PyHessian: Neural Networks Through the Lens of the Hessian

Zhewei Yao*†, Amir Gholami*†, Kurt Keutzer*, Michael W. Mahoney*
*University of California, Berkeley
†Equal contribution.
{zheweiy, amirgh, keutzer, mahoneymw}@berkeley.edu

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378171

***Abstract*—We present PyHessian, a new scalable framework that enables fast computation of Hessian (i.e., second-order derivative) information for deep neural networks. PyHessian enables fast computations of the top Hessian eigenvalues, the Hessian trace, and the full Hessian eigenvalue/spectral density; it supports distributed-memory execution on cloud/supercomputer systems; and it is available as open source [1]. This general framework can be used to analyze neural network models, including the topology of the loss landscape (i.e., curvature information) to gain insight into the behavior of different models/optimizers. As an example, we analyze the effect of residual connections and Batch Normalization layers on the trainability of neural networks. One recent claim, based on simpler first-order analysis, is that residual connections and Batch Normalization make the loss landscape "smoother," thus making it easier for Stochastic Gradient Descent to converge to a good solution. Our second-order analysis, easily enabled by PyHessian, shows new finer-scale insights, demonstrating that while conventional wisdom is sometimes validated, in other cases it is simply incorrect. In particular, we find that Batch Normalization does not necessarily make the loss landscape smoother, especially for shallow networks.**

I. Introduction

Residual neural networks [12] (ResNets) are widely used Neural Networks (NNs) for various learning tasks. The two main architectural components of ResNets are residual connections [12] and Batch Normalization (BN) layers [13]. However, going beyond motivating stories to characterize precisely when and why these two popular architectural ingredients help or hurt training/generalization—especially in terms of *measurable* properties of the model—is still largely unsolved. Relatedly, characterizing whether other suggested architectural changes will help or hurt training/generalization is still done in a largely ad hoc manner (e.g., it is often motivated by plausible but untested intuitions, and it is typically not characterized in terms of measurable properties of the model).

In this work, we present and apply PyHessian, an open-source scalable framework with which one can directly analyze Hessian information, i.e., second-derivative information, w.r.t. model parameters, in order to address these and related questions. PyHessian computes Hessian information by applying known techniques from Numerical Linear Algebra (NLA) [4, 10, 17] and Randomized NLA (RandNLA) [3, 7, 8, 18, 25, 27] (that are approximate but come with rigorous theory). PyHessian enables computing Hessian information—including top Hessian eigenvalues, Hessian trace, and Hessian eigenvalue spectral density (ESD), and it supports distributed framework—allowing distributed-memory execution on both cloud (e.g., AWS, Google Cloud) and supercomputer systems, for fast and efficient Hessian computation. As an application of PyHessian, we use it to analyze the impact of residual connection and BN on the trainability of NNs, leading to new insights.

In more detail, our main contributions are the following:

- We introduce PyHessian, a new framework for direct and efficient computation of Hessian information, including the top eigenvalue, the trace, and the full ESD, in NNs [1]. We provide a self-contained description of the methods implemented in PyHessian, and we apply PyHessian to study how residual connections and BN affect training.
- We observe that removing the BN layer from ResNet (denoted below as ResNet$_{-BN}$) leads to *rapid increase of the Hessian spectrum* (the top eigenvalue, the trace, and the ESD support range). This increase is significantly more rapid for deeper models. See Figure 2, 3, 4, and 5.
- We observe that, for shallow networks (ResNet20), removing the BN layer results in a flatter Hessian spectrum, as compared to standard ResNet20 with BN. See Figure 2 and 3. This observation is the *opposite of the common belief* that the addition of BN layers make the loss landscape smoother (which we observe to hold only for deeper networks).
- We observe that, for deep networks (ResNet32/38), removing BN results in converging to sharper local minima, as compared to ResNet with BN. See Figure 1, 2, 4, 5.
- We show that removing residual connections from ResNet generally makes the top eigenvalue, the trace, and the Hessian ESD support range increase slightly. This increase is consistent for both shallow and deep models (ResNet20/32/38). See Figure 2, 3, 4, 5.
- We perform Hessian analysis for different stages of ResNet models (details in Section IV-A), and we find that, generally, *BN is more important for the final stages* than for earlier stages. In particular, removing BN from the last stage significantly degrades testing performance, with a strong correlation with the Hessian trace. See the comparison between orange curve and blue curve in Figure 6, and the accuracy reported in Table II.

Due to space constraints, we do not include all of our results here; more detailed results on PyHessian may be found in the longer technical report version of this paper [26].

The basic components implemented in PyHessian (power

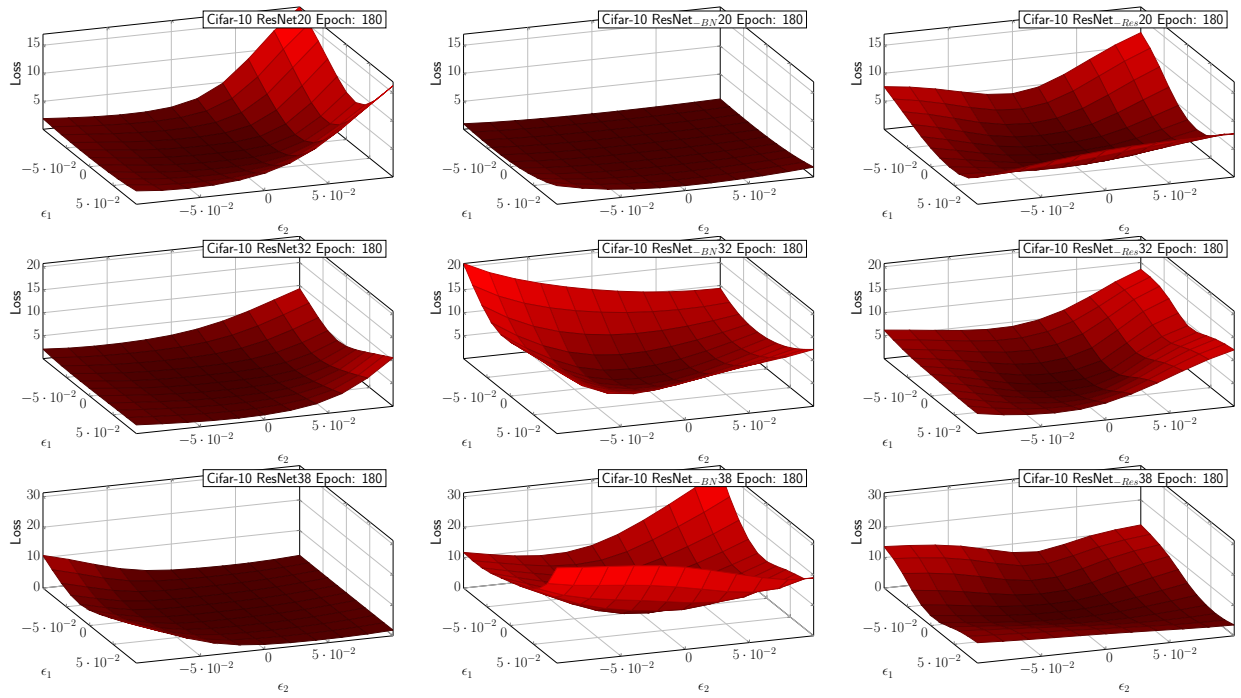

Fig. 1: *The parametric loss landscapes of ResNet20 (top), ResNet32 (middle), and ResNet38 (bottom) on Cifar-10 are plotted by perturbing the model parameters at the end of training across the first and second Hessian eigenvector. Results for the original ResNet architecture (left), ResNet without BN (middle; denoted asResNet$_{-BN}$), and ResNet without residual connection (right; denoted as ResNet$_{-Res}$). It can be clearly seen that removing BN from ResNet20 actually leads to a smoother loss landscape, which is opposite to the common belief that adding BN leads to a smoother loss landscape [24]. We only observed the claimed smoothness property for the deeper ResNet32/38 model (second/third row). This smoothness can be quantified by measuring the trace of the Hessian operator, which is reported in Figure 2, as well as the full Hessian ESD, shown in Figure 3, 4 and 5.*

method, randomized Hutchinson method, and stochastic Lanczos method) are well-known, but (for many users) they are not easy to use. One of the challenges in using more powerful second-order methods to perform NN analysis is that they are sophisticated and their implementation can be subtle (as opposed to heavily-parameterized first order methods, which can be implemented in a few lines of code for a class project). This partly explains the near ubiquitousness of first-order methods in machine learning and NN analysis. We developed PyHessian to address this issue, and we have open-sourced PyHessian to encourage reproducibility and as a scalable framework that can be used for research on second-order methods. Since the first release of PyHessian, several other high-quality implementations of second-order methods have appeared and been applied to NN problems [2, 28].

II. Related work

Hessian and Large-scale Hessian Computation: Hessian-based analysis/computation is widely used in scientific computing. However, due to the (incorrect, but in our experience widespread) belief that Hessian-based computations are infeasible for large NN problems, the majority of work in machine learning (except for quite small problems) only performs the first-order analysis.[1] However, using implicit or matrix-free methods, *it is not necessary to form the Hessian matrix explicitly in order to extract second-order information* [6, 20]. Instead, it is possible to use stochastic methods from RandNLA to extract this information, without explicitly forming the Hessian matrix. For example, [3, 4] proposed fast algorithms for trace computation; and [17, 25] provided efficient randomized algorithms to estimate the ESD of a positive semi-definite matrix. These algorithms only require an oracle for computing the product of the Hessian matrix with a given random vector. *It is possible to compute this so-called "matvec" and extract Hessian information without explicitly forming the Hessian [5, 19]. In particular, using the so-called R-operator, the Hessian matvec can be computed with the same computational graph used for backpropagating the gradient [19].*

Hessian eigenvalues of small NN models were analyzed [22, 23]; and the work of [21] studied the geometry of NN loss landscapes by computing the distribution of Hessian eigenvalues at critical points. More recently, [27] used a deflated power-iteration method to compute the top eigenvalues for deep NNs during training. Moreover, the work of [9] measured the Hessian ESD, based on the stochastic Lanczos

[1]The naïve view arises since the Hessian matrix is of size (say) $m \times m$. Thus, like most linear algebra computations, exact full spectral computations (which are sufficient but never necessary for NN problems) cost $\mathcal{O}(m^3)$ time.

algorithm of [17, 25]. Here, we extend the analysis of [9, 27] by studying how the depth of the NN model as well as its architecture affect the Hessian spectrum (in terms of top eigenvalue, trace, and full ESD). Furthermore, we also perform a block diagonal Hessian spectrum analysis, and we observe a fine-scale relationship between the Hessian spectrum and the impact of adding/removing residual connections and BN.

Hessian-based analysis has also been used in the context of NN training and inference. For example, [15] analytically computes Hessian information for a single linear layer and uses the Hessian spectrum to determine the optimal learning rate to accelerate training. In [14], the authors approximated the Hessian as a diagonal operator and used the inverse of this diagonal matrix to prune NN parameters.

A major limitation in most of this prior work is that tests are typically restricted to small/simple NN models that may not be representative of NN workloads that are encountered in practice. This is in part due to the lack of a scalable and easily programmable framework that could be used to test second-order methods for a wide range of state-of-the-art models. Addressing this is the main motivation behind our development of PyHessian, which is released as open-source software and is available to researchers [1]. In this paper, we illustrate how PyHessian can be used for analyzing the NN behaviour during training, even for very deep state-of-the-art models.

Residual Connections and Batch Normalization: Residual connections [12] and BN [13] are two of the most important ingredients in modern convolutional NNs. There have been different hypotheses offered for why these two components help training/generalization. First, the empirical study of [16] found that deep NNs with residual connections exhibit a significantly smoother loss landscape, as compared to models without residual connections. This was achieved by the so-called filter-normalized random direction method to plot 3D loss landscapes, i.e., not through direct analysis of the Hessian spectrum. This result is interesting, but it is hard to draw conclusions with perturbations in two directions, for a model that has millions of parameters (and thus millions of possible perturbation directions).

Second, the motivation for why BN helps training/generalization was originally attributed to reducing the so-called Internal Covariance Shift (ICS) [13]. However, this was disputed in the recent study of [24]. In particular, the work of [24] used first-order analysis to analyze the loss landscape, and found that adding a BN layer results in a smoother loss landscape. Importantly, they found that adding BN does not reduce the so-called ICS. Again, while interesting, such first-order analysis may not fully capture the topology of the landscape (and, as we will show with our second-order analysis, *this smoothness claim is not correct in general*).

The work of [24] also performed an interesting theoretical analysis, showing a connection between adding the BN layer and the Lipschitz constant of the gradient (i.e., the top Hessian eigenvalue). It was argued that adding the BN layer leads to a smaller Lipschitz constant. However, the theoretical analysis is only valid for the per-layer Lipschitz constant, as it ignores the complex interaction between different layers. It cannot be extended to the Lipschitz constant of the entire model (and, as we will show, this result does not hold for shallow networks).

III. Methodology

For a supervised learning problem, we seek to minimize:

$$\min_{\theta} L(\theta) = \frac{1}{N}\sum_{i=1}^{N} l(M(x_i), y_i, \theta), \tag{1}$$

where $\theta \in \mathbb{R}^m$ is the learnable weight parameter, $l(M(x), y, \theta)$ is the loss function, (x, y) is the input pair, M is the NN architecture, and N is the size of training data. Below, we first provide a self-contained description of how PyHessian computes second-order statistics, and then we discuss the impact of architectural components on the model trainability.

A. Neural Network Hessian Matvec

For a NN with m parameters, the gradient of the loss w.r.t. model parameters is a vector $\frac{\partial L}{\partial \theta} = g_\theta \in \mathbb{R}^m$, and the second derivative of the loss is a matrix, $H = \frac{\partial^2 L}{\partial \theta^2} = \frac{\partial g_\theta}{\partial \theta} \in \mathbb{R}^{m\times m}$, commonly called the Hessian. A typical NN model involves millions of parameters, and thus even forming the Hessian is computationally infeasible. However, it is possible to compute properties of the Hessian spectrum *without* explicitly forming the Hessian matrix. Instead, we need an oracle to compute the application of the Hessian to a random vector v. This can be achieved by observing the following:

$$\frac{\partial g_\theta^T v}{\partial \theta} = \frac{\partial g_\theta^T}{\partial \theta} v + g_\theta^T \frac{\partial v}{\partial \theta} = \frac{\partial g_\theta^T}{\partial \theta} v = Hv. \tag{2}$$

Here, the first equality is the chain rule, the second is due to the independence of v to θ, and the third equality is the definition of the Hessian. Importantly, the cost of this Hessian matrix vector multiply (hereafter referred to as Hessian matvec) is the same as one gradient backpropagation.

Having this oracle, we can easily compute the top k Hessian eigenvalues using power iteration [27]; see Algorithm III.1. However, since the top eigenvalues may not be representative of how the loss landscape behaves, we also compute the trace and ESD of the Hessian, as described next.

Algorithm III.1: Power Iteration for Top Eigenvalue Computation

Input: Parameter: θ.
Compute the gradient of θ by backpropagation, *i.e.*, compute $g_\theta = \frac{dL}{d\theta}$.
Draw a random vector v from $N(0,1)$ (same dimension as θ).
Normalize v, $v = \frac{v}{\|v\|_2}$
for $i = 1, 2, \ldots$ **do** `// Power Iteration`
  Compute $gv = g_\theta^T v$ `// Inner product`
  Compute Hv by backpropagation, $Hv = \frac{d(gv)}{d\theta}$ `// Get Hessian vector product`
  Normalize and reset v, $v = \frac{Hv}{\|Hv\|_2}$

B. Hutchinson Method for Hessian Trace Computation

The trace of the Hessian can also be computed using RandNLA, and in particular Hutchinson's method [3, 4] for the fast computation of the trace, using only Hessian matvec computations (as given in Eq. 2). In particular, since we are interested in the Hessian, i.e., a symmetric matrix, suppose we have a random vector v, whose components are i.i.d. sampled from a Rademacher distribution (or Gaussian distribution with mean 0 and variance 1). Then, we have the identity

$$\begin{aligned} Tr(H) &= Tr(HI) = Tr(H\mathbb{E}[vv^T]) = \mathbb{E}[Tr(Hvv^T)] \\ &= \mathbb{E}[v^T Hv], \end{aligned} \tag{3}$$

where I is the identity matrix of appropriate size. That is, the trace of H can be estimated by computing $\mathbb{E}[v^T Hv]$, where we compute the expectation by drawing multiple random samples. Note that Hv can be efficiently computed from Eq. 2, and then $v^T Hv$ is simply a dot product between the Hessian matvec and the original vector v. See Algorithm III.2 for a description.

Algorithm III.2: Hutchinson Method for Trace Computation

Input: Parameter: θ.
Compute the gradient of θ by backpropagation, *i.e.*, compute $g_\theta = \frac{dL}{d\theta}$.
for $i = 1, 2, \ldots$ **do** `// Hutchinson Steps`
  Draw a random vector v from Rademacher distribution (same dimension as θ).
  Compute $gv = g_\theta^T v$ `// Inner product`
  Compute Hv by backpropagation, $Hv = \frac{d(gv)}{d\theta}$ `// Get Hessian vector product`
  Compute and record $v^T Hv$
Return the average of all computed $v^T Hv$.

C. Full Hessian Eigenvalue Spectral Density

To provide finer-grained information on the Hessian spectrum than is provided by the top eigenvalues or the trace, we need to compute the full empirical spectral density (ESD) of the Hessian eigenvalues. This is defined as

$$\phi(t) = \frac{1}{m}\sum_{i=1}^{m} \delta(t - \lambda_i), \tag{4}$$

where $\delta(\cdot)$ is the Dirac distribution and λ_i is the i-th eigenvalue of H, in descending order.

Recent work in NLA/RandNLA has provided efficient matrix-free algorithms to estimate this ESD [10, 17, 25] through Stochastic Lanczos Quadrature (SLQ). Here, we briefly describe SLQ in simple terms. This approach was also used in [9] to compute the Hessian ESD. For more details, see [10, 17, 25].

Here is a summary of our approach to compute the ESD $\phi(t)$. First, we approximate $\phi(t)$ (of Eq. 4) by $\phi_\sigma(t)$ (Eq. 5 below) by applying a Gaussian kernel (*first approximation*), and we express this in the same expectation form as in the Hutchinson algorithm (Eq. 9 below). Next, since the computation inside the expectation depends directly on t and the unknown eigenvalues (denoted by λ_is), we simplify the problem by using Gaussian quadrature (Eq. 13 below) (*second approximation*). Then, since the weights and λ_is in the Gaussian quadrature are unknown, we use the stochastic Lanczos algorithm to approximate the weights and λ_is (Eq. 14 below) (*third approximation*). Finally, we approximate the expectation of the eigenvalue distribution as a sum (Eq. 15 below) (*forth approximation*).

Algorithm III.3: Stochastic Lanczos Quadrature for ESD Computation

Input: Parameter: θ, degree q and n_v.
Compute the gradient of θ by backpropagation, *i.e.*, compute $g_\theta = \frac{dL}{d\theta}$.
for $i = 1, 2, \ldots n_v$ **do** `// Different Seeds`
  Draw a random vector v from N(0,1) and normalize it (same dimension as θ).
  Get the tri-diagonal matrix T through Lanczos algorithm.
  Compute $\tau_k^{(i)}$ and $\tilde{\lambda}_k^{(i)}$ from T
  $\phi_\sigma^{z_i} = \sum_{k=1}^{q} \tau_k f(\tilde{\lambda}_k; t, \sigma)$
Return $\phi(t) = \frac{1}{n_v}\sum_{l=1}^{n_v}\left(\sum_{i=1}^{q} \tau_i^{(l)} f(\tilde{\lambda}_i^{(l)}; t, \sigma)\right)$

In more detail, for the first approximation, we apply a Gaussian kernel, f, with variance σ^2 to Eq. 4 to obtain

$$\phi_\sigma(t) = \frac{1}{m}\sum_{i=1}^{m} f(\lambda_i; t, \sigma), \tag{5}$$

where $f(\lambda; t, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} exp(-(t-\lambda)^2/(2\sigma^2))$ is the Gaussian kernel. Clearly, $\phi_\sigma(t) \to \phi(t)$, as $\sigma \to 0$. Thus, if we had an algorithm to approximate Eq. 5, then we could take the limit and reduce the standard deviation of the Gaussian kernel to approximate Eq. 4. In our context, the question of how to compute $\phi_\sigma(t)$ amounts to computing the density distribution of the Hessian convolved with a Gaussian kernel.

To do this, observe that

$$Tr(f(H)) = Tr(Qf(\Lambda)Q^T) = Tr(f(\Lambda)), \tag{6}$$

where $Q\Lambda Q^T$ is the eigendecomposition of H, and let $f(H)$ be the matrix function, defined as

$$f(H) \triangleq Qf(\Lambda)Q^T \triangleq Q\text{diag}(f(\lambda_1), ..., f(\lambda_m))Q^T. \tag{7}$$

We can plug Eq. 6 into Eq. 5 to get

$$\phi_\sigma(t) = \frac{1}{m} Tr(f(H; t, \sigma)). \tag{8}$$

For a given value of t, the trace $Tr(f(H; t, \sigma))$ can be efficiently computed using the Hutchinson algorithm (described in §III-B). That is, we draw a random Rademacher vector v and compute the expectation $\mathbb{E}[v^T f(H; t, \sigma)v]$ to get

$$\phi_\sigma(t) = \frac{1}{m}\mathbb{E}[v^T f(H; t, \sigma)v]. \tag{9}$$

However, this is still intractable, as the trace computation needs to be repeated for every value of t (which scales with the number of model parameters).

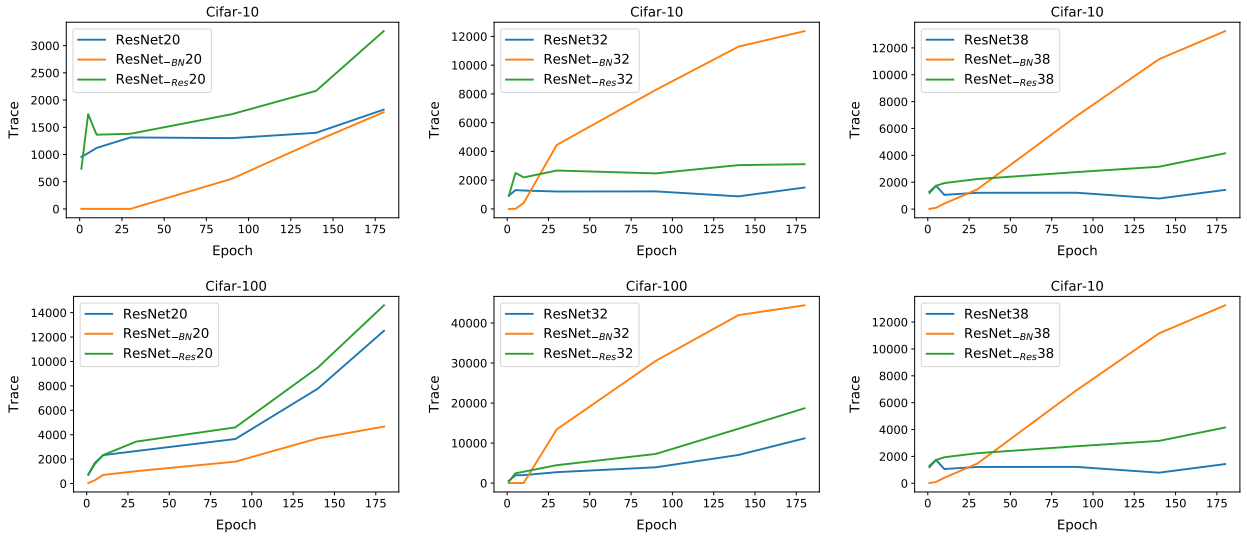

Fig. 2: *The Hessian trace of the entire network for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20/32/38 on Cifar-10 (first row) and Cifar-100 (second row). It can be clearly seen that removing BN from the architecture (shown in orange) generally results in a rapid increase of the Hessian trace. This increase is more pronounced for deeper networks such as ResNet32 (middle) and ResNet38 (right). Importantly, the Hessian trace of ResNet20 without BN is lower than the original model (blue). This is in contrast to the claim of [24]. Also, we generally observe that residual connections smooth the Hessian trace for both shallow and deep networks (compare blue and green lines).*

To get around this, we relax this problem further [17, 25]. Define $\phi_\sigma^v(t) = v^T f(H;t,\sigma)v$, in which case we have

$$\begin{aligned}\phi_\sigma^v(t) &= v^T f(H;t)v = v^T Q f(\Lambda;t) Q^T v \\ &= \sum_{i=1}^{m} \mu_i^2 f(\lambda_i;t),\end{aligned} \tag{10}$$

where μ_i is the magnitude (or dot product) of v along the i-th eigenvector of H. Now let us define a probability distribution w.r.t. α with the cumulative distribution function, $\pi(\alpha)$, as the following piece-wise function:

$$\pi(\alpha) = \begin{cases} 0 & \alpha \leq \lambda_m, \\ \sum_{i=1}^{j} \mu_i^2 & \lambda_j \leq \alpha \leq \lambda_{j-1}, \\ \sum_{i=1}^{m} \mu_i^2 & \lambda_1 \leq \alpha. \end{cases} \tag{11}$$

Then, by the Riemann-Stieltjes integral, it follows that

$$\phi_\sigma^v(t) = \int_{\lambda_m}^{\lambda_1} f(\alpha;t) d\pi(\alpha). \tag{12}$$

This integral can be estimated by Gauss quadrature rule [11],

$$\phi_\sigma^v(t) \approx \sum_{i=1}^{q} \omega_i f(t_i;t,\sigma), \tag{13}$$

where (ω_i, t_i) is the weight-node pair to estimate the integral. The stochastic Lanczos algorithm can then be used to estimate accurately this quantity [10, 17, 25]. Specifically, for the q-step Lanczos algorithm, we have q eigenpairs $(\tilde{\lambda}_i, \tilde{v}_i)$. Let $\tau_i = (\tilde{v}_i[1])^2$, where $\tilde{v}_i[1]$ is the first component of $\tilde{v}_i$, in which case it follows that

$$\phi_\sigma^v(t) \approx \sum_{i=1}^{q} \omega_i f(t_i;t,\sigma) \approx \sum_{i=1}^{q} \tau_i f(\tilde{\lambda}_i;t,\sigma). \tag{14}$$

Therefore, as in the Hutchinson algorithm, with multiple different runs (e.g., n_v times) of Lanczos algorithm, ϕ_σ can be approximated by

$$\phi_\sigma(t) = Tr(f(H)) \approx \frac{1}{n_v} \sum_{l=1}^{n_v} \left(\sum_{i=1}^{q} \tau_i^{(l)} f(\tilde{\lambda}_i^{(l)};t,\sigma) \right). \tag{15}$$

See Algorithm III.3 for a description of the SLQ algorithm.

IV. Results

As an example of how PyHessian can be used in the analysis of NNs, here we provide extensive empirical results to study the impact of BN and residual connection on the Hessian spectrum. We start in §IV-A by discussing experimental settings, followed by presenting the Hessian spectrum results for the entire model in §IV-B and different ResNet stages in §IV-C.

A. Experimental Setting

Using PyHessian, we measure all three Hessian spectrum metrics (i.e., top eigenvalues, trace, and full ESD) throughout the training process of SGD with momentum. We consider various ResNet [12] architectures, and in particular ResNet20/32/38 on the Cifar-10, and we analyze these models and variants with/without BN and with/without residual connections. We also experimented with the same networks tested on Cifar-100 dataset, and all of the observations were consistent.

For clarity, we refer to the networks without BN as ResNet$_{-BN}$, and the networks without residual connection as ResNet$_{-Res}$. We train each model (ResNet, ResNet$_{-BN}$, and ResNet$_{-Res}$) for 180 epochs, with five different initial learning rates (0.1, 0.05, 0.01, 0.005, 0.001) on Cifar-10, and ten different initial learning rates (0.1, 0.05, 0.01, 0.005, 0.001, 0.0005,0.0004, 0.0003, 0.0002, 0.00001) on Cifar-100. The

optimizer is SGD with momentum (0.9). The learning rate decays by a factor of 10 at epoch 80 and 120. We pick the best performing result for analysis. We analyze the spectrum throughout training at all checkpoints. The accuracy of each model is reported in Table I.

Table I: *Accuracy of ResNet, ResNet$_{-BN}$, and ResNet$_{-Res}$, with different depths, on Cifar-10/100. The accuracy drops if the BN layer is removed (denoted by ResNet$_{-BN}$), and this degradation is more pronounced for deeper models. Removing the residual connection (denoted as ResNet$_{-Res}$) also results in slight performance degradation.*

	Cifar-10			Cifar-100		
Model\Depth	20	32	38	20	32	38
ResNet	92.01	92.05	92.37	66.47	68.26	69.06
ResNet$_{-BN}$	87.27	66.57	53.65	62.82	25.89	11.25
ResNet$_{-Res}$	90.66	89.80	88.92	64.59	62.08	62.75

B. Full Network Hessian Analysis

We start with the original ResNet model with BN and residual connections. Hereafter, we refer to this as ResNet. The behaviour of the Hessian trace throughout training is shown in Figure 2. Furthermore, we show the evolution of the Hessian ESD throughout training in Figure 3, 4, and Figure 5.

a) ***Batch Normalization****:* A BN layer is crucial for training NN models, and removing this component can adversely affect the generalization performance, as shown in Table I. The drop in performance is very significant for deeper models.

The first interesting observation is that removing the BN layer (denoted by ResNet$_{-BN}$) exhibits different behaviour for shallow versus deep models. For example, for ResNet20 we see that removing BN results in *smaller* trace and Hessian ESD values, as compared to baseline, as shown in Figure 2 (orange curve versus blue curve), and 3 (second versus first column).

In more detail, from the evolution plot of Figure 3 on Cifar-10 throughout training, it can be seen that the ESD of ResNet$_{-BN}$ 20 initially reduces significantly and centers around zero. That is, that the model gets attracted to areas with a significantly large number of small/degenerate Hessian directions. This continues until epoch 30 on, at which point the training gets attracted to regions of the loss landscape with several non-degenerate Hessian directions.

This clearly shows that training without BN makes training harder, but it does not necessarily mean that the Hessian spectrum is going to be larger than the baseline model, despite the claim made by [24]. In fact, we only observe the smoothing behaviour proposed by [24] for deeper NN models. For example, observe the Hessian trace plot of ResNet32/38, shown in Figure 2 (middle and right plot). Here, the Hessian trace of ResNet$_{-BN}$ 32 increases to 10000 from zero, as compared to 2000 for ResNet. The Hessian ESD also exhibits the same behaviour, as shown in Figure 4 and 5. We can clearly see that the range of eigenvalues of ResNet$_{-BN}$ is significantly larger, as compared to ResNet.

The Hessian ESD of ResNet32 and ResNet38 *throughout the training process* is shown in Figure 4, 5. Again, we see the interesting behaviour that without the BN layer, the spectrum initially converges to degenerate Hessian directions, before finding non-degenerate directions in later epochs of training. The Hessian trace and the range of the Hessian ESD significantly increases as the model gets deeper.

These plots show the numerical values of the Hessian spectrum. However, the results could perhaps be more intuitively presented via parametric plots of the loss landscape. We plot the parametric 3D loss landscapes of ResNet20/32/38 on Cifar-10 with/without BN in Figure 1 (compare left and middle columns). These plots are computed by perturbing the model parameters across the first and second eigenvectors of the Hessian. For ResNet20, it can be clearly seen that removing the BN layer (middle plot) results in convergence to a flatter local minimum, as compared to ResNet20 with BN. This observation is the opposite of the common belief that adding the BN layer makes the loss landscape smoother [24]. However, for ResNet38, we can also see that removing the BN layer results in convergence to a point with a higher value of the loss. The visualizations corroborate our finding that initially, ResNet$_{-BN}$ finds points with degenerate Hessian directions, before converging to a point with non-degenerate directions.

In summary, our empirical results highlight two points. First, our findings—easily-enabled by PyHessian—show several fine-scale behaviours when the BN layer is removed. Importantly, we find that the observation made in [24] only holds for deeper models, and are not necessarily true for shallow networks. Second, using the scalable Hessian-based techniques implemented in PyHessian, one can easily ask such questions, i.e., one can test hypotheses that these or other claims hold more generally. We observed exactly similar behaviour for Cifar-100 dataset as shown in Figure 2 for the trace. Due to space constraints, the ESD plots for Cifar-100 are not included; see the technical report version [26] for these and other results.

b) ***Residual Connection****:* We next study the impact of residual connections on the smoothness of the loss landscape. Removing residual connections leads to slightly poorer generalization, as shown in Table I, although the degradation is much smaller than seen when removing the BN layer. We report the behaviour of the Hessian trace for ResNet$_{-Res}$ in Figure 2 for ResNet20/32/38 on Cifar-10/100. It can clearly be seen that the trace of ResNet$_{-Res}$ is consistently higher than that of ResNet, for both shallow and deep models on different datasets. In addition, from the Hessian ESD in Figure 3, 4, 5, we can see that the top eigenvalues as well as the support range of the ESD of ResNet$_{-Res}$ increase for deeper models. These results are in line with the findings of [16]. We also visualize the loss landscape of these models in Figure 1, It can clearly be seen that the converging point for ResNet$_{-Res}$ becomes sharper, as compared with ResNet, as the depth grows.

Again, our empirical results highlight two points. First, we make observations—easily-enabled by PyHessian—that provide a finer-scale understanding of seemingly-contradictory claims in the previous literature. Second, using the scalable Hessian-based techniques that are implemented in PyHessian, one can easily formulate and test hypotheses that these or other

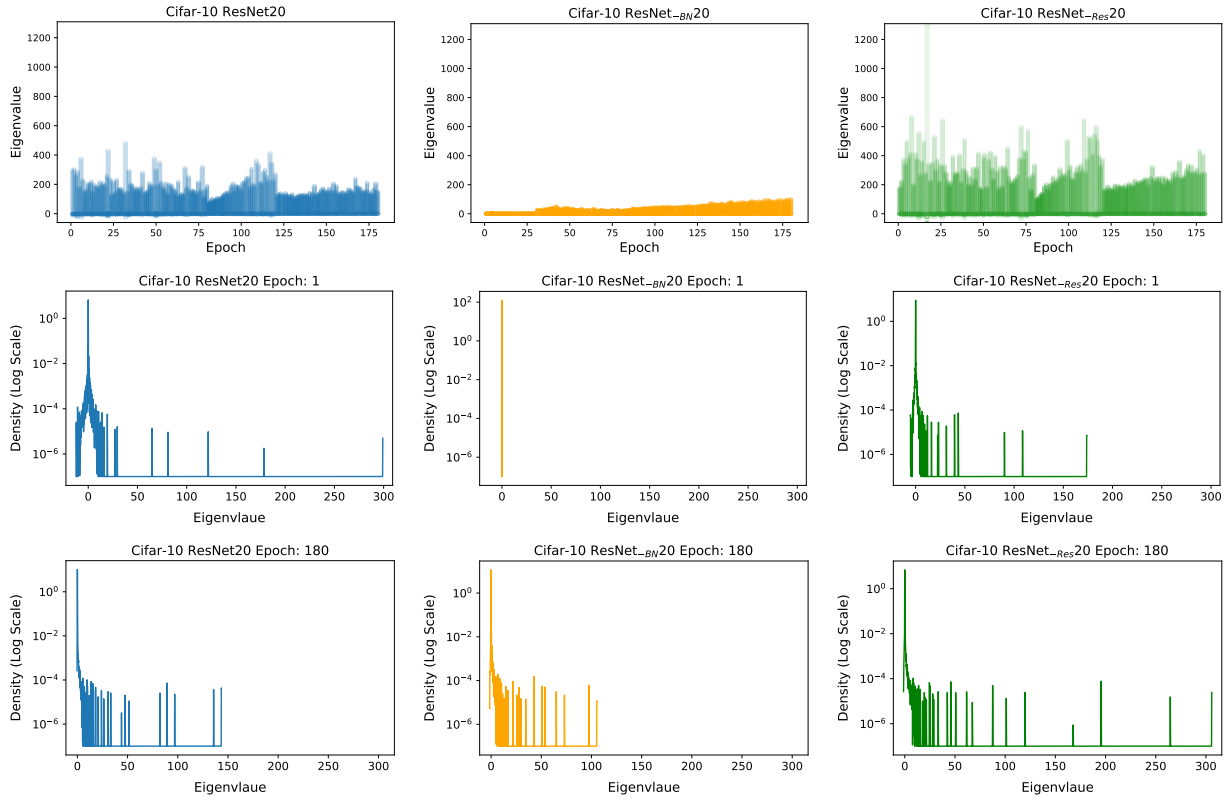

Fig. 3: *(first row) The Hessian ESD throughout training for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 20 on Cifar-10 shown in different columns. For a fixed epoch, every point corresponds to a Hessian eigenvalue. These plots show several important phenomena. First note that removing batch normalization (middle column) does not lead to a non-smooth loss landscape as was claimed by [24]. We can clearly see that this is true throughout training. However, removing the residual connection makes the loss landscape non-smooth throughout training. (middle/last row) The Hessian ESD at epoch 1 and epoch 180. This clearly shows that removing BN leads to a maximum eigenvalue of 100, whereas the baseline has a maximum Hessian eigenvalue of 150.*

claims hold more generally. Similar to the previous section, we saw exactly similar behaviour for Cifar-100; see the technical report version [26] for the results.

C. Stage-wise Hessian Analysis

We also analyzed the impact of removing BN and residual connection from different stages of the model. We define each stage of ResNet as blocks with the same activation resolution.

We plot the Hessian trace for the three stages of ResNet32 on Cifar-10 in Figure 6. We can clearly see that removing the BN from the last stage of ResNet generally results in a more rapid increase in the Hessian trace, as compared to removing BN from the first or second stage. Interestingly, this has a direct correlation with the final generalization performance reported in Table II. We can see that removing BN in the third stage results in a higher accuracy drop, as compared to removing it from other stages. We generally observe the same behaviour on Cifar-100 as reported in Table II.

As for the residual connections, we can see that removing them results in a relatively smaller increase in the Hessian trace, and correspondingly the impact of removing the residual connections on accuracy is smaller, as compared to removing BN; see Table III for Cifar-10/100.

Table II: *Accuracy of ResNet models on Cifar-10/100 with different depths is shown in the first row. Accuracy of the corresponding architectures, but with BN removed from one of the stages, is shown in the next three rows, respectively. For instance, the last row is a ResNet model with no BN layer in the third stage. We observe a general correlation between the accuracy drop and stage based Hessian analysis shown Figure 6. In particular, we see that stages which significantly affect accuracy also exhibit a significant increase in the Hessian trace.*

	Cifar-10			Cifar-100		
Model\Depth	20	32	38	20	32	38
ResNet	92.01	92.05	92.37	66.47%	68.26%	69.06%
RM BN stage 1	91.28	91.98	92.20	65.69%	65.74%	67.31%
RM BN stage 2	91.49	91.94	91.70	65.62%	64.68%	66.46%
RM BN stage 3	90.59	88.57	86.96	65.63%	64.57%	61.04%

V. Conclusions

We have developed PyHessian, an open-source framework for analyzing NN behaviour through the lens of the Hessian [1]. PyHessian enables direct and efficient computation of Hessian-based statistics, including the top eigenvalues, the trace, and the full ESD, with support for distributed-memory execution on cloud/supercomputer systems. Importantly, since it uses matrix-free techniques, PyHessian accomplishes this

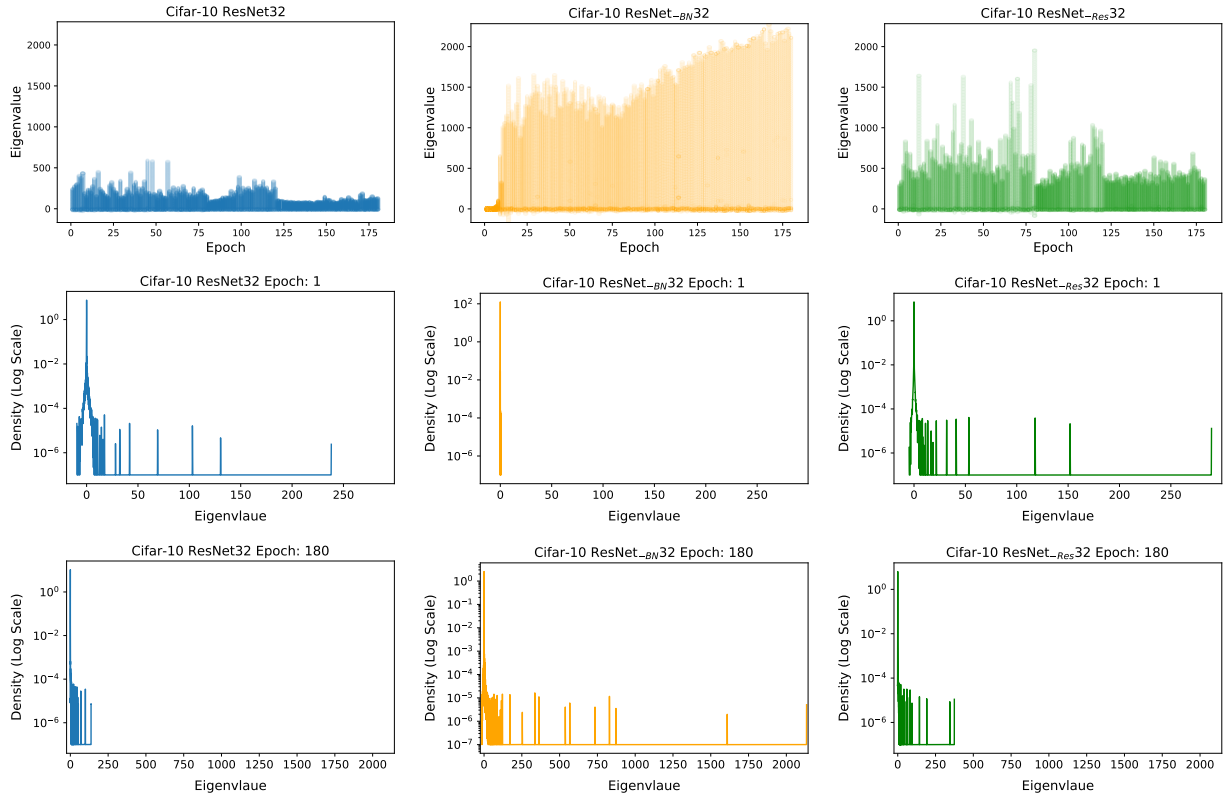

Fig. 4: *(first row) The Hessian ESD throughout training for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 32 on Cifar-10 shown in different columns. For a fixed epoch, every point corresponds to a Hessian eigenvalue. These plots show several important phenomena. Removing the residual connection or batch normalization makes the loss landscape non-smooth throughout training. (middle/last row) The Hessian ESD at epoch 1 and epoch 180.*

Table III: *Accuracy of ResNet on Cifar-10/100 is reported for baseline (first row), along with architectures where the residual connection is removed at different stages.*

	Cifar-10			Cifar-100		
Model\Depth	20	32	38	20	32	38
ResNet	92.01%	92.05%	92.37%	66.47%	68.26%	69.06%
RM Res stage 1	91.52%	92.27%	91.74%	66.46%	66.94%	67.61%
RM Res stage 2	91.06%	91.07%	91.08%	65.70%	66.05%	66.70%
RM Res stage 3	91.54%	92.09%	92.14%	66.21%	66.38%	66.03%

without the need to form the full Hessian. This means that we can compute second-order statistics for state-of-the-art NNs in times that are only marginally longer than the time used by popular gradient-based techniques.

As an application, we have shown how PyHessian can be used to study in detail the impact of popular NN architectural changes (such as adding/modifying BN and residual connections) on the NN loss landscape. We found that adding BN layers does not necessarily result in a smoother loss landscape, as claimed by [24]. We have observed this phenomenon only for deeper models, where removing the BN layer results in convergence to "sharp" local minima that have high training loss and poor generalization, but it does not seem to hold for shallower models. We also showed that removing residual connections resulted in a slightly coarser loss landscape, a finding illustrated with parametric 3D visualizations, and which all three Hessian spectrum metrics confirmed.

Acknowledgments

This work was supported by a gracious fund from Amazon Machine Learning Research Award (MLRA). We acknowledge gracious support from Intel corporation, Intel VLAB team, Google Cloud, Google TFTC team, and Nvidia, as well as valuable feedback from Prof. Dave Patterson, Prof. Joseph Gonzalez, and Lianmin Zheng. Amir Gholami was supported through from Samsung SAIT and NSF. Michael Mahoney would like to acknowledge the UC Berkeley CLTC, ARO, IARPA, NSF, and ONR for providing partial support of this work. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

References

[1] https://github.com/amirgholami/pyhessian, 2020.

[2] Rohan Anil, Vineet Gupta, Tomer Koren, Kevin Regan, and Yoram Singer. Second order optimization made practical. *arXiv preprint arXiv:2002.09018*, 2020.

[3] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive

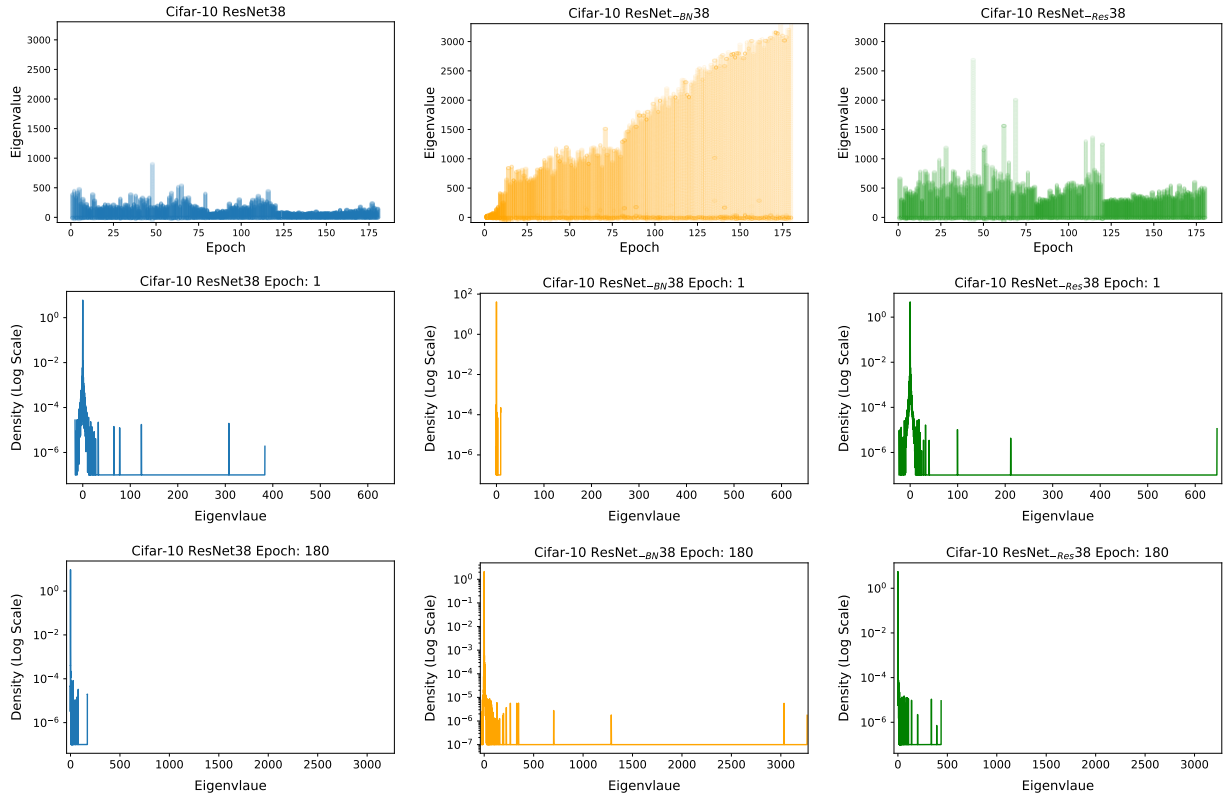

Fig. 5: *(first row) The Hessian ESD throughout training for ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ with depth 38 on Cifar-10 shown in different columns. For a fixed epoch, every point corresponds to a Hessian eigenvalue. These plots show several important phenomena. Removing the residual connection or batch normalization makes the loss landscape non-smooth throughout training. (middle/last row) The Hessian ESD at epoch 1 and epoch 180.*

semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):8, 2011.

[4] Zhaojun Bai, Gark Fahey, and Gene Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(1-2):71–89, 1996.

[5] Sue Becker and Yann Le Cun. Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37, 1988.

[6] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018.

[7] Petros Drineas, Ravi Kannan, and Michael W Mahoney. Fast Monte Carlo algorithms for matrices II: Computing a low-rank approximation to a matrix. *SIAM Journal on computing*, 36(1):158–183, 2006.

[8] Petros Drineas and Michael W Mahoney. Lectures on randomized numerical linear algebra. In *The Mathematics of Data*, IAS/Park City Mathematics Series, pages 1–48. AMS/IAS/SIAM, 2018.

[9] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via Hessian eigenvalue density. *arXiv preprint arXiv:1901.10159*, 2019.

[10] Gene H Golub and Gérard Meurant. *Matrices, moments and quadrature with applications*. Princeton University Press, 2009.

[11] Gene H Golub and John H Welsch. Calculation of Gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[14] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[15] Yann LeCun, Ido Kanter, and Sara A Solla. Second order properties of error surfaces: Learning time and generalization. In *Advances in neural information processing systems*, pages 918–924, 1991.

[16] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing*

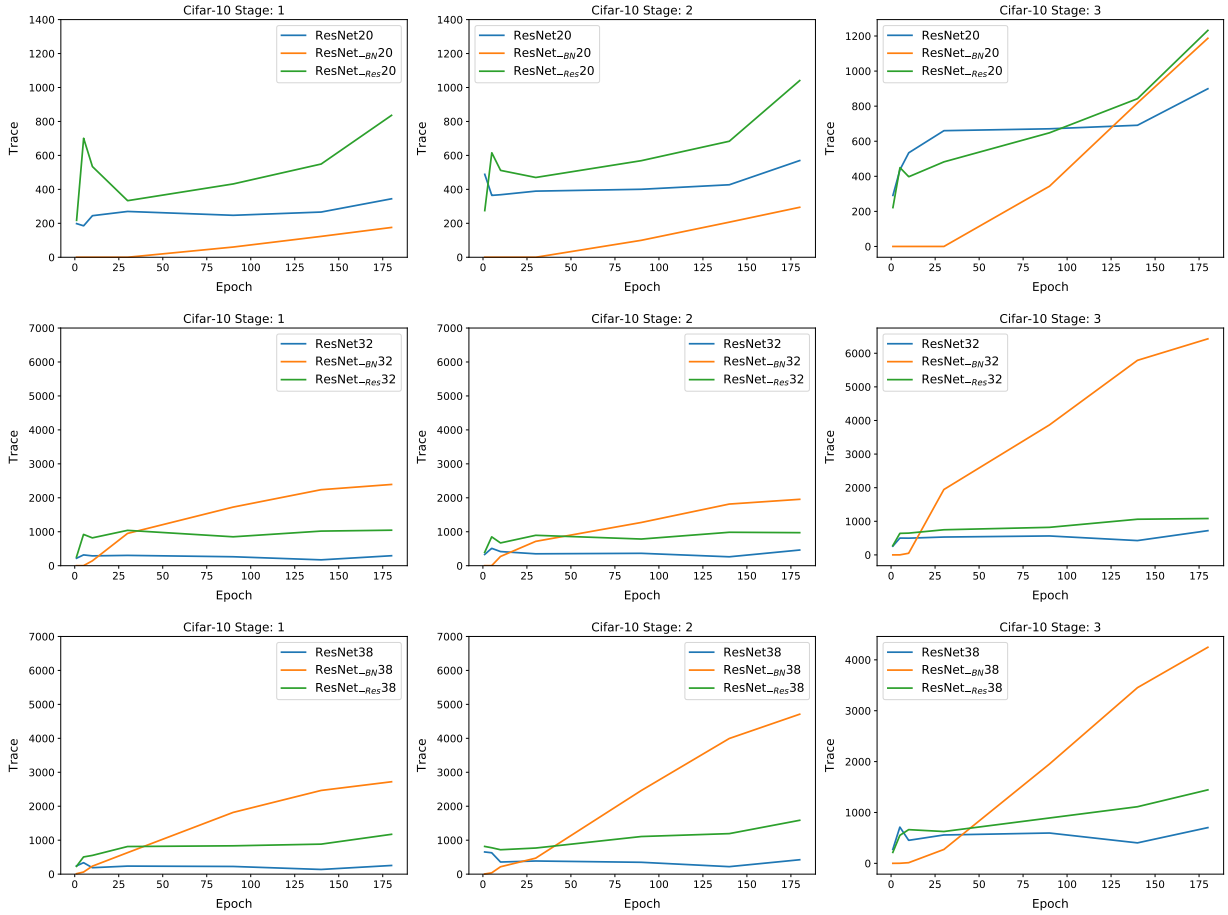

Fig. 6: *Stage-wise Hessian trace of ResNet/ResNet$_{-BN}$/ResNet$_{-Res}$ 20/32/38 on Cifar-10. Removing BN layer from the third stage significantly increases the trace, compared to removing BN layer from the first/second stage. This has a direct correlation with the final generalization performance, as shown in Table II.*

Systems, pages 6389–6399, 2018.

[17] Lin Lin, Yousef Saad, and Chao Yang. Approximating spectral densities of large matrices. *SIAM review*, 58(1):34–65, 2016.

[18] Michael W Mahoney. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011.

[19] James Martens. Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[20] Barak A Pearlmutter. Fast exact multiplication by the Hessian. *Neural computation*, 6(1):147–160, 1994.

[21] Jeffrey Pennington and Yasaman Bahri. Geometry of neural network loss surfaces via random matrix theory. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2798–2806. JMLR. org, 2017.

[22] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the Hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[23] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the Hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[24] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018.

[25] Shashanka Ubaru, Jie Chen, and Yousef Saad. Fast estimation of tr(f(a)) via stochastic Lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

[26] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. PyHessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

[27] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. *Advances in Neural Information Processing Systems*, 2018.

[28] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. ADAHESSIAN: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.