제1 고지 : 미분 자동 계산

STEP 8 : 재귀에서 반복문으로

- 복잡한 계산 그래프를 다루는데, 재귀적 역전파 계산은 효율성이 떨어진다.
  ### 8.1 현재의 Variable 클래스

In [ ]:
```
import numpy as np
class Variable:
    def __init__(self, data: np.ndarray) -> None:
        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 (재귀)
        """
        f = self.creator  # 1. 함수를 가져온다
        if f is not None:
            x = f.input  # 2. 함수의 입력을 가져온다
            x.grad = f.backward(self.grad)  # 3. 역전파를 계산한다
            x.backward()  # 하나 앞 변수의 backward 메서드를 호출한다 (재귀)
        # NOTE : 만약 creator가 None 이면 역전파가 중단된다. creator가 없으므로 해당 V
```

8.2 반복문을 이용한 구현

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

8.3 동작 확인

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
        self.creator = None  # creator

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

    def backward(self):
        """
        자동 역전파 (반복)
        """
        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(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


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)


x = Variable(np.array(0.5))
A = Square()
B = Exp()
C = Square()

a = A(x)
b = B(a)
y = C(b)
```

```
## 자동 역전파
y.grad = np.array(1.0)
y.backward()
print(f"자동 역전파 : {x.grad}")
```

```
자동 역전파 : 3.297442541400256
```

In [ ]:

```
# Dezero ~ Pytorch
## Dezero
x = Variable(np.array(1.0))
A = Tanh()
B = Sigmoid()
a = A(x)
b = B(a)

b.grad = np.array(1.0)
b.backward()
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()
print(f"PyTorch : {x.grad}")
```

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