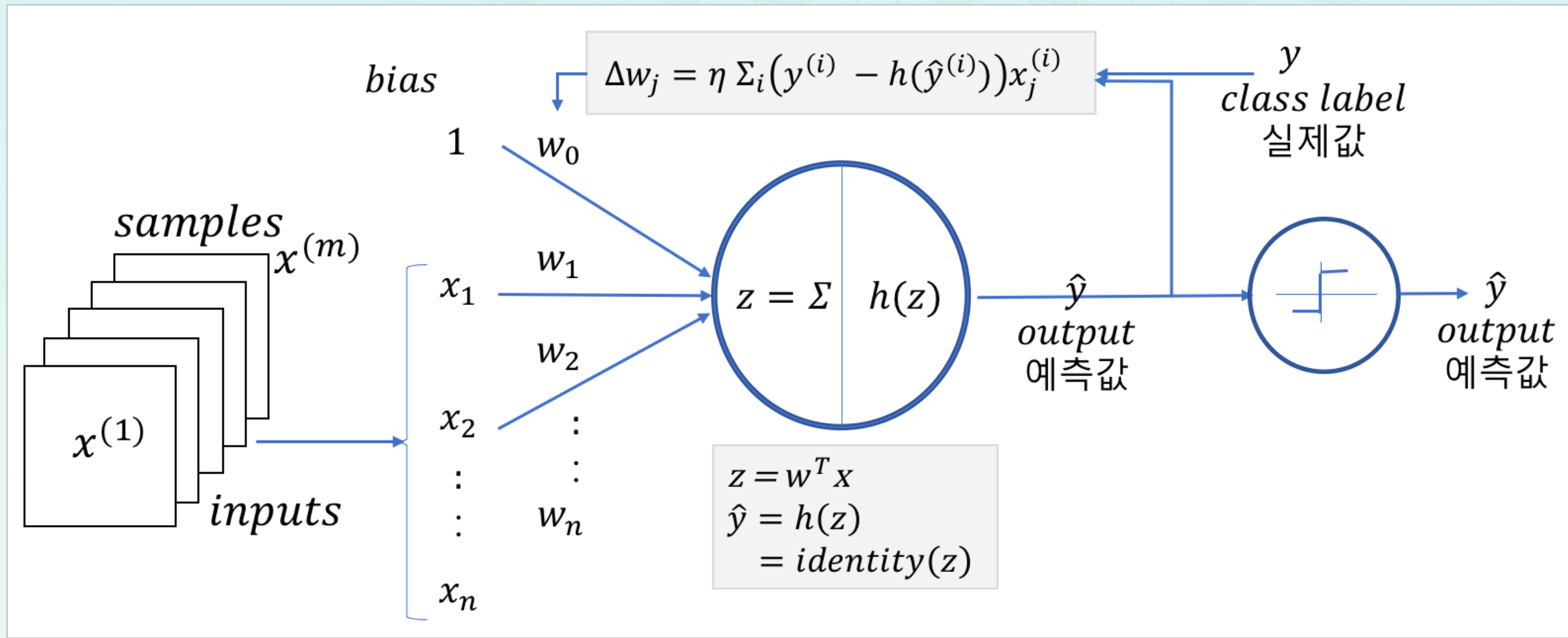


- 학습률 ($\eta : \downarrow$) 너무 작은 경우

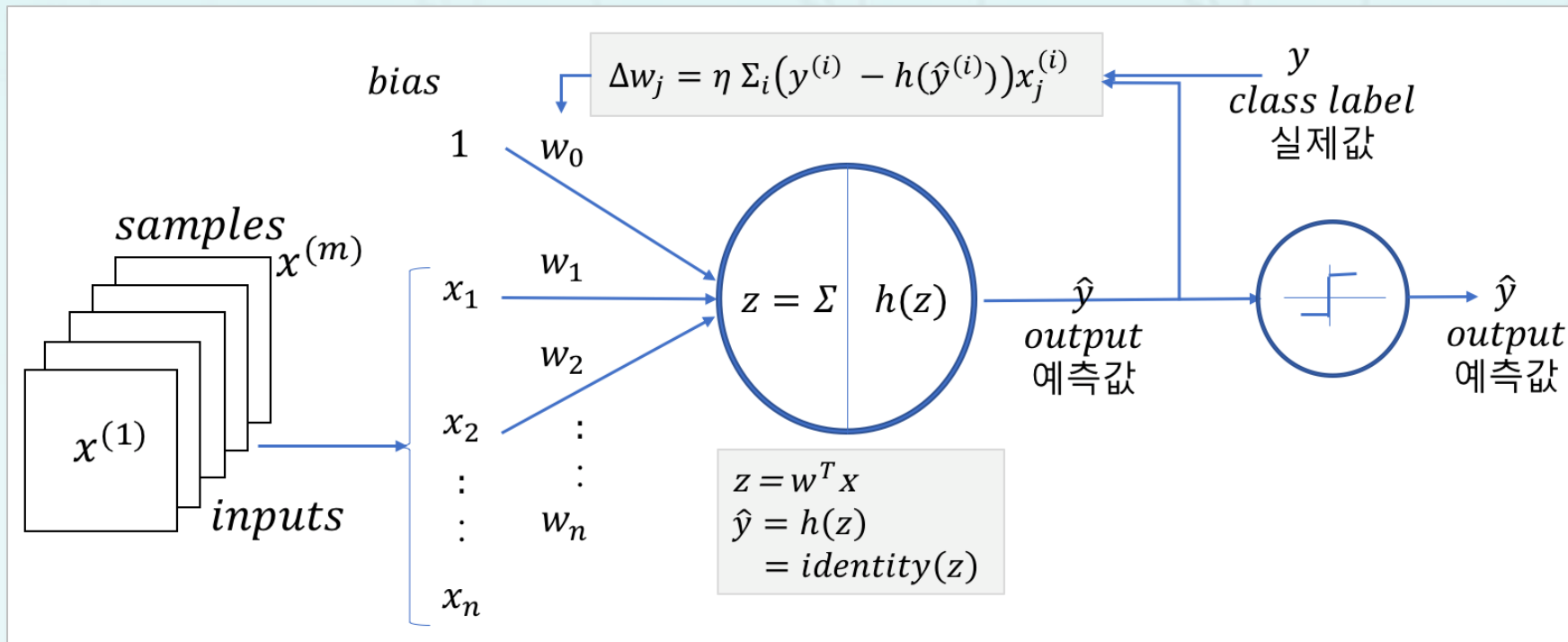


2. 아달라인 경사하강법 구현: 객체지향 아달라인

2. 아달라인 경사하강법 구현: 객체지향 아달라인



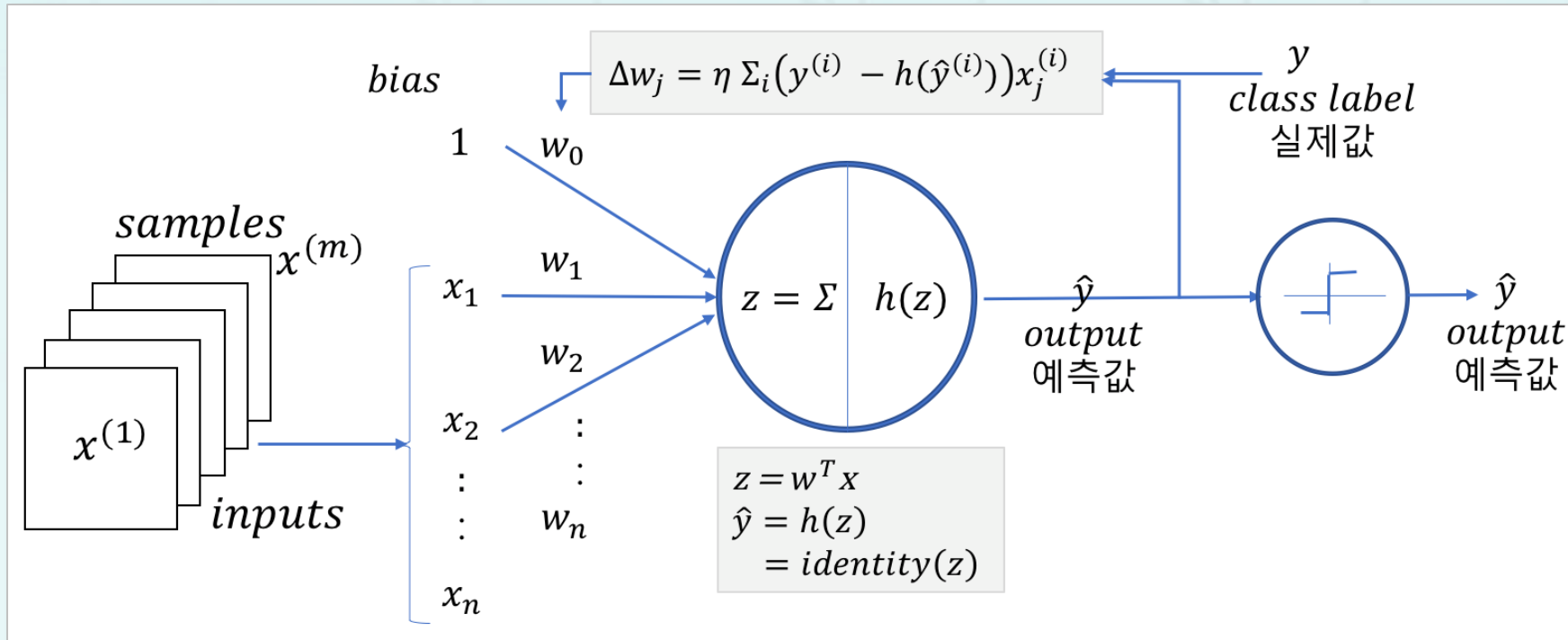
2. 아달라인 경사하강법 구현: 객체지향 아달라인



[속성:데이터]

1. 입력 (x)
2. 출력 (y)
3. 순입력 (z)
4. 레이블 (\hat{y})
5. 가중치 (w)
6. 학습률 (η)
7. 반복횟수 (epochs)
8. 랜덤시드
(`random_seed`)

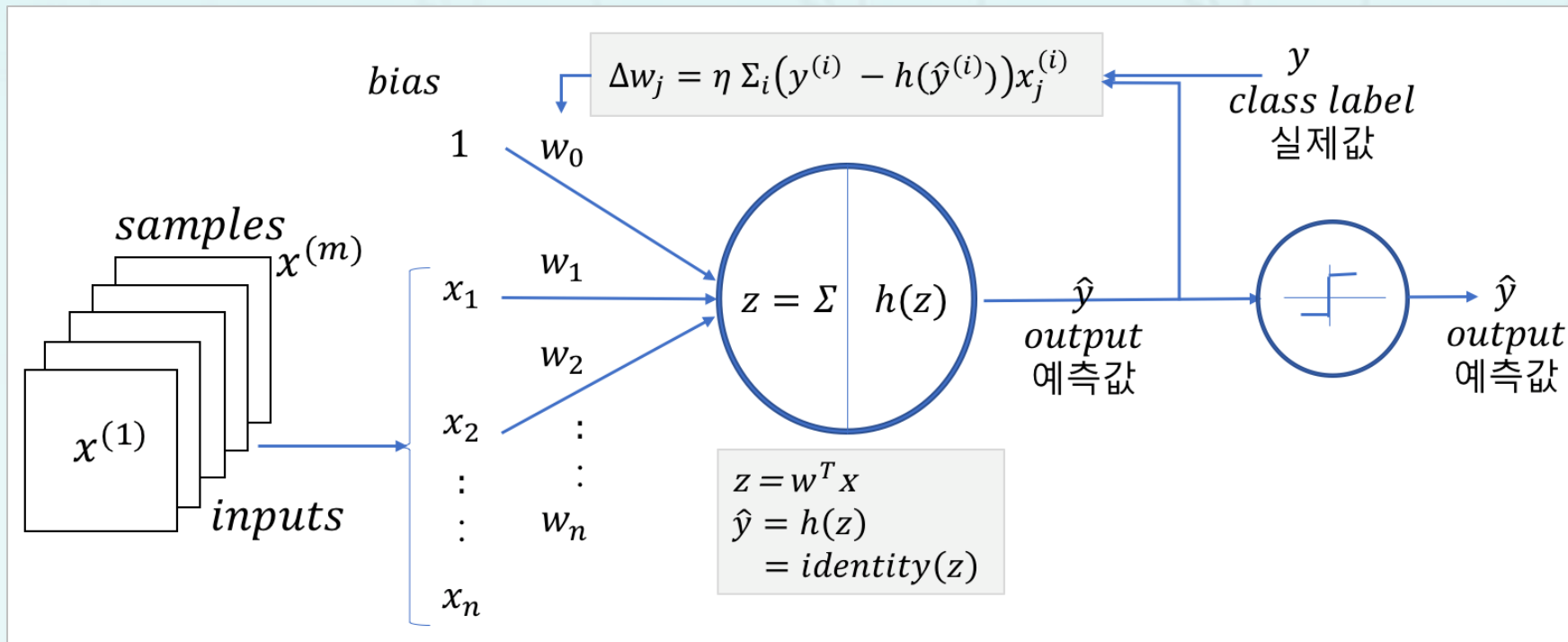
2. 아달라인 경사하강법 구현: 객체지향 아달라인



[속성:데이터]

1. 입력 (x)
2. 출력 (y)
3. 순입력 (z)
4. 레이블 (\hat{y})
5. 가중치 (w)
6. 학습률 (η)
7. 반복횟수 (epochs)
8. 랜덤시드 (random_seed)

2. 아달라인 경사하강법 구현: 객체지향 아달라인




[기능:함수]

1. 학습 (fit)
2. 순입력 (net_input)
3. 활성화 (activate)
4. 예측 (predict)

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - 이름: **Adaline Gradient Descent**



```
1 class AdalineGD():
2     """ADaptive LInear NEuron classifier.
3     """
4     def __init__(self, eta=0.01, epochs=10,
5                 random_seed=1):
6         self.eta = eta
7         self.epochs = epochs
8         self.random_seed = random_seed
9
10    def fit(self, X, y, X0=False):
11        if X0 == False:
12            X = np.c_[ np.ones(len(y)), X]
13            np.random.seed(self.random_seed)
```

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화

```
1 class AdalineGD():
2     """ADaptive LInear NEuron classifier.
3     """
4     → def __init__(self, eta=0.01, epochs=10,
5           random_seed=1):
6         self.eta = eta
7         self.epochs = epochs
8         self.random_seed = random_seed
9
10    def fit(self, X, y, X0=False):
11        if X0 == False:
12            X = np.c_[ np.ones(len(y)), X]
13            np.random.seed(self.random_seed)
```

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스


- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```


2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - **X**: 입력값
 - **y**: 클래스 레이블
 - **X0**: 편향 여부



```
def fit(self, X, y, X0=False):
    if X0 == False:
        X = np.c_[ np.ones(len(y)), X]

    np.random.seed(self.random_seed)
    self.w = np.random.random(X.shape[1])

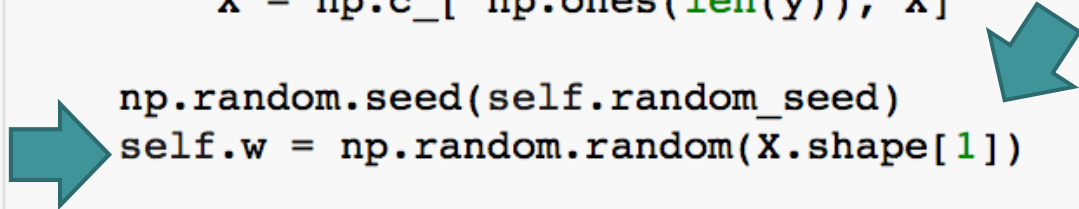
    self.maxy, self.miny = y.max(), y.min()
    self.cost_ = []
    self.w_ = np.array([self.w])
    for i in range(self.epochs):
        z = self.net_input(X)
        yhat = self.activate(z)
        errors = (y - yhat)
        self.w += self.eta * np.dot(errors, X)
        cost = 0.5*np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```


2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```




2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```




2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체 생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```



2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - `fit()` : 학습 메소드

```
z = self.net_input(X)
yhat = self.activate(z)
errors = (y - yhat)
self.w += self.eta * np.dot(errors, X)
```

$$\begin{aligned} W_{new} &= W_{old} + \Delta w \\ &= W_{old} + \eta \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)} \end{aligned}$$

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - `fit()` : 학습 메소드

```
z = self.net_input(X)
yhat = self.activate(z)
errors = (y - yhat)
self.w += self.eta * np.dot(errors, X)
```

$$\begin{aligned} W_{new} &= W_{old} + \Delta w \\ &= W_{old} + \eta \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)} \end{aligned}$$

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5 * np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```


2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - `fit()` : 학습 메소드

```
z = self.net_input(X)
yhat = self.activate(z)
errors = (y - yhat)
self.w += self.eta * np.dot(errors, X)
```

$$\begin{aligned} W_{new} &= W_{old} + \Delta w \\ &= W_{old} + \eta \sum_i \left(y^{(i)} - h(z^{(i)}) \right) x_j^{(i)} \end{aligned}$$

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - `fit()` : 학습 메소드

```
z = self.net_input(X)
yhat = self.activate(z)
errors = (y - yhat)
self.w += self.eta * np.dot(errors, X)
```

$$W_{new} = W_{old} + \Delta W$$

$$= W_{old} + \eta \sum_i (y^{(i)} - h(z^{(i)})) x_j^{(i)}$$


```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```


2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5 * np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```




2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - `fit()` : 학습 메소드

```
cost = 0.5 * np.sum(errors**2)
```

$$J(w) = \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$


$$J(w) = \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \hat{y}^{(i)})^2$$


2. 아달라인 경사하강법 구현: 객체지향 아달라인

■ 클래스

- 이름: **AdalineGD**
- 생성자: **__init__()**
 - 객체생성, 인스턴스 변수 초기화
- **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치
 - **return self**
- **net_input()** : 순입력 메소드

```
1 def fit(self, X, y, X0=False):
2     if X0 == False:
3         X = np.c_[ np.ones(len(y)), X]
4
5     np.random.seed(self.random_seed)
6     self.w = np.random.random(X.shape[1])
7
8     self.maxy, self.miny = y.max(), y.min()
9     self.cost_ = []
10    self.w_ = np.array([self.w])
11    for i in range(self.epochs):
12        z = self.net_input(X)
13        yhat = self.activate(z)
14        errors = (y - yhat)
15        self.w += self.eta * np.dot(errors, X)
16        cost = 0.5*np.sum(errors**2)
17        self.cost_.append(cost)
18        self.w_ = np.vstack([self.w_, self.w])
19    return self
```



2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - 이름: **AdalineGD**
 - 생성자: **__init__()**
 - 객체생성, 인스턴스 변수 초기화
 - **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치
 - **return self**
 - **net_input()** : 순입력 메소드

```
23         self.cost_.append(cost)
24         self.w_ = np.vstack([self.w_, self.w])
25         return self
26
27     def net_input(self, X):
28         z = np.dot(X, self.w)
29         return z
30
31     def activate(self, X):
32         return X
33
34     def predict(self, X):
35         mid = (self.maxy
36               + self.miny) / 2
37         return np.where(
38             self.net_input(X) > mid,
39             self.maxy,
40             self.miny)
```

2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - 이름: **AdalineGD**
 - 생성자: **__init__()**
 - 객체생성, 인스턴스 변수 초기화
 - **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치
 - **return self**
 - **net_input()**: 순입력 메소드
 - **activate()**: 활성화 메소드

```
23         self.cost_.append(cost)
24         self.w_ = np.vstack([self.w_, self.w])
25         return self
26
27     def net_input(self, X):
28         z = np.dot(X, self.w)
29         return z
30
31     def activate(self, X):
32         return X
33
34     def predict(self, X):
35         mid = (self.maxy
36               + self.miny) / 2
37         return np.where(
38             self.net_input(X) > mid,
39             self.maxy,
40             self.miny)
```


2. 아달라인 경사하강법 구현: 객체지향 아달라인

- 클래스
 - 이름: **AdalineGD**
 - 생성자: **__init__()**
 - 객체생성, 인스턴스 변수 초기화
 - **fit()**: 학습 메소드
 - 매개변수
 - 가중치: 1차원 배열
 - **cost_**: 각 **epoch**의 손실
 - **w_**: 각 **epoch**의 가중치
 - **return self**
 - **net_input()**: 순입력 메소드
 - **activate()**: 활성화 메소드
 - **predict()**: 예측 메소드

```
23         self.cost_.append(cost)
24         self.w_ = np.vstack([self.w_, self.w])
25         return self
26
27     def net_input(self, X):
28         z = np.dot(X, self.w)
29         return z
30
31     def activate(self, X):
32         return X
33
34     def predict(self, X):
35         mid = (self.maxy
36               + self.miny) / 2
37         return np.where(
38             self.net_input(X) > mid,
39             self.maxy,
40             self.miny)
```

아달라인 경사하강법 구현

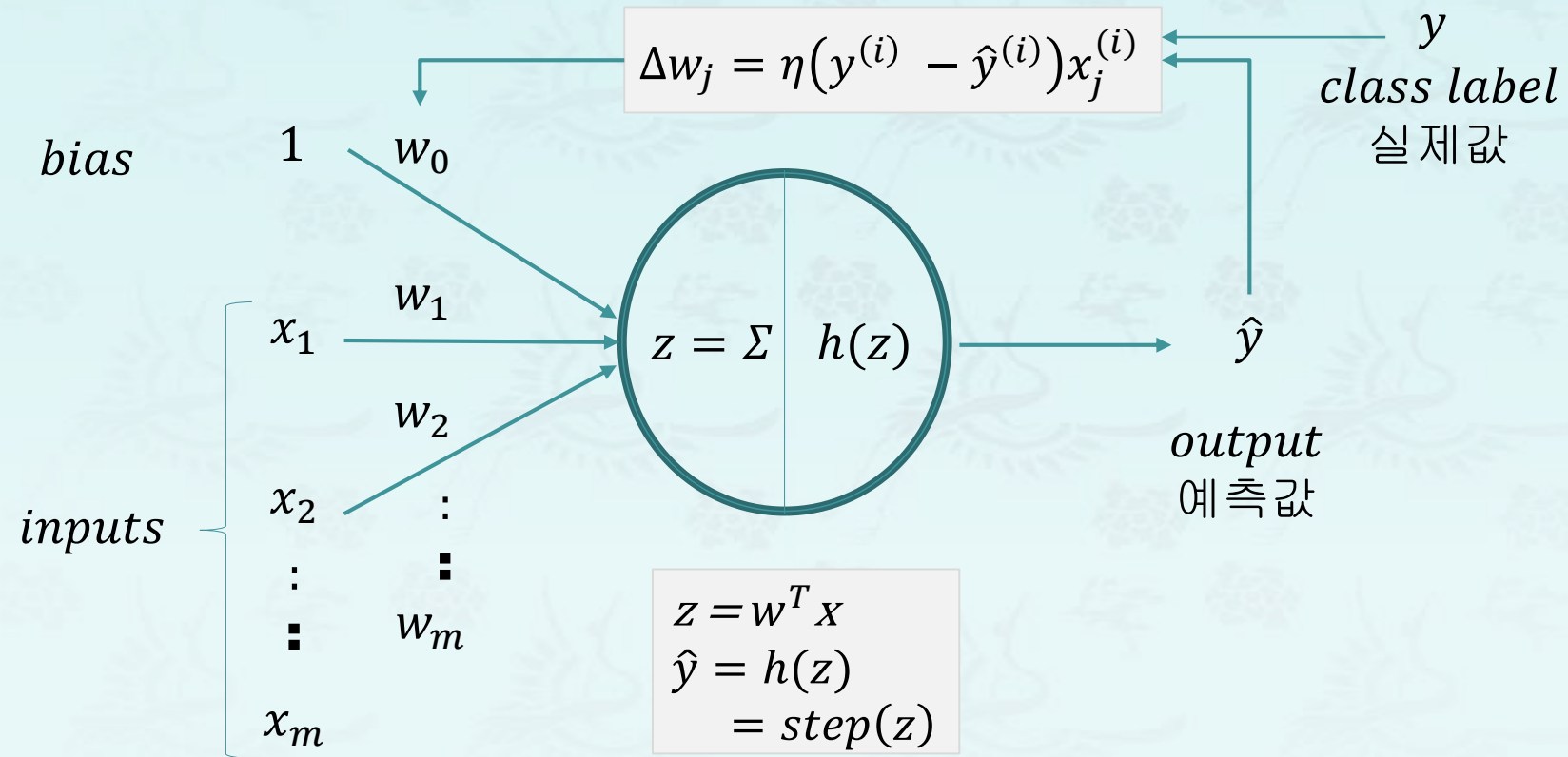
- 학습 정리
 - 경사하강법을 적용하여 가중치 조정하기
 - 스텝의 방향과 스텝의 크기(Δw)
 - 학습률의 크기
 - 아달라인 알고리즘 구현하기
- **8.2 아달라인 경사하강법 적용**

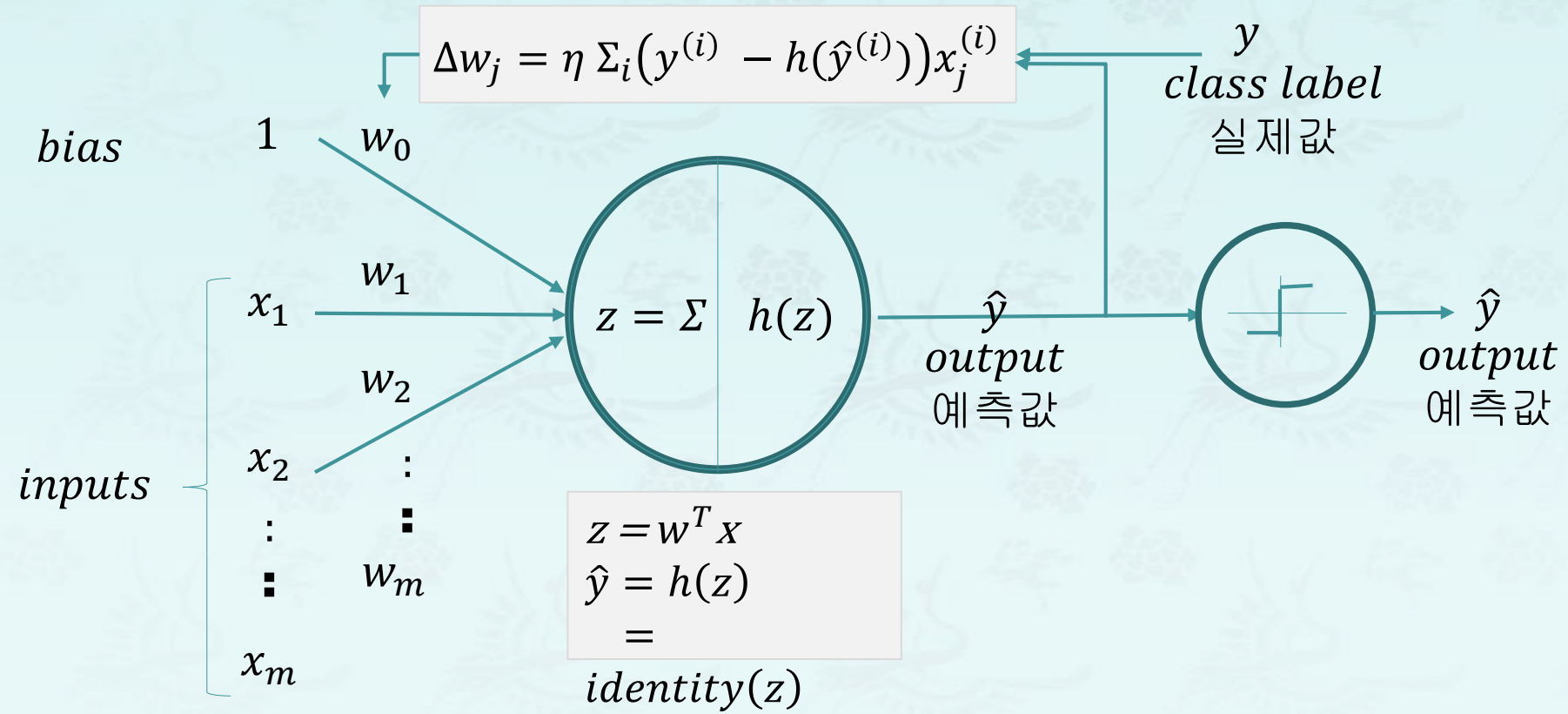
9주차(1/3)

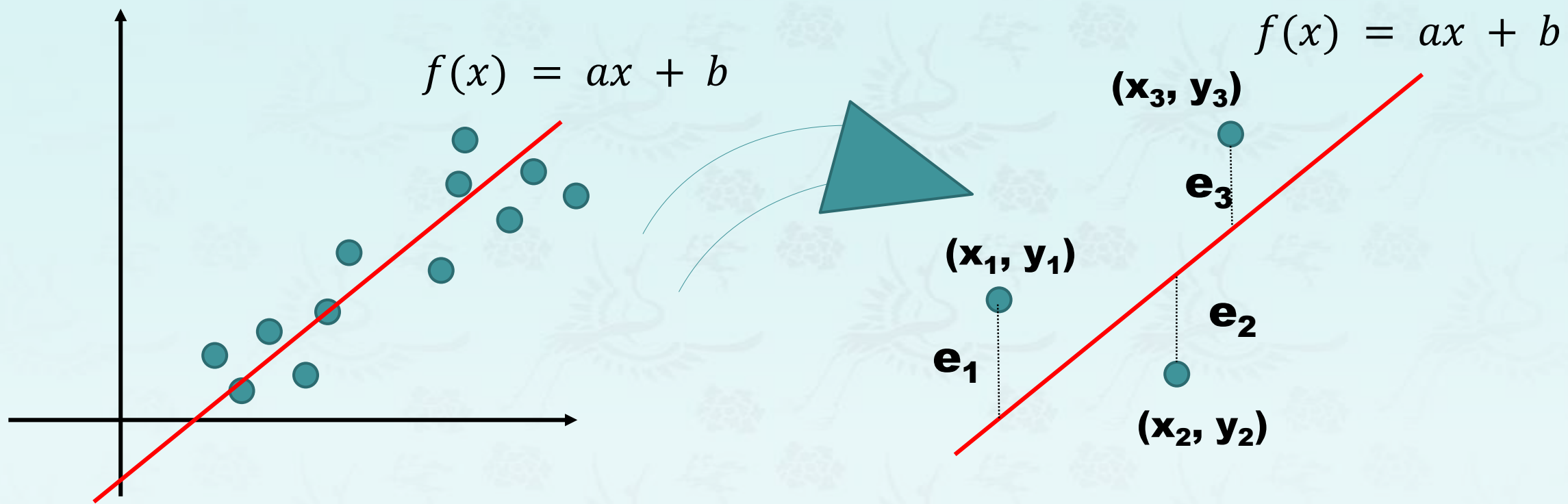
아달라인 경사하강법 구현

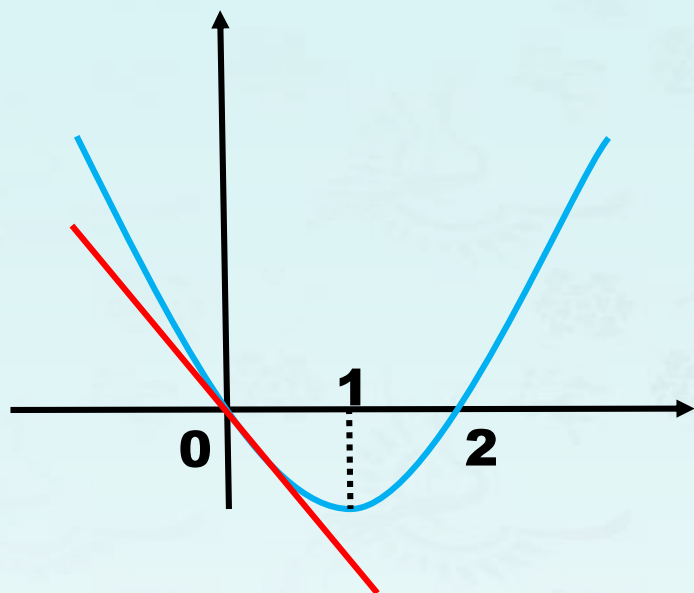
파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

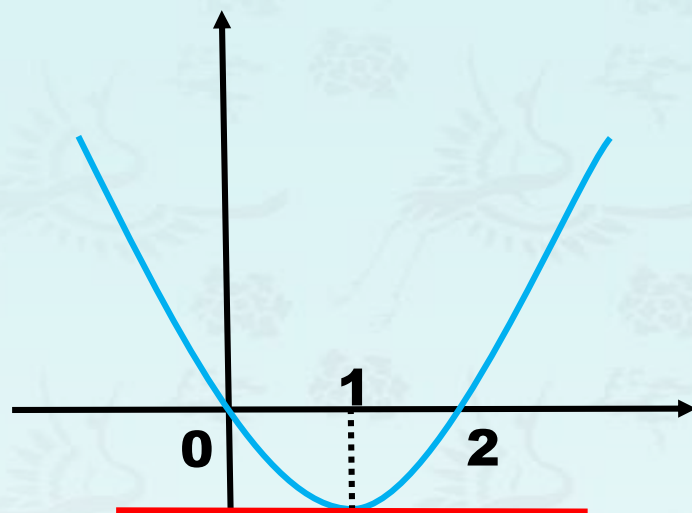




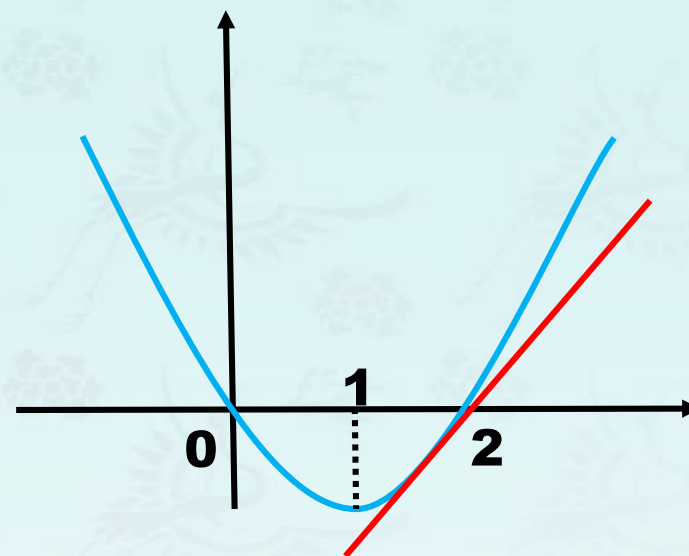




$$f'(x_1) < 0$$



$$f'(x_2) = 0$$



$$f'(x_3) > 0$$

