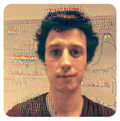

What does the gradient flowing through batch normalization looks like ?

This past week, I have been working on the assignments from the Stanford CS class CS231n: Convolutional Neural Networks for Visual Recognition. In particular, I spent a few hours deriving a correct expression to backpropagate the batchnorm regularization (Assigment 2 - Batch Normalization) . While this post is mainly for me not to forget about what insights I have gained in solving this problem, I hope it could be useful to others that are struggling with back propagation.

Batch normalization

Batch normalization is a recent idea introduced by Ioffe et al, 2015 to ease the training of large neural networks. The idea behind it is that neural networks tend to learn better when their input features are uncorrelated with zero mean and unit variance. As each layer within a neural network see the activations of the previous layer as inputs, the same idea could be apply to each layer. Batch normalization does exactly that by normalizing the activations over the current batch in each hidden layer, generally right before the non-linearity.

To be more specific, for a given input batch x of size (N, D) going through a hidden layer of size H, some weights w of size (D, H) and a bias b of size (H), the common layer structure with batch norm looks like

1. Affine transformation

   $$h = XW + b$$

   where h contains the results of the linear transformation (size (N, H)).

2. Batch normalization transform

   $$y = \gamma\hat{h} + \beta$$

   where γ and β are learnable parameters and

   $$\hat{h} = (h - \mu)(\sigma^2 + \epsilon)^{-1/2}$$

   contains the zero mean and unit variance version of h (size (N, H)). Indeed, the parameter μ (H) and σ^2 (H) are the respective average and standard deviation of each activation over the full batch (of size N). Note that, this expression implicitly assume broadcasting as h is of size (N, H) and both μ and σ have size equal to (H). A more correct expression would be

$$\hat{h}_{kl} = (h_{kl} - \mu_l)(\sigma_l^2 + \epsilon)^{-1/2}$$

where

$$\mu_l = \frac{1}{N} \sum_p h_{pl}$$

$$\sigma_l^2 = \frac{1}{N} \sum_p (h_{pl} - \mu_l)^2.$$

with $k = 1, \ldots, N$ and $l = 1, \ldots, H$.

3. Non-linearity activation, say ReLu for our example

$$a = ReLu(y)$$

which now see a zero mean and unit variance input and where a contains the activations of size (N, H). Also note that, as γ and β are learnable parameters, the network can unlearn the batch normalization transformation. In particular, the claim that the non-linearity sees a zero mean and unit variance input is only certainly true in the first forward call as γ and β are usually initialized to 1 and 0 respectively.

Derivation

Implementing the forward pass of the batch norm transformation is straightforward

```
# Forward pass
mu = 1/N*np.sum(h,axis =0) # Size (H,)
sigma2 = 1/N*np.sum((h-mu)**2,axis=0)# Size (H,)
hath = (h-mu)*(sigma2+epsilon)**(-1./2.)
y = gamma*hath+beta
```

The tricky part comes with the backward pass. As the assignment proposes, there are two strategies to implement it.

1. Write out a computation graph composed of simple operations and backprop through all intermediate values
2. Work out the derivatives on paper.

The 2nd step made me realize I did not fully understand backprogation before this assignment. Backpropation, an abbreviation for “backward propagation of errors”, calculates the gradient of a loss function $\mathcal{L}$ with respect to all the parameters of the network. In our case, we need to calculate the gradient with respect to γ, β and the input h.

Mathematically, this reads $\frac{d\mathcal{L}}{d\gamma}, \frac{d\mathcal{L}}{d\beta}, \frac{d\mathcal{L}}{dh}$ where each gradient with respect to a quantity contains a vector of size equal to the quantity itself. For me, the aha-moment came when I decided to properly write the expression for these gradients. For instance, the gradient with respect to the input h literally reads

$$\frac{d\mathcal{L}}{dh} = \begin{pmatrix} \frac{d\mathcal{L}}{dh_{11}} & .. & \frac{d\mathcal{L}}{dh_{1H}} \\ .. & \frac{d\mathcal{L}}{dh_{kl}} & .. \\ \frac{d\mathcal{L}}{dh_{N1}} & \ldots & \frac{d\mathcal{L}}{dh_{NH}} \end{pmatrix}.$$

To derive a close form expression for this expression, we first have to recall that the main idea behind backpropagation is chain rule. Indeed, thanks to the previous backward pass, i.e. into ReLu in our example, we already know

$$\frac{d\mathcal{L}}{dy} = \begin{pmatrix} \frac{d\mathcal{L}}{dy_{11}} & \ldots & \frac{d\mathcal{L}}{dy_{1H}} \\ \ldots & \frac{d\mathcal{L}}{dy_{kl}} & \ldots \\ \frac{d\mathcal{L}}{dy_{N1}} & \ldots & \frac{d\mathcal{L}}{dy_{NH}} \end{pmatrix}.$$

where

$y_{kl} = \gamma_l \hat{h}_{kl} + \beta_l.$

We can therefore chain the gradient of the loss with respect to the input h_{ij} by the gradient of the loss with respect to **ALL** the outputs y_{kl} which reads

$$\frac{d\mathcal{L}}{dh_{ij}} = \sum_{k,l} \frac{d\mathcal{L}}{dy_{kl}} \frac{dy_{kl}}{dh_{ij}},$$

which we can also chain by the gradient with respect to the centred input $\hat{h}_{kl}$ to break down the problem a little more

$$\frac{d\mathcal{L}}{dh_{ij}} = \sum_{k,l} \frac{d\mathcal{L}}{dy_{kl}} \frac{dy_{kl}}{d\hat{h}_{kl}} \frac{d\hat{h}_{kl}}{dh_{ij}}.$$

The second term in the sum simply reads $\frac{dy_{kl}}{d\hat{h}_{kl}} = \gamma_l$. All the fun part actually comes when looking at the third term in the sum.

Instead of jumping right into the full derivation, let's focus on just the translation for one moment. Assuming the batch norm as just being a translation, we have

$$\hat{h}_{kl} = h_{kl} - \mu_l$$

where the expression of μ_l is given above. In that case, we have

$$\frac{d\hat{h}_{kl}}{dh_{ij}} = \delta_{i,k}\delta_{j,l} - \frac{1}{N}\delta_{j,l}.$$

where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise. Therefore, the first term is 1 only if $k = i$ and $l = j$ and the second term is $1/N$ only when $l = j$. Indeed, the gradient of $\hat{h}$ with respect to the j input of the i batch, which is precisely what the left hand term means, is non-zero only for terms in the j dimension. I think if you get this one, you are good to backprop whatever function you encounter so make sure you understand it before going further.

This is just the case of translation though. What if we consider the real batch normalization transformation ?

In that case, the transformation considers both translation and rescaling and reads

$$\hat{h_{kl}} = (h_{kl} - \mu_l)(\sigma_l^2 + \epsilon)^{-1/2}.$$

Therefore, the gradient of the centred input $\hat{h}_{kl}$ with respect to the input h_{ij} reads

$$\frac{d\hat{h}_{kl}}{dh_{ij}} = (\delta_{ik}\delta_{jl} - \frac{1}{N}\delta_{jl})(\sigma_l^2 + \epsilon)^{-1/2} - \frac{1}{2}(h_{kl} - \mu_l)\frac{d\sigma_l^2}{dh_{ij}}(\sigma_l^2 + \epsilon)^{-3/2}$$

where

$$\sigma_l^2 = \frac{1}{N}\sum_p \left(h_{pl} - \mu_l\right)^2.$$

As the gradient of the standard deviation σ_l^2 with respect to the input h_{ij} reads

$$\begin{aligned}
\frac{d\sigma_l^2}{dh_{ij}} &= \frac{1}{N}\sum_p 2\left(\delta_{ip}\delta_{jl} - \frac{1}{N}\delta_{jl}\right)\left(h_{pl} - \mu_l\right) \\
&= \frac{2}{N}(h_{il} - \mu_l)\delta_{jl} - \frac{2}{N^2}\sum_p \delta_{jl}\left(h_{pl} - \mu_l\right) \\
&= \frac{2}{N}(h_{il} - \mu_l)\delta_{jl} - \frac{2}{N}\delta_{jl}\left(\frac{1}{N}\sum_p h_{pl} - \mu_l\right) \\
&= \frac{2}{N}(h_{il} - \mu_l)\delta_{jl}
\end{aligned}$$

we finally have

$$\frac{d\hat{h}_{kl}}{dh_{ij}} = (\delta_{ik}\delta_{jl} - \frac{1}{N}\delta_{jl})(\sigma_l^2 + \epsilon)^{-1/2} - \frac{1}{N}(h_{kl} - \mu_l)(h_{il} - \mu_l)\delta_{jl}(\sigma_l^2 + \epsilon)^{-3/2}.$$

Wrapping everything together, we finally find that the gradient of the loss function $\mathcal{L}$ with respect to the layer inputs finally reads

$$
\begin{aligned}
\frac{d\mathcal{L}}{dh_{ij}} &= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \frac{dy_{kl}}{d\hat{h}_{kl}} \frac{d\hat{h}_{kl}}{dh_{ij}} \\
&= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \gamma_l \bigg((\delta_{ik}\delta_{jl} - \frac{1}{N}\delta_{jl})(\sigma_l^2 + \epsilon)^{-1/2} - \frac{1}{N}(h_{kl} - \mu_l)(h_{il} - \mu_l)\delta_{jl}(\sigma_l^2 + \epsilon)^{-3/2} \bigg) \\
&= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \gamma_l \bigg((\delta_{ik}\delta_{jl} - \frac{1}{N}\delta_{jl})(\sigma_l^2 + \epsilon)^{-1/2} \bigg) - \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \gamma_l \bigg(\frac{1}{N}(h_{kl} - \mu_l)(h_{il} - \mu_l)\delta_{jl}(\sigma_l^2 + \\
&= \frac{d\mathcal{L}}{dy_{ij}} \gamma_j (\sigma_j^2 + \epsilon)^{-1/2} - \frac{1}{N} \sum_k \frac{d\mathcal{L}}{dy_{kj}} \gamma_j (\sigma_j^2 + \epsilon)^{-1/2} - \frac{1}{N} \sum_k \frac{d\mathcal{L}}{dy_{kj}} \gamma_j \big((h_{kj} - \mu_j)(h_{ij} - \mu_j)(\sigma_j^2 \\
&= \frac{1}{N} \gamma_j (\sigma_j^2 + \epsilon)^{-1/2} \Bigg(N \frac{d\mathcal{L}}{dy_{ij}} - \sum_k \frac{d\mathcal{L}}{dy_{kj}} - (h_{ij} - \mu_j)(\sigma_j^2 + \epsilon)^{-1} \sum_k \frac{d\mathcal{L}}{dy_{kj}} (h_{kj} - \mu_j) \Bigg)
\end{aligned}
$$

The gradients of the loss with respect to γ and β is much more straightforward and should not pose any problem if you understood the previous derivation. They read

$$
\begin{aligned}
\frac{d\mathcal{L}}{d\gamma_j} &= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \frac{dy_{kl}}{d\gamma_j} \\
&= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \hat{h}_{kl} \delta_{lj} \\
&= \sum_{k} \frac{d\mathcal{L}}{dy_{kj}} (h_{kj} - \mu_j)(\sigma_j^2 + \epsilon)^{-1/2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{d\mathcal{L}}{d\beta_j} &= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \frac{dy_{kl}}{d\beta_j} \\
&= \sum_{kl} \frac{d\mathcal{L}}{dy_{kl}} \delta_{lj} \\
&= \sum_{k} \frac{d\mathcal{L}}{dy_{kj}}
\end{aligned}
$$

After the hard work derivation are done, you can simply just drop these expressions into python for the calculation. The implementation of the batch norm backward pass looks like

```
mu = 1./N*np.sum(h, axis = 0)
var = 1./N*np.sum((h-mu)**2, axis = 0)
dbeta = np.sum(dy, axis=0)
dgamma = np.sum((h - mu) * (var + eps)**(-1. / 2.) * dy, axis=0)
dh = (1. / N) * gamma * (var + eps)**(-1. / 2.) * (N * dy - np.sum
    - (h - mu) * (var + eps)**(-1.0) * np.sum(dy * (h - mu), axis=
```

and with that, you good to go !

Conclusion

In this post, I focus on deriving an analytical expression for the backward pass to implement batch-norm in a fully connected neural networks. Indeed, trying to get an expression by just looking at the centered inputs and trying to match the dimensions to get $d\gamma$, $d\beta$ and dh simply do not work this time. In contrast, working the derivative on papers nicely leads to the solution ;)

To finish, I'd like to thank all the team from the CS231 Stanford class who do a fantastic work in vulgarizing the knowledge behind neural networks.

For those who want to take a look to my full implementation of batch normalization for a fully-connected neural networks, you can found it here.

Written on January 28, 2016