Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs

Jonathan Frankle [1] **David J. Schwab** [2 3] **Ari S. Morcos** [3]

Abstract

Batch normalization (BatchNorm) has become an indispensable tool for training deep neural networks, yet it is still poorly understood. Although previous work has typically focused on its normalization component, BatchNorm also adds two per-feature trainable parameters: a coefficient and a bias. However, the role and expressive power of these parameters remains unclear. To study this question, we investigate the performance achieved when training only these parameters and freezing all others at their random initializations. We find that doing so leads to surprisingly high performance. For example, a sufficiently deep ResNet reaches 83% accuracy on CIFAR-10 in this configuration. Interestingly, BatchNorm achieves this performance in part by naturally learning to disable around a third of the random features without any changes to the training objective. Not only do these results highlight the under-appreciated role of the affine parameters in BatchNorm, but—in a broader sense—they characterize the expressive power of neural networks constructed simply by shifting and rescaling random features.

1. Introduction

Batch normalization (BatchNorm) is nearly ubiquitous in deep convolutional neural networks (CNNs) for computer vision (Ioffe & Szegedy, 2015). Computing BatchNorm proceeds in two steps during training (Algorithm 1 in Appendix A). First, each pre-activation[1] is normalized according to the mean and standard deviation across the mini-batch. These normalized pre-activations are then scaled and shifted by a trainable per-feature coefficient γ and a bias β.

In the time since BatchNorm was first proposed, the research community has sought to understand *why* it makes it possible to train deeper networks and leads to benefits like faster convergence. This work typically centers on the normalization aspect of BatchNorm, explicitly eliding γ or β or treating BatchNorm as a black box without particular consideration for these parameters (e.g., Santurkar et al., 2018; Bjorck et al., 2018; Yang et al., 2019; Luo et al., 2019).

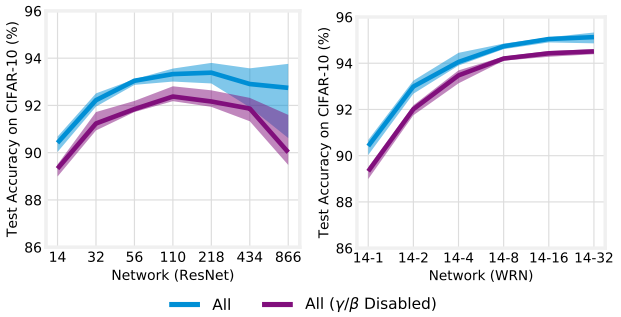

Figure 1. Test accuracy when training all parameters of the deep (left) and wide (right) ResNets in Table 1 with γ and β enabled (blue) and frozen at their initial values (purple). Except on the deepest ResNets, accuracy is about half a percent lower when γ and β are disabled.

In this paper, we focus our attention specifically on the role and expressive power of γ and β. BatchNorm is ubiquitous in modern deep learning, meaning these parameters are present by default in numerous models that researchers and practitioners train every day. Although Ioffe & Szegedy (2015) include γ and β to "restore the representation power of the network" after normalization, we understand little about what purpose this post-normalization shifting and rescaling serves in practice (if any), what role it takes on in the learned representation, and the expressive power of trainable parameters placed in this position in the network.

We investigate these questions empirically on ResNets for CIFAR-10. It is challenging to study γ and β using standard training: their presence has little effect on the overall accuracy of all but the deepest networks (Figure 1). By this metric alone, γ and β might appear redundant; however, we learn far more when we examine these parameters in isolation. To do so, we freeze all other weights at initialization and train *only* γ and β. Although the networks still retain the same number of features, only a small fraction

[1]MIT CSAIL [2]CUNY Graduate Center, Initiative for the Theoretical Sciences [3]Facebook AI Research. Correspondence to: Jonathan Frankle, Ari S. Morcos <jfrankle@mit.edu, arimorcos@fb.com>.

[1]He et al. (2016) find that performance is better when BatchNorm is applied before activation rather than after in ResNets.

of parameters (at most 0.6%) are trainable. In a broader study of freezing various parts of networks, Rosenfeld & Tsotsos (2019) briefly examined a small-scale version of this experiment with abbreviated training. However, since we are singularly interested in the role and expressive power of BatchNorm parameters, we investigate this setting in substantially greater detail: we train to completion and increase the number of features via width and depth to reach considerably higher accuracy; we vary the network initialization scheme to sample different random features and the BatchNorm initialization scheme to explore its effect; and we discover mechanisms by which these networks use this limited capacity to represent meaningful functions.

In summary we make the following contributions:

- We find that sufficiently deep convolutional ResNets (e.g., ResNet-866) train to surprisingly high accuracy on CIFAR-10 (e.g., 83%) when exclusively training the γ and β parameters associated with BatchNorm.
- When examining the values of the BatchNorm parameters, we find that the γ naturally removes features from the network, disabling between a quarter and half of all channels when training BatchNorm alone.
- We observe that, when training all parameters, γ values are smaller as networks are deeper, which we believe moderates the activations in the deepest networks where, without γ, accuracy drops (Figure 1, left).

Returning to our original questions, we find that γ and β are noteworthy both for their role in BatchNorm when training all parameters and for their expressive power. Although γ and β have little impact on the performance of shallower networks, they appear helpful for deep networks to maintain accuracy. Moreover, they have remarkable expressive power in their own right, making it possible to reach high accuracy even when all other parameters are frozen at initialization.

This last observation has broader implications for our understanding of neural networks composed of random features. By freezing all other parameters at initialization, we are training networks constructed by learning shifts and rescalings of random features. In this light, our results demonstrate that the random features available at initialization provide sufficient raw material to represent high-accuracy functions for image classification. Although prior work considers models with random features and a trainable affine output layer (e.g., Rahimi & Recht, 2009; Jaeger, 2003; Maass et al., 2002), we reveal the expressive power of networks configured such that trainable affine parameters appear after each layer of random features.

2. Related Work

BatchNorm. It is well-known that BatchNorm makes it possible to train deeper networks (He et al., 2015a) and that it causes stochastic gradient descent to converge sooner (Ioffe & Szegedy, 2015). However, the underlying mechanisms by which BatchNorm does so are debated. The original authors argue that it reduces *internal covariate shift* (ICS), in which "the distribution of each layer's inputs changes during training...requiring lower learning rates" (Ioffe & Szegedy, 2015). Santurkar et al. (2018) cast doubt on the ICS explanation by artificially inducing ICS after BatchNorm with little change in training times. Empirical evidence suggests that BatchNorm makes the optimization landscape smoother (Santurkar et al., 2018); is a "safety precaution" against exploding activations that lead to divergence in deep networks at high learning rates (Bjorck et al., 2018); and allows the network to better utilize neurons (Balduzzi et al., 2017; Morcos et al., 2018). Theoretical results suggest that BatchNorm decouples optimization of the magnitude and direction of weights (Kohler et al., 2019), which *weight normalization* (an alternative to BatchNorm; Salimans & Kingma, 2016) does explicitly; that it causes gradient magnitudes to enter an equilibrium (Yang et al., 2019); and that it leads to a novel form of regularization (Luo et al., 2019).

We focus on the role and expressive power of the affine parameters in particular, whereas the aforementioned work addresses the effect of BatchNorm on the overall optimization process. In service of this broader goal, related work generally emphasizes the normalization aspect of BatchNorm, in some cases eliding one (Kohler et al., 2019) or both of γ and β (Santurkar et al., 2018; Yang et al., 2019). Other work treats BatchNorm as a black-box without specific consideration for γ and β (Santurkar et al., 2018; Bjorck et al., 2018; Morcos et al., 2018; Balduzzi et al., 2017).

Random features. There is a lengthy history of constructing models out of random features dating back to the original perceptron (Block, 1962), which learned a linear combination of *associators*, each the inner product of the input and a fixed random vector. Since then, work on random features has served a variety of purposes. To simplify then-standard techniques, Rahimi & Recht (2009) show theoretically and empirically that learning linear combinations of random features can perform nearly as well as SVMs and Adaboost (Schapire, 2003). To fit sequential data, work on *reservoir computing* (Schrauwen et al., 2007), also known as *echo state networks* (Jaeger, 2003) or *liquid state machines* (Maass et al., 2002), learns the linear output layer of a recurrent neural network of random features; such models are capable of behaving in a stable fashion and learning useful functions. To theoretically study SGD on overparameterized networks, recent work has used two layer models in which the first layer is sufficiently wide that it changes little during training (e.g., Du et al., 2019); in the limit, these networks may be modeled with the first layer frozen at its random initialization (Yehudai & Shamir, 2019).

In all cases, these lines of work study models composed of a trainable linear layer on top of random nonlinear features. In contrast, our models have affine trainable parameters *throughout* the network, namely after each random feature in each layer. Moreover, due to the practice of placing BatchNorm before the activation function (He et al., 2016), our affine parameters occur prior to the nonlinearity.

Freezing weights at random initialization. Neural networks are initialized randomly (He et al., 2015b; Glorot & Bengio, 2010), and performance with these weights is no better than random guessing. However, it is still possible to reach high accuracy while retaining some or all of these weights. Zhang et al. (2019a) show that many individual layers in trained CNNs can be reset to their random i.i.d. initializations with little impact on accuracy. Zhou et al. (2019) and Ramanujan et al. (2019) reach high accuracy on CIFAR-10 and ImageNet merely by learning which individual weights to remove at initialization. Like our work, these findings suggest that the raw material present at initialization is sufficient to express useful representations. However, Zhou et al. and Ramanujan et al. explicitly prune individual weights, while we train shifts and rescalings of entire random features that naturally learn sparse representations.

Closest to our work, Rosenfeld & Tsotsos (2019) explore freezing various parts of networks at initialization; in doing so, they briefly examine training only γ and β for ten epochs on a DenseNet and WRN. They report reaching 61% and 30% accuracy on CIFAR-10, while we reach 80% and 70% for comparable parameter-counts. Our singular purpose is to understand the role and expressive power of these parameters, so we explore this setting in far greater depth: we train to completion, we increase the number of features to maximize performance, we explore different ways of sampling random features, we study alternate initializations for BatchNorm, and we uncover mechanisms by which these networks learn to represent functions.

3. Methodology

ResNet architecture. We train convolutional neural networks with residual connections (*ResNets*) on CIFAR-10. We focus on ResNets because they make it possible to add features by arbitrarily (a) increasing network depth without interfering with optimization and (b) increasing network width without parameter-counts becoming so large that training is infeasible. Training deep ResNets generally requires BatchNorm, so it is a natural setting for our experiments. We use the class of ResNets designed for CIFAR-10 by He et al. (2015a) and parameterized by depth and width scaling factors D and W; the exact details of the architectures and training hyperparameters are in Appendix B.

Varying depth. To vary the depth of the ResNets, we set W = 1 and vary D. We name each ResNet by the total number of trainable layers; for example, when D=5, we refer to the network as *ResNet-32* because it has $3 * 5$ blocks of two layers each (30 layers) plus the initial convolution and the fully-connected output layer making 32 layers in total. Table 1 (top) lists the specific architectures we consider and the parameter-counts of various parts of the networks. All networks have the same number of shortcut connections and output layer parameters, but deeper networks have more features and, therefore, more BatchNorm parameters.

Name	D	W	BatchNorm	Shortcut	Output	Total
ResNet-14	2	1	1.1K	2.7K	650	175K
ResNet-32	5	1	2.5K	2.7K	650	467K
ResNet-56	9	1	4.3K	2.7K	650	856K
ResNet-110	18	1	8.3K	2.7K	650	1.7M
ResNet-218	36	1	16K	2.7K	650	3.4M
ResNet-434	72	1	32K	2.7K	650	7.0M
ResNet-866	144	1	65K	2.7K	650	14M
WRN-14-1	2	1	1.1K	2.7K	650	175K
WRN-14-2	2	2	2.2K	10K	1.3K	697K
WRN-14-4	2	4	4.5K	42K	2.6K	2.8M
WRN-14-8	2	8	9.0K	165K	5.1K	11M
WRN-14-16	2	16	18K	658K	10K	44M
WRN-14-32	2	32	35K	2.6M	21K	177M

Table 1. Deep (top) and wide (bottom) ResNets that we consider and the number of parameters in each part of the network. ResNet-14 is identical to WRN-14-1, with WRN-14-2 twice the width of ResNet-14, WRN-14-4 four times the width, and so on. Each network contains 3 sets of D residual blocks and $16W$, $32W$, and $64W$ filters in each set of blocks, respectively.

Varying width. To vary the width of the ResNets, we set D=2 (ResNet-14) and vary W. We name each wide ResNet by the number of trainable layers and W: WRN-14-W.[2] Table 1 (bottom) lists the specific architectures we consider and the parameter-counts of various parts of the networks. As the network width doubles, the number of BatchNorm and output layer parameters doubles, but the number of convolutional and shortcut parameters quadruples because the number of incoming and outgoing channels both double.

Replicates. All experiments are shown as the mean across five runs with different initializations, data orders, and augmentation. Error bars are one standard deviation.

4. Training Only BatchNorm

Our goal is to better understand the role and expressive power of the affine parameters in BatchNorm, which typically receive less research attention than the normalization aspect (Santurkar et al., 2018; Bjorck et al., 2018; Yang et al., 2019; Luo et al., 2019). When training the full network,

[2]WRN-14-W is the same as WRN-16-W by Zagoruyko & Komodakis (2016). For consistency as we vary width and depth, we follow the naming convention of He et al. (2015a).

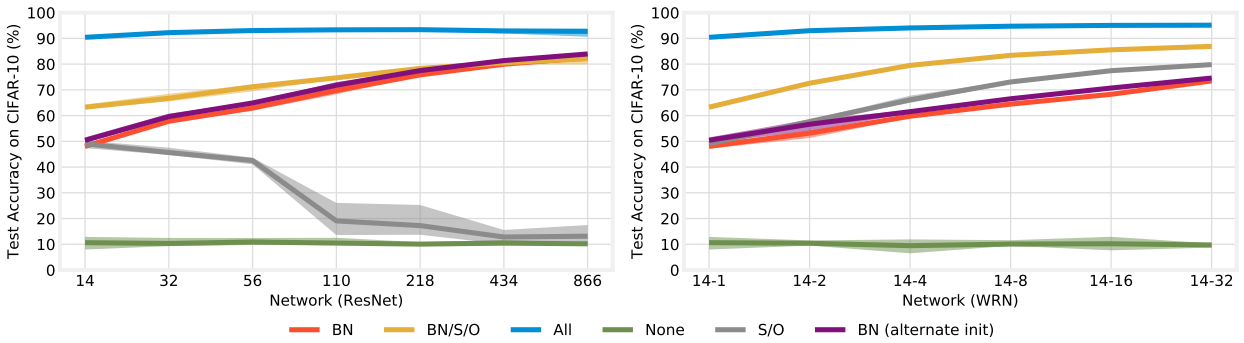

Figure 2. Test accuracy for the experiments in Section 4 across all ResNet depths (left) and widths (right). Each line is a different combination of parameters that are enabled for training. BN = BatchNorm parameters. S = Shortcut parameters. O = Output layer. Note that normalization is present in all cases.

these parameters have relatively little effect on accuracy (Figure 1). To isolate these parameters, we therefore perform the following experiment: we freeze all other weights at initialization and train *only* the γ and β parameters.

In Figure 2, we plot accuracy when training only γ and β in red as we vary depth (left) and width (right). We also include two baselines: when all parameters are trainable (i.e., training normally) in blue and no parameters are trainable (i.e., only updating normalization statistics μ and σ, which we expect should lead to chance performance) in green.

Case study: ResNet-110. As a case study, we focus first on ResNet-110. When all 1.7M parameters are trainable (blue), the network reaches 93.3% test accuracy on CIFAR-10. When no parameters are trainable (green), the network unsurpisingly remains at 10.1% accuracy—random guessing. When training just the 8.3K BatchNorm parameters that can only shift and rescale random features, the network achieves test accuracy of 69.7%. This accuracy is well above what we would expect by chance, suggesting that these parameters have significant representational capacity. We note, however, that there is still a large gap between this performance and that of training all parameters.

Changing training so drastically may also affect the rate at which learning occurs. In Figure 3, we plot the test accuracy throughout training of ResNet-110 in both of these configurations. Although training only BatchNorm (red) reaches lower accuracy than training all parameters (blue), the two training curves improve and plateau at approximately the same rate. When the learning rate drops at epochs 80 and 120, accuracy jumps similarly in both cases. In summary, training only BatchNorm does not seem to drastically alter the time needed to converge.

While our motivation is to study the role of the affine parameters in BatchNorm, our results also have implications for the expressive power of neural networks composed of random features. All of the features in the network (i.e., convolutions and linear output layer) are fixed at their random initializations; all that the BatchNorm parameters can

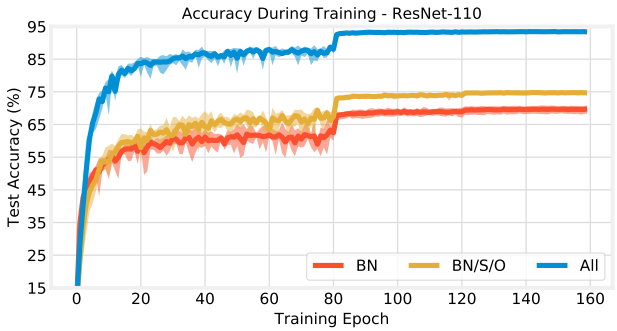

Figure 3. Test accuracy of ResNet-110 during training when training all parameters, just BatchNorm, and BatchNorm with shortcut and output parameters. Learning appears to occur at a similar rate in all experiments, although they reach different accuracies.

do is shift and rescale the normalized activation maps that these features produce in each layer. From this perspective, our experiment can be seen as training neural networks parameterized by shifts and rescalings of random features. In this light, our results show that it is possible to reach high accuracy on CIFAR-10 using only the random features that were available at initialization.

Varying depth and width. Viewed from the random features perspective, the expressivity of the network will be limited by the number of features available for the BatchNorm parameters to combine. In this fashion, if we increase the number of features, we expect that accuracy will improve. We can increase the number of features available to the BatchNorm layers in two ways: either by increasing the network's width or by increasing its depth.

Figure 2 presents the performance when increasing the depth (left) and width (right). As expected, the accuracy of training just BatchNorm improves as we deepen or widen the network. ResNet-14 achieves 48% accuracy when training only BatchNorm, but deepening the network to 866 layers or widening it by a factor of 32 increases accuracy to 83% and 74%, respectively. Although these accuracies fall short of those when training all parameters (93% and 95%), they are surprisingly high considering we are training just 65K and

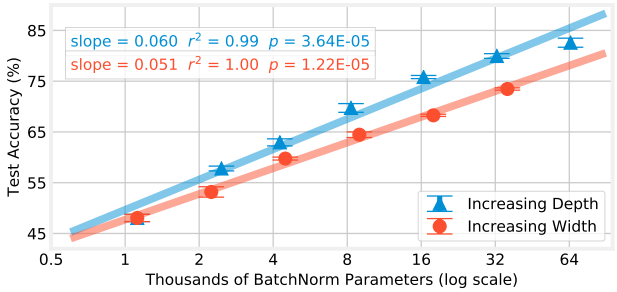

Figure 4. When training only the BatchNorm parameters: the relationship between BatchNorm parameter count (x-axis) and test accuracy (y-axis) as we increase depth (blue) and width (red).

35K parameters out of 14M and 177M total and that these parameters can merely scale and shift random features.

We also observe that the accuracy is six percentage points higher for ResNet-434 than for WRN-14-32 even though both have the same number of BatchNorm parameters. This raises a further question: for a fixed budget of BatchNorm parameters (and, thereby, a fixed number of random features), is performance always better when increasing depth rather than increasing width? Figure 4 plots the relationship between the number of BatchNorm parameters (x-axis) and test accuracy (y-axis) when increasing depth (blue) and width (red) from the common starting point of ResNet-14. In both cases, accuracy increases linearly as the number of BatchNorm parameters doubles. Interestingly, the trend is 17% steeper when increasing depth than width, meaning that, for the networks we consider, increasing depth leads to higher accuracy than increasing width for a fixed BatchNorm parameter budget.[3]

Making shortcuts and outputs trainable. It is possible to train such deep ResNets due to shortcut connections, which propagate gradients to the earlier layers of the network (He et al., 2015a; Balduzzi et al., 2017). In our ResNets, nearly all shortcut connections use the identity function and have no trainable parameters. However, the two shortcuts that downsample (those where the number of filters increase) use 1x1 convolutions. It is possible that, by freezing these parameters, we have inhibited the ability of our networks to propagate gradients to lower layers and take full advantage of the BatchNorm parameters. The same reasoning may apply to the output layer, through which all gradients pass.

To evaluate the impact of this restriction, we enable training for the shortcut and output layer parameters in addition to the BatchNorm parameters (Figure 2, yellow). For ResNet-110, making these additional 3.4k parameters trainable improves accuracy by five points to 74.7%, suggesting that freezing these parameters indeed affected performance. However, on deeper networks, the returns from making shortcut and output parameters trainable diminish, with no improvement in accuracy on ResNet-866. If freezing these parameters were really an impediment to gradient propagation, we would expect the deepest networks to benefit most, so this evidence does not support our hypothesis.

As an alternate explanation, we propose that accuracy improves simply due to the presence of more trainable parameters. As evidence for this claim, the shallowest networks—for which shortcut and output parameters make up a larger proportion of weights—benefit most when these parameters are trainable. Doing so quadruples the parameter-count of ResNet-14, which improves from 48% to 63% accuracy, and adds a further 2.6M parameters (315x the number of BatchNorm parameters) to WRN-14-32, which improves from 74% to 87% accuracy. Furthermore, we freeze BatchNorm and train only the shortcuts and outputs, performance of shallower networks is even better than training just BatchNorm, reaching 49% for ResNet-14 and 80% for WRN-14-32.

Varying the feature initialization scheme. To this point, we have shown that ResNets comprising shifts and rescalings of random features can reach high accuracy on CIFAR-10. However, we have only studied one class of random features: those produced by He normal initialization (He et al., 2015b). It is possible that other initializations may produce random features that result in higher accuracy. For example, Ramanujan et al. (2019) find higher-performance sparse subnetworks at initialization when sampling from a normal distribution rather than a uniform distribution and when binarizing the initializations at $\pm\sigma$ (where σ is the per-layer standard deviation). We explored the performance of training only BatchNorm when initializing the rest of the network according to a normal distribution (as we have used thus far), a uniform distribution, binarized weights, and samples from a normal distribution that were orthogonalized using SVD. We find that changing the initialization scheme has no effect on the accuracy of the networks (results in Appendix C, Figure A2), suggesting that the role of initialization for random features may be less pronounced than when searching for a sparse subnetwork. With that said, we have only studied existing initializations designed to make features easy to optimize; it is possible that other, non-standard initializations (such as those that reflect the features learned by CNNs) may lead to higher performance.

Varying the BatchNorm initialization scheme. Similarly, the BatchNorm initialization scheme we use ($\gamma \sim \mathcal{U}[0, 1]$, $\beta = 0$, the default in our version of PyTorch) was designed to make it easier to train all parameters. It is possible that a different initialization scheme may be more suitable when training only BatchNorm. We studied three other schemes: ($\gamma = 1$, $\beta = 0$), ($\gamma \sim \mathcal{U}[-1, 1]$, $\beta = 0$), and ($\gamma = 1$,

[3] We expect that performance will eventually saturate and that further expansion will lead to diminishing improvements; however, we do not encounter this point in our experiments, in which we use the deepest and widest ResNets that fit on a V100.

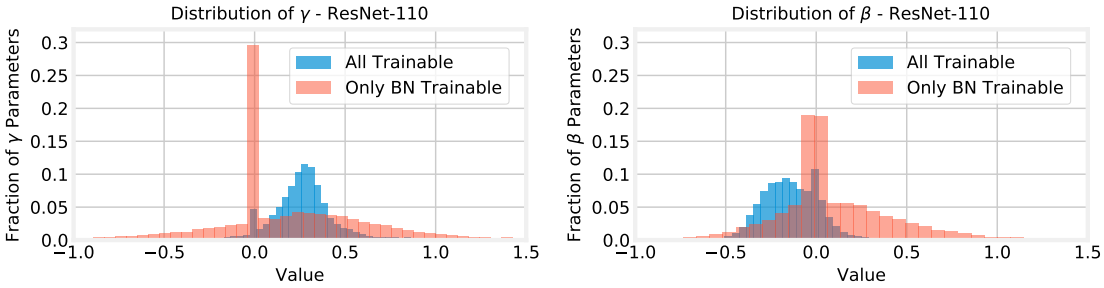

Figure 5. The distribution of the γ (left) and β (right) parameters of BatchNorm in ResNet-110 when training all parameters (*All*) and when just training the BatchNorm parameters (*BN*). These values are aggregated across all layers, and totals are averaged across five replicates. When only training BatchNorm, more than a quarter of all γ parameters have a magnitude < 0.01.

$\beta = 1$). The first scheme is another standard practice for BatchNorm, the second scheme tests centering γ around 0, and the third tests increasing the initial value of β (which has the effect of ensuring that a greater fraction of normalized features pass through the ReLU activation). The first two schemes led to no discernible change in performance when training all parameters or BatchNorm only. However, the third scheme, where $\beta = 1$, increased the accuracy of the BatchNorm-only experiments by 1 to 3 percentage points across all depths and widths (purple line in Figure 2), improving ResNet-866 to 84% accuracy and WRN-14-32 to 75%. Interestingly, doing so *lowered* the accuracy when all parameters are trainable and caused many runs to fail entirely (Appendix D). We conclude that (1) ideal initialization schemes for the BatchNorm parameters in the BatchNorm-only and standard scenarios appear to be different and (2) the standard training regime is indeed sensitive to the choice of BatchNorm initializations.

Summary. Our goal in this section was to study the expressive power of γ and β in BatchNorm. We found that training only these parameters leads to surprisingly high accuracy (albeit lower than that of training all parameters). By increasing the quantity of these parameters and the random features available to combine, we found that we can further improve this accuracy. Finally, the shortcut and output parameters only moderately affect accuracy except when they substantially alter the parameter-count, BatchNorm-only accuracy improves when setting β to 1, and other initialization changes had no discernible effect.

5. Examining the Values of γ and β

In the previous experiments, we showed that training just γ and β leads to high accuracy. Considering the severe restrictions placed on the network by freezing all features at their random initializations, we are interested in *how* the network achieves this performance. In what ways does the learned representation change between this training regime and that in which all parameters are trainable? This comparison offers insights into the role γ and β take on in the context of standard training and when training only BatchNorm.

Case study: ResNet-110. To study these questions, we plot the distributions of the values of γ and β learned by ResNet-110 when all parameters are trainable (blue) and when only γ and β are trainable (red) in Figure 5. When training all parameters, the distribution of γ is roughly Gaussian with a mean of 0.27. The standard deviation of 0.21 is small enough that that 95% of γ values are positive. When training just BatchNorm, the distribution of γ has a similar mean (0.20) but a much wider standard deviation (0.48), meaning that 25% of the γ values are negative.

Most notably, the BatchNorm-only case has a spike at 0: 27% of all γ values have a magnitude of less than 0.01 and 37% have a magnitude less than 0.1. In other words, the network learns to effectively disable between a quarter and a third of all features, implicitly pruning the network channels in a structured fashion (He et al., 2017). Other than standard weight decay for this architecture, we take no additional steps to induce this sparsity; it simply occurs naturally when we train in this fashion. This behavior indicates that an important part of the network's representation is the set of random features that it learns to *ignore*. When all parameters are trainable, there is a similar but much smaller spike at 0, with 9% of γ magnitudes less than 0.01 and 14% of γ magnitudes less than 0.1. This data suggests that disabling features is a natural behavior of γ, although it is exaggerated when only γ and β are trainable.

The values of β are also distributed in a roughly Gaussian fashion. When all parameters are trainable, the mean is below zero (-0.13) and the standard deviation is smaller (0.18), while, in the BatchNorm-only case, the mean is positive (0.09) and the standard deviation is larger (0.37). In both cases, we see a spike near 0, which is again much larger when only training BatchNorm. However, our interpretation is different due to the additive role of β: it will only eliminate a feature when it is extremely negative, not when it is near 0. We therefore find the γ values near zero to be more noteworthy, since they may directly sparsify the network.

Varying depth and width. The same γ behavior holds across all depths and widths, seemingly disabling a large fraction of features. When training only BatchNorm, $|\gamma| <$

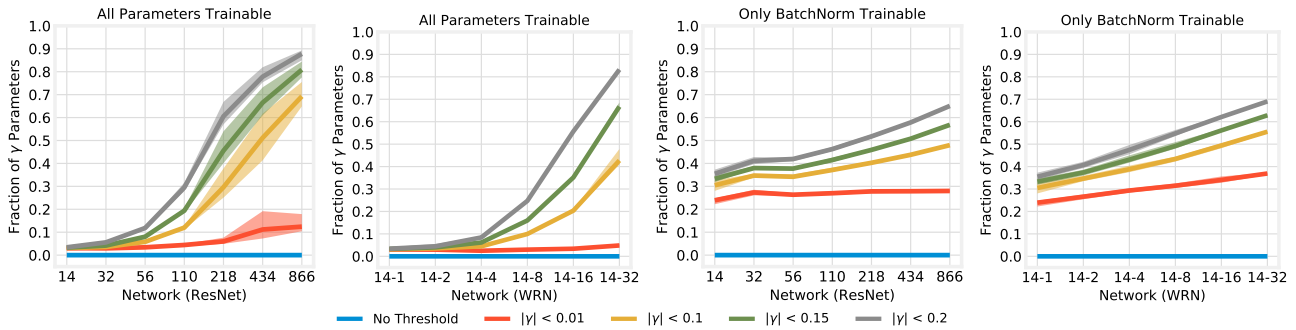

Figure 6. Fraction of γ parameters for which $|\gamma|$ is smaller than various thresholds when all parameters are trainable (left plots) and only BatchNorm is trainable (right plots). The first and third plots vary network depth and the second and fourth plots vary width.

0.1 in between a quarter and a third of all cases (Figure 6 right, red). When all parameters are trainable (Figure 6 left, red), this occurs for about 3% of γ values in all but the deepest ResNets. For the particular threshold of 0.01, the fraction of features disabled remains relatively consistent as we increase depth and width, although it increases to about 10% for the deepest networks when all parameters are trainable and to about 37% for the widest networks when only BatchNorm is trainable.

The choice of threshold 0.01 is arbitrary. In Figure 6, we also use several bigger thresholds. For these bigger thresholds, as we make the network deeper or wider, a much larger fraction of γ parameters is below a given threshold. When all parameters are trainable, this trend is especially pronounced. For example, the fraction of γ parameters with a magnitude less than 0.1 increases from 3% for ResNet-14 to 69% for ResNet-866. We see the same trend when training only BatchNorm, although the increase is less dramatic. In general, this data means that more BatchNorm parameters are close to 0 for deeper and wider networks. We hypothesize that the BatchNorm parameters are moderating the scale of the normalized activations in order to keep them from exploding. This might explain why disabling γ and β when training the full network disproportionately hurts deeper networks in Figure 1. However, this role must be of limited importance except in extreme cases, since disabling γ and β has little effect on shallower networks and WRNs.

Explicitly clamping γ. We hypothesized that small values of γ, especially when only training BatchNorm, meant that the network was entirely removing unhelpful features—that it was using γ to implicitly *sparsify* features. Yet just because many values of γ are *close* to zero does not mean they can be set *equal* to zero; it may be that small values of γ play some important role in the representation.

To evaluate the extent to which small values of γ are, in effect, sparsifying the network, we explicitly set these parameters to zero and measure the accuracy of the network afterwards (Figure 7). Clamping all values of $\gamma < 0.01$ to zero has no effect on the accuracy of any network, suggesting that these features are indeed expendable. This is true both when all parameters are trainable and when only BatchNorm is trainable; in the latter case, this means that between 24% (ResNet-14) and 37% (WRN-14-32) of features can safely be disabled. This result confirms our hypothesis that values of γ that are close to zero reflect features that are irrelevant to the representation learned by the network.

As we increase the threshold to 0.1, accuracy remains at its full value for all but the deepest and widest networks. In the BatchNorm-only case, ResNet-110 and deeper networks lose only one to two percentage points of accuracy despite the fact that between 38% and 49% of features are disabled. When all parameters are trainable, accuracy drops substantially for ResNet-434, ResNet-866, and WRN-14-32. As Figures 6 reflects, these networks have smaller values of γ and lose the most features with these thresholds.

Thus far, our clamping has focused on values of γ that are close to zero. However, other papers have shown that it is possible to achieve high accuracy by learning a *binarized* mask in which individual weights at initialization are either retained or pruned (Zhou et al., 2019; Ramanujan et al., 2019). From the perspective of these papers, our clamping experiments impose structured feature-wise sparsity rather than the unstructured weight-wise sparsity they explore. However, unlike these papers, we do not explicitly optimize for values of γ that are 0 or 1. The closest analog in our setting would involve clamping values of γ to either 0 or 1. We explored clamping γ in this manner, e.g., setting all values of γ below a threshold to 0 and all others to ± 1 or $\pm \mu$ (where μ is the mean magnitude of γ). However, none of these experiments resulted in accuracy better than random guessing, suggesting that the values of γ that are far from zero are important to the learned representation.

Summary. Our goal in this section was to understand how the internal representations of the networks differed when training all parameters and training only BatchNorm. To do so, we examined the values of γ and β. When all parameters were trainable, we found that γ values became smaller in deeper networks, which might explain the role these param-

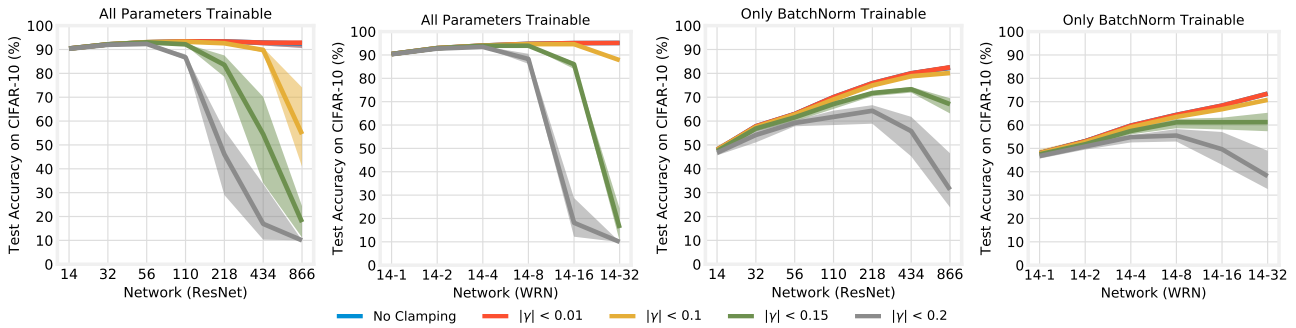

Figure 7. Accuracy when clamping γ values below various thresholds to 0. The line for *No Clamping* is obscured behind $|\gamma| < 0.01$.

eters play in the accuracy of ResNet-866. When training only BatchNorm, we found γ to have a larger variance and a spike at 0. This spike corresponds to channels that we could disable entirely with no effect on the performance, suggesting that training only BatchNorm led to representations that remove a quarter to a third of all features. However, sparsity was not entirely responsible for the representation; the fact that we were unable to binarize values of γ suggests that specific values of non-zero γ parameters are also important.

6. Discussion and Conclusions

Our results demonstrate that it is possible to reach surprisingly high accuracy when training only the affine parameters associated with BatchNorm and freezing all other parameters at their original initializations. We make several observations about the implications of these results.

BatchNorm. Although the research community typically focuses on the normalization aspect of BatchNorm, our results emphasize that the affine parameters are remarkable in their own right. With respect to their role in BatchNorm, these parameters matter relatively little to the performance of shallower networks (Figure 1). However, for particularly deep ResNets, the presence of these parameters begins to affect accuracy. We connect this behavior to our observation in Section 5 that values of γ are smaller as the networks become deeper; we hypothesize that smaller γ values moderate the activations in order to keep them from exploding.

With respect to the expressive power of the affine parameters, we find that γ and β alone are able to create surprisingly high-accuracy neural networks even when all other parameters are frozen at their original initializations. We conclude that these parameters have substantial expressive power in their own right, which is particularly impressive considering they can only scale and shift the activation maps that result from randomly initialized convolutions.

Random features. From a different perspective, our experiment is a novel way of training networks constructed out of random features. While prior work considers training only a linear output layer on top of random nonlinear features, we distribute affine parameters throughout the network after each feature in each layer. This configuration gives the network greater expressive power and the ability to reach relatively high accuracy on CIFAR-10. Unlike Rahimi & Recht (2009), we do not propose that our method provides practical improvements; it is still necessary to fully backpropagate to update the deep BatchNorm parameters. While we do not contend that our configuration offers immediate theoretical insights as does the scheme in Yehudai & Shamir (2019), it would be interesting to better understand its theoretical capabilities. We see our results as further evidence (alongside the work of Zhou et al. (2019) and Ramanujan et al. (2019)) that the raw material present at random initialization is sufficient to create performant networks.

Limitations and future work. There are several ways to expand our study to improve the confidence and generality of our results. We only consider ResNets trained on CIFAR-10, and it would be valuable to see how our results scale to more challenging settings (e.g., ImageNet). We use standard hyperparameters and do not search for hyperparameters that specifically perform well when training only BatchNorm. Throughout the paper, we acknowledge many negative results, particularly when varying the initialization schemes for features and BatchNorm; there is a large design space of initializations that it would be valuable to further explore.

In follow-up work, we are especially interested in better understanding the relationship between random features and the representations learned by the BatchNorm parameters. In particular, are there initialization schemes for the convolutional layers that allow training only BatchNorm to reach better performance than using conventional initializations? In addition, is it possible to rejuvenate convolutional filters that are eliminated by BatchNorm (in a manner similar to Cohen et al. (2016)) to improve the overall accuracy of the network? Finally, can we better understand the role of a per-feature bias and coefficient outside the context of BatchNorm? For example, we could add these parameters when using techniques that train deep networks without BatchNorm, such as WeightNorm (Salimans & Kingma, 2016) and FixUp initialization (Zhang et al., 2019b).

References

Balduzzi, D., Frean, M., Leary, L., Lewis, J., Ma, K. W.-D., and McWilliams, B. The shattered gradients problem: If resnets are the answer, then what is the question? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 342–350. JMLR. org, 2017.

Bjorck, N., Gomes, C. P., Selman, B., and Weinberger, K. Q. Understanding batch normalization. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7694–7705. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7996-understanding-batch-normalization.pdf.

Block, H.-D. The perceptron: A model for brain functioning. i. *Reviews of Modern Physics*, 34(1):123, 1962.

Cohen, J. P., Lo, H. Z., and Ding, W. Randomout: Using a convolutional gradient norm to win the filter lottery. *CoRR*, abs/1602.05931, 2016. URL http://arxiv.org/abs/1602.05931.

Du, S. S., Zhai, X., Poczos, B., and Singh, A. Gradient descent provably optimizes over-parameterized neural networks. 2019.

Glorot, X. and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256, 2010.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2015a.

He, K., Zhang, X., Ren, S., and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015b.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, 2017.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045167.

Jaeger, H. Adaptive nonlinear system identification with echo state networks. In *Advances in neural information processing systems*, pp. 609–616, 2003.

Kohler, J., Daneshmand, H., Lucchi, A., Hofmann, T., Zhou, M., and Neymeyr, K. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. In Chaudhuri, K. and Sugiyama, M. (eds.), *Proceedings of Machine Learning Research*, volume 89 of *Proceedings of Machine Learning Research*, pp. 806–815. PMLR, 16–18 Apr 2019. URL http://proceedings.mlr.press/v89/kohler19a.html.

Luo, P., Wang, X., Shao, W., and Peng, Z. Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJlLKjR9FQ.

Maass, W., Natschläger, T., and Markram, H. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural computation*, 14(11):2531–2560, 2002.

Morcos, A., Barrett, D. G., Rabinowitz, N. C., and Botvinick, M. On the importance of single directions for generalization. In *Proceeding of the International Conference on Learning Representations*, 2018.

Rahimi, A. and Recht, B. Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning. In *Advances in neural information processing systems*, pp. 1313–1320, 2009.

Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., and Rastegari, M. What's hidden in a randomly weighted neural network?, 2019.

Rosenfeld, A. and Tsotsos, J. K. Intriguing properties of randomly weighted networks: Generalizing while learning next to nothing. In *2019 16th Conference on Computer and Robot Vision (CRV)*, pp. 9–16. IEEE, 2019.

Salimans, T. and Kingma, D. P. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in neural information processing systems*, pp. 901–909, 2016.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization?

In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 2483–2493. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/7515-how-does-batch-normalization-help-optimization.pdf.

Schapire, R. E. The boosting approach to machine learning: An overview. In *Nonlinear estimation and classification*, pp. 149–171. Springer, 2003.

Schrauwen, B., Verstraeten, D., and Van Campenhout, J. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pp. 471–482, 2007.

Yang, G., Pennington, J., Rao, V., Sohl-Dickstein, J., and Schoenholz, S. S. A mean field theory of batch normalization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=SyMDXnCcF7.

Yehudai, G. and Shamir, O. On the power and limitations of random features for understanding neural networks. In *Advances in Neural Information Processing Systems*, pp. 6594–6604, 2019.

Zagoruyko, S. and Komodakis, N. Wide residual networks. In Richard C. Wilson, E. R. H. and Smith, W. A. P. (eds.), *Proceedings of the British Machine Vision Conference (BMVC)*, pp. 87.1–87.12. BMVA Press, September 2016. ISBN 1-901725-59-6. doi: 10.5244/C.30.87. URL https://dx.doi.org/10.5244/C.30.87.

Zhang, C., Bengio, S., and Singer, Y. Are all layers created equal? 2019a.

Zhang, H., Dauphin, Y. N., and Ma, T. Residual learning without normalization via better initialization. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=H1gsz30cKX.

Zhou, H., Lan, J., Liu, R., and Yosinski, J. Deconstructing lottery tickets: Zeros, signs, and the supermask. In *Advances in Neural Information Processing Systems*, 2019.

A. Formal Restatement of BatchNorm

Algorithm 1 Batch normalization at train-time.

1: Let $x^{(1)}, ..., x^{(n)}$ be the pre-activations for a particular unit in a neural network for inputs 1 through n in a mini-batch.
2: Let $\mu = \frac{1}{n}\sum_{i=1}^{n} x^{(i)}$
3: Let $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x^{(i)} - \mu)^2$
4: The batch-normalized pre-activation $\hat{x}^{(i)} = \gamma\frac{x^{(i)}-\mu}{\sqrt{\sigma^2}} + \beta$ where γ and β are trainable parameters.
5: The activations are $f(\hat{x}^{(i)})$ where f is the activation function.

B. Details of ResNets for CIFAR-10

ResNet architectures. We use the ResNets for CIFAR-10 as described by He et al. (2015a). Each network has an initial 3x3 convolutional layer from the three input channels to $16W$ channels. Afterwards, the network contains $3D$ residual blocks. Each block has two 3x3 convolutional layers surrounded by a shortcut connection with the identity function and no trainable parameters. The first set of D blocks have $16W$ filters, the second set of D blocks have $32W$ filters, and the third set of D blocks have $64W$ filters. The first layer in each set blocks downsamples by using a stride of 2; the corresponding shortcut connection has a 1x1 convolution that also downsamples. After the convolutions, each remaining channel undergoes average pooling and a fully-connected layer that produces ten output logits. Each convolutional layer is followed by batch normalization *before* the activation function is applied (He et al., 2016).

Hyperparameters. We initialize all networks using He normal initialization (He et al., 2015b), although we experiment with other initializations in Section 4. The γ parameters of BatchNorm are sampled uniformly from $[0, 1]$ and the β parameters are set to 0. We train for 160 epochs with SGD with momentum (0.9) and a batch size of 128. The initial learning rate is 0.1 and drops by 10x at epochs 80 and 120. We perform data augmentation by normalizing per-pixel, randomly flipping horizontally, and randomly translating by up to four pixels in each direction. We use standard weight decay of 1e-4 on all parameters, including BatchNorm.

C. Varying the Feature Initialization

In Section 4, we discuss our attempts to vary the feature initialization scheme. We found that none of the alternative schemes we tried led to improved performance. For completeness, we present this data in Figure A2.

D. Varying the BatchNorm Initialization

In Section 4, we discuss our attempts to vary the BatchNorm initialization scheme. We find that initializing β to 1 and γ to 1 improves performance when training only BatchNorm but hurts performance when training all parameters. In Figure A3, we plot all of the experiments in Section 4 with this alternate BatchNorm initialization across all depths and widths. Accuracy is lower (or the network fails to learn) for all experiments *except* training only BatchNorm, for which accuracy improves by 1-3 percentage points.

In Figure A4, we plot the distribution of γ and β under this BatchNorm initialization scheme for ResNet-110 when training all parameters and when training only BatchNorm. When all parameters are trainable, a much larger portion of γ values are close to zero than for the standard initialization scheme (Figure 5). It is possible that, to compensate for the large values of β, the γ values shrink, disabling features and reducing performance.

We find the opposite for when only training BatchNorm: far fewer values of γ are close to zero than when using the standard initialization scheme (Figure 5). It is possible that this initialization scheme made it possible for the BatchNorm-only configuration to better utilize the features in the network, potentially explaining the improved performance over the standard initialization.

E. Ablating γ, β, and Normalization

In this experiment, we examine the specific contributions of γ, β, and the normalization step in producing the accuracies we observe when only training BatchNorm in Section 4. Figure A1 plots the accuracy of training BatchNorm (*γ/β/N*, i.e., γ and

β are trainable and normalization is enabled), just γ and normalization (γ*/N*), just β and normalization (β*/N*), and γ and β without normalization (γ*/*β). We also include baselines with all parameters trainable (*All*) and no parameters trainable (*None*); normalization is enabled in both cases.

Training the γ parameters (γ*/N*) results in higher accuracy than training the β parameters (β*/N*), only a few percentage points lower than training both together (γ*/*β*/N*). This result is unsurprising: without γ, all features will have the same scale due to normalization. Disabling normalization (γ*/*β) has a similar effect to disabling β: accuracy decreases by a few percentage points. In other words, the normalization itself seems to allow the network to reach higher accuracy. On deeper networks, normalization is essential: training does not succeed when normalization is disabled and accuracy is no different than the *None* baseline.

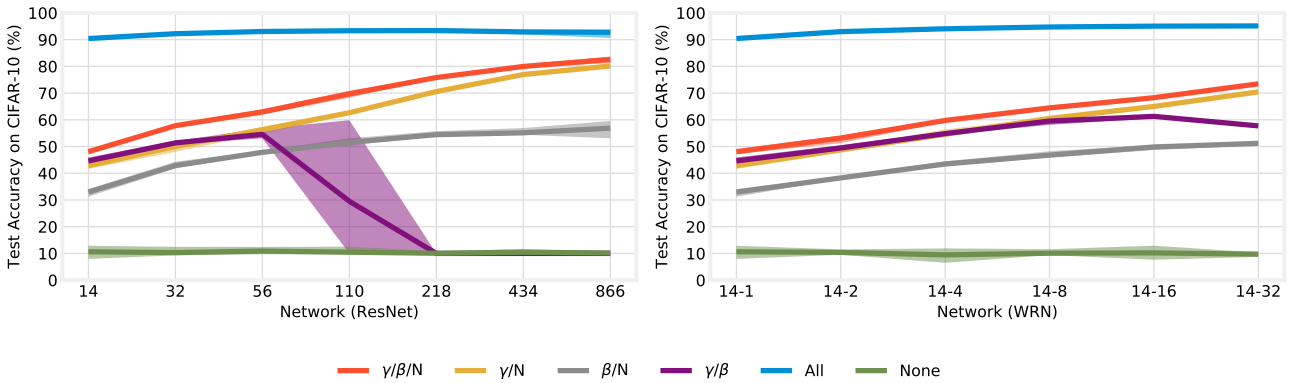

Figure A1. Test accuracy for the experiments in Appendix E across all ResNet depths (left) and widths (right). Each line is a different combination of making γ parameters trainable (γ), making β parameters trainable (β), and enabling normalization (N). We also include baselines where all parameters are trainable (*All*) and no parameters are trainable (*None*); in both cases, normalization is enabled.

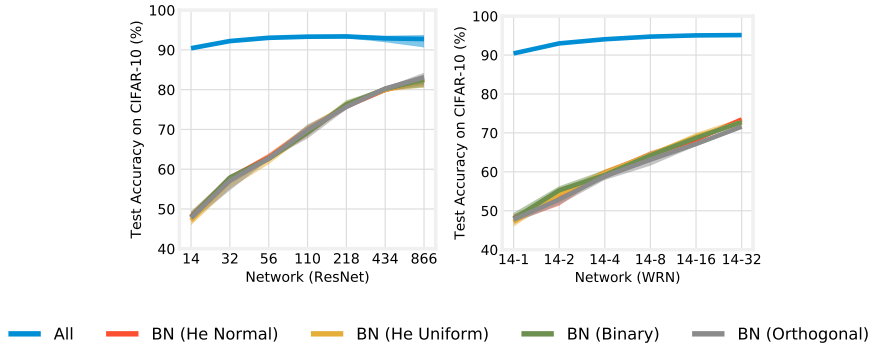

Figure A2. Varying the initialization scheme when training only the BatchNorm parameters.

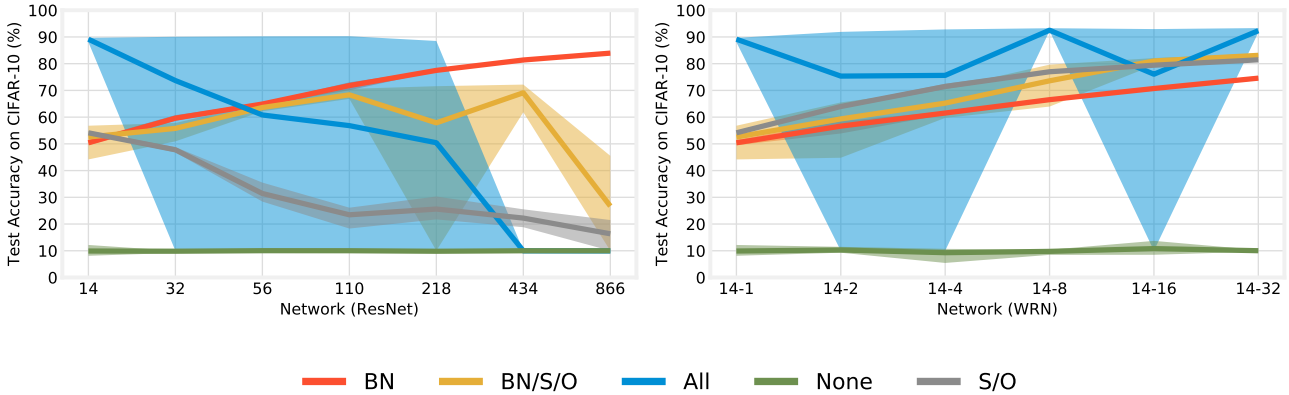

Figure A3. Test accuracy for the experiments in Section 4 across all ResNet depths (left) and widths (right) when using an alternate BatchNorm initialization scheme where $\beta = 1$ and $\gamma = 1$. Each line is a different combination of parameters that are enabled for training. BN = BatchNorm parameters. S = Shortcut parameters. O = Output layer. Note that normalization is present in all cases.

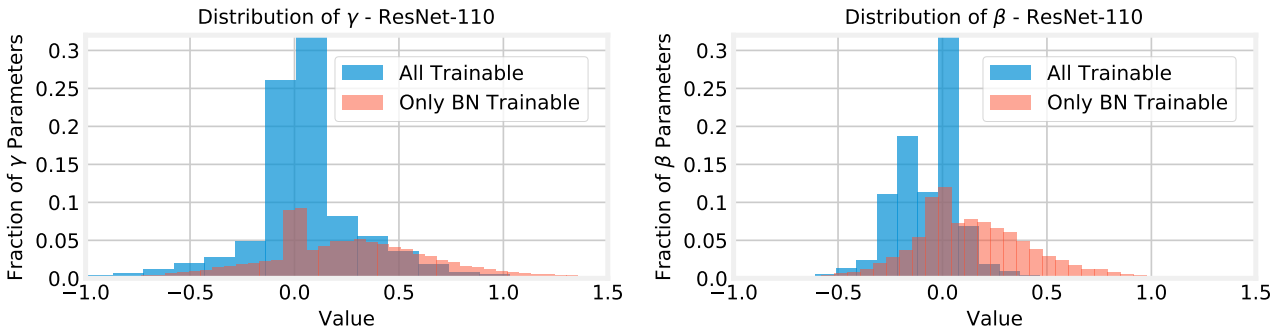

Figure A4. The distribution of the γ (left) and β (right) parameters of BatchNorm in ResNet-110 when training all parameters (*All*) and when just training the BatchNorm parameters (*BN*) when using the alternate BatchNorm initialization scheme where $\gamma = 1$ and $\beta = 1$. These values are aggregated across all layers, and totals are averaged across five replicates.