
Deep Curvature Suite

Diego Granzio*
Oxford University

Xingchen Wan*
Oxford University

Timur Garipov*
Samsung AI Moscow

Abstract

Despite providing rich information into neural networks geometry and applications in Bayesian neural networks and second order optimisation, accessing curvature information is still a daunting engineering challenge and hence inaccessible to most practitioners. In some cases, proxy diagonal approximations, which we show on both real and synthetic examples can be arbitrarily bad. We hence provide an open-source software package to the community, to enable easy access to curvature information for real networks and datasets, not just toy examples and is typically an order of magnitude faster than competing packages. We address and disprove many common misconceptions in the literature, namely that the Lanczos algorithm learns eigenvalues from the top down. We also prove using high dimensional concentration inequalities that for specific classes of matrices a single random vector is sufficient for accurate spectral estimation. We showcase our package practical utility on a series of examples based on realistic modern neural networks tested on CIFAR-10/100 datasets.

such as TensorFlow (Abadi et al., 2016) and PyTorch (Paszke et al., 2017) have become widely adopted, with higher level packages allowing users to state their model, dataset and optimiser in a few lines of code (Chollet, 2015), effortlessly achieving state of the art performance.

However, the pace of development of software packages extracting second order information, representing the curvature at a point in weight space, has not kept abreast. Researchers aspiring to evaluate or use curvature information need to implement their own libraries, which are rarely shared or kept up to date. Naive implementations are computationally intractable for all but the smallest of models. Hence, researchers typically either completely ignore curvature information or use highly optimistic approximations, such as the diagonal elements of the matrix or of a surrogate matrix, with limited justification or empirical analysis of the harshness of the aforementioned approximations.

Although the combination of fast Hessian vector products (Pearlmutter, 1994), advanced linear algebraic techniques (Golub & Meurant, 1994) and high dimensional geometry (Hutchinson, 1990) holds the key to solving to making significant process in this space, the lack of focus given to these areas, means that implementations of these methods (to the extent that they exist at all), are either inefficient or incorrect. In this paper we

1 Introduction

The success of deep learning models trained with gradient based optimizers on a range of tasks, from image classification/segmentation, natural language processing to reinforcement learning, often beating human level performance, has led to an explosion in the availability and ease of use of high performance software for their implementation. Automatic differentiation packages

- Motivate the use of curvature information, for both optimization, generalization, understanding the quality of optima, the effect of normalization techniques and evaluating theoretical assumptions
- Provide a primer on iterative methods, particularly the combination of the Lanczos algorithm with random seed vectors, typical pitfalls and errors in its implementation and interpretation. We highlight its link to orthogonal polynomials, the problem of

*equal contribution

moments, high dimensional concentration theorems and how it can be used to generate a highly representative spectral approximation with minimal computational or memory overhead

- Provide a primer on typical matrix assumptions used for evaluating spectra in deep learning, such as the diagonal or diagonal generalised Gauss-Newton approximation and evaluate their efficacy on both small random matrices corresponding to known eigenvalue distributions and deep neural networks
- We provide an open-source 2nd-order PyTorch based software package, the **Deep Curvature** suite¹, which allows for spectral visualisation of the Hessian and Generalised Gauss Newton, loss surface eigen-traversal, gradient, hessian and loss variance calculation on large expressive modern networks. We present examples of its usage on VGG networks (Simonyan & Zisserman, 2014) and ResNets (He et al., 2016) in a matter minutes on a single GPU. We further provide an implementation of iterative stochastic Newton methods for deep learning algorithms.

We use the GPytorch implementation (Gardner et al., 2018) of the *Lanczos algorithm* (Meurant & Strakoš, 2006), which we introduce in Section 3 and discuss the most common misconceptions in the literature in Section 4. We also note that the GPyTorch implementation (Gardner et al., 2018) has been used in a similar way to efficiently compute Hessian eigenspectra in Izmailov et al. (2019) and Maddox et al. (2019).

2 The Importance of Curvature in Deep Learning

The curvature at a point in weight-space informs us about the local conditioning of the problem, which determines the rate of convergence for first order methods and informs us about the optimal learning and momentum rates (Nesterov, 2013). The most common areas where curvature information is employed are analyses of the **Loss Surface** and **Newton** type methods in optimization.

2.1 Loss Surfaces

Loss surface visualization of deep neural networks, have often focused on two dimensional slices of random vectors (Li et al., 2017) or the changes in the loss traversing a set of random vectors drawn from the d -dimensional Gaussian distribution (Izmailov et al., 2018). Recent

¹Available at https://github.com/xingchenwan/MLRG_DeepCurvature

empirical analyses of the neural network loss surfaces invoking full eigen-decomposition (Sagun et al., 2016, 2017) have been limited to toy examples with < 5000 parameters. Other works have used the diagonal of the Fisher information matrix (Chaudhari et al., 2016), an assumption we will challenge in this paper. Theoretical analysis relating the loss surface to spin-glass models from condensed matter physics and random matrix theory (Choromanska et al., 2015b,a) rely on a number of unrealistic assumptions². Given that the spectra of many classes of random matrices are known (Tao, 2012; Akemann et al., 2011), it may be helpful to visualise the spectra of large real networks and commonly used datasets to evaluate whether they match theoretical predictions. Other important areas of loss surface investigation include understanding the effectiveness of batch normalization (Ioffe & Szegedy, 2015). Recent convergence proofs (Santurkar et al., 2018) bound the maximal eigenvalue of the Hessian with respect to the activations and bounds with respect to the weights on a per layer basis. Bounds on a per layer basis do not imply anything about the bounds of the entire Hessian and furthermore it has been argued that the full spectrum must be calculated to give insights on the alteration of the landscape (Kohler et al., 2018). Recent work making curvature information more available, again through diagonal approximations, explicitly disallows the use of batch normalization (Dangel et al., 2019). Our software package extends seamlessly to batch normalization, allowing for evaluation in both train and eval mode.

2.2 Newton Methods in Deep Learning

All second order methods solve the minimisation problem for the loss, L associated with parameters \mathbf{p} and perturbation \mathbf{d} to the second order in Taylor expansion,

$$\mathbf{d}^* = \operatorname{argmin}_{\mathbf{d}} L(\mathbf{p} + \mathbf{d})$$

$$L(\mathbf{p} + \mathbf{d}) = L(\mathbf{p}) + \nabla L^T \mathbf{d} + \frac{1}{2} \mathbf{d}^T \bar{\mathbf{H}} \mathbf{d} \quad (1)$$

Where instead of the true Hessian $\mathbf{H} = \nabla \nabla L(\mathbf{p}) \in \mathbb{R}^{n \times n}$, a surrogate positive definite approximation to the Hessian $\bar{\mathbf{H}}$, such as the Gauss-Newton, is employed so to make sure the minimum is lower bounded; its solution is

$$\mathbf{d} = -\bar{\mathbf{H}}^{-1} \nabla L(\mathbf{p}) = - \sum_i^N \frac{1}{\lambda_i} \mathbf{u}_i \mathbf{u}_i^T \nabla L(\mathbf{p}) \quad (2)$$

where \mathbf{u}_i correspond to the generalised Hessian eigenvectors. The parameters are updated with $\mathbf{p} = \mathbf{p} - \alpha \mathbf{d}$, in which α is the global learning rate.

²Such as input independence.

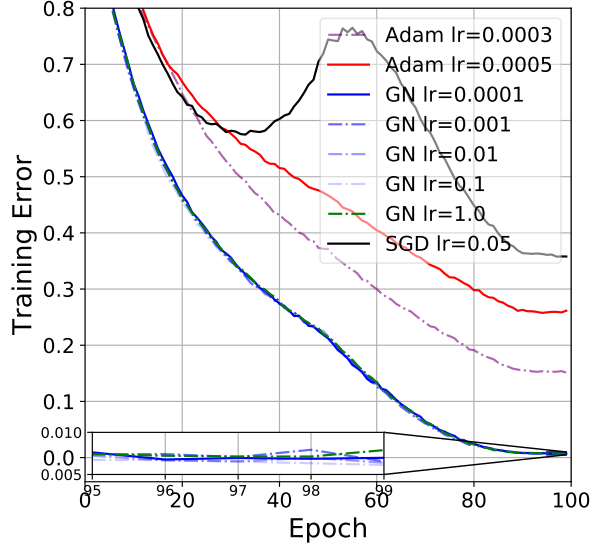


Figure 1: Training Error of stochastic Lanczos Newton methods on CIFAR-100 VGG-16 against baselines.

Despite the success of second order optimisation for difficult problems on which SGD is known to stall, such as recurrent neural networks (Martens & Sutskever, 2012), or auto-encoders (Martens, 2016). Researchers wanting to implement second order methods such as (Vinyals & Povey, 2012; Martens & Sutskever, 2012; Dauphin et al., 2014) face the aforementioned problems of difficult implementation. As a minor contribution, we also include two stochastic Lanczos based optimisers in our code. We plot the training error of the VGG-16 network on CIFAR-100 dataset against epoch in Figure 1. We keep the ratio of damping constant to learning rate constant, where $\delta = 10\alpha$, for a variety of learning rates in $\{1, 0.1, 0.01, 0.001, 0.0001\}$ with a batch size of 128 for both the gradient and the curvature, all of which post almost identical performance. We also compare against different learning rates of Adam, both of which converge significantly slower per iteration compared to our stochastic Newton methods, and we in black plot SGD, with a typical learning rate of 0.05 and momentum 0.9, which has an unstable optimisation trajectory.

Bayesian Neural Networks As a minor remark, we note that Bayesian neural networks use the Laplace approximation, featuring the inverse of the Hessian multiplied by a vector (Bishop, 2006). Our code allows for an estimation of this quantity, which may also be of use to the community and serve as an alternative for KFAC-Laplace (Ritter et al., 2018).

3 Learning to love Lanczos

The Lanczos Algorithm (Algorithm 1) is an iterative algorithm for learning a subset of the eigenvalues/eigenvectors of any Hermitian matrix, requiring only matrix vector products. It can be regarded as a far superior adaptation of the power iteration method, where the Krylov subspace $\mathcal{K}(\mathbf{H}, \mathbf{v}) = \text{span}\{\mathbf{v}, \mathbf{H}\mathbf{v}, \mathbf{H}^2\mathbf{v}, \dots\}$ is orthogonalised using Gram-Schmidt. Beyond having improved convergence to the power iteration method (Bai et al., 1996) by storing the intermediate orthogonal vectors in the corresponding Krylov subspace, Lanczos produces estimates of the eigenvectors and eigenvalues of smaller absolute magnitude, known as Ritz vectors/values. Despite its known superiority to the power iteration method and relationship to orthogonal polynomials and hence when combined with random vectors the ability to estimate the entire spectrum of a matrix, these properties are often ignored or forgotten by practitioners, we hence include a full primer in Appendix B. We also explicitly debunk some key persistent myths in the next section.

4 Common Misconceptions

- We can learn the negative and interior eigenvalues by shifting and inverting the matrix sign $\mathbf{H} \rightarrow -\mathbf{H} + \mu\mathbf{I}$
- Lanczos learns the largest m $[\lambda_i, \mathbf{u}_i]$ pairs of $\mathbf{H} \in \mathbb{R}^{P \times P}$ with high probability (Dauphin et al., 2014)

Since these two related beliefs are prevalent, we disprove them explicitly in this section, with Theorems 1 and 2.

Theorem 1. *The shift and invert procedure $\mathbf{H} \rightarrow -\mathbf{H} + \mu\mathbf{I}$, changes the Eigenvalues of the Tri-diagonal matrix \mathbf{T} (and hence the Ritz values) to $\lambda_i = -\lambda_i + \mu$*

Proof. Following the equations from Algorithm 1

$$\begin{aligned}
 \mathbf{w}_1^T &= (-\mathbf{H} + \mu\mathbf{I})\mathbf{v}_1 \ \& \ \alpha_1 = \mathbf{v}_1^T \mathbf{H} \mathbf{v}_1 + \mu\mathbf{I} \\
 \mathbf{w}_2 &= \mathbf{w}_1 - \alpha_1 \mathbf{v}_1 = (\mathbf{H} + \mu\mathbf{I})\mathbf{v}_1 - (\mathbf{v}_1^T \mathbf{H} \mathbf{v}_1 + \mu\mathbf{I})\mathbf{v}_1 \\
 \mathbf{w}_2 &= (\mathbf{H} - \mathbf{v}_1^T \mathbf{H} \mathbf{v}_1)\mathbf{v}_1 \ \& \ \mathbf{v}_2 = \mathbf{w}_2 / \|\mathbf{w}_2\| \\
 \alpha_2 &= \mathbf{v}_2^T (-\mathbf{H} + \mu\mathbf{I})\mathbf{v}_2 = -\mathbf{v}_2^T \mathbf{H} \mathbf{v}_2 + \mu \\
 \beta_2 &= \|\mathbf{w}_2\|
 \end{aligned}
 \tag{3}$$

Assuming this for $m - 1$, and repeating the above steps for m we prove by induction and finally arrive at the modified tridiagonal Lanczos matrix $\tilde{\mathbf{T}}$

$$\begin{aligned}
 \tilde{\mathbf{T}} &= -\mathbf{T} + \mu\mathbf{I} \\
 \tilde{\lambda}_i &= -\lambda_i + \mu \ \forall 1 \leq i \leq m
 \end{aligned}
 \tag{4}$$

□

Remark. No new Eigenvalues of the matrix \mathbf{H} are learned. Although it is clear that the addition of the identity does not change the Krylov subspace, such procedures are commonplace in code pertaining to papers attempting to find the *smallest eigenvalue*. This disproves the first misconception.

Theorem 2. For any matrix $\mathbf{H} \in \mathbb{R}^{P \times P}$ such that $\lambda_1 > \lambda_2 > \dots > \lambda_P$ and $\sum_{i=1}^m \lambda_i < \sum_{i=m+1}^P \lambda_i$ in expectation over the set of random vectors \mathbf{v} the m eigenvalues of the Lanczos Tridiagonal matrix \mathbf{T} do not correspond to the top m eigenvalues of \mathbf{H}

Proof. Let us consider the matrix $\tilde{\mathbf{H}} = \mathbf{H} - \frac{\lambda_{m+1} + \lambda_m}{2} \mathbf{I}$,

$$\begin{cases} \lambda_i > 0, & \forall i \leq m \\ \lambda_i < 0, & \forall i > m \end{cases} \quad (5)$$

Under the assumptions of the theorem, $\text{Tr}(\tilde{\mathbf{H}}) < 0$ and hence by Theorem 8 and Equation 23 there exist no $w_i > 0$ such that

$$\sum_{i=1}^m w_i \lambda_i^k = \frac{1}{P} \sum_{i=1}^P \lambda_i^k \quad 1 \leq k \leq m \quad (6)$$

is satisfied for $k = 1$, as the LHS is manifestly positive and the RHS is negative. By Theorem 1 this holds for the original matrix \mathbf{H} . \square

Remark. Given that Theorem 2 is satisfied over the expectation of the set of random vectors, which by the CLT is realised by Monte Carlo draws of random vectors as $d \rightarrow \infty$ the only way to really span the top m eigenvectors is to have selected a vector which lies in the m dimensional subspace of the P dimensional problem corresponding to those vectors, which would correspond to knowing those vectors *a priori*, defeating the point of using Lanczos at all.

Another way to see this is Theorem 6, which gives a bound on the distance between the smallest Lanczos-Ritz value and the minimal eigenvalue. Intuitively, as the Ritz values and weights form a discrete m -moment spectral approximation to the Hessian spectrum, hence the support of the discrete density, cannot approximately match the largest m eigenvalues. This can be seen in Figure 2c where we run Lanczos with $m = 30$ steps and capture the shape of the spectral density of a $\mathbf{H} \in \mathbb{R}^{10000 \times 10000}$ matrix including the negative eigenvalue of largest magnitude.

5 Deep Curvature

Based on the Lanczos algorithm, we are in a position to introduce to our package, the **Deep Curvature suite**, a

software package that allows analysis and visualisation of deep neural network curvature. The main features and functionalities of our package are:

- **Network training and evaluation** we provide a range of pre-built modern popular neural network structures, such as VGG and variants of ResNets, and various optimisation schemes in addition to the ones already present in the PyTorch frameworks, such as K-FAC and SWATS. These facilitates faster training and evaluation of the networks (although it is worth noting that any PyTorch-compatible optimisers or architectures can be easily integrated into our analysis framework).
- **Eigenspectrum analysis of the curvature matrices** Powered by the Lanczos techniques implemented in GPyTorch (Gardner et al., 2018) and outlined in Section 3, *with a single random vector* we use the Pearlmutter matrix-vector product trick for fast inference of the eigenvalues and eigenvectors of the common curvature matrices of the deep neural networks. In addition to the standard Hessian matrix, we also include the feature for inference of the eigen-information of the Generalised Gauss-Newton matrix, a commonly used positive-definite surrogate to Hessian³.
- **Advanced Statistics of Networks** In addition to the commonly used statistics to evaluate network training and performance such as the training and testing losses and accuracy, we support computations of more advanced statistics: For example, we support squared mean and variance of gradients and Hessians (and GGN), squared norms of Hessian and GGN, L2 and L-inf norms of the network weights and etc. These statistics are useful and relevant for a wide range of purposes such as the designs of second-order optimisers and network architecture.
- **Visualisations** For all main features above, we include accompanying visualisation tools. In addition, with the eigen-information obtained, we also feature visualisations of the **loss landscape** by studying the sensitivity of the neural network to perturbations of weights. While similar tools have been available, we would like to emphasise that one key difference is that, instead of the *random* directions

³The computation of the GGN-vector product is similar with the computational cost of two backward passes in the network. Also, GGN uses *forward-mode automatic differentiation* (FMAD) in addition to the commonly employed *backward-mode automatic differentiation* (RMAD). In the current PyTorch framework, the FMAD operation can be achieved using two equivalent RMAD operations.

as featured in some other packages, we explicitly perturb the weights in the *eigenvector* directions, which should yield more informative results.

Package Structure The main interface functions are organised as followed:

- **`.core`** The functions under `core` directories are the main analysis tools of the package. `train_network` allows network training and saving of the required statistics for subsequent spectrum learning. Based on the output of it, we additionally include tools for spectrum analysis (`compute_eigspectrum`) and advanced loss statistics (such as covariance of gradients and second order information like Hessian variance) in `compute_loss_stats` and `build_loss_landscape`.

We provide some pre-built network architectures (such as VGG and ResNet architectures) and optimizers apart from PyTorch natives (such as KFAC, SWATS optimizers). We additionally support Stochastic Weight Averaging proposed in (Izmailov et al., 2018). However, it is worth noting that any PyTorch compatible networks and optimizers can be easily integrated in our framework.

- **`.visualise`** This directory defines the various pre-defined visualisation functions for different purposes, including the visualisation of training, spectrum and the loss landscape.

To facilitate a quick start of our package, we have included an illustrated example of analysis on the VGG-16 network on CIFAR-100 dataset in Appendix D.

6 Examples on Small Random Matrices

In this section, we use some examples on small random matrices to showcase the power of our package that uses the Lanczos algorithm with random vectors to learn the spectral density. Here, we look at known random matrices with elements drawn from specific distributions which converge to known spectral densities in the asymptotic limit. Here we consider **Wigner Ensemble** (Wigner, 1993) and the **Marcenko Pastur** (Marchenko & Pastur, 1967), both of which are extensively used in simulations or theoretical analyses of deep neural network spectra (Pennington & Bahri, 2017; Choromanska et al., 2015a; Anonymous, 2020).

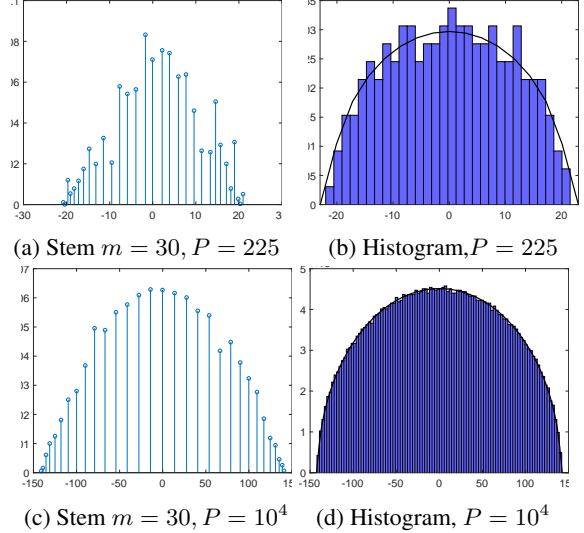


Figure 2: Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\mathbf{H} \in \mathbb{R}^{P \times P}$, where each element is drawn from a normal distribution with unit variance, converging to the Wigner semi circle.

6.1 Wigner Matrices

Wigner matrices can be defined in Definition A.1, and their distributions of eigenvalues are governed by the semi-circle distribution law (Theorem 3).

Theorem 3. *Let $\{M_P\}_{P=1}^{\infty}$ be a sequence of Wigner matrices, and for each P denote $X_P = M_P/\sqrt{P}$. Then μ_{X_P} , converges weakly, almost surely to the semi circle distribution,*

$$\sigma(x)dx = \frac{1}{2\pi} \sqrt{4 - x^2} \mathbf{1}_{|x| \leq 2} \quad (7)$$

For our experiments, we generate random matrices $\mathbf{H} \in \mathbb{R}^{P \times P}$ with elements drawn from the distribution $\mathcal{N}(0, 1)$ for $P = \{225, 10000\}$ and plot histogram of the spectra found by eigendecomposition, along with the predicted Wigner density (scaled by a factor of \sqrt{P}) in Figures 2b & 2d and compare them along with the discrete spectral density approximation learned by lanczos in $m = 30$ steps using a single random vector $d = 1$ in Figures 2a & 2c. It can be seen that even for a small number of steps $m \ll P$ and a single random vector, Lanczos impressively captures not only the support of the eigenvalue spectral density but also its shape. We note as discussed in section 4 that the 30 Ritz values here do not span the top 30 eigenvalues even approximately.

6.2 Marcenko-Pastur

An equally important limiting law for the limiting spectral density of many classes of matrices constrained to be positive definite, such as covariance matrices, is the

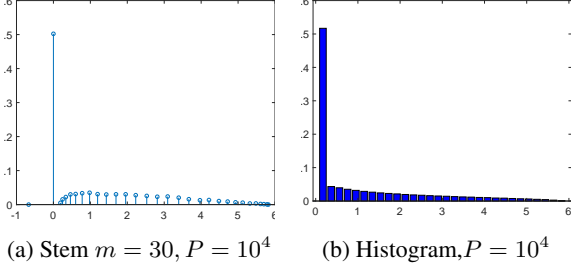


Figure 3: Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\mathbf{H} \in \mathbb{R}^{P \times P}$, where $\mathbf{H} = \mathbf{X}\mathbf{X}^T/k$, where each element of $\mathbf{X}^{P \times k}$, $k = 0.5P$ is a drawn from a normal distribution with unit variance, converging to the Marcenko-Pastur distribution with $q = 0.5$.

Marcenko-Pastur law (Marchenko & Pastur, 1967). Formally, given a matrix $\mathbf{X} \in \mathbb{R}^{P \times T}$ with i.i.d zero mean entries with variance $\sigma^2 < \infty$. Let $\lambda_1 \geq \lambda_2, \dots \geq \lambda_P$ be eigenvalues of $\mathbf{Y}_n = \frac{1}{T}\mathbf{X}\mathbf{X}^T$. The random measure $\mu_P(A) = \frac{1}{P} \#\{\lambda_j \in A\}$, $A \in \mathbb{R}$

Theorem 4. Assume that $P, N \rightarrow \infty$ and the ratio $P/N \rightarrow q \in (0, \infty)$ (this is known as the Kolmogorov limit) then $\mu_P \rightarrow \mu$ in distribution where

$$\begin{cases} (1 - \frac{1}{q})\mathbb{1}_{0 \in A} + \nu_{1/q}(A), & \text{if } q > 1 \\ \nu_q(A), & \text{if } 0 \leq q \leq 1 \end{cases} \quad (8)$$

$$d\nu_q = \frac{\sqrt{(\lambda_+ - x)(x - \lambda_-)}}{\lambda x 2\pi \sigma^2}, \lambda_{\pm} = \sigma^2(1 \pm \sqrt{q})^2 \quad (9)$$

Here, we construct a random matrix $\mathbf{X} \in \mathbb{P} \times \mathbb{T}$ with independently drawn elements from the distribution $\mathcal{N}(0, 1)$ and then form the matrix $\frac{1}{T}\mathbf{X}\mathbf{X}^T$, which is known to converge to the Marcenko-Pastur distribution. We use $P = \{225, 10000\}$ and $T = 2P$ and plot the associated histograms from full eigendecomposition in Figures 4b & 4d along with their $m = 30, d = 1$ Lanczos stem counterparts in Figures 4a & 4c. Similarly we see a faithful capturing not just of the support, but also of the general shape. We note that both for Figure 2 and Figure 4, the smoothness of the discrete spectral density for a single random vector increases significantly, even relative to the histogram.

We also run the same experiment for $P = 10000$ but this time with $T = 0.5P$ so that exactly half of the eigenvalues will be 0. We compare the Histogram of the eigenvalues in Figure 3b against its $m = 30, d = 1$ Lanczos stem plot in Figure 3a and find both the density at the origin, along with the bulk and support to be faithfully captured.

6.3 Comparison to Diagonal Approximations

As a proxy for deep neural network spectra, often the diagonal of the matrix (Bishop, 2006) or the diagonal

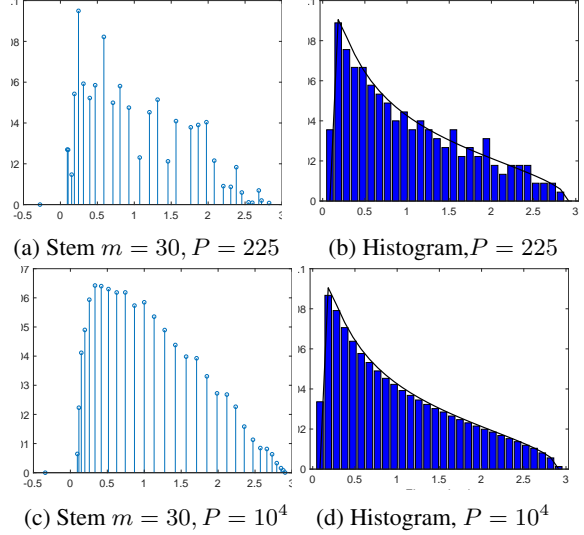


Figure 4: Lanczos stem plot for a single random vector with $m = 30$ steps compared to actual eigenvalue histogram for matrices of the form $\mathbf{H} \in \mathbb{R}^{P \times P}$, where $\mathbf{H} = \mathbf{X}\mathbf{X}^T/k$, where each element of $\mathbf{X}^{P \times k}$, $k = 2P$ is a drawn from a normal distribution with unit variance, converging to the Marcenko-Pastur distribution with $q = 2$.

of a surrogate matrix, such as the Fisher information, or that implied by the values of the Adam Optimizer (Chaudhari et al., 2016) is used. We plot the true eigenvalue estimates for random matrices pertaining to both the Marcenko-Pastur (Fig. 5a) and the Wigner density (Fig. 5b) in blue, along with the Lanczos estimate in red and the diagonal approximation in yellow. We see here that the diagonal approximation in both cases, fails to adequately the support or accurately model the spectral density, whereas the lanczos estimate is nearly indistinguishable from the true binned eigen-spectrum. This is of-course obvious from the mathematics of the unnormalised Wigner matrix. The diagonal elements are simply draws from the normal distribution $\mathcal{N}(0, 1)$ and so we expect the diagonal histogram plot to approximately follow this distribution (with variance 1). However the second moment of the Wigner Matrix can be given by the Frobenius norm identity

$$\mathbb{E}\left(\frac{1}{P} \sum_i \lambda_i^2\right) = \mathbb{E}\left(\frac{1}{P} \sum_{i,j=1}^P \mathbf{H}_{i,j}^2\right) = \mathbb{E}\left(\frac{1}{P} \chi_{P^2}^2\right) = P \quad (10)$$

Similarly for the Marcenko-Pastur distribution, We can easily see that each element of \mathbf{H} follows a chi-square distribution of $1/T\chi_T^2$, with mean 1 and variance $2/T$.

6.4 Synthetic Example

The curvature eigenspectrum of neural network often features a large spike at zero, a right-skewed bulk and

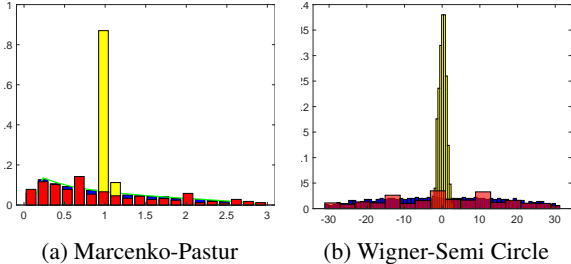


Figure 5: Two randomly generated matrices $\mathbf{H} \in \mathbb{R}^{500 \times 500}$ with the histogram of the true eigenvalues in blue, the Lanczos estimate $m = 30, d = 1$ in red and the diagonal approximation in yellow

some outliers (Sagun et al., 2016, 2017).⁴ In order to simulate the spectrum of a neural network, we generate a Matrix $\mathbf{H} \in \mathbb{R}^{1000 \times 1000}$ with 470 eigenvalues drawn from the uniform distribution from $[0, 15]$, 20 drawn from the uniform $[0, 60]$ and 10 drawn from the uniform $[-10, 0]$. The matrix is rotated through a rotation matrix U , i.e $\mathbf{H} = \mathbf{U}\mathbf{D}\mathbf{U}^T$ where \mathbf{D} is the diagonal matrix consisting of the eigenvalues and the columns are gaussian random vectors which are orthogonalised using Gram-Schmidt orthogonalisation. The resulting eigenspectrum is given in a histogram in Figure 6a and then using the same random vector, successive Lanczos stem plots for different number of iterations $m = [5, 30]$ are shown in Figure 6. Figure 6b, for a low number of steps, the degeneracy at $\lambda = 0$ is learned, as are the largest and smallest eigenvalues, some information is retained about the bulk density, but some of the outlier eigenvalues around $\lambda \approx 20$ and $\lambda \approx 30$ are completely missed out, along with all the negative outliers except the largest. For $m = 30$ even the shape of the bulk is accurately represented, as shown in Figure 6d. Here, we would like to emphasise that learning the outliers is important in the neural network context, as they relate to important properties of the network and the optimisation process (Ghorbani et al., 2019).

On the other hand, we note that the diagonal estimate in Figure 6c gives absolutely no spectral information, with no outliers shown (maximal and minimal diagonal elements being 5.3 and 3.3 respectively and it also gets the spectral mass at 0 wrong. This builds on section 6.3, as furthering the case against making diagonal approximations in general. In neural networks, the diagonal approximation is similar to positing no correlations between the weights. This is a very harsh assumption and usually a more reasonable assumption is to posit that the correlations between weights in the same layer are larger than between different layers, leading to a block diagonal approximation (Martens, 2016), however often when the layers have millions of parameters, full diagonal approx-

⁴Some examples of this can be found in later sections on real-life neural network experiments - see Figures 7 and 8.

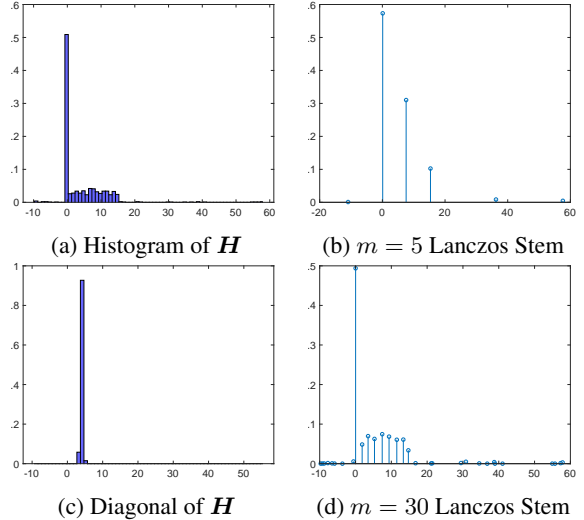


Figure 6: Generated matrices $\mathbf{H} \in \mathbb{R}^{1000 \times 1000}$ with known eigenspectrum and Lanczos stem plots for different values of $m = \{5, 15, 30\}$

imations are still used. (Bishop, 2006; Chaudhari et al., 2016).

7 Neural Network Examples

We showcase our spectral learning algorithm and visualization tool on real networks trained on real data-sets and we test on VGG networks (Simonyan & Zisserman, 2014). We train our neural networks using stochastic gradient descent with momentum $\rho = 0.9$, using a linearly decaying learning rate schedule. The learning rate at the t -th epoch is given by:

$$\alpha_t = \begin{cases} \alpha_0, & \text{if } \frac{t}{T} \leq 0.5 \\ \alpha_0 \left[1 - \frac{(1-r)(\frac{t}{T} - 0.5)}{0.4} \right], & \text{if } 0.5 < \frac{t}{T} \leq 0.9 \\ \alpha_0 r, & \text{otherwise} \end{cases} \quad (11)$$

where α_0 is the initial learning rate. $T = 300$ is the total number of epochs budgeted for all experiments. We set $r = 0.01$. We explicitly give an example code run in C We compare our method against recently developed open-source tools which calculate on the fly diagonal Hessian and Generalised Gauss-Newton diagonal approximations (Dangel et al., 2019).

7.1 VGG-16 CIFAR-100 Dataset

We train a 16-layer VGG network, comprising of $P = 15, 291, 300$ parameters on the CIFAR-100 dataset, using $\alpha_0 = 1$. Even for this relatively small model, the open-source Hessian and GGN exact diagonal computations require over 125GB of GPU memory and so to avoid re-implementing the library to support multiple GPUs and node communication we use the Monte

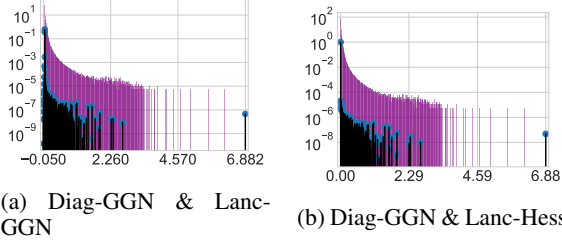


Figure 7: Diagonal Generalised Gauss Newton monte carlo approximation (Diag-GGN) against $m = 100$ Lanczos using Gauss-Newton vector products (Lanc-GGN) or Hessian vector products (Lanc-Hess)

Carlo approximation to the GGN diagonal against both our GGN-Lanczos and Hessian-Lanczos spectral visualizations. We plot a histogram of the Monte Carlo approximation of the diagonal GGN (Diag-GGN) against both the Lanczos GGN (Lanc-GGN) and Lanczos Hessian (Lanc-Hess) in Figure 7. Note that as the Lanc-GGN and Lanc-Hess are displayed as stem plots (with the discrete spectral density summing to 1 as opposed to the histogram area summing to 1).

We note that the Gauss-Newton approximation quite closely resembles its Hessian counterpart, capturing the majority of the bulk and the outlier eigenvectors at $\lambda_1 \approx 6.88$ and the triad near $\lambda_i \approx 2.29$. The Hessian does still have significant spectral mass on the negative axis, around 37%. However most of this is captured by a Ritz value at -0.0003 , with this removed, the negative spectral mass is only 0.05%. However as expected from our previous section, the Diag-GGN gives a very poor spectral approximation. It vastly overestimates the bulk region, which extends well beyond $\lambda \approx 1$ implied by Lanczos and adds many spurious outliers between 3 and the misses the largest outlier of 6.88.

Computational Cost Using a single NVIDIA GeForce GTX 1080 Ti GPU, the Gauss-Newton takes an average 26.5 seconds for each Lanczos iteration with the memory usage 2850Mb. Using the Hessian takes an average of 27.9 seconds for each Lanczos iteration with 2450Mb memory usage.

8 Effect of Varying Random Vectors

Given that the proofs for the moments of Lanczos matching those of the underlying spectral density, are true over the expectation over the set of random vectors and in practice we only use a Monte Carlo average of random vectors, or in our experiments using stem plots, just a single random vector. We justify this with the following Lemma

Lemma 1. *Let $\mathbf{u} \in \mathbb{R}^{P \times 1}$ random vector, where \mathbf{u}_i*

is zero mean and unit variance and finite 4'th moment $\mathbb{E}[\mathbf{u}_i^4] = m_4$. Then for $\mathbf{H} \in \mathbb{R}^{P \times P}$, then

$$\begin{aligned} i) \mathbb{E}[\mathbf{u}^T \mathbf{H} \mathbf{u}] &= \text{Tr } \mathbf{H} \\ ii) \text{Var}[\mathbf{u}^T \mathbf{H} \mathbf{u}] &\leq (2 + m_4) \text{Tr}(\mathbf{H}^T \mathbf{H}) \end{aligned}$$

Proof.

$$\mathbb{E}[\mathbf{u}^T \mathbf{H} \mathbf{u}] = \sum_{i,j=1}^P \mathbf{H}_{i,j} \mathbb{E}[\mathbf{u}_i \mathbf{v}_j] = \sum_{i=1}^P \mathbf{H}_{i,i} = \text{Tr } \mathbf{H} \quad (12)$$

$$\begin{aligned} \mathbb{E}[|\mathbf{u}^T \mathbf{H} \mathbf{u}|^2] &= \sum_{i,j} \sum_{k,l} \mathbf{H}_{i,j} \mathbf{H}_{k,l}^T \mathbb{E}[\mathbf{u}_i \mathbf{u}_j^T \mathbf{u}_k \mathbf{u}_l^T] \\ &= \sum_{i,j} \sum_{k,l} \mathbf{H}_{i,j} \mathbf{H}_{k,l}^T [\delta_{i,j} \delta_{k,l} + \delta_{i,l} \delta_{j,k} + \delta_{i,k} \delta_{j,l} + m_4 \delta_{i,j,k,l}] \\ &= (\text{Tr } \mathbf{H})^2 + (2 + m_4) \text{Tr}(\mathbf{H}^2) \end{aligned} \quad (13)$$

□

Remark. Let us consider the signal to noise ratio for some positive definite $\mathbf{H} \succ c\mathbf{I}$

$$\begin{aligned} \left(\frac{\sqrt{\text{Var}[\mathbf{u}^T \mathbf{H} \mathbf{u}]}{\mathbb{E}[\mathbf{u}^T \mathbf{H} \mathbf{u}]} \right)^2 &\propto \frac{1}{1 + \frac{\sum_{i \neq j}^P \lambda_i \lambda_j}{\sum_k^P \lambda_k^2}} = \frac{1}{1 + \frac{P-1 \langle \lambda_i \lambda_j \rangle}{\langle \lambda_k^2 \rangle}} \\ &\leq \frac{1}{1 + \frac{P-1}{\kappa^2}} \end{aligned} \quad (14)$$

where $\langle \dots \rangle$ denotes the arithmetic average. For the extreme case of all eigenvalues being identical, the condition number $\kappa = 1$ and hence this reduces to $1/P \rightarrow 0$ in the $P \rightarrow \infty$ limit, whereas for a rank-1 matrix, this ratio remains 1. For the MP density, which well models neural network spectra, κ is not a function of P as $P \rightarrow \infty$ and hence we also expect this benign dimensional scaling to apply.

We verify this high dimensional result experimentally, by running the same spectral visualisation as in Section 7.1 but using two different random vectors. We plot the results in Figure 8. We find both figures 8a & 8b to be close to visually indistinguishable. There are minimal differences in the extremal eigenvalues, with former giving $\{\lambda_1, \lambda_n\} = \{6.8885, -0.0455\}$ and the latter $\{6.8891, -0.0456\}$, but the degeneracy at 0, bulk, triplet of outliers at 2.27 and the large outlier at 6.89 is unchanged. We include the code to run in C.1

8.1 Why we don't kernel smooth

Concurrent work, which has also used Lanczos with the Pearlmutter trick to learn the Hessian (Yao et al., 2018; Ghorbani et al., 2019), typically uses n_v random vectors and then uses kernel smoothing, to give a final density. In

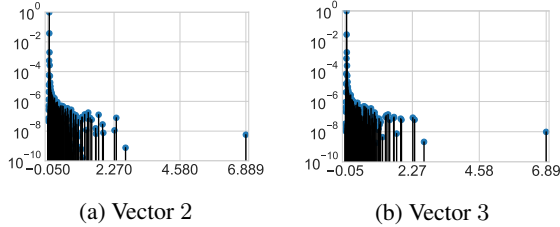


Figure 8: VGG16 Epoch 300 end of training Lanczos stem plot for different random vectors

this section we argue that beyond costing a factor of n_v more computationally, that the extra compute extended in order to get more accurate moment estimates, which we already argued in Section 8 are asymptotically error free, is wasted due to the kernel smoothing (Granzio et al., 2019). The smoothed spectral density takes the form:

$$\tilde{p}(\lambda) = \int k_\sigma(\lambda - \lambda') p(\lambda') d\lambda' = \sum_{i=1}^n w_i k_\sigma(\lambda - \lambda_i) \quad (15)$$

We make some assumptions regarding the nature of the kernel function, $k_\sigma(\lambda - \lambda_i)$, in order to prove our main theoretical result about the effect of kernel smoothing on the moments of the underlying spectral density. Both of our assumptions are met by (the commonly employed) Gaussian kernel.

Assumption 1. *The kernel function $k_\sigma(\lambda - \lambda_i)$ is supported on the real line $[-\infty, \infty]$.*

Assumption 2. *The kernel function $k_\sigma(\lambda - \lambda_i)$ is symmetric and permits all moments.*

Theorem 5. *The m -th moment of a Dirac mixture $\sum_{i=1}^n w_i \delta(\lambda - \lambda_i)$, which is smoothed by a kernel k_σ satisfying assumptions 1 and 2, is perturbed from its unsmoothed counterpart by an amount $\sum_{i=1}^n w_i \sum_{j=1}^{r/2} \binom{r}{2j} \mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) \lambda_i^{m-2j}$, where $r = m$ if m is even and $m - 1$ otherwise. $\mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j})$ denotes the $2j$ -th central moment of the kernel function $k_\sigma(\lambda)$.*

Proof. The moments of the Dirac mixture are given as,

$$\langle \lambda^m \rangle = \sum_{i=1}^n w_i \int \delta(\lambda - \lambda_i) \lambda^m d\lambda = \sum_{i=1}^n w_i \lambda_i^m. \quad (16)$$

The moments of the modified smooth function (Equation

equation 15) are

$$\begin{aligned} \langle \tilde{\lambda}^m \rangle &= \sum_{i=1}^n w_i \int k_\sigma(\lambda - \lambda_i) \lambda^m d\lambda \\ &= \sum_{i=1}^n w_i \int k_\sigma(\lambda') (\lambda' + \lambda_i)^m d\lambda' \\ &= \langle \lambda^m \rangle + \sum_{i=1}^n w_i \sum_{j=1}^{r/2} \binom{r}{2j} \mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) \lambda_i^{m-2j}. \end{aligned} \quad (17)$$

We have used the binomial expansion and the fact that the infinite domain is invariant under shift reparameterization and the odd moments of a symmetric distribution are 0. \square

Remark. The above proves that kernel smoothing alters moment information, and that this process becomes more pronounced for higher moments. Furthermore, given that $w_i > 0$, $\mathbb{E}_{k_\sigma(\lambda)}(\lambda^{2j}) > 0$ and (for the GGN $\lambda_{i,j} > 0$, the corrective term is manifestly positive, so the smoothed moment estimates are biased.

9 Local loss landscape

The Lanczos algorithm with enforced orthogonality initialised with a random vector gives a moment matched discrete approximation to the Hessian spectrum. However this information is local to the point in weight space \mathbf{w} and the quadratic approximation may break down within the near vicinity. To investigate this, we use the **loss landscape visualisation** function of our package: We display this for the VGG-16 on CIFAR-100 in Figure 11. We see for the training loss 9a that the eigenvector corresponding to the largest eigenvalue $\lambda = 6.88$ only very locally corresponds to the sharpest increase in loss for the training, with other extremal eigenvectors, corresponding to the eigenvalues $\lambda = \{2.67, 2.35\}$ overtaking it in loss change relatively rapidly. Interestingly for the testing loss, all the extremal eigenvectors change the loss much more rapidly, contradicting previous assertions that the test loss is a "shifted" version of the training loss (He et al., 2019; Izmailov et al., 2018). We do however note some small asymmetry between the changes in loss along the opposite ends of the eigenvectors. The flat directions remain flat locally and some of the eigenvectors corresponding to negative values correspond to decreases in test loss. We include the code in C.2

CIFAR-10 Dataset

To showcase the ability of our software to handle multiple datasets we display the Hessian of the VGG-16 trained in an identical fashion as its CIFAR-100 counterpart of CIFAR-10 in Figure 13, along with the a plot

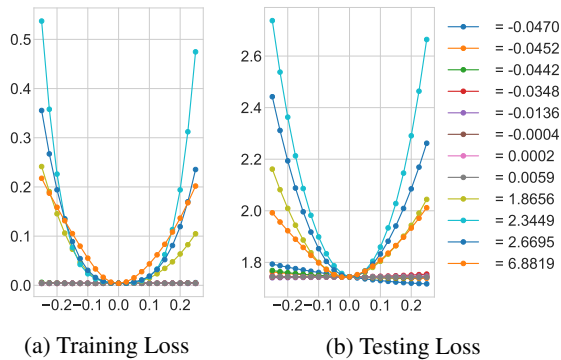


Figure 9: VGG-16 CIFAR-100 Loss surface visualised along 6 negative and position eigenvalues

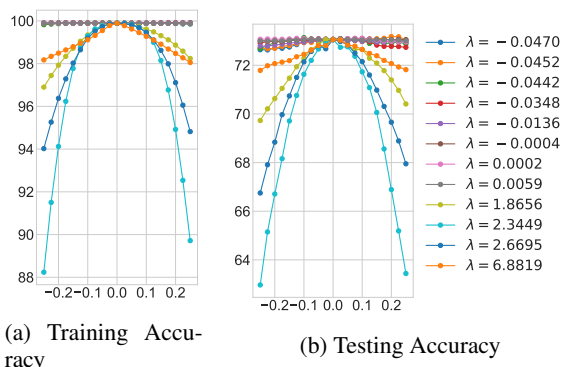


Figure 10: VGG-16 CIFAR-100 Accuracy surface visualised along 6 negative and position eigenvalues

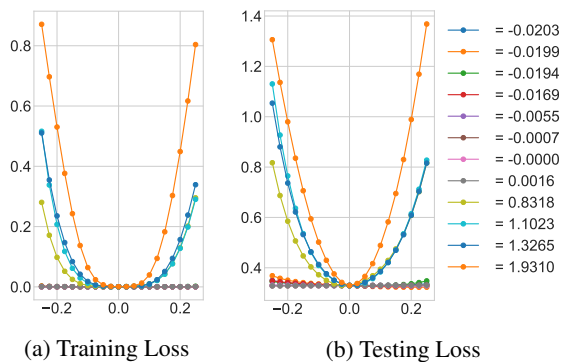


Figure 11: VGG-16 CIFAR-10 Loss surface visualised along 6 negative and position eigenvalues

of a selection of Ritz vectors traversing the training loss surface in Figure 11a and testing loss surface in Figure 11b along with also the training accuracy surface (Figure 12a) and testing accuracy surface (Figure 12b).

10 Conclusion

We introduce the **Deep Curvature** suite in **PyTorch** framework, based on the Lanczos algorithm implemented in **GPyTorch** (Gardner et al., 2018), that allows deep learning practitioners to learn spectral density in-

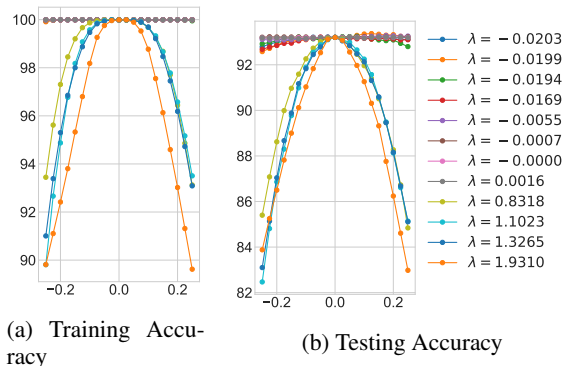


Figure 12: VGG-16 CIFAR-10 Accuracy surface visualised along negative and positive eigenvalues

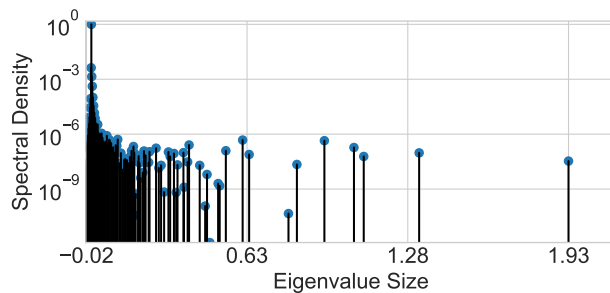


Figure 13: VGG-16 CIFAR-10 epoch 300 Hessian

formation as well as eigenvalue/eigenvector pairs of the curvature matrices at specific points in weight space. Together with the software, we also include a succinct summary of the linear algebra, iterative method theory including proofs of convergence and misconceptions and stochastic trace estimation that form the theoretical underpinnings of our work. Finally, we also included various examples of our package of analysis of both synthetic data and real data with modern neural network architectures.

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 *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.
- Akemann, G., Baik, J., and Di Francesco, P. *The Oxford handbook of random matrix theory*. Oxford University Press, 2011.
- Anonymous. Towards understanding the true loss surface of deep neural networks using random matrix theory and iterative spectral methods. In *Submitted to International Conference on Learning Representations, 2020*. URL <https://openreview.net/forum?id=H1gza2NtwH>. under review.

- 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.
- Bishop, C. M. *Pattern recognition and machine learning*. Springer, 2006.
- Chaudhari, P., Choromanska, A., Soatto, S., LeCun, Y., Baldassi, C., Borgs, C., Chayes, J., Sagun, L., and Zecchina, R. Entropy-SGD: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.
- Chollet, F. Keras. <https://github.com/fchollet/keras>, 2015.
- Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., and LeCun, Y. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pp. 192–204, 2015a.
- Choromanska, A., LeCun, Y., and Arous, G. B. Open problem: The landscape of the loss surfaces of multilayer networks. In *Conference on Learning Theory*, pp. 1756–1760, 2015b.
- Dangel, F., Kunstner, F., and Hennig, P. Backpack: Packing more into backprop. *arXiv preprint arXiv:1912.10985*, 2019.
- Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems*, pp. 2933–2941, 2014.
- Gardner, J., Pleiss, G., Weinberger, K. Q., Bindel, D., and Wilson, A. G. Gpytorch: Blackbox matrix-matrix Gaussian process inference with GPU acceleration. In *Advances in Neural Information Processing Systems*, pp. 7576–7586, 2018.
- Ghorbani, B., Krishnan, S., and Xiao, Y. An investigation into neural net optimization via Hessian eigenvalue density. *arXiv preprint arXiv:1901.10159*, 2019.
- Golub, G. H. and Meurant, G. Matrices, moments and quadrature. *Pitman Research Notes in Mathematics Series*, pp. 105–105, 1994.
- Golub, G. H. and Van Loan, C. F. *Matrix computations*, volume 3. JHU press, 2012.
- Granzio, D., Ru, B., Zohren, S., Dong, X., Osborne, M., and Roberts, S. Meme: An accurate maximum entropy method for efficient approximations in large-scale machine learning. *Entropy*, 21(6):551, 2019.
- He, H., Huang, G., and Yuan, Y. Asymmetric valleys: Beyond sharp and flat local minima. *arXiv preprint arXiv:1902.00744*, 2019.
- 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.
- Hutchinson, M. F. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.
- Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.
- Izmailov, P., Podoprikin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *Uncertainty in Artificial Intelligence (UAI)*, 2018.
- Izmailov, P., Maddox, W. J., Kirichenko, P., Garipov, T., Vetrov, D., and Wilson, A. G. Subspace inference for bayesian deep learning. *Uncertainty in Artificial Intelligence (UAI)*, 2019.
- Kohler, J., Daneshmand, H., Lucchi, A., Zhou, M., Neymeyr, K., and Hofmann, T. Exponential convergence rates for batch normalization: The power of length-direction decoupling in non-convex optimization. *arXiv preprint arXiv:1805.10694*, 2018.
- Li, H., Xu, Z., Taylor, G., and Goldstein, T. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.
- Lin, L., Saad, Y., and Yang, C. Approximating spectral densities of large matrices. *SIAM Review*, 58(1):34–65, 2016.
- Maddox, W. J., Izmailov, P., Garipov, T., Vetrov, D. P., and Wilson, A. G. A simple baseline for bayesian uncertainty in deep learning. In *Advances in Neural Information Processing Systems*, pp. 13132–13143, 2019.
- Marchenko, V. A. and Pastur, L. A. Distribution of eigenvalues for some sets of random matrices. *Matematicheskii Sbornik*, 114(4):507–536, 1967.
- Martens, J. *Second-order optimization for neural networks*. PhD thesis, University of Toronto, 2016. URL http://www.cs.toronto.edu/~jmartens/docs/thesis_phd_martens.pdf.
- Martens, J. and Sutskever, I. Training deep and recurrent networks with Hessian-free optimization. In *Neural networks: Tricks of the trade*, pp. 479–535. Springer, 2012.
- Meurant, G. and Strakoš, Z. The Lanczos and conjugate gradient algorithms in finite precision arithmetic. *Acta Numerica*, 15:471–542, 2006.
- Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in Pytorch. 2017.

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

Pennington, J. and Bahri, Y. Geometry of neural network loss surfaces via random matrix theory. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2798–2806. JMLR.org, 2017.

Ritter, H., Botev, A., and Barber, D. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=Skdvd2xAZ>.

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? In *Advances in Neural Information Processing Systems*, pp. 2483–2493, 2018.

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

Tao, T. *Topics in random matrix theory*, volume 132. American Mathematical Soc., 2012.

Ubaru, S. and Saad, Y. Applications of trace estimation techniques.

Vinyals, O. and Povey, D. Krylov subspace descent for deep learning. In *Artificial Intelligence and Statistics*, pp. 1261–1268, 2012.

Wigner, E. P. Characteristic vectors of bordered matrices with infinite dimensions i. In *The Collected Works of Eugene Paul Wigner*, pp. 524–540. Springer, 1993.

Yao, Z., Gholami, A., Lei, Q., Keutzer, K., and Mahoney, M. W. Hessian-based analysis of large batch training and robustness to adversaries. In *Advances in Neural Information Processing Systems*, pp. 4949–4959, 2018.

A Mathematical Definitions

Definition A.1. Let $\{Y_i\}$ and $\{Z_{ij}\}_{1 \leq i \leq j}$ be two real-valued families of zero mean, i.i.d random variables, Furthermore suppose that $\mathbb{E}Z_{12}^2 = 1$ and for each $k \in \mathbb{N}$

$$\max(E|Z_{12}^k, E|Y_1|^k) < \infty \quad (18)$$

Consider a $P \times P$ symmetric matrix M_P , whose entries are given by

$$\begin{cases} M_P(i, i) = Y_i \\ M_P(i, j) = Z_{ij} = M_P(j, i), \quad \text{if } x \geq 1 \end{cases} \quad (19)$$

The Matrix M_P is known as a real symmetric Wigner matrix.

B Lanczos Algorithm Primer

B.1 Why does anyone use Power Iterations?

The Lanczos method be be explicitly derived by considering the optimization of the Rayleigh quotient (Golub & Van Loan, 2012)

$$r(\mathbf{v}) = \frac{\mathbf{v}^T \mathbf{H} \mathbf{v}}{\mathbf{v}^T \mathbf{v}} \quad (20)$$

over the entire Krylov subspace $\mathcal{K}_m(\mathbf{H}, \mathbf{v})$ as opposed to power iteration which is a particular vector in the Krylov subspace $\mathbf{u} = \mathbf{H}^m \mathbf{v}$. Despite this, practitioners looking to learn the leading eigenvalue, very often resort to the power iteration, likely due to its implementational simplicity. We showcase the power iterations relative inferiority to Lanczos with the following convergence theorems

Theorem 6. Let $H^{P \times P}$ be a symmetric matrix with eigenvalues $\lambda_1 \geq \dots \geq \lambda_P$ and corresponding orthonormal eigenvectors z_1, \dots, z_P . If $\theta_1 \geq \dots \geq \theta_m$ are the eigenvalues of the matrix T_m obtained after m Lanczos steps and q_1, \dots, q_m the corresponding Ritz eigenvectors then

$$\begin{aligned} \lambda_1 \geq \theta_1 &\geq \lambda_1 - \frac{(\lambda_1 - \lambda_n) \tan^2(\theta_1)}{(c_{m-1}(1 + 2\rho_1))^2} \\ \lambda_P \leq \theta_k &\leq \lambda_m + \frac{(\lambda_1 - \lambda_n) \tan^2(\theta_1)}{(c_{m-1}(1 + 2\rho_1))^2} \end{aligned} \quad (21)$$

where c_m is the Chebyshev polyomial of order k . $\cos \theta_1 = |\mathbf{q}_1^T \mathbf{z}_1|$ & $\rho_1 = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$

Proof. see (Golub & Van Loan, 2012). □

λ_1/λ_2	$m = 5$	$m = 10$	$m = 15$	$m = 20$
1.5	$\frac{1.1 \times 10^{-4}}{3.9 \times 10^{-2}}$	$\frac{2 \times 10^{-10}}{6.8 \times 10^{-4}}$	$\frac{3.9 \times 10^{-16}}{1.2 \times 10^{-5}}$	$\frac{7.4 \times 10^{-22}}{2.0 \times 10^{-7}}$
1.1	$\frac{2.7 \times 10^{-2}}{4.7 \times 10^{-1}}$	$\frac{5.5 \times 10^{-5}}{1.8 \times 10^{-1}}$	$\frac{1.1 \times 10^{-7}}{6.9 \times 10^{-2}}$	$\frac{2.1 \times 10^{-10}}{2.7 \times 10^{-2}}$
1.01	$\frac{5.6 \times 10^{-1}}{9.2 \times 10^{-1}}$	$\frac{1.0 \times 10^{-1}}{8.4 \times 10^{-1}}$	$\frac{1.5 \times 10^{-2}}{7.6 \times 10^{-1}}$	$\frac{2.0 \times 10^{-3}}{6.9 \times 10^{-1}}$

Table 1: L_{k-1}/R_{k-1} For different values of spectral gap λ_1/λ_2 and iteration number m , Table from (Golub & Van Loan, 2012)

Theorem 7. *Assuming the same notation as in Theorem 6, after m power iteration steps the corresponding extremal eigenvalue estimate is lower bounded by*

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - (\lambda_1 - \lambda_n) \tan^2(\theta_1) \left(\frac{\lambda_2}{\lambda_1} \right)^{2m-1} \quad (22)$$

From the rapid growth of orthogonal polynomials such as Chebyshev, we expect Lanczos superiority to significantly emerge for larger spectral gap and iteration number. To verify this experimentally, we collect the non identical terms in the equations 21 and 22 of the lower bounds for λ_1 derived by Lanczos and Power iteration and denote them L_{k-1} and R_{k-1} respectively. For different values of λ_1/λ_2 and iteration number m we give the ratio of these two quantities in Table 1. As can be clearly seen, the Lanczos lower bound is always closer to the true value, this improves with the iteration number m and its relative edge is reduced if the spectral gap is decreased.

B.2 The Problem of Moments: Spectral Density Estimation Using Lanczos

In this section we show that that the Lanczos Tri-Diagonal matrix corresponds to an orthogonal polynomial basis which matches the moments of $v^T \mathbf{H}^m v$ and that when v is a zero mean random vector with unit variance, this corresponds to the moment of the underlying spectral density.

Stochastic trace estimation Using the expectation of quadratic forms, for zero mean, unit variance random vectors

$$\begin{aligned} \mathbb{E}_v \text{Tr}(v^T \mathbf{H}^m v) &= \text{Tr} \mathbb{E}_v (v v^T \mathbf{H}^m) = \text{Tr}(\mathbf{H}^m) \\ &= \sum_{i=1}^P \lambda_i^m = P \int_{\lambda \in \mathcal{D}} \lambda^m d\mu(\lambda) \end{aligned} \quad (23)$$

where we have used the linearity of trace and expectation. Hence in expectation over the set of random vectors, the trace of the inner product of v and $\mathbf{H}^m v$ is equal to the m 'th moment of the spectral density of \mathbf{H} .

Lanczos-Stieltjes The Lanczos tri-diagonal matrix T can be derived from the Moment matrix M , corresponding to the discrete measure $d\alpha(\lambda)$ satisfying the moments $\mu_i = v^T \mathbf{H}^i v = \int \lambda^i d\alpha(\lambda)$ (Golub & Meurant, 1994)

$$M = \begin{bmatrix} 1 & v^T \mathbf{H} v & \dots & v^T \mathbf{H}^{m-1} v \\ v^T \mathbf{H} v & v^T \mathbf{H}^2 v & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ v^T \mathbf{H}^{m-1} v & \dots & \dots & v^T \mathbf{H}^{2m-2} v \end{bmatrix}$$

and hence for a zero mean unit variance initial seed vector, the eigenvector/eigenvalue pairs of T contain information about the spectral density of \mathbf{H} as shown in section B.2. This is given by the following Theorem

Theorem 8. *The eigenvalues of T_k are the nodes t_j of the Gauss quadrature rule, the weights w_j are the squares of the first elements of the normalized eigenvectors of T_k*

Proof. See Golub & Meurant (1994) □

A quadrature rule is a relation of the form,

$$\int_a^b f(\lambda) d\mu(\lambda) = \sum_{j=1}^M \rho_j f(t_j) + R[f] \quad (24)$$

for a function f , such that its Riemann-Stieltjes integral and all the moments exist on the measure $d\mu(\lambda)$, on the interval $[a, b]$ and where $R[f]$ denotes the unknown remainder. The first term on the RHS of equation 24 using Theorem 8 can be seen as a discrete approximation to the spectral density matching the first m moments $v^T \mathbf{H}^m v$ (Golub & Meurant, 1994; Golub & Van Loan, 2012)

For n_v starting vectors, the corresponding discrete spectral density is given as

$$p(\lambda) = \frac{1}{n_v} \sum_{l=1}^{n_v} \left(\sum_{k=1}^m (\tau_k^{(l)})^2 \delta(\lambda - \lambda_k^{(l)}) \right), \quad (25)$$

where $\tau_k^{(l)}$ corresponds to the first entry of the eigenvector of the k -th eigenvalue, λ_k , of the Lanczos tri-diagonal matrix, T , for the l -th starting vector (Ubaru & Saad; Lin et al., 2016).

B.3 Computational Complexity

For large matrices, the computational complexity of the algorithm depends on the Hessian vector product, which for neural networks is $\mathcal{O}(mNP)$ where P denotes the number of parameters in the network, m is the number of Lanczos iterations and N is the number of data-points. The full re-orthogonalisation adds two matrix vector products, which is of cost $\mathcal{O}(m^2P)$, where typically $m^2 \ll N$. Each random vector used can be seen

Algorithm 1 Lanczos Algorithm

```
1: Input: Hessian vector product  $\{Hv\}$ , number of
   steps  $m$ 
2: Output: Ritz eigenvalue/eigenvector pairs  $\{\lambda_i, u_i\}$ 
   & quadrature weights  $\tau_i$ 
3: Set  $v := v/\sqrt{v^T v}$ 
4: Set  $\beta := 0, v_{old} := v$ 
5: Set  $V(:, 1) := v$ 
6: for  $j$  in  $1, \dots, m$  do
7:    $w = Hv - \beta v_{old}$ 
8:    $T(j, j) = \alpha = w^T w$ 
9:    $w = w - \alpha w - VV^T w$ 
10:   $\beta = \sqrt{w^T w}$ 
11:   $v_{old} = v$ 
12:   $v = w/\beta$ 
13:   $V(:, j+1) = v$ 
14:   $T(j, j+1) = T(j+1, 1) = \beta$ 
15: end for
16:  $\{\lambda_i, e_i\} = eig(T)$ 
17:  $u_i = Ve_i$ 
18:  $\tau_i = (e_i^T [1, 0, 0\dots 0])^2$ 
```

as another full run of the Lanczos algorithm, so for d random vectors the total complexity is $\mathcal{O}(dmP(N+m))$

Importance of keeping orthogonality The update equations of the Lanczos algorithm lead to a tri-diagonal matrix $T = \mathbb{R}^{m \times m}$, whose eigenvalues represent the approximated eigenvalues of the matrix H and whose eigenvectors, when projected back into the Krylov-subspace, $\mathcal{K}(H, v)$, give the approximated eigenvectors of H . In finite precision, it is known (Meurant & Strakoš, 2006) that the Lanczos algorithm fails to maintain orthogonality between its Ritz vectors, with corresponding convergence failure. In order to remedy this, we re-orthonormalise at each step (Bai et al., 1996) (as shown in line 9 of Algorithm 1) and observe a high degree of orthonormality between the Ritz eigenvectors. Orthonormality is also essential for achieving accurate spectral resolution as the Ritz value weights are given by the squares of the first elements of the normalised eigenvectors. For the practitioner wishing to reduce the computational cost of maintaining orthogonality, there exist more elaborate schemes (Meurant & Strakoš, 2006; Golub & Meurant, 1994).

C Code Run

In our interface, to train the network, we call:

```
train_network(
    dir='result/VGG16-CIFAR100/',
    dataset='CIFAR100',
```

```
# dataset='CIFAR10', if testing on
  CIFAR-10 data-set.
data_path='data/'
data_path='data/',
epochs=300,
model='VGG16'
# model='PreResNet110', if training
  with Preactivated ResNet with 110
  layers instead.
optimizer='SGD',
schedule='linear',
# This will direct the learning rate
  schedule to be the linear schedule
  defined in Equation 19.
optimizer_kwargs={
  'lr': 0.05,
  'momentum': 0.9,
  'weight_decay': 5e-4}
```

C.1 Running the Example C100

To replicate this example, use the following command:

```
compute_eigenspectrum(
    dataset='CIFAR100',
    # dataset='CIFAR10', if testing on
      CIFAR-10 data-set.
    data_path='data/'
    data_path='data/',
    model='VGG16',
    checkpoint_path='result/VGG16-
      CIFAR100/checkpoint-00300.pt',
    save_spectrum_path='result/VGG16-
      CIFAR100/spectra/spectrum-00300-
      ggn-lanczos',
    # change accordingly, if considering
      the Hessian matrix
    save_eigvec=True,
    lanczos_iters=100,
    curvature_matrix='ggn-lanczos',
    # curvature_matrix='hessian-lanczos',
      if considering the Hessian matrix
      instead
)
```

C.2 Running the Example C10

To replicate this result, use the following command:

```
build_loss_landscape(
    dataset='CIFAR100',
    # dataset='CIFAR10', if testing on
      CIFAR-10 data-set.
    data_path='data/'
    model='VGG16',
    # Change accordingly for other
      architectures such as PreResNet110
    dist=0.25,
    n_points=21,
    spectrum_path='result/VGG16-CIFAR100/
      spectra/spectrum-00300-ggn-lanczos
      ',
```

```

    # change accordingly for the Hessian
    result
    checkpoint_path='result/VGG16-
    CIFAR100/checkpoint-00300.pt',
    save_path='result/VGG16-CIFAR100/
    losslandscape-00300.npz'
)

plot_loss_landscape('result/VGG16-
    CIFAR100/losslandscape-00300.npz')
plt.show()

```

D An Illustrated Example

We give an illustration on an example of using the MLRG-DeepCurvature package and more details, including detailed documentation of each user function and the output of this particular example, can be found at our open-source repository. We begin by importing the necessary functions and packages:

```

from core import *
from visualise import *
import matplotlib.pyplot as plt

```

In this example, we train a VGG16 network on CIFAR-100 for 100 epochs. In a test computer with AMD Ryzen 3700X CPU and NVIDIA GeForce RTX 2080 Ti GPU, each epoch of training takes less than 10 seconds:

```

train_network(
    dir='result/VGG16-CIFAR100/',
    dataset='CIFAR100',
    data_path='data/',
    epochs=100,
    model='VGG16',
    optimizer='SGD',
    optimizer_kwargs={
        'lr': 0.03,
        'momentum': 0.9,
        'weight_decay': 5e-4
    }
)

```

This step generates a number of training statistics files (starting with stats-) and checkpoint files (checkpoint-00XXX.pt, where XXX is the epoch number) that contain the `state_dict` of the optimizer and the model. We may additionally visualise the training processes by looking at the basic statistics by calling `plot_training` function. With the checkpoints generated, we may now compute analyse the eigenspectrum of the curvature matrix evaluated at the desired point of training. For example, if we would like to evaluate the Generalised Gauss-Newton (GGN) matrix at the end of the training with 20 Lanczos iterations, we call:

```

compute_eigenspectrum(
    dataset='CIFAR100',
    data_path='data/',

```

```

    model='VGG16',
    checkpoint_path='result/VGG16-
    CIFAR100/checkpoint-00100.pt',
    save_spectrum_path='result/VGG16-
    CIFAR100/spectra/spectrum-00100-
    ggn_lanczos',
    save_eigvec=True,
    lanczos_iters=20,
    curvature_matrix='ggn_lanczos',
)

```

This function call saves the spectrum results (including eigenvalues, eigenvectors and other related statistics) in the `save_spectrum_path` path string defined. To visualise the spectrum as stem plot similar to Figure 7, we simply call:

```

plot_spectrum('lanczos', path='result/
    VGG16-CIFAR100/spectra/spectrum-00100-
    ggn_lanczos.npz')
plt.show()

```

Finally, with the eigenvalues and eigenvectors computed, we might be interested in knowing how sensitive the network is to perturbation along these directions. To achieve this, we first construct a loss landscape by setting the number of query points and maximum perturbation to apply. To achieve that, we call:

```

build_loss_landscape(
    dataset='CIFAR100',
    data_path='data/',
    model='VGG16',
    dist=1.,
    n_points=21,
    spectrum_path='result/VGG16-CIFAR100/
    spectra/spectrum-00100-ggn_lanczos',
    checkpoint_path='result/VGG16-
    CIFAR100/checkpoint-00100.pt',
    save_path='result/VGG16-CIFAR100/
    losslandscape-00100.npz'
)

plot_loss_landscape('result/VGG16-
    CIFAR100/losslandscape-00100.npz')
plt.show()

```

where in this example, we set the maximum perturbation to be 1 (`dist` argument) and number of query points along each direction to be 21 (`n_points` argument). This will produce diagrams similar to Figure 11 that show the effect of perturbation in the loss and accuracy for both training and testing.