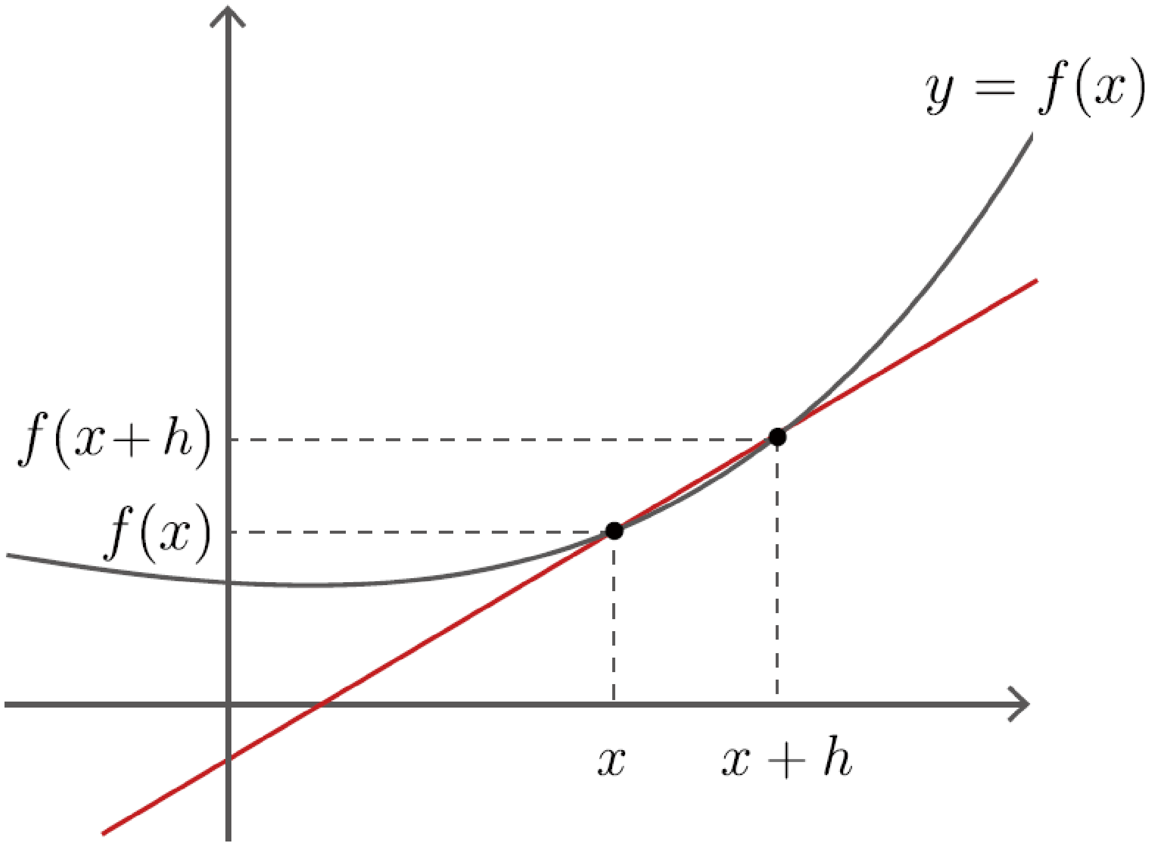
제1 고지 : 미분 자동 계산

STEP 4 : 수치 미분

4.1 미분이란

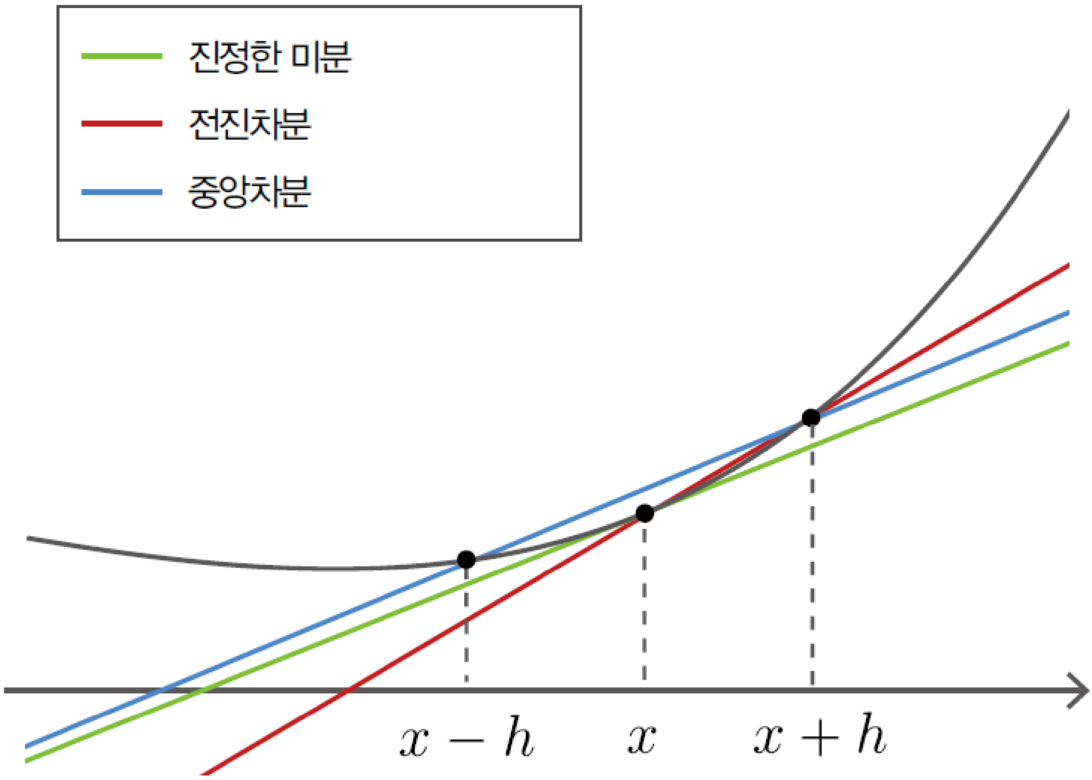
그림 4-1 곡선 $y = f(x)$ 위의 두 점을 지나는 직선



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

- **평균변화율의 극한 = 순간 변화율** 로, 어떤 시스템(함수)이 있을때, 이 시스템이 어떤 변수(요인)에 의해 어떻게 영향을 받는지를 분석하는 도구
- 예를 들어, $f'(0.5) = 3.297$ 의 의미는 x 를 0.5 에서 작은 값 만큼 변화시키면 y 는 3.297 배만큼 영향 ### 4.2 수치미분 구현

그림 4-2 진정한 미분, 전진차분, 중앙차분 비교



- 컴퓨터는 극한을 취할 수 없으므로 h 를 극한과 비슷한 **1e-4 와 같은 매우 작은 값**을 이용하여 계산하는데, 이런 미세한 차이를 이용하여 미분 값을 근사하여 구하는 방법이 `수치 미분(numerical differentiation)`
- 차분을 구하는 방법으로는 `forward difference(전진 차분)` 와 `centered difference(중앙 차분)` 있는데, `centered difference` 를 적용하는 것이 더 근사하다 (Taylor series 를 통한 증명)
  - `forward difference` : $x \sim x+h$
  - `centered difference` : $x-h \sim x+h$

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{(x+h) - (x-h)} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

In [ ]:

```
import torch
import numpy as np
import torch.nn as nn


class Variable:
    def __init__(self, data: np.ndarray) -> None:
        self.data = data

class Function:
    """
    Function Base Class
    """

    def __call__(self, input: Variable) -> Variable:
        x = input.data # 입력 변수
        y = self.forward(x) # 구체적 계산
        return Variable(y) # 출력 변수

    def forward(self, x):
        """
```

```
        구체적인 함수 계산 담당
                # NOTE : 0차원의 ndarray 의 경우 np.float64로 변환되는데(넘파이가 의도
        """
        raise NotImplementedError


class Exp(Function):
    """
    y=e ^ x
    """

    def forward(self, x: np.ndarray) -> np.ndarray:
        return np.exp(x)

class Square(Function):
    """
    y= x ^ 2
    """

    def forward(self, x: np.ndarray) -> np.ndarray:
        return x**2


class Sigmoid(Function):
    """
    y = 1 / (1 + e ^(-x))
    """

    def forward(self, x: np.ndarray) -> np.ndarray:
        return 1 / (1 + np.exp(-x))


class Tanh(Function):
    """
    y= ( e^x - e^{-x} ) / ( e^x + e^{-x} )
    """

    def forward(self, x: np.ndarray) -> np.ndarray:
        return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))
```

In [ ]:

```
def numerical_diff(f: Function, x: Variable, eps: float = 1e-4):
    """
    calculate centered difference
    """
    x0 = Variable(x.data - eps)  # x - h
    x1 = Variable(x.data + eps)  # x + h
    y0 = f(x0)
    y1 = f(x1)
    return (y1.data - y0.data) / (2 * eps)  # (f(x+h) - f(x-h)) / 2h
```

4.3 합성 함수의 미분

In [ ]:

```
def f_composition(x: Variable) -> Variable:
    A = Square()
    B = Exp()
    C = Square()
    return C(B(A(x)))


x = Variable(np.array(0.5))
```

```
dy = numerical_diff(f_composition, x)
print(dy)
```

```
3.2974426293330694
```

4.4 수치 미분의 문제점

- 수치 미분의 결과에는 오차가 포함되어 있는데, 어떤 계산인지에 따라 오차가 커질 수 있다.
- 다변수 미분할 경우 변수 각각을 미분해야 하기때문에, 매개변수를 수백만 개 이상 사용하는 신경망에서는 비효율적이다.

💡 이러한 문제를 해결하기 위해서 등장한 것이 **역전파(backpropagation)** 인데, 수치 미분은 역전파 계산을 테스트 하기 위해서 **gradient checking** 하는데 활용될 수 있다.