# 딥러닝 스터디

\# 밑바닥 부터 시작하는 딥러닝

김제우

딥러닝스터디

# 목차

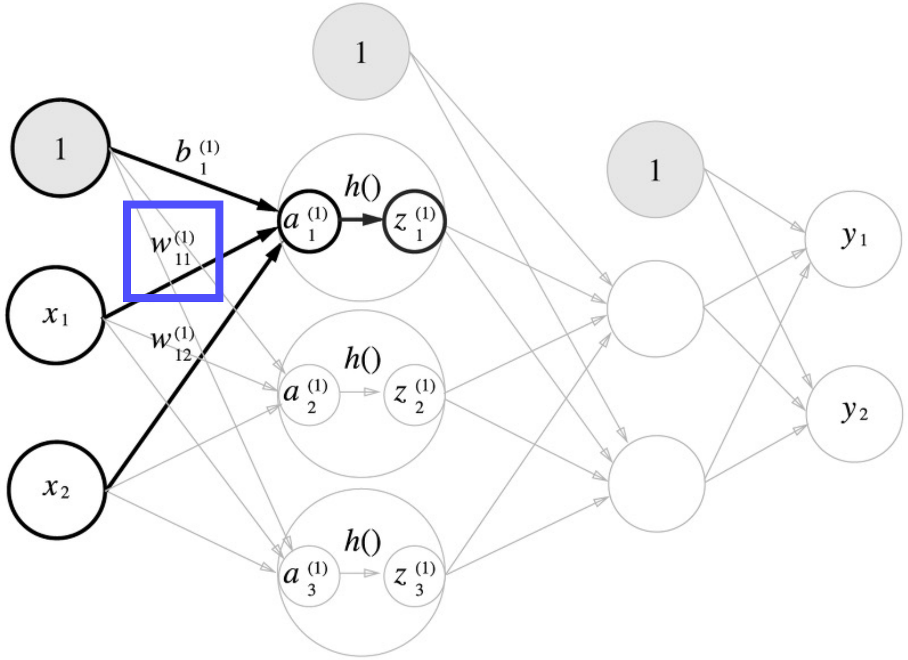
# 5. 오차역전파법

## 손실 함수 리뷰

- 가중치를 아주 조금 변화 시켰을 때 손실 함수가 어떻게 변하는지를 알아야 가중치를 조절할 수 있음!

우리가 알려고 하는것! $\frac{\partial f}{\partial w}$

$f$ = 손실함수

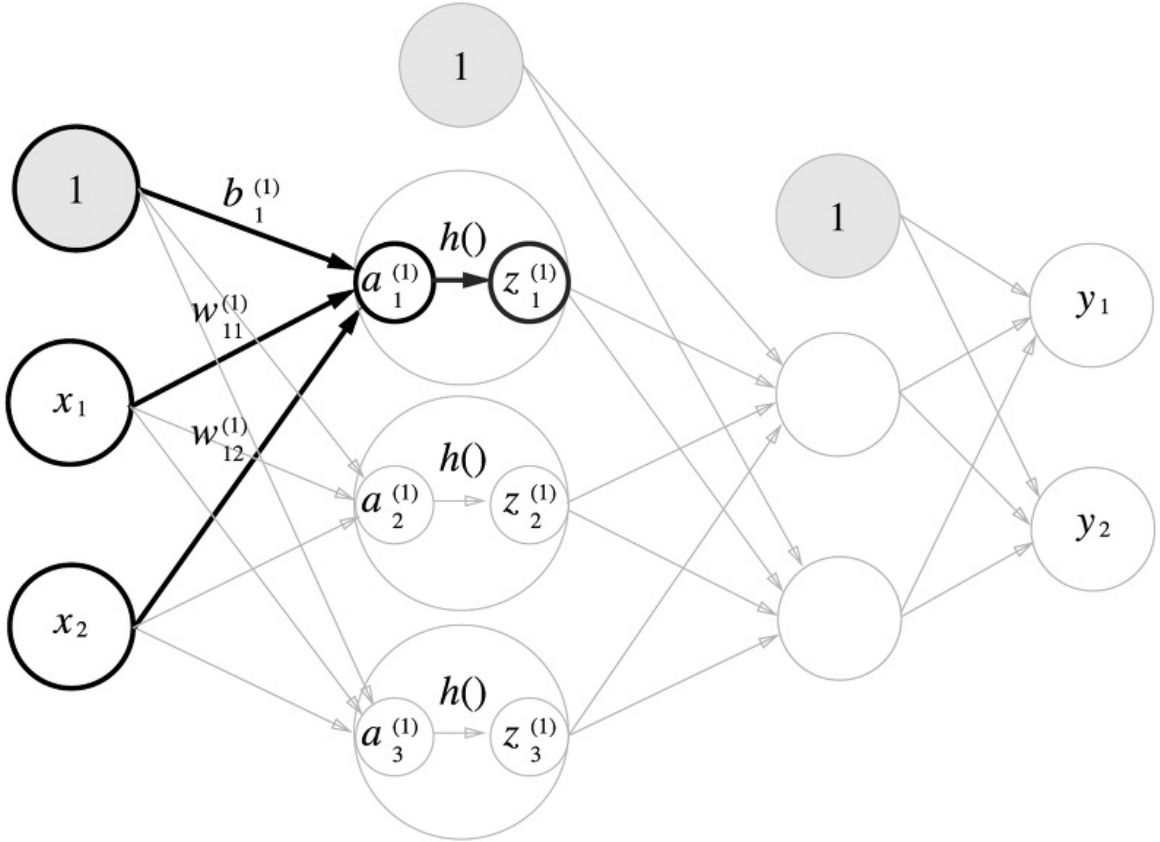



loss = 0.6774

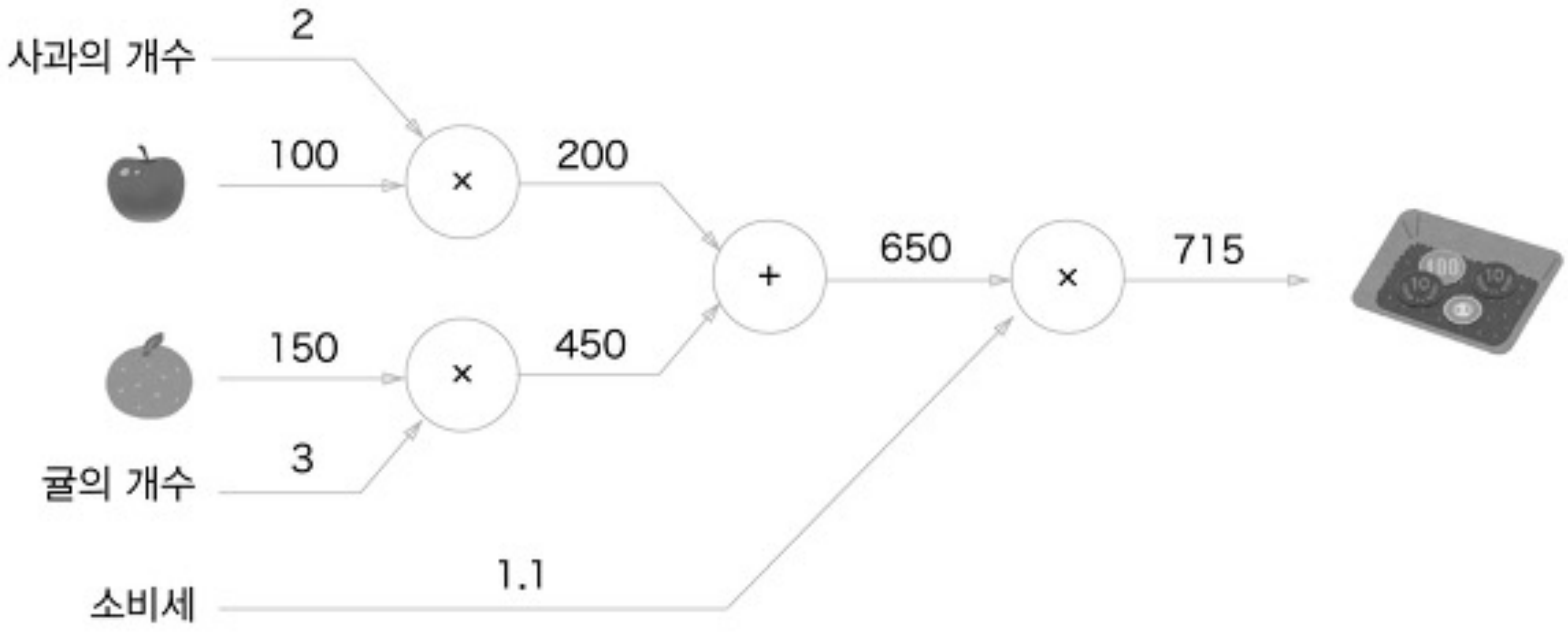
## 오차역전파법

- 가중치 매개변수의 기울기를 효율적으로 계산하는 법

- 제대로 이해하는 방법 두가지
  - **수식을 이용하는 법**
  - **계산 그래프를 통한 것**

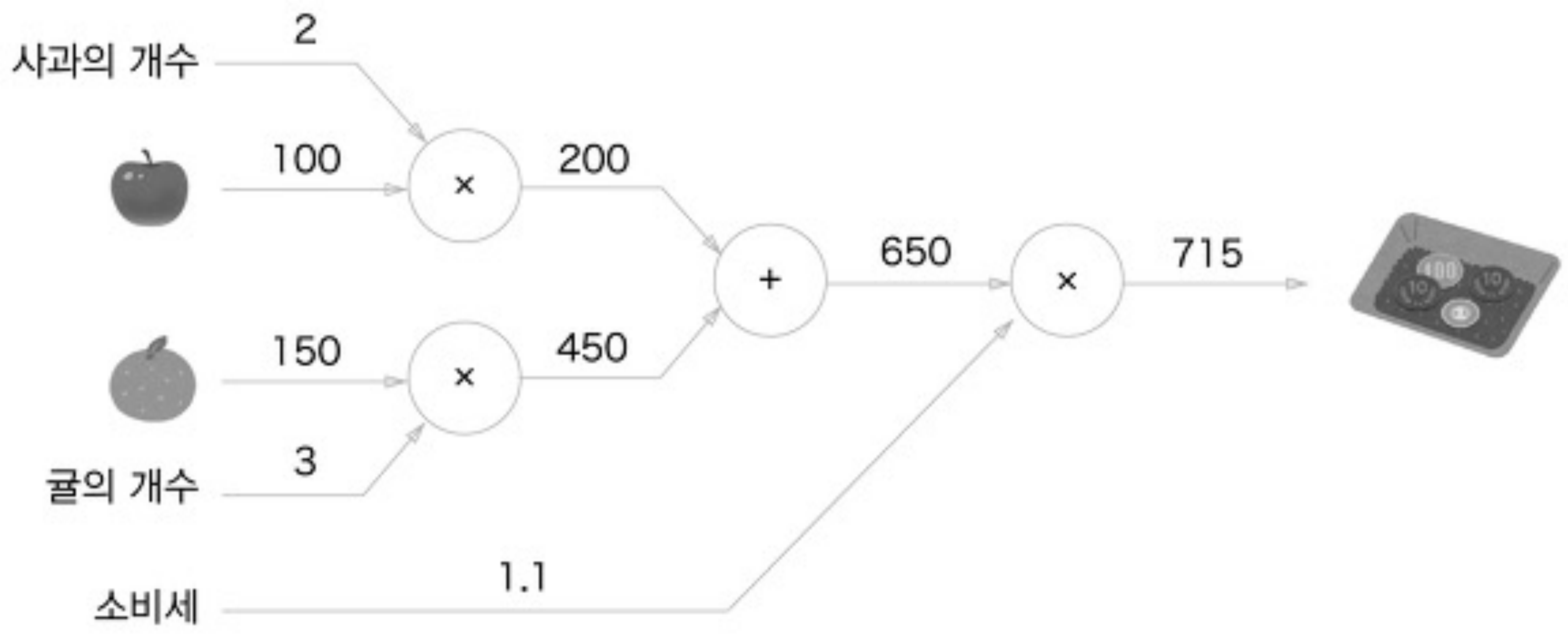


## 5.1 계산 그래프




Forward propagation 순전파

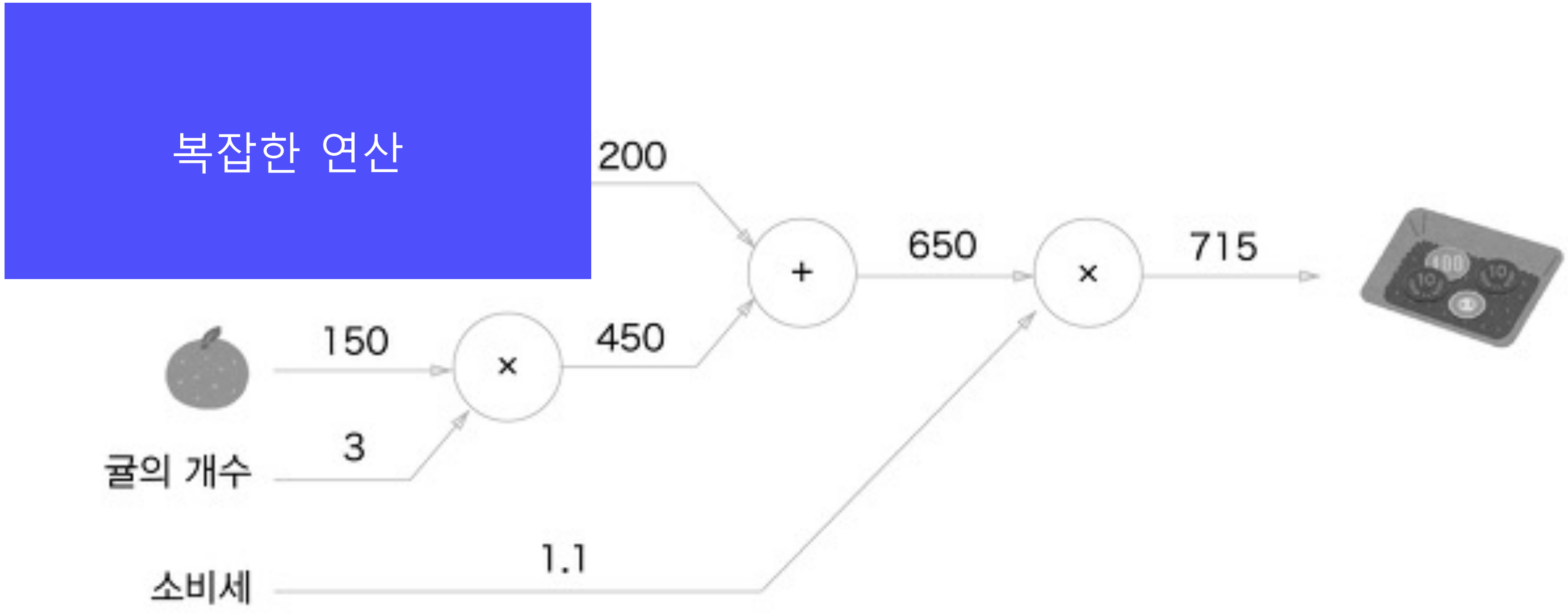
## 5.1 계산 그래프




Backward propagation 역전파

## 5.1.2 국소적 계산

- 자신과 직접 관계된 작은 범위만으로 결과를 출력할 수 있다.
    - 구하려는 곳과 관련이 없는 부분은 신경 쓰지 않아도 된다!



## 5.1.3 왜 계산 그래프로 푸는가?

1. 국소적으로 계산 가능 -> 문제 단순화 가능!
2. 계산 그래프는 중간 계산 결과를 모두 보관할 수 있음
   - 중간 단계별 미분 값을 구하기 편리함

사과값($x$)이 아주 조금 올랐을 때 전체 금액($L$)이 얼마나 증가하느냐

$$\frac{\partial L}{\partial x}$$

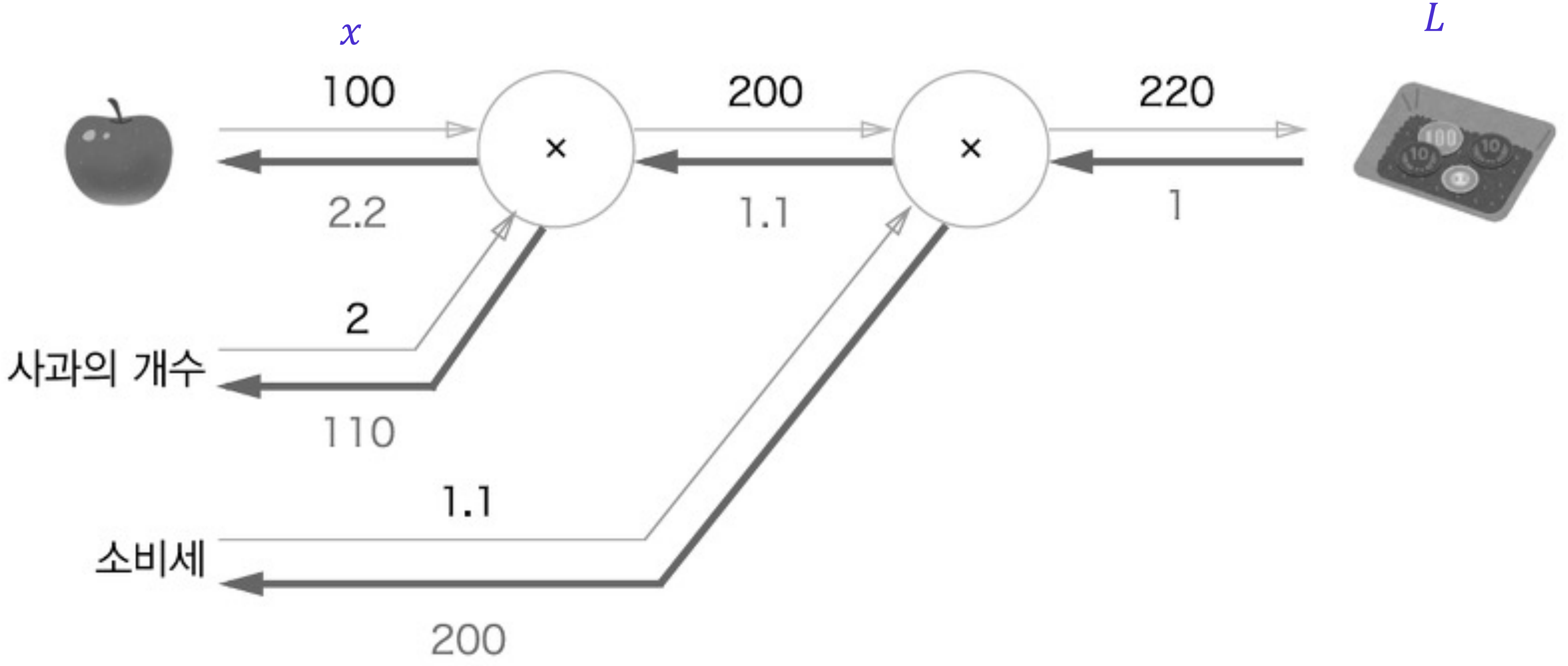
가중치값($w$)이 아주 조금 변화할 때 손실 함수($Loss$)이 얼마나 변화하느냐

$$\frac{\partial Loss}{\partial w}$$

## 5.1.3 왜 계산 그래프로 푸는가?

사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$

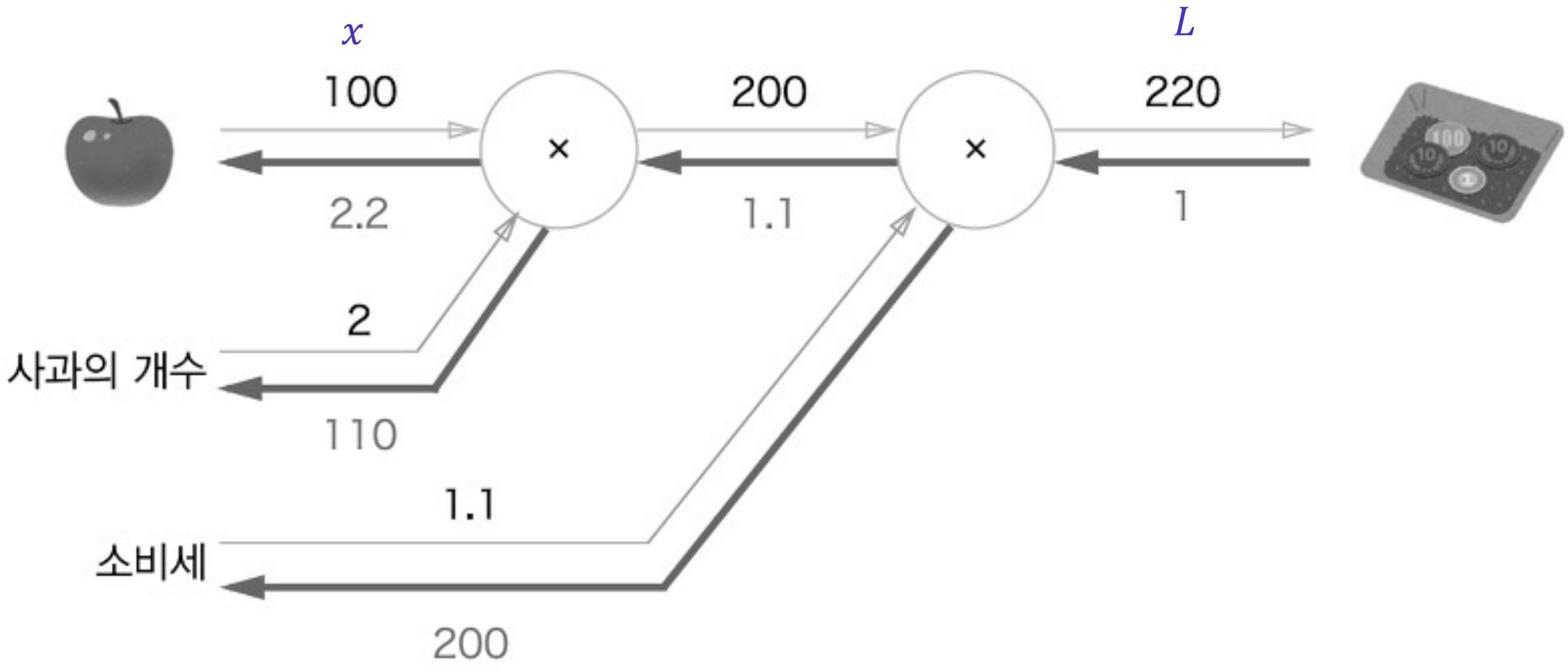



사과값($x$)이 아주 조금 올랐을 때 전체 금액($L$)이 얼마나

## 5.1.3 왜 계산 그래프로 푸는가?

사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$

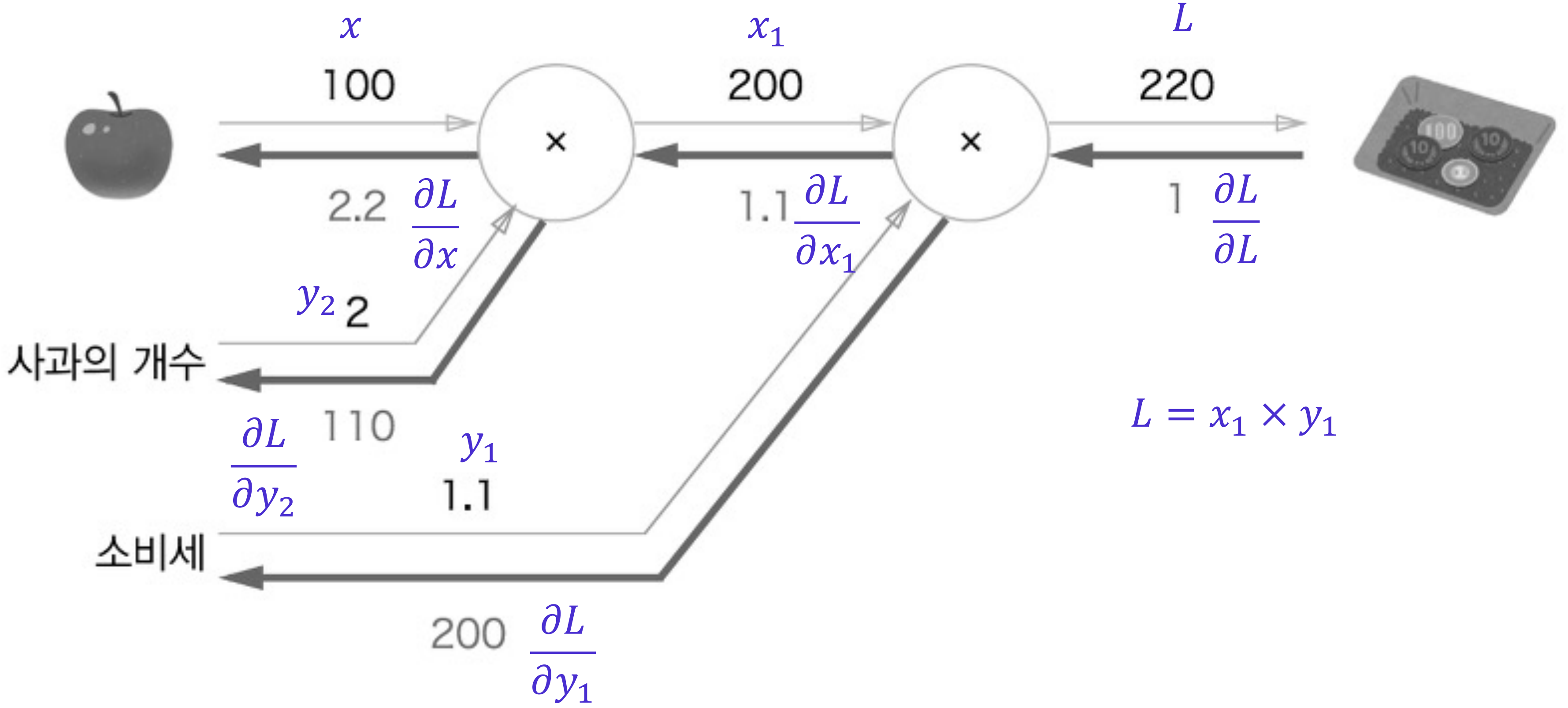



사과값($x$)이 아주 조금 올랐을 때 전체 금액($L$)이 얼마나

## 5.1.3 왜 계산 그래프로 푸는가?

사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$

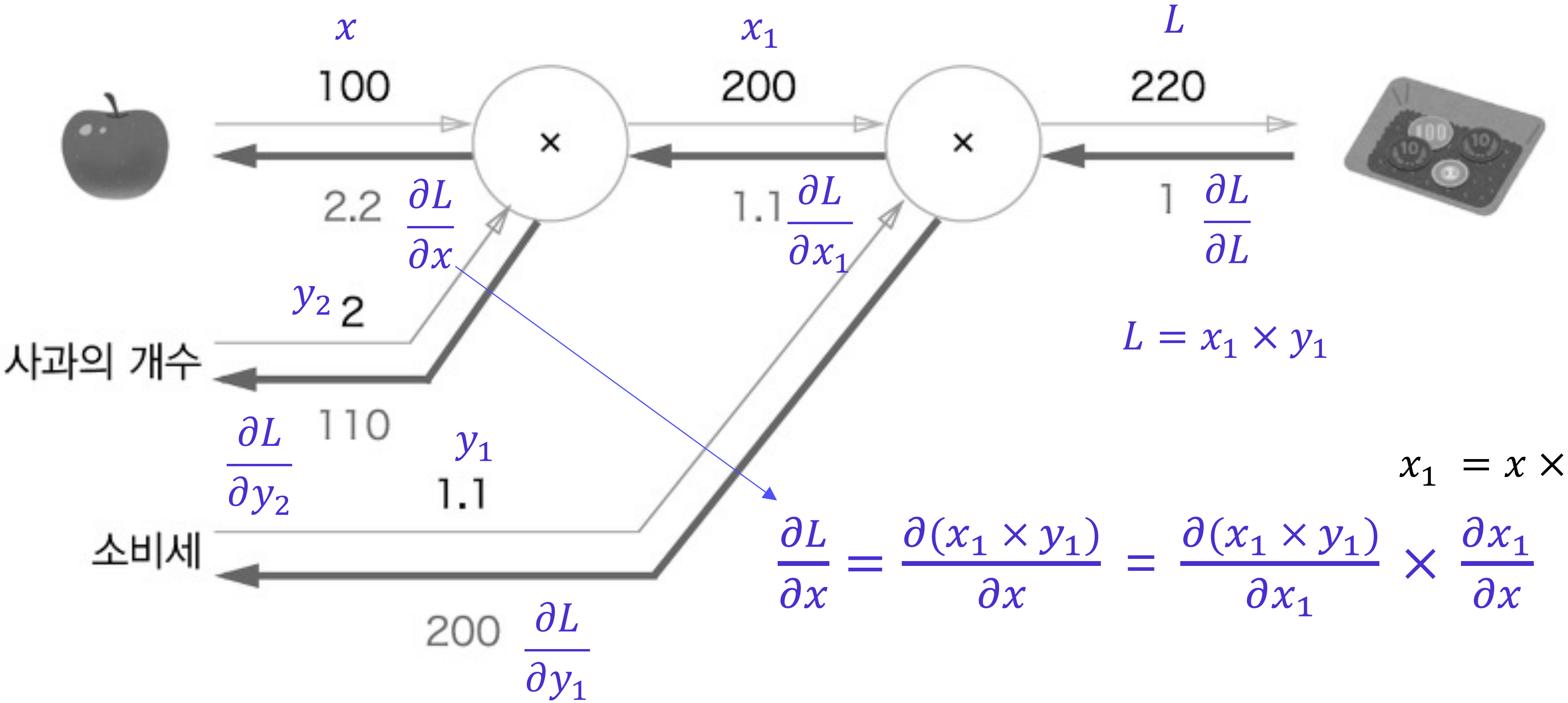



사과값$(x)$이 아주 조금 올랐을 때 전체 금액$(L)$이 얼마나

## 5.1.3 왜 계산 그래프로 푸는가?

사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$




$$x_1 = x \times y_2$$

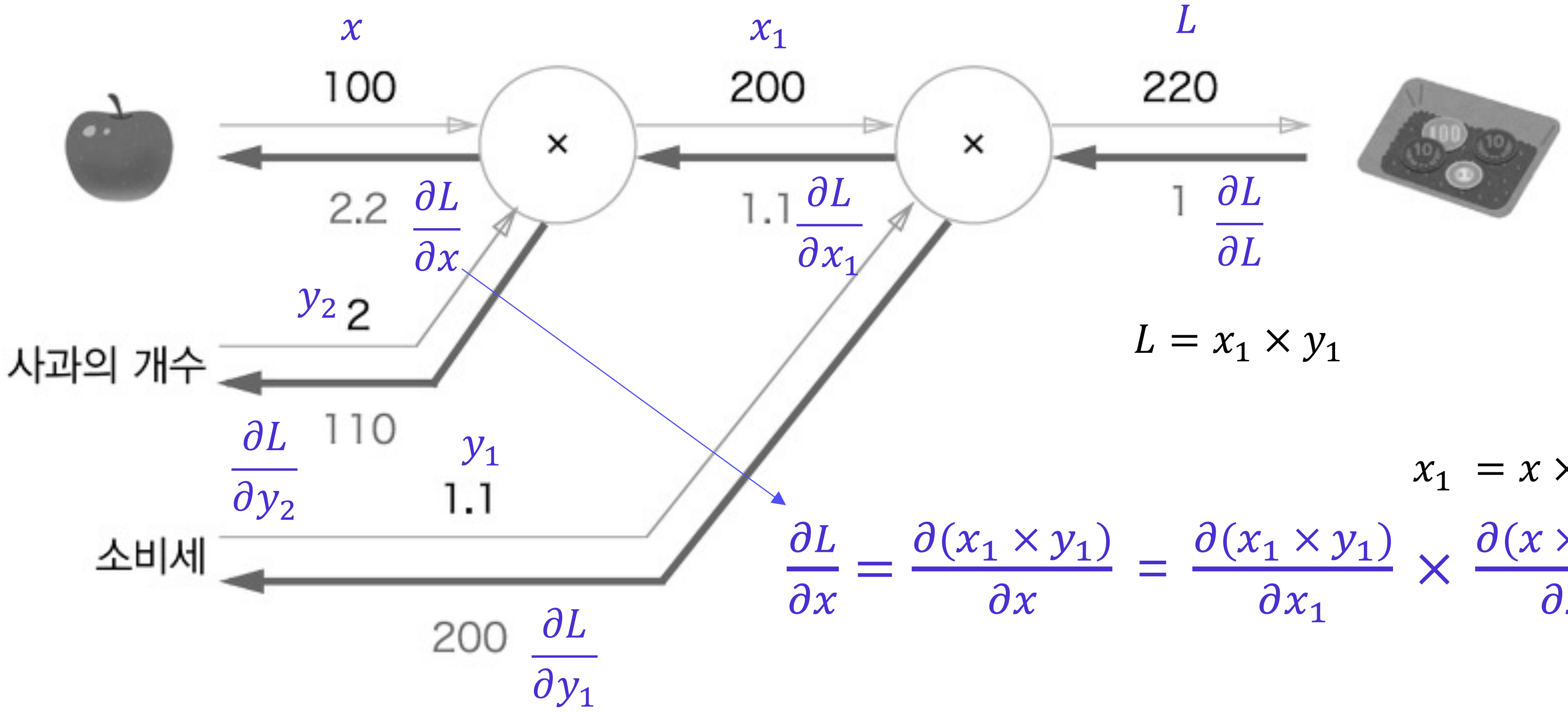
$$\frac{\partial L}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x_1} \times \frac{\partial x_1}{\partial x}$$

사과값($x$)이 아주 조금 올랐을 때 전체 금액($L$)이 얼마나

## 5.1.3 왜 계산 그래프로 푸는가?

### 사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$




$$L = x_1 \times y_1$$

$$x_1 = x \times y_2$$

$$\frac{\partial L}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x_1} \times \frac{\partial (x \times y_2)}{\partial x}$$

사과값($x$)이 아주 조금 올랐을 때 전체 금액($L$)이 얼마나

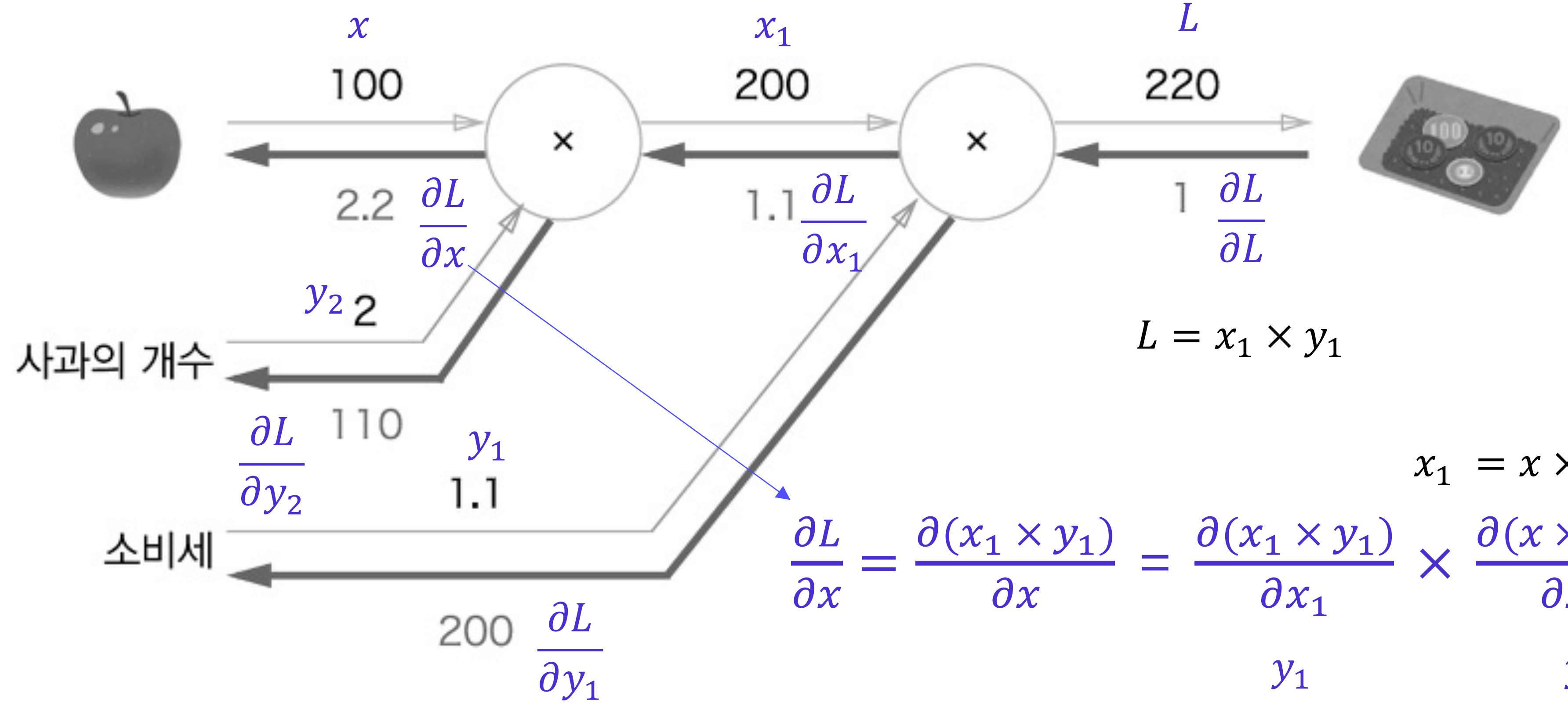
# 오차역전파법



## 5.1.3 왜 계산 그래프로 푸는가?

사과 그래프의 미분값 역전파

$$\frac{\partial L}{\partial x}$$




$$L = x_1 \times y_1$$

$$x_1 = x \times y_2$$

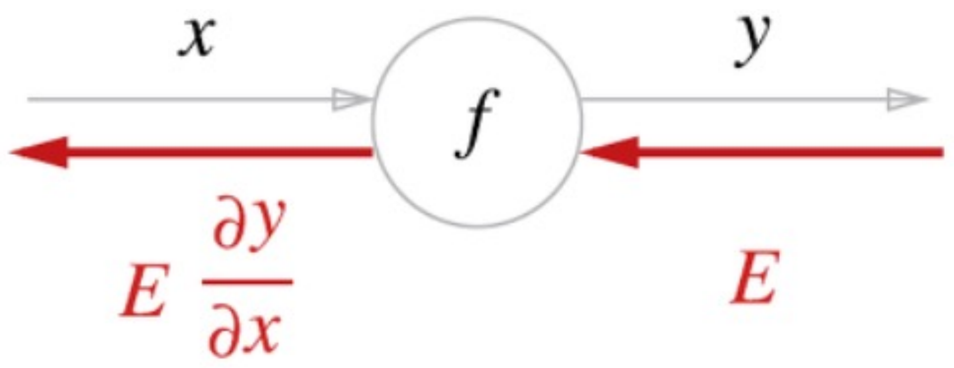
$$\frac{\partial L}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x} = \frac{\partial (x_1 \times y_1)}{\partial x_1} \times \frac{\partial (x \times y_2)}{\partial x}$$

$y_1$ $y_2$

사과값$(x)$이 아주 조금 올랐을 때 전체 금액$(L)$이 얼마나

## 5.2 연쇄법칙

### 5.2.1 계산 그래프의 역전파

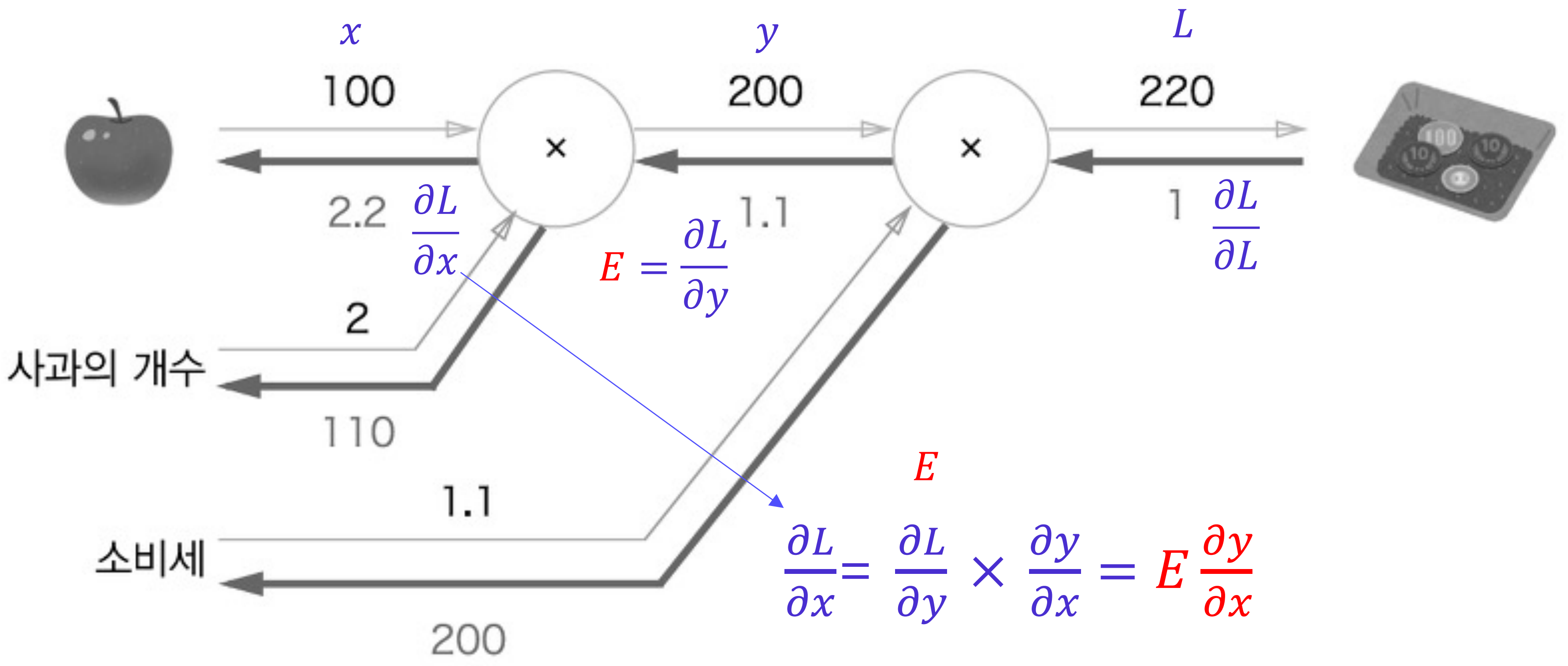



E(이전 step의 미분값)× $\frac{\partial y}{\partial x}$ (현재 지점에서의 기울기)

## 5.2 연쇄법칙

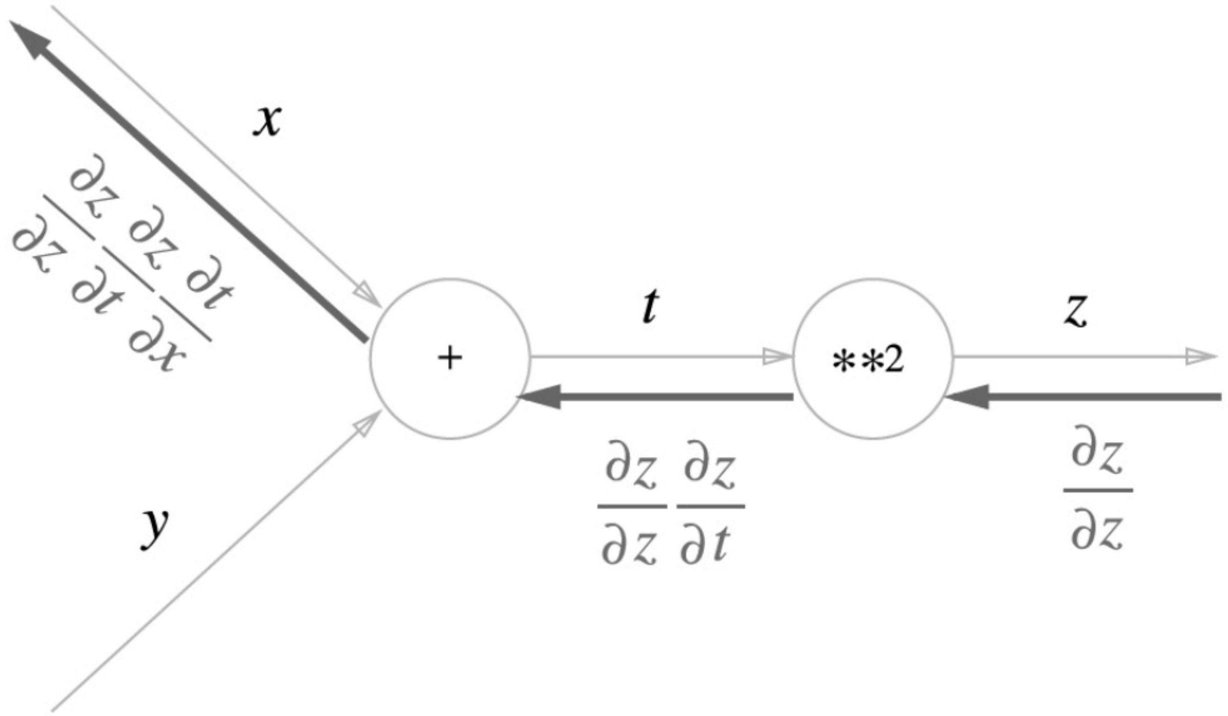
### 5.2.3 연쇄법칙과 계산 그래프

$$\frac{\partial L}{\partial x}$$



## 5.2 연쇄법칙

### 5.2.3 연쇄법칙과 계산 그래프

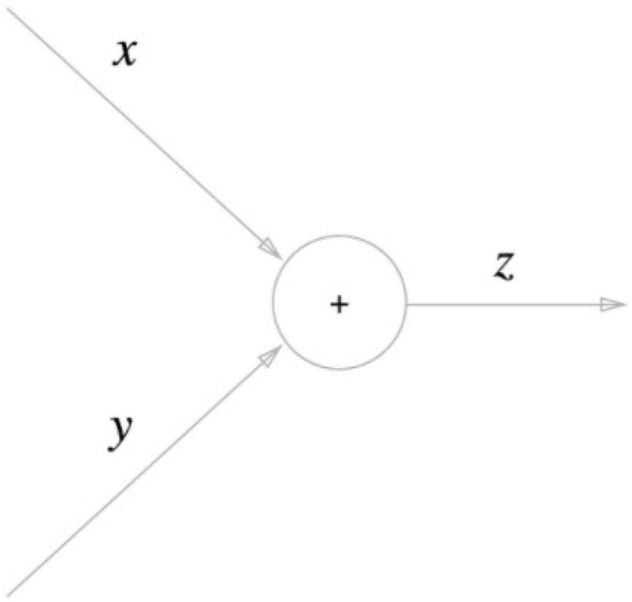


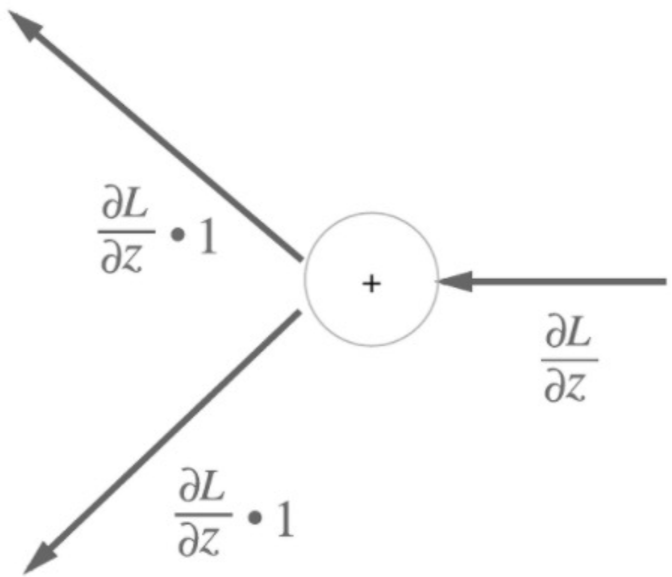
# 5.3 역전파

## - 5.3.1덧셈 노드의 역전파

$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1 \qquad \frac{\partial z}{\partial y} = 1$$



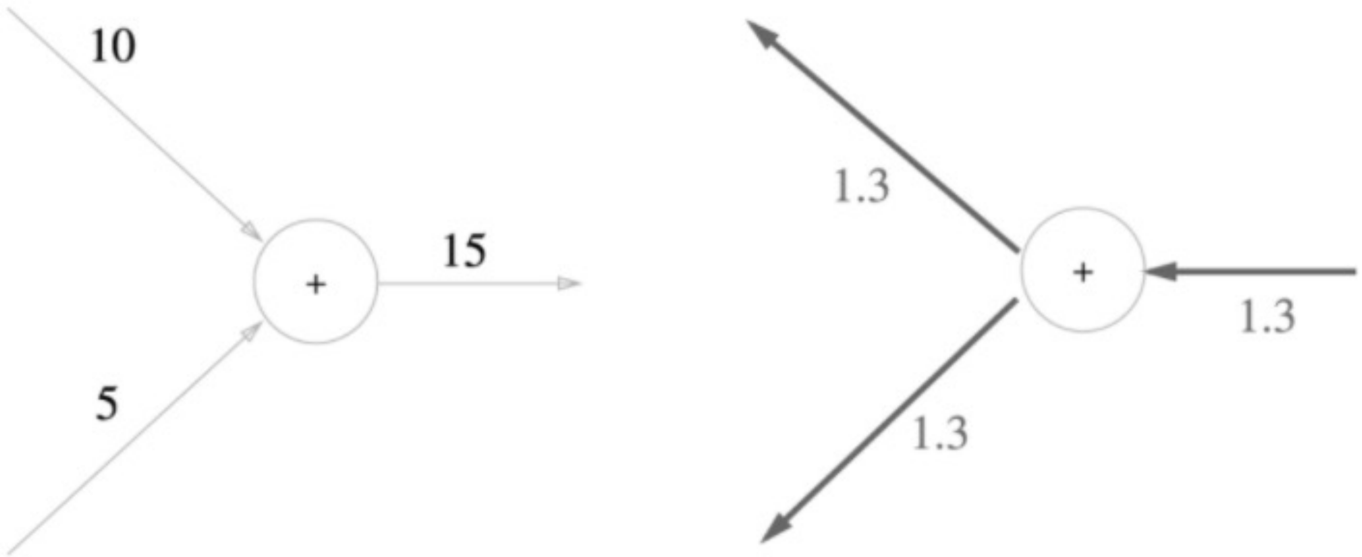

덧셈 노드의 역전파는 상류에서 전해진 미분에 1을 곱한다!

## 5.3 역전파

- 5.3.1덧셈 노드의 역전파

$$z = x + y$$

$$\frac{\partial z}{\partial x} = 1 \qquad \frac{\partial z}{\partial y} = 1$$



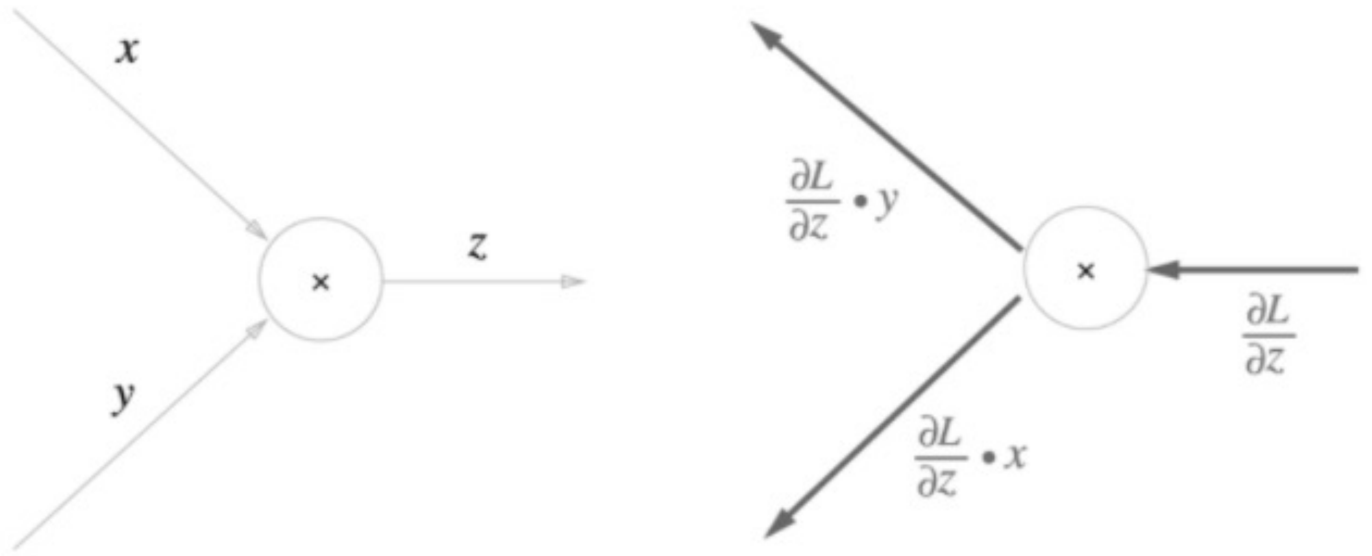

덧셈 노드의 역전파는 상류에서 전해진 미분에 1을 곱한다!

## 5.3 역전파

### - 5.3.2 곱셈 노드의 역전파

$$z = xy$$

$$\frac{\partial z}{\partial x} = y \qquad \frac{\partial z}{\partial y} = x$$



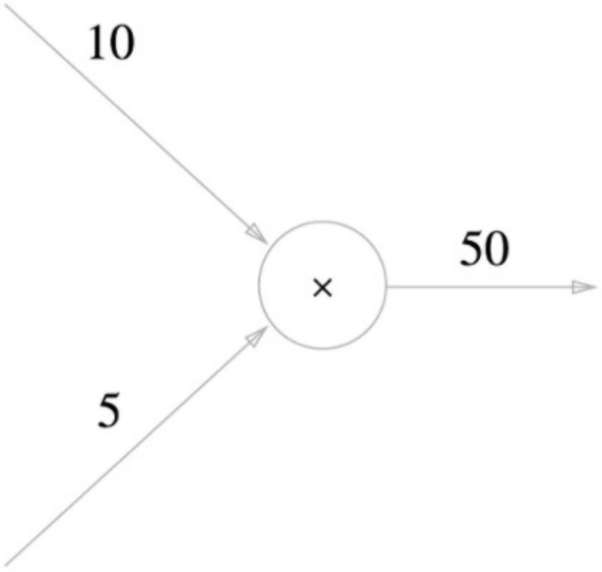

곱셈 노드의 역전파는 상류의 값에 순전파 때의 입력 신호들을 '서로 바꾼 값'을 곱한다.

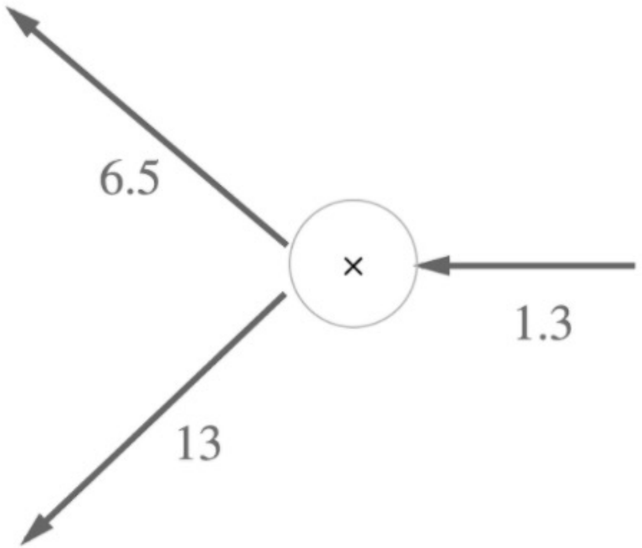
## 5.3 역전파

### - 5.3.2 곱셈 노드의 역전파

$$z = xy$$

$$\frac{\partial z}{\partial x} = y \qquad \frac{\partial z}{\partial y} = x$$

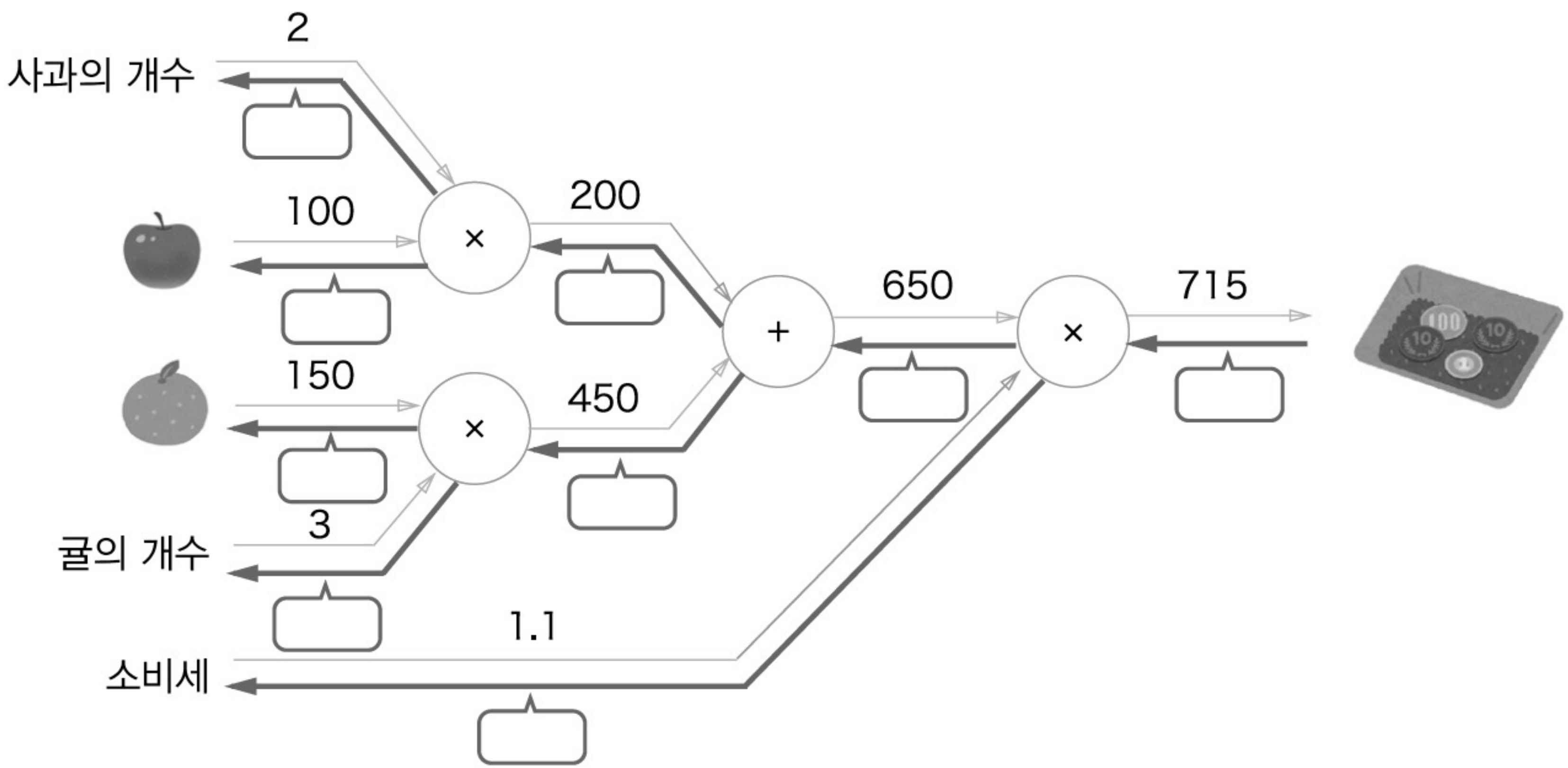



곱셈 노드의 역전파는 상류의 값에 순전파 때의 입력 신호들을 '서로 바꾼 값'을 곱한다.

## 5.3 역전파

- 5.3.3 사과 쇼핑의 예

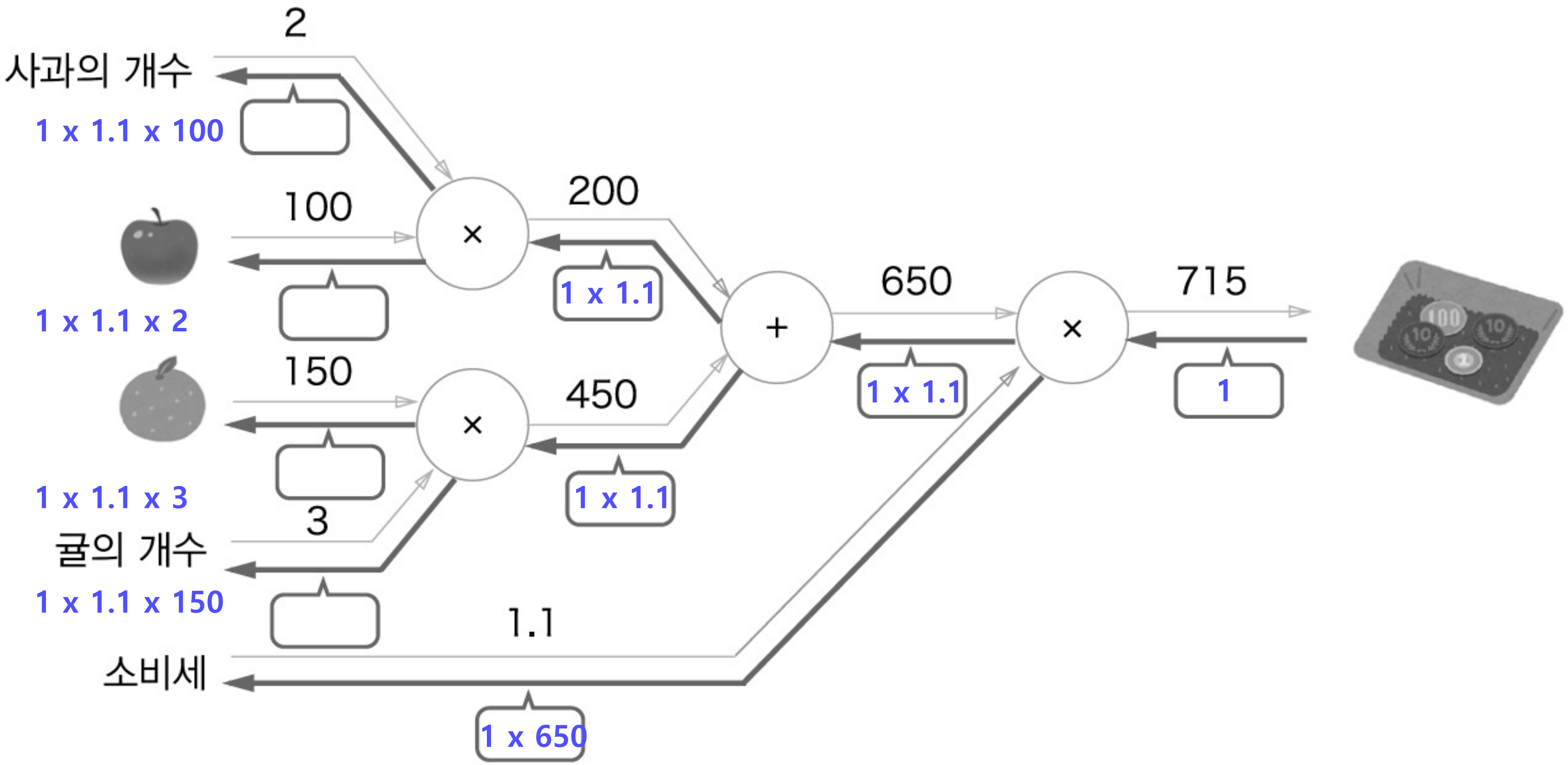


# 오차역전파법



## 5.3 역전파

- 5.3.3 사과 쇼핑의 예



## 5.4 단순한 계층 구현하기

### - 5.4.1 곱셈 계층

```
class MulLayer:
    def __init__(self):           # 변수 x와 y초기화
        self.x = None
        self.y = None

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

        return out

    def backward(self, dout): # dout은 상류에서 넘어온 미분(dout)
        dx = dout * self.y    # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

# 5.4 단순한 계층 구현하기

## - 5.4.2 덧셈 계층

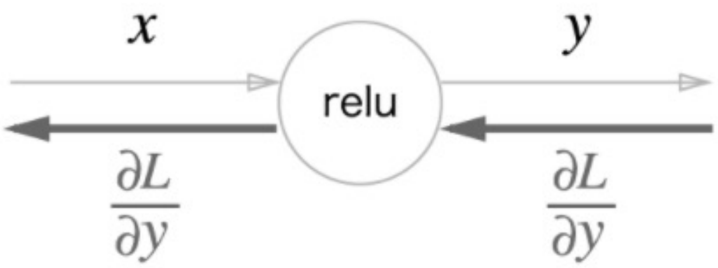
```
class AddLayer:
    def __init__(self):
        pass                    # 덧셈 계층에서는 초기화가 필요 없습
니다.

    def forward(self, x, y):
        out = x + y

        return out

    def backward(self, dout):
        dx = dout * 1           # 상류에서 내려온 미분을 그대로 하류
로 흘립니다.
        dy = dout * 1

        return dx, dy
```

## 5.5 활성화 함수 계층 구현하기

### - 5.5.1 ReLU 계층

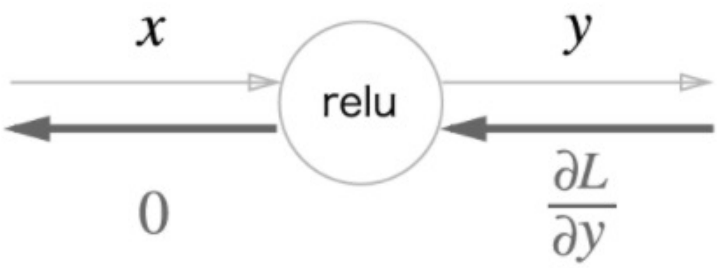
$$y = \begin{cases} x & (x > 0) \\ 0 & (x \leq 0) \end{cases} \qquad \frac{\partial y}{\partial x} = \begin{cases} 1 & (x > 0) \\ 0 & (x \leq 0) \end{cases}$$

$x > 0$




그대로 (덧셈 노드와 같음)

$x \leq 0$

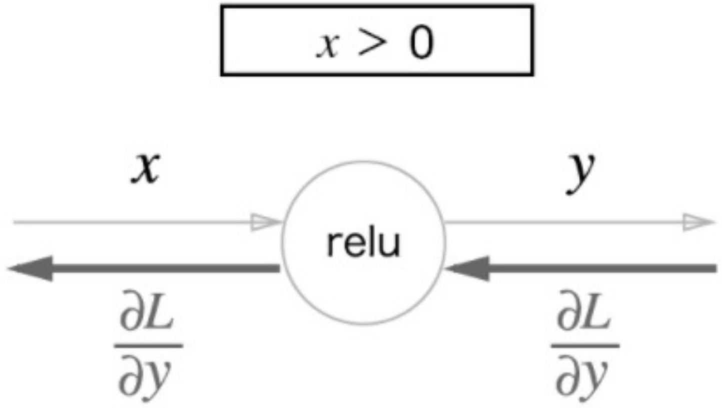



신호를 하류로 내보내지 않음

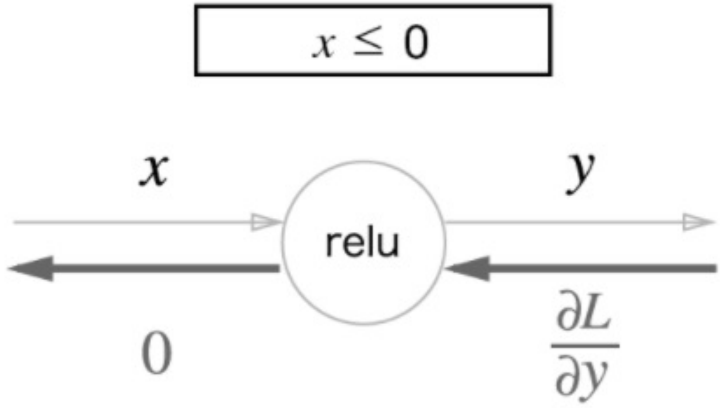
# 5.5 활성화 함수 계층 구현하기

## - 5.5.1 ReLU 계층

**순전파 때 신호가 흘렀으면 on 없었으면 off하는 스위치와 같다.**




순전파




신호를 하류로 내보내지 않음

# 5.5 활성화 함수 계층 구현하기

- 5.5.1 ReLU 계층

```
class Relu:
    def __init__(self):
        self.mask = None                 # mask 인스턴트 변수
                                         # mask는 True/False로 구성된
넘파이 배열로,
                                         # 순전파의 입력인 x의 원소 값이 0
이하인 인덱스는 True,
                                         # 그 외(0보다 큰 원소)는 False
로 유지합니다.

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()                   # x를 복사
        out[self.mask] = 0               # mask = True 인 인덱스를 0 으
로 바꾸기

        return out

    def backward(self, dout):
        dout[self.mask] = 0              # mask = True 인 인덱스를 0 으
로 바꾸기
        dx = dout

        return dx
```

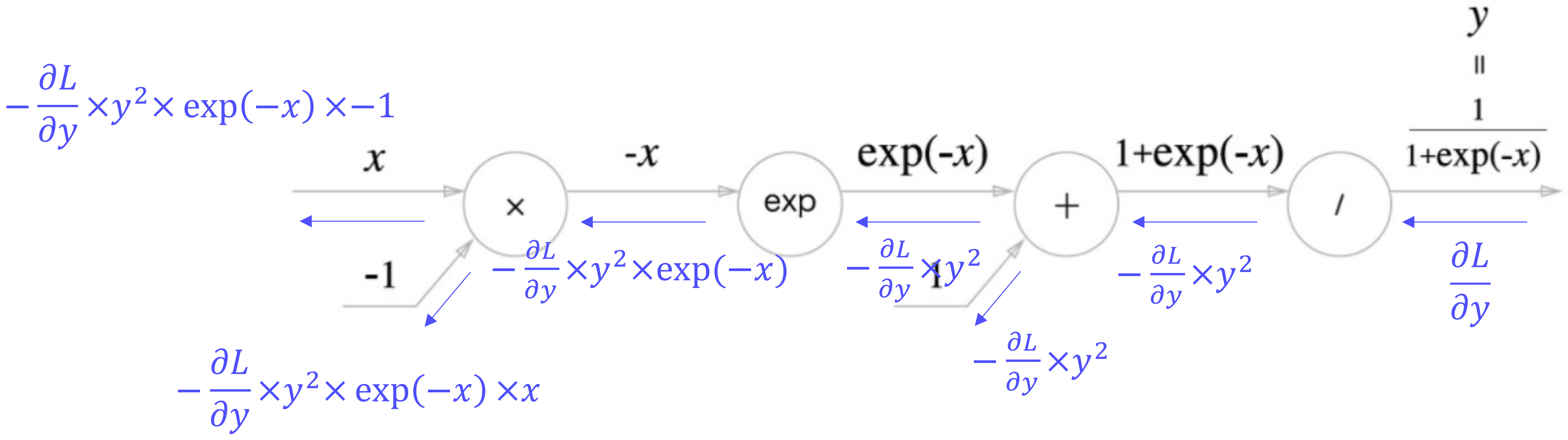
## 5.5 활성화 함수 계층 구현하기

- 5.5.2 Sigmoid 계층

$$y = \exp(x)$$

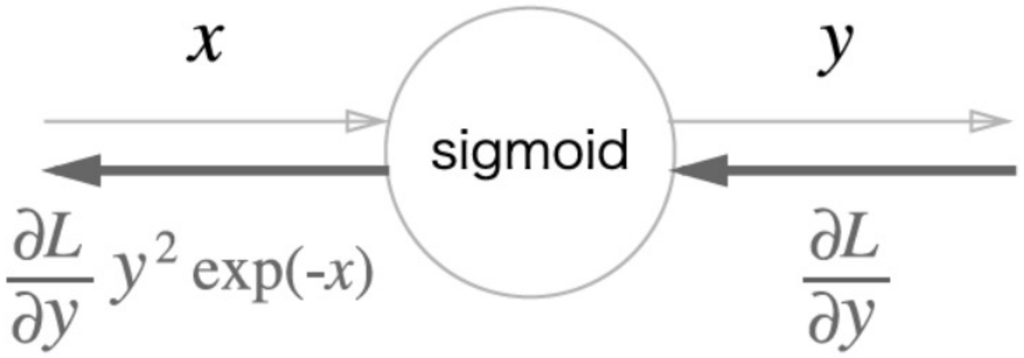
$$\frac{\partial y}{\partial x} = \exp(x)$$

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



## 5.5 활성화 함수 계층 구현하기

- 5.5.2 Sigmoid 계층



## 5.5 활성화 함수 계층 구현하기

- 5.5.2 Sigmoid 계층

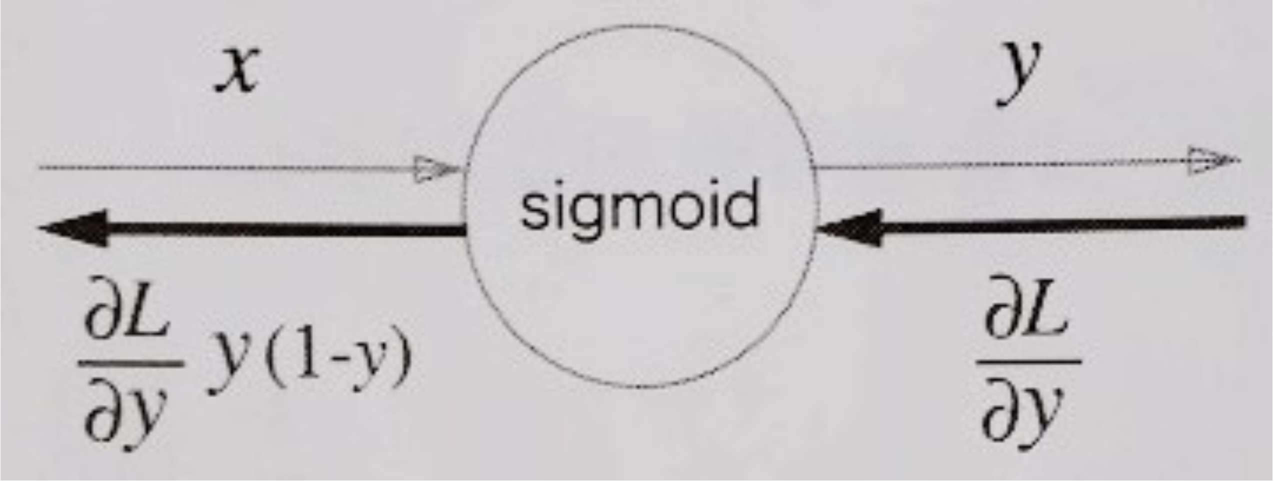
$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+\exp(-x))^2} \exp(-x)$$

$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)}$$

$$= \frac{\partial L}{\partial y} y(1-y)$$

## 5.5 활성화 함수 계층 구현하기

- 5.5.2 Sigmoid 계층



# 5.5 활성화 함수 계층 구현하기

## - 5.5.2 Sigmoid 계층

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

    def forward(self, x):
        out = 1 / (1 + np.exp(-x))
        self.out = out

        return out

# 순전파의 출력을 인스턴스 변수 out에 보관했다가, 역전파 계산 때 그 값을 사용
합니다.
    def backword(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

# 5.6 Affine/Softmax 계층 구현하기

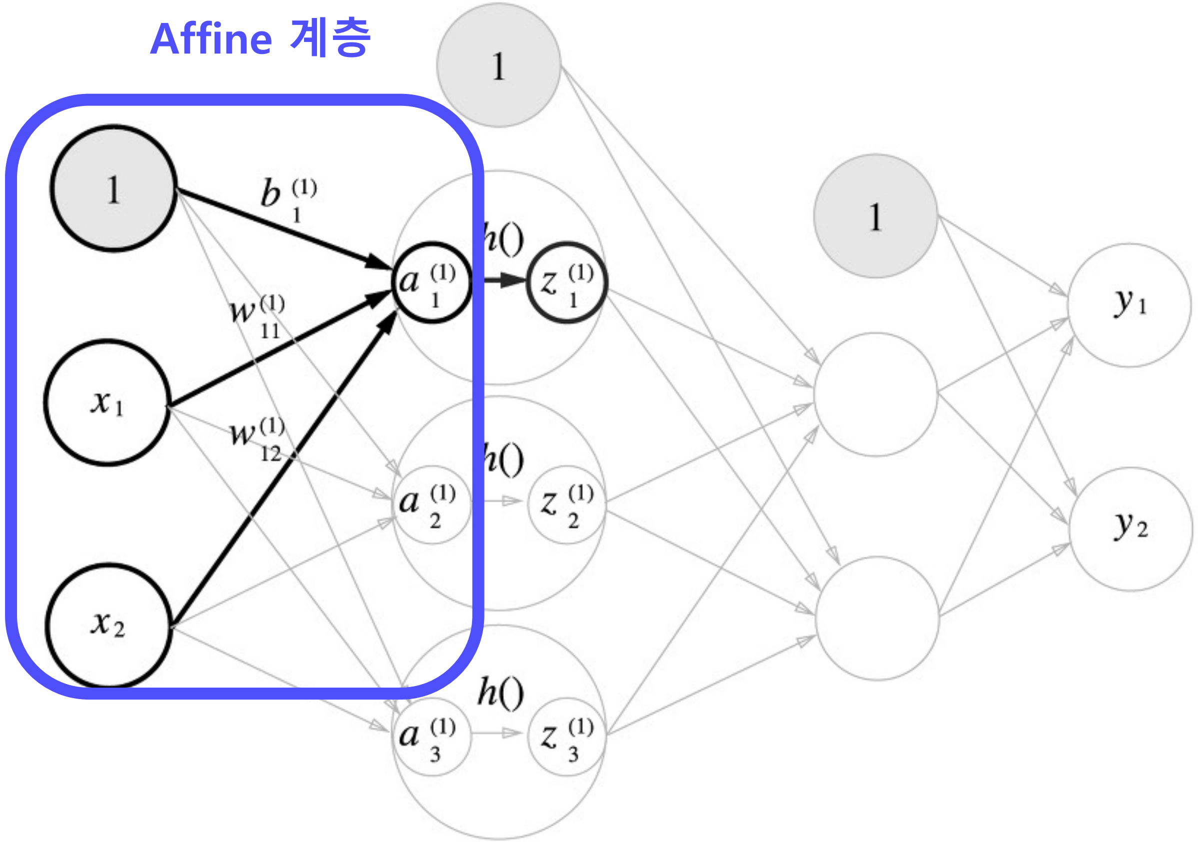
## - 5.6.1 Affine 계층

$x_1 w_{11} + x_2 w_{12} + b_1 = a_1$

$x_1 w_{21} + x_2 w_{22} + b_2 = a_2$

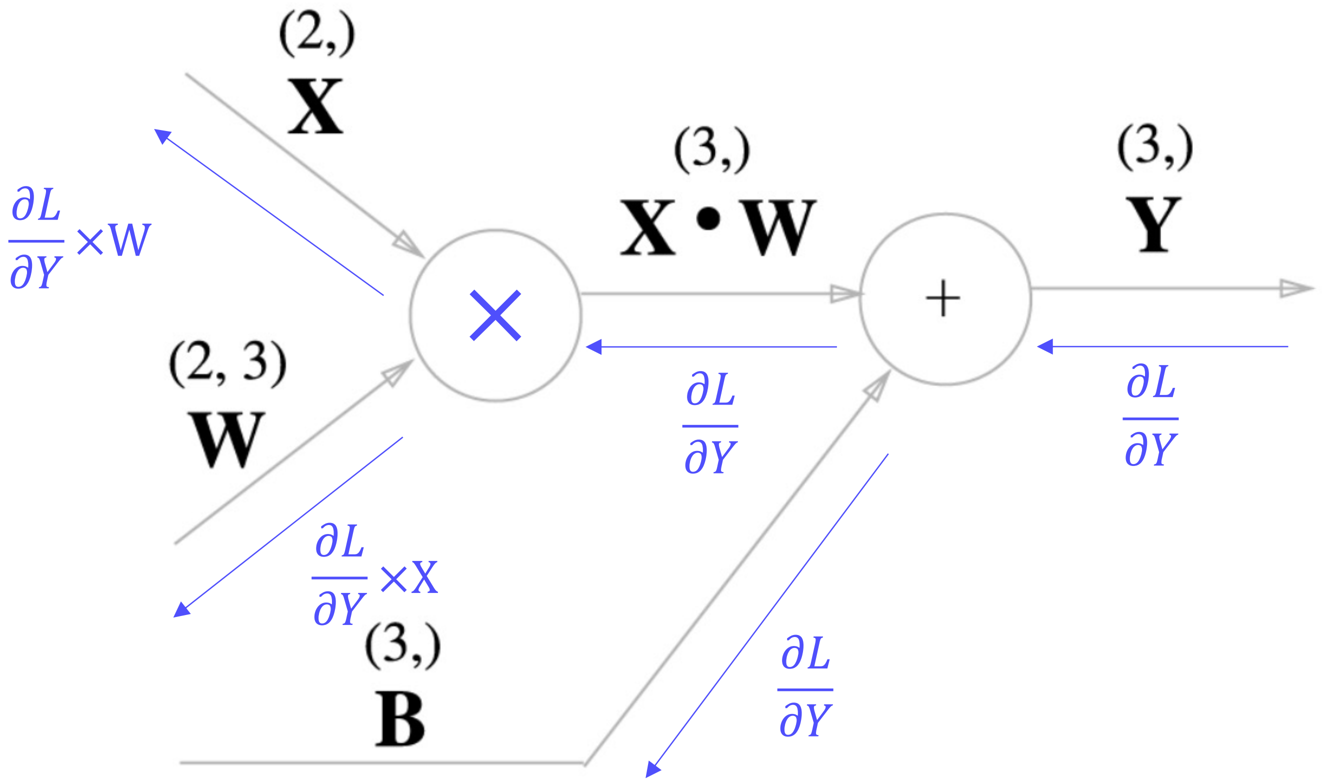
$x_1 w_{31} + x_2 w_{32} + b_3 = a_3$

$XW + B = Affine()$



## 5.6 Affine/Softmax 계층 구현하기

- 5.6.1 Affine 계층



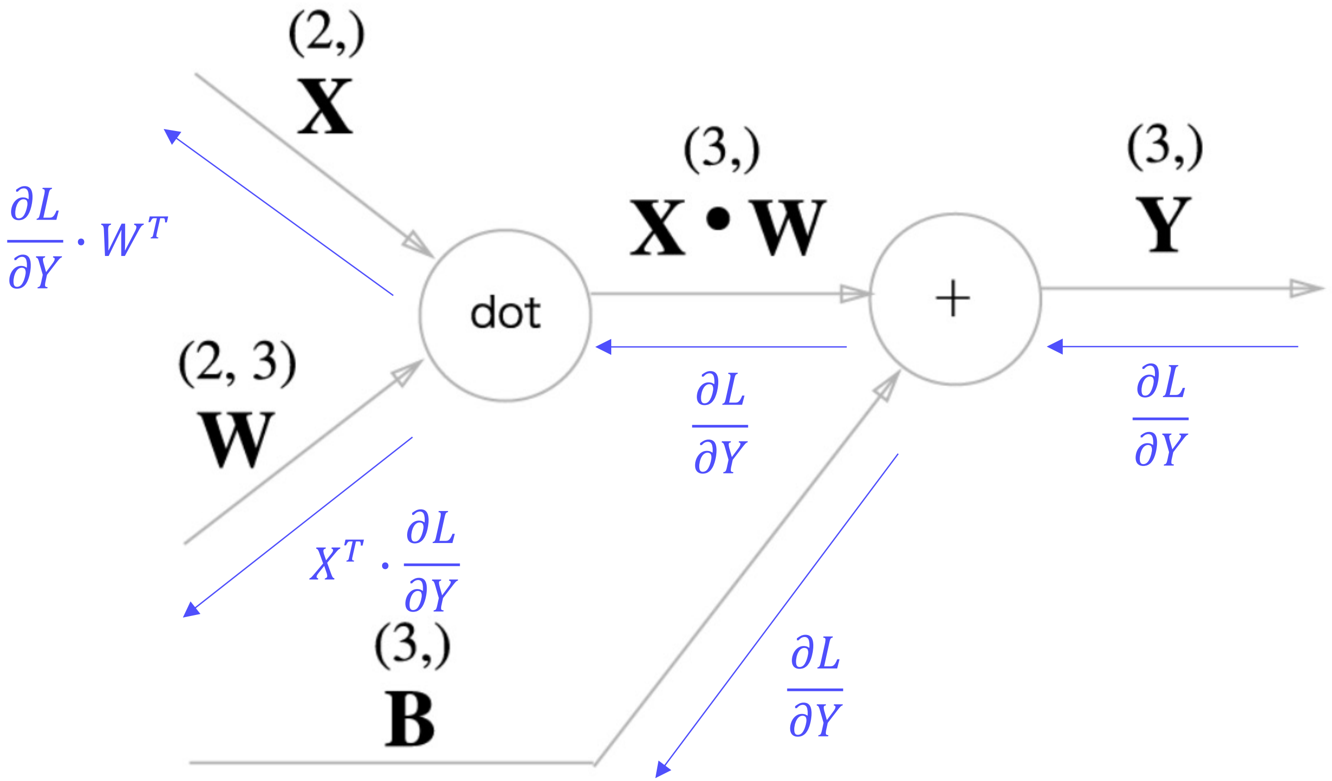
## 5.6 Affine/Softmax 계층 구현하기

- 5.6.1 Affine 계층

$$Y = XW$$

$$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

$$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$



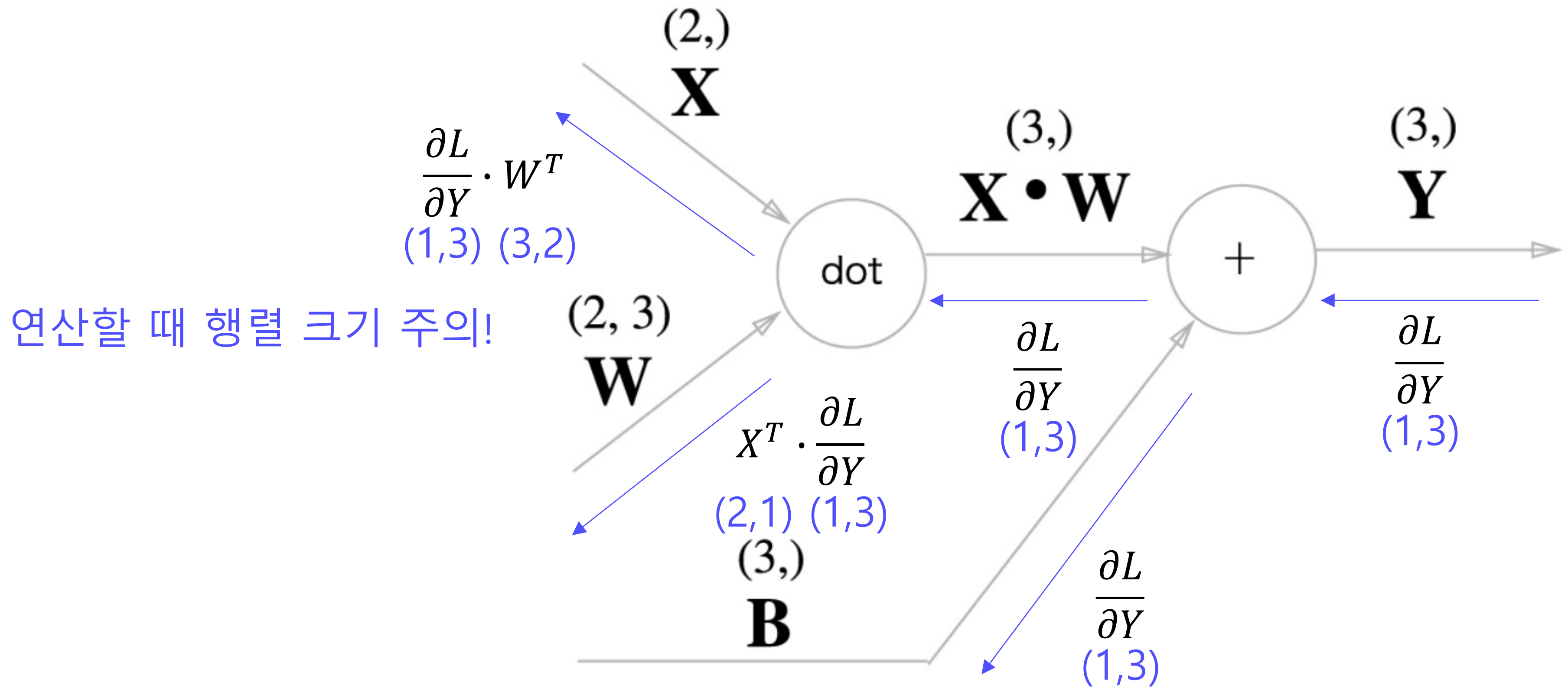

참고

$$W = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix}$$

$$W^T = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix}$$

# 5.6 Affine/Softmax 계층 구현하기

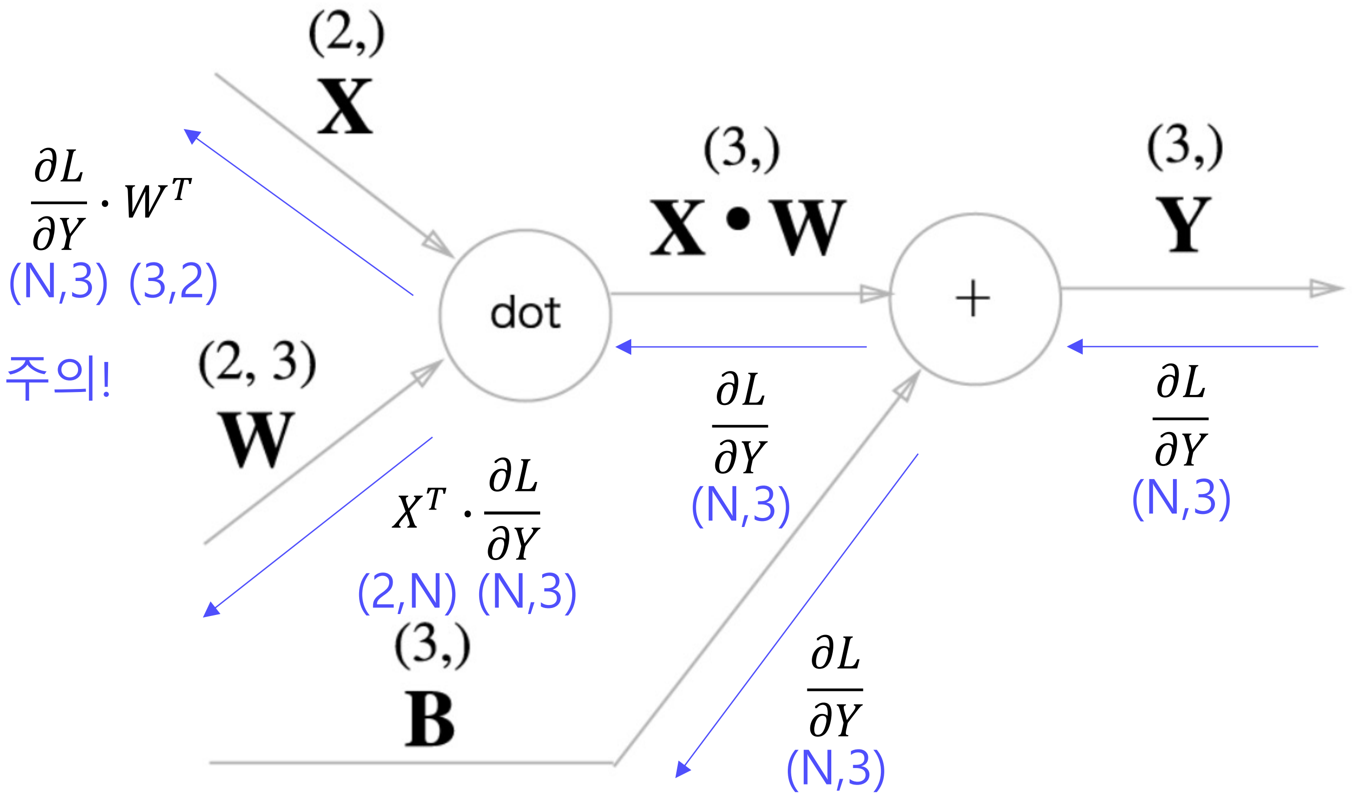
## - 5.6.1 Affine 계층



# 5.6 Affine/Softmax 계층 구현하기

## - 5.6.2 배치용 Affine 계층

연산할 때 행렬 크기 주의!

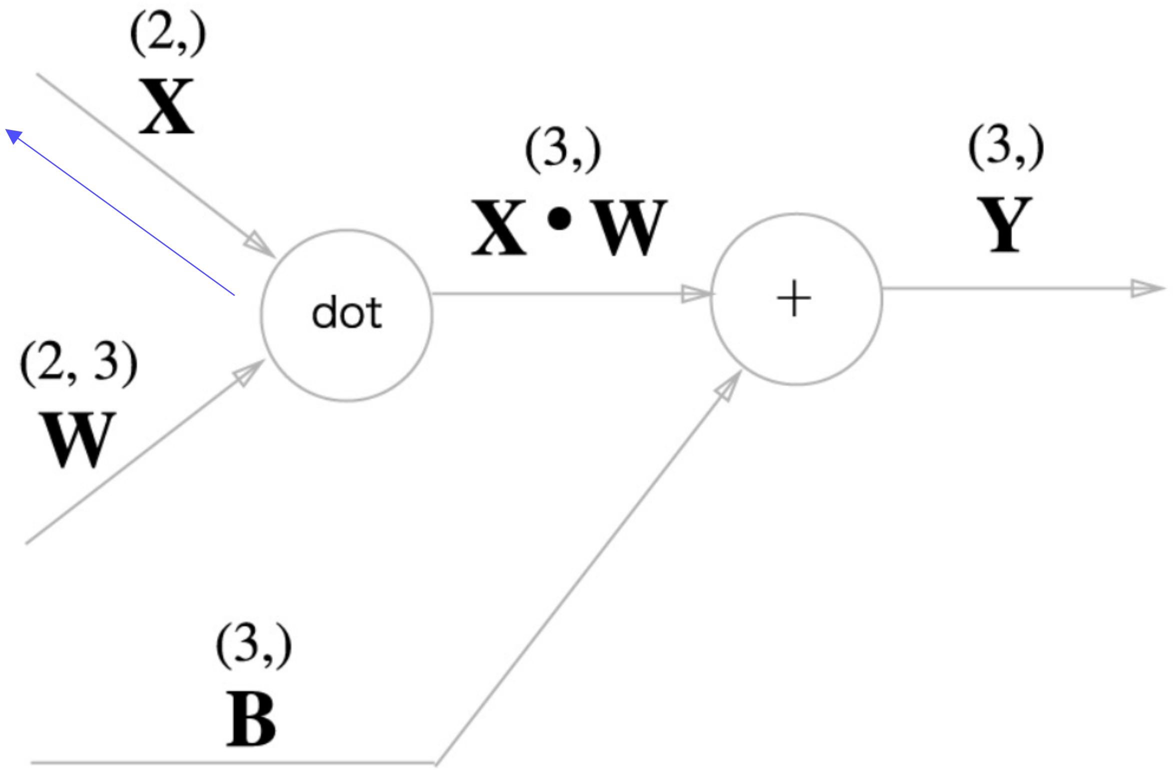


# 5.6 Affine/Softmax 계층 구현하기

## - 5.6.2 배치용 Affine 계층

$$\mathbf{X} = (x_0, x_1, \cdots, x_n)$$

$$\frac{\partial L}{\partial \mathbf{X}} = \left( \frac{\partial L}{\partial x_0}, \frac{\partial L}{\partial x_1}, \cdots, \frac{\partial L}{\partial x_n} \right)$$



# 5.6 Affine/Softmax 계층 구현하기

## - 5.6.2 배치용 Affine 계층

$$
\begin{cases}
\dfrac{\partial L}{\partial X} = \dfrac{\partial L}{\partial Y} \cdot W^T \\[2ex]
\dfrac{\partial L}{\partial W} = X^T \cdot \dfrac{\partial L}{\partial Y} \\[2ex]
\dfrac{\partial L}{\partial B} = \dfrac{\partial L}{\partial Y}
\end{cases}
$$

```
class Affine:
    def __init__(self, W, b):              # 매개변수 초기화
        self.W = W
        self.b = b

        self.x = None
        self.original_x_shape = None       # 입력 데이터가 텐서(4차원
데이터)인 경우도 고려

        # 가중치와 편향 매개변수의 미분
        self.dW = None
        self.db = None

    def forward(self, x):

        # 텐서 대응(4차원 데이터)
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)      # -1의 의미는 원래 배열의
길이와 남은 차원으로 부터 추정입니다.
                                           # 요소가 12개일때
shape(3,-1) 은 (3 X 4) 행렬이 됩니다.
        self.x = x

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

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)          # (3)
        self.dW = np.dot(self.x.T, dout)     # (2)
        self.db = np.sum(dout, axis=0)       # (1)

        dx = dx.reshape(*self.original_x_shape)  # 기존의 입력
데이터 모양으로 변경(텐서 대응)
        return dx
```

## 5.6 Affine/Softmax 계층 구현하기

- 5.6.3 Softmax-with-Loss 계층

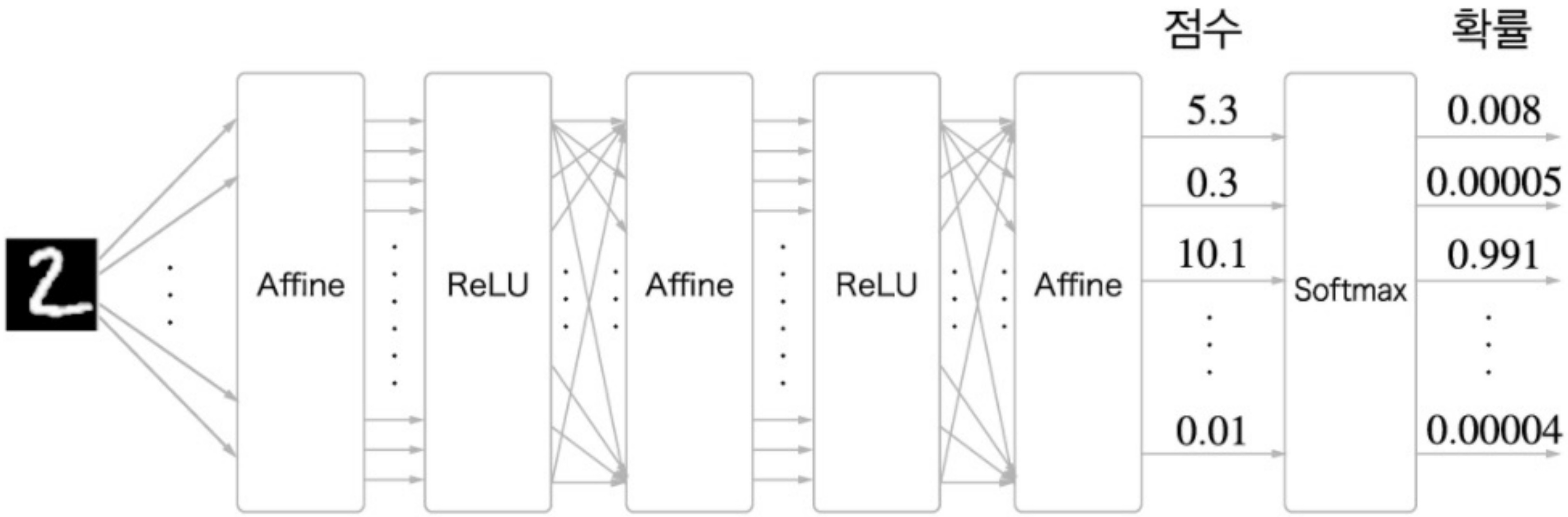
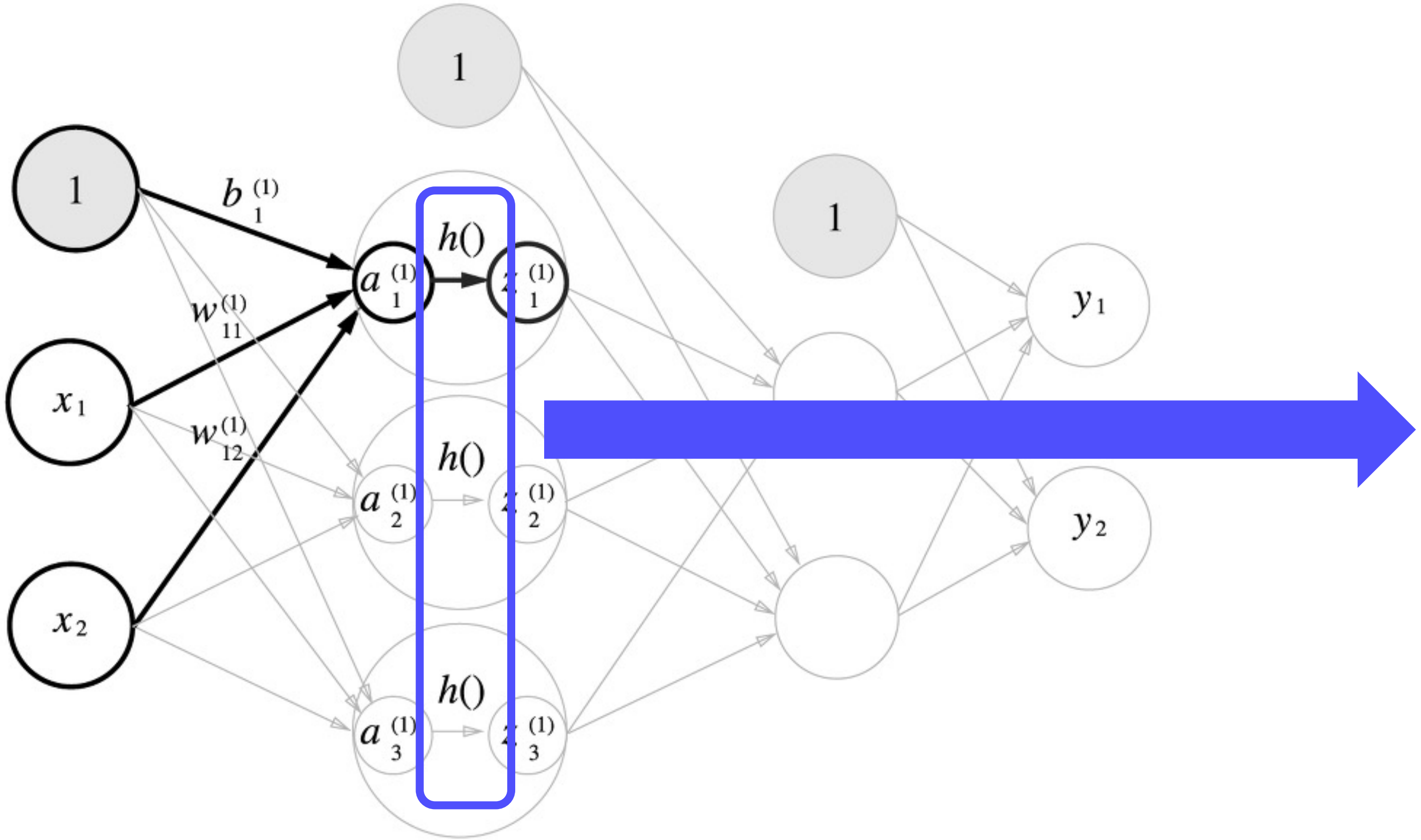



Image Classification Forward

# 신경망



## 3층 신경망 구현하기

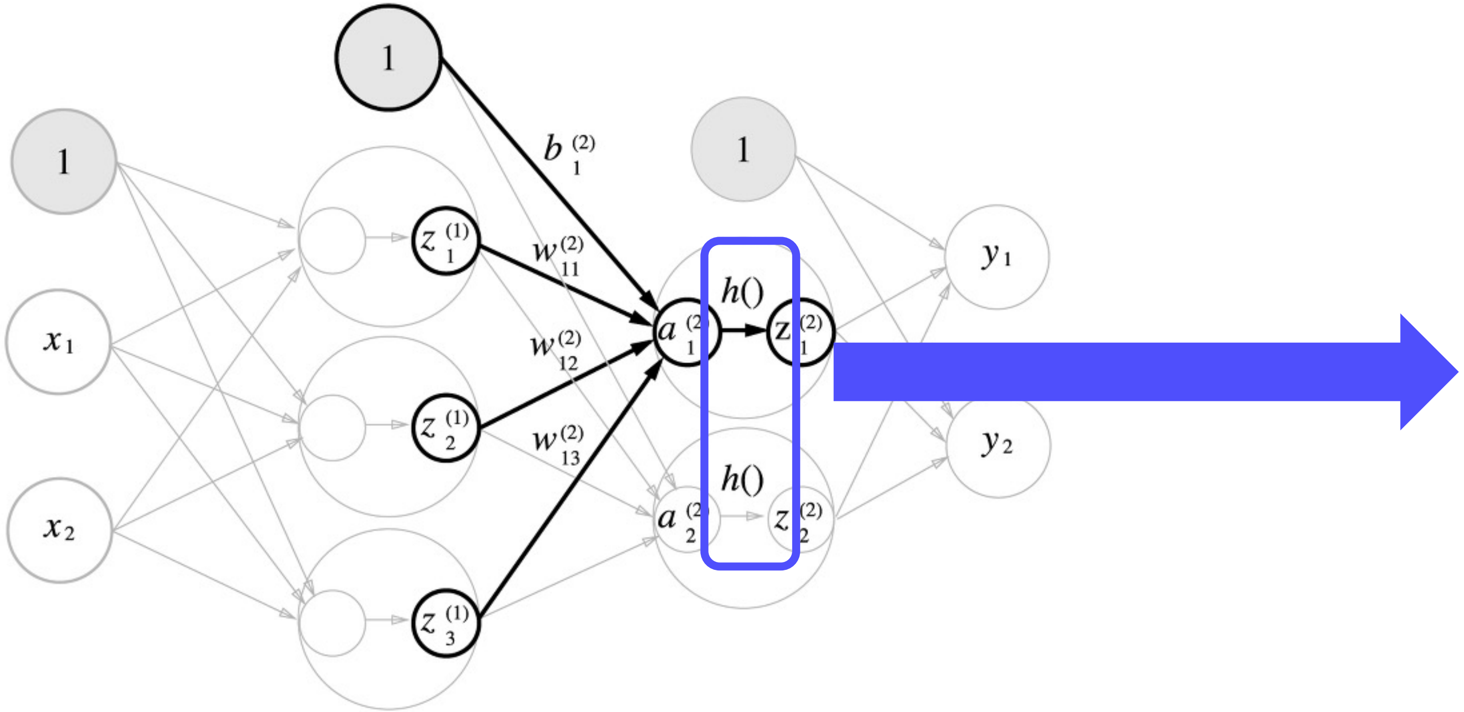



**relu()**
**sigmoid()**
**tanh()**

# 신경망

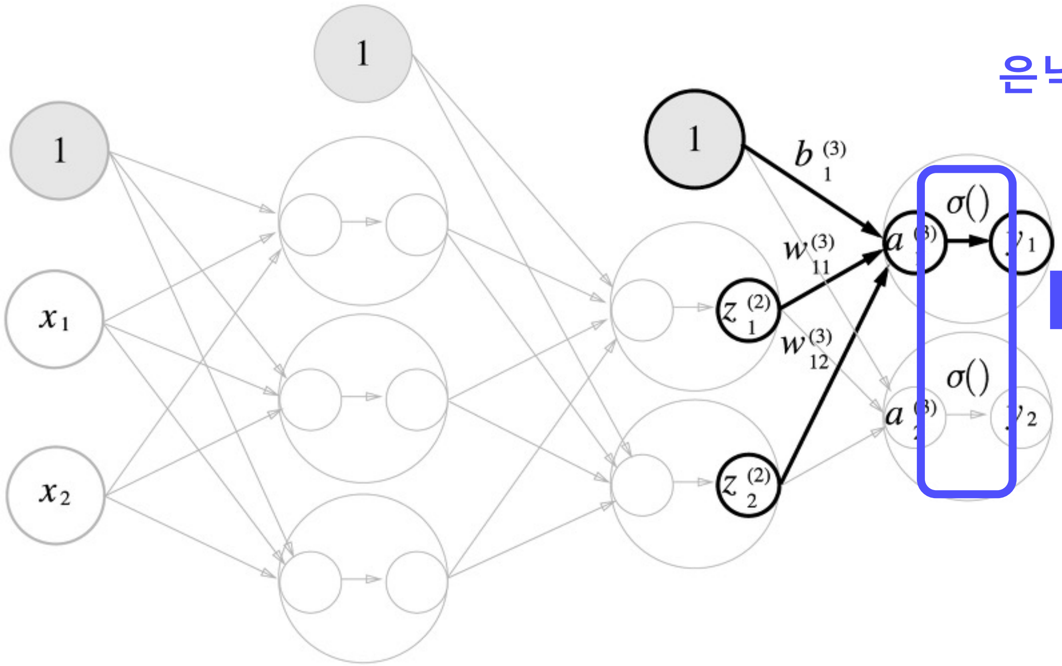


## 3층 신경망 구현하기




**relu()**
**sigmoid()**
**tanh()**

## 3층 신경망 구현하기




은닉층의 활성함수와 출력층의 활성함수가 다름!

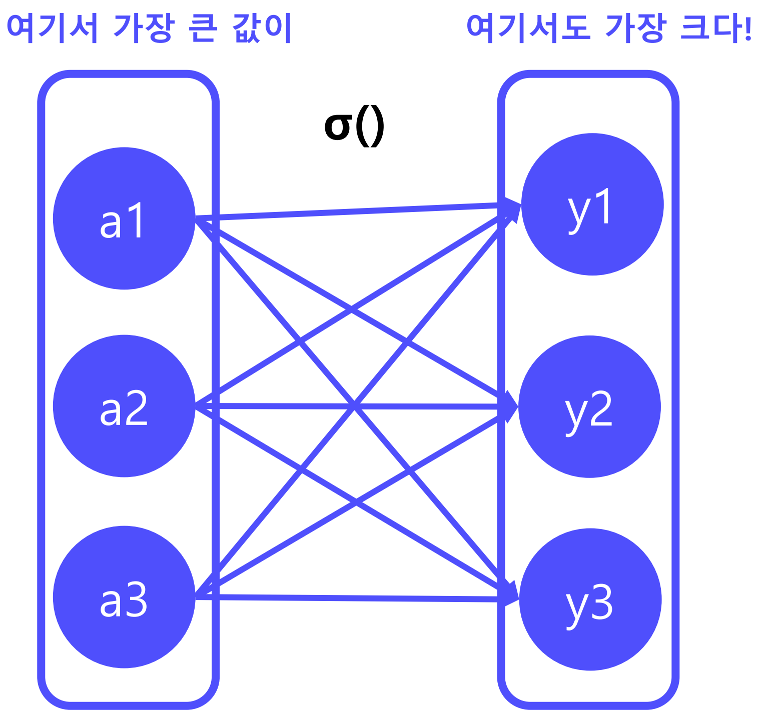
**항등 함수 : 회귀**
**소프트 맥스 함수 : 분류**

# 출력층 설계하기

항등 함수 : 입력을 그대로 출력함. (회귀)

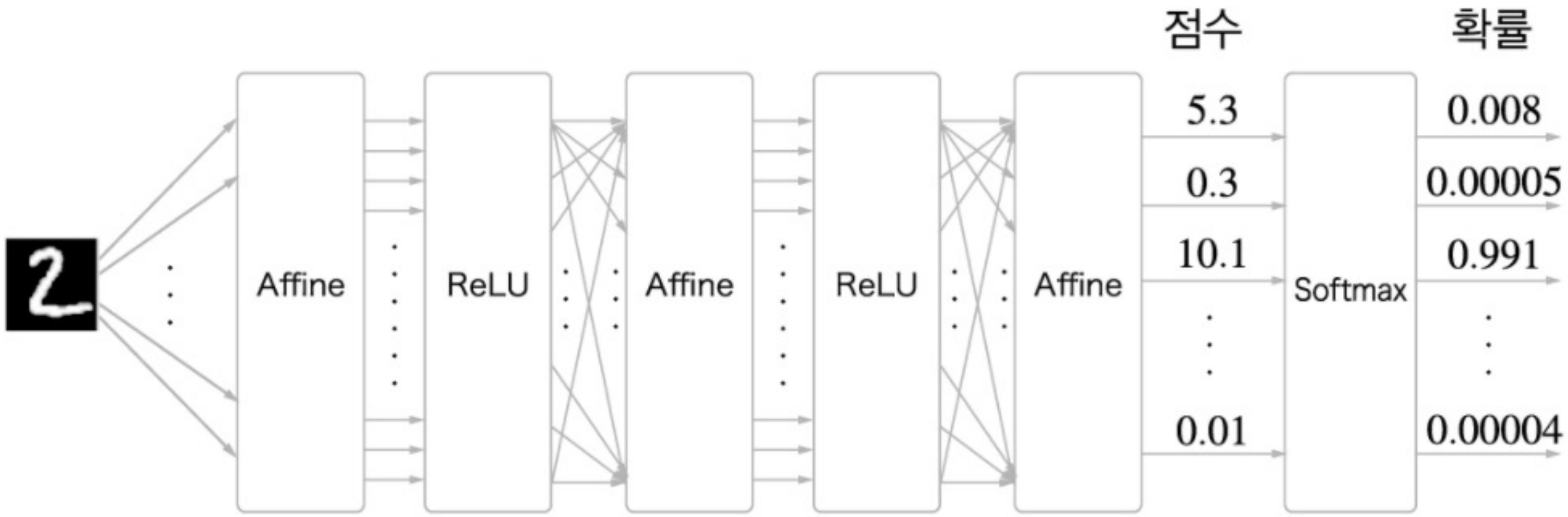
소프트 맥스 함수 : 입력을 0~1사이의 값들로 만든다.(분류)

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$



# 5.6 Affine/Softmax 계층 구현하기

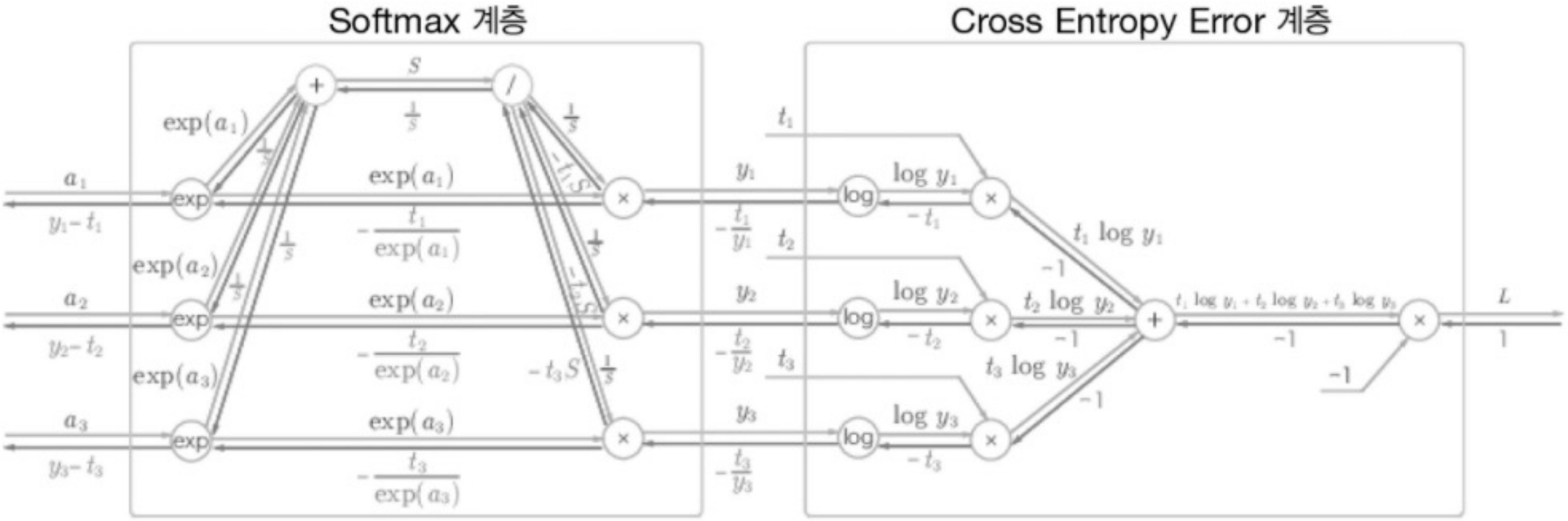
## - 5.6.3 Softmax-with-Loss 계층




**softmax는 점수를 0~1사이의 값으로 변환 하는 것 (출력의 합이 1이 되도록)**

# 5.6 Affine/Softmax 계층 구현하기

## - 5.6.3 Softmax-with-Loss 계층



## 5.6 Affine/Softmax 계층 구현하기

- 5.6.3 Softmax-with-Loss 계층

$$E = -\sum_{k} t_k \log y_k$$

너 100점

신경망

답안 

2
3
3
5

손실 함수

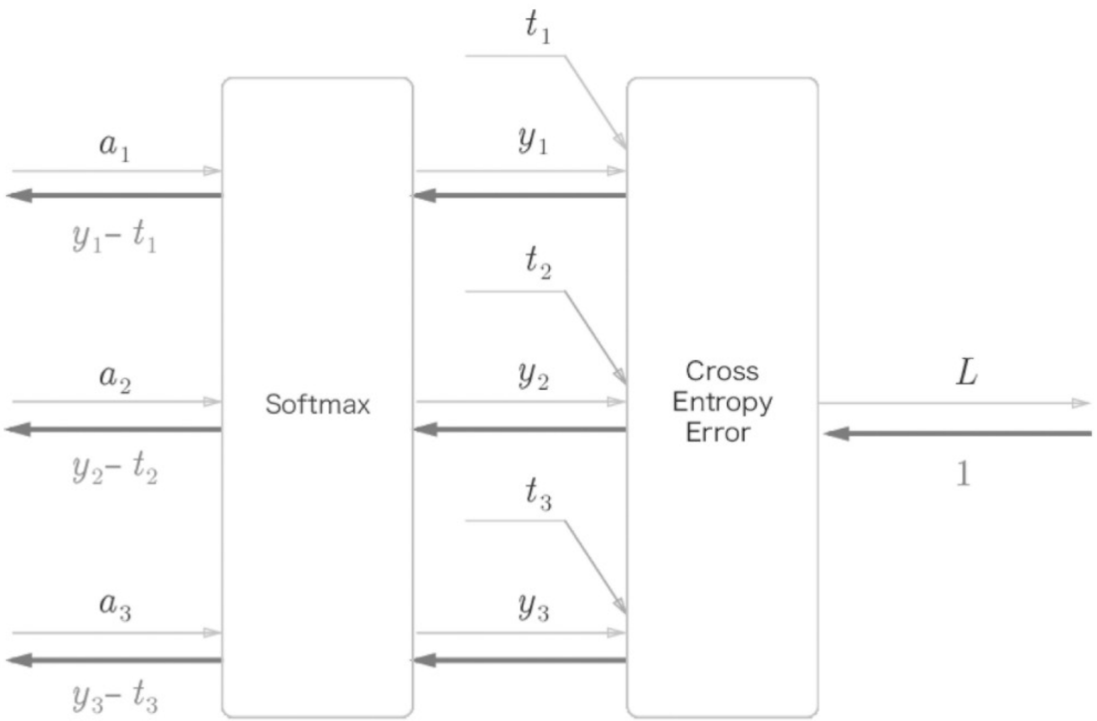
 답지 t

1+1 = 2
1+2 = 3
1+3 = 3
1+4 = 5

$$y_k - t_k$$

깔끔하게 떨어짐!

우연이 아니고
Cross entropy 오차는
이렇게 깔끔하게
떨어지게 설계되어 있음!



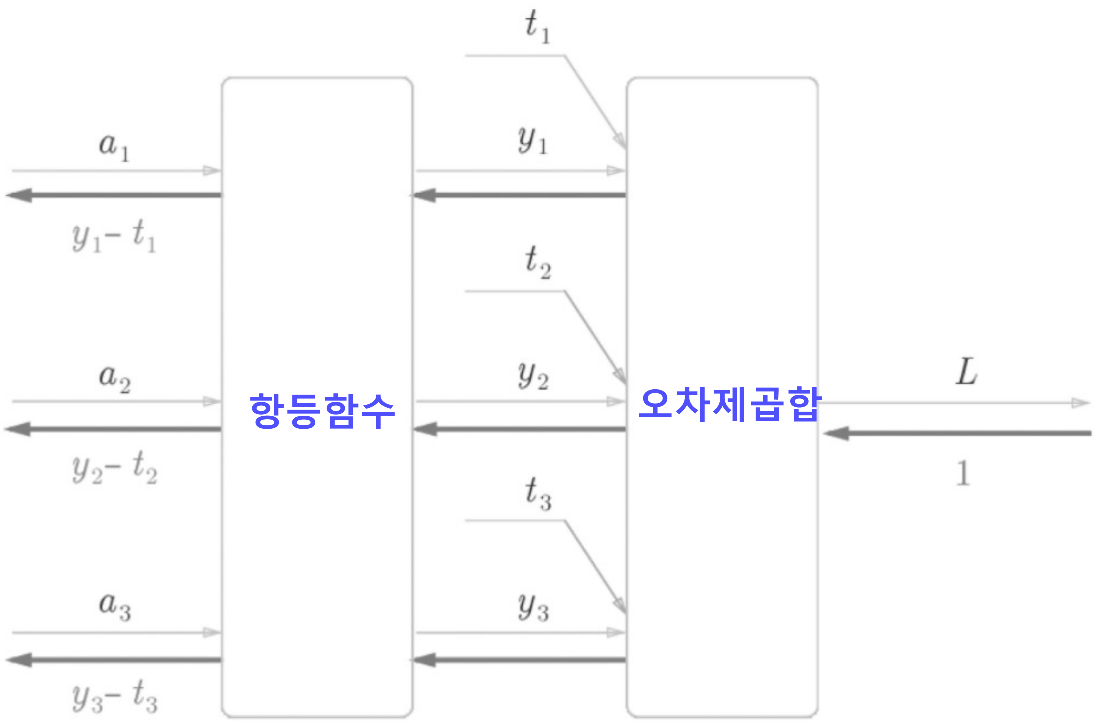

Classification

# 5.6 Affine/Softmax 계층 구현하기

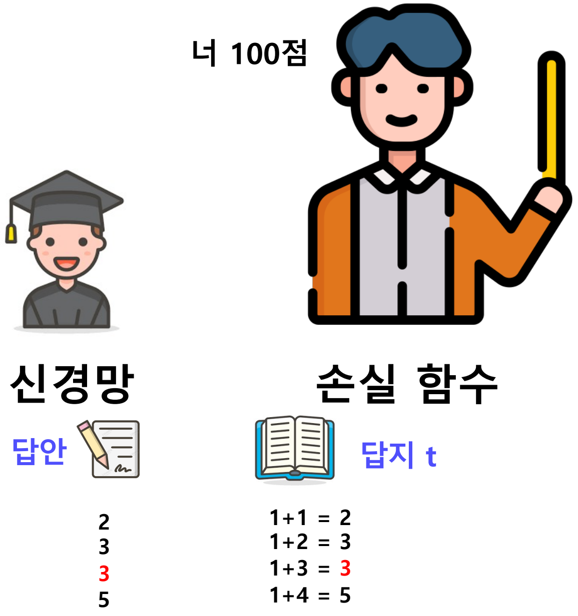
## - 5.6.3 항등함수

$y_k - t_k$ 형태로
깔끔하게 떨어짐!

우연이 아니고
SSE 오차는
이렇게 깔끔하게
떨어지게 설계되어 있음!




Regression



## 5.6 Affine/Softmax 계층 구현하기

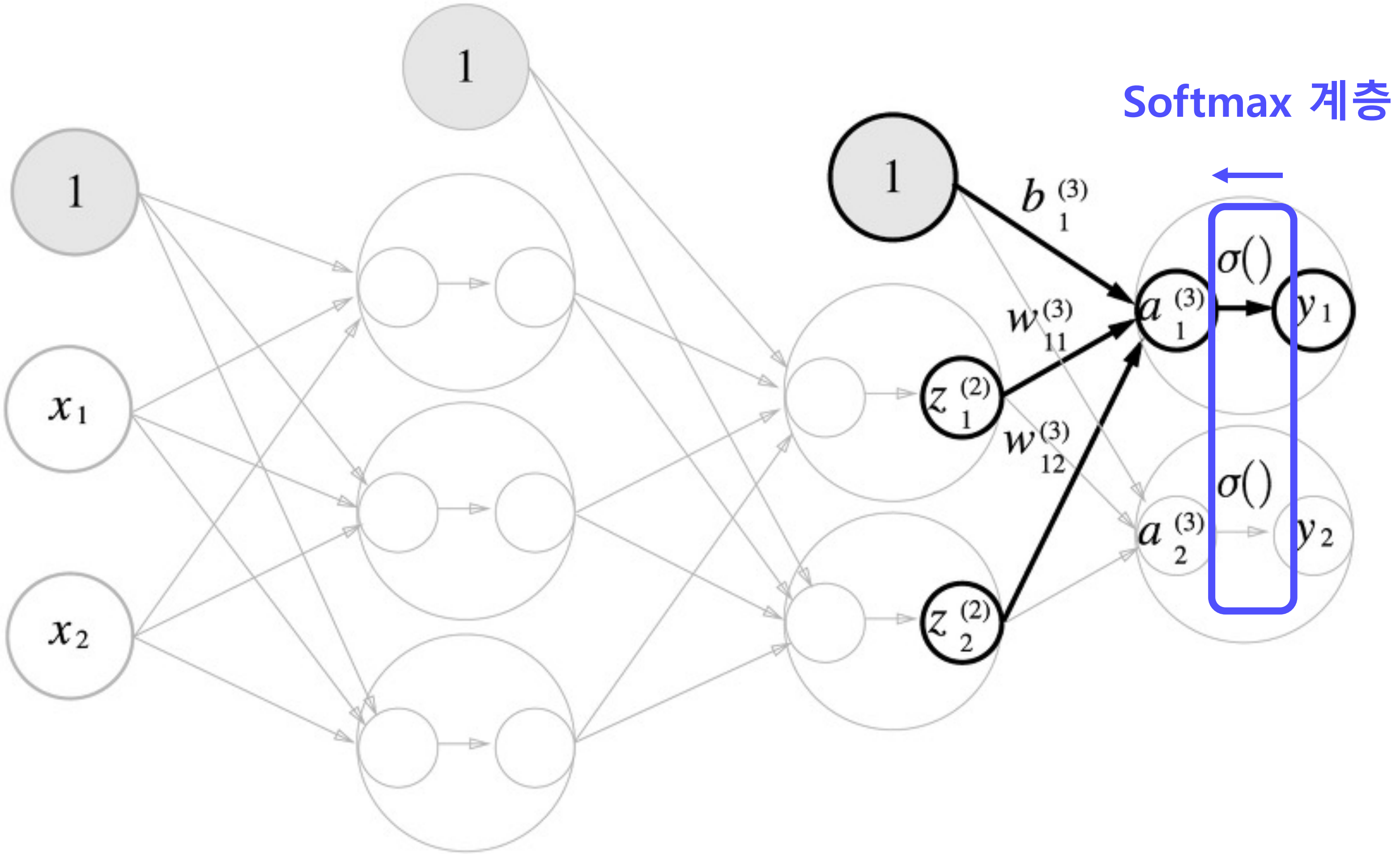
$$y_k - t_k$$

```
class SoftmaxWithLoss:
    def __init--(self):
        self.loss = None   # 손실
        self.y = None      # softmax의 출력
        self.t = None      # 정답 레이블(원-핫 벡터)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)
        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        dx = (self.y - self.t) / batch_size  # 배치의 수로 나눠서
데이터 1개당 오차를 앞 계층으로 전파

        return dx
```

## 학습 알고리즘 구현하기

- 전제
  - 신경망에는 적응 가능한 가중치와 편향이 있고, 가중치와 편향을 훈련 데이터에 적응하도록 조정하는 과정을 '학습'이라 합니다.
- 1단계 미니배치
  - 훈련 데이터 중 일부를 가져와서 미니배치를 만들고 미니배치의 손실 함수 값을 줄이는 것이 목표입니다.
- 2단계 기울기 산출
  - 미니배치의 손실 함수 값을 줄이기 위해 각 가중치 매개변수의 기울기를 구합니다.
  - 기울기는 손실 함수 값을 가장 작게 하는 방향을 제시합니다.
- 3단계 매개변수 갱신
  - 가중치 매개변수를 기울기 방향으로 아주 조금 갱신합니다.
- 4단계 반복
  - 1~3단계 반복

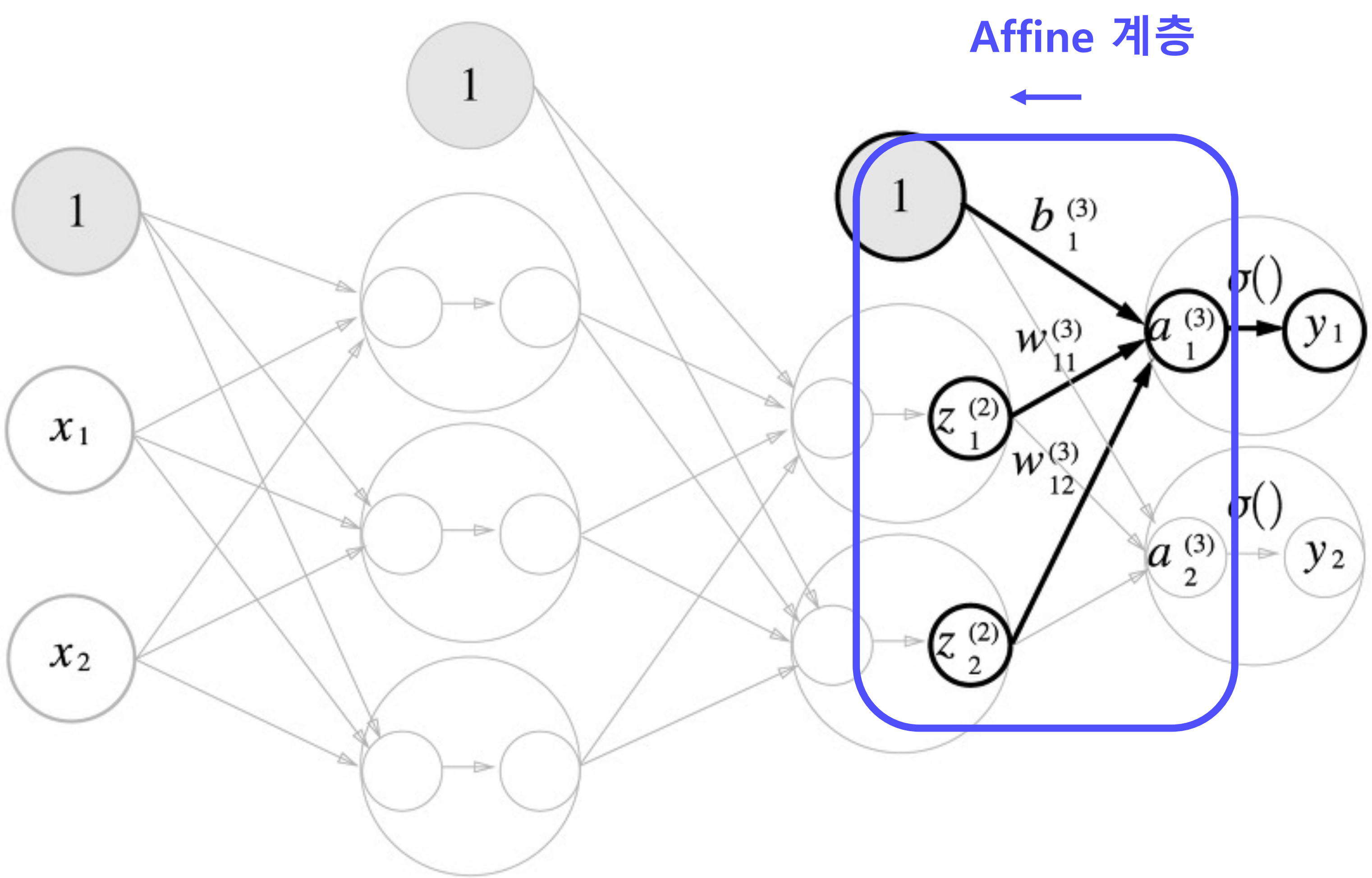
## 단계별 Backpropagation (오차역전파법)



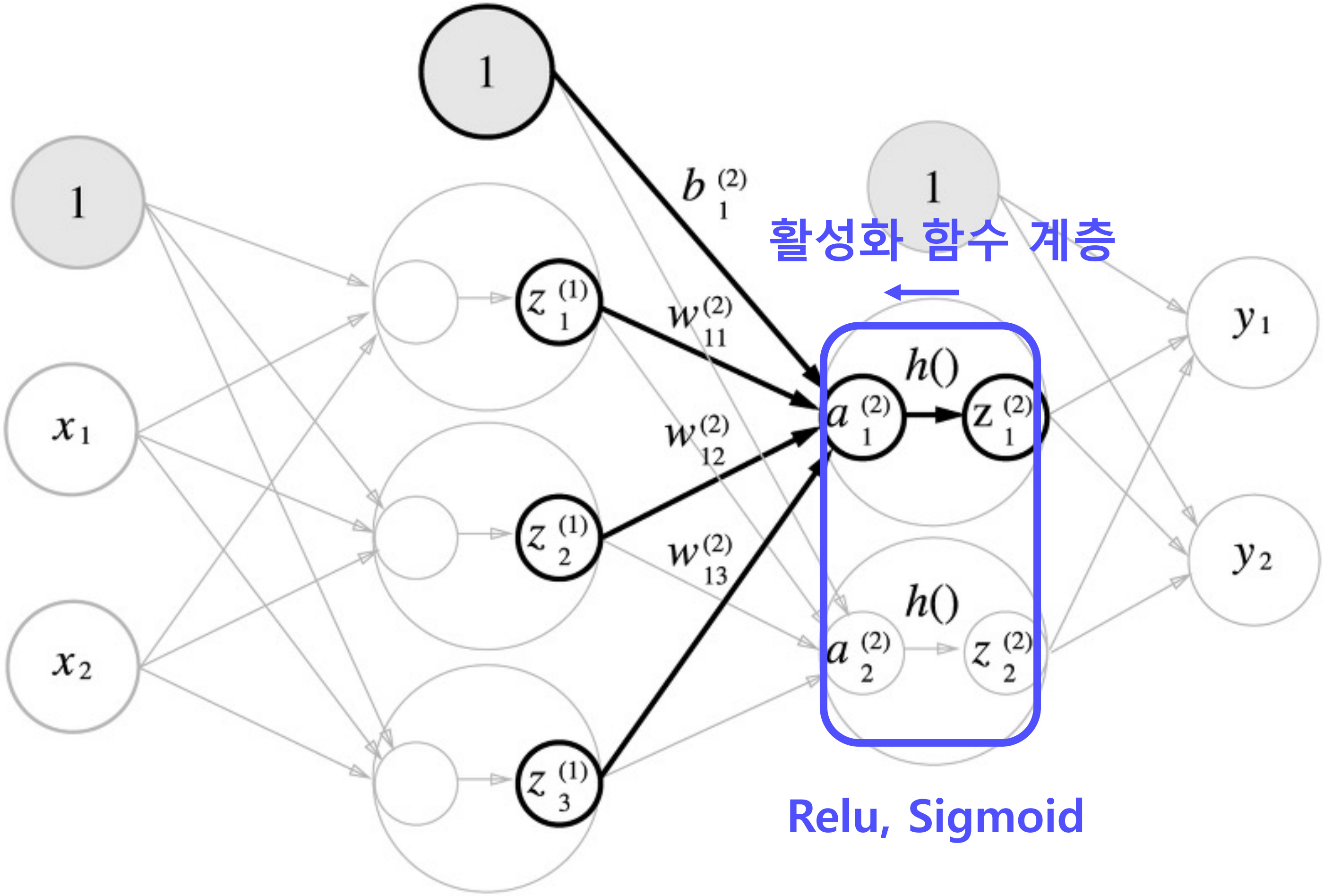
# 오차역전파법



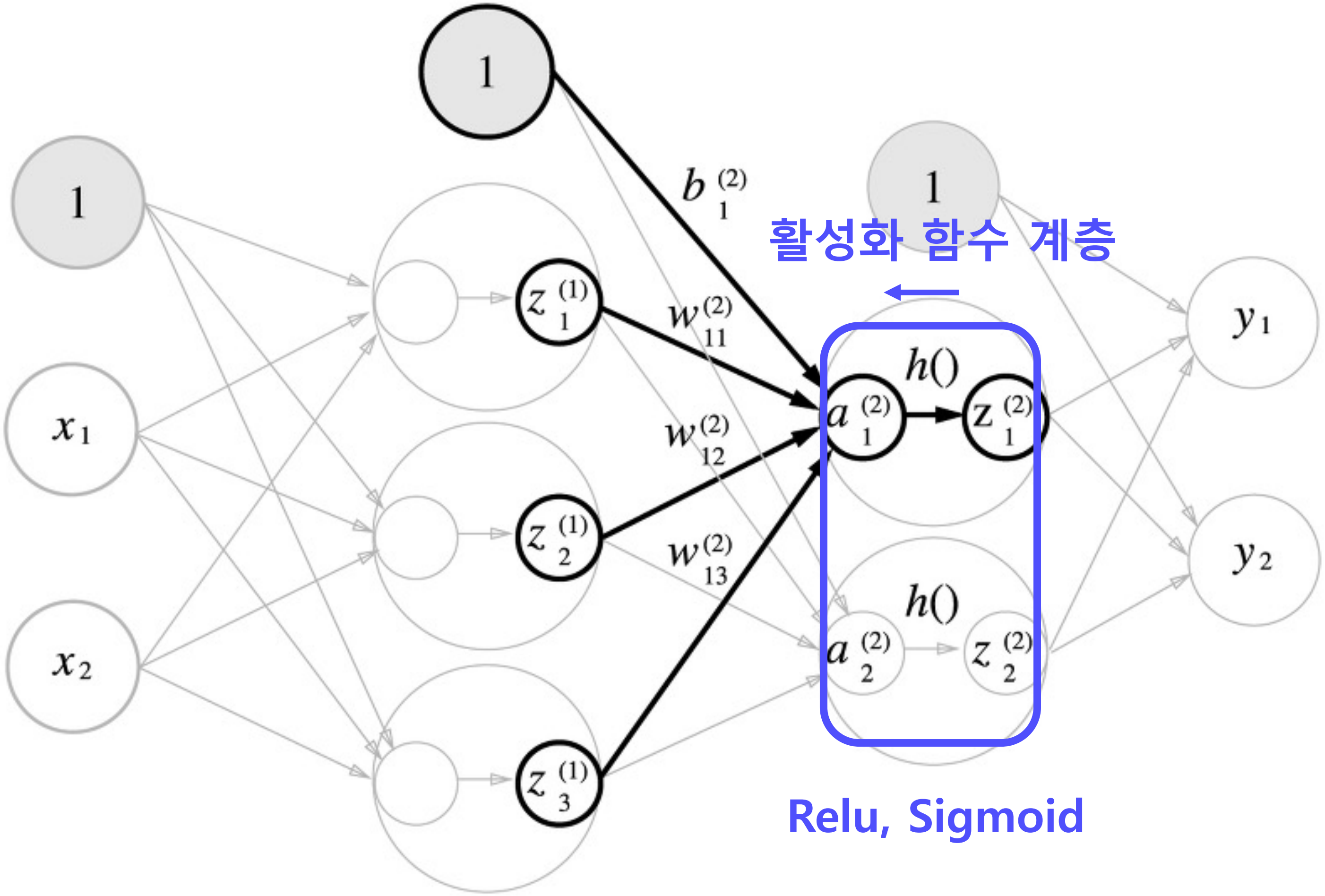
## 단계별 Backpropagation (오차역전파법)



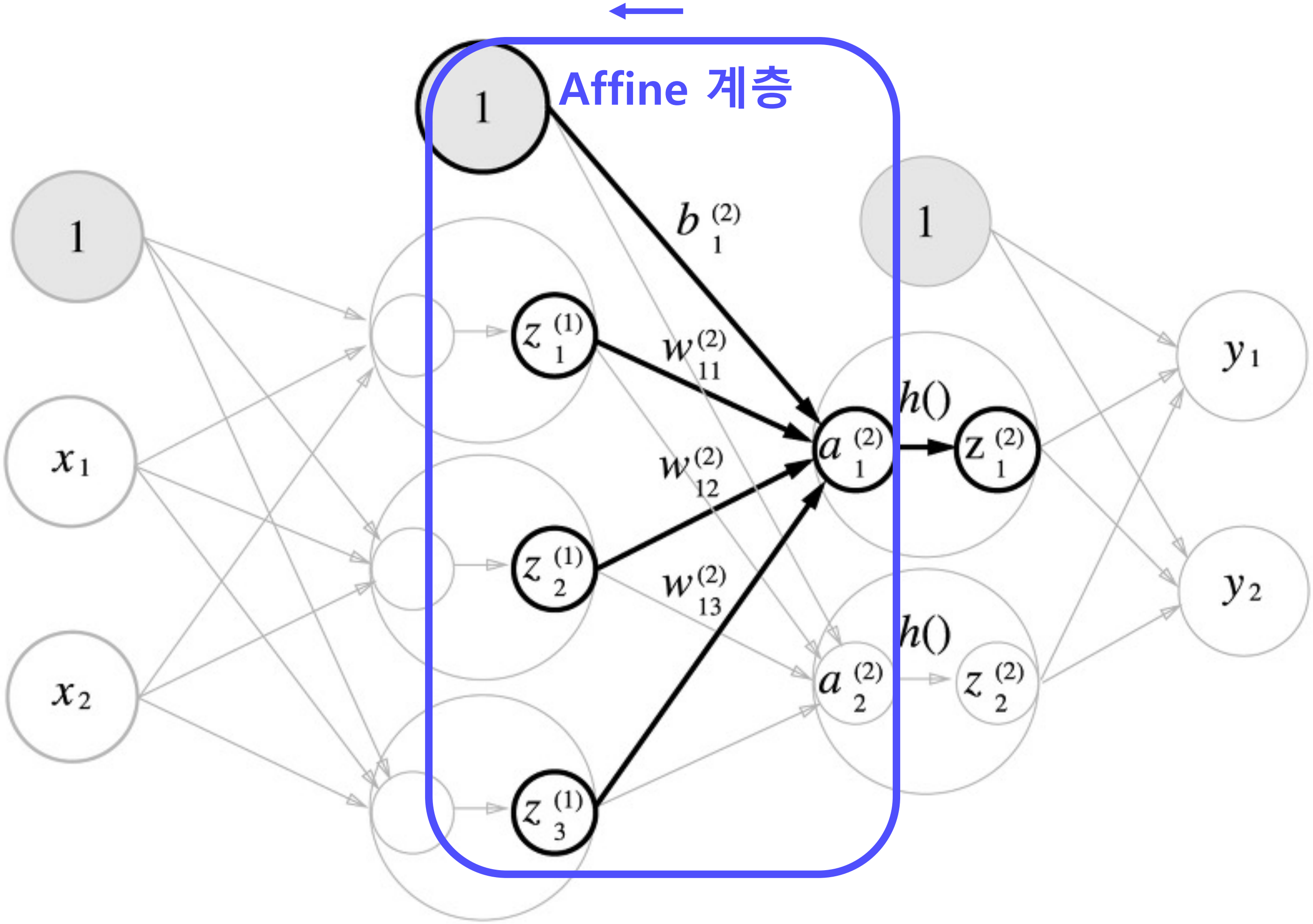
## 단계별 Backpropagation (오차역전파법)



## 단계별 Backpropagation (오차역전파법)



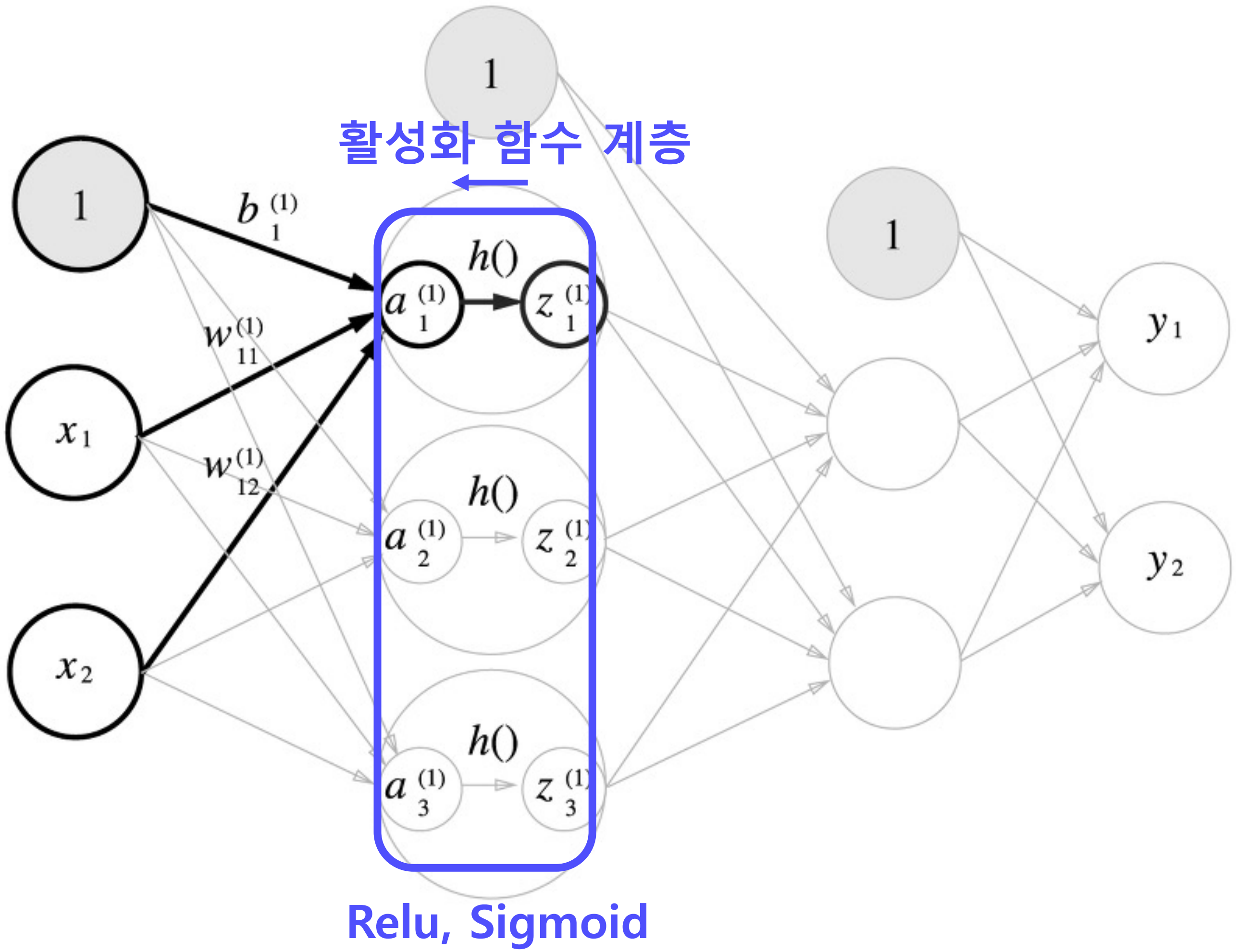
단계별 Backpropagation (오차역전파법)



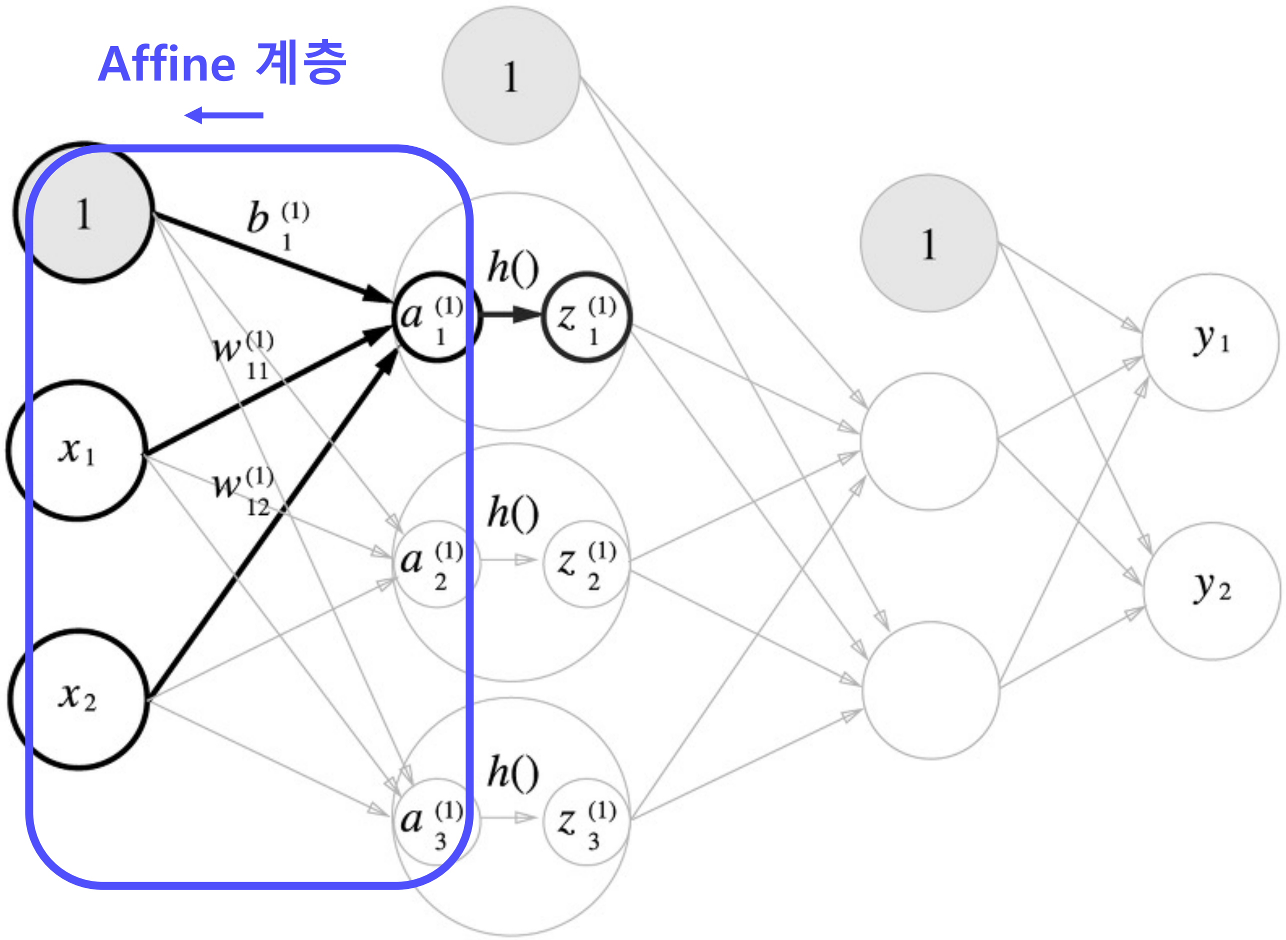
# 오차역전파법



## 단계별 Backpropagation (오차역전파법)



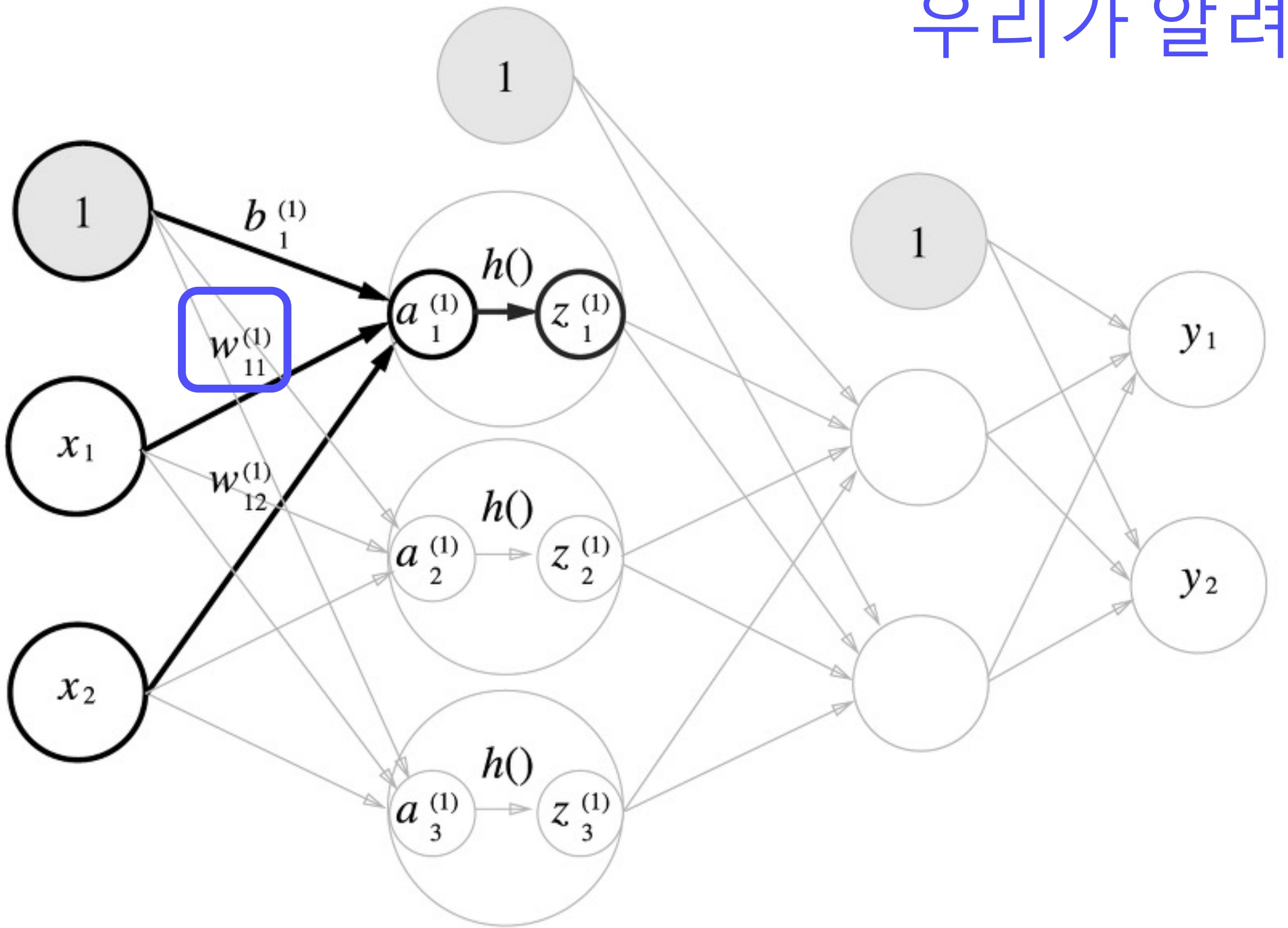
단계별 Backpropagation (오차역전파법)



단계별 Backpropagation (오차역전파법)

우리가 알려고 하는것! $\frac{\partial f}{\partial w}$

$f$ = 손실함수




**Loss()**

## 정리

- 계산 그래프를 이용하면 계산 과정을 시각적으로 파악할 수 있다.
- 계산 그래프의 노드는 국소적 계산으로 구성된다. 국소적 계산을 조합해 전체 계산을 구성한다.
- 계산 그래프의 순전파는 통상의 계산을 수행한다. 한편, 계산 그래프의 역전파로는 각 노드의 미분을 구할 수 있다.
- 신경망의 구성 요소를 계층으로 구현하여 기울기를 효율적으로 계산할 수 있다. (오차역전파법)
- 수치 미분과 오차역전파법의 결과를 비교하면 오차역전파법의 구현에 잘못이 없는지 확인할 수 있다. (기울기 확인)