CH.01

학습: 훈련 데이터로부터 가중치 매개변수의 최적값을 자동으로 획득하는 것

- 베이즈 정리

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

- 두 확률 변수의 사전 확률과 사후 확률 사이의 관계를 나타내는 정리
- p(A) : A의 사전 확률(Prior Probability), 사건 B가 발생하기 전 가지고 있던 사건 A의 확률
- p(A|B) : A의 사후 확률(Posterior Probability), 사건 B가 발생한 후 갱신된 사건 A의 확률
- p(B|A) : 가능도, 사건 A가 발생한 경우 사건 B의 확률
- p(B) : B의 사전 확률, 정규화 상수 or 증거

- 평균 제곱 오차(MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{Y}_i - Y_i)^2$$

- 손실 함수, 현재 신경망이 훈련 데이터를 얼마나 잘 처리하지 못하는가를 나타냄
- 최적의 매개변수 값을 탐색하는 지표

- 미분(변화율)

$$\frac{d}{dx}f(x) = \lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

- 특정 순간의 변화량, 가중치 매개변수를 시간에 따라 변화시킴

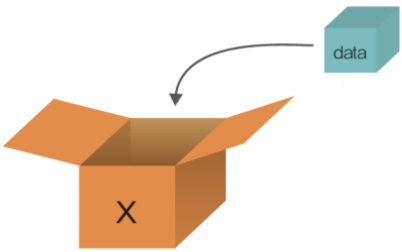
- 선형대수학

$$\begin{aligned} x_1 + x_2 + x_3 &= 3 \\ x_1 - x_2 + 2x_3 &= 2 \\ 2x_1 \quad + 3x_3 &= 5 \end{aligned} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -1 & 2 \\ 2 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \\ 5 \end{bmatrix}$$

- 행렬,벡터
- 가중치 매개변수

- 변수 : 어떤 관계나 범위 안에서 여러 가지 값으로 임의로 변할 수 있는 수

- 변수와 클래스 관계



X: 클래스 , data: 변수

- __init__ (self,): 초기화를 위한 함수, 생성자

  - self : 클래스 그 자체를 의미, 클래스의 메소드에는 첫 인자는 반드시 self로 하는 것이 좋다. Why? 메소드 호출 시 항상 파이썬 자체에서 인스턴스의 주소 값을 넘겨주기 때문이다. 따라서 self로 주소 값을 받아주는 역할을 해줘야한다.

```
class Variable:
    #객체 생성을 위한 초기화 함수
    def __init__(self,data):
        self.data = data
```

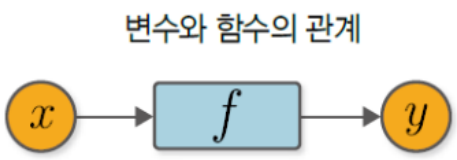
```
import numpy as np

data = np.array(1.0)
#x는 Variable 클래스의 인스턴스
x = Variable(data)
print(x.data)
```

- 넘파이의 다차원 배열

  - numpy.ndarry
  - 다차원 배열
    - 숫자 등의 원소가 일정하게 모여 있는 데이터 구조

- 함수: 변수 사이의 대응 관계를 정하는 역할

- 계산 그래프



  - 노드(node)와 에지(edge)로 구성된 데이터 구조

- __call__ : 인스턴스가 호출됐을 때 실행

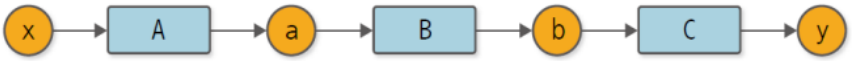
```
class Function:
    #Function클래스의 인스턴스를 호출 시 실행
    def __call__(self, input):
        x = input.data
        y = x**2 |
        output = Variable(y)
        return output
        print("__call__ 실행")
```

→ Function클래스의 인스턴스를 생성하면 빈 인스턴스가 생성 But __call__ 메소드를 호출하기 위한 목적으로 빈 인스턴스를 생성한다.

- 클래스 상속:

```
#부모 클래스
class Function:
    #Function클래스의 인스턴스를 호출 시 실행
    def __call__(self, input):
        x = input.data
        # y = x**2
        y = self.forward(x)
        output = Variable(y)
        return output
    def forward(self,in_data):
        raise NotImplementedError()

#Function 클래스를 상속
#Function 클래스의 자식 클래스
class Square(Function):
        #Function의 forward 메소드를 오버라이딩
        #재정의
        def forward(self, x):
            return x**2
```

- 합성 함수: 여러 함수로 구성된 함수

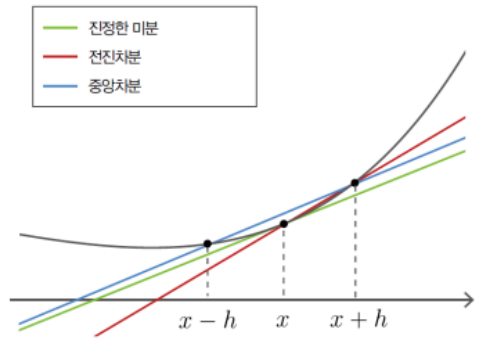


- 수치 미분: 함수의 미분 값을 근사적으로 계산하는 방법, 함수를 입력 값 주변에서 작은 변화량을 적용하여 해당 점에서의 미분 값을 추정함

- 역전파: 수치 미분을 대신하는 더 효율적인 알고리즘

중앙 차분

그림 4-2 진정한 미분, 전진차분, 중앙차분 비교




```
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)
```

```
f = Square()
x = Variable(np.array(2.0))
dy = numerical_diff(f, x)
print(dy)
```

- 중앙차분 > 전진차분

- 역전파(오차역전파법)

  - 하나의 학습 데이터에 대한 비용함수의 기울기를 계산
  - 비용함수는 실제 값(정답)과 모델이 예측한 값 간의 차이

- 연쇄 법칙: 여러 함수를 사슬처럼 연결하여 사용

- 역전파 원리 도출

  - 손실함수: 모델의 예측 값과 실제 관측된 값 사이의 차이
    - 손실 함수: 전체 데이터셋 <-> 비용함수: 각각의 개별적인 데이터 포인트

그림 5-2 출력 쪽의 미분부터 순서대로 계산

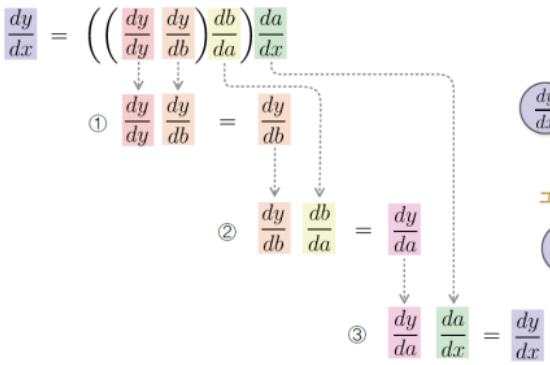


그림 5-3 $\frac{dy}{dx}$를 구하는 계산 그래프

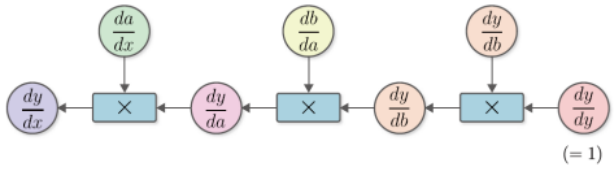
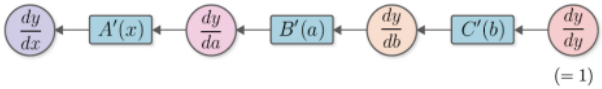
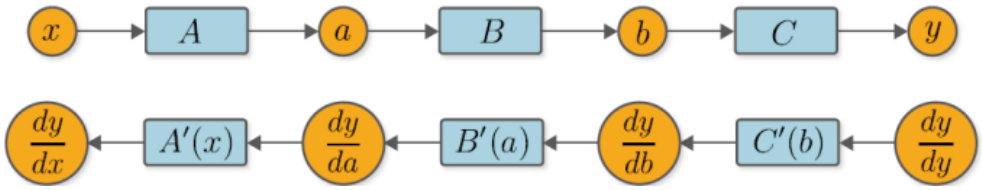


그림 5-4 단순화한 역전파 계산 그래프($A'(x)$의 곱셈을 '$A'(x)$'라는 노드로 간략하게 표현)



계산 그래프(역전파를 구하려면 먼저 순전파를 하고, 이때 각 함수가 입력 변수의 값을 기억)

그림 5-5 순전파(위)와 역전파(아래)



순전파: 변수는 '통상 값', 역전파: 변수는 '미분 값',

Ch.02

- 역전파

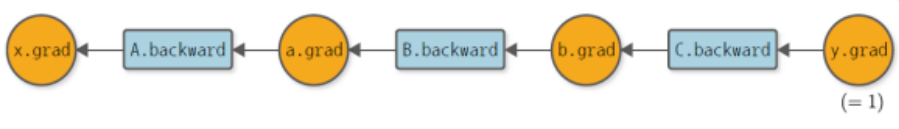
```
class Variable:
    def __init__(self,data):
        self.data = data
        #미분값을 대입할 인스턴스 변수
        self.grad = None

class Function:
    def __call__(self, input):
        x = input.data
        y = self.forward(x)
        output = Variable(y)
        #입력 변수를 기억하기 위함
        self.input = input
        return output

    def forward(self,x):
        raise NotImplementedError()

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

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

        def backward(self, gy):
            x = self.input.data
            gx = 2 * x + gy
            return gx
```

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

    def backward(self,gy):
        x = self.input.data
        gx = np.exp(x) + gy
        return gx
```

```
A = Square()
B = Exp()
C = Square()

x = Variable(np.array(0.5))
a = A(x)
b = B(a)
y = C(b)

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)
```

- gy는 미분 값 (질문)
- x = self.input.data는 역전파 과정에서 해당 함수의 입력 값에 대한 정보를 사용하여 미분 값을 계산하기 위함 (질문)

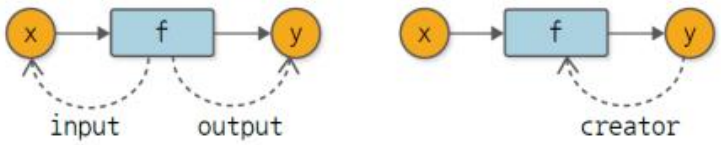


(역전파 계산 그래프)

- 역전파 자동화

  - Define-by-Run: 동적 계산 그래프, 딥러닝에서 수행하는 계산들을 계산 시점에 '연결'하는 방식

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



➔ 함수 입장: 변수는 '입력', '출력'

➔ 변수 입장: 함수는 '창조자', '부모'

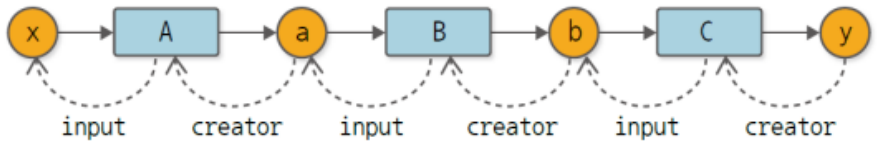
+ assert: 계산 그래프를 거꾸로 거슬러 올라가며 조건 충족 여부를 확인

```
assert y.creator == C
assert y.creator.input == b
```

Linked list 데이터 구조

➔ 노드들의 연결로 이루어진 데이터 구조

그림 7-3 계산 그래프 역추적(y에서 시작)



backward '메서드의 재귀적 호출' → '반복문을 이용한 구현'

```
class Variable:
    def __init__(self,data):
        self.data = data
        #미분값을 대입할 인스턴스 변수
        self.grad = None
        self.creator = None

    def set_creator(self,func):
        self.creator = func

    def backward(self):
        f = self.creator
        if f is not None:
            x = f.input
            x.grad = f.backward(self.grad)
            x.backward()
```

```
class Variable:
    def __init__(self,data):
        self.data = data
        #미분값을 대입할 인스턴스 변수
        self.grad = None
        self.creator = None

    def set_creator(self,func):
        self.creator = func

    def backward(self):
        funcs = [self.creator]
        while funcs:
            f = funcs.pop()
            x, y = f.input, f.output
            x.grad = f.backward(y.grad)
            if x.creator is not None:
                funcs.append(x.creator)
```