딥러닝 기초

Data Learning Basic

나동빈(dongbinna@postech.ac.kr)

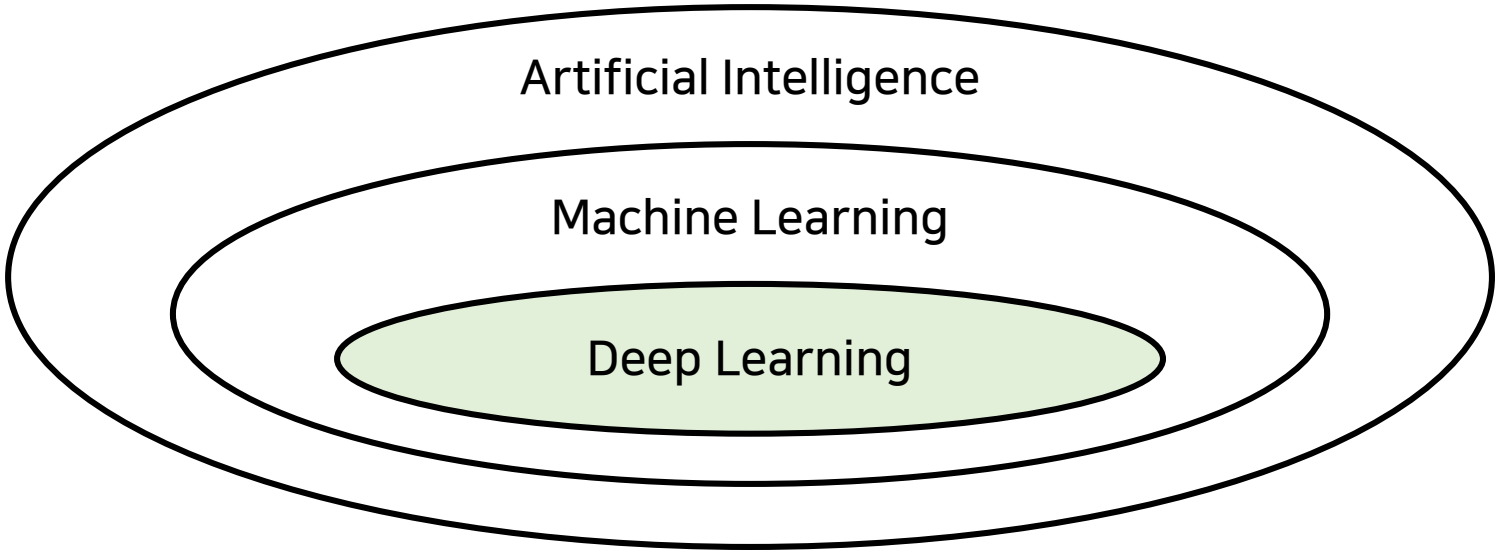
Pohang University of Science and Technology

학습자 목표

- **인공지능**의 정의를 이해할 수 있습니다.
- 딥러닝 모델 구현에 필요한 **기초적인 수학 원리**를 이해할 수 있습니다.
- PyTorch를 활용하여 딥러닝 모델을 구현하고 학습시킬 수 있습니다.
- 딥러닝을 활용한 **실전 프로젝트 예제**를 구현할 수 있습니다.

인공지능

- **인공지능**이란 기계를 통해 인공적으로 구현된 지능을 의미합니다.
  - **기계 학습**: 데이터를 반복적으로 학습하여 데이터에 잠재된 특징을 발견합니다.
  - **딥러닝**: 깊은 인공 신경망을 활용하여 더 높은 정확도를 얻습니다.

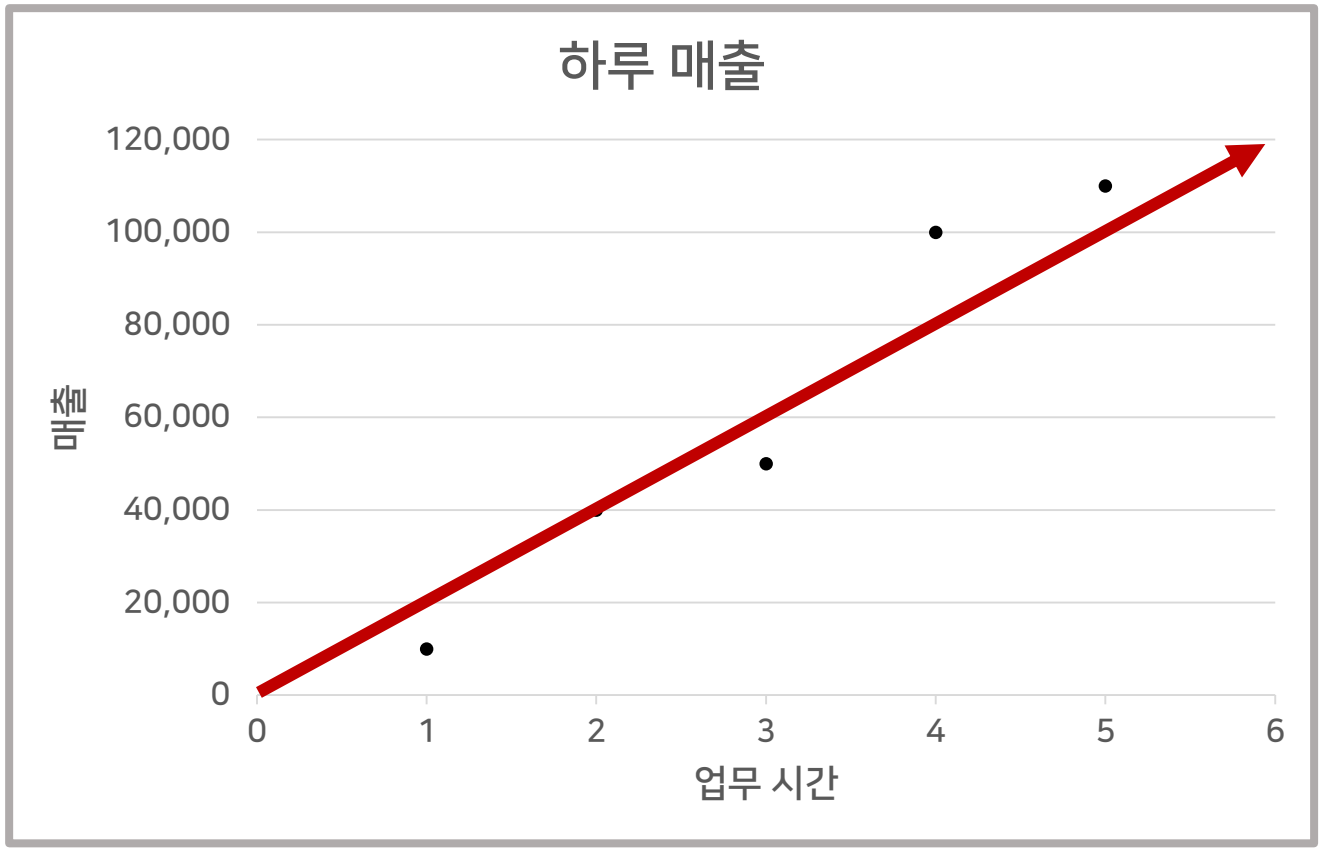


인공지능의 학습 방법 1. 지도 학습(Supervised Learning)

- **지도 학습**은 명시적인 정답을 제공하면서 학습시키는 유형입니다.
  - 회귀(Regression)
    - 특정한 데이터가 주어졌을 때 결과를 연속적인 값으로 예측합니다.
    - **예시**: “영어 공부를 7시간 했다면, 몇 점이 나올까요?”
  - 분류(Classification)
    - 종류에 따라서 데이터를 분류합니다.
    - **예시**: “이 이미지는 고양이인가요? 강아지인가요?”

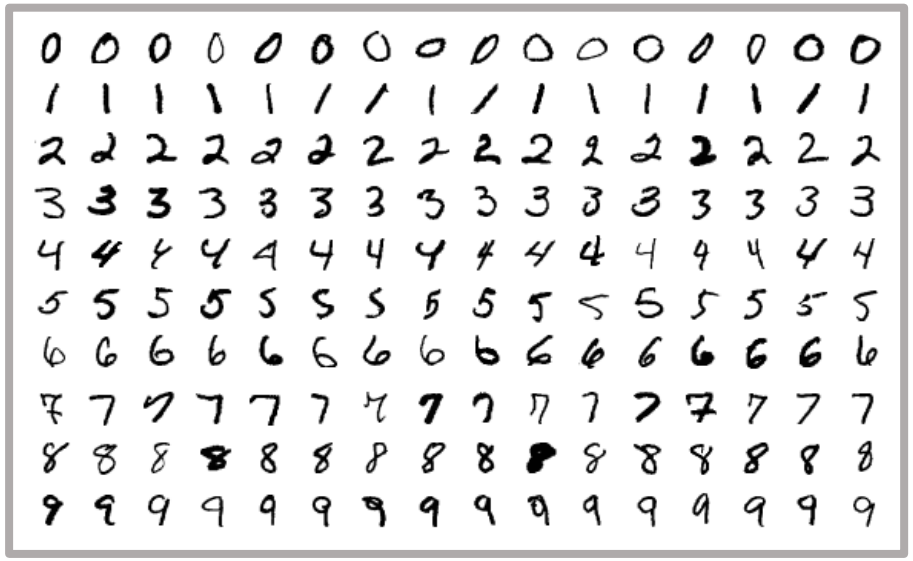
인공지능의 학습 방법 1. 지도 학습(Supervised Learning)

- 회귀(Regression)
  - 학습 시간에 따른 영어 점수 예측
  - 거리에 따른 이동 시간 예측
  - 업무 시간에 따른 매출 예측

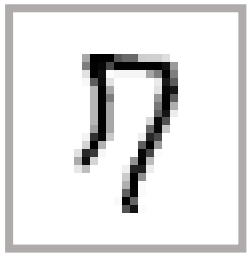


인공지능의 학습 방법 1. 지도 학습(Supervised Learning)

- **분류(Classification)**
  - 손글씨 분류
  - 강아지/고양이 분류
  - 배경 분류



학습 데이터셋



테스트 이미지

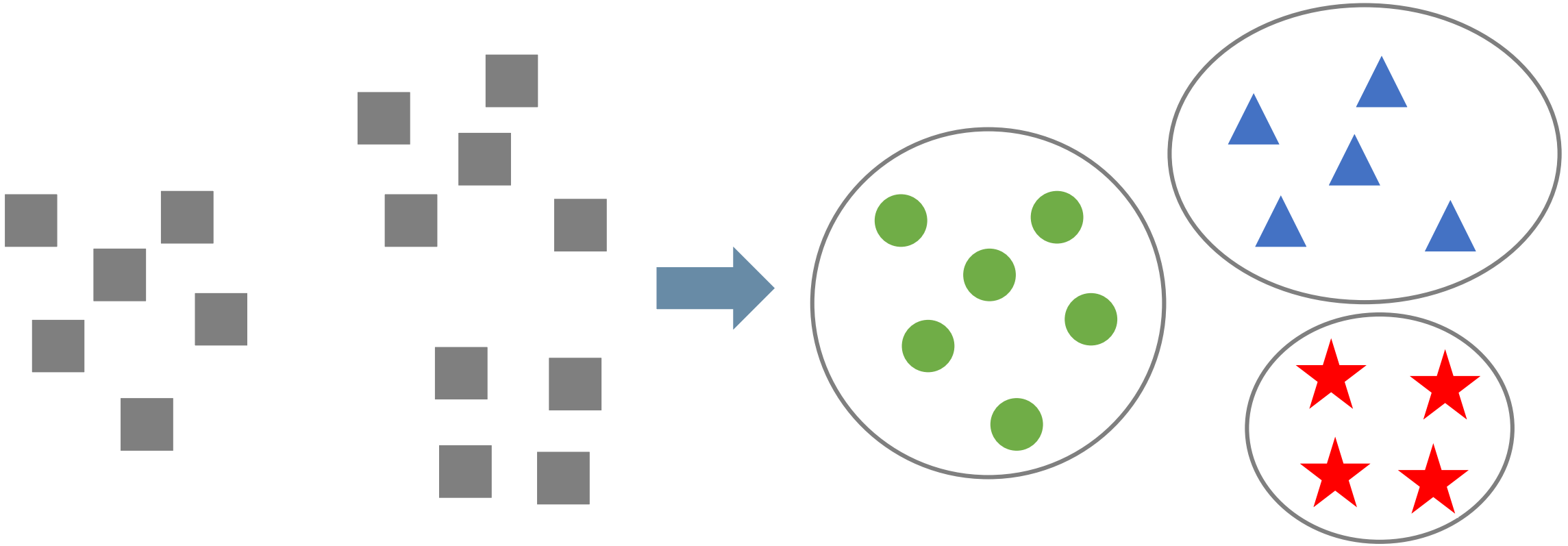
이 이미지는 **7**입니다.

인공지능의 학습 방법 2. 비지도 학습(Unsupervised Learning)

- **비지도 학습**은 명시적인 정답을 제공하지 않으면서 학습시키는 유형입니다.
  - **클러스터링(Clustering)**
    - 데이터를 특정한 기준으로 묶습니다.
    - **예시**: “사용자들을 3가지 집단으로 나누고 싶어요.”
  - **차원 축소(Dimensionality Reduction)**
    - 차원을 줄여 데이터 내 유의미한 특징을 추출합니다.
    - **예시**: “이 이미지들을 2차원 공간에 투영시켜서 시각화 할 수 있을까요?”

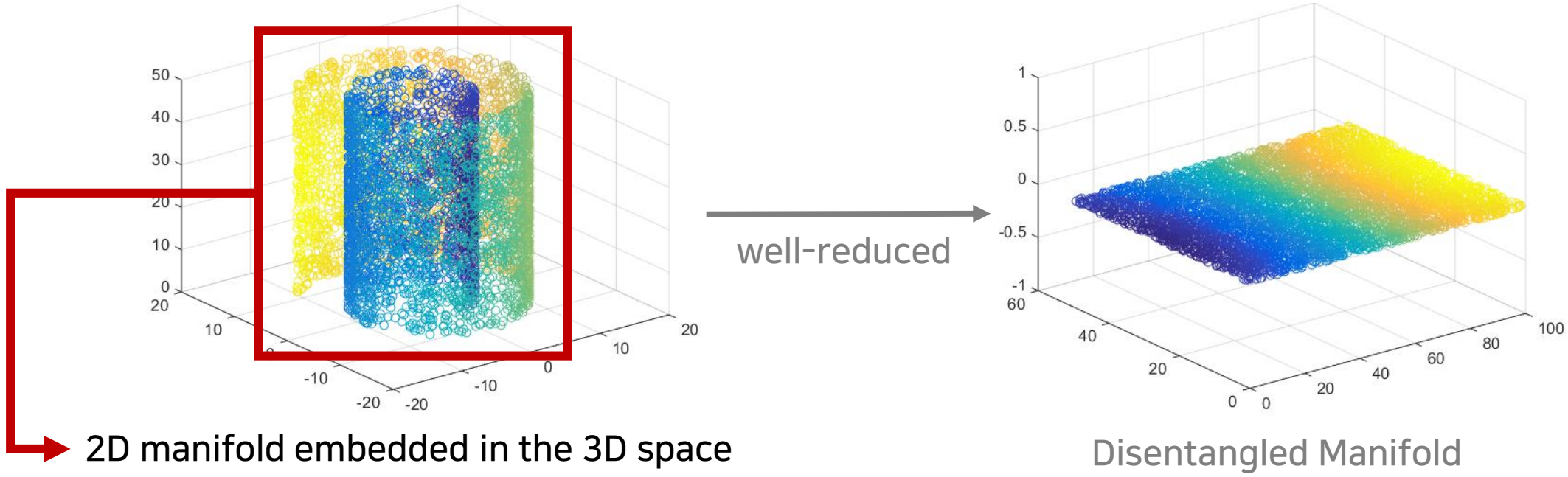
인공지능의 학습 방법 2. 비지도 학습(Unsupervised Learning)

- 클러스터링(Clustering)
  - 비슷한 유형의 사용자끼리 그룹화



인공지능의 학습 방법 2. 비지도 학습(Unsupervised Learning)

- **차원 축소(Dimension Reduction)**
  - 고차원 데이터의 차원을 축소하여 새로운 차원의 데이터를 생성합니다.
  - 예시: 데이터 시각화, 데이터 압축을 통한 복잡도 개선



딥러닝을 위한 도구

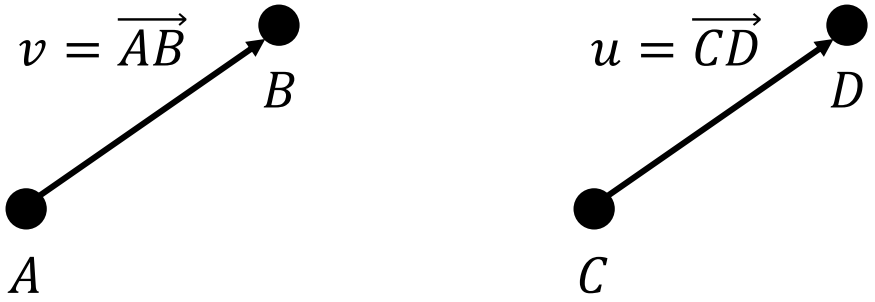
- 파이토치(PyTorch)
  - https://pytorch.org/
  - PyTorch는 빠르고 유연한 딥러닝 연구 플랫폼입니다.
  - GPU를 활용한 연산 가속을 지원합니다.

딥러닝을 위한 도구

- Google Colab
  - 나만의 머신 러닝 개발 환경을 1초 만에 가질 수 있도록 해주는 서비스입니다.
  - PyTorch을 포함한 머신 러닝 관련 라이브러리가 기본적으로 설치되어 있습니다.
  - 무료 서비스일 뿐만 아니라 GPU 런타임을 지원합니다.
  - 다른 사람과 함께 코드를 공유하며 협업하기에 좋은 개발 환경입니다.

벡터(Vector)

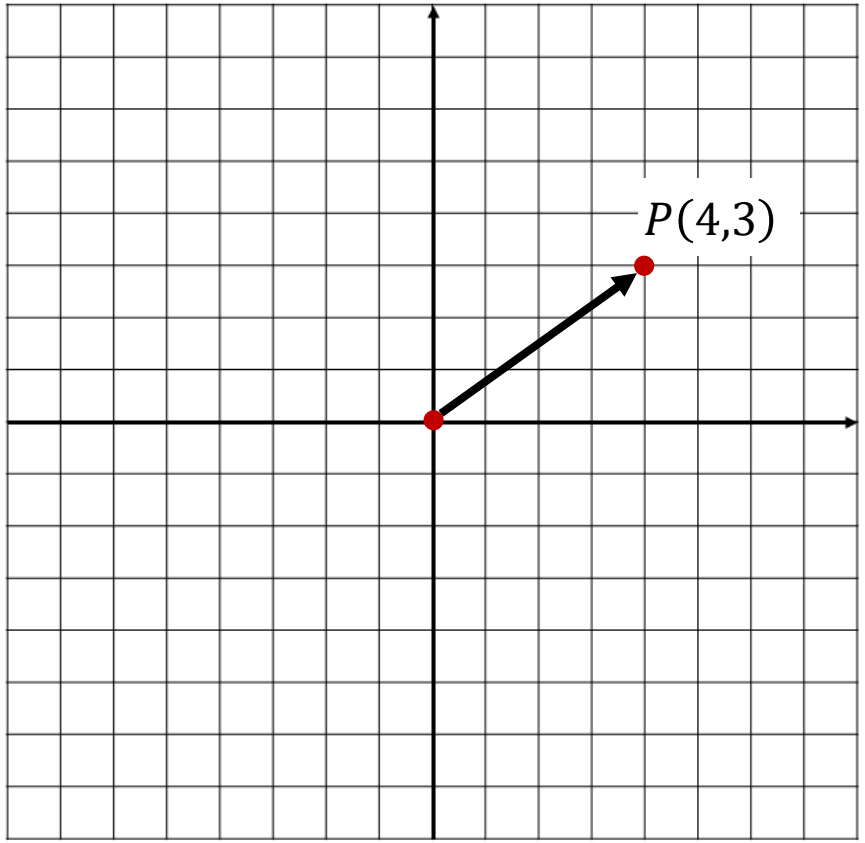
- **벡터**란 크기와 방향을 모두 가진 물리적 양을 나타낼 때 사용합니다.
  - **변위벡터**: 점 A(시점)와 점 B(종점)으로 구성되며 $v = \overrightarrow{AB}$로 표현합니다.
  - **동치**: 두 벡터의 길이와 방향이 같을 때 동치(equivalent)라고 합니다.



- 어떤 경우에는 좌표계를 도입하고 벡터를 대수적으로 다루는 것이 최선인 경우가 있습니다.
  - **변위벡터** 대신에 **위치벡터**를 효과적으로 사용할 수 있습니다.

벡터(Vector)

- 성분(Component)
  - 직교 좌표계의 원점에 벡터 x의 시점을 놓을 때의 종점
    - $x = < x_1, x_2 >$
  - 일반적으로 평면에 있는 점을 (x_1, x_2) 형태로 표현합니다.
    - 성분의 경우 $< x_1, x_2 >$로 표현합니다.
- 위치벡터(Position vector)
  - 원점으로부터 점 $P(4, 3)$에 이르는 벡터 $\overrightarrow{OP}$



행렬(Matrix)

- **행렬(Matrix)**
  - 행렬이란 M행, N열로 나열된 실수의 2차원 배열입니다. (M, N은 양의 정수)
  - 프로그래밍에서는 2차원 배열을 행렬처럼 이용할 수 있습니다.
- **행렬의 필요성**
  - 현실 세계의 많은 문제는 행렬을 이용해 해결할 수 있습니다.

$$\begin{array}{c} \\ \text{행} \end{array} \begin{array}{c} \text{열} \\ \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \end{array}$$

행렬의 연산

- **행렬의 덧셈과 뺄셈**
  - 두 행렬의 합이나 차를 계산할 때는 동일한 위치에 상응하는 원소끼리 계산합니다.
  - 기본적으로는 두 행렬의 크기가 같을 때 사용할 수 있습니다.

5	6
3	2

+

2	3
5	4

=

7	9
8	6

행렬의 연산

- **행렬과 스칼라의 연산**
  - 행렬은 상수(스칼라)와 연산할 수 있습니다.
  - 스칼라 연산을 할 때는 각 원소에 대하여 연산을 수행합니다.

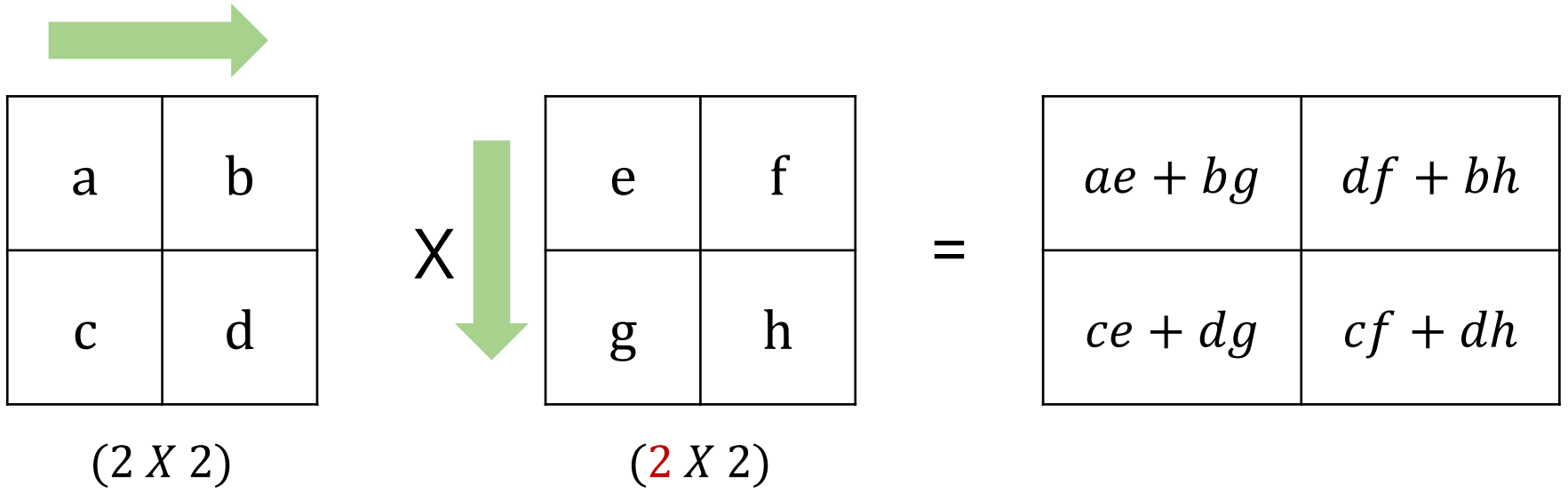
5	6
3	2

$+ \quad 2 \quad =$

7	8
5	4

행렬의 곱셈

- **행렬과 스칼라의 연산**
  - 두 행렬 A와 B는 곱할 수 있습니다.
  - 행렬 A의 열의 개수와 행렬 B의 행의 개수가 같아야 합니다.
  - 두 행렬의 곱 AB에서 행렬 AB의 크기는 A의 행과 B의 열의 개수를 가집니다.



행렬 사용 예시

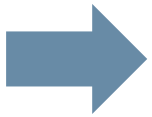
- 2년간의 판매 실적의 합 구하기

2019년

	키보드	마우스
상반기	1,200개	3,400개
하반기	1,400개	3,800개

2020년

	키보드	마우스
상반기	1,900개	2,900개
하반기	1,700개	3,200개



1200	3400
1400	3800

+

1900	2900
1700	3200

=

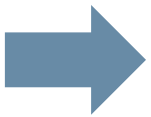
3100	6300
3100	7000

행렬 사용 예시

- 가장 체력을 많이 증가시킬 수 있는 선택지 고르기

	물약 1	물약 2	물약 3
선택지 1	3개	1개	4개
선택지 2	3개	2개	3개
선택지 3	4개	2개	2개

	체력 증가량
물약 1	10
물약 2	20
물약 3	30



3	1	4
3	2	3
4	2	2

$(3\ X\ 3)$

X

10
20
30

$(3\ X\ 1)$

=

170
160
140

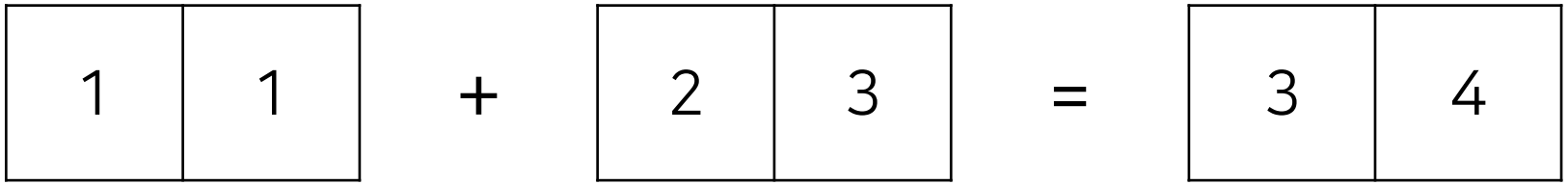
PyTorch의 Tensor

- Tensor 만들기

```
import torch

# 리스트를 Tensor로 변환
a = torch.tensor([1, 2, 3])
print(a)

# 값이 초기화되지 않은 행렬을 생성
a = torch.empty(4, 5)
print(a)

# 랜덤 값(uniform)으로 초기화된 행렬을 생성 (기본: 실수형)
a = torch.rand(4, 5)
print(a)

# 0으로 초기화된 행렬을 생성 (정수형)
a = torch.zeros((5, 3), dtype=torch.long)
print(a)
```

PyTorch의 Tensor

- Tensor 더하기




```
import torch

# Tensor를 만들고 크기를 출력
a = torch.tensor([1, 1])
print(a.size())

# Tensor 더하기
b = torch.tensor([2, 3])
c = a + b

print(c)
```

PyTorch의 Tensor

- Tensor 곱하기
  - 기본적인 Tensor의 곱셈은 동일한 위치의 원소끼리 곱하는 연산을 의미합니다.

0	1
2	3

*

1	2
3	4

=

0	2
6	12

PyTorch의 Tensor

- Tensor 곱하기

```
import torch

# Tensor 곱하기
a = torch.tensor([[0, 1], [2, 3]])
b = torch.tensor([[1, 2], [3, 4]])
c = a * b

print(c)
```

PyTorch의 Tensor

- Tensor의 행렬 곱
  - Tensor는 행렬 곱 연산을 지원합니다.

```
import torch

# Tensor 곱하기
a = torch.tensor([[0, 1], [2, 3]])
b = torch.tensor([[1, 2], [3, 4]])
c = torch.matmul(a, b)

print(c)
```

PyTorch의 Tensor

- 서로 다른 형태의 Tensor 연산
  - **브로드캐스트**: 형태가 다른 행렬을 연산할 수 있도록 형렬의 형태를 동적으로 변환합니다.

0	1	2	3
4	5	6	7
0	1	2	3
4	5	6	7

+

0
1
2
3

=

0	1	2	3
4	5	6	7
0	1	2	3
4	5	6	7

+

0	0	0	0
1	1	1	1
2	2	2	2
3	3	3	3

PyTorch의 Tensor

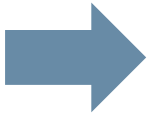
- 서로 다른 형태의 Tensor 연산
  - **브로드캐스트**: 형태가 다른 행렬을 연산할 수 있도록 형렬의 형태를 동적으로 변환합니다.

```
import torch

# Tensor 브로드캐스트
a = torch.tensor([1, 2, 3])
b = torch.tensor([[1, 2, 3], [1, 2, 3]])
c = a + b

print(c)
```

PyTorch의 Tensor

- **Tensor의 형태 변경**
  - Tensor는 인덱싱 표기법을 사용할 수 있습니다.
  - 다양한 방법으로 형태를 변경할 수 있습니다.

0	1	2	3
4	5	6	7
0	1	2	3
4	5	6	7



x[:, 2] =

2
6
2
6

PyTorch의 Tensor

- Tensor의 형태 변경

```
import torch

a = torch.tensor([
    [0, 1, 2, 3],
    [4, 5, 6, 7],
    [0, 1, 2, 3],
    [4, 5, 6, 7]
])

print(a[:, 2])
a = a.view(16)
print(a)
a = a.view(4, 4)
print(a)
a = a.view(-1, 8)
print(a)
```

PyTorch의 Tensor

- Tensor와 NumPy 변환

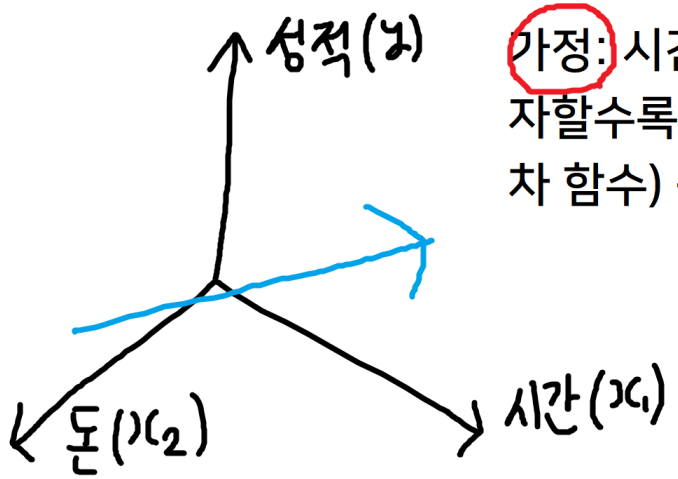
```
import torch
import numpy as np

a = torch.tensor([
    [0, 1, 2, 3],
    [4, 5, 6, 7]
])
print(a)

b = a.numpy()
print(b)

c = torch.from_numpy(b)
print(c)
```

PyTorch의 Tensor

- Tensor에 대한 추가적인 내용 살펴보기
  - https://pytorch.org/docs/stable/torch.html

선형 회귀

- 현실 세계의 많은 데이터는 **선형적인 구조**를 내재합니다.
  - 예시: "많은 시간을 공부할수록 그 시간에 비례하여 실력이 늘지 않을까?"
  - 예시: "많은 돈을 투자할수록, 더 성능이 좋은 제품을 살 수 있지 않을까?"



가정: 시간과 돈을 투자하면 투자할수록 성적이 선형적으로(1차 함수) 증가한다.

$$y = w_1 x_1 + w_2 x + b$$

선형 방정식
= 일차 방정식
= linear equation

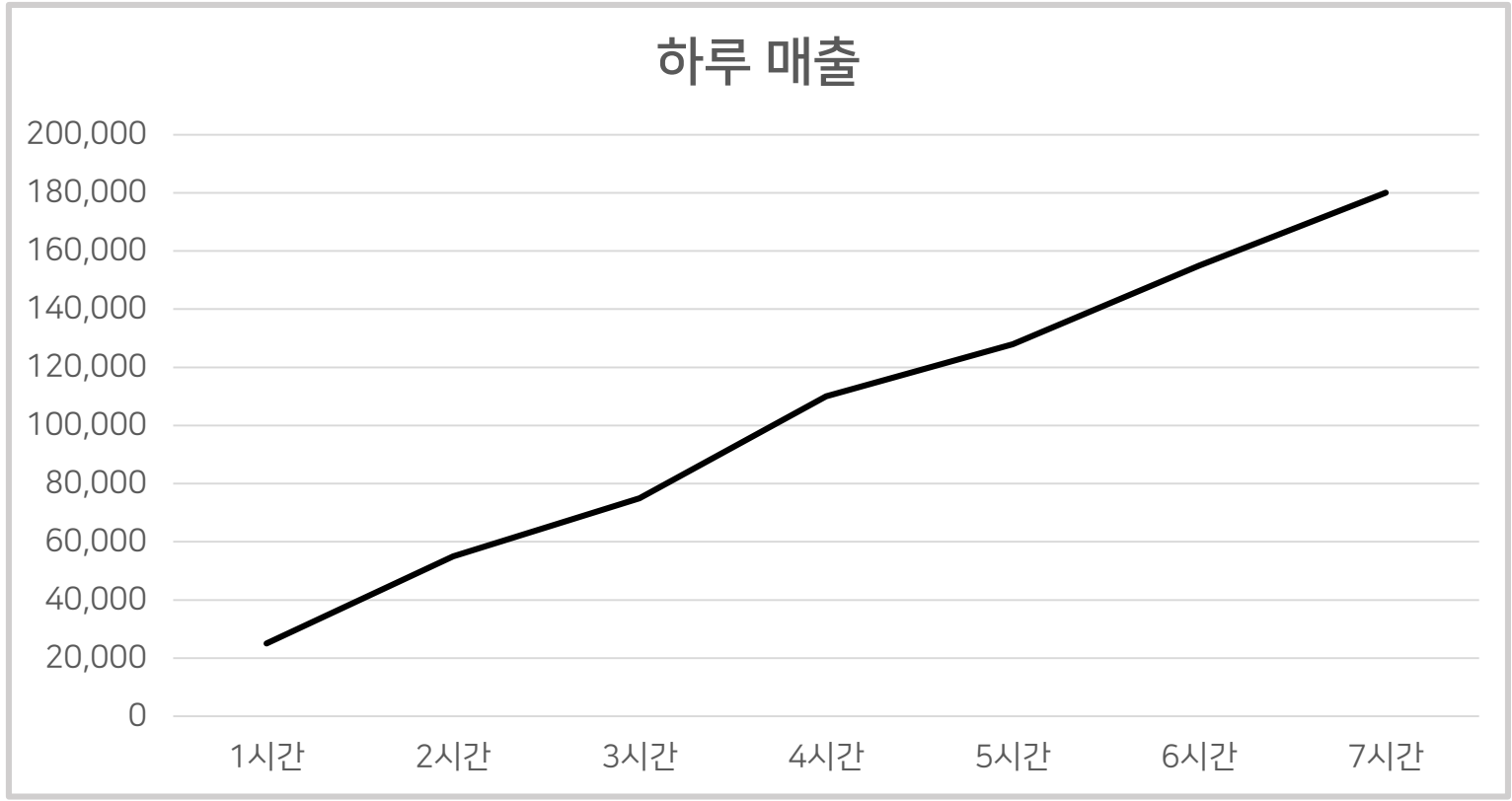
선형 회귀

- **장사꾼의 매출**
  - 어느 날, 한 장사꾼의 노동 시간과 매출액은 다음과 같았습니다.
  - 8시간을 일한다면 매출은 얼마일까요?

하루 노동 시간	매출
1	25,000
2	55,000
3	75,000
4	110,000
5	128,000
6	155,000
7	180,000

선형 회귀

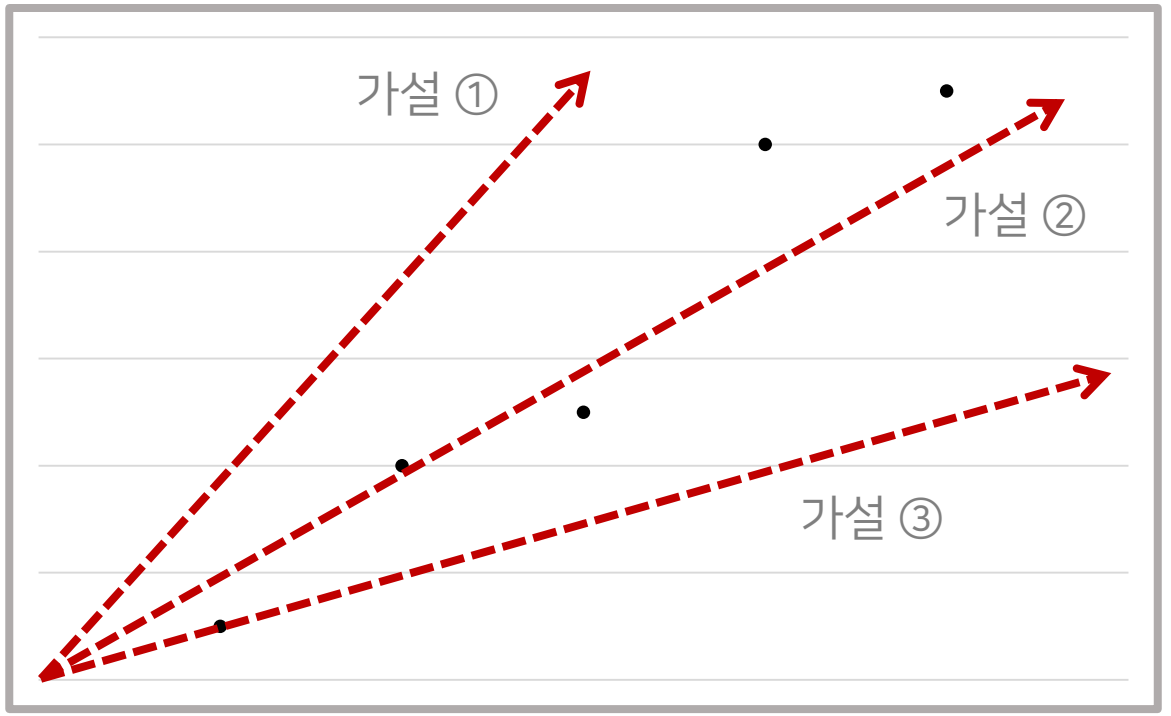
- **장사꾼의 매출**
  - 매출 데이터를 그래프로 표현해 봅시다.



"**선형 함수**로
모델링 할 수 있겠구나!"

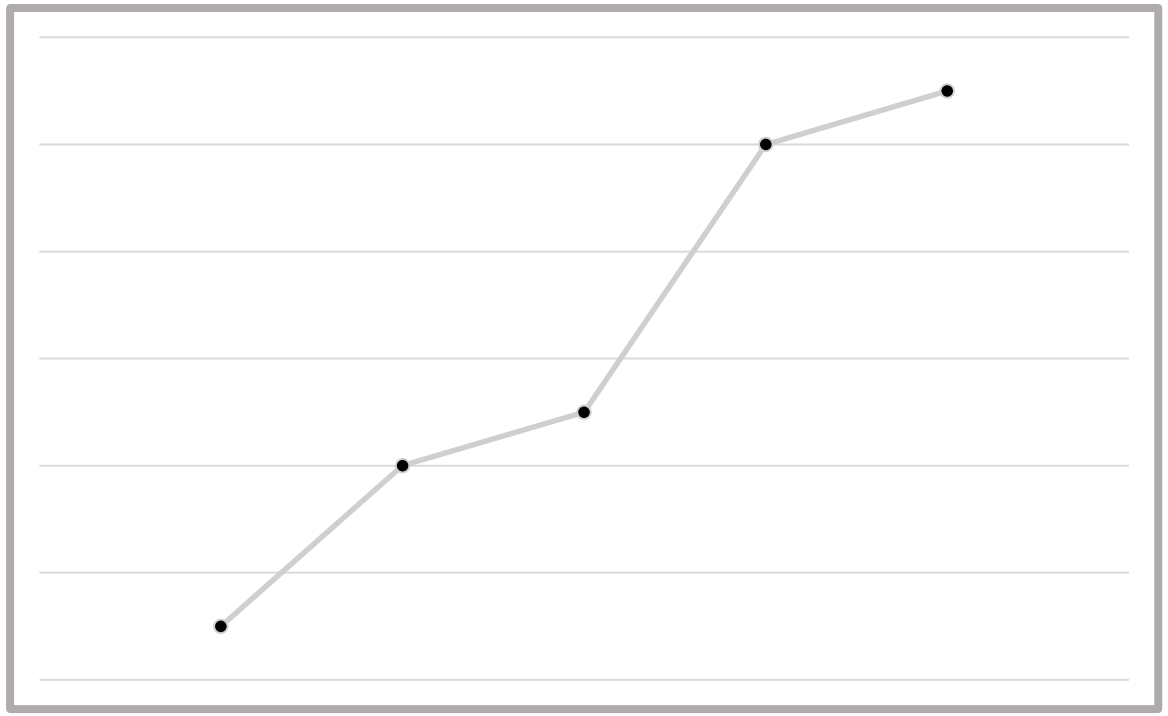
선형 회귀

- **선형 회귀(Linear regression)**
  - 주어진 데이터를 학습시켜서 가장 합리적인 선형 함수를 찾아내는 접근 방법을 의미합니다.
  - 데이터는 3개 이상일 때 의미가 있습니다.
  - 데이터를 가장 잘 나타내는 선형 함수는?



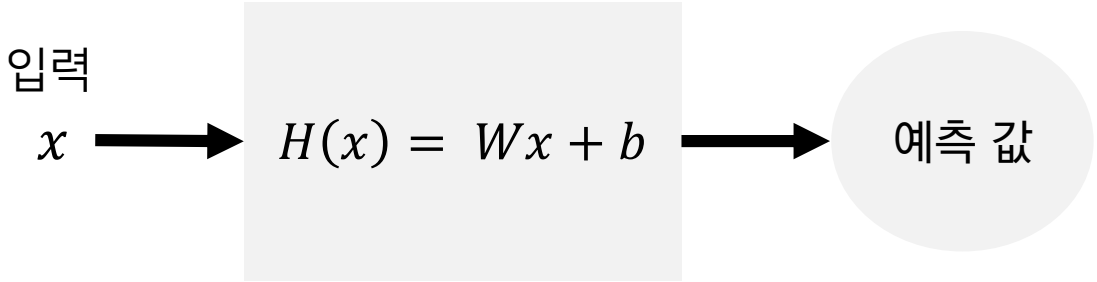
좋은 가설의 기준

- X와 Y의 관계를 하나의 직선으로 완벽하게 표현할 수는 없을 수 있습니다.
- 그렇다면 그나마 합리적인 직선은 어떻게 찾을 수 있을까요?

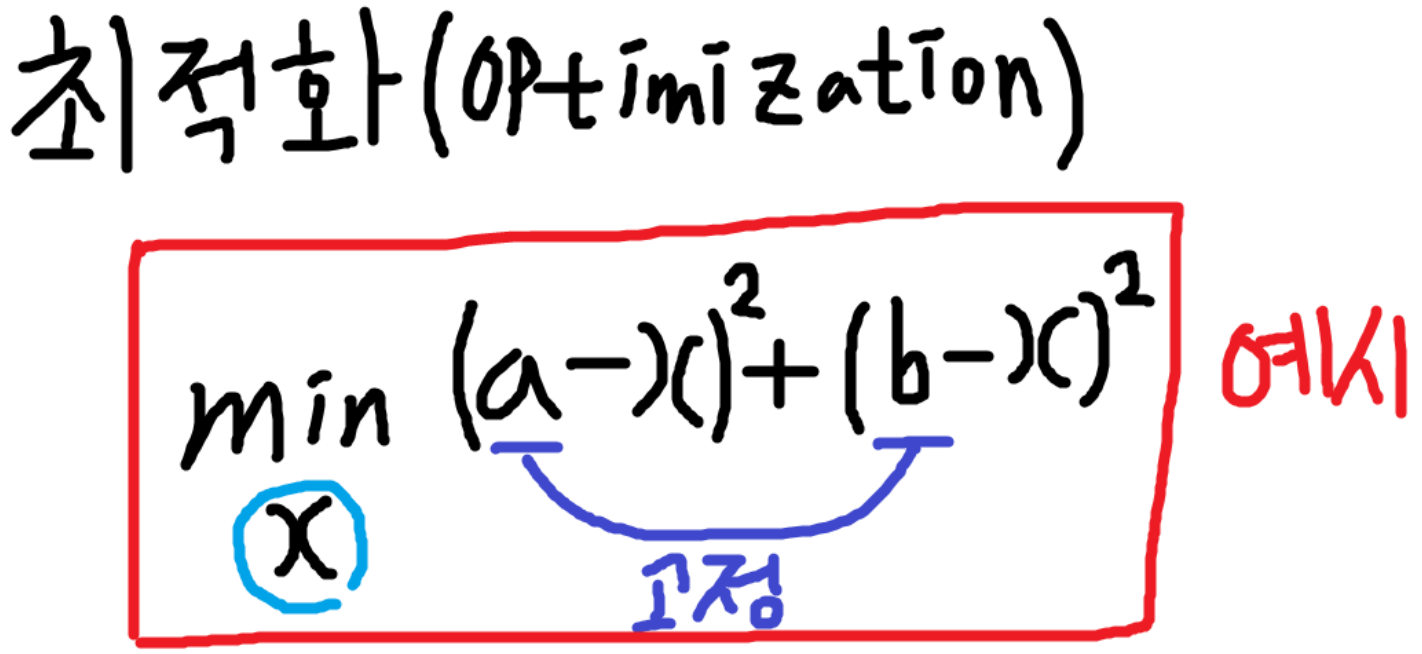


좋은 가설의 기준

- 선형 함수를 이용해 직선을 표현할 수 있습니다.
- 가설 함수(파라미터)를 수정해 나가면서 가장 합리적인 식을 찾아낼 수 있습니다.
- 가설 함수: $H(x) = Wx + b$



최적화 문제



흔히 알려진 머신러닝, 딥러닝 문제는 아무튼 최적화 문제를 해결하는 형태

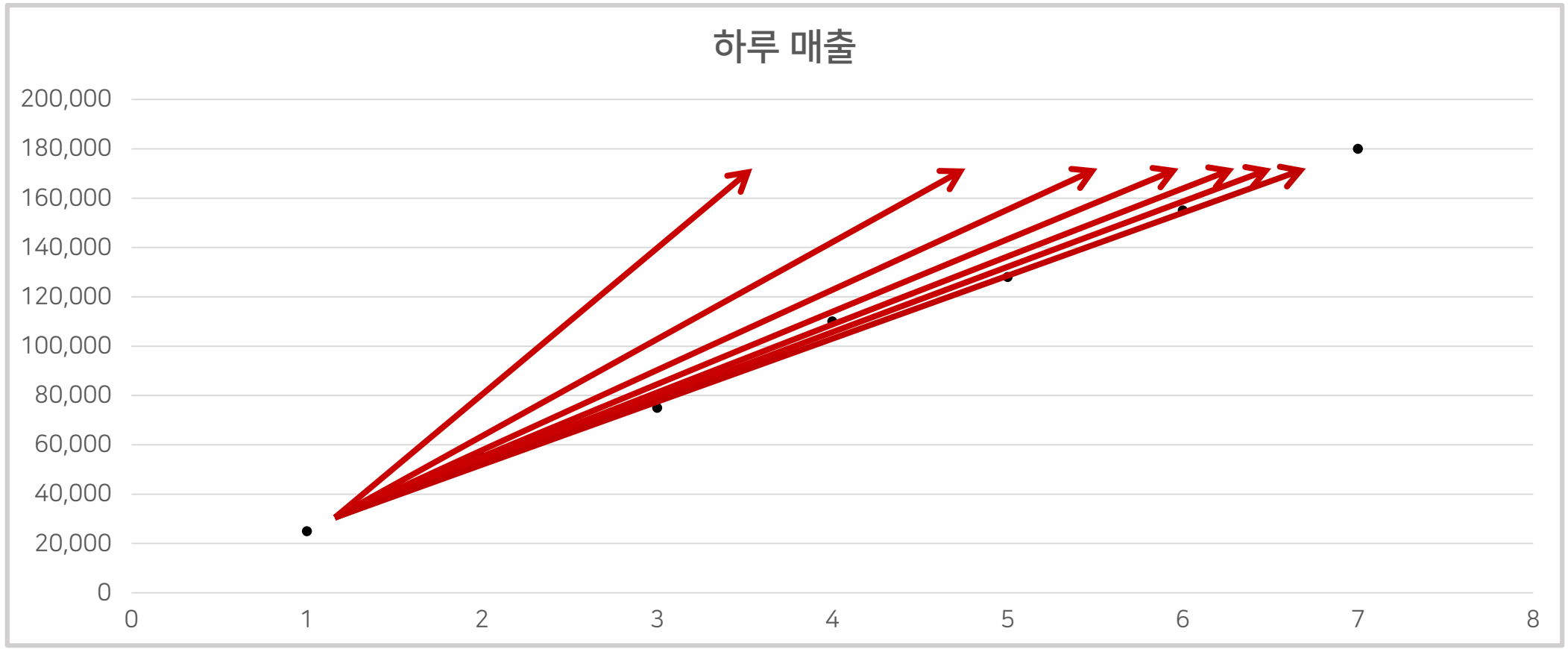
학습(Training) 개요

- 선형 회귀에서의 학습은 주어진 데이터를 이용해 선형 함수를 수정해 나가는 것입니다.
  - 학습을 거쳐 가장 합리적인 파라미터를 도출합니다.
- 학습을 많이 해도 완벽한 식을 찾아내지 못할 수 있습니다.
- 하지만 실제 사례에서는 근삿값을 찾는 것만으로도 충분할 때가 많습니다.
  - **딥러닝**은 함수를 적절히 근사해주는 소프트웨어입니다.

*Universal Approximation Theorem

학습(Training) 개요

- 학습이 이루어지는 과정을 직관적으로 이해해 봅시다.



비용 함수(Cost Function)

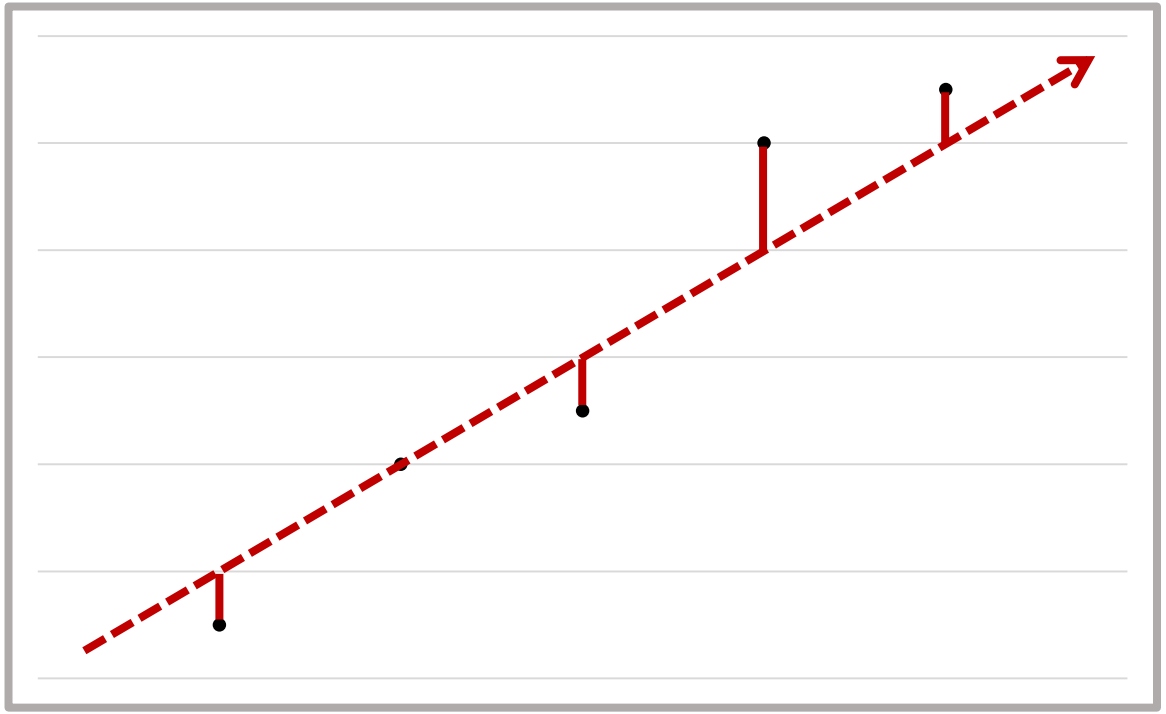
- **비용(Cost)**
  - 가설이 얼마나 정확한지 판단하는 기준입니다.
  - 가설이 정확하지 않다면, 비용이 많이 발생합니다.
  - 비용을 줄이는 방향으로 학습을 진행합니다.
  - 일반적으로 비용을 계산할 때는 **실제 값과 예상값이 얼마나 다른가**를 기준으로 설정합니다.

비용 함수(Cost Function)

- **비용(Cost)**
  - 단순히 비용을 **실제 값 – 예상 값**으로 설정하면 될까요?

$H(x) = Wx + b$ 일 때

$C = y - H(x)$라면?



비용 함수(Cost Function)

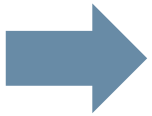
- MSE (Mean Squared Error)
  - 모든 데이터에 대한 (**실제 값** – **예상 값**)^{2}의 합으로 비용을 계산합니다.
  - 따라서 다음 식을 최소화하는 파라미터 (W, b)를 찾는 것이 목표입니다.
  - m: 데이터의 개수

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x_i) - y_i)^2$$

비용 함수(Cost Function)

- **비용 구하기 예제**
  - 데이터가 다음과 같다면 어떤 선형 함수가 적절할까요?

X	Y
1	2
2	4
3	6



$$y = 2x$$

  - 데이터의 개수가 작을 때 인간은 선형 함수를 간단히 찾을 수 있습니다.
  - 하지만 데이터의 개수가 많을 때도 인간이 쉽게 계산할 수 있을까요?
  - 이러한 비용을 **기계적으로** 줄이는 방법이 필요합니다.

비용 함수(Cost Function)

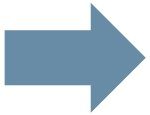
- 비용 구하기 예제
  - $W = 1, b = 2$일 때의 비용을 계산합시다.
  - $y = x + 2$

X	Y
1	2
2	4
3	6

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}(H(x_i) - y_i)^2$$

$$= \frac{(3-2)^2 + (4\ -4)^2 + (5\ -6)^2}{3}$$

$$= 2/3$$



비용을 더 줄이려면, W와 b를 어떻게 바꾸어야 할까요?

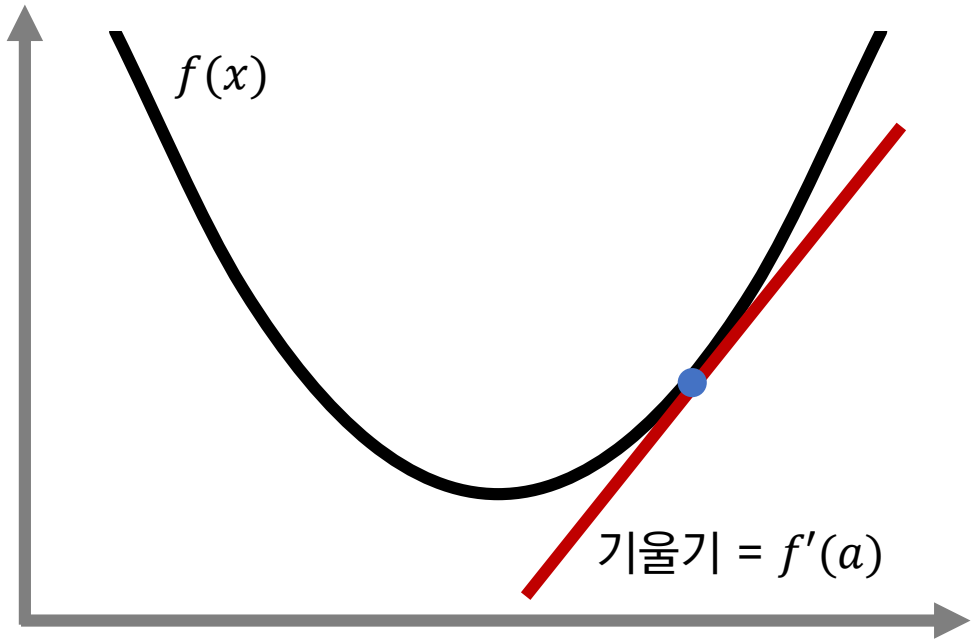
미분(Ordinary Derivative)

- 미분을 한 마디로 정의하면?

기울기(gradient)를 구해주는 작업

미분(Ordinary Derivative)

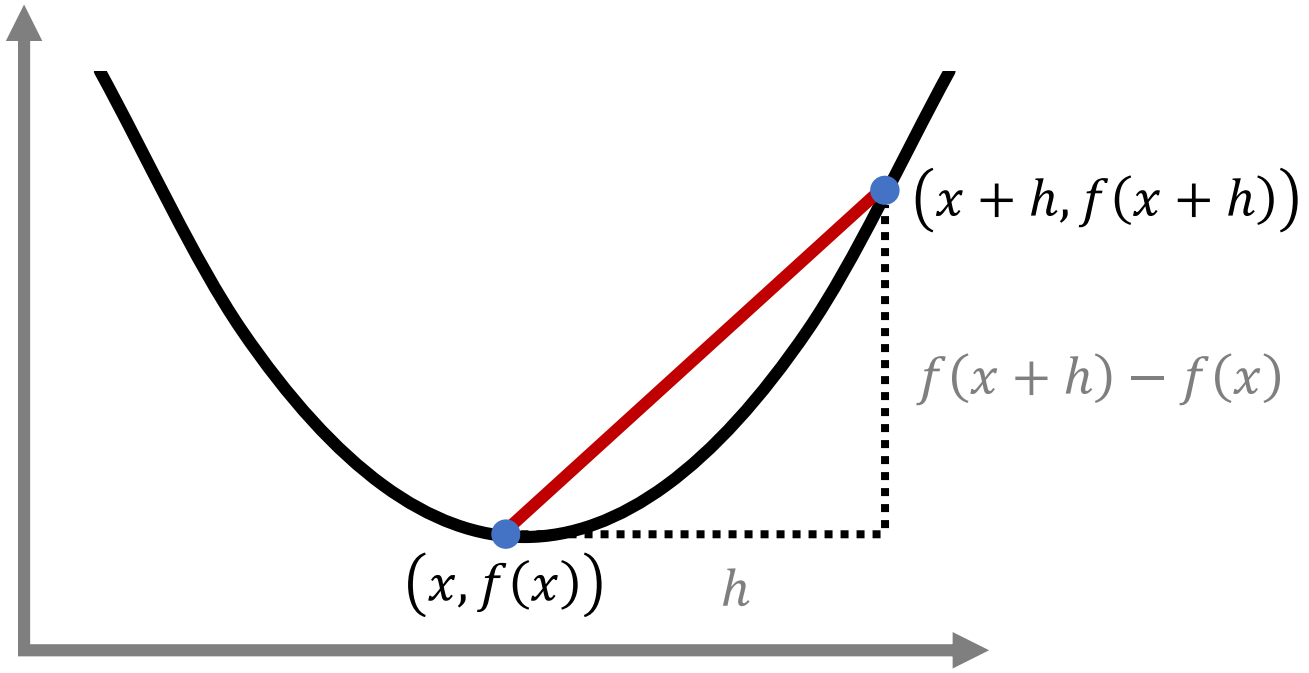
- **도함수(Derivative)** $f'(x)$란?
  - 입력(파라미터) x에 대하여, 함수 f의 기울기(gradient)를 알려주는 함수
  - 입력(파라미터) x에 대하여, 함수 f가 얼마나 민감하게 변화하는지(순간 변화율)를 알려주는 함수
- **미분(Differentiation)**이란? 도함수 $f'(x)$를 계산하는 작업



따라서 어떠한 함수 $f(x)$가 있을 때, 특정한 점 a의 위치에서의 기울기(gradient) 혹은 순간 변화율 값을 구하고 싶다면 $f'(a)$를 계산하면 됩니다.

미분(Ordinary Derivative)

- 기울기의 정의
  - $f(x)$의 변화량 / x의 변화량
  - 특정 함수에서 x가 h만큼 변할 때의 기울기를 계산해 봅시다.



- x가 h만큼 변할 때의 기울기는?

$$\frac{f(x+h)-f(x)}{h}$$

도함수란 x에서의 순간 변화율을 알려주는 함수이므로, $h=0$일 때의 값을 계산해야 합니다.

미분(Ordinary Derivative)

- 따라서 도함수 $f'(x)$는 다음과 같이 계산할 수 있습니다.

$$f'(x) = \frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

- 미분은 인공지능 분야에서 뉴럴 네트워크를 **학습(training)**을 시키기 위한 과정에서 사용됩니다.
- 뉴럴 네트워크의 파라미터를 기울기 값을 기준으로 학습을 시키는 방법을 주로 이용합니다.
- 실제로 h를 0.0001 정도로 설정하여 근사할 수도 있으나 기계학습 라이브러리에서는 실제로 도함수를 계산하여 학습을 진행하며, 레이어가 많으므로 **연쇄법칙(chain-rule)**을 이용하게 됩니다.
- 기본 미분 공식

$$(constant)' = 0 \qquad \left(ax^k\right)' = kax^{k-1}$$

미분(Ordinary Derivative)

- 기본 미분 공식

$$(constant)' = 0 \qquad \left(ax^k\right)' = kax^{k-1}$$

- 지수 함수 미분 공식

$$(e^x)' = e^x$$

$$(a^x)' = a^x \ln a$$

$$\left(\mathrm{e}^{f(x)}\right)' = \mathrm{e}^{f(x)} * f'(x)$$

$$\left(a^{f(x)}\right)' = a^{f(x)} * \ln a * f'(x)$$

- 로그 함수 미분 공식

$$(\ln x)' = \frac{1}{x}$$

$$(\log_a x)' = \frac{1}{x \ln a}$$

$$(\ln f(x))' = \frac{f'(x)}{f(x)}$$

- 삼각 함수 미분 공식

$$(\sin x)' = \cos x$$

$$(\cos x)' = -sin\, x$$

$$(\tan x)' = sec^2\, x$$

편미분(Partial Derivative)

- **편미분**: 다변수 함수(multivariate function)에서 하나의 변수를 기준으로 미분하는 작업
- 미분할 때 다른 변수는 모두 상수(constant) 취급합니다.

$$f(x,y) = 2x^2 + xy + 5y$$

$$\frac{\partial f(x,y)}{\partial x} = \frac{\partial(2x^2 + xy + 5y)}{\partial x} = 4x + y$$

$$\frac{\partial f(x,y)}{\partial y} = \frac{\partial(2x^2 + xy + 5y)}{\partial y} = x + 5$$

편미분(Partial Derivative)

- **편미분**: 다변수 함수(multivariate function)에서 하나의 변수를 기준으로 미분하는 작업
- 미분할 때 다른 변수는 모두 상수(constant) 취급합니다.

$y = x^2 + 5$ $\quad y' = 2x$

$y = 3x + 6$ $\quad y' = 3$

$$f(x) = 3x_1 + 4x_2^2 + 5x_3^2 + 7x_1x_2$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

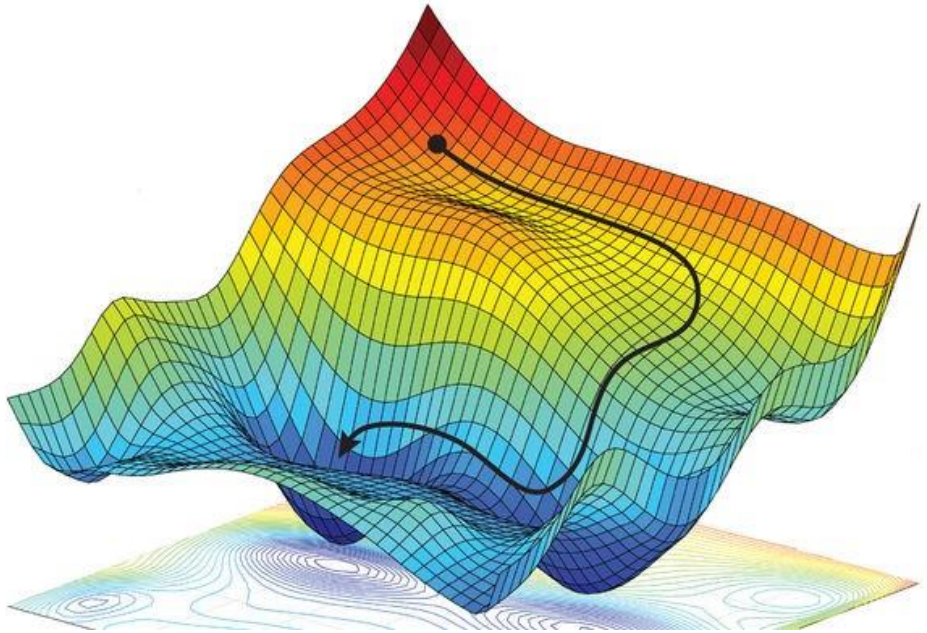
$$\frac{\partial f(x)}{\partial x_1} = 3 + 7x_2$$

$$\frac{\partial f(x)}{\partial x_2} = 8x_2 + 7x_1$$

$$\frac{\partial f(x)}{\partial x_3} = 10x_3$$

편미분(Partial Derivative)

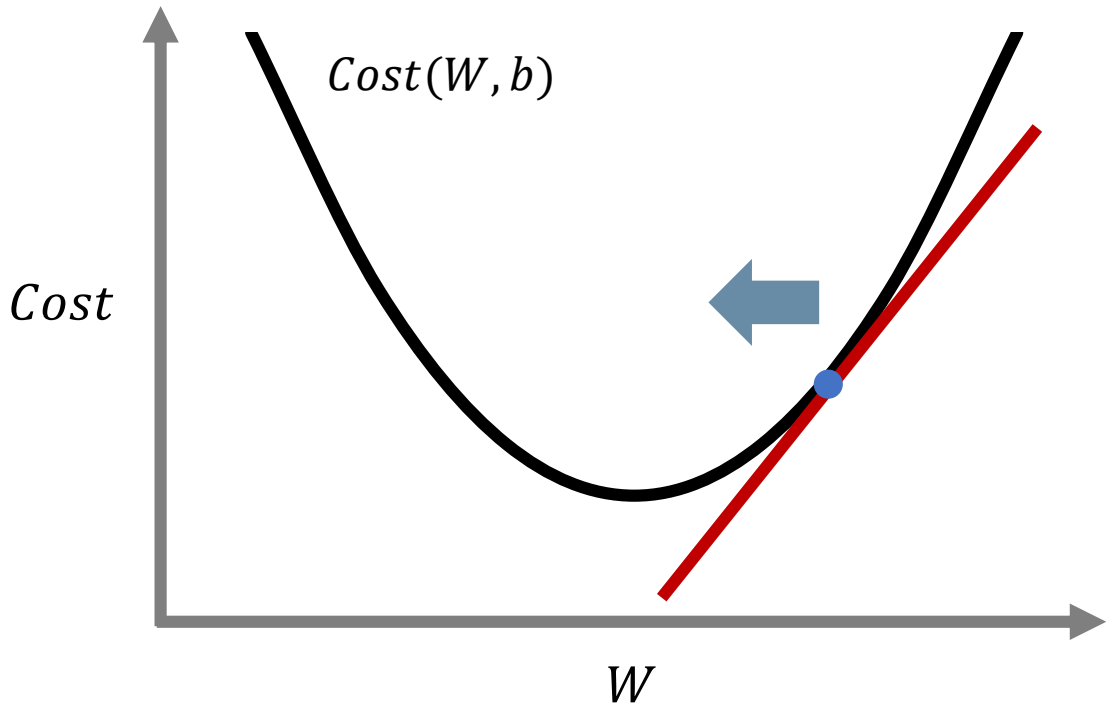
- 실제로 딥러닝 모델에서 **입력(input)**이나 **가중치(weight)** 값들이 다변수 벡터 형태입니다.
  - 따라서 딥러닝 모델에서 학습(training) 과정은 편미분을 통해 이루어집니다.



- 또한 딥러닝 모델은 다수의 레이어로 구성되어 있기 때문에 **연쇄법칙(chain-rule)**을 이용합니다.

경사 하강(Gradient Descent)

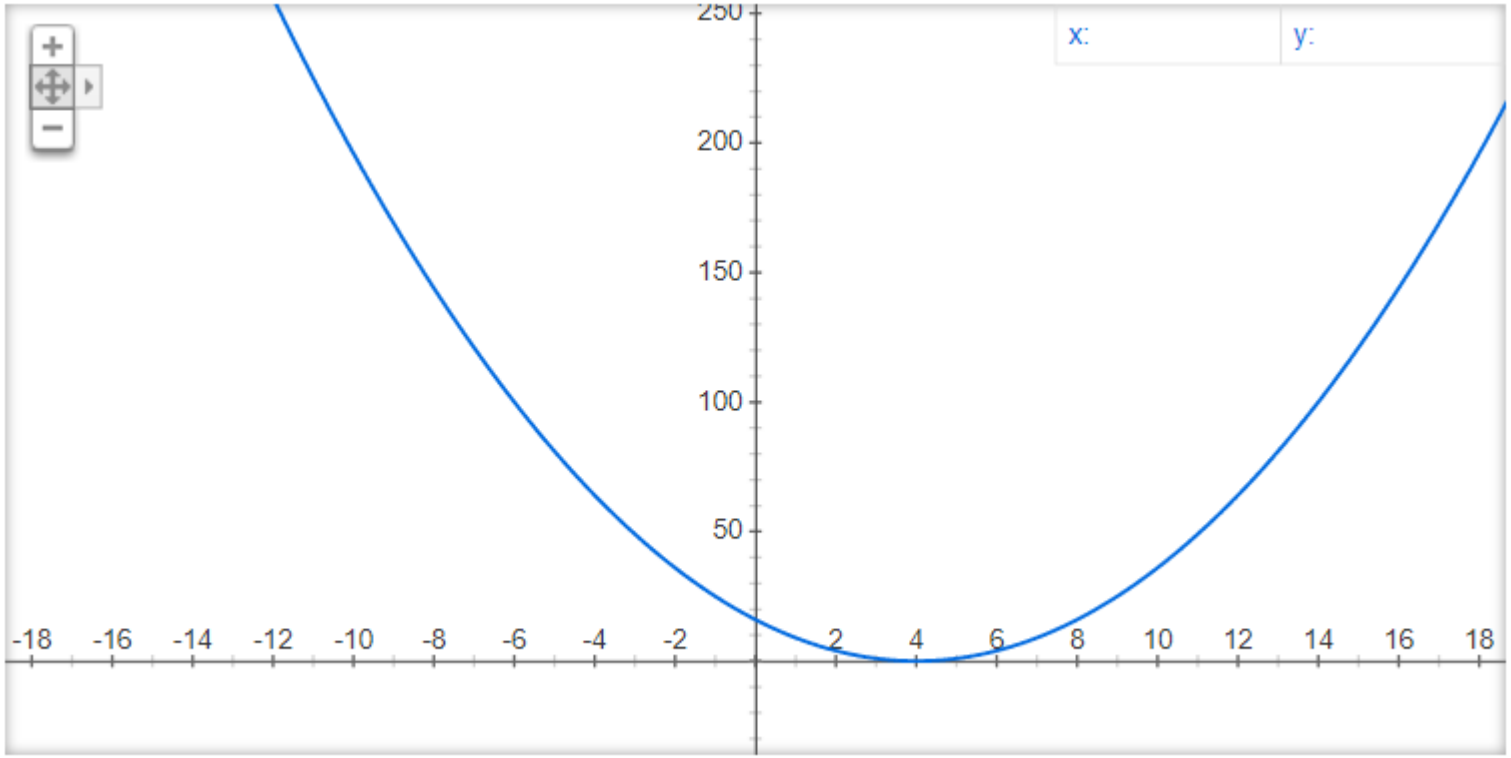
- 미분을 이용하면 특정 값에서의 기울기를 구할 수 있습니다.
- **경사 하강**: 기울기를 구하여 비용을 줄이는 방법입니다.



"현재 기울기가 양수(+)구나?
가중치를 음수(-) 방향으로 이동시키자!"

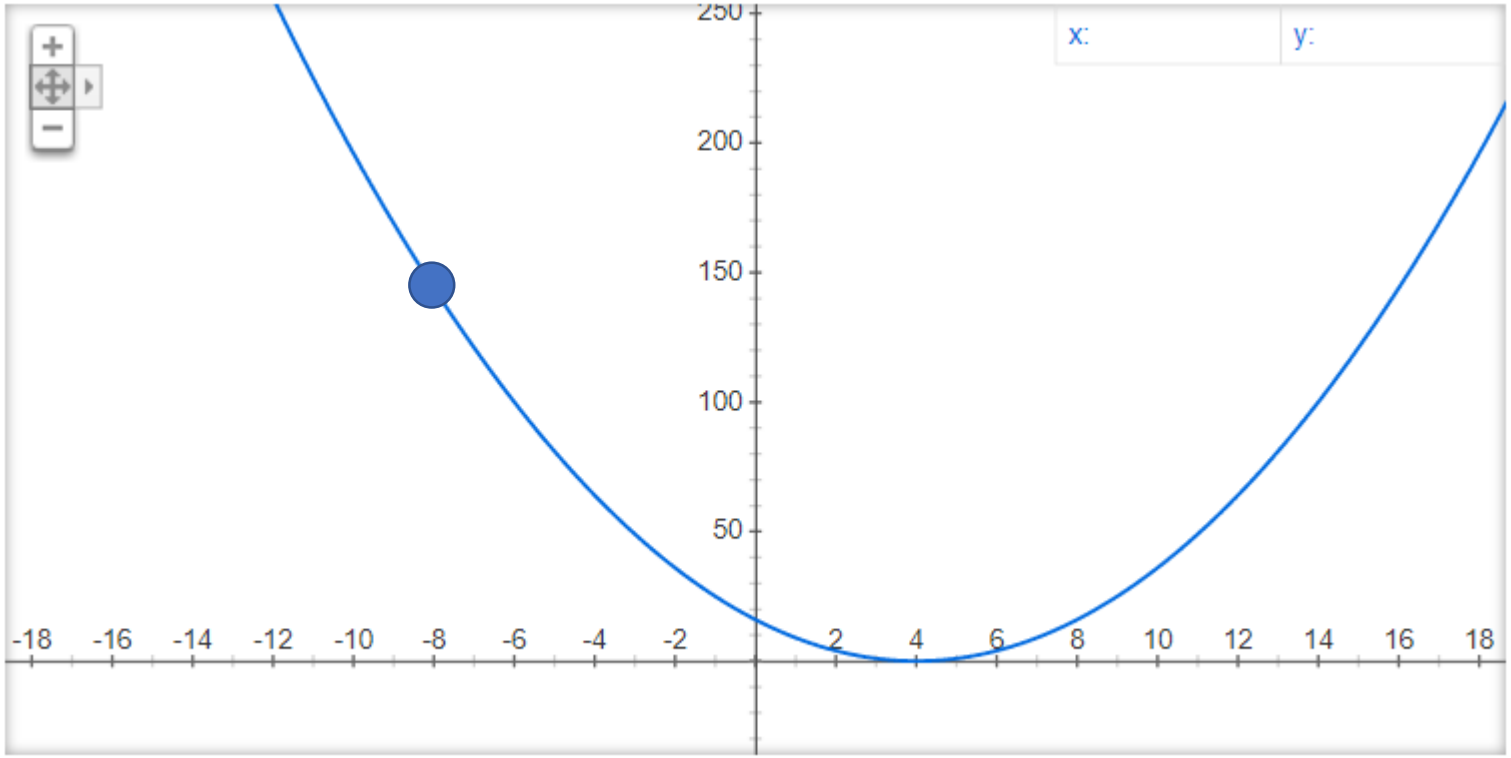
경사 하강(Gradient Descent)

- 그래프 $(x-4)^2$가 있다고 가정합시다. 가장 y 값이 작을 때를 찾고 싶습니다.



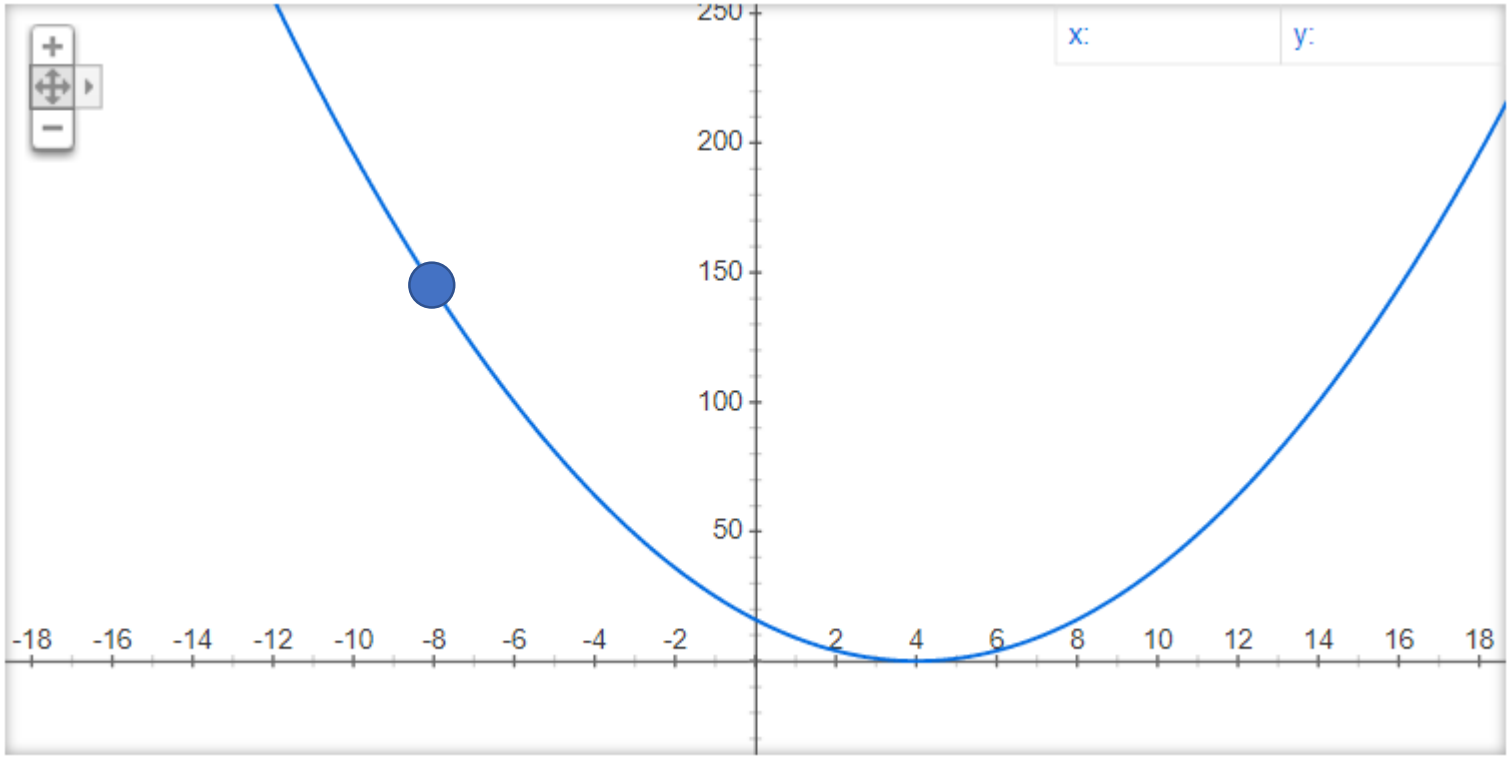
경사 하강(Gradient Descent)

- 예를 들어 $x = -8$일 때 y 값은 144입니다.



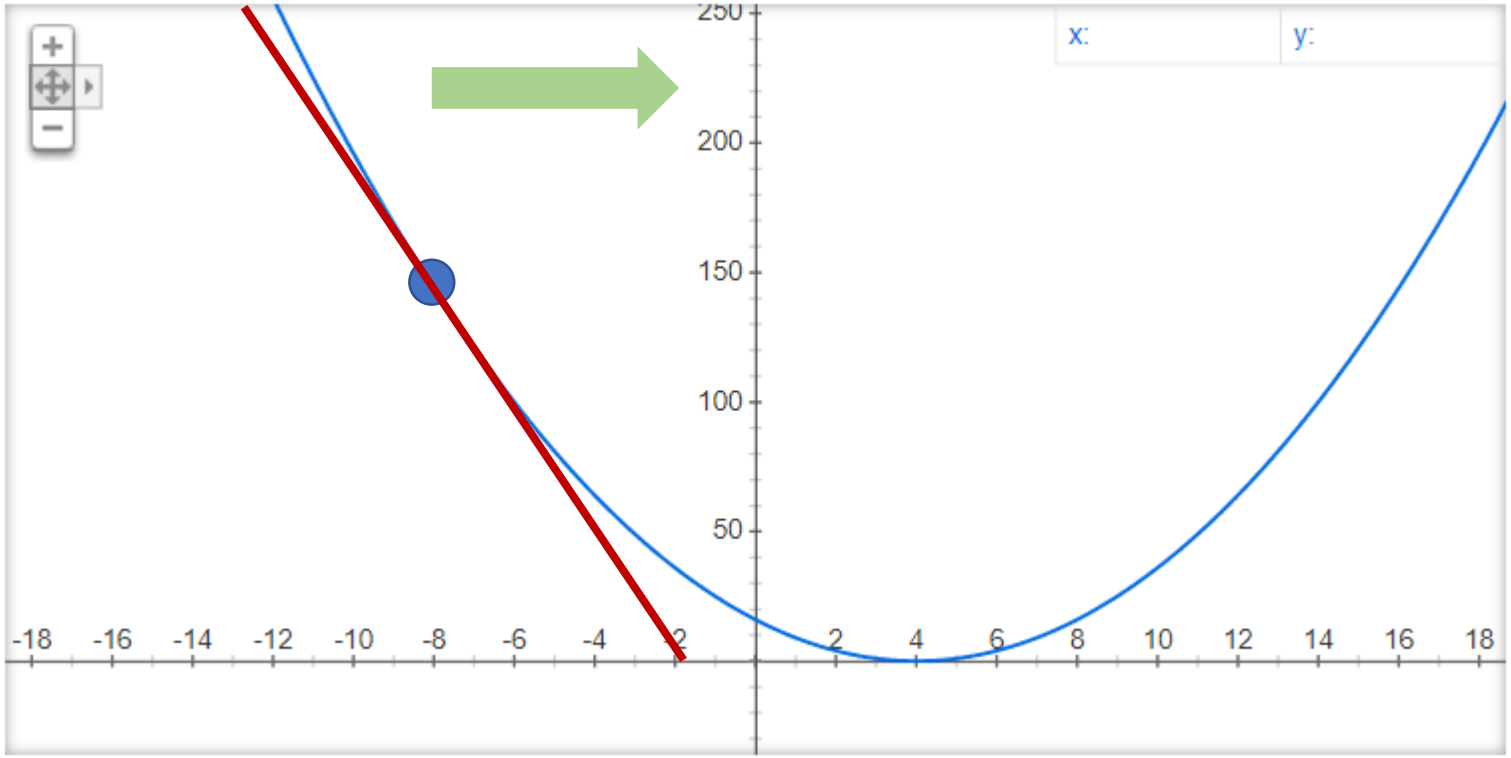
경사 하강(Gradient Descent)

- 도함수 $f'(x)$는 $2x - 8$입니다. 따라서 $x = -8$에서의 기울기는 -24입니다.



경사 하강(Gradient Descent)

- 기울기 값의 반대 방향으로 이동하면 y 값이 줄어들게 되므로 비용을 줄일 수 있습니다.

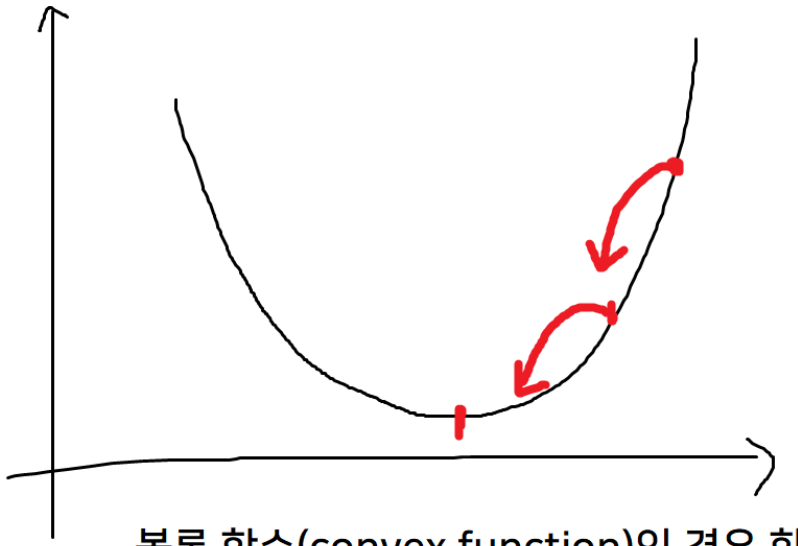


경사 하강(Gradient Descent)

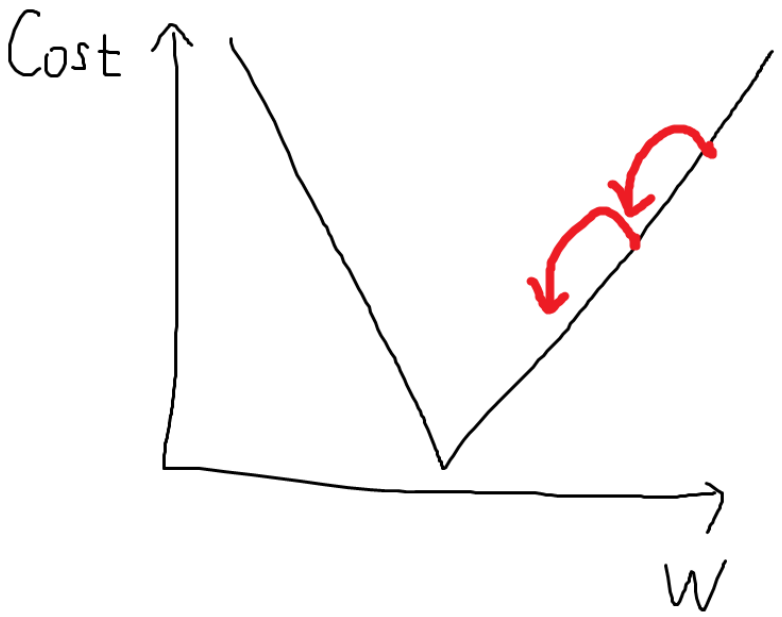
- 현재의 **기울기(gradient)**를 통해 업데이트 방향을 결정합니다.
- 실제 함수에서는 **여러 극소점(local minimum)이 존재**할 수 있습니다.
  - **극소점**: 기울기가 0인 낮은 지점을 의미합니다.
- **최소점(global minimum)**을 찾지 못해도(혹은 없어도) 최소점에 가까운 지점까지 비용을 낮추기 위해 파라미터를 업데이트합니다.

비용 함수(Cost Function)

- MSE (Mean Squared Error)



볼록 함수(convex function)인 경우 항상 경사 하강 (gradient descent)로 쉽게 해결할 수 있습니다.



선형 회귀 예시: H(x) = Wx

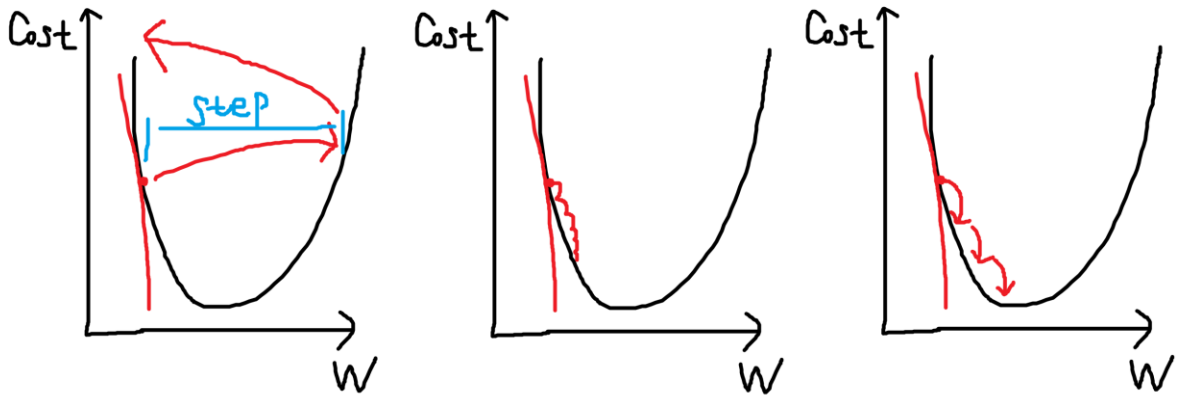
- 먼저 **간단히 $H(x) = Wx$를 고려**합시다.
- 비용 함수는 다음과 같습니다.

$$cost(W) = \frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)^2$$

- 그렇다면 $(Wx_i - y_i)^2$의 합을 어떻게 하면 가장 작게 할 수 있을까요?

선형 회귀 예시: H(x) = Wx

- **가중치(W)에 대한 비용 함수의 기울기**를 구한 뒤에 기울기의 반대 방향으로 W를 업데이트합니다.
- 어느 정도의 크기로 이동해야 할까요?
  - 예를 들어 $W = 4$에서 기울기가 7일 때, $W = -3$의 위치로 이동하는 것은 과할 수 있습니다.
  - 너무 많이 이동하면 튕겨 나갈 수 있으므로, 학습률(learning rate)을 곱하여 이동합니다.
- 학습률이 0.01이라면, $-7 * 0.01$만큼만 이동합니다.



선형 회귀 예시: H(x) = Wx

- 또한 미분을 수행하지 않고 **수치적으로 근사**하여 기울기를 계산할 수 있습니다.
- 파라미터 w가 있을 때 다음의 공식을 이용해 기울기(gradient)를 계산해 봅시다.

$$cost'(w) = \frac{dcost(W)}{dW} = \lim_{h \to 0} \frac{cost(W + h) - cost(W)}{h}$$

- dW 값은 0.001과 같이 작은 값으로 설정할 수 있습니다.
- **실제 딥러닝 프레임워크는 미분을 수행하여 계산**합니다.

선형 회귀 예시: H(x) = Wx

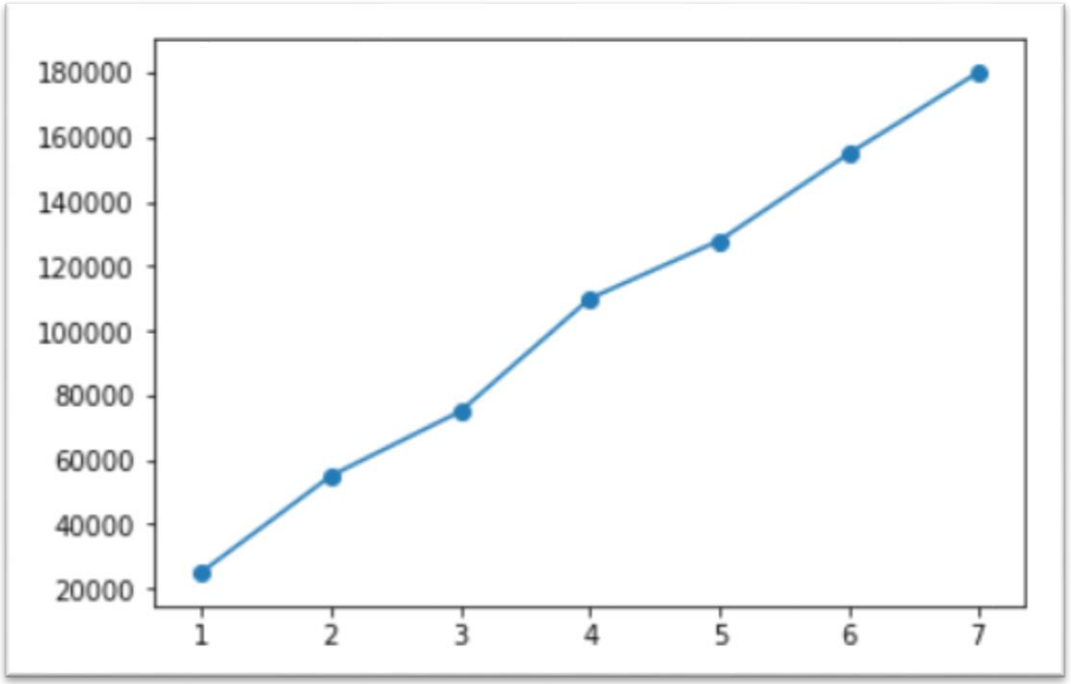
- 학습 목적의 데이터를 준비합니다.

하루 노동 시간	하루 매출
1	25,000
2	55,000
3	75,000
4	110,000
5	128,000
6	155,000
7	180,000

선형 회귀 예시: H(x) = Wx

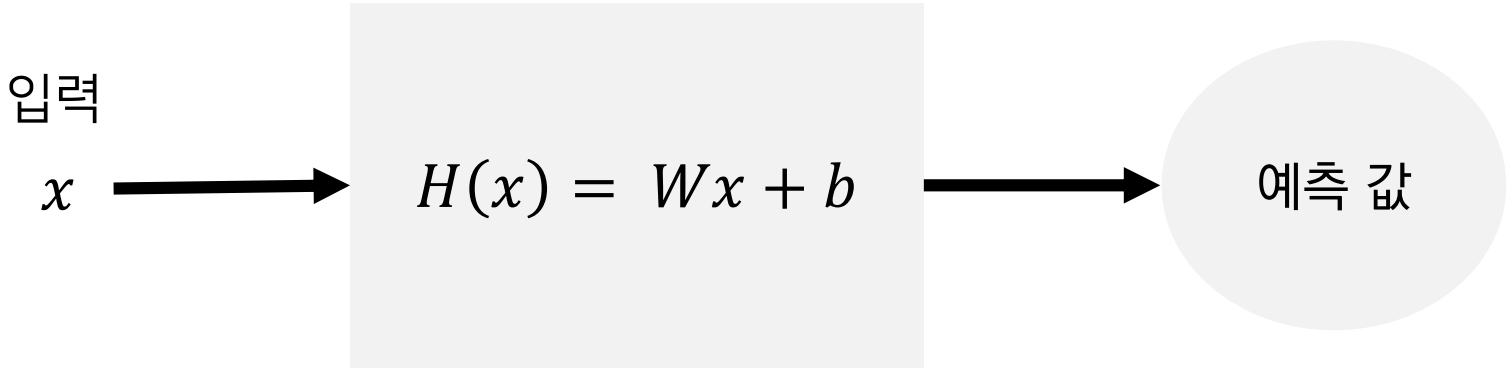
- **Python만을 이용한 구현**: 데이터 준비하기 (Linear Regression 바닥부터 구현)

```
import matplotlib.pyplot as plt

X = [1, 2, 3, 4, 5, 6, 7]
Y = [25000, 55000, 75000, 110000, 128000, 155000, 180000]

plt.plot(X, Y)
plt.scatter(X, Y)
```



선형 회귀 예시: H(x) = Wx

- **순방향(Forward)**: 모델에 입력을 넣어 결과를 출력하는 과정
  - 쉽게 말하면, 함수에 입력을 넣어서 결과를 구하는 과정



선형 회귀 예시: H(x) = Wx

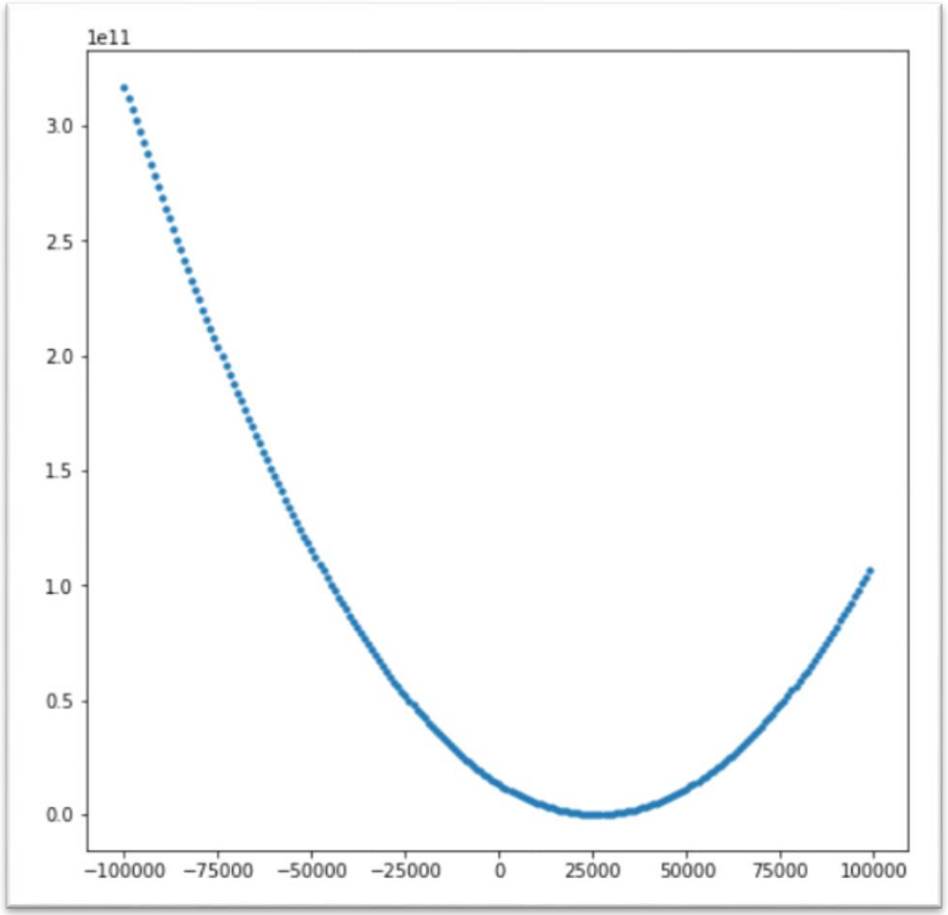
- **Python만을 이용한 구현**: 가설 클래스 정의하기

```
# 가설 모델(학습 시킬 대상)
class H():
    def __init__(self, w):
        self.w = w
    # 결과를 반환하는 함수
    def forward(self, x):
        return self.w * x
    # 가설의 비용을 구하는 함수 (낮추어야 할 대상)
    def get_cost(self, X, Y):
        cost = 0
        for i in range(len(X)):
            cost += (self.forward(X[i]) - Y[i]) ** 2
        cost = cost / len(X)
        return cost
```

선형 회귀 예시: H(x) = Wx

- **Python만을 이용한 구현**: 가중치(W)에 따른 비용 확인

```
cost_list = []
w_list = []

# w를 -300,000부터 300,000까지 바꾸어 보며 비용 확인
for i in range(-300, 300):
    w = i * 1000
    h = H(w)
    cost = h.get_cost(X, Y)
    w_list.append(w)
    cost_list.append(cost)

# 결과적으로 약 25,000 정도일 때 최소 비용임을 확인
plt.figure(figsize=(8, 8))
plt.scatter(w_list, cost_list, s=10)
```



선형 회귀 예시: H(x) = Wx

- **Python만을 이용한 구현**: 미분 없이 기울기 계산하기(모델 클래스)

```
# 기울기를 계산하는 함수
def get_gradient(self, X, Y):
    cost = self.get_cost(X, Y)
    dw = 0.001
    self.w = self.w + dw
    next_cost = self.get_cost(X, Y)
    self.w = self.w - dw
    dcost = next_cost - cost
    gradient = dcost / dw
    return gradient, next_cost
# w 값을 변경하는 함수
def set_w(self, w):
    self.w = w
# w 값을 반환하는 함수
def get_w(self):
    return self.w
```

선형 회귀 예시: H(x) = Wx

- **Python만을 이용한 구현**: 학습 진행하기

```
w = 4
h = H(w)
learning_rate = 0.001

for i in range(1001):
    gradient, cost = h.get_gradient(X, Y)
    h.set_w(h.get_w() + learning_rate * -gradient)
    if i % 100 == 0:
        print("[ epoch: %d, cost: %.2f ]" % (i, cost))
        print("w = %.2f, w_gradient = %.2f" % (h.get_w(), gradient))
```

선형 회귀 예시: H(x) = Wx

- Python만을 이용한 구현: 결과 예측하기

```
print("f(x) = %.2fx" %(h.get_w()))
print("예측값: [%.2f]" %(h.forward(8)))
```

선형 회귀 예시: H(x) = Wx

- 미분을 활용한 수식 간소화

$$W := W - \alpha \frac{\partial}{\partial W}(cost(W))$$

$$W := W - \alpha \frac{\partial}{\partial W}\left(\frac{1}{m}\sum_{i=1}^{m}(Wx_i - y_i)^2\right)$$

$$W := W - \alpha \frac{1}{m}\sum_{i=1}^{m} 2(Wx_i - y_i)\, x_i$$

$$W := W - \alpha \frac{2}{m}\sum_{i=1}^{m} (Wx_i - y_i)\, x_i$$

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

$(Wx_i - y_i)^2 = W^2x_i^2 - 2Wx_iy_i + y_i^2$

$\frac{\partial}{\partial W} = 2x_i^2W - 2x_iy_i$

$= 2(Wx_i - y_i)x_i$

$\nabla_W f(x) = x_i \quad (Wx_i - y_i)x_i$

$\nabla_W g'(x) = x_i$

선형 회귀 예시: H(x) = Wx

- **Python만을 이용한 구현**: 미분을 이용해 기울기 계산하기(모델 클래스)

```
# 미분으로 기울기를 계산하는 함수
def get_gradient_using_derivative(self, X, Y):
    gradient= 0
    for i in range(len(X)):
        gradient += (h.forward(X[i]) - Y[i]) * X[i]
    gradient = 2 * gradient / len(X)
    cost = self.get_cost(X, Y)
    return gradient, cost
```

선형 회귀 예시: H(x) = Wx + b

- 비용(Cost) 함수를 다음과 같이 원래 형태대로 W와 b를 모두 사용하는 방식으로 다시 정의합시다.

$$cost(W, b) = \frac{1}{m}\sum_{i=1}^{m}\left((Wx_i + b) - y_i\right)^2$$

- 업데이트 할 파라미터의 수가 2개가 되었습니다.
- 어떻게 미분할 수 있을까요?

선형 회귀 예시: H(x) = Wx + b

- 편미분을 활용한 수식 간소화

$$cost(W,b) = \frac{1}{m}\sum_{i=1}^{m}\left((Wx_i + b) - y_i\right)^2$$

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

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

$$W := W - \alpha\frac{2}{m}\sum_{i=1}^{m}(Wx_i + b - y_i)\,x_i$$

$$b := b - \alpha\frac{2}{m}\sum_{i=1}^{m}(Wx_i + b - y_i)$$

선형 회귀 예시: H(x) = Wx + b

- Python만을 이용해 구현해보기
  - Linear Regression 바닥부터 구현(Bias 포함) 실습
- PyTorch를 이용해 구현해보기

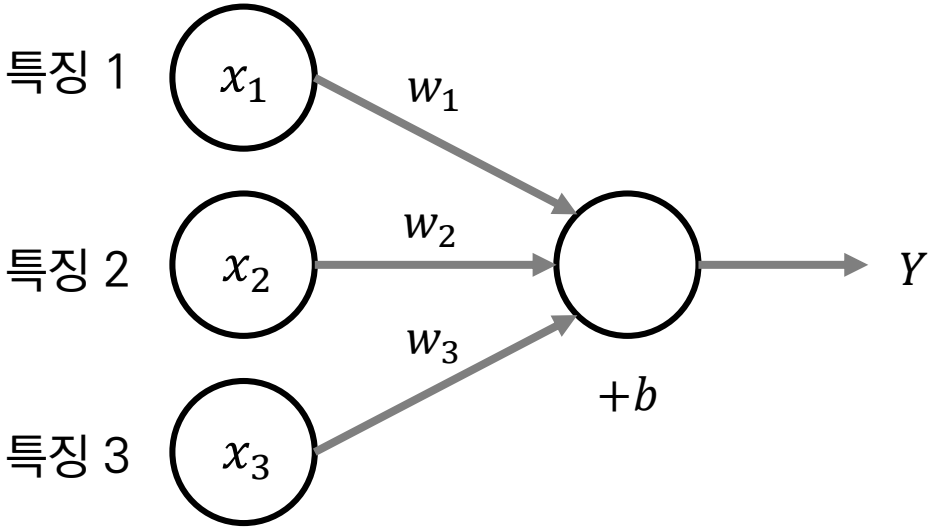
다변수 선형 회귀

- 현실 세계에서는 “**매출액**”을 결정하기 위한 다양한 변수가 존재합니다.
  1. 근무 시간
  2. 종업원의 수
  3. 매장의 크기

다변수 선형 회귀

- 다변수 선형 회귀에서는 변수가 여러 개입니다.

$$H(x) = Wx + b$$



$$H(x_1, x_2, x_3) = w_1\, x_1 + w_2\, x_2 + w_3\, x_3 + b \text{ (선형 함수)}$$

$$= \sum_i W_i x_i + b$$

다변수 선형 회귀

- 한 번에 여러 개의 입력이 들어온다면?
  - 행렬 곱을 이용하면 한 번의 연산으로 해결이 가능합니다.
  - 현대의 GPU는 이러한 행렬 곱 연산을 굉장히 효율적으로 수행할 수 있도록 해줍니다.

$$\text{데이터의 개수}\begin{bmatrix} \text{특징1} & \text{특징2} & \text{특징3} \\ x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \\ x_{31} & x_{32} & x_{33} \\ x_{41} & x_{42} & x_{43} \end{bmatrix} * \begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} = \begin{pmatrix} x_{11}w_1 + x_{12}w_2 + x_{13}w_3 \\ x_{21}w_1 + x_{22}w_2 + x_{23}w_3 \\ x_{31}w_1 + x_{32}w_2 + x_{33}w_3 \\ x_{41}w_1 + x_{42}w_2 + x_{43}w_3 \end{pmatrix}$$

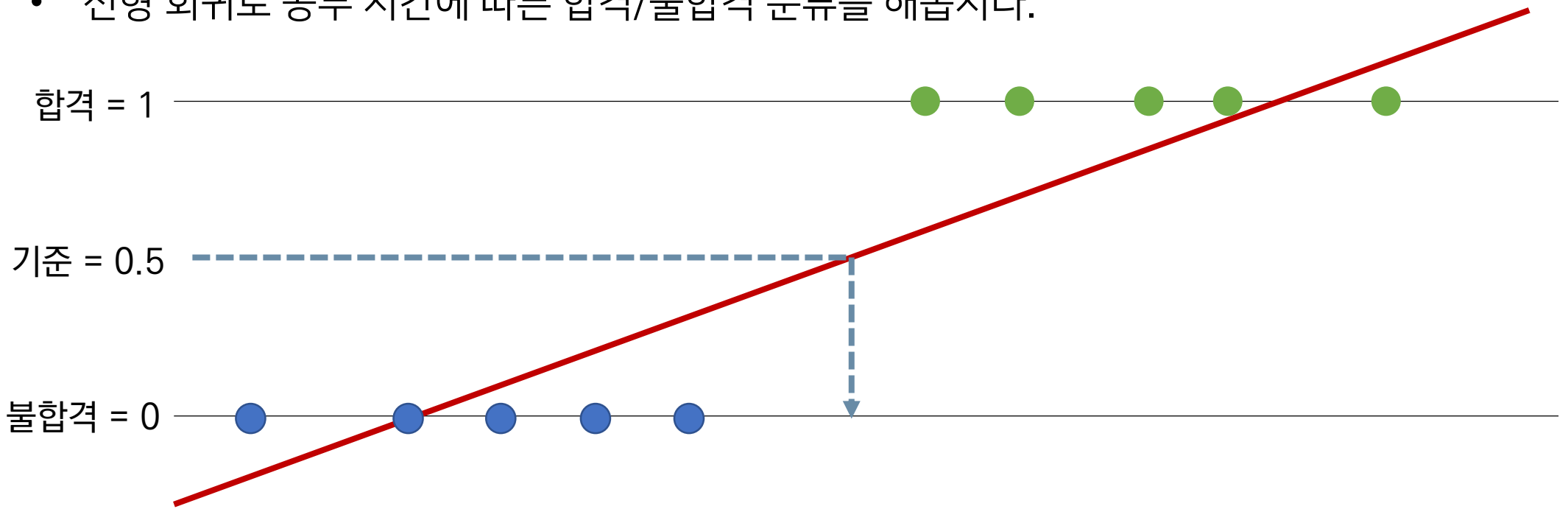
$$H(X) = XW$$

다변수 선형 회귀

- PyTorch를 이용해 구현해보기
- 실습 코드 살펴보기
  - 다변수 선형 회귀(multivariable linear regression) 구현

분류(Classification) 문제

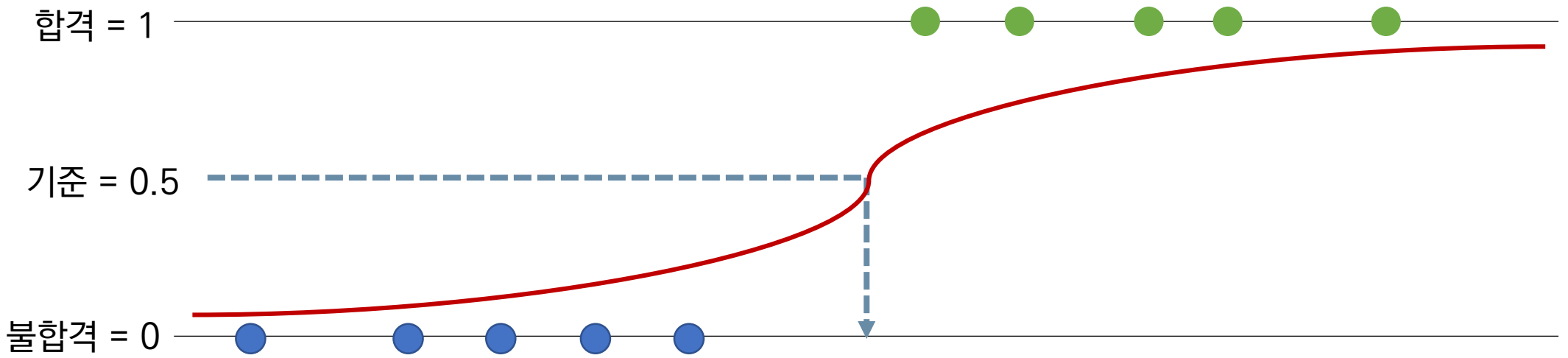
- 앞서 공부한 선형 회귀를 이용하여 분류 문제를 해결할 수 있을까요?
  - 선형 회귀로 공부 시간에 따른 합격/불합격 분류를 해봅시다.



- 결과 값을 0부터 1사이로 제한해야 분류(Classification) 문제를 해결하기에 효과적일 것입니다.

로지스틱 회귀(Logistic Regression)

- 회귀를 사용하여 데이터가 어떤 클래스에 속할 확률을 0부터 1사이의 값으로 예측합니다.
  - 예시: 공부 시간에 따른 합격/불합격 분류기
  - 예시: 이미지 특징에 따른 강아지/고양이 분류기

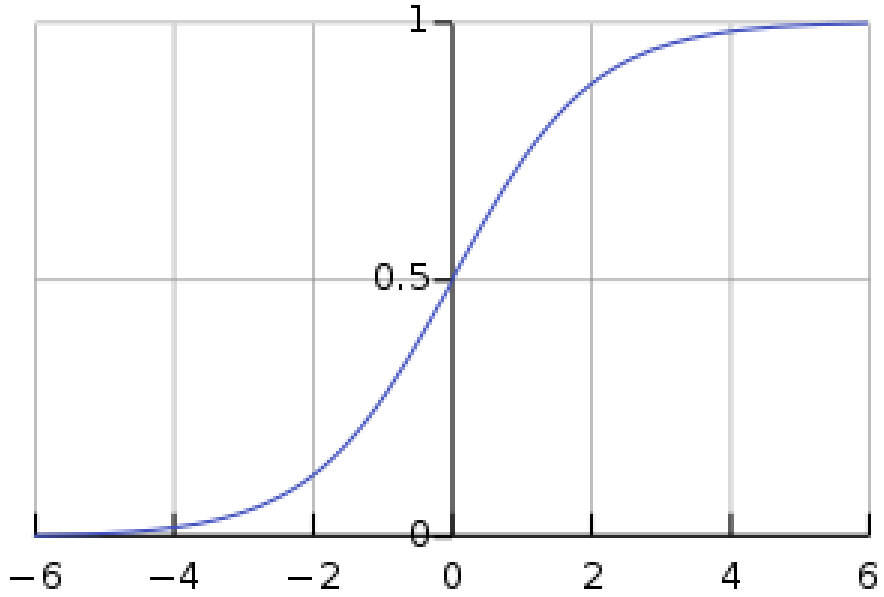


이진 분류와 Sigmoid

- Sigmoid 함수 (Logistic 함수)는 다음과 같습니다.

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

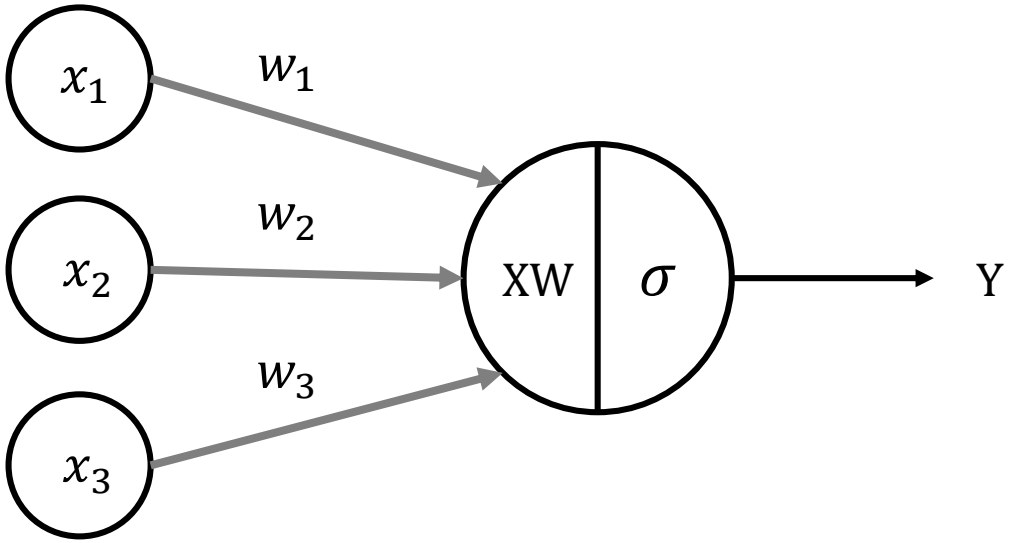
  - 모든 위치에서 **미분이 가능**합니다.
  - 0부터 1 사이의 확률 값을 반환합니다.
  - 로지스틱 회귀(Logistic Regression)에 적합합니다.



이진 분류와 Sigmoid

- 실제 결과 값을 뽑기 전에 Sigmoid 함수에 넣으면 어떨까요?

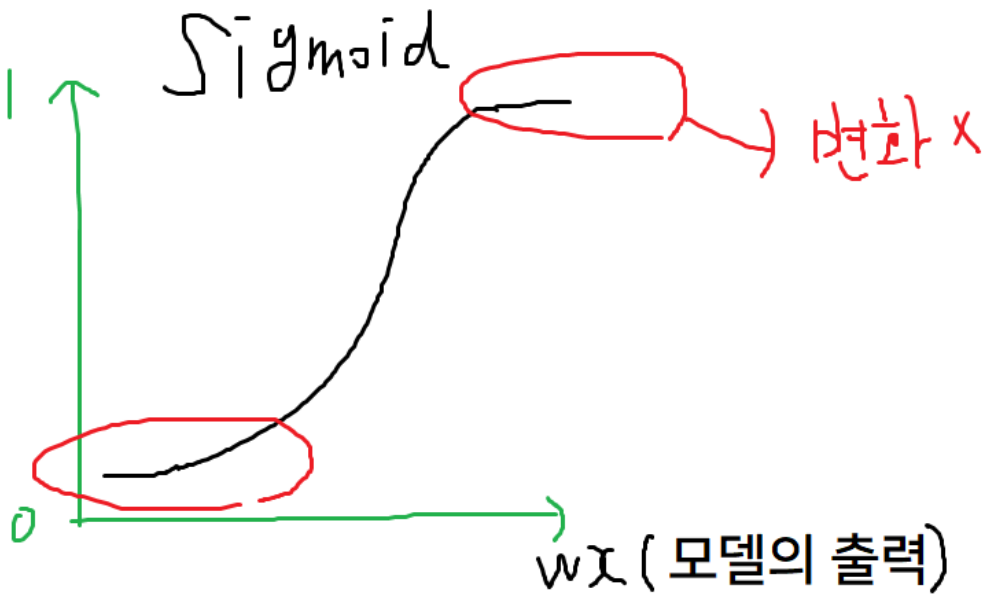
$$H(X) = \frac{1}{1 + e^{-XW}}$$

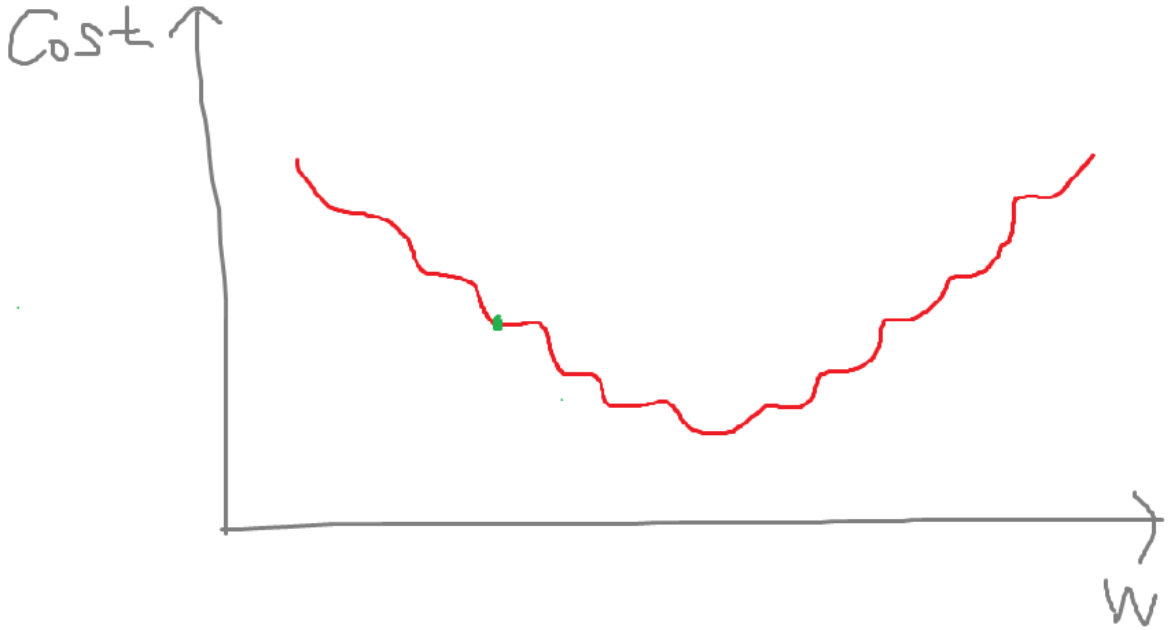


- 결과는 항상 0부터 1 사이의 확률 값으로 한정됩니다.
- 다만 기존에 사용했던 MSE 비용 함수를 그대로 사용하기 어렵습니다. (non-convexity)

이진 분류와 Sigmoid

- 다만 기존에 사용했던 MSE 비용 함수를 그대로 사용하기 어렵습니다. (non-convexity)



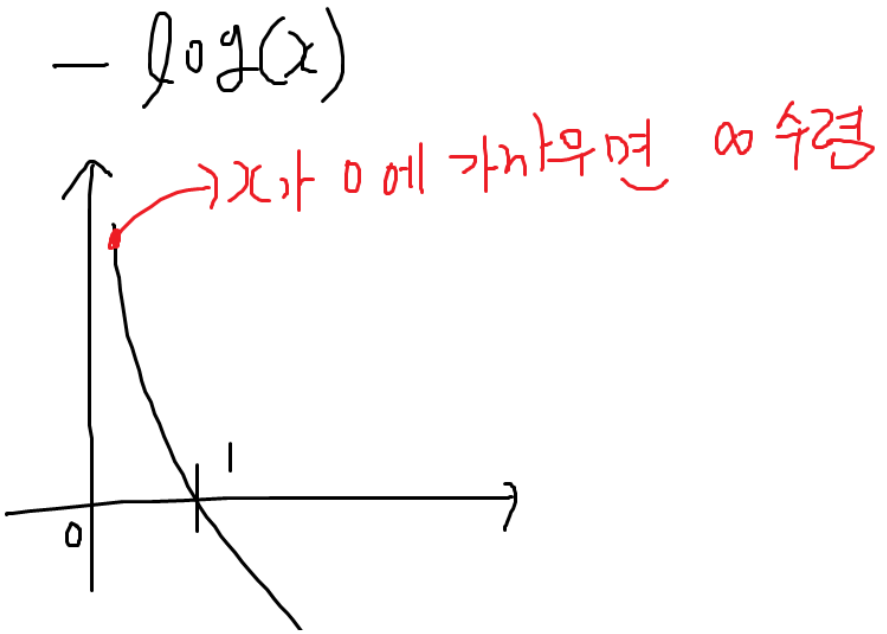


엔트로피(Entropy)

- Entropy는 특정 시스템이 얼마나 불안정한지 알려주는 근거로 사용될 수 있습니다.

$$H(P,Q) = -\sum P(x)\log(Q(x))$$

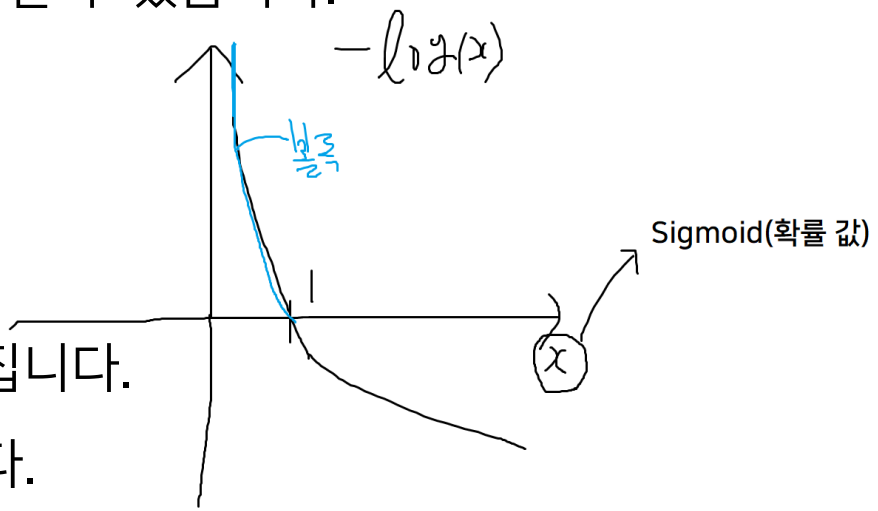
- $P(x)$: 실제 확률
- $Q(x)$: 예측 확률
- 예를 들어 예측 확률이 0.1이고, 실제 확률이 1이라고 해봅시다.
  - 이는 예측이 많이 틀린 것입니다.
  - 따라서 매우 큰 값(정보량)이 도출됩니다.



Logistic Regression의 비용 함수

- 볼록 함수(convex function) 형태를 보장할 수 있는 비용 함수를 새롭게 정의합니다.
- $y = 1$일 때와 $y = 0$일 때를 나누어서 분류 문제의 비용 함수를 설정할 수 있습니다.

$$c(H(x), y) = \begin{cases} -\log\big(H(x)\big) & : y = 1 \\ -\log\big(1 - H(x)\big) & : y = 0 \end{cases}$$

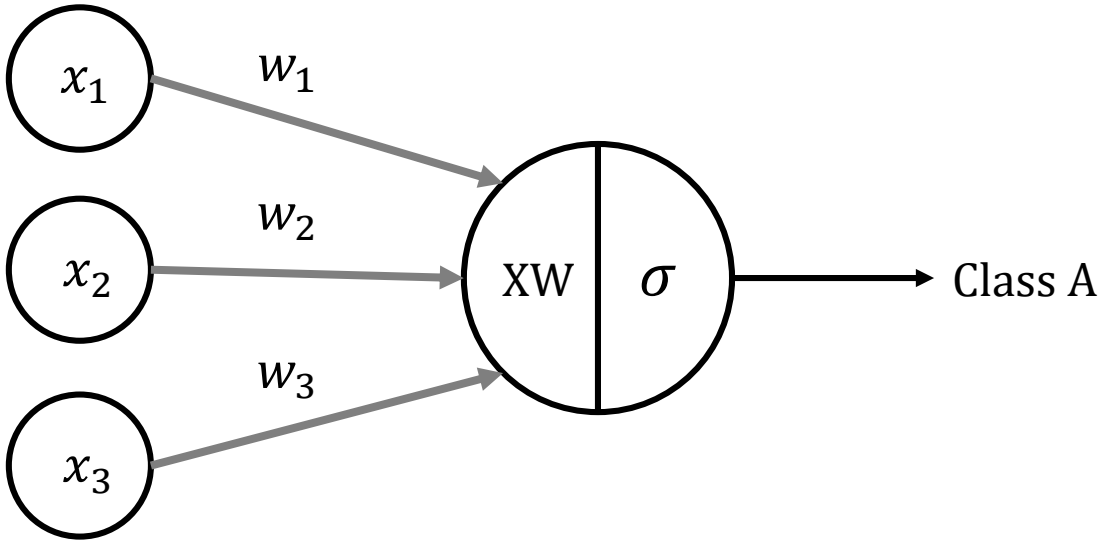


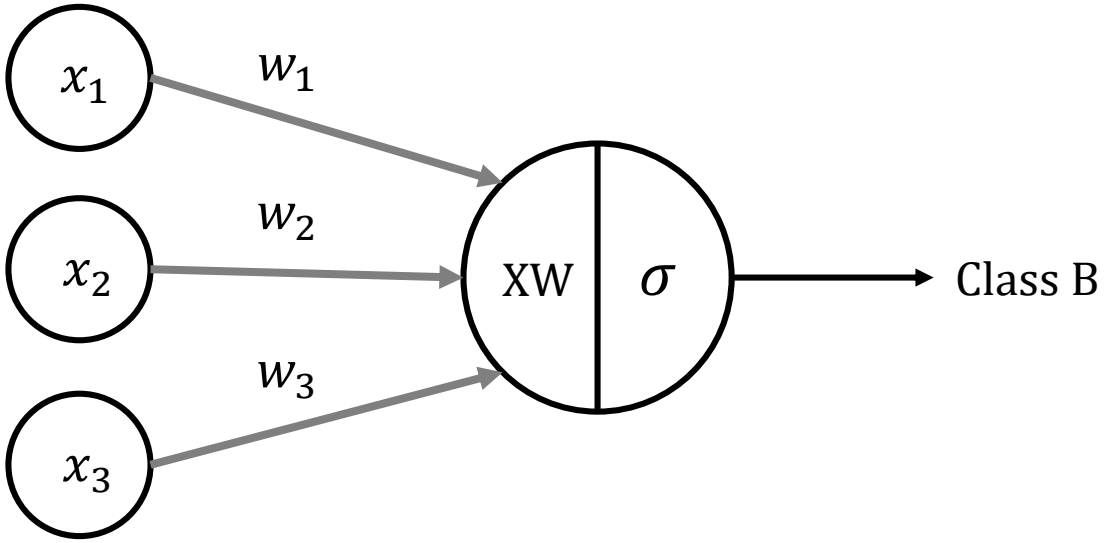
- 만약 판단 결과가 틀리게 되면 log 함수에 의하여 큰 피드백이 가해집니다.
- 결과적으로 비용 함수는 조건식 없이 다음과 같은 형태로 사용합니다.

$$c(H(x), y) = -ylog\big(H(x)\big) - (1 - y)\log(1 - H(x))$$

Multinomial Classification

- 클래스가 여러 개인 문제를 해결하는 방법을 알아봅시다.
  - 각각의 클래스 마다 별도의 모델을 두어서 확률 값을 각각 구하는 아이디어를 사용할 수 있을까요?





Multinomial Classification

- 클래스가 여러 개인 문제를 해결하는 방법을 알아봅시다.
  - 행렬 곱을 이용하면 하나의 네트워크만을 이용할 수 있습니다.

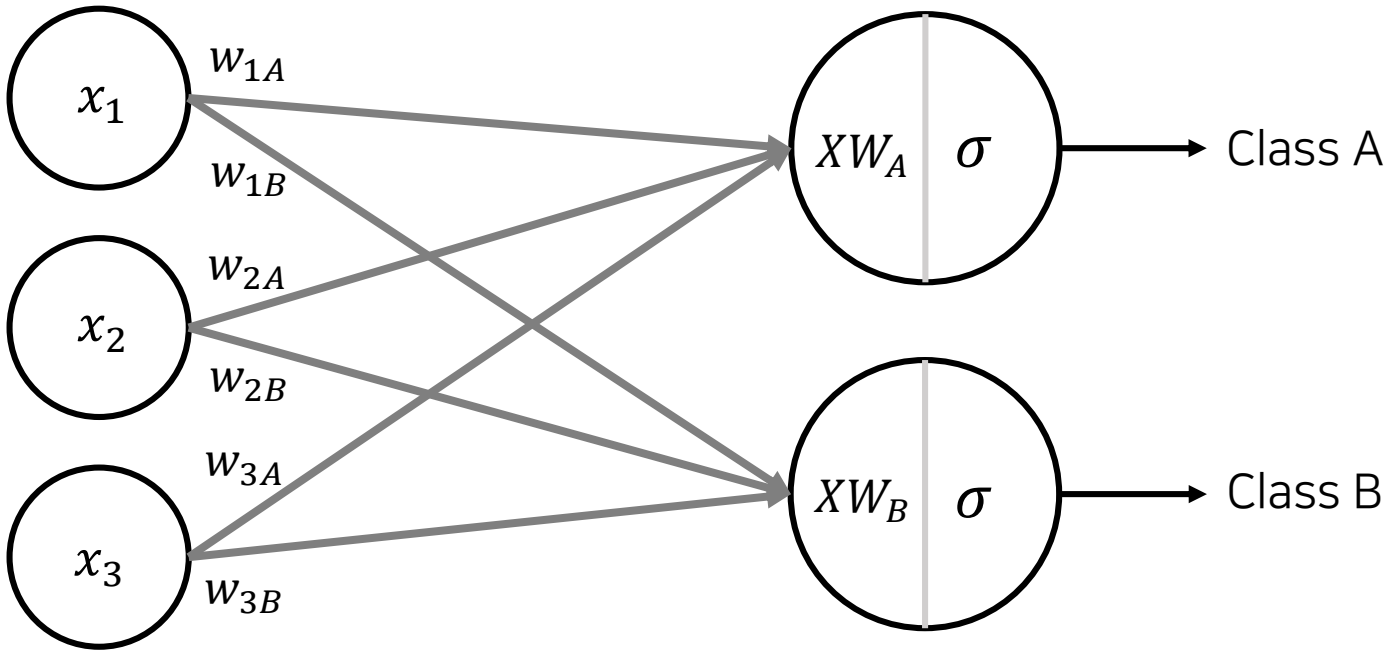
$$\underset{\text{특징1}\ \ \text{특징2}\ \ \text{특징3}}{H(x_1 \quad x_2 \quad x_3\ )} * \begin{pmatrix} w_{1A} & w_{1B} \\ w_{2A} & w_{2B} \\ w_{3A} & w_{3B} \end{pmatrix} = \underset{\text{Class A}}{(x_1 w_{1A} + x_2 w_{2A} + x_3 w_{3A}} \quad \underset{\text{Class B}}{x_1 w_{1B} + x_2 w_{2B} + x_3 w_{3B})}$$

$$H(X) = \frac{1}{1 + e^{-XW}} = (s(x_1 w_{1A} + x_2 w_{2A} + x_3 w_{3A}) \quad s(x_1 w_{1B} + x_2 w_{2B} + x_3 w_{3B}))$$

- 단 Sigmoid 함수를 각 클래스마다 별도로 취해야 할까요?

Multinomial Classification

- 클래스가 여러 개인 문제를 해결하는 방법을 알아봅시다.



Sigmoid를 쓴다면?

- 아웃풋 뉴런(클래스) 각각 0부터 1사이의 값을 가집니다.
- 모델의 결과를 모두 합쳤을 때 1이 되면 더 좋을 것입니다.
- 예시) 이미지가 들어왔을 때
  - 고양이: 70%
  - 강아지: 30%

Softmax Function

- 모델의 결과 확률을 모두 합한 값이 1이 되도록 만들기 위해 Softmax를 사용합니다.
  - 아래 예시는 클래스의 개수가 5개인 상황을 가정합니다.

Logits (모델의 최종 출력)

$$\begin{bmatrix} 4.3 \\ 5.5 \\ 7.2 \\ 2.4 \\ 5.0 \end{bmatrix}$$

Softmax

$$\frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$$

Probabilities

$$\begin{bmatrix} 0.04 \\ 0.13 \\ 0.74 \\ 0.01 \\ 0.08 \end{bmatrix}$$

- Binary classification → Sigmoid를 사용
- Multinomial classification → Softmax 사용

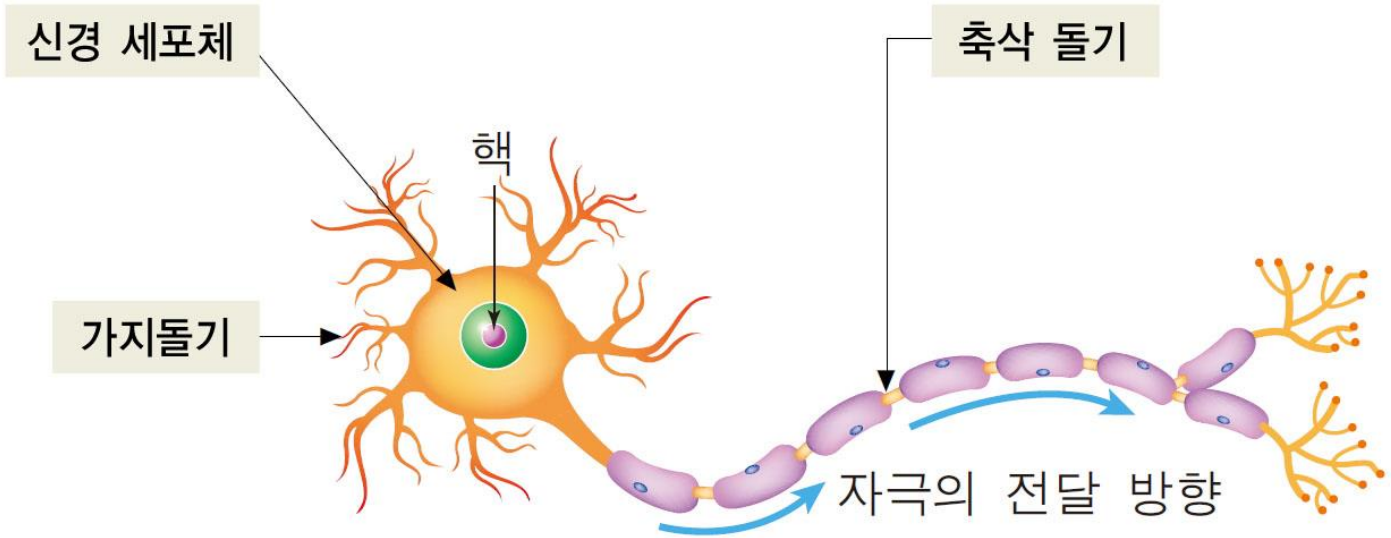
Cross-entropy 비용 함수

- 마지막 레이어에서 Softmax를 사용할 때 비용 함수를 어떻게 설정할 수 있을까요?
  - **크로스 엔트로피(Cross-entropy)** 비용 함수를 이용합니다.

$$S = \begin{bmatrix} 0.04 \\ 0.13 \\ 0.74 \\ 0.01 \\ 0.08 \end{bmatrix} \qquad CrossEntropy(S, L) = -\sum_i L_i \log(S_i) \qquad L = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

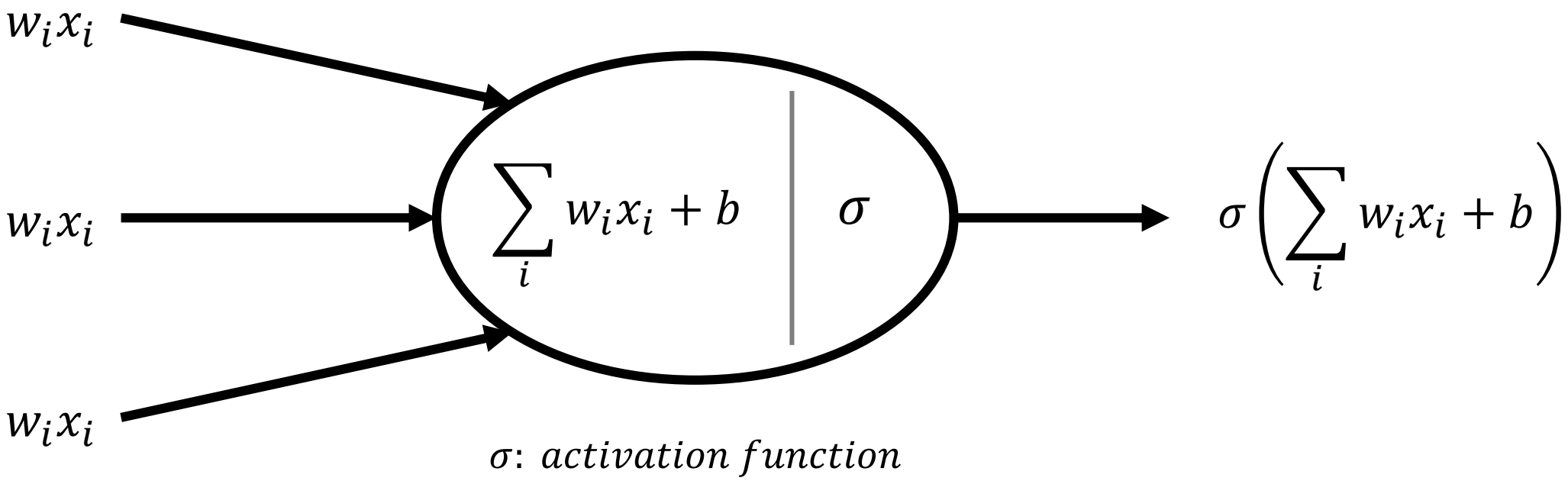
딥러닝 개요: 뉴런(Neuron)

- 뉴런은 **뇌를 구성하는 기본 단위**입니다.
- 사람의 뇌는 1,000억 개 이상의 뉴런으로 구성된 복잡한 회로와 같습니다.



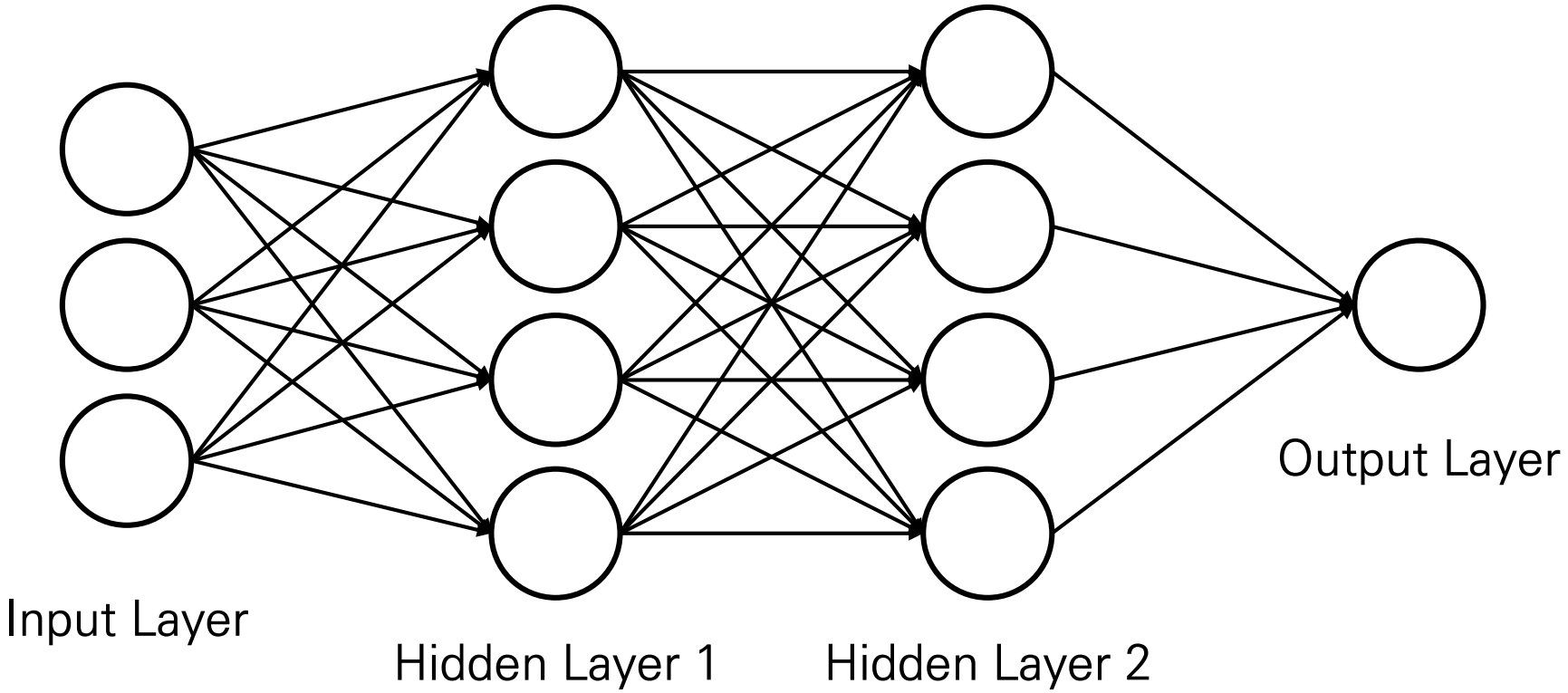
딥러닝 개요: 뉴런(Neuron)

- 하나의 뉴런 출력 값을 수학적으로 모델링할 수 있습니다. (그냥 앞서 확인했던 선형 함수)
  - [핵심] 활성화 함수(Sigmoid 등)를 이용하여 모델 전체에 **비선형성(non-linearity)**을 추가합니다.



Universal Approximation Theorem (무엇이든 근사가 가능하다)

- **하나 이상의 은닉층(hidden layer)**를 가지고 비선형 활성화 함수를 가진 뉴럴 네트워크는 임의의 연속인 다변수 함수(고차 함수도 가능)를 근사할 수 있습니다. (1989, G. Cybenko)



딥러닝(심층 신경망)이 하는 역할이 무엇일까요?

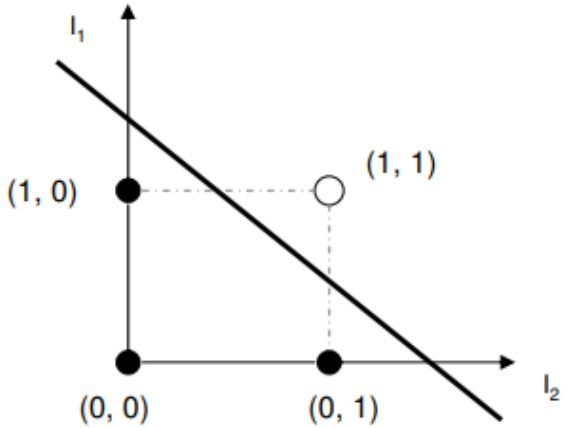
"특정한 함수를 (수학적으로) 근사할 수 있는 기계"

- 딥러닝 모델은 **입력**을 넣었을 때 적절한 **출력**이 무엇인지 맞히는 방식으로 동작합니다.

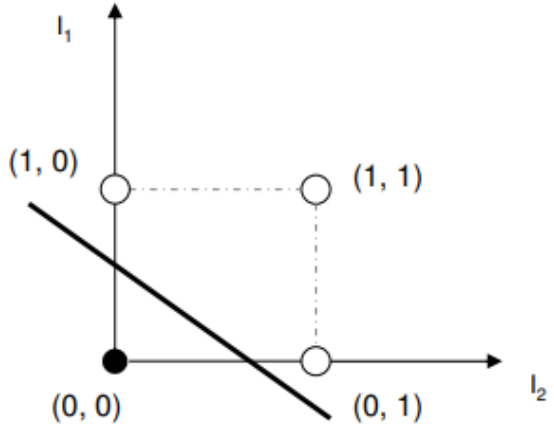
- 세상은 우리가 설명하지 못하는 다양한 **함수**로 구성되며, 딥러닝은 이것을 근사할 수 있도록 합니다.

XOR 문제

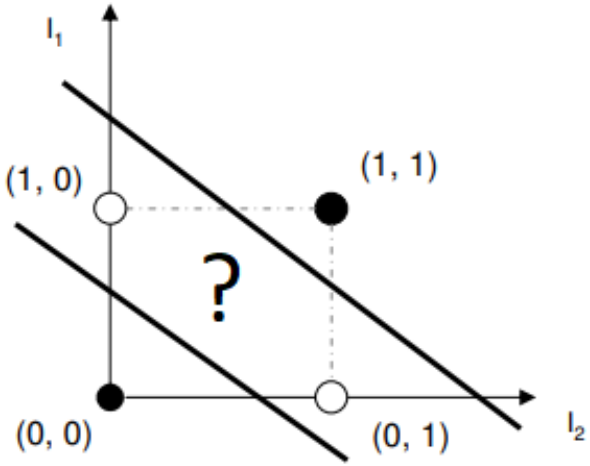
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



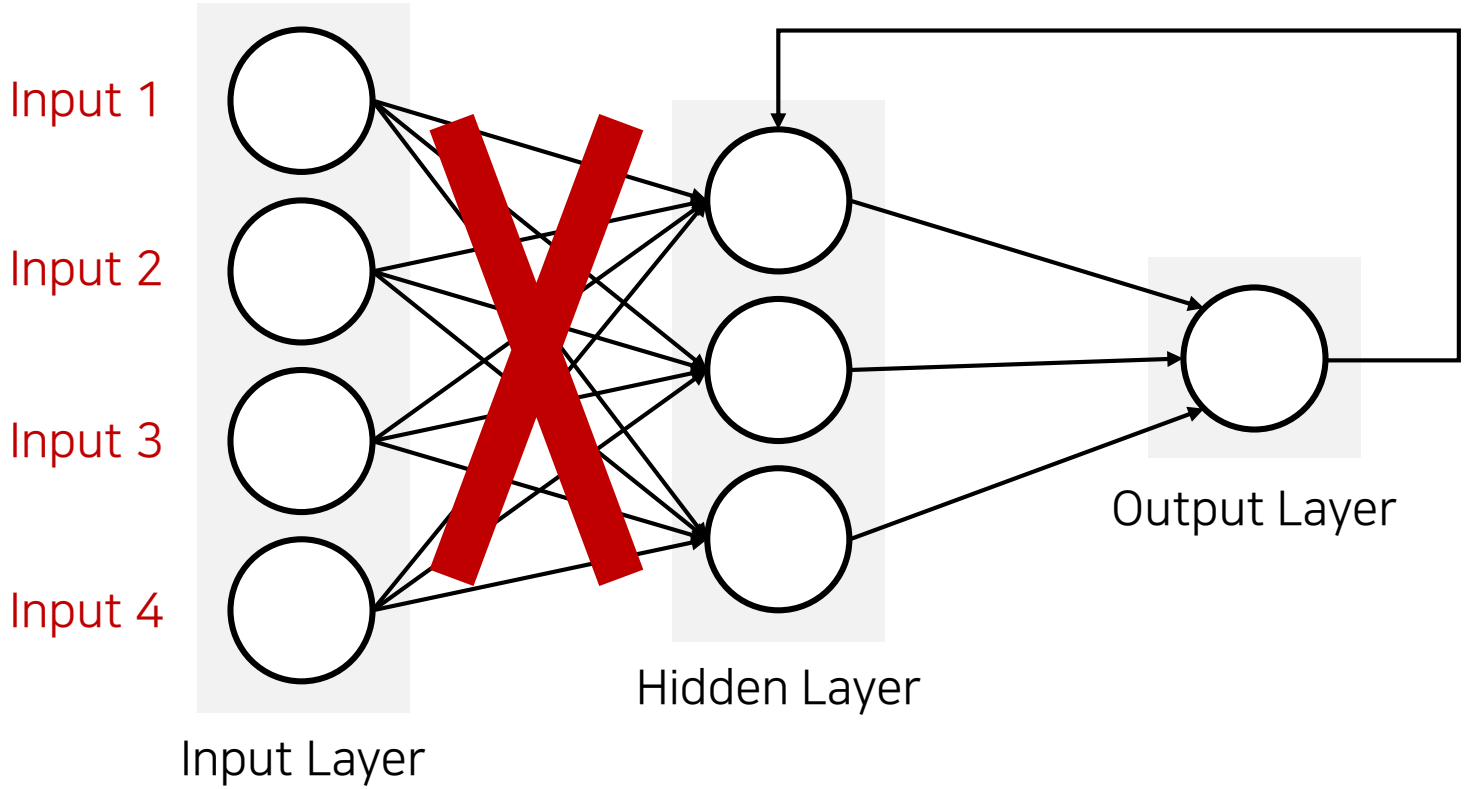
XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0



- 이는 **선형적으로 분리 되지 않는 문제**입니다.
- 더불어 1960년대에는 뉴럴 네트워크를 깊게 쌓아 올리더라도 앞쪽 레이어를 학습할 방법이 없었습니다.
- **레이어를 깊게 쌓고, 비선형 활성화 함수를 이용해 비선형적인 분류 모델을 학습하 수 있게 되어 해결 가능.**

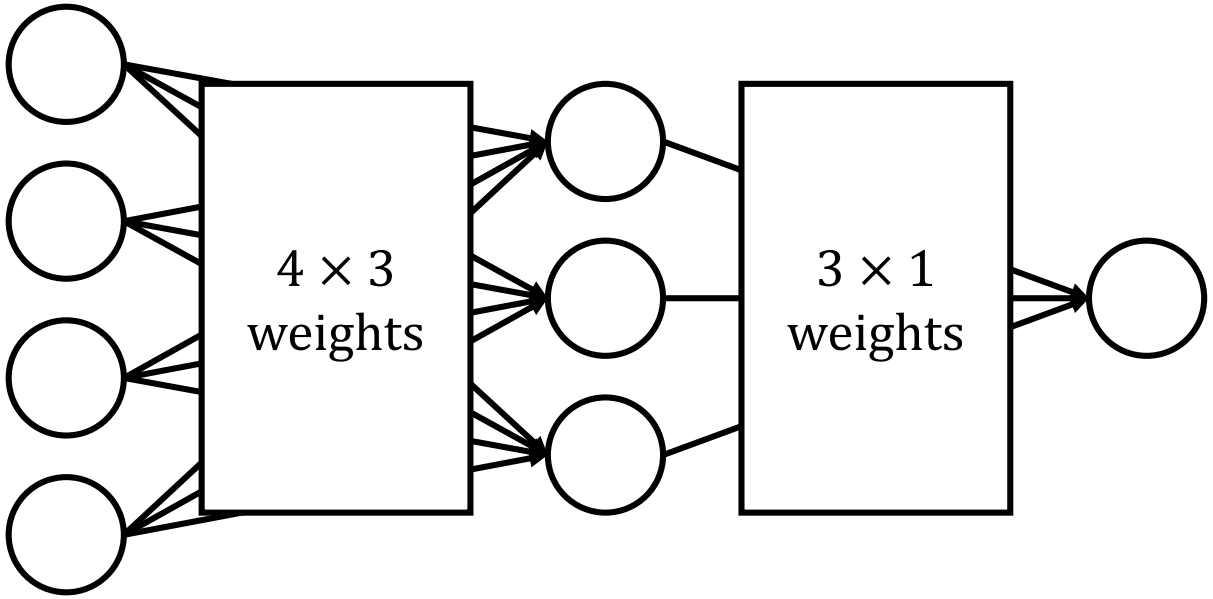
역전파(Backward-propagation)

- 앞에서 뒤로만 순전파(Forward-propagation)를 시키는 경우 앞쪽의 가중치는 학습되지 않습니다.
- 따라서 역전파(Backward-propagation)가 필요합니다.



합성함수의 미분(Chain-rule)

- 인공지능 모델은 여러 개의 **다수의 레이어**로 구성되어 있습니다.
- 그러므로 손실 함수(loss function)의 기울기를 계산하는 것은 합성함수를 미분하는 것과 같습니다.



- 따라서 **합성함수의 미분법**은 인공지능 분야에서 매우 중요합니다.

합성함수의 미분(Chain-rule)

- 합성함수의 도함수는 다음과 같습니다.

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

- 도함수의 정의를 이용해 합성함수의 미분 결과를 유도할 수 있습니다.

$$f\big(g(x)\big)' = \lim_{h\to 0}\frac{f\big(g(x+h)\big) - f\big(g(x)\big)}{h}$$

$$= \lim_{h\to 0}\frac{f\big(g(x+h)\big) - f\big(g(x)\big)}{g(x+h)-g(x)} * \frac{g(x+h)-g(x)}{h}$$

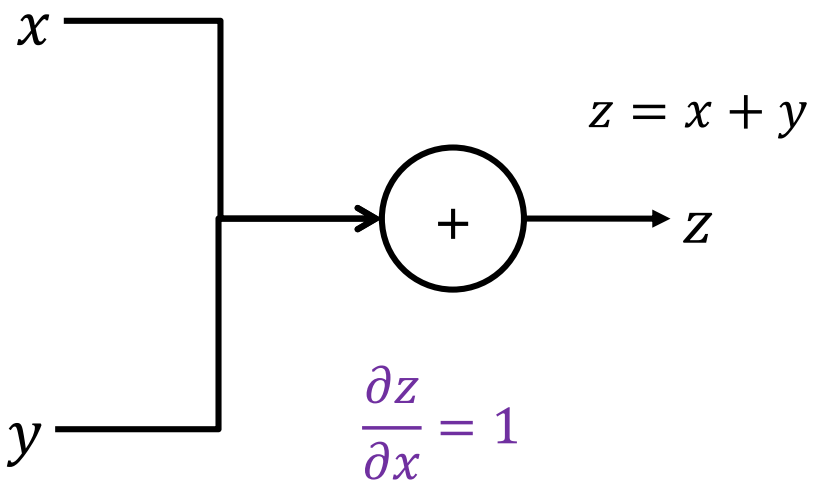
$$= f'\big(g(x)\big)g'(x)$$

$z = g\big(f(x)\big)$의 미분법은?

$$\frac{dz}{dx} = \frac{dz}{dy} * \frac{dy}{dx}$$

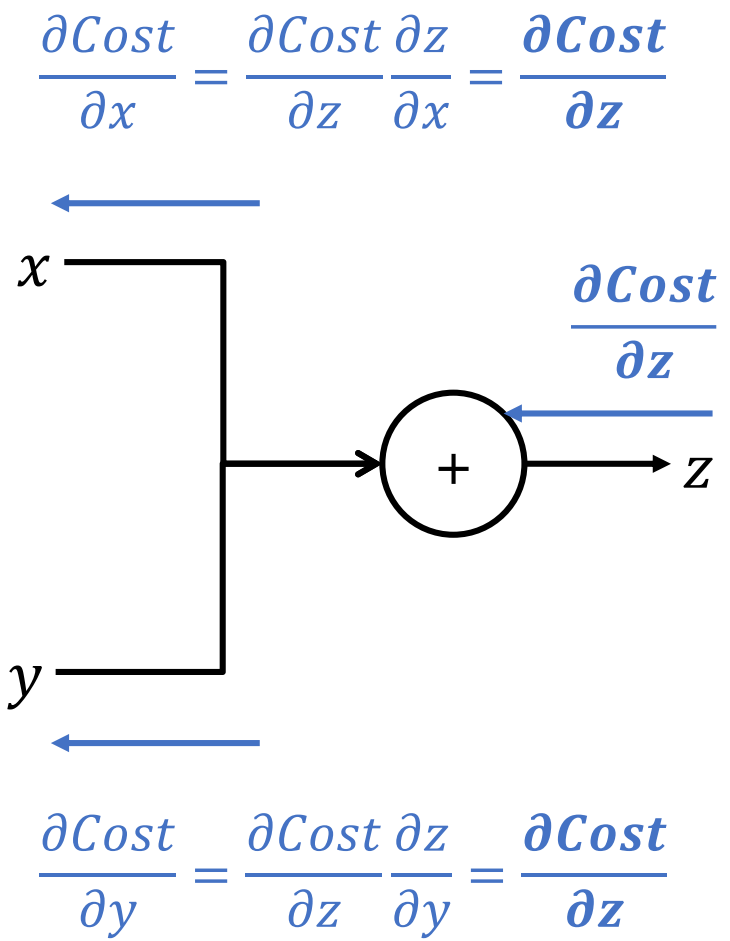
합성함수의 미분(Chain-rule)

- **더하기** 노드에 대한 편미분



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

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



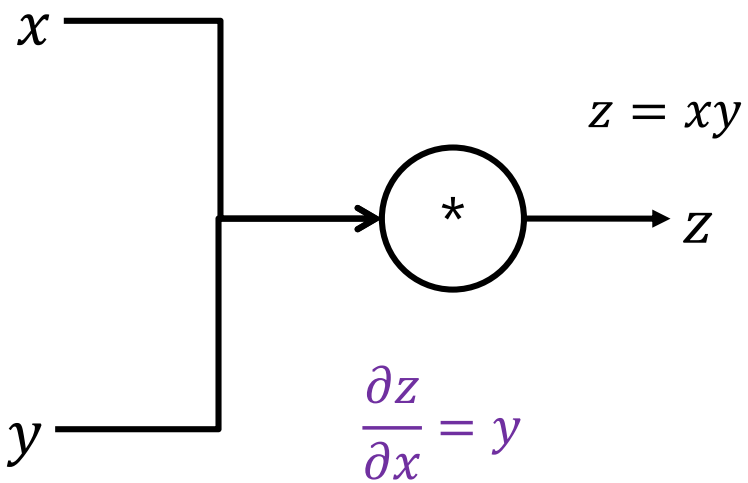
$$\frac{\partial Cost}{\partial x} = \frac{\partial Cost}{\partial z}\frac{\partial z}{\partial x} = \boldsymbol{\frac{\partial Cost}{\partial z}}$$

$$\boldsymbol{\frac{\partial Cost}{\partial z}}$$

$$\frac{\partial Cost}{\partial y} = \frac{\partial Cost}{\partial z}\frac{\partial z}{\partial y} = \boldsymbol{\frac{\partial Cost}{\partial z}}$$

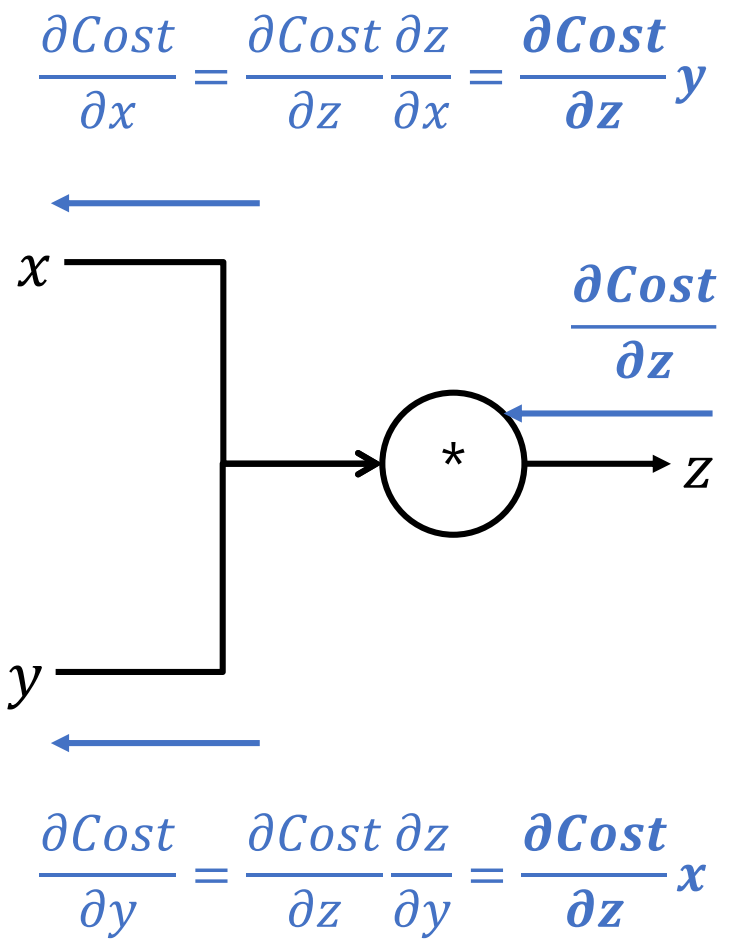
합성함수의 미분(Chain-rule)

- **곱하기** 노드에 대한 편미분



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

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



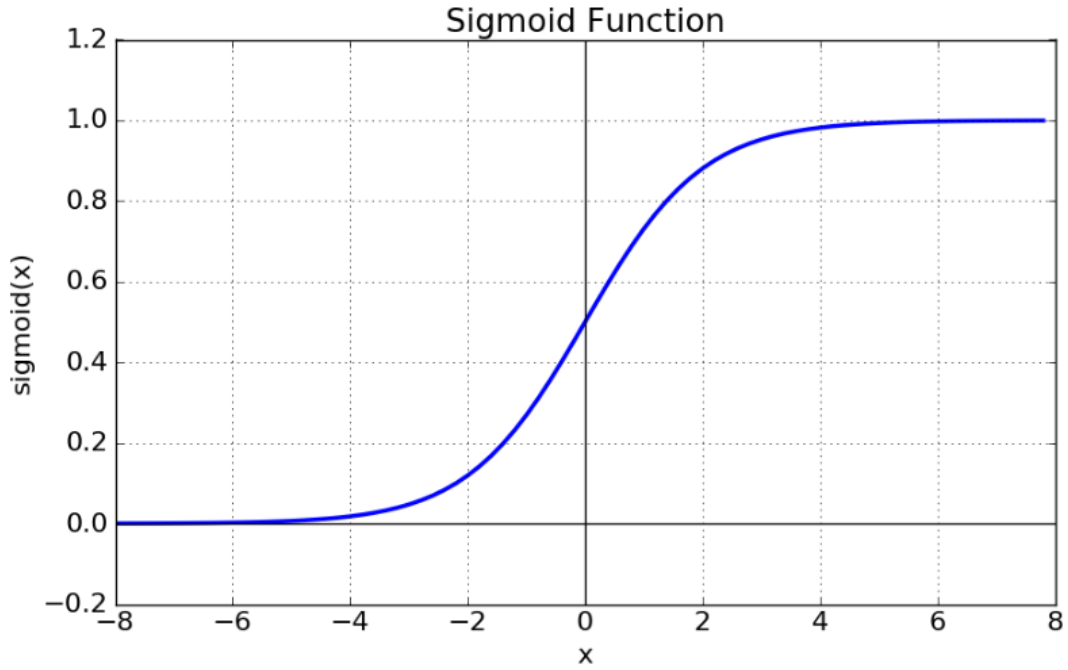
$$\frac{\partial Cost}{\partial x} = \frac{\partial Cost}{\partial z}\frac{\partial z}{\partial x} = \frac{\partial Cost}{\partial z}y$$

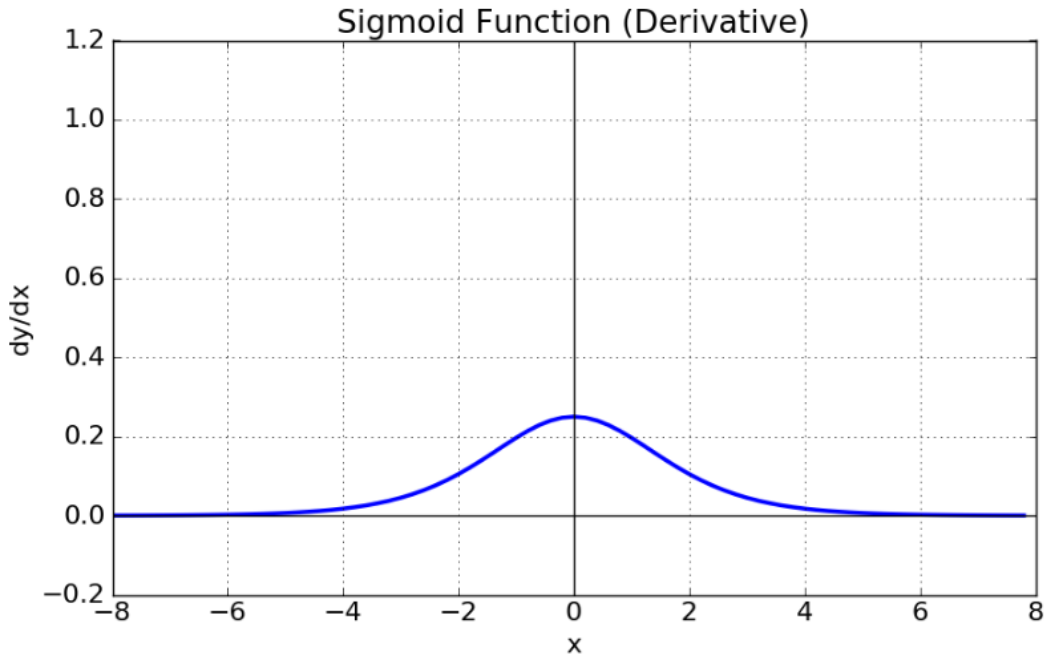
$$\frac{\partial Cost}{\partial z}$$

$$\frac{\partial Cost}{\partial y} = \frac{\partial Cost}{\partial z}\frac{\partial z}{\partial y} = \frac{\partial Cost}{\partial z}x$$

대표적인 활성 함수

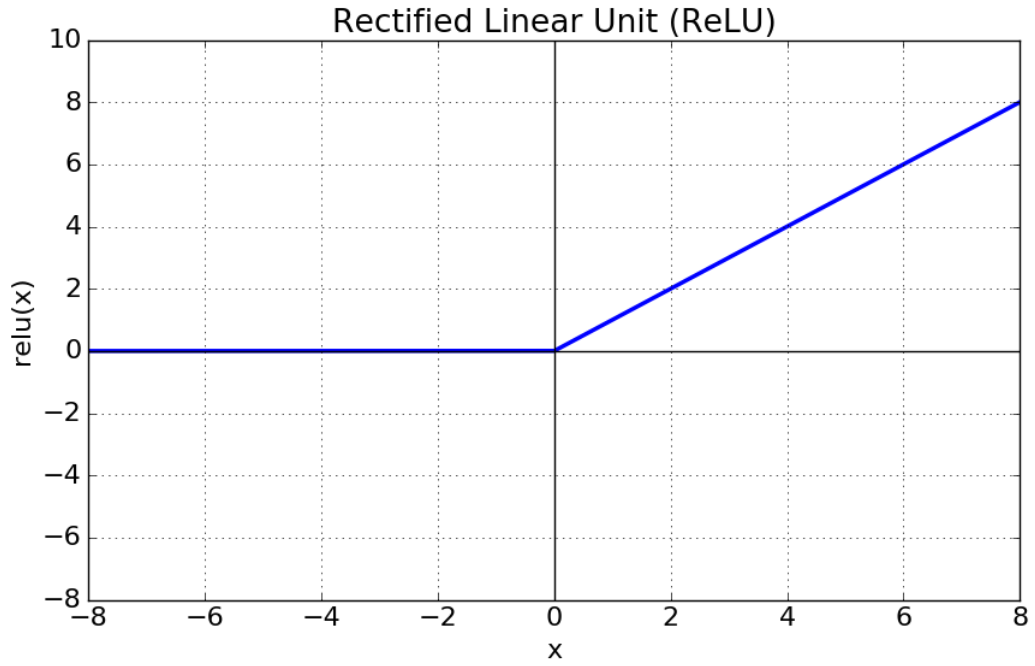
- Sigmoid
  - 뉴럴 네트워크 초기 연구에 많이 사용되었으나 최근에는 많이 사용되지 않습니다.
  - 대표적인 문제점: Gradient vanishing (네트워크가 매우 깊어지면 앞쪽 레이어는 기울기가 작음)





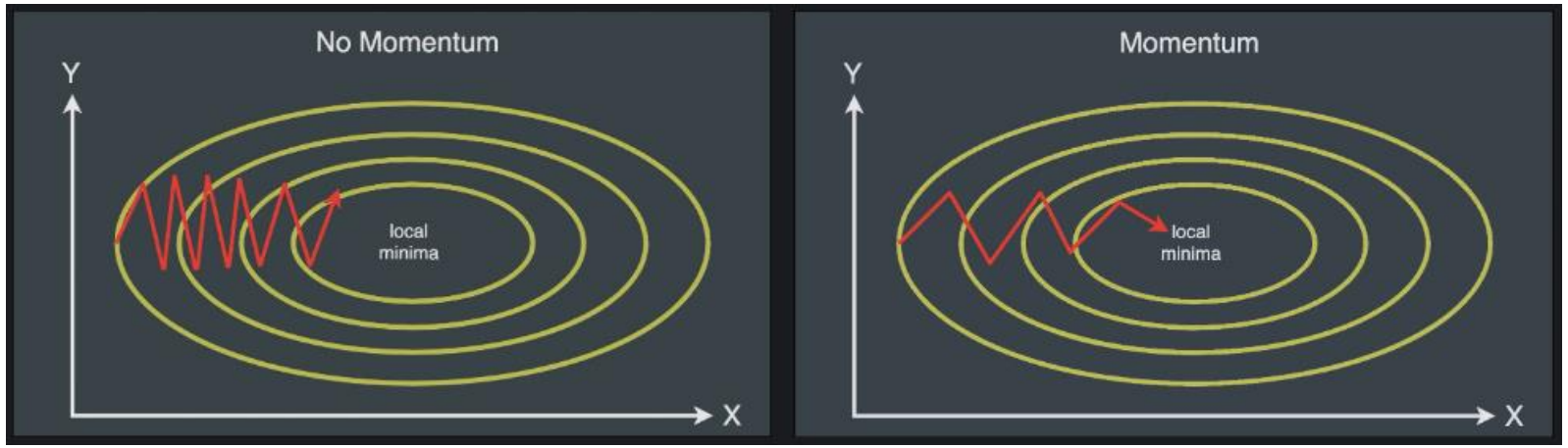
대표적인 활성 함수

- ReLU
  - 입력 값이 0 이상일 때만 입력 값을 그대로 출력하는 함수입니디 $f(x) = max(0, x)$
  - Gradient vanishing 문제를 효과적으로 해결하며 학습 속도를 높입니다.



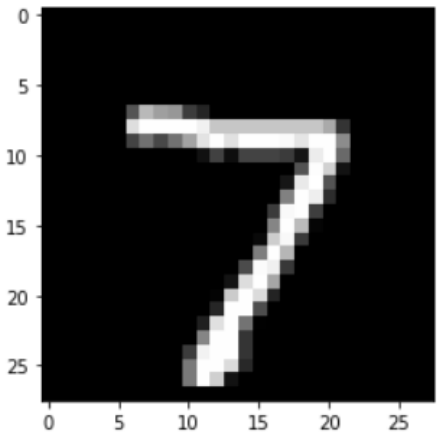
최적화(Optimizer)

- GD: 전체 데이터를 한 번에 확인한 뒤에 기울기를 계산해 학습
- SGD: 데이터셋을 쪼갠 뒤에 미니 배치(Mini-batch) 단위로 학습
  - Momentum: 내려오던 방향으로(관성) 조금 더 많이 학습
  - Adagrad: 안 가본 곳을 초반에 빠르게 학습하고, 많이 가본 곳은 세밀하게 학습
    - RMSProp: 이전 맥락을 확인하며 세밀하게 학습
  - Adam: RMSProp + Momentum의 개념을 적절히 활용



DNN for MNIST

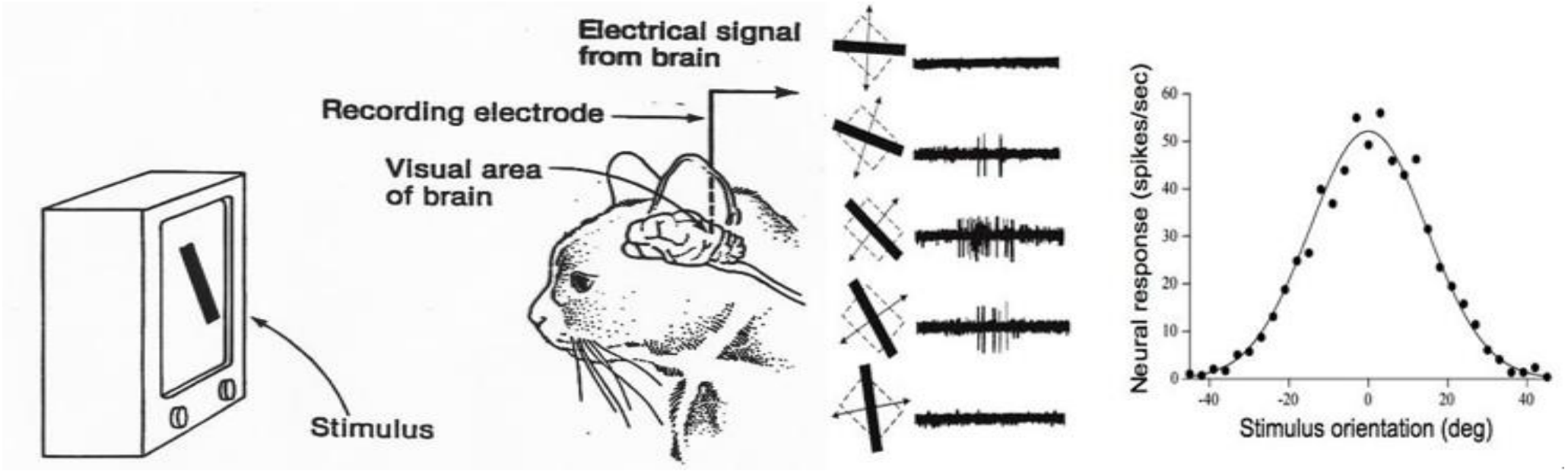
- PyTorch를 이용해 손글씨 분류기 만들기
- 실습 코드 살펴보기
  - 기본적인 DNN을 이용해 MNIST 분류하기 실습



숫자 7 (95%)

CNN

- **고양이 시각피질 반응 연구**: 고양이의 시각피질에는 사선을 인식하는 기능이 있습니다.
  - 다양한 패턴에 따라 뉴런이 반응하는 정도가 서로 다르다는 실험 결과가 도출됩니다.

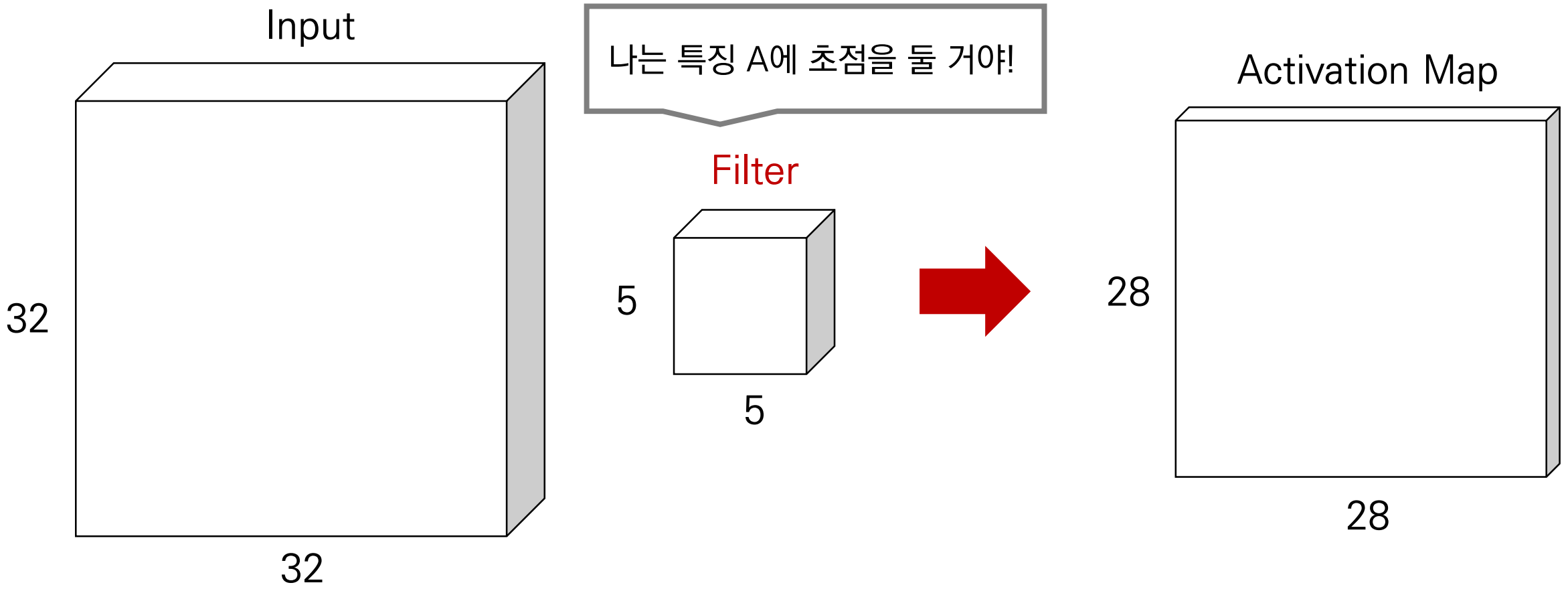


CNN

- 기본적으로 모든 뉴런을 **완전 연결(Fully-Connected)** 방식으로 연결하면 어떨까요?
  - 파라미터의 수가 많으며 학습 시간이 길어집니다.
- 어떻게 하면 이미지의 공간 정보를 적절히 유지하며 학습할 수 있을까요?
  - CNN을 이용하면 많은 파라미터를 공유하고 이미지 특성을 효과적으로 학습할 수 있습니다.

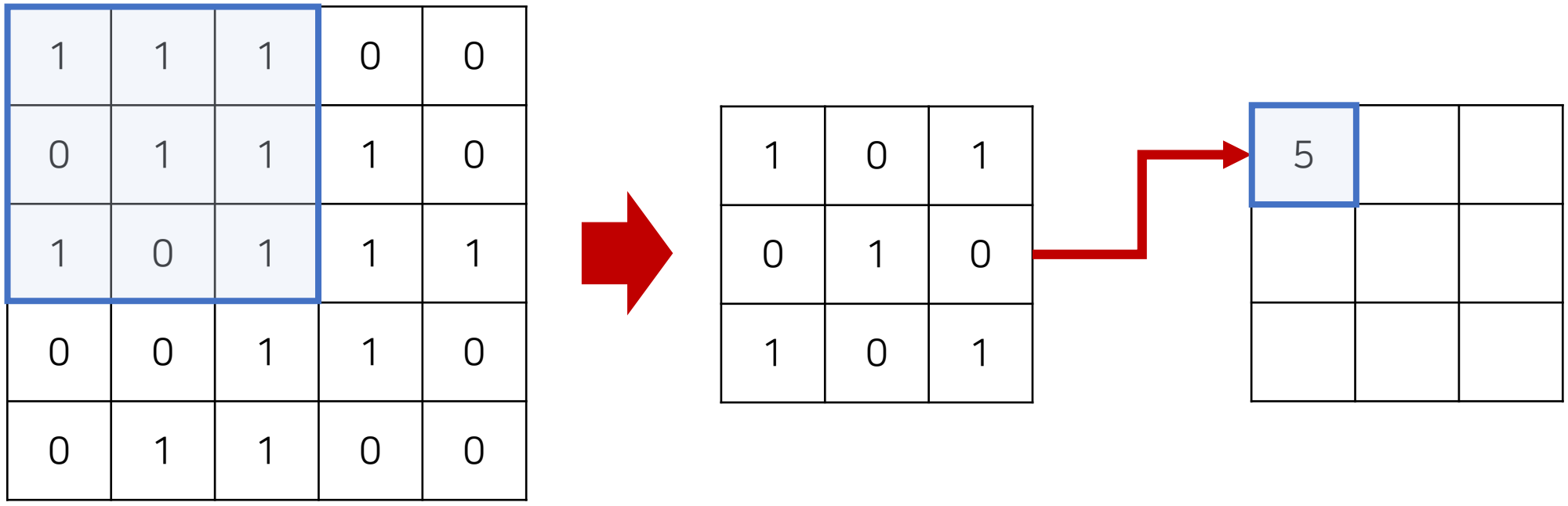
CNN

- CNN을 위한 준비물



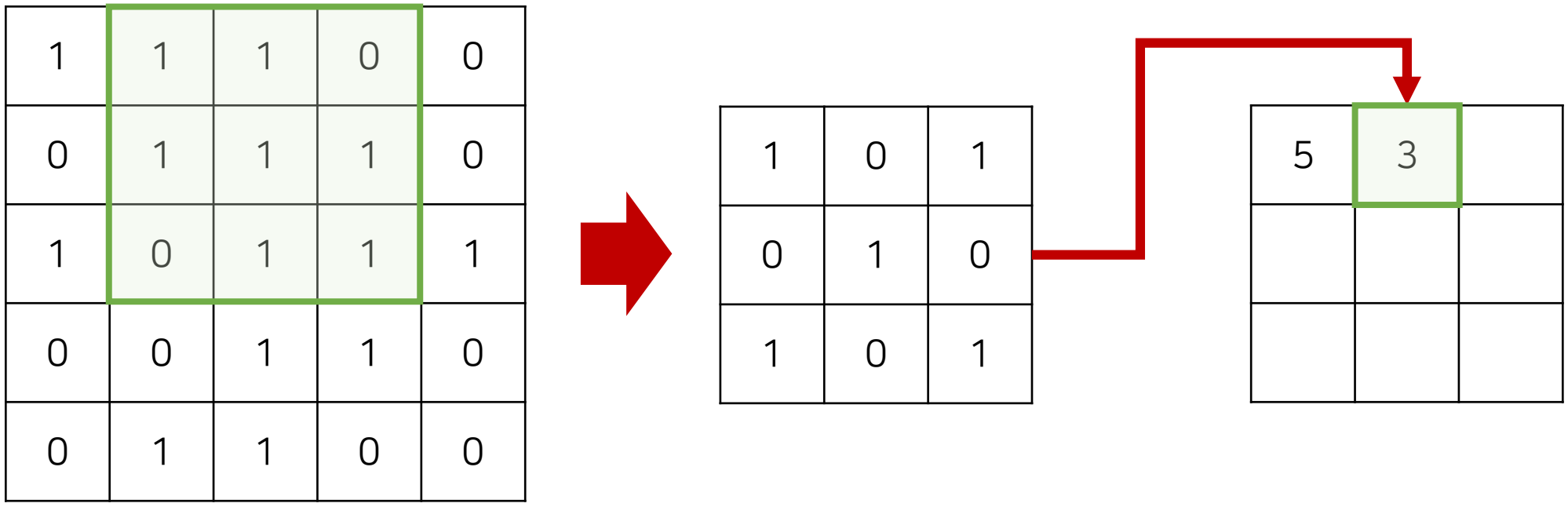
CNN

- 하나의 필터는 이미지에 대하여 슬라이딩 하면서 특징 맵(feature map)을 계산합니다.



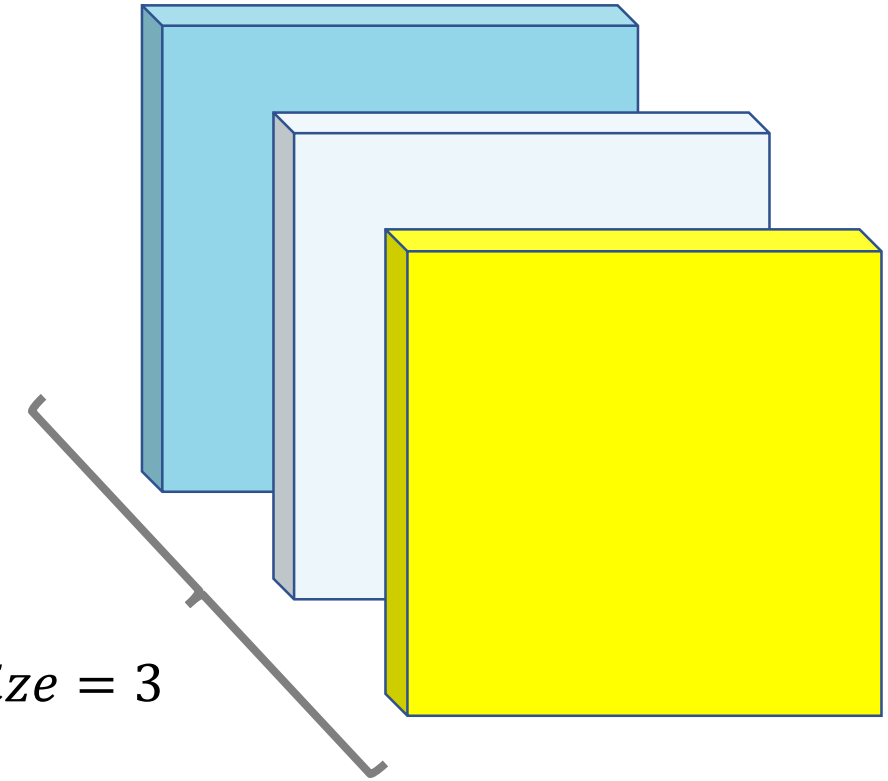
CNN

- 하나의 필터는 이미지에 대하여 슬라이딩 하면서 특징 맵(feature map)을 계산합니다.

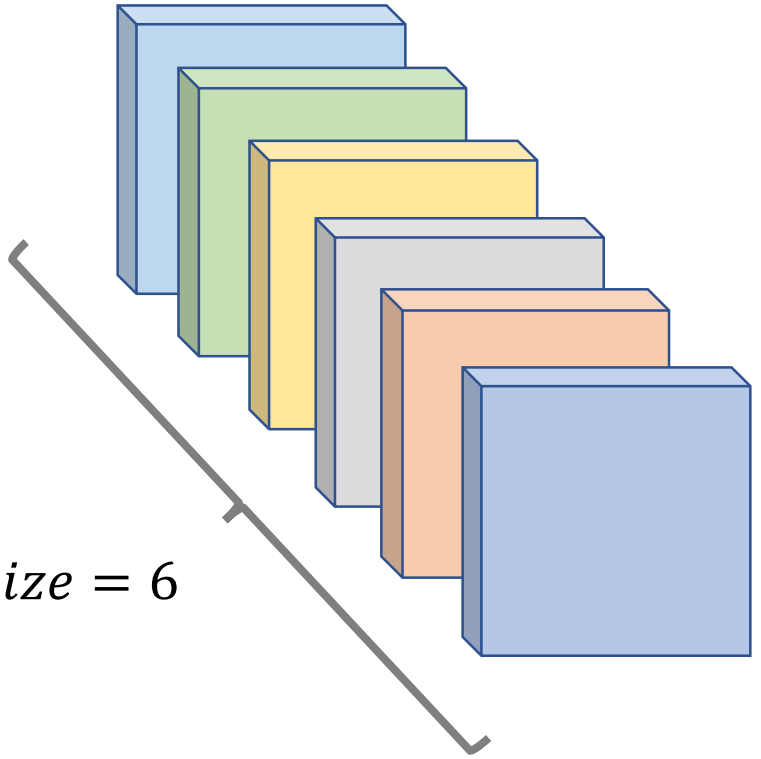


CNN 모델의 특징 맵 (Feature Map)

- 일반적으로 CNN에서 레이어가 깊어질수록 채널의 수가 많아지고 너비와 높이는 줄어듭니다.
- 컨볼루션 레이어의 **서로 다른 필터들은** 각각 적절한 **특징(feature)**값을 추출하도록 학습됩니다.



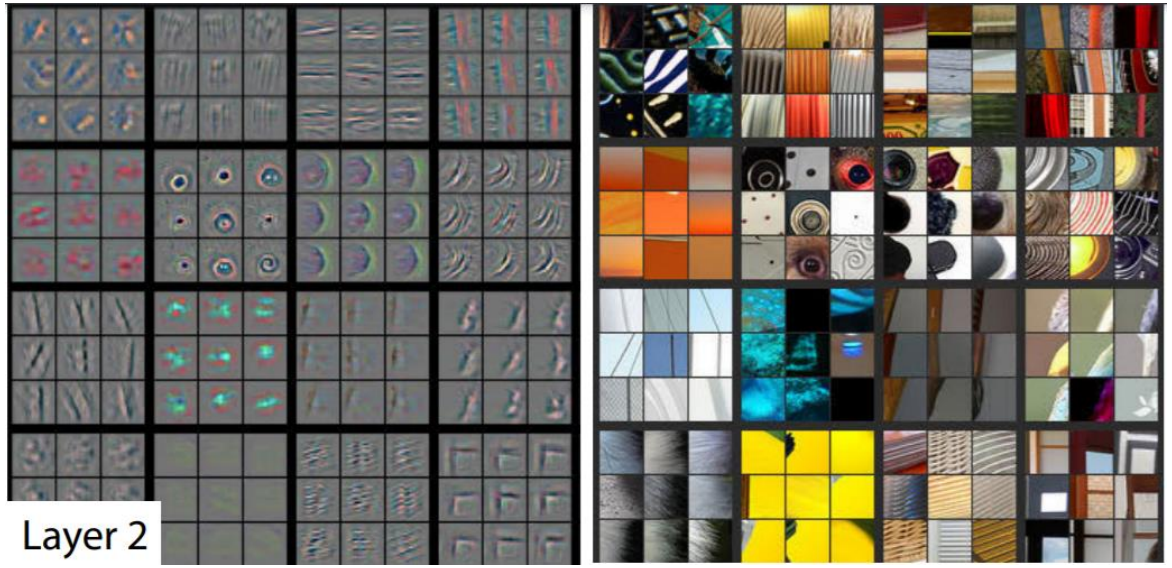
$Channel\ size = 3$

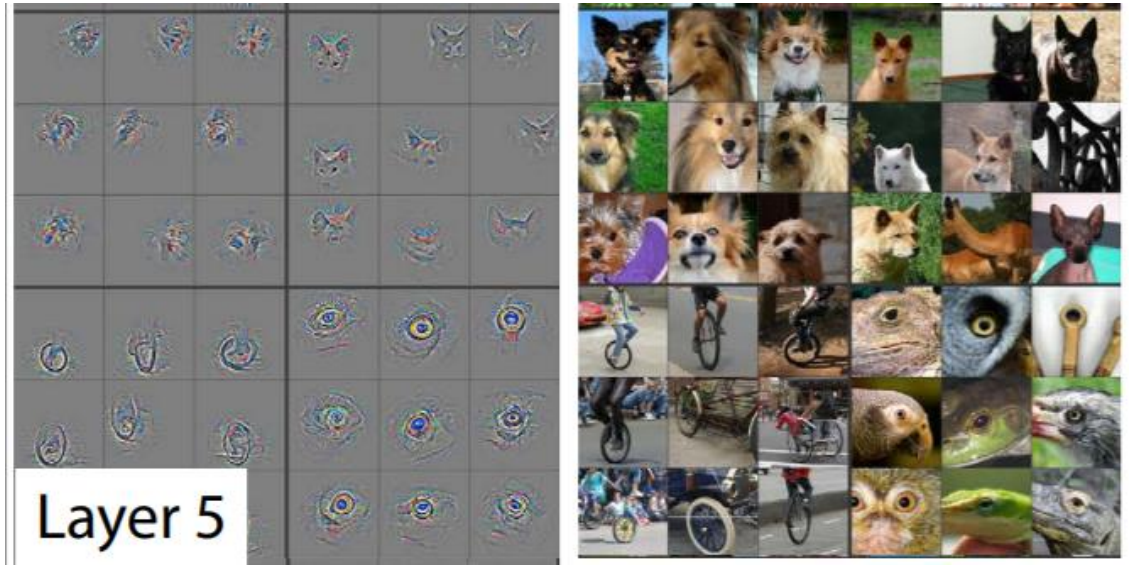


$Channel\ size = 6$

CNN의 필터(Filter)

- 실제로 각 필터는 특정한 특징(feature)를 인식하기 위한 목적으로 사용됩니다.
- 각 필터는 특징이 반영된 특징 맵(feature map)을 생성합니다.
- 얕은 층에서는 local feature, 깊은 층에서는 고차원적인 global feature를 인식하는 경향이 있습니다.

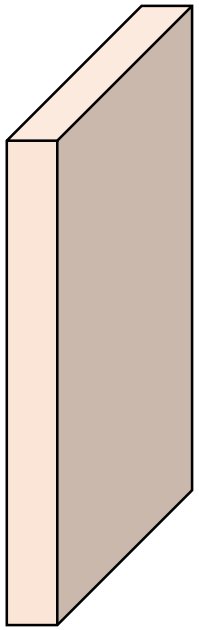




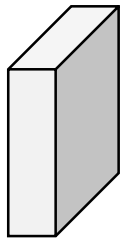
*Visualizing and Understanding Convolutional Networks

CNN: 예시 살펴보기

- 하나의 입력 이미지(왼쪽)와 하나의 필터(오른쪽)가 있다고 가정합시다.



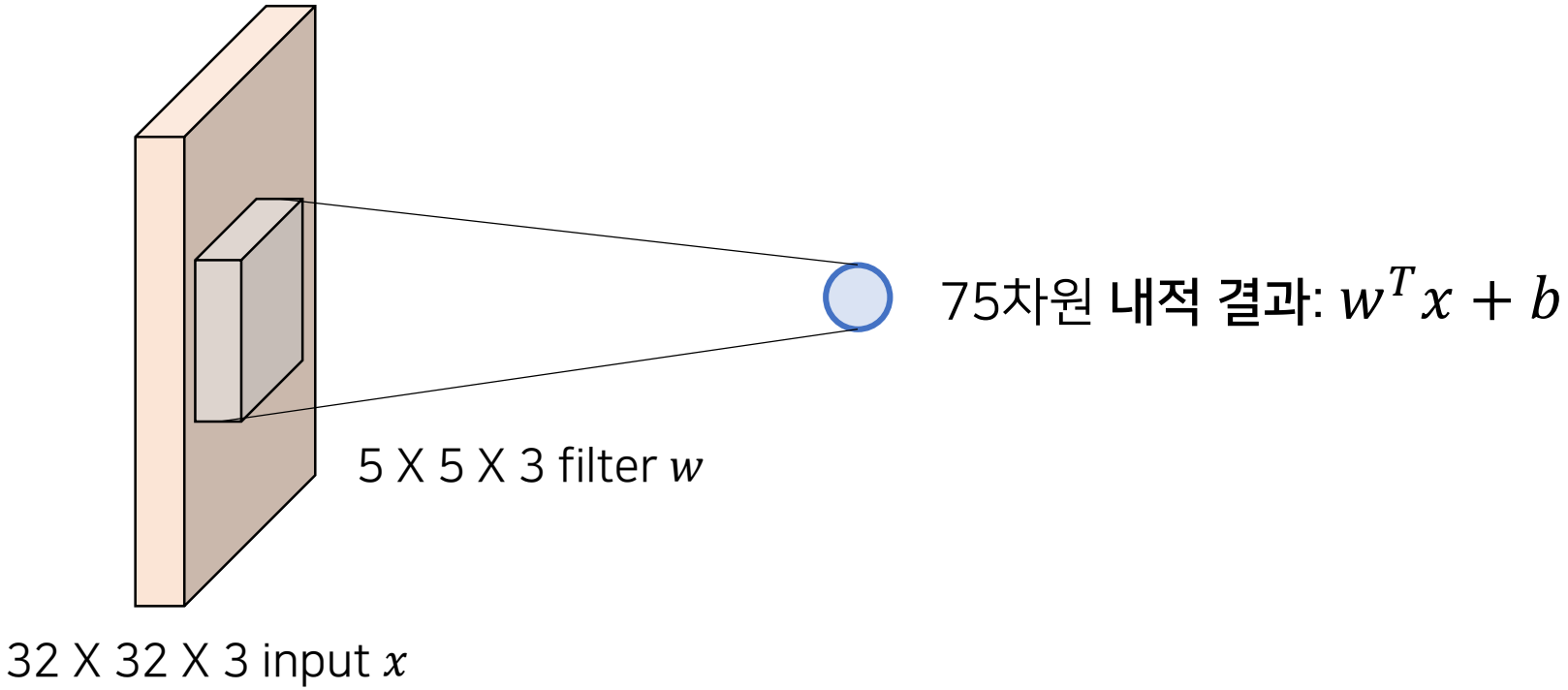
32 X 32 X 3 input



5 X 5 X 3 filter

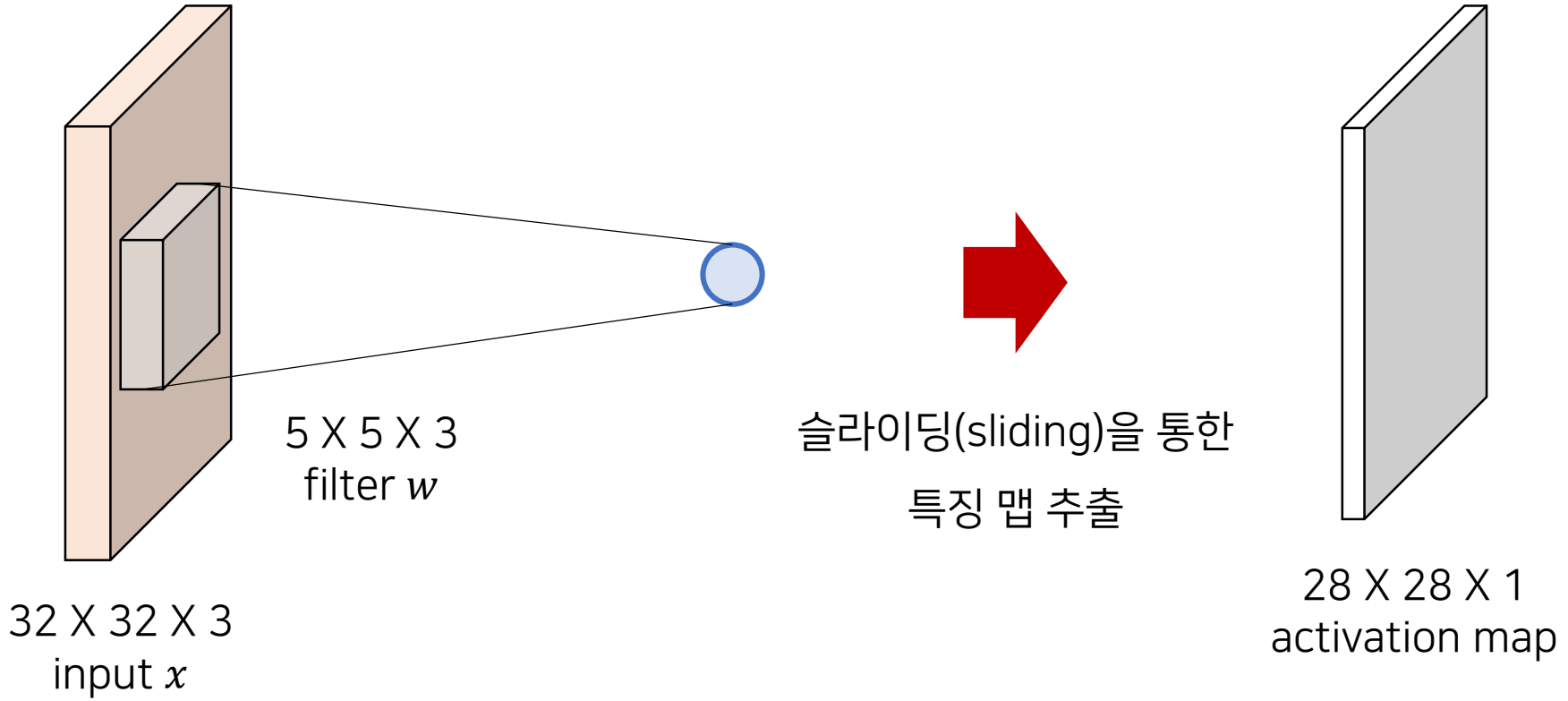
CNN: 예시 살펴보기

- 입력 이미지의 로컬 영역과 필터(filter) 사이에서 내적(dot product)을 계산해 각 위치의 결과를 구합니다.



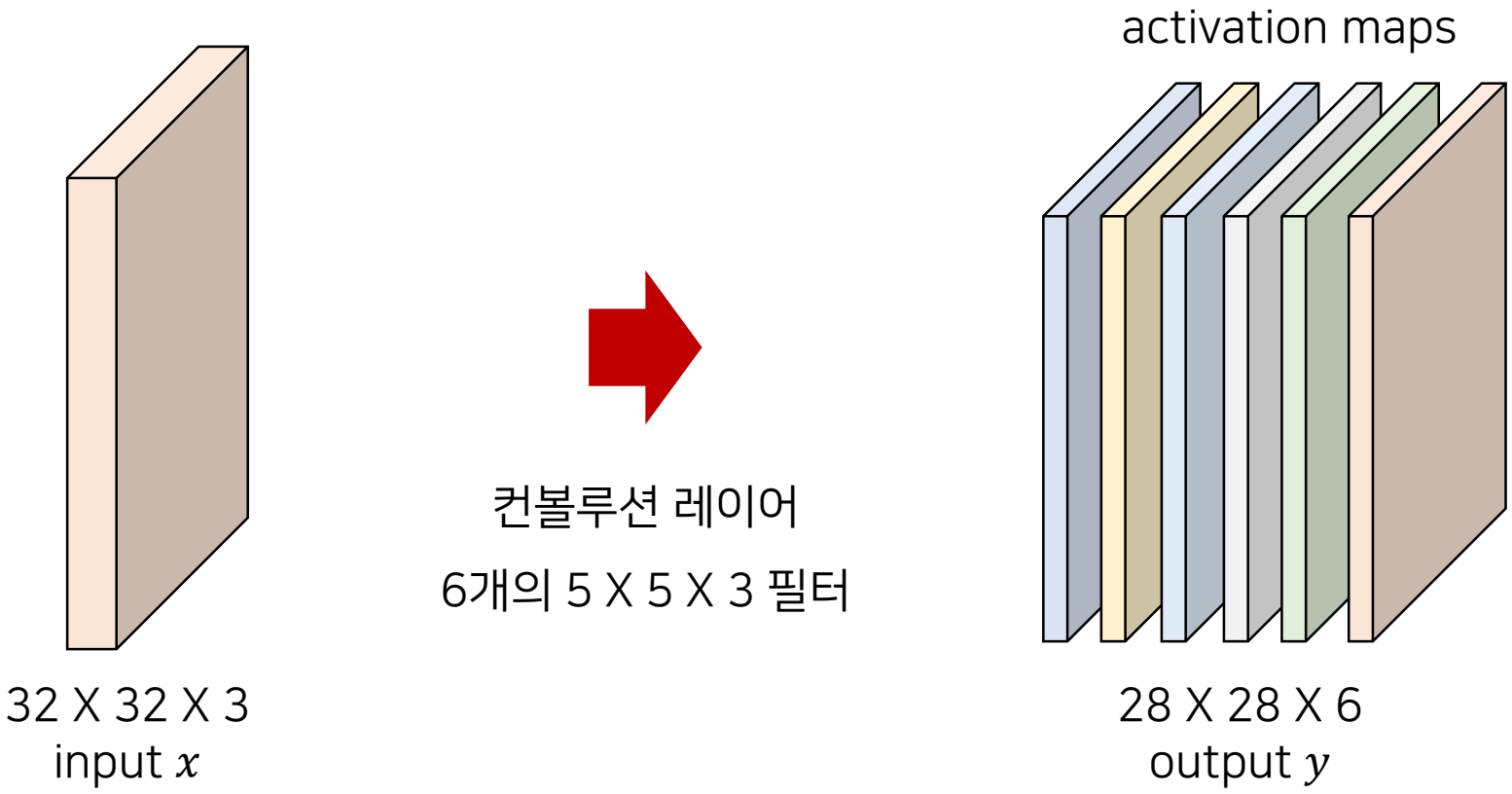
CNN: 예시 살펴보기

- 각 위치에서의 콘볼루션 연산 결과를 모아서 특징 맵(feature map)을 생성합니다.



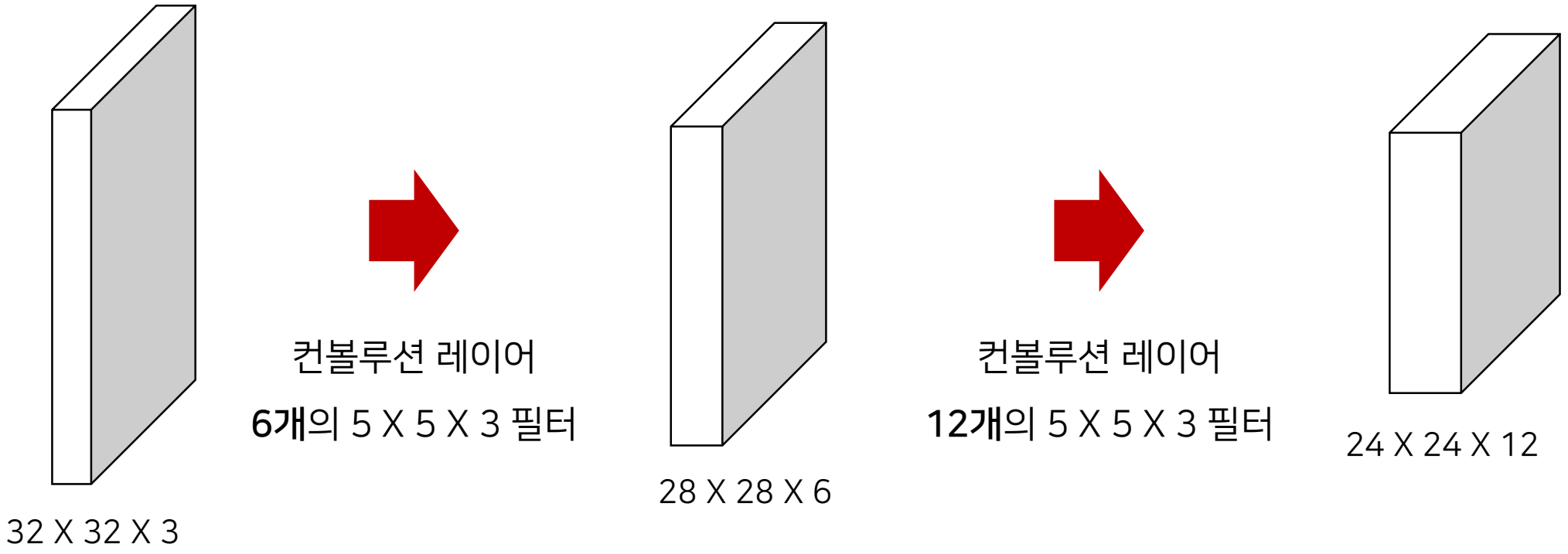
CNN: 예시 살펴보기

- 6개의 개별적인 필터(Filter)를 가진 Convolution Layer를 이용하면 다음과 같습니다.



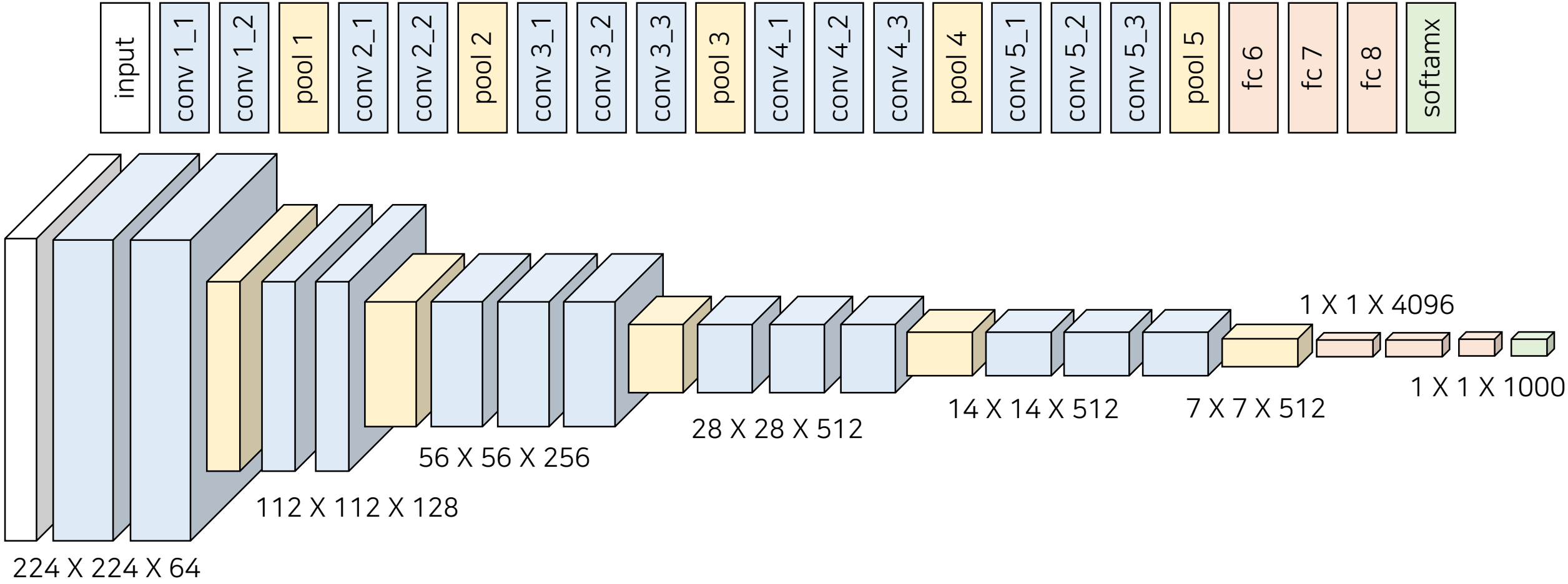
CNN: 예시 살펴보기

- 실제 CNN Layer는 **여러 번 중첩되어 사용**될 수 있습니다.



VGG Network (ICLR 2015)

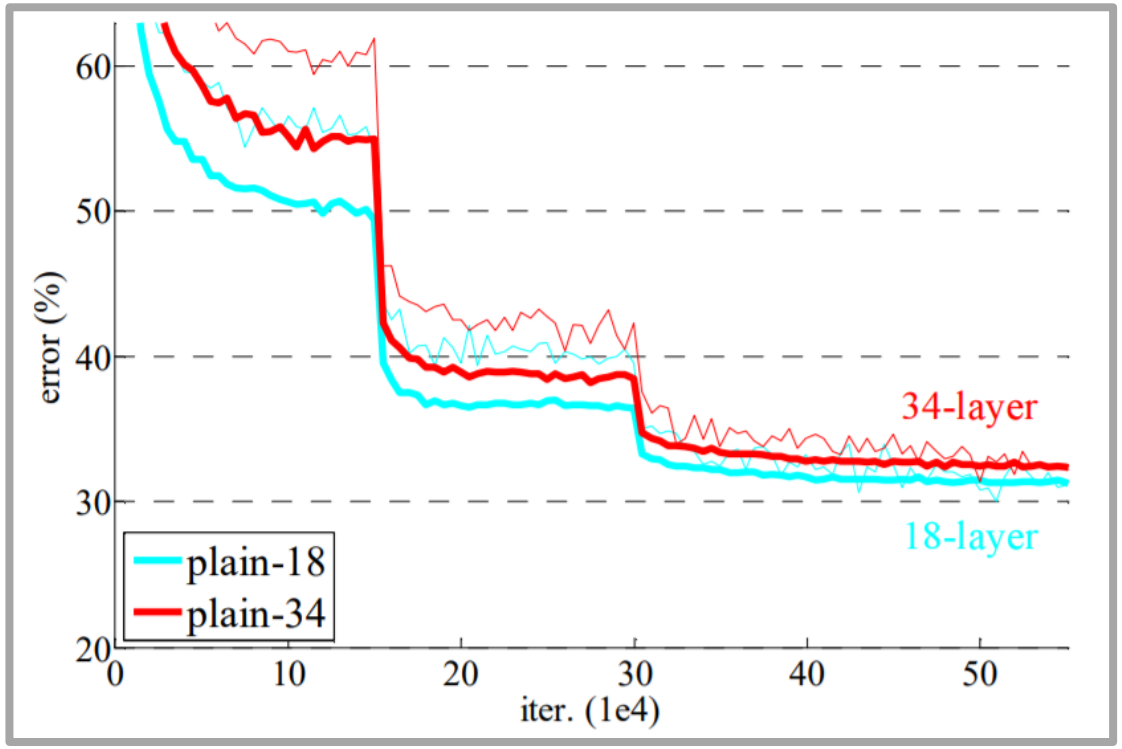
- **VGG 네트워크**는 작은 크기의 **3x3 컨볼루션 필터(filter)**를 이용해 **레이어의 깊이**를 늘려 우수한 성능을 보입니다.



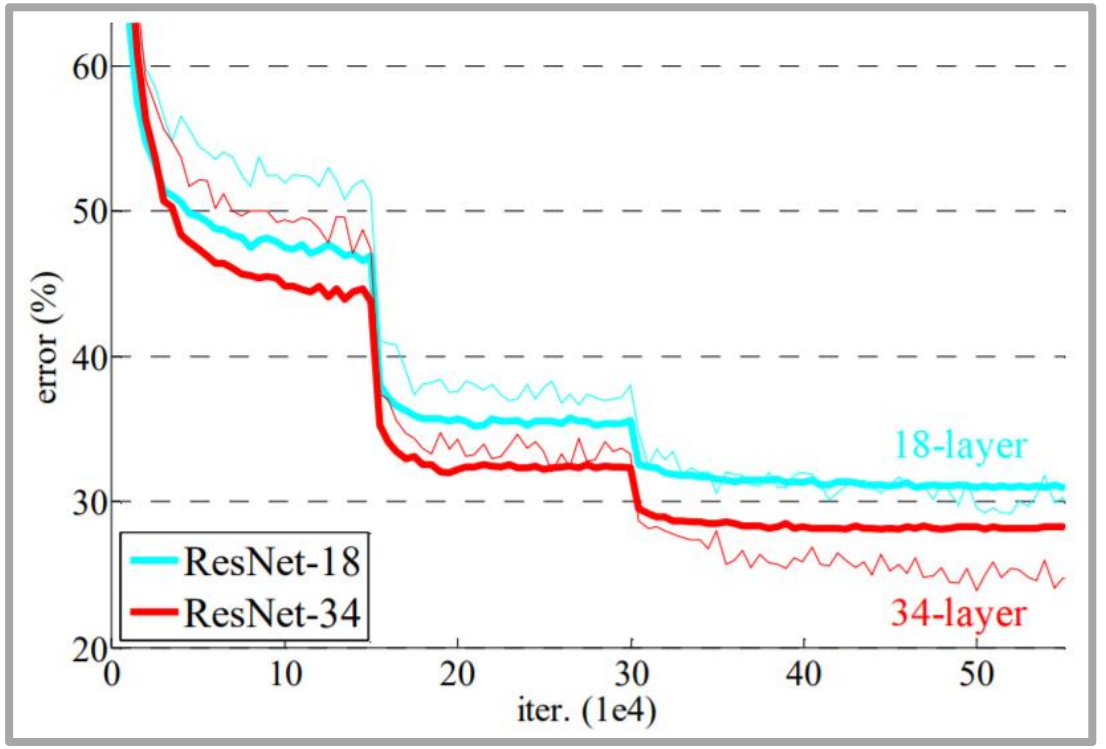
ResNet (CVPR 2016)

- 본 논문에서는 **깊은 네트워크**를 학습시키기 위한 방법으로 **잔여 학습(residual learning)**을 제안합니다.

< ImageNet top-1 training error >



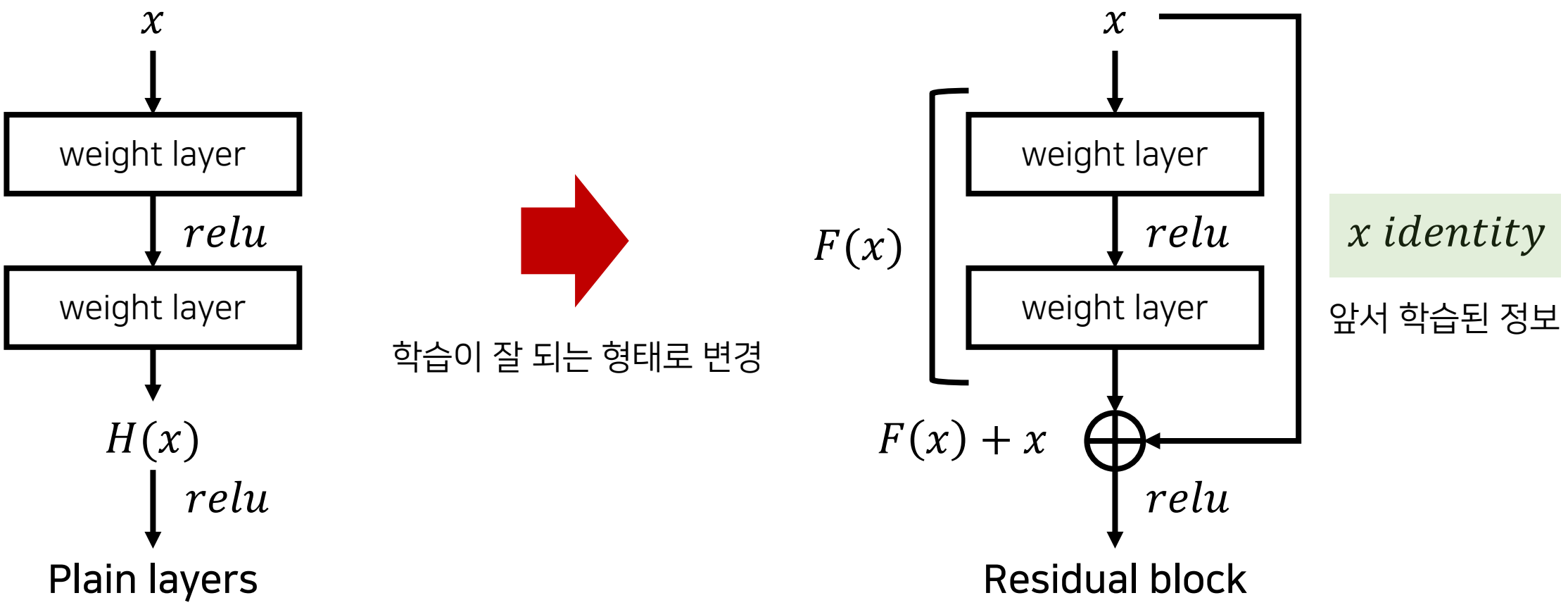
일반적인 CNN



잔여 학습을 적용한 CNN

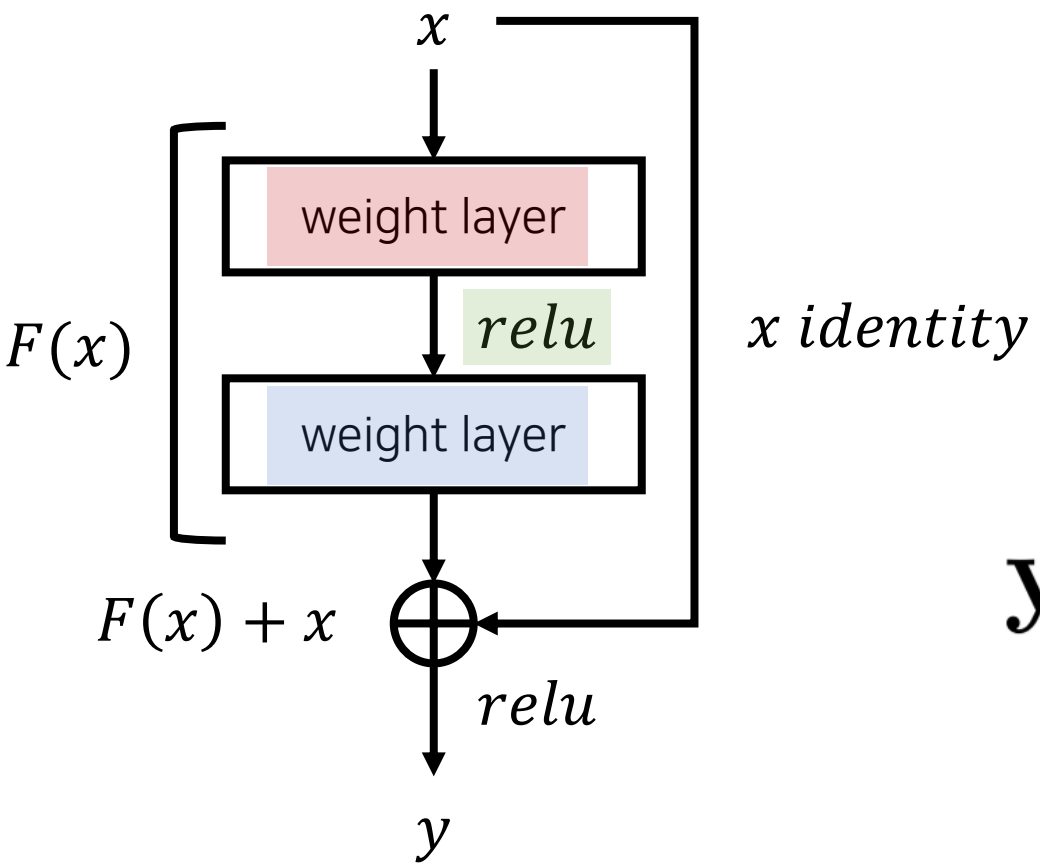
ResNet (CVPR 2016)

- **잔여 블록(residual block)**을 이용해 네트워크의 **최적화(optimization) 난이도**를 낮춥니다.
  - 실제로 내재한 mapping인 $H(x)$를 곧바로 학습하는 것은 어려우므로 대신 $F(x) = H(x) - x$를 학습합니다.



ResNet (CVPR 2016)

- **잔여 블록(residual block)**을 이용해 네트워크의 **최적화(optimization) 난이도**를 낮춥니다.



$$\mathcal{F} = W_2\sigma(W_1\mathbf{x})$$

일반적인 형태

$$\mathbf{y} = \underbrace{\mathcal{F}(\mathbf{x}, \{W_i\})}_{\text{multiple convolutional layers}} + \underbrace{W_s\mathbf{x}}_{\text{shortcut}}$$

ResNet (CVPR 2016)

- 이전까지의 아키텍처와 다르게 **레이어가 깊어질수록 성능이 향상**됩니다. (단, 레이어가 과도하게 깊으면 오히려 감소)

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

Top-1 validation error rates (%)

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

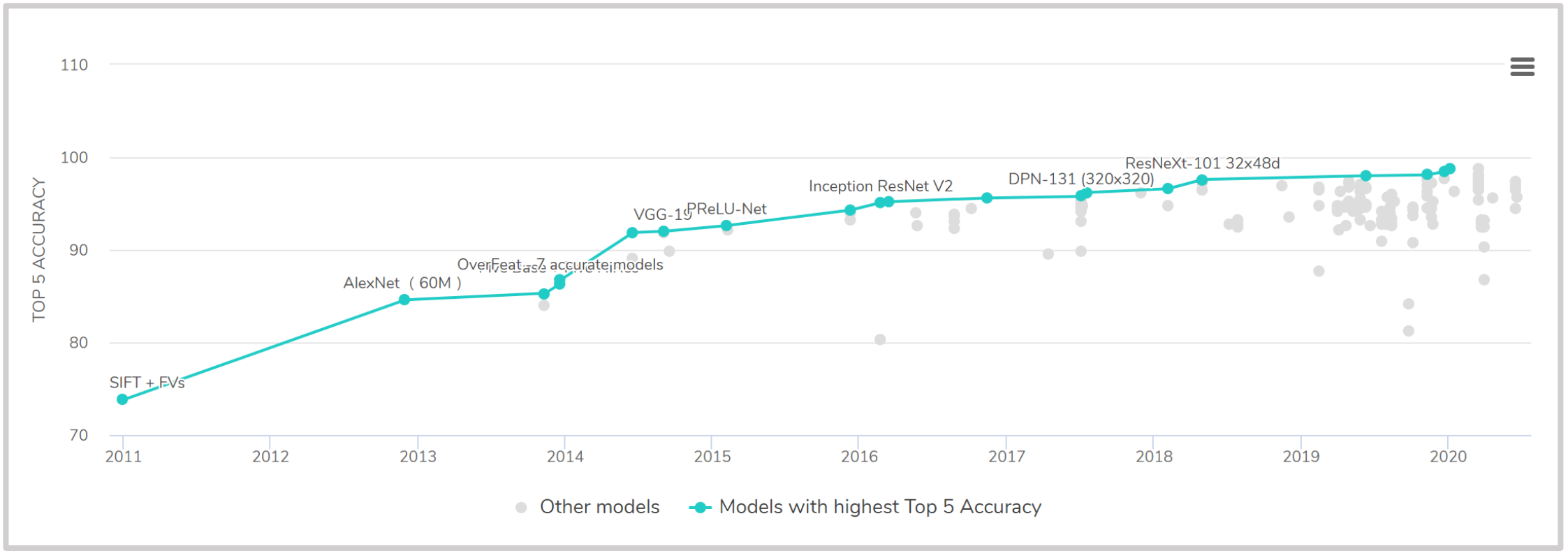
Top-5 test error rates (%) of **ensembles**

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Validation error rates (%) of **single-model**

CNN을 활용한 다양한 아키텍처

- State-of-the-art 동향: https://paperswithcode.com/sota/image-classification-on-imagenet



Transfer Learning을 활용한 마동석 분류기 만들기

- PyTorch를 이용해 마동석/김종국 분류기 만들기
- 실습 코드 살펴보기
  - 이미지 크롤링 + Transfer Learning + Web API 실습

마동석 88%

김종국 94%

김종국 95%