

STEP 1. 상자글서의 변수

```
class Variable:
```

```
    def __init__(self, data):
```

```
        self.data = data
```

← 주어진 인수를 인스턴스 변수 data에 대입.

```
import numpy as np
```

```
data = np.array(1.0)
```

```
x = Variable(data)
```

```
print(x.data)
```

⇒ 1.0

Variable 인스턴스

실제 데이터를 x에 담겨있음.

STEP 2. 변수를 넣는 함수

- Variable 인스턴스를 변수로 다룰 수 있는 함수 구현

Class Function:

```
def *__call__ (self, input):  
    x = input.data    # 데이터를 꺼냄.  
    y = x ** 2        # 실제 계산  
    output = Variable(y)  # Variable 객체는 되돌림.  
    return output
```

```
x = Variable (np.array (10))
```

```
f = Function ()
```

```
y = f(x)
```

```
print (type(y))    # type() 함수: 객체의 클래스를 알려줌.
```

```
print (y.data)
```

↓
y의 클래스는 Variable이며,
데이터는 y.data에 저장되어 있음.

→ Function 클래스 수정

RULE

- Function 클래스는 기법클래스, 모든 함수에 공통되는 기능 구현
- 구체적인 함수는 Function 클래스를 상속한 클래스에서 구현

Class Function:

```
def __call__(self, input):  
    x = input.data  
    y = self.forward(x)    # 구체적인 계산은 forward 메서드에서  
    output = Variable(y)  # Variable 형태로 되돌림  
    return output  
  
def forward(self, x):  
    raise NotImplementedError() → forward 메서드의 구체적인 논리는  
                                하위 클래스에서 구현
```

↓ Function 클래스를 상속하여
입력값을 제공하는 클래스 구현

```
class Square(Function):  
    def forward(self, x):  
        return x ** 2
```

```
x = Variable(np.array(10))
```

```
f = Square()
```

```
y = f(x)
```

```
print(type(y))
```

```
print(y.data)
```

STEP 3. 함수 연결

```
class Exp (Function):  
    def forward (self, x):  
        return np.exp (x)
```

Function 클래스의 `-- call --` 메서드는 입력과 출력이 모두 Variable 인스턴스이므로
DeZero 함수들을 연이어 사용 가능.

Example :

$$y = (e^{x^2})^2$$



```
A = Square ()
```

```
B = Exp ()
```

```
C = Square ()
```

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

```
a = A(x)
```

```
b = B(a)
```

```
y = C(b)
```

```
print (y.data)
```

STEP 4. 수치미분

$f(x)$ 라는 함수에 대한 미분

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

극한 \rightarrow $y=f(x)$ 그래프의 두 점을
지나는 직선의 기울기

수치미분이란?

- 미세한 차이 ($0.0001 = 1e-4$)를 이용해 함수의 변화량을 구하는 방법.
- 작은값을 이용하여 '진정한 미분'을 근사
- 근사오차를 줄이는 방법? '중앙차분 (centered difference)'
 - 직선의 기울기: $\frac{f(x+h) - f(x-h)}{2h}$

numerical_diff (f, x, eps = 1e-4)

- f: 미분의 대상이 되는 함수 (Function 의 인스턴스)
- x: 미분을 계산하는 변수 (Variable 인스턴스)
- eps (= epsilon) : 작은값

```
def numerical_diff (f, x, eps = 1e-4):
```

```
    x0 = Variable (x.data - eps)
```

```
    x1 = Variable (x.data + eps)
```

```
    y0 = f(x0)
```

```
    y1 = f(x1)
```

```
    return (y1.data - y0.data) / (2 * eps)
```

↓ Square 클래스를 대상으로 미분

```
f = Square()
```

```
x = Variable (np.array (2.0))
```

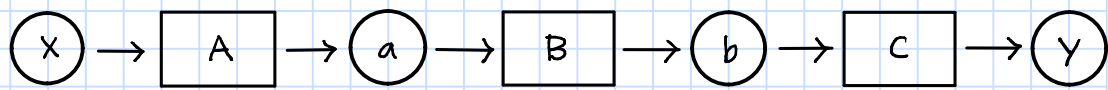
```
dy = numerical_diff (f, x)
```

```
print (dy)
```

STEP 5. 역전파 이론

- 역전파를 통해 미분을 효율적으로 계산 & 결과값 모으는 더 좋음.

- 함수값 $y = F(x)$

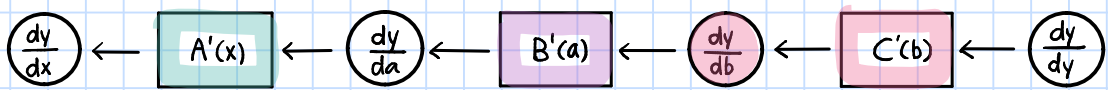


• $a = A(x)$, $b = B(a)$, $y = C(b)$

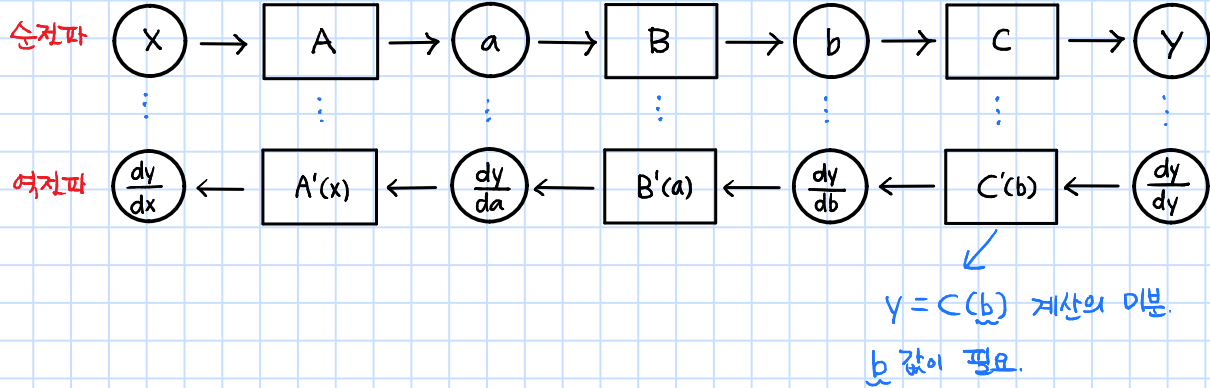
• x 에 대한 y 의 미분 $\frac{dy}{dx} = \frac{dy}{db} \frac{db}{da} \frac{da}{dx} \rightarrow \frac{dy}{dy} \frac{dy}{db} \frac{db}{da} \frac{da}{dx}$

← 역방향 (출력 → 입력)으로 계산

• $\frac{dy}{dx} = \left(\left(\frac{dy}{dy} \frac{dy}{db} \right) \frac{db}{da} \right) \frac{da}{dx}$
 $\quad \quad \quad \text{C'(b)} \quad \text{B'(a)} \quad \text{A'(x)}$



- 변수 y, b, a, x 에 대한 미분값이 오른쪽 → 왼쪽 전파. (역전파)
- 전파되는 데이터가 모두 y 의 미분값!
- 순실행수의 각 매개변수에 대한 미분을 계산
 : 출력 → 입력 방향은 전파하면 한 번의 전파만으로 모든 매개변수에 대한 미분을 계산할 수 있음.



- 역전파에는 순전파 시 이용한 데이터가 필요.
- 역전파를 구현하려면 먼저 순전파를 하고, 각 함수가 입력변수 (ex. x, a, b)의 값을 기억해두어야 함

STEP 6. 수동 역전파

6.1. Variable 클래스 추가 구현

class Variable :

def __init__(self, data):

self.data = data

self.grad = None # 미분값, None 초기화 (실제 역전파 시 미분값 계산하여 대입)

6.2 Function 클래스 추가 구현

- 미분을 계산하는 역전파 (backward 메서드)
- forward 호출 시 건네받은 Variable 인스턴스 유지

class Function :

def __call__(self, input):

x = input.data

y = self.forward(x)

output = Variable(y)

self.input = input # 입력변수를 기억 (보관)

return output

def forward(self, x):

raise NotImplementedError()

def backward(self, gy):

raise NotImplementedError()

6.3. Square 와 Exp 클래스 추가 구현

```
class Square(Function):
```

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

```
        y = x ** 2
```

```
        return y
```

```
    def backward(self, gy):
```

```
        x = self.input.data
```

```
        gx =  $\frac{2 * x}{y = x^2 \text{의 미분}} * \underset{\substack{\text{ndarray 인스턴스} \\ \text{(출력 쪽에서 전파되는 미분값을 전달하는 역할)}}}{gy} \rightarrow y = x^2 \text{의 미분 } \frac{dy}{dx} = 2x$ 
```

```
        return gx
```

```
class Exp(Function):
```

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

```
        y = np.exp(x)
```

```
        return y
```

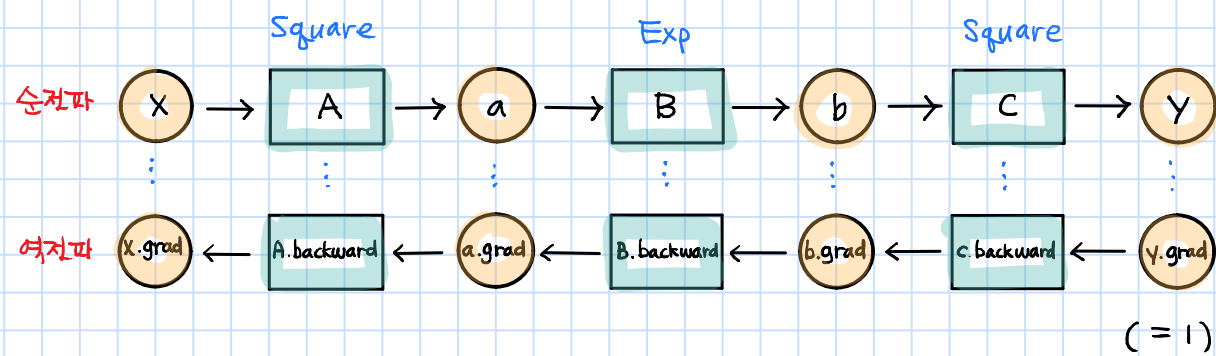
```
    def backward(self, gy):
```

```
        x = self.input.data
```

```
        gx = np.exp(x) * gy  $\rightarrow y = e^x \text{의 미분 } \frac{dy}{dx} = e^x$ 
```

```
        return gx
```


6.4 역전파 구현



순전파 코드

```
A = Square()
```

```
B = Exp()
```

```
C = Square()
```

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

```
a = A(X)
```

```
b = B(a)
```

```
y = C(b)
```

역전파 구현

```
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)
```

역전파는 $\frac{dy}{dy} = 1$ 에서 시작.

C → B → A 순으로 backward 메서드 호출.

```
class Variable :
```

```
    def __init__(self, data):
```

```
        self.data = data
```

```
        self.grad = None    # 미분값 , None 초기화 ( 실제 역전파 시 미분값 계산하여 대입 )
```

class Function:

```
def __call__(self, input):
```

```
    x = input.data
```

```
    y = self.forward(x)
```

```
    output = Variable(y)
```

```
    self.input = input    # 입력변수를 기억(보관)
```

```
    return output
```

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

```
    raise NotImplementedError()
```

```
def backward(self, gy):
```

```
    raise NotImplementedError()
```

```
class Square(Function):
```

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

```
        y = x ** 2
```

```
        return y
```

```
    def backward(self, gy):
```

```
        x = self.input.data
```

```
        gx =  $\frac{2 * x}{y=x^2 \text{의 미분}}$  *  $\frac{dy}{dx} = 2x$  * gy
```

→ $y = x^2$ 의 미분 $\frac{dy}{dx} = 2x$
ndarray 인스턴스
(출력 쪽에서 전제되는 미분값을 전달하는 역할)

```
        return gx
```

```
class Exp(Function):
```

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

```
        y = np.exp(x)
```

```
        return y
```

```
    def backward(self, gy):
```

```
        x = self.input.data
```

```
        gx = np.exp(x) * gy →  $y = e^x$  의 미분  $\frac{dy}{dx} = e^x$ 
```

```
        return gx
```

* 1: `--call--`

- 파이썬 특수 메소드
- `f = Function()` 형태의 함수의 인스턴스를 변수 `f`에 대입해두고,
나중에 `f(...)` 형태로 `--call--` 메서드 호출 가능

