Backporpagation

HY-KIERA

reference

http://cs231n.github.io/optimization-2

Book 『Deep Learning from Scratch』

How to update the gradient

Calculus

backpropagation

etc.

Backpropagation

: a way of computing gradients of expressions through recursive application of chain rule

$\nabla f(x)$: the gradient of f at x (where x is a vector of inputs)

Gradient

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

$\nabla f(x)$: the vector of partial derivatives

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

e.g.

$$f(x, y) = \max(x, y) \quad \text{->} \quad \frac{\partial f}{\partial x} = 1(x \geq y) \qquad \frac{\partial f}{\partial y} = 1(y \geq x)$$

The (sub)gradient is 1 on the input that was larger and 0 on the other input

Gradient

Calculus

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

$$\text{-> } f(x+h) = f(x) + h\frac{df(x)}{dx}$$

But, derivative is only informative for tiny, infinitesimally small changes on the inputs,

As indicated by the $\lim_{h \to 0}$ in its definition.

Chain Rule

$$f(x, y, z) = (x + y)z \quad \text{->}$$

$$q = x + y \qquad \frac{\partial f}{\partial q} = \mathrm{z}, \qquad \frac{\partial f}{\partial z} = q$$

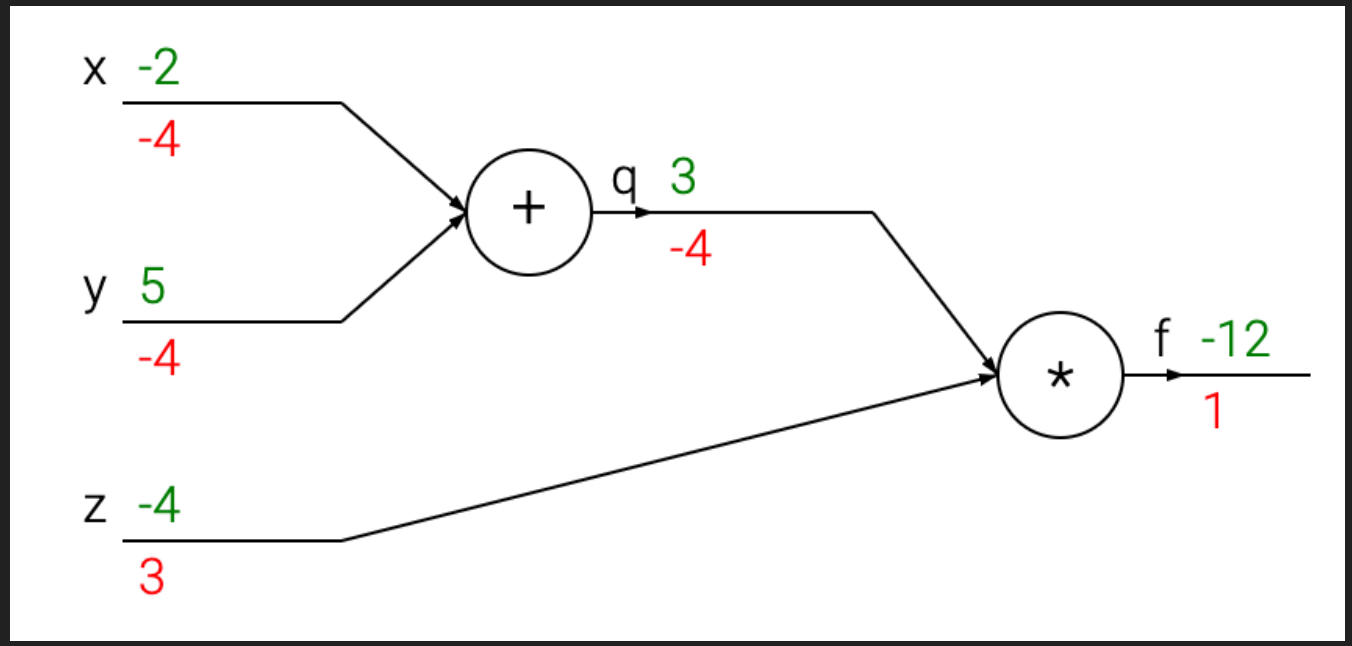
$$f = qz \qquad \frac{\partial q}{\partial x} = 1, \qquad \frac{\partial q}{\partial y} = 1$$

We do not necessarily care about the gradient on the intermediate value q-the value of $\frac{\partial q}{\partial x}$ is not useful. Instead, we are ultimately interested in the gradient of f with respect to its inputs x, y, z.

chain rule -> the correct way to "chain" these gradient expressions together is through multiplication.

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

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



forward

backward - backpropagation

Backpropagation

- A beautifully local process

Every gat in a circuit diagram gets some inputs and can right away compute two things:

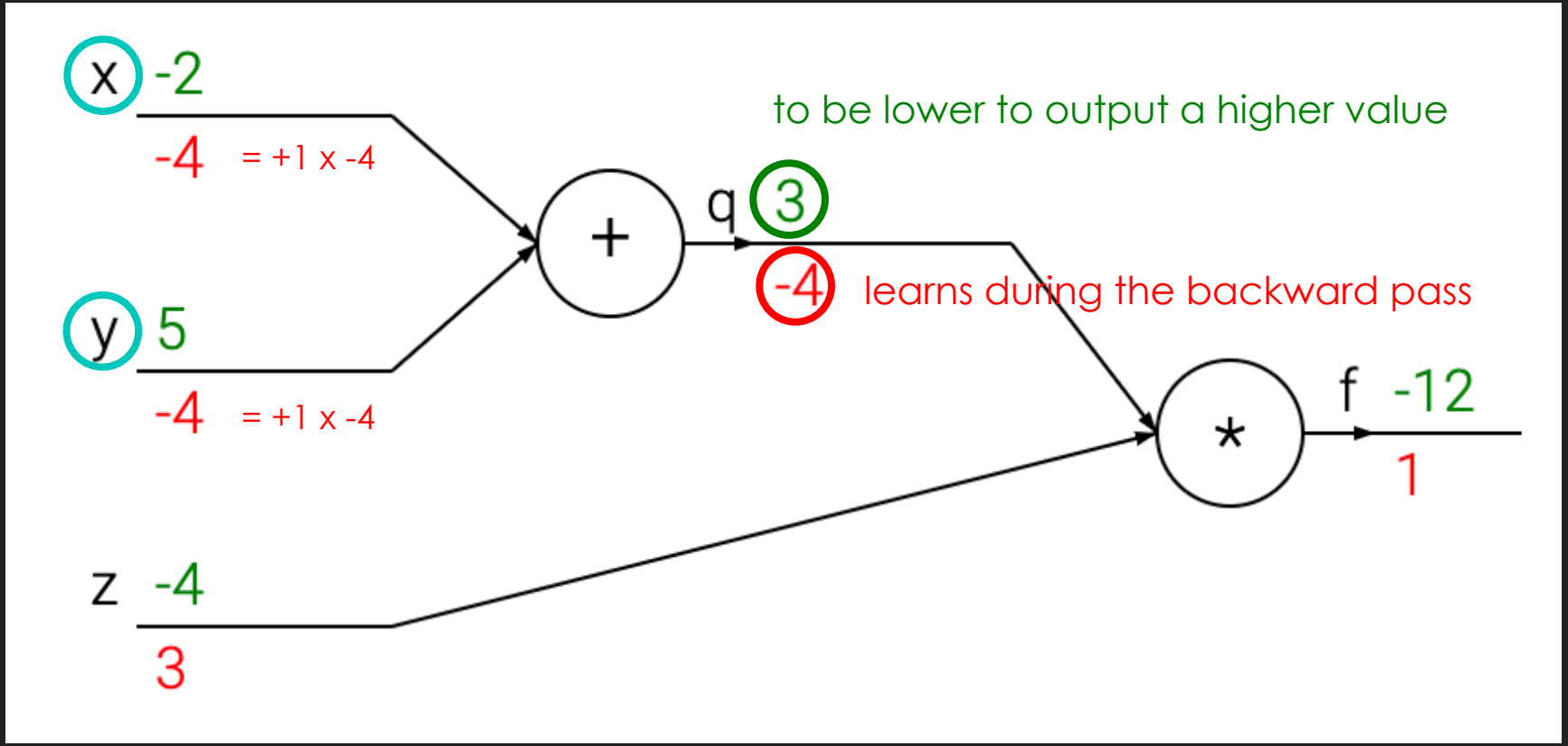
1. Its output value
2. The *local* gradient of its inputs with respect to its output value

-> Completely independently without being aware of any of the details of the full circuit that they are embedded in.

However, once the forward pass is over, during backpropagation the gate will eventually learn about the gradient of its output value on the final output of the entire circuit.

-> Chain Rule

Inputs[-2,5] -> local gradient : +1



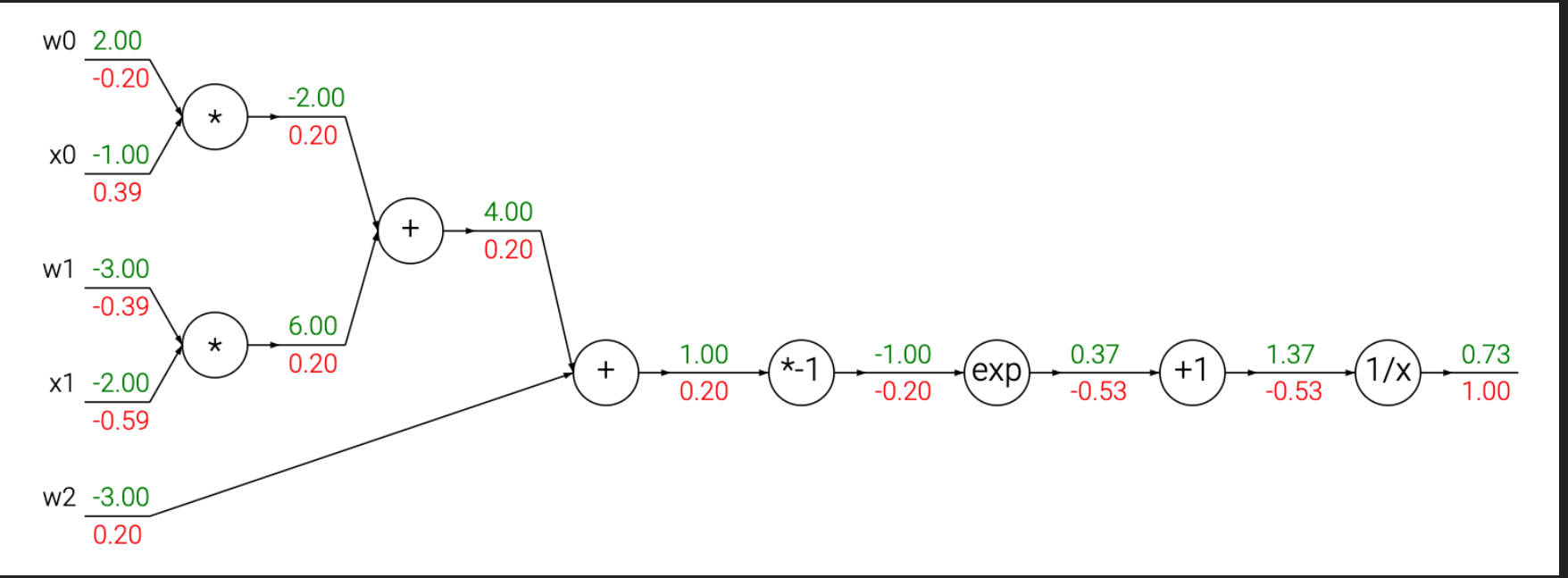
⇒ If x,y were to decrease(responding to their negative gradient) then

the add gate's output would decrease, which in turn makes the multiply gate's output increase

Backpropagation

A 2-dim neuron using sigmoid activation function

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



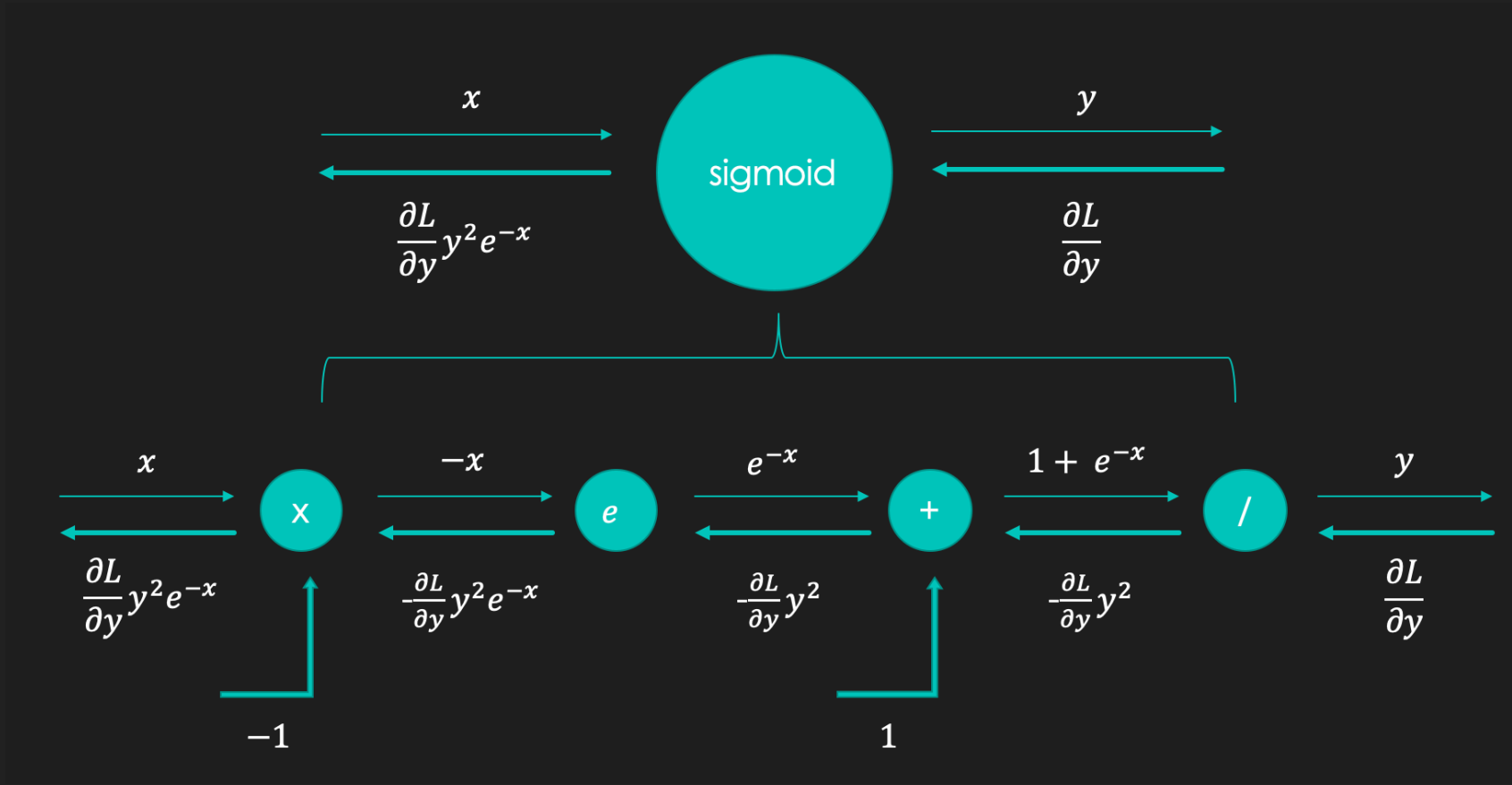
Backpropagation

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

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

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

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



Backpropagation

Patterns in backward flow

Add gate

Always takes the gradient on its output and distributes it equally to all its inputs, regardless of what their values were during the forward pass.

Max gate

Routes the gradient.
Unlike the add gate, the max gate distributes the gradient (unchanged) to exactly one of its inputs (the input that had the highest value during the forward pass)

Multiply gate

Its local gradient are the input values (except switched)