Catalogue of Loss Surfaces

Shristi Gupta
Indian Institute of Technology Hyderabad
IITH Main Road, Near NH-65, Sangareddy,
Kandi, Telangana 502285
ai19acmtech11002@iith.ac.in

V V N Krishna Kanth
Indian Institute of Technology Hyderabad
IITH Main Road, Near NH-65, Sangareddy,
Kandi, Telangana 502285
ai19acmtech11005@iith.ac.in

Abstract

While dealing with optimizations in deep neural networks, it is often observed that the loss surfaces are usually not convex in nature. Due to this drawback, many networks might end up at a sub-optimal point. However, multiple researches have shown ways to convert such surfaces into less complicated surfaces, some of which are convex. In this study, we propose to list out all such possible conversions from the existing literature.

1. Introduction

Deep neural networks are widely used today for a variety of applications today. Optimization is at the heart of the training of such networks. Often, these optimizations pose challenges to practitioners, as they are usually non-convex. Due to this, the updates are sub-optimal in nature, giving high losses and hence wrong predictions in most cases. Given this scenario, there have been multiple researches dealing with conversion of such loss surfaces into simpler surfaces or even convex surfaces in a few cases and get a close-to-optimal solution. Even if a complete conversion into convex problem is not possible, researches have shown it to be possible to get a surface that gives an optimum that better than the original problem. In this study, we hereby show a few of such works.

2. Related work

The practice of learning about loss surfaces has been existing for a long time now and has now become more important due to the advent presence of deep neural networks in every area and their non-convex nature while performing the task of optimization, which is in the core in the working of any of these networks. The related work to the proposed study could be broadly divided into three main areas of understanding. They are Visualization of Loss Surface as shown by Hao Li et al. in [1], using Hessian Analysis as shown by Behrooz Ghorbani et al. in [2], and from Optimization Perspective as shown by Haowei He et al. in [3]. In addition to these new studies, there have been a lot of important base studies. Of them, a few were published importantly in 1993 and 1998. There is another thesis work that we have referred to compile more work, which was presented in 2000. These studies are mentioned later in this study and the can be referred in works by An Mei Chen et al. in [4], Leonard G.C.Hamey in [5], and Marcus R. Gallagher in [6].

In addition to all these standard references, we found an important empirical work on loss surfaces by Im et al. in [7]. This also acts as one of the base studies for this study.

There is also a preprint study that we have considered to be interesting and have hence mentioned in the work. The study could be found by Wojciech Marian Czarnecki et al. in [8].

2.1. Brief description of related work

As part of the work by Hao Li et al. in [1], the authors have explored the structure of neural loss function, and the effect of loss landscapes on generalization using a range of visualization methods. A simple “filter normalization” method was introduced that helps visualize loss function curvature. Side-by-side comparisons between loss functions was also made. Then, using a variety of visualizations, it was explored on how network architecture affects the loss landscape, and how training parameters affect the shape of minimizers. Comparison of various well-known architectures with skip connections, and their corresponding loss surfaces were also made in this study.

In the study related to Hessian Analysis by Behrooz Ghorbani et al. in [2], a tool to study the evolution of the entire Hessian spectrum throughout the optimization process. Using this tool, a number of hypotheses concerning smoothness, curvature and sharpness were studied. Then a thorough analysis of the crucial structural feature of the spectrum was performed. It was found that in non-batch normalized networks, rapid appearance of large isolated eigenvalues in the spectrum, along with a concentration of the gradient in the corresponding eigenspaces was evident. In batch normalized networks, these two effects are almost absent. Through theory and experiments, it was explained on how they affect the optimization speed.

In the work of Haowei He et al. in [3], the authors have defined the concept of asymmetric valley. These are local minima wherein the loss along one of the sides changes

abruptly and very slowly on the other. Since algorithms like the gradient descent is known optimize well, under mild conditions, it was shown that the generalization is more biased towards the flat side. It was further proven that the simply averaging the weights along the Stochastic Gradient Descent trajectory gives rise to such biased solutions implicitly. It further goes on to say that Batch Normalization is responsible for the creation of asymmetric valleys.

Moving on to an older work mentioned as part of An Mei Chen et al. in [4], the author has tried to show the geometric structure of feedforward neural networks of the equioutput weight space transformations is explored for the case of multilayer perceptron networks with tanh activation functions. Many feedforward neural network architectures have the property that their overall input-output function is unchanged by certain weight permutations and sign flips. It was shown that these transformations form an algebraic group isomorphic to a direct product of Weyl groups. Results concerning the root spaces of the Lie algebras associated with these Weyl groups were then used to derive sets of simple equations for minimal sufficient search sets in weight space. These sets, which take the geometric forms of a wedge and a cone, occupy only a minute fraction of the volume of weight space. A separate analysis showed that large numbers of copies of a network performance function optimum weight vector were created by the action of the equioutput transformation group and that these copies all lie on the same sphere. Some implications of these results for learning were discussed.

The study mentioned in the work of Leonard G.C.Hamey in [5], is a case study pertaining to local minima in feedforward neural networks. A new methodology for analysis was presented, based upon consideration of trajectories through weight space by which a training algorithm might escape a hypothesized local minimum. This analysis was then applied to the classical XOR problem, which has previously been considered to exhibit local minima. The analysis proved the absence of local minima, eliciting significant aspects of the structure of the error surface. This study is significant because it can be helpful in the study of the existence of local minima in feedforward neural networks, and also for the development of training algorithms which avoid or escape entrapment in local minima.

The work of Marcus R. Gallagher in [6] is a PhD thesis, from which we have analyzed for information. The work concentrates on visualizing and describing the loss surfaces. The concept of the error surface is explored using three related methods. Firstly, Principal Component Analysis (PCA) was proposed as a method for visualizing the learning trajectory followed by an algorithm on the error surface. It was found that PCA provides an effective method for performing such a visualization, as well as providing an indication of the significance of individual weights to the training process. Secondly, sampling methods were used to explore the error surface and to measure certain properties of the error surface, providing the necessary data for an intuitive description of the error surface. A number of practical MLP error surfaces were found to contain a high degree of ultrametric structure, in common with other known configuration spaces of complex systems. Thirdly, a class of global optimization algorithms were also developed, which focused on the construction and evolution of a model of the error surface (or search space) as an integral part of the optimization process. The relationships between this algorithm class, the Population-Based Incremental Learning algorithm, evolutionary algorithms, and cooperative search were discussed.

Another important research result was obtained in a study mentioned by Im et al. in [7]. This is a watershed study for us since it provides us with the groundwork on which we would build our study in this work. The work can be broadly into four major parts. The first category of work speaks about ‘Different optimization methods to find different local minima’. The next category concerns with ‘Different optimization methods to find different “types” of local minima’. The third one deals about ‘Different local minima being found even at later stages of training’. Experimental results were also provided.

Finally, in the work by Wojciech Marian Czarnecki et al. in [8], we found the work that studies the low-level details of the loss surface of deep networks. This study is a development over the work published by Ivan Skorokhodovet al. in [9]. While the older work demonstrated that one can empirically find arbitrary 2D binary patterns inside loss surfaces of popular neural networks, the newer work proves a couple of important points. It first proves that the work by Ivan Skorokhodovet al. in [9] can be generalized property of deep universal approximators. Then it goes on to prove that this property holds for arbitrary smooth patterns, for other dimensionalities, for every dataset, and any neural network that is sufficiently deep and wide. Its analysis predicts not only the existence of all such low-dimensional patterns, but also two other properties that were observed empirically: that it is easy to find these patterns; and that they transfer to other data-sets.

3. Motivation

There has been interest in trying to understand loss surfaces of deep neural networks (especially considering the non-convex nature of the problem, and the success of them despite the non-convexity), and many papers have been published for a very long time in this context. Given that it is difficult to survey the complete literature every time and given the practical importance of this study and that it would help us more in understanding, we have chosen this particular field.

4. Proposed methodology

The methodology includes the study of literature, dating from 1990s to as latest as 2020. It was observed that a lot of research has been conducted towards the end of the 20th century, before it restarted in the second decade of the 21st century. The project also includes comparative study whenever possible so that various approaches could be explained to understand a given surface and hence finding an optimal way to solve it. Also, we showcase the results from various websites on the world wide web. The major idea is to obtain and organise as many references as possible.

5. Results

Below, we present the list of the studies that we would like to compile. We would start with the work mentioned by Im et al. as part of [7]. The result is one of the basic studies for Hessian Analysis. After this, we would mention more works in basic study provided in the works by An Mei Chen et al. in [4], Leonard G.C.Hamey in [5], and Marcus R. Gallagher in [6]. Finally, we would conclude by listing the works by Hao Li et al. in [1], Behrooz Ghorbani et al. in [2], and Haowei He et al. in [3]. These works have been shortlisted after carefully considering about 30-35 works related to this field and filtering further from them.

5.1. Empirical Analysis of Deep Network Loss Surfaces

This section summaries the results of the study mentioned by Im et al. as a part of [7]. Several of the empirical analysis in this work are based on methods mentioned by Goodfellow et al. in [10], in which properties of the loss function are examined by projecting it onto a single, carefully chosen dimension. But in this study, the authors use a similar visualization technique, but choose different low-dimensional subspaces for the projection of the loss function. These subspaces are based on the initial weights as well as the final weights learned using the different optimization algorithms. They are chosen to answer a variety of questions about the loss function and how the different optimization algorithms interact with this loss function. In addition, they have explored the use of two-dimensional projections of the loss function, allowing us to better visualize the space between final parameter configurations. This was done via barycentric and bilinar interpolation for tripletsand quartets of points, respectively. Figure 1 provides a few examples of this.

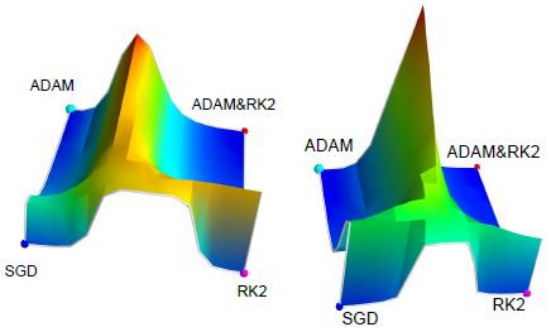


Figure 1: Visualization of the loss surface at weights interpolated between the final points of four different algorithms from the same initialization.

5.1.1. Different optimization methods find different solutions

Figure 2 describes the Visualization of the loss surface near and between the final points found by different optimization methods for the VGG network on CIFAR10. The lower left part of the Figure 1 presents the different optimization methods each reaching a distinct valley. The 3D visualizations indicate that final configurations reached by different methods with same initial configuration are separated by a hump at least up to 2D slice. This is somewhat straightforward, but it is still interesting to see the bifurcation of behavior. The goals were to observe the complexity of loss surfaces in 2D slices, and to determine whether the complexity of the local region found by different optimization methods are different.

This paper (Im et al. in [7]) assumes that Deep Neural Network surfaces are highly symmetrical. We can permute the indices of hidden units in the same layer with corresponding weights and still have the same functional outputs. In practice, what we really need is the functional outputs themselves. So, the authors presented whether the deep networks found by different optimization methods are the same functions or not. The upper right part of the Figure 2 presents the functional output difference between different optimization methods (additional information could be found in section 3.1 of Im et al. in [7]).

These figures illustrate that the local minima found by different methods correspond to effectively different networks, and the hump indicates that they approach different functions.

5.1.2. Different optimization algorithms find different types of solutions

In the work of Im et al. as part of [7], an interesting observation is the shape of basin found by different methods. The authors trained deep neural networks multiple times with different initializations.

As shown in Figure 3, “we see that the surfaces look strikingly similar for different runs of the same algorithm, but characteristically different for different algorithms. Thus, we found evidence that the different algorithms land in qualitatively different types of local minima.”

The absolute size differences between basins found by different methods were interesting to see (additional

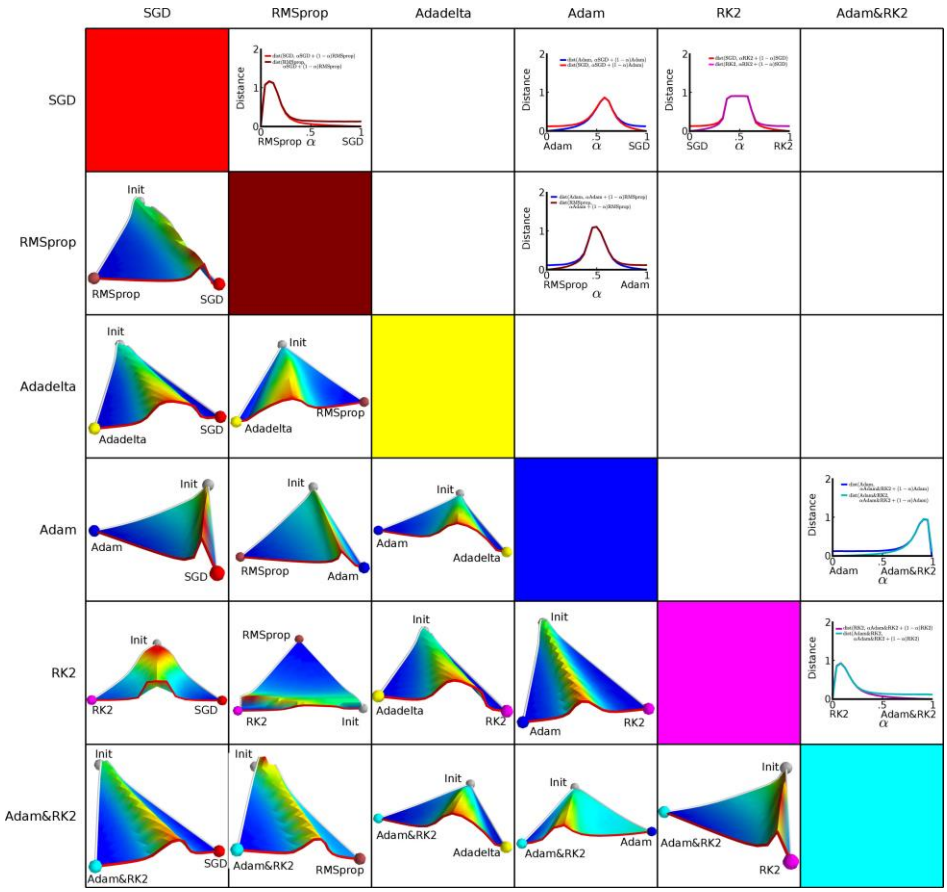
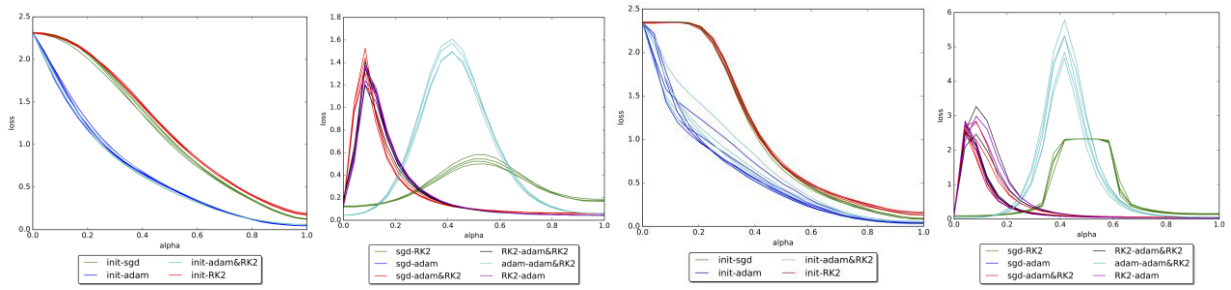


Figure 2: Visualization of the loss surface near and between the final points found by different optimization methods for the VGG network on CIFAR10 (Im et al. in [7]).



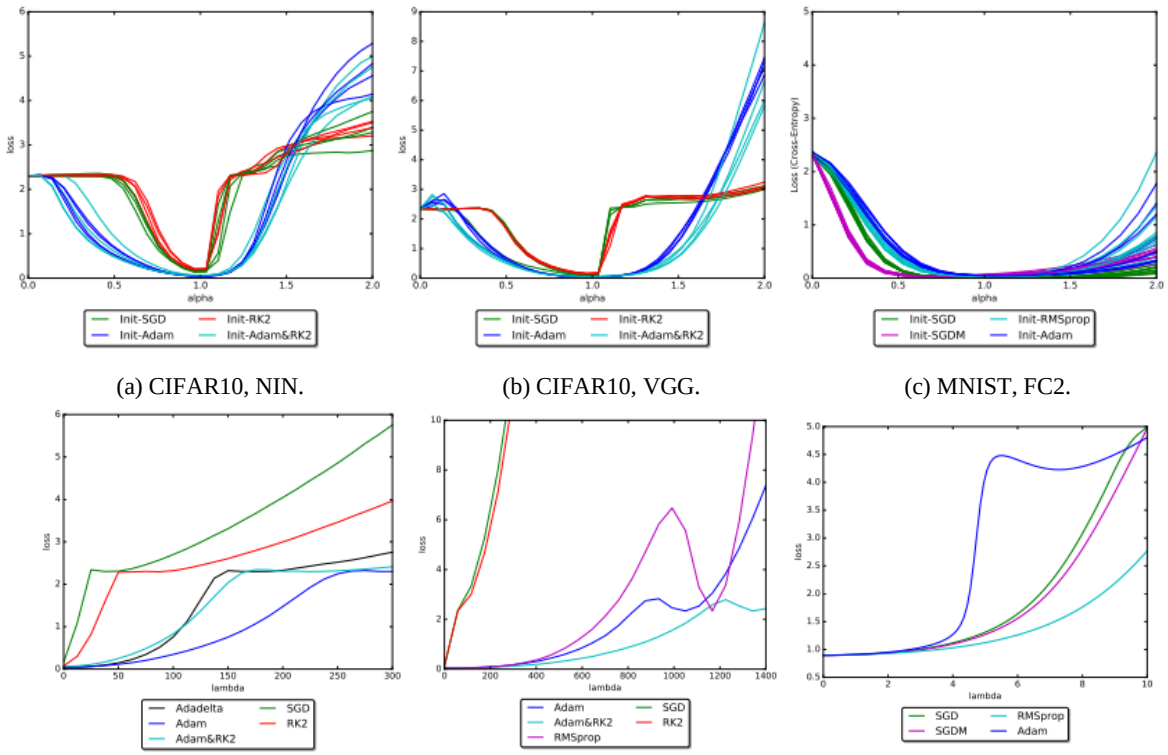
(a) CIFAR10, NIN, Initial to Final.

(b) CIFAR10, NIN, Final to Final.

(c) CIFAR10, VGG, Initial to Final.

(d) CIFAR10, VGG, Final to Final.

Figure 3: Loss function visualizations for multiple re-runs of each algorithm. Each re-run corresponds to a different initialization (Im et al. in [7]).



(a) CIFAR10, NIN. (b) CIFAR10, VGG. (c) MNIST, FC2.

(d) CIFAR10, NIN. (e) CIFAR10, VGG. (f) MNIST, FC2.

Figure 4: Comparison of the size of the projected space local minima basins for different algorithms for NIN (a-b) and VGG (c-d) on CIFAR10 and for FC2 on MNIST (e-f).

information could be found in section 3.2 of Im et al. in [7]). Note that loss value at 2.4 is the initial loss rate.

It was also observed that the size of the basins in the projected spaces around the local minima found by Adam and Adam&RK2 are larger than those found by SGD and RK2 for NIN and VGG on CIFAR10, i.e. that the training loss is small for a wider range of α values. In Figure 4 (a, c, e), α, is a multiplier of the weight vector. If the norm of the weight vector found by one algorithm is larger than that found by another, then a change of Δα for this curve would correspond to a larger absolute change in the weight vector.

5.1.3 Different solutions found at later stages of training

The authors have next studied on when the critical point gets chosen. In other words, the learning curve usually drops dramatically at the beginning of learning, and then the rate of decay slows down as learning progresses.

It can be imagined that the destined local minimum after the fast decay period is already chosen and we are just settling down at the slow decay rate - at least we may think like this after staring at different optimization methods converging on a convex loss surface for a long time.

In the paper, the authors experiment with this idea by switching the descent direction at different points in the training period. For example, switching from ADAM to SGD at the end stage of training. This was to check whether the valley stayed the same even after changing the descending directions. In order to make sure that they are not overshooting (due to large learning rate), they also tried with several learning rates.

To sum up, even when the authors switch optimization algorithms, the local minima found were all different. “This directly disagrees with the notion that the local minimum has effectively been chosen before the “minimization” phase, but instead that which local minimum is found is still in flux this late in optimization” ([7]). We do not observe the basin of attraction even in the later stage of training and the learning curve does not reflect this.

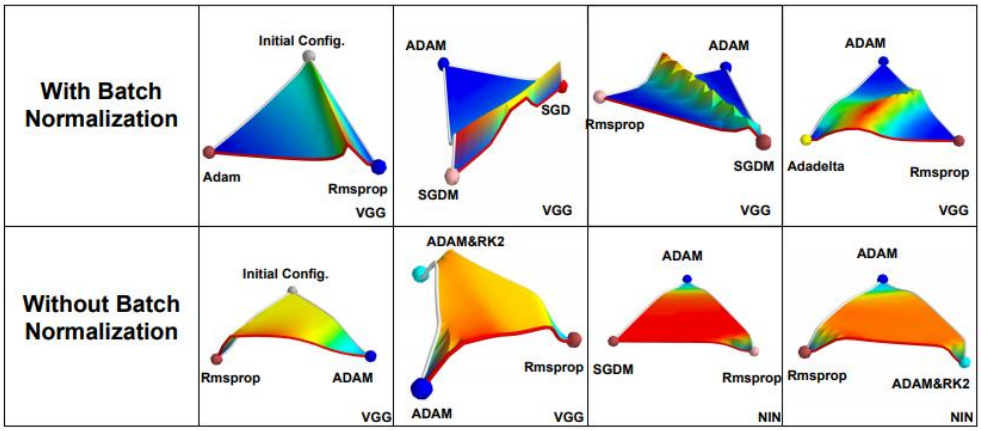
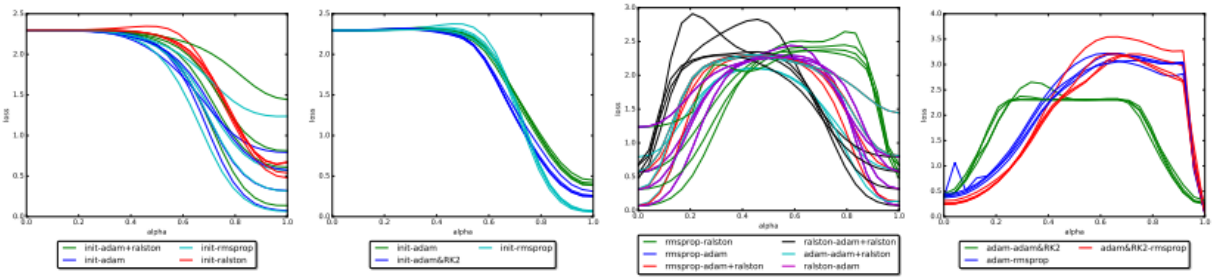


Figure 5: Visualization of the loss surface with and without batch-normalization.



(a) NIN - Initial to Final. (b) VGG - Initial to Final. (c) NIN - Final to Final. (d) VGG - Final to Final.

Figure 6: Loss function visualizations for multiple re-runs of each algorithm without batch normalization. Each re-run corresponds to a different initialization. Without batch normalization, re-runs are much less consistent.

5.1.4. Effect of batch-normalization

Finally, one of the most effective techniques that was proposed and has been widely applied is batch-normalization. To understand how batch normalization affects the types of solutions found, set experiments were performed by comparing loss surfaces near solutions found with and without batch normalization for each of the optimization methods. This visualization was done by interpolating between the initial weights and the final weights as well as between pairs of final weights found with different algorithms. The visualization can be seen in Figure 5.

It can be observed that there are clear quantitative differences between optimization with (Figure 3) and without (Figure 6) batch normalization.

From this, it could be concluded that without batch normalization, the quality of the solutions found by a given algorithm is much more dependent on the initialization. In addition, the surfaces between different solutions are more complex in appearance: with batch normalization we see sharp unimodal jumps in performance but without batch normalization we obtain wide bumpy shapes that are not necessarily unimodal.

5.2. On the Geometry of Feedforward Neural Network Error Surfaces

The work that was presented as part of this study is related to the early feedforward neural networks. The authors of this work are An Mei Chen et al. [4]. They have considered the feedforward networks that remain unchanged to certain weight permutations and weight flips. The geometry of these equioutput weight space transformation has been explored by the study.

As a part of this study, we would present a few important theorems that mentioned as part of the above-mentioned paper.

The first theorem mentioned by the authors is: An equioutput transformation is an analytical (i.e., continuous and expandable in a power series around any point) mapping g: W → W from weight space to weight space which leaves the output of the neural network unchanged. In other words,

$$y'\big(x, g(w)\big) = y'(x, w) \qquad (1)$$

for all x ∈ $\mathbb{R}^n$ and all w ∈ W.

First, consider two types of equioutput transformations: hidden unit weight interchanges and hidden unit weight sign flips. For simplicity, we will refer to these transformations as interchanges and sign pips. An interchange consists of a network weight vector component permutation in which the weight vectors of two hidden units on the same hidden layer are simply interchanged without changing the orderings of the weights within the units. (Note: the term unit weight vector refers to the vector with components equal to the weights within a single unit.) A compensatory interchange of the weights of the next layer units that receive the inputs from the two interchanged units then removes the effect on the network output of the exchange of weights in the previous layer.

The other type of equioutput transformation is where the weight vector of a hidden layer unit is multiplied by -1 (resulting in a sign flip of the output of the unit). A compensatory sign flip is then carried out on all of the weights of units of the next layer associated with the input from the sign flipped unit output.

Then it was shown that the following theorem holds: All equioutput transformations of W to W are compositions of interchange and sign pip transformations. The proof for this theorem can be found in section 2 of the work by An Mei Chen et al., citied in [4].

However, the authors were not able to establish that the theorem holds for continuity and not analicity of the equioutput transformation. They have, nevertheless, postulated it.

The authors have gone forward next to show that the set of all equioutput transformations form a group. The theorem mentioned as part of the paper is as following: The set of all equioutput transformations on W forms a non-Abelian group G of order #G, with

$$\#G = \prod_{l=2}^{K-1} (M_l!)(2^{M_l}) \qquad (2)$$

where $(M_l!)$ of unit weights represents the different interchange transformations for the layer l.

The proof of this theorem could be found in Section 3 of the work by An Mei Chen et al., citied in [4].

As a part of notation, the authors have defined a group O_k as a set of transformations on the vector space $\mathbb{R}^k$ generated by changing the sign of any coordinate component by interchanging any two coordinate components. The next theorem proved as part of this work is as below: The group G is isomorphic to $O_{M_2} \times O_{M_2} \times \ldots \times O_{M_{k-1}}$.

The proof can be found in Section 3 of the work by An Aei Chen et al., citied in [4].

The authors in this study have defined equivalent weight vectors, minimal sufficient search set, and open minimal sufficient set.

Two network weights vectors **u** and **v** in W are equivalent iff there exists g ∈ G such that $g(\mathbf{u}) = \mathbf{v}$.

A minimal sufficient search set is a subset S of W such that each w in W is equivalent to exactly one element in S.

An open minimal sufficient search set is the interior of a minimal sufficient search set whose closure is equal to the closure of its interior.

The authors have shown that there exists an open minimal sufficient search sets in the geometric forms of a wedge and a cone bounded by hyperplanes. These sets are formulated in terms of simple linear and quadratic inequalities, respectively. The wedge interior described by the inequalities:

$$\begin{aligned} w_{li0} &> 0 \quad \text{for } 1 \le i \le M_l,\ 2 \le l \le (K-1) \\ w_{li0} - w_{l(i+1)0} &> 0 \quad \text{for } 1 \le i \le (M_l - 1),\ 2 \le l \le (K-1) \end{aligned} \qquad \ldots(3)$$

is an open minimal sufficient search set for weight space W, as is the cone interior described by

$$\begin{aligned} w_{k10} w_{li0} &> 0 \quad \text{for } 1 \le i \le M_l,\ 2 \le l \le (K-1) \\ w_{K10}(w_{li0} - w_{l(i+1)0}) &> 0 \quad \text{for } 1 \le i \le (M_l - 1),\ 2 \le l \le (K-1) \end{aligned} \qquad \ldots(4)$$

The proof can be found in Section 4 of the work by An Aei Chen et al., citied in [4].

Another interesting fact about the transformations in group G is that they are isometrics. Thus for any **w** ∈ W and any g ∈ G,

$$|\boldsymbol{g}(\boldsymbol{w})| = |\boldsymbol{w}| \qquad (5)$$

Given that the elements of G are isometries, if W* is a finite weight vector which optimizes the network performance function, then the points obtained by applying the elements of G to w* all lie on the same sphere around the origin in weight space. In general, these points are all different and there are #G of them. We call such a set of points a spherical throng of optima.

The copies of an optimal weight vector w* in its spherical throng will, on average, have half of their weights modified from their corresponding values in w* by a transformation in G. Thus, the copies of w*in its throng will, in general, be scattered all over the sphere. It is easy to see that, given any member w of the throng, the nearest neighbor member on the sphere will typically be a vector that is the same as w except for a single sign flip or interchange transformation. Thus, nearest neighbors in the throng will tend to have most of their components the same.

The areal density of a spherical throng of optima depends upon the magnitude of w*, since the number of members of the throng is, in general, #G, which depends only on the architecture of the network. If this magnitude (|w*|) is small, then the density will be high.

The existence of simple formulas for minimal sufficient search sets raises the question of whether such formulas will be of use in learning. They probably will not. In the case of gradient descent learning they would not be of use, since if we are going downhill, and happen to cross the boundary of the minimal sufficient search set, we should just continue the descent. Even if we wanted to move to the corresponding point of the performance surface within the fundamental wedge we could not do so, since we do not yet have a formula for finding this point.

5.3. XOR has no local minima: A case study

As a part of the study by Leonard G.C.Hamey cited as [5], the author has presented a case study. It was till then perceived that XOR problem had exhibited a local minima. This study has proven the absence of such behavior in the XOR problem. The reason for mentioning this work as part of this study is because it proposes a new methodology for analyzing the loss surface which is based upon consideration of trajectories through weight space by which a training algorithm might escape a hypothesized local minimum. This work is important for the study of the existence of local minima in feedforward neural networks, and also for the development of training algorithms which avoid or escape entrapment in local minima.

The network, along with the notations, that is used to analyze this case is shown in Figure 7.

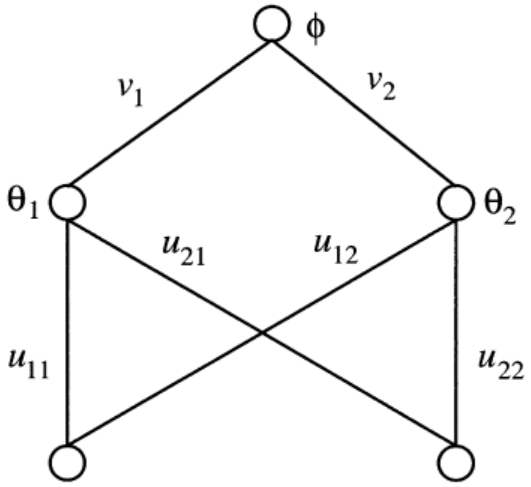


Figure 7: Feedforward network to solve the XOR task, showing notation for connecting and bias weights.

A local minimum can be defined mathematically in the following way. Let $g : \mathcal{D} \rightarrow \mathbb{R}$ be a scalar differentiable function on domain $\mathcal{D}$. The usual definition of a local minimum is that $\boldsymbol{w}_0$ is a local minimum or relative minimum if $g(\boldsymbol{w}_0) \leq g(\boldsymbol{w})$ for all $\boldsymbol{w}$ in a neighbourhood of $\boldsymbol{w}_0$. If $g(\boldsymbol{w}_0) < g(\boldsymbol{w})$ for all $w \neq w_0$ in the neighbourhood then $\boldsymbol{w}_0$ is said to be a strict relative minimum. Unfortunately, these definitions are unsuitable for an analysis of all the error surfaces of artificial neural networks, since such surfaces often exhibit asymptotes that approach a minimum error value. An alternative definition of local minimum is needed. This study provides one such definitions.

A trajectory A from $\boldsymbol{w}_1$ to $\boldsymbol{w}_2$ is a continuous function $A : \mathbb{R} \rightarrow \mathcal{D}$ such that $A(a_1) = \boldsymbol{w}_1$ and $A(a_2) = \boldsymbol{w}_2$, where $a_2 > a_1$. A trajectory from $\boldsymbol{w}_1$ to $\boldsymbol{w}_2$ is denoted by $A_{\boldsymbol{w}_1 \rightarrow \boldsymbol{w}_2}$. A non-ascending trajectory $A_{\boldsymbol{w}_1 \rightarrow \boldsymbol{w}_2}$ has the additional property that the function g is non-increasing along the trajectory; i.e. $\frac{\partial g(A(a))}{\partial a} \leq 0$ for $a_1 \leq a \leq a_2$ wherever $g \circ A$ is differentiable (where $\circ$ denotes function composition).

This alternative definition of local minimum is in terms of regions of the domain that are known to contain one or more minima. The minimum region $M_g(\boldsymbol{w}_0)$ is defined by the function g and the point $\boldsymbol{w}_0$ and consists of the (possibly open) connected set of points in the domain $\mathcal{D}$ which are reachable by a non-ascending trajectory from $\boldsymbol{w}_0$. Under this definition, two local minimum regions $M_g(\boldsymbol{u})$ and $M_g(\boldsymbol{v})$ are distinct, and so contain distinct local minima, if they do not intersect.

The minimum region $M_g(\boldsymbol{w}_0)$ is the (possibly open) connected set of points w for which there exists a non-ascending trajectory $A_{\boldsymbol{w}_0 \rightarrow \boldsymbol{w}}$.

It should now be noted that for functions defined on Euclidean space which are bounded below and everywhere differentiable, regional minima only exist with values corresponding to stationary points of the function. It may readily be seen that asymptotic regional minima correspond to stationary points if the function is bounded below, since the function must approach a limit value as the infinite boundary of the domain is approached by a non-ascending trajectory. Finite regional minima are also seen to correspond to stationary points since a finite regional minimum has a finite point with the minimum value. That point is a relative local minimum and hence a stationary point since the function is everywhere differentiable. This result is relevant to the analysis of feedforward neural networks since the error function is normally defined on a Euclidean weight space and is bounded below and everywhere differentiable. Hence in order to characterize the regional minima of the XOR error surface, it is only necessary to consider the stationary points.

An important contribution of the study by Leonard G.C.Hamey mentioned in [5] is the Approximation Lemma that states as below:

Let $A_{\boldsymbol{a} \rightarrow \boldsymbol{b}}$ be a non-ascending trajectory which passes through **d** (Figure 8). Let $B_{\boldsymbol{a} \rightarrow \boldsymbol{b}}$ be an arbitrary smooth trajectory which does not have any points other than **a** and **b** in common with *A*. The approximation lemma guarantees the existence of both a point **q** where $g(\boldsymbol{p}) = g(\boldsymbol{d})$ and a

non-ascending trajectory $A^*_{a \to q}$. The trajectory A^* has no points in common with the subtrajectory $A_{d \to b}$ except possibly the end point **b** (and then only if $g(\boldsymbol{b}) = g(\boldsymbol{d})$).

The proof could be found in the Section 6.2 of the work by Leonard G.C.Hamey cited in [5].

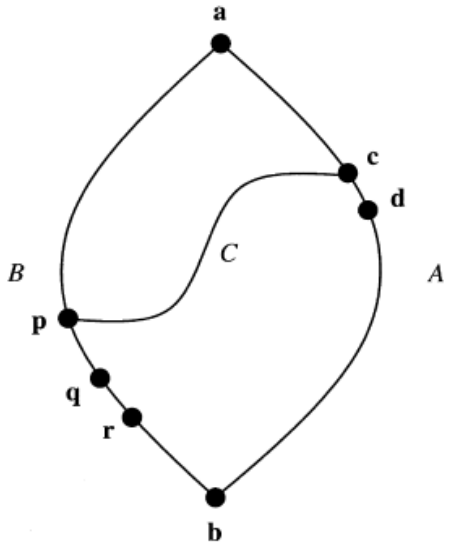


Figure 8: Development of the approximation lemma.

As a part of Section 5 of the study by Leonard G.C.Hamey mentioned in [5], the author has developed non-ascending trajectories from the stationary points to points with reduced error. Using this premise, the author has come up with this following important theorem, whose proof could be found in Section 6.4 of the mentioned work.

The XOR task with two hidden nodes has no regional local minima.

A direct consequence of this theorem is that, for every finite weight configuration of the XOR task with two hidden nodes, there exists a non-ascending trajectory to a global minimum of error. Perhaps the most obvious implication of the present result is that local minima are not as common as once thought. The strong evidence of entrapment in learning the XOR task has previously been taken as evidence of a local minimum, but this judgement has now been shown to be incorrect.

5.4. Visualizing the Loss Landscape of Neural Nets

Neural-network training requires minimizing a high-dimensional non-convex loss function. Visualizations have the potential to help us answer trivial questions about how neural networks work. To clarify questions on generalization of minima and how the loss function was able to minimize, high-resolution visualizations are used by Hao Li, Zheng Xu, Gavin Taylor, Christopher Studer and Tom Goldstein in their work cited [1] to provide empirical characterization of neural loss function and how the choices effect the landscape. They also explore how trainability and geometry of neural minimizers (i.e, sharpness/flatness, surrounding landscape relates to the non-convex nature of neural loss function and how they affect the generalization properties.

For this, a "filter normalization" scheme is used that enables to do side –by-side comparisons of different minima found during training. Visualizations are then used to explore sharpness/flatness of minimizers found by different methods and the effect of network architecture choices.

5.4.1. The Basics of Loss Function Visualization

One simple and lightweight way to plot loss functions is to choose parameter vectors θ *and* $\theta'$and plot the value of loss function along the line connecting these two points. We can parameterize using a scalar parameter α and weighted average $\theta(\alpha) = (1 - \alpha)\theta + \alpha\theta'$. Finally We plot the function $f(x) = L\left(\left(\theta(\alpha)\right)\right)$. This strategy was taken by Goodfellow et al. [10]. This method is widely used to study the "flatness" or "sharpness" of minima and its dependence on batch-size. However, it was difficult to visualize non-convexities using 1-D plots and this method does not consider batch-normalization or invariance symmetries in the network.

Contour Plots and Random Directions: To use this, a center point is chosen as θ^* and two direction vectors as δ and γ and one plots the function of the form:

$$f(\alpha, \beta) = L(\theta^* + \alpha\delta + \beta\gamma)$$

This explores trajectories of minimization methods and also shows that different optimization algorithms find different local minima within the 2D projected space.

5.4.2. Proposed Visualization: Filter-wise normalization.

This study heavily relies on contour plots and random directions approach, using random direction vectors α and γ each sampled from a random Guassian distribution with appropriate scaling. Because of the scale invariance in weights, this approach fails to capture intrinsic geometry of loss surfaces and cannot be used to compare the geometry of two different minimizers.

To remove this scaling effect, the authors plot loss functions using filter-wise normalized directions. To obtain such directions for a network with parameters θ, we begin by producing a random Guassian direction d with dimensions compatible with θ. We make replacement as $d_{i,j} \leftarrow \left(\frac{d_{i,j}}{\|d_{i,j}\|}\left\|\theta_{i_1 j}\right\|\right)$ where $d_{i,j}$ represents the *j*th filter of the *i*th layer of d and $\|\,.\,\|$ denotes the Frobenius norm. Note that filter-normalization is not limited to convolutional (Conv) layers but also applies to fully connected (FC) layers. The FC layer is equivalent to Conv layer with a 1x1

output feature map and the filter corresponds to the weights that generate the neuron.

5.4.3. What makes Neural Networks Trainable? Insights on the (Non)Convexity Structures of Loss Surfaces.

Our ability to find global optimizers to neural loss is not universal. It depends strongly on the initial parameters from which training starts. Using visualization methods, we do an empirical study of neural architectures to explore why the non-convexity of loss functions seems to be problematic in some situations, but not in others.

Experimental setup: Three classes of neural networks are considered 1) ResNets that are optimized for performance on CIFAR-10. ResNet-20/56/110 is considered where each name is labelled with the number of layers it has, 2) "VGG-like" networks that do not contain shortcut/skip connections. These are produced by simply removing the shortcut connections from ResNets and being called as ResNet-20/56/110-noshort. 3) "Wide" ResNets that have more filters per layer than CIFAR-10 optimized networks. All the models are trained on CIFAR-10 dataset using SGD with Nestrov momentum, batch-size 128, and 0.0005 weight decay for 300 epochs. Learning rate was initialized as 0.1 and decreased by a factor of 10 at epochs 150, 225, and 275. High resolution 2d plots of the minimizers are shown in figure 9 and 10.

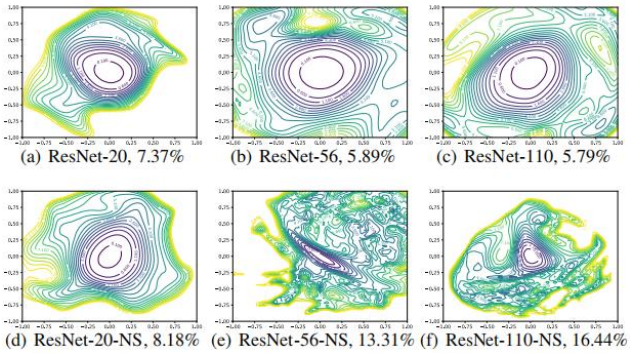


Figure 9: 2D visualization of the loss surface of ResNet-no short with different depth

The effect of Network Depth: From figure 9, we see that network depth has substantial effect on the loss surface of the neural network when skip connections are not used. as network depth increases, the loss surface of the VGG-like nets spontaneously transitions from (nearly) convex to chaotic. ResNet-56-noshort has substantial non-convexities and large regions where the gradient directions (which are normal to the contours depicted in the plots) do not point towards the minimizer at the center. Also, the loss function becomes extremely large as we move in some directions. ResNet-110-noshort displays even more substantial non-convexities, and becomes extremely steep as we move in all directions shown in the plot. Furthermore, note that the minimizers at the center of the deep VGG-like nets seem to be fairly sharp. In the case of ResNet-56-noshort, the minimizer is also fairly ill-conditioned, as the contours near the minimizer have significant eccentricity.

Shortcut Connections to the Rescue: Shortcut connections have a substantial effect of the geometry of the loss functions. In Figure 9, we see that residual connections prevent the transition to chaotic behavior as depth increases. In fact, the width and shape of the 0.1-level contour is almost identical for the 20- and 110-layer networks. Interestingly, the effect of skip connections seems to be most important for deep networks. For the shallower networks (ResNet-20 and ResNet-20-noshort), the effect of skip connections is fairly unnoticeable. However residual connections prevent the explosion of non-convexity that occurs when networks get deep.

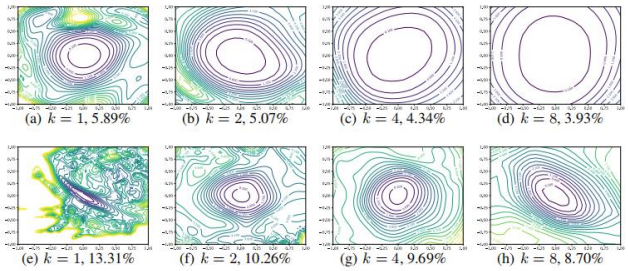


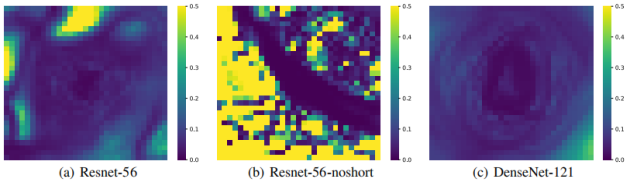
Figure 10: Wide-ResNet-56 on CIFAR-10 both with shortcut connections (top) and without (bottom).

Wide Models vs Thin Models: The authors compare the narrow CIFAR-optimized ResNets (ResNet-56) with Wide-ResNets[11] by multiplying the number of filters per layer by $k = 2$, 4, and 8. From Figure 10, we see that wider models have loss landscapes less chaotic behaviour. Increased network width resulted in flat minima and wide regions of apparent convexity. We see that increased width prevents chaotic behaviour, and skip connections dramatically widen minimizers. Finally, we see that sharpness correlates extremely well with test error.

Implications for Network Initialization: One interesting property seen in Figure 5 is that loss landscapes for all the networks considered seem to be partitioned into a well-defined region of low loss value and convex contours, surrounded by a well-defined region of high loss value and non-convex contours. This partitioning of chaotic and convex regions may explain the importance of good initialization strategies, and also the easy training behaviour of "good" architectures. When using normalized random initialization strategies such as those proposed by Glorot and Bengio [12], typical neural networks attain an initial loss value less than 2.5. The well behaved loss landscapes in Figure 9 (ResNets, and shallow VGG-like nets) are dominated by large, flat, nearly convex attractors that rise

to a loss value of 4 or greater. For such landscapes, a random initialization will likely lie in the "well- behaved" loss region, and the optimization algorithm might never "see" the pathological non-convexities that occur on the high-loss chaotic plateaus. Chaotic loss landscapes (ResNet-56/110-noshort) have shallower regions of convexity that rise to lower loss values. For sufficiently deep networks with shallow enough attractors, the initial iterate will likely lie in the chaotic region where the gradients are uninformative. In this case, the gradients shatter and training is impossible. SGD was unable to train a 156 layer network without skip connections (even with very low learning rates), which adds weight to this hypothesis.

Landscape Geometry Affects Generalization: Both Figures 9 and 10 show that landscape geometry has a substantial effect on generalization. First, note that visually flatter minimizers consistently correspond to lower test error, which further strengthens our assertion that filter normalization is a natural way to visualize loss function geometry. Second, we notice that chaotic landscapes (deep networks without skip connections) result in worse training and test error, while more convex landscapes have lower error values. In fact, the most convex landscapes (Wide-ResNets in the top row of Figure 10), generalize the best of all, and show no noticeable chaotic behaviour.



(a) Resnet-56 (b) Resnet-56-noshort (c) DenseNet-121

Figure 11: For each point in the filter-normalized surface plots, the authors calculate the maximum and minimum eigenvalue of the Hessian, and map the ratio of these two.

Looking at convexity: The authors view the loss surface under a dramatic dimensionality reduction, and we need to be careful interpreting these plots. For this reason, they quantify the level of convexity in loss functions but computing the *principle curvatures*, which are simply eigenvalues of the Hessian. A truly convex function has no negative curvatures (the Hessian is positive semi-definite), while a non-convex function has negative curvatures. It can be shown that the principle curvatures of a dimensionality reduced plot (with random Gaussian directions) are weighted averages of the principle curvatures of the full-dimensional surface (the weights are Chi-square random variables).

This has several consequences. First of all, if non-convexity is present in the dimensionality reduced plot, then non-convexity must be present in the full-dimensional surface as well. However, apparent convexity in the low-dimensional surface does not mean the high-dimensional function is truly convex. Rather it means that the positive curvatures are dominant (more formally, the *mean* curvature, or average eigenvalue, is positive). While this analysis is reassuring, one may still wonder if there is significant "hidden" non-convexity that these visualizations fail to capture. To answer this question, the authors calculate the *minimum* and *maximum* eigenvalues of the Hessian, λ_{min} and λ_{max}.4 Figure 11 maps the ratio $|\lambda_{min}/\lambda_{max}|$ across the loss surfaces studied above (using the same minimizer and the same random directions). Blue colour indicates a more convex region (near-zero negative eigenvalues relative to the positive eigenvalues), while yellow indicates significant levels of negative curvature. We see that the convex-looking regions in our surface plots do indeed correspond to regions with insignificant negative eigenvalues (i.e., there are not major non-convex features that the plot missed), while chaotic regions contain large negative curvatures. For convex-looking surfaces like DenseNet, the negative eigenvalues remain extremely small (less than 1% the size of the positive curvatures) over a large region of the plot.

The authors presented visualization techniques that provide insights into the consequences of a variety of choices faced by a neural network practitioner, including network architecture and batch size.

5.5. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density

To understand the dynamics of optimization in deep neural networks, Behrooz Ghorbani, Shankar Krishnan and Ying Xiao in their work cited [2] develop a tool to study the evolution of entire Hessian spectrum throughout the optimization process. The authors study a number of hypotheses concerning smoothness, curvature and sharpness in the deep learning literature. They analyze a crucial feature of the spectra in non: in on-batch normalized networks and observe the rapid appearance of large isolated eigenvalues in the spectrum along with a surprising concentration of the gradient in the corresponding eigen spaces. They introduce Hessian spectrum ImageNet-scale neural networks to characterize and explain these effects and how they affect optimization speed.

5.5.1. Accurate and Scalable Estimation of Hessian Eigenvalue Densities for $n > 10^7$

To understand the Hessian, the authors compute the eigenvalue (or spectral) density, defined as $\phi(t) = \frac{1}{n}\delta(t - \lambda_i)$ where δ is the Dirac delta operator. The naïve approach requires λ_i; however when the number of parameters, n is large , this is not tractable. They relax the problem with Guassian density of variance , σ^2 to obtain:

$$\phi_\sigma(t) = \frac{1}{n}\sum_{i=1}^{n} f(\lambda_i; t, \sigma^2) \qquad (6)$$

Where $f(\lambda_i; t, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(-t-\lambda)^2}{2\sigma^2}\right)$. For small enough $,\sigma^2$, $\phi_\sigma(t)$, provides all practically relevant information regarding the eigenvalues of *H*. Explicit representation of the Hessian matrix is infeasible when *n* is large, but using Pearlmutter's trick [19] we are able to compute Hessian-vector products for any chosen vector.

Algorithm 1 Two Stage Estimation of $\varphi\sigma(t)$
Draw *k* i.i.d realizations of *v*, {*v*1, . . . , *vk*}.
Step 1: Estimate $\varphi(vi)\ \sigma\ (t)$ by a quantity $\hat{\phi}^{(vi)}(t)$:

- Run the Lanczos algorithm(Golub, G. H. and Welsch, J. H. cited [16]) for *m* steps on matrix *H* starting from *vi* to obtain tridiagonal matrix *T*.
- Compute eigenvalue decomposition $T = ULU^T$.
- Set the nodes $`i = L_{ii}$ and weights $\omega_i = U^2_{1,i}$.
- Output $\hat{\phi}^{(v_i)}(t) = \sum_{i-}^{m} \omega i\, f(l_i; t, \sigma^2)$

Step 2: Set $\hat{\phi}_\sigma(t) = \frac{1}{i,} \sum_{i-1}^{k} \hat{\phi}^{(v_i)}(t)$

5.5.2. Implementation, Validation and Runtime

The authors implemented a large scale version of Algorithm 1 in TensorFlow (Abadi et al. [13]); the main component is a distributed Lanczos Algorithm. To validate their system, they computed the exact eigenvalue distribution on the 15910 parameter MNIST model. Their proposed framework achieves $L_1(\phi_\sigma, \hat{\phi}_\sigma) \equiv \int_{-\infty}^{\infty} |(t)\phi_\sigma - \hat{\phi}_\sigma(t)|\, dt \approx 0.0012$ which corresponds to an extremely accurate solution. The largest model they've run the algorithm on is Inception V3 on ImageNet. The runtime is dominated by the Hessian-vector products within the Lanczos algorithm; we run *mk* full-batch Hessian vector products. The remaining cost of full reorthogonalization is negligible ($O(km^2n)$ floating point operations). For a Resnet-18 on ImageNet, running a single draw takes about half the time of training the model.

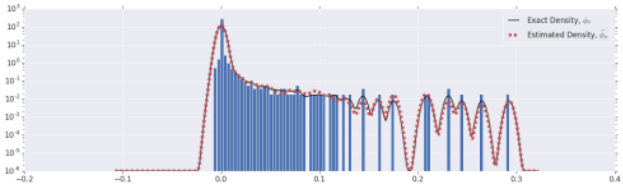


Figure 12. Comparison of the estimated smoothed density (dashed) and the exact smoothed density (solid) in the interval [[0.2, 0.4]]$\sigma^2 = 10^{-5}$, k = 10 and degree 90 quadrature. For completeness, the histogram of the exact eigenvalues is also plotted.

5.5.3. Spectral Densities throughout training

Observations made include that in Resnt-32 after just 400 momentum steps, large eigen values disappear (fig. 13). There is notable negative density towards the end of optimization (fig.14). Spectral densities of Resnet-32 preceding and following a learning rate decrease (at step 40000). The Hessian prior to the learning rate drop appears sharper (fig. 15). The Hessian without residual connections appears to be smoother (fig.16).

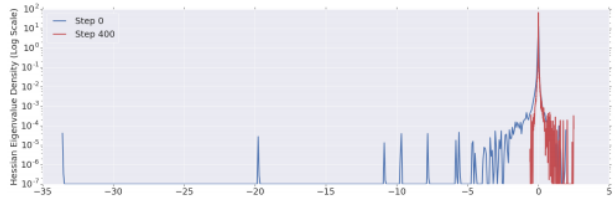


Figure 13. The evolution of the spectrum of a Resnet-32 in the beginning.

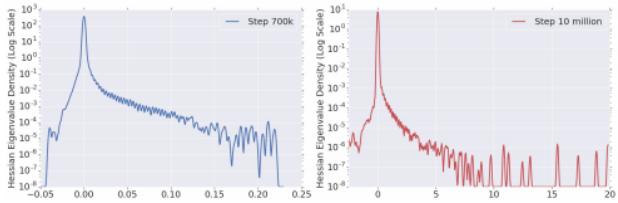


Figure 14. Spectral densities of Resnet-18 towards the beginning and end of optimization.

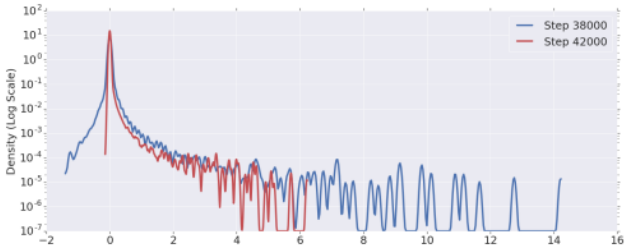


Figure 15. Spectral densities of resnet-32 preceding and following a learning rate decrease.

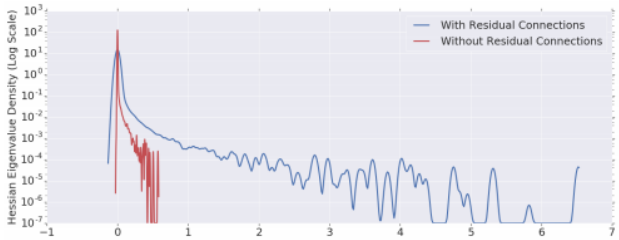


Figure 16. Spectral Densities of Resnet-32 with and without residual connections (at step 40000).

5.5.3. Outlier Eigenvalues Slow Optimization: Batch Norm Suppresses Outliers

Experiments by the authors reveal that the model without batch normalization(red) consistently shows much higher eigenvalue fraction (fig. 17). In contrast with non-BN networks, outliers grow much larger and further from the bulk. Normalization layer induces an odd dependency on parameter scale-scaling the (batch normalized) weights lead to unchanged activations, and inversely scales the gradients. Thus, for studying the optimization performance of batch normalization, the authors suggest of having at least a scale-invariant quantity – which is the metric of an outlier as : $\zeta(t) \coloneqq \lambda_1\left(\nabla^2 \mathcal{L}(\theta_t)\right) / \lambda_k\left(\nabla^2 \mathcal{L}(\theta_t)\right)$.

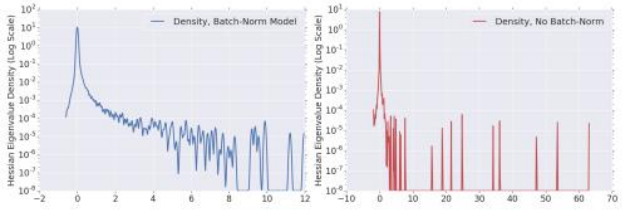


Figure 17. The eigenvalue comparison of the Hessian of the Resnet- 32 model with BN (blue) and without BN (red). To allow comparison on the same plot, the densities have been normalized by their respective 10th largest eigenvalue. The Hessians are computed after 48k steps of training.

5.5.4. Mechanisms by which outliers Slow Optimization.

"*Why* do outlier eigenvalues slow optimization?" One answer to this question is obvious. Large λ_1 implies that one must use a very low learning rate; but this an incomplete explanation – λ_1 has to be large *with respect to the rest of the spectrum*. To make this explicit, consider a simple quadratic approximation to the loss around the optimum,

$$\theta^*: \mathcal{L}(\theta) \approx \mathcal{L}(\theta^*) + \frac{1}{2}(\theta - \theta^*) \qquad (7)$$

where without loss of generality, assuming $H = diag(\lambda_1, \cdots, \lambda_n)$ with each element > 0. I can be easily shown that when optimized with gradient descent with a learning rate $\eta < 2/\lambda_1$ sufficiently small for convergence, we have:

$$\left|\hat{\theta}_t - \theta^*|_i \leq |1 - \frac{2\lambda_i}{\lambda_1}\right| |\hat{\theta}_t - \theta^*|_i \qquad (8)$$

For all directions where λ_i is small with respect to λ_1, we expect convergence to be slow. One might hope that these small λ_i do not contribute significantly to the loss; unfortunately, when the authors measure this in a Resnet-32 with no batch normalization, a small ball around 0 accounts for almost 50% of the total L_1 energy of the Hessian eigenvalues for a converged model (the L_1 reflects the loss function $\Sigma_i \lambda_i (\theta - \theta^*)_i^2$). Therefore, for successful optimization, one is forced to optimize these slowly converging directions.

A second, more pernicious reason lies in the interaction between the large eigenvalues of the Hessian and the stochastic gradients. Define the gradient covariance at time t to be $\sum(t) = \frac{1}{N}\sum_{i=1}^{N} \nabla \mathcal{L}_i \nabla \mathcal{L}_i^T$. The eigenvalue density of Σ characterizes how the energy of the (mini-batch) gradients is distributed. As with the Hessian, we observe that in non-BN networks the spectrum of Σ has outliers (Figure 18). In addition, the authors numerically verify that the outlier subspaces of H and Σ mostly coincide: throughout the training, for a Resnet-32, 99% of the energy of the outlier Hessian eigenvectors lie in the outlier subspace of Σ. Moreover, we observe that almost all of the gradient energy is concentrated in these subspaces. The authors observe that when BN is introduced in the model, this concentration subsides substantially.

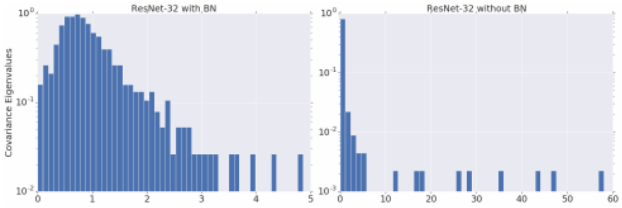


Figure 18. The histogram of the eigenvalues of Σ for a Resnet-32 with (left) and without (right) BN after 9k training steps. In no BN case, almost 99% of the energy is in the top few subspaces. For easier comparison, the distributions are normalized to have the same mean.

The authors observe that magnitude of the largest eigen value of the Hessian in between the two models is roughly same throughout training. Given that full batch BN network trains much more slowly, this observation shows that analysis based on the top eigen value of the Hessian does not provide full picture of the optimization hardness.

5.6. Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling.

The following work has been studied from the PhD thesis of Marcus R. Gallagher cited at [6]. Here we discuss the nature of various error surfaces that one encounters as a machine learning practitioner.

5.6.1. Linear Networks:

Baldi and Hornik [14, 15] have proven that for a linear MLP network, the error surface has a unique global minimum, and that all other critical points are saddle points. This result holds for any number of hidden layers. The error surface is not convex in this general case, but for the simpler case with no hidden layers (i.e. a single layer of weights), Baldi and Hornik show that the error surface is strictly convex [15]. The authors also obtain results for linear

autoassociative, or *autoencoder* networks, which are related to the calculation of principal components.

5.6.2. Exclusive OR (XOR):

The analysis of the XOR error surface indicates that local minima are not the cause of poor training performance for algorithms such as backpropagation on this problem. Other features of the error surface such as saddle points and plateaus, seem more likely explanations of training difficulties.

5.6.3. Auto-encoders:

A simple encoder is the linear encoder (which can be reduced to a single hidden layer, if a dimensionality reduction is to be performed), as mentioned in Section 3.5 above. For such a network, it can be shown that the error surface has a unique minimum [14, 15] corresponding to the projection onto the subspace generated by the first principal components (PC's) or "principal vectors" of a covariance matrix associated with the training patterns.

5.6.4. The Importance of Local Minima

The most well-known difficulty that arises in general optimization problems is the issue of local minima. Mathematical programming and optimization research was originally concerned with univariate problems (finding the minimum of a function of a single variable), or with solving systems of equations or inequalities involving only a few variables. In the one-dimensional case, the concept of a local minima follows closely from the issue of convexity. The conceptual picture is that if there are no local minima, then the optimization problem is trivial, and the cost function resembles a parabolic bowl or single valley.

The author says that while local minima are often discussed in describing the actions of training algorithms and other observations in practical scenarios (particularly in the early literature), often the observations in question are due to factors other than local minima. These factors are discussed in the following section.

5.6.5. Ill-Conditioning and Properties of the Gradient

The author says that the rate of change of the gradient, that is properties of the second-derivatives (Hessian matrix) of the error surface determines its relative steepness or flatness. If all the eigenvalues λj of the Hessian are equal, the Hessian is a diagonal matrix, corresponding to an error surface which is (locally) a perfectly circular bowl, with cross-sections of the error surface being hyperspheres in N-dimensional space. A gradient descent algorithm with step size $\eta = 1/\lambda j$ will reach the minimum in a single step. When the eigenvalues have different values, the error surface assumes a locally elliptical shape, with the slope in some directions being greater than in others. The behaviour of a gradient descent algorithm depends on how well-matched η is to the λj .

Too large a step size will result in oscillations across the contours of the surface and may cause divergence. On the other hand, too small a step size will make convergence to the minimum very slow.

Clearly, the condition number of the Hessian (i.e. ratio of eigenvalues $\lambda max/\lambda min$) is a significant factor in how different training algorithms negotiate the error surface. Algorithms that do not use gradient information directly, will be affected implicitly through their reliance on the values of the error function, which will vary according to this ratio. Algorithms such as Quasi-Newton (QN) and Levenberg-Marquardt, which use second-order information, may not converge much faster than gradient methods in such a situation, and due to their increased computation effort may actually result in slower execution times [18]. When this ratio is "large", the Hessian/error surface is said to be ill-conditioned.

The results discussed by the author in this section provide a description of the structural features or properties of the error surface in MLP training situations. The error surface is a high-dimensional search space which is everywhere differentiable and non-negative. Symmetries exist and produce a high degree of redundancy on the error surface, such as spherical throngs of stationary points. Local minima are not a major feature of MLP error surfaces in the sense that is often perceived from optimization problems in other areas. The main geometrical features are large, flat regions, some asymptotically approaching infinity, as well as step-like transitions, narrow valleys and ridges. The Hessian matrix is often ill-conditioned, meaning that the presence of flat and steep regions of the error surface is very prominent. These changes of gradient may vary over several orders of magnitude. The effects of ill-conditioning provide a satisfactory explanation for many of the difficulties associated with backpropagation and other training algorithms. More complex structure in the error surface can be thought of partly as the superposition of a number of these features.

5.6.6 Fitness Distance Correlation

The author also discusses one correlation measure which can be adapted to continuous error surfaces, that is the Fitness Distance Correlation (FDC) of Jones [17]. FDC is intended to provide a measure of the global structure of the surface in question, by examining how the value of the cost (fitness) function varies with the distance between a given point and a global optimum. A sample of random points on the surface is chosen, and the standard sample correlation coefficient is calculated:

$$r = \frac{cov(D, E)}{\sigma_D \sigma_E} \tag{9}$$

where, *cov*(D,E) is the covariance of D and E, D is a set of

distances for a sample of points, E is the corresponding set of fitness function values for the sample and σ_X is the standard deviation of X.

The relationship between error and distance from a global minimum of the error surface can be investigated using the fitness-distance correlation (FDC) from evolutionary computation, with the restriction that the location of the global minimum is known a *priori*. FDC is calculated experimentally in the student-teacher learning paradigm, by the author and the scatterplots reveal key features of the error surface, such as error plateaus, globally bowl-like regions and transitions in error values as a function of distance.

5.7. Asymmetric Valleys: Beyond Sharp and Flat Local Minima

The work that we would present in this section is a summary of the work by Haowei He et al. citied in [3]. The authors speak about the Stochastic Gradient Descent's working in spite of non-convex nature of the loss surface.

Traditionally, loss surfaces were known to be mostly sharp or flat. It was observed, however, that modern deep networks are more than being flat or sharp with mild assumptions. Specifically, at a local minimum there exist many asymmetric directions such that the loss increases abruptly along one side, and slowly along the opposite side. Such surfaces are asymmetric valleys. It was observed that the local geometry of the loss function of neural networks is usually asymmetric. In other words, there exist many directions such that the loss increases abruptly along one side, and grows rather slowly along the opposite side. The authors also argue that flatness does affect generalization. They observe that flat minima tend to generalize better.

Before moving into the actual discussion about this work, we would like to mention a couple of definition formally. The first is the asymmetric direction and the other being asymmetric valleys.

Intuitively, asymmetric direction can be understood as a direction u along which the loss function grows at different rates at the positive/negative direction. Formally, it can be stated as following:

Given constants $p > 0, r > \zeta > 0, c > 1$, a direction u is (r, p, c, ζ) - asymmetric with respect to point $w \in \mathbb{R}^d$ and loss function $\hat{L}$, if $\nabla_l \hat{L}(w + lu) < p$ and $\nabla_l \hat{L}(w - lu) < -cp$ for $l \in (\zeta, r)$.

The constant ζ handles the small neighbourhood around w with very small gradients.

An asymmetric valley is defined formally as the following:

Given constants $p, r > \zeta > 0, c > 1$, a local minimum $\widehat{w}^*$ of $\hat{L} \in \mathbb{R}^d \rightarrow \mathbb{R}$ is a (r, p, c, ζ) – asymmetric valley if there exists at least one direction u such that u is (r, p, c, ζ) – asymmetric with respect to $\widehat{w}^*$ and $\hat{L}$.

The authors have shown that when the loss landscape of a local minimum is asymmetric, a solution with bias towards the flat side of the valley has better generalization performance. They have then provided algorithm to achieve this task. In order to do so, they have taken the average of SGD iterates during the course of training. This was analyzed in two ways: one-dimensional case and higher dimensional cases.

In the one-dimensional case, we know that for asymmetric functions, as long as the learning rate is not too small, SGD will oscillate between the flat side and the sharp side. For each round of oscillation, the average iterates in each round has a bias on the flat side. Consequently, it was then generalized that by aggregating all rounds of oscillation, averaging SGD iterates leads to a bias as well. This observation was formalized as the following theorem:

Assume that a local minimizer $w^* = 0$ is a $(r, a_+, c, 0)$ – asymmetric valley, where $b_- < \nabla L(w) \leq a_- < 0$ for $w < 0$, and $0 < b_+ \leq \nabla L(w) \leq a_+$ for $w \geq 0$. Assume -$a_- = ca_+$ for a large c and $-\frac{b_- - v}{b_+} = c' < \frac{e^{\frac{c}{3}}}{6}$. The SGD updating rule is $w_{t+1} = w_t - \eta(\nabla L(w) + w_t)$ where w_t is the noise and $|w_t| < v$, and assume $v \leq a_+$. Then we have

$$\mathbb{E}[\bar{w}] > c_0 > 0,$$

where c_0 is a constant that only depends on η, a_+, a_-, b_+, b_-, and v.

This theorem can be intuitively seen in Figure 19.

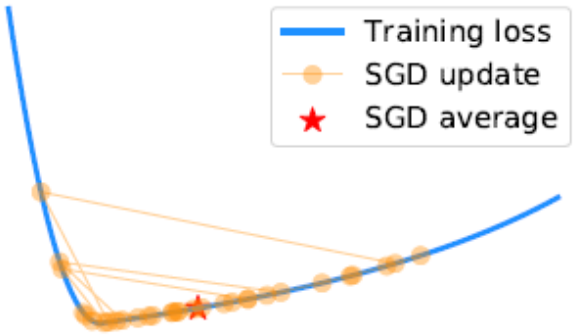


Figure 19: SGD iterates and their average on an asymmetric function: the oscillation case.

A similar argument can stand for the high-dimensional space also. More information about this could be found in Section 5.2 of Haowei He et al. cited in [3].

The authors have further mention that the reason for the very formation of the asymmetric valleys is because of the usage of Batch Normalization in newer Deep Networks. An important observation that was made in the study is that the proposed methodology works more effectively on the batch normalization parameters. This suggests that usage of batch normalization would only increase the generalization of the neural network SGD and the above approach of using the

SGD average helps in even better and quicker generalization.

6. Conclusion

As a part of our research, we have presented multiple works related to the loss surfaces of neural networks that have a lot importance in understanding the working of Deep Learning. In most of the cases, we have considered that Gradient Descent and its various flavours have provided us with very decent results. We also see that batch-normalized approach speeds up the training process as in 5.4. While there are more studies that are currently being worked on to understand the loss surfaces, we have tried to present a catalogue that would not only include the current works, but also the works of legacy nature, which form the solid bedrock of the current research, which in turn must work with more complicated, non-convex loss surfaces.

References

[1] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the Loss Landscape of Neural Nets. Advances in Neural Information Processing Systems 31 (NIPS 2018).

[2] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An Investigation into Neural Net Optimization via Hessian Eigenvalue Density. Proceedings of the 36th International Conference on Machine Learning, PMLR 97:2232-2241, 2019.

[3] Haowei He, Gao Huang, and Yang Yuan. Asymmetric Valleys: Beyond Sharp and Flat Local Minima. International Conference on Machine Learning, 2019.

[4] An Mei Chen, Haw-minn Lu, and Robert Hecht-Nielsen. On the Geometry of Feedforward Neural Network Error Surfaces. Neural Computation - MIT Press, 5(6):910-927, 1993.

[5] Leonard G.C.Hamey. XOR has no local minima: A case study in neural network error surface analysis - Elsevier. 11(4):669-681, 1998.

[6] Marcus R. Gallagher. Multi-layer Perceptron Error Surfaces: Visualization, Structure and Modelling. PhD Thesis, Department of Computer Science and Electrical Engineering, University of Queensland, 2000.

[7] Im, Daniel & Tao, Michael & Branson, Kristin. An Empirical Analysis of Deep Network Loss Surfaces, 2006.

[8] Wojciech Marian Czarnecki, Simon Osindero, Razvan Pascanu, and Max Jaderberg. A Deep Neural Network's Loss Surface Contains Every Low-dimensional Pattern. arXiv:1912.07559v2, 2020.

[9] Ivan Skorokhodov, and Mikhail Burtsev. Loss Landscape Sightseeing with Multi-Point Optimization, 2019.

[10] Ian J. Goodfellow, Oriol Vinyals, and Andrew M. Saxe. Qualitatively characterizing neural network optimization problems. Proceedings of the International Conference on Learning Representations (ICLR), 2015.

[11] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[12] Xavier Glorot and Yoshua Bengio. Understanding the diffificulty of training deep feedforward neural networks,In *AISTATS*, 2010

[13] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. Tensorflflow: a system for large-scale machine learning. In *OSDI*, volume 16, pp. 265–283, 2016.

[14] P. Baldi and K. Hornik. Neural networks and principal component analysis: Learning from examples without local minima. Neural Networks, 2(1):53–58, 1989.

[15] P. Baldi and K. Hornik. Learning in linear networks: a survey.IEEE Transactions on Neural Networks, 6(4):837–858, 1995

[16] Golub, G. H. and Welsch, J. H. Calculation of gauss quadrature rules. Mathematics of computation, 23(106):221–230, 1969.

[17] T. Jones. Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico, 1995.

[18] M. Lehr. Scaled Stochastic Methods for Training Neural Networks. PhD thesis, Stanford University, January 1996.

[19] Pearlmutter, B. A. Fast exact multiplication by the hessian. Neural computation, 6(1):147–160, 1994.