[4주차] Introduction to Neural Networks

1기 김지수
1기 이선민

EURON

목차

1. REVIEW

Loss Function

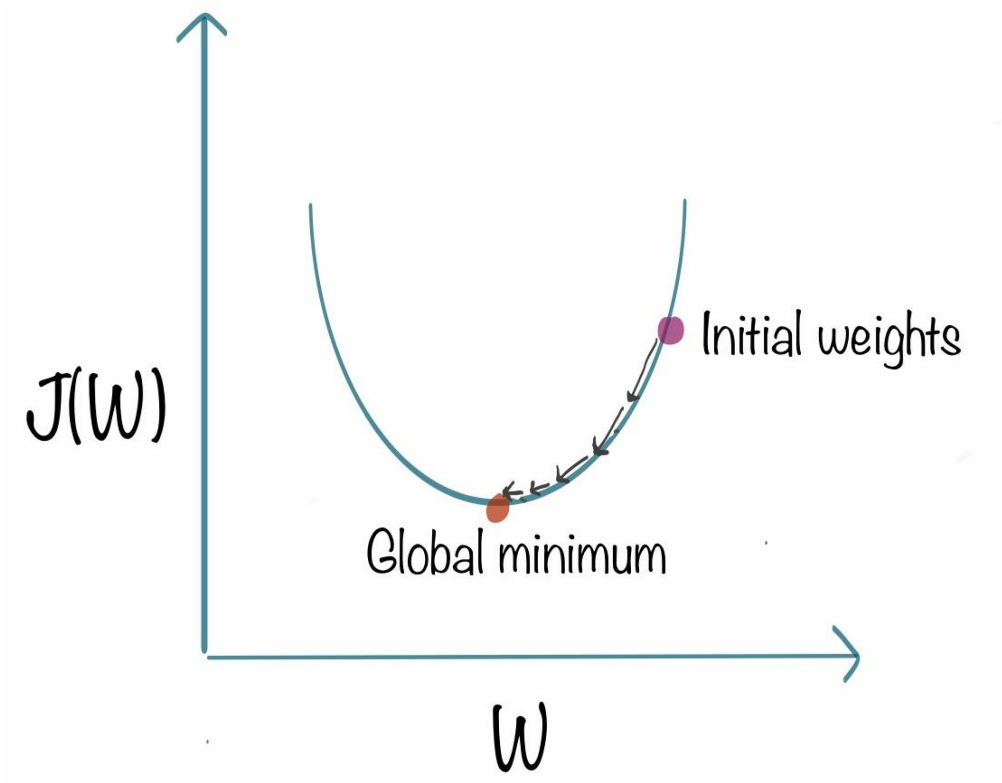
계산했을 때 얼만큼 결과가 나쁜지를 양적으로 측정해 판단하는 방법

Optimization

최적화 : loss function의 결과값을 최소화하는 모델의 W를 찾는 것

Gradient Descent

Gradient Descent



1. Numerical gradient

2. Analytic gradient

Numerical gradient

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

current W:	W + h (second dim):	gradient dW:
[0.34,	[0.34,	[-2.5,
-1.11,	-1.11 + **0.0001**,	**0.6**,
0.78,	0.78,	?,
0.12,	0.12,	?,
0.55,	0.55,	
2.81,	2.81,	
-3.1,	-3.1,	
-1.5,	-1.5,	
0.33,…]	0.33,…]	?,…]
loss 1.25347	**loss 1.25353**	

(1.25353 - 1.25347)/0.0001
= 0.6

$$\boxed{\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}}$$

Analytic gradient

: 미분으로 공식을 유도해 gradient를 계산하는 방법

$$f(x) = x^2 \quad \rightarrow \quad f'(x) = 2x$$

$$f(x) = e^x \quad \rightarrow \quad f'(x) = e^x$$

2. Derivative = 기울기?

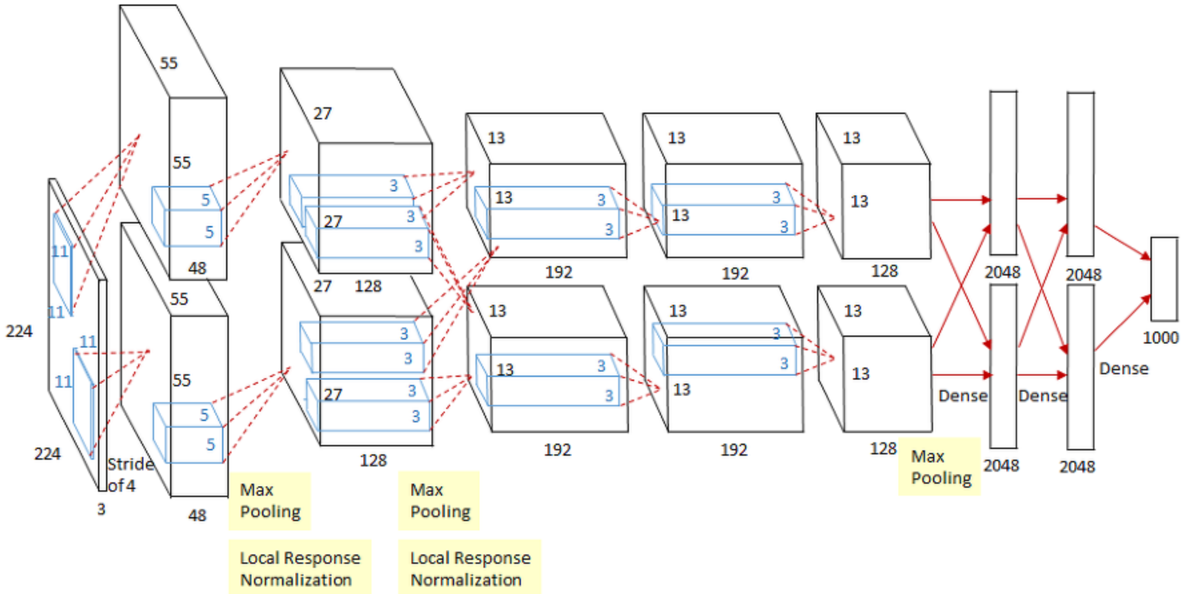
Gradient = 다변수 함수의 **모든 입력값**에서 **모든 방향에** 대한 **순간변화율**
= **편미분값의 벡터**

$$f(x) = x^2 \quad \rightarrow \quad f'(x) = 2x$$

$$f(x, y) = x + y \quad \rightarrow \quad \frac{\partial f}{\partial x} = 1\,, \quad \frac{\partial f}{\partial y} = 1$$

$$\nabla f = [\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}] = [1\,, 1]$$

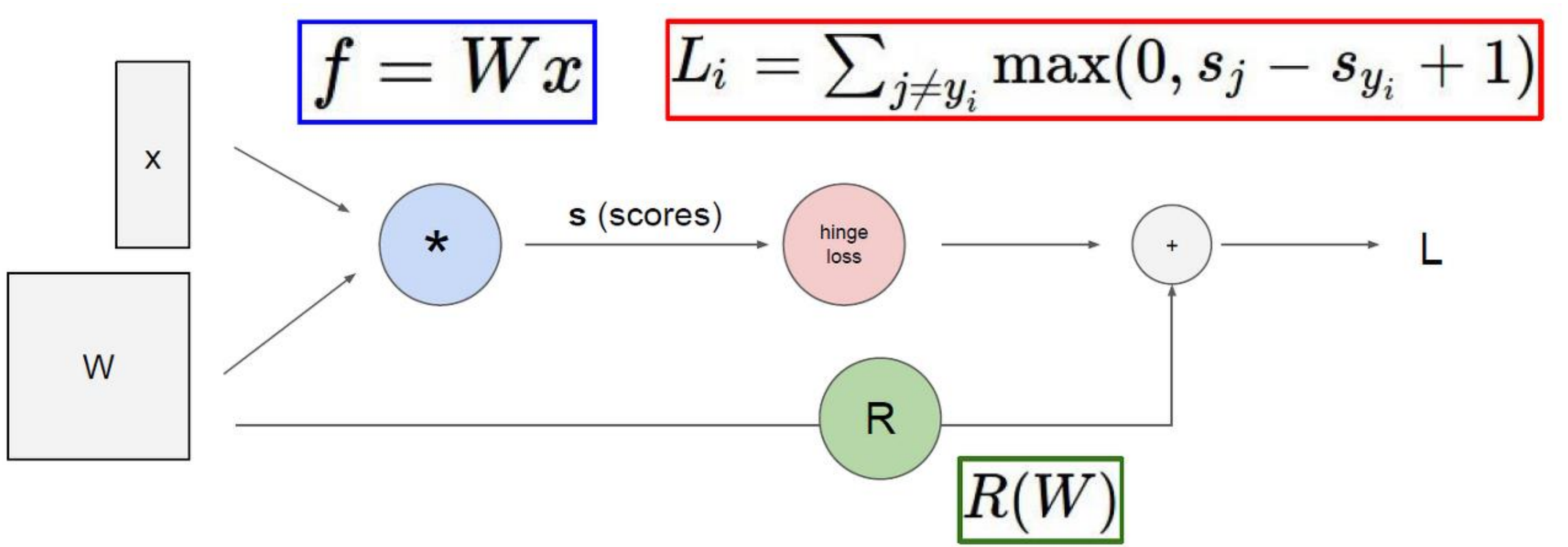
1. REVIEW



< alexnet >

2. Computational Graph

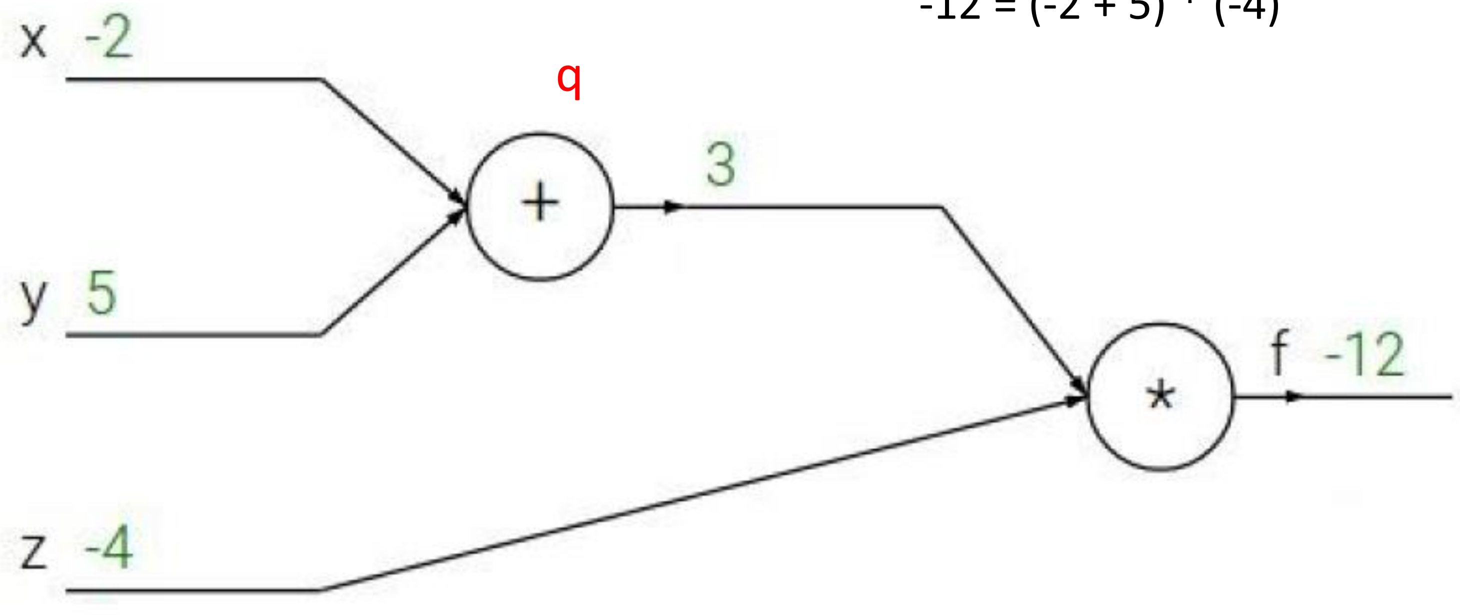
: 어떤 함수의 일련의 연산 과정을 그래프로 나타낸 것



2. Computational Graph

$$f(x, y, z) = (x + y)z$$

-12 = (-2 + 5) * (-4)



2. Computational Graph

Backpropagation: a simple example

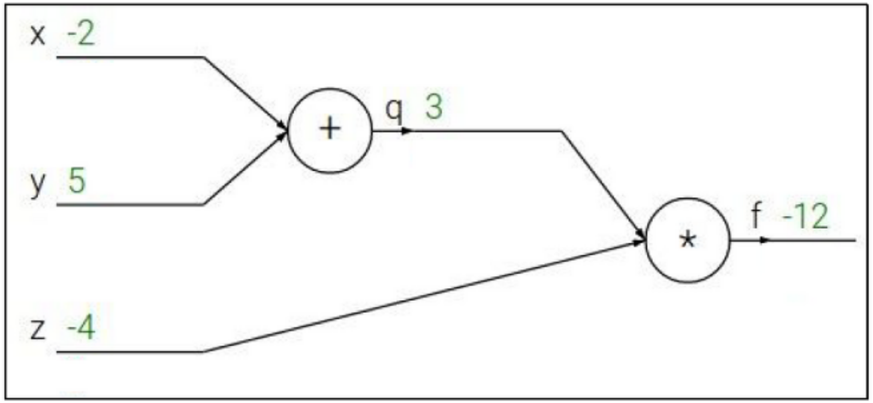
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

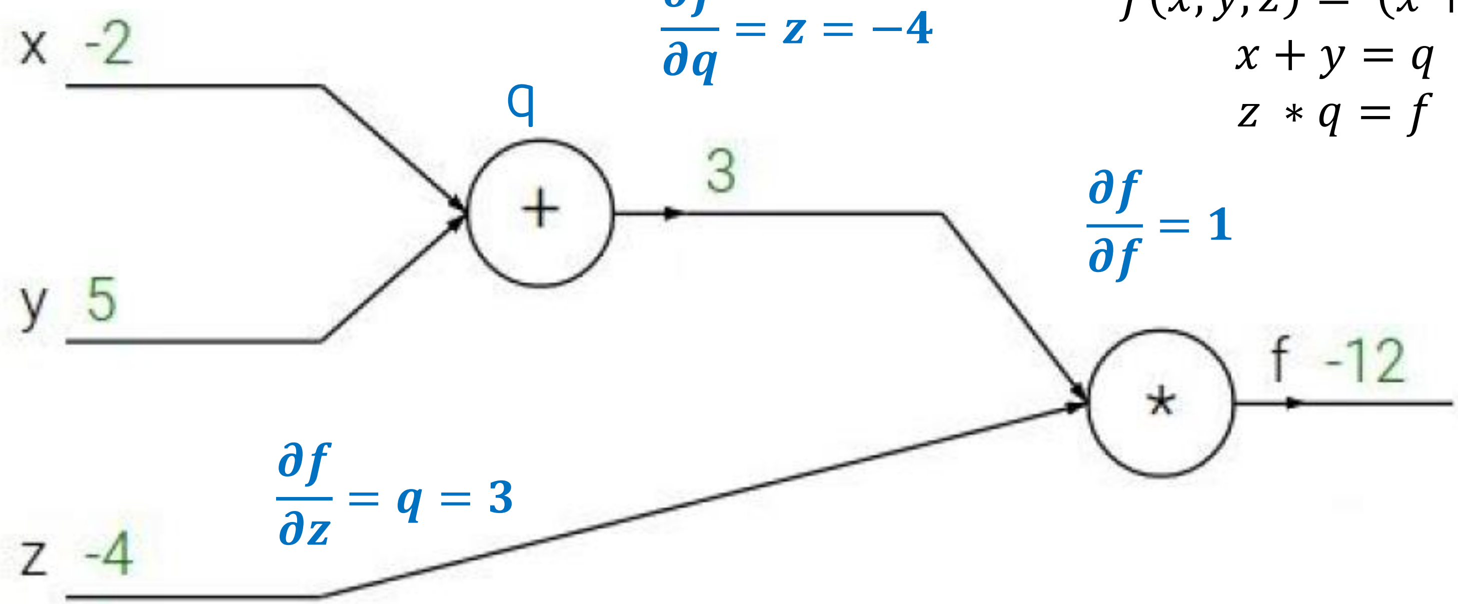
Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



3. Backward Propagation

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

$$f(x,y,z) = (x+y)z$$
$$x + y = q$$
$$z * q = f$$



$$\frac{\partial f}{\partial q} = z = -4$$

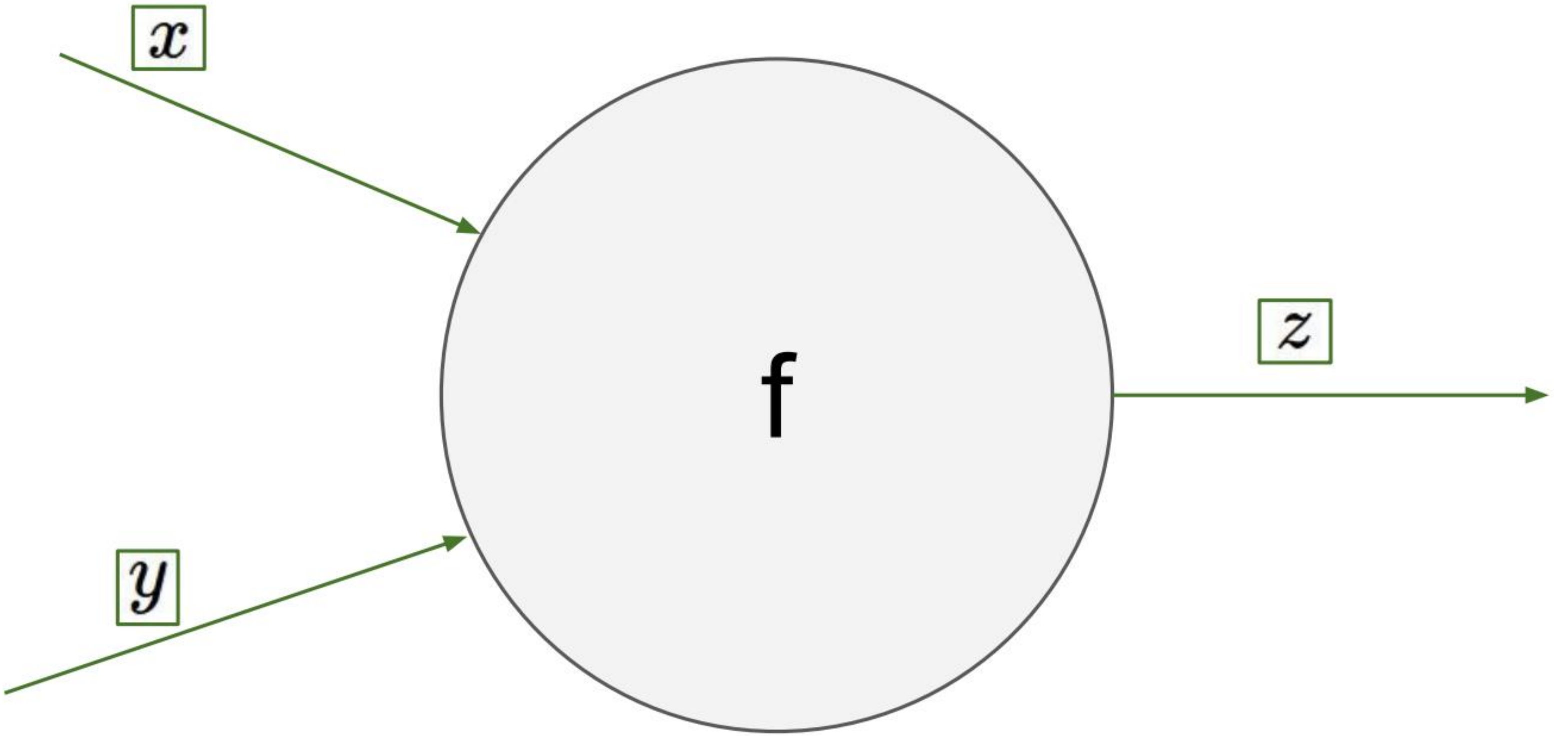
$$\frac{\partial f}{\partial f} = 1$$

$$\frac{\partial f}{\partial z} = q = 3$$

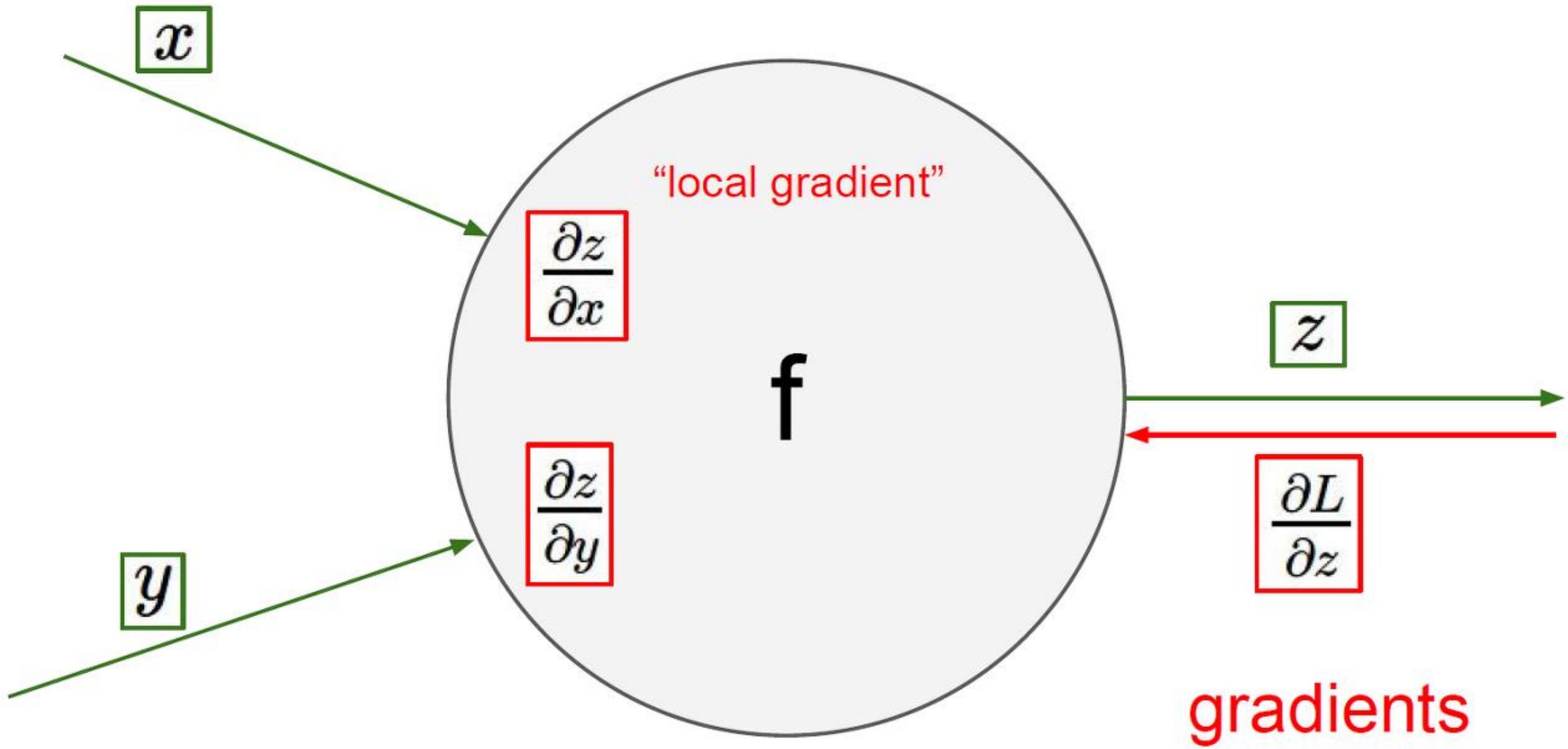
$$\frac{\partial f}{\partial x} = ?$$

$$\frac{\partial f}{\partial x} = \cancel{\frac{\partial f}{\partial q} * \frac{\partial q}{\partial x}} = -4 * 1$$

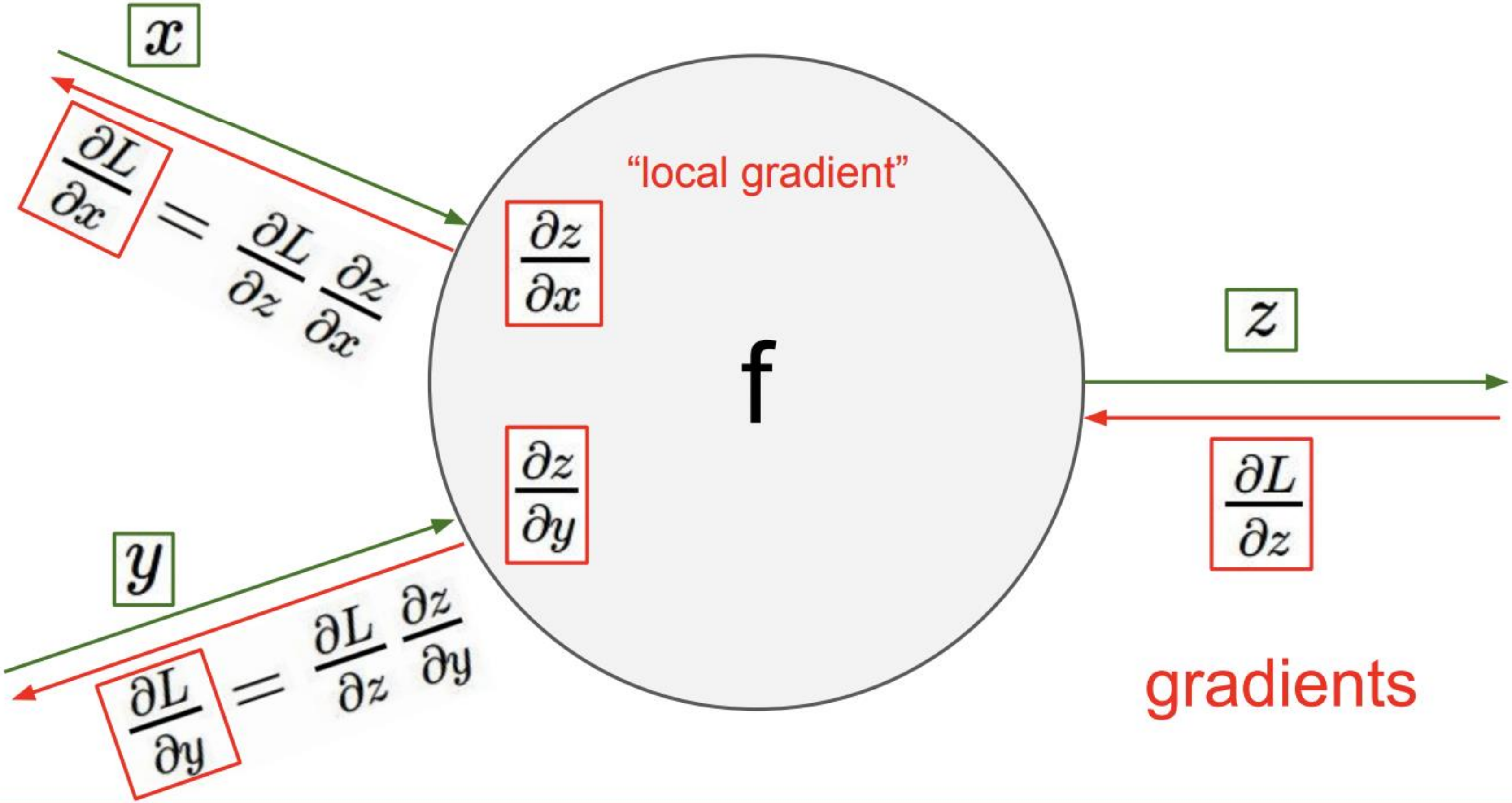
3. Backward Propagation



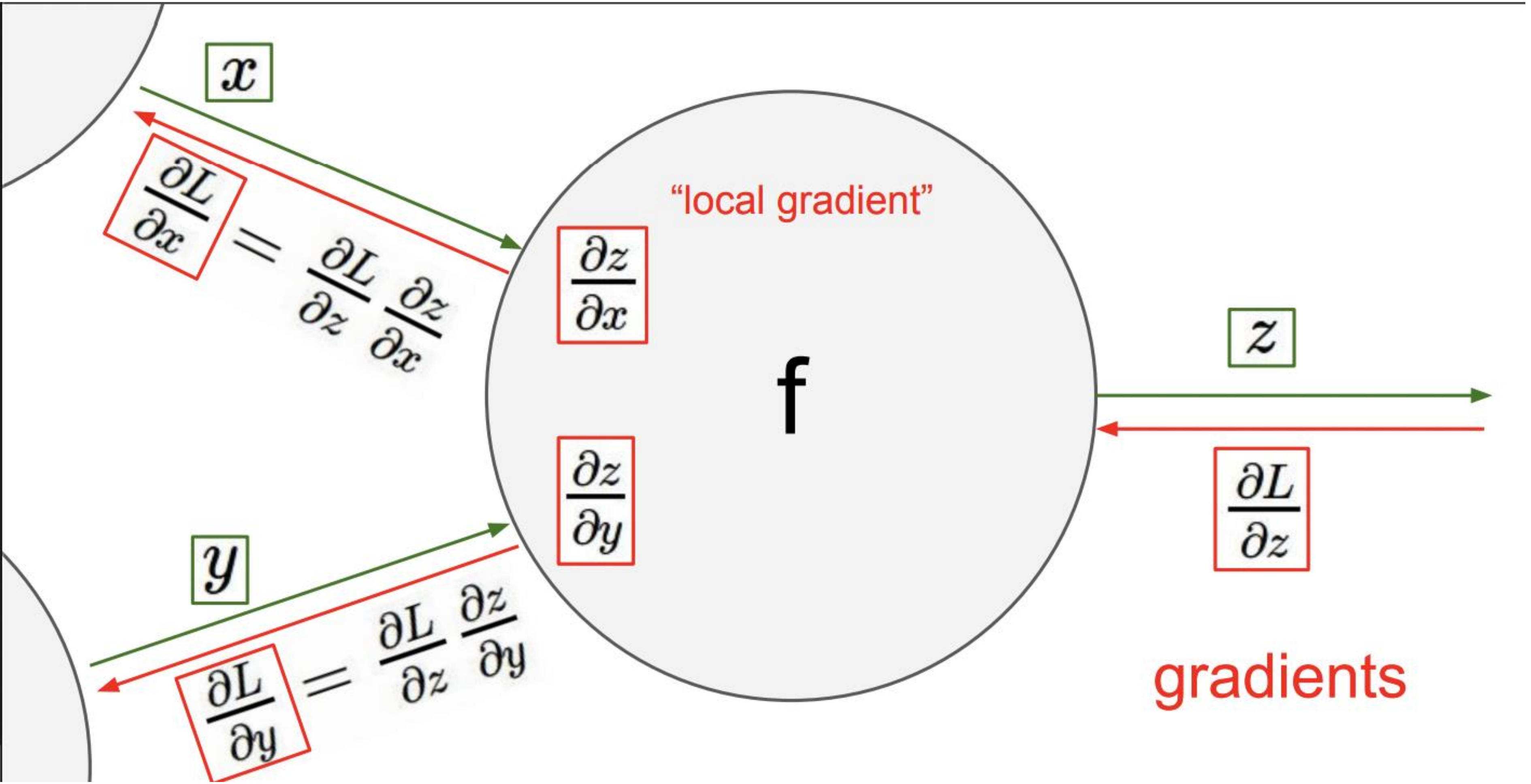
3. Backward Propagation



3. Backward Propagation

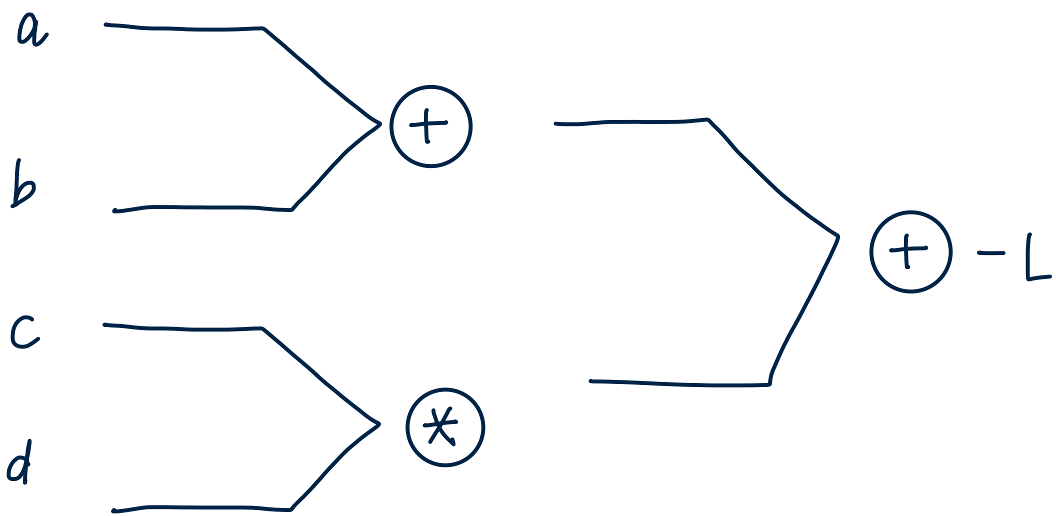


3. Backward Propagation



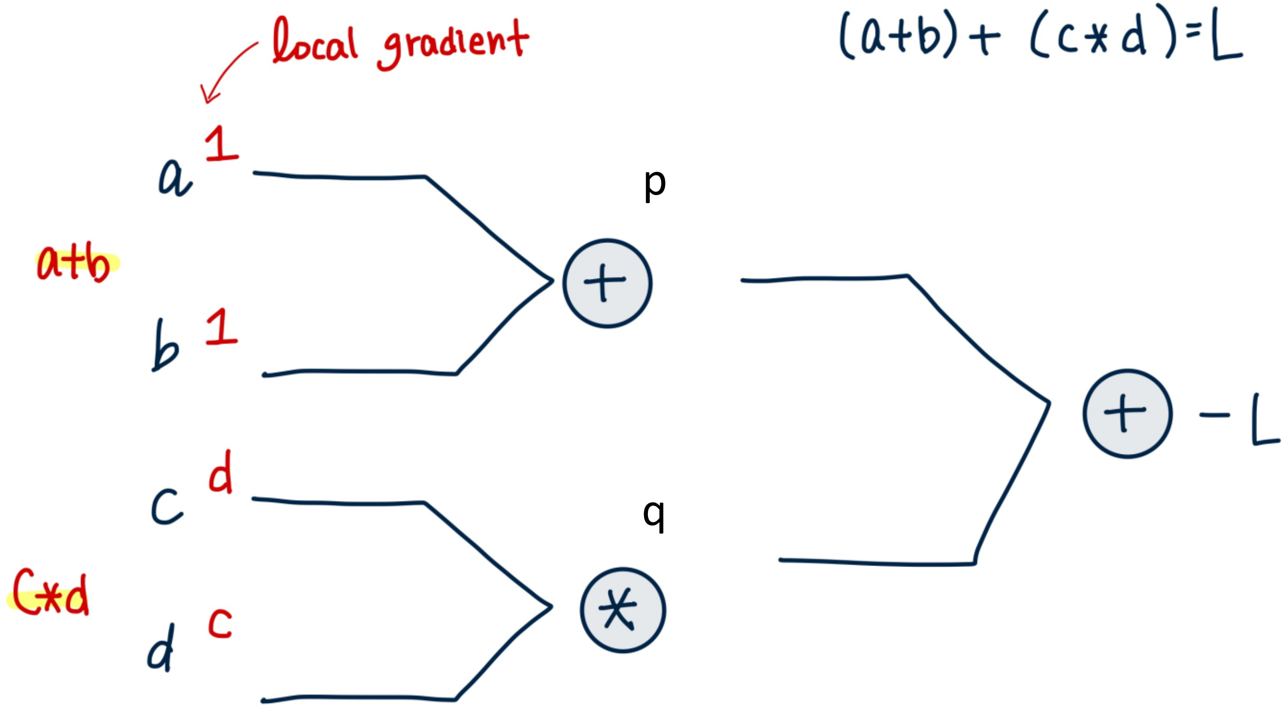
3. Backward Propagation

$(a+b)+(c*d)=L$



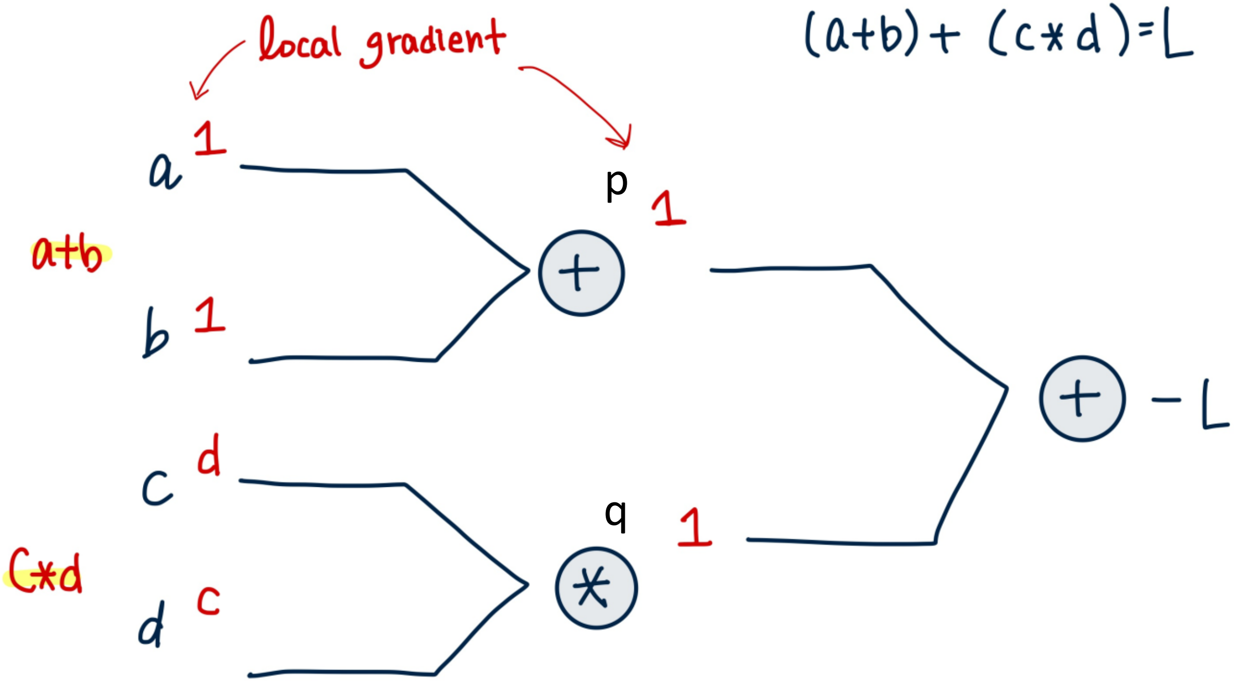
3. Backward Propagation

$(a+b)+(c*d)=L$

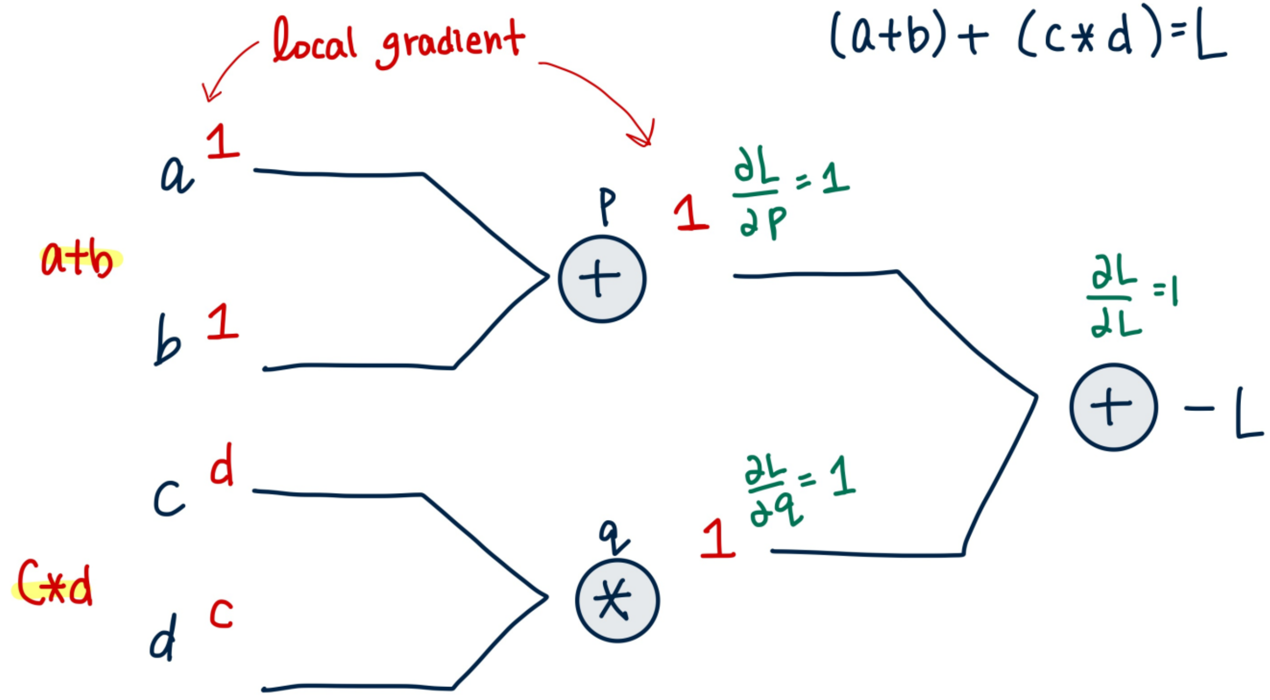


3. Backward Propagation

$(a+b)+(c*d)=L$



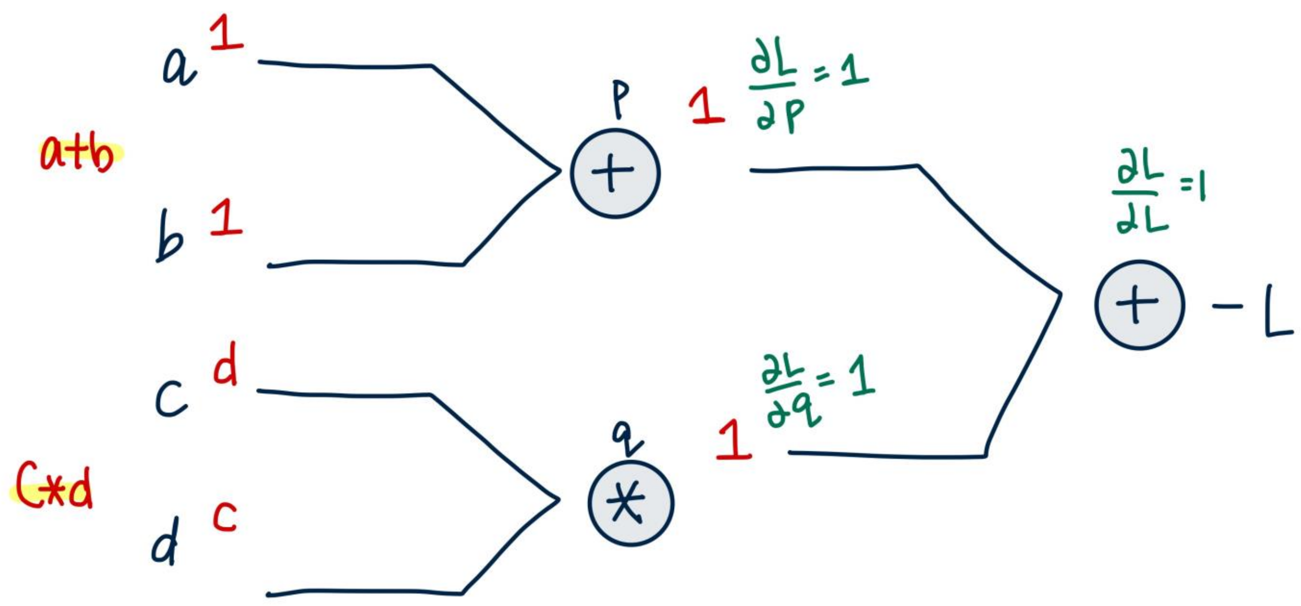
3. Backward Propagation



3. Backward Propagation

$$\frac{\partial L}{\partial a} = \frac{\partial L}{\partial p} \times \frac{\partial p}{\partial a} \Rightarrow 1 \times 1 = 1$$

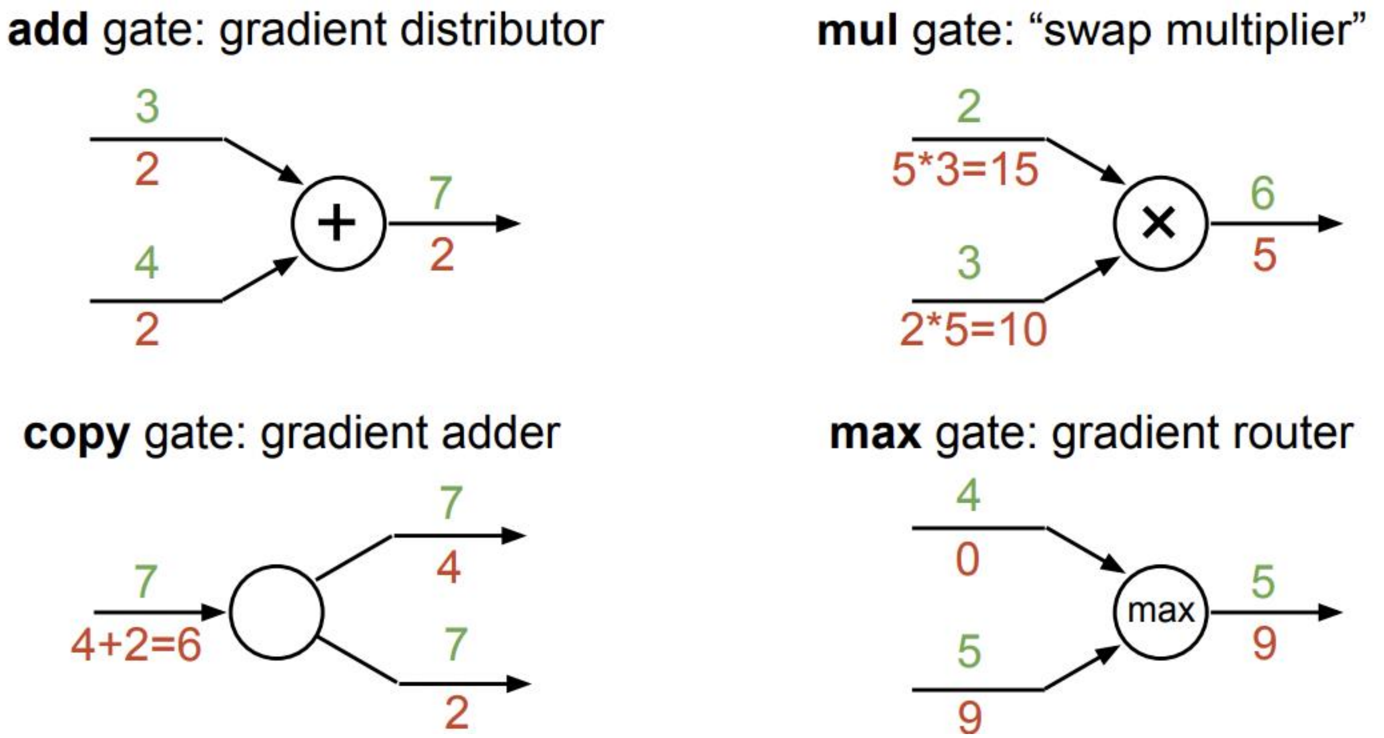
$$(a+b) + (c * d) = L$$



"Forward Pass - Backward Propagation"

3. Gate

연두 : 입력값
빨강 : gradient



local gradient = upstream에서 온 **gradient1** +
upstream에서 온 **gradient2**

4. Code Review

```
w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the
sigmoid gradient derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the
circuit
```

4. Code Review

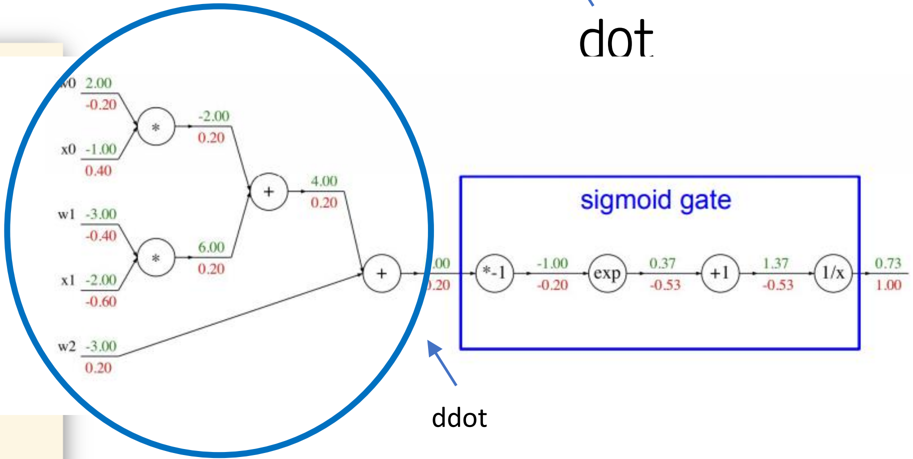
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

dot

```python
w = [2,-3,-3] # assume some random weights and data
x = [-1, -2]

# forward pass
dot = w[0]*x[0] + w[1]*x[1] + w[2]
f = 1.0 / (1 + math.exp(-dot)) # sigmoid function

# backward pass through the neuron (backpropagation)
ddot = (1 - f) * f # gradient on dot variable, using the
sigmoid gradient derivation
dx = [w[0] * ddot, w[1] * ddot] # backprop into x
dw = [x[0] * ddot, x[1] * ddot, 1.0 * ddot] # backprop into w
# we're done! we have the gradients on the inputs to the
circuit
```

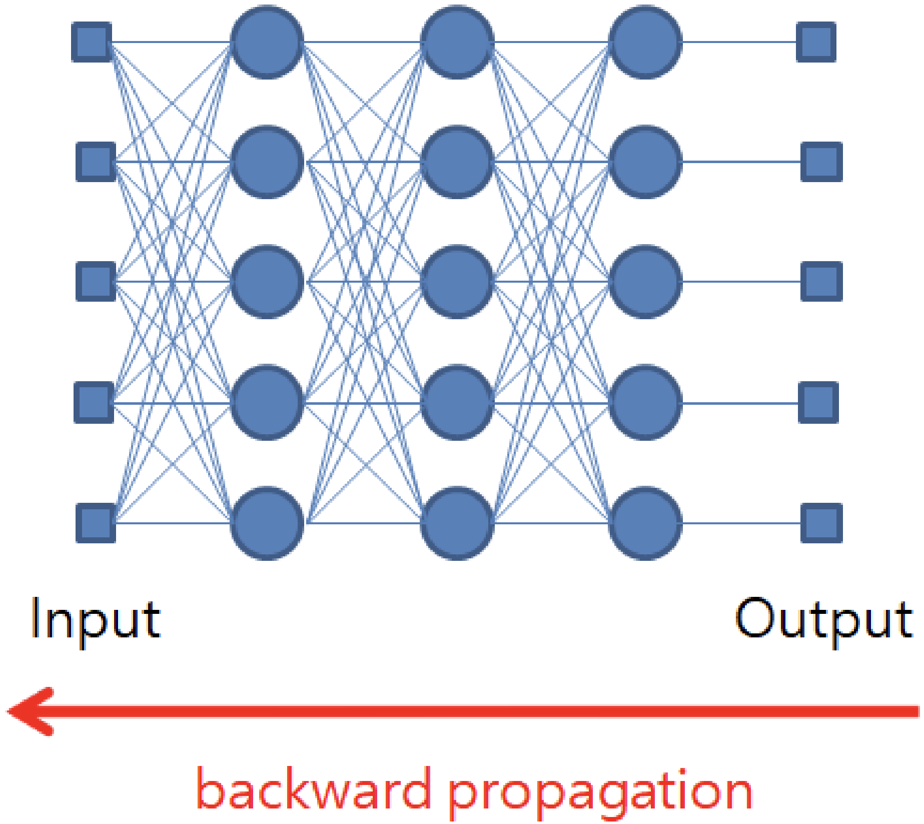


ddot

〈 Sigmoid $\sigma(x)$ gradient derivation 〉

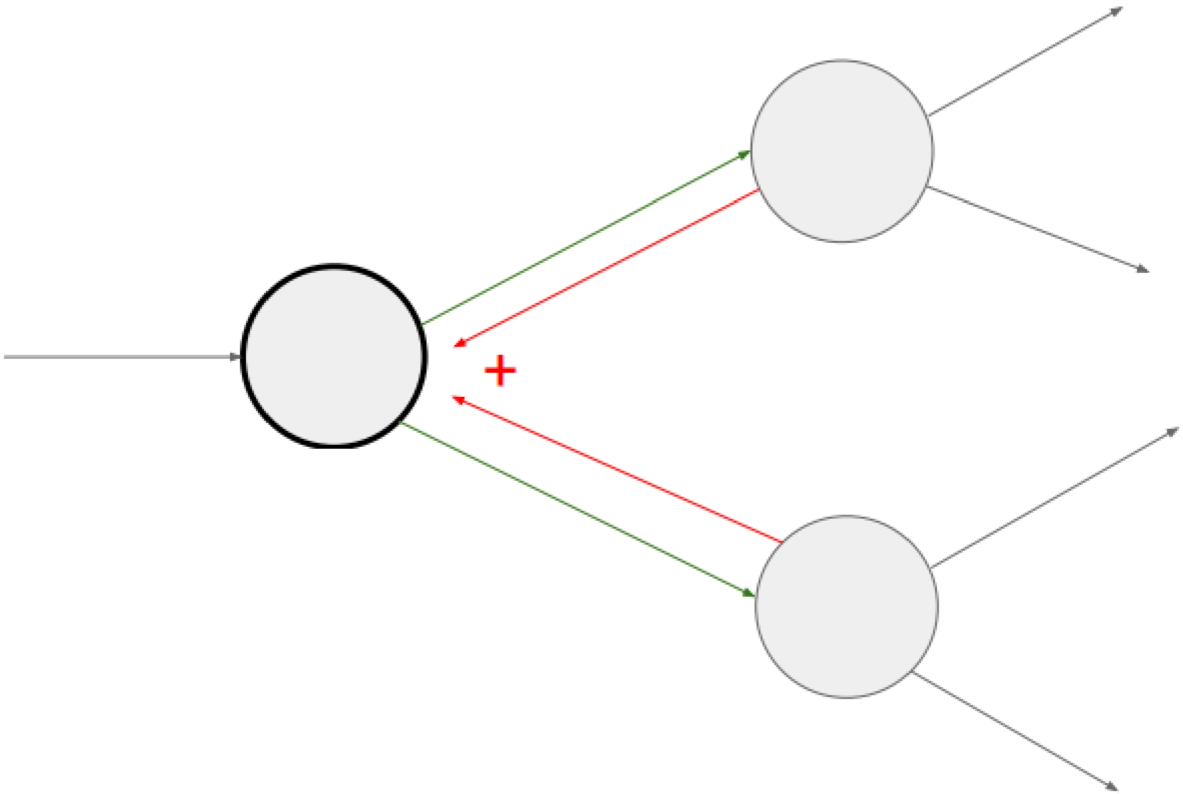
$$\sigma'(x) = \sigma(x)(1-\sigma(x))$$

4. Back Propagation (vector)

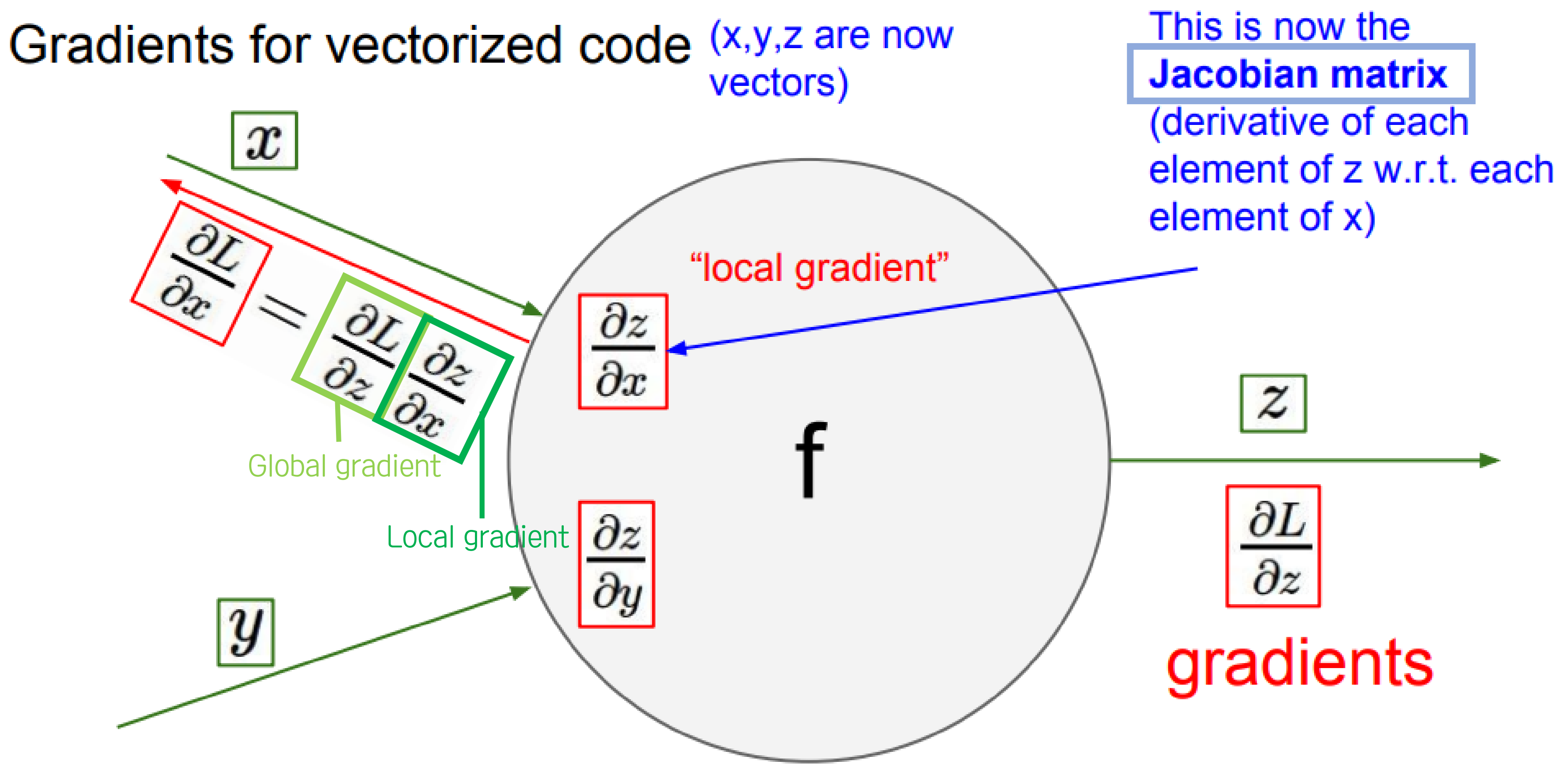


1. Back Propagation (vector)

Gradient가 1개가 아니라 “여러 개” 라면?



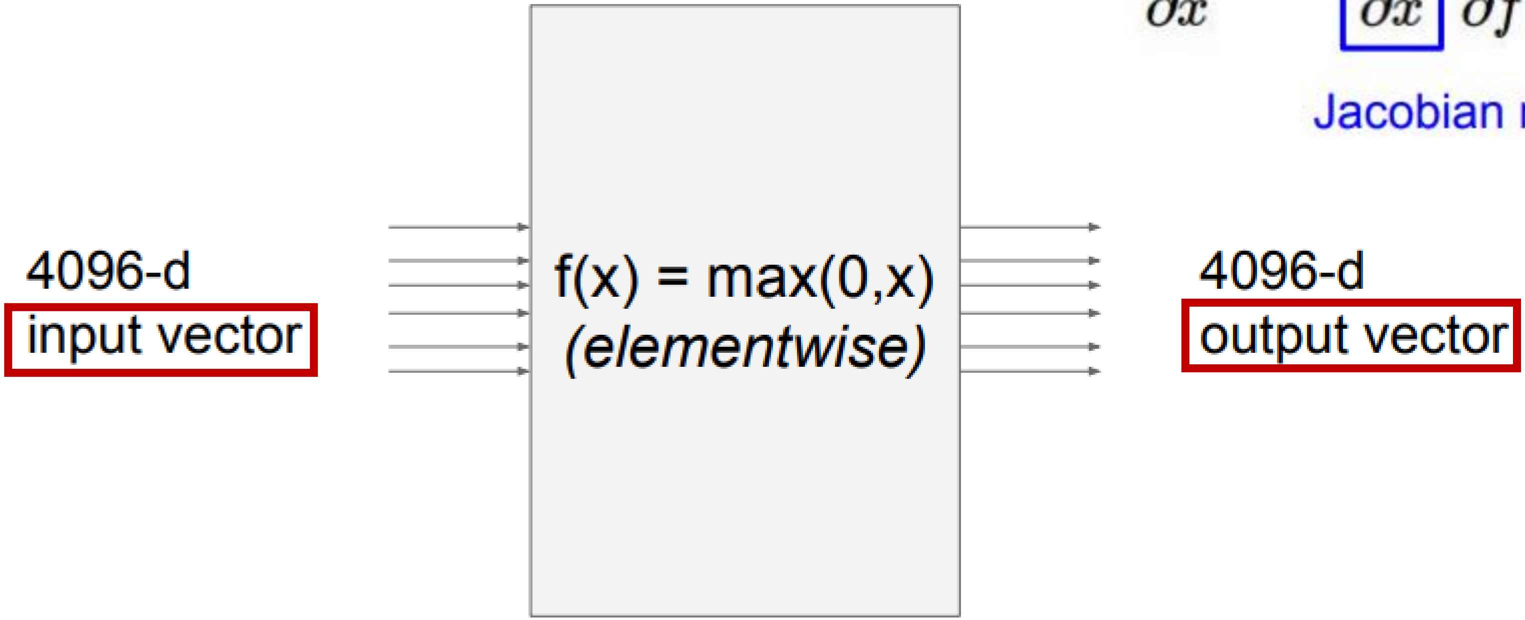
2. Jacobian Matrix



3. Vectorized Operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix



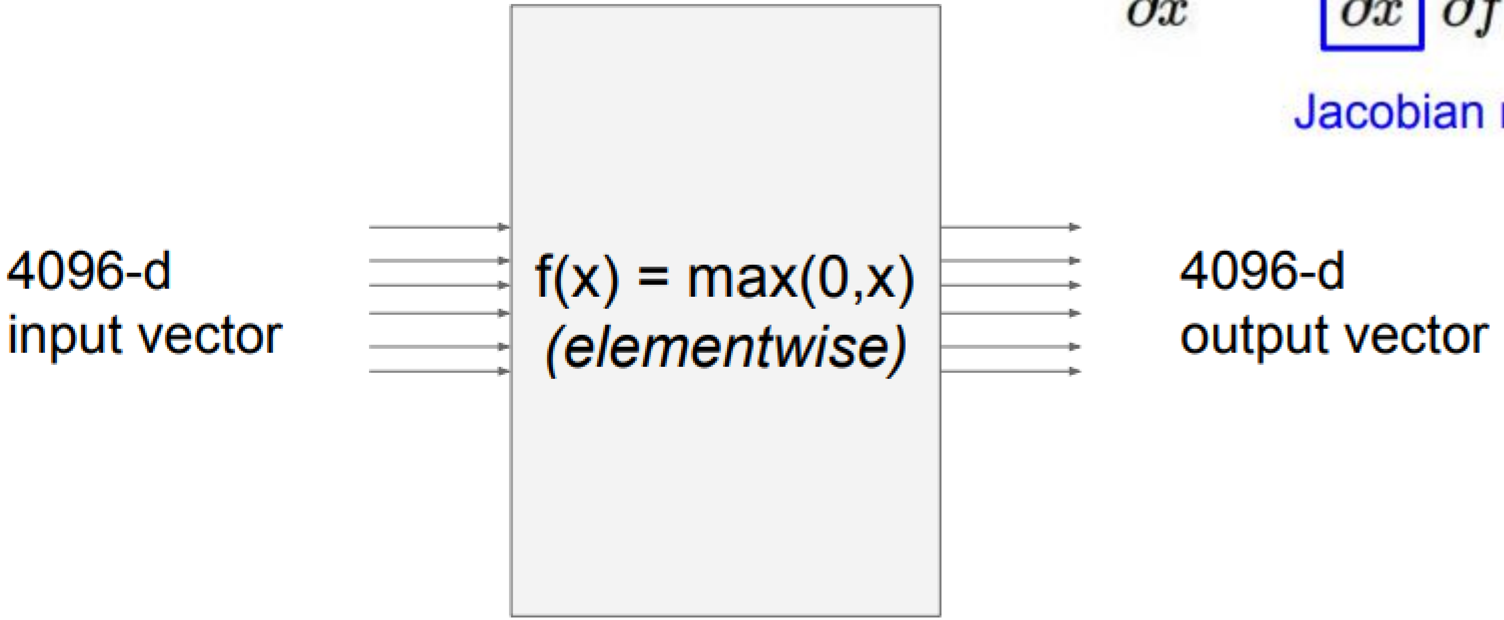
Q) Jacobian matrix의 크기는?

A) Input vector (4096 x 1) * Output vector (4096 x 1)
= 4096 x 4096

3. Vectorized Operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

Jacobian matrix



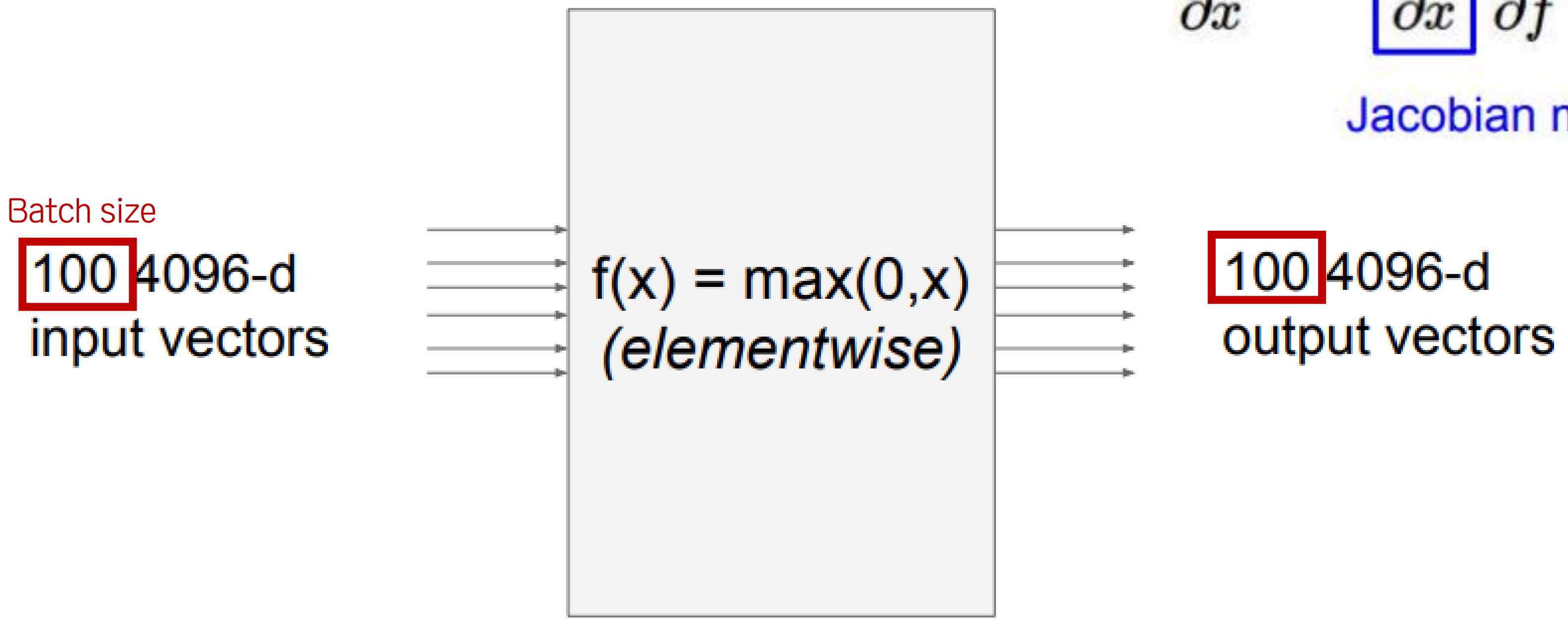
Q) Jacobian matrix의 형태는?

A) 대각선에 1,0 이 혼재되어 있는 형태인 대각행렬

3. Vectorized Operations

$$\frac{\partial L}{\partial x} = \boxed{\frac{\partial f}{\partial x}} \frac{\partial L}{\partial f}$$

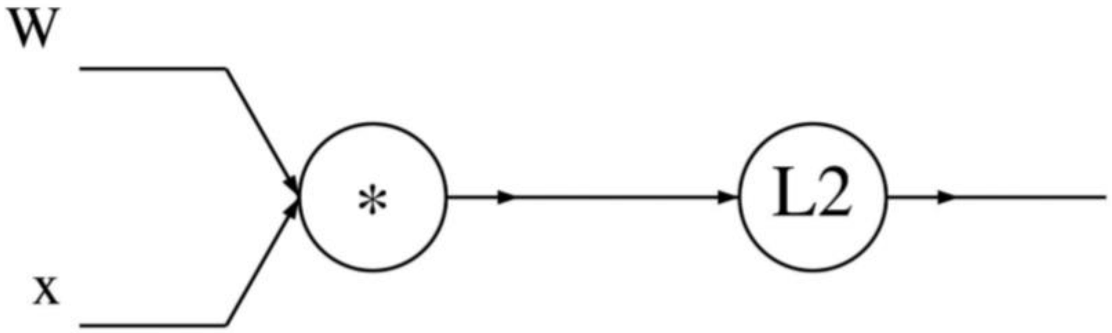
Jacobian matrix



Jacobian matrix = [409600 x 409600]

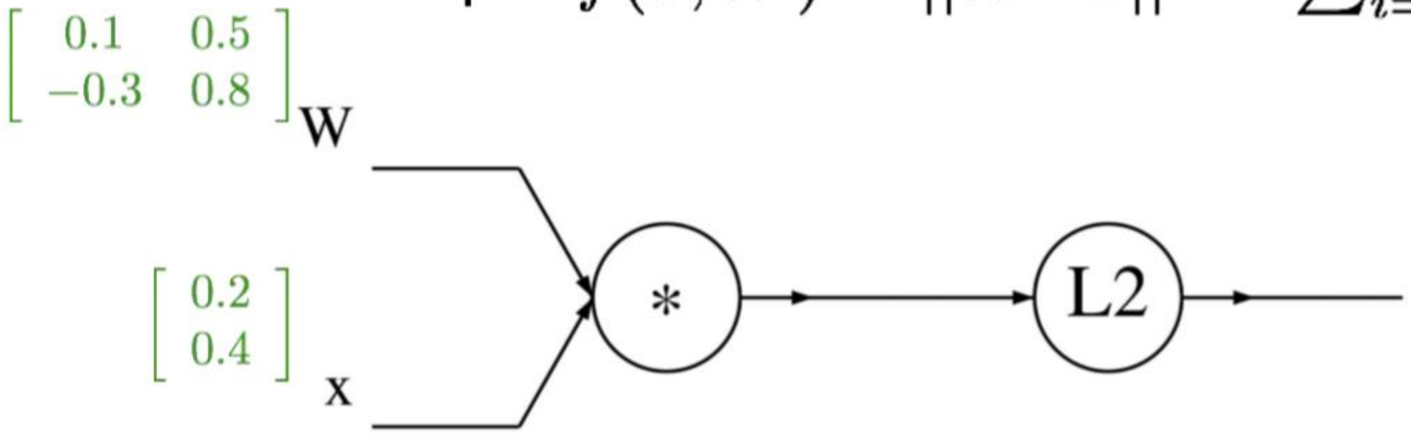
4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

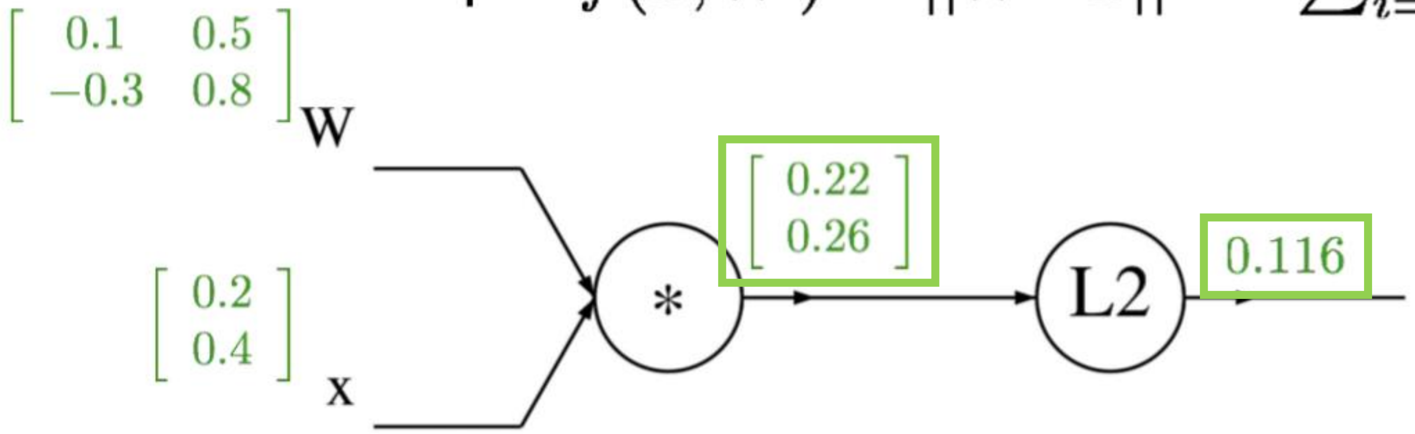


$$\boxed{q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

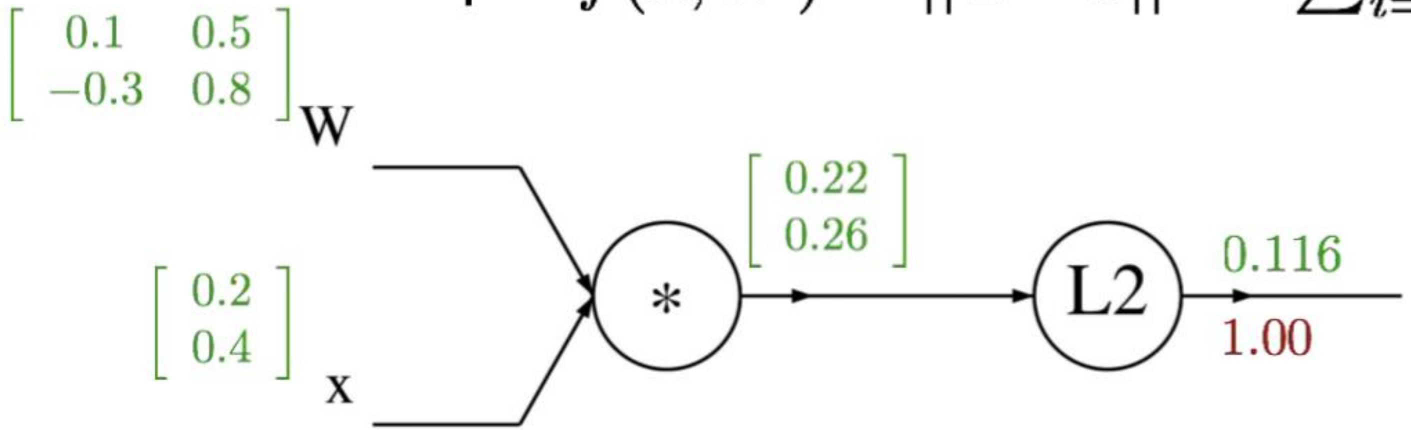
0.1 x 0.2 + 0.5 x 0.4 = 0.22

- 0.3 x 0.2 + 0.8 x 0.4 = 0.26

$(0.22)_2 + (0.26)_2 = 0.116$

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

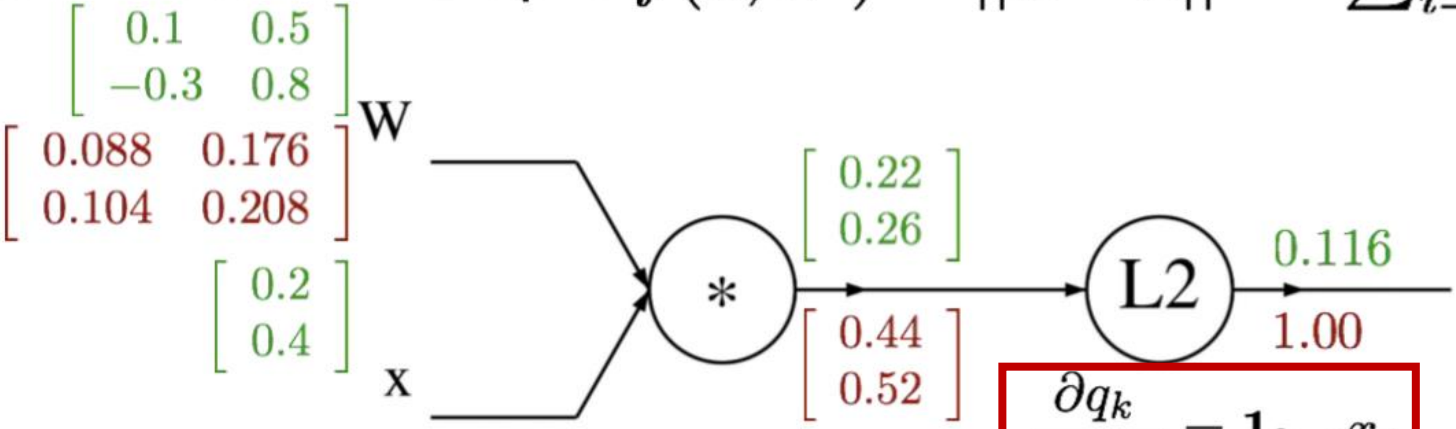
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\boxed{\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j}$$

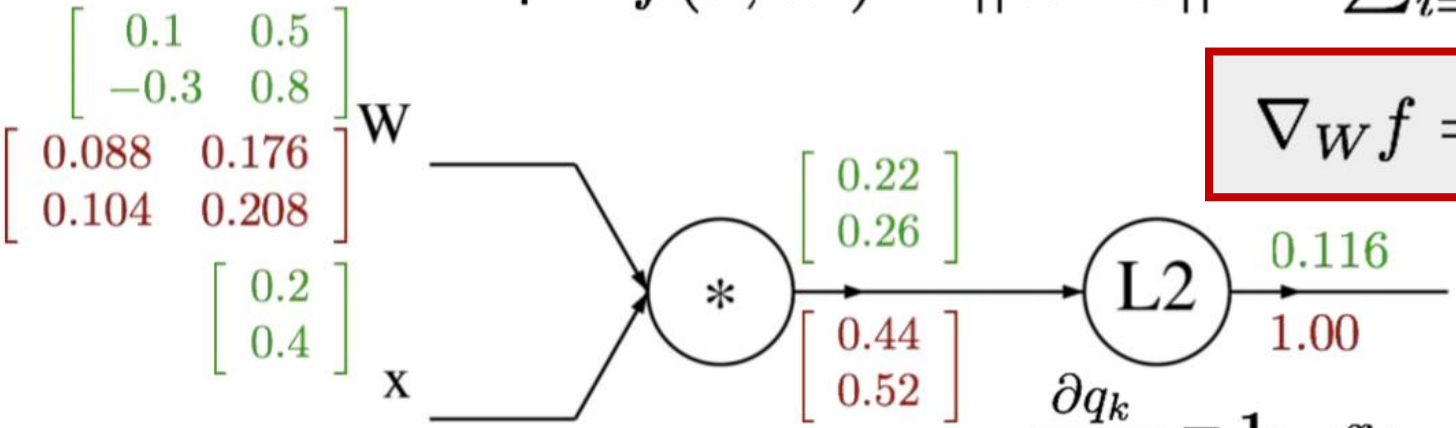
$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= \boxed{2q_i x_j}$$

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\boxed{\nabla_W f = 2q \cdot x^T}$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}} = \sum_k (2q_k)(\mathbf{1}_{k=i} x_j) = 2q_i x_j$$

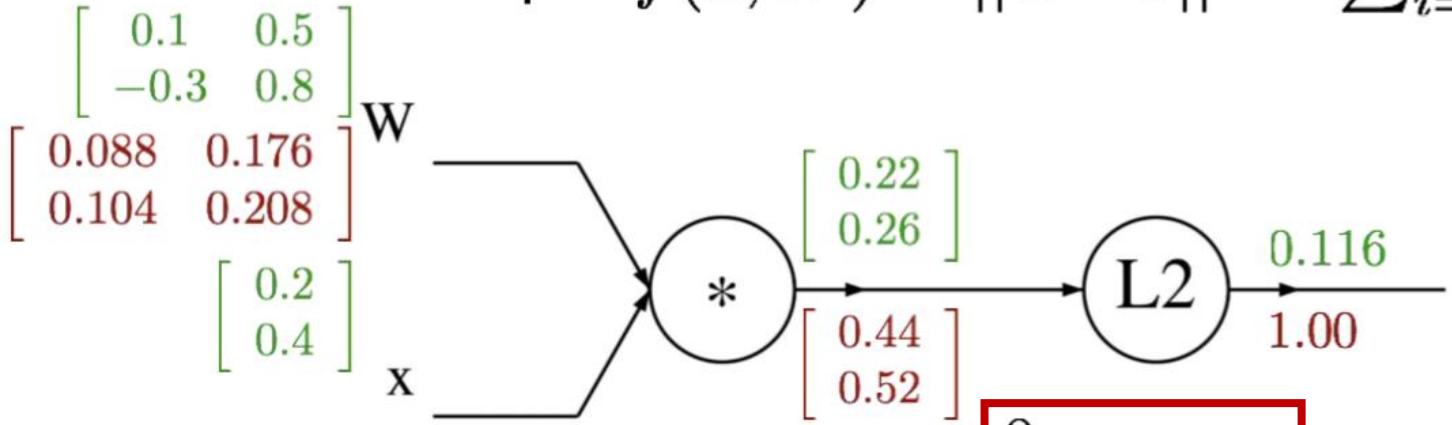
$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

변수에 대해 Gradient를 항상 체크하는 것이 중요→ 변수와 같은 shape!!

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$



$$\boxed{\frac{\partial q_k}{\partial x_i} = W_{k,i}}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

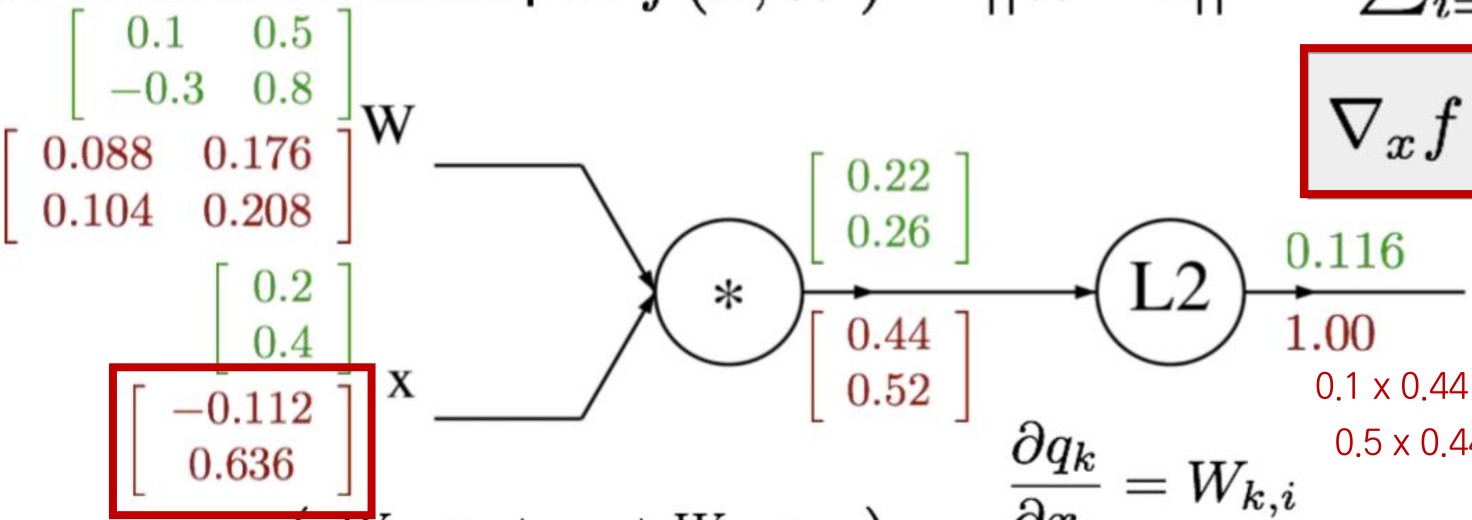
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

4. Vectorized example

A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\boxed{\nabla_x f = 2W^T \cdot q}$$



0.1 x 0.44 + -0.3 x 0.52 = -0.112

0.5 x 0.44 + 0.8 x 0.52 = 0.636

$$\frac{\partial q_k}{\partial x_i} = W_{k,i}$$

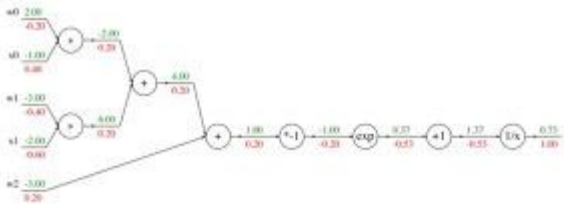
$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i} = \sum_k 2 q_k W_{k,i}$$

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

5. Modularized Implementation

Modularized implementation: forward / backward API

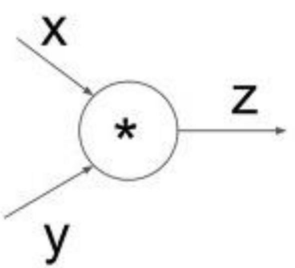


Graph (or Net) object *(rough psuedo code)*

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

5. Modularized Implementation

Modularized implementation: forward / backward API



(x,y,z are scalars)

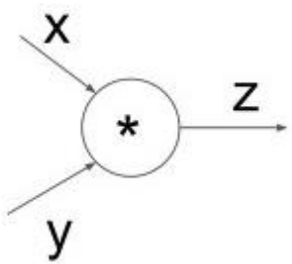
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        return z
    def backward(dz):
        # dx = ... #todo
        # dy = ... #todo
        return [dx, dy]
```

$$\frac{\partial L}{\partial z}$$

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

5. Modularized Implementation

Modularized implementation: forward / backward API



(x,y,z are scalars)

```python
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

5. Neural Network

인공 신경망

1. Neural Network

기존 Linear score function

$$f = Wx$$

현재 2-layer Neural Network

$$f = W_2 \boxed{\max(0, W_1 x)}$$

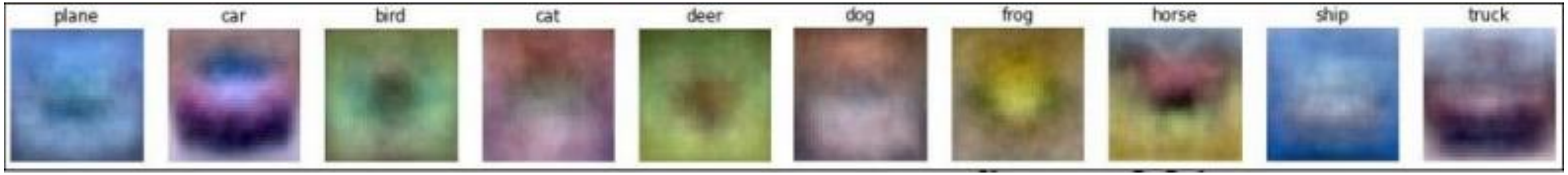
relu

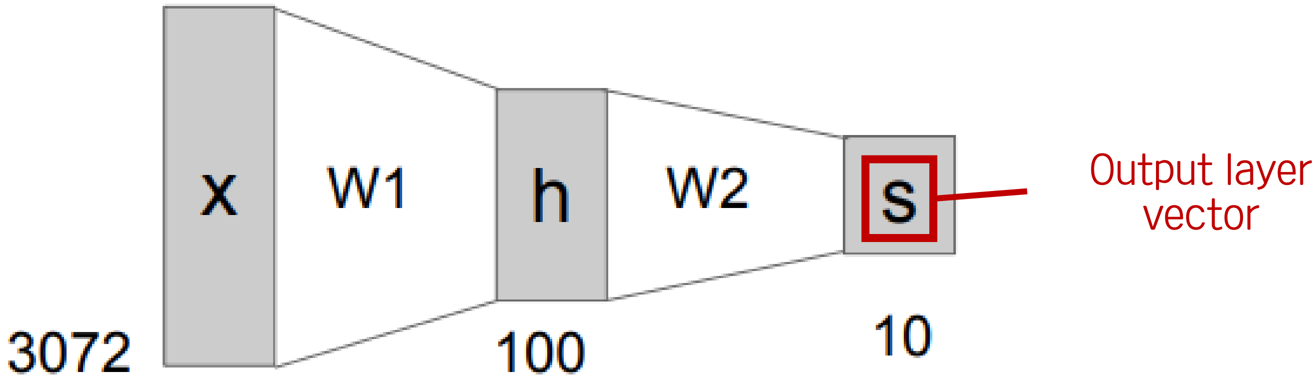
Q) 왜 max(0,W1x) 식을 쓰는가?

A) non-linearity (비선형식)이 쓰여야하기 때문

1. Neural Network

CIFAR-10 데이터





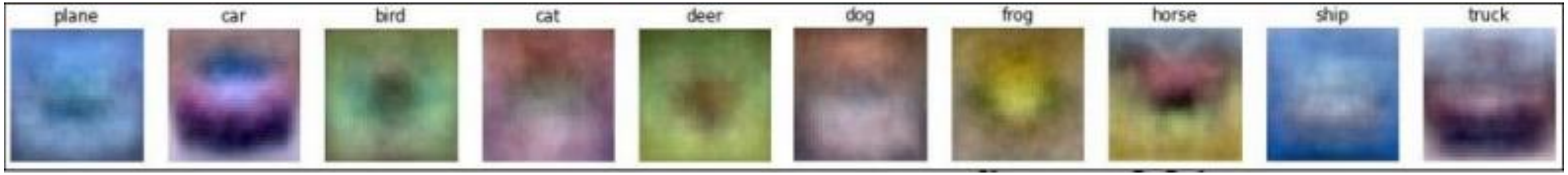
32 * 32 * 3
= 3072개

W1 * weight
= W1 * x

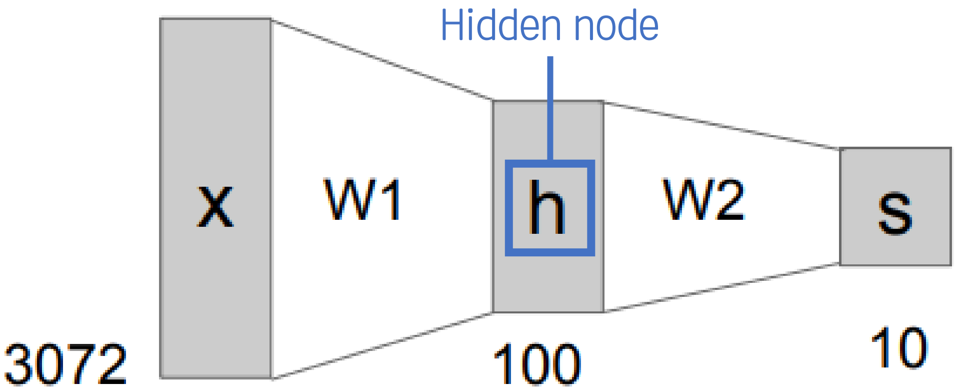
W2 * weight
= W1 * max(0,W1x)

1. Neural Network

KNN의 한계점을 "어떻게" 극복하였는가?



CIFAR-10 데이터



KNN : one-class, one-classifier

Neural network : hidden node 각각이 하나의 feature 담당

1. Neural Network

KNN

One-class　　One-classifier

자동차 = 빨간색 자동차

자동차 X

자동차

Neural network

Multi-classifier　　One-class

100 nodes

1) 빨간색 자동차 = 자동차

79) 노란색 자동차 = 자동차

자동차

자동차

Hidden layer의 개수는 hyper parameter로
성능이 가장 잘 나올 수 있도록 선정하는 것이 중요

1. Neural Network

Data-driven approach	
Non-parametric approach	Parametric approach
KNN	Neural network
One-class, one-classifier	One-class, multi-classifier

1. Neural Network

기존 Linear score function

$$f = Wx$$

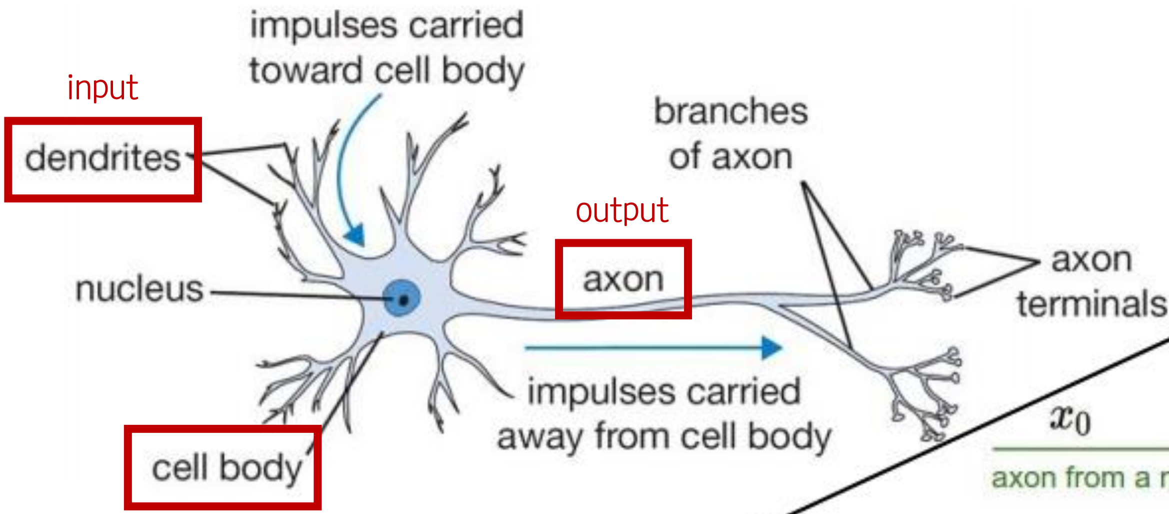
현재 2-layer Neural Network

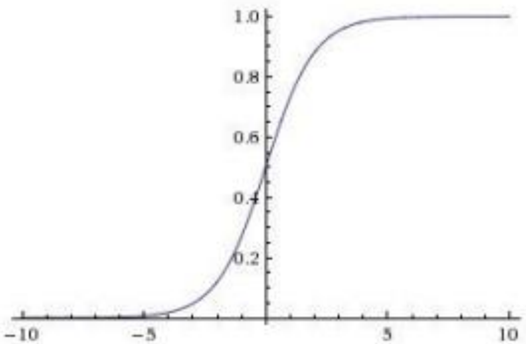
$$f = W_2 \max(0, W_1 x)$$

3-layer Neural Network

$$f = W_3 \boxed{\max(0, W_2 \max(0, W_1 x))}$$

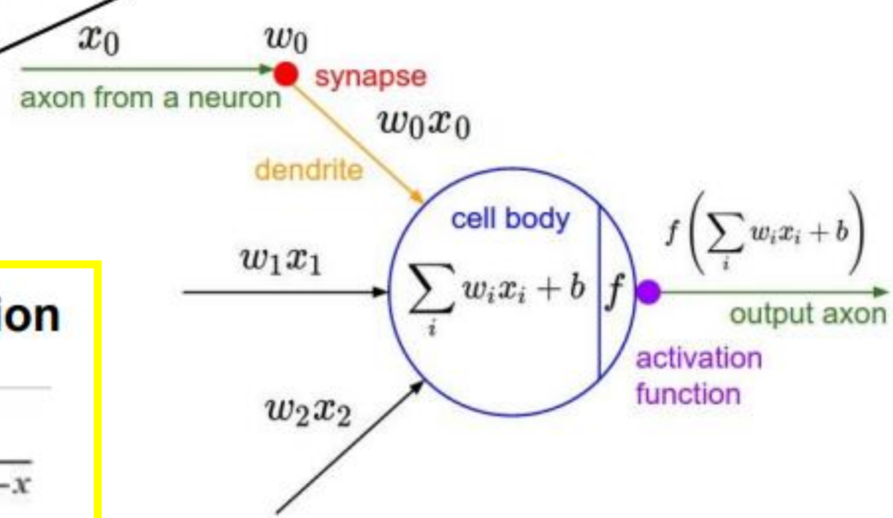
2. Neuron



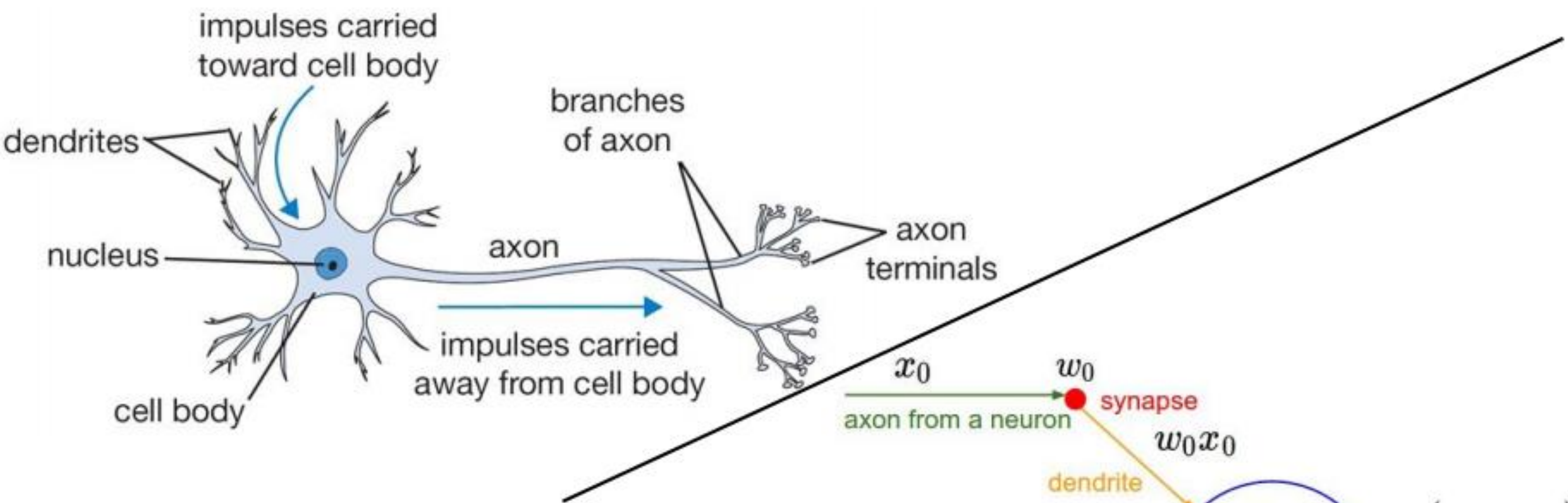


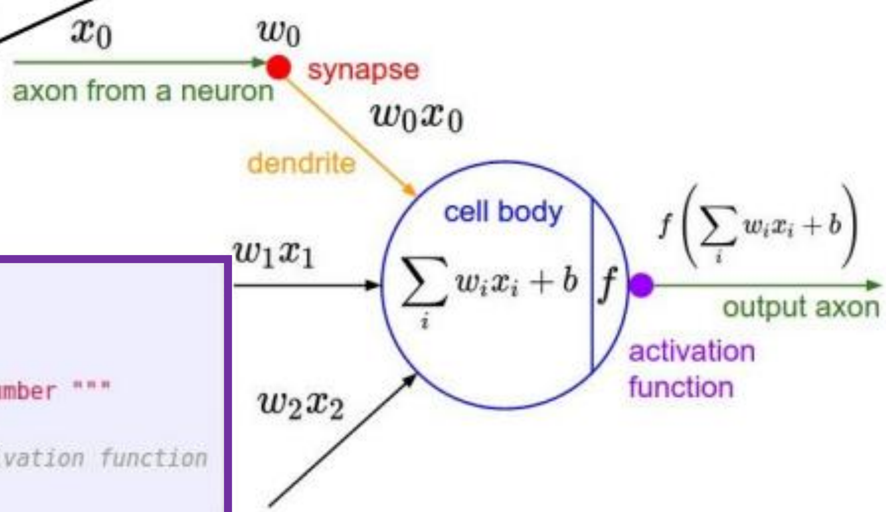
sigmoid activation function

$$\frac{1}{1+e^{-x}}$$



2. Neuron





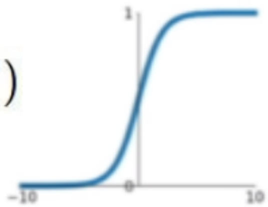
Sigmoid function code

```python
class Neuron:
  # ...
  def neuron_tick(inputs):
    """ assume inputs and weights are 1-D numpy arrays and bias is a number """
    cell_body_sum = np.sum(inputs * self.weights) + self.bias
    firing_rate = 1.0 / (1.0 + math.exp(-cell_body_sum)) # sigmoid activation function
    return firing_rate
```

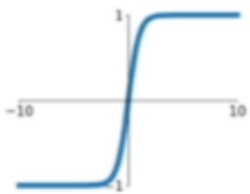
3. Activation Functions

		단점	보완
Sigmoid	Logistic 함수, x의 값에 따라 0~1의 값을 출력하는 S자형 함수	·Vanishing Gradient Problem ·느린 학습	
Tanh	Sigmoid 함수를 변형해서 얻은 쌍곡선 함수	·Vanishing Gradient Problem	·느려지는 문제 해결
ReLU	Sigmoid와 Tanh가 갖는 Gradient Vanishing Problem을 해결한 경사함수	·Dying ReLU	·Vanishing Gradient Problem 문제 해결 ·빠른 학습, 적은 연산 비용, 간단한 구현

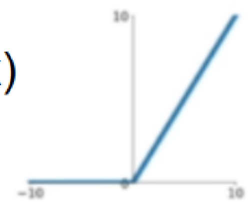
$$\sigma(x) = 1/(1+e^{-x})$$



tanh(x)



max(0,x)

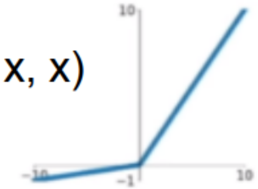


3. Activation Functions

Leaky ReLU

Dying ReLU을 해결한 함수

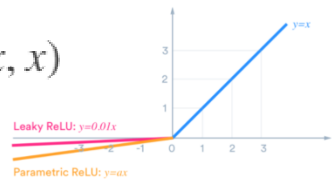
max(0.1x, x)



PReLU

x가 음수인 영역에서 기울기를 학습한 함수

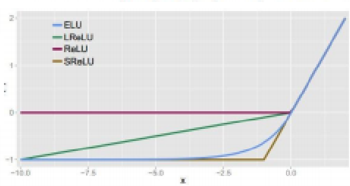
$$f(x) = max(\alpha x, x)$$



ELU

Exponential Linear Unit, ReLU의 모든 장점을 포함하며
Dying ReLU 문제를 해결한 함수

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$$



Maxout

ReLU의 장점을 모두 갖고 Dying ReLU을
해결한 함수

$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

단점

·x가 음수인 영역의 값에 대해
정의하지 못함

·exp 함수를 계산하는 비용

·복잡하고 많은 양의 계산

보완

·Dying ReLU 해결

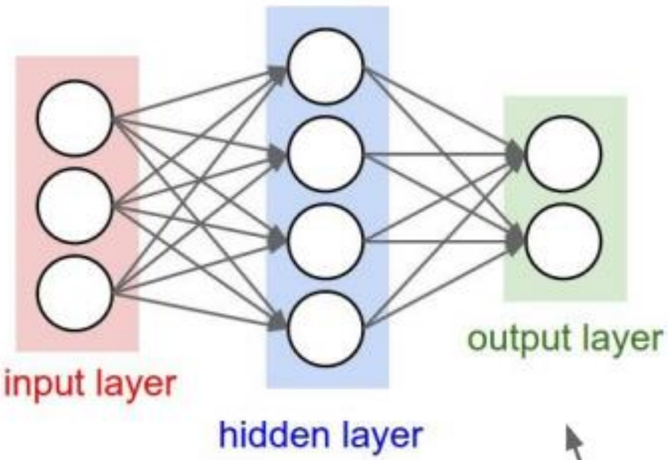
·새로운 파라미터 α 를 추가하여
x가 음수인 영역에서도 기울기를
학습

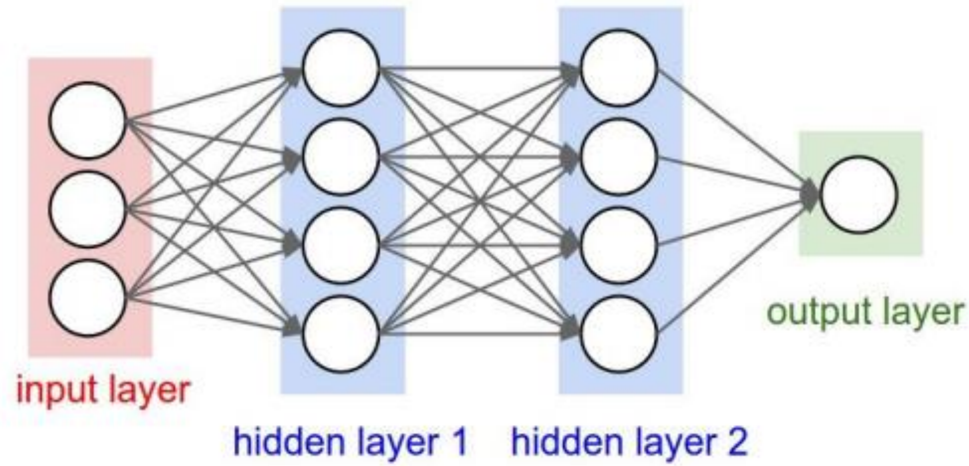
·Dying ReLU 해결

·Dying ReLU 해결

4. Neural Network : Architectures

Layer의 구분 기준은 Weight를 가지는 것





"2-layer Neural Net", or
"1-hidden-layer Neural Net"

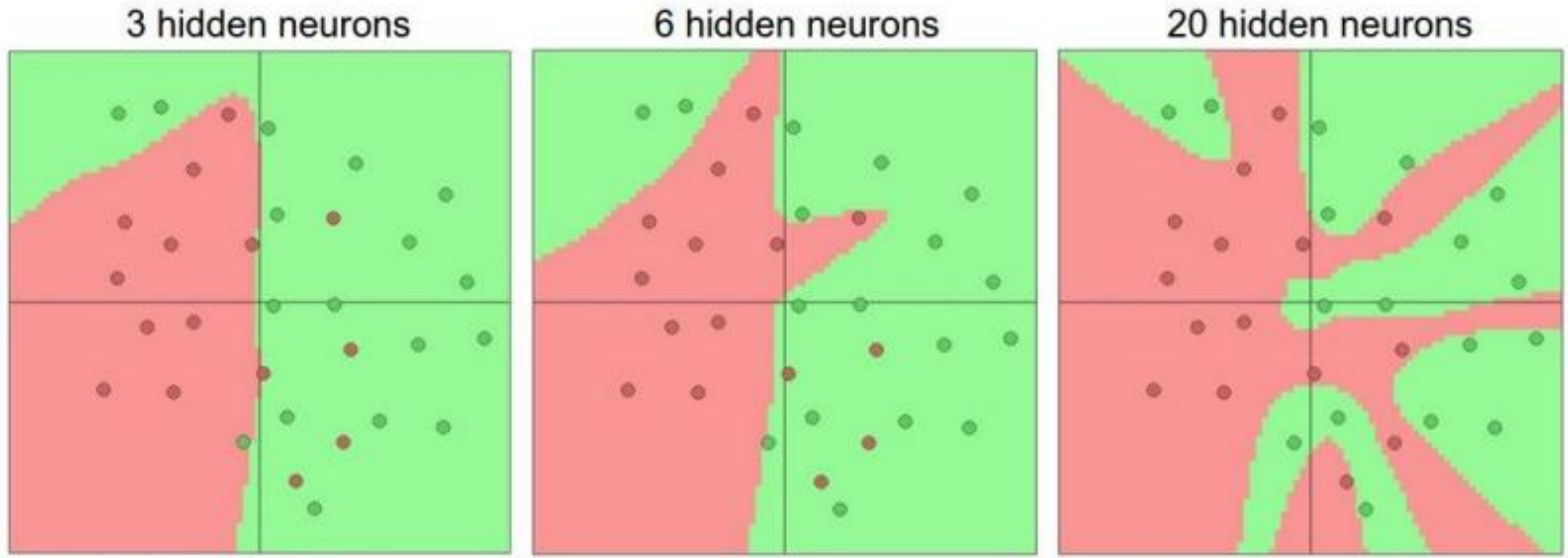
"3-layer Neural Net", or
"2-hidden-layer Neural Net"

"Fully-connected" layers

Q) Fully-connected layer로 구성하는 이유?
A) Layer 간의 연산을 간단하게 한 줄로 표현할 수 있기 때문

5. Layers Setting

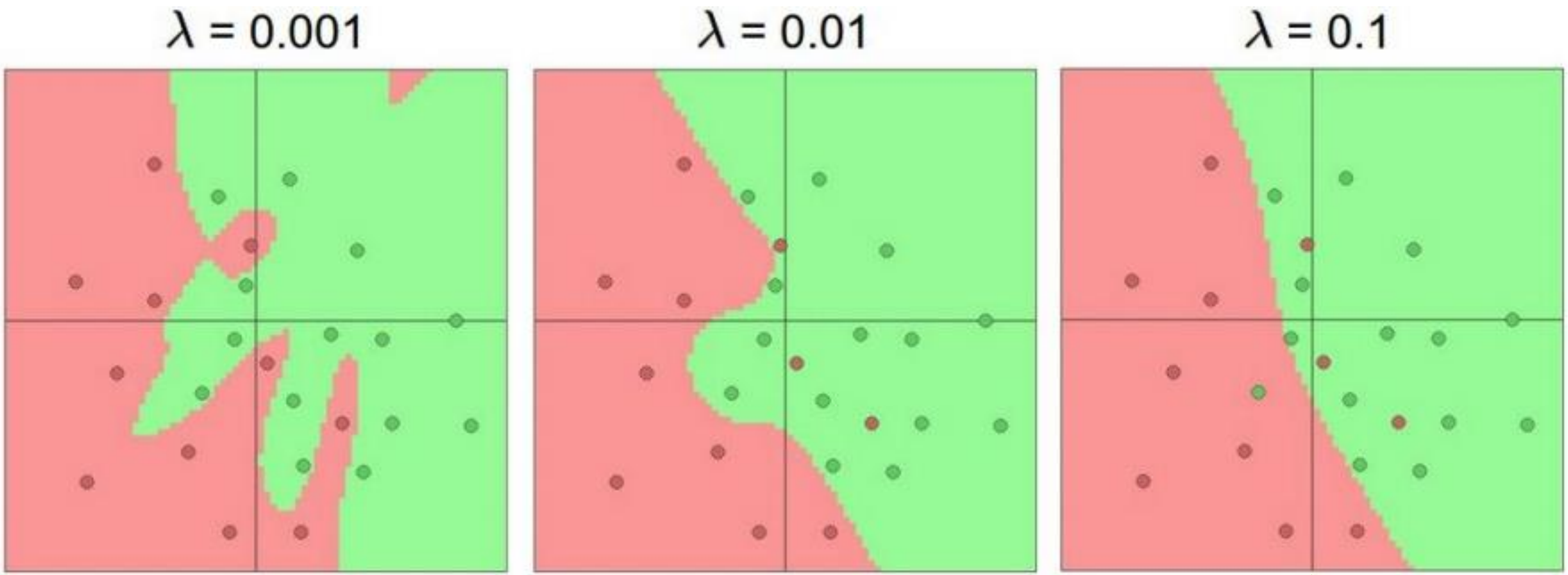
Layer의 개수를 설정하는 example



More neurons = More capacity

5. Layers Setting

Neural Net의 개수 ≠ regularizer의 역할



Training data에 overfitting 되지 않고
Test data에 일반화 되고 있음

데이터의 overfitting이 일어나지 않도록 network를
잘 구성하는 방법은 regularizer strength를 더 높여주어야 한다.

Neural Network

Bigger = Better

layer가 깊으면 깊을수록 좋음
(regularization을 잘했다는 전제
하에)

감사합니다

Q&A