Lecture 4:
Backpropagation and Neural Networks

목차

- Backpropagation
- Neural Networks

Backpropagation

Gradient descent (경사 하강법)

- Numerical gradient (수치 gradient)
  - Finite difference approximation

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

  - $\frac{1-e^x}{1+e^{-x}}$ 의 식에서 $x = 1$일 때의 기울기를 알고 싶을 때, 위의 식에서 x에 1을 대입하고, h 에 0.01을 대입하여 기울기를 구할 수 있다.
- Analytic gradient (해석 gradient)
  - 분석적으로 미적분 사용
  - $x^2 - 2x - 3$ 의 식이 있을 때, $(x-3)(x+1)$ 의 식으로 변형시켜 x의 값이 3 또는 -1 인 것을 알 수 있다.

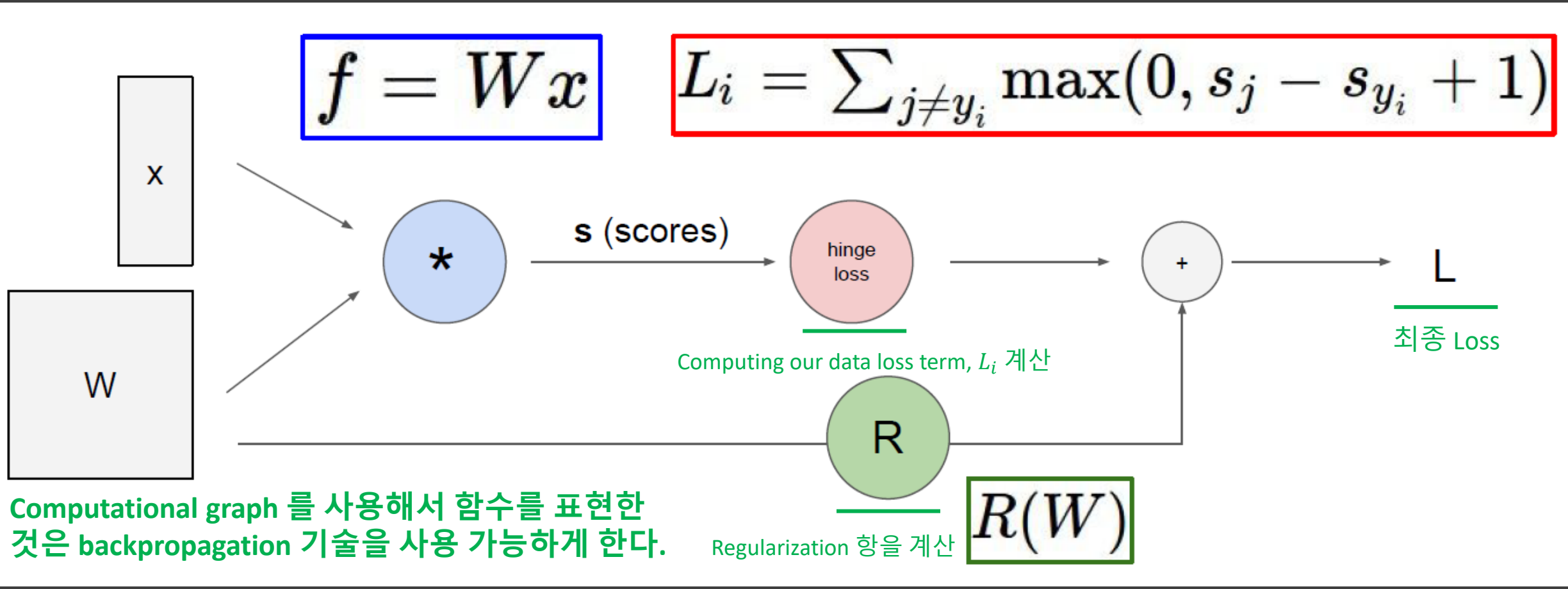
$$f(x) = 3x$$

$$\frac{df}{dx} = \lim_{h \to 0} \frac{3(x+h) - 3x}{h} = \lim_{h \to 0} \frac{3x + 3h - 3x}{h} = \lim_{h \to 0} \frac{3h}{h} = 3$$

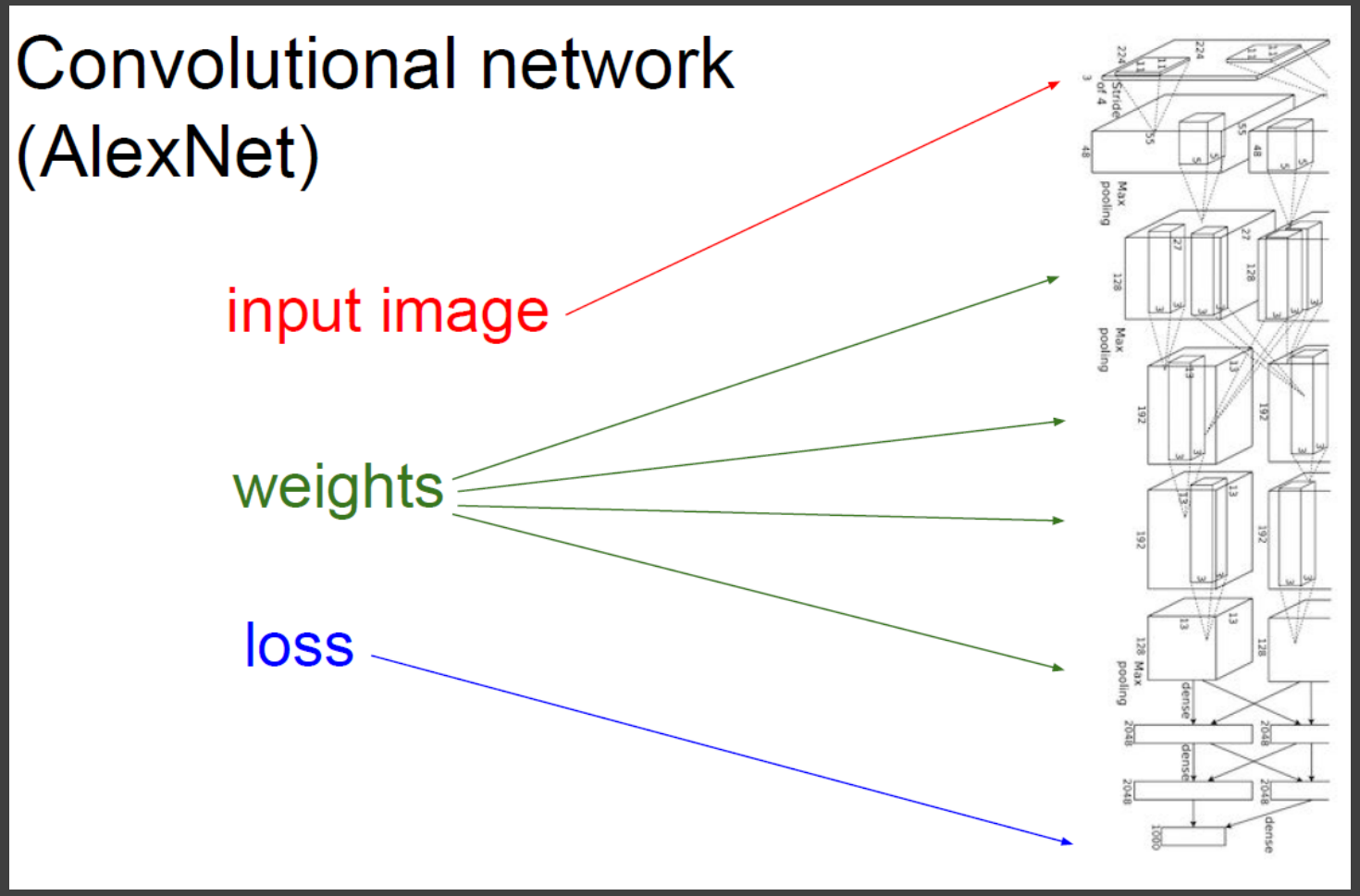
Numerical gradient	Analytic gradient
Slow(계산량多)	Fast
Approximate(근사)	Exact
Easy to write	Error-prone

- 실제로는 analytic gradient 를 계산한 다음 이를 numerical gradient 와 비교하여 구현의 정확성을 확인 ➔ Gradient check

Computational graphs



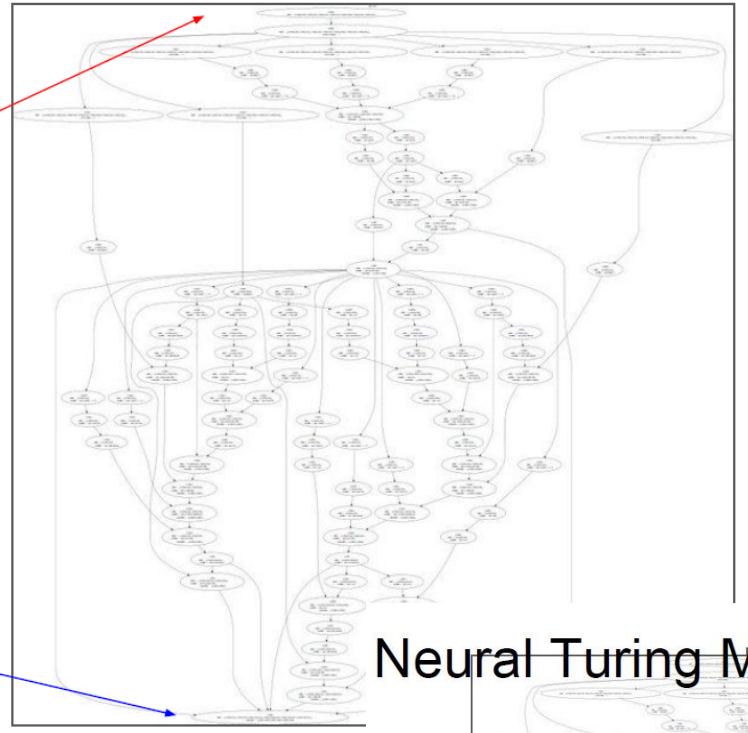
Backpropagation 은 complex functions 를 사용할 때 매우 유용하다.



Neural Turing Machine

input image

loss



Neural Turing Machine

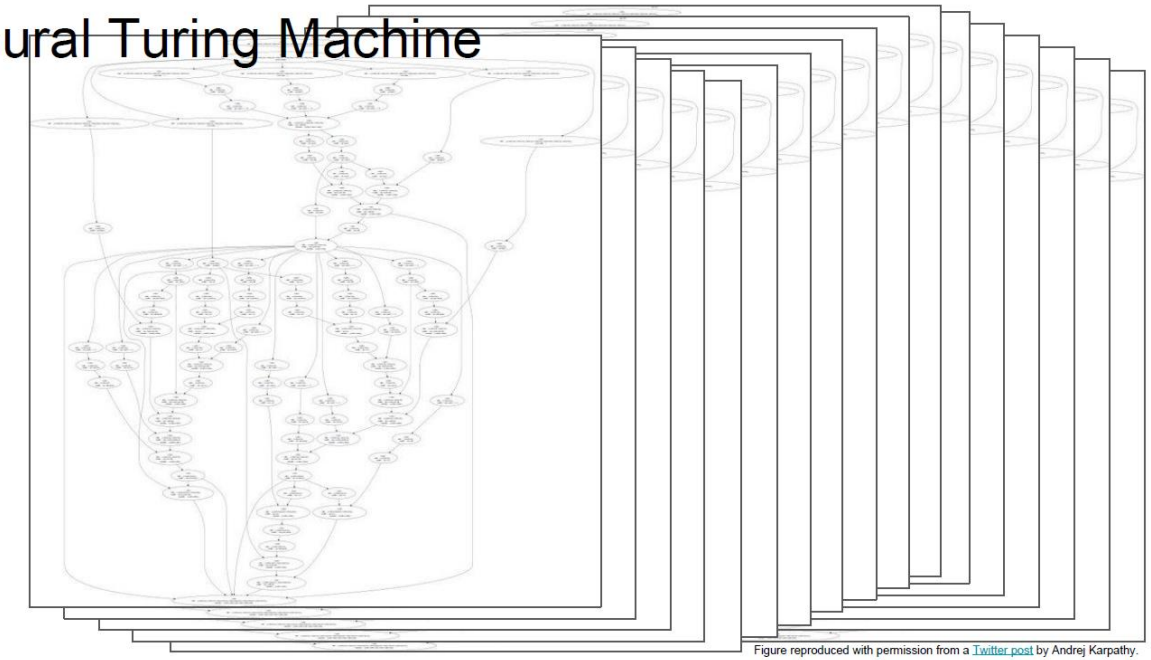


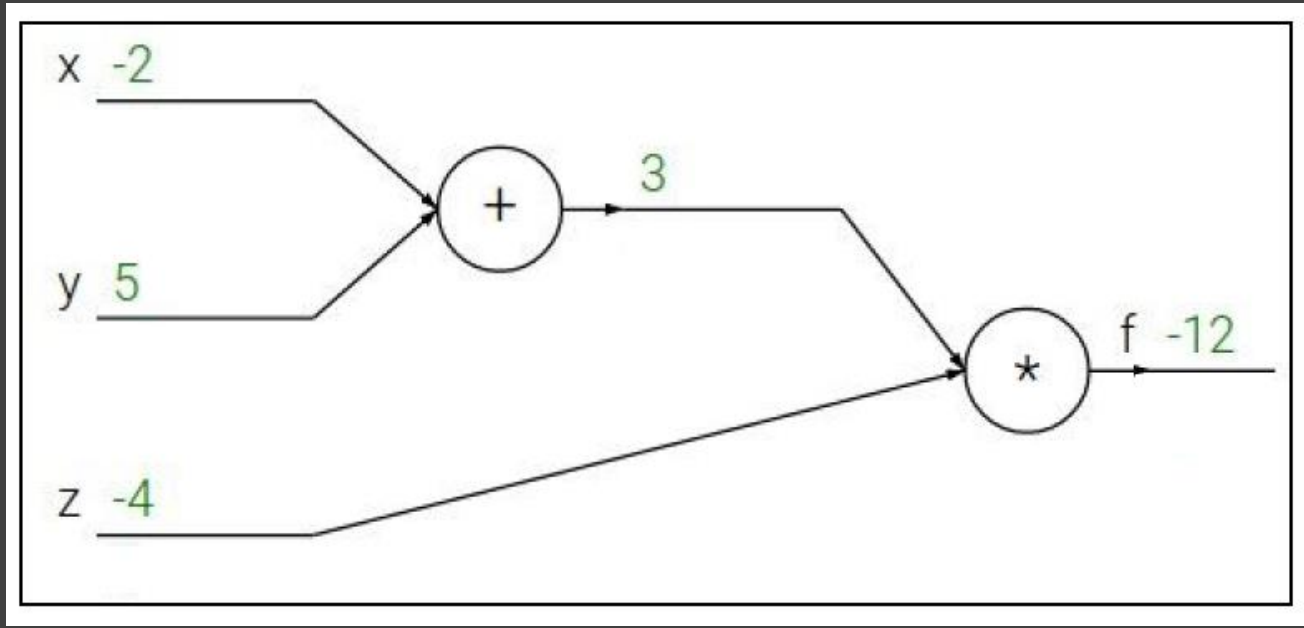
Figure reproduced with permission from a Twitter post by Andrej Karpathy.

Backpropagation: a single example

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

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

- Function f 의 출력에 대한 어떤 변수(x, y, z)의 gradient 를 찾기 원한다.
  - 가장 먼저 해야 할 것은 함수 f 를 이용해서 computational graph 로 나타내는 것!!!



Function f

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

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

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

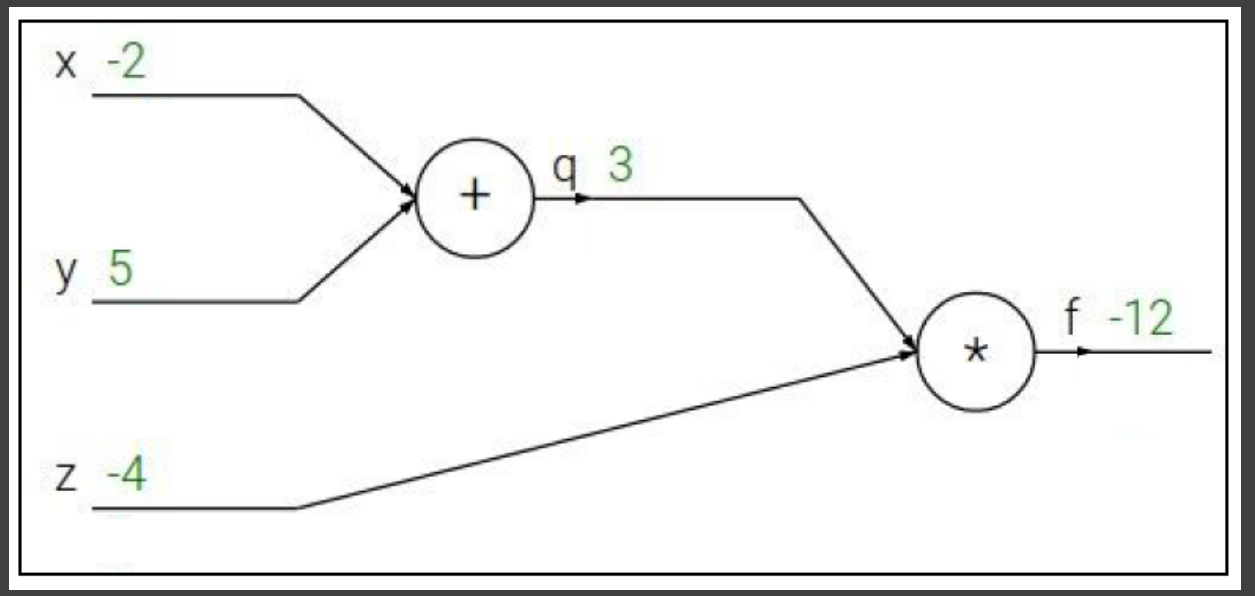
x 와 y 에 대한 q 의 gradient

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

q 와 z 에 대한 f 의 gradient

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

Computational graph



Backpropagation 은 chain rule 의 재귀적인 응용이다.
Chain rule 에 의해 computational graph 의 가장 끝에서부터 gradient 를 계산한다.

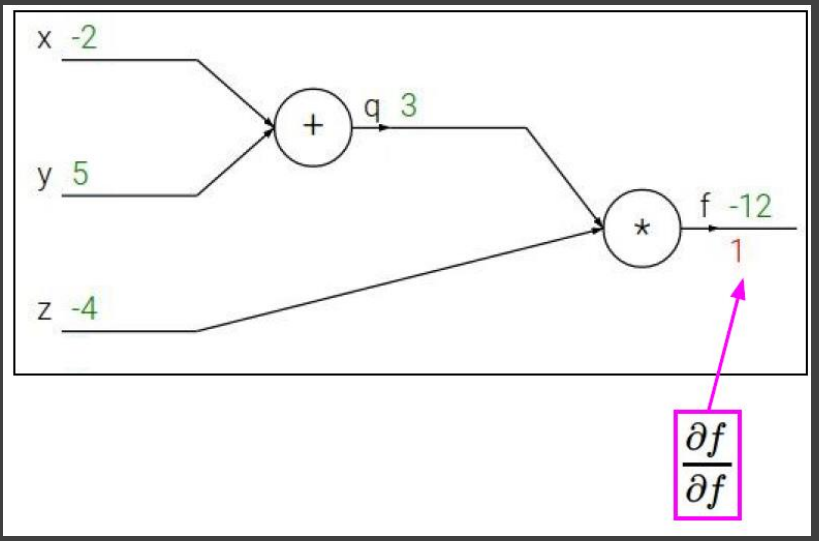
$f(x, y, z) = (x + y)z$

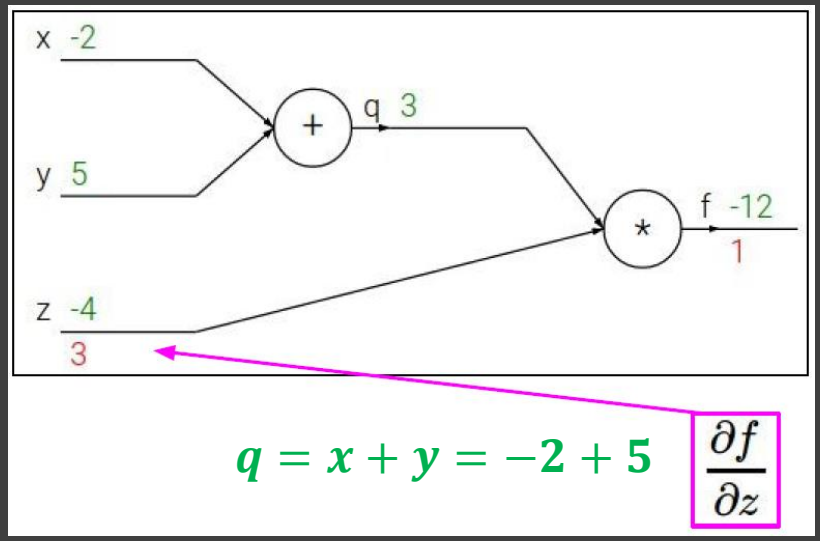
e.g. x = -2, y = 5, z = -4

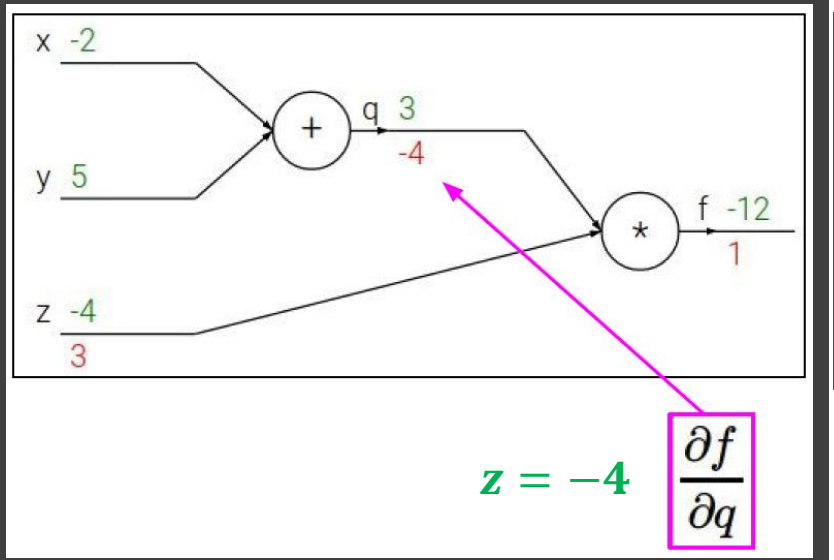
$q = x + y \quad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$

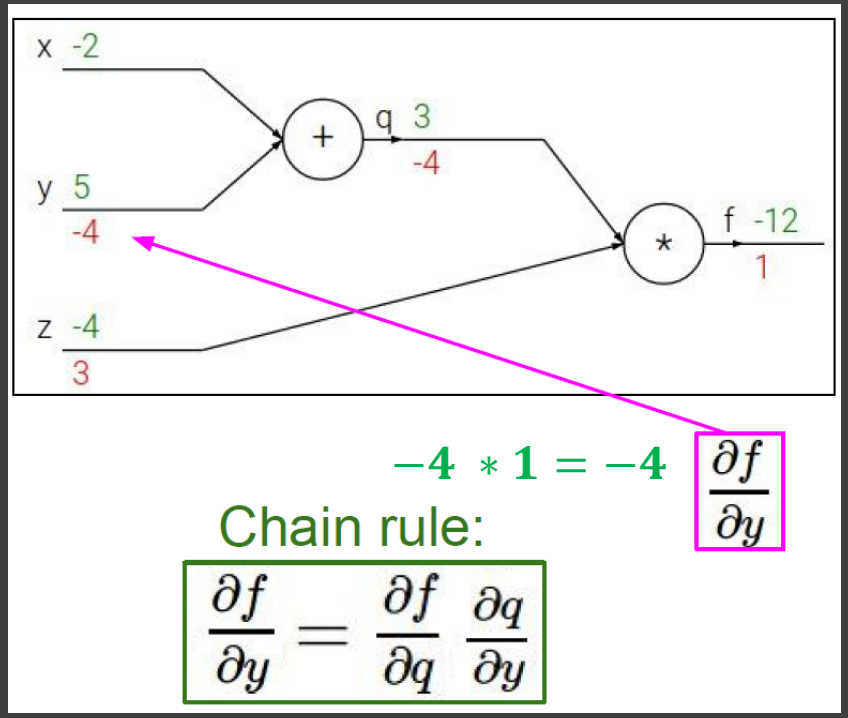
$f = qz \quad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$

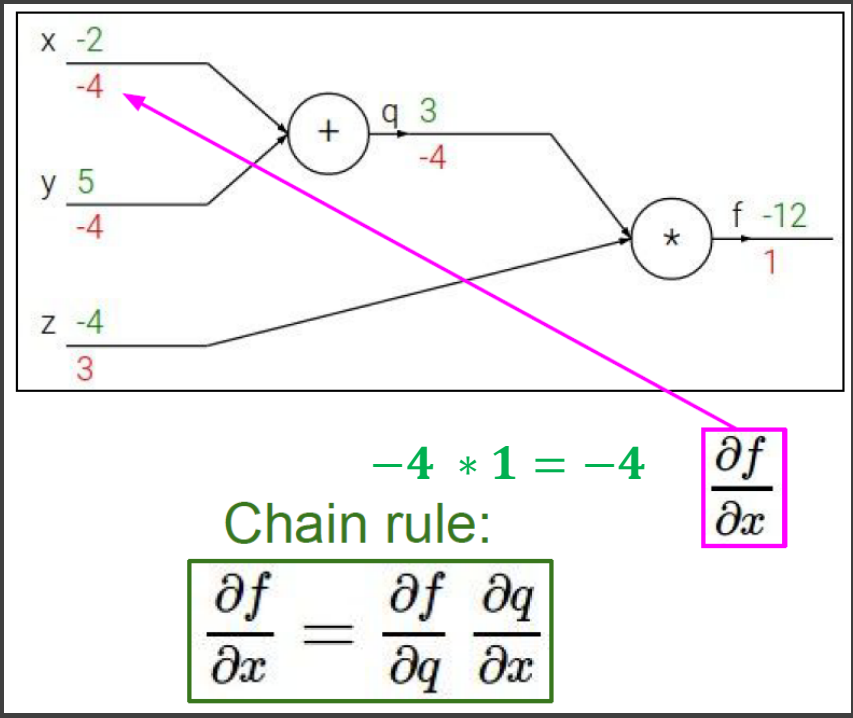
초록색 숫자 : values
빨간색 숫자 : gradients

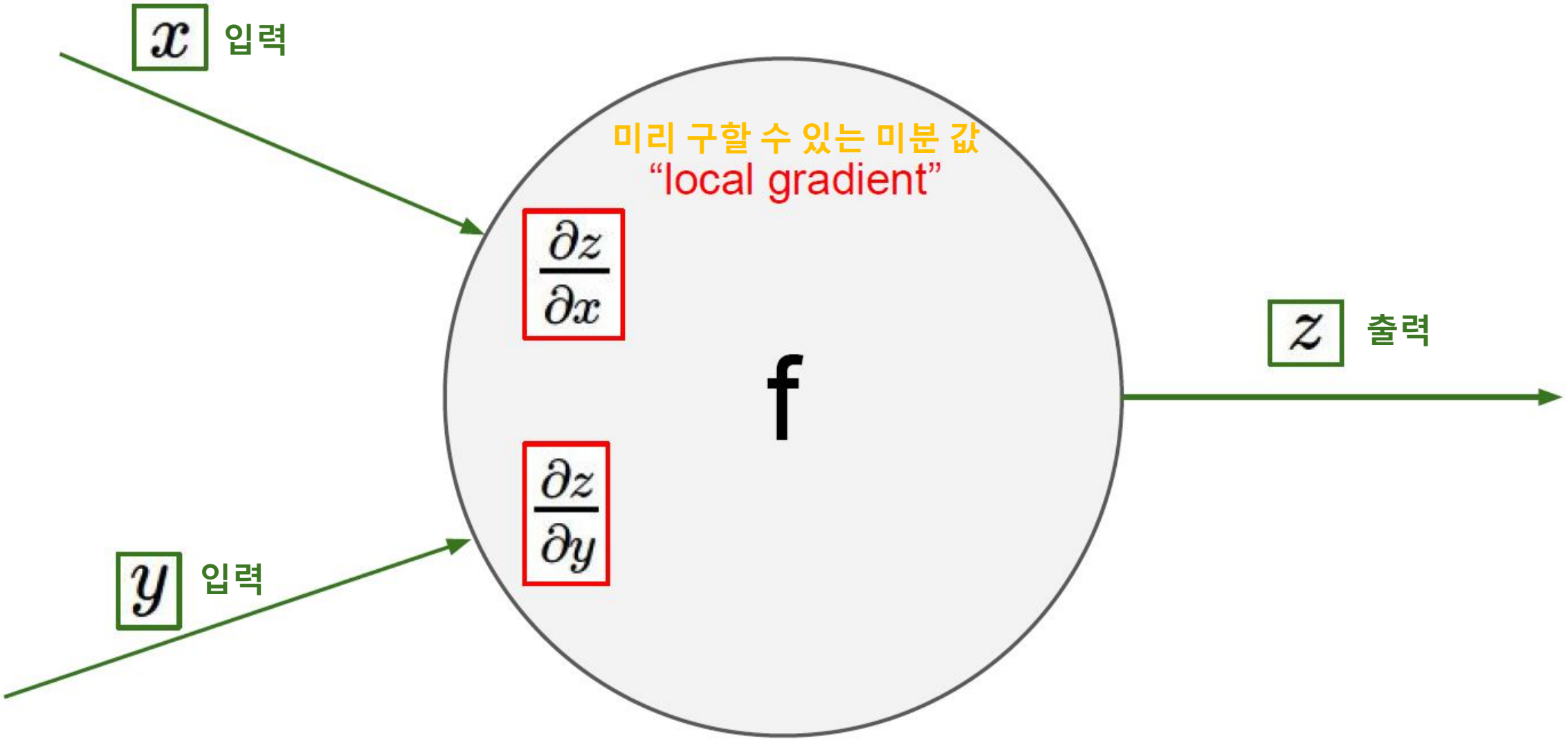


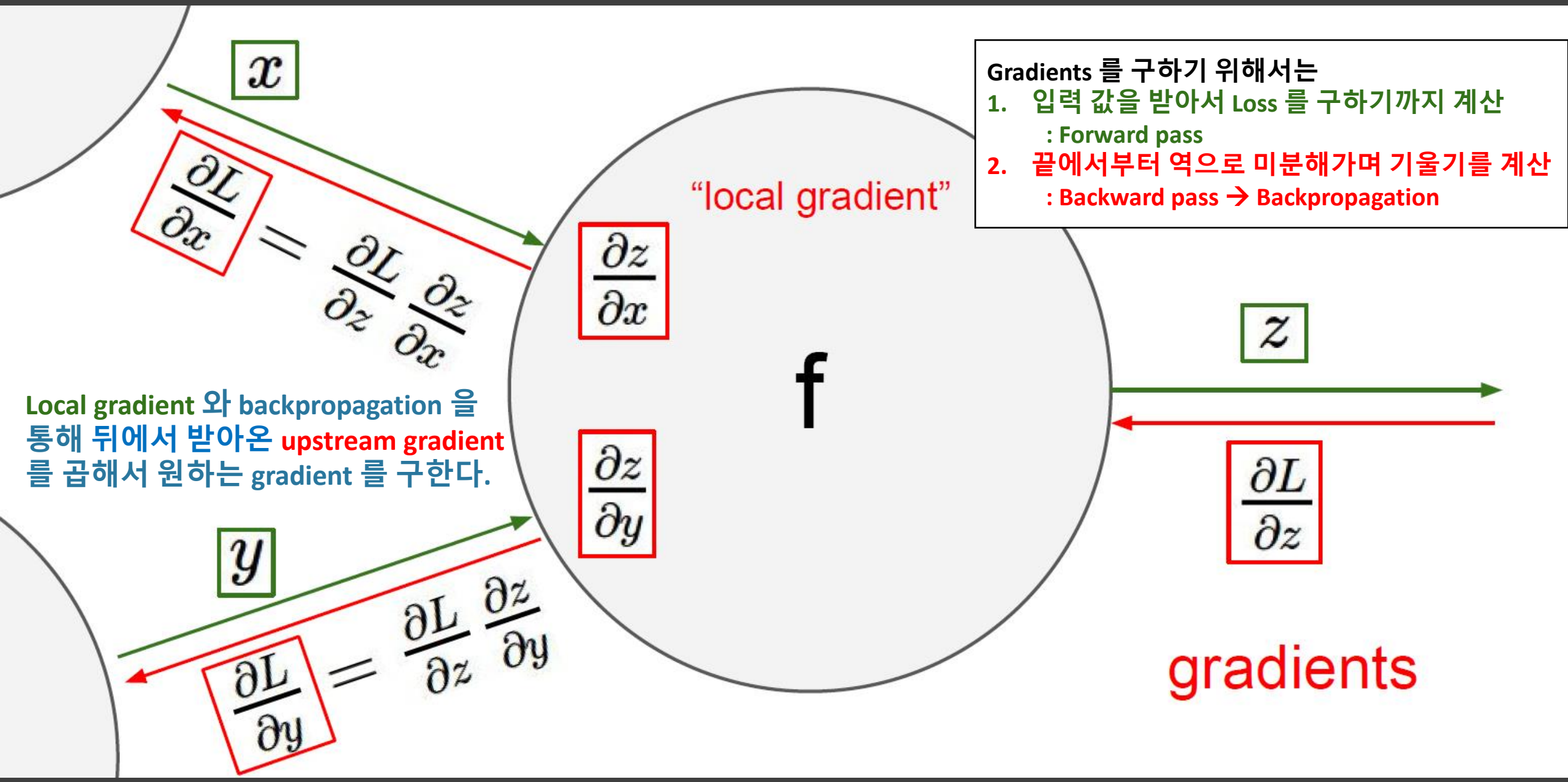









x 입력
미리 구할 수 있는 미분 값
"local gradient"
$\frac{\partial z}{\partial x}$
f
$\frac{\partial z}{\partial y}$
z 출력
y 입력



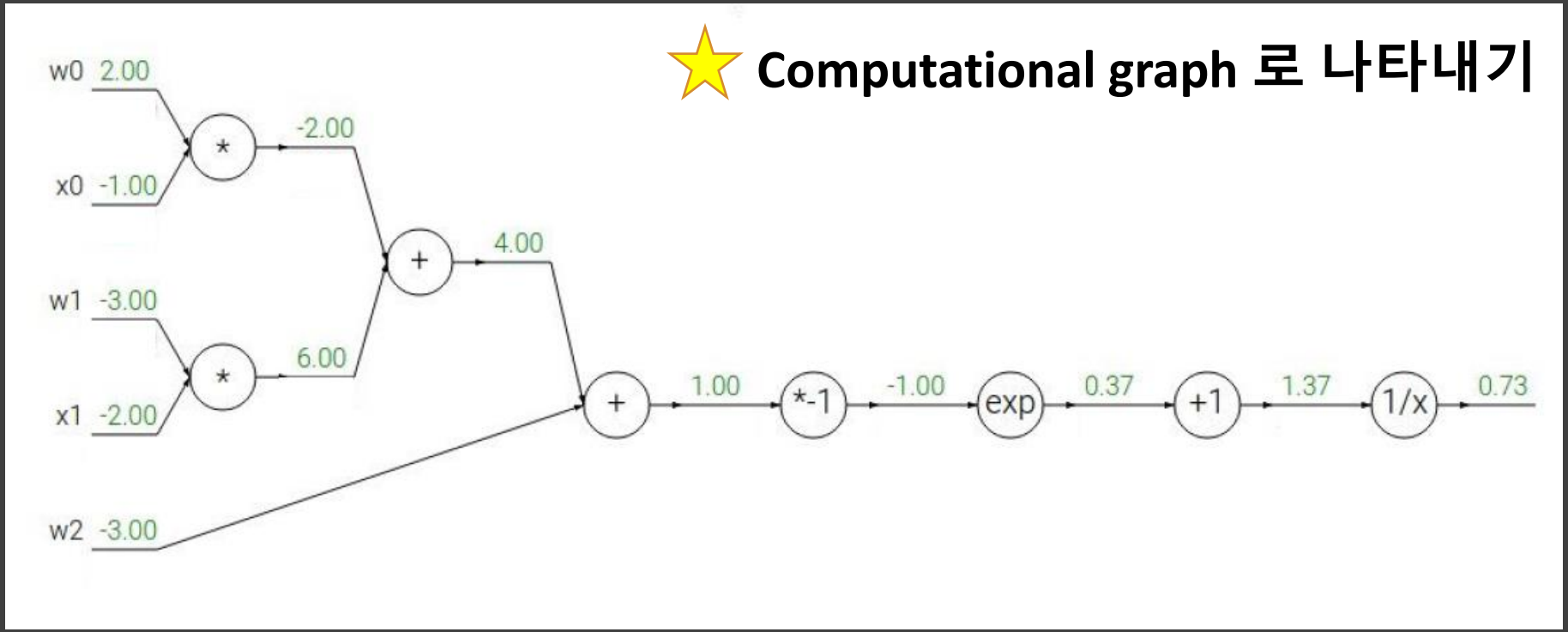
Gradients 를 구하기 위해서는

1. 입력 값을 받아서 Loss 를 구하기까지 계산
   : Forward pass
2. 끝에서부터 역으로 미분해가며 기울기를 계산
   : Backward pass → Backpropagation

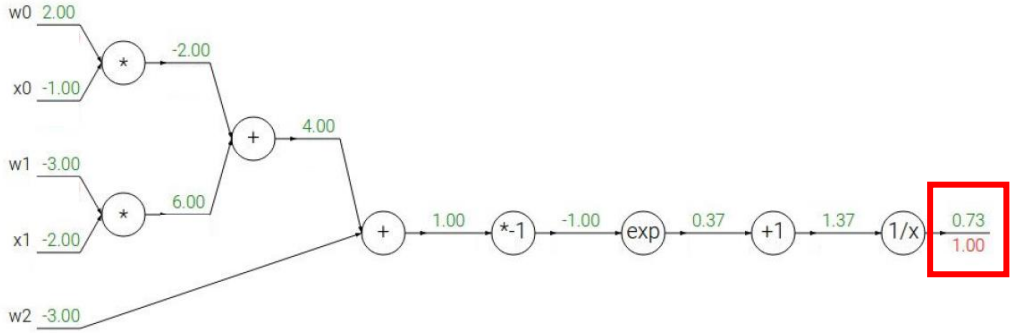
Another example :

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

복잡해서 backpropagation 이 얼마나 유용한 방식인지 알 수 있다.



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



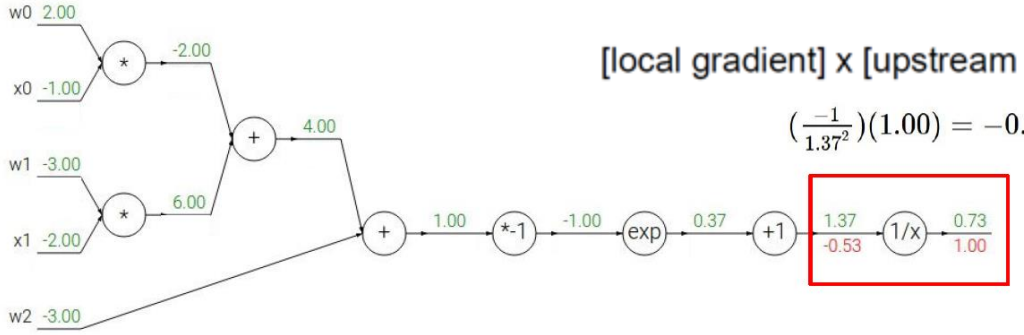
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

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



[local gradient] x [upstream gradient]

$(\frac{-1}{1.37^2})(1.00) = -0.53$

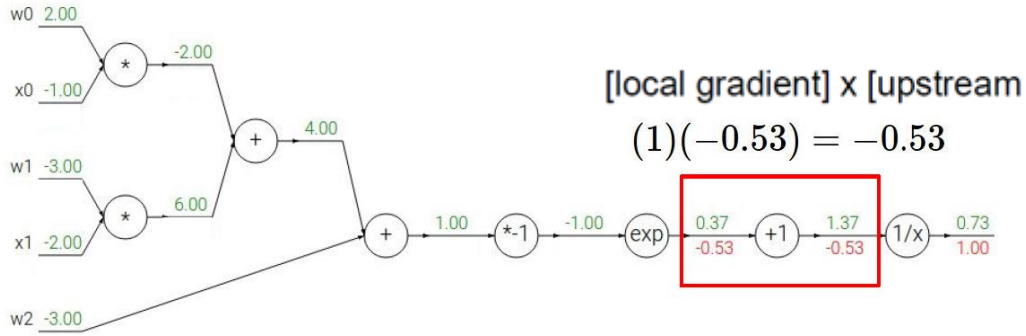
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

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



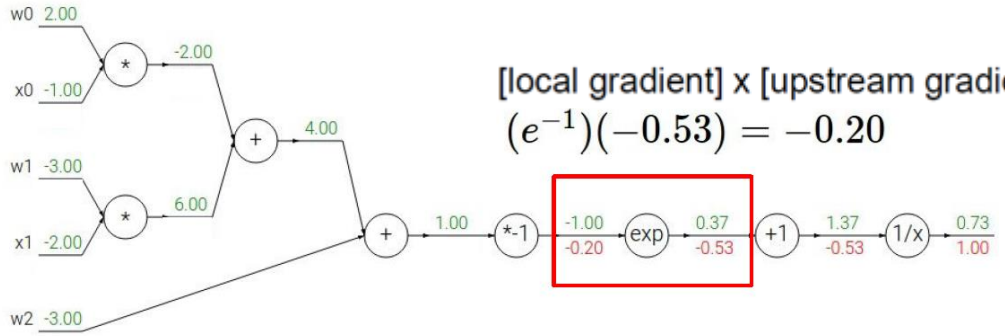
[local gradient] x [upstream gradient]

$(1)(-0.53) = -0.53$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$

$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

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



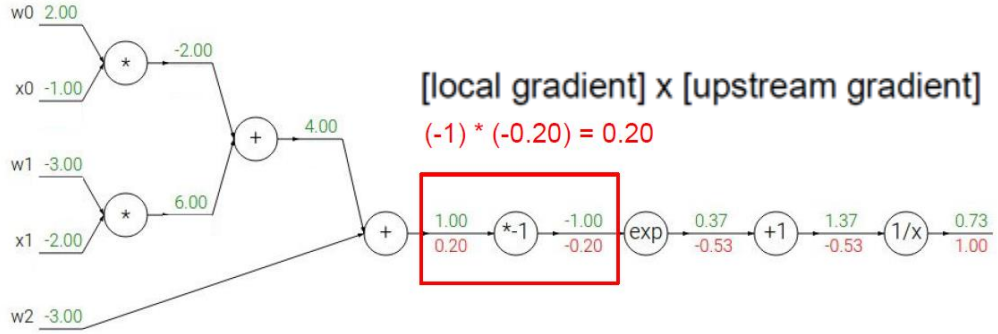
[local gradient] x [upstream gradient]

$(e^{-1})(-0.53) = -0.20$

$f(x) = e^x$	$\rightarrow$	$\frac{df}{dx} = e^x$
$f_a(x) = ax$	$\rightarrow$	$\frac{df}{dx} = a$

$f(x) = \frac{1}{x}$	$\rightarrow$	$\frac{df}{dx} = -1/x^2$
$f_c(x) = c + x$	$\rightarrow$	$\frac{df}{dx} = 1$

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



[local gradient] x [upstream gradient]

(-1) * (-0.20) = 0.20

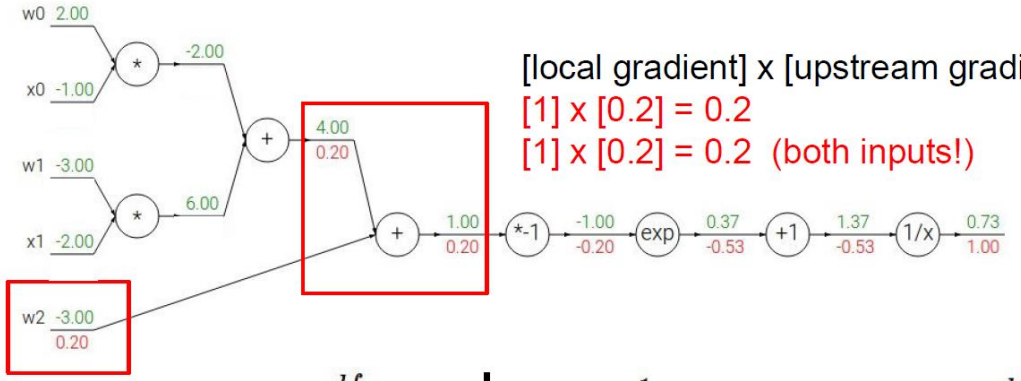
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

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



[local gradient] x [upstream gradient]

[1] x [0.2] = 0.2

[1] x [0.2] = 0.2 (both inputs!)

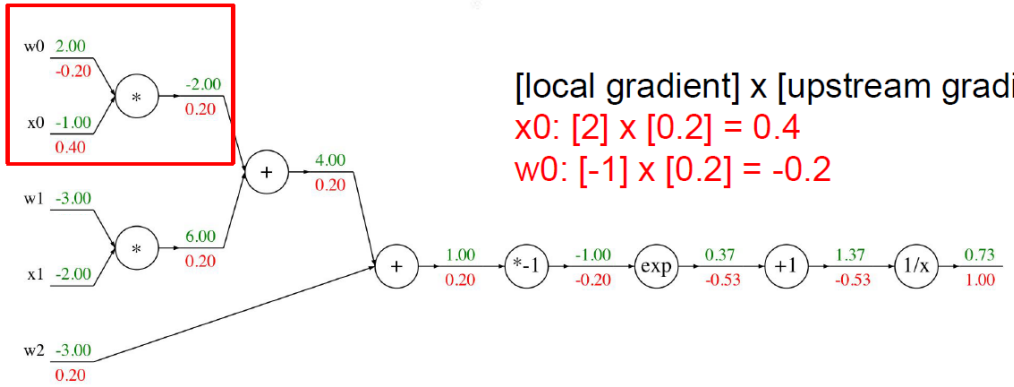
$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$

$f_a(x) = ax \rightarrow \frac{df}{dx} = a$

$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$

$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$

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



[local gradient] x [upstream gradient]
x0: [2] x [0.2] = 0.4
w0: [-1] x [0.2] = -0.2

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

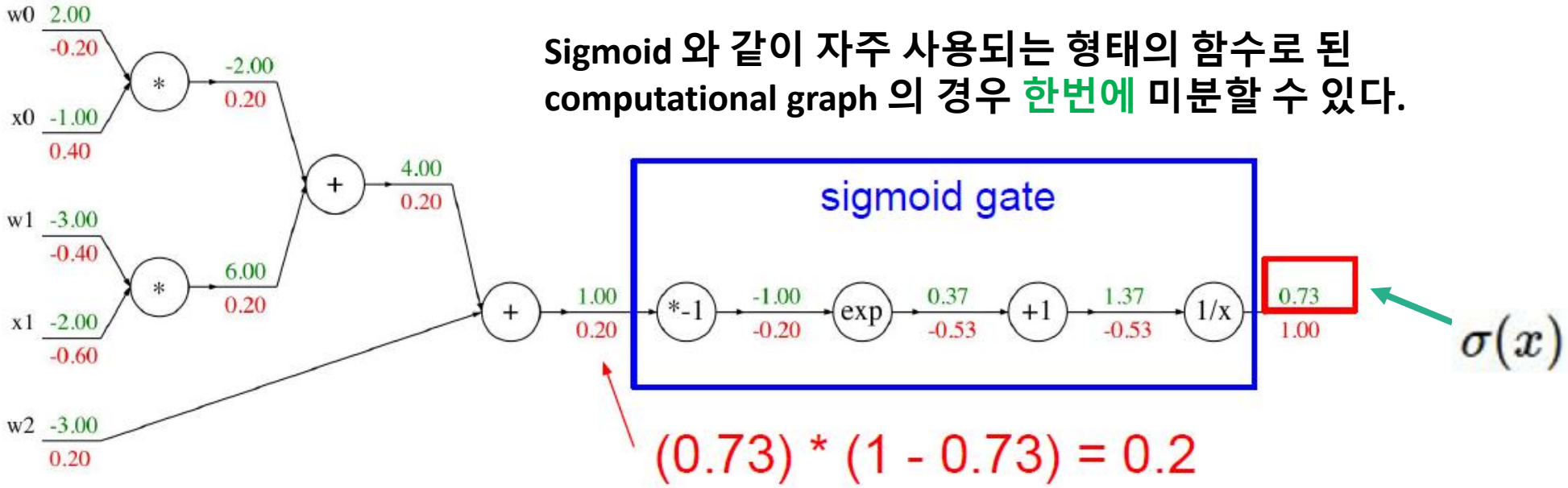
$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

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

$$\sigma(x) = \frac{1}{1+e^{-x}}$$ sigmoid function

$$\frac{d\sigma(x)}{dx} = \frac{e^{-x}}{(1+e^{-x})^2} = \left(\frac{1+e^{-x}-1}{1+e^{-x}}\right)\left(\frac{1}{1+e^{-x}}\right) = (1-\sigma(x))\,\sigma(x)$$

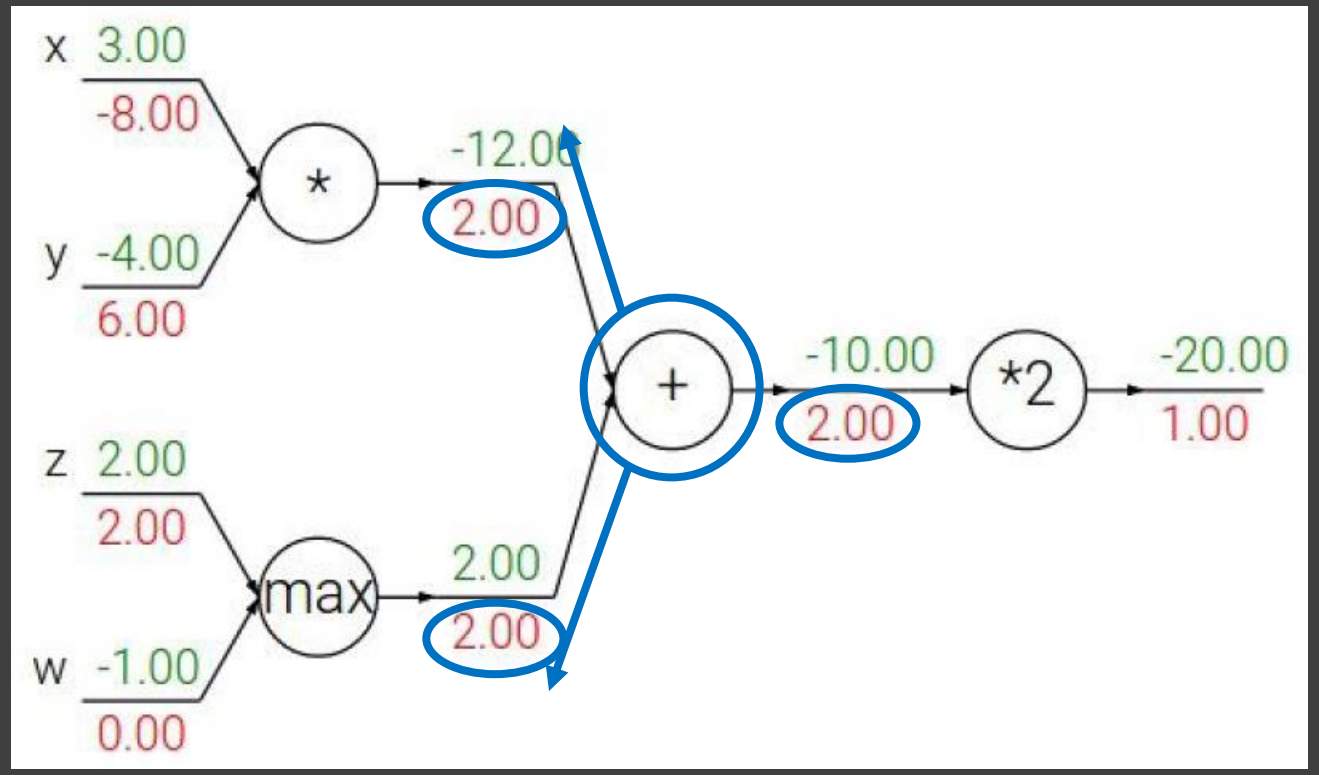


Sigmoid 와 같이 자주 사용되는 형태의 함수로 된 computational graph 의 경우 한번에 미분할 수 있다.

simple graph & complex computation of gradients vs simple computation of gradients & complex graph

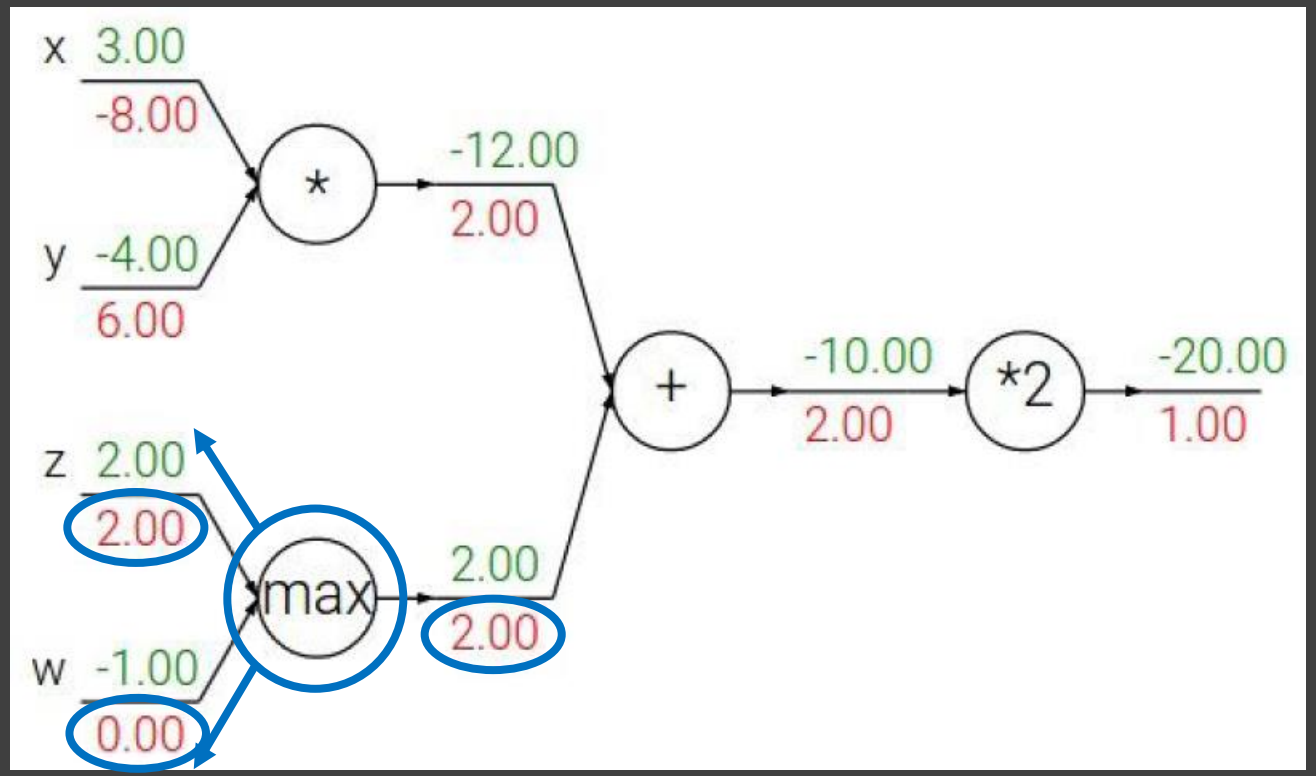
Patterns in backward flow

- add gate : gradient distributor



add gate 의 local gradient 는 1
→ local gradient * upstream gradient 를 하면 upstream gradient 가 그대로 나오게 된다.
→ add gate 는 gradient 를 그대로 전해주는 역할을 한다
→ distributor

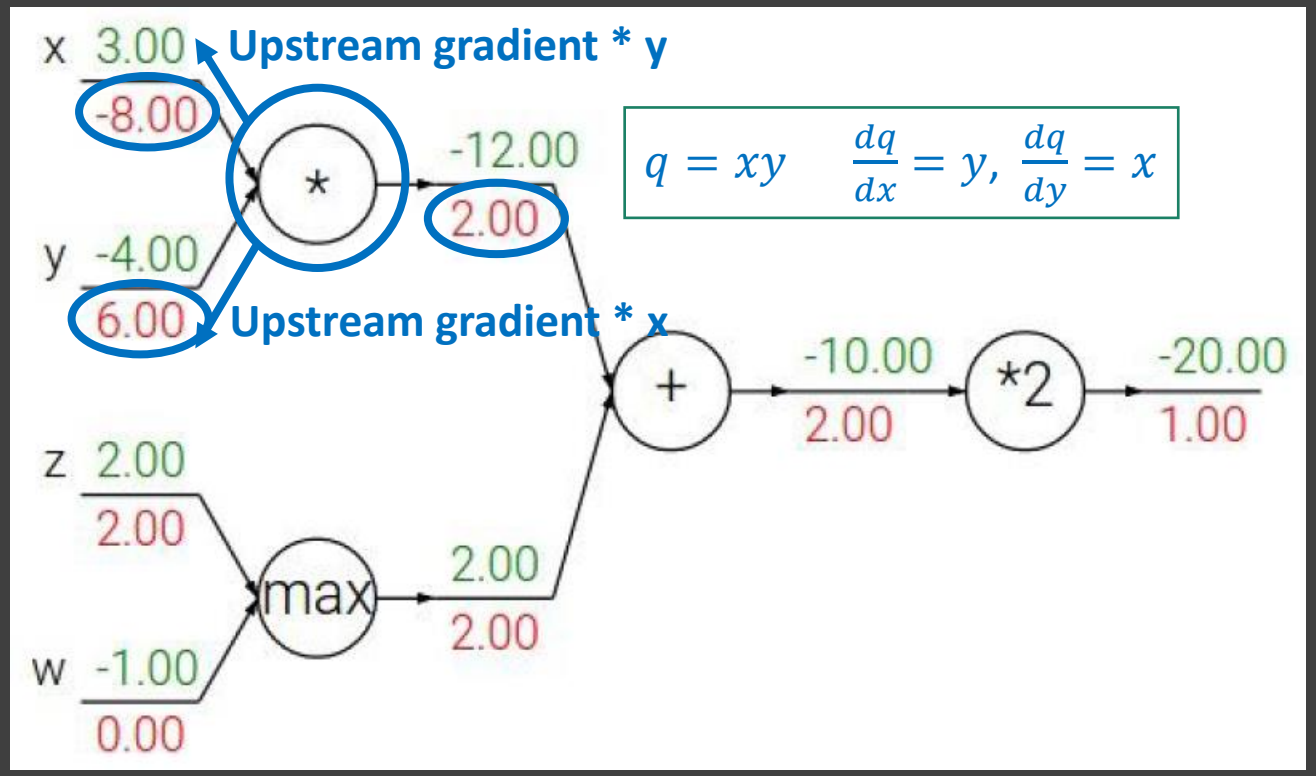
- max gate : gradient router



max gate 는 큰 value 를 가진 브랜치에 gradient 를 그대로 전하고, 작은 value 를 가진 브랜치에 0 을 전한다.

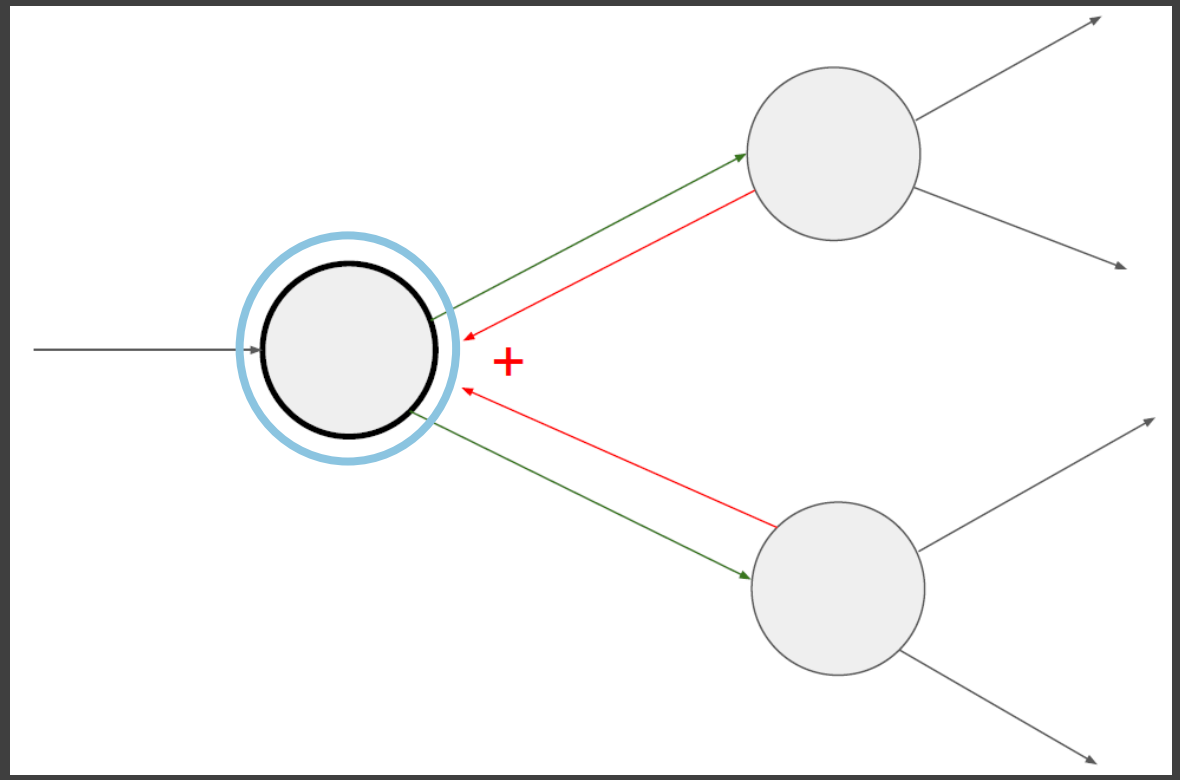
→ router

- mul gate : gradient switcher



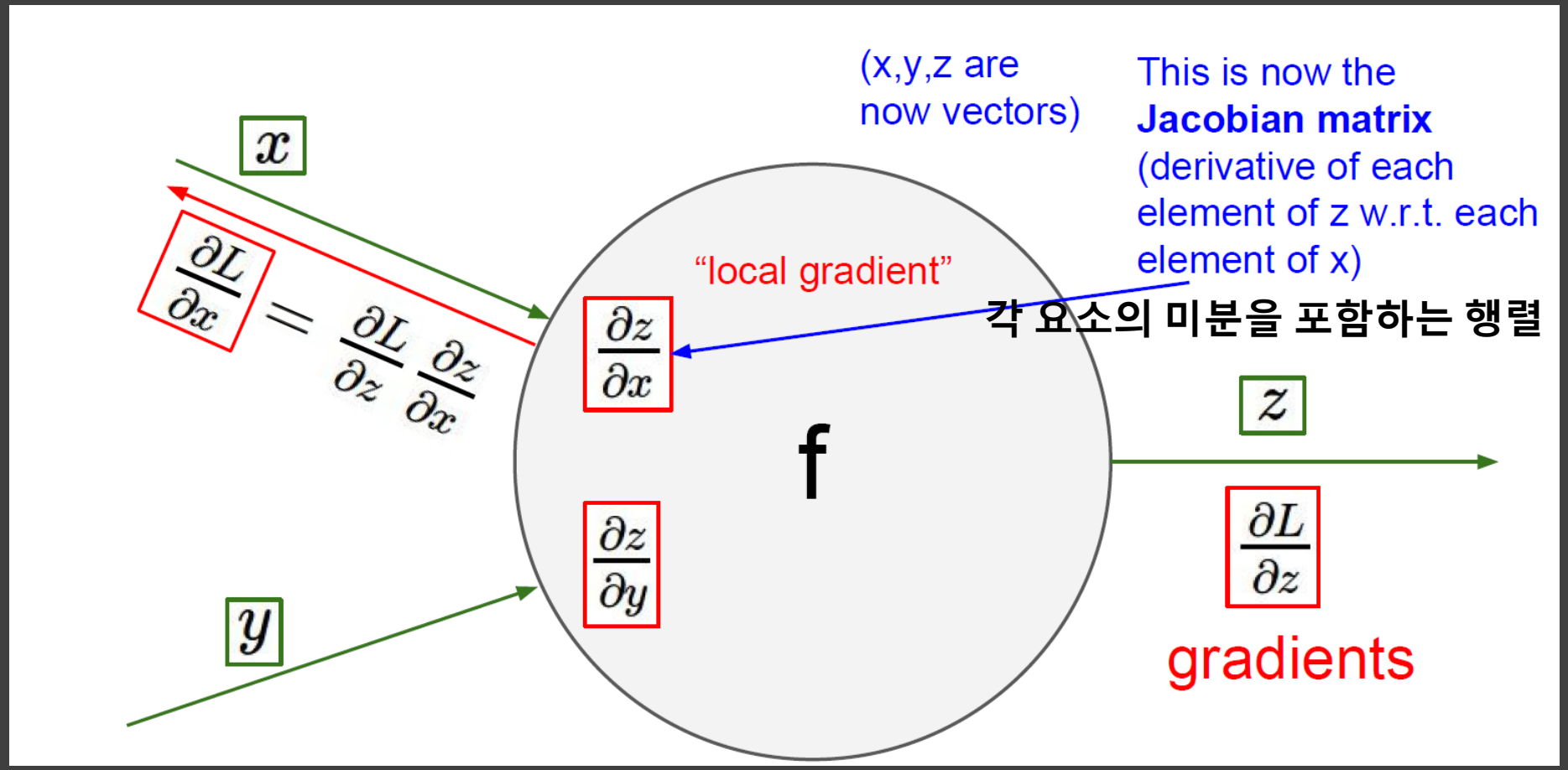
x 와 y 의 값이 스위치 되어
upstream gradient 와 곱해진다.
→ switcher

Gradients add at branches



Upstream gradient 가 여러 개 일 때 여러 노드와 연결된 하나의 노드에서 복수개의 gradient 들이 더해진다.

Gradients for vectorized code



여기서 잠깐!!! Jacobian Matrix 란?

- 하나의 함수가 n 개의 변수들을 가지고 있다고 하자. 그리고 그런 함수가 m 개 있다고 하자. $F_1(x_1, \ldots, x_n), \ldots, F_m(x_1, \ldots, x_n)$
- 이 함수들의 편미분들을 전부 계산하려고 한다.
- 이와 같이 편미분을 할 변수들이 많고, 그 변수들로 이루어져 있는 함수가 많을 때 Jacobian matrix 를 사용해서 미분을 한다.

$$J = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix}$$

F 가 2 개의 변수를 가진 2개의 함수로 구성되어 있으면, Jacobian Matrix의 크기는 2x2

F 가 2 개의 변수를 가진 3개의 함수로 구성되어 있으면, Jacobian Matrix 의 크기는 3x2

Jacobian matrix 예시

Example 1 [edit]

Consider the function $\mathbf{f} : \mathbb{R}^2 \to \mathbb{R}^2$, with $(x, y) \mapsto (f_1(x, y), f_2(x, y))$, given by

$$\mathbf{f}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} f_1(x, y) \\ f_2(x, y) \end{bmatrix} = \begin{bmatrix} x^2 y \\ 5x + \sin y \end{bmatrix}.$$

Then we have

$$f_1(x, y) = x^2 y$$

and

$$f_2(x, y) = 5x + \sin y$$

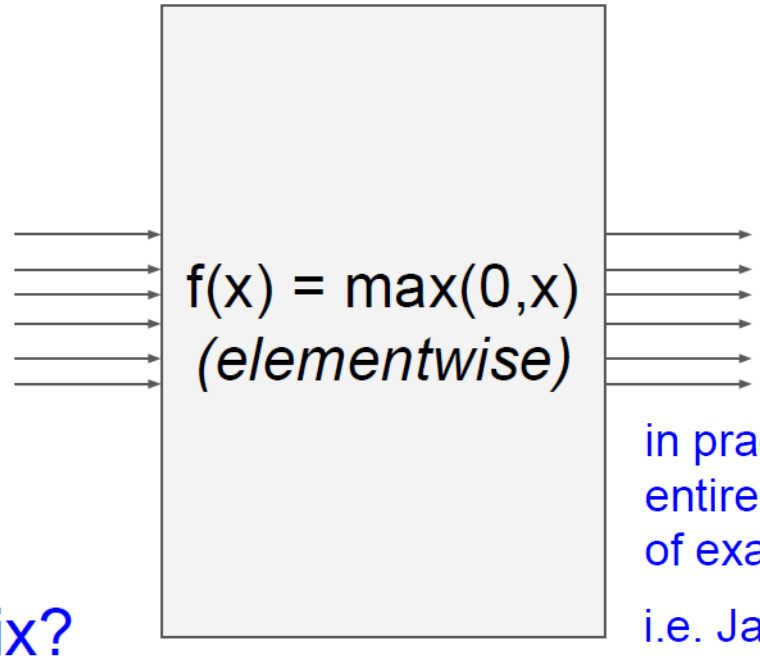
and the Jacobian matrix of $\mathbf{f}$ is

$$\mathbf{J}_{\mathbf{f}}(x, y) = \begin{bmatrix} \dfrac{\partial f_1}{\partial x} & \dfrac{\partial f_1}{\partial y} \\ \dfrac{\partial f_2}{\partial x} & \dfrac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} 2xy & x^2 \\ 5 & \cos y \end{bmatrix}$$

Vectorized operations

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

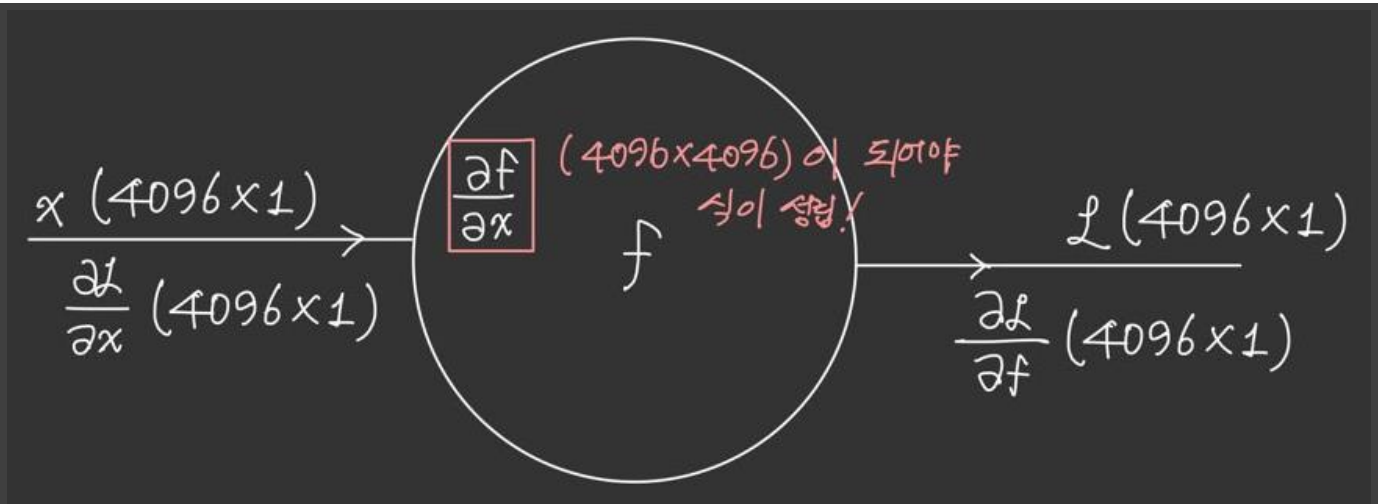
Jacobian matrix



Q: what is the size of the Jacobian matrix? [4096 x 4096!]

in practice we process an entire minibatch (e.g. 100) of examples at one time:

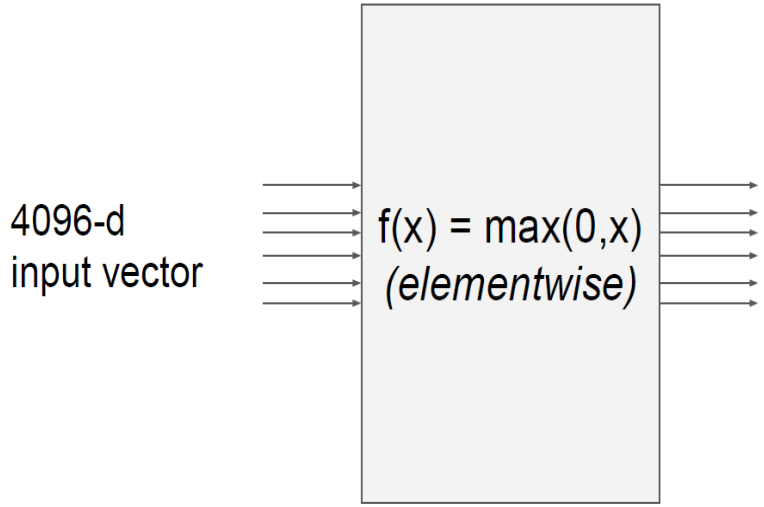
i.e. Jacobian would technically be a [409,600 x 409,600] matrix :\



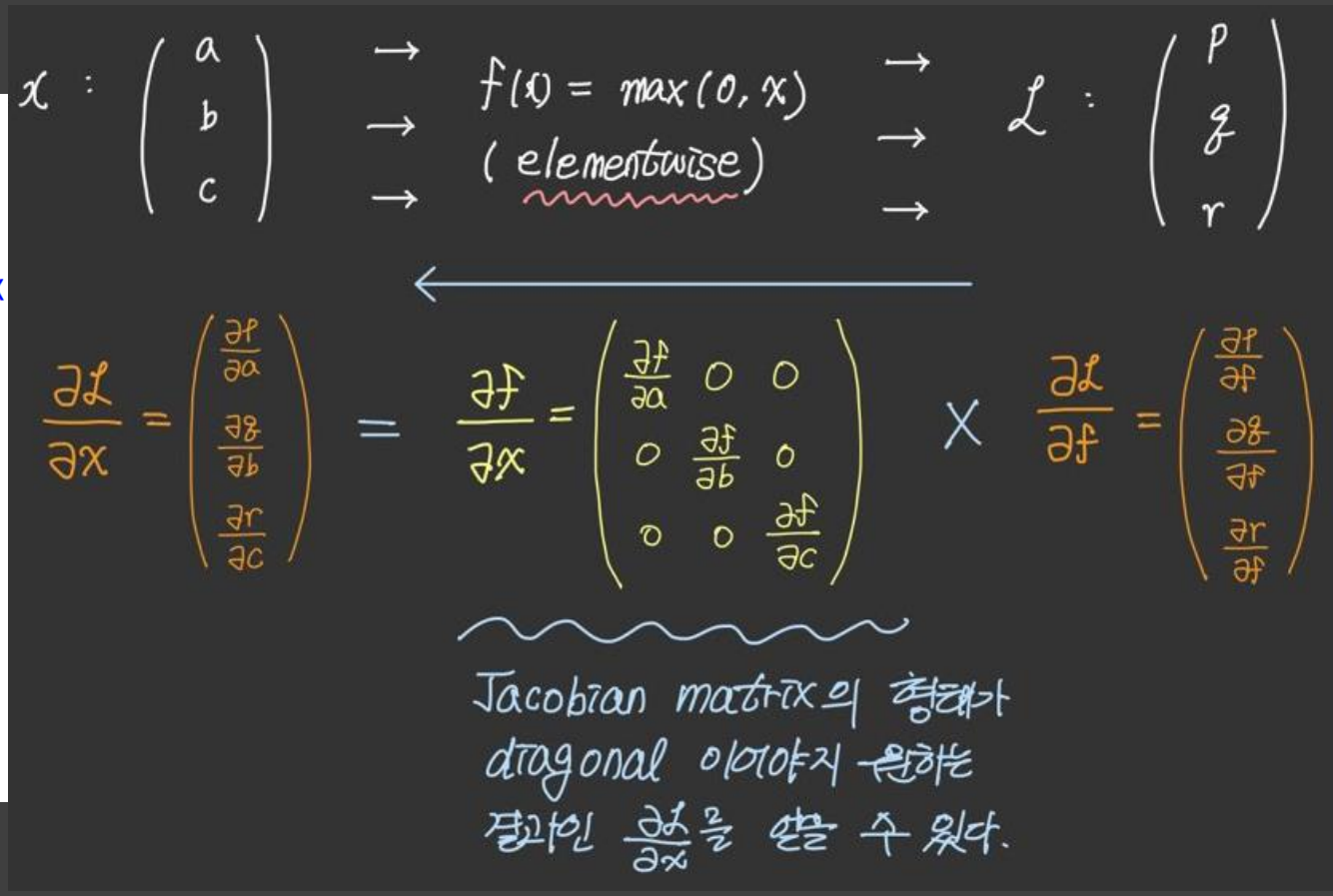
Vectorized operations

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

Jacobian matrix



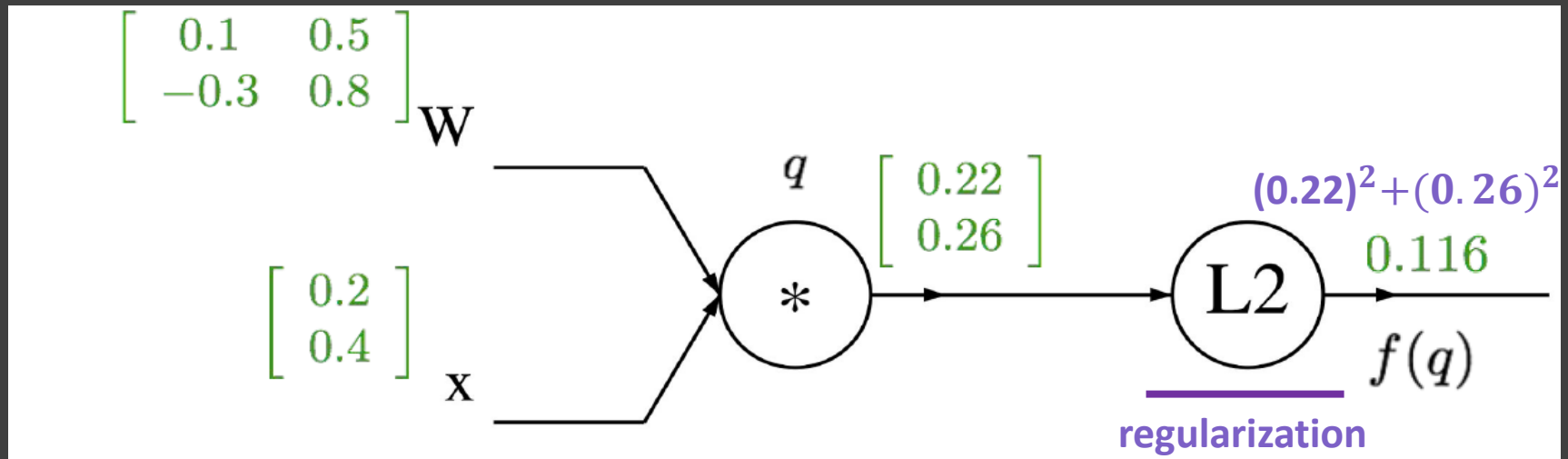
Q2: what does it look like?



➔ **Diagonal matrix**

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

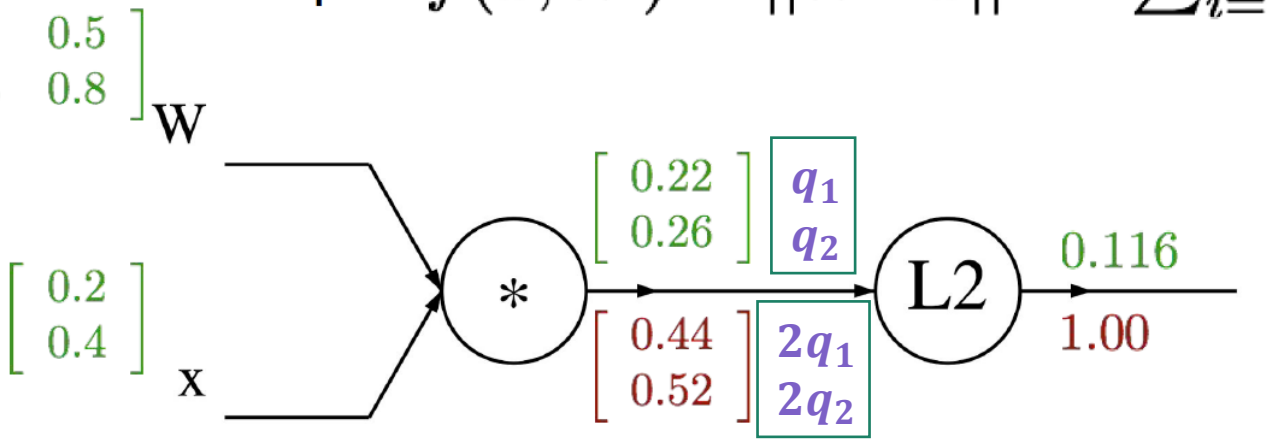
$\in \mathbb{R}^n$ $\in \mathbb{R}^{n \times n}$



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

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

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



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

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

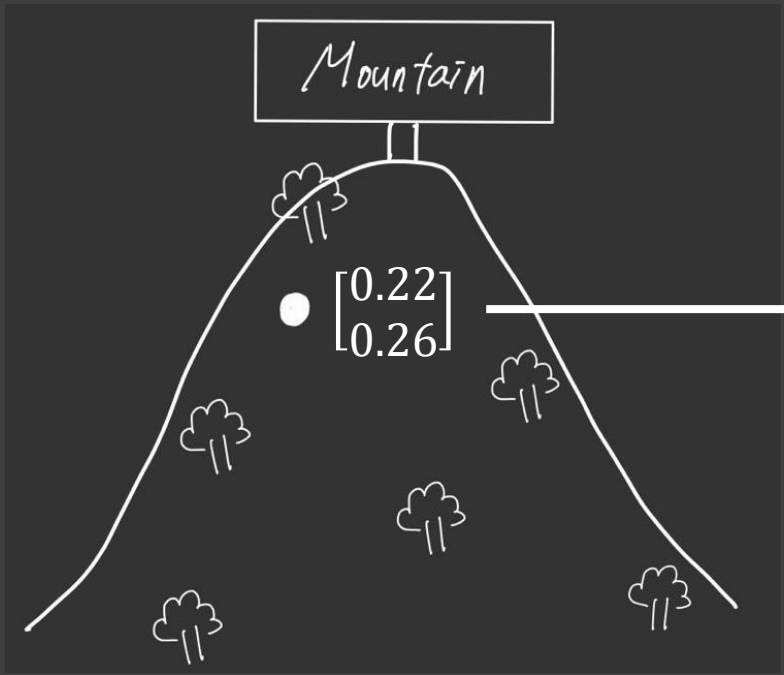
$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\nabla_q f = 2q$$

잠깐 짚고 넘어가기!

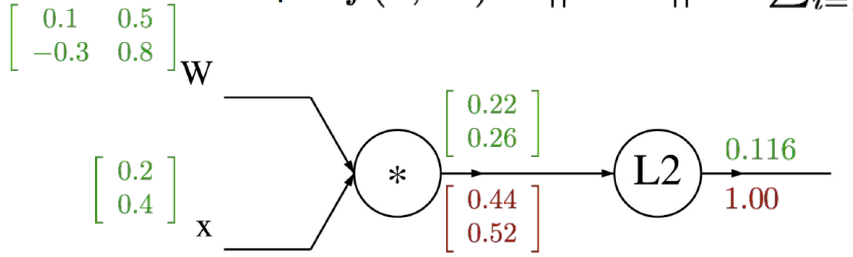
< $\begin{bmatrix}0.44\\0.52\end{bmatrix}$ 에 대한 직관 >

x 가 Loss 에 미치는 영향, y가 Loss 에 미치는 영향을 알고 싶다.



이 좌표에서 x 방향으로 움직이면 0.44 만큼 움직이고,
y 방향으로 움직이면 0.52 만큼 움직인다.

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



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

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

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

$$\nabla_q f = 2q$$

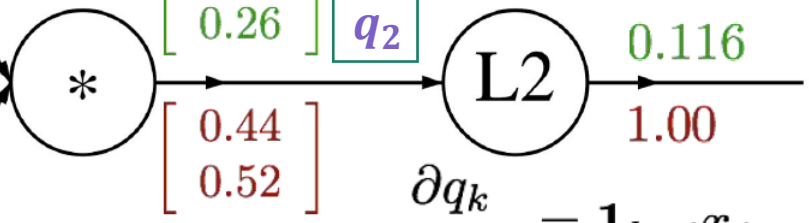
A vectorized example: $f(x, W) = ||W \cdot x||^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

$$\begin{bmatrix} 2q_1x_1 & 2q_1x_2 \\ 2q_2x_1 & 2q_2x_2 \end{bmatrix}$$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix} \text{ W}$$

$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix} \text{ x}$$



q_1
q_2

$$\nabla_W f = 2q \cdot x^T$$

Always check: The gradient with respect to a variable should have the same shape as the variable

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

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

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

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

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

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

벡터의 Gradient 들을 계산했으면, 그 결과가 변수의 shape 과 같은가 항상 체크 해야 한다. 그래야 error 를 막을 수 있다!

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

$W_{1,1}\ W_{1,2}$
$W_{2,1}\ W_{2,2}$

$$\begin{bmatrix} 0.1 & 0.5 \\ -0.3 & 0.8 \end{bmatrix}$$ W

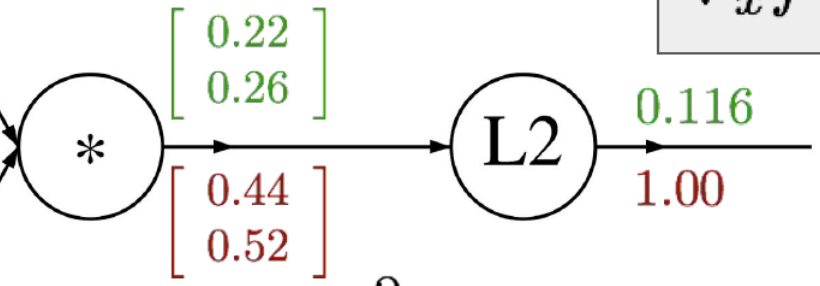
$$\begin{bmatrix} 0.088 & 0.176 \\ 0.104 & 0.208 \end{bmatrix}$$

$$\begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix}$$ x

$2(q_1W_{1,1} + q_2W_{2,1})$
$2(q_1W_{1,2} + q_2W_{2,2})$

$$\begin{bmatrix} -0.112 \\ 0.636 \end{bmatrix}$$

$$\nabla_x f = 2W^T \cdot q$$



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

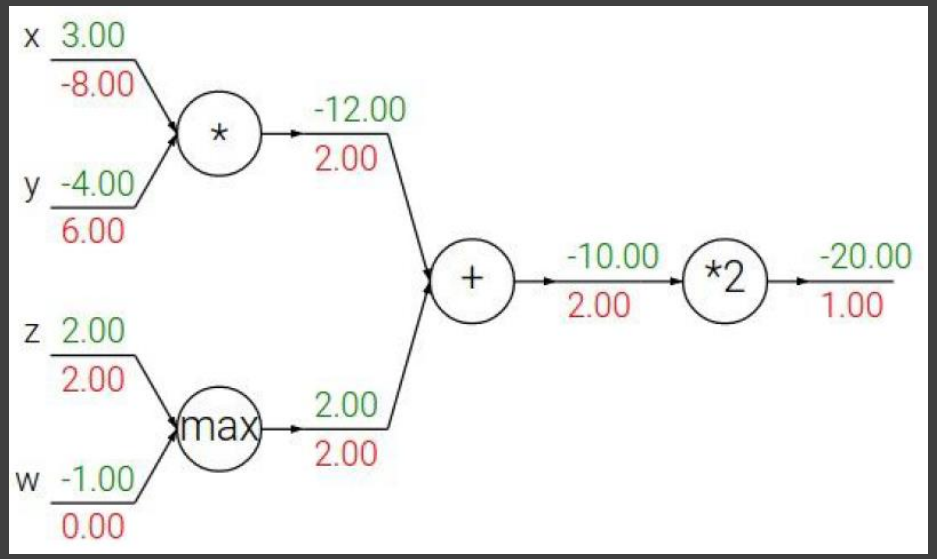
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

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

$$\frac{\partial f}{\partial x_i} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial x_i}$$

$$= \sum_k 2q_k W_{k,i}$$

Modularized implementation : forward/backward API

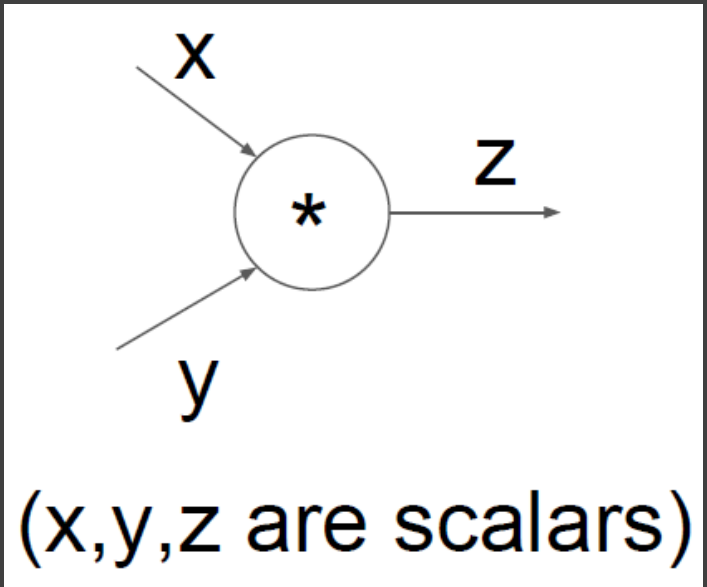


Graph (or Net) object *(rough psuedo code)*

```
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
```

앞에서부터 정렬된 순서로 수행!

뒤에서부터 계산




```
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]
```

Forward pass 의 값 x, ,y, z 를 저장
→ Forward pass 가 끝난 후
backward pass 에서 더 많이
사용하기 때문에

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

Upstream gradient

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

Example: Caffe layers

Popular deep learning framework

Branch: master ▾ caffe / src / caffe / layers / Create new file | Upload files | Find file | History

shelhamer committed on GitHub Merge pull request #4630 from BIGene/load_hdf5_fix — Latest commit e687e71 21 days ago

..

absval_layer.cpp	dismantle layer headers	a year ago
absval_layer.cu	dismantle layer headers	a year ago
accuracy_layer.cpp	dismantle layer headers	a year ago
argmax_layer.cpp	dismantle layer headers	a year ago
base_conv_layer.cpp	enable dilated deconvolution	a year ago
base_data_layer.cpp	Using default from proto for prefetch	3 months ago
base_data_layer.cu	Switched multi-GPU to NCCL	3 months ago
batch_norm_layer.cpp	Add missing spaces besides equal signs in batch_norm_layer.cpp	4 months ago
batch_norm_layer.cu	dismantle layer headers	a year ago
batch_reindex_layer.cpp	dismantle layer headers	a year ago
batch_reindex_layer.cu	dismantle layer headers	a year ago
bias_layer.cpp	Remove incorrect cast of gemm int arg to Dtype in BiasLayer	a year ago
bias_layer.cu	Separation and generalization of ChannelwiseAffineLayer into BiasLayer	a year ago
bnll_layer.cpp	dismantle layer headers	a year ago
bnll_layer.cu	dismantle layer headers	a year ago
concat_layer.cpp	dismantle layer headers	a year ago
concat_layer.cu	dismantle layer headers	a year ago
contrastive_loss_layer.cpp	dismantle layer headers	a year ago
contrastive_loss_layer.cu	dismantle layer headers	a year ago
conv_layer.cpp	add support for 2D dilated convolution	a year ago
conv_layer.cu	dismantle layer headers	a year ago
crop_layer.cpp	remove redundant operations in Crop layer (#5138)	2 months ago
crop_layer.cu	remove redundant operations in Crop layer (#5138)	2 months ago
cudnn_conv_layer.cpp	dismantle layer headers	a year ago
cudnn_conv_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_lcn_layer.cpp	dismantle layer headers	a year ago
cudnn_lcn_layer.cu	dismantle layer headers	a year ago
cudnn_lrn_layer.cpp	dismantle layer headers	a year ago
cudnn_lrn_layer.cu	dismantle layer headers	a year ago
cudnn_pooling_layer.cpp	dismantle layer headers	a year ago
cudnn_pooling_layer.cu	dismantle layer headers	a year ago
cudnn_relu_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_relu_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_sigmoid_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_softmax_layer.cpp	dismantle layer headers	a year ago
cudnn_softmax_layer.cu	dismantle layer headers	a year ago
cudnn_tanh_layer.cpp	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
cudnn_tanh_layer.cu	Add cuDNN v5 support, drop cuDNN v3 support	11 months ago
data_layer.cpp	Switched multi-GPU to NCCL	3 months ago
deconv_layer.cpp	enable dilated deconvolution	a year ago
deconv_layer.cu	dismantle layer headers	a year ago
dropout_layer.cpp	supporting N-D Blobs in Dropout layer Reshape	a year ago
dropout_layer.cu	dismantle layer headers	a year ago
dummy_data_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cpp	dismantle layer headers	a year ago
eltwise_layer.cu	dismantle layer headers	a year ago
elu_layer.cpp	ELU layer with basic tests	a year ago
elu_layer.cu	ELU layer with basic tests	a year ago
embed_layer.cpp	dismantle layer headers	a year ago
embed_layer.cu	dismantle layer headers	a year ago
euclidean_loss_layer.cpp	dismantle layer headers	a year ago
euclidean_loss_layer.cu	dismantle layer headers	a year ago
exp_layer.cpp	Solving issue with exp layer with base e	a year ago
exp_layer.cu	dismantle layer headers	a year ago

Caffe Sigmoid Layer

```cpp
#include <cmath>
#include <vector>

#include "caffe/layers/sigmoid_layer.hpp"

namespace caffe {

template <typename Dtype>
inline Dtype sigmoid(Dtype x) {
  return 1. / (1. + exp(-x));
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Forward_cpu(const vector<Blob<Dtype>*>& bottom,
    const vector<Blob<Dtype>*>& top) {
  const Dtype* bottom_data = bottom[0]->cpu_data();
  Dtype* top_data = top[0]->mutable_cpu_data();
  const int count = bottom[0]->count();
  for (int i = 0; i < count; ++i) {
    top_data[i] = sigmoid(bottom_data[i]);
  }
}

template <typename Dtype>
void SigmoidLayer<Dtype>::Backward_cpu(const vector<Blob<Dtype>*>& top,
    const vector<bool>& propagate_down,
    const vector<Blob<Dtype>*>& bottom) {
  if (propagate_down[0]) {
    const Dtype* top_data = top[0]->cpu_data();
    const Dtype* top_diff = top[0]->cpu_diff();
    Dtype* bottom_diff = bottom[0]->mutable_cpu_diff();
    const int count = bottom[0]->count();
    for (int i = 0; i < count; ++i) {
      const Dtype sigmoid_x = top_data[i];
      bottom_diff[i] = top_diff[i] * sigmoid_x * (1. - sigmoid_x);
    }
  }
}

#ifdef CPU_ONLY
STUB_GPU(SigmoidLayer);
#endif

INSTANTIATE_CLASS(SigmoidLayer);


}  // namespace caffe
```

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

$$(1-\sigma(x))\,\sigma(x)$$

* top_diff (chain rule)

Upstream gradient

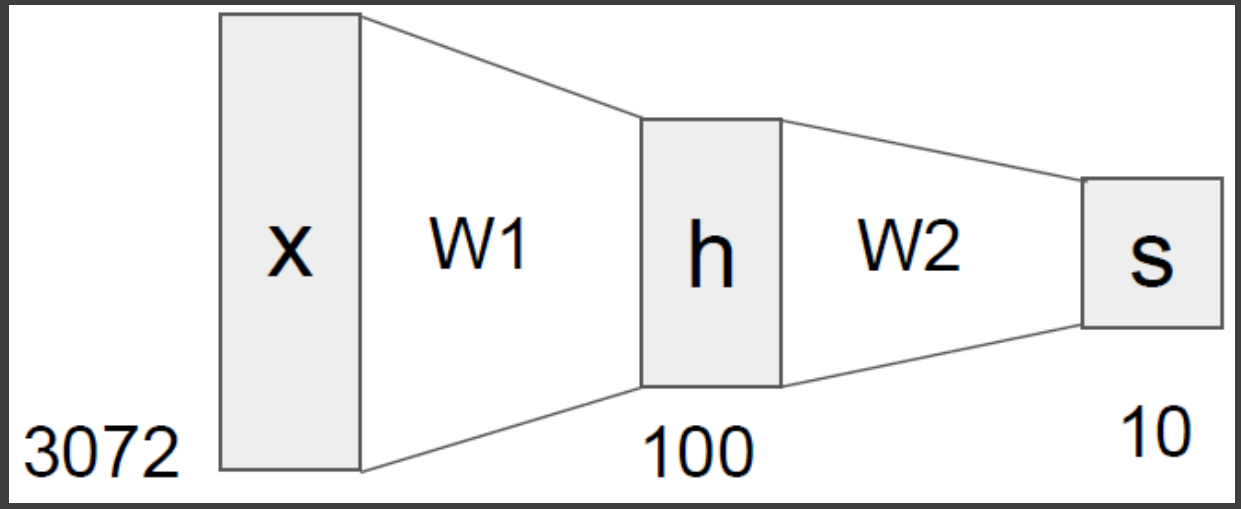
Summary of backpropagation

- Neural nets 은 굉장히 크고 복잡하기 때문에 모든 파라미터들의 gradients 를 손으로 계산하는 것은 비현실적이다. 그래서 gradients 의 계산에 backpropagation 을 사용한다.
- Backpropagation 은 chain rule 을 재귀적으로 적용한 구조이다.
- Graph structure 구현
  - Forward 구현 : 연산을 하고 gradient 계산에 필요한 중간 변수들을 메모리에 저장한다. 이것은 나중에 backward pass 에서 사용된다.
  - Backward 구현 : 입력에 대한 Loss 함수의 gradients 를 계산하기 위해 chain rule 을 사용한다.

Neural Networks

Neural networks : without the brain stuff

- (Before) Linear score function $f = Wx$
- (Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$



Neural networks : without the brain stuff

- (Before) Linear score function $f = Wx$
- (Now) 2-layer Neural Network $f = W_2 \max(0, W_1 x)$
- 3-layer Neural Network $f = W_3 \max(0, W_2 \max(0, W_1 x))$

Full implementation of training a 2-layer Neural Network needs ~20 lines :

```python
import numpy as np
from numpy.random import randn

N, D_in, H, D_out = 64, 1000, 100, 10
x, y = randn(N, D_in), randn(N, D_out)
w1, w2 = randn(D_in, H), randn(H, D_out)

for t in range(2000):
  h = 1 / (1 + np.exp(-x.dot(w1)))
  y_pred = h.dot(w2)
  loss = np.square(y_pred - y).sum()
  print(t, loss)

  grad_y_pred = 2.0 * (y_pred - y)
  grad_w2 = h.T.dot(grad_y_pred)
  grad_h = grad_y_pred.dot(w2.T)
  grad_w1 = x.T.dot(grad_h * h * (1 - h))

  w1 -= 1e-4 * grad_w1
  w2 -= 1e-4 * grad_w2
```

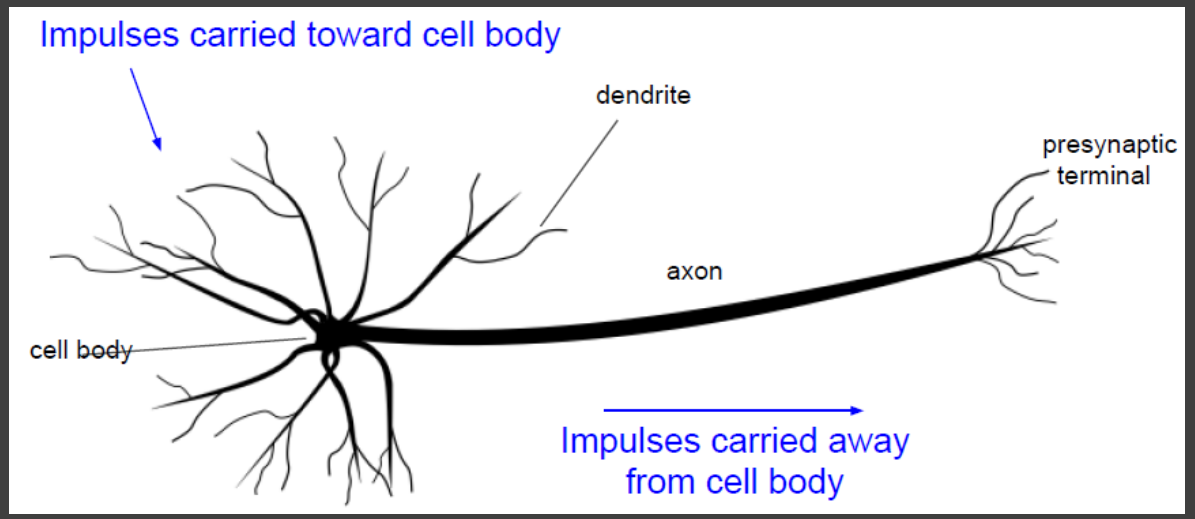
변수 저장

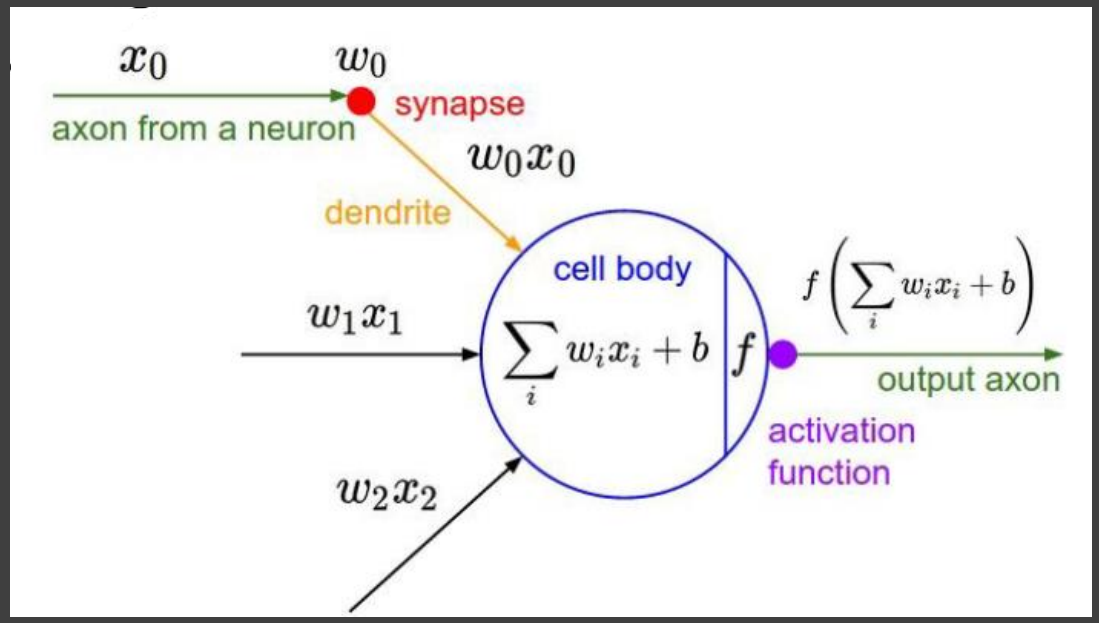
반복문을 2000번 돌리기

Forward pass

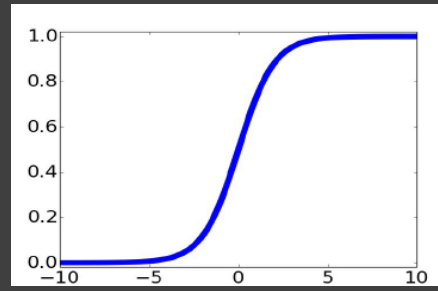
Backward pass

경사 하강법 → w 를 갱신





뉴런의 자극 처리 메커니즘과 neural network 의 메커니즘이 유사하다.
Activation function 을 통해 결과값이 나오는 것도 유사하게 볼 수 있다.
하지만 정확히는 뉴런의 메커니즘과 같지 않기 때문에 참고만 하는 것이 좋다.



sigmoid activation function

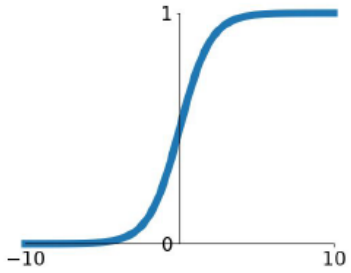
$$\frac{1}{1+e^{-x}}$$

```
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 func
    return firing_rate
```

Activation functions

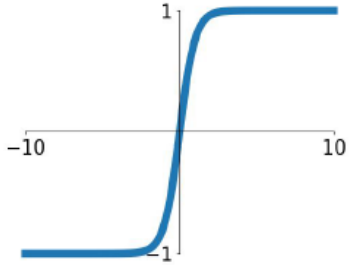
Sigmoid

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



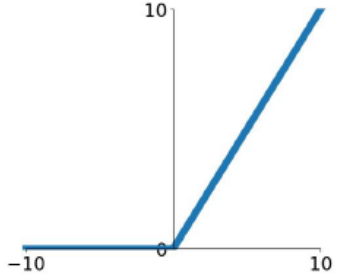
tanh

$$\tanh(x)$$



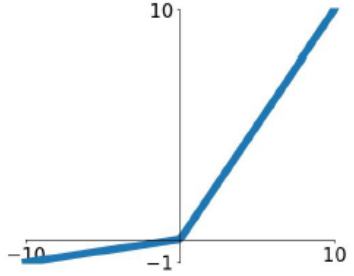
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

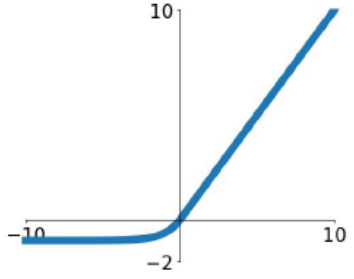


Maxout

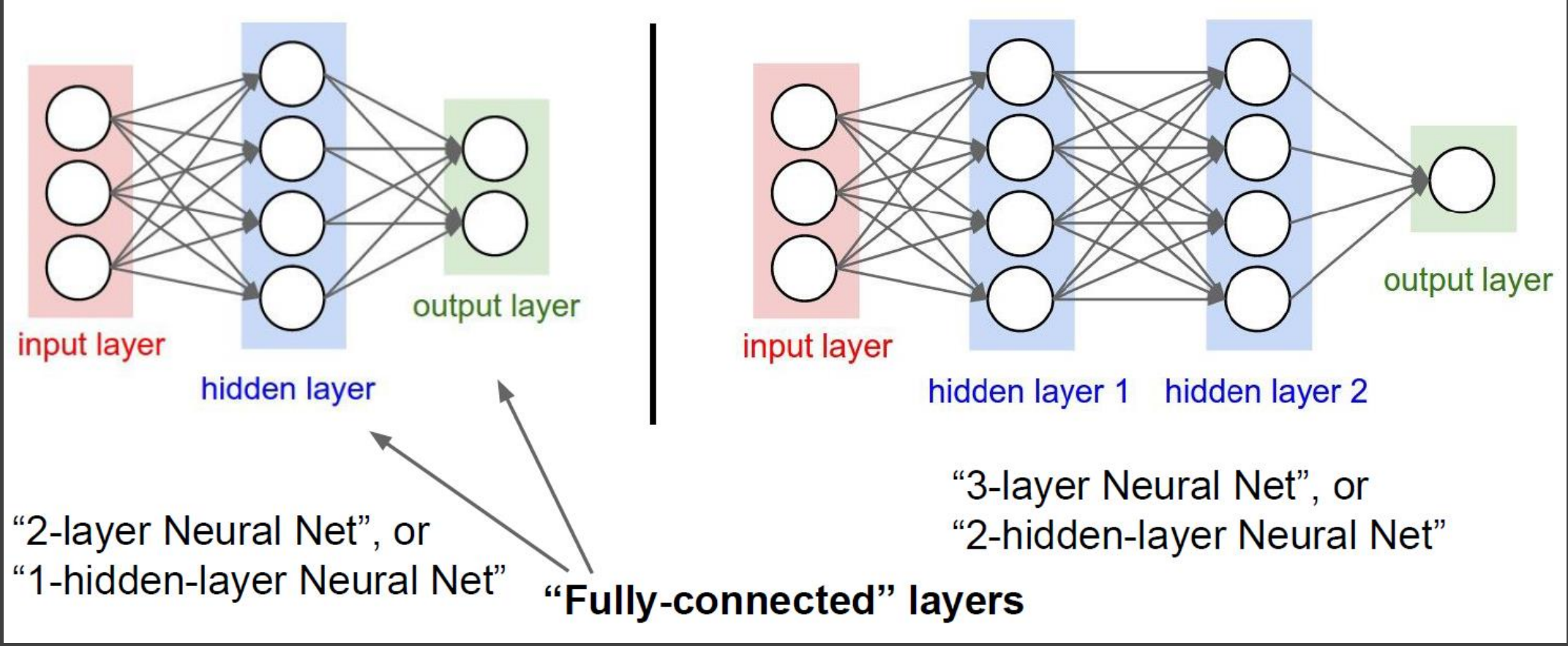
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Neural networks : Architectures

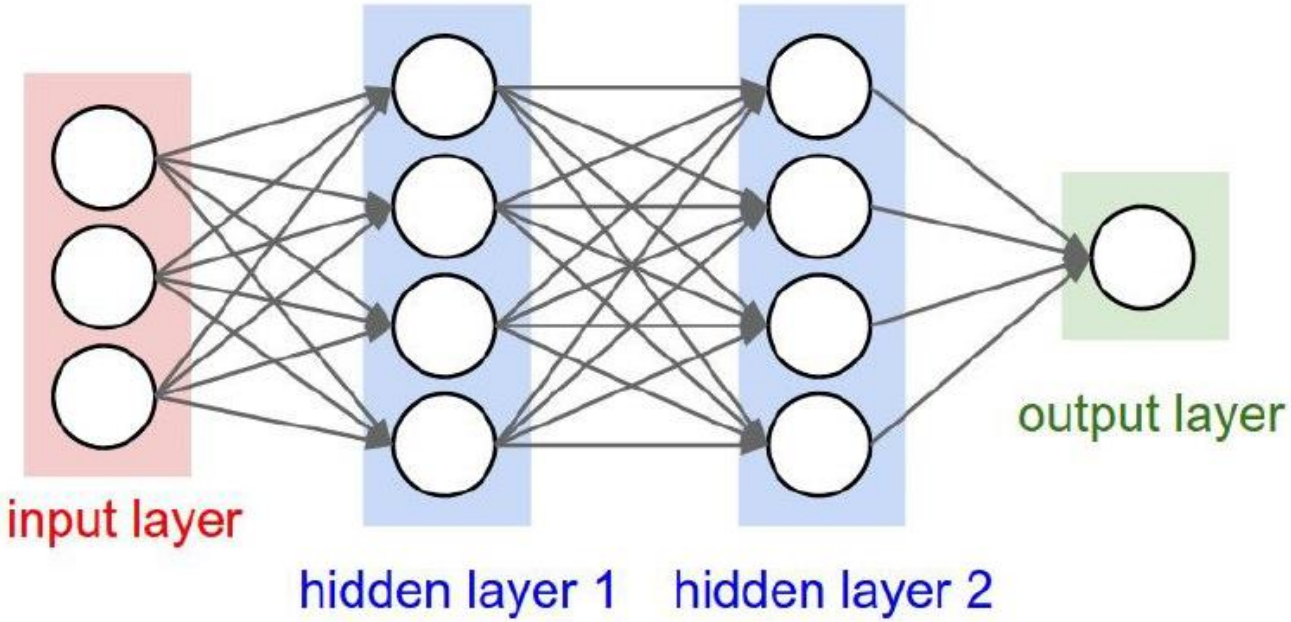


Example feed-forward computation of a neural network

```
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
```

We can efficiently evaluate an entire layer of neurons.

Example feed-forward computation of a neural network




```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Summary

- Neuron 을 fully-connected layers 로 재배열하는지에 대해 이야기했다.
- layer 를 코드로 옮길 때, 효율적인 vectorized code 를 사용할 수 있다.
- Neural networks 는 실제 *neural* 과는 다르다.