Batch Normalization and Optimization

Alexey Bauman, Nikolay Skuratov

Project report

1 Abstract

This work is an attempt to reproduce several results from article "How Does Batch Normalization Help Optimization?"[1], by Shibani Santurkar, et. al.

Batch Normalization is a very often used technique that enables faster and more stable training of deep neural networks. Despite its wide use, effectiveness of this technique is still poorly understood. The popular belief is that this effectiveness stems from controlling the change of the layers' input distributions during training to reduce the so-called "internal covariate shift" (ISC). In this work we are going to check this belief and find out that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead,we obtain, that it makes the optimization landscape significantly smoother. It means that behavior of the gradients are more predictive and stable allowing for faster training.

2 Introduction

BatchNorm is a technique that aims to improve the training of neural networks by stabilizing the distributions of layer inputs. This is achieved by introducing additional network layers that control the first two moments (mean and variance) of these distributions. On figures 1 and 2 Batch Norm effect could be seen. There is a consistent gain in training speed in models with BatchNorm layers.

Currently, the most widely accepted explanation of BatchNorm's success, as well as its original motivation, relates to so-called internal covariate shift (ICS). Informally, ICS refers to the change in the distribution of layer inputs caused by updates to the preceding layers. It is conjectured that such continual change negatively impacts training. The goal of BatchNorm was to reduce ICS and thus remedy this effect.

First of all we want to show, that there is no any link between the performance gain of BatchNorm and the reduction of internal covariate shift. Or that this link is tenuous, at best. In fact, we find that in a certain sense BatchNorm might not even be reducing internal covariate shift.

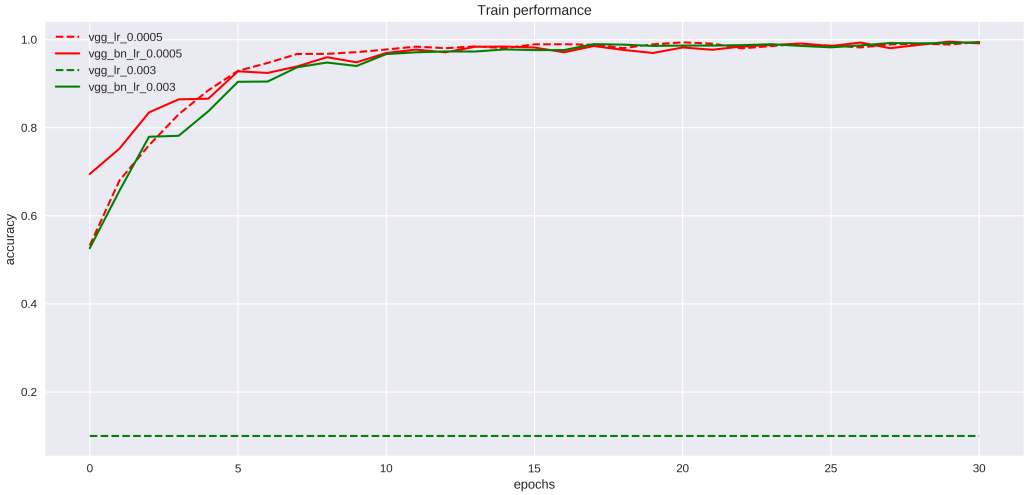

Figure 1: Comparison of training (optimization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm

Then we are trying understand the reason of BatchNorm's success. We figured out that it makes the landscape of the corresponding optimization problem significantly more smooth. So gradients are more predictive and thus allows for use of larger range of learning rates and faster network convergence.

3 Main Part

3.1 Batch normalization and internal covariate shift

Assume, each layer can be seen as solving an empirical risk minimization problem where given a set of inputs, it is optimizing some loss function (that possibly involves later layers). An update to the parameters of any previous layer will change these inputs, thus changing this empirical risk minimization problem itself. Specifically, they try to capture this phenomenon from the perspective of the resulting distributional changes in layer inputs. So we define:

Let L be the loss $W_1^{(t)}, \ldots, W_k^{(t)}$, be the parameters of each of the k layers and $\left(x^{(t)}, y^{(t)}\right)$ be the batch of input-label pairs used to train the network at time t. We define internal covariate shift (ICS) of activation i at time t to be the difference $\left\|G_{t,i} - G'_{t,i}\right\|_2$, where

$$G_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}\left(W_1^{(t)}, \ldots, W_k^{(t)}; x^{(t)}, y^{(t)}\right)$$

$$G'_{t,i} = \nabla_{W_i^{(t)}} \mathcal{L}\left(W_1^{(t+1)}, \ldots, W_{i-1}^{(t+1)}, W_i^{(t)}, W_{i+1}^{(t)}, \ldots, W_k^{(t)}; x^{(t)}, y^{(t)}\right)$$

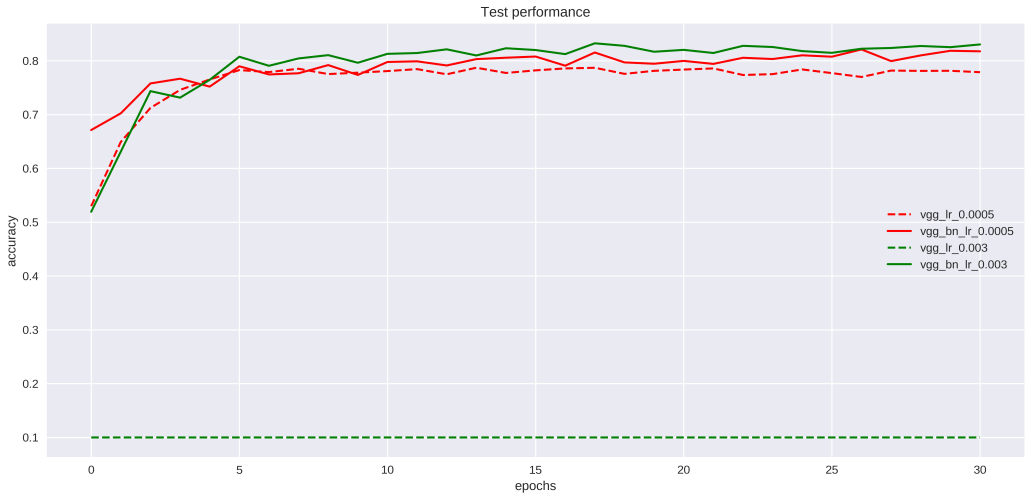

Figure 2: Comparison of test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm

Here, $G_{t,i}$ corresponds to the gradient of the layer parameters that would be applied during a simultaneous update of all layers (as is typical). On the other hand, $G'_{t,i}$ is the same gradient after all the previous layers have been updated with their new values. The difference between G and G' thus reflects the change in the optimization landscape of W_i caused by the changes to its input. It thus captures precisely the effect of cross-layer dependencies that could be problematic for training.

Equipped with this definition, we measure the extent of ICS with and without BatchNorm layers. From here was seen, that networks with BatchNorm often exhibit an increase in ICS. From optimization point of view, controlling the distributions layer inputs as done in BatchNorm, might not even reduce the internal covariate shift. This makes proposition about Batch Normalization and ICS not convenient.

3.2 The smoothing effect of BatchNorm

So what stands behind BatchNorm? After all, in order to understand how batch normalization affects the training performance it would be logical to examine the effect that BatchNorm has on the corresponding optimization landscape. To this end, recall that our training is performed using gradient descent method and this method draws on the first-order optimization paradigm. In this paradigm, we use the local linear approximation of the loss around the current solution to identify the best update step to take. Consequently, the performance of these algorithms is largely determined by how predictive of the nearby loss landscape this local approximation is.

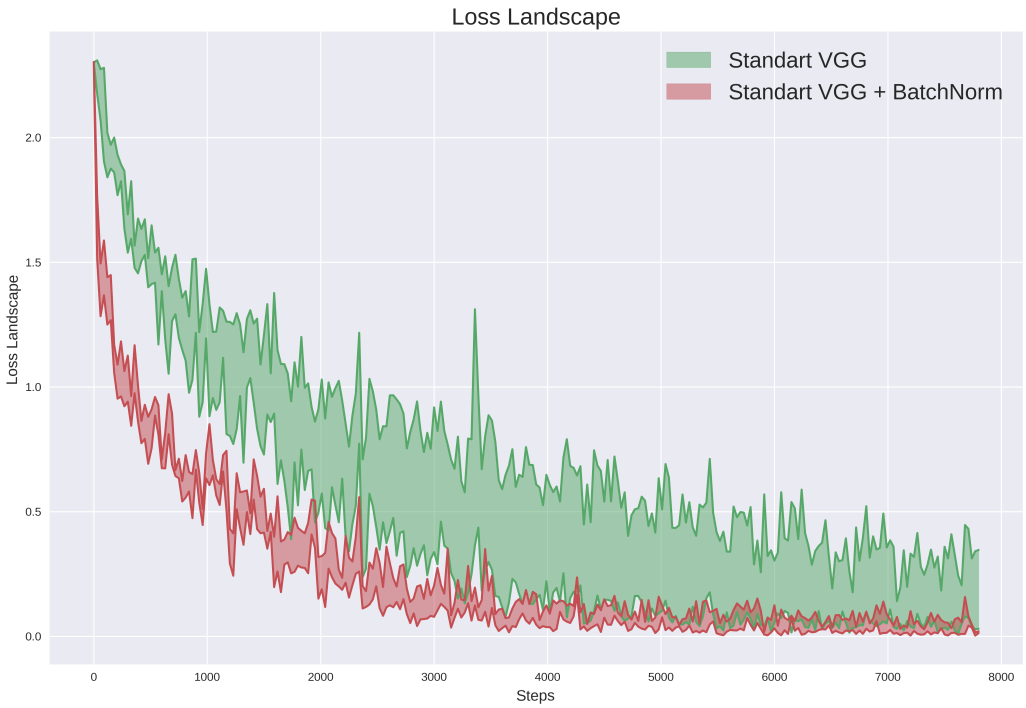

Figure 3: Loss landscape

BatchNorm reparametrizes the underlying optimization problem to make its landscape significantly more smooth. The first manifestation of this impact is improvement in the Lipschitzness of the loss function. That is, the loss changes at a smaller rate and the magnitudes of the gradients are smaller too. There is, however,an even stronger effect at play. Namely, BatchNorm's reparametrization makes gradients of the loss more Lipschitz too. In other words, the loss exhibits a significantly better "effective" β-smoothness.

Finally we mesuared:

1. Loss landscape or variation of the value of the loss (Figure 3);
2. Gradient predictiveness or the change of the loss gradient (Figure 4);
3. Maximum difference in gradient over distance (Figure 5).

Finally we observe, that BatchNorm impacts stability of the loss. We see, that non-BatchNorm network has a very wide range of values along the direction of the gradient, especially in the initial phases of training. Even though later it converges. So networks with BatchNorm indicate that the steps taken during training are unlikely to drive the loss uncontrollably high.

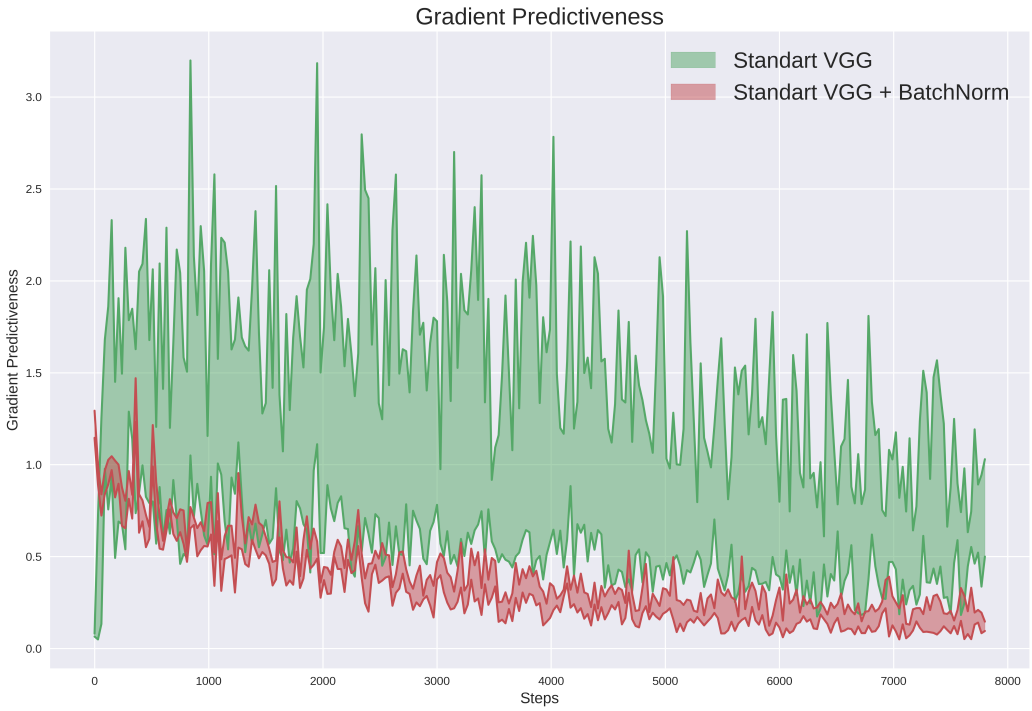

Figure 4: Gradient predictiveness

Also we discovered, that BatchNorm increase stability and predictiveness of the gradients and stability/Lipschitzness of the gradients of the loss. It means good gradient predictiveness implies that the gradient evaluated at a given point stays relevant over longer distances, hence allowing for larger step sizes.

In the end, the "smoothing" effect of BatchNorm makes the optimization landscape much easier to navigate, which could explain the faster convergence and robustness to hyperparameters observed in practice.

Indeed, let's show that BatchNorm effectively reparametrizes the training problem, making it more amenable to first-order methods. Specifically, batch normalization makes the optimization with respect to the activations y easier. This, in turn, translates into improved (worst-case) bounds for the actual optimization problem (which is with respect to the weights W and not the activations y).

Let $g = \nabla_y L$ be the gradient of the loss L with respect to a batch of activations y, and let $\hat{g} = \nabla_y \hat{L}$ be analogously defined for the network with (a single) BatchNorm layer.

Finally:

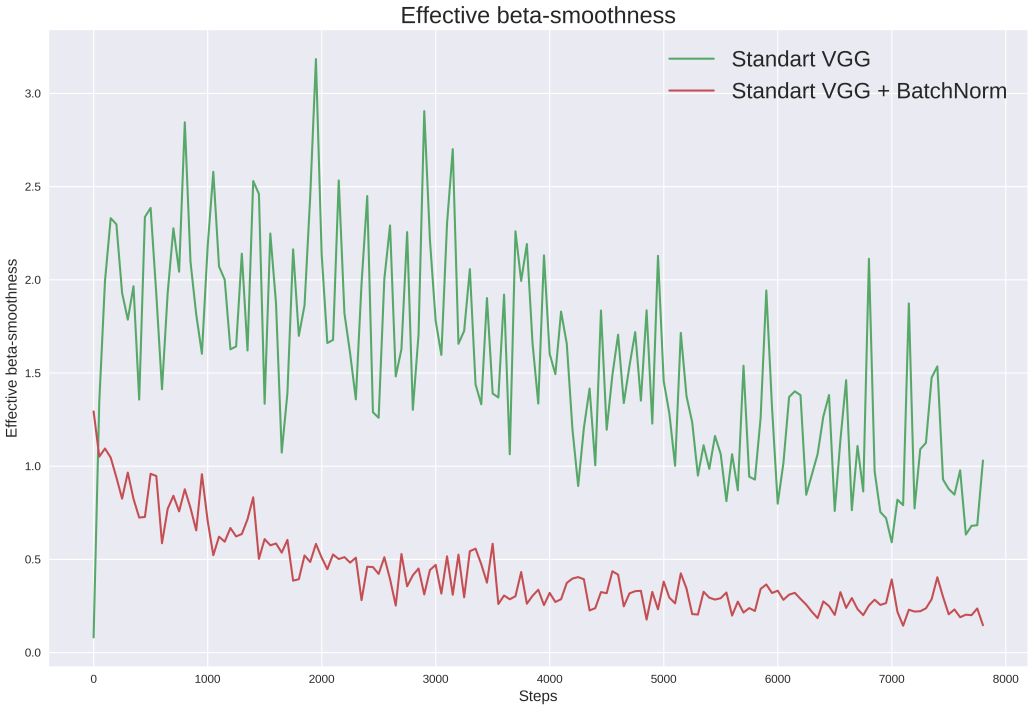

Figure 5: "Effective" β-smoothness

$$\|\hat{g}\|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\|g\|^2 - \mu(g)^2 - \frac{1}{\sqrt{m}}\langle g, \hat{y}\rangle^2\right)$$

So, indeed, inserting the BatchNorm layer reduces the Lipschitz constant of the loss wrt y. Using Taylor expansion we can update obtained bound.

Let H be the Hessian matrix of the loss wrt the a batch of activations y in the standard network and, again, let $\hat{H}$ be defined analogously for the network with (a single) BatchNorm layer inserted.

$$\hat{g}^\top \hat{H} \hat{g} \leq \frac{\gamma^2}{\sigma^2}\left(g^\top H g - \frac{1}{m\gamma}\langle g, \hat{y}\rangle ||\hat{g}\|^2\right)$$

This result can be translates into an analogous bound with respect to W in the setting where the inputs are chosen in a worst-case manner.

4 Conclusion

To conclude, our results are correct in comparison with corresponding experiments of the paper, so we checked possible roots of BatchNorm's effectiveness,

but there is still much to be understood. In particular, while we saw, that it increased smoothness of the optimization landscape as a direct result of employing BatchNorm layers.

5 GIT

Here you can find link for our code:
https://github.com/AlexeyGB/batch-norm-helps-optimization

References

[1] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry *"How Does Batch Normalization Help Optimization?"*, 2018

[2] Sergey Ioffe and Christian Szegedy *"Batch normalization: Accelerating deep network training by reducing internal covariate shift."* , arXiv preprint arXiv:1502.03167, 2015.

[3] David Balduzzi, Marcus Frean, Lennox Leary, JP Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. *"The shattered gradients problem: If resnets are the answer, then what is the question?"* arXiv preprint arXiv:1702.08591, 2017.

[4] Diederik P Kingma and Jimmy Ba *"Adam: A method for stochastic optimization."* arXiv preprint arXiv:1412.6980, 2014.

[5] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. *"On the importance of single directions for generalization."* arXiv preprint arXiv:1803.06959, 2018.

6 Appendix A

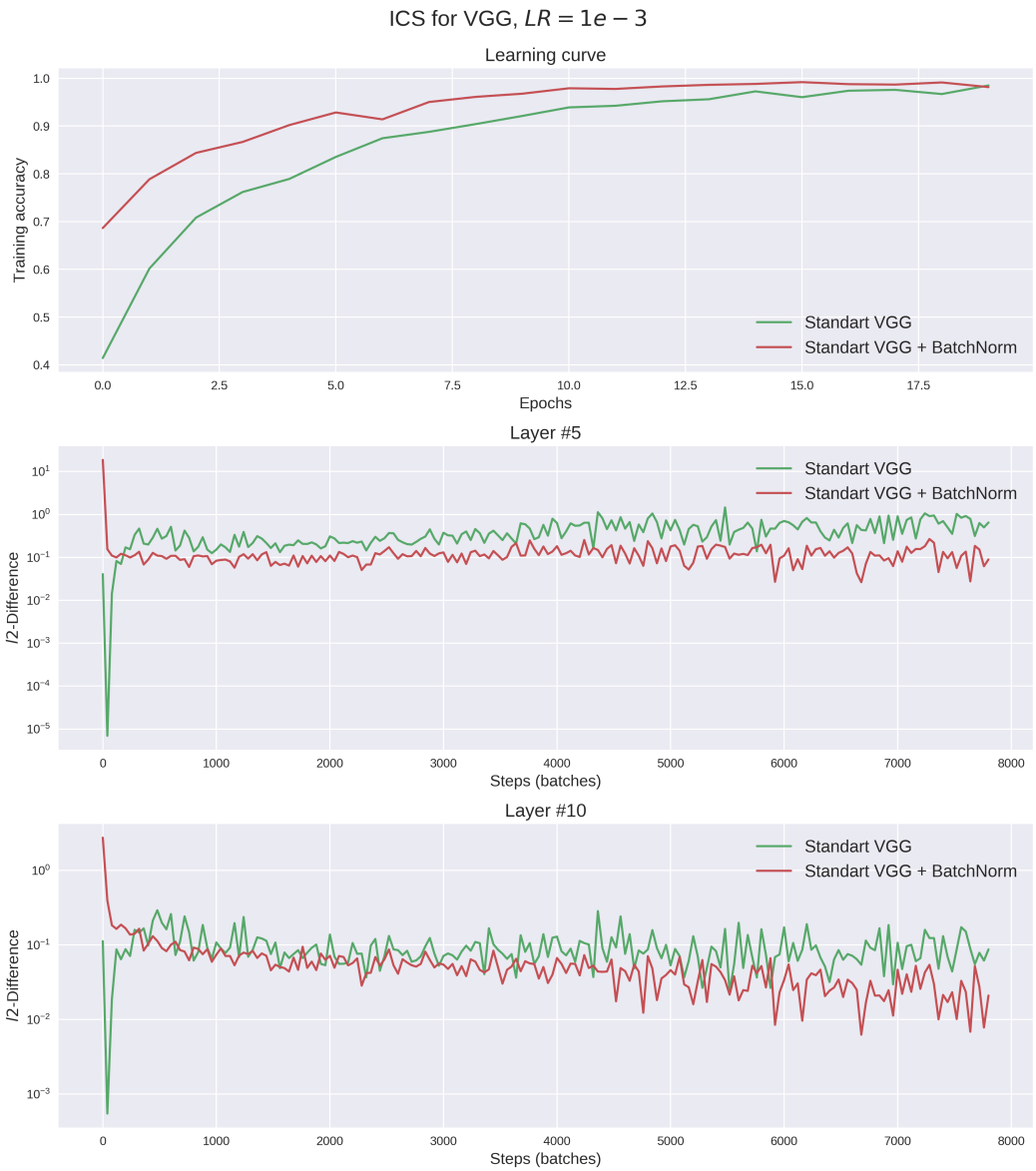

Figure 6: Measurement of ICS (as defined in 3.1) in VGG with and without BatchNorm layers with learning rate $1e-3$.

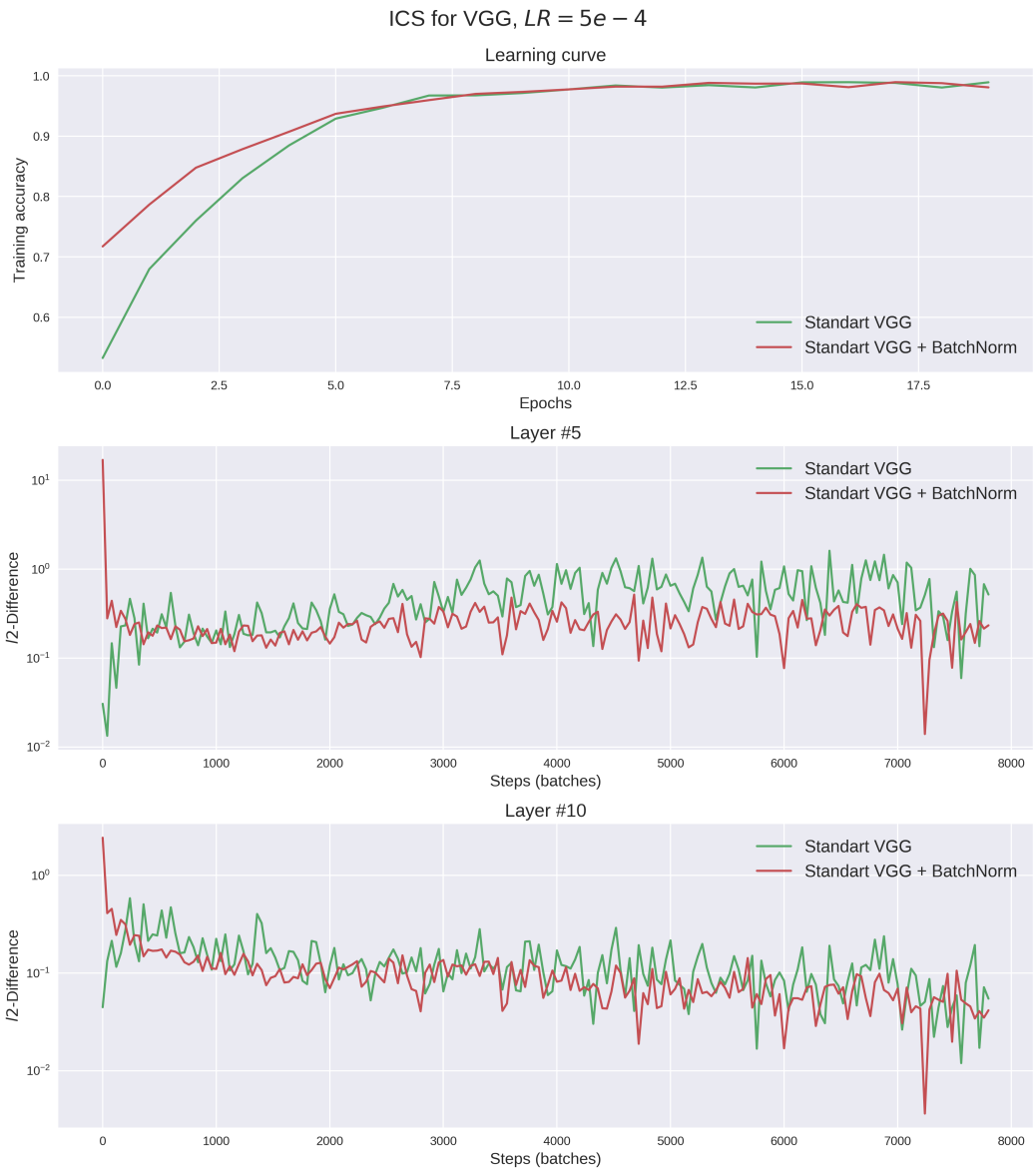

Figure 7: Measurement of ICS (as defined in 3.1) in VGG with and without BatchNorm layers with learning rate $5e-4$.