# 딥러닝의 정석 Chapter 3

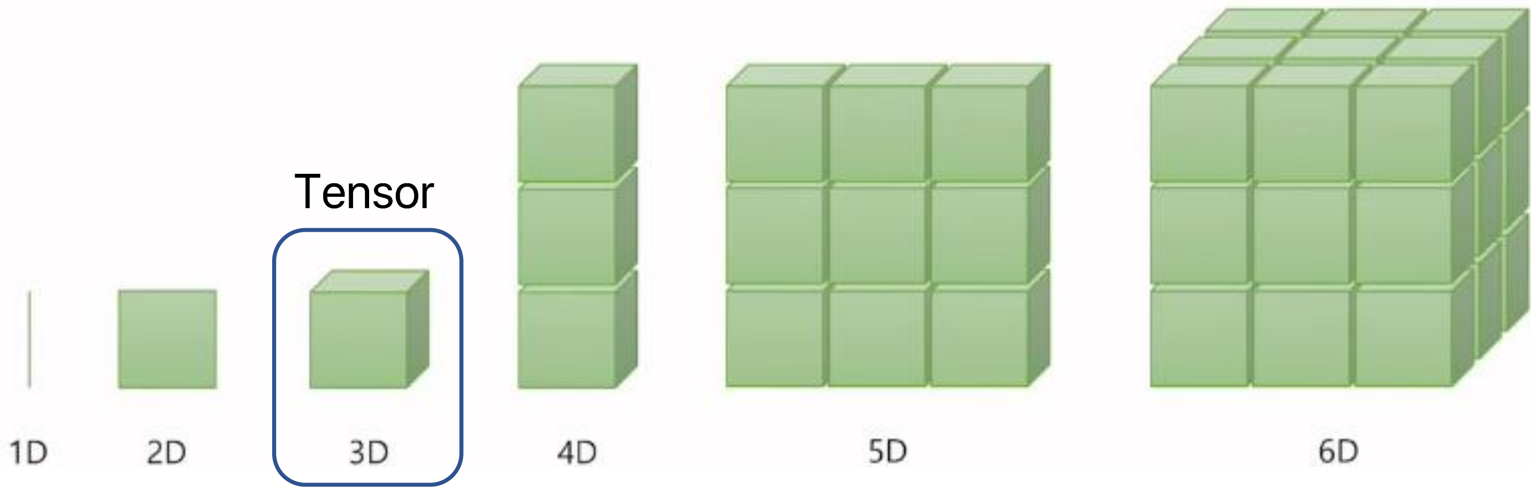
텐서플로로 신경망 구현하기 – based on Pytorch

# Index

- PyTorch
    - Tensor
    - Autograd
    - PyTorch Functions
    - CPU / GPU
- 로지스틱 회귀모델
- MNIST

# Pytorch

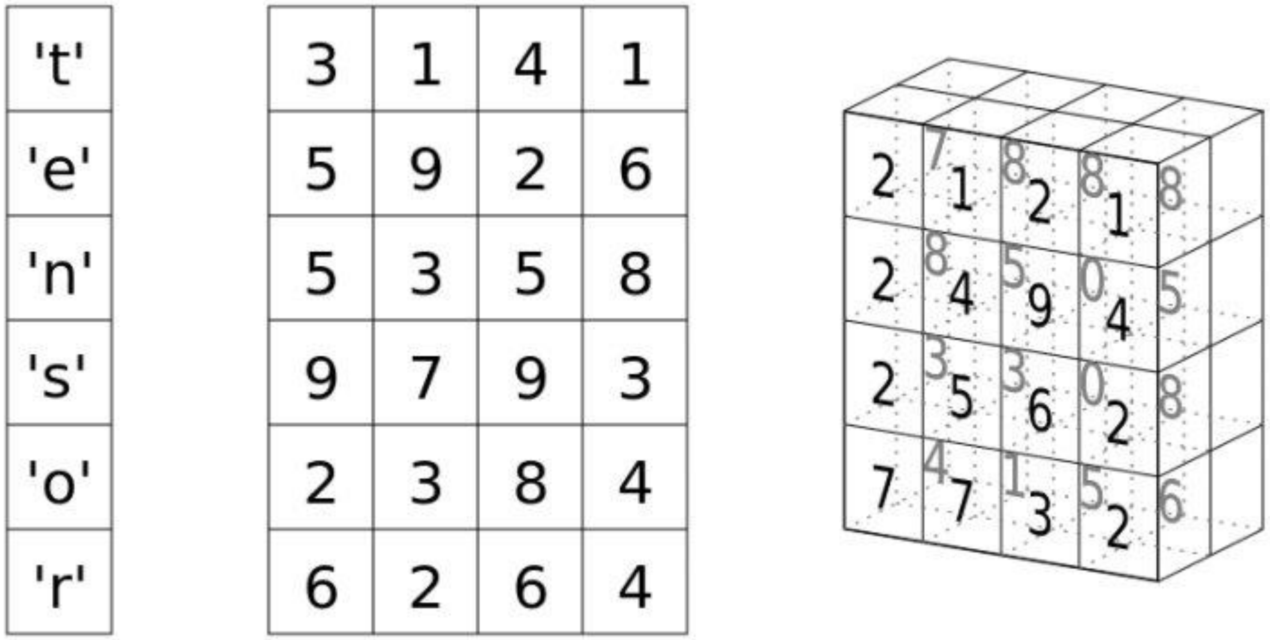
- Tensor
  - GPU에서 동작 가능
  - 계산, 그래프, 변화도 추적기능
- 자동 미분 기능 제공 (Automatic Differentiation) - Autograd




CV (batch size, width, height)
NLP (batch size, length, dim)

# Tensor

- PyTorch는 Tensor를 기반으로 작동하는 함수 제공



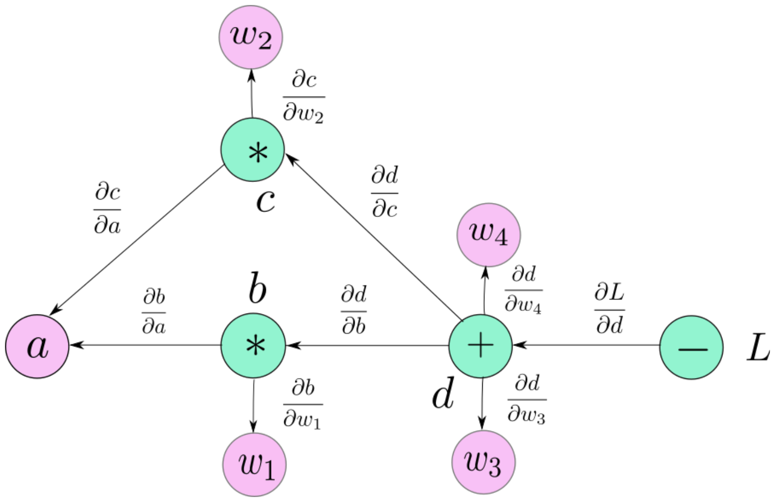

tensor of dimensions [6]
(vector of dimension 6)

tensor of dimensions [6,4]
(matrix 6 by 4)

tensor of dimensions [4,4,2]

# Autograd

- 미분자동화
- 전방향 : 비용함수 계산, input에 따른 output계산
    - 비용함수 : 예측값과 실제값의 오차
    - MSE (Mean Squared Error)
    - CEE (Cross Entropy Error)
- 역방향(Backpropagation) : 비용 함수에 따른 학습 파라미터의 변화도(기울기) 계산
    - 연쇄 규칙 이용



# Autograd

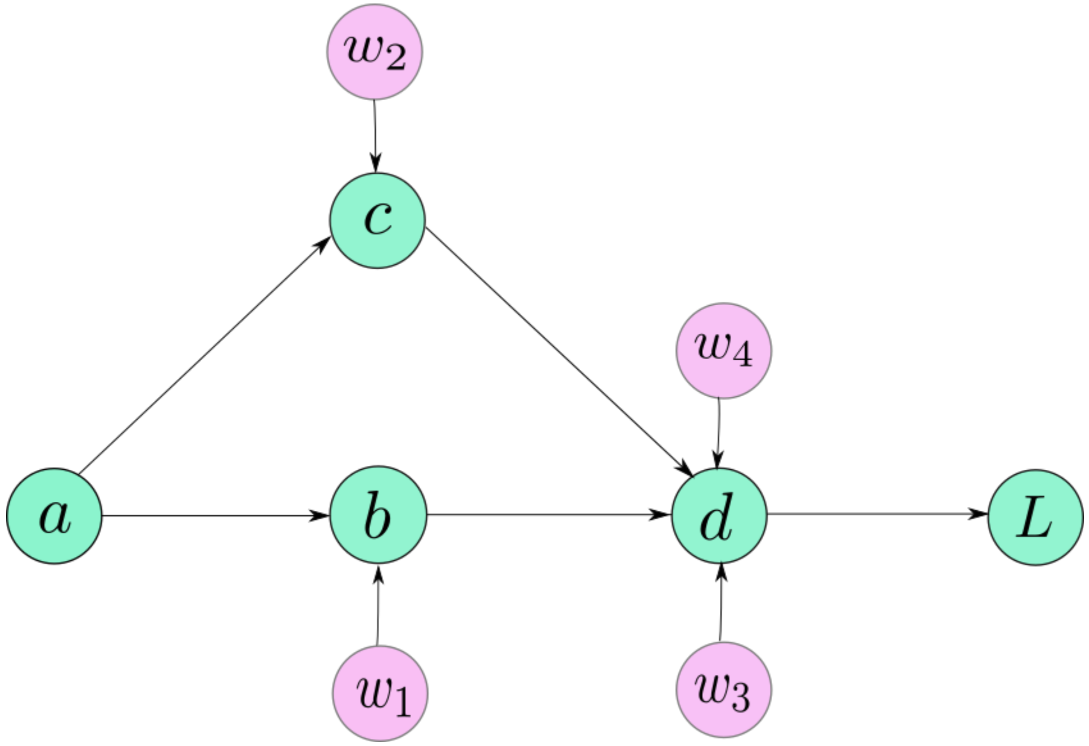
- 그래프 만들기

$$b = w_1 * a$$

$$c = w_2 * a$$

$$d = w_3 * b + w_4 * c$$

$$L = 10 - d$$



# Autograd

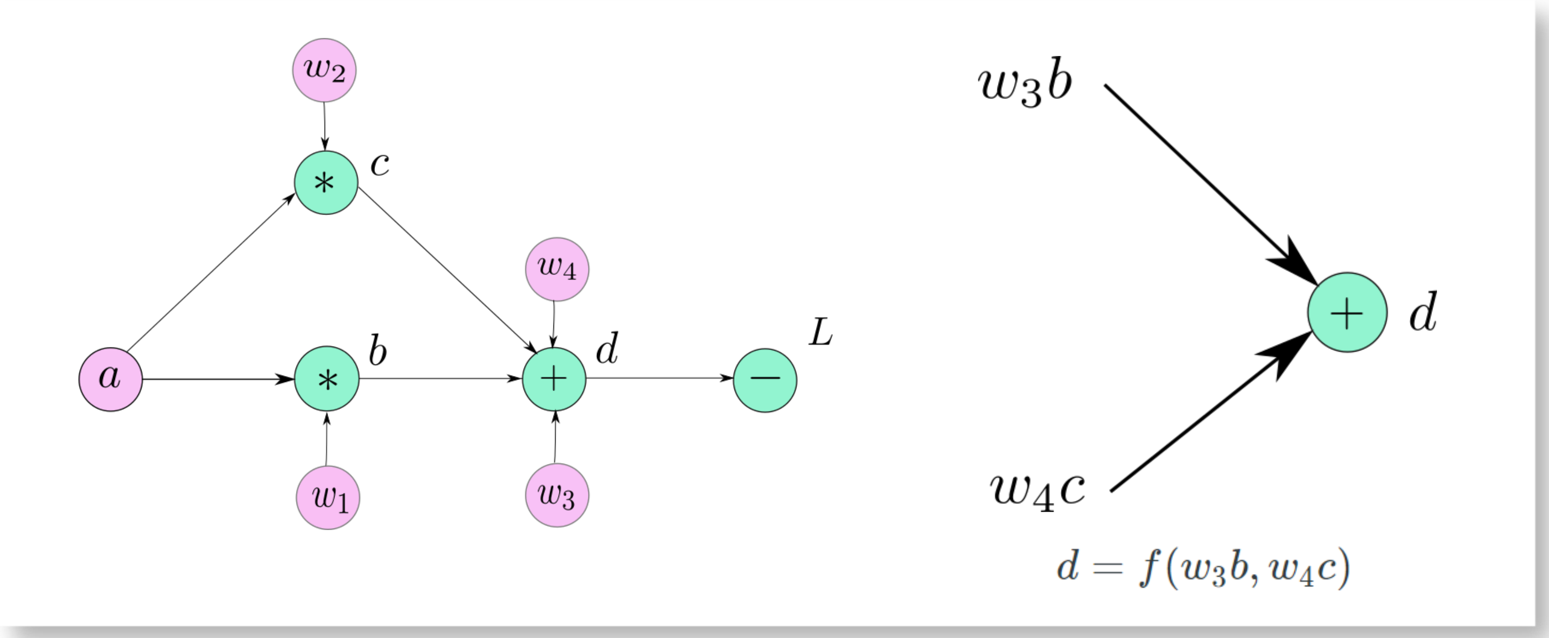
- Computing the Output
- Computing the Gradients

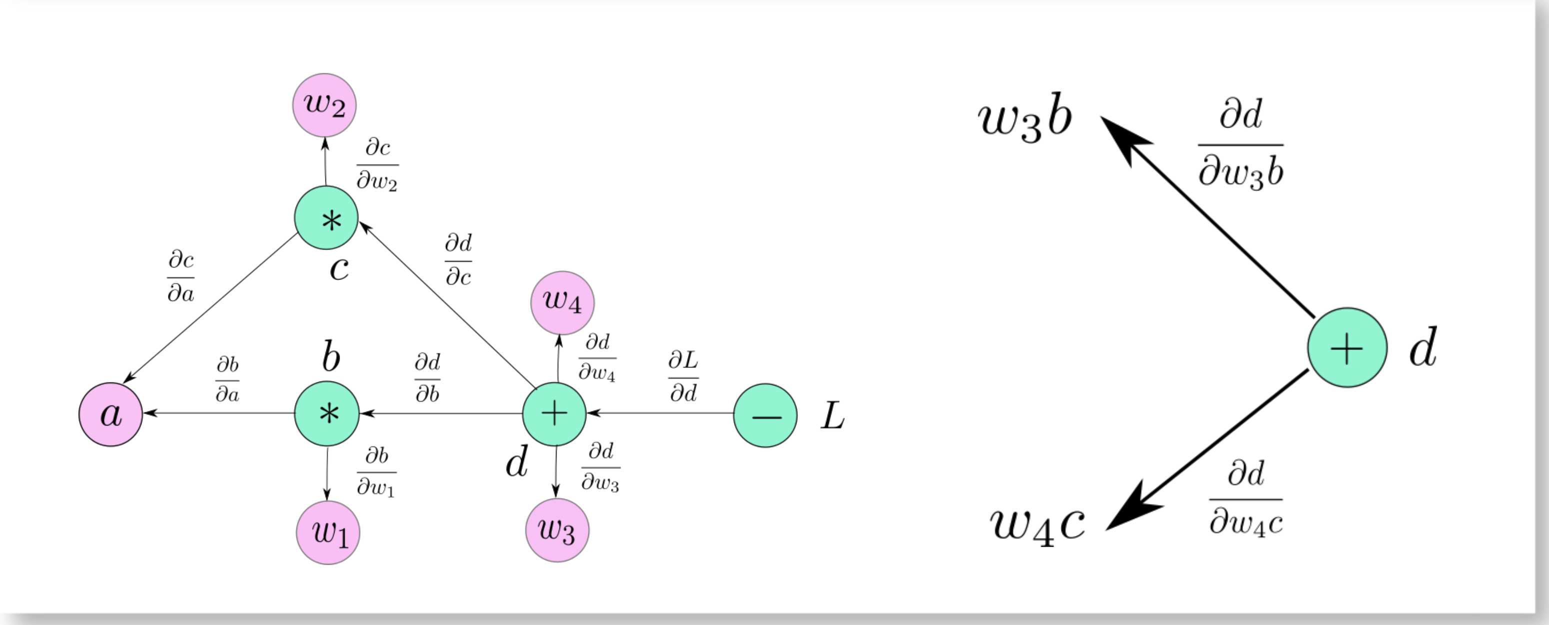
$$\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_4}$$

$$\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial w_3}$$

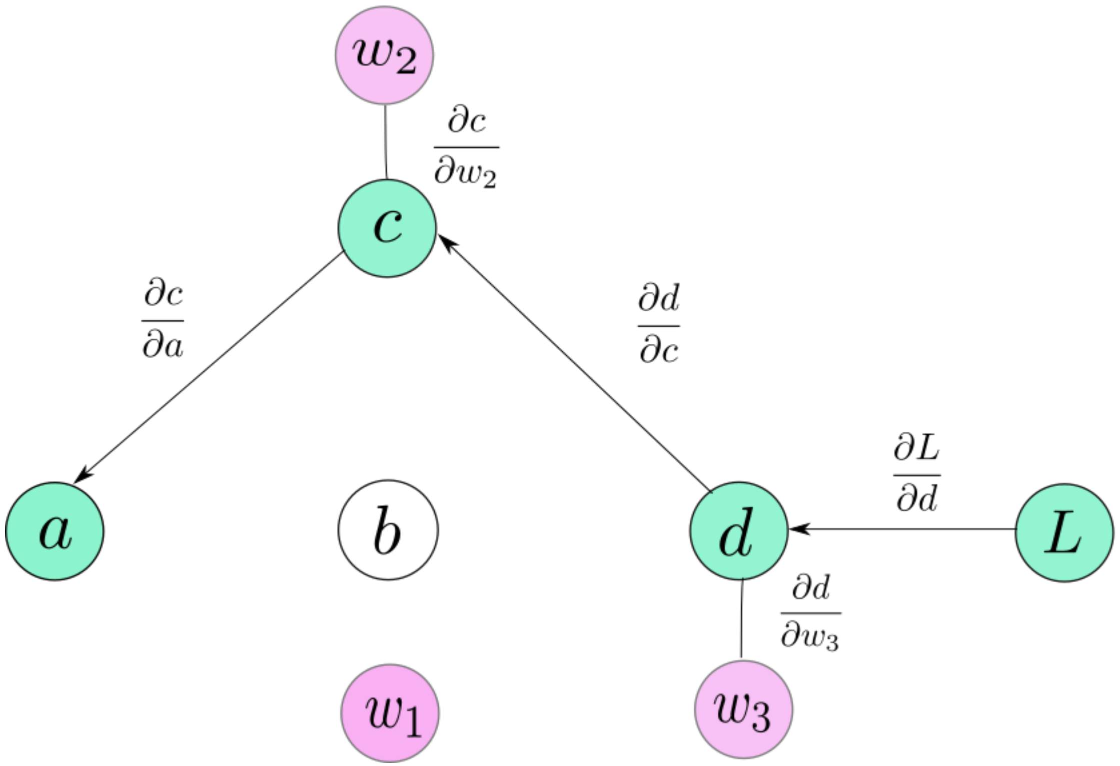
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial c} * \frac{\partial c}{\partial w_2}$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial d} * \frac{\partial d}{\partial b} * \frac{\partial b}{\partial w_1}$$



# Autograd



$w_2$
$\frac{\partial c}{\partial w_2}$
$c$
$\frac{\partial c}{\partial a}$
$\frac{\partial d}{\partial c}$
$\frac{\partial L}{\partial d}$
$a$
$b$
$d$
$L$
$\frac{\partial d}{\partial w_3}$
$w_1$
$w_3$

# PyTorch Functions

- 수식 연산
    - torch.nn.Autograd.Function 에 정의되어 있다.
- Forward Function

```
b = w1 * a
c = w2 * a
d = (w3 * b) + (w4 * c)

L = 10 - d
```

- Backward Function

```
def backward( input_grad ) :
    d.Tensor.grad = input_grad

    for i in self.inputs:
        if i.grad_fn is not None :
            new_input_grad = input_grad * local_grad( d.Tensor, i )
            i.grad_fn.backward( new_input_grad )
        else :
            pass
```

# Logistic Regression

- 입력이 대상에 속할 확률을 계산하는 방법
- 은닉층이 없다
- ( 1, batch ) 사이즈의 아웃풋

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

$$L_0 = \frac{1}{m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2 = \frac{1}{m} \sum_{i=1}^{m} (\frac{1}{1+e^{-w^T x^{(i)}}} - y^{(i)})^2$$

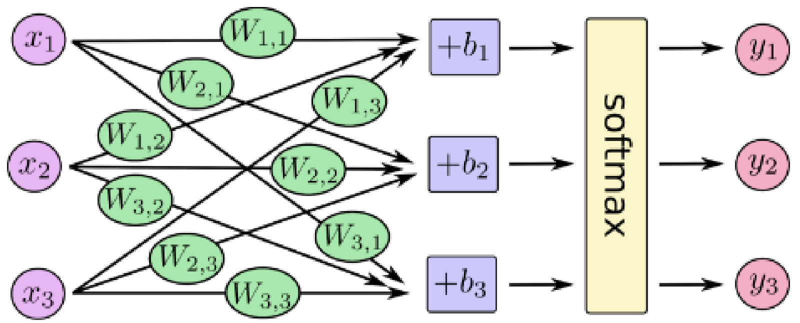
# Softmax

- Logistic Regression을 일반화 한 것

$$f(x^{(i)}, W, b) = Wx(i) + b$$

$$h(x^{(i)}) = \frac{e^{w_{y_j}^T x^{(i)}}}{\sum_{j=1}^{k} e^{w_j^T x^{(i)}}}$$

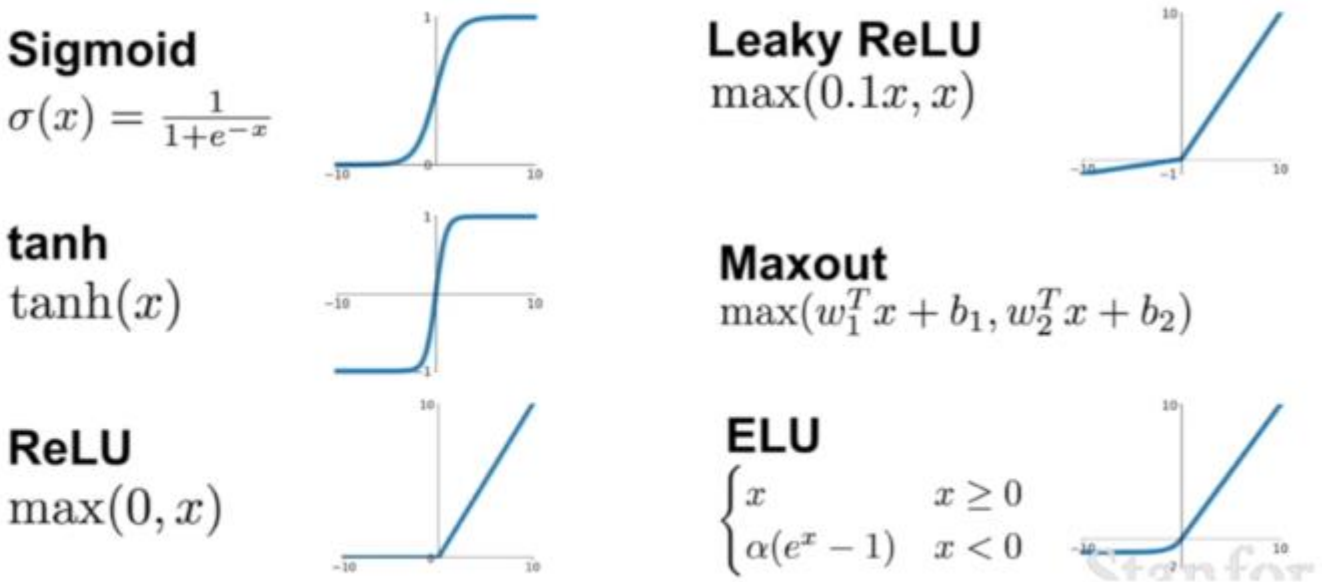
Activation Function




If we write it as an equation, we can get:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax} \begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_1 + W_{1,3}x_1 + b_1 \\ W_{2,1}x_2 + W_{2,2}x_2 + W_{2,3}x_2 + b_2 \\ W_{3,1}x_3 + W_{3,2}x_3 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

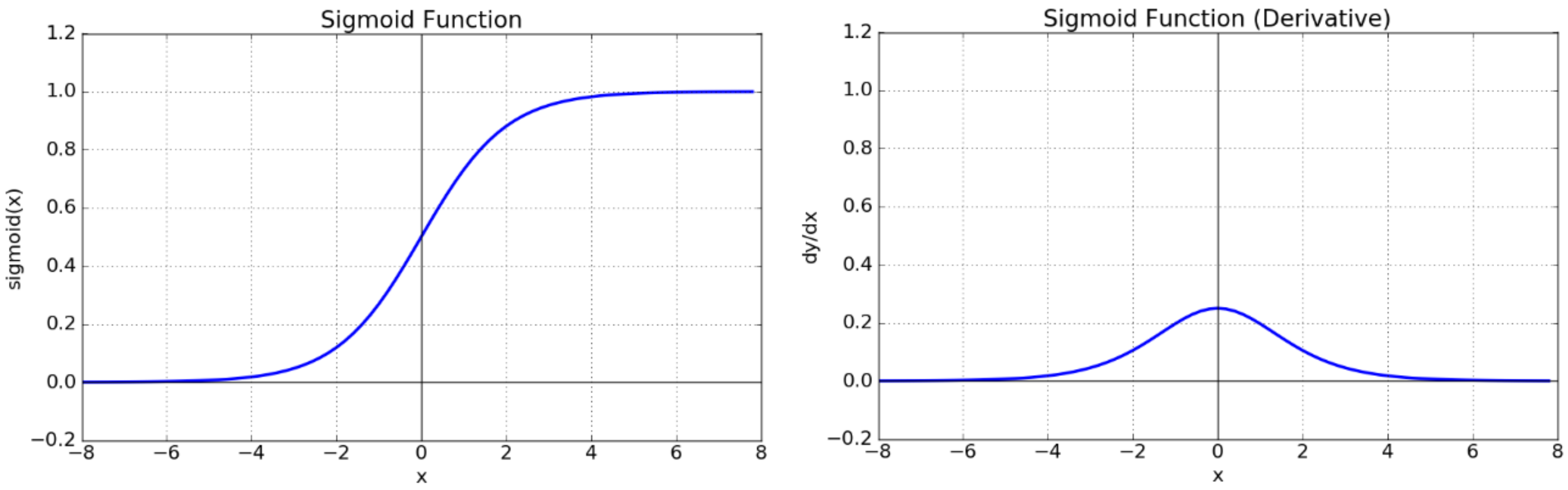
# Activation Functions



# Activation Functions

- Sigmoid

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



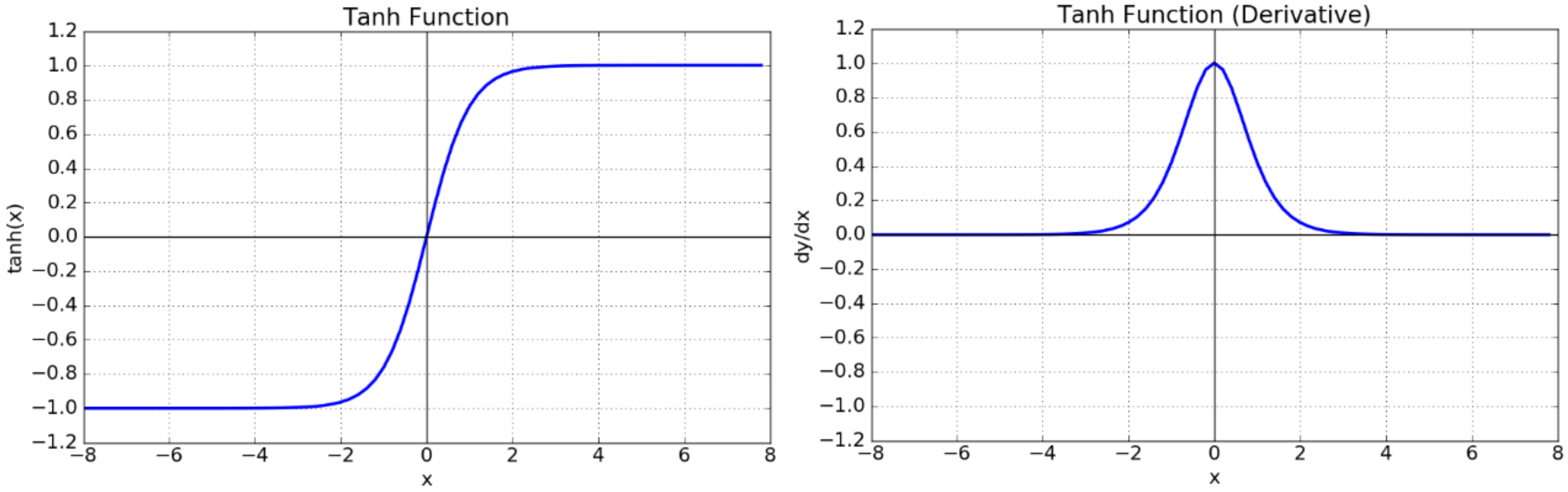
# Activation Functions

- Tanh

$$tanh(x) = 2\sigma(2x) - 1$$

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

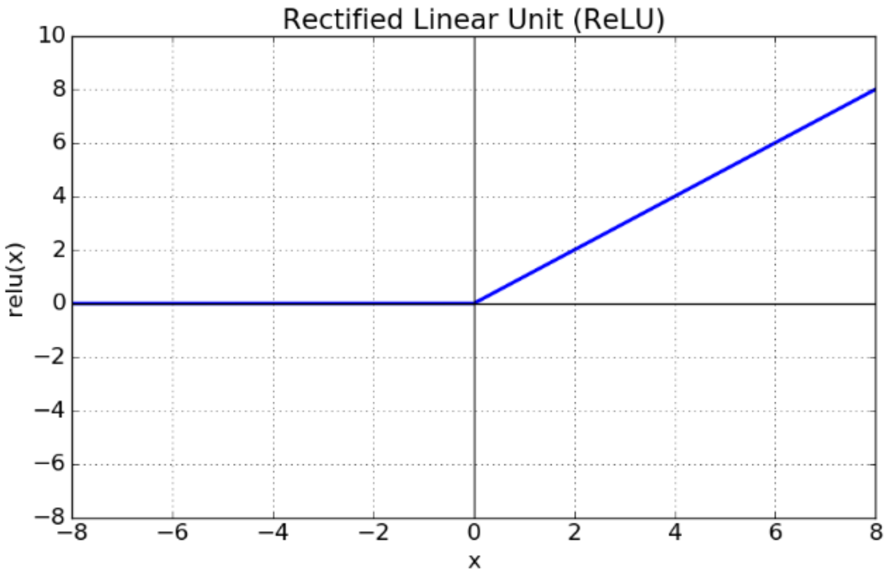
$$tanh^{'}(x) = 1 - tanh^2(x)$$



# Activation Functions

- ReLU (경사함수)
- 가장 많이 사용되는 활성화 함수
- Gradient Vanishing 문제 해결

$$f(x) = max(0, x)$$



# Thank You