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.

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

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

Chain Rule

$$f(x, y, z) = (x + y)z \quad \text{->} \quad \begin{array}{lll} q = x + y & \dfrac{\partial f}{\partial q} = \mathrm{z}, & \dfrac{\partial f}{\partial z} = q \\ f = qz & \dfrac{\partial q}{\partial x} = 1, & \dfrac{\partial q}{\partial y} = 1 \end{array}$$

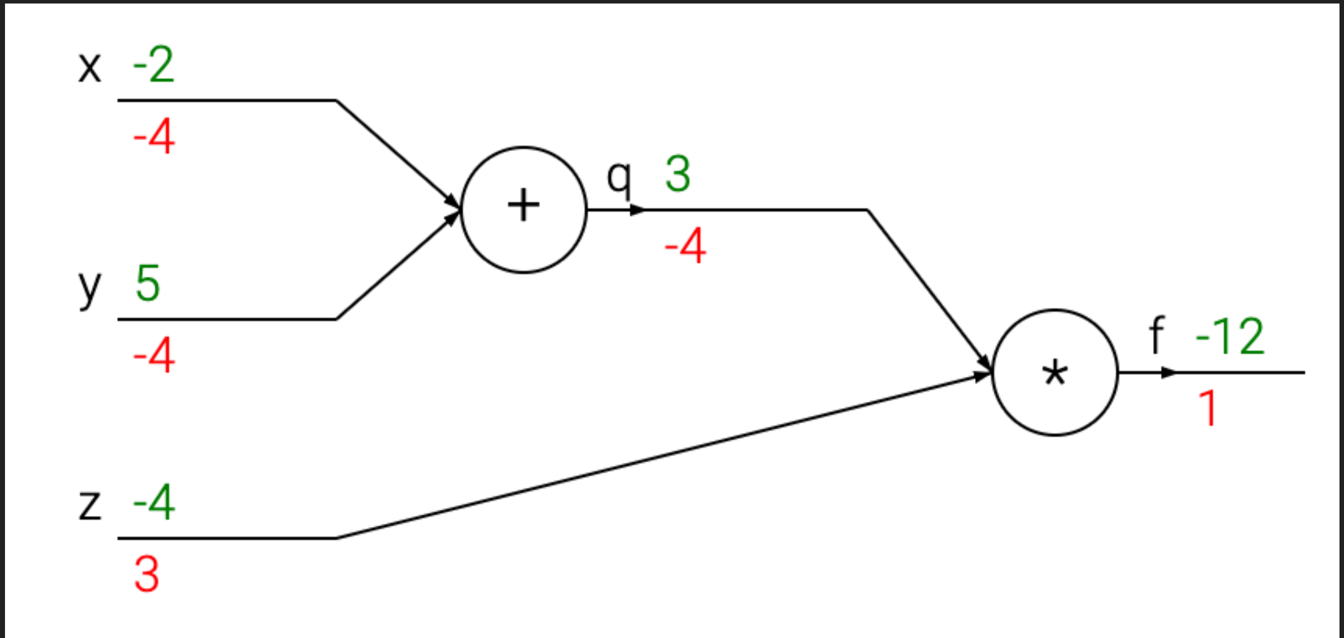
We donot necessarily care about the gradient on the intermediate value q-the value of $\frac{\partial q}{\partial x}$ is not useful.

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

=> This extra multiplication (for each input) due to the chain rule can turn a single and relatively useless gate into an cog in a complex circuit such as an entire neural network.

Backpropagation

sigmoid

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