머신러닝을 위한 수학

04 Gradient Descent

목차

🗂️ 커리큘럼

1. 최적화 문제란?

최적화 문제가 무엇이고 왜 필요한지를 학습합니다.

2. Gradient Descent란?

Gradient Descent를 이용한 최적화 문제의 해결법을 학습합니다.

📁 커리큘럼

3. Gradient Descent 수행하기

몇가지 최적화 문제를 정해진 단계에 따라 Gradient Descent를 수행해 해결해합니다.

4. 예시문제

대표 예시 문제를 Gradient Descent로 풀어봅니다.

추천대상

1. 머신러닝 입문자

머신러닝을 얼핏 알지만, 이해는 못하는 사람

2. 데이터 분석 입문자

파이썬 라이브러리를 실용적으로 활용해보고 싶은 사람

3. 벡터 행렬을 모르는 사람

머신러닝의 이해에 필수적인 벡터와 행렬에 대해 모르는 사람

수강목표

1. 머신러닝의 전반에 대해 이해합니다.

인공지능과 머신러닝의 차이를 알고, 일반적인 머신러닝의 구조를 이해합니다.

2. 머신러닝 속의 본질적 수학 지식을 이해합니다.

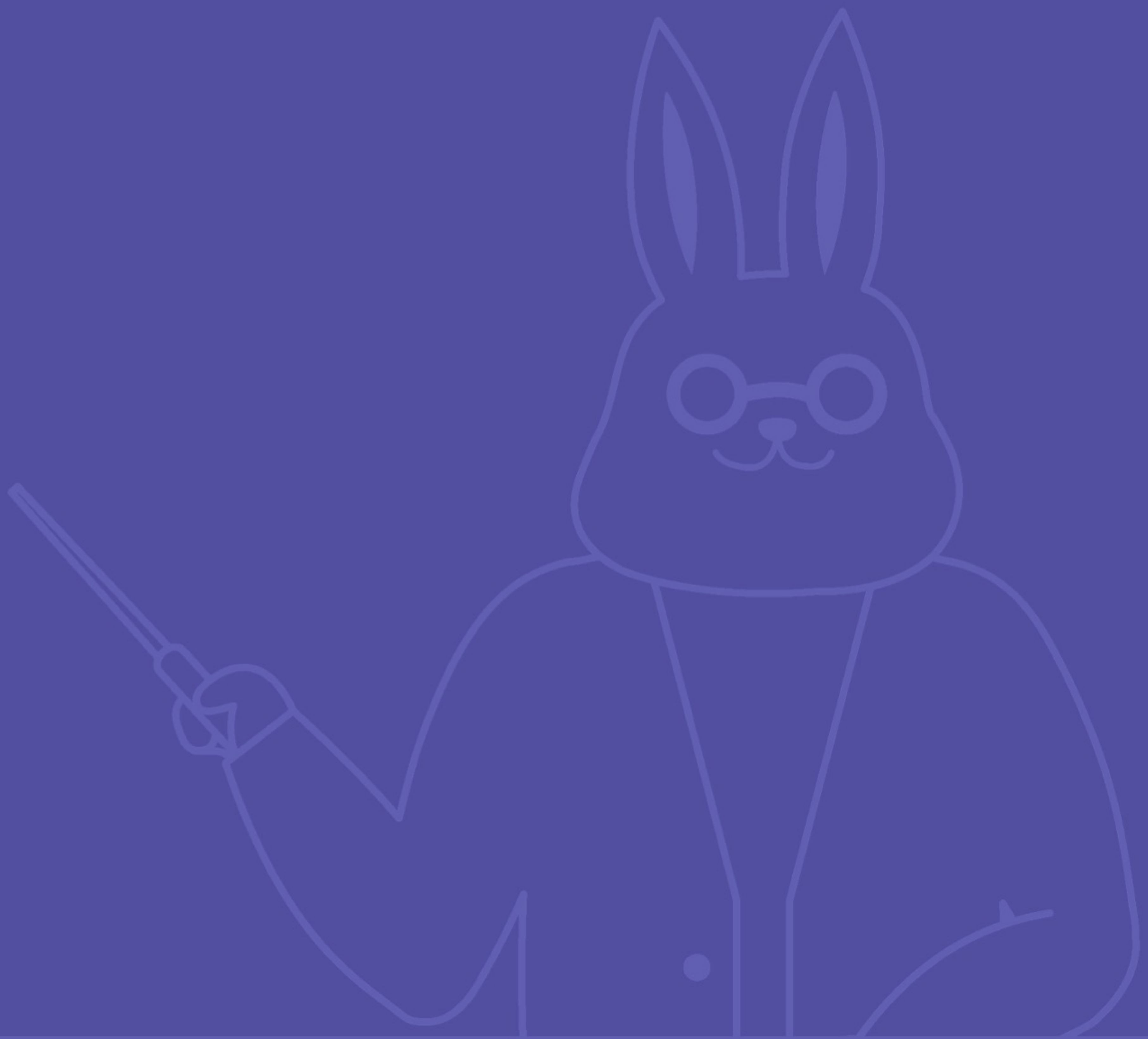
막연하고 이해하기 어려웠던 지식을 체계적으로 배우며 익힙니다.

3. 어떤 머신러닝 기법을 마주하더라도 두렵지 않습니다.

익힌 수학 지식으로 머신러닝 속의 여러 기법들을 접해도 어렵지 않습니다.

01

최적화 문제란?



최적화 문제

- 정의: 어떤 목적 함수(objective function)의 출력값을 최대 혹은 최소화하는 파라미터를 찾는 문제.

- 예시)

판매자의 입장에서 물건을 높은 가격에 많이 판매하여 높을 매출(이익)을 얻고자 할 것이고, 소비자 입장에서는 낮은 가격에 좋은 퀄리티의 물건을 사고자 한다.

- 예시)

판매자의 입장에서 물건 1,000개를 높은 가격에 판매하여 높을 매출(이익)을 얻고자 할 것이고, 소비자 입장에서는 낮은 가격에 좋은 퀄리티의 물건을 사고자 한다.

분석 – 판매자

- 물건 1개당 원가: c
- 물건 1개당 판매액: y
- 목적함수 = 전체 이익 = $\mathbf{1000} \times (\boldsymbol{y} - \boldsymbol{c})$ ←최대화!

- 우리가 하는 것은? (What to do?)
- 현재 환경, 원가 등을 입력(x)으로 고려해 목적함수를 최대로 하는 $y = f(x)$를 찾는 문제로 바꿔서 풀 것임.

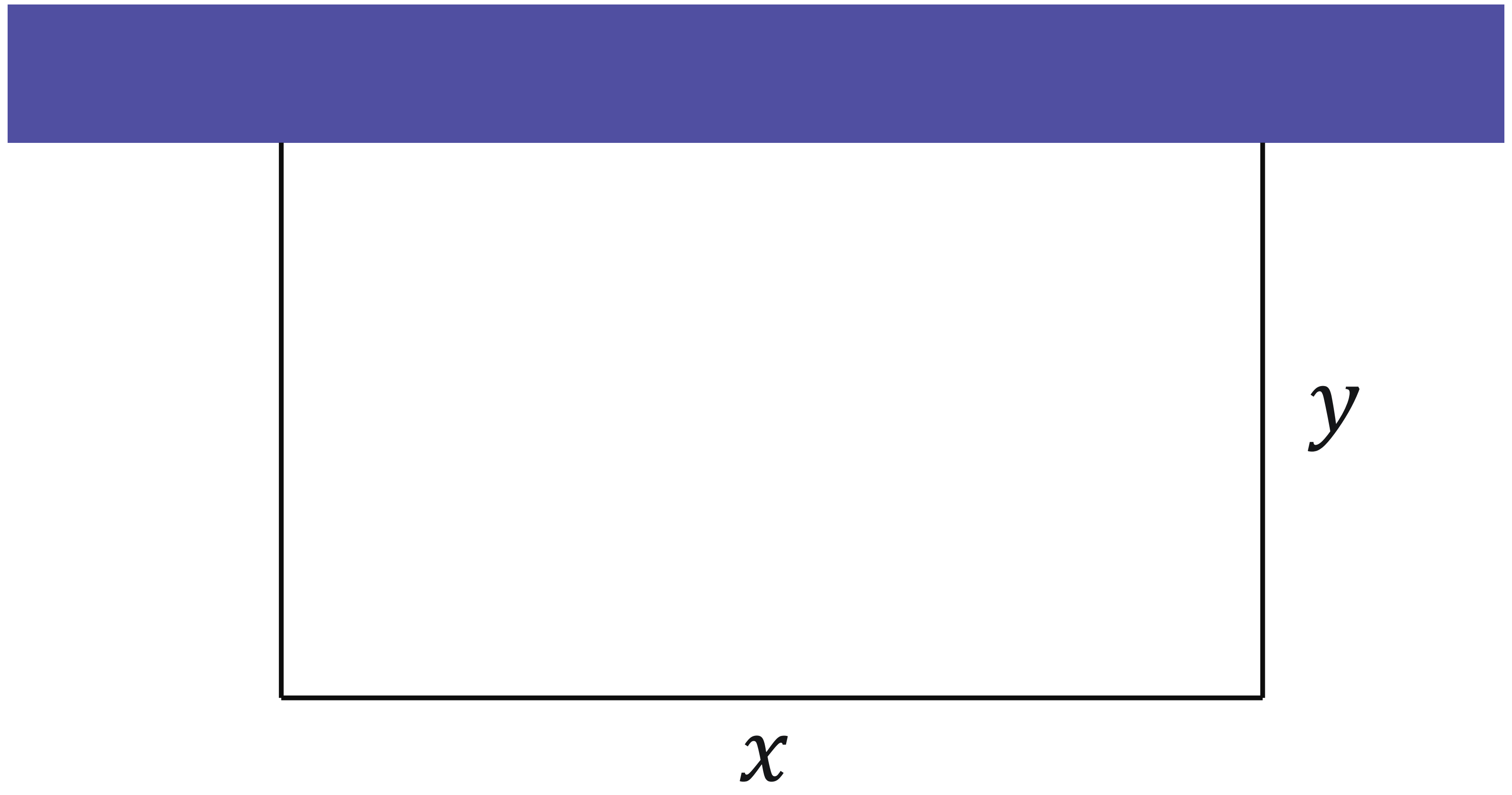
- 목적함수 = 전체 이익 = $1000 \times (y - c)$

- 우리가 하는 것은? (What to do?)
- 이런 문제 속에서 X에 대한 조건이 주어짐.
- 가령, “원가의 가격은 1,000원에서 10,000원 사이이다.”, “소비자는 10대이다.” 등의 조건임.

- 목적함수 = 전체 이익 = $1000 \times (y - c)$

- 대표 최적화 수학 문제

아래 그림처럼 직사각형 모양의 울타리를 벽에 치고자 한다. 울타리의 길이가 15m일 때, 창고의 넓이를 최대로 하려면 창고의 가로, 세로의 길이를 어떻게 해야하는가?



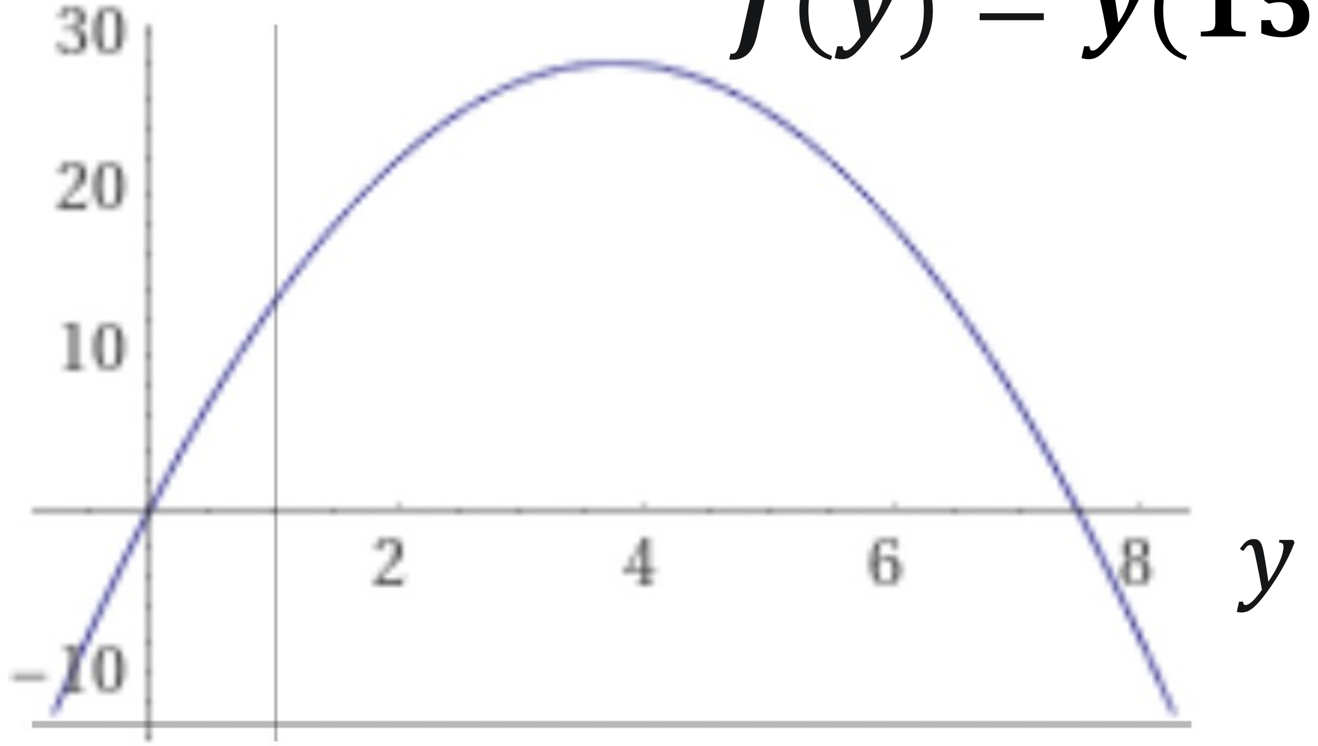
- 풀이) 아래처럼 문제를 바꿀 수 있음.

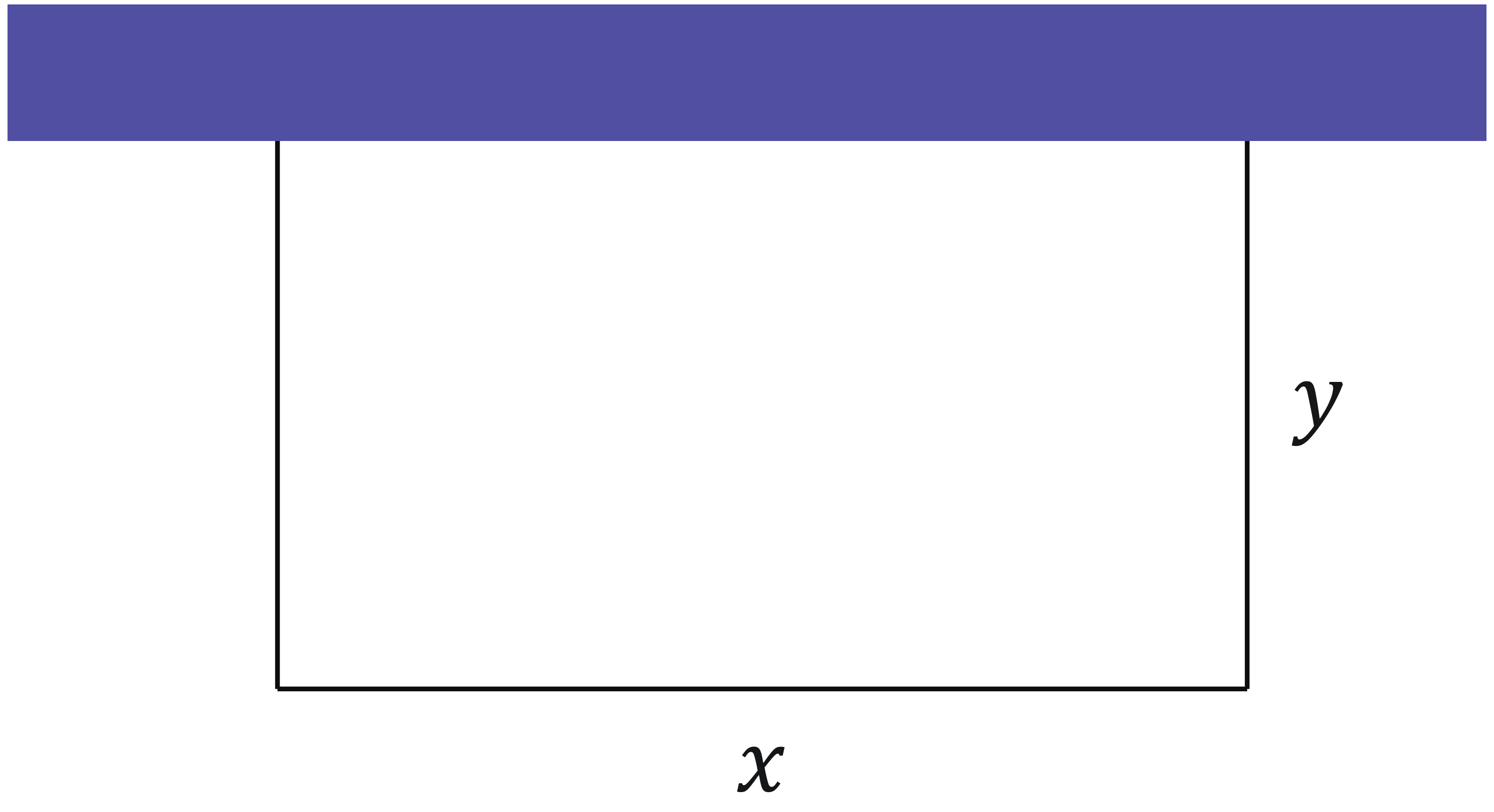
조건: $x + 2y = 15, x \geq 0, y \geq 0$

목적 함수: $xy \leftarrow$ 최대화

문제 변형: $max_y\ y(15 - 2y)$

$f(y) = y(15 - 2y)$

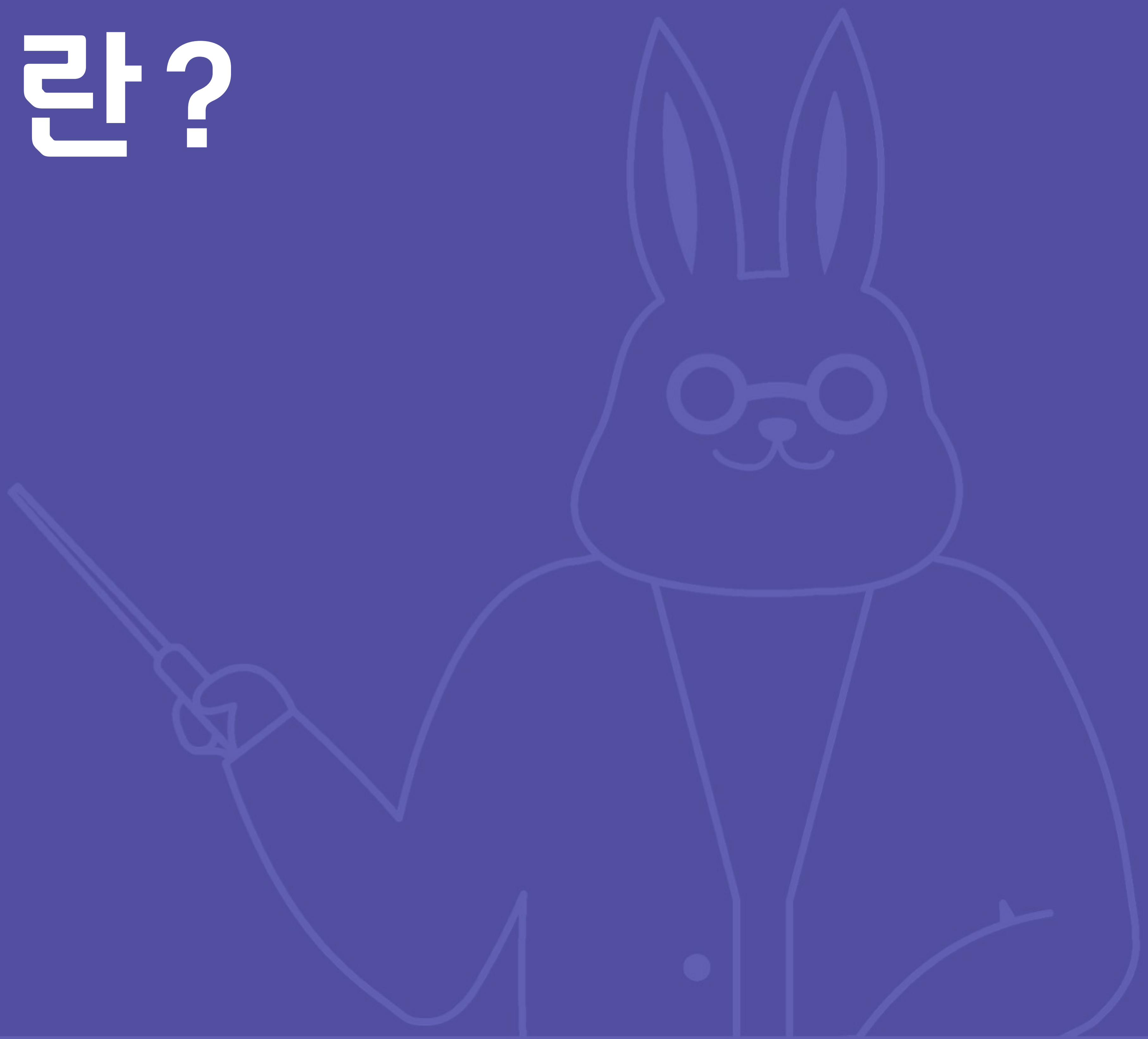




답: $y = \frac{15}{4}, x = \frac{15}{2}$

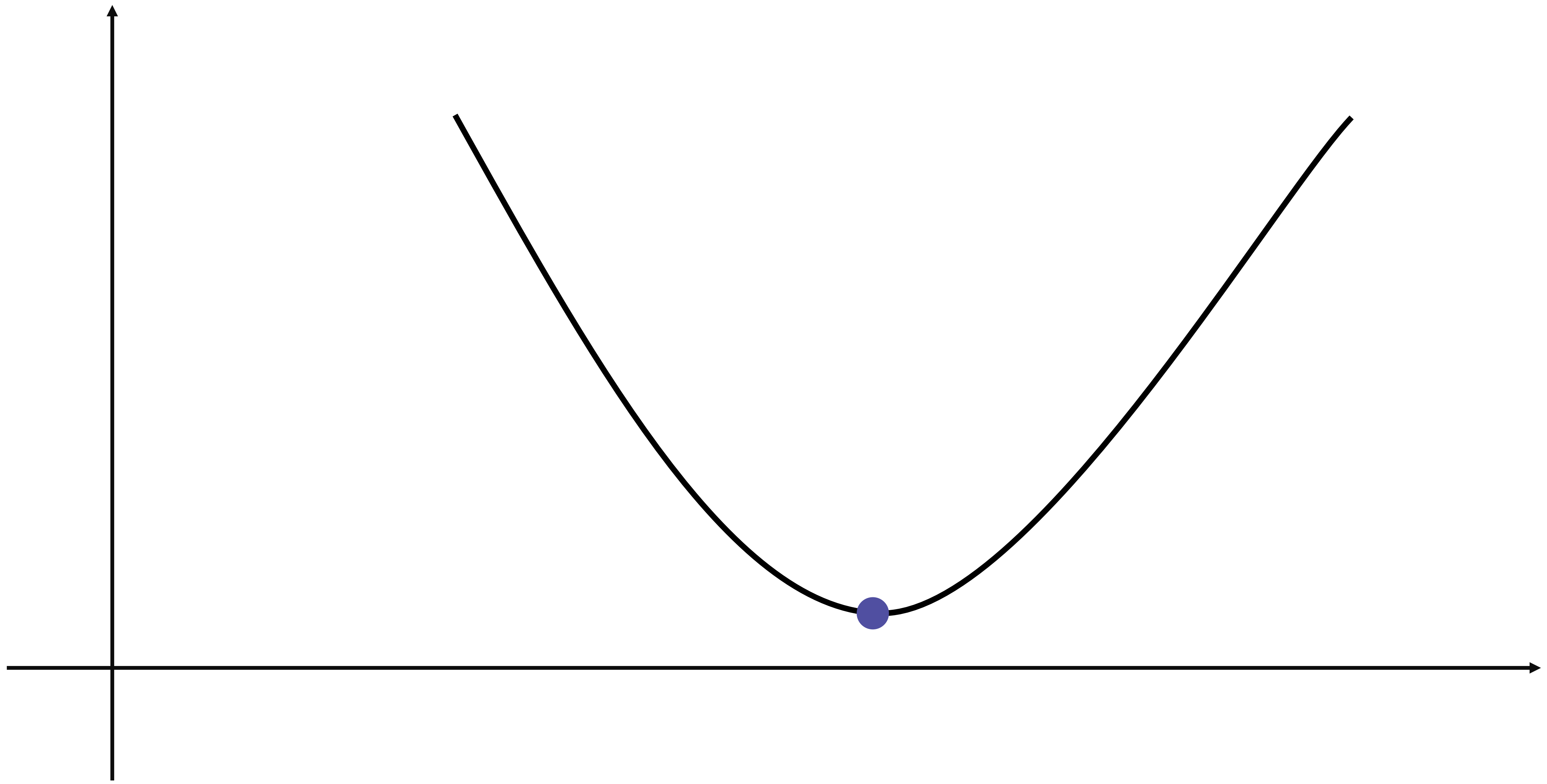
02

Gradient Descent란?



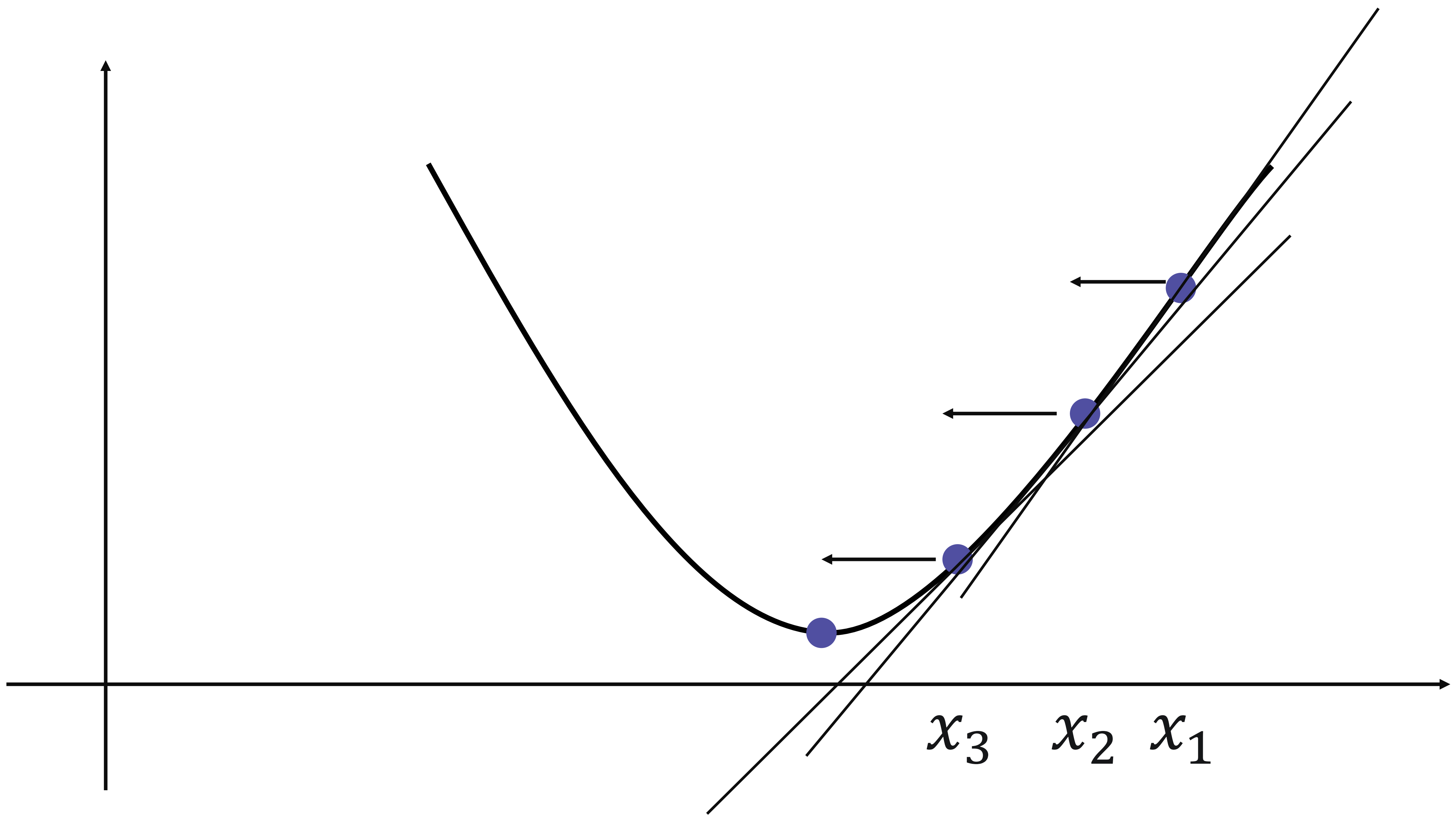
Gradient란?

- 접선의 기울기를 의미함.
- Gradient Descent는 **경사하강법**이라고도 부르며, 목적 함수의 최소화가 목표일 때 사용함.



*목적 함수($f(x)$)의 최대화가 목표일 때는 $-f(x)$로 치환해서 항상 최소화 문제로 바꿀 수 있음.

- 초기 지점을 출발으로 1차 미분계수(gradient)를 이용해 최소값으로 이동해 나감.
- 접선의 기울기를 이동 거리(step size; γ)만큼씩 곱해서 이동함.

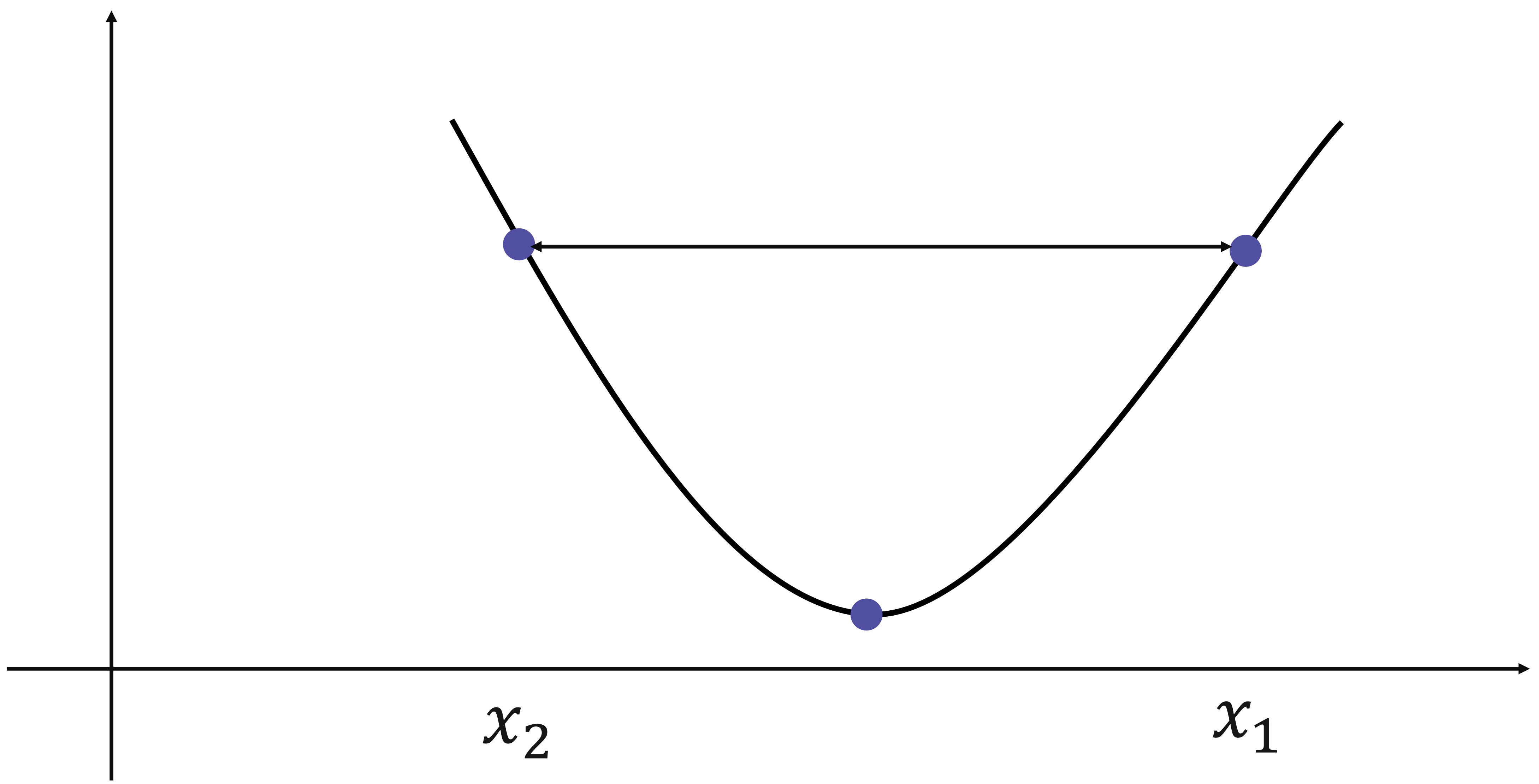


첫번째 이동거리: $\gamma \frac{df(x)}{dx}\Big|_{x=x_1}$

두번째 이동거리: $\gamma \frac{df(x)}{dx}\Big|_{x=x_2}$

세번째 이동거리: $\gamma \frac{df(x)}{dx}\Big|_{x=x_3}$

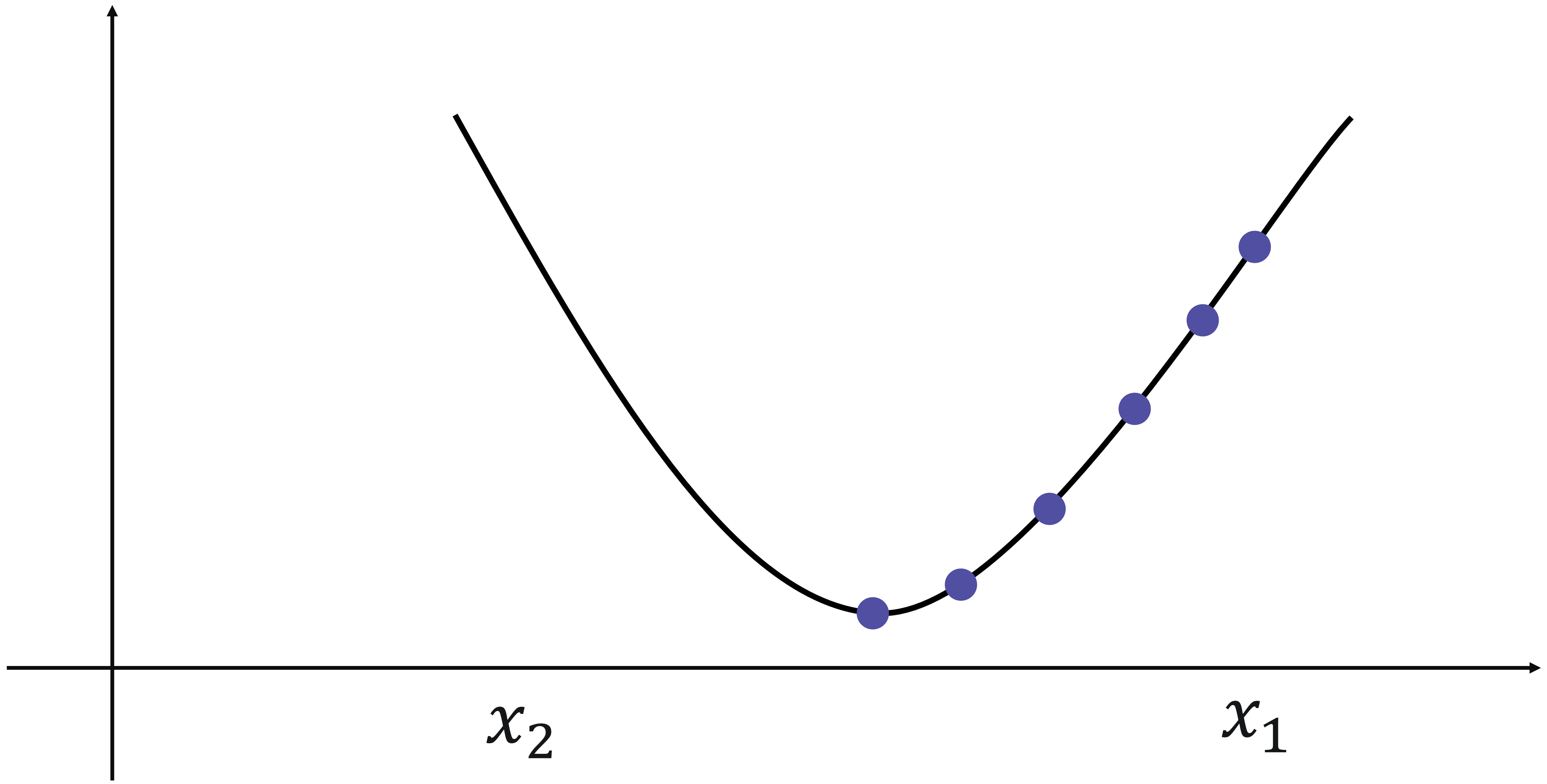
- Step size가 너무 크면 최소값으로 수렴되지 않음.
- 반대로, 너무 작으면 수렴하는데 오래 걸림.
- 적절한 Step size를 잡아주는 것이 중요.
- 최적화 문제에서의 목적함수를 아래처럼 푼다고 해석하면 됨.



Worst case

*영원히 최소값에 도달하지 못함.

- 결국, 최적화 문제는 목적 함수가 최소값을 가지게 하는 x를 찾는 문제임.
- Gradient Descent는 이 x를 찾는 알고리즘 중 하나임.



*step size를 작게 하면, 수렴하는데 오래 걸림.

- 왜 사용하는가? 바로 최소값을 구하면 되는 것 아닌가?

1. **실제 분석에서 마주하는 함수는 굉장히 복잡하다. 당연히, 아래처럼 2차원 그림으로 표현 못하는 고차원 함수들이 많다. (예: $e^{x_1+x_2-x_3} + \log_{x_3} x_1 + x_2 x_3$)**
2. 미분 계수 구하는 과정이 컴퓨터가 gradient descent가 더 편하다.
3. 데이터가 많을 때는 계산량 측면에서 더욱 효율적이다.

- 왜 사용하는가? 바로 최소값을 구하면 되는 것 아닌가?

1. 실제 분석에서 마주하는 함수는 굉장히 복잡하다. 당연히, 아래처럼 2차원 그림으로 표현 못하는 고차원 함수들이 많다. (예: $e^{x_1+x_2-x_3} + \log_{x_3} x_1 + x_2 x_3$)
2. **미분 계수 구하는 과정이 컴퓨터를 이용한 gradient descent가 더 편하다.**
3. 데이터가 많을 때는 계산량 측면에서 더욱 효율적이다.

- 왜 사용하는가? 바로 최소값을 구하면 되는 것 아닌가?

1. 실제 분석에서 마주하는 함수는 굉장히 복잡하다. 당연히, 아래처럼 2차원 그림으로 표현 못하는 고차원 함수들이 많다. (예: $e^{x_1+x_2-x_3} + \log_{x_3} x_1 + x_2 x_3$)
2. 미분 계수 구하는 과정이 컴퓨터가 gradient descent가 더 편하다.
3. **데이터가 많을 때는 계산량 측면에서 더욱 효율적이다.**

03

Gradient Descent 수행하기

Gradient Descent 실습을 위한 라이브러리 불러오기

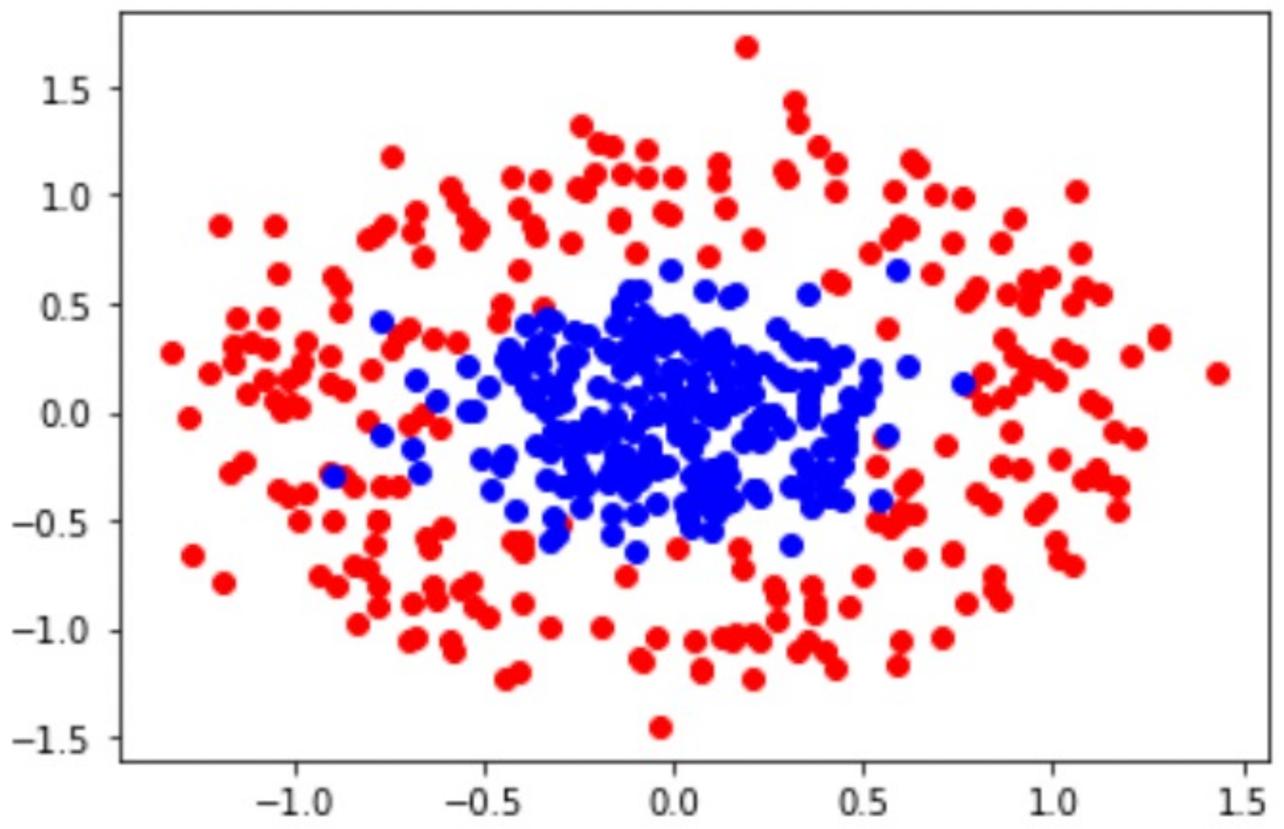
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
```

데이터 만들기

```
n_pts = 500
x, y = datasets.make_circles(n_samples=n_pts, random_state=123,
noise=0.2, factor=0.3)

x_data = np.array(X)
y_data = np.array(y.reshape(500, 1))
```

데이터 출력하기

```
def scatter_plot():
plt.scatter(X[y==0, 0], X[y==0, 1], color='red')
plt.scatter(X[y==1, 0], X[y==1, 1], color='blue')

scatter_plot()
```



목표: 빨간 점과 파란 점을 분리하는 경계(decision boundary) 만들기
아래와 같은 pipeline을 만든다.

입력	함수	출력
(x_1, x_2)	$f(x_1, x_2)$	0 or 1

0 : 빨간 점
1 : 파란 점

목적 함수 = ∑ (데이터 예측값 –데이터 실제값)^2 (혹은 합을 대신한 평균)

목표: 빨간 점과 파란 점을 분리하는 경계(decision boundary) 만들기

$f(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2 + d$ 와 같은 함수를 정한다.

Gradient Descent를 통해 목적함수를 최소로 하는 a, b, c, d를 찾는다.

입력	함수	출력
(x_1, x_2)	$f(x_1, x_2)$	0 or 1

0 : 빨간 점
1 : 파란 점

목적 함수 = (모든 데이터 예측값 합 – 모든 데이터의 실제값 합)^2

입력
(x_1, x_2)

함수
$f(x_1, x_2)$

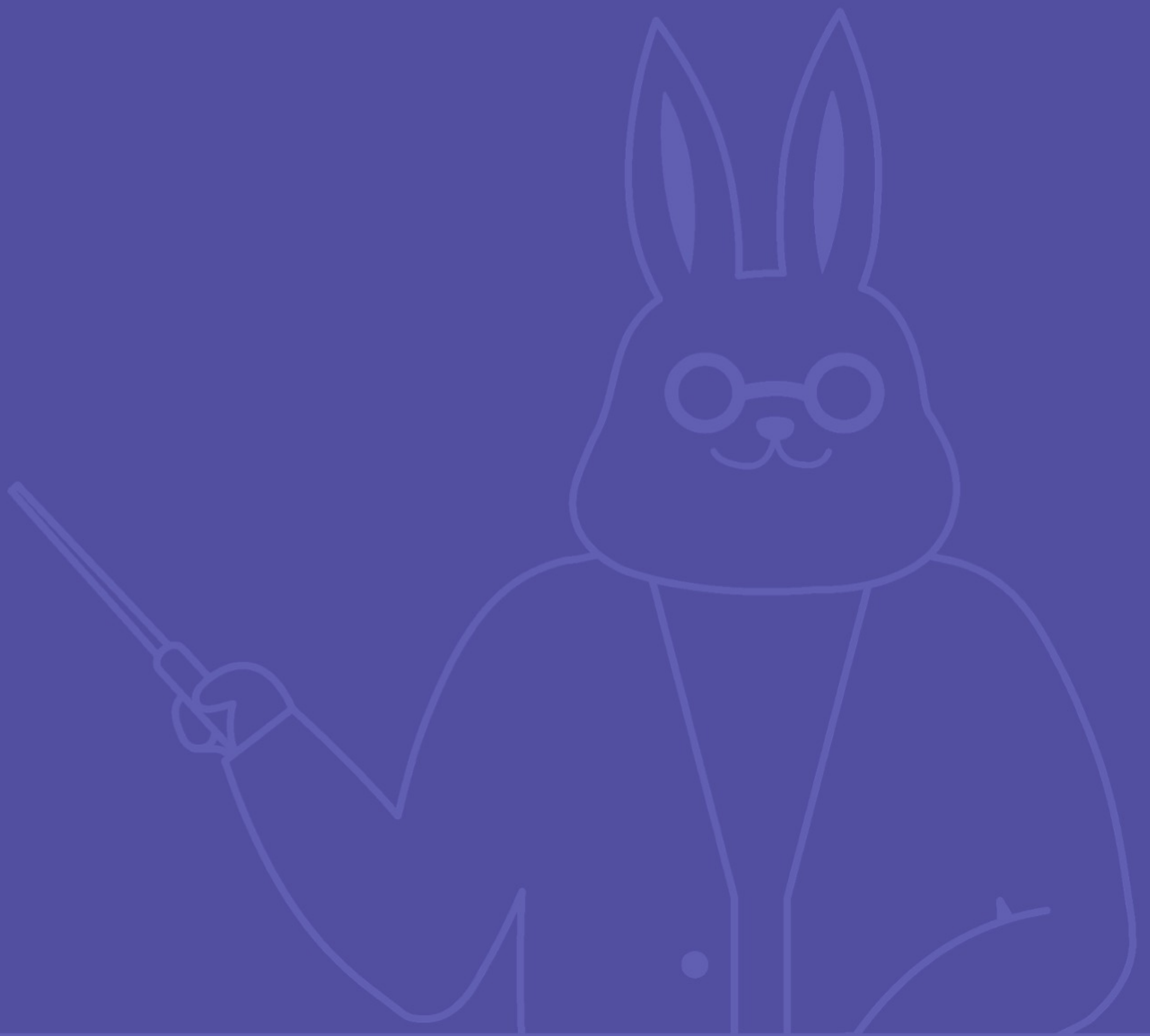
출력
0
or
1

0 : 빨간 점
1 : 파란 점

노트

04

예시 문제

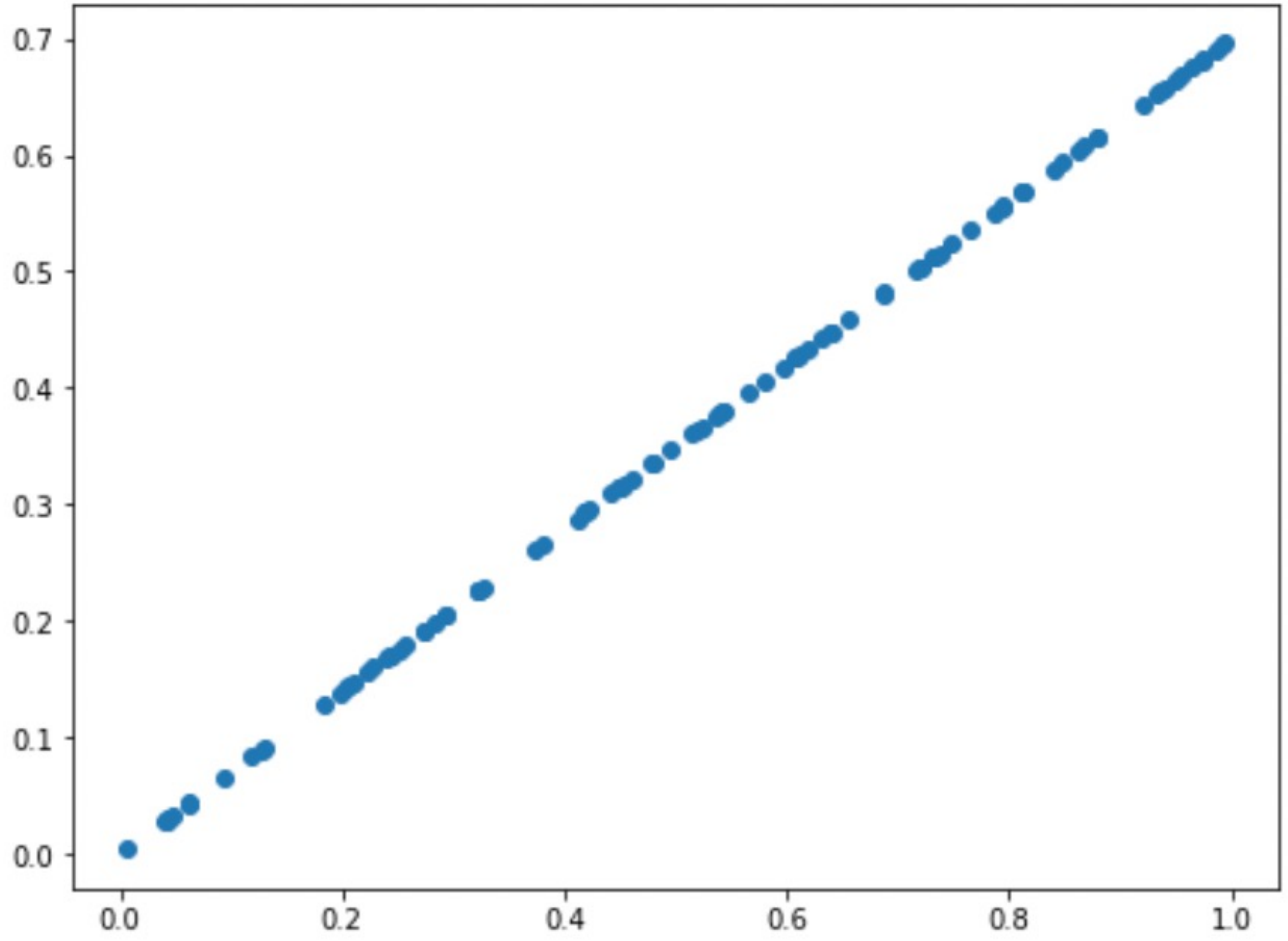


코드를 통해 Gradient Descent를 푸는 예시문제 살펴보기

```
import numpy as np
import matplotlib.pyplot as plt

x = np.random.rand(100)
y = 0.7 * x
```

문제 시각화 : 그림의 **(x,y)**를 예측하는 함수 만들기. 주어진 **x**에 대응되는 **y** 예측하기.

```
plt.figure(figsize=(8,6))
plt.scatter(x,y)
plt.show()
```

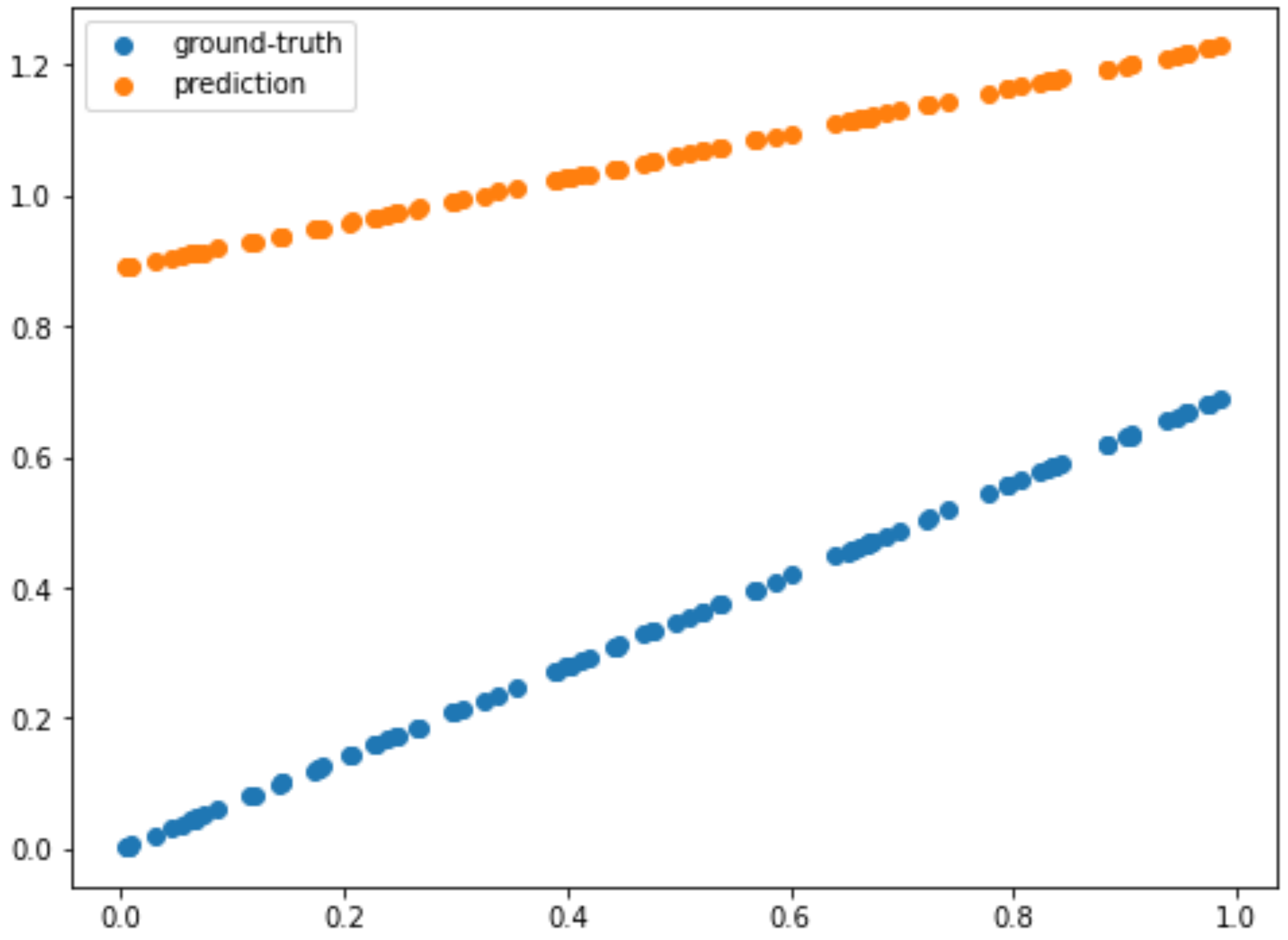


Y=Wx + b로 예상하여, W와 b를 gradient descent를 통해 찾을 것임.

```
W = np.random.uniform(0,1)
b = np.random.uniform(0,1)

step_size = 0.5

for epoch in range(100):
    y_pred = W * x + b

    objective = np.abs(y_pred - y).mean()

    w_grad = step_size * ((y_pred-y)*x).mean()
    b_grad = step_size * (y_pred-y).mean()

    W = W - w_grad
    b = b - b_grad
```

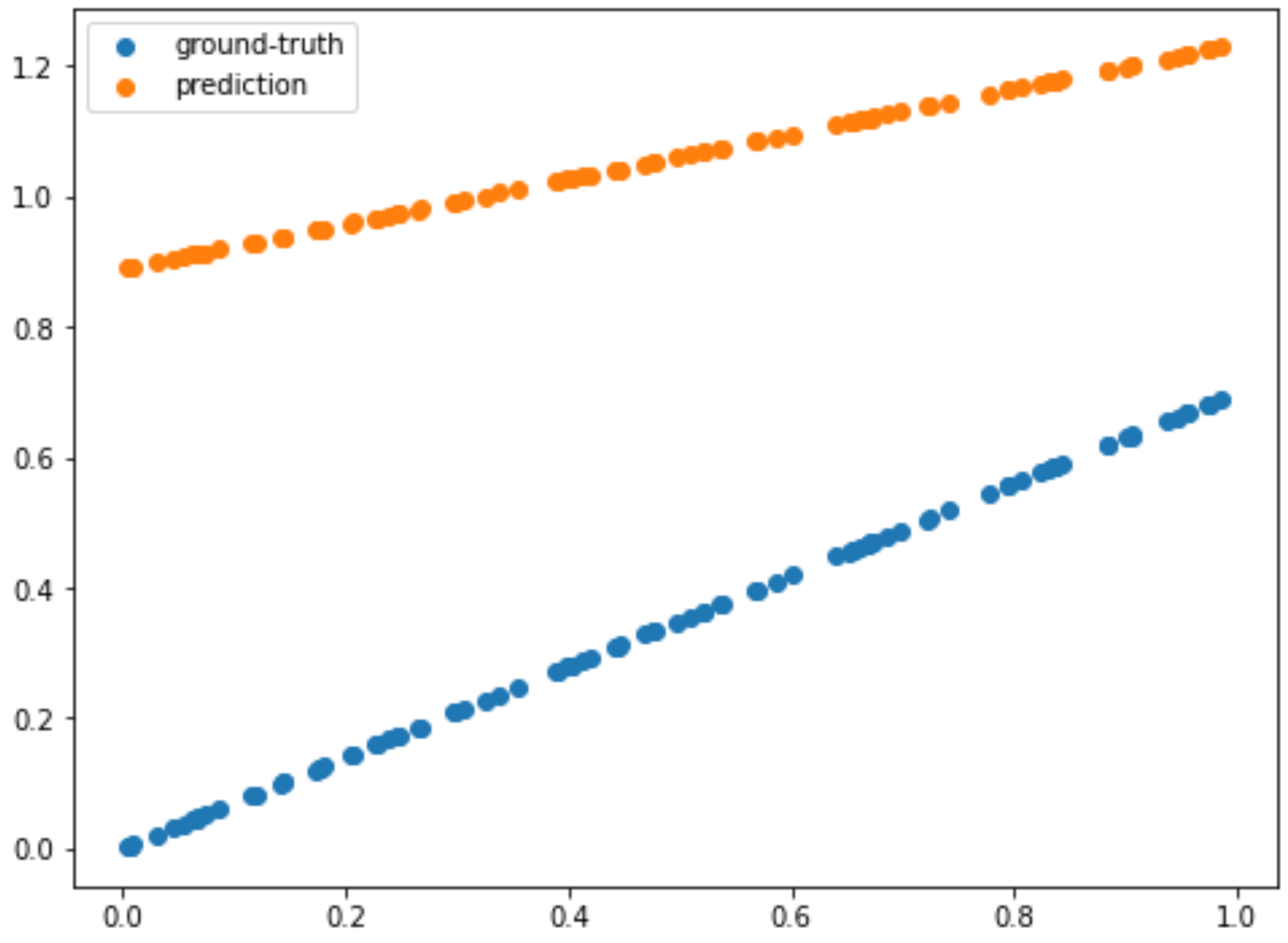
학습 전 후의 예측함수의 변화 과정 시각화

```
def compare_pred(x, pred, y):
    plt.figure(figsize=(8,6))
    plt.scatter(x,y, label = 'ground-truth')
    plt.scatter(x,pred, label = 'prediction')
    plt.legend()
    plt.show()

pred = W * x + b
compare_pred(x, pred, y)
```

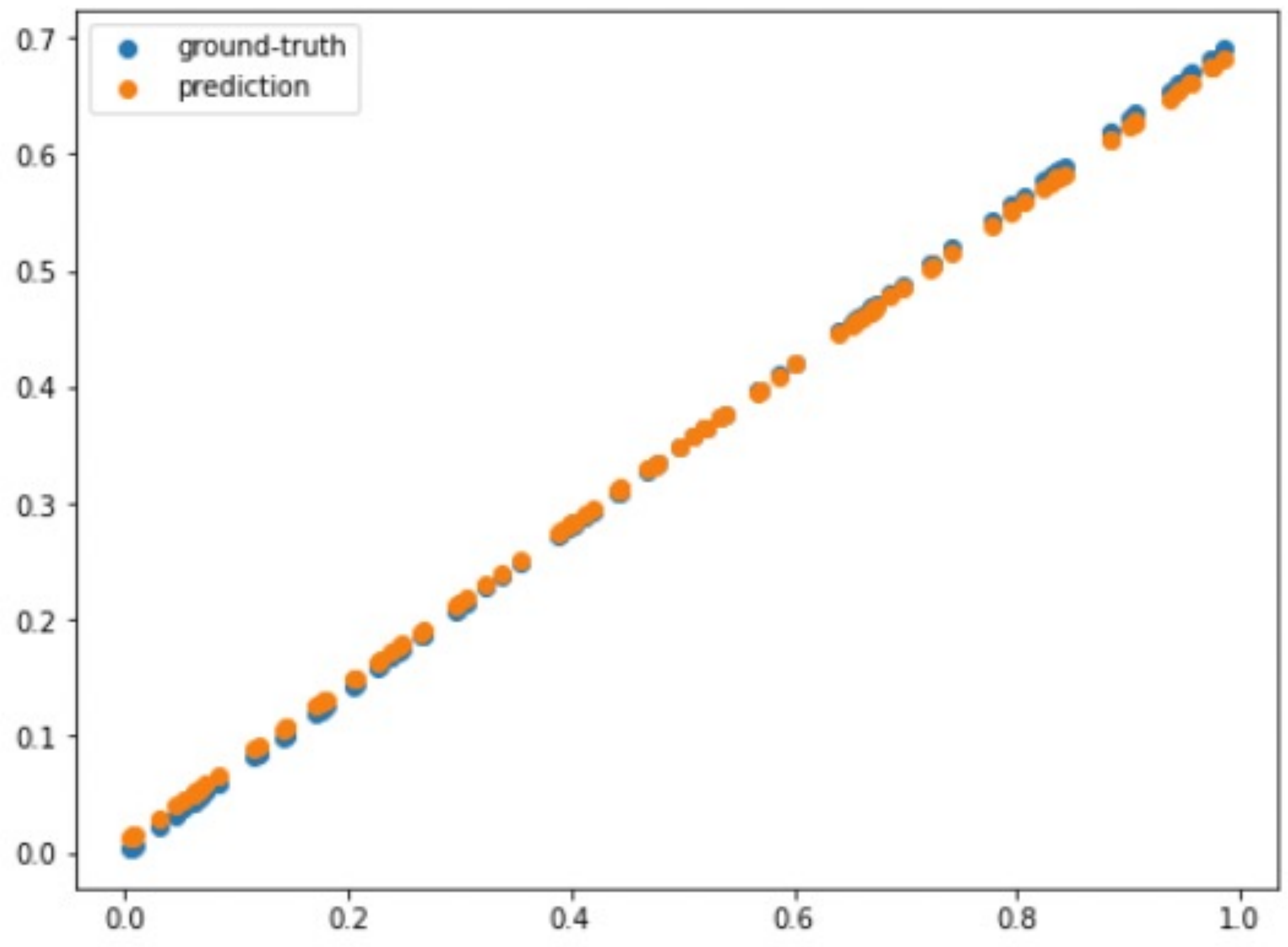


학습 전 시각화 모습

학습 전 후의 예측함수의 변화 과정 시각화



학습 전 시각화 모습



학습 후 시각화 모습

크레딧

/* elice */

코스 매니저

콘텐츠 제작자

강사

감수자

디자이너

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io