Salil Deshpande 3/23/2020

Approximating The Spectra Of Neural Network Hessians

Introduction

This project is based on the work done by Yao et al. in their January 2020 paper, *PyHessian: Neural Networks Through the Lens of the Hessian* [1]. As a part of that paper, they developed a PyTorch library, PyHessian, that is built to compute spectral quantities of Neural Network hessians [2]. As a part of this project, I extended and made some modifications to some of their algorithms. My code can be found at [3].

Background

In supervised learning problems, we seek to minimize an empirical loss function,

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

where $\{(x_i, y_i)\}_{i=1}^{N}$ represent pairs of input/output data, M represents the model, θ represents the learning parameters, and $l(M(x_i), y_i, \theta)$ represents a loss function based on prediction $M(x_i)$ and labeled output y_i. The Hessian matrix and of this empirical loss function holds important information about how the model is behaving and can help inform design choices related to the model architecture [2]. In particular, the top eigenvalues and trace of these Hessians have proven to be effective in providing insight about training performance and giving insight into loss function geometry.

The top eigenvectors of the Hessian describe sources of curvature in neural network loss landscapes. Identifying the top few eigenvalue/eigenvector pairs has given risen to a coarse-grained technique for visualizing loss landscapes. The plots in Figure 1, originally published in [1], show the loss function at the training optimum when perturbed along the directions of the top eigenvectors. This is a visualization of the "most significant" 2D surface that approximates the loss landscape.

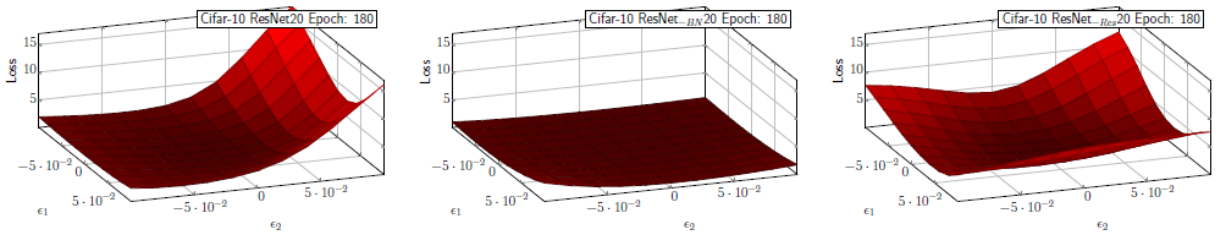

Figure 1: A set of loss function visualizations published in [1]**, created by perturbing the loss function at the training optimum along the directions of the top two eigenvectors.**

Top eigenvalue information has also been used to track the progress of training and give crucial insights into inadequacies of current training methods. Neural network loss space is extremely high-dimensional and many directions have small slopes and little curvature. It has been observed that during training, gradient updates often move in directions of high curvature and do not optimize the model in the flatter directions. However, a large fraction of the path to the optima comes from low-curvature directions. [4] demonstrates that if the top eigenvalues are high outliers, that can cause important low-curvature directions to be ignored, and for training to become less efficient. This insight implies that by

tracking the top eigenvalues of the Hessian during training and choosing to train in low-curvature directions which have not been exploited, a lower minimum can be reached.

The top few eigenvalues also seem to give insight into how amenable a dataset may be for classification. One interesting result seems to indicate that as a data set becomes easier to group in a topological sense, the Hessian spectrum will have little change other than the top eigenvalues increasing [5].

The trace of the Hessian is considered to be an effective metric for quantifying model smoothness. This measure is motivated by the fact that the trace of any matrix is equal to the sum of its eigenvalues and the eigenvalues of the Hessian describe the curvature of the loss function. Because of this, tracking the Hessian trace over the course of training has been used to assess model smoothness, which is an indicator for training efficiency. Note that model smoothness is considered to be an indicator for training efficiency because it is believed that neural networks often converge to and are often attracted to flat, low-curvature basins. This belief is supported by the observation that when looking at the Hessian, training is a process by which many positive eigenvalues “migrate” to 0, thereby decreasing the Hessian trace [5].

Because Hessian trace has been used as a metric for training efficiency, it has been used as a quantitative criterion for comparing network architectures [1]. By tracking the trace over the course of training for different architectures, one can quantify how efficient their training was. This is a particularly useful metric to compare and distinguish architectures that have similar test accuracies. One can even determine which layers within an architecture are slow and need improvement by identifying layers that maintain high trace.

There are a few important properties of neural network Hessians that must be discussed before proceeding. First, neural network Hessians are extremely singular. Neural networks often have hundreds of thousands of parameters, and neural network Hessians correspondingly have hundreds of thousands of eigenvalues. Many of these eigenvalues are 0, and many more are extremely close to 0. Figure 2 below shows the Cumulative Spectral Density of the neural network I worked with in this project; the plot shows how well over 99% of the eigenvalues are below 0.3. The second important property of neural network Hessians to note is that they are often not fully PSD. More than just being an artifact of numerical errors, some negative eigenvalues are thought to represent or be related to prediction loss within individual training samples [6].

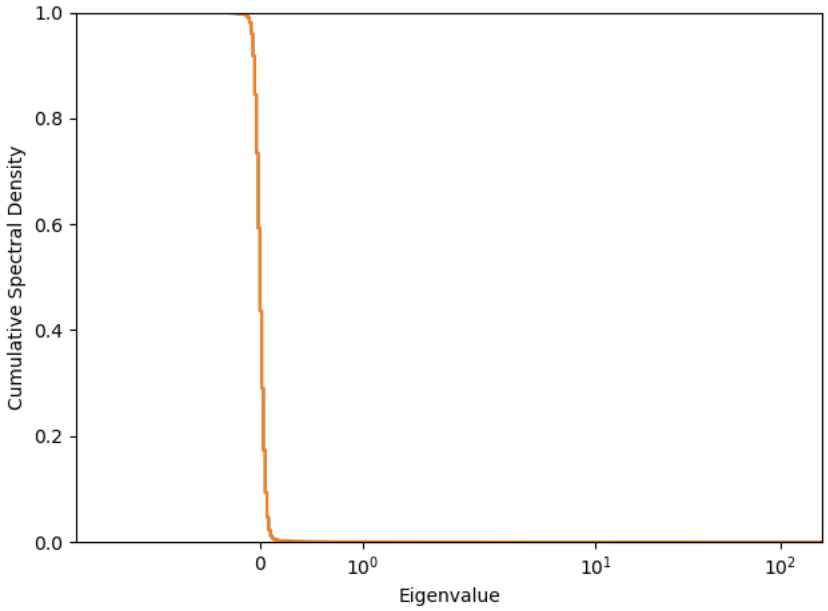

Figure 2: Cumulative Spectral Density of a ResNet20 NN trained on the Cifar-10 dataset. The curve plots what proportion of eigenvalues are above a given value.

Unfortunately, it turns out that neural network Hessians are very difficult to study, since it is often infeasible and unrealistic to build full Hessians. This is because there is no simple closed-form representation of the loss function from which second derivatives can be calculated and because the Hessian is often extremely large and can have billions of entries.

But, while building the Hessian itself is very difficult, the same computational graph used in backpropagation can be utilized to compute the product of the Hessian and a vector in a reasonable amount of time [1]. This product, known as a matvec product, is the building block of algorithms which seek to access the Hessian. Methods which seek to access Hessian spectra cannot rely on matrix manipulations and are thus called matrix-free methods.

In this paper, I consider matrix-free methods for approximating Hessian spectra. In particular, I look at methods that compute top eigenvalues and trace of the Hessian. I provide some comparison of the methods I study and the methods built-in to the PyHessian library.

My experiments were performed on a ResNet20 residual neural network trained on the Cifar-10 dataset. The Hessian had a size of $272{,}474 \times 272{,}474$.

Top-k Eigenvalues

PyHessian computes the top-k eigenvalues using power iteration. The top eigenvalue is computed using standard power iteration. Subsequent eigenvalues are computed by applying power iteration to a test vector that is orthogonalized with respect to previously identified eigenvectors.

The rate of convergence of power iteration depends on how closely the eigenvalues are distributed. In particular, when computing eigenvalue λ_i, this power iteration scheme will converge with at a rate of $O\left({}^{\lambda_{i+1}}/_{\lambda_i}\right)$. This scheme also has a small quadratic dependence due to having to orthogonalize eigenvector $i+1$ with respect to the previous i eigenvectors. This orthogonalization,

however, takes much less time than the time it takes to compute matvec products, and is thus has negligible effect on the overall speed of the algorithm.

Figure 3 below shows the time that it took PyHessian to compute the top 20 eigenvalues of the Hessian. Overall, it took around 1 hour and 55 minutes and 349 matvec products to compute the top 20 eigenvalues. And, because the eigenvalues became more closely packed after the first few, the number of iterations required to calculate each eigenvalue increased for the smaller eigenvalues.

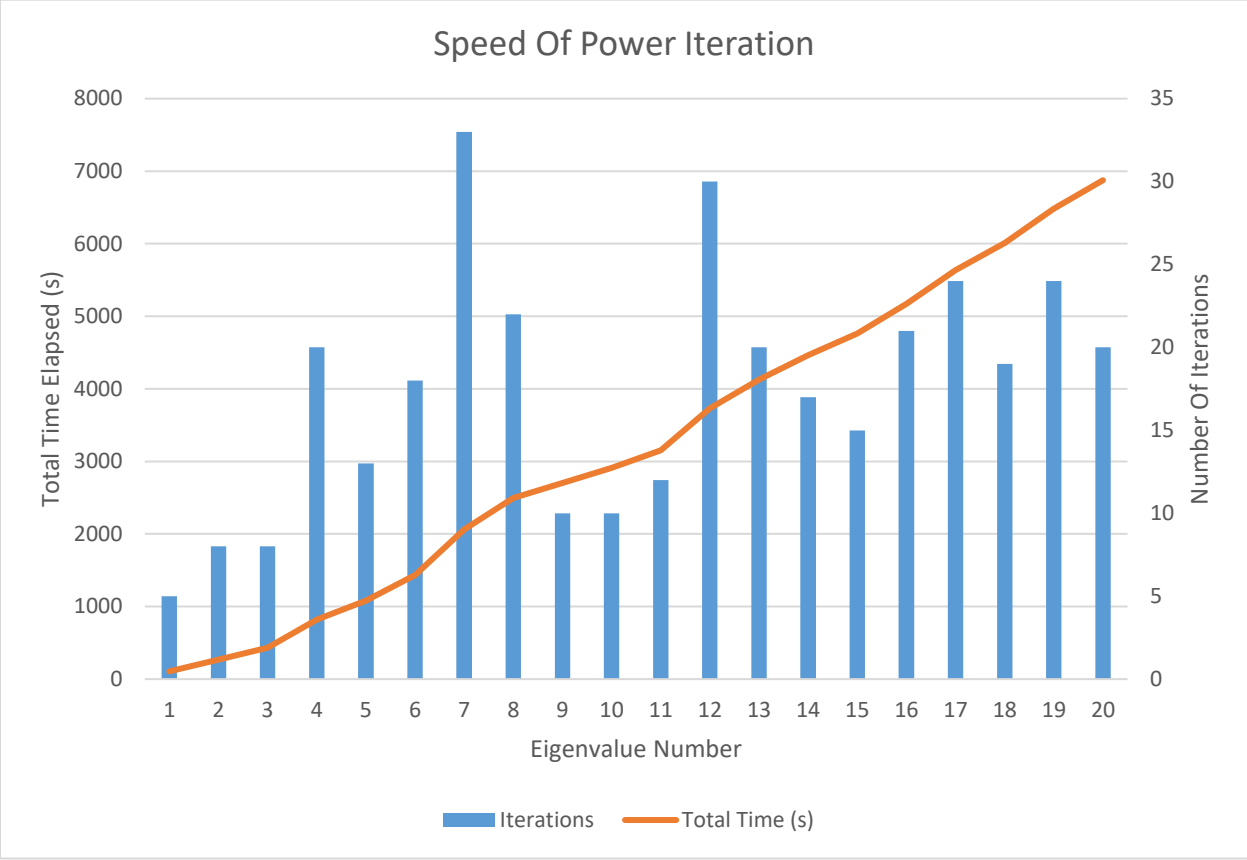

Figure 3: Total speed of Power Iteration for computing top 20 eigenvalues of NN. Read the orange curve on the left axis and the bars on the right axis.

As an alternative to the power method, I studied computing the top-k eigenvalues using the Lanczos Method. Algebraically, the Lanczos Method works by iteratively generating a Graham Shmidt orthonormal basis of a symmetric matrix's Krylov subspace. That is, starting from randomly chosen vector q_1, this algorithm finds an orthonormal set of vectors $\{q_i\}_{i=1}^{k}$ that span $\{q_1, Hq_1, \dots, H^{k-1}q_1\}$. The vectors $\{q_i\}_{i=1}^{k}$ are computed in such a way that they form a partial orthogonalization of the matrix such that when $\{q_i\}_{i=1}^{k}$ form the columns of Q, $Q^T HQ$ is equal to a matrix $T \in \mathbb{R}^{k\times k}$ tridiagonal (Upper Hessenberg and in particular, tridiagonal). After k iterations, the eigenvalues of T give an approximation for the top-k eigenvalues of H. In successive iterations, the estimates for the top-k eigenvalues are refined and converge towards their true value. More implementation and analysis details can be found in [7] [8].

I began the orthogonalization with a vector drawn from the standard normal distribution. Then, I computed tridiagonal T using the Lanczos algorithm. I had to run the algorithm for 100 iterations in

order to build $T \in \mathbb{R}^{100\times100}$. Eigenvalues of T can then be calculated using QR iterations, where each QR decomposition is performed using Givens Rotations. One advantage of the Lanczos Method is that because T is tridiagonal, QR iterations will run in linear time when the QR decompositions are performed with shift and deflation [7]. Figure 4 below depicts the convergence of the top eigenvalues as the algorithm proceeds.

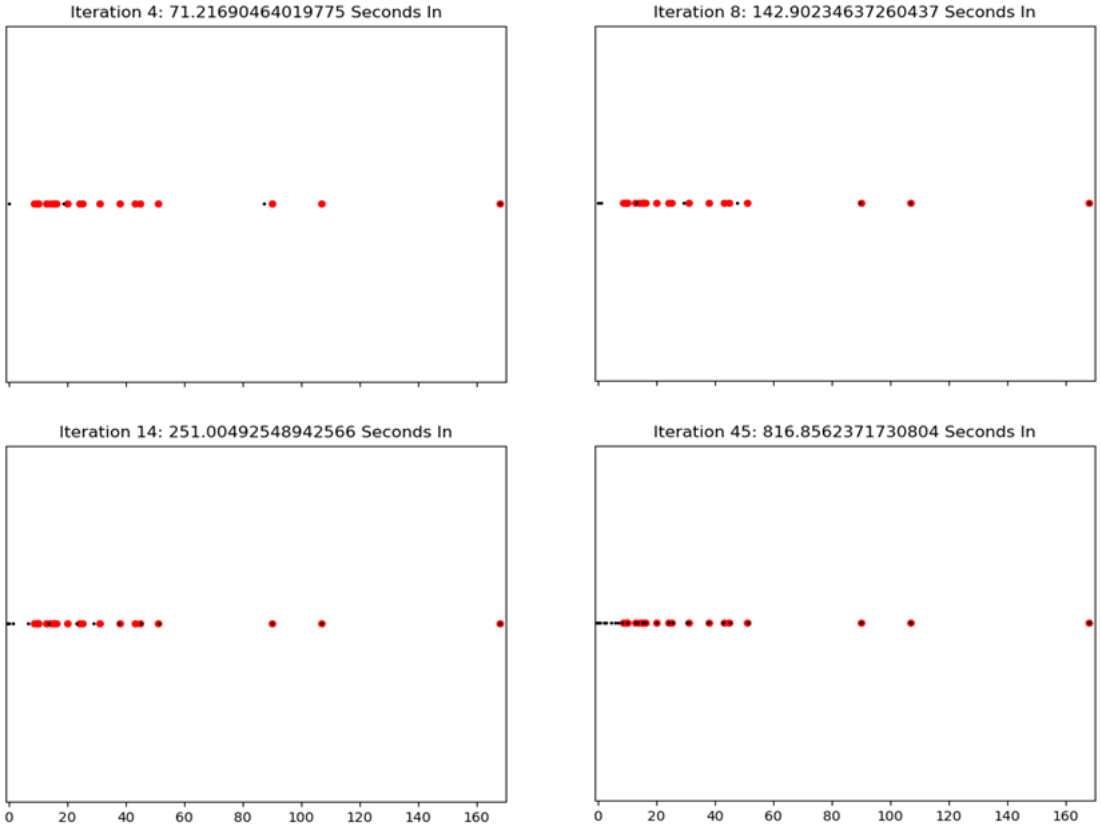

Figure 4: Convergence of Lanczos Method. The red dots represent the true eigenvalues as calculated with power iteration and the black dots represent the eigenvalues as calculated with the Lanczos Method.

100 iterations ran in around 30 minutes. By iteration 4, which was 70 seconds in, the first eigenvalue had converged. By iteration 8, which was 142 seconds in, the top 2 eigenvalues had converged. By iteration 14, which was a little over 4 minutes in, the top 4 eigenvalues had converged. By iteration 45, which was around 13 and a half minutes in, the top 20 eigenvalues had all converged. Figure 5 below shows the time it took the Lanczos Method to complete 100 iterations.

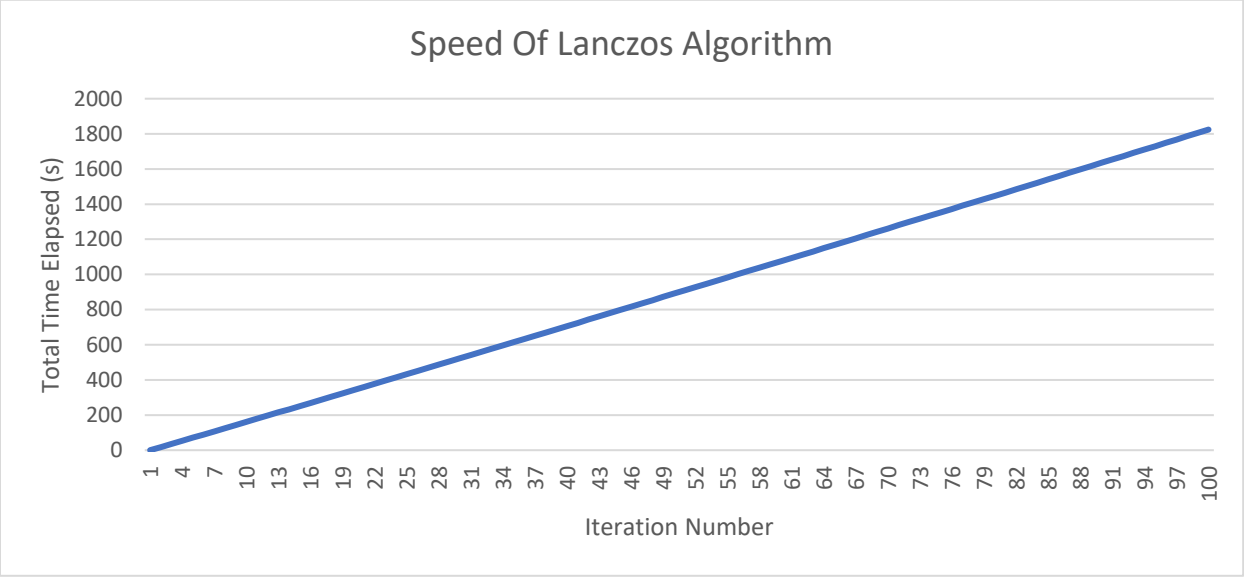

Figure 5: The speed of the Lanczos Algorithm. This algorithm runs in linear time. This plot does not include the time to find eigenvalues of T tridiagonal by QR Iterations.

In order to compute the top 20 eigenvalues of the Hessian, Power Iteration needed to compuate 349 matvec products while the Lanczos Method needed to compute 45. This disparity explains the difference in efficiency of the two algorithms. The Lanczos Method performs much better than Power Iteration because it has to perform much fewer matvec products, which are the calculations which take the most amount of time in this setting. This advantage is gained because the Lanczos Method simultaneously converges multiple eigenvalues every iteration.

Hessian Sketching

I explored Hessian Sketching as a randomized method to produce a matrix whose spectrum is an effective approximation of the Hessian. Having a sketch of the Hessian would be appealing because it could be used to perform traditional linear algebraic calculations without having to rely on matrix-free methods. Note that because it is easy to compute matvec products, it is just as easy to compute the product of the Hessian and an arbitrary matrix – given a matrix $A \in \mathbb{R}^{n\times d}$, for instance, the k columns of HA can all be computed as matvec products.

Motivated by this, I looked at sketches of the form S^THS. I chose $S \in \mathbb{R}^{n\times d}$ to have columns $\left\{-\frac{1}{\sqrt{d}}, +\frac{1}{\sqrt{d}}\right\}^n$ – Rademacher vectors scaled by $d^{-1/2}$. This sketch is unusual because $\mathbb{E}[S^TS] \neq I_d$ but rather $\mathbb{E}[S^TS] = \frac{n}{d}I_d$. This sketch has a number of attractive properties.

First, this sketch is trace preserving. As shown in the formula below, each of the d elements along the diagonal of the sketch represent one of the d elements in the sum that define the Hutchinson Trace Estimator.

$$\mathbb{E}[Tr(S^THS)] = \mathbb{E}\left[\sum_{i=1}^{d} s_i^THs_i\right] = \frac{1}{d}\mathbb{E}\left[\sum_{i=1}^{d}(\{-1,1\}^n)^TH(\{-1,1\}^n)\right] = Tr(H)$$

Secondly, this sketch seems to maintain the positive semi-definiteness of the Hessian to a large extent. Figure 6 below shows the Cumulative Spectral Density of a Hessian Sketch. The sketch has some negative eigenvalues as does the original Hessian, but very few of the eigenvalues are negative. In the particular realization of the sketch shown below, 4 out of the 50 eigenvalues were negative, and they were very close to 0. I observed the same behavior for a larger sketch size of 100.

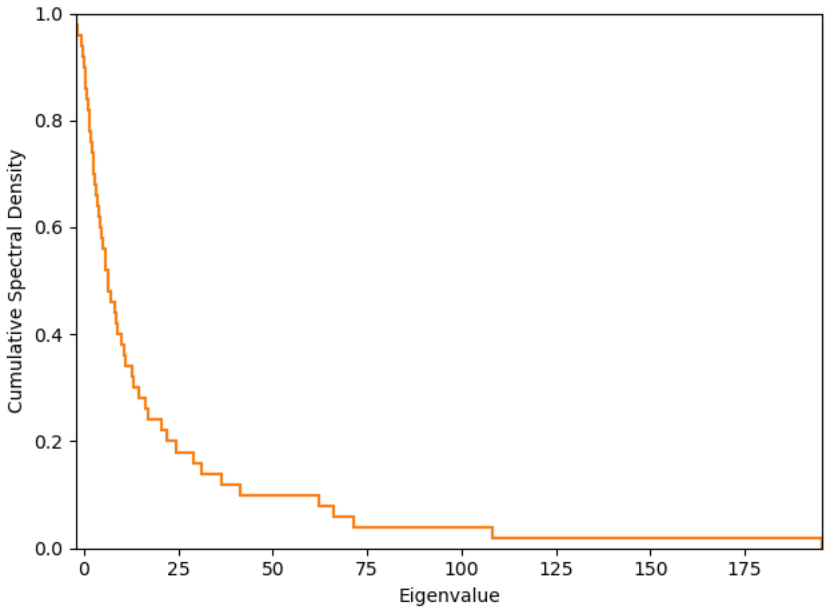

Figure 6: Cumulative Spectral Density of a Hessian Sketch of size 50. 4 eigenvalues were negative and 10 eigenvalues were of absolute value less than 10. The maximum eigenvalue was 195.

The implication of these properties is as follows. This sketch is of a much lower dimension than the Hessian, meaning that it has much fewer eigenvalues. Because it seems to generally maintain the PSD property, the sketch's eigenvalues are non-negative (or close to 0 when negative). Moreover, because the sketch is trace preserving, the sum of the its eigenvalues are equal to the sum of the Hessian's eigenvalues. Therefore, the sketch's eigenvalues should over-estimate the Hessian's eigenvalues, and over-estimate them in such a way as to compensate for the eigenvalues "lost" or "ignored" due to the dimension reduction. That is, the over-estimates should be proportional to the value of the "lost" eigenvalues.

It is clearly true that the average eigenvalue in the sketch is greater than the average eigenvalue of the Hessian. But, a priori, there is no guarantee that all the eigenvalues of the sketch are greater than the eigenvalues of the Hessian. I wondered, however, if the eigenvalues of the sketch might uniformly over-estimate the eigenvalues of the Hessian. If so, then despite being an over-estimate the eigenvalue approximations could prove to be useful. In particular, I hypothesized that when the Hessian was very singular, the sketch will ignore a larger percentage zero eigenvalues, preserving the significant eigenvalues and making the eigenvalue approximation more accurate.

I began by testing my hypothesis on small, randomly generated Hessian matrices. Because the mapping of functions to PSD matrices is surjective, I generated random Hessians by generating random PSD matrices. I generated matrices $H \in \mathbb{R}^{1000\times1000}$ to have a variable number of zero eigenvalues and non-zero eigenvalues pulled from Student's T-1 distribution. It was important to be able to control the

proportion of zero eigenvalues because a characteristic property of neural network Hessians is that they are very singular. I chose to pull the remaining eigenvalues from Student's T-1 distribution because the distributions of neural network eigenvalues often have long tails (see Figure 2), and Student's T-1 distribution has large, non-exponentially decaying tails.

For six different proportions of zero eigenvalues, I generated 3 Hessian matrices as specified above. For each matrix, I generated 10 realizations of a Hessian sketch for each of 6 different sketch sizes. For each of these sketches, I computed the top 20 eigenvalues and compared them to the top 20 eigenvalues of the corresponding original Hessian. Plots showing the percent error between sketched eigenvalue and true eigenvalue are shown below in Figures 7-12.

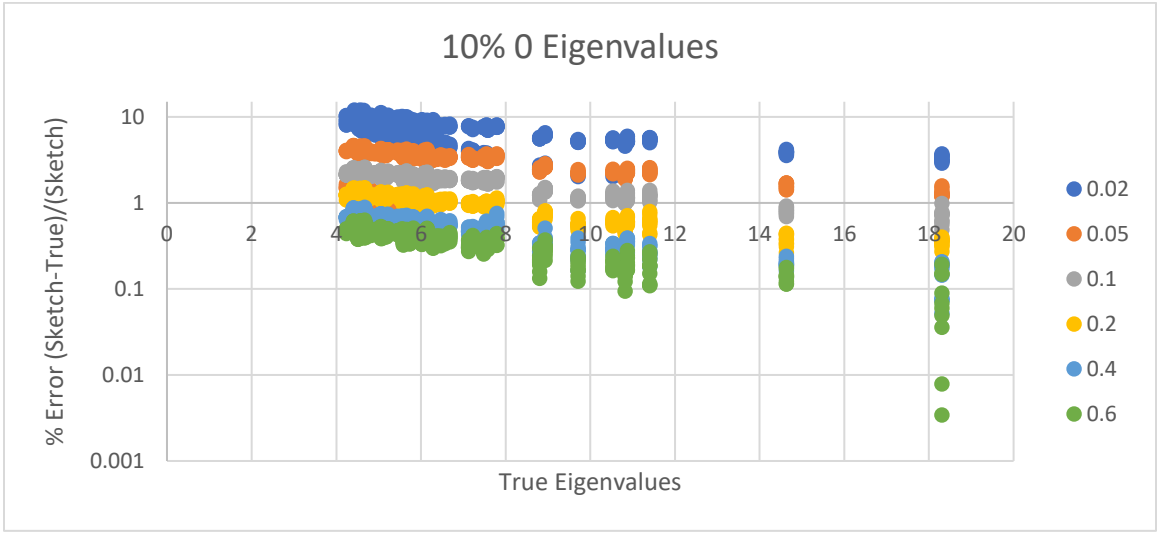

Figure 7: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 10% 0 eigenvalues.

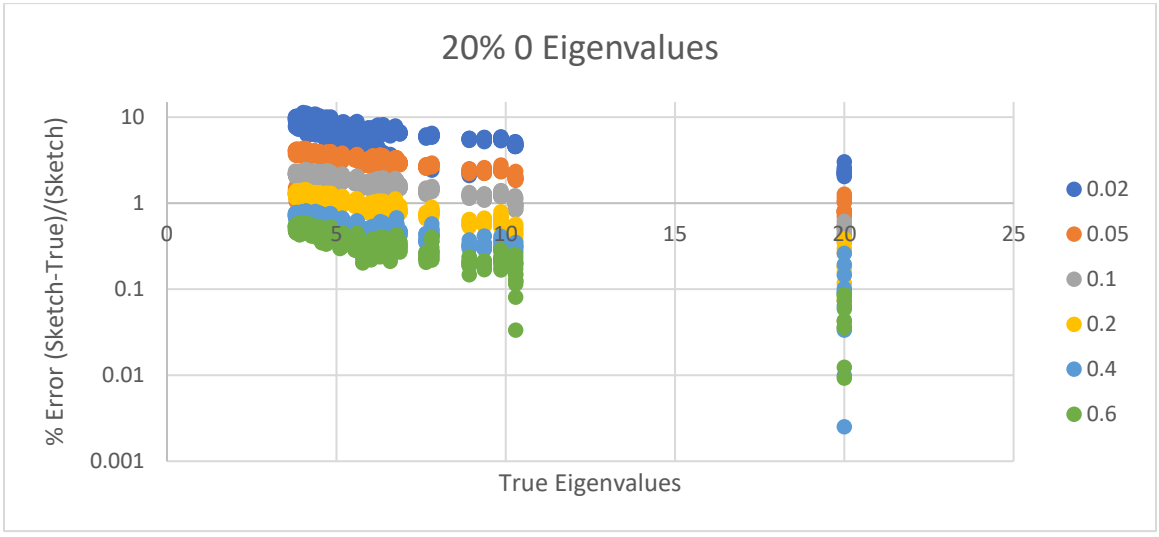

Figure 8: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 20% 0 eigenvalues.

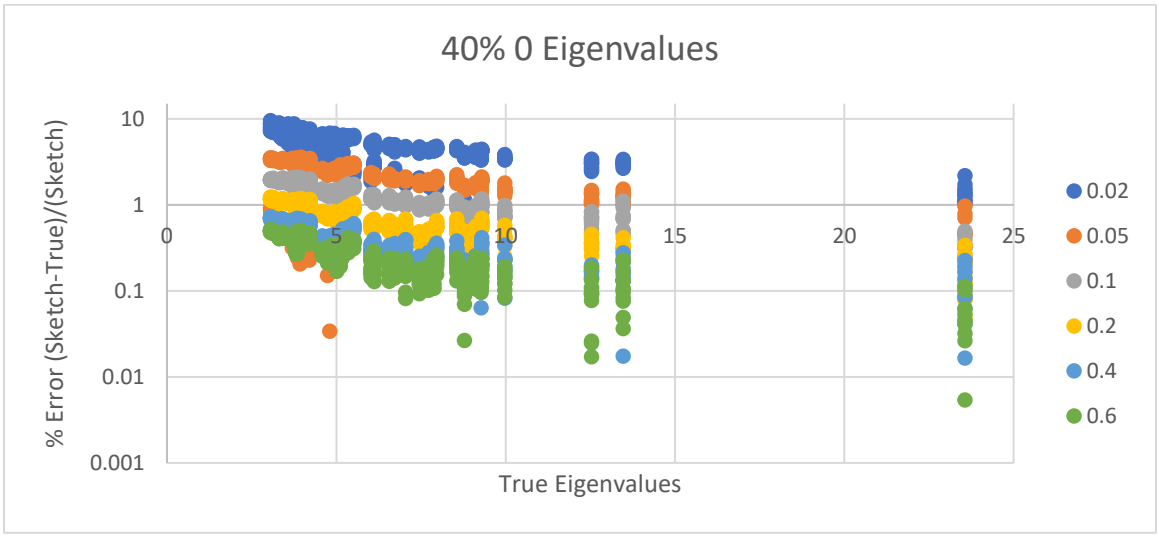

Figure 9: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 40% 0 eigenvalues.

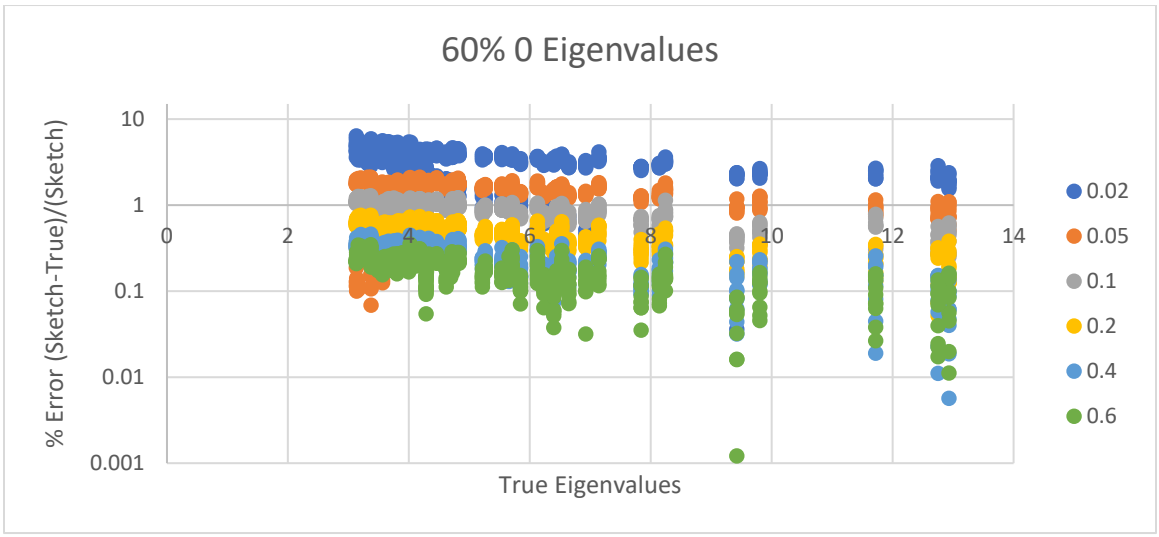

Figure 10: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 60% 0 eigenvalues.

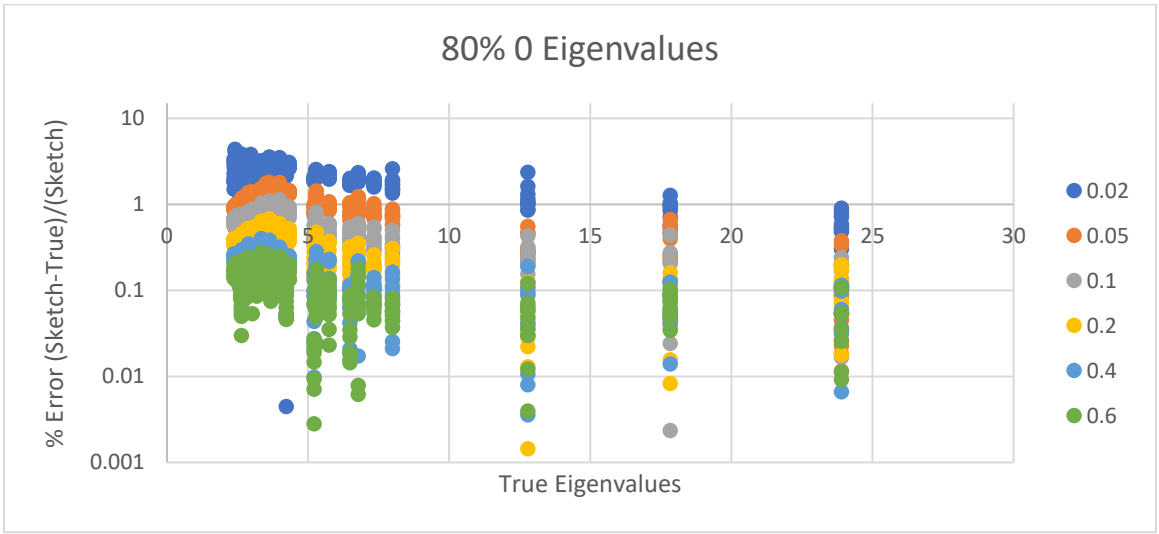

Figure 11: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 80% 0 eigenvalues.

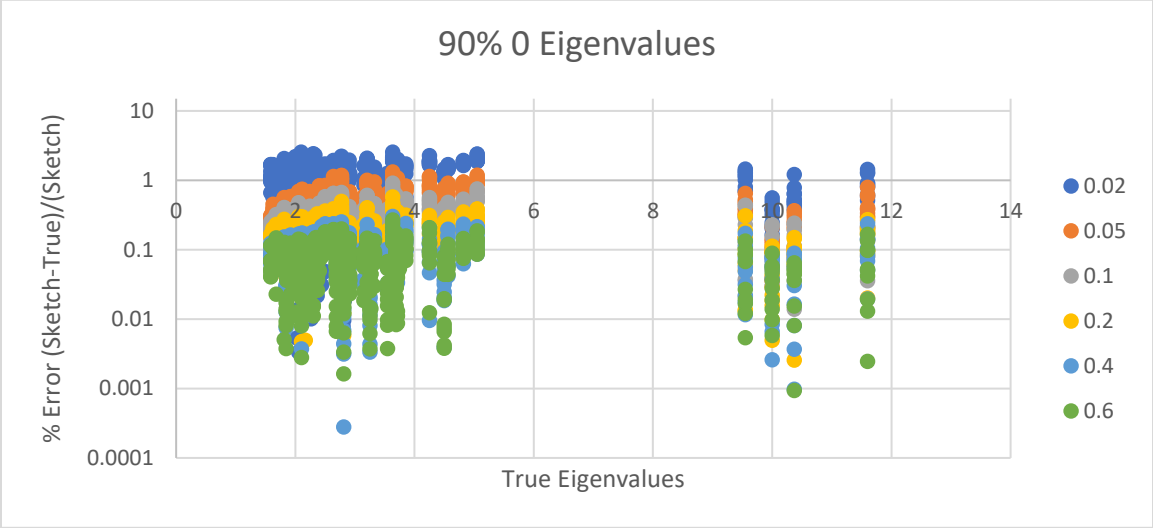

Figure 12: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 90% 0 eigenvalues.

Note that in the error plots, the percent error and not the absolute percent error is plotted. Despite this, all of the errors were positive, meaning that in all realizations of all of the sketches, the top eigenvalues were always over-estimated and never under-estimated. This seems to imply that in order to maintain the trace estimate at a lower dimension, the distribution of eigenvalues ended up being uniformly shifted up. The eigenvalue over-estimates are uniform and seem to decrease for higher eigenvalues (note the slight downward trend in each of the series of data).

Within each plot, each series of data represents the ratio of the sketched dimension to the original dimension. The 0.02 series, for example, had a sketch size ratio of $20:1000$. Across all of the plots, one very clear trend is that as the sketch size increases, the eigenvalue estimates become more accurate. This trend is expected because as the sketch size increases and fewer eigenvalues are lost to the dimension reduction, the error should decrease.

Across the plots, there is another noticeable trend for error estimates. As the Hessians become more singular, the eigenvalue estimates become more accurate. This trend can be seen more clearly in Figures 13-18. In the plots below, the series in each plot represent how singular the proportion of eigenvalues that are 0. At all sketching levels, as the Hessian becomes more singular, the error decreases.

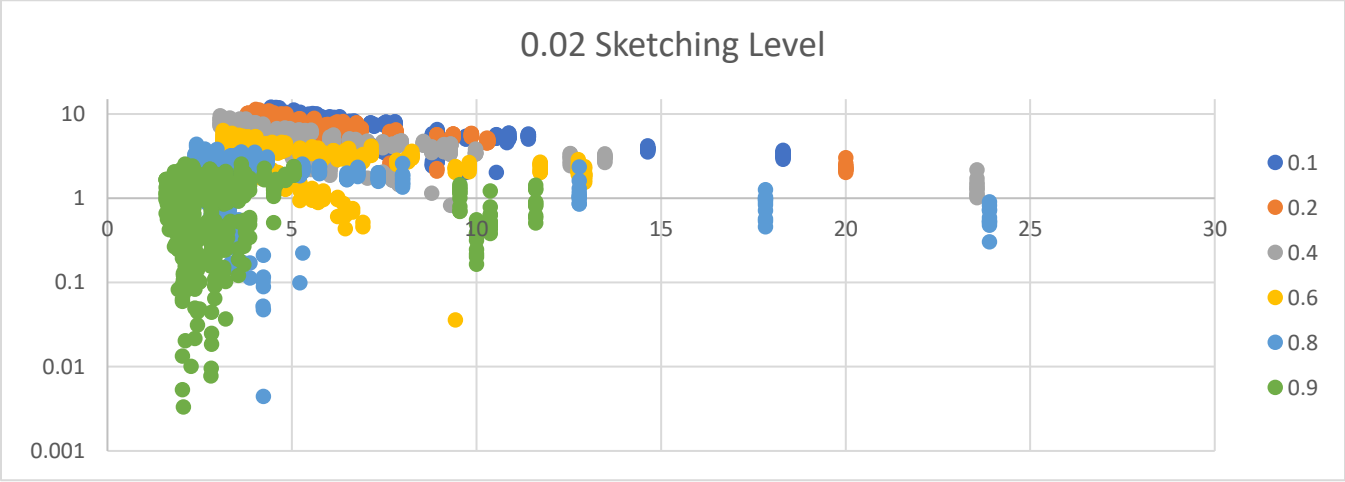

Figure 13: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.02 Sketch-Size Ratio

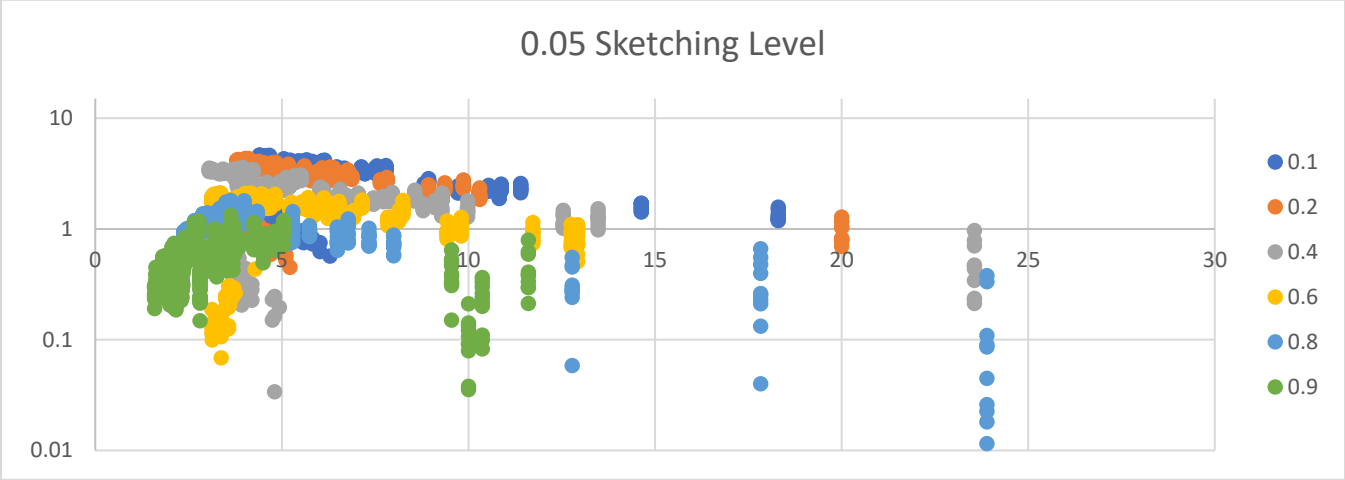

Figure 14: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.05 Sketch-Size Ratio

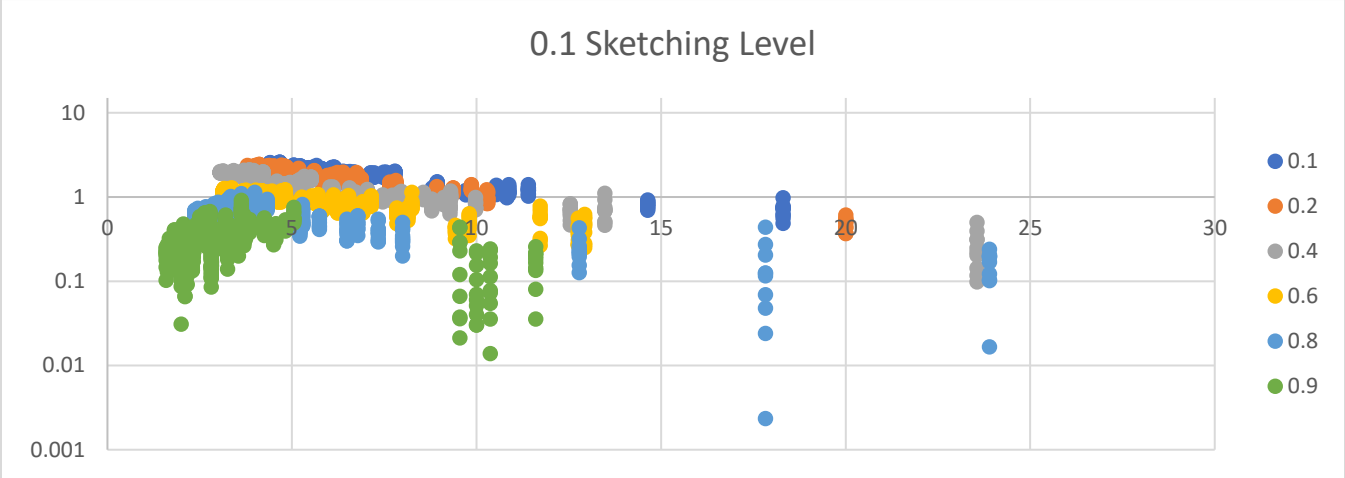

Figure 15: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.1 Sketch-Size Ratio

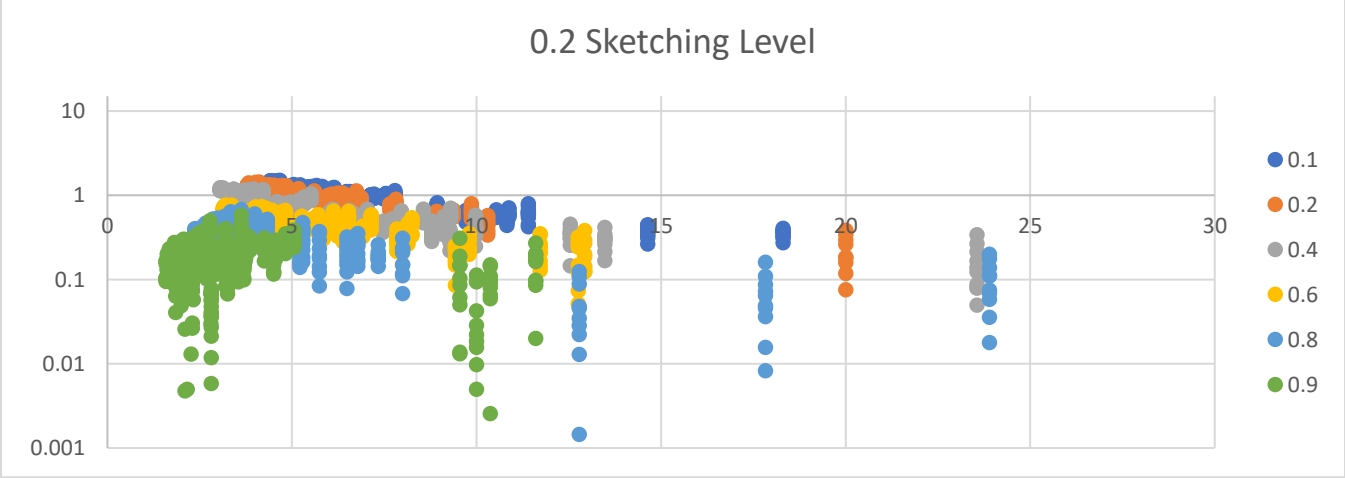

Figure 16: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.2 Sketch-Size Ratio

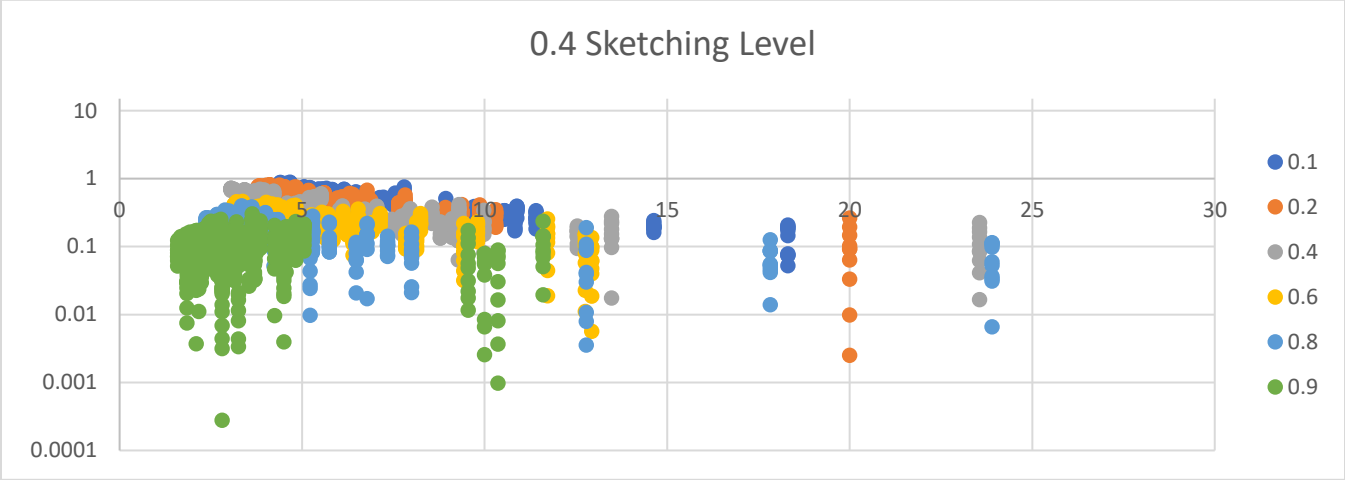

Figure 17: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.4 Sketch-Size Ratio

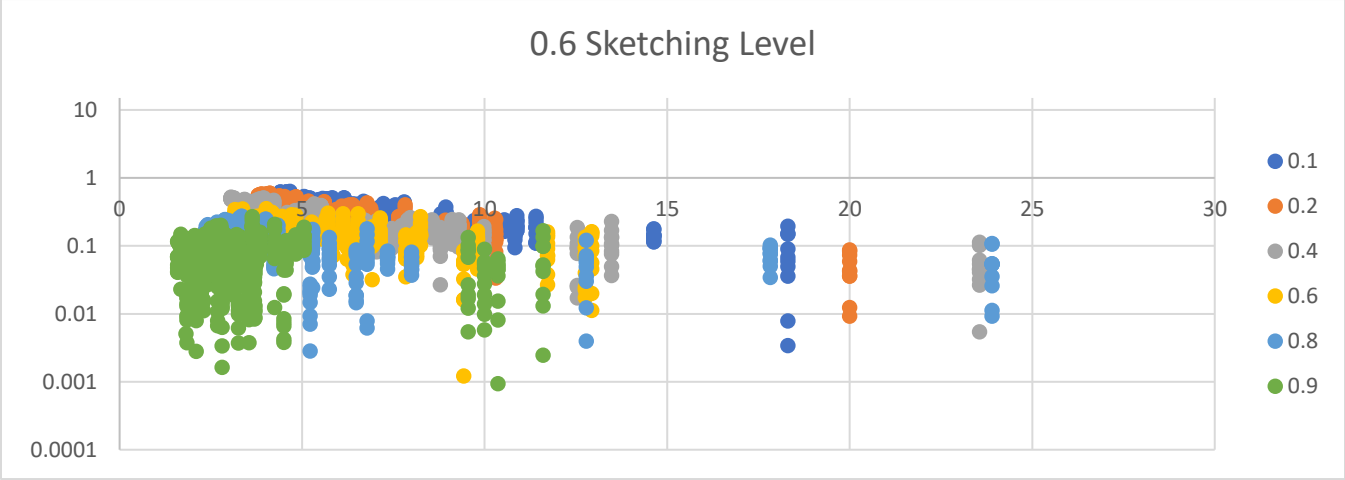

Figure 18: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. 0.6 Sketch-Size Ratio

After these tests, I performed the same sketching technique on the neural network Hessian. The original dimension of the Hessian was 272,474, and I ended up making one sketch size of size 50 and another of size 100. These correspond to sketch size ratios of around 0.02% and 0.04%, which are two orders of magnitude lower than any of the sketch sizes in my preliminary tests. The neural network Hessian was well over 99% singular. It took 18 minutes to compute the sketch of size 50 and its eigenvalues, and 35 minutes to compute the sketch of size 100 and its eigenvalues. Figure 19 below shows the error in the eigenvalue estimates for both sketch sizes.

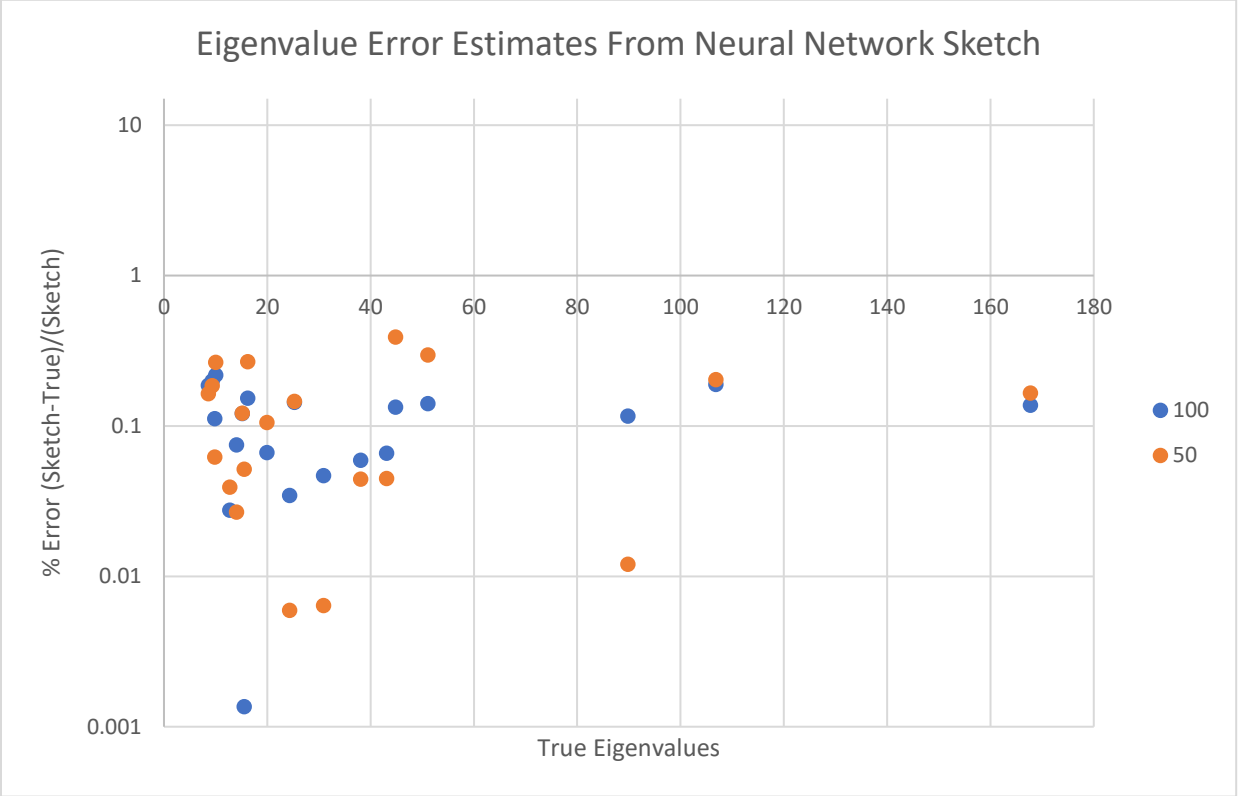

Figure 19: Error between the top 20 sketched eigenvalues and the top 20 Hessian eigenvalues. This Hessian is of a ResNet20 network trained on the Cifar-10 dataset.

The sketch of size 50 had a maximum error of 39% with a mean error of 13% and a median error of 11.3%. The sketch of size 100 had a maximum error of 21% with a mean error of 11.1% and a median error of 11.8%. The accuracy gained by doubling the sketch size ratio from 0.02% to 0.04% was minimal.

Conclusions

In conclusion, the Lanczos Method presents a big improvement over the standard Power Iteration when trying to calculate the top-k eigenvalues. Because the cost of matvec products is high and the Lanczos algorithm requires fewer matvec products than power iteration, the Lanczos Method ends up taking much less time to achieve the same accuracies.

The Hessian Sketching method I presented can be used to estimate the top-k eigenvalues of the Hessian as well as well as the trace of the Hessian. In terms of accuracy for computing the top-k eigenvalues, this sketching method works best for extremely singular Hessians like those of neural network loss functions. On a real neural network, with sketch size ratios of 0.02% and 0.04%, the top 20 eigenvalues were uniformly over-estimated by around $10-20\%$. There was little marginal benefit from increasing the sketch size ratio from 0.02% to 0.04%. In terms of accuracy for computing the Hessian trace, this method, which in essence performs Hutchinson Trace Estimation is optimally accurate because the sketching matrix is created from Rademacher vectors.

When comparing the Lanczos Method and Hessian Sketching, the Lanczos Method is more effective for simply computing the top-k eigenvalues. The Lanczos Method provides much more accurate estimates

of the eigenvalues and it can be considered to be faster. Completing 50 iterations of the Lanczos algorithm and computing a sketch of size 50 took comparable amounts of time, but it took the sketching method a little longer because a quadratic number of vector inner products had to be computed to compute the elements of $S^T HS$.

One advantage that Hessian Sketching does provide over the Lanczos Method, however, is that Hessian Sketching simultaneously produces an optimally accurate trace estimate. Computing a similar trace estimate on top of the Lanczos Method would require another set of matvec products and would end up taking much longer. For reference, PyHessian has a Hutchinson Trace Estimate function that computed a trace estimate with 50 vectors (equivalent to the trace produced by a sketch of size 50) in 14 minutes. The Lanczos Method on its own took around 15 minutes to complete 50 iterations; to compute the top 20 eigenvalues using 50 iterations of Lanczos Method and compute the trace would take a total of 29 minutes. In contrast, Hessian Sketching took 18 minutes to compute a sketch of size 50 and the eigenvalues and trace that followed.

Future Research

A primary line of future research would be to verify whether or not the methods presented in this project are robust. This could be done by testing the performance of these methods on a variety of different architectures and data sets. While some of the results on this project look promising, I only performed tests on a single architecture and a single dataset.

Another line of research would be to more deeply study the Hessian Sketching method presented here as well as study how the Hessian Sketch can be utilized and improved. The biggest weakness of the Hessian Sketching method is that it does not provide particularly accurate estimates for the top eigenvalues. If the eigenvalue estimates provided by Hessian Sketching could be used as a starting point for Rayleigh Quotient Iteration, however, the two methods in conjunction could be used to provide fast and accurate estimates for the top-k eigenvalues. Another observation to make is that increasing the sketch size from 50 to 100 had minimal effect on the accuracy of the eigenvalue estimates. Based on this, I wonder if it is possible to achieve similar accuracies with even smaller and cheaper sketches. Ideally, one could compute cheap, fast eigenvalue estimates using Hessian Sketching and then, with the cubic convergence provide by Rayleigh Quotient Iteration, achieve faster eigenvalue estimates than the Lanczos Method.

The difficulty with this idea is that it is not trivial to implement Rayleigh Quotient Iteration, or more generally Inverse Iteration, in a matrix-free manner. This is because those methods require inverses of matrices to be calculated. Further research would first need to explore how Rayleigh Quotient Iteration could be achieved in a matrix-free manner.

An extremely exciting application of this research would be to explore how fast approximations of the top eigenvalues and eigenvectors, as well as estimates of the Hessian itself could be used to improve Stochastic Gradient Descent and create second-order methods. Right now, one of the reasons that we are limited to first-order methods in neural network settings is that second-order information is difficult to access. However, with techniques that could approximate Hessian spectra, one might imagine a Quasi-Newton type method where an estimate of the Hessian is updated every few epochs and utilized in training optimization.

References

[1] Z. Yao, A. Gholami, K. Keutzer and M. Mahoney, "PyHessian: Neural Networks Through the Lens of the Hessian," 2020.

[2] A. Gholami, "PyHessian GitHub Page," [Online]. Available: https://github.com/amirgholami/pyhessian.

[3] S. Deshpande, "My PyHessian GitHub Page," [Online]. Available: https://github.com/iamsalil/PyHessian.

[4] B. Ghorbani, S. Krishnan and Y. Xiao, "An Investigation into Neural Network Optimization via Hessian Eigenvalue Density," in *ICML*, 2019.

[5] L. Sagun, B. Leon and L. Yann, "Eigenvalues of the Hessian in Deep Learning: Singularities And Beyond," in *ICLR*, 2017.

[6] Z. Yao, A. Gholami, K. Keutzer and M. Mahoney, "Hessian-based Analysis of Large Batch Training and Robustness to Adversaries," in *NeurIPS*, 2018.

[7] E. Darve and M. Wootters, Numerical Linear Algebra With Julia (Draft), TBD, TBD.

[8] A. Quarteroni, R. Sacco and F. Saleri, "Iterative Methods For Solving Linear Systems," in *Numerical Mathematics*, Springer, 2006, pp. 160-170.