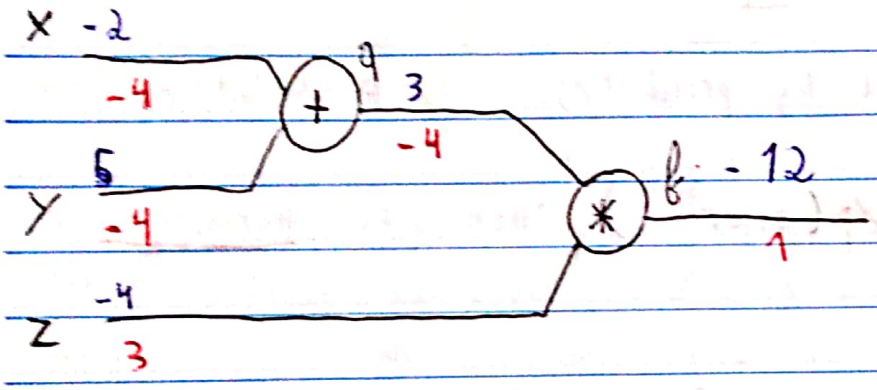
Backpropagation - Neural Networks 1

Backpropagation is done using calculus, it would take too much time to calculate using limits or so on. Also, it's done by calculating derivatives (gradients) in a graph.



We want:

$$\frac{df}{dx}, \frac{df}{dy}, \frac{df}{dz}$$

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

$q = x + y$ $\qquad$ $f = q \cdot z$

what is $\frac{df}{df}$? the gradient of f in respect to f

is the identity function, which is 1, the gradient of the identity is 1

Now what is $\frac{df}{dz}$?

Since $f = q \cdot z$ $\frac{df}{dq} = z$ and $\frac{df}{dz} = q$

So $\frac{df}{dz} = 3$. What it says, is that the

influence of z is positive and with a force of 3. If I increase z by an amount h, f will react by increasing 3h.

So $\frac{df}{dq} = -4$. And again, that means that if I

increase q by an amount h, f will react by decreasing 4h, which is -4h

now to finish this little graph, what is

$\frac{df}{dy}$ and $\frac{df}{dx}$?

Chain rule ~

we can define $\frac{df}{dy}$ as $\frac{df}{dq} \cdot \frac{dq}{dy}$ so

$\frac{df}{dy} = \frac{df}{dq} \cdot \frac{dq}{dy}$ we call $\frac{dq}{dy}$ the local gradient of y on q.

and we call $\frac{df}{dq}$ kind of the global gradient of q on F. We know $\frac{df}{dq} = -4$, and since q is a + gate. $\frac{dq}{dx} = 1$ and $\frac{dq}{dy} = 1$, which means that if we add 1 on X or Y, q also changes by increasing 1.

So: $\frac{df}{dy} = \frac{df}{dq} \cdot \frac{dq}{dy} = -4 \cdot 1 = -4$.

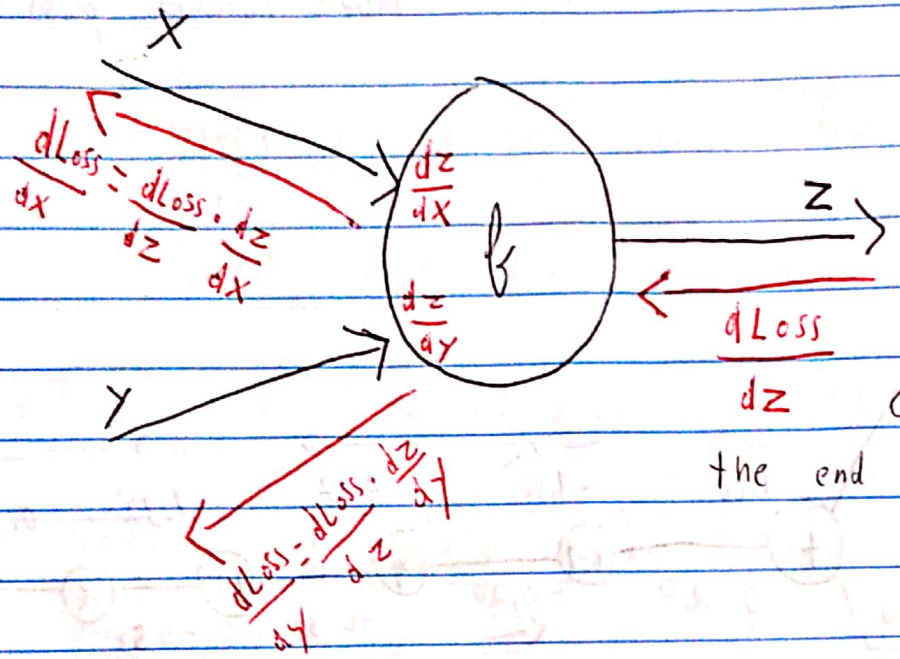
$\frac{df}{dx} = \frac{df}{dq} \cdot \frac{dy}{dx} = -4 \cdot 1 = -4$

And that means that if you increase X or Y by an amount h, they will be multiplied by z before getting into f, so f will also react by $-4h$.

Generalizing

We've seen that usually we have this gates which perform a known function, so we describe this process as:

Local gradients



dz comes from loss at the end backwards.

Where f could be a + gate, a X gate or any Function, like a Sigmoid For example.

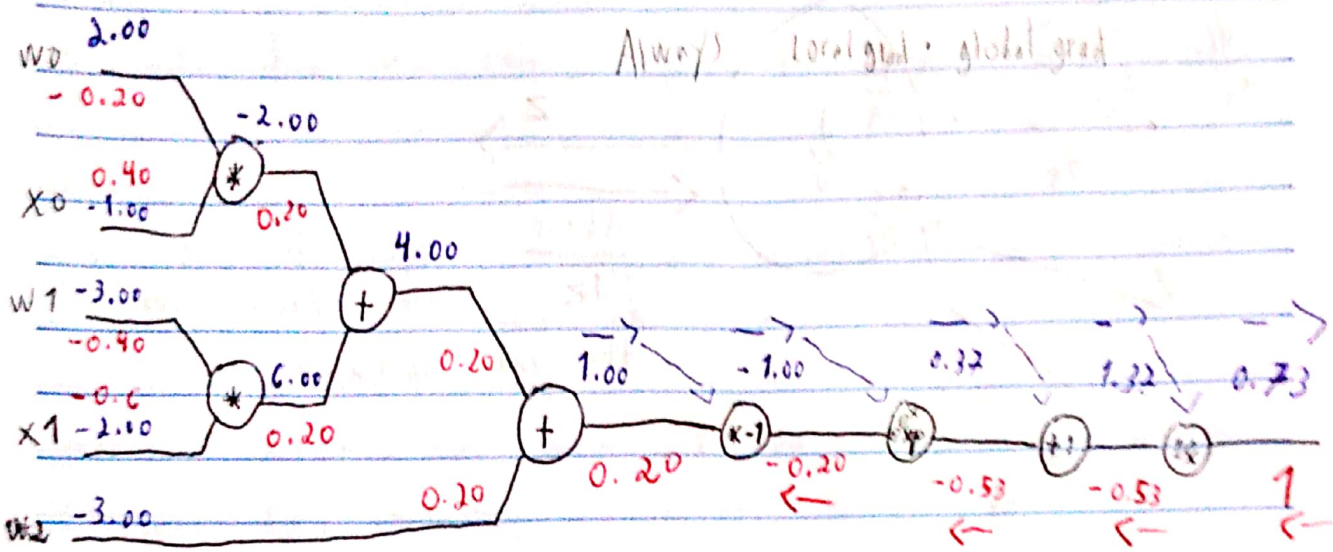
we can Calculate the local gradient right away, since we know the f(x) Function.

Now let's take another example:

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

Red = gradient

blue = Forward pass



Always local grad · global grad

Identity gradient is 1

Local gradient For $\frac{1}{x}$: $x^{-1} = \frac{1}{x}$

$$\frac{df}{dx} = -1x^{-1-1} = -1 \cdot x^{-2} = -\frac{1}{x^2}$$

$$-\frac{1}{1.37^2} \cdot 1 = -0.53 \cdot 1 = -0.53$$ → identity

The +1 gate doesn't change the final result, the Local gradient is 1, multiplied by the previous gradient, which For that gate is the global gradient.

So again -0.53

Next we have -1 getting into the euler gate (e^x) where $x = -1$,

The derivative of e^x is also e^x. So the gradient here is e^{-1} (local gradient) multiplied by the global gradient, that for this gate is -0.53

$$e^{-1} \cdot (-0.53) \tilde{=} 0.367 \cdot (-0.53) \tilde{=} 0.195 \tilde{=} 0.20$$

For the $\cdot(-1)$ gate we have the gradient multiplied by -1, so we just do it again

$$-0.20 \cdot (-1) = 0.20$$

On the + gate again local gradients are 1, since the global is 0.20, for any input in this gate the gradient is 0.20.

* Patterns in backward Flow *

As we see, the plus gate works like a gradient distributor, its spreads equally the gradient that came in to it's children.

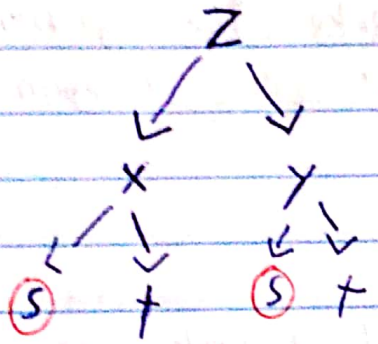
For the multiply gate, its works kind of a gradient "switcher", as you multiply what came in into the other variable for each element.

Also the max function, like $\max(x, y)$ is a gradient router, because you have a local gradient of 1 on the biggest number, and 0 on the smaller. Thus, it routes the gradient that came in into

the biggest input.

When we have 2 or more branches of gradients coming in a gate, the correct thing to do by multivariate chain rule, is to add them.

Also something interesting, this Chain rule process with gradients is also called partial derivatives.



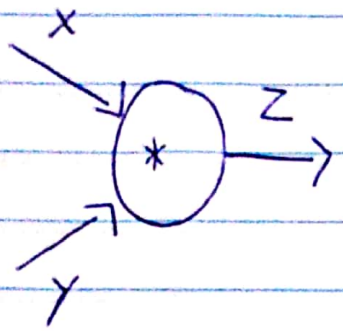
$$\frac{dz}{ds} = \frac{dz}{dx}\cdot\frac{dx}{ds} + \frac{dz}{dy}\cdot\frac{dy}{ds}$$

$\frac{dz}{ds}$ is called the partial derivative of z with respect to s. And as you can see, we add the branches.

This graphs are also called DAG (Directed acyclic graph), which is a directed graph with no directed cycles. Thus, there's no loops.

Implementation

Yes, I'll write python3 on my notebook:



x, y, z are scalars

```
class MultiplyGate:

  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 # [dL/dz * dz/dx]
    dy = self.x * dz # [dL/dz * dz/dy]
    return [dx, dy]
```

As you can see, everytime we perform operations on forward, we need that every single gate to remember the calculations it has performed, the ones we need acess on backward pass, in this case x and y because its a multiply gate. During the forward pass a huge amount of things gets cached in the memory, because we need them on Backward pass.

You might be familiar with PyTorch's:

```
with torch.no_grad():
```

And that's it, you use on inference in order to Avoid this memory consume when doing inference, on the Final model, or in the validation pass.