제1 고지 : 미분 자동 계산

STEP 10 : 테스트

10.1 파이썬 단위 테스트

- `unittest` 를 이용해 단위테스트를 진행할때, **test 로 시작하는 메서드**를 만든다.

In [ ]:

```
import unittest
import numpy as np


def as_array(x):

    """
    0차원 ndarray / ndarray가 아닌 경우
    """
    if np.isscalar(x):
        return np.array(x)
    return x


class Variable:
    def __init__(self, data: np.ndarray) -> None:
        if data is not None:
            if not isinstance(data, np.ndarray):
                raise TypeError(f"{type(data)}은(는) 지원하지 않습니다.")
        self.data = data
        self.grad = None  # gradient
        self.creator = None  # creator

    def set_creator(self, func) -> None:
        self.creator = func

    def backward(self):
        """
        자동 역전파 (반복)
        """
        if self.grad is None:
            self.grad = np.ones_like(self.data)
        funcs = [self.creator]
        while funcs:
            f = funcs.pop()  # 1. 함수를 가져온다
            x, y = f.input, f.output  # 2. 함수의 입력 / 출력을 가져온다
            x.grad = f.backward(y.grad)  # 3. 역전파를 계산한다

            if x.creator is not None:
                funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다.


class Function:
    """
    Function Base Class

    """

    def __call__(self, input: Variable) -> Variable:
        x = input.data
```

```
        y = self.forward(x)
        self.input = input  # 역전파 계산을 위해 입력변수 보관
        output = Variable(as_array(y))
        output.set_creator(self)  # 출력 변수에 creator 설정 ( 연결을 동적으로 만드는
        self.output = output  # 출력도 저장

        return output

    def forward(self, x: np.ndarray) -> np.ndarray:
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gy: np.ndarray) -> np.ndarray:
        """
        역전파
        """
        raise NotImplementedError()


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

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

    def backward(self, gy: np.ndarray) -> np.ndarray:
        x = self.input.data
        gx = 2 * x * gy
        return gx


def square(x):
    f = Square()
    return f(x)


class SquareTest(unittest.TestCase):
    def test_forward(self):
        x = Variable(np.array(2.0))
        y = square(x)
        expected = np.array(4.0)

        self.assertEqual(y.data, expected)

    def test_backward(self):
        x = Variable(np.array(3.0))
        y = square(x)
        y.backward()
        expected = np.array(6.0)
        self.assertEqual(x.grad, expected)

    def test_gradient_check(self):
        x = Variable(np.random.rand(1))
        y = square(x)
        y.backward()
        num_grad = numerical_diff(square, x)
        flg = np.allclose(x.grad, num_grad)
        self.assertTrue(flg)
```

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

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

    def backward(self, gy: np.ndarray) -> np.ndarray:
        x = self.input.data
        gx = np.exp(x) * gy
        return gx


def exp(x):
    f = Exp()
    return f(x)


def numerical_diff(f: Function, x: Variable, eps: float = 1e-4) -> np.ndarr
    """
    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
```

In [ ]:

```
import unittest
class SquareTest(unittest.TestCase):
    def test_forward(self):
        x = Variable(np.array(2.0))
        y = square(x)
        expected = np.array(4.0)
```

10.2 square 함수의 역전파 테스트

In [ ]:

```
class SquareTest(unittest.TestCase):
    def test_forward(self):
        x = Variable(np.array(2.0))
        y = square(x)
        expected = np.array(4.0)

        self.assertEqual(y.data, expected)

    def test_backward(self):
        x = Variable(np.array(3.0))
        y = square(x)
        y.backward()
        expected = np.array(6.0)
        self.assertEqual(x.grad, expected)
```

10.3 기울기 확인을 위한 자동 테스트

In [ ]:

```
class SquareTest(unittest.TestCase):
    def test_forward(self):
        x = Variable(np.array(2.0))
        y = square(x)
        expected = np.array(4.0)
```

```
        self.assertEqual(y.data, expected)

    def test_backward(self):
        x = Variable(np.array(3.0))
        y = square(x)
        y.backward()
        expected = np.array(6.0)
        self.assertEqual(x.grad, expected)

    def test_gradient_check(self):
        x = Variable(np.random.rand(1))
        y = square(x)
        y.backward()
        num_grad = numerical_diff(square, x)
        flg = np.allclose(x.grad, num_grad)
        self.assertTrue(flg)
```

10.4 테스트 정리

- 특정 디렉토리에 있는 모든 테스트 파일들은 다음 명령으로 한꺼번에 실행
- 기본적으로는 지정한 디렉토리에서 이름이 `test*.py` 형태인 파일을 인식

  `$ python -m unittest discover [폴더]`