Calculus and Backpropagation

Massimiliano Tomassoli
(reverse(*5102mnhuik*)@gmail.com)

06/15/2016

1 Foreword

A few weeks ago I posted a tutorial about *Calculus and Backpropagation* on */r/MachineLearning* and /r/math. Since then, I received several requests to rewrite it in LaTeX. I can't say I was thrilled at the prospect of having to relearn *TikZ* (I used *matplotlib* for my other papers), but I finally took the plunge and here we are.

What follows is the *original* text with corrections and some additional clarifications. As always, let me know if you find any mistakes or if you have any suggestions.

Since I don't have any *space limitations* and I suspect—but maybe I'm wrong—that many of you are going to read this tutorial in digital form, I decided that the convenience of having all the pictures in the right place should have the priority over space saving considerations. You're free to try to change my mind on this (and any other) matter: I just try to do what seems to work best.

2 What to expect

This started out as an answer to a question that was asked on this forum, but I got carried away and wrote a full-fledged tutorial! It took me 10+ hours to complete it, so I hope you'll find it useful. In particular, I hope it'll help beginners understand Backprop once and for all.

I should warn you that I don't believe that giving specific and ad hoc derivations of the backprop is any useful in the long run. Many popular tutorials and books choose this approach, which, I think, isn't helping. I strongly believe that *abstraction* and *modularization* are the right way to explain things, when possible.

If you took some calculus course, but you didn't develop an intuition for it, maybe this short tutorial is what you need.

3 Starting from the start

For simplicity, we'll assume that our functions are *differentiable* at any point of their domain and that every scalar is a real number.

Let's say we have an $R \to R$ function $h(x)$. Let's focus on a particular x and consider the portion of h around x, i.e. h restricted to the interval $[x-dx, x+dx]$. Let's call it $h\{x, dx\}$. If dx is big, $h\{x, dx\}$ may have some curvature, but if we reduce dx more and more, $h\{x, dx\}$ will become flatter and flatter.

The main idea of *derivatives* is that if dx is infinitesimally small (but not zero), then $h\{x, dx\}$ is linear, that is, h is linear in $[x - dx, x + dx]$. If h is linear in that interval, then we must have $h(x+dx) = h(x)+cdx$, for some c. In other words, if $dx > 0$ and $c > 0$, we start from $(x, h(x))$ and when we move to the right by dx we go up by cdx, for some c.

It turns out that the slope c of the linear curve is $h'(x)$, also written as $\frac{dh}{dx}(x)$, which is called *the derivative of h at x*. This makes sense; in fact, if we call dh the change in h, for $dx \to 0$, we have:

$$\begin{aligned} h(x + dx) = h(x) + h'(x)dx &\implies \\ h(x + dx) - h(x) = h'(x)dx &\implies \\ dh = h'(x)dx &\implies \\ \frac{dh}{dx} = \frac{dh}{dx}(x) = h'(x). & \end{aligned}$$

Should we write just $\frac{dh}{dx}$ or $\frac{dh}{dx}(x)$ to indicate that the derivative is evaluated at the point x? You'll get an answer in section 5.

Note that by rewriting $h(x + dx) - h(x)$ as dh we lose some information: x and dx. To make things rigorous, we should say that dh is really a function of dx:

$$dh(x; dx) = h'(x)dx.$$

$dh(x; dx)$ is *the differential of h at x* and can be seen as the *best linear approximation* to $h(x + dx) - h(x)$ at the point x. Again, note that $dh(x; dx)$ and $h(x + dx) - h(x)$ are seen as functions of dx and not of x, which can be seen as a fixed parameter in this context.

We may say that $dh(x; dx)$ is that function such that

$$\lim_{dx \to 0} \frac{h(x + dx) - h(x) - dh(x; dx)}{dx} = 0,$$

also written as

$$h(x + dx) - h(x) - dh(x; dx) = o(dx).$$

The derivative of h at x is just $\frac{dh(x;dx)}{dx}$ (note that this is a *real* fraction!), which is the slope of the linear approximation dh. But we are *applied mathematicians* so we just write dh and we don't care about what *pure mathematicians* say.

4 Chain rule

Let's consider $h(x) = g(f(x))$ at the point t. What's the change in h if we move from t to $t+dx$? To answer that, we need to start with f. So what's the change in f if we move from t to $t + dx$? That's

$$df = f'(t)dx. \tag{1}$$

So f changes by df. Now note that if f is at t, then g is at $f(t)$, so what's the change in g from $f(t)$ to $f(t) + df$? That's

$$dg = g'(f(t))df. \tag{2}$$

Therefore, if we change x by dx, f changes by df and, as a consequence, g changes by dg. From equalities 1 and 2, we get

$$\begin{aligned} dg = g'(f(t))df = g'(f(t))f'(t)dx \quad &\Longrightarrow \\ h'(t) = \frac{dg}{dx} = g'(f(t))f'(t). & \end{aligned}$$

That's the *chain rule*. Note that we rewrote $\frac{dg}{dx}$ as $h'(t)$ and not $g'(t)$. To understand why, keep reading.

5 A note about *notation*

In the chain rule we wrote that $h'(t) = \frac{dg}{dx}$. Why not $h'(t) = \frac{dh}{dx}$ or maybe $g'(t) = \frac{dg}{dx}$?

In *real analysis*, one says that $h(x) = g(f(x))$ and that the *derivative* of h at t with respect to x is

$$h'(t) = g'(f(t))f'(t)$$

where I chose to write t instead of x to emphasize that x is the name of the variable whereas t is the point where we calculate the derivative, but usually one just writes

$$h'(x) = g'(f(x))f'(x).$$

On the other hand, *applied* mathematicians, who love their $\frac{df}{dx}$ notation (called *Leibniz notation*) usually give variables and functions the same name. For instance, they write

$$f = f(x) \tag{3}$$
$$g = g(f). \tag{4}$$

The idea is that f is both a *variable* which depends on x and the *function* which expresses the mapping between the variables x and f. Note that the f in equality 4 is the *variable* f. Do you see how expressions 3 and 4 are similar to each other while in the pure math notation they're different because f is a function?

This allows us to write

$$\frac{dg}{dx} = \frac{dg}{df}\frac{df}{dx}$$

where it's ***as if*** the term df canceled out when multiplying two fractions (strong emphasis on ***as if***!).

Some authors even mix the two notations. I'll indicate the points at which the derivatives are evaluated but applied mathematicians usually do not because those points are implicit in the way the variables are defined. If $x = t$, $f = f(x)$ and $g = g(f)$, then it must be the case that, for instance, $\frac{dg}{df}$ is

$$g'(f) = g'(f(x)) = g'(f(t)).$$

I encourage you to become *flexible* and be able to handle any notation you come across. I hope this little aside clarified things instead of making them more confusing.

6 Chain rule in $\mathbb{R}^n$

If we are in $\mathbb{R}^n$, things get more complicated, but not by much. Let's say we have

$$h(x_1, x_2) = g(f_1(x_1, x_2), f_2(x_1, x_2))$$

where h, g, f_1 and f_2 take *two* values and return *one* value, i.e. they are $\mathbb{R}^2 \to \mathbb{R}$ functions. If we define f as an $\mathbb{R}^2 \to \mathbb{R}^2$ function which takes two values x_1, x_2 and returns two values $f_1(x_1, x_2), f_2(x_1, x_2)$, then we can write:

$$h(x_1, x_2) = g(\boldsymbol{f}(x_1, x_2)).$$

As you can see, vectors ($\boldsymbol{f}$ is a *vector function*) are typeset in **lowercase bold**. If we now define $\boldsymbol{x}$ as (x_1, x_2), we can write:

$$h(\boldsymbol{x}) = g(\boldsymbol{f}(\boldsymbol{x})).$$

Now we have *partial derivatives* $\frac{\partial \boldsymbol{f}}{\partial x_1}, \frac{\partial \boldsymbol{f}}{\partial x_2}$, etc., but the idea is the same. We use "∂" for partial derivatives and "d" for derivatives, but they're conceptually equivalent. If we change x_1 then $\boldsymbol{f}$ changes and so g changes as well. Let's say we are at $\boldsymbol{x} = (t, u)$ and we change x_1 and x_2 by ∂x_1 and ∂x_2, respectively. Analogously to before, we get:

$$\partial \boldsymbol{f} = \boldsymbol{f}_{x_1}(t, u)\partial x_1$$

where $\boldsymbol{f}_{x_1}(t, u)$ is the partial derivative at (t, u) of $\boldsymbol{f}$ with respect to x_1.

In general, the partial derivative of a function with respect to a particular variable z is just the derivative of that function with respect to z pretending that the other variables are constant (say some fixed parameters). In other words,

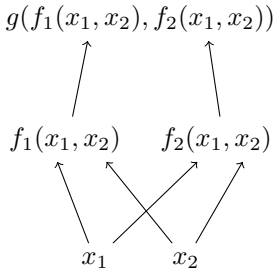
the partial derivative tells us by how much the function changes if we *change one particular variable and keep all the other variables fixed.* For instance,

$$\frac{\partial(5x^2 - xy^2)}{\partial x} = 10x - y^2 \qquad y^2 \text{ is just a constant, like } 5$$
$$\frac{\partial(5x^2 - xy^2)}{\partial y} = -2xy \qquad \text{now } x \text{ is just a constant.}$$

Let's get back to $h(\boldsymbol{x}) = g(\boldsymbol{f}(\boldsymbol{x}))$ and remember that that's equivalent to

$$h(x_1, x_2) = g(f_1(x_1, x_2), f_2(x_1, x_2)).$$

A graph will help us see what changes what:



As we can see, x_1 changes both f_1 and f_2 which both change g. Since the changes are linear, they just add up. Basically, changing g by simultaneously changing f_1 and f_2, is like changing g by first changing f_1 and then changing f_2 (or first f_2 and then f_1). It's like saying that if you are at $(0, 0)$ and you want to reach $(3, 4)$ it doesn't matter if you first go to $(3, 0)$ or $(0, 4)$. The order doesn't matter and, moreover, the total change is just the sum of the individual changes.

Let y_1, y_2 be the variables of g and let's compute $\frac{\partial h}{\partial x_1}(u, t)$, i.e. how much h changes if we change x_1 when we are at (u, t):

$$\begin{aligned}
\partial f_1 &= (f_1)_{x_1}(u, t)\partial x_1 \\
\partial f_2 &= (f_2)_{x_1}(u, t)\partial x_1 \\
\partial h &= g_{y_1}(f_1(u, t), f_2(u, t))\partial f_1 + \\
&\quad g_{y_2}(f_1(u, t), f_2(u, t))\partial f_2.
\end{aligned}$$

As we can see, x_1 modifies f_1 and f_2 which, together, modify g. Always note at which points the derivatives are calculated!

To get $\frac{\partial h}{\partial x_1}$ we must substitute:

$$\begin{aligned}
\partial h &= g_{y_1}(f_1(u,t), f_2(u,t))\partial f_1 + \\
&\quad g_{y_2}(f_1(u,t), f_2(u,t))\partial f_2 \\
&= g_{y_1}(f_1(u,t), f_2(u,t))(f_1)_{x_1}(u,t)\partial x_1 + \\
&\quad g_{y_2}(f_1(u,t), f_2(u,t))(f_2)_{x_1}(u,t)\partial x_1 \\
&= [g_{y_1}(f_1(u,t), f_2(u,t))(f_1)_{x_1}(u,t) + \\
&\quad g_{y_2}(f_1(u,t), f_2(u,t))(f_2)_{x_1}(u,t)]\partial x_1.
\end{aligned}$$

Therefore:

$$\begin{aligned}
h_{x_1}(u,t) = \frac{\partial h}{\partial x_1} &= g_{y_1}(f_1(u,t), f_2(u,t))(f_1)_{x_1}(u,t) + \\
&\quad g_{y_2}(f_1(u,t), f_2(u,t))(f_2)_{x_1}(u,t).
\end{aligned}$$

We can also rewrite this more compactly:

$$\frac{\partial g}{\partial x_1} = \frac{\partial g}{\partial f_1}\frac{\partial f_1}{\partial x_1} + \frac{\partial g}{\partial f_2}\frac{\partial f_2}{\partial x_1},$$

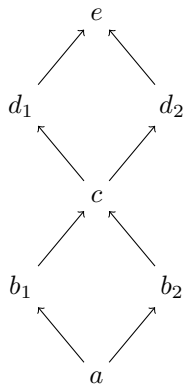
where we assumed the following definitions (see section 5):

$$\begin{aligned}
(x_1, x_2) &= (u, t) \\
f_1 &= f_1(x_1, x_2) \\
f_2 &= f_2(x_1, x_2) \\
g &= g(f_1, f_2).
\end{aligned}$$

As I said before, I'm throwing at you different notations on purpose so that you'll get familiar with them as soon as possible, if you're not already.

7 Why backprop?

Now let's consider the following graph:



We want to compute $\frac{de}{da}$. Note that we don't write $\frac{\partial e}{\partial a}$. That's because e can be seen as a function of the only a, thus we write $\frac{de}{da}$ like we did in the *1D case* (in a way, we *are* in the 1D case). However, note that e is defined as a function which takes two values. It's the composition represented by the entire graph that's a function of the only a.

We can see that there are 4 paths from a to e, so a influences e in 4 ways and we have:

$$\begin{aligned}\frac{de}{da} &= \text{path}[a, b_1, c, d_1, e] + \\ &\quad \text{path}[a, b_1, c, d_2, e] + \\ &\quad \text{path}[a, b_2, c, d_1, e] + \\ &\quad \text{path}[a, b_2, c, d_2, e] \\ &= \frac{db_1}{da}\frac{\partial c}{\partial b_1}\frac{dd_1}{dc}\frac{\partial e}{\partial d_1} + \\ &\quad \frac{db_1}{da}\frac{\partial c}{\partial b_1}\frac{dd_2}{dc}\frac{\partial e}{\partial d_2} + \\ &\quad \frac{db_2}{da}\frac{\partial c}{\partial b_2}\frac{dd_1}{dc}\frac{\partial e}{\partial d_1} + \\ &\quad \frac{db_2}{da}\frac{\partial c}{\partial b_2}\frac{dd_2}{dc}\frac{\partial e}{\partial d_2}.\end{aligned}$$

Note that we sum paths and multiply along the paths. Let's examine one path:

$$\frac{db_1}{da}\frac{\partial c}{\partial b_1}\frac{dd_1}{dc}\frac{\partial e}{\partial d_1}.$$

This means that we change a so we change b_1, so we change c, so we change d_1, and so we change e. Note that the number of paths is *exponential* with respect to the length of the path. Every time we add a bifurcation the total number of paths *doubles*.

Computing the partial changes along the single paths is a waste of time because many computations are repeated. Let's simplify things. Here's the stupid way again:

$$\begin{aligned}\frac{de}{da} &= \text{path}[a, b_1, c, d_1, e] + \\ &\quad \text{path}[a, b_1, c, d_2, e] + \\ &\quad \text{path}[a, b_2, c, d_1, e] + \\ &\quad \text{path}[a, b_2, c, d_2, e].\end{aligned}$$

Note that, in general,

$$\text{path}[a, \ldots, b, \ldots, c] = \text{path}[a, \ldots, b]\text{path}[b, \ldots, c],$$

that is, we can split paths into two or more subpaths. For instance,

$$\begin{aligned}\text{path}[a, b_1, c, d_1, e] &= \frac{db_1}{da}\frac{\partial c}{\partial b_1}\frac{dd_1}{dc}\frac{\partial e}{\partial d_1} \\ &= \left(\frac{db_1}{da}\frac{\partial c}{\partial b_1}\right)\left(\frac{dd_1}{dc}\frac{\partial e}{\partial d_1}\right) \\ &= \text{path}[a, b_1, c]\text{path}[c, d_1, e].\end{aligned}$$

This readily suggests a smarter way to compute $\frac{de}{da}$:

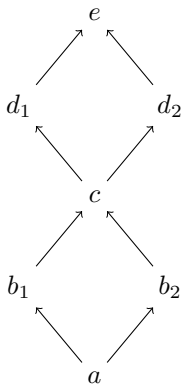
$$\begin{aligned}\frac{de}{da} &= (\text{path}[a, b_1, c] + \text{path}[a, b_2, c])(\text{path}[c, d_1, e] + \text{path}[c, d_2, e]) \\ &= \left(\frac{db_1}{da}\frac{\partial c}{\partial b_1} + \frac{db_2}{da}\frac{\partial c}{\partial b_2}\right)\left(\frac{dd_1}{dc}\frac{\partial e}{\partial d_1} + \frac{dd_2}{dc}\frac{\partial e}{\partial d_2}\right).\end{aligned}$$

Note that this is just

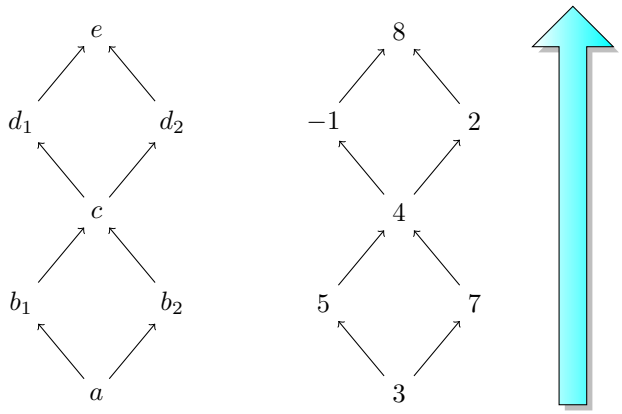
$$\frac{de}{da} = \frac{dc}{da}\frac{de}{dc}.$$

8 Backprop in action

Let's consider the same graph again:



We want to evaluate $\frac{de}{da}$ at $a = 3$. During the forward phase, we compute the values of the variables (defined through functions which we omitted for more clarity):

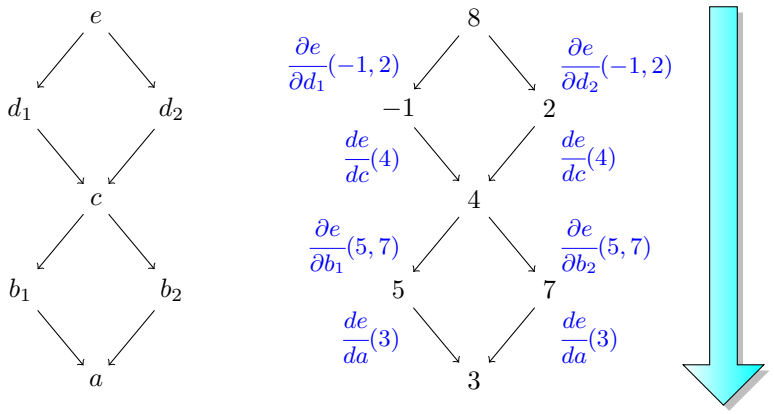


Just to clarify, every variable in the graph depends directly on the variable(s) just below. For instance, c depends on b_1 and b_2, while b_1 depends on a. In other words, there are some functions f and g such that

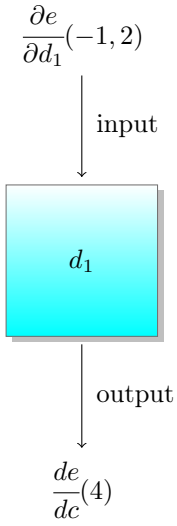
$$\begin{aligned} c &= f(b_1, b_2) \\ b_1 &= g(a). \end{aligned}$$

We want to compute $\frac{de}{da}(3)$ so we let $a = 3$. Now we must compute the values of all the other variables going up. I just put some arbitrary numbers in the graph to make things more concrete.

Now we perform the *backward phase* which is usually called *backprop*, short for *backward propagation*:



Let's examine block d_1 in detail:



During backprop, d_1 receives $\frac{\partial e}{\partial d1}(-1, 2)$ in input and outputs $\frac{de}{dc}(4)$. Here's how d_1 does it:

$$\frac{de}{dc}(4) = \frac{\partial e}{\partial d_1}(-1, 2)\frac{dd_1}{dc}(4).$$

Note: in the expression above we're only considering the $\frac{de}{dc}(4)$ coming from the left path (i.e. $c \leftarrow d_1 \leftarrow e$), but in reality we should sum both the $\frac{de}{dc}(4)$ to get the *"total"* $\frac{de}{dc}(4)$. Unfortunately, I don't know how to make my notation more clear without coming up with some weird convention.

There's an important point to be made. We can write $\frac{\partial e}{\partial d_1}(-1, 2)$ because e can be seen as a function of d_1 and d_2 alone. $\frac{de}{dc}(4)$ is also correct because e can also be seen as a function of c. We can't write $\frac{\partial e}{\partial d_1}(-1)$ because e depends not only on d_1 but also on d_2. I'll explain this better in the next section.

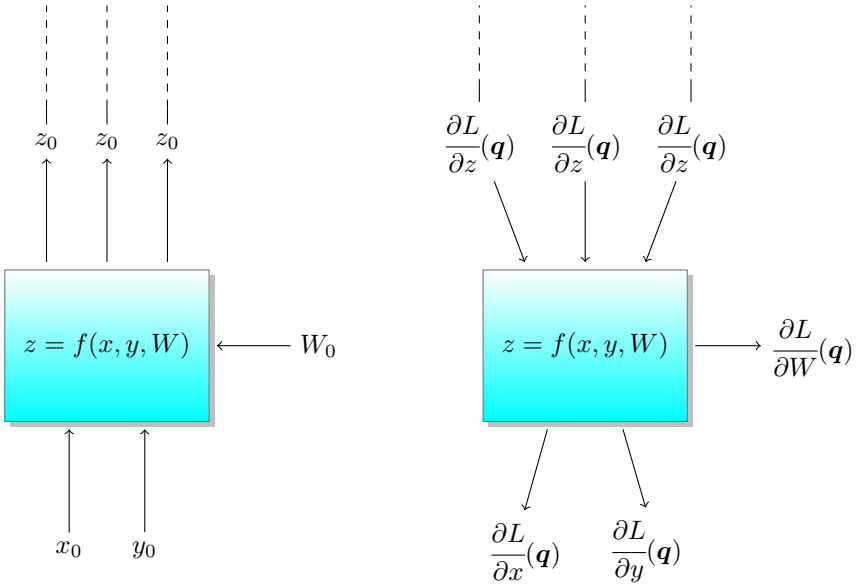
It goes without saying—but I'm saying it anyway—that we're focusing on a single block because once we know how the forward/backward propagation works with respect to a single block, then we know how it works with respect to the entire graph. This is the *modularization* I was talking about in section 2 at the beginning. Libraries such as *Tensorflow*[1] and *Theano*[2] are based on this very modularization so it's important that you understand it very well.

9 Backprop with blocks

Let's consider a more general case, now:

Forward phase

Backward phase



Note:

$$z_0 = f(x_0, y_0, W_0)$$

$$\boldsymbol{q} = \text{all the input sent to } L$$

We are the block z depicted above and we want to compute gradients/derivatives of the loss function L with respect to our inputs x, y and W (they're inputs in the forward phase). In particular, W is our parameter, but we can see it as a normal input. There's nothing special about it, except for the fact that it isn't computed from other values.

Note that the three z_0 on the left are all equal, but the three $\frac{\partial L}{\partial z}(\boldsymbol{q})$ on the right are all different because they come from different paths. In other words, z influences L indirectly by influencing three different blocks which block z is connected to (not shown in the picture).

What's $\boldsymbol{q}$? Why not just z_0? The problem is that L may receive input from other blocks on other paths. The variable $\boldsymbol{q}$ represents all the input received by L. Since z_0 influences L, it's clear that z_0 influences the input $\boldsymbol{q}$, but it may not completely determine it.

Let's say $L = (\ldots)k$, where k is some input. If $k = 0$, then $L = 0$ as well and all the derivatives become 0, including $\frac{\partial L}{\partial x}(\boldsymbol{q})$, $\frac{\partial L}{\partial y}(\boldsymbol{q})$ and $\frac{\partial L}{\partial W}(\boldsymbol{q})$! So, all the input is important because it determines at which point the derivatives are computed.

We receive three instances of $\frac{\partial L}{\partial z}(\boldsymbol{q})$, each of which measures, as you should

know quite well by now, the increment in L when z is incremented from z_0 to $z_0 + \epsilon$ for a little ϵ (the bigger the epsilon, the worse the estimate, unless there is no nonlinearity involved).

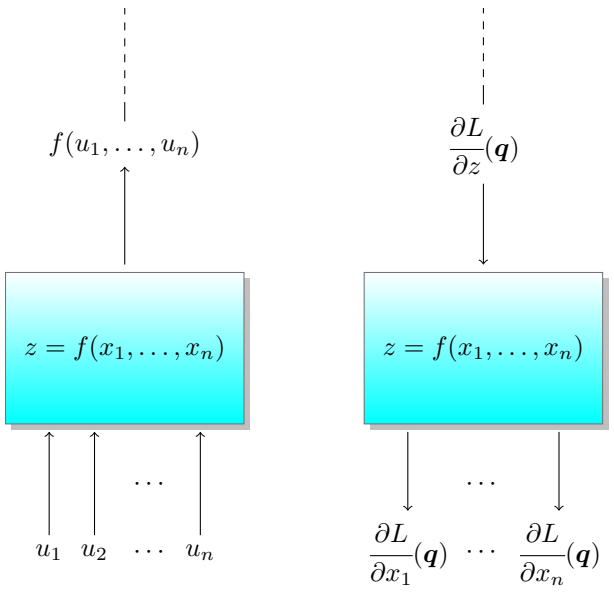
We, the block, know how z is computed from x_0, y_0 and W_0 so we know how to determine how z changes when we move away from $z_0 = (x_0, y_0, W_0)$. Here are the derivations:

$$\begin{aligned}
\frac{\partial L}{\partial z}(\boldsymbol{q}) &\leftarrow \text{sum of the three } \frac{\partial L}{\partial z}(\boldsymbol{q}) \text{ we received in input (from above)} \\
\frac{\partial L}{\partial x}(\boldsymbol{q}) &= \frac{\partial L}{\partial z}(\boldsymbol{q}) \frac{\partial z}{\partial x}(x_0, y_0, W_0) \\
&= \frac{\partial L}{\partial z}(\boldsymbol{q}) f_x(x_0, y_0, W_0) \\
\frac{\partial L}{\partial y}(\boldsymbol{q}) &= \frac{\partial L}{\partial z}(\boldsymbol{q}) \frac{\partial z}{\partial y}(x_0, y_0, W_0) \\
&= \frac{\partial L}{\partial z}(\boldsymbol{q}) f_y(x_0, y_0, W_0) \\
\frac{\partial L}{\partial W}(\boldsymbol{q}) &= \frac{\partial L}{\partial z}(\boldsymbol{q}) \frac{\partial z}{\partial W}(x_0, y_0, W_0) \\
&= \frac{\partial L}{\partial z}(\boldsymbol{q}) f_W(x_0, y_0, W_0).
\end{aligned}$$

Note that while $\frac{\partial L}{\partial x}$ depends on $\boldsymbol{q}$ (all the input to L), $\frac{\partial z}{\partial x}$ depends on x_0, y_0 and W_0, i.e. all the input to z. Again—I'll never grow tired of saying it—$\frac{\partial z}{\partial x}$ depends on all the inputs x_0, y_0, and W_0 because we need to compute the derivative with respect to x at the point (x_0, y_0, W_0). It's the same old story: we need to consider all the input even if we're deriving just with respect to a part of it.

So, the input from below tells us where we are (it was computed during the forward phase) and we compute the partial derivatives of f at *that* point with respect to the inputs. Once we know $\frac{\partial L}{\partial z}$ and $\frac{\partial z}{\partial x}$ (or y, W) we can compute $\frac{\partial L}{\partial x}$ by multiplying them (by the way, note that, as we said before, it's as if ∂z canceled out).

10 Generalization I



Note:

$\boldsymbol{q}$ = all the input sent to L

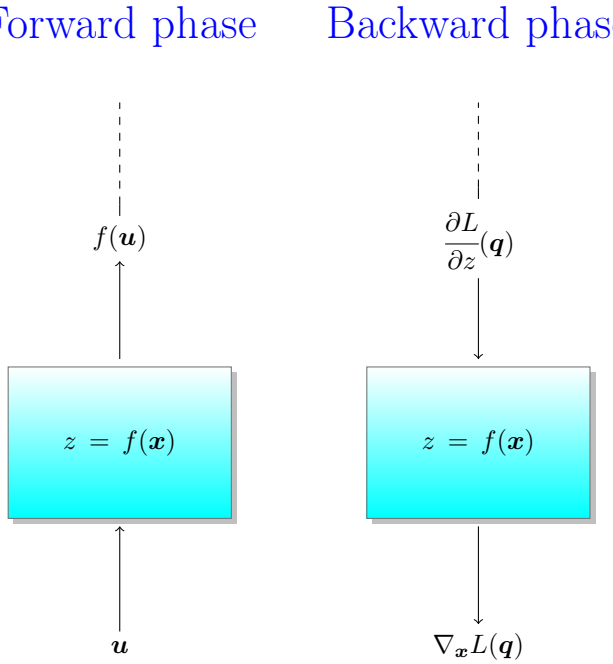
One $\frac{\partial L}{\partial z}$ is enough because we saw that if there are more than one we can just add them up.

The derivations are:

$$\begin{aligned}\frac{\partial L}{\partial x_i}(\boldsymbol{q}) &= \frac{\partial L}{\partial z}(\boldsymbol{q})\frac{\partial z}{\partial x_i}(u_1, ..., u_n) \\ &= \frac{\partial L}{\partial z}(\boldsymbol{q}) f_{x_i}(u_1, ..., u_n).\end{aligned}$$

11 Generalization I (vector form)

This is equivalent to the previous case but lists of scalars have been replaced with vectors (which, as before, are typeset in **lowercase bold**).



Note:

$\boldsymbol{q}$ = all the input sent to L

The derivations are:

$$\frac{\partial L}{\partial x_i}(\boldsymbol{q}) = \frac{\partial L}{\partial z}(\boldsymbol{q})\frac{\partial z}{\partial x_i}(\boldsymbol{u}) = \frac{\partial L}{\partial z}(\boldsymbol{q}) f_{x_i}(\boldsymbol{u}).$$

The gradient of L at $\boldsymbol{q}$ with respect to the vector $\boldsymbol{x}$ is defined as

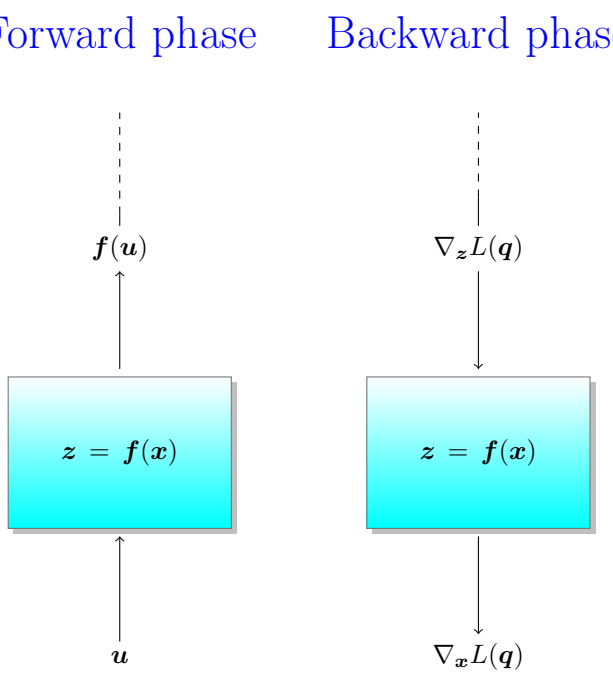
$$\nabla_{\boldsymbol{x}} L(\boldsymbol{q}) = \begin{bmatrix} \frac{\partial L}{\partial x_1}(\boldsymbol{q}) \\ \vdots \\ \frac{\partial L}{\partial x_n}(\boldsymbol{q}) \end{bmatrix}.$$

The derivation can thus be rewritten as

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{q}) = \frac{\partial L}{\partial z}(\boldsymbol{q}) \nabla_{\boldsymbol{x}} z(\boldsymbol{u}) \qquad \text{(scalar times a column vector)}$$

12 Generalization II

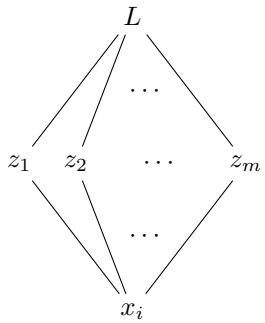
Now $\boldsymbol{z}$ is a vector as well, i.e. $\boldsymbol{f}$ is an $\mathbb{R}^n \to \mathbb{R}^m$ function, or an m-dimensional vector of $\mathbb{R}^n \to \mathbb{R}$ functions.



Note:
$\boldsymbol{q}$ = all the input sent to L

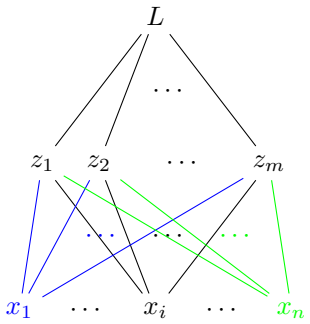
You should be pretty comfortable with this by now, but let's repeat what it means. Modifying x_i may modify every single z_j because $\boldsymbol{f}$ may use every single x_i to compute every single z_j. Then every z_j may modify L.

We can represent this with a graph:



This graph has m paths

Note that this is just a partial graph because it's enough to focus on a single x_i. The complete graph would be like this:



This graph has nm paths

Now we can write the expression for $\frac{\partial L}{\partial x_i}(\boldsymbol{q})$:

$$\begin{aligned}\frac{\partial L}{\partial x_i}(\boldsymbol{q}) &= \sum_{j=1}^{m} \text{path}[x_i, z_j, L] \\ &= \sum_{j=1}^{m} \frac{\partial z_j}{\partial x_i}(\boldsymbol{u}) \frac{\partial L}{\partial z_j}(\boldsymbol{q}) \\ &= \left[\frac{\partial \boldsymbol{z}}{\partial x_i}(\boldsymbol{u})\right]^T \nabla_{\boldsymbol{z}} L(\boldsymbol{q}).\end{aligned}$$

The term $\frac{\partial \boldsymbol{z}}{\partial x_i}(u)$ is a *Jacobian* and is defined like this:

$$\frac{\partial \boldsymbol{z}}{\partial x_i}(\boldsymbol{u}) = \begin{bmatrix} \frac{\partial z_1}{\partial x_i}(\boldsymbol{u}) \\ \vdots \\ \frac{\partial z_m}{\partial x_i}(\boldsymbol{u}) \end{bmatrix}.$$

The Jacobian is a generalization of the gradient and it's, in general, the derivative of an $\mathbb{R}^n \to \mathbb{R}^m$ function. If a function $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable at $\boldsymbol{u}$, then $\boldsymbol{f}$ can be *locally* approximated by a *linear* function $\mathbb{R}^n \to \mathbb{R}^m$ expressed by the Jacobian:

$$\boldsymbol{f}(\boldsymbol{u} + d\boldsymbol{u}) \approx \boldsymbol{f}(\boldsymbol{u}) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u}) d\boldsymbol{u}.$$

If $\boldsymbol{f}$ is linear, we get an equality, because the best linear approximation to a linear function is the linear function itself. If f is $\mathbb{R} \to \mathbb{R}$, this becomes

$$f(u + du) \approx f(u) + f'(u) du.$$

We haven't properly defined the (general) Jacobian yet. Let $\boldsymbol{f}(\boldsymbol{x})$ be an $\mathbb{R}^n \to \mathbb{R}^m$ differentiable (at least at $\boldsymbol{u}$) function. The Jacobian of $\boldsymbol{f}$ at $\boldsymbol{u}$ with respect to $\boldsymbol{x}$ is the $m \times n$ *matrix* $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u})$ defined as

$$\left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]_{i,j} = \frac{\partial f_i}{x_j}(\boldsymbol{u}),$$

or, more explicitly,

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u}) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(\boldsymbol{u}) & \frac{\partial f_1}{\partial x_2}(\boldsymbol{u}) & \cdots & \frac{\partial f_1}{\partial x_n}(\boldsymbol{u}) \\ \frac{\partial f_2}{\partial x_1}(\boldsymbol{u}) & \frac{\partial f_2}{\partial x_2}(\boldsymbol{u}) & \cdots & \frac{\partial f_2}{\partial x_n}(\boldsymbol{u}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1}(\boldsymbol{u}) & \frac{\partial f_m}{\partial x_2}(\boldsymbol{u}) & \cdots & \frac{\partial f_m}{\partial x_n}(\boldsymbol{u}) \end{bmatrix}.$$

As we said before, $\boldsymbol{f}$ can be seen as a vector of $\mathbb{R}^n \to \mathbb{R}$ functions each of which takes $\boldsymbol{x}$ and returns a single coordinate of $\boldsymbol{z} = \boldsymbol{f}(\boldsymbol{x})$. Therefore, $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u})$ is a matrix whose i-th row is the *transpose* of the gradient of f_i at $\boldsymbol{u}$ with respect to $\boldsymbol{x}$:

$$\left[\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]_{i,\cdot} = [\nabla_{\boldsymbol{x}} f_i(\boldsymbol{u})]^T \qquad \text{(row vector)}$$

To remember the definition of the Jacobian, note that with matrices the order is always *rows→columns*:

1. if $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ then $\boldsymbol{A}$ has m rows and n columns;
2. $A_{i,j}$ is the element on the i-th row and j-th column;
3. $\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}$ is the matrix where $\boldsymbol{z}$ changes vertically across rows and $\boldsymbol{x}$ changes horizontally across columns.

The gradient, when it exists, is the transpose of the Jacobian. In fact, if $f(\boldsymbol{x})$ is $\mathbb{R}^n \to \mathbb{R}$ then

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{u}) = \left[\frac{\partial f}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]^T \qquad \text{(column vector)}$$

Let's get back to our blocks now. We derived the following:

$$\begin{aligned} \frac{\partial L}{\partial x_i}(\boldsymbol{q}) &= \sum_{j=1}^{m} \frac{\partial z_j}{\partial x_i}(\boldsymbol{u}) \frac{\partial L}{\partial z_j}(\boldsymbol{q}) \\ &= \left[\frac{\partial \boldsymbol{z}}{\partial x_i}(\boldsymbol{u})\right]^T \nabla_{\boldsymbol{z}} L(\boldsymbol{q}). \end{aligned}$$

From this we get

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{q}) = \left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]^T \nabla_{\boldsymbol{z}} L(\boldsymbol{q}). \tag{5}$$

To remember equality 5, note that $\boldsymbol{z}$ changes *horizontally* in $\left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]^T$ while it changes *vertically* in $\nabla_{\boldsymbol{z}} L(\boldsymbol{q})$, so, since matrix multiplication is *row by column*, this is what we want.

This formula works even when we're dealing with general *tensors* $\mathbf{X}$, $\mathbf{U}$, and $\mathbf{Z}$. The trick is to *vectorize* the tensors. For instance, consider the following 3-dimensional tensor:

$$\boldsymbol{X}_{1,\cdot,\cdot} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
$$\boldsymbol{X}_{2,\cdot,\cdot} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}.$$

We can vectorize $\mathbf{X}$ as follows:

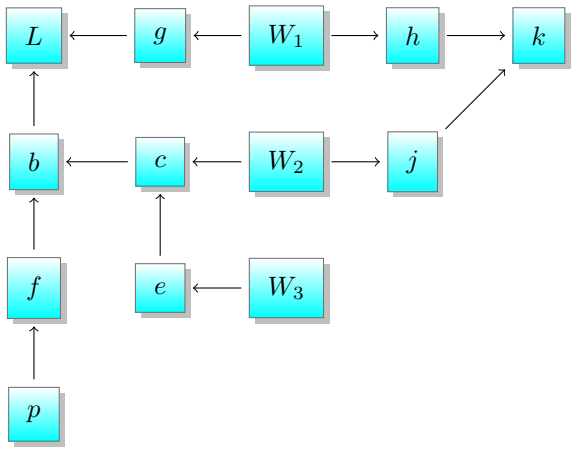
$$\text{vec}(\mathbf{X}) = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & a & b & c & d & e & f & g & h & i \end{bmatrix}$$

and so, for instance, $\text{vec}(\mathbf{X})_{14} = X_{2,2,2} = e$. Of course, all the tensors must be vectorized consistently or we'll get wrong results.

13 Dynamic Programming

Although the algorithm is called backprop, which suggests that we retrace our steps, we can also use *dynamic programming*. That is, we can compute the derivatives recursively in a *lazy* way (i.e. only when needed) and save the already computed derivatives in a table lest we repeat computations.

For instance, consider this graph:



We only want to compute $\frac{\partial L}{\partial W_1}$, $\frac{\partial L}{\partial W_2}$ and $\frac{\partial L}{\partial W_3}$. I'll write the steps performed by a dynamic programming algorithm which computes the 3 derivatives. I'll

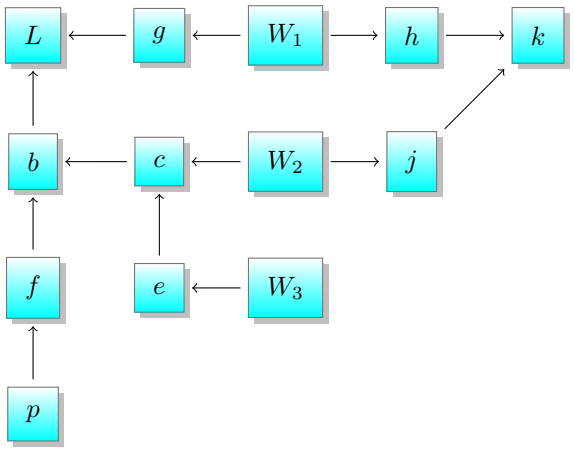
use the following format:

```
operation 1              (op 1 calls recursively op 1a and op 1b)
  operation 1a
  operation 1b           (op 1b calls recursively op 1b1 and op 1b2)
    operation 1b1
    operation 1b2
operation 2
```

Here's the graph again (for your convenience) and the steps, assuming that the "forward" phase has already taken place:

In the code on the right:

- "$A \to B$" means "compute A and store it in B";
- "$A \leftarrow B$" means "read A from B".



$\frac{\partial L}{\partial W_1} \to \text{table}[W_1]$
$\quad \frac{\partial g}{\partial W_1}$
$\quad \frac{\partial L}{\partial g} \to \text{table}[g]$
$\frac{\partial L}{\partial W_2} \to \text{table}[W_2]$
$\quad \frac{\partial c}{\partial W_2}$
$\quad \frac{\partial L}{\partial c} \to \text{table}[c]$
$\quad\quad \frac{\partial b}{\partial c}$
$\quad\quad \frac{\partial L}{\partial b} \to \text{table}[b]$
$\frac{\partial L}{\partial W_3} \to \text{table}[W_3]$
$\quad \frac{\partial e}{\partial W_3}$
$\quad \frac{\partial L}{\partial e}$
$\quad\quad \frac{\partial c}{\partial e}$
$\quad\quad \frac{\partial L}{\partial c} \leftarrow \text{table}[c]$

Note that we don't visit every node of the graph and that we don't recompute $\frac{\partial L}{\partial c}$ which is needed for both $\frac{\partial L}{\partial W_2}$ and $\frac{\partial L}{\partial W_3}$.

14 Efficiency

Backprop is not optimum. In fact, computing derivatives over a graph is *NP-complete* because expressions can be simplified in non-obvious ways. For instance, $s'(x) = s(x)(1 - s(x))$, where s is the *sigmoid* function. Since $s(x) = 1/(1 + \exp(-x))$, an algorithm might waste time computing and composing derivatives without coming up with the simplified expression I wrote above. This argument is only valid if the graph is analyzed once and then used many times to compute the derivatives.

There's another thing to be said about the efficiency of backprop or its dynamic programming variant described above. We saw that in general each

block of the graph performs the following computation:

$$\nabla_{\boldsymbol{x}} L(\boldsymbol{q}) = \left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}(\boldsymbol{u})\right]^T \nabla_{\boldsymbol{z}} L(\boldsymbol{q}).$$

This is a *matrix-vector* multiplication which returns another vector. So, in general, along a path $x \to a \to b \to \cdots \to y \to z \to L$ we have something like

$$\nabla_{\boldsymbol{x}} L = \left[\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right]^T \left[\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right]^T \cdots \left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}\right]^T \nabla_{\boldsymbol{z}} L. \tag{6}$$

Backprop computes this product from right to left (*foldr*):

$$\nabla_{\boldsymbol{x}} L = \left(\left[\frac{\partial \boldsymbol{a}}{\partial \boldsymbol{x}}\right]^T \left(\left[\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right]^T \cdots \left(\left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}\right]^T \nabla_{\boldsymbol{z}} L\right) \cdots\right)\right).$$

If D is the maximum number of dimensions of the vectors involved and N is the number of *matrix-vector* multiplications, the whole product takes $O(ND^2)$ time. Computing the same product from left to right (*foldl*) would take $O(ND^3)$ time because it would involve *matrix-matrix* multiplications. So it seems that backprop does the right thing.

But what happens if x is just a scalar and $\boldsymbol{L}$ a vector? The situation is reversed! Now we have a (row) vector on the left and all matrices on the right:

$$\left[\frac{\partial \boldsymbol{L}}{\partial x}\right]^T = \left[\frac{\partial \boldsymbol{a}}{\partial x}\right]^T \left[\frac{\partial \boldsymbol{b}}{\partial \boldsymbol{a}}\right]^T \cdots \left[\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{y}}\right]^T \left[\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{z}}\right]^T \tag{7}$$

Note that to derive equality 7 you just need to rewrite the two gradients in equality 6 as Jacobians (remembering that the gradient and the Jacobian, when equivalent, are one the *transpose* of the other) and the formula will hold even when $\boldsymbol{L}$ is a vector and x a scalar.

That's it. I hope you found this tutorial useful. Let me know if you find any mistakes or something is unclear.

References

[1] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[2] Rami Al-Rfou et al. "Theano: A Python framework for fast computation of mathematical expressions". In: *arXiv e-prints* abs/1605.02688 (May 2016). URL: `http://arxiv.org/abs/1605.02688`.