An Investigation into Neural Net Optimization via Hessian Eigenvalue Density

Behrooz Ghorbani [1 2] **Shankar Krishnan** [2] **Ying Xiao** [2]

Abstract

To understand the dynamics of optimization in deep neural networks, we develop a tool to study the evolution of the entire Hessian spectrum throughout the optimization process. Using this, we study a number of hypotheses concerning smoothness, curvature, and sharpness in the deep learning literature. We then thoroughly analyze a crucial structural feature of the spectra: in non-batch normalized networks, we observe the rapid appearance of large isolated eigenvalues in the spectrum, along with a surprising concentration of the gradient in the corresponding eigenspaces. In batch normalized networks, these two effects are almost absent. We characterize these effects, and explain how they affect optimization speed through both theory and experiments. As part of this work, we adapt advanced tools from numerical linear algebra that allow scalable and accurate estimation of the entire Hessian spectrum of ImageNet-scale neural networks; this technique may be of independent interest in other applications.

1. Introduction

The Hessian of the training loss (with respect to the parameters) is crucial in determining many behaviors of neural networks. The eigenvalues of the Hessian characterize the local curvature of the loss which, for example, determine how fast models can be optimized via first-order methods (at least for convex problems), and is also conjectured to influence the generalization properties. Unfortunately, even for moderate sized models, exact computation of the Hessian eigenvalues is computationally impossible. Previous studies on the Hessian have focused on small models, or are limited to computing only a few eigenvalues (Sagun et al., 2016; 2017; Yao et al., 2018). In the absence of such concrete information about the eigenvalue spectrum, many researchers have developed clever *ad hoc* methods to understand notions of smoothness, curvature, sharpness, and poor conditioning in the landscape of the loss surface. Examples of such work, where some surrogate is defined for the curvature, include the debate on flat vs sharp minima (Keskar et al., 2016; Dinh et al., 2017; Wu et al., 2017; Jastrzębski et al., 2017), explanations of the efficacy of residual connections (Li et al., 2018b; Orhan & Pitkow, 2017) and batch normalization (Santurkar et al., 2018), the construction of low-energy paths between different local minima (Draxler et al., 2018), qualitative studies and visualizations of the loss surface (Goodfellow et al., 2014), and characterization of the intrinsic dimensionality of the loss (Li et al., 2018a; Fort & Scherlis, 2018). In each of these cases, detailed knowledge of the entire Hessian spectrum would surely be informative, if not decisive, in explaining the phenomena at hand.

In this paper, we develop a tool that allows us access to the entire spectrum of a deep neural network. The tool is both highly accurate (we validate it to a double-precision accuracy of 10^{-14} for a 15000 parameter model), and highly scalable (we are able to generate the spectra of Resnets (He et al., 2016) and Inception V3 (Szegedy et al., 2016) on ImageNet in a small multiple of the time it takes to train the model). The underlying algorithm is extremely elegant, and has been known in the numerical analysis literature for decades (Bai et al., 1996) (based on foundational work by Golub and Welsch (1969)); here we introduce it to the machine learning community, and build (and release) a system to run it at modern deep learning scale.

This algorithm allows us to peer into the optimization process with unprecedented clarity. By generating Hessian spectra with fine time resolution, we are able to study all phases of training, and are able to comment fruitfully on a number of hypotheses in the literature about the geometry of the loss surface. Our main experimental result focuses on the role of outlier eigenvalues, we analyze how the outlier eigenvalues affect the speed of optimization; this in turn provides significant insight into how batch normalization (Ioffe & Szegedy, 2015), one of the most popular innovations in training deep neural nets, speeds up optimization.

[1]Department of Electrical Engineering, Stanford University. Work was done while author was an intern at Google. [2]Machine Perception, Google Inc.. Correspondence to: Behrooz Ghorbani <ghorbani@stanford.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

We believe our tool and style of analysis will open up new avenues of research in optimization, generalization, architecture design etc. So we release our code to the community to accelerate a Hessian based analysis of deep learning.

1.1. Contributions

In this paper, we empirically study the full Hessian spectrum of the loss function of deep neural networks. Our contributions are as follows:

In Section 2, we introduce a tool and a system, for estimating the full Hessian spectrum, capable of tackling models with tens of millions of parameters, and millions of data points. We both theoretically prove convergence properties of the underlying algorithm, and validate the system to double precision accuracy 10^{-14} on a toy model.

In Section 3, we use our tool to generate Hessian spectra along the optimization trajectory of a variety of deep learning models. In doing so, we revisit a number of hypotheses in the machine learning literature surrounding curvature and optimization. With access to the entire Hessian spectrum, we are able to provide new perspectives on a variety of interesting problems: we concur with many of the coarse descriptions of the loss surface, but disagree with a number of hypotheses about how learning rate and residual connections interact with the loss surface. Our goal is not necessarily to provide proofs or refutation – at the very least, that would require the study of a more diverse set of models – but to provide strong evidence for/against certain interesting ideas, and simultaneously to highlight some applications of our tool.

In Section 4, we observe that models with significant outlier Hessian eigenvalues exhibit slow training behavior. We provide a theoretical justification for this in Section 4.1 – we argue that a non-trivial fraction of energy of the Hessian is distributed across the bulk in tiny eigenvalues, and that a coupling between the stochastic gradients and the outlier eigenvalues prevents progress in those directions. We then show that batch normalization pushes these outliers back into the bulk, and are able to isolate this effect by ablating the batch normalization operation. In Section 4.2, we confirm the predictions of our hypothesis by studying a careful intervention to batch normalization that causes the resurgence of outlier eigenvalues, and dramatic slowdowns in optimization.

1.2. Related Work

Empirical analysis of the Hessian has been of significance interest in the deep learning community. Due to computational costs of computing the exact eigenvalues ($O(n^3)$ for an explicit $n \times n$ matrix), most of the papers in this line of research either focus on smaller models or on low-dimensional projections of the loss surface. Sagun et al. (2016; 2017) study the spectrum of the Hessian for small two-layer feed-forward networks. They show that the spectrum is divided into two parts: (1) a bulk concentrated near zero which includes almost all of the eigenvalues and (2) roughly "number of classes - 1" outlier eigenvalues emerging from the bulk. We extend this analysis in two ways. First, we calculate the Hessian for models with $> 10^7$ parameters on datasets with $> 10^6$ examples – we find that many, but not all of the above observations hold at this scale, and refine some of their observations. Secondly, we leverage the scalability of our algorithm to compute and track the Hessian spectrum throughout the optimization (as opposed to only at the end). Observing this evolution allows us to study how individual architecture choices affect optimization. There is an extensive literature regarding estimating the eigenvalues distribution of large matrices (for a small survey, see Lin et al. (2016)). The algorithm we use is due to Golub and Welsch (1969); the application of this to trace estimators is due to Bai et al. (1996). While many of these algorithms have theoretical guarantees, their empirical success is highly dependent on the problem structure. We perform a thorough comparison of our work to the recent proposal of Adams et al. (2018) in Appendix D.

Batch Normalization (BN) (Ioffe & Szegedy, 2015) is one of the most influential innovations in optimizing deep neural networks as it substantially reduces the training time and the dependence of the training on initialization. There has been much interest in determining the underlying reasons for this effect. The original BN paper suggests that as the model trains, the distribution of inputs to each layer changes drastically, a phenomenon called internal covariance shift (ICS). They suggest that BN improves training by reducing ICS. There has been a series of exciting new works exploring the effects of BN on the loss surface. Santurkar et al. (2018) empirically show that ICS is not necessarily related to the success of the optimization. They instead prove that under certain conditions, the Lipschitz constant of the loss and β-smoothness of the loss with respect to the activations and weights of a linear layer are improved when BN is present. Unfortunately, these bounds are on a per-layer basis; this yields bounds on the diagonal blocks of the overall Hessian, but does not directly imply anything about the overall β-smoothness of the entire Hessian. In fact even exact knowledge of β for the entire Hessian and parameter norms (to control the distance from the optimum) is insufficient to determine the speed of optimization: in Section 4.2, we exhibit two almost identical networks that differ only in the way batch norm statistics are calculated; they have almost exactly the same largest eigenvalue and the parameters have the same scale, yet the optimization speeds are vastly different.

During the preparation of this paper, (Papyan, 2018) ap-

peared on Arxiv which briefly introduces the same spectrum estimation methodology and studies the Hessian on small subsamples of MNIST and CIFAR-10 at the end of the training. In comparison, we provide a detailed exposition, error analysis and validation of the estimator in Section 2, and present optimization results on full datasets, up to and including ImageNet.

1.3. Notation

Neural networks are trained iteratively. We call the estimated weights at optimization iteration t, $\hat{\theta}_t$, $0 \leq t \leq T$. We define the loss associated with batch i be $\mathcal{L}_i(\theta)$. The full-batch loss is defined as $\mathcal{L}(\theta) \equiv \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_i(\theta)$ where N is the number of batches.[1] The Hessian, $\nabla^2\mathcal{L}(\theta) \in \mathbb{R}^{n\times n}$ is a symmetric matrix such that $\nabla^2\mathcal{L}(\theta)_{i,j} = \frac{\partial^2}{\partial_{\theta_i}\partial_{\theta_j}}\mathcal{L}(\theta)$. Note that our Hessians are all "full-batch" Hessians (i.e., they are computed using the entire dataset). When there is no confusion, we represent $\nabla^2\mathcal{L}(\hat{\theta}_t)$ with $H \in \mathbb{R}^{n\times n}$. Throughout the paper, H has the spectral decomposition $Q\Lambda Q^T$ where $\Lambda = diag(\lambda_1, \ldots, \lambda_n)$, $Q = [q_1, \ldots, q_n]$ and $\lambda_1 \geq \lambda_2 \cdots \geq \lambda_n$.

2. Accurate and Scalable Estimation of Hessian Eigenvalue Densities for $n > 10^7$

To understand the Hessian, we would like to compute the eigenvalue (or spectral) density, defined as $\phi(t) = \frac{1}{n}\sum_{i=1}^{n}\delta(t-\lambda_i)$ where δ is the Dirac delta operator. The naive approach requires calculating λ_i; however, when the number of parameters, n, is large this is not tractable. We relax the problem by convolving with a Gaussian density of variance σ^2 to obtain:

$$\phi_\sigma(t) = \frac{1}{n}\sum_{i=1}^{n} f(\lambda_i; t, \sigma^2) \tag{1}$$

where $f(\lambda; t, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{(t-\lambda)^2}{2\sigma^2}\right)$. For small enough σ^2, $\phi_\sigma(t)$ provides all practically relevant information regarding the eigenvalues of H. Explicit representation of the Hessian matrix is infeasible when n is large, but using Pearlmutter's trick (Pearlmutter, 1994) we are able to compute Hessian-vector products for any chosen vector.

2.1. Stochastic Lanczos Quadrature

It has long been known in the numerical analysis literature that accurate stochastic approximations to the eigenvalue density can be achieved with much less computation than a full eigenvalue decomposition. In this section, we describe the *stochastic Lanczos quadrature* algorithm (Golub & Welsch, 1969; Lin et al., 2016). Although the algorithm is already known, its mathematical complexity and potential as a research tool warrant a clear exposition for a machine learning audience. We give the pseudo-code in Algorithm 1, and describe the individual steps below, deferring a discussion of the various approximations to Section 2.2.

Since H is diagonalizable and f is analytic, we can define $f(H) = Qf(\Lambda)Q^T$ where $f(\cdot)$ acts point-wise on the diagonal of Λ. Now observe that if $v \sim N(0, \frac{1}{n}I_{n\times n})$, we have

$$\phi_\sigma(t) = \frac{1}{n}\mathrm{tr}\left(f(H, t, \sigma^2)\right) = \mathbb{E}\left[v^T f(H, t, \sigma^2)v\right] \tag{2}$$

Thus, as long as $\phi_\sigma^{(v)}(t) \equiv v^T f(H, t, \sigma^2)v$ concentrates fast enough, to estimate $\phi_\sigma(t)$, it suffices to sample a small number of random v's and average $\phi_\sigma^{(v)}(t)$.

Algorithm 1 Two Stage Estimation of $\phi_\sigma(t)$

Draw k i.i.d realizations of v, $\{v_1, \ldots, v_k\}$.

I. Estimate $\phi_\sigma^{(v_i)}(t)$ by a quantity $\widehat{\phi}^{(v_i)}(t)$:
- Run the Lanczos algorithm for m steps on matrix H starting from v_i to obtain tridiagonal matrix T.
- Compute eigenvalue decomposition $T = ULU^T$.
- Set the nodes $\ell_i = L_{ii}$ and weights $\omega_i = U_{1,i}^2$.
- Output $\widehat{\phi}^{(v_i)}(t) = \sum_{i=1}^{m}\omega_i f(\ell_i; t, \sigma^2)$.

II. Set $\widehat{\phi}_\sigma(t) = \frac{1}{k}\sum_{i=1}^{k}\widehat{\phi}^{(v_i)}(t)$.

By definition, we can write

$$\begin{aligned}\phi_\sigma^{(v)}(t) = v^T Qf(\Lambda; t, \sigma^2)Q^T v &= \sum_{i=1}^{n}(v^T q_i)^2 f(\lambda_i; t, \sigma^2) \\ &= \sum_{i=1}^{n}\beta_i^2 f(\lambda_i; t, \sigma^2)\end{aligned} \tag{3}$$

where $\beta_i \equiv (v^T q_i)$. Instead of summing over the discrete index variable i, we can rewrite this as a Riemann-Stieltjes integral over a continuous variable λ weighted by μ:

$$\phi_\sigma^{(v)}(t) = \int_{\lambda_n}^{\lambda_1} f(\lambda; t, \sigma^2)d\mu(\lambda) \tag{4}$$

where μ is a CDF (note that the probability density $d\mu$ is a sum of delta functions that directly recovers Equation 3)[2].

$$\mu(\lambda) = \begin{cases} 0 & \lambda < \lambda_n \\ \sum_{i=1}^{k}\beta_i^2 & \lambda_k \leq \lambda < \lambda_{k+1} \\ \sum_{i=1}^{n}\beta_i^2 & \lambda \geq \lambda_1 \end{cases}.$$

[1] We define the loss in terms of per-batch loss (as opposed to the per sample loss) in order to accommodate batch normalization.

[2] Technically μ is a positive measure, not a probability distribution, because $||v||^2$ only concentrates on 1. This wrinkle is irrelevant.

To evaluate this integral, we apply a quadrature rule (a quadrature rule approximates an integral as a weighted sum – the well-known high-school trapezoid rule is a simple example). In particular, we want to pick a set of weights ω_i and a set of nodes l_i so that

$$\phi_\sigma^{(v)}(t) \approx \sum_{i=1}^{m} \omega_i f(\ell_i; t, \sigma^2) \equiv \widehat{\phi}^{(v)}(t) \tag{5}$$

The hope is that there exists a good choice of $(\omega_i, \ell_i)_{i=1}^m$ where $m \ll n$ such that $\phi_\sigma^{(v)}(t)$ and $\widehat{\phi}^{(v)}(t)$ are close for all t, and that we can find the nodes and weights efficiently for our particular integrand f and the CDF μ. The construction of a set of suitable nodes and weights is a somewhat complicated affair. It turns out that if the integrand were a polynomial g of degree d, with d small enough compared to m, it is possible to compute the integral exactly,

$$\int g d\mu = \sum_{i=1}^{m} w_i g(l_i). \tag{6}$$

Theorem 2.1 ((Golub & Meurant, 2009) Chapter 6)**.** *Fix m. For all $(\beta_i, \lambda_i)_{i=1}^n$, there exists an approximation rule generating node-weight pairs $(\omega_i, \ell_i)_{i=1}^m$ such that for any polynomial, g with $deg(g) \leq 2m-1$, (6) is true. This approximation rule is called the **Gaussian quadrature**. The degree $2m-1$ achieved is maximal: for a general $(\beta_i, \lambda_i)_{i=1}^n$, no other approximation rule can guarantee exactness of Equation (6) for a higher polynomial.*

The Gaussian quadrature rule always generates non-negative weights. Therefore, as $f(\cdot; t, \sigma) \geq 0$, it is guaranteed that $\widehat{\phi} \geq 0$ which is a desirable property for a density estimate. For these reasons, despite the fact that our integrand f is not a polynomial, we use the Gaussian quadrature rule. For the construction of the Gaussian quadrature nodes and weights, we rely on a deep connection between Gaussian quadrature and Krylov subspaces via orthogonal polynomials. We refer the interested reader to the excellent (Golub & Meurant, 2009) for this connection.

Theorem 2.2 ((Golub & Welsch, 1969))**.** *Let $V = [v, Hv, \cdots, H^{m-1}v] \in \mathbb{R}^{n \times m}$ and $\tilde{V}$ be the incomplete basis resulting from applying QR factorization on V. Let $T \equiv \tilde{V}^T H \tilde{V} \in \mathbb{R}^{m \times m}$ and ULU^T be the spectral decomposition of T. Then the Gaussian quadrature nodes $\ell_i = L_{i,i}$, and the Gaussian quadrature weights $\omega_i = U_{1,i}^2$, for $i = 1, \ldots, m$.*

Theorem 2.2 presents a theoretical way to compute the Gaussian quadrature rule (i.e., apply the H matrix repeatedly and orthogonalize the resulting vectors). There are well-known algorithms that circumvent calculating the numerically unstable V, and compute T and $\tilde{V}$ directly. We use Lanczos algorithm (Lanczos, 1950) (with full re-orthogonalization) to perform this computation in a numerically stable manner.

2.2. Accuracy of Gaussian Quadrature Approximation

Intuition suggests that as long as $f(\cdot; t, \sigma^2)$ is close to some polynomial of degree at most $2m-1$, our approximation must be accurate (i.e., Theorem 2.1). Crucially, it is not necessary to know the exact approximating polynomial, its mere existence is sufficient for an accurate estimate. There exists an extensive literature on bounding this error; Ubaru et al. (2017) prove that under suitable conditions that

$$|\widehat{\phi}^{(v)}(t) - \phi_\sigma^{(v)}(t)| \leq c \frac{1}{(\rho^2 - 1)\rho^{2m}} \tag{7}$$

where $\rho > 1$. The constant ρ is closely tied to how well $f(\cdot; t, \sigma^2)$ can be approximated by Chebyshev polynomials. [3] In our setting, as σ^2 decreases, higher-order polynomials become necessary to approximate f well. Therefore, as σ^2 decreases, ρ decreases and more Lanczos iterations become necessary to approximate the integral well.

To establish a suitable value of m, we perform an empirical analysis of the error decay when H corresponds to a neural network loss Hessian. In Appendix B, we study this error on a 15910 parameter feed-forward MNIST network, where the model is small enough that we can compute $\phi_\sigma^{(v)}(t)$ exactly. For $\sigma^2 = 10^{-5}$, a quadrature approximation of order 80 achieves maximum double-precision accuracy of 10^{-14}. Following these results, we use $\sigma^2 = 10^{-5}, m = 90$ for our experiments. Equation 7 implies that the error decreases exponentially in m, and since GPUs are typically run in single precision, our m is an extremely conservative choice.

2.3. Concentration of the Quadratic Forms

Although $\phi_\sigma^{(v)}(\cdot)$ is an unbiased estimator for $\phi_\sigma(\cdot)$, we must still study its concentration towards its mean. We prove:

Claim 2.3. *Let t be a fixed evaluation point and k be the number of realizations of v in step II of Algorithm 1. Let $a = \|f(H; t, \sigma^2)\|_F$ and $b = \|f(H; t, \sigma^2)\|_2$. Then $\forall x > 0$,*

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \frac{2a}{n\sqrt{k}}\sqrt{x} + \frac{2b}{kn}x\right) \leq 2\exp(-x).$$

Alternatively, since $f(\cdot)$ is a Gaussian density, we can give norm independent bounds: $\forall x > 0$,

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \epsilon(x)\right) \leq 2\exp(-x). \tag{8}$$

where $\epsilon(x) \equiv \sqrt{\frac{2}{\pi\sigma^2}}(\sqrt{\frac{x}{nk}} + \frac{x}{nk})$.

[3] We refer the interested reader to (Ubaru et al., 2017; Demanet & Ying, 2010) for more details

Claim (2.3) shows that $\widehat{\phi}_\sigma(t)$ concentrates exponentially fast around its expectation. Note in particular the $\sqrt{n}$ and higher powers in the denominator – since the number of parameters $n > 10^6$ for cases of interest, we expect the deviations to be negligible. We plot these error bounds and prove Claim 2.3 in Appendix A.

2.4. Implementation, Validation and Runtime

We implemented a large scale version of Algorithm 1 in TensorFlow (Abadi et al., 2016); the main component is a distributed Lanczos Algorithm. We describe the implementation and performance in Appendix C. To validate our system, we computed the exact eigenvalue distribution on the 15910 parameter MNIST model. Our proposed framework achieves $L_1(\phi_\sigma, \widehat{\phi}_\sigma) \equiv \int_{-\infty}^{\infty} |\phi_\sigma(t) - \widehat{\phi}(t)| dt \approx 0.0012$ which corresponds to an extremely accurate solution. The largest model we've run our algorithm on is Inception V3 on ImageNet. The runtime is dominated by the Hessian-vector products within the Lanczos algorithm; we run mk full-batch Hessian vector products. The remaining cost of full reorthogonalization is negligible ($O(km^2n)$ floating point operations). For a Resnet-18 on ImageNet, running a single draw takes about half the time of training the model.

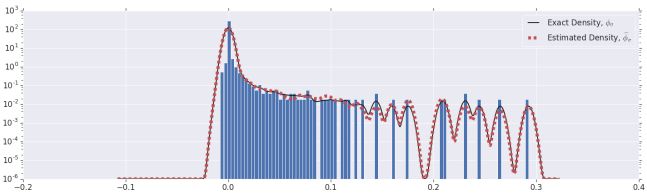

Figure 1. Comparison of the estimated smoothed density (dashed) and the exact smoothed density (solid) in the interval $[-0.2, 0.4]$. We use $\sigma^2 = 10^{-5}, k = 10$ and degree 90 quadrature. For completeness, the histogram of the exact eigenvalues is also plotted.

In Appendix D, we compare our approach to a recent proposal (Adams et al., 2018) to use Chebyshev approximation for estimating the spectral density.

3. Spectral Densities Throughout Training

The tool we developed in Section 2 gives us an unprecedented ability to examine the loss landscape of deep neural networks. In particular, we can track the spectral density throughout the entire optimization process. Our goal in this section is to provide direct curvature evidence for (and against) a number of hypotheses about the loss surface and optimization in the literature. While we cannot conclusively prove or refute any hypothesis (given the space constraints), we believe that the evidence is very strong in many of these cases.

For our analysis, we study a variety of Resnet and VGG (Simonyan & Zisserman, 2014) architectures on both CIFAR-10 and ImageNet. Details are presented in Appendix F. The Resnet-32 on CIFAR-10 has 4.6×10^5 parameters; all other models have at least 10^7. For consistency, our plots in this section are of Resnet spectral densities; we have reproduced all these results on non-residual (VGG) architectures.

At initialization, we observe that large negative eigenvalues dominate the spectrum. However, as Figure 2 shows, in only very few steps ($< 1\%$ of the total number of steps; we made no attempt to optimize this bound), these large negative eigenvalues disappear and the overall shape of the spectrum stabilizes. Sagun et al. (2016) had observed a similar disappearance of negative eigenvalues for toy feed-forward models after the training, but we are able to pinpoint this phase to the very start of optimization. This observation is readily reproducible on ImageNet.

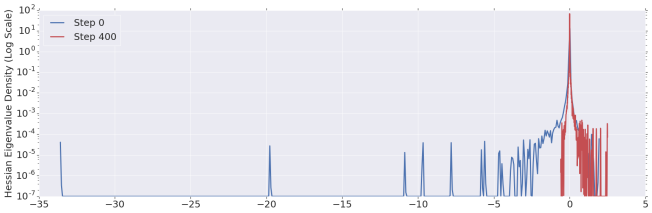

Figure 2. The evolution of the spectrum of a Resnet-32 in the beginning of training. After just 400 momentum steps, large negative eigenvalues disappear.

Throughout the rest of the optimization, the spectrum is almost entirely flat, with the vast majority ($> 99.99\%$ of eigenvalues being close to 0). This is in accordance with the ideas of Li et al. (2018a), who hypothesize that the loss surface has low intrinsic dimensionality, and also with results of Sagun et al. on toy models. In the case of K-class classification with small two-layer feed-forward networks, Sagun et al. had observed that the Hessian spectrum contains roughly K outliers which are a few orders of magnitudes larger than the rest of the eigenvalues. Contrary to this, we find that the emergence of these outliers is highly dependent on whether BN is present in the model or not. We study this behavior in depth in Section 4.

Sagun et al. also observe that the negative eigenvalues at the end of the training are orders of magnitude smaller than the positive ones. While we are able to observe this on CIFAR-10, what happens on ImageNet seems to be less clear (Figure 3). We believe that the observation of Sagun et al. may be an artifact of the size of the datasets used – on MNIST and CIFAR-10 one can easily attain zero classification loss (presumably a global minimum); on ImageNet, even a much larger model will fail to find a zero loss solution.

It is received wisdom in deep learning that low learning rates lead SGD to be attracted to sharp minima; this idea is explicit in Kleinberg et al (2018) and implicit in Jastrzęb-

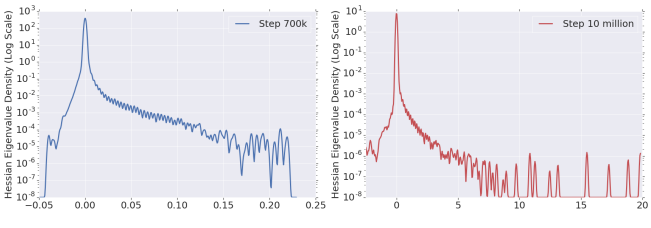

Figure 3. Spectral densities of Resnet-18 on ImageNet towards the start, and at the end of optimization. There is a notable negative density towards the end of optimization.

ski et al. (2017), where it hypothesized that that lower (constant) learning rates correspond to sharper optima. We consider this question by inspecting the spectral densities immediately preceding and following a learning rate drop. According to the hypothesis, we would then expect the spectral density to exhibit more extremal eigenvalues. In fact, we find the exact opposite to be true in Figure 4 – not only do the large eigenvalues contract substantially after the learning rate drop at 40k steps, we have a lower density at all values of λ except in a tiny ball around 0. This is an extremely surprising result, and violates the common intuition that lower learning rates allow one to slip into small, sharp crevices in the loss surface. We note that this is not a transient phenomenon – the spectrum before and afterwards are stable over time.

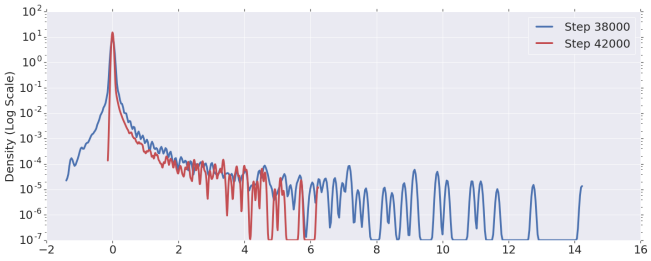

Figure 4. Spectral densities of Resnet-32 preceding and following a learning rate decrease (at step 40000). The Hessian prior to the learning rate drop appears sharper.

Finally, Li et al. (2018b) recently hypothesized that adding residual connections significantly smooths the optimization landscape, producing a series of compelling two-dimensional visualizations. We compared a Resnet-32 with and without residual connections, and we observe in Figure 5 that without residual connections all eigenvalues contract substantially towards zero. This is contrary to the visualizations of Li et al.

4. Outlier Eigenvalues Slow Optimization; Batch Norm Suppresses Outliers

In some of the spectral densities presented so far, perhaps the most salient feature is the presence of a small number

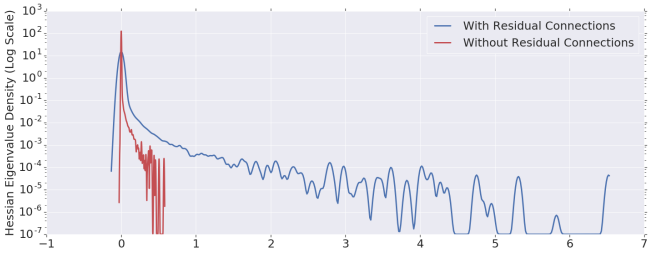

Figure 5. Spectral densities of Resnet-32 with and without residual connections (at step 40000). The Hessian without residual connections appears to be smoother.

of outlier eigenvalues that are located far from the bulk of the spectrum. We noticed that these outliers are much larger and much further from the bulk for some architectures than others. Suspecting that batch normalization was the crucial difference, we ran a series of ablation experiments contrasting the spectral density in the presence and absence of batch normalization (i.e., we added BN to models that did not already have it, and removed BN from models that already did). Figure 8 contrasts the the Hessian spectrum in the presence of BN vs the spectrum when BN is removed. The experiment yields the same results on VGG on CIFAR-10 (Figure 9), and Resnet-18 on ImageNet (Figure 7), and at various points through training.

Our experiments reveal that, in the presence of BN, the largest eigenvalue of the Hessian, $\lambda_1(H)$ tend to not to deviate as much from the bulk. In contrast, in non-BN networks, the outliers grow much larger, and further from the bulk. To probe this behavior further we formalize the notion of an outlier with a metric: $\zeta(t) := \lambda_1(\nabla^2 \mathcal{L}(\theta_t))/\lambda_K(\nabla^2 \mathcal{L}(\theta_t))$. This provides a scale-invariant measure of the presence of outliers in the spectrum. In particular, if $K-1$ (as suggested by Sagun et al.) outliers are present in the spectrum, we expect $\zeta \gg 1$. Figure 6 plots $\zeta(t)$ throughout training. It is evident that *relative* large eigenvalues appear in the spectrum. Normalization layer induces an odd dependency on parameter scale – scaling the (batch normalized) weights leads to unchanged activations, and inversely scales the gradients. Obviously, we can not conclude that the problem is much easier! Thus, for studying the optimization performance of batch normalization, we must have at least a scale-invariant quantity – which $\zeta(t)$ is. In contrast, the analysis in (Santurkar et al., 2018) varies wildly with scale[4].

Informed by the experimental results in this section, we hypothesize a mechanistic explanation for why batch normalization speeds up optimization: it does so via suppression of outlier eigenvalues which slow down optimization.

[4]We have also tried normalizing individual weights matrices and filters, but this leads to blowup in some gradient components.

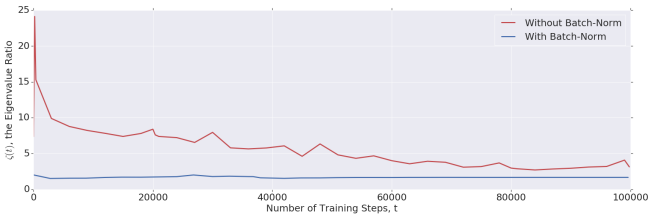

Figure 6. $\zeta(t)$ for Resnet-32 throughout training. The model without BN (red) consistently shows much higher eigenvalue fraction.

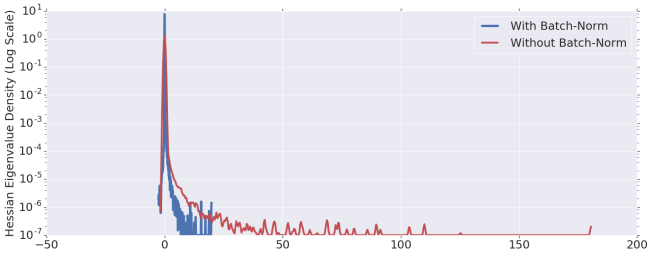

Figure 7. The eigenvalue comparison of the Hessian of Resnet-18 trained on ImageNet dataset with (blue) and without (red) BN. The Hessians are computed at the end of training.

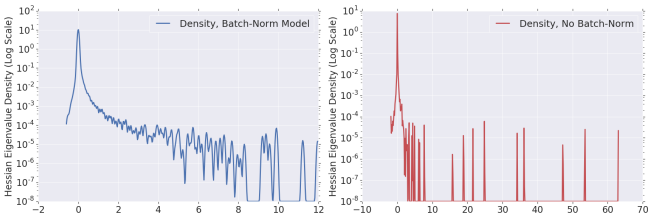

Figure 8. The eigenvalue comparison of the Hessian of the Resnet-32 model with BN (blue) and without BN (red). To allow comparison on the same plot, the densities have been normalized by their respective 10^{th} largest eigenvalue. The Hessians are computed after $48k$ steps of training.

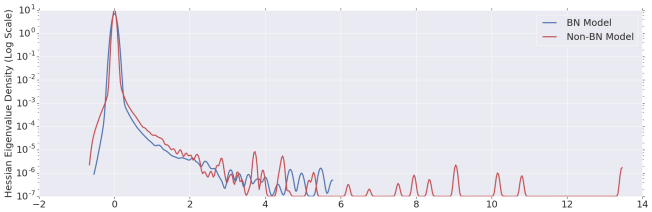

Figure 9. The eigenvalue comparison of the Hessian of the VGG network with BN (blue) and without BN (red). The Hessians are computed after 5058 steps of training.

4.1. Mechanisms by Which Outliers Slow Optimization

In this section, we seek to answer the question "*Why* do outlier eigenvalues slow optimization?" One answer to this question is obvious. Large λ_1 implies that one must use a very low learning rate; but this an incomplete explanation – λ_1 has to be large *with respect to the rest of the spectrum*. To make this explicit, consider a simple quadratic approximation to the loss around the optimum, θ^*: $\mathcal{L}(\theta) \approx \mathcal{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*)$ where without loss of generality, we assume $H = diag(\lambda_1, \cdots, \lambda_n)$ with $\lambda_i > 0$. We can easily show that when optimized with gradient descent with a learning rate $\eta < 2/\lambda_1$ sufficiently small for convergence, we have:

$$|\hat{\theta}_t - \theta^*|_i \leq \left|1 - \frac{2\lambda_i}{\lambda_1}\right|^t |\hat{\theta}_0 - \theta^*|_i \tag{9}$$

For all directions where λ_i is small with respect to λ_1, we expect convergence to be slow. One might hope that these small λ_i do not contribute significantly to the loss; unfortunately, when we measure this in a Resnet-32 with no batch normalization, a small ball around 0 accounts for almost 50% of the total L_1 energy of the Hessian eigenvalues for a converged model (the L_1 reflects the loss function $\sum_i \lambda_i(\theta - \theta^*)_i^2$). Therefore, for successful optimization, we are forced to optimize these slowly converging directions[5].

A second, more pernicious reason lies in the interaction between the large eigenvalues of the Hessian and the stochastic gradients. Define the gradient covariance at time t to be $\Sigma(t) = \frac{1}{N}\sum_{i=1}^{N} \nabla\mathcal{L}_i \nabla\mathcal{L}_i^T$. The eigenvalue density of Σ characterizes how the energy of the (mini-batch) gradients is distributed (the tools of Section 2 apply just as well here). As with the Hessian, we observe that in non-BN networks the spectrum of Σ has outliers (Figure 11). In addition, we numerically verify that the outlier subspaces of H and Σ mostly coincide: throughout the training, for a Resnet-32, 99% of the energy of the outlier Hessian eigenvectors lie in the outlier subspace of Σ. Moreover, we observe that almost all of the gradient energy is concentrated in these subspaces (Figure 10), reproducing an observation of Gur-Ari et al. (2018). We observe that when BN is introduced in the model, this concentration subsides substantially.

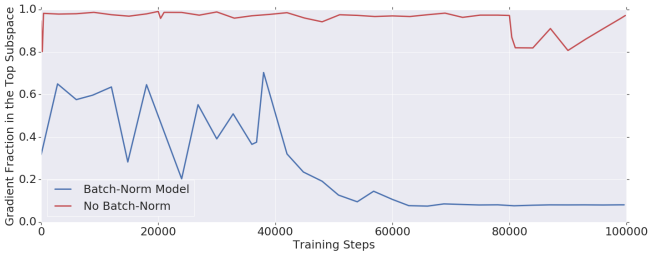

Figure 10. $\frac{\|P\nabla\mathcal{L}(\hat{\theta}_i)\|_2^2}{\|\nabla\mathcal{L}\hat{\theta}_i\|_2^2}$ for a Resnet-32. Here P is the projection operator to the subspace spanned by the 10 most dominant eigenvectors of $\nabla^2\mathcal{L}(\hat{\theta}_i)$. Almost all the variance of the gradient of the non-BN model is in this subspace.

[5]While the loss function in deep nets is not quadratic, the intuition that the result above provides is still valid in practice.

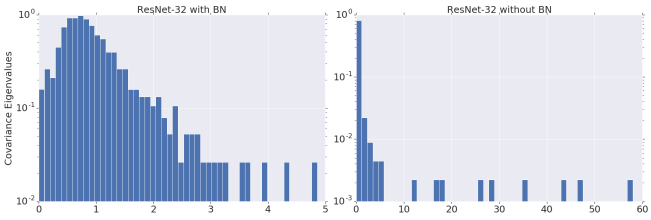

Figure 11. The histogram of the eigenvalues of Σ for a Resnet-32 with (left) and without (right) BN after $9k$ training steps. In no BN case, almost 99% of the energy is in the top few subspaces. For easier comparison, the distributions are normalized to have the same mean.

Since almost all of the gradient energy is in the very few outlier directions, the projection of the gradient in the complement of this subspace is minuscule. Thus, most gradient updates do not optimize the model in the flatter directions of the loss. As argued earlier, a significant portion of the loss comes from these flatter directions and a large fraction of the path towards the optimum lies in these subspaces. The fact that the gradient vanishes in these directions forces the training to be very slow. In Figure 12, we examines the normalized inner product between the path towards the optimum, $\theta^* - \hat{\theta}_t$, [6] and the gradients, $\nabla\mathcal{L}(\hat{\theta}_t)$, throughout the training trajectory. The figure suggests that the gradient is almost uninformative about the optimum. The situation in BN networks is significantly better as the gradient is less influenced by the high curvature directions of the loss.

We give a theoretical explanation (Theorem E.1) for why outliers in H can cause the concentration of the gradient phenomenon by studying a simple stochastic quadratic model in Appendix E.

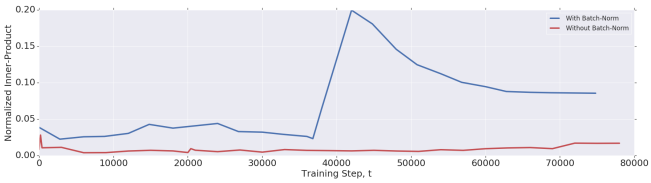

Figure 12. Normalized inner product between $\nabla\mathcal{L}(\theta_t)$ and $\theta_t - \theta^*$ throughout the optimization for a Resnet-32 model.

4.2. Testing Our Hypothesis

Our hypothesis that batch norm suppresses outliers, and hence speeds up training, is simple enough to allow us to make predictions based on it. The original batch normalization paper (Ioffe & Szegedy, 2015) observed that the normalization parameters of BN, σ_B and μ_B, have to be computed (and back-propagated through) using the mini-batch. If σ_B, μ_B are computed using the complete dataset, the training becomes slow and unstable. Therefore, we postulate that when σ_B and μ_B are calculated from the population (i.e. full-batch) statistics, the outliers persist in the spectrum.

To test our prediction, we train a Resnet-32 on Cifar-10 once using mini-batch normalization constants (denoted by mini-batch-BN network), and once using full-batch normalization constants (denoted by full-batch-BN network). The model trained with full-batch statistics trains much slower (Appendix G). Figure 13 compares the spectrum of the two networks in the early stages of the training (the behavior is the same during the rest of training). The plot suggests strong outliers are present in the spectrum with full-batch-BN. This observation supports our hypothesis. Moreover, we observe that the magnitude of the largest eigenvalue of the Hessian in between the two models is roughly the same throughout the training. Given that full-batch-BN network trains much more slowly, this observation shows that analyses based on the top eigenvalue of the Hessian do not provide the full-picture of the optimization hardness.

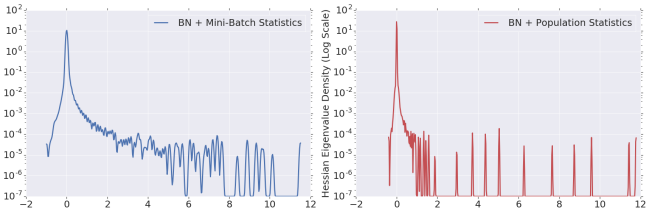

Figure 13. The Hessian spectrum for a Resnet-32 after $6k$ steps. The network on the left is trained with BN and mini-batch statistics. The network on the right is trained with population statistics.

5. Conclusion

We presented tools from advanced numerical analysis that allow for computing the spectrum of the Hessian of deep neural networks in an extremely accurate and scalable manner. We believe this tool is valuable for the research community as it gives a comprehensive view of the local geometry of the loss. This information can be used to further our understanding of neural networks.

We used this toolbox to study how the loss landscape locally evolves throughout the optimization. We uncovered surprising phenomena, some of which run contrary to the widely held beliefs in the machine learning community. In addition, we provided simple and clear answers to how batch-normalization speeds up training. We believe that BN is only one of the many architecture choices that can be studied using our framework. Studying these other architecture choices can be an interesting avenue for future research.

[6] We use the parameter at the end of the training as a surrogate for θ^*.

Acknowledgements

We would like to thank Sergey Ioffe, Rasmus Larsen, Ali Rahimi, Hossein Mobahi, and Alan Mackey for insightful discussions and suggestions.

References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

Adams, R. P., Pennington, J., Johnson, M. J., Smith, J., Ovadia, Y., Patton, B., and Saunderson, J. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

Bai, Z., Fahey, G., and Golub, G. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(1-2):71–89, 1996.

Bellec, P. Concentration of quadratic forms under a Bernstein moment assumption. Technical report, Technical report, Ecole Polytechnique, 2014.

Demanet, L. and Ying, L. On chebyshev interpolation of analytic functions. *preprint*, 2010.

Dinh, L., Pascanu, R., Bengio, S., and Bengio, Y. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

Draxler, F., Veschgini, K., Salmhofer, M., and Hamprecht, F. A. Essentially no barriers in neural network energy landscape. *arXiv preprint arXiv:1803.00885*, 2018.

Fort, S. and Scherlis, A. The goldilocks zone: Towards better understanding of neural network loss landscapes. *arXiv preprint arXiv:1807.02581*, 2018.

Gil, A., Segura, J., and Temme, N. M. *Numerical methods for special functions*, volume 99. Siam, 2007.

Github. Tensorflow models. https://github.com/tensorflow/models/blob/master/official/, 2017.

Golub, G. H. and Meurant, G. *Matrices, moments and quadrature with applications*, volume 30. Princeton University Press, 2009.

Golub, G. H. and Welsch, J. H. Calculation of gauss quadrature rules. *Mathematics of computation*, 23(106):221–230, 1969.

Goodfellow, I. J., Vinyals, O., and Saxe, A. M. Qualitatively characterizing neural network optimization problems, 2014.

Gur-Ari, G., Roberts, D. A., and Dyer, E. Gradient descent happens in a tiny subspace. *arXiv preprint arXiv:1812.04754*, 2018.

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

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

Jastrzębski, S., Kenton, Z., Arpit, D., Ballas, N., Fischer, A., Bengio, Y., and Storkey, A. Three factors influencing minima in sgd. *arXiv preprint arXiv:1711.04623*, 2017.

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., and Tang, P. T. P. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Kleinberg, R., Li, Y., and Yuan, Y. An alternative view: When does sgd escape local minima? *arXiv preprint arXiv:1802.06175*, 2018.

Lanczos, C. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018a.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. Visualizing the loss landscape of neural nets. In *Advances in Neural Information Processing Systems*, pp. 6391–6401, 2018b.

Lin, L., Saad, Y., and Yang, C. Approximating spectral densities of large matrices. *SIAM review*, 58(1):34–65, 2016.

Orhan, A. E. and Pitkow, X. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

Papyan, V. The full spectrum of deep net hessians at scale: Dynamics with sample size. *arXiv preprint arXiv:1811.07062*, 2018.

Pearlmutter, B. A. Fast exact multiplication by the hessian. *Neural computation*, 6(1):147–160, 1994.

Sagun, L., Bottou, L., and LeCun, Y. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

Sagun, L., Evci, U., Guney, V. U., Dauphin, Y., and Bottou, L. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

Santurkar, S., Tsipras, D., Ilyas, A., and Madry, A. How does batch normalization help optimization?(no, it is not about internal covariate shift). *arXiv preprint arXiv:1805.11604*, 2018.

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

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., and Wojna, Z. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

Ubaru, S., Chen, J., and Saad, Y. Fast estimation of tr(f(a)) via stochastic lanczos quadrature. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1075–1099, 2017.

Wu, L., Zhu, Z., et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

Yao, Z., Gholami, A., Lei, Q., Keutzer, K., and Mahoney, M. W. Hessian-based analysis of large batch training and robustness to adversaries. *arXiv preprint arXiv:1802.08241*, 2018.