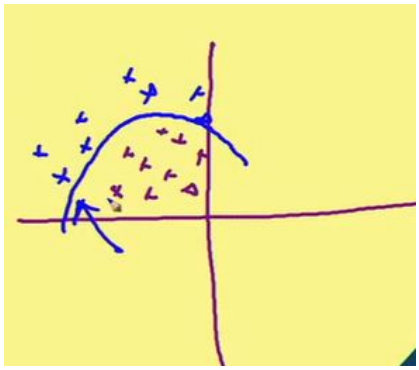
Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift (Summary)

This paper offers a normalization technique that speeds up the training of deep neural networks by reducing the **internal covariate shift**.

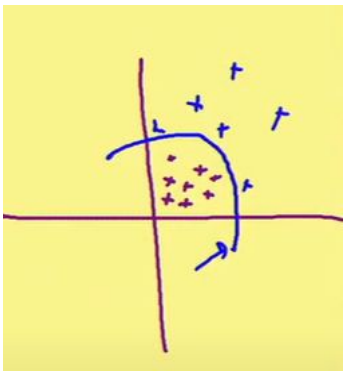
Internal covariate shift: It is the variation in the values of hidden layers i.e during back-propagation the weights are constantly updated leading to change in its output values, due to which the distribution of outputs regularly changes. Thus, BN aims to reduce these covariate shifts in order to make training faster.

This also encounters the change in the distribution of training data in different mini-batches which lead to a change in decision boundaries in different mini-batches.

Example-



Mini batch-1



Mini batch-2

This not only encounters the above problems but also regularizes the model(preventing the overfitting by creating noise in input), thus reducing the need for dropouts (though we should use them as they give better results in some new research paper but also encounters some loss of information)

Why is this needed?

We could just adjust the mean to 0 and the standard deviation to 1 but this leads to exploding of parameters also if gradient descent considers this normalization than derivatives involves so many time-consuming calculations.

$$\frac{\partial \text{Norm}(x, \mathcal{X})}{\partial x} \text{ and } \frac{\partial \text{Norm}(x, \mathcal{X})}{\partial \mathcal{X}};$$

So BN was introduced.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Mean and variance are calculated on the mini-batch
(**Note**: In the case of convolution networks means and standard deviation are calculated by taking out a particular channel from the feature map of all the mini-batch examples. This has so many limitations as it requires a full training set, these limitations were later overcome in layer normalization, instance normalization, group normalization(best for validation dataset in case of imagenet) Here I am not covering all these due to limitation of summary words.)
Normalize: It makes the mean 0 and standard deviation 1

Scale and shift: It further changes the mean and standard deviation according to the network. As here gamma and betta are trainable parameters so they will be trained in the form of subnetwork through gradient descent.

$$\frac{\partial \ell}{\partial \widehat{x}_i} = \frac{\partial \ell}{\partial y_i} \cdot \gamma$$

$$\frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} = \sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot (x_i - \mu_{\mathcal{B}}) \cdot \frac{-1}{2}(\sigma_{\mathcal{B}}^2 + \epsilon)^{-3/2}$$

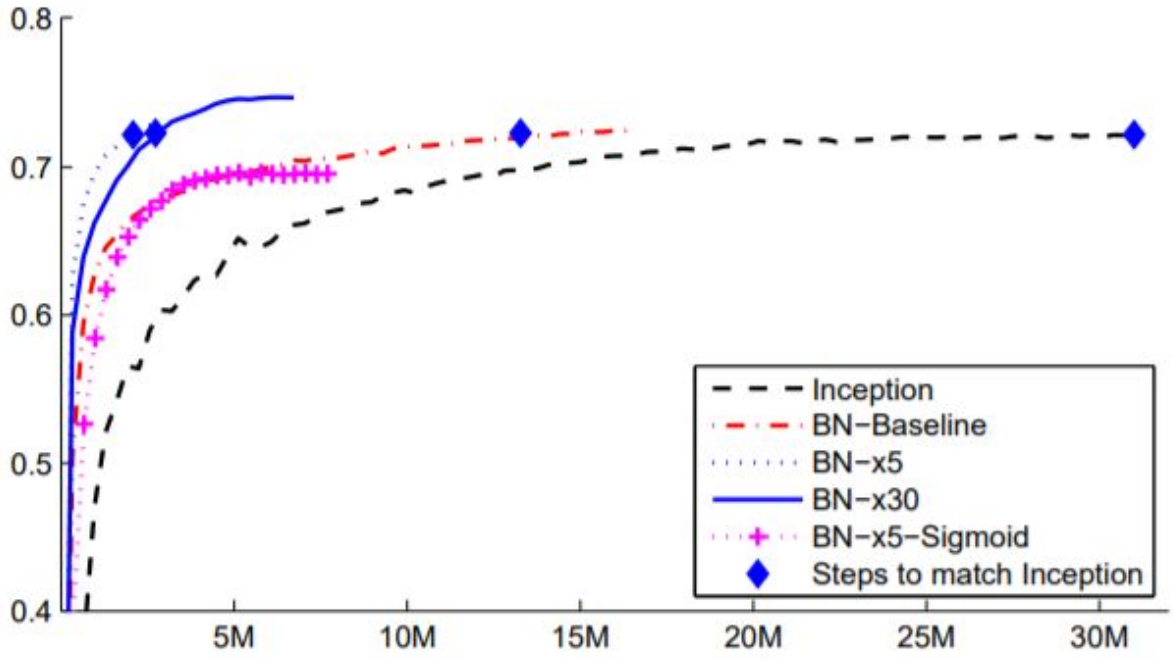
$$\frac{\partial \ell}{\partial \mu_{\mathcal{B}}} = \left(\sum_{i=1}^{m} \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{\sum_{i=1}^{m} -2(x_i - \mu_{\mathcal{B}})}{m}$$

$$\frac{\partial \ell}{\partial x_i} = \frac{\partial \ell}{\partial \widehat{x}_i} \cdot \frac{1}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_{\mathcal{B}}^2} \cdot \frac{2(x_i - \mu_{\mathcal{B}})}{m} + \frac{\partial \ell}{\partial \mu_{\mathcal{B}}} \cdot \frac{1}{m}$$

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

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

Results of BN on image classifications



X-axis-training steps, Y-axis- validation accuracy

Here xY means increasing the learning rate Y times. By watching the curve we can say that by using batch normalization we can achieve the same accuracy in fewer training steps.

Special note: batch normalization also reduces the stretching of loss function curve i.e, it reduces the stretching of the bowl shaping curve (generating due to different ranges of features) leading to faster training and allowing us to use larger training rates.

--By Dhruv Grover