A. Concentration of Quadratic Forms

The following lemma is one result on the concentration of quadratic forms:

Lemma A.1 (Concentration of Quadratic Forms, (Bellec, 2014)). *Let $\zeta \sim N(0, \sigma^2 I_n)$. Let $A \in \mathbb{R}^{n\times n}$ be any matrix. Then, $\forall x > 0$,*

$$P(\zeta^T A\zeta - \mathbb{E}\left[\zeta^T A\zeta\right] > 2\sigma^2\|A\|_F\sqrt{x} + 2\sigma^2\|A\|_2 x) \leq \exp(-x). \tag{10}$$

We are now ready to prove Claim 2.3.

Proof. Consider the block-diagonal matrix $A = \oplus_{i=1}^{k} f(H; t, \sigma^2)$. Then, $\widehat{\phi}_\sigma(t) = w^T A w$ where w is the concatenation of the k realizations of v divided by $\sqrt{k}$. Now observe that w is i.i.d $\mathcal{N}(0, \frac{1}{kn})$. Therefore, by Lemma A.1,

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \frac{2\|A\|_F}{kn}\sqrt{x} + \frac{2\|A\|_2}{kn}x\right) \leq 2\exp(-x).$$

Now observe that $\|A\|_F = \sqrt{k}\|f(H; t, \sigma^2)\|_F$ and $\|A\|_2 = \|f(H; t, \sigma^2)\|_2$. Therefore, we get

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \frac{2a}{n\sqrt{k}}\sqrt{x} + \frac{2b}{kn}x\right) \leq 2\exp(-x). \tag{11}$$

From (11) is clear that the bound deteriorates as a and b increase. Since $f(\cdot)$ is the Gaussian density, we know $b \leq \frac{1}{\sqrt{2\pi}\sigma}$ and $a \leq \sqrt{n}b$. Substituting these worst case scenario values in (11), we get

$$P\left(|\phi_\sigma(t) - \widehat{\phi}_\sigma(t)| > \sqrt{\frac{2}{\pi\sigma^2}}\left(\sqrt{\frac{x}{nk}} + \frac{x}{nk}\right)\right) \leq 2\exp(-x). \tag{12}$$

This proves our assertion. □

Figure 14 shows how $\epsilon(x)$ changes with respect to probability bound $2\exp(-x)$ in the worst case bound (8). We can see that even with modest values of k, we can achieve tight bounds on ϵ with high probability.

B. Numerical Verification on Small Models

Figure 15 shows how fast $\phi_\sigma^{(v)}$ converges to $\widehat{\phi}^{(v)}(t)$ as m increases in terms of total variation (L_1) distance.

Before going to large scale experiments, we empirically demonstrate the accuracy of our proposed framework on a small model where the Hessian eigenvalues can be computed exactly. Let's consider a feed-forward neural network trained on 1000 MNIST examples with 1 hidden layer of size 20, corresponding to $n = 15910$ parameters. The Hessian of networks of this type were studied earlier in (Sagun et al., 2017) where it was shown that, after training, the spectrum consists of a bulk near zero and a few outlier eigenvalues. In our example, the range $[-0.2, 0.4]$ roughly corresponds to the bulk and $(0.4, 10)$ corresponds to the outlier eigenvalues. Figures 1 and 16 compare our estimates with the exact smoothed density on each of these intervals. Our results show that with a modest number of quadrature points (90 here) we are able to approximate the density extremely well. Our proposed framework achieves $L_1(\phi_\sigma, \widehat{\phi}_\sigma) \approx 0.0012$ which corresponds to an extremely accurate solution. As demonstrated in Figure 16, our estimator detects the presence of outlier eigenvalues. Therefore, the information at the edges of ϕ_σ is also recovered.

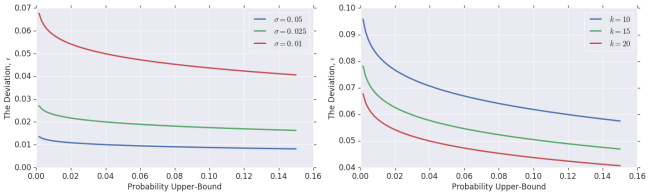

Figure 14. Examination of the worst-case tail bound for a network with $n = 5 \times 10^5$ parameters. Left figure: we set $k = 20$ and change the kernel parameter σ. Right figure: we set $\sigma = 0.01$ and change k.

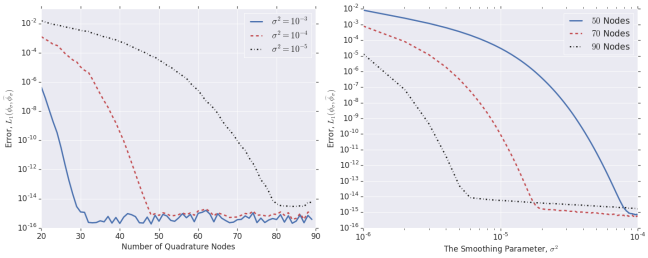

Figure 15. The left plot shows the accuracy of the Gaussian quadrature approximate as the number of nodes increases. A degree 80 approximation achieves double-precision accuracy of 10^{-14}. The right plot shows how the accuracy changes as the kernel width, σ^2, increases. For our large-scale experiments, we use $\sigma^2 = 10^{-5}$ and 90 quadrature nodes.

C. Implementation Details

The implementation of Algorithm 1 for a single machine is straightforward and can be done in a few lines of code. Scaling it to run on a 27 million parameter Inception V3 (Szegedy et al., 2016) on ImageNet (where we performed our largest scale experiments) requires a significant engineering effort.

The major component is a distributed Lanczos algorithm.

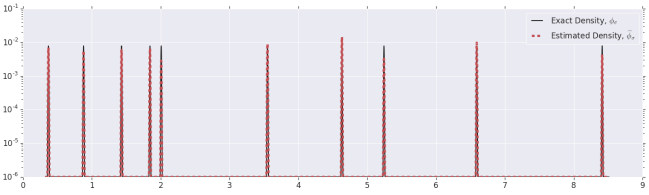

Figure 16. Comparison of the estimated smoothed density (dashed) and the exact smoothed density (solid) in the interval $[0.4, +\inf)$. We use $\sigma^2 = 10^{-5}$, $k = 10$ and degree 90 quadrature.

Because modern deep learning models and datasets are so large, it is important to be able to run Hessian-vector products in parallel across multiple machines. At each iteration of the Lanczos algorithm, we need to compute a Hessian-vector product on the entire dataset. To do so, we split the data across all our workers (each one of which is endowed with one or more GPUs), each worker computes mini-batch Hessian-vector products, and these products are summed globally in an accumulator. Once worker i is done on its partition of the data, it signals via semaphore i to the chief that it is done. When all workers are done, the chief computes completes the Lanczos iteration by applying a QR orthogonalization step to total Hessian-vector product. When the chief is done, it writes the result to shared memory and raises all the semaphores to signal to the workers to start on a new iteration.

For the Hessian-vector products, we are careful to eliminate all non-determinism from the computation, including potential subsampling from the data, shuffle order (this affects e.g., batch normalization), random number seeds for dropout and data augmentation, parallel threads consuming data elements for summaries etc. Otherwise, it is unclear what matrix the Lanczos iteration is actually using.

Although GPUs typically run in single precision, it is important to perform the Hessian-vector accumulation in double precision. Similarly, we run the orthogonalization in the Lanczos algorithm in double precision. TensorFlow variable updates are not atomic by default, so it is important to turn on locking, especially on the accumulators. TensorFlow lacks communication capability between workers, so the coordination via semaphores (untrainable tf.Variables) is crude but necessary.

For a CIFAR-10, on 10 Tesla P100 GPUs, it takes about an hour to compute 90 Lanczos iterations. For ImageNet, a Resnet-18 takes about 20 hours to run 90 Lanczos iterations. An Inception V3 takes far longer, at about 3 days, due to needing to use 2 GPUs per worker to fit the computation graph. We were unable to run any larger models due to an unexpected OOM bugs in TensorFlow. It should be straightforward to obtain a 50-100% speedup – we use the default TensorFlow parameter server setup, and one could easily reduce wasteful network transfers of model parameters from parameter servers for every mini-batch, and conversely from transferring every mini-batch Hessian-vector product back to the parameter servers. We made no attempt to optimize these variable placement issues.

For the largest models, TensorFlow graph optimizations via Grappler can dramatically increase peak GPU memory usage, and we found it necessary to manage these carefully.

D. Comparison with Other Spectrum Estimation Methods

There is an extensive literature on estimation of spectrum of large matrices. A large fraction of the algorithms in this literature relay on explicit polynomial approximations to f. To be more specific, these methods approximate $f(\cdot, t, \sigma^2)$ with a polynomial of degree m, $g_m(\cdot)$. In step I of Algorithm 1, $\phi_\sigma^{(v)}(t)$ is approximated by

$$\widehat{\phi}_{poly}^{(v)}(t) := \sum_{i=1}^{n} \beta_i^2 g_m(\lambda_i). \tag{13}$$

If $g_m(\cdot)$ is a good approximation for $f(\cdot; t, \sigma^2)$, we expect $\widehat{\phi}_{poly}^{(v)}(t) \approx \phi_\sigma^{(v)}(t)$.

Since g_m is a polynomial, (13) can be exactly evaluated as soon as

$$\mu_j^{(v)} \equiv \sum_{i=1}^{n} \beta_i^2 \lambda_i^j, \qquad 1 \le j \le m \tag{14}$$

are known. Note that by definition,

$$\mu_j^{(v)} = \sum_{i=1}^{n} (v^T q_i)^2 \lambda_i^j = v^T Q \Lambda^j Q^T v = v^T H^j v$$

Therefore, if done carefully, $\{\mu_j^{(v)}\}_{j=1}^m$ can be computed by performing m Hessian-vector products in total. Hence, by performing km Hessian-vector products one can run Algorithm 1 with k different realizations of v.

This approximation framework is arguably simpler than Gaussian quadrature method as it does not have to cope with complexities of Lanczos algorithm. Therefore, it is has been extensively used in the numerical linear algebra literature. The polynomial approximation step is usually done via Chebyshev polynomials. This class of polynomials enjoy strong computational and theoretical properties that make them suitable for approximating smooth functions. For more details on Chebyshev polynomials we refer the reader to (Gil et al., 2007).

Recently, there has been a proposal to use Chebyshev approximation for estimating the Hessian spectrum for deep

networks (Adams et al., 2018). For completeness, we compare the performance of this algorithm with the Gaussian quadrature rule on the feed-forward network defined earlier.

Figure 17 shows the performance of the Chebyshev method in approximating $\phi_\sigma(t)$. The hyper-parameters are selected such that the performance of the Chebyshev method in Figure 17 is directly comparable with the performance of Gaussian quadrature in Figure 1. In particular, both approximations take the same amount of computation (as measured by the number of Hessian-vector products) and they both use the same kernel width ($\sigma^2 = 10^{-5}$). As the figure shows, the Chebyshev method utterly fails to provide a decent approximation to the spectrum. As it can be seen from the figure, almost all of the details of the spectrum are masked by the artifacts of the polynomial approximation. In general, we expect the Chebyshev method to require orders of magnitude more Hessian-vector products to match the accuracy of the Gaussian quadrature.

It is not a surprise that explicit polynomial approximation fails to provide a good solution. For small kernel widths, extremely high order polynomials are necessary to approximate the kernel well. Figure 18 shows how well Chebyshev polynomials approximate the kernel f with $\sigma^2 = 10^{-5}$. The figure suggests that even with a 500 degree approximation, there is a significant difference between the polynomial approximation and the exact kernel.

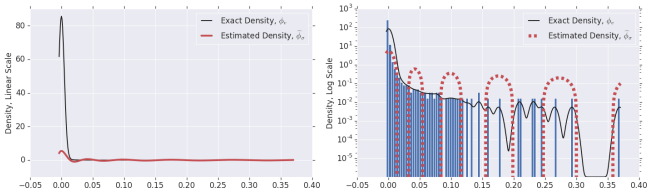

Figure 17. Estimated Hessian spectral density using Chebyshev approximation method for the feed-forward model. The left plot shows the densities in the linear scale and the right plot shows the densities in the log scale. Degree 90 polynomial was used to estimate the density. $\sigma^2 = 10^{-5}$ was used as the kernel parameter. To factor out the effects of noise in moment estimation, exact eigenvalue moments were provided to the algorithm.

E. Gradient Concentration in the Quadratic Case

In this section, we theoretically show the phenomenon of gradient concentration on a simple quadratic loss function with stochastic gradient descent. The loss function is of the form

$$\mathcal{L}(\theta) = \frac{1}{2}(\theta - \theta^*)^T H(\theta - \theta^*),$$

where the ordered (in decreasing order) eigenpairs of H are $(\lambda_i, q_i), i = 1, \cdots, n$ (implies $Hq_i = \lambda_i q_i$) and the

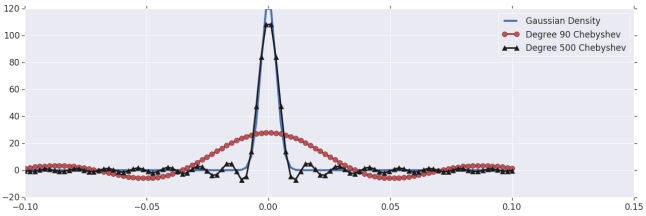

Figure 18. Demonstrating the quality of Chebyshev polynomial approximation to the Gaussian kernel with $\sigma^2 = 10^{-5}$. The plot suggests that approximations of order 500 or more are necessary to achieve accurate results. Such high order approximations are statistically unstable and extremely computationally expensive.

iteration starts at $\theta_0 \sim \mathcal{N}(0, I_n)$. We model the stochastic loss (from which we compute the gradients for SGD) as

$$\hat{\mathcal{L}}(\theta) = \frac{1}{2}(\theta - \theta^* + z)^T H(\theta - \theta^* + z),$$

where z is a random variable such that $\mathbb{E}[z] = 0$ and $\mathbb{E}\left[zz^T\right] = \mathcal{S}$. In order to understand gradient concentration, we look at the alignment of individual SGD updates with individual eigenvectors of the Hessian. We are now ready to prove the following theorem.

Theorem E.1. *Consider a single gradient descent iteration, $\theta_{t+1} = \theta_t - \eta \nabla \hat{\mathcal{L}}$ with a constant learning rate $\eta \approx c/\lambda_1$ for a constant $c < 1$. Then,*

$$\mathbb{E}\left[\langle q_i, (\theta_{t+1} - \theta_t)\rangle^2\right] \to \alpha \cdot \left(\frac{\lambda_i}{\lambda_1}\right)^2 \cdot (q_i^T \mathcal{S} q_i) \qquad (15)$$

for some sufficiently large constant α as $t \to \infty$.

Proof. Each stochastic gradient step has the form $\theta_t = \theta_{t-1} - \eta H(\theta_{t-1} + z_{t-1})$. Expanding the recurrence induced by gradient step over t steps, we can write

$$\theta_t = (I_n - \eta H)^t \theta_0 - \eta H \sum_{j=0}^{t-1} (I_n - \eta H)^j z_{t-j-1}.$$

Therefore a single update $\theta_{t+1} - \theta_t = -\eta H(\theta_t + z_t)$ can be expanded as

$$\theta_{t+1} - \theta_t = -\eta H[(I_n - \eta H)^t \theta_0 - \eta H \sum_{j=0}^{t-1} (I_n - \eta H)^j z_{t-j-1} + z_t]$$

We can write the above equation as $-\eta H(T_1 + T_2)$, where

$$T_1 = (I_n - \eta H)^t \theta_0$$
$$T_2 = -\eta H \sum_{j=0}^{t-1} (I_n - \eta H)^j z_{t-j-1} + z_t$$

Consider the dot product of this update with one of the eigenvectors q_i. Clearly from the form of the update $\mathbb{E}\left[\langle q_i, \theta_{t+1} - \theta_t \rangle\right] \xrightarrow{t \to \infty} 0$. We now quantify the variance of the update in the direction of q_i. Using the identity $Hq_i = \lambda_i q_i$, it is easy to see that

$$q_i^T \eta H T_1 = \eta \lambda_i (1 - \eta \lambda_i)^t q_i^T \theta_0$$
$$q_i^T \eta H T_2 = -\eta^2 \lambda_i^2 \sum_{j=0}^{t-1} (1 - \eta \lambda_i)^j q_i^T z_{t-j-1} + \eta \lambda_i q_i^T z_t$$

Squaring the sum of the two terms above and taking expectations, only the squared terms survive. We write the

$$\mathbb{E}\left[\langle q_i, (\theta_{t+1} - \theta_t) \rangle^2\right] = \eta^2 \lambda_i^2 (1 - \eta \lambda_i)^{2t} + \left(\eta^4 \lambda_i^4 \sum_{j=0}^{t-1} (1 - \eta \lambda_i)^{2j} + \eta^2 \lambda_i^2 \right) \cdot (q_i^T \mathcal{S} q_i)$$

As $t \to \infty$, the first term above goes to 0. This suggests that in the absence of noise in the gradients there is no reason to expect any alignment of the gradient updates with the eigenvectors of the Hessian. However, the second term (after some algebraic simplification) can be written as

$$\frac{2\eta^2 \lambda_i^2}{2 - \eta \lambda_i} \cdot (q_i^T \mathcal{S} q_i).$$

Parameterizing $\eta = c/\lambda_1$ completes the proof.

□

A couple of observations are appropriate here. We can see that as the separation of eigenvalues increases, gradient updates align quadratically with the top eigenspaces. By manipulating the alignment of $\mathcal{S}$ with the top eigenspaces of H, we can dramatically change the concentration of updates. For example, if $\mathcal{S}$ was similar to H, the alignment with the top eigenspaces can be enhanced. If $\mathcal{S}$ was similar to H^{-1}, the alignment with the top eigenspaces can be diminished. We have seen that, even in practice, if we could control the noise in the gradients, we can hamper or improve optimization in significant ways.

F. Experimental Details

On CIFAR-10, our models of interest are:

Resnet-32: This model is a standard Resnet-32 with 460k parameters. We train with SGD and a batch size of 128, and decay the learning from 0.1 by factors of 10 at step 40k, 60k, 80k. This attains a validation of 92% with data augmentation (and around 85% without)

VGG-11: This model is a slightly modified VGG-11 architecture. Instead of the enormous final fully connected layers, we are able to reduce these to 256 neurons with only a little degradation in validation accuracy (81% vs 83% with a 2048 size fully connected layers). We train with a constant SGD learning rate of 0.1, and a batch size of 128. This model has over 10 million parameters.

To ensure that our models have a finite local minimum, we introduce a small label smoothing of 0.1. This does not affect the validation accuracy; the only visible effect is that the lowest attained cross entropy loss is the entropy 0.509.

On Imagenet, our primary model of interest is Resnet-18. We use the model in the official Tensorflow Models repository (Github, 2017). However, we train the model on 299×299 resolution images, in an asynchronous fashion on 50 GPUs with an exponentially decaying learning rate starting at 0.045 and batch size 32. This attains 71.9% validation accuracy. This model has over 11 million parameters.

G. Batch normalization with population statistics

The population loss experiment is quite difficult to run on CIFAR-10 (we were unable to make Inception V3 train in this way without using a tiny learning rate of 10^{-6}). In particular, it is important to divide the learning rate by a factor of 100, and also to spend at least 400 steps at the start of optimization with a learning rate of 0: this allows the batch normalization population statistics to stabilize with a better initialization than the default mean of 0.0 and variance of 1.0.

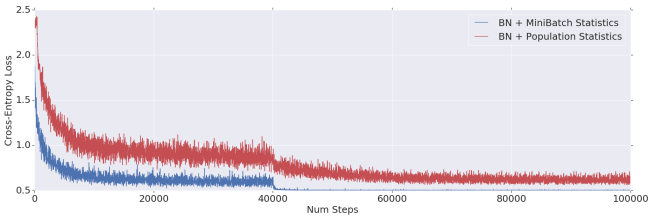

Figure 19. Optimization progress (in terms of loss) of batch normalization with mini-batch statistics and population statistics.