Understanding Batch Normalization

Johan Bjorck *
Cornell University
njb225@cornell.edu

Carla Gomes
Cornell University
gomes@cs.cornell.ed

Bart Selman
Cornell University
selman@cs.cornell.edu

Abstract

Batch normalization is a ubiquitous deep learning technique that normalizes activations in intermediate layers. It is associated with improved accuracy and faster learning, but despite its enormous success there is little consensus regarding why it works. We aim to rectify this and take an empirical approach to understanding batch normalization. Our primary observation is that the higher learning rates that batch normalization enables have a regularizing effect that dramatically improves generalization of normalized networks, which is both demonstrated empirically and motivated theoretically. We show how activations become large and how the convolutional channels become increasingly ill-behaved for layers deep in unnormalized networks, and how this results in larger input-independent gradients. Beyond just gradient scaling, we demonstrate how the learning rate in unnormalized networks is further limited by the magnitude of activations growing exponentially with network depth for large parameter updates, a problem batch normalization trivially avoids. Motivated by recent results in random matrix theory, we argue that ill-conditioning of the activations is due to fluctuations in random initialization, shedding new light on classical initialization schemes and their consequences.

1 Introduction

Batch normalization extends the successful idea of normalizing the input to neural networks [26] and additionally normalizes the activations in intermediate layers to have zero mean and unit variance [20] . This technique enables faster training, higher learning rates, improves accuracy and has a regularizing effect [20]. Perhaps not surprisingly, it has had a profound impact on computer vision [18] [15] and beyond [42] [1]. Despite its ubiquity, there is little understanding regarding why it works. The original paper [20] claims that BN alleviates internal covariate shift – the tendency of the input distribution to different layers to shift during training. However, other explanations such as improved the stability of concurrent updates [11] or conditioning [39] have been given.

Inspired by recent empirical work providing insight into deep learning [49] [22] [33] , we aim to rectify this and take a principled empirical approach to understanding BN. Our primary contributions are: **1)** We show empirically and motivate theoretically that the benefits of batch normalization are primarily mediated via the higher learning rate. It is demonstrated how a larger learning rate increases the regularizing effect of SGD, drawing upon recent results in generalization. **2)** We demonstrate how the activations grow with network depth and how this results in the gradient-information becomes less input-sensitive for unnormalized networks. This leads to large gradients which limits possible learning rates. **3)** We show how learning rate in unnormalized networks is further limited by divergence due the magnitude of activations growing exponentially with network depth, a problem batch normalization trivially avoids. **4)** We argue that the dramatic growth of activations with network depth is a result of fluctuations in the random initialization, using recent results from random matrix theory.

---

*Corresponding author. Article currently under review.

1.1 The Batch Normalization Algorithm

As in [20], we consider batch normalization for convolutional neural networks. Both the input $\texttt{INPUT}(b, c, x, y)$ and output $\texttt{OUTPUT}(b, c, x, y)$ are four-dimensional tensors, the dimensions corresponding to examples within a batch b, feature-dimension c and two spatial dimensions x, y respectively. We will refer to the feature-dimensions as the channels, in the input image they correspond to the RGB channels. Batch normalization normalizes the activations $\texttt{INPUT}(b, c, x, y)$ belonging to channel c by subtracting the mean $\mu_c = \sum_{b,x,y} \texttt{INPUT}(b, c, x, y)$ and dividing by the standard deviation σ_c (plus ϵ for numerical stability) which is calculated similarly. During testing, running averages of the mean and variances are used. Normalization is followed by a channel-wise affine transformation parametrized by γ_c, β_c as follows

$$\texttt{OUTPUT}(b, c, x, y) \leftarrow \gamma_c \frac{\texttt{INPUT}(b, c, x, y) - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c \qquad \forall\, b, c, x, y \tag{1}$$

1.2 Experimental Setup

To investigate batch normalization we will use an experimental setup of the original Resnet paper[15]: image classification on CIFAR10 [24] with a 110 layer Resnet. We use SGD with momentum and weight decay, employ standard data augmentation and image preprocessing techniques and decrease learning rate when learning plateaus, all as in [15] and with the same parameter values. The original network can be trained with initial learning rate 0.1 over 165 epochs, but for experiments with batch normalization turned off such training fails. We always report the best results among initial learning rates from $\{0.1, 0.003, 0.001, 0.0003, 0.0001, 0.00003\}$ and use enough epochs such that learning plateaus. For further details, we refer to Appendix B.

2 Disentangling the benefits of BN

When we remove batch normalization from Resnet the initial learning rate needs to be decreased to 0.0001 for convergence and training takes 2400 epochs. We refer to this architecture as an unnormalized network, and as we see in Figure 1 it performs poor compared to the original Resnet architecture. Batch normalization indeed gives faster training, higher accuracy and enable higher learning rates. To disentangle how these benefits are related, we train a batch normalized network using the learning rate and the number of epochs of an unnormalized network, as well as an initial learning rate of 0.003 which requires 1320 epochs for training. These results are also illustrated in Figure 1, where we see that a batch normalized networks performs no better than an unnormalized network with such a learning schedule, and has a lower training loss indicating overfitting. A learning rate of 0.003 gives results in between these extremes. This suggests that it is the higher learning rate that batch normalization enables that mediates the majority of benefits of batch normalization; it improves regularization, accuracy and gives faster convergence.

2.1 Learning rate and generalization

To explain these observations we consider a simple model of SGD; the loss function $f(x)$ is the sums over the losses of individual examples in our dataset $f(x) = \frac{1}{N} \sum f_i(x)$. We model SGD as sampling a set B of examples from the dataset with replacements, and then with learning rate α estimating the gradient step as $\alpha \nabla f(x) \approx \frac{\alpha}{|B|} \sum_{i \in B} \nabla f_i(x)$. If we subtract and add $\alpha \nabla f(x)$ from this expression we can divide the estimated gradient $\nabla_{SGD}(x)$ into the true gradient, and a noise term

$$\alpha \nabla_{SGD}(x) = \underbrace{\alpha \nabla f(x)}_{\text{gradient}} + \underbrace{\frac{\alpha}{|B|} \sum_{i \in B} \left(\nabla f_i(x) - \nabla f(x) \right)}_{\text{error term}}$$

Now we note, that since we sample uniformly we have $\mathbb{E}\left[\frac{\alpha}{|B|} \sum_{i \in B} \left(\nabla f_i(x) - \nabla f(x) \right) \right] = 0$. Thus the gradient estimate is unbiased, but will typically be noisy. Let define an architecture dependent
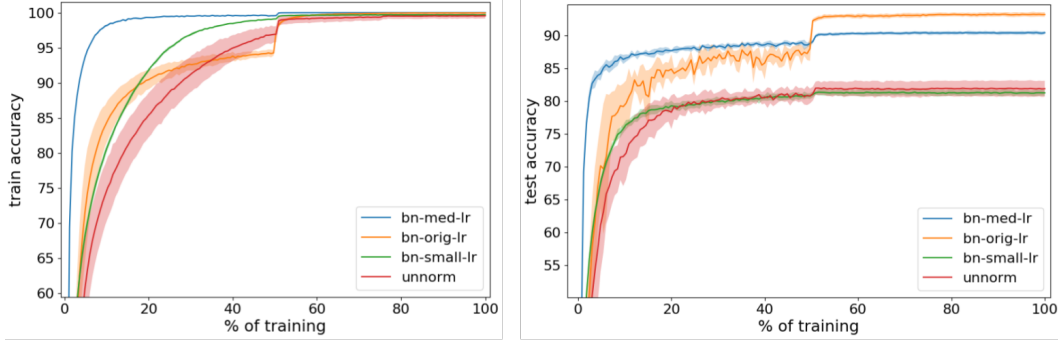
Figure 1: We here train a batch normalized 110-layer Resnet with the learning rate (0.0001) and the number of epochs (2400) required to train the same network without batch normalization. All experiments are repeated five times, with the shaded region corresponding to standard deviation and the thick lines corresponds to the mean. In such circumstances a batch normalized network barely reaches the performance of an unnormalized network. We also study a batch normalized network using its original learning rate (0.1) and (0.003) and consistently find that the gap between training and test accuracy is much larger for a batch normalized network using a lower learning rate, suggesting a regularizing effect of the higher learning rate.

noise quantity M as $M = \mathbb{E}\left[\left(\nabla f_i(x) - \nabla f(x)\right)^2\right]$. Using linear algebra, this lets us write the noise of the gradient estimate given by SGD as

$$\mathbb{E}\left[\|\alpha \nabla f(x) - \alpha \nabla_{SGD}(x)\|^2\right] = \frac{\alpha^2}{\|B\|}M \tag{2}$$

This expression suggests that the batch size and learning rate control the noise-level for SGD in the same way. This has indeed been observed in practice in the context of parallelizing neural networks [12] [43] and derived in other theoretical models [21]. It is widely believed that the noise in SGD has an important role in regularizing neural networks [49] [5]. Most pertinent to us is the work of [22], where it is empirically demonstrated that large batches lead sharp minima which generalize poorly. The intuition is that larger SGD noise from smaller batches would bias the network towards wider minima with better generalization. This suggests that higher learning rate is associated with more regularization since learning rates plays the same role as batch size according to (2). We thus posit that the higher learning rates that BN enables is responsible for increasing the SGD noise which increases regularization and accuracy, a conclusion in accordance with Figure 1.

3 Delving Deeper Into The Effect of Batch Normalization

3.1 Gradient and Activation Magnitudes

Having found that the benefits of BN are mediated chiefly by allowing larger learning rates, we now ask why BN enables these larger gradient steps. In our experiments, the learning rate has been limited by the network diverging for too large rates, which typically happens in the first few mini-batches. Hence, we focus on the gradients at initialization. When comparing the gradients between batch normalized and unnormalized networks one consistently finds that the gradients of comparable parameters in the latter are larger and distributed with heavier tails. Representative distributions for gradients within a convolutional kernel are illustrated in Figure 2. We see that the difference in gradient magnitude is about two order of magnitudes.

The dramatic impact of batch normalization on the gradients suggests that either the mean or variance of some channels stray far away from 0 and 1 respectively, otherwise the normalization would have little effect. This is indeed the case, and as it turns out the mean and variance of channels in an unnormalized network typically grows with the depth of the network at initialization, see Figure 3. By definition, these large means imply that some signal that is independent of the input image is being propagated forward in the network, and we will see that the data-independent network output has dramatic effects.
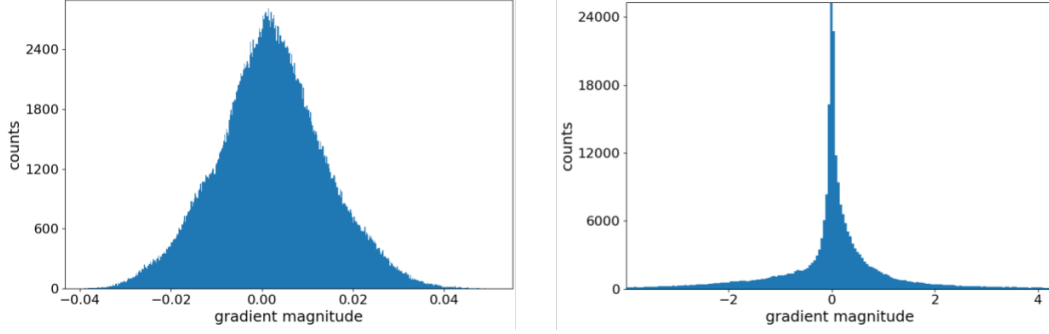
3

Figure 2: We here illustrate a histogram over the gradients at initialization for layer 55. To the left, we see a batch normalized network, and to the right, we see an unnormalized network. Note that we have to use different scales for the two plots because the gradients for the unnormalized network are almost two orders of magnitude larger than for the normalized on. For the unnormalized network, the gradients are also distributed with heavier tails.

In Figure 3 we also note that the network transforms normalized input into an output that reaches scales of up to 10^2 for largest output channels. It is natural to suspect that such a dramatic relationship between output and input results in large gradients. To test this intuition, we train a Resnet that uses batch normalization only at the top of the network, normalizing the output of the residual blocks but no intermediate activation. Such an architecture indeed scales the gradients similar to fully batch normalized network, allows for learning rates up to 0.03 and give a final accuracy of 90.1, see Appendix G. That this simple trick gives more than two-thirds of the improvement in accuracy that complete batch normalization gives suggests that normalizing the output the most important effect of batch normalization, and also highlights the problem unnormalized network output.
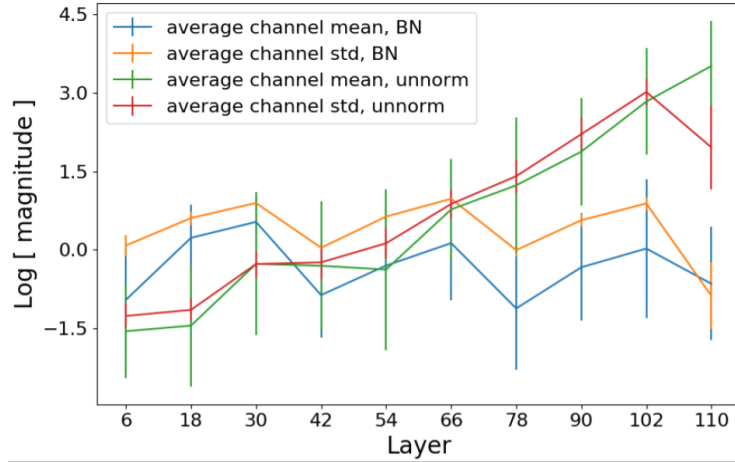

Figure 3: Above we illustrate how the average channel means and variances grow with the network depth at initialization. We additionally provide the standard deviations of the channel means/variances within a layer via the error bars. The means/variances that batch normalization measures, or would measure, are used. Since we are interested in the magnitude we use the absolute value for the means. Notice the logarithmic scale. It's clear that for a batch normalized network the mean and variances stays relatively constant throughout the network. For an unnormalized network, they seem to grow almost exponentially with depth.

3.2 Gradients at the output layer

In Figure 3 we have seen that the channel means increase with the network depth. For the final output layer corresponding to the classification, a large channel mean implies that the network is biased towards the corresponding class. Our previous experiments with batch normalization on the top suggest that this has an important effect upon the gradients. In Figure 4 we illustrate $\frac{\partial L_b}{\partial \text{OUTPUT}(b,j)}$,

where L_b is the loss for image b in our mini-batch, and activations j corresponds to class j at the final layer. Since the final layer is non-convolutional it just has two indices. In Figure 3 we see that for an unnormalized network, one specific class indeed seems favored independent of the input, let us call it j'. For all images b where the correct class does not correspond to j', we see $\frac{\partial L_b}{\partial \text{OUTPUT}(b,j')}$ is roughly constant and relatively large.

These activations are not parameters in our network, but they affect the gradients of our parameters. Let us consider the bias-vector B in the final layer, which has as many entries as there are classes. Its gradient for component j is calculated over the mini-batch as $\sum_b \frac{\partial L_b}{\partial \text{OUTPUT}(b,j)}$. Now, for the unnormalized network and class j' which the network is biased towards, since the summands of the gradient are essentially almost large and positive, the gradient with respect to $B_{j'}$ will be large. For the normalized network, however, we see in Figure 4 that these summands for any channel j generally don't have the same size and sign, and hence the gradient of the corresponding bias will be smaller. Since the gradients are similar between examples within the mini-batch, the total gradient, which is just the sum of gradients for images within the mini-batch, will be large. Since the gradients further down in the network are just linear functions of the gradients of activations at the final layer, this suggests that the gradients for all network parameters would be dominated by their contribution to the activations of the favored class j' at initialization. This is indeed the case, see Appendix L.
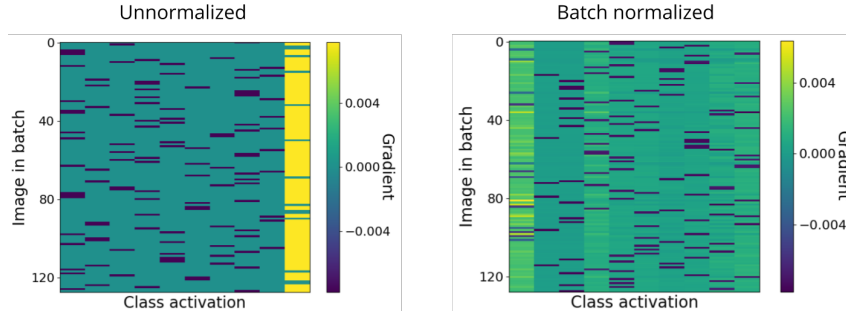

Figure 4: We here illustrate $\frac{\partial L_b}{\partial z_i}$ for activations z_i at the final layer, corresponding to unique classes, and image b within a mini-batch, at initialization. For an unnormalized network, these gradients are almost the same for essentially all examples in a mini-batch, suggesting that the gradients are input-independent. For a batch normalized network, the dependence upon the input is much larger.

3.3 Gradients of convolutional parameters

We now ask, can a similar reason explain why the gradients for convolutional weights are larger for unnormalized networks. Let us consider a convolutional weight $K(c_o, c_i, i, j)$, where the first two dimensions correspond to the outgoing/ingoing channels and the two latter to the spatial dimensions. For notational clarity we consider 3-by-3 convolutions and define $S = \{-1, 0, 1\} \times \{-1, 0, 1\}$ which indexes into K along spatial dimensions. Using definitions from section 1.1 we have

$$\text{OUTPUT}(b, c, x, y) = \sum_{c'} \sum_{i,j \in S} \text{INPUT}(b, c', x+i, y+j) K(c, c', i, j) \tag{3}$$

Now for some parameter $K(c_o, c_i, i, j)$ inside the convolutional weight K, its derivate with respect to the loss is given by the backprop equation [37] and (3) as

$$\frac{\partial L}{\partial K(c_o, c_i, i, j)} = \sum_{b,x,y} d^{bxy}_{c_o c_i ij}, \qquad \text{where} \qquad d^{bxy}_{c_o c_i ij} = \frac{\partial L}{\partial \, \text{OUTPUT}(b, c_o, x, y)} \text{INPUT}(b, c_i, x+i, y+j)$$
$$\tag{4}$$

We see that the gradient for $K(c_o, c_i, i, j)$ is the sum over the gradients of examples within the mini-batch, and over the spatial dimensions we slide the kernel K over. Inspired by Section 3.2, we now want to investigate if the summands (4) in all agree across examples within a mini-batch

and across spatial dimensions. These sums are probed at initialization in Table 4, where we see that for an unnormalized network the absolute value of (4) and the sum of the absolute values of the summands generally agree to within a factor 2 or less. For a batch normalized network, these expressions differ by a factor of 10^2, which explains the remarkable difference is gradient magnitude between normalized and unnormalized networks seen in Figure 2.

These results suggest that the summands in (4) for an unnormalized network have the same sign across both spatial dimensions and examples within a batch, and that they thus encode information that is neither input-dependent or dependent upon spatial dimensions. Note that the input to a convolutional layer is always preceded by a Relu layer, hence the input is always negative and the sign of the summands $d^{bxy}_{c_o c_i ij}$ are decided by the sign of $\frac{\partial L}{\partial\ \mathtt{OUTPUT}(b,c_o,x,y)}$. We probe these questions further in Appendix M, where we investigate individual parameters instead of averages over them. We argue that in an unnormalized network, the learning rate for the input-dependent components would be limited by the largest learning rate for which the much larger input-independent components makes the network diverge, resulting in little noise and poor regularization.

	$\sum_{bxy} \lvert d^{bxy}_{c_o c_i ij}\rvert$	$\sum_b \lvert \sum_{xy} d^{bxy}_{c_o c_i ij}\rvert$	$\sum_{xy} \lvert \sum_b d^{bxy}_{c_o c_i ij}\rvert$	$\lvert \sum_{bij} d^{bxy}_{c_o c_i ij}\rvert$
Layer 18, BN	7.44509e-05	4.56393e-06	1.30080e-05	2.95631e-07
Layer 54, BN	1.90210e-05	2.49563-06	3.31031e-06	1.68555e-07
Layer 90, BN	6.62426e-06	3.24293e-06	1.12625e-06	1.62689e-07
Layer 18	6.31433e-05	3.65324e-05	4.06954e-05	3.63081e-05
Layer 54	2.19251e-04	8.63927e-05	1.54601e-04	8.35946e-05
Layer 90	2.59182e-04	1.59253e-04	1.80002e-04	1.23079e-04

Table 1: We here study the gradients of a convolutional weight as described in (4) at initialization. The gradient of some parameter is the sum of gradients over the spatial dimensions where the parameter is used and images in a mini-batch. We here compare various ways of using the absolute value to estimate how such a sum behaves. The expressions in the column head give a value for every parameter inside a kernel, we report the average value among parameters in the convolutional kernels, times a constant factor (this factor being one over the product of spatial dimensions and batch size). For an unnormalized network we see that the absolute value of the sum is roughly equal to the sum of the absolute value of the summands, but for a batch normalized network, those two differ by roughly two orders of magnitude. This suggests that the gradients align over spatial dimensions and over examples within a mini-batch for an unnormalized network and that we have large data-independent components of the gradient. We also see that for the unnormalized networks the gradients become larger deeper into the network, this is natural as the gradient size scales with the activations, which are larger deeper into the network as per Figure 3. For an extended table, see Appendix O.

3.4 Batch normalization beyond gradient scales

We have seen that the gradients become smaller by roughly two orders of magnitude when batch normalization is used. A batch-normalized network can, however, tolerate learning rates roughly four orders of magnitude larger (see Appendix J) and thus the gradient scale cannot alone explain why batch normalization allows for larger learning rates. Possible learning rates for unnormalized networks are limited by divergence, with too large learning rate invariably leading to the loss exploding during the first few mini-batches. Let us define divergence as when the loss on a mini-batch becomes higher than 10^3, in our experiments the network has never been able to recover, in the sense of later converging to acceptable accuracy, after such a loss. With this definition, we can now precisely find the gradient update responsible for divergence. In Figure 5 we plot the loss landscape of an unnormalized neural network along such a gradient update, where the loss is calculated on the responsible batch. We see that the loss seemingly explodes along the gradient direction in a way that is not indicated at the origin. In Appendix N we zoom in on the loss landscape along the gradient direction and compare it to the same figure of a batch normalized network, where one can take much larger steps while decreasing the loss. It is interesting to see what happens with the network moments along such a 'diverging update', which we illustrate in Figure 6. There we see that the means and variances, especially in higher layers, seemingly explodes in the last parts of the path. This suggests that the magnitude of the activations grows exponentially with the depth of the network, and it's clear why extremely large means and variance at the very top layer would result in a large loss. The weight updates interacting in this fashion is a non-linear phenomenon that the gradients are 'unaware' of.

It is clear that batch normalization inhibits such exponential growth by normalizing the activations for every layer. This guarantees that even for extremely large gradient steps, we will 'land safely' in a region in parameter space where the network output is normalized. We posit that this is the second primary mechanism by which batch normalization allows for larger learning rates. To investigate this hypothesis we experiment with a version of batch-normalization that calculates the mean and variances every other batch and uses the latest statistics for other batches in Appendix J. Such a scheme does not guarantee 'safe landing' and indeed it only allows for learning rates up to 0.01. Under this hypothesis, the folklore observations that less deep networks allow for larger learning rates becomes natural. For less deep networks there aren't as many layers in which exponential growth can propagate. See Appendix J for experiments regarding possible learning rates versus network depth, where we indeed find that less deep networks can tolerate larger learning rates.
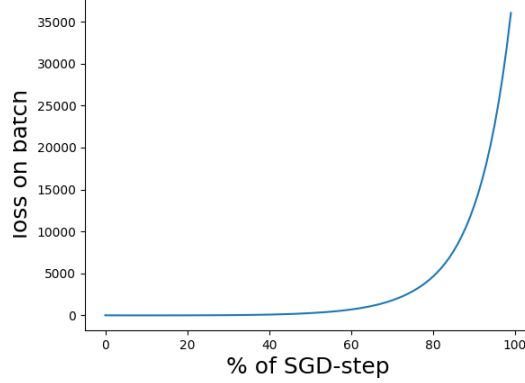

Figure 5: We here show the batch loss along the gradient direction for an unnormalized network, during the gradient update that makes the network diverge. The learning rate used is 0.002. We see that the loss explodes towards the end of the update path, far from the point where the gradient is calculated.
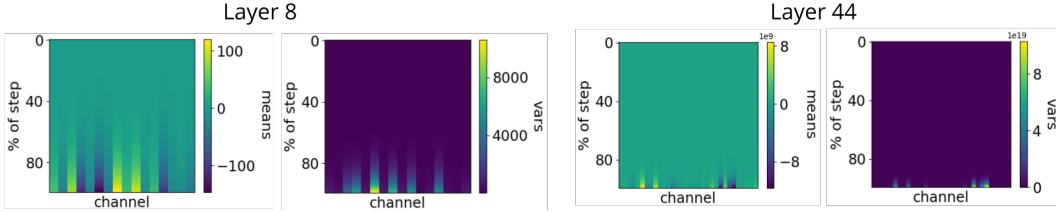

Figure 6: We here illustrate how the channel means and variances behave during the gradient update that makes an unnormalized network diverge. We see that the moments explode in the higher layers, but stay relatively constant in lower levels. This suggests that the magnitude of neuron activations grows exponentially with the depth of the network, and that these large activations cause the large loss.

3.5 Further observations

Table 4 suggests that for an unnormalized network the gradients are similar across spatial dimensions and images within a batch. It's unclear however how they vary with the input/output channels c_i, c_o. To study this we consider the matrix $\sum_{ij} |\sum_{bxy} d^{bxy}_{c_o c_i ij}|$ which is indexed by c_i, c_o at initialization, which intuitively measures the average gradient magnitude of kernel parameters between input channel c_i and output channel c_o. Representative results are illustrated in Figure 7, where we see clearly that some channels constantly are associated with larger gradients while others have extremely small gradients by comparison. Since some channels have large means, we expect in light of (4) that weights outgoing from such channels would have large gradients which would party explain the structure in in Figure 7. This is indeed the case, see Appendix I for details.

In Appendix H we additionally investigate the effect of the individual transformations of batch normalization in (1), where one sees that the affine transformation has a relatively small impact. We also note that it is possible to slightly increase the learning rate for an unnormalized network after some training with a smaller learning(see Appendix J), achievable learning rates remain far from

7

what a batch normalized network tolerates, suggesting that its issues persist during training. We defer further studies into the effect of batch normalization during training for future work.
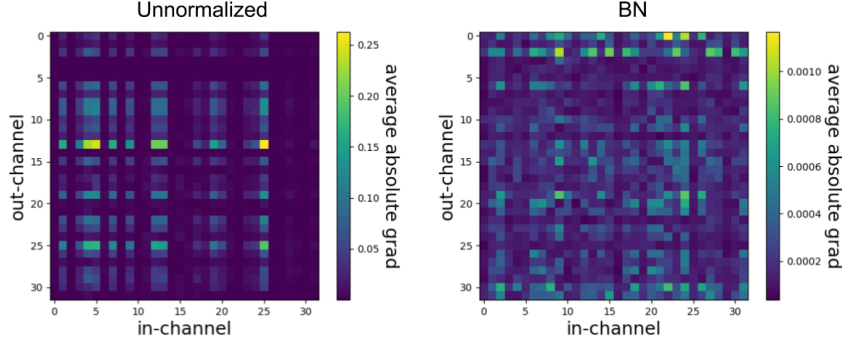

Figure 7: We here illustrate the average gradients for parameters between in and out channels for layer 45 at initialization. For an unnormalized network we see a low-rank structure, some in/out-channels have consistently large gradients while others have consistently small gradients. This relationship disappears with batch normalization. We see that the gradient updates for some parameters will be very small for an unnormalized network.

4   Random initialization revisited

We have seen remarkable scaling of gradients and activations of unnormalized neural networks at initialization in Figure 3 and 7, and argue that this must be a natural consequence of random initialization. This observation seems to be at odds with the trusted Xavier initialization scheme [10] which we use. Doesn't such initialization guarantee a network where information flows smoothly between layers? These initialization schemes are derived from the desiderata that the variance of channels should be constant when randomization is taken over random weights. We argue that this condition is much too weak, for example, a pathological initialization that sets weights to 0 or 100 with some probability could fulfill it. In [10] simplifying assumptions are made that essentially results in a linear neural network, and we will see how recent results in random matrix theory can give further insights.

Let us consider a simple toy model: a linear feed-forward neural network where $A_t \dots A_2 A_1 x = y$, for weight matrices $A_1, A_2 \dots A_n$. While such a model clearly abstracts away many important points they have proven to be valuable models for theoretical studies [10] [48] [13] [29]. CNNs can, of course, be flattened into fully-connected layers with shared weights. Now, if the matrices are initialized randomly, the network can simply be described by a product of random matrices. Such products have recently garnered attention in the field of random matrix theory, from which we have the following recent result.

Theorem 1 *Singular value distribution of products of independent Gaussian matrices [27]. Assume that $X = X_1 X_2 \dots X_M$, where X_i are independent $N \times N$ Gaussian matrices s.t. $\mathbb{E}[X_{i,jk}] = 0$ and $\mathbb{E}[X_{i,jk}^2] = \sigma_i^2/N$ for all matrices i and indices j, k. In the limit $N \to \infty$, the expected singular density $\rho_X(x)$ of X for $x \in \left(0, (M+1)^{M+1}/M^M\right)$ is given by*

$$\rho(x) = \frac{1}{\pi x} \frac{\sin((M+1)\varphi)}{\sin(M\varphi)} \sin \varphi, \qquad \text{where} \qquad x = \frac{\left(\sin((M+1)\varphi)\right)^{M+1}}{\sin \varphi (\sin(M\varphi))^M} \qquad (5)$$

These somewhat messy equations are illustrated in Appendix D. We notice in (5) that the distribution blows up as $x^{-M/(M+1)}$ near the origin and that it goes to zero at $(M+1)^{M+1}/M^M$. For large M this tends to $(M+1)e$ so we would expect the largest singular increase linearly with the depth of the network for infinitely large matrices. In Figure 8 we investigate the singular value distribution for practically sized matrices. By multiplying more matrices, which represents a deeper linear network, the singular values distribution becomes significantly more heavy-tailed. Intuitively this means that
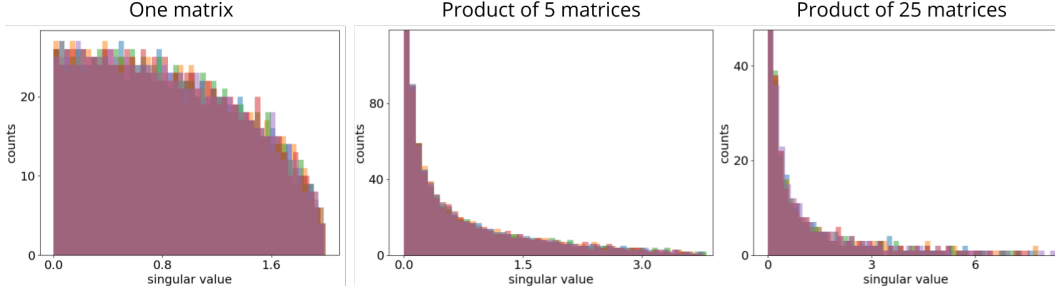
Figure 8: We here illustrate the distributions of singular values of random square matrices and product of independent such matrices. The matrices have dimension 1000 and entries independently drawn from a standard gaussian distribution. We overlap five different runs to show the variation. We see that the distribution of singular values becomes more heavy-tailed as more matrices are multiplied together. It is also clear that the maximum singular value increases with the number of matrix factors, which we further confirm in Appendix E.

the ratio between the largest and smallest singular value (the condition number) will increase with depth, which we verify in Appendix E.

In the literature for linear neural networks, one often considers the optimization problem $\min_{A_i,\ i=1,2\ \ldots\ t} \|A_t\ \ldots\ A_2 A_1 x - y\|^2$. This is formally similar to the problem of solving a linear system $\min_x \|Ax - y\|^2$, and indeed one can show that these problems are similar when one only optimizes over a single weight matrix A_i, see Appendix C. A central result is that the complexity of solving $\min_x \|Ax - y\|$ via gradient descent can be characterized by the condition number κ of A, the ratio between largest σ_{max} and smallest singular value σ_{min}. It is well known that increasing κ has the following effects on solving a linear system with gradient descent: **1)** convergence becomes slower, **2)** a smaller learning rate is needed, **3)** the ratio between gradients in different subspaces increases [3].

There are many parallels between these results from numerical optimization, and what is observed in practice in deep learning. Based upon theorem 1 and our numerical simulations, we expect the conditioning of a linear neural network at initialization for more shallow networks to be better which would allow a higher learning rate. And indeed, for Resnets with fewer layers one can use a much larger learning rate, see Appendix J. It thus seems like batch normalization alleviates issues associated with poor conditioning. For example, we have seen in Figure 7 that batch normalization ensures that all gradients are scaled similarly. Theorem 1 suggests that unnormalized networks become increasingly ill-conditioned with depth, which suggests that batch normalization becomes increasingly important for deep networks. Worse conditioning would lead to the optimal learning rate depending on the architecture. It has been observed that batch normalized network are more robust to learning rates, and we hypothesize that this is due the optimal learning rate depending less upon architectural parameters. Additionally, based upon Theorem 1 deeper into linear networks one would expect the channels to have larger variance due to the larger singular values, and as we have seen in Figure 3 this also happens for nonlinear neural networks.

The Xavier [10] and Kaming initialization schemes [14] amounts to a random matrix with iid entries that are all scaled proportionally to $n^{-1/2}$, the same exponent as in Theorem 1, with different constant factors. Theorem 1 suggests that such an initialization will yield ill-conditioned matrices, independent of these scale factors. Results Theorem 1 typically holds for a large family of distributions of the matrix entries, a phenomenon referred to as universality [45], which suggests that these problems cannot be circumvented by simply changing entry distribution. We believe that the shortcoming of Xavier-initalization for deeper networks explains why batch normalization is needed to make networks robust to initialization schemes.

5   Related Work

The original batch normalization authors posit that internal covariate explains its benefits [20], we do not claim that internal covariate shift does not exist, but we believe that the success of BN can be explained without it. We argue that the best reason to doubt that internal covariate shift is the

reason that batch normalization works comes from results in [31], where an initialization scheme that ensures that all layers are normalized is proposed. In this setting, internal covariate shift would not disappear, however, the authors show that such initialization can be used instead of BN with little performance loss. Their results are in line with our explanation – that a major benefit of normalization is to remove artifacts from fluctuations in the random initialization. Due to space limitations, we defer discussion regarding variants of batch normalization, random matrix theory, generalization as well as further related work to Appendix A.

6   Conclusions

We have investigated batch normalization and its benefits, showing how the latter are mainly mediated by larger learning rates, and argued how this increases the regularization of SGD which improves generalization. We have demonstrated how unnormalized networks have large and ill-behaved outputs, and how this results in gradients that are input independent which limits potential learning rates. Additionally, we have shown how the potential for large parameters updates to unnormalized networks that result in activations whose magnitude grows exponentially with depth further limits possible learning rates. Via recent results in random matrix theory, we have explained the ill-conditioned activations as natural consequences of the random initialization. Future work includes studying relatives of batch normalization such as [1] and putting our observations on a more solid theoretical footing.

References

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

[2] Mark F Bear, Barry W Connors, and Michael A Paradiso. *Neuroscience*, volume 2. Lippincott Williams & Wilkins, 2007.

[3] Dimitri P Bertsekas and Athena Scientific. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

[4] Pratik Chaudhari, Anna Choromanska, Stefano Soatto, and Yann LeCun. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

[5] Pratik Chaudhari and Stefano Soatto. Stochastic gradient descent performs variational inference, converges to limit cycles for deep networks. *arXiv preprint arXiv:1710.11029*, 2017.

[6] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *Advances in Neural Information Processing Systems*, pages 4470–4478, 2017.

[7] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015.

[8] Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[9] Alan Edelman. Eigenvalues and condition numbers of random matrices. *SIAM Journal on Matrix Analysis and Applications*, 9(4):543–560, 1988.

[10] 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.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[12] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

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

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[15] 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*, pages 770–778, 2016.

[16] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[17] Elad Hoffer, Itay Hubara, and Daniel Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1729–1739, 2017.

[18] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[19] Sergey Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1942–1950, 2017.

[20] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[21] Stanislaw Jastrzkebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

[22] 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.

[23] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 2017.

[24] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009.

[25] César Laurent, Gabriel Pereyra, Philémon Brakel, Ying Zhang, and Yoshua Bengio. Batch normalized recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2657–2661. IEEE, 2016.

[26] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[27] Dang-Zheng Liu, Dong Wang, Lun Zhang, et al. Bulk and soft-edge universality for singular values of products of ginibre random matrices. In *Annales de l'Institut Henri Poincaré, Probabilités et Statistiques*, volume 52, pages 1734–1762. Institut Henri Poincaré, 2016.

[28] Cosme Louart, Zhenyu Liao, and Romain Couillet. A random matrix approach to neural networks. *arXiv preprint arXiv:1702.05419*, 2017.

[29] Haihao Lu and Kenji Kawaguchi. Depth creates no bad local minima. *arXiv preprint arXiv:1702.08580*, 2017.

[30] Dominic Masters and Carlo Luschi. Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*, 2018.

[31] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[32] Carles Roger Riera Molina and Oriol Pujol Vila. Solving internal covariate shift in deep learning with linked neurons. *arXiv preprint arXiv:1712.02609*, 2017.

[33] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5949–5958, 2017.

[34] Jeffrey Pennington and Yasaman Bahri. Geometry of neural network loss surfaces via random matrix theory. In *International Conference on Machine Learning*, pages 2798–2806, 2017.

[35] Jeffrey Pennington and Pratik Worah. Nonlinear random matrix theory for deep learning.

[36] James Renegar. Incorporating condition measures into the complexity theory of linear programming. *SIAM Journal on Optimization*, 5(3):506–524, 1995.

[37] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[38] Sirpa Saarinen, Randall Bramley, and George Cybenko. Ill-conditioning in neural network training problems. *SIAM Journal on Scientific Computing*, 14(3):693–714, 1993.

[39] 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*, pages 901–909, 2016.

[40] Fabian Schilling. The effect of batch normalization on deep convolutional neural networks, 2016.

[41] Hidetoshi Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of statistical planning and inference*, 90(2):227–244, 2000.

[42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[43] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[44] Terence Tao. Singularity and determinant of random matrices. `https://terrytao.files.wordpress.com/2008/03/determinant.pdf`.

[45] Terence Tao. *Topics in random matrix theory*, volume 132. American Mathematical Soc., 2012.

[46] Patrick Van Der Smagt and Gerd Hirzinger. Solving the ill-conditioning in neural network learning. In *Neural networks: tricks of the trade*, pages 193–206. Springer, 1998.

[47] Shuang Wu, Guoqi Li, Lei Deng, Liu Liu, Yuan Xie, and Luping Shi. L1-norm batch normalization for efficient training of deep neural networks. *arXiv preprint arXiv:1802.09769*, 2018.

[48] Chulhee Yun, Suvrit Sra, and Ali Jadbabaie. Global optimality conditions for deep neural networks. *arXiv preprint arXiv:1707.02444*, 2017.

[49] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[50] Tullio Zolezzi. Condition number theorems in optimization. *SIAM Journal on Optimization*, 14(2):507–516, 2003.

Appendices

A   Extended related work

A.1   On internal covariate shift and other explanations

The benefits of normalizing the inputs when training neural networks was known already in the 90s [26], here the rationale was that nonzero means biased the gradient updates. In the original BN paper [20], *internal covariate shift*, a concept dating back to [41], is proposed as the problem that BN alleviates. Internal covariate shift is explained as the tendency for features that some intermediate layer trains against to shift during training, although no precise formal definition is given. This explanationg seems to have been deemed satisfying in some papers [47] [32], while other papers credits the succes of normalization to improved conditioning [39] or stability of concurrent updates [11]. The only work on empirically on batch normalization known to us is the master thesis [40], where most benefits of batch normalization are verified and where the interaction with batch normalization and activation functions are studied. This paper instead takes an first-principles approach to understanding batch normalization, we do not claim that internal covariate shift does not exist, but we believe that the success of BN can be explained without it. Indeed, as internal covariate shift is only vaguely defined, falsifying becomes hard.

A.2   Random matrix theory and conditioning

The classical definition of conditioning for linear systems is well known [3], but for general optimization problems there are other metric for conditioning [50] [36]. In the 90s there was some work on the conditioning of training sigmoid neural networks, see for example [46] [38], where the main focus seems to have been the conditioning of the Hessian matrix and ways to improve it. For modern neural networks conditioning is sometimes mentioned in passing, see for example [39], but there is little work focusing specifically defining and emprically studying conditioning. Random matrix theory has previously been used for broadly studying neural networks, see for example [35] [34] [28] [7]. Unlike this work, we consider products of random matrices, for which theory has recently been developed. The conditioning of random matrices in the context of solving a linear system has been studied by [9], where the conditioning and running time of conjugate gradient descent is studied as functions of the matrix size.

A.3   Alternatives to Batch Normalization

Many alternatives to batch normalization have been proposed [39] [23] [19], however, the original formulation remains the most popular and is generally competitive with alternatives for image classification. For recurrent neural networks it is problematic to apply batch normalization [25] and to rectify this normalization schemes for RNNs have been proposed [8] [1]. In the distributed setting where a single batch is split among workers communicating batch statistics comes with communication overhead, to alleviate this [17] has proposed to let workers calculate their statistics independently, the paper additionally derives similar results as us for the relationship between batch size and learning rates. Some of these papers above have mentioned that normalization techniques improve conditioning [39], while others have mentioned the stability of concurrent updates [11]. However, these issues are typically mentioned in passing and mathematical rationale are rarely given.

A.4   Generalization in deep learning

Our work corroborates the emerging consensus regarding the approximate equivalence between batch size and learning rates [43] [12] [21] [30], the importance of the regularizing effects of SGD and it's relationship to generalization [22] [49] [5]. Other relevant work regarding generalization within deep learning are [16] [4] [33].

A.5   Relationships to neuroscience

Interestingly, normalization processes also occur in neuroscience. After becoming activated, the neuron enters a **relative refractory period** during which greater stimulus is required to initiate a

second activation [2]. This limits the number of times a neuron can become activated in some time span, and we hypothesize that this improves the conditioning of the signaling similarly to batch normalization.

B   Details on experimental setup

Our setup follows that of the original RESNET paper [15] – we consider image classification on CIFAR10 [24]. We use the original RESNET architecture for CIFAR10 with depth 110 as described in [15]. As per [15] we train our model with the cross-entropy using SGD with momentum (0.9) and weight decay (5e-4), using a batch size 128. We use standard image preprocessing of channel-wise mean and variance normalization and standard data augmentation of random horizontal flipping and random 32-by-32 cropping with 4-pixel zero-padding. For initialization, we use the Xavier-scheme [10].

When batch normalization is removed, the learning rates and the number of iterations need to be changed for satisfying performance. We standardize the learning rate and number of epochs used in our experiments as follows. We consider the initial learning rates from the set $\{0.1, 0.003, 0.001, 0.0003, 0.0001, 0.00003\}$ and report the best result, as measured by test accuracy at termination, for every architecture. Initially, all models are trained for 300 epochs and as in [15] we divide the learning rate by 10 after epoch 50% and 75%, at which point learning has typically plateued. If learning doesn't plateu with 300 epochs for some architecture, we double the number of epochs until it does.

Our code is implemented in pytorch and has been run on AWS p3.2xlarge instances. The source code for all experiments is to be made publically available at a later date. We chose the CIFAR10 dataset for its low computational overhead, and since its size allows us to manually analyze all components of the architecture and dataset. RESNET, on the other hand, is a widely familiar architecture that achieves results close to the state of the art, additionally, it is the basis for later architectural development [18] [6].

C   Relationship between linear neural networks and solving linear systems

We consider the two problems of optimizing a linear neural network

$$\min_{A_i,\ i=1,2\ ...\ t} \|A_t\ ...\ A_2 A_1 x - y\|^2 \tag{6}$$

and solving a linear system via optimizing

$$\min_x \|Ax - y\|^2 \tag{7}$$

Proposition 1 *For any matrix A_i, problem* (6) *can be transformed into one or more independent problems of type* (7) *assuming the constituent matrices are all full rank.*

Proof sketch. To show the similarity between (7) and (6) we consider the problem we are trying to solve when just changing one of the matrices A_i, say A_1. If we let $A_t...A_2 = \bar{A}$ we have

$$\min_{A_1} \|\bar{A} A_1 x - y\|^2$$

Now, we will write x in terms of its SVD decomposition. We will further assume that the number of samples are larger are larger than the number of input features so that $m > n$, which typically the case for training modern neural networks. We also assume that x is full rank, this is a standard assumption [29], and given noisy input the probability of getting singular rank matrices is zero [44]. This means that we have $x = U\begin{pmatrix}\Sigma\ 0\end{pmatrix}V^*$. Here U and V are orthonormal matrices, Σ is a diagonal matrix and 0 is a rectangular matrix of all zeros. Now, since the square norm is invariant under orthonormal matrices we have

15

$$\|\bar{A}A_1 x - y\|^2 = \|\bar{A}A_1 U \begin{pmatrix} \Sigma & 0 \end{pmatrix} V^* - y\|^2$$

$$= \|\bar{A}A_1 U \begin{pmatrix} \Sigma & 0 \end{pmatrix} - yV\|^2 = \|\bar{A}A_1 U\Sigma - (yV)_{1:n}\|^2 + C \qquad (8)$$

Here we note that because the last columns of $\begin{pmatrix} \Sigma & 0 \end{pmatrix}$ is zero, the corresponding columns in $\bar{A}A_1 U \begin{pmatrix} \Sigma & 0 \end{pmatrix}$ will be zero. These columns will then correspond to some term in the loss function that does not depend on A_1. In the RHS of (8) we have simply separated this term, where $(yV^T)_{1:n}$ is the matrix consisting of just the n first columns in yV^T. Now if we let $\bar{y} = (yV^T)_{1:n}$ and make a change of variables such that $A_1 U\Sigma = \bar{A}_1$ we now have the optimization problem

$$\min_{\bar{A}_1} \|\bar{A}\bar{A}_1 - \bar{y}\|^2$$

Now, if we consider some column of $\bar{A}_1$ and a corresponding column of $\bar{y}$, we have a problem of the typ (7). We note that the problem of finding all columns of $\bar{A}_1$ decouple in this way.

D   Distribution of the singular values

The distribution for singular values of products of matrices have been found the limit of large matrices, we will follow the exposition of [27]. For the product of M matrices random matrices, the density $\rho(x)$ is most easily expressed in terms of φ, which is related to x as follows

$$x = \frac{\big(\sin((M+1)\varphi)\big)^{M+1}}{\sin\varphi(\sin(M\varphi))^M}, \qquad 0 < \varphi < \frac{\pi}{M+1} \qquad (9)$$

We plot this relationship in Figure 9. Noting that this parametrization is strictly decreasing in φ it clearly gives a one-to-one mapping between x and φ. Given this parametrization, the density is given as follows

$$\rho(\varphi) = \frac{1}{\pi x} \frac{\sin((M+1)\varphi)}{\sin(M\varphi)} \sin\varphi \qquad (10)$$

The expression is complicated, however we can note that it blows up as $x \to 0$. In Figure 9 show the relationship between x and φ defined by (9). In Figure 10 we plot the expression (10) for some values of M.

E   Condition vs Depth

In Figure 11 we show numerical simulations regarding how the condition number κ, the ratio between the largest and smallest singular value, of a product of random matrices increase with the number of matrix factors. In Figure 12 we show of the maximum singular value increases with the factors.

F   Activation Histograms

In Figure 13 we plot a histogram of the activations for some representative channels for an unnormalized network at initialization. Figure 14 shows the same plots, but for a batch normalized network.

G   BN on top

In Figure 15 and 16 we show the learning curves for a network that uses batch normalization only at the top of the network, right after the output of the residual blocks and right before the average pooling leading into the fully connected layer. With such an architecture, one gets only somewhat weaker performance than a completely batch normalized network, using a learning rate of 0.03 and 330 epochs.
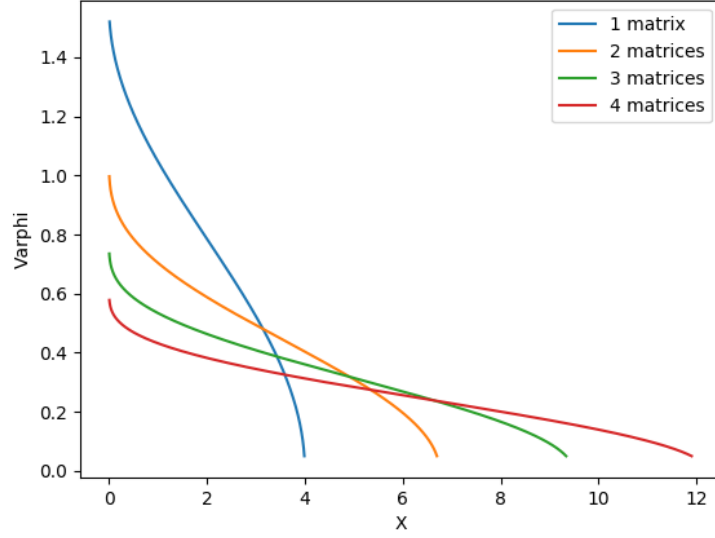
Figure 9: We here plot the relationship between x and φ according to (9) for different number of matrices multiplied together. We note that the relationship is injective.
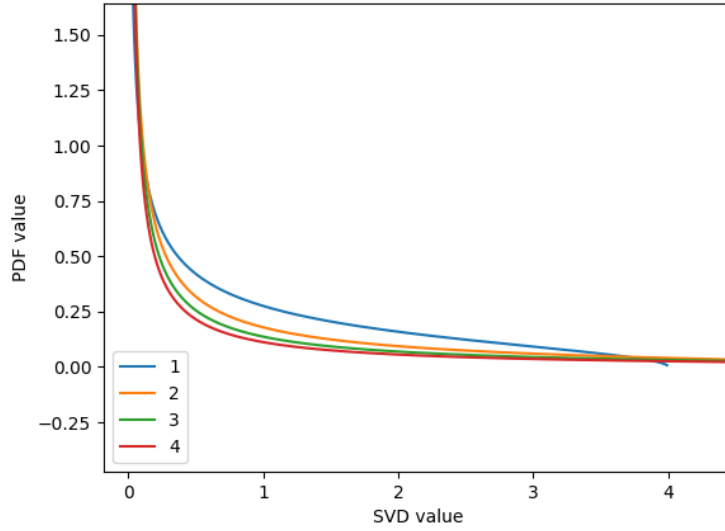

Figure 10: We here plot the distribution of singular values given in originally given in [27].

H   The parts of batch normalization

Having seen the dramatic benefits of batch normalization, we now turn our attention to investigate the impact of the normalization and the affine transformation in (1) individually. We investigate the impact of the affine transformation by simply removing the parameters γ, β and the transformations they encode from the batch normalization algorithm, and then training the network. Results are illustrated to the left in Figure 17. The whole affine transformation improves the final accuracy only by roughly 0.6%, and while far from inconsequential the affine transformation introduces additional parameters which typically improves accuracy. The effect of the normalization in (1) is similarly investigated by removing those components, results in the absence of any affine transformation are shown to the right in Figure 17. When normalization is turned off, smaller learning rates and longer training is necessary however we consistently find that the highest learning rates that don't diverge give the best results, see Table 2. We see that the normalization has much more dramatic effects on
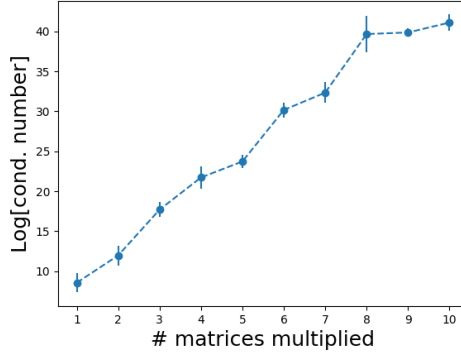
Figure 11: Above we illustrate the condition number of products of independent random square matrices as a function of the number of product matrices, the means and standard deviations over five runs are indicated with error bars. All matrices have size 1000 and entries independent gaussian entries, as we see, the condition number increases dramatically as the number of matrix products grows. Notice the logarithmic scale.
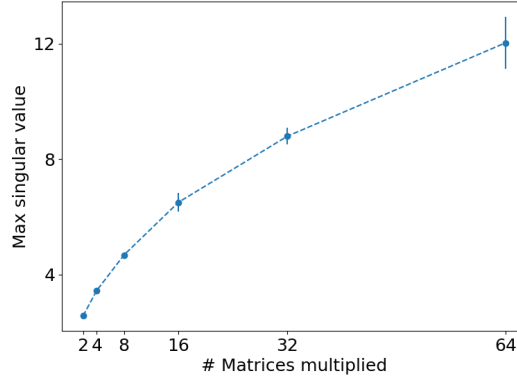

Figure 12: Above we illustrate the maximum singular value number of products of independent random square matrices as a function of the number of product matrices, the means and standard deviations over five runs are indicated with error bars. All matrices have size 1000 and entries independent gaussian entries, as we see, the maximum singular value increases as the number of matrix products grows, although not at the linear rate we would expect from infinitely large matrices.

accuracy and enables larger learning rate and faster convergence, but that both the mean and variance normalization are important.

Architecture	Initial learning rate	Epochs
$\gamma, \beta, \mu, \sigma^2$	0.1	165
γ, μ, σ^2	0.1	165
β, μ, σ^2	0.1	165
μ, σ^2	0.1	165
σ^2	0.01	330
μ	0.003	330
None	0.0001	2640

Table 2: The optimal learning parameters when different components of batch normalization are used. The architecture represents the components that are used in (1)
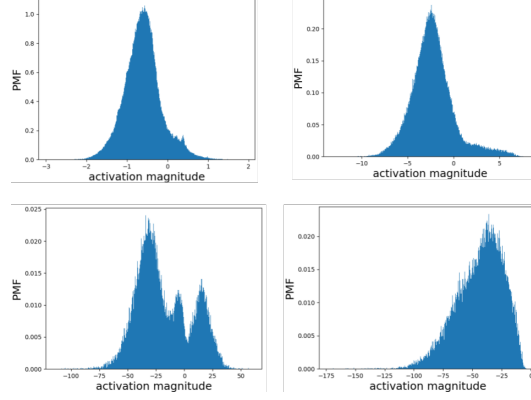
Figure 13: Above we illustrate some representative histograms over the activations for channels in different layers for an unnormalized network, sampled over spatial dimensions. Starting in the top left corner and going clockwise, the layers are 9, 45, 81 and 83. We see that for the deeper layers, the distributions increasingly have nonzero means and becomes more multimodal.
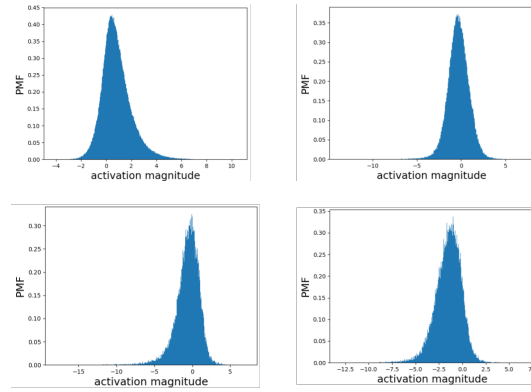

Figure 14: Above we illustrate some representative histograms over the activations for channels in different layers for an unnormalized network, sampled over spatial dimensions. Starting in the top left corner and going clockwise, the layers are 9, 45, 81 and 83. We see that the distributions remain unimodal with means close to zero, even deep into the network.

I   Explaining Figure 7

I.1   The effect of non-zero means

In Figure 3 we note that the means of different channels are nonzero in general. In histograms of the activations for an unnormalized network, one sees that some channels are essentially always on or always off, as their activations before Relu almost always have the same sign, see Appendix F. Let us consider how this affects the gradients. If we calculate the loss L on one example and take neurons i, j with activations z_i, z_j to be connected between adjacent layers by weight w_{ij} withing a feed-forward network, the gradients are given by backpropagation equation [37] as

$$\frac{\partial L}{\partial w_{ij}} = z_i \frac{\partial L}{\partial z_j} \tag{11}$$

We see in (12) that the derivate of w_{ij} scales linearly with the activation z_i. Thus, if some channel has a large mean we expect the gradient of weights outgoing from these channels to be large. To investigate this we plot the average gradient calculated over the dataset versus the mean channel activations for all kernel parameters. The results for layer 35 are illustrated in Figure 18 and as we can see the channel means have a large impact on the gradients. We also see that batch normalization
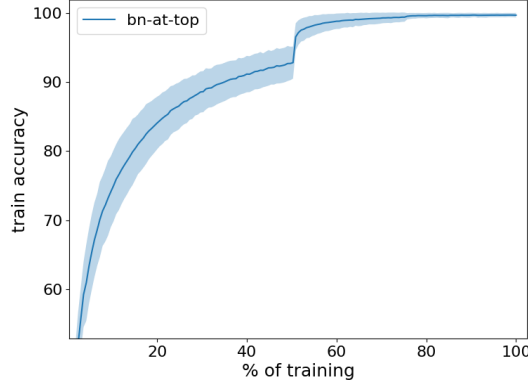
19

Figure 15: We here illustrate the training accuracy for an architecture where batch normalization is used only at the network top.
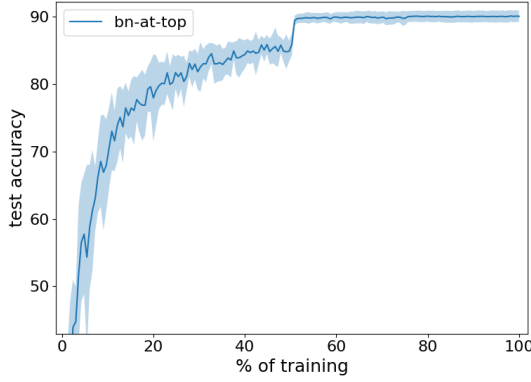

Figure 16: We here illustrate the test accuracy for an architecture where batch normalization is used only at the network top.

essentially removes this dependence, so that the gradients of all channels have essentially the same scale.

I.2 Channel gradients

Let us consider the backpropagation equation [37] for a feed-forward network. We calculate the loss L on one example and take neurons i, j with activations z_i, z_j to be connected between adjacent layers by weight w_{ij}.

$$\frac{\partial L}{\partial w_{ij}} = z_i \frac{\partial L}{\partial z_j} \tag{12}$$

In the linear model, we can view the neuron j is the input to the linear neural network ahead, which is simply a matrix. If a SVD-subspace that j is the input to has a large singular value it has a large impact on the output, and we expect the gradient $\frac{\partial L}{\partial z_j}$ to be large, which in turn makes $\frac{\partial L}{\partial w_{ij}}$ large. Within a single layer, the gradients will be scaled differently according to which subspaces the belong to, and with higher condition number κ the difference between these scales would increase. We investigate whether this intuition holds in our non-linear convolutional networks. In Figure 19 we plot gradients of the convolutional channels which are equivalent to the gradients of the bias terms. We see that the magnitude of the gradients differs a lot in earlier layers while being similarly scaled in higher layers. This is expected since the depth of the network "ahead" of lower layers are deeper for the early layers, which would cause them to have worse conditioning according to previous observations in RMT.
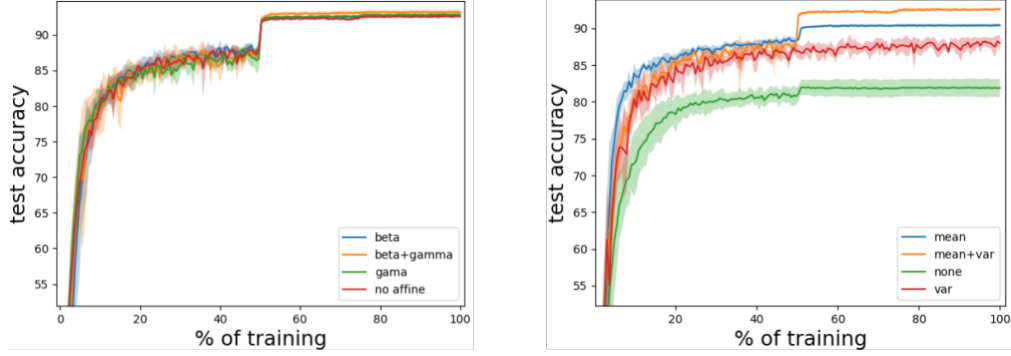
Figure 17: Illustrated above is the training curve for a 110-layer Resnet where parts of the batch norm algorithm are disabled. All experiments are repeated five times, with the shaded region corresponding to standard deviation and the thick lines the mean. To the left we illustrate the effect of the affine transformation parametrizes by γ and β in (1), when mean and variance normalization is used. To the right, we illustrate the effect of the mean and variance normalization in absence of the affine transformation. The mean and standard deviation over five runs are indicated by thick and opaque lines respectively.
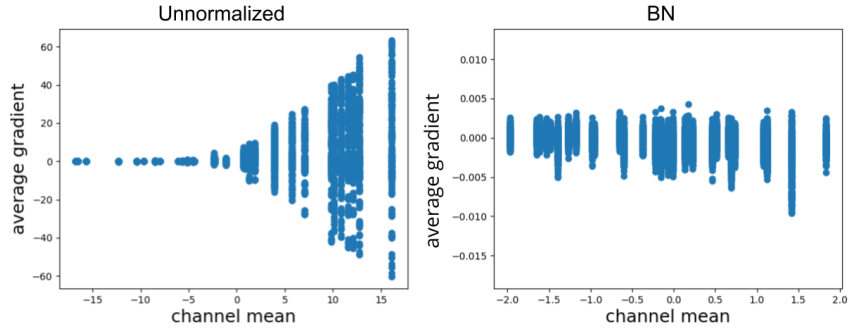

Figure 18: Here we illustrate the relationship between channel mean and gradients. Each point represents a single weight in a convolutional kernel, the x-axis is the mean activation before Relu for the in-channel of that weight while the y-axis is the average gradient of that weight at initialization. For an unnormalized network, large negative means leads to some channels being 'dead' after Relu such that their gradients are extremely small, while channels with large positive means have larger gradients. When using batch normalization this relationship disappears and all channels have equally large gradients on average.

J Learning rates for other architectures

We here give the highest initial learning rates achievable for some miscellaneous architectures, considering learning rates in the set $\{0.1, 0.003, 0.001, 0.0003, 0.0001, 0.00003\}$. We consider the original Resnet architecture with 32 and 56 layers respectively, and also an unnormalized network where we first train for one epoch with learning rate 0.0001. We refer to this as a warm start. This strategy allows us to study the ability of the network to rectify artifacts from the initialization. We also consider the scheme where the mean and variances are computed every other batch, and for other other batches the latest moments are used. This strategy is referred to as Alternating normalization. Achievable learning rates are rates for which the networks don't diverge or remain at the accuracy of random guessing. The results are illustrated in Table 3. We have additionally investigated the maximal learning rate that a batch normalized network can tolerate without diverging, and found the network to not diverge for 3.0 but diverge for 10.0.

K Dominating

As we will see this is quite common.

With some channels having large means relative to its peers, one suspects that these channels have the largest effect upon the neurons one layer above. This is indeed the case, which can be experimentally
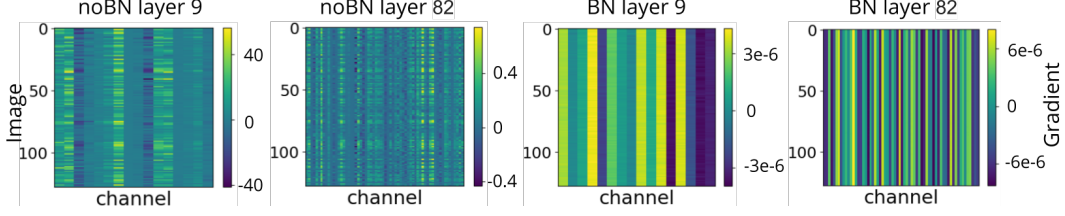
21

Figure 19: We here plot the gradients of the convolutional channels for the unnormalize and batch normalized network. We see that the gradients have larger scales for early layers in the unnormalized network, but not in later layers. With batch normalization, the gradients are similarly scaled across all layers.

Architecture	Initial learning rate
32 layer Resnet	0.1
56 layer Resnet	0.01
Unnormalized, warm start	0.01
Alternating normalization	0.01

Table 3: We here give the highest achievable learning rates for which some miscellaneous architectures. We see that less deep Resnets enables larger learning rates, and that warm-starting unnormalized nets allow for higher learning rates.

verified by finding the average magnitude of the output of different channels, due to space limitations these experiments are given in the appendix. If effectively only one channel matters the network does not use all its parameters and loses representation power.

L Class-wise experiments

We here investigate how the different channels at the top layer, corresponding to the different unique classes we wish to classify, influence the gradients. In Figure 3 we have seen that the channel means increase with network depth, and in Figure 4 we have seen that this causes the network to always guess one specific class c at initialization. This, in turn, biases the gradients w.r.t. the activations at the top layer corresponding to class c to be larger. One would suspect that the gradients of all network parameters are primarily computed by their influence over the activations corresponding to class c. To investigate this we simply do a class-wise mask of the gradients that are being propagated from the top layer. We set $\frac{\partial L}{\partial z_{ij}}$, where z_{ij} is the activation for example i within a mini-batch and class j, to zero for all classes j except for one class j' and then backpropagate this information downwards in the network. This allows us to investigate the gradients of all network parameters w.r.t individual classes. In Figure 20 we show that indeed one class is responsible for the majority of the gradients for a representative convolutional kernel. This channel is the one class c that the network almost always guesses. We further investigate this in Figure 21 where we show that the dominant class c gives gradients that are at least one order of magnitude larger than other classes.

M Kernel gradients experiments

According to (4) the gradients for one parameter in a convolutional kernel is given by $\sum_{b,x,y} d^{bxy}_{c_o c_i ij}$. In Table 4 we have seen that we have for an unnormalized network $|\sum_{b,x,y} d^{bxy}_{c_o c_i ij}| \approx \sum_{b,x,y} |d^{bxy}_{c_o c_i ij}|$ while these two differ by about two orders of magnitude at initialization. In Figure 22 we show how the terms $d^{bxy}_{c_o c_i ij}$ are distributed by a typical kernel parameter in an unnormalized network for which $|\sum_{b,x,y} d^{bxy}_{c_o c_i ij}| \approx \sum_{b,x,y} |d^{bxy}_{c_o c_i ij}|$ at initialization. Table 4 also just give summary statistics, to further investigate the relationship between $|\sum_{b,x,y} d^{bxy}_{c_o c_i ij}|$ and $\sum_{b,x,y} |d^{bxy}_{c_o c_i ij}|$ for individual parameters we plot a scatterplot of these two values for all parameters within a convolutional kernel. A typical result for both unnormalized and batch-normalized networks are shown in Figure 23.
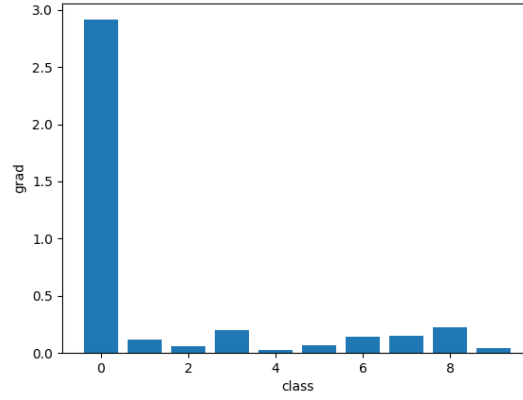
Figure 20: We here show how the absolute value of gradients for all parameters in the convolutional kernel in layer 52are computed across the different classes at the top layer. It is clear that one specific class has far larger impact upon the gradients. Similar results are found for other layers.
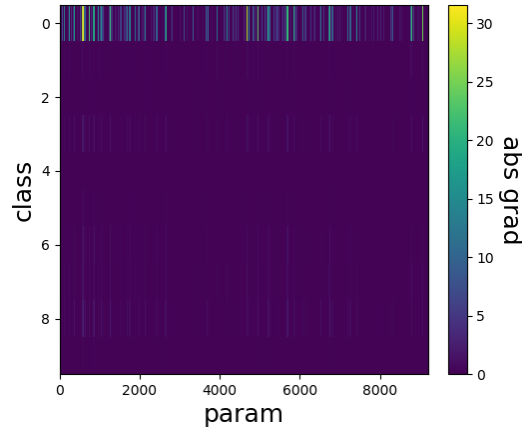

Figure 21: We here show the average absolute value of gradients for all parameters in the convolutional kernel in layer 52 across the different classes at the top layer. We see that one class has about one order of magnitude larger gradients than the other classes. Similar results are found for other layers.
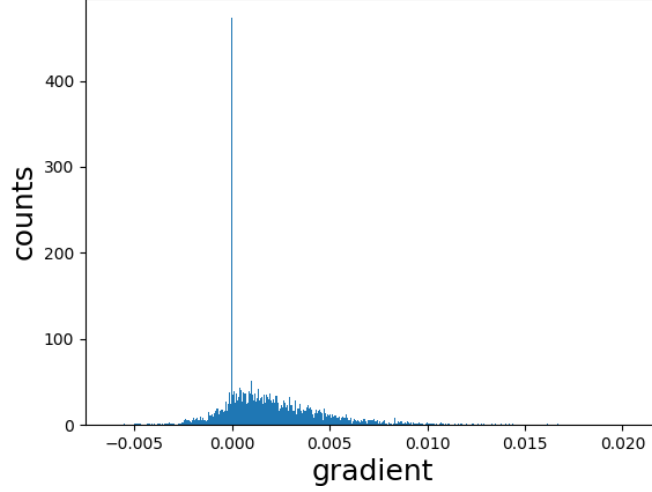
N   Loss on batch

O   Activation Histograms

Figure 22: We here show the distribution of $d^{bxy}_{c_o c_i ij}$, defined as per (4), for a single parameter in layer 90 for which $|\sum_{b,x,y} d^{bxy}_{c_o c_i ij}| \approx \sum_{b,x,y} |d^{bxy}_{c_o c_i ij}|$. We see that not all gradients are identical, but that the distribution are skewed far from 0. For parameters with smaller gradients these distributions are typically centered around 0.
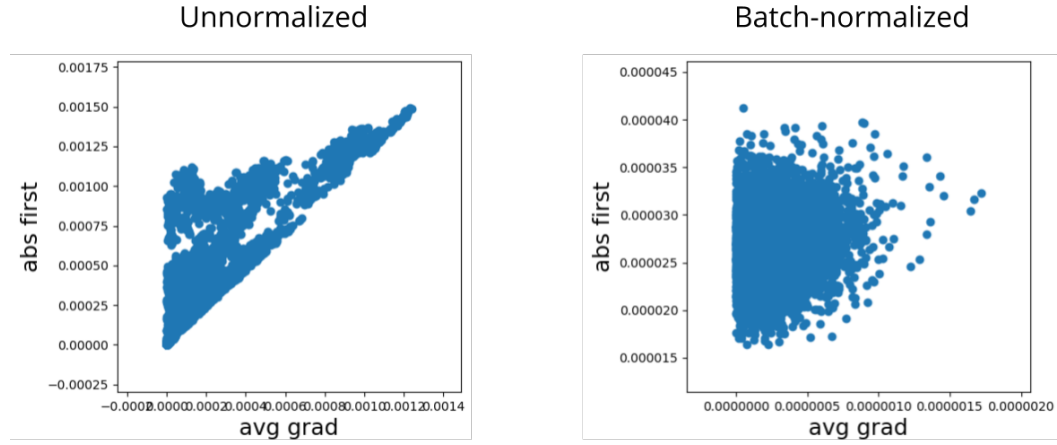

Figure 23: We here use the two values $|\sum_{b,x,y} d^{bxy}_{c_o c_i ij}|$ and $\sum_{b,x,y} |d^{bxy}_{c_o c_i ij}|$ for individual parameters in layer 54 as coordinates in a scatterplot. For an unnormalized networks we see an almost straight line that many parameters fall on, this represents parameters for which the gradients of the outgoing activations are similar across spatial dimensions and examples within a batch. For a normalized network this relationship disappears, and the gradients becomes much smaller.
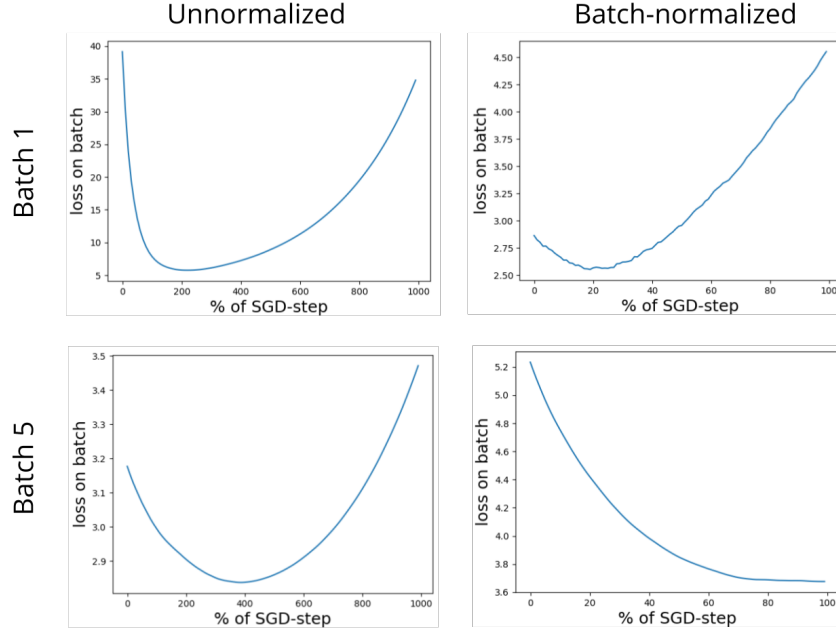
Figure 24: We here show the loss of a minibatch, along the direction defined by gradient computed at that batch, for the first and the fifth minibatch of the complete training. The landscape is shown for both unnormalized and batch-normalized networks using their original learning rates 0.0001 and 0.1 respectively. Clearly one can take larger steps that decrease the loss for the normalized network.

	$\sum_{bxy} |d^{bxy}_{c_o c_i ij}|$	$\sum_{b} |\sum_{xy} d^{bxy}_{c_o c_i ij}|$	$\sum_{xy} |\sum_{b} d^{bxy}_{c_o c_i ij}|$	$|\sum_{bij} d^{bxy}_{c_o c_i ij}|$
Layer 18, BN	7.44509e-05	4.56393e-06	1.30080e-05	2.95631e-07
Layer 19, BN	5.20086e-05	3.42920-06	9.15506-06	2.62803e-07
Layer 54, BN	1.90210e-05	2.49563-06	3.31031e-06	1.68555e-07
Layer 55, BN	1.62719e-05	2.16015e-06	2.82755e-06	1.70799e-07
Layer 90, BN	6.62426e-06	3.24293e-06	1.12625e-06	1.62689e-07
Layer 91, BN	5.44784e-06	3.07908e-06	9.09220e-07	1.98679e-07
Layer 18	6.31433e-05	3.65324e-05	4.06954e-05	3.63081e-05
Layer 19	5.34921e-05	1.99664e-05	2.90620e-05	1.95912e-05
Layer 54	2.19251e-04	8.63927e-05	1.54601e-04	8.35946e-05
Layer 55	6.82446e-05	2.08266e-05	4.50409e-05	1.95185e-05
Layer 90	2.59182e-04	1.59253e-04	1.80002e-04	1.23079e-04
Layer 91	1.37431e-04	8.61152e-05	9.31557e-05	6.46625e-05

Table 4: We here show the extended version of Table 4. In the Resnet architecture two consecutive layers play different roles, and thus we give values for neighboring layers.