Dynamics of the Hessian of the Loss during training

Abdulrahman Alabdulkareem
MIT CSAIL

Abstract

The Hessian of the Loss in Deep Learning is a topic of great interest recently. In this work, we derive an efficient analytical computation of both the maximum eigenvalue and the trace of the Hessian of the Loss in addition to an analytical and empirical convergence rate of the algorithm. The algorithm is efficiently implemented using nothing but commonly used Machine Learning techniques: the forward pass and backpropagation. We then extensively train over 150 models spanning over 10 thousand epochs with different hyperparameters and optimizers and run our algorithm after each epoch to empirically calculate properties of the hessian during training. We provide empirical results for the relationship between generalizability and the hessian, in addition to several other observations that arised from this data.

1 Introduction

The loss landscape and the Hessian of the Loss in Deep Learning is an ever growing area of research. While the field is dominated by gradient based methods, second-order derivatives provide extra insight into the landscape. This extra insight can possibly extrapolate into generalizability claims and other details about the models performance.

Calculating the second-order derivative of the loss (Hessian) is an intractable computation for any deep learning model of interest, however, the eigenvalues of the Hessian are potentially computable in a reasonable manner using numerical methods. In this work, we derive an efficient algorithm to calculate the maximum eigenvalue and the trace of the Hessian. Many recent works studied the relationship between the Hessian and different important factors of the model.

Jastrzębski et al. [2018] studied the correlation between the width of the hessian of the minima found at the end of training in relation to the learning rate and batch-size ratio. Thus, finding the empirically optimal combination of these hyperparameters. In later works, Jastrzębski et al. [2019] studied the loss landscape by analyzing the search direction of SGD during training in relation to the hessian, specifically the sharpness of the curvature of the training loss along with the step size and batch size. They suggest that such analysis of the behavior of optimizers is critical for a deeper understanding of generalization in training. They concluded that the steps taken by SGD correlate well with the direction of the sharpest curvature of the loss and that the SGD steps along this direction only up to a certain point when reaching the maximum sharpness.

Other works such as Sagun et al. [2016] looked at the hessian of the loss function at the end of training and noted several observations, such as the singularity of the hessian and adding more parameters to the model only makes the hessian more singular. They also note that the spectrum at the end of training comprises two phases, a cluster around zero that depends on the number of parameters in the model and a cluster separated away that depends on the data. One key note is that this work only considers models that are comprised of MLPs with two hidden layers trained on only 1000 MNIST images. Chaudhari et al. [2019] reported the same clustering relationship.

While not explicitly studying the hessian, Wen et al. [2018] study the generalizability of DNNs by increasing the smoothness of the loss, which can be linked with our work of analyzing the dynamics of

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the hessian while learning. Similarly, Wang et al. [2018] theoretically shows the relationship between a model's generalization and the smoothness of the hessian (specifically, its Lipschitz constant) using a PAC-Bayes framework. Directly optimizing for loss flatness has been shown to empirically improve generalization by utilizing methods such as Sharpness-Aware Minimization (SAM) Foret et al. [2021]. Many other works Chaudhari et al. [2019], Keskar et al. [2017] similarly show that empirically, sharper minima correlate with poor generalizability.

While the previously mentioned works all tend to point towards the same direction, other works seem to call into question the causal relationship between properties of the hessian and the model's generalizability. Further literature review of this counterargument is provided in Appendix 7.8.

In the next section, we provide an analytical derivation for our algorithm (full pseudocode in appendix 7.1) as well as the convergence rates.

2 Derivation

In this section we describe our derivations for calculating the maximum eigenvalue and trace of the Hessian of the training loss with respect to the neural network parameters.

For some training data $X \in \mathbb{R}^{n \times d}$ and $Y \in \mathbb{R}^n$, and some network parameters $\theta \in \mathbb{R}^p$, the Hessian of the Loss function $\mathcal{L}(X, Y; \theta)$ is $H(\mathcal{L}, \theta) = \frac{\partial^2 \mathcal{L}(X,Y;\theta)}{\partial \theta^2}$ which is a $\mathbb{R}^{p \times p}$ matrix. Calculating the exact hessian naively requires $p^2/2$ evaluations (the $\frac{1}{2}$ term due to it's symmetric property) where each evaluation is of the form $\frac{\partial^2 \mathcal{L}(X,Y;\theta)}{\theta_i \theta_j}$ this calculation quickly becomes intractable for any neural network size of interest (large p). Thus, the calculation of the full hessian matrix is computationally infeasible and numerical techniques must be employed.

Many techniques exist to calculate the eigenvalues and trace of the hessian. Ours is an algorithm which efficiently computes both the trace and the maximum eigenvalue of a neural network by efficiently utilizing automatic differentiation (Backpropagation) provided by popular machine learning frameworks such as PyTorch Paszke et al. [2019]. Some of our algorithms computation is based on Sankar et al. [2021].

Below we describe the analytical derivation of our technique for both calculations of the trace and maximum eigenvalue, then we describe the algorithm to calculate them efficiently using automatic differentiation.

2.1 Derivation of Trace Calculation

Let $v \in \mathbb{R}^n$ be a random vector where each element is an independent standard gaussian $v_i \sim \mathcal{N}(0, 1)$ for $i \in \{1, \cdots, n\}$, then the trace of the hessian is equal to

$$Trace(H(\mathcal{L}, \theta)) = \mathbb{E}\left[v^T \frac{\partial}{\partial \theta} f(X, Y, \theta, v)\right] \tag{1}$$

where f is a scalar function defined as

$$f(X, Y, \theta, v) = v^T \frac{\partial \mathcal{L}(X, Y; \theta)}{\partial \theta} \tag{2}$$

Proof and further discussion about the significance and efficiency of this implementation in the Appendix 7.2.

Let i_{tr} be the number of random Gaussian vectors v_i we sample and Backpropagate through until we consider the trace to be well approximated. We derive a theoretical lower-bound on i_{tr} to guarantee that our empirical approximation of the trace is within $\epsilon\%$ of the actual trace (or $\pm\epsilon_{abs}$) with probability $1 - \delta$

$$i_{tr} \geq \frac{\sigma[X]}{\sqrt{\delta} * \max\{\epsilon * |\mu|, \epsilon_{abs}\}} \tag{3}$$

Proof and further discussion in Appendix 7.3

2.2 Derivation of Maximum Eigenvalue Calculation

Let $v_0 \in \mathbb{R}^n$ be a random gaussian vector, we follow the recurrent relation

$$v_{i+1} = \frac{\partial}{\partial \theta} f(X, Y, \theta, v_i) \tag{4}$$

Where $f(\cdot)$ is defined in (2). Then, the maximum eigenvalue of the hessian is equal to

$$maxeig\left(H(\mathcal{L}, \theta)\right) = \lim_{i \to \infty} v_i^T v_{i+1} \tag{5}$$

With a rate of convergence of $\left|\frac{\lambda_1(H(\mathcal{L},\theta))}{\lambda_2(H(\mathcal{L},\theta))}\right|$ where ${}_1(\cdot)$ is the maximum eigenvalue and ${}_2(\cdot)$ is the second largest eigenvalue.

Proof and further discussion in Appendix 7.4

3 Algorithm

We combine our results into an efficient algorithm (In Appendix 7.1). We ran this algorithm on a consumer grade machine with a GTX 1060 and were able to estimate both the trace and maximum eigenvalue for a 1 million parameter CNN in 30 seconds.

4 Emperical Results

For all our experiments we use CIFAR 10 for our dataset and a CNN model composed of two convolution layers with 32 channels each followed by a two hidden layer NN and ReLU as the activation function.

Aggregating all our experimental data spanning over 10 thousands epochs and 150 models with different hyperparameters and optimizers, we provide two new insights regarding the trace and maximum eigenvalue throughout training (a third insight mentioned in Appendix 7.7 for space limitations.).

4.1 Hyperparameters are fundementally different explorers

Conjecture 1 The behaviour of the loss landscape exploration during SGD is fundamentally different for different hyperparameters, even at the early stages of training.

Our first insight is that, despite having similar training loss and the validation accuracy throughout training, the trace and maximum eigenvalue are fundamentally different when varying hyperparameters such as the learning rate and batch size. This is true throughout all of training and not just as the end stages of learning.

At a first glance, the prevalent idea that regarding differences in the learning rate is that a smaller rate merely causes the optimization to move slower through the loss landscape allowing for more fine-tuning but causing slower convergence and the possibility of overfitting. However, we find that throughout all our experiments, the difference in the eigenvalues of the hessian are stark even in the very early stages of training.

Figure 1 shows the 8 models trained with different learning rates (0.01 and 0.05) as well as 4 models trained with different batch sizes (32 and 64). We can see that despite the training loss and validation curves being relatively close to each other (despite one group of models sometimes requiring more epochs to catch up to the other), the two groups have distinctively different maximum eigenvalues from as early as the first epoch of training. We decided to train multiple models with the same hyperparameters (included in the figures) to make sure such drastic differences were not merely due to different weight initializations.

While the idea that different hyperparameters cause different eigenvalues at the end of training is not new Kaur et al. [2022], Sankar et al. [2021], what is significant about our results is that this difference is achieved in the very early stages of training well before the weights are near convergence.

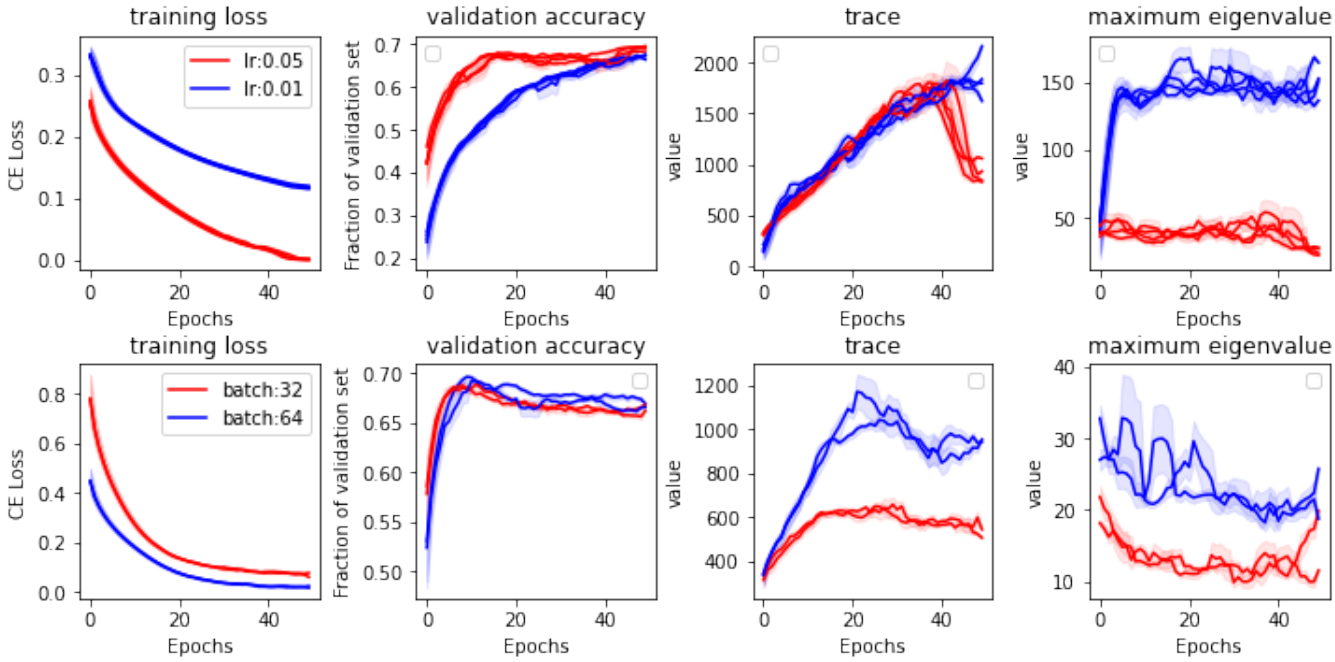

Figure 1: Top: 4 models trained with a learning rate of 0.05 (in red) and 4 models with a learning rate of 0.01 (in blue). Bottom: 2 models trained with a batch size of 32 (in red) and 2 models with a batch size of 64 (in blue). All curves are a running average with a window size of 5 epochs with error bars showing the running standard deviation

4.2 Hessian properties and Domain Shift

Conjecture 2 The dynamics of the Hessian's maximum eigenvalue give extra knowledge of how well the model performs under distribution shift.

We observe that the maximum eigenvalue gives us extra knowledge on the models out of distribution (OOD) performance. We simulate a distribution shift on our dataset by adding Gaussian noise after the typical normalization step in our processing pipeline (visualization of the OOD images in appendix 7.5).

We select 1500 epochs from our experiment results and analyze the dynamics of the test accuracy (for both in/out of distribution) as a relationship with the maximum eigenvalue. We use ordinary least squares regression analysis to analyze this dataset.

In table 1 we show the results of running four OLS Regression Models. First we focus on the models that use only the maximum eigenvalue as independent variable (first and third columns). We note that the results show the maximum eigenvalue is statistically significant in predicting the test accuracy for both in and out of distribution datasets. This is typical and what we would expect as the maximum eigenvalue is a good indicator of training which means it is typical that it would correlate with Test Accuracy.

However, to accurately analyze whether the maximum eigenvalue is a good indicator of test accuracy, we build two additional models that take into consideration the validation accuracy (second and fourth columns). The In-Distribution model (second column) shows us that when we add the validation accuracy as an independent variable in the model, the maximum eigenvalue is no longer statistically significant ($p>0.05$). This is expected as both the validation and test accuracy come from the exact same distribution, thus no extra knowledge can be attained from the eigenvalues as the validation accuracy is an unbiased estimator.

The significance of the results is from the model that predicts the Out of Distribution Test Accuracy (fourth column) that shows that when analyzing the OOD Test accuracy, the In-Distribution validation accuracy no longer provides all the info we need and that the maximum eigenvalue is a statistically

significant variable in the prediction (p<0.01). This is consistent with the intuition that as the maximum eigenvalue decreases, the loss landscape is flatter on the current point which means that for the same validation accuracy, the OOD Test Accuracy increases. In other words, the relationship indicates that flatter minima should be more robust for perturbations of the dataset (OOD).

Table 1: OLS Regression Results on the change of Test Accuracy for In/Out of distribution test set

	In-Distribution Test Accuracy		OOD Test Accuracy	
	only maxeig	maxeig+val acc	only maxeig	maxeig+val acc
$\Delta\lambda_{max}$	-0.05*** (0.004)	-0.0017† (0.001)	-0.083*** (0.006)	-0.0453*** (0.005)
Δ*validation acc%*		0.9737*** (0.006)		0.7277*** (0.033)
constant	0.82*** (0.063)	0.0149** (0.014)	0.6719*** (0.091)	0.0694† (0.084)
F-test	172***	17,410***	206***	383***
AIC	6732	2181	7841	7419
N	1470	1470	1470	1470

Note: $^{**}\ p < 0.05$; $^{***}\ p < 0.01$; $\dagger\ p > 0.05$(not significant)

5 Limitations of our work

One limitation of our work is that all our experiments were run on two types of models (CNNs and MLPs). Further work might be to reproduce these results with more diverse and larger models such as Transformers and RNNs. Another limiation of our work is our limited dataset selection (CIFAR10). Dataset diversity in this type of analysis has a great amount of room for improvement as the Hessian's relationship greatly depends on the dataset used, as pointed out in Sagun et al. [2016].

The dataset and models in our experiments were fairly limited due to the fact that we are running all experiments on a consumer grade laptop not designed for deep learning, we fully trained hundreds of models and partially trained thousands in a relatively short time span. This is positive evidence for our efficiency claim as our algorithm was efficient enough to achieve such results with very little compute, thus our methods can scale to orders of magnitude larger datasets and models given the amount of compute and resources that production-ready deep learning architectures provide.

6 Conclusion

Potential future work developing on our described results could be regarding conjecture 1, analyzing the theoretical reason for why the hyperparameters affect the eigenvalues early in the training phase which could lead to better understanding of current optimizers as the phenomena mentioned in conjecture 1 existed even when tweaking optimizer specific hyperparameters. Future work on conjecture 2 could produce optimizations that are more robust to out of distribution data and distribution shifts.

The dynamics of the hessian of the loss during training remains an open and interesting area of research. In this work, we provided an efficient algorithm to calculate both the maximum eigenvalue and the trace of the hessian using efficient techniques such as backpropagation. Then we analyzed the dynamics of the trace and maximum eigenvalue during training and posed several analysis regarding the relationship of the hyperparameters with the loss landscape and the relationship between the eigenvalues and the Out of Distribution generalizability of the models. We believe that these insights can further build into a larger narrative that can guides our understanding of deep learning models.

References

Richard H. Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, September 1995. ISSN 1064-8275. doi: 10.1137/0916069. URL https://epubs.siam.org/doi/10.1137/0916069. Publisher: Society for Industrial and Applied Mathematics.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124018, 2019.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. PMLR, 2017.

Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. Sharpness-Aware Minimization for Efficiently Improving Generalization, April 2021. URL http://arxiv.org/abs/2010.01412. arXiv:2010.01412 [cs, stat].

Stanisław Jastrzębski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three Factors Influencing Minima in SGD, September 2018. URL http://arxiv.org/abs/1711.04623. arXiv:1711.04623 [cs, stat].

Stanisław Jastrzębski, Zachary Kenton, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. On the Relation Between the Sharpest Directions of DNN Loss and the SGD Step Length, December 2019. URL http://arxiv.org/abs/1807.05031. arXiv:1807.05031 [cs, stat].

Simran Kaur, Jeremy Cohen, and Zachary C. Lipton. On the Maximum Hessian Eigenvalue and Generalization, September 2022. URL http://arxiv.org/abs/2206.10654. arXiv:2206.10654 [cs, stat].

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima, February 2017. URL http://arxiv.org/abs/1609.04836. arXiv:1609.04836 [cs, math].

Henry W Lin, Max Tegmark, and David Rolnick. Why does deep and cheap learning work so well? *Journal of Statistical Physics*, 168(6):1223–1247, 2017.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. 2016.

Adepu Ravi Sankar, Yash Khasbage, Rahul Vigneswaran, and Vineeth N Balasubramanian. A deeper look at the hessian eigenspectrum of deep neural networks and its applications to regularization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9481–9488, 2021.

Grzegorz Swirszcz, Wojciech Marian Czarnecki, and Razvan Pascanu. Local minima in training of neural networks, February 2017. URL http://arxiv.org/abs/1611.06310. arXiv:1611.06310 [cs, stat].

Huan Wang, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Identifying generalization properties in neural networks. 2018.

Wei Wen, Yandan Wang, Feng Yan, Cong Xu, Chunpeng Wu, Yiran Chen, and Hai Li. SmoothOut: Smoothing Out Sharp Minima to Improve Generalization in Deep Learning, December 2018. URL http://arxiv.org/abs/1805.07898. arXiv:1805.07898 [cs, stat].

7 Appendix

7.1 Full Algorithm

In Algorithm 1 we provide the full algorithm from our derived calculations. The significance of our algorithm is that it fully utilizes Backpropagation in its calculation and doesn't require any additional optimizers or numerical methods.

Algorithm 1 Efficient computation of the Trace and Maximum Eigenvalue of the Hessian

procedure LOSSGRADIENT($\mathcal{L}, X, Y, \theta, frac$) ▷ Returns $\frac{\partial \mathcal{L}(X,Y;\theta)}{\partial \theta}$ on fraction of data
    size ← frac * len(X)
    ind ← random.sample(len(X), size, seed=42) ▷ randomly pick subset
    $X_{frac} \leftarrow X[ind]$
    $Y_{frac} \leftarrow Y[ind]$
    loss ← $\mathcal{L}(X_{frac}, Y_{frac}, \theta)$
    return Jacobian(loss, θ) ▷ Apply Backpropagation

procedure DFDθ($\mathcal{L}, X, Y, \theta, frac, v$) ▷ Returns $\frac{\partial f}{\partial \theta}$ as defined (11)
    grad ← $LossGradient(\mathcal{L}, X, Y, \theta, frac)$
    f ← $v^T *$ grad ▷ Dot-product between the gradient of the Loss and v
    return Jacobian(f, θ) ▷ Apply Backpropagation

procedure TRACE($\mathcal{L}, X, Y, \theta, r_{tr}, i_{tr}$) ▷ Returns the approximate of the Trace (10)
    dim ← $len(\theta)$
    running ← [ ]
    for `i = 1 to` i_{tr} **do**
        $v \sim \mathcal{N}(0, 1, dim)$
        dfdt ← DFDθ($\mathcal{L}, X, Y, \theta, r_{tr}, v$)
        tr ← $v^T *$ dfdt ▷ $v^T Av$ in (7)
        running.append(tr)
    return $mean($ running $)$ ▷ Takes the expectation over v in $v^T Av$

procedure MAXEIG($\mathcal{L}, X, Y, \theta, r_{eig}, i_{eig}$) ▷ Returns the approximate of the Maximum eigenvalue (10)
    dim ← $len(\theta)$
    best_approx ← 0
    $v \sim \mathcal{N}(0, 1, dim)$
    for `i = 1 to` i_{eig} **do**
        $v \leftarrow v/norm(v)$ ▷ Normalize, L2 norm
        dfdt ← DFDθ($\mathcal{L}, X, Y, \theta, r_{eig}, v$)
        best_approx ← $v^T *$ dfdt
        v ← dfdt ▷ calculate Rayleigh Quotient from (18)
    return best_approx

7.2 Trace derivation

Many techniques exist to calculate the eigenvalues and trace of the hessian. Keskar et al. [2017] analyze the magnitude of eigenvalues of the Hessian, which they refer to as the sharpness of a minimizer, by considering p vectors ($p = 100$) which define a manifold that then creates an ϵ box around the function f which they then maximize the function within. Once this stochastic procedure is complete, a final transformation on the obtained maximum gives the sharpness of f. Such a maximization problem requires the use of L-BFGS to solve Byrd et al. [1995].

Our technique, which is a modified algorithm from Sankar et al. [2021], efficiently computes both the trace and the maximum eigenvalue of a neural network by efficiently utilizing automatic differentiation (Backpropagation).

We start with an important numerical result that approximates the trace of a matrix by a series of Matrix-Vector multiplications. Let $A \in \mathbb{R}^{n \times n}$ be a real symmetric matrix we want to approximate the trace of, and let $v \in \mathbb{R}^n$ be a random vector where each element is an independent standard gaussian $v_i \sim \mathcal{N}(0, 1)$ for $i \in \{1, \cdots, n\}$ such that

$$\mathbb{E}[v] = 0 \qquad \mathbb{E}[vv^T] = I \tag{6}$$

Then we have that

$$\mathbb{E}[v^T A v] = \mathbb{E}[\sum_{i=1}^{n} \lambda_i] = Trace(A) \tag{7}$$

Which is proven by Lin et al. [2016].

In our case the matrix A is the hessian which is represented as the second derivative of a scalar $A = H(\mathcal{L}, \theta) = \frac{\partial^2 \mathcal{L}(X,Y;\theta)}{\partial \theta^2}$ thus we can rewrite the expectation as the following term

$$Trace(H(\mathcal{L}, \theta)) = \mathbb{E}\left[v^T \frac{\partial^2 \mathcal{L}(X, Y; \theta)}{\partial \theta^2} v\right] \tag{8}$$

The above equation is still intractable due to the hessian calculation in the middle, however, we can use a simple property from Matrix Calculus that states that $\frac{\partial a(x)}{\partial x} b = \frac{\partial b^T a(x)}{\partial x}$ where $a(x)$ is a vector that depends on the vector x, and b is a vector not dependant on x. Using this property on (7) we get

$$Trace(H(\mathcal{L}, \theta)) = \mathbb{E}\left[v^T \frac{\partial^2 \mathcal{L}(X, Y; \theta)}{\partial \theta^2} v\right] = \mathbb{E}\left[v^T \left(\frac{\partial}{\partial \theta} \frac{\partial \mathcal{L}(X, Y; \theta)}{\partial \theta}\right) v\right] \tag{9}$$

$$= \mathbb{E}\left[v^T \frac{\partial}{\partial \theta} \left(v^T \frac{\partial \mathcal{L}(X, Y; \theta)}{\partial \theta}\right)\right] = \mathbb{E}\left[v^T \frac{\partial}{\partial \theta} f(X, Y, \theta, v)\right] \tag{10}$$

where f is a scalar function defined as

$$f(X, Y, \theta, v) = v^T \frac{\partial \mathcal{L}(X, Y; \theta)}{\partial \theta} \tag{11}$$

The significance of equation (10) is that we split the intractable computation of the Hessian into two parts that, we can see, are both extremely efficient to calculate. The first part is the scalar function f only requiring the derivative of a scalar Loss ($\mathcal{L}(\cdot)$) with respect to a vector (θ) which is very efficiently computed by Automatic Differentiation using Backpropagation, followed by a dot-product $v^T \frac{\partial \mathcal{L}(\cdot)}{\partial \theta}$. The second part of the computation is similarly a derivative of a scalar ($f(\cdot)$) with respect to a vector (θ) which is equivalently calculated through Backpropagation, followed by a simple dot-product $v^T \frac{\partial f(\cdot)}{\partial \theta}$. Thus, the term inside the expectation only needs two backward propagation.

Two important hyperparameters arise from this procedure, firstly is i_{tr} which is the number of random Gaussian vectors v_i we need to sample and Backpropagate through until we consider the trace to be well approximated, and secondly is r_{tr} which is the fraction of the original size data that we use to Backpropagate through since we can empirically see that the trace of the loss from a small fraction of the data quickly converges to that of the entire dataset. For all our experiments, we set $i_{tr} = 10$ and $r_{tr} = 0.1$.

7.3 Trace error estimation

For the trace calculation in (10), each iteration we are drawing one sample from a distribution that has mean equal to the trace we want. Thus we can do a simple analysis using Chebyshev's inequality

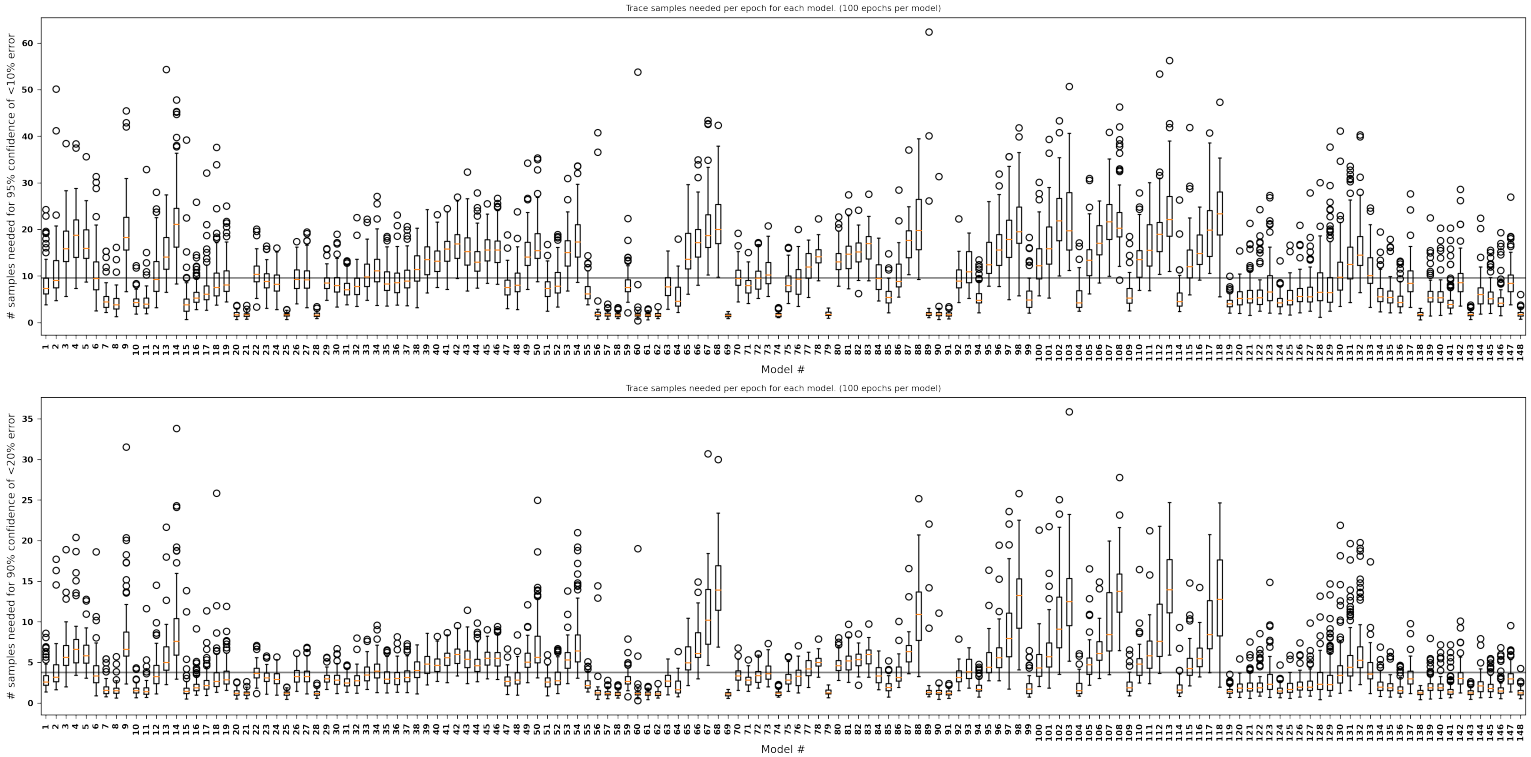

Figure 2: The number of samples needed (i_{tr}) to empirically calculate the trace up to a tolerate of 10% (or ± 1) with 95% confidence. Each box plot represents a model trained with a different optimizer (SGD, Adam, AdaGrad, etc) and hyperparameters (lr, batch size, weight decay, etc) and was trained for 100 epochs. Each boxplot is taken over the points (1 per epoch) which represents the value of n as calculated in (14). Note that individual points are shown only when lying outside of the boxplot whiskers which cover $1.5 * IQR$, such points represent outliers. The horizontal line represents the global mean over all epochs and models. The top figure is calculated with values $\epsilon = 10\%,\ \epsilon_{abs} = 1,$ and $1 - \delta = 95\%$ and gives a global mean of 9.64. The bottom figure represent relaxed conditions $\epsilon = 20\%,\ \epsilon_{abs} = 1,$ and $1 - \delta = 90\%$ with a global mean of 3.78

to estimate how far we are from our goal depending on how many samples we draw.

$$P(|X - \mu| \geq k\sigma) \leq \frac{1}{k^2} \tag{12}$$

$$P(|\bar{X} - \mu| \geq k) \leq \frac{Var[\bar{X}]}{k^2} \tag{13}$$

Where $\bar{X}$ is the mean of i_{tr} samples from the distribution and is our approximation of the trace while μ is the true value of the trace. For our choice of k, we intuitively want our approximation of the trace to be at most ϵ percent away from the true trace with probability $1 - \delta$. We optionally handle the case when μ is (approximately) zero by considering an absolute error on the estimation. Thus, we get a bound on how many samples we have to draw

$$P\left(\left(|\bar{X} - \mu| \geq \epsilon_{abs}\right) \cap \left(|\bar{X} - \mu| \geq \epsilon * |\mu|\right)\right) \tag{14}$$

$$= P(|\bar{X} - \mu| \geq \max\left\{\epsilon * |\mu|, \epsilon_{abs}\right\}) \leq \frac{Var[X]}{i_{tr}^2 * \max\left\{\epsilon^2 * \mu^2, \epsilon_{abs}^2\right\}} \leq \delta \tag{15}$$

$$i_{tr} \geq \frac{\sigma[X]}{\sqrt{\delta} * \max\left\{\epsilon * |\mu|, \epsilon_{abs}\right\}} \tag{16}$$

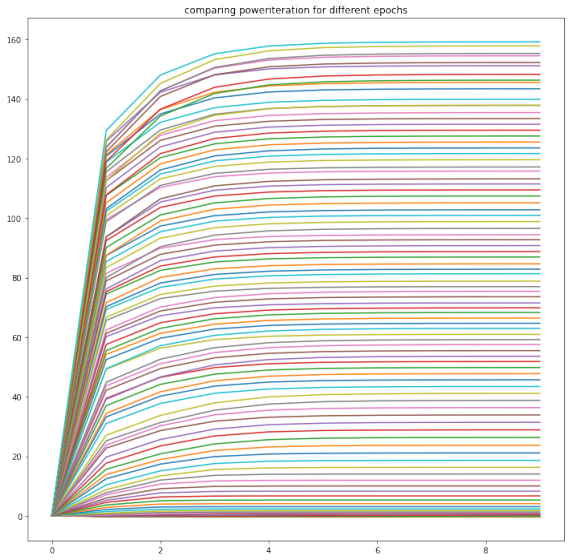

Figure 3: Convergence of our calculation of the maximum eigenvalue for different epochs of a random model. The convergence is much sooner than 10 iterations for all different invocations.

We decided to choose $\epsilon = 10\%$, $\epsilon_{abs} = 1$ and $\delta = 5\%$, which means that we want our approximation to be within 10% of the actual trace (or ± 1) with at least 95% confidence. We also pick another set of values $\epsilon = 20\%,\ \epsilon_{abs} = 1,$ and $1 - \delta = 90\%$ representing a set of more relaxed conditions.

Analytically calculating or bounding the variance of the random variable calculated in (10) is a very challenging task as the hessian is a function of both the dataset and the model architecture, weight, and indirectly, the optimizer. This analytical calculation of the variance is beyond the scope of this paper. Additionally, μ is the true value of the trace which we obviously don't have. However, we can empirically calculate the mean and variance of $v^T Av$ by first fixing the dataset to CIFAR10 and our model architecture to the aforementioned CNN. Then, we aggregate our experiments over 10 thousands epochs spanning 150 models with different hyperparameters and optimizers. Each epoch we sample (10) several times (specifically 10 times) and use that sample to get an empirical mean and standard deviation of the stochasticity of (10) to get an empirical value for n using (14).

In figure 2 we calculate the needed i_{tr} by using empirical estimate of (14), the two plots represent the two set of conditions with the top representing the i_{tr} needed to obtain a more accurate value of the trace. We can see that for the relaxed conditions, the average required $i_{tr} \geq 4$ (represented by the horizontal line). While for the stricter conditions $i_{tr} \geq 10$

Note that the samples of $v^T Av$ from one epoch estimate the trace of the hessian at that specific epoch, moving to the next epoch updates the weights which changes the hessian. Thus, for each epoch, our empirical estimate of i_{tr} (a single point in 2) comes from random variable (estimates of (14)) that has it's own mean and standard deviation that has some interesting patterns but was left out due to brevity.

Our key takeaway is that $i_{tr} = 10$ is sufficient

7.4 Derivation of Maximum Eigenvalue Calculation

To calculate the maximum eigenvalue of the Hessian we employ the simple numerical eigenvalue algorithm, Power Iteration.

To get the maximum eigenvalue (assuming uniqueness of the max) of a matrix $A \in \mathbb{R}^{n \times n}$. We start from a random vector $v_0 \in \mathbb{R}^n$, we follow the recurrent relation

$$v_i = \frac{Av_{i-1}}{\|Av_{i-1}\|} \quad i \in \{1, 2, \cdots\} \tag{17}$$

The maximum eigenvalue is approximated by the Rayleigh quotient of v_i for sufficiently large i.

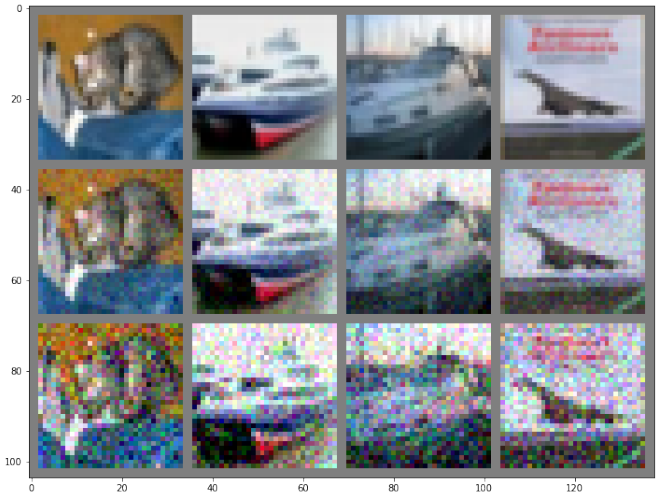

Figure 4: 4 random samples from CIFAR10. Top: original samples. Middle: Gaussian noise (0.1). Bottom: Gaussian noise (0.25). The bottom row of images is what we consider as OOD samples, as even to humans such samples are much harder to classify than the noiseless in-distribution samples

$$\rho(A) = \lim_{i \to \infty} \frac{v_i^T v_{i+1}}{v_i^T v_i} = \lim_{i \to \infty} \frac{v_i^T v_{i+1}}{\|v_i\|} = \lim_{i \to \infty} \frac{v_i^T v_{i+1}}{\|\frac{Av_{i-1}}{\|Av_{i-1}\|}\|} = \lim_{i \to \infty} v_i^T v_{i+1} \tag{18}$$

The rate of convergence for power iteration can be easily derived as $\left|\frac{\lambda_1(H(\mathcal{L},\theta))}{\lambda_2(H(\mathcal{L},\theta))}\right|$ where $_1(\cdot)$ is the maximum eigenvalue and $_2(\cdot)$ is the second largest eigenvalue.

Then we derive an efficient implementation of v_i by noting that we only need to define the dot-product term Av which for the hessian becomes $H(\mathcal{L},\theta)v = \frac{\partial^2 \mathcal{L}(X,Y;\theta)}{\partial \theta^2} v$ which as we derived in 10 is simply the term $\frac{\partial}{\partial \theta} f(X, Y, \theta, v)$ which is efficiently computed using Backpropagation. We are left with similar hyperparameter choices as with the trace. i_{eig} representing how many power-iterations to apply, and r_{eig} for the size of the fraction of the data. for our experiemnts we set $i_{eig} = i_{tr} = 10$ and $r_{eig} = r_{tr} = 0.1$

In figure 3 we see that for 100 different invocations of our maximum eigenvalue calculation, the algorithm converges much sooner than the 10 steps we take, thus we can safely assume from this empirical result that 10 steps of our algorithm will yield a fairly good approximation of the maximum eigenvalue.

7.5 OOD noise

We simulate a distribution shift in our dataset by adding Gaussian noise with 0 mean and a standard deviation of 0.25. Figure 4 gives a visualization of the level of noise that is added to simulate a distribution shift. Even to a human the samples start to become much harder to identify the label they belong to.

7.6 OOD Test Accuracy OLS

In figure 5 we visualize the linear relationship between the change in the maximum eigenvalue and the change in the OOD Test Accuracy. We can see that for most epochs where the maximum eigenvalue decreases (flatter loss landscape) the OOD Test Accuracy increases. This relationship makes intuitive sense as flatter minima should be more robust for perturbations of the dataset (OOD).

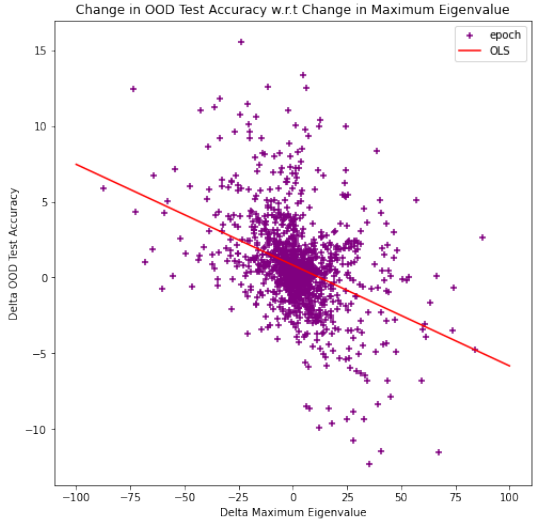

Figure 5: Ordinary least squares regression line where the indepenendent variable is the per-epoch change in the maximum eigenvalue of the hessian (x-axis) and the dependent variable is the per-epoch change in the Out of Distribution Test Accuracy. The negative relationship is as expected indicating flatter loss landscapes generalize to slightly out of distribution samples.

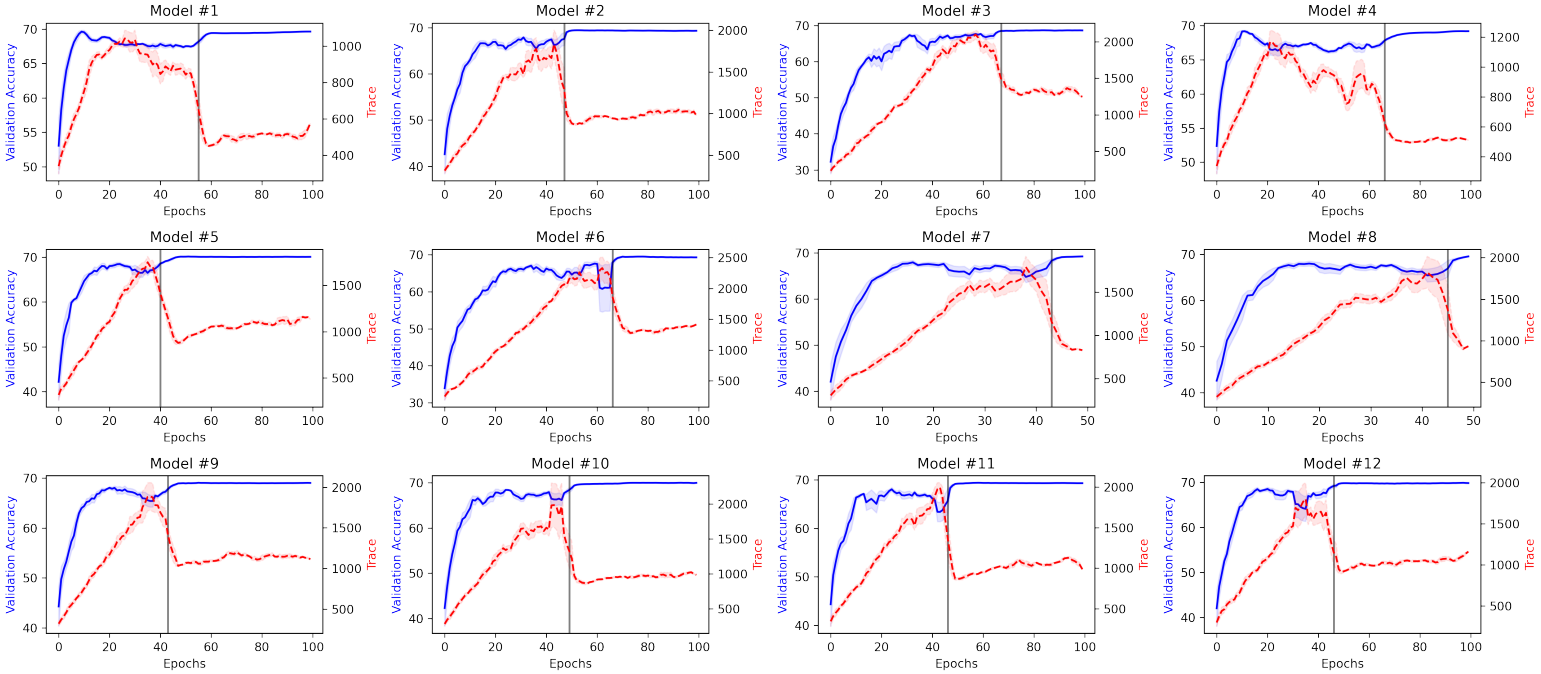

Figure 6: Validation accuracy (blue line) and Trace (dashed red line) for 12 models across training. For readability, a vertical line is placed at the epoch where the trace is mid-drop. For all models the same vertical line intersects the roughly the same epoch where the validation accuracy reaches its peak and plateaus.

7.7 Sudden sharp drop phenomena

Conjecture 3 The sudden sharp decline in the trace of the hessian during training signifies that training has stopped.

We have noticed that in many of our trained models there is a clear and distinct sharp decline in the value of the trace that we refer to as the sharp drop phenomena. This phenomena is seemingly directly linked with the stabilization in the validation and test accuracy. We conjecture that this phenomena is linked with a stop in learning.

In figure 6 we provide 12 models that exhibit this phenomena during training. We superimpose in one plot the value of the trace and the validation accuracy across the training epochs. The vertical line is placed to visually illustrate the epoch in which the trace is in the middle of the drop phenomena. For all models, this same line approximately intersects the spot in which the validation peaks and plateaus.

This similar phenomena was alluded to or shown without an explicit discussion in previous works Jastrzębski et al. [2019], **?**, 2018]

One potential question for future work is the causality of this relationship, could we possibly delay this dropping phenomena and cause the validation accuracy increasing? Possible future work could also expand on this phenomena and possibly provide improvements to optimizers using this knowledge.

7.8 Counterargument to the Hessian

While many past works tend to point towards the same link between the hessian and the model's performance, other works seem to call into question the causal relationship between properties of the hessian and the model's generalizability.

Dinh et al. [2017] argue that, theoretically, popular definitions of the hessian's flatness are somewhat flawed and are separate from the model's generalizability when considering the broad class of deep neural networks with ReLU activations. In their third theorem, they analytically show that for any non-zero local optima in the loss landscape, a transformation on the weights can generate an equivalent model with an arbitrarily large spectrum by carefully manipulating the geometry of the function. Additionally, their fifth theorem shows that there exists transformation to give an equivalent model that contain (r) arbitrarily large eigenvalues (where r can be a large number even for thin and deep NNs). Thus, the hessian would be sharp in multiple dimensions. This finding is in contrast to Sagun et al. [2016] conjecture that over-parameterized models lead to hessian deficiency. Dinh et al. [2017] concludes that using different definitions of flatness, generalizability and flatness are not theoretically linked unless further assumptions are made.

Other works, such as Swirszcz et al. [2017], consider the characteristics of the loss landscape during training for specific cases and datasets when learning fails. They highlight the importance of building theoretical convergence theorems that handle different scenarios for the structure of the data instead of divorcing the theory from the specifics of the data. These highlights are further supported by Lin et al. [2017].

In a similar vein, Kaur et al. [2022] provide multiple findings that further question the relation between the hessian and generalization. They show that there are several regimes where the maximum eigenvalue of the hessian can be reduced while the model's generalization only improves slightly or none at all (such as when using large batch sizes). They also show that some techniques that are detrimental to learning, such as an excessively large dropout rate, lead to poor generalizability while still lowering the maximum eigenvalue. While other techniques, such as batch normalization, lead to better generalization while not always lowering the maximum eigenvalue.