Neural Network

5. 오차역전파

Chain rule

$$\mathbb{R} \xrightarrow{f} \mathbb{R} \xrightarrow{g} \mathbb{R} \xrightarrow{h} \mathbb{R}$$

Numerical Differentiation

$$\frac{h(g(f(x+\varepsilon))) - h(g(f(x)))}{\varepsilon}$$

Chain rule

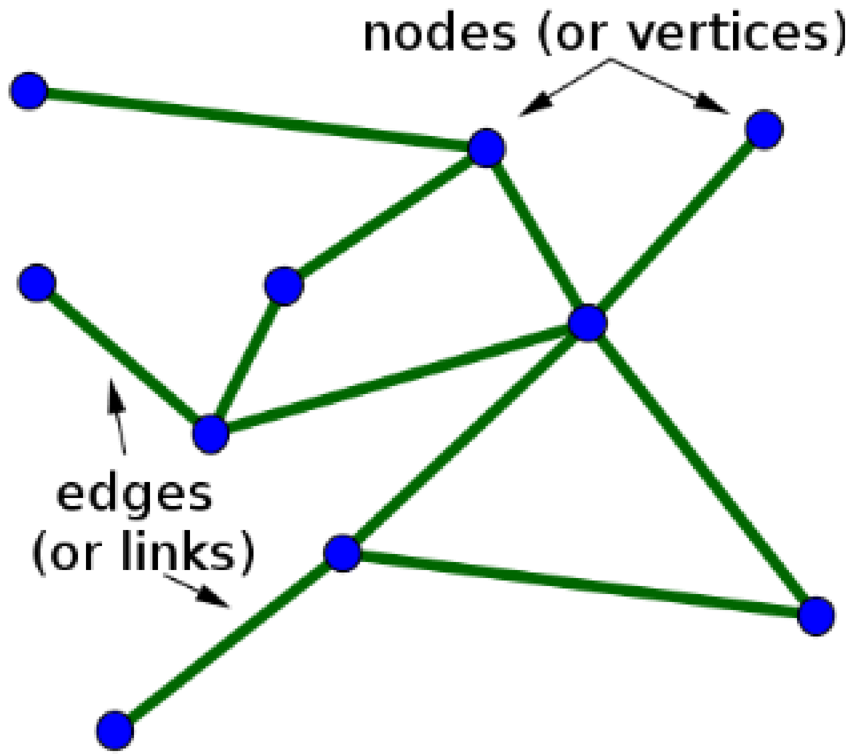
$$h'(g(f(x)))g'(f(x))f'(x)$$

$$f(x) = 2x + 1,\; g(y) = y^2,\; h(z) = \sin(z)$$

$$\frac{\sin\left((2(x+\varepsilon)+1)^2\right) - \sin\left((2x+1)^2\right)}{\varepsilon}$$

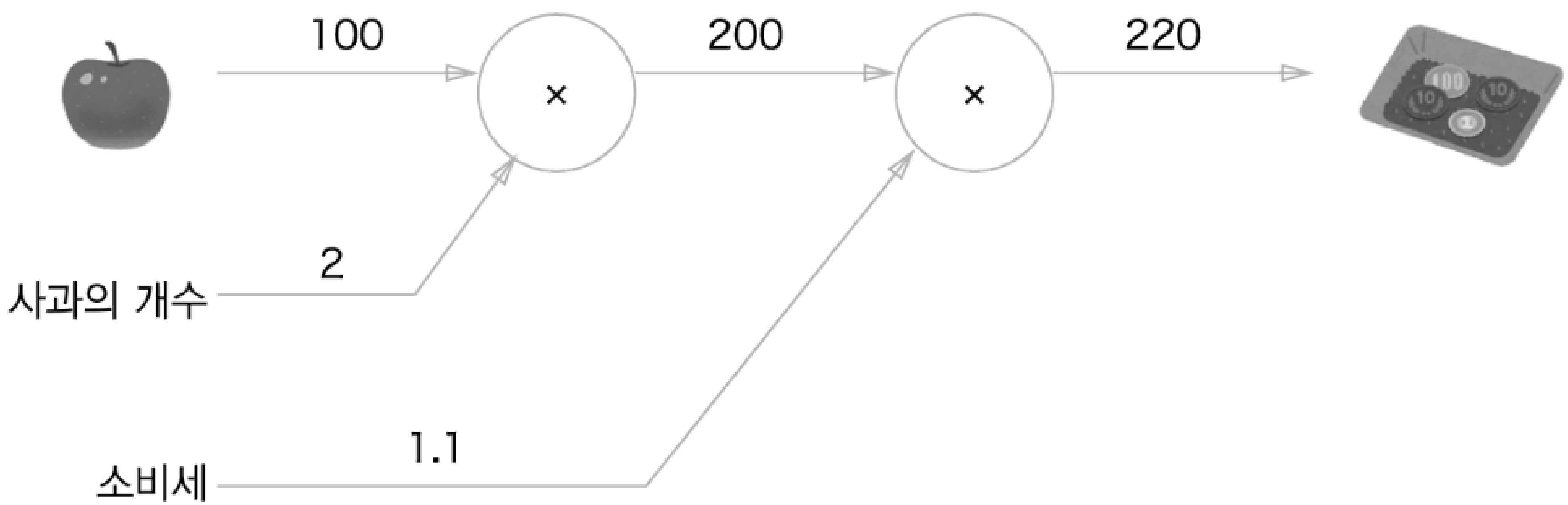
$$\cos\left((2x+1)^2\right) \times 2(2x+1) \times 2$$

Graph



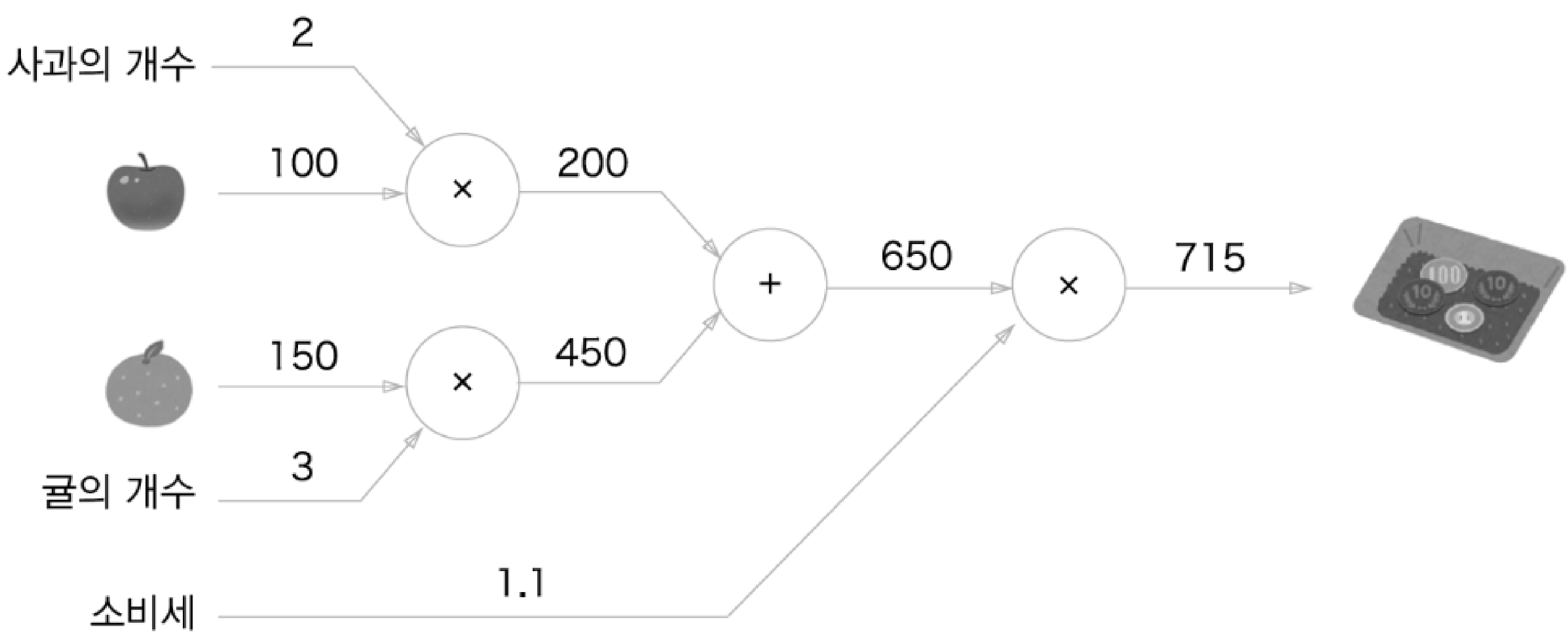
Forward

- 사과 가격: 100원/개, 사과 수: 2개, 소비세: 10%
- 총 금액: 100 x 2 x 1.1 = 220

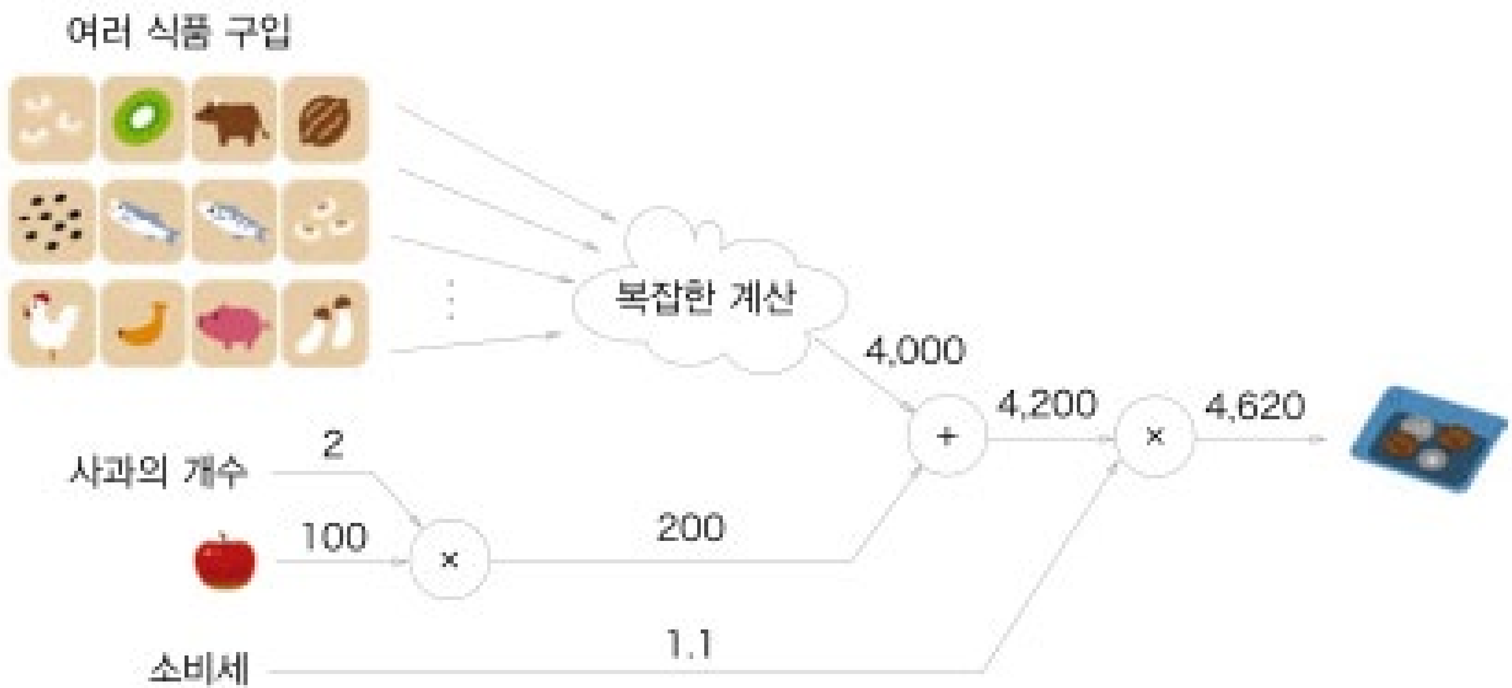


Forward

- 사과 가격: 100원/개, 사과 수: 2개
- 귤 가격: 150원/개, 귤 수: 3개
- 총 금액: (100 x 2 + 150 x 3) x 1.1 = 715

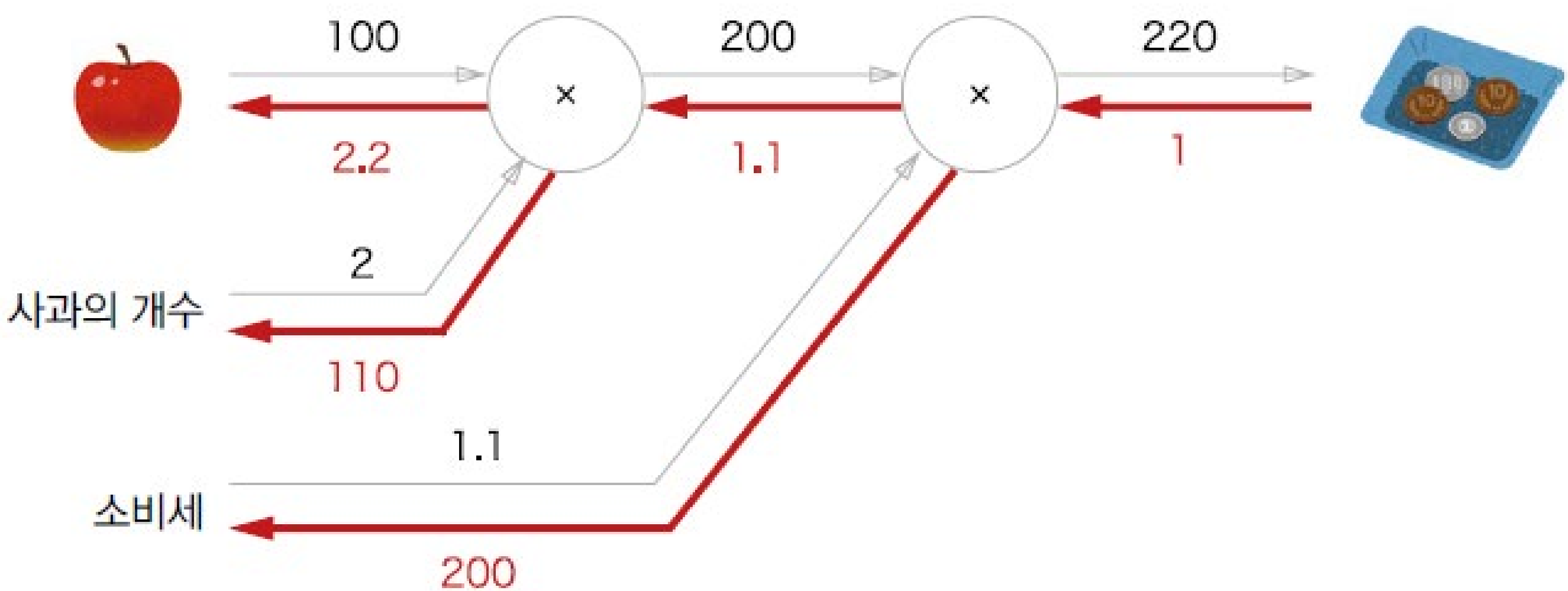


Computation Graph



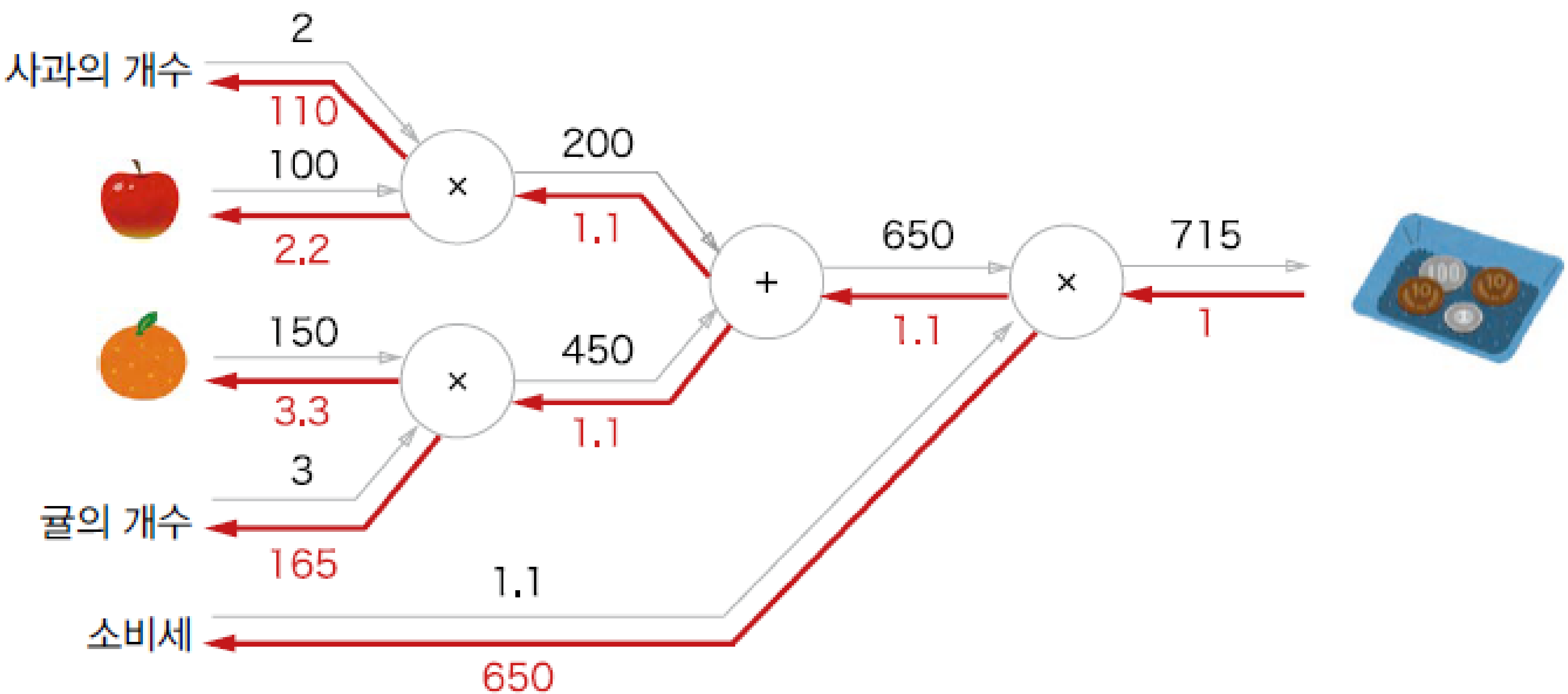
Backpropagation

- 곱셈 노드: 반대편 값을 곱함



Backpropagation

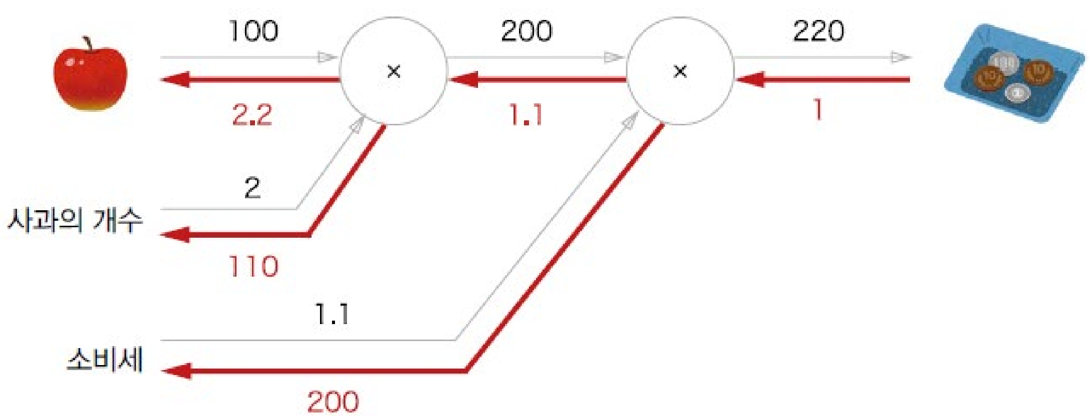
- 덧셈 노드: 유지



- 사과 가격: x 원/개, 사과 수: m 개, 소비세: t
- 총 금액: $f(x,m,t) = xmt$

$$\frac{\partial f}{\partial x}(x,m,t) = mt, \quad \frac{\partial f}{\partial m}(x,m,t) = xt, \quad \frac{\partial f}{\partial t}(x,m,t) = xm$$

$$\frac{\partial f}{\partial x}(100,2,1.1) = 2.2, \quad \frac{\partial f}{\partial m}(100,2,1.1) = 110, \quad \frac{\partial f}{\partial t}(100,2,1.1) = 200$$

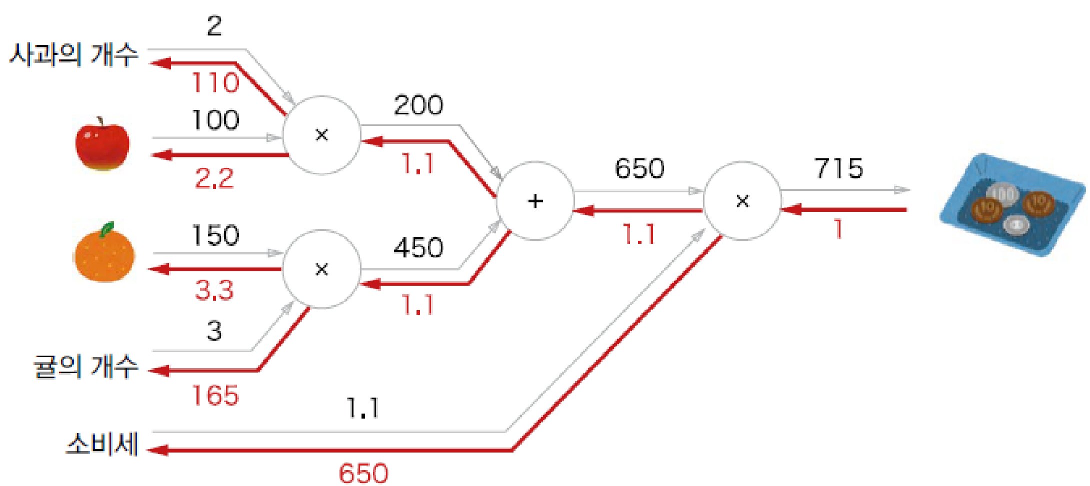


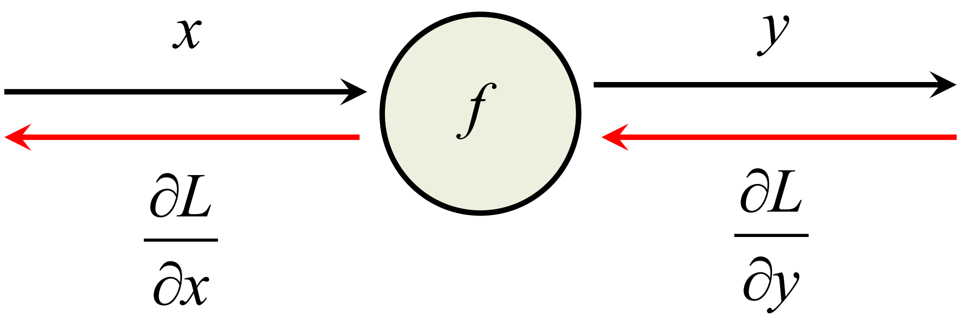
- 사과 가격: x 원/개, 사과 수: m 개
- 귤 가격: y 원/개, 귤 수: n 개, 소비세: t
- 총 금액: $f(x, m, y, n, t) = (xm + yn)t$

$$\frac{\partial f}{\partial x}(x, m, y, n, t) = mt, \quad \frac{\partial f}{\partial m}(x, m, y, n, t) = xt \quad \frac{\partial f}{\partial y}(x, m, y, n, t) = nt, \quad \frac{\partial f}{\partial n}(x, m, y, n, t) = yt, \quad \frac{\partial f}{\partial t}(x, m, y, n, t) = xm + yn$$

$$\frac{\partial f}{\partial x}(100, 2, 150, 3, 1.1) = 2.2, \quad \frac{\partial f}{\partial m}(100, 2, 150, 3, 1.1) = 110$$

$$\frac{\partial f}{\partial y}(100, 2, 150, 3, 1.1) = 3.3, \quad \frac{\partial f}{\partial n}(100, 2, 150, 3, 1.1) = 165, \quad \frac{\partial f}{\partial t}(100, 2, 150, 3, 1.1) = 650$$



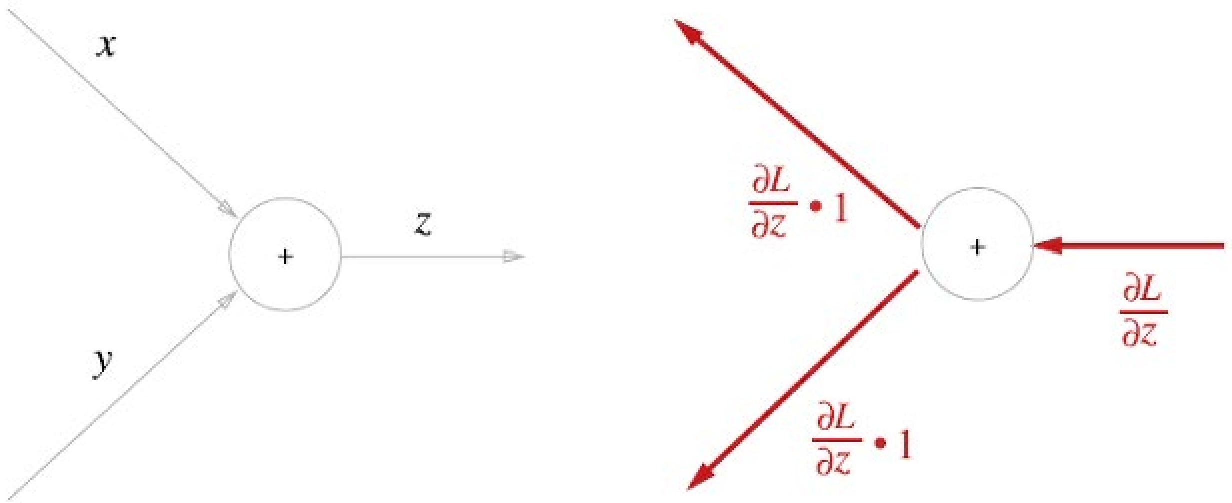


$$y = f(x)$$

$$L(y) = L(f(x))$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x}$$

Addition

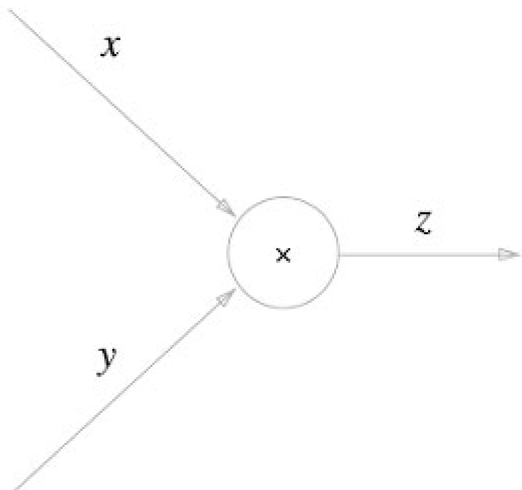


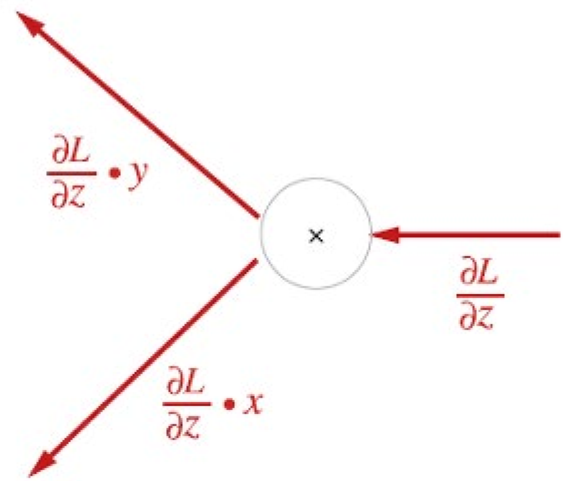
$$L(z) = L(x + y)$$

$$\frac{\partial}{\partial x} L(x + y) = L'(x + y) \times 1$$

$$\frac{\partial}{\partial y} L(x + y) = L'(x + y) \times 1$$

Multiplication





$$L(z) = L(xy)$$

$$\frac{\partial}{\partial x} L(xy) = L'(xy) \times y$$

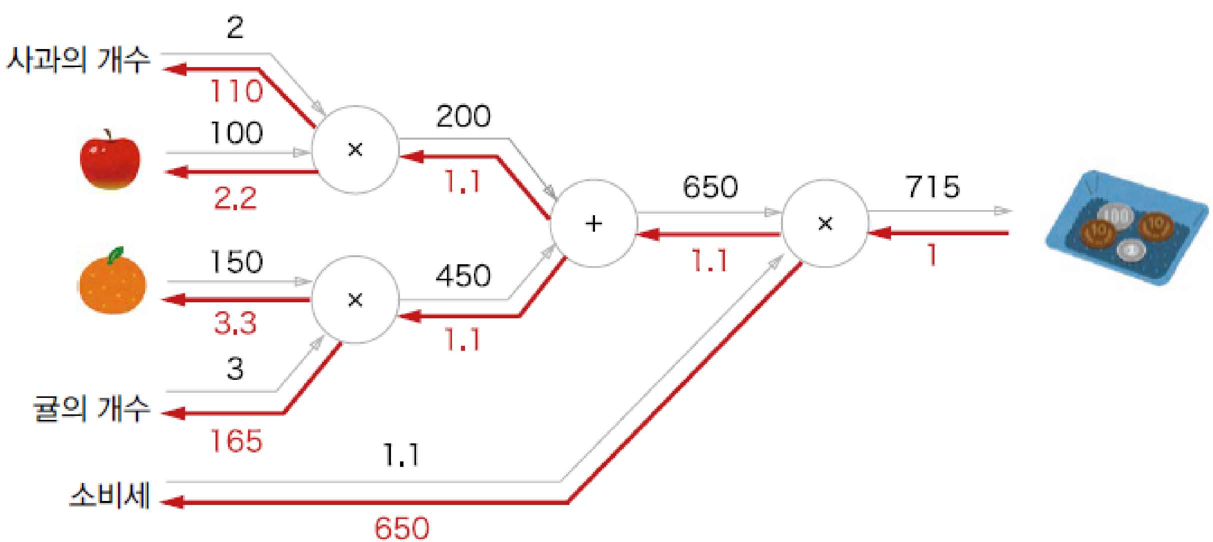
$$\frac{\partial}{\partial y} L(xy) = L'(xy) \times x$$

- 사과 가격: x 원/개, 사과 수: m 개
- 귤 가격: y 원/개, 귤 수: n 개, 소비세: t
- 총 금액: $f(x,m,y,n,t) = (xm+yn)t$

$$f_1(x) = mx, \qquad f_2(z) = z + yn, \qquad f_3(w) = wt$$

$$f_3 \circ f_2 \circ f_1(x) = (xm+yn)t$$

$$(f_3 \circ f_2 \circ f_1)'(x) = f_3'(f_2(f_1(x))) \times f_2'(f_1(x)) \times f_1'(x) = t \times 1 \times m$$



Addition

```python
class AddLayer:
    def __init__(self):
        pass

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

    def backward(self, dout):
        dx = dout * 1
        dy = dout * 1
        return dx, dy
```

```python
apple = 100
apple_num = 2
orange = 150
orange_num = 3
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_orange_layer = MulLayer()
add_apple_orange_layer = AddLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num) #(1)
orange_price = mul_orange_layer.forward(orange, orange_num) #(2)
all_price = add_apple_orange_layer.forward(apple_price, orange_price) #(3)
price = mul_tax_layer.forward(all_price, tax) #(4)

# 역전파
dprice = 1
dall_price, dtax = mul_tax_layer.backward(dprice) #(4)
dapple_price, dorange_price = add_apple_orange_layer.backward(dall_price) #(3)
dorange, dorange_num = mul_orange_layer.backward(dorange_price) #(2)
dapple, dapple_num = mul_apple_layer.backward(dapple_price) #(1)

print(price) # 715
print(dapple_num, dapple, dorange, dorange_num, dtax) # 110 2.2 3.3 165 650
```

Multiplication

```python
class MulLayer:
    def __init__(self):
        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):
        dx = dout * self.y # x와 y를 바꾼다.
        dy = dout * self.x

        return dx, dy
```

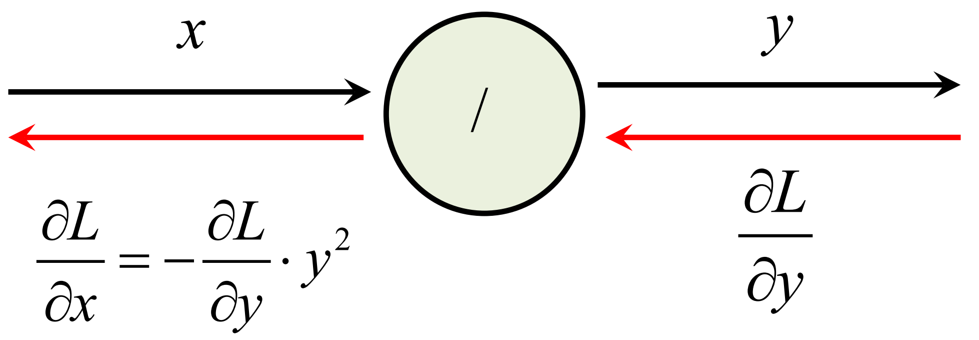
```python
apple = 100
apple_num = 2
tax = 1.1

# 계층들
mul_apple_layer = MulLayer()
mul_tax_layer = MulLayer()

# 순전파
apple_price = mul_apple_layer.forward(apple, apple_num)
price = mul_tax_layer.forward(apple_price, tax)

print(price) # 220
```

Inverse

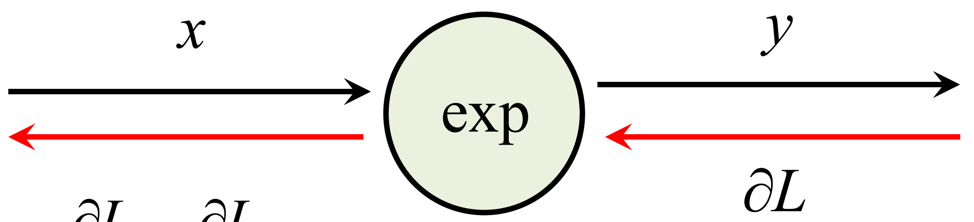


$$y = 1/x$$

$$\frac{dy}{dx} = -x^{-2} = -y^2$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = -\frac{\partial L}{\partial y} \cdot y^2$$

Exponential



$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot y$$

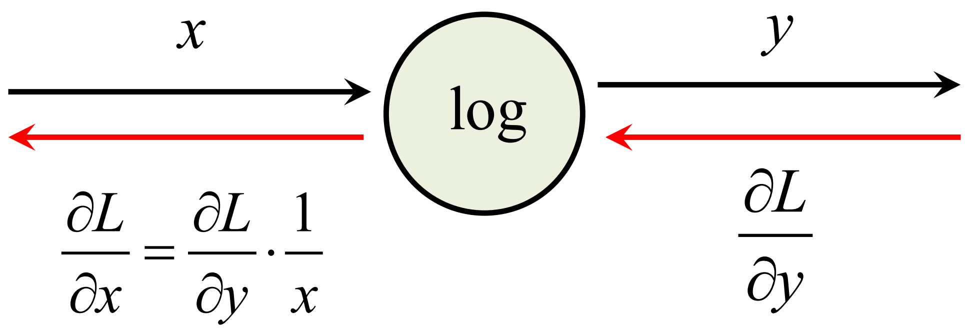
$$\frac{\partial L}{\partial y}$$

$$y = e^x$$

$$\frac{dy}{dx} = e^x = y$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{\partial L}{\partial y} \cdot y$$

Log

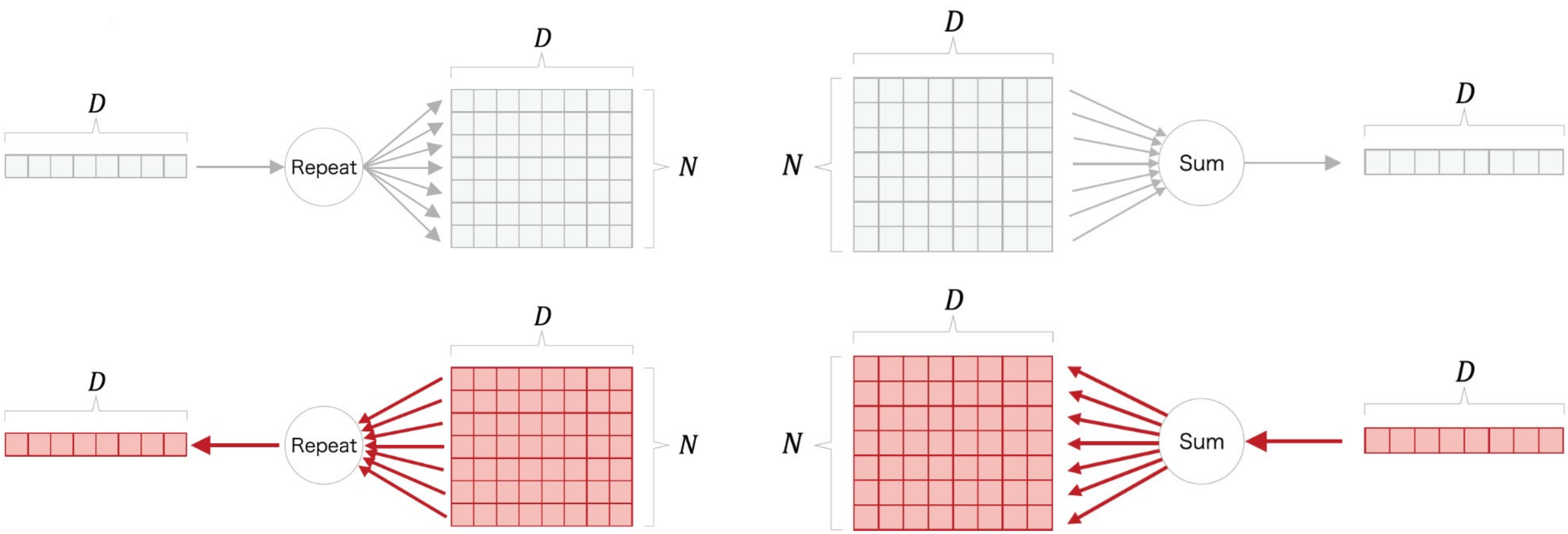


$$y = \log x$$

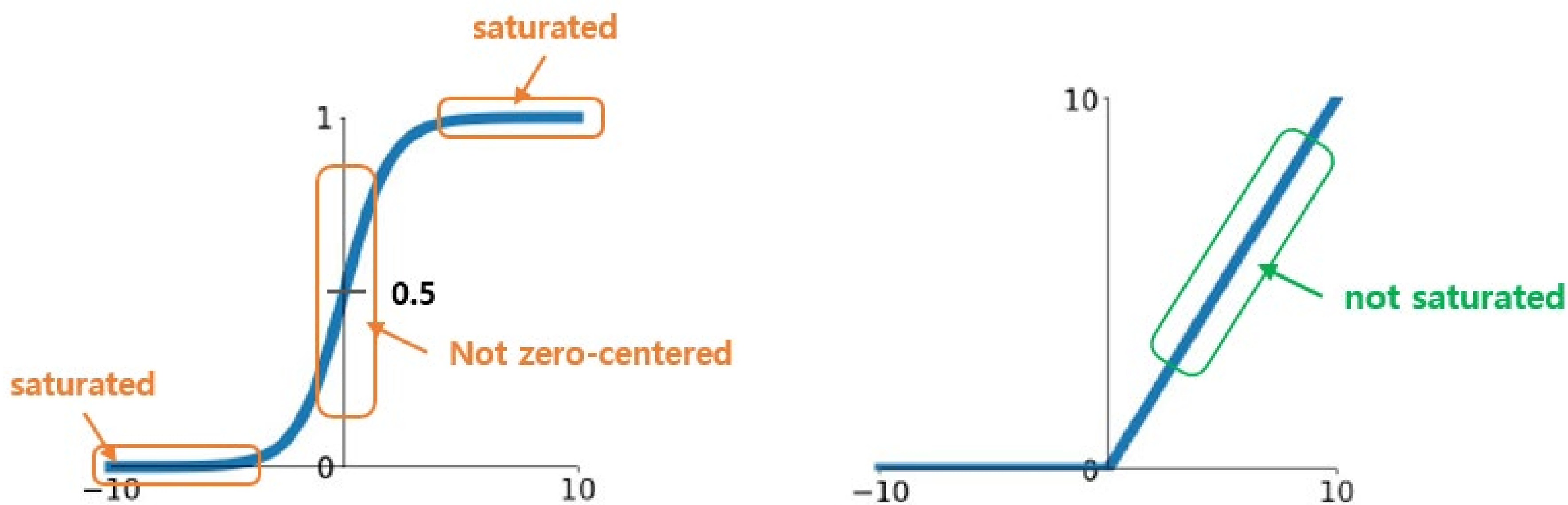
$$\frac{dy}{dx} = \frac{1}{x}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{1}{x}$$

Repeat & Sum



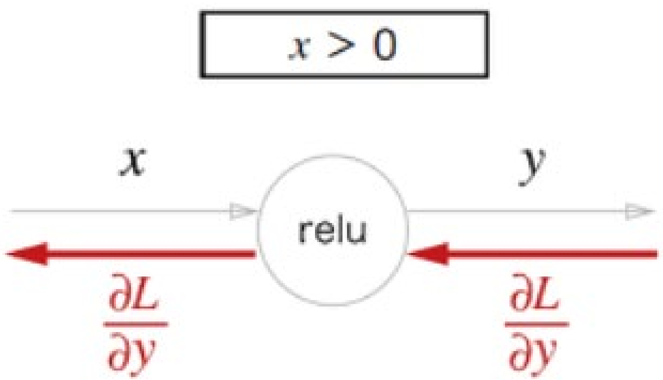
Sigmoid vs ReLU

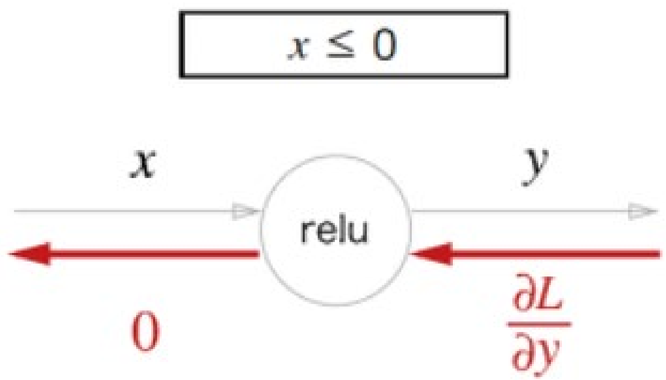


ReLU

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

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





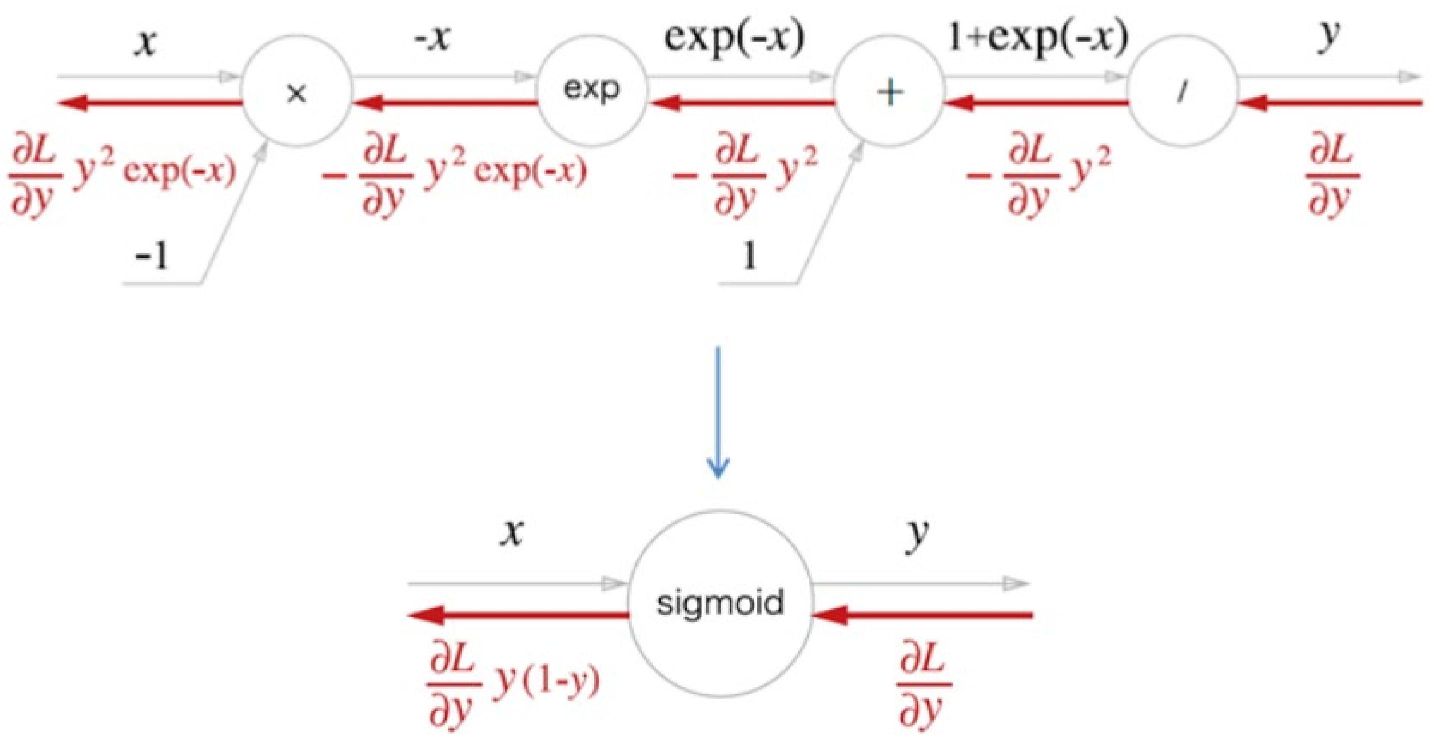
Sigmoid

$$y = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$$

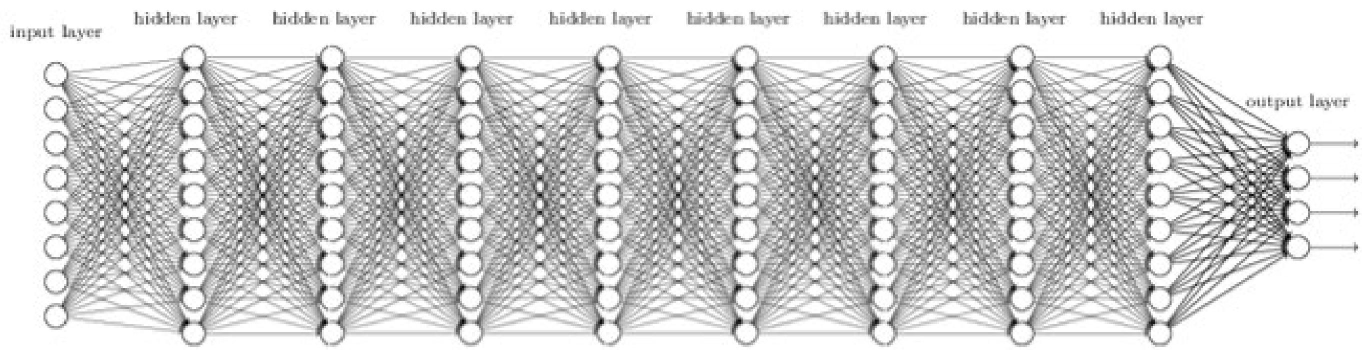
$$\begin{aligned}
y' &= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1+e^{-x}-1}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} - \frac{1}{(1+e^{-x})^2} \\
&= y - y^2 \\
&= y(1-y)
\end{aligned}$$

Sigmoid

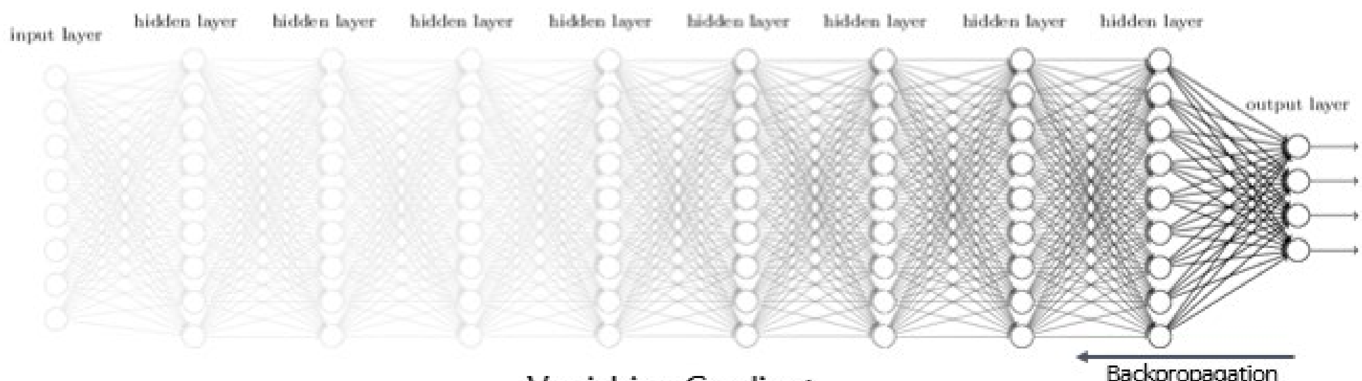
$$y = \frac{1}{1+e^{-x}} = (1+e^{-x})^{-1}$$



Vanishing Gradient



Deep Neural Network



Vanishing Gradient

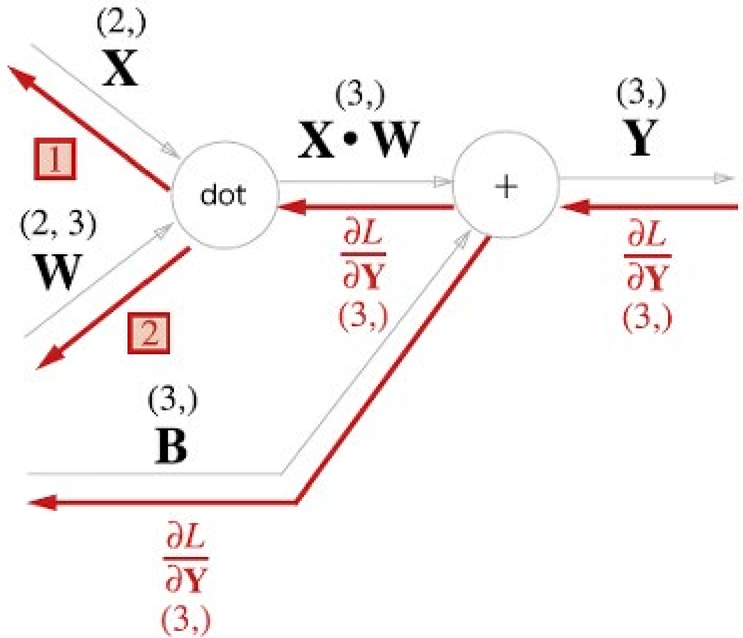
Affine

$$\mathrm{Y} = \mathrm{XW} + \mathrm{B}$$

$$\begin{pmatrix} y_1 & y_2 & y_3 \end{pmatrix} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} + \begin{pmatrix} b_1 & b_2 & b_3 \end{pmatrix}$$

$$\boxed{1} \quad \underset{(2,)}{\frac{\partial L}{\partial \mathbf{X}}} = \underset{(3,)}{\frac{\partial L}{\partial \mathbf{Y}}} \; \underset{(3,\,2)}{\mathbf{W}^{\mathrm{T}}}$$

$$\boxed{2} \quad \underset{(2,\,3)}{\frac{\partial L}{\partial \mathbf{W}}} = \underset{(2,\,1)}{\mathbf{X}^{\mathrm{T}}} \; \underset{(1,\,3)}{\frac{\partial L}{\partial \mathbf{Y}}}$$



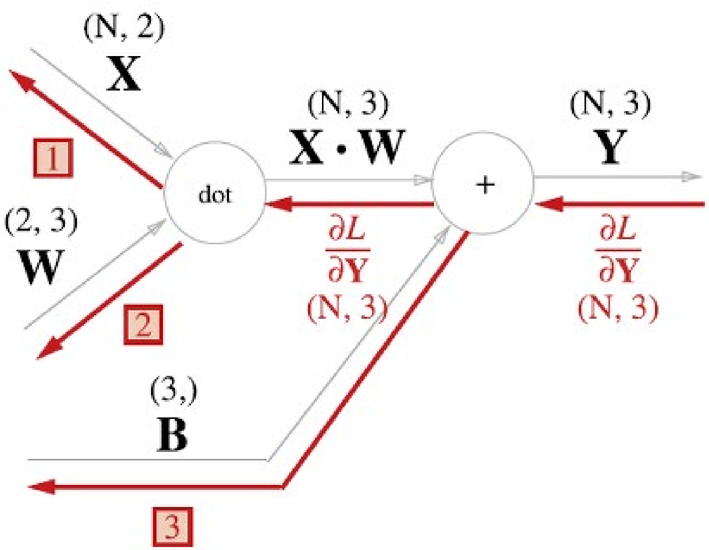
Affine (Batch)

$$\mathbf{Y} = \mathbf{X}\mathbf{W} + \mathbf{B}$$

$$\begin{pmatrix} y_{11} & y_{12} & y_{13} \\ y_{21} & y_{22} & y_{23} \\ & \vdots & \\ y_{N1} & y_{N2} & y_{N3} \end{pmatrix} = \begin{pmatrix} x_{11} & x_{21} \\ x_{21} & x_{22} \\ & \vdots \\ x_{N1} & x_{N2} \end{pmatrix} \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix} + \begin{pmatrix} b_1 & b_2 & b_3 \\ b_1 & b_2 & b_3 \\ & \vdots & \\ b_1 & b_2 & b_3 \end{pmatrix}$$

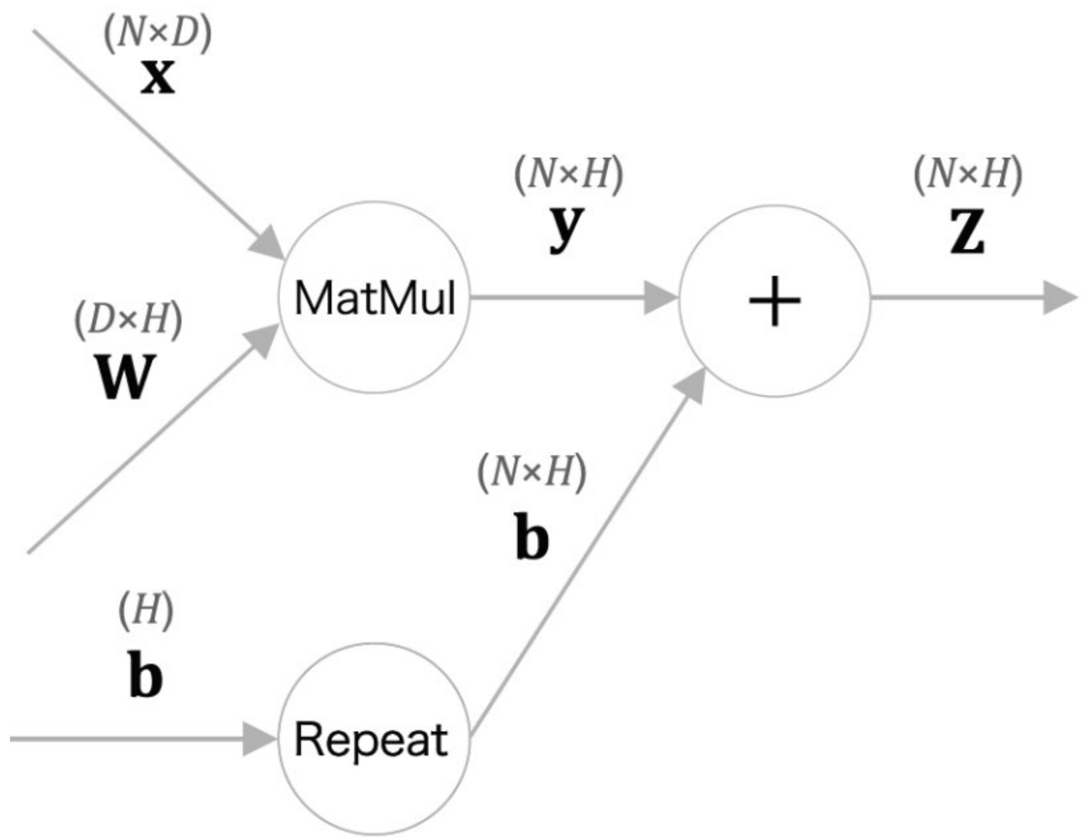
1 $$\underset{(N,\,2)}{\frac{\partial L}{\partial \mathbf{X}}} = \underset{(N,\,3)}{\frac{\partial L}{\partial \mathbf{Y}}} \cdot \underset{(3,\,2)}{\mathbf{W}^{\mathrm{T}}}$$

2 $$\underset{(2,\,3)}{\frac{\partial L}{\partial \mathbf{W}}} = \underset{(2,\,N)}{\mathbf{X}^{\mathrm{T}}} \cdot \underset{(N,\,3)}{\frac{\partial L}{\partial \mathbf{Y}}}$$



3 $\underset{(3)}{\frac{\partial L}{\partial \mathbf{B}}} = \underset{(N,\,3)}{\frac{\partial L}{\partial \mathbf{Y}}}$ 의 첫 번째 축(0축, 열방향)의 합

Affine (Batch)



Softmax-with-Loss

$$\xrightarrow{(a_1, a_2, a_3)} \boxed{\text{Softmax}} \xrightarrow{(y_1, y_2, y_3)} \boxed{\text{Cross Entropy}} \xrightarrow{L}$$

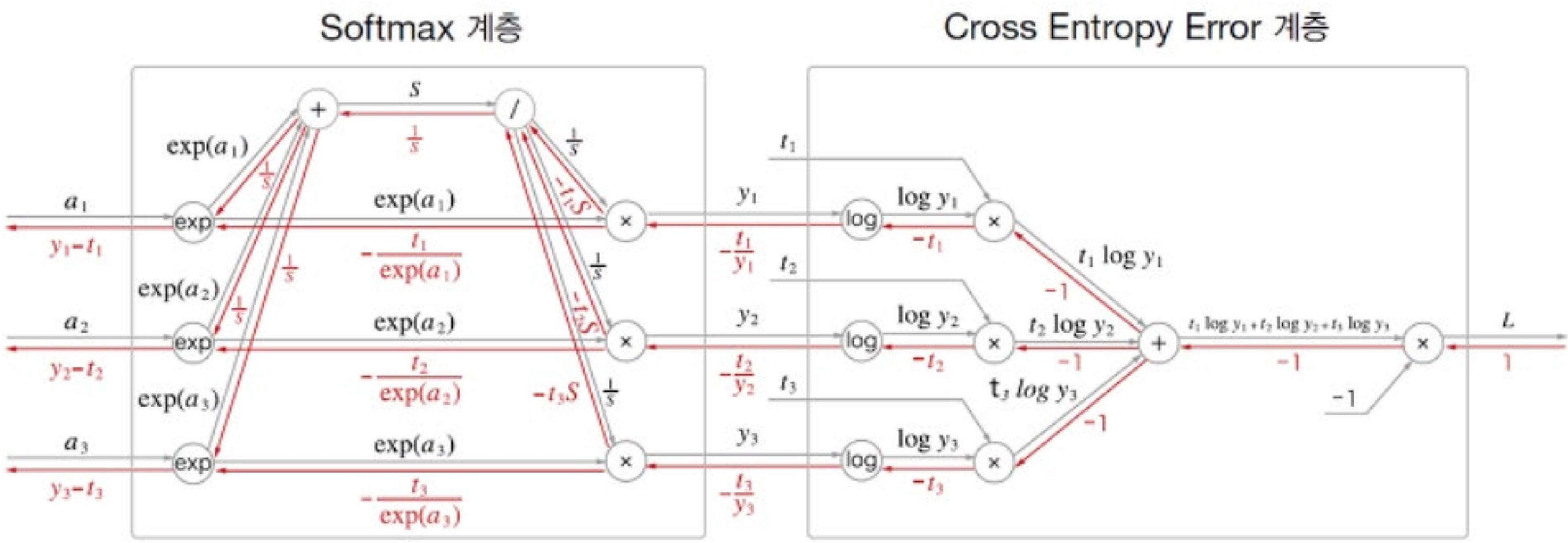
$$\begin{aligned}
L(a_1, a_2, a_3) &= -t_1 \log y_1 - t_2 \log y_2 - t_3 \log y_3 \\
&= -t_1 \log \frac{e^{a_1}}{e^{a_1} + e^{a_2} + e^{a_3}} - t_2 \log \frac{e^{a_2}}{e^{a_1} + e^{a_2} + e^{a_3}} - t_3 \log \frac{e^{a_3}}{e^{a_1} + e^{a_2} + e^{a_3}} \\
&= -t_1 \log e^{a_1} - t_2 \log e^{a_2} - t_3 \log e^{a_3} + (t_1 + t_2 + t_3) \log(e^{a_1} + e^{a_2} + e^{a_3}) \\
&= -t_1 a_1 - t_2 a_2 - t_3 a_3 + \log(e^{a_1} + e^{a_2} + e^{a_3})
\end{aligned}$$

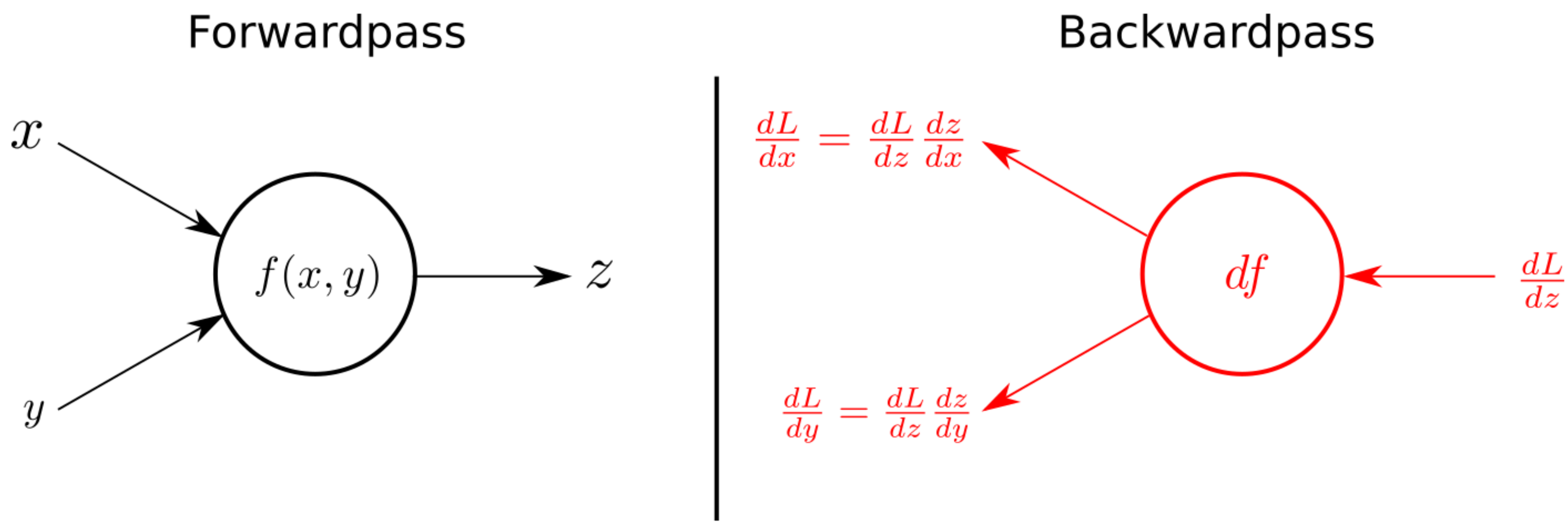
$$\begin{aligned}
\frac{\partial L}{\partial a_i} &= -t_i + \frac{e^{a_i}}{e^{a_1} + e^{a_2} + e^{a_3}} = -t_i + y_i \\
\frac{\partial L}{\partial a} &= (y_1 - t_1, y_2 - t_2, y_3 - t_3) = y - t
\end{aligned}$$

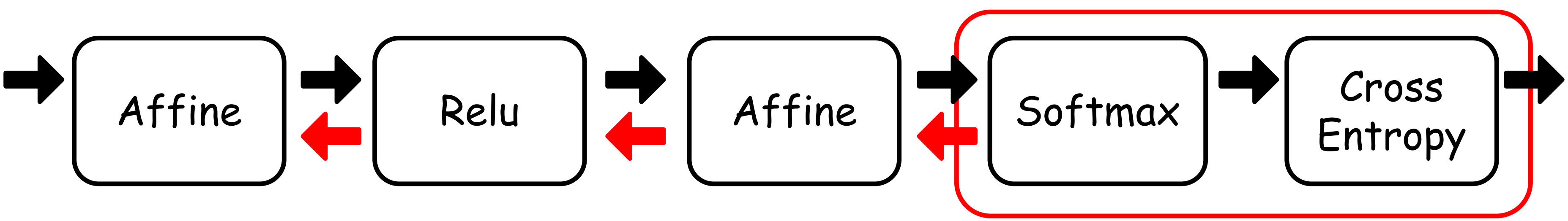
Softmax-with-Loss

$$\left(\frac{e^{a_1}}{e^{a_1}+e^{a_2}+e^{a_3}}, \frac{e^{a_2}}{e^{a_1}+e^{a_2}+e^{a_3}}, \frac{e^{a_3}}{e^{a_1}+e^{a_2}+e^{a_3}}\right)$$

$$-t_1 \log y_1 - t_2 \log y_2 - t_3 \log y_3$$




Forwardpass
Backwardpass
x
y
$f(x, y)$
z
df
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dz}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$


Affine
Relu
Affine
Softmax
Cross Entropy

인스턴스 변수	설명
params	딕셔너리 변수로, 신경망의 매개변수를 보관 params['W1']은 1번째 층의 가중치, params['b1']은 1번째 층의 편향 params['W2']는 2번째 층의 가중치, params['b2']는 2번째 층의 편향
layers	순서가 있는 딕셔너리 변수로, 신경망의 계층을 보관 layers['Affine1'], layers['Relu1'], layers['Affine2']와 같이 각 계층을 순서대로 유지
lastLayer	신경망의 마지막 계층 이 예에서는 SoftmaxWithLoss 계층

메서드	설명
__init__(self, input_size, hidden_size, output_size, weight_init_std)	초기화를 수행한다. 인수는 앞에서부터 입력층 뉴런 수, 은닉층 뉴런 수, 출력층 뉴런 수, 가중치 초기화 시 정규분포의 스케일
predict(self, x)	예측(추론)을 수행한다. 인수 x는 이미지 데이터
loss(self, x, t)	손실 함수의 값을 구한다. 인수 x는 이미지 데이터, t는 정답 레이블
accuracy(self, x, t)	정확도를 구한다.
numerical_gradient(self, x, t)	가중치 매개변수의 기울기를 수치 미분 방식으로 구한다(앞 장과 같음).
gradient(self, x, t)	가중치 매개변수의 기울기를 오차역전파법으로 구한다.

Question?

자료 출처

Deep learning from scratch, 한빛미디어, 사이토고키

https://github.com/youbeebee/deeplearning_from_scratch