제1 고지 : 미분 자동 계산

STEP 6 : 수동 역전파

6.1 Variable 클래스 추가 구현

- 순전파 시 `data` 와 더불어 이에 대응하는 미분값(`grad`)도 저장

In [ ]:
```
import numpy as np
class Variable:
    def __init__(self, data: np.ndarray) -> None:
        self.data = data
        self.grad = None  # gradient
```

6.2 Function 클래스 추가 구현

- 역전파 계산을 위해 `input` 을 저장
- 역전파 기능 `backward()` 추가

In [ ]:
```
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
        return Variable(y)

    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()
```

6.3 Square 와 Exp 클래스 추가 구현

In [ ]:
```
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


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
```

6.4 역전파 구현

그림 6-1 역전파할 대상(합성 함수)

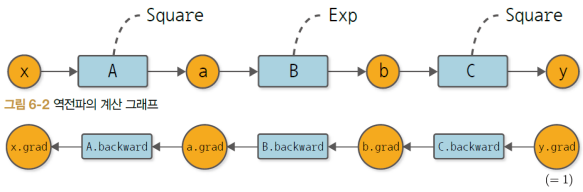

그림 6-2 역전파의 계산 그래프

In [ ]:

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


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


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
        return Variable(y)

    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()
```

```python
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


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


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

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

    def backward(self, gy: np.ndarray) -> np.ndarray:
        """
        d/dx sigmoid(x) = sigmoid(x)(1-sigmoid(x))
        """
        x = self.input.data
        sigmoid = lambda x: 1 / (1 + np.exp(-x))
        return gy * sigmoid(x) * (1 - sigmoid(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))

    def backward(self, gy: np.ndarray) -> np.ndarray:
        """
        d/dx tanh(x) = 1-tanh(x)^2
        """
        x = self.input.data
        tanh = lambda x: (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x)
        return gy * (1 - tanh(x) ** 2)


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

```
    """
    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


# Dezero
A = Square()
B = Exp()
C = Square()


x = Variable(np.array(0.5))

#  순전파
a = A(x)
b = B(a)
y = C(b)

# 역전파
y.grad = np.array(1.0)
b.grad = C.backward(y.grad)
a.grad = B.backward(b.grad)
x.grad = A.backward(a.grad)
print(x.grad)
```

```
3.297442541400256
```

In [ ]:

```
# Dezero ~ Pytorch
## Dezero
A = Tanh()
B = Sigmoid()

x = Variable(np.array(1))

# 순전파
a = A(x)
b = B(a)

# 역전파
b.grad = np.array(1.0)
a.grad = B.backward(b.grad)
x.grad = A.backward(a.grad)
print(f"Dezero : {x.grad}")

## Pytorch
x = torch.tensor([1.0], requires_grad=True)
A = nn.Tanh()
B = nn.Sigmoid()
a = A(x)
b = B(a)
b.backward()  #  NOTE : step07 에서 Dezero Variable 클래스에서 해당 기능(역전파 자동
print(f"PyTorch : {x.grad}")
```

```
Dezero : 0.09112821805819912
PyTorch : tensor([0.0911])
```