8주차(1/3)

아달라인 경사하강법 구현

파이썬으로 배우는 기계학습

한동대학교
김영섭 교수

아달라인 경사하강법 구현

- 학습 목표
  - 경사하강법의 가중치 조정 알고리즘을 학습한다.
  - 아달라인 경사하강법을 구현한다.

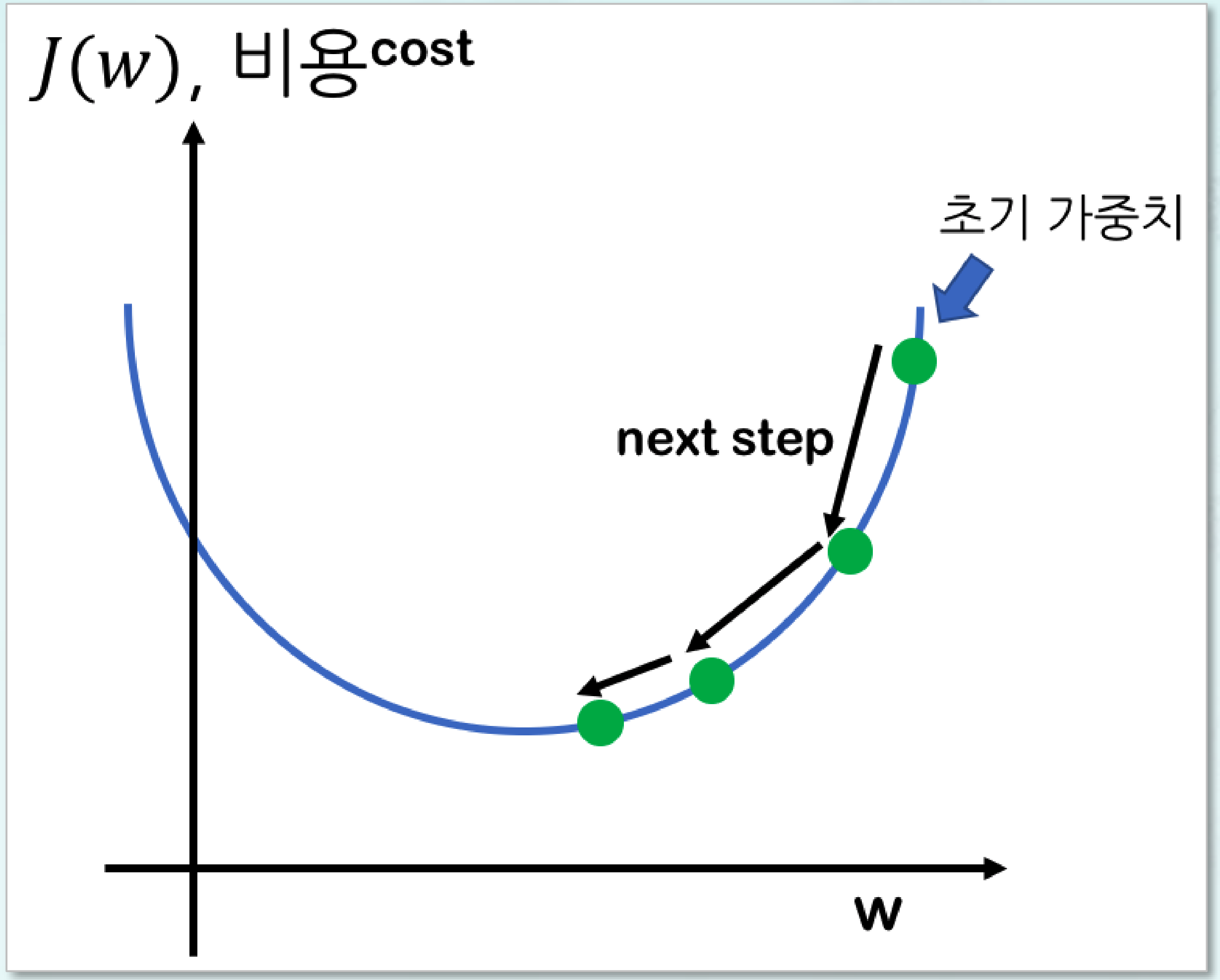
- 학습 내용
  - 경사하강법 가중치 조정 알고리즘
  - 경사하강법 스텝의 방향과 크기
  - 학습률
  - 아달라인 경사하강법 구현

1. 경사하강법: 비용함수

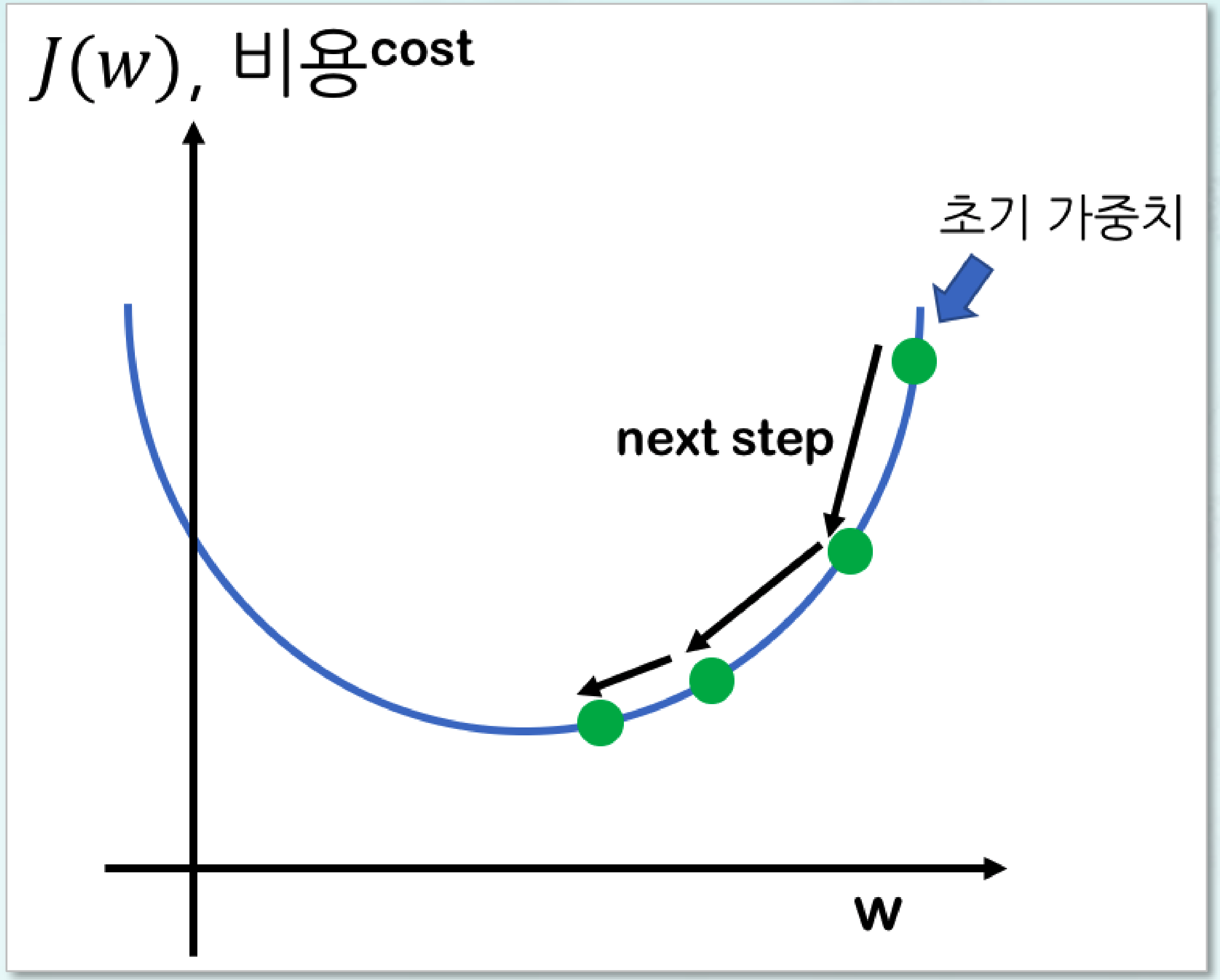
- 최소 제곱법을 이용한 비용 함수 $J(w)$

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

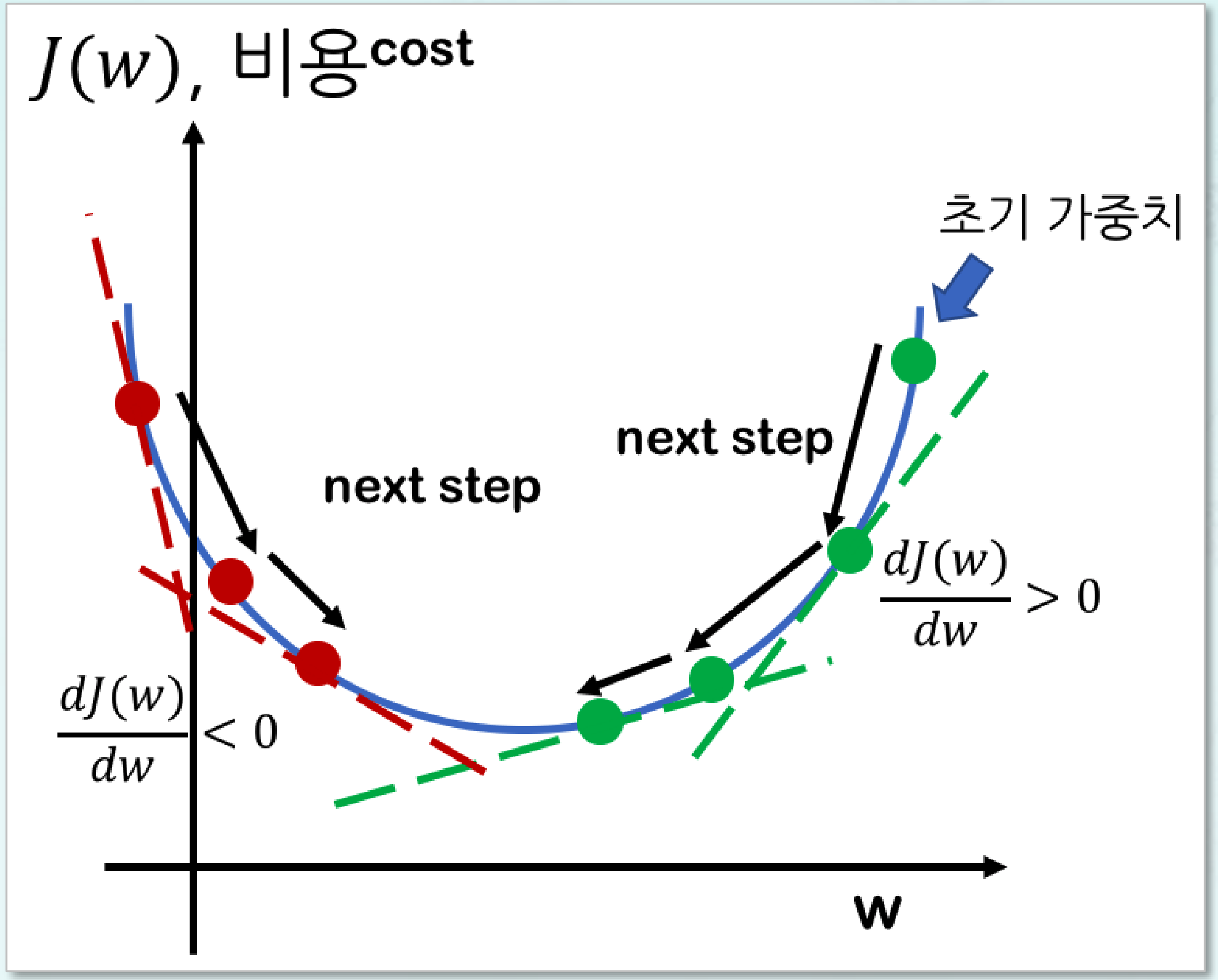
1. 경사하강법: 비용함수



1. 경사하강법: 스텝 방향

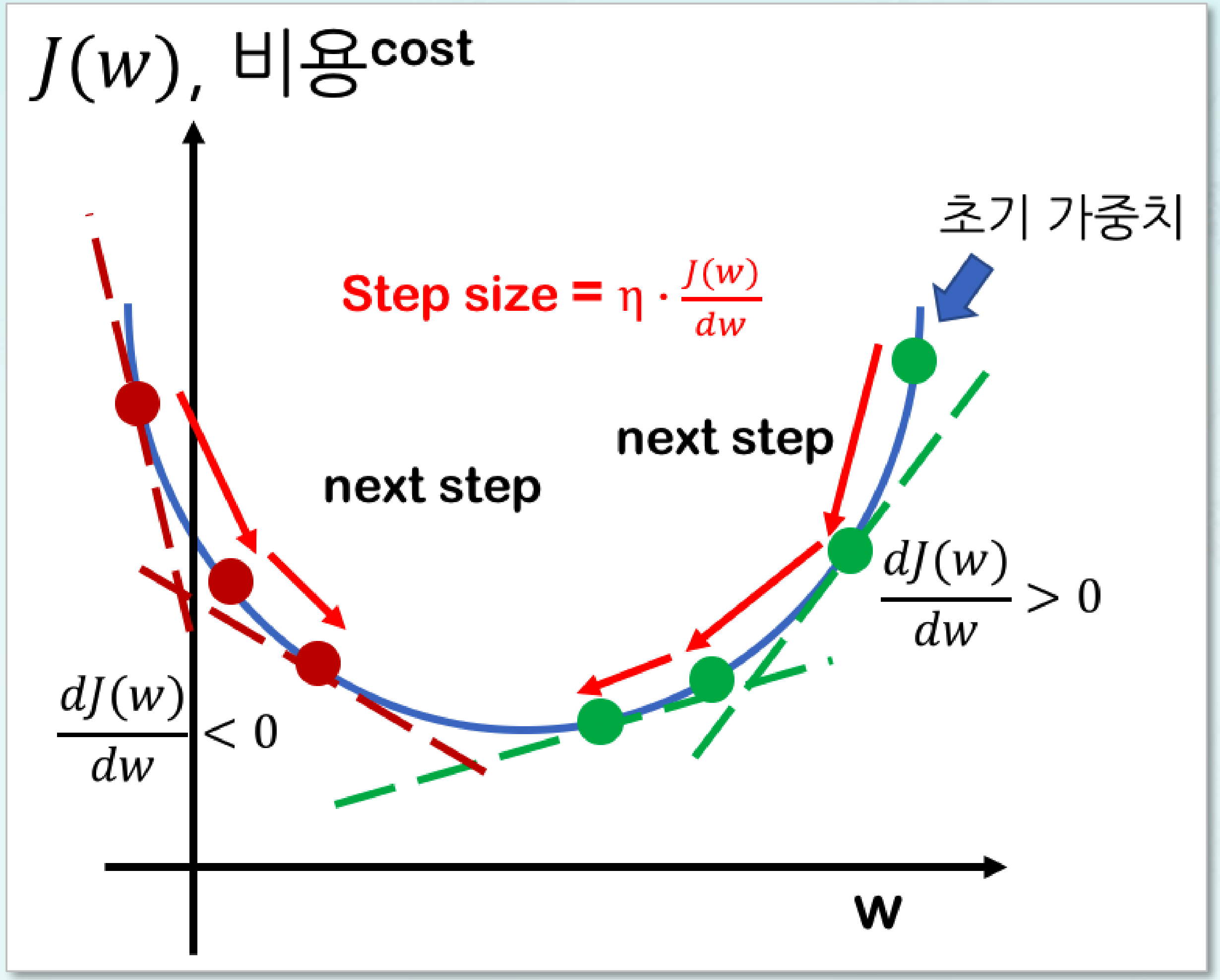


1. 경사하강법: 스텝 방향



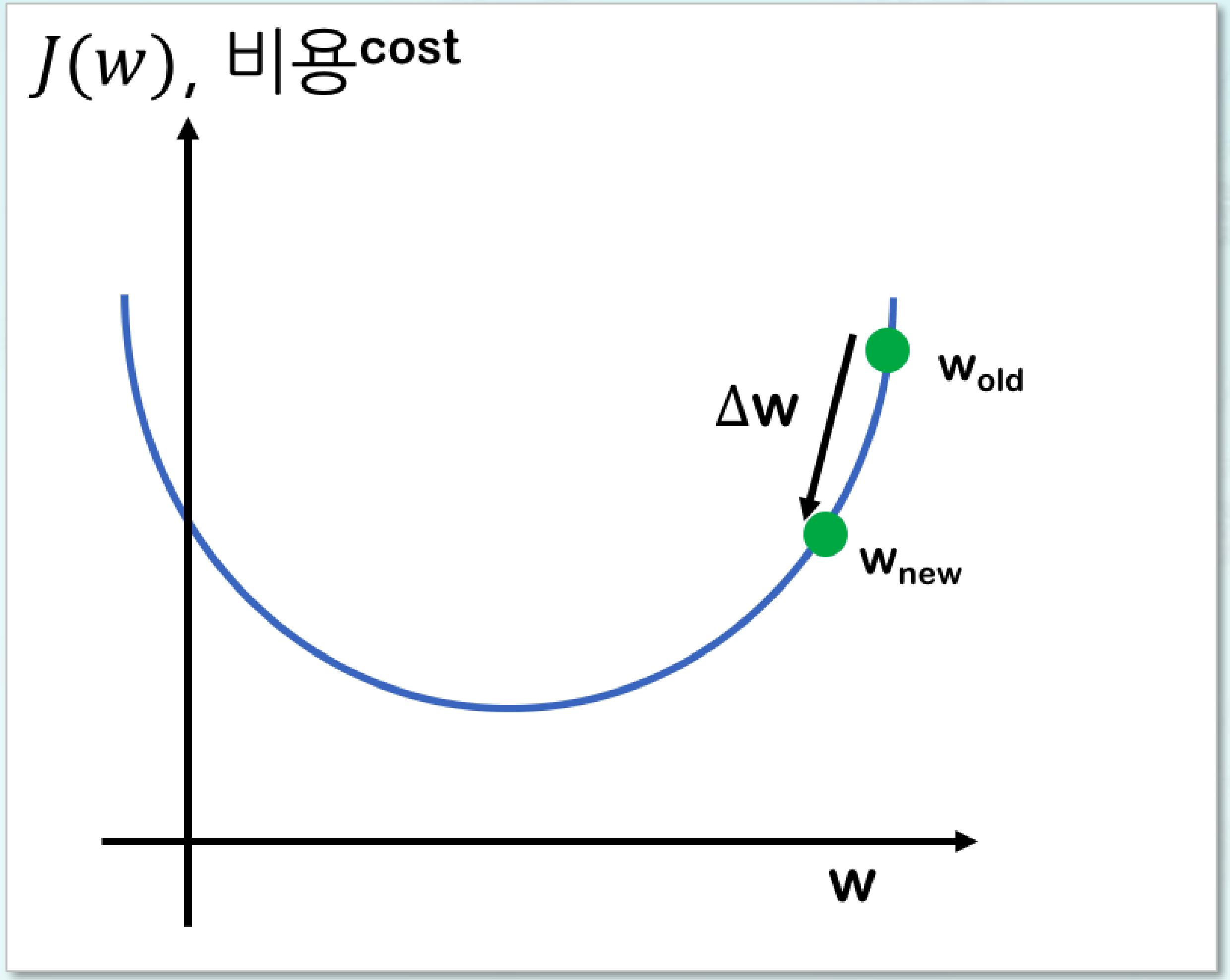
- $-\frac{dJ(w)}{dw}$

1. 경사하강법: 스텝 크기



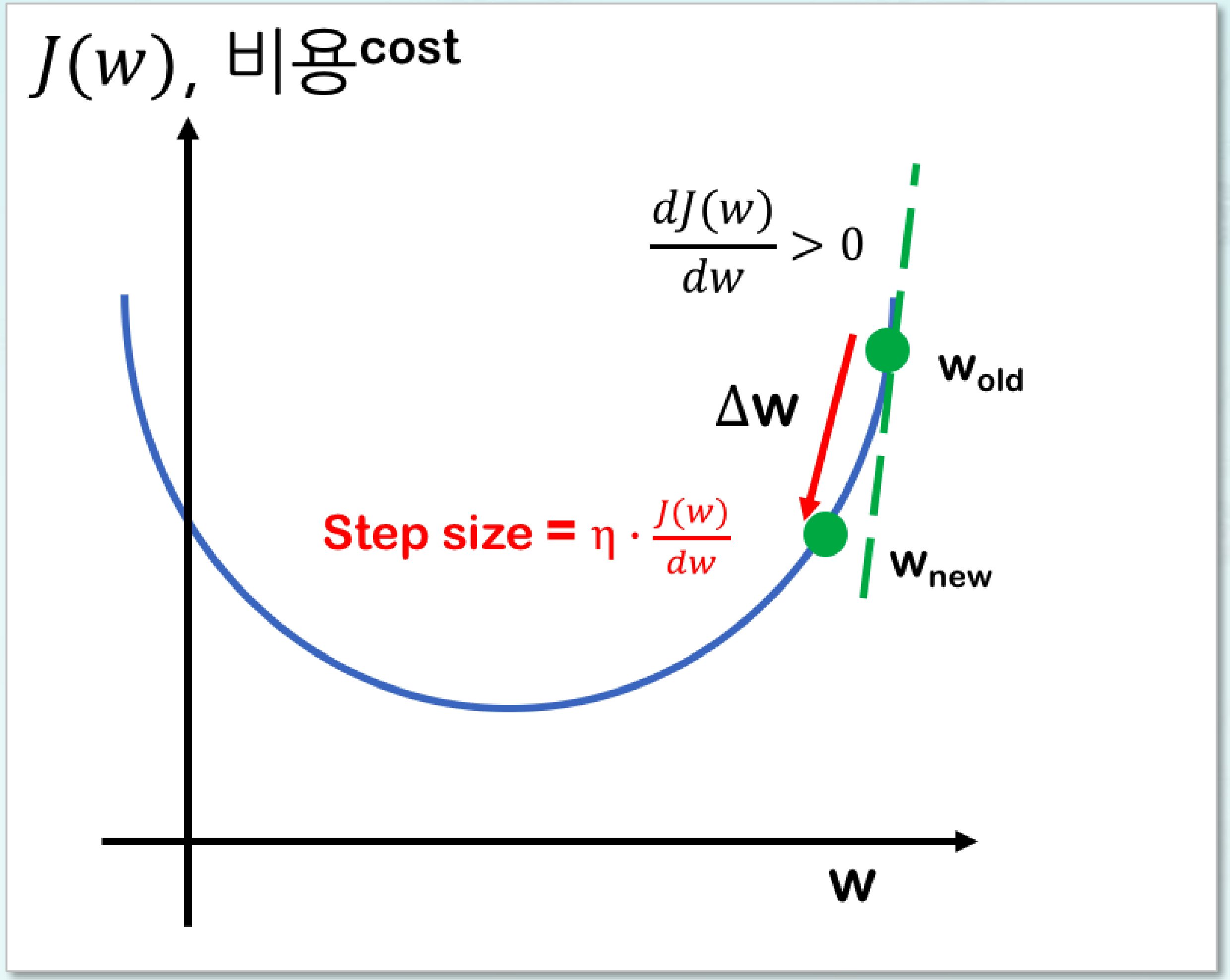
- $-\eta \cdot \frac{dJ(w)}{dw} = \Delta w$

1. 경사하강법: 가중치 조정



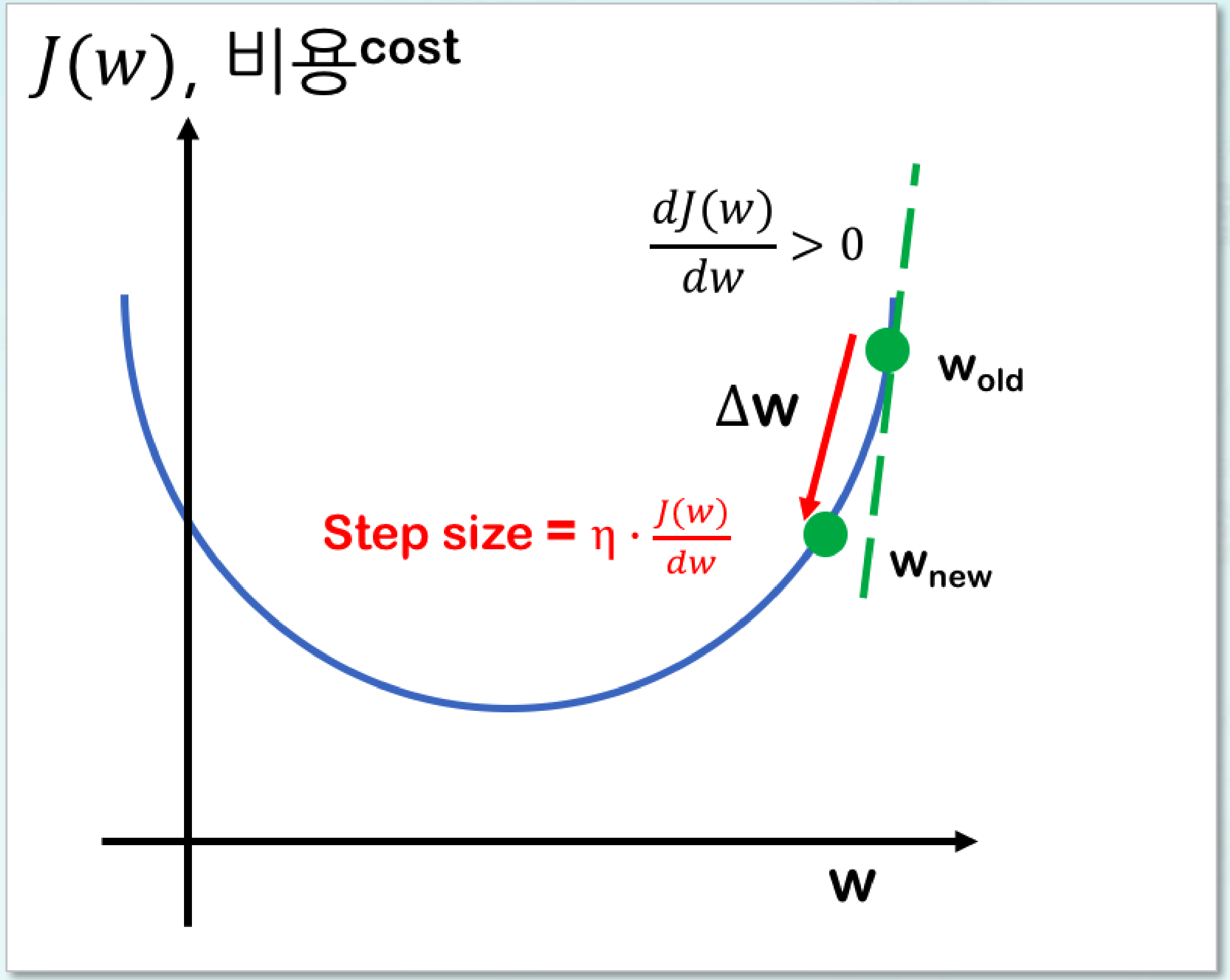
$$w_{new} = w_{old} + \Delta w$$

1. 경사하강법: 가중치 조정



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

1. 경사하강법: 가중치 조정



$$w_{new} = w_{old} + \Delta w$$
$$= w_j + \boxed{\eta \frac{\partial J(w)}{\partial w_j}}$$

1. 경사하강법: 가중치 조정

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

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

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

$$= \frac{1}{2}\sum_i 2\left(y^{(i)} - h\left(z^{(i)}\right)\right)\boxed{(-\sum_i x_j^{(i)})}$$

합성함수 미분법

$$f\left(g(x)\right)' = f'\left(g(x)\right)g'(x)$$

$$h(\cdot) = \ idedenty\ function$$
$$h(z) = \ z,\ z = \sum wx$$

$y^{(i)}, x_j^{(i)}$ 상수 취급

1. 경사하강법: 가중치 조정

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

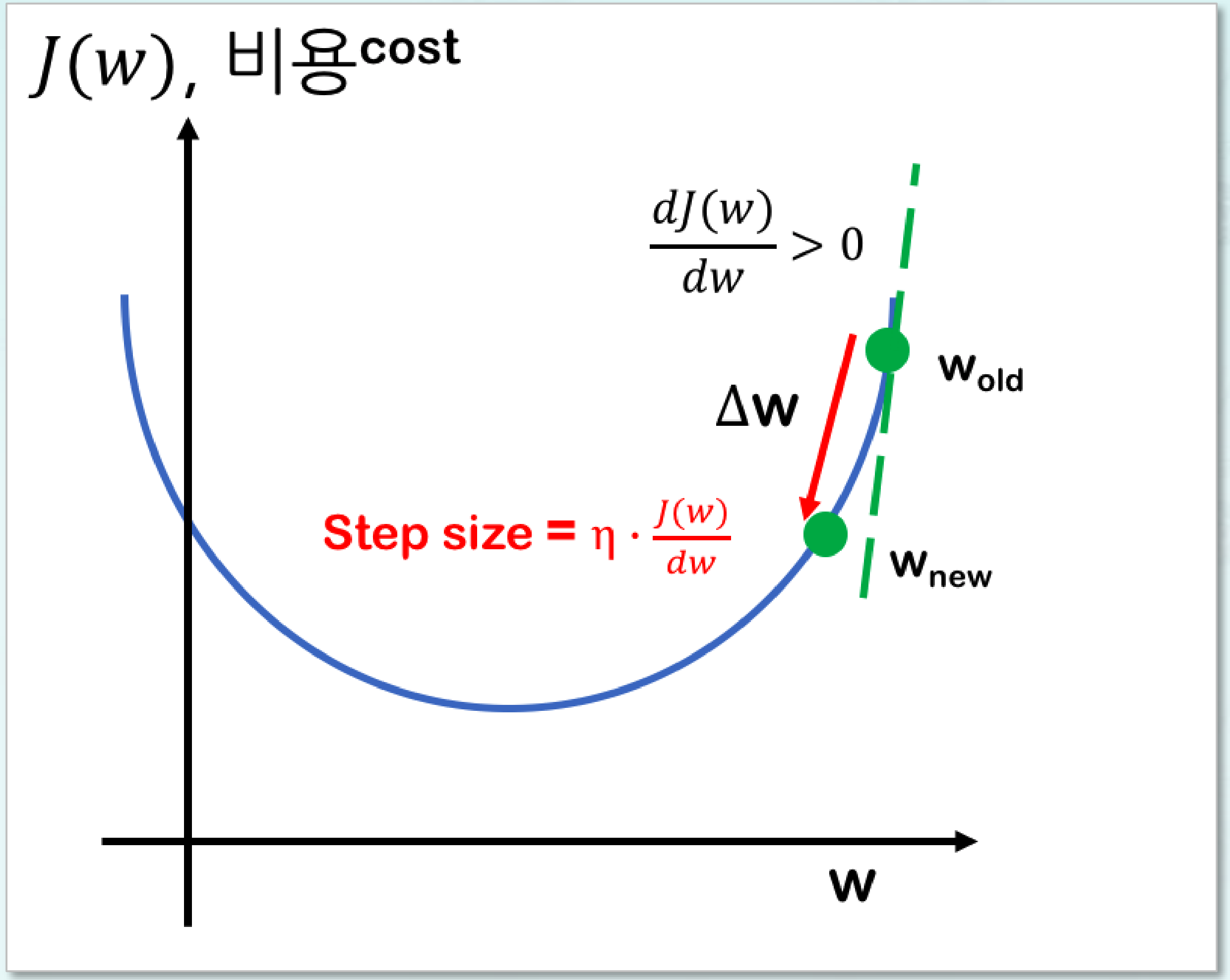
$$= \frac{1}{2}\sum_i 2\left(y^{(i)} - h\left(z^{(i)}\right)\right)\frac{\partial}{\partial w_j}(y^{(i)} - h(z^{(i)}))$$

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

$$= \frac{1}{2}\sum_i 2\left(y^{(i)} - h\left(z^{(i)}\right)\right)(-\sum_i x_j^{(i)})$$

$$= \boxed{-\sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right) x_j^{(i)}}$$

1. 경사하강법: 가중치 조정



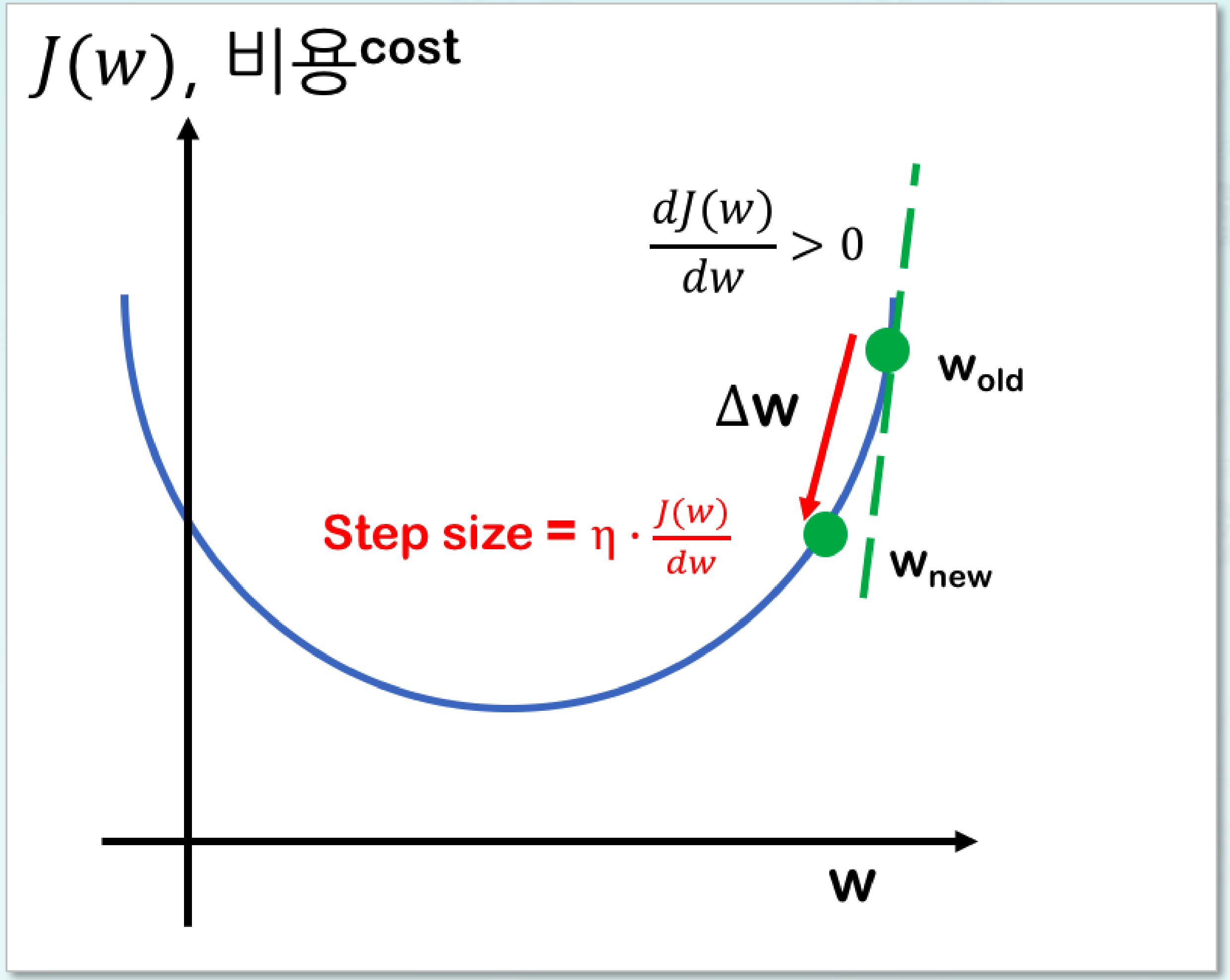
$$w_{new} = w_{old} + \Delta w$$

$$= w_{old} + \eta \frac{\partial J(w)}{\partial w_j}$$

$$= \boxed{w_{old} + \eta \sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right) x_j^{(i)}}$$

1. 경사하강법: 가중치 조정

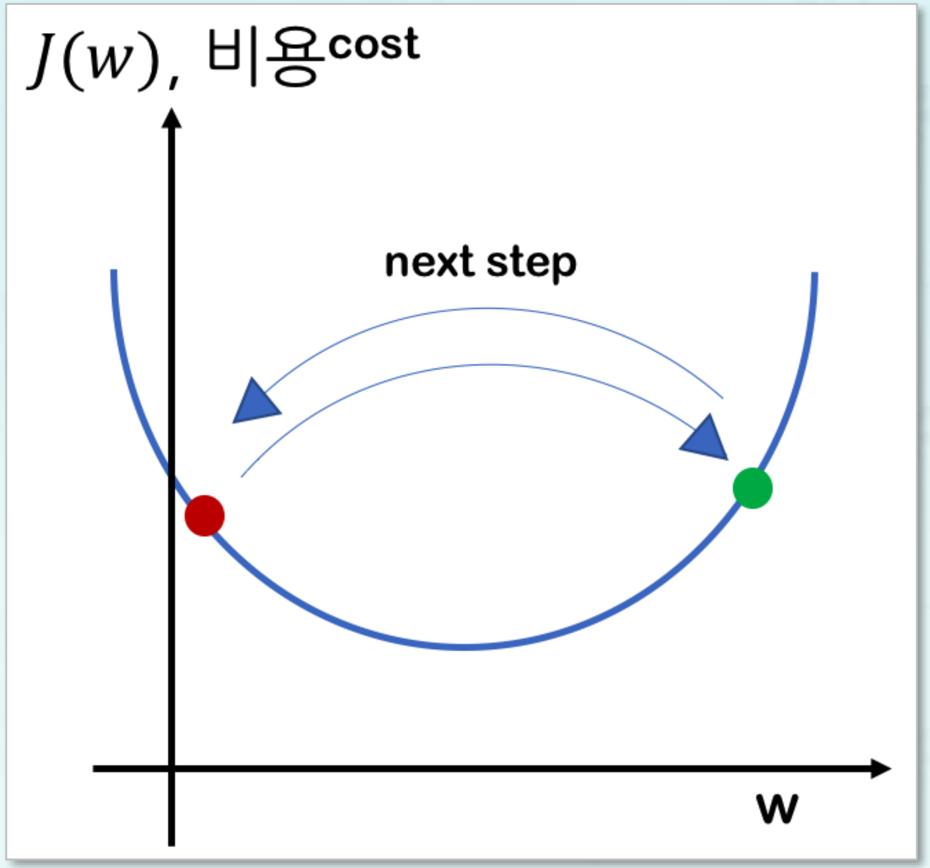
- **J(w)가 최소값을 수렴하지 않는 경우?**



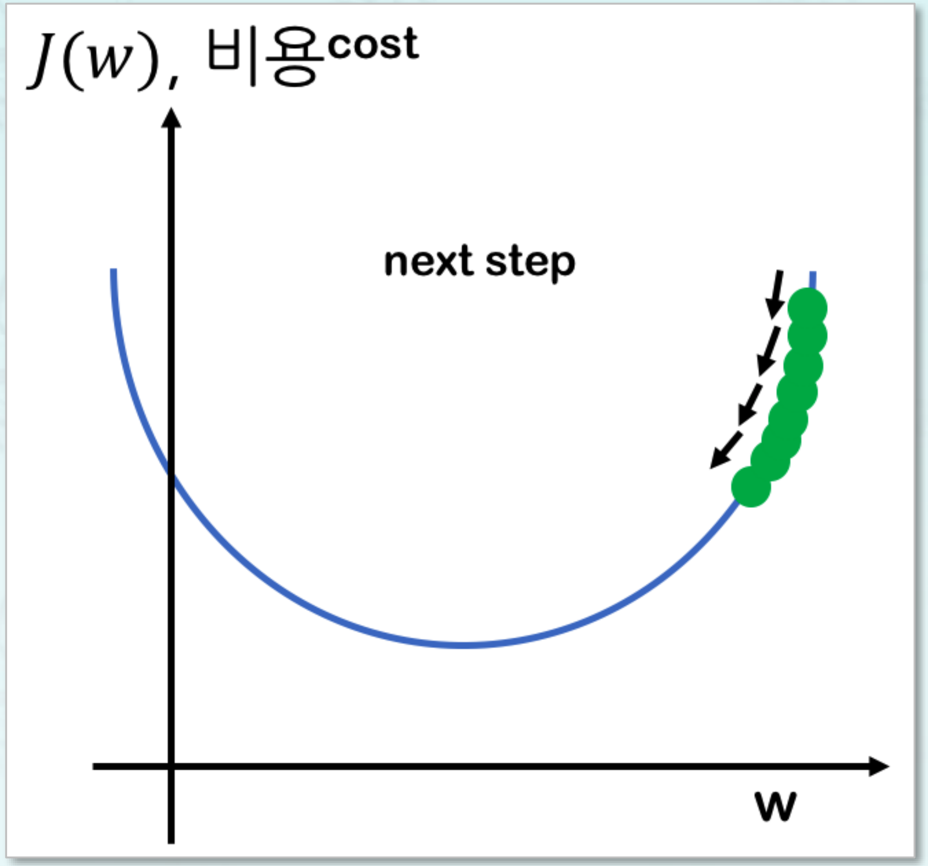
$$
\begin{aligned}
w_{new} &= w_{old} + \Delta w \\
&= w_{old} + \eta \frac{\partial J(w)}{\partial w_j} \\
&= \boxed{w_{old} + \eta \sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right) x_j^{(i)}}
\end{aligned}
$$

1. 경사하강법: 학습률

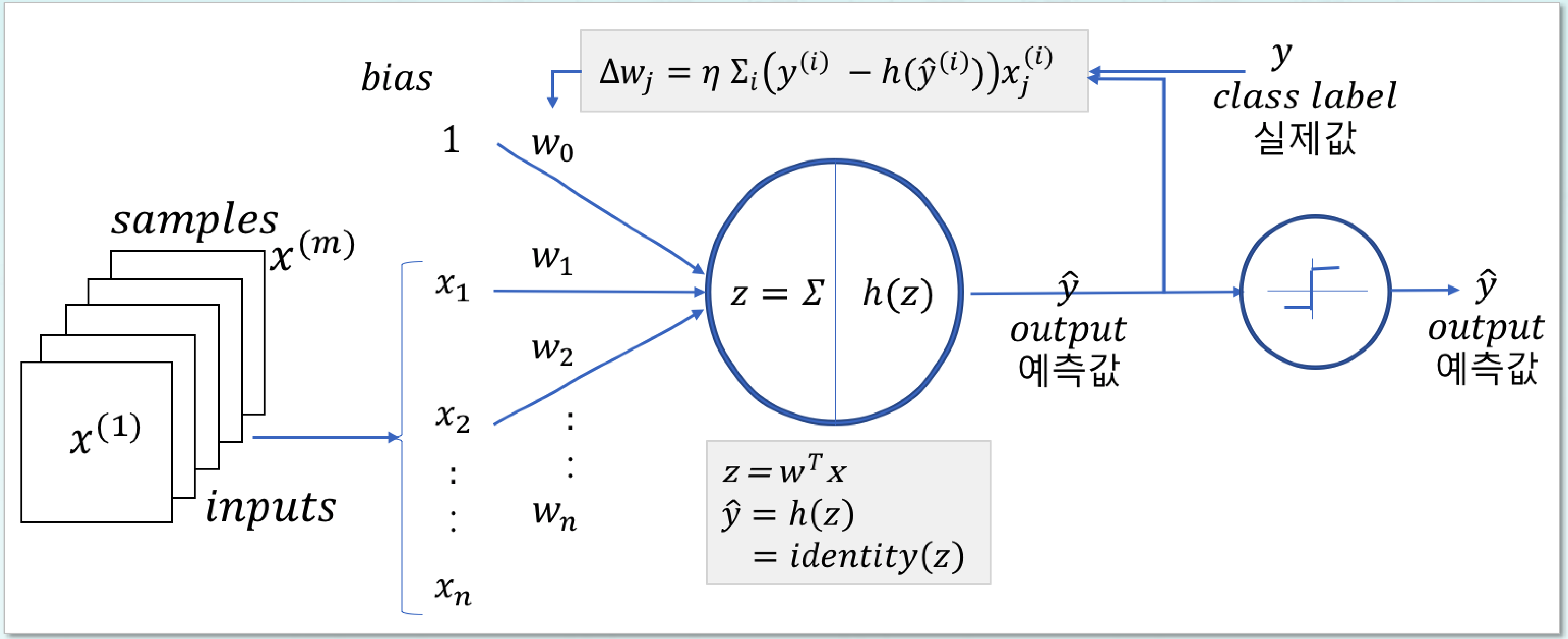
- 학습률이 너무 클 경우 $(\eta : \uparrow)$



- 학습률이 너무 작은 경우 $(\eta : \downarrow)$



2. 객체지향 아달라인 구현 준비: 아달라인 작업 흐름도



2. 객체지향 아달라인 구현 준비: 필요한 속성 (인스턴스 변수)

1. 입력 (x)
2. 출력 (y)
3. 순입력 (z)
4. 레이블 (yhat)
5. 가중치 (w)
6. 학습률 (eta)
7. 반복횟수 (epochs)
8. 랜덤시드 (random_seed)

2. 객체지향 아달라인 구현 준비: 필요한 메소드

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

3. 객체지향 아달라인 구현 코딩: 클래스 이름

- **Adaline Gradient Descent**

```python
class AdalineGD(object):
    """Adaptive Linear Neuron Classifier"""
    def __init__(self, eta=0.01, epochs=10, random_seed=1):
        self.eta = eta
        self.epochs = epochs
        self.random_seed = random_seed

    def fit(self, X, y):
        np.random.seed(self.random_seed)
        self.w = np.random.random(size = X.shape[1] + 1)
```

3. 객체지향 아달라인 구현 코딩: 생성자

- __init__()
  - 객체 생성, 인스턴스 변수 초기화

```python
class AdalineGD(object):
    """Adaptive Linear Neuron Classifier"""
    def __init__(self, eta=0.01, epochs=10, random_seed=1):
        self.eta = eta
        self.epochs = epochs
        self.random_seed = random_seed

    def fit(self, X, y):
        np.random.seed(self.random_seed)
        self.w = np.random.random(size = X.shape[1] + 1)
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

- 가중치 : **1**차원 배열로 설정, 편향 포함

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

- **cost_ : 각 epoch의 손실**
- **w_ : 각 epoch의 가중치**

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

가중치
조정 구문

3. 객체지향 아달라인 구현 코딩: fit() 메소드

$$w_{new} = w_{old} + \Delta w$$
$$= w_{old} + \eta \sum_i \left(y^{(i)} - h\left(\boxed{z^{(i)}}\right) \right) x_j^{(i)}$$

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

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

$$= w_{old} + \eta \sum_i \left(y^{(i)} - \boxed{h\left(z^{(i)}\right)} \right) x_j^{(i)}$$

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

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

$$= w_{old} + \eta \sum_i \boxed{\left(y^{(i)} - h\left(z^{(i)}\right)\right)} x_j^{(i)}$$

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

$$w_{new} = w_{old} + \Delta w$$
$$= w_{old} + \boxed{\eta \sum_i \left(y^{(i)} - h\left(z^{(i)}\right)\right) x_j^{(i)}}$$

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

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

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

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: fit() 메소드

- **cost_ : 각 epoch의 손실**
- **w_ : 각 epoch의 가중치**

```python
def fit(self, X, y):
    np.random.seed(self.random_seed)
    # w size is increased by one for bias
    self.w = np.random.random(size=X.shape[1]+1)

    self.maxy = y.max()
    self.miny = y.min()

    self.cost_ = []
    self.w_ = np.array([self.w])

    for i in range(self.epochs):
        Z = self.net_input(X)
        yhat = self.activation(Z)
        errors = (y - yhat)
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0] += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self
```

3. 객체지향 아달라인 구현 코딩: net_input() 메소드

- 순입력 메소드
- 입력값과 가중치를 곱해서 순입력 값 반환

```python
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0]  += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self

    def net_input(self, X):
        z = np.dot(X, self.w[1:]) + self.w[0]
        return z

    def activation(self, X):
        return X

    def predict(self, X):
        mid = (self.maxy + self.miny) / 2
        Z = self.net_input(X)
        yhat = self.activation(Z)
        return np.where(yhat > mid, self.maxy, self.miny)
```

3. 객체지향 아달라인 구현 코딩: activation() 메소드

- 활성화 함수
- **Identity** 함수
- 매개변수 **X**를 받은 그대로 반환

```
            self.w[1:] += self.eta * np.dot(errors, X)
            self.w[0]  += self.eta * np.sum(errors)
            cost = 0.5 * np.sum(errors**2)
            self.cost_.append(cost)
            self.w_ = np.vstack([self.w_, self.w])
        return self

    def net_input(self, X):
        z = np.dot(X, self.w[1:]) + self.w[0]
        return z

    def activation(self, X):
        return X

    def predict(self, X):
        mid = (self.maxy + self.miny) / 2
        Z = self.net_input(X)
        yhat = self.activation(Z)
        return np.where(yhat > mid, self.maxy, self.miny)
```

3. 객체지향 아달라인 구현 코딩: predict() 메소드

- 예측 메소드
- 계단함수 사용
- 이미 학습된 가중치로 입력 분류

```
        self.w[1:] += self.eta * np.dot(errors, X)
        self.w[0]  += self.eta * np.sum(errors)
        cost = 0.5 * np.sum(errors**2)
        self.cost_.append(cost)
        self.w_ = np.vstack([self.w_, self.w])
    return self

    def net_input(self, X):
        z = np.dot(X, self.w[1:]) + self.w[0]
        return z

    def activation(self, X):
        return X

    def predict(self, X):
        mid = (self.maxy + self.miny) / 2
        Z = self.net_input(X)
        yhat = self.activation(Z)
        return np.where(yhat > mid, self.maxy, self.miny)
```

8-1 아달라인 경사하강법 구현

- 학습 정리
  - 경사하강법을 적용한 가중치 조정
  - 스텝의 방향과 스텝의 크기(Δw)
  - 학습률의 크기
  - 아달라인 알고리즘 구현