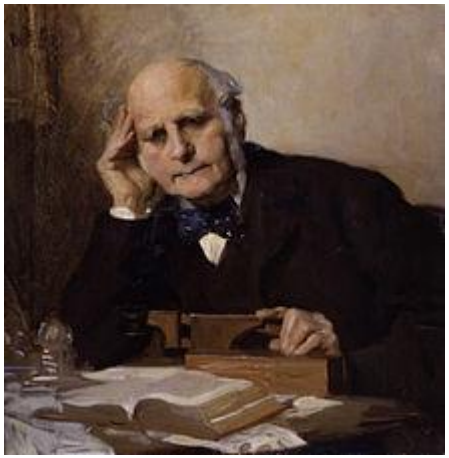
구현으로 배우는 딥러닝 이해

강사명 : 김정인

email : jikim@imguru.co.kr

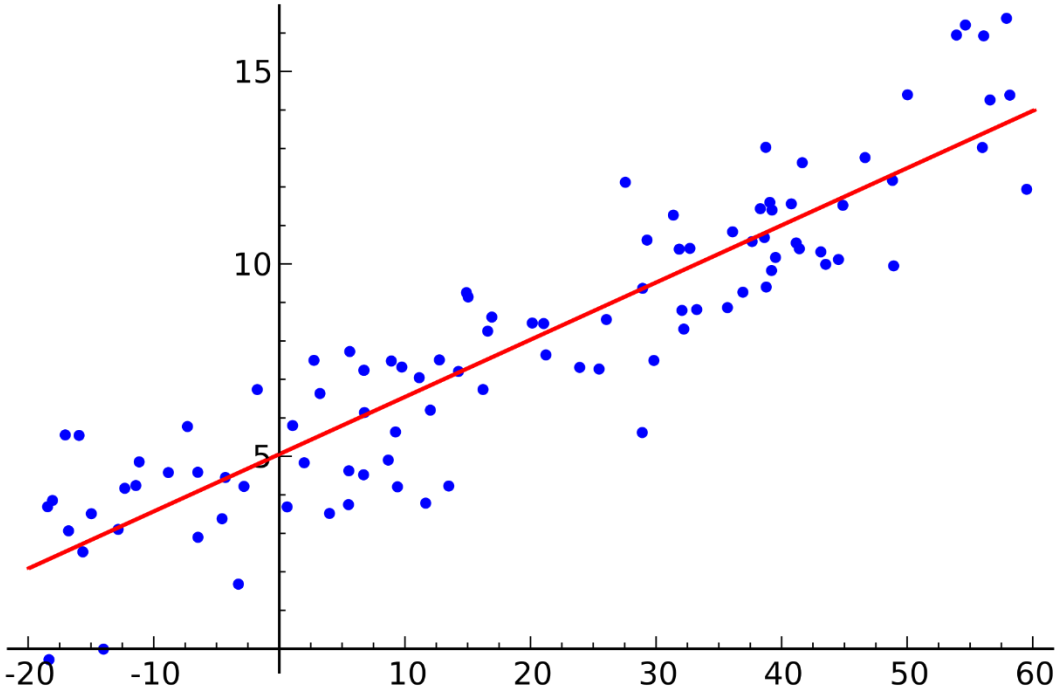
Regression - 회귀

"Regression toward the mean"



Sir Francis Galton
(1822 ~ 1911)

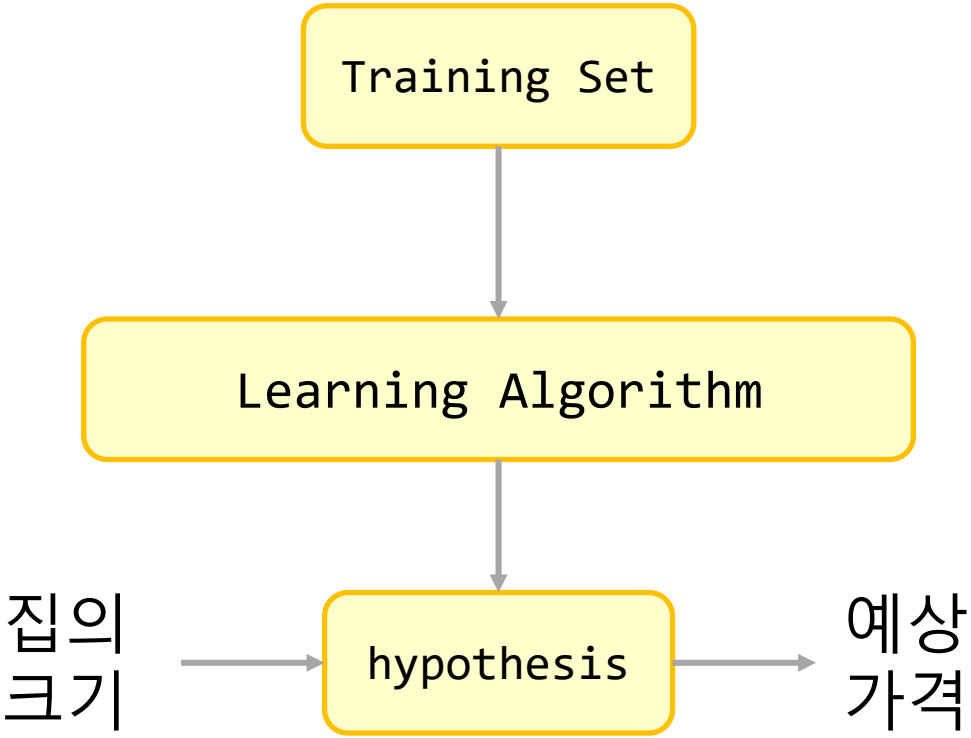
Linear Regression - 선형 회귀



$$y = ax + b$$

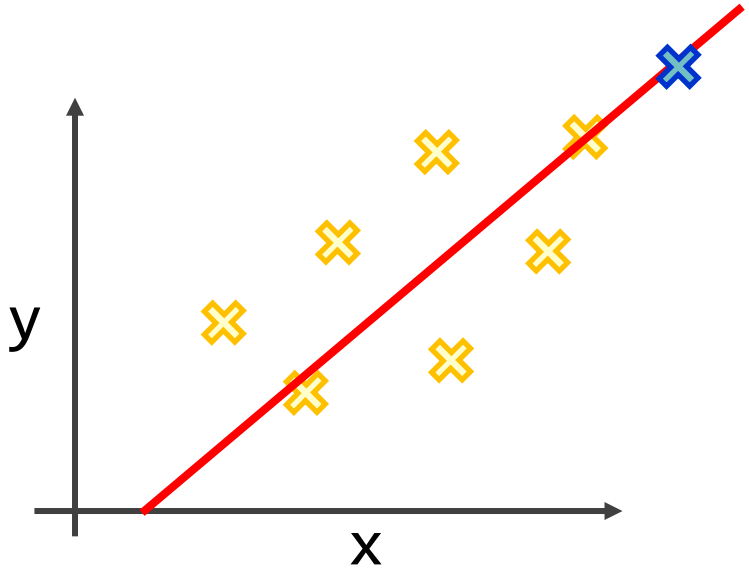
$$y = wx + b$$

https://en.wikipedia.org/wiki/Linear_regression



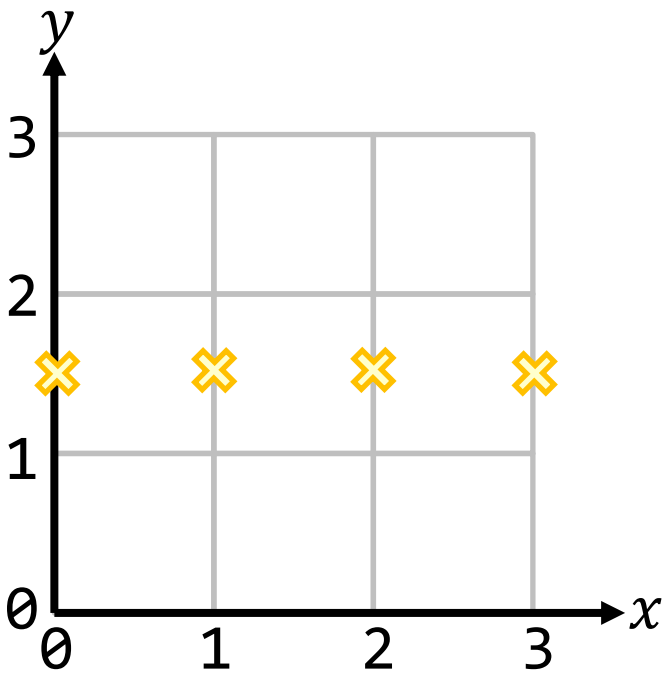
hypothesis ?

$$\hat{y} = wx + b$$

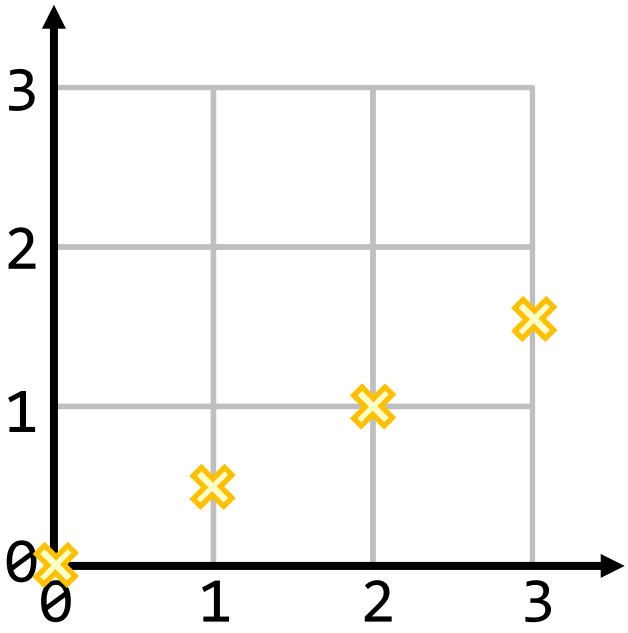


Cost function

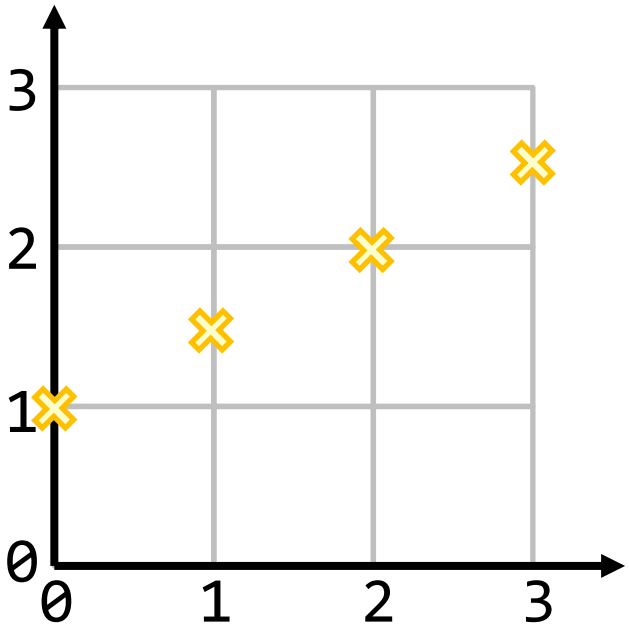
$$\hat{y} = wx + b$$



w = 0
b = 1.5



w = 0.5
b = 0



w = 0.5
b = 1

Hypothesis :

$$\hat{y} = wx + b$$

Parameters:

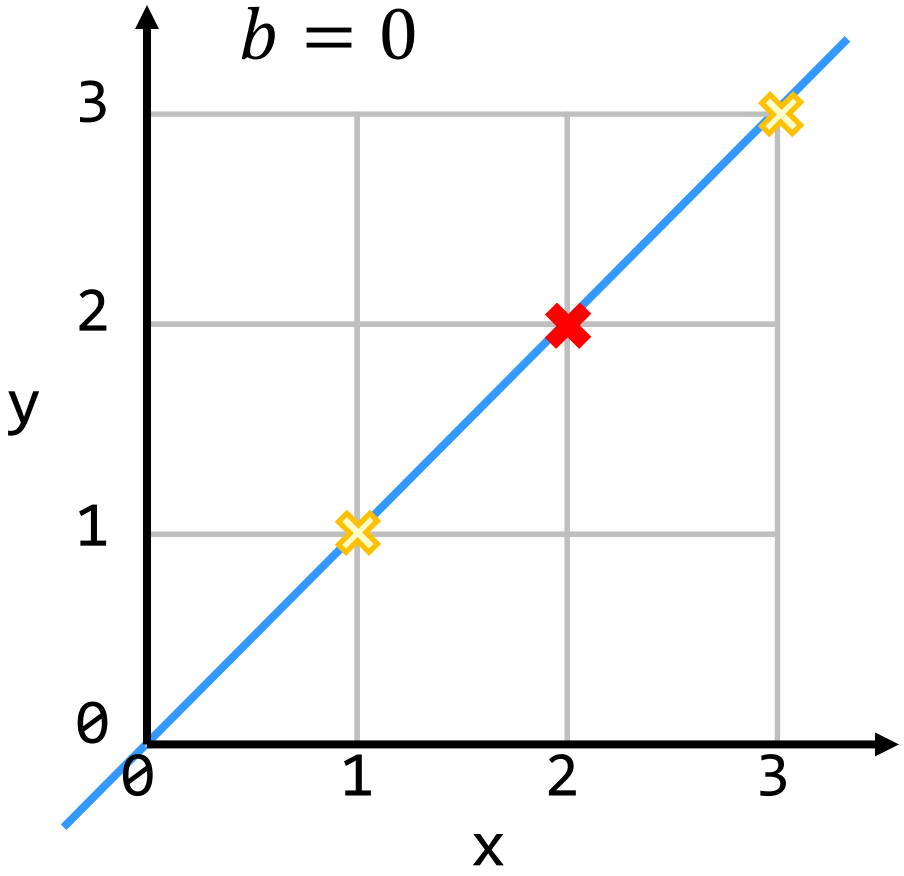
$$w, b$$

Cost Function

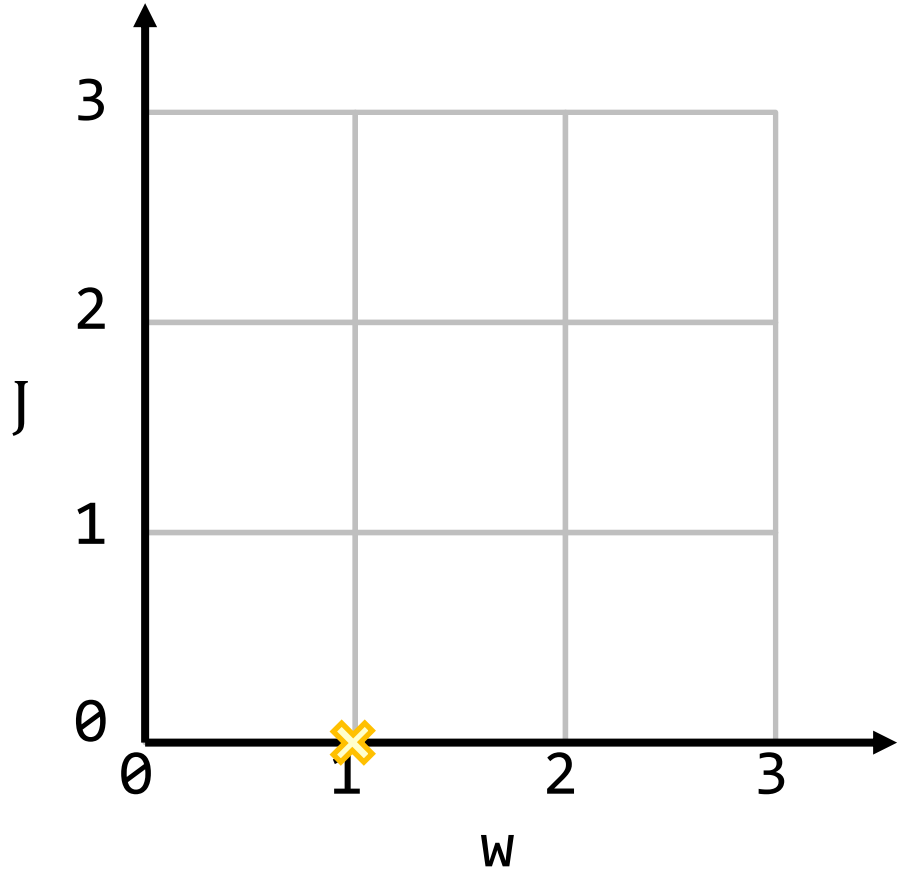
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

Cost function

$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 0$$

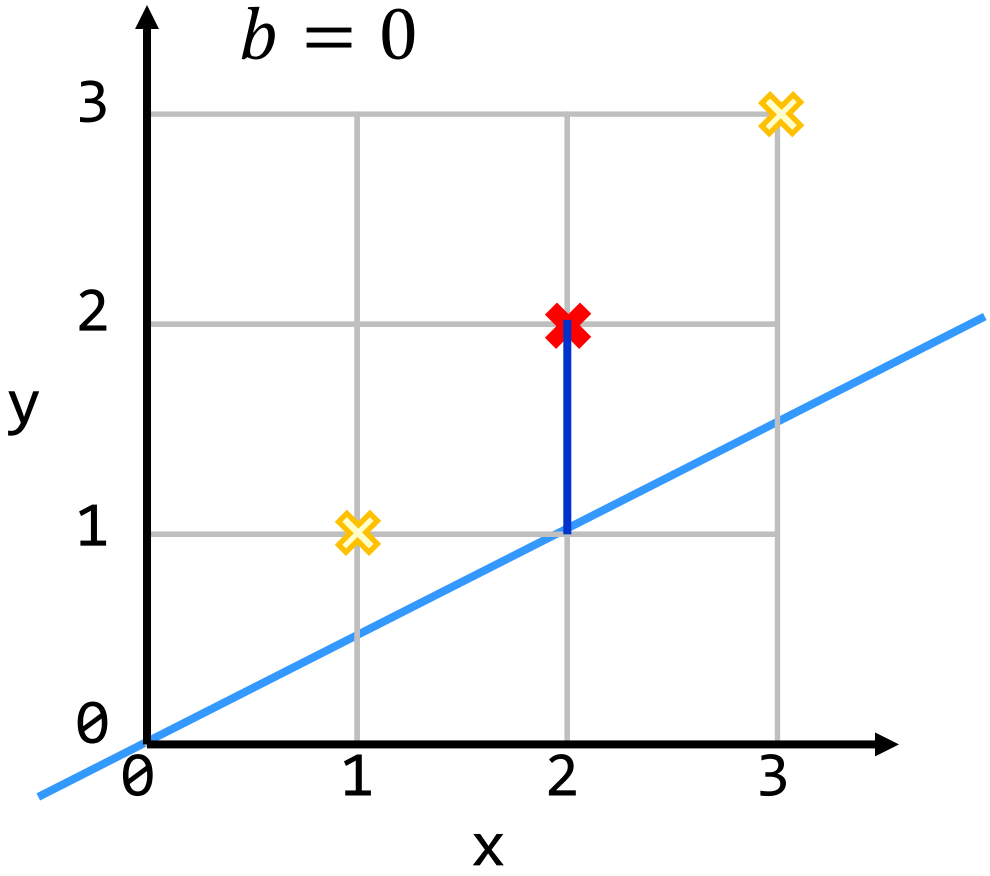


$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$

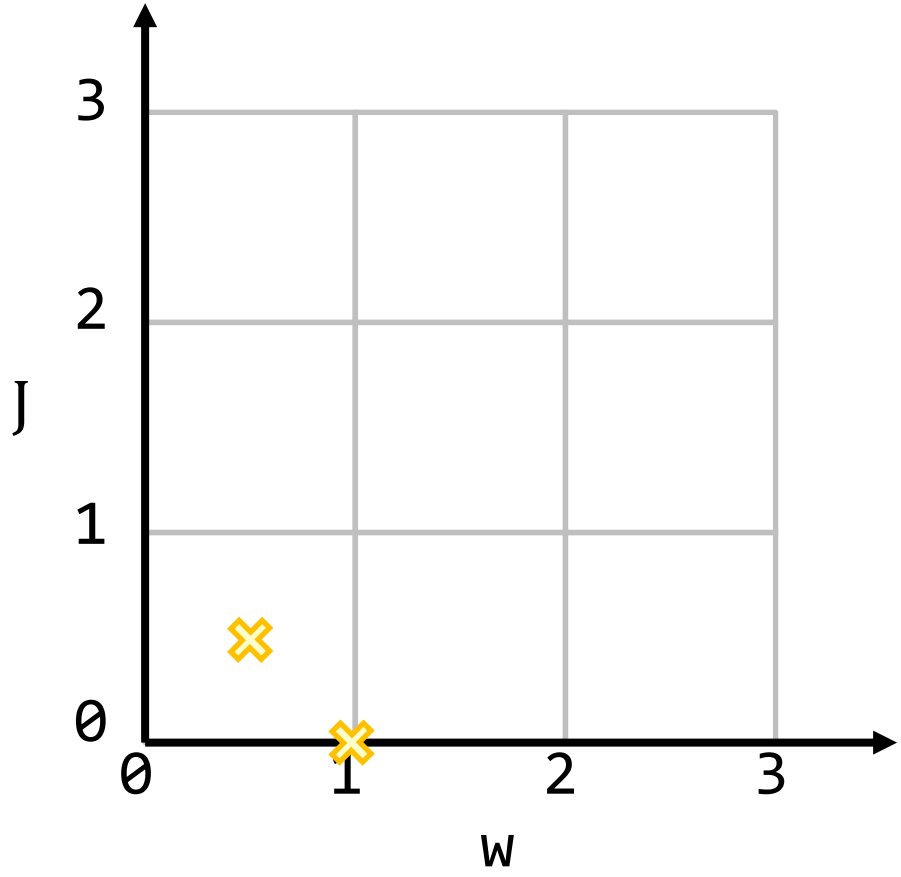


Cost function

$$\hat{y} = wx + b$$
$$w = 0.5$$
$$b = 0$$

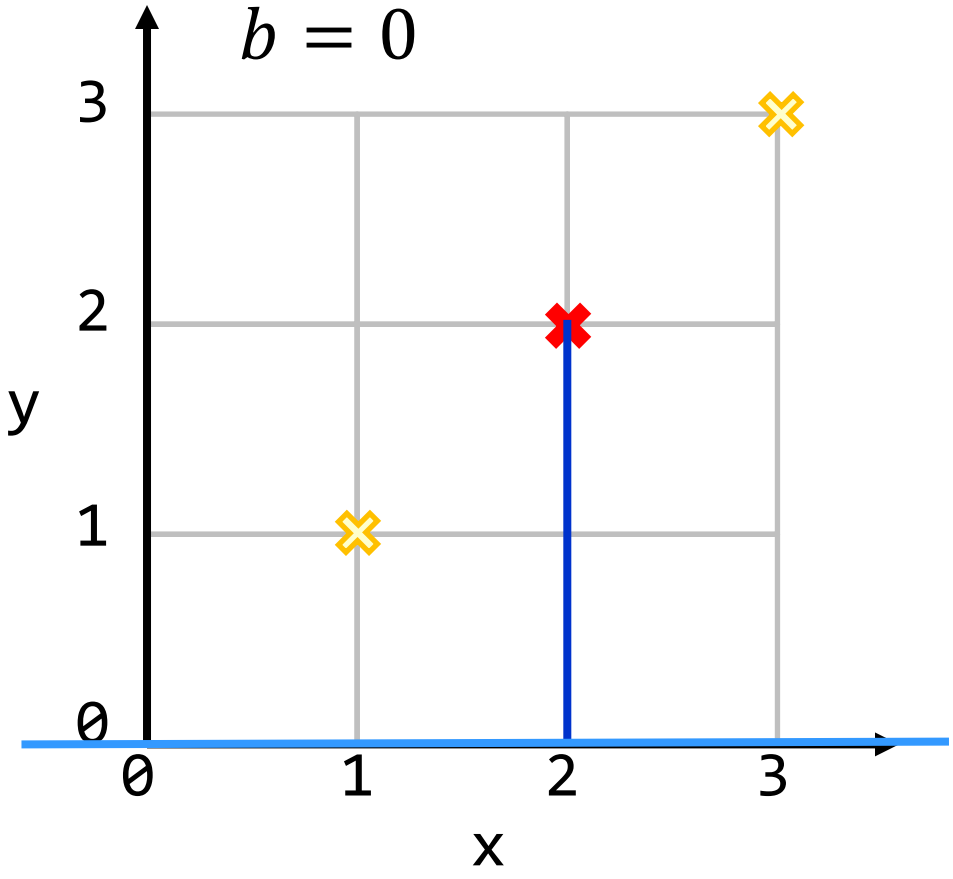


$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$

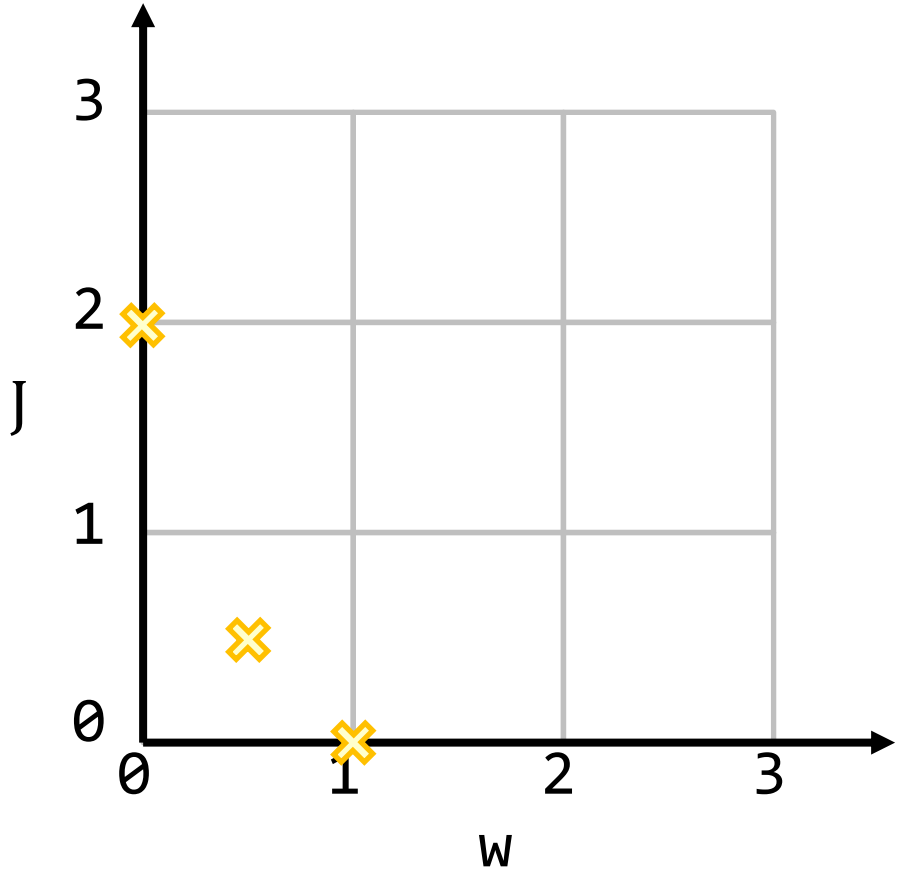


Cost function

$$\hat{y} = wx + b$$
$$w = 0$$
$$b = 0$$



$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

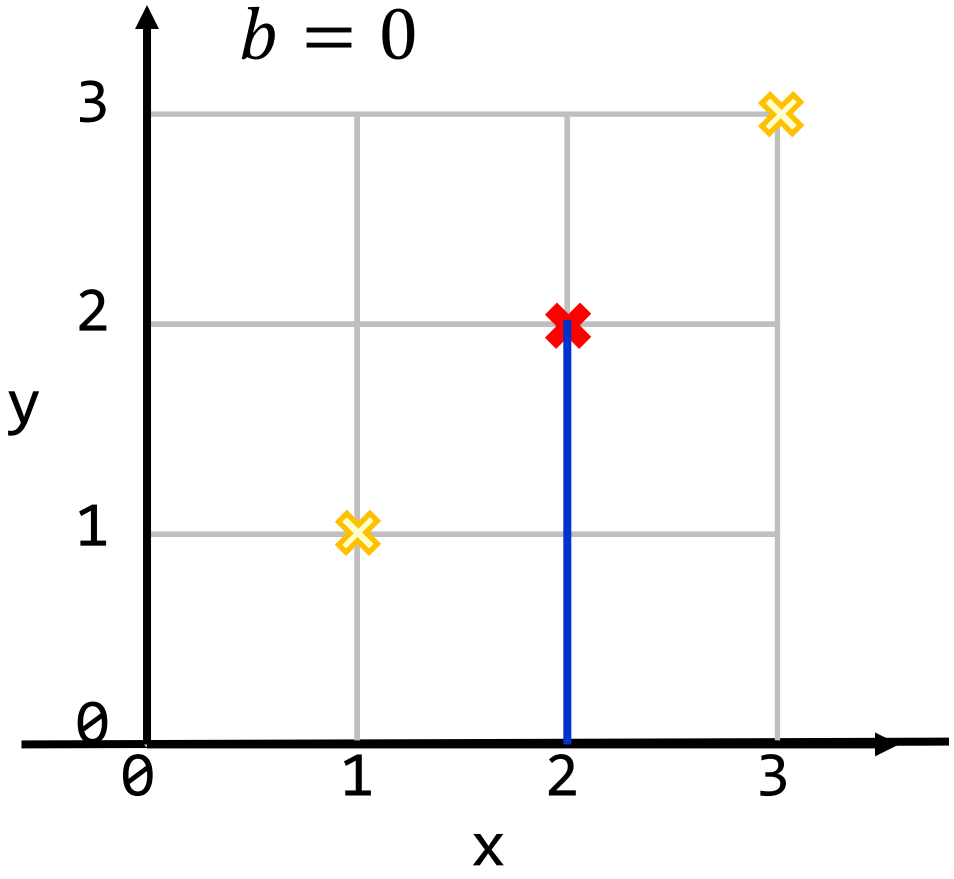


Cost function

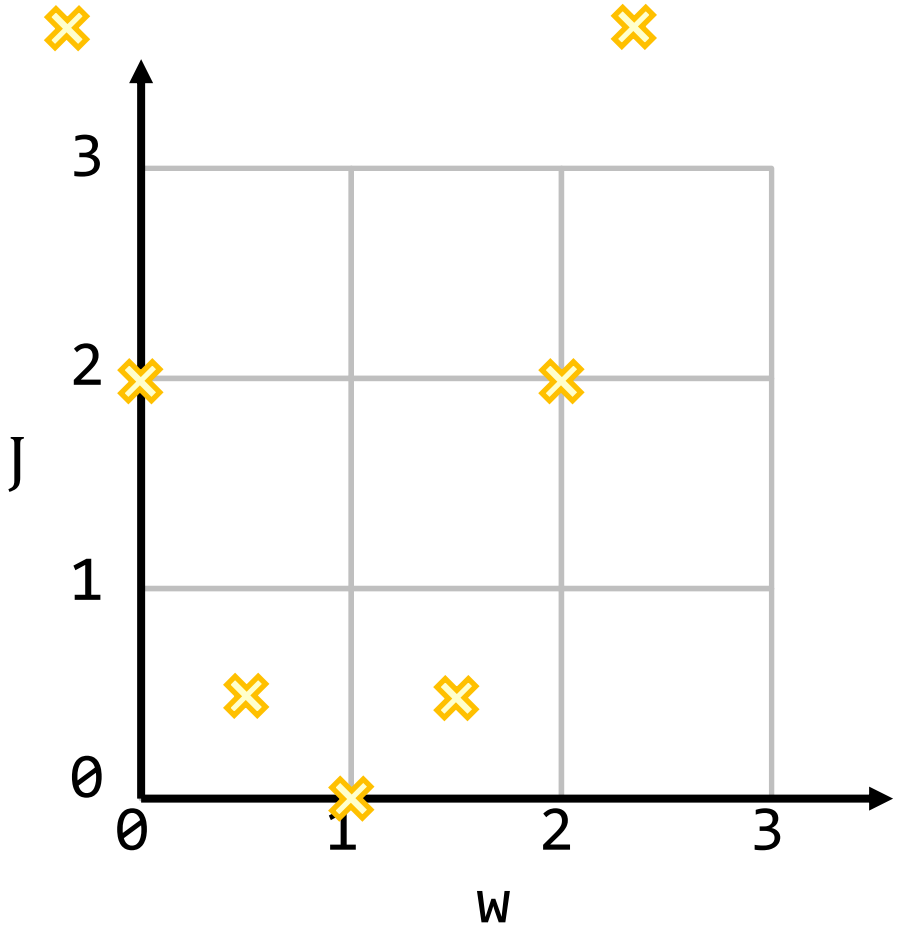
$$\hat{y} = wx + b$$

$$w = 0$$

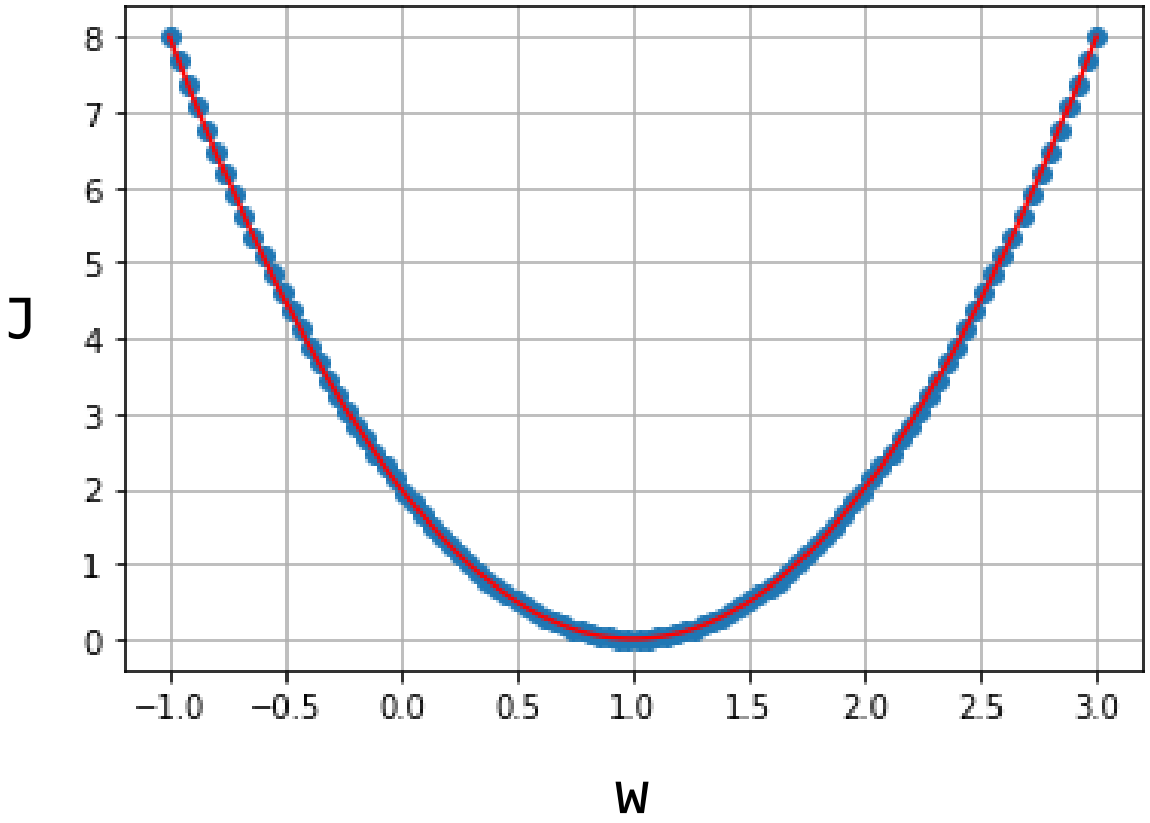
$$b = 0$$



$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

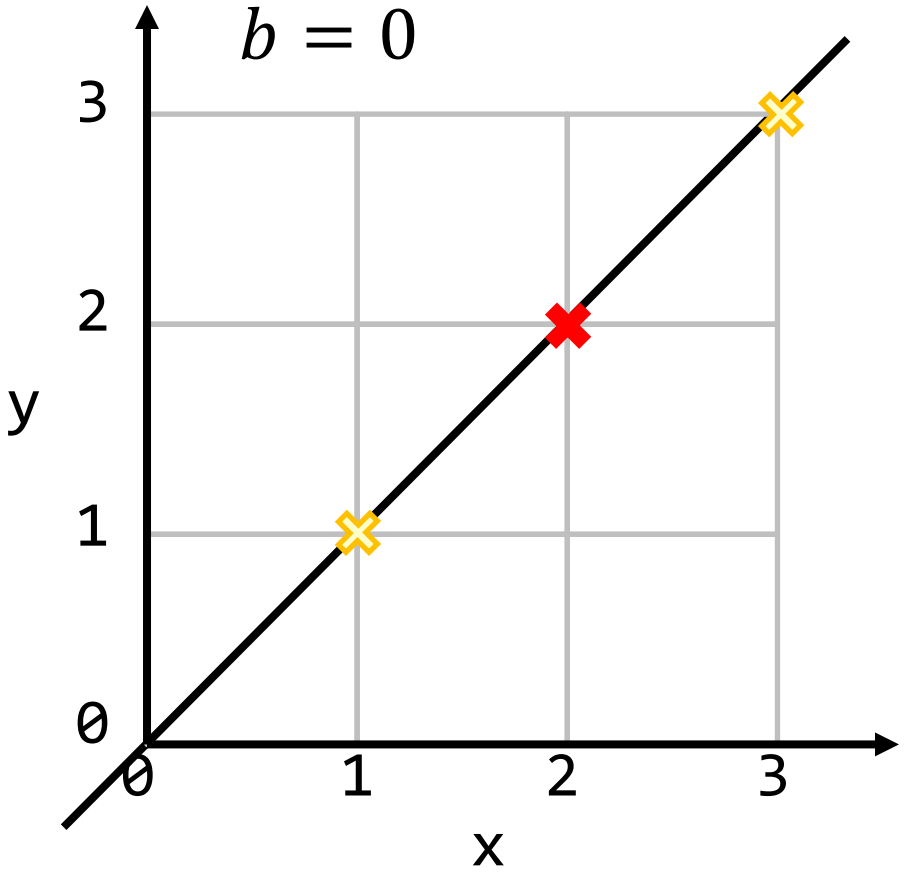


Cost function

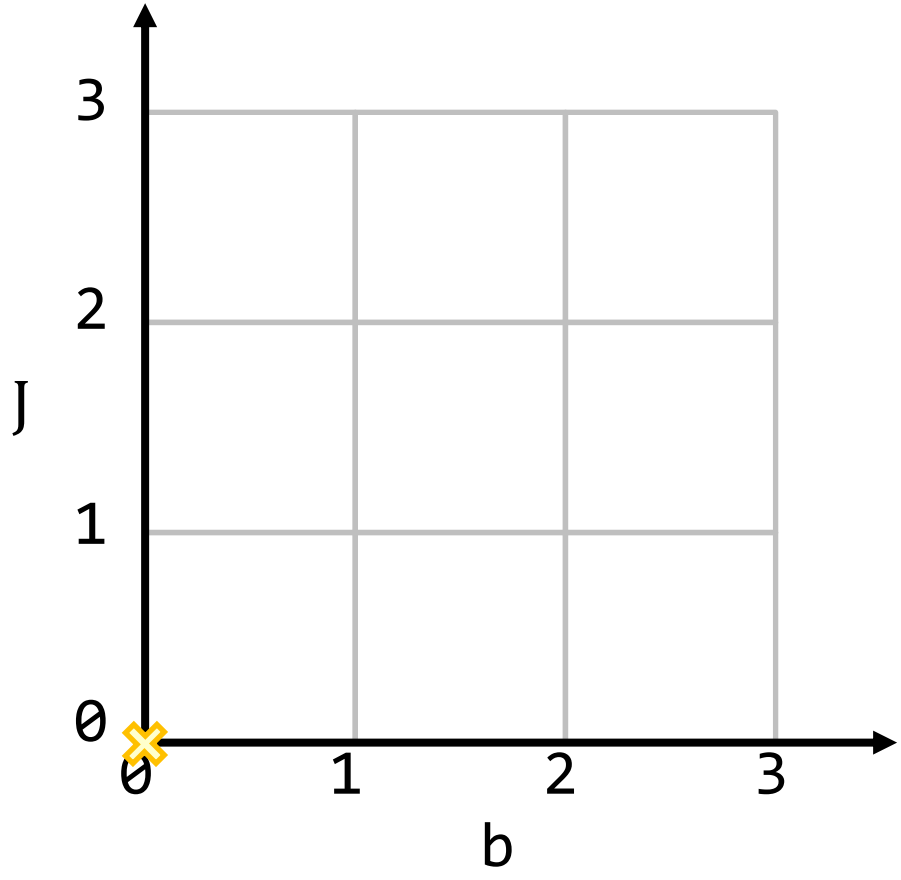


Cost function

$$\hat{y} = wx + b$$
$$w = 1$$
$$b = 0$$



$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

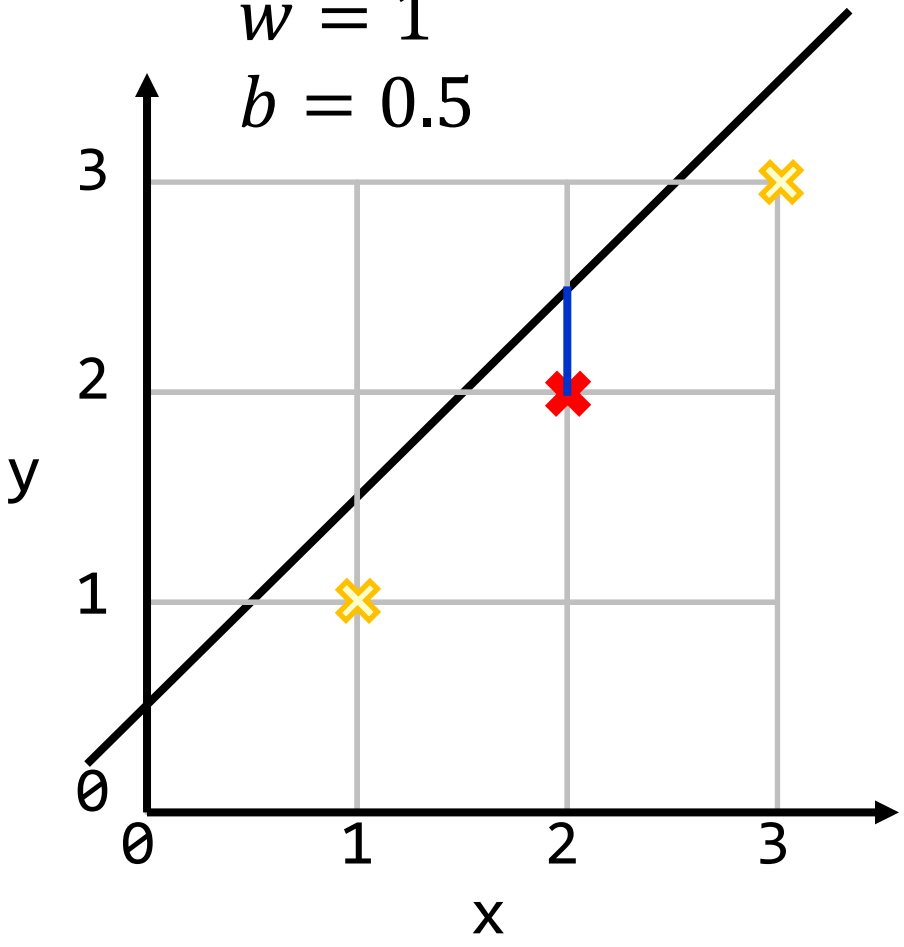


Cost function

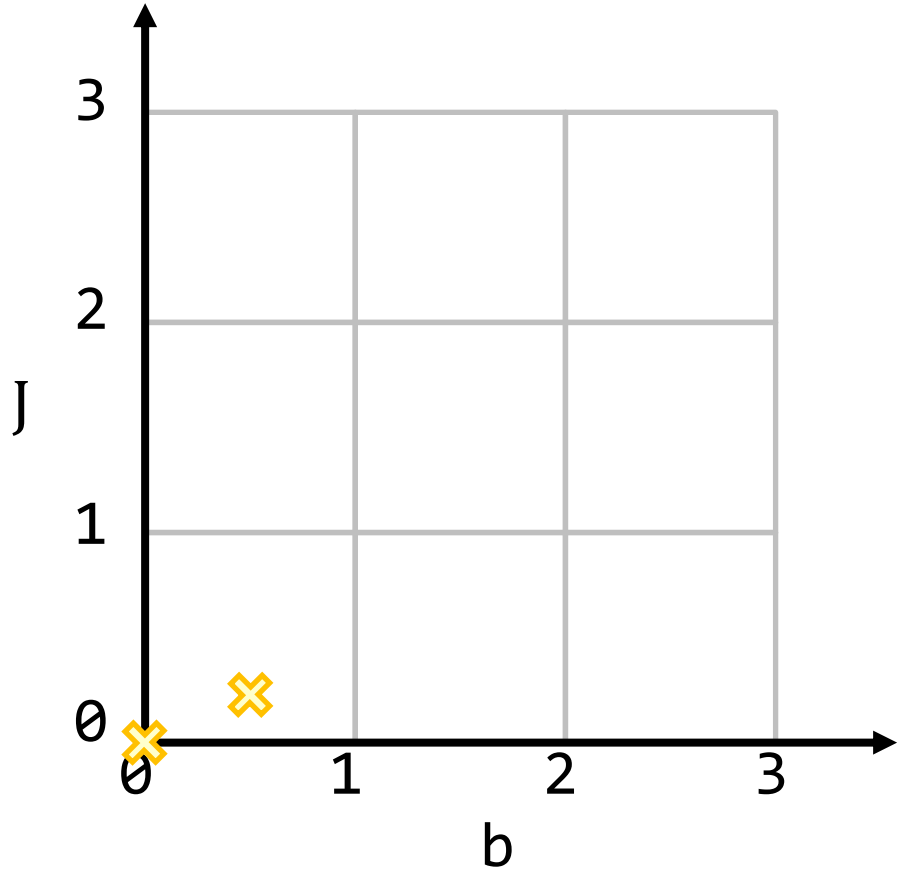
$$\hat{y} = wx + b$$

$$w = 1$$

$$b = 0.5$$



$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

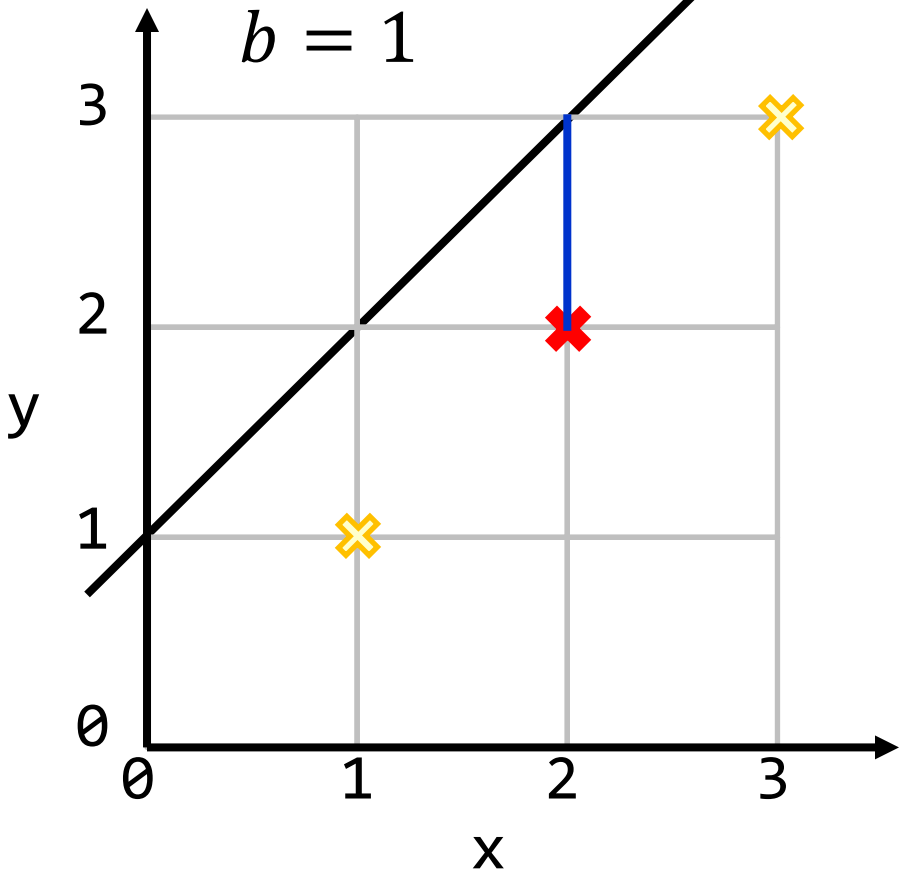


Cost function

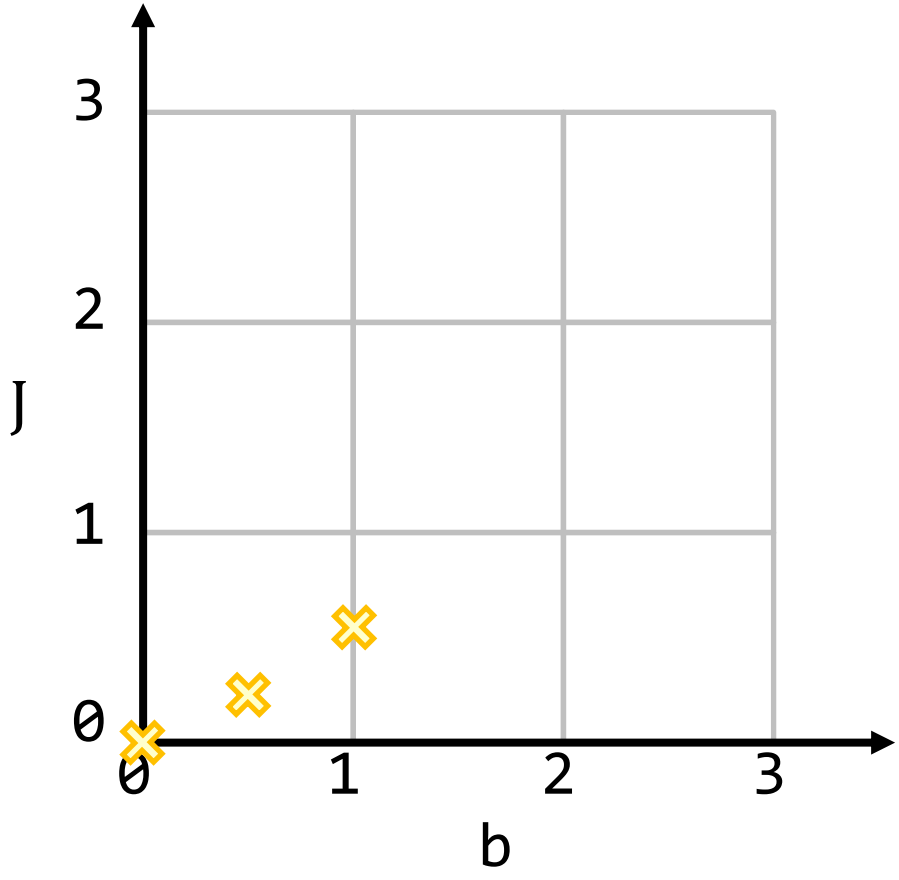
$\hat{y} = wx + b$

$w = 1$

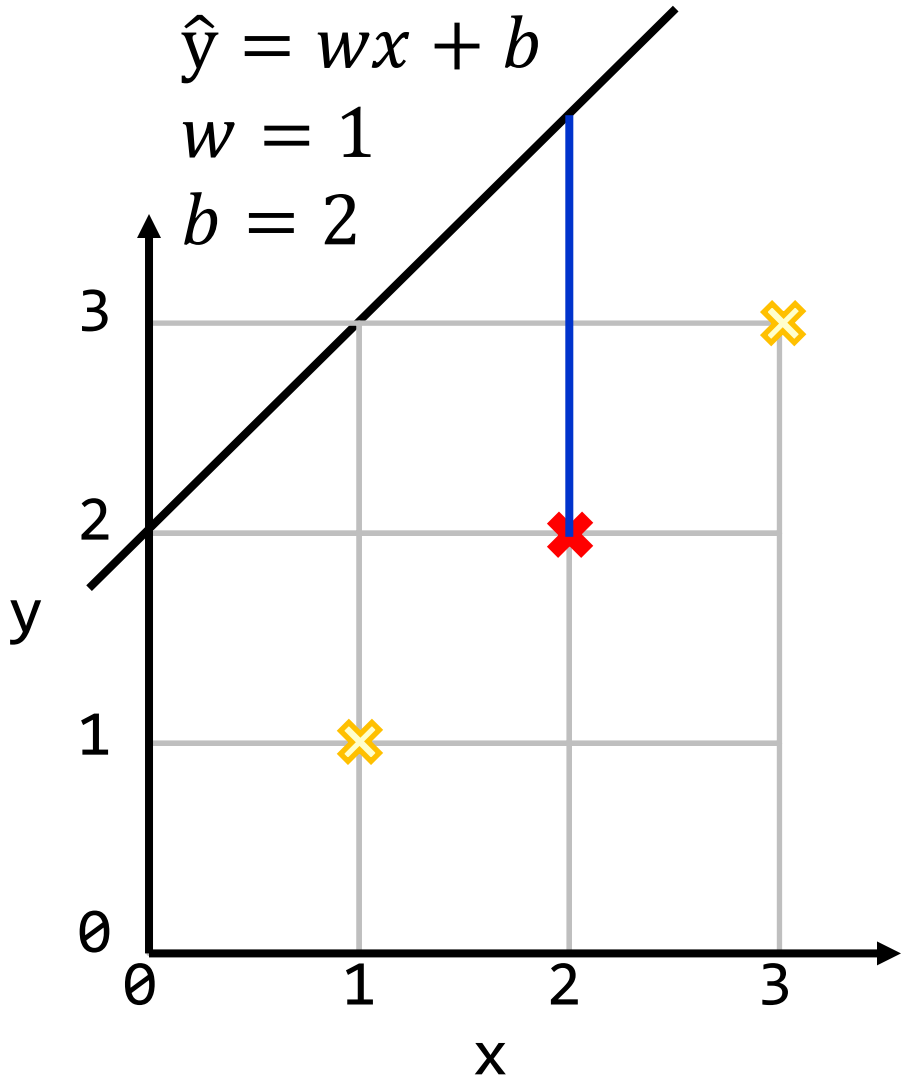
$b = 1$



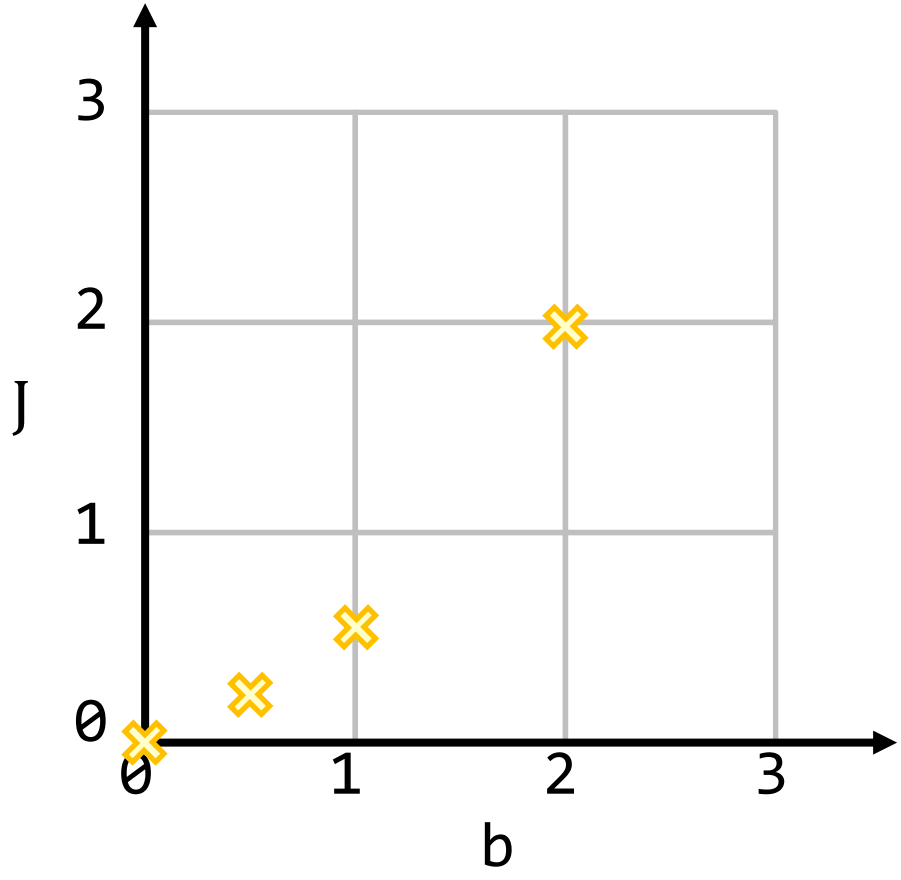
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



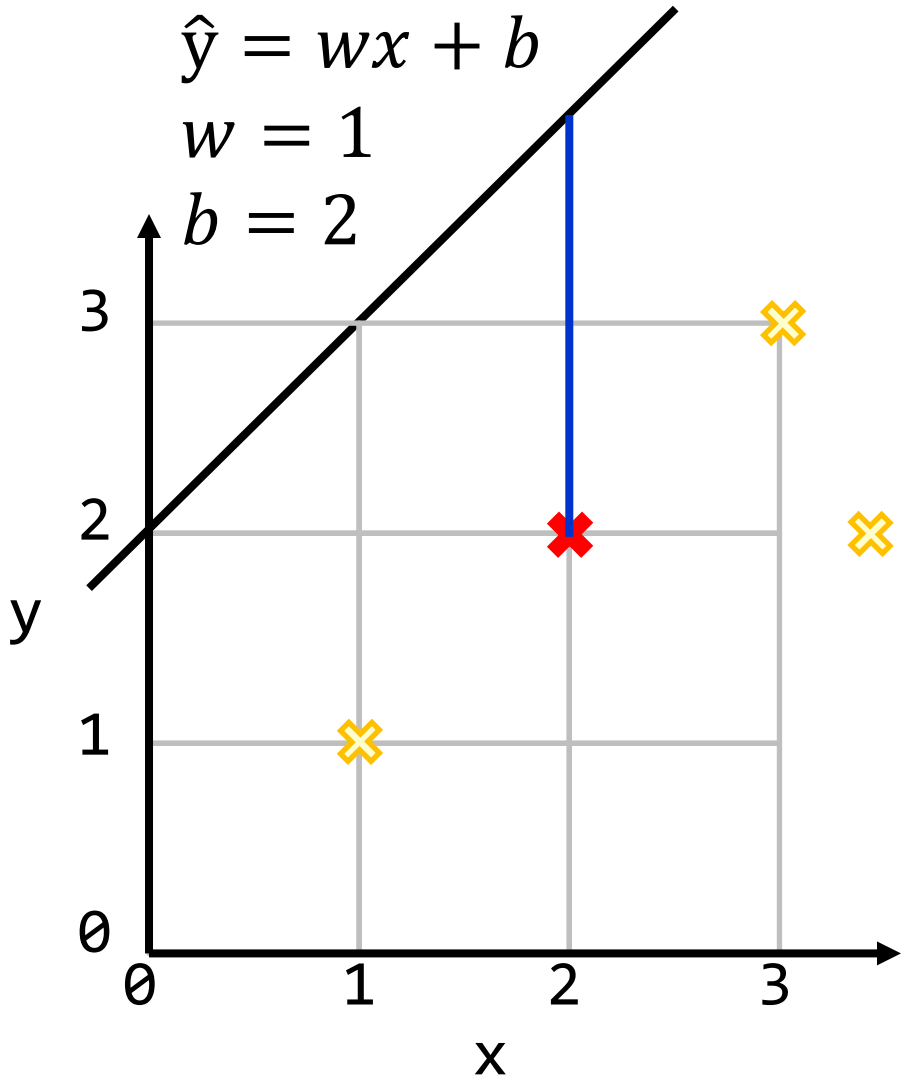
Cost function



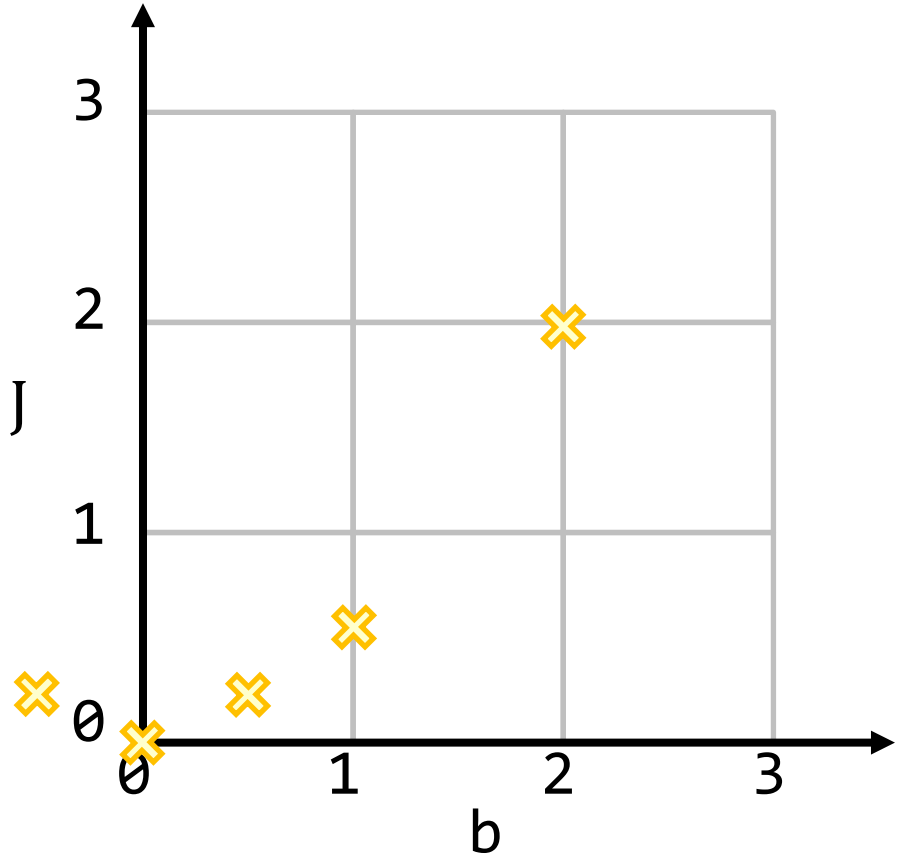
$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



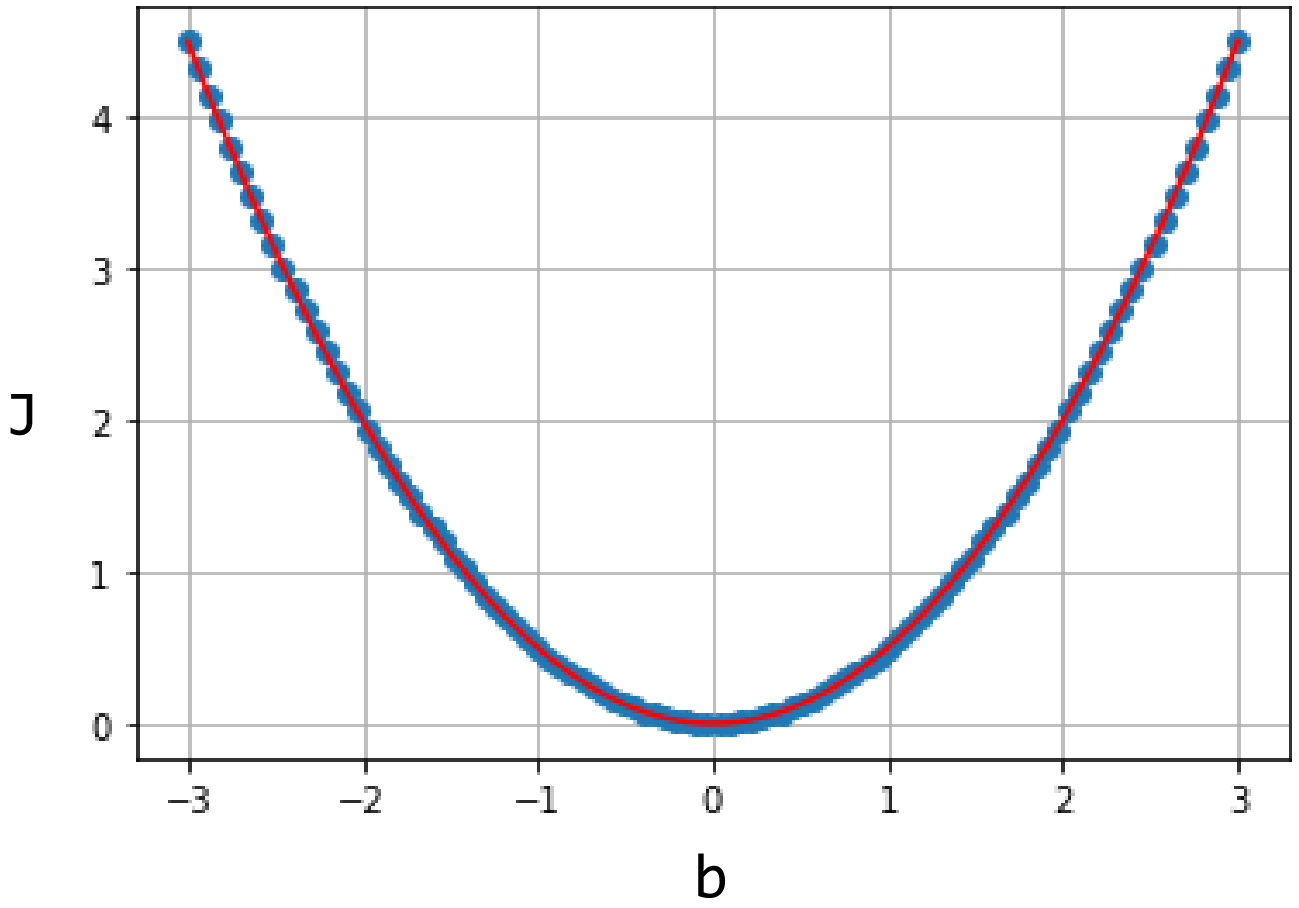
Cost function



$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$



Cost function



Gradient descent algorithm

- 비용 함수 최소화
- 경사 하강은 많은 최소화 문제에 사용된다.
- 주어진 비용 함수, 비용 (W, b)에 대해 비용을 최소화하기 위해 W, b를 찾는다.
- 일반적인 함수 : 비용 (w1, w2,...)에 적용 가능

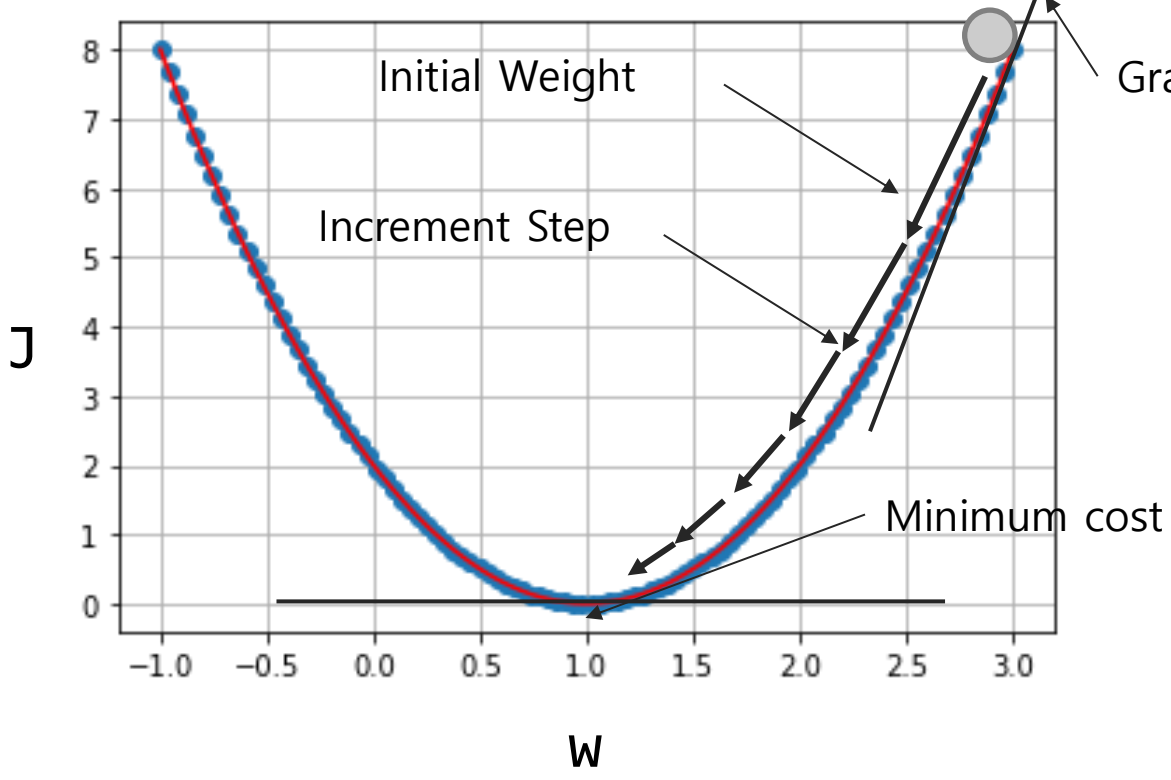
작동 방식

- 초기 추측으로 시작
  - 0,0 (또는 다른 값)에서 시작
  - W와 b를 약간 변경하여 cost(W, b)의 비용을 줄이려고 노력
- 매개 변수를 변경할 때마다 가능한 가장 낮은 cost(W, b)을 감소시키는 기울기를 선택
- 반복
- 최소한의 지역으로 수렴 할 때까지 수행

Gradient descent algorithm

$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$



$$w = w - \text{기울기}$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

$$x = 2$$
$$y = 2$$

Gradient descent algorithm

수렴까지 반복

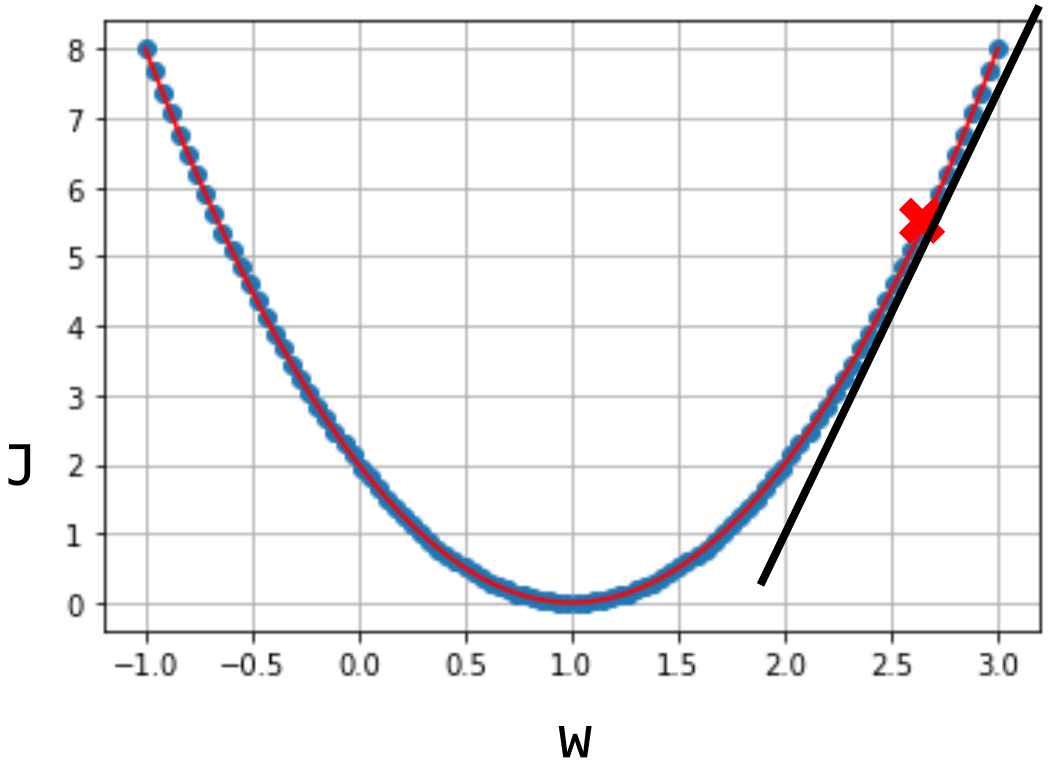
{

$$w = w - \alpha \frac{\partial}{\partial w} J(w, b)$$

}

learning rate

derivative



$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$

$\frac{\partial}{\partial w} J(w, b)$는 $\frac{1}{2}(\hat{y} - y)^2$ 을 w에 대해서 편미분 하면 된다.

Gradient descent algorithm

$$= \frac{\partial}{\partial w}\frac{1}{2}((wx + b) - y)^2$$

$$= \frac{\partial}{\partial w}\frac{1}{2}((wx + b)^2 - 2(wx + b)y + y^2)$$

$$= \frac{\partial}{\partial w}\frac{1}{2}((wx + b)^2 - 2ywx - 2by + y^2)$$

$$= \frac{\partial}{\partial w}\frac{1}{2}(w^2x^2 + 2wxb + b^2 - 2ywx - 2by + y^2)$$

$$= \frac{1}{2}(2wx^2 + 2xb - 2yx)$$

$$= x(wx + b - y)$$

$$= x(\hat{y} - y)$$

$$\hat{y} = wx + b$$

$$J(w, b) = \frac{1}{2}(\hat{y} - y)^2$$

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

$$\frac{\partial}{\partial w} (wx + b) = x$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = x(\hat{y} - y)$$

Chain Rule 사용

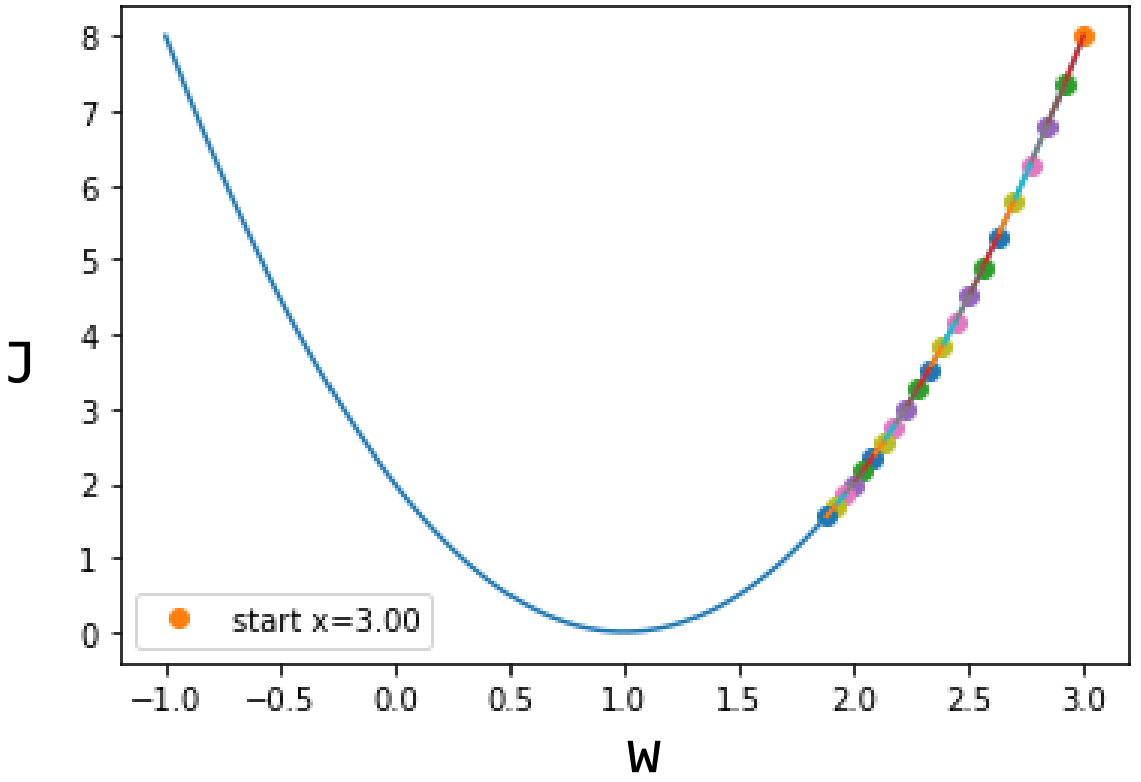
$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2} (\hat{y} - y)^2$$

$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

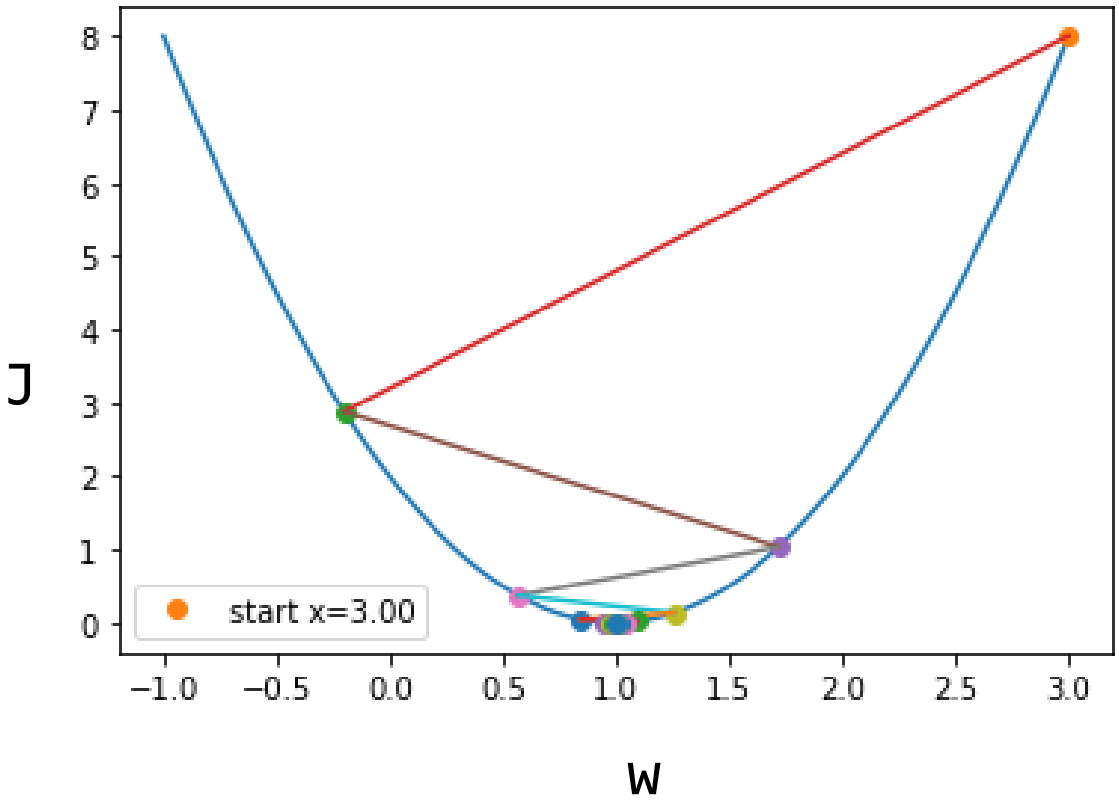
수렴까지 반복
{
$w = w - \alpha * (\hat{y} - y) * x$
}

α 가 0.01인 경우



수렴까지 반복
{
$w = w - \alpha * (\hat{y} - y) * x$
}

α 가 **0**.4인 경우

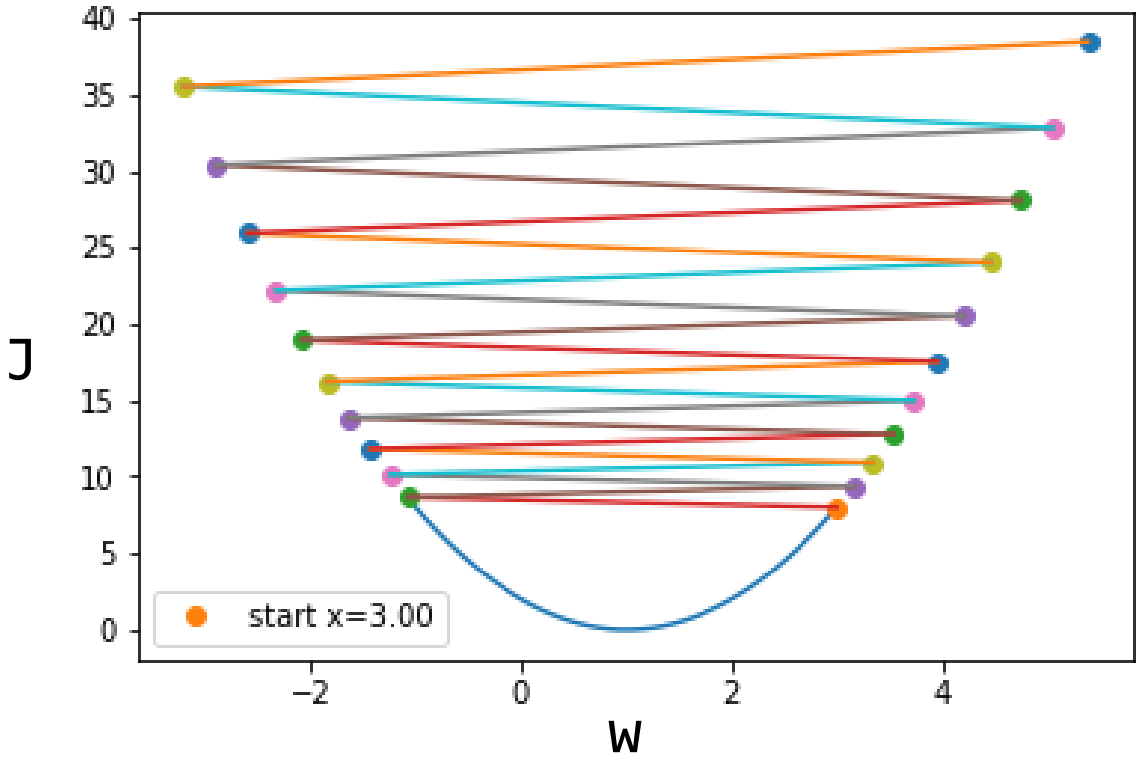


수렴까지 반복
{

$$w = w - \alpha * (\hat{y} - y) * x$$

}

α 가 0.51인 경우



Gradient descent algorithm

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

$$\frac{\partial \hat{y}}{\partial w} = wx + b = 1$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = (\hat{y} - y)$$

Chain Rule 사용

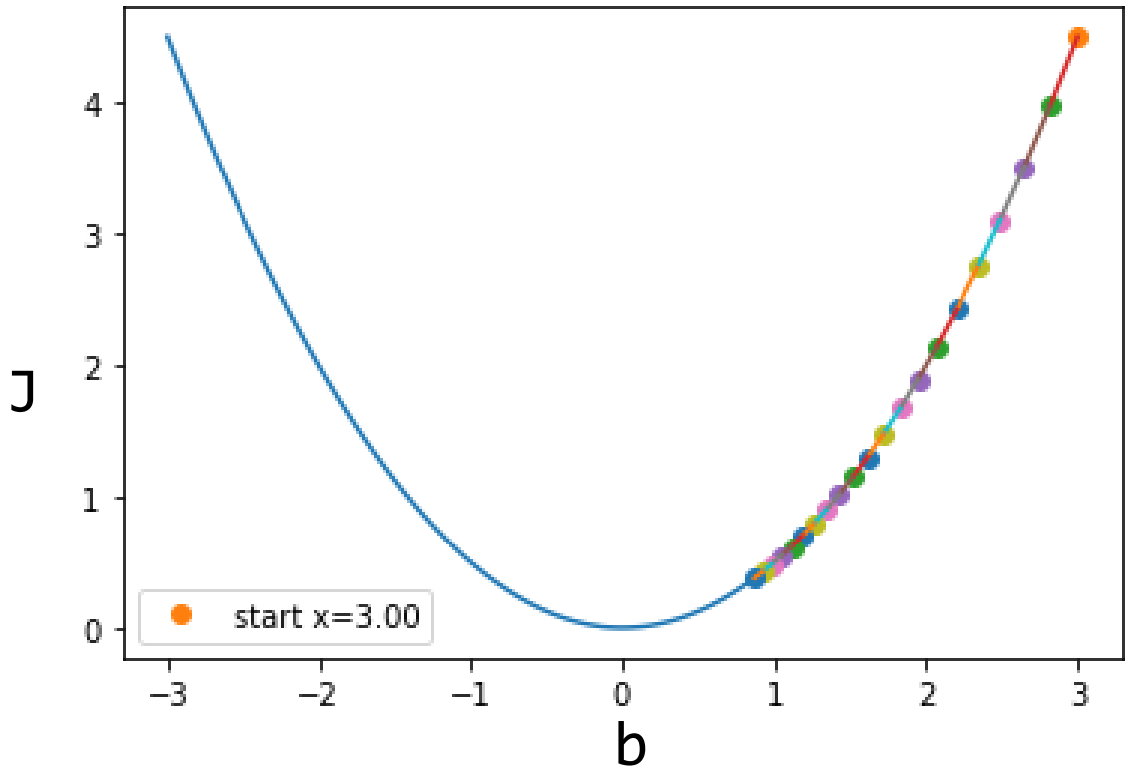
$$\hat{y} = wx + b$$

$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$

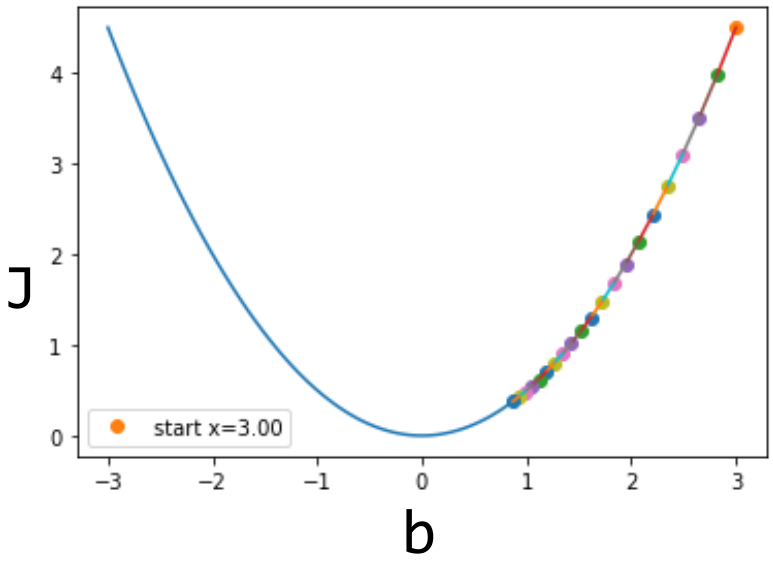
$$\frac{\partial}{\partial b} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial b}$$

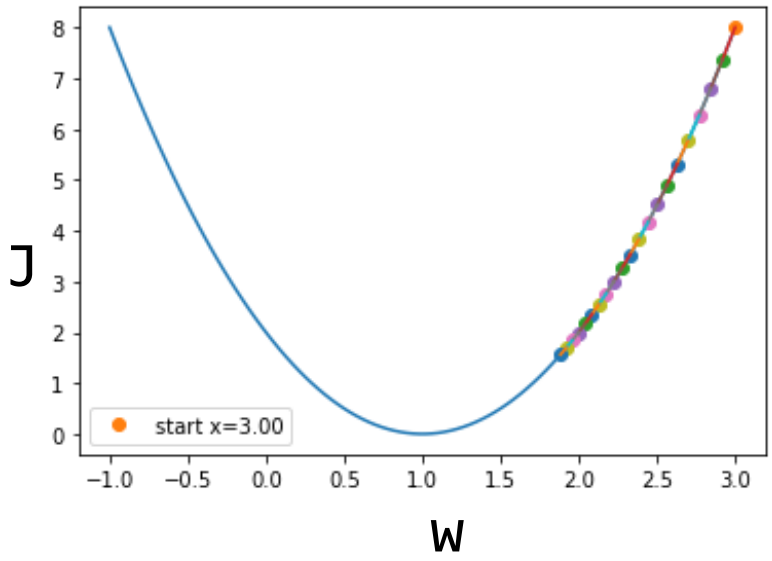
수렴까지 반복
{
$b = b - \alpha * (\hat{y} - y)$
}

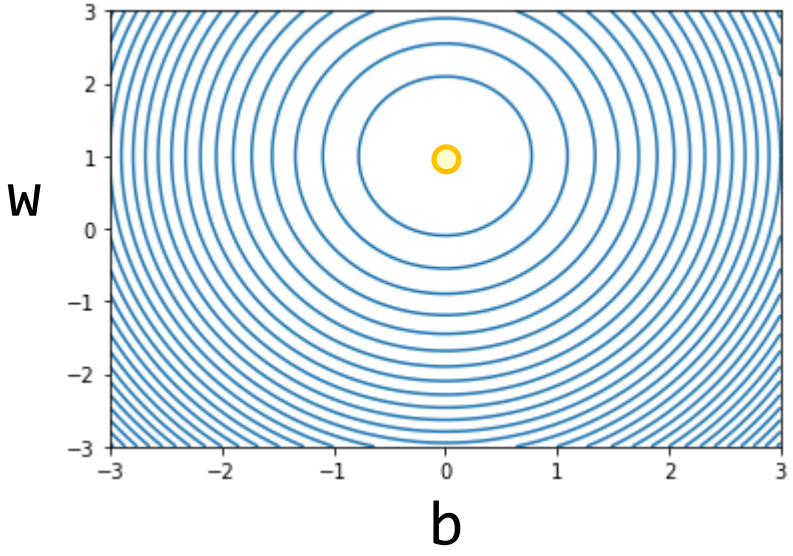
α 가 **0.03**인 경우

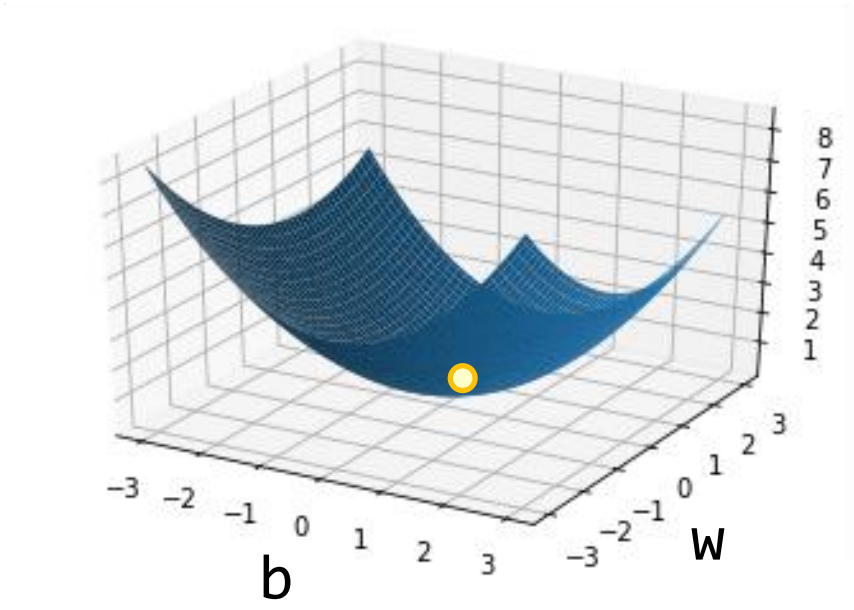


Gradient descent algorithm

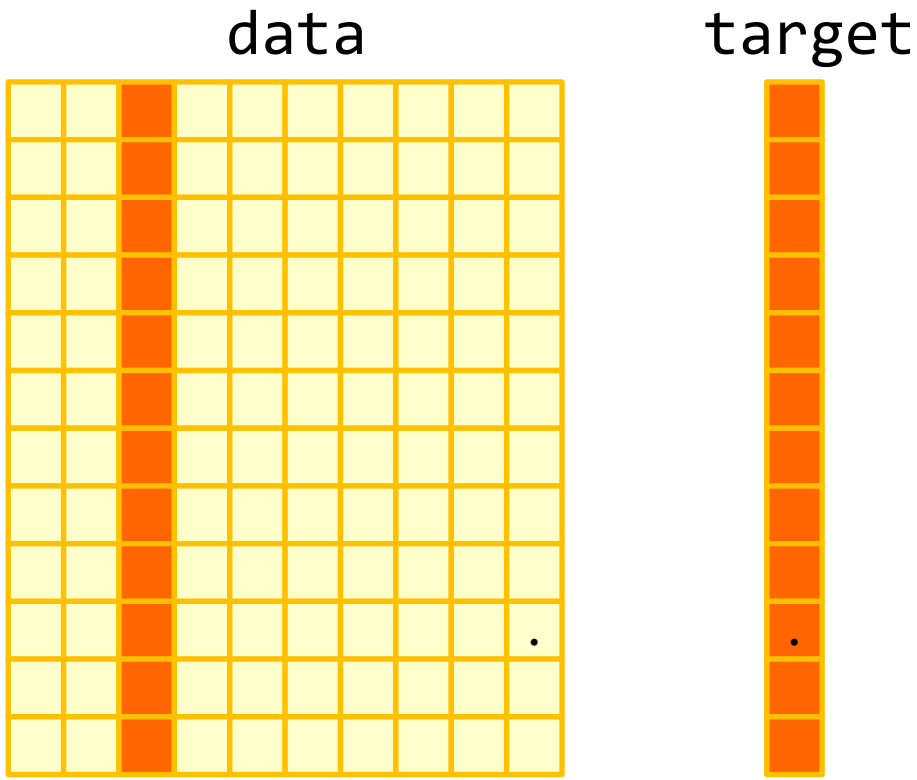






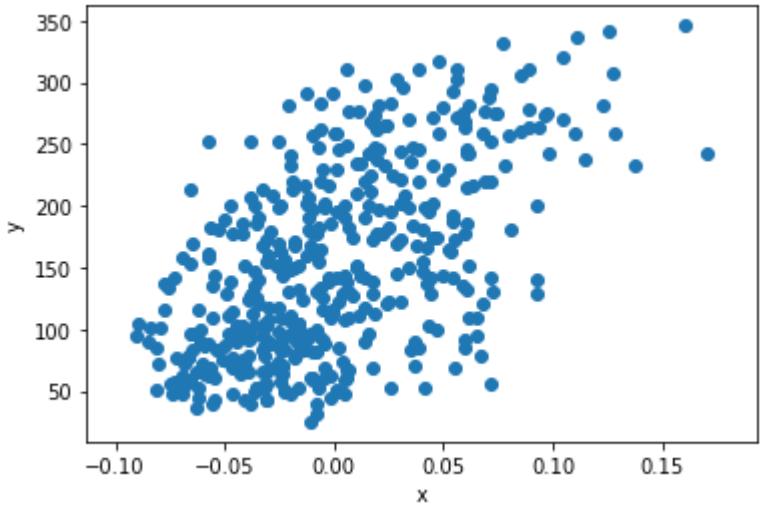


data set 준비(sklearn 이용)



data set 준비(sklearn 이용)

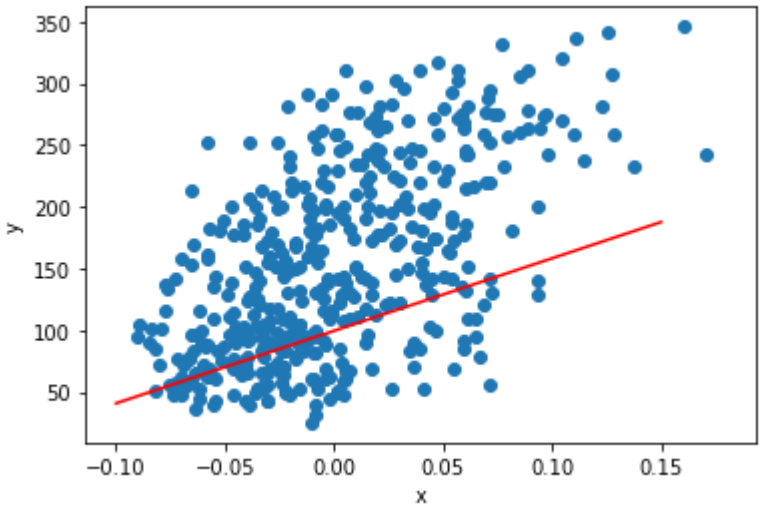
```
from sklearn.datasets import load_diabetes
diabetes = load_diabetes()
import matplotlib.pyplot as plt

x = diabetes.data[:, 2]
y = diabetes.target

plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



역전파 구현

1. batch=1
2. 점의 전체 개수만큼 1번 순회 (epoch=1)
   : SGD=>원소 하나마다 w,b 갱신

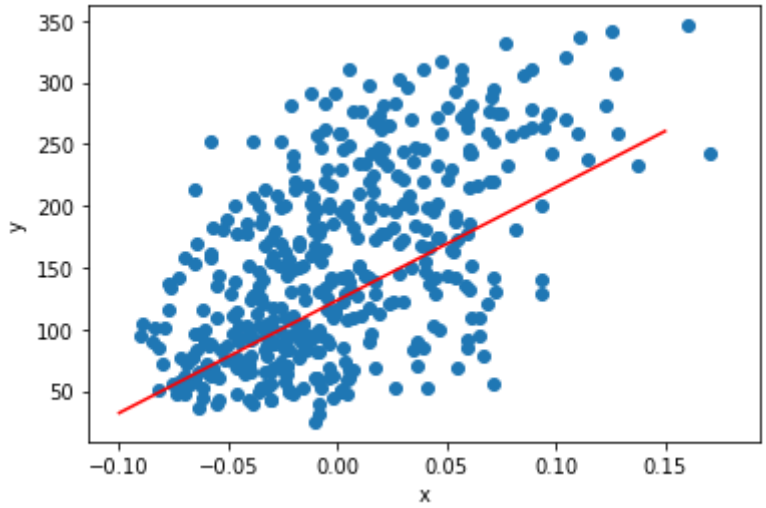
```
w = 1.0
b = 1.0
rate = 0.01
for x_i, y_i in zip(x, y):
    y_hat = x_i * w + b
    err = y_hat - y_i
    w = w - rate * err * x_i
    b = b - rate * err

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]], 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



역전파 구현

1. batch=1
2. epoch=100번 만큼 외부 루프에서 순회
3. 점의 전체 개수만큼 1번 순회
   : SGD=>원소 하나마다 w,b 갱신

```
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
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]], 'r-')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```



Logistic Regression

- 로지스틱 회귀 란?
  - Classification(분류)
  - Logistic vs Linear

- 동작 방식
  - 가설 표현
  - Sigmoid 함수
  - Decision Boundary(결정경계)
  - Cost Function
  - Optimizer (Gradient Descent)

Classification

Binary Classification(이진 분류) 란?

: 값은 0 또는 1

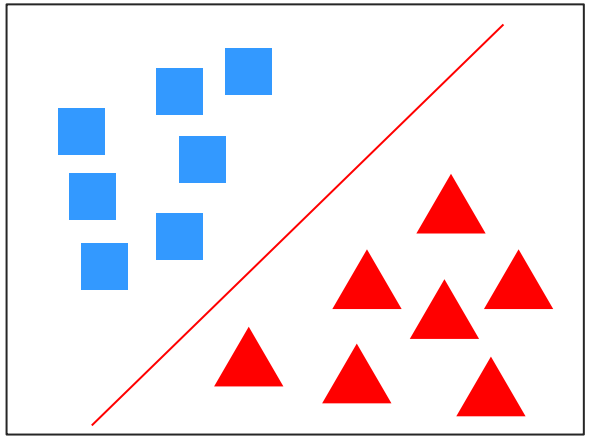
- Exam : Pass or Fail
- Spam : Not Spam or Spam
- Face : Real or Fake
- Tumor : Not Malignant or Malignant

머신 러닝으로 시작하려면 변수를 인코딩해야 한다.

```
x_train = [[1,2], [2,3], [3,1], [4,3], [5,3], [6,2]]
y_train = [[0], [0], [0], [1], [1], [1]]
```

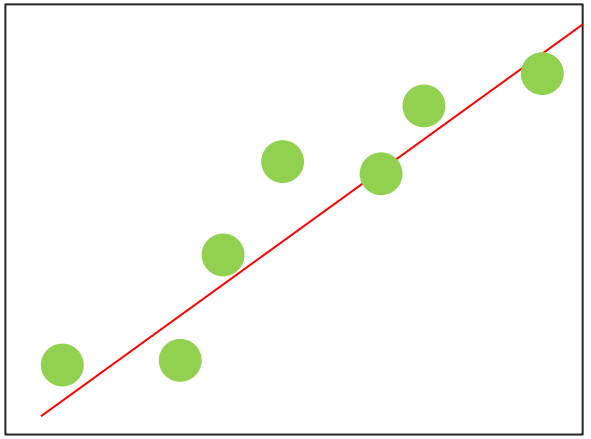
Logistic vs Linear

로지스틱 회귀와 선형 회귀의 차이점은?



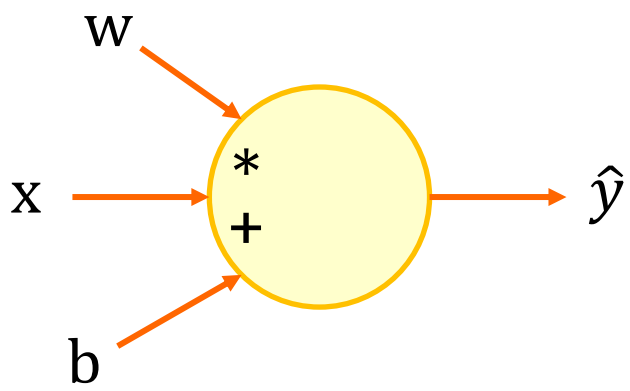
Discrete(분리) : 분류 목적

신발 사이즈 / 회사의 근로자수

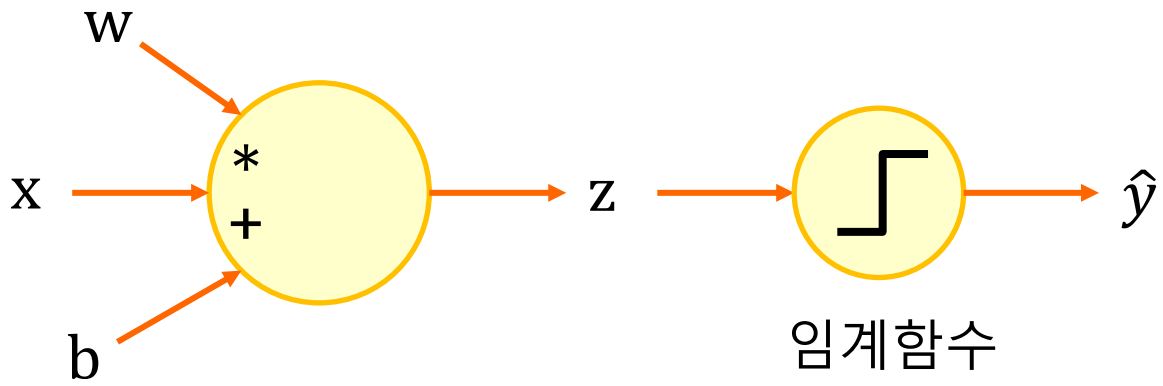


Continuous (지속적인) : 값의 예측

시간/ 무게 / 높이

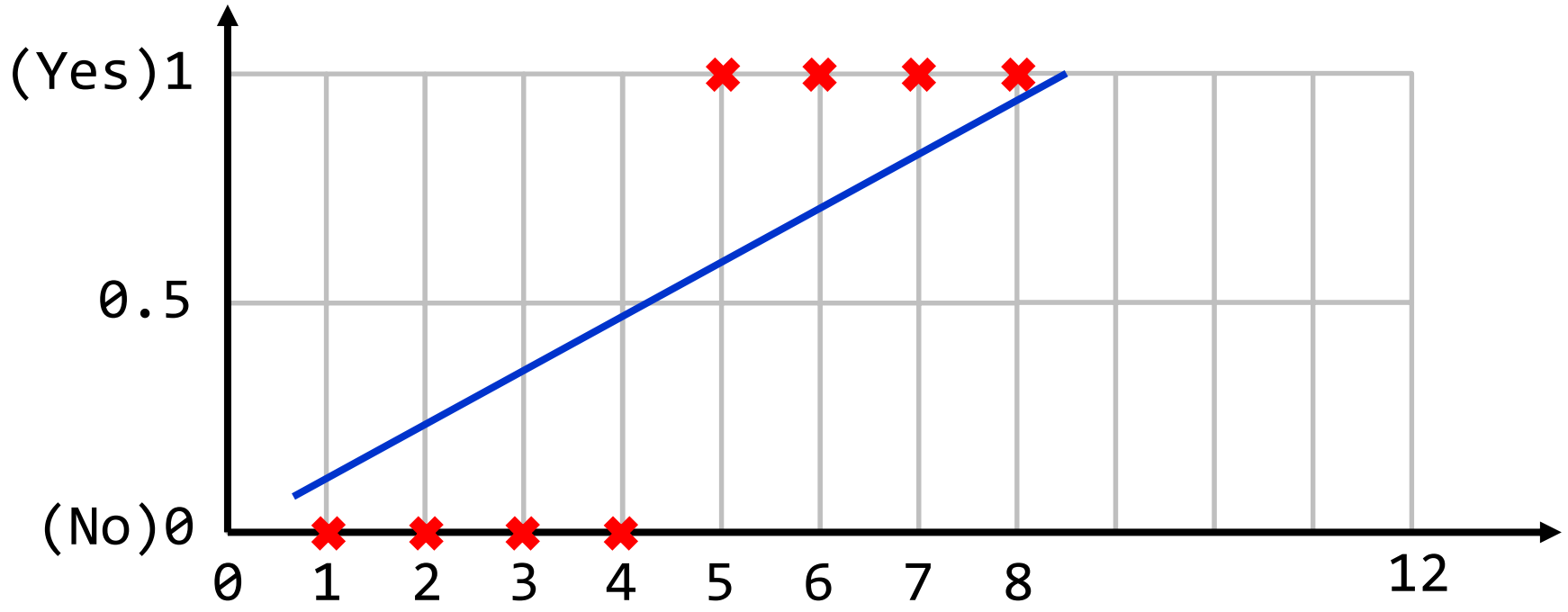


퍼셉트론



임계함수

Sigmoid 함수 사용 이유

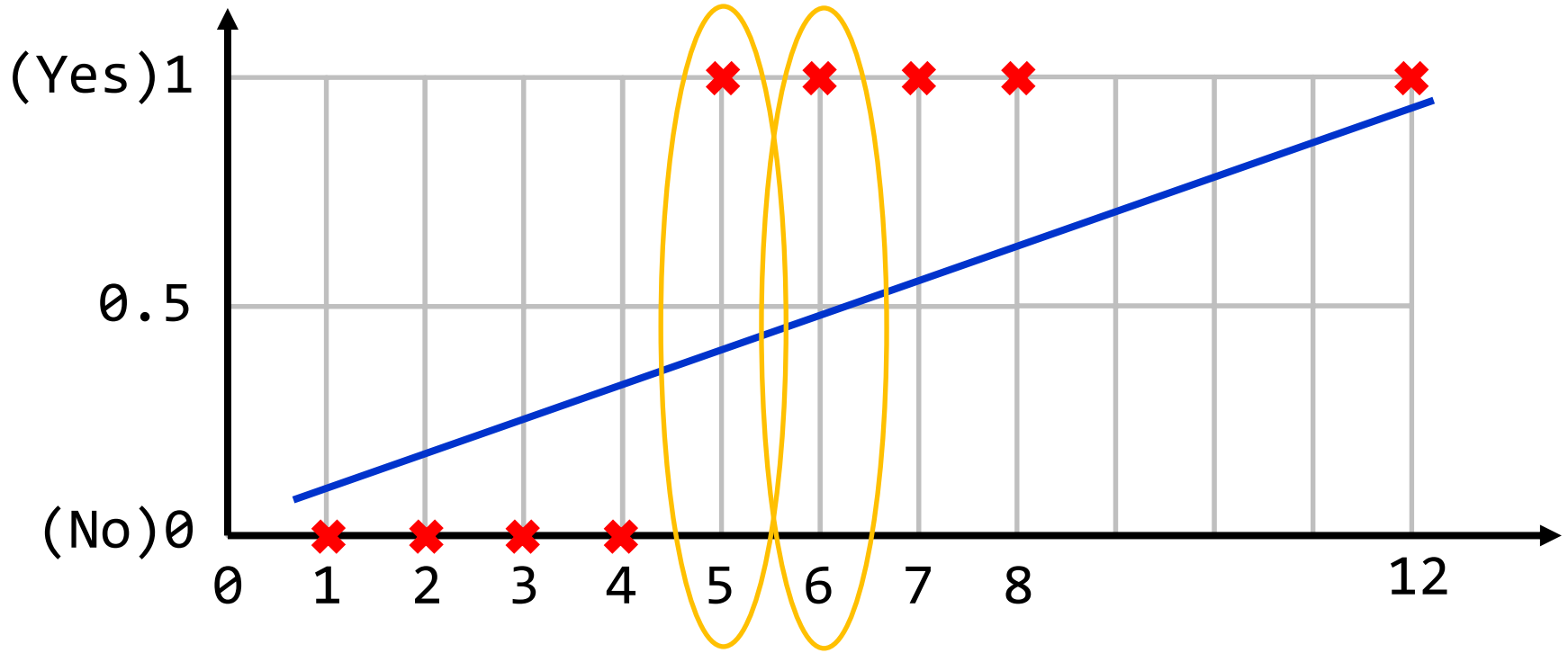


Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"
If $\hat{y} < 0.5$, predict "y=0"

1 => 양성 5 => 악성
2 => 양성 6 => 악성
3 => 양성 7 => 악성
4 => 양성 8 => 악성

Sigmoid 함수 사용 이유

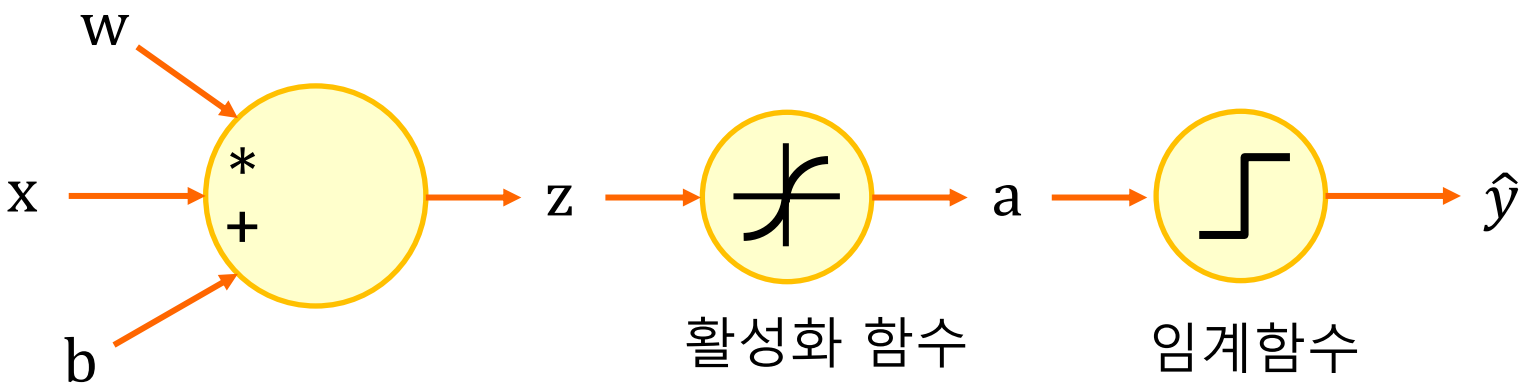


Threshold classifier output at 0.5:

If $\hat{y} \geq 0.5$, predict "y=1"
If $\hat{y} < 0.5$, predict "y=0"

1 => 양성
2 => 양성
3 => 양성
4 => 양성
5 => 양성
6 => 양성
7 => 악성
8 => 악성

Sigmoid 함수 사용 이유



Sigmoid 함수 사용 이유

Classification: y=0 or 1

$\hat{y}$ can be > 1 or < 0

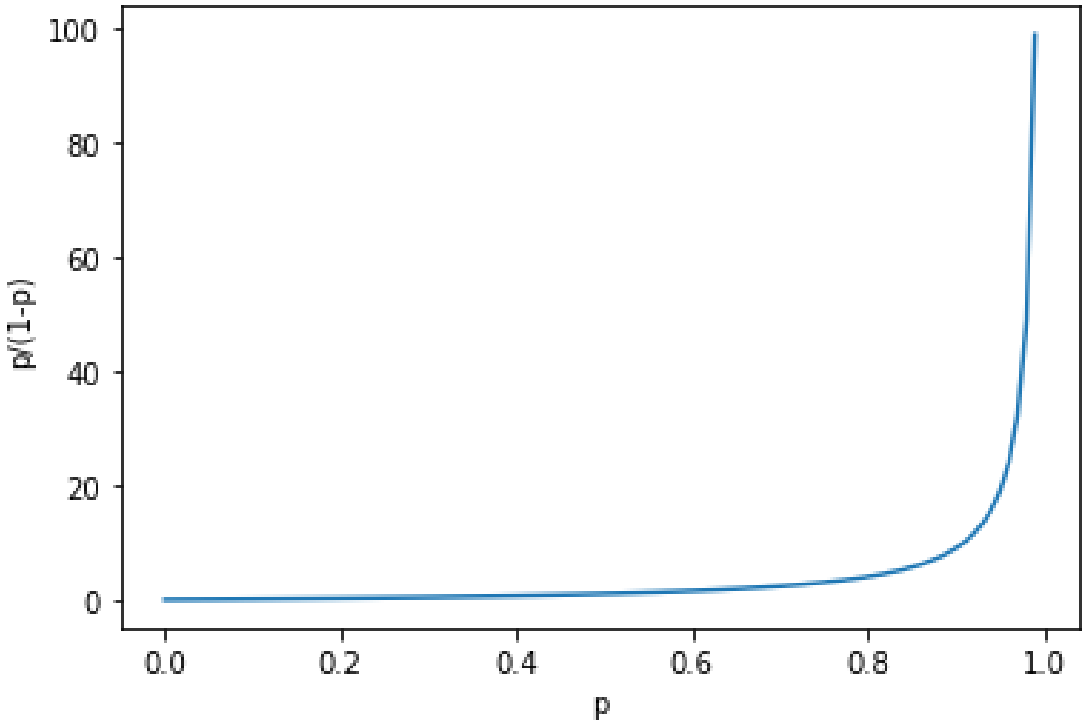
Logistic Regression: $0 < \hat{y} < 1$

분류 문제 사용

오즈 비(odds ratio)

$$\text{OR(odds ratio)} = \frac{p}{1-p} \quad \text{(p=성공확률)}$$

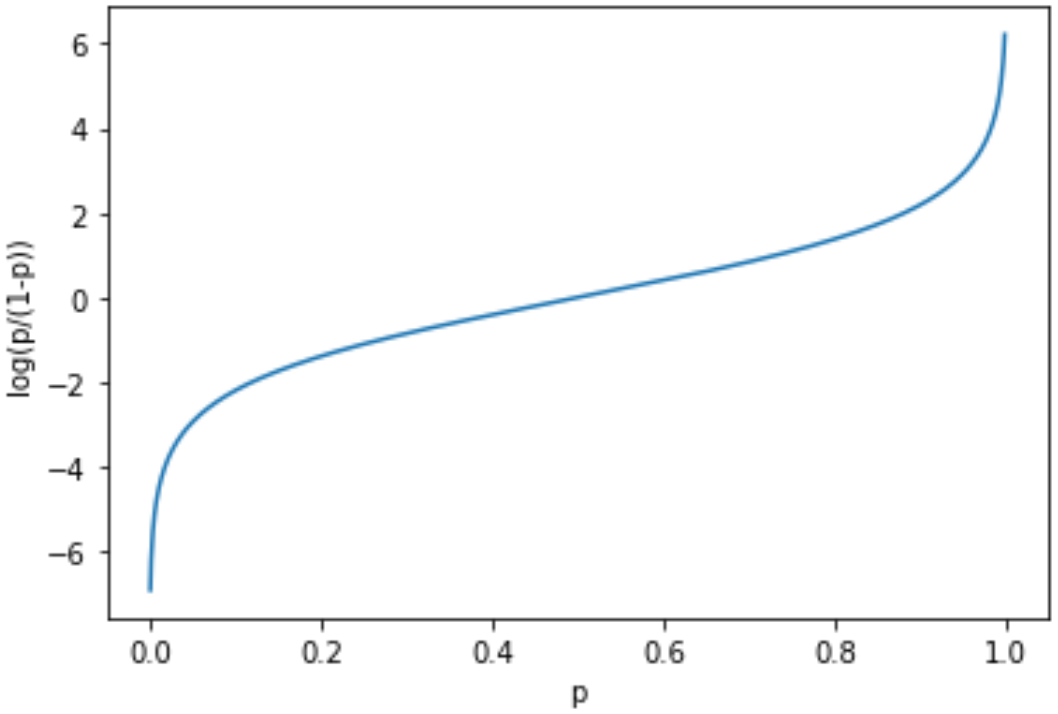
오즈 비를 그래프로 그리면 다음과 같다.
p가 0부터 1까지 증가할 때 오즈 비의 값은 처음에는
천천히 증가하지만 p가 1에 가까워지면 급격히 증가한다.



로짓 함수(Logit function)

$$\text{logit(p)} = \log(\frac{p}{1-p})$$ (p=성공확률)

로짓 함수는 p가 0.5일 때 0이 되고 p가 0과 1일 때 각각 무한대로 음수와 양수가 되는 특징을 가진다.



시그모이드 함수(Sigmoid function)

$$\text{logit(p)} = \log(\frac{p}{1-p}) = z$$

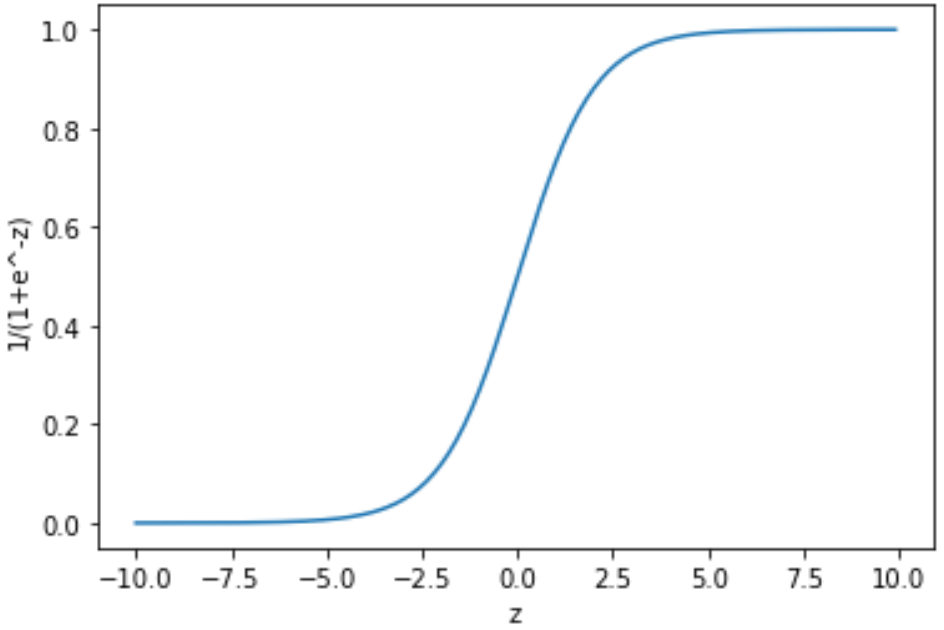
로지틱 함수의 유도 : p에대해 정리

$$\log(\frac{p}{1-p}) = z$$

$$e^{\log(\frac{p}{1-p})} = e^{z}$$

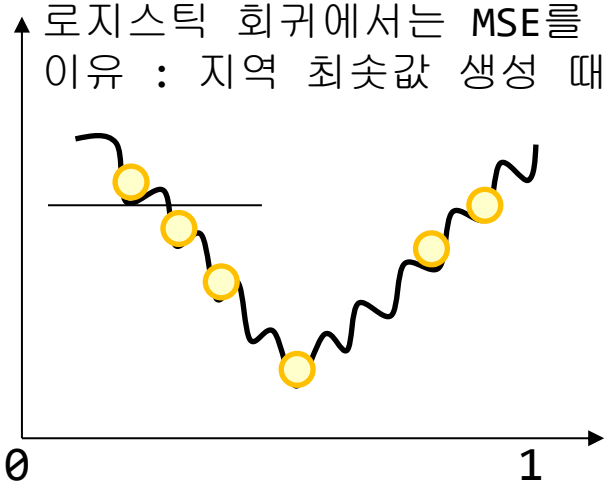
$$\frac{p}{1-p} = e^{z}$$

$$\text{p} = (1-p) * e^{z}$$
$$\text{p} = e^{z} - p * e^{z}$$
$$\text{p} + \text{p}* e^{z} = e^{z}$$
$$\text{p} = \frac{e^{z}}{1+e^{z}} = \frac{1}{1+e^{-z}}$$

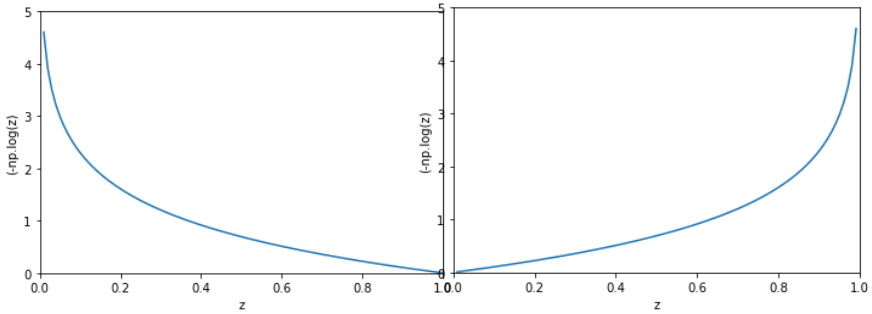


A convex logistic regression cost function

로지스틱 회귀에서는 MSE를 사용할 수 없다.
이유 : 지역 최솟값 생성 때문



Binary Cross Entropy 함수



$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$

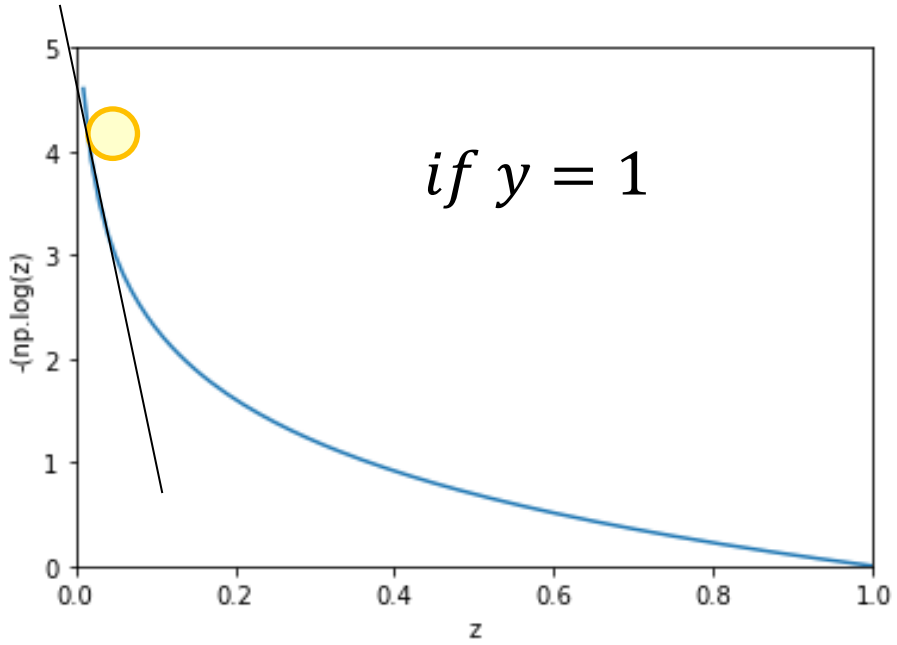
$$J(w) = \frac{1}{2}(sigmoid(\hat{y}) - y)^2$$

$$\text{cost}(\hat{y}, y) = -(\text{y} * \log(a) + (1-y)\log(1-\text{a}))$$

로지스틱 회귀 손실 함수

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$



$$\text{Cost} = 0 \ \text{ if } \ y = 1,\ \hat{y} = 1$$

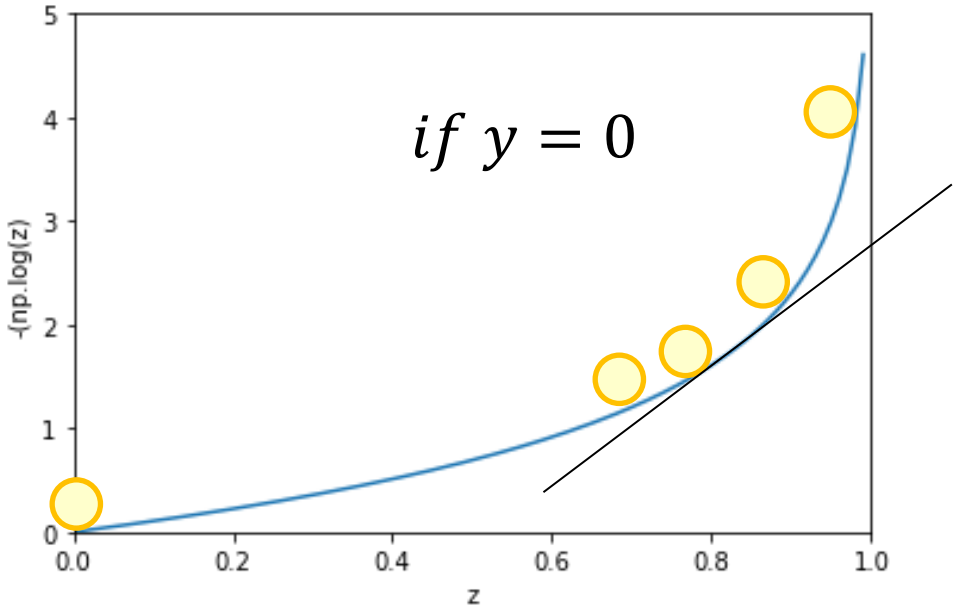
$$\text{But as} \quad \hat{y} \ \rightarrow \ 0$$

$$\text{Cost} \ \rightarrow \ \infty$$

로지스틱 회귀 손실 함수

Logistic regression cost function

$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$



$\text{Cost} = 0$ if $y = 0$, $\hat{y} = 0$

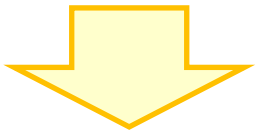
But as $\hat{y} \rightarrow 1$

$\text{Cost} \rightarrow \infty$

로지스틱 회귀 손실 함수

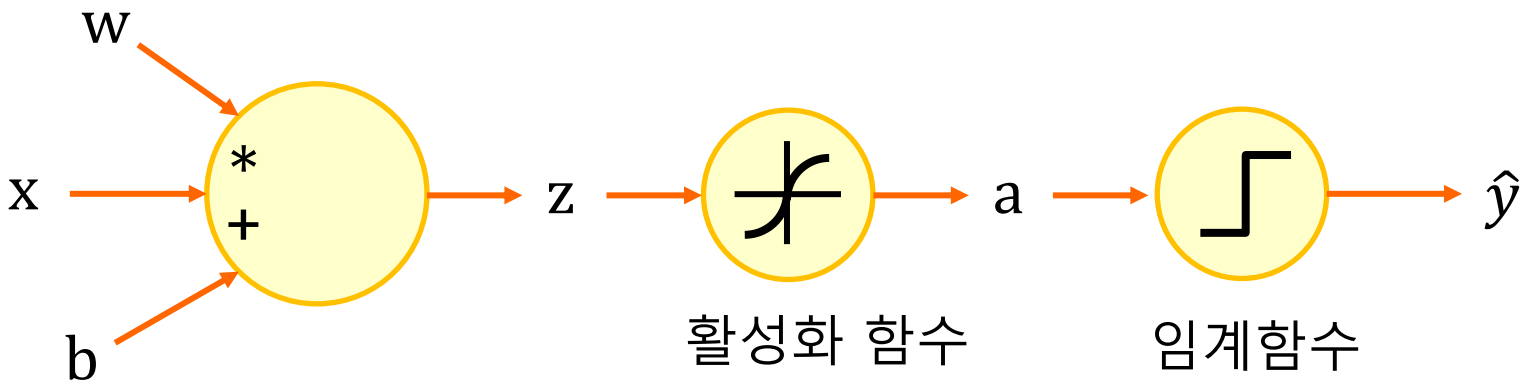
Logistic regression cost function

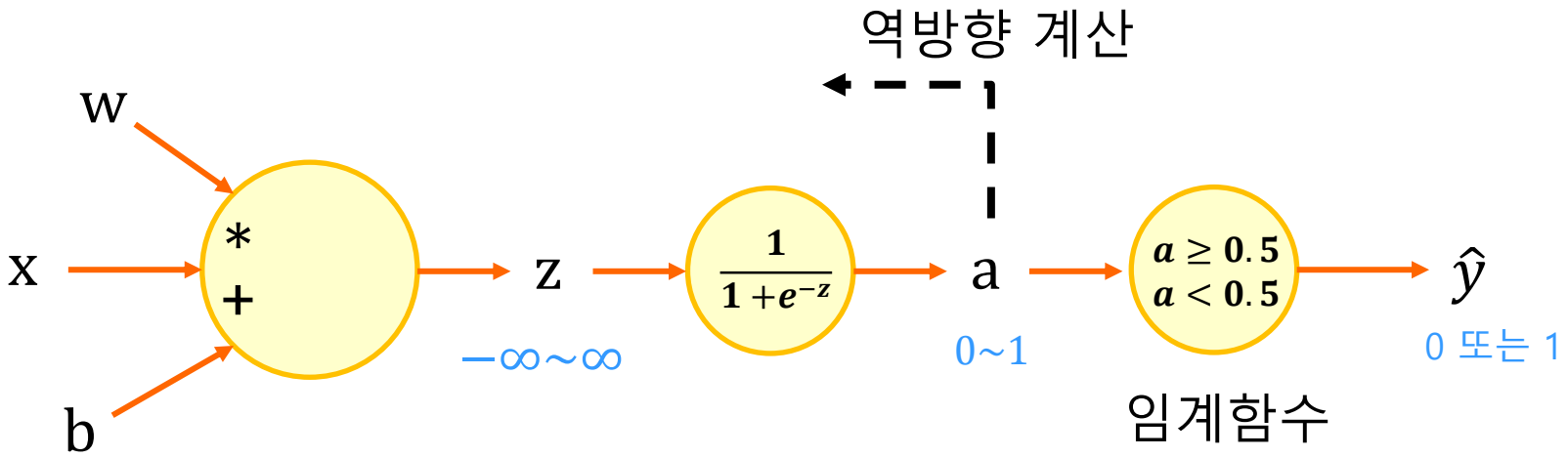
$$\text{Cost}(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1-\hat{y}) & if\ y = 0 \end{cases}$$



$$L = -(y * \log(a) + (1-y)\log(1-a))$$

로지스틱 회귀 손실 함수

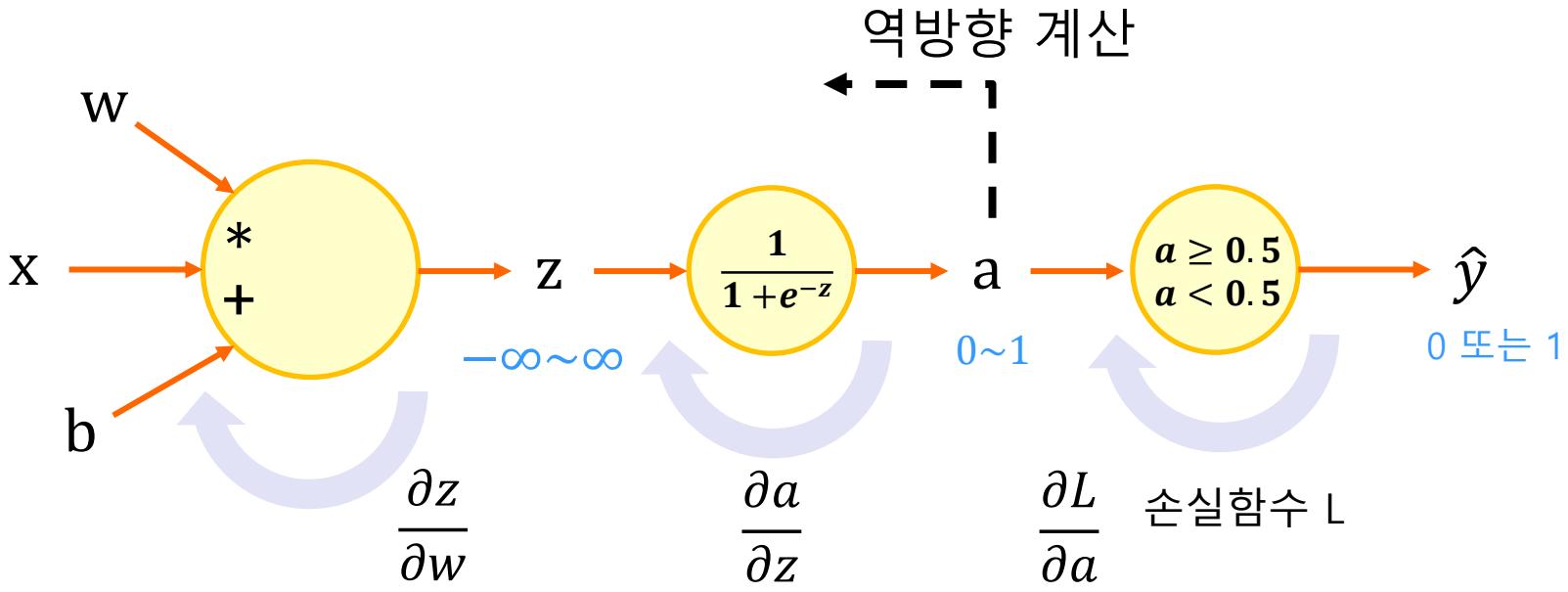

w
x
b
*
+
z
a
$\hat{y}$
활성화 함수
임계함수


역방향 계산
w
x
b
*
+
z
$\frac{1}{1+e^{-z}}$
a
$a \geq 0.5$
$a < 0.5$
$\hat{y}$
$-\infty \sim \infty$
0~1
0 또는 1
임계함수

로지스틱 회귀 손실 함수

특성이 하나인 경우 => x 1개

z = wx + b



chain rule을 이용하여 각 단계의 미분 결과를 곱한다.

w에 대하여 미분

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w} = (a-y)x$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial w} = x$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} = (a-y)1$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial b} = 1$$

로지스틱 회귀 손실 함수

특성이 두개인 경우 => x 2개

$y = w_1x_1 + w_2x_2 + b$

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

$$\frac{\partial}{\partial w_1} w_1x_1 + w_2x_2 + b = x_1$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1} = x_1(\hat{y} - y)$$

Chain Rule 사용

$$\hat{y} = w_1x_1 + w_2x_2 + b$$

$$J(w_1, w_2, b) = \frac{1}{2}(\hat{y} - y)^2$$

$$\frac{\partial}{\partial w_1} J(w_1, w_2, b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

로지스틱 회귀 손실 함수

특성이 두개인 경우 => x 2개

$y = w_1x_1 + w_2x_2 + b$

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

$$\frac{\partial}{\partial w_2} w_1x_1 + w_2x_2 + b = x_2$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2} = x_2(\hat{y} - y)$$

Chain Rule 사용

$$\hat{y} = w_1x_1 + w_2x_2 + b$$

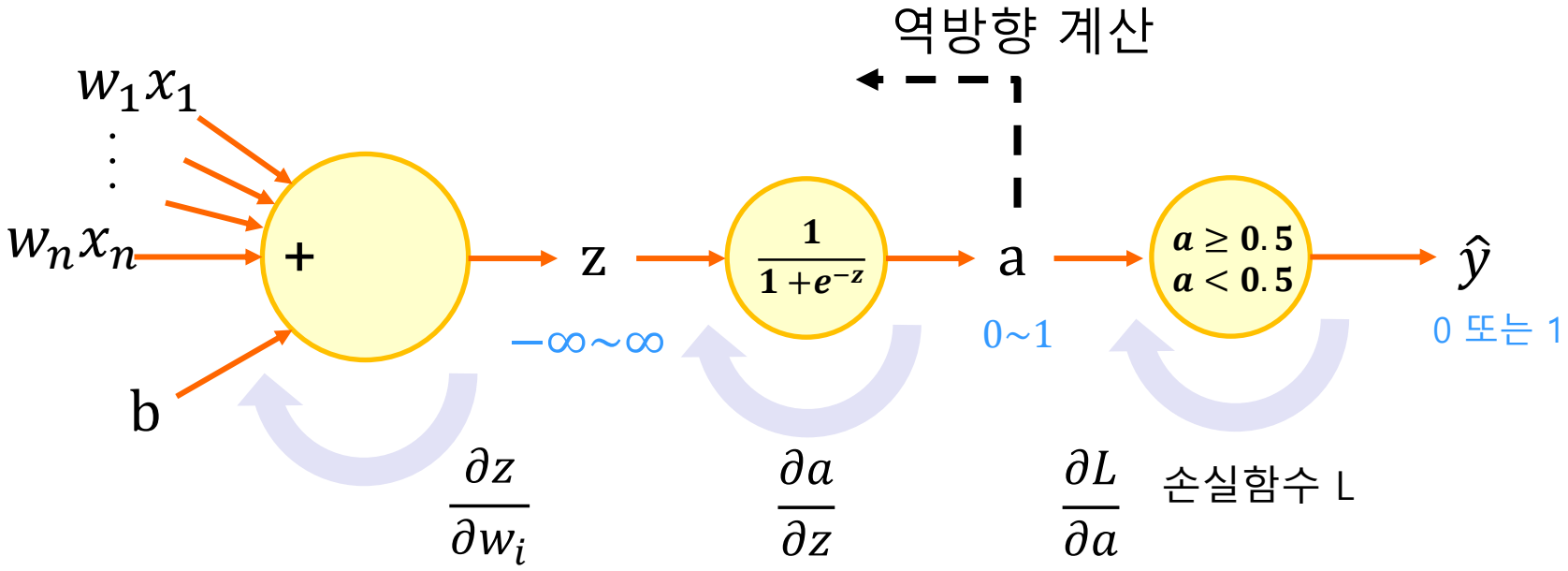
$$J(w_1, w_2, b) = \frac{1}{2}(\hat{y} - y)^2$$

$$\frac{\partial}{\partial w_2} J(w_1, w_2, b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

로지스틱 회귀 손실 함수

특성이 여러개인 경우 => x n개

$z = w_1x_1 + w_2x_2 + \ldots + w_nx_n + b$



chain rule을 이용하여 각 단계의 미분 결과를 곱한다.

w에 대하여 미분

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial w_i} = (a-y)x_i$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

$$\frac{\partial z}{\partial w_i} = x_i$$

b에 대하여 미분

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial a}\frac{\partial a}{\partial z}\frac{\partial z}{\partial b} = (a-y)1$$

$$\frac{\partial L}{\partial a} = -(y\frac{1}{a} - (1-y)\frac{1}{1-a})$$

$$\frac{\partial a}{\partial z} = a(1-a)$$

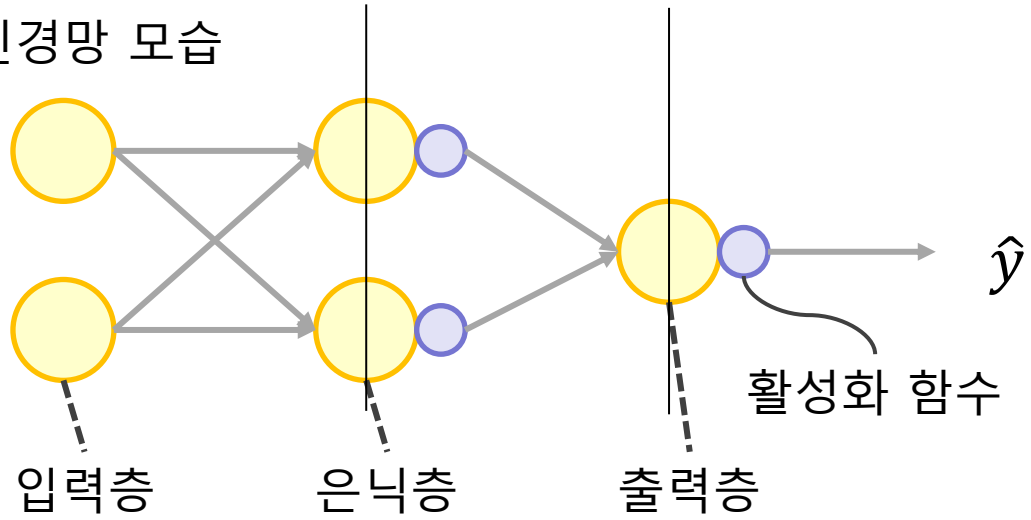
$$\frac{\partial z}{\partial b} = 1$$

제곱 오차의 미분과 로지스틱 손실 함수의 미분은 $\hat{y}$ 이 a로
바뀌었을 뿐 동일 하다.
따라서 선형함수의 결과 값을 activation 함수인 시그모이드를 적용한 값이 a 이다.

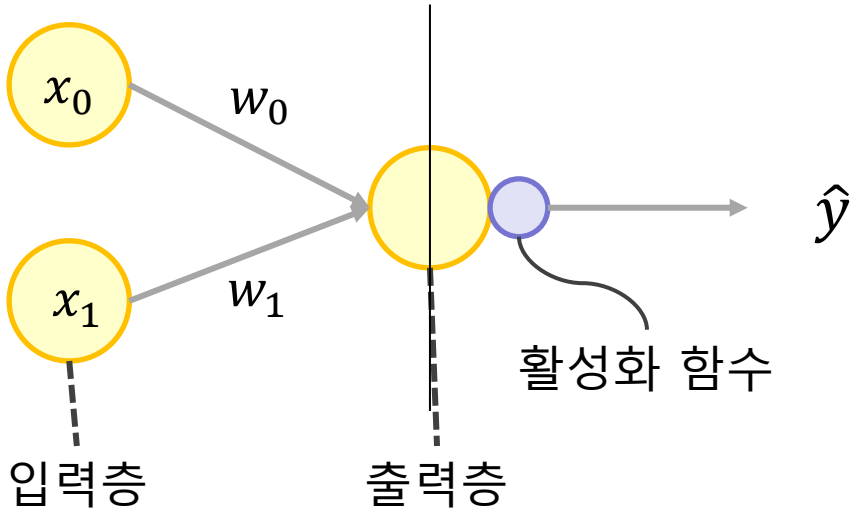
	제곱 오차의 미분	로지스틱 손실 함수의 미분
가중치에 대한 미분	$\frac{\partial SE}{\partial w_i} = (\hat{y} - y)x_i$	$\frac{\partial L}{\partial w_i} = (a - y)x_i$
절편에 대한 미분	$\frac{\partial SE}{\partial b} = (\hat{y} - y)1$	$\frac{\partial L}{\partial b} = (a - y)1$

신경망 구조

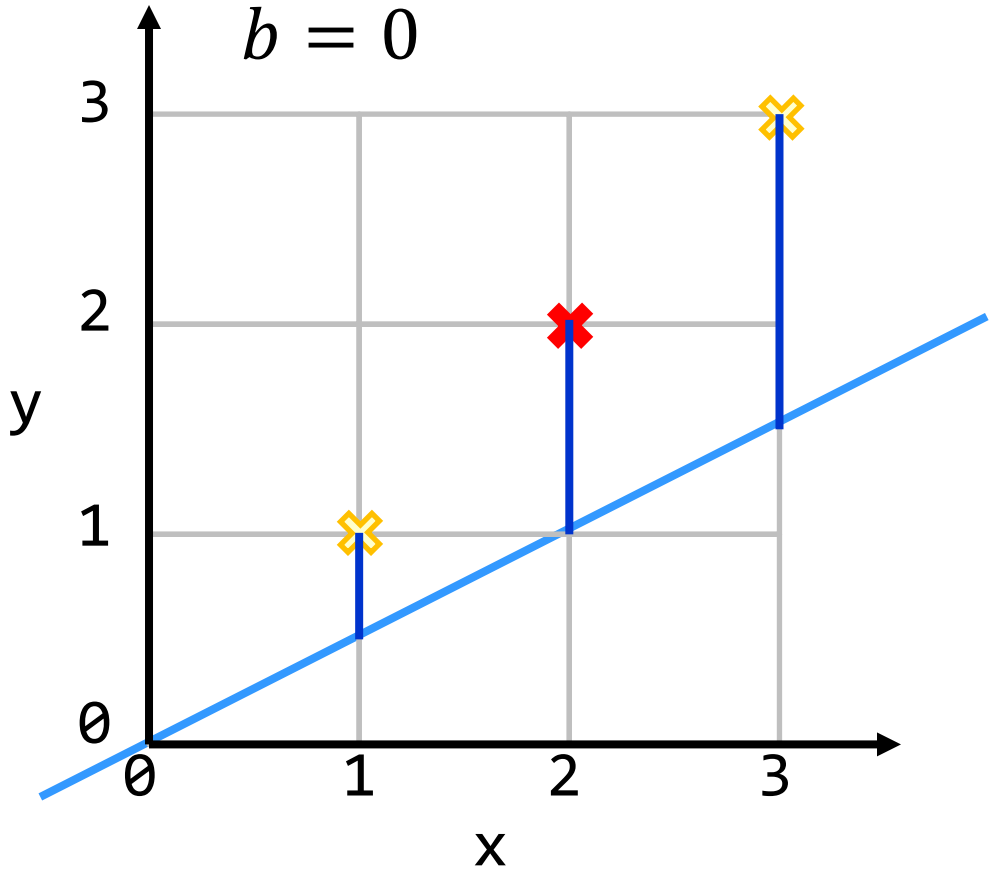
일반적인 신경망 모습



단일층 신경망



$$\hat{y} = wx + b$$
$$w = 0.5$$
$$b = 0$$



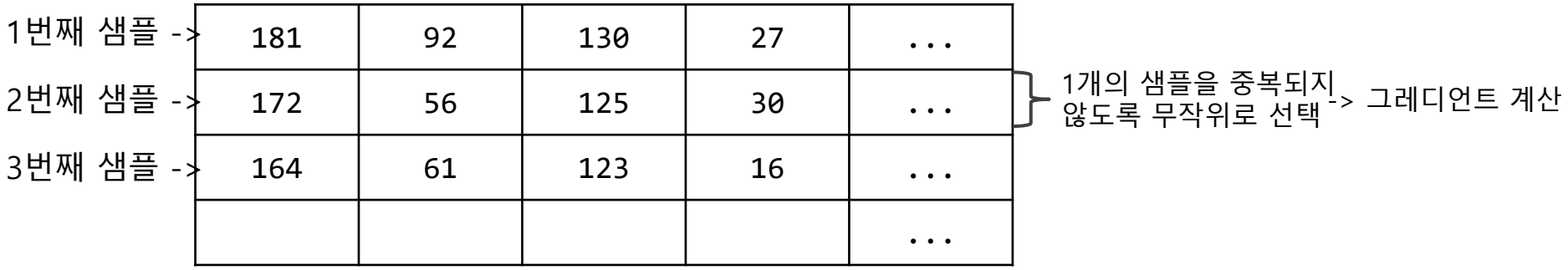
$\frac{1}{2}(\hat{y} - y)^2$ 확률적 경사하강

미니배치 경사하강

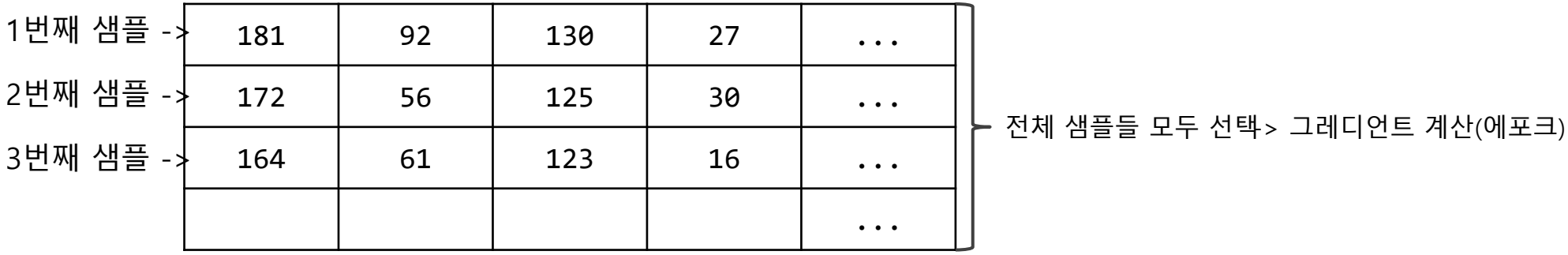
$\frac{1}{2m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2$ 배치 경사하강

(0.25+1+2.25)

여러 가지 경사 하강법



확률적 경사 하강법



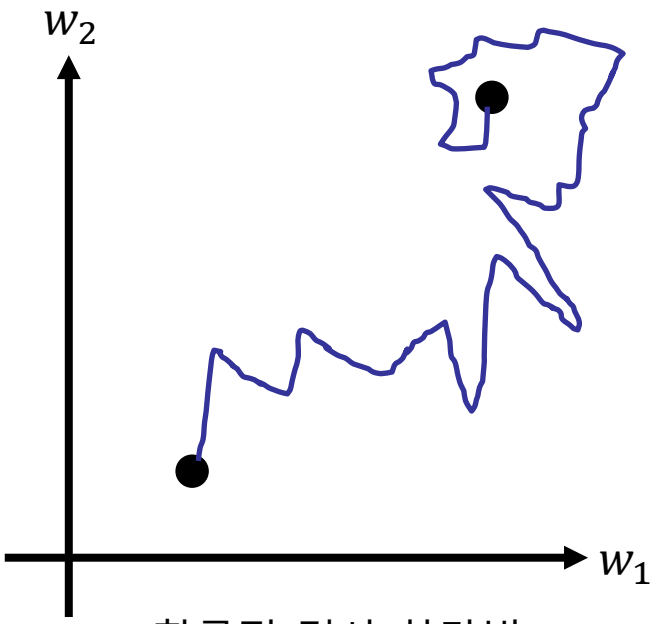
배치 경사 하강법

여러 가지 경사 하강법

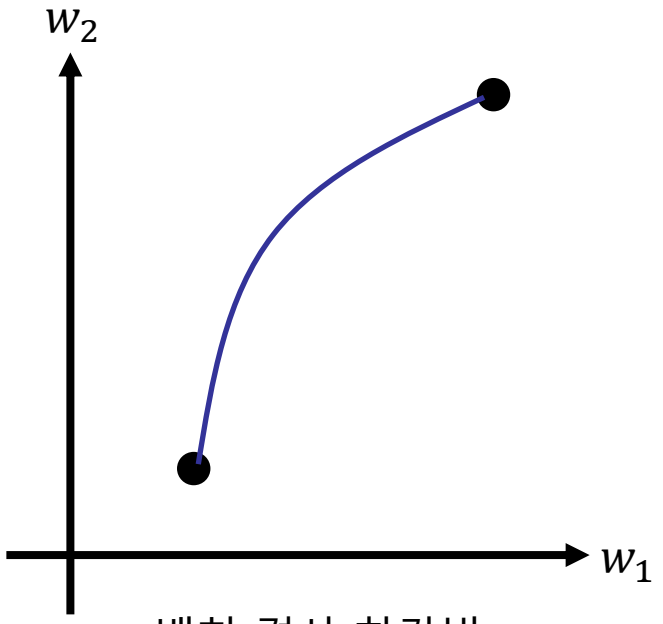
1번째 샘플 ->	181	92	130	27	...
2번째 샘플 ->	172	56	125	30	...
3번째 샘플 ->	164	61	123	16	...
					...

전체 샘플 중 몇 개의
샘플을 중복되지 않도록> 그레디언트 계산
무작위로 선택

미니 배치 경사 하강법



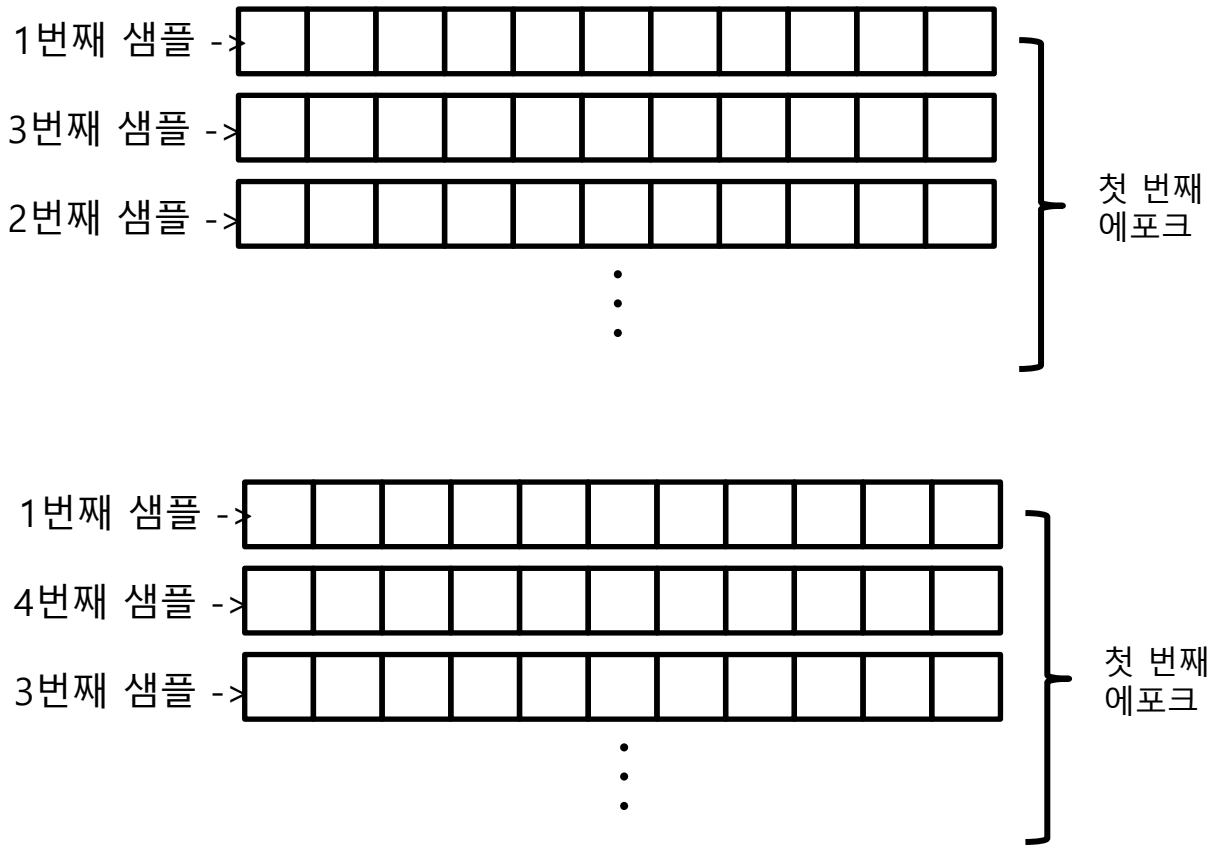
확률적 경사 하강법



배치 경사 하강법

매 에포크 마다 훈련 세트의 샘플 순서 섞기

SGD : 확률적 경사하강

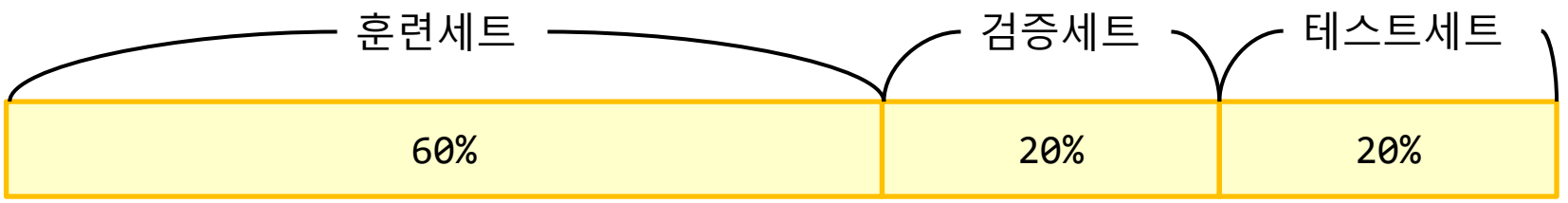


np.random.permutation() 함수를 사용하여 인덱스를 섞을수 있다.

검증 세트 분리

"테스트 세트로 모델를 튜닝하면 실전에서 좋은 성능을 기대하기 어렵다"

검증 세트 분리



데이터 전처리와 특성의 스케일

	당도	무게	...
사과1	4	540	...
사과2	8	700	...
사과3	2	480	...

사과의 당도는 1~10이고 사과의 무게의 범위는 500~1000이다.
이런 경우 '두 특성의 스케일 차이가 크다'라고 말한다.

스케일 조정

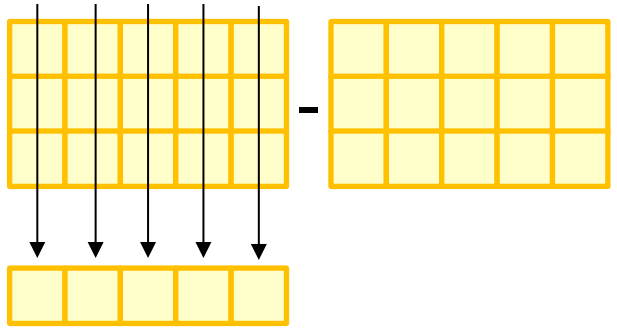
$$z = \frac{x - \mu}{s}$$

표준화는 특성 값에서 평균을 빼고 표준 편차로 나누면 된다.

$$s = \sqrt{\frac{1}{m}\sum_{i=0}^{m}(x_i - \mu)^2}$$

표준 편차 공식

(3, 5)



```
train_mean = np.mean(x_train, axis=0)
train_std = np.std(x_train, axis=0)
x_train_scaled = (x_train - train_mean) / train_std
```

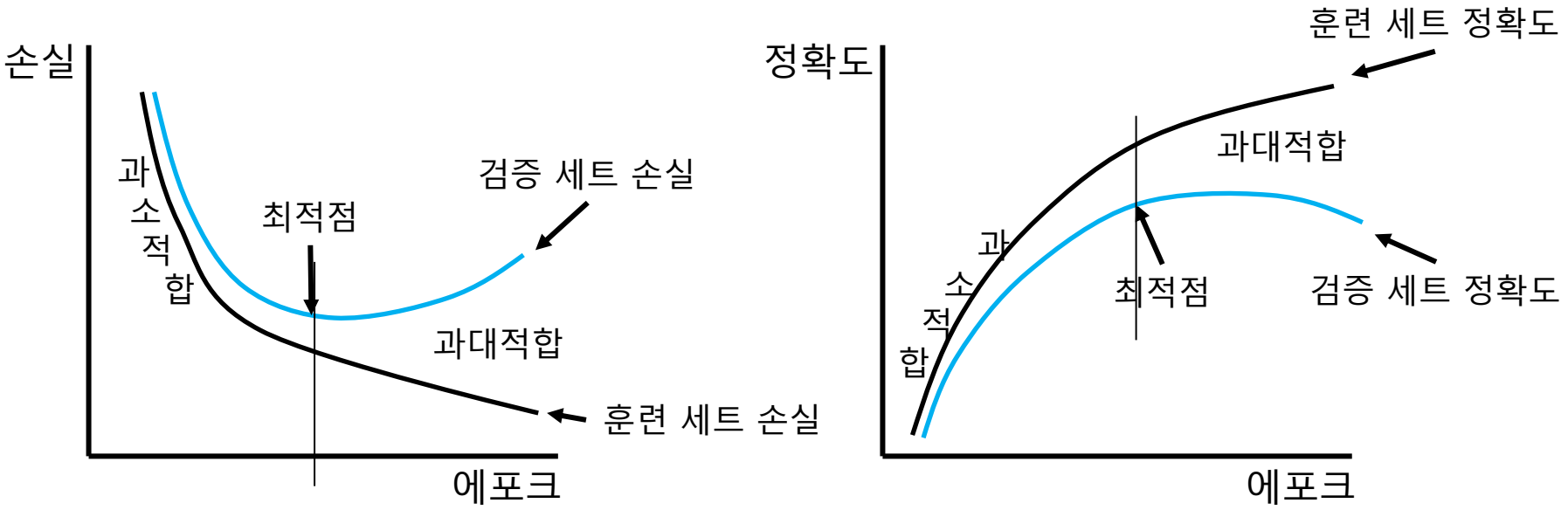
정의

과대적합 :

훈련 세트에서는 좋은 성능을 내지만 검증 세트에서는 낮은 성능을 내는 경우

과소적합 :

훈련 세트와 검증세트의 성능에는 차이가 크지 않지만 모두 낮은 성능을 내는 경우



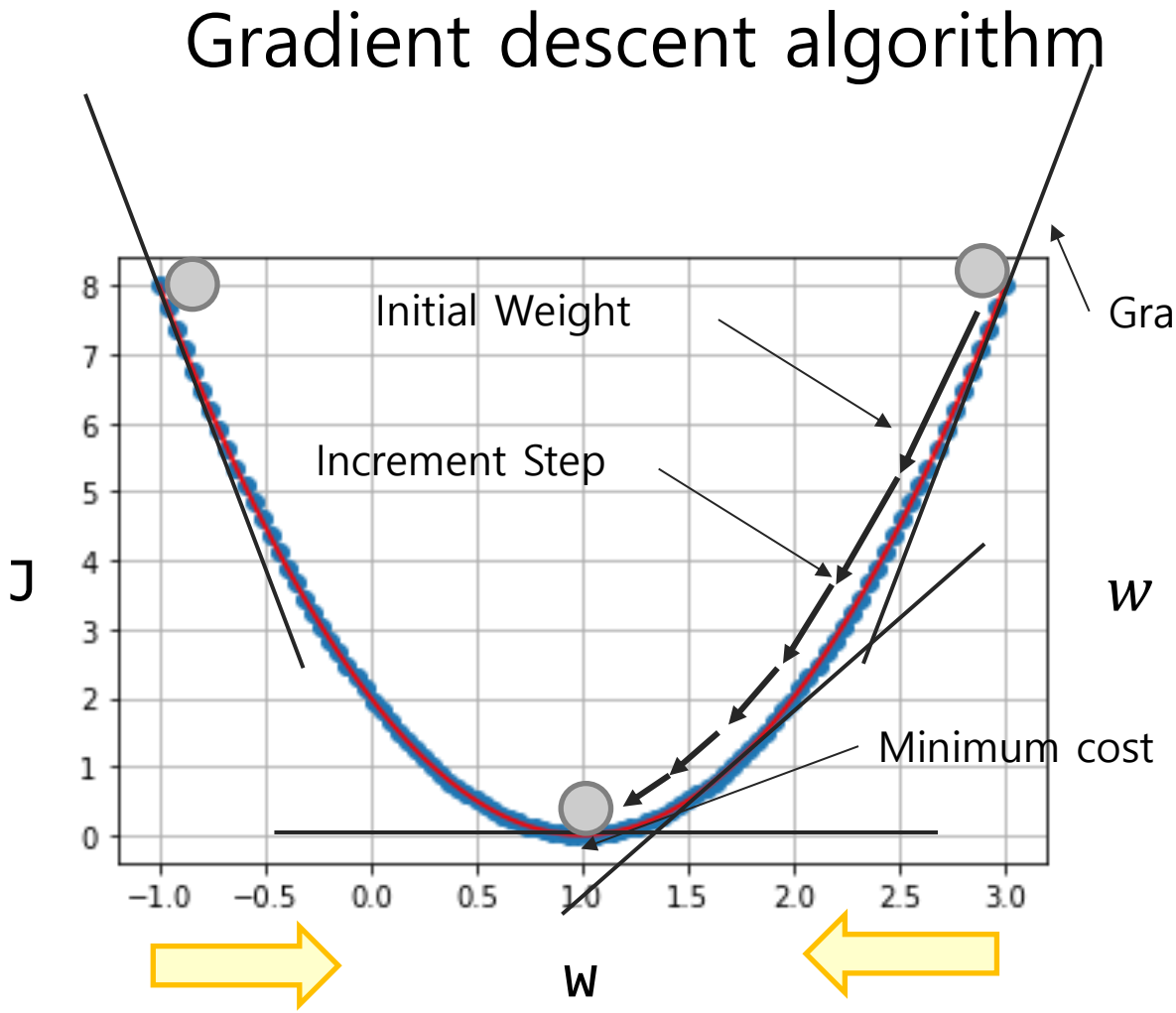
L1 규제는 손실 함수에 가중치의 절대값인 L1 노름(norm)을 추가한다.

$$\left|\left|w\right|\right|_1 = \sum_{i=1}^{n} |w_i|$$

$$L = -(ylog(a) + (1-y)\log(1-a))$$

L1 노름을 그냥 더하지 않고 규제의 양을 조절하는 파라미터 α 를 곱한 후 더한다.

$$L = -(ylog(a) + (1-y)\log(1-a)) + \alpha \sum_{i=1}^{n} |w_i|$$



$$\hat{y} = wx + b$$

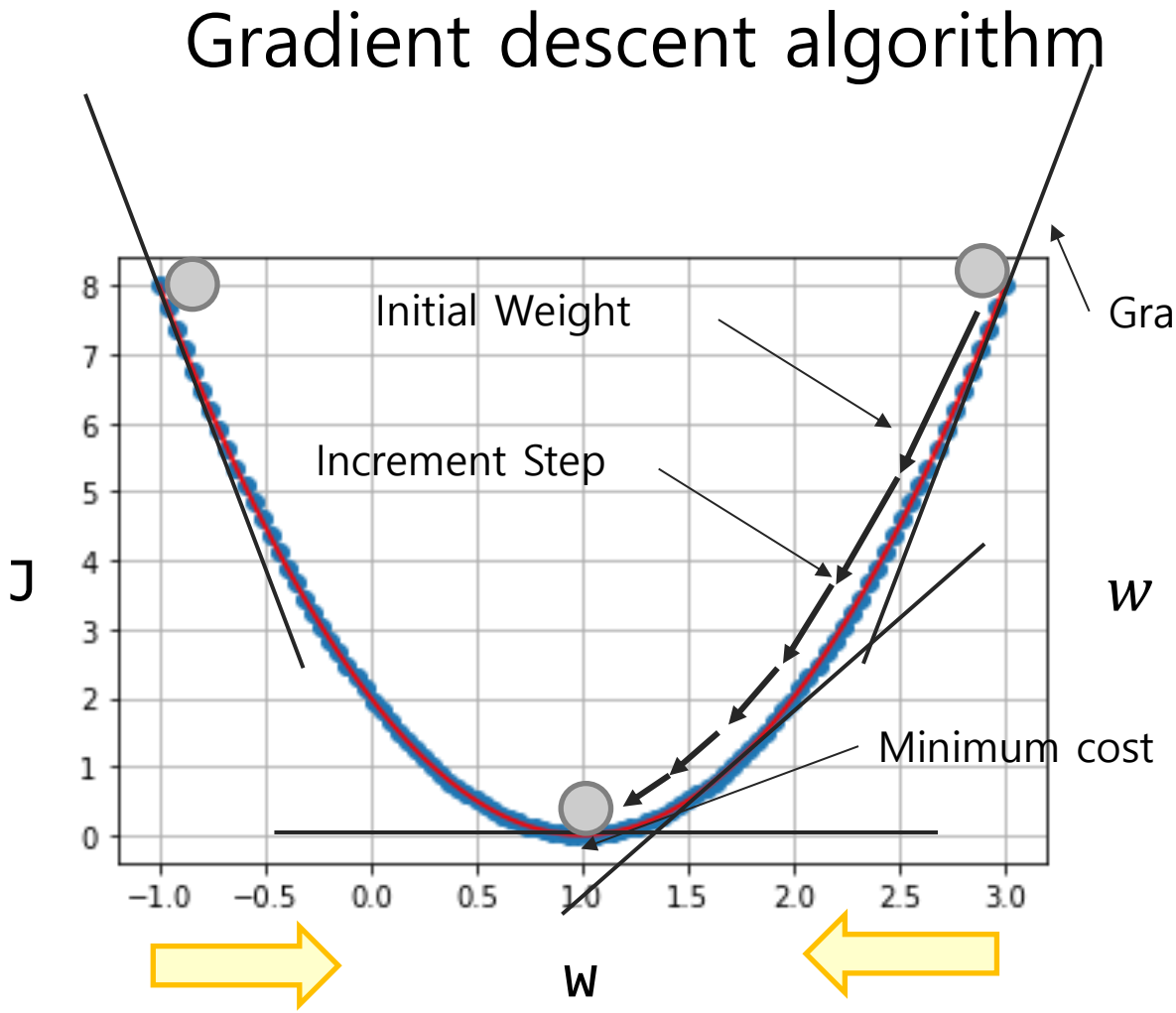
$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

`0.08`

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{sign(w)}$$

`-0.08 + 0.01`

$$x = 2$$
$$y = 2$$



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{sign(w)}$$

+0.08 - 0.01

$$x = 2$$

$$y = 2$$

L1 규제의 미분

L1 규제는 손실 함수에 가중치의 절대값인 L1 노름(norm)을 추가한다.

$$\frac{\partial}{\partial w} L = (a - y)x + \alpha * sign(w)$$

$$w = w - \eta \frac{\partial L}{\partial w} = w - \eta(a - y)x + \alpha * sign(w)$$

파이썬으로 작성된 L1 규제 적용된 오차 역전파 구현

```
w_grad += alpha * np.sign(w)
```

회귀 모델에 L1 규제를 추가한 것을 라쏘 모델이라 한다.

L2 규제는 손실 함수에 가중치에 대한 L2 노름(norm)의 제곱을 더한다.

$$\left|\left|w\right|\right|_2 = \sum_{i=1}^{n} |w_i|^2$$

$$L = -(ylog(a) + (1-y)\log(1-a))$$

L1 노름을 그냥 더하지 않고 규제의 양을 조절하는 파라미터 α 를 곱한 후 더한다.

$$L = -(ylog(a) + (1-y)\log(1-a)) + \frac{1}{2}\alpha \sum_{i=1}^{n} |w_i|^2$$

L2 규제의 미분

L2 규제는 손실 함수에 가중치에 대한 L2 노름(norm)의 제곱을 더한다.

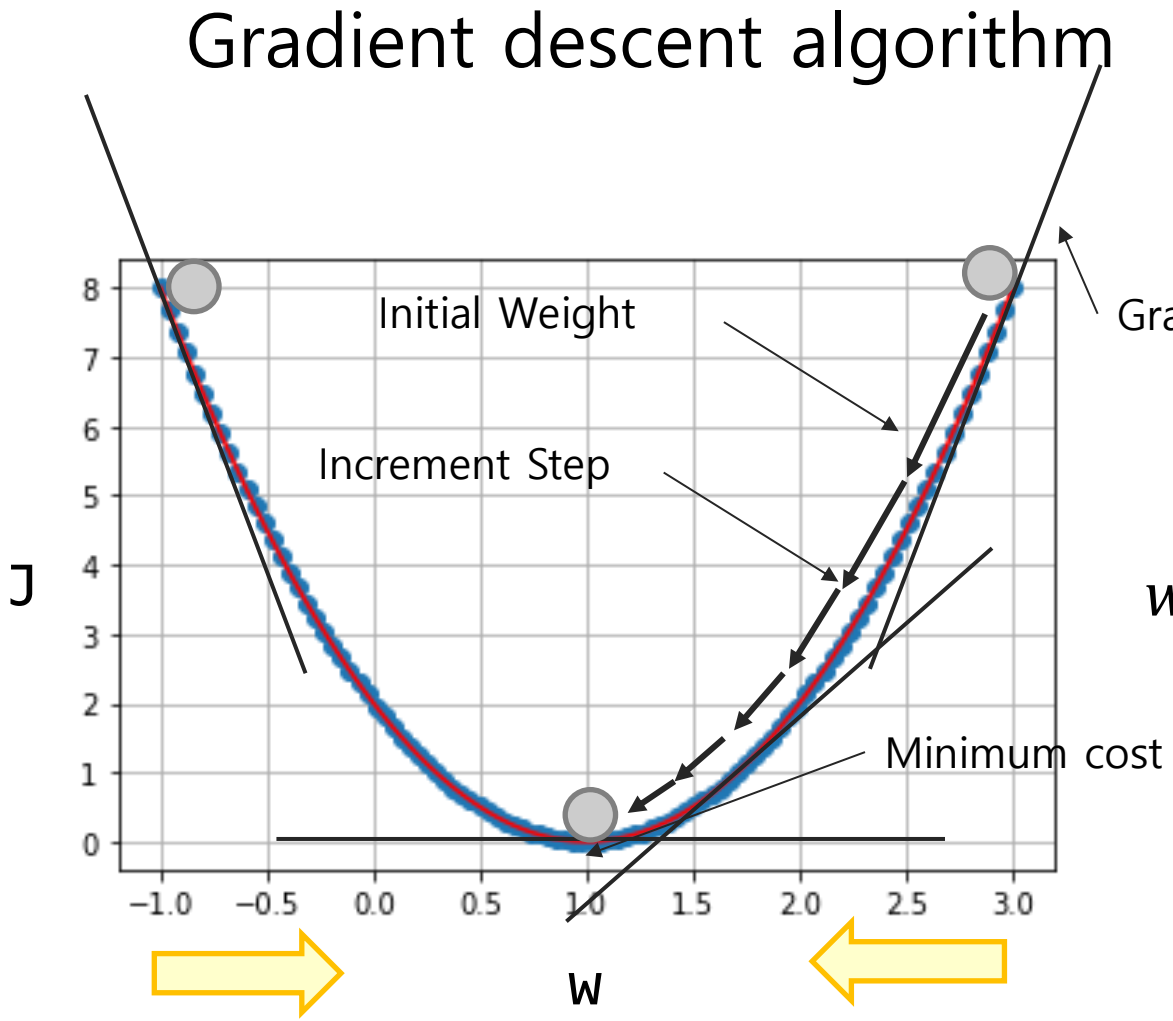
$$\frac{\partial}{\partial w} L = -(y - a)x + \alpha * w$$

$$w = w - \eta \frac{\partial L}{\partial w} = w + \eta((y - a)x - \alpha * w)$$

파이썬으로 작성된 L2 규제 적용된 오차 역전파 구현

```
w_grad += alpha * w
```

회귀 모델에 L2 규제를 추가한 것을 릿지 모델이라 한다.



$$\hat{y} = wx + b$$

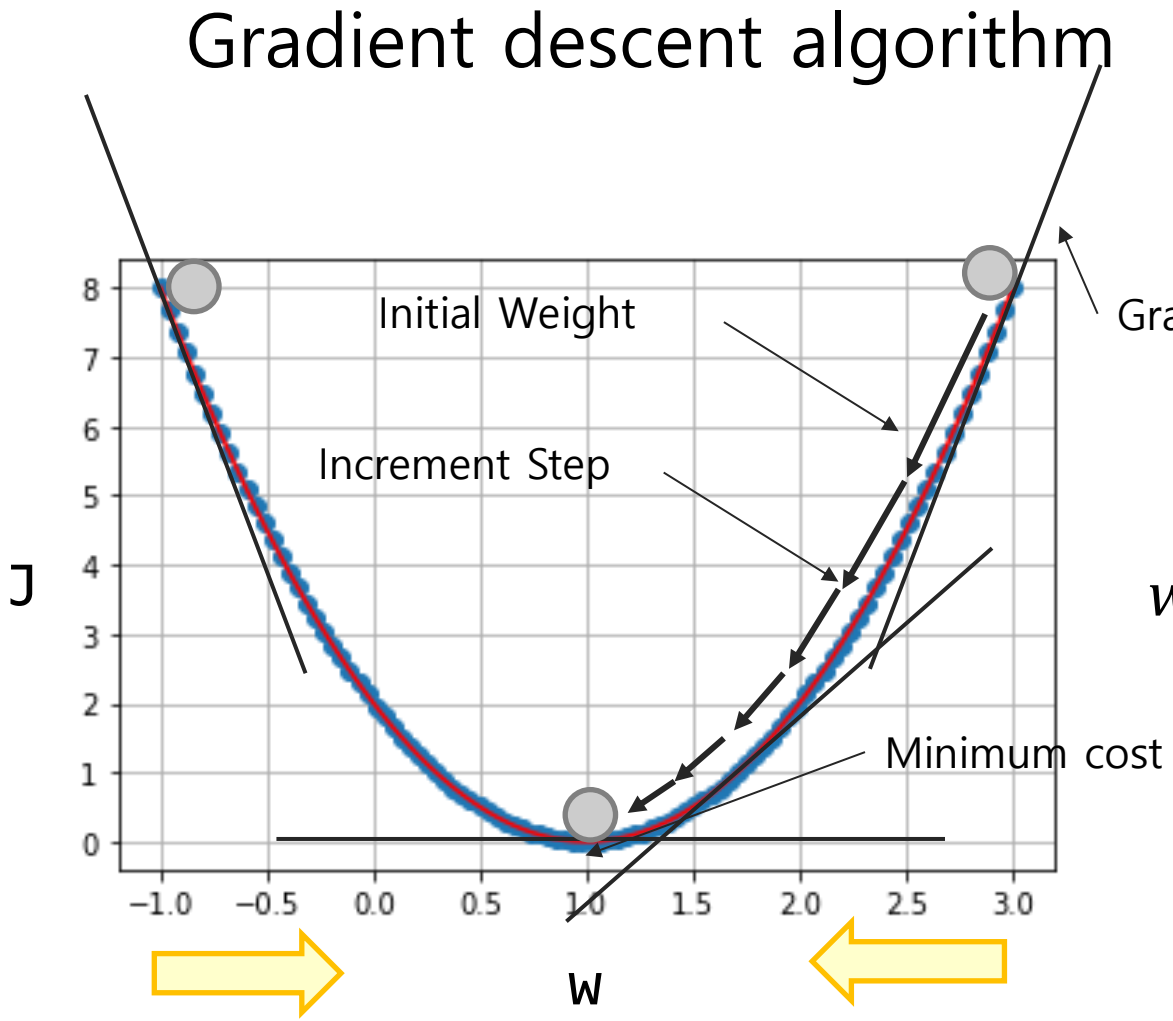
$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

0.08

$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{w}$$

-0.08 + 0.03

$$x = 2$$
$$y = 2$$



$$\hat{y} = wx + b$$

$$w = w - \alpha(\hat{y} - y)\mathrm{x}$$

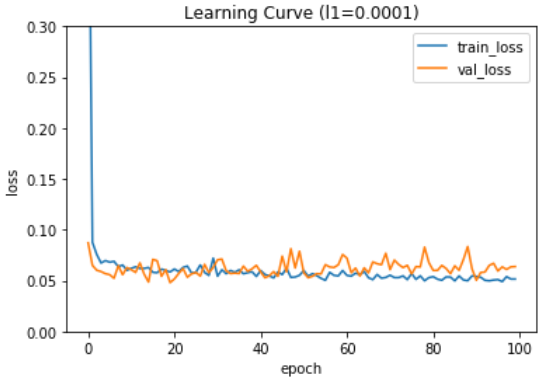
```
0.08
```

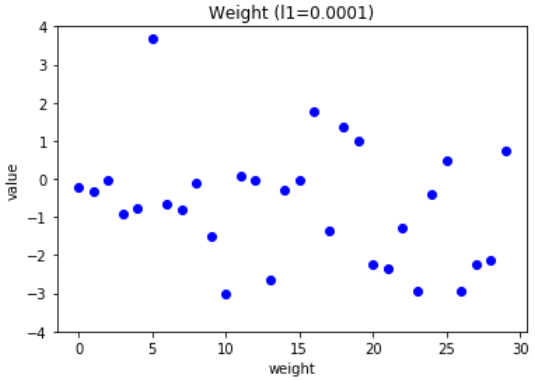
$$w = w - \alpha(\hat{y} - y)\mathrm{x} + 0.01 * \mathrm{w}$$

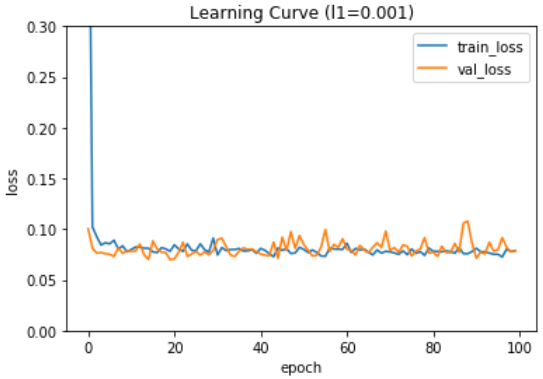
```
+0.08 - 0.01
```

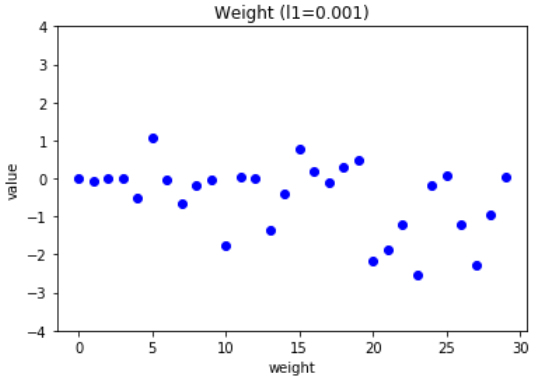
$$x = 2$$
$$y = 2$$

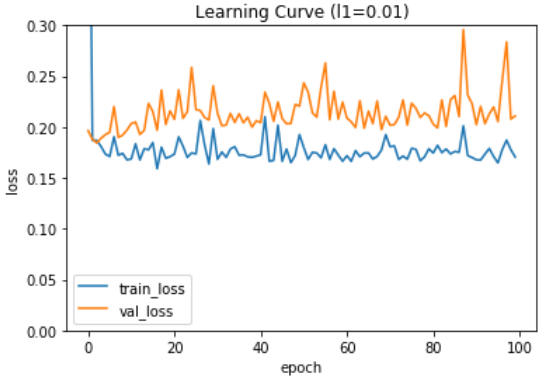
L1 규제 적용

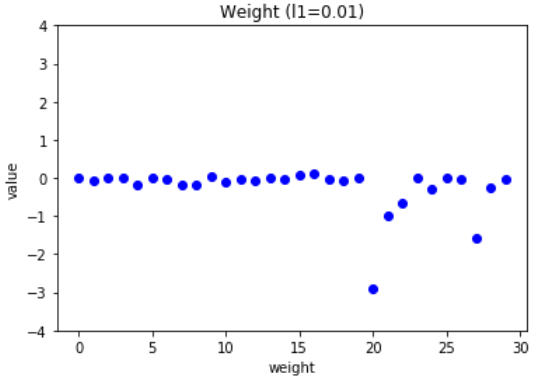

Learning Curve (l1=0.0001)
train_loss
val_loss
loss
epoch


Weight (l1=0.0001)
value
weight

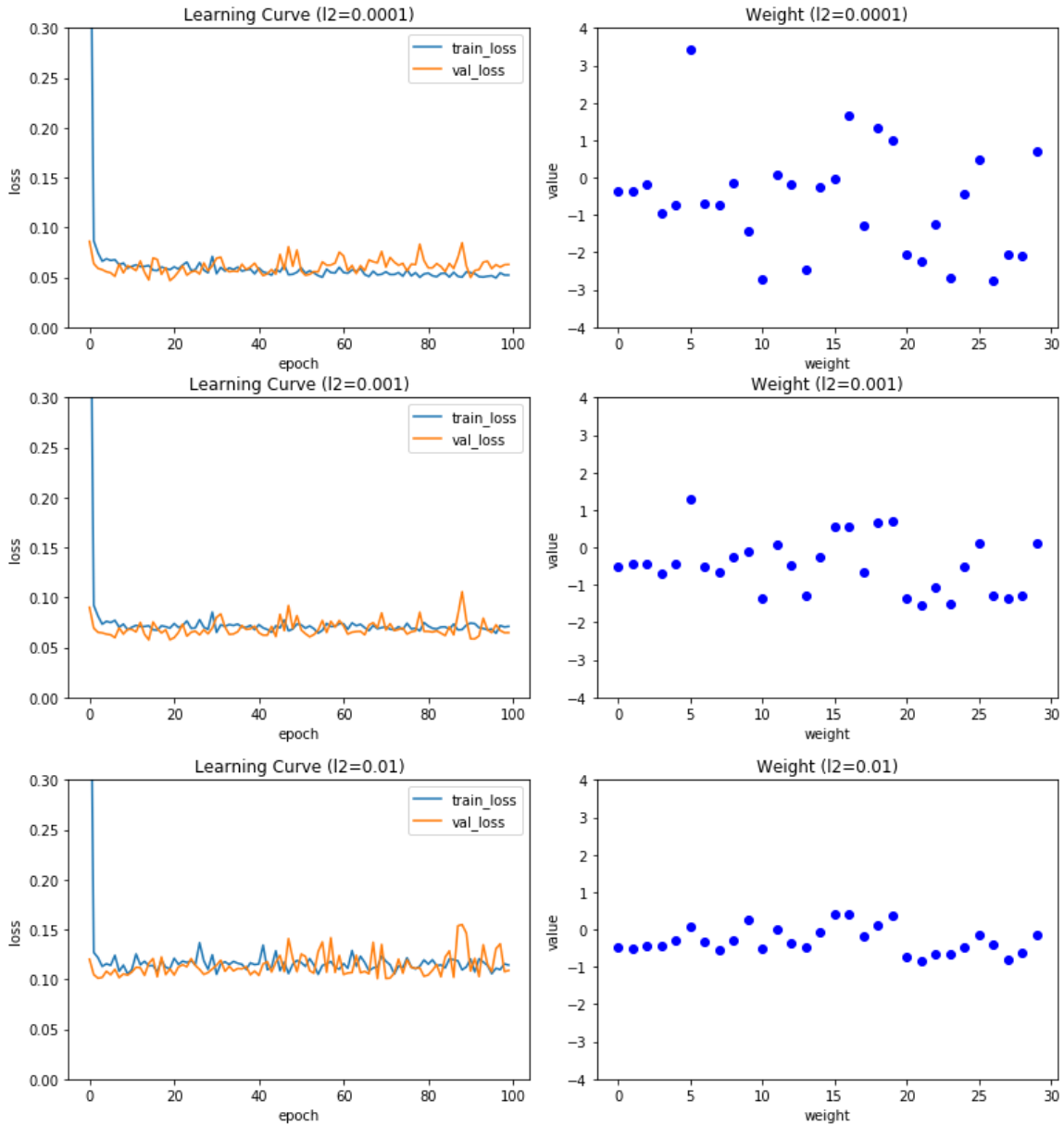

Learning Curve (l1=0.001)
train_loss
val_loss
loss
epoch


Weight (l1=0.001)
value
weight

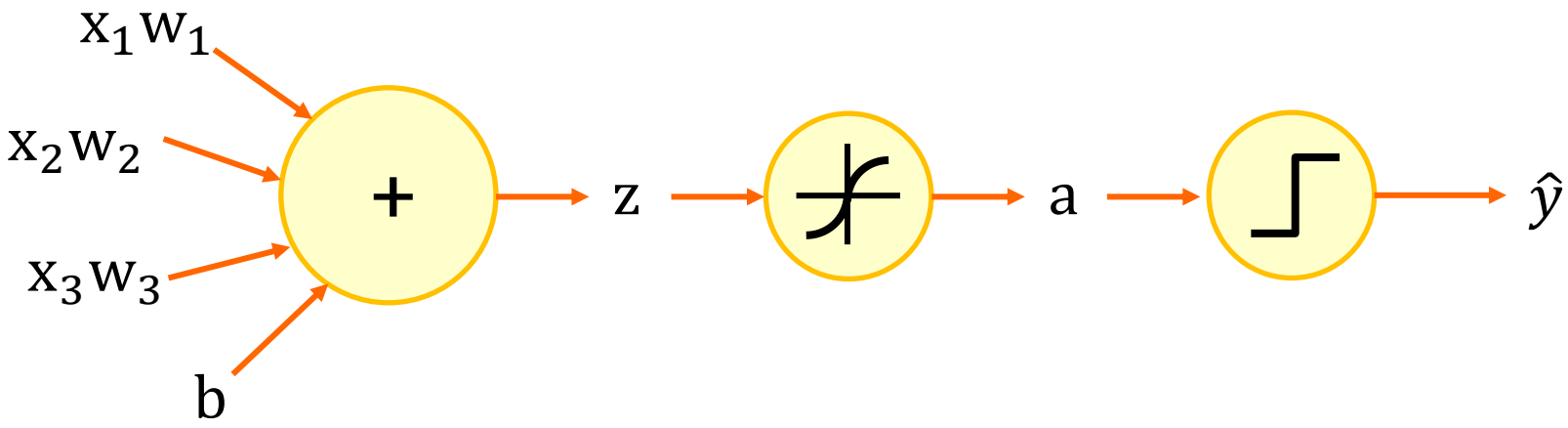

Learning Curve (l1=0.01)
train_loss
val_loss
loss
epoch


Weight (l1=0.01)
value
weight

L2 규제 적용


Learning Curve (l2=0.0001)
train_loss
val_loss
loss
epoch
Weight (l2=0.0001)
value
weight
Learning Curve (l2=0.001)
train_loss
val_loss
loss
epoch
Weight (l2=0.001)
value
weight
Learning Curve (l2=0.01)
train_loss
val_loss
loss
epoch
Weight (l2=0.01)
value
weight

벡터 연산과 행렬 연산

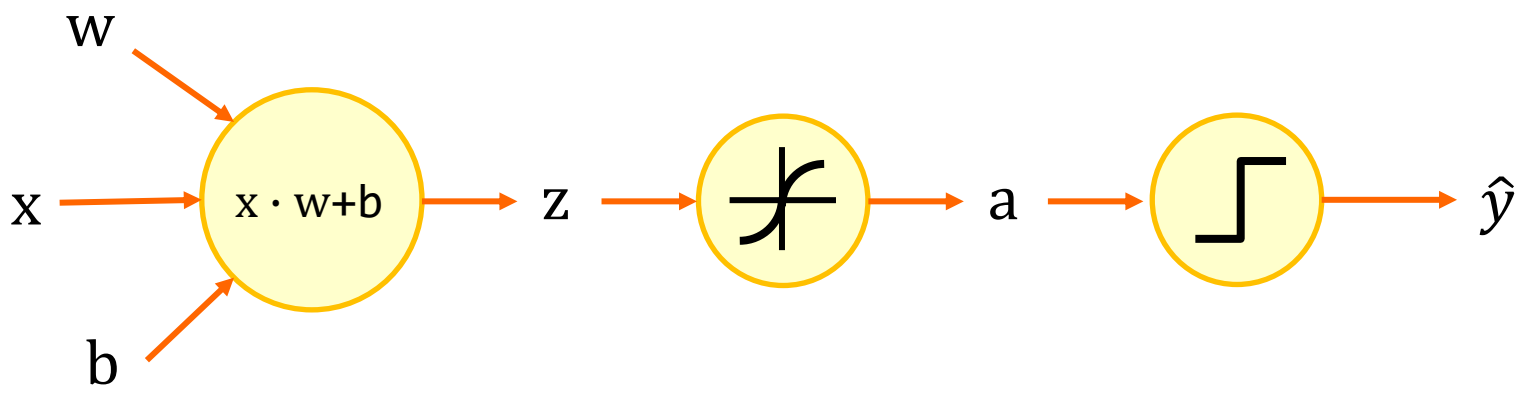



```
def forpass(self, x):
    z = np.sum(x * self.w) + self.b
    return z
```

넘파이의 원소별 곱셈

$$x = [x_1, x_1, \cdots, x_n]$$
$$w = [w_1, w, \cdots, w_n]$$
$$x * w = [x_1 * w_1, x_2 * w_2, \cdots, x_n * w_1]$$

벡터 연산과 행렬 연산



점 곱을 행렬 곱셈으로 표현

$$XW = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = x_1 \times w_1 + x_2 \times w_2 + x_3 \times w_3$$

(1, 3)(3, 1) => (1,1)

```
z = np.dot(x , self.w) + self.b
```

배치경사 하강법

$$XW = \begin{bmatrix} x_1^{(1)} & x_2^{(1)} & x_3^{(1)} \\ x_1^{(2)} & x_2^{(2)} & x_3^{(2)} \\ & \vdots & \\ x_1^{(m)} & x_2^{(m)} & x_3^{(m)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} x_1^{(1)}w_1 + x_2^{(1)}w_2 + x_3^{(1)}w_3 \\ x_1^{(2)}w_1 + x_1^{(2)}w_1 + x_1^{(2)}w_1 \\ \vdots \\ x_1^{(m)}w_1 + x_2^{(m)}w_2 + x_3^{(m)}w_3 \end{bmatrix}$$

행렬곱 가능과 곱 결과 크기

$$(m, n) \cdot (n, k) = (m, k)$$

첫 번째 행렬의 열(n)과 두 번째 행렬의 행(n)의 크기는 반드시 같아야 한다.

곱 결과의 크기는 첫 번째 행렬의 행(m)과 두 번째 행렬의 열(k)의 크기가 된다.

정방향 계산을 행렬 곱셉으로
표현

$$XW = \begin{bmatrix} x_1^{(1)} & \cdots & x_{30}^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(364)} & \cdots & x_{30}^{(364)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30} \end{bmatrix} + \begin{bmatrix} b \\ b \\ \vdots \\ b \end{bmatrix} = \begin{bmatrix} z^{(1)} \\ z^{(2)} \\ \vdots \\ z^{(364)} \end{bmatrix}$$

$$= \begin{bmatrix} x_1^{(1)} & \cdots & x_{30}^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(364)} & \cdots & x_{30}^{(364)} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{30} \end{bmatrix} + b = \begin{bmatrix} x_1^{(1)}w_1 + x_2^{(1)}w_2 + \cdots + x_{30}^{(1)}w_{30} + b \\ \vdots \\ \vdots \\ x_1^{(364)}w_1 + x_2^{(364)}w_2 + \cdots + x_{30}^{(364)}w_{30} + b \end{bmatrix}$$

배치경사 하강법

그레디언트 계산

$$X^T E = \begin{bmatrix} x_1^{(1)} & \cdots & x_1^{(364)} \\ x_2^{(1)} & \cdots & x_2^{(364)} \\ \vdots & \cdots & \vdots \\ x_{30}^{(1)} & \cdots & x_{30}^{(364)} \end{bmatrix} \begin{bmatrix} e^{(1)} \\ e^{(3)} \\ \vdots \\ e^{(364)} \end{bmatrix} = \begin{bmatrix} g_1 \\ g_2 \\ \vdots \\ g_{30} \end{bmatrix}$$

그레디언트는 X와 E의 행렬곱이다.
그러나 벡터 연산에서 X의 크기는 (364, 30) 이고
E는 (364, 1) 이므로 행렬의 곱을 할 수 없다.

이때 X를 전치하면 행과 열이 바뀌므로
X^T (30, 364) 가 되므로

X^TE 는 (30, 364) •(364, 1) 이므로 곱이 가능하고
결과는 (30, 1) 이 되므로 그레디언트와 같은 행렬을 구할 수 있다.

$$W = W - X^T E$$
$$(30, 364)(364, 1)$$
$$(30,1)$$

$X \Rightarrow (364, 30)$

$W \Rightarrow (30, 1)$

$\hat{y} \Rightarrow (364, 1)$

$a \Rightarrow (364, 1)$

$err \Rightarrow (a - y) \Rightarrow (364, 1)$

행렬곱의 미분(MatMul)

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

$$= \hat{y} - y = E$$

$$\frac{\partial}{\partial w} wx + b$$

$$= X$$

$$\frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w} = X^T E$$

Chain Rule 사용

$$\hat{y} = XW + b$$

$$J(w,b) = \frac{1}{2}(\hat{y} - y)^2$$

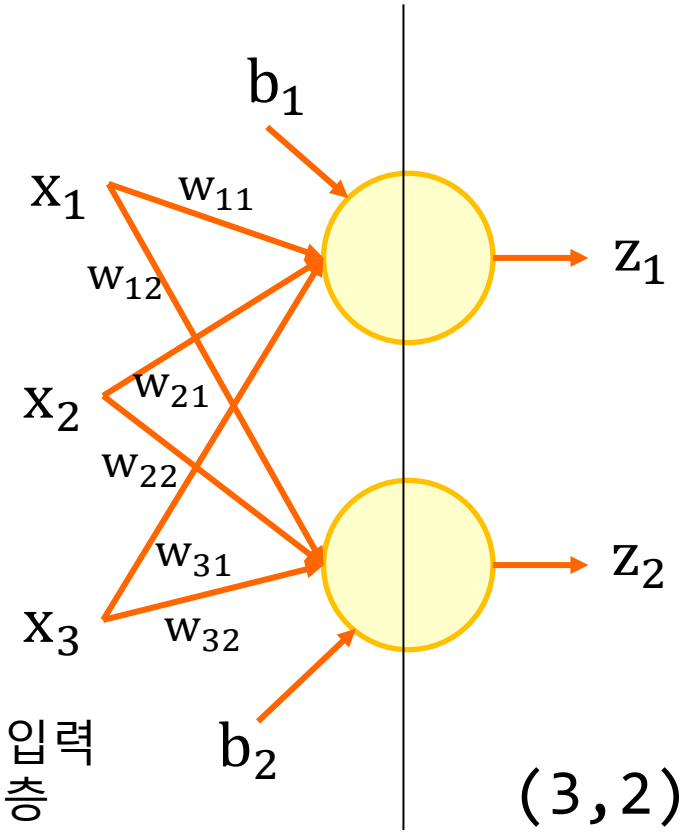
$$\frac{\partial}{\partial w} J(w,b) = \frac{\partial J}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w}$$

전체식을 W로 미분한 경우
사라진 W의 자리에는 뒤에서 날라온
미분값인 E를 쓰고
남아 있는 X는 전치하여
행렬 곱을 한다.

하나의 층에 여러개의 뉴런 사용

https://playground.tensorflow.org/

하나의 층에 여러개의 뉴런 사용



$$XW_1 + b_1 = z_1$$

$$XW_2 + b_2 = z_2$$

(3,2) => (특성수,뉴런수) => (입력수,출력수)

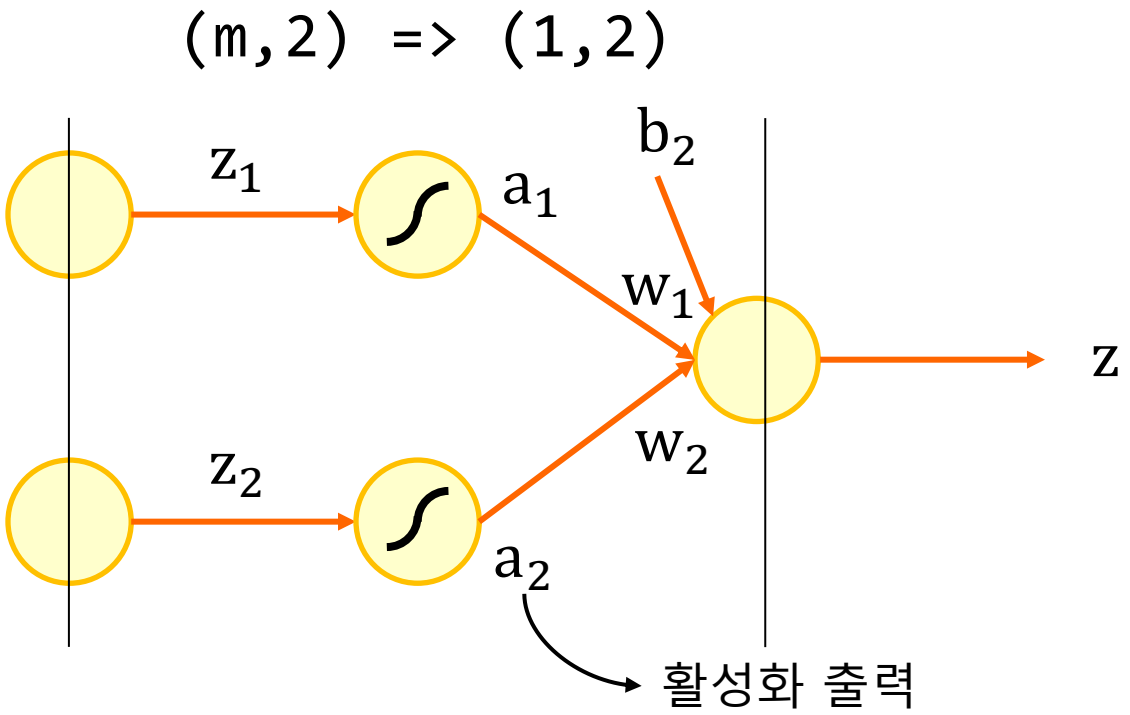
$$x_1w_{11} + x_2w_{21} + x_3w_{31} + b_1 = z_1$$

$$x_1w_{12} + x_2w_{22} + x_3w_{32} + b_2 = z_2$$

$$[x_1 \quad x_2 \quad x_3]\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} + [b_1 \quad b_2] = [z_1 \quad z_2]$$

(m,n) (n,2) => (m,2)

하나의 층에 여러개의 뉴런 사용



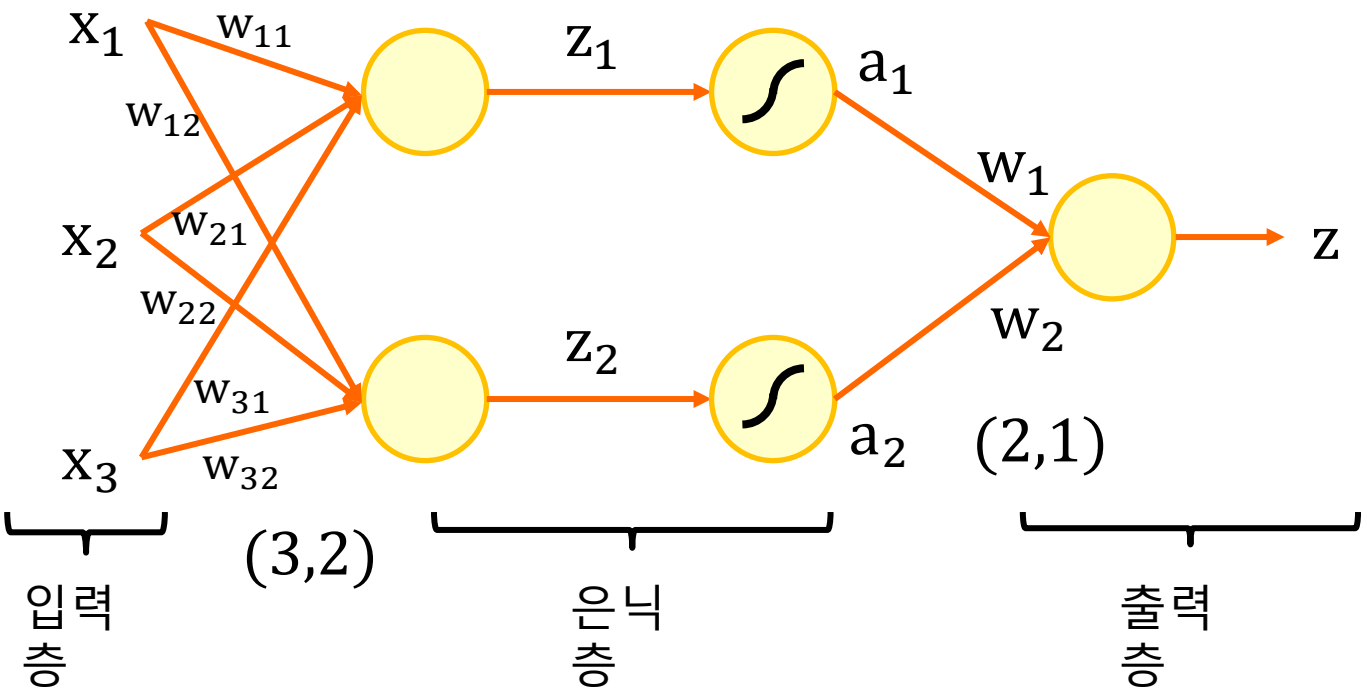
$$a_1 w_1 + a_2 w_2 + b_2 = z$$

$$[a_1 \quad a_2] \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b_2 = z$$

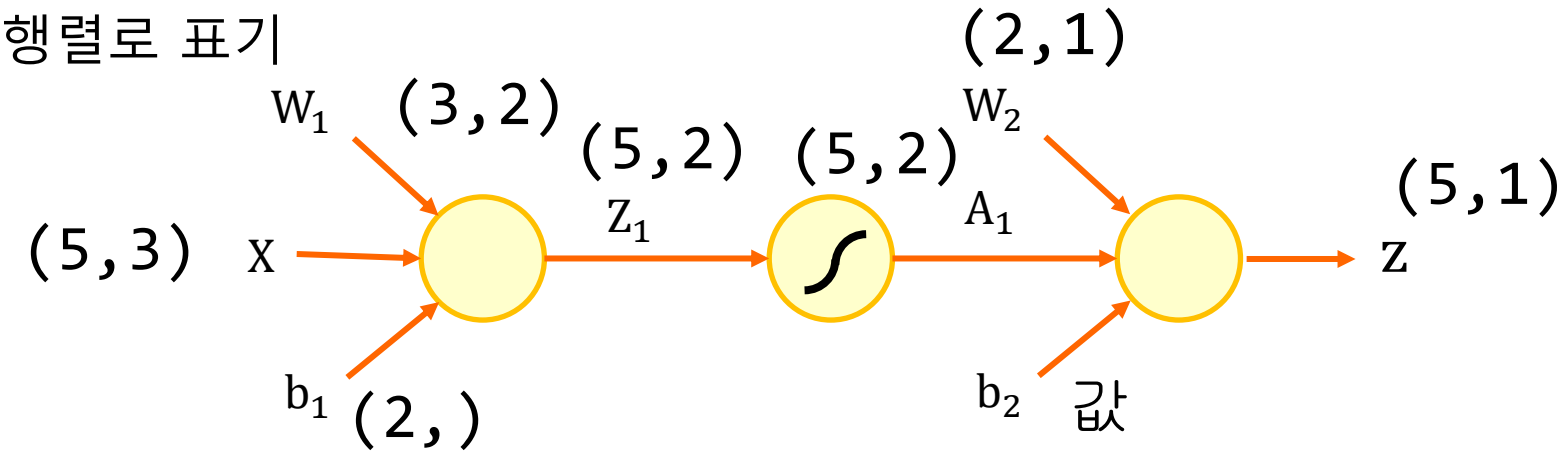
(m,2)(2,1) => (m,1)

$$A_1 W_2 + b_2 = z_2$$

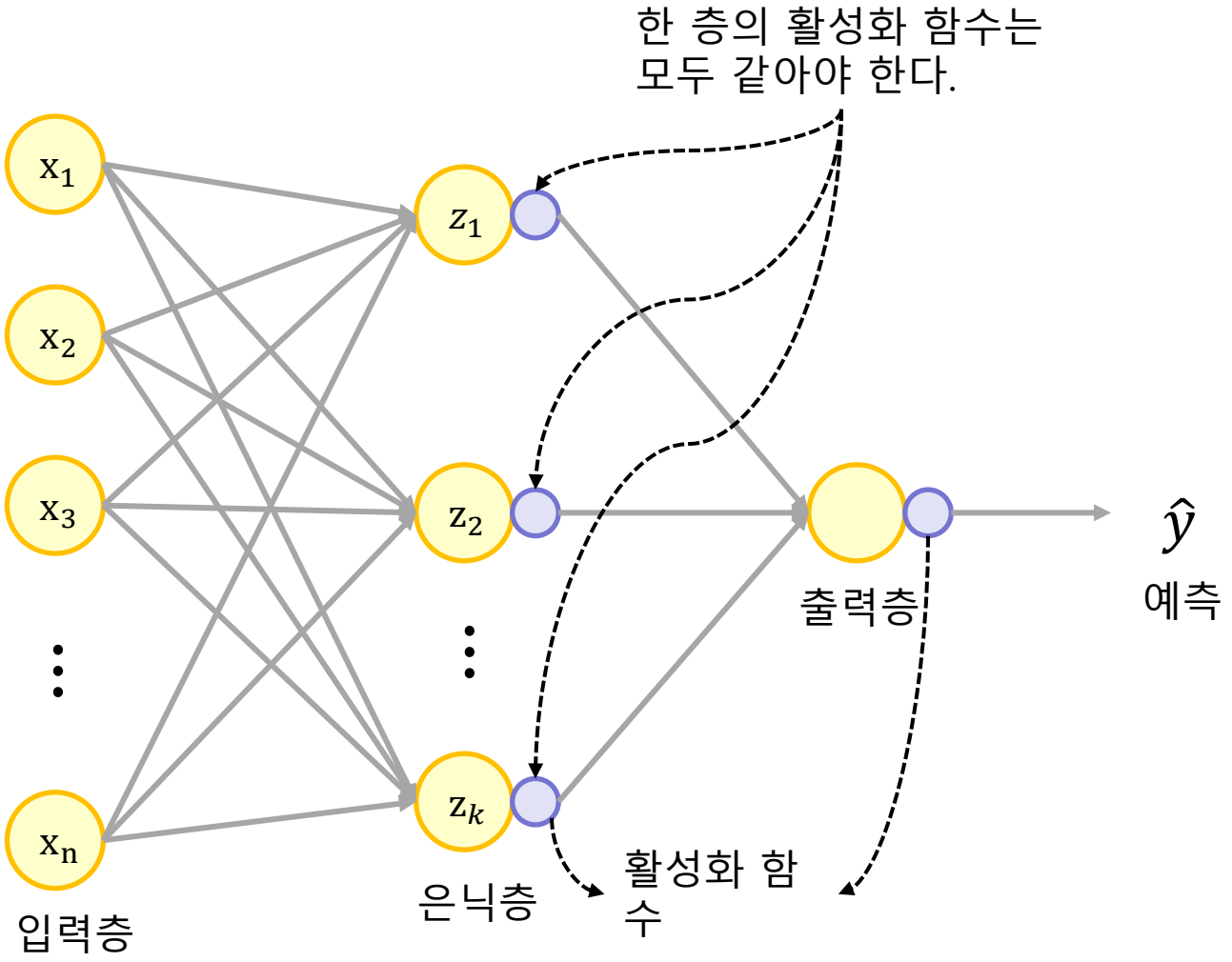
하나의 층에 여러개의 뉴런 사용



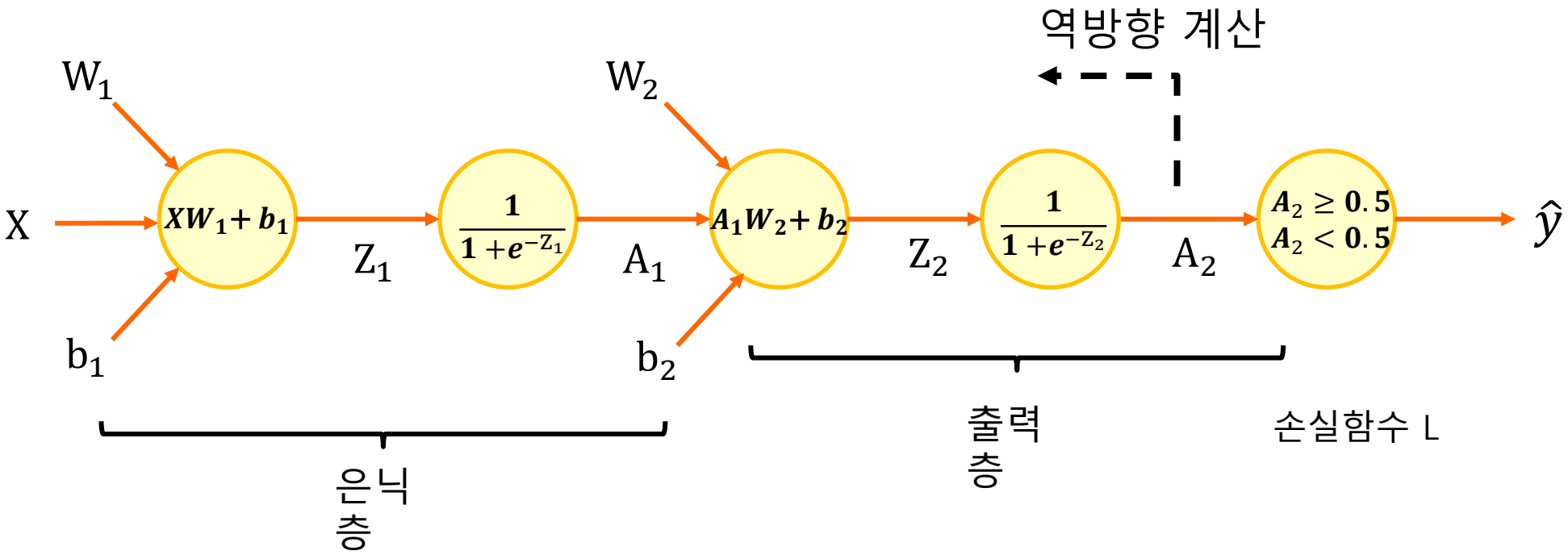
행렬로 표기



2층의 완전연결 신경망


한 층의 활성화 함수는
모두 같아야 한다.
x_1
x_2
x_3
x_n
z_1
z_2
z_k
$\hat{y}$
예측
출력층
은닉층
활성화 함수
입력층
X
W_1
A
W_2
(m, n)(n, k) => (m, k)(k, 1) => (m, 1)

다층 신경망 backprop 이해

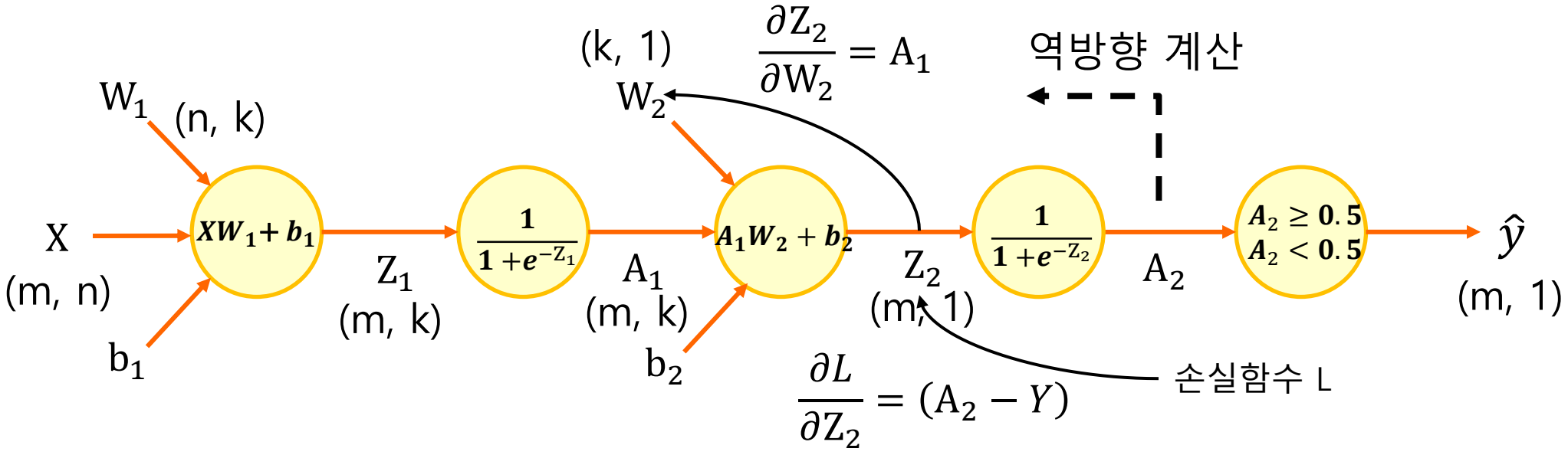


가중치 W_2 대하여 손실 함수를 미분한다.

$$\frac{\partial L}{\partial W_2} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial W_2}$$

$$\frac{\partial L}{\partial W_2} = (A_1^T \cdot Err)$$

np.dot(self.A1.T, Err)



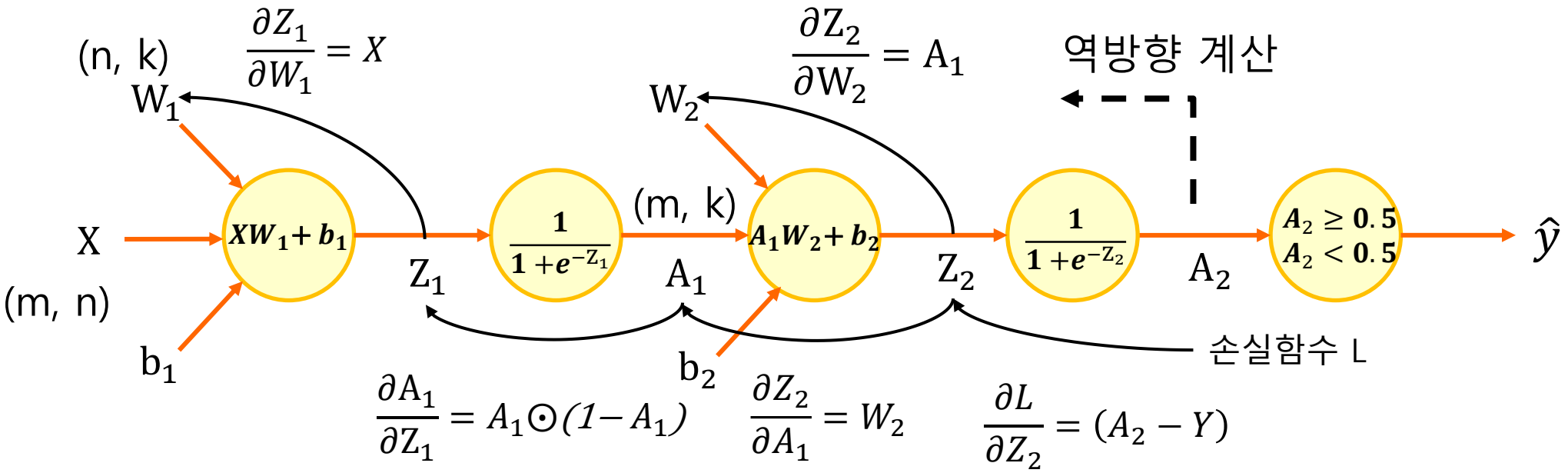
가중치 W_1 대하여 손실 함수를 미분한다. ⊙ : 멤버 곱

a ⊙ (1-a)

$$\frac{\partial L}{\partial W_1} = \frac{\partial L}{\partial Z_2}\frac{\partial Z_2}{\partial A_1}\frac{\partial A_1}{\partial Z_1}\frac{\partial Z_1}{\partial W_1} = X^T\left((A_2 - Y)\,W_2^T \odot A_1 \odot (1 - A_1)\right)$$

(m, 1)(1, k) => (m, k)

(n,m)(m, k) =>(n, k)



미니배치 경사 하강법

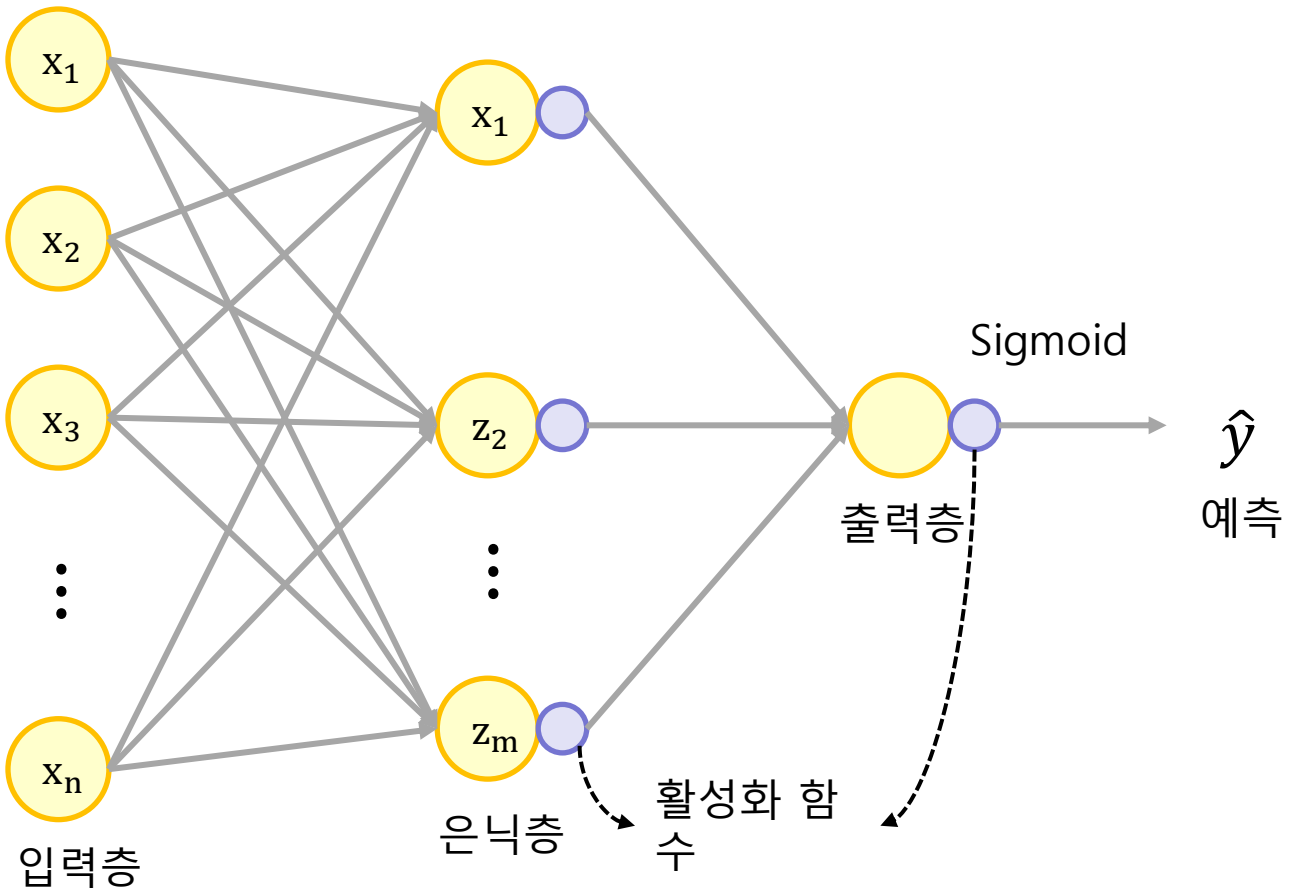
미니배치 경사 하강법 이란?

배치 경사 하강법과 비슷하지만 에포크마다 전체 데이터를 사용하는 것이 아니라 조금씩 나누어 계산을 수행하고 그레디언트를 구하여 가중치를 업데이트 한다.

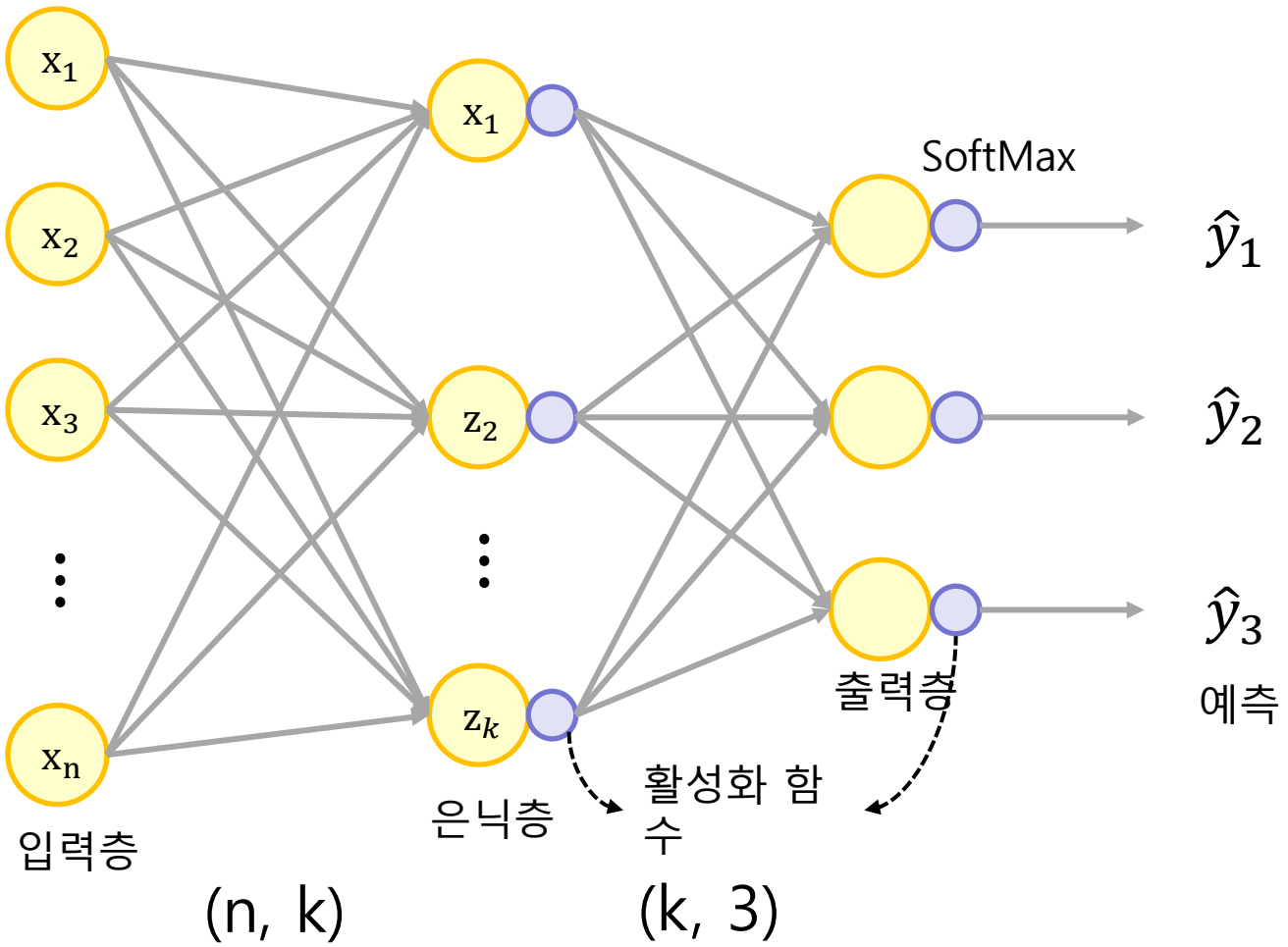
가중치 업데이트 방법

- 작게 나눈 미니 배치만큼 가중치를 업데이트 한다.
- 미니 배치의 크기는 보통 16,32,64등 2의 배수를 사용한다.
- 미니 배치의 크기가 1이면 확률적 경사 하강법이 된다.
- 미니 배치의 크기가 작으면 확률적 경사 하강법 처럼 작동하고 크면 배치 경사 하강법 처럼 작동한다.
- 미니 배치의 크기도 하이퍼파라미터이고 튜닝의 대상이다.

다중 분류 신경망



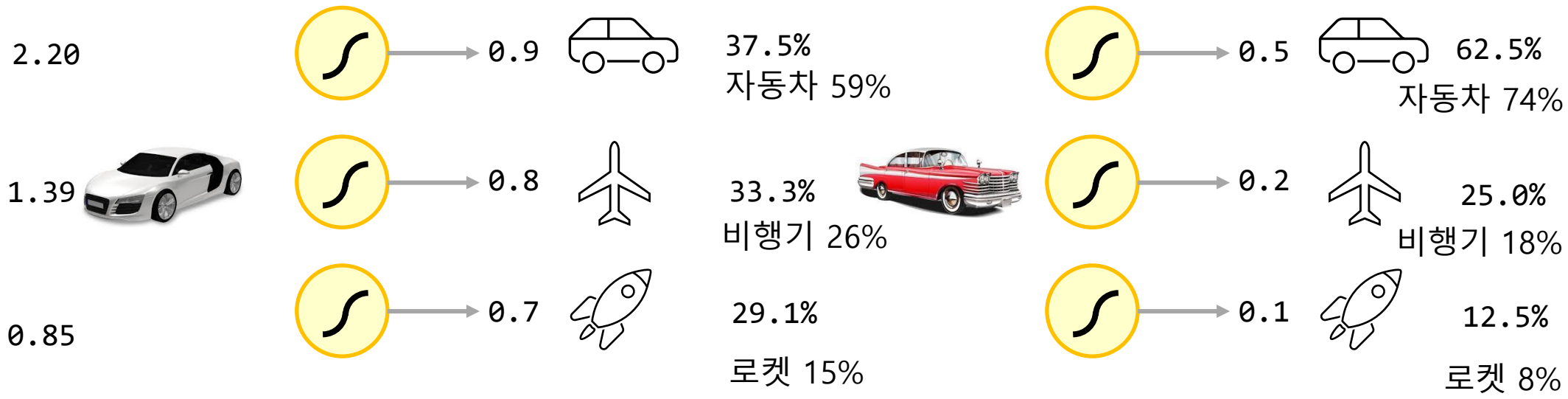
다중 분류 신경망


x_1
x_2
x_3
x_n
입력층
(n, k)
x_1
z_2
z_k
은닉층
활성화 함수
(k, 3)
SoftMax
출력층
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_3$
예측

다중 분류 신경망

활성화 출력의 합이 1이 아니면 비교하기 어렵다.

소프트맥스 함수를 적용해 출력 강도를 정규화 한다.



소프트맥스 함수를 적용해 출력 강도를 정규화 한다.

$$\frac{e^{z_i}}{e^{z_1} + e^{z_2} + e^{z_3}}$$

z_1 → $\frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3}}$

z_2 → $\frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3}}$

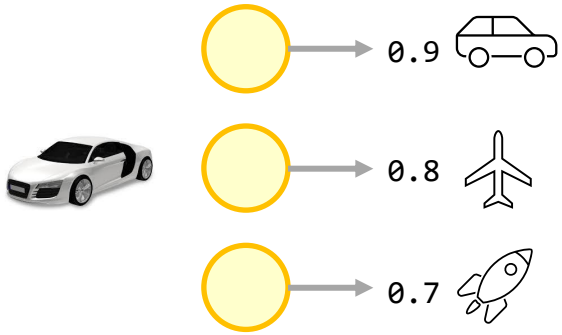
z_3 → $\frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3}}$

출력층

소프트맥스 함수를 적용해 출력 강도를 정규화 한다.

시그 모이드 함수를 z에 대해 정리하면

$$\hat{y} = \frac{1}{1 + e^{-z}} \Rightarrow z = -\log\left(\frac{1}{\hat{y}} - 1\right)$$



$$z_1 = -\log\left(\frac{1}{0.9} - 1\right) = 2.20 \qquad z_2 = -\log\left(\frac{1}{0.8} - 1\right) = 1.39 \qquad z_2 = -\log\left(\frac{1}{0.7} - 1\right) = 0.85$$

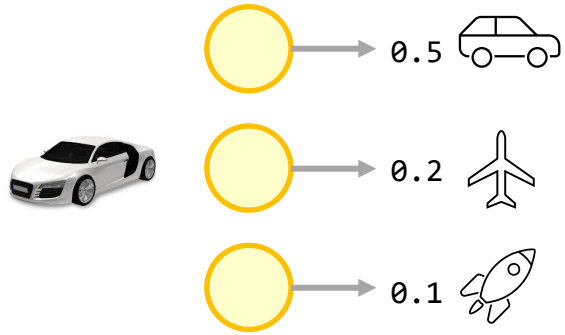
$$\hat{y} = \frac{e^{2.20}}{e^{2.20}+e^{1.39}+e^{0.85}} = 0.59 \qquad \hat{y} = \frac{e^{1.39}}{e^{2.20}+e^{1.39}+e^{0.85}} = 0.26 \qquad \hat{y} = \frac{e^{0.85}}{e^{2.20}+e^{1.39}+e^{0.85}} = 0.15$$

자동차 59%　　비행기 26%　　로켓 15%

소프트맥스 함수를 적용해 출력 강도를 정규화 한다.

시그 모이드 함수를 z에 대해 정리하면

$$\hat{y} = \frac{1}{1 + e^{-z}} \Rightarrow z = -\log\left(\frac{1}{\hat{y}} - 1\right)$$



$$z_1 = -\log\left(\frac{1}{0.5} - 1\right) = 0.00 \qquad z_2 = -\log\left(\frac{1}{0.2} - 1\right) = -1.39 \qquad z_2 = -\log\left(\frac{1}{0.1} - 1\right) = -2.20$$

$$\hat{y} = \frac{e^{0.00}}{e^{0.00} + e^{-1.39} + e^{-2.20}} = 0.74 \qquad \hat{y} = \frac{e^{-1.39}}{e^{0.00} + e^{-1.39} + e^{-2.20}} = 0.18 \qquad \hat{y} = \frac{e^{-2.20}}{e^{0.00} + e^{-1.39} + e^{-2.20}} = 0.08$$

자동차 74%　　비행기 18%　　로켓 8%

크로스 엔트로피 손실 함수

$$L = -\sum_{c=1}^{c} y_c \log(a_c) = -(y_1 \log(a_1) + y_2 \log(a_2) + \cdots + y_c \log(a_c))$$

크로스 엔트로피 손실 함수 미분

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1} + \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_1} + \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_1}$$

크로스 엔트로피 손실 함수 미분

$$\frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial a_1}\frac{\partial a_1}{\partial z_1} + \frac{\partial L}{\partial a_2}\frac{\partial a_2}{\partial z_1} + \frac{\partial L}{\partial a_3}\frac{\partial a_3}{\partial z_1}$$

$$\frac{\partial L}{\partial a_1} = -\frac{y_1}{a_1} \qquad \frac{\partial L}{\partial a_2} = -\frac{y_2}{a_2} \qquad \frac{\partial L}{\partial a_3} = -\frac{y_3}{a_3}$$

$$\frac{\partial L}{\partial z_1} = \left(-\frac{y_1}{a_1}\right)\frac{\partial a_1}{\partial z_1} + \left(-\frac{y_2}{a_2}\right)\frac{\partial a_2}{\partial z_1} + \left(-\frac{y_3}{a_3}\right)\frac{\partial a_3}{\partial z_1}$$

크로스 엔트로피 손실 함수 미분

$$\frac{\partial \mathrm{L}}{\partial \mathrm{z}_1} = \left(-\frac{\mathrm{y}_1}{\mathrm{a}_1}\right)\frac{\partial \mathrm{a}_1}{\partial \mathrm{z}_1} + \left(-\frac{\mathrm{y}_2}{\mathrm{a}_2}\right)\frac{\partial \mathrm{a}_2}{\partial \mathrm{z}_1} + \left(-\frac{\mathrm{y}_3}{\mathrm{a}_3}\right)\frac{\partial \mathrm{a}_3}{\partial \mathrm{z}_1}$$

$$\frac{\partial \mathrm{a}_1}{\partial \mathrm{z}_1} = \mathrm{a}_1(1-\mathrm{a}_1) \qquad \frac{\partial \mathrm{a}_2}{\partial \mathrm{z}_1} = -\mathrm{a}_2\mathrm{a}_1 \qquad \frac{\partial \mathrm{a}_3}{\partial \mathrm{z}_1} = -\mathrm{a}_3\mathrm{a}_1$$

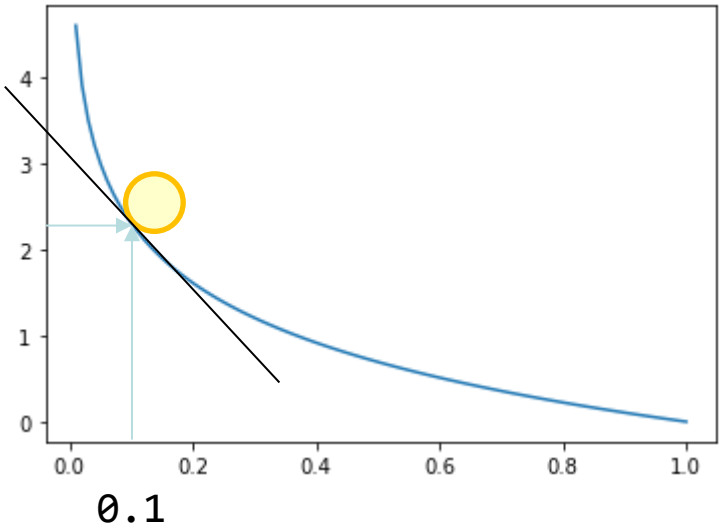
$$\frac{\partial \mathrm{L}}{\partial \mathrm{z}_1} = \left(-\frac{\mathrm{y}_1}{\mathrm{a}_1}\right)\mathrm{a}_1(1-\mathrm{a}_1) + \left(-\frac{\mathrm{y}_2}{\mathrm{a}_2}\right)(-\mathrm{a}_2\mathrm{a}_1) + \left(-\frac{\mathrm{y}_3}{\mathrm{a}_3}\right)(-\mathrm{a}_3\mathrm{a}_1)$$
$$= (\mathrm{a}_1 - y_1)$$

$$\boldsymbol{\frac{\partial L}{\partial z} = (a - y)}$$

개 고양이 사슴

```
a = [0.7,  0.2,    0.1]
y = [  0,    0,      1]
```

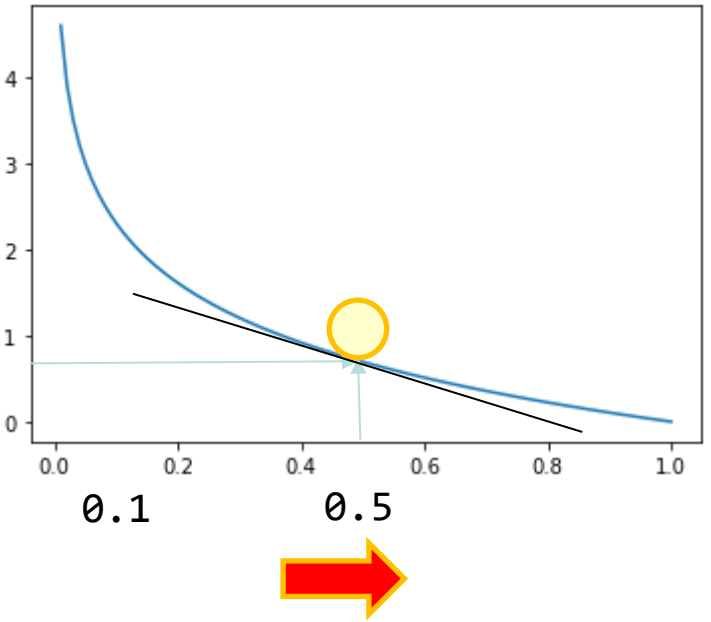
$$L = -\sum_{c=1}^{3} y_c \log(a_c) = -(y_1 \log(a_1) + y_2 \log(a_2) + y_3 \log(a_3))$$




```
err = (a-y)
err = (0.1-1) = -0.9
```

```
          개      고양이     사슴
a = [0.1,   0.4,      0.5]
y = [  0,     0,        1]
```

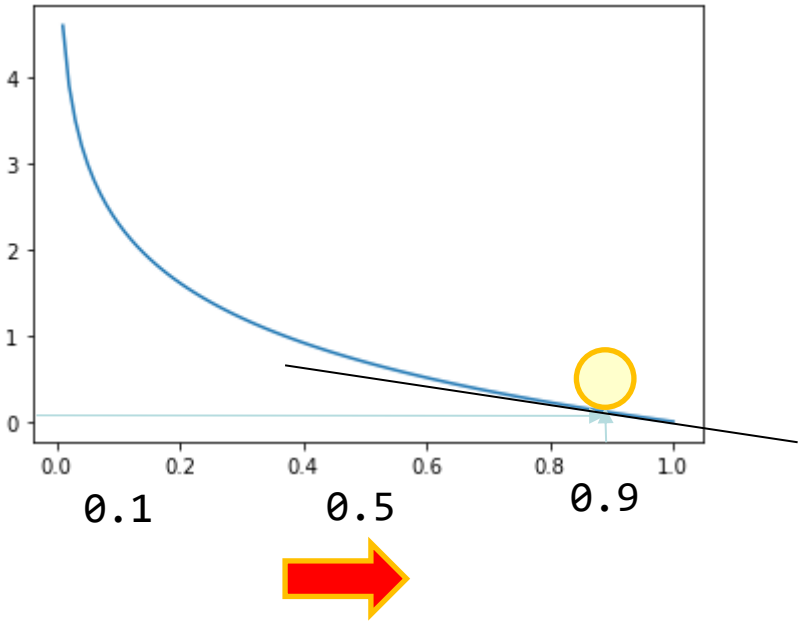
$$L = -\sum_{c=1}^{3} y_c \log(a_c) = -(y_1 \log(a_1) + y_2 \log(a_2) + y_3 \log(a_3))$$




```
err = (a-y)
err = (0.5-1) = -0.5
```

```
          개      고양이     사슴
a = [0.0,   0.1,      0.9]
y = [  0,     0,        1]
```

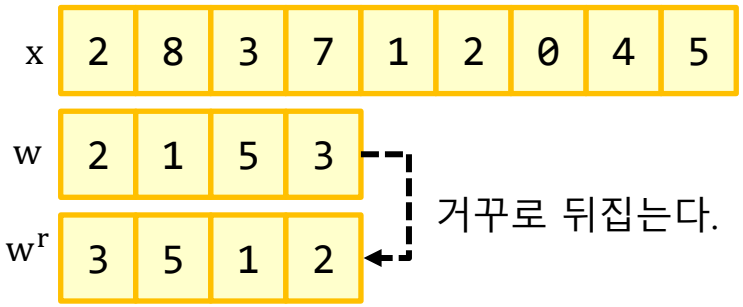
$$L = -\sum_{c=1}^{3} y_c \log(a_c) = -(y_1 \log(a_1) + y_2 \log(a_2) + y_3 \log(a_3))$$



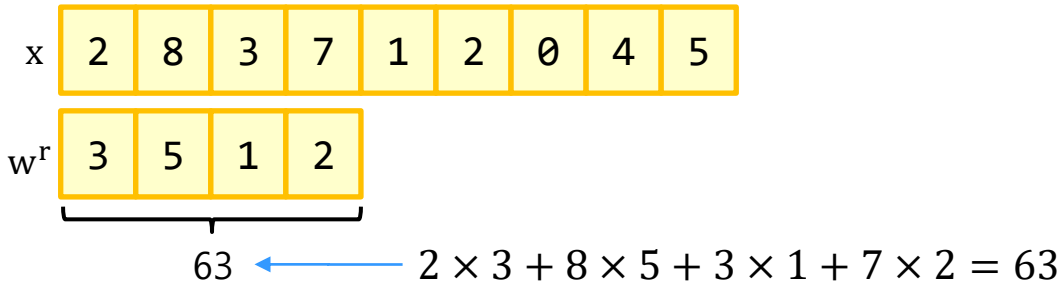

```
err = (a-y)
err = (0.9-1) = -0.1
```

합성곱 연산

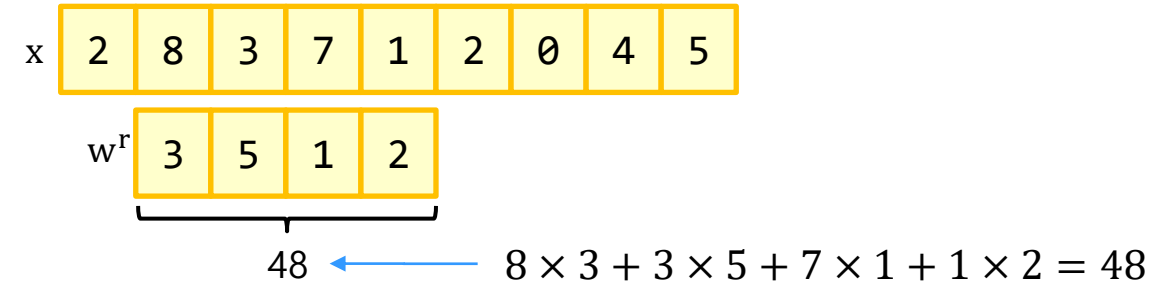
배열 하나 선택해 뒤집기



첫 번째 합성곱

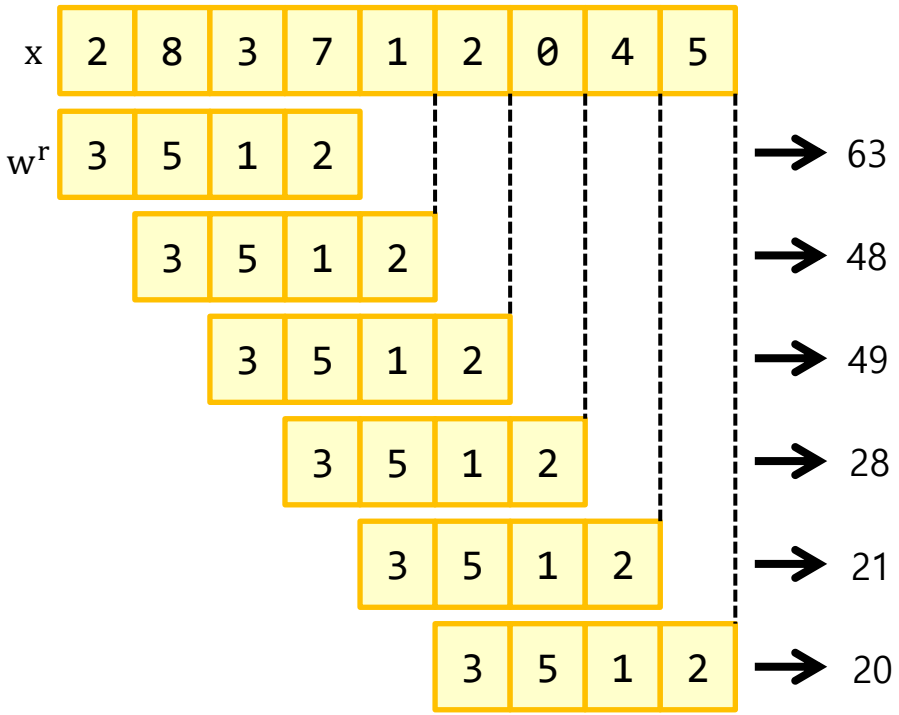


두 번째 합성곱



합성곱 연산

전체 합성곱



합성곱 연산

합성곱 구현

```
import numpy as np
x = np.array([2, 8, 3, 7, 1, 2, 0, 4, 5])
w = np.array([2, 1, 5, 3])
```

flip() 함수를 이용한 배열 뒤집기

```
w_r = np.flip(w)
print(w_r)
```

[3 5 1 2]

넘파이의 점 곱으로 합성곱 연산

```
for i in range(6):
    print(np.dot(x[i:i+4], w_r.reshape(-1,1)))
```

[63]
[48]
[49]
[28]
[21]
[20]

$$\begin{bmatrix} 2 & 8 & 3 & 7 \\ 8 & 3 & 7 & 1 \\ 3 & 7 & 1 & 2 \\ 7 & 1 & 2 & 0 \\ 1 & 2 & 0 & 4 \\ 2 & 0 & 4 & 5 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 5 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 63 \\ 48 \\ 49 \\ 28 \\ 21 \\ 20 \end{bmatrix}$$

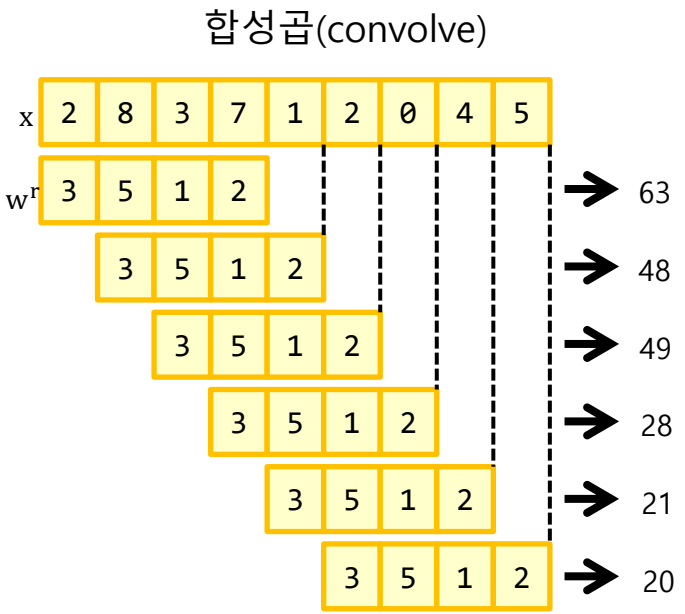
합성곱 연산

싸이파이로 합성곱 수행

```
from scipy.signal import convolve
convolve(x, w, mode='valid')
```

array([63, 48, 49, 28, 21, 20])

합성곱 신경망은 진짜 합성곱을 사용하지 않는다.
합성곱 대신 교차상관을 사용한다.



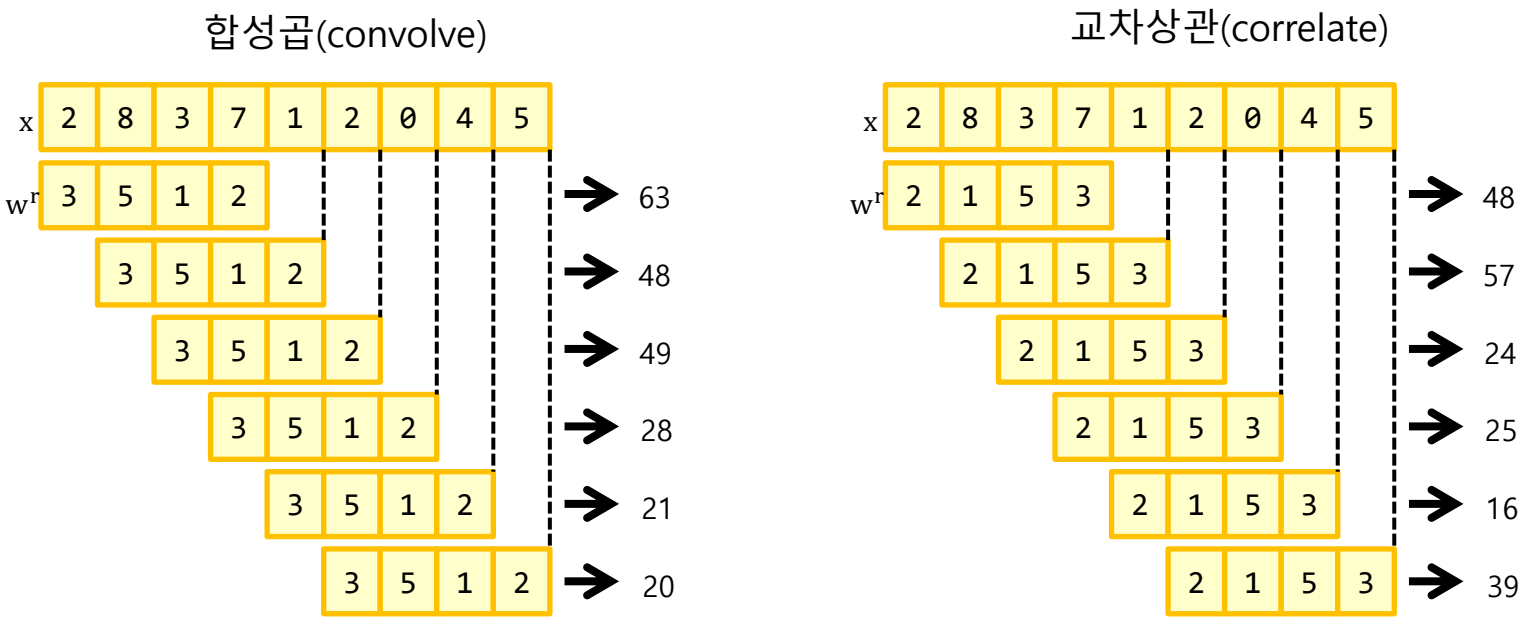
합성곱 연산

싸이파이로 교차상관 수행

```
from scipy.signal import correlate
correlate(x, w, mode='valid')
```

```
array([48, 57, 24, 25, 16, 39])
```

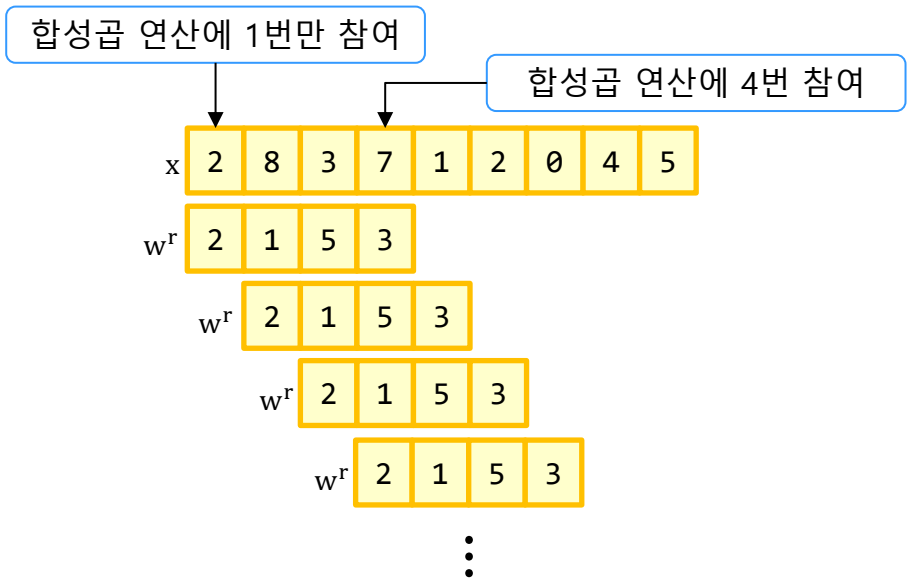
합성곱 신경망은 진짜 합성곱을 사용하지 않는다.
합성곱 대신 교차상관을 사용한다.



합성곱 연산

패딩과 스트라이드 이해

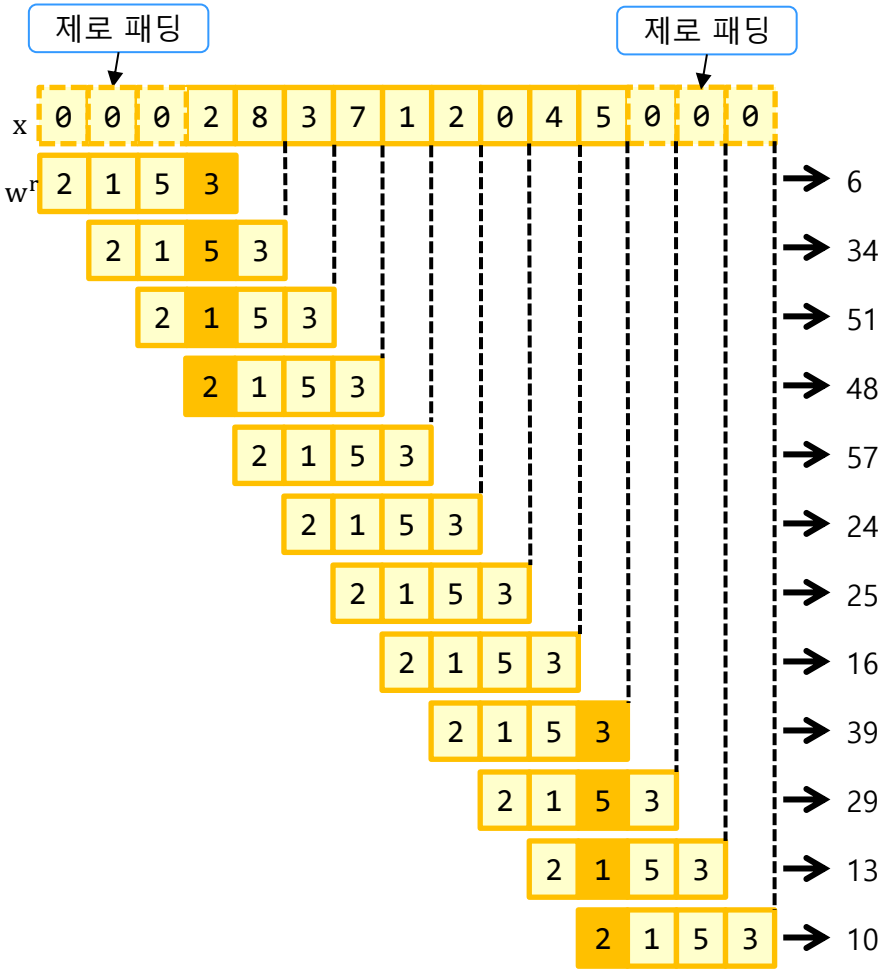
밸리드 패딩은 원본 배열의 원소가 합성곱 연산에 참여하는 정도가 다르다.



합성곱 연산

패딩과 스트라이드 이해

풀 패딩은 원본 배열의 원소의 연산 참여도를 동일하게 만든다.




```
correlate(x, w, mode='full')
```

array([6, 34, 51, 48, 57, 24, 25, 16, 39, 29, 13, 10])

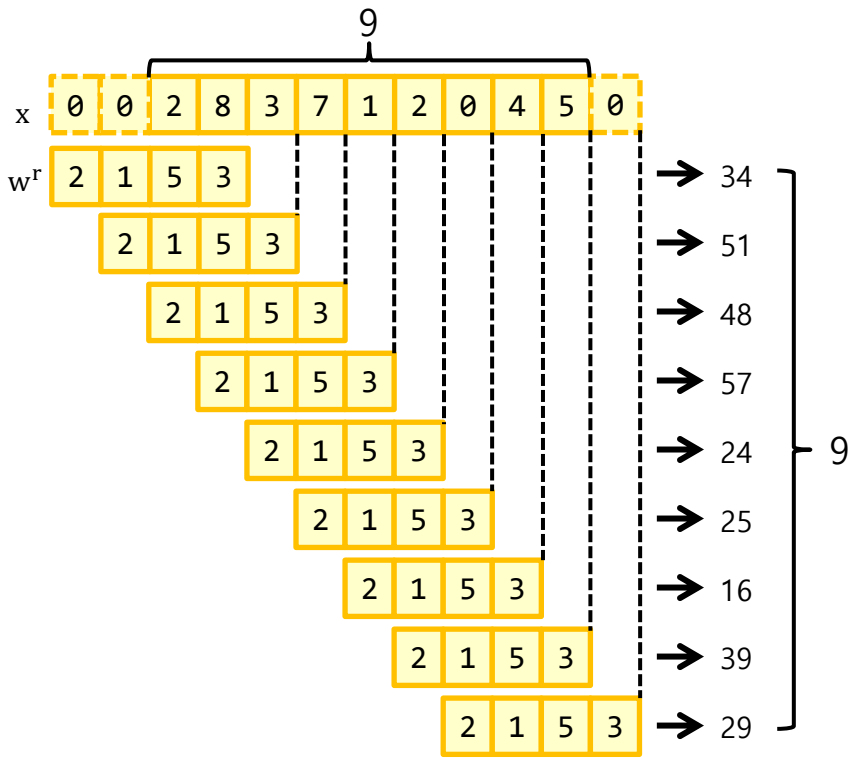
합성곱 연산

패딩과 스트라이드 이해

세임 패딩은 출력 배열의 길이를 원본 배열의 원소의 길이와 동일하게 만든다.

```
correlate(x, w, mode='same')
```

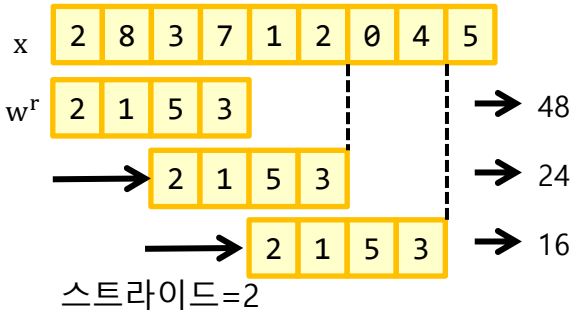
array([34, 51, 48, 57, 24, 25, 16, 39, 29])



합성곱 연산

패딩과 스트라이드 이해

스트라이드는 미끄러지는 간격을 조정한다.




```
correlate(x, w, mode='same')
```

array([34, 51, 48, 57, 24, 25, 16, 39, 29])

합성곱 연산

2차원 배열에서 합성곱 수행 (mode='valid')

x

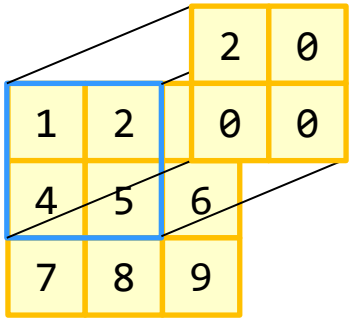
1	2	3
4	5	6
7	8	9

w

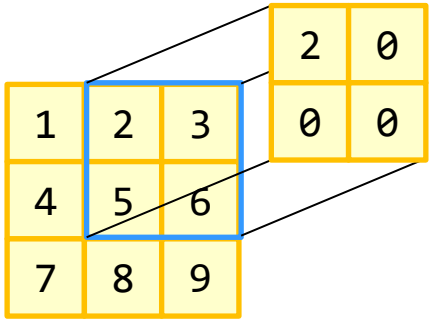
2	0
0	0

```
x = np.array([[1, 2, 3],
              [4, 5, 6],
              [7, 8, 9]])
w = np.array([[2, 0], [0, 0]])
from scipy.signal import correlate2d
correlate2d(x, w, mode='valid')
```

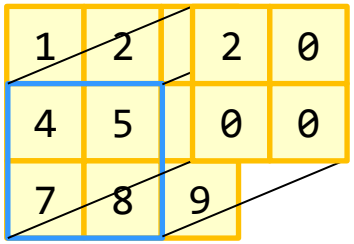
```
array([[ 2,  4],
       [ 8, 10]])
```



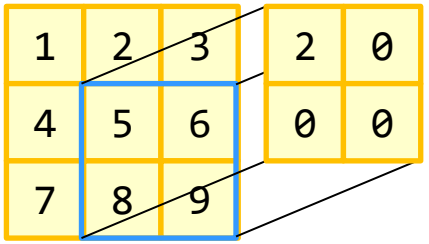
=> 2



=> 4



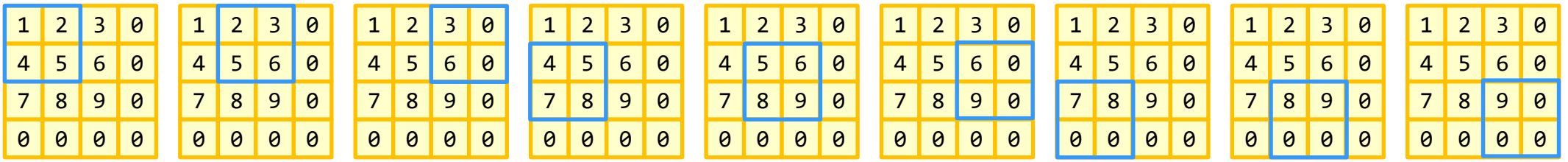
=> 8



=> 10

합성곱 연산

2차원 배열에서 same padding

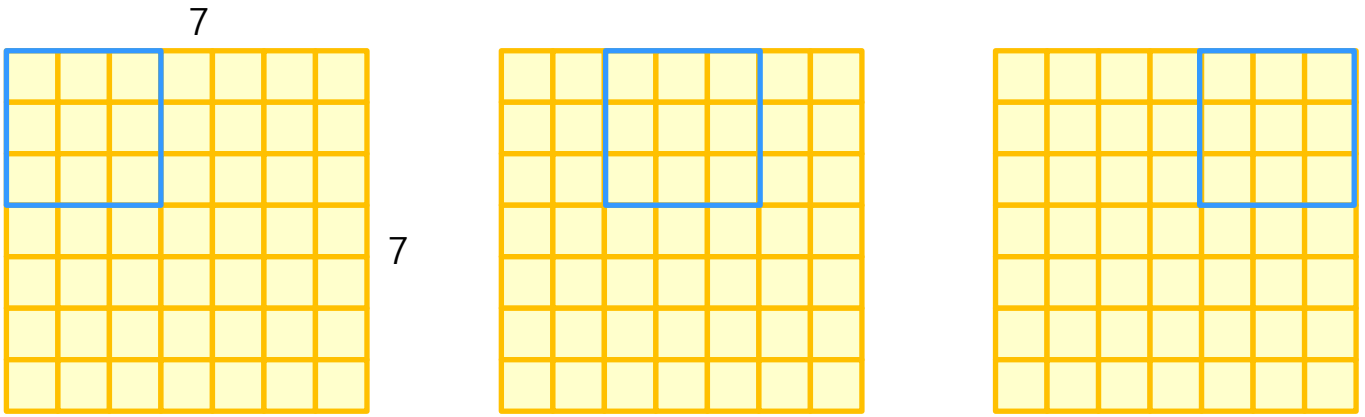



```
correlate2d(x, w, mode='same')
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12],
       [14, 16, 18]])
```

합성곱 연산

2차원 배열에서 스트라이드 이해



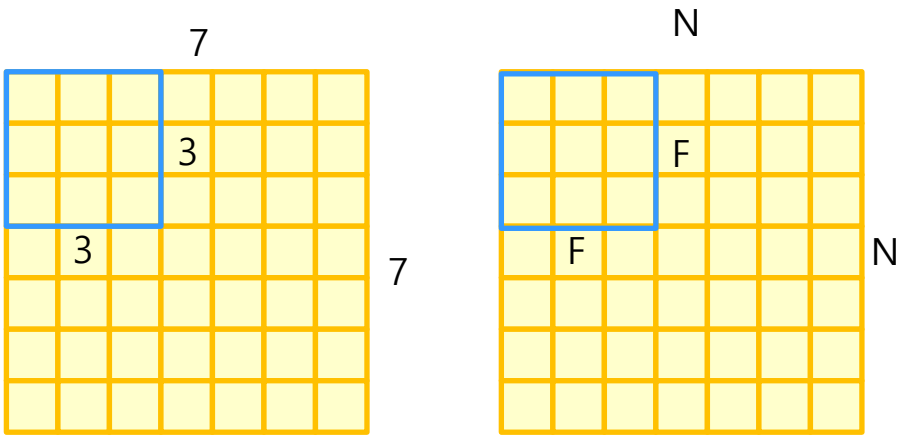
7x7 input
3x3 filter

stride = 2

=> 3x3 output

합성곱 연산

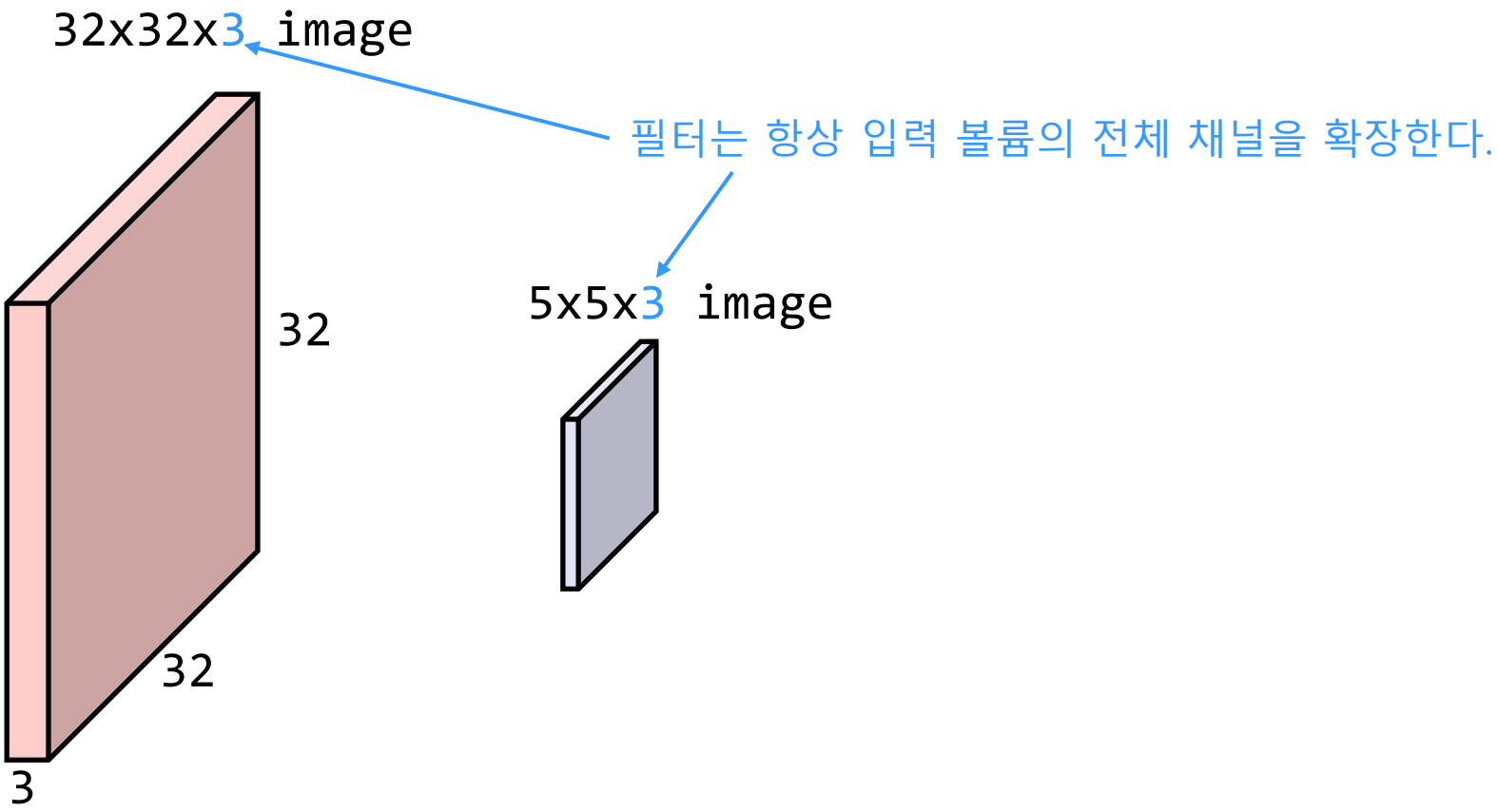
2차원 배열에서 스트라이드 이해



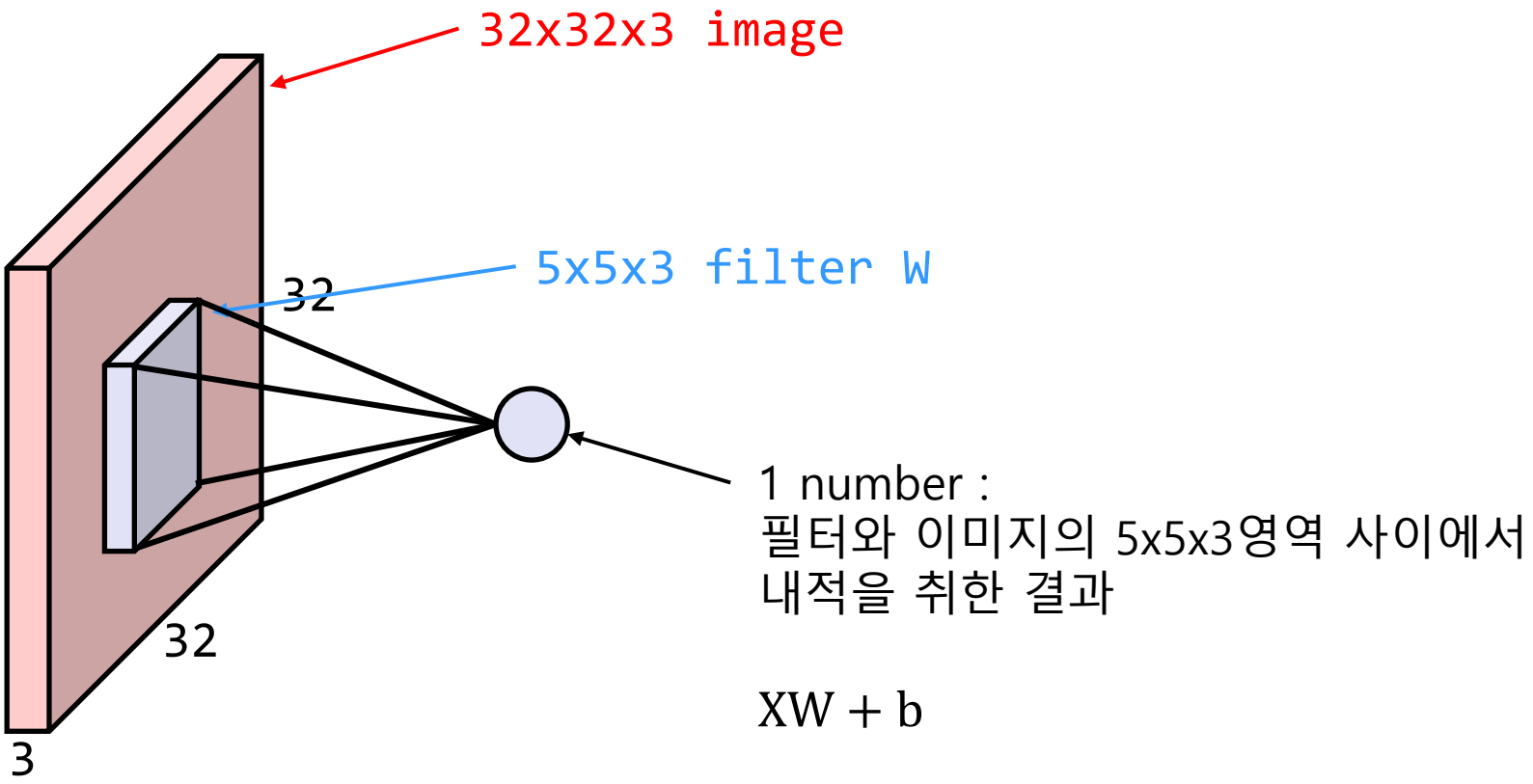
Output size :
(N - F) / stride + 1

예) N = 7, F = 3
stride 1 => (7-3)/1+1 = 5
stride 2 => (7-3)/2+1 = 3
stride 3 => (7-3)/3+1 = 2.33

Convolution Layer

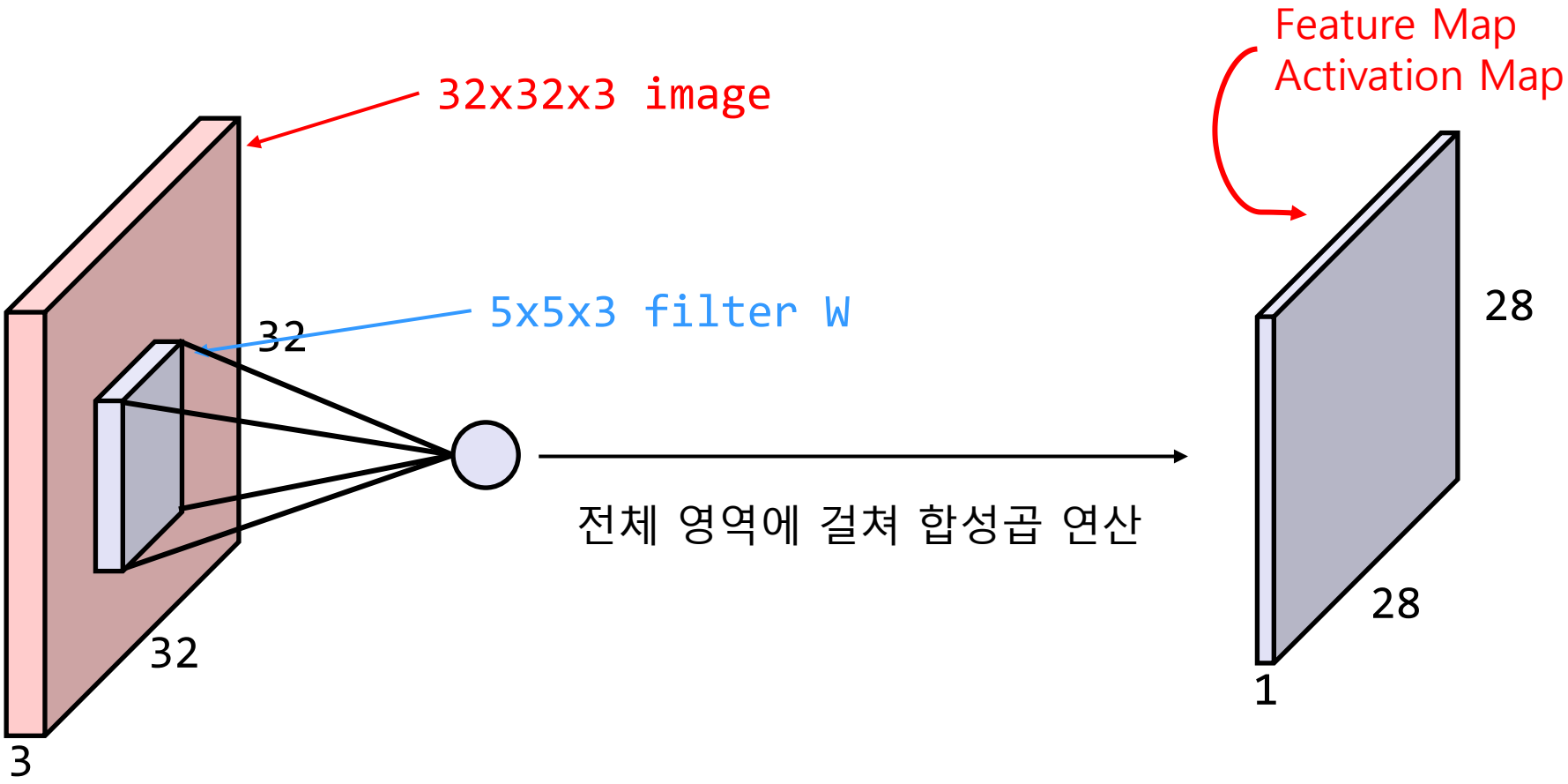


Convolution Layer

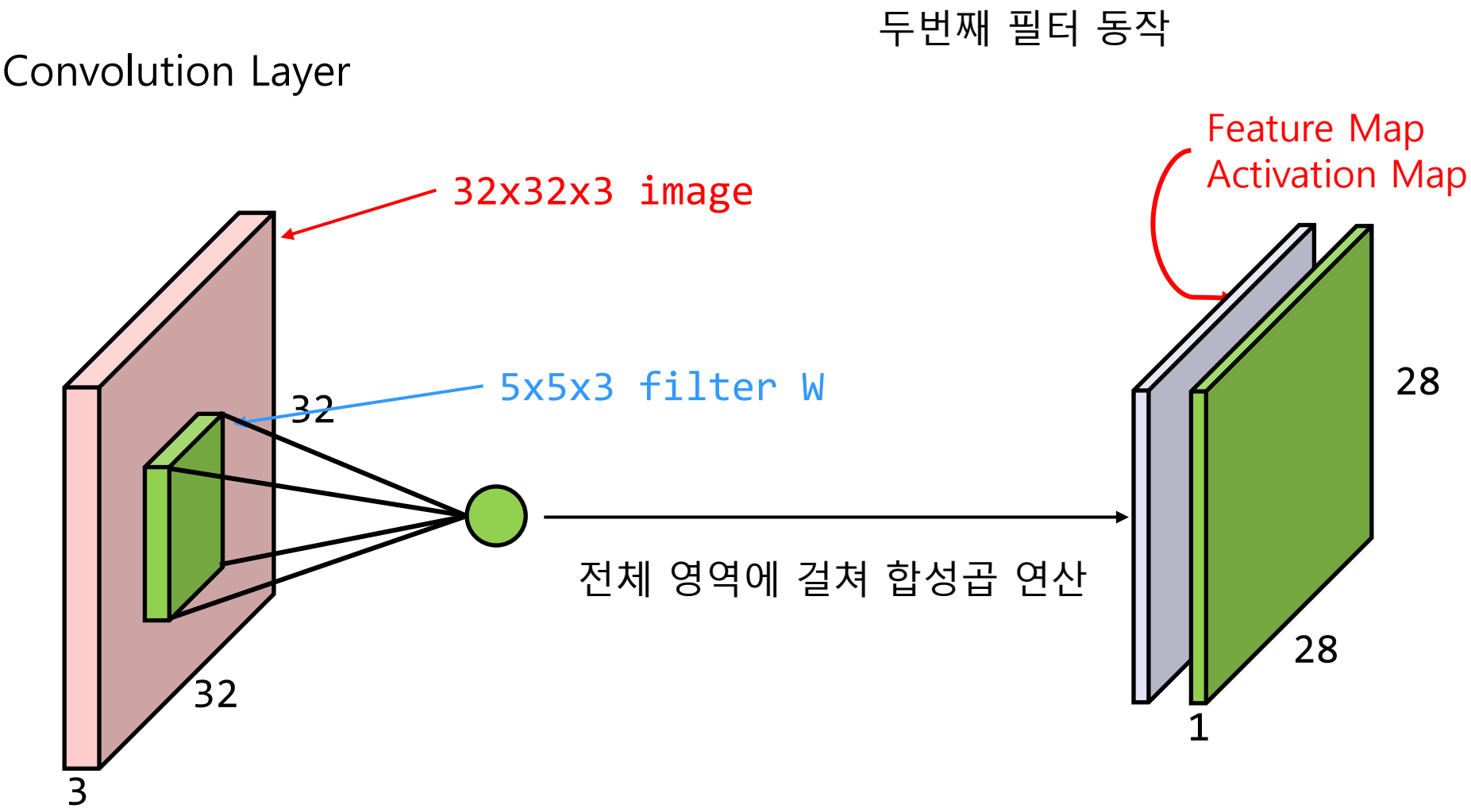


합성곱 연산

Convolution Layer



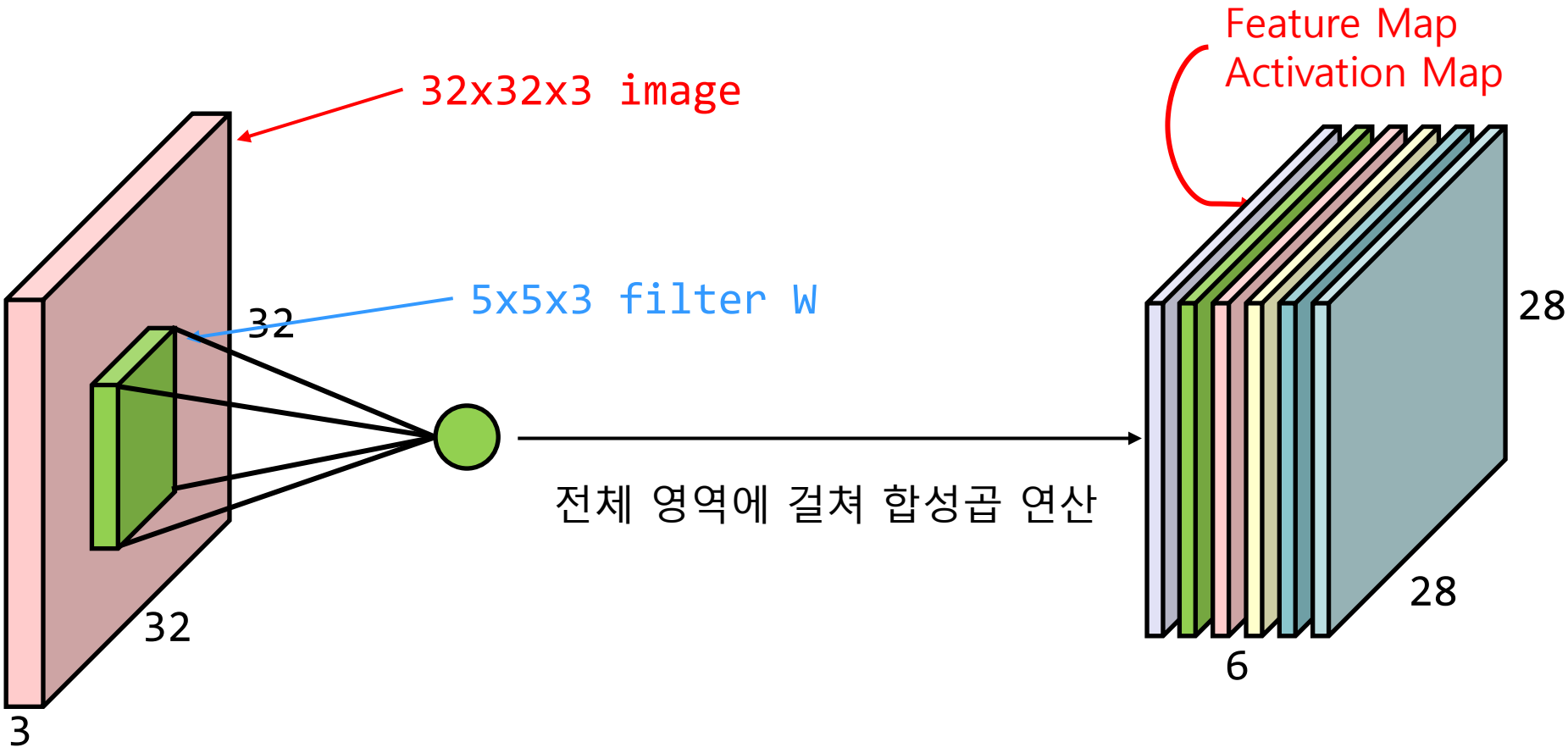
합성곱 연산


두번째 필터 동작
Convolution Layer
Feature Map
Activation Map
32x32x3 image
5x5x3 filter W
32
32
3
전체 영역에 걸쳐 합성곱 연산
28
28
1

합성곱 연산

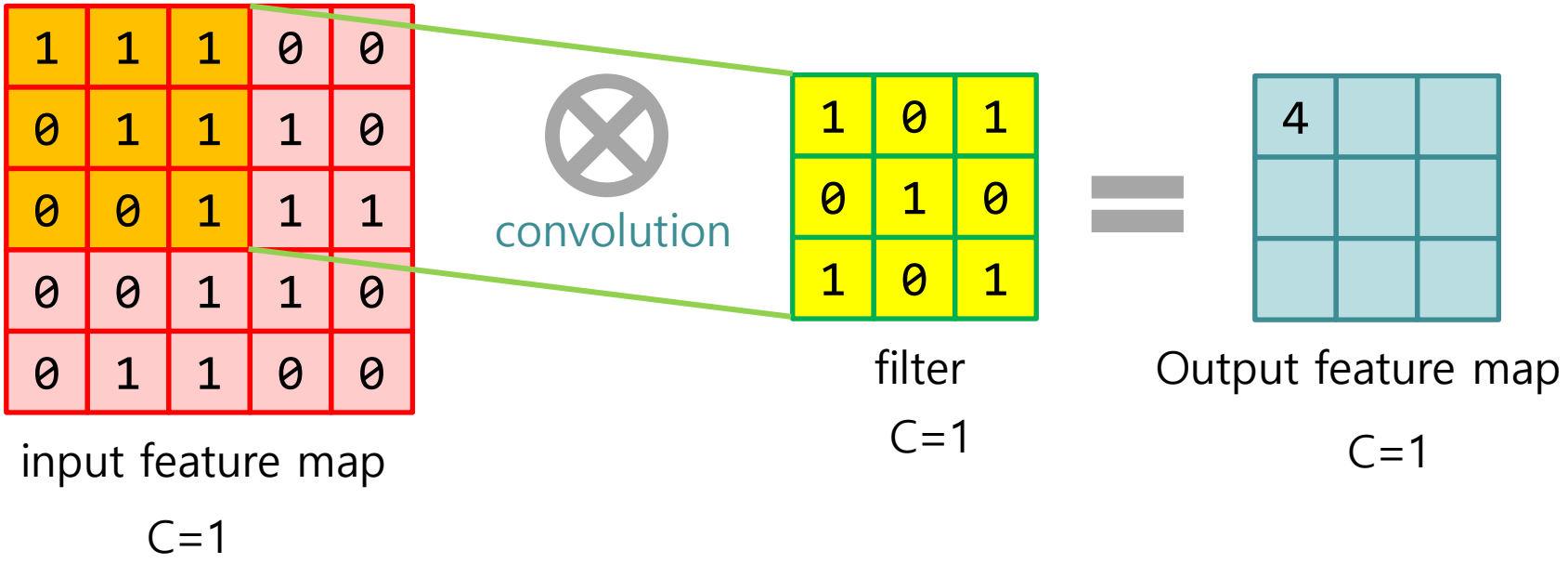
Convolution Layer

6개의 5x5필터가 있다면, 6개의 개별 feature map이 생성됨

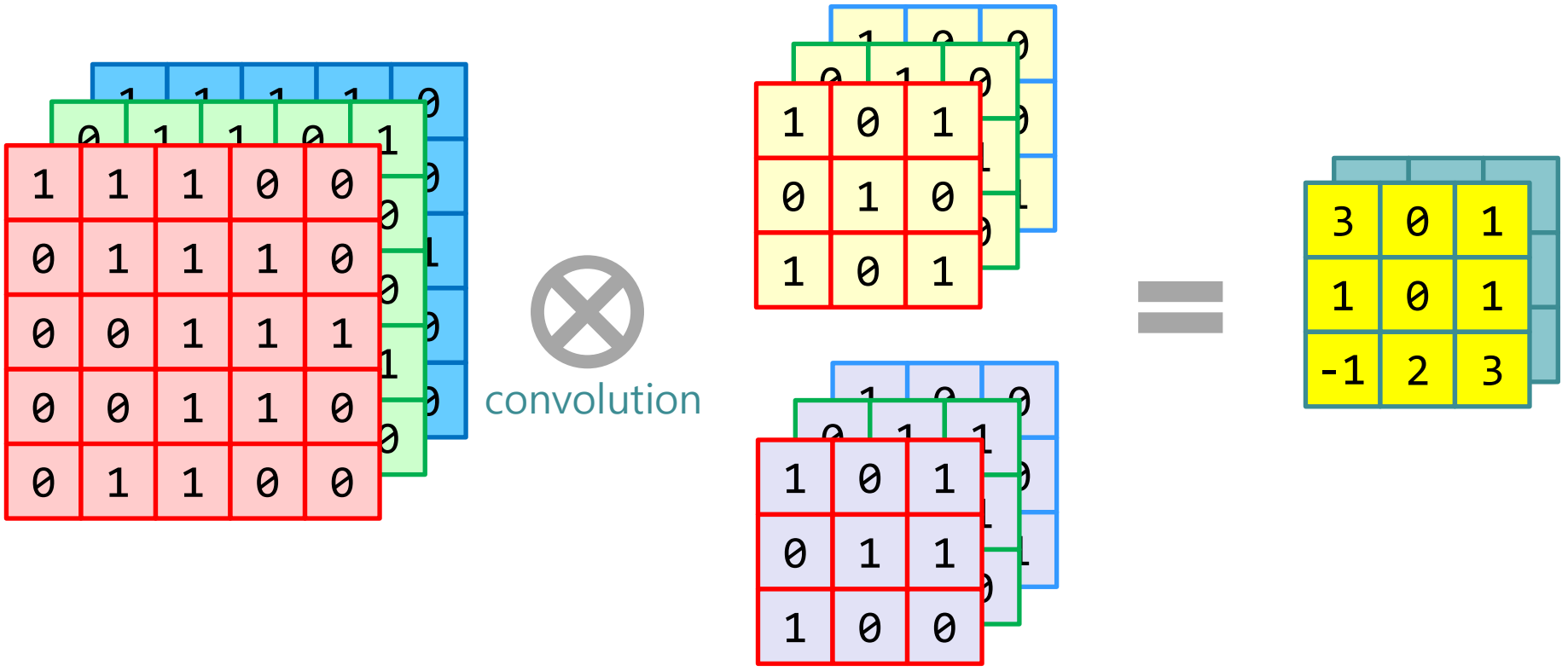


Convolution Layer - 계산

1x1 + 1x0 + 1x1 + 0x0 + 1x1 + 1x0 + 0x1 + 0x0 +1x1 = 4



Convolution Layer - 계산

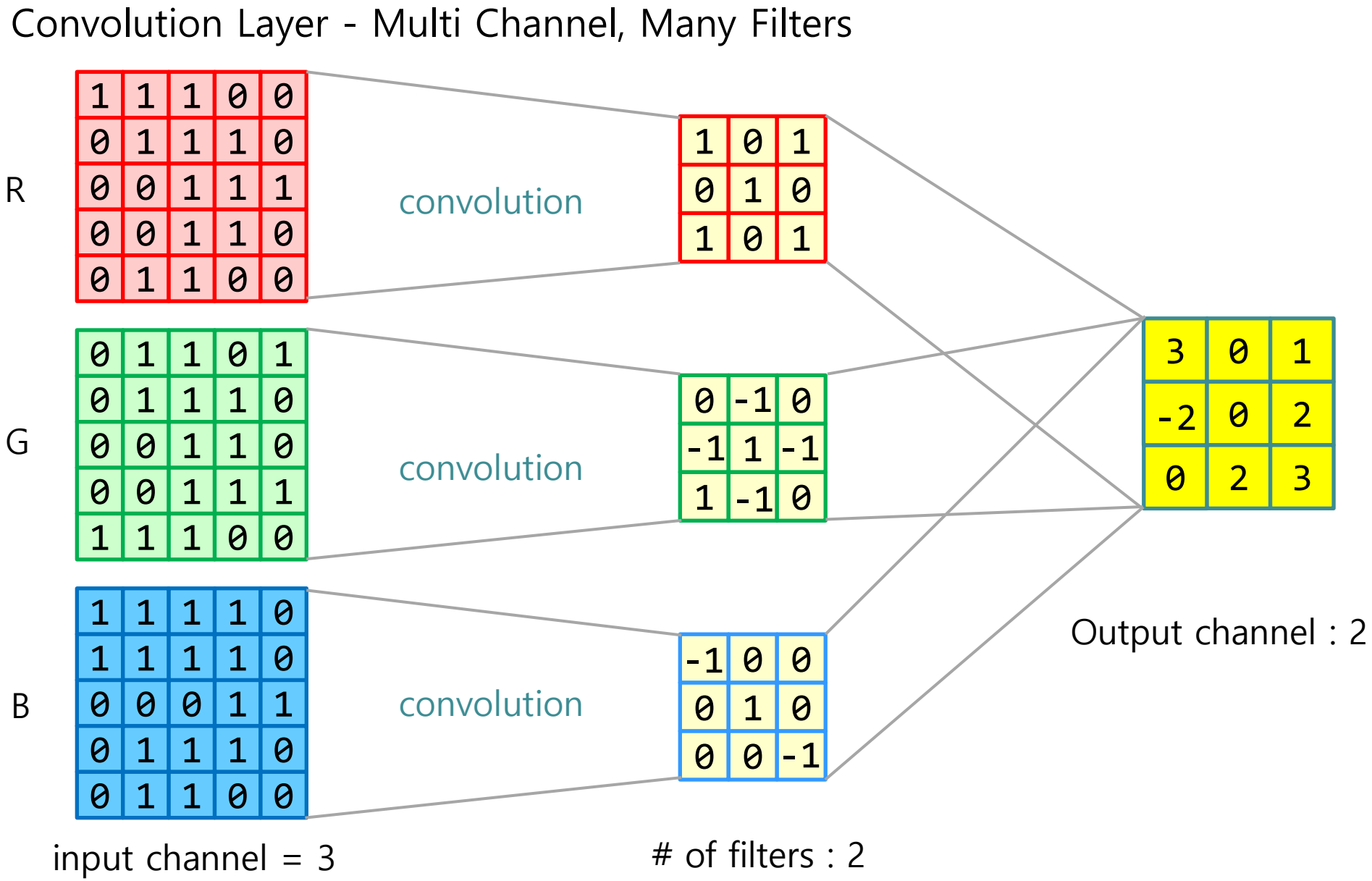


input channel = 3

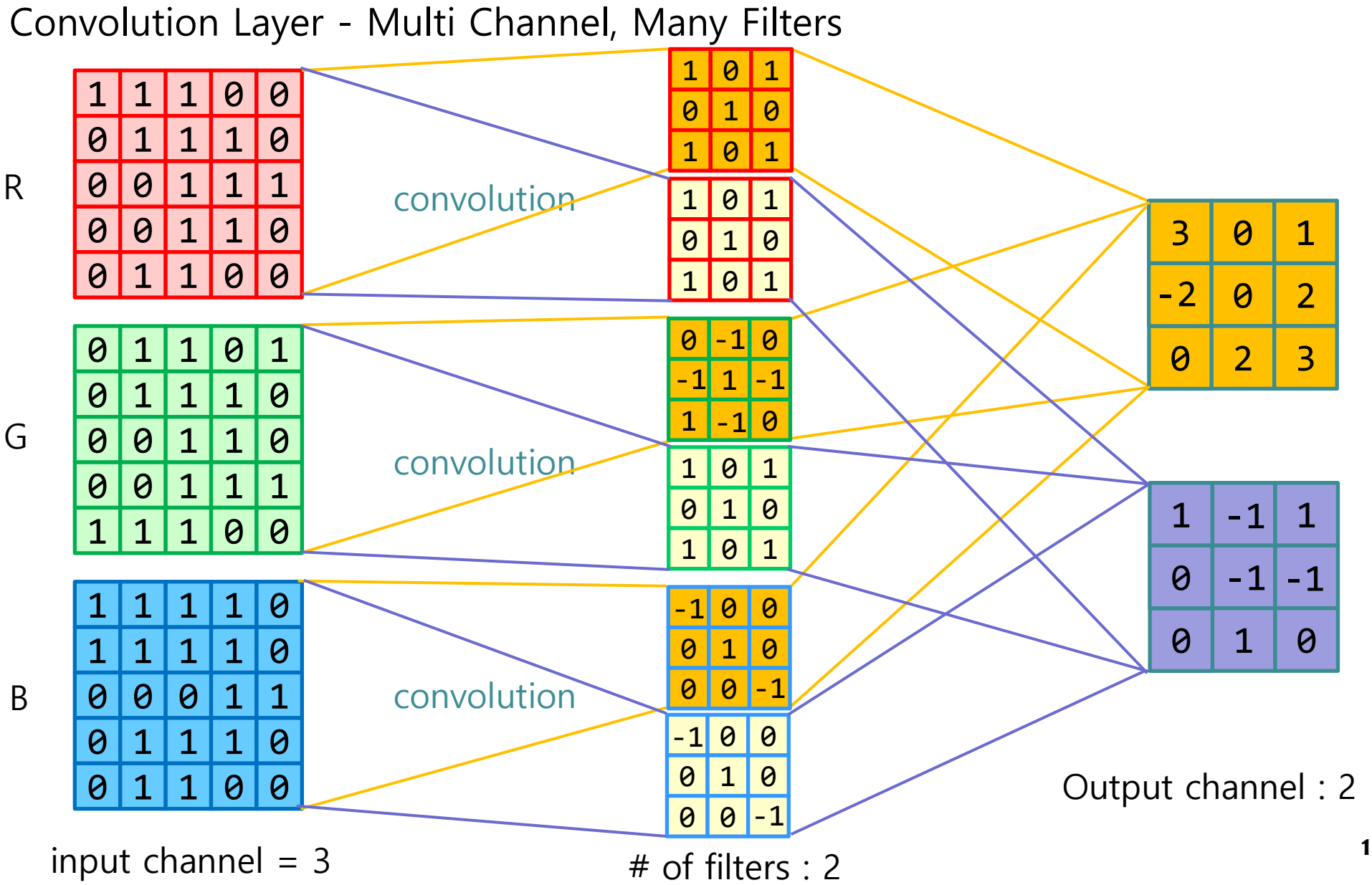
# of filters : 2

Output channel : 2

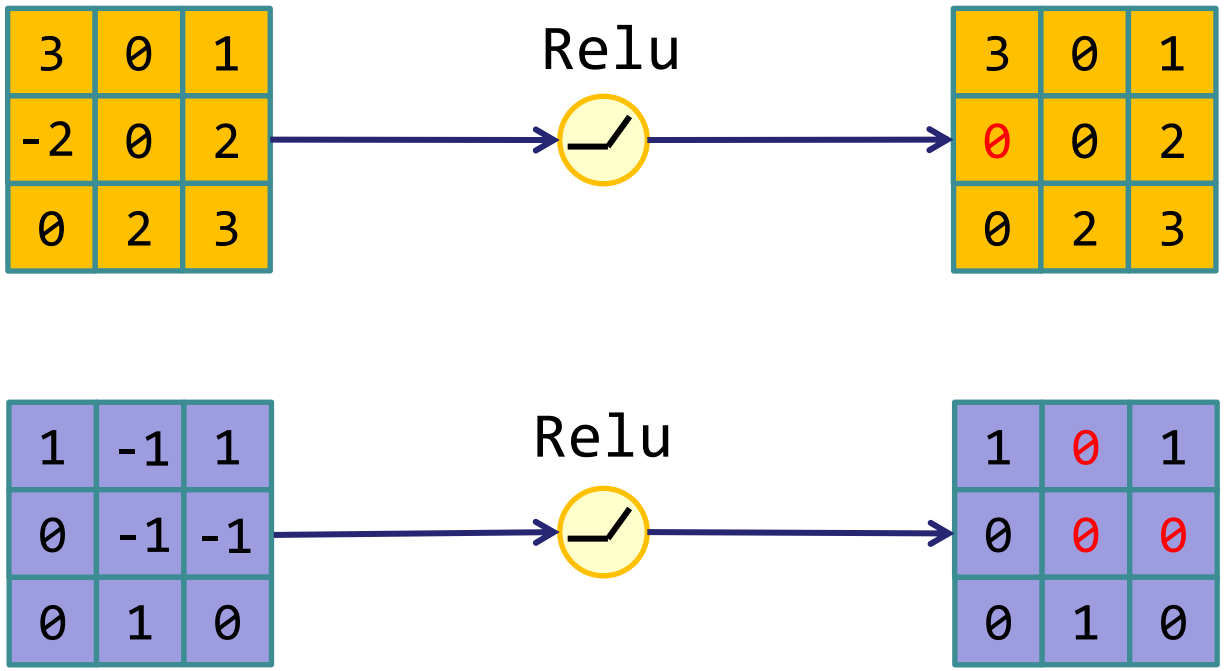
합성곱 연산


Convolution Layer - Multi Channel, Many Filters
R
G
B
convolution
convolution
convolution
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
0 1 1 0 1
0 1 1 1 0
0 0 1 1 0
0 0 1 1 1
1 1 1 0 0
1 1 1 1 0
1 1 1 1 0
0 0 0 1 1
0 1 1 1 0
0 1 1 0 0
1 0 1
0 1 0
1 0 1
0 -1 0
-1 1 -1
1 -1 0
-1 0 0
0 1 0
0 0 -1
3 0 1
-2 0 2
0 2 3
Output channel : 2
input channel = 3
# of filters : 2

합성곱 연산


Convolution Layer - Multi Channel, Many Filters
R
G
B
convolution
convolution
convolution
input channel = 3
# of filters : 2
Output channel : 2

Activation Function



합성곱 연산

tf.keras.layers.Conv2D

Arguments:

- **`filters`**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

- **`kernel_size`**: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

- **`strides`**: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.

- **`padding`**: one of `"valid"` or `"same"` (case-insensitive).

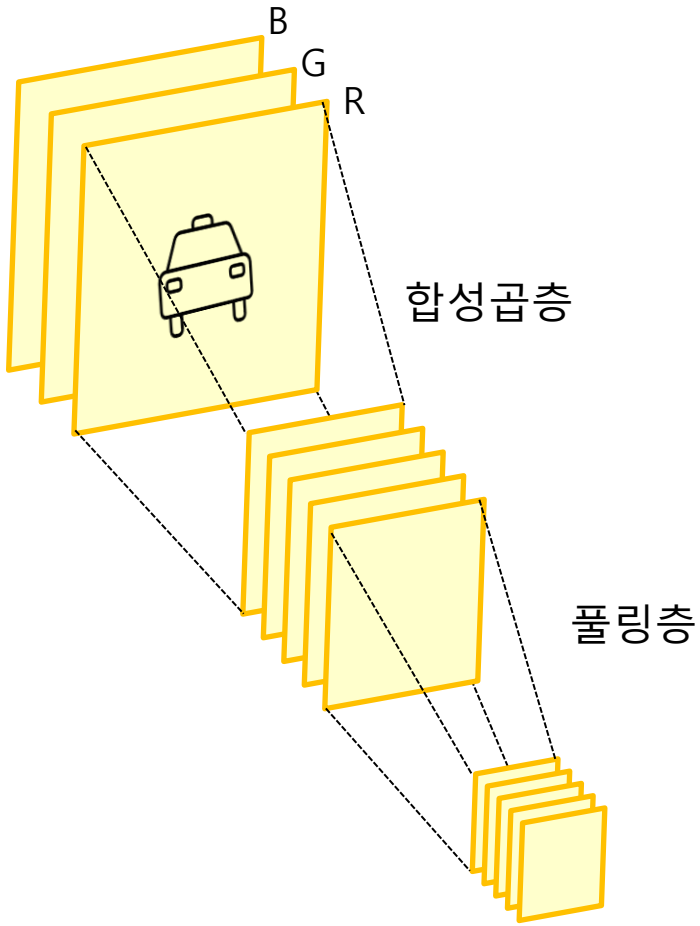
- **`data_format`**: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch_size, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch_size, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

tf.keras.layers.Conv2D

- **activation**: Activation function to use. If you don't specify anything, no activation is applied (see keras.activations).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix (see keras.initializers).
- **bias_initializer**: Initializer for the bias vector (see keras.initializers).
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix (see keras.regularizers).
- **bias_regularizer**: Regularizer function applied to the bias vector (see keras.regularizers).

풀링 연산

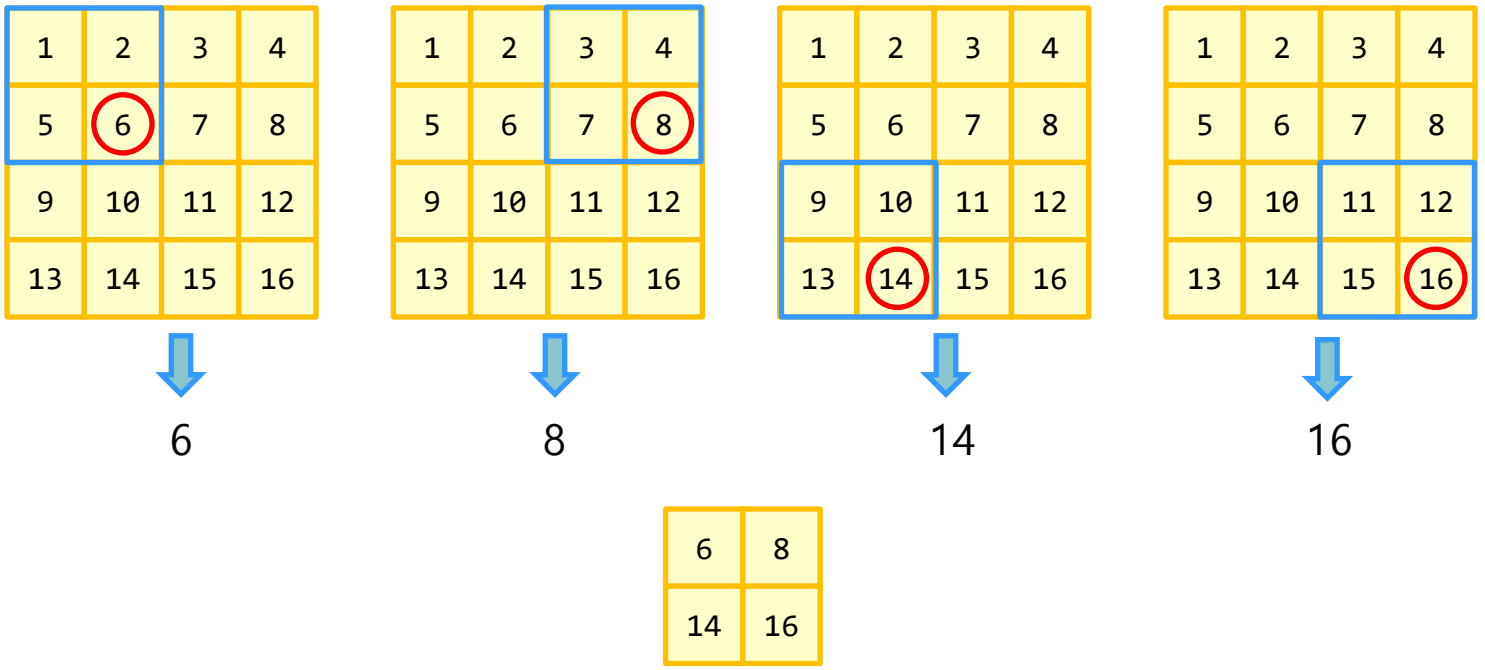
합성곱층과 풀링층을 거치면서 변환되는 과정



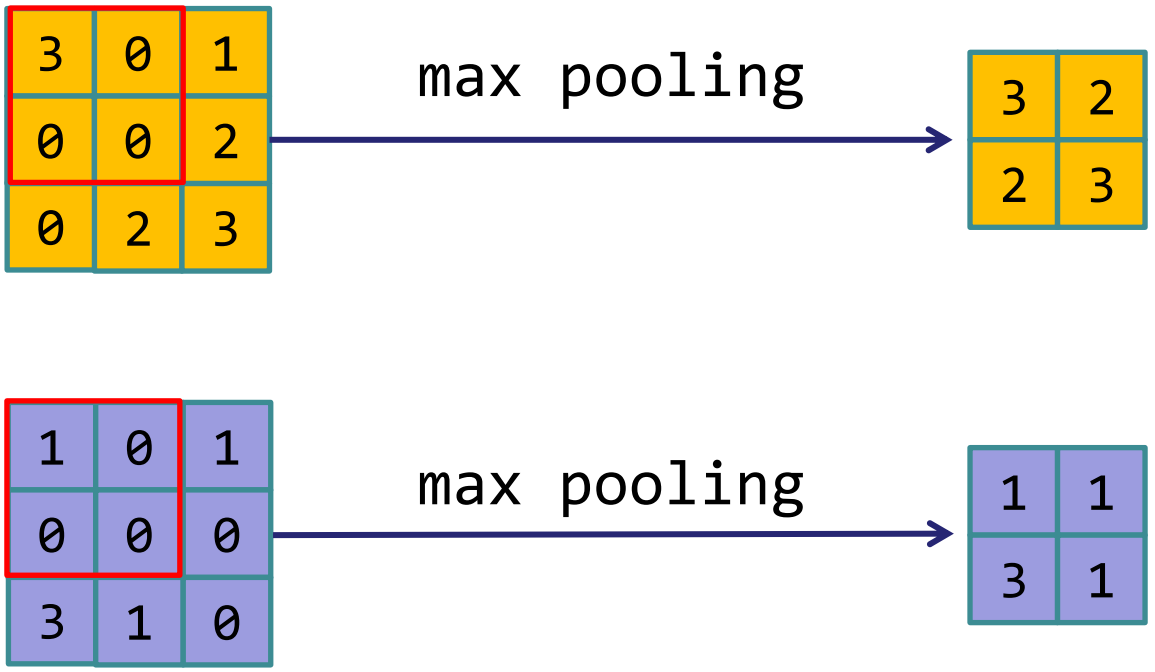
풀링 연산

풀링이란? 특성 맵을 스캔하며 최대값을 고르거나 평균값을 계산하는 것을 말함

최대 풀링

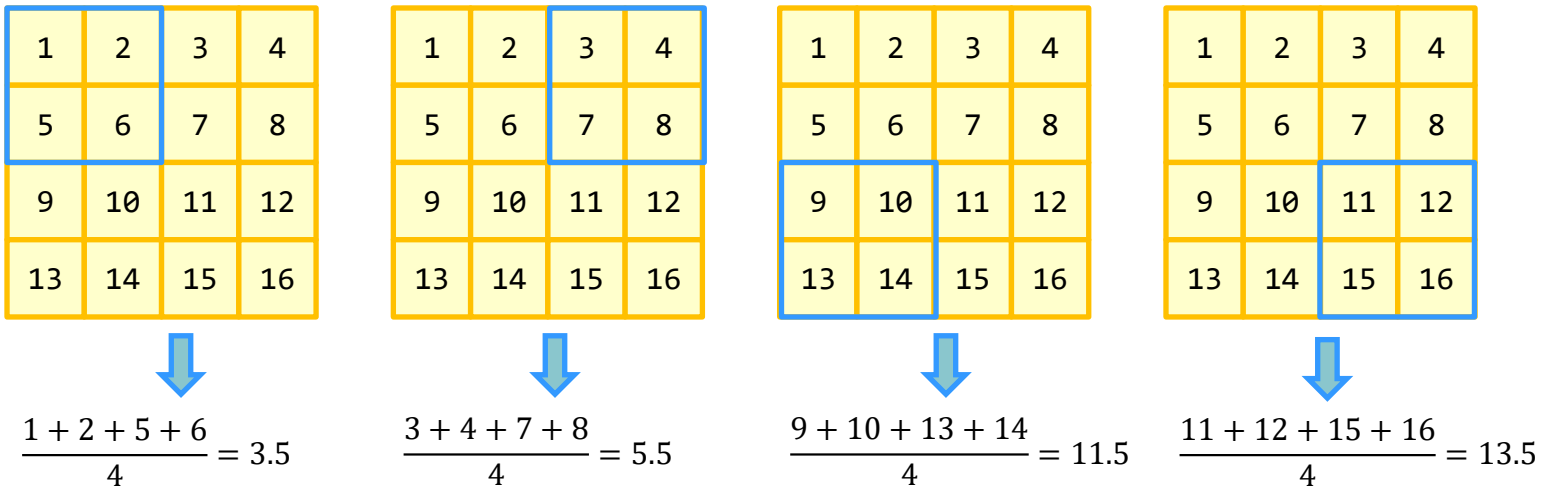


Pooling(max pooling, 2x2 filter, stride 1)



풀링 연산

Pooling(average pooling, 2x2 filter, stride 2)



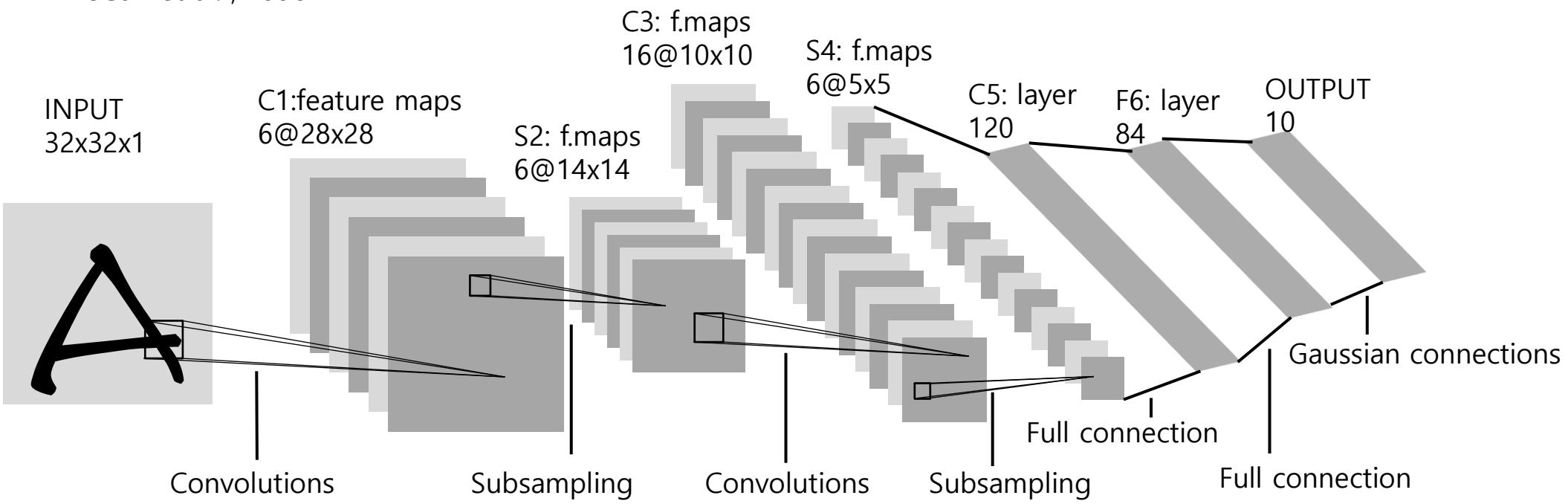
tf.keras.layers.MAXPool2D

- **`pool_size`**: integer or tuple of 2 integers, window size over which to take the maximum. `(2, 2)` will take the max value over a 2x2 pooling window. If only one integer is specified, the same window length will be used for both dimensions.
- **`strides`**: Integer, tuple of 2 integers, or None. Strides values. Specifies how far the pooling window moves for each pooling step. If None, it will default to `pool_size`.
- **`padding`**: One of `"valid"` or `"same"` (case-insensitive). "valid" adds no zero padding. "same" adds padding such that if the stride is 1, the output shape is the same as input shape.
- **`data_format`**: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".

전체 합성곱 구조

LeNet-5

LeCun et al. , 1998



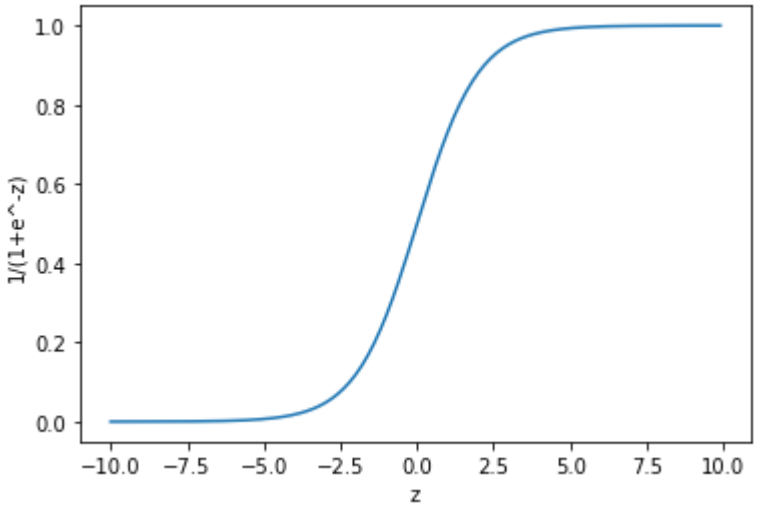
Conv filters 5x5, stride 1
Subsampling 2x2, stride 2

활성화 함수

활성함수는 네트워크에 비선형성(nonlinearity)을 추가하기 위해 사용됨
- 활성함수 없이 layer를 쌓은 네트워크는 1-layer 네트워크와 동일하기 때문에 활성 함수는 비성형함수로 불리기도 한다.
- 멀티레이어 퍼셉트론을 만들 때 활성함수를 사용하지 않으면 쓰나마나이다.

1. 시그모이드 함수 (Sigmoid Function)

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- 결과값이 [0,1] 사이로 제한됨
- 뇌의 뉴런과 유사하여 많이 쓰였음

활성화 함수

- 문제점

1) 그레디언트가 죽는 현상이 발생한다 (Gradient vanishing 문제)

gradient 0이 곱해 지니까 그 다음 layer로 전파되지 않는다. 즉, 학습이 되지 않는다.

2) 활성함수의 결과 값의 중심이 0이 아닌 0.5이다.

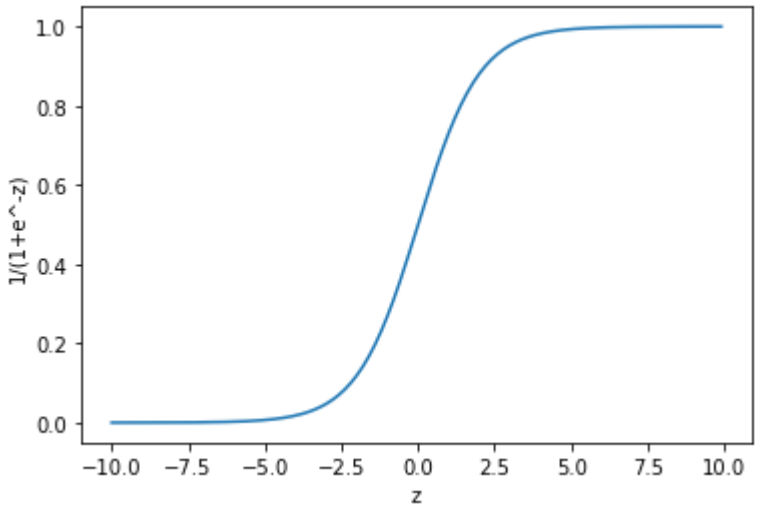
3) 계산이 복잡하다 (지수함수 계산)

!! Gradient Vanishing

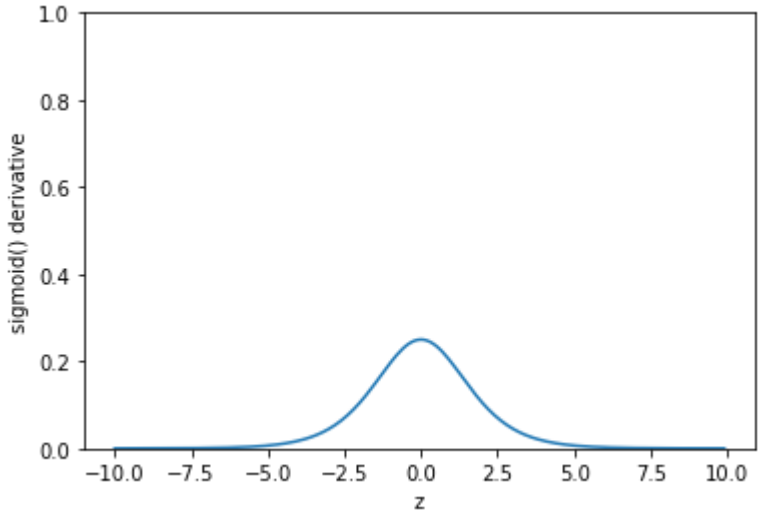
- 시그모이드와 같이 결과값이 포화(saturated)되는 함수는 gradient vanishing 현상을 야기
- 이전 레이어로 전파되는 그라디언트가 0에 가까워 지는 현상
- 레이어를 깊게 쌓으면 파라미터의 업데이트가 제대로 이루어지지 않음
- 양 극단의 미분값이 0에 가깝기 때문에 발생하는 문제

활성화 함수

시그모이드 함수



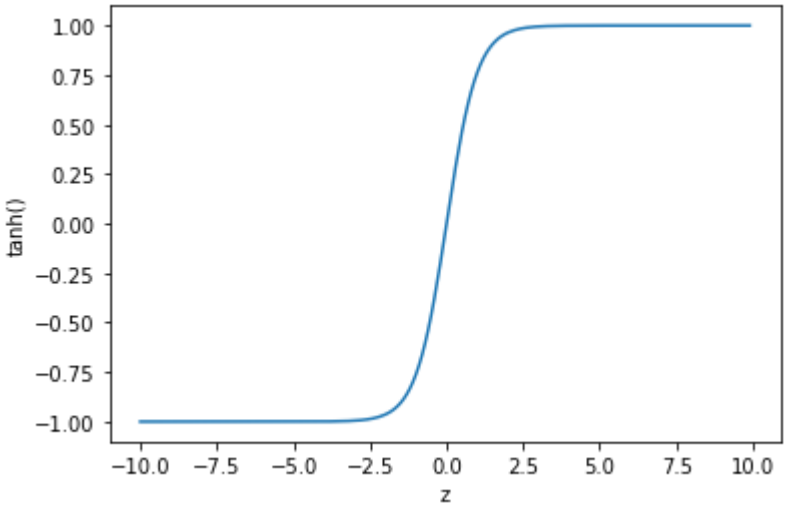
시그모이드 미분값



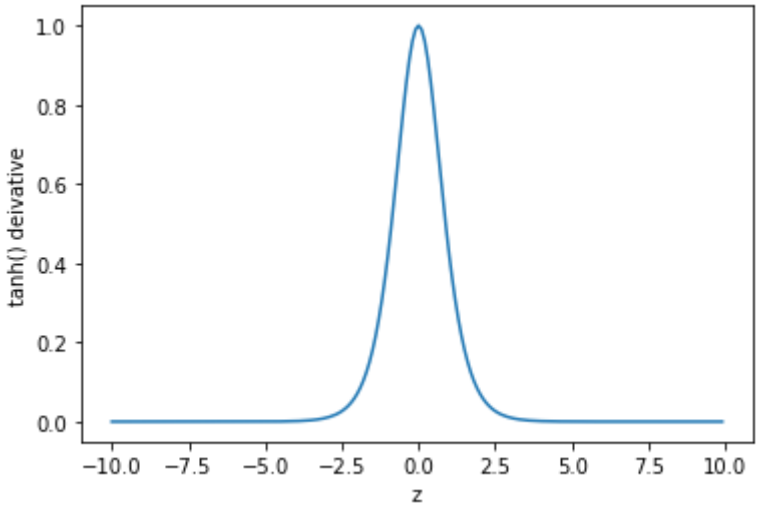
양쪽 꼬리가 0에 수렴하며 최대값이 0.25를 넘지 않는다.

2. 하이퍼 볼릭 탄젠트(tanh)

$$\tanh(x) = 2 * \text{sigmoid}(2 * x) - 1$$



$$\tanh(x)' = (1 - \tanh(x))(1 + \tanh(x))$$

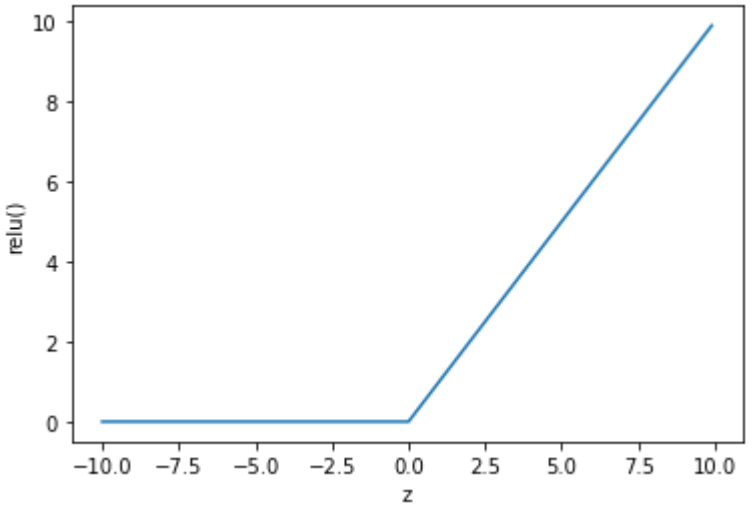


- 결과값이 [-1, 1] 사이로 제한됨. 결과값 중심이 0이다.
- 나머지 특성은 시그모이드와 비슷함. 시그모이드 함수를 이용하여 유도 가능
- 그러나, 여전히 gradient vanishing 문제가 발생

활성화 함수

3. 렐루(ReLU, Rectified Linear Unit)

$$f(x) = \max(0, x)$$

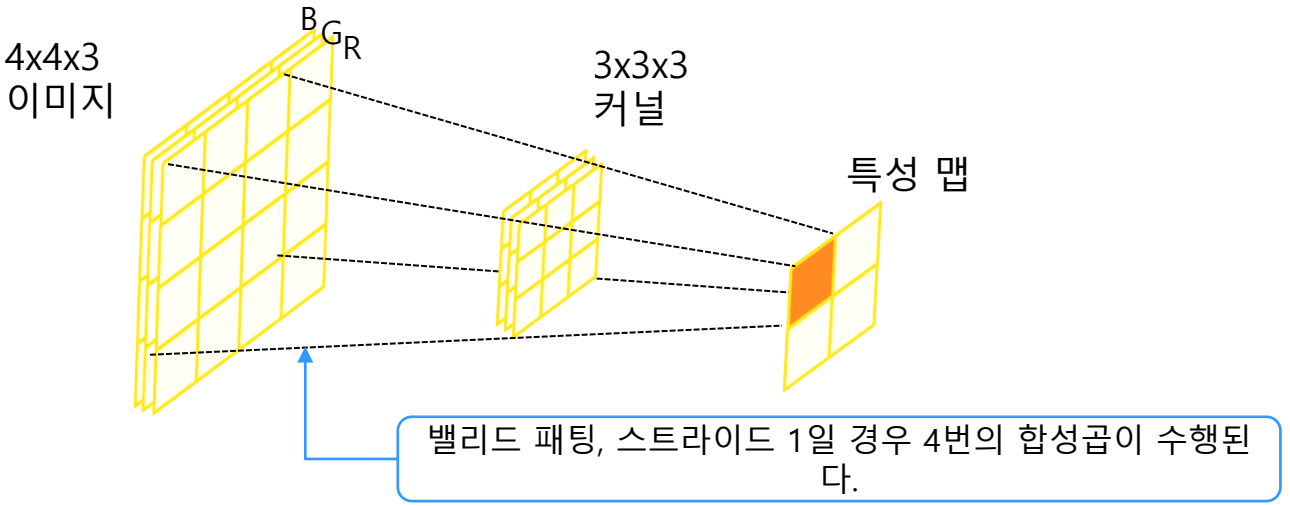


최근 뉴럴 네트워크에서 가장 많이 쓰이는 활성 함수
선형아니야? NO! 0에서 확 꺾이기 때문에 비선형이라고 본다.

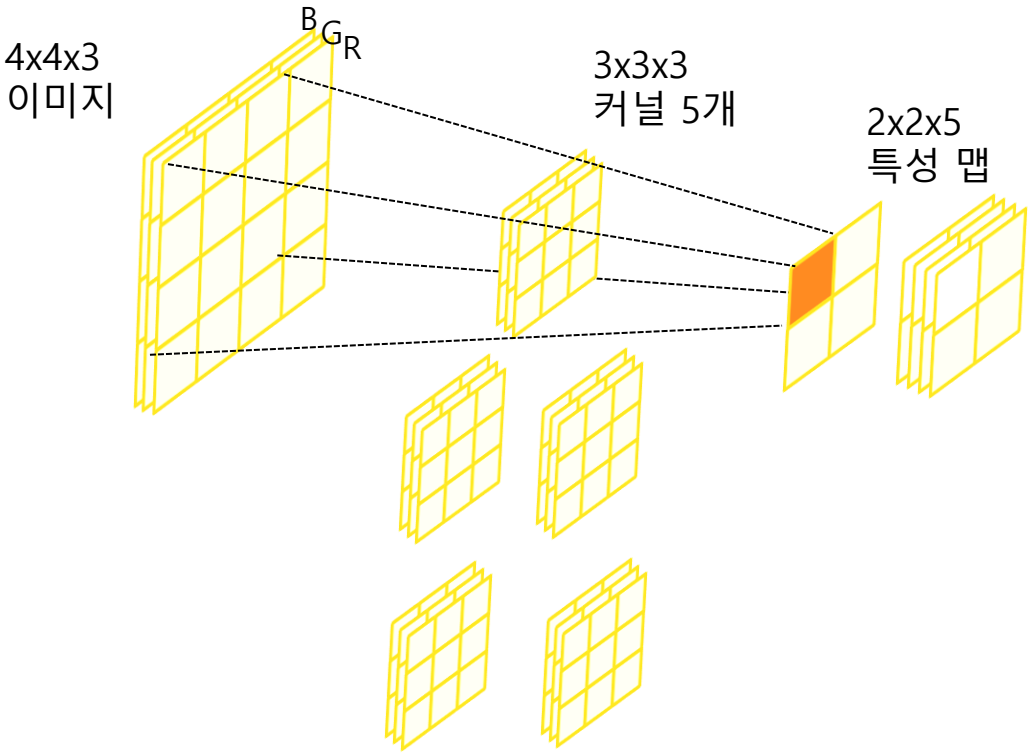
- 장점
(1) 양 극단값이 포화되지 않는다. (양수 지역은 선형적)
(2) 계산이 매우 효율적이다 (최대값 연산 1개)
(3) 수렴속도가 시그모이드류 함수대비 6배 정도 빠르다.

- 단점
(1) 중심값이 0이 아님 (마이너한 문제)
(2) 입력값이 음수인 경우 항상 0을 출력함 (마찬가지로 파라미터 업데이트가 안됨)

합성곱 층에서 일어나는 일

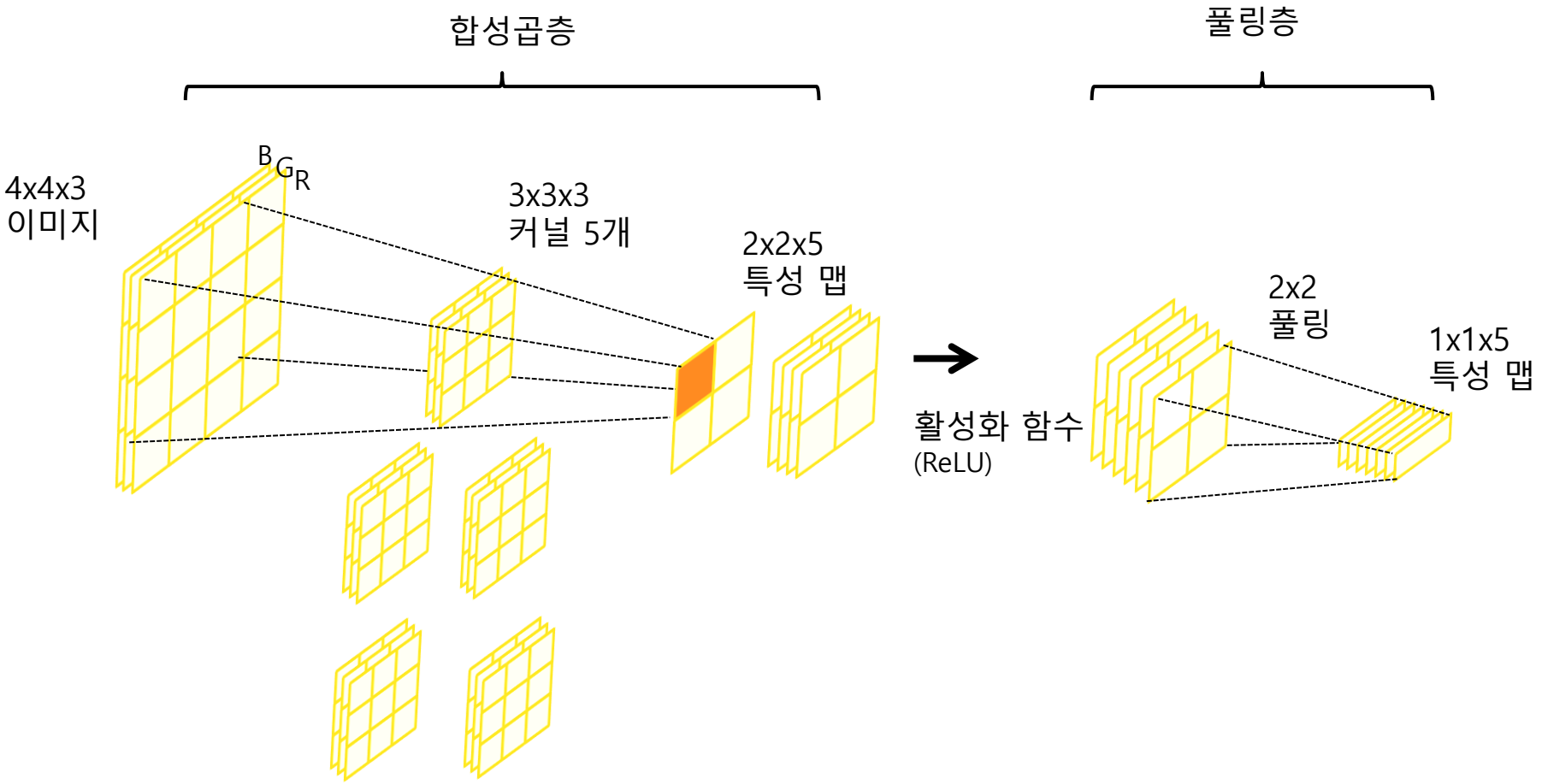


합성곱 층에서 일어나는 일



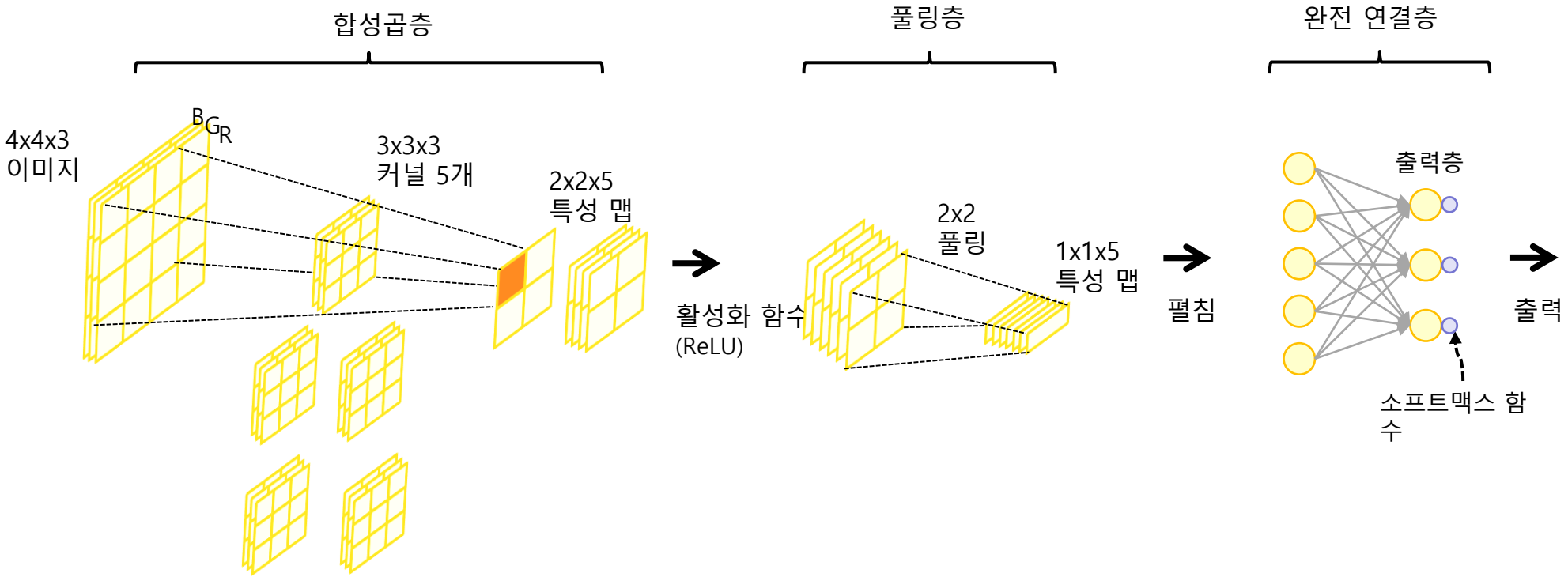
활성화 함수

풀링 층에서 일어나는 일

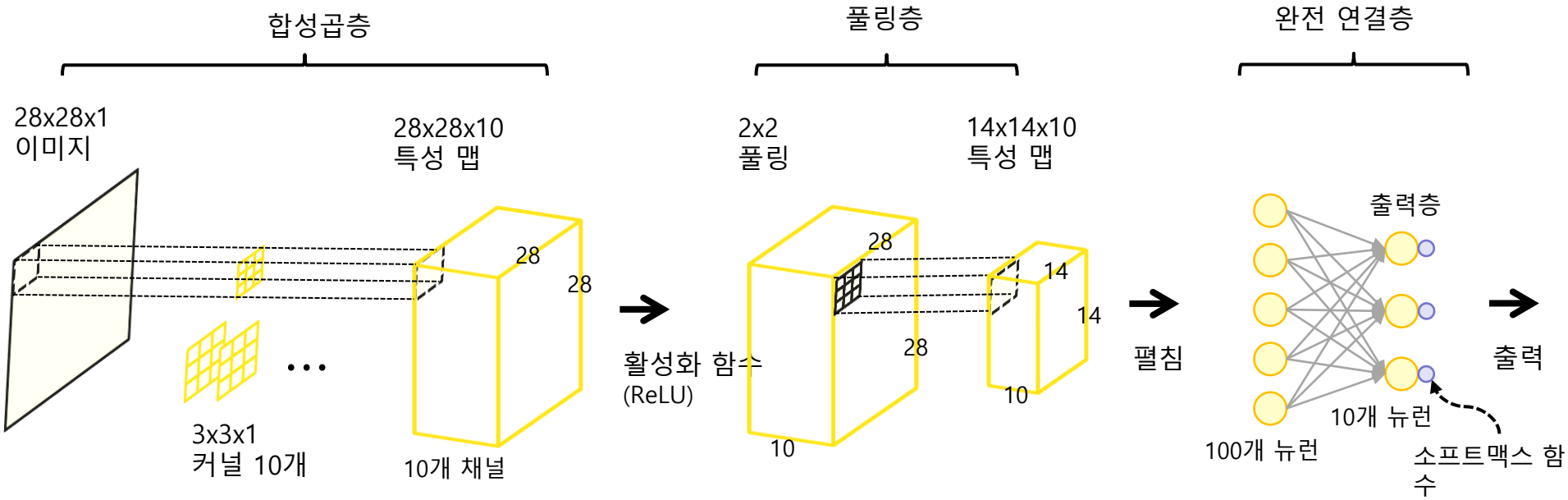


활성화 함수

특성 맵을 펼쳐 완전 연결 신경망에 주입한다.



합성곱 신경망의 전체 구조



- 28x28 크기의 흑백 이미지와 3x3 크기의 커널 10개로 합성곱 수행
- 2x2 크기의 최대 풀링을 수행하여 14x14x10로 특성 맵의 크기를 줄인다.
- 특성 맵을 일렬로 펼쳐서 100개의 뉴런을 가진 완전 연결층과 연결 시킨다.
- 10개의 클래스를 구분하기 위한 소프트맥스 함수에 연결한다.