머신 러닝 – 수치 미분

2021.07.14 AAI Lab. 세미나

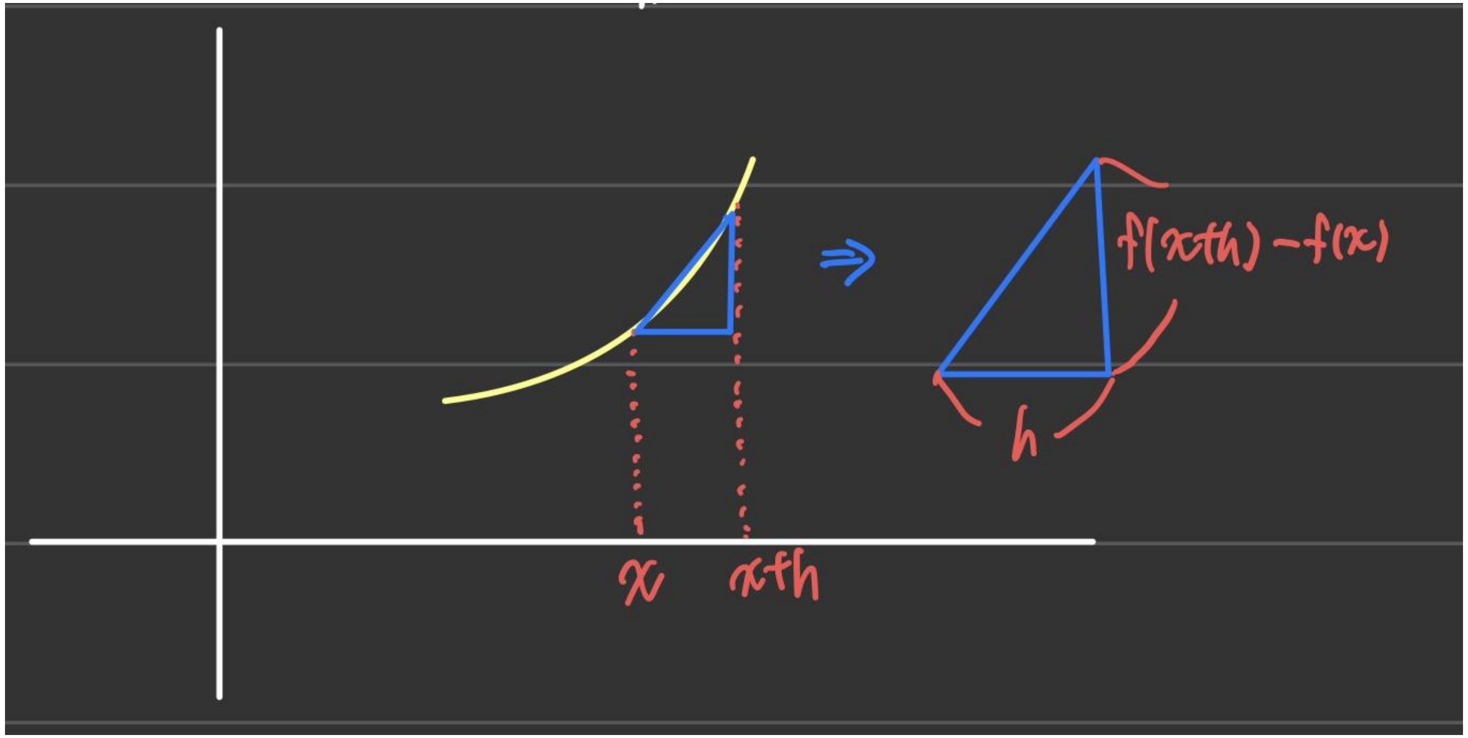
미분?

- 입력 x를 현재 값에서 아주 조금 변화(h) 시키면, 함수 $f(x)$의 값이 변하는 정도를 나타냄.

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

- 순간변화율, 접선의 기울기, 미분계수

미분?



편미분?

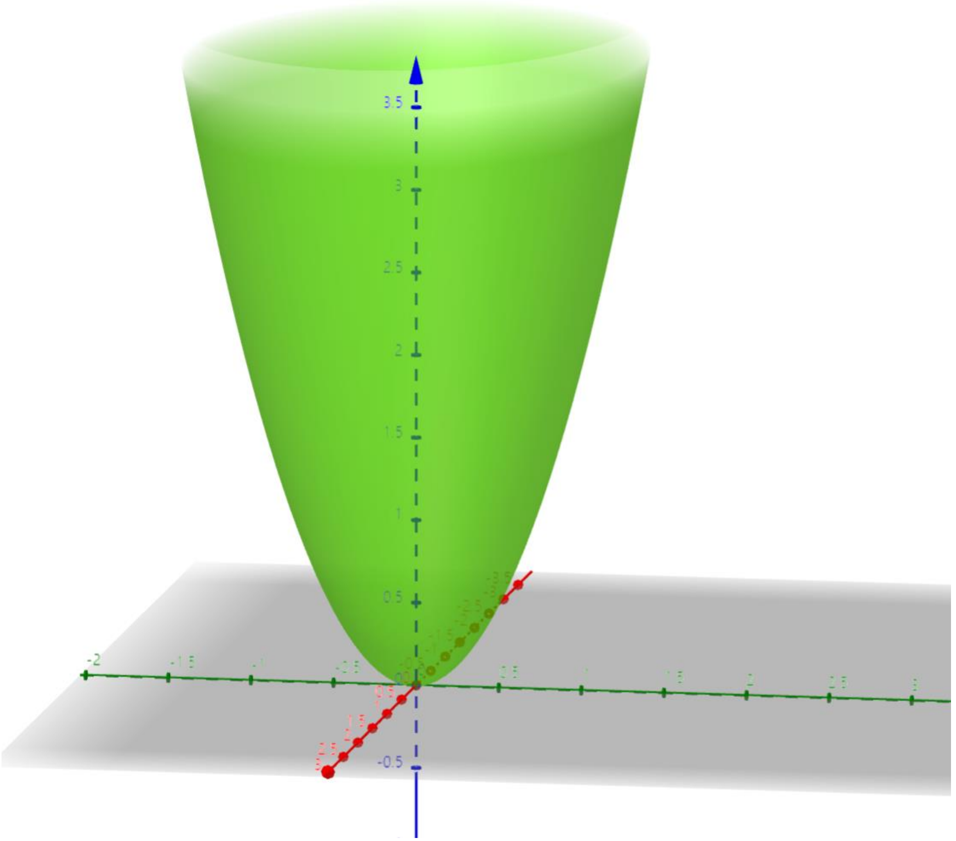
- 다변수 함수에서 각 변수에 따른 함수 값의 변화율
- ∂ : “partial” 이라고 읽음

- $f_x(x, y) = \frac{\partial}{\partial x} f(x, y) = \lim_{h \to 0} \frac{f(x+h, y) - f(x, y)}{h}$ // x로 편미분
- $f_y(x, y) = \frac{\partial}{\partial y} f(x, y) = \lim_{h \to 0} \frac{f(x, y+h) - f(x, y)}{h}$ // y로 편미분

- 미분하지 않는 나머지 변수는 고정점(상수)로 취급

편미분 해석적 의미

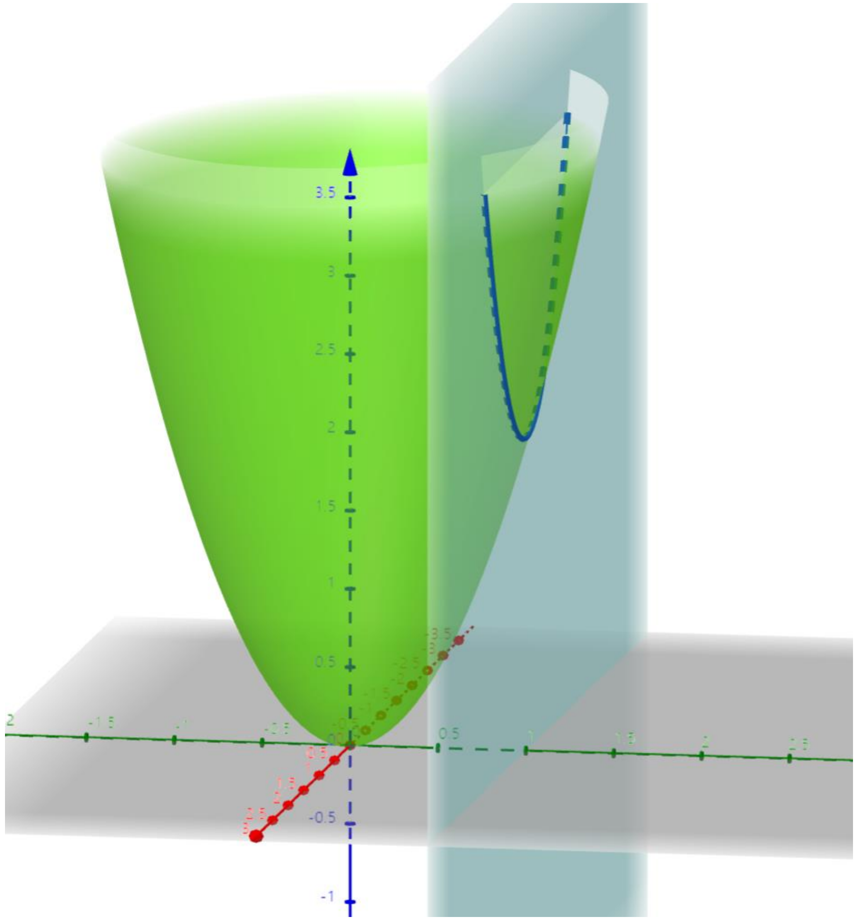
$$z = f(x, y) = x^2 + 2y^2$$



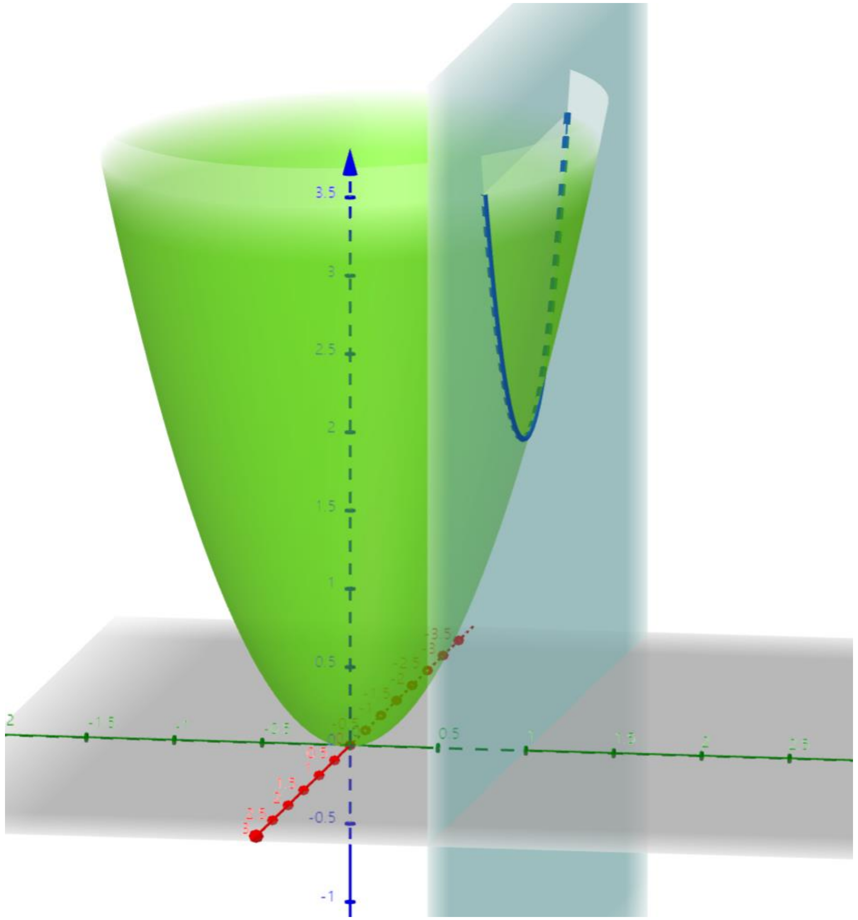
$$f_x(2,1) = ?$$

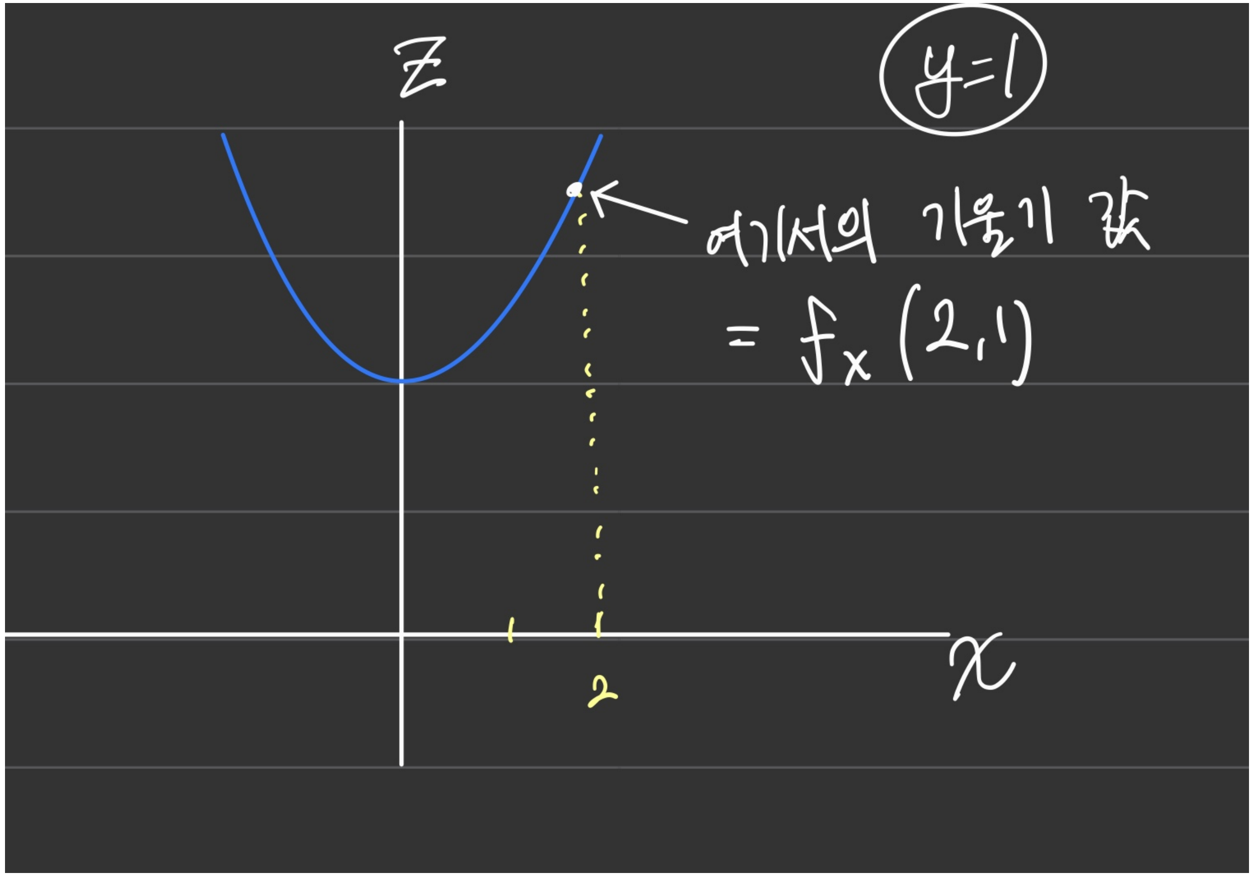
편미분 해석적 의미

$f_x(2,1) = ?$

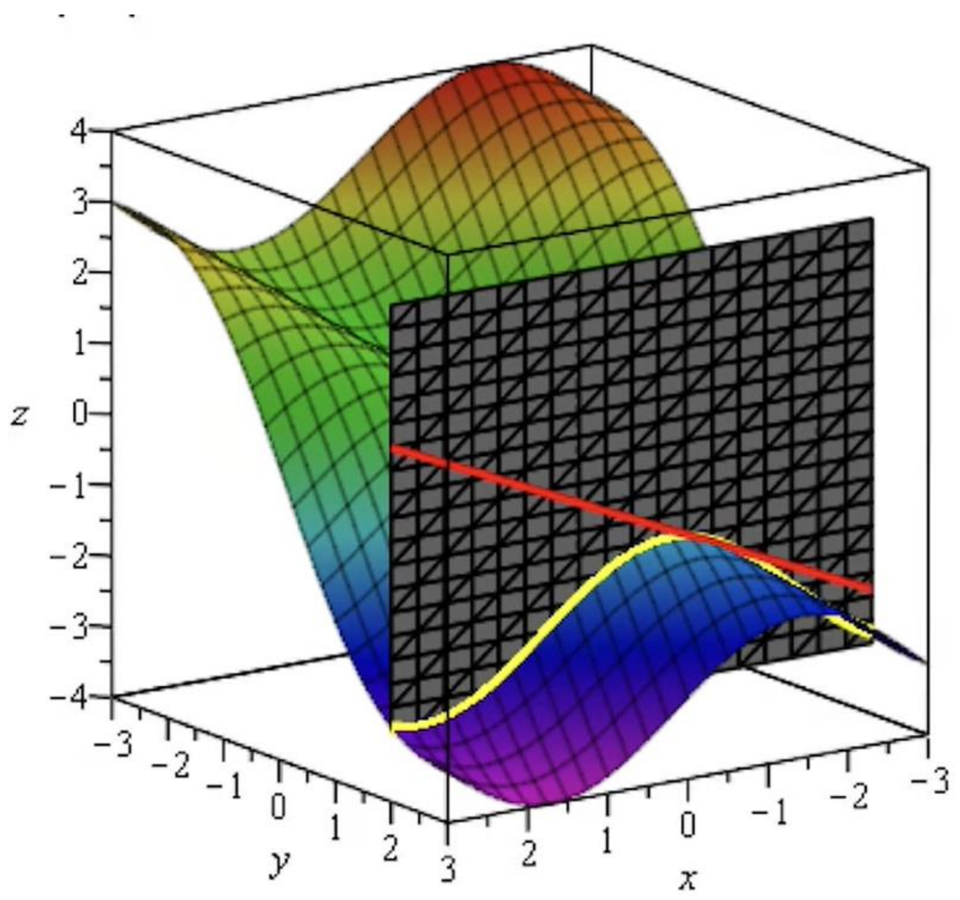


편미분 해석적 의미





편미분 해석적 의미



$f_x(-1, 2) = ?$

수치 미분

- 전방 차분 (forward scheme)

$$\lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$$

- 후방 차분 (backward scheme)

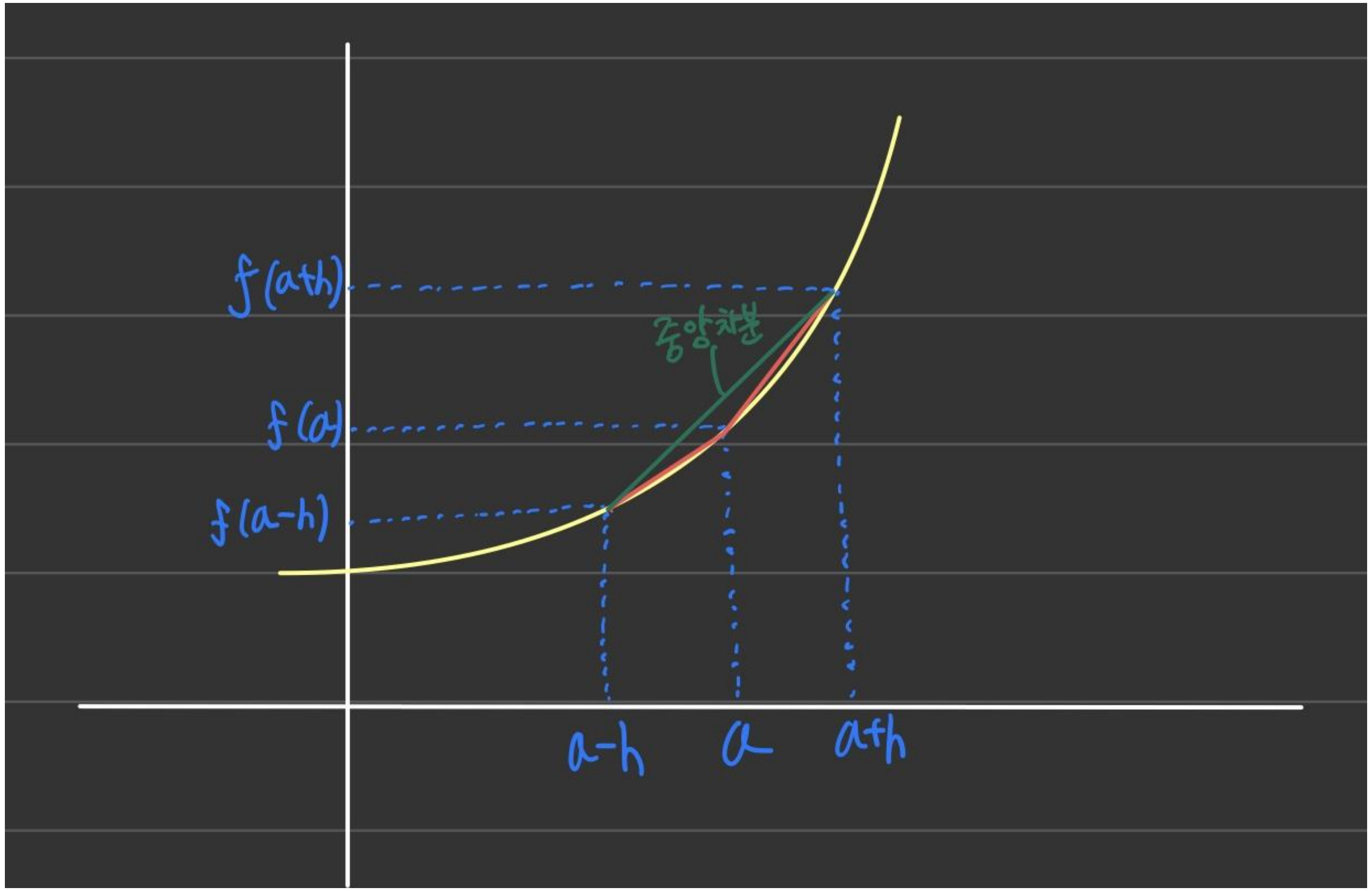
$$\lim_{h \to 0} \frac{f(a) - f(a-h)}{h}$$

- 중앙 차분 (central scheme)

$$\lim_{h \to 0} \frac{f(a+h) - f(a-h)}{2h}$$

중앙 차분을 사용하는 이유

- 오차가 가장 적다



증명

중앙 차분 구현

```
In [6]: def numerical_diff(f, x):
            h = 10e-4
            return (f(x+h)-f(x-h))/(2*h)

        def f(x):
            return x**2

        numerical_diff(f,x)
```

```
3.9999999999995595
```

$f(x) = x^2, f'(x) = 2x \rightarrow f'(2) = 4$

아주 작은 오차가 있으므로 잘 구현되었다고 할 수 있다.

이론적으로 h 값이 작을수록 오차가 작아지지만, 컴퓨터의 계산문제(부동 소수점)때문에 일반적으로 $h = 10^{-4}$ 를 사용하면 좋은 성능을 보인다고 알려져 있다.

Gradient?

- ∇ : "**gradient**" or "nabla" or "del"
- 기울기 벡터

$\mathbb{R}^3$에서 gradient연산자는 $\nabla = \frac{\partial}{\partial x}\mathbf{i} + \frac{\partial}{\partial y}\mathbf{j} + \frac{\partial}{\partial z}\mathbf{k}$ 로 정의된다.

($\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ 는 $\mathbb{R}^3$ 단위 직교 기저)

→ $\nabla f(x, y, z) = \left\langle \frac{\partial f(x,y,z)}{\partial x}, \frac{\partial f(x,y,z)}{\partial y}, \frac{\partial f(x,y,z)}{\partial z} \right\rangle$

Gradient?

- 다변수 함수의 모든 입력 값에서 모든 방향에 대한 순간변화율
- 특징: gradient가 가리키는 방향은 순간변화율이 가장 큰 방향

(경사가 가장 가파른 방향)

"산을 오를 때(내려갈 때) gradient의 방향으로 간다면 정상에 가장 빠르게 도착한다" (이동거리가 가장 짧다)

→ gradient descent 방법

Gradient 구현

In [ ]:

```
import numpy as np

def numerical_gradient(f,x):
    h = 1e-4
    grad = np.zeros_like(x)

    for idx in range(x.size):
        tmp_val = x[idx]

        x[idx] = tmp_val + h
        fxh1 = f(x)

        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2)/(2*h)
        x[idx] = tmp_val
    return grad
```