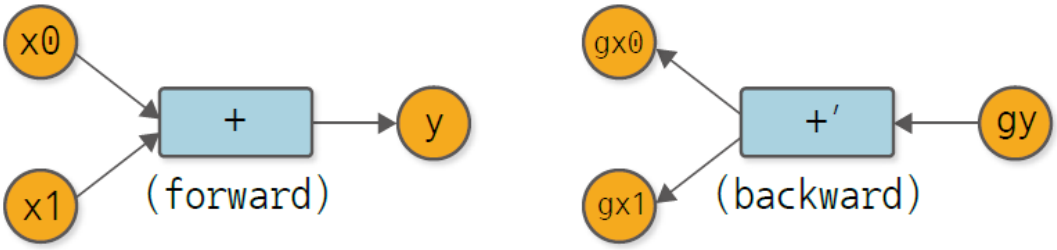
제2 고지 : 자연스러운 코드로

STEP 13 : 가변길이 인수 (역전파 편)

13.1 가변 길이 인수에 대응한 Add 클래스의 역전파

그림 13-1 덧셈 계산 그래프에서 순전파와 역전파(+′는 y = x0 + x1을 미분하는 함수)



Add 클래스를 살펴보면, 그림과 같이

- 순전파 : 입력이 2개, 출력이 1개
- 역전파 : 입력이 1개, 출력이 2개

예를들어, $f(\mathbf{x}) = x_0 + x_1$ 일때, $\nabla_{\mathbf{x}} f = \begin{bmatrix} \frac{\partial f}{\partial x_0} \\ \frac{\partial f}{\partial x_1} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

In [ ]:

```
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, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ######################################
        ys = self.forward(*xs) # 1. 리스트 언팩
        if not isinstance(ys,tuple): # 2. 튜플이 아닌 경우 추가 지원
            ys = (ys,)
        ######################################
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, x0,x1):
        y = x0 + x1
        return y
    def backward(self, gy):
        # 역전파시 , 입력이 1개 , 출력이 2개
        return gy,gy
```

13.2 Variable 클래스 수정

`Add` 클래스의 `backward()` 연산이 여러개의 출력값을 반환하므로, `Variable` 클래스의 `backward()` 를 수정해야한다.

1. 순전파의 결과가 **여러개의 출력인 경우**를 처리

2. 역전파 기준 **여러 개의 입력(=순전파의 여러 개 출력)** 을 처리.
3. 역전파 **결과값이 하나인 경우(=역전파의 출력이 1개인 경우)** 튜플로 변환.
4. **역전파 결과가 여러개의 출력인 경우** 각각 대응

In [ ]:

```
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()
            ########################################
            gys = [output.grad for output in f.outputs] # 1. 순전파의 결과가 **
            gxs = f.backward(*gys) # 2. 역전파 기준 **여러 개의 입력(=순전파의 여러 
            if not isinstance(gxs,tuple): # 3. 역전파 **결과값이 하나인 경우(=역전파
                gxs = (gxs,)
            for x,gx in zip(f.inputs,gxs): # 4. **역전파 결과가 여러개의 출력인 경우
                x.grad = gx
                if x.creator is not None:
                    funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다
            ########################################


class Function:
    """
    Function Base Class

    """

    def __call__(self, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ########################################
        ys = self.forward(*xs) # 1. 리스트 언팩
        if not isinstance(ys,tuple): # 2. 튜플이 아닌 경우 추가 지원
            ys = (ys,)
```

```
        ######################################
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, x0,x1):
        y = x0 + x1
        return y
    def backward(self, gy):
        # 역전파시 , 입력이 1개 , 출력이 2개
        return gy,gy
def add(x0,x1):
    return Add()(x0,x1)
```

13.3 Square 클래스 구현

STEP 11 에서 `Function` 클래스의 인스턴스 변수 `input -> inputs` 로 여러 입력을 받도록 수정이 되었으므로, 해당 부분을 고려해서 구현.

In [ ]:

```
class Square(Function):
    def forward(self, x):
        y= x**2
        return y
    def backward(self, gy):
        x = self.inputs[0].data  #  수정 전 : x= self.input.data
        gx = 2 * x * gy
        return gx

def square(x):
    return Square()(x)
```

$$z = x^2 + y^2$$

$$\Rightarrow \frac{dz}{d\mathbf{x}} = \nabla_{\mathbf{x}} f = \begin{bmatrix} \frac{\partial z}{\partial x_1} & \frac{\partial z}{\partial y_1} \end{bmatrix} = \begin{bmatrix} 2x & 2y \end{bmatrix}$$

$$\Rightarrow \left.\frac{dz}{d\mathbf{x}}\right|_{(2,3)} = \begin{bmatrix} 4 & 6 \end{bmatrix}$$

In [ ]:

```
import torch
# Dezero ~ PyTorch
## Dezero
x = Variable(np.array(2.0))
y= Variable(np.array(3.0))

z = add(square(x),square(y))
z.backward()
print(f"Dezero : z.data = {z.data}")
print(f"Dezero : x.grad = {x.grad}")
print(f"Dezero : y.grad = {y.grad} ")

print("="*50)
## PyTorch
x= torch.tensor([2.0],requires_grad=True)
y= torch.tensor([3.0],requires_grad=True)
z =x**2+y**2
z.backward()
print(f"PyTorch : z.data = {z.data}")
print(f"PyTorch : x.grad = {x.grad}")
print(f"PyTorch : y.grad = {y.grad}")
```

```
Dezero : z.data = 13.0
Dezero : x.grad = 4.0
Dezero : y.grad = 6.0
==================================================
PyTorch : z.data = tensor([13.])
PyTorch : x.grad = tensor([4.])
PyTorch : y.grad = tensor([6.])
```

코드

In [ ]:

```
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()
            ########################################
            gys = [output.grad for output in f.outputs] # 1. 순전파의 결과가 **
            gxs = f.backward(*gys) # 2. 역전파 기준 **여러 개의 입력(=순전파의 여러 
            if not isinstance(gxs,tuple): # 3. 역전파 **결과값이 하나인 경우(=역전파
                gxs = (gxs,)
            for x,gx in zip(f.inputs,gxs): # 4. **역전파 결과가 여러개의 출력인 경우
                x.grad = gx
                if x.creator is not None:
                    funcs.append(x.creator)  # 하나 앞의 함수를 리스트에 추가한다
            ########################################


class Function:
    """
    Function Base Class

    """

    def __call__(self, *inputs):  #  1. * 를  활용하여 임의 개수의 인수
        xs = [x.data for x in inputs]
        ########################################
        ys = self.forward(*xs) # 1. 리스트 언팩
        if not isinstance(ys,tuple): # 2. 튜플이 아닌 경우 추가 지원
            ys = (ys,)
        ########################################
        outputs = [Variable(as_array(y)) for y in ys]

        for output in outputs:
            output.set_creator(self)
        self.inputs = inputs
        self.outputs = outputs

        # 2. 리스트의 원소가  하나라면  첫번째 원소를 반환
        return outputs if len(outputs) > 1 else outputs[0]

    def forward(self, xs):
        """
        구체적인 함수 계산 담당
        """
        raise NotImplementedError()

    def backward(self, gys):
        """
        역전파
        """
        raise NotImplementedError()


class Add(Function):
    def forward(self, x0,x1):
        y = x0 + x1
        return y
    def backward(self, gy):
        # 역전파시 , 입력이 1개 , 출력이 2개
        return gy,gy

def add(x0,x1):
    return Add()(x0,x1)

class Square(Function):
```

```
    def forward(self, x):
        y= x**2
        return y
    def backward(self, gy):
        x = self.inputs[0].data  #  수정 전 : x= self.input.data
        gx = 2 * x * gy
        return gx

def square(x):
    return Square()(x)
```