Probability

C_1과 C_2를 구분하는 binary classification에서 하나의 데이터 포인트 x가 주어졌을 때, 이것이 C1으로 분류될 확률을 y라고 보면, 같은 데이터가 C2로 분류될 확률은 1-y이다.

$$P(C_1 \mid x) = y$$

$$P(C_2 \mid x) = 1 - y$$

$$Choose \begin{cases} C_1 & if\ y > 0.5 \\ C_2 & if\ y \leq 0.5 \end{cases} \Leftrightarrow = \underbrace{\frac{y}{1-y} > 1}_{\text{Odds}} \Leftrightarrow \underbrace{\log_e \frac{y}{1-y} > 0}_{\text{Logit}}$$

다시 말해 x가 주어졌을 때, 결과값 y가 0.5보다 크면 x는 C_1클래스 이고, y가 0.5보다 작으면 x는 C_2클래스일 것이라고 볼 수 있다. 이를 간략하게 표현한 것이 Odds라는 단어이다.

https://m.blog.naver.com/je_un/222229440878

Odds

Odds(승산)는 Probability의 또 다른 표현법으로 **실패 비율 대비 성공 비율**을 이야기한다. 다시 말해, **성공 확률이 실패 확률에 비해 얼마나 더 큰가**이다. 즉, Odds가 클수록 성공확률이 크다는 의미이다.

Odds는 도박에서 얻을 확률(Pay off)과 잃을 확률(Stake)의 비율을 뜻하는 영어단어이다. C_1일 확률을 '얻는다'로 보고, C_2일 확률을 '잃는다'고 본다. 이때 y가 1-y보다 커서 비율이 1보다 커질 때는 C_1 클래스로 분류하고, 그 반대는 C_2 클래스로 분류한다는 뜻이다.

만약 어떠한 이벤트가 일어날 확률이 60%라고 할 때, Odds는 0.6/0.4 = 1.5 와 같이 계산된다.
따라서 임의의 사건 A가 발생할 확률 p에 대한 Odds는 아래와 같이 정의된다.

$$odds = \frac{P(A)}{P(A^c)} = \frac{p}{1-p}$$

Logit

logit은 Odds에 자연로그를 씌운 것으로 아래와 같이 정의된다. logit + odds = logistic + probit = logit 에서 나온 말이다. log 변환은 통계학에서 자주 사용하는 변환이다.

$$L = \log(Odds) = \ln \frac{p}{1-p}$$

Odds는 그 값이 1보다 큰지가 결정의 기준이고, logit은 0보다 큰지가 결정의 기준이다.
이러한 logit함수와 sigmoid(logistic) 함수는 서로 역함수(y=x에 대하여 대칭) 관계이다.

$$p = \frac{1}{1+e^{-L}} = \frac{e^{-L}}{e^{-L}+1}$$

logit 함수에서 sigmoid 함수를 유도해보자면 아래와 같다.

$$\begin{aligned} Logit(p) &= \log_e\left(\frac{1}{1-p}\right) = L \\ &= \frac{p}{1-p} = \exp(L) \\ &= \frac{1}{p} - 1 = \frac{1}{\exp(L)} \\ &= \frac{1}{p} = \frac{1+\exp(L)}{\exp(L)} \\ &= p = \frac{\exp(L)}{1+\exp(L)} \\ &= p = \frac{1}{1+\exp(-L)} \\ &= Sigmoid(L) \end{aligned}$$

$$\therefore Logit(p) = L,\ Sigmoid(L) = p$$

결국 logit(y) =t , sigmoid(t) = y 로 다시 표현할 수 있게 된다.

로지스틱 회귀는 왜 시그모이드 함수를 사용하는가?

binary classification에서 확률 p의 범위는 [0,1]이다. 따라서 단순 선형함수 y=Wx + b로 풀기 힘들다는 것은 이전에 언급한 바가 있다. 그래서 우리는 Odds와 Logit을 이용해야 한다.

Odds의 범위는 [0,∞], log(Odds(p))는 [-∞,∞]범위로 넓혀진다. 즉 Logit을 사용하면 범위가 실수 전체가 된다. 이는 단순 선형 함수로 풀기 힘든 binary classification 문제를 log(Odds(p)) = Wx+b 로 선형회귀분석 할 수 있도록 만들어준다.

$$\log\left(Odds(p)\right) = Wx + b$$

이 식을 위에서 유도한 것처럼 p로 정리하면 아래의 수식(시그모이드 함수)이 나온다.

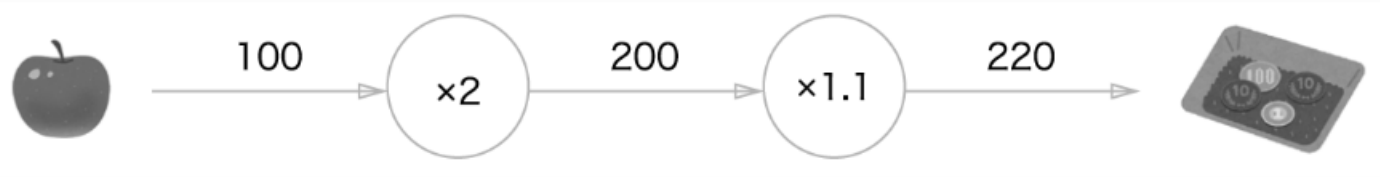
$$p(x) = \frac{1}{1 + e^{-(Wx+b)}}$$

x 데이터가 주어졌을 때 성공확률을 예측하는 Logistic Regression은 결국 Sigmoid 함수의 W와 b를 찾는 문제가 된다. 또한 시그모이드 함수는 [0,1] 범위인 확률을 [-∞,∞]로 넓히기 때문에 보통 멀티 클래스 분류문제에서 softmax 함수의 입력으로 사용된다.

오차역전파

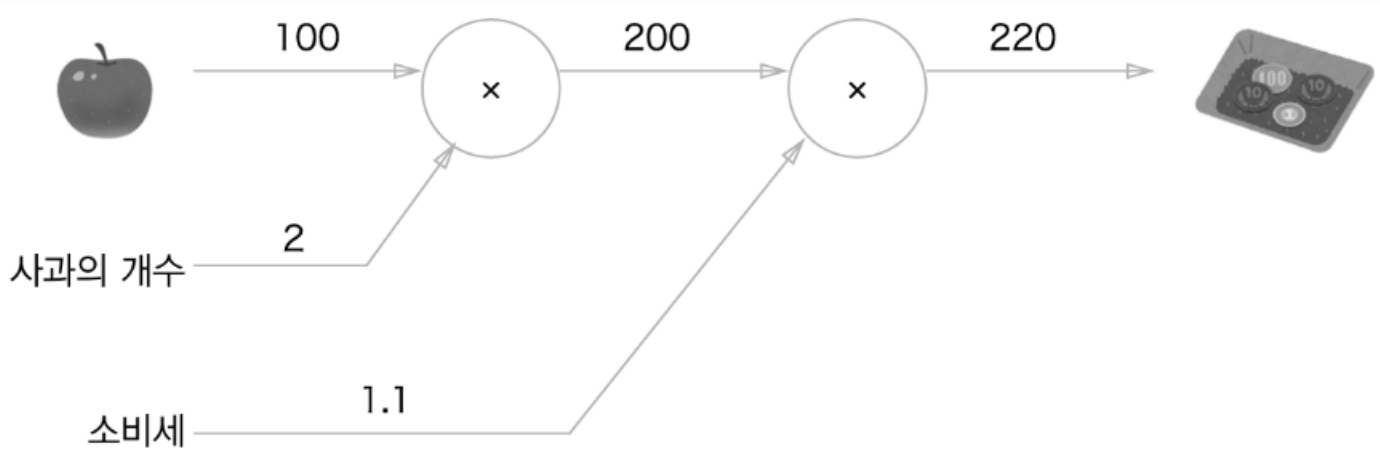
계산 그래프로 나타내기

계산 그래프는 계산 과정을 노드와 화살표(에지)로 표현합니다. 노드는 원으로 표기하고 원 안에 연산 내용을 적습니다. 계산 결과는 화살표 위에 적어 왼쪽에서 오른쪽으로 전해지게 합니다.



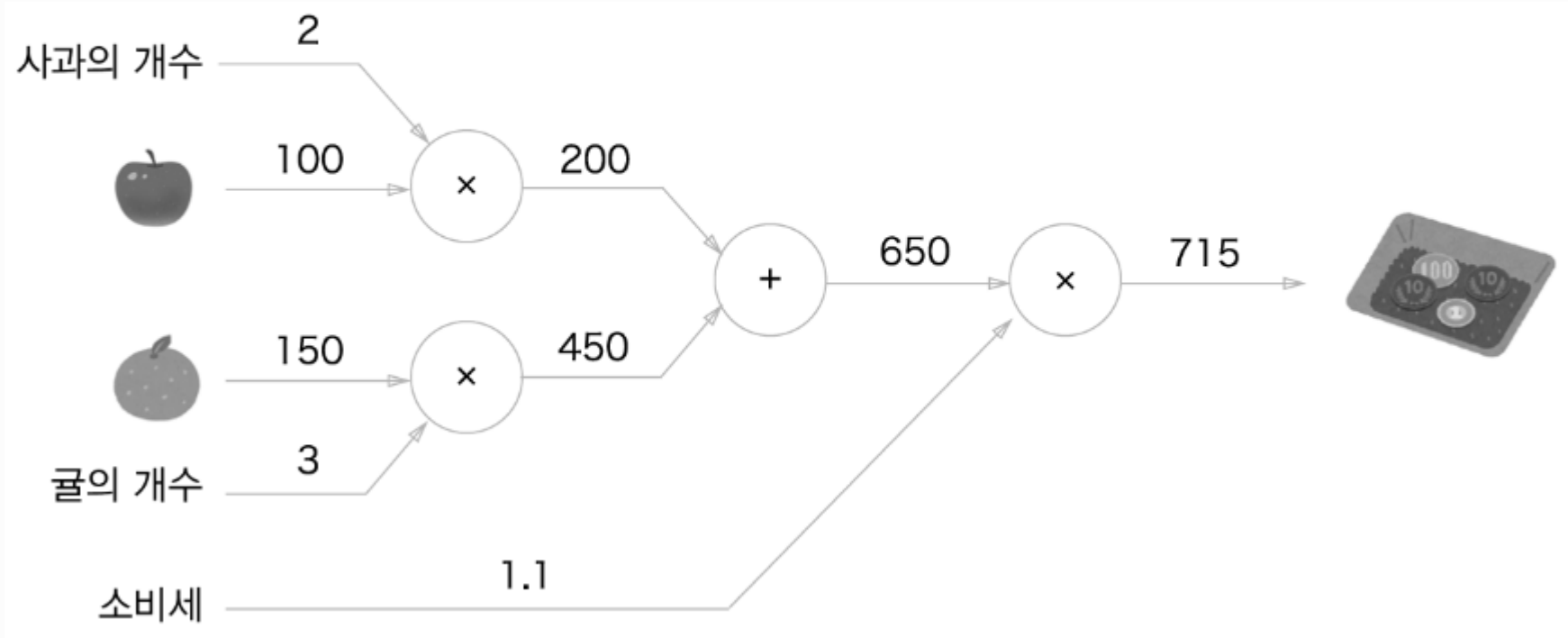
노드에는 연산만을 나타내도록 표현할 수 있습니다.

문제1: 수퍼에서 1개에 100원인 사과를 2개 샀습니다. 이때 지불 금액을 구하세요. 단 소비세가 10% 부과됩니다.



https://compmath.korea.ac.kr/appmath2021/BackPropagation.html

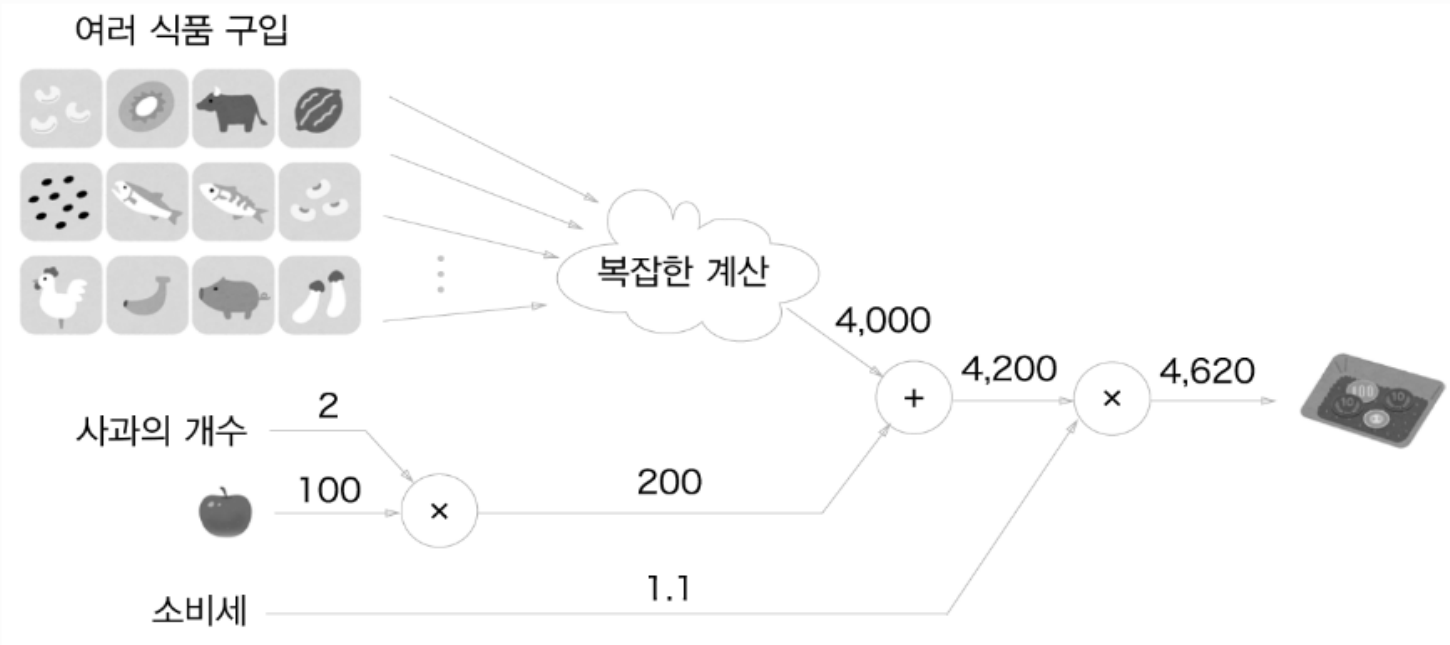
문제2: 수퍼에서 1개에 100원인 사과를 2개, 귤을 3개 샀습니다. 사과는 1개에 100원, 귤은 1개 150원입니다. 소비세가 10%일 때 지불 금액을 구하세요.



국소적 계산

국소적 계산이란 전체에 어떤 일이 벌어지든 상관없이 자신과 관계된 정보만으로 결과를 출력하는 것을 말합니다.

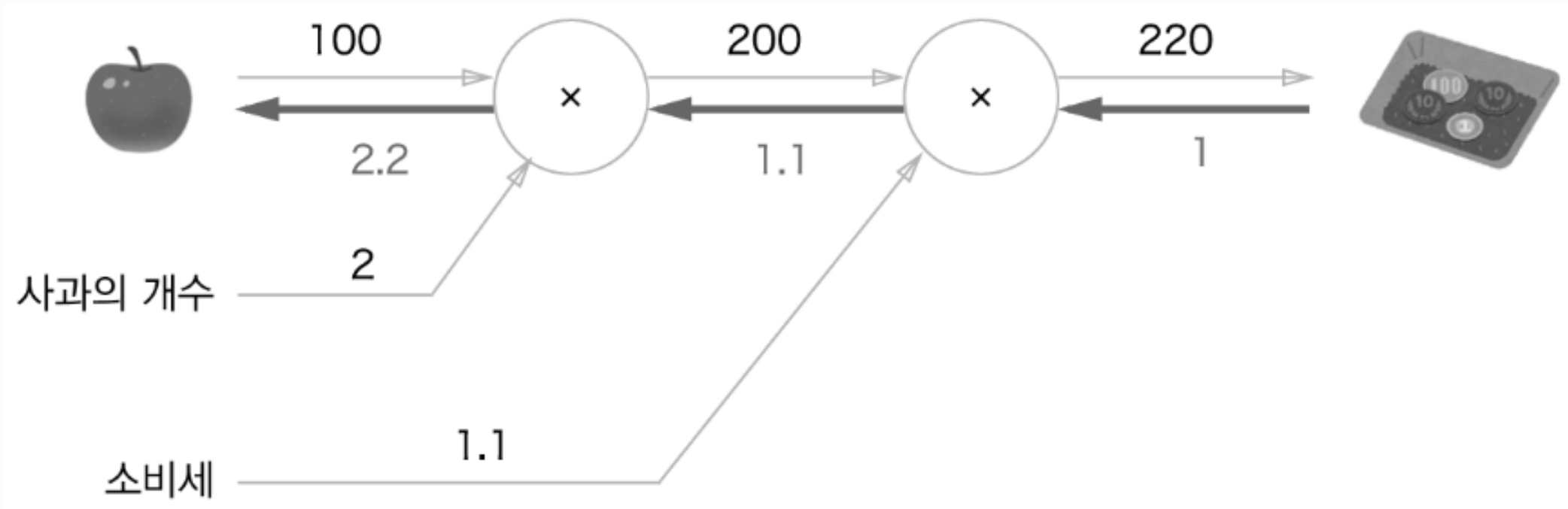
수퍼에서 사과 2개를 포함하여 여러 식품을 구입하는 경우를 생각해봅니다. 여러 식품을 구입하여 총 금액이 4000원이 되었습니다. 사과와 구입한 금액을 더하는 계산은 다른 식품들이 어떻게 계산되었는지 상관하지 않고 사과값과 4000원을 더하면 된다는 것이 국소적 계산입니다.



계산 그래프 장점

계산 그래프의 장점은 앞에서 살펴본 바와 같이 국소적 계산입니다. 아무리 복잡한 계산이더라도 각 노드에서는 단순한 계산에 집중할 수 있다는 것입니다. 또 다른 이점은 중간 계산 결과를 저장할 수 있다는 것입니다. 가장 큰 이유 중 하나는 역전파를 이용해 미분을 효율적으로 계산할 수 있다는 점에 있습니다.

사과 가격에 대한 지불 금액의 미분값은 계산 그래프에서 역전파를 하면 구할 수 있습니다. 역전파는 순전파와 반대 방향의 화살표(굵은 선)로 그립니다. 이 전파는 국소적 미분을 전달하고 그 미분값은 화살표 아래에 적습니다. 이 예에서는 역전파는 오른쪽에서 왼쪽으로 `1 -> 1.1 -> 2.2` 순으로 전달합니다. 이 결과로 사과 가격에 대한 지불 금액의 미분값은 `2.2`라 할 수 있습니다. 사과가 1원 오르면 지불 금액은 2.2원 오르게 됩니다.



CHAIN RULE

$$f = wx + b$$

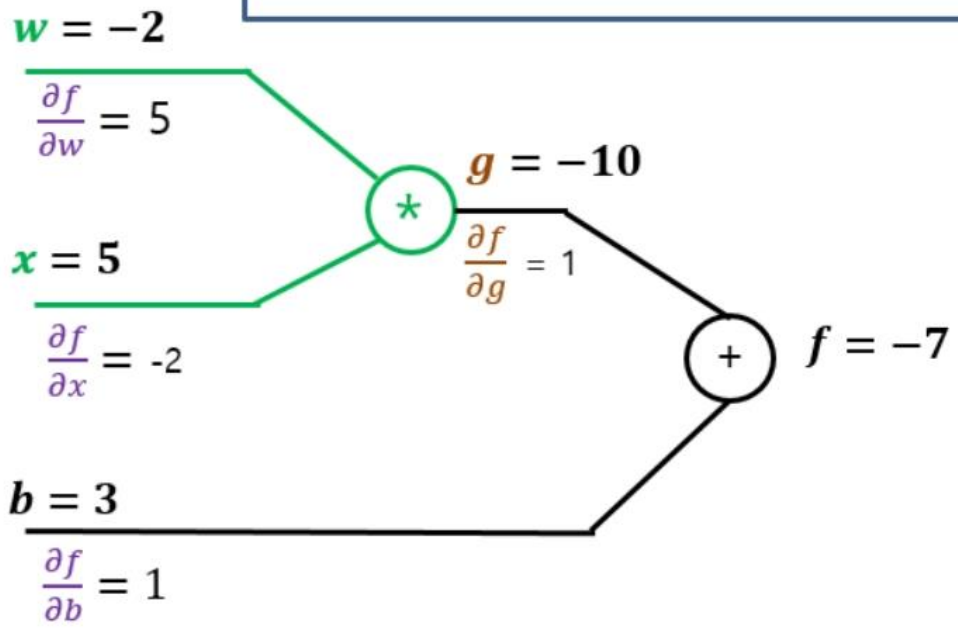
$g = wx$, $f = g + b$

$\frac{\partial g}{\partial w} = x, \frac{\partial g}{\partial x} = w$ $\quad$ $\frac{\partial f}{\partial g} = 1, \frac{\partial f}{\partial b} = 1$

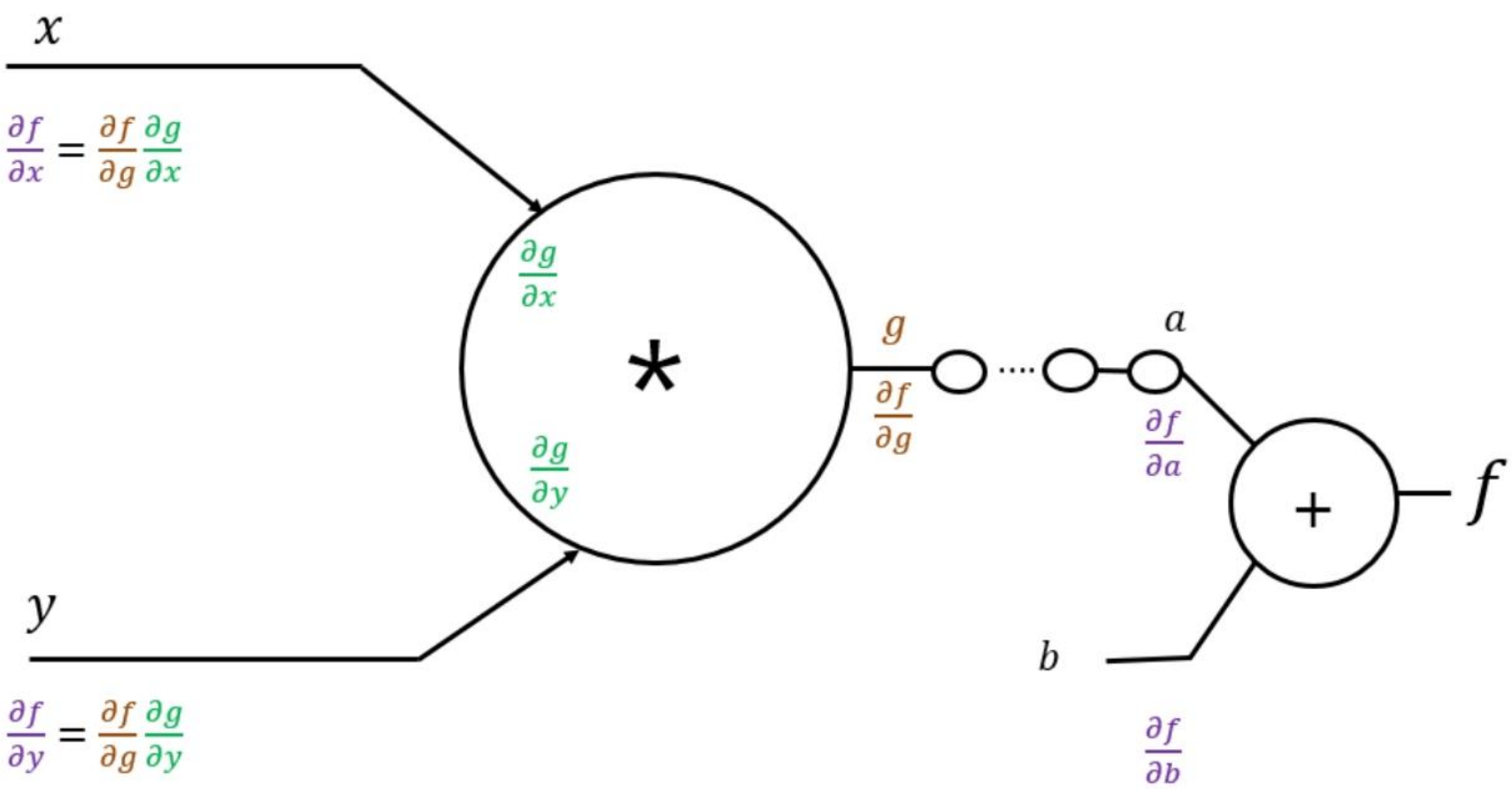
① forward($w = -2, x = 5, b = 3$)
② backward

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = 1 * w = -2$$

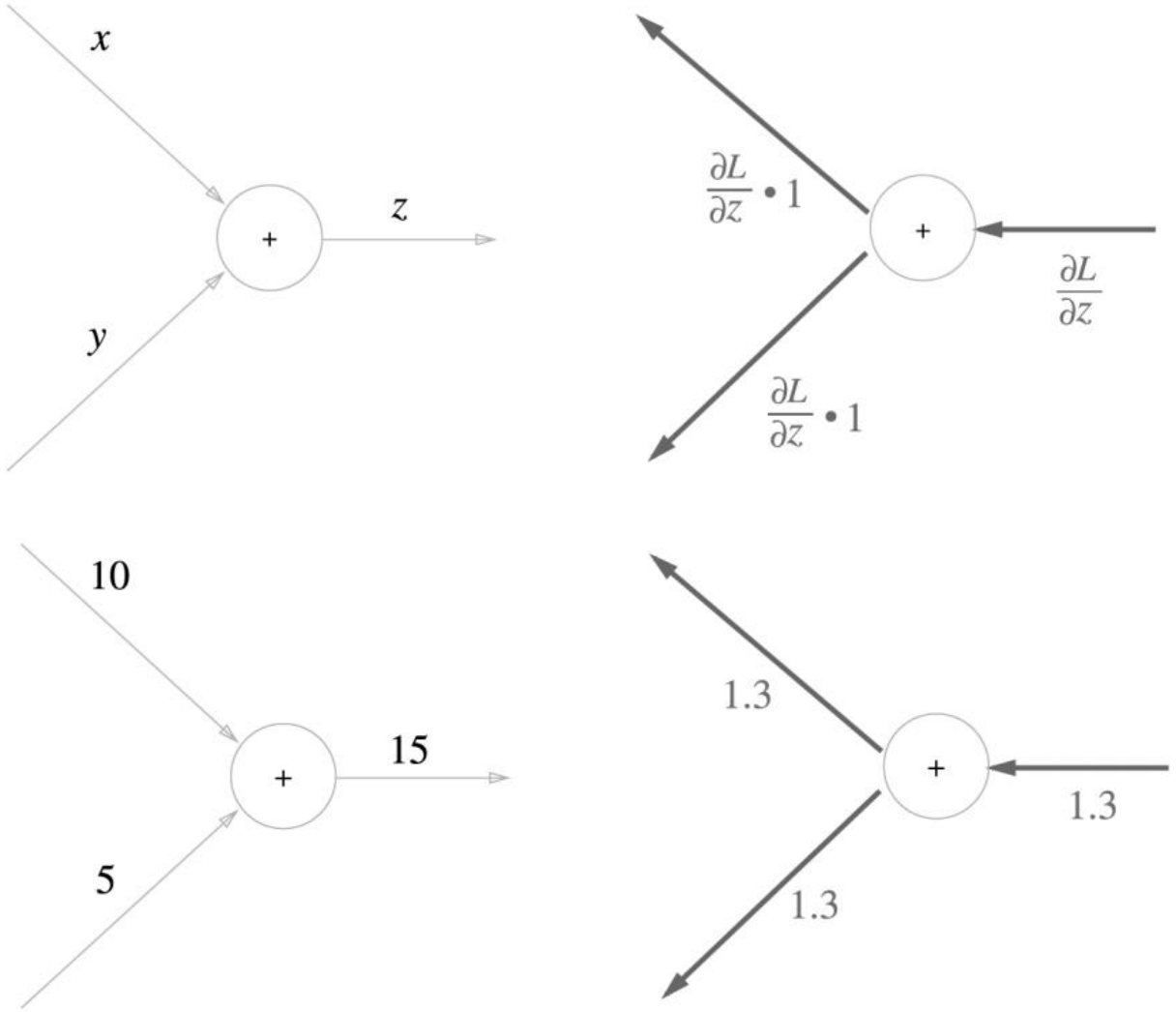
$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = 1 * x = 5$$



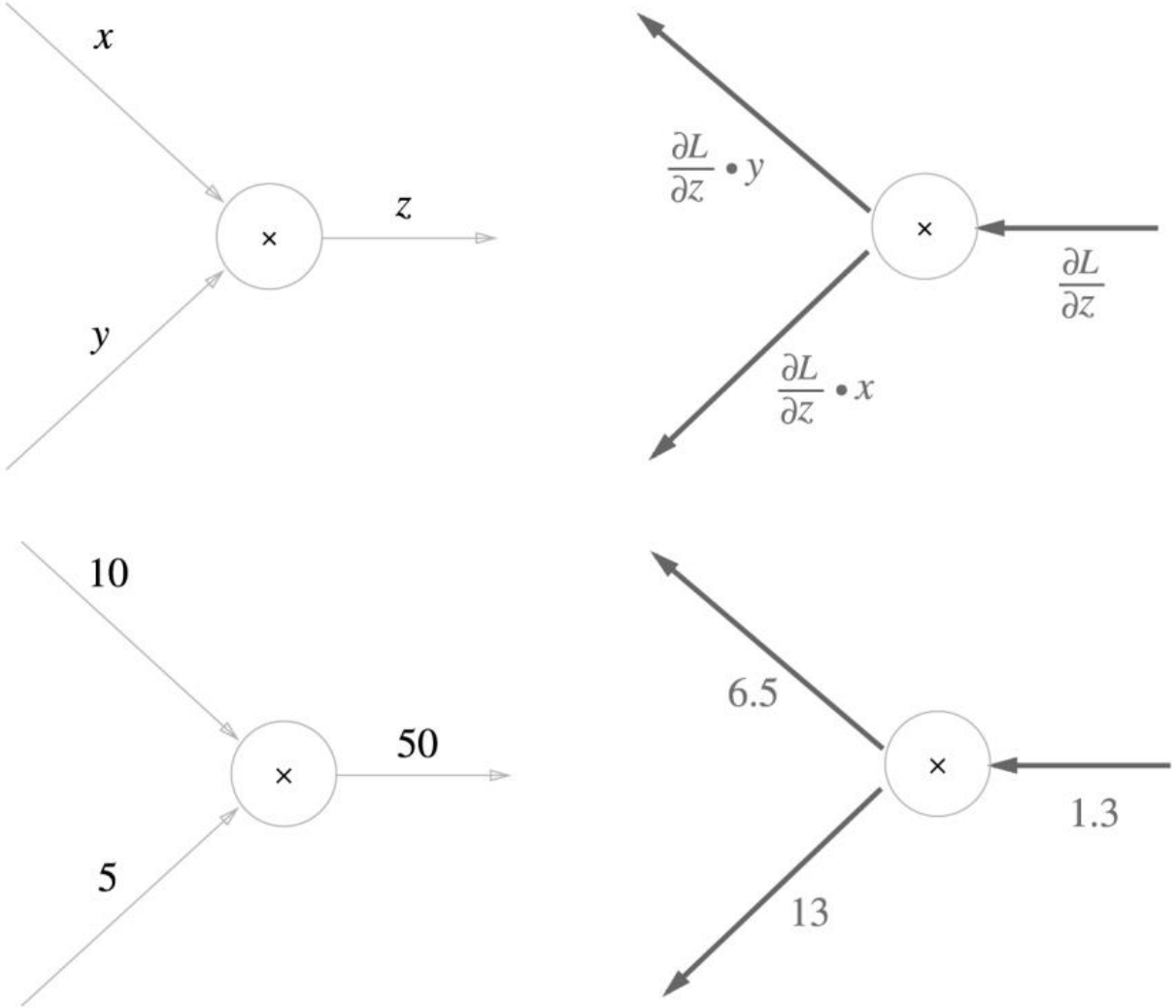
CHAIN RULE


x
$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$
$\frac{\partial g}{\partial x}$
*
$\frac{\partial g}{\partial y}$
y
$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial y}$
g
$\frac{\partial f}{\partial g}$
a
$\frac{\partial f}{\partial a}$
+
f
b
$\frac{\partial f}{\partial b}$

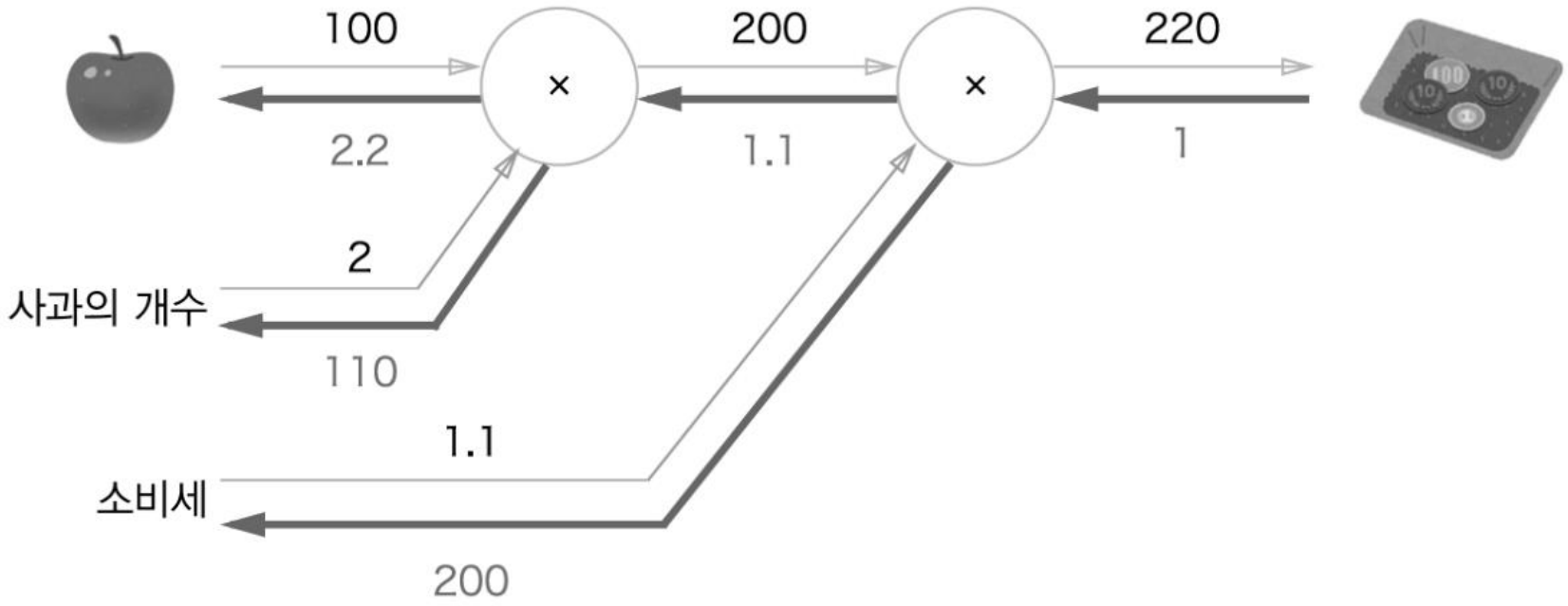
계산 그래프 (덧셈)


x
y
+
z
10
5
+
15
$\frac{\partial L}{\partial z} \cdot 1$
+
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial z} \cdot 1$
1.3
1.3
+
1.3
1.3

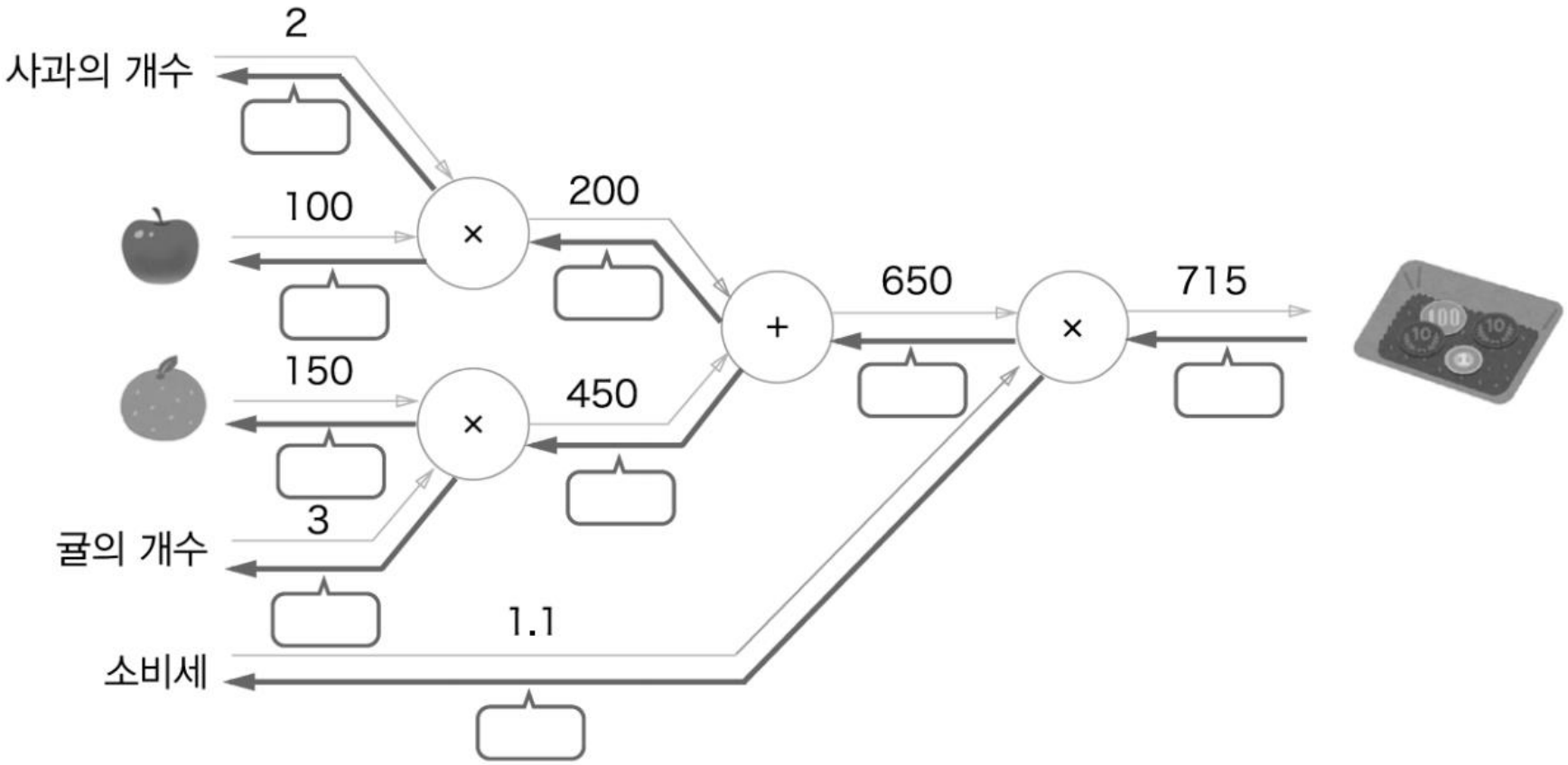
계산 그래프 (곱셈)


x
y
×
z
$\frac{\partial L}{\partial z} \cdot y$
$\frac{\partial L}{\partial z}$
$\frac{\partial L}{\partial z} \cdot x$
10
5
50
6.5
1.3
13

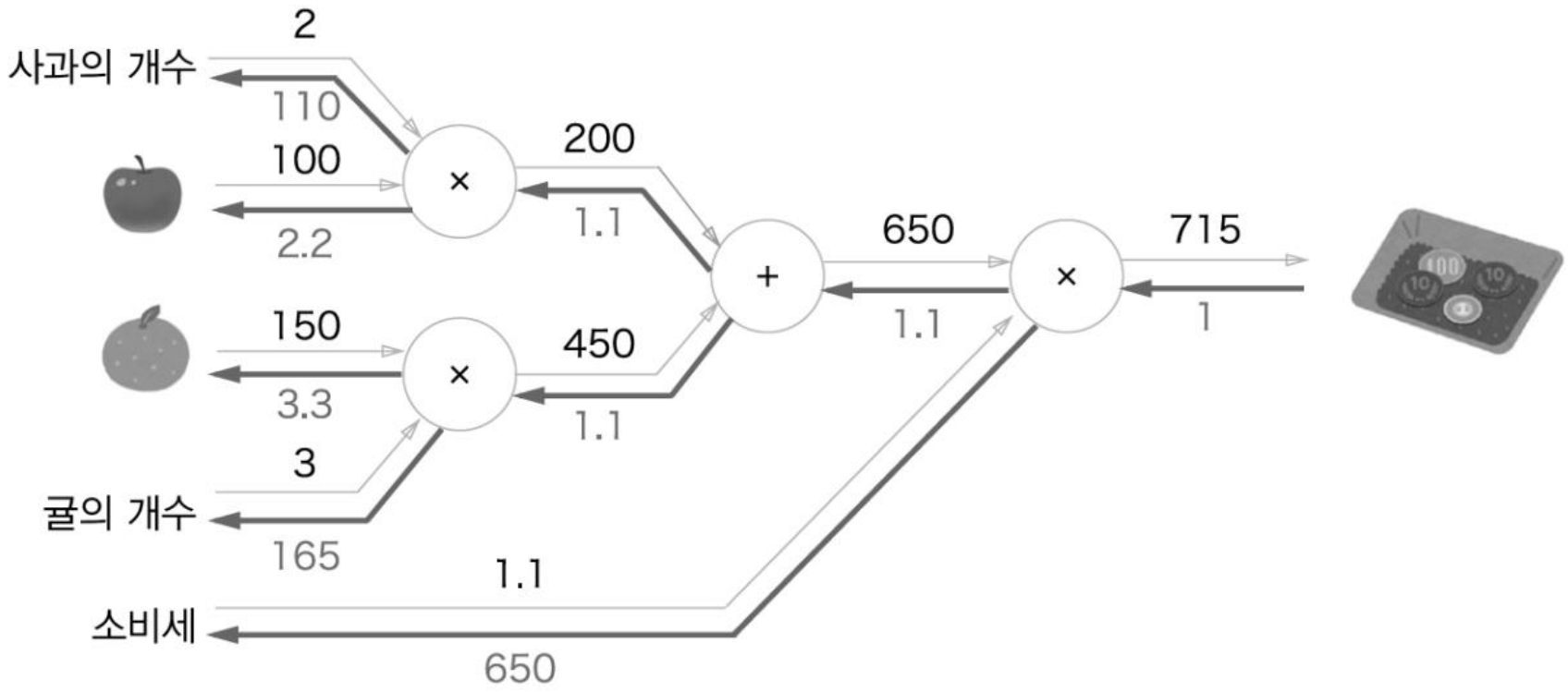
사과 그래프



과일 그래프 (퀴즈)

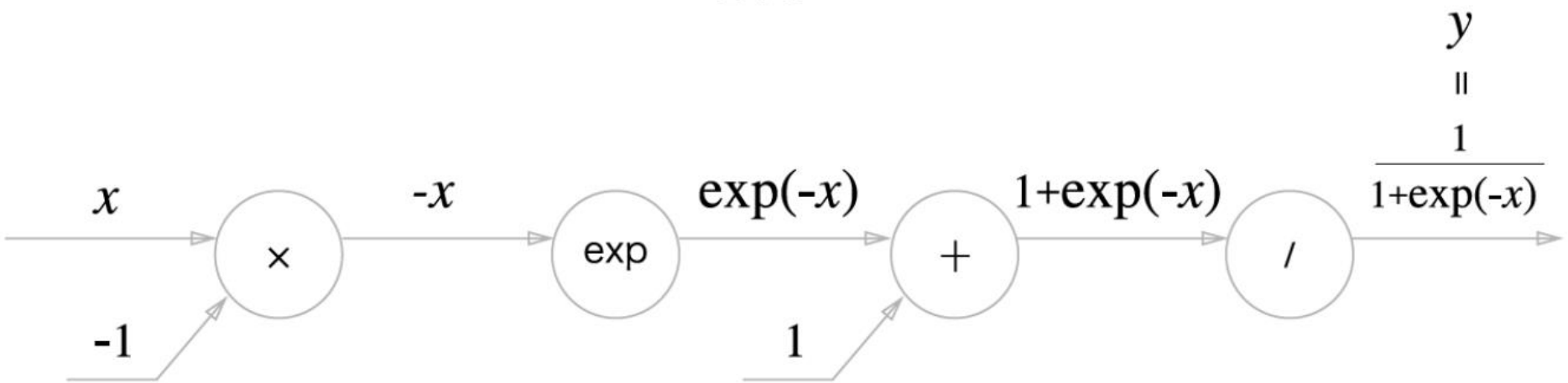


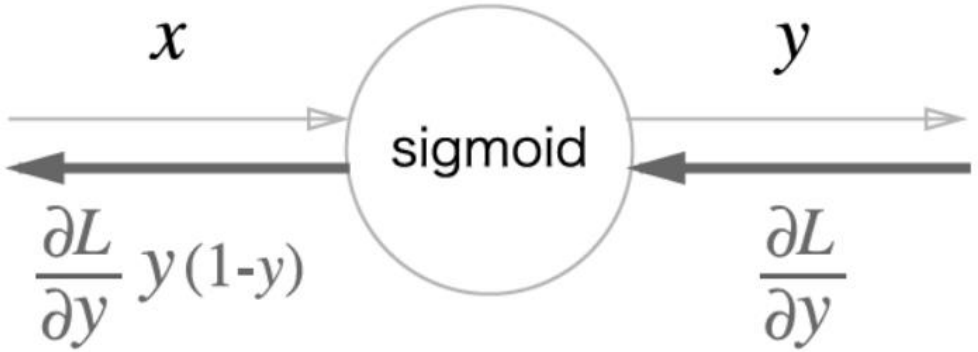
과일 그래프


2
사과의 개수
110
100
2.2
×
200
1.1
150
3.3
×
450
1.1
3
귤의 개수
165
+
650
1.1
×
715
1
1.1
소비세
650

SIGMOID

$$g(z)=\frac{1}{1+e^{-z}}$$



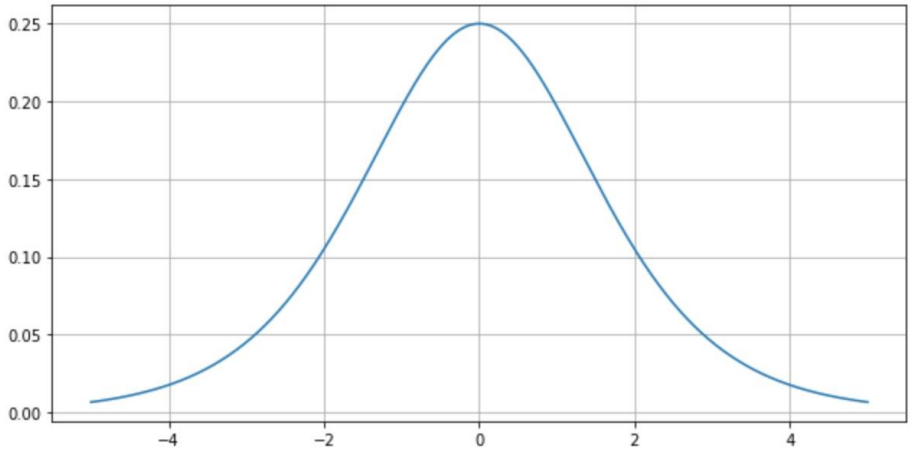


sigmoid 미분 과정

sigmoid 함수 미분 절차는 다음과 같습니다.

$$
\begin{aligned}
\frac{d}{dx} sigmoid(x) &= \frac{d}{dx}(1+e^{-x})^{-1} \\
&= (-1)\frac{1}{(1+e^{-x})^2}\frac{d}{dx}(1+e^{-x}) \\
&= (-1)\frac{1}{(1+e^{-x})^2}(0+e^{-x})\frac{d}{dx}(-x) \\
&= (-1)\frac{1}{(1+e^{-x})^2}e^{-x}(-1) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
&= \frac{(1+e^{-x})}{(1+e^{-x})^2} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}}(1-\frac{1}{1+e^{-x}}) \\
&= sigmoid(x)(1-sigmoid(x))
\end{aligned}
$$

$$\frac{d}{dx} sigmoid(x) = sigmoid(x)(1-sigmoid(x))$$

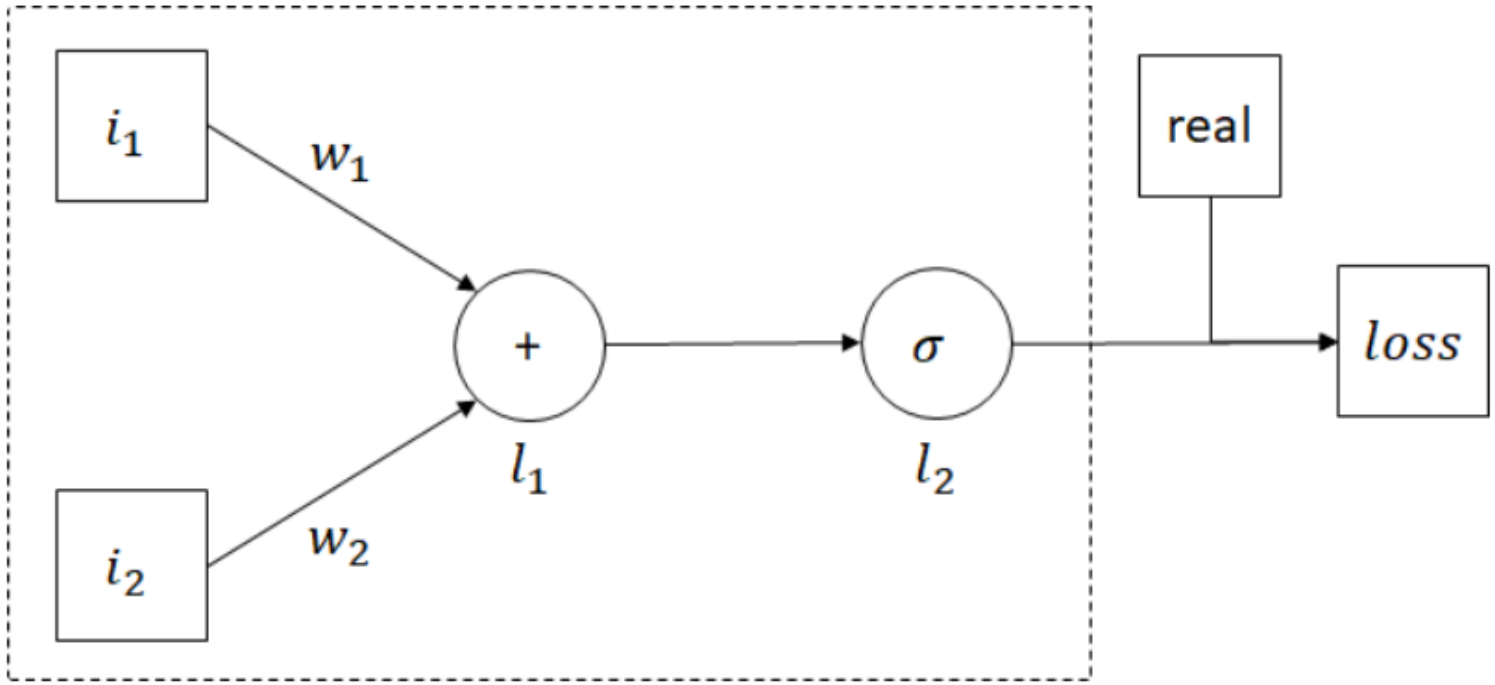


미분계수를 보면 최댓값은 0.25입니다. deep learning에서 학습을 위하여 역전파(Backpropagation)를 계산하는 과정에서 activation function의 미분 값을 곱하는 과정이 포함됩니다. sigmoid를 활성 함수로 사용하는 경우 은닉층의 깊이가 깊다면 오차율 계산이 어렵다는 문제가 발생합니다. 이것이 딥러닝에 sigmoid를 사용할 때 "**vanishing gradient problem**"이 발행하는 이유

http://taewan.kim/post/sigmoid_diff/

거듭제곱 미분

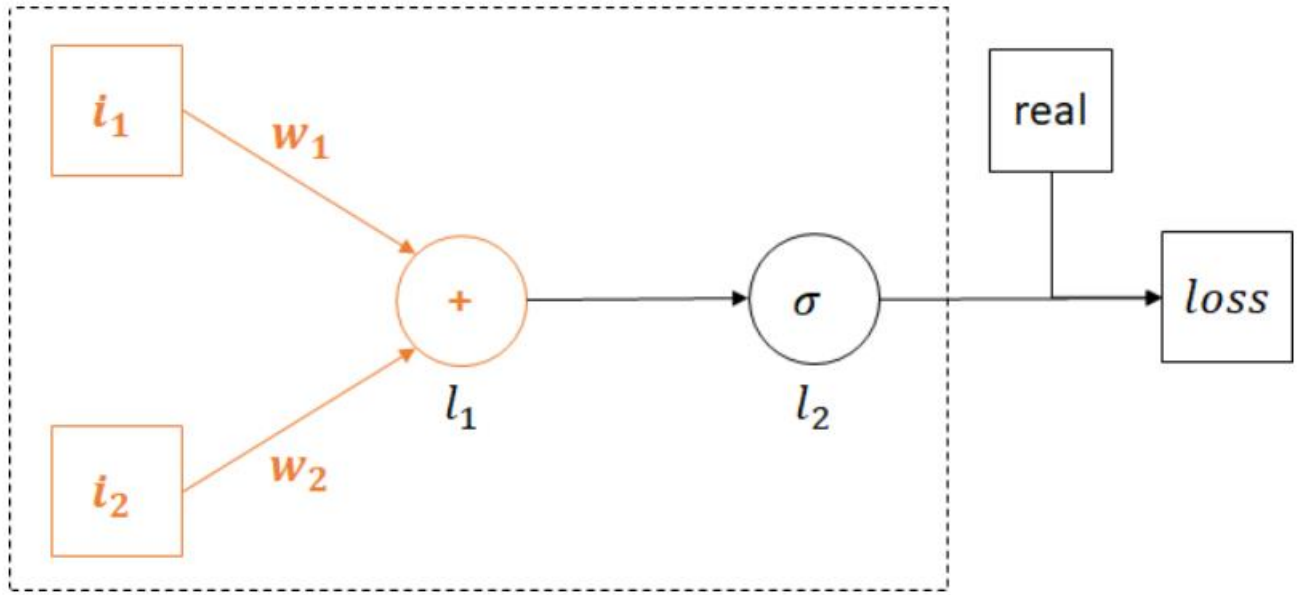
$f(x)=x^n$이 도함수가 $f'(x)=nx^{n-1}$ (n이 실수)



https://yonghyuc.wordpress.com/2019/09/10/backpropagation-weight-update/

Forward Pass

먼저 forward pass를 진행해 보자!



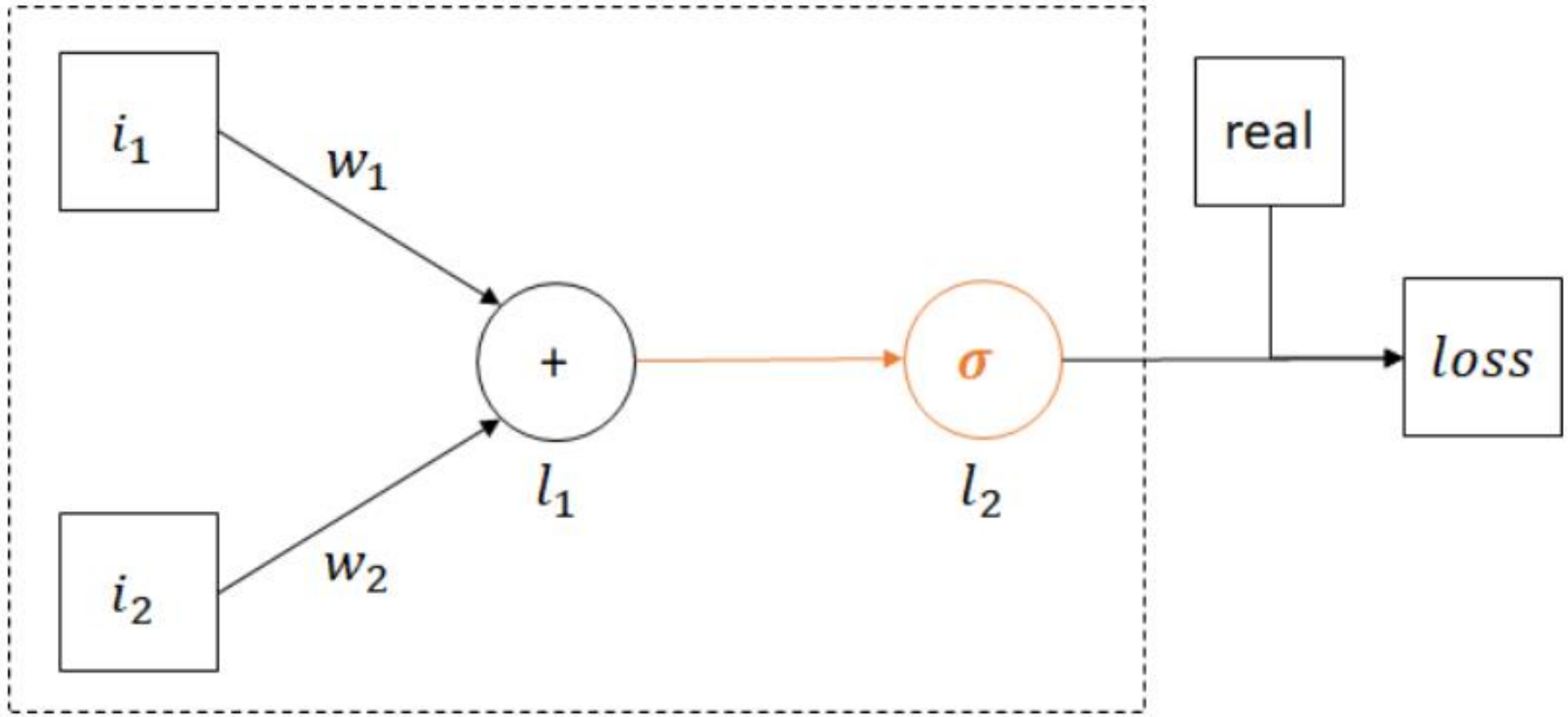
i_1, i_2에 각각 2, 5가 들어왔고, w_1, w_2가 각각 0.5, 0.1로 설정되어있다고 한다면

l_1는 $2 * 0.5 + 5 * 0.1 = 1.5$가 된다.

이를 matrix 형태로 나타내면

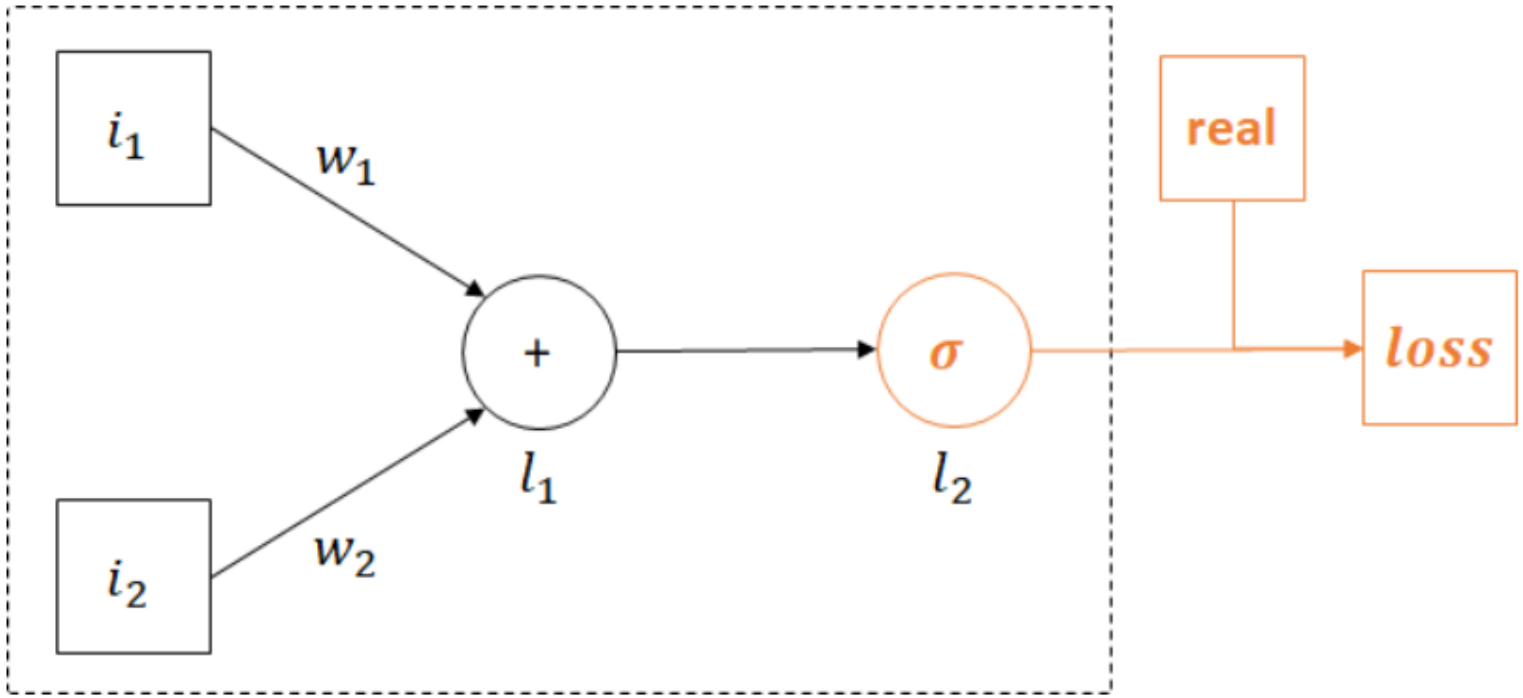
$\begin{bmatrix} 2 & 5 \end{bmatrix} \begin{bmatrix} 0.5 \\ 0.1 \end{bmatrix} = 1.5$ 로 표현할 수 있다.

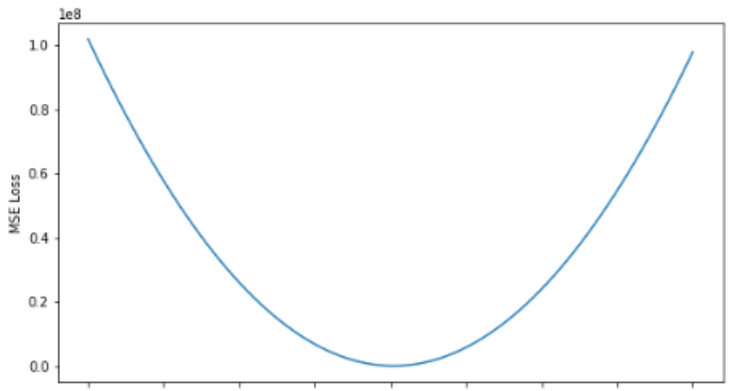
이제 non-linear function인 sigmoid 함수를 수행하면



$\sigma(1.5) = 0.818$이 나온다.

이제 마지막으로 MSE를 통해 loss만 구하면 된다.





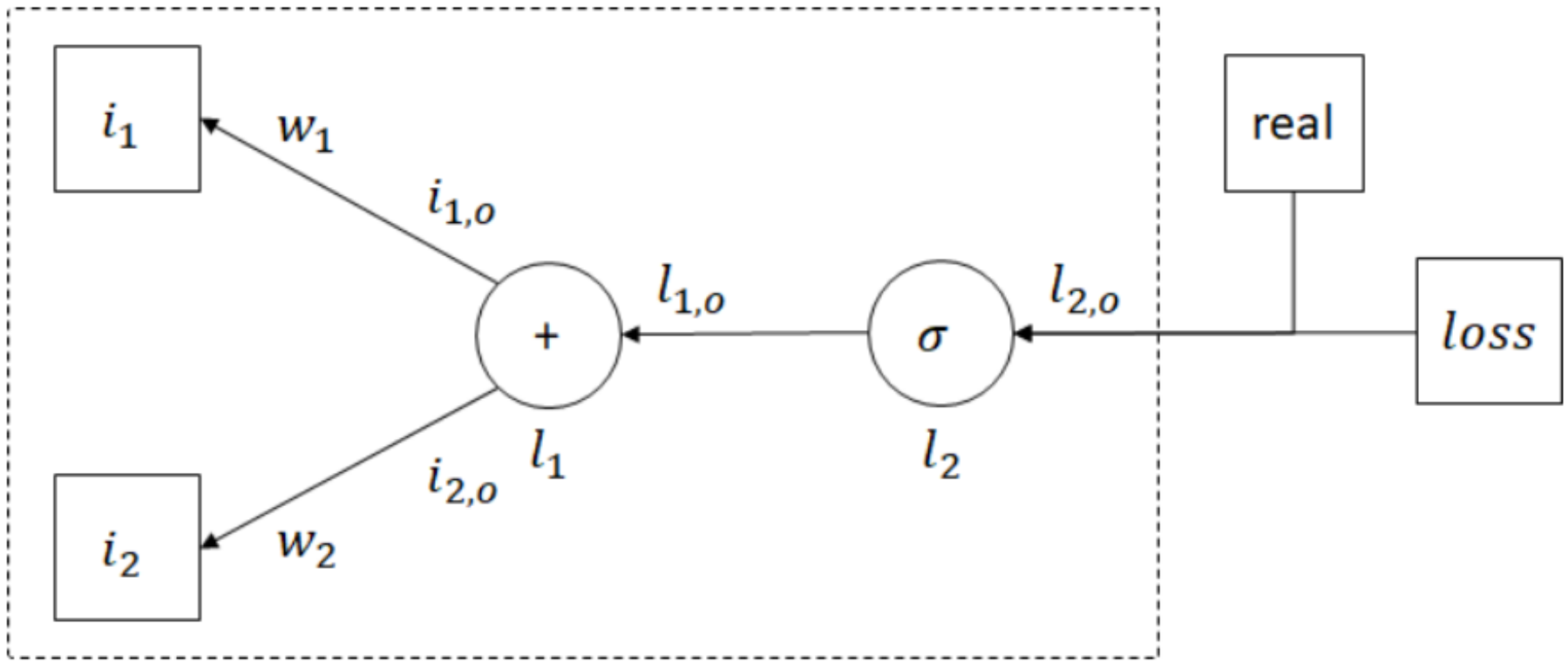
$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2$$

MSE와 loss graph (출처1) (출처2)

실제 값이 0 이라면, loss는 $\frac{1}{1}(0-0.818)^2 = 0.67$이 된다.

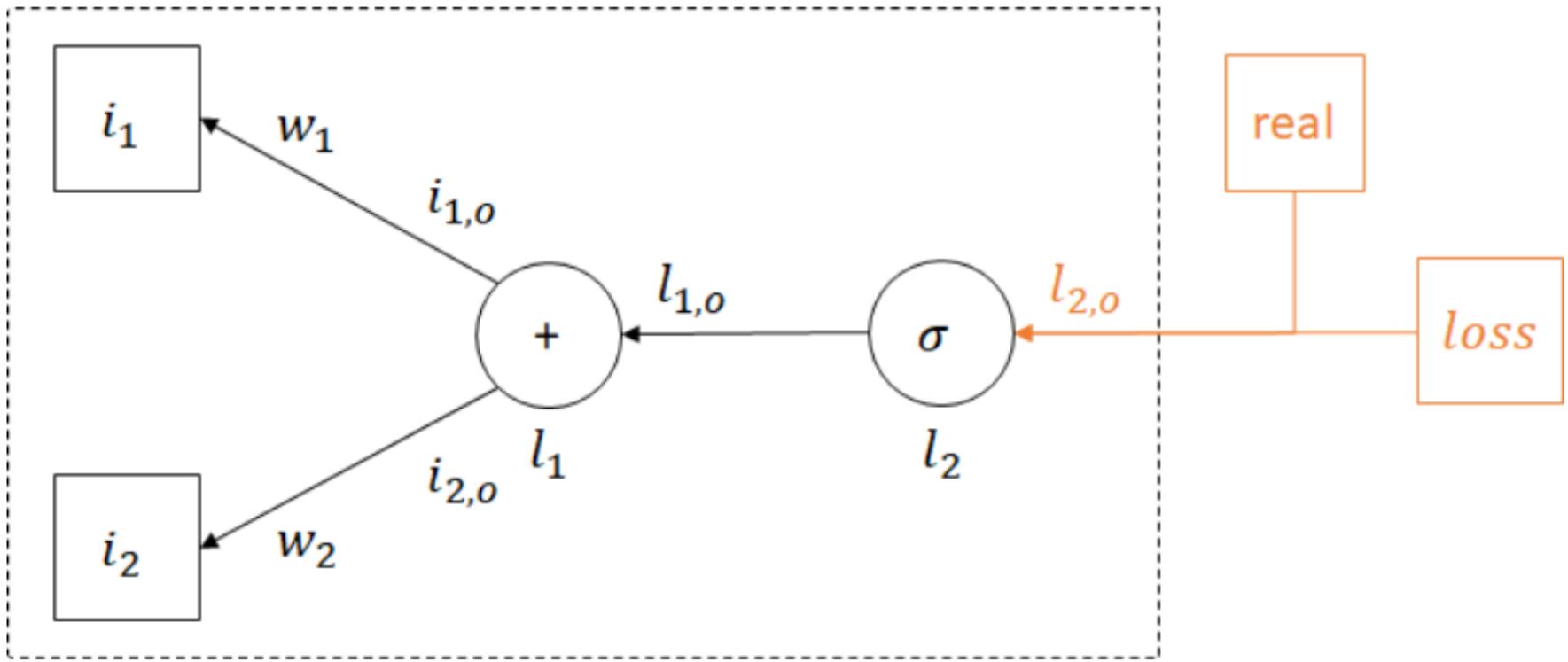
Backward Pass (Back-propagation)

이제 backward pass를 통해 우리의 weight (w_1, w_2)를 update해보자.



backward pass는 forward pass의 정 반대 방향으로 이루어 진다.

(설명을 하기 위해 각 node의 output을 추가적으로 명시하였다.)

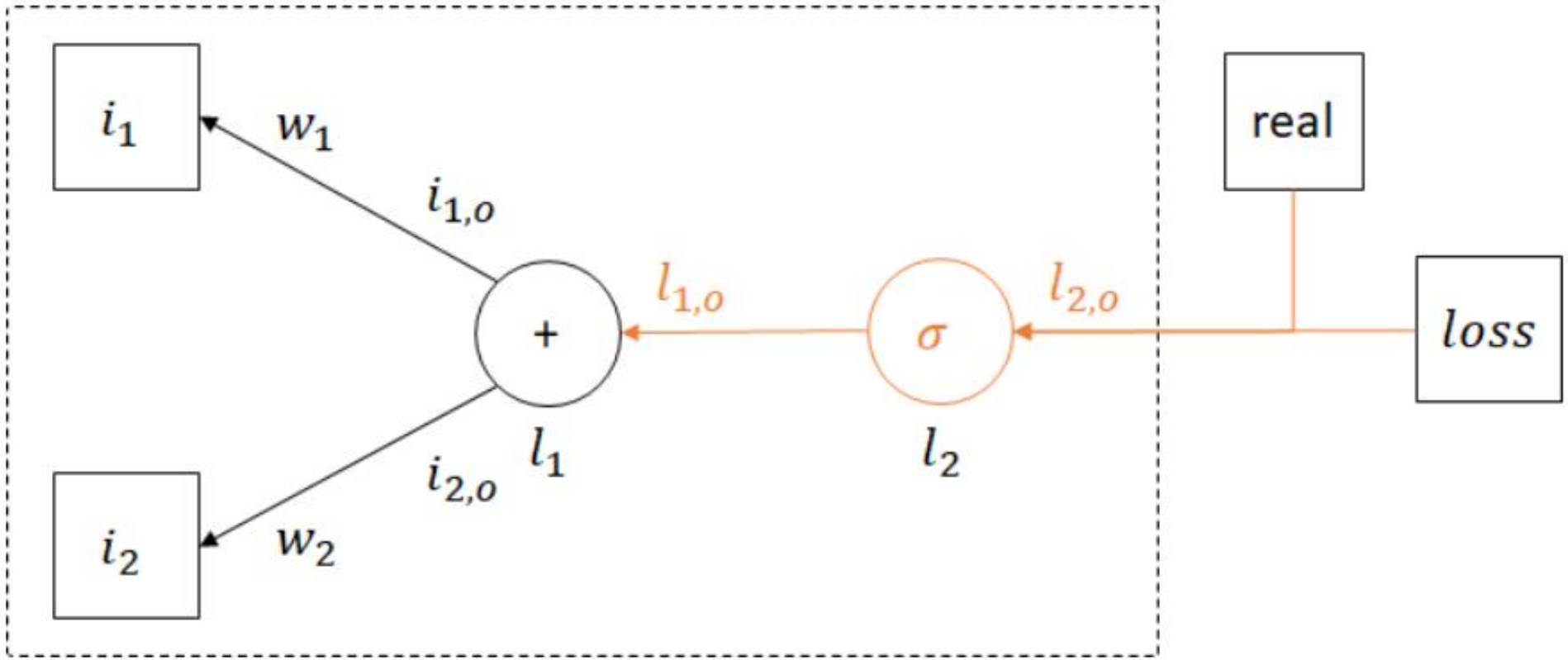


이제 첫번째로 $\frac{\partial loss}{\partial l_{2,o}}$를 구해보자.

MSE수식인 $\frac{1}{n}\Sigma(y-\tilde{y})^2$를 미분하면 $-\frac{2}{n}(y-\tilde{y})$가 되고

결국 $\frac{\partial loss}{\partial l_{2,o}} = -\frac{2}{1}(0-0.818) = 1.636$ 가 된다.

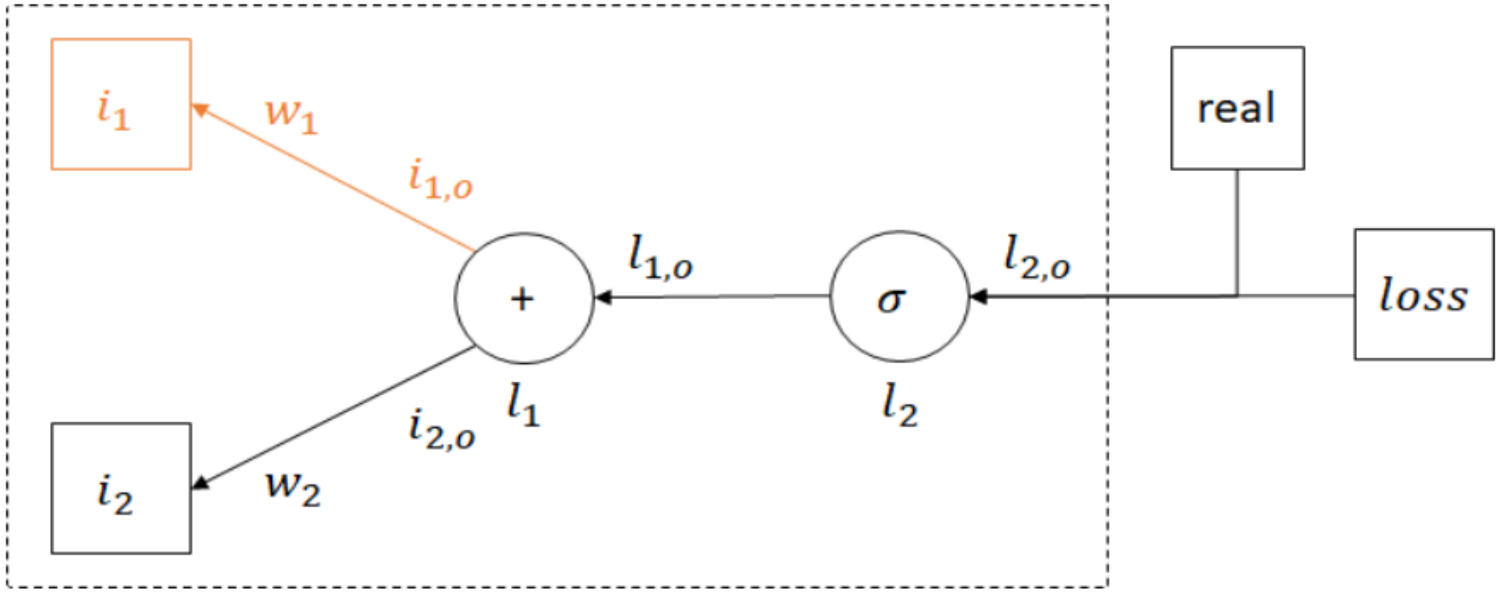
이제부터는 chain rule에 의해 연속적으로 이루어 지게 된다.



Sigmoid의 미분 값은 $\sigma'(x) = \sigma(x)(1 - \sigma(x))$이고

chain rule에 의해 $\frac{\partial loss}{\partial l_{1,o}} = \frac{\partial loss}{\partial l_{2,o}} * \frac{\partial l_{2,o}}{\partial l_{1,o}}$가 되므로

$\frac{\partial loss}{\partial l_{1,o}} = 1.636 * (0.818) * (1 - 0.818) = 0.244$가 된다.



여기서 i_1과 i_2가 더하기 (+)로 연결되고 있다.

back-propagation에서 더하기 기호는 gradient distributor 라고도 불리는데

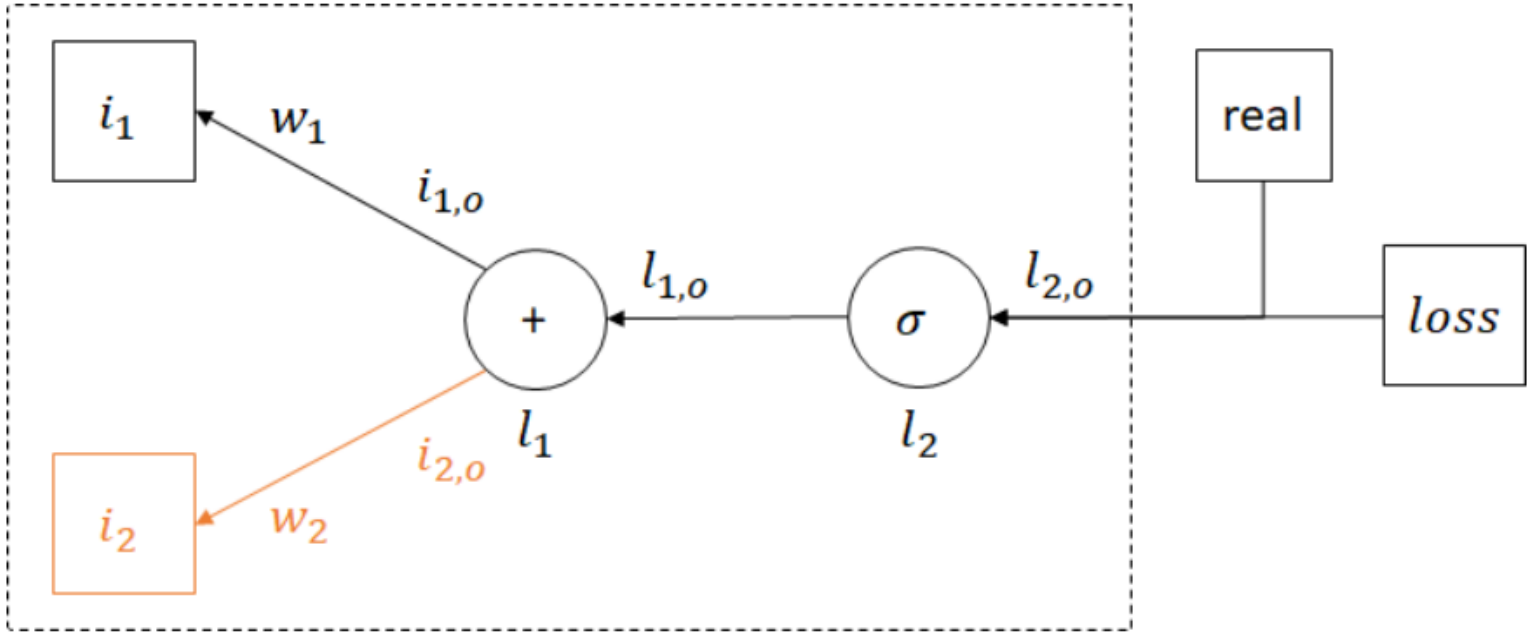
이전 단계에서 받은 gradient를 모든 방향으로 단순히 전달해 주기 때문이다.

즉, 여기서는 이전에 받아온 $\frac{\partial loss}{\partial l_{1,o}}$ 을 $i_{1,o}$와 $i_{2,o}$ 에게 각각 나눠준다.

또한 w_1와 i_1의 관계를 살펴보면

$i_{2,0} = w_1 * i_1$이므로 w_1의 gradient는 결국 i_1이 된다고 할 수 있다.

따라서 $\frac{\partial loss}{\partial w_1} = \frac{\partial loss}{\partial i_{1,o}} * \frac{\partial i_{1,o}}{\partial w_1} = 0.244 * 2 = 0.488$가 된다.



마찬가지로 $\frac{\partial loss}{\partial w_2} = \frac{\partial loss}{\partial i_{2,o}} * \frac{\partial i_{2,o}}{\partial w_2} = 0.244 * 5 = 1.22$가 된다.

이제 optimizing이 가능한 w_1과 w_2를 update하면 되는데
기본적으로 앞에서 구한 gradient에 learning rate를 곱한 값으로 weight를 update한다.
(여기서는 learning rate를 1로 설정한다)

$$w_1' = w_1 + (-\frac{\partial loss}{\partial w_1} * lr) = 0.5 + (-0.488 * 1) = 0.0122$$

이제 w_1는 0.5에서 0.136으로 update된다.
w_2 도 동일하게 계산하면 0.1에서 -1.12로 update된다.