Supplementary Materials: Layer-wise Conditioning Analysis in Exploring the Learning Dynamics of DNNs

Lei Huang[1] Jie Qin[1] Li Liu[1] Fan Zhu[1] Ling Shao[2,1]

[1]Inception Institute of Artificial Intelligence (IIAI), Abu Dhabi, UAE
[2]Mohamed bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE
{lei.huang, jie.qin, li.liu, fan.zhu, ling.shao}@inceptioniai.org

A Proof of Theorems

In this section, we provide proofs for the three theorems in the paper. For completeness, we provide notations described in the paper again. We consider a Multiple Layer Perceptron (MLP), $f_\theta(\mathbf{x})$, represented as a layer-wise linear and nonlinear transform, as follows:

$$\mathbf{h}_k = \mathbf{W}_k \mathbf{x}_{k-1}, \quad \mathbf{x}_k = \phi(\mathbf{h}_k), \quad k = 1, ..., K, \tag{1}$$

where $\mathbf{x}_0 = \mathbf{x}$, $\mathbf{W}_k \in \mathbb{R}^{d_k \times d_{k-1}}$ and the learnable parameters $\theta = \{\mathbf{W}_k, k = 1, ..., K\}$. To simplify the notation, we set $\mathbf{x}_K = \mathbf{h}_K$ as the output of the network $f_\theta(\mathbf{x})$. We denote the covariance matrix of the layer input $\Sigma_\mathbf{x} = \mathbb{E}_{p(\mathbf{x})}(\mathbf{x}\mathbf{x}^T)$ and the covariance matrix of the layer output-gradient $\Sigma_{\nabla\mathbf{h}} = \mathbb{E}_{q(\mathbf{y}|\mathbf{x})}(\frac{\partial \ell}{\partial \mathbf{h}}^T \frac{\partial \ell}{\partial \mathbf{h}})$. Here, we follow matrix notation, where all the vectors are column vectors, except the gradient vectors, which are row vectors.

Proposition 1. *Given $\Sigma_\mathbf{x}$, $\Sigma_{\nabla\mathbf{h}}$ and $F = \Sigma_\mathbf{x} \otimes \Sigma_{\nabla\mathbf{h}}$, we have: 1) $\lambda_{max}(F) = \lambda_{max}(\Sigma_\mathbf{x}) \cdot \lambda_{max}(\Sigma_{\nabla\mathbf{h}})$; 2) $\kappa(F) = \kappa(\Sigma_\mathbf{x}) \cdot \kappa(\Sigma_{\nabla\mathbf{h}})$.*

Proof. The proof is mainly based on the conclusion from Theorem 4.2.12 in [5], which is restated as follows:

Lemma 1. *Let $\mathbf{A} \in \mathbb{R}^{m\times m}$ and $\mathbf{B} \in \mathbb{R}^{n\times n}$. Furthermore, let λ_a be the arbitrary eigenvalue of $\mathbf{A}$ and λ_b be the arbitrary eigenvalue of $\mathbf{B}$. We have $\lambda_a \cdot \lambda_b$ as an eigenvalue of $\mathbf{A} \otimes \mathbf{B}$. Furthermore, any eigenvalue of $\mathbf{A} \otimes \mathbf{B}$ arises as a product of the eigenvalues of $\mathbf{A}$ and $\mathbf{B}$.*

Based on the definitions of $\Sigma_\mathbf{x}$ and $\Sigma_{\nabla\mathbf{h}}$, we have that $\Sigma_\mathbf{x}$ and $\Sigma_{\nabla\mathbf{h}}$ are positive semidefinite. Therefore, all the eigenvalues of $\Sigma_\mathbf{x}/\Sigma_{\nabla\mathbf{h}}$ are non-negative. Let $\lambda(\mathbf{A})$ denote the eigenvalue of matrix $\mathbf{A}$. Based on Lemma 1, we have $\lambda(F) = \lambda(\Sigma_\mathbf{x})\lambda(\Sigma_{\nabla\mathbf{h}})$. Since $\lambda(\Sigma_\mathbf{x})$ and $\lambda(\Sigma_{\nabla\mathbf{h}})$ are non-negative, we thus have $\lambda_{max}(F) = \lambda_{max}(\Sigma_\mathbf{x}) \cdot \lambda_{max}(\Sigma_{\nabla\mathbf{h}})$. Similarly, we can prove that $\lambda_{min}(F) = \lambda_{min}(\Sigma_\mathbf{x}) \cdot \lambda_{min}(\Sigma_{\nabla\mathbf{h}})$. We thus have $\kappa(F) = \kappa(\Sigma_\mathbf{x}) \cdot \kappa(\Sigma_{\nabla\mathbf{h}})$.

Theorem 1. *Given a rectifier neural network (Eqn. 1) with nonlinearity $\phi(\alpha\mathbf{x}) = \alpha\phi(\mathbf{x})$ ($\alpha > 0$), if the weight in each layer is scaled by $\widehat{\mathbf{W}}_k = \alpha_k \mathbf{W}_k$ ($k = 1, ..., K$ and $\alpha_k > 0$), we have the scaled layer input: $\widehat{\mathbf{x}}_k = (\prod_{i=1}^{k} \alpha_i)\mathbf{x}_k$. Assuming that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_K} = \mu \frac{\partial \mathcal{L}}{\partial \mathbf{h}_K}$, we have the output-gradient: $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \mu(\prod_{i=k+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, and weight-gradient: $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = (\mu \prod_{i=1, i\neq k}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k}$, for all $k = 1, ..., K$.*

Proof. (1) We first demonstrate that the scaled layer input $\widehat{\mathbf{x}}_k = (\prod_{i=1}^{k} \alpha_i)\mathbf{x}_k$ ($k = 1, ..., K$), using mathematical induction. It is easy to validate that $\widehat{\mathbf{h}}_1 = \alpha_1 \mathbf{h}_1$ and $\widehat{\mathbf{x}}_1 = \alpha_1 \mathbf{x}_1$. We assume that $\widehat{\mathbf{h}}_t = (\prod_{i=1}^{t} \alpha_i)\mathbf{h}_t$ and $\widehat{\mathbf{x}}_t = (\prod_{i=1}^{t} \alpha_i)\mathbf{x}_t$ hold, for $t = 1, ..., k$. When $t = k + 1$, we have

$$\widehat{\mathbf{h}}_{k+1} = \widehat{\mathbf{W}}_{k+1}\widehat{\mathbf{x}}_{k+1} = \alpha_{k+1}\mathbf{W}_{k+1}(\prod_{i=1}^{k} \alpha_i)\mathbf{x}_k = (\prod_{i=1}^{k+1} \alpha_i)\mathbf{h}_{k+1}. \tag{2}$$

We thus have

$$\widehat{\mathbf{x}}_{k+1} = \phi(\widehat{\mathbf{h}}_{k+1}) = \phi((\prod_{i=1}^{k+1} \alpha_i)\mathbf{h}_{k+1}) = (\prod_{i=1}^{k+1} \alpha_i)\phi(\mathbf{h}_{k+1}) = (\prod_{i=1}^{k+1} \alpha_i)\mathbf{x}_{k+1}. \tag{3}$$

By induction, we have $\widehat{\mathbf{x}}_k = (\prod_{i=1}^{k} \alpha_i)\mathbf{x}_k$, for $k = 1, ..., K$. We also have $\widehat{\mathbf{h}}_k = (\prod_{i=1}^{k} \alpha_i)\mathbf{h}_k$ for $k = 1, ..., K$.

(2) We then demonstrate that the scaled output-gradient $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \mu(\prod_{i=k+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$ for $k = 1, ..., K$. We also provide this using mathematical induction. Based on back-propagation, we have

$$\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{k-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}\mathbf{W}_k, \quad \frac{\partial \mathcal{L}}{\partial \mathbf{h}_{k-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_{k-1}}\frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{h}_{k-1}}, \tag{4}$$

and

$$\frac{\partial \widehat{\mathbf{x}}_{k-1}}{\partial \widehat{\mathbf{h}}_{k-1}} = \frac{\partial(\prod_{i=1}^{k-1} \alpha_i)\mathbf{x}_{k-1}}{\partial(\prod_{i=1}^{k-1} \alpha_i)\mathbf{h}_{k-1}} = \frac{(\prod_{i=1}^{k-1} \alpha_i)\partial \mathbf{x}_{k-1}}{(\prod_{i=1}^{k-1} \alpha_i)\partial \mathbf{h}_{k-1}} = \frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{h}_{k-1}}, \quad k = 2, ..., K. \tag{5}$$

Based on the assumption that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_K} = \mu \frac{\partial \mathcal{L}}{\partial \mathbf{h}_K}$, we have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_K} = \mu(\prod_{i=K+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_K}$[1].

[1] We denote $\prod_{i=a}^{b} \alpha_i = 1$ if $a > b$.

We assume that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_t} = \mu(\prod_{i=t+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ holds, for $t = K, ...k$. When $t = k-1$, we have

$$\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{x}}_{k-1}} = \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k}\widehat{\mathbf{W}}_k = \mu(\prod_{i=k+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}\alpha_k \mathbf{W}_k = \mu(\prod_{i=k}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{k-1}}. \tag{6}$$

We also have

$$\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_{k-1}} = \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{x}}_{k-1}}.\frac{\partial \widehat{\mathbf{x}}_{k-1}}{\partial \widehat{\mathbf{h}}_{k-1}} = \mu(\prod_{i=k}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{x}_{k-1}}.\frac{\partial \mathbf{x}_{k-1}}{\partial \mathbf{h}_{k-1}} = \mu(\prod_{i=k}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{k-1}}. \tag{7}$$

By induction, we thus have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \mu(\prod_{i=k+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, for $k = 1, ..., K$.

(3) Based on $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}^T \mathbf{x}_{k-1}^T$, $\widehat{\mathbf{x}}_k = (\prod_{i=1}^{k} \alpha_i)\mathbf{x}_k$ and $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \mu(\prod_{i=k+1}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, it is easy to prove that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = (\mu \prod_{i=1, i\neq k}^{K} \alpha_i)\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k}$ for $k = 1, ..., K$.

Theorem 2. *Under the same condition of Theorem 1, for the normalized network with* $\mathbf{h}_k = \mathbf{W}_k \mathbf{x}_{k-1}$ *and* $\mathbf{s}_k = BN(\mathbf{h}_k)$*, we have:* $\widehat{\mathbf{x}}_k = \mathbf{x}_k$, $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k}$*, for all* $k = 1, ..., K$.

Proof. (1) Following the proof in Theorem 1, by mathematical induction, it is easy to demonstrate that $\widehat{\mathbf{h}}_k = \alpha_k \mathbf{h}_k$, $\widehat{\mathbf{s}}_k = \mathbf{s}_k$ and $\widehat{\mathbf{x}}_k = \mathbf{x}_k$, for all $k = 1, ..., K$.

(2) We also use mathematical induction to demonstrate $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$ for all $k = 1, ..., K$.

We first show the formulation of the gradient back-propagating through each neuron of the BN layer as:

$$\frac{\partial \mathcal{L}}{\partial h} = \frac{1}{\sigma}(\frac{\partial \mathcal{L}}{\partial s} - \mathbb{E}_B(\frac{\partial \mathcal{L}}{\partial s}) - \mathbb{E}_B(\frac{\partial \mathcal{L}}{\partial s}s)s), \tag{8}$$

where σ is the standard deviation and $\mathbb{E}_B$ denotes the expectation over mini-batch examples. We have $\hat{\sigma}_K = \alpha_K \sigma_K$ based on $\widehat{\mathbf{h}}_K = \alpha_K \mathbf{h}_K$. Since $\widehat{\mathbf{s}}_K = \mathbf{s}_K$, we have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{s}}_K} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_K}$. Therefore, we have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_K} = \frac{\sigma_K}{\hat{\sigma}_K}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_K} = \frac{1}{\alpha_K}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_K}$ from Eqn. 8.

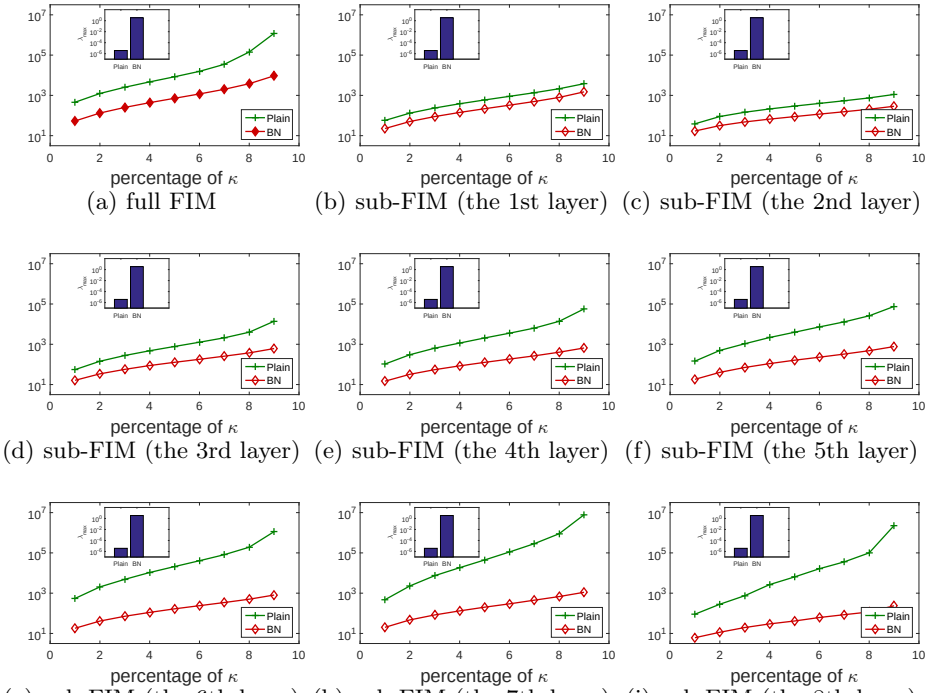
Assume that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_t} = \frac{1}{\alpha_t}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_t}$ for $t = K, ..., k+1$. When $t = k$, we have:

$$\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{x}}_k} = \frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_{k+1}}\widehat{\mathbf{W}}_{k+1} = \frac{1}{\alpha_{k+1}}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_{k+1}}\alpha_{k+1}\mathbf{W}_{k+1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_k}. \tag{9}$$

Following the proof for Theorem 1 , it is easy to get $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{s}}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_k}$. Based on $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{s}}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_k}$ and $\widehat{\mathbf{s}}_k = \mathbf{s}_k$, we have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \frac{\sigma_k}{\hat{\sigma}_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$ from Eqn. 8.

By induction, we have $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, for all $k = 1, ..., K$.

(3) Based on $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = \frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}^T \mathbf{x}_{k-1}^T$, $\widehat{\mathbf{x}}_k = \mathbf{x}_k$ and $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{h}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$, we have that $\frac{\partial \mathcal{L}}{\partial \widehat{\mathbf{W}}_k} = \frac{1}{\alpha_k}\frac{\partial \mathcal{L}}{\partial \mathbf{W}_k}$, for all $k = 1, ..., K$.

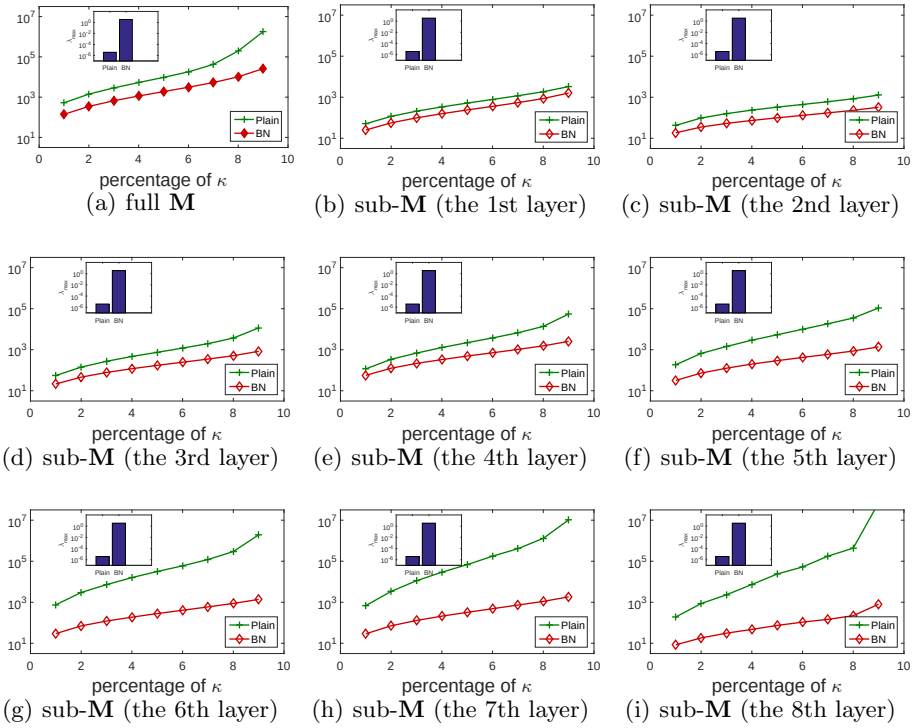


(a) full FIM (b) sub-FIM (the 1st layer) (c) sub-FIM (the 2nd layer)

(d) sub-FIM (the 3rd layer) (e) sub-FIM (the 4th layer) (f) sub-FIM (the 5th layer)

(g) sub-FIM (the 6th layer) (h) sub-FIM (the 7th layer) (i) sub-FIM (the 8th layer)

Fig. I. Conditioning analysis for unnormalized ('Plain') and normalized ('BN') networks. We show the maximum eigenvalue λ_{max} and the generalized condition number κ_p for comparison between the full FIM $\mathbf{F}$ and sub-FIMs $\{F_k\}$. The experiments are performed on an 8-layer MLP with 24 neurons in each layer, for MNIST classification. The input image is center-cropped and resized to 12×12 to remove uninformative pixels. We report the corresponding spectrum at random initialization [8].

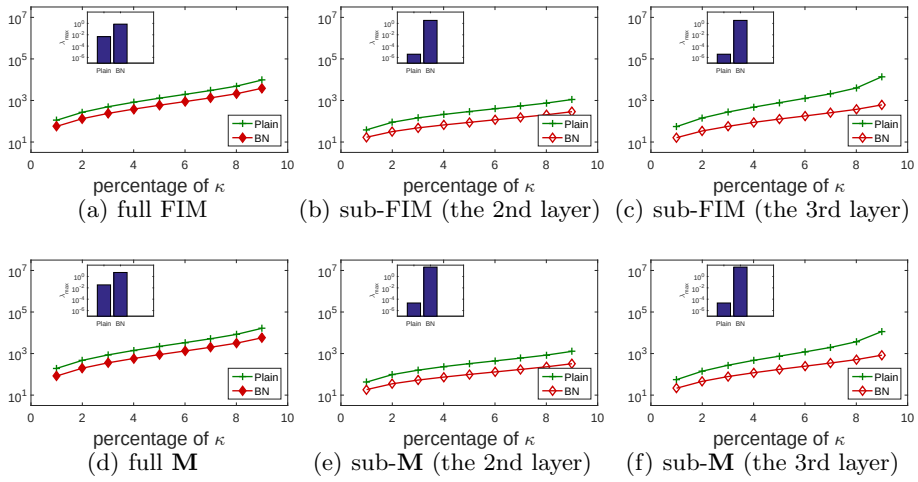


(a) full $\mathbf{M}$ (b) sub-$\mathbf{M}$ (the 1st layer) (c) sub-$\mathbf{M}$ (the 2nd layer)

(d) sub-$\mathbf{M}$ (the 3rd layer) (e) sub-$\mathbf{M}$ (the 4th layer) (f) sub-$\mathbf{M}$ (the 5th layer)

(g) sub-$\mathbf{M}$ (the 6th layer) (h) sub-$\mathbf{M}$ (the 7th layer) (i) sub-$\mathbf{M}$ (the 8th layer)

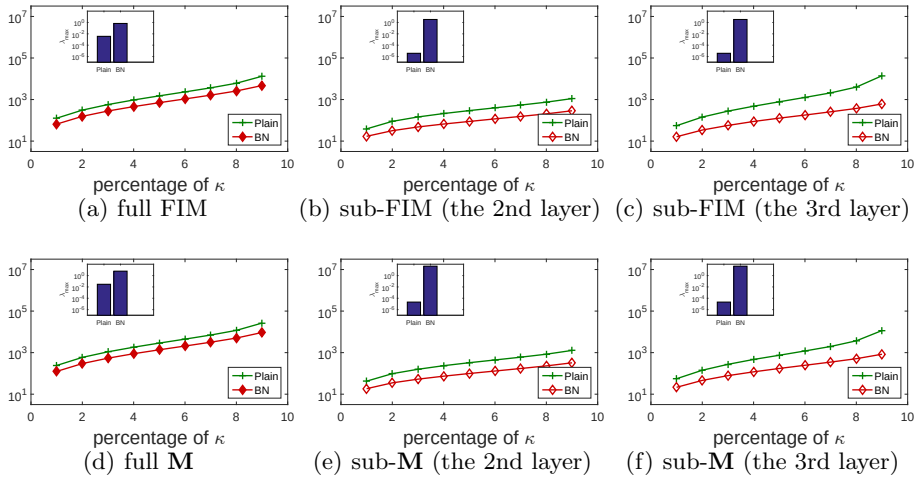
Fig. II. Conditioning analysis for unnormalized ('Plain') and normalized ('BN') networks. We show the maximum eigenvalue λ_{max} and the generalized condition number κ_p for comparison between the full second moment matrix of sample gradient $\mathbf{M}$ and sub-$\{\mathbf{M}_k\}$. The experiments are performed on an 8-layer MLP with 24 neurons in each layer, for MNIST classification. The input image is center-cropped and resized to 12×12 to remove uninformative pixels. We report the corresponding spectrum at random initialization [8].



(a) full FIM (b) sub-FIM (the 2nd layer) (c) sub-FIM (the 3rd layer)

(d) full $\mathbf{M}$ (e) sub-$\mathbf{M}$ (the 2nd layer) (f) sub-$\mathbf{M}$ (the 3rd layer)

Fig. III. Conditioning analysis for unnormalized ('Plain') and normalized ('BN') networks. The experiments are performed on a 4-layer MLP with 24 neurons in each layer, for MNIST classification. The input image is center-cropped and resized to 12×12 to remove the uninformative pixels. We report the corresponding spectrum at random initialization [8].



(a) full FIM (b) sub-FIM (the 2nd layer) (c) sub-FIM (the 3rd layer)

(d) full $\mathbf{M}$ (e) sub-$\mathbf{M}$ (the 2nd layer) (f) sub-$\mathbf{M}$ (the 3rd layer)

Fig. IV. Conditioning analysis for unnormalized ('Plain') and normalized ('BN') networks. The experiments are performed on a 4-layer MLP with 36 neurons in each layer, for MNIST classification. The input image is center-cropped and resized to 12×12 to remove the uninformative pixels. We report the corresponding spectrum at random initialization [8].

B Comparison Between the Analyses with Full Curvature and Sub-curvature Matrices

In Section 3 of the paper, we conduct experiments to analyze the training dynamics of the unnormalized ('Plain') and batch normalized [6] ('BN') networks, on an 8-layer MLP, by analyzing the spectrum of the full Fisher Information Matrix (FIM) and sub-FIMs. We only report the sub-FIMs with respect to the 3rd and 6th layer, due to the page limit. Here, we show the corresponding results for all sum-FIMs in Figure I.

We also conduct experiments to analyze 'Plain' and 'BN', using the second moment matrix of sample gradient $\mathbf{M}$. The results are shown in Figure II. We have the same observation as when using the full FIM. It is interesting to use sub-Hessian for the conditioning analysis, since Hessian provides a good characterization of the landscapes [9]. We believe that the max eigenvalue of sub-Hessian can also indicate the magnitude of the weight-gradient in each layer, like the sub-FIM/sub-M. However, the general condition number shown in this paper is not well defined for sub-Hessian, since it has negative eigenvalues.

We further conduct experiments on a 4-layer MLP with 24 neurons in each layer, and a 4-layer MLP with 36 neurons in each layer. The corresponding results with different curvature matrices are shown in Figures III and IV, respectively.

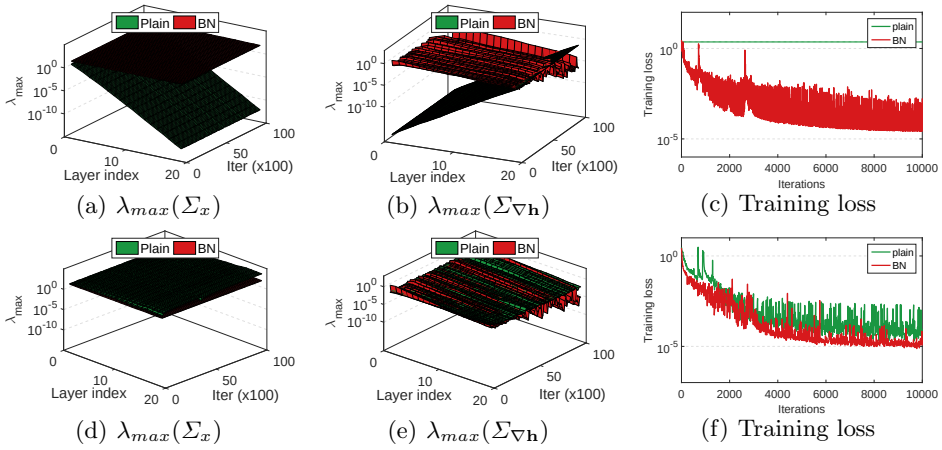
Complexity Analysis Here, we provide the complexity analysis. For simplifying notation, we consider a L layer MLP with d neurons in each layer. The example number is N. For full conditioning analysis, the cost includes computing the curvature matrix ($O(N(Ld^2)^2)$) and the eigen-decomposition ($O((Ld^2)^3)$). The computation cost is reduced to $O(NL(d^2)^2) + O(L(d^2)^3)$ for layer-wise conditioning analysis, and further reduced to $O(NLd^2) + O(Ld^3)$ using our efficient approximation. Note that we only consider naive implementation without any tricks in acceleration (e.g., using implicitly restarted Lanczos method [9]).

C More Experiments in Exploring Batch Normalized Networks

In this section, we provide more experimental results relating to the exploration of batch normalization (BN) [6] by layer-wise conditioning analysis, which is discussed in Section 4 of the paper. We include experiments that train neural networks with Stochastic Gradient Descent (SGD), experiments relating to weight domination and experiments relating to dying/full neurons, as discussed in the paper.

C.1 Experiments with SGD

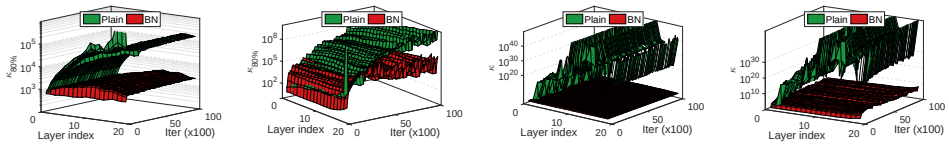
Here, we perform experiments on the Multiple Layer Perceptron (MLP) for MNIST classification and Convolutional Neural Networks (CNNs) for CIFAR-10 and ImageNet [1] classification.



(a) $\lambda_{max}(\Sigma_x)$ (b) $\lambda_{max}(\Sigma_{\nabla \mathbf{h}})$ (c) Training loss

(d) $\lambda_{max}(\Sigma_x)$ (e) $\lambda_{max}(\Sigma_{\nabla \mathbf{h}})$ (f) Training loss

Fig. V. Analysis of the magnitude of the layer input (indicated by $\lambda_{max}(\Sigma_x)$) and layer output-gradient (indicated by $\lambda_{max}(\Sigma_{\nabla \mathbf{h}})$). We use the SGD with a batch size of 1024 to train the 20-layer MLP for classification. The results of (a)(b)(c) are under random initialization [8], while (d)(e)(f) use He-initialization [2].



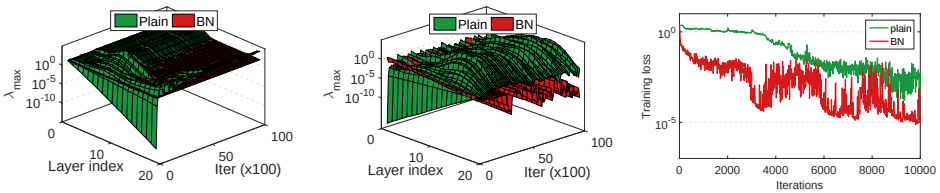
(a) $\kappa_{80\%}(\Sigma_x)$ (b) $\kappa_{80\%}(\Sigma_{\nabla \mathbf{h}})$ (c) $\kappa_{100\%}(\Sigma_x)$ (d) $\kappa_{100\%}(\Sigma_{\nabla \mathbf{h}})$

Fig. VI. Analysis on the condition number of the layer input (indicated by $\kappa_p(\Sigma_x)$) and layer output-gradient (indicated by $\kappa_p(\Sigma_{\nabla \mathbf{h}})$). The experimental setups are the same as in Figure V.

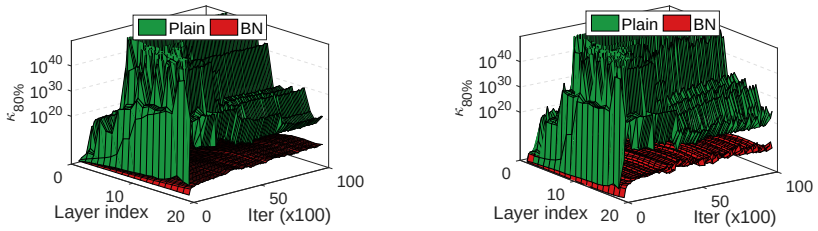
MLP for MNIST Classification Here, we use the same experimental setup as the experiments described in the paper for MNIST classification, except that we use SGD with a batch size of 1024. The results are shown in Figures V and VI. We obtain similar results as those obtained using full gradient descent, as described in Section 4 of the paper.

We also conduct experiments using the Adam optimizer [7]. We again use a batch size of 1024 to train the 20-layer MLP for classification. We report the best training loss among the learning rates in $\{0.001, 0.0005, 0.0001\}$. The results under random initialization [8] are shown in Figure VII. We observe that: 1) 'Plain' with the Adam optimizer can well adjust the magnitude of the layer input (Figure VII (a)) and layer output-gradient (Figure VII (b)) during training, compared to 'Plain' with the naive SGD optimizer (Figure V (a) and (b)); 2) 'BN' can better stabilize the training; 3) 'BN' has better conditioning than 'Plain' during training.

CNN for CIFAR-10 Classification We perform a layer-wise conditioning analysis on the VGG-style and residual network [3] architectures. Note that we view the activation in each spatial location of the feature map as an independent

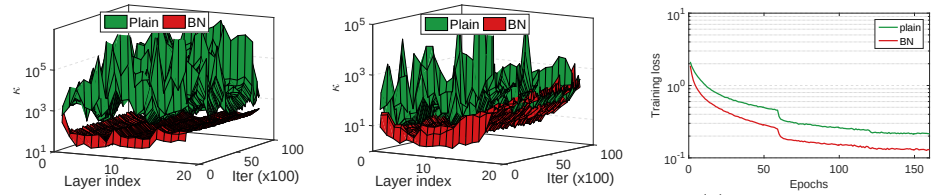


(a) $\lambda_{max}(\Sigma_x)$ (b) $\lambda_{max}(\Sigma_{\nabla \mathbf{h}})$ (c) Training loss



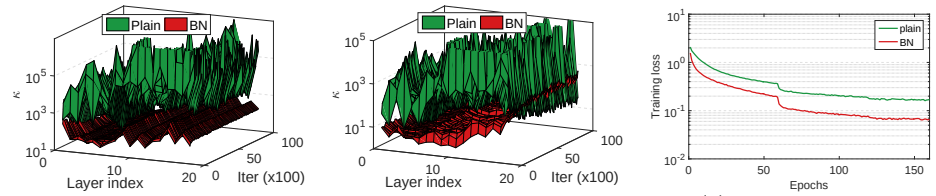
(d) $\kappa_{80\%}(\Sigma_x)$ (e) $\kappa_{80\%}(\Sigma_{\nabla \mathbf{h}})$

Fig. VII. Layer-wise conditioning analysis results with Adam optimizer [7]. We use a batch size of 1024 to train the 20-layer MLP for classification. Figures (a) and (b) show the magnitude of the layer input and layer output-gradient, respectively. Figure (c) shows the training loss with respect to the epochs. Figures (d) and (e) show the condition number of the layer input and layer output-gradient, respectively.



(a) $\kappa_{100\%}(\Sigma_x)$ (b) $\kappa_{100\%}(\Sigma_{\nabla \mathbf{h}})$ (c) Training loss

Fig. VIII. Layer-wise conditioning analysis on the VGG-style network for CIFAR-10 classification. Figures (a) and (b) show the condition number of the layer input (indicated by $\kappa_p(\Sigma_x)$) and layer output-gradient (indicated by $\kappa_p(\Sigma_{\nabla \mathbf{h}})$), respectively. Figure (c) shows the training loss with respect to the epochs.



(a) $\kappa_{100\%}(\Sigma_x)$ (b) $\kappa_{100\%}(\Sigma_{\nabla \mathbf{h}})$ (c) Training loss

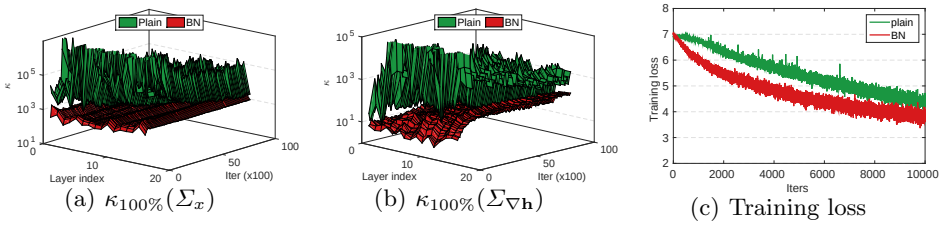
Fig. IX. Layer-wise conditioning analysis on the residual network [3] for CIFAR-10 classification. Figures (a) and (b) show the condition number of the layer input (indicated by $\kappa_p(\Sigma_x)$) and layer output-gradient (indicated by $\kappa_p(\Sigma_{\nabla \mathbf{h}})$), respectively. Figure (c) shows the training loss with respect to the epochs.

example, when calculating the covariance matrix of the convolutional layer input and output-gradient. This process is similar to the process proposed in BN to normalize the convolutional layer [6].

We use the 20-layer residual network described in the paper [3] for CIFAR-10 classification. The VGG-style network is constructed based on the 20-layer residual network, removing the residual connections.

We use the same setups as described in [3], except that we do not use weight decay in order to simplify the analysis and run the experiments on one GPU. Since the unnormalized networks (including the VGG-style and residual network) do not converge with the large learning rate of 0.1, we run additional experiments with a learning rate of 0.01, and report these results.

Figures VIII and Figure IX show the results for the VGG-Style and residual network, respectively. We obtain the same observations as those made for the MLP for MNIST classification.



(a) $\kappa_{100\%}(\Sigma_x)$ (b) $\kappa_{100\%}(\Sigma_{\nabla \mathbf{h}})$ (c) Training loss

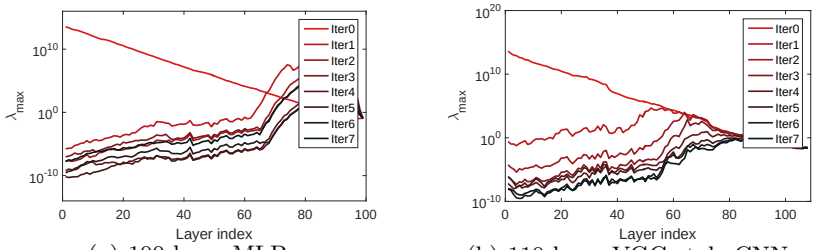
Fig. X. Layer-wise conditioning analysis on the residual network [3] for ImageNet classification. Figures (a) and (b) show the condition number of the layer input (indicated by $\kappa_p(\Sigma_x)$) and layer output-gradient (indicated by $\kappa_p(\Sigma_{\nabla \mathbf{h}})$), respectively. Figure (c) shows the training loss with respect to the training iterations.

CNN for ImageNet Classification We also perform a layer-wise conditioning analysis on ImageNet using a 18-layer residual network [3]. We use the same setups as described in [3], except that we run the experiments on one GPU. Figure X show the results. We also obtain the same observations as those made for the MLP for MNIST classification.

C.2 Experiments Relating to Weight Domination

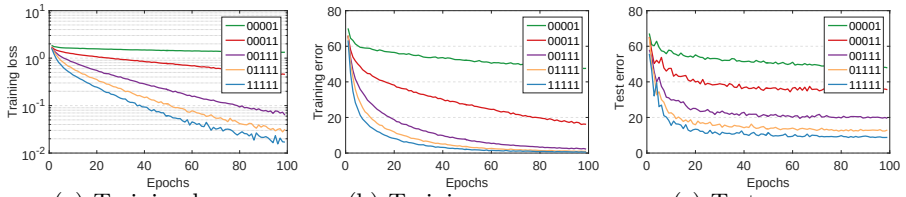
Gradient Explosion of BN In Section 4.1 of the paper, we mention that, even for the network with BN, there is still the possibility that the magnitude of the weight in certain layers is significantly increased. Here, we provide the experimental results.

We conduct experiments on a 100-layer batch normalized MLP with 256 neurons in each layer for MNIST classification. We calculate the maximum eigenvalues of the sub-FIMs, and provide the results for the first seven iterations



(a) 100-layer MLP (b) 110-layer VGG-style CNN

Fig. XI. Experiments relating to gradient explosion of BN in deep networks without residual connections. We show the results of (a) a 100-layer MLP for MNIST classification and (b) a 110-layer VGG-style CNN for CIFAR-10 classification.



(a) Training loss (b) Training error (c) Test error

Fig. XII. Exploring the effectiveness of weight domination on a 16-layer VGG network with BN for CIFAR-10 classification. We simulate weight domination in a specific layer by blocking its weight updates. We denote '0' in the legend as the state of weight dominant (the first digit represents the first three consecutive convolutional layers).

in Figure XI (a). We observe that the weight-gradient has exponential explosion at initialization ('Iter0'). After a single step, the first-step gradients dominate the weights due to gradient explosion in lower layers, hence the exponential growth in the magnitude of the weight. This increased magnitude of weight leads to small weight gradients ('Iter1' to 'Iter7'), which is caused by BN, as discussed in the paper. Therefore, some layers (especially the lower layers) of the network enter the state of *weight domination*. We obtain similar observations on the 110-layer VGG-style network for CIFAR-10 classification, as shown in Figure XI (b).

Investigation of Weight Domination Weight domination sometimes harms the learning of the network, because this state limits the representational ability of the corresponding layer. We conducted experiments on a five-layer MLP and provided the results in the paper. Here, we also conduct experiments on CNNs for CIFAR-10 datasets, shown in Figure XII. We observe that the network with certain layers being in states of *weight domination* can still decrease the loss, but with degenerated performance.

C.3 Experiments Relating to Dying Neurons

In Section 4.2 of the paper, we mentioned that 'Plain' has dying/full neurons, and the number of dying/full neurons increases as the layer number increases. Figure XIII shows the details of this phenomenon.

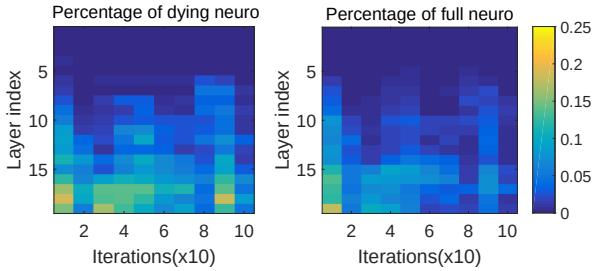
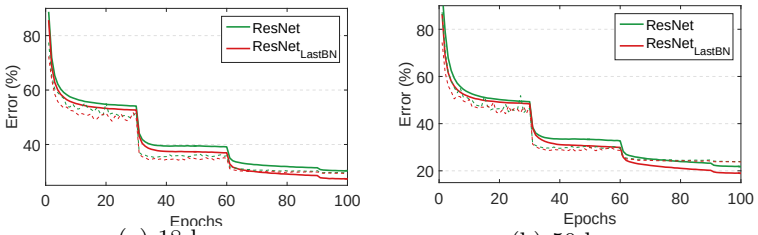


Fig. XIII. Dying and full neurons during training. The experiments are performed on a 20-layer MLP with 256 neurons in each layer, for MNIST classification. We show the results corresponding to the He-initialization [2].



(a) 18-layer (b) 50-layer

Fig. XIV. Comparison of top-1 training errors (solid lines) and test errors (dashed lines) of ResNet and ResNet$_{LastBN}$ on ImageNet.

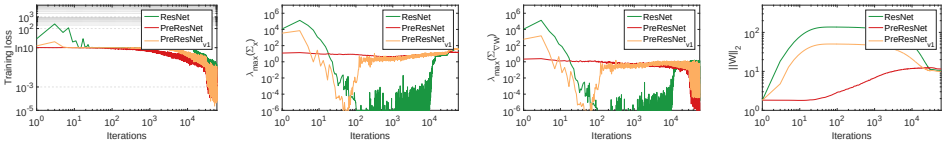
D More Results for Deep Residual Network

D.1 Results on ImageNet classification

As mentioned in Section 5.1 of the paper, we validate the effectiveness of ResNet$_{LastBN}$ on the large-scale ImageNet classification, with 1000 classes [1]. Here, we provide the details.

We use the official 1.28M training images as the training set, and evaluate the top-1 classification errors on the validation set with 50k images. We perform the experiments using the 18-layer and 50-layer networks. We follow the same setup as described in [3], except that 1) we train over 100 epochs with an extra lowered learning rate at the 90th epoch; 2) we use one GPU for the 18-layer network and four GPUs for 50-layer network.

Figures XIV (a) and (b) show the training results for the 18-layer and 50-layer residual networks, respectively. We find ResNet$_{LastBN}$ has a better optimization efficiency than ResNet, at both depths. Table A shows the validation errors. ResNet$_{LastBN}$ has a slightly improved performance, under the standard hyper-parameters configuration. Due to the improved optimization efficiency of ResNet$_{LastBN}$, the advantage can be further amplified if we add the magnitude of the regularization, *e.g.*, when we use a weight decay (WD) of 0.0002 or add a dropout (DR) of 0.3, ResNet$_{LastBN}$ achieves better performance.



(a) Training loss (b) $\lambda_{\Sigma_{\mathbf{x}}}$ (c) $\lambda_{\Sigma_{\frac{\partial \mathcal{L}}{\partial \mathbf{W}}}}$ (d) $\|\mathbf{W}\|_2$

Fig. XV. Conditioning analysis on the last linear layer of 1202-layer 'ResNet', 'PreResNet' and 'PreResNet$_{v1}$' for CIFAR-10 classification.

Table A. Comparison of top-1 validation errors (%, single model and single-crop) on the 18- and 50-layer residual networks for ImageNet classification.

	depth-18			depth-50		
Method	standard	WD=0.0002	DR=0.3	standard	WD=0.0002	DR=0.3
ResNet	29.78	30.07	30.62	23.97	24.37	23.81
ResNet$_{LastBN}$	**29.38**	**28.96**	**29.32**	**23.76**	**23.49**	**23.47**

D.2 Conditioning Analysis on PreResNet

As mentioned in Section 5.2 of the paper, we investigate which component in PreResNet [4] (*e.g.*, the pre-activation or the extra BN layer) benefits the optimization behaviors, using our conditioning analysis. Here, we provide the details. We use the exact same setups as the previous experiments on ResNet. Figure XV shows the results of conditioning analysis on the last linear layer of the 1202-layer network.

D.3 More Results of Putting a BN Layer before the Last Linear Layer

We also observe that the method of putting a BN layer before the last linear layer is useful in other architectures, in which the last linear layer's input $\mathbf{x}$ has significantly varying magnitude (indicated by $\lambda_{\Sigma_{\mathbf{x}}}$) during training (*e.g.*, the ResNets case). We conduct experiments on the VGG-style networks (without BN) and ResNets (without BN) for CIFAR-10 classification. The setup is the same as in Section C.1. We vary the depth ranging in 20, 44, 56, 110. We observe that for all depths, putting a BN layer before the last linear layer improves the performance .

Table B. Experiments of putting a BN layer before the last linear layer on the VGG-style networks (without BN) and ResNets (without BN) for CIFAR-10 classification. We report the test error (%).

method	depth-20	depth-44	depth-56	depth-110
VGG	14.88 ± 0.47	33.90 ± 7.1	89.93 ± 0.12	90 ± 0
VGG$_{LastBN}$	**12.89 ± 0.20**	**18.40 ± 0.90**	**23.40 ± 1.74**	**73.99 ± 16**
ResNet	11.21 ± 0.14	9.90 ± 0.31	9.48 ± 0.07	8.93 ± 0.22
ResNet$_{LastBN}$	**10.31 ± 0.25**	**8.99 ± 0.25**	**8.61 ± 0.05**	**8.37 ± 0.07**

References

1. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR (2009) 7, 12
2. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: ICCV (2015) 8, 12
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016) 8, 9, 10, 12
4. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: ECCV (2016) 13
5. Horn, R.A., Johnson, C.A.: Topics in Matrix Analysis. Cambridge University Press (1991) 1
6. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML (2015) 7, 10
7. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. CoRR **abs/1412.6980** (2014) 8, 9
8. LeCun, Y., Bottou, L., Orr, G.B., Müller, K.R.: Effiicient backprop. In: Neural Networks: Tricks of the Trade. pp. 9–50 (1998) 4, 5, 6, 8
9. Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T.: Visualizing the loss landscape of neural nets. In: NeurIPS (2018) 7