UNIVERSIDAD DE BUENOS AIRES
Facultad de Ciencias Exactas y Naturales
Departamento de Matemática

Tesis de Licenciatura

The Complexity of Deep Structures

Martin Arjovsky

Director: Pablo Groisman
Director: Yoshua Bengio

Fecha de Presentación

Contents

Chapter 1

Introduction

How to make models of the world based on experience and data instead of human experts is one of the most important challenges today. Given the massive amount of data available on the web and on domains such as medicine, vision, speech and natural language, we need fully automated methods to learn from this knowledge. Machine learning techniques and especially deep neural networks have been tremendously succesful to this effect in recent years. However, the reason of this success remains a partial mistery. While neural networks have the capacity to potentially capture any phenomenon, finding the network that best fits the current data or problem (also called learning) was thought to be an impossible quest. Learning in this setting is posed as minimizing a certain error function between what our model predicts and what our data tells us the world to be. A good model of the world is found by minimizing this error function. As it was seen in the 80s and 90s, these functions are full of local minima and are incredibly nonconvex, so the fact that in a lot of cases they be succesfully minimized by current algorithms remains an open intrigue.

However, very recently, some work has started to shed some light on this. Some models in physics whose Hamiltonians resemble the cost function of neural networks have been close to fully studied, and have some very relevant properties. The core of it is the fact that, while there are an exponential number of local minima, they all appear to be in some small error range, and close to the global minimum. However, some other interesting problems arize, such as saddle point proliferation: the fact that there are exponentially more saddle points than local minima as we increase the dimension of the problem.

This thesis provides a review on the methods and consequences of this recent work, and states it's connections to deep learning. Furthermore, we study how these properties impact optimization algorithms for neural networks, and we devise a variant that's well suited to fully take advantage of them.

This paper is divided in three parts (without counting this tiny introduction).

In Chapter 2, we introduce the machine learning problem, with a focus on function approximation and especially on deep learning. Readers who are already confident with the material can skip the chapter. We recommend the excelent textbooks [MRT12], and [GBC16] for a complete view of the foundations of machine learning and deep learning respectively. For further read on optimization, we refer the reader to the great paper [BCN16].

In Chapter 3, we display the techniques that start to attack the problem of studying the complexity of deep structures. We focus mainly on spin glass models, because they are the most understood. In this scenario, it's possible to provide concrete qualitative and quantitative estimates of relevant quantities. Later, we provide parallels with deep linear networks and multilayer perceptrons. Finally, we state clearly the similarities and differences between the well studied models and the optimization problem posed in learning deep neural networks. We state concretely which are the next problems to be tackled, where current methods might fall and where they might succeed. The main paper refered by this section is [AAC13], and to a minor extent [CHM+15] (even though we employ a different approach), and [DPG+14].

Finally, in Chapter 4, we study optimization algorithms under the scope provided by the theory. We show why Newton type methods fail catastrophically at learning deep neural networks. We study the computational complexity of different alternatives, and by joining these ideas we create a new second order optimization algorithm. This novel algorithm is designed to dismay the computational complexity in traditional second order methods, and is designed to be especially well suited for the nonconvex optimization problems arizing in deep learning. This part of the work will cover mainly [Arj15], [DPG+14], [Mar10], [ABB00], and [Pea94].

We conclude in Chapter 5, with a brief summary of results, contributions, and future work.

Chapter 2

Machine Learning Basics

Even though machine learning today encompasses many different areas, all machine learning algorithms share one core principle: using experience to improve performance. This experience typically comes in the form of data. Depending on the context, this data might have different shapes or sizes. What type of experience is available and how we measure performance defines the problem we are trying to solve.

2.1 Generalities

In what we call supervised learning, we typically have an unknown distribution $\mathcal{D}$ with support on $\mathcal{X} \subseteq \mathbb{R}^d$ and an unknown function $f : \mathcal{X} \to \mathcal{Y}$ with $\mathcal{Y} \subseteq \mathbb{R}$. Our task is, given a new x sampled from $\mathcal{D}$, to return $f(x)$. This is, we want to approximate this f (our target function) by some other function $h : \mathcal{X} \to \mathcal{Y}$ called a hypothesis. Because of the setting, this version of supervised learning falls under the broad class of function approximation problems, which are essentially about figuring out how to approximate a function of which we have limited information.

As an example, $\mathcal{X}$ might be the space of k-by-l natural images, and $f(x)$ might be the answer to "does image x have an animal?". If we call $d = k \times l$, we can represent $\mathcal{X} = [0, 255]^d$ as the vector of pixel intensity values. Similarly, we can say $\mathcal{Y} = \{0, 1\}$ with $f(x) = 0$ meaning there is no animal in the image x, and $f(x) = 1$ meaning there is one.

Even though the function f is unknown, we do have access to some data of the form of a set $\{x^{(i)}, y^{(i)}\}_{i=1}^n \subseteq \mathbb{R}^d \times \mathbb{R}$, where the $x^{(i)}$ are iid samples of $\mathcal{D}$, and $y^{(i)} = f(x^{(i)})$. In our little example, this can mean that we took a set of pictures and manually wrote down for each one if there is an animal in it or not.

The next step is to measure performance. If we have some notion of difference (not necessarily a metric) $l : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ called the *loss*, ideally we would like to measure our approximation based on our *expected loss* under $\mathcal{D}$. I.e.

$$L(h) = \mathbb{E}_{x\sim\mathcal{D}}\left[l(h(x), f(x))\right]$$

Since both $\mathcal{D}$ and f are unknown, this approximation is intractable. What we do have access to is an unbiased estimator of this, called *empirical loss*:

$$\hat{L}(h) = \frac{1}{n}\sum_{i=1}^{n} l(h(x^{(i)}), y^{(i)})$$

It is important to notice also that the empirical loss is the expected loss under the empirical distribution. If we use $l(\hat{y}, y) = (\hat{y} - y)^2$ we get the usual mean squared error:

$$\hat{L}(h) = \frac{1}{n}\sum_{i=1}^{n} (h(x^{(i)}) - y^{(i)})^2$$

However, many different losses can be used (such as any p-distance). In our example, if we expand $\mathcal{Y}$ to be $[0, 1]$, we can interpret $h : \mathcal{X} \to [0, 1]$ as the probability of the image containing an animal. Therefore, we can take l to be the negative log-likelihood of this model:

$$\begin{aligned} l(\hat{y}, y) &= -\left(y\log(\hat{y}) + (1-y)\log(1-\hat{y})\right) \\ \hat{L}(h) &= \frac{1}{n}\sum_{i=1}^{n} -\left(y^{(i)}\log(h(x^{(i)})) + (1-y^{(i)})\log(1-h(x^{(i)}))\right) \end{aligned}$$

We will usually consider consider $h \in \mathcal{H}$ in a specific class of functions we know how to compute. For example, we can consider the class of linear models $\mathcal{H} = \{h_{(w,b)}(x) = w^T x + b : w \in \mathbb{R}^d, b \in \mathbb{R}\}$. Usually, h can be parameterized by a vector $\theta \in \mathbb{R}^m$. In the linear case, clearly, $\theta = (w, b)$. Therefore, the expected and empirical losses can be thought of as functions of θ instead of h. We therefore would like to solve the following optimization problem:

$$\min_{\theta\in\mathbb{R}^m} L(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[l(h_\theta(x), f(x)] \tag{2.1}$$

Furthermore, in our examples, $l(\hat{y}, y)$ and $h_\theta(x)$ are infinitely differentiable as functions of $\hat{y}$ and (θ, x) respectively. From now on, we will asume that they belong to C^k with $k > 1$. Therefore, by differentiation under the integral sign

$$\begin{aligned} \nabla_\theta L(\theta) &= \mathbb{E}_{x\sim\mathcal{D}}[\nabla_\theta l(h_\theta(x), f(x))] \\ &\sim \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta l(h_\theta(x^{(i)}), y^{(i)}) \\ &= \nabla_\theta \hat{L}(\theta) \end{aligned}$$

Where the tilde is meant to mean an unbiased estimate. Since the negative gradient is the direction of maximum descent, the basic (and widely used) procedure followed to solve this problem is: initialize θ_0 at random and iterate

$$\theta_{k+1} \leftarrow \theta_k - \alpha \nabla_\theta \hat{L}(\theta_k) \tag{2.2}$$

where α is a parameter of the algorithm called the *learning rate*. When the full dataset is used to measure the gradient of the empirical risk, this method is known as *Batch Gradient Descent*. When the set of samples used for the gradient is changed at every iteration, this approach is known as *Stochastic Gradient Descent*, popularly abbreviated by SGD.

If $l(h_\theta(x), y)$ is a convex function on θ, then L and $\hat{L}$ are convex as well, since integrating on another variable preserves convexity. The most important property to us about convex functions is that they only have one critical point, which is the global minimum [BV04]. If $l(h_\theta(x), y)$ is strongly convex in θ (i.e. the Hessian has it's eigenvalues bounded below by a constant larger than 0) then you can prove that batch gradient descent converges exponentially fast to the global minimum of $\hat{L}$ [BCN16]. What this means is that if $\hat{L}_*$ is the global minimum of $\hat{L}$ then $(\hat{L}(\theta_k) - \hat{L}_*) \in \mathcal{O}(\gamma^k)$ for some $\gamma < 0$.

Without assuming strong convexity, [BCN16] proves that for any $\epsilon > 0$ there is a learning rate α such that if you use SGD then

$$\mathbb{E}[\|\nabla_\theta L(\theta_k)\|_2^2] \in \mathcal{O}\left(\epsilon + \frac{1}{k}\right)$$

It is crucial to notice that this is the expected loss, the function we trully want to minimize. Furthermore, you can show that by diminishing α carefully at each iteration, you can achieve $\|\nabla_\theta L(\theta_k)\| \overset{\mathbb{P}}{\to} 0$. Using more involved techniques such as the ones found in the foundational paper [RM51] and in [RS85], one can achieve almost sure convergence.

The main point to carry about this last bound is that gradient methods make the gradient go to zero. For a convex function, this means that we are reaching a critical point. But the only critical point is the global minimum! Therefore, gradient methods are extremelly useful on minimizing convex functions. You can easily show that if you're optimizing over linear models, for most loss functions, the expected and empirical losses end up being convex [BV04].

However, the functions we typically want to aproximate are incredibly complicated and nonlinear, so linear models are not expressive enough. In our example of going from images to having an animal in the picture, it is very unlikely that a linear combination of pixel intensities will be able to determine this, and more complex models are needed [KSH12].

If the loss is nonconvex, then we have no guarantee that the critical point found is the global minimum. Furthermore, it doesn't even have to be a local minimum, it can be a saddle point. Under (albeit strong) assumptions, [LSJR16] shows that batch gradient descent converges to a local minimum of the empirical risk. However, does the local minimum found have low cost? Is it close to the global minimum? Do other methods fall in saddle points or minima? Do saddle points have low error? These are the questions we attempt to answer in this thesis.

To do that, we study a widely successful class of models that have arbitrary expressive power, known as deep neural networks.

2.2 Deep Learning

Deep neural networks are a broad class of families of functions. In here, we will focus mainly on *deep feedforward neural networks*, also called *multilayer perceptrons*, often abreviated as MLPs. MLPs are functions of the form

$$h_\theta(x) = w^T(g^{(p)} \circ F^{(p)} \circ \cdots \circ g^{(1)} \circ F^{(1)})(x) + b \tag{2.3}$$

where $F^{(i)} : \mathbb{R}^{d_{i-1}} \to \mathbb{R}^{d_i}$ is an affine transformation of the form $F^{(i)}(x) = A_{(i)}x + b_{(i)}$. As well, $g^{(i)} : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$ is a **pointwise** (coordinatewise) function. We will do a small abuse of notation and call the function $g^{(i)} : \mathbb{R} \to \mathbb{R}$ applied on each coordinate with the same name.

This type of network is called a p-layer MLP, and each operation $g^{(i)} \circ F^{(i)}$ is called a *hidden layer*, or simply layer. Note that 0-layer MLPs are exactly the linear models. Typically, networks have $g^{(i)} = g$ for all layers. These g's are called *activation functions*, and are usually required to be differentiable almost everywhere and continuous, so it's possible to do gradient descent on the network. Typical activation functions are

- Rectified linear units (ReLUs): $g(x) = \max(0, x)$.
- Sigmoid units: $g(x) = \frac{1}{1+e^{-x}}$.
- Tanh units: $g(x) = \tanh(x)$.

For fixed p and $d_1, \ldots, d_p$, neural networks are parameterized by

$$\theta = (A_{(1)}, b_{(1)}, \ldots, A_{(p)}, b_{(p)}, w, b) \in \mathbb{R}^m$$

which are called the weights.

To highlight the capabilities of multilayer perceptrons, we first state the universal approximation theorem. This theorem shows that any continuous function defined on a compact set can be approximated arbitrarily well by a neural network.

Theorem 2.2.1 (Universal Approximation)**.** *Let $g : \mathbb{R} \to \mathbb{R}$ be a nonconstant, bounded, and monotonically increasing function. Let $f : \mathcal{X} \to \mathbb{R}$ be any continuous function on* *the compact set $\mathcal{X} \subseteq \mathbb{R}^d$. Then, for any $\epsilon > 0$ there is a one-hidden layer neural network h_θ with activation g such that*

$$\|f - h_\theta\|_\infty < \epsilon$$

Proof. The proof is a straightforward application of the Stone-Weierstrass theorem and can be found in [Hor91]. □

While the universal approximation theorem is a nice guarantee, the size of the hidden layer grows very fast as ϵ goes down, so it's necessary to ask wether these methods work in practice.

The theorem states that there is a θ that achieves low error, but whether we can actually find it is a whole other story. We can apply the chain rule to equation (2.3) and therefore calculate the gradient $\nabla_\theta \hat{L}(\theta)$ to do batch or stochastic gradient descent.

Let m be the dimension of θ and n the amount of examples used to estimate $\nabla_\theta \hat{L}(\theta)$ each time. The efficient implementation of the chain rule is popularly known as *back-propagation* [RHW86], and it's computation cost is $\mathcal{O}(mn)$ per iteration, which means that it can be scaled to extremely large networks and number of examples. Furthermore, in the SGD setting, the number of samples **per iteration** is very small ($n = 32$ is very usual), so the computational efficiency allows to use massive models, representing very complex functions. For example, $m > 10^8$ is fairly common [HZRS16] [TYRW14] [DCM+12].

However, it is possible to show that the problem (2.1) of minimizing the expected loss is incredibly nonconvex, and that the number of local minimum can grow exponentially with m on some corner cases [AHW96].

Somewhat surprisingly then, we are able to find a plethora of modern results showing the wide applicability of deep neural networks trained by gradient descent. These models have surpased dramatically the state of the art on complex problems such as object and face recognition [KSH12] [TYRW14], speech recognition [CJLV16], machine translation and language modelling [SVL14] [JVS+16], robotics [TDH+16], and beating world champions in Go and Atari [SHM+16] [MKS+13].

Therefore, why is it that we can achieve low loss, even on the prescence of an overwhelming number of local minimum (and perhaps saddle points)? Does this mean that some local minima are hard to get stuck on? Maybe they have low error? Do all critical points have low error? Are we avoiding saddle points? How close (in terms of loss) are different critical points to the global minimum?

The next chapter formalizes some of this questions in a setting close to learning deep networks, and answers them.

Chapter 3

Theory

This chapter will mainly be devoted to the study of spin glasses. These are a class of cost functions that share a lot of similarities with the empirical risk when the hypothesis is a neural network. We will study the (random) number of critical points on spin glasses, and their values with concrete estimates to provide a clear picture of the loss surface.

Since to provide parallels between spin glasses and neural networks we first need to become ourselves familiar with the former, we leave the analysis of the similarities and differences to section 3.5.

3.1 Spin Glasses

We define the Hamiltonian of the p-spin glass model as the random function defined on the sphere $H_{N,p} : S^{N-1}(\sqrt{N}) \to \mathbb{R}$ such that

$$H_{N,p}(\sigma) = \frac{1}{N^{(p-1)/2}} \sum_{i_1,\dots,i_p=1}^{N} x_{i_1,\dots,i_p} \sigma_{i_1} \dots \sigma_{i_p}$$

where $x_{i_1,\dots,i_p}$ are standard Gaussian independent random variables.

For these models, we are interested in examining the number and nature of critical points. To do this, we introduce the definition of the *index* of a critical point. The index of a critical point is the number of negative eigenvalues in the Hessian at that point. For a critical point σ, it's index $i(\sigma)$ (also noted $i(\nabla^2 H_{N,p})$) counts the number of descent directions among the Hessian's eigenvectors. Trivially, for a minimum the index is 0, and N for a strict maximum.

Let $B \subseteq \mathbb{R}$ be a Borel set and $k \in \mathbb{N}$, then we define by $\mathrm{Crt}_{N,k}$ the (random) number of critical points of index k whose value lies in $NB = \{Nx : x \in B\}$. Analogously, we define $\mathrm{Crt}_N(B)$ as the (random) number of critical points whose value lies in NB

$$\mathrm{Crt}_{N,k}(B) = \sum_{\sigma : \nabla H_{N,p}(\sigma)=0} \mathbb{1}\{H_{N,p}(\sigma) \in NB\}\mathbb{1}\{i(\nabla^2 H_{N,p}(\sigma)) = k\}$$
$$\mathrm{Crt}_N(B) = \sum_{\sigma : \nabla H_{N,p}(\sigma)=0} \mathbb{1}\{H_{N,p}(\sigma) \in NB\}$$

Let $u \in \mathbb{R}$ be a number, we also note $\mathrm{Crt}_{N,k}(u) = \mathrm{Crt}_{N,k}((-\infty, u))$ and $\mathrm{Crt}_N(u)$ analogously. It is important to stop for a second and think about what these quantities mean. The quantity $\mathrm{Crt}_{N,0}(u)$, for example, tells us the mean number of minima under a certain threshold. If we can find out structure on the values of u's as a function of $\mathrm{Crt}_{N,0}(u)$, we can gain information on the value of local minima.

The following section states and proves the *central identity*, which provides a formula for $\mathbb{E}[\mathrm{Crt}_{N,k}(B)]$. The core of the proof is the Kac-Rice formula, which will be introduced in a general fashion, highlighting it's potential applicability to other models, possibly ones closer to deep neural networks.

Once we have a formula for $\mathbb{E}[\mathrm{Crt}_{N,k}(B)]$, we show on section 3.3 how to provide estimates for the number of critical points below a certain value, and finally on 3.4 how to characterize the values of critical points of a given index. This is the section that will study, for example, the question of whether all local minima have low error, which is of crucial importance.

3.2 The Kac-Rice Formula

The Kac-Rice formula is an extremely general construct that allows one to calculate the expected number of zeros of a random function f defined on a Riemannian manifold. We can also calculate the expected number of zeros t such that the value of another random function $h(t)$ lies in a borel set B. The formula will be expressed mainly in terms of the densities of f, the determinant of ∇f and h. Setting $f = \nabla H_{N,p}$, the critical points of $H_{N,p}$ will be the zeros of f, and we can set $h = (H_{N,p}, \nabla^2 H_{N,p})$ to lie on $NB \times \{\mathbf{A} \subseteq \mathbb{R}^{N\times N} : i(\mathbf{A}) = k\}$ for the formula to yeld $\mathbb{E}[\mathrm{Crt}_{N,k}(B)]$. For the case of spin glasses, all the densities involved are known, so precise computations can be done.

We now state the Kac-Rice formula as in [AT07] in its more general fashion, to highlight that it covers cases far more general than where f and h are simple Gaussian fields, such as in spin glasses.

Given a Riemannian n-manifold M with a Riemannian metric g, we define an orthonormal framing (or frame) to be $(E_i)_{1\le i\le n}$ a set of vector fields such that for every $x \in M$, $\{E_i(x)\}_{i=1}^n$ is an orthonormal basis of (T_xM, g_x). In that case, ∇f_E denotes the vector field whose coordinates are given by $(\nabla f_E)_i \equiv E_i(f)$.

Theorem 3.2.1 (Kac-Rice formula). *Let M be a compact, oriented, N-dimensional C^1 manifold with a C^1 Riemannian metric g. Let $f = (f^1, \dots, f^N) : M \to \mathbb{R}^N$ and $h = (h^1, \dots, h^k) : M \to \mathbb{R}^K$ be random fields on M. For an open set $B \subseteq \mathbb{R}^K$ for which ∂B has dimension $K-1$ and a point $u \in \mathbb{R}^N$, let*

$$N_u \equiv N_u(M) \equiv N_u(f, h; M, B)$$

denote the number of points $t \in M$ for which

$$f(t) = u \qquad \text{and} \qquad h(t) \in B$$

Assume the following conditions are satisfied for some orthonormal frame field E:

- *All components of f, ∇f_E and h are a.s. continuous and have finite variances over M.*
- *For all $t \in M$, the marginal densities $p_t(x)$ of $f(t)$ (implicitly assumed to exist) are continuous ar $x = u$.*
- *The conditional densities $p_t(x|\nabla f_E(t), h(t))$ of $f(t)$ given $h(t)$ and $(\nabla f_E)(t)$ (implicitly assumed to exist) are bounded above and continuous at $x = u$, uniformly in $t \in M$.*
- *The conditional densities $p_t(z|f(t) = x)$ of $\det(\nabla f^i_{E_j}(t))$ given $f(t) = x$ are continuous for z and x in neighbourhoods of 0 and u, respectively, uniformly in $t \in M$.*
- *The conditional densities $p_t(z|f(t) = x)$ of $h(t)$ given $f(t) = x$ are continuous for all z and for x in a neighbourhood u, uniformly in $t \in M$.*
- *The following moment condition holds*

$$\sup_{t \in M} \max_{1 \le i,j \le N} \mathbb{E}\left[|\nabla f^j_{E_i}(t)|^N\right] < \infty$$

- *The moduli of continuity with respect to the canonical metric induced by g of each component of h, each component of f, and each $\nabla f^j_{E_i}$ all satisfy for any $\epsilon > 0$*

$$\mathbb{P}\left(\omega(\eta) > \epsilon\right) = o\left(\eta^N\right) \qquad \text{as} \qquad \eta \downarrow 0$$

Then

$$\mathbb{E}[N_u] = \int_M \mathbb{E}\left[|\det(\nabla f_E)| \mathbb{1}_B(h) \middle| f = u\right] p_t(u)\, Vol_g \tag{3.1}$$

where p_t is the density of f at point t and Vol_g the volume element on M induced by the metric g.

Proof. The proof (while important) is quite involved and beyond the scope of this thesis. We therefore refer the avid reader to [AT07] (Theorem 12.1.1) for more details. □

As a corollary for the Gaussian case, we have:

Corollary 3.2.1. *Let (M, g) be a Riemannian manifold satisfying the conditions of Theorem 3.2.1. Let f and h be centered Gaussian fields over M. Then, if f, h and ∇f_E are a.s. continuous over M, and if for each $t \in M$, the joint distributions of $(f(t), \nabla f_E(t), h(t))$ are nondegenerate, then* (3.1) *holds.*

We now turn back to the case of spin glass models, where the Kac-Rice formula and some work will allow us to gain an operable expression (deemed the *central identity* for $\mathbb{E}[\mathrm{Crt}_{N,k}(B)]$. Before stating and proving the central identity we need some preliminary notation.

The Gaussian orthogonal ensemble (GOE) is the probability distribution of an $N \times N$ symmetric matrix M such that its entries are independent centered Gaussians with variance $\mathbb{E}M_{i,j}^2 = \frac{1+\delta_{i,j}}{2N}$. We note $\mathbb{E}_{\mathrm{GOE}} = \mathbb{E}_{\mathrm{GOE}}^N$ the expectation under the GOE measure. We call $\lambda_0^N \leq \lambda_1^N \leq \cdots \leq \lambda_N^N$ the ordered eigenvalues of M, $L_N = \frac{1}{N}\sum_{i=0}^N \delta_{\lambda_i^N}$ the (random) spectral measure of M, and $\rho_N(x)$ the density of the (non-random) probability measure $\mathbb{E}_{\mathrm{GOE}}(L_N)$ [1]. The function $\rho_N(x)$ is usually called the (normalized) one-point correlation function and satisfies

$$\int_{\mathbb{R}} f(x)\rho_N(x)dx = \frac{1}{N}\mathbb{E}_{\mathrm{GOE}}^N\left[\sum_{i=0}^{N-1} f(\lambda_i^N)\right]$$

We are now ready to state the central identity.

Theorem 3.2.2 (Central identity). *Let $N, p \geq 2, k \in \{0, \dots, N-1\}$ be natural numbers and $B \subseteq \mathbb{R}$ a Borel set. Then*

$$\mathbb{E}[\mathrm{Crt}_{N,k}(B)] = 2\sqrt{\frac{2}{p}}(p-1)^{\frac{N}{2}}\mathbb{E}_{GOE}^N\left[e^{-N\frac{p-2}{2p}(\lambda_k^N)^2}\mathbb{1}\left\{\lambda_k^N \in \sqrt{\frac{p}{2(p-1)}}B\right\}\right] \tag{3.2}$$

Summing over k, we also arrive at the identity for the mean number of critical points

Theorem 3.2.3. *The following identity holds for all $N, p \geq 2$, and for all Borel sets $B \subseteq \mathbb{R}$*

$$\mathbb{E}[\mathrm{Crt}_N(B)] = 2N\sqrt{\frac{2}{p}}(p-1)^{\frac{N}{2}}\int_{\sqrt{\frac{p}{2(p-1)}}B} \exp\left(-N\frac{p-2}{2p}x^2\right)\rho_N(x)\,\mathrm{d}x$$

[1] This density exists because λ_i^N are absolutely continuous random variables [Tao12]. Let B be a Borel set with Lebesgue measure 0, then $\mathbb{E}[L_N(B)] = \frac{1}{N}\sum_{i=0}^N \mathbb{E}[\delta_{\lambda_i^N}(B)] = \frac{1}{N}\sum_{i=0}^N \mathbb{P}(\lambda_i^N \in B) = 0$.

The first step to proving the central identity is to use the Kac-Rice formula for our particular case, as we hinted before. We therefore state it as a lemma.

Let $(E_i)_{1\le i<N}$ be an orthonormal framing of S^{N-1}. We write ϕ_σ for the density of the gradient vector $(E_i(\sigma)(f))_{1\le i<N}$ and $\det \nabla^2 f(\sigma)$ the determinant of the matrix $(\nabla^2 f(E_i,E_j)(\sigma))_{1\le i,j<N}$.

Lemma 3.2.1. *Let f be a centered random Gaussian field on S^{N-1} and let $\mathcal{A} = (U_\alpha, \psi_\alpha)_{\alpha\in I}$ be a finite atlas on S^{N-1}. Set $f^\alpha = f\circ\psi_\alpha^{-1} : \psi_\alpha(U_\alpha) \subseteq \mathbb{R}^{N-1} \to \mathbb{R}$ and define $f_i^\alpha = \partial f^\alpha/\partial x_i$, $f_{i,j}^\alpha = \partial^2 f^\alpha/\partial x_i \partial x_j$. Assume that for all $\alpha\in I$ and all $x,y\in\psi_\alpha(U_\alpha)$ the joint distribution of $(f_i^\alpha(x), f_{i,j}^\alpha(x))_{1\le i\le j<N}$ is non-degenerate, and*

$$\max_{i,j} |\mathrm{Var}(f_{i,j}^\alpha(x)) + \mathrm{Var}(f_{i,j}^\alpha(y)) - 2\mathrm{Cov}(f_{i,j}^\alpha(x), f_{i,j}^\alpha(y))| \le K_\alpha |\ln|x-y||^{-1-\beta}$$

for some $\beta>0$ and $K_\alpha>0$. For a Borel set $B\subseteq\mathbb{R}$, let

$$\mathrm{Crt}_{N,k}^f = \sum_{\sigma:\nabla f(\sigma)=0} \mathbb{1}\{f(\sigma)\in B, i(\nabla^2 f(\sigma)) = k\}$$

Then, using $\mathrm{d}\sigma$ to denote the usual surface measure on S^{N-1},

$$\mathbb{E}[\mathrm{Crt}_{N,k}^f(B)] = \int_{S^{N-1}} \mathbb{E}\left[|\det\nabla^2 f(\sigma)| \mathbb{1}\{f(\sigma)\in B, i(\nabla^2 f(\sigma)) = k\} \mid \nabla f(\sigma) = 0\right] \phi_\sigma(0)\,\mathrm{d}\sigma \tag{3.3}$$

Proof. Equation (3.3) is just a renaming of (3.1) when we calculate the zeros of ∇f and set $h(\sigma) = (f(\sigma), \nabla^2 f(\sigma))$ to lie in the borel set $B\times\{\mathbf{A}\subseteq\mathbb{R}^{(N-1)\times(N-1)} : i(\mathbf{A}) = k\}$. The checking of the conditions of Theorem 3.2.1 can be seen in the proof of Lemma 3.1 in [AAC13]. □

For the rest of the proof of theorem 3.2.2 we will work with the normalized version of $H_{N,p}$, defined as

$$f_{N,p}(\sigma) = \frac{1}{\sqrt{N}} H_{N,p}(\sqrt{N}\sigma)$$

This makes it easier to work with, since (as it is easily checked), the Gaussian process $f_{N,p}$ has variance 1 at every $\sigma\in S^{N-1}$. Furthermore, we will typically drop the subscript and note $f\equiv f_{N,p}$.

We now give a lemma by [AAC13] that fully describes the joint density of $(f(\sigma), \nabla f(\sigma), \nabla^2 f(\sigma)$. Perhaps more importantly, it also describes precisely the density of the Hessian $\nabla^2 f(\sigma)$ conditioned on $f(\sigma) = x$.

Lemma 3.2.2. *Let $(f_i(\sigma))_{1\le i<N}$ be the gradient and $(f_{i,j}(\sigma))_{1\le i,j\le N}$ the Hessian matrix at $\sigma\in S^{N-1}$, that is $f_i = E_i f(\sigma), f_{i,j} = \nabla^2 f(E_i,E_j)(\sigma)$. Then, for all $1\le i,j,k<N$, $f(\sigma), f_i(\sigma), f_{j,k}(\sigma)$ are centered Gaussian random variables whose joint distribution is determined by*

- $\mathbb{E}[f(\sigma)^2] = 1$.
- $\mathbb{E}[f(\sigma)f_i(\sigma)] = \mathbb{E}[f_i(\sigma)f_{j,k}(\sigma)] = 0$.
- $\mathbb{E}[f(\sigma)f_{i,j}(\sigma)] = -p\delta_{i,j}$.
- $\mathbb{E}[f_i(\sigma)f_j(\sigma)] = p\delta_{i,j}$.
- $\mathbb{E}[f_{i,j}f_{k,l}] = p(p-1)(\delta_{i,k}\delta_{j,l} + \delta_{i,l}\delta_{j,k}) + p^2\delta_{i,j}\delta_{k,l}$.

As well, under the conditional distribution $\mathbb{P}[\cdot|f(\sigma) = x]$, $x \in \mathbb{R}$, the random variables $f_{i,j}(\sigma), 1 \leq i, j < N$, are independent Gaussian random variables satisfying

- $\mathbb{E}[f_{i,j}(\sigma)] = -xp\delta_{i,j}$.
- $\mathbb{E}[f_{i,j}(\sigma)^2] = (1 + \delta_{i,j})p(p-1)$.

Alternatively, the random matrix $(f_{i,j}(\sigma))$ has the same distribution as

$$M\sqrt{2(N-1)p(p-1)} - xp\mathbf{I}_{N-1} \tag{3.4}$$

where M is an $(N-1) \times (N-1)$ GOE, and $\mathbf{I}_{N-1}$ is the identity.

Before we dive into the proof, let's stop for one crucial bit of intuition. Equation (3.4) tells us that as the value of $f(\sigma)$ goes down (analogously, our cost decreases), the one but massively important change reflected in the Hessian is that it's eigenvalues shift to the right (increase). This means that as we go down in our cost, the index is likely to decrease, leaving less descent directions. Furthermore, this change is simple enough that it will allow us to invert this principle, and ask how the value $f(\sigma)$ changes statistically as we consider different types of critical points. For example, if we have a local minimum all the eigenvalues will be positive, which will make $x = f(\sigma)$ likely to be low. The precise formulation of this idea will be stated and proved in section 3.4.

Proof. Without loss of generality we can assume that σ is the north pole $\mathbf{n} = (0, \ldots, 0, 1)$ of S^{N-1}. This is because the distribution of f is trivially checked to be rotation-invariant. We define $\psi : S^{N-1} \to \mathbb{R}^{N-1}$ as the function that drops the last coordinate, which is a chart of $\mathbf{n}$ in a neighbourhood U. We set $\bar{f} = f \circ \psi^{-1}$, which is a Gaussian process on $\psi(U)$ with covariance

$$C(x,y) = \text{Cov}(\bar{f}(x), \bar{f}(y)) = \left(\sum_{i=1}^{N-1} x_i y_i + \sqrt{(1 - \sum_{i=1}^{N-1} x_i^2)(1 - \sum_{i=1}^{N-1} y_i^2)} \right)^p$$

What follows is the crucial step of the proof. Being $\bar{f}$ a Gaussian process in $\mathbb{R}^d$, we can easily calculate the joint of $(\bar{f}(\sigma), \nabla \bar{f}(\sigma), \nabla^2 \bar{f}(\sigma))$ via the formula (see [AT07], eq 5.5.4)

$$\operatorname{Cov}\left(\frac{\partial^k \bar{f}(x)}{\partial x_{i_1} \dots \partial x_{i_k}}, \frac{\partial^l \bar{f}(y)}{\partial y_{j_1} \dots \partial y_{j_l}}\right) = \frac{\partial^{k+l} C(x,y)}{\partial x_{i_1} \dots \partial x_{i_k} \partial y_{j_1} \dots \partial y_{j_l}} \tag{3.5}$$

More than that, we can choose an orthonormal frame (E_i) such that $E_i(\mathbf{n}) = \partial / \partial x_i$ with respect to ψ. Since the Christoffel symbols $\Gamma^i_{k,l}(\mathbf{n})$ are all equal to 0, then the Hessian $(f_{i,j}(\mathbf{n}))$ coincides with $(\bar{f}_{i,j}(0))$. Because of this, the only thing left in the proof is to verify that the Theorem holds for $\bar{f}$, which is easily checked via equation (3.5). The distribution of the conditional Hessian again can easily be computed since it is well known how Gaussian distributions change under conditioning (see for example [AT07] pages 10-11). □

Before we prove the central identity we require one final lemma regarding the determinant of the shifted GOE. Since the proof is mainly an algebraic manipulation, we will omit it and refer to the literature.

Lemma 3.2.3. *Let M be a $(N-1) \times (N-1)$ GOE matrix and X be an independent Gaussian normal random variable with mean m and variance t^2. Then, for any Borel set $G \subseteq \mathbb{R}$*

$$\begin{aligned}&\mathbb{E}\left[|\det(M - XI)| \mathbb{1}\{i(M - XI) = k, X \in G\}\right] \\ &= \frac{\Gamma\left(\frac{N}{2}\right)(N-1)^{-\frac{N}{2}}}{\sqrt{(\pi t^2}} \mathbb{E}^N_{\text{GOE}}\left[\exp\left(\frac{N(\lambda^N_k)^2}{2} - \frac{\left(\left(\frac{N}{N-1}\right)^{\frac{1}{2}} \lambda^N_k - m\right)^2}{2t^2}\right) \mathbb{1}\left\{\lambda^N_k \in \left(\frac{N-1}{N}\right)^{\frac{1}{2}} G\right\}\right]\end{aligned} \tag{3.6}$$

Proof. See [AAC13], proof of lemma 3.3. □

We are now ready to prove the central identity.

Proof of Theorem 3.2.2. We first check that f verifies the hypothesis of Lemma 3.2.1. We again take the chart of the north pole $\psi : U \subseteq S^{N-1} \to \mathbb{R}^{N-1}$ such that $\psi(x_1, \dots, x_N) = (x_1, \dots, x_{N-1})$. From Lemma 3.2.2, it is not hard to check that the joint distribution $(f_i(\sigma), f_{i,j}(\sigma))$ is non-degenerate for $\sigma = \mathbf{n}$. Since the covariances are continuous, it has to be non-degenerate in a neighbourhood U' of the north pole. However, this implies that it's non-degenerate in all S^{N-1} by covering the sphere with rotations of U' around the center of the sphere. Therefore, the conditions of Lemma 3.2.1 are satisfied.

By remembering that the distribution of $f(\sigma)$ is rotation invariant, the integrand of (3.3) doesn't depend on σ. Therefore, by (3.3) and the definitions of $\operatorname{Crt}_{N,k}(B)$ and f, we get

$$\mathbb{E}[\mathrm{Crt}_{N,k}(B)] = \omega_N \mathbb{E}\left[|\det \nabla^2 f(\mathbf{n})| \mathbb{1}\{i(\nabla^2 f(\mathbf{n})) = k\} \mathbb{1}\{f(\mathbf{n}) \in \sqrt{N}B\} \middle| \nabla f(\mathbf{n}) = 0 \right] \phi_{\mathbf{n}}(0) \tag{3.7}$$

where $\omega_N = \frac{2\phi^{N/2}}{\Gamma(N/2)}$ is the volume of the sphere S^{N-1}. Since the density of the gradient $\nabla f(\mathbf{n})$ is the same as the density of $(\bar{f}_i(0))_{1 \le i < N}$, then by Lemma 3.2.2 we have $\phi_{\mathbf{n}}(0) = (2\pi p)^{-(N-1)/2}$.

To compute the expectation in (3.7), we use the fact that, by 3.2.2, f and it's Hessian are independent of the gradient. After that, we condition on $f(\mathbf{n})$ and get

$$\begin{aligned} &\mathbb{E}\left[|\det \nabla^2 f(\mathbf{n})| \mathbb{1}\{i(\nabla^2 f(\mathbf{n})) = k\} \mathbb{1}\{f(\mathbf{n}) \in \sqrt{N}B\} \middle| \nabla f(\mathbf{n}) = 0 \right] \\ &\quad = \mathbb{E}\left[|\det \nabla^2 f(\mathbf{n})| \mathbb{1}\{i(\nabla^2 f(\mathbf{n})) = k\} \mathbb{1}\{f(\mathbf{n}) \in \sqrt{N}B\} \right] \\ &\quad = \mathbb{E}\left[\mathbb{E}\left[|\det \nabla^2 f(\mathbf{n})| \mathbb{1}\{i(\nabla^2 f(\mathbf{n})) = k\} \mathbb{1}\{f(\mathbf{n}) \in \sqrt{N}B\} \middle| f(\mathbf{n}) \right] \right] \end{aligned} \tag{3.8}$$

Applying Lemma 3.2.2, the interior expectation turns into

$$\begin{aligned} &\mathbb{E}\left[|\det \nabla^2 f(\mathbf{n})| \mathbb{1}\{i(\nabla^2 f(\mathbf{n})) = k\} \mathbb{1}\{f(\mathbf{n}) \in \sqrt{N}B\} \middle| f(\mathbf{n}) \right] \\ &= (2(N-1)p(p-1))^{\frac{N-1}{2}} \mathbb{E}_{\mathrm{GOE}}^{N-1}\Big[|\det(M - p^{1/2}(2(N-1)(p-1))^{-1/2} f(\mathbf{n}) \mathbf{I}_{N-1})| \\ &\times \mathbb{1}\left\{ i(M - p^{1/2}(2(N-1)(p-1))^{-1/2} f(\mathbf{n}) \mathbf{I}_{N-1}) = k, f(\mathbf{n}) \in \sqrt{N}B \right\} \Big] \end{aligned} \tag{3.9}$$

Plugging (3.9) into (3.8) we can use Lemma 3.2.3 with $m = 0$, $t^2 = \frac{p}{2(N-1)(p-1)}$ and $G = \sqrt{\frac{Np}{2(N-1)(p-1)}}B$. Plugging back this expectation and the value of $\phi_{\mathbf{n}}(0)$, ω_N, and after a bit of algebra we get

$$\mathbb{E}[\mathrm{Crt}_{N,k}(B)] = 2\sqrt{\frac{2}{p}}(p-1)^{\frac{N}{2}} \mathbb{E}_{\mathrm{GOE}}^{N}\left[e^{-N\frac{p-2}{2p}(\lambda_k^N)^2} \mathbb{1}\left\{ \lambda_k^N \in \sqrt{\frac{p}{2(p-1)}}B \right\} \right]$$

concluding the proof of the central identity. The proof of Theorem 3.2.3 is straightforward summing the central identity for all k and replacing the expectation of the empirical measure with an integral with the density ρ_N. □

3.3 The Complexity of Spin Glasses

We now use the central identity to provide estimates on the growth of the number of critical values. Let us define $E_\infty = E\infty(p) = 2\sqrt{\frac{p-1}{p}}$.

Let also $I_1 : (-\infty, -E_\infty] \to \mathbb{R}$ be given by

$$I_1(u) = \frac{2}{E_\infty^2} \int_u^{-E\infty} (z^2 - E_\infty^2)^{1/2} \, \mathrm{d}z = -\frac{u}{E_\infty^2} \sqrt{u^2 - E_\infty^2} - \log\left(-u + \sqrt{u^2 - E_\infty^2}\right) + \log E_\infty$$

Clearly I_1 is a strictly decreasing function, with $I_1(-E_\infty) = 0$ and $I_1(u) \in \mathcal{O}(u^2)$ as $u \to -\infty$

We now define the following two functions that will describe the asymptotic growth of the complexity of spin glass models.

$$\Theta_{k,p}(u) = \begin{cases} \frac{1}{2}\log(p-1) - \frac{p-2}{4(p-1)}u^2 - (k+1)I_1(u) & \text{if } u \leq -E_\infty \\ \frac{1}{2}\log(p-1) - \frac{p-2}{p} & \text{if } u \geq -E_\infty \end{cases}$$

$$\Theta_p(u) = \begin{cases} \frac{1}{2}\log(p-1) - \frac{p-2}{4(p-1)}u^2 - I_1(u) & \text{if } u \leq -E_\infty \\ \frac{1}{2}\log(p-1) - \frac{p-2}{4(p-1)}u^2 & \text{if } -E_\infty \leq u \leq 0 \\ \frac{1}{2}\log(p-1) & \text{if } 0 \leq u \end{cases}$$

Remembering that $\mathrm{Crt}_{N,k}(u) \equiv \mathrm{Crt}_{N,k}(B)$ for $B = (-\infty, u)$, we now state the logarithmic estimates for the complexity of spin glasses.

Theorem 3.3.1. *For all $p \geq 2$ and $k \geq 0$ fixed,*

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_{N,k}(u)] = \Theta_{k,p}(u)$$

Theorem 3.3.2. *For all $p \geq 2$,*

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_{N}(u)] = \Theta_{p}(u)$$

Corollary 3.3.1. *The mean number of critical points of a given index and the total mean number of critical points is given by*

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_{N,k}(\mathbb{R})] = \frac{1}{2}\log(p-1) - \frac{p-2}{p}$$
$$\lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_{N}(\mathbb{R})] = \frac{1}{2}\log(p-1)$$

A few observations to carry away from this last corollary. First of all, it is clear then that the number of critical points grows exponentially with the dimension N. Furthermore, the mean number of critical points of any given index grows exponentially with N, with a speed that doesn't depend (in these logarithmic estimates) on the index. This of course applies as well to $k = 0$, which tells us that the "fear" of the neural network community in the 90s that there are an exponential number of local minima may be correct. However, as we will see later, almost all of these have roughly the same error.

What's perhaps even more interesting is the study of the proportion of minima with respect to all critical points. A corollary that we devise follows

Corollary 3.3.2. *Let $N \geq 0$, $p > 2$, then*

$$\lim_{N\to\infty} \frac{1}{N} \log \frac{\mathbb{E}[\mathrm{Crt}_{N,0}(\mathbb{R})]}{\mathbb{E}[\mathrm{Crt}_{N}(\mathbb{R})]} = -\frac{p-2}{p} < 0$$

What this tells us is that **in proportion, there are exponentially more saddle points than minima**. This is a phenomenon we deem *saddle point proliferation*, and we'll go back to it's consequences on chapter 4.

We now turn ourselves to the proofs of Theorems 3.3.1 and 3.3.2. We first state a lemma that establishes a large deviation principle for the k-th largest eigenvalue of the GOE.

Lemma 3.3.1. *Let X be an $N \times N$ real symmetric random matrix whose entries $X_{i,j}$ are independent (up to symettry) centered Gaussian random variables with variance $\mathbb{E}X_{i,j}^2 = \sigma^2 N^{-1}(1+\delta_{i,j})$. Let $\lambda_1 \leq \cdots \leq \lambda_N$ be it's ordered eigenvalues.*

Then, for each fixed $k \geq 1$, the k-th largest eigenvalue λ_{N-k+1} of X satisfies an LDP with speed N and a good rate function

$$I_k(x;\sigma) = kI_1(x,\sigma) = \begin{cases} k\int_{2\sigma}^{x} \sigma^{-1}\sqrt{(\frac{z}{2\sigma})^2 - 1}\,\mathrm{d}z & \text{if } x \geq 2\sigma \\ \infty & \text{otherwise} \end{cases}$$

Proof. See [AAC13], Theorem A.1. □

Proof of Theorem 3.3.1. By Lemma 3.3.1 and the symmetry between the largest and the smallest eigenvalues, the $(k+1)$-th smallest eigenvalue λ_k^N of M a GOE, satisfies the LDP with the good rate function $J_k(u) = (k+1)I_1(-u; 2^{-1/2})$. If we set $t = u\sqrt{\frac{p}{2(p-1)}}$ and $\phi(x) = -\frac{p-2}{2p}x^2$, then by the central identity

$$\lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_{N,k}(u)] = \frac{1}{2}\log(p-1) + \lim_{N\to\infty} \log \mathbb{E}_{\mathrm{GOE}}^{N}\left[e^{N\phi(\lambda_k^2)}\mathbb{1}_{\lambda_k \leq t}\right] \tag{3.10}$$

By Vardhan's Lemma, we have

$$\begin{aligned} \sup_{x\in(-\infty,t)} (\phi(x) - J_k(x)) &\leq \liminf_{N\to\infty} \frac{1}{N} \log \mathbb{E}_{\mathrm{GOE}}^{N}\left[e^{N\phi(\lambda_k^2)}\mathbb{1}_{\lambda_k < t}\right] \\ &\leq \limsup_{N\to\infty} \frac{1}{N} \log \mathbb{E}_{\mathrm{GOE}}^{N}\left[e^{N\phi(\lambda_k^2)}\mathbb{1}_{\lambda_k \leq t}\right] \\ &\leq \sup_{x\in(-\infty,t]} (\phi(x) - J_k(x)) \end{aligned}$$

When $t \leq -\sqrt{2}$, both suprema are equal to $\phi(t) - J_k(t)$. If $t > -\sqrt{2}$, they are both equal to $\phi(\sqrt{2})$. By definition of ϕ, t, J_k, E_∞ and the fact that $I_k(x;\sigma) = I_k(x/\sigma;1)$, we have that (3.10) equals

$$\begin{cases} \frac{1}{2}\log(p-1) - \frac{p-2}{4(p-1)}u^2 - (k+1)I_1(u) & \text{if } u \leq -E_\infty \\ \frac{1}{2}\log(p-1) - \frac{p-2}{p} & \text{if } u \geq -E_\infty \end{cases}$$

concluding the proof. □

Proof of Theorem 3.3.2. Let t and ϕ be as in the previous proof. By Theorem 3.2.3 we have

$$\begin{aligned} \lim_{N\to\infty} \frac{1}{N} \log \mathbb{E}[\mathrm{Crt}_N(u)] &= \lim_{N\to\infty} \frac{1}{N} \log \left(2N\sqrt{\frac{2}{p}}(p-1)^{N/2} \int_{-\infty}^{t} e^{N\phi(x)} \rho_N(x)\,\mathrm{d}x \right) \\ &= \frac{1}{2}\log(p-1) + \lim_{N\to\infty} \frac{1}{N} \log \int_{-\infty}^{t} e^{N\phi(x)} \rho_N(x)\,\mathrm{d}x \end{aligned}$$

Let's study now the contribution by the integral. For $t \leq -\sqrt{2}$, using that $\mathbb{1}_{\lambda_0 \leq t} \geq \mathbb{1}_{\lambda_i \leq t}$ for any i and the definition of the spectral measure, we have

$$\frac{1}{N}\mathbb{E}\left[e^{N\phi(\lambda_0)}\mathbb{1}_{\lambda_0\leq t}\right] \leq \int_{-\infty}^{t} e^{N\phi(x)}\rho_N(x)\,\mathrm{d}x \leq \mathbb{E}\left[e^{N\phi(\lambda_0)}\mathbb{1}_{\lambda_0\leq t}\right]$$

Taking the logarithm and dividing by N, both sides of the inequality converge to $\phi(t) - I_1(-t; 2^{-1/2})$, by the same argument in the previous proof. Using the definition of t and ϕ, this proves the case for $u \leq -E_\infty$.

For $t \in (-\sqrt{2}, 0]$, we cannot use λ_0 in the lower bound. Because of that, we write for $\epsilon > 0$,

$$\frac{1}{N}\mathbb{E}\left[e^{N\phi(t-\epsilon)}\mathbb{1}_{L_N((t-\epsilon,t))>0}\right] \leq \int_{-\infty}^{t} e^{N\phi(x)}\rho_N(x)\,\mathrm{d}x \leq \mathbb{E}\left[e^{N\phi(t)}\mathbb{1}_{\lambda_0\leq t}\right]$$

By te LDP for λ_0, $\mathbb{P}(\lambda_0 \geq t) \to 1$ as $N \to \infty$. Similarly, since L_N converges to the semicircular law, we have $\mathbb{P}(L_N((t-\epsilon,t)) > 0) \to 1$ as $N \to \infty$. After taking the logarithm and dividing by N, we have

$$\phi(t-\epsilon) \leq \lim_{N\to\infty} \frac{1}{N} \log \int_{-\infty}^{t} e^{N\phi(x)}\rho_N(x)\,\mathrm{d}x \leq \phi(t)$$

Since ϕ is continuous, the claim for the theorem follows for $t \in (-\sqrt{2}, 0]$, or $u \in (-E_\infty, 0]$. The proof for $u > 0$ is analogous and left to the reader. □

3.4 Critical Points in Level Sets, Energy Landscapes

We now turn to the most interesting question, which is what is the value range of a specific type of critical point. We first define $E_k(p)$ to be the onle real value such that $\Theta_{k,p}(-E_k(p)) = 0$.

We first state without proof a theorem of the ground state (global minimum) of the Hamiltonian. Let $GS^N = \frac{1}{N} \min_{\sigma \in S^{N-1}(\sqrt{N})} H_{N,p}(\sigma)$.

Theorem 3.4.1. *For $p \geq 3$,*

$$\liminf_{N\to\infty} GS^N \geq -E_0(p)$$

Moreover, for $p \geq 4$ even,

$$\lim_{N\to\infty} GS^N = -E_0(p) \qquad \textit{in probability.}$$

Now, we state the main theorem regarding the value range of critical points.

Theorem 3.4.2. *Let for an integer $k \geq 0$ and $\epsilon > 0$, $B_{N,k}(\epsilon)$ be the event "there is a critical value of index k of the Hamiltonian $H_{N,p}$ above the level $-N(E_\infty(p) - \epsilon)$, that is $B_{N,k} = \{\mathrm{Crt}_{N,k}((-E_\infty(p) + \epsilon, \infty)) > 0\}$. Then, for all $k \geq 0$ and $\epsilon > 0$,*

$$\limsup_{N\to\infty} \frac{1}{N^2} \log \mathbb{P}(B_{N,k}(\epsilon)) < 0$$

This means that the probability of a critical point of fixed index (nondiverging with N) having value larger than $-N(E_\infty(p) - \epsilon)$ decreases exponentially. If we we consider the normalized value $f_{N,p}$ and index $k = 0$ we get that **the probability of a local minimum having cost outside of** $(\min_\sigma f_{N,p}(\sigma), -E_\infty(p) + \epsilon)$ **decreases exponentially with** N. Moroever, since GS^N is larger than $-E_0(p)$ with overwhelming probability, the value of a local minimum will lie in the range $(-E_0, -E_\infty)$ (again, with overwhelming probability). The power of this bound comes from the fact that the range $(-E_0, -E_\infty)$ is indeed very small. For example, for the case of $p = 3$, this range is close to $(-1.66, -1.63)$, which is an extremely small range considering that $f_{N,p}$ has variance 1 and mean 0. The conclusion is that as N increases, with overwhelming probability the local minima will have low error, which if translated to the case of neural networks, explains the massive success these algorithms have had in the past years.

Proof of Theorem 3.4.2. We again follow the proof by [AAC13]. We want to show that there are no critical points of a finite index of the Hamiltonian above the level $N(-E_\infty + \epsilon)$. Let k and ϵ be as in the statement of the theorem. Then, by Markov's inequality and the central identity,

$$\begin{aligned}\mathbb{P}(\mathrm{Crt}_{N,k}((-E_\infty+\epsilon,\infty))>0) &\leq \mathbb{E}[\mathrm{Crt}_{N,k}((-E_\infty+\epsilon,\infty))] \\ &\leq c(p)(p-1)^{N/2}\mathbb{E}\left[\exp\left(-\frac{N(p-2)(\lambda_k^2)}{2p}\right)\mathbb{1}\{\lambda_k \geq -\sqrt{2}+\epsilon'\}\right] \\ &\leq c(p)(p-1)^{N/2}\mathbb{P}(\lambda_k \geq -\sqrt{2}+\epsilon') \qquad (3.11)\end{aligned}$$

with $\epsilon' = \epsilon\sqrt{2}/E_\infty$. By the LDP for the empirical spectral measure L_N, we know for some $C(\epsilon') > 0$,

$$\mathbb{P}(\lambda_k \geq -\sqrt{2}+\epsilon) \leq e^{-C_\epsilon N^2}$$

Combining this last inequality with (3.11) concludes the proof. □

3.5 Deep Neural Networks and Next Steps

In these section, we (informally) state the differences and similarities between spin glasses and neural networks. We also conjecture how to adapt some methods of the proofs to the case of deep linear networks, and provide insights on to where the current methods might fail.

[CHM$^+$15] show striking similarities between the complexity of the empirical risk $\hat{L}$ and the one in the normalized version of the hypothesis for a neural network. This is supported as well by [SMG13]. Because of this, we concern ourselves with the landscape of the hypothesis classes and leave the learning problem setting to further work.

We now consider the Hamiltonian of the spin glass models as a hypothesis class parameterized by σ, but present it with the neural network notation.

In the following, p is the number of hidden layers. H is the number of hidden units per layer. $w_{i,j}^{(k)}$ is the weight from unit i in layer k to unit j in layer $k+1$. The second index is omitted for $k = p$ (i.e. a weight from the final hidden layer to the output), since the output is one dimensional. The input layer is indexed at 0, the first hidden layer at 1, and so on.

Spin Glasses

$$h_w(x) = \frac{1}{Z}\sum_{i_0,\ldots i_p=1}^{H} x_{i_0,\ldots,i_p} w_{i_1} w_{i_2}\ldots w_{i_p}$$

Where $Z = H^{p/2}$

Linear Networks

$$h_w(x) = \sum_{i_0,\ldots,i_p=1}^{H} x_{i_0} w_{i_0,i_1}^{(0)} w_{i_1,i_2}^{(1)}\ldots w_{i_{p-1},i_p}^{(p-1)} w_{i_p}^{(p)}$$

Multilayer Perceptrons

$$h_w(x) = \sum_{i_p=1}^{H} w_{i_p}^{(p)} \phi \left(\sum_{i_{p-1}=1}^{H} w_{i_{p-1},i_p}^{(p-1)} \phi \left(\cdots \sum_{i_1=1}^{H} w_{i_1,i_2}^{(1)} \phi \left(\sum_{i_0=1}^{H} w_{i_0,i_1}^{(0)} x_{i_0} \right) \right) \right)$$

Clearly if $\phi(x) = x$ then the multilayer perceptron becomes a linear network.

Observations

Both in spin glasses and linear networks, the hypothesis is an homogeneus polynomial of degree p with p^H terms and random coefficients. Most importantly, the hypothesis are linear in any variable fixed the other ones. This means the cost function's nonconvexity comes from the multiplicative interactions between the weights. Assuming the x_i's are centered Gaussians, both hypothesis are Gaussian processes with mean 0 and easy to calculate covariance kernels, which is a core assumption in the techniques.

If we normalize the weights on a deep linear network, we come to a surprising realization. The main tool used for the proofs was Lemma 3.2.2, which described the joint of $(f, \nabla f, \nabla^2 f)$. The proof of this lemma came by taking a chart ψ of the north pole that drops the last coordinate, and realizing that $\bar{f} = f \circ \psi^{-1}$ is a Gaussian process on $\mathbb{R}^N$. Since the Christoffel symbols at the north pole are 0, the Hessians from $\bar{f}$ and f coincide. Furthermore, there is a simple formula (3.5) for calculating the joint of $\bar{f}$, the problem is solved. The main realization is that this all holds true as well when f is the normalized version of the hypothesis of a linear network. The only core problem is that the joint distribution is not invariant under rotations so we can't cover the sphere in the same way. However, there are other close symmetries in the network, so it might still be possible to adapt the argument. It's likely that if this method is successful, the Hessian of f might again be close to a GOE, so it might be possible to apply Lemma 3.2.3 and the known LDPs to get similar results.

The main problem for extending these methods to MLPs is that the nonlinearities make the distribution completely non-Gaussian. While the Kac-Rice formula is likely to hold, all the distributions in it are not studied, so we don't have equivalent Lemmas. Similar results may come if we can describe the joint of $(f, \nabla f, \nabla^2 f)$ taking advantage of the hierarchical structure in MLPs. After all, by the change of variable theorem, we know how applying a pointwise nonlinearity and an affine transformation alters the distribution. However, none of the LDPs will hold, and it is likely that adapting them will require a significant effort.

However, it is extremely important to point out that even if there's an obvious difference in representational power from linear to nonlinear networks; the learning dynamics, the complexity, and the saddle point structure of both of them seem to be remarkably similar [SMG13].

Chapter 4

Algorithmic Implications

As we have seen in the previous section, optimization problems similar to the one in neural networks have some interesting properties. One of the main points to carry through is that there appears to be an exponentially high numer of saddle points in the loss surface. We also saw that typically all local minima are very close in error to the global minimum, so finding any minimum guarantees that we will have low cost with very high probability. In this chapter, we study the consequences of these theoretical results and intuitions on optimization algorithms, and propose some modifications that leverage these consequences.

4.1 Newton's Method

Newton's method [BV04] is probably the most studied optimization algorithm to present day. If you have a function $L : \mathbb{R}^m \to \mathbb{R}$ that you want to minimize, Newton's method proposes to initialize some point θ_0 at random, and then perform the following iteration until convergence:

$$\theta_{k+1} \leftarrow \theta_k - (\nabla^2_\theta L(\theta_k))^{-1} \nabla_\theta L(\theta_k) \tag{4.1}$$

where $\nabla^2_\theta L(\theta_k)$ is the Hessian matrix of L at point θ_k. When L is not accessible directly as in the case of function approximation, we can just use the Hessian and the gradient of the empirical loss $\hat{L}$ as we did with stochastic gradient descent.

If we compare updates (2.2) and (4.1) we see that the main difference is the addition of the Hessian term. While this may seem somewhat arbitrary, we show now that Newton's update has a very interesting and straightforward derivation.

Consider the second order Taylor of L around θ_k

$$\bar{L}(\theta) = L(\theta_k) + \nabla_\theta L(\theta_k)^T (\theta - \theta_k) + \frac{1}{2} (\theta - \theta_k)^T \nabla^2_\theta L(\theta_k)(\theta - \theta_k)$$

In the case L is a strictly convex function, $\bar{L}$ is strictly convex as well, and we can minimize it analitically by solving $\nabla_\theta \bar{L}(\theta) = 0$. But we have that

$$\nabla_\theta \bar{L}(\theta) = \nabla_\theta L(\theta_k) + \nabla_\theta^2 L(\theta_k)(\theta - \theta_k)$$

Therefore, $\nabla_\theta L(\theta) = 0$ if and only if

$$\theta = \theta_k - (\nabla_\theta^2 L(\theta_k))^{-1} \nabla_\theta L(\theta_k)$$

Which is exactly update (4.1). This way, Newton can be seen as succesively approximating the loss by it's second order Taylor, and minimizing it. Analogously, gradient descent can be seen as minimizing a first order approximation in a ball of radius λ for some $\lambda > 0$ that depends on the learning rate [BV04]. One would expect then for Newton's algorithm to work better than gradient descent, since it provides a better minimization of the "surrogate" loss function at each step. In fact, it is easy to show under mild assumptions that for a strongly convex function, $(L(\theta_k) - L(\theta_*)) \in \mathcal{O}(\gamma^{k^2})$ for some $\gamma < 1$ [BV04]. Compared to the $\mathcal{O}(\gamma^k)$ of gradient descent, this is a massive improvement. In fact, for convex problems, Newton's method converges typically in 10-50 iterations, while gradient descent may take several thousands [BV04].

Sadly, Newton's method never achieved widespread adoption in the deep learning community. There are two reasons for this. The first one is computational. If m is the dimension of θ, each iteration of gradient descent has cost $\mathcal{O}(m)$, Newton's iteration has cost $\mathcal{O}(m^3)$, due to the cost of inverting $\nabla_\theta^2 L(\theta_k)$. Since m tends to be extremely large in neural networks, this is a serious issue. However, several approaches can be used to deal with this challenge, and we will talk about them in sections 4.3 and 4.4. The second reason is that, even on problems where the computational costs can be overcome, it still fails to converge to a good solution. Why this happens and a first step to a solution, proposed by [DPG+14], is the topic of the next section.

4.2 Saddle-Free Newton

When we did Newton's update derivation, we did one critical assumption: that the loss was convex. Even more, the place we used that assumption was to say that the surrogate loss $\bar{L}$ was minimized at a $\bar{\theta}$ that satisfies $\nabla_\theta \bar{L}(\bar{\theta}) = 0$. This is clearly false if L is nonconvex, because then $\nabla_\theta^2 L(\theta_k)$ can have negative eigenvalues. Even more, when $\nabla_\theta \bar{L}(\bar{\theta}) = 0$, $\bar{\theta}$ has to be a **saddle point** of $\bar{L}$. Therefore, for nonconvex functions Newton's method approximates the loss by it's second order Taylor and drives θ_k to it's saddle point. Since our intuition tells us that the loss function of neural networks may suffer from saddle point proliferation, it might be that Newton's method is driving us to a close saddle point, and not a minimum. This claim was evidenced empirically on many real life problems by [DPG+14].

The main problem of Newton's algorithm when the functions nonconvex is that on the direction where the Hessian has negative eigenvalues, it changes the sign of the update of gradient descent. However, gradient descent always follows a descent direction, so changing the sign of the update doesn't really makes sense. To cope with this, [DPG+14] propose an alternative method, deemed *saddle-free Newton.* To introduce this method, we first define a few terms.

Definition 4.2.1. Let $\mathbf{D} = \begin{pmatrix} \lambda_1 & & \\ & \cdots & \\ & & \lambda_m \end{pmatrix} \in \mathbb{R}^{m \times m}$ be a diagonal matrix, we define it's absolute value $|\mathbf{D}| \in \mathbb{R}^{m \times m}$ by

$$|\mathbf{D}| = \begin{pmatrix} |\lambda_1| & 0 & \dots & 0 \\ 0 & |\lambda_2| & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & |\lambda_m| \end{pmatrix}$$

This is, we change the elements on the diagonal to their absolute value.

Definition 4.2.2. Let $\mathbf{H} \in \mathbb{R}^{m \times m}$ be a diagonalizable matrix with real eigenvalues (e.g. a real symmetric matrix). Let $\mathbf{H} = \mathbf{J}\mathbf{D}\mathbf{J}^{-1}$ be it's diagonalization. Then, we define $\mathbf{H}$'s absolute value to be $|\mathbf{H}| = \mathbf{J}|\mathbf{D}|\mathbf{J}^{-1}$. We therefore change the sign of the eigenvalues to be positive whenever needed.

A simple but important observation to be made is that if $\mathbf{H}$ is a real symmetric matrix, then $|\mathbf{H}|$ is positive semi-definite.

The saddle-free Newton method (popularly abbreviated as SFN) then proceeds as follows. At first, initialize θ_0 at random, and then iterate

$$\theta_{k+1} \leftarrow \theta_k - (|\nabla_\theta^2 L(\theta_k)|)^{-1} \nabla_\theta L(\theta_k)$$

The change in the negative eigenvalues is indeed critical. Taking the one dimensional case as an example, $-L'$ will take you always wherever the function is decreasing. Dividing by L'' when $L'' < 0$ makes no sense, since it will take you in a direction that increases L. However, if L'' is positive, we don't want to change the sign to keep the route the gradient follows. Therefore, dividing by $|L''|$ achieves a rescaling, but doesn't change the fact that this is still a descent direction.

One thing we need to notice is that if L is convex, then SFN and Newton's method are identical, because the Hessian is positive semi-definite so taking the absolute value doesn't change anything. Furthermore, if we are sufficiently close to a strict local minimum, and L is twice continuously differentiable (the assumptions made by all classical theorems on Newton's convergence [BV04]), then we get the same convergence results as Newton. This is because the Hessian at the minimum is positive definite, and since it's eigenvalues are a continuous function of θ [Tao12], they all have to keep being positive on a neighbourhood of the minimum.

If we use a small variant of the algorithm where we use $|\nabla_\theta^2 L| + \lambda \mathbf{I}_m$ instead of just $|\nabla_\theta^2 L|$ for some $\lambda > 0$ we can also retrieve the same guarantees as gradient descent for convex and nonconvex functions [BCN16]. This addition is popularly known as damping and is widely used in practice, since it limits how close to zero the division by the matrix eigenvalues can be. It is important to recall that the algorithm is derived by a second order approximation, and that therefore we should take some care not to assign an incredibly large step on any iteration. This is what damping achieves.

On some extremely hard real word neural network applications, [DPG$^+$14] shows the superior performance of SFN

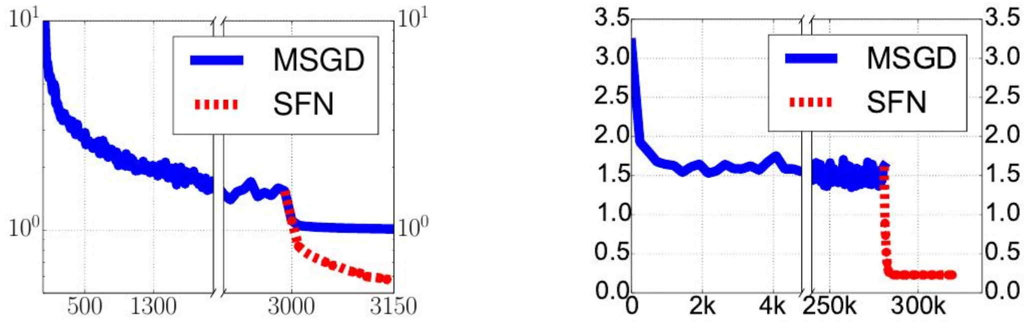

Figure 4.1: Learning curves for the training of a deep autoencoder (left) and a recurrent neural network (right), two hard optimiation problems. The x-axis shows the number of iterations, and the y-axis shows the cost function. Blue is SGD and red is SFN. As we see, even when SGD has converged, SFN decreases the cost massively in very few iterations.

The cost per iteration of saddle-free Newton is $\mathcal{O}(m^3)$, the complexity of diagonalizing the Hessian matrix. Since it's the same cost as Newton's method, in principle it doesn't seem to carry any computational disadvantage. However, as we will see in the next section, the approximation schemes that rend Newton's method viable in practice, do not transfer in a straightforward way to this algorithm.

4.3 Hessian-Free Optimization

One of the main disadvantages of Newton's algorithm and SFN is their computational complexity. If m is the number of parameters (or the dimension of θ), each ieration has cost $\mathcal{O}(m^3)$. Neural networks typically have at least 10^5 parameters for small networks and about 10^7 for medium sized ones. This means each iteration would take roughly 4 days on a single CPU for a small net[1], and about ten thousand years on a medium network. Running this on a GPU would make it about 15 times faster, but going from

[1]Networks that have about 10^5 parameters are mainly recurrent neural networks, which would have cost $\mathcal{O}(m^3T)$ per Newton iteration, with T the length of the sequence fed into the network, typically on the order of 200. This means that instead of 4 days per iteration, each one would take more than a year.

ten thousand years to a couple of centuries isn't really a big advantage. Therefore, using either Newton's method or SFN without any approximation scheme on neural networks is impossible.

One of the first approaches to using full second order methods in neural networks is Hessian-free optimization (HF) [Mar10]. HF is a combination of three main elements, with some modifications that we will mention later. The core elements of HF are:

- The conjugate gradient method [HS52] is an algorithm for solving linear systems of the form $\mathbf{A}x = b$ when $\mathbf{A} \in \mathbb{R}^{m \times m}$ is positive definite. Recall that in the end, all that's required for computing a Newton step is solving the system $\nabla^2_\theta L(\theta_k)x = \nabla_\theta L(\theta_k)$. The main advantage of this method is that it doesn't require access to $\mathbf{A}$, it only requires a way to calculate $\mathbf{A}v$ for any vector $v \in \mathbb{R}^m$. In the case of neural networks, we will see that there's a fast way to calculate $\nabla^2_\theta L(\theta)v$, but storing or even calculating the entire Hessian is prohibitively expensive.

  Another advantage of the conjugate gradient method is that it's iterative. It takes m iterations to converge, where each iteration requires calculating one $\mathbf{A}v$ product. In the typical case, calculating $\mathbf{A}v$ has cost $\mathcal{O}(m^2)$, so the overall cost remains the same. However, sometimes less than m iterations are required (as little as 20) to achieve a good approximation of x, and there are cases where calculating $\mathbf{A}v$ is surprisingly cheap.

- The Gauss-Newton method [Bjo96] is an algorithm similar to Newton, but with one important modification: replacing the Hessian with a positive semi-definite matrix $\mathbf{G}_\theta L(\theta)$ called the Gauss-Newton matrix. This matrix has the property that it converges to the Hessian when close to a minimum, and as is positive semi-definite it keeps the descent direction. However, the behaviour when the loss is nonconvex is not well understood. Furthermore, [MD08] argues against using the Gauss-Newton matrix on neural networks, showing it suffers from poor conditioning and drops the negative curvature information, which is argued to be crucial. Note that this is a major difference with SFN, which leverages the negative curvature information, keeping the scaling in these directions.

- Let m be the number of parameters, and n the number of examples used in computing $\hat{L}$ for one iteration of Newton or Gauss-Newton with loss $\hat{L}$. The $\mathcal{R}$-operator [Pea94] [Sch02] is a method that allows for computing $\nabla^2_\theta \hat{L}(\theta)v$ and $\mathbf{G}_\theta \hat{L}(\theta)v$ in $\mathcal{O}(mn)$ (same as the cost of gradient descent) time for any vector $v \in \mathbb{R}^m$ for the particular case of neural networks. Note that this is incredibly fast, since a typical matrix-vector product takes $\mathcal{O}(m^2)$. Furthermore, if we run l iterations of conjugate gradients to calculate the Gauss-Newton step (i.e. solving the system $\mathbf{G}_\theta \hat{L}(\theta_k)x = \nabla \hat{L}(\theta_k)$) has cost $\mathcal{O}(lmn)$, which is remarkably fast for low k and n.

Another component of HF, which is albeit minor in comparison to the previous three, is damping. Instead of using $\mathbf{G}_\theta \hat{L}(\theta)$, HF uses $\mathbf{G}_\theta \hat{L}(\theta) + \lambda \mathbf{I}_m$ for some $\lambda > 0$. This makes the system solved positive definite and reduces it's condition number, speeding up conjugate gradients and guaranteeing its convergence.

The overall HF optimization method looks as follows

```
$g_k \leftarrow \nabla_\theta \hat{L}(\theta_k)$
Pick a $\lambda > 0$ with some heuristic
Define the function: $B_k(v) = \mathbf{G}_\theta \hat{L}(\theta_k)v + \lambda v$
$\Delta\theta_k \leftarrow$ CG-Minimize($B_k, -g_k$)
$\theta_{k+1} \leftarrow \theta_k + \Delta\theta_k$
```

In the algorithm, CG-Minimize($\mathbf{A}, b$) means to run l iterations of the conjugate gradient method on the system $\mathbf{A}x = b$. We can see that the cost of calculating $\Delta\theta_k$ has therefore cost $\mathcal{O}(lmn)$. Since l is typically ranging from 20 to 200 and n is typically less than 1000, this is not prohibitively large. Note that even if each HF iteration is several orders of magnitude slower than an SGD iteration, the procedure converges extremely fast, and is in general stopped after less than 250 iterations, in comparison with the hundreds of thousands or millions in SGD.

For completeness, we mention how the damping parameter λ is chosen in HF. This is done is via a Levenberg-Marquardt[Lev44, Mar63] style heuristic: first, we initialize $\lambda = \lambda_0$. After that, if q_θ is the second order (or Gauss Newton) approximation to $\hat{L}(\theta)$ we define

$$\rho = \frac{\hat{L}(\theta + \Delta\theta) - \hat{L}(\theta)}{q_\theta(\theta + \Delta\theta) - q_\theta(\theta + \Delta\theta)}$$

and at each iteration if $\rho < \frac{1}{4}$ we do $\lambda \leftarrow \frac{3}{2}\lambda$, if $\rho > \frac{3}{4}$ we do $\lambda \leftarrow \frac{2}{3}\lambda$. The main idea is that ρ measures similarity between the actual loss and our second order approximation. If the approximation is reliable, we can afford a larger stepsize on low curvature directions and therefore we can reduce the damping parameter.

In conclusion, Hessian-free optimization allows a for way to approximate full second order methods such as Gauss-Newton for neural networks. Its main benefit is that it goes from the prohibitive $\mathcal{O}(m^3 + m^2 n)$ cost of full Newton to an approximation scheme that takes $\mathcal{O}(lmn)$ time, for low l. A nice guarantee is also that if $l = m$, we get back exactly the Gauss- Newton method. This is due to the fact that conjugate gradients solve the system exactly with m iterations.

An extremely important thing to notice is that HF uses a small constant amount of memory, $\mathcal{O}(m)$, irregardless of how good the approximation is. The only limitant factor is time. This is in contrast to low rank approximation algorithms, for example. These techniques require $\mathcal{O}(mk)$ memory for a k-rank approximation. Since m is very big in practice, low rank methods tend to be extremely limited.

4.4 Saddle-Free Hessian-Free Optimization

Unlike the Gauss-Newton method, SFN doesn't drop the negative curvature information, which seems to be extremely important in the case of nonconvex objectives, and especially in neural networks [MD08]. However, the $\mathcal{O}(m^3 + m^2 n)$ complexity makes it intractable in practice. One could wonder why do we not simply do a Hessian-free optimization style algorithm with $|\nabla^2_\theta \hat{L}|$ instead of $\mathbf{G}_\theta \hat{L}$. The main problem is that it's not clear how to calculate $|\nabla^2_\theta \hat{L}| v$ for any vector $v \in \mathbb{R}^m$ in $\mathcal{O}(mn)$ time like with the Gauss-Newton matrix.

The main disadvantage of calculating $|\nabla^2_\theta \hat{L}| v$ is that in order to replace λ_i with $|\lambda_i|$ we need to change the sign of only the negative λ_i's, for which we (in principle) would need to calculate them and check their signs. However, there's one simple way to go from x to $|x|$ without knowing the sign of x. We can just operate $|x| = \sqrt{x^2}$.

As we know, if $\mathbf{H} = \mathbf{J}\mathbf{D}\mathbf{J}^{-1}$, then $\mathbf{H}^2 = \mathbf{J}\mathbf{D}^2\mathbf{J}^{-1}$, and $\mathbf{D}^2$ is simply the result of squaring it's diagonal. Furthermore, we can define the square root of a positive definite matrix $\mathbf{H}^{\frac{1}{2}}$ by $\mathbf{H}^{\frac{1}{2}} = \mathbf{J}\mathbf{D}^{\frac{1}{2}}\mathbf{J}^{-1}$, where $\mathbf{D}^{\frac{1}{2}}$ is the result of taking the square root of it's entries. One can alternatively define $\mathbf{H}^{\frac{1}{2}}$ as the only positive definite matrix $\mathbf{A}$ such that $\mathbf{A}^2 = \mathbf{H}$. It is trivial to verify now that $|\mathbf{H}| = (\mathbf{H}^2)^{\frac{1}{2}}$.

Let us remember that we can use the $\mathcal{R}$-operator to calculate $\nabla^2_\theta \hat{L} v$ very quickly. Therefore, we can calculate $(\nabla^2_\theta \hat{L})^2 v$ quickly by applying the $\mathcal{R}$- technique twice. Let $\mathbf{A}$ be a positive definite matrix. If we can figure out how to calculate $\mathbf{A}^{\frac{1}{2}} u$ based only on knowing how to calculate $\mathbf{A}v$ for any vector v then we would be done, because setting $\mathbf{A} = (\nabla^2_\theta \hat{L})^2$ would give us a way to calculate $|\nabla^2_\theta \hat{L}| u$.

[ABB00] describes a method for calculating $\mathbf{A}^{\frac{1}{2}} v$ when $\mathbf{A}$ is tridiagonal that we describe now. Let us consider the following initial value problem:

$$\begin{cases} x'(t) = -\frac{1}{2} \left(t\mathbf{A} + (1-t)\mathbf{I}\right)^{-1} \left(\mathbf{I} - \mathbf{A}\right) x(t) \\ x(0) = v \end{cases} \tag{4.2}$$

Under the condition that $\|\mathbf{A}\|$ is sufficiently small, equation (4.2) has a unique solution [ABB00], which is

$$x(t) = \left(t\mathbf{A} + (1-t)\mathbf{I}\right)^{\frac{1}{2}} v$$

Most importantly, $x(0) = v$ and $x(1) = \mathbf{A}^{\frac{1}{2}} v$. While the original paper uses the fact that $\mathbf{A}$ is tridiagonal to solve the systems inside the calculations of $x'(t)$ fast enough, we can also solve them by applying the conjugate gradient methods. Since we can calculate products of the form $\mathbf{A}u$, we can also calculate $\left(t\mathbf{A} + (1-t)\mathbf{I}\right) u$ easily as well. By plugging this initial value problem into an ODE solver (such as RKF45), we can therefore recover x and obtain $x(1) = \mathbf{A}^{\frac{1}{2}} v$ as we wanted.

Going back to saddle-free Newton, we can calculate $\Delta\theta_k$ in two steps:

$$\begin{aligned} y &\leftarrow -\alpha \left((\nabla^2_\theta \hat{L}(\theta_k))^2 \right)^{\frac{1}{2}} \nabla \hat{L}(\theta_k) = -\alpha |\nabla^2_\theta \hat{L}(\theta_k)| \nabla \hat{L}(\theta_k) \\ \Delta\theta_k &\leftarrow \left(\left(\nabla^2_\theta \hat{L}(\theta_k) \right)^2 \right)^{-1} y \end{aligned}$$

The Hessian-free version we devised to follow this two steps, that we call *Saddle-free Hessian-free Optimization* (SFHF) [Arj15], follows naturally:

$$\begin{aligned} y &\leftarrow \text{ODE-solve} \Big(\text{Equation (4.2)}, \, \mathbf{A} = \nabla^2_\theta \hat{L}(\theta_k), \, v = -\alpha \nabla \hat{L}(\theta_k) \Big) \\ \Delta\theta_k &\leftarrow \text{CG-Minimize} \left(\left(\nabla^2_\theta \hat{L}(\theta_k) \right)^2, y \right) \end{aligned}$$

The complexity of SFHF is $\mathcal{O}(mnll')$, where l is the number of conjugate gradient iterations we use to solve the linear systems involved, and l' the number of evaluations $x'(t)$ evaluations we do to solve the differential equation. l' typically is about 40, but l is considerably smaller than in Hessian-free optimization. This makes the overall algorithm only a bit slower than Hessian-free. However, the fact that HF ignores the negative curvature makes it possible that SFHF converges in fewer iterations (and to a lower error), leading to a faster algorithm overall. While preliminary experiments support this hypothesis, more research needs to be done to arrive to a full conclusion.

Chapter 5

Conclusion

The optimization problem of learning deep neural networks is highly nonconvex. In spite of this, looking deeper into the structure of the loss, the theory of spin glasses points us to three likely and interesting conclusions:

- There are indeed an exponential number of local minima.
- However, all local minima lie within a small range of error with overwhelming proability. Almost all local minima will have similar error to the global minimum.
- There are exponentially more saddle points than minima, a phenomenon called saddle point proliferation.

We conjecture that it's likely the proof methods used for spin glasses can be extended to linear networks, and show a path of research to do so. However, due to the non-Gaussian behaviour created by the nonlinearities in multilayer perceptrons, it is not clear how to transition to full neural networks. The hierarchical structure of the hypothesis of neural networks might yield an understanding of the involved distributions, but much work needs to be done.

While the three consequences are overall great news, since all we need is to find a local minimum, this is horrible news for Newton's method, which jumps straight to a close saddle point. Leveraging this knowledge and the previous work on approximated second order methods, we create a new algorithm called saddle-free Hessian-free optimization, that's especailly designed to tackle the computational issues of using the full Hessian, and avoide saddle points.

Bibliography

[AAC13] Antonio Auffinger, Gérard Ben Arous, and Jiří Cerný. Random matrices and complexity of spin glasses. *Communications in Pure and Applied Mathematics*, 66:165–201, 2013.

[ABB00] Edward J. Allen, James Baglama, and Steven K. Boyd. Numerical approximation of the product of the square root of a matrix with a vector. *Linear Algebra and its Applications*, 2000.

[AHW96] Peter Auer, Mark Herbster, and Manfred K. Warmuth. Exponentially many local minima for single neurons. In *Advances in Neural Information Processing Systems 8 (NIPS 1995)*, pages 316–322. MIT Press, 1996.

[Arj15] Martin Arjovsky. Saddle-free hessian-free optimization for deep learning. *arXiv preprint arXiv:1506.00059*, 2015.

[AT07] Robert J. Adler and Jonathan E. Taylor. *Random fields and geometry.* Springer monographs in mathematics. Springer, New York, 2007.

[BCN16] Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *arXiv preprint arXiv:1606.04838*, 2016.

[Bjo96] Ake Bjorck. *Numerical Methods for Least Squares Problems.* Society for Industrial and Applied Mathematics, 1996.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization.* Cambridge University Press, New York, NY, USA, 2004.

[CHM$^+$15] Anna Choromanska, Mikael Henaff, Michael Mathieu, Gérard Ben Arous, and Yann LeCun. The loss surfaces of multilayer networks. *Journal of Machine Learning Research*, 38:192–204, 2015.

[CJLV16] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell: A neural network for large vocabulary conversational speech recognition. In *ICASSP*, 2016.

[DCM$^+$12] Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25*, 2012.

[DPG$^+$14] Yann N. Dauphin, Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, Surya Ganguli, and Yoshua Bengio. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27*, pages 2933–2941. Curran Associates, Inc., 2014.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2016.

[Hor91] Kurt Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[HS52] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436, 1952.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[JVS$^+$16] Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[Lev44] Kenneth Levenberg. A method for the solution of certain non-linear problems in least squares. *The Quarterly of Applied Mathematics*, (2):164–168, 1944.

[LSJR16] Jason D. Lee, Max Simchowitz, Michael I. Jordan, and Benjamin Recht. Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*, 2016.

[Mar63] Donald W. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11(2):431–441, 1963.

[Mar10] James Martens. Deep learning via hessian-free optimization. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10), June 21-24, 2010, Haifa, Israel*, pages 735–742, 2010.

[MD08] Eiji Mizutani and Stuart E. Dreyfus. 2008 special issue: Second-order stagewise backpropagation for hessian-matrix analyses and investigation of negative curvature. *Neural Networks*, 21(2-3):193–203, March 2008.

[MKS$^+$13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

[MRT12] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[Pea94] Barak A. Pearlmutter. Fast exact multiplication by the hessian. *Neural Computation*, 6:147–160, 1994.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[RM51] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[RS85] Herbert Robbins and David Siegmund. *A Convergence Theorem for Non Negative Almost Supermartingales and Some Applications*, pages 111–135. Springer New York, New York, NY, 1985.

[Sch02] Nicol N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14(7):1723–1738, 2002.

[SHM$^+$16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan 2016.

[SMG13] Andrew M. Saxe, James L. McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

[SVL14] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[Tao12] Terence Tao. *Topics in Random Matrix Theory*. Graduate studies in mathematics. American Mathematical Soc., 2012.

[TDH$^+$16] Eric Tzeng, Coline Devin, Judy Hoffman, Chelsea Finn, Xingchao Peng, Sergey Levine, Kate Saenko, and Trevor Darrell. Towards adapting deep visuomotor representations from simulated to real environments. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.