제1 고지 : 미분 자동 계산

STEP 9 : 함수를 더 편리하게

9.1 : 파이썬 함수로 이용하기

- 인스턴스 생성 후 계산을 하는 과정을 간소화 하기 위해 **함수 정의를 지원**

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
```

In [ ]:

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

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

In [ ]:

```
x = Variable(np.array(0.5))
a = square(x)
b = exp(a)
y = square(b)

## 자동 역전파
y.grad = np.array(1.0)
y.backward()
print(x.grad)
```

```
3.297442541400256
```

9.2 bacward 메서드 간소화

- 역전파 계산시 `y.grad = np.array(1.0)` 를 선언하는 번거로움을 줄임

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)
        """
        ##########################
        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.
```

In [ ]:

```
x = Variable(np.array(0.5))
a = square(x)
b = exp(a)
y = square(b)

## 자동 역전파
y.backward()
print(x.grad)
```

```
3.297442541400256
```

9.3 ndarray만 취급하기

- `Variable` 클래스는 `np.ndarray` 타입만 데이터를 받도록 정의
- 주의해야할것은 0차원 `np.ndarray` 인스턴스를 사용하여 계산하면 결과 데이터 타입이 `np.float32` , `np.float64` 등 으로 달라지는 것을 방지하기 위해 `as_array()` 지원

In [ ]:

```
import numpy as np
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.
```

In [ ]:

```
def as_array(x):

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

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)) # ndarray 인스턴스 선언
        ##########################################
        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()
```

코드

In [ ]:

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

##########################################
# 9.3 ndarray만 취급하기
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:
        ##########################################
        # 9.3 ndarray만 취급하기
        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)
        """
        ##########################################
        # 9.2 bacward 메서드 간소화
        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
        ##########################################
        # 9.3 ndarray만 취급하기
        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

##########################################
# 9.1 파이썬 함수로 이용하기
def square(x):
    f = Square()
    return f(x)
##########################################

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

##########################################
# 9.1 파이썬 함수로 이용하기
def exp(x):
    f = Exp()
    return f(x)
##########################################


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))

##############################################
# 9.1 파이썬 함수로 이용하기
def sigmoid(x):
    f = Sigmoid()
    return f(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)


##############################################
# 9.1 파이썬 함수로 이용하기
def tanh(x):
    f = Tanh()
    return f(x)
##############################################

x = Variable(np.array(0.5))
a = square(x)
b = exp(a)
y = square(b)

## 자동 역전파
y.backward()
print(x.grad)
```

```
3.297442541400256
```

In [ ]:

```
# Dezero ~ Pytorch
## Dezero
x = Variable(np.array(1.0))
a = tanh(x)
b = sigmoid(a)

b.backward()
print(f"Dezero : {x.grad}")

## Pytorch
x = torch.tensor([1.0], requires_grad=True)
a = F.tanh(x)
b = F.sigmoid(a)
b.backward()  #  NOTE : step07 에서 Dezero Variable 클래스에서 해당 기능 구현
print(f"PyTorch : {x.grad}")
```

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