How to derive a custom layer in a neural network

Cristi Vicas

Jul 2022

I bet I am not the only one who can grasp the math behind the NNs but as soon as I close the explanations, everything gets wiped out! So I tried to gather explanations in one place. Traceable, just in case I screwed something. This document is for future me, when I will be hacking again at this low level. But maybe will help others, too. So, enjoy!

1 Chain rule in math

Let $F(x) = (f \circ g)(x) = f(g(x))$, where $\circ$ is the composition operator. That is, when we see $(f \circ g)(x)$ it means that the input will be processed by last function $g(x)$ then, the result, by the second to last function in the composition $f(g(x))$ so on, for all the functions in the composition.
To compute the derivative of $F(x)$ math comes in our help with the chain rule:

$$F'(x) = f'(g(x))g'(x) \tag{1}$$

Above expression is what I learned during high school. But in ML we see other notation. If we write $y = f(u)$ and $u = g(x)$, that is, we "blow" the composition rule from left to right and name each component, we can write:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} \tag{2}$$

Look at that u, is very important. This substitution is ubiquitos in chain rule.

Rule of thumb? Add dummy variables for each internal function inside the composition. WARNING! when the pesky d comes to play, these du terms can't be readily simplified! Unless your math is really strong!

1.1 Example

Compute

$$f(x) = \underbrace{ln}_{\substack{\text{That's}\\ \text{our } f}}(\underbrace{x^{-4} + x^4}_{\substack{\text{That's}\\ \text{our } g}}) \tag{3}$$

First, we "blow out" the chain:

$$y = f(u) = ln(u) \tag{4}$$

$$u = g(x) = x^{-4} + x^4 \tag{5}$$

$$\frac{dy}{dx} = \underbrace{\frac{dln(u)}{du}}_{\substack{\text{Dummy} \\ \text{variable } u, \\ \text{only here}}} \underbrace{\frac{d(x^{-4} + x^4)}{dx}}_{\text{No } u \text{ here}} \tag{6}$$

Then, we solve each derivative, independently. First one is a logarithm in u, the other is a polynomial in x.

$$\frac{dy}{dx} = \frac{1}{u}(-4x^{-5} + 4x^3) \tag{7}$$

We know from our "dummy" replacement that $u = x^{-4} + x^4$. Replace and simplify (if you can):

$$\frac{dy}{dx} = \frac{1}{x^{-4} + x^4}(-4x^{-5} + 4x^3) \tag{8}$$

$$\frac{dy}{dx} = \frac{-4x^{-5} + 4x^3}{x^{-4} + x^4} \tag{9}$$

Basically, replace the u after computing the f' with the content of u, that is, $(g(x))$.

Source: `https://tutorial.math.lamar.edu/classes/calci/chainrule.aspx`

1.2 Multivariate functions

Let $z = f(x, y)$ and $x = g(t)$ and $y = h(t)$. Let us compute $\frac{dz}{dt}$

$$\frac{dz}{dt} = \frac{\delta f}{\delta x}\frac{dx}{dt} + \frac{\delta f}{\delta y}\frac{dy}{dt} \tag{10}$$

The $\frac{\delta f}{\delta x}$ means that we treat $f(x, y)$ as if y is constant and x is the only variable. Then, we derive.
Source: `https://tutorial.math.lamar.edu/classes/calciii/chainrule.aspx`

For now, that's all the math.

2 Extending PyTorch

The goal here is to add a new function to pytorch. A new type of layer. A new way of processing the input.

Source: `https://pytorch.org/docs/stable/notes/extending.html`

When extending pytorch one have to define a new Function. This Function will have a forward method, that will be applied to some inputs and generate the forward output. At a later stage, the backward method will be called with the gradients wrt to the output. The backward method have to return the gradients wrt to its inputs.
Take care, as input is both the actual data and the weights/parameters of the function. So the gradient wrt to parameters will be applied to update the parameters (weights) and the gradients wrt to the "regular" inputs are forwarded to the next backprop step.
We will try to reproduce a linear layer, no weights, with sigmoid activation, as a new function. And compare somehow, forward/backward steps, with a regular, sigmoid activated linear layer.

"backward() - gradient formula. It will be given as many Tensor arguments as there were outputs, with each of them representing gradient w.r.t. that output. It should return as many Tensor s as there were inputs, with each of them containing the gradient w.r.t. its corresponding input. "

2.1 One layer net

Hmm, let's see if I got all the intuitions right. Basic example, one neuron, no activation and some loss.
Notations:

- Input x
- Outgoing output (forward step, to the layer above) $a = x$. Later a will be different
- Weights (or one weight if there is one input): w
- The weigthed input: $z = wa$
- Loss l, we don't care
- Cost $C = l(z)$, Usually cost is a function of output, desired output and loss. Loss is a function, desired output do not change, they will not affect our incursions in the derivative world.

Pytorch wants from us the gradient of cost wrt to our inputs and to our weights. That is:

$$\frac{\delta C}{\delta w} \tag{11}$$

$$\frac{\delta C}{\delta x} \tag{12}$$

And pytorch framework gives us

$$\frac{\delta C}{\delta z} \tag{13}$$

With our setup done, let's see how can we give PyTorch what it wants. We will start from the expression of the cost C, we will derive it and see how things fit. Soo, the final cost is:

$$C = l(z(x)) \tag{14}$$
$$C = l(wx) \tag{15}$$

Let's derivate C with respect to w. We keep in mind the chain rule (2) and the expression of $C = l(z)$:

$$\frac{\delta C}{\delta w} = \frac{\delta l(z)}{\delta z}\frac{\delta z}{\delta w} \tag{16}$$

We don't need dummy u, we already defined the z. Moving on with 2nd term, $z = wx$:

$$\frac{\delta C}{\delta w} = \frac{\delta l(z)}{\delta z}\frac{\delta (wx)}{\delta w} \tag{17}$$

Also, we know the 1st term? $l(z)$ is C. Hmmmm:

$$\frac{\delta C}{\delta w} = \underbrace{\frac{\delta C}{\delta z}}_{\substack{\text{How nice, we have it from pytorch!}}}\underbrace{\frac{\delta (wx)}{\delta w}}_{\substack{\text{This is simple}}} \tag{18}$$

Some sort of empirical rule arise! When we want other derivatives, below our current level, we express them in terms to the known output! Let's see a more complex example.

2.2 A layer with activation

Let's do some notations. For one neuron, in layer l:

- Incoming inputs (from layer below) a^{l-1}
- Outgoing output (forward step, to the layer above) a^l
- Weights (or one weight if there is one input): w^l
- The weigthed input: $z^l = w^l a^{l-1}$
- The activated output, a^l=$\sigma(z^l)$ with a sigmoid.

Also, the cost of the entire training step is C. I googled that

$$\sigma'(x)=\sigma(x)\,(1-\sigma(x)) \tag{19}$$

Note that the activation is nothing fancy, "classical" sigmoid.

Following the explanations on the PyTorch manual, for tha backward part of our Function object, we receive the cost gradient wrt to the outputs of the layer, that is:

$$\frac{\delta C}{\delta a^l} \tag{20}$$

We must compute the derivatives needed by the backwards functions. Focus on C wrt to w:

$$\frac{\delta C}{\delta w^l} = \underbrace{\frac{\delta C}{\delta a^l}}_{\text{This is what we know from pytorch}} \underbrace{\frac{\delta a^l}{\delta w^l}}_{\text{Because chain rule!}} \tag{21}$$

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l} \underbrace{\frac{\delta \sigma(z^l)}{\delta w^l}}_{\text{We replaced the value of } a^l} \tag{22}$$

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l} \underbrace{\frac{\delta \sigma(z^l)}{\delta z^l}\frac{\delta z^l}{\delta w^l}}_{\text{Exploded using chain rule!}} \tag{23}$$

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l} \underbrace{\frac{\delta \sigma(z^l)}{\delta z^l}}_{\text{Term only in } z^l. \text{ From highschool, this is } \sigma'(z^l)} \underbrace{\frac{\delta z^l}{\delta w^l}}_{\text{We'll go on exploding } z^l} \tag{24}$$

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l}\sigma'(z^l) \underbrace{\frac{\delta w^l a^{l-1}}{\delta w^l}}_{\text{This is simple}} \tag{25}$$

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l}\sigma'(z^l)a^{l-1} \tag{26}$$

Checking various tutorials and official documentation, we see that the intuitions were correct!

$$\frac{\delta C}{\delta w^l} = \frac{\delta C}{\delta a^l}\sigma'(z^l)a^{l-1} \tag{27}$$

$$\frac{\delta C}{\delta a^{l-1}} = \frac{\delta C}{\delta a^l}\sigma'(z^l)w^l \tag{28}$$

We have the derivative of the cost wrt to our weights (this will be used by the optimizer to "teach" our layer's weights w) and the derivative of the cost wrt to our input layers (this will be the input to the backwards function of the layer below)
So, we really don't care what is from the first function in the composition chain (usually the loss) up to our first function in our layer. Everything is nicely wrapped in the derivative of the cost wrt to our output. We need to mambo-jambo the math to return the derivative of the cost wrt to our input (to keep the chain going) and to our learnable parameters (to be able to descend on the Cost gradient).

2.3 For multi neuron layer

Ok, single neuron, conquered. For more neurons, we have to sum along various axis. Thoroughly deductions below, so I don't screw some sumations at the implementation time.

From `http://neuralnetworksanddeeplearning.com/chap2.html#warm_up_a_fast_matrix-based_approach_to_computing_the_output_from_a_neural_network`

Let:

- n^{l-1} be the number of inputs in layer l
- n^l be the number of outputs of the layer l
- w^l of size (n^l, n^{l-1}) be the weight matrix of the layer l
- $\frac{\delta C}{\delta a^l}$ that is, the derivative of the cost function wrt to the output of the current layer, is known and "deduced" by the backpropagation algorithm before entering the backward() part of the layer's code

Let w^l_{jk} be the weight inside layer l that connects the k'th neuron in layer $(l-1)$ to the j'th neuron in layer l. Let z^l_j be the weighted output of the neuron j in layer l.
Then,

$$z^l_j = \sum_k \left(w^l_{jk} a^{l-1}_k\right) \tag{29}$$

w^l is a matrix of shape $n^l \times n^{l-1}$ so the product between w^l and a^{l-1} is $(n^l, n^{l-1}) \times (n^{l-1}, 1) = (n^l, 1)$

Knowing that $a^l{=}\sigma(z^l)$ and knowing from the gradient descent loop the value for $\frac{\delta C}{\delta a^l}$ we can compute:

$$\frac{\delta C}{\delta w_{jk}^l} = \frac{\delta C}{\delta a_j^l}\frac{\delta a_j^l}{\delta w_{jk}^l} \tag{30}$$

$$\text{We know } a_j^l = \sigma(z_j^l) \tag{31}$$

$$\frac{\delta C}{\delta w_{jk}^l} = \frac{\delta C}{\delta a_j^l}\frac{\delta a_j^l}{\delta z_j^l}\frac{\delta z_j^l}{\delta w_{jk}^l} \tag{32}$$

$$= \frac{\delta C}{\delta a_j^l}\sigma'(z_j^l)\frac{\delta z_j^l}{\delta w_{jk}^l} \tag{33}$$

$$\text{We know } z_j^l = \sum_k \left(w_{jk}^l a_k^{l-1}\right) \tag{34}$$

$$= \frac{\delta C}{\delta a_j^l}\sigma'(z_j^l)\frac{\delta}{\delta w_{jk}^l}\sum_i \left(w_{ji}^l a_k^{l-1}\right) \tag{35}$$

$$= \frac{\delta C}{\delta a_j^l}\sigma'(z_j^l)\left(a_k^{l-1}\right) \tag{36}$$

$$\tag{37}$$

At equation (35), the derivative exists only if k == i.
Let the shape for $\frac{\delta C}{\delta a^l}$ be (n^l) and the shape of $\sigma'(z_j^l)$ the same, (n^l). Then, the

$$\frac{\delta C}{\delta w^l} = \left(\frac{\delta C}{\delta a^l}\cdot\sigma'(z_j^l)\right)^T * a^{l-1} \tag{38}$$

where $\cdot$ is elementwise multiplication and $*$ is matrix multiplication.
In the same manner,

$$\frac{\delta C}{\delta a_k^{l-1}} = \frac{\delta C}{\delta a^l}\frac{\delta a^l}{\delta a^{l-1}} \tag{39}$$

$$a_j^l = \sigma(z_j^l) \tag{40}$$

$$z_j^l = \sum_k \left(w_{jk}^l a_k^{l-1}\right) \tag{41}$$

$$... = ... \tag{42}$$

$$\frac{\delta C}{\delta a^{l-1}} = \left(\frac{\delta C}{\delta a^l}\cdot\sigma'(z_j^l)\right) * w^l \tag{43}$$

that is, $\left(\frac{\delta C}{\delta a^l}\cdot\sigma'(z_j^l)\right)$ have the size $(1, n^{l-1})$ and the w have the size (n^l, n^{l-1})

3 Custom layer with Gauss Kernel

I want to filter my data with a Gauss kernel. The mean μ and standard deviation σ of the filter should be learned from the data. So we must have the derivatives for them!
Oh, cherry on top, because reasons, the mean is not zero. The convolution filter will be asymetric.

3.1 Gauss Filter

A Gauss kernel is:

$$g(m) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\frac{(m-\mu)^2}{\sigma^2}} \tag{44}$$

The derivative with respect to σ is:

$$\begin{aligned}
\frac{dg}{d\sigma} &= \frac{1}{\sqrt{2\pi}} \underbrace{\frac{d}{d\sigma}\left(\frac{1}{\sigma} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2}\right)}_{\text{We apply product rule here}} && (45)\\
&= \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma} \frac{d}{d\sigma} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} + \frac{1}{\sqrt{2\pi}} \left(\frac{d}{d\sigma}\frac{1}{\sigma}\right) e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (46)\\
&= \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \frac{d}{d\sigma}\left(-1/2\left(\frac{m-\mu}{\sigma}\right)^2\right) \\
&\quad + \frac{1}{\sqrt{2\pi}} \frac{-1}{\sigma^2} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (47)\\
&= \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \left(\frac{(m-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right) && (48)
\end{aligned}$$

The derivative wrt to μ is:

$$\begin{aligned}
\frac{dg}{d\mu} &= \frac{1}{\sigma\sqrt{2\pi}} \frac{d}{d\mu}\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right) e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (49)\\
&= \frac{1}{\sigma\sqrt{2\pi}} \frac{1}{2\sigma^2} 2(m-\mu) e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (50)\\
&= \frac{m-\mu}{\sigma^3\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (51)
\end{aligned}$$

and in a simila manner, wrt to m because the chain rule must go on:

$$\frac{dg}{dm} = \frac{1}{\sigma\sqrt{2\pi}}\frac{d}{dm}\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (52)$$

$$= \frac{\mu-m}{\sigma^3\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (53)$$

3.2 Unscaled Gaussian derivatives

One can compute the unscaled versions, that is:

$$g(m) = e^{-\frac{1}{2}\frac{(m-\mu)^2}{\sigma^2}} \quad (54)$$

The unscaled derivative with respect to σ is:

$$\frac{dg}{d\sigma} = \frac{d}{d\sigma}\left(e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2}\right) \quad (55)$$

$$= \frac{d}{d\sigma}e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (56)$$

$$= e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2}\frac{d}{d\sigma}\left(-1/2\left(\frac{m-\mu}{\sigma}\right)^2\right) \quad (57)$$

$$= -\frac{1}{2}\frac{d}{d\sigma}\left((m-\mu)^2\sigma^{-2}\right)e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (58)$$

$$= \frac{(m-\mu)^2}{\sigma^3}e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (59)$$

The unscaled derivative wrt to μ is:

$$\frac{dg}{d\mu} = \frac{d}{d\mu}\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (60)$$

$$= \frac{1}{2\sigma^2}2(m-\mu)e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (61)$$

$$= \frac{m-\mu}{\sigma^2}e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (62)$$

and in a simila manner, the unscaled version wrt to m:

$$\frac{dg}{dm} = \frac{d}{dm}\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (63)$$

$$= \frac{\mu-m}{\sigma^2}e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \quad (64)$$

3.3 Deriving the layer's equations

The layer have as input, a vector X of length N, and two parameters, σ, μ. The output is a vector O of length N. The output is a convolution of the input X with a kernel g defined above.

We also know (by the def of the layer) that:

$$O(n) = \sum_{m=-M/2}^{M/2} X(n-m)g(m) \tag{65}$$

The whole network have a cost C. The gradient for this cost, is backpropagated and we know, as input to the backwards step, the $\frac{\delta C}{\delta O}$, that is the gradient of the cost wrt to each of the layer's outputs.
As in (30) we need to determine eg $\frac{dC}{d\mu}$ by knowing $\frac{dC}{dO}$. So, here, we need to determine $\frac{dO}{d\mu}$ so the chain rule can be applied. Let's see how the summation unrolls. We know that μ is one scalar so $\frac{dC}{d\mu}$ must be also a scalar.

$$\frac{dC}{d\mu} = \sum_n \frac{\delta C}{\delta O(n)} \frac{\delta O(n)}{\delta \mu} \tag{66}$$

The layer derivative equations:

$$\begin{aligned}
\frac{\delta O(n)}{\delta \mu} &= \frac{\delta}{\delta \mu} \sum_m X(n-m)g(m) && (67)\\
&= \sum_m \frac{\delta}{\delta \mu} X(n-m)g(m) && (68)\\
&= \sum_m X(n-m) \frac{\delta}{\delta \mu} g(m) && (69)\\
&= \sum_m X(n-m) \frac{m-\mu}{\sigma^3 \sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} && (70)
\end{aligned}$$

We take the (70) and do the elementwise product as in (66) to get to $\frac{dC}{d\mu}$. So we will have a convolution of the original signal with a kernel (as shown in (70)) and then the result, elementwise multiplied with the output gradient. The product is summed up and this is the gradient wrt to the μ.

In a similar fashion:

$$\frac{\delta O(n)}{\delta \sigma} = \sum_m \frac{\delta}{\delta \mu} X(n-m)g(m) \tag{71}$$

$$= \sum_m X(n-m)\frac{\delta}{\delta \mu} g(m) \tag{72}$$

$$= \sum_m X(n-m)\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{m-\mu}{\sigma}\right)^2} \left(\frac{(m-\mu)^2}{\sigma^4} - \frac{1}{\sigma^2}\right) \tag{73}$$

To derive $\frac{\delta O}{\delta X}$ we have to understand how data flows from input to output. We have at input, a vector of values $X(i)$. Those, are convoluted, and each $X(i)$ participate in each output value, $O(k)$. Because of this, we know that each output is affected by each input.
When providing the backpropagation derivatives one might expect that the $\frac{\delta C}{\delta X}$ (which is one number) is a sumation over all inputs $X(i)$. Moreover, the chain rule derivative, applied so we will have the known term $\frac{\delta C}{\delta O}$ that is, how the cost varies wrt to each output, must consider that each input affects each output.

$$\frac{\delta C}{\delta X(i)} = \sum_k \frac{\delta C}{\delta O(k)} \frac{\delta O(k)}{\delta X(i)} \tag{74}$$

Focusing on $\frac{\delta O(k)}{\delta X(i)}$:

$$\frac{\delta O(k)}{\delta X(i)} = \frac{\delta}{\delta X(i)} \sum_m X(k-m)g(m) \tag{75}$$

$$k - m = i, \text{to have a non null derivative} \tag{76}$$

$$= g(k-i) \tag{77}$$

Plugging back in the previous equation

$$\frac{\delta C}{\delta X(i)} = \sum_k \frac{\delta C}{\delta O(k)} \frac{\delta O(k)}{\delta X(i)} \tag{78}$$

$$= \sum_k \frac{\delta C}{\delta O(k)} g(k-i) \tag{79}$$

$$= \sum_k \frac{\delta C}{\delta O}(k) g(k-i) \tag{80}$$

The final equation is the correlation (not convolution) between the derivative wrt to the output (known quantity) and the kernel. If the kernel is symmetric, this is just convolution.

3.3.1 Multuiple outputs per input

In some cases one wants to have one input convoluted with many kernels and for each convolution to have only one output. In this case, there will be one cost per output as defined in previous equations. For this particular case, the gradient wrt to that input is just the output gradient multiplied by the flipped kernel:

$$\frac{\delta C}{\delta X(i)} = \frac{\delta C}{\delta O} g(k-i) \tag{81}$$

The variable i will iterate through whole input.
But, stepping back one step, sometimes we want to have multiple kernels. In this case each input will "draw" its gradient from each of such output. So we will have to sum in that direction.