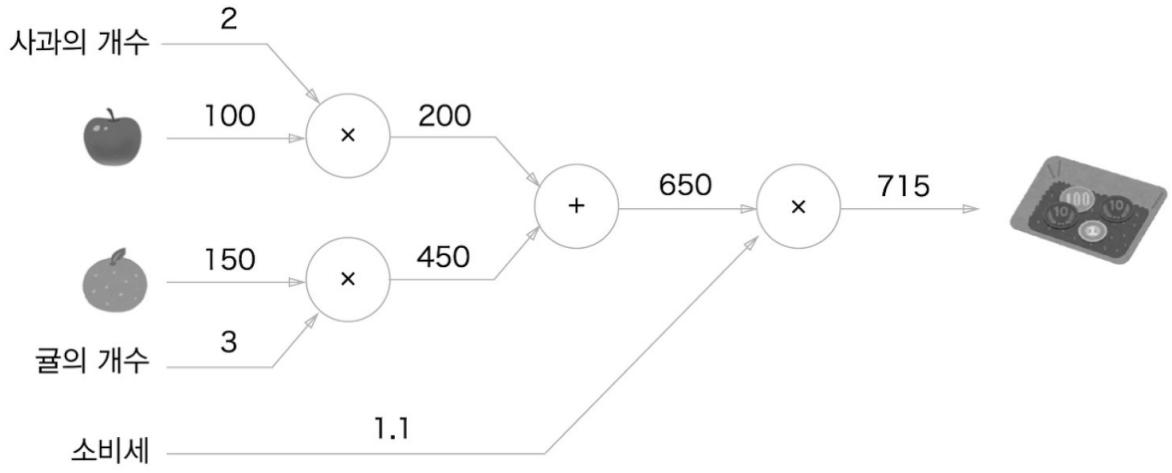
오차 역전파

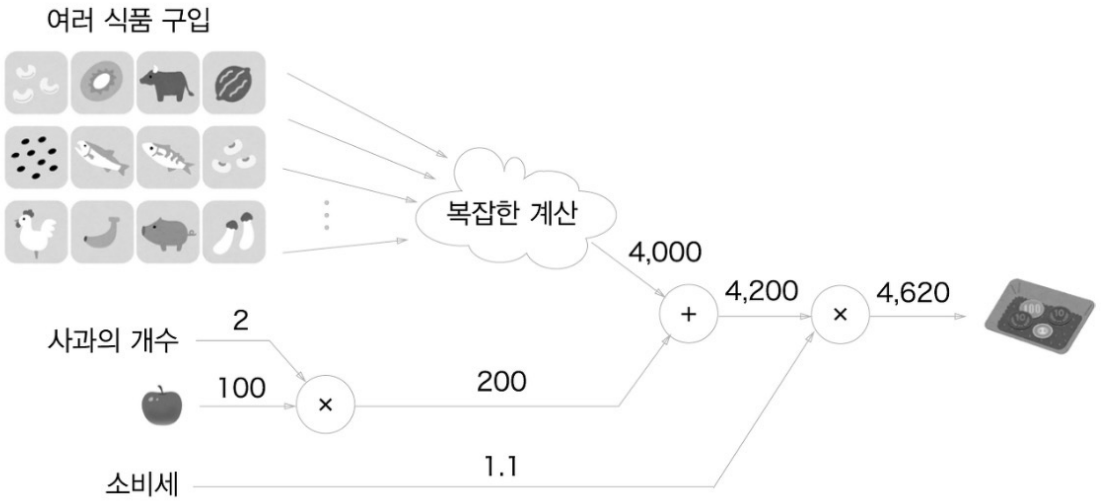
오차역전파

계산 그래프

- 계산 과정을 그래프로 나타낸 것
- 그래프는 복수의 노드와 에지로 표현된다

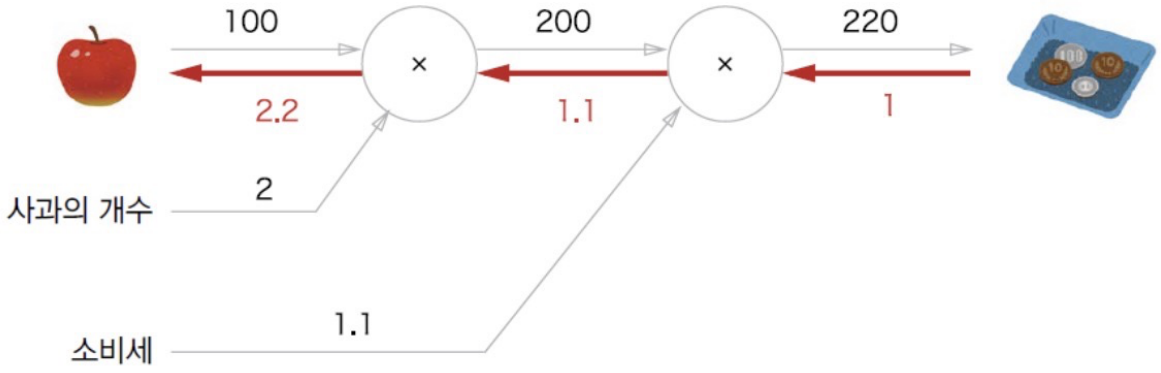


- 계산 그래프는 국소적 계산을 통해 최종 결과를 얻는다.
- 국소적 계산 = 전체에서 어떤 일이 벌어지든 상관없이 자신과 관계된 정보만으로 결과를 출력
- 전체 계산이 아무리 복잡하더라도 단순하게 처리 가능



왜 이걸 써야되지?

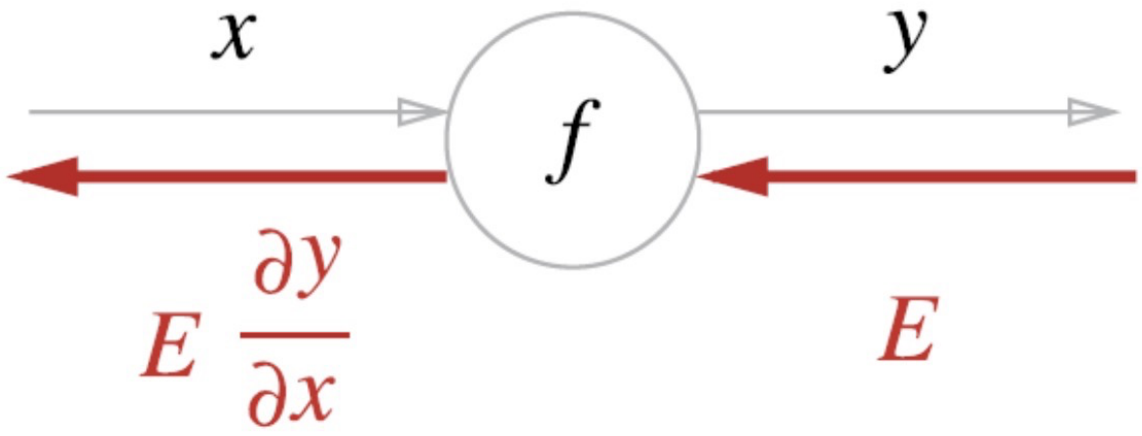
- 복잡한 계산도 단순처리 가능
- 중간 계산 결과 모두 보관
- 역전파를 통해 미분을 효율적으로 계산



- 역전파는 순전파와 반대 방향의 굵은 화살표 그림
- 사과가 1원 오르면 최종 금액은 2.2원 오른다는 의미

연쇄법칙

- 국소적 미분 = 순전파 때 y=f(x) 계산의 미분을 구하는 것
- 역전파의 계산은 신호 E에 국소적 미분($\frac{\partial y}{\partial x}$)을 곱한 후 다음 노드로 전달함
- 국소적인 미분을 상류에서 전달된 값에 곱해 앞쪽 노드로 전달

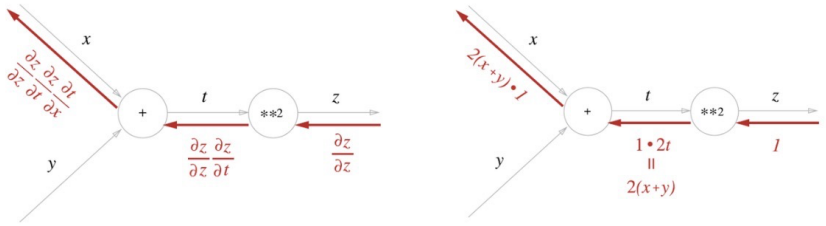


어떻게 가능한건가?

⇒ 합성 함수 미분의 성질

$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t} * \frac{\partial t}{\partial x}$

합성 함수의 미분은 합성 함수를 구성하는 각 함수의 미분의 곱으로 나타낼 수 있다.

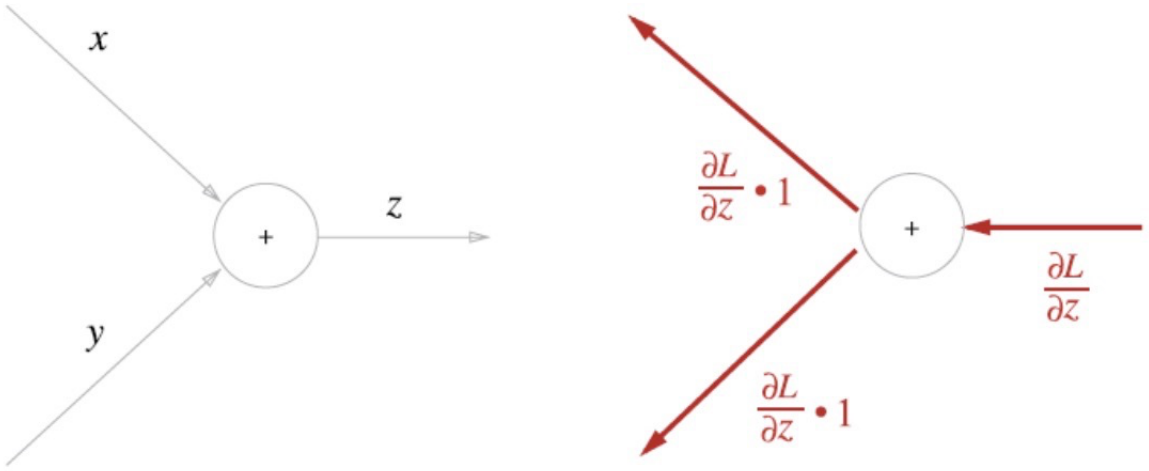


노드별 역전파

덧셈노드의 역전파

입력값을 그대로 흘려보낸다

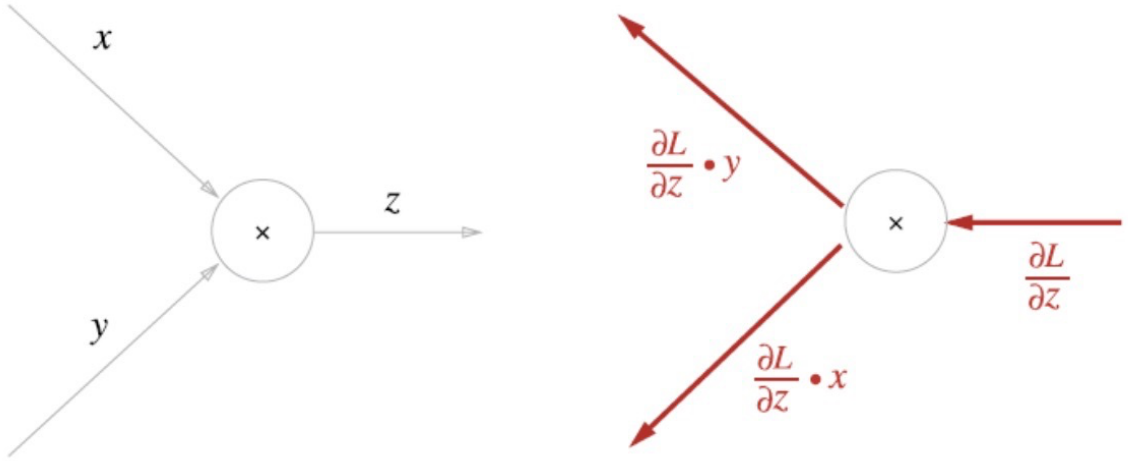
상류 노드에서 최종적으로 출력된 값이 L이라고 가정했을 때




```
class AddLayer:
    def __init__(self):
        #덧셈 계층은 흘러 보내기만 하면 되서
        #초기화 필요 없음
        pass

    def forward(self,x:float,y:float) -> float:
        out = x+y
        return out

    def backward(self,dout:float) -> Tuple[float,float]:
        dx = dout * 1
        dy = dout * 1
        return dx,dy
```

곱셈 노드의 역전파

상류의 값에 입력 신호들을 서로 바꾼 값을 곱하여 하류로 보냄



그렇기에 덧셈노드와 달리 곱셈 노드 구현시에는 순전파의 입력 신호를 변수에 저장해야 함

```
from typing import Tuple


class MulLayer:
    def __init__(self):
        self.x = None
        self.y = None

    def forward(self, x:float,y:float) -> float:
        self.x = x
        self.y = y
        out = x * y

        return out

    def backward(self, dout:float) -> Tuple[float,float]:
        # x와 y를 바꾼다
        dx = dout * self.y
        dy = dout * self.x

        return dx,dy
```

나누기 노드의 역전파

$y = \frac{1}{x}$일 때, $\frac{\partial y}{\partial x} = -\frac{1}{x^2} = -y^2$

역전파 때 상류에서 흘러온 값에 -y^2을 곱한다

(순전파의 출력을 제곱한 후 마이너스를 붙인 값)

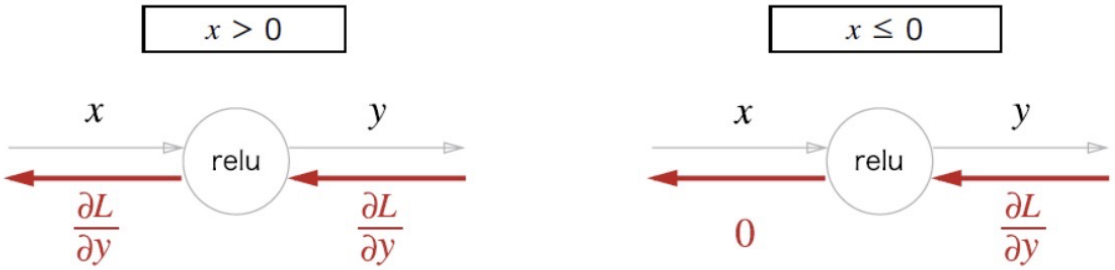
활성화 함수 계층

RELU

$$y = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

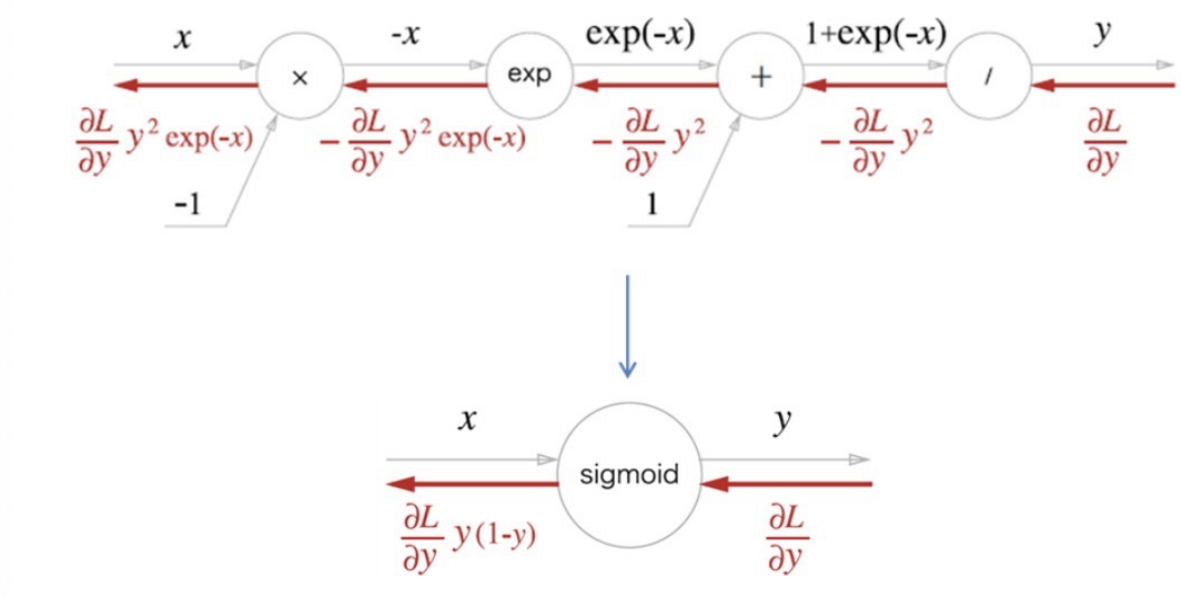
$$\frac{\partial y}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

- 순전파 때의 입력인 x가 0보다 크면 역전파 때 상류의 값을 그래프의 하류로 흘림
- 순전파 때 입력인 x가 0 이하이면 역전파때 0을 보냄



시그모이드 함수

$$\frac{1}{1+exp(-x)}$$



1. 나누기 노드 : 순전파의 출력을 제곱한 후 마이너스를 붙힘

   $y = \frac{1}{x}$일 때, $\frac{\partial y}{\partial x} = -\frac{1}{x^2} = -y^2$

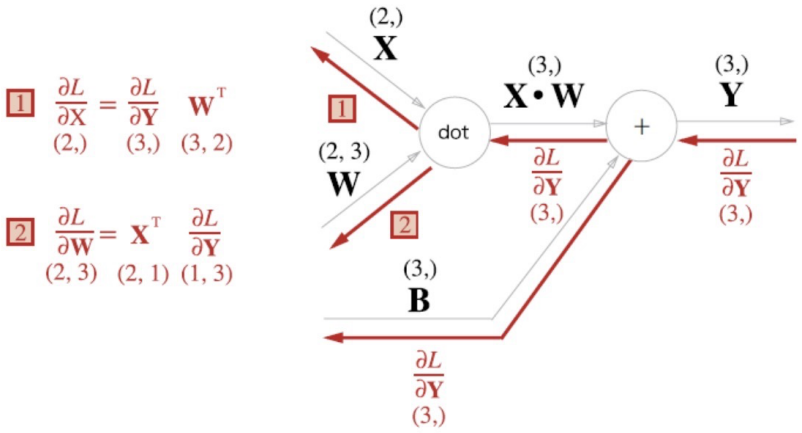
2. 플러스 노드 : 상류값 여과 없이 하류로 보냄
3. exp 노드 : exp(x)연산 수행(exp는 미분해도 exp)
4. 마이너스 노드 : -1 곱

역전파의 최종 출력은 순전파의 입력 x와 출력 y만으로 계산 가능

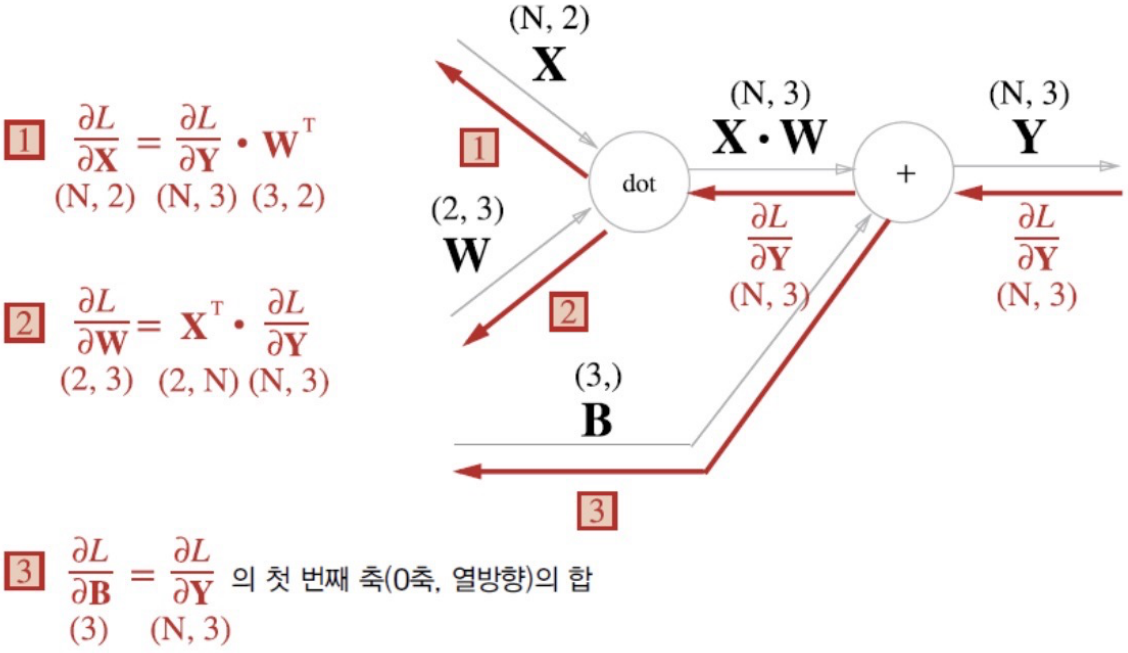
```
def sigmoid(x):
    return 1 / (1+ np.exp(-x))

class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1 - self.out) * self.out
```

Affine 계층

- 신경망의 순전파 때 수행하는 행렬 곱을 기하학에서 어파인 변환이라고 함



배치용



편향(B) 덧셈 주의 : 순전파 때의 편향 덧셈은 X*W에 대한 편향이 각 데이터에 더해짐

ex) N = 2일 때 편향은 그 두 데이터 각각의 계산 결과에 더해짐

```
import numpy as np

# 순전파 때의 편향 덧셈
X_dot_W = np.array([[0,0,0], [10,10,10]])
B = np.array([1,2,3])
```

```
print(X_dot_W)
print(X_dot_W + B)

##결과 값
[[ 0  0  0]
 [10 10 10]]
[[ 1  2  3]
 [11 12 13]]


# 역전파 때의 편향 덧셈
dY = np.array([[1,2,3],[4,5,6]])
print(dY)
dB = np.sum(dY, axis=0)
print(dB)

##결과 값
[[1 2 3]
 [4 5 6]]
[5 7 9]
```

```
class Affine:
    def __init__(self, W, b):
        self.W = W
        self.b = b
        self.x = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.x = x
        out = np.dot(x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
```

```
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        return dx
```

에포크(epoch)

모든 샘플에 대해 한 번 실행되는 것

ex) 에포크가 5 ⇒ 처음부터 끝까지 다섯번 재사용

배치 사이즈

샘플을 한번에 몇개 씩 처리할지 정하는 부분

batch_size = 16 ⇒ 총 샘플 중 16개씩 집어 넣어라

참조

https://www.youtube.com/watch?v=1Q_etC_GHHk

https://brunch.co.kr/@chris-song/22

https://m.yes24.com/Goods/Detail/34970929