A Comprehensive and Modularized Statistical Framework for Gradient Norm Equality in Deep Neural Networks

Zhaodong Chen, Lei Deng, *Member*, *IEEE*, Bangyan Wang, Guoqi Li, *Member*, *IEEE*, Yuan Xie, *Fellow*, *IEEE*

Abstract—The rapid development of deep neural networks (DNNs) in recent years can be attributed to the various techniques that address gradient explosion and vanishing. In order to understand the principle behind these techniques and develop new methods, plenty of metrics have been proposed to identify networks that are free of gradient explosion and vanishing. However, due to the diversity of network components and complex serial-parallel hybrid connections in modern DNNs, the evaluation of existing metrics usually requires strong assumptions, complex statistical analysis, or has limited application fields, which constraints their spread in the community. In this paper, inspired by the Gradient Norm Equality and dynamical isometry, we first propose a novel metric called Block Dynamical Isometry, which measures the change of gradient norm in individual blocks. Because our Block Dynamical Isometry is norm-based, its evaluation needs weaker assumptions compared with the original dynamical isometry. To mitigate challenging derivation, we propose a highly modularized statistical framework based on free probability. Our framework includes several key theorems to handle complex serial-parallel hybrid connections and a library to cover the diversity of network components. Besides, several sufficient conditions for prerequisites are provided. Powered by our metric and framework, we analyze extensive initialization, normalization, and network structures. We find that our Block Dynamical Isometry is a universal philosophy behind them. Then, we improve some existing methods based on our analysis, including an activation function selection strategy for initialization techniques, a new configuration for weight normalization, a depth-aware way to derive coefficients in SeLU, and initialization/weight normalization in DenseNet. Moreover, we propose a novel normalization technique named second moment normalization, which has 30% fewer computation overhead than batch normalization without accuracy loss and has better performance under micro batch size. Last but not least, our conclusions and methods are evidenced by extensive experiments on multiple models over CIFAR-10 and ImageNet.

Index Terms—Deep Neural Networks, Free Probability, Gradient Norm Equality

1 Introduction

It has become a common sense that deep neural networks (DNNs) are more effective compared with the shallow ones. However, the training of very deep models usually suffers from gradient explosion and vanishing. To this end, plenty of schemes and network structures have been proposed. For instance: He et al. (2015) [1], Mishkin & Matas (2015) [2], Xiao et al. (2018) [3] and Zhang et al. (2019) [4] suggest that the explosion and vanishing can be mitigated with proper initialization of network parameters. Ioffe & Szegedy (2015) [5], Salimans & Kingma (2016) [6] and Qiao et al. (2019) [7] propose several normalization schemes that can stabilize the neural networks during training. From the perspective of network structures, He et al. (2016) [8] and Huang et al. (2017) [9] demonstrate that neural networks with shortcuts can effectively avoid gradient vanishing and explosion.

The work was partially supported by National Science Foundation (Grant No. 1725447), Tsinghua University Initiative Scientific Research Program, Tsinghua-Foshan Innovation Special Fund (TFISF), and National Natural Science Foundation of China (Grant No. 61876215). Zhaodong Chen and Lei Deng contributed equally to this work, corresponding author: Guoqi Li.
Zhaodong Chen, Lei Deng, Bangyan Wang, and Yuan Xie are with the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106, USA (email: {chenzd15thu, leideng, bangyan, yuanxie}@ucsb.edu). Guoqi Li is with the Department of Precision Instrument, Center for Brain Inspired Computing Research, Tsinghua University, Beijing 100084, China (email: liguoqi@mail.tsinghua.edu.cn).

It's natural to ask: *is there a common philosophy behind all these studies?* Such philosophy may inspire novel hyper-parameter selection strategies and network structures.

Great efforts have been made to pursue this philosophy. In particular, He et al. (2015) [1] and Mishkin & Matas (2015) [2] preserve the information flow in the forward pass. Poole et al. (2016) [10], Xiao et al. (2018) [3], Yang et al. (2019) [11] and Schoenholz et al. (2016) [12] study the stability of the statistics fixed point with dynamical mean-field theory. They identify an order-to-chaos phase transition in deep neural networks, and networks sit on the border between two phases are trainable even with a depth of 10,000 [3]. Pennington et al. (2017, 2018) [13], [14], Tarnowski et al. (2018) [15], and Ling & Qiu (2018) [16] argue that networks achieving dynamical isometry (all the singular values of the network's input-output Jacobian matrix remain close to 1) do not suffer from gradient explosion or vanishing. Philipp et al. (2019) [17] directly evaluate the statistics of the gradient and propose a metric called gradient scale coefficient (GSC) that can verify whether a network would suffer gradient explosion. Arpit & Bengio (2019) [18] find that networks with Gradient Norm Equality property usually have better performance. Gradient Norm Equality means that the Frobenius Norm of the gradient is more or less equal in different layers' weights, therefore the information flow in the backward pass can be preserved and the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

gradient explosion and vanishing are prevented. However, most of these studies only provide explanations for existing methods, and few of them are applied in discovering novel algorithms for cutting-edge DNN models. The major reason is that these studies lack handy statistical tools to apply in complex network structures.

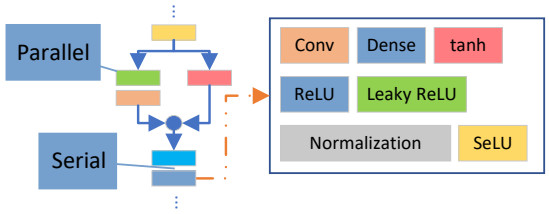


Fig. 1. Illustration of complex network structure.

As illustrated in Fig. 1, modern neural networks are usually composed of several different kinds of linear or nonlinear components like convolution, activation function, and normalization. These components are connected either in parallel or serial. The diversity of network components and different kinds of connections result in two challenges: nontrivial prerequisites and complex derivation. Because of the former one, some studies rely on strong assumptions. For example, to calculate the quadratic mean norm (qm norm) of the Jacobian matrices, Philipp et al. (2019) [17] assume that the norm of the product of Jacobian matrices has approximate decomposability. The free probability used in Pennington et al. (2017, 2018) [13], [14], Tarnowski et al. (2018) [15], and Ling & Qiu (2018) [16] requires the involved matrices to be freely independent with each other [19], which is not commonly held and difficult to verify [20]. Because of the complex derivation, existing studies usually require strong statistics backgrounds, which constrains their spread in the community. For example, the derivation with free probability requires the probability density of the eigenvalues in each Jacobian matrix, which then goes through several complex transforms and series expansions [16]. Last, these challenges also limit the applicable scope of existing studies such that they only support simple serial networks with few kinds of components [1], [3], [10], [11], [12], [18].

In this paper, we analyze neural networks with the workflow as follows. First, we break the network down into serial blocks (e.g. residual block in ResNet [8]) and analyze how the Frobenius norm of the gradient evolves during propagation in Section 3. Based on the analysis, we propose a new metric, block dynamical isometry (Definition 3.1), that characterizes whether gradient norm equality is achieved based on the first and second moments of Jacobian matrices' eigenvalues (referred as spectrum-moment) in each serial block. Second, to calculate the spectrum-moment of each block, we develop a highly modularized statistical framework in Section 4. As each serial block is constructed by connecting basic components like ReLU and Convolutions in serial or parallel, we extend the conclusions in Ling & Qiu (2018) [16] and provide two main theorems in Section 4.2. Specifically, when the spectrum-moments of individual components are provided, Theorem 4.1 & 4.2 compute the spectrum-moment of the whole block with elementary arithmetic when components are connected in serial and parallel, respectively. The prerequisites of these theorems are thoroughly discussed in Section 4.3 in which we present several handy sufficient conditions. At last, in Section 4.4, a library (Table 3) is developed that summarizes the spectrum-moments and whether the sufficient conditions hold for extensive components of neural networks.

Comparing with previous studies, our norm-based metric relies on much weaker prerequisites that are easier to verify. The highly modularized framework simplifies the complex statistical derivation to elementary arithmetic, and our library covers the diversity of network components.

TABLE 1
Study cases.

Type	Study Cases
I. Existing Techniques	1. Initialization [1], [3], [4]
	2. Normalization [5], [6], [21]
	3. Self-normalizing Neural Network [22]
	4. ResNet [8] & DenseNet [9]
II. Improvements	1. Impact of Activation Functions in Initialization Techniques (Section 5.1)
	2. Scaled Weight Standardization (Section 5.2)
	3. Depth-aware Self-Normalizing Neural Network (Section 5.3)
	4. Initialization and Weight Normalization in DenseNet (Section 5.5)
III. Novel Method	1. Second Moment Normalization (Section 5.2)

To show the effectiveness of our framework, we present three types of study cases in Section 5 and 6 as summarized in Table 1. In Type I, we show that theoretical interpretations of comprehensive existing techniques can be provided with a few lines of derivation under our framework. These interpretations also prove that our block dynamical isometry is a universal philosophy behind them.

In Type II, we present several improvements over existing studies based on the insights from Type I study cases. In particular, in case II-1, we systematically evaluate the impact of different activation functions on the spectrum-moments and identify that although tanh used in Xiao et al. (2018) [3] is more stable, leaky ReLU with relatively higher negative slope coefficient is more effective in networks with moderate depth. Besides, we modify the PReLU activation function proposed by He et al. (2015) [1] and give a novel one called sPReLU that automatically learns a good negative slope coefficient. In case II-2, we combine weight normalization with the initialization techniques and propose a method called scaled weight standardization. In case II-3, we identify that the coefficients in the SeLU activation function should be given according to the depth of the network, and provide a new way to find these coefficients. In case II-4, we show that initialization and weight normalization techniques for vanilla CNNs can be directly applied to DenseNet. All these novel methods are verified by extensive experiments on CIFAR-10 and ImageNet.

In Type III, we propose a new normalization technique called second moment normalization that has 30% lower computation overhead than BN. On ImageNet-ResNet50 with proper regularization like mixup [23], our second moment normalization achieves 0.07% and 1.96% lower top-1 error than BN under normal and micro batch size scenarios, respectively.

For the sake of clarity, we provide a description of the default notations used throughout this paper in Table 2. Our codes in PyTorch are publicly available at https://github.com/apuaaChen/GNEDNN_release.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 2
Default notations.

Numbers, Arrays and Matrices			
a	a scalar	$\mathbf{a}$	a column vector
$\mathbf{A}$	a matrix	$\mathbf{n}, n \in \mathbb{R}$	a vector or matrix
$\mathbf{I}$	square identify matrix		
Operators			
$Tr(\mathbf{A})$	the trace of $\mathbf{A}$	$tr(\mathbf{A})$	the normalized trace of $\mathbf{A}$, e.g. $tr(\mathbf{I}) = 1$
$E[x]$	the expectation of r.v. x	$D[x]$	the variance of r.v. x
$\lambda_{\mathbf{A}}$	the eigenvalues of $\mathbf{A}$	$\alpha_k(a)$	the k^{th} order moment of r.v. a
$\mathbf{f}(\mathbf{a})$	a mapping function taking $\mathbf{a}$ as input	$\mathbf{f_a}$	the Jacobian matrix $\frac{\partial \mathbf{f}(\mathbf{a})}{\partial \mathbf{a}}$
$\phi(\mathbf{A}) := E[tr(\mathbf{A})]$	the expectation of $tr(\mathbf{A})$	$\varphi(\mathbf{A})$	$\phi(\mathbf{A^2}) - \phi^2(\mathbf{A})$
$height(\mathbf{A})$	the height of matrix $\mathbf{A}$	$width(\mathbf{A})$	the width of matrix $\mathbf{A}$
$len(\mathbf{a})$	the length of vector $\mathbf{a}$		
Index			
$[\mathbf{A}]_{i,j}$	element(i, j) of $\mathbf{A}$		

2 Related Work

2.1 Theorems of Well-behaved Neural Networks

Dynamical Isometry. A neural network is dynamical isometry as long as every singular value of its input-output Jacobian matrix remains close to 1, thus the norm of every error vector and the angle between error vectors are preserved. With the powerful theorems of free probability and random matrix, Pennington et al. (2017) [13] investigate the spectrum density distribution of plaint fully-connected serial network with Gaussian/orthogonal weights and ReLU/hard-tanh activation functions; Tarnowski et al. (2018) [15] explore the density of singular values of the input-output Jacobian matrix in ResNet and identify that dynamical isometry can be always achieved regardless of the choice of the activation function. However, their studies only cover ResNet whose major branch consists of Gaussian and scaled orthogonal linear transforms and activation functions, and fail to provide a theoretical explanation of batch normalization. Although our derivations of Theorem 4.1 and 4.2 are inspired by the Result 2 & 1 in Ling & Qiu (2018) [16], their discussions are limited to the spectrum of ResNet due to two reasons. First, their derivation requires the detailed spectrum density of involved components; second, they fail to realize that although the trace operator is cyclic-invariant, the normalized trace operator is not when rectangle matrices are involved, so that their Result 2 can only handle square Jacobian matrices. Last but not least, a universal problem in existing dynamical isometry related studies is that the derivation is based on the strong assumption that all the involved matrices are freely independent, which is uncommonly held and difficult to verify [20].

Order-to-Chaos Phase Transition. Poole et al. (2016) [10] and Schoenholz et al. (2016) [12] analyze the signal propagation in simple serial neural networks and observe that there is an order-to-chaos phase transition determined by a quantity: $\chi := \phi\left((\mathbf{DW})^T \mathbf{DW}\right)$ [13], where $\mathbf{D}$ is the Jacobian matrix of activation function, $\mathbf{W}$ denotes the weight and ϕ represents the expectation of the normalized trace of a given matrix. The network is in the chaotic phase if $\chi > 1$ and in the order phase when $\chi < 1$. The chaotic phase results in gradient explosion while the order phase causes gradient vanishing. Due to the lack of convenient mathematic tools for analysis of "ϕ", the current application of the order-to-chaos phase transition is also limited to vanilla serial networks.

Gradient Scale Coefficient (GSC). Philipp et al.(2018) [17] propose a metric that evaluates how fast the gradient explodes. Let $0 \leq l \leq k \leq L$ be the index of the network's layers, the GSC is defined as

$$GSC(k, l) = \frac{\phi\left(\left(\Pi_{i=l}^{k} \mathbf{J_i}\right)^T \left(\Pi_{i=l}^{k} \mathbf{J_i}\right)\right) ||f_k||_2^2}{||f_l||_2^2}. \quad (1)$$

To efficiently calculate this metric, the authors suggest that $GSC(k, l) = \Pi_{i=l}^{k-1} GCS(i + 1, i)$, which is derived under the assumption: $\phi\left(\left(\Pi_{i=l}^{k} \mathbf{J_i}\right)^T \left(\Pi_{i=l}^{k} \mathbf{J_i}\right)\right) = \Pi_{i=l}^{k} \phi(\mathbf{J_i}^T \mathbf{J_i})$. In our work, we provide not only a solid derivation for this assumption but also theoretical tools for networks with parallel branches, which makes our method more solid and applicable in more general situations.

2.2 Techniques that Stabilize the Network

Initialization. It has long been observed that neural networks with proper initialization converge faster and better. Thus, handful initialization schemes have been proposed: He et al. (2015) [1] introduce Kaiming initialization that maintains the second moment of activations through plaint serial neural networks with rectifier activations; Zhang et al. (2019) [4] extend initialization techniques to networks with shortcut connections like ResNet and achieve advanced results without BN; Xiao et al. (2018) [3] provide an orthogonal initialization scheme for serial neural networks, which makes 10,000-layer networks trainable.

Normalization. Batch normalization (BN) [5] has become a standard implementation in modern neural networks [8], [9]. BN leverages the statistics (mean & variance) of mini-batches to standardize the pre-activations and allows the network to go deeper without significant gradient explosion or vanishing. Despite BN's wide application, it has been reported that BN introduces high training latency [24], [25] and its effectiveness drops when the batch size is small [26]. Moreover, BN is also identified as one of the major roots causing quantization loss [27]. To alleviate these problems, Salimans & Kingma (2016) [6] instead normalize the weights, whereas it is less stable compared with BN [25].

Self-normalizing Neural Network. Klambauer et al. (2017) [22] introduce a novel activation function called "scaled exponential linear unit" (SeLU), which can automatically force the activation towards zero mean and unit variance for better convergence

Shortcut Connection. In He et al. (2016) [8], the concept of shortcut in neural networks was first introduced and then further developed by Huang et al. (2017) [9], which results in two most popular CNNs named ResNet and DenseNet. These models demonstrate that the shortcut connections make deeper models trainable.

3 Gradient Norm Equality

We analyze how the Forbenius norm of backward gradient evolves through the network in Section 3.1 and derive our metric for gradient norm equality in Section 3.2.

3.1 Dynamic of Gradient Norm

Without loss of generality, let's consider a neural network consists of serial blocks:

$$\mathbf{f}(\mathbf{x_0}) = \mathbf{f}_{\mathbf{L},\theta_\mathbf{L}} \circ \mathbf{f}_{\mathbf{L-1},\theta_{\mathbf{L-1}}} \circ ... \circ \mathbf{f}_{\mathbf{1},\theta_\mathbf{1}}(\mathbf{x_0}), \tag{2}$$

where $\theta_\mathbf{i}$ is the vectorized parameter of the i^{th} layer. We represent the loss function as $\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y})$ wherein $\mathbf{y}$ denotes the label vector. At each iteration, $\theta_\mathbf{i}$ is updated by $\theta_\mathbf{i} - \mathbf{\Delta}\theta_\mathbf{i} = \theta_\mathbf{i} - \eta\frac{\partial}{\partial\theta_\mathbf{i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y})$, where η is the learning rate. With the chain rule, we have

$$\begin{aligned}\frac{\partial}{\partial \mathbf{f_i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}) &= \left(\frac{\partial \mathbf{f_{i+1}}}{\partial \mathbf{f_i}}\right)^T \frac{\partial}{\partial \mathbf{f_{i+1}}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}),\\ \frac{\partial}{\partial \theta_\mathbf{i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}) &= \left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T \frac{\partial}{\partial \mathbf{f_i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}).\end{aligned} \tag{3}$$

For the sake of simplicity, we denote $\frac{\partial \mathbf{f_j}}{\partial \mathbf{f_{j-1}}} := \mathbf{J_j} \in \mathbb{R}^{m_j \times n_j}$, and $\mathbf{\Delta}\theta_\mathbf{i}$ is given by

$$\mathbf{\Delta}\theta_\mathbf{i} = \eta\left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T \left(\Pi_{j=L}^{i+1}\mathbf{J_j}\right)^T \frac{\partial}{\partial \mathbf{f}(\mathbf{x})}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}). \tag{4}$$

Further, we denote $\mathbf{K_{i+1}} := \left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T \left(\Pi_{j=L}^{i+1}\mathbf{J_j}\right)^T$ and $\mathbf{u} := \frac{\partial}{\partial \mathbf{f}(\mathbf{x})}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y})$. We represent the scale of $\mathbf{\Delta}\theta_\mathbf{i}$ with its Forbenius norm: $||\mathbf{\Delta}\theta_\mathbf{i}||_2^2 = \eta^2\mathbf{u}^T\mathbf{K_{i+1}^T}\mathbf{K_{i+1}}\mathbf{u}$. As $\mathbf{K_{i+1}^T}\mathbf{K_{i+1}}$ is a real symmetric matrix, it can be broken down with eigendecomposition: $\mathbf{K_{i+1}^T}\mathbf{K_{i+1}} = \mathbf{Q}^T\mathbf{\Lambda}\mathbf{Q}$, where $\mathbf{Q}$ is an orthogonal matrix. Therefore we have:

$$\begin{aligned}||\mathbf{\Delta}\theta_\mathbf{i}||_2^2 &= \eta^2(\mathbf{Qu})^T\mathbf{\Lambda}(\mathbf{Qu}),\\ E\left[||\mathbf{\Delta}\theta_\mathbf{i}||_2^2\right] &= \eta^2 E\left[\sum_j \lambda_j[\mathbf{Qu}]_j^2\right].\end{aligned} \tag{5}$$

With the symmetry, we can assume $\forall i, j,\ E[[\mathbf{Qu}_i]^2] = E[[\mathbf{Qu}_j]^2]$, $E[\lambda_i] = E[\lambda_j]$ and λ_j is independent of $[\mathbf{Qu}]_j$, then we have

$$\begin{aligned}E\left[||\mathbf{\Delta}\theta_\mathbf{i}||_2^2\right] &= \eta^2\sum_j E[\lambda_j]E\left[[\mathbf{Qu}]_i^2\right]\\ &\approx \eta^2\phi\left(\mathbf{K_{i+1}^T}\mathbf{K_{i+1}}\right)E\left[||\mathbf{u}||_2^2\right].\end{aligned} \tag{6}$$

If $E[||\Delta\theta_\mathbf{i}||_2^2] \to 0$, the update of parameters of the i^{th} layer would be too tiny to make a difference and thus the gradient vanishing occurs; If $E[||\Delta\theta_\mathbf{i}||_2^2] \to \infty$, the parameters of the i^{th} layer would be drastically updated and thus the gradient explosion happens. Therefore, the network is stable when $\phi\left(\mathbf{K_{i+1}^T}\mathbf{K_{i+1}}\right)$ neither grows nor diminishes exponentially with the decreasing of i, which accords with the definition of gradient norm equality [18].

3.2 Block Dynamical Isometry

In order to simplify the derivation of

$$\phi\left(\mathbf{K_{i+1}^T}\mathbf{K_{i+1}}\right) = \phi\left(\left(\Pi_{j=L}^{i+1}\mathbf{J_j}\right)\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T\left(\Pi_{i=L}^{i+1}\mathbf{J_j}\right)^T\right), \tag{7}$$

we temporarily propose Hypothesis 3.1, which is inspired by the assumption on approximate decomposability of the norm of the product of Jacobians in Philipp et al. (2018) [17].

Hypothesis 3.1. *Under some prerequisites, given a set of Jacobian matrices* $\{\mathbf{J_L}, ..., \mathbf{J_{i+1}}, \frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\}$*, we have*

$$\begin{aligned}&\phi\left(\left(\Pi_{j=L}^{i+1}\mathbf{J_j}\right)\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T\left(\Pi_{i=L}^{i+1}\mathbf{J_j}\right)^T\right)\\ &\qquad= \phi\left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\left(\frac{\partial \mathbf{f_i}}{\partial \theta_\mathbf{i}}\right)^T\right)\Pi_{j=L}^{i+1}\phi\left(\mathbf{J_j}\mathbf{J_j}^T\right).\end{aligned} \tag{8}$$

With the theoretical tools developed in Section 4, this hypothesis can be easily proved and the prerequisites can be confirmed (Remark 4.1). In Hypothesis 3.1, the only term that may result in the unsteady gradient is $\Pi_{j=L}^{i+1}\phi(\mathbf{J_j}\mathbf{J_j}^T)$. Therefore, the gradient norm equality can be achieved by forcing $\forall j, \phi(\mathbf{J_j}\mathbf{J_j}^T) \approx 1$.

However, the above condition is not sufficient for neural networks with finite width. We have $tr\left(\mathbf{J_j}\mathbf{J_j}^T\right) = \frac{1}{m_j}\sum_{i=1}^{m_j}\lambda_i$, where λ_i denotes the i^{th} eigenvalue of $\mathbf{J_j}\mathbf{J_j}^T$. Under the assumption that $\forall p, q \neq p, \lambda_p$ is independent of λ_q, the variance of $tr(\mathbf{J_j}\mathbf{J_j}^T)$ is given by

$$\begin{aligned}D[tr(\mathbf{J_j}\mathbf{J_j}^T)] &= \frac{1}{m_j}\sum_{i=1}^{m_j}E[\lambda_i^2] - E^2[\lambda_i]\\ &= \phi\left(\left(\mathbf{J_j}\mathbf{J_j}^T\right)^2\right) - \phi^2\left(\mathbf{J_j}\mathbf{J_j}^T\right) := \varphi\left(\mathbf{J_j}\mathbf{J_j}^T\right).\end{aligned} \tag{9}$$

As a result, for networks that are not wide enough, in order to make sure that the $\phi(\mathbf{J_j}\mathbf{J_j}^T)$ of each block sits steadily around 1, we expect $\varphi(\mathbf{J_j}\mathbf{J_j}^T)$ of each block to be small. Therefore, our metric can be formally formulated as below.

Definition 3.1. ***(Block Dynamical Isometry)*** *Consider a neural network that can be represented as a sequence of blocks as Equation* (2) *and the* j^{th} *block's Jacobian matrix is denoted as* $\mathbf{J_j}$*. If* $\forall j,\ \ \phi(\mathbf{J_j}\mathbf{J_j}^T) \approx 1$ *and* $\varphi(\mathbf{J_j}\mathbf{J_j}^T) \approx 0$*, we say the network achieves block dynamical isometry.*

As $\phi(\mathbf{J_j}\mathbf{J_j}^T)$ and $\varphi(\mathbf{J_j}\mathbf{J_j}^T)$ can be regarded as the first and second moment of eigenvalues of $\mathbf{J_j}$, we name them as **spectrum-moments**. While $\phi(\mathbf{J_j}\mathbf{J_j}^T) \approx 1$ addresses the problem of gradient explosion and vanishing, $\varphi(\mathbf{J_j}\mathbf{J_j}^T) \approx 0$ ensures that $\phi(\mathbf{J_j}\mathbf{J_j}^T) \approx 1$ is steadily achieved. Actually, we find that in many cases, $\phi(\mathbf{J_j}\mathbf{J_j}^T) \approx 1$ is enough to instruct the design or analysis of a neural network.

We name our metric as "Block Dynamical Isometry" is because it is quite similar to the dynamical isometry discussed in Saxe et al. (2013) [28] and Pennington et al. (2017;2018) [13], [14]. The original dynamical isometry expects that every singular value of the whole network's input-output Jacobian matrix remains close to 1, while ours expects them to have average value 1 and small variance in every sequential block. Definition 3.1 allows us to use divide and conquer in the analysis of a complex neural network: a network can be first divided into several blocks connected in serial, and then conquered individually.

4 Analysis of Spectrum-Moments

Definition 3.1 shows that we can identify networks that achieve gradient norm equality by examining the spectrum-moment, i.e. $\phi(\mathbf{J}\mathbf{J}^T)$ and $\varphi(\mathbf{J}\mathbf{J}^T)$, of its serial blocks. Let the Jacobian matrices of the block's components be $\{\mathbf{J_i}\}$, we have $\mathbf{J} = \Pi_i \mathbf{J_i}$ and $\mathbf{J} = \sum_i \mathbf{J_i}$ when they are connected in serial and parallel, respectively. In this section, we develop a highly modularized framework to compute the spectrum-moments of each block. While Section 4.1 presents the inspiration of our framework, Section 4.2 introduces Theorem 4.1 and 4.2 that build bridges between $\phi(\mathbf{J}\mathbf{J}^T)$ and $\varphi(\mathbf{J}\mathbf{J}^T)$ and the spectrum moment of individual components, $\phi(\mathbf{J_i}\mathbf{J_i}^T)$ and $\varphi(\mathbf{J_i}\mathbf{J_i}^T)$, in serial and parallel connections. Unlike previous studies that use arbitrary assumptions like freely independent, the prerequisites of the theorems in our paper are thoroughly discussed in Section 4.3, and two sufficient conditions, Proposition 4.1 and 4.2, are provided. Therefore, only moderate assumptions are required in our paper. A library (Table 3) that summarizes the spectrum-moment of common neural network components and whether prerequisites hold is introduced in Section 4.4. Last but not least, the proof of Hypothesis 3.1 is discussed in Remark 4.1. The workflow with our framework is as follows. For each block, we look up the spectrum-moment of its components from Table 3, check the prerequisites with sufficient conditions in Section 4.3, and compute the spectrum-moments of the block with Theorem 4.1 and 4.2. Finally, we exam the block with block dynamical isometry in Definition 3.1.

4.1 Inspiration: Propagation of Forbenius Norm in a Rotational-Invariant System

Definition 4.1. ***(Rotational-Invariant Distribution)*** *Given a random vector $\mathbf{g_i}$, we say it has rotational-invariant distribution if it has the same distribution with $\mathbf{U}\mathbf{g_i}$, for any unitary matrix $\mathbf{U}$ independent of $\mathbf{g_i}$.*

Let the gradient of the i^{th} layer be $\frac{\partial}{\partial \mathbf{f_i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}) = \mathbf{g_i}$, where $\mathbf{g_i}$ has a rotational-invariant distribution (Definition 4.1). Under this assumption, intuitively, its elements share the same second moment. The gradient of its previous layer is $\frac{\partial}{\partial \mathbf{f_{i-1}}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}) = \mathbf{g_{i-1}} = \mathbf{J_i}\frac{\partial}{\partial \mathbf{f_i}}\mathcal{L}(\mathbf{f}(\mathbf{x}), \mathbf{y}) = \mathbf{J_i}\mathbf{g_i}$, where $\mathbf{J_i}$ is a Jacobian matrix. As the values in the Jacobian matrix are trainable, we can also assume that it is a random matrix. With the singular value decomposition, we have $\mathbf{J_i} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$, where $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices, and $\mathbf{\Sigma}$ is a diagonal matrix whose diagonal entries are the singular values $(\sigma_1, \sigma_2, ...)$ of $\mathbf{J_i}$. When we calculate $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\mathbf{g_i}$, $\mathbf{V}^H$ first rotates the origin distribution to the new orthogonal basis, and then $\mathbf{\Sigma}$ stretches each basis by the corresponding singular value. At last. $\mathbf{U}$ rotates the distribution to the output orthogonal basis.

On the one hand, the distribution of $\mathbf{g_i}$ is invariant under the rotation of $\mathbf{V}^H$. On the other hand, since $\mathbf{U}$ is a unitary matrix, it doesn't change the L_2 norm of $\mathbf{\Sigma}\mathbf{V}^H\mathbf{g_i}$. Therefore, we have

$$\begin{aligned} E\left[||\mathbf{g_{i-1}}||_2^2\right] &= E\left[||[\sigma_1[\mathbf{g_i}]_1, \sigma_2[\mathbf{g_i}]_2, ..., \sigma_m[\mathbf{g_i}]_m]^T||_2^2\right] \\ &= \sum_{j=1}^{m} E[\sigma_j^2]E[[\mathbf{g_i}]_j^2] = \phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) E\left[||\mathbf{g_i}||_2^2\right]. \end{aligned} \tag{10}$$

The above derivation is valid when $\mathbf{g_i}$ is invariant under rotation. Therefore, if we want to calculate the L_2 norm of the gradient of all the layers with Equation (10), we have to make sure that any rotation of its intermediate value will not change $\phi\left((\Pi_i\mathbf{J_i})(\Pi_i\mathbf{J_i})^T\right)$.

4.2 Main Theorems

Inspired by the previous discussions as well as Tarnowski et al. (2018) [15], we formulate the main theorems of this paper as below.

Definition 4.2. **($\mathbf{k^{th}}$ *Moment Unitarily Invariant)*** *Let $\{\mathbf{A_i}\} := \{\mathbf{A_1}, \mathbf{A_2}..., \mathbf{A_L}\}$ be a series independent random matrices. Let $\{\mathbf{U_i}\} := \{\mathbf{U_1}, \mathbf{U_3}..., \mathbf{U_L}\}$ be a series independent haar unitary matrices independent of $\{\mathbf{A_1}, \mathbf{A_2}..., \mathbf{A_L}\}$. We say that $(\Pi_i\mathbf{A_i})(\Pi_i\mathbf{A_i})^T$ is the k^{th} moment unitarily invariant if $\forall 0 < p \leq k$, we have*

$$\phi\left(\left((\Pi_i\mathbf{A_i})(\Pi_i\mathbf{A_i})^T\right)^p\right) = \phi\left(\left((\Pi_i\mathbf{U_i}\mathbf{A_i})(\Pi_i\mathbf{U_i}\mathbf{A_i})^T\right)^p\right). \tag{11}$$

And we say that $(\sum_i \mathbf{A_i})(\sum_i \mathbf{A_i})^T$ is k^{th} moment unitarily invariant if $\forall 0 < p \leq k$, we have

$$\phi\left(\left((\sum_i \mathbf{A_i})(\sum_i \mathbf{A_i})^T\right)^p\right) = \phi\left(\left((\sum_i \mathbf{U_i}\mathbf{A_i})(\sum_i \mathbf{U_i}\mathbf{A_i})^T\right)^p\right). \tag{12}$$

Definition 4.3. ***(Central Matrix)*** *A matrix $\mathbf{A}$ is called a central matrix if $\forall i, j$, we have $E[[\mathbf{A}]_{i,j}] = 0$.*

Definition 4.4. ***(R-diagonal Matrices)*** *(Definition 17 in Cakmak (2012) [29]) A random matrix $\mathbf{X}$ is R-diagonal if it can be decomposed as $\mathbf{X} = \mathbf{U}\mathbf{Y}$, such that $\mathbf{U}$ is Haar unitary and free of $\mathbf{Y} = \sqrt{\mathbf{X}\mathbf{X}^H}$.*

Theorem 4.1. ***(Multiplication).*** *Given $\mathbf{J} := \Pi_{i=L}^1 \mathbf{J_i}$, where $\{\mathbf{J_i} \in \mathbb{R}^{m_i \times m_{i-1}}\}$ is a series of independent random matrices. If $(\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T$ is at least the 1^{st} moment unitarily invariant (Definition 4.2), we have*

$$\phi\left((\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T\right) = \Pi_{i=L}^1\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right). \tag{13}$$

If $(\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T$ is at least the 2^{nd} moment unitarily invariant (Definition 4.2), we have

$$\begin{aligned} &\varphi\left((\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T\right) \\ &= \phi^2\left((\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T\right)\sum_i \frac{m_L}{m_i}\frac{\varphi\left(\mathbf{J_i}\mathbf{J_i}^T\right)}{\phi^2\left(\mathbf{J_i}\mathbf{J_i}^T\right)}. \end{aligned} \tag{14}$$

(Proof: Appendix A.2)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 3
Common components in neural networks (Proof: Appendix A.6).

Part	$\phi(\mathbf{J}\mathbf{J}^T)$	$\varphi(\mathbf{J}\mathbf{J}^T)$	**Def. 4.5**	**Def. 4.3**
Activation Functions [1]				
ReLU($P(x>0)=p$)	p	$p-p^2$	$\surd$	$\times$
leaky ReLU($P(x>0)=p$), γ: negative slop coefficient	$p+\gamma^2(1-p)$	$\gamma^4(1-p)+p-(p+\gamma^2(1-p))^2$	$\surd$	$\times$
tanh	1	0	$\surd$	$\times$
Linear Transformations				
Dense($\mathbf{u}:=\mathbf{K}\mathbf{y}$), $\mathbf{K}\in\mathbb{R}^{m\times n}\sim i.i.d.N(0,\sigma^2)$	$n\sigma^2$	$mn\sigma^4$	$\surd$	$\surd$
CONV($\mathbf{u}:=\mathbf{K}\star\mathbf{y}$), $\mathbf{K}\in\mathbb{R}^{c_{out}\times c_{in}\times k_h\times k_w}\sim i.i.d.N(0,\sigma^2)$	$c_in\widetilde{k_hk_w}\sigma^2$, [1]		$\surd$	$\surd$
Orthogonal($\mathbf{u}:=\mathbf{K}\mathbf{y}$, $\mathbf{K}\mathbf{K}^T=\beta^2\mathbf{I}$)	β^2	0	$\surd$	$\surd$
Normalization				
Data Normalization($\mathbf{u}:=norm(\mathbf{y})$), $D[\mathbf{y}\in\mathbb{R}^{m\times 1}]=\sigma_B^2$	$\frac{1}{\sigma_B^2}$	$\frac{2}{m\sigma_B^4}$	$\surd$	$\times$

[1] The $\widetilde{k_hk_w}$ denotes the effective kernel size, which can be simply calculated from Algorithm 2.

Theorem 4.2. ***(Addition).*** *Given $\mathbf{J}:=\sum_i\mathbf{J_i}$, where $\{\mathbf{J_i}\}$ is a series of independent random matrices. If at most one matrix in $\{\mathbf{J_i}\}$ is not a central matrix (Definition 4.3), we have*

$$\phi\left(\mathbf{J}\mathbf{J}^T\right)=\sum_i\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right). \tag{15}$$

If $(\sum_i\mathbf{J_i})(\sum_i\mathbf{J_i})^T$ is at least the 2^{nd} moment unitarily invariant (Definition 4.2), and $\forall i,\mathbf{U_iJ_i}$ is R-diagonal (Definition 4.4), we have

$$\varphi\left(\mathbf{J}\mathbf{J}^T\right)=\phi^2\left(\mathbf{J}\mathbf{J}^T\right)+\sum_i\varphi\left(\mathbf{J_i}\mathbf{J_i}^T\right)-\phi^2\left(\mathbf{J_i}\mathbf{J_i}^T\right). \tag{16}$$

(Proof: Appendix A.3)

Definition 4.2 defines the rotational-invariant system described in Section 4.1, and Theorem 4.1 and 4.2 handle the serial and parallel connections in neural networks.

4.3 Discussion of Prerequisites

Although sufficient conditions of Theorem 4.1 and 4.2 are provided, it is still difficult to judge whether a series of Jacobian matrices satisfies them. In this section, we further provide a few sufficient conditions of Definition 4.2.

Definition 4.5. ***(Expectant Orthogonal Matrix)*** *A random matrix $\mathbf{J}$ is called an expectant orthogonal matrix if it satisfies: ① $\forall i,p\neq i,E[[\mathbf{J}^T\mathbf{J}]_{p,i}]=0$; ② $\forall i,j,E[[\mathbf{J}^T\mathbf{J}]_{i,i}]=E[[\mathbf{J}^T\mathbf{J}]_{j,j}]$.*

Proposition 4.1. *$(\Pi_{i=L}^1\mathbf{J_i})(\Pi_{i=L}^1\mathbf{J_i})^T$ is at least the 1^{st} moment unitary invariant if: ① $\forall i,j\neq i$, $\mathbf{J_i}$ is independent of $\mathbf{J_j}$; ② $\forall i\in[2,L]$, $\mathbf{J_i}$ is an expectant orthogonal matrix. (Proof: Appendix A.4)*

Remark 4.1. *With Proposition 4.1, as long as $\forall j,\mathbf{J_j}$ is an expectant orthogonal matrix, $(\Pi_{j=L}^{i+1}\mathbf{J_j})\frac{\partial\mathbf{f_i}}{\partial\theta_i}$ is the 1^{st} moment unitarily invariant. According to Theorem 4.1, as long as $(\Pi_{j=L}^{i+1}\mathbf{J_j})\frac{\partial\mathbf{f_i}}{\partial\theta_i}$ is the 1^{st} moment unitarily invariant, the decomposition in Equation* (8) *holds and Hypothesis 3.1 is confirmed.*

Proposition 4.2. ***(Properties of Expectant Orthogonal Matrices and Central Matrices)***

- *If $\{\mathbf{J_i}\}$ is a series independent expectant orthogonal matrices, $\Pi_i\mathbf{J_i}$ is also an expectant orthogonal matrix.*
- *If $\mathbf{J_i}$ is a central matrix, for any random matrix $\mathbf{A}$ independent of $\mathbf{J_i}$, $\mathbf{J_iA}$ and $\mathbf{AJ_i}$ are also central matrices.*

(Proof: Appendix A.5)

Proposition 4.1 and 4.2 are two sufficient conditions that allow us to judge whether a network structure satisfies the prerequisites by evaluating its components. For the 2^{nd} moment unitary invariant, we will discuss in specific cases when required. Notably, as the conditions we provided are sufficient but not necessary, Theorem 4.1 and 4.2 may still hold for networks that do not satisfy these conditions.

4.4 Components Library

We provide a library that summarizes some commonly used components in neural networks. We theoretically analyze the expectation and variance of their input-output Jacobian matrix $\mathbf{J}$'s eigenvalues as well as whether $\mathbf{J}$ satisfies Definition 4.5 and 4.3 of all these components under moderate assumptions. The detailed proofs are in Appendix A.6.

5 Serial Neural Networks

A serial neural network is the neural network whose components are connected in serial, such as LeNet [30] and VGG [31]. We have the following proposition for serial networks:

Proposition 5.1. *For any neural network, if it is composed of parts given in Table 3 and its Jacobian matrix can be calculated by $\mathbf{J}=(\Pi_i\mathbf{J_i})$, then $(\Pi_i\mathbf{J_i})(\Pi_i\mathbf{J_i})^T$ is at least the 1^{st} moment unitary invariant.*

Proof. According to Table 3, all the components satisfy Definition 4.5, so they are all expectant orthogonal matrices. Under the assumption that the Jacobian matrices of different components are independent, according to Proposition 4.1, we have $(\Pi_i\mathbf{J_i})(\Pi_i\mathbf{J_i})^T$ is at least the 1^{st} moment unitary invariant. □

Proposition 5.1 reflects that Equation (13) is applicable in this section. We will show that with our framework, the conclusions of several previous studies including initialization [1], [3], normalization [5], [6], [21], self-normalizing neural network [22] and DenseNet [9] can be easily reached or even surpassed with several lines of derivation.

5.1 Initialization Techniques

It has long been aware that a good initialization of network parameters can significantly improve the convergence and make deeper networks trainable [1], [2], [3], [32], [33]. In this subsection, we will discuss some of the most popular

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

initialization techniques. We consider a simple network block with a single linear transform (the weight kernel is $\mathbf{K} \in \mathbb{R}^{m \times n}$) and an activation function. The Jacobian matrix of the whole block is denoted as $\mathbf{J_i}$. Since the activation functions are commonly applied right after linear transforms, we assume that the mean of input pre-activations is zero, thus $p = P(x > 0) = 1/2$. Moreover, Equation (14) can be applied if the kernel follows i.i.d. Gaussian distribution.

Proposition 5.2. *A neural network is the ∞^{th} moment unitarily invariant if it is composed of cyclic central Gaussian transform with i.i.d. entries and any network components. (Proof: Appendix A.8)*

Kaiming Normal(KM) [1]. We denote the Jacobian matrix of the i^{th} layer as $\mathbf{J_i}$. With Equation (13)-(14), we have

$$\begin{aligned} \phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) &= n\sigma^2 \times \frac{1}{2} = \frac{1}{2}\sigma^2 n, \\ \varphi\left(\mathbf{J_i}\mathbf{J_i}^T\right) &= \phi^2(\mathbf{J_i}\mathbf{J_i}^T)\left(\frac{m}{m}\frac{\frac{1}{4}}{(\frac{1}{2})^2} + \frac{m}{m}\frac{mn\sigma^4}{(n\sigma^2)^2}\right). \end{aligned} \tag{17}$$

We can force $\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1$ to achieve the block dynamical isometry, which yields $\sigma = \frac{\sqrt{2}}{\sqrt{n}}$ and $\varphi(\mathbf{J_i}\mathbf{J_i}^T) = 1 + \frac{m}{n}$. For fully-connected layers, n denotes the width of the weight matrix; for convolutional layers, $n = c_{in}\widetilde{k_h k_w}$, and $c_{in} = 1$ for point-wise convolution [34]. Although using the effective kernel size $\widetilde{k_h k_w}$ provides more accurate estimation, we empirically find that replacing $k_h k_w$ with $\widetilde{k_h k_w}$ only has trifling impact on accuracy. The reason is that most of the feature maps are large enough, and the cutting-off effect caused by padding (see Appendix A.6) is less significant compared with other factors like parameter update. The optimal σ for other activation functions like leaky ReLU and tanh can be obtained in the same way, and we summarize the results in Table 4.

TABLE 4
Optimal σ for ReLU, leaky ReLU and tanh with Gaussian kernel.

	ReLU	leaky ReLU. γ: negative slope coefficient	tanh
$\phi(\mathbf{J_i}\mathbf{J_i}^T)$	$\frac{1}{2}\sigma^2 n$	$\frac{1}{2}\sigma^2 n(1+\gamma^2)$	$\approx \sigma^2 n$
Optimal σ	$\frac{\sqrt{2}}{\sqrt{n}}$	$\sqrt{\frac{2}{n(1+\gamma^2)}}$	$\frac{1}{\sqrt{n}}$
$\varphi(\mathbf{J_i}\mathbf{J_i}^T)$ under optimal σ	$1 + \frac{m}{n}$	$\left(\frac{1-\gamma^2}{1+\gamma^2}\right)^2 + \frac{m}{n}$	$\frac{m}{n}$

sPReLU. Instead using a fixed negative slope coefficient like leaky ReLU, PReLU [1] replaces it with a trainable parameter α. Although He et al. (2015) [1] initialize weights with $\frac{\sqrt{2}}{\sqrt{n}}$, we find it might be problematic when the network is deep: for an L-layer network, we have $\Pi_{i=L}^{1}\phi(\mathbf{J_i}\mathbf{J_i}^T) = (1+\alpha^2)^L$. He et al. (2015) [1] found that the learned α in some layers are significantly greater than 0, therefore the original setup may not be stable in relatively deep networks.

Since α is kept updating during training, it is difficult to address the above issue by initialization. So we modify PReLU by simply rescaling the output activations with $\frac{1}{\sqrt{1+\alpha^2}}$ as follows, which is named as "sPReLU":

$$sPReLU(x) = \frac{1}{\sqrt{1+\alpha^2}}\begin{cases} x & if \;\; x > 0 \\ \alpha x & if \;\; x \leq 0 \end{cases}. \tag{18}$$

With the rescaling, we have $\Pi_{i=L}^{1}\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1$. However, leaving α without any constraint may lead to an unstable training process, thus we clip it within $[0,\ 0.5]$.

Orthogonal Initialization. [3] Because our target is to have $\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1$ and $\varphi(\mathbf{J_i}\mathbf{J_i}^T) \approx 0$, one intuitive idea is to initialize $\mathbf{J_i}$ to have orthogonal rows. Proposition 5.3 demonstrates that Equation (14) is applicable for blocks consisting of orthogonal kernels and any activation functions.

Proposition 5.3. *A neural network block composed of an orthogonal transform layer and any activation function is at least the 2^{nd} moment unitarily invariant. (Proof: Appendix A.9)*

As illustrated in Table 3, from the perspective of ϕ, β^2 is equivalent with $n\sigma^2$, therefore we can easily obtain the optimal β for different activation functions.

TABLE 5
Optimal β for ReLU, leaky ReLU and tanh with orthogonal kernel.

	ReLU	leaky ReLU. γ: negative slope coefficient	tanh
Optimal β	$\sqrt{2}$	$\sqrt{\frac{2}{1+\gamma^2}}$	≈ 1
$\varphi(\mathbf{J_i}\mathbf{J_i}^T)$ under optimal β	1	$\left(\frac{1-\gamma^2}{1+\gamma^2}\right)^2$	≈ 0

Comparison. Table 4&5 show that with proper initialization, all the activation functions can achieve $\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1$, whereas their $\varphi(\mathbf{J_i}\mathbf{J_i}^T)$ are quite different. For example, ReLU has the highest φ, while tanh has the lowest with more stability. However, since rectifiers like ReLU have non-saturating property [35] and produce sparse representations [36], they are usually more effective than tanh. Besides, unlike rectifiers that preserve the forward and backward flows simultaneously [18], we find that the second moment of the forward information is diminished with tanh.

Leaky ReLU provides us an opportunity to trade off between stability and nonlinearity. Although its nonlinearity is the most effective when γ is around a certain value (i.e. $1/5.5$ [37]), a relatively greater γ can effectively reduce φ. However, the optimal γ has to be explored experimentally.

sPReLU has a similar effect with leaky ReLU, as argued in He et al. (2015) [1], it learns a greater α to keep more information in the first few layers, which provides more stability. In the later stage when the nonlinearity is required, the learned α is relatively small to preserve the nonlinearity.

The comparison of $\varphi(\mathbf{J_i}\mathbf{J_i}^T)$ under optimal initialization in Table 4&5 also indicates that the orthogonal initialization provides much lower φ compared with the Gaussian initialization, since the orthogonal kernel's φ is 0.

Relationship to Existing Studies. In some neural network structures, our theorems can even be used to analyze the information flow in the forward pass owing to Proposition 5.4 and 5.5.

Definition 5.1. ***(General Linear Transform)*** *Let $\mathbf{f}(\mathbf{x})$ be a transform whose Jacobian matrix is $\mathbf{J}$. $\mathbf{f}$ is called general linear transform when it satisfies:*

$$E\left[\frac{||\mathbf{f}(\mathbf{x})||_2^2}{len(\mathbf{f}(\mathbf{x}))}\right] = \phi\left(\mathbf{J}\mathbf{J}^T\right) E\left[\frac{||\mathbf{x}||_2^2}{len(\mathbf{x})}\right]. \tag{19}$$

Proposition 5.4. *Data normalization with 0-mean inputs, linear transforms and rectifier activation functions are general linear transforms (Definition 5.1). (Proof: Appendix A.10)*

Proposition 5.5. *For a serial neural network* $\mathbf{f}(\mathbf{x})$ *composed of general linear transforms and its input-output Jacobian matrix is* $\mathbf{J}$*, we have*

$$E\left[\frac{||\mathbf{f}(\mathbf{x})||_2^2}{len(\mathbf{f}(\mathbf{x}))}\right] = \phi\left(\mathbf{J}\mathbf{J}^T\right) E\left[\frac{||\mathbf{x}||_2^2}{len(\mathbf{x})}\right]. \tag{20}$$

(Proof: Appendix A.11)

According to Proposition 5.4, the rectifier activation functions and linear transforms in He et al. (2015) [1] are all general linear transforms. Therefore, Proposition 5.5 shows that $\phi(\mathbf{J}\mathbf{J}^T)$ also describes the evolution of the second moment/variance of activations in the forward pass, which is equivalent to He et al. (2015) [1] but more convenient.

5.2 Normalization Techniques

Even if the parameters in a neural network are properly initialized, there is no guarantee that their statistic properties remain unchanged during training, especially under high learning rate. To address this issue, normalization techniques are introduced to maintain the parameters' statistic properties during training.

Weight Normalization (WN) [6]. Let $\mathbf{W}$ denote the weight matrix, WN can be represented as $\hat{\mathbf{W}} = \frac{g}{||\mathbf{W}||}\mathbf{W}$, where g is a constant scaling factor and $\hat{\mathbf{W}}$ is what we use for training and inference. To further improve the performance, a mean-only batch normalization is usually applied [6]. Under this setup, the standard deviation of a normalized kernel is $\sigma_{\hat{\mathbf{W}}} = g$ and $\phi(\mathbf{J}\mathbf{J}^T) = ng^2$. Salimans & Kingma (2016) [6] take $g = e^s/\sqrt{n}$, which may not be the optimal setup of activation functions, for there is no guarantee that $\phi(\mathbf{J}\mathbf{J}^T) = e^{2s} \approx 1$. Therefore, it has been observed that WN is less stable in deep networks [25].

Scaled Weight Standardization (sWS). Inspired by WN, we propose a new weight-related normalization technique, which is defined as: $\hat{\mathbf{K}} = \frac{g}{\sigma_{\mathbf{K}}}(\mathbf{K} - \mu_{\mathbf{K}})$, where $\mu_{\mathbf{K}}$ and $\sigma_{\mathbf{K}}$ denote the kernel's mean and variance, respectively. Therefore, we have $\mu_{\hat{\mathbf{K}}} = 0$ and $\sigma_{\hat{\mathbf{K}}} = ng^2$, and the mean-only batch normalization is no longer required. As the most intuitive idea is to normalize the weights to "Kaiming Normal (KM)" during the training, the optimal g values for different activation functions are listed in Table 6.

TABLE 6
Optimal g for ReLU, leaky ReLU and tanh with sWS.

	ReLU	leaky ReLU. γ: negative slope coefficient	tanh
Optimal g	$\frac{\sqrt{2}}{\sqrt{n}}$	$\sqrt{\frac{2}{n(1+\gamma^2)}}$	$\frac{1}{\sqrt{n}}$

Similar conclusion for ReLU has been reached by Arpit et al. (2016) [21]. It proposes a scheme called Normprop which normalizes the hidden layers with theoretical estimation as follows:

$$o_i = \frac{1}{\sqrt{\frac{1}{2}(1-\frac{1}{\pi})}}\left[ReLU\left(\frac{\gamma_i \mathbf{W}_i^T \mathbf{x}}{||\mathbf{W_i}||_F} + \beta_i\right) - \sqrt{\frac{1}{2\pi}}\right], \tag{21}$$

where o_i denotes the i^{th} output, and γ_i and β_i are trainable parameters initialized with $1/1.21$ and 0, respectively. $\mathbf{W}_i$ is the weight corresponding to the i^{th} input $\mathbf{x}_i$. Because of $||\mathbf{W}_i||_F = \sqrt{n\sigma_W^2}$ and $\frac{1}{1.21\sqrt{\frac{1}{2}(1-\frac{1}{\pi})}} \approx 1.415 \approx \sqrt{2}$, the scaling of the weight is exactly the same with our derivation.

Data Normalization (DN) [5], [26], [38]. This has become a regular component of deep neural networks, for it enables us to train deeper networks, use large learning rates and apply arbitrary initialization schemes [5]. In DN, the pre-activations are normalized to $N(0,1)$ by

$$\hat{\mathbf{x}} = \frac{\mathbf{x} - E[\mathbf{x}]}{\sqrt{D[\mathbf{x}] + \epsilon}}, \quad \mathbf{y} = \gamma\hat{\mathbf{x}} + \beta. \tag{22}$$

DN can be explained by slightly extending Proposition 5.5.

Proposition 5.6. *We consider a serial network block composed of general linear transforms (Definition 5.1). The* 2^{nd} *moment of the block's input activation is* $\alpha_2^{(0)}$ *and the block's Jacobian matrix is* $\mathbf{J}$*. If the Jacobian matrix of its last component* $\mathbf{J_1}$ *satisfies* $\phi(\mathbf{J_1}\mathbf{J_1}^T) = \frac{\beta}{\alpha_2^{(l-1)}}$ *wherein* β *is a constant value and* $\alpha_2^{(l-1)}$ *is the* 2^{nd} *moment of its input data, then we have* $\phi(\mathbf{J}\mathbf{J}^T) = \frac{\beta}{\alpha_2^{(0)}}$*.*

Proof. Since the network is composed of general linear transforms, with Proposition 5.5, we have

$$\alpha_2^{(l-1)} = \Pi_{i=l-1}^{1}\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right)\alpha_2^{(0)}. \tag{23}$$

Therefore, we further have

$$\phi\left(\mathbf{J}\mathbf{J}^T\right) = \Pi_{i=l-1}^{1}\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right)\frac{\beta}{\Pi_{i=l-1}^{1}\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right)\alpha_2^{(0)}} = \frac{\beta}{\alpha_2^{(0)}}. \tag{24}$$

□

According to Ioffe & Szegedy (2015) [5], DN is performed right after the linear transforms, thus its inputs have zero-mean, and we further have $\sigma_B^2 = \alpha_{2,B}$. For instance, the input of the block shown in Fig. 2 is the output of a BN layer, therefore its 2^{nd} moment $\alpha_2^{(0)}$ is 1. With Proposition 5.6, we have $\phi(\mathbf{J}\mathbf{J}^T) = 1$, thus DN can effectively address gradient explosion or vanishing.

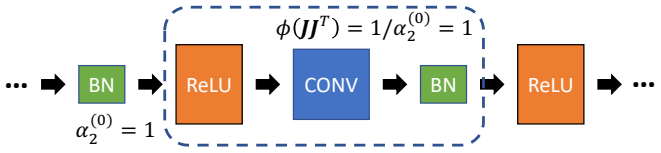


Fig. 2. Example block for Proposition 5.6.

Proposition 5.6 can be interpreted from another perspective. As illustrated in Equation (23), in a network composed of general linear transforms, the pre-activations' 2^{nd} moment $\alpha_2^{(l-1)}$ contains the information about the status of all layers it has passed through, and a network component with $\phi \propto \frac{1}{\alpha_2^{(l-1)}}$ can effectively offset the influence of these layers. This explains why DN techniques like BN are more stable with less awareness of the initialization and sustainable to the high learning rate.

Comparison. The common topic for all the normalization techniques is standardizing the 1^{st} and 2^{nd} moments of the pre-activations, and the only difference is what the moments are estimated upon. Specifically, DN gets its 1^{st} and

2^{nd} moments from the pre-activations, while WN estimates the 2^{nd} moment from the weight kernel. However, different sources of estimation will result in different execution efficiency, stability, and convenience.

For execution efficiency, as the weight kernels usually contain fewer data compared with pre-activations, estimating moments from the weight kernels usually has lower computational overhead. For stability, while WN depends on the "Gaussian Assumption", which is not necessarily held during training, Proposition 5.6 is valid for any linear transforms. Moreover, each WN sweeps the snow from its own doorstep, whereas DN improves the condition of all the layers before it, thus even if one or two DN layers malfunction, the following ones would compensate for them. For convenience, as WN is born out of weight initialization, its hyper-parameters require careful selection, which makes it less suitable for complex network structures. Oppositely, DN can automatically improve the network's condition without handcrafted hyper-parameters.

Second Moment Normalization (SMN). Inspired by the above comparison, we propose the last piece of puzzle of normalization methods: SMN, wherein the 1^{st} moment is obtained from the weight kernel while the 2^{nd} moment is estimated from the pre-activations. In SMN, the pre-activations are normalized by

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{\sqrt{E\left[[\mathbf{x}]_i^2\right]}}, \mathbf{y} = \gamma\hat{\mathbf{x}} + \beta. \tag{25}$$

Since our derivation is based on the assumption that the weight kernels have zero expectation, which may be violated during training, we further add weight centralization onto each weight:

$$\hat{\mathbf{K}} = \mathbf{K} - E[[\mathbf{K}]_i]. \tag{26}$$

For stability and convenience, similar to DN, we have

$$\phi\left(\hat{\mathbf{x}}_{\mathbf{x}}\hat{\mathbf{x}}_{\mathbf{x}}^T\right) \approx \frac{1}{\alpha_2^2}, \ \ \varphi\left(\hat{\mathbf{x}}_{\mathbf{x}}\hat{\mathbf{x}}_{\mathbf{x}}^T\right) \approx 0, \tag{27}$$

where $\hat{\mathbf{x}}_{\mathbf{x}}$ satisfies Definition 4.5 but defies Definition 4.3. The proof is in Appendix A.7. Because of $\phi(\hat{\mathbf{x}}_{\mathbf{x}}\hat{\mathbf{x}}_{\mathbf{x}}^T) \approx \frac{1}{\alpha_2^2}$, the 2^{nd} moment normalization can achieve similar effect with DN when applied right after linear transforms. Therefore, SMN is as stable and convenient as DN. For execution efficiency, we have

$$\begin{aligned} &SMN: \ \ \hat{\mathbf{x}} = \frac{\mathbf{x}}{\sqrt{E\left[[\mathbf{x}]_i^2\right]}}, \ \ \hat{\mathbf{K}} = \mathbf{K} - E[[\mathbf{K}]_i], \\ &WN: \ \ \hat{\mathbf{x}} = \mathbf{x} - E\left[[\mathbf{x}]_i\right], \ \ \hat{\mathbf{K}} = \frac{\mathbf{K}}{E[[\mathbf{K}]_i^2]]}. \end{aligned} \tag{28}$$

SMN can be viewed as a reversed version of WN, and there is only one additional element-wise square operator compared with WN, so it has fewer computational overhead than DN. Following the analysis in Chen et al. (2019) [24], in Appendix A.15, we find that SMN has 30% fewer computation overhead than BN.

We provide the detailed algorithm for SMN in the convolutional layer in Algorithm 1. Inspired by Ioffe & Szegedy (2015) [5], we centralize the mean of the weight kernels of each output channel rather than shifting the mean of of the whole weight to zero. Similarly, the 2^{nd} moment is also standardized in a channel-wise manner. Also, the trainable parameters γ and β in BN are introduced to represent the identity transform [5]. Besides the 2^{nd} moment, according to prior work [39], we can also use L1-norm to further reduce the complexity:

$$\hat{\mathbf{x}} = \frac{\mathbf{x}}{E\left[|[\mathbf{x}]_i|\right]}, \ \ \hat{\mathbf{K}} = \mathbf{K} - E[[\mathbf{K}]_i]. \tag{29}$$

Although our SMN can statistically replace BN, it somehow has weaker regularization ability, because estimating the 1^{st} moment from pre-activations introduces Gaussian noise that can regularize the training process [40]. Fortunately, this can be compensated by addition regularization like mixup [23].

Algorithm 1: Second Moment Normalization

Data: Input pre-activation $\mathbf{x} \in [batch\_size, c_{in}, H_i, W_i]$; Convolving kernel: $\mathbf{K} \in [c_{out}, c_{in}, h, w]$; Scaling factor $\gamma \in [c_{out}]$; Bias $\beta \in [c_{out}]$

Result: Normalized pre-activation $y \in [batch\_size, c_{out}, H_o, W_o]$;

begin

$\mu_K = mean(\mathbf{K}[c_{out}, :])$

$\hat{\mathbf{K}} = \mathbf{K} - \mu_K$ //weight centralization

$\mathbf{x} = \hat{\mathbf{K}} * \mathbf{x}$

$\alpha_2 = mean(square(\mathbf{x})[c_{out}, :])$

$\mathbf{y} = \beta + \frac{\gamma}{\sqrt{\alpha_2}}\mathbf{x}$

return $\mathbf{y}$

5.3 Self-Normalizing Neural Network

Klambauer et al. (2017) [22] propose a self-normalizing property empowered by SeLU activation function given by

$$SeLU(x) = \lambda \begin{cases} x & if \ \ x > 0 \\ \alpha e^x - \alpha & if \ \ x \leq 0 \end{cases} \tag{30}$$

where $\alpha \approx 1.6733$, $\lambda \approx 1.0507$. However, the setup in [22] only works for weights whose entries follow $N(0, \frac{1}{n})$. Here we generally let the linear transform have $\phi(\mathbf{J}\mathbf{J}^T) = \gamma_0$.

Proposition 5.7. *Let* $\mathbf{J}$ *be the Jacobian matrix of SeLU. When the pre-activations obey* $N(0, \sigma^2)$*, we have the following conclusions:*

$$\begin{aligned} &\phi\left(\mathbf{J}\mathbf{J}^T\right) = \lambda^2\alpha^2 e^{2\sigma^2} cdf(-2\sigma^2, N(0, \sigma^2)) + \frac{\lambda^2}{2}, \\ &E[SeLU^2(\mathbf{x})] = \frac{1}{2}\lambda^2\sigma^2 + \frac{1}{2}\lambda^2\alpha^2 + \\ &\lambda^2\alpha^2\left(e^{2\sigma^2} cdf(-2\sigma^2, N(0, \sigma^2)) - 2e^{\frac{\sigma^2}{2}} cdf(-\sigma^2, N(0, \sigma^2))\right), \\ &E[SeLU(\mathbf{x})] = \lambda\alpha e^{\frac{\sigma^2}{2}} cdf(-\sigma^2, N(0, \sigma^2)) - \frac{\lambda\alpha}{2} + \sqrt{\frac{\sigma^2}{2\pi}}\lambda. \end{aligned} \tag{31}$$

(Proof: Appendix A.12)

Let SeLU be applied layer-wisely and the 2^{nd} moment of output activations have a fixed point of 1. With Proposition

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

5.5, the variance of pre-activations equals γ_0. Then, with Proposition 5.7, the optimal α and λ can be solved from

$$\begin{aligned}&\left(\lambda^2\alpha^2 e^{2\gamma_0} cdf(-2\gamma_0, N(0,\gamma_0)) + \frac{\lambda^2}{2}\right)\gamma_0 = 1+\epsilon,\\ &\lambda^2\alpha^2\left(e^{2\gamma_0} cdf(-2\gamma_0, N(0,\gamma_0)) - 2e^{\frac{\gamma_0}{2}} cdf(-\gamma_0, N(0,\gamma_0))\right)\\ &+\frac{1}{2}\lambda^2\alpha^2 + \frac{1}{2}\lambda^2\gamma_0 = 1.\end{aligned} \tag{32}$$

The former equation constrains $\phi(\mathbf{J_i}\mathbf{J_i}^T) \approx 1$ and the latter one ensures the fixed point of the 2^{nd} moment. ϵ is a small constant near 0, which prevents SeLU from degenerating back to ReLU. When $\epsilon = 0, \gamma_0 = 1$, the only solution of Equation (32) is $\lambda = \sqrt{2}, \alpha = 0$, which is equivalent to the KM initialization with ReLU. One explanation is that if $\epsilon = 0$, we would have $\alpha_2(x_{out})/\alpha_2(x_{in}) = \phi(\mathbf{J}\mathbf{J}^T)$, which is only held when the network satisfies Proposition 5.5. Notably, the original SeLU in [22] can be solved from Equation (32) by letting $\gamma_0 = 1, \epsilon \approx 0.0716$.

Although $\phi(\mathbf{J_i}\mathbf{J_i}^T) \approx 1$ can be achieved from multiple initialization schemes, SeLU's strength comes from its attractive fixed point [22], which is effective even when the assumptions and initial statistic properties are violated. However, this attractive property takes over 80-page proofs in [22], so it is challenging to extend to more general situation. In this work, we provide an empirical understanding by analogizing it with data normalization.

In Proposition 5.6, we demonstrate that a network component with $\phi(\mathbf{J_l}\mathbf{J_l}^T) = \frac{\beta}{\alpha_2^{(l-1)}}$ can stabilize the general linear network block based on the information contained in $\alpha_2^{(l-1)}$, here we discuss a more general situation in which $\phi(\mathbf{J_l}\mathbf{J_l}^T) = h_l(\alpha_2^{(l-1)})$ where h_l is a real function. We further assume that the network component satisfies Definition 5.1. When $h_l(\alpha_2^{(l-1)})$ satisfies

$$\begin{aligned}1 < h_l(\alpha_2^{(l-1)}) < \frac{\beta}{\alpha_2^{(l-1)}}, \quad if\ \ \alpha_2^{(l-1)} < \beta;\\ 1 > h_l(\alpha_2^{(l-1)}) > \frac{\beta}{\alpha_2^{(l-1)}}, \quad if\ \ \alpha_2^{(l-1)} > \beta.\end{aligned} \tag{33}$$

Since $\Pi_{i=l}^1\phi(\mathbf{J_i}\mathbf{J_i}^T) = h_l(\alpha_2^{(l-1)})\alpha_2^{(l-1)}/\alpha_2^{(0)}$, we have

$$\left|\Pi_{i=l-1}^1\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) - \frac{\beta}{\alpha_2^{(0)}}\right| > \left|\Pi_{i=l}^1\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) - \frac{\beta}{\alpha_2^{(0)}}\right|, \tag{34}$$

which illustrates that $\Pi_i^1\phi(\mathbf{J_i}\mathbf{J_i}^T)$ converges to the fixed point of $\frac{\beta}{\alpha_2^{(0)}}$. As the convergence may take several layers, we call it as "partial normalized". Similarly, when $\forall\alpha_2^{(l-1)}$, $h_l(\alpha_2^{(l-1)})$ satisfies

$$\begin{aligned}h_l(\alpha_2^{(l-1)}) > \frac{\beta}{\alpha_2^{(l-1)}}, \quad if\ \ \alpha_2^{(l-1)} < \beta;\\ 0 < h_l(\alpha_2^{(l-1)}) < \frac{\beta}{\alpha_2^{(l-1)}}, \quad if\ \ \alpha_2^{(l-1)} > \beta,\end{aligned} \tag{35}$$

we have

$$\left(\Pi_{i=l}^1\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) - \frac{\beta}{\alpha_2^{(0)}}\right)\left(\Pi_{i=l-1}^1\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) - \frac{\beta}{\alpha_2^{(0)}}\right) < 0. \tag{36}$$

$\Pi_i^1\phi(\mathbf{J_i}\mathbf{J_i}^T)$ swings around the fixed point of $\frac{\beta}{\alpha_2^{(0)}}$ but there is no guarantee for convergence, so we name its as "over normalized".

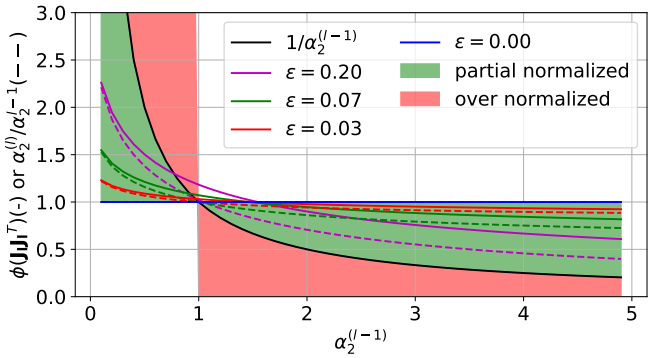


Fig. 3. $\phi(\mathbf{J_l}\mathbf{J_l}^T)$ (the solid line) and $\alpha_2^{(l)}/\alpha_2^{(l-1)}$ (the dashed line) of SeLU under different ϵ. We have $\gamma_0 = 1$ for all the configurations.

For SeLU, we have $\alpha_2^{(0)} = \beta = 1$, and we plot $\phi(\mathbf{J_l}\mathbf{J_l}^T) \sim \alpha_2^{(l-1)}$ and $\alpha_2^{(l)}/\alpha_2^{(l-1)} \sim \alpha_2^{(l-1)}$ of different configurations in Fig. 3. It shows that 1) when ϵ is relatively small, we have $\phi(\mathbf{J_l}\mathbf{J_l}^T) \approx \alpha_2^{(l)}/\alpha_2^{(l-1)}$, and SeLU can be seen as a general linear transform; 2) when $\epsilon > 0$, $\phi(\mathbf{J_l}\mathbf{J_l}^T)$ is in the "partial normalized" region, which suggests that it will take a few layers to converge to a fixed point. Moreover, the $\phi(\mathbf{J_l}\mathbf{J_l}^T)$ of the configurations with greater ϵ is closer to $\frac{1}{\alpha_2^{(l-1)}}$, leading to faster convergence; whereas a too large ϵ will result in gradient explosion, because of $\Pi_{i=L}^1\phi(\mathbf{J_i}\mathbf{J_i}^T) = (1+\epsilon)^L$. For a neural network with finite depth, we have

$$(1+\epsilon)^L = 1 + L\epsilon + \sum_{i=2}^{L} C_L^i\epsilon^i. \tag{37}$$

As a result, taking $\epsilon < \frac{1}{L}$ can effectively constrain the gradient norm while maintaining good normalization efficiency.

5.4 Shallow Network Trick

Let's consider a neural network with sequential blocks:

$$\mathbf{f}(\mathbf{x_0}) = \mathbf{f_{L,\theta_L}} \circ \mathbf{f_{L-1,\theta_{L-1}}} \circ ... \circ \mathbf{f_{1,\theta_1}}(\mathbf{x_0}), \tag{38}$$

and the Jacobian matrix of the i^{th} block is $\mathbf{J_i}$. We assume that $\mathbf{J} = \Pi_{i=L}^1\mathbf{J_i}$ is at least the 1^{st} moment unitarily invariant (Definition 4.2). With Theorem 4.1, we have $\phi(\mathbf{J}\mathbf{J}^T) = \Pi_i\phi(\mathbf{J_i}\mathbf{J_i}^T)$. In order to prevent the gradient explosion or vanishing, we expect $\forall i, \phi(\mathbf{J_i}\mathbf{J_i}^T) \approx 1$, which can be achieved with all the techniques discussed above. However, it might be influenced by many factors including the update of parameters under a large learning rate, invalid assumptions or systematic bias (like the cutting-off effect of padding), thus the actual $\phi(\mathbf{J_i}\mathbf{J_i}^T)$ can be represented as $1+\gamma_i$ and $\phi(\mathbf{J}\mathbf{J}^T)$ can be $\Pi_{i=1}^L(1+\gamma_i)$. Even if γ_i for single layer is small enough, when L is large, the influence of single γ_i might accumulate and result in gradient explosion or vanishing. As a result, techniques like initialization, weight standardization and SeLU are less stable under large learning rates in deep networks. Fortunately, this can be addressed by the following proposition.

Proposition 5.8. ***(Shallow Network Trick).*** *Assuming that for each of L sequential blocks in a neural network, we have*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

$\phi(\mathbf{J_i}\mathbf{J_i}^T) = \omega + \tau\phi(\widetilde{\mathbf{J}_\mathbf{i}}\widetilde{\mathbf{J}_\mathbf{i}}^T)$ *where* $\mathbf{J_i}$ *is its Jacobian matrix. Given* $\lambda \in \mathbb{N}^+ < L$*, if* $C_L^\lambda(1-\omega)^\lambda$ *and* $C_L^\lambda\tau^\lambda$ *are small enough, the network would be as stable as a* λ*-layer network when both networks have* $\forall\ i,\ \phi(\mathbf{J_i}\mathbf{J_i}^T) \approx 1$.

Proof. Because of $\phi(\mathbf{J_i}\mathbf{J_i}^T) = \omega + \tau\phi(\mathbf{J_i}\mathbf{J_i}^T)$, the optimal $\phi(\mathbf{J_i}\mathbf{J_i})$ is $\frac{1-\omega}{\tau}$. We consider both the absolute and relative errors of $\phi(\mathbf{J_i}\mathbf{J_i})$ by representing it as $\frac{1-\omega}{\tau}(1+\gamma_i)$ and $\frac{1-\omega}{\tau}+\delta_i$, respectively. For both kinds of error, we have

$$\begin{aligned}\phi\left(\mathbf{J}\mathbf{J}^T\right) &= \Pi_{i=1}^L(1+(1-\omega)\gamma_i) = 1+\sum_{i=1}^L C_L^i(1-\omega)^i\Pi_j\gamma_j,\\ \phi\left(\mathbf{J}\mathbf{J}^T\right) &= \Pi_{i=1}^L(1+\tau\delta_i) = 1+\sum_{i=1}^L C_L^i\tau^i\Pi_j\delta_j.\end{aligned} \tag{39}$$

When $\omega \to 1^-$, we have $\tau \to 0^+$, $\lim_{i\to\infty}(1-\omega)^i = \lim_{i\to\infty}\tau^i = 0$, and the error would diminish as i is large.

Here we borrow the concept of effective depth proposed in Philipp et al. (2018) [17]: assuming that $C_L^i(1-\omega)^i\Pi_j\gamma_j$ and $C_L^i\tau^i\Pi_j\delta_j$ are neglectable when $i > \lambda, \lambda < L$, all the errors are only influential within λ layers, thus it would be as stable as a λ-layer shallow network. □

5.5 DenseNet

We denote the activations as $\mathbf{x_i} \in \mathbb{R}^{c_i s_{fm}\times 1}$ where c_i is the number of channels and s_{fm} is the size of feature maps, and denote $\delta_i = c_i - c_{i-1}$. In DenseNet [9], the output of each layer within a dense block is concatenated with the input on the channel dimension to create dense shortcut connections, which is illustrated as follows:

$$\mathbf{x}_i = [\mathbf{x}_{i-1}, \mathbf{H}_i(\mathbf{x}_{i-1})],\ \ \frac{\partial\mathbf{x}_i}{\partial\mathbf{x}_{i-1}} = \begin{bmatrix}\mathbf{I}\\ \mathbf{H}_i\end{bmatrix} := \mathbf{J_i}, \tag{40}$$

where $\mathbf{H}_i \in \mathbb{R}^{\delta_i s_{fm}\times c_{i-1}s_{fm}}$, $\mathbf{I} \in \mathbb{R}^{c_{i-1}s_{fm}\times c_{i-1}s_{fm}}$. Since

$$\mathbf{J_i}^T\mathbf{J_i} = \begin{bmatrix}\mathbf{I} & \mathbf{H}_i^T\end{bmatrix}\begin{bmatrix}\mathbf{I}\\ \mathbf{H}_i\end{bmatrix} = \left[\mathbf{I}+\mathbf{H}_i^T\mathbf{H}_i\right] \tag{41}$$

and $\mathbf{H}_l$ is composed of the parts defined in Table 3, with Proposition 4.2, the non-diagonal entries of $\mathbf{I}+(\mathbf{H}_l)^T\mathbf{H}_l$ have a zero expectation while the diagonal entries share an identical expectation, and $\mathbf{J_l}$ satisfies Proposition 4.1. Therefore, $\phi(\mathbf{J_i}\mathbf{J_i}^T)$ can be calculated by

$$\phi\left(\mathbf{J_i}\mathbf{J_i}^T\right) = \frac{c_{i-1}}{c_i} + \frac{\delta_i}{c_i}\phi\left(\mathbf{H_i}\mathbf{H_i}^T\right). \tag{42}$$

As a result, in order to achieve block dynamical isometry, we expect $\phi(\mathbf{H_i}\mathbf{H_i}^T) \approx 1$, which can be achieved with the methods discussion in previous subsections. We will evaluate some configurations in Section 7.2. Equation (42) also reveals that DenseNet is an instance of the shallow network trick (Proposition 5.8), thus it is more stable compared with vanilla serial neural networks under the same depth.

6 Serial-parallel hybrid Networks

Serial-parallel hybrid networks consist of a sequence of blocks connected in serial, while each block may be composed of several parallel branches. Famous serial-parallel hybrid networks include Inception [41], ResNet [8], and NASNet [42]. With Proposition 5.1, as long as the input-output Jacobian matrices of all the blocks satisfy Definition 4.5, the network is at least the 1^{st} moment unitary invariant.

Proposition 6.1. *Let* $\{\mathbf{J_i}\}$ *denote a group of independent input-output Jacobian matrices of the parallel branches of a block.* $\sum_i \mathbf{J_i}$ *is an expectant orthogonal matrix, if it satisfies: 1)* $\forall i$, $\mathbf{J_i}$ *is an expectant orthogonal matrix; 2) at most one matrix in* $\{\mathbf{J_i}\}$ *is not central matrix. (Proof: Appendix A.13)*

According to Proposition 4.2, as long as each branch is composed of the parts in Table 3 and at most one branch does not contain a zero-mean linear transform, with Proposition 6.1, the series-parallel hybrid network is at least the 1^{st} moment unitarily invariant.

ResNet [8] is one of the most popular network structures that can avoid gradient explosion and vanishing, it is also the simplest serial-parallel hybrid network. The Jacobian matrix of each residual block is $\mathbf{J_i} = \mathbf{I} + \widetilde{\mathbf{J}_\mathbf{i}}$, with Equation (15), we have

$$\phi\left(\mathbf{J_i^{(l)}}\mathbf{J_i^{(l)}}^T\right) = 1 + \phi\left(\widetilde{\mathbf{J_i^{(l)}}}\widetilde{\mathbf{J_i^{(l)}}}^T\right). \tag{43}$$

From the above equation, ResNet can be viewed as an extreme example of the shallow network trick (Proposition 5.8) wherein $(1-\omega) \to 0$. As a result, its extremely low effective depth provides higher stability.

Data Normalization in ResNet. The 2^{nd} moment of the activations of ResNet with BN does not stay at a fixed point but keeps increasing through the layers [4]. Let's consider a ResNet whose l^{th} block is represented as $\mathbf{x_{l+1}} = BN(\mathbf{f}(\mathbf{x_l})) + \mathbf{x_l}$. Since the 2^{nd} moment of BN's output is 1, under the assumption that the outputs of the major branch and the shortcut branch are independent, we have $\alpha_2^{(l+1)} = 1 + \alpha_2^{(l)}$. At the down-sampling layers, since the shortcut connection is also handled by BN, $\alpha_2^{(l+1)}$ would be reset to 2. We denote the Jacobian matrix of the major branch as $\widetilde{\mathbf{J}^{(l)}}$, with Proposition 5.5, it's easy to obtain $\phi(\widetilde{\mathbf{J}^{(l)}}\widetilde{\mathbf{J}^{(l)}}^T) = \frac{1}{\alpha_2^{(l-1)}}$, and then we have

$$\begin{aligned}\phi\left(\mathbf{J_i^{(l+1)}}\mathbf{J_i^{(l+1)}}^T\right) &= 1+\frac{1}{\alpha_2^l} = \frac{\alpha_2^{(l+1)}}{\alpha_2^{(l)}},\\ \Pi_{i=L}^l\phi\left(\mathbf{J_i^{(i)}}\mathbf{J_i^{(i)}}^T\right) &= \frac{\alpha_2^{(L)}}{\alpha_2^{(l-1)}}.\end{aligned} \tag{44}$$

As the 2^{nd} moment of the activations in ResNet linearly rather than exponentially increases, and such an increasing is periodically stopped by down-sampling. Thus with Equation (44), gradient explosion or vanishing will not happen in ResNet when the depth is finite.

Fixup Initialization [4]. Without loss of generality, we consider a ResNet consisting of L residual blocks, wherein each block has m convolutional layers activated by ReLU. The feature maps are down-sampled for d times throughout the network. We assume that the convolutional layers are properly initialized such that $\phi(\mathbf{J_i^{(c)}}\mathbf{J_i^{(c)}}^T) = \alpha$, and the convolutional layers in the down-sampling shortcuts are initialized to have $\phi(\mathbf{J_i^{(c)}}\mathbf{J_i^{(c)}}^T) = \alpha_d$. For a single block whose number of input channels equals the number of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

output channels, we have $\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1 + (\frac{\alpha}{2})^m$; for the down-sampling block, we have $\phi(\mathbf{J_i}\mathbf{J_i}^T) = \alpha_d + (\frac{\alpha}{2})^m$. When L is finite, we have the following proposition:

Proposition 6.2. ***("Plus One" Trick).*** *Assume that for each of the L sequential blocks of a series-parallel hybrid neural network, we have $\phi(\mathbf{J_i}\mathbf{J_i}^T) = 1 + \phi(\widetilde{\mathbf{J}}_\mathbf{i}\widetilde{\mathbf{J}}_\mathbf{i}^T)$ where $\mathbf{J_i}$ is its Jacobian matrix. The network has gradient norm equality as long as*

$$\phi\left(\widetilde{\mathbf{J}}_\mathbf{i}\widetilde{\mathbf{J}}_\mathbf{i}^T\right) = O(\frac{1}{L^p}), p > 1. \tag{45}$$

(Proof: Appendix A.14)

As a result, it is optimal to have $\alpha_d = 1, \alpha = 2L^{-\frac{p}{m}}, p > 1$. For Gaussian weights, we can initialize the weights with $N(0, L^{-p/m}\frac{2}{n})$. As KM initializes the weights to $N(0, \frac{2}{n})$, the Fixup initialization is just equivalent to scaling the weights initialized with KM by $L^{-p/2m}$; for orthogonal weights, we have $\beta = L^{-p/2m}\sqrt{2}$. For the down-sampling convolutions, it should be initialized to have Gaussian weights with $N(0, \frac{1}{n})$ or orthogonal weights with $\beta = 1$.

Zhang et al. (2019) [4] observe that although ResNet with the Fixup initialization can achieve gradient norm equality, it does not regularize the training as BN does. To solve this problem, additional scalar multiplier and bias are added before each convolution, linear, and element-wise activation layer. The multipliers and biases are trainable under a learning rate of $1/10$ to improve stability. Moreover, further regularization like mixup [23] is used. Although we reach the same conclusion claimed in [4], our derivation is much simpler owing to the highly modularized framework.

7 Experiments

In this section, we first verify the correctness of our key theorems: Theorem 4.1 and 4.2 with numerical experiments in Section 7.1. Then, in Section 7.2, we perform extensive experiments to support our conclusions in previous sections on CIFAR-10. In Section 7.3, we further test several methods that yield interesting results in Section 7.2 on ImageNet. At last, in Section 7.4, we compare the memory and computation overhead of different methods.

7.1 Numerical Experiments

We use a simple fully-connected layer with ReLU as the basic building block. The entries in the weight follows i.i.d. $N(0, \sigma_i^2)$ and the entries of input features follow i.i.d. $N(\mu, \sigma^2)$. For Theorem 4.1, the building blocks are connected in serial. With Proposition 5.2, such a network certainly satisfies the prerequisites. For Theorem 4.2, the blocks are connected in parallel. As Central i.i.d. Gaussian matrices are asymptotically R-diagonal (Equation 4.45 in Cakmak (2012) [29]) and with Theorem 32 in Cakmak (2012) [29], all the blocks of the given network are R-diagonal. The networks are determined by a joint state $[\{m_i\}, \{\sigma_i\}, \mu, \sigma, N]$. The N denotes the total number of building blocks, and m is the input dimension of each block.

We repeat the experiment for 100 times and the joint state for Theorem 4.1 and 4.2 are uniformly drawn from $[\{U_i(1000, 5000)\}, \{U(0.1, 5)\}, U(-5, 5), U(0.1, 5), U_i(2, 20)]$, where $U(a, b)$ denote the uniform distribution within $[a, b]$, and $U_i(a, b)$ represent the discrete uniform distribution on integers from a to b. We denote the input-output Jacobian matrix of the whole network as $\mathbf{J}$, and we evaluate our theorems by measuring how well $\left(\phi(\mathbf{J}\mathbf{J}^T)/\phi(\mathbf{J}\mathbf{J}^T)_t, \varphi(\mathbf{J}\mathbf{J}^T)/\varphi(\mathbf{J}\mathbf{J}^T)_t\right)$ concentrates around $(1, 1)$, where $\phi(\mathbf{J}\mathbf{J}^T), \varphi(\mathbf{J}\mathbf{J}^T)$ are directly calculated from the defined Jacobian matrices while $\phi(\mathbf{J}\mathbf{J}^T)_t, \varphi(\mathbf{J}\mathbf{J}^T)_t$ are theoretical values.

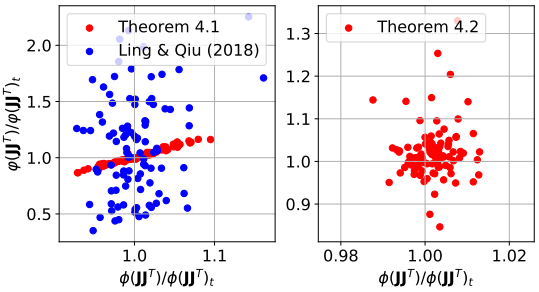


Fig. 4. Verification of Theorem 4.1 and 4.2. Each point denotes the result of one experiment.

The results are shown in Fig. 4. We can see that despite the numerical error, the experiment results well concentrate around $(1, 1)$. Besides, while the Result 2 in Ling & Qiu (2018) [16] is quite similar to our Theorem 4.1, their result can only handle the situations when the input and output feature map sizes are equal. Note that the estimation error of $\varphi(\mathbf{J}\mathbf{J}^T)$ with the theory in [16] is much greater than ours.

7.2 Experiments on CIFAR-10

Here we validate the conclusions yielded by our theorems on CIFAR-10 classification. The basic models we use are shown in Table 7, where "[]" denotes a vanilla network block, "()" denotes a network block with shortcut connection, and "{}" denotes a dense block whose major branch's output is concatenated with its input in the channel dimension. The shortcut connections in down-sampling layers are handled by average pooling and zero padding following Zhang et al. (2019) [4]. All the models are trained with a batch size of 128. We use SGD as the optimizer with momentum=0.9 and weight decay=0.0005. Besides, we clip the gradient within $[-2, 2]$ for all the experiments to increases the stability.

For all the experiments of serial networks except for DenseNet, the "serial network" in Table 7 is applied, which is equivalent to a ResNet-32 without shortcut connections. The models are trained for 130 epochs. The initial learning rate is set to 0.01 and decayed to 0.001 at epoch 80. For experiments on DenseNet, the models are trained for 130 epochs. The initial learning rate is set to 0.1 and decayed by $10\times$ at epoch 50, 80. For experiments on ResNet, we follow the configuration in Zhang et al. (2019) [4], i.e. all the models are trained for 200 epochs with an initial learning rate of 0.1 that is decayed by 10 at epoch 100, 150.

To support our conclusions, we evaluate all the configurations from two perspectives: module performance (test accuracy) and gradient norm distribution. Each configuration is trained from scratch 4 times to reduce the random variation and the test accuracy is averaged among the last 10 epochs. The gradient norm of each weight is represented by the L_2 norm of the weights' gradient, $||\mathbf{\Delta}\theta_\mathbf{i}||_2^2/\eta^2$, which is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 7
Network Structures for CIFAR-10.

	Out Size	Serial Network	ResNet	DenseNet
conv1	32×32	$3 \times 3, 16, s\ 1$		$3 \times 3, 24, s\ 1$
block1	32×32	$\begin{bmatrix} 3 \times 3, 16, s\ 1 \\ 3 \times 3, 16, s\ 1 \end{bmatrix} \times 5$	$\begin{pmatrix} 3 \times 3, 16, s\ 1 \\ 3 \times 3, 16, s\ 1 \end{pmatrix} \times 9$	$\begin{Bmatrix} 1 \times 1, 48, s\ 1 \\ 3 \times 3, 12, s\ 1 \end{Bmatrix} \times 8$
ds1	16×16	$\begin{bmatrix} 3 \times 3, 32, s\ 2 \\ 3 \times 3, 32, s\ 1 \end{bmatrix} \times 1$	$\begin{pmatrix} 3 \times 3, 32, s\ 2 \\ 3 \times 3, 32, s\ 1 \end{pmatrix} \times 1$	$1 \times 1, 60, s\ 2$
block2	16×16	$\begin{bmatrix} 3 \times 3, 32, s\ 1 \\ 3 \times 3, 32, s\ 1 \end{bmatrix} \times 4$	$\begin{pmatrix} 3 \times 3, 32, s\ 1 \\ 3 \times 3, 32, s\ 1 \end{pmatrix} \times 8$	$\begin{Bmatrix} 1 \times 1, 48, s\ 1 \\ 3 \times 3, 12, s\ 1 \end{Bmatrix} \times 8$
ds2	8×8	$\begin{bmatrix} 3 \times 3, 64, s\ 2 \\ 3 \times 3, 64, s\ 1 \end{bmatrix} \times 1$	$\begin{pmatrix} 3 \times 3, 64, s\ 2 \\ 3 \times 3, 64, s\ 1 \end{pmatrix} \times 1$	$1 \times 1, 78, s\ 2$
block3	8×8	$\begin{bmatrix} 3 \times 3, 64, s\ 1 \\ 3 \times 3, 64, s\ 1 \end{bmatrix} \times 4$	$\begin{pmatrix} 3 \times 3, 64, s\ 1 \\ 3 \times 3, 64, s\ 1 \end{pmatrix} \times 8$	$\begin{Bmatrix} 1 \times 1, 48, s\ 1 \\ 3 \times 3, 12, s\ 1 \end{Bmatrix} \times 8$
	1×1	average pooling, 10-d fc, softmax		

collected from the first 3 epochs (1173 iterations). For clarity, we color the range from 15 percentile to 85 percentile and represent the median value with a solid line.

Initialization in Serial Network. To support our conclusions in Section 5.1, we evaluate the initialization techniques in a 32-layer serial network on CIFAR-10. The test accuracy of all configurations is summarized in Table 8, and the gradient distribution is illustrated in Fig. 5. We evaluates two kinds of orthogonal initialization strategies: the orthogonal initialization [1] in Saxe et al. (2013) [28] and the delta orthogonal initialization [2] in Xiao et al. (2018) [3].

TABLE 8
Test accuracy of initialization techniques on CIFAR-10 with different activation functions and configurations (CI=95%).

Activation Function	Approach	Test Acc.
tanh	BN	$\mathbf{85.77\% \pm 0.77\%}$
	KM	$83.33\% \pm 1.02\%$
	Orth	$83.13\% \pm 0.54\%$
	Delta Orth [3]	$83.31\% \pm 0.38\%$
ReLU	BN	$\mathbf{88.70\% \pm 0.31\%}$
	KM	$85.13\% \pm 1.35\%$
	Orth	$85.53\% \pm 0.64\%$
	Delta Orth	$86.10\% \pm 1.33\%$
lReLU,$\gamma = 0.18$	BN	$\mathbf{89.19\% \pm 0.41\%}$
	KM	$87.96\% \pm 1.09\%$
	Orth	$88.51\% \pm 0.37\%$
	Delta Orth	$87.97\% \pm 1.34\%$
lReLU,$\gamma = 0.3$	BN	$89.58\% \pm 0.51\%$
	Orth	$89.24\% \pm 0.44\%$
	Delta Orth	$\mathbf{90.12\% \pm 0.64\%}$
lReLU,$\gamma = 0.5$	BN	$88.60\% \pm 0.34\%$
	Orth	$88.91\% \pm 0.27\%$
	Delta Orth	$\mathbf{89.53\% \pm 0.32\%}$
PReLU [1]	BN	$\mathbf{88.96\% \pm 0.35\%}$
	KM	$88.11\% \pm 0.99\%$
	Orth	$87.39\% \pm 3.06\%$
	Delta Orth	$82.00\% \pm 7.39\%$
sPReLU (ours)	BN	$88.96\% \pm 0.35\%$
	KM	$88.87\% \pm 0.32\%$
	Orth	$89.16\% \pm 0.32\%$
	Delta Orth	$\mathbf{89.73\% \pm 0.34\%}$

To begin with, as illustrated in Fig. 5, the gradient distributions of all configurations with tanh, ReLU, leaky ReLU and sPReLU are more or less neutral, and Table 8 shows that all these configurations can converge, which demonstrates the effectiveness of the initialization schemes under relatively deep network and moderate learning rate.

1. pytorch.org/docs/stable/nn.init.html#torch.nn.init.orthogonal_
2. We use the implementation for orthogonal initialization provided in https://github.com/JiJingYu/delta_orthogonal_init_pytorch

Second, the gradient norm distribution of tanh is more concentrated and neutral compared with rectifiers, whereas its test accuracy is much lower. Both these phenomena accord with our predictions in Section 5.1: tanh is more stable compared with rectifier neurons, whereas rectifiers are more effective. Besides, the gradient explosion occasionally happens with PReLU. Moreover, with $\gamma = 0.18$, leaky ReLU outperforms ReLU by $+2.83\%$, $+2.98\%$, and $+1.87\%$ on Gaussian, orthogonal and delta orthogonal weights, respectively, which can be partially attributed to the additional stability provided by leaky ReLU. The reason is that the gradient norm is more concentrated with leaky ReLU, as illustrated in Fig. 5(b)-(c). Fig. 5(e)-(f) compare the gradient norm of leaky ReLU with different negative slope coefficient γ, and models with a larger γ have flatter distribution, whereas a too large γ would result in weak nonlinearity. This trade-off between stability and nonlinearity is also illustrated in Table 8: while the test accuracy of $\gamma = 0.18$ is $+0.59\%$ higher than that of $\gamma = 0.5$ when the network is stabilized with BN, the latter one is $+0.4\%$ or $+1.56\%$ higher with orthogonal or delta orthogonal weights, respectively. The highest test accuracy is achieved when $\gamma = 0.3$. With delta orthogonal weights, it is even $+0.54\%$ higher than the BN baseline. For sPReLU, Table 8 shows that compared with PReLU, sPReLU achieves $+0.76\%$ accuracy gain on Gaussian weights, $+1.77\%$ on orthogonal initialization and $+7.73\%$ on delta orthogonal weights with a much narrower confidence interval. Besides, sPReLU achieves comparable results with leaky ReLU under $\gamma = 0.3$ without the need of hand-crafted hyper-parameter. Last but not least, for all the activation functions except tanh and PReLU, orthogonal and delta orthogonal weights achieve better results compared with Gaussian weights in Table 8 and demonstrate more concentrated gradient norm in Fig. 5. For tanh, one possible explanation is that tanh diminishes the flow of information in the forward pass, and the noise introduced by the Gaussian distribution might partially alleviate this problem. All in all, our discussions in Section 5.1 predict most of phenomena in our experiences, which demonstrates the effectiveness of our theorem.

Normalization in Serial Network. In this part, we evaluate the performance of different normalization techniques. The test accuracy of all configurations is summarized in Table 9, and the gradient distribution is shown in Fig. 6.

According to Table 9, our SMN and its L1-norm version achieve comparable test accuracy compared with BN, and its gradient norm distribution in the 32-layer serial network

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

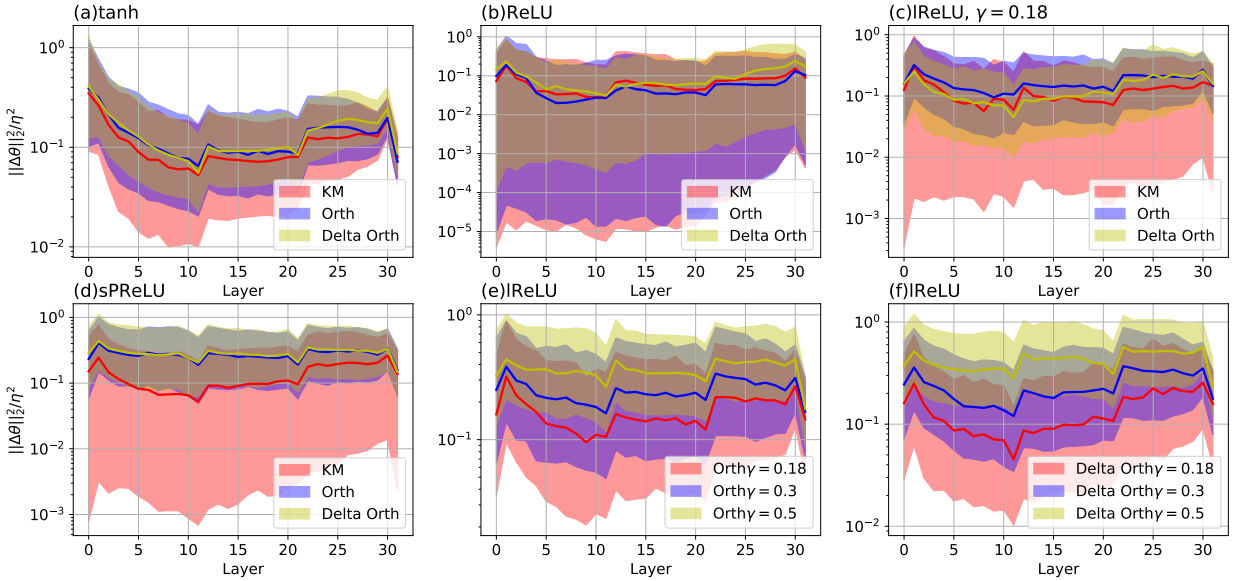


Fig. 5. Gradient norm distribution throughout the network under different configurations. The colored regions represent the range from 15 percentile to 85 percentile, while the solid line is the median. "lReLU" denotes "leaky ReLU".

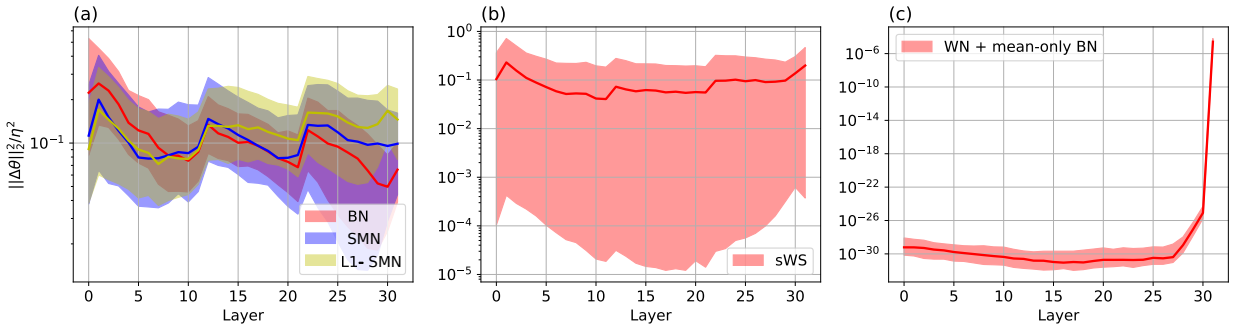


Fig. 6. Gradient norm distribution throughout the network under different normalization techniques.

TABLE 9
Test accuracy of normalization techniques on CIFAR-10 in serial networks (CI=95%).

Approach	Test Acc.
Batch Normalization (BN)	$\mathbf{88.70\% \pm 0.31\%}$
Second Moment Normalization (SMN) (ours)	$\mathbf{88.50\% \pm 0.26\%}$
L1-norm SMN (L1-SMN) (ours)	$88.34\% \pm 0.61\%$
Scaled Weight Standardization (sWS) (ours)	$\mathbf{88.06\% \pm 0.49\%}$
Weight Norm + mean-only BN [6]	10% (not converge)

is also akin to that of BN, both of which demonstrate the effectiveness of our novel normalization technique. While the original weight normalization does not converge due to the improper hyper-parameter, our scaled weight standardization demonstrates a neutral gradient norm distribution, and its test accuracy is only 0.64% lower than the BN baseline. The only difference between SMN and sWS is that SMN estimates the 2^{nd} moment from pre-activations while sWS estimates from weight kernels, and the former one's distribution is obviously narrower than the latter one. This evidences our earlier conclusion that the 2^{nd} moment should be obtained from pre-activations for stability.

Self-Normalizing Neural Network. In this part, we evaluate SeLU under different setups of γ_0 and ϵ with Gaussian or orthogonal weights. The test accuracy of all configurations is summarized in Table 10, and the gradient distribution is illustrated in Fig. 7.

For the orthogonal initialization, we find that the delta

TABLE 10
Test accuracy of SeLU under different configurations (CI=95%). All the methods except for [22] are ours.

Weight Initialization	γ_0	ϵ	Test Acc.
KM	**1**	[22]	$89.00\% \pm 0.51\%$
		0.00	$85.13\% \pm 1.35\%$
		0.03	$\mathbf{89.42\% \pm 0.29\%}$
		0.07	$89.25\% \pm 0.58\%$
	2	0.03	$89.42\% \pm 0.55\%$
Orth	**1**	[22]	$89.10\% \pm 0.33\%$
		0.00	$85.53\% \pm 0.64\%$
		0.03	$\mathbf{89.49\% \pm 0.32\%}$
		0.07	$89.10\% \pm 0.39\%$
	2	0.03	$89.34\% \pm 0.39\%$
BN with ReLU			$88.70\% \pm 0.31\%$

orthogonal initialization [3] is less stable compared with the orthogonal initialization [28], which might be caused by that the sparse kernel in the delta orthogonal does not work well under the central limit theorem. As shown in Table 10, when $\epsilon = 0.07$, the test accuracy of our model is similar to the result in Klambauer et al. (2017) [22], which demonstrates that their work is a special case of ours. As our analysis suggests that ϵ should be slightly smaller than $\frac{1}{L}$, for the 32-layer network, we choose $\epsilon = 0.03$, and the test accuracy is $+0.42\%$ and $+0.39\%$ higher than the original configuration with the Gaussian and orthogonal initialization, which indicates that the original choices of α and λ are not optimal. As illustrated in Fig. 7, a higher ϵ

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

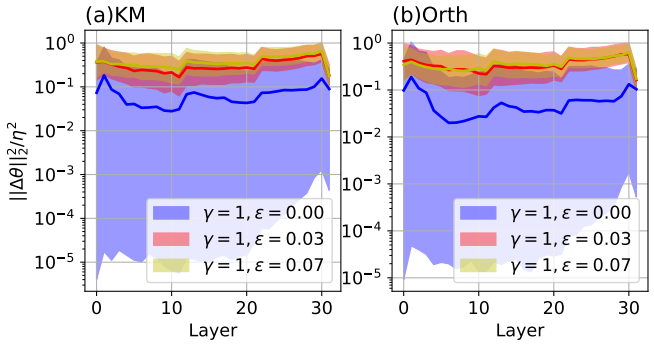


Fig. 7. Gradient norm distribution throughout the network with SeLU under different configurations.

results in more neutral gradient norm distribution, which also accords to our prediction. Besides, our method can still achieve comparable results when $\gamma_0 = 2$. With SeLU, the orthogonal initialization does not have significant advantages over the Gaussian initialization, this reflects the normalization effectiveness of SeLU.

DenseNet. Here we evaluate the performance of some initialization and normalization techniques on DenseNet.

TABLE 11
Test accuracy on DenseNet (CI=95%).

Approach	Test Acc.
Kaiming Init + ReLU [1]	$89.37\% \pm 0.43\%$
Orthogonal Init + leaky ReLU, $\gamma = 0.3$ (ours)	$89.56\% \pm 0.30\%$
Orthogonal Init + SeLU, $\gamma_0 = 2, \epsilon = 0.03$ (ours)	$\mathbf{90.51\% \pm 0.35\%}$
Batch Normalization (BN)	$\mathbf{92.10\% \pm 0.54\%}$
Scaled Weight Standardization (sWS) (ours)	$91.35\% \pm 0.46\%$
Second Moment Normalization (SMN) (ours)	$\mathbf{92.06\% \pm 0.25\%}$

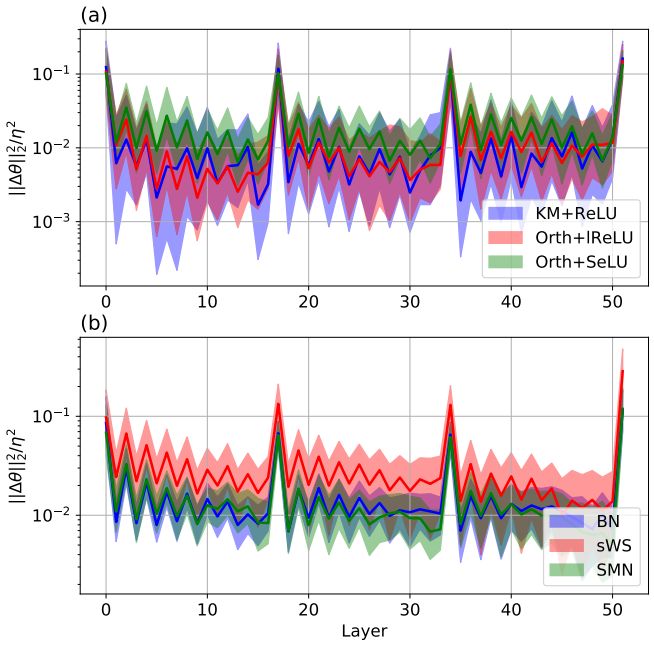


Fig. 8. Gradient norm distribution throughout DenseNet under different configurations.

We take KM initialization as the baseline for initialization techniques and BN as the baseline for normalization techniques. As listed in Table 11, leaky ReLU yields $+0.19\%$ higher accuracy than the initialization baseline. For normalization techniques, while the accuracy of sWS is 0.75% lower than BN, SMN we proposed is only -0.04% lower on accuracy, which further demonstrates its effectiveness. SeLU with $\epsilon = 0.03$ surpasses other initialization techniques, whereas its accuracy is relatively lower than that with SMN and BN.

As illustrated in Fig. 8, even in the 52-layer network with a learning rate of 0.1, the gradient norm is still more concentrated than serial networks without dense connections, which verifies our conclusion that the dense connections can effectively stabilize the network. In Fig. 8(a), SeLU's gradient is more neutral compared with others; in Fig. 8(b), while SMN has a similar gradient distribution with BN, that of sWS is relatively higher. These phenomenons accord with the accuracy results in Table 11.

ResNet. Here we evaluate the performance of Fixup initialization and SMN on ResNet-56. The accuracy is summarized in Table 12 and the gradient norm distribution is illustrated in Fig. 9. Fixup initialization with bias, scale, and mixup regularization achieves higher accuracy compared with BN, which illustrates its effectiveness. Moreover, although in Zhang et al. (2019) [4] p is set to 2, we empirically show that $p = 1.5$ can yield slightly higher accuracy. The test accuracy of SMN is 0.43% lower than BN, which can be reduced to 0.17% with mixup regularization. However, as Fig. 9 shows, SMN shares the similar gradient distribution with BN. These results imply that since the mean is estimated from weight kernels, compared with BN, SMN has a weaker regularization effect during training, and the data augmentation like mixup can partially compensate for it.

TABLE 12
Test accuracy on ResNet-56 under different configurations (CI=95%).

Method	Remarks	Test Acc.
Fixup	$p = 2$ [4]	$90.38\% \pm 0.81\%$
	$p = 2$, b&s [4]	$92.38\% \pm 0.15\%$
	$p = 2$, b& s, mixup [4]	$93.93\% \pm 0.53\%$
	$p = 1.5$, b& s, mixup (ours)	$\mathbf{94.28\% \pm 0.40\%}$
BN		$93.71\% \pm 0.27\%$
	mixup	$\mathbf{94.10\% \pm 0.48\%}$
SMN (ours)		$93.28\% \pm 0.82\%$
	mixup	$\mathbf{93.93\% \pm 0.56\%}$

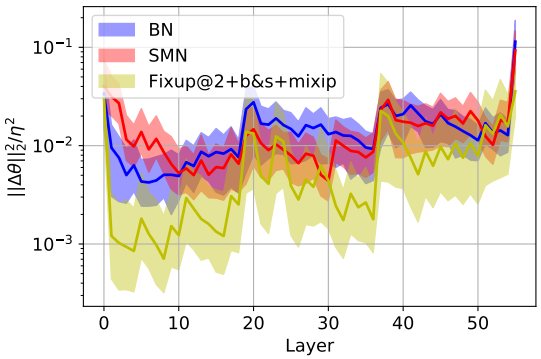


Fig. 9. Gradient norm distribution throughout ResNet-56 under different configurations.

7.3 Experiments on ImageNet

Unlike previous theoretical studies [3], [13], [15], [43], [44] that only evaluate their conclusions on small datasets like MNIST and CIFAR-10, we further validate some of the important conclusions on ImageNet to demonstrate that they are still valid on large-scale networks and datasets.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2020.3010201, IEEE Transactions on Pattern Analysis and Machine Intelligence

We choose Conv MobileNet V1 [34] and ResNet 50 [8] for serial and parallel networks, respectively. Conv MobileNet V1 is one of the latest serial networks, which has relatively good accuracy (71.7% reported in Howard et al. (2017) [34]) on ImageNet and is not over-parameterized like VGG [31]. The "Conv" means we use traditional convolutions instead of depthwise separable convolution, which is majorly due to two reasons. First, we find the latter one takes hundreds of epochs to converge. Second, as in depthwise convolution we have $c_{in} = 1$, it is too small for most of the mentioned techniques on serial networks. The Conv MobileNet V1 consists of 15 convolutional layers and a fully-connected layer at the end, wherein all the blocks are connected in serial and there are no shortcut connections between them. Originally, it is stabilized with BN. Since most methods for serial networks are not stable under the high learning rate, we follow the training scheme in Simonyan& Zisserman (2014) [31], i.e. the model is trained for 90 epochs, the batch size is set to 512, and the initial learning rate is set to 0.02 with decay by $10\times$ at epoch 60, 75. For ResNet-50, we follow the classic training scheme in He et al. (2016) [8], i.e. the model is trained for 90 epochs, the batch size is set to 256, and the initial learning rate is set to 0.1 with decay by $10\times$ at epoch 30, 60. All the results are averaged over the last 10 epochs.

We also evaluate the performance of each method under micro local batch size scenario, in which BN has high error due to the inaccurate batch statistics estimation [24], [26]. We keep the original global batch size and partition the samples onto more GPUs such that each GPU has 4 samples in each iteration, and the statistics in normalization techniques are estimated locally on each GPU. The results of BN and SMN with micro local batch size are marked by "4 images/GPU" in Table 13 and 14. As other techniques do not rely on the local estimated statistics and their results are invariant under different local batch size, we only report them once.

On Conv MobileNet V1. Previous experiments on CIFAR-10 illustrate that leaky ReLU and SeLU with $\epsilon \approx 1/L$ surpass the accuracy of BN. Here we further evaluate their performance on ImageNet. The detailed configurations are: 1) Leaky ReLU, $\gamma = 0.3$ with the orthogonal initialization; 2) SeLU, $\epsilon = 0.06$ or 0.03, $\gamma_0 = 1$ with the Gaussian initialization. We choose $\epsilon = 0.06$ for it is slightly smaller than $\frac{1}{L} = 0.067$. The results are given in Table 13.

TABLE 13
Test error of methods on Conv MobileNet V1 (CI=95%).

Method	Top-1 Error	Top-5 Error
SeLU [22]	Explode in the first epoch	
SeLU $\epsilon = 0.06$ (ours)	$\mathbf{30.37\% \pm 0.10\%}$	$\mathbf{11.67\% \pm 0.03\%}$
SeLU $\epsilon = 0.03$ (ours)	$30.87\% \pm 0.07\%$	$11.84\% \pm 0.04\%$
ReLU, Gaussian [1]	$31.16\% \pm 0.08\%$	$11.87\% \pm 0.06\%$
lReLU, Gaussian (ours)	$29.39\% \pm 0.08\%$	$\mathbf{10.82\% \pm 0.07\%}$
lReLU, Orth (ours)	$\mathbf{29.36\% \pm 0.13\%}$	$10.82\% \pm 0.08\%$
lReLU, Delta Orth (ours)	$29.47\% \pm 0.09\%$	$10.92\% \pm 0.08\%$
BN	$28.58\% \pm 0.07\%$	$10.16\% \pm 0.05\%$
BN (4 sample/GPU)	$35.90\% \pm 0.24\%$	$14.57\% \pm 0.04\%$

The original configuration of SeLU [22] suffers from the gradient explosion due to the too large ϵ. Via $\epsilon = 0.06$, we reach 30.37% top-1 error with Gaussian weights. However, for smaller ϵ, i.e. 0.03, the top-1 error is 0.5% higher, for its normalization effectiveness is lower. For leaky ReLU with $\gamma = 0.3$ and the Gaussian initialization, the top-1 error is only 0.89% higher than the BN baseline and 1.77% lower than the ReLU + Gaussian baseline. Under micro batch size scenario, Initialization techniques and SeLU achieve better performance than BN.

On ResNet-50. For ResNet-50, we test the performance of the Fixup initialization and our SMN. For the former one, we test both Zhang et al. (2019) [4]'s original configuration and ours with $p = 1.5$. The scalar multiplier and bias are added and the interpolation coefficient in mixup is set to 0.7, just following [4]. For the latter one, we directly replace BN in original ResNet-50 with SMN without any further modification. For the BN baseline, the interpolation coefficient in mixup is set to 0.2, which is reported to be the best [4]. The results are summarized in Table 14.

TABLE 14
Test error of methods on ResNet-50 (CI=95%).

Method	Top-1 Error	Top-5 Error
BN	$24.35\% \pm 0.15\%$	$7.49\% \pm 0.09\%$
BN+mixup	$23.81\% \pm 0.13\%$	$\mathbf{6.86\% \pm 0.08\%}$
SMN (ours)	$24.90\% \pm 0.19\%$	$7.65\% \pm 0.14\%$
SMN+mixup (ours)	$\mathbf{23.74\% \pm 0.23\%}$	$6.94\% \pm 0.10\%$
L1-MN+mixup (ours)	$24.04\% \pm 0.19\%$	$7.10\% \pm 0.14\%$
Fixup [4]	$24.77\% \pm 0.15\%$	$7.72\% \pm 0.13\%$
Fixup (ours)	$24.72\% \pm 0.12\%$	$7.70\% \pm 0.10\%$
WN [6]	33% [25]	Not reported
BN(4 sample/GPU)	$28.89\% \pm 0.21\%$	$9.60\% \pm 0.18\%$
SMN(4 sample/GPU)	$26.93\% \pm 0.23\%$	$8.41\% \pm 0.24\%$

Without the mixup regularization, the top-1 error of our SMN is 0.55% higher than BN. However, we also observe that its top-1 training error is 0.68% lower, which implies that the test accuracy loss is mainly due to the lack of regularity. Inspired by Zhang et al. (2019) [4], we further utilize the mixup [23] to augment the input data with the interpolation coefficient of 0.2, which is the same with the baseline configuration. Then, the top-1 error becomes 23.74%, which is 0.07% lower than BN. Also, we evaluate the L1-norm configuration with the mixup regularization, and the top-1 error is still comparable with BN. Notably, we will show that SMN has 30% less computation overhead than BN in Section 7.4, which makes it a powerful substitute for BN. For the Fixup initialization, our configuration can reach the same test error of the configuration in Zhang et al. (2019) [4]. In micro local batch size scenario, on one hand, Fixup initialization has better performance over normalization techniques as it does not depend on the estimated statistics. On the other hand, SMN achieves 2% lower top-1 error than BN. The explanation is that while BN estimates both the first and second moments from the pre-activations, SMN only estimates the second moment, which results in less dependence on the estimation.

7.4 Training Overhead.

We explore the training overhead brought by different methods. For SeLU, WN, and BN, we directly use the primitive APIs provided by PyTorch. For sWS and SMN, fused CUDA kernels are developed. The overhead is defined as extra operations or data movements over the valiant "Conv-ReLU" block. For memory overhead, we measure the total bytes transferred between GPU's off-chip DRAM and the on-chip L2 cache. For computation overhead, we evaluate the total number of floating-point operations executed by GPU. All the results are directly measured with NVIDIA's

profiling tool on Tesla V100 GPU. For easier comparison, the overhead results are normalized to the overhead of BN. All the methods are evaluated on three different feature map sizes marked with N-C-W-H chosen from ResNet-50.

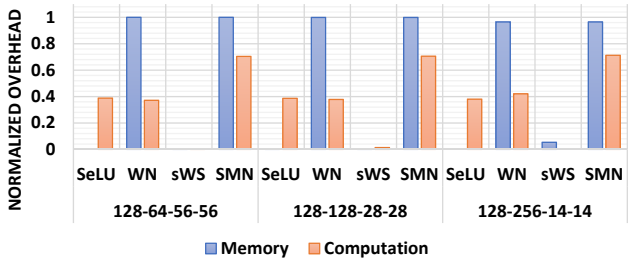


Fig. 10. Normalized memory and computation overhead of different methods and benchmarks.

First, SeLU introduces extra computation overhead as it is more complex than rectifiers like ReLU. However, as the activation functions are basically memory-bound, the computation overhead does not have significant influence on latency. Second, the overhead of sWS is much lower than that of WN and neglectable on all three benchmarks. While it has the similar execution pattern of BN, the number of entries in the weights is much smaller than the number of pre-activations. The overhead of WN mainly comes from the mean-only BN. Last but not least, comparing with BN, our SMN eliminates 30% of the computation overhead.

8 Conclusion

In this paper, we propose a novel metric, block dynamical isometry, that can characterize DNNs using the gradient norm equality property. A comprehensive and highly modularized statistical framework based on advanced tools in free probability is provided to simplify the evaluation of our metric. Compared with existing theoretical studies, our framework can be applied to networks with various components and complex connections, which is much easier to use and only requires weaker prerequisites that are easy to verify. Powered by our novel metric and framework, unlike previous studies that only focus on a particular network structure or stabilizing methodology, we analyze extensive techniques including initialization, normalization, self-normalizing neural network, and shortcut connections. Our analysis not only shows that the our block dynamical isometric is a universal philosophy behind these methods but also provides inspirations for the improvement of existing techniques and the development of new methods. As study cases, we introduce an activation function selection strategy for initialization, a novel configuration for weight normalization, a depth-aware way to derive coefficient in SeLU, and the second moment normalization. These methods achieve advanced results on both CIFAR-10 and ImageNet with rich network structures. Besides what we have presented in this paper, there is still potential in our framework that is not fully exploited. For instance, our analysis in Section 5.3 shows "SeLU" may not be the only choice for self-normalizing neural networks. Moreover, although we focus on CNNs in this paper, the methodology also has the potential to improve other models like recurrent neural networks and spiking neural networks. Our framework can also be utilized in other norm-based metrics like GSC [17].

References

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

[2] D. Mishkin and J. Matas, "All you need is a good init," *arXiv preprint arXiv:1511.06422*, 2015.

[3] L. Xiao, Y. Bahri, J. Sohl-Dickstein, S. S. Schoenholz, and J. Pennington, "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks," *arXiv preprint arXiv:1806.05393*, 2018.

[4] H. Zhang, Y. N. Dauphin, and T. Ma, "Fixup initialization: Residual learning without normalization," *arXiv preprint arXiv:1901.09321*, 2019.

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[6] T. Salimans and D. P. Kingma, "Weight normalization: A simple reparameterization to accelerate training of deep neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 901–909.

[7] S. Qiao, H. Wang, C. Liu, W. Shen, and A. Yuille, "Weight standardization," *arXiv preprint arXiv:1903.10520*, 2019.

[8] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[10] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli, "Exponential expressivity in deep neural networks through transient chaos," in *Advances in neural information processing systems*, 2016, pp. 3360–3368.

[11] G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and S. S. Schoenholz, "A mean field theory of batch normalization," *arXiv preprint arXiv:1902.08129*, 2019.

[12] S. S. Schoenholz, J. Gilmer, S. Ganguli, and J. Sohl-Dickstein, "Deep information propagation," *arXiv preprint arXiv:1611.01232*, 2016.

[13] J. Pennington, S. Schoenholz, and S. Ganguli, "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice," in *Advances in neural information processing systems*, 2017, pp. 4785–4795.

[14] J. Pennington, S. S. Schoenholz, and S. Ganguli, "The emergence of spectral universality in deep networks," *arXiv preprint arXiv:1802.09979*, 2018.

[15] W. Tarnowski, P. Warchoł, S. Jastrzebski, J. Tabor, and M. A. Nowak, "Dynamical isometry is achieved in residual networks in a universal way for any activation function," *arXiv preprint arXiv:1809.08848*, 2018.

[16] Z. Ling and R. C. Qiu, "Spectrum concentration in deep residual learning: a free probability appproach," *arXiv preprint arXiv:1807.11694*, 2018.

[17] G. Philipp, D. Song, and J. G. Carbonell, "Gradients explode-deep networks are shallow-resnet explained," 2018.

[18] D. Arpit and Y. Bengio, "The benefits of over-parameterization at initialization in deep relu networks," *arXiv preprint arXiv:1901.03611*, 2019.

[19] J. A. Mingo and R. Speicher, *Free probability and random matrices*. Springer, 2017, vol. 35.

[20] J. Chen, T. Van Voorhis, and A. Edelman, "Partial freeness of random matrices," *arXiv preprint arXiv:1204.2257*, 2012.

[21] D. Arpit, Y. Zhou, B. U. Kota, and V. Govindaraju, "Normalization propagation: A parametric technique for removing internal covariate shift in deep networks," *arXiv preprint arXiv:1603.01431*, 2016.

[22] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-normalizing neural networks," in *Advances in neural information processing systems*, 2017, pp. 971–980.

[23] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[24] Z. Chen, L. Deng, G. Li, J. Sun, X. Hu, L. Liang, Y. Ding, and Y. Xie, "Effective and efficient batch normalization using a few uncorrelated data for statistics estimation," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[25] I. Gitman and B. Ginsburg, "Comparison of batch normalization and weight normalization algorithms for the large-scale image classification," *arXiv preprint arXiv:1709.08145*, 2017.

[26] Y. Wu and K. He, "Group normalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–19.

[27] T. Sheng, C. Feng, S. Zhuo, X. Zhang, L. Shen, and M. Aleksic, "A quantization-friendly separable convolution for mobilenets," in *2018 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications (EMC2)*. IEEE, 2018, pp. 14–18.

[28] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[29] B. Cakmak, "Non-hermitian random matrix theory for mimo channels," Master's thesis, Institutt for elektronikk og telekommunikasjon, 2012.

[30] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[32] P. A. Sokol and I. M. Park, "Information geometry of orthogonal initializations and training," *arXiv preprint arXiv:1810.03785*, 2018.

[33] S. Hayou, A. Doucet, and J. Rousseau, "Mean-field behaviour of neural tangent kernel for deep neural networks," *arXiv preprint arXiv:1905.13654*, 2019.

[34] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[36] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 2011, pp. 315–323.

[37] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[38] D. Ulyanov, A. Vedaldi, and V. Lempitsky, "Instance normalization: The missing ingredient for fast stylization," *arXiv preprint arXiv:1607.08022*, 2016.

[39] S. Wu, G. Li, L. Deng, L. Liu, D. Wu, Y. Xie, and L. Shi, "L1-norm batch normalization for efficient training of deep neural networks," *IEEE transactions on neural networks and learning systems*, 2018.

[40] P. Luo, X. Wang, W. Shao, and Z. Peng, "Towards understanding regularization in batch normalization," 2018.

[41] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[42] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8697–8710.

[43] E. Haber and L. Ruthotto, "Stable architectures for deep neural networks," *Inverse Problems*, vol. 34, no. 1, p. 014004, 2017.

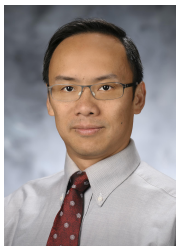
[44] R. Burkholz and A. Dubatovka, "Exact information propagation through fully-connected feed forward neural networks," *arXiv preprint arXiv:1806.06362*, 2018.

Zhaodong Chen received his B.E. degree from Tsinghua University, China in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Engineering, University of California, Santa Barbara. He is currently doing research in Scalable Energy-Efficient Architecture Lab. His research interest lays in deep learning and computer architecture.

Lei Deng received the B.E. degree from University of Science and Technology of China, Hefei, China in 2012, and the Ph.D. degree from Tsinghua University, Beijing, China in 2017. He is currently a Postdoctoral Fellow at the Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA, USA. His research interests span the area of brain-inspired computing, machine learning, neuromorphic chip, computer architecture, tensor analysis, and complex networks. Dr. Deng has authored or co-authored over 40 refereed publications. He was a PC member for *ISNN* 2019. He currently serves as a Guest Associate Editor for *Frontiers in Neuroscience* and *Frontiers in Computational Neuroscience*, and a reviewer for a number of journals and conferences. He was a recipient of MIT Technology Review Innovators Under 35 China 2019.

Bangyan Wang received his B.E. degree from Tsinghua University, China in 2017. He is currently a Ph.D. student at the Department of Electrical and Computer Engineering, University of California, Santa Barbara. His current research interests include domain-specific accelerator design and tensor analysis.

Guoqi Li received the B.E. degree from the Xi'an University of Technology, Xi'an, China, in 2004, the M.E. degree from Xi'an Jiaotong University, Xi'an, China, in 2007, and the Ph.D. degree from Nanyang Technological University, Singapore, in 2011. He was a Scientist with Data Storage Institute and the Institute of High Performance Computing, Agency for Science, Technology and Research (ASTAR), Singapore, from 2011 to 2014. He is currently an Associate Professor with the Center for Brain Inspired Computing Research (CBICR), Tsinghua University, Beijing, China. His current research interests include machine learning, brain-inspired computing, neuromorphic chip, complex systems and system identification. Dr. Li has authored/co-authored over 80 journal/conference papers. He has been actively involved in professional services such as serving as the International Technical Program Committee Member/PC Member/Publication Chair/Tutorial/Workshop Chair, and Track Chair for international conferences. He is currently an Editorial-Board Member for Control and Decision and Frontiers in Neuroscience, and a Guest Associate Editor for Frontiers in Neuroscience. He was the recipient of the 2018 First Class Prize in Science and Technology of the Chinese Institute of Command and Control, Best Paper Awards (EAIS 2012 and NVMTS 2015), and the 2018 Excellent Young Talent Award of Beijing Natural Science Foundation.



Yuan Xie received B.S. degree in Electronic Engineering from Tsinghua University, Beijing, China in 1997, and M.S. and Ph.D. degrees in Electrical Engineering from Princeton University, NJ, USA in 1999 and 2002. He was an Advisory Engineer with IBM Microelectronic Division, VT, USA from 2002 to 2003. He was a Full Professor with Pennsylvania State University, PA, USA, 2003-2014. He was a Visiting Researcher with Interuniversity Microelectronics Centre (IMEC), Leuven, Belgium from 2005 to 2007 and in 2010. He was a Senior Manager and Principal Researcher with AMD Research China Lab, Beijing, China from 2012 to 2013. He is currently a Professor with the Department of Electrical and Computer Engineering, University of California at Santa Barbara, CA, USA. His interests include VLSI design/Electronics Design Automation (EDA)/computer architecture, and embedded systems. Dr. Xie is an expert in computer architecture who has been inducted to ISCA/MICRO/HPCA Hall of Fame and IEEE/AAAS/ACM Fellow. He was a recipient of multiple Best Paper Award (e.g. HPCA 2015), Best Paper Nominations (e.g. MICRO 2013), 2016 IEEE Micro Top Picks Award, 2008 IBM Faculty Award, and 2006 NSF CAREER Award. He has extensively collaborated with industry partners and helped the transition of research ideas to industry.