MimicNorm: Weight Mean and Last BN Layer Mimic the Dynamic of Batch Normalization

Abstract

Substantial experiments have validated the success of Batch Normalization (BN) Layer in benefiting convergence and generalization. However, BN requires extra memory and float-point calculation. Moreover, BN would be inaccurate on micro-batch, as it depends on batch statistics. In this paper, we address these problems by simplifying BN regularization while keeping two fundamental impacts of BN layers, *i.e.*, data decorrelation and adaptive learning rate. We propose a novel normalization method, named *MimicNorm*, to improve the convergence and efficiency in network training. MimicNorm consists of only two light operations, including modified weight mean operations (subtract mean values from weight parameter tensor) and one BN layer before loss function (last BN layer). We leverage the neural tangent kernel (NTK) theory to prove that our weight mean operation whitens activations and transits network into the chaotic regime like BN layer, and consequently, leads to an enhanced convergence. The last BN layer provides auto-tuned learning rates and also improves accuracy. Experimental results show that MimicNorm achieves similar accuracy for various network structures, including ResNets and lightweight networks like ShuffleNet, with a reduction of about 20% memory consumption. The code is publicly available at https://github.com/Kid-key/MimicNorm.

1 Introduction

Deep learning applications have led to revolutionary consequences. In addition to the development of hardware, competent algorithms for training complex deep networks play a crucial role in practical scenarios. Batch Normalization (BatchNorm or BN) has emerged as one of the most successful techniques to improve the accuracy and speed up training. BN layers are adopted in most state-of-the-art neural networks to stabilize the training process, especially for deep networks, *e.g.*, ResNets (He et al. 2016). Inspired by BN regulation, a variety of normalization techniques have been developed for natural language processing and unsupervised tasks, including Instance Normalization (Ulyanov, Vedaldi, and Lempitsky 2016), Layer Normalization (Ba, Kiros, and Hinton 2016), and Group Normalization (Wu and He 2018). Despite widespread adoption of BN layers, little consensus has been reached on understanding them. Ioffe and

Szegedy (2015) normalize activations in intermediate layers of deep neural networks (DNNs) to reduce the internal covariate shift (ICS) and constrain the distribution on each layer. However, Santurkar et al. (2018) argue that the effectiveness of BN has little relation to ICS and identify the key impact of BN as the smaller Lipschitz constant and smoother landscape. Arora, Li, and Lyu (2019) consider the auto rate-tuning mechanism based on the scale-invariant property of BN networks. The BN layers automatically decrease the learning rates and enhance the convergence speed to a first-order stationary point. Luo et al. (2019) demonstrate that BN equivalently imposes Population Normalization (PN) and gamma decay. BN layers also show a tangled impact with weight decay regularization (Zhang et al. 2019).

In this paper, we focus on the mechanism of BN as the guidance for efficient network design. BN layers require buffers for both forward and backward pass that account for 20% total memory consumption of the networks. Specifically, we aim at addressing the problems listed as below.

1. **Fundamental impacts of BN layers.** BN network normalizes activations into zero mean and unit variance in a layer-by-layer fashion in the forward pass, and adjusts the gradient propagation in the backward process. It is of necessity to disentangle and identify the fundamental impacts by BN layers on network training.

2. **Efficient alternative to BN layers.** Simplified and pure regularization can be developed as an efficient alternative to BN layers by maintaining the fundamental impacts and reducing the memory consumption.

Neural tangent kernel (NTK) leverages the kernel method to analyze infinite-width networks (Jacot, Gabriel, and Hongler 2018). NTK is determined by the correlations of gradients to control the network convergence. In this paper, we investigate the impact of BN layers on correlation propagation. We demonstrate in theory that both weight mean and BN layers lead to chaotic NTK (Jacot, Gabriel, and Hongler 2019), and consequently, achieve linear convergence rates. Furthermore, we empirically find that one BN layer before loss function (last BN layer in the rest of this paper) avoids uncontrolled variance and enables large learning rates. The contributions of this paper are summarized as below.

1. We demonstrate that, for deep infinite-width networks with ReLU activation function, weight mean operations
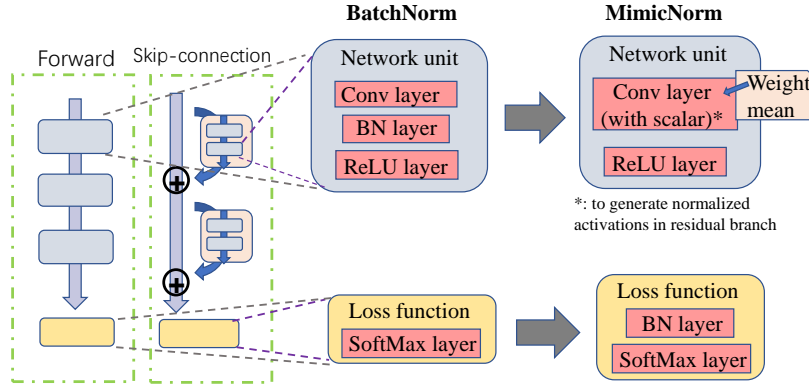
Figure 1: Illustrative diagram of *MimicNorm*. Modern neural networks are composed of stacked basic units and a loss function, usually CrossEntropy for classification. To remove BN layers, we introduce weight mean for each convolutional layers and modify initialization to generate stable intermediate activations. Besides, we apply a learnable scalar multiplier for skip-connection to normalize residual. And one last BN layer is inserted before the loss function to adjust the learning rate.

and BN layers have same NTK expressions and enable network training with equivalent linear convergence rates, as elaborated in Section 3.

2. We propose an efficient alternative to BN networks, namely *MimicNorm*, that consists of the weight norm and last BN layer to enable automatic tuning of learning rates, as presented in Figure 1 and Section 4.

Experimental results show that MimicNorm achieves the same accuracy on various BN networks with a reduction of memory consumption. For skip-connection structures (*e.g.*, ResNet50 and ResNet110) and lightweight networks (*e.g.*, ShufflNetV2), our method reduces the memory consumption by 20% in comparison to BN networks.

2 Background

We begin with an overview of recent development of neural tangent kernel and the related concepts to describe the training process of DNNs (Jacot, Gabriel, and Hongler 2018, 2019; Lee et al. 2019; Xiao, Pennington, and Schoenholz 2019). For brevity, we keep our discussion within deep Fully-Connected Neural Networks (FCNNs). Similar but intricate analysis of convolutional networks can be found in (Arora et al. 2019; Bietti and Mairal 2019; Yang 2019). A general fully-connected network can be represented by the mapping from data space $\mathbb{R}^{n_0}$ to output space $\mathbb{R}^{n_L}$ consisting of L cascade hidden layers.

$$h^1 = \frac{\sigma_w}{\sqrt{2n_0}}\mathbf{W}^0 x + \sigma_b b^0 \; ,$$
$$h^{l+1} = \frac{\sigma_w}{\sqrt{n_l}}\mathbf{W}^l \phi(h^l) + \sigma_b b^l \quad 1 \le l < L, \qquad (1)$$

where ϕ is the nonlinear element-wise function, and we take ReLU activation function $\phi(x) = \max(x, 0)$ in this paper. Each layer has weight matrix $\mathbf{W}^l$ and bias vector b^l as parameters, randomly initialized from standard normal distribution $\mathcal{N}(0, 1)$. The hyper-parameters σ_w and σ_b control the scale of the weights and biases. For DNNs, we also require the stable conditions for σ_w and σ_b such that activations are bounded under $L \to \infty$. For ReLU network, we have stable configuration that $\sigma_w^2/2 + \sigma_b^2 = 1$, equivalently $\mathbb{E}(\|h^l\|) = \|x\|, \forall 1 \le l \le L$. We denote the output of an FCNN at time t as $F_{\theta_t}(x) \equiv h^L$ with parameter set $\theta = \{\mathbf{W}^l, b^l\}_{0 \le l < L}$. The loss function measuring the discrepancy between prediction and the true label, is denoted as $\iota(h^L, y) : \mathbb{R}^{n_L} \times \mathbb{R}^{n_L} \to \mathbb{R}$, where y is the label for x. On training set $\mathcal{D} = \{(F_{\theta_t}(x), y) | x \in \mathcal{X}\}$, training process minimizes the loss values $\mathcal{L} = \mathbb{E}_{\mathcal{D}}[\iota(x, y)]$. In the following analysis, we normalize the input data such that $\|x\| = 1$.

Neural Network Gaussian Process Kernel

As the width $n_l, \; l = 1, 2, ..., L$ asymptotically goes to infinite, we conclude that the activations in each layer are i.i.d. zero-mean Gaussian random variables, whose behaviors are governed by the covariance matrix. As such, the evolution of activations through layers are approximated as Gaussian process and the correlation coefficient with given pair of data is called neural network Gaussian Process (NNGP) kernel[1] (Xiao, Pennington, and Schoenholz 2019). We define random vectors u, v from probability space $\Omega_n(\rho)$ (in the scalar case where $n = 1$, we will ignore the subscript), where u, v are n-dimension Gaussian random variables with covariance value ρ, such that $(u, v) \sim \mathcal{N}(\mathbf{0}, \left(\begin{smallmatrix} 1 & \rho \\ \rho & 1 \end{smallmatrix}\right) \otimes \mathbf{I_n})$.

Given two input data x and x' with covariance $\mathcal{K}^0(x, x') = \frac{1}{n_0} x^T x'$, the kernel value in the l-th layer is calculated in a recursive manner.

$$\mathcal{K}^{l+1}(x, x') \equiv \mathcal{T}(\mathcal{K}^l(x, x')) = \frac{\sigma_w^2}{n_l n_{l+1}} \qquad (2)$$
$$\mathbb{E}_{(u,v) \sim \Omega_{n_l}(\mathcal{K}^l(x,x'))} \left(\phi(u)^T \mathbf{W}^{l^T} \mathbf{W}^l \phi(v) \right) + \sigma_b^2.$$

---

[1] In other papers, this kernel may be referred to as dual kernel (Daniely, Frostig, and Singer 2016) or activation kernel (Jacot, Gabriel, and Hongler 2019).

Parameter $\mathbf{W}^l$ is initialized as Gaussian random matrix, and the term $\mathbf{A}^l = \mathbf{W}^{l^T}\mathbf{W}^l$ is called *Wishart matrix*, named after J. Wishart (Wishart 1928) and discussed in Edelman (1988)[2]. Previous papers (Jacot, Gabriel, and Hongler 2018; Pennington, Schoenholz, and Ganguli 2017) consider the infinite width scenario approximate that $\frac{1}{n_l}A^l \to \mathbf{I}_{n_l}$ and derive the relation:

$$\mathcal{K}^{l+1}(x, x') = \sigma_w^2 \hat{\phi}(\mathcal{K}^l(x, x')) + \sigma_b^2, \qquad (3)$$

where $\hat{\phi}(\rho) \equiv \mathbb{E}_{(u,v)\sim\Omega(\rho)}(\phi(u)\phi(v))$ is the dual activation function defined on $[-1, 1]$. It tells how the activation function changes the covariance value. Cho and Saul (2009) derives the closed-form dual activation functions for ReLU:

$$\hat{\phi}(\rho) = \frac{\sqrt{1 - \rho^2} + (\pi - \arccos(\rho))\rho}{2\pi}. \qquad (4)$$

The transition operator $\mathcal{T}$ is fundamental in analysing dynamics of networks. Stable initialization condition is expressed as $\mathcal{T}(1) = 1$, implying activations of all layers have the same variance. Daniely, Frostig, and Singer (2016) prove that there exists one and only one stable fixed point of $\mathcal{T}$. For ReLU, this fixed point is $\rho^* = 1$.

Neural Tangent Kernel

Considering the gradient descent method to train neural networks, the derivative with parameters is $\nabla_\theta \mathcal{L} = (\nabla_\theta F_\theta(\mathcal{X}))^T \nabla_F \mathcal{L}$, and such gradient flow of network follows the differential equation at time t:

$$\frac{dF_{\theta_t}}{dt} = -\nabla_\theta F_{\theta_t}(\mathcal{X})\nabla_\theta \mathcal{L} = -\Theta_t(\mathcal{X}, \mathcal{X})\nabla_F \mathcal{L}. \quad (5)$$

The kernel $\Theta_t = \sum_{p=1}^{|\theta|} \nabla_{\theta_p} F_{\theta_t}(\mathcal{X})\nabla_{\theta_p} F_{\theta_t}(\mathcal{X})^T$ is called *Neural Tangent Kernel* (NTK), which maps a pair of data into an $n_L \times n_L$ matrix: $\mathbb{R}^{n_0} \times \mathbb{R}^{n_0} \to \mathbb{R}^{n_L \times n_L}$.

The key conclusion (Jacot, Gabriel, and Hongler 2018; Arora et al. 2019; Bietti and Mairal 2019) of NTK is its convergence behavior as the layer width goes to infinity, which converges in probability to a constant kernel:

$$\Theta_t \to \Theta \otimes I_{n_L}, \qquad (6)$$

where the scale kernel is calculated recursively by previous NNGP kernel:

$$\Theta(x, x') = \sum_{l=1}^{L} \mathcal{K}^l(x, x') \prod_{r=l+1}^{L} \dot{\mathcal{T}}(\mathcal{K}^r(x, x')), \quad (7)$$

where $\dot{\mathcal{T}}$ denotes the derivative of $\mathcal{T}$. Since the NTK describes the dynamics for neural network during training, the condition number of NTK implies the convergence of neural network. Lee et al. (2019) prove that the NTK dynamics is linear approximation of neural network with initialization. Despite Equation (6) is derived under an infinite width, the condition number of NTK is still an effective criteria indicating the training process.

---

[2]Formally, the Wishart matrix is defined as $W^T W$ or WW^T, depending on the shape of W.
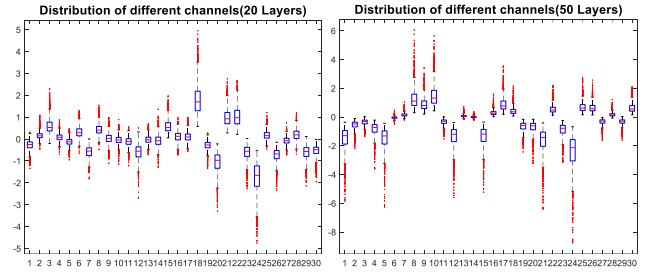

Figure 2: Straight network freezes NNGP kernel and correlates activations. We randomly select 30 channels in the 20-th and 50-th layers, and show their distributions at initial iteration. Notice that most values are located within boxes, we conclude that the intra-channel similarities and inter-channel variations increase with depth. In other words, different inputs generate similar activations.

3 Equivalence of Weight Mean and BN

It is well known that whitening input is beneficial to accelerating and improving learning algorithms. However, input whitening is insufficient on deep learning methods, as the distributions of activations vary with layers. BN is designed to reduce those variations, *i.e.*, internal covariate shift (ICS). From the perspective of kernel method, the ICS refers to the frozen NNGP kernel. Figure 2 illustrates the assorted distributions on different layers and channels. We test a 51-layer FCNN with ReLU activation that is initialized as $(\sigma_w, \sigma_b) = (\sqrt{2}, 0)$ with batchsize 2000, and plot their distributions on very beginning iteration. The intra-channel values concentrate and inter-channel values disperse with the growth of depth. When the depth increases to infinity, network generates constant values, and NNGP kernel freezes.

Fundamental Impacts of BN

Recall the transition operator $\mathcal{T}$ for straight network, the only fixed point is $\rho^* = 1$, which indicates that the different input will correlate as layer increases,

$$\lim_l \mathcal{K}^l(x, x') = \rho^* = 1 \quad \forall x, x' \in \mathcal{X}. \qquad (8)$$

This frozen NNKG kernel also dominates the value of NTK (see Equation (7)) and hinders convergence of network. Xiao, Pennington, and Schoenholz (2019) characterize the network trainability by analyzing the spectrum of NTK. They divide the network training dynamic into three different phases according to the value $\chi_1 = \dot{\mathcal{T}}(1)$, and prove the following theorem.

Theorem 1. *[(Xiao, Pennington, and Schoenholz 2019)] The limiting condition number κ of NTK is characterized into three parts: 1) the ordered phase with $\chi_1 < 1$ has divergent κ, 2) the critical line with $\chi_1 = 1$ has condition number linear to training data-size $\kappa \sim |\mathcal{X}|$, 3) the chaotic phase with $\chi_1 > 1$ has constant limitation that $\kappa \to 1$.*

In over-parameterized networks, the condition number of NTK controls the trainability. Only in chaotic phase, network can achieve linear convergent rate. For straight ReLU

networks, the stable initialization cannot achieve chaotic NTK, such that its training process is usually troubling.

Proposition 1. *With definitions in Theorem 1, the NTK of a deep FCNN is in ordered phase or critical line with $\chi_1 \leq 1$, where equality holds when parameters are initialized with $(\sigma_w, \sigma_b) = (\sqrt{2}, 0)$.*

Note that the derivative of dual activation in Equation (4) is $\mathrm{d}\hat{\phi}(1)/\mathrm{d}\rho = 1/2$, combining this fact with Equation (3) derives this proposition. The critical initialization $(\sigma_w, \sigma_b) = (\sqrt{2}, 0)$ is the default setting for ReLU network as it stabilizes gradient propagation without vanishing or exploding (Pennington, Schoenholz, and Ganguli 2017). However, this initialization also freezes the NNGP kernel and leads to nearly singular NTK.

Besides the slow convergence rate, the straight network also leads to inferior accuracy. An intuitive explanation comes from lower network expressivity. As network outputs similar value on each channel, as shown in Figure 2, the channels with positive value behave like linear unit and negative channels are totally shut down. Therefore, straight ReLU network is more like a narrower linear system, which is incapable to fit complex functions.

Furthermore, we show that BN networks are in chaotic regime. We study the NNGP kernel and NTK in BN network. Batch normalization whitens intermediate activations to avoid the frozen NNGP kernel. We consider the common setting where BN layer locates behind FC layer. For simplicity, we consider training with full-batch and the BN operator is defined as:

$$BN(h^l) = \frac{h^l - m^l}{\sqrt{\nu^l}}, \; m^l = \mathbb{E}_{\mathcal{X}}[h^l(x)], \nu^l = \mathbb{E}_{\mathcal{X}}(h^l(x) - m^l)^2.$$
(9)

After BN layer, each channel is normalized to have the same distribution with zero mean and unit variance. As a result, the outputs of BN networks are expected to decorrelate and the NNGP kernel tends to an identical matrix.

Theorem 2. *For a deep FCNN with BN layers following each FC layer, the criterion converges in probability that $\chi_1 \xrightarrow{p} \frac{1}{1-1/\pi}$ as layer width goes to infinity, such that BN networks are in chaotic regime.*

Please refer to Appendix A for proof. BN is believed to reduce the ICS that mathematically refers to unfreezing the NNGP kernel in our opinion. Theorem 2 implies that one of the fundamental benefit from BN layer is to reduce the condition number of NTK and improve convergence rate.

Transition Operation by Weight Mean

Our weight mean operation subtracts the average value along each output channel. Moreover, we also rescale the weight parameters by multiplying $1/\sqrt{1-1/\pi}$ to satisfy the stable condition. This scaling compensates the variance decrease as we subtract mean values along each channel. We firstly show the experimental results to prove that the weight mean works similarly as BN layers. Figure 3 exhibits distributions on networks with BN layers and our weight mean
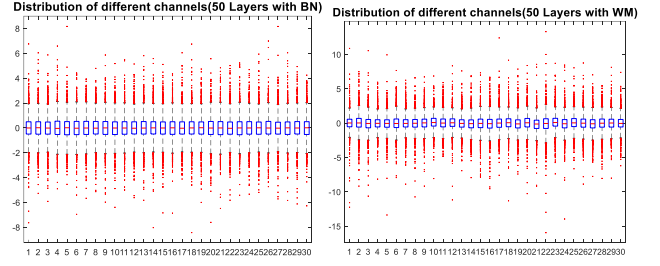

Figure 3: BN layer and weight mean operation unfreeze NNGP kernel. This figure plots activation distributions in the 50-th layers with BN layers (left) and weight mean operation (right). Activation in each channel follows the same distribution.

operation. Both two methods lessen the inter-channel variation and decorrelate data.

To analyze the mechanism of mean operation, We consider weight matrix $\mathbf{W}$ with shape (n_o, n_i) and denote its post-mean matrix as $\bar{\mathbf{W}} = \mathbf{W} - w\mathbf{1}^T$, where w is an n_o-dimension vector. Following the definition in Equation (2), the transition operator $\bar{\mathcal{T}}$ is calculated by:

$$\bar{\mathcal{T}}(\rho) = \frac{\sigma_w^2}{n_i n_o} \mathbb{E}_{(u,v)\sim\Omega_{n_i}(\rho)} \left(\phi(u)^T \bar{\mathbf{W}}^T \bar{\mathbf{W}} \phi(v) \right) + \sigma_b^2.$$
(10)

There is an intuitive explanation about how weight mean reduces correlation. Since each component of weight matrix is initialized as Gaussian random variable, the mean vector w is also Gaussian with variance scaled by layer width n. In ReLU network, activations are clipped to be non-negative values, such that the input of FC layer $\phi(u)$ contains large positive component. As a result, weight mean diminishes a positive value $w\mathbf{1}^T\phi(u)$ for every channel and decorrelate data. For transition operator, we have the following results.

Lemma 1. *For ReLU activation $\phi(x) = max(x,0)$, and define $\Phi(\rho) = \mathbb{E}_{(u,v)\sim\Omega_{n_i}(\rho)} \left(\phi(u)^T \mathbf{W}^T \mathbf{W} \phi(v) \right)$, $\bar{\Phi}(\rho) = \mathbb{E}_{(u,v)\sim\Omega_{n_i}(\rho)} \left(\phi(u)^T \bar{\mathbf{W}}^T \bar{\mathbf{W}} \phi(v) \right)$, the following equation holds*

$$\mathbb{E}_W[\bar{\Phi}(\rho)] = \frac{n_i - 1}{n_i} \mathbb{E}_W[\Phi(\rho) - \Phi(0)].$$
(11)

The proof of Lemma 1 is provided in Appendix A. The norm of output is reduced by subtracting a constant value. From Equation (11), we quantify the decline of norm of activations. In general configuration of deep networks, bias terms are initialized to zero (*i.e.*, $\sigma_b = 0$). We rescale the initial variance of weight σ_w to achieve stable condition $\bar{\mathcal{T}}(1) = 1$.

Theorem 3. *Considering zero-bias configuration, 1) straight deep ReLU networks are stably initialized by $\sigma_w^2 = 2$, and have transition operator $\mathcal{T}(\rho) = 2\hat{\phi}(\rho)$ with $\chi_1 = 1$; 2) deep ReLU networks with weight mean are stably initialized by $\sigma_w^2 = \frac{2n}{(n-1)(1-1/\pi)}$ where n is the input dimension, and have $\bar{\mathcal{T}}(\rho) = \frac{\hat{\phi}(\rho) - \hat{\phi}(0)}{\hat{\phi}(1) - \hat{\phi}(0)}$ with $\chi_1 = \frac{1}{2(\hat{\phi}(1) - \hat{\phi}(0))} = \frac{1}{1-1/\pi}$.*
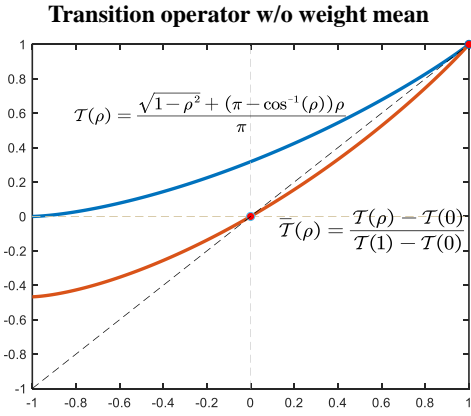
Transition operator w/o weight mean

$$\mathcal{T}(\rho) = \frac{\sqrt{1-\rho^2} + (\pi - \cos^{-1}(\rho))\rho}{\pi}$$

$$\overline{\mathcal{T}}(\rho) = \frac{\mathcal{T}(\rho) - \mathcal{T}(0)}{\mathcal{T}(1) - \mathcal{T}(0)}$$

Figure 4: Transition operator for networks with and without weight mean operation. Operator $\mathcal{T}$ for straight network (blue curve) has stable fixed point at $\rho^* = 1$ and freezes NNGP kernel. While the stable fixed point of operator $\overline{\mathcal{T}}$ (orange curve) locates at $\rho^* = 0$ and mean weight turns network into chaotic regime.

Figure 4 plots the function relationships and fixed points of transition operators with and without weight mean. Weight mean changes the fixed stable point from $\rho^* = 1$ to $\rho^* = 0$, which indicates that input data decrease their correlation as depth increases and the input data become decorrelated. As for NTK, the criterion χ_1 with weight mean is identical to BN networks and implies chaotic kernel. Therefore, networks with weight mean have the same convergence rate as BN layers.

4   Implementation of MimicNorm

In this paper, we propose *MimicNorm* to replace BN layers and reduce memory consumption. There are also some potential benefits from eliminating intermediate BN layers, such as stable training for micro-batch scenario. The main components of *MimicNorm* are weight mean for convolutional layers and one last BN layer, as illustrated in Figure 1.

Weight Mean Operation

Our weight mean operation subtracts mean values along each channel, similar to Weight Standardization (Qiao et al. 2019). Besides, we modify the initialization according to Theorem 3, with a scaling factor of $\frac{1}{\sqrt{1-1/\pi}} \approx 1.2$ in straight forward structure. As for skip-connection, we add one scalar multiplier in the end of each residual branch, similar to Skip-Init De and Smith (2020). We initialize this scalar to generate normalized activations. Considering the residual network
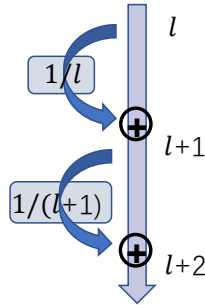

Figure 5: Batch Normalization suppresses variance in l-th residual branch by a factor $1/l$.

illustrated in Figure 5, the input of the l-th residual block is expected to have accumulated variance l. And to normalize the output, this residual branch implicitly downscales variance by l. In this way, the learnable scalar has an initial value $1/\sqrt{l}$. Our method also works well for lightweight networks like ShuffleNet, which contains depthwise convolutional layers. Our weight mean ignores those layers, since the total number (depending on kernel size, usually $3 \times 3 = 9$) of weights in one channel is too small.

Last BN Layer

Empirical results have shown that the straight networks are only trainable with smaller learning rates (Bjorck et al. 2018). We observe that a larger learning rate usually causes output explosion within several iterations. We conjecture that it is the gradient that encourages variance of outputs. An intuitive explanation is that two contradictory forces control the evolution of variance, namely scattering inter-class data and congregating intra-class ones. A larger learning rate weights the scattering effect and may lead to divergence. BN layer auto-tunes the learning rates, as backpropagated gradients shrink by variance. However, this auto-tuned learning rate can be implemented by one last BN layer. Therefore, the unit variance operations in former layers are unnecessary.

As for implementation, our last BN layer locates after the final fully-connected layer and normalizes (B, C) output matrix, where B is batchsize and C is the number of categories. Compared with former BN layers, the last one is much lighter and requires less resources. Besides, we also disable affine parameters to keep the same distribution of classification categories.

5   Experiments

We employ MimicNorm on various networks and compare it with original BN networks and other implementation without BN layers, *i.e.* Fixup initialization (Zhang, Dauphin, and Ma 2019). Fixup method initializes classification and residual branch to be zero, downscales weight parameters in convolutional layers, and induces scalar biases (and multiplier) before and after convolutional layers. They avoid divergence by repressing output to be zero and succeed in training residual networks without BN layers. Our experiments are evaluated in CIFAR-100 dataset and ImageNet dataset. We also do an ablation study to investigate the isolated impact. Besides, some empirical discussions about weight mean and last BN layer are attached in Appendix B. Experiments in this paper are implemented with Pytorch framework, version 1.5, and code is available at https://github.com/Kidkey/MimicNorm.

Evaluations on CIFAR-100

In this section, we show results of our method and BN networks on CIFAR-100 dataset, which are evaluated by one GPU with 12GB memory, GeForce RTX 2080 Ti. Our optimizer is firstly warmed up within 2 epochs from 0 to 0.1, and then uses multi-step scheduler decayed by 0.1 at epochs [80,110,135]. The default batchsize is 256. However, ResNet50 requires more memory to deploy, while

Model	Method	Accuracy (%)	Memory
	BatchNorm	67.90 ± 0.14	2971MB
VGG11	**MimicNorm**	67.87 ± 0.20	2851MB
	NoNorm	65.33 ± 0.08	2831MB
	BatchNorm	72.28 ± 0.05	3745MB
VGG16	**MimicNorm**	72.47 ± 0.02	3489MB
	NoNorm	$68.97 \pm 0.36^{*}$	3469MB
	BatchNorm	74.72 ± 0.31	4447MB
ResNet18	Fixup	70.48 ± 0.18	4743MB
	MimicNorm	74.96 ± 0.14	4157MB
	BatchNorm	78.53 ± 0.26	8533MB
ResNet50	Fixup	$72.81 \pm 0.10^{*}$	10283MB
	MimicNorm	77.91 ± 0.20	7533MB
SqueezeNet	BatchNorm	70.88 ± 0.08	3687MB
	MimicNorm	71.15 ± 0.10	2693MB
ShuffleNetV2	BatchNorm	71.10 ± 0.49	3915MB
	MimicNorm	70.93 ± 0.13	3141MB

* training is unstable, and fails to converge sometimes.

Table 1: Comparison with various network structures on CIFAR-100 dataset. We evaluate three times and calculate mean and std of accuracy. On all structures, our method achieves comparable accuracy and beneficial memory consumption.

we halve the batchsize into 128. Table 1 presents the accuracy and memory consumption on straight forward networks (VGG11/16 (Simonyan and Zisserman 2015)), skip connected networks (ResNet18/50 (He et al. 2016)) and lightweight networks (SqueezeNet (Hu, Shen, and Sun 2018), ShuffleNetV2 (Ma et al. 2018)).

For straight structures, our method exhibits the same impacts of BN layers, *i.e.* improving accuracy and stabilizing training process. VGG16 network may divergence with learning rate 0.1, and both our method and BN network are stable. We observe that our method only requires 20MB more memory, which demonstrates the low complexity of the last BN layer. For the ResNet structures, we also compare with Fixup initialization, which is able to train deep ResNets without normalization. Our method saves about 10% memory with similar accuracy. While Fixup initialization dramatically deteriorates the performance and increases memory consumption. Despite Fixup initialization removes all BN layers, this method induces learnable scalar parameters. To calculate the gradients of scalar parameters, Pytorch implementation stores lots of temporary tensors consuming huge memory. We also validate our method on lightweight networks, which are more sensitive to hardware resources. Our method obtains more than 20% memory saving with the same results. To the best of our knowledge, our method is the first attempt to train lightweight networks without intermediate BN layers.

Ablation Studies

We further separately evaluate the impact of weight mean and the last BN layer. As illustrated in previous sections, weight mean operation unfreezes the NNGP kernel and
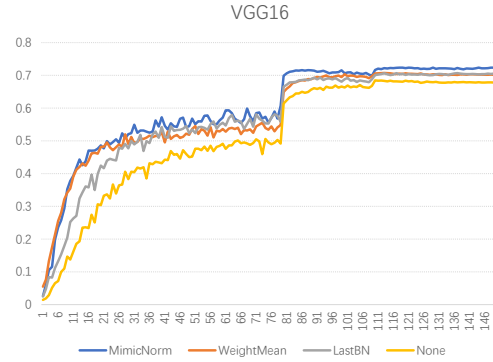

Figure 6: Ablation study on VGG16 network. we observe that both weight mean and last BN layer contribute to the improvement of performance. while weight mean accelerates training at the initial phrase.

Learning Rate	None	Weight Mean	Mimic Norm	Batch Norm
0.01	22.12	34.81	38.39	37.82
0.02	26.76	43.07	44.32	40.49
0.05	25.63	42.63	49.53	37.97
0.1	1.24	36.49	46.92	32.4
0.2	1.77	1.00	39.41	25.5
0.5	1.00	1.00	22.03	8.26
1.0	1.00	1.00	9.83	3.93

Table 2: Best accuracy (%) on the first 10 epochs with different learning rate. The last BN layer makes network trainable among larger learning rates.

accelerates training process. The last BN layer auto-tunes learning rate to avoid exploded variance.

Figure 6 compares the training processes of different methods on VGG16. In the beginning, weight mean enables faster training, in accordance with the theoretical prediction. The last BN layer also accelerates training and improves accuracy, while our method combining weight mean and the last BN layer achieves the best accuracy. Table 2 evaluates the stability of the last BN layer, where all the methods are test using different learning rates. For small learning rates, all the methods succeed in training and weight mean achieves faster convergence. For large learning rates, last BN layer is essential for training. Moreover, Figure 6 shows our method yields a larger accuracy within the first 10 epochs in comparison to BN implementation.

Despite our analysis is based on straight forward structures, the empirical results show that weight mean also accelerates training for ResNets. Ablation results on Resnet18 and Resnet50 are presented in Figure 7. Note that our method adds a scalar at the end of each residual branch, and initializes it according to SkipInit method. Our method surpasses state-of-the-art methods, with faster training and better accuracy. We also observe one last BN layer in ResNet18 generates better results than weight mean. However, for deeper network, one last BN layer shows inferiority than
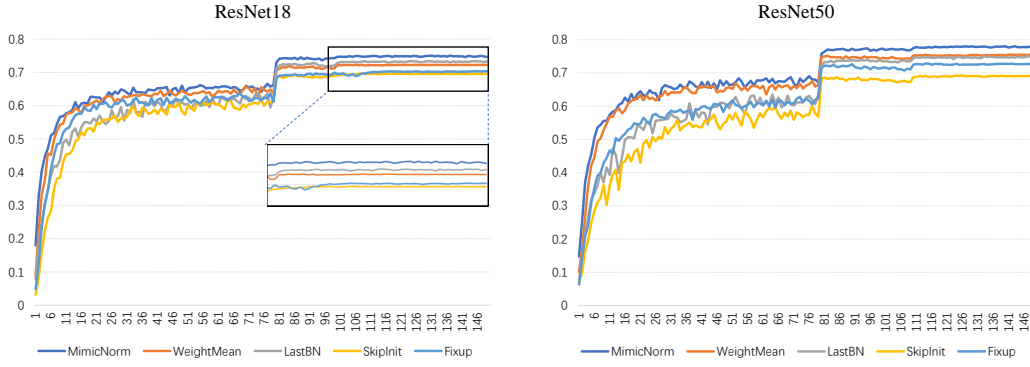
Figure 7: Comparison of training curves on ResNet18 and ResNet50. Weight mean accelerates training at the beginning and last BN layer improves accuracy. Besides, our method also gains large superiority over Fixup initialization.

Model	Method	Accuracy (%)	Total memory
VGG16*	BatchNorm	73.89	62162MB
	MimicNorm	73.31	56410MB
	NoNorm	71.59	56388MB
ResNet18	BatchNorm	69.76	8057MB
	MimicNorm	70.15	5603MB
ResNet50	BatchNorm	76.48	27083MB
	Fixup	72.40	29267MB
	MimicNorm	76.84	22489MB
ResNet101*	BatchNorm	77.37	37168MB
	MimicNorm	77.52	30363MB
ShuffleNet V2_0.5	BatchNorm	59.77	4385MB
	MimicNorm	58.31	3603MB

* Model deployment with batchsize 256 requires 2 GPUs.

Table 3: Test results on ImageNet dataset with batchsize 256. Results show our method gets comparable accuracy with BN layers, and exceeds Fixup method largely. Note for ResNet50/110 and ShuffleNetV2, Our method saving nearly 20% memory.

weight mean because of the much worse performance at the initial phase. Thus, weight mean is more important for deep networks as it avoids the frozen NNGP kernel, and one last layer is able to compensate most accuracy drop for shallow networks.

Evaluations on ImageNet

On ImageNet dataset, we experiment on Tesla V100 GPUs. We follow exactly the same data augmentation strategy as official Pytorch implementation: random flipping and sampling, and resizing to 224×224 images. The training optimizer is also general SGD optimizer with momentum factor 0.9 and weight decay 1e-4. We train network for 90 epochs with the first 2 epochs warming up and decrease learning rate from 0.1 to 0.0001 at milestones [30, 60, 85].

Table 3 shows the results on several network structures. Similar to our finding on CIFAR-100 dataset, our method has similar accuracy on all structures and saves much more memory. We save about 10% memory for VGG16, less than ResNets and lightweight structures, because the VGG nets have less BN layers and larger size of feature maps. On ResNet structures, we achieve impressive performance, higher accuracy than BN networks and about 20% memory saving. In comparison, Fixup initialization is inferior both in accuracy and memory. As for ShuffleNetV2, we modify the training strategy according to the original paper (Zhang et al. 2018) as the results under our default setting is much less than claimed accuracy, both for our method and BN implementation. We use less data augmentation and train for 200 epochs. Our method slightly underperforms BN implementation, which may be a result of depthwise convolutions. Note that our method skips weight mean for depthwise convolutional layers and there is no regularization for them. Despite the slight deficit, we save nearly 20% total memory even for lightweight network structure.

6 Conclusion and Discussion

In this paper, we propose MimicNorm to replace BN layers and reduce memory consumption, which consists of two heart components, weight mean and last BN layer. We theoretically demonstrate that, in an infinite width DNN, weight mean and BN layers unfreeze the NNGP kernel leading to chaotic NTK, and enable linear convergence rate. For the finite width network, NTK describes the one-order approximation at initialization. Thus, networks with chaotic NTK converge faster at first several epochs. And empirical evidences confirm the theoretical prediction. On the other hand, last BN layer adjusts learning rate by the variance of output. We show last BN layer stabilizes training process on larger learning rates. Additionally, we also find that the auto-tuned learning rate leads to better performance and its dynamic is still unclear. In our implementation, last BN layer could be folded into loss function, which implies that a variance-aware loss function may facilitate network training.

References

Arora, S.; Du, S. S.; Hu, W.; Li, Z.; Salakhutdinov, R. R.; and Wang, R. 2019. On exact computation with an infinitely

wide neural net. In *Advances in Neural Information Processing Systems*, 8141–8150.

Arora, S.; Li, Z.; and Lyu, K. 2019. Theoretical analysis of auto rate-tuning by batch normalization. In *7th International Conference on Learning Representations*. New Orleans, LA, USA.

Ba, J. L.; Kiros, J. R.; and Hinton, G. E. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450* .

Bietti, A.; and Mairal, J. 2019. On the inductive bias of neural tangent kernels. In *Advances in Neural Information Processing Systems*, 12893–12904.

Bjorck, N.; Gomes, C. P.; Selman, B.; and Weinberger, K. Q. 2018. Understanding batch normalization. In *Advances in Neural Information Processing Systems 31*, 7694–7705. Montreal, QC, Canada.

Cho, Y.; and Saul, L. K. 2009. Kernel methods for deep learning. In *Advances in neural information processing systems*, 342–350.

Daniely, A.; Frostig, R.; and Singer, Y. 2016. Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity. In *Advances In Neural Information Processing Systems*, 2253–2261.

De, S.; and Smith, S. L. 2020. Batch normalization biases deep residual networks towards shallow paths. *arXiv preprint arXiv:2002.10444* .

Edelman, A. 1988. Eigenvalues and condition numbers of random matrices. *SIAM journal on matrix analysis and applications* 9(4): 543–560.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7132–7141.

Ioffe, S.; and Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning*, 448–456. Lille, France.

Jacot, A.; Gabriel, F.; and Hongler, C. 2018. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in neural information processing systems*, 8571–8580.

Jacot, A.; Gabriel, F.; and Hongler, C. 2019. Freeze and chaos for dnns: an NTK view of batch normalization, checkerboard and boundary effects. *arXiv preprint arXiv:1907.05715* .

Lee, J.; Xiao, L.; Schoenholz, S.; Bahri, Y.; Novak, R.; Sohl-Dickstein, J.; and Pennington, J. 2019. Wide neural networks of any depth evolve as linear models under gradient descent. In *Advances in neural information processing systems*, 8572–8583.

Luo, P.; Wang, X.; Shao, W.; and Peng, Z. 2019. Towards Understanding Regularization in Batch Normalization. In *International Conference on Learning Representations*.

Ma, N.; Zhang, X.; Zheng, H.-T.; and Sun, J. 2018. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European conference on computer vision (ECCV)*, 116–131.

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

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

Santurkar, S.; Tsipras, D.; Ilyas, A.; and Madry, A. 2018. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems 31*, 2483–2493. Montreal, QC, Canada.

Simonyan, K.; and Zisserman, A. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd Int. Conf. Learn. Rep.* San Diego, CA, USA.

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

Wishart, J. 1928. The generalised product moment distribution in samples from a normal multivariate population. *Biometrika* 32–52.

Wu, Y.; and He, K. 2018. Group normalization. In *The European Conference on Computer Vision (ECCV)*, 3–19. Munich, Germany.

Xiao, L.; Pennington, J.; and Schoenholz, S. S. 2019. Disentangling trainability and generalization in deep learning. *arXiv preprint arXiv:1912.13053* .

Yang, G. 2019. Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation. *arXiv preprint arXiv:1902.04760* .

Zhang, G.; Wang, C.; Xu, B.; and Grosse, R. 2019. Three Mechanisms of Weight Decay Regularization. In *International Conference on Learning Representations*.

Zhang, H.; Dauphin, Y. N.; and Ma, T. 2019. Fixup initialization: Residual learning without normalization. In *7th International Conference on Learning Representations*. New Orleans, LA, USA.

Zhang, X.; Zhou, X.; Lin, M.; and Sun, J. 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 6848–6856.