Implementing Batch Normalization

Victor BUSA victor.busa@gmail.com

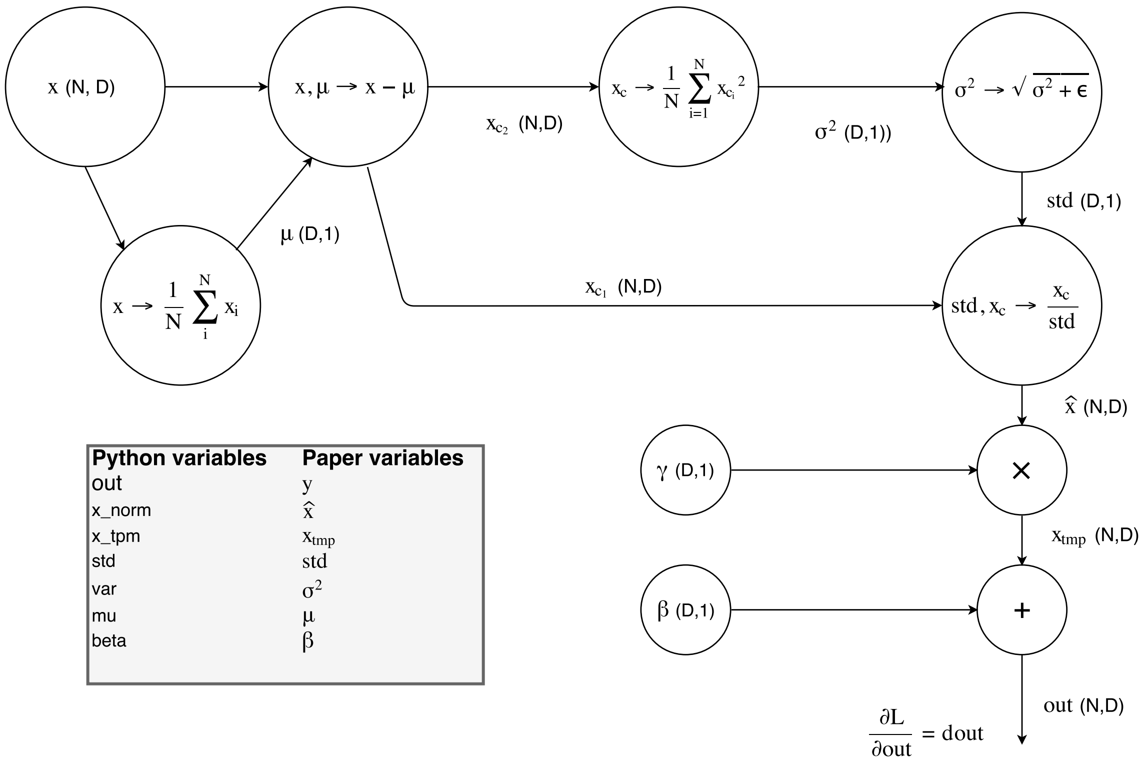
April 12, 2017

In this paper, I will describe how the gradient flow through the batch normalization layer. This work is based on the course gave at Stanford in 2016 (cs231n class about Convolutional Neural Network). Actually, one part of the 2nd assignment consist in implementing the batch normalization procedure. In my previous paper I didn't use a flowchart. Here I will use one so everybody can understand precisely how one can implement batch normalization precisely. Also I will derive the python code associated with each part. Note that the full code is in layers.py of assignment2. Finally I will also implement a faster way of computing the backward pass.

Chapter 1

Backward pass: Naive implementation

1.1 Batch normalization flowchart


x (N, D)
$x \rightarrow \frac{1}{N}\sum_{i}^{N} x_i$
μ (D,1)
$x, \mu \rightarrow x - \mu$
x_{c_2} (N,D)
x_{c_1} (N,D)
$x_c \rightarrow \frac{1}{N}\sum_{i=1}^{N} {x_{c_i}}^2$
σ^2 (D,1))
$\sigma^2 \rightarrow \sqrt{\sigma^2 + \epsilon}$
std (D,1)
$\text{std}, x_c \rightarrow \frac{x_c}{\text{std}}$
$\hat{x}$ (N,D)
γ (D,1)
$\times$
x_{tmp} (N,D)
β (D,1)
$+$
out (N,D)
$\frac{\partial L}{\partial out} = \text{dout}$

Python variables	Paper variables
out	y
x_norm	$\hat{x}$
x_tpm	x_{tmp}
std	std
var	σ^2
mu	μ
beta	β

Figure 1.1: Graph of Batch Normalization layer

The Forward pass of the Batch normalization is straightforward. We just have to look at **Figure 1** and implement the code in Python so I will directly focus on the backward pass. Let's first define some notations:

- L design the loss (the quantity computed at the end of all the layers in a neural network)
- $\frac{\partial L}{\partial y}$ correspond to the gradient of the loss L relatively to the last quantity computed during the forward pass of the batch normalization procedure. Note that in python we write *dout* to design such derivative (*dout* is then the gradient of L w.r.t y)
- to make it clear each time I write dx (python notation) it will correspond to the gradient of the loss L w.r.t to x, hence $dx = \frac{\partial L}{\partial x}$
- x is a $N \times D$ matrix. Where N is the size of the batch.

So, now that we have defined our notations. Let's define the problem. What do we want? Actually during the backward pass we want the gradient of L w.r.t to all the inputs we used to compute y. By looking at **Figure 1**, we see that we want 3 different gradients:

- $\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta}$ (in python notation: *dbeta*)
- $\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \gamma}$ (in python notation: *dgamma*)
- $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x}$ (in python notation: *dx*)

As we already know *dout* ($\frac{\partial L}{\partial y}$), we just have to compute the partial derivatives of y w.r.t the inputs β, γ, x. Let's start to compute the backward pass through each step of the Figure 1.

1.2 Computation of dbeta

We want to compute $\frac{\partial L}{\partial \beta}$. By using chain rule we can write: $\frac{\partial L}{\partial \beta} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial \beta}$. As we already know $\frac{\partial L}{\partial y}$ (*dout*), we only need to compute $\frac{\partial y}{\partial \beta}$. However we can notice that y is a (N,D) matrix and β is a (N,1) vector. So we can compute $\frac{\partial y}{\partial \beta}$ directly. We will instead focus on computing $\forall i \in [1, D]$, $\frac{\partial y}{\partial \beta_i}$. To do so we use the chain rule in higher dimension.
But let's first see what the y matrix looks like. Indeed, we need to pay attention to the fact that y is obtained using **row-wise summation/multiplication**:

$$y = \gamma \odot \widehat{x} + \beta$$

where I used $\odot$ to highlight the fact that in this relation we are dealing with a row-wise multiplication. So, now let's visualize y:

$$
\begin{aligned}
y &= \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_D \end{bmatrix} \odot \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & x_{2D} \\ \vdots & \ddots & \ddots & \vdots \\ x_{N1} & x_{N2} & \dots & x_{ND} \end{bmatrix} + \begin{bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_D \end{bmatrix} \\
&= \begin{bmatrix} \gamma_1 x_{11} + \beta_1 & \gamma_2 x_{12} + \beta_2 & \dots & \gamma_D x_{1D} + \beta_D \\ \gamma_1 x_{21} + \beta_1 & \gamma_2 x_{22} + \beta_2 & \dots & \gamma_D x_{2D} + \beta_D \\ \vdots & \ddots + & \ddots & \vdots \\ \gamma_1 x_{k1} + \beta_1 & \gamma_2 x_{k2} + \beta_2 & \dots & \gamma_D x_{kD} + \beta_D \\ \vdots & \ddots & \ddots & \vdots \\ \gamma_1 x_{N1} + \beta_1 & \gamma_2 x_{N2} + \beta_2 & \dots & \gamma_D x_{ND} + \beta_D \end{bmatrix}
\end{aligned} \tag{1.1}
$$

now that we see what y looks like we can easily notice that

$$\forall i \in [1, D] \;\; \frac{dy_{kl}}{d\beta_i} = \frac{d(\gamma_l \widehat{x}_{kl} + \beta_l)}{d\beta_i} = \frac{d\beta_l}{d\beta_i} = 1\{i = l\}$$

We can now use the chain rule in higher dimension to compute $\frac{\partial L}{\partial \beta_i}$:

$$
\begin{aligned}
\frac{dL}{d\beta_i} &= \sum_{k,l} \frac{dL}{dy_{kl}} \frac{dy_{kl}}{d\beta_i} \\
&= \sum_{k,l} \frac{dL}{dy_{kl}} 1\{i = l\} = \sum_{k} \frac{dL}{dy_{ki}}
\end{aligned} \tag{1.2}
$$

Finally we have that $\frac{\partial L}{\partial \beta}$ is a (D,1) vector (same shape as β) that has on each cell the sum of the corresponding row of $\frac{\partial L}{\partial y}$ (*dout*). In python we can compute this quantity using this piece of code:

```
# Gradient flowing along beta axes
dbeta = np.sum(dout, axis=0)

# Gradient flowing along xtmp axes
dx_tmp = dout
```

We can retain that:

- The first gate being an additive gate we only need to multiply the output gradient (y) by 1 to get the gradient that flows through x_{tmp} axes.

- If we are doing a **row-wise summation** during the forward pass, we will need to sum up the flowing gradient over **all columns** during the backward pass.

1.3 Computation of dgamma

We want to compute $\frac{\partial L}{\partial \gamma}$. Once again we will use chain rule: $\frac{\partial L}{\partial \gamma} = \frac{\partial L}{\partial x_{tmp}} \frac{\partial x_{tmp}}{\partial \gamma}$. We already know $\frac{\partial L}{\partial x_{tmp}} = \frac{\partial L}{\partial y} (= dout)$ according to the previous paragraph. So we only need to compute: $\frac{\partial x_{tmp}}{\partial \gamma} = \frac{\partial y}{\partial \gamma}$. As y is a (N,D) and γ is a (D,1) vector we will use the chain rule in higher dimension to compute this quantity:

$$\begin{aligned} \frac{dL}{d\gamma_i} &= \sum_{k,l} \frac{dL}{dy_{kl}} \frac{dy_{kl}}{d\gamma_i} \\ &= \sum_{k,l} \frac{dL}{dy_{kl}} \frac{d(\gamma_l \widehat{x}_{kl} + \beta_l)}{d\gamma_i} \\ &= \sum_{k,l} \frac{dL}{dy_{kl}} \widehat{x}_{kl} 1\{i = l\} = \sum_{k} \frac{dL}{dy_{ki}} \widehat{x}_{ki} \end{aligned} \tag{1.3}$$

Finally we have that $\frac{\partial L}{\partial \gamma}$ is a (D,1) vector (same shape as γ) that has on each cell the sum of the row of the $\gamma \widehat{x}$ matrix. In python we can compute this quantity using this piece of code:

```
# Gradient flowing along gamma axes
dgamma = np.sum(dout * x_norm, axis=0)

# Gradient flowing along x_norm axes
dx_norm = gamma * dout
```

1.4 Computation of dx

To get the gradient of L w.r.t x we need to backpropgate the gradient through each gate of the Figure 1

1.4.1 First we need to compute $\frac{\partial L}{\partial x_{c_1}} = dxc1$

$\frac{\partial L}{\partial x_{c_1}} = \frac{\partial L}{\partial \widehat{x}} \frac{\partial \widehat{x}}{\partial x_{c_1}}$. we already know according to step 2 that $\frac{\partial L}{\partial \widehat{x}} = dx\_norm = gamma * dout$, so we have:

- $\frac{\partial \widehat{x}}{\partial x_{c_1}} = std^{-1}$ and then the gradient passed along x_{c_1} axes is $dxc1 = dx\_norm * std^{-1}$
- $\frac{\partial \widehat{x}}{\partial std} = \sum_{i=1}^{N} x_c * std^{-2}$ and the gradient passed along std axes is $dstd = -dx\_norm * \sum_{i=1}^{N} x_c * std^{-2}$

Why do we have a summation over N for the gradient that flows along std axes ? For the same reason as previously we need to use the chain rule in higher dimension:

$$\begin{aligned}\frac{dL}{dstd_i} &= \sum_{k,l} \frac{dL}{d\widehat{x}_{kl}} \frac{d\widehat{x}_{kl}}{dstd_i} \\ &= \sum_{k,l} \frac{dL}{d\widehat{x}_{kl}} \frac{d\frac{x_{c_{kl}}}{std_k}}{dstd_i} = \sum_{k,l} \frac{dL}{d\widehat{x}_{kl}} x_{c_{kl}} \frac{d}{dstd_i}\left(\frac{1}{std_k}\right) \\ &= -\sum_{k,l} \frac{dL}{d\widehat{x}_{kl}} x_{c_{kl}} 1\{k=i\} std_l^{-2} = \sum_{l} \frac{dL}{d\widehat{x}_{il}} x_{c_{il}} std_i^{-2}\end{aligned} \tag{1.4}$$

Note that we can divide the $x_c, std \rightarrow \frac{x_c}{std}$ gate into a **multiply** and a **reverse** gate. In python we can implement this gradient using:

```
# Gradient flowing along std axes
dstd = -np.sum(dx_norm * xc * (std ** -2), axis=0)

# Gradient flowing along xc1 axes
dxc1 = dx_norm * (std ** -1)
```

1.4.2 Then we compute $\frac{\partial L}{\partial \sigma^2} = dvar$

Again we apply chain rule: $\frac{\partial L}{\partial \sigma^2} = \frac{\partial L}{\partial std} \frac{\partial std}{\partial \sigma^2}$. We already know $\frac{\partial L}{\partial std}$ via the previous computation. Let's then compute: $\frac{\partial std}{\partial \sigma^2}$:

$$\frac{\partial std}{\partial \sigma^2} = \frac{\partial}{\partial \sigma^2}\left(\sqrt{\sigma^2+\epsilon}\right) = 1/2 * (\sigma^2+\epsilon)^{-1} = 1/2 * std^{-1}$$

so finally we have: $\frac{\partial L}{\partial \sigma^2} = 1/2 * dstd * std^{-1}$ and in python we can write:

```
# Gradient flowing along var axes
dvar = 0.5 * dstd * (std ** -1)
```

1.4.3 We also need to compute $\frac{\partial L}{\partial x_{c_2}} = dxc2$

By chain rule we have: $\frac{\partial L}{\partial x_{c_2}} = \frac{\partial L}{\partial \sigma^2} \frac{\partial \sigma^2}{\partial x_{c_2}}$, so we just need to compute: $\frac{\partial \sigma^2}{\partial x_{c_2}}$. But here σ^2 is a vector and x_{c_2} is a matrix so we will instead compute $\frac{\partial L}{\partial x_{c2_{kl}}}$ $\forall k \in [1, N]$, $\forall l \in [1, D]$:

$$\begin{aligned}\frac{dL}{dx_{c2_{kl}}} &= \sum_{i} \frac{dL}{d\sigma_i^2} \frac{d\sigma_i^2}{dx_{c2_{kl}}} \\ &= \sum_{i} \frac{dL}{d\sigma_i^2} \frac{1}{N} \frac{d}{dx_{c2_{kl}}}\left(\sum_{p=1}^{N} x_{c2_{pi}}^2\right) = \sum_{i} \frac{dL}{d\sigma_i^2} \frac{2}{N} 1\{l=i\} x_{c2_{kl}} \\ &= \frac{2}{N} \frac{dL}{d\sigma_l^2} x_{c2_{kl}}\end{aligned} \tag{1.5}$$

So finally we can easily see that in term of matrix multiplication we have : $\frac{\partial L}{\partial x_{c_2}} = dvar * \frac{2}{N} x_c$ In python we can write:

```
# Gradient flowing along xc2 axes
# very important 2.0 / N and not 2 / N
# because we are using python 2.7
dxc2 = (2.0 / N) * dvar * xc
```

1.4.4 Again we need $\frac{\partial L}{\partial x_c} = dmu$

here we have two different gradients that are coming to the $\mu \rightarrow x - \mu$ gate so we have to add those two different gradients. So we have $\frac{\partial L}{\partial x_c} = \frac{\partial L}{\partial x_{c_1}} + \frac{\partial L}{\partial x_{c_2}} = \frac{\partial L}{\partial \hat{x}} * std^{-1} + \frac{2}{N} \frac{\partial L}{\partial var} * x_c$ In python we have:

```
# dxc = dxc1 + dxc2 (two incoming gradients)
dxc = dxc1 + dxc2 # (= dx_norm*std**-1 + (2 / N) * dvar * xc)
```

Also, using the same procedure as in step 1 and 2, the gradient that flows to μ is the sum over N of the incoming gradient: $\frac{\partial L}{\partial \mu} = -\sum_{i=1}^{N} \frac{\partial L}{\partial x_{c_{ij}}}$, Hence in python we have:

```
# Gradient flowing along mu axes
dmu = -np.sum(dxc, axis=0)
```

1.4.5 Finally we are able to compute $\frac{\partial L}{\partial x} = dx$

Finally we can recover $\frac{\partial L}{\partial x}$. Again using the chain rule and the fact that the last gate receives 2 incoming gradients, we have:

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial \mu} \frac{\partial \mu}{\partial x} + \frac{\partial L}{\partial x_c} \frac{\partial x_c}{\partial x}$$

Let's compute $\frac{\partial \mu}{\partial x}$ first. As μ is a vector and x is a matrix we will instead compute $\frac{\partial L}{\partial x_{k,l}}$ using the chain rule in higher dimension. Note that this term will refer only to $\frac{\partial L}{\partial \mu} \frac{\partial \mu}{\partial x}$, I will compute $\frac{\partial L}{\partial x_c} \frac{\partial x_c}{\partial x}$ just after:

$$\begin{aligned}\frac{dL}{dx_{kl}} &= \sum_i \frac{dL}{d\mu_i}\frac{d\mu_i}{dx_{kl}} \\ &= \sum_i \frac{dL}{d\mu_i}\frac{1}{N}\frac{d}{dx_{kl}}\left(\sum_{p=1}^{N} x_{pi}\right) = \sum_i \frac{dL}{d\mu_i}\frac{1}{N}1\{l=i\} \\ &= \frac{1}{N}\frac{dL}{d\mu_l}\end{aligned} \tag{1.6}$$

So finally rewriting with matrix notations we have :

$$\frac{\partial L}{\partial \mu}\frac{\partial \mu}{\partial x} = \frac{1}{N}\frac{\partial L}{\partial \mu}$$

Then, now let's compute $\frac{\partial L}{\partial x_c}\frac{\partial x_c}{\partial x}$:

$$\frac{\partial L}{\partial x_c}\frac{\partial x_c}{\partial x} = \frac{\partial L}{\partial x_c}\frac{\partial x_c}{\partial x} = \frac{\partial L}{\partial x_c}\frac{\partial}{\partial x}(x-\mu) = \frac{\partial L}{\partial x_c}I_{ND} = \frac{\partial L}{\partial x_c} \tag{1.7}$$

Here I_{ND} is the identity matrix of size (N, D). Finally we have:

$$\frac{\partial L}{\partial x} = \frac{1}{N}\frac{\partial L}{\partial \mu} + \frac{\partial L}{\partial x_c}$$

In python we can write:

```
#final gradient dL/dx
dx = dxc + dmu / N
```

Chapter 2

Backward pass: Faster implementation

In this part we will derive a faster implementation of the backward pass using the chain rule in higher dimension. We will first define the problem correctly. I will use the notation of the CS231n assignment to be sure we agree on the same notations.

2.1 Goal

Our objective didn't change. We still want to compute $\frac{\partial L}{\partial x}$, $\frac{\partial L}{\partial \gamma}$ and $\frac{\partial L}{\partial \beta}$. We already saw in the first part how to compute $\frac{\partial L}{\partial \gamma}$ and $\frac{\partial L}{\partial \beta}$ directly. We will hence only focus on how to compute $\frac{\partial L}{\partial x}$ straight away.

2.2 Problem

Before attacking the problem, let's define it correctly: We have :

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1l} & \dots & x_{1D} \\ x_{21} & x_{22} & \dots & x_{2l} & \dots & x_{2D} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ x_{k1} & x_{k2} & \dots & x_{kl} & \dots & x_{kD} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ x_{N1} & x_{N2} & \dots & x_{Nl} & \dots & x_{ND} \end{bmatrix} \mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_k \\ \vdots \\ \mu_D \end{bmatrix} \sigma^2 = \begin{bmatrix} {\sigma_1}^2 \\ {\sigma_2}^2 \\ \vdots \\ {\sigma_k}^2 \\ \vdots \\ {\sigma_D}^2 \end{bmatrix}$$

so actually when we are writing

$$\widehat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

it actually means that $\forall k \in [1, N]$, $\forall i \in [1, D]$

$$\widehat{x}_{kl} = (x_{kl} - \mu_l)({\sigma^2}_l + \epsilon)^{-1/2}$$

We want to compute $\frac{\partial L}{\partial x}$. To do so we will use the chain rule in higher dimension:

$$\frac{dL}{dx_{ij}} = \sum_{\substack{k\in[1,N]\\ l\in[1,D]}} \frac{dL}{d\widehat{x}_{kl}} \frac{d\widehat{x}_{kl}}{dx_{ij}}$$

We don't know the derivatives in the summation and we don't know how to compute $\frac{dL}{d\widehat{x}_{kl}}$ because we don't have access to L directly. Yet we have access to $\frac{dL}{dy}$ (that is our *dout* in Python notation). So we will introduce this term in the chain rule and it give us:

$$\frac{dL}{dx_{ij}} = \sum_{\substack{k\in[1,N]\\ l\in[1,D]}} \frac{dL}{dy_{kl}} \frac{dy_{kl}}{d\widehat{x}_{kl}} \frac{d\widehat{x}_{kl}}{dx_{ij}} \tag{2.1}$$

So now we will only need to compute $\frac{dy_{kl}}{d\widehat{x}_{kl}}$, and $\frac{d\widehat{x}_{kl}}{dx_{ij}}$ because we have access to the expression of both y and $\widehat{x}$. So let's do it:

$$\frac{dy_{kl}}{d\widehat{x}_{kl}} = \frac{d\gamma_l \widehat{x}_{kl} + \beta_l}{d\widehat{x}_{kl}} = \gamma_l \tag{2.2}$$

That one was straightforward ! Now let's compute the other derivative:

$$\begin{aligned} \frac{d\widehat{x}_{kl}}{dx_{ij}} &= \frac{d(x_{kl} - \mu_l) * ({\sigma^2}_l + \epsilon)^{-1/2}}{dx_{ij}} \\ &= \frac{d(x_{kl} - \mu_l)}{dx_{ij}} ({\sigma^2}_l + \epsilon)^{-1/2} + (x_{kl} - \mu_l) \frac{d(\sigma_l^2 + \epsilon)^{-1/2}}{dx_{ij}} \end{aligned} \tag{2.3}$$

So now let's compute the first derivative:

$$\begin{aligned} \frac{d(x_{kl} - \mu_l)}{dx_{ij}} &= \frac{dx_{kl}}{dx_{ij}} - \frac{d\mu_l}{dx_{ij}} = 1\{i = k,\ j = l\} - \frac{d}{dx_{ij}} \left(\frac{1}{N} \sum_{i=1}^{N} x_{il} \right) \\ &= 1\{i = k,\ j = l\} - \frac{1}{N} 1\{j = l\} \end{aligned} \tag{2.4}$$

This one was quite straightforward, let's handle the other derivative:

$$\frac{d(\sigma_l^2 + \epsilon)^{-1/2}}{dx_{ij}} = -\frac{1}{2} \frac{d(\sigma_l^2 + \epsilon)}{dx_{ij}} (\sigma_l^2 + \epsilon)^{-3/2} \tag{2.5}$$

So we need to compute $\frac{d(\sigma_l^2+\epsilon)}{dx_{ij}}$:

$$\begin{aligned} \frac{d(\sigma_l^2 + \epsilon)}{dx_{ij}} &= \frac{d}{x_{ij}} \left(\frac{1}{N} \sum_{q=1}^{N} (x_{ql} - \mu_l)^2 \right) \\ &= \frac{2}{N} \sum_{q=1}^{N} (x_{ql} - \mu_l) \frac{d}{dx_{ij}} (x_{ql} - \mu_l) \end{aligned} \tag{2.6}$$

Using equation (2.4) we have:

$$\begin{aligned}\frac{d(\sigma_l^2+\epsilon)}{dx_{ij}} &= \frac{2}{N}\sum_{q=1}^{N}(x_{ql}-\mu_l)(1\{i=q,\ j=l\}-\frac{1}{N}1\{j=l\}) \\ &= \frac{2}{N}\left[\sum_{q=1}^{N}(x_{ql}-\mu_l)1\{i=q,\ j=l\}-\frac{1}{N}\sum_{q=1}^{N}(x_{ql}-\mu_l)1\{j=l\})\right] \\ &= \frac{2}{N}\left[(x_{il}-\mu_l)1\{j=l\}-\frac{1}{N}1\{j=l\}\left(\sum_{q=1}^{N}x_{ql}-\mu_l\right)\right]\end{aligned} \tag{2.7}$$

To simplified even more this last expression, let's focus on the sum:

$$\begin{aligned}\sum_{q=1}^{N}x_{ql}-\mu_l &= \sum_{q=1}^{N}x_{ql}-\sum_{q=1}^{N}\mu_l \\ &\triangleq N\mu_l-\mu_l\sum_{q=1}^{N}1 = N\mu_l-N\mu_l=0\end{aligned} \tag{2.8}$$

So finally, the second term in (2.7) disappear and we have:

$$\frac{d(\sigma_l^2+\epsilon)}{dx_{ij}} = \frac{2}{N}(x_{il}-\mu_l)1\{j=l\} \tag{2.9}$$

Combining (2.3), (2.5) and (2.9) we finally have:

$$\frac{d(\sigma_l^2+\epsilon)^{-1/2}}{dx_{ij}} = \left(1\{i=k,\ j=l\}-\frac{1}{N}1\{j=l\}\right)(\sigma_l^2+\epsilon)^{-1/2}-\frac{1}{N}(x_{kl}+\mu_l)(\sigma_l^2+\epsilon)^{-3/2}(x_{il}-\mu_l)1\{j=l\} \tag{2.10}$$

Finally we can recover the full $\frac{\partial L}{\partial x}$ using (2.1), (2.2), (2.10) and we have:

$$\frac{dL}{dx_{ij}} = \sum_{\substack{k\in[1,N]\\ l\in[1,D]}}\frac{dL}{dy_{kl}}\frac{dy_{kl}}{d\widehat{x}_{kl}}\frac{d\widehat{x}_{kl}}{dx_{ij}}$$

$$\begin{aligned}&= \sum_{\substack{k\in[1,N]\\ l\in[1,D]}}\frac{dL}{dy_{kl}}\gamma_l\left(\left[1\{i=k,\ j=l\}-\frac{1}{N}1\{j=l\}\right](\sigma_l^2+\epsilon)^{-1/2}-\frac{1}{N}(x_{kl}+\mu_l)(\sigma_l^2+\epsilon)^{-3/2}(x_{il}-\mu_l)1\{j=l\}\right) \\ &= \sum_{\substack{k\in[1,N]\\ l\in[1,D]}}\frac{dL}{dy_{kl}}\gamma_l\left[1\{i=k,\ j=l\}-\frac{1}{N}1\{j=l\}\right](\sigma_l^2+\epsilon)^{-1/2} \\ &\quad -\sum_{\substack{k\in[1,N]\\ l\in[1,D]}}\frac{dL}{dy_{kl}}\gamma_l\frac{1}{N}(x_{kl}+\mu_l)(\sigma_l^2+\epsilon)^{-3/2}(x_{ij}-\mu_l)1\{j=l\} \\ &= \frac{1}{N}(\sigma_l^2+\epsilon)^{-1/2}\gamma_j\sum_{k=1}^{N}\frac{dL}{dy_{kl}}(1\{i=k\}N-1)-\frac{1}{N}(\sigma_l^2+\epsilon)^{-3/2}(x_{ij}-\mu_j)\sum_{k=1}^{N}\frac{dL}{dy_{kj}}\gamma_j(x_{kj}-\mu_j) \\ &= \frac{1}{N}(\sigma_j^2+\epsilon)^{-1/2}\gamma_j\left(\left[N\sum_{k=1}^{N}\frac{dL}{dy_{kj}}1\{i=k\}-\sum_{k=1}^{N}\frac{dL}{dy_{kj}}\right]-(\sigma_j^2+\epsilon)^{-1}(x_{ij}-\mu_j)\sum_{k=1}^{N}\frac{dL}{dy_{kj}}(x_{kj}-\mu_j)\right) \\ &= \frac{1}{N}(\sigma_j^2+\epsilon)^{-1/2}\gamma_j\left(N\frac{dL}{dy_{ij}}-\sum_{k=1}^{N}\frac{dL}{dy_{kj}}-(\sigma_j^2+\epsilon)^{-1}(x_{ij}-\mu_j)\sum_{k=1}^{N}\frac{dL}{dy_{kj}}(x_{kj}-\mu_j)\right)\end{aligned}$$

So Finally we could have come up with a expression for $\frac{dL}{dx_{ij}}$. We just need to recall that $\frac{dL}{dx}$ is a (N,D) matrix (same shape as x) that looks like:

$$\begin{bmatrix} \frac{dL}{dx_{11}} & \frac{dL}{dx_{12}} & \cdots & \frac{dL}{dx_{1l}} & \cdots & \frac{dL}{dx_{1D}} \\ \frac{dL}{dx_{21}} & \frac{dL}{dx_{22}} & \cdots & \frac{dL}{dx_{2l}} & \cdots & \frac{dL}{dx_{2D}} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ \frac{dL}{dx_{k1}} & \frac{dL}{dx_{k2}} & \cdots & \frac{dL}{dx_{kl}} & \cdots & \frac{dL}{dx_{kD}} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \\ \frac{dL}{dx_{N1}} & \frac{dL}{dx_{N2}} & \cdots & \frac{dL}{dx_{Nl}} & \cdots & \frac{dL}{dx_{ND}} \end{bmatrix} \tag{2.11}$$

Having this in mind we can actually come up with the python implementation that looks like:

```
N = dout.shape[0]
dx = (1. / N) * (var + eps)**(-1./2) * gamma \
               * (N * dout - np.sum(dout, axis=0)\
               - (var + eps)**(-1.0) * (x - mu.T) \
               * np.sum(dout * (x - mu.T), axis=0))
dbeta = np.sum(dout, axis=0)
dgamma = np.sum(dout * x_norm, axis=0)
```

2.3 Conclusion

We saw how we can implement batch normalization in Python. To do so we have drawn a graph of all the elementary operations we needed to compute the forward pass. The backward pass can then be computed directly using this graph. The thing to retain is that we used the chain rule in higher dimension all along. Once we understand how it works it is quite straightforward.