Empirical Analysis of the Hessian of Over-Parametrized Neural Networks

Levent Sagun
Courant Institute, NYU
sagun@cims.nyu.edu

Utku Evci
Computer Science Department, NYU
ue225@nyu.edu

V. Uğur Güney
ugurguney@gmail.com

Yann Dauphin
Facebook AI Research
yann@dauphin.io

Léon Bottou
Facebook AI Research
leonb@fb.com

Abstract

We study the properties of common loss surfaces through their Hessian matrix. In particular, in the context of deep learning, we empirically show that the spectrum of the Hessian is composed of two parts: (1) the bulk centered near zero, (2) and outliers away from the bulk. We present numerical evidence and mathematical justifications to the following conjectures laid out by Sagun et al. [2016]: Fixing data, increasing the number of parameters merely scales the bulk of the spectrum; fixing the dimension and changing the data (for instance adding more clusters or making the data less separable) only affects the outliers. We believe that our observations have striking implications for non-convex optimization in high dimensions. First, the *flatness* of such landscapes (which can be measured by the singularity of the Hessian) implies that classical notions of basins of attraction may be quite misleading. And that the discussion of wide/narrow basins may be in need of a new perspective around over-parametrization and redundancy that are able to create *large* connected components at the bottom of the landscape. Second, the dependence of small number of large eigenvalues to the data distribution can be linked to the spectrum of the covariance matrix of gradients of model outputs. With this in mind, we may reevaluate the connections within the data-architecture-algorithm framework of a model, hoping that it would shed light into the geometry of high-dimensional and non-convex spaces in modern applications. In particular, we present a case that links the two observations: a gradient based method appears to be first climbing uphill and then falling downhill between two points; whereas, in fact, they lie in the same basin.

1 Introduction

In this paper we study the geometry of the loss surface of supervised learning problems through the lens of their second order properties. To introduce the framework, suppose we are given data in the form of input-label pairs, $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N$ where $x \in \mathbb{R}^d$ and $y \in \mathbb{R}$ that are sampled i.i.d. from a possibly unknown distribution ν, a model that is parametrized by $w \in \mathbb{R}^M$; so that the number of examples is N and the number of parameters of the system is M. Suppose also that there is a predictor $f(w, x)$. The supervised learning process aims to solve for w so that $f(w, x) \approx y$. To make the '$\approx$' precise, we use a non-negative loss function that measures how close the predictor is to the true label, $\ell(f(w, x), y)$. We wish to find a parameter w^* such that:

$$w^* = \arg\min \mathcal{L}(w), \tag{1}$$

where

$$\mathcal{L}(w) := \frac{1}{N}\sum_{i=1}^{N} \ell(f(w, x^i), y^i). \tag{2}$$

In particular, one is curious about the relationship between $\mathcal{L}(w)$ and $\mathcal{L}'(w) := \int \ell d(\nu)$. By the law of large numbers, at a given point w, $\mathcal{L}_w \to \mathcal{L}'_w$ almost surely as $N \to \infty$ for fixed M. However in modern applications, especially in deep learning, the number of parameters M is comparable to the number of examples N (*if not much larger*). And the behaviour of the two quantities may be drastically different (for a recent analysis on provable estimates see [Mei et al., 2016]).

A classical algorithm to find w^* is gradient descent (GD), in which the optimization process is carried out using the gradient of $\mathcal{L}$. A new parameter is found iteratively by taking a step in the direction of the negative gradient whose size is scaled with a constant step size η that is chosen from line-search minimization. Two problems emerge: (1) Gradient computation can be expensive, (2) Line-search can be expensive. More involved algorithms, such as Newton type methods, make use of second order information [Nocedal and Wright, 2006]. Under sufficient regularity conditions we may observe: $\mathcal{L}(w + \Delta w) \approx \mathcal{L}(w) + \Delta w \nabla \mathcal{L}(w) + \Delta w^T \nabla^2 \mathcal{L}(w) \Delta w$. A third problem emerges beyond an even more expansive computational cost of the Hessian: (3) Most methods require the Hessian to be non-degenerate to a certain extent.

When the gradients are computationally expensive, one can alternatively use it's stochastic version (SGD) that replaces the above gradient with the gradient of averages of losses over *subsets* (such a subset will be called the *mini-batch*) of $\mathcal{D}$ (see [Bottou, 2010] for a classical reference). The benefit of SGD on real-life time limits is obvious, and GD may be impractical for practical purposes in many problems. In any case, the stochastic gradient can be seen as an approximation to the true gradient, and hence it is important to understand how the two directions are related in the parameter space. Therefore, the discussion around the geometry of the loss surface can be enlightening in the comparison of the two algorithms: Does SGD locate solutions of a different nature than GD? Do they follow different paths? If so, which one is better in terms of generalization performance?

For the second problem of expensive line-search, there are two classical solutions: using a small, constant step size, or scheduling the step size according to a certain rule. In practice, in the context of deep learning, the values for both approaches are determined heuristically, by trial and error. More involved optimal step size choices involve some kind of second order information that can be obtained from the Hessian of the loss function [Schaul et al., 2013]. From a computational point of view, obtaining the Hessian is extremely expensive, however obtaining some of its largest and smallest eigenvalues and eigenvectors are not that expensive. Is it enough to know only those eigenvalues and eigenvectors that are large in magnitude? How do they change through SGD?

For the third problem, let's look at the Hessian a little closer. A critical point is defined by w such that $||\nabla \mathcal{L}(w)|| = 0$ and the nature of it can be determined by looking at the *signs* of its Hessian matrix. If all eigenvalues are positive the point is called a local minimum, if r of them are negative and the rest are positive, then it is called a saddle point with index r. At the critical point, the eigenvectors indicate the directions in which the value of the function locally changes. Moreover the changes are proportional to the corresponding -signed- eigenvalue. Under sufficient regularity conditions, it is rather straightforward to show that gradient based methods converge to points where gradient is zero. Recently Lee et al. [2016] showed that they indeed converge to minimizers. However, a significant and untested assumption to establish these convergence results is that the loss is non-degenerate. A relaxation of the above convergence to the case of non-isolated critical points can be found in [Panageas and Piliouras, 2016]. What about the critical points of machine learning loss functions? Do they satisfy the non-degeneracy assumptions? If they don't, can we still apply the results of provable theorems to gain intuition?

1.1 A historical overview

One of the first instances of the comparison of GD and SGD in the context of neural networks dates back to late eighties and early nineties. Bottou [1991] points out that large eigenvalues of the Hessian of the loss can create the illusion of a local minima and GD can get stuck there, it further claims that the help of the inherent noise in SGD may help getting out of this obstacle. The origin of this observation is due to Bourrely [1989], as well as numerical justifications. However, given the computational limits of the time, these experiments relied on low-dimensional neural networks with few hidden units. The picture may be drastically different in higher dimensions. In fact, provable results in statistical physics tell us that, in certain real-valued non-convex functions, the local minima concentrate at an error level near that of the global minima. A theoretical review on this can be found in [Auffinger et al., 2013], while Sagun et al. [2014] and Ballard et al. [2017] provide an experimental simulation as well as a numerical study for neural networks. They notably find that high error local minima traps do not appear when the model is *over-parametrized*.

These concentration results can help explain why we find that the solutions attained by different optimizers like GD and SGD often have comparable training accuracies. However, while these methods find comparable solutions in terms of training error there is no guarantee they generalize equally. A recent work in this direction compares the generalization performance of *small batch* and *large batch* methods [Keskar et al., 2016]. They demonstrate that the *large batch* methods always generalize a little bit worse even when they have similar training accuracies. The paper further makes the observation that the basins found by *small batch* methods are wider, thereby contributing to the claim that wide basins, as opposed to narrow ones, generalize better [1].

The final part of the historical account is devoted to the observation of flatness of the landscape in neural networks and it's consequences through the lens of the Hessian. In early nineties, Hochreiter and Schmidhuber [1997] remarks that there are parts of the landscape in which the weights can be perturbed without significantly changing the loss value. Such regions at the bottom of the landscape are called *the flat minima*, which can be considered as another way of saying a *very wide minima*. It is further noted that such minima have better generalization properties and a new loss function that makes use of the Hessian of the loss function has been proposed that targets the *flat minima*. The computational complexity issues have been attempted to be resolved using the R-operator of Pearlmutter [1994]. However, the new loss requires all the entries of the Hessian, and even with the R-operator it is unimaginably slow for today's large networks. More recently, an *exact* numerical calculation of the Hessian have been carried out by Sagun et al. [2016]. It turns out that the Hessian is degenerate at any given point including the randomly chosen initial point, that the spectrum of it is composed of two parts: (1) the bulk, and (2) the outliers. The bulk is mostly full of zero eigenvalues with a fast decaying tail, and the outliers are only a handful which appears to depend on the data. This implies that, locally, most directions in the weight space are flat, and leads to little or no change in the loss value, except for the directions of eigenvectors that correspond to the large eigenvalues of the Hessian. Based on exactly this observation combined with the K-replica method of the previous paragraph, a recent work produced promising practical results [Chaudhari et al., 2016].

1.2 Overview of results

In an attempt to develop an understanding for some of the classical and modern problems described above and hopefully to get further insight into non-convex optimization over high dimensional spaces, we study the loss function through its second order properties through the spectrum of its Hessian matrix.

The first observation is that the Hessian is not slightly singular but extremely so, having almost all of its eigenvalues at or near zero. And this observations seems to hold even at random points of the space. For example, let us consider an artificial neural network with two layers (totaling $5K$ parameters) and trained using gradient descent. Figure 1 shows the full spectrum of the Hessian at the random initial point of training and after the final training point. Intuitively, this kind of singularity should be expected to emerge if (1) the data is essentially low dimensional but lives in a larger ambient space, and (2) the model is over-parametrized. In the context of deep learning, over-parametrization is a key feature of the model, and in most cases the data is expected to be essentially on a low dimensional manifold.

We study the Hessian using a decomposition [LeCun et al., 1998] as a sum of two matrices, $\nabla^2\mathcal{L}(w) = G(w) + H(w)$ where G is the sample covariance matrix of the gradients of model outputs and H is the Hessian of the model outputs. We show that the approximation $\nabla^2\mathcal{L}(w) \approx G(w)$ becomes more and more accurate as training progresses, and in fact we can prove equality at the global minimum. This connection to the covariance of the gradients explains why having more parameters than samples causes such degeneracy of the Hessian. We further observe that the singularity holds true even when M is comparable to N. Also, we review results on the spectrum of sample covariance matrices,

[1] A general remark is that the terms wide-narrow, high-low, are all relative. Everything can be scaled and skewed to take desired shapes. An example is the recent, Dinh et al. [2017], work shows how sharp minima can still generalize with proper modifications to the loss function. In this work we pay attention to fix as many variables as we can to get a consistent comparison across different setups.

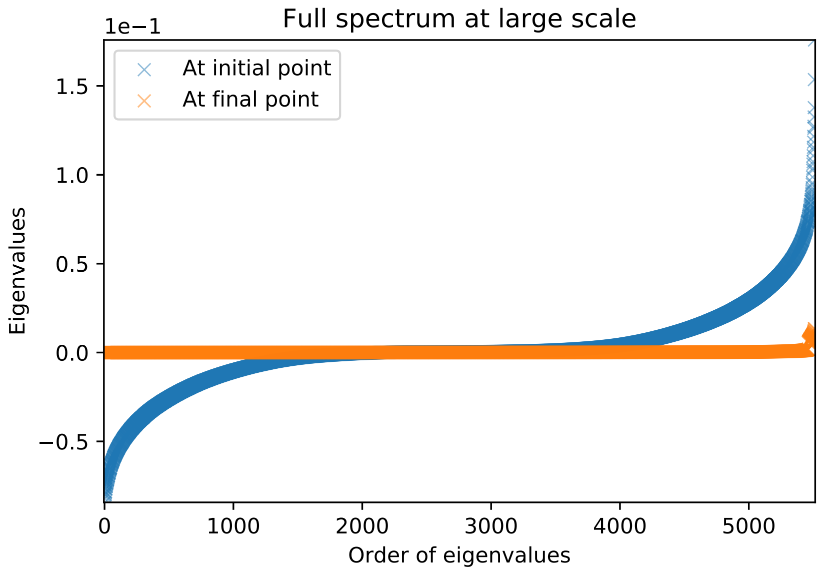

Figure 1: Spectrum at a random point, and at the end of GD when the random point is taken as the initial point.

and attempt to relate those results to machine learning models. Finally, we empirically examine the spectrum, carefully tuning its components, to find uncover intricate dependencies within the data-architecture-algorithm triangle. We begin by discussing the decomposition of the Hessian.

2 Generalized Gauss-Newton decomposition of the Hessian

In order to study its spectrum, we will describe how the Hessian can be decomposed into two meaningful matrices. Suppose the loss function is given as a composition of two functions, the model function $f : \mathbb{R}^M \times \mathbb{R}^d \longrightarrow \mathbb{R}$ is the real valued output of a network that depends on the parameters and a given example; and the loss function $\ell : \mathbb{R} \times \mathbb{R} \longrightarrow \mathbb{R}^+$ which is constrained to be convex in its first component [2]. For ease of reading, we suppressed the dependencies of functions ℓ and f to data (x, y) and unless noted otherwise the gradients are with respect to w. The gradient of the loss per fixed sample is given by

$$\nabla \ell(f(w)) = \frac{d}{ds}\ell(s)\big|_{s=f(w)} \nabla f(w) \tag{3}$$

and then the Hessian can be written as

$$\nabla^2 \ell(f(w)) = \frac{d^2}{ds^2} f(s)\big|_{s=f(w)} \nabla f(w) \nabla f(w)^T + \frac{d}{ds} f(s)\big|_{s=f(w)} \nabla^2 f(w) \tag{4}$$

where $\star^T$ denotes the transpose operation, in which case is the gradient column-vector. Note that since ℓ is convex $\ell''(s) \geq 0$ we can take its square root. This allows us to define g, H, the scaled gradient and the scaled Hessian of f,

$$g(w) = \sqrt{\ell''(f(w))} \nabla f(w) \tag{5}$$

$$H(w) = \ell'(f(w)) \nabla^2 f(w). \tag{6}$$

Combining these results we can rewrite the Hessian of the loss as follows:

$$\nabla^2 \mathcal{L}(w) = \underbrace{\frac{1}{N} \sum_{i=1}^{N} g_i(w) g_i(w)^T}_{G(w)} + \underbrace{\frac{1}{N} \sum_{i=1}^{N} H_i(w)}_{H(w)} \tag{7}$$

where the subscript i indicates the index of the example $(x, y) \in \mathcal{D}$. This way we see that the Hessian of the loss can be decomposed in two parts where the generalized Gauss-Newton matrix G is the average of rank one matrices obtained by gradients of outputs, and H is the average of the Hessians of outputs. Some classical examples are as follows:

[2] Even then, in many cases the overall loss function is non-convex in the space of its parameters.

Example 1. The mean-square loss when $\ell(s, y) = (s - y)^2$ where $s = f(w, x)$.

Example 2. The hinge-loss when $\ell(s, y) = \max\{0, sy\}$. Note that the hinge-loss is piecewise differentiable and its second derivative is zero.

Example 3. Negative log-likelihood loss when $\ell(s_y, y) = -s_y + \log \sum_{y'} \exp s_{y'}$ and $f_y(w; x) = s_y$ for classification problems. For instance, we can have $y \in \{1, \dots, k\}$ that indicates the correct class of the input x, and the machine outputs a vector of length k described by $f(w, x) = (f_1(w, x), \dots, f_k(w, x)) \in \mathbb{R}^k$ before its fed to the *softmax* operator.

We have various choices for f:

1. $f(w, x) = \langle w, x \rangle$ for the Perceptron.
2. $f(w, x) = (W_l \dots W_1 x)$ for a toy layered network.
3. $f(w, x) = W^T x$ for W an $d \times k$ for k-class logistic regression.
4. $f(w, x) = W_l \sigma \circ \dots \sigma \circ W_1 x$ for a fully connected neural network with non-linearities described by σ that acts on vectors entry-wise.

The output can also come from a convolutional network, or any other deep learning model, so such a decomposition is fairly general. What we can say about it may heavily depend on its specifics. In general, it doesn't appear to be so straightforward to discuss the spectrum of the sums of matrices by looking at the individual ones. Nevertheless, looking at the decomposition, we can still speculate on what we should expect. G is the sum of rank-1 matrices. The sum runs over data, and the data typically have redundancies. For instance, if the problem is classification, and the data is sampled from k clusters, then it is not unreasonable to expect that the non-trivial eigenvector of gg^T's are somewhat aligned when the on the examples coming from the same cluster. This suggests that G may be approximated by a rank-k matrix. Moving to the second term: Looking at H we have the Hessian of f. The structure of this matrix depends on the architecture.

2.1 The spectrum of the generalized Gauss-Newton matrix

In this section, we will show that the spectrum of the Generalized Gauss-Newton matrix can be characterized theoretically under some conditions. Suppose that we can express the scaled gradient as $g = Tx$ with the matrix $T \in M \times d$ depending only on the parameters w - which is the case for linear models. Then we can write G as

$$G = \frac{1}{N} \sum_{i \in \mathcal{D}} g_i g_i^T = \frac{1}{N} T X X^T T^T \tag{8}$$

where $X = \{x^1, \dots, x^N\}$ is an $d \times N$ matrix Furthermore, without loss of generality, we assume that the examples are normalized such that the entries of X are independent with zero mean and unit variance. One of the first steps in studying G goes through understanding its principle components. In particular, we would like to understand how the eigenvalues and eigenvectors of G are related to the ones of Σ where $\Sigma := \mathbb{E}(G) = \frac{1}{N} T X X^T T^T = T T^T$.

In the simplest case, we have $\Sigma = Id$ so that the gradients are uncorrelated and the eigenvalues of G are distributed according to the Marčenko-Pastur law in the limit where $N, M \to \infty$ and $\alpha := \frac{M}{N}$. The result dates back to sixties and can be found in [Marčenko and Pastur, 1967]. Note that if $M > N$ then there are $M - N$ trivial eigenvalues of G at zero. Also, the width of the nontrivial distribution essentially depends on the ratio α. Clearly, setting the expected covariance to identity is very limiting. One of the earliest relaxations appear in [Baik et al., 2005]. They prove a phase transition for the largest eigenvalues of the sample covariance matrix which has been known as the *BBP phase transition*. A case that may be useful for our setup is as follows:

Theorem 1 (Baik, Arous, Péché, et al., 2005). *If $\Sigma = diag(\ell, 1, \dots, 1)$, $\ell > 1$, and $M, N \to \infty$ with $\frac{M}{N} = \alpha \geq 1$. Let $c = 1 + \sqrt{\alpha}$, and let's call the top eigenvalue of the sample covariance matrix as λ_{max} then:*

- *If $1 \leq \ell < c$ then λ_{max} is at the right edge of the spectrum with Tracy-Widom fluctuations.*
- *If $c < \ell$ then λ_{max} is an outlier that is away from bulk centered at $\ell(1 + \frac{\alpha}{\ell - 1})$ with Gaussian fluctuations.*

Typically, due to the correlations in the problem we don't have Σ to be the identity matrix or a diagonal matrix with spikes. This makes the analysis of their spectrum a lot more difficult. A solution

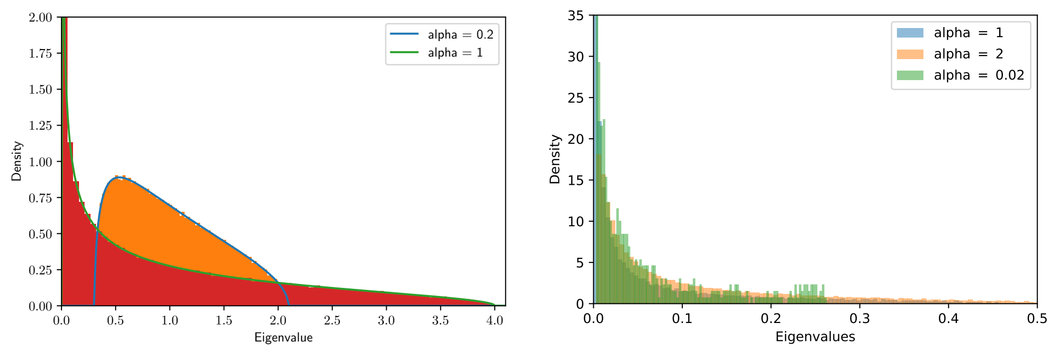

Figure 2: Spectrum of the logistic regression loss with tanh unit: when data has a single Gaussian blob (left), when data has two Gaussian blobs (right). In the latter case, the spectrum has outlier eigenvalues at 454.4, 819.5, and 92.7 for $alpha = 1, 2, 0.02$, respectively.

for this slightly more general case with non-trivial correlations has been provided only recently by Bloemendal et al. [2016]. We will briefly review these results here see how they are related to the first term of the above decomposition.

Theorem 2 (Bloemendal, Knowles, Yau, and Yin, 2016)**.** *If $d = M$, $\Sigma - Id$ has bounded rank, $\log N$ is comparable to $\log M$, and entries of X are independent with mean zero and variance one, then the spectrum of Σ can be precisely mapped to the one of G as $M, N \to \infty$ for fixed $\alpha = \frac{M}{N}$. Let $K = min\{M, N\}$, and the decomposition of the spectrum can be described as follows:*

- ***Zeros:** $M - K$ many eigenvalues located at zero (if $M > N$).*
- ***Bulk:** Order K many eigenvalues are distributed according to Marčenko-Pastur law.*
- ***Right outliers:** All eigenvalues of Σ that exceed a certain value produce large-positive outlier eigenvalues to the right of the spectrum of G.*
- ***Left outliers:** All eigenvalues of Σ that are close to zero produce small outlier eigenvalues between 0 and the left edge of the bulk of G.*

Moreover, the eigenvectors of outliers of G are close to the corresponding ones of Σ.

2.2 Applications and experiments on artificial data

Previous section essentially describes the way in which one obtains outlier eigenvalues in the sample covariance matrix assuming the population covariance is known. Here are some examples:

Example 4 (Perceptron)**.** The Hessian for the Perceptron with regular hinge-loss is the zero matrix. It has no local curvature, the weight updates are driven by the examples. Though this doesn't mean that the landscape is trivial: the landscape is composed of constant gradient patches whose boundaries are determined by the sign changes of f.

Example 5 (Logistic regression)**.** A slightly less trivial version could be considered for the log-loss $\ell(s, y) = -y \log \frac{1}{1+e^{-s}} - (1 - y) \log(1 - \frac{1}{1+e^{-s}})$ and a single neuron with sigmoid. Note that $\ell(s, y)$ is convex in s for fixed y, and we can apply the decomposition using ℓ and $f(w, x) = \langle w, x \rangle$. In this case we have $M = d$ and the assumptions can be checked easily for the matrix g. Also, note that the second part of the Hessian is zero since $\nabla^2 f(w, x) = 0$. So the Hessian of the loss is just the first term: G. It is straightforward to calculate that the gradient per sample is of the form $g = c(w, x, y) Id_M x$ for a positive constant $c = c(w, x, y)$ that depends on y. If we had only one class in the data, then the we wouldn't have the dependency on y and this case would fall into the classical Marčhenko-Pastur law (left pane of 2). However once we have more than one class, we can't get rid of the y dependency and the spectrum changes. What should we expect *with* the dependency? It turns out that in that case the weights have one large outlier eigenvalue, and a bulk that's close to zero (right pane of 2).

Example 6 (Toy layered network)**.** Take the same ℓ from the previous example, and set $f(w) = W_l \dots W_1 x$ where W_l is a column vector so that f is real valued, and w is the *flattened* weight vectors. Note that we can write both g_i and H_i in weight-matrix times input form. We emphasize that the the weight-matrix may depend on the label of its input. We also note that H_i has a block structure that is separated among its layers and its terms are somewhat sparse.

alg-size	cov	mean&std	nc	gap	ratio	heur	cutoff
GD tanh 4022	1.0	1.19e-05, 1.19e-05	2	**2**	**3**	N/A	N/A
GD tanh 4115	1.0	2.52e-05, 2.52e-05	5	2	17	N/A	N/A
GD tanh 4270	1.0	3.75e-05, 3.75e-05	10	2	4	N/A	N/A
GD tanh 4580	1.0	5.37e-05, 5.37e-05	20	3	4	N/A	N/A
GD ReLU 4022	1.0	7.13e-06, 7.13e-06	2	**3**	4	**2**	0.003
GD ReLU 4115	1.0	1.55e-05, 1.55e-05	5	**6**	7	**5**	0.003
GD ReLU 4270	1.0	3.04e-05, 3.04e-05	10	**11**	12	**10**	0.003
GD ReLU 4580	1.0	5.65e-05, 5.65e-05	20	4	**21**	**21**	0.003
GD ReLU 5510	1.0	1.16e-04, 1.16e-04	50	**50**	**51**	**49**	0.003
SGD ReLU 4022	1.0	8.96e-06, 8.96e-06	2	**3**	4	**2**	0.002
SGD ReLU 4115	1.0	1.62e-05, 1.62e-05	5	**6**	7	7	0.002
SGD ReLU 4270	1.0	2.97e-05, 2.97e-05	10	2	14	15	0.002
SGD ReLU 4580	1.0	4.53e-05, 4.53e-05	20	2	3	**21**	0.002
SGD ReLU 5510	1.0	6.78e-05, 6.78e-05	50	**50**	**51**	**49**	0.002
SGD ReLU 4022	10.0	9.49e-05, 9.49e-05	2	**2**	4	**2**	0.015
SGD ReLU 4115	10.0	1.68e-04, 1.68e-04	5	**5**	**6**	**5**	0.015
SGD ReLU 4270	10.0	1.92e-04, 1.92e-04	10	2	**11**	**9**	0.015
SGD ReLU 4580	10.0	3.10e-04, 3.10e-04	20	**20**	**21**	**19**	0.015
SGD ReLU 5510	10.0	1.67e-04, 1.67e-04	50	**50**	**51**	**49**	0.005

Table 1: Counting outliers for matching the number of blobs. Dictionary of table elements: {cov: scale of covariance for inputs, mean&std: of the eigenvalues, nc: number of clusters, gap: largest consecutive gaps, ratio: largest consecutive ratios, heur: heuristic threshold, cutoff: the value of the heur.}

Example 7 (A network with non-linearity)**.** The general structure of the Hessian should carry over to the case when the non-linearity is ReLU, too, since this case can be considered as a piece-wise matrix multiplication where the boundaries of the pieces depend on the data. As we have seen in the logistic regression case, this dependency can make or break things. Also, it is harder in the case of sigmoid non-linearities since now we may not have the nice matrix multiplication representation. Due to these considerations this case seems to be a non-trivial candidate for counting large eigenvalues.

In the subsequent experiment, we used a feed-forward neural network with a 100 dimensional input layer, two hidden layers each of which with 30 hidden units, and a k dimensional output layer that is combined with softmax for k-class classification. We sampled random k Gaussian clusters in the input space and normalized the data globally. Then we carried out the training for the following sets of parameters: $k : \{2, 5, 10, 20, 50\}$, algorithm: {GD, SGD}, non-linearity: {tanh, ReLU}, initial multipler for the covariance of the input distribution: $\{1, 10\}$. Then we counted the number of large eigenvalues according to three different cutoff methods: largest consecutive gap, largest consecutive ratio, and a heuristic method of determining the threshold by searching for the elbow in the scree plot (see Figure 4 right pane). In Table 1 we marked the ones that are off by ± 1.

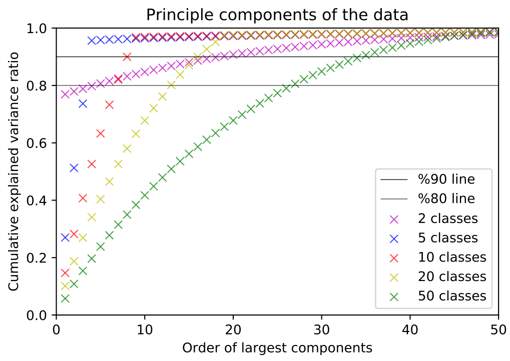

Figure 3: PCA of data.

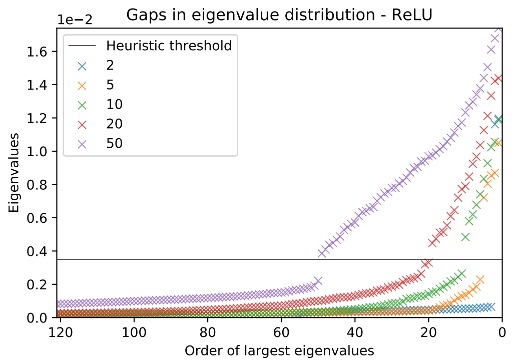

Figure 4: Scree plot.

3 Larger scale experiments

In this section we show two experiments in an attempt to address the following questions: (1) How does the spectrum (and hence the geometry) change when we just increase the size of the system? (2) What does over-parametrization imply on the discussion around GD vs. SGD or large batch vs small batch?

3.1 MNIST and growing the network size

We test the effect of growing the size of the network with fixed data, architecture, and algorithm. We sample 1K examples from the MNIST dataset, and train all the networks with the same step size until a sufficient stopping condition is satisfied. Then we compute the exact Hessian and plot its eigenvalues in Figures 5 and 6. Note that we use different ways of plotting the left pane and the right pane: the left pane is scaled to indicate the *ratios* of eigenvalues whereas the right pane shows the *counts*. The fact that this particular choice creates consistent plots should not be surprising since this confirms our previous suspicions about the fact that adding neurons in the system proportionally enlarges the bulk, but the outliers on the right depend on data, so their number should not, in principle, change with the increasing dimensionality of the parameter space.

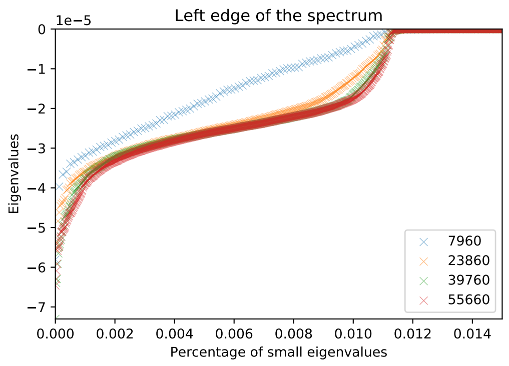

Figure 5: Left edge - *scaled*.

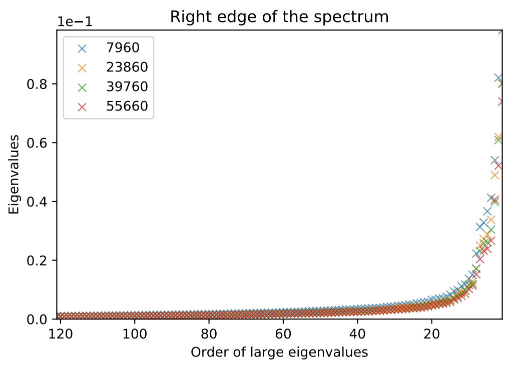

Figure 6: Right edge.

3.2 CIFAR10 and large/small batch comparison

A common way to plot training profiles in larger scale neural networks is to stop every epoch to reserve extra computational power to calculate various statistics of the model at its current position. This becomes problematic when one compares training with different batch sizes, primarily because the larger batch model takes fewer steps in a given epoch. Recall that the overall loss is averaged, therefore, for a fixed point in the weight space, the empirical average of the gradients is an unbiased estimator for the expected gradient. Hence it is reasonable to expect that the norms of the large batch methods match to the ones of the small batch. And for a fair comparison, one should use the same learning rate for both training procedures. This suggests that a better comparison between GD and SGD (or LB and SB) should be scaled with the number of steps, so that, on average both algorithms are able to take similar number of steps of comparable sizes. Moreover, since random initial points in high dimensional spaces are almost always orthogonal, the fact that training with random initial points may lead to different basins should not be a surprising observation. However, the observation that LB converges to tighter basins that is separated by wells from the wider basins sound by SB has been an observation that triggered attention. To test this idea even further we conduct the following experiment:

1. *Part I:* Train full CIFAR10 data for a bare AlexNet (no momentum, no dropout, no batch normalization) with a batch-size of $1,000$. Record every 100 steps for 250 times.
2. *Part II:* Continue training from the end point of the previous step with a smaller batch-size of 32. Everything else, including the constant learning rate is kept the same. And train another 250 periods each of which with 100 steps.

The key observation is the jump in the training and test losses, and a drop in the corresponding accuracies (Figure 7). Toward the end of *Phase II* the small batch reaches to a slightly better accuracy.

And this looks in line with the observations in Keskar et al. [2016], in that, it appears that the LB solution and SB solutions are separated by a barrier, and that the latter of which generalizes better. Moreover, the line interpolations extending away from either end points appear to be confirming the sharpness of LB solution. However, we find the line interpolation connecting the end points of *Part I* and *Part II* turns out to *not* contain any barriers (Figure 8). This suggests that while the small and large batch method converge to different solutions, these solutions have been in the same basin all along. This raises the striking possibility that that other seemingly different solutions may be similarly connected by a flat region to form a larger basin.

One way to interpret the results goes through the Gauss-Newton decomposition introduced earlier. When we decrease the batch size, we increase the noise in the covariance of the gradients, and hence the first term starts to dominate. Even when the weight space has large flat regions, the fluctuations of the stochastic noise should be precisely in the directions of the large eigenvalues.

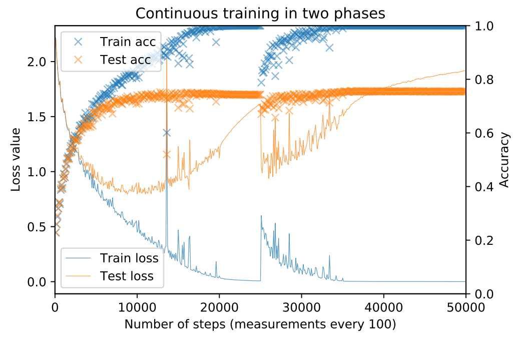

Figure 7: Large batch training immediately followed by small batch training on the full dataset of CIFAR10 with a *raw* version of AlexNet.

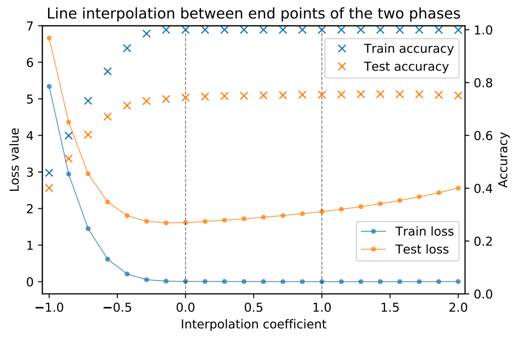

Figure 8: Loss and accuracy evaluation on the line interpolated between the end points of LB solution and SB solution.

4 Conclusion

We have shown that the level of singularity of the Hessian cannot be ignored. We use the generalized Gauss-Newton decomposition of the Hessian to argue the cluster of zero eigenvalues are to be expected in practical applications. This allows us to divide the process of training into two parts with initial fast decay and final slow progress. We examine the option that the inherent geometry at the bottom of the landscape could be accountable for the slow progress in the final phase due to its extremely flat structure.

One of the most striking implications of flatness may be the connected structure of the solution space. We may wonder whether two given solutions can be connected by a continuous path of solutions. This question have been explored in a recent work: in Freeman and Bruna [2016] it shown that for one hidden layer rectified neural networks the solution space is connected which is consistent with the flatness of the landscape.

We speculate that the classical notions of basins of attractions may not be the suitable objects to study for neural networks. Rather, we may look at the interiors of level sets (also called the *excursion sets*) of the landscape. Given a loss-level u the *excursion set* below u is the set of all points of the domain that has loss value less than or equal to u: $A(u) = \{w : w \in \mathbb{R}^M \& \mathcal{L}(w) \leq u\}$. Exploring $A(u)$ is challenging and gradient based methods seem to be less suitable for this task. For further research we will examine the distinction between the gradient based part and the excursion set exploration of training.

References

Antonio Auffinger, Gérard Ben Arous, and Jiří Černỳ. Random matrices and complexity of spin glasses. *Communications on Pure and Applied Mathematics*, 66(2):165–201, 2013.

Jinho Baik, Gérard Ben Arous, Sandrine Péché, et al. Phase transition of the largest eigenvalue for nonnull complex sample covariance matrices. *The Annals of Probability*, 33(5):1643–1697, 2005.

Andrew J Ballard, Ritankar Das, Stefano Martiniani, Dhagash Mehta, Levent Sagun, Jacob D Stevenson, and David J Wales. Energy landscapes for machine learning. *Physical Chemistry Chemical Physics*, 2017.

Alex Bloemendal, Antti Knowles, Horng-Tzer Yau, and Jun Yin. On the principal components of sample covariance matrices. *Probability Theory and Related Fields*, 164(1-2):459–552, 2016.

Léon Bottou. Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, 91(8), 1991.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Physica-Verlag HD, 2010.

J Bourrely. Parallelization of a neural network learning algorithm on a hypercube. *Hypercube and distributed computers. Elsiever Science Publishing*, 1989.

Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gradient descent into wide valleys. *arXiv preprint arXiv:1611.01838*, 2016.

Laurent Dinh, Razvan Pascanu, Samy Bengio, and Yoshua Bengio. Sharp minima can generalize for deep nets. *arXiv preprint arXiv:1703.04933*, 2017.

C Daniel Freeman and Joan Bruna. Topology and geometry of deep rectified network optimization landscapes. *arXiv preprint arXiv:1611.01540*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

Y LeCun, L Bottou, GB ORR, and K-R Müller. Efficient backprop. *Lecture notes in computer science*, pages 9–50, 1998.

Jason D Lee, Max Simchowitz, Michael I Jordan, and Benjamin Recht. Gradient descent converges to minimizers. *University of California, Berkeley*, 1050:16, 2016.

Vladimir A Marčenko and Leonid Andreevich Pastur. Distribution of eigenvalues for some sets of random matrices. *Mathematics of the USSR-Sbornik*, 1(4):457, 1967.

Song Mei, Yu Bai, and Andrea Montanari. The landscape of empirical risk for non-convex losses. *arXiv preprint arXiv:1607.06534*, 2016.

Jorge Nocedal and Stephen J Wright. Numerical optimization, second edition. *Numerical optimization*, pages 497–528, 2006.

Ioannis Panageas and Georgios Piliouras. Gradient descent only converges to minimizers: Non-isolated critical points and invariant regions. *arXiv preprint arXiv:1605.00405*, 2016.

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

Levent Sagun, V Uğur Güney, Gérard Ben Arous, and Yann LeCun. Explorations on high dimensional landscapes. *ICLR 2015 Workshop Contribution, arXiv:1412.6615*, 2014.

Levent Sagun, Léon Bottou, and Yann LeCun. Singularity of the hessian in deep learning. *arXiv preprint arXiv:1611.07476*, 2016.

Tom Schaul, Sixin Zhang, and Yann LeCun. No more pesky learning rates. *ICML (3)*, 28:343–351, 2013.