Spectral Properties of Neural Network Architectures and Optimization

Nadezhda Ilieva

Master Research Project

Supervisor
Dr. Nikita Doikov
MLO EPFL

Supervisor
Prof. Martin Jaggi
MLO EPFL

January 2025

Contents

Chapter 1

Introduction

The Hessian matrix is a fundamental structure in optimization and machine learning. In the context of neural networks, the Hessian of the training loss with respect to the model parameters is important in determining many behaviors of neural networks. The spectrum of the Hessian, defined by its eigenvalues, can describe the optimization problem, i.e., indicate whether the problem is convex or concave. In addition, it can help determine the local geometry at stationary points, identifying local minima, local maxima, or saddle points. Figure (1.1) illustrates the optimization landscapes around a stationary point for different eigenvalue configurations: all positive eigenvalues indicate a local minimum, all negative eigenvalues indicate a local maximum, and a mix of positive and negative eigenvalues corresponds to a saddle point.

The Hessian and its spectrum can be used in the design of optimization algorithms, with the Newton method being a key example that uses the gradient and the inverse of the Hessian to compute an update for the parameters. In addition, Hessian eigenvalues can influence the convergence rate of first-order optimization algorithms and are hypothesized to affect the generalization performance of neural networks.

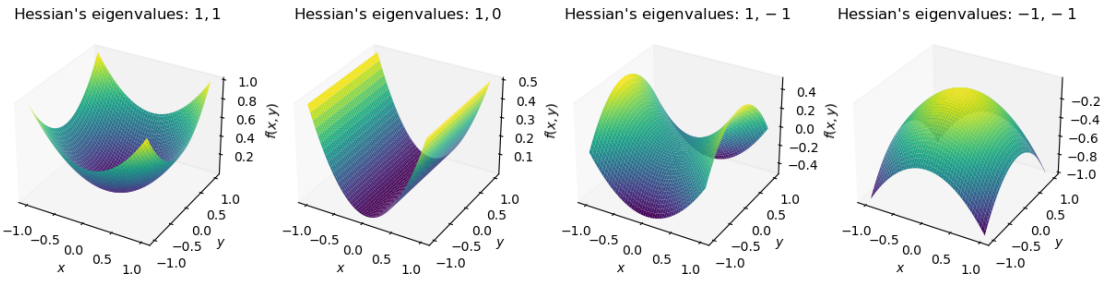

Figure 1.1: Optimization landscapes around a stationary point for different eigenvalue configurations

Unfortunately, even for moderately sized models, the exact computation of the Hessian eigenvalues is impossible. In this project, we investigate tools to estimate the spectrum of the Hessian matrix and study the properties of optimization landscapes of different neural network architectures at various stages of training.

1.1 Related work

Ghorbani et al. [1] studied the evolution of the Hessian spectrum during optimization and highlighted the rapid emergence of large isolated eigenvalues in non-batch normalized networks, which significantly influence optimization speed. Yao et al. [2] introduced PyHessian, a framework for fast Hessian computations, and revealed that batch normalization does not always smooth loss landscapes. Zhang et al. [3] identified "block heterogeneity" in Transformers, showing that SGD struggles in heterogeneous scenarios while Adam's coordinate-wise learning rates mitigate this limitation. All of the presented works rely on methods for estimating the spectrum of the Hessian matrix. Specifically, they use the *stochastic Lanczos quadrature* algorithm, which relies on the Lanczos method.

1.2 Notation

Neural networks are trained by minimizing a loss function that quantifies the error between predicted outputs and actual labels in a given dataset. The dataset, denoted as $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, consists of N data points, where $\mathbf{x}_i$ represents the input features and y_i is the corresponding label for the i-th data point. The loss function $f(\boldsymbol{\theta})$ depends on the network parameters $\boldsymbol{\theta}$ and is defined as the average over individual losses computed for each data point:

$$f(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{\theta}), \tag{1.1}$$

where $f_i(\boldsymbol{\theta})$ is the loss contribution from the i-th data point. The goal of training is to find the parameter set $\boldsymbol{\theta}^*$ that minimizes the loss function:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta}). \tag{1.2}$$

The Hessian matrix of the loss function, denoted as $\nabla^2 f(\boldsymbol{\theta}) \in \mathbb{R}^{n \times n}$ is a square matrix of the second-order partial derivatives of f:

$$\nabla^2 f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_2} & \cdots \\ \frac{\partial^2 f}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} \tag{1.3}$$

It can also be expressed as the average of the individual Hessians for each data point:

$$\nabla^2 f(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} \nabla^2 f_i(\boldsymbol{\theta}), \tag{1.4}$$

where $\nabla^2 f_i(\boldsymbol{\theta})$ is the Hessian of $f_i(\boldsymbol{\theta})$. Finally, the spectrum of the Hessian matrix $\nabla^2 f(\boldsymbol{\theta})$ consists of its eigenvalues, denoted as $\{\lambda_i\}$

Chapter 2

Background

2.1 Hessian-vector product

The Hessian-vector product is an operation that computes the product of the Hessian matrix $\nabla^2 f(\boldsymbol{\theta})$ of a function f with a vector $\boldsymbol{v}$, without explicitly forming $\nabla^2 f(\boldsymbol{\theta})$. It is defined as the directional derivative of the gradient $\nabla f(\boldsymbol{\theta})$ in the direction $\boldsymbol{v}$, which can be expressed as

$$\nabla^2 f(\boldsymbol{\theta})\boldsymbol{v} = \lim_{\varepsilon \to 0} \frac{1}{\varepsilon}[\nabla f(\boldsymbol{\theta} + \varepsilon \boldsymbol{v}) - \nabla f(\boldsymbol{\theta})] = \nabla[\langle \nabla f(.), \boldsymbol{v} \rangle](\boldsymbol{\theta}) \tag{2.1}$$

In addition, the Hessian-vector product has a computational complexity of the same order of magnitude as the complexity of computing a gradient. Hence, it can be efficiently performed in modern automatic differentiation frameworks as Jax [4] and Pytorch [5], as described and implemented in [6] and [2].

2.2 Power method

The power method is a simple iterative algorithm for estimating the largest eigenvalue of a matrix $\mathbf{A}$ in absolute value and its corresponding eigenvector. Its formulation is given in Algorithm (1).

Algorithm 1 Power method

1: **Input:** A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and an initial vector $\mathbf{x}_0 \in \mathbb{R}^n$ with $\|\mathbf{x}^{(0)}\| = 1$
2: **Output:** An approximation of the largest eigenvalue and its corresponding eigenvector
3: $i = 0$
4: **repeat**
5: $\quad i = i + 1$
6: $\quad \mathbf{y}_i = \mathbf{A}\mathbf{x}_{i-1}$
7: $\quad \mathbf{x}_i = \frac{\mathbf{y}_i}{\|\mathbf{y}_i\|_2}$
8: $\quad \lambda_i = \mathbf{x}_i^T \mathbf{A}\mathbf{x}_i$
9: **until** a convergence criterion is satisfied

The algorithm begins with an initial vector $\mathbf{x}_0$ of unit norm, which serves as an approximation of the dominant eigenvector of $\mathbf{A}$. At each iteration, the matrix $\mathbf{A}$ is applied to the current approximation to compute $\mathbf{y}_i = \mathbf{A}\mathbf{x}_{i-1}$. The resulting vector is then normalized to maintain unit

norm, yielding $\mathbf{x}_i = \mathbf{y}_i / \|\mathbf{y}_i\|_2$. An approximation of the largest eigenvalue is computed at each iteration using the Rayleigh quotient $\lambda_i = \mathbf{x}_i^T \mathbf{A} \mathbf{x}_i$. The iterations continue until a predefined convergence criterion is satisfied. Typically, convergence is declared when the change in λ_i or $\mathbf{x}_i$ between successive iterations is below a specified tolerance.

The power method only requires matrix-vector multiplications with $\mathbf{A}$, which can be provided by a "black-box" function that outputs $\mathbf{A}\mathbf{x}_i$ for a given $\mathbf{x}_i$. This means that we do not need to compute $\mathbf{A}$ explicitly; instead, it suffices to have a mechanism to compute the matrix-vector product with $\mathbf{A}$. When interested in finding the largest eigenvalue of the Hessian matrix, this black-box function corresponds to the Hessian-vector product introduced in Section 2.1.

While simple and widely used, the power method suffers from two significant drawbacks: slow convergence and inefficient use of computations. The convergence rate is dictated by the ratio $|\lambda_2/\lambda_1|^k$, where λ_1 and λ_2 are the largest and second largest eigenvalues in magnitude, respectively, and k represents the number of iterations. If the gap between these two eigenvalues is small, the method requires many iterations to converge. Additionally, this method does not make efficient use of prior computations. Although each iteration explores the directions spanned by $\mathbf{A}\mathbf{x}_0, \mathbf{A}^2\mathbf{x}_0, \ldots, \mathbf{A}^k\mathbf{x}_0$, the power method discards this information and relies only on the direction of $\mathbf{A}^k\mathbf{x}_0$.

2.3 Lanczos algorithm

Motivated by the shortcomings of the power method described in Section 2.2, in this section we will present the Lanczos algorithm [7], following the presentations given in [8] and [9]. The Lanczos algorithm is an iterative method designed to compute the largest and smallest eigenvalues of a large, sparse, symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$. The extremal eigenvalues are estimated by generating a sequence of tridiagonal matrices $\{\mathbf{T}_m\}$, where $\mathbf{T}_m = \mathbf{V}_m^T \mathbf{A} \mathbf{V}_m$, and $\mathbf{V}_m$ is a matrix with orthonormal columns. With each iteration, the extremal eigenvalues of $\mathbf{T}_m$ provide progressively better approximations to the extremal eigenvalues of $\mathbf{A}$. One formulation of the algorithm, referred to as *Lanczos with full reorthogonalization* is given in Algorithm (2).

The Lanczos algorithm begins by initializing the first Lanczos vector, $\mathbf{v}_1$, as a random vector sampled from a normal Gaussian distribution. The algorithm then iteratively constructs a sequence of orthonormal vectors, called Lanczos vectors, which span a Krylov subspace of increasing dimension. At each iteration, the algorithm computes a new vector $\mathbf{w}$ by multiplying the current Lanczos vector $\mathbf{v}_i$ with the matrix $\mathbf{A}$. This matrix-vector multiplication is the most computationally intensive step of the algorithm. However, the matrix $\mathbf{A}$ does not need to be explicitly formed; it is sufficient to have a method for efficiently computing the product $\mathbf{A}\mathbf{v}_i$. As discussed earlier in the context of the power method, if $\mathbf{A}$ is a Hessian matrix, this product can be computed using a Hessian-vector product.

The diagonal element α_i of the tridiagonal matrix $\mathbf{T}_m$ is computed as $\alpha_i = \mathbf{v}_i^T \mathbf{w}$, and a contribution of $\mathbf{v}_i$ is removed from $\mathbf{w}$ by setting

$$\mathbf{w} = \mathbf{w} - \alpha_i \mathbf{v}_i. \tag{2.2}$$

To maintain orthogonality between the Lanczos vectors, a reorthogonalization step is performed using the Gram-Schmidt process. In this step, the new vector $\mathbf{w}$ is orthogonalized with respect

Algorithm 2 Lanczos with Full Reorthogonalization

Input: A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and number of iterations m
Output: $\mathbf{V}_m$, $\mathbf{T}_m$
for $i = 1, \ldots, m$ **do**
    if $i == 1$ **then**
        sample $\mathbf{v}_1 \sim \mathcal{N}(0, I)$
        $\mathbf{v}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|_2}$
        $\mathbf{w} = \mathbf{A}\mathbf{v}_1$
    else
        $\mathbf{w} = \mathbf{A}\mathbf{v}_i - \beta_{i-1}\mathbf{v}_{i-1}$
    end if
    $\alpha_i = \mathbf{v}_i^T \mathbf{w}$
    $\mathbf{w} = \mathbf{w} - \alpha_i \mathbf{v}_i$
    $\mathbf{w} = \mathbf{w} - \sum_{j=1}^{i} (\mathbf{w}^T \mathbf{v}_j)\mathbf{v}_j$
    $\beta_i = \|\mathbf{w}\|_2$
    $\mathbf{v}_{i+1} = \frac{\mathbf{w}}{\beta_i}$
end for

$$\mathbf{V}_m = \left[\mathbf{v}_1 \middle| \mathbf{v}_2 \middle| \ldots \middle| \mathbf{v}_m \right]$$

$$\mathbf{T}_m = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ldots & 0 \\ 0 & \beta_2 & \alpha_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & \beta_{m-1} & \alpha_m \end{bmatrix}$$

return $\mathbf{V}_m$, $\mathbf{T}_m$

to all previously computed Lanczos vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_i$ by subtracting their projections:

$$\mathbf{w} = \mathbf{w} - \sum_{j=1}^{i} (\mathbf{w}^T \mathbf{v}_j)\mathbf{v}_j. \tag{2.3}$$

This ensures that $\mathbf{w}$ remains orthogonal to the span of all preceding Lanczos vectors, preserving the theoretical properties of the algorithm even in the presence of numerical round-off errors. Once $\mathbf{w}$ is fully reorthogonalized, its norm is computed as $\beta_i = \|\mathbf{w}\|_2$, and the next Lanczos vector is defined as $\mathbf{v}_{i+1} = \mathbf{w}/\beta_i$. The entries α_i and β_i are used to populate the diagonal and off-diagonal entries of the tridiagonal matrix $\mathbf{T}_m$.

As the iterations proceed, the tridiagonal matrix $\mathbf{T}_m$ grows, capturing more of the spectral properties of $\mathbf{A}$. The eigenvalues of $\mathbf{T}_m$ provide increasingly accurate approximations to the extremal eigenvalues of $\mathbf{A}$. After m iterations, the algorithm terminates, returning the tridiagonal matrix $\mathbf{T}_m$ and the matrix of Lanczos vectors $\mathbf{V}_m$. To estimate the spectrum of $\mathbf{A}$, a final step is performed: the eigendecomposition of the tridiagonal matrix $\mathbf{T}_m$.

Similarly to Algorithm (2), we consider two additional variants of the Lanczos algorithm: *Lanczos with double full reorthogonalization* and *Lanczos with no reorthogonalization*. In the double full reorthogonalization variant, the Gram-Schmidt reorthogonalization step is performed

twice consecutively, as opposed to just once in Algorithm (2). In the no reorthogonalization variant, the reorthogonalization step is completely omitted. Detailed formulations of these two Lanczos variants are given in Appendix (A.1).

2.3.1 Loss of orthogonality

The Lanczos algorithm ensures that, under exact arithmetic, the vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_m$ form an orthonormal basis, and the obtained eigenvalues and eigenvectors are good approximations to those of the original matrix. However, in practice the operations are performed in floating-point arithmetic where orthogonality degrades quickly. This results in new Lanczos vectors becoming linearly dependent on the previously constructed vector set.

Fortunately, the loss of orthogonality introduces predictable behavior. The price to pay is the appearance of multiple copies of converged eigenvalues, referred to as *ghost eigenvalues*. Instead of $\mathbf{T}_k$ having a single eigenvalue close to $\lambda_i(\mathbf{A})$, it may have several eigenvalues nearly equal to $\lambda_i(\mathbf{A})$. This is acceptable if eigenvalue multiplicities are not important and delayed convergence of interior eigenvalues is tolerable. For accurate multiplicities of the eigenvalues, orthogonality must be preserved, which increases the computational cost.

2.3.2 Convergence behavior

We already described that the Lanczos algorithm progressively constructs a sequence of tridiagonal matrices $\{\mathbf{T}_k\}$. We should note that $\mathbf{T}_k$ is a principal submatrix of $\mathbf{T}_{k+1}$, which is obtained by deleting both the $k+1$-th row and column. Hence, we can apply Cauchy interlace theorem, which tells us that the eigenvalues of $\mathbf{T}_k$ interlace with the eigenvalues of $\mathbf{T}_{k+1}$, i.e.

$$\lambda_i(\mathbf{T}_{k+1}) \geq \lambda_i(\mathbf{T}_k) \geq \lambda_{i+1}(\mathbf{T}_{k+1}) \geq \lambda_{i+1}(\mathbf{T}_k) \tag{2.4}$$

In other words $\lambda_i(\mathbf{T}_k)$ increases monotonically with k for any fixed i, not just $i = 1$ (the largest eigenvalue). The same is true for the smallest eigenvalues. Due to this behaviour the largest eigenvalue $\lambda_1(\mathbf{T}_k)$ of $\mathbf{T}_k$ should converge to the largest eigenvalue of $\mathbf{A}$, $\lambda_1(\mathbf{A})$, where $\mathbf{A}$ is the input matrix to the Lanczos algorithm. Similarly, the i-th largest eigenvalue $\lambda_i(\mathbf{T}_k)$ of $\mathbf{T}_k$ must increase monotonically and converge to the i-th largest eigenvalue of $\mathbf{A}$. The i-th smallest eigenvalue $\lambda_{k+1-i}(\mathbf{T}_k)$ must similarly decrease monotonically and converge to the i-th smallest eigenvalue $\lambda_{n+1-i}(\mathbf{A})$ of $\mathbf{A}$.

To summarize this discussion, extreme eigenvalues, i.e., the largest and smallest ones, converge first, and the interior eigenvalues converge last. Furthermore, convergence is monotonic, with the i-th largest (smallest) eigenvalue of $\mathbf{T}_k$ increasing (decreasing) to the i-th largest (smallest) eigenvalue of $\mathbf{A}$, provided that the Lanczos algorithm doesn't stop prematurely with some $\beta_k = 0$.

Chapter 3

Lanczos algorithm on synthetic matrices

Though our ultimate goal is to apply the Lanczos algorithm to estimate the spectrum, or rather the leading positive and negative eigenvalues, of neural network architectures, we begin by implementing and running different variants of the algorithm on synthetic matrices. By synthetic matrix, we refer to a matrix constructed from a predefined list of eigenvalues. In this section, we briefly outline the experimental setup and report the obtained results. We do not expect to uncover novel insights, but instead aim to verify theoretical claims (e.g., convergence properties and loss of orthogonality) in practice across different sets of eigenvalues.

We investigate the behavior of Lanczos with full reorthogonalization, Lanczos with double full reorthogonalization, and Lanczos without reorthogonalization. Specifically, we compare the runtimes of these variants for different matrix sizes, and assess their performance in terms of L_1 and L_2 distances to the ground-truth eigenvalues. Additionally, we investigate the convergence behavior of the Lanczos algorithm, focusing on the number of correctly estimated extreme eigenvalues as a function of the number of iterations.

3.1 Introduction

We aim to apply three different variants of the Lanczos algorithm: with full reorthogonalization (Algorithm (2)), with double full reorthogonalization (where the Gram-Schmidt reorthogonalization process is performed twice consecutively), and without reorthogonalization (where the reorthogonalization step is omitted) on synthetic matrices of specified sizes. As explained before, the Lanczos algorithm takes as input a real symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and a number of iterations m and produces a tridiagonalization of $\mathbf{A}$, i.e. a matrix $\mathbf{T}_m = \mathbf{V}_m^T \mathbf{A} \mathbf{V}_m$. To obtain the full spectrum of $\mathbf{A}$, we will run the algorithm for $m = n$ iterations. Upon obtaining $\mathbf{T}_n$, to obtain the spectum of $\mathbf{A}$, we apply eigendecomposition on the tridiagonal matrix $\mathbf{T}_n$.

Figure (3.1) shows the groundtruth eigenvalues of a 100×100 matrix alongside the eigenvalue approximations obtained by the three variants of the Lanczos algorithm. The x-axis represents the eigenvalue index, ranging from 1 to 100, corresponding to the dimensionality of the problem. The y-axis represents the eigenvalue magnitude. From the results, we observe that Lanczos with full reorthogonalization and double full reorthogonalization both perform well. This is clear

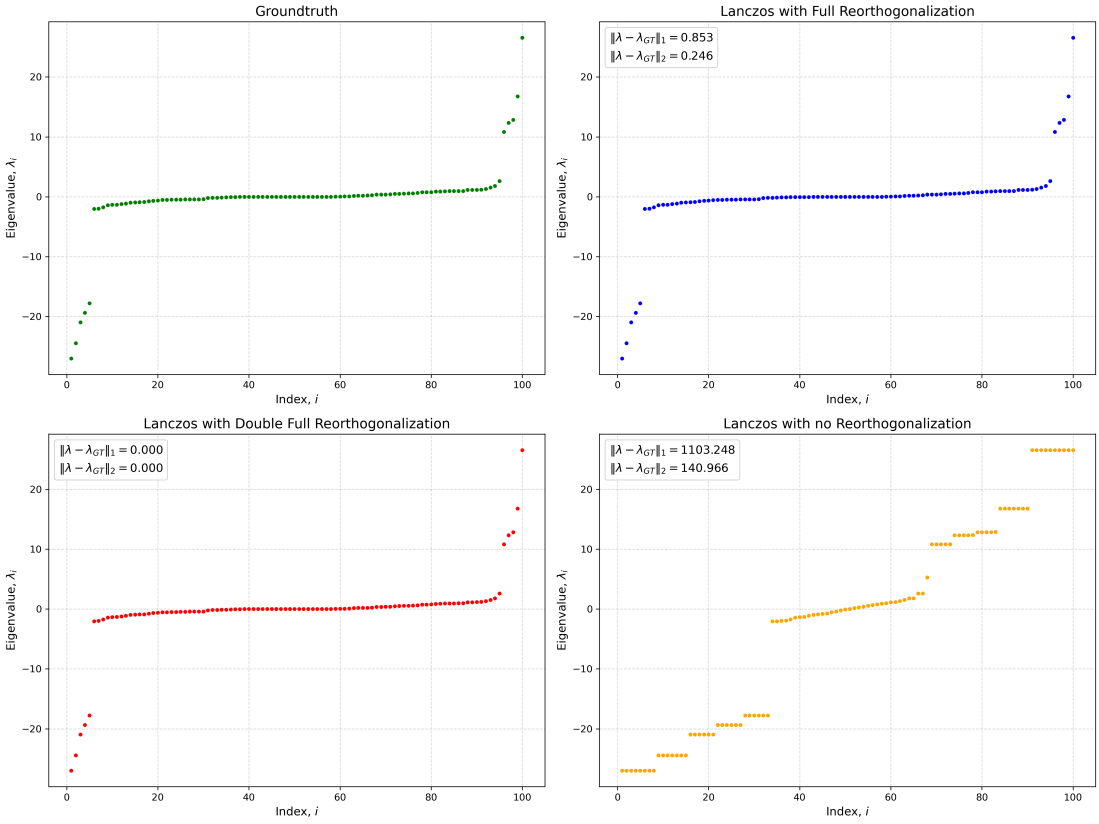

Figure 3.1: Comparison of the groundtruth eigenvalues with the eigenvalue approximations obtained by the three variants of the Lanczos algorithm. The groundtruth spectrum is created such that most eigenvalues are clustered around zero (sampled from a Gaussian distribution with mean 0 and variance 1) with 10 extreme positive and negative eigenvalues well-separated from the rest. For the approximated spectra, the L_1 and L_2 distances from the groundtruth spectrum are also reported in the legends.

both visually, as their spectra closely resemble the groundtruth spectrum, and quantitatively, as reflected in their low L_1 and L_2 distances. These methods manage to accurately capture both the clustering of eigenvalues near zero and the extreme eigenvalues. In contrast, Lanczos without reorthogonalization produces a spectrum where multiple eigenvalues have converged to the same value, as indicated by the horizontal lines in the plot. Despite this issue, the method successfully identifies the extreme eigenvalues, as seen in the plot. However, the L_1 and L_2 distances in this case are not meaningful, as they directly compare the vectors of eigenvalues.

3.2 Runtime analysis

In this section, we compare the runtimes of different variations of the Lanczos algorithm applied to matrices of various sizes. Specifically, we run Lanczos with full reorthogonalization, Lanczos with double full reorthogonalization, and Lanczos without reorthogonalization on matrices of sizes 10×10, 50×50, 100×100, 500×500, and 1000×1000. The matrices are generated such that

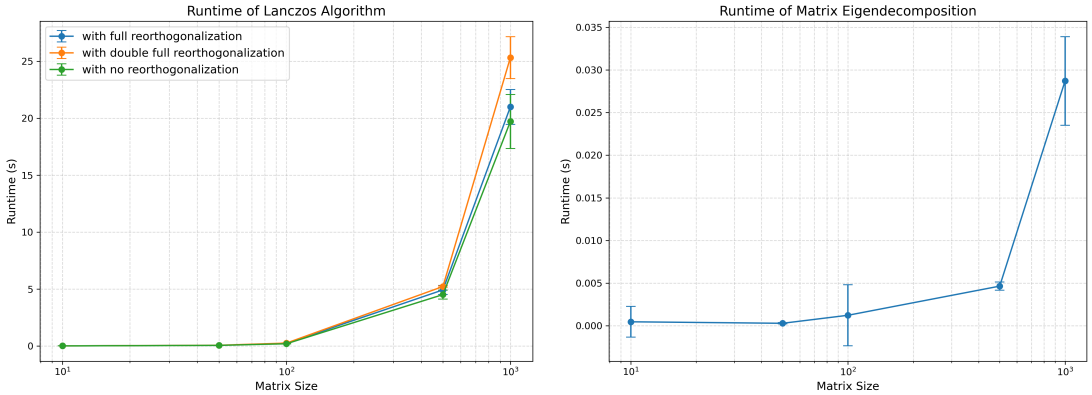

Figure 3.2: Runtimes of the Lanczos algorithm and matrix eigendecomposition as functions of matrix size. The left plot shows the runtime of the three variants of the Lanczos algorithm: with full reorthogonalization, with no reorthogonalization, and with double full reorthogonalization. The right plot shows the runtime of the eigendecomposition of the tridiagonal matrix $\mathbf{T}_n$ resulting from the Lanczos algorithm. Error bars indicate the standard deviation of runtimes across 10 repeated runs (left) and 30 repeated runs (right), while the points represent the means.

their eigenvalues are randomly sampled from a Gaussian distribution with mean 0 and variance 1.

The Lanczos algorithm described in Algorithm (2) terminates after m iterations, producing a matrix $\mathbf{V}_m$ whose columns are the Lanczos vectors and a tridiagonal matrix $\mathbf{T}_m$. As previously described, we run the algorithm for $m = n$ iterations, i.e., until it fully completes. Since our goal is to approximate the spectrum of $\mathbf{A}$ by obtaining the spectrum of $\mathbf{T}_n$, the final step involves performing an eigendecomposition of $\mathbf{T}_n$. In this section, we report the runtimes of the Lanczos algorithm itself and the runtimes of the final matrix eigendecomposition.

Figure (3.2) shows the runtimes for different stages of the spectrum computation, i.e. tridiagonalization by the Lanczos algorithm and eigendecompostition, as functions of matrix size. In the left plot, the x-axis represents the size of the square matrix $\mathbf{A}$, and the y-axis represents the runtime of the Lanczos algorithm in seconds. Three curves correspond to the three algorithm variations: with full reorthogonalization (blue), with double full reorthogonalization (orange), and with no reorthogonalization (green). The right plot shows the runtime of the eigendecomposition performed on the tridiagonal matrix $\mathbf{T}_n$ generated by the Lanczos algorithm.

From the left plot, we observe that for small matrices, all variants of the Lanczos algorithm exhibit negligible runtime differences. However, as the matrix size grows, the runtime increases, with double full reorthogonalization requiring the longest computation time and no reorthogonalization being the fastest. Full reorthogonalization lies in between the other two configurations. The right plot shows that the runtime of the eigendecomposition grows noticeably as the matrix size increases, though it remains relatively small compared to the runtime of the Lanczos algorithm itself for large matrices.

3.3 Convergence analysis

Figure (3.3) illustrates the convergence of eigenvalues computed by the Lanczos algorithm with full reorthogonalization over its iterations. The x-axis corresponds to the iteration number, which ranges from 1 to 100 (the dimension of the problem), while the y-axis represents the magnitude of the computed eigenvalues. Each tick (**x**) marks an eigenvalue, with the number of eigenvalues growing by one at every step due to the iterative nature of the algorithm. Specifically, at iteration m, m eigenvalues are computed, leading to a progressively more detailed approximation of the spectrum of the matrix. The color coding in the charts highlights the order of the eigenvalues as they converge. The largest and smallest eigenvalues at each iteration are shown in purple, the second largest and smallest in red, followed by green, and so on. The final vertical line, labeled "G" on the x-axis, represents the groundtruth spectrum, which is the same as in Figure (3.1).

From the plots, we observe the convergence behavior of the eigenvalues. As theoretically discussed in Section 2.3.2, the extreme eigenvalues (i.e., the largest and smallest) converge first. This behavior is visible in the plots, particularly for the extreme eigenvalues, although the full convergence of the interior eigenvalues is less apparent in this visualization. Figure (3.4) illustrates the evolution of eigenvalues computed by the Lanczos algorithm with no reorthogonalization over 100 iterations. The overall appearance of this plot resembles that of the full reorthogonalization case, particularly in how the extreme eigenvalues converge first. However, the lack of reorthogonalization introduces a key difference: the loss of orthogonality among the Lanczos vectors. This results in previously converged eigenvalues "jumping" and converging again to larger or smaller eigenvalues, leading to duplicates.

This analysis poses a natural follow-up question: *How does the number of iterations relate to the number of correctly estimated eigenvalues?* In other words, can we express the number of correctly estimated eigenvalues as a function of the number of iterations of the Lanczos algorithm? Additionally, it would be interesting to observe how the distribution of eigenvalues affects this result, i.e., does the relationship change for different eigenvalue configurations? To answer this question, we examine four different eigenvalue distributions:

- **Normally distributed (Gaussian) spectrum:** The eigenvalues follow a standard Gaussian distribution with mean 0 and variance 1. An example of this distribution is given in Figure (A.1).
- **Uniformly distributed spectrum:** The eigenvalues are uniformly distributed in a specified range $[a, b]$. An example of this distribution is given in Figure (A.2)
- **Banded spectrum:** The spectrum is composed of a few well-separated Gaussian distributions, forming distinct clusters or bands of eigenvalues. An example of this distribution is given in Figure (A.3)
- **Custom spectrum:** This spectrum resembles the spectrum observed in Figure (3.1). Most of the eigenvalues follow a Gaussian distribution with mean 0 and variance 1, while a small fraction are outliers well-separated from the rest. Additionally, many eigenvalues are intentionally set to 0. This distribution was chosen because it closely resembles the eigenvalue distribution of the Hessian matrix in neural networks.

We focus on matrices of size 1000×1000 and run the Lanczos algorithm for 10, 25, 50, 100, 250, 500, and 1000 iterations, with the results shown in Figure (3.5) for Lanczos with full reorthogonalization and Lanczos with no reorthogonalization. In both plots, the x-axis represents the number of iterations of the algorithm, plotted on a logarithmic scale, while the y-axis

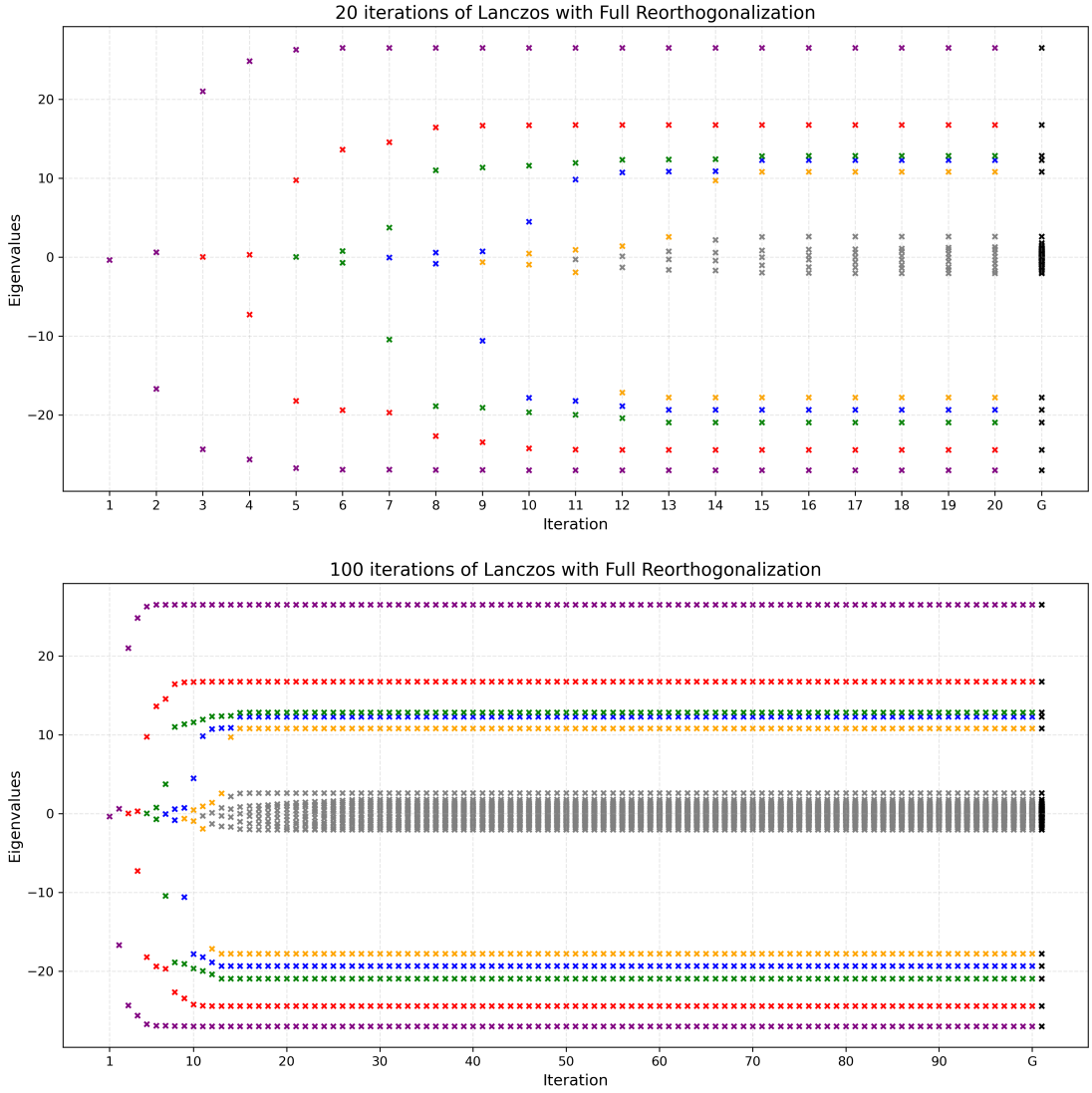

Figure 3.3: Eigenvalues found during each iteration of the Lanczos algorithm with full reorthogonalization. The top plot shows the eigenvalues computed over the first 20 iterations, while the bottom plot extends this to 100 iterations. Each tick ($\mathbf{x}$) represents an eigenvalue identified at a specific iteration. The final vertical line, labeled as "G," represents the groundtruth spectrum, which matches the spectrum shown in Figure (3.1).

shows the number of correctly estimated eigenvalues, also on a logarithmic scale. The number of estimated eigenvalues is determined by counting the converged eigenvalues within a distance of $\varepsilon = 10^{-5}$ from the true eigenvalues. Each curve corresponds to one of the four eigenvalue distributions: Gaussian, Uniform, Banded, and Custom. Figure (3.6) provides an alternative visualization, comparing the performance of Lanczos with full reorthogonalization, Lanczos with double full reorthogonalization and Lanczos with no reorthogonalization for each eigenvalue dis-

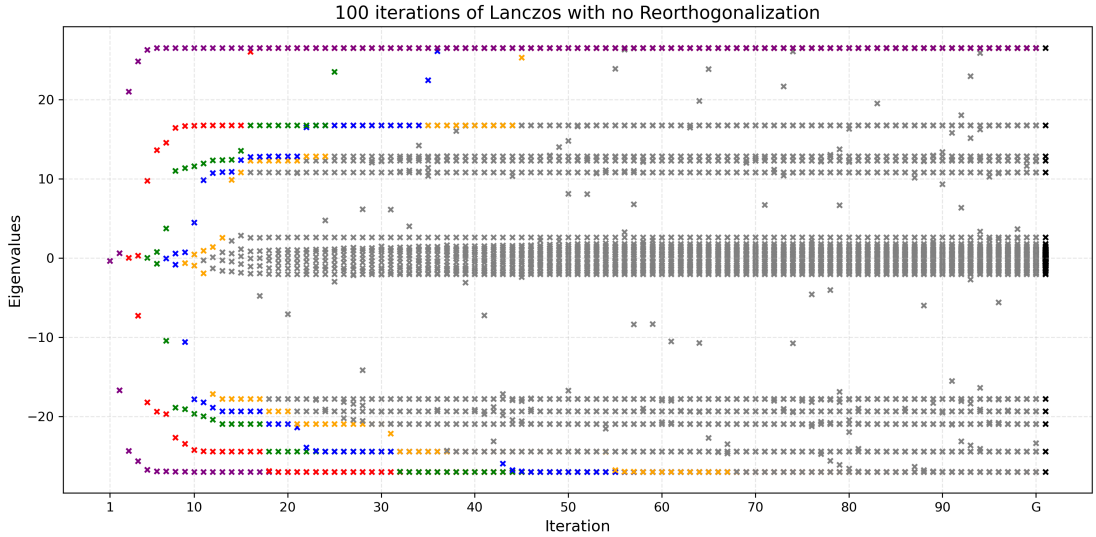

Figure 3.4: Eigenvalues computed during 100 iterations of the Lanczos algorithm with no reorthogonalization. The spectrum of the matrix used is the same as in Figure (3.1). Similar to Figure (3.3), the x-axis represents the iteration number, and the y-axis represents the eigenvalues computed at each step. The final vertical line labeled "G" represents the groundtruth spectrum.

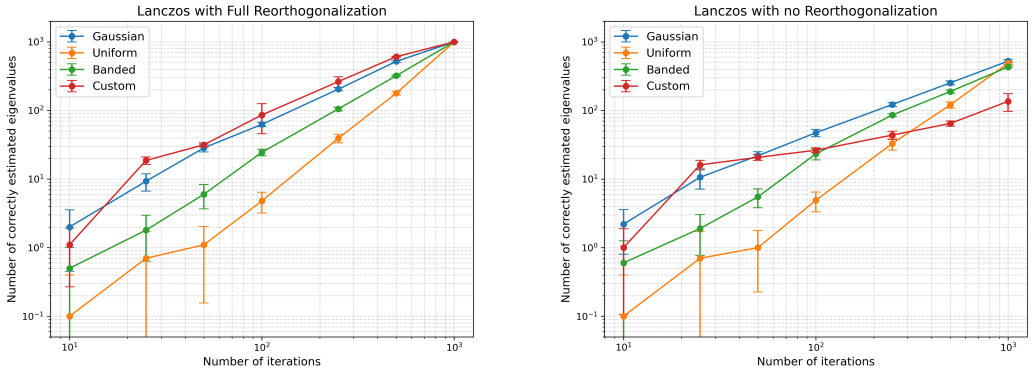

Figure 3.5: Comparison of the number of correctly estimated eigenvalues as a function of the number of iterations for Lanczos with full reorthogonalization (left) and Lanczos with no reorthogonalization (right). The results are shown for four eigenvalue distributions: Gaussian, Uniform, Banded, and Custom. The x-axis represents the number of iterations (logarithmic scale), and the y-axis shows the number of correctly estimated eigenvalues (logarithmic scale). Error bars represent the variability across 10 runs.

tribution.

From both Figure (3.5) and Figure (3.6), we observe that Lanczos with full reorthogonalization and double full reorthogonalization successfully obtain all eigenvalues when run to 1000

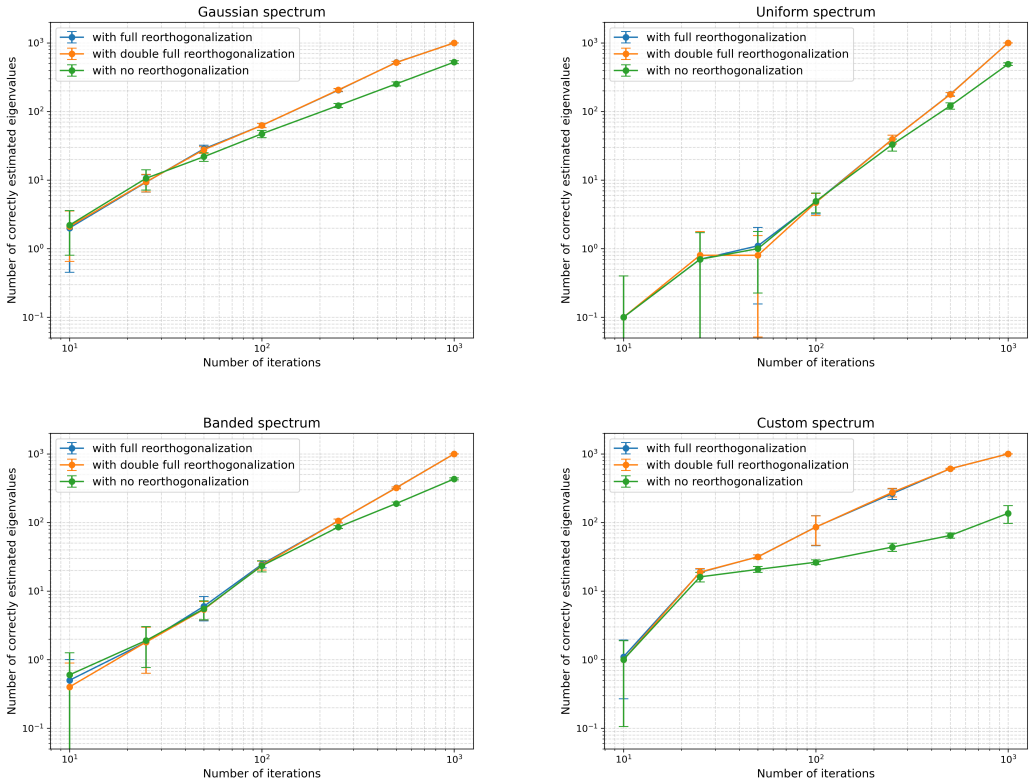

Figure 3.6: Number of correctly estimated eigenvalues as a function of the number of iterations for each eigenvalue distribution: Gaussian (top left), Uniform (top right), Banded (bottom left), and Custom (bottom right). Results are shown for three Lanczos variants: with full reorthogonalization, double full reorthogonalization, and no reorthogonalization.

iterations (the dimensionality of the matrix) for all eigenvalue distributions. Furthermore, as shown in Figure (3.6), the curves for these two methods almost always overlap, which indicates that they have similar performance. In contrast, Lanczos without reorthogonalization does not obtain all eigenvalues for any eigenvalue distribution, even after 1000 iterations. In Figure (3.6), the curves of Lanczos without reorthogonalization are generally lower, though they occasionally approach or overlap those of Lanczos with full and double full reorthogonalization.

Additionally, Figure (3.5) shows the differences across eigenvalue distributions more clearly for Lanczos with full reorthogonalization and without reorthogonalization. While all eigenvalues are eventually found after 1000 iterations when using Lanczos with full reorthogonalization, certain distributions yield more eigenvalues with fewer iterations. Specifically, the custom and Gaussian spectra consistently show faster convergence, followed by the banded spectrum, and then the uniform spectrum. A similar trend is observed in the right plot for Lanczos without reorthogonalization; however, one exception is the custom spectrum, which performs worse, ultimately finding fewer eigenvalues than other configurations even with the full number of iterations.

Chapter 4

Lanczos algorithm on the Hessian of neural networks

Following the insights from Chapter 3, we are ready to apply the Lanczos algorithm and explore the spectrum of the Hessian matrix of neural network architectures during various stages of training.

4.1 Introduction

In this section we consider a fully connected neural network with one hidden layer with sigmoid activations. The input is a 784-dimensional vector, the hidden layer has 16 units and the output is a 10 dimensional vector. The total number of parametetes of this model is $12,730$. We trained this model on three different datasets: MNIST [10], FashionMNIST [11], and CIFAR-10 [12]. The details of these datasets are given below.

- **MNIST:** A dataset of 70,000 grayscale images of handwritten digits (10 classes). Each image is of size 28×28 pixels. The dataset is split into 60,000 training images and 10,000 test images.
- **FashionMNIST:** A dataset of 70,000 grayscale images of clothing items (10 classes). Each image is of size 28×28 pixels. The dataset is split into 60,000 training images and 10,000 test images.
- **CIFAR-10:** A dataset of 60,000 RGB images of objects (10 classes). Each image, originally of size 32×32 pixels, was converted to grayscale and resized to 28×28 pixels to match the input dimensions of MNIST and FashionMNIST. The dataset is split into 50,000 training images and 10,000 test images.

The details about the training configuration and results are given in Table (4.1).

Figure (4.1) shows the groundtruth spectra of the full Hessian (i.e. the Hessian computed with respect to the full training loss, i.e. using all samples from the training dataset) for the three models trained on MNIST, FashionMNIST, and CIFAR-10, both at initialization and after training. Computing these spectra was computationally intensive, as it involved constructing the full Hessian matrix using Hessian-vector products with one-hot vectors, followed by performing

Dataset	Batch Size	Epochs	Training Loss	Test Loss	Test Accuracy
MNIST	64	15	0.2021	0.2049	94.05%
FashionMNIST	64	20	0.3541	0.4042	85.66%
CIFAR-10	64	30	1.7793	1.8097	35.42%

Table 4.1: Training configuration and results for each dataset. The table reports the batch size, number of epochs, training loss, test loss, and test accuracy achieved on MNIST, FashionMNIST, and CIFAR-10 datasets.

the eigendecomposition of the resulting large matrix.

From the plots, we observe that the spectra of the models consist of a bulk of eigenvalues around 0 with a few separated eigenvalues. This aligns with the findings of Sagun et al. [13], who show that the spectrum is divided into two parts: a bulk concentrated around zero and approximately *"number of classes - 1"* outlier eigenvalues. In all the plots, we observe around 10 outlier eigenvalues, consistent with the number of classes in each dataset. Additionally, we see that all models at initialization have negative eigenvalues, whereas for the trained models, the negative part of the spectrum has diminished, with the largest negative eigenvalue being close to 0.

Figure (4.2) shows the Hessian spectra estimated using the Lanczos algorithm with full re-orthogonalization for the three models trained on MNIST, FashionMNIST, and CIFAR-10, both at initialization and after training. The Lanczos algorithm was run for 100 iterations, resulting in 100 estimated eigenvalues. Additionally, we show the 50 smallest and largest eigenvalues from the groundtruth spectra, as previously displayed in Figure (4.1).

As discussed before, the extreme eigenvalues (i.e. the largest and the smallest) converge first. Quantitatively, the number of eigenvalues converged to a precision of 10^{-5} is summarized in Table 4.2. For example, for the model at initialization on MNIST, the algorithm converged to 41 eigenvalues, while for the trained model on MNIST, it converged to 47 eigenvalues.

Dataset	Training Stage	Converged Eigenvalues
MNIST	Initialization	41
	After Training	47
FashionMNIST	Initialization	44
	After Training	57
CIFAR-10	Initialization	69
	After Training	54

Table 4.2: Number of eigenvalues converged to a precision of 10^{-5} for each dataset and training stage.

4.2 Batch size effect

Until this point, we were interested in obtaining the spectrum of the full Hessian, i.e., the Hessian of the loss with respect to all data points. However, in practice, models are trained with large

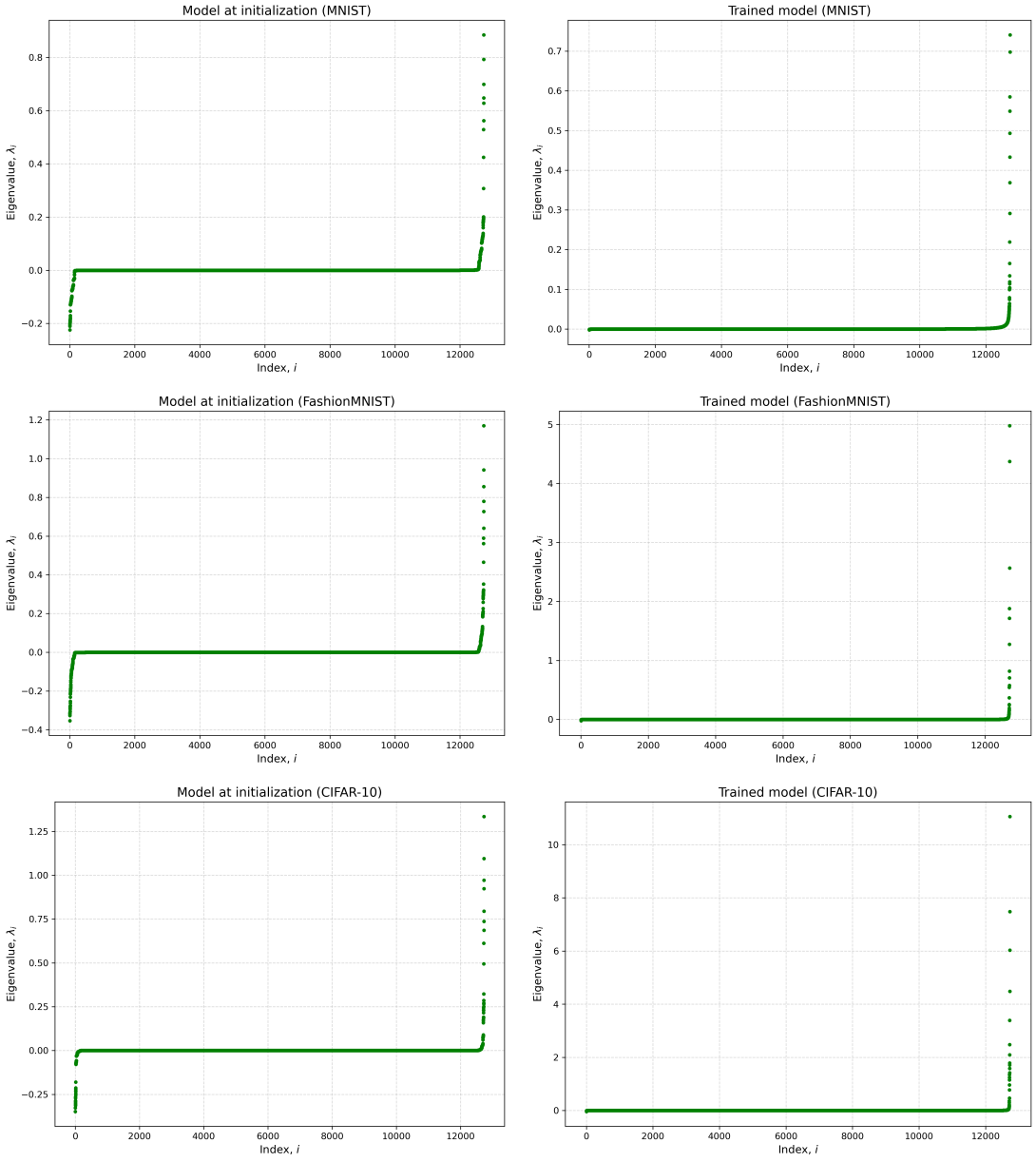

Figure 4.1: Groundtruth spectra of the full Hessian for models trained on MNIST, FashionMNIST, and CIFAR-10 at initialization and after training.

amounts of data. For example, the GPT-3 model was trained on 570GB of filtered text data [14], while Google's BERT was pre-trained on the BookCorpus (800M words) and English Wikipedia (2.5B words) datasets [15]. This immense volume of data makes computing the Hessian for the full loss computationally prohibitive. In this section, we will investigate the spectrum of the Hessian computed over a smaller batch of data. We will refer to the number of data points used to compute the Hessian as the *Hessian batch size*. Precisely, we aim to understand how the

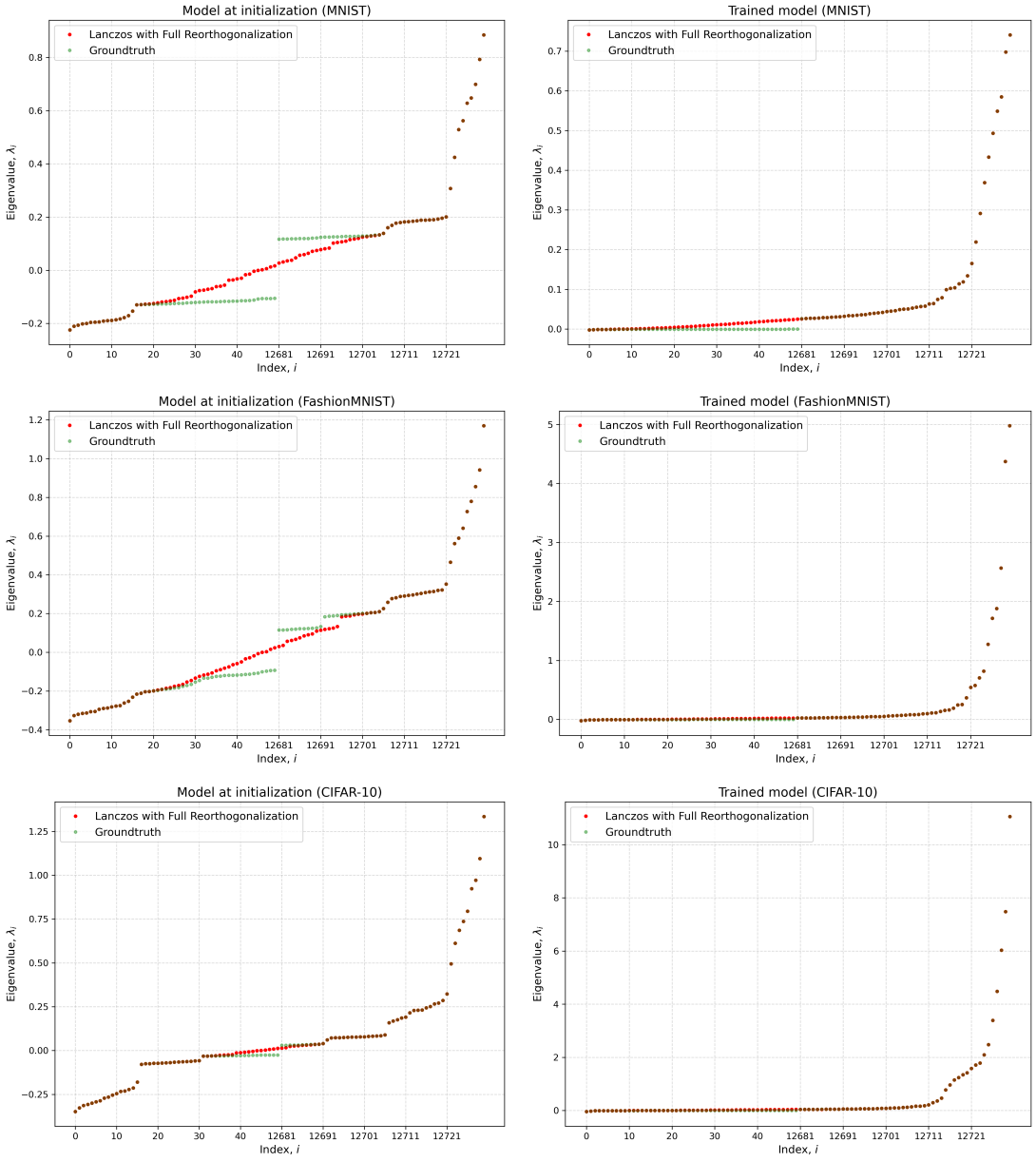

Figure 4.2: Comparison of the groundtruth spectra of the full Hessian with the spectra estimated using the Lanczos algorithm with full reorthogonalization for models trained on MNIST, FashionMNIST, and CIFAR-10 at initialization and after training.

approximations of the spectrum change with different sizes of the Hessian batch size.

Figure (4.3) shows the runtimes of 100 iterations of Lanczos with full reorthogonalization for various Hessian batch sizes, applied on the models discussed in Section 4.1. We observe that as we increase the size of the Hessian batch, the runtime increases exponentially.

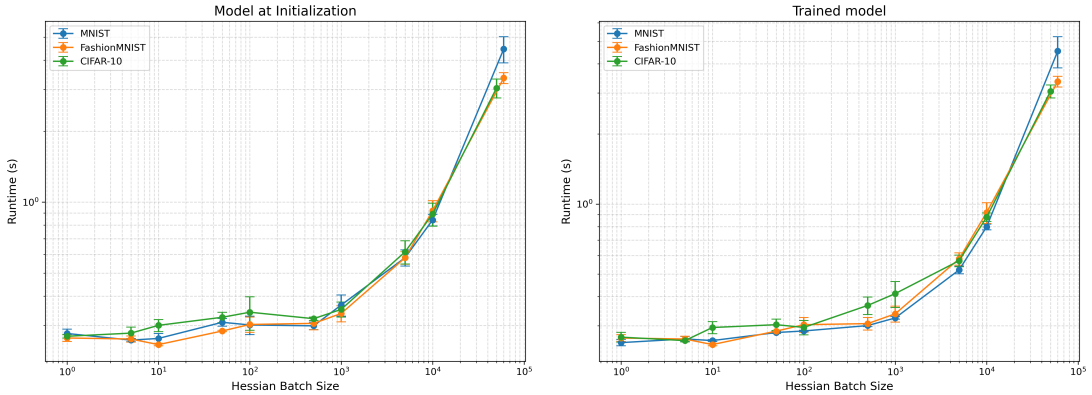

Figure 4.3: Runtimes of 100 iterations of the Lanczos algorithm with full reorthogonalization for different Hessian batch sizes, applied to models at initialization and after training on MNIST, FashionMNIST, and CIFAR-10. Error bars represent the variability in runtime across 10 runs.

Figure (4.4) shows the Hessian spectrum approximations obtained after 100 iterations of the Lanczos algorithm with full reorthogonalization for various Hessian batch sizes. The batch sizes include powers of 10 as well as the full batch, which corresponds to the Hessian of the loss on the entire training dataset. These approximations aim to capture the Hessian spectrum of the trained network on MNIST, as described in Section 4.1. Additional results for the networks trained on FashionMNIST and CIFAR-10 can be found in Appendix A.3.2. Each row of the figure corresponds to a different Hessian batch size and includes three plots, representing the results from three random batches of the specified size. Each plot also displays the gradient norm ∇f_i of the loss with respect to the batch, along with the L_1 and L_2 distances to the eigenvalues obtained for the full Hessian. Additionally, the spectrum for the full Hessian is overlaid on each plot with low opacity for comparison.

In general, the results suggest that as the batch size increases, the estimated spectra converge more closely to the spectrum of the full Hessian. We can observe this both visually from the plots, and by comparing the L_1 and L_2 distances. Smaller batch sizes, such as 1, 10, and 100, exhibit noticeable variations in the estimated spectra across different random batches. Furthermore, the smaller batches yield negative eigenvalues, which do not appear in the spectrum of the full Hessian.

4.3 NanoGPT

In this section, we present the results of applying the Lanczos method to a transformer architecture. Specifically, the method was used to obtain the Hessian spectrum of NanoGPT [16], a small-sized implementation of a GPT model [14]. The specific model we trained is character-level, meaning it predicts the next character in a sequence. The model was trained on the TinyShakespeare dataset [17], which consists of a corpus of Shakespearean text. The parameters of the model were configured as follows: it has 4 layers and 4 attention heads, with an embedding size of 128 and a block size of 64. This configuration results in a total of $804,096$ parameters. The training process used a batch size of 12 and ran for 2000 iterations.

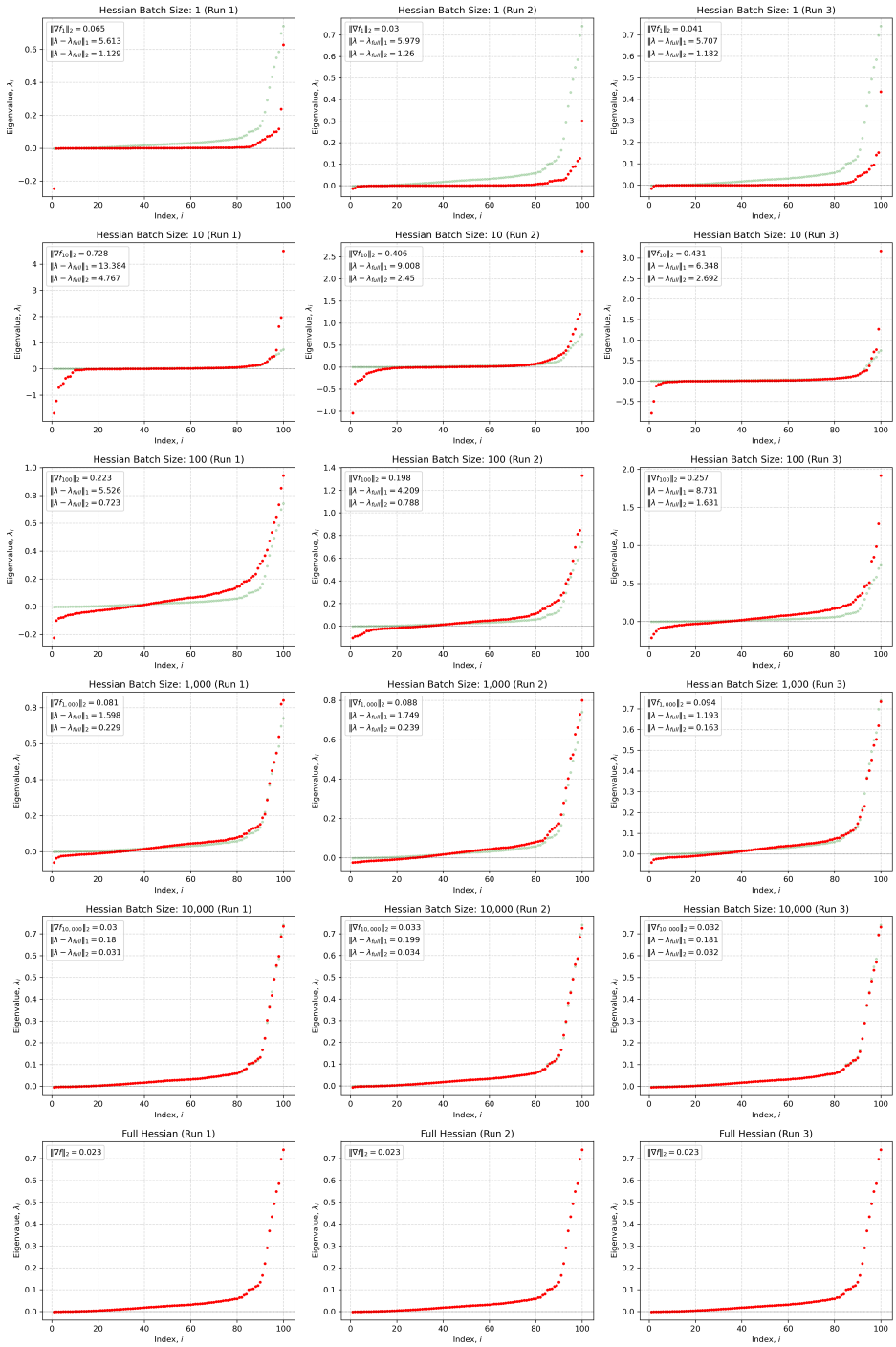

Figure 4.4: Eigenvalue distributions of the Hessian for varying Hessian batch sizes obtained by 100 iterations of Lanczos with full reorthogonalization. For each Hessian batch size, we show the results from three runs on randomly selected batches.

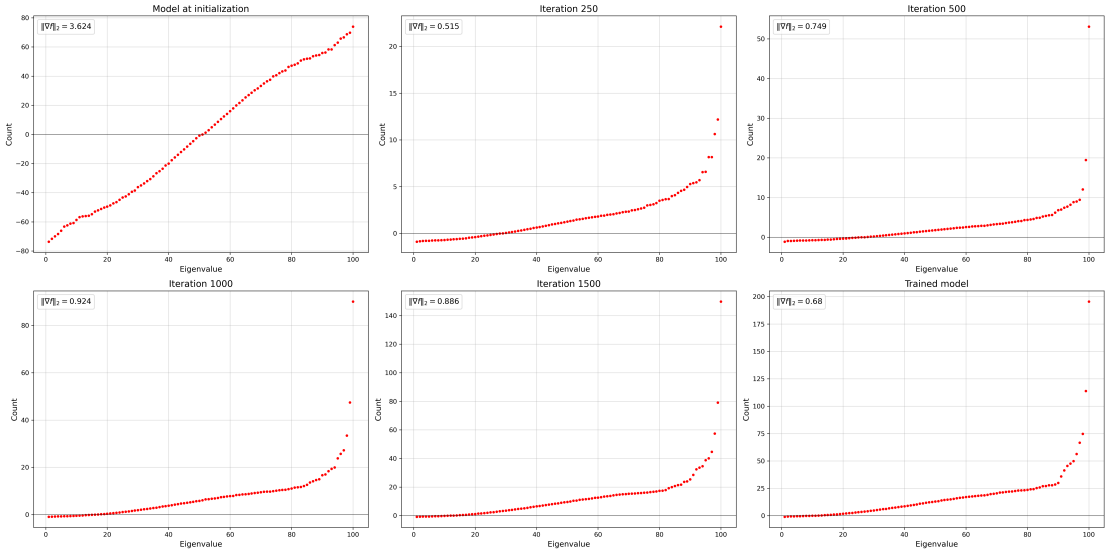

Figure 4.5: Approximations of the Hessian spectrum of the NanoGPT model during training. The six subplots correspond to different training stages: initialization, 250 iterations, 500 iterations, 1000 iterations, 1500 iterations, and the fully trained model.

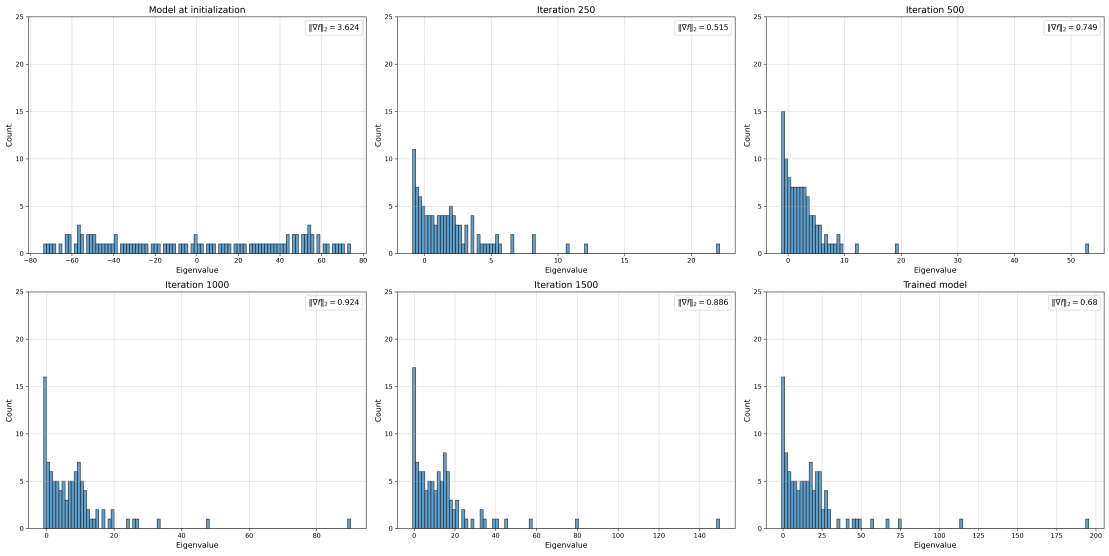

Figure 4.6: Alternative visualization of the Hessian spectra given in Figure (4.5)

Figure (4.5) and Figure (4.6) show the approximations of the spectrum of the NanoGPT model at various stages of training: initialization, after 250 iterations, 500 iterations, 1000 iterations, 1500 iterations, and at the final iteration (i.e. the trained model). This approximation was obtained using Lanczos 100 iterations with full reorthogonalization and a Hessian batch size of 400 (i.e. 400 blocks of size 64 = 25200 examples).

At initialization, the spectrum of the Hessian of nanoGPT is characterized by extreme positive and negative eigenvalues, with the largest positive eigenvalue around 80 and the most negative eigenvalue around -80. After 250 iterations, the spectrum changes significantly: the negative eigenvalues become much less negative, with the leading negative eigenvalue around -1. The positive eigenvalues also shrink, with the largest eigenvalue reduced to approximately 20. As training progresses, the negative eigenvalues continue to diminish, with the leading negative eigenvalue approaching zero. At the same time, the leading positive eigenvalue increases, reaching approximately 50 at 500 iterations, over 80 at 1000 iterations, above 140 at 1500 iterations, and around 200 by the end of training. Additionally, the separation between the leading positive eigenvalue and the second-largest eigenvalue becomes more pronounced.

Bibliography

[1] B. Ghorbani, S. Krishnan, and Y. Xiao, "An investigation into neural net optimization via hessian eigenvalue density," 2019. [Online]. Available: https://arxiv.org/abs/1901.10159

[2] Z. Yao, A. Gholami, K. Keutzer, and M. Mahoney, "Pyhessian: Neural networks through the lens of the hessian," 2020. [Online]. Available: https://arxiv.org/abs/1912.07145

[3] Y. Zhang, C. Chen, T. Ding, Z. Li, R. Sun, and Z.-Q. Luo, "Why transformers need adam: A hessian perspective," 2024. [Online]. Available: https://arxiv.org/abs/2402.16788

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang, "JAX: composable transformations of Python+NumPy programs," 2018. [Online]. Available: http://github.com/jax-ml/jax

[5] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019. [Online]. Available: https://arxiv.org/abs/1912.01703

[6] M. Dagréou, P. Ablin, S. Vaiter, and T. Moreau, "How to compute hessian-vector products?" in *ICLR Blogposts 2024*, 2024, https://iclr-blogposts.github.io/2024/blog/bench-hvp/. [Online]. Available: https://iclr-blogposts.github.io/2024/blog/bench-hvp/

[7] C. Lanczos, "An iteration method for the solution of the eigenvalue problem of linear differential and integral operators," *Journal of research of the National Bureau of Standards*, vol. 45, no. 4, p. 255, 10 1950. [Online]. Available: https://doi.org/10.6028/jres.045.026

[8] J. W. Demmel, *Applied Numerical Linear Algebra*, 1 1997. [Online]. Available: https://doi.org/10.1137/1.9781611971446

[9] G. H. Golub and C. F. Van Loan, *Matrix Computations - 4th Edition.* Philadelphia, PA: Johns Hopkins University Press, 2013. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9781421407944

[10] L. Deng, "The mnist database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[11] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1708.07747

[12] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.

[13] L. Sagun, L. Bottou, and Y. LeCun, "Eigenvalues of the hessian in deep learning: Singularity and beyond," 2017. [Online]. Available: https://arxiv.org/abs/1611.07476

[14] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[16] A. Karpathy, "NanoGPT," https://github.com/karpathy/nanoGPT, 2022.

[17] ——, "char-rnn," https://github.com/karpathy/char-rnn, 2015.

Appendix A

Appendix

A.1 Lanczos algorithm

In this section, we provide the detailed formulations of the Lanczos algorithm with no reorthogonalization, shown in Algorithm (3), and the Lanczos algorithm with double full reorthogonalization, shown in Algorithm (4).

Algorithm 3 Lanczos with no Reorthogonalization

Input: A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and number of iterations m
Output: $\mathbf{V}_m$, $\mathbf{T}_m$
for $i = 1, \ldots, m$ **do**
    if $i == 1$ **then**
        sample $\mathbf{v}_1 \sim \mathcal{N}(0, I)$
        $\mathbf{v}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|_2}$
        $\mathbf{w} = \mathbf{A}\mathbf{v}_1$
    else
        $\mathbf{w} = \mathbf{A}\mathbf{v}_i - \beta_{i-1}\mathbf{v}_{i-1}$
    end if
    $\alpha_i = \mathbf{v}_i^T \mathbf{w}$
    $\mathbf{w} = \mathbf{w} - \alpha_i \mathbf{v}_i$
    $\beta_i = \|\mathbf{w}\|_2$
    $\mathbf{v}_{i+1} = \frac{\mathbf{w}}{\beta_i}$
end for

$$\mathbf{V}_m = \left[\mathbf{v}_1 \middle| \mathbf{v}_2 \middle| \ldots \middle| \mathbf{v}_m\right]$$

$$\mathbf{T}_m = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ldots & 0 \\ 0 & \beta_2 & \alpha_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & \beta_{m-1} & \alpha_m \end{bmatrix}$$

return $\mathbf{V}_m$, $\mathbf{T}_m$

Algorithm 4 Lanczos with Double Full Reorthogonalization

Input: A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ and number of iterations m
Output: $\mathbf{V}_m$, $\mathbf{T}_m$
for $i = 1, \ldots, m$ **do**
    if $i == 1$ **then**
        sample $\mathbf{v}_1 \sim \mathcal{N}(0, I)$
        $\mathbf{v}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|_2}$
        $\mathbf{w} = \mathbf{A}\mathbf{v}_1$
    else
        $\mathbf{w} = \mathbf{A}\mathbf{v}_i - \beta_{i-1}\mathbf{v}_{i-1}$
    end if
    $\alpha_i = \mathbf{v}_i^T \mathbf{w}$
    $\mathbf{w} = \mathbf{w} - \alpha_i \mathbf{v}_i$
    $\mathbf{w} = \mathbf{w} - \sum_{j=1}^{i} (\mathbf{w}^T \mathbf{v}_j)\mathbf{v}_j$
    $\mathbf{w} = \mathbf{w} - \sum_{j=1}^{i} (\mathbf{w}^T \mathbf{v}_j)\mathbf{v}_j$ (double reorthogonalization)
    $\beta_i = \|\mathbf{w}\|_2$
    $\mathbf{v}_{i+1} = \frac{\mathbf{w}}{\beta_i}$
end for

$$\mathbf{V}_m = \left[\mathbf{v}_1 \middle| \mathbf{v}_2 \middle| \ldots \middle| \mathbf{v}_m \right]$$

$$\mathbf{T}_m = \begin{bmatrix} \alpha_1 & \beta_1 & 0 & \ldots & 0 \\ \beta_1 & \alpha_2 & \beta_2 & \ldots & 0 \\ 0 & \beta_2 & \alpha_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & \beta_{m-1} & \alpha_m \end{bmatrix}$$

return $\mathbf{V}_m$, $\mathbf{T}_m$

A.2 Lanczos on synthetic matrices

A.2.1 Normally distributed spectrum

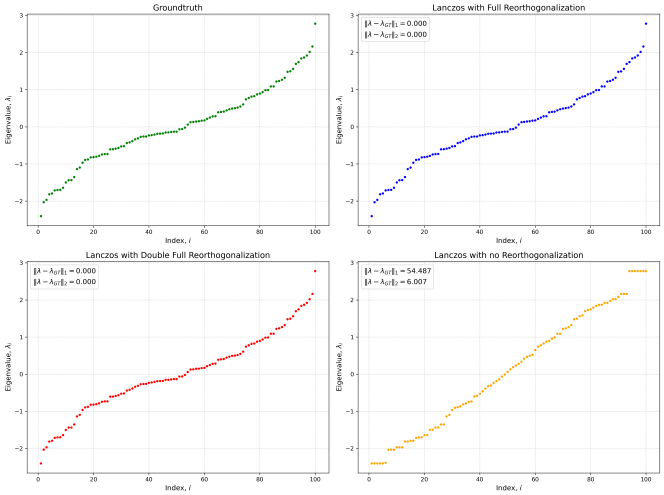

Figure A.1: Comparison of the groundtruth eigenvalues with the eigenvalue approximations obtained by the three variants of the Lanczos algorithm. The eigenvalues follow a standard Gaussian distribution with mean 0 and variance 1.

A.2.2 Uniformly distributed spectrum

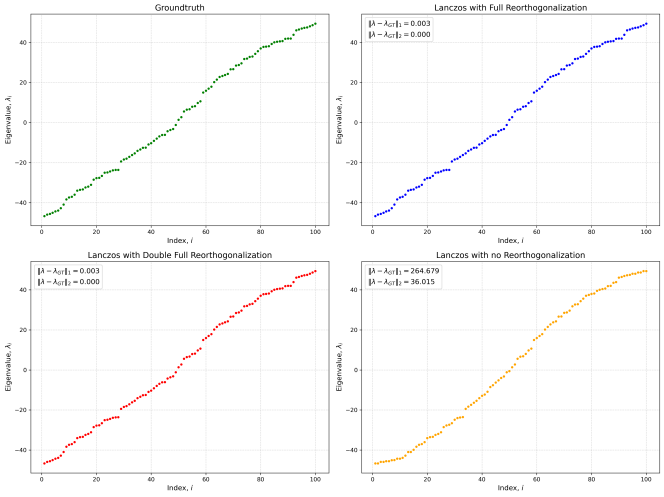

Figure A.2: Comparison of the groundtruth eigenvalues with the eigenvalue approximations obtained by the three variants of the Lanczos algorithm. The eigenvalues are uniformly distributed in a specified range $[-50, 50]$.

A.2.3 Banded spectrum

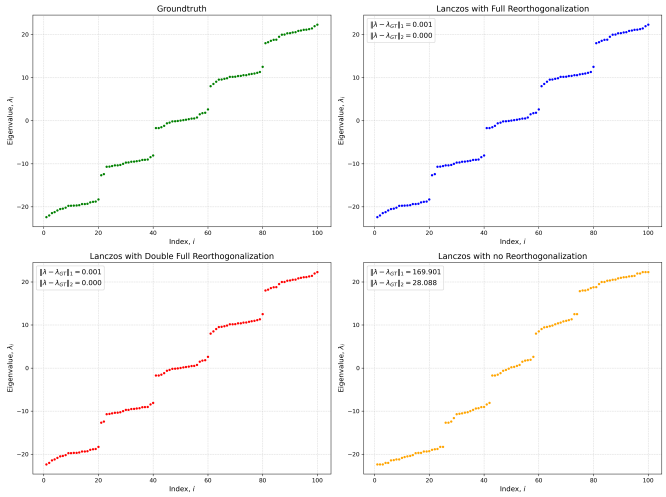

Figure A.3: Comparison of the groundtruth eigenvalues with the eigenvalue approximations obtained by the three variants of the Lanczos algorithm. The spectrum is composed of a few well-separated Gaussian distributions, forming distinct clusters or bands of eigenvalues.

A.3 Lanczos on neural networks

A.3.1 Hessian spectrum along the training trajectory

Figures (A.4), (A.5) and (A.6) show the evolution of the Hessian spectrum of the model described in Section 4.1 trained on MNIST, FashionMNIST and CIFAR-10, respectively. The Hessian spectra shown are estimated through 100 iterations of the Lanczos algorithm with full reorthogonalization.

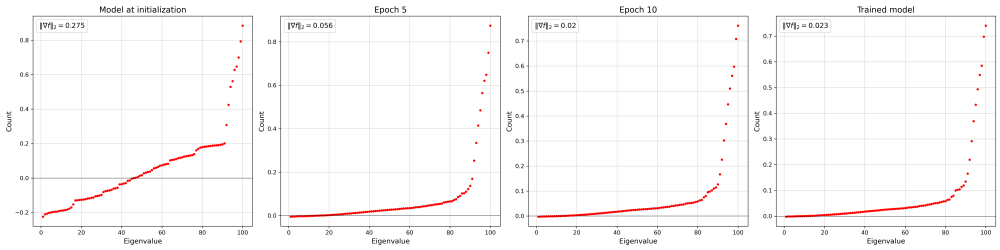

Figure A.4: Approximations of the Hessian spectrum for the model discussed in Section 4.1, trained on MNIST, evaluated throughout the training process. The four subplots correspond to different training stages: initialization, 5 epochs, 10 epochs and the fully trained model.

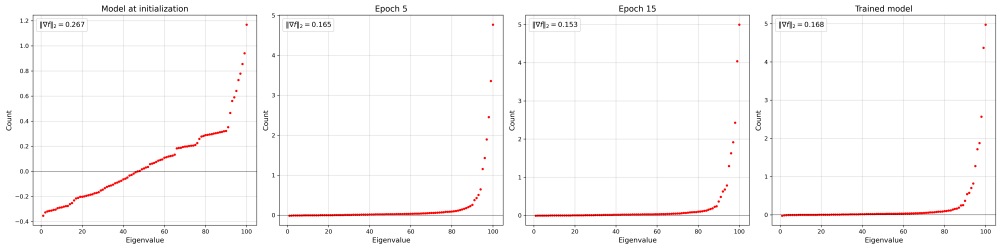

Figure A.5: Approximations of the Hessian spectrum for the model discussed in Section 4.1, trained on FashionMNIST, evaluated throughout the training process. The four subplots correspond to different training stages: initialization, 5 epochs, 15 epochs and the fully trained model.

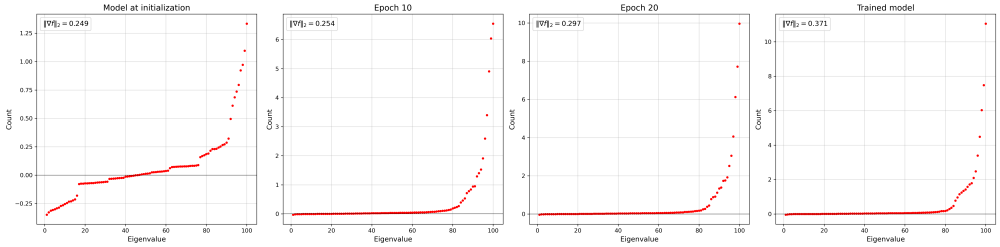

Figure A.6: Approximations of the Hessian spectrum for the model discussed in Section 4.1, trained on MNIST, evaluated throughout the training process. The four subplots correspond to different training stages: initialization, 10 epochs, 20 epochs and the fully trained model.

A.3.2 Batch size effect

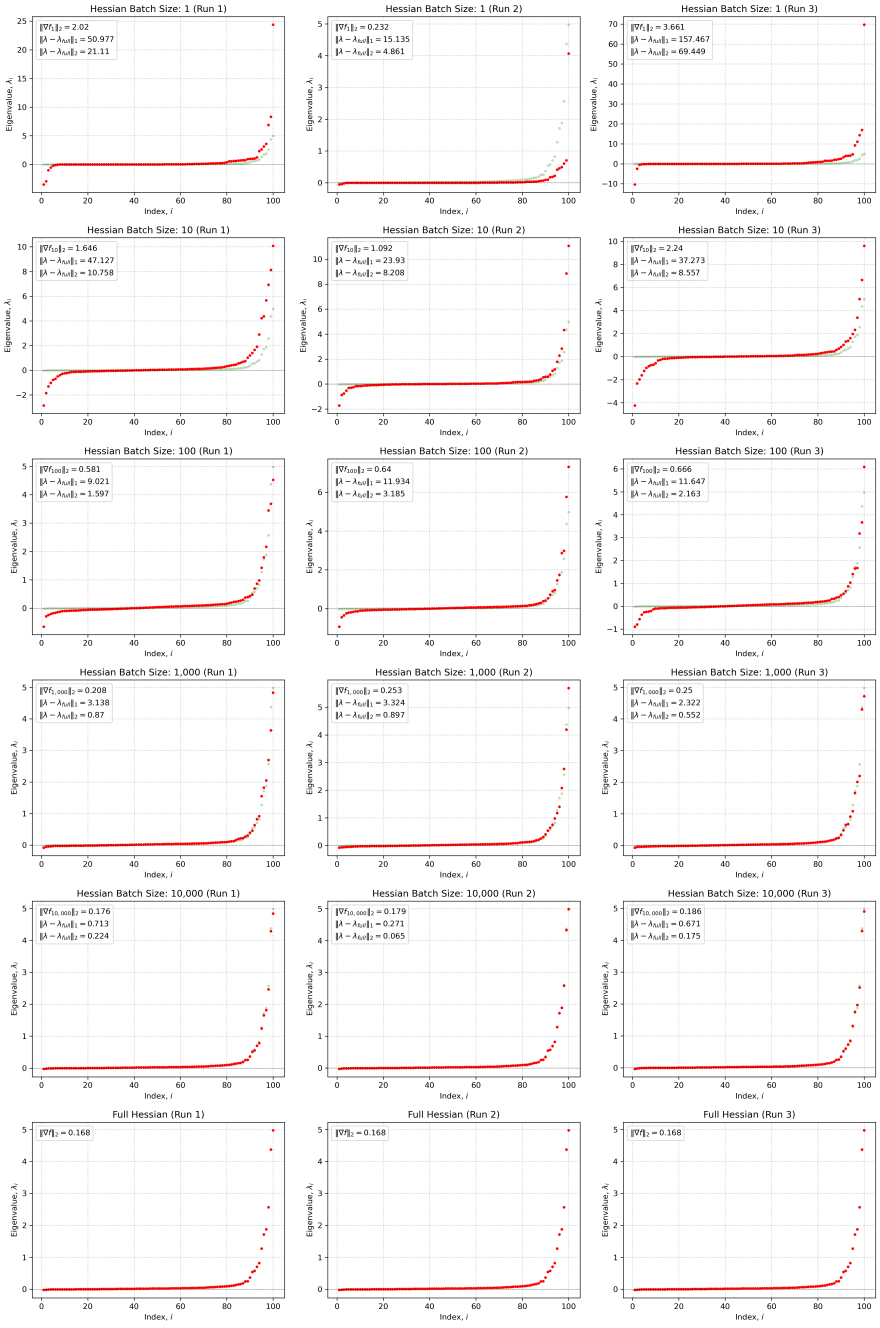

Figure A.7: Eigenvalue distributions of the Hessian for varying Hessian batch sizes obtained by 100 iterations of Lanczos with full reorthogonalization. For each Hessian batch size, we show the results from three runs on randomly selected batches. Model: Described in Section 4.1; Dataset: FashionMNIST

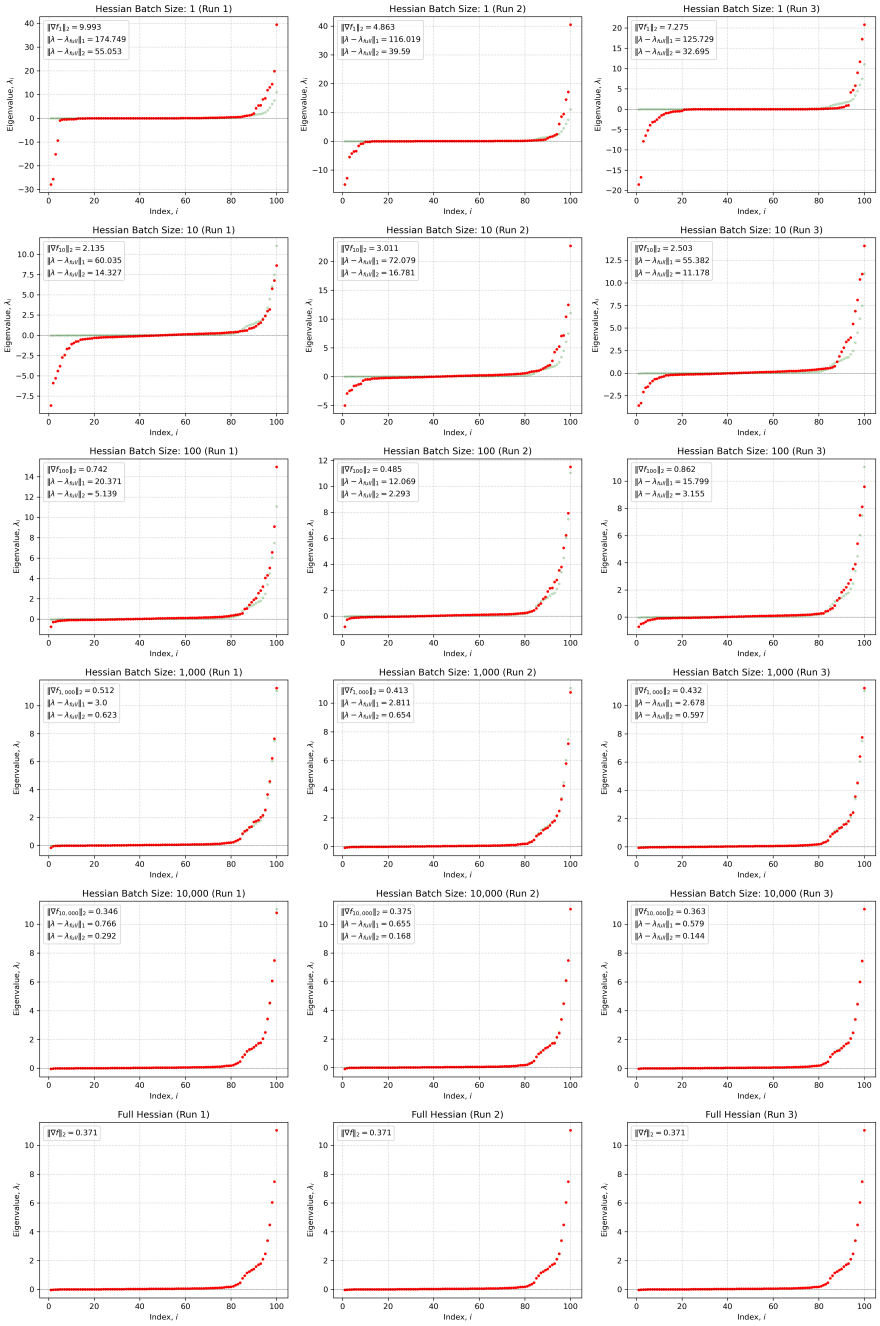

Figure A.8: Eigenvalue distributions of the Hessian for varying Hessian batch sizes obtained by 100 iterations of Lanczos with full reorthogonalization. For each Hessian batch size, we show the results from three runs on randomly selected batches. Model: Described in Section 4.1; Dataset: CIFAR-10