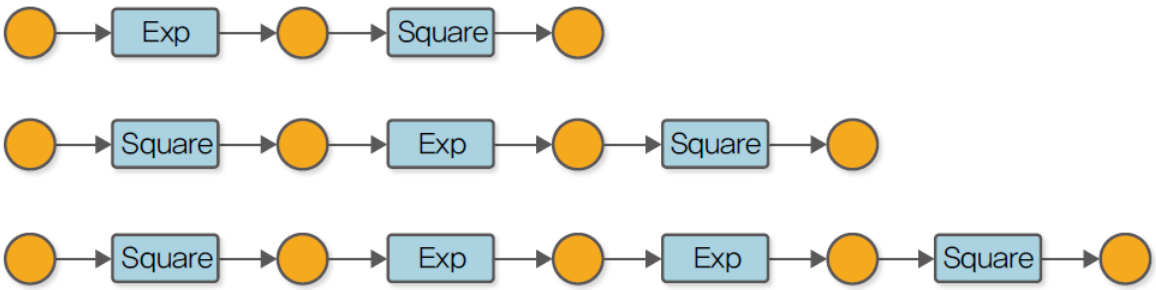
제1 고지 : 미분 자동 계산

STEP 7 : 역전파 자동화

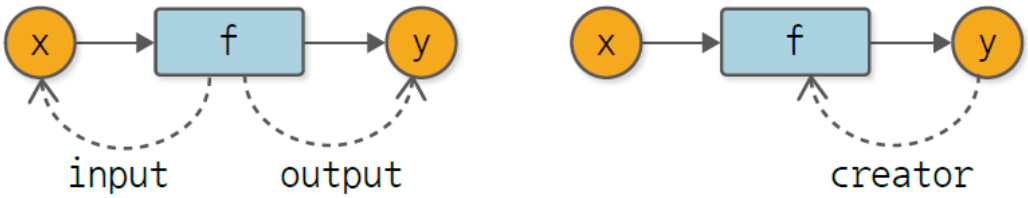
그림 7-1 다양한 계산 그래프(변수명은 생략하고 함수명은 클래스 이름으로 대신함)



- 이 전에는 역전파 계산 코드를 수동으로 조합해야 했다. 이 과정은 번거롭기 때문에 자동화 하려 한다.
- 즉, 순전파 계산을 한 번 진행하면 어떤 계산이라도 상관없이 역전파가 이루어지는 구조를 만든다.
- **Define-by-Run :** 딥러닝 에서 수행하는 계산들을 **계산 시점에 연결하는 방식**으로 **동적 계산 그래프**

7.1 역전파 자동화의 시작

그림 7-2 함수 입장에서 본 변수와의 관계(왼쪽)와 변수 입장에서 본 함수와의 관계(오른쪽)



- 변수와 함수의 관계를 이해하는 것이 중요
  - 함수 입장 에서 바라본 변수 : 입력 과 출력
  - 변수 입장 에서 바라본 함수 : 변수를 만들어 내는 creator

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
```

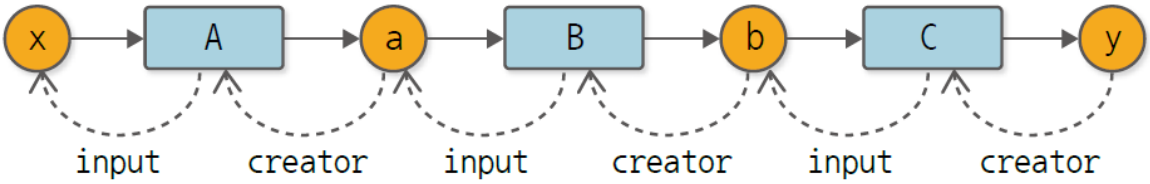
In [ ]:

```
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
```

```
        ################################
        output = Variable(y)
        output.set_creator(self)  # 출력 변수에 creator 설정 ( 연결을 동적으로 만드는
        self.output = output  # 출력도 저장
        ################################
        return output
```

그림 7-3 계산 그래프 역추적(y에서 시작)



In [ ]:

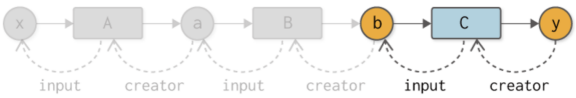
```
x = Variable(np.array(0.5))
A = Square()
B = Exp()
C = Square()

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

# 함수 ~ 변수 연결 테스트
assert y.creator == C
assert y.creator.input == b
assert y.creator.input.creator == B
assert y.creator.input.creator.input == a
assert y.creator.input.creator.input.creator == A
assert y.creator.input.creator.input.creator.input == x
```

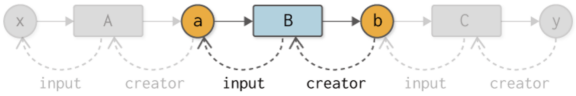
7.2 역전파 도전

- 변수와 함수의 관계를 이용하여 역전파를 계산하는 과정은 크게 3가지로 이루어진다.
  1. 함수를 가져온다
  2. 함수의 입력을 가져온다
  3. 함수의 `backward()` 를 호출한다



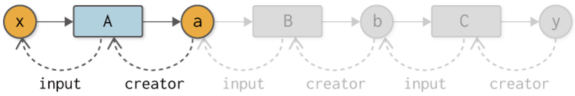
In [ ]:

```
## b.grad
y.grad = np.array(1.0)
C = y.creator  ## 1. 함수를 가져온다
b = C.input  ## 2. 함수의 입력을 가져온다
b.grad = C.backward(y.grad)  ## 3. 역전파를 계산한다
```



In [ ]:

```
## a.grad
B = b.creator  ## 1. 함수를 가져온다
a = B.input  ## 2. 함수의 입력을 가져온다
a.grad = B.backward(b.grad)  ## 3. 역전파를 계산한다
```



In [ ]:
```
## x.grad
A = a.creator  ## 1. 함수를 가져온다
x = A.input  ## 2. 함수의 입력을 가져온다
x.grad = A.backward(a.grad)  ## 3. 역전파를 계산한다
```

7.3 backward 메서드 추가

- 위의 똑같은 처리 흐름을 자동화 하기 위해 `Variable` 클래스에 `backward()` 추가

In [ ]:
```
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
        """
        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
```

In [ ]:
```
# 자동화된 역전파
x = Variable(np.array(0.5))
A = Square()
B = Exp()
C = Square()

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

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

```
3.297442541400256
```

코드

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
        """
        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 va


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)


# Dezero

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

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

# 함수 ~ 변수 연결 테스트
assert y.creator == C
assert y.creator.input == b
assert y.creator.input.creator == B
assert y.creator.input.creator.input == a
assert y.creator.input.creator.input.creator == A
```

```
assert y.creator.input.creator.input.creator.input == x


# 함수 ~ 변수 연결을 통한 역전파
## 수동 역전파
## 1. 함수를 가져온다
## 2. 함수의 입력을 가져온다
## 3. 역전파를 계산한다

## b.grad
y.grad = np.array(1.0)
C = y.creator  ## 1. 함수를 가져온다
b = C.input  ## 2. 함수의 입력을 가져온다
b.grad = C.backward(y.grad)  ## 3. 역전파를 계산한다


## a.grad
B = b.creator  ## 1. 함수를 가져온다
a = B.input  ## 2. 함수의 입력을 가져온다
a.grad = B.backward(b.grad)  ## 3. 역전파를 계산한다

## x.grad
A = a.creator  ## 1. 함수를 가져온다
x = A.input  ## 2. 함수의 입력을 가져온다
x.grad = A.backward(a.grad)  ## 3. 역전파를 계산한다

print(f"수동 역전파 : {x.grad}")


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

```
수동 역전파 : 3.297442541400256
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])
```