How Does Batch Normalization Help Optimization?

Shibani Santurkar* MIT shibani@mit.edu
Dimitris Tsipras* MIT tsipras@mit.edu
Andrew Ilyas* MIT ailyas@mit.edu
Aleksander Mądry MIT madry@mit.edu

Abstract

Batch Normalization (BatchNorm) is a widely adopted technique that enables faster and more stable training of deep neural networks (DNNs). Despite its pervasiveness, the exact reasons for BatchNorm's effectiveness are 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". In this work, we demonstrate that such distributional stability of layer inputs has little to do with the success of BatchNorm. Instead, we uncover a more fundamental impact of BatchNorm on the training process: it makes the optimization landscape significantly smoother. This smoothness induces a more predictive and stable behavior of the gradients, allowing for faster training.

1 Introduction

Over the last decade, deep learning has made impressive progress on a variety of notoriously difficult tasks in computer vision [16, 7], speech recognition [5], machine translation [29], and game-playing [18, 25]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 23], and algorithmic and architectural techniques [27, 12, 20, 28]. One of the most prominent examples of such advances was batch normalization (BatchNorm) [10].

At a high level, 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.

The practical success of BatchNorm is indisputable. By now, it is used by default in most deep learning models, both in research (more than 6,000 citations) and real-world settings. Somewhat shockingly, however, despite its prominence, we still have a poor understanding of what the effectiveness of BatchNorm is stemming from. In fact, there are now a number of works that provide alternatives to BatchNorm [1, 3, 13, 31], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [22].)

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.

Even though this explanation is widely accepted, we seem to have little concrete evidence supporting it. In particular, we still do not understand the link between ICS and training performance. The chief goal of this paper is to address all these shortcomings. Our exploration lead to somewhat startling discoveries.

*Equal contribution.

32nd Conference on Neural Information Processing Systems (NeurIPS 2018), Montréal, Canada.

Our Contributions. Our point of start is demonstrating that there does not seem to be 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*.

We then turn our attention to identifying the roots of BatchNorm's success. Specifically, we demonstrate that BatchNorm impacts network training in a fundamental way: it makes the landscape of the corresponding optimization problem *significantly more smooth*. This ensures, in particular, that the gradients are more predictive and thus allows for use of larger range of learning rates and faster network convergence. We provide an empirical demonstration of these findings as well as their theoretical justification. We prove that, under natural conditions, the Lipschitzness of both the loss and the gradients (also known as β-smoothness [21]) are improved in models with BatchNorm.

Finally, we find that this smoothening effect is not uniquely tied to BatchNorm. A number of other natural normalization techniques have a similar (and, sometime, even stronger) effect. In particular, they all offer similar improvements in the training performance.

We believe that understanding the roots of such a fundamental techniques as BatchNorm will let us have a significantly better grasp of the underlying complexities of neural network training and, in turn, will inform further algorithmic progress in this context.

Our paper is organized as follows. In Section 2, we explore the connections between BatchNorm, optimization, and internal covariate shift. Then, in Section 3, we demonstrate and analyze the exact roots of BatchNorm's success in deep neural network training. We present our theoretical analysis in Section 4. We discuss further related work in Section 5 and conclude in Section 6.

2 Batch normalization and internal covariate shift

Batch normalization (BatchNorm) [10] has been arguably one of the most successful architectural innovations in deep learning. But even though its effectiveness is indisputable, we do not have a firm understanding of why this is the case.

Broadly speaking, BatchNorm is a mechanism that aims to stabilize the distribution (over a mini-batch) of inputs to a given network layer during training. This is achieved by augmenting the network with additional layers that set the first two moments (mean and variance) of the distribution of each activation to be zero and one respectively. Then, the batch normalized inputs are also typically scaled and shifted based on trainable parameters to preserve model expressivity. This normalization is applied before the non-linearity of the previous layer.

One of the key motivations for the development of BatchNorm was the reduction of so-called *internal covariate shift* (ICS). This reduction has been widely viewed as the root of BatchNorm's success. Ioffe and Szegedy [10] describe ICS as the phenomenon wherein the distribution of inputs to a layer in the network changes due to an update of parameters of the previous layers. This change leads to a constant shift of the underlying training problem and is thus believed to have detrimental effect on the training process.

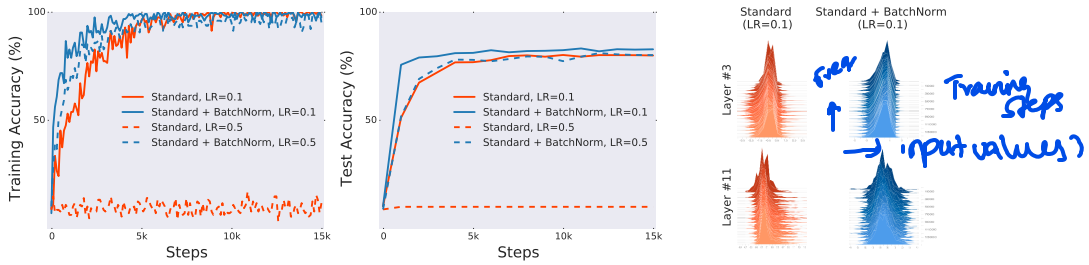


Figure 1: Comparison of (a) training (optimization) and (b) test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm (details in Appendix A). There is a consistent gain in training speed in models with BatchNorm layers. (c) Even though the gap between the performance of the BatchNorm and non-BatchNorm networks is clear, the difference in the evolution of layer input distributions seems to be much less pronounced. (Here, we sampled activations of a given layer and visualized their distribution over training steps.)

Despite its fundamental role and widespread use in deep learning, the underpinnings of BatchNorm's success remain poorly understood [22]. In this work we aim to address this gap. To this end, we start by investigating the connection between ICS and BatchNorm. Specifically, we consider first training a standard VGG [26] architecture on CIFAR-10 [15] with and without BatchNorm. As expected, Figures 1(a) and (b) show a drastic improvement, both in terms of optimization and generalization performance, for networks trained with BatchNorm layers. Figure 1(c) presents, however, a surprising finding. In this figure, we visualize to what extent BatchNorm is stabilizing distributions of layer inputs by plotting the distribution (over a batch) of a random input over training. Surprisingly, the difference in distributional stability (change in the mean and variance) in networks with and without BatchNorm layers seems to be marginal. This observation raises the following questions:

(1) *Is the effectiveness of BatchNorm indeed related to internal covariate shift?*
(2) *Is BatchNorm's stabilization of layer input distributions even effective in reducing ICS?*

We now explore these questions in more depth.

2.1 Does BatchNorm's performance stem from controlling internal covariate shift?

The central claim in [10] is that controlling the mean and variance of distributions of layer inputs is directly connected to improved training performance. Can we, however, substantiate this claim?

We propose the following experiment. We train networks with *random* noise injected *after* BatchNorm layers. Specifically, we perturb each activation for each sample in the batch using i.i.d. noise sampled from a *non-zero* mean and *non-unit* variance distribution. We emphasize that this noise distribution *changes* at each time step (see Appendix A for implementation details).

Note that such noise injection produces a severe covariate shift that skews activations at every time step. Consequently, every unit in the layer experiences a *different* distribution of inputs at *each* time step. We then measure the effect of this deliberately introduced distributional instability on BatchNorm's performance. Figure 2 visualizes the training behavior of standard, BatchNorm and our "noisy" BatchNorm networks. Distributions of activations over time from layers at the same depth in each one of the three networks are shown alongside.

Observe that the performance difference between models with BatchNorm layers, and "noisy" BatchNorm layers is almost non-existent. Also, both these networks perform much better than standard networks. Moreover, the "noisy" BatchNorm network has qualitatively *less stable* distributions than even the standard, non-BatchNorm network, yet it *still performs better* in terms of training. To put

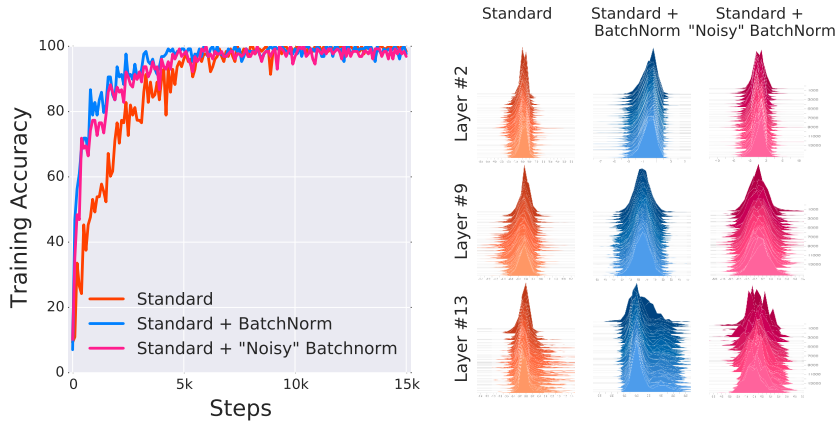
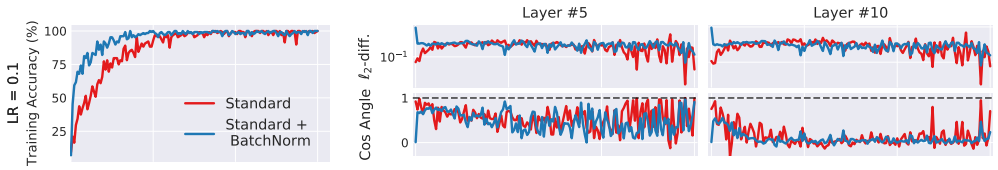
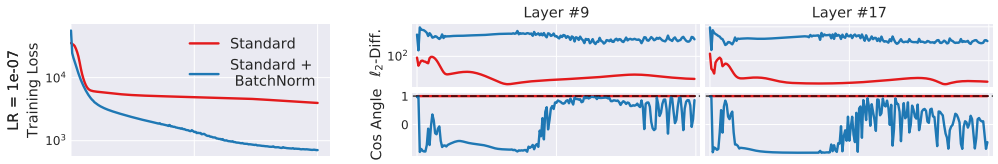


Figure 2: Connections between distributional stability and BatchNorm performance: We compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). In the later case, we induce distributional instability 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. We sampled activations of a given layer and visualized their distributions (also cf. Figure 7).



(a) VGG



(b) DLN

Figure 3: Measurement of ICS (as defined in Definition 2.1) in networks with and without BatchNorm layers. For a layer we measure the cosine angle (ideally 1) and ℓ_2-difference of the gradients (ideally 0) before and after updates to the preceding layers (see Definition 2.1). Models with BatchNorm have similar, or even worse, internal covariate shift, despite performing better in terms of accuracy and loss. (Stabilization of BatchNorm faster during training is an artifact of parameter convergence.)

the magnitude of the noise into perspective, we plot the mean and variance of random activations for select layers in Figure 7. Moreover, adding the same amount of noise to the activations of the standard (non-BatchNorm) network prevents it from training entirely.

Clearly, these findings are hard to reconcile with the claim that the performance gain due to BatchNorm stems from increased stability of layer input distributions.

2.2 Is BatchNorm reducing internal covariate shift?

Our findings in Section 2.1 make it apparent that ICS is not directly connected to the training performance, at least if we tie ICS to stability of the mean and variance of input distributions. One might wonder, however: Is there a broader notion of internal covariate shift that *has* such a direct link to training performance? And if so, does BatchNorm indeed reduce this notion?

Recall that 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. This phenomenon is at the core of the intuition that Ioffe and Szegedy [10] provide regarding internal covariate shift. Specifically, they try to capture this phenomenon from the perspective of the resulting *distributional* changes in layer inputs. However, as demonstrated in Section 2.1, this perspective does not seem to properly encapsulate the roots of BatchNorm's success.

To answer this question, we consider a broader notion of internal covariate shift that is more tied to the underlying optimization task. (After all the success of BatchNorm is largely of an optimization nature.) Since the training procedure is a first-order method, the gradient of the loss is the most natural object to study. To quantify the extent to which the parameters in a layer would have to "adjust" in reaction to a parameter update in the previous layers, we measure the difference between the gradients of each layer before and after updates to all the previous layers. This leads to the following definition.

Definition 2.1. *Let $\mathcal{L}$ be the loss, $W_1^{(t)}, \ldots, W_k^{(t)}$ be the parameters of each of the k layers and $(x^{(t)}, y^{(t)})$ 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 $||G_{t,i} - G'_{t,i}||_2$, where*

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

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

(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 4: Analysis of the optimization landscape of VGG networks. At a particular training step, we measure the variation (shaded region) in loss (a) and ℓ_2 changes in the gradient (b) as we move in the gradient direction. The "effective" β-smoothness (c) refers to the maximum difference (in ℓ_2-norm) in gradient over distance moved in that direction. There is a clear improvement in all of these measures in networks with BatchNorm, indicating a more well-behaved loss landscape. (Here, we cap the maximum distance to be $\eta = 0.4\times$ the gradient since for larger steps the standard network just performs worse (see Figure 1). BatchNorm however continues to provide smoothing for even larger distances.) Note that these results are supported by our theoretical findings (Section 4).

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. To isolate the effect of non-linearities as well as gradient stochasticity, we also perform this analysis on (25-layer) deep linear networks (DLN) trained with full-batch gradient descent (see Appendix A for details). The conventional understanding of BatchNorm suggests that the addition of BatchNorm layers in the network should increase the correlation between G and G', thereby reducing ICS.

Surprisingly, we observe that networks with BatchNorm often exhibit an *increase* in ICS (cf. Figure 3). This is particularly striking in the case of DLN. In fact, in this case, the standard network experiences almost no ICS for the entirety of training, whereas for BatchNorm it appears that G and G' are almost uncorrelated. We emphasize that this is the case *even though BatchNorm networks continue to perform drastically better* in terms of attained accuracy and loss. (The stabilization of the BatchNorm VGG network later in training is an artifact of faster convergence.) This evidence suggests that, from optimization point of view BatchNorm might not even reduce the internal covariate shift.

3 Why does BatchNorm work?

Our investigation so far demonstrated that the generally asserted link between the internal covariate shift (ICS) and the optimization performance is tenuous, at best. But BatchNorm *does* significantly improve the training process. Can we explain why this is the case?

Aside from reducing ICS, Ioffe and Szegedy [10] identify a number of additional properties of BatchNorm. These include prevention of exploding or vanishing gradients, robustness to different settings of hyperparameters such as learning rate and initialization scheme, and keeping most of the activations away from saturation regions of non-linearities. All these properties are clearly beneficial to the training process. But they are fairly simple consequences of the mechanics of BatchNorm and do little to uncover the underlying factors responsible for BatchNorm's success. *Is there a more fundamental phenomenon at play here?*

3.1 The smoothing effect of BatchNorm

Indeed, we identify the key impact that BatchNorm has on the training process: it reparametrizes the underlying optimization problem to *make its landscape significantly more smooth*. The first manifestation of this impact is improvement in the Lipschitzness[2] 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,

[2]Recall that f is L-Lipschitz if $|f(x_1) - f(x_2)| \leq L\|x_1 - x_2\|$, for all x_1 and x_2.

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[3].

These smoothening effects impact the performance of the training algorithm in a major way. To understand why, recall that in a vanilla (non-BatchNorm), deep neural network, the loss function is not only non-convex but also tends to have a large number of "kinks", flat regions, and sharp minima [17]. This makes gradient descent–based training algorithms unstable, e.g., due to exploding or vanishing gradients, and thus highly sensitive to the choice of the learning rate and initialization.

Now, the key implication of BatchNorm's reparametrization is that it makes the gradients more reliable and predictive. After all, improved Lipschitzness of the gradients gives us confidence that when we take a larger step in a direction of a computed gradient, this gradient direction remains a fairly accurate estimate of the actual gradient direction after taking that step. It thus enables any (gradient–based) training algorithm to take larger steps without the danger of running into a sudden change of the loss landscape such as flat region (corresponding to vanishing gradient) or sharp local minimum (causing exploding gradients). This, in turn, enables us to use a broader range of (and thus larger) learning rates (see Figure 10 in Appendix B) and, in general, makes the training significantly faster and less sensitive to hyperparameter choices. (This also illustrates how the properties of BatchNorm that we discussed earlier can be viewed as a manifestation of this smoothening effect.)

3.2 Exploration of the optimization landscape

To demonstrate the impact of BatchNorm on the stability of the loss itself, i.e., its Lipschitzness, for each given step in the training process, we compute the gradient of the loss at that step and measure how the loss changes as we move in that direction – see Figure 4(a). We see that, in contrast to the case when BatchNorm is in use, the loss of a vanilla, i.e., non-BatchNorm, network has a very wide range of values along the direction of the gradient, especially in the initial phases of training. (In the later stages, the network is already close to convergence.)

Similarly, to illustrate the increase in the stability and predictiveness of the gradients, we make analogous measurements for the ℓ_2 distance between the loss gradient at a given point of the training and the gradients corresponding to different points along the original gradient direction. Figure 4(b) shows a significant difference (close to two orders of magnitude) in such gradient predictiveness between the vanilla and BatchNorm networks, especially early in training.

To further demonstrate the effect of BatchNorm on the stability/Lipschitzness of the gradients of the loss, we plot in Figure 4(c) the "effective" β-smoothness of the vanilla and BatchNorm networks throughout the training. ("Effective" refers here to measuring the change of gradients as we move in the direction of the gradients.). Again, we observe consistent differences between these networks. We complement the above examination by considering *linear* deep networks: as shown in Figures 9 and 12 in Appendix B, the BatchNorm smoothening effect is present there as well.

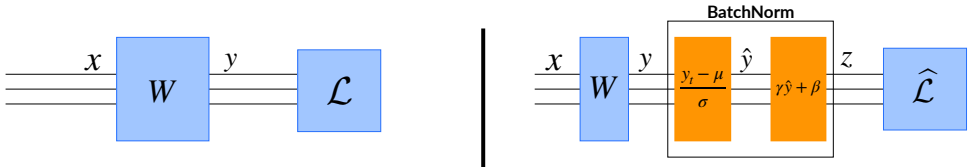
Finally, we emphasize that even though our explorations were focused on the behavior of the loss along the gradient directions (as they are the crucial ones from the point of view of the training process), the loss behaves in a similar way when we examine other (random) directions too.

3.3 Is BatchNorm the best (only?) way to smoothen the landscape?

Given this newly acquired understanding of BatchNorm and the roots of its effectiveness, it is natural to wonder: *Is this smoothening effect a unique feature of BatchNorm?* Or could a similar effect be achieved using some other normalization schemes?

To answer this question, we study a few natural data statistics-based normalization strategies. Specifically, we study schemes that fix the first order moment of the activations, as BatchNorm does, and then normalizes them by the average of their ℓ_p-norm (*before* shifting the mean), for $p = 1, 2, \infty$. Note that for these normalization schemes, the distributions of layer inputs are no longer Gaussian-like (see Figure 14). Hence, normalization with such ℓ_p-norm does not guarantee anymore any control over the distribution moments nor distributional stability.

[3]Recall that f is β-smooth if its gradient is β-Lipschitz. It is worth noting that, due to the existence of non-linearities, one should not expect the β-smoothness to be bounded in an absolute, global sense.



(a) Vanilla Network

(b) Vanilla Network + BatchNorm Layer

Figure 5: The two network architectures we compare in our theoretical analysis: (a) the vanilla DNN (no BatchNorm layer); (b) the same network as in (a) but with a BatchNorm layer inserted after the fully-connected layer W. (All the layer parameters have exactly the same value in both networks.)

The results are presented in Figures 13, 11 and 12 in Appendix B. We observe that all the normalization strategies offer comparable performance to BatchNorm. In fact, for deep linear networks, ℓ_1–normalization performs even better than BatchNorm. Note that, qualitatively, the ℓ_p–normalization techniques lead to *larger distributional shift* (as considered in [10]) than the vanilla, i.e., unnormalized, networks, yet they still *yield improved optimization performance*. Also, all these techniques result in an improved smoothness of the landscape that is similar to the effect of BatchNorm. (See Figures 11 and 12 of Appendix B.) This suggests that the positive impact of BatchNorm on training might be somewhat serendipitous. Therefore, it might be valuable to perform a principled exploration of the design space of normalization schemes as it can lead to better performance.

4 Theoretical Analysis

Our experiments so far suggest that BatchNorm has a fundamental effect on the optimization landscape. We now explore this phenomenon from a theoretical perspective. To this end, we consider an arbitrary linear layer in a DNN (we do not necessitate that the entire network be fully linear).

4.1 Setup

We analyze the impact of adding a single BatchNorm layer after an arbitrary fully-connected layer W at a given step during the training. Specifically, we compare the optimization landscape of the original training problem to the one that results from inserting the BatchNorm layer *after* the fully-connected layer – normalizing the output of this layer (see Figure 5). Our analysis therefore captures effects that stem from the reparametrization of the landscape and not merely from normalizing the inputs x.

We denote the layer weights (identical for both the standard and batch-normalized networks) as W_{ij}. Both networks have the same arbitrary loss function $\mathcal{L}$ that could potentially include a number of additional non-linear layers after the current one. We refer to the loss of the normalized network as $\widehat{\mathcal{L}}$ for clarity. In both networks, we have input x, and let $y = Wx$. For networks with BatchNorm, we have an additional set of activations $\hat{y}$, which are the "whitened" version of y, i.e. standardized to mean 0 and variance 1. These are then multiplied by γ and added to β to form z. We assume β and γ to be constants for our analysis. In terms of notation, we let σ_j denote the standard deviation (computed over the mini-batch) of a batch of outputs $y_j \in \mathbb{R}^m$.

4.2 Theoretical Results

We begin by considering the optimization landscape with respect to the activations $\boldsymbol{y_j}$. We show that batch normalization causes this landscape to be more well-behaved, inducing favourable properties in Lipschitz-continuity, and predictability of the gradients. We then show that these improvements in the activation-space landscape translate to favorable worst-case bounds in the weight-space landscape.

We first turn our attention to the gradient magnitude $\left|\left|\nabla_{\boldsymbol{y_j}} \mathcal{L}\right|\right|$, which captures the Lipschitzness of the loss. The Lipschitz constant of the loss plays a crucial role in optimization, since it controls the amount by which the loss can change when taking a step (see [21] for details). Without any assumptions on the specific weights or the loss being used, we show that the batch-normalized

landscape exhibits a better Lipschitz constant. Moreover, the Lipschitz constant is significantly reduced whenever the activations $\hat{\boldsymbol{y}}_{\boldsymbol{j}}$ correlate with the gradient $\nabla_{\hat{\boldsymbol{y}}_j}\widehat{\mathcal{L}}$ or the mean of the gradient deviates from 0. Note that this reduction is additive, and has effect even when the scaling of BN is identical to the original layer scaling (i.e. even when $\sigma_j = \gamma$).

Theorem 4.1 (The effect of BatchNorm on the Lipschitzness of the loss)**.** *For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,*

$$\left|\left|\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right|\right|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\left|\left|\nabla_{\boldsymbol{y}_j}\mathcal{L}\right|\right|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{\boldsymbol{y}_j}\mathcal{L}\right\rangle^2 - \frac{1}{m}\left\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).$$

First, note that $\langle \mathbf{1}, \partial L/\partial y\rangle^2$ grows quadratically in the dimension, so the middle term above is significant. Furthermore, the final inner product term is expected to be bounded away from zero, as the gradient with respect to a variable is rarely uncorrelated to the variable itself. In addition to the additive reduction, σ_j tends to be large in practice (cf. Appendix Figure 8), and thus the scaling by $\frac{\gamma}{\sigma}$ may contribute to the relative "flatness" we see in the effective Lipschitz constant.

We now turn our attention to the second-order properties of the landscape. We show that when a BatchNorm layer is added, the quadratic form of the loss Hessian with respect to the activations in the gradient direction, is both rescaled by the input variance (inducing resilience to mini-batch variance), and decreased by an additive factor (increasing smoothness). This term captures the second order term of the Taylor expansion of the gradient around the current point. Therefore, reducing this term implies that the first order term (the gradient) is more predictive.

Theorem 4.2 (The effect of BN to smoothness)**.** *Let $\hat{\boldsymbol{g}}_{\boldsymbol{j}} = \nabla_{\boldsymbol{y}_j}\mathcal{L}$ and $\boldsymbol{H}_{jj} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}$ be the gradient and Hessian of the loss with respect to the layer outputs respectively. Then*

$$\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj}\left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2}\left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle \left|\left|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right|\right|^2$$

If we also have that the $\boldsymbol{H}_{jj}$ preserves the relative norms of $\hat{\boldsymbol{g}}_{\boldsymbol{j}}$ and $\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}$,

$$\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}\left(\nabla_{\boldsymbol{y}_j}\widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2}\left(\hat{\boldsymbol{g}}_{\boldsymbol{j}}^\top \boldsymbol{H}_{jj}\hat{\boldsymbol{g}}_{\boldsymbol{j}} - \frac{1}{m\gamma}\left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle \left|\left|\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right|\right|^2\right)$$

Note that if the quadratic forms involving the Hessian and the inner product $\langle \hat{\boldsymbol{y}}_j, \hat{\boldsymbol{g}}_{\boldsymbol{j}}\rangle$ are non-negative (both fairly mild assumptions), the theorem implies more predictive gradients. The Hessian is positive semi-definite (PSD) if the loss is locally convex which is true for the case of deep networks with piecewise-linear activation functions and a convex loss at the final layer (e.g. standard softmax cross-entropy loss or other common losses). The condition $\langle \hat{\boldsymbol{y}}_j, \hat{\boldsymbol{g}}_{\boldsymbol{j}}\rangle > 0$ holds as long as the negative gradient $\hat{\boldsymbol{g}}_{\boldsymbol{j}}$ is pointing towards the minimum of the loss (w.r.t. normalized activations). Overall, as long as these two conditions hold, the steps taken by the BatchNorm network are more predictive than those of the standard network (similarly to what we observed experimentally).

Note that our results stem from the reparametrization of the problem and not a simple scaling.

Observation 4.3 (BatchNorm does more than rescaling)**.** *For any input data X and network configuration W, there exists a BN configuration (W, γ, β) that results in the same activations y_j, and where $\gamma = \sigma_j$. Consequently, all of the minima of the normal landscape are preserved in the BN landscape.*

Our theoretical analysis so far studied the optimization landscape of the loss w.r.t. the normalized activations. We will now translate these bounds to a favorable worst-case bound on the landscape with respect to layer weights. Note that a (near exact) analogue of this theorem for minimax gradient predictiveness appears in Theorem C.1 of Appendix C.

Theorem 4.4 (Minimax bound on weight-space Lipschitzness)**.** *For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network (with identical loss $\mathcal{L}$), if*

$$g_j = \max_{||X||\leq\lambda}||\nabla_W\mathcal{L}||^2, \qquad \hat{g}_j = \max_{||X||\leq\lambda}\left|\left|\nabla_W\widehat{\mathcal{L}}\right|\right|^2 \implies \hat{g}_j \leq \frac{\gamma^2}{\sigma_j^2}\left(g_j^2 - m\mu_{g_j}^2 - \lambda^2\left\langle \nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).$$

Finally, in addition to a desirable landscape, we find that BN also offers an advantage in initialization:

Lemma 4.5 (BatchNorm leads to a favourable initialization)**.** *Let $\boldsymbol{W^*}$ and $\widehat{\boldsymbol{W}}^*$ be the set of local optima for the weights in the normal and BN networks, respectively. For any initialization W_0*

$$\left|\left|W_0 - \widehat{W}^*\right|\right|^2 \leq ||W_0 - W^*||^2 - \frac{1}{||W^*||^2}\left(||W^*||^2 - \langle W^*, W_0\rangle\right)^2,$$

if $\langle W_0, W^*\rangle > 0$, where $\widehat{W}^*$ and W^* are closest optima for BN and standard network, respectively.

5 Related work

A number of normalization schemes have been proposed as alternatives to BatchNorm, including normalization over layers [1], subsets of the batch [31], or across image dimensions [30]. Weight Normalization [24] follows a complementary approach normalizing the weights instead of the activations. Finally, ELU [3] and SELU [13] are two proposed examples of non-linearities that have a progressively decaying slope instead of a sharp saturation and can be used as an alternative for BatchNorm. These techniques offer an improvement over standard training that is comparable to that of BatchNorm but do not attempt to explain BatchNorm's success.

Additionally, work on topics related to DNN optimization has uncovered a number of other BatchNorm benefits. Li et al. [9] observe that networks with BatchNorm tend to have optimization trajectories that rely less on the parameter initialization. Balduzzi et al. [2] observe that models without BatchNorm tend to suffer from small correlation between different gradient coordinates and/or unit activations. They report that this behavior is profound in deeper models and argue how it constitutes an obstacle to DNN optimization. Morcos et al. [19] focus on the generalization properties of DNN. They observe that the use of BatchNorm results in models that rely less on single directions in the activation space, which they find to be connected to the generalization properties of the model.

Recent work [14] identifies simple, concrete settings where a variant of training with BatchNorm provably improves over standard training algorithms. The main idea is that decoupling the length and direction of the weights (as done in BatchNorm and Weight Normalization [24]) can be exploited to a large extent. By designing algorithms that optimize these parameters separately, with (different) adaptive step sizes, one can achieve significantly faster convergence rates for these problems.

6 Conclusions

In this work, we have investigated the roots of BatchNorm's effectiveness as a technique for training deep neural networks. We find that the widely believed connection between the performance of BatchNorm and the internal covariate shift is tenuous, at best. In particular, we demonstrate that existence of internal covariate shift, at least when viewed from the – generally adopted – distributional stability perspective, is *not* a good predictor of training performance. Also, we show that, from an optimization viewpoint, BatchNorm might not be even reducing that shift.

Instead, we identify a key effect that BatchNorm has on the training process: it reparametrizes the underlying optimization problem to make it more stable (in the sense of loss Lipschitzness) and smooth (in the sense of "effective" β-smoothness of the loss). This implies that the gradients used in training are more predictive and well-behaved, which enables faster and more effective optimization. This phenomena also explains and subsumes some of the other previously observed benefits of BatchNorm, such as robustness to hyperparameter setting and avoiding gradient explosion/vanishing. We also show that this smoothing effect is not unique to BatchNorm. In fact, several other natural normalization strategies have similar impact and result in a comparable performance gain.

We believe that these findings not only challenge the conventional wisdom about BatchNorm but also bring us closer to a better understanding of this technique. We also view these results as an opportunity to encourage the community to pursue a more systematic investigation of the algorithmic toolkit of deep learning and the underpinnings of its effectiveness.

Finally, our focus here was on the impact of BatchNorm on training but our findings might also shed some light on the BatchNorm's tendency to improve generalization. Specifically, it could be the case that the smoothening effect of BatchNorm's reparametrization encourages the training process to converge to more flat minima. Such minima are believed to facilitate better generalization [8, 11]. We hope that future work will investigate this intriguing possibility.

Acknowledgements

We thank Ali Rahimi and Ben Recht for helpful comments on a preliminary version of this paper.

Shibani Santurkar was supported by the National Science Foundation (NSF) under grants IIS-1447786, IIS-1607189, and CCF-1563880, and the Intel Corporation. Dimitris Tsipras was supported in part by the NSF grant CCF-1553428 and the NSF Frontier grant CNS-1413920. Andrew Ilyas was supported in part by NSF awards CCF-1617730 and IIS-1741137, a Simons Investigator Award, a Google Faculty Research Award, and an MIT-IBM Watson AI Lab research grant. Aleksander Mądry was supported in part by an Alfred P. Sloan Research Fellowship, a Google Research Award, and the NSF grants CCF-1553428 and CNS-1815221.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. In *arXiv preprint arXiv:1607.06450*, 2016.

[2] 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? *International Conference on Machine Learning (ICML)*, 2017.

[3] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deepnetwork learning by exponential linear units (elus). In *International Conference on Learning Representations (ICLR)*, 2016.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[5] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *International Conference on Acoustics, Speech, and Signal Processing (IEEE-ICASSP)*, 2013.

[6] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. In *International Conference on Learning Representations (ICLR)*, 2017.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 1997.

[9] Daniel Jiwoong Im, Michael Tao, and Kristin Branson. An empirical analysis of deep network loss surfaces. *arXiv preprint arXiv:1612.04010*, 2016.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, 2015.

[11] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations (ICLR)*, 2017.

[12] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

[13] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[14] Jonas Kohler, Hadi Daneshmand, Aurelien Lucchi, Ming Zhou, Klaus Neymeyr, and Thomas Hofmann. Towards a theoretical understanding of batch normalization. *arXiv preprint arXiv:1805.10694*, 2018.

[15] Alex Krizhevsky. Learning multiple layers of features from tiny images. In *Technical report*, 2009.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

[17] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.

[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 2015.

[19] Ari S Morcos, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. On the importance of single directions for generalization. In *International Conference on Learning Representations (ICLR)*, 2018.

[20] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *International Conference on Machine Learning (ICML)*, 2010.

[21] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Springer Science & Business Media, 2014.

[22] Ali Rahimi and Ben Recht. Back when we were kids. *NIPS Test of Time Award*, 2017.

[23] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. In *International Journal of Computer Vision (IJCV)*, 2015.

[24] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[25] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[26] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[27] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014.

[28] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the Importance of Initialization and Momentum in Deep Learning. In *International Conference on Acoustics, Speech and Signal Processing (IEEE-ICASSP)*, 2013.

[29] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2014.

[30] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[31] Yuxin Wu and Kaiming He. Group normalization. In *European Conference on Computer Vision (ECCV)*, 2018.

A Experimental Setup

In Section A.1, we provide details regarding the architectures used in our analysis. Then in Section A.2 we discuss the specifics of the setup and measurements used in our experiments.

A.1 Models

We use two standard deep architectures – a VGG-like network, and a deep *linear* network (DLN). The VGG model achieves close to state-of-the-art performance while being fairly simple[4]. Preliminary experiments on other architectures gave similar results. We study DLNs with full-batch training since they allow us to isolate the effect of non-linearities, as well as the stochasticity of the training procedure. Both these architectures show clear a performance benefits with BatchNorm.

Specific details regarding both architectures are provided below:

1. **Convolutional VGG architecture on CIFAR10 (VGG):**

   We fit a VGG-like network, a standard convolutional architecture [26], to a canonical image classification problem (CIFAR10 [15]). We optimize using standard stochastic gradient descent and train for $15,000$ steps (training accuracy plateaus). We use a batch size of 128 and a fixed learning rate of 0.1 unless otherwise specified. Moreover, since our focus is on training, we do not use data augmentation. This architecture can fit the training dataset well and achieves close to state-of-the art test performance. Our network achieves a test accuracy of 83% with BatchNorm and 80% without (this becomes 92% and 88% respectively with data augmentation).

2. **25-Layer Deep Linear Network on Synthetic Gaussian Data (DLN):**

   DLN are a factorized approach to solving a simple regression problem, i.e., fitting Ax from x. Specifically, we consider a deep network with k fully connected layers and an ℓ_2 loss. Thus, we are minimizing $\|W_1 \dots W_k x - Ax\|_2^2$ over W_i [5]. We generate inputs x from a Gaussian Distribution and a matrix A with i.i.d. Gaussian entries. We choose k to be 25, and the dimensions of A to be 10×10. All the matrices W_i are square and have the same dimensions. We train DLN using full-batch gradient descent for $10,000$ steps (training loss plateaus).The size of the dataset is 1000 (same as the batch size) and the learning rate is 10^{-6} unless otherwise specified.

For both networks we use standard Glorot initialization [4]. Further the learning rates were selected based on hyperparameter optimization to find a configuration where the training performance for the network was the best.

A.2 Details

A.2.1 "Noisy" BatchNorm Layers

Consider $a_{i,j}$, the j-th activation of the i-th example in the batch. Note that batch norm will ensure that the distribution of $a_{\cdot,j}$ for some j will have fixed mean and variance (possibly learnable).

At every time step, our noise model consists of perturbing each activation for each sample in a batch with noise i.i.d. from a non-zero mean, non-unit variance distribution D_j^t. The distribution D_j^t itself is time varying and its parameters are drawn i.i.d from another distribution D_j. The specific noise model is described in Algorithm 1. In our experiments, $n_\mu = 0.5$, $n_\sigma = 1.25$ and $r_\mu = r_\sigma = 0.1$. (For convolutional layers, we follow the standard convention of treating the height and width dimensions as part of the batch.)

[4] We choose to not experiment with ResNets [7] since they seem to provide several similar benefits to BatchNorm [6] and would introduce conflating factors into our study.

[5] While the factorized formulation is equivalent to a single matrix in terms of expressivity, the optimization landscape is drastically different [6].

Algorithm 1 "Noisy" BatchNorm

% **For constants** n_m**,** n_v**,** r_m**,** r_v**.**

for each layer at time t **do**

  $a^t_{i,j} \leftarrow$ *Batch-normalized activation for unit j and sample i*

  for each j **do** ▷ Sample the parameters (m^t_j, v^t_j) of D^t_j from D_j

    $\mu^t \sim U(-n_\mu, n_\mu)$

    $\sigma^t \sim U(1, n_\sigma)$

  for each i **do** ▷ Sample noise from D^t_j

    for each j **do**

      $m^t_{i,j} \sim U(\mu - r_\mu, \mu + r_\mu)$

      $s^t_{i,j} \sim \mathcal{N}(\sigma, r_\sigma)$

      $a^t_{i,j} \leftarrow s^t_{i,j} \cdot a_{i,j} + m^t_{i,j}$

While plotting the distribution of activations, we sample random activations from any given layer of the network and plot its distribution over the batch dimension for fully connected layers, and over the batch, height, width dimension for convolutional layers as is standard convention in BatchNorm for convolutional networks.

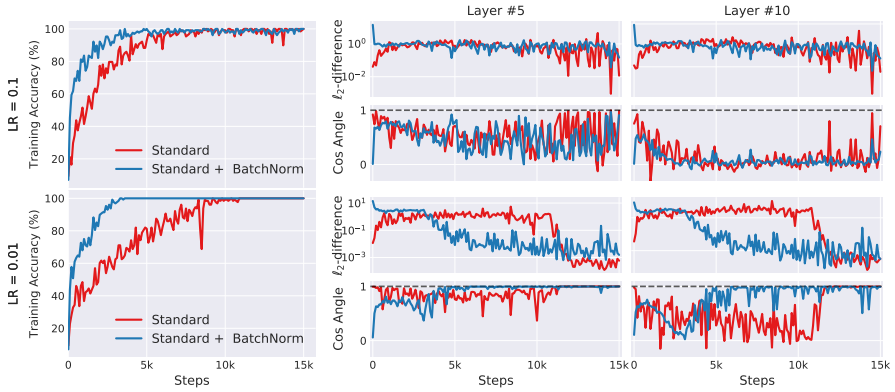
A.2.2 Loss Landscape

To measure the smoothness of the loss landscape of a network during the course of training, we essentially take steps of different lengths in the direction of the gradient and measure the loss values obtained at each step. Note that this is not a training procedure, but an evaluation of the local loss landscape at every step of the training process.

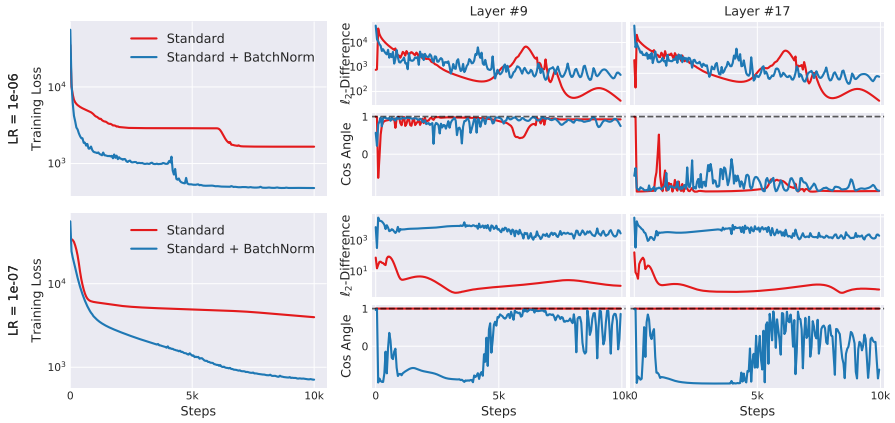
For VGG we consider steps of length ranging from $[1/2, 4] \times$ *step size*, whereas for DLN we choose $[1/100, 30] \times$ *step size*. Here *step size* denotes the hyperparameter setting with which the network is being trained. We choose these ranges to roughly reflect the range of parameters that are valid for standard training of these models. The VGG network is much more sensitive to the learning rate choices (probably due to the non-linearities it includes), so we perform line search over a restricted range of parameters. Further, the maximum step size was chosen slightly smaller than the learning rate at which the standard (no BatchNorm) network diverges during training.

B Omitted Figures

Additional visualizations for the analysis performed in Section 3.1 are presented below.



(a) VGG



(b) DLN

Figure 6: Measurement of ICS (as defined in Definition 2.1) in networks with and without BatchNorm layers. For a layer we measure the cosine angle (ideally 1) and ℓ_2-difference of the gradients (ideally 0) before and after updates to the preceding layers (see Definition 2.1). Models with BatchNorm have similar, or even worse, internal covariate shift, despite performing better in terms of accuracy and loss. (Stabilization of BatchNorm faster during training is an artifact of parameter convergence.)

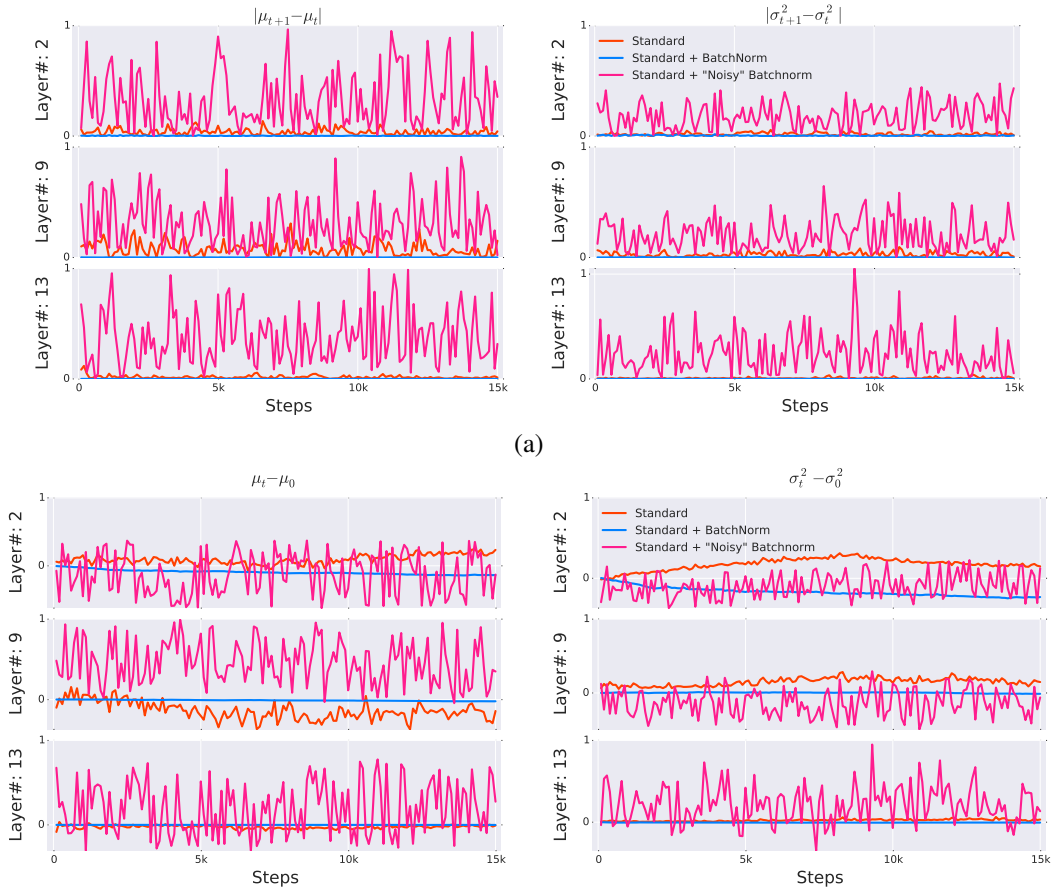


Figure 7: Comparison of change in the first two moments (mean and variance) of distributions of example activations for a given layer between two successive steps of the training process. Here we compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). "Noisy" BatchNorm layers have significantly higher ICS than standard networks, yet perform better from an optimization perspective (cf. Figure 2).

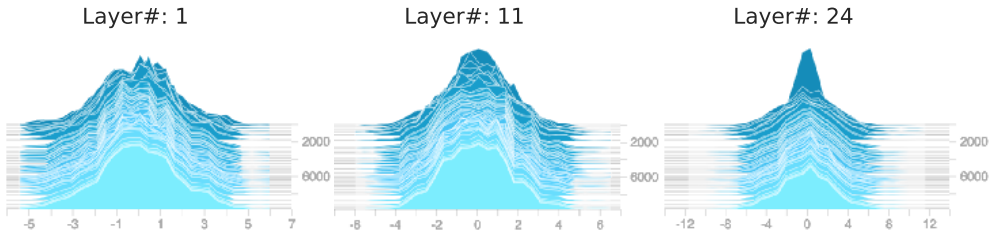
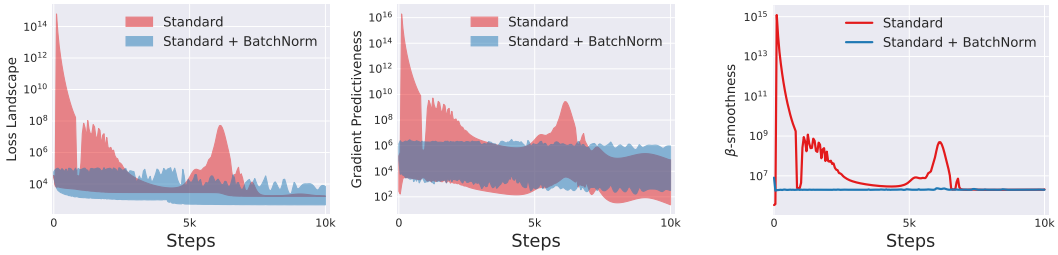
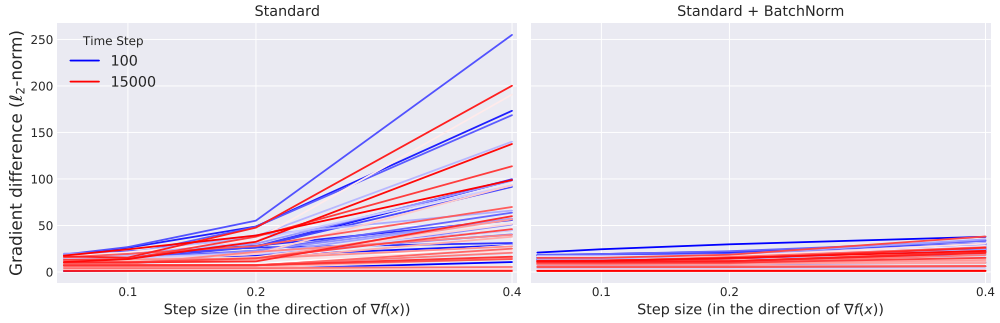


Figure 8: Distributions of activations from different layers of a 25-Layer deep linear network. Here we sample a random activation from a given layer to visualize its distribution over training.

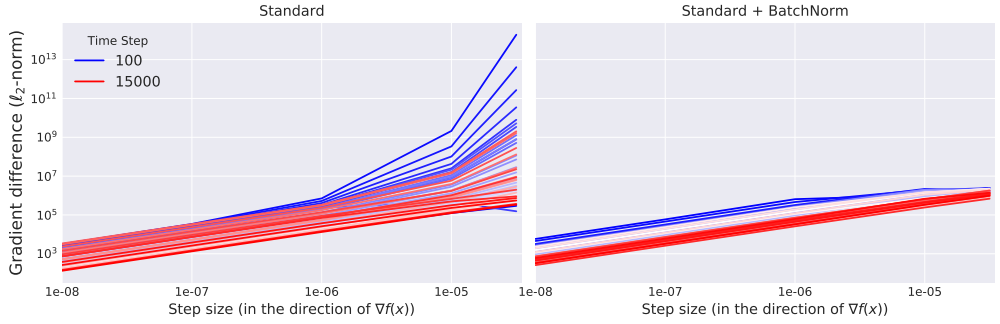


(a) loss landscape (b) gradient predictiveness (c) "effective" β-smoothness

Figure 9: Analysis of the optimization landscape during training of deep linear networks with and without BatchNorm. At a particular training step, we measure the variation (shaded region) in loss (a) and ℓ_2 changes in the gradient (b) as we move in the gradient direction. The "effective" β-smoothness (c) captures the maximum β value observed while moving in this direction. There is a clear improvement in each of these measures of smoothness of the optimization landscape in networks with BatchNorm layers. (Here, we cap the maximum distance moved to be $\eta = 30\times$ the gradient since for larger steps the standard network just performs works (see Figure 1). However, BatchNorm continues to provide smoothing for even larger distances.)



(a) VGG



(b) DLN

Figure 10: Comparison of the predictiveness of gradients with and without BatchNorm. Here, at a given step in the optimization, we measure the ℓ_2 error between the current gradient, and new gradients which are observed while moving in the direction of the current gradient. We then evaluate how this error varies based on distance traversed in the direction of the gradient. We observe that gradients are significantly more predictive in networks with BatchNorm and change slowly in a given local neighborhood. This explains why networks with BatchNorm are largely robust to a broad range of learning rates.

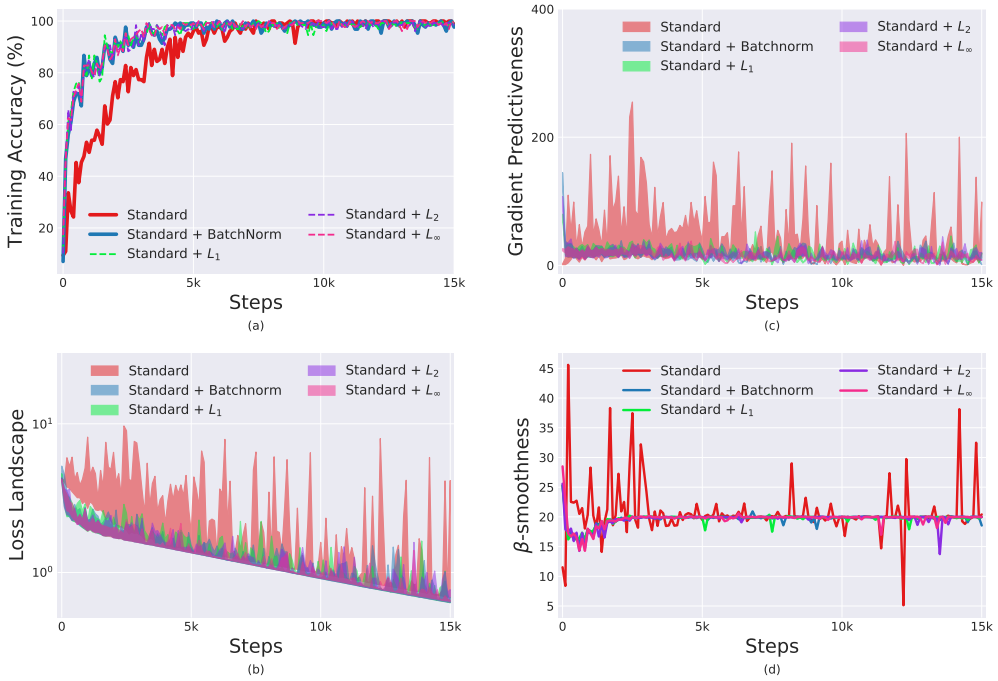


Figure 11: Evaluation of VGG networks trained with different ℓ_p normalization strategies discussed in Section 3.3. *(a):* Comparison of the training performance of the models. *(b, c, d):* Evaluation of the smoothness of optimization landscape in the various models. At a particular training step, we measure the variation (shaded region) in loss (*b*) and ℓ_2 changes in the gradient (*c*) as we move in the gradient direction. We also measure the maximum β-smoothness while moving in this direction (*d*). We observe that networks with any normalization strategy have improved performance and smoothness of the loss landscape over standard training.

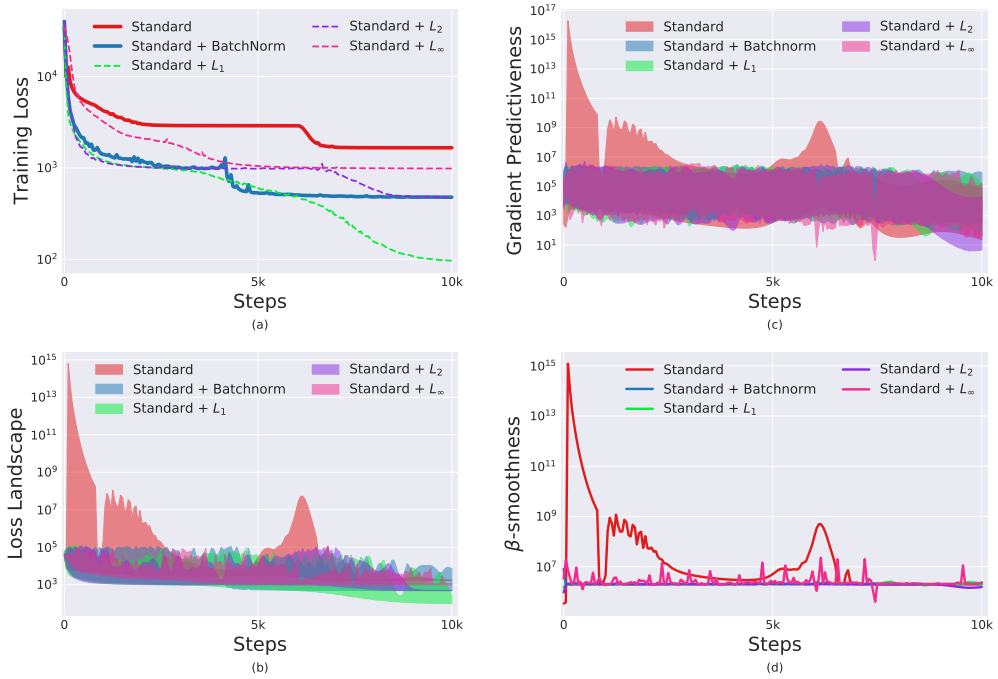
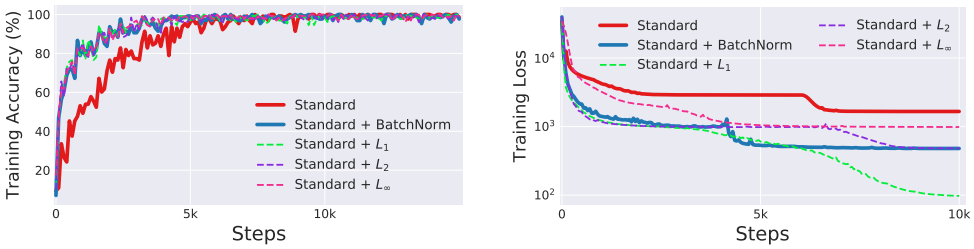


Figure 12: Evaluation of deep linear networks trained with different ℓ_p normalization strategies. We observe that networks with any normalization strategy have improved performance and smoothness of the loss landscape over standard training. Details of the plots are the same as Figure 11 above.



(a) VGG (b) Deep Linear Model

Figure 13: Evaluation of the training performance of ℓ_p normalization techniques discussed in Section 3.3. For both networks, all ℓ_p normalization strategies perform comparably or even better than BatchNorm. This indicates that the performance gain with BatchNorm is not about distributional stability (controlling mean and variance).

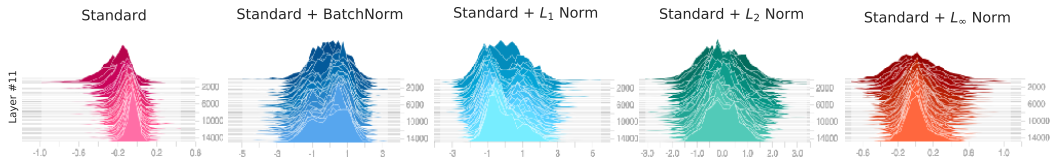


Figure 14: Activation histograms for the VGG network under different normalizations. Here, we randomly sample activations from a given layer and visualize their distributions. Note that the ℓ_p-normalization techniques leads to larger distributional covariate shift compared to normal networks, yet yield improved optimization performance.

C Proofs

We now prove the stated theorems regarding the landscape induced by batch normalization.

We begin with a few facts that can be derived directly from the closed-form of Batch Normalization, which we use freely in proving the following theorems.

C.1 Useful facts and setup

We consider the same setup pictured in Figure 5 and described in Section 4.1. Note that in proving the theorems we use partial derivative notation instead of gradient notation, and also rely on a few simple but key facts:

Fact C.1 (Gradient through BatchNorm)**.** *The gradient $\frac{\partial f}{\partial A^{(b)}}$ through BN and another function $f := f(C)$, where $C = \gamma \cdot B + \beta$, and $B = BN_{0,1}(A) := \frac{A-\mu}{\sigma}$ where $A^{(b)}$ are scalar elements of a batch of size m and variance σ^2 is*

$$\frac{\partial f}{\partial A^{(b)}} = \frac{\gamma}{m\sigma}\left(m\frac{\partial f}{\partial C^{(b)}} - \sum_{k=1}^{m}\frac{\partial f}{\partial C^{(k)}} - B^{(b)}\sum_{k=1}^{m}\frac{\partial f}{\partial C^{(k)}}B^{(k)}\right)$$

Fact C.2 (Gradients of normalized outputs)**.** *A convenient gradient of BN is given as*

$$\frac{\partial \hat{y}^{(b)}}{\partial y^{(k)}} = \frac{1}{\sigma}\left(\mathbf{1}[b=k] - \frac{1}{m} - \frac{1}{m}\hat{y}^{(b)}\hat{y}^{(k)}\right), \tag{1}$$

and thus

$$\frac{\partial z_j^{(b)}}{\partial y^{(k)}} = \frac{\gamma}{\sigma}\left(\mathbf{1}[b=k] - \frac{1}{m} - \frac{1}{m}\hat{y}^{(b)}\hat{y}^{(k)}\right), \tag{2}$$

C.2 Lipschitzness proofs

Now, we provide a proof for the Lipschitzness of the loss landscape in terms of the layer activations. In particular, we prove the following theorem from Section 4.

Theorem 4.1 (The effect of BatchNorm on the Lipschitzness of the loss)**.** *For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network with (identical) loss $\mathcal{L}$,*

$$\left|\left|\nabla_{\boldsymbol{y_j}}\widehat{\mathcal{L}}\right|\right|^2 \leq \frac{\gamma^2}{\sigma_j^2}\left(\left|\left|\nabla_{\boldsymbol{y_j}}\mathcal{L}\right|\right|^2 - \frac{1}{m}\left\langle \mathbf{1}, \nabla_{\boldsymbol{y_j}}\mathcal{L}\right\rangle^2 - \frac{1}{m}\left\langle \nabla_{\boldsymbol{y_j}}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).$$

Proof. Proving this is simply a direct application of Fact C.1. In particular, we have that

$$\frac{\partial\widehat{\mathcal{L}}}{\partial {y_j}^{(b)}} = \left(\frac{\gamma}{m\sigma_j}\right)\left(m\frac{\partial\widehat{\mathcal{L}}}{\partial {z_j}^{(b)}} - \sum_{k=1}^{m}\frac{\partial\widehat{\mathcal{L}}}{\partial {z_j}^{(k)}} - \hat{y}_j^{(b)}\sum_{k=1}^{m}\frac{\partial\widehat{\mathcal{L}}}{\partial {z_j}^{(k)}}\hat{y}_j^{(k)}\right), \tag{3}$$

which we can write in vectorized form as

$$\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} = \left(\frac{\gamma}{m\sigma_j}\right)\left(m\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\left\langle \mathbf{1}, \frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}}\right\rangle - \hat{\boldsymbol{y}}_{\boldsymbol{j}}\left\langle \frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle\right) \tag{4}$$

Now, let $\mu_g = \frac{1}{m}\left\langle \mathbf{1}, \partial\widehat{\mathcal{L}}/\partial \boldsymbol{z_j}\right\rangle$ be the mean of the gradient vector, we can rewrite the above as the following (in the subsequent steps taking advantage of the fact that $\hat{\boldsymbol{y}}_{\boldsymbol{j}}$ is mean-zero and norm-$\sqrt{m}$:

$$\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} = \left(\frac{\gamma}{\sigma_j}\right)\left(\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right) - \frac{1}{m}\hat{\boldsymbol{y}}_{\boldsymbol{j}}\left\langle\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right), \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle\right) \tag{5}$$

$$= \frac{\gamma}{\sigma}\left(\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right) - \frac{\hat{\boldsymbol{y}}_{\boldsymbol{j}}}{||\hat{\boldsymbol{y}}_{\boldsymbol{j}}||}\left\langle\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right), \frac{\hat{\boldsymbol{y}}_{\boldsymbol{j}}}{||\hat{\boldsymbol{y}}_{\boldsymbol{j}}||}\right\rangle\right) \tag{6}$$

$$\left|\left|\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}}\right|\right|^2 = \frac{\gamma^2}{\sigma^2}\left|\left|\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right) - \frac{\hat{\boldsymbol{y}}_{\boldsymbol{j}}}{||\hat{\boldsymbol{y}}_{\boldsymbol{j}}||}\left\langle\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right), \frac{\hat{\boldsymbol{y}}_{\boldsymbol{j}}}{||\hat{\boldsymbol{y}}_{\boldsymbol{j}}||}\right\rangle\right|\right|^2 \tag{7}$$

$$= \frac{\gamma^2}{\sigma^2}\left(\left|\left|\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right)\right|\right|^2 - \left\langle\left(\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}} - \mathbf{1}\mu_g\right), \frac{\hat{\boldsymbol{y}}_{\boldsymbol{j}}}{||\hat{\boldsymbol{y}}_{\boldsymbol{j}}||}\right\rangle^2\right) \tag{8}$$

$$= \frac{\gamma^2}{\sigma^2}\left(\left|\left|\frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}}\right|\right|^2 - \frac{1}{m}\left\langle \mathbf{1}, \frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}}\right\rangle^2 - \frac{1}{m}\left\langle \frac{\partial\widehat{\mathcal{L}}}{\partial \boldsymbol{z_j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle^2\right) \tag{9}$$

Exploiting the fact that $\partial\widehat{\mathcal{L}}/\partial z_j = \partial\mathcal{L}/\partial y$ gives the desired result. □

Next, we can use this to prove the minimax bound on the Lipschitzness with respect to the weights.

Theorem 4.4 (Minimax bound on weight-space Lipschitzness)**.** *For a BatchNorm network with loss $\widehat{\mathcal{L}}$ and an identical non-BN network (with identical loss $\mathcal{L}$), if*

$$g_j = \max_{||X||\leq\lambda}||\nabla_W\mathcal{L}||^2, \qquad \hat{g}_j = \max_{||X||\leq\lambda}\left|\left|\nabla_W\widehat{\mathcal{L}}\right|\right|^2 \implies \hat{g}_j \leq \frac{\gamma^2}{\sigma_j^2}\left(g_j^2 - m\mu_{g_j}^2 - \lambda^2\left\langle\nabla_{\boldsymbol{y}_j}\mathcal{L}, \hat{\boldsymbol{y}}_j\right\rangle^2\right).$$

Proof. To prove this, we start with the following identity for the largest eigenvalue λ_0 of $M \in \mathbb{R}^{d\times d}$:

$$\lambda_0 = \max_{x\in\mathbb{R}^d; ||x||_2=1} x^\top M x, \tag{10}$$

which in turn implies that for a matrix X with $||X||_2 \leq \lambda$, it must be that $v^\top X v \leq \lambda||v||^2$, with the choice of $X = \lambda I$ making this bound tight.

Now, we derive the gradient with respect to the weights via the chain rule:

$$\frac{\partial \widehat{\mathcal{L}}}{\partial W_{ij}} = \sum_{b=1}^{m} \frac{\partial \widehat{\mathcal{L}}}{\partial {y_j}^{(b)}} \frac{\partial {y_j}^{(b)}}{\partial W_{ij}} \tag{11}$$

$$\frac{\partial \widehat{\mathcal{L}}}{\partial W_{ij}} = \sum_{b=1}^{m} \frac{\partial \widehat{\mathcal{L}}}{\partial {y_j}^{(b)}} {x_i}^{(b)} \tag{12}$$

$$= \left\langle \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}}, \boldsymbol{x_i} \right\rangle \tag{13}$$

$$\frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}} = \boldsymbol{X}^\top \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right), \tag{14}$$

where $\boldsymbol{X} \in \mathbb{R}^{m \times d}$ is the input matrix holding $X_{bi} = x_i^{(b)}$. Thus,

$$\left|\left| \frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}} \right|\right|^2 = \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right)^\top \boldsymbol{X}\boldsymbol{X}^\top \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right), \tag{15}$$

and since we have $||X||_2 \leq \lambda$, we must have $||XX^\top||_2 \leq \lambda^2$, and so recalling (10),

$$\max_{||X||_2 < \lambda} \left|\left| \frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}} \right|\right|^2 \leq \lambda^2 \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right)^\top \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right) = \lambda^2 \left|\left| \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j}} \right|\right|^2, \tag{16}$$

and applying Theorem 4.1 yields:

$$\hat{g_j} := \max_{||X||_2 < \lambda} \left|\left| \frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}} \right|\right|^2 \leq \frac{\lambda^2 \gamma^2}{\sigma^2} \left(\left|\left| \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j} \right|\right|^2 - \frac{1}{m} \left\langle \mathbf{1}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j} \right\rangle^2 - \frac{1}{\sqrt{m}} \left\langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j}, \hat{\boldsymbol{y}}_j \right\rangle^2 \right). \tag{17}$$

Finally, by applying (10) again, note that in fact in the normal network,

$$g_j := \max_{||X||_2 < \lambda} \left|\left| \frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}} \right|\right|^2 = \lambda^2 \left|\left| \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j} \right|\right|^2, \tag{18}$$

and thus

$$\hat{g}_j \leq \frac{\gamma^2}{\sigma^2} \left(g_j^2 - m\mu_{g_j}^2 - \lambda^2 \left\langle \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j}, \hat{\boldsymbol{y}}_j \right\rangle^2 \right).$$

□

Theorem 4.2 (The effect of BN to smoothness)**.** *Let $\hat{\boldsymbol{g}}_{\boldsymbol{j}} = \nabla_{\boldsymbol{y}_j} \mathcal{L}$ and $\boldsymbol{H}_{jj} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j}$ be the gradient and Hessian of the loss with respect to the layer outputs respectively. Then*

$$\left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j} \left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2} \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right)^\top \boldsymbol{H}_{jj} \left(\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j}\right) - \frac{\gamma}{m\sigma^2} \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \left\| \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j} \right\|^2$$

If we also have that the $\boldsymbol{H}_{jj}$ preserves the relative norms of $\hat{\boldsymbol{g}}_{\boldsymbol{j}}$ and $\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}$,

$$\left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j} \left(\nabla_{\boldsymbol{y}_j} \widehat{\mathcal{L}}\right) \leq \frac{\gamma^2}{\sigma^2} \left(\hat{\boldsymbol{g}}_j^\top \boldsymbol{H}_{jj} \hat{\boldsymbol{g}}_{\boldsymbol{j}} - \frac{1}{m\gamma} \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \left\| \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j} \right\|^2\right)$$

Proof. We use the following notation freely in the following. First, we introduce the hessian with respect to the final activations as:

$$\boldsymbol{H}_{jk} \in \mathbb{R}^{m \times m}; H_{jk} := \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{z}_j \partial \boldsymbol{z}_k} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_k},$$

where the final equality is by the assumptions of our setup. Once again for convenience, we define a function $\mu_{(\cdot)}$ which operates on vectors and matrices and gives their element-wise mean; in particular, $\mu_{(\boldsymbol{v})} = \frac{1}{d} \mathbf{1}^\top v$ for $v \in \mathbb{R}^d$ and we write $\boldsymbol{\mu}_{(\cdot)} = \mu_{(\cdot)} \mathbf{1}$ to be a vector with all elements equal to μ. Finally, we denote the gradient with respect to the batch-normalized outputs as $\hat{\boldsymbol{g}}_{\boldsymbol{j}}$, such that:

$$\hat{\boldsymbol{g}}_{\boldsymbol{j}} = \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{z}_j} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{y}_j},$$

where again the last equality is by assumption.

Now, we begin by looking at the Hessian of the loss with respect to the pre-BN activations $\boldsymbol{y}_{\boldsymbol{j}}$ using the expanded gradient as above:

$$\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y}_j \partial \boldsymbol{y}_j} = \frac{\partial}{\partial \boldsymbol{y}_j} \left(\left(\frac{\gamma}{m\sigma_j}\right) \left[m\hat{\boldsymbol{g}}_{\boldsymbol{j}} - m\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)} - \hat{y}_j^{(b)} \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \right] \right) \tag{19}$$

Using the product rule and the chain rule:

$$= \frac{\gamma}{m\sigma} \left(\frac{\partial}{\partial \boldsymbol{z}_{\boldsymbol{q}}} \left[m\hat{\boldsymbol{g}}_{\boldsymbol{j}} - m\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)} - \hat{\boldsymbol{y}}_j \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \right] \right) \cdot \frac{\partial \boldsymbol{z}_{\boldsymbol{q}}}{\partial \boldsymbol{y}_{\boldsymbol{j}}} \tag{20}$$

$$+ \left(\frac{\partial}{\partial \boldsymbol{y}_{\boldsymbol{j}}} \left(\frac{\gamma}{m\sigma_j} \right) \right) \cdot \left(m\hat{\boldsymbol{g}}_{\boldsymbol{j}} - m\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)} - \hat{\boldsymbol{y}}_j \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \right) \tag{21}$$

Distributing the derivative across subtraction:

$$= \left(\frac{\gamma}{\sigma_j}\right) \left(\boldsymbol{H}_{jj} - \frac{\partial \boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)}}{\partial \boldsymbol{z}_{\boldsymbol{j}}} - \frac{\partial}{\partial \boldsymbol{z}_{\boldsymbol{j}}} \left(\frac{1}{m} \hat{\boldsymbol{y}}_j \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \right) \right) \cdot \frac{\partial \boldsymbol{z}_{\boldsymbol{j}}}{\partial \boldsymbol{y}_{\boldsymbol{j}}} \tag{22}$$

$$+ \left(\hat{\boldsymbol{g}}_{\boldsymbol{j}} - \boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)} - \frac{1}{m} \hat{\boldsymbol{y}}_j \left\langle \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j \right\rangle \right) \left(\frac{\partial}{\partial \boldsymbol{y}_{\boldsymbol{j}}} \left(\frac{\gamma}{\sigma_j} \right) \right) \tag{23}$$

We address each of the terms in the above (22) and (23) one by one:

$$\frac{\partial \boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_j)}}{\partial \boldsymbol{z}_{\boldsymbol{j}}} = \frac{1}{m} \frac{\partial \mathbf{1}^\top \hat{\boldsymbol{g}}_{\boldsymbol{j}}}{\partial \boldsymbol{z}_{\boldsymbol{j}}} = \frac{1}{m} \mathbf{1} \cdot \mathbf{1}^\top \boldsymbol{H}_{jj} \tag{24}$$

$$\frac{\partial}{\partial \boldsymbol{z_j}}\left(\hat{\boldsymbol{y}}_j\left\langle\hat{\boldsymbol{y}}_j, \hat{\boldsymbol{g}}_{\boldsymbol{j}}\right\rangle\right)=\frac{1}{\gamma} \frac{\partial}{\partial \hat{\boldsymbol{y}}_j}\left(\hat{\boldsymbol{y}}_j\left\langle\hat{\boldsymbol{y}}_j, \hat{\boldsymbol{g}}_{\boldsymbol{j}}\right\rangle\right) \tag{25}$$

$$=\frac{1}{\gamma} \frac{\partial \hat{\boldsymbol{y}}_j}{\partial \hat{\boldsymbol{y}}_j}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle+\hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj}+\frac{1}{\gamma} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top} \frac{\partial \hat{\boldsymbol{y}}_j}{\partial \hat{\boldsymbol{y}}_j} \tag{26}$$

$$=\frac{1}{\gamma} \boldsymbol{I}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle+\hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj}+\frac{1}{\gamma} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top} \boldsymbol{I} \tag{27}$$

$$\frac{\partial}{\partial \boldsymbol{y_j}}\left(\frac{\gamma}{\sigma_j}\right)=\gamma \sqrt{m} \frac{\partial\left(\left(\boldsymbol{y_j}-\boldsymbol{\mu}_{(\boldsymbol{y_j})}\right)^{\top}\left(\boldsymbol{y_j}-\boldsymbol{\mu}_{(\boldsymbol{y_j})}\right)\right)^{-\frac{1}{2}}}{\partial \boldsymbol{y_j}} \tag{28}$$

$$=\frac{-1}{2} \gamma \sqrt{m}\left(\left(\boldsymbol{y_j}-\boldsymbol{\mu}_{(\boldsymbol{y_j})}\right)^{\top}\left(\boldsymbol{y_j}-\boldsymbol{\mu}_{(\boldsymbol{y_j})}\right)\right)^{-\frac{3}{2}}\left(2\left(\boldsymbol{y_j}-\boldsymbol{\mu}_{(\boldsymbol{y_j})}\right)\right) \tag{29}$$

$$=-\frac{\gamma}{m \sigma^2} \hat{\boldsymbol{y}}_j \tag{30}$$

Now, we can use the preceding to rewrite the Hessian as:

$$\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j} \partial \boldsymbol{y_j}}=\left(\frac{\gamma}{m \sigma_j}\right)\left(m \boldsymbol{H}_{jj}-\mathbf{1} \cdot \mathbf{1}^{\top} \boldsymbol{H}_{jj}-\frac{1}{\gamma} \boldsymbol{I}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle-\hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj}-\frac{1}{\gamma}\left(\hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top}\right)\right) \cdot \frac{\partial \boldsymbol{z_j}}{\partial \boldsymbol{y_j}} \tag{31}$$

$$-\frac{\gamma}{m \sigma^2}\left(\hat{\boldsymbol{g}}_{\boldsymbol{j}}-\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_{\boldsymbol{j}})}-\frac{1}{m} \hat{\boldsymbol{y}}_j\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle\right) \hat{\boldsymbol{y}}_j^{\top} \tag{32}$$

Now, using Fact C.2, we have that:

$$\frac{\partial \boldsymbol{z_j}}{\partial \boldsymbol{y_j}}=\left(\frac{\gamma}{\sigma_j}\right)\left(\boldsymbol{I}-\frac{1}{m} \mathbf{1} \cdot \mathbf{1}^{\top}-\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}\right), \tag{33}$$

and substituting this yields (letting $\boldsymbol{M}=\mathbf{1} \cdot \mathbf{1}^{\top}$ for convenience):

$$\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j} \partial \boldsymbol{y_j}}=\frac{\gamma^2}{m \sigma^2}\left(m \boldsymbol{H}_{jj}-\boldsymbol{M} \boldsymbol{H}_{jj}-\frac{1}{\gamma} \boldsymbol{I}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle-\hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj}-\frac{1}{\gamma}\left(\hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top}\right)\right) \tag{34}$$

$$-\frac{\gamma^2}{m \sigma^2}\left(\boldsymbol{H}_{jj} \boldsymbol{M}-\frac{1}{m} \boldsymbol{M} \boldsymbol{H}_{jj} \boldsymbol{M}-\frac{1}{m \gamma} \boldsymbol{M}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle-\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj} \boldsymbol{M}-\frac{1}{m \gamma}\left(\hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top} \boldsymbol{M}\right)\right) \tag{35}$$

$$-\frac{\gamma^2}{m \sigma^2}\left(\boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}-\frac{1}{m} \boldsymbol{M} \boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}-\frac{1}{m \gamma} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_{\boldsymbol{j}}\right\rangle-\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}-\frac{1}{m \gamma}\left(\hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}\right)\right) \tag{36}$$

$$-\frac{\gamma}{m \sigma^2}\left(\hat{\boldsymbol{g}}_{\boldsymbol{j}}-\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_{\boldsymbol{j}})}-\frac{1}{m} \hat{\boldsymbol{y}}_j\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle\right) \hat{\boldsymbol{y}}_j^{\top} \tag{37}$$

Collecting the terms, and letting $\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}=\hat{\boldsymbol{g}}_{\boldsymbol{j}}-\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_{\boldsymbol{j}})}$:

$$\frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j} \partial \boldsymbol{y_j}}=\frac{\gamma^2}{m \sigma^2}\Bigg[m \boldsymbol{H}_{jj}-\boldsymbol{M} \boldsymbol{H}_{jj}-\hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj}-\boldsymbol{H}_{jj} \boldsymbol{M}+\frac{1}{m} \boldsymbol{M} \boldsymbol{H}_{jj} \boldsymbol{M} \tag{38}$$

$$+\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj} \boldsymbol{M}-\boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}+\frac{1}{m} \boldsymbol{M} \boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}+\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top} \boldsymbol{H}_{jj} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}\Bigg] \tag{39}$$

$$-\frac{\gamma}{m \sigma^2}\left(\hat{\boldsymbol{g}}_{\boldsymbol{j}} \hat{\boldsymbol{y}}_j^{\top}-\boldsymbol{\mu}_{(\hat{\boldsymbol{g}}_{\boldsymbol{j}})} \hat{\boldsymbol{y}}_j^{\top}-\frac{3}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle+\left(\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle \boldsymbol{I}+\hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top}\right)\left(\boldsymbol{I}-\frac{1}{m} \boldsymbol{M}\right)\right) \tag{40}$$

$$=\frac{\gamma^2}{\sigma^2}\Bigg[\left(\boldsymbol{I}-\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}-\frac{1}{m} \boldsymbol{M}\right) \boldsymbol{H}_{jj}\left(\boldsymbol{I}-\frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^{\top}-\frac{1}{m} \boldsymbol{M}\right) \tag{41}$$

$$-\frac{1}{m \gamma}\left(\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}} \hat{\boldsymbol{y}}_j^{\top}+\hat{\boldsymbol{y}}_j \overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}^{\top}-\frac{3}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{g}}_{\boldsymbol{j}}^{\top} \hat{\boldsymbol{y}} \hat{\boldsymbol{y}}_j^{\top}+\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle\left(\boldsymbol{I}-\frac{1}{m} \boldsymbol{M}\right)\right)\Bigg] \tag{42}$$

Now, we wish to calculate the effective beta smoothness with respect to a batch of activations, which corresponds to $g^\top Hg$, where g is the gradient with respect to the activations (as derived in the previous proof). We expand this product noting the following identities:

$$\boldsymbol{M}\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}} = 0 \tag{43}$$

$$\left(I - \frac{1}{m}\boldsymbol{M} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)^2 = \left(I - \frac{1}{m}\boldsymbol{M} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right) \tag{44}$$

$$\hat{\boldsymbol{y}}_j^\top\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right) = \boldsymbol{0} \tag{45}$$

$$\left(\boldsymbol{I} - \frac{1}{m}\boldsymbol{M}\right)\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}} = \left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}} \tag{46}$$

Also recall from (5) that:

$$\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}} = \frac{\gamma}{\sigma}\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}^\top\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right) \tag{47}$$

Applying these while expanding the product gives:

$$\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}^\top \cdot \frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}\partial\boldsymbol{y_j}} \cdot \frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}} = \frac{\gamma^4}{\sigma^4}\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}^\top\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)\boldsymbol{H}_{jj}\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}} \tag{48}$$

$$- \frac{\gamma^3}{m\sigma^4}\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}^\top\left(\boldsymbol{I} - \frac{1}{m}\hat{\boldsymbol{y}}_j\hat{\boldsymbol{y}}_j^\top\right)\overline{\hat{\boldsymbol{g}}_{\boldsymbol{j}}}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle \tag{49}$$

$$= \frac{\gamma^2}{\sigma^2}\left(\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}\right)^\top\boldsymbol{H}_{jj}\left(\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}\right) - \frac{\gamma}{m\sigma^2}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle\left|\left|\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}\right|\right|^2 \tag{50}$$

This concludes the first part of the proof. Note that if $\boldsymbol{H}_{jj}$ preserves the relative norms of $\hat{\boldsymbol{g}}_{\boldsymbol{j}}$ and $\nabla_{\boldsymbol{y_j}}\widehat{\mathcal{L}}$, then the final statement follows trivially, since the first term of the above is simply the induced squared norm $\left|\left|\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}\right|\right|^2_{\boldsymbol{H}_{jj}}$, and so

$$\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}^\top \cdot \frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}\partial\boldsymbol{y_j}} \cdot \frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}} \leq \frac{\gamma^2}{\sigma^2}\left[\hat{\boldsymbol{g}}_{\boldsymbol{j}}^\top\boldsymbol{H}_{jj}\hat{\boldsymbol{g}}_{\boldsymbol{j}} - \frac{1}{m\gamma}\left\langle\hat{\boldsymbol{g}}_{\boldsymbol{j}}, \hat{\boldsymbol{y}}_j\right\rangle\left|\left|\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}}\right|\right|^2\right] \tag{51}$$

□

Once again, the same techniques also give us a minimax separation:

Theorem C.1 (Minimax smoothness bound)**.** *Under the same conditions as the previous theorem,*

$$\max_{||X||\leq\lambda}\left(\frac{\partial\widehat{\mathcal{L}}}{\partial W_{\cdot j}}\right)^\top\frac{\partial\widehat{\mathcal{L}}}{\partial W_{\cdot j}\partial W_{\cdot j}}\left(\frac{\partial\widehat{\mathcal{L}}}{\partial W_{\cdot j}}\right) < \frac{\gamma^2}{\sigma^2}\left[\max_{||X||\leq\lambda}\left(\frac{\partial\mathcal{L}}{\partial W_{\cdot j}}\right)^\top\frac{\partial\mathcal{L}}{\partial W_{\cdot j}\partial W_{\cdot j}}\left(\frac{\partial\mathcal{L}}{\partial W_{\cdot j}}\right) - \lambda^4\kappa\right],$$

where κ is the separation given in the previous theorem.

Proof.

$$\frac{\partial\mathcal{L}}{\partial W_{ij}\partial W_{kj}} = \boldsymbol{x_i}^\top\frac{\partial\mathcal{L}}{\partial\boldsymbol{y_j}\partial\boldsymbol{y_j}}\boldsymbol{x_k} \tag{52}$$

$$\frac{\partial\widehat{\mathcal{L}}}{\partial W_{ij}\partial W_{kj}} = \boldsymbol{x_i}^\top\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}\partial\boldsymbol{y_j}}\boldsymbol{x_k} \tag{53}$$

$$\frac{\partial\widehat{\mathcal{L}}}{\partial W_{\cdot j}\partial W_{\cdot j}} = \boldsymbol{X}^\top\frac{\partial\widehat{\mathcal{L}}}{\partial\boldsymbol{y_j}\partial\boldsymbol{y_j}}\boldsymbol{X} \tag{54}$$

$$\tag{55}$$

Looking at the gradient predictiveness using the gradient we derived in the first proofs:

$$\beta := \left(\frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}}\right)^\top \frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j} \partial W_{\cdot j}} \left(\frac{\partial \widehat{\mathcal{L}}}{\partial W_{\cdot j}}\right) \tag{56}$$

$$= \hat{\boldsymbol{g}}_{\boldsymbol{j}}^\top \left(\boldsymbol{I} - \frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^\top\right) \boldsymbol{X}\boldsymbol{X}^\top \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j} \partial \boldsymbol{y_j}} \boldsymbol{X}\boldsymbol{X}^\top \left(\boldsymbol{I} - \frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^\top\right) \hat{\boldsymbol{g}}_{\boldsymbol{j}} \tag{57}$$

Maximizing the norm with respect to X yields:

$$\max_{||X|| \leq \lambda} \beta = \lambda^4 \hat{\boldsymbol{g}}_{\boldsymbol{j}}^\top \left(\boldsymbol{I} - \frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^\top\right) \frac{\partial \widehat{\mathcal{L}}}{\partial \boldsymbol{y_j} \partial \boldsymbol{y_j}} \left(\boldsymbol{I} - \frac{1}{m} \hat{\boldsymbol{y}}_j \hat{\boldsymbol{y}}_j^\top\right) \hat{\boldsymbol{g}}_{\boldsymbol{j}}, \tag{58}$$

at which the previous proof can be applied to conclude. □

Lemma 4.5 (BatchNorm leads to a favourable initialization)**.** *Let $\boldsymbol{W^*}$ and $\boldsymbol{\widehat{W}^*}$ be the set of local optima for the weights in the normal and BN networks, respectively. For any initialization W_0*

$$\left|\left|W_0 - \widehat{W}^*\right|\right|^2 \leq ||W_0 - W^*||^2 - \frac{1}{||W^*||^2}\left(||W^*||^2 - \langle W^*, W_0\rangle\right)^2,$$

if $\langle W_0, W^*\rangle > 0$, where $\widehat{W}^*$ and W^* are closest optima for BN and standard network, respectively.

Proof. This is as a result of the scale-invariance of batch normalization. In particular, first note that for any optimum W in the standard network, we have that any scalar multiple of W must also be an optimum in the BN network (since $BN((aW)x) = BN(Wx)$ for all $a > 0$). Recall that we have defined $k > 0$ to be propertial to the correlation between W_0 and W^*:

$$k = \frac{\langle W^*, W_0\rangle}{||W^*||^2}$$

Thus, for any optimum W^*, we must have that $\widehat{W} := kW^*$ must be an optimum in the BN network. The difference between distance to this optimum and the distance to W is given by:

$$\left|\left|W_0 - \widehat{W}\right|\right|^2 - ||W_0 - W^*||^2 = ||W_0 - kW^*||^2 - ||W_0 - W^*||^2 \tag{59}$$

$$= \left(||W_0||^2 - k^2\,||W^*||^2\right) - \left(||W_0||^2 - 2k\,||W^*||^2 + ||W^*||^2\right) \tag{60}$$

$$= 2k\,||W^*||^2 - k^2\,||W^*||^2 - ||W^*||^2 \tag{61}$$

$$= -\,||W^*||^2 \cdot (1-k)^2 \tag{62}$$

□