How Does Batch Normalization Help Optimization? (No, It Is Not About Internal Covariate Shift)

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. These findings bring us closer to a true understanding of our DNN training toolkit.

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 [28], and game-playing [17, 24]. This progress hinged on a number of major advances in terms of hardware, datasets [15, 22], and algorithmic and architectural techniques [26, 12, 19, 27]. 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 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 4,000 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, 30], but none of them seem to bring us any closer to understanding this issue. (A similar point was also raised recently in [21].)

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 in its support. In particular, we still do not understand the link between ICS and training performance.

[*]Equal contribution.

Preprint. Work in progress.

The chief goal of this paper is to address all these shortcoming. Our exploration lead to somewhat startling discoveries.

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 true 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 be *significantly more smooth*. This ensures, in particular, that the gradients are more predictive and thus allow for use of larger range of learning rates and faster network convergence. We provide 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 [20]) 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 has 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 concepts 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 discuss further related work in Section 4 and conclude in Section 5.

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 if 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.

Despite its fundamental role and widespread use in deep learning, the underpinnings of BatchNorm's success remain poorly understood [21]. In this work we aim to finally address this gap. To this end, we start by investigating the connection between ICS and BatchNorm. Specifically, we consider first training a standard VGG [25] 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 much more surprising finding. In this figure, we visualize to what extent BatchNorm is stabilizing distributions of layer inputs. 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?*

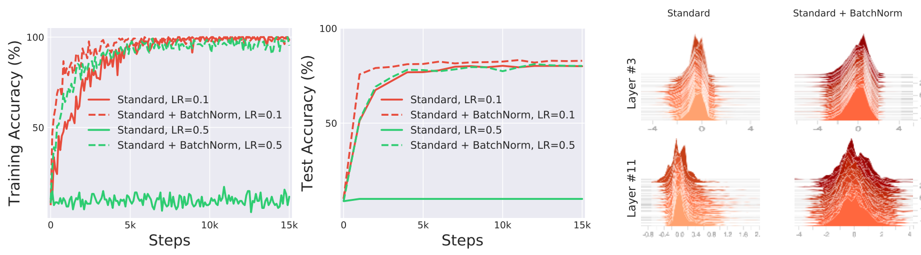


Figure 1: Comparison of (a) training (optimization) and (b) test (generalization) performance of a standard VGG network trained on CIFAR-10 with and without BatchNorm (details in Appendix A). There is a consistent gain in training speed in models with BatchNorm layers. (c) Even though the gap between 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.)

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).

Observe that such noise injection produces a severe covariate shift that skews activations at every time step. Consequently, each 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 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 setups perform much better than standard networks. In particular, "noisy" BatchNorm network has *less stable* distributions than even the

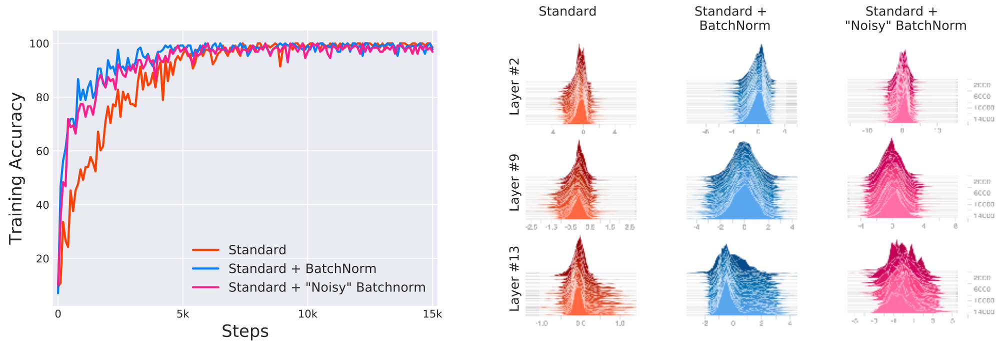
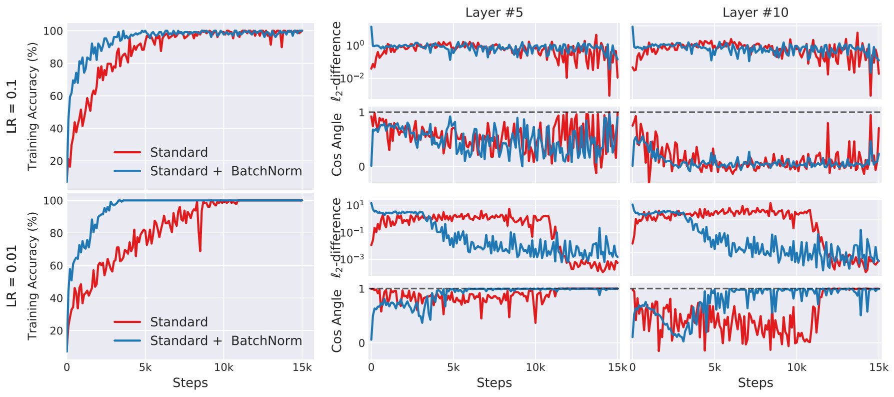
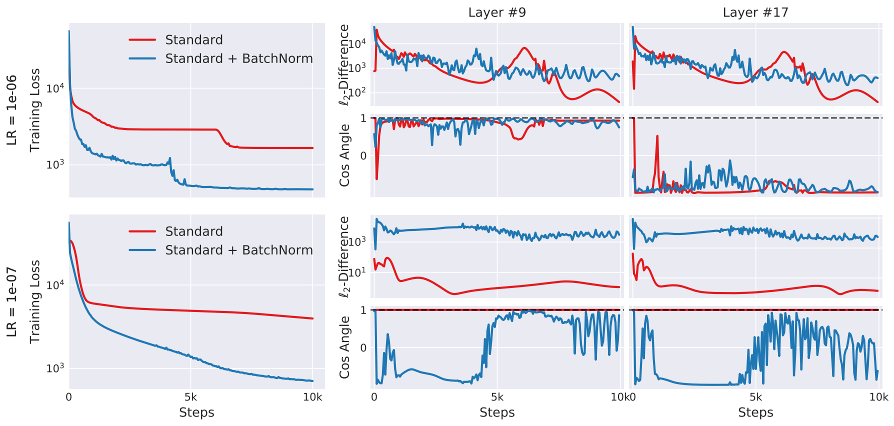


Figure 2: Connections between distributional stability and BatchNorm performance: We compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). In the later case, we induce 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 8).



(a) VGG



(b) DLN

Figure 3: Measurement of internal covariate shift 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 2.2). 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.)

standard, non-BatchNorm network, yet it *still performs better* in terms of training. (Figure 8 in Appendix B plots the variation in the mean and variance of the corresponding distributions.)

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 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. *Let $\mathcal{L}$ be the loss, $W_1^{(t)}, \ldots, W_k^{(t)}$ be the parameters 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 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, controlling the distributions layer inputs as done in BatchNorm, might not even reduce the internal covariate shift.

3 Why does BatchNorm work?

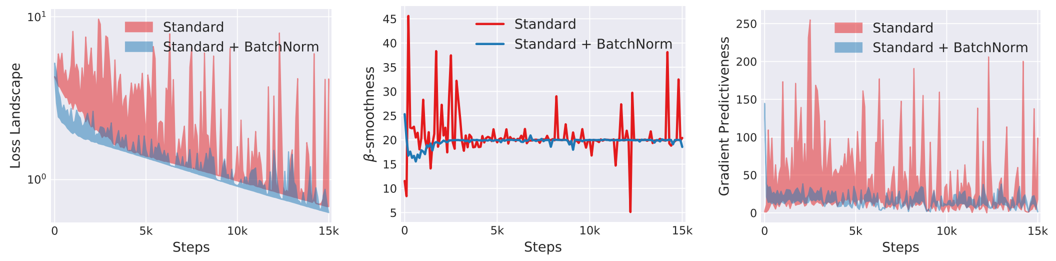
Our investigation so far demonstrated that the generally asserted link between the internal covariate shift (ICS) and the optimization performance is tenuous, at best. But BatchNorm *does* significantly improve the training process. Can we explain why this is the case?

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, an even stronger effect at play. Namely, BatchNorm's reparametrization makes *gradients* of the loss

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



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

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 (c) as we move in the gradient direction. The "effective" β-smoothness (b) refers to 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 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.)

more Lipschitz too. In other words, the loss exhibits a significantly better "effective"[3] β-smoothness (Recall that a function f is β-smooth if its gradients are β-Lipschitz, i.e., if $\|\nabla f(x_1) - \nabla f(x_2)| \leq \beta\|x_1 - x_2\|$, for each x_1 and x_2.).

These smoothening effects impact the performance of the training algorithm in a major way. To understand why, recall that in a vanilla, i.e., 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. This makes gradient descent–based training algorithms very 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 11 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.)

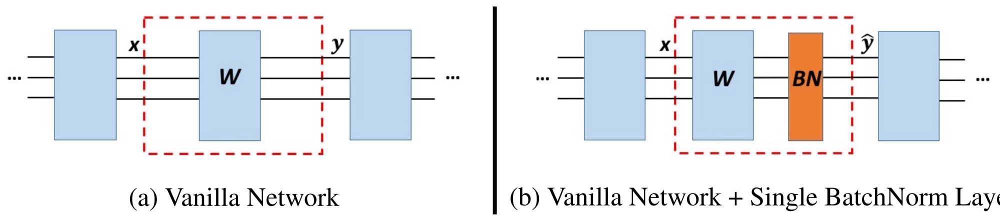
Below, we provide empirical as well as theoretical evidence to support and illustrate these findings.

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 indeed wildly fluctuates, especially in the initial phases of training. (In the later stages, the network is already close to convergence.)

Similarly, to demonstrate the effect of BatchNorm on the stability/Lipschitzness of the gradients of the loss, we plot in Figure 4(b) the "effective" β-smoothness of the vanilla and BatchNorm networks throughout the training. ("Effective" refers here to measuring the change of gradients as move in the direction of the gradients.). Once more, we observe drastic and consistent differences between these two networks.

[3]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. Still, as we will see, locally, i.e., as far as our training trajectory is concerned, the loss does behave as if it was β-smooth.



(a) Vanilla Network (b) Vanilla Network + Single BatchNorm Layer

Figure 5: The two network architectures we compare in our theoretical investigations: (a) the vanilla deep linear network, i.e., deep linear network without BatchNorm layer; (b) the same network as in (a) but with a single BatchNorm layer inserted after the fully-connected layer W. (All the layer parameters have exactly the same value in both networks.)

To further 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(c) shows a significant difference (close to two orders of magnitude) in such gradient predictiveness between the vanilla and BatchNorm networks.

We complement the above examination by considering *linear* deep networks, i.e., a network without non-linearities. As shown in Figures 10 and 13 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 Theoretical analysis

Our experiments above suggests there is a fundamental principle at play here. We now explore this underlying phenomenon from a theoretical perspective. To this end, we consider a deep linear network (DLN), i.e., a deep neural network with fully-connected layers and no non-linearities. (The key phenomena we want to capture arises already in this setting – see Figures 10 and 13 in Appendix B.)

Our goal is to analyze the impact of adding a single BatchNorm layer to such a "vanilla" DLN *at a given step* in the training. Specifically, for an intermediate fully-connected layer W of the network, we compare the optimization landscape of the original training problem to the one that results from inserting a BatchNorm layer right *after* the layer W – see Figure 5.[4] This means that it is the outputs y – and not the inputs x – of the layer W that undergo batch normalization. Thus, our analysis captures effects that stem from the reparametrization of the landscape and not merely from normalization of the inputs x. (In fact, we do not assume any whitening of these inputs x.) Note that in our analysis, all the layer parameters have exactly the same values for both the networks.

We begin by examining the local sensitivity of the network. To this end, let $|\partial y_k^l/\partial W_{ij}|$ (resp. $|\partial \hat{y}_k^l/\partial W_{ij}|$) be the sensitivity of the k-th output of the layer W on the l-th mini-batch example to the parameter W_{ij} of that layer when there is no (resp. when there is) BatchNorm layer inserted. (Observe that we can focus on the case of $k = j$ as both $\partial y_k^l/\partial W_{ij}$ and $\partial \hat{y}_k^l/\partial W_{ij}$ is trivially zero otherwise.) Also, let $\sigma(u)$ denote the standard deviation (computed over the mini-batch) of a network input/output u. (Note that $\sigma(\hat{y}_j) = 1$, for all j.) We show that insertion of a BatchNorm makes these sensitivities become significantly smaller (when averaged over the whole mini-batch). Specifically, the following theorem is proved in Appendix C.

Theorem 3.1 (The effect of BatchNorm on the outputs' sensitivity)**.** *We have that, for any i and j,*

$$\sum_{l=1}^{m}\left(\frac{\partial \hat{y}_j^l}{\partial W_{ij}}\right)^2 = \frac{1}{\sigma(y_j)^2}\left(\sum_{l=1}^{m}\left(\frac{\partial y_j^l}{\partial W_{ij}}\right)^2 - m^{-1}\alpha_{ij}^2\right),$$

[4]We consider here BatchNorm layers that normalize each its input to have mean 0 and variance 1 but it is not hard to extend our analysis to the case of having a trainable mean and variance. Also, again, we observe experimentally that the phenomenon we want to capture arises already in this setting.

where α_{ij} is the correlation between x_is and y_js.

Observe that we expect the correlation α_{ij} to be non-zero; if it was zero then the input data is essentially useless in classification, and if W is close to the identity (e.g. in a ResNet), then $\alpha^2 \approx m^2\sigma(x_i)^2$. Furthermore, we expect $\sigma(y_j) \geq 1$, since the contrary implies that the vanilla network normalized the variance of y_j to a stronger degree than the BatchNorm layer. (In fact, experimentally we observe that $\sigma(y_j)$ is consistently large (around 5; cf. Appendix Figure 9).) Consequently, the above theorem tells us that using BatchNorm leads to reduction of the (average squared) parameter sensitivity of a given output unless the corresponding parameter is not helpful or that output is "batch normalized" to begin with.

We now turn our attention to more direct properties of the optimization landscape: the size of the gradient $\nabla_W \mathcal{L}$ of the loss wrt layer parameters W, which captures the Lipschitzness of the loss. To this end, for an input-output pair (x_i, y_j), define c_{ij} to be the correlation (over the mini-batch) between the input x_i and $\partial\mathcal{L}/\partial y_j$. The complete statement of the following theorem along with the proof is given in Appendix C.

Theorem 3.2 (The effect of BatchNorm on the Lipschitzness of the loss)**.** *For every i and j,*

$$\left|\nabla_{W_{ij}} \widehat{\mathcal{L}}\right|^2 \leq \frac{\left(|c_{ij}|^{-1/2}+1\right)^2}{\sigma(y_j)^2} \left|\nabla_{W_{ij}} \mathcal{L}\right|^2$$

where $\widehat{\mathcal{L}}$ is the loss function of the network with the BatchNorm layer inserted.

Note in practice $|c_{ij}| \gg 0$. In particular, if $|c_{ij}| = 0$ then $\nabla_{W_{ij}}\mathcal{L} = 0$. Consequently, the above theorem tells us that whenever the outputs do not have BatchNorm-like normalization, inserting BatchNorm layer *always* makes the loss be more Lipschitz. Again, we note that $\sigma(y_j)$s tend to be large in practice (cf. Appendix Figure 9). Furthermore, one should observe that this increase in Lipschitzness (i.e., decrease of the Lipschitz constant) is not merely a manifestation of a trivial rescaling of the objective function. As shown in the following lemma (whose proof appears in Appendix C), after introducing a BatchNorm layer the (near-)optimal solutions remain (near-)optimal. So, as our measure of empirical loss minimization remains the same, BatchNorm layers indeed have a directly beneficial effect on the optimization landscape.

Lemma 3.3. *Consider the setup of Theorem 3.2 (0-mean input and $(\gamma, \beta) = (1, 0)$), where $\mathcal{L}$ is the ℓ_2-norm loss on zero-mean labels ℓ. Then, as long as an optimum is ε-attainable with BN[5], we have that for any W such that $\mathcal{L}_W(x) < \varepsilon$, the same W satisfies:*

$$\widehat{\mathcal{L}}_W(x) < 3\varepsilon\left(\frac{1+\varepsilon}{1-2\varepsilon}\right)$$

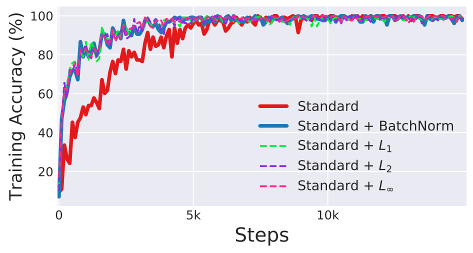
Finally, we analyze the impact of adding a BatchNorm layer on the minor $\nabla^2\mathcal{L}_{W,W}$ of the Hessian $\nabla^2\mathcal{L}$ corresponding to layer W. We show that under similar conditions as the preceding theorem (correlation with input and elevated variance) we can actually show that *each element* of the Hessian with respect to W shrinks by a positive factor. While the full theorem statement (along with the proof) are given in Appendix C, we give a simplified bound in the following.

Theorem 3.4 (The effect of BatchNorm on β-smoothness)**.** *Under certain correlation and variance conditions, we have that $\exists\ \kappa_{ij}, \kappa_{kl} > 0, \gamma_{ijkl}$ such that each entry of the Hessian (i.e. the second partial derivative with respect to W_{ij} and W_{kl}) is bounded as*

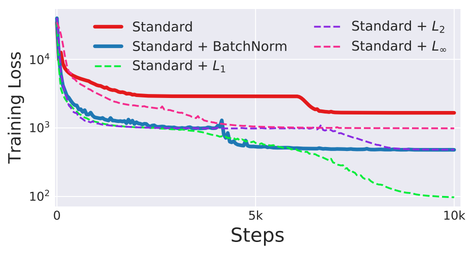
$$\left|\nabla^2_W \widehat{\mathcal{L}}\right|_{ijkl} \leq \frac{\kappa_{ij}\kappa_{kl} + \gamma_{ijkl}}{\sigma(y_j)\sigma(y_l)} \left|\nabla^2_W \mathcal{L}\right|_{ijkl}$$

The fact that entries of the Hessian are bounded element-wise translates into a bound on the spectral norm of the Hessian (and therefore an improvement in β-smoothness). (Recall, once again that $\sigma(y_j)$ tends to be large in practice; see Appendix Figure 9.) Thus, the loss landscape induced by BatchNorm is better-behaved both in terms of continuity and smoothness, in agreement with our empirical results.

[5] i.e. so long as $\exists\ W'$ s.t. $\widehat{\mathcal{L}}_{W'}(x) < \varepsilon$



(a) VGG



(b) Deep Linear Model

Figure 6: Evaluation of the training performance of ℓ_p normalization techniques discussed in Section 3.4. 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).

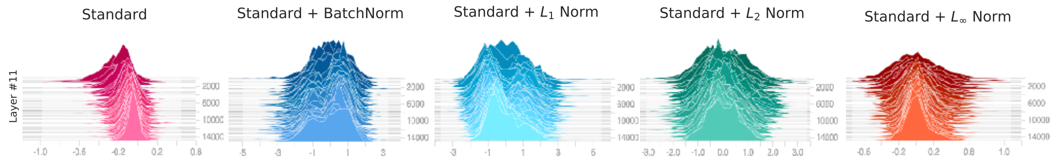


Figure 7: Activation histograms for the VGG network under different normalizations. Here, we randomly sample activations from a given layer and visualize their distributions. Note that the ℓ_p-normalization techniques leads to larger distributional covariate shift compared to normal networks, yet yield improved optimization performance.

3.4 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 (see Figure 7). Hence, normalization with such ℓ_p-norm does not guarantee anymore any control over the distribution moments nor distributional stability.

The results are presented in Figure 6, as well as Figures 12 and 13 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 the ℓ_p–normalization techniques lead to *larger distributional covariate 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 12 and 13 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 similar normalization schemes as it can lead to even better performance.

4 Related work

A number of normalization schemes have been proposed as alternatives to BatchNorm. Layer Normalization [1] performs normalization over the entire layer instead of the batch, which is suitable for contexts where the notion of a batch is problematic (e.g. recurrent neural networks). Group Normalization [30] normalizes across a subset of the batch and can be applied in the case of a variable batch size. Instance Normalization [29] focuses on image classification and normalizes across the height and width dimensions. Weight Normalization [23] follows a complementary approach and normalized the weights instead of the activations. By (re)parametrizing each weight vector as a scalar and a (unit norm) direction, it effectively decouples these quantities. 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 provide much insight into the underpinnings of 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. [18] 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 directly 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 [23]) 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.

5 Conclusions

In this work, we have investigated the roots of BatchNorm's effectiveness as a technique for training deep neural networks. We find that a 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 the 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 dispel a number of common misconceptions about BatchNorm but also bring us closer to a real understanding of this fundamental technique and the problem of training deep networks, in general. 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 true 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. And 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. Aleksander Mądry was supported in part by an Alfred P. Sloan Research Fellowship, a Google Research Award, and the NSF grant CCF-1553428.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *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? *arXiv preprint arXiv:1702.08591*, 2017.

[3] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[5] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[6] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv preprint arXiv:1611.04231*, 2016.

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

[8] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 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. *arXiv preprint arXiv:1502.03167*, 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. *arXiv preprint arXiv:1609.04836*, 2016.

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

[13] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 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 and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[17] 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*, 518(7540):529, 2015.

[18] 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.

[19] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010.

[20] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

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

[22] 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. *International Journal of Computer Vision (IJCV)*, 115(3), 2015.

[23] 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*, 2016.

[24] 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*, 529(7587):484–489, 2016.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1), 2014.

[27] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, 2013.

[28] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

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

[30] Yuxin Wu and Kaiming He. Group normalization. *arXiv preprint arXiv:1803.08494*, 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 – the *non-linear* VGG network, and the deep *linear* network (DLN). The VGG model is a natural choice for such analysis since it is a state-of-the-art deep network [6]. We study DLNs 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 the VGG network, a standard convolutional architecture [25], 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 [7]. 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.

[6]We choose to not experiment with ResNets [7] since they seem to provide several similar benefits to BatchNorm [6] and would make our study less clean.

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

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^t_j. The distribution D^t_j 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.)

Algorithm 1 "Noisy" BatchNorm

```
1:  %For some constants $n_m$, $n_v$, $r_m$, $r_v$, for some layer $l$
2:
3:  for $t = 1$ to $T$ do
4:      $a^t_{i,j} \leftarrow$ Batch-normalized activation for unit j and sample i
5:
6:      %Sample the parameters $(m^t_j, v^t_j)$ of $D^t_j$ from $D^*_j$
7:      for $j = 1$ to $M$ do
8:          $(\mu^t_j, \sigma^t_j) \overset{i.i.d.}{\sim} (U(-n_\mu, n_\mu), U(1, n_\sigma))$
9:      %Sample noise from $D^t_j$
10:     for $i = 1$ to $N$ do
11:         for $j = 1$ to $M$ do
12:             $(m^t_{i,j}, s^t_{i,j}) \overset{i.i.d.}{\sim} (U(\mu_j - r_\mu, \mu_j + r_\mu), \mathcal{N}(\sigma_j, r_\sigma))$
13:             $a^t_{i,j} \overset{i.i.d.}{\leftarrow} s^t_{i,j} \cdot a_{i,j} + m^t_{i,j}$
```

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 $[1/100, 30] \times$ *learning rate*. 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 explodes.

B Omitted Figures

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

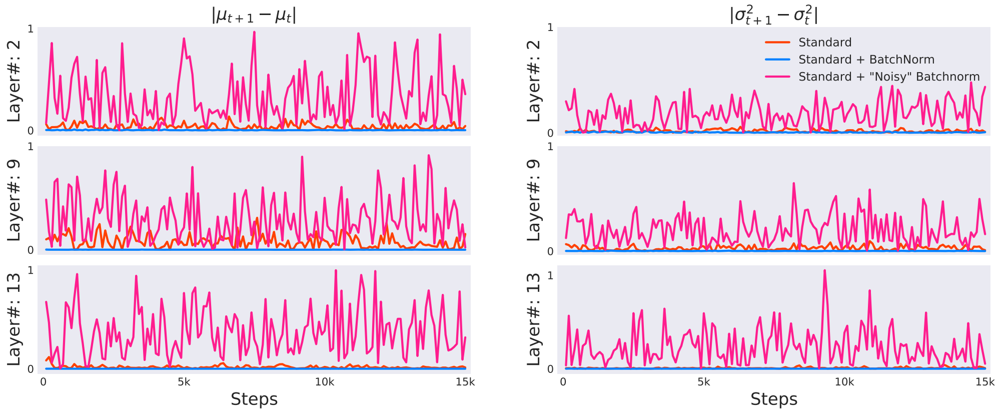


Figure 8: Comparison of change in the first two moments (mean and variance) of distributions of example activations for a given layer between two successive steps of the training process. Here we compare VGG networks trained without BatchNorm (Standard), with BatchNorm (Standard + BatchNorm) and with explicit "covariate shift" added to BatchNorm layers (Standard + "Noisy" BatchNorm). We observe that the difference in ICS between networks with and without BatchNorm layers is marginal. In fact, "noisy" BatchNorm layers have significantly higher ICS than standard networks, yet perform better from an optimization perspective (cf. Figure 2).

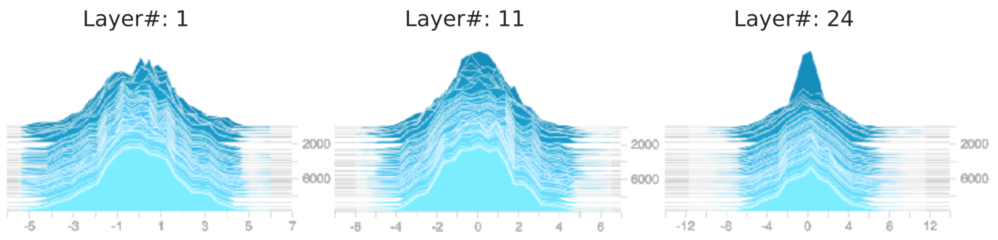
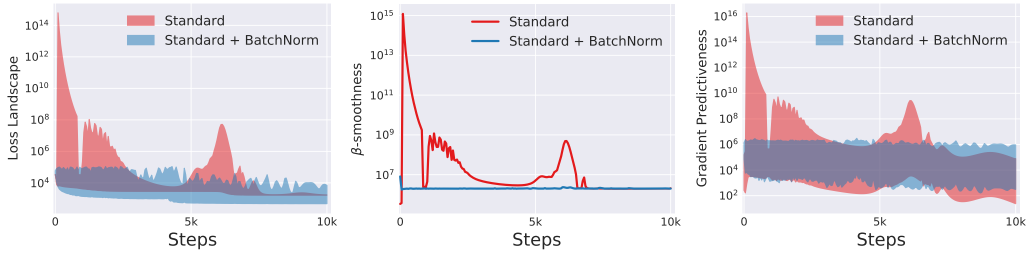


Figure 9: Distributions of activations from different layers of a 25-Layer deep linear network. (Here we sample a random activation from a given layer to visualize its distribution.)

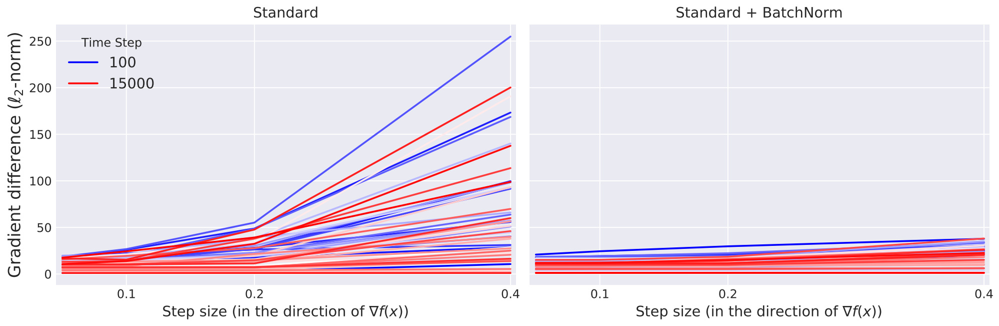


(a) loss landscape

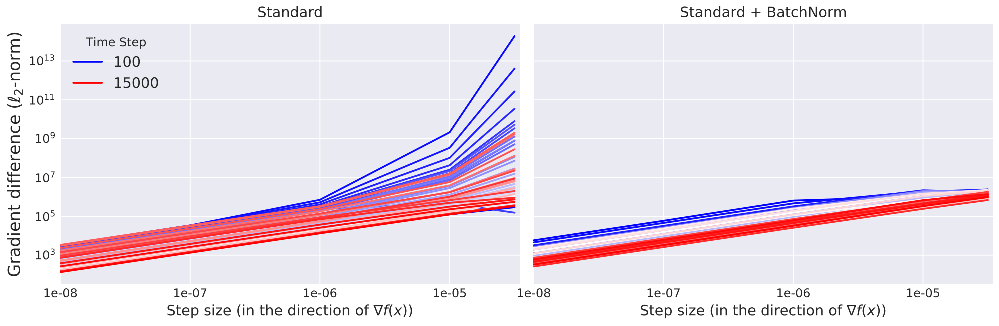
(b) "effective" β-smoothness

(c) gradient predictiveness

Figure 10: 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 (c) as we move in the gradient direction. The "effective" β-smoothness (b) 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 11: 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.

C Proof of Theorems

We now prove the previously stated theorems regarding the better landscape (in terms of Lipschitz constant and β-smoothness) induced by the insertion of a batch normalization (BatchNorm) layer.

For the general setup, we refer the reader back to Section 3.3. Crucially, we denote an element of activation i and batch index b as $x_i^{(b)}$. Here x refers to the input from another layer (or potentially the input layer itself). (W, b) are the weights and biases parameterizing the first linear layer. z is defined as the output of this linear layer, and y is defined such that $y = z$ in the standard case and $\hat{y} = BN(z)$ in the Batch Normalized network. We consider any arbitrary loss $\mathcal{L}(y)$ appearing after y. Our first result shows that the output of the linear layer after batch-normalization is better behaved in terms of Lipschitz constant than without BatchNorm. Then, we show that the overall smoothness of the loss landscape induced by BatchNorm layers. Throughout the proofs, we make use of the following defined variables for convenience:

$$\alpha_{ij} = \sum_{b=1}^{m} x_i^{(b)} z_j^{(b)}$$

$$\sigma_{xi} = \sigma(x_i)$$

$$\sigma_{zj} = \sigma(z_j)$$

We also use a set of facts that are useful for proving both the following theorems:

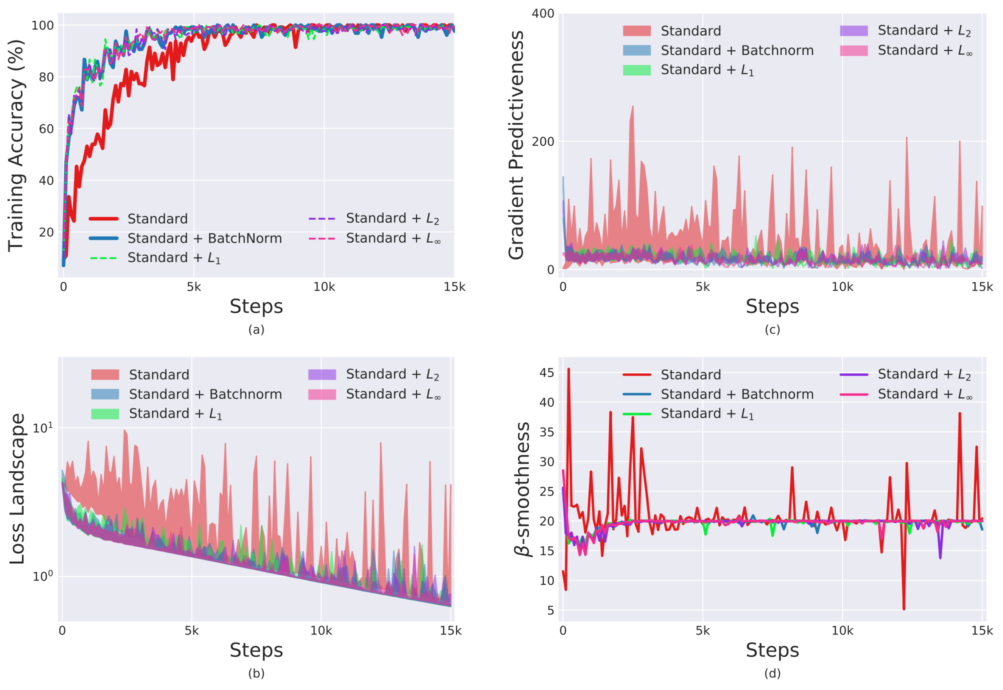


Figure 12: Evaluation of VGG networks trained with different ℓ_p normalization strategies discussed in Section 3.4. *(a):* Comparison of the training performance of the models. *(b, c, d):* Evaluation of the smoothness of optimization landscape in the various models. At a particular training step, we 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.

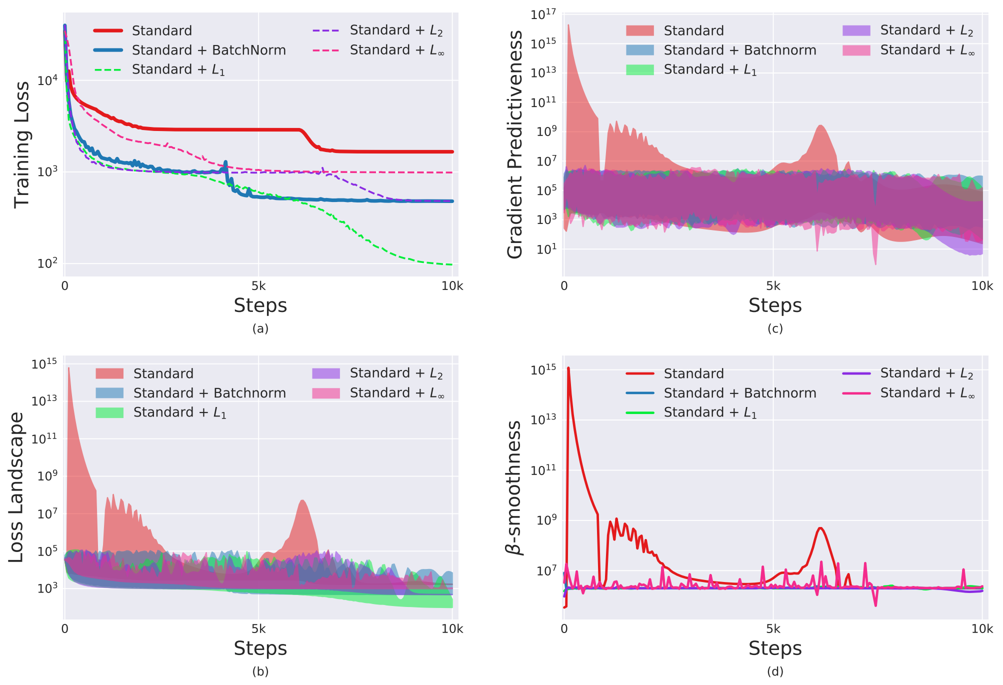


Figure 13: Evaluation of deep linear networks trained with different ℓ_p normalization strategies. We observe that networks with any normalization strategy have improved performance and smoothness of the loss landscape over standard training. Details of the plots are the same as Figure 12 above.

Fact C.1 (Gradient through BatchNorm)**.** *The gradient through BN and another function f, $\frac{\partial f}{\partial z^{(b)}}$ where $L = L(y)$, $y = BN(z)$, $z^{(b)}$ are scalar elements of a batch of size m and variance σ_z^2*

$$\frac{1}{m\sigma_x}\left(m\frac{\partial f}{\partial x^{(b)}} - \sum_{k=1}^{m}\frac{\partial f}{\partial x^{(k)}} - f\right)$$

Using this derived gradient, we can derive a few convienient gradients that we apply in the proofs.

Fact C.2 (Convenient Gradients)**.** *For the special case where z_i is mean-zero (which is the case we consider, for simplicity), a few convenient gradients of BN are given as*

$$\frac{\partial y^{(b)}}{\partial z^{(k)}} = \frac{1}{\sigma_z}\left(\boldsymbol{1}[b=k] - \frac{1}{m} - \frac{1}{m}y^{(b)}y^{(k)}\right) \tag{1}$$

Now, consider the case where z is a vector, which is the result of a matrix multiplication $z = Wx + b$, such that $z_i = \sum_j W_{ij} \cdot x_j$. Then,

$$\frac{\partial y_j^{(b)}}{\partial W_{ij}} = \sum_{k=1}^{m}\frac{\partial y_j^{(b)}}{\partial z_j^{(k)}}\frac{\partial z_j^{(k)}}{\partial W_{ij}} = \frac{1}{\sigma_z}\left(x_i^{(b)} - \frac{\alpha_{ij}}{m}y_j^{(b)}\right) \tag{2}$$

We use these observations in proving our main two theorems:

Theorem C.3 (The effect of BatchNorm on the outputs' sensitivity)**.** *We have that, for any i and j,*

$$\sum_{l=1}^{m}\left(\frac{\partial y_j^l}{\partial W_{ij}}\right)^2 = m\sigma(x_i)^2 \qquad \text{and} \qquad \sum_{l=1}^{m}\left(\frac{\partial \hat{y}_j^l}{\partial W_{ij}}\right)^2 = \frac{1}{\sigma(y_j)^2}\left(m\sigma(x_i)^2 - \alpha_{ij}^2\right),$$

where α_{ij} is the correlation between x_is and y_js.

Proof. To prove this, the normal case follows straightforwardly:

$$\sum_{b=1}^{m}\left(\frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}}\right)^2 = \sum_{b=1}^{m}\left(x_i^{(b)}\right)^2 = m\sigma_{xi}^2 \tag{3}$$

We can simply apply the gradient we derived in (1) for the BatchNorm case:

$$\begin{aligned}
\sum_{b=1}^{m}\left(\frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}}\right)^2 &= \sum_{b=1}^{m}\left(\frac{mx_i^{(b)} - \alpha_{ij}\hat{y}_j^{(b)}}{m\sigma_j}\right)^2 && (4)\\
&= \frac{1}{\sigma_{zj}^2}\sum_{b=1}^{m}\left(x_i^{(b)}\right)^2 - \frac{1}{m}2\alpha_{ij}\hat{y}_j^{(b)}x_i^{(b)} + \frac{1}{m^2}\alpha_{ij}^2\left(\hat{y}_j^{(b)}\right)^2 && (5)\\
&= \frac{1}{\sigma_{zj}^2}\left(m\sigma_{xi}^2 - \frac{2}{m}\alpha_{ij}^2 + \frac{1}{m^2}\alpha_{ij}^2(m)\right) && (6)\\
&= \frac{1}{\sigma_{zj}^2}\left(m\sigma_{xi}^2 - \frac{1}{m}\alpha_{ij}^2\right) && (7)
\end{aligned}$$

□

Now, we proceed to the second theorem, which makes a statement regarding the behaviour of the overall landscape, assuming some degree of correlation between the input and the gradients. We prove an alternate statement of the theorem here which fully describes the loss landscape; simple rearranging yields the statement in the main text:

Theorem C.4. *For a given i, let j be such that $\sigma(y_j)^2 \geq \frac{(1+\sqrt{|c_{ij}|})^2}{|c_{ij}|-\varepsilon}$, for some $\varepsilon > 0$. We have that*

$$\left(\frac{\partial \widehat{\mathcal{L}}}{\partial W_{ij}}\right)^2 \leq \left(1 - |c_{ij}|^{-1} \cdot \varepsilon\right) \cdot \left(\frac{\partial \mathcal{L}}{\partial W_{ij}}\right)^2,$$

where $\widehat{\mathcal{L}}$ is the loss function of the network with the BatchNorm layer inserted.

Proof. For the normal network,

$$\frac{\partial G}{\partial W_{ij}} = \sum_{b=1}^{m} \frac{\partial G}{\partial z_j^{(b)}} \frac{\partial z_j^{(b)}}{\partial W_{ij}} = \sum_{b=1}^{m} \frac{\partial G}{\partial z_j^{(b)}} x_i^{(b)} \tag{8}$$

Thus, using the correlation assumption, we can bound the squared magnitude as:

$$\left(\frac{\partial \mathcal{L}}{\partial W_{ij}}\right)^2 = c \sum_{b=1}^{m} \left(\frac{\partial G}{\partial z_j^{(b)}}\right)^2 \sum_{b=1}^{m} \left(x_i^{(b)}\right)^2 = c\sigma_{x_i}^2 \sum_{b=1}^{m} \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\right)^2 \tag{9}$$

In the case of Batch Normalization:

$$\frac{\partial G}{\partial W_{ij}} = \sum_{b=1}^{m} \frac{\partial G}{\partial \hat{y}_j^{(b)}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \tag{10}$$

$$= \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \left(\frac{1}{\sigma_j}\right) \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)}\right) \tag{11}$$

$$\sigma_j^2 \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\right)^2 = \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)}\right)^2 - 2\frac{\alpha_{ij}}{m} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)} \tag{12}$$

$$+ \frac{\alpha_{ij}^2}{m^2} \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right)^2$$

Now, by assumption, we let c be the correlation between $\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}$ and $x_i^{(b)}$, such that we can simplify the preceding. To do this simplification, we introduce the following notation:

$$S_j = \sum_{b=1}^{m} \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\right)^2$$

Now, using this:

$$= \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)}\right)^2 - 2\frac{\alpha_{ij}}{m} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)} + \frac{\alpha_{ij}^2}{m^2} \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right)^2 \tag{13}$$

$$= cS_j m\sigma_{xi}^2 - 2\frac{\alpha_{ij}}{m} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)} + \frac{\alpha_{ij}^2}{m^2} \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right)^2 \tag{14}$$

$$\leq cS_j m\sigma_{xi}^2 + 2\left|\frac{\alpha_{ij}}{m} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} x_i^{(b)} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right| + \frac{\alpha_{ij}^2}{m^2} \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right)^2 \tag{15}$$

$$\leq cS_j m\sigma_{xi}^2 + 2\frac{\alpha_{ij}}{m} S_j (m\sigma_{xi}) + \frac{\alpha_{ij}^2}{m^2} \left(\sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right)^2 \tag{16}$$

$$\leq cS_j m\sigma_{xi}^2 + 2\sqrt{c}\alpha_{ij} S_j \sigma_{xi} + \frac{\alpha_{ij}^2}{m} S_j \tag{17}$$

Once again, recall the squared magnitude in a normal network:

$$mc\sigma_{x_i}^2 \sum_{b=1}^{m} \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \right)^2 = mc\sigma_{xi}^2 S_j$$

To conclude the proof of the first statement, note that if

$$\sigma_{yj}^2 \geq \frac{(1+\sqrt{c})^2}{c-\epsilon}$$

for some $\epsilon > 0$, then

$$\left(\frac{\partial \mathcal{L}}{\partial W_{ij}} \right)^2_{normal} - \left(\frac{\partial \mathcal{L}}{\partial W_{ij}} \right)^2_{bn} \geq \epsilon m\sigma_{xi}^2 \sum_{b=1}^{m} \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j} \right)^2 \geq \frac{\epsilon}{c} \cdot \left(\frac{\partial \mathcal{L}}{\partial W_{ij}} \right)^2_{normal}$$

This concludes the proof. □

We also show that the increased Lipschitzness is not simply a result of rescaling the loss (for more context see Section 3.3). We restate the lemma and give its proof here:

Lemma C.5. *Consider the setup of Theorem 3.2 (0-mean input and $(\gamma, \beta) = (1, 0)$), where $\mathcal{L}$ is the ℓ_2-norm loss on zero-mean labels ℓ. Then, as long as an optimum is ε-attainable with BN*[8]*, we have that for any W such that $\mathcal{L}_W(x) < \varepsilon$, the same W satisfies:*

$$\widehat{\mathcal{L}}_W(x) < 3\varepsilon \left(\frac{1+\varepsilon}{1-2\varepsilon} \right)$$

Proof. We use attainability under BN to first bound the variance of the mean-zero targets ℓ:

$$||BN(W'x) - \ell||_2 < \varepsilon \tag{18}$$
$$||BN(W'x)||_2 - ||\ell||_2 < \varepsilon \tag{19}$$
$$||\ell||_2 > 1 - \varepsilon \tag{20}$$
$$\text{Similarly } ||\ell||_2 < 1 + \varepsilon \tag{21}$$
$$\tag{22}$$

Then, we use the optimality of W in the standard setting to bound the standard deviation $||Wx||_2$:

$$||Wx - \ell||_2 < \varepsilon \tag{23}$$
$$||Wx||_2 > ||\ell||_2 - \varepsilon \tag{24}$$
$$> 1 - 2\varepsilon \tag{25}$$
$$||Wx||_2 < ||\ell||_2 + \varepsilon \tag{26}$$
$$||Wx||_2 < 1 + 2\varepsilon \tag{27}$$

Applying these we can prove the result:

$$||BN(Wx) - \ell||_2 = ||\frac{Wx}{||Wx||_2} - \ell|| \tag{28}$$
$$= \frac{1}{||Wx||_2} ||Wx - ||Wx||_2 \ell|| \tag{29}$$
$$= \frac{1}{||Wx||_2} \left(||Wx - \ell|| + |1 - ||Wx||_2 ||\ell||_2 \right) \tag{30}$$
$$= \frac{1}{1-2\varepsilon} \left(\varepsilon + 2\varepsilon(1+\varepsilon) \right) \tag{31}$$
$$= \frac{3\varepsilon + 2\varepsilon^2}{1-2\varepsilon} \tag{32}$$
$$< 3\varepsilon \left(\frac{1+\varepsilon}{1-2\varepsilon} \right) \tag{33}$$

□

[8]i.e. so long as $\exists\, W'$ s.t. $\widehat{\mathcal{L}}_{W'}(x) < \varepsilon$

Now, we move to the second statement, regarding the elements of the Hessian $\frac{\partial \mathcal{L}}{\partial W_{ij} \partial W_{kl}}$. We tackle this with a similar method, by first explicitly expanding by definition. Although the main theorem simply gives bounds in terms of κ and γ, we prove an expanded yet equivalent version of the theorem here which fully specifies each variable and contains no asymptotics. Setting $\kappa_{ij} = 1 + \sqrt{|d_{ij}|}$ and $\gamma_{ijkl} = f(i, j, k, l)$ in the following, and rearranging, yields the theorem given in the main text:

Theorem C.6. *For a given (i, k) representing input indices, if feature indices (j, l) are such that*

$$\sigma(z_j)\sigma(z_l) \geq \frac{(1 + \sqrt{d_{kl}})(1 + \sqrt{d_{ij}}) + f(j, l, i, k)}{\sqrt{d_{kl} d_{ij}} - \varepsilon},$$

then we have that

$$\left| \frac{\partial^2 \hat{L}}{\partial W_{ij} \partial W_{kl}} \right| \leq \left(1 - \frac{\varepsilon}{\sqrt{d_{ij} d_{kl}}} \right) \left| \frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kl}} \right|,$$

where

$$f(j, l, i, k) = \begin{cases} 0 & \textit{if } j \neq l \\ 6 ||\partial \mathcal{L} / \partial \hat{y}_j||_2 & \textit{if } l = j \textit{ and } k \neq i \\ 3 ||\partial \mathcal{L} / \partial \hat{y}_j||_2 & \textit{if } (i, j) = (k, l) \end{cases}$$

Proof.

$$\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kl}} = \frac{\partial}{\partial W_{kl}} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \tag{34}$$

$$= \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \right) \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} + \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \tag{35}$$

$$= \sum_{b=1}^{m} \left(\frac{\partial}{\partial \hat{y}_j^{(b)}} \frac{\partial \mathcal{L}}{\partial W_{kl}} \right) \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} + \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \tag{36}$$

$$= \sum_{b=1}^{m} \left(\frac{\partial}{\partial \hat{y}_j^{(b)}} \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_l^{(q)}} \cdot \frac{\partial \hat{y}_l^{(q)}}{\partial W_{kl}} \right) \right) \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} + \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \tag{37}$$

We approach each term in this sum separately, beginning with the first:

$$\sum_{b=1}^{m} \left(\frac{\partial}{\partial \hat{y}_j^{(b)}} \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_l^{(q)}} \cdot \frac{\partial \hat{y}_l^{(q)}}{\partial W_{kl}} \right) \right) \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \tag{38}$$

$$= \sum_{b=1}^{m} \left(\frac{\partial}{\partial \hat{y}_j^{(b)}} \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_l^{(q)}} \cdot \frac{\partial \hat{y}_l^{(q)}}{\partial W_{kl}} \right) \right) \frac{1}{\sigma_{zj}} \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)} \right) \tag{39}$$

$$= \frac{1}{\sigma_{zl} \sigma_{zj}} \sum_{b=1}^{m} \left(\frac{\partial}{\partial \hat{y}_j^{(b)}} \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_l^{(q)}} \cdot \left(x_k^{(q)} - \frac{\alpha_{kl}}{m} \hat{y}_l^{(q)} \right) \right) \right) \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)} \right) \tag{40}$$

Now, we must consider two cases. If $j = l$, then:

$$= \frac{1}{\sigma_{zl} \sigma_{zj}} \sum_{b=1}^{m} \left(\sum_{q=1}^{m} \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_l^{(q)}} \cdot \left(x_k^{(q)} - \frac{\alpha_{kl}}{m} \hat{y}_l^{(b)} \right) - \frac{1}{m} \sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(q)}} \cdot x_k^{(b)} \hat{y}_l^{(q)} \right) \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)} \right) \tag{41}$$

Otherwise,

$$= \frac{1}{\sigma_{zl} \sigma_{zj}} \sum_{b=1}^{m} \left(\sum_{q=1}^{m} \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_l^{(q)}} \cdot \left(x_k^{(q)} - \frac{\alpha_{kl}}{m} \hat{y}_l^{(b)} \right) \right) \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)} \right) \tag{42}$$

For convenience, we define the following coefficient:

$$X_{ijkl} = \sum_{b,q=1}^{m} \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_l^{(q)}} \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} \hat{y}_j^{(b)} \right) \left(x_k^{(q)} - \frac{\alpha_{kl}}{m} \hat{y}_l^{(q)} \right)$$

Then, (41) can be written as, when $j = l$:

$$\frac{1}{\sigma_{zj}^2} \left(X_{ijkl} - \frac{1}{m} \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(q)}} \cdot \hat{y}_j^{(q)} \right) \left(\sum_{b=1}^{m} x_i^{(b)} x_k^{(b)} \right) + \left(\frac{\alpha_{ij}\alpha_{kj}}{m^2} \right) \left(\sum_{q=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(q)}} \cdot \hat{y}_j^{(q)} \right) \right) \tag{43}$$

And otherwise simply

$$\sigma_{zj}^{-1} \sigma_{zl}^{-1} X_{ijkl}. \tag{44}$$

Now we consider the second term of (37), which we can expand and simplify:

$$\begin{aligned} \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} &= \frac{1}{\sigma_j} \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \left(x_i^{(b)} - \frac{\alpha_{ij}}{m} y_j^{(b)} \right) \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (45) \\ &= -\frac{1}{\sigma_{zj}} \sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \left(\frac{\alpha_{ij}}{m} y_j^{(b)} \right) \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (46) \\ & && (47) \end{aligned}$$

As with the first term, we to consider the case where $j = l$, and the case where $j \neq l$. For the former,

$$\begin{aligned} = -\frac{1}{m\sigma_j} \sum_{b=1}^{m} \left(y_j^{(b)} \frac{\partial \alpha_{ij}}{\partial W_{kl}} + \alpha_{ij} \frac{\partial y_j^{(b)}}{\partial W_{kl}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (48) \\ && (49) \end{aligned}$$

To calculate the derivative of α_{ij} with respect to W_{kl}, we note that this is 0 for $j \neq l$, so we instead consider the gradient of α_{ij} with respect to W_{kj}:

$$\begin{aligned} \frac{\partial \alpha_{ij}}{\partial W_{kl}} = \sum_{b=1}^{m} \frac{\partial \alpha_{ij}}{\partial y_j^{(b)}} \frac{\partial y_j^{(b)}}{\partial W_{kj}} &= \frac{1}{\sigma_{zj}} \sum_{b=1}^{m} x_i^{(b)} \left(x_k^{(b)} - \frac{\alpha_{kj}}{m} y_j^{(b)} \right) && (50) \\ &= \frac{1}{\sigma_{zj}} \left(\left(\sum_{b=1}^{m} x_i^{(b)} x_k^{(b)} \right) - \frac{\alpha_{ij}\alpha_{kj}}{m} \right) && (51) \end{aligned}$$

Now, we can apply this to get our two simplified cases. For notational convenience, we denote

$$\beta_{i,k} = \frac{1}{m} \sum_{b=1}^{m} x_i^{(b)} x_k^{(b)},$$

the correlation between x_i and x_k across the batch.

Now, first note that if $j \neq l$, our entire term is simply 0! So we consider the case where $j = l$:

$$\begin{aligned} &\sum_{b=1}^{m} \left(\frac{\partial}{\partial W_{kl}} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{ij}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (52) \\ &= -\frac{1}{m\sigma_{zj}} \sum_{b=1}^{m} \left(\hat{y}_j^{(b)} \frac{\partial \alpha_{ij}}{\partial W_{kl}} + \alpha_{ij} \frac{\partial \hat{y}_j^{(b)}}{\partial W_{kl}} \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (53) \\ &= -\frac{1}{m\sigma_{zj}^2} \sum_{b=1}^{m} \left(\hat{y}_j^{(b)} \left[m\beta_{ik} - \frac{\alpha_{ij}\alpha_{kj}}{m} \right] + \alpha_{ij} \left[x_k^{(b)} - \frac{\alpha_{kj}}{m} \hat{y}_j^{(b)} \right] \right) \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (54) \\ &= -\frac{\beta_{ik}}{\sigma_{zj}^2} \sum_{b=1}^{m} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)} + \frac{2\alpha_{ij}\alpha_{kj}}{m^2\sigma_{zj}^2} \sum_{b=1}^{m} \hat{y}_j^{(b)} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} - \frac{\alpha_{ij}}{m\sigma_{zj}^2} \sum_{b=1}^{m} x_k^{(b)} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} && (55) \end{aligned}$$

Reiterating the first term with β notation:

$$\frac{1}{\sigma_{zj}^2}\left(X_{ijkl} - \frac{1}{m}\left(\sum_{q=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(q)}} \cdot \hat{y}_j^{(q)}\right) m\beta_{ik} + \left(\frac{\alpha_{ij}\alpha_{kj}}{m^2}\right)\left(\sum_{q=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(q)}} \cdot \hat{y}_j^{(q)}\right)\right) \tag{56}$$

Summing these two together yields (recall that we are still in the case where $j = l$):

$$\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kj}} = \frac{1}{\sigma_{zj}^2}\Bigg(X_{ijkl} - 2\beta_{ik}\left(\sum_{b=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right) \tag{57}$$

$$+ 3\frac{\alpha_{ij}\alpha_{kj}}{m^2}\left(\sum_{b=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right) - \frac{\alpha_{ij}}{m}\sum_{b=1}^m x_k^{(b)} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\Bigg)$$

$$\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kj}} = \frac{1}{\sigma_{zj}^2}\left(X_{ijkl} - \left(2\beta_{ik} - 3\frac{\alpha_{ij}\alpha_{kj}}{m^2}\right)\left(\sum_{b=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right) - \frac{\alpha_{ij}}{m}\sum_{b=1}^m x_k^{(b)} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\right) \tag{58}$$

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kj}}\right| \leq \frac{1}{\sigma_{zj}^2}\Bigg(|X_{ijkl}| + 2\sigma_{xi}\sigma_{xk}\left(\sum_{b=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right) \tag{59}$$

$$+ 3\sigma_{xi}\sigma_{xk}\left(\sum_{b=1}^m \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}} \hat{y}_j^{(b)}\right) + \sigma_{xi}\sum_{b=1}^m x_k^{(b)} \frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\Bigg)$$

$$\text{Now, let} \quad G := \sqrt{\sum_{b=1}^m \left(\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(b)}}\right)^2} \tag{60}$$

We can use this to bound the quantity as follows:

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kj}}\right| \leq \frac{1}{\sigma_{zj}^2}\left(|X_{ijkl}| + 6m\sigma_{xi}\sigma_{xk}G\right) \tag{61}$$

Note that for $i = k$ this constant factor in the bound can be improved to 3 rather than 6, since the β correlation terms are necessarily positive and the α is squared, so one of the terms in the subtraction expression in Eq. 58 cancels. However, even this loose bound is enough to give us smoothness guarantees. The final step is to write out X_{ijkl} for $j = l$:

$$X_{ijkj} = \sum_{b,q=1}^m \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}}\left(x_i^{(b)} - \frac{\alpha_{ij}}{m}\hat{y}_j^{(b)}\right)\left(x_k^{(q)} - \frac{\alpha_{kj}}{m}\hat{y}_j^{(q)}\right) \tag{62}$$

Now is when we take advantage of the specified assumption, which is that:

$$\left(\sum_{b=1}^m \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}} x_i^{(b)}\right)^2 = d_{ij}\sum_{b=1}^m \left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}}\right)^2 \sum_{b=1}^m \left(x_i^{(b)}\right)^2$$

Using these constants d_{ij}, we can bound X_{ijkj} as:

$$|X_{ijkj}| \leq m\sqrt{\sum_{b,q=1}^m \left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}}\right)^2}\left(\sqrt{d_{ij}d_{kj}}\sigma_{xi}\sigma_{xk} + \sqrt{d_{ij}}\sigma_{xi}\sigma_{xk} + \sqrt{d_{kj}}\sigma_{xi}\sigma_{xk} + \sigma_{xi}\sigma_{xk}\right) \tag{63}$$

$$= m\sigma_{xk}\sigma_{xi}\sqrt{\sum_{b,q=1}^m \left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}}\right)^2}\left(1 + \sqrt{d_{ij}}\right)\left(1 + \sqrt{d_{kj}}\right) \tag{64}$$

Final bound for $j = l$ case Now, to conclude the proof we examine a standard network and compare. In a standard network, if $j = l$, then:

$$\frac{\partial^2 \mathcal{L}}{\partial W_{ij} \partial W_{kj}} = \sum_{b,q=1}^m \frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)} \partial \hat{y}_j^{(q)}} x_i^{(b)} x_k^{(q)},$$

and thus by our assumption we have:

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kj}}\right| = m\sqrt{d_{ij}\cdot d_{kj}}\cdot \sigma_{xi}\sigma_{xk}\sqrt{\sum_{b,q=1}^{m}\left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)}\partial \hat{y}_j^{(q)}}\right)^2}$$

Subtracting the two expressions yields a difference of:

$$m\sigma_{xk}\sigma_{xi}\sqrt{\sum_{b,q=1}^{m}\left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)}\partial \hat{y}_j^{(q)}}\right)^2}\left(\sqrt{d_{ij}d_{kj}} - \frac{(1+\sqrt{d_{ij}})(1+\sqrt{d_{kj}})+6G}{\sigma_{zj}^2}\right)$$

This tells us that if

$$\sigma_{zj}^2 \geq \frac{(1+\sqrt{d_{kj}})(1+\sqrt{d_{ij}})+6\left\|\frac{\partial \mathcal{L}}{\partial \hat{y}_j^{(\cdot)}}\right\|}{\sqrt{d_{kj}d_{ij}}-\varepsilon},$$

then

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kj}}\right|_{\text{bn}} \leq \left(1-\frac{\varepsilon}{\sqrt{d_{ij}d_{kj}}}\right)\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kj}}\right|_{\text{normal}}$$

Final bound for $j \neq l$ case As we noted in equation 44, if $j \neq l$, the gradient of the first term is

$$\sigma_{zj}^{-1}\sigma_{zl}^{-1}X_{ijkl},$$

and the gradient of the second term is 0, and so in the BN network,

$$\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kl}} = \sigma_{zj}^{-1}\sigma_{zl}^{-1}X_{ijkl}.$$

By exactly the same logic shown above for the $l = j$ case, we have that

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kl}}\right|_{bn} \leq \frac{m\sigma_{xi}\sigma_{xk}}{\sigma_{zj}\sigma_{zl}}\sqrt{\sum_{b,q=1}^{m}\left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)}\partial \hat{y}_l^{(q)}}\right)^2}(1+\sqrt{d_{ij}})(1+\sqrt{d_{kl}})$$

In a vanilla NN, we get the same second derivative as the previous case, namely

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kl}}\right|_{\text{normal}} = m\sqrt{d_{ij}\cdot d_{kl}}\cdot \sigma_{xi}\sigma_{xk}\sqrt{\sum_{b,q=1}^{m}\left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)}\partial \hat{y}_l^{(q)}}\right)^2}$$

Subtracting these yields

$$m\sigma_{xk}\sigma_{xi}\sqrt{\sum_{b,q=1}^{m}\left(\frac{\partial^2 \mathcal{L}}{\partial \hat{y}_j^{(b)}\partial \hat{y}_l^{(q)}}\right)^2}\left(\sqrt{d_{ij}d_{kl}} - \frac{(1+\sqrt{d_{ij}})(1+\sqrt{d_{kl}})}{\sigma_{zj}\sigma_{zl}}\right)$$

Which tells us that if $j \neq l$, and

$$\sigma_{zj}\sigma_{zl} \geq \frac{(1+\sqrt{d_{kl}})(1+\sqrt{d_{ij}})}{\sqrt{d_{kj}d_{ij}}-\varepsilon},$$

$$\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kl}}\right|_{\text{bn}} \leq \left(1-\frac{\varepsilon}{\sqrt{d_{ij}d_{kl}}}\right)\left|\frac{\partial^2 \mathcal{L}}{\partial W_{ij}\partial W_{kl}}\right|_{\text{normal}}$$

This concludes the proof of the thorem.

□