Stochastic Whitening Batch Normalization

Luca Nunziante

September 2022

Abstract

This report stems from [6] where a novel approach to data whitening in Neural Networks is introduced. Namely, they propose an algorithm to gradually estimate the whitening matrix during training in an online fashion to overcome the computational bottleneck of the previously existing methods. The first part of this report recalls the standard Batch Normalization method and reports some enlightening results on its effect. Then, the concept of whitening is briefly introduced, followed by the presentation of the newly proposed whitening algorithm and the network used. The work ends with the performance comparison between Batch Normalization and Stochastic Whitening Batch Normalization.

1 Batch Normalization

Batch Normalization (BN) is a technique widely used to train Deep Neural Networks and it was firstly introduced in [5] to reduce the Internal Covariate Shift (ICS). ICS can be summarised as being the change in the the distribution of each layer's inputs during training due to the change in the parameters of the previous layers, and also due to a possible change in the input mini-batch features distribution. The mitigation to this was the normalization of layer inputs, i.e.

$$y = \gamma(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}) + \beta \tag{1}$$

where x is the input, μ, σ^2 are mean and variance of the features along the mini-batch[1], ϵ is a small positive stabilizing factor, and γ, β are learnable parameters used to restore the representation power of the network. In many cases, this technique brings a sensible performance improvement.
However, it has been proved in [5] that there is a tenuous link, if any, between the performance gain thanks to BN and the reduction of internal covariate shift, as can be easily seen in Figure 1 where the distribution of the activations of a given layer are visualized.
In [5] they find that BN works because it reparametrizes the underlying optimization problem to make its landscape significantly smoother.

[1]In case of Convolutional Neural Networks, mean and variance are compute separately for each channel and Eq. 1 is applied channel-wise.

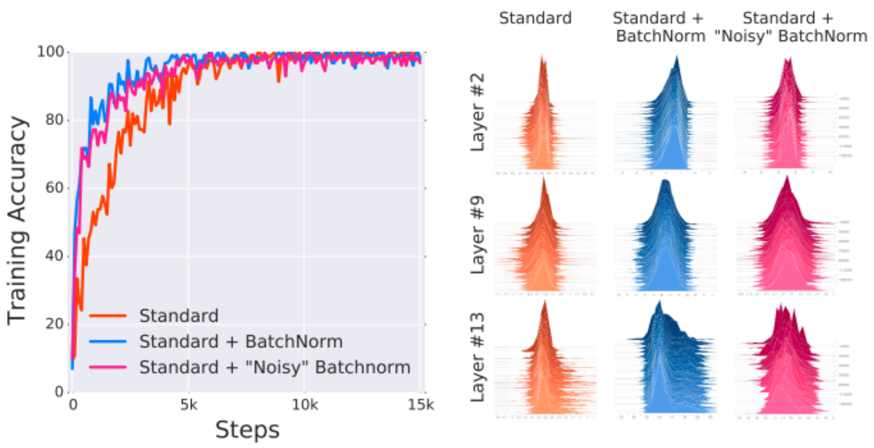


Figure 1: Connections between distributional stability and BN performance: the comparison is done between the same VGG network trained without BN (Standard), with BN (Standard + BN) and with explicit "covariate shift" added to BN layers (Standard + "Noisy" BN). In the later case, instability was induced by adding time-varying, non-zero mean and non-unit variance noise independently to each batch normalized activation. The "noisy" BatchNorm model nearly matches the performance of standard BatchNorm model, despite complete distributional instability. Figure and caption taken from [5].

2 Whitening

Before introducing the Stochastic Whitening Batch Normalization (SWBN) algorithm, we briefly review the whitening concept.
A random zero mean vector $z \in \mathbb{R}^c$ is said to be white if the expectation $E[\cdot]$ of its covariance satisfies $E[zz^T] = \boldsymbol{I_d}$ where $\boldsymbol{I_d}$ is the identity matrix. Whitening consists in transforming a random zero mean vector x into z with a linear transformation, i.e. it consists in finding $\boldsymbol{W} \in \mathbb{R}^{c\times c}$ such that $z = \boldsymbol{W}x$.
The SWBN algorithm aims at finding the whitening matrix gradually by minimizing an appropriate loss, instead of finding it by Singular Value Decomposition of $E[xx^T]$ at each step as other methods do, like PCA and ZCA: because of the ICS, that we have seen may still be present using batch normalization, the whitening matrix may be very different at each training step hence doing the whole whitening process every time may be wasteful.
The online learning of $\boldsymbol{W}$ is done by minimizing an appropriate whitening loss function, or whitening criterion, and at the core of SWBN there are the following two

$$\begin{aligned} C_{KL} &= \frac{1}{2}(tr(\boldsymbol{W\Sigma_x W}^T) - ln\ det(\boldsymbol{W\Sigma_x W}^T)) \\ C_{Fro} &= \frac{1}{2}||\boldsymbol{I} - \boldsymbol{W\Sigma_x W}^T||_{Fro} \end{aligned} \tag{2}$$

where $||\cdot||_{Fro}$ is the Frobenius norm, and considering $y = \boldsymbol{W}x$, and $\boldsymbol{\Sigma_y} = E[yy^T] = \boldsymbol{W}E[xx^T]\boldsymbol{W}^T = \boldsymbol{W\Sigma_x W}^T$ it is clear that they both have a minimum for $\boldsymbol{W\Sigma_x W}^T = \boldsymbol{I}$.

This whitening loss is completely decoupled from the task loss, unlike other approaches like Decov [1] that introduces a layer-wise additional loss to minimize the covariance between different features in a layer.

3 SWBN algorithm

The paper [6] explains the algorithm for SWBN but does not provide any implementation: I decided to create a custom layer from scratch in keras on tensorflow that implements SWBN, and I integrated it in a custom model.

The weights of this layer are the same of a batch normalization layer, i.e. the four in Eq. 1 where mean and variance are not trainable but estimated, with the addition of a whitening matrix that is a square matrix $\mathbb{R}^{c\times c}$ where c is the number of channels, or features, of the layer input. Since this matrix will be updated solely according to the chosen criterion in Eq. 2, it will be flagged as non trainable and learning will happen thanks to the closed form update provided in Eq. 4.

Note that for the criterion derived from the Kullback-Leiber divergence, the update is not the actual gradient of the criterion with respect to $\boldsymbol{W}$ but the relative one, as the actual one would require the inversion of $\boldsymbol{W}$ that is computationally expensive. The relative gradient consists in finding a multplicative perturbation[2] Γ such that $f((I+\Gamma)W) - f(W) < 0$ where f is the criterion to minimize, as opposed to the actual gradient where we find an additive perturbation. Instead, when using C_{Fro} it is possible to use the actual gradient.

The input to the layer is considered to be a bidimensional matrix $X \in \mathbb{R}^{c\times n}$, since it is reshaped like so if that is not the case.

During training, we compute the mean and variance $\mu, \nu \in \mathbb{R}^c$ along the current mini-batch, update the mean and variance estimates as in Eq. 3 and apply batch normalization following Eq. 1 where $\mu = \mu_E, \sigma^2 = \nu_E$, but without the factors γ, β, obtaining the batch normalized samples X_s.

Then, we compute the covariance matrix Σ_x of the batch normalized samples and update the whitening matrix as $W = W - \alpha\Delta W$ where the update matrix is taken from Eq. 4.

Finally, we enforce the whitening matrix to be symmetric, i.e. $W \leftarrow 0.5\cdot(W + W^T)$

$$\begin{aligned} \mu_E &\leftarrow \eta\mu_E + (1-\eta)\mu \\ \nu_E &\leftarrow \eta\nu_E + (1-\eta)\nu \end{aligned} \tag{3}$$

$$\Delta W = \begin{cases} (W\Sigma_x W^T - I_d)W\Sigma_x & \text{for } C_{KL}, \\ \frac{(W\Sigma_x W^T - I_d)W\Sigma_x}{||I_d - W\Sigma_x W^T||_{Fro}} & \text{for } C_{Fro} \end{cases} \tag{4}$$

[2]From now on matrices will not be bolded for ease and clarity of notation

Once the whitening matrix is updated, the whitened features are computed as $X_w = WX_s$.
In the testing phase, the SWBN layer just normalizes the data by subtracting μ_E and scaling by $\sqrt{\nu_E + \epsilon}$ obtaining X_s, and then outputs the whitened features after shifting and scaling.
The constant parameters are chosen according to the reference paper, i.e. $\epsilon = 10^{-8}, \eta = 0.95, \alpha = 10^{-5}$.
The authors of [6] also advise to make the Whitening Learning Rate (WLR) α decrease according to the learning rate (lr) of the network: to accomplish this, when creating the layer one also needs to specify the desired ratio $\frac{\alpha}{lr}$ that is kept constant throughout training.

4 ResNetV2-56

In this work, following what is done in [6], I take a model and make two copies: one with BN layers and one with SWBN layers in their place. The network used is ResNetV2-56 that was introduced in [3], and the novelty with respect to the ResNet is due to the organization within a residual unit, as can be seen in Figure 2.
The network's implementation is taken from the code provided with [4], with appropriate modifications.

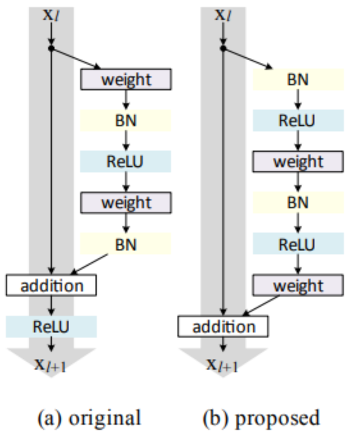


Figure 2: In (a) we see the original Residual Unit proposed in [2], while in (b) there is the novel 'pre-activation' organization proposed in [3], from where this Figure is taken, and that is used in this work. The 'weight' block is a Convolutional layer: the second one always is a Convolution with unit strides.

4.1 ResNetV2-56 architecture

The Network's architecture is as follows:

- One convolutional layer with 16 kernels of size 3×3 and stride 1. Every convolutional layer in this network has this kernel size, so it will be omitted henceforth;
- Three blocks, where each block is made of nine Residual Units as the one in Figure 2b. In the same block, all the units have the same number of filters in their Conv layers. However, while they share the same number of filters, they vary in the stride. Namely, the first Conv layer of the first unit of every block has a stride specified from outside, while in the other units it has stride always equal to one. The second Conv layer, instead, always has stride equal to 1 in every unit of every block. This means that the first unit of every block, recalling Figure 2b, must have another Conv layer, with kernels of size 1, in place of the big grey arrow in order to make the shapes match before the addition.
- A final sequence of:
  - Batch Normalization or SWBN layer, and ReLU application
  - Global average pooling and flattening of the tensor, to obtain a feature vector
  - Fully connected layer with 10 output units for classification

5 Training configuration

The network is trained on the CIFAR-10 dataset, that has 50k RGB 32×32 images for training, and 10k for testing. Training lasts for 150 epochs and at the end of each epoch, the model is validated on the whole test set.
The training configuration, hyperparameters, and data augmentation used are the same reported in [6] and [3]. Namely, the learning rate (lr) starts at 0.1 and is divided by 10 at 32k and 48k iterations; the mini-batch size is set to 128, hence each epoch is made of 391 iterations and the lr changes will occur during epochs n. 81 and 122 as can be seen in Figure 3; the weight decay is set to 10^{-4}; the optimization algorithm used is Stochastic Gradient Descent with momentum, and the momentum is set to 0.9.
The data augmentation applied consists in a random horizontal flip of the image, and a random translation of 15% in any direction.
In the BN layer - and also in the SWBN - the parameters are initialized as follows: $\gamma = 1, \beta = 0, \mu_E = 0, \nu_E = 1$, and in the SWBN layer the matrix W starts as the identity.
Training on Google Colaboratory takes about 3 hours for the model with SWBN and about half the time using BN, with an average of 73s and 35s per epoch respectively. The trainable parameters are about 855k in both cases, but in the

Model	BN	SWBN-KL
ResNetV2-56	92.92	**93.4**

Table 1: Test accuracies [%] on CIFAR-10 reported in [6]. The shown values are the result of an average over 10 runs.

SWBN case, due to the whitening matrices, the non trainable ones are about 100k with respect to the 4k in the model with BN.

6 Results

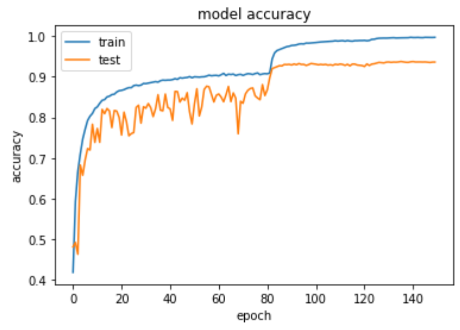
In this section, the results of the paper in terms of performance on an Image Classification task are compared with the results I replicated using the same model.
Due to the randomness in the data augmentation - and also in the initialization of the weights of the kernels in the Convolutional layers - to compare BN and SWBN more than one run are needed.
As a consequence of limited computational resources, instead of doing ten runs - as done in [6] - for each of the three methods[3], I did five for BN and five for SWBN with the criterion derived from the KL divergence since it showed slightly better results.

6.1 Performance comparison

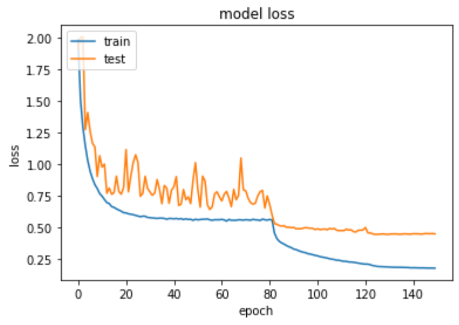
An extract of the paper results is reported in Table 1, where we see that with respect to the baseline there is a slight improvement of about half a percentage point on the test set accuracy. Similar results are reported also when using the dataset CIFAR-100, while a more sensible improvement is seen with depeer models trained on the ImageNet dataset on 1.28 milion training images, ad 50k test images to classify into 1000 classes.
My work's aim was to replicate the results in Table 1, and what I have obtained is reported in Table 2 and the model trained with my SWBN-KL implementation shows better performance of both my baseline and the paper's.
The authors do not mention whether the accuracy they report is the best one obtained during training or the one of the model at the end of training, but since my validation set is the whole test set, the accuracy I consider is always the best one obtained during training[4]. Note that the same conclusions would hold considering the model at the end of training.
In Figure 3 we can see the model metrics during the run that obtained the maximum accuracy overall, i.e. using the SWBN-KL method, and the peak of validation accuracy is reached at epoch 135.

[3]BN and SWBN using each criterion in Eq. 2

[4]The best one occurred always after the 130th epoch, so after the last change in the learning rate



(a) Model accuracy



(b) Model loss

Figure 3: Accuracy and loss - over the best run - of the model with SWBN-KL over the training set (blue) and the test set (orange). In these plots the effectiveness of the learning rate scheduling is clear, with a performance gain after each change, i.e. at epoch n. 82 and 122.

Value	BN	SWBN-KL
Max:	93.62	**93.78**
Average:	93.5	**93.67**

Table 2: Maximum and average test accuracy [%] of the ResNetV2-56 model on the CIFAR-10 dataset using standard Batch Normalization and SWBN-KL implemented by me. The metrics are obtained with 5 runs per method.

7 Conclusions

Stochastic Whitening Batch Normalization is an extension of standard Batch Normalization that whitens the data in an online fashion. It improves the generalization capability of the network, as can be seen in the results reported, however it also introduces overhead during training.
In deeper networks, and in few-shots classification problems, it shows a more sensible impact reaching an increase of up to 2 and a half percentage points with respect to the BN baseline in the latter scenario.

References

[1] Michael Cogswell, Faruk Ahmed, Ross B. Girshick, C. Lawrence Zitnick, and Dhruv Batra. Reducing overfitting in deep networks by decorrelating representations. *CoRR*, abs/1511.06068, 2016.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. volume 9908, pages 630–645, 10 2016.

[4] Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. In *International Conference on Learning Representations*, 2022.

[5] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *NeurIPS*, 2018.

[6] Shengdong Zhang, Ehsan Nezhadarya, Homa Fashandi, Jiayi Liu, Darin Graham, and Mohak Shah. Stochastic whitening batch normalization. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10973–10982, 2021.