NORMALITY NORMALIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

The normal distribution plays a central role in information theory – it is at the same time the best-case signal and worst-case noise distribution, has the greatest representational capacity of any distribution, and offers an equivalence between uncorrelatedness and independence for joint distributions. Accounting for the mean and variance of activations throughout the layers of deep neural networks has had a significant effect on facilitating their effective training, but seldom has a prescription for precisely what distribution these activations should take, and how this might be achieved, been offered. Motivated by the information-theoretic properties of the normal distribution, we address this question and concurrently present normality normalization: a novel normalization layer which encourages normality in the feature representations of neural networks using the power transform and employs additive Gaussian noise during training. Our experiments comprehensively demonstrate the effectiveness of normality normalization, in regards to its generalization performance on an array of widely used model and dataset combinations, its strong performance across various common factors of variation such as model width, depth, and training minibatch size, its suitability for usage wherever existing normalization layers are conventionally used, and as a means to improving model robustness to random perturbations.

1 INTRODUCTION

The normal distribution is unique – information theory shows that among all distributions with the same mean and variance, a signal following this distribution encodes the maximal amount of information (Shannon, 1948). This can be viewed as a desirable property in learning systems such as neural networks, where the activations of successive layers equivocates to successive representations of the data.

Moreover, a signal following the normal distribution is maximally robust to random perturbations (Cover & Thomas, 2006), and thus presents a desirable property for the representations of learning systems; especially deep neural networks, which are susceptible to random (Ford et al., 2019) and adversarial (Szegedy et al., 2014) perturbations. Concomitantly, the normal distribution is information-theoretically the worst-case perturbative noise distribution (Cover & Thomas, 2006), which suggests models gaining robustness to Gaussian noise should be robust to any other form of random perturbations.

Furthermore, normality in the representations of deep neural networks imbues them with other useful properties, such as producing probabilistic predictions with calibrated uncertainty estimates (Guo et al., 2017), and rendering them amenable to a Bayesian interpretation (Lee et al., 2017). This suggests that developing a method for enforcing and maintaining normality throughout model training is of general value.

We show that encouraging deep learning models to encode their activations using the normal distribution in conjunction with applying additive Gaussian noise during training, helps improve generalization. We do so by means of a novel layer – normality normalization – so-named because it applies the power transform, a technique used to gaussianize data (Box & Cox, 1964; Yeo & Johnson, 2000), and because it can be viewed as an augmentation of existing normalization techniques such as batch (Ioffe & Szegedy, 2015), layer (Ba et al., 2016), instance (Ulyanov et al., 2016), and group (Wu & He, 2018) normalization.

Our experiments comprehensively demonstrate the general effectiveness of normality normalization in terms of its generalization performance, its strong performance across various common factors of variation such as model width, depth, and training minibatch size, which furthermore help highlight when and why it is effective, its suitability for usage wherever existing normalization layers are conventionally used, and its effect on improving model robustness under random perturbations.

In Section 2 we outline some of the desirable properties normality can imbue in learning models, which serve as motivating factors for the development of normality normalization. In Section 3 we provide a brief background on the power transform, before presenting normality normalization in Section 4. In Section 5 we describe our experiments, analyze the results, and explore some of the properties of models trained with normality normalization. In Section 6 we comment on related work and discuss some possible future directions. Finally in Section 7 we contextualize normality normalization in the broader deep learning literature, and provide a few concluding remarks.

2 Motivation

In this section we present motivating factors for encouraging normality in feature representations in conjunction with using additive random noise during learning. Section 5 substantiates the applicability of the motivation through the experimental results.

2.1 Mutual Information Game & Noise Robustness

2.1.1 Overview of the Framework

The normal distribution is at the same time the best possible signal distribution, and the worst possible noise distribution; a result which can be studied in the context of the Gaussian channel (Shannon, 1948), and through the lens of the mutual information game (Cover & Thomas, 2006). In this framework, X and Z denote two independent random variables, representing the input signal and noise, and $Y = X + Z$ is the output. The mutual information between X and Y is denoted by $I\left(X;Y\right)$; X tries to maximize this term, while Z tries to minimize it. Both X and Z can encode their signal using any probability distribution, so that their respective objectives are optimized for.

Information theory answers the question of what distribution X should choose to maximize $I\left(X;Y\right)$. It also answers the question of what distribution Z should choose to minimize $I\left(X;Y\right)$. As shown by the following theorem, remarkably the answer to both questions is the same – the normal distribution.

Theorem 2.1. *(Cover & Thomas, 2006) Mutual Information Game. Let X, Z be independent, continuous random variables with non-zero support over the entire real line, and satisfying the moment conditions $\mathbb{E}\left\{X\right\} = \mu_x$, $\mathbb{E}\left\{X^2\right\} = \mu_x^2 + \sigma_x^2$ and $\mathbb{E}\left\{Z\right\} = \mu_z$, $\mathbb{E}\left\{Z^2\right\} = \mu_z^2 + \sigma_z^2$. Further let X^*, Z^* be normally distributed random variables satisfying the same moment conditions, respectively. Then the following series of inequalities holds*

$$I\left(X^*; X^* + Z\right) \geq I\left(X^*; X^* + Z^*\right) \geq I\left(X; X + Z^*\right). \tag{1}$$

Proof. Without loss of generality let $\mu_x = 0$ and $\mu_z = 0$. The first inequality hinges on the entropy power inequality. The second inequality hinges on the maximum entropy of the normal distribution given first and second moment constraints. See (Cover & Thomas, 2006) for details. □

This leads to the following minimax formulation of the game

$$\min_{Z} \max_{X} I\left(X; X+Z\right) = \max_{X} \min_{Z} I\left(X; X+Z\right), \tag{2}$$

which implies that any deviation from normality, for X or Z, is suboptimal from that player's perspective.

2.1.2 Relation to Learning

How might this framework relate to the learning setting? First, previous works have shown that adding noise to the inputs (Bishop, 1995) or to the intermediate activations (Srivastava et al., 2014) of neural networks can be an effective form of regularization, leading to better generalization. Moreover, the mutual information game shows that, among encoding distributions, the normal distribution is maximally robust to random perturbations. Taken together these suggest that encoding activations using the normal distribution is the most effective way of using noise as a regularizer, because a greater degree of regularizing noise in the activations can be tolerated for the same level of corruption.

Second, the mutual information game suggests gaining robustness to Gaussian noise is optimal because it is the worst-case noise distribution. This suggests adding Gaussian noise – specifically – to activations during training should have the strongest regularizing effect. Moreover, gaining robustness to noise has previously been demonstrated to imply better generalization (Arora et al., 2018).

2.2 Maximal Representation Capacity and Maximally Compact Representations

The entropy of a random variable is a measure of the number of bits it can encode (Shannon, 1948), and therefore of its representational capacity (Cover & Thomas, 2006). The normal distribution is the maximum entropy distribution for specified mean and variance. This suggests that a unit which encodes features using the normal distribution has maximal representation capacity given a fixed variance budget, and therefore encodes information as compactly as possible. This may then suggest that it is efficient for a unit (and by extension layer) to encode its activations using the normal distribution.

2.3 Maximally Independent Representations

Previous work has explored the beneficial effects of decorrelating features in neural networks (Huang et al., 2018; 2019; Pan et al., 2019). Furthermore, other works have shown that preventing feature co-adaptation is beneficial for training deep neural networks (Hinton et al., 2012).

For any set of random variables, for example representing the pre-activation values of various units in a neural network layer, uncorrelatedness does not imply independence in general. But for random variables whose marginals are normally distributed, then as shown by Appendix Lemma E.1, uncorrelatedness does imply independence when they are furthermore jointly normally distributed.

We use these results to motivate the following argument: encouraging normality in the feature representations of units by using normality normalization, together with uncorrelatedness in these features, would lead to the desirable property of maximal independence; in the setting where increased unit-wise normality also lends itself to increased joint normality.

3 Background: Power Transform

Before introducing normality normalization, we briefly outline the power transform (Yeo & Johnson, 2000), which our proposed normalization layer hinges on.

Consider a random variable H from which a sample $\boldsymbol{h} = \{h_i\}_{i=1}^N$ is obtained. In the context of normalization layers, N represents the number of samples being normalized in a given neural network layer; for example in batch normalization, $N = nHW$ for convolutional layers, where n is the minibatch size, and H, W are respectively the height and width of the activation.

The power transform gaussianizes $\boldsymbol{h}$ by applying the following function for each h_i:

$$\psi(h;\lambda) = \begin{cases} \frac{1}{\lambda}\left((1+h)^{\lambda} - 1\right), & h \geq 0, \lambda \neq 0 \\ \log(1+h), & h \geq 0, \lambda = 0 \\ \frac{-1}{2-\lambda}\left((1-h)^{2-\lambda} - 1\right), & h < 0, \lambda \neq 2 \\ -\log(1-h), & h < 0, \lambda = 2 \end{cases}. \tag{3}$$

The parameter λ is derived using maximum likelihood estimation (MLE), so that the transformed variable is as normally distributed as possible, by minimizing the following negative log-likelihood (NLL)[1]:

$$\mathcal{L}(\boldsymbol{h};\lambda) = \frac{1}{2}\left(\log(2\pi) + 1\right) + \frac{1}{2}\log\left(\hat{\sigma}^2(\lambda)\right) - \frac{(\lambda-1)}{N}\sum_{i=1}^{N}\log(1+h_i), \tag{4}$$

[1]To simplify the presentation, we momentarily defer the cases $\lambda = 0$ and $\lambda = 2$, and outline the NLL for $h \geq 0$ only, as the case for $h < 0$ follows closely by symmetry.

where $\hat{\mu}(\lambda) = \frac{1}{N}\sum_{i=1}^{N}\psi(h_i;\lambda)$ and $\hat{\sigma}^2(\lambda) = \frac{1}{N}\sum_{i=1}^{N}(\psi(h_i;\lambda) - \hat{\mu}(\lambda))^2$.

4 Normality Normalization

To gaussianize a unit's pre-activations $\boldsymbol{h}$, normality normalization estimates $\hat{\lambda}$ using the method we present in Subsection 4.1, applies the power transform given by Equation 3, and adds Gaussian noise with scaling as described in Subsection 4.2. These steps are done between the normalization and affine transformation steps conventionally performed in other normalization layers.

Algorithm 1: Normality Normalization

Input: $\boldsymbol{u} = \{u_i\}_{i=1}^{N}$
Output: $\boldsymbol{v} = \{v_i\}_{i=1}^{N}$
Learnable Parameters: γ, β
Noise Factor: $\xi \geq 0$

Normalization:
$\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} u_i$
$\hat{\sigma}^2 = \frac{1}{N}\sum_{i=1}^{N}(u_i - \hat{\mu})^2$
$h_i = \frac{u_i - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon}}$

Power Transform and Scaled Additive Noise:
$\hat{\lambda} = 1 - \frac{\mathcal{L}'(\boldsymbol{h};\lambda=1)}{\mathcal{L}''(\boldsymbol{h};\lambda=1)}$
$x_i = \psi\left(h_i;\hat{\lambda}\right)$
with gradient tracking disabled:
  $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$
  $s = \frac{1}{N}\sum_{i=1}^{N}|x_i - \bar{x}|$
sample $z_i \sim \mathcal{N}(0,1)$
$y_i = x_i + z_i \cdot \xi \cdot s$

Affine Transform:
$v_i = \gamma \cdot y_i + \beta$

4.1 Estimate of $\hat{\lambda}$

Differentiating Equation 4 w.r.t. λ and setting the resulting expression to 0 does not lead to a closed-form solution for $\hat{\lambda}$, which suggests an iterative method for its estimation; for example gradient descent, or a root-finding algorithm (Brent, 1971). However, motivated by the NLL's convexity in λ (Yeo & Johnson, 2000), we use a quadratic series expansion for its approximation, which we outline in Appendix A.

With the quadratic form of the NLL, we can estimate $\hat{\lambda}$ with one step of the Newton-Raphson method

$$\hat{\lambda} = 1 - \frac{\mathcal{L}'(\boldsymbol{h};\lambda=1)}{\mathcal{L}''(\boldsymbol{h};\lambda=1)}, \tag{5}$$

where the series expansion has been taken around[2] $\lambda_0 = 1$. The expressions for $\mathcal{L}'(\boldsymbol{h};\lambda=1)$ and $\mathcal{L}''(\boldsymbol{h};\lambda=1)$ are outlined in Appendix A.

Appendix B provides empirical evidence substantiating the similarity between the NLL and its second-order series expansion around $\lambda_0 = 1$, and furthermore demonstrates the accuracy of obtaining the estimates $\hat{\lambda}$ using one step of the Newton-Raphson method.

Subsequent to estimating $\hat{\lambda}$, the power transform is applied to each of the pre-activations to obtain $x_i = \psi\left(h_i;\hat{\lambda}\right)$.

We next discuss a few facets of the method.

Justification for the Second Order Method The justification for using the Newton-Raphson method for computing $\hat{\lambda}$ is as follows:

- A first-order gradient-based method would require iterative refinements to its estimates of $\hat{\lambda}$ in order to find the minima, which would significantly affect runtime. In contrast, the Newton-Raphson method is guaranteed to find the minima of the quadratic loss in one step.
- A first-order gradient-based method for computing $\hat{\lambda}$ would require an additional hyperparameter for the step size. Due to the quadratic nature of the loss, the Newton-Raphson method necessarily does not require any such additional hyperparameter.
- The minibatch statistics $\hat{\mu}$ and $\hat{\sigma}^2$ are the MLEs for their respective optimization problems, and are available in closed-form. It is therefore natural to seek a closed-form expression for the MLE of $\hat{\lambda}$, which is facilitated by using the Newton-Raphson method.

[2]The previously deferred cases of $\lambda = 0$ and $\lambda = 2$ are thus inconsequential, in the context of computing an estimate $\hat{\lambda}$, by continuity of the quadratic form of the series expansion for the NLL. However, these two cases still need to be considered when applying the transformation function itself.

Location of Series Expansion The choice of taking the series expansion around $\lambda_0 = 1$ is justified using the following two complementary factors:

- $\hat{\lambda} = 1$ corresponds to the identity transformation, and hence having $\lambda_0 = 1$ as the point where the series expansion is taken, facilitates its recovery if this is optimal.
- It equivocates to assuming the least about the nature of the deviations from normality in the sample statistics, since it avoids biasing the form of the series expansion for the loss towards solutions favoring $\hat{\lambda} < 1$ or $\hat{\lambda} > 1$.

Order of Normalization and Power Transform Steps Applying the power transform after the normalization step is beneficial, because having zero mean and unit variance activations simplifies several terms in the computation of $\hat{\lambda}$, as shown in Appendix A, and improves numerical stability.

No Additional Learned Parameters Despite having increased normality in the features, this came at no additional cost in terms of the number of learnable parameters relative to existing normalization techniques.

Test Time In the case where normality normalization is used to augment batch normalization, in addition to computing global estimates for μ and σ^2, we additionally compute a global estimate for λ. These are obtained using the respective training set running averages for these terms, analogously with batch normalization. At test time, these global estimates μ, σ^2, λ are used, rather than the test minibatch statistics themselves.

4.2 Additive Gaussian noise with Scaling

Normality normalization applies additive random noise to the output of the power transform; a step which is motivated using information-theoretic principles in Subsection 2.1.

For each input indexed by $i \in \{1, \dots, N\}$, during training[3] we have $y_i = x_i + z_i \cdot \xi \cdot s$, where x_i is the i-th input's post-power transform value, $z_i \sim \mathcal{N}(0, 1)$, $\xi \geq 0$ is the noise factor, and $s = \frac{1}{N} \left\| \boldsymbol{x} - \bar{\boldsymbol{x}} \right\|_1$ represents the zero-centered norm of the post-power transform values, normalized by the sample size N.

Importantly, scaling each of the sampled noise values z_i for a given channel's minibatch[4] by the channel-specific scaling factor s leads to an appropriate degree of additive noise for each of the channel's constituent terms x_i. This is significant because for a given minibatch, each channel's norm will differ from the norms of other channels.

Furthermore, we treat s as a constant, so that its constituent terms are not incorporated during backpropagation.[5] This is significant because the purpose of s is to scale the additive random noise by the minibatch's statistics, and not for it to contribute to learning directly by affecting the gradients of the constituent terms.

Note that we employ the ℓ_1-norm for $\boldsymbol{x}$ rather than the ℓ_2-norm because it lends itself to a more robust measure of dispersion (Pham-Gia & Hung, 2001).

Algorithm 1 provides a summary of normality normalization.

5 Experimental Results & Analysis

5.1 Experimental Setup

For each model and dataset combination results presented, $M = 6$ models were trained, each with differing random initializations for the model parameters. No data augmentations were employed in

[3]We do not apply additive random noise with scaling at test time.

[4]For clarity the present discussion assumes the case where normality normalization is used to augment batch normalization. However, the discussion applies equally to other normalization layers, such as layer, instance, and group normalization.

[5]Implementationally, this is done by disabling gradient tracking when computing these terms.

the experiments. Wherever a result is reported numerically, it is obtained using the mean performance and one standard error from the mean across the M runs. The best performing models for a given dataset and model combination are shown in bold. Wherever a result is shown graphically, unless otherwise stated it is displayed using the mean performance and its 95% confidence interval across the M runs. The training configurations of the models are outlined in Appendix C. Code is made available in the Supplementary Materials.

5.2 Generalization Performance

We evaluate batch normality normalization (BatchNormalNorm) and batch normalization (BatchNorm) on a variety of models and datasets, as shown in Table 1. A similar evaluation is done for layer normality normalization (LayerNormalNorm) and layer normalization (LayerNorm), shown in Table 2.

Normality Normalization is Performant BatchNormalNorm generally outperforms BatchNorm across multiple architectures and datasets, with a similar trend holding between LayerNormalNorm and LayerNorm.

Table 1: Validation accuracy for several ResNet (RN) architecture and dataset combinations, when using BatchNormalNorm (BNN) vs. BatchNorm (BN). For each table entry, representing a dataset and model combination, $M = 6$ models were trained, each with differing random initializations for the model parameters. No data augmentations were employed during training.

Dataset	Model	BN	BNN
CIFAR10	RN18	88.89 ± 0.07	$\mathbf{90.41 \pm 0.09}$
CIFAR100	RN18	62.02 ± 0.17	$\mathbf{65.82 \pm 0.11}$
STL10	RN34	58.82 ± 0.52	$\mathbf{63.86 \pm 0.45}$
TinyImageNet Top1	RN34	58.22 ± 0.12	$\mathbf{60.57 \pm 0.14}$
TinyImageNet Top5	RN34	81.74 ± 0.16	$\mathbf{83.31 \pm 0.13}$
Caltech101	RN50	72.60 ± 0.35	$\mathbf{74.71 \pm 0.51}$
Food101	RN50	61.15 ± 0.44	$\mathbf{63.51 \pm 0.33}$

Table 2: Validation accuracy across several benchmarks for a vision transformer (ViT) architecture (see training details for model specifications), when using LayerNormalNorm (LNN) vs. LayerNorm (LN). For each table entry, representing a dataset and model combination, $M = 6$ models were trained, each with differing random initializations for the model parameters. No data augmentations were employed during training (see the Appendix for experiments using data augmentations).

Dataset	LN	LNN
SVHN	88.46 ± 0.10	$\mathbf{89.96 \pm 0.12}$
CIFAR10	66.56 ± 0.13	$\mathbf{70.55 \pm 0.23}$
CIFAR100	37.98 ± 0.45	$\mathbf{44.60 \pm 0.36}$
Food101	38.63 ± 0.53	$\mathbf{48.92 \pm 0.32}$
ImageNet100 Top1	50.78 ± 0.33	$\mathbf{62.39 \pm 0.68}$
ImageNet100 Top5	75.45 ± 0.50	$\mathbf{84.03 \pm 0.42}$

5.3 Effectiveness Across Normalization Layers

Figure 1 demonstrates the general effectiveness of normality normalization across various normalization layer types. Here we further extended group normalization (GroupNorm) to group normality normalization (GroupNormalNorm) and instance normalization (InstanceNorm) to instance normality normalization (InstanceNormalNorm). This provides further evidence that normality normalization can be employed wherever normalization layers are conventionally used.

5.4 Effectiveness Across Model Configurations

Network Width Figure 2 shows that BatchNormalNorm outperforms BatchNorm across varying WideResNet architecture model widths. Of particular note is that BatchNormalNorm shows strong performance even in the regime of relatively small network widths, whereas BatchNorm's performance deteriorates. This may indicate that for small-width networks, which do not exhibit the Gaussian process limiting approximation attributed to large-width networks (Neal, 1996; Lee et al., 2017; Jacot et al., 2018; Lee et al., 2019), normality normalization provides a correcting effect. This could, for example, be beneficial for hardware-limited deep learning applications.

Network Depth Figure 3 shows that BatchNormalNorm outperforms BatchNorm across varying model depths. This suggests normality normalization is beneficial both for small and large-depth models. Furthermore, the increased benefit to performance for BatchNormalNorm in deeper networks suggests normality normalization may correct for an increased tendency towards non-normality as a function of model depth.

Training Minibatch Size Figure 4 shows that BatchNormalNorm maintains a high level of performance across minibatch sizes used during training, which provides further evidence for normality normalization's general effectiveness across a variety of configurations.

5.5 Normality of Representations

Figure 5 shows representative Q–Q plots (Wilk & Gnanadesikan, 1968), a method for assessing normality, together with an aggregate measure of normality across model layers, for post-power transform feature values when using BatchNormalNorm, and post-normalization values when using BatchNorm. The figure corresponds to models which have been trained to convergence. It demonstrates the greater normality obtained when using normality normalization.

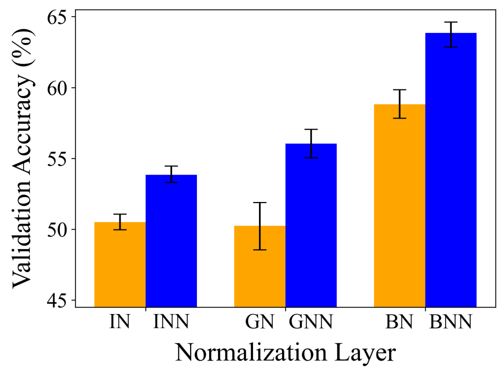

Figure 1: **Normality normalization is effective for various normalization layers.** Validation accuracy for ResNet34 architectures evaluated on the STL10 dataset. Each bar represents the performance of the ResNet34 architecture, when using the given normalization layer across the entire network. INN: InstanceNormalNorm, IN: InstanceNorm, GNN: GroupNormalNorm, GN: GroupNorm, BNN: BatchNormalNorm, BN: BatchNorm.

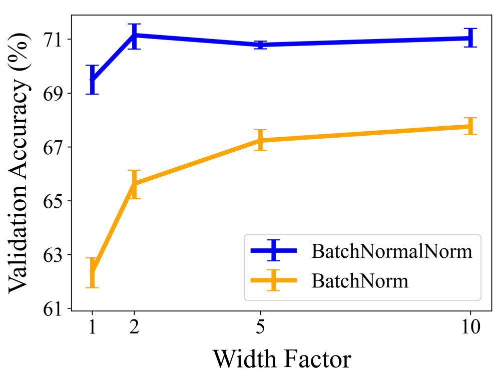

Figure 2: **Normality normalization is effective for small and large width networks.** Validation accuracy on the STL-10 dataset for WideResNet architectures with varying width factors when controlling for depth of 28, when using BatchNormalNorm vs. BatchNorm.

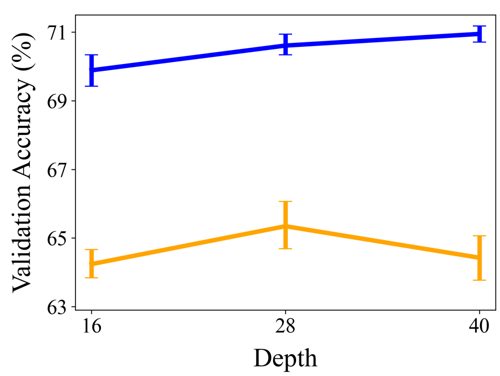

Figure 3: **Normality normalization is effective for networks of various depths.** Validation accuracy on the STL10 dataset for WideResNet architectures with varying depths when controlling for a width factor of 2, when using BatchNormalNorm vs. BatchNorm.

5.6 Noise Robustness

We use the following framework to measure a model's robustness to noise (a similar setting is used by Arora et al. (2018)). For a given data point, consider a pair of units in a neural network, the first in the k-th layer and the second in the ℓ-th layer. For the unit in the k-th layer, let x denote the data point's post-normalization value. Let $\phi_{k,\ell}(x)$ be the same data point's post-normalization value for the unit in the subsequent layer ℓ, where the function $\phi_{k,\ell}$ encapsulates all the intermediate computations between the two normalization layers k and ℓ.

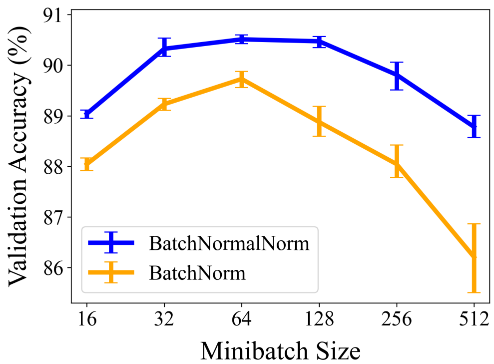

Figure 4: **Normality normalization is effective across minibatch sizes used during training.** Validation accuracy for ResNet18 architectures evaluated on the CIFAR10 dataset, with varying minibatch sizes used during training, when using BatchNormalNorm vs. BatchNorm.

Let $y = x + z \cdot \delta \cdot \frac{1}{N} \left\| \boldsymbol{x} - \bar{\boldsymbol{x}} \right\|_1$, where as in Subsection 4.2 $z \sim \mathcal{N}(0, 1)$, $\delta \geq 0$, and here $\left\| \boldsymbol{x} - \bar{\boldsymbol{x}} \right\|_1$ represents a global estimate for the zero-centered norm of the post-normalized values, derived from the training set in its entirety. We then define noise robustness as follows:

Definition 5.1 (Noise Robustness)**.** For given realization of the noise sample z, let $\zeta_{k,\ell}^{\delta}(x, y)$-robustness be defined as:

$$\zeta_{k,\ell}^{\delta}(x, y) := \frac{\left\| \phi_{k,\ell}(x) - \phi_{k,\ell}(y) \right\|_1}{\left\| \phi_{k,\ell}(x) \right\|_1}. \quad (6)$$

Thus $\zeta_{k,\ell}^{\delta}(x, y)$ measures the relative discrepancy between $\phi_{k,\ell}(x)$ and $\phi_{k,\ell}(y)$ when noise factor δ is used, and effectively represents the noise's attenuation from layer k to layer ℓ. Averaging $\zeta_{k,\ell}^{\delta}(x, y)$ over all data points, and over all units in the k-th and ℓ-th layers, leads to a consolidated estimate of the noise robustness.

Table 3 demonstrates the increased robustness to noise obtained when using BatchNormalNorm in comparison to BatchNorm. This substantiates the applicability of the noise robustness framework presented in Subsection 2.1, and consequently of the benefit of gaussianizing learned representations in normality normalization.

6 Related Work & Future Directions

Power Transforms Various power transforms have been developed (Box & Cox, 1964; John & Draper, 1980; Yeo & Johnson, 2000) for increasing normality in data. Box & Cox (1964) defined a power transform which is convex in its parameter, but is only defined for positive variables. Yeo & Johnson (2000) presented an alternative power transform which was furthermore defined for the

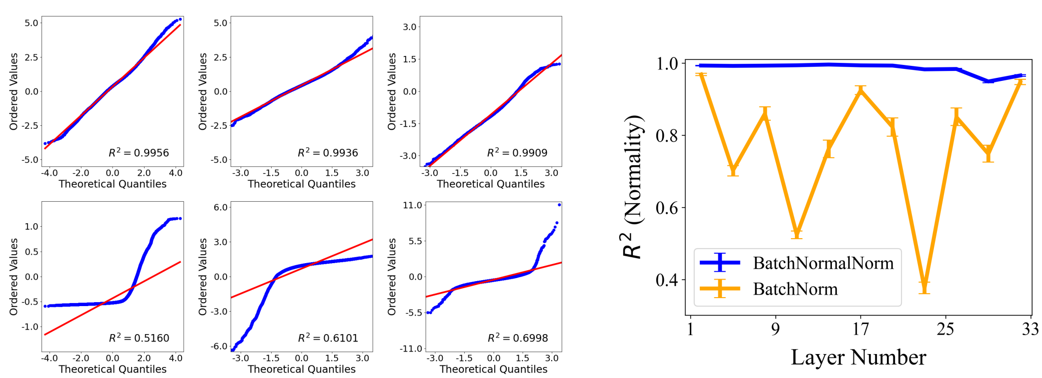

Figure 5: Left: Representative QQ-plots of feature values for models trained to convergence with BatchNormalNorm (post-power transform) (top row) vs. BatchNorm (post-normalization) (bottom row), measured for the same validation minibatch (ResNet34/STL10). Left to right: increasing layer number. The x-axis represents the theoretical quantiles of the normal distribution, and the y-axis the sample's ordered values. A higher R^2 value for the line of best fit signifies greater normality in the features. BatchNormalNorm induces greater normality in the features throughout the model, in comparison to BatchNorm.
Right: A plot showing the average R^2 values for both normalization layers across model depth, taken across QQ-plots corresponding to 20 channels and 10 validation minibatches. The plot demonstrates that normality normalization leads to higher normality across the model layers.

Table 3: **Normality normalization is robust to noise at test time.** Evaluation of robustness to noise, using the relative error $\zeta^{\delta}_{k,\ell}$ for various layers k and ℓ, for various models trained with BatchNormalNorm (BNN) and BatchNorm (BN). Models were evaluated using noise factor $\delta = 0.5$. Top: ResNet18/CIFAR100, bottom: ResNet34/STL10. The layer k at which noise is added is denoted on the left side of each row, and each column denotes a subsequent layer ℓ. For each entry, $\zeta^{\delta}_{k,\ell}$ was averaged over the entire validation set, and over all channels in the k-th and ℓ-th layers. This was subsequently averaged across $T = 6$ Monte Carlo draws for the random noise, and the values presented are furthermore the average across each of the $M = 6$ trained models. In each table entry, the top value represents the relative error for BatchNormalNorm, and the bottom value for BatchNorm, with the best value shown in bold. Lower is better. The tables provide evidence that models trained using normality normalization are generally more robust to random noise at test time.

		L5	L9	L13	L17
L1	BNN	**0.051 ± 0.001**	**0.076 ± 0.001**	**0.100 ± 0.001**	**0.387 ± 0.004**
	BN	0.177 ± 0.007	0.333 ± 0.011	0.419 ± 0.021	1.922 ± 0.076
L5			**0.027 ± 0.002**	**0.038 ± 0.003**	**0.149 ± 0.012**
			0.059 ± 0.009	0.073 ± 0.009	0.373 ± 0.041
L9				**0.044 ± 0.001**	**0.151 ± 0.002**
				0.063 ± 0.003	0.249 ± 0.009
L13					**0.257 ± 0.001**
					0.367 ± 0.020

		L9	L17	L25	L33
L1	BNN	**0.373 ± 0.018**	**0.709 ± 0.034**	**0.452 ± 0.044**	**0.565 ± 0.053**
	BN	0.615 ± 0.046	1.280 ± 0.080	1.900 ± 0.537	2.621 ± 0.257
L9			**0.141 ± 0.004**	**0.080 ± 0.003**	**0.099 ± 0.007**
			0.120 ± 0.005	0.099 ± 0.011	0.307 ± 0.013
L17				**0.102 ± 0.006**	**0.121 ± 0.011**
				0.104 ± 0.006	0.324 ± 0.012
L25					**0.051 ± 0.006**
					0.120 ± 0.006

entire real line, preserved the convexity property with respect to its parameter (concavity for negative input values), and additionally addressed skewed input distributions.

It is worth noting that many power transforms were developed with the aim of improving the validity of statistical tests relying on the assumption of normality in the data. This is in contrast with the present work, which uses an information-theoretic motivation for gaussianizing.

Gaussianization Non-parametric techniques, for example those using quantile functions (Gilchrist, 2000), offer an alternative approach to gaussianizing but are not easily amenable to the deep learning setting where models are trained using backpropagation and gradient descent. Employing iterative gaussianization techniques (Chen & Gopinath, 2000; Laparra et al., 2011) offers an interesting direction for future work.

Normalization Layers The properties of normality normalization explored in the present work can be utilized to extend normalization layers studied in the context of specific architectures (Shen et al., 2020), and those tailored for general manifolds (Brooks et al., 2019; Chen et al., 2024). Furthermore, as a result of normality normalization's gaussianizing effect, the analysis of works which have sought to better understand the effects of existing normalization layers, and to motivative new ones, may be facilitated (Bjorck et al., 2018; Santurkar et al., 2018; Hoffer et al., 2018; Yang et al., 2019; Luo et al., 2019; Xu et al., 2019; Daneshmand et al., 2020; 2021; Joudaki et al., 2023).

Adversarial Robustness It would be interesting to tie the present work with those suggesting robustness to ℓ_2-norm constrained adversarial perturbations increases when training with Gaussian noise (Cohen et al., 2019; Salman et al., 2019). Furthermore, it has been suggested that adversarial examples and images corrupted with Gaussian noise may be related (Ford et al., 2019). This might indicate gaining robustness to Gaussian noise not only in the inputs, but throughout the model, can lead to greater adversarial robustness.

However, gaussianizing activations, and training with Gaussian noise, may only be a defense in the distributional sense; exact knowledge of the weights (and consequently of the activation values), as is often assumed in the adversarial robustness setting, is not captured by the noise-based robustness framework, which is only concerned with distributional assumptions over the activation values. Nevertheless it does suggest that, on average, greater robustness may be attainable.

Neural Networks as Gaussian Processes Neal (1996) showed that in the limit of infinite width, a single layer neural network at initialization approximates a Gaussian process. This result has been extended to the multi-layer setting by (Lee et al., 2017), and Jacot et al. (2018); Lee et al. (2019) suggest the Gaussian process approximation may remain valid beyond network initialization. However, these analyses still necessitate the infinite width limit assumption.

Recent work has shown that batch normalization lends itself to a non-asymptotic approximation to normality throughout the layers of neural networks at initialization (Daneshmand et al., 2021). Given its gaussianizing effect, layers trained with normality normalization may be amenable to a non-asymptotic approximation to Gaussian processes – throughout training. This could help to further address the disparity in the analysis of neural networks in the infinite width limit, for example as in mean-field theory, with the finite width setting (Joudaki et al., 2023).

7 Conclusion

Among the methodological developments that have spurred the advent of deep learning, their success has often been attributed to their effect on the model's ability to learn and encode representations effectively, whether in the activations or in the weights. This can be seen, for example, by considering the importance of initializing model weights suitably, or by the effect different activation functions have on learning dynamics.

Seldom has a prescription for precisely what distribution a deep learning model should use to effectively encode its activations, and exactly how this can be achieved, been investigated. The present work addresses this – first by motivating the normal distribution as the probability distribution of choice, and subsequently by materializing this choice through normality normalization.

It is perhaps nowhere clearer what representational benefit normality normalization provides, than when considering that no additional learnable parameters, relative to existing normalization layers, were introduced. This highlights – and precisely controls for the effect of – the importance of encouraging models to encode their representations effectively.

We presented normality normalization: a novel, principledly motivated, normalization layer. Our experiments and analysis comprehensively demonstrated the effectiveness of normality normalization, in regards to its generalization performance on an array of widely used model and dataset combinations, its consistently strong performance across various common factors of variation such as model width, depth, and training minibatch size, its suitability for usage wherever existing normalization layers are conventionally used, and through its effect on improving model robustness to random perturbations.

Reproducibility Statement Comprehensive training details are provided in Subsection 5.1, and Appendix Subsection C.1 and Subsection C.2. Moreover, we provide the details needed to reproduce our experiments throughout Section 5. Subsection 5.1 provides details regarding the number of training runs employed when reporting a result (numerical or graphical), how the results were aggregated, and how the error bars were obtained. When reporting error bars, what the randomness is with respect to (ex: random initialization for model parameters) is clearly outlined. The codebase in its completeness, with accompanying instructions, are provided in the Supplementary Materials – these precisely reproduce and provide full coverage for our experimental setup. Additionally, the program's command-line options are clearly described. All dataset access instructions, and preparatory & preprocessing steps for the datasets, are provided in full. In Appendix Subsection C.3 we comprehensively cite all models, datasets, and machine learning related frameworks we used. We specify the licenses and terms of use of these items, and our work respects their terms. Appendix Subsection C.4 provides details on the compute resources used for the experiments.

REFERENCES

Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. In *International Conference on Machine Learning*, 2018.

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Chris M. Bishop. Training with noise is equivalent to tikhonov regularization. *Neural Computation*, 7(1):108–116, Jan 1995. ISSN 0899-7667. doi: 10.1162/neco.1995.7.1.108.

Nils Bjorck, Carla P Gomes, Bart Selman, and Kilian Q Weinberger. Understanding batch normalization. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.

G. E. P. Box and D. R. Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, 26(2):211–252, 1964. ISSN 00359246.

Richard P. Brent. An algorithm with guaranteed convergence for finding a zero of a function. *Comput. J.*, 14:422–425, 1971.

Daniel Brooks, Olivier Schwander, Frederic Barbaresco, Jean-Yves Schneider, and Matthieu Cord. Riemannian batch normalization for spd neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Scott Chen and Ramesh Gopinath. Gaussianization. In T. Leen, T. Dietterich, and V. Tresp (eds.), *Advances in Neural Information Processing Systems*, volume 13. MIT Press, 2000.

Ziheng Chen, Yue Song, Yunmei Liu, and Nicu Sebe. A lie group approach to riemannian batch normalization. In *The Twelfth International Conference on Learning Representations*, 2024.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In Geoffrey Gordon, David Dunson, and Miroslav Dudík (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pp. 215–223, Fort Lauderdale, FL, USA, 11–13 Apr 2011. PMLR.

Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. Certified adversarial robustness via randomized smoothing. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 1310–1320. PMLR, 09–15 Jun 2019.

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN 0471241954.

Hadi Daneshmand, Jonas Kohler, Francis Bach, Thomas Hofmann, and Aurelien Lucchi. Batch normalization provably avoids ranks collapse for randomly initialised deep networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 18387–18398. Curran Associates, Inc., 2020.

Hadi Daneshmand, Amir Joudaki, and Francis Bach. Batch normalization orthogonalizes representations in deep random networks. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 4896–4906. Curran Associates, Inc., 2021.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

Bruno Ebner and Norbert Henze. Tests for multivariate normality—a critical review with emphasis on weighted l^2-statistics. *TEST*, 29(4):845–892, 2020. ISSN 1133-0686, 1863-8260. doi: 10.1007/s11749-020-00740-0.

Nic Ford, Justin Gilmer, Nicholas Carlini, and Ekin Dogus Cubuk. Adversarial examples are a natural consequence of test error in noise. In *International Conference on Machine Learning*, 2019.

W.G. Gilchrist. *Statistical modelling with quantile functions*. 01 2000.

Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 1321–1330. PMLR, 06–11 Aug 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

N. Henze and B. Zirkler. A class of invariant consistent tests for multivariate normality. *Communications in Statistics - Theory and Methods*, 19(10):3595–3617, 1990. doi: 10.1080/03610929008830400.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *ArXiv*, abs/1207.0580, 2012.

Elad Hoffer, Ron Banner, Itay Golan, and Daniel Soudry. Norm matters: efficient and accurate normalization schemes in deep networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

Lei Huang, Dawei Yang, Bo Lang, and Jia Deng. Decorrelated batch normalization. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 791–800, 2018.

Lei Huang, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Iterative normalization: Beyond standardization towards efficient whitening. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4869–4878, 2019.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Francis Bach and David Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 448–456, Lille, France, 07–09 Jul 2015. PMLR.

Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

J. A. John and N. R. Draper. An alternative family of transformations. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 29(2):190–197, 1980. ISSN 00359254, 14679876.

Amir Joudaki, Hadi Daneshmand, and Francis Bach. On bridging the gap between mean field and finite width deep random multilayer perceptron with batch normalization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 15388–15400. PMLR, 23–29 Jul 2023.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diega, CA, USA, 2015.

Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009.

Valero Laparra, Gustavo Camps-Valls, and Jesús Malo. Iterative gaussianization: From ica to random rotations. *IEEE Transactions on Neural Networks*, 22(4):537–549, 2011. doi: 10.1109/TNN.2011.2106511.

Ya Le and Xuan Yang. Tiny imagenet visual recognition challenge. In *CS 231N*, 2015.

Jaehoon Lee, Yasaman Bahri, Roman Novak, Samuel S. Schoenholz, Jeffrey Pennington, and Jascha Narain Sohl-Dickstein. Deep neural networks as gaussian processes. *ArXiv*, abs/1711.00165, 2017.

Jaehoon Lee, Lechao Xiao, Samuel Schoenholz, Yasaman Bahri, Roman Novak, Jascha Sohl-Dickstein, and Jeffrey Pennington. Wide neural networks of any depth evolve as linear models under gradient descent. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Fei-Fei Li, Marco Andreeto, Marc'Aurelio Ranzato, and Pietro Perona. Caltech 101, Apr 2022.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Ping Luo, Xinjiang Wang, Wenqi Shao, and Zhanglin Peng. Towards understanding regularization in batch normalization. In *International Conference on Learning Representations*, 2019.

Radford M. Neal. *Bayesian Learning for Neural Networks*. Springer-Verlag, Berlin, Heidelberg, 1996. ISBN 0387947248.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

Xingang Pan, Xiaohang Zhan, Jianping Shi, Xiaoou Tang, and Ping Luo. Switchable whitening for deep representation learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martín Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

T. Pham-Gia and T.L. Hung. The mean and median absolute deviations. *Mathematical and Computer Modelling*, 34(7):921–936, 2001. ISSN 0895-7177. doi: https://doi.org/10.1016/S0895-7177(01)00109-1.

Hadi Salman, Jerry Li, Ilya Razenshteyn, Pengchuan Zhang, Huan Zhang, Sebastien Bubeck, and Greg Yang. Provably robust deep learning via adversarially trained smoothed classifiers. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3): 379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Sheng Shen, Zhewei Yao, Amir Gholami, Michael Mahoney, and Kurt Keutzer. PowerNorm: Rethinking batch normalization in transformers. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8741–8751. PMLR, 13–18 Jul 2020.

Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15: 1929–1958, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In Yoshua Bengio and Yann LeCun (eds.), *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *ArXiv*, abs/1607.08022, 2016.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(95):2837–2854, 2010.

Sida Wang and Christopher Manning. Fast dropout training. In Sanjoy Dasgupta and David McAllester (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 118–126, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

M. B. Wilk and R. Gnanadesikan. Probability plotting methods for the analysis of data. *Biometrika*, 55(1):1–17, 1968. ISSN 00063444, 14643510.

Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

Jingjing Xu, Xu Sun, Zhiyuan Zhang, Guangxiang Zhao, and Junyang Lin. Understanding and improving layer normalization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Greg Yang, Jeffrey Pennington, Vinay Rao, Jascha Sohl-Dickstein, and Samuel S. Schoenholz. A mean field theory of batch normalization. In *International Conference on Learning Representations*, 2019.

In-Kwon Yeo and Richard A. Johnson. A new family of power transformations to improve normality or symmetry. *Biometrika*, 87(4):954–959, 2000. ISSN 00063444.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Ming Zhou and Yongzhao Shao. A powerful test for multivariate normality. *Journal of Applied Statistics*, 41(2):351–363, 2014. doi: 10.1080/02664763.2013.839637. PMID: 24563571.

A Series Expansion of the Power Transform Loss

Let $\mathcal{L}_2\left(\boldsymbol{x};(\lambda,\lambda_0=1)\right)$ denote the second-order series expansion of the power transform's NLL centered at $\lambda_0=1$, i.e.

$$\mathcal{L}_2\left(\boldsymbol{x};(\lambda,\lambda_0=1)\right)=\mathcal{L}\left(\boldsymbol{x};\lambda=1\right)+\left(\lambda-1\right)\mathcal{L}'(\boldsymbol{x};\lambda=1)+\frac{\left(\lambda-1\right)^2}{2}\mathcal{L}''(\boldsymbol{x};\lambda=1)\,. \tag{7}$$

We have[6]

$$\begin{aligned}
\mathcal{L}\left(\boldsymbol{x};\lambda=1\right)&=\mathcal{L}\left(\boldsymbol{x};\lambda\right)\Big|_{\lambda=1}=\frac{1}{2}\log\left(2\pi+1\right)+\frac{1}{2}\log\left(\hat{\sigma}^2\left(\lambda=1\right)\right),\\
\mathcal{L}'(\boldsymbol{x};\lambda=1)&=\frac{\partial\mathcal{L}\left(\boldsymbol{x};\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}=\frac{1}{2\hat{\sigma}^2\left(\lambda=1\right)}\frac{\partial\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}-\frac{1}{N}\sum_{i=1}^{N}\log\left(1+x_i\right),\\
\mathcal{L}''(\boldsymbol{x};\lambda=1)&=\frac{\partial^2\mathcal{L}\left(\boldsymbol{x};\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}\\
&=\frac{-1}{2\left(\hat{\sigma}^2\left(\lambda=1\right)\right)^2}\left(\frac{\partial\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}\right)^2+\frac{1}{2\hat{\sigma}^2\left(\lambda=1\right)}\frac{\partial^2\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1},
\end{aligned}\tag{8}$$

where

$$\frac{\partial\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda}=\frac{2}{N}\sum_{i=1}^{N}\left[\left(\psi\left(x_i;\lambda\right)-\hat{\mu}\left(\lambda\right)\right)\left(\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}-\frac{\partial\hat{\mu}\left(\lambda\right)}{\partial\lambda}\right)\right], \tag{9}$$

$$\therefore\frac{\partial\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}=\frac{2}{N}\sum_{i=1}^{N}\left[\left(x_i-\hat{\mu}\left(\lambda=1\right)\right)\left(\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}-\frac{\partial\hat{\mu}\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}\right)\right], \tag{10}$$

with

$$\begin{aligned}
\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}&=\left(1+x_i\right)\left(\log\left(1+x_i\right)\right)-x_i,\\
\frac{\partial\hat{\mu}\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}&=\frac{1}{N}\sum_{i=1}^{N}\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}\Big|_{\lambda=1},
\end{aligned}\tag{11}$$

and

$$\begin{aligned}
\frac{\partial^2\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda^2}=\frac{2}{N}\sum_{i=1}^{N}\Bigg[&\left(\left(\psi\left(x_i;\lambda\right)-\hat{\mu}\left(\lambda\right)\right)\left(\frac{\partial^2\psi\left(x_i;\lambda\right)}{\partial\lambda^2}-\frac{\partial^2\hat{\mu}\left(\lambda\right)}{\partial\lambda^2}\right)\right)\\
&+\left(\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}-\frac{\partial\hat{\mu}\left(\lambda\right)}{\partial\lambda}\right)^2\Bigg],
\end{aligned}\tag{12}$$

$$\begin{aligned}
\therefore\frac{\partial^2\hat{\sigma}^2\left(\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}=\frac{2}{N}\sum_{i=1}^{N}\Bigg[&\left(\left(x_i-\hat{\mu}\left(\lambda=1\right)\right)\left(\frac{\partial^2\psi\left(x_i;\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}-\frac{\partial^2\hat{\mu}\left(\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}\right)\right)\\
&+\left(\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}-\frac{\partial\hat{\mu}\left(\lambda\right)}{\partial\lambda}\Big|_{\lambda=1}\right)^2\Bigg],
\end{aligned}\tag{13}$$

with

$$\begin{aligned}
\frac{\partial^2\psi\left(x_i;\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}&=\left(1+x_i\right)\left(\log\left(1+x_i\right)\right)^2-2\frac{\partial\psi\left(x_i;\lambda\right)}{\partial\lambda}\Big|_{\lambda=1},\\
\frac{\partial^2\hat{\mu}\left(\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}&=\frac{1}{N}\sum_{i=1}^{N}\frac{\partial^2\psi\left(x_i;\lambda\right)}{\partial\lambda^2}\Big|_{\lambda=1}.
\end{aligned}\tag{14}$$

Furthermore, because the power transform is applied after the normalization step (see main text), $\hat{\mu}\left(\lambda=1\right)=0$ and $\hat{\sigma}^2\left(\lambda=1\right)=1$.

[6] To simplify the presentation, we outline the series expansion for $x\geq 0$ only, as $x<0$ follows closely by symmetry.

B EVALUATION OF $\hat{\lambda}$ ESTIMATES

Figure 6 provides representative examples substantiating the similarity between the NLL and its second-order series expansion around $\lambda_0 = 1$. The figure furthermore demonstrates the accuracy of obtaining the estimates $\hat{\lambda}$ using one step of the Newton-Raphson method.

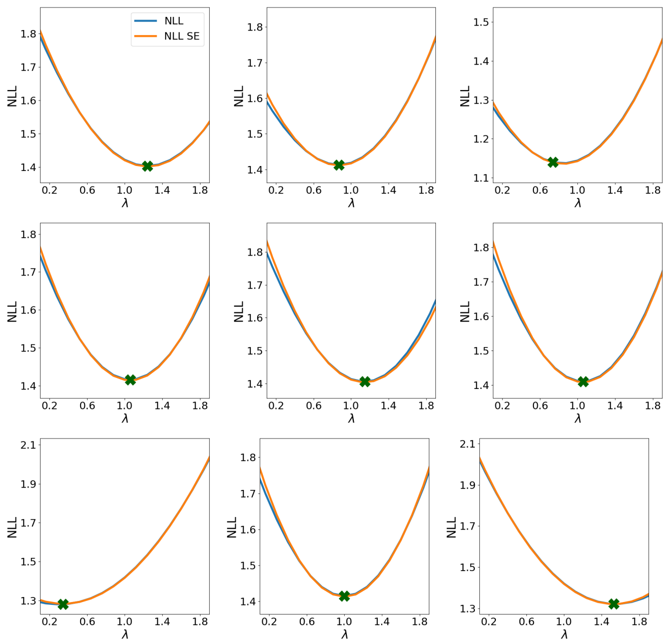

Figure 6: Normality normalization estimates for $\hat{\lambda}$ for a given training minibatch (ResNet18/CIFAR10). Left to right: increasing layer number. Top to bottom: estimates from various channels. Normality normalization's quadratic series expansion for the loss (NLL SE) closely approximates the original loss (NLL), leading to accurate estimates of $\hat{\lambda}$ (marked by $\times$).

C TRAINING DETAILS

C.1 RESNET AND WIDERESNET EXPERIMENTS

The training configuration of the model and dataset combinations shown in Table 1, Figure 2, Figure 3, and Figure 4, which use batch normality normalization (BatchNormalNorm/BNN) and batch normalization (BatchNorm/BN), and Figure 1, which use instance normality normalization (InstanceNormalNorm/INN), instance normalization (InstanceNorm/IN), group normality normalization (GroupNormalNorm/GNN), and group normalization (GroupNorm/GN), are as follows.

We used a variety of residual network (ResNet) (He et al., 2016) and wide residual network (WideResNet) (Zagoruyko & Komodakis, 2016) architectures in our experiments. For all experiments except those using the TinyImageNet, Caltech101, Food101, and ImageNet datasets, models were trained from random initialization for 200 epochs, with a factor of 10 reduction in learning rate at each 60-epoch interval. For the experiments using the TinyImageNet, Caltech101, and Food101 datasets, models were trained from random initialization for 100 epochs, with a factor of 10 reduction in learning rate at epochs $40, 70, 90$. For the experiments using the ImageNet dataset, models were trained from random initialization for 120 epochs, with a factor of 10 reduction in learning rate at epoch 90. A group size of 32 was used in all of the relevant group normalization experiments. For the Caltech101 dataset, each run used a random 90/10% allocation to obtain the training and validation splits respectively[7]. Each such run used its own unique random seed to generate the splits for that run, which facilitates greater precision in the reporting of our aggregate results across the runs.

In all of our experiments involving the ResNet18, ResNet34, and WideResNet architectures, stochastic gradient descent (SGD) with learning rate 0.1, weight decay 5×10^{-4}, momentum 0.9, and minibatch size 128 was used. In the experiments involving the ResNet50 architecture on the TinyImageNet, Caltech101, and Food101 datasets, SGD with learning rate 0.0125, weight decay 1×10^{-4}, momentum 0.9, and minibatch size 32 was used. In the experiments involving the ResNet50 architecture on the ImageNet dataset, SGD with learning rate 0.1, weight decay 1×10^{-4}, momentum 0.9, and minibatch size 256 was used when training with BNN, and SGD with learning rate 0.05, weight decay 1×10^{-4}, momentum 0.9, and minibatch size 128 was used when training with BN. A noise factor of $\xi = 0.4$, was used, as preliminary experiments demonstrated increases typically resulted in training instability. We also investigated several hyperparameter configurations, including for the learning rate, learning rate scheduler, weight decay, and minibatch size, across all the models and found the present configurations to generally work best across all of them.

C.2 Vision Transformer Experiments

The training configuration of the model and dataset combinations shown in Table 2, which use layer normality normalization (LayerNormalNorm/LNN) and layer normalization (LayerNorm/LN), are as follows. We used a vision transformer (Vaswani et al., 2017; Dosovitskiy et al., 2021) model in our experiments consisting of 8 transformer layers, 8 attention heads, hidden dimension size of 768, and MLP dimension size of 2304. A patch size of 4 was used throughout, except for the Food101 and ImageNet100 experiments where it was set to 16.

For all experiments except those using the Food101 and ImageNet100 datasets, models were trained from random initialization for 200 epochs, with a factor of 10 reduction in learning rate at each 60-epoch interval. For the experiments using the Food101 and ImageNet100 datasets, models were trained from random initialization for 100 epochs, with a factor of 10 reduction in learning rate at epochs $40, 70, 90$. For the ImageNet100 dataset, for each run we randomly sampled 100 classes from the ImageNet dataset, and used all the data corresponding to these 100 classes in their respective training and validation sets. Each such run used its own unique random seed to sample the 100 classes for that run, which facilitates greater precision in the reporting of our aggregate results across the runs. For each ImageNet100 training run, we applied weighted random sampling to sample training examples based on the training set's corresponding inverse class frequency for the data point; we found this to help across all the model configurations used.

The AdamW optimizer (Kingma & Ba, 2015; Loshchilov & Hutter, 2019) with learning rate 1×10^{-3}, weight decay 5×10^{-2}, $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 1 \times 10^{-8}$, and minibatch size 32 was used. A noise factor of $\xi = 1.0$, was used, as preliminary experiments demonstrated increases typically resulted in training instability. We also investigated several hyperparameter configurations, including for the learning rate, learning rate scheduler, weight decay, and minibatch size, across all the models and found the present configurations to generally work best across all of them.

C.3 Datasets and Frameworks

The datasets we used were CIFAR10, CIFAR100 (Krizhevsky, 2009), STL10 (Coates et al., 2011), SVHN (Netzer et al., 2011), Caltech101 (Li et al., 2022), TinyImageNet (Le & Yang, 2015), Food101 (Bossard et al., 2014), and ImageNet (Deng et al., 2009). We trained our models using the PyTorch (Paszke et al., 2019) machine learning framework. The STL10, SVHN, Caltech101, TinyImagenet, and ImageNet datasets are available for non-commercial use. The CIFAR10 and CIFAR100 datasets are publicly available under the MIT license. The Food101 dataset is available under the CC BY 4.0 license. The PyTorch framework is distributed under the BSD license.

C.4 Computational Resources

We used a cluster of NVIDIA GPUs having variable availability - approximately 6 V100 GPUs and 6 P100 GPUs were available to us at most times. We used a 3:1 CPU to GPU ratio for multi-process

[7]The official Caltech101 dataset does not come with its own training/validation split.

data loading; these CPUs were all Intel CPUs of various specifications. All of the experiments shared these compute resources. Most individual experiments ran in a few hours (with variability based on model and dataset size). All of our experiments were completed in a span of approximately 3–4 weeks. We report all experimental results. We only omit preliminary experiments whose sole objectives were for investigating the hyperparameter configurations which generally worked best across all model and dataset combinations.

D Additional Experiments

In all the figures presented, unless otherwise stated each plotted value represents the mean performance across $M = 6$ models, each of which was trained with differing random initializations for the model parameters.

D.1 Other Noise-Based Techniques

Here we contrast the proposed method of additive Gaussian noise with scaling, which was described in Subsection 4.2, with two other noise-based techniques.

The first is Gaussian dropout (Srivastava et al., 2014; Wang & Manning, 2013; Kingma et al., 2015), where for each input indexed by $i \in \{1, \ldots, N\}$, during training we have $y_i = x_i \cdot (1 + z_i) \cdot \sqrt{\frac{1-p}{p}}$, where x_i is the i-th input's post-power transform value, $z_i \sim \mathcal{N}(0, 1)$, and $p \in (0, 1]$ is the retention rate.

The second is additive Gaussian noise, but without scaling by each channel's minibatch statistics. This corresponds to our proposed method in the case where s is fixed to the mean of a standard half-normal distribution, i.e. $s = \sqrt{\frac{2}{\pi}}$ across all channels; and thus does not depend on the channel statistics.[8]

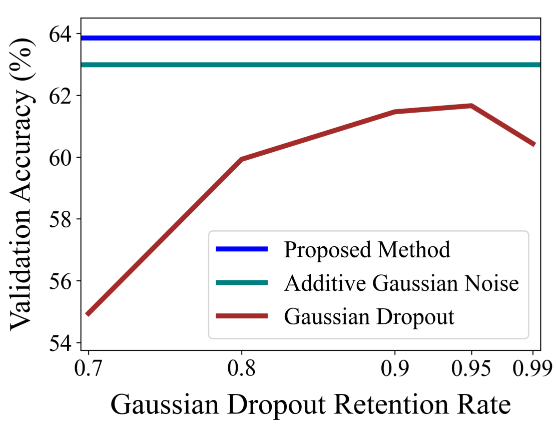

Figure 7: **Additive Gaussian noise with scaling is effective.** Validation accuracy for models trained with BatchNormalNorm (ResNet34/STL10), but with varying forms for the noise component of the normalization layer. See text for details.

Figure 7 shows that additive Gaussian noise with scaling, the proposed noising method in this work, is more effective than Gaussian dropout, giving further evidence for the utility of the proposed method. It is also more effective than additive Gaussian noise (without scaling), which suggests the norm of the channel statistics plays an important role when using additive random noise, which adds further novelty and value to the proposed method.

One reason why additive Gaussian noise with scaling may work better than Gaussian dropout, is because the latter scales activations multiplicatively, which means the effect of the noise is incorporated in the backpropagated errors. In contrast, the proposed method noise component does not contribute to the gradient updates directly, because it is additive. This would suggest that models trained with normality normalization obtain higher generalization performance because they must become robust to misattribution of gradient values during backpropagation, relative to the corrupted activation values during the forward pass.

[8]This value of s precisely mirrors how we calculated s in Subsection 4.2, since recall there we have $s = \frac{1}{N} \left\| \boldsymbol{x} - \bar{\boldsymbol{x}} \right\|_1$.

D.2 CONTROLLING FOR THE POWER TRANSFORM AND THE ADDITIVE NOISE

Figure 8 demonstrates that both components of normality normalization – the power transform, and the additive gaussian noise with scaling – each contribute meaningfully to the increased performance in models trained with normalization normalization.

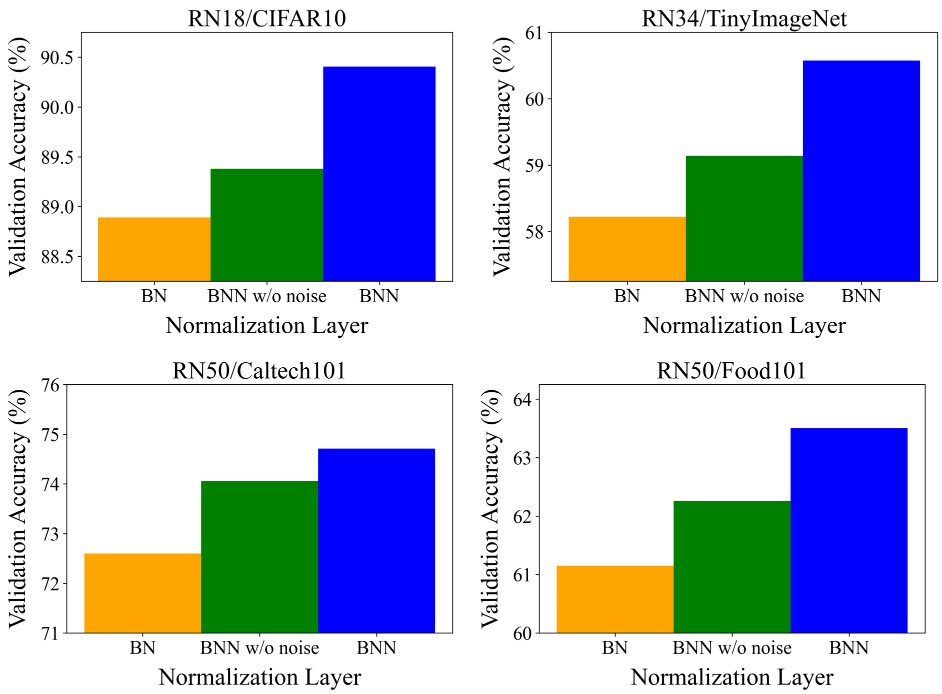

Figure 8: **Controlling for the effects of the power transform and the additive Gaussian noise with scaling components.** Each subplot demonstrates the performance for models trained with the use of additive Gaussian noise with scaling (BNN), and without (BNN w/o noise), while using BatchNorm as a baseline. Subplot titles indicate the model and dataset combination.

Furthermore, we evaluate the performance of models trained with BatchNormalNorm, across various values of ξ, in Figure 9. These results demonstrate that the value of $\xi = 0.4$ works consistently well across differing model and dataset combinations.

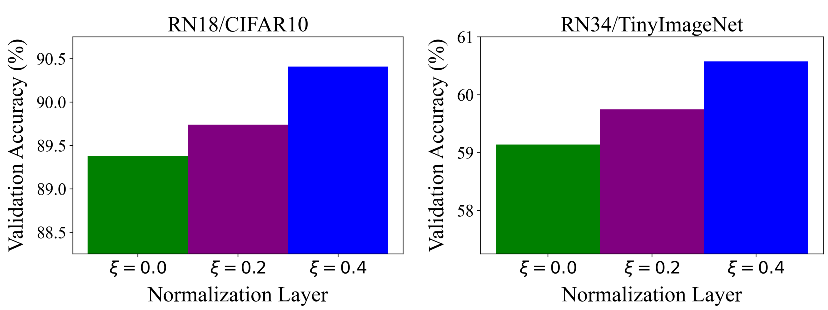

Figure 9: **Varying the noise factor.** Each subplot demonstrates the performance for models trained with BatchNormalNorm (BNN), with varying noise factors ξ. Subplot titles indicate the model and dataset combination.

D.3 EXPERIMENTS WITH DATA AUGMENTATIONS

Here we demonstrate that the proposed normalization layer scales with existing techniques for improving model performance, such as data augmentations.

Table 4 demonstrates the improved performance for vision transformer (ViT) models trained with data augmentations, and demonstrates that models trained with LayerNormalNorm (LNN) maintain their superior performance compared to those trained with LayerNorm (LN), even as both benefit from the use of data augmentations.

Table 5 demonstrates an improvement for the ResNet50 model trained with BatchNormalNorm with $\xi = 0$ (BNN w/o noise) vs. BatchNorm (BN) on the large-scale ImageNet dataset; this further demonstrates that the performance of normality normalization continues to scale with class number and dataset size. Furthermore, by setting $\xi = 0$ (BNN w/o noise), this further acts as a control for the effect of the power transform in BatchNormalNorm, and further complements the findings we presented in Appendix Subsection D.2.

For the models trained on the SVHN dataset, we used mild random translations and rotations. For the models trained on the CIFAR10 and CIFAR100 datasets, we used random cropping, random horizontal flips, and mild color jitters. For the models trained on the Food101 and ImageNet datasets, we used random cropping with resizing, random horizontal flips, and moderate color jitters.

Table 4: Validation accuracy across several benchmarks for a vision transformer (ViT) architecture (see training details for model specifications), when using LayerNormalNorm (LNN) vs. LayerNorm (LN). Here augmentations were employed (see text for details). For each table entry, representing a dataset and model combination, $M = 6$ models were trained, each with differing random initializations for the model parameters.

Dataset	LN	LNN
SVHN	94.46 ± 0.33	$\mathbf{95.94 \pm 0.18}$
CIFAR10	73.71 ± 0.42	$\mathbf{75.47 \pm 0.49}$
CIFAR100	49.56 ± 0.42	$\mathbf{52.89 \pm 0.51}$
Food101	55.43 ± 0.57	$\mathbf{63.04 \pm 0.72}$

Table 5: Validation accuracy for ResNet architectures when using BatchNormalNorm with $\xi = 0$ (BNN w/o noise) vs. BatchNorm (BN). See text for discussion.

Dataset	Model	BN	BNN w/o noise
ImageNet Top1	RN50	71.60	**71.94**
ImageNet Top5	RN50	90.71	**90.83**

D.4 Effect of Degree of Gaussianization

Here we consider what effect differing degrees of gaussianization have on model performance, as measured by the proximity of the estimate $\hat{\lambda}$ to its MLE solution, which was given by Equation 5.

We control the proximity to the MLE solution, using a parameter $\alpha \in [0, 1]$ in the following equation:

$$\hat{\lambda} = 1 - \alpha \frac{\mathcal{L}'(\boldsymbol{h}; \lambda = 1)}{\mathcal{L}''(\boldsymbol{h}; \lambda = 1)}, \tag{15}$$

where $\alpha = 1$ corresponds to the MLE, and decreasing values of α reduce the strength of the gaussianization.

Figure 10 demonstrates that the method's performance increases with increasing α, and obtains its best performance for $\alpha = 1$. This provides further evidence that increasing gaussianity improves model performance.

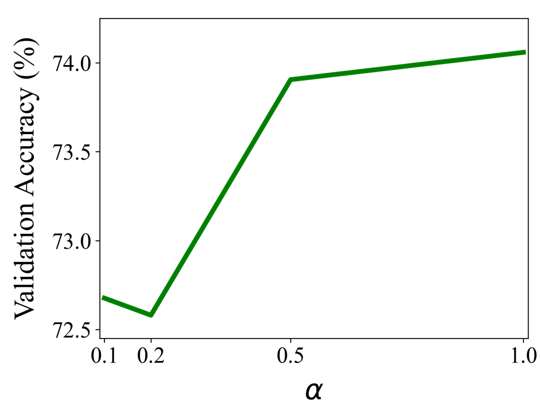

Figure 10: **Increasing gaussianity improves model performance.** Validation accuracy for models trained using BatchNormalNorm without noise (ResNet50/Caltech101), and with varying strengths for the gaussianization (parameterized by α) when applying the power transform. See text for details.

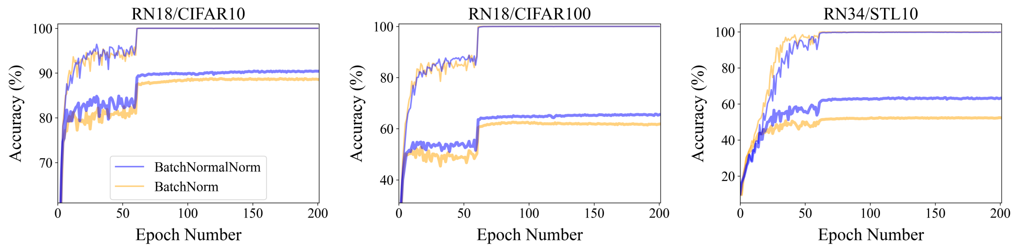

Figure 11: Training and validation curves for models trained with BatchNormalNorm vs. BatchNorm. Bolded lines represent validation accuracy; unbolded lines represent training accuracy.

D.5 Training Convergence

Figure 11 shows that the trends in the training and validation curves generally do not differ when using BatchNormalNorm as compared to BatchNorm. This suggests that the understanding deep learning practitioners have obtained for training models with conventional normalization layers, remains applicable when augmenting those normalization layers with normality normalization.

D.6 Speed Benchmarks

Figure 12 shows the average per-sample running time for models using BatchNormalNorm and BatchNorm. The values are calculated by taking the average minibatch runtime at train/evaluation time, for the entire training/validation set, then normalizing by the number of samples in the minibatch. Values are obtained using an NVIDIA V100 GPU.

The plots shows a close correspondence for test-time performance, with a larger deviation at training time. However, it is worth noting that the operations performed in BatchNormalNorm do not benefit from the low-level optimizations in modern deep learning libraries, afforded to the constituent operations of BatchNorm.

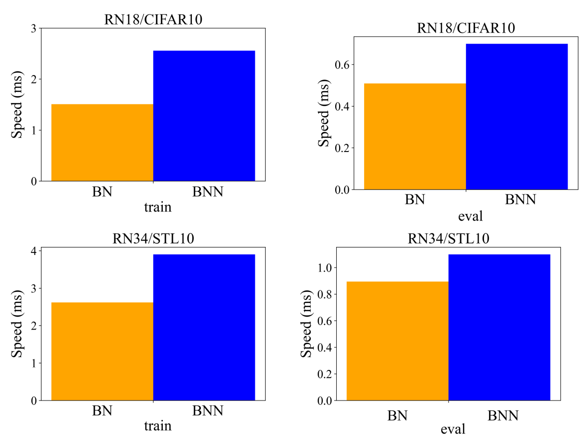

Figure 12: Runtime comparison between models using BatchNormalNorm (BNN) and BatchNorm (BN) for two sets of model & dataset combinations; top: ResNet18/CIFAR10, right: ResNet34/STL10. The left hand plot shows the running time during training, and the right hand plot shows the running time during evaluation. See text for details.

Furthermore, the present work serves as a foundation, both conceptual and methodological, for future works which may continue to leverage the benefits of gaussianizing. We believe improvements to the runtime of normality normalization can be obtained in future work, by leveraging approximations to the operations performed in the present form of normality normalization, or by leveraging low-level optimizations.

D.7 Normality at Initialization

Figure 13 shows representative Q–Q plots, together with an aggregate measure of normality across model layers, for post-power transform feature values when using BatchNormalNorm, and post-normalization values when using BatchNorm, for models at initialization. It demonstrates that at initialization, the pre-activations are close-to Gaussian regardless of the normalization layer employed; and thus that only the model trained with BatchNormalNorm enforces and maintains normality throughout training, as evidenced by Figure 5. Note that the Q–Q plots presented in Figures

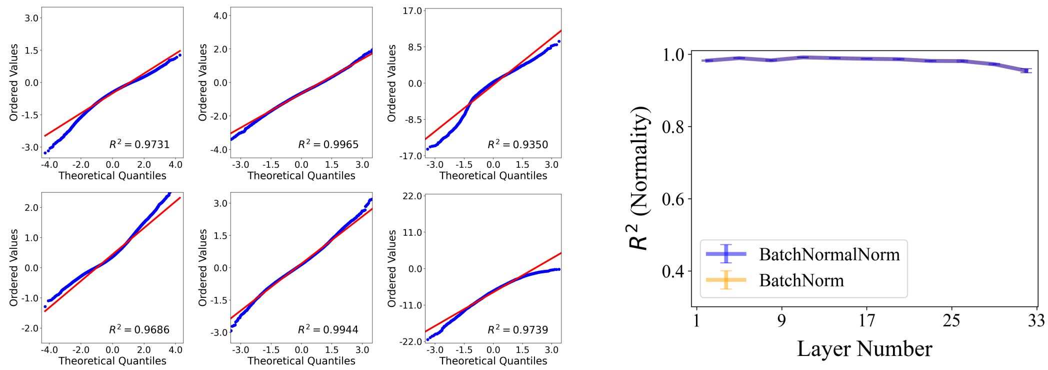

Figure 13: As in Figure 5, but for networks at initialization. The plots demonstrate that, at initialization, networks with either BatchNormalNorm or BatchNormal have close-to Gaussian pre-activations. However, as the networks are trained, BatchNormalNorm enforces and retains normality while BatchNorm does not, as evidenced by Figure 5.

5 and 13, are obtained for precisely the same minibatch and channel combinations, which acts as as a control.

D.8 Uncorrelatedness, Joint Normality, and Independence Between Features

Following the motivation we presented in Subsection 2.3, here we sought to explore the potential effect normality normalization may have on decorrelating features, the extent to which it may increase joint normality in the features, and the extent to which it may increase the independence between features.

We use the following experimental setup. For each layer of a ResNet34/STL10 model trained to convergence using either BatchNormalNorm or BatchNorm, we compute the correlation, joint normality, and mutual information over 20 pairs of channels, and across 10 validation minibatches.

We evaluate joint normality using the negative of the HZ-statistic (Henze & Zirkler, 1990) (higher values indicate greater joint normality), and evaluate independence using the adjusted mutual information (AMI) metric (lower values indicate a greater degree of independence) Vinh et al. (2010)[9]

We evaluate joint normality across pairs of channels rather than across all of the channels in a layer, because measures of joint normality are sensitive to small deviations in sample statistics for finite sample sizes (Zhou & Shao, 2014; Ebner & Henze, 2020). Wherever we measure AMI, we use the square root of the number of sampled features as the number of bins (a generally accepted rule of thumb) when discretizing the features, and we use uniform binning, which is appropriate for (close to) normally distributed data.

Figure 14 demonstrates that models trained with BatchNormalNorm have lower correlation between unit features, higher joint normality, and have greater independence, across the model's layers. This is of value in context of the benefits feature independence is though to provide, which we explored in Subsection 2.3.

E Lemmas

Lemma E.1. *Bivariate Normality Minimizes Mutual Information. Let* $X_1 \sim \mathcal{N}\left(x_1; \mu_1, \sigma_1^2\right)$ *and* $X_2 \sim \mathcal{N}\left(x_2; \mu_2, \sigma_2^2\right)$*. Then their mutual information* $I\left(X_1; X_2\right)$ *is minimized when the random variables are furthermore jointly normally distributed, i.e.* $\left(X_1, X_2\right) \sim \mathcal{N}\left(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}\right)$*, with* $\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$,

[9]The AMI is a variation of mutual information, which adjusts for random chance. It is also bounded between 0 and 1, which makes it easier to interpret.

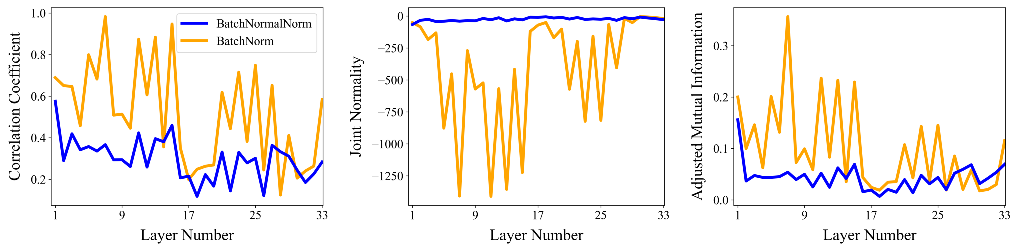

Figure 14: **Normality normalization induces greater feature independence.** Correlation, joint normality, and adjusted mutual information between pairs of channels for models trained to convergence using BatchNormalNorm vs. BatchNorm (ResNet34/STL10). The results are obtained by averaging the corresponding statistics across 20 channel pairs, and across 10 validation minibatches. Here joint normality is quantified using the negative of the HZ-statistic.

$\boldsymbol{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}$, $\boldsymbol{\Sigma} = \begin{bmatrix} \sigma_1^2 & \rho\sigma_1\sigma_2 \\ \rho\sigma_1\sigma_2 & \sigma_2^2 \end{bmatrix}$, *and* ρ *the correlation coefficient between* X_1, X_2. *Furthermore* $I\left(X_1; X_2\right) = \frac{1}{2}\log\left(\frac{1}{1-\rho^2}\right)$.

Proof. Consider two possible distributions, f, g, for the joint distribution over (X_1, X_2), where f denotes the probability density function (PDF) of the bivariate normal distribution, and g can be any joint distribution. Our goal is to show that the mutual information between X_1, X_2, when they are distributed according to g, is lower-bounded by the mutual information between X_1, X_2 when they are distributed according to f.

For clarity of presentation, let the number of variables f and g take as arguments be clear from context, so that it is understood when they are used to denote their marginal distributions. Furthermore let $I_g\left(X_1; X_2\right)$ represent the mutual information when (X_1, X_2) are distributed according to g, with the notation extending analogously to their joint $h_g\left(X_1, X_2\right)$ and marginal $h_g\left(X_1\right)$, $h_g\left(X_2\right)$ entropies under g.

We then have

$$\begin{aligned} I_g\left(X_1; X_2\right) &= h_g\left(X_1\right) + h_g\left(X_2\right) - h_g\left(X_1, X_2\right) \\ &= h_f\left(X_1\right) + h_f\left(X_2\right) - h_g\left(X_1, X_2\right) \\ &\geq h_f\left(X_1\right) + h_f\left(X_2\right) - h_f\left(X_1, X_2\right) \\ &= I_f\left(X_1; X_2\right) \\ &= \frac{1}{2}\log\left(2\pi e\sigma_1^2\right) + \frac{1}{2}\log\left(2\pi e\sigma_2^2\right) - \frac{1}{2}\log\left(\left(2\pi e\right)^2\left(1-\rho^2\right)\sigma_1^2\sigma_2^2\right) \\ &= \frac{1}{2}\log\left(\frac{1}{1-\rho^2}\right), \end{aligned} \tag{16}$$

where the second equality follows because by assumption the marginals are normally distributed, the inequality follows because the normal distribution maximizes entropy, and in the second-last equality we have used the expressions for the entropy of the univariate and bivariate normal distributions. □

Consequently, when the random variables are jointly normally distributed, $\rho = 0$ implies $I\left(X_1; X_2\right) = 0$; thus uncorrelatedness implies independence.[10]

[10] The preceding result extends straightforwardly to the general multivariate setting, i.e. with more than two random variables.