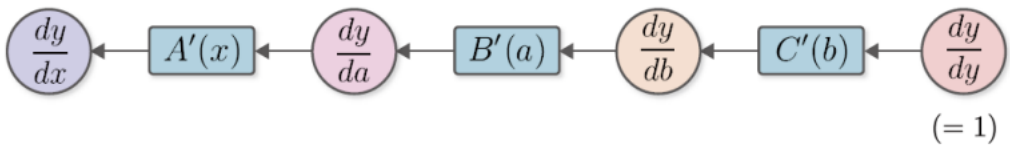
정리 노트(2주차)

산업데이터사이언스학부
201904213
심성빈

역전파

-합성 함수의 미분은 구성 함수들의 미분의 곱으로 분해할 수 있음
-어떤 순서로 곱해도 상관 X
-보통의 계산과는 반대 방향으로 미분을 계산

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



-변수 y, b, a, x에 대한 미분값이 오른쪽에서 왼쪽으로 전파되는데 이것이 역전파

머신러닝은 주로 대량의 매개변수를 입력 받아서 마지막에 손실 함수를 거쳐 출력을 내는 형태로 진행

순전파 계산 그래프

-통상적인 계산 / 변수는 통상값 존재

역전차 계산 그래프

-변수는 통상값과 미분값이 존재하고 함수는 통상 계산(순전파)과 미분값을 구하기 위한 계산(역전파)이 존재하는 것으로 생각 할 수 있음
-> 역전파를 어떻게 구현할지 짐작 가능
-역전파시는 순전파사 이용한 데이터가 필요
-역전파를 구하려면 먼저 순전파를 하고, 이때 각 함수가 입력 변수의 값을 기억해야함

수동 역전파

-Variable, Function클래스를 확장하여 구현

Variable

-통상값(data)과 더불어 그에 대응하는 미분값(gard)도 저장하도록 확장

-gard, data는 넘파이의 다차원 배열(ndarray)로 가정

-gard는 역전파 수행시 미분값을 계산하여 대입

(gard -> 기울기 / 벡터나 행렬 등 다변수에 대한 미분)

Function

-기존의 Function은 순전파(forward)기능만 지원

-> 미분을 계산하는 역전파 backward 메서드

-함수에 입력한 변수(Varialbe 인스턴스)가 필요할때 self.input에서 가져와 사용

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

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

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

Square, Exp

-Square는 제곱을 계산하는 클래스

-Exp는 y=e의 x을 계산을 하는 클래스

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

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

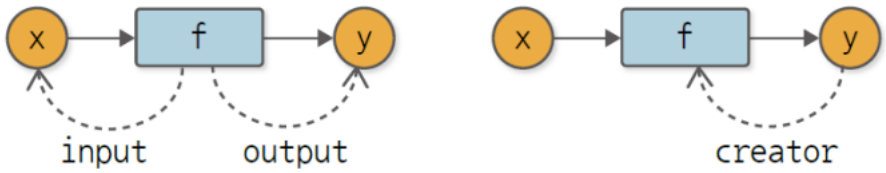
역전파 자동화

-일반적인 계산을 한번만 해주면 어떤 계산이라도 상관없이 역전파가 자동으로 이루어지는 구조를 만드는것

(Define -by- Run :동적 계산 그래프 / 딥러닝에서 수행하는 계산들을 계산 시점에 연결하는 방식)

역전파 자동화의 시작

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



-변수는 입력과 출력

-Variable클래스에 creator인스턴스 변수 추가, set_creator() 메서드 추가

-연결된 Variable과 Function으로 계산 그래프를 거꾸로 거슬러 올라갈 수 있음--

(- assert문으로 조건을 충족하느지 여부 확인

-계산 그래프는 함수와 변수 사이의 연결로 구성

-연결이 실제로 계산을 수행하는 시점에 만들어짐

->Define -by-Run 특징)

-Linked list데이터 구조

(- 노드들의 연결로 이루어진 데이터 구조)

반복작업 자동화

-Variable클래스에 backwrad라는 새로운 메서드 추가하여 반복작업을 자동화

Backward 메서드의 재귀적 호출

-자신보다 하나 앞에 놓인 변수의 backward메소드 호출

```
class Variable:
    def __init__(self, data):
        self.data = data
        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()
```

재귀를 사용한 구현을 반복문을 이용한 구현으로 수정

-반복문 방식이 더 효율적

-> 재귀는 함수를 재귀적으로 호풀할 때마다 중간 결과를 메모리에 유지하면서 처리

-재귀적 호출할때와 비슷하게 하지만 backward부분에서 while문을 사용하고

funcs.pop()을 호출하여 처리할 함수를 꺼냄

```
class Variable:
    def __init__(self, data):
        self.data = data
        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)
```

테스트 프로그램의 중요성

-버그 예방 / 테스트를 자동화해야 소프트웨어 품질을 유지

단위 테스트

-가능한 가장 작은 소프트웨어를 실행하여 예상대로 동작하는지 확인하는 테스트

-대상 단위의 크기를 작게 설정해서 최대한 간단하게 작성

-화이트 박스 테스트

-단위 테스트는 TDD(Test Driven Develop)와 함께 할 때 더 강력

통합 테스트

-여러 모듈을 모아 개발된 의도대로 동작되는지 확인하는 테스트

-단위 테스트에서 발견하기 어려운 버그를 찾을 수 있는 장점

- 신뢰성이 떨어질 수 있는 점과 어디서 에러가 발생했는지 확인하기 쉽지X

생각해본 사항

-제가 평소에 하던 내용과 다른 내용이라 비슷하면서 다른 느낌이 들어 처음에 이해하기 힘들었으나 정리 노트를 쓰면서 정리가 되어 좋았습니다.