

Deriving the Gradient for the Backward Pass of Batch Normalization

Sep 14, 2016

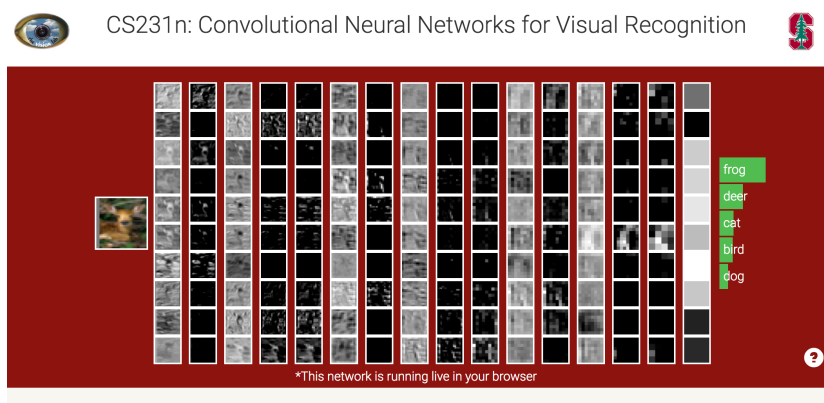


Image Courtesy

I recently sat down to work on assignment 2 of Stanford's [CS231n](#). It's lengthy and definitely a step up from the first assignment, but the insight you gain is tremendous.

Anyway, at one point in the assignment, we were tasked with implementing a Batch Normalization layer in our fully-connected net which required writing a forward and backward pass.

The forward pass is relatively simple since it only requires standardizing the input features (zero mean and unit standard deviation). The backwards pass, on the other hand, is a bit more involved. It can be done in 2 different ways:

- **staged computation**: we can break up the function into several parts, derive local gradients for them, and finally multiply them with the chain rule.
- **gradient derivation**: basically, you have to do a "pen and paper" derivation of the gradient with respect to the inputs.

It turns out that second option is faster, albeit nastier and after struggling for a few hours, I finally got it to work. This post is mainly a clear summary of the derivation along with my thought process, and I hope it can provide others with the insight and intuition of the chain rule. There is a similar tutorial online already (but I couldn't follow along very well) so if you want to check it out, head over to [Clément Thorey's Blog](#).

Finally, I've summarized the original [research paper](#) and accompanied it with a small numpy implementation which you can view on my [Github](#). With that being said, let's jump right into the blog.

Notation

Let's start with some notation.

- **BN** will stand for Batch Norm.
- f represents a layer upwards of the BN one.
- γ is the linear transformation which scales x by γ and adds β .
- \hat{x} is the normalized inputs.
- μ is the batch mean.
- σ^2 is the batch variance.

The below table shows you the inputs to each function and will help with the future derivation.

$f(y)$	$y(\hat{x}, \gamma, \beta)$	$\hat{x}(\mu, \sigma^2, x)$
--------	-----------------------------	-----------------------------

Goal: Find the partial derivatives with respect to the inputs, that is $\frac{\partial f}{\partial \gamma}$, $\frac{\partial f}{\partial \beta}$ and $\frac{\partial f}{\partial x_i}$.

Methodology: derive the gradient with respect to the centered inputs \hat{x}_i (which requires deriving the gradient w.r.t μ and σ^2) and then use those to derive one for x_i .

Chain Rule Primer

Suppose we're given a function $u(x, y)$ where $x(r, t)$ and $y(r, t)$. Then to determine the value of $\frac{\partial u}{\partial r}$ and $\frac{\partial u}{\partial t}$ we need to use the chain rule which says that:

$$\frac{\partial u}{\partial r} = \frac{\partial u}{\partial x} \cdot \frac{\partial x}{\partial r} + \frac{\partial u}{\partial y} \cdot \frac{\partial y}{\partial r}$$

That's basically all there is to it. Using this simple concept can help us solve our problem. We just have to be clear and precise when using it and not get lost with the intermediate variables.

Partial Derivatives

Here's the gist of BN taken from the paper.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$; Parameters to be learned: γ, β	
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

We're gonna start by traversing the table from left to right. At each step we derive the gradient with respect to the inputs in the cell.

Cell 1

$f(y)$	$y(\hat{x}, \gamma, \beta)$	$\hat{x}(\mu, \sigma^2, x)$
--------	-----------------------------	-----------------------------

Let's compute $\frac{\partial f}{\partial y_i}$. It actually turns out we don't need to compute this derivative since we already have it - it's the upstream derivative `dout` given to us in the function parameter.

Cell 2

$f(y)$	$y(\hat{x}, \gamma, \beta)$	$\hat{x}(\mu, \sigma^2, x)$
--------	-----------------------------	-----------------------------

Let's work on cell 2 now. We note that y is a function of three variables, so let's compute the gradient with respect to each one.

Starting with γ and using the chain rule:

$$\begin{aligned}\frac{\partial f}{\partial \gamma} &= \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \gamma} \\ &= \boxed{\sum_{i=1}^m \frac{\partial f}{\partial y_i} \cdot \hat{x}_i}\end{aligned}$$

Notice that we sum from $1 \rightarrow m$ because we're working with batches! If you're worried you wouldn't have caught that, think about the dimensions. The gradient with respect to a variable should be of the same size as that same variable so if those two clash, it should tell you you've done something wrong.

Moving on to β we compute the gradient as follows:

$$\begin{aligned}\frac{\partial f}{\partial \beta} &= \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \beta} \\ &= \boxed{\sum_{i=1}^m \frac{\partial f}{\partial y_i}}\end{aligned}$$

and finally \hat{x}_i :

$$\begin{aligned}\frac{\partial f}{\partial \hat{x}_i} &= \frac{\partial f}{\partial y_i} \cdot \frac{\partial y_i}{\partial \hat{x}_i} \\ &= \boxed{\frac{\partial f}{\partial y_i} \cdot \gamma}\end{aligned}$$

Up to now, things are relatively simple and we've already done 2/3 of the work. We **can't** compute the gradient with respect to x_i just yet though.

Cell 3

$f(y)$	$y(\hat{x}, \gamma, \beta)$	$\hat{x}(\mu, \sigma^2, x)$
--------	-----------------------------	-----------------------------

We start with μ and notice that σ^2 is a function of μ , therefore we need to add its contribution to the partial - (I've highlighted the missing partials in red):

$$\frac{\partial f}{\partial \mu} = \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial \mu} + \frac{\partial f}{\partial \sigma^2} \cdot \frac{\partial \sigma^2}{\partial \mu}$$

Let's compute the missing partials one at a time.

From

$$\hat{x}_i = \frac{(x_i - \mu)}{\sqrt{\sigma^2 + \epsilon}}$$

we compute:

$$\frac{\partial \hat{x}_i}{\partial \mu} = \frac{1}{\sqrt{\sigma^2 + \epsilon}} \cdot (-1)$$

and from

$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

we calculate:

$$\frac{\partial \sigma^2}{\partial \mu} = \frac{1}{m} \sum_{i=1}^m 2 \cdot (x_i - \mu) \cdot (-1)$$

We're missing the partial with respect to σ^2 and that is our next variable, so let's get to it and come back and plug it in here.

Ok so in the expression of the partial:

$$\frac{\partial f}{\partial \sigma^2} = \frac{\partial f}{\partial \hat{x}} \cdot \frac{\partial \hat{x}}{\partial \sigma^2}$$

let's compute $\frac{\partial \hat{x}}{\partial \sigma^2}$ in more detail. I'm gonna rewrite \hat{x} to make its derivative easier to compute:

$$\hat{x}_i = (x_i - \mu)(\sqrt{\sigma^2 + \epsilon})^{-0.5}$$

$(x_i - \mu)$ is a constant therefore:

$$\begin{aligned} \frac{\partial \hat{x}}{\partial \sigma^2} &= \sum_{i=1}^m (x_i - \mu) \cdot (-0.5) \cdot (\sqrt{\sigma^2 + \epsilon})^{-0.5-1} \\ &= -0.5 \sum_{i=1}^m (x_i - \mu) \cdot (\sqrt{\sigma^2 + \epsilon})^{-1.5} \end{aligned}$$

With all that out of the way, let's plug everything back in our previous partial!

$$\begin{aligned} \frac{\partial f}{\partial \mu} &= \left(\sum_{i=1}^m \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{\partial f}{\partial \sigma^2} \cdot \frac{1}{m} \sum_{i=1}^m -2(x_i - \mu) \right) \\ &= \left(\sum_{i=1}^m \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{\partial f}{\partial \sigma^2} \cdot (-2) \cdot \frac{1}{m} \sum_{i=1}^m x_i - \frac{1}{m} \sum_{i=1}^m \mu \right) \\ &= \left(\sum_{i=1}^m \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{\partial f}{\partial \sigma^2} \cdot (-2) \cdot \mu - \frac{m \cdot \mu}{m} \right) \\ &= \sum_{i=1}^m \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}} \end{aligned}$$

Thus we have:

$$\frac{\partial f}{\partial \mu} = \sum_{i=1}^m \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}}$$

EDIT: Just to make it clear, there's a summation in $\frac{\partial \hat{x}_i}{\partial \mu}$ because we want the dimensions to add up with respect to `dfdmu` and not `dxhatdmu`.

We finally arrive at the last variable x . Again adding the contributions from any parameter containing x we obtain:

$$\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial \hat{x}_i} \cdot \frac{\partial \hat{x}_i}{\partial x_i} + \frac{\partial f}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} + \frac{\partial f}{\partial \sigma^2} \cdot \frac{\partial \sigma^2}{\partial x_i}$$

The missing pieces are super easy to compute at this point.

$$\frac{\partial \hat{x}_i}{\partial x_i} = \frac{1}{\sqrt{\sigma^2 + \epsilon}}$$

$$\frac{\partial \mu}{\partial x_i} = \frac{1}{m}$$

$$\frac{\partial \sigma^2}{\partial x_i} = \frac{2(x_i - \mu)}{m}$$

That's it, our final gradient is

$$\frac{\partial f}{\partial x_i} = \left(\frac{\partial f}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{\partial f}{\partial \mu} \cdot \frac{1}{m} \right) + \left(\frac{\partial f}{\partial \sigma^2} \cdot \frac{2(x_i - \mu)}{m} \right)$$

Let's plug in the partials and see if we can simplify the expression some more.

$$\begin{aligned} \frac{\partial f}{\partial x_i} &= \left(\frac{\partial f}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{\partial f}{\partial \mu} \cdot \frac{1}{m} \right) + \left(\frac{\partial f}{\partial \sigma^2} \cdot \frac{2(x_i - \mu)}{m} \right) \\ &= \left(\frac{\partial f}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma^2 + \epsilon}} \right) + \left(\frac{1}{m} \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot \frac{-1}{\sqrt{\sigma^2 + \epsilon}} \right) - \left(0.5 \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot (x_j - \mu) \cdot (\sqrt{\sigma^2 + \epsilon})^{-1.5} \cdot \frac{2(x_i - \mu)}{m} \right) \\ &= \left(\frac{\partial f}{\partial \hat{x}_i} \cdot (\sigma^2 + \epsilon)^{-0.5} \right) - \left(\frac{(\sigma^2 + \epsilon)^{-0.5}}{m} \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \right) + \left(\frac{(\sigma^2 + \epsilon)^{-0.5}}{m} \cdot \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot \frac{(x_j - \mu)}{\sqrt{\sigma^2 + \epsilon}} \right) \\ &= \left(\frac{\partial f}{\partial \hat{x}_i} \cdot (\sigma^2 + \epsilon)^{-0.5} \right) - \left(\frac{(\sigma^2 + \epsilon)^{-0.5}}{m} \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \right) + \left(\frac{(\sigma^2 + \epsilon)^{-0.5}}{m} \cdot \hat{x}_i \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot \hat{x}_j \right) \end{aligned}$$

Finally, we factorize by the `sigma + epsilon` factor and obtain:

$$\frac{\partial f}{\partial x_i} = \frac{(\sigma^2 + \epsilon)^{-0.5}}{m} \left[m \frac{\partial f}{\partial \hat{x}_i} - \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} - \hat{x}_i \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot \hat{x}_j \right]$$

Recap

For organizational purposes, let's summarize the main equations we were able to derive. Using $\frac{\partial f}{\partial \hat{x}_i} = \frac{\partial f}{\partial y_i} \cdot \gamma$, we obtain the gradient with respect to our inputs:

$$\frac{\partial f}{\partial \beta} = \sum_{i=1}^m \frac{\partial f}{\partial y_i}$$

$$\frac{\partial f}{\partial \gamma} = \sum_{i=1}^m \frac{\partial f}{\partial y_i} \cdot \hat{x}_i$$

$$\frac{\partial f}{\partial x_i} = \frac{m \frac{\partial f}{\partial \hat{x}_i} - \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} - \hat{x}_i \sum_{j=1}^m \frac{\partial f}{\partial \hat{x}_j} \cdot \hat{x}_j}{m \sqrt{\sigma^2 + \epsilon}}$$

Python Implementation

Here's an example implementation using the equations we derived. `dx` is 88 characters long so I'm still wondering how the course instructors were able to write it less than 80 - maybe shorter variable names?

```
def batchnorm_backward(dout, cache):

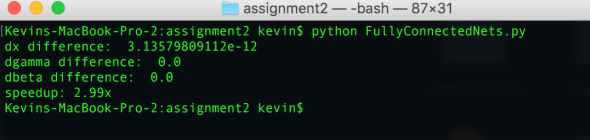
    N, D = dout.shape
    x_mu, inv_var, x_hat, gamma = cache

    # intermediate partial derivatives
    dxhat = dout * gamma

    # final partial derivatives
    dx = (1. / N) * inv_var * (N*dxhat - np.sum(dxhat, axis=0)
        - x_hat*np.sum(dxhat*x_hat, axis=0))
    dbeta = np.sum(dout, axis=0)
    dgamma = np.sum(x_hat*dout, axis=0)

    return dx, dgamma, dbeta
```

This version of the batchnorm backward pass can give you a significant boost in speed. I timed both versions and got a superb threefold increase in speed:



```
assignment2 — -bash — 87x31
Kevins-MacBook-Pro-2:assignment2 kevin$ python FullyConnectedNets.py
dx difference: 3.13579809112e-12
dgamma difference: 0.0
dbeta difference: 0.0
speedup: 2.99x
Kevins-MacBook-Pro-2:assignment2 kevin$
```

Conclusion

In this blog post, we learned how to use the chain rule in a staged manner to derive the expression for the gradient of the batch norm layer. We also saw how a smart simplification can help significantly reduce the complexity of the expression for `dx`. We finally implemented it the backward pass in Python using the code from CS231n. This version of the function resulted in a 3x speed increase!

If you're interested in the staged computation method, head over to [Kratzert's nicely written post](#).

Cheers!



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS **Daniel Lowell** • 4 months ago

Excellent write up. This is the clearest description of the backwards pass I've found so far.

One question on the algorithm for spatial versus per activation.

Using cuDNN's language here, gamma for spatial has tensor dimensions of NCHW = {1, C, 1, 1}. While per activation has NCHW = {1, C, H, W}. In batch norm the channels are an independent index.

For per activation in your above code you need gamma for summation per element an image batch, which means you need a matrix for the dxhat calculation of $dxhat = dout * gamma$ \leftarrow gamma is a matrix of dimensions HW (if we toss out the C dimension for parallel execution).

However if you're doing this for spatial, gamma is actually a scalar per channel. Which means it can be factored and further simplify the final algorithm.

So for spatial backwards pass:

$dy = dout$ (this just for my own clarity)

$invVar = (\sigma^2 + \epsilon)^{-1/2}$

$M = N * H * W$

i = batch index

j = height index

k = width index

[see more](#)  • Reply • Share ▾

ALSO ON KEVIN ZAKKA'S BLOG

My Short Term Goals For 2017

2 comments • 8 months ago •



kaiqlate — Wow, Kevin! I just discovered you via your EXCELLENT summary of Andrew Ng's Nuts and Bolts talk. I subscribed to your blog and will take on the ...

A Complete Guide to K-Nearest-Neighbors with Applications in Python and R

14 comments • a year ago •



Eugene Krevenets (Hyzhak) — thanks for tutorial, it seems you have forgotten to import `accuracy_score` from `sklearn.metrics` import ...

Deep Learning Paper Implementations: Spatial Transformer Networks - Part I

15 comments • 8 months ago •






kevinzakka — Thanks dude, appreciate the feedback. I've fixed the nifty mistake, hadn't noticed :)

Nuts and Bolts of Applying Deep Learning

9 comments • a year ago •



Daniel Seita — I tried to watch the video but I couldn't understand what Ng was talking about because the video does not have YouTube captions in them. :(...

 [Subscribe](#)  [Add Disqus to your site](#) [Add Disqus](#) [Add Disqus](#)  [Privacy](#)