Classe di Scienze

Corso di perfezionamento in Data Science

XXXV ciclo

An Optimization Perspective on Deep Neural Networks

Settore Scientifico Disciplinare **INF/01**

Candidato
Dr. Dario Balboni

Relatore
Prof. Davide Bacciu

Anno accademico 2024/2025

An Optimization Perspective on Deep Neural Networks

Dario Balboni

Abstract

Despite the impressive performance achieved by Deep Neural Networks in recent years and their widespread adoption by major companies worldwide, many aspects of their behavior remain poorly understood.

A significant body of theory exists for very wide and infinite models, particularly related to *Neural Tangent Kernels*, and numerous empirical studies aim to guide practitioners working with practically relevant models. However, there is a concerning lack of actionable theoretical results for the types of models commonly deployed in practice.

This thesis positions itself in the gap between pure theory and practice, aiming to derive practical guidelines for practitioners based on a principled approach to neural networks grounded in optimization theory. This approach leverages the recently rediscovered Polyak–Łojasiewicz condition, a generalization of strong convexity suited to describing overparameterized models, and recognizes that certain classical results in convex optimization theory remain applicable to this new setting.

The end result of this thesis is an adaptive learning rate algorithm that requires minimal hyperparameter tuning, performing on par with grid-searched SGD while significantly reducing computational cost.

Contents

Introduction

Since its inception in the 1950s, the general aim of the field of Artificial Intelligence has been the creation of systems that behave "intelligently" from a human point-of-view [Gra+18; Sil+16; Ber+19; MWHN18; Lit+17; Van+10] and are thus capable of performing complex tasks and "thinking".

Recent developments in Deep Supervised Learning are particularly interesting from this standpoint, as they have shown that machines are capable of performing certain classes of tasks at human-level performance or higher (e.g., text generation, voice transcription, semantic image segmentation) [Abi+18], with major consequences for society.

Despite these advancements, many argue that significant differences exist between the ways humans learn and behave and the supervised learning setting [Zad19; Tav+18], and various hypotheses have been proposed regarding the optimal approach to creating "general intelligence" machines.

Proponents of the "Reward is Enough" hypothesis [SSPS21] argue that Reinforcement Learning is the essential optimization framework we should solve to reach human-level intelligence. Supporting evidence includes similarities in behavior between RNNs trained with RL and known human neurological patterns, such as the spontaneous emergence of meta-learning RL algorithms on related tasks [Bot+20][1].

An alternative hypothesis is the "scaling hypothesis," which posits that once we find a scalable architecture, such as self-attention, we can simply train ever larger networks with more varied data, and more sophisticated behavior will emerge naturally as the easiest way to optimize for all the task data [Kap+20], i.e., more powerful neural networks are essentially scaled-up versions of weaker neural networks, in much the same way that human brains look like scaled-up primate brains [Her12].

It is notable that these two hypotheses, along with standard supervised learning, share a common requirement: they all need sufficiently general function approximation models and efficient training algorithms. However, they differ in the specific type of optimization problem that must be solved: single-level optimization for supervised learning and the scaling hypothesis, and bi-level optimization for reinforcement learning [Liu+21].

Setting philosophical disputes aside, we believe that the chosen function approximation model and optimization algorithm warrant more systematic study. In this thesis, we focus specifically

[1] However, it is important to note that the emergence of meta-learning algorithms can be seen as a consequence of Hutter Search [Hut02], which can be paraphrased as "one of the shortest algorithms to compute something is also one of the fastest." Thus, it becomes "easier" to find a model that generalizes rather than memorizing each example explicitly when many are given.

on the supervised optimization problem for deep network models. In the rest of the Introduction, we will discuss the importance of this problem, the main challenges involved, and our proposed approach.

Deep Networks as Practical Function Approximators. A wide variety of model classes has been used in Machine Learning as function approximators, from simple models like Random Forests [Bre01], Linear Models [HK70], and Distance-based Nearest Neighbours [CH67] to more complex ones such as Support Vector Machines [Cor95], Gaussian Processes [WR06], Kernel Methods [SSM98], and Deep Neural Networks [RHW86]. Why should we focus on understanding Deep Networks?

First, we should note that Deep Networks are particularly effective for certain types of real-world data, such as text, images, and audio recordings: it is important to remember that simpler models often perform better for tabular data, and for certain tasks, numerical simulations using differential equations are more appropriate[2].

Second, they intuitively model correlations within data (e.g., through weight sharing in convolutional kernels), which are challenging to express as a kernel in the context of Gaussian Processes or Kernel Methods and simultaneously ensure more efficient data utilization [CBR19] since multiple patches provide more data to train the same convolutional elements[3].

Third, they can be trained incrementally and use a fixed amount of memory irrespective of the data size, are easily parallelizable, and can be stopped at any time while still yielding a useful model as output[4].

Deep Networks are currently state-of-the-art models for various tasks as mentioned, but their theoretical understanding is still being developed. Particularly concerning is the lack of generalization guarantees on the found models [Zha+16; Zha+21], which are needed in sensitive deployments (and which both Kernel Methods and Gaussian Processes have), and the many optimization-related confounders affecting the validity of proposed methods: the results of an approach are fundamentally biased by computational resources and the optimization algorithm used, all else being equal, yet this is often overlooked in research proposing new methods or architectures.

A better understanding of Deep Networks in their optimization setting can lead to guarantees on the reached minima and better comparisons between proposed methods and model improvements. This can also have wide-reaching implications for the speed of model optimization, impacting many practical deployments, as today's AI capabilities are severely limited by the massive energy, hardware, and computation required.

Theoretical Problems of Neural Networks. A major challenge in the theoretical analysis of Neural Networks is that they are inherently non-convex [ITB16], while classical optimization theory deals mostly with convex functions [Nes+18]. Thus, a new mathematical framework is needed to study this type of optimization problem.

[2]However, some may argue that this limitation could be temporary, and that Deep Networks will eventually outperform these techniques.

[3]Of course, nothing theoretically excludes this possibility for Kernel Methods or Gaussian Processes.

[4]In this respect, other models like Kernel Methods are catching up with incremental techniques [RCR15] and are now capable of training over billions of points [MCRR20].

Another issue arises from initialization procedures (which rely on randomness) and the path dependence exhibited by optimization algorithms like Stochastic Gradient Descent on overparameterized models, which can reach different minima depending on the starting point.

Furthermore, the large number of hyperparameters required in a network model (activation function, number of neurons per layer, number of layers, distribution of the weight initialization, etc.) complicates a clear mathematical treatment and makes it more challenging to compare approaches across variations of these settings.

These challenges make a formal mathematical approach unlikely in the near future, and our treatment of the matter will thus rely on mathematical intuition validated by experiments, rather than full mathematical rigor, as suggested by Breiman [Bre95]. We will thus work with locally-optimal algorithms and tuning procedures derived from infinite-width limits that demonstrate superior performance on practical examples of networks similar to those used in practice.

Algorithms and Architectures as the Future Scaling Bottleneck. The scaling hypothesis has remained relevant in recent years since the introduction of transformers in 2017 [Vas+17], despite major critiques from prominent researchers. However, we believe that the scaling hypothesis will eventually reach its limits

, requiring a deeper examination of the architectural biases we introduce in networks and the types of optimization performed to ensure efficient use of data and to keep training costs low, especially if we aim to democratize such transformative technology as AI.

Another compelling reason to develop more advanced training algorithms is that they enable smaller players with limited hardware resources to achieve competitive performance, supporting the development of AI as a common good.

Impact on Society and the Role of Academia. In this light, the field of artificial intelligence has the potential to transform the structure of work in our society within a decade, following dynamics similar to those of the internet in the 1990s and early 2000s, but possibly with more profound and faster changes.

This concentration of ownership is problematic for the average person, as these new AI tools, which enhance productivity for some and potentially displace others, are controlled by a few large corporations, and all freely released models lag significantly behind the state-of-the-art.

Academia has largely refrained from addressing this imbalance and is instead relinquishing its role in driving the development of AI as a public good, focusing on research that requires minimal investment, which, while valuable, primarily benefits private entities equipped with the hardware infrastructure necessary to maintain their dominance.

One way to counter this trend is for the public sector to invest in larger infrastructure and to prioritize research aimed at reducing hardware, data, and computational demands to train foundational models, so that these models can be freely released for everyone to build upon, rather than allowing large corporations to dictate the uses of this new technology and extract oligopolistic profits.

Structure of the Thesis. The first part of the thesis presents the necessary background for this research, while the second part focuses on novel contributions and potential future directions.

In particular, in Chapter 1 we present the branch of Mean Field Theory, which uses the Central Limit Theorem to study the behavior of signal propagation inside neural networks at initialization, and which provides a basis for addressing randomness in initialization procedures; in Chapter 2 we cover the basic theory of Kernel Methods and examine the relations between Very Wide Deep Neural Networks and Kernel Methods via the Neural Tangent Kernel derivation, which can be seen as an alternative way to handle randomness through limit theorems; in Chapter 3 we present the relationship between Deep Neural Networks and optimization theory; in Chapter 4 we introduce the theory of Polyak–Łojasiewicz functions, a generalization of convexity that underpins our preliminary exploration of alternative mathematical frameworks for interpreting neural network behavior.

Later, we discuss the author's contributions: in Chapter 5 we explore a connection between Deep Neural Networks and Deep Kernel Methods; in Chapter 6 we test the theory of Polyak–Łojasiewicz functions on real-world models to validate its predictions on convergence, conditioning, and generalization of neural models [BB22]; in Chapter 7 we provide the theoretical base for the soundness of an automatic learning rate tuning algorithm, describe how to appropriately tune the learning rate in Polyak–Łojasiewicz functions and we derive an efficient adaptive learning procedure to train DNNs from their local quadratic structure [BB24]; finally, in Chapter 8 we highlight possible future directions for further development of the exposed theory.

Part I

Prior Art

Overview. In this first part, we present three approaches to analyzing Deep Neural Networks: in Chapter 1 using Mean Field Theory (MFT), in Chapter 2 we examine the networks infinite-width limit corresponding to the Neural Tangent Kernel (NTK), and in Chapters 3 and 4 we explore their optimization behavior via the Polyak–Łojasiewicz (PL) condition. These methods provide the foundation needed to understand the original contributions in the second part of the thesis.

Each of these approaches is mathematically rigorous but corresponds to the study of a simplified version of the network. In MFT, one studies the network by computing expectations and variances of quantities over random initializations, thereby taking an ensemble view of the various networks that can arise from a fixed architecture with random initialization. In the NTK approach, the network is analyzed in the limit where the number of neurons per layer goes to infinity, reducing the problem to the study of specific Kernel Methods that depend only on the network architecture. The PL condition, on the other hand, allows for studying individual networks independently, as well as tracking them throughout the optimization trajectory, but only within a local neighborhood of the current parameters.

Aim. The general aim of our inquiry is to derive theory-backed insights that can help us understand Deep Neural Networks in terms of guarantees on learning convergence and the generalization of the reached minima. In the second part, we conduct a thorough investigation into the predictions that can be made on realistic networks using the PL condition.

Our more practical goal is to derive a parameter-free optimization algorithm. As mentioned in the introduction, the validity of a proposed method or model is often obscured by optimization-related confounding factors when applied to specific tasks, making it extremely difficult to assess whether one model truly outperforms another. A theoretically-backed, efficient, and hyperparameter-free optimization algorithm would provide tremendous advantages in this regard and is, in our opinion, currently achievable. As a step in this direction, we introduce in the second part a learning rate adaptation rule derived from measurable network parameters, resulting in a learning-rate-free optimization algorithm based on SGD.

Scope. We limit the scope of our research to the standard supervised learning problem with a well-defined target. However, we note that the techniques we use are typically extendable to more complex optimization settings, albeit with an increase in theoretical complexity.

Within this scope, we are interested in predictions or guarantees on the convergence of the learning process and the degree of generalization displayed by the resulting minima, i.e., the expected accuracy when using the trained networks on new data drawn from the same distribution as the training data.

Chapter 1

Mean Field Theory

The approach of Mean Field Theory, a traditional physics technique used to tackle high-dimensional problems in the context of Statistical Mechanics, was initially applied to Fully Connected Networks by Schoenholz, Gilmer, Ganguli, and Sohl-Dickstein [SGGS17], to Convolutional Networks by Xiao et al. [Xia+18], and later to Recurrent Networks by Chen, Pennington, and Schoenholz [CPS18]. The effectiveness of this theory is highlighted by the fact that Xiao et al. [Xia+18] successfully trained a CNN with ten thousand layers without employing any special tricks.

This theory focuses on selecting initialization procedures for neural networks that enhance training efficiency by improving the propagation of signals during both the forward and backward passes in deep networks of various architectures. To facilitate better signal propagation within the network, the approach identifies an optimal set of initialization hyperparameters (mean and variance of the network weights distribution) that represents a fixed point for the activation variances as they progress through the layers. This ensures that both the forward and backward variances neither explode nor vanish.

Consequently, this approach can be regarded as a general solution to the exploding or vanishing gradient problem [BSF94]. It can be applied to analyze different network architectures with minimal effort, thereby providing a foundation for initializing new architectures. The method simplifies the network problem by examining it in terms of ensemble behavior across all possible random initializations, allowing its conclusions to assist in identifying network ensembles that perform better on average. However, it is important to note that there may still be atypical instances that behave unexpectedly, although these cases become increasingly rare as the width of each layer increases, due to concentration of measure arguments.

The field has been rigorously formalized through a series of papers by Yang [Yan19a; Yan19b; Yan20a; Yan20c; YL21; YH21; Yan+22; GY22], which established a framework of tensor programs. This framework also elucidated the conditions under which the so-called Gradient Independence Assumption holds. This assumption posits that the weights in the forward pass are independent of the weights in the backward pass, a simplification that has been utilized in multiple papers to ease calculations. Although this assumption is generally false, it holds true in many significant architectural cases, and we will explore the underlying concepts in the concluding section of this chapter.

Finally, it is worth noting that the calculations employed in Mean Field Theory assume that

the weights are Gaussian distributed for the sake of simplicity. A perturbative theory for these calculations can be derived by expanding the distributions themselves using an Edgeworth series expansion at higher orders, as demonstrated by Roberts, Yaida, and Hanin [RYH22].

Overview of Mean Field Theory. Mean Field Theory (MFT) is a widely employed approximation method in physics, mathematics, and related disciplines, where systems comprising a large number of interacting components are studied. The central idea of MFT is to simplify the analysis of complex, many-body problems by replacing the detailed interactions between individual components with an average or "mean field." This approach transforms a problem with many degrees of freedom into a simpler, effectively single-body problem, while retaining the essential features of the system's collective behavior.

The theory was initially developed in the context of statistical physics to describe phase transitions and critical phenomena, such as in the Ising model of ferromagnetism. In the Ising model, for example, spins on a lattice interact with their nearest neighbors, creating a computationally challenging problem due to the exponential growth of configurations. MFT approximates this by assuming that each spin interacts with an average magnetization of the system, thereby reducing the complexity to solving self-consistent equations for the magnetization.

Mathematically, the mean field approximation involves expressing the interaction term in a system's Hamiltonian or governing equation as a function of average quantities. For instance, in the case of the Ising model, the Hamiltonian H is given by:

$$H = -J \sum_{\langle i,j \rangle} s_i s_j - h \sum_i s_i,$$

where J is the interaction strength, h is the external field, and s_i denotes the spin at site i. Under MFT, the spin interaction term $\langle s_i s_j \rangle$ is replaced by the product of averages $\langle s_i \rangle \langle s_j \rangle$, leading to a self-consistent equation for the average magnetization $m = \langle s_i \rangle$:

$$m = \tanh\left(\frac{Jzm + h}{k_B T}\right),$$

where z is the coordination number (number of nearest neighbors), k_B is Boltzmann's constant, and T is the temperature.

Beyond statistical physics, MFT has been successfully applied in diverse areas such as population dynamics, neural networks, and economics. In the context of neural networks, for instance, the Hopfield model employs MFT to approximate the state of neurons by assuming that the influence of all other neurons on a given neuron can be replaced by a mean synaptic input. Similarly, in population dynamics, the Lotka-Volterra equations can be analyzed using mean-field approaches by considering the average interaction rates between species.

While MFT often provides valuable insights and simplifies otherwise intractable problems, its accuracy depends on the system under study. It tends to work well in high-dimensional systems or when fluctuations are negligible, but it may fail near critical points or in systems with strong correlations where fluctuations play a significant role. Extensions and refinements, such as the inclusion of fluctuations via renormalization group theory, can address some of these limitations.

Mean Field Theory is thus a versatile and foundational tool that has contributed significantly

to our understanding of complex systems by providing an intuitive and computationally manageable framework for analyzing collective phenomena.

1.1 Mean Field Theory of Fully Connected Networks

We begin our exploration of Mean Field Theory in the context of Fully Connected Networks, as introduced by Poole et al. [Poo+16] and Schoenholz, Gilmer, Ganguli, and Sohl-Dickstein [SGGS17].

Mathematical Setup. Consider a fully connected feedforward neural network with L layers, where each layer has a width of N_l, and let $\phi : \mathbb{R} \to \mathbb{R}$ denote some nonlinearity. We assume that the weights and biases are sampled independently from Gaussian distributions:

$$W_{ij}^l \sim \mathcal{N}(0, \sigma_w^2/N_l), \quad b_i^l \sim \mathcal{N}(0, \sigma_b^2).$$

We define z_i^l as the pre-activations of the l-th layer and y_i^{l+1} as the activations of that layer, with the input example a to the network given by $y_i^0 = x_{i;a}$. The signal propagation through the network is described by:

$$z_i^l = \sum_j W_{ij}^l y_j^l + b_i^l, \quad y_i^{l+1} = \phi(z_i^l).$$

Mean Field Treatment. The function computed by the network at initialization is significantly influenced by the specific random sample of the weight and bias Gaussian matrices. The Mean Field approximation involves replacing z_i^l with a Gaussian distribution whose first two moments match those of z_i^l.

Signal Propagation and Covariance. Consider the evolution of two inputs, $x_{i;a}$ and $x_{i;b}$[1], as they progress through the network. Since the weights and biases are independent and have zero mean, the first two moments of the pre-activations in the same layer are:

$$\mathbb{E}\Big[z_{i;a}^l\Big] = 0, \quad \mathbb{E}\Big[z_{i;a}^l z_{j;b}^l\Big] = q_{ab}^l \delta_{ij},$$

where δ_{ij} is the Kronecker delta, and q_{ab}^l represents the covariance of the pre-activations in the l-th layer. A recursive relation for q_{ab}^l can be derived:

$$q_{ab}^l = \sigma_w^2 \int \phi(u_1)\phi(u_2)\mathcal{D}z_1\mathcal{D}z_2 + \sigma_b^2,$$

where $u_1 = \sqrt{q_{aa}^{l-1}} z_1$, $u_2 = \sqrt{q_{bb}^{l-1}}\left(c_{ab}^{l-1} z_1 + \sqrt{1 - \left(c_{ab}^{l-1}\right)^2} z_2\right)$, $\mathcal{D}z$ is the measure for a standard Gaussian distribution, and $c_{ab}^l = q_{ab}^l / \sqrt{q_{aa}^l q_{bb}^l}$ is the correlation between two inputs after l layers.

[1] Here a and b are arbitrary indices for the input examples, but i refers to their vector components.

It is evident that for any choice of σ_w^2, σ_b^2, and a bounded ϕ, the covariances for the same input converge to a fixed point given by $q^* = \lim_{l\to\infty} q_{aa}^l$.

Stability Condition of the Fixed Point. The value $c^* = 1$ serves as a fixed point of the recurrence relation, and the condition for its stability depends on

$$\chi_1 = \frac{\partial c_{ab}^l}{\partial c_{ab}^{l-1}} = \sigma_w^2 \int \phi'(\sqrt{q^*}z)^2 \mathrm{d}z,$$

indicating that it is stable if $\chi_1 < 1$ and unstable otherwise. Consequently, $\chi_1 = 1$ delineates a critical line that separates a phase where all inputs become asymptotically correlated from a phase where they become asymptotically decorrelated. Therefore, to ensure optimal signal propagation within the network, we should choose σ_w^2 and σ_b^2 such that $\chi_1 = 1$, allowing us to confidently assert that two samples will remain correlated as they progress through all layers of the network.

Gradient Propagation and Jacobian Matrix. The backpropagation of gradients for a given loss E is governed by the two equations:

$$\frac{\partial E}{\partial W_{ij}^l} = \delta_i^l \phi(z_j^{l-1}), \quad \delta_i^l = \phi'(z_i^l) \sum_j \delta_j^{l+1} W_{ji}^{l+1},$$

where $\delta_i^l = \frac{\partial E}{\partial z_i^l}$. The scale of fluctuations in the gradient of weights within a layer is proportional to $\mathbb{E}\left[\left(\delta_i^l\right)^2\right]$; however, the δ_i^l will not follow a Gaussian distribution even in the limit of large layer widths.

Nevertheless, we can derive a relation for the variance of the errors, $\tilde{q}_{aa}^l = \mathbb{E}\left[\left(\delta_i^l\right)^2\right]$, by leveraging the Gaussian distribution of the pre-activations. This derivation requires the assumption that the weights used during forward propagation are drawn independently from those used in backpropagation, a condition known as the Gradient Independence Assumption (GIA) [SGGS17], that we will treat more in depth in Section 1.3. From this, we can establish the equation

$$\tilde{q}_{aa}^l = \tilde{q}_{aa}^{l+1} \frac{N_{l+1}}{N_l} \chi_1.$$

Focusing on networks where all layers have the same width, i.e., $N_{l+1} = N_l$, this equation highlights the critical importance of maintaining χ_1 at exactly one to prevent the gradient from either exploding or vanishing.

1.2 Dynamical Isometry

We have seen how important it is to keep the mean value of the Jacobian matrix close to one, as this enables better-conditioned backpropagation of gradients, helps avoid the problems of gradient explosion or vanishing, and thus leads to faster training dynamics. This means that, on average, a randomly chosen vector will preserve its norm during backpropagation; however, it provides no guarantees regarding the worst-case shrinkage or growth of such a vector. A stronger

requirement to ensure this is to require that every singular value of the Jacobian remains close to one. Under this stronger condition, error vectors backpropagate faithfully by preserving their norm and the angles between any two of them, resulting in better overall conditioning of the network. The imposition of this additional condition is referred to as Dynamical Isometry.

Mathematical Setup. Using the same notation as in the preceding section, in an L-layer deep feed-forward neural network, the input-output Jacobian J is given by

$$J = \frac{\partial z^L}{\partial y_0} = \prod_{l=1}^{L} D^l W^l,$$

where D^l is a diagonal matrix with entries $D^l_{ij} = \phi'(z^l_i)\delta_{ij}$.

It is now clear that the input-output Jacobian J is closely related to the conditioning of the backpropagation operator that maps output errors to weight matrix updates at a given layer: if the input-output Jacobian is well-conditioned, then the backpropagation operator will also be well-conditioned. Thus, the desired condition is that the variance of the eigenvalues of JJ^T is minimized.

Empirical Results. For deep linear networks, Saxe, McClelland, and Ganguli [SMG13] have shown that orthogonal weight initialization achieves dynamical isometry, and their learning time in epochs becomes independent of depth, in stark contrast to what occurs when using Gaussian weight initialization.

Pennington, Schoenholz, and Ganguli [PSG17] extend the analysis to the case of non-linear dense networks, demonstrating both theoretically and empirically that controlling the shape of the entire distribution of a deep network's Jacobian singular values has a dramatic effect on learning speed, even for non-linear networks.

We encourage interested readers to consult the cited papers for an in-depth understanding of the techniques employed, while we shift our focus to address an assumption that has frequently been made in the Mean Field Theory literature, but which was only proven years later by Yang [Yan19a].

1.3 The Gradient Independence Assumption

The reader will recall from the Mean Field Theory section that we posited that the elements of the weight matrices and the biases in neural networks during both the forward and backward phases of the learning algorithm were derived from two independent samples of the same Gaussian distribution.

This assumption simplifies the mathematical analysis of all network architectures in Mean Field Theory, enabling researchers to apply classical probabilistic results such as the Central Limit Theorem and the Law of Large Numbers to derive meaningful insights into neural network behavior.

Despite its analytical convenience, the Gradient Independence Assumption has notable limi-

tations. In real-world neural networks, the weights in the forward and backward passes are certainly not independent. Moreover, factors such as weight sharing in CNNs, recurrent connections in RNNs, and the dependencies introduced during training violate the assumption of independence. As training progresses, weights become correlated due to the shared optimization process, and their distributions deviate from the initial Gaussian assumption.

Violating the Gradient Independence Assumption has significant implications for both theoretical and practical aspects of neural network analysis. Theoretical predictions based on the GIA may no longer hold, leading to potential inaccuracies in the expected behavior of signal propagation and gradient dynamics. This can affect the stability and efficiency of training algorithms, as the assumed conditions for convergence and performance are not met. Fortunately, the framework of Tensor Programs provides a solid mathematical foundation for the results obtained under the GIA and can offer precise conditions under which the GIA effectively holds.

Yang Tensor Programs. The Tensor Programs introduced by Yang [Yan19a] provide a unifying framework for analyzing and understanding neural networks of any architecture. Tensor Programs formalize the computations in neural networks using a series of mathematical operations represented as sequences of tensors. This framework allows for a rigorous and scalable analysis of the large width limit of neural networks, extending beyond the simple cases typically considered in Mean Field Theory and providing structured insights into their behavior that were previously difficult to obtain.

A Tensor Program $\mathcal{T}$ is formally defined as a sequence of lines involving operations between vectors and matrices, adhering to specific rules, along with dimension annotations, and can include the following types:

- **VecIn (G):** A vector input x
$$l : g^l := x \in \mathbb{R}^{n^l}$$
- **MatIn (A):** A matrix input A
$$l : A^l := A \in \mathbb{R}^{n_1^l \times n_2^l}$$
- **T (A):** Transpose of an A-var
$$l : A^l := (A^j)^\top \in \mathbb{R}^{n_1^l \times n_2^l} = \mathbb{R}^{n_2^j \times n_1^j}$$
- **MatMul (G):** If A^k and g^j have $n_2^k = n^j$, then an assignment via a linear mapping
$$l : g^l := A^k g^j \in \mathbb{R}^{n^l} = \mathbb{R}^{n_1^k}$$
or similarly for H-vars
$$l : g^l := A^k h^j \in \mathbb{R}^{n^l} = \mathbb{R}^{n_1^k}$$
where $j, k < l$.
- **LinComb (G):** If $n^{j_1} = \cdots = n^{j_k}$, then an assignment via a linear combination of G-vars that appeared in previous lines:

With $a^l_{j_i} \in \mathbb{R}$,

$$l : g^l := a^l_{j_1} g^{j_1} + \cdots + a^l_{j_k} g^{j_k} \in \mathbb{R}^{n^l} = \mathbb{R}^{n^{j_1}}$$

- **Nonlin (H):** If $n^{j_1} = \cdots = n^{j_k}$, then an assignment via some general (possibly nonlinear) function $f^l : \mathbb{R}^k \to \mathbb{R}$, acting coordinate-wise,

$$l : h^l := f^l(g^{j_1}, \ldots, g^{j_k}) \in \mathbb{R}^{n^l} = \mathbb{R}^{n^{j_1}}$$

Here, (G) denotes those variables referred to as G-vars, and similarly, we have A-vars and H-vars. Variables introduced by **VecIn** and **MatIn** are also called (vector and matrix) input vars. The initial l : indicates the line number, and each new variable formed from this line is labeled with a superscript l. A partial program with n^l and unspecified input G- and A-vars is called a (program) skeleton, typically denoted by Greek letters like π.

This sequence of operations is capable of representing the entire complexity of the possible computations within a neural network, from the initial input through successive layers of processing to the final output, both in the forward and backward passes in most common architectural models.

Gradient Independence Assumption under Tensor Programs. Yang [Yan19a] proves the following theorem: in a multilayer perceptron (MLP) with nonlinearities that have polynomially bounded weak derivatives, the GIA leads to the correct Jacobian equation and the accurate computation of the singular value distribution, provided that the readout layer is sampled independently from other parameters and has a mean of zero.

The proof is rather convoluted, and we advise the reader to consult the cited paper for details. However, the general intuition is that by properly formalizing neural networks as Tensor Programs, Yang is able to induct on the line number of the specifying program, taking the limit for the dimensions towards infinity, and thus appropriately applying the Central Limit Theorem in a structured manner for the various specific operations involved in the network program definition. Crucially, the convergence to Gaussian behavior for the backward pass depends on the zero-mean distribution of certain output weights, which thereby validates the GIA in typical cases.

1.4 Extensions of the Techniques

Applications to Other Architectures. The technique elucidated for achieving dynamical isometry in Fully Connected Networks is not limited to this specific architecture; rather, it can be extended to a variety of other architectures with minor adjustments. This has been demonstrated with Convolutional Neural Networks by Xiao et al. [Xia+18], Recurrent Neural Networks by Chen, Pennington, and Schoenholz [CPS18], and Transformers by Bachlechner, Majumder, Mao, and McAuley [BMMM] and He et al. [He+23], making it a powerful approach to address gradient-scaling issues during the training of deep networks.

Impact of the Training Algorithm. The Mean Field treatment of Deep Networks that we presented only considers the norm of the gradient to tackle a fundamental numerical issue dur-

ing training, namely vanishing or exploding gradients. While this approach is beneficial when optimizing with Stochastic Gradient Descent, it remains unclear whether it also aids other optimizers, such as Adam, RMSprop, or second-order methods. Each of these algorithms introduces unique dynamics into the training process, potentially interacting with the initialization scheme in unexpected ways. It is essential to extend the analysis to include these alternative optimization strategies, ensuring that the advantages of dynamical isometry are preserved regardless of the chosen optimizer, or to understand how the conditions should be adjusted for various optimizers.

Challenges in the Treatment of Heterogeneous Architectures. Furthermore, the techniques assume a certain homogeneity in the distribution of weights and activations, which may not hold uniformly across all layers or architectures. The practical implementation of neural networks often involves heterogeneous layers that vary in width, activation functions, and connectivity patterns. Addressing these variations necessitates a more nuanced approach to initialization, potentially involving layer-wise adjustments to the orthogonal initialization scheme. Such refinements could ensure that dynamical isometry is maintained even in the presence of architectural heterogeneity, thereby enhancing the robustness and generality of the initialization strategies.

Data-Dependent Initialization Methods. Moreover, the exploration of data-dependent initialization methods could be beneficial; in these methods, the initialization parameters are adaptively modified based on the input data distribution. This could involve pre-training phases or iterative adjustment mechanisms that fine-tune the initialization parameters to align more closely with the empirical characteristics of the data.

Chapter 2

Neural Tangent Kernels

The introduction of Neural Tangent Kernels by Jacot, Gabriel, and Hongler [JGH18] established a significant connection between very wide networks and Kernel Methods. This connection has provided researchers with greater insight into the behavior of large models and has led to the production of numerous intriguing theoretical papers on related topics, including the extension of NTKs to various architectures by Yang [Yan20a].

The core idea of this technique involves linearizing the networks concerning their weights at initialization and examining the behavior as the number of weights approaches infinity with a specific parameterization. By applying the Central Limit Theorem, the limit can be computed as an expectation over the random initialization distribution. It can be demonstrated that the resulting differential equation corresponds to Ridge Regression for a fixed kernel, which is derived from the tangent vectors of the network function at the current point, thus earning the name Neural Tangent Kernel.

In this chapter, we will first provide a brief introduction to Kernel Methods and their generalization bounds, and in the final section, we will explain the essence of the theory of Neural Tangent Kernels.

2.1 Introduction to Kernel Methods

In this section, we present a general introduction to Kernel Methods, highlighting the Representer Theorem and the Kernel Trick. For a more comprehensive exposition, we recommend the work by Micchelli and Pontil [MP05].

Kernel methods are a class of algorithms utilized for pattern analysis, which gained prominence with the development of Support Vector Machines (SVMs) by Vapnik [Vap95] in 1995. These methods have been widely applied across various domains, including image recognition, bioinformatics, and natural language processing, due to their capability to efficiently handle complex, nonlinear relationships in data.

These methods transform the input data into a higher-dimensional (sometimes infinite-dimensional) space using a function, enabling the resolution of complex problems that are not linearly separable in the original space but can be effectively linearly separated in the higher-dimensional space. The concept of kernel methods revolves around the use of kernel functions to implicitly

map data into this higher-dimensional space, making them particularly effective for tasks such as classification, regression, and clustering.

One notable advantage of kernel methods is their flexibility: they can be applied to various types of data, including vectors, sequences, and graphs. Furthermore, they can be integrated with different machine learning algorithms, such as SVMs, principal component analysis (PCA), and clustering algorithms. Kernel methods also offer a means to incorporate domain-specific knowledge through the selection of the kernel function, which can be tailored to capture the specific characteristics of the data.

We now proceed to explain the mathematical setting of Kernel Methods.

Reproducing Kernel Hilbert Spaces (RKHS). A Reproducing Kernel Hilbert Space (RKHS) is a Hilbert space of functions in which evaluation at each point can be represented as an inner product. Formally, a Hilbert space $\mathcal{H}$ of functions $f : X \to \mathbb{R}$ defined on a set X is an RKHS if there exists a function $K : X \times X \to \mathbb{R}$ such that for all $x \in X$ and $f \in \mathcal{H}$:

$$f(x) = \langle f, K(x, \cdot)\rangle_{\mathcal{H}},$$

where $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ denotes the inner product in the Hilbert space of functions. The function K is referred to as the reproducing kernel of $\mathcal{H}$, and it satisfies the reproducing property:

$$K(x, y) = \langle K(x, \cdot), K(y, \cdot)\rangle_{\mathcal{H}}.$$

Properties of Kernels. For a function K to qualify as a valid kernel, it must be symmetric and positive-definite, specifically:

$$K(x, y) = K(y, x)$$

and for any set of points $\{x_1, \ldots, x_n\} \subset X$ and for all $c_1, \ldots, c_n \in \mathbb{R}$:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0.$$

These properties ensure that the kernel function can be utilized to define a Hilbert space and a valid inner product within it, thereby enabling the application of kernel functions as a substitute for the more abstract formalism of RKHS. Furthermore, the smoothness or regularity conditions imposed on the function K imply corresponding smoothness or regularity for the functions that can be represented in the respective RKHS, allowing for precise control over the class of functions to consider in our data modeling.

Representer Theorem. The Representer Theorem is a fundamental result in the theory of kernel methods, asserting that the solution to a regularized Empirical Risk Minimization problem in an RKHS can be expressed as a finite linear combination of kernel functions evaluated at the training points.

Theorem Statement. Given n samples (x_i, y_i) drawn from an unknown distribution $\mathcal{D}$, we consider the regularized empirical risk minimization problem:

$$\min_{f \in \mathcal{H}} \sum_{i=1}^{n} L(x_i, y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

where L is a loss function that is convex in its third argument, λ is a regularization parameter, and $\mathcal{H}$ is an RKHS with reproducing kernel K. The solution f can then be expressed as:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x),$$

where the coefficients $\alpha_i \in \mathbb{R}$.

Proof of the Representer Theorem. Consider the optimization problem:

$$J(f) = \sum_{i=1}^{n} L(x_i, y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

Assume that the solution f can be decomposed into components within the span of the kernel functions evaluated at the training points x_i and a part orthogonal to this span:

$$f = f_{\parallel} + f_{\perp}$$

$$f_{\parallel} = \sum_{i=1}^{n} \alpha_i K(x_i, \cdot)$$

$$\|f\|_{\mathcal{H}}^2 = \|f_{\parallel}\|_{\mathcal{H}}^2 + \|f_{\perp}\|_{\mathcal{H}}^2.$$

Since $f_{\perp}$ is orthogonal to the span of $\{K(x_i, \cdot)\}_{i=1}^{n}$, it contributes nothing to the empirical risk term because $f_{\perp}(x_i) = 0$ for all i. Thus, we have:

$$J(f) = \sum_{i=1}^{n} L(x_i, y_i, f_{\parallel}(x_i)) + \lambda(\|f_{\parallel}\|_{\mathcal{H}}^2 + \|f_{\perp}\|_{\mathcal{H}}^2).$$

Minimizing $J(f)$ with respect to f, we observe that adding $f_{\perp}$ increases the regularization term without reducing the empirical risk term. Therefore, f must lie entirely within the span of the kernel functions evaluated at the data points:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x).$$

Significance of the Representer Theorem. The Representer Theorem indicates that for any convex loss function over the entire RKHS $\mathcal{H}$, the Empirical Risk Minimization problem is solved by a function that resides in the span of the basis functions defined by the examples. Consequently, the minimization process can be conducted within a finite-dimensional subspace of the full Hilbert space.

The specific optimization procedure employed may vary, and the choice of loss function will generally depend on the particular problem being addressed. However, the kernel framework allows one to define an abstract class of functions for minimization and guarantees a solution of a specific form, which is computationally efficient given the input examples. Additionally, it provides generalization guarantees, as will be discussed later; this framework can also be extended to approximate functions with multidimensional outputs.

Multidimensional output kernel methods extend kernel techniques to vector-valued functions, enabling the learning of functions that produce multiple outputs. These methods utilize operator-valued kernels to manage vector-valued functions within an RKHS framework.

Operator-Valued Kernels. An operator-valued kernel $K : X \times X \to L(Y)$ maps pairs of inputs to linear operators on the output space Y. For instance, for a vector-valued function $f : X \to Y$ where Y is a Hilbert space, the kernel K can be employed to define:

$$f(x) = \sum_{i=1}^{n} K(x_i, x)\alpha_i$$

where α_i are elements of Y.

This flexibility facilitates the modeling of complex dependencies among multiple outputs, making operator-valued kernels particularly advantageous for multitask learning and structured output prediction.

The Kernel Trick. Linear algorithms such as Support Vector Machines [Vap95] and Principal Component Analysis [Hot33] are very well understood and have both efficient algorithms and clear generalization theories on the obtained models. May of such algorithms rely only on the computation of inner products between data points, which enables to extend such algorithms from typical Euclidean Spaces to Hilbert Spaces without much variation, and in turn enables non-linear applications of these algorithms by considering a mapping function $\phi : X \to \mathcal{H}$ from an input space to a high-dimensional Hilbert Space, on which the linear algorithm itself is applied.

The kernel trick is a way to operate implicitly in high-dimensional feature spaces without explicitly mapping the data to those spaces. When the Hilbert Space is an RKHS infact, the kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$ can be used instead of the explicit computation of the inner product $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle_{\mathcal{H}}$.

This property significantly reduces the computational burden and enables the application of algorithms to problems where the explicit mapping would be computationally infeasible.

Similarity to Linear Models. The parallels between kernel methods and linear models imply that many techniques and insights developed for linear models can be extended to kernel methods: regularization techniques, dual formulations, and gradient-based optimization methods commonly used in linear models can be adapted for kernel methods with minor modifications.

2.2 Kernel Methods Generalization Bounds

Mathematical Setting. Let $\mathcal{H}$ be a Reproducing Kernel Hilbert Space (RKHS) with kernel K, and let $\{(x_i, y_i)\}_{i=1}^n$ be a training set drawn independently and identically distributed (i.i.d.) from a distribution $\mathcal{D}$. Our goal is to minimize the regularized empirical risk:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2.$$

Introduction to Generalization Bounds. Generalization bounds are crucial in machine learning as they provide a measure of how well a model trained on a finite dataset will perform on unseen data. This is important because, in practice, we seek models that not only fit the training data well but also generalize effectively to new, unseen examples.

These bounds aim to quantify the difference between the empirical risk

$$R_n(f) = \frac{1}{n} \sum_{i=1}^{n} L(x_i, y_i, f(x_i)),$$

which represents the average loss on the training set, and the expected risk

$$R(f) = \mathbb{E}_{(x,y)\sim\mathcal{D}}[L(x, y, f(x))],$$

which denotes the expected loss over the entire data distribution. This quantification is achieved by employing tools from statistical learning theory, such as concentration inequalities.

Form of the Generalization Bounds. A typical generalization bound for the aforementioned setting considers a confidence level $\delta > 0$. It allows us to assert that, given n examples, with high probability (at least $1 - \delta$), we have

$$R(f) \leq R_n(f) + \mathcal{O}\left(\frac{\|f\|_{\mathcal{H}}}{\sqrt{n}} \log\left(\frac{1}{\delta}\right)\right),$$

where the RKHS norm of a function $f = \sum_i \alpha_i K(x_i, x)$ is defined as:

$$\|f\|_{\mathcal{H}}^2 = \sum_{i,j} \alpha_i \alpha_j K(x_i, x_j).$$

The bounds in this form highlight a trade-off between the empirical risk and the complexity term of the function space, which depends on the sample size and the norm of the function in the RKHS. This is also why Empirical Risk Minimization is typically performed with regularization over the norm of the resulting function.

Below, we provide a sketch of how one would prove such generalization bounds, but we refer interested readers to the treatment by Bousquet and Elisseeff [BE02] regarding stability-induced bounds and to the book by Scholkopf and Smola [SS18] for a broader overview.

Rademacher Complexity. Before presenting the proof, we need to briefly introduce the concept of Rademacher complexity by Bartlett and Mendelson [BM02], which measures the richness or complexity of a hypothesis class of functions $\mathcal{H}$. It is used to bound how well a hypothesis chosen from this class performs on unseen data. The intuition behind Rademacher complexity is that it quantifies how likely it is for a hypothesis class to fit random noise, providing insight into the capacity of the class to generalize.

Formally, for a set of training points $S = \{x_1, \ldots, x_n\}$, the empirical Rademacher complexity of a class of functions $\mathcal{H}$ is defined as:

$$R_n(\mathcal{H}) = \mathbb{E}_\sigma \left[\sup_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i h(x_i) \right],$$

where σ_i are i.i.d. Rademacher random variables taking values ± 1 with equal probability. The supremum is attained by a function h that perfectly correlates with the noise σ, thus representing how easily a specific hypothesis class can fit random noise labels.

Proof Sketch. We decompose the generalization error into empirical risk and complexity terms:

$$R(f) - R_n(f) = (R(f) - \mathbb{E}[R_n(f)]) + (\mathbb{E}[R_n(f)] - R_n(f)),$$

where the expectations are taken with respect to the sampling of the dataset from the unknown data-generating distribution $\mathcal{D}$. The first term reflects how well the estimated function from n samples approximates that obtained from the full distribution, thus depending on the complexity of the estimating function class. The second term is a concentration term that consists of the difference between the expected empirical risk and the observed empirical risk, indicating how much the empirical risk calculated on a finite sample deviates from its expectation.

We apply a concentration inequality, such as Bernstein's or Hoeffding's, to bound the concentration term $\mathbb{E}[R_n(f)] - R_n(f)$, and we use the Rademacher complexity $\mathcal{R}_n(\mathcal{H})$ of the RKHS to bound $\mathbb{E}[R(f) - R_n(f)]$. In the context of Kernel Methods, the hypothesis space is the RKHS, and the Rademacher complexity can be controlled by properties of the kernel, leading to a bound that depends on the norm $\|f\|_{\mathcal{H}}$ of the function in the RKHS.

By combining the two terms, we obtain a probabilistic bound of the form $\mathbb{P}\left(R(f) - R_n(f) \leq \epsilon\right) \geq 1 - \delta$, where ϵ depends on the complexity of the hypothesis class and the sample size n. For kernel methods, this bound typically takes the form $\epsilon = \mathcal{O}\left(\frac{\|f\|_{\mathcal{H}}}{\sqrt{n}}\right)$.

2.3 Neural Tangent Kernels

Finally, we arrive at the introduction of Neural Tangent Kernels by Jacot, Gabriel, and Hongler [JGH18], which relate randomly initialized Deep Neural Networks in their infinite width limit to the behavior of Kernel Methods. We will then continue with the methods used to calculate these Tangent Kernels, the accuracy that can be derived from them, and the strategies to accelerate the computation of Kernel Methods, which are traditionally hindered by an $O(n^3)$ computational complexity.

Linearization of Models at Initialization. Neural networks are typically initialized with parameters drawn from specific random distributions. This random initialization is crucial for breaking symmetry and ensuring a good starting point for the optimization process. We can consider feed-forward networks as a single function $f : X \times \Theta \to Y$ that takes an input $x \in X$, some parameters $\theta \in \Theta$, and outputs a prediction $\hat{y} \in Y$.

At initialization, the network's parameters θ_0 are sampled from a distribution, often Gaussian, to ensure proper variance scaling and gradient flow during training. We can approximate the behavior of the network at initialization by exploiting the Gaussian distribution of weights and linearizing it in the parameter space around the initial parameters θ_0:

$$f(x;\theta) \approx f(x;\theta_0) + \nabla_\theta f(x;\theta_0) \cdot (\theta - \theta_0).$$

This approximation becomes particularly powerful for very wide networks, where the central limit theorem implies that the network's behavior can be effectively captured by its first-order dynamics, as shown by Lee et al. [Lee+19]. The linearization essentially transforms the problem into a linear regression in a high-dimensional parameter space, simplifying the analysis and understanding of the network's behavior at the onset of training.

While we do not delve deeply into the treatment of linearization, it is important to keep this in mind, as it allows us to treat wide networks essentially as linearizations for theoretical analysis.

Neural Tangent Kernel. Define the NTK $K(x, x';\theta)$ as the inner product of the gradients of the network outputs with respect to the parameters for two inputs x and x':

$$K(x, x';\theta) = \nabla_\theta f(x;\theta) \cdot \nabla_\theta f(x';\theta).$$

Neural Networks Evolve as Kernel Optimization under Gradient Descent. Let us denote by $\theta_t \in \Theta$ the specific parameters at time t, and define

$$\hat{y}_j^t := f(x_j;\theta_t).$$

We aim to minimize the Network Empirical Loss

$$\mathcal{L}(\theta) := \sum_{i=0}^{n} \ell(x_i, y_i, f(x_i;\theta))$$

via gradient descent, which means that we impose

$$\dot{\theta}_t := -\eta \nabla_\theta \mathcal{L}(\theta) = -\eta \sum_{i=0}^{n} \frac{\partial \ell}{\partial \hat{y}}(x_i, y_i, f(x_i;\theta_t)) \cdot \frac{\partial f}{\partial \theta}(x_i;\theta_t),$$

and the predictions evolve accordingly to

$$\frac{\mathrm{d}}{\mathrm{d}t}\hat{y}_j^t = \frac{\partial f}{\partial \theta}(x_j;\theta_t) \cdot \dot{\theta}_t = -\eta \sum_{i=0}^{n} \frac{\partial \ell}{\partial \hat{y}}(x_i, y_i, f(x_i;\theta_t)) K(x_i, x_j;\theta_t),$$

exactly as they would under gradient descent applied to a Kernel Empirical Minimization Prob-

lem. The exception here is that the kernel K is variable and depends on θ_t, but for very wide networks, it can be shown that this kernel is effectively constant.

The Tangent Kernel is Constant in Very Wide Networks. Jacot, Gabriel, and Hongler [JGH18] demonstrate that for neural networks that are sufficiently wide, the Neural Tangent Kernel remains constant during training by proving that, as the width N of the network layers increases, the NTK converges to a deterministic limit.

They assume that the weights W are initialized according to a normal distribution with mean zero and variance inversely proportional to the layer width, $W \sim \mathcal{N}(0, \sigma_w^2/N)$, and that the biases are initialized similarly, $b \sim \mathcal{N}(0, \sigma_b^2)$. This initialization ensures that the variance of the pre-activations remains stable as the network width increases.

During training, the NTK typically evolves according to the dynamics of the gradient descent updates. However, as the width N approaches infinity, the fluctuations in the NTK due to randomness in initialization and training diminish. They are able to show that

$$\lim_{N \to \infty} K(x, x'; \theta_t) = K(x, x'; \theta_0)$$

for all training steps t. This implies that during training, the NTK does not change, effectively linearizing the training dynamics and indicating that the learning dynamics for very wide networks can be fully captured by the initial NTK.

Computational Procedures for NTK Calculation. Once seen how powerful and general the Kernel Methods are, together with the fact that the initial kernel of very wide neural networks is deterministic and doesn't evolve during training, one question immediately emerges: what happens if we are able to effectively compute the NTKs for very wide networks of a specific architecture, say CNNs, and try to apply those to classify images? Will these Kernel-based Methods be as good as conventional fully trained Convolutional Networks?

Arora et al. [Aro+19] showed an efficient procedure to calculate the NTKs from the description of the neural network itself via dynamic programming, and also provide a non-asymptotic proof that the NTKs capture the behavior of a fully-trained wide network under weaker conditions that those that were obtained by Jacot, Gabriel and Hongler.

Leveraging the newly introduced Convolutational NTKs, the work done by Lee et al. [Lee+20] went on to check whether finite-width networks work better or worse than infinite-width ones, with mixed results. We reproduce here verbatim their own words, highly informative on what they found.

> "We conduct a comprehensive empirical study that examines the relationship between wide neural networks and kernel methods, addressing several open questions regarding infinitely wide neural networks. Our experimental findings indicate that while kernel methods outperform fully connected finite-width networks, they fall short compared to convolutional finite-width networks. Furthermore, neural network Gaussian process (NNGP) kernels frequently surpass neural tangent (NT) kernels in performance. Centered and ensembled finite networks demonstrate reduced posterior variance, causing them to behave more like infinite networks. However, the

use of weight decay and large learning rates disrupts the correspondence between finite and infinite networks.

Additionally, we discover that the NTK parameterization outperforms the standard parameterization for finite-width networks, while diagonal regularization of kernels exhibits behavior akin to early stopping. We also note that limitations in floating-point precision hinder kernel performance beyond a certain dataset size, and that regularized ZCA whitening enhances accuracy. Interestingly, the performance of finite networks is influenced by width in a non-monotonic manner, which is not entirely captured by the double descent phenomenon. Moreover, CNN equivariance proves beneficial only for narrow networks that are distant from the kernel regime.

Our experiments further motivate an improved layer-wise scaling for weight decay, which enhances generalization in finite-width networks. Finally, we propose refined best practices for utilizing NNGP and NT kernels for prediction, including a novel ensembling technique. By implementing these best practices, we achieve state-of-the-art results on CIFAR-10 classification for kernels corresponding to each architecture class we investigated." – Lee et al. [Lee+20]

NTK as one of the many possible Parameterizations. The NTK parameterization differs from the standard one by the usage of a $\frac{1}{N}$ constant in the weight variance, that was inserted to ensure the existence of a limit for $N \to \infty$. However, there are other ways to ensure the existence of that limit by parameterizing in a different way the initialization distributions of weights so that the limit is always a Kernel, but a different one.

Yang and Hu [YH21] show that the NTK parameterization is not able to learn features in the limit, and propose simple modifications to the standard parameterization to allow for feature learning in the infinite-width limit, and show that this new parametrization called μP is able to outperform both the NTK baselines and finite-width networks, with the latter approaching the infinite-width feature learning performance as width increases.

Speeding up Kernel Methods. Kernel methods provide a robust theoretical framework for understanding generalization in machine learning. By transforming neural networks into kernel methods, we can leverage the rich theory of kernels to gain insights into the behavior of these models. Unfortunately, classical algorithms to train Kernel Methods suffer from a computation complexity that is cubic in the number of examples, and thus they have remained relegated to nice theoretical tools, instead of being effectively used in the very big data regime.

More recently in a series of papers, Rudi, Camoriano, and Rosasco; Rudi, Carratino, and Rosasco; Meanti, Carratino, Rosasco, and Rudi [RCR15; RCR17; MCRR20] have offered methods to speed up kernel training, making it feasible to apply kernel methods to large-scale datasets without loss on the generalization guarantees.

One of the key techniques for accelerating kernel methods is the Nyström approximation [RCR15]. This method involves subsampling the kernel matrix to reduce computational complexity. Specifically, the Nyström method approximates the kernel matrix $K = \mathcal{K}(X, X)$ by selecting a subset $Y \subseteq X$ of the data points and computing a low-rank approximation, where $\mathcal{K}(X, X) \approx \mathcal{K}(X, Y)\mathcal{K}(Y, Y)^{+}\mathcal{K}(Y, X)$, where $\mathcal{K}(X, Y)$ is a matrix of kernel evaluations between the selected subset and the entire dataset, and $^{+}$ denotes the Moore-Penrose pseudoinverse.

Rudi, Camoriano, and Rosasco [RCR15] provide theoretical results showing that subsampling methods can achieve the same (optimal) learning error as kernel regularized least squares, provided the subsampling level is suitably chosen, which are further supported by empirical results: experiments on benchmark datasets show that subsampling techniques, such as the Nyström approximation and leverage score sampling, can achieve state-of-the-art performance while significantly reducing computational costs. These methods provide a form of computational regularization, tailoring the computational resources to the generalization properties of the data at hand.

We refer the interested reader to the papers detailing both the final result of this line of research in the form of a method for optimal large scale kernels by Rudi, Carratino, and Rosasco [RCR17] and the techniques used for handling billions of points efficiently by Meanti, Carratino, Rosasco, and Rudi [MCRR20].

Chapter 3

DNN and Optimization

3.1 Overview of Optimization Theory

Optimization theory is a vital discipline in mathematics and applied science that focuses on identifying the best solution from a set of feasible alternatives. This process involves maximizing or minimizing an objective function by systematically selecting input values from an allowable set and calculating the function's value. In the field of Machine Learning, optimization algorithms are essential for training models, where the objective function typically represents a loss or cost that needs to be minimized.

Optimization problems can be categorized based on the characteristics of the objective function and the constraints involved. These categories include linear and nonlinear problems, convex and non-convex problems, as well as unconstrained and constrained problems. Each type of problem has specific optimization algorithms tailored for it, making optimization a diverse and rich field within mathematics.

Single-Level Optimization. Single-level optimization addresses problems where the objective function must be minimized. Mathematically, a single-level optimization problem can be formulated as:

$$\begin{aligned} & \min_{x \in \mathbb{R}^n} f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_j(x) = 0, \quad j = 1, \ldots, p \end{aligned}$$

where $f(x)$ is the objective function, $g_i(x)$ represents the inequality constraints, and $h_j(x)$ denotes the equality constraints. The objective is to find the vector x that minimizes the objective function while satisfying all constraints. This scenario is typical in supervised learning for classification tasks.

Bilevel Optimization. Bilevel optimization involves two levels of optimization problems, one nested within the other, and is frequently employed in game theory to model leader-follower scenarios. In this context, the leader optimizes their objective while considering the follower's

response, resulting in a hierarchical decision-making process.

Mathematically, a bilevel optimization problem can be expressed as:

$$\begin{aligned} &\min_{x\in\mathbb{R}^n} \max_{y\in\mathbb{R}^m} F(x,y) \\ \text{subject to} \quad &g_i(x,y) \leq 0, \quad i = 1,\ldots,p \\ &h_j(x,y) = 0, \quad j = 1,\ldots,q \end{aligned}$$

where $F(x,y)$ is the function that one player aims to minimize, while the other seeks to maximize it, subject to the constraints $g_i(x,y)$ and $h_j(x,y)$. This type of problem is relevant in the optimization processes for Generative Adversarial Networks [Goo+14].

First-Order Algorithms for Non-Convex Functions. First-order optimization algorithms utilize the gradient of the objective function to direct the search for the optimum. Gradient Descent is a widely adopted first-order method applicable to both convex and non-convex functions, exhibiting slightly different properties: in the case of convex functions, it is guaranteed to find the minimum, whereas in non-convex problems, it may become trapped at a saddle point.

The literature on first-order optimization algorithms for both convex and non-convex problems is extensive. For those interested, we recommend classical texts on convex optimization by Nesterov [Nes13] and Boyd and Vandenberghe [BV04], a focused work on non-convex optimization by Sra, Nowozin, and Wright [SNW11], and a book on online convex optimization by Hazan et al. [Haz+16]. In the following sections, we will introduce the main notations to assist the reader in understanding the subsequent material.

Gradient Descent updates the solution x iteratively by moving in the direction opposite to the gradient of the objective function $f(x)$ according to the rule

$$x_{k+1} = x_k - \alpha \nabla f(x_k),$$

where α is referred to as the step size or learning rate. The selection of α is crucial, as it significantly impacts the convergence rate and stability of the algorithm.

Stochastic Gradient Descent (SGD) is a variant of Gradient Descent that employs a random subset of data points to compute the gradient, enhancing scalability for large datasets. The update rule for SGD is:

$$x_{k+1} = x_k - \alpha \nabla f_{i_k}(x_k),$$

where f_{i_k} denotes the objective function evaluated at a randomly chosen data point i_k. Although the stochastic nature of SGD may reduce the accuracy of each update, it compensates with increased speed, as only a single gradient computation for one sample is required. This method is widely used in training large-scale machine learning models.

Momentum and acceleration techniques improve the performance of first-order algorithms. Momentum incorporates a fraction of the previous update into the current update, facilitating faster convergence and reducing oscillations:

$$v_{k+1} = \beta v_k + \alpha \nabla f(x_k)$$

$$x_{k+1} = x_k - v_{k+1}$$

where β is the momentum coefficient. Variations of momentum are referred to as accelerated methods, such as Nesterov's Accelerated Gradient (NAG) [Nes83], which provide stronger guarantees on faster convergence.

Preconditioning modifies the gradient direction to account for the geometry of the problem, thereby enhancing convergence. Preconditioned Gradient Descent employs a preconditioner matrix P:

$$x_{k+1} = x_k - \alpha P^{-1} \nabla f(x_k)$$

to accelerate convergence; this approach is particularly beneficial when one can identify easily invertible preconditioning matrices that closely approximate the Hessian of the optimization objective.

Second-Order Algorithms for Convex Functions. Second-order algorithms utilize both the gradient and the Hessian of the objective function, providing more accurate information about the curvature. It is crucial that the function is convex, ensuring that its Hessian is a positive definite matrix. This guarantees that the direction of the update remains in the half-space that leads to a decrease in the objective function at each step.

Newton's Method is a well-known second-order technique for convex optimization problems, characterized by the update rule:

$$x_{k+1} = x_k - H_f(x_k)^{-1} \nabla f(x_k)$$

where $H_f(x_k)$ represents the Hessian matrix of $f(x)$ at x_k. Newton's Method achieves quadratic convergence, making it highly efficient for problems where the Hessian can be easily computed and inverted.

Quasi-Newton methods approximate the Hessian matrix, which reduces computational costs while still retaining some advantages of second-order information. Notable quasi-Newton methods include Davidon-Fletcher-Powell (DFP) [Dav91; FP63], which updates an approximation of the Hessian inverse using differences in gradients; Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Bro70b; Bro70a; Fle70; Gol70; Sha70], which refines the Hessian inverse approximation with a rank-two update; Limited-memory BFGS (L-BFGS) [LN89], which stores only a limited number of vectors to approximate the Hessian, making it suitable for large-scale problems; and Symmetric Rank-One (SR1) [CGT91], which updates the Hessian approximation using a symmetric rank-one update.

For instance, the update rule for BFGS is:

$$H_{k+1} = H_k + \frac{y_k y_k^T}{y_k^T s_k} - \frac{H_k s_k s_k^T H_k}{s_k^T H_k s_k}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.

In the final part of this section, we will introduce two lesser-known quasi-Newton methods: the BHHH algorithm [BHHH74] and Conjugate Gradient Descent [HS+52; FR64], which will be significant in our subsequent discussions.

BHHH Algorithm. The BHHH algorithm, named after Berndt, Hall, Hall, and Hausman [BHHH74], is a specialized optimization method frequently employed in econometrics for maximum likelihood estimation. It is particularly advantageous when the likelihood function is complex and challenging to maximize using conventional techniques.

The BHHH algorithm iteratively updates parameter estimates by utilizing the outer product of score vectors. The update rule is given by:

$$\theta_{k+1} = \theta_k + \left(\sum_{i=1}^{n} s_i(\theta_k) s_i(\theta_k)^T \right)^{-1} \sum_{i=1}^{n} s_i(\theta_k),$$

where $s_i(\theta_k)$ represents the score vector (the gradient of the log-likelihood) for the i-th observation at the parameter estimate θ_k. This method leverages the information matrix to provide a more accurate update direction, thereby enhancing convergence.

Conjugate Gradient Descent. Conjugate Gradient Descent (CGD) [HS+52; FR64] is a first-and-a-half-order optimization algorithm designed for solving large-scale linear systems and quadratic optimization problems. It integrates principles of gradient descent with conjugacy based on an inner product, enabling efficient navigation of the optimization landscape when the local quadratic geometry is partially known, without the need for the costly computation of Hessian inversions.

CGD iteratively refines the solution by considering the direction of the previous step and ensuring that each new direction is conjugate to the previous ones. The update rule consists of two key steps: computing the search direction and updating the solution. The search direction p_k is defined as:

$$p_k = -\nabla f(x_k) + \beta_k p_{k-1},$$

where β_k is a scalar that ensures conjugacy, and the solution is then updated as

$$x_{k+1} = x_k + \alpha_k p_k,$$

with α_k being the step size determined by either a line search or a fixed constant.

The conjugacy condition is analytically resolved, leading to the establishment of a step-size selection rule, among which the most commonly used are [HZ06]:

1. **Exact Line Search**: Determines the optimal step size by minimizing the objective function along the search direction.
2. **Fletcher-Reeves**: Utilizes the gradient norm ratio

$$\beta_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}.$$

3. **Polak-Ribiere**: Adjusts β_k using

$$\beta_k = \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{\nabla f(x_{k-1})^T \nabla f(x_{k-1})}.$$

4. **Hestenes-Stiefel**: Computes β_k based on the change in gradients and the previous search

direction

$$\beta_k = \frac{\nabla f(x_k)^T (\nabla f(x_k) - \nabla f(x_{k-1}))}{p_{k-1}^T (\nabla f(x_k) - \nabla f(x_{k-1}))}.$$

CGD is particularly beneficial for large-scale problems because it requires only matrix-vector multiplications, thus avoiding the explicit computation and storage of large matrices, which would be necessary in a full second-order method. This makes it suitable for high-dimensional optimization challenges in machine learning and scientific computing. The method typically progresses more significantly than Gradient Descent in each iteration, especially for well-conditioned problems where there is a substantial difference in curvature across various conjugate directions.

3.2 DNN as an Optimization Problem

Setting and Notation. Consider the typical supervised learning setting, where n examples $(x_i, y_i) \in X \times Y$ are sampled i.i.d. from an unknown probability distribution $\mathcal{D} \in \mathcal{P}(X \times Y)$; a parametric function $f : X \times \Theta \to \hat{Y}$ and a loss function $\ell : X \times Y \times \hat{Y} \to \mathbb{R}$ convex in the third argument are given; we are interested in the setting of parameters $\theta \in \Theta$ that results in the smallest possible loss, i.e.

$$\theta^* := \operatorname*{argmin}_{\theta \in \Theta} \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(x_k, y_k, f(x_k, \theta))]. \tag{3.1}$$

Since the distribution $\mathcal{D}$ is unknown, we will typically rely on the minimization of the structural risk given by the loss on the empirical examples together with a (usually convex) regularized $R : \Theta \to \mathbb{R}$ expressing our preference for certain areas of the parameters:

$$\operatorname*{argmin}_{\theta \in \Theta} \mathcal{L}(\theta) := \frac{1}{n} \sum_{k=1}^{n} \ell(x_k, y_k, f(y_k, \theta)) + R(\theta). \tag{3.2}$$

This makes the supervised training of DNNs a single-level optimization problem, albeit a non-convex one due to the highly non-linear nature of the neural network $f(x_i; \theta)$. Common choices for the loss function ℓ include the mean squared error for regression tasks and the cross-entropy loss for classification tasks.

As previously mentioned, the optimization problem can be approached using some form of Gradient Descent, where the parameter updates are given by:

$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}(\theta_t),$$

with α representing the learning rate and $\nabla_\theta \mathcal{L}(\theta_t)$ being the gradient of the loss function.

Peculiarities of the Deep Network Optimization Problems. For our later treatment of supervised learning as an optimization problem, it is essential to notice the peculiarities of Deep Learning optimization problems with respect to classical convex problems that are typically studied, as if one wants to develop an algorithm specifically tailored to them, it has to respect their defining properties.

The first thing that differentiates Deep Networks from generic convex optimization is the role of randomness in the initialization of the parameters θ, which allow us to analyze that part of the network using the Central Limit Theorem (or, more generally, to exploit phenomena of concentration of measure) and a first-order locally valid linearization of the network function; the role of randomness isn't just relagated to the initialization, but can continue throught the whole training, as we have seen in the treatment of Neural Tangent Kernels in Chapter 2.

On the other hand, the loss function ℓ is convex in the third argument, and one would like to use the fact that convex functions have a positive definite Hessian to obtain a proper local approximation to the original minimization problem.

Bilevel Optimization in the Context of GANs. Generative Adversarial Networks (GANs) [Goo+14] exemplify bilevel optimization in machine learning, where two neural networks—the generator G and the discriminator D—are trained simultaneously through an adversarial process. The objective is to find a Nash equilibrium [Nas50] in a minimax game where G generates realistic data samples, and D distinguishes between real and fake samples. The optimization problem for GANs can be formulated as:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))].$$

In this setting, the discriminator D aims to maximize its ability to correctly classify real samples from the data distribution p_{data} and fake samples generated by G. Concurrently, the generator G aims to minimize the discriminator's ability to make these classifications, effectively trying to "fool" the discriminator. This adversarial training process enables GANs to generate highly realistic samples, making them powerful tools for generative modeling.

The training of GANs typically alternates between updating D and G using gradient-based optimization methods, such as Stochastic Gradient Descent-Ascent [BGBL23] or Competitive Gradient Descent by Schäfer and Anandkumar [SA19].

Generative Adversarial Networks (GANs) Goodfellow et al. [Goo+14], have shown remarkable success in generating high-quality samples across diverse domains. However, standard GANs often suffer from training instabilities and mode collapse, where the generator fails to capture the diversity of the data distribution. To address these challenges, Arjovsky, Chintala, and Bottou [ACB17] proposed the Wasserstein GAN (WGAN), which use the Wasserstein distance derived from optimal transport problems as a new metric for training the GANs.

The Wasserstein GAN reformulates the min-max optimization of standard GANs into the following form:

$$\min_G \max_{f\in\mathcal{F}_L} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{z\sim\mathbb{P}_z}[f(G(z))],$$

where G is the generator, f is the critic (or discriminator), $\mathcal{F}_L$ is the set of 1-Lipschitz functions (enforced via constraints such as weight clipping), $\mathbb{P}_r$ is the real data distribution, and $\mathbb{P}_z$ is the distribution of generated data. The generator G seeks to minimize the Wasserstein distance, while the critic f approximates it. For what concerns us it is important to notice that the type of problem that has to be solved remains the same, namely bilevel min-max optimization.

The Wasserstein distance offers a more principled approach to measuring the discrepancy between the generator's output distribution and the true data distribution. Unlike the Jensen-

Shannon divergence used in standard GANs, the Wasserstein distance provides smoother gradients for the generator during training, even in cases where the distributions have little or no overlap. This property significantly improves the stability of the training process and mitigates issues of mode collapse.

Chapter 4

Polyak–Łojasiewicz Theory

The Polyak–Łojasiewicz (PL) condition was originally introduced by Polyak [Pol63] as a general condition that ensures linear convergence of first-order optimization methods and was recently rediscovered in the deep learning community to address the need for convergence proofs of overparameterized non-convex models.

The line of works that tries to prove convergence to global minima in deep learning started with the influential work of Du et al. [Du+19], in which they rely on the existence of a descent direction at almost all points of the optimization landscape for an over-parametrized neural network in which every layer has more neurons than the amount of samples in the dataset.

This line of works has been followed by the revival of the Polyak–Łojasiewicz condition in more recent times, which has been used in many theoretical papers to prove convergence results for machine learning and deep learning models, e.g. in the work of Karimi, Nutini, and Schmidt [KNS16], and many variants have been introduced to address specific problems like the Proxy-Polyak–Łojasiewicz inequality by Frei and Gu [FG21] and the PL* condition by Liu, Zhu, and Belkin [LZB20] in which they relate the Tangent Kernel smallest eigenvalue to the PL coefficient μ, and are able to bound the coefficient from below in a neighbourhood of the initialization.

While theoretical results are important to understand the behavior of networks in the wide-limit setting, it is the author's opinion that all theoretical results should be also empirically tested for their adherence to reality in non-limit setups, and thus to understand whether the theory we have is sufficient to properly explain the behavior of real models, or if other phenomena occur that are not currently explainable.

In this Chapter, we will present the theory of Polyak–Łojasiewicz functions under the different aspects of convergence guarantees, conditioning of the optimization problem, and generalization results of the found minima, which will prepare the stage for our empirical analysis in Chapter 6 of how well real networks behavior can be explained by the predictions made by the theory of Polyak–Łojasiewicz functions.

4.1 Convergence

We introduce the background on PL functions that we use throughout the thesis, referring the reader to Karimi, Nutini, and Schmidt [KNS16] for known convergence results. We remark that

the presented theory concerning convergence speeds has an analogue in the two-sided PL setting of [YKH20], thereby extending its possible applications to minimax optimization problems, such as those present in Generative Adversarial Networks [Goo+14] optimization.

Introduction to the Polyak-Łojasiewicz Condition. The Polyak-Łojasiewicz condition bridges the gap between strong convexity and general non-convexity: it is particularly useful because it can ensure convergence even for non-convex functions, making it highly relevant in the training of neural networks and other machine learning models.

The PL Condition. The PL condition for a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is defined by the existence of a constant $\mu > 0$ such that

$$\forall x \in \mathbb{R}^n \qquad \frac{1}{2}\|\nabla f(x)\|^2 \geq \mu(f(x) - f^*) \tag{4.1}$$

where f^* is the global minimum value of f, and we define the optimality gap $\xi(x) := f(x) - f^*$. In this case we say that f is μ-PL.

This inequality implies that the norm of the gradient provides an upper bound for the distance to the optimal function value. Intuitively, the condition ensures that the function does not have flat regions and that the gradient is sufficiently informative to guide the optimization process towards a point of global minimum value.

Smoothness. Given a function $f : \mathbb{R}^n \to \mathbb{R}$ for an $L > 0$ we say that f is L-smooth if and only if for every $x, y \in \mathbb{R}^n$ we have

$$f(y) \leq f(x) + \langle \nabla f(x), y - x \rangle + \frac{L}{2}\|y - x\|^2.$$

Convergence under the PL Condition. PL functions enjoy the useful property of exponential convergence to a point of minimum value via common first-order otimization methods [KNS16; CR17; GGGM21]. Additionally, PL functions theory encompasses convex optimization because strongly convex functions satisfy the Polyak–Łojasiewicz condition with the same coefficient [KNS16]. In this thesis we only consider minimization via gradient descent, but the extension to other algorithms is standard.

Let be given a function f that satisfies the μ-PL condition and is L-smooth, then gradient descent with a fixed step size α converges linearly to a global minimum value point:

Lemma 1 (Convergence speed and radius for PL functions, [KNS16])**.** *Let $f : X \subseteq \mathbb{R}^m \to \mathbb{R}$ be μ-PL and L-smooth. Choose an initial point $x_0 \in X$ and let the sequence of iterates evolve according to the rule*

$$x^{k+1} = x^k - \frac{1}{L}\nabla f(x^k). \tag{4.2}$$

Letting $\gamma := 1 - \frac{\mu}{L}$, the optimality gap decreases exponentially following the formula

$$f(x^{k+1}) - f^* \leq \gamma\Big(f(x^k) - f^*\Big). \tag{4.3}$$

Moreover, the distance from the initial point is bounded by

$$\frac{1}{L}\left\|x^{k+1}-x^0\right\| \leq \sqrt{\frac{2(f(x^0)-f^*)}{L}}\frac{1}{1-\sqrt{\gamma}}. \tag{4.4}$$

Proof. We start by evaluating the change in the function value in terms of the L-smoothness:

$$f(x_{k+1})-f(x_k) \leq \nabla f(x_k)^T(x_{k+1}-x_k)+\frac{L}{2}\|x_{k+1}-x_k\|^2.$$

Then by substituting the gradient descent update rule

$$x_{k+1}-x_k = -\alpha\nabla f(x_k)$$

in the above equation we get

$$f(x_{k+1})-f(x_k) \leq -\alpha\|\nabla f(x_k)\|^2+\frac{L\alpha^2}{2}\|\nabla f(x_k)\|^2 = -\alpha\left(1-\frac{L\alpha}{2}\right)\|\nabla f(x_k)\|^2.$$

And using the PL condition $\frac{1}{2}\|\nabla f(x_k)\|^2 \geq \mu(f(x_k)-f^*)$ we obtain:

$$f(x_{k+1})-f(x_k) \leq -2\alpha\mu\left(1-\frac{L\alpha}{2}\right)(f(x_k)-f^*),$$

which, if $\alpha \leq \frac{1}{L}$, becomes

$$f(x_{k+1})-f(x_k) \leq -\alpha\mu(f(x_k)-f^*),$$

and implies that

$$f(x_{k+1})-f^* \leq (1-2\alpha\mu)(f(x_k)-f^*).$$

Linear convergence is thus established. □

Other Convergence Results. Karimi, Nutini, and Schmidt [KNS16] prove linear convergence of various gradient based optimization algorithms for a diverse set of useful problems, including greedy coordinate descent, sign-based gradient descent, stochastic methods in the variance-reduced setting, and some proximal gradient methods. Chen, Yao, and Luo [CYL22] extend the Polyak–Łojasiewicz condition to the minimax optimization setting, and Reddi et al. [Red+16] analyze stochastic variance reduction methods on Polyak–Łojasiewicz functions.

4.2 Importance of the Lower Polyak–Łojasiewicz Condition

Our interest in the PL condition is motivated by its connections with the minimization algorithms: notice first that $\nabla f(x) = 0$ automatically implies that $f(z) = f^*$, and thus that x is a global minimum; moreover, the condition naturally arises when studying stochastic gradient descent on f.

Infact, consider the equation for stochastic gradient descent on f:

$$\dot{\theta}_t = -\eta \tilde{\nabla} f(\theta_t), \tag{4.5}$$

where $\tilde{\nabla} f(\theta_t)$ is an approximation of the full gradient at that point. If we consider $\tilde{\nabla} f(\theta_t) = \nabla f_I(\theta_t)$ where I is a random batch of b examples, we certainly have that $\mathbb{E}_I[\nabla f_I(\theta_t)] = \nabla f(\theta_t)$ and we can observe what happens when we study the logarithmic decrease of the optimality gap:

$$\mathbb{E}_I\left[\frac{d}{dt}(\log \xi(\theta_t))\right] = \frac{1}{\xi(\theta_t)}\mathbb{E}_I\left[\nabla f(\theta_t)\cdot \dot{\theta}_t\right] = -\frac{\eta}{\xi(\theta_t)}\mathbb{E}_I[\nabla f(\theta_t)\cdot \nabla f_I(\theta_t)] = -\eta\frac{\|\nabla f(\theta_t)\|^2}{\xi(\theta_t)}$$

which tells us that the PL condition is the minimal one ensuring an exponential convergence of the optimality gap under continuous stochastic gradient descent since it allows us bound its mean log decrease from above. This also holds on discrete-time stochastic gradient descent when assuming an upper condition [KNS16].

4.3 Relations to Deep Networks

In this section, we explore the connection between over-parameterized deep networks and Polyak–Łojasiewicz functions, following the exposition of Liu, Zhu, and Belkin [LZB20].

Recall the notation of empirical risk minimization problem introduced in Section 3.2. Liu, Zhu, and Belkin [LZB20] focus on the tangent kernel $K(\theta)$, which is the Gram matrix of the network's Jacobian matrix $J(\theta)$. For a given parameter θ, the Jacobian $J(\theta)$ is defined as:

$$J(\theta) = \left[\nabla_\theta f(x_1;\theta), \nabla_\theta f(x_2;\theta), \ldots, \nabla_\theta f(x_n;\theta)\right],$$

and the tangent kernel $K(\theta)$ is then given by:

$$K(\theta) = J(\theta)^T J(\theta).$$

To demonstrate that $\mathcal{L}(\theta)$ satisfies the PL condition, the authors show that the smallest eigenvalue of the tangent kernel $K(\theta)$ is bounded away from zero with high probability. This result is achieved through several key steps.

First, they establish that at initialization, due to the random initialization of parameters, the tangent kernel is well-conditioned, meaning that its smallest eigenvalue $\lambda_{\min}(K(\theta_0))$ is positive. This is a consequence of the randomness in the initialization, which ensures that the gradients $\nabla_\theta f(x_i;\theta)$ are sufficiently independent.

Next, they demonstrate that as training progresses, the parameters θ do not drift far from their initial values. This stability is crucial because it ensures that the well-conditioning of the tangent kernel is maintained throughout training. Mathematically, they show that for each step k of gradient descent, $\|\theta_k - \theta_0\| \leq \delta$ for some small δ. This is achieved using concentration inequalities and properties of random matrices.

They then leverage matrix perturbation theory to show that if the initial tangent kernel $K(\theta_0)$ is well-conditioned, the tangent kernel $K(\theta_k)$ at step k remains well-conditioned. Specifically,

they use the Weyl inequality to bound the change in the smallest eigenvalue:

$$|\lambda_{\min}(K(\theta_k)) - \lambda_{\min}(K(\theta_0))| \leq \|K(\theta_k) - K(\theta_0)\|_2,$$

where $\|\cdot\|_2$ denotes the spectral norm. Given the bounded change in parameters, the perturbation in the tangent kernel is also bounded, ensuring that $\lambda_{\min}(K(\theta_k))$ remains positive.

Finally, the authors conclude that since the smallest eigenvalue of the tangent kernel remains positive, the loss function $\mathcal{L}(\theta)$ satisfies the PL condition, ensuring that the gradient descent algorithm converges linearly to the global minimum.

Limitations. However, the approach has limitations. First, the reliance on over-parameterization means that the results are most applicable to very large networks. If the network is not sufficiently over-parameterized, the smallest eigenvalue of the tangent kernel may not remain positive, and the PL condition may not hold. Second, the analysis is probabilistic and relies on random initialization, which is critical for ensuring the initial well-conditioning of the tangent kernel. In practical applications, deviations from these assumptions can lead to failures in finding the global minimum due to specific realizations of the randomness in the initialization.

Furthermore, while the analysis provides significant insights for fully-connected networks, extending these results to other architectures, such as convolutional or recurrent networks, poses additional challenges. The intricate dependencies in such networks complicate the application of the tangent kernel analysis.

4.4 Conditioning

Conditioning is extremely important in general optimization theory, since it provides valuable information about the speed of convergence to the objective.

Conditioning Number of a Matrix. Let us recall that the conditioning number $\kappa(M)$ of a rectangular matrix M is the ratio between its highest singular value $\sigma_{\max}(M) = \lambda_{\max}(\sqrt{M^T M})$ and its lowest singular value $\sigma_{\min}(M) = \lambda_{\min}(\sqrt{M^T M})$.

In the case of Polyak–Łojasiewicz functions, we consider the conditioning of the Jacobian matrix, which is strictly related to the PL constant of the function, as the below Lemma shows.

Lemma 2 (Convergence and Conditioning, [LZB20, Theorem 4.1])**.** *Let $F : \Omega \subseteq \mathbb{R}^m \to \mathbb{R}^n$ be a function and $\mathcal{L} : \mathbb{R}^n \to \mathbb{R}$ a μ-PL and L-smooth function on $F(\Omega)$. Define $K(\theta) := \nabla F(\theta)^T \nabla F(\theta)$, which is a $n \times n$ positive semidefinite matrix and let $\lambda_* := \min_{\theta\in\Omega} \lambda_{min}(K(\theta))$, $\lambda^* = \max_{\theta\in\Omega} \lambda_{max}(K(\theta))$. Then $h := \mathcal{L} \circ F$ is $\mu\lambda_*$-PL and $L\lambda^*$-smooth on Ω since*

$$\begin{aligned} \|\nabla h(\theta)\|^2 &= \left\|\nabla\mathcal{L}(F(\theta))^T K(\theta) \nabla\mathcal{L}(F(\theta))\right\| && (4.6)\\ &\geq \lambda_{min}(K(\theta))\|\nabla\mathcal{L}(F(\theta))\|^2 && (4.7)\\ &\geq \lambda_*\mu(\mathcal{L}(F(\theta)) - \mathcal{L}_*) = \lambda_*\mu(h(\theta) - h_*). && (4.8) \end{aligned}$$

where $\mathcal{L}_* := \min_{\zeta\in F(\Omega)} \mathcal{L}(\zeta) = \min_{\theta\in\Omega} \mathcal{L}(F(\theta)) = h_*$. The bound on smoothness can be proved in a similar way by majorization with $\lambda_{max}(K(\theta))$.

Improvement of Conditioning with Depth. Agarwal, Awasthi, and Kale [AAK20] explored how the depth of neural networks affects their conditioning and optimization properties. For the first time, they proved that deeper networks lead to better-conditioned optimization problems, which facilitates more efficient training. In contrast, earlier studies had to limit the maximum depth of the networks they examined, as deeper architectures were detrimental to their theoretical analysis.

The provide results on conditioning at initialization and during training of wide neural networks, namely that conditioning improves exponentially with the depth of the network. Their results hold for very wide networks, where the networks' weights displace from their initialization can be bounded with $O(1/\sqrt{m})$, m being the number of weights in a single layer, and thus when the Neural Tangent Kernel well-approximates its infinite-width deterministic limit [JGH18].

Intuitively, this occurs because each layer in a neural network transforms the data in a manner that can only be controlled in terms of the mean, using principles from Mean Field Theory (Chapter 1). As the depth of the network increases, the transformations become more complex, but they also lead to a more even distribution of eigenvalues of the tangent kernel matrix, thereby improving the overall conditioning.

Due to the difficulty of extracting quantitative results for smaller networks from the work of Agarwal, Awasthi, and Kale [AAK20], we present the ideas of their arguments to give a bit of context on the qualitative experimental analysis that will be performed in Chapter 6.

Conditioning in Neural Networks. Let $X \subseteq \mathbb{R}^d$ and let n samples $z_i = (x_i, y_i) \sim \mathcal{D} \in \mathcal{P}(X \times Y)$ be extracted from a distribution $\mathcal{D}$, and let $f^m : X \times \Theta \to \mathbb{R}^k$ be a function representing the first m layers of a neural network, which is parametric in the weights $\theta \in \Theta \subseteq \mathbb{R}^t$. Let us define the function $F^m : \Theta \subseteq \mathbb{R}^t \to \mathbb{R}^{nk}$ by $F^m(\theta) := (f^m(x_1; \theta), \dots, f^m(x_n; \theta))$.

We are then interested in studying the eigenvalues of the two different Gram matrices

$$G_{ij}^{m;\theta} := \langle f^m(x_i; \theta), f^m(x_j; \theta) \rangle, \qquad K^{m;\theta} := \nabla F^m(\theta)^T \nabla F^m(\theta), \tag{4.9}$$

which are related with network optimization speed [AAK20], since $G^{m;\theta}$ is the kernel matrix when the neural network is interpreted as a feature extractor and only the last layer is trained; and $K^{m;\theta}$ is instead connect to the PL constant of the network via Lemma 2.

The main observation in the proof of Agarwal, Awasthi, and Kale [AAK20] is the following relation between the entries of two G matrices at different layers:

$$G^{m+1;\theta} = \hat{\sigma}(G^{m;\theta}) \tag{4.10}$$

where $\hat{\sigma}$ is the dual activation function defined by Daniely, Frostig, and Singer [DFS16] and is applied entrywise.

This observation allows to control the entries in the G matrices at different layers: off-diagonal entries converge to zero when iterating, while diagonal entries remain fixed at their initial values; this observation allows to bound the highest and lowest eigenvalues, and a similar idea applies to the K matrix via its relations with the deterministic limit of the Neural Tangent Kernel. More details can be found in Appendix B.1.3.

Conditioning during Training. Because of Lemma 2, convergence speed is ultimately connected with the minimum eigenvalue of the K matrix encountered during training, and it is thus interesting to study how conditioning evolves during the optimization trajectory.

Existing theories bound conditioning during training using the distance between weights at the current time and at the beginning of training, making use of the well-known fact that for invertible linear operators A and B such that $\sigma_{\min}(A)\|A - B\|_{\text{op}} < 1$ one has (see [Ma12]):

$$|\sigma_{\min}(A) - \sigma_{\min}(B)| \leq \|A - B\|_{\text{op}} \leq \|A - B\|_F, \tag{4.11}$$

where $\|\cdot\|_{\text{op}}$ is the operator norm and $\|\cdot\|_F$ is the Frobenius matrix norm.

Equation (4.11) can then be used with $A = G^{m;\theta(t_0)}$ and $B = G^{m;\theta(0)}$ to obtain the required bound, as the right hand side can be expanded in terms of distance between weights [LZB20]. Such an approach is essentially pessimistic, as it assumes that conditioning is optimal at the start and that it degrades during training.

4.5 Generalization results

We have already established the significance of generalization results for a comprehensive theory and their utility in applications related to Kernel Methods. In this section, we will focus on the available generalization results for Polyak–Łojasiewicz functions.

The Need for Generalization Bounds. The necessity of having generalization bounds for deep networks is highlighted by the work of Zhang et al. [Zha+16] in their paper "Understanding deep learning requires rethinking generalization." This study reveals a fundamental issue: neural networks can achieve zero training loss, meaning they can memorize the entire training set, yet still generalize effectively to unseen data. This finding is counterintuitive because classical learning theory, which relies on concepts such as VC dimension and Rademacher complexity, suggests that highly complex models capable of perfectly fitting the training data should overfit and perform poorly on new data.

The experiments conducted in the paper demonstrated that neural networks could memorize random labels while still achieving low test errors, indicating that standard measures of model complexity do not fully capture a network's ability to generalize. This discrepancy between empirical success and theoretical understanding underscores the need for robust generalization results to ensure that models perform reliably in real-world scenarios.

Generalization Guarantees via Stability. Stability is a concept introduced by Bousquet and Elisseeff [BE02] that refers to the sensitivity of an algorithm to small changes in the training set: an algorithm is considered stable if a small perturbation in the training data results in a small change in the output hypothesis.

The authors introduced the definition of uniform stability, which measures the maximum change in the algorithm's output when a single training example is replaced, formally defined as

$$\sup_{z \in Z} |L(A_S, z) - L(A_{S'}, z)| \leq \beta,$$

where A_S is the algorithm trained on the dataset S, $A_{S'}$ is the algorithm trained on S' (which differs from S by one example), L is the loss function, and β is a stability constant. They demonstrated that if an algorithm is uniformly stable with a small stability constant, it will generalize well.

Pointwise hypothesis stability is a weaker notion but still sufficient to establish generalization bounds; defined as

$$E_S[|L(A(S), z_i) - L(A(S_{-i}), z_i)|] \leq \gamma,$$

where S_{-i} is the dataset S with the i-th example removed.

Stability and Generalization in Polyak–Łojasiewicz Neural Networks. Charles and Papailiopoulos [CP18] use the stability framework to analyze neural networks by leveraging the geometric properties of loss functions satisfying the Polyak-Łojasiewicz condition. They demonstrate that under the PL condition, various first-order optimization methods exhibit stability properties that guarantee generalization. More formally:

Lemma 3 (Generalization Bound for PL risk, [CP18, Theorem 3])**.** *Suppose that for every randomly extracted dataset S of n elements, the empirical risk R_S is μ-PL, and that it is optimized by gradient descent. Let w_S^* be a point of minimum value towards which gradient descent is converging, whose existance is guaranteed by PL-ness of the objective, i.e. assume that $|R_S(w_S) - R_S(w_S^*)| \leq \varepsilon_A$. Moreover assume that $\ell(x, y, f(x; \cdot))$ is G-Lipschitz, then for any δ with probability at least $1 - \delta$ we have the estimates*

$$\beta_{ptw} \leq 2\sqrt{\varepsilon_A}\sqrt{\frac{2G^2}{\mu}} + \frac{1}{n-1}\frac{2G^2}{\mu} \tag{4.12}$$

$$|R_{\mathcal{D}}(w_S) - R_S(w_S)| \leq \sqrt{\frac{M^2 + 12Mn\beta_{ptw}}{2n\delta}} \tag{4.13}$$

where $0 \leq \ell(x, y, y') \leq M$ and $R_{\mathcal{D}}(w) := \mathbb{E}_{(x,y)\sim\mathcal{D}}[\ell(x, y, f(x; w))]$.

Proof Sketch. They show that if the gradient descent algorithm converges to a global minimum, the stability bound can be derived using the geometry of the loss function around the minima. Under the μ-PL condition, they prove that the algorithm's output changes little when the training set is perturbed. This is because the gradient's norm provides a strong measure of suboptimality, ensuring that as long as the gradient descent steps are small, the output hypothesis remains close to the optimal solution.

Part II

Contributions

General Picture. The thesis abstract contribution consists in a modular approach to understanding Deep Neural Networks, clearly delineating the roles of network architecture, initialization, required generalization bounds, and the optimization algorithm. It also proposes an optimization-derived hypothesis regarding the correct way to train a network: to incorporate generalization bounds as regularization within a Structured Risk Minimization framework, followed by identifying an effective algorithm for the resulting problem.

While this may appear to be an obvious way to structure the network optimization problem, the literature contains numerous examples where it is unclear if this is the ultimate goal of the methods presented, and whether they genuinely improve the situation concerning this specific objective. For instance, we introduced Mean Field Theory techniques in Chapter 1, which were employed to address variants of the vanishing or exploding gradient problem [BSF94], without critically assessing whether this approach effectively contributes to solving the network optimization problem. In this Part, we will observe that the vanishing and exploding gradients are merely symptoms of poor optimization conditioning, and that focusing on reducing this conditioning *tout court* is a more effective overall strategy, with specific effects depending on whether the chosen optimization algorithm employs some form of preconditioning.

We favor the optimization approach (in contrast to both the Tangent Kernel and Mean Field approaches) because it is applicable to individual networks, even though we must theoretically assume certain hypotheses about the model that are only satisfied in ensemble or infinite-width limit cases. However, these assumptions are necessary only for performing local measurements on relevant network quantities during the optimization trajectory. Nevertheless, we will illustrate the advantages of this approach through a couple of examples presented in this Part; we want to emphasize that these examples serve as an exploration of the natural consequences of the approach and are by no means exhaustive. At the end of this Part, we will discuss the interpretations of the results obtained and other potential future applications of this approach.

Modelling Assumptions. In practical terms, we outline some general assumptions regarding the weights of the model during training, which serve as a foundation for the theoretical analysis of network optimization. These assumptions are novel in themselves, provably true in limit settings, empirically validated on real networks, and provide a robust basis for a first-order development of the theory. We will demonstrate how these specific assumptions enable us to study the predictivity of the Polyak–Łojasiewicz Theory on exemplary networks, as well as derive the optimal local learning rate for use with SGD, which are the applications of the framework presented in this Part.

Connections with Deep Kernel Methods. In Chapter 5, we expose a theoretical parallel between Deep Neural Networks and Deep Kernel Methods. This framework allows for proving new properties of Deep Neural Networks without reference to their internal architecture, which is directly embedded in the form of the Kernel, thereby facilitating generalized proofs.

Predictivity of the Polyak–Łojasiewicz Theory. In Chapter 6, we assess the predictivity of the Polyak–Łojasiewicz Theory introduced in the first Part concerning convergence, conditioning, and generalization. This study is motivated by the need to preliminarily understand whether this specific optimization theory can effectively elucidate the behavior of neural net-

works during training. The significance of the Polyak–Łojasiewicz condition in the optimization analysis is underscored by its recovery as the most general inequality that ensures a linear rate of convergence when Stochastic Gradient Descent is employed, and the empirical results obtained are promising.

Semidefinite Landscape Approximation and Adaptive Learning Rate Tuning. In Chapter 7, using our custom modeling assumptions, we derive a positive semidefinite local quadratic approximation of the Deep Network objective within the context of the Polyak–Łojasiewicz Theory. This can be viewed as an analogue of the Berndt-Hall-Hall-Haussman algorithm [BHHH74] for likelihood functions. We justify the choice of assumptions based on specific network limit cases that can be analyzed mathematically, and we proceed to develop heuristics for selecting the optimal learning rate by considering the local quadratic approximation of the objective.

In the same Chapter, we utilize the previously obtained Semidefinite Approximation to create a new learning algorithm in which the learning rate of SGD is automatically adjusted during training. This algorithm outperforms a grid search over ten learning rates of SGD while requiring only twice the computational resources.

Future Research. The potential developments of the proposed theory can be broadly categorized into two main areas: adaptations of the theory to different optimization problems, such as bilevel optimization for Generative Adversarial Networks [Goo+14], Continual Learning [MC89], and Reinforcement Learning [Sut18]; and a deeper exploration toward hyperparameter-free learning algorithms and strategies for achieving faster convergence by optimizing initialization methods to reduce conditioning. These topics will be elaborated upon in the final chapter of this part.

Chapter 5

Deep Neural Networks relations with Deep Kernel Methods

Deep neural networks archieved remarkable performances in recent years on a moltitude of tasks and a substantial research effort is being spent to uncover the fundamental reasons of such performances. The vast majority of these works lack generality with respect to the network architectures and results are restricted to feedforward models only. On the other hand more architectures are used on production systems, and much more are continuously being created to improve state-of-the-art results.

We argue that this results in duplication of efforts when a known result has to be rederived for another architecture and in a poorer understanding of the real reasons of success of certain combinations of architectures and hyperparameters, as well as in theoretical results having much smaller practical impact because of architecture changes.

In this Chapter we propose a general framework based on Deep Kernel Methods of Bohn, Rieger, and Griebel [BRG19] for the study of neural networks separatedly from the architectural constraints, which are instead encoded as properties of the specific kernels. We show that our framework includes Fully Connected, Convolutional and Recurrent Layers.

We showcase its flexibility by deriving known results of Li, Ding, and Sun [LDS18] on the non existance of certain suboptimal local minima in wide networks in an architecture-independent way. We argue that the framework is terser than the single architectures definitions and we suggest that, via the usage of kernels, it could provide a way to insert prior knowledge into the network architecture.

5.1 Introduction

Deep Neural Networks have gained widespread adoption and have been shown to be capable of archieving superhuman performance in a moltitude of tasks, among which object recognition [CMS12] with multiple applications in the medical area [Lit+17], human face recognition [MWHN18], game playing, both real time multiplayer [Ber+19] and zero-sum turn-based games [Sil+16], as well as in performing repetitive surgical tasks [Van+10]. Despite this, their theoretical understanding remains limited [Zha+16] and many branches of research have tried to tackle

the problem from different angles. Out of these, two frameworks that have helped in obtaining a more in-depth understanding of neural networks are certainly the Mean Field Theory approach by Schoenholz, Gilmer, Ganguli, and Sohl-Dickstein [SGGS17] and Neural Tangent Kernels by Jacot, Gabriel, and Hongler [JGH18].

Both frameworks focus on the study of wide or infinite neural networks, which theoretical study seems to be easier, especially for what concerns their training process. Other theoretical works focus on the analysis of a particular kind of networks, such as Fully Connected Networks [LDS18], Convolutional Neural Networks [NH18] or ResNets [Du+19]. Due to the very specialized proofs, it is usually not clear whether such results also hold for other architectures or not, and results on recurrent networks are scarce.

This is certainly a problem in critical applications of neural networks, where a greater computational effort for training a wide network may be compensated by the available theoretical guarantees about training and generalization. However imagine that all available papers only prove this in the case of Fully Connected Networks, while the application requires Convolutional or Recurrent Networks. How should the intended "user" of such results proceed, if he has no results concerning the architecture of interest?

Moreover, from the point of view of the theory itself, a decoupling between network architectures and other network hyperparameters (width, depth, loss function, etc.) has to be introduced in order to be able to distinguish results relative to peculiar architectures, such as ResNets, from results that concern all neural networks under certain hypotheses.

The framework we propose in this Chapter tries to decouple these two aspects in a fairly natural way by using Concatenated Kernel Methods [BRG19]. We generalize a known result by Li, Ding, and Sun [LDS18] in this framework to show its flexibility.

Contributions. The contribution of the current work is the proposal of a framework in which general properties of neural networks can be proven in an architecture-independent way, capable of unifying also recurrent networks, for which only few results currently exist. Such framework allows for a better understanding of which properties of networks are due to the architecture being studied and which are instead more general. By tying finite neural networks to kernel methods, the framework also suggests how prior knowledge may be injected into neural networks architectures, even if this is not touched upon in the Chapter.

A second contribution is the proof of a generalization of a result by Li, Ding, and Sun [LDS18] about the disappearance of bad basins in wide networks. The novelty is that the proof is carried out in an architecture-independent way, thus effectively proving that the disappearance of bad basins is just a property of the width of the optimization problem and is not influenced even by the choice of architecture.

We prove that the set of weights for which the last layer kernel matrix is not of full rank is of zero measure, which enables us to prove a weak form of globality via an argument of Li, Ding, and Sun [LDS18].

The Chapter is structured as follows: in Section 5.2 we provide the reader with an introduction to Deep Kernel Methods, in Section 5.3 we explore the link between the proposed framework and existing neural network architectures (FCN, CNN, RNN), and set the notation which will be used troughout the Chapter. Finally Section 5.4 provides a proof in this framework of the

already cited result of Li, Ding, and Sun [LDS18]. Even if the result is known, we highlight that the proof itself is technically different and the obtained result holds for all the neural network architectures that fit in our framework, of which notable examples are FCNs, CNNs and RNNs.

We also want to emphasize that the current work is *not* based on Neural Tangent Kernels by Jacot, Gabriel, and Hongler [JGH18], which instead assumes infinite width networks to represent networks as a single "flat" kernel. Our framework instead preserves network structure and provides an alternative interpretation of neural networks.

5.2 Deep Kernel Methods

Deep Kernel Methods (DKM), introduced in the work of Bohn, Rieger, and Griebel [BRG19], basically consist in the concatenation of multiple kernel methods one after the other. We thereby set the notation that will be used in talking about DKMs and state a representer theorem for them. We refer the reader to Micchelli and Pontil [MP05] for details on vector-valued Reproducing Kernel Hilbert Spaces (RKHS).

Let $L \in \mathbb{N}$ be the number of concatenated kernels, and for $l = 1, \dots, L$ let $\mathcal{H}^l$ be the l-th RKHS of functions with finite dimensional domain Z_{l-1} and finite dimensional range Z_l. As common in other works, we will denote by $(\cdot, \cdot)_l$ the Hilbertian inner product in Z_l and by $\langle \cdot, \cdot \rangle_l$ the inner product in the RKHS $\mathcal{H}^l$, omitting the l when it is clear from the context. Given $d_0, \dots, d_L \in \mathbb{N}$ we will restrict ourself to consider the case in which $Z_l = \mathbb{R}^{d_l}$ for $l = 0, \dots, L$ and we will denote by e_i for $i = 1, \dots, d_l$ the standard base vectors of $\mathbb{R}^{d_l}$. In this context the associated kernel function to $\mathcal{H}^l$ will be $K^l : Z_{l-1} \times Z_{l-1} \to \mathcal{L}(Z_l, Z_l)$, where $\mathcal{L}(Z_l, Z_l)$ is the space of linear functions from Z_l to Z_l.

We are also given N observations $\mathcal{D} = \{(x_k, y_k)\}_{k=1,\dots,N} \subseteq Z_0 \times Z_L$ for $k = 1, \dots, N$ and we assume that an arbitrary loss function $\mathcal{L} : (Z_0 \times Z_L \times Z_L)^N \to \mathbb{R}$ is given, and we define the cost of a function $\tilde{f} : Z_0 \to Z_L$ relative to the loss function $\mathcal{L}$ as $C(f) = \mathcal{L}((x_k, y_k, \tilde{f}(x_k))_k)$.

Using a DKM we allow more flexibility in the functions that we want to fit, but still retaining the more rigid structure imposed by kernel methods. Despite this added flexibility a representer theorem has been derived by [BRG19], which allows to reduce some kinds of optimization problems on the functional space to finite-dimensional ones.

Theorem 1 (Representer theorem for Deep Kernel Methods, [BRG19])**.** *Given a DKM and an arbitrary loss function $\mathcal{L}$ as per the above notation, let $\phi_1, \dots, \phi_L : \mathbb{R} \to \mathbb{R}$ be strictly increasing functions.*

Then for $f^l \in \mathcal{H}^l$, any minimizer $(f^l)_{l=1}^L$ of

$$\mathcal{L}((x_i, y_i, f^L \circ \dots \circ f^1(x_i))_i) + \sum_{l=1}^{L} \phi_l(\left\| f^l \right\|^2_{\mathcal{H}^l})$$

fulfills $f^l \in \tilde{V}^l \subseteq \mathcal{H}^l$ for $l = 1, \dots, L$ where

$$\tilde{V}^l = \operatorname{Span}\left\{ K^l(f^{l-1} \circ \dots \circ f^1(x_i), \cdot) e_{k_l} \mid k_l = 1, \dots, d_l \right\}$$

where K^l denotes the reproducing kernel of $\mathcal{H}^l$ and e_{k_l} is the k_l-th unit vector.

The representer theorem basically says that a minima for the loss function $\mathcal{L}$ can be found on a well-specified linear subspace $\tilde{V}^l$, that depends on the given input points and on previous layers. The resulting optimization problem is however non-convex and thus its solution seems unfeasible.

In this Chapter we prove some favorable properties of the loss landscape of a slight variation of the above minimization problem, where we require that the spaces at each level are actually fixed before starting the optimization, and assuming that the kernel functions are real analytic.

5.3 Linking DKMs to Neural Networks

In this section, after setting the notation that will be used in Section 5.4, we define the kernels corresponding to Fully Connected layers, Convolutional Layers and Recurrent Layers. In the context of the above notation for DKM, we introduce the notion of Parametric Kernel, which will allow us to consider recurrent networks in this framework. Parametric Kernels are kernels that simply accept an outside parameter that changes the space of representable functions, be leaves fixed the space on which the functions operate.

Definition 1 (Parametric Kernel)**.** *We will call parametric kernel a function $K : U \times X \times X \to \mathcal{L}(Y, Y)$ where Y is an Hilbert Space such that for every $u \in U$ the resulting function $K_u(x, x') = K(u, x, x')$ is a kernel for an appropriate space of functions from X to Y. We will denote $\mathcal{H}_u$ the resulting RKHS.*

Notation. For fixed functions $f^l \in \mathcal{H}^l_u$, we will denote by $\tilde{f} := f^L \circ \ldots \circ f^1 : Z_0 \to Z_L$ the concatenation of all functions, by $z^0_i := x_i$ the points in the dataset, and by $z^{l+1}_i := f^{l+1}(z^l_i)$ their images at every step of the function composition.

We will denote by $w^l_i \in Z_l$ for $i = 1, \ldots, p_l$ fixed points that will form the basis of the space on which the considered functions f^l will lie: in other words, the function $f^l : Z_{l-1} \to Z_l$ will be expressed as $f^l_{\alpha^l, u^l}(\cdot) = \sum_{ij} \alpha^l_{ij} K^l_{u^l}(w^{l-1}_i, \cdot) e_j$ where $\alpha^l \in \mathbb{R}^{p_{l-1} d_l}$ is the vector of the coefficients.

Subscript indices can cause some confusion in the notation because we may use more indexes to refer to the same dimension of a tensor. To differentiate them a comma is inserted between the indexes. More formally we define $\mathcal{T}[(n^1_1, \ldots, n^1_{k_1}), (n^2_1, \ldots, n^2_{k_2}), \ldots, (n^r_1, \ldots, n^r_{k_r})]$ to be the space of tensors with r dimensions, each of which is indexed with k_i indexes of various running dimensions.

For example we may say $A_{ij,kr} \in \mathcal{T}[(n, n), (p, q)]$ to denote that A is a $n^2 \times pq$ matrix and its indexes have different ranges: $i \in \{1, \ldots, n\}$, $j \in \{1, \ldots, n\}$, $k \in \{1, \ldots, p\}$ and $r \in \{1, \ldots, q\}$. We will also identify $\mathcal{T}[\vec{n}^1]$ with $\mathcal{T}[\vec{n}^1, 1]$ as commonly done with vectors and single-column matrices.

In the next section we will reason about the loss landscape of a fixed wide DKM, so that we will assume, unless otherwise specified, that the function spaces $\mathcal{H}^l$ parameterized by u, the dataset $\mathcal{D}$, the functional basis space w^l_i and the loss function $\mathcal{L}$ are fixed, while we will treat the function coefficients α^l_{ij} and the kernel parameters u^l as variables.

To ease notation we will not specify the dependency on α's or on u's of the quantities $X^l_{k\omega} = (z^l_k, e_\omega) \in \mathcal{T}[(N, d_l)]$ which represent basically the coordinates in the space Z_l of the points after

passing through the l-th layer and of $H^l_{ij,k\omega} = (K^l_{u^l}(w^{l-1}_i, z^{l-1}_k)e_j, e_\omega) \in \mathcal{T}[(p_{l-1}, d_l), (N, d_l)]$ which represents the product matrix of the embedded points z^{l-1}_k with respect to the chosen base w^{l-1}_i. Notice that it holds $X^l_{k\omega} = \sum_{ij} \alpha^{l-1}_{ij} H^{l-1}_{ij,k\omega}$.

We will also denote the loss calculated on the output points as $\mathcal{L}_\mathcal{D}(X^L) = \mathcal{L}((x_k, y_k, z^L_k)_k)$ where the z^L are computed accordingly to the above definitions, and with the symbol $A_{I,J}$ we will denote the minor obtained from matrix A by using only the rows $i \in I$ and the columns $j \in J$, where we will use uppercase letters to denote sets.

Finally, we define the cost function in its parametric view $C_\alpha : \mathbb{R}^{p_0 d_1} \times \ldots \times \mathbb{R}^{p_{L-1} d_L} \to \mathbb{R}$ as $C_\alpha(\alpha^1, \ldots, \alpha^L, u^1, \ldots, u^L) = C(\tilde{f}_{\vec{\alpha}, \vec{u}})$ where $\tilde{f}_{\vec{\alpha}, \vec{u}} = f^L_{\alpha^L, u^L} \circ \ldots \circ f^1_{\alpha^1, u^1}$. We will omit some alphas from the parameters of C_α if they are fixed in the current context and we will denote by $\vec{\alpha}$ the full vector of $\alpha^1, \ldots, \alpha^L$ and by $\vec{u}$ the vector of kernel parameters.

5.3.1 Neural Network Kernels

The connection with Artificial Neural Networks (ANN) arises when we consider each layer of an ANN as a linear combination of some space transformation over the previous layers, where the space transformation is induced by the broadcasted non-linear activation function. In this Section, we introduce the DKM formulation of the main neural layer types, namely fully connected, convolutional and recurrent.

Definition 2 (Fully-connected Kernel)**.** *Given an activation function $\sigma : \mathbb{R} \to \mathbb{R}$ the Kernel to consider for an FC layer from $\mathbb{R}^s \to \mathbb{R}^t$ is the following*

$$K(x, x')y := \left(\sum_{i=1}^{s} \sigma(x_i)\sigma(x'_i) \right) y \tag{5.1}$$

where $y \in \mathbb{R}^t$.

Remark 1. *In considering the above kernel it can be easily shown that the functions that can be expressed in the related RKHS are the same that can be expressed in a layer of a FCN. To this aim consider $c, r \in \mathbb{R}$ such that $\sigma(c) \neq \sigma(r)$ and define $\gamma^i \in \mathbb{R}^s$ by $\gamma^i_j = \delta_{ij} r + (1 - \delta_{ij})c$ with the usual Kronecker delta.*

Considering $K(\gamma^i, x)e_j = \sigma(r)\sigma(x_k)e_j + \sigma(c) \sum_{k \neq i} \sigma(x_k)e_j$ one can see how these are another base for the standard base functions of neural networks. This is more evident when $\sigma(c) = 0$, so that now $K(\gamma^i, x)e_j = \sigma(r)\sigma(x_i)e_j$ is a direct multiple of a standard base function.

Definition 3 (Convolutional Kernel)**.** *The Kernel for a Convolutional layer applied on inputs $x, x' \in \mathbb{R}^{whc}$ where w is the width, h is the height and c is the number of channels being considered is the following*

$$K(x, x')e_{\alpha\beta\gamma} = \left(\sum_{i=0}^{w} \sum_{j=0}^{h} \sigma(x_{ij\gamma})\sigma(x'_{(\alpha-i)(\beta-j)\gamma}) \right) e_{\alpha\beta\gamma}$$

where a term of the sum is null if any of the inputs evaluated at that index are not defined, and $\alpha \in \{-w, \ldots, w\}$, $\beta \in \{-h, \ldots, h\}$ and $\gamma \in \{1, \ldots c\}$.

Notice that the kernel in Definition 3 implements a variation of a single-strided convolution

with padding between inputs of the same size. To allow for biases terms and restructuring of the output of such kernel one can apply the following remark.

Remark 2. *Since for both the above two kernels we have that $\forall i\ K(x,x')e_i$ is a scalar multiple of e_i, one can also select only some output dimensions, for example to adapt to the description of the exact type of convolution used in real networks.*

Their inputs can also be augmented to easily allow for bias terms in what is commonly known as the bias trick without changing the theory in this Chapter. For expository reasons we stick to these simpler examples.

To include recurrent kernels we will suppose that $X = (\mathbb{R}^q)^T$, i.e. that every element of X is infact a sequence of T points in $\mathbb{R}^q$, and we will write $x_{i;t} \in \mathbb{R}^q$ where $i \in \{1,\dots,N\}$ and $t \in \{1,\dots,T\}$.

Definition 4 (Recurrent Kernel)**.** *Given $\sigma : \mathbb{R} \to \mathbb{R}$ the activation function, we want to find a kernel version of the typical RNN recurrence equation $h_{t+1} = \sigma(Wx_t + Uh_t)$. To do so we will first apply a linear FCN kernel to obtain the $z_{;t} = Wx_{;t}$ part, and after that we will use a parametric kernel K_u, where u is a $q \times q$ matrix and define*

$$\begin{aligned} K_u(z,z')e_{i;t} &= (\Phi_u(z)_{;t} \cdot \Phi_u(z')_{;t})e_{i;t} \\ \Phi_u(z)_{;t} &= \sigma(z_{;t} + u\sigma(z_{;t-1} + u \dots \sigma(z_{;2} + u\sigma(z_{;1}))\dots)) \end{aligned}$$

Notice that each element of the output sequence only depends on a finite number of elements of the input sequence, as this will be used in a later proof to rely on the theory of analytical functions on finite-dimensional spaces.

5.4 Loss landscape of Wide Deep Kernel Methods

In this section we prove our main theorem about the disappearance of basins in wide networks. We begin with a motivating lemma, that will show the need to study the rank of kernel matrices at different layers.

Lemma 4. *Suppose that $p_{L-1} \geq N$ and that the last layer kernel matrix $H^L \in \mathcal{T}[(p_{L-1}, d_l), (N, d_l)]$ is of full rank at the point $\vec{\alpha}$. If in addition $\nabla C_\alpha(\vec{\alpha}) = 0$, i.e. $\vec{\alpha}$ is a stationary point of the parameter loss, then $\nabla \mathcal{L}_\mathcal{D}(X^L) = 0$, i.e. the outputs of the current network are a stationary point for the point loss.*

Proof. Remembering that $X^L_{k\omega} = \sum_{ij} \alpha^L_{ij} H^L_{ij,k\omega}$, via a simple computation we obtain

$$0 = \frac{\partial C_\alpha}{\partial \alpha^L_{ij}}(\vec{\alpha}) = \sum_{k\omega} \frac{\partial \mathcal{L}_\mathcal{D}}{\partial X^L_{k\omega}}(X^L) \cdot \frac{\partial X^L_{k\omega}}{\partial \alpha^L_{ij}} = \sum_{k\omega} \nabla \mathcal{L}_\mathcal{D}(X^L)_{k\omega} H^L_{ij,k\omega} = \left(H^L\ \nabla \mathcal{L}_\mathcal{D}(X^L)\right)_{ij} \quad (5.2)$$

and since H^L is of full-rank and with more rows than columns, it holds $\operatorname{Ker} H^L = 0$ which implies that $\nabla \mathcal{L}_\mathcal{D}(X^L) = 0$ as desired. □

In the case where the point loss function $\mathcal{L}_\mathcal{D}$ is convex the above lemma says that a stationary point of the optimized parameter loss is actually a global minima for the empiric loss of the

current problem, provided that the network is wide enough. The requirement that the last layer kernel matrix be full rank motivates the study of full-rank points $\vec{\alpha}$ in the optimization landscape.

The notation for the next part of the section will focus on a single RKHS $\mathcal{H}$ of functions from $X = \mathbb{R}^s$ to $Y = \mathbb{R}^t$, and K is the associated parametric kernel of $\mathcal{H}$. For the case of parametric kernels we will assume u to be fixed, i.e. that the stated properties hold $\forall u \in U$, but we will state them without u to ease the notation. We will sometimes assume that $K : X \times X \to \mathcal{L}(Y, Y)$ is an analytical function, in the sense that for any fixed $y \in Y$ the function $K(\cdot, \cdot)y : X \times X \to Y$ is a real analytic function, in order to use the following lemma:

Lemma 5 (Dichotomy for zero sets of real analytic functions, [Mit15])**.** *Let $f : \mathbb{R}^m \to \mathbb{R}$ a real analytic function. If f is not identically zero, then its zero set $\mathcal{Z}_f = \{z \in \mathbb{R}^m \mid f(z) = 0\}$ is of zero lebesgue measure.*

We start by recalling the following definition, given by [MP05] about the linear independence of points in a kernel space.

Definition 5 (Linear independence of points in a Kernel Space, [MP05])**.** *For distinct points $x_1, \ldots, x_m \in X$ we say that the linear functionals $L_{x_i} : \mathcal{H} \to \mathbb{R}$ defined by $L_{x_i} f = f(x_i)$ are linearly indepent if and only if there does not exists $c_1, \ldots, c_m \in Y$ not all zero such that we have*

$$\forall f \in \mathcal{H} \quad \sum_{i=1}^{m} (c_j, f(x_j)) = 0. \tag{5.3}$$

We will say that the points $x_1, \ldots, x_m$ are linearly dependent (linearly independent) iff the corresponding linear functionals $L_{x_1}, \ldots, L_{x_m}$ are linearly dependent (linearly independent).

The importance of finding independent points is made explicit by our next lemma (notice that here the matrix H does not represent the Hessian).

Lemma 6 (Characterization of dependent points)**.** *The points $x_1, \ldots, x_m$ are dependent if and only if the matrix $H_{ij,rs} = (K(x_i, x_r)e_j, e_s)$ is singular.*

Proof sketch By using the reproducing property we rewrite Equation (5.3) as

$$\forall f \in \mathcal{H} \quad \left\langle \sum_{i=1}^{m} K(x_j, \cdot)c_j, f \right\rangle = 0$$

and we exploit the fact that the orthogonality relation is equivalently tested only on the subspace $\text{Span}\{K(x_i, \cdot)e_j \mid i = 1, \ldots, m; j = 1, \ldots, t\}$. A detailed proof can be found in Appendix A. □

Given the above relation of dependent points with kernel matrix singularity, we will need to find at least a set of N independent points to chose them as the base in which functions will be represented. The following two lemmas provide this guarantee in the case of positive definite kernels and the previously introduced neural network kernels.

Lemma 7 (Independent points in a positive definite kernel)**.** *Let K be a positive definite Kernel and $x_1, \ldots, x_m$ be points in X. Then these points are all independent provided that they are distinct.*

Proof sketch Using Lemma 6 we need to show that the matrix $H_{ij,rs} = (K(x_i, x_r)e_j, e_s)$ has null kernel. K being positive definite implies that H is also a positive definite matrix, thus having a trivial kernel. A detailed proof can be found in Appendix A □

We will need this preliminar lemma for the proof of independence of points in the case of fully connected kernels.

Lemma 8. *Let $J_{ij} = 1$ be the ones square matrix $n \times n$ and $I_{ij} = \delta_{ij}$ be the $n \times n$ identity matrix. Then $H = BJ + (A - B)I$ has determinant* $\det H = (A - B)^{n-1}((n - 1)B + A)$.

Proof. The eigenvalues of J are n with multiplicity 1 and 0 with multiplicity $n - 1$. Moreover, $JI = IJ$ and thus the eigenvalues of H are $(n - 1)B + A$ with multiplicity 1 and $A - B$ with multiplicity $n - 1$. □

Lemma 9 (Independent points in FCN kernels)**.** *The neural network space for $X = \mathbb{R}^s$ and $Y = \mathbb{R}^t$ has s independent points if $\sigma(r) \neq -(s - 1)\sigma(c)$.*

Proof. Consider points γ^i already defined in Remark 1. To prove independency of those points let us define $H_{ij,k\omega} = (K(\gamma^i, \gamma^k)e_j, e_\omega)$ and prove that it is invertible. A brief calculation yields

$$H_{ij,k\omega} = \delta_{j\omega}(\delta_{ik}A + (1 - \delta_{ik})B)$$

where $A = (s - 1)\sigma(c)^2 + \sigma(r)^2$ and $B = (s - 2)\sigma(c)^2 + 2\sigma(c)\sigma(r)$. Since $\sigma(c) \neq \sigma(r)$ and $\sigma(r) \neq -(s - 1)\sigma(c)$ we have $A \neq B$ and therefore we have that matrix H is invertible because of the previous Lemma 8. □

Lemma 9 also holds for CNN and RNN kernels. For the proof in the latter two cases the reader can refer to Appendix A.

We are now ready to state and prove the main lemma, which guarantees that in wide networks for almost any choice of coefficient at a certain layer, we can assume that the layer matrix is full-rank.

Lemma 10 (Full-rank matrix with chosen base)**.** *Let $K : X \times X \to \mathcal{L}(Y, Y)$ be an analytical Kernel function, $(w_1, \ldots, w_p) \in X^p$ be a tuple of points with $p \geq N$ such that at least a subset of N of them consists of independent points relative to K. Let $H \in \mathcal{T}[(N, s), h]$ be a fixed full-rank matrix with $h \geq Ns$ and $A \in \mathbb{R}^h$ any vector, and define the points $x_k \in X$ by $x_k(A) = \sum_\omega (x_k, e_\omega)e_\omega = \sum_\omega (HA)_{k\omega}e_\omega$.*

Then there exist a set of zero measure $\Theta \subseteq \mathbb{R}^h$ such that given $A \notin \Theta$ we have

$$\left(\exists \delta_1, \ldots, \delta_m \in Y \text{ such that } \forall ij \quad \sum_{k=1}^{N} (K(w_i, x_k(A))e_j, \delta_k) = 0\right) \implies \forall k \quad \delta_k = 0$$

or equivalently the matrix $R_{ij,k\omega}(A) = (K(w_i, x_k(A))e_j, e_\omega) \in \mathcal{T}[(p, t), (N, t)]$ is of full-rank.

Before proving the lemma, let us rephrase its statement to make it more clear: we are saying that whenever some independent basis points w are chosen, and the previous layer kernel matrix is full rank, then the current layer kernel matrix is almost always of full rank.

Proof. We notice first that $A \mapsto R(A) \in \mathcal{T}[(p,t),(N,t)]$ is a real analytic function of A since K is analytic. For $R(A)$ to be full-rank we need that it has at least one $Nt \times Nt$ minor determinant that is not zero. To this aim we consider the sum over the determinant of all $Nt \times Nt$ minors:

$$\varphi(A) = \sum_{|I|=Nt} \sum_{|J|=Nt} (\det R(A)_{I,J})^2.$$

Since the determinant can be expressed as a polinomial in the matrix entries, we obtain that $\varphi(A)$ is a real analytic function of A and $\varphi(A) = 0$ iff all $Nt \times Nt$ minors of the matrix have vanishing determinant, which happens iff $R(A)$ is not of full-rank.

By Lemma 5 we know that either the zet $\mathcal{Z}_\varphi = \left\{A \in \mathbb{R}^h \mid \varphi(A) = 0\right\}$ is of zero measure (which would easily imply the statement of the lemma) or otherwise it must be that the function φ is constantly zero.

In the latter case, since the matrix H is full-rank we obtain that the function $\Psi : \mathbb{R}^h \to X^N$ defined by $\Psi(A) = (x_1(A), \dots, x_N(A))$ is surjective. Let now $w_{i_1}, \dots w_{i_N}$ be N independent points from the w's. Then it must exist $A^* \in \mathbb{R}^h$ such that $\Psi(A^*) = (w_{i_1}, \dots, w_{i_N})$ because of surjectivity.

Define $I = \{i_1, \dots, i_N\}$, $J = \{1, \dots, t\}$, $F = \{1, \dots, N\}$, $G = \{1, \dots, t\}$ and consider the matrix $R^* := R(A^*)$: since $\varphi(A^*) = 0$ we know that all its $Nt \times Nt$ minors have vanishing determinant, but at the same time $\det R^*_{IJ,FG} = \det(K(w_i, w_f)e_j, e_g)_{ij,fg} \neq 0$ because of Lemma 6, obtaining a contradiction. □

Using the above lemma we can prove than in a neural network with analytic activation function, almost every point has a full-rank kernel matrix, provided that every intermediate layer of the network is large enough:

Theorem 2. *Suppose a DKM has $\forall 1 \leq l < L \quad d_l \geq N$, all Kernels K^l are analytic functions and the chosen basis w are such that $p_l \geq N$ for $l = 1, \dots, L-1$ and each layer base has at least N independent points, then for almost every choice (in the lebesgue measure sense) of the coefficients $\vec{\alpha}$, the last layer kernel matrix H^L is full-rank. Moreover if such point is a stationary point for the parameter loss C_α, then the outputs of the current network are a stationary point for the point loss.*

Proof. The proof proceeds by induction on layers, starting from the first one. For the first layer it is enough to use Lemma 10 with $h = Ns$, $H = \mathrm{Id}_{Ns}$ and to recognize that in this case $x_k = \sum_\omega A_{k\omega} e_\omega$ are exactly the initial inputs coordinates.

For the inductive step assume that $H^l \in \mathcal{T}[(p_l, d_l), (N, d_l)]$ is of full rank. Using Lemma 10 with $H = (H^l)^\top$ and $A = \alpha^l$ we obtain that for almost all choices of α^l, the kernel matrix $H^{l+1} = R$ is of full rank.

A subtlety in the above steps is that we implicitly make the choices of the "bad" zero measure sets Θ^l sequentially (i.e. depending on previous choices). To be able to say that there exist a single $\Theta \subseteq \prod_{l=1}^L \mathbb{R}^{p_{l-1} d_l}$ of zero measure in the whole coefficient space, we need to consider a

parameter $(\alpha^1, \dots, \alpha^L)$ "spoiled" whenever just one of those α^i is spoiled, i.e.

$$\Theta = \bigcup_{l=1}^{L} \mathbb{R}^{p_0 d_1} \times \dots \times \mathbb{R}^{p_{l-2} d_{l-1}} \times \Theta^l \times \mathbb{R}^{p_l d_{l+1}} \times \dots \times \mathbb{R}^{p_{L-1} d_L}$$

Fortunately Θ is of zero measure when considering the lebesgue measure on the product space. □

Remark 3 (GAN optimization landscape)**.** *Let us suppose now that the loss function to be optimized can be written in the form $\mathcal{L}((x_k, y_k, f(x_k), g(f(x_k))))$ as it is the case in GAN optimization [Goo+14], for both f and g two functions derived by some DKM.*

The reader can see that the above proofs still hold in this setting, even though one must be careful in performing the inductive proof steps first for the layers of f and then for the layers of g, due to the above mentioned subtlety.

In the context of GANs the optimization problem is not that of minimization, but rather is a more complex minimax optimization, but the fact that such properties of the loss landscape still hold is another point in favour of the currently proprosed method.

5.4.1 Weak-globality of the loss landscape

A sensible question that may come to mind at this point is if these results are actually useful or not. Ultimately we are infact interested in first order stationary point of the loss, which are usually "rare" and thus may all be inside the degenerate set Θ. Leveraging on an already known result of [LDS18] we show how such almost-everywhere characterization are indeed useful to derive properties about the optimization landscapes.

Definition 6 (PT property, [LDS18])**.** *A function $g : \mathbb{R}^k \to \mathbb{R}$ is said to have the PT property if, from almost every point x_0 there is a descending path towards the global minimum.*

Theorem 3 (DKM landscape satisfies the PT property)**.** *Suppose all hypothesis of Theorem 2 and that the loss function $\mathcal{L}$ is convex in $(f(x_k))_k \in Z_L^N$. Then the function C_α has the PT property.*

Proof. Consider a point $\vec{\alpha}$ such that the DKM has all the layer kernel matrices full-rank, a condition that we already proved holds almost everywhere. For this point we have $X^L = \alpha^L H^L$ and H^L is full-rank.

Let $X(t) \in Z_L^N$ for $t \in [0, 1]$ be a path from X^L to the global minimum G of $\mathcal{L}$, which exists because of convexity of $\mathcal{L}$. Then the path $\alpha^L(t) = (1-t)\alpha^L + tV$ where V is such that $G = VH^L$ (which exists because H^L is of full-rank) is a descending path to the global minimum for C_α:

$$C_\alpha(\alpha^L(t)) = C_\alpha((1-t)\alpha^L + tV) \leq (1-t)C_\alpha(\alpha^L) + tC_\alpha(V)$$

because of Jensen inequality. □

We recall some definitions and results from [LDS18], that show how the previosuly obtained properties imply easyness of the optimization problem.

Definition 7 (Set-wise strict local minimum, [LDS18]). *We say that a compact subset $X \subseteq \mathbb{R}^k$ is a strict local minimum of $f : \mathbb{R}^k \to \mathbb{R}$ in the sense of sets if there exists $\varepsilon > 0$ such that for all $x \in X$ and $y \in \mathbb{R}^k \setminus X$ satisfying $\|x - y\|_2 \leq \varepsilon$, it holds that $f(x) < f(y)$.*

Definition 8 (Weakly global function, [LDS18]). *We say that $f : \mathbb{R}^k \to \mathbb{R}$ is a weakly global function if it is continuous and every set-wise strict local minimum contains a (pointwise) global minimum of f.*

The theorem that lets us derive the implied property of weakly global function for all analytic kernels is the following one, which as we have already seen directly applies to the case of Deep Kernel Methods because of Theorem 3.

Theorem 4 (PT property implies weakly global, [LDS18]). *Let $f : \mathbb{R}^k \to \mathbb{R}$ be a function with the PT property. Then f is weakly global.*

5.5 Conclusions

There are basically two possible problems in the optimization of a smooth non-convex objective: suboptimal local minima and non-strict saddle points. We have proven that suboptimal local minima do not exist in the sense stated by the weakly global property (Definition 8) for a vaste range of kernels and wide networks, generalizing a result of [LDS18]. The proof was carried out in an architecture-independent way and by relying only on general properties of the Kernels.

We have shown that our framework can be useful in proving properties that are only dependent on the basic form of the optimization problem for neural networks, and not upon the specific choice of architectures. This could help in a more abstract study of neural networks, and could provide reasons for the success or failure of an architecture for a certain task by analyzing the associated Kernel.

We have shown how a common approach to the various architectures of neural networks is archievable, integrating also Recurrent Networks in the study whose optimization properties have remained almost unstudied to date. Our framework also shows that the optimization of Deep Kernel Methods has commonalities to that of Neural Networks, and we hope that this link will be exploited by both literatures.

Chapter 6

Testing Polyak–Łojasiewicz Theory on Wide Deep Neural Networks

6.1 Introduction

A staggering aspect about neural networks is that they are seemingly able to overfit the training sample and yet generalize to unseen data. Similar behaviour has been actually observed in other learning paradigms in overparameterized settings, such as with linear regression [BLLT20] and kernel methods [BMM18; TB20], but neural networks are certainly the model characterized by the most surprising predictive performance on real-world data.

Several recent works [SGGS17; JGH18; Du+19; Yan+20] have attempted to explain the theoretical underpinnings of why neural networks learn and generalize. Inspired by the relation of infinitely wide neural networks with Neural Tangent Kernel [JGH18; Lee+19], much of this research has focused on developing a theory for *very wide networks*, i.e. where the number of parameters $m = \Theta(n^\alpha)$ is polynomially bigger than the number of examples n [DZPS18; ALS19; LDS18; OS20] with $\alpha > 1$, a condition which is not usually satisfied in real-world models[1]. While these theoretical results are being obtained under increasingly milder (and hence more realistic) overparametrization assumptions, they are typically expressed only in terms of asymptotic rates, and it is difficult to determine whether these results help in explaining the success of actually deployed models.

Another line of research focuses on strongly experimental work [Lee+20], which is exclusively tailored to obtain empirical suggestions for practitioners. Our approach is instead focused on measuring crucial quantities during training of realistic networks with the aim to inform theoretical research and to expose points of discrepancy.

The Problem. Neural networks are function approximators usually trained via Empirical Risk Minimization, but they cannot be understood via classical optimization theory because they are extremely non-convex [LZB22]. In recent years many researchers have considered

[1] Consider, for instance, VGG19 which has 144M parameters distributed on 19 layers and was trained on 1.3M images: already a quadratic polynomial would require an order of 1 trillion parameters per layer for VGG, which is clearly unattainable for current practice. On the other hand VGG19 is mildly overparameterized, i.e. the number of parameters per layer is slightly greater than the number of used images.

alternatives to convexity, concentrating on variants of the Polyak–Łojasiewicz (PL) condition [Pol63] like PL* functions by Liu, Zhu, and Belkin [LZB20] and the Proxy-PL condition by Frei and Gu [FG21].

For the PL condition many linear convergence results for first-order optimization methods are known [KNS16; CR17; GGGM21], and the PL coefficient (akin to strong-convexity coefficient) can be lower bounded in a region nearby the network random initialization [LZB20], thus providing convergence guarantees if the network weights remain in the vicinity of initial weights, which is the case for very wide networks. While being a very promising theory, it is currently not clear if such theorems do hold for smaller networks like those deployed in practice.

Contributions. Our aim is to precisely analyze which points of the current theories fail to hold empirically when applied to mildly overparameterized networks, i.e. networks in which the number of weights grows linearly with respect to the number of examples $m \simeq cn$. This work is, to our knowledge, the first one to perform a detailed empirical analysis of abstract theories tailored at exposing the uneffectiveness of existing theories in certain areas.

We perform quantitative measures of key local quantities related to conditioning, convergence and optimization in the PL function theory to check the impact of reducing the networks width from polynomial in the number of examples to linear. We show how such measures can characterize the training progress of real neural models on machine vision tasks providing, in Section 6.3, an empirical analysis that compares the prediction of the theory with the observed behavior of trained models, which enables us to spot phenomena that aren't currently fully explained.

Differences with related works. Our work builds on theories in the area of PL functions and their applicability to neural networks; in particular we consider a general theory of PL functions convergence by Liu, Zhu, and Belkin [LZB20], stability bounds by Charles and Papailiopoulos [CP18] for generalization, and the work by Agarwal, Awasthi, and Kale [AAK20] for the analysis of conditioning.

Known applications of the theory of Polyak–Łojasiewicz functions to neural networks are only concerned with the theoretical side. For example, Liu, Zhu, and Belkin [LZB20] deals with theoretical convergence issues on wide networks and do not consider the interplay between conditioning and network depth; we focus instead on empirical convergence monitoring and generalization on mildly overparameterized networks, operating far from the kernel regime, and on realistic models. Moreover we consider quantitative issues like convergence speed of real networks which are not addressed by Liu, Zhu, and Belkin [LZB20].

For what concerns the empirical analysis on wide neural networks, Lee et al. [Lee+20] performed a large-scale experiment to study the performance of finite-width networks compared with their infinite limits, as predicted by the NTK theory [JGH18]. Our approach differs in that we focus on adherence of the empirical behavior to the presented theory at the level of single optimization steps, while [Lee+20] restricts to comparing the final outcomes of the optimization process for various finite- and infinite-width models, and thus mainly serves to guide empirical practitioners.

The exposed bound in Equation (4.12) gives us a way to quantitatively measure stability on real neural networks, by locally estimating their Lipschitz coefficient G and their PL constant μ as

detailed in Section 6.2, to produce Figures 6.11 and 6.12 that we will comment in Section 6.3.

Let us briefly comment on the role of the PL constant μ and of small-deviations of the model with respect to its parameters and its inputs as captured by the Lipschitz coefficient G and its smoothness constant L: as we can see in the definition of γ in Lemma 1 a smaller smoothness constant and a higher PL coefficient are of benefit to fast convergence.

Concerning generalization we notice similarly the importance of a large μ and a small Lipschitz coefficient in the quantity $2G^2/\mu$, which appears twice in Equation (4.12).

We observe moreover that the first term in Equation (4.12) depends on the amount of optimization performed by the algorithm, and vanishes for a perfect fitting of the training data, which suggests a connection between overfitting and generalization for PL models.

6.2 Measure of PL Coefficients.

One of our contributions to this work is to devise an empirical way to most accurately measure the local PL coefficients. Indeed, the empirical measure of the global PL coefficient (μ in Equation (4.1)) of a function is hard because one has to guarantee that

$$\forall x \in X \quad \mu \leq \frac{\|\nabla f(x)\|^2}{2(f(x) - f^*)}, \tag{6.1}$$

which essentially requires to know the function f globally.

What we did was instead to measure the local PL coefficients $\mu : X \to \mathbb{R}$ defined by

$$\mu(x) := \frac{\|\nabla f(x)\|^2}{2(f(x) - f^*)}, \tag{6.2}$$

where $f^* = 0$ in our case because we know we are in the setting of overparameterized models. This may at first seem like a rough simplification which could lead to vastly incorrect results; we thus motivate more in depth this choice.

Indeed, under appropriate smoothness of f, the function $\mu : X \to \mathbb{R}$ is itself smooth, and thus it is at least locally stable. To quantify how much stable this quantity is in a local neighbourhood of a point, one can leverage the fact that, in our case, the function to minimize is a sum of independent terms, i.e.

$$\mu(x) = \frac{\mathbb{E}_{z\sim\mathcal{D}}\Big[\|\nabla_x g(z;x)\|^2\Big]}{2\mathbb{E}_{z\sim\mathcal{D}}[g(z;x)]} \tag{6.3}$$

where $f(x) = \sum_{i=1,\dots,n} g(z_i;x)$. Moreover, it is not hard to devise a convergence theorem akin to Lemma 1 using local coefficients $\mu(x)$, which effectively shows their usefulness in predicting the behavior of the optimization process.

Lemma 11. *Consider a function $f : X \to \mathbb{R}$ which is L-smooth. Then the gradient method with a step-size of $1/L$*

$$x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k) \tag{6.4}$$

satifies the equation

$$f(x_k) - f^* \leq (f(x_0) - f^*) \prod_{i=0}^{k-1} \left(1 - \frac{\mu(x_i)}{L}\right) \tag{6.5}$$

which gives a global convergence rate.

Proof. By using the update rule in the L-smoothness condition we get

$$f(x_{k+1}) - f(x_k) \leq -\frac{1}{2L}\|\nabla f(x_k)\|^2. \tag{6.6}$$

Now by using the local PL inequality we get

$$f(x_{k+1}) - f(x_k) \leq -\frac{\mu(x_k)}{L}(f(x_k) - f^*) \tag{6.7}$$

Rearranging and subtracting f^* from both sides gives $f(x_{k+1}) - f^* \leq \left(1 - \frac{\mu(x_k)}{L}\right)(f(x_k) - f^*)$, which can be applied recursively to obtain the stated result. □

6.3 Empirical Analysis

In this Section we describe and analyze the performed experiments to check how well real-world models can be described by the exposed theories. The reader should keep in mind the discussed theories on PL convergence, conditioning and generalization discussed in Chapter 4.

Experimental Setting. We train several mildly overparameterized Fully Connected Networks (FCNs) on random subsets of CIFAR10 [KH+09][2]. The networks are initialized with Gaussian Kaiming initialization [HZRS15] to preserve the variance of activations in the forward pass; activation functions are normalized according to Agarwal, Awasthi, and Kale [AAK20], i.e. having zero mean and unitary variance on standardized normal inputs. Input data is normalized to satisfy a scaled requirement of the unitary norm required by Agarwal, Awasthi, and Kale [AAK20] such that $\|x_i\| = \sqrt{m}$, where m is the width of the first layer.

Conditioning at Initialization. We consider FCN networks consisting of 30 layers of varying widths (1000, 2000, 5000), different activation functions (ReLU and Tanh), over multiple numbers of randomly extracted examples (100, 200, 500, 1000), either renormalizing[3] after application of each layer or not, and averaging on three random seeds. We measure conditioning of $G^{m;\theta}$ (Equation (4.9)) at initialization over all layers. The experiment has been run on a Tesla V100 PCIe 16GB GPU.

Figure 6.1 shows that conditioning at initialization effectively decreases exponentially even for finite-width networks; it is smaller for higher width-examples ratio, and continues to decrease

[2]Usage of a single dataset to validate the theory has been dictated by the heavy computational requirements of the experiments. Such a choice should nonetheless not impact the validity of the presented results, because the tested theories do not contain free parameters to be fitted on the dataset, and experiments are repeated for multiple random extractions of a subset of the data, thus leaving less chance for an overfit of the results to the dataset.

[3]i.e. rescaling each datapoint such that its norm is the square root of the number of neurons in the layer.

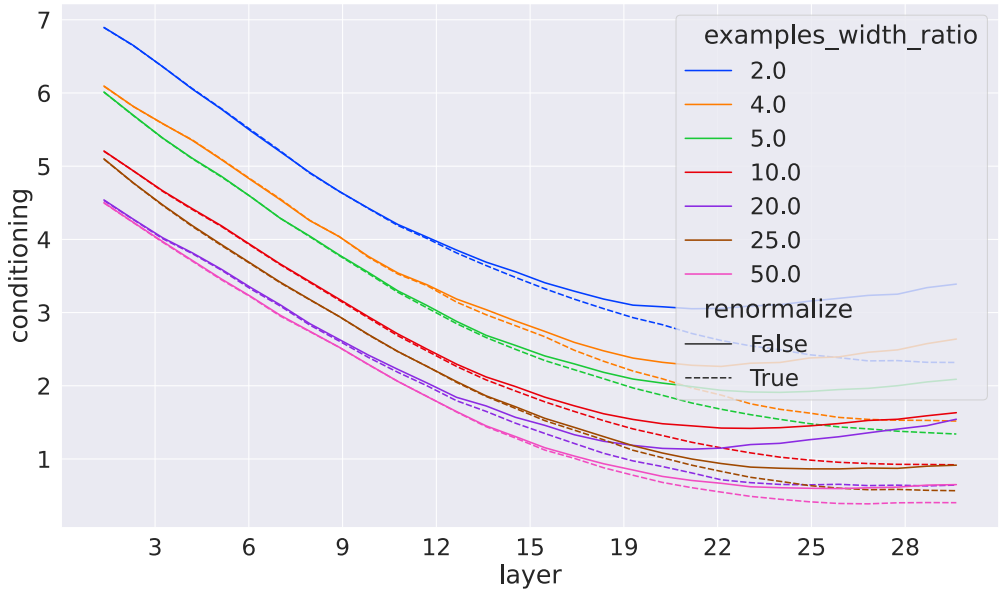


Figure 6.1: Mean log-conditioning of $G^{m;\theta}$ for ReLU FCNs at initialization; colors represent different examples-width-ratios of the tested networks. We can see the initial linear decrease, followed by a stagnation phase for the log-conditioning; in the case of non-renormalized networks, the conditioning later starts to increase again, possibly due to small-width effects.

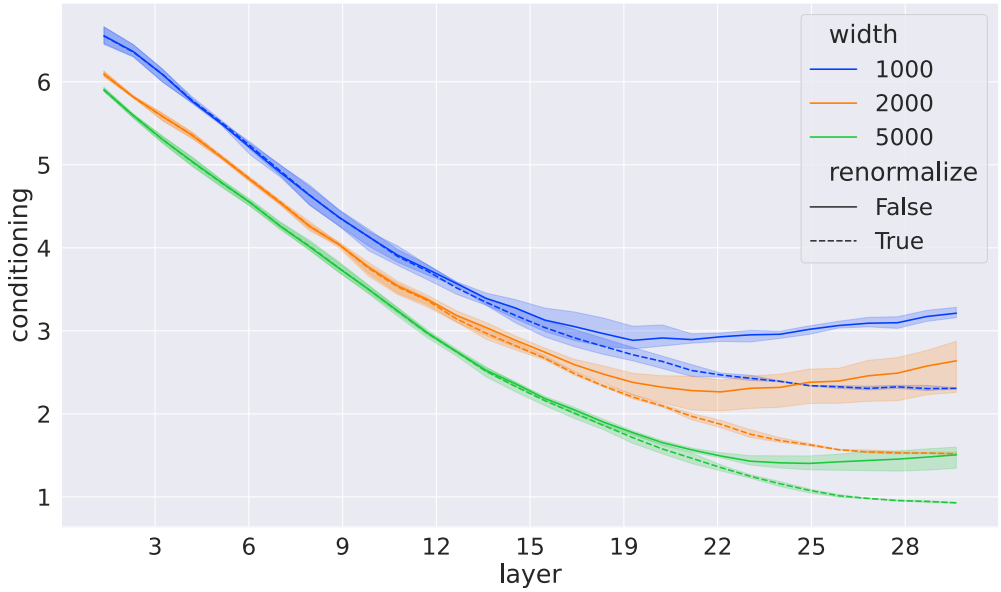


Figure 6.2: Mean log-conditioning of $G^{m;\theta}$ for 500 examples at varying widths (colored differently); shaded regions denote 95% confidence intervals. We can see that the wider the network, the lower conditioning is. Again we can notice the same behaviour concerning the difference between renormalized and non-renormalized networks.

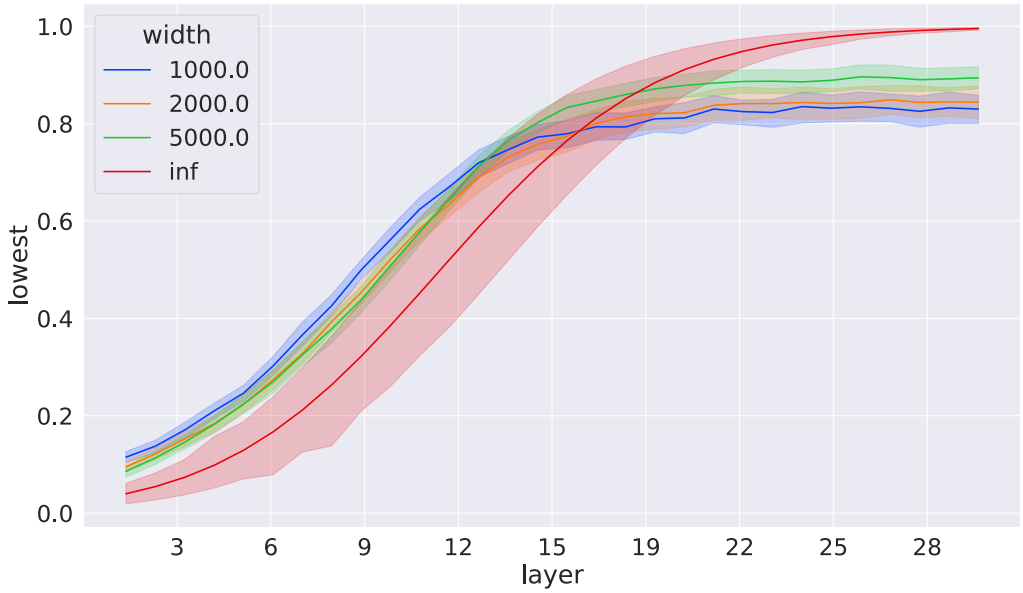


Figure 6.3: Lowest eigenvalue of $G^{m;\theta}$ for ReLU networks of different widths with renormalization rescaled with $\sqrt{m}$ according to Section 6.3. Infinite width network line has been obtained using Lemma 24. We can notice how the rescaled eigenvalues increase a lot in the intermediate layers, to stabilize at the end at values around one, which is however only reached in the case of infinite-width networks.

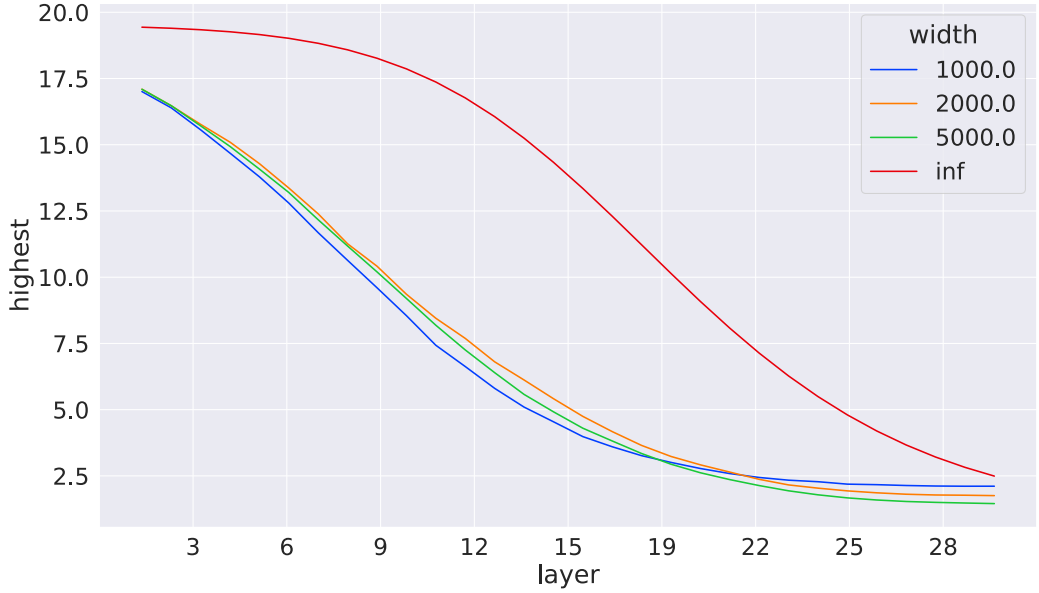


Figure 6.4: Rescaled highest eigenvalue for ReLU networks of different widths with renormalization. To make the graph more clear only the mean is reported. Again we can see how the highest eigenvalue effectively decreaes in the intermediate layers, and later stabilizes; it is interesting to notice the different qualitative curve behaviour in the case of infinite-width networks.

at each layer when the examples are renormalized; on the other hand, if the examples are not renormalized, we can find a layer after which conditioning starts to increase again, probably due to small-width effects in the sampling of gaussian weights that deviate from the distribution assumptions of Agarwal, Awasthi, and Kale [AAK20][4]. Moreover, larger widths allow to reach a smaller conditioning for the same sample size (Figure 6.2) and also allow for a latter departure between the normalizing and non-normalizing behavior.

In Figure 6.3 we can see that the lowest eigenvalue (with a rescaling by square root of width computed accordingly to Section 6.3) effectively increases as it passes through different layers, also at finite widths. However it is also clear that it produces a marked increase in the first layers and it stops at farther layers, with higher values for wider networks. Analogously we see an inverse trend for normalized highest eigenvalue in Figure 6.4. These observations highlight the need for a more thoughtful conditioning theory for finite-width networks so that useful hints can be obtained for an optimal tradeoff between computational resources and convergence speed with respect to network width and normalization layers.

Basing on the adherence between the two behaviours in the initial layers, we speculate that layer normalization strategies may be removed from those without significant losses in accuracy. We leave a verification of this claim to future works.

Training Speed and Conditioning. In the second experiment we consider networks of 6 and 9 FCN layers, of width 500, with different activation functions (ReLU, Tanh), and train them using full-batch gradient descent (without momentum) with mean-squared-error loss and cross-entropy loss[5] over 80 epochs, with various learning rates (0.0005, 0.001, 0.005, 0.01) and different number of randomly sampled examples (50, 100, 250, 500) over three seeds. To have meaningful results on networks that actually learn, the analysis has been conducted only on those configurations that have reached accuracy $> 12\%$ at the end of training. The experiment has been run on an A100 SXM4 40GB GPU.

We find that the upper bound on loss decrease given by Lemma 1 matches well the actual loss decreases at later epochs, while in initial epochs the estimate is too conservative (Figure 6.5).

Figure 6.7 and Figure 6.8 show how for networks trained with cross-entropy loss, conditioning worsens very slowly during training, while for higher learning rates it initially increases very rapidly, but then decreases steadily, ending up lower than the start. The reader can see how pessimistic are current estimates of eigenvalues based on Equation (4.11) during training in Figure 6.6, in which we can see how the bound degrades significantly, and even becomes vacuous for an high number of examples. We emphasize the importance for theoretical research to look at possible explanations for this behaviour, which could greatly simplify the study of neural networks optimization, as the main theoretical difficulties in providing convergence guarantees lie in the possibility that the lowest eigenvalue may tend to zero during training.

Practical Conditioning Proxy. Given our findings about conditioning at initialization (Figure 6.1) and the importance of conditioning for training speed, in principle we would like to

[4]According to the original paper renormalization shouldn't have any effect, but they effectively consider infinite-width networks implicitly in the definition of the dual activation function (see Lemma 23).

[5]We include cross-entropy since it is the most common loss for classification networks, even though it is not a PL function. To make it tractable we report the theory of weakly-PL functions in Appendix B.1.2.

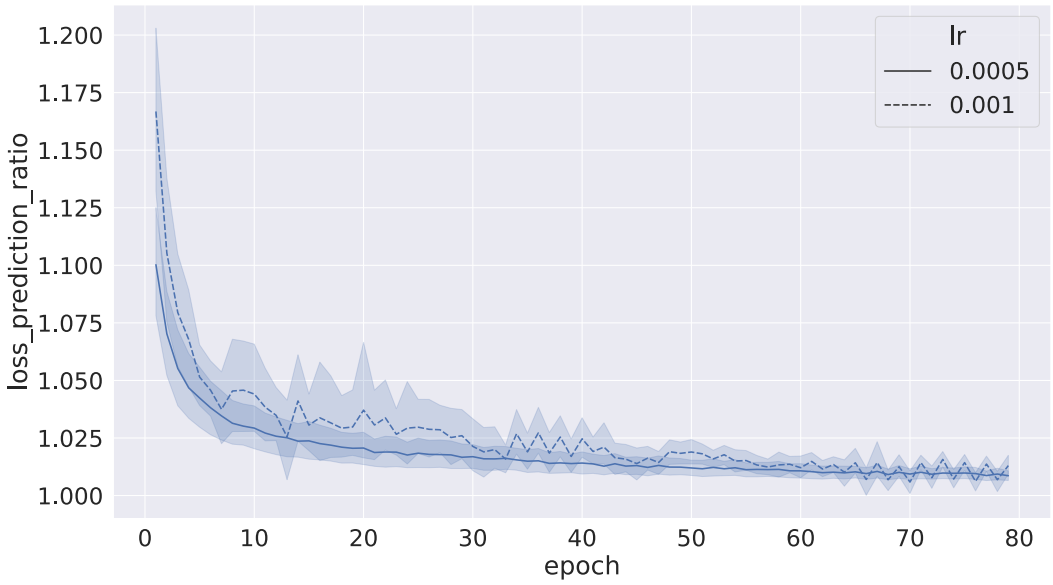


Figure 6.5: Ratio between predicted training loss (calculated according to the bound in Lemma 1 where the PL and smoothness coefficients are empirically calculated on the network at the previous epoch using the details in Section 6.2) and measured loss decrease at single epochs for 6-layers ReLU networks trained under MSE at different learning rates. We notice how the initial few epochs the measured loss decrease is bigger than the predicted one, while for the final epochs the prediction is almost perfect.

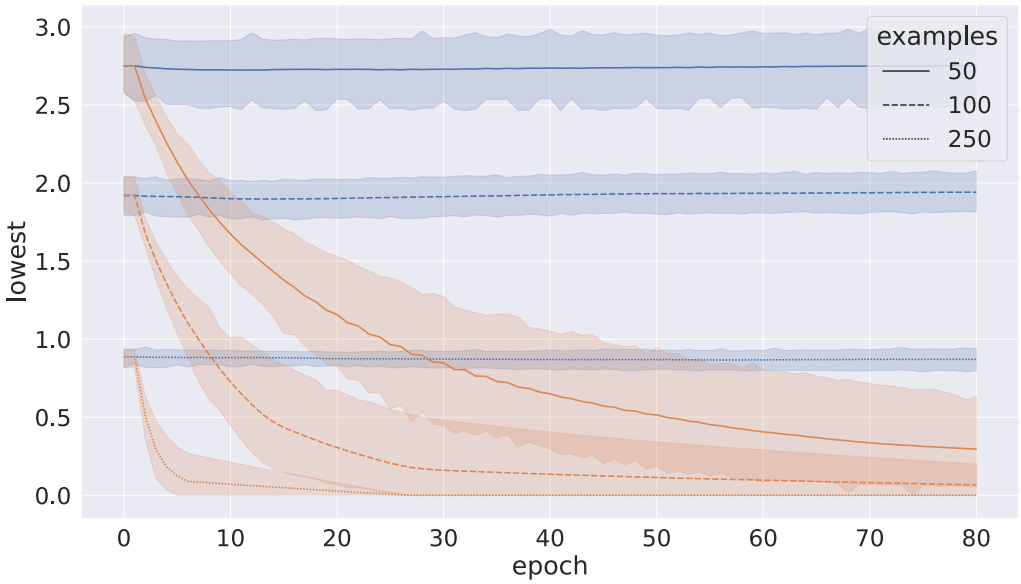


Figure 6.6: Progression of the lowest eigenvalues of $G^{m;\theta}$ for ReLU networks trained with MSE is in blue; lower bounds obtained using Equation (4.11) are in orange.

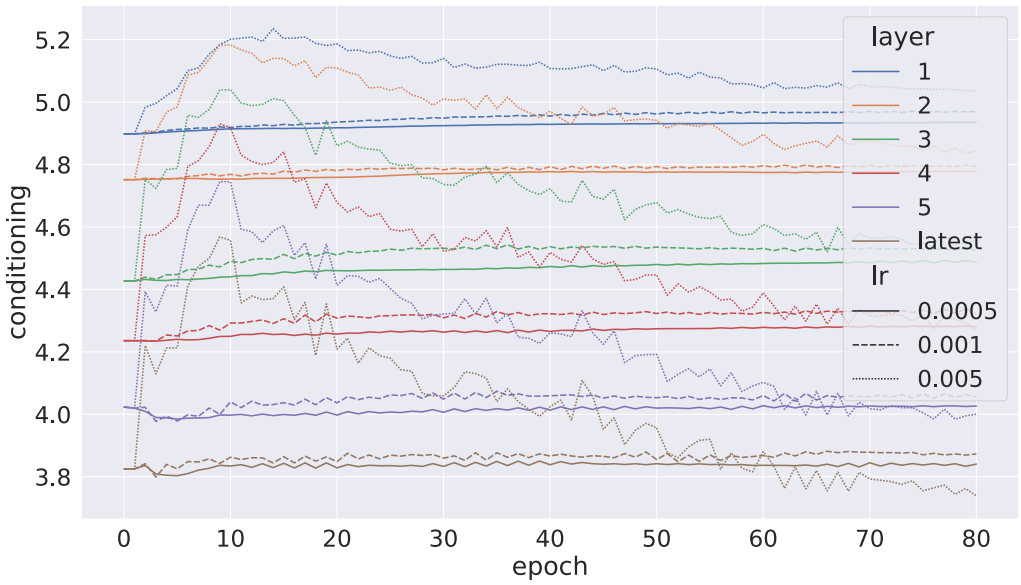


Figure 6.7: Progression of log-conditioning for ReLU networks trained under Categorical Cross-Entropy. We notice that, contrary to the expectations, for smaller learning rates the conditioning increases very slowly, while for higher learning rates it initially increases, then starts to gradually decrease, meaning that the problem becomes easier to solve the more the network is trained.

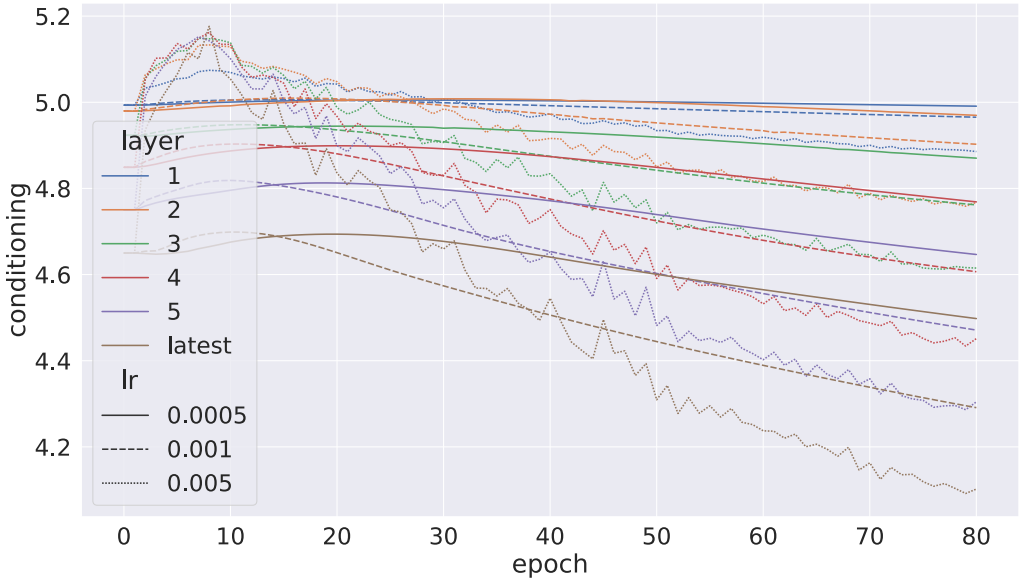


Figure 6.8: Progression of log-conditioning for Tanh networks trained under Categorical Cross-Entropy. Here we only notice the initial increase followed by a steady decrease.

avoid training a network that is too deep for its width, which has the drawback of raising the conditioning from some point onward. Measuring conditioning directly is inefficient, as it requires to solve the eigenvalue problem for very big square matrices, whose size is the product of the networks width and the number of examples, making it impractical.

As we have already observed in Section 4.4, we expect off-diagonal entries of $G^{m;\theta}$ to go to zero as the signal propagates throught the layers, hence we propose to measure this proxy information instead of measuring eigenvalues directly, as this can be used to bound the highest and lowest eigenvalues via Gershgorin Circle Theorems, greatly simplifying practical approximated measurements of conditioning[6].

In Figure 6.9 and Figure 6.10 the difference in behaviour between the normalized layers and the non-normalized ones is extremely evident, and their divergence point aligns perfectly with the raise in conditioning that we have already observed in Figure 6.1, making these measures a good proxy even for smaller networks than those observed by Agarwal, Awasthi, and Kale [AAK20].

Generalization. By measuring the local PL coefficients and estimates of the network Lipschitz coefficients, we are able to use Lemma 3 and compute a generalization bound for networks trained on MSE loss. Figures 6.11 and 6.12 shows the expected marked decrease with increasing optimization. Despite this, the obtained bounds are vacuous[7].

Scaling of the eigenvalues We are interested in understanding the change in conditioning due to varying the width of the network and the number of examples used to train it. To this aim we compare the measured eigenvalues to the ones predicted by random matrix theory [Bha13; Tao12]. In particular if we treat the matrix $F^m(\theta)$ as a random Gaussian matrix, using the result of Rudelson and Vershynin [RV09] we would expect that $\lambda_{\max} \sim \sqrt{\text{width} \cdot \text{examples}}$ and $\lambda_{\min} \sim \sqrt{\text{width}} - \sqrt{\text{examples}}$ and thus

$$\kappa(G^{m;\theta}) = \frac{\lambda_{\max}}{\lambda_{\min}} \sim \frac{1}{\sqrt{\frac{1}{\text{examples}}} - \sqrt{\frac{1}{\text{width}}}} \tag{6.8}$$

and this is exactly what we can observe in Figure 6.13, which shows that such relation holds at every layer.

Prediction of training loss with cross-entropy Using Lemma 22 for the cross-entropy loss and estimating $\|x_k - x^*\| \simeq 2\frac{f(x_k)-f^*}{G}$ where G is the Lipschitz constant of the function f, we obtain the results showed in Figures 6.14 and 6.15.

[6] A cheaper measure of network conditioning at a certain point in its training trajectory may be useful to stop training when the expected perfomance gains in terms of loss decrease are not justifyied by their expected costs. Moreover it could be used in the architecture search paradigm to filter out networks that have worse conditioning at initialization, as it is indicative of the loss optimization possibilities for that architecture.

[7] To the authors opinion, this outcome could be due both to the lack of sharpness on generalization estimates based on stability, as well as by the small number of examples used in each test (more examples per test could give non-vacuous estimates, even though these would require a lot more memory than what is currently available on typical GPUs) or by the large number of weights of the FCN architecture, which impact a lot on measures that depend on Lipschitz constants. We thus find that the results on generalization are inconclusive, and that more experiments are necessary to understand the reasons behind the vacuousness of the bounds.

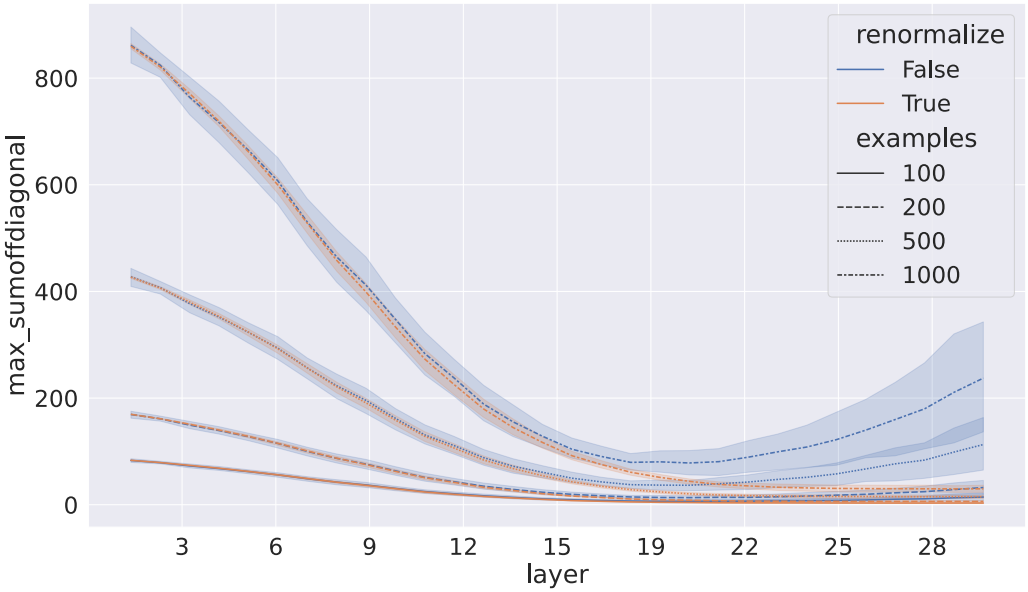


Figure 6.9: Maximum row-sum of the off-diagonal entries of the matrix $G^{m;\theta}$. Again we notice the typical decrease which stabilizes for normalized networks, and which starts growing again for non-normalized networks.

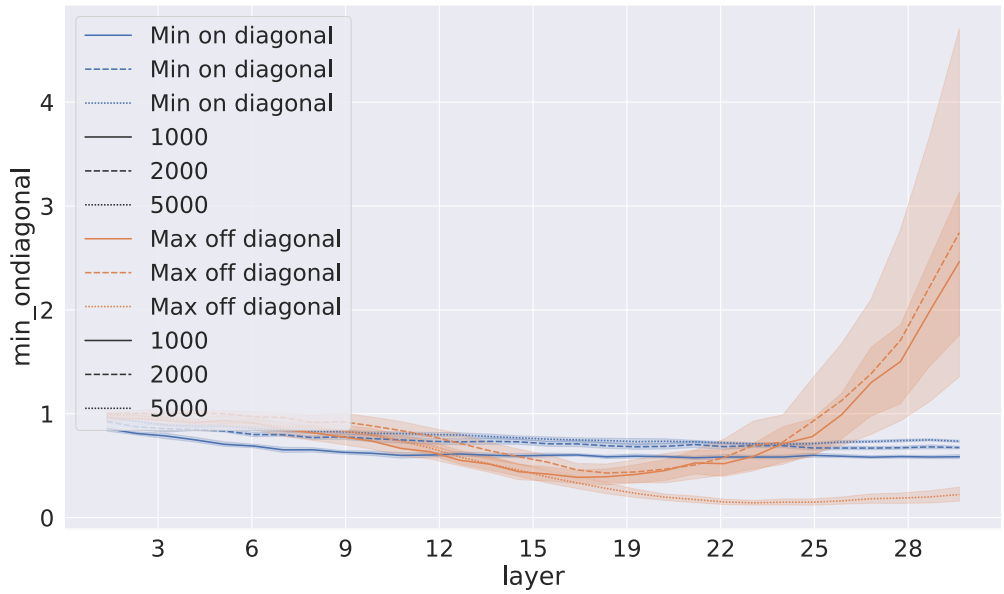


Figure 6.10: Minimum value of the on-diagonal entries and the maximum value of the off-diagonal entries of $G^{m;\theta}$ for non-renormalized networks. As it can be seen, there are some intermediate layers in which the minimum entry on the diagonal is higher than the maximum off the diagonal, thus empirically validating the theoretical calculations of Agarwal, Awasthi, and Kale [AAK20]. The condition later inverts, possibly due to finite-width effects.

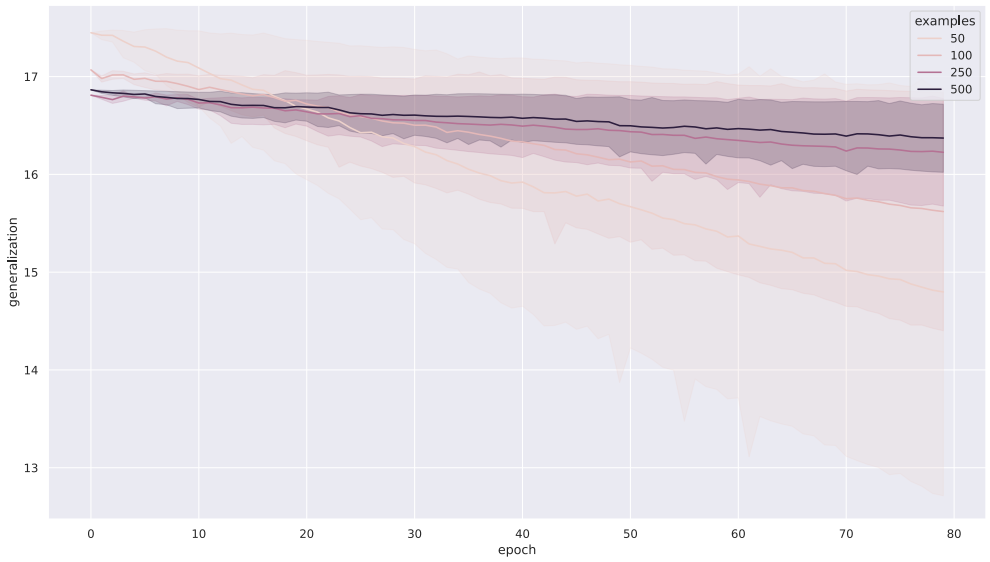


Figure 6.11: Generalization bounds for ReLU networks trained under MSE on logits obtained using Equation (4.13). All the bounds are vacuous, even if it is interesting to notice that the bounds effectively decrease with the number of epochs, but not as much as it would be expected with the number of considered examples.

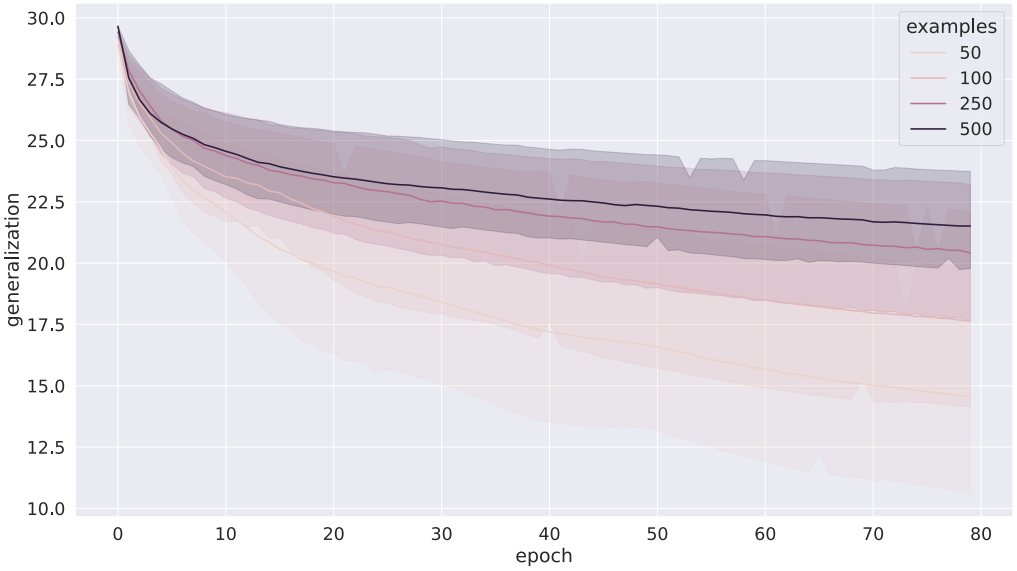


Figure 6.12: Generalization bounds for ReLU networks trained under MSE. Here we notice the same patterns.

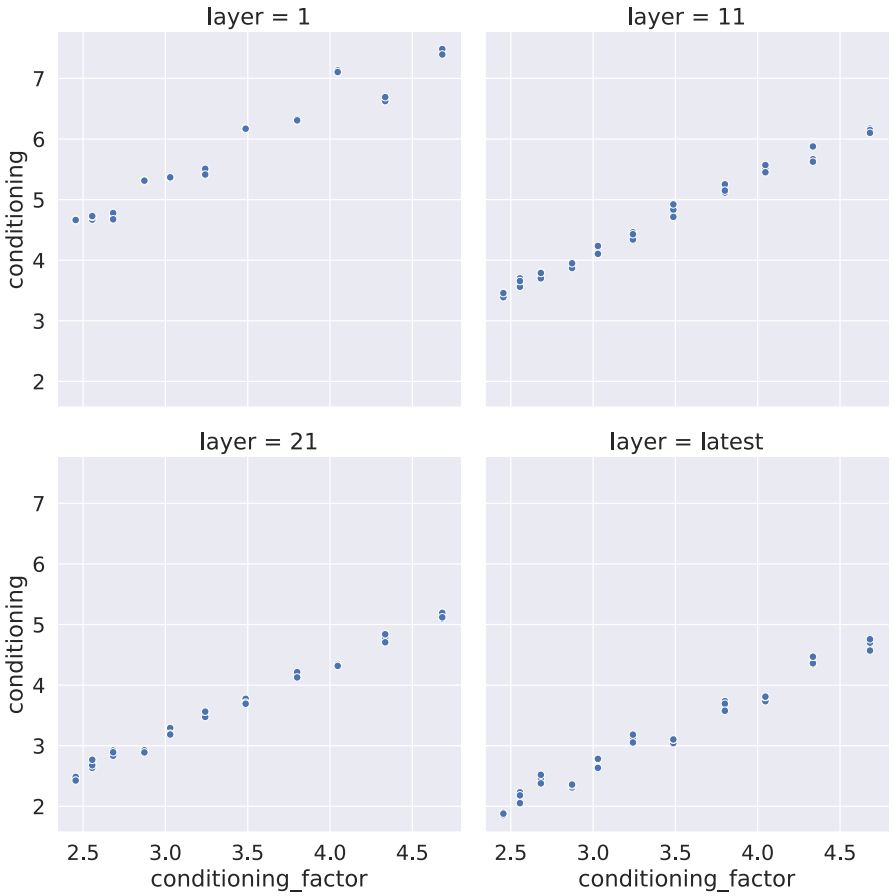


Figure 6.13: Plot of logarithmic conditioning for non-renormalized Tanh networks with respect to the factor $\frac{\sqrt{\text{examples}\cdot\text{width}}}{\sqrt{\text{width}}-\sqrt{\text{examples}}}$. We can observe that the conditioning is effectively proportional to the factor according to Equation (6.8).

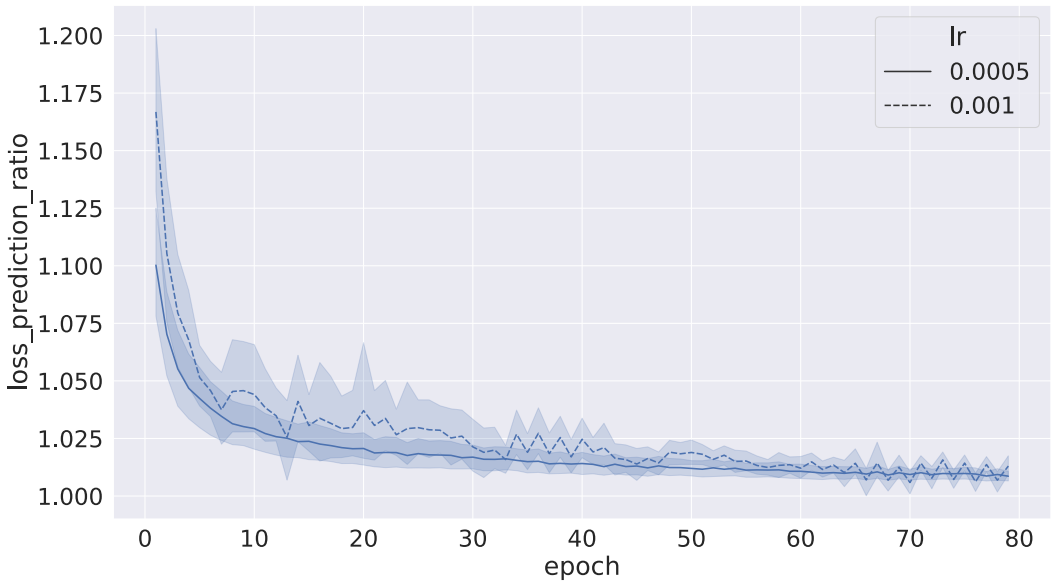


Figure 6.14: Loss prediction ratio for ReLU networks trained with cross-entropy loss.

We can observe how the prediction accuracy are similar to what we have obtained for the mean-squared-error loss: our estimates are too conservative for the first epochs, but align well with successive epochs. Further analyses are needed to perfectly align theoretical predictions with experimental ones.

Eigenvalues during Training We report in this section all the graphs about the behavior of the minimum measured eigenvalues of the $G^{m;\theta}$ matrix during training. As it can be seen from Figures 6.16 to 6.19, they don't decrease as much as predicted, effectively remaining constant in the case of MSE loss and also slightly improving over time for cross-entropy loss.

We also report in Figures 6.20 to 6.23 multiple graphs about the lowest predicted eigenvalues according to Equation (4.11) to show how much those bounds are pessimistic. We can notice that in multiple cases the bounds even become vacuous after the first epochs of training.

The results presented in Figures 6.20 and 6.21 illustrate the behavior of the lowest eigenvalues for ReLU and Tanh activation functions when trained under cross-entropy loss. The blue lines represent the actual lowest eigenvalues obtained during training, while the orange lines depict the predicted bounds derived from theoretical analysis. Notably, the ReLU networks exhibit a distinct pattern in their eigenvalue distribution, suggesting a more stable learning process compared to Tanh networks, which show greater variability. This discrepancy highlights the influence of activation functions on the convergence properties of neural networks, particularly in the context of cross-entropy loss.

In contrast, Figures 6.22 and 6.23 focus on the performance of the same activation functions under mean squared error (MSE) loss. The trends observed in these figures reveal that both ReLU and Tanh networks maintain a similar pattern in their lowest eigenvalues, albeit with a noticeable shift in the predicted bounds. The stability of the ReLU networks persists, while the Tanh networks continue to demonstrate fluctuations in their eigenvalue behavior. This suggests that while MSE loss may provide a different optimization landscape, the inherent characteristics of the activation functions still play a crucial role in shaping the learning dynamics of the networks.

6.4 Conclusions

The performed experiments on realistically-sides models have exposed some discrepancies with the available theories. We expose the most important ones in the authors opinion, and briefly comment on their possible consequences.

- The different behavior of conditioning with respect to renormalized and non-renormalized networks (Figure 6.1), contrary to the observations of Agarwal, Awasthi, and Kale [AAK20] which only seem to hold on very wide networks, as hinted by the role that width plays in delaying the difference between the two settings (Figure 6.2).

  A better understanding of this phenomenon could lead to find optimal network widths in the trade-off between computational requirements and better loss convergence, as well as the possibility to remove normalization layers on early layers of the networks, which doesn't seem to bring conditioning benefits according to Figure 6.1.

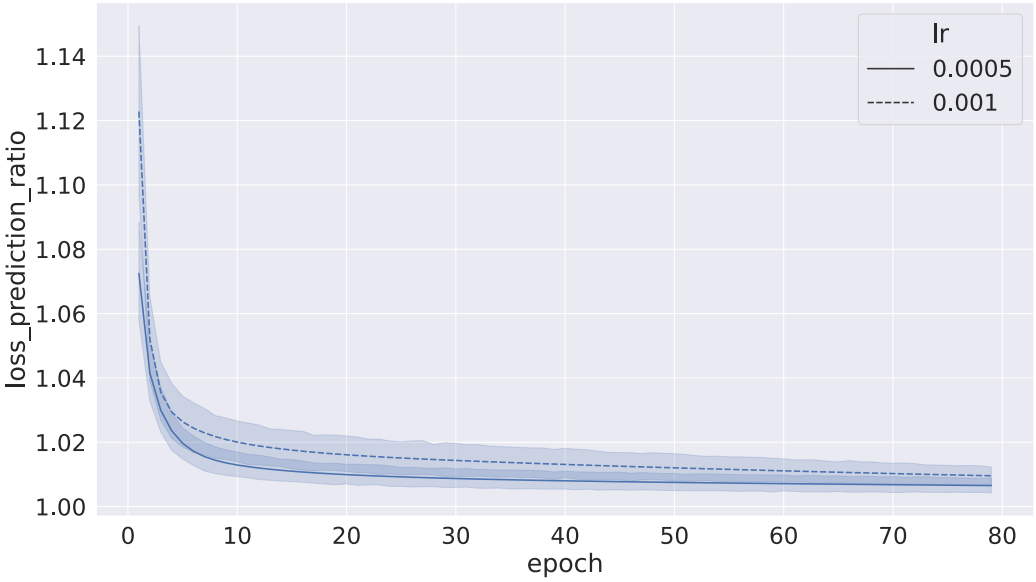


Figure 6.15: Loss prediction ratio for Tanh networks trained with cross-entropy loss.

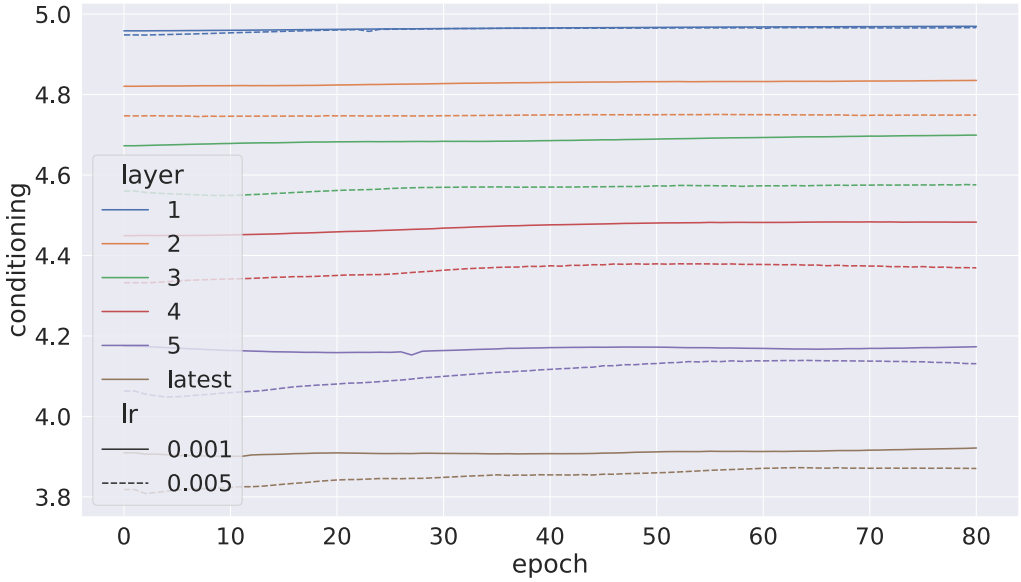


Figure 6.16: Log-conditioning during training for ReLU Networks under MSE Loss.

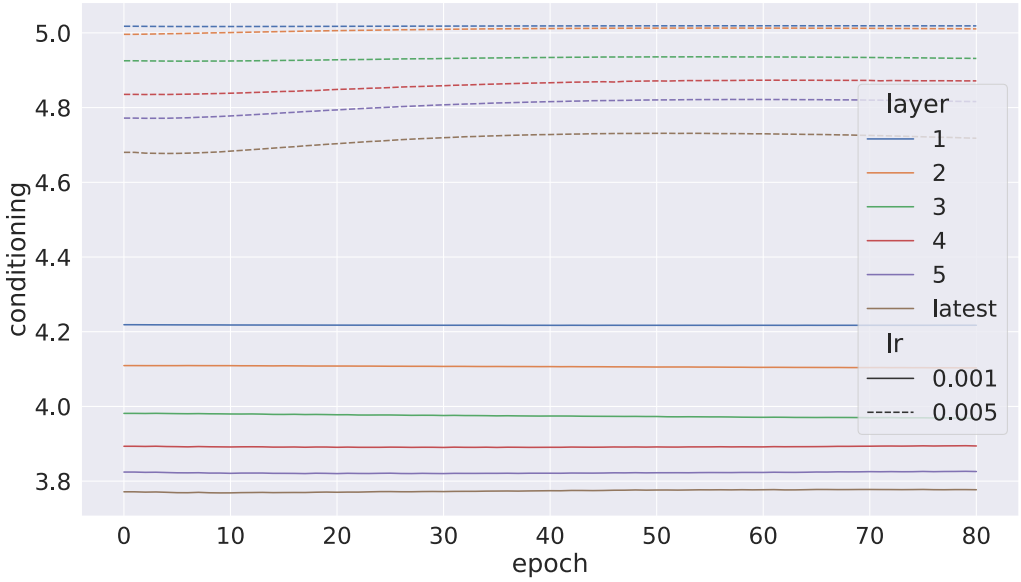


Figure 6.17: Log-conditioning during training for Tanh Networks under MSE Loss.

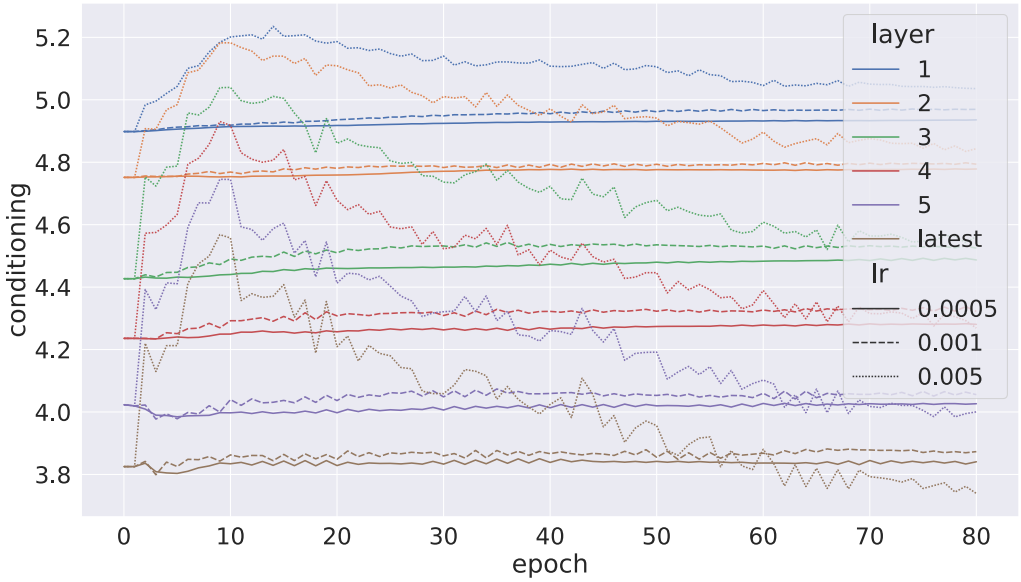


Figure 6.18: Log-conditioning during training for ReLU Networks under cross-entropy Loss.

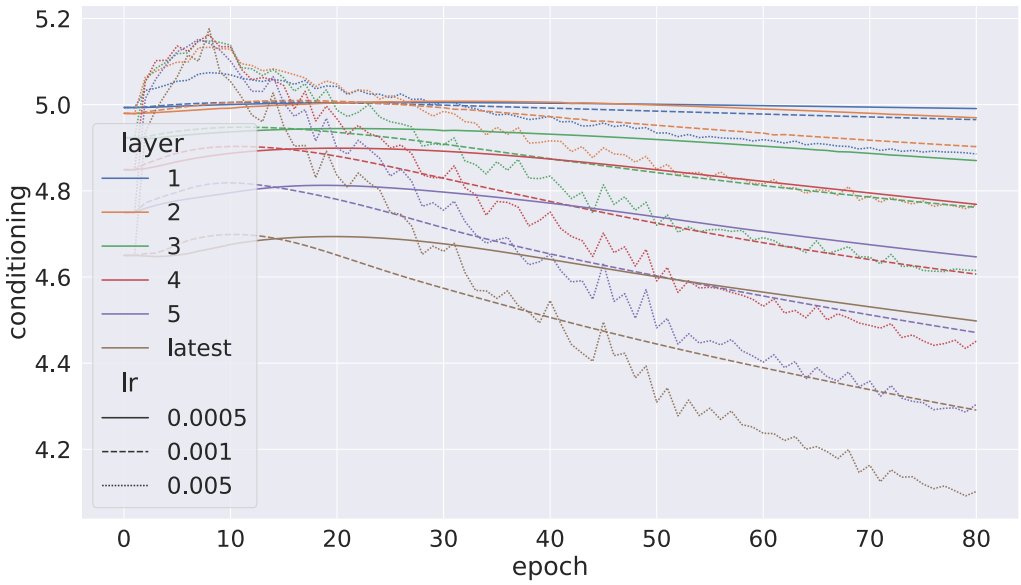


Figure 6.19: Log-conditioning during training for Tanh Networks under cross-entropy Loss.

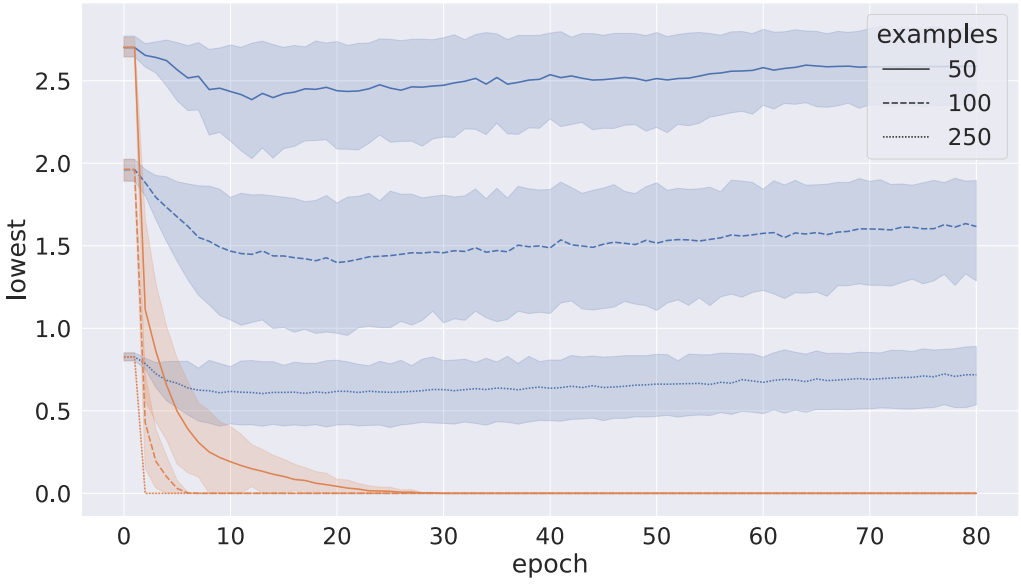


Figure 6.20: Actual lowest eigenvalues (in blue) and predicted bounds (in orange) for ReLU networks trained under cross-entropy loss.

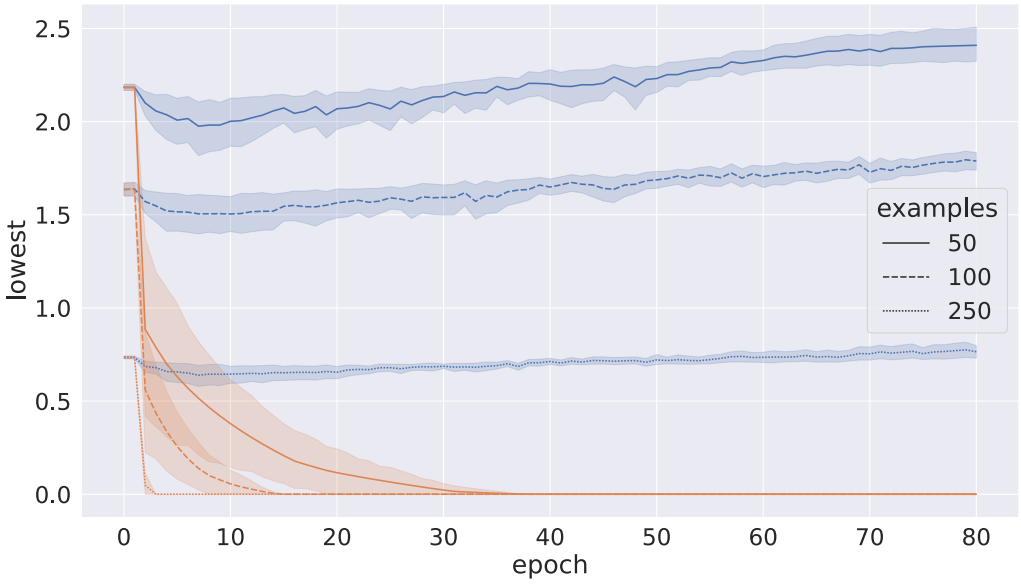


Figure 6.21: Actual lowest eigenvalues (in blue) and predicted bounds (in orange) for Tanh networks trained under cross-entropy loss.

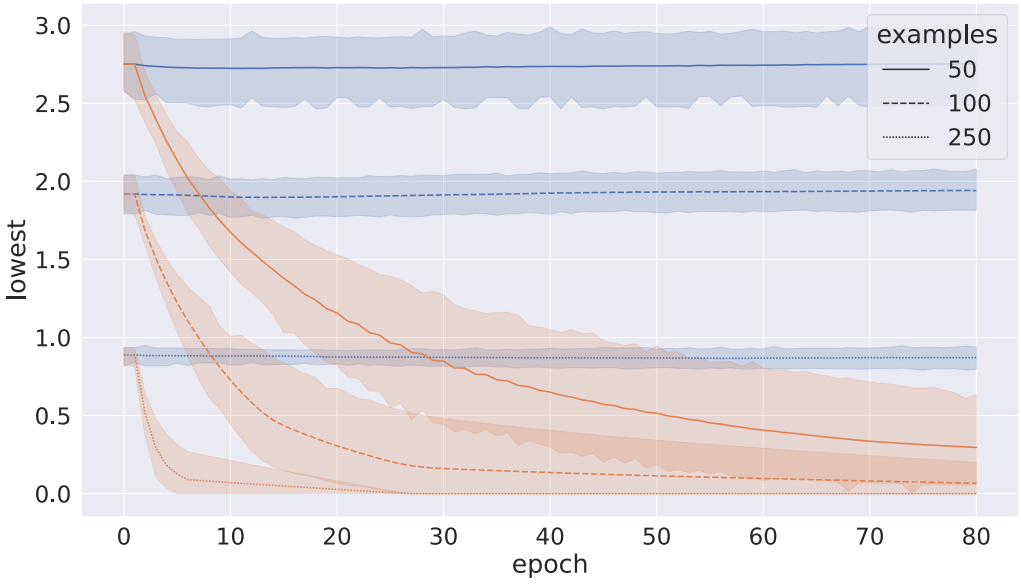


Figure 6.22: Actual lowest eigenvalues (in blue) and predicted bounds (in orange) for ReLU networks trained under MSE loss.

- The unexpected finding that conditioning worsens very slowly (and can possibly improve) during training, as current theories assume pessimistically that it worsens the farther the networks parameters are from their initialization (Figures 6.6 to 6.8).

  If this finding were confirmed by other experiments and theoretical reasons, existing theories would simplify considerably, as convergence guarantees all rely on conditioning not worsening too much during training.

- The fact that the available generalization bounds give vacuous estimates (Figures 6.11 and 6.12). Stricter generalization bounds could give guarantees to all the settings in which neural models are deployed in critical tasks, such as in automotive or in clinical screening.

Extensions to other architectures. Extensions of the presented theories to CNN, ResNets and Transformers are needed to form a more complete picture of network behavior. Those can be investigated within the same framework, but require different theoretical calculations in Lemma 21, Lemma 23 and an adaptation of the results of Agarwal, Awasthi, and Kale [AAK20] for each architecture, and are thus left to future works.

This work is, to our knowledge, the first that tries to explicitly quantify abstract theories about the inner working of neural networks, and to compare the bounds obtained with real-world experiments. We hope that such an approach can provide useful feedback to theoretical researchers by pointing out aspects that are not yet completely explained.

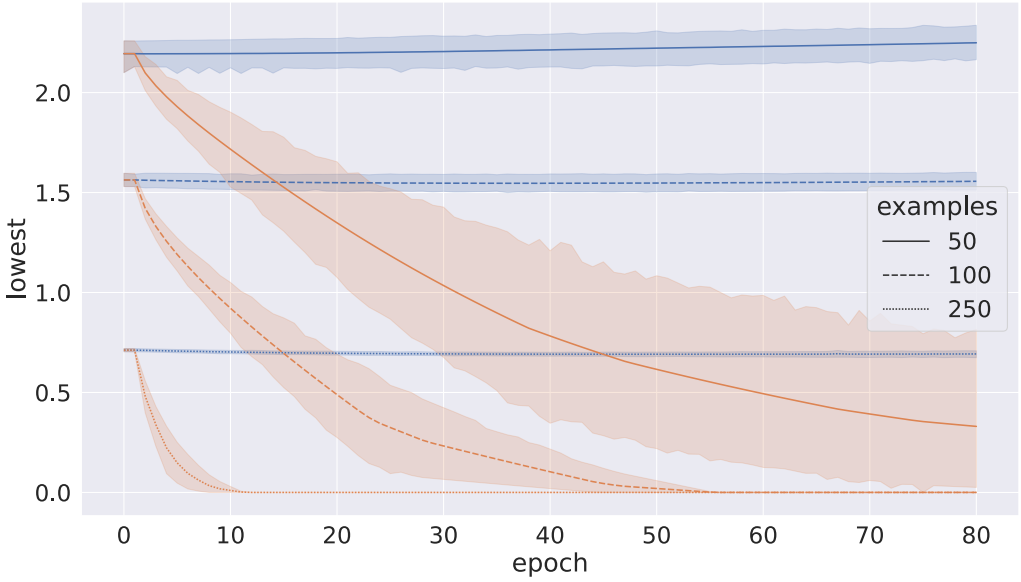


Figure 6.23: Actual lowest eigenvalues (in blue) and predicted bounds (in orange) for Tanh networks trained under MSE loss.

Chapter 7

Learning Rate Adaptation in Polyak–Łojasiewicz Optimization

7.1 Introduction

In this Chapter we theoretically analyze the deep network optimization problem under stochastic gradient descent without momentum, arguing for the relevance of the Polyak–Łojasiewicz condition and the conditioning of the finite tangent kernel matrix at each point of the optimization process. We do so on a less theoretically strict but more fine-grained approach than Liu, Zhu, and Belkin [LZB20], by separately considering the convex loss function and the neural network approximator, and carrying out a local analysis of the loss geometry to provide concrete algorithms for learning rate adaptation during the optimization.

This analysis allows us to derive theoretically justifyied ways to tune the learning rate during the optimization process, as well as computationally inexpensive proxies that can be practically used in the optimization process, thus relieving practitioners from the customary hyperparameter search on the learning rate.

The theoretical presentation is later followed by an in-depth empirical analysis which verifies the assumptions made by the theory on realistic deep neural networks, and compares the resulting optimization algorithm with grid search over various learning rate settings.

We think that such a mixed theoretical-practical approach has great potential in the development of practically relevant hyperparameter-free optimization algorithms for neural networks. The removal of hyperparameters diminishes the ecological impact of training big models by eliminating the standard hyperparameter search, and a solid theoretical foundation gives the practitioners stronger guarantees on the optimization level reached by deployed networks.

Overview of the Main Idea. We rely on the study on condition numbers in first-order optimization algorithms by Guille-Escuret, Girotti, Goujaud, and Mitliagkas [GGGM21], which identify the optimization conditions amenable to experimental hyperparameter tuning procedures because of their robustness to small perturbations in the objective function; two of them are important for our aim: $\mathrm{PL}^-(\mu) \cap \mathrm{PL}^+(L)$ and $\mathrm{PL}^-(\mu) \cap \mathrm{SC}^+(L)$.

Both rely on the lower Polyak–Łojasiewicz condition, a condition introduced by Polyak [Pol63] which is intimately related to the rate of decrease of the optimality gap in the gradient descent setting; the difference between them is that the first one $\mathrm{PL}^-(\mu) \cap \mathrm{PL}^+(L)$ is a pair of continuous (in the Guille-sense) conditions but requires star-convexity of the objective to hold to provide convergence guarantees, while the second one $\mathrm{PL}^-(\mu) \cap \mathrm{SC}^+(L)$ is not continuous (and thus not amenable to reliable hyperparameter tuning procedures) but is able to provide strong convergence guarantees without additional assumptions. What we are going to show is that, in the case of deep network models, $\mathrm{SC}^+(L) \approx \mathrm{PL}^+(L)$, thus obtaining both convergence guarantees and the possibility of continuously tuning the learning rate.

To do so, we first characterize the relation of the $\mathrm{SC}^+(L_{\mathrm{SC}})$ condition with the norm of the Hessian of the objective, and distinguish two terms in the Hessian; we show that the second of these terms is negligible with respect to the first one in typical neural networks optimization settings, and this allows us to derive a bound $L_{\mathrm{SC}} \simeq \lambda^*(\theta)\lambda_{\max}(\nabla^2\mathcal{L}(F(\theta)))$, which is thus expected to provide convergence guarantees when used as local hint to set the optimal learning rate.

We later study the $\mathrm{PL}^+(L_{\mathrm{PL}})$ condition in our setting and derive the same bound for it, i.e. $L_{\mathrm{PL}} = \lambda^*(\theta)\lambda_{\max}(\nabla^2\mathcal{L}(F(\theta)))$. The continuity of the PL^+ condition and the equality between the two derived bounds $L_{\mathrm{PL}} = L_{\mathrm{SC}}$ ensures that we can use the optimal learning rate tuning procedure derived from the $\mathrm{PL}^-(\mu) \cap \mathrm{SC}^+(L)$ pair (which already ensures convergence) while also ensuring the stability of that procedure.

The arguments presented above show a theoretically backed way to perform learning rate tuning in the context of Deep Neural Network, but we argue that the presented procedure is too conservative, usually giving too small a learning rate, and thus devise a more optimistic way of performing learning rate tuning starting from the Hessian approximation that is presented below, and we compare its empirical performance to SGD with grid search over multiple learning and to other means of approximating the Hessian, such as Gauss-Newton algorithm.

7.2 Setting and Notation

Let us recall the form of the Deep Network Optimization problem in Equation (3.2):

$$\operatorname*{argmin}_{\theta\in\Theta} \frac{1}{n}\sum_{k=1}^{n} \ell(x_k, y_k, f(y_k, \theta)) + R(\theta).$$

To set some notation, let us define $F : \Theta \to \hat{Y}^{\otimes n} \subseteq \mathbb{R}^{dn}$ and $\mathcal{L} : \hat{Y}^{\otimes n} \to \mathbb{R}$ by

$$F(\theta) = \left(,_{k=1}^{n} f(y_k, \theta)\right) \tag{7.1}$$

$$\mathcal{L}(,_{k=1}^{n} \hat{y}_k) = \frac{1}{n}\sum_{k=1}^{n} \ell(x_k, y_k, \hat{y}_k) \tag{7.2}$$

and define $h(\theta) := \mathcal{L}(F(\theta))$, and $h_I(\theta) = \frac{1}{|I|}\sum_{i\in I} \ell(x_i, y_i, f(x_i, \theta))$.

Notice how this view clearly separates the two functions $\mathcal{L}$ and F which have completely different properties: F can be seen as an overparameterization of the "real" objective function, and is

randomly initialized and thus amenable to techniques related to statistics and the central limit theorem, while $\mathcal{L}$ will be assumed convex, i.e. $\nabla^2\mathcal{L} \geq 0$, and with its minimum in correspondance with $\hat{y}_k = y_k$, so that it is amenable to classical optimization techniques. In what follows we will also ignore the regularization term for simplicity of the exposition, although it isn't hard to integrate in the theory following the projected Polyak–Łojasiewicz setting in Karimi, Nutini, and Schmidt [KNS16].

7.3 Condition Numbers in First Order Optimization

We briefly present the work of Guille-Escuret, Girotti, Goujaud, and Mitliagkas [GGGM21] on the study of continuity in conditions numbers of first-order optimization algorithms and ways to design robust tunings of their hyperparameters. We start by defining possible conditions that an objective can satisfy.

Definition 9 (Optimization Conditions)**.** *Let $\mu, L > 0$. Given a function $h : Z \subseteq \mathbb{R}^d \to \mathbb{R}$, we define $h_* := \inf_{z \in Z} h(z)$ its lower value; $\xi(z) := h(z) - h_*$ the optimality gap at a certain point. $\forall x, y \in Z$ we define the conditions:*

- ***Lower PL:*** $h \in PL^-(\mu)$ *iff* $\frac{1}{2}\|\nabla h(x)\|_2^2 \geq \mu(h(x) - h_*)$
- ***Upper PL:*** $h \in PL^+(L)$ *iff* $\frac{1}{2}\|\nabla h(x)\|_2^2 \leq L(h(x) - h_*)$
- ***Upper Strong Convexity:*** $h \in SC^+(L)$ *iff* $h(y) - h(x) \leq \langle \nabla h(x), y - x\rangle + \frac{L}{2}\|y - x\|_2^2$

Guille-Escuret, Girotti, Goujaud, and Mitliagkas introduce the notion of continuity of an optimization condition, defined in term of a star-norm $\|\cdot\|_*$ on functions; if a condition is continuous then for two functions f and g which are near in the star-norm, i.e. $\|f - g\|_*$ is small, the relative first-order optimization algoritm behaves similarly on them, and this allows for hyperparameter tuning procedures that are robust to small perturbations of the considered functions. We won't go into the technical details of their findings, and simply restate that PL^- and PL^+ are continuous conditions, while SC^+ is not.

They provide convergence guarantees for certain pairs of conditions, and relative linear rates [GGGM21, Table 2]. We are particularly interested in two of them:

- Under $\mathrm{SC}^+(L) \cap \mathrm{PL}^-(\mu)$, i.e. functions which are L-smooth and μ-PL, we have a linear rate of $\gamma = 1 - \frac{\mu}{L}$, as already proved by Karimi, Nutini, and Schmidt [KNS16], but this pair of conditions is not continuous, and thus not amenable to good hyperparameter tuning.
- Under $\mathrm{PL}^+(L) \cap \mathrm{PL}^-(\mu)$ and also assuming star-convexity of the objective with respect to a minimizer, the linear rate is $\gamma = 1 - \frac{\mu}{4L}$, and the pair of conditions is continuous.

In the next Sections we will study how these conditions change under reparameterization, and we will argue that, in the case of neural networks, we can measure a common bound to the $\mathrm{PL}^+(L) \cap \mathrm{PL}^-(\mu)$ conditioning and the $\mathrm{SC}^+(L) \cap \mathrm{PL}^-(\mu)$ one, thus ensuring both convergence and continuity without additional assumptions.

7.4 Change of Conditioning under Reparameterization

We now analyze how the optimization conditions change under reparameterization of $\mathcal{L}$ by F.

Let us remark that the upper strong convexity $\mathrm{SC}^+(L)$ condition is usually named L-smoothness, and it is equivalent to asking that the operator norm of the Hessian of h is bounded from above by L, i.e. $\left\|\nabla^2 h(\theta)\right\|_{op} \leq L$, so that a study of the relation between the Hessians of h and $\mathcal{L}$ is in order:

$$\frac{\partial^2 h(\theta)}{\partial\theta_i \partial\theta_j} = \sum_{\mu\nu} \frac{\partial F_\mu(\theta)}{\partial\theta_i} \frac{\partial^2 \mathcal{L}(F(\theta))}{\partial F_\mu \partial F_\nu} \frac{\partial F_\nu(\theta)}{\partial\theta_j} + \sum_{\mu} \frac{\partial \mathcal{L}(F(\theta))}{\partial F_\mu} \frac{\partial^2 F_\mu(\theta)}{\partial\theta_i \partial\theta_j} \tag{7.3}$$

$$= \nabla F(\theta) \nabla^2 \mathcal{L}(F(\theta)) \nabla F(\theta)^T + \nabla \mathcal{L}(F(\theta)) \cdot \nabla^2 F(\theta). \tag{7.4}$$

We can distinguish two terms in Equation (7.3): the first one is the kernel-driven term of the hessian, as the second term vanishes when F is linear like in the case of Kernel Methods; the second term can be seen as a perturbation of the kernel-driven Hessian induced by the curvature of the reparameterization function. In the next Section we will argue that the second term is negligible in multiple settings, and thus we can safely consider just the first part for the learning rate tuning procedure.

We now derive the formulas for how the $\mathrm{PL}^\pm$ conditions changes under reparameterization. Let us first define the finite tangent kernel matrix $K : \Theta \to \mathcal{M}(dn \times dn)$ by

$$K(\theta) = \nabla F(\theta) \nabla F(\theta)^T \tag{7.5}$$

and denote its highest and lowest eigenvalues by $\lambda^*(\theta) := \lambda_{\max}(K(\theta))$ and $\lambda_*(\theta) := \lambda_{\min}(K(\theta))$.

Lemma 12 (Transformations of the PL conditions under reparameterization)**.** *Assume that F is differentiable and let λ^* and λ_* be the maximum and minimum eigenvalues of the finite kernel matrix over the optimization region, i.e.*

$$\lambda_* := \min_{\theta\in\Theta} \lambda_*(\theta), \qquad \lambda^* := \max_{\theta\in\Theta} \lambda^*(\theta). \tag{7.6}$$

Then $\mathcal{L} \in PL^-(\mu) \implies \mathcal{L} \circ F \in PL^-(\lambda_* \mu)$ and $\mathcal{L} \in PL^+(L) \implies \mathcal{L} \circ F \in PL^+(\lambda^* L)$.

Proof. We consider only the first implication as the second one is obtained in the same way via majorization with the maximum eigenvalue of the kernel matrix:

$$\|\nabla h(\theta)\|^2 = \left\|\nabla \mathcal{L}(F(\theta))^T K(\theta) \nabla \mathcal{L}(F(\theta))\right\| \tag{7.7}$$

$$\geq \lambda_{\min}(K(\theta)) \|\nabla \mathcal{L}(F(\theta))\|^2 \tag{7.8}$$

$$\geq \lambda_* \mu \ (\mathcal{L}(F(\theta)) - \mathcal{L}_*) = \lambda_* \mu \ (h(\theta) - h_*) \tag{7.9}$$

where $\mathcal{L}_* := \min_{\zeta\in F(\Theta)} \mathcal{L}(\theta) = \min_{\theta\in\Theta} \mathcal{L}(F(\theta)) = h_*$. □

7.5 Negligibility of the Hessian Perturbation Term

In this Section we argue why the second term of the Hessian in Equation (7.3) is negligible in multiple settings for overparameterized networks with a linear output layer, which allows us to only consider the kernel-driven term for the tuning procedure.

1. Liu, Zhu, and Belkin [LZB22, Section 4.2] show that for wide networks with linear output layer the norm of the Hessian in a ball of a certain radius around the random initialization is on the order of $O(1/\sqrt{m})$ where m is the width of the network, while the kernel-driven term is $O(1)$. Thus the perturbation term is more negligible the wider the network is.

2. During the optimization trajectory, we have $\lim_{t\to\infty} \nabla\mathcal{L}(F(\theta_t)) = 0$, since the overparameterization of F ensures that we converge toward an interpolating θ for which $\forall k \quad \hat{y}_k = y_k$. On the other hand we know that $\nabla^2\mathcal{L}(F(\theta_t)) \geq 0$ remains positive semi-definite at the limit, and we thus can bound the kernel-driven term from below using $\lambda_*(\theta)$. The kernel-driven term thus becomes more dominant the more the optimization process progresses.

3. At the beginning of the optimization process we expect cancellations to occur in the second term due to the sum of multiple $\nabla^2 F_\mu(\theta)$ weighted with $\nabla_\mu\mathcal{L}(F(\theta))$, which we expect to behave randomly. More precisely, suppose that F has a linear output layer whose parameters are independent from the parameters of the other layers, and in which each coordinate is sampled from a distribution which is symmetric around zero (e.g. with i.i.d. uniform or gaussian entries); suppose moreover that the loss function $\mathcal{L}$ is $\mathcal{D}$-symmetric[1], i.e.

$$\forall \hat{y} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(x, y, \hat{y})] = \mathbb{E}_{(x,y)\sim\mathcal{D}}[\mathcal{L}(x, y, -\hat{y})], \tag{7.10}$$

   and thus by applying $\frac{\partial}{\partial\hat{y}}$ it holds that

$$\forall \hat{y} \quad \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\frac{\partial\mathcal{L}}{\partial\hat{y}}(x, y, -\hat{y})\right] = -\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\frac{\partial\mathcal{L}}{\partial\hat{y}}(x, y, \hat{y})\right]. \tag{7.11}$$

   Then by the symmetry of the randomness distribution in the last layer and its linearity, it holds that $f(x, \theta)$ has a symmetric distribution, and thus

$$\mathbb{E}_{(x,y)\sim\mathcal{D}}\mathbb{E}_{\theta^{\text{last}}}\left[\frac{\partial\mathcal{L}}{\partial F_\mu}(F(\theta))\right] = 0. \tag{7.12}$$

   By the Gradient Independence Assumption, which has been properly formalized in the limit of infinite networks[2] by Yang [Yan20b], we can treat $\nabla_\mu\mathcal{L}(F(\theta))$ independently from $\nabla^2 F_\mu(\theta)$, as the first one uses the outer layer weights in the forward pass, while the second one uses the outer layer weights in the backward pass, since it is made of derivatives with

[1] The $\mathcal{D}$-symmetric assumption is typically satisfyied in balanced settings, for example consider a regression task carried out under the mean squared error loss, with the regression target that is symmetric around zero. In other cases, for example if the regression target is symmetric around a positive constant c, the requirement of $\mathcal{D}$-symmetry suggests that the loss function is modified accordingly (shifting targets by a positive constant), in order for the cancellation arguments to carry through.

[2] [Yan20b, Condition 1] explicitly requires that the output layer is sampled independently and with zero mean from all other parameters and it is not used anywhere else in the interior of the network, a condition which we require in this reasonment.

respect to the network weights, thus by their independence we have

$$\mathbb{E}_{\theta^{\text{last}}}\mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\frac{\partial\mathcal{L}}{\partial\hat{y}}(x,y,f(x,\theta))\cdot\frac{\partial^2 f}{\partial\theta_i\partial\theta_j}(x,\theta)\right]=0, \tag{7.13}$$

i.e. on average over the network initialization and with a big number of examples, we expect the second term to be negligible also at the network initialization.

The presented arguments are strong hints for the negligibility of the second Hessian term during the whole optimization trajectory under certain network conditions, that we think can be made formal by careful quantification of all the arguments we presented; approximating the Hessian with its kernel-driven term allows us to conclude our argument:

Lemma 13 (Learning Rate Tuning Formula)**.** *For a convex loss function $\mathcal{L}$ and an overparameterized neural networks F with a linear output layer satisfying the symmetricity condition, a very good*[3] *local learning rate guaranteeing both convergence and robustness to tuning is:*

$$\eta(\theta):=\frac{1}{2\ \lambda^*(\theta)\ \left\|\nabla^2\mathcal{L}(F(\theta))\right\|_{op}}. \tag{7.14}$$

Proof. We first observe that for the optimization to progress we only need the condition pair $\mathrm{SC}^+(L)\cap\mathrm{PL}^-(\mu)$ to hold locally, and there is no need for global constant, e.g. we don't need to have a global upper bound on the Hessian of the objective but only an upper bound for a large enough region near the current point so that the current step is entirely contained in it.

Observe that, for a convex loss function $\mathcal{L}$, its PL^+ and SC^+ constants are the same, since they both coincide with the maximum Hessian eigenvalue over a neighbouring region, denoted by ζ.

By the arguments at the start of the Section we can bound the SC^+ constant of h by

$$\left\|\nabla^2 h(\theta)\right\|_{op}\simeq\left\|\nabla F(\theta)\nabla^2\mathcal{L}(F(\theta))\nabla F(\theta)^T\right\|_{op}\leq\zeta\cdot\lambda^*(\theta). \tag{7.15}$$

By Lemma 12 we can bound the PL^+ constant of h by the same value:

$$\|\nabla h(\theta)\|^2\leq\lambda^*(\theta)\|\nabla\mathcal{L}(F(\theta))\|^2\leq\zeta\cdot\lambda^*(\theta)\cdot\xi(\theta). \tag{7.16}$$

We remind the reader that the optimal learning rate under the $\mathrm{PL}^-(\mu)\cap\mathrm{SC}^+(L)$ condition pair is $1/2L$ [KNS16] and that every learning rate $\eta<1/L$ is enough for the optimization to converge, albeit more slowly.

Thus $\zeta\cdot\lambda^*(\theta)$ is a bound both for the SC^+ and the PL^+ constant of h, and a learning rate tuning based on it inherits the convergence of the $\mathrm{SC}^+\cap\mathrm{PL}^-$ pair and the continuity of $\mathrm{PL}^+\cap\mathrm{PL}^-$. □

The empirical uncertainties related to the performed Hessian approximation, and to the measuring of $\lambda^*(\theta)$ at a single point instead of over a neighbourhood, as well as the estimation errors

[3] The bounds in the derivation of the learning rate could be made stricter by considering the mean curvature of $\nabla^2\mathcal{L}$ instead of the maximum curvature assuming that the backward gradients are isotropically distributed over all directions; this stronger bound could be useful in the case of weakly-convex losses. On the other hand, the dependence on $\lambda^*(\theta)$ cannot be improved much, since it is known that for very deep networks the conditioning of the $K(\theta)$ matrix tends to one in the number of layers [AAK20], thus we would have $\lambda_*(\theta)\simeq\lambda^*(\theta)$ with coinciding upper and lower estimates for $\|\nabla h(\theta)\|^2$.

that stochastic samples of examples introduces in the estimation of the correct SC^+ constant shouldn't prevent convergence because the employed method is still convergent for learning rates $< 1/L$ as we already remarked[4].

7.6 Learning Rate Tuning and Empirical Proxies

The main advantages of the proposed method with respect to a traditional hyperparameter optimization are to relieve practictioners from having to guess the initial learning rate and to train multiple models to see which does better; to adapt the learning rate locally in dependence of the measured smoothness of the function, which may help in avoiding divergence issues that occur later on in training, and which have given rise to a plethora of heuristic methods to reduce the learning rate during training.

In this Section we comment on cheaper proxies that can be used to speed up the computation of the required $\lambda^*(\theta)$: the experimental measure of $\lambda^*(\theta)$ can infact be done via iterative power methods [Hou06] which only need to implicitly compute the matrix-vector product with $K(\theta)$, easily implementable using the single $\nabla F(\theta)$ with two jacobian-vector products, and thus can be carried out with a linear amount of additional memory[5].

The number of necessary iteration of the power method to obtain convergence to the highest eigenvalue may however be high enough to render the method too slow for practical purposes[6].

A cheaper method relies on Gershgorin Circle Theorems [Ger31] to estimate the highest eigenvalue from the sum of off-diagonal entries of some of the $K(\theta)$ matrix columns, thus requiring a small number of passes on the $\nabla F(\theta)$ matrix[7]. We expect this to be a reliable proxy, by the

[4]Despite these assurances, these assumptions will need to be empirically checked to make sure that the presented situation do in fact occur in deep networks, and to verify how much the arguments given for the vanishing of the second Hessian term do hold in practice.

[5]The iterative power method is a fundamental algorithm used to approximate the dominant eigenvalue (i.e., the eigenvalue with the largest magnitude) and its corresponding eigenvector of a square matrix A. The method works by exploiting the property that repeated multiplication of a matrix by a vector amplifies the component of the vector in the direction of the eigenvector corresponding to the largest eigenvalue.

To begin, an initial vector $\mathbf{v}^{(0)}$ (often chosen randomly or as a vector with all entries equal) is multiplied by the matrix A: $\mathbf{v}^{(k+1)} = A\mathbf{v}^{(k)}$. After each iteration, the resulting vector is normalized, typically by dividing by its norm, to avoid numerical overflow and to stabilize the algorithm. The iteration continues until the sequence of vectors $\mathbf{v}^{(k)}$ converges to the eigenvector corresponding to the dominant eigenvalue.

Once the dominant eigenvector is approximated, the corresponding eigenvalue can be computed using the Rayleigh quotient: $\lambda = \frac{\mathbf{v}^T A\mathbf{v}}{\mathbf{v}^T\mathbf{v}}$, where $\mathbf{v}$ is the normalized eigenvector approximation. The method converges quickly if the matrix A has a unique largest eigenvalue in magnitude and the initial vector has a non-zero component in the direction of the dominant eigenvector. However, convergence can be slow or fail if the dominant eigenvalue has multiplicity or is close in magnitude to other eigenvalues.

[6]However, as we expect $F(\theta)$ (and thus $\lambda^*(K(\theta))$) to move slowly when updating θ because of its random-matrix behavior, one can just reuse the previously found eigenvectors as initialization for the power method, which should result in a much faster convergence.

[7]The Gershgorin Circle Theorems provide a method to estimate the location of eigenvalues of a square matrix. The theorems state that every eigenvalue of a matrix lies within at least one Gershgorin disk, which is defined for each row (or column) of the matrix. Specifically, for a matrix $A = [a_{ij}]$, the Gershgorin disk corresponding to the i-th row is centered at a_{ii} (the diagonal entry) with a radius equal to the sum of the absolute values of the off-diagonal entries in that row, $R_i = \sum_{j \neq i} |a_{ij}|$. Formally, the eigenvalues of A satisfy:

$$\lambda \in \bigcup_{i=1}^{n} \{z \in \mathbb{C} : |z - a_{ii}| \leq R_i\}.$$

This theorem is particularly useful for estimating the dominant eigenvalue of a matrix. When the off-diagonal

random matrix nature of $K(\theta)$ and the theoretical fact that, in wide networks with many layers and normalized input examples, the off-diagonal entries of $K(\theta)$ tend to zero in the number of layers [AAK20], which helps in achieving stronger bounds in Gershgorin Circle Theorems.

In what follow, we discuss a more practical way to perform learning rate tuning: as we have said, the bounds of Lemma 13 are pessimistic, and could be made to work better by considering the mean eigenvalue of $\nabla^2\mathcal{L}$ instead of its operator norm; an alternative way to achieve the same result is to consider at each step the local approximation of the $\mathcal{L} \circ F$ function by its quadratic Taylor series, with the second term of Equation (7.3) removed, which will give us a sound way to derive the optimal learning rate at each step.

While this may seem just an exercise in optimization theory, we want to highlight the relevance of such a procedure in practical setups: in recent years the great improvements in Deep Learning accuracy have been coupled with huge energy expenses generated by the procedures needed to train large models [LVL22]: for example, the common practice of employing grid search over the model hyperparameters is responsible for a multiple-fold increase. Our adaptive learning rate strategy for SGD is a solid first step toward hyperparameter-free optimization of Deep Networks without sacrificing performance, as we will see that it is able to outperform SGD with grid search over learning rates at just twice the cost of a single SGD run.

7.7 Proposed Learning Rate Tuning Method

In Section 7.5 we have motivated why the second term in the Hessian expansion can be safely ignored and that the approximation $\nabla^2 h_S(\theta) \simeq Q_S(\theta) := \nabla F_S(\theta)^T \nabla^2 \mathcal{L}_S(F_S(\theta)) \nabla F_S(\theta)$ is exact in limit settings[8], and sound in typical cases due to the expected cancellations in the second term; moreover, the approximation is Positive Semi-Definite (PSD)[9] as $\nabla^2\mathcal{L}(F(\theta))$ is PSD because of the convexity of $\mathcal{L}$.

The chosen approximation provides a Hessian that doesn't necessarily vanish as the optimization progresses, in contrast with the more standard Gauss-Newton method[10], and enables it to better follow the local geometry of the optimization problem.

Per-minibatch Computation of the Learning Rate. Consider a random minibatch $I \subseteq S$ and the update equation $\theta' = \theta - \eta \nabla h_I(\theta)$: we want to search for the η minimizing $h_I(\theta')$, that we approximate as

$$\operatorname*{argmin}_{\eta} \; h_I(\theta) - \eta \nabla h_I(\theta)^T \nabla h_I(\theta) + \frac{1}{2}\eta^2 \nabla h_I(\theta)^T Q_I(\theta) \nabla h_I(\theta),$$

from which the optimal learning rate is $\eta(\theta) := \frac{\nabla h_I(\theta)^T \nabla h_I(\theta)}{\nabla h_I(\theta)^T Q_I(\theta) \nabla h_I(\theta)}$.

entries are small relative to the diagonal entries, as in the context of $K(\theta)$ matrices for wide neural networks, the disks tend to shrink, leading to tighter bounds on the eigenvalues. This makes Gershgorin Circle Theorems an efficient and memory-light tool for approximating the largest eigenvalue without explicitly computing the full spectrum of $K(\theta)$.

[8] Both on very wide networks (and NTK [JGH18]) and on all networks at empirical convergence.

[9] It is important that the approximation is PSD as this ensures that the taken direction is a descent direction, and enables the usage of Conjugate Gradient Descent [HZ06] on the objective.

[10] Which approximate the Hessian using $\nabla\mathcal{L}(F(\theta))^T \nabla\mathcal{L}(F(\theta))$ instead of $\nabla^2\mathcal{L}(F(\theta))$.

A difficulty in a direct application of the above formula is the high variance of the obtained estimates for the learning rate, especially problematic in later optimization phases; we thus consider an exponentially weighted average between different mini-batches approximations of the learning rate[11]. Let us call η_k the approximation for the k-th minibatch; the proposed effective rate is

$$\hat{\eta}_k := \exp\left(\frac{\sum_{i=0}^{k} \beta^{k-i} \log \eta_i}{\sum_{i=0}^{k} \beta^{k-i}}\right),$$

where $\beta \in [0, 1]$ is a parameter determining how much earlier estimates are considered[12].

For clarity of exposition, Algorithm 1 details the full algorithm pseudo-code.

inputs
- $S := \{(x_i, y_i) \in X \times Y\}_{i=1}^{n}$; `// Input dataset`
- $\theta_0 \in \Theta$; `// Initial parameter`
- $b \in \mathbb{N}$; `// Size of the minibatches`
- $\beta \in (0, 1) = 0.99$; `// Discount factor in exponential mean`
- $\varepsilon = 1\text{e}{-10}$; `// Hessian Regularization Parameter`

initialization: $c_0 \leftarrow 0.0, s_0 \leftarrow 0.0$

for $k \leftarrow 0$ **to** ∞ **do**
- $I \leftarrow$ `Sample(`S, b`)` ; `// Sample minibatch from the dataset`
- $G \leftarrow \nabla h_I(\theta_k)$; `// Backward pass on the minibatch`
- $v \leftarrow \nabla F_I(\theta_k)^T G$; `// Minibatch jacobian-vector product`
- $Q \leftarrow \nabla^2 \mathcal{L}_I(F_I(\theta_k))$; `// Minibatch per-example hessian`
- $\eta \leftarrow (G^T G)/(v^T Q v + \varepsilon G^T G)$; `// Minibatch LR estimate`
- $s_{k+1} \leftarrow \beta s_k + \log \eta$; `// LR estimate update`
- $c_{k+1} \leftarrow \beta c_k + 1.0$; `// Denominator update`
- $\hat{\eta} \leftarrow \exp(s_{k+1}/c_{k+1})$; `// Effective LR`
- $\theta_{k+1} \leftarrow \theta_k - \hat{\eta} G$; `// Parameter update`

end

Algorithm 1: ADaptive LEarning Rate SGD

7.8 Experimental Results

To validate the effectiveness of the proposed algorithm, we perform experiments on Convolutional architectures with and without residual connections and on Vision Transformers (ViT)[13] [Dos+20] of different widths and number of layers, different activation functions, and differ-

[11] The underlying intuition is that gradient steps become smaller as training progresses, and thereby the local curvature becomes more stable from one minibatch to the next one, which enables a simple averaging procedure to aggregate the information contained in multiple mini-batches without a strong need to increase the mini-batch size.

[12] We need to average the *logarithms* of η_k in the above formulas, since the denominator $\nabla h_I(\theta)^T Q_I(\theta) \nabla h_I(\theta)$ can be arbitrarily near zero, as this better reflects the variations expected in different estimates.

In contrast, in the Gauss-Newton approximation, we average the η_k as they are, since in that case the denominator is with very high probability strictly greater than zero, and the distribution of the learning rates can be suitably approximated by a Gaussian distribution.

Moreover, we would ideally like to have different $\beta_k \to 1^-$, since at the start of the optimization earlier estimates aren't much informative due to large steps, while near the optimization end values of $\beta \simeq 1$ are optimal to ensure full usage of the information contained in each minibatch.

We suggest $\beta = 0.99$ as a good fixed value ensuring a decent tradeoff between being fast enough at the start and reach of the best end results; we leave β-autotuning to future research.

[13] The used networks are trained without momentum nor weight decay as this would prevent a proper comparison; this explains why the obtained results aren't SOTA.

ent random seeds, totaling more than one hundred variations[14]. For each network variation, we compare: (i) the best result of a cost-intensive grid search over ten SGD learning rates[15] ranging in logarithmic scale from 1e−5 to 3e−1; (ii) our algorithm (ADLER) with $\beta = 0.99, \varepsilon = 1e{-}10$; and (iii) Gauss-Newton approximation (GN) with $\beta = 0.99$.

For a fair comparison, we compare ADLER to other first-order algorithms operating over an efficient approximation of the local geometric structure. We don not compare ADLER with ADAM, as the latter has both a form of momentum and a preconditioner, which are not considered by our algorithm.

Models	**Activation**	**ADLER**	**GN**	**SGD**
CNN+ResNet	relu	**0.399 ± 0.026**	0.303 ± 0.034	0.304 ± 0.056
CNN+ResNet	sigmoid	0.312 ± 0.081	0.357 ± 0.062	**0.360 ± 0.022**
CNN+ResNet	tanh	**0.364 ± 0.040**	0.337 ± 0.031	0.339 ± 0.019
ViT	relu	**0.268 ± 0.007**	0.219 ± 0.033	0.240 ± 0.007
ViT	sigmoid	**0.270 ± 0.004**	0.212 ± 0.042	0.231 ± 0.039
ViT	tanh	**0.245 ± 0.024**	0.213 ± 0.037	0.238 ± 0.023

Table 7.1: Final accuracy for CIFAR100 under different activations and models.

Figure 7.1 and Table 7.1 show that ADLER performs much better than the Gauss-Newton approximation; moreover, it is on par with SGD grid search for CIFAR10 and achieves a better performance on CIFAR100. Accuracy performance apart, the method is quite effective in terms of the optimization of the given objective function, as it can be appreciated from the logarithmic slope of the training loss compared to the other methods (Figure 7.1).

Infact, the method is so effective at optimization that we see that the test loss starts to increase dramatically after the first ten epochs, while at the same time the accuracy performance stabilizes (Figure 7.1). We think that this overfitting effect is due to the employment of non-regularized ERM, and can be mitigated by a policy to increase the mini-batch size in time, which would allow the method to obtain more accurate geometrical information.

Let us now analyze the way in which the learning rates are varied throughout the learning epochs: in Figure 7.2 we can observe that they resemble cyclical learning rate strategies [Smi17][16] in the case of ReLU networks; moreover we observe that the learning rates produced by GN inexorably decrease with time and are generally smaller than either of the other methods, which hints at the reason underlying its worse overall performance. ADLER achieves a comparable accuracy to grid-search SGD on CIFAR10[17], while it shows a marked performance improvement on the more challenging CIFAR100 dataset (Table 7.1). While this may seem a contradiction, we think the low performance on CIFAR10 can be due to the fact that on easier problems the local geometry appears flatter and thus the algorithm takes greater steps; these steps may sometimes be so big that the local approximation no longer holds. For this reason, we look forward to integrating a trust-region bound [Yua00] in future extensions of the method.

[14]The full code and the experiment data are available at gitlab.com/dbalboni/ADLER.
[15]As our method does not have momentum terms, we only compare against naive SGD.
[16]In which learning rates are increased and decreased cyclically throught training.
[17]The accuracy of the two methods is less than a standard deviation apart.

7.9 Conclusions

In this Chapter (1) we have shown the theoretical base that allows learning rate tuning procedures to be used on Deep Neural Networks without disturbing their convergence properties based on the Polyak–Łojasiewicz condition; (2) we have presented an algorithm that relieves practitioners from computational and energy costs of grid search on SGD learning rates, without loss of predictive performance when optimizing popular convolutional architectures and Vision Transformers; (3) we have proposed a sound PSD Hessian approximation that can be easily instantiated in Conjugate Gradient methods, thus enabling to possibly match the performance of accelerated algorithms, and marked a first step towards the removal of hyperparameters from optimization procedures in Deep Learning.

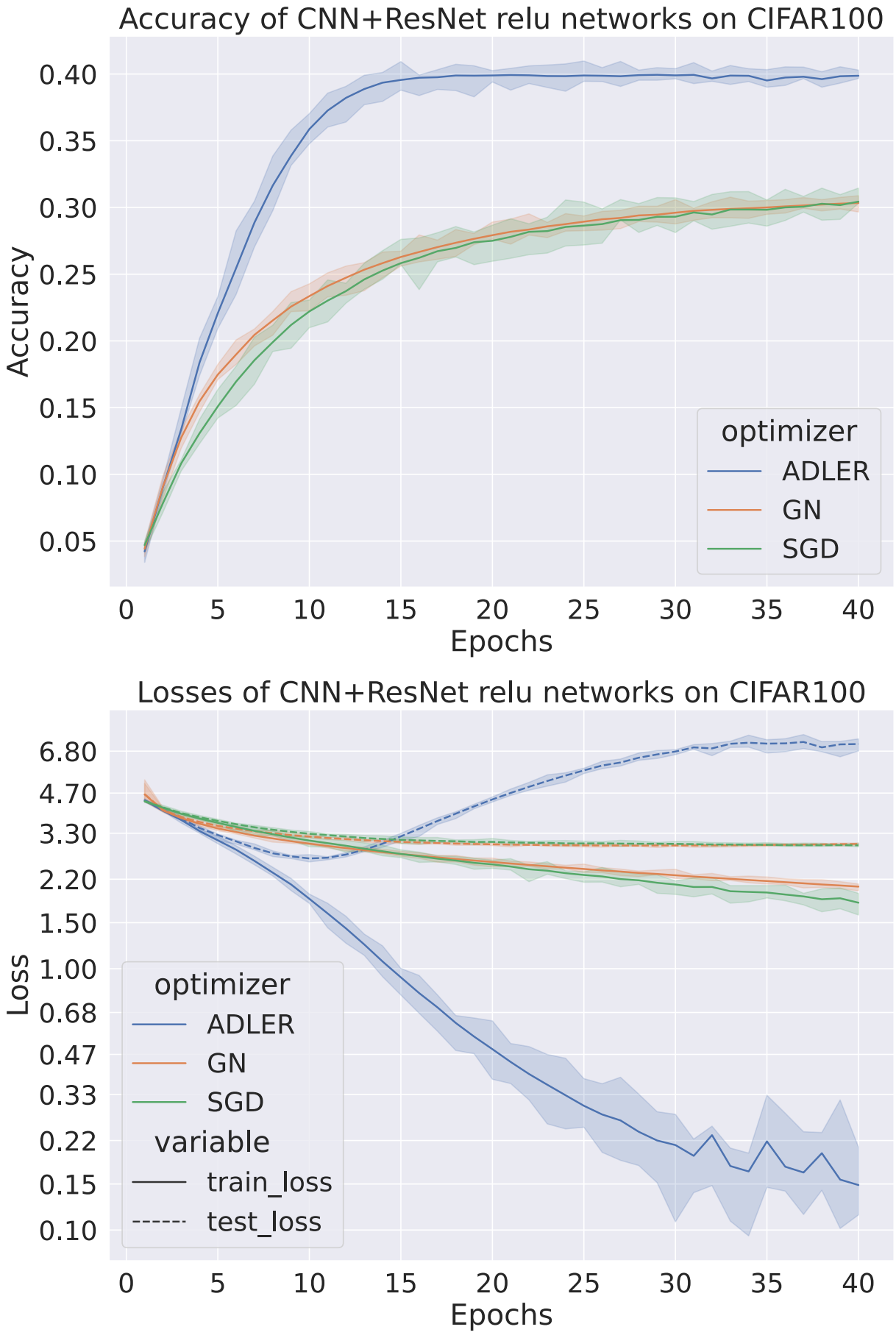
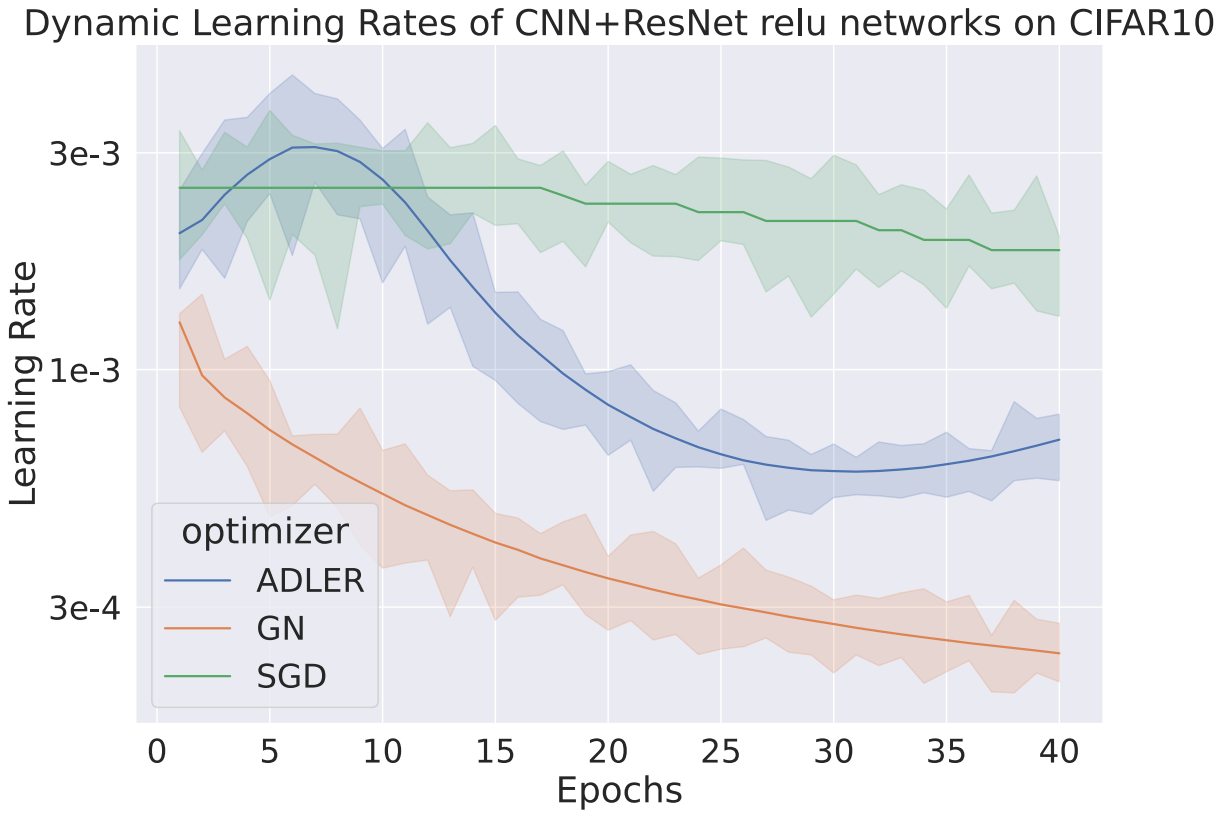


Figure 7.1: Evolution of ReLU networks accuracy and training loss on CIFAR100; optimizers are shown in different colors with shaded 95% confidence intervals.



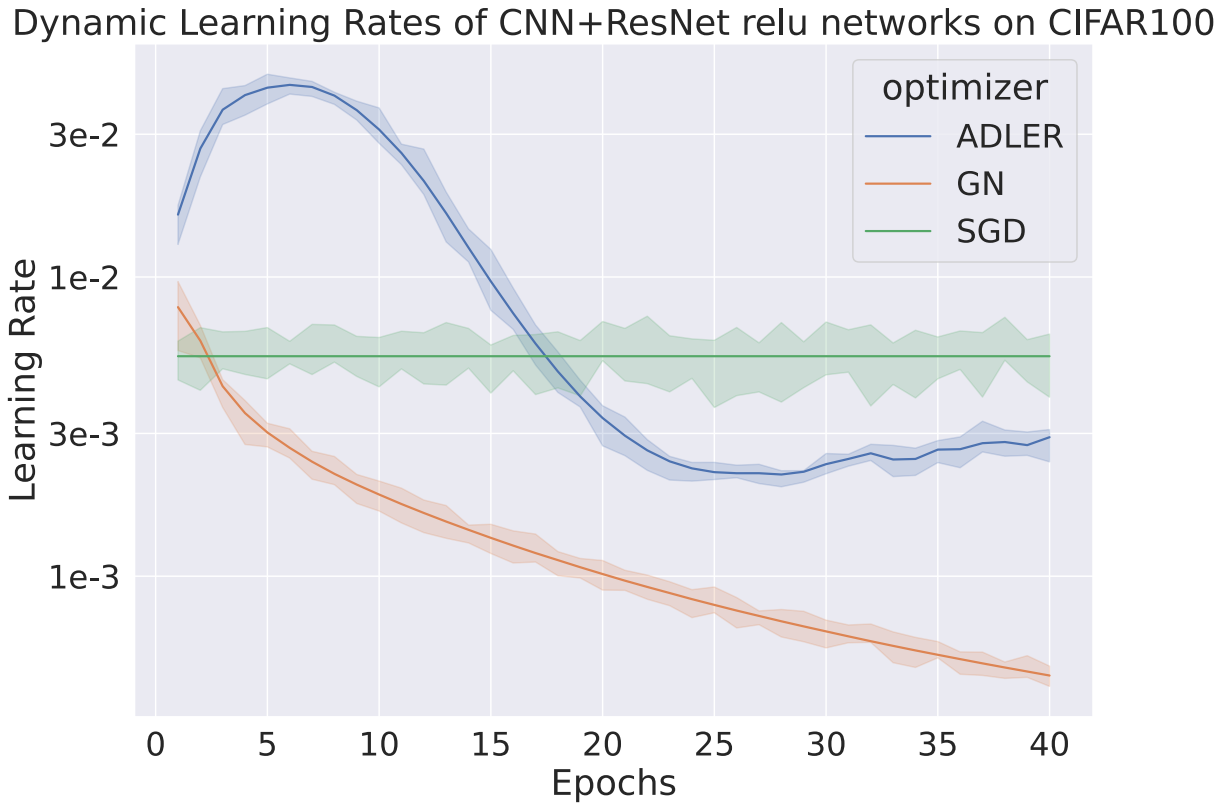


Figure 7.2: Evolution of learning rates during optimization: for SGD we plot the one achieving the best performance at each epoch, while for the purpose of training, it is actually kept fixed to its grid search value.

Chapter 8

Conclusions and Future Research

In wrapping up, we addressed fundamental questions about how to understand and optimize deep neural networks. Beginning with Mean Field Theory (Chapter 1) and Neural Tangent Kernels (Chapter 2), we developed ways to simplify and interpret the complex behavior of these models. By viewing Deep Networks from an optimization perspective (Chapter 3) and applying the Polyak–Łojasiewicz condition (Chapter 4), we gained practical insights into managing convergence and generalization challenges inherent in non-convex optimization, particularly for deep networks. This theoretical groundwork laid the foundation for our contributions.

Deep Kernel Networks. In Chapter 5, we established a connection between Deep Neural Networks and Deep Kernel Methods, demonstrating that convergence proofs for Neural Networks do not need to be conducted for each architecture individually by utilizing the framework of deep kernel methods. We rederived a known result of Li, Ding, and Sun [LDS18] for Deep Kernel Networks and in more generality.

Testing the Predictions of the Polyak–Łojasiewicz Theory. In Chapter 6, we conducted a comprehensive literature review on the Polyak–Łojasiewicz condition and its potential applications in modeling Deep Networks, focusing on aspects such as convergence, conditioning, and generalization. We also performed experiments to validate the alignment with the theoretical framework.

The performed experiments on realistically-sides models have exposed some discrepancies with the available theories. We expose the most important ones in the authors opinion, and briefly comment on their possible consequences.

- The different behavior of conditioning with respect to renormalized and non-renormalized networks (Figure 6.1), contrary to the observations of Agarwal, Awasthi, and Kale [AAK20] which only seem to hold on very wide networks, as hinted by the role that width plays in delaying the difference between the two settings (Figure 6.2).

  A better understanding of this phenomenon could lead to find optimal network widths in the trade-off between computational requirements and better loss convergence, as well as the possibility to remove normalization layers on early layers of the networks, which doesn't seem to bring conditioning benefits according to Figure 6.1.

- The unexpected finding that conditioning worsens very slowly (and can possibly improve) during training, as current theories assume pessimistically that it worsens the farther the networks parameters are from their initialization (Figures 6.6 to 6.8).

  If this finding were confirmed by other experiments and theoretical reasons, existing theories would simplify considerably, as convergence guarantees all rely on conditioning not worsening too much during training.

- The fact that the available generalization bounds give vacuous estimates (Figures 6.11 and 6.12). Stricter generalization bounds could give guarantees to all the settings in which neural models are deployed in critical tasks, such as in automotive or in clinical screening.

Learning Rate Adaptation in Polyak–Łojasiewicz Function Optimization. In Chapter 7, we first illustrated how learning rate tuning can be effectively applied to Polyak–Łojasiewicz functions, leveraging the continuity argument of first-order optimization algorithms as discussed by Guille-Escuret, Girotti, Goujaud, and Mitliagkas [GGGM21]. We derived a robust positive semi-definite Hessian approximation for deep network models, leading to a novel approach for dynamically adjusting the learning rate. This approach reduces reliance on preset hyperparameters, and its effectiveness has been corroborated by empirical evidence.

The approach we have taken here can also be expanded for greater efficiency and refined for other types of optimization, such as generative adversarial networks and reinforcement learning. In the following sections, we will discuss the limitations of the proposed framework and potential future research directions.

8.1 Limitations of the Approach

The Polyak–Łojasiewicz condition requires that the objective function satisfies a specific inequality, establishing a relationship between the gradient norm and the function value. While less stringent than strong convexity, this condition is still restrictive for many non-convex functions, particularly in deep learning, where loss landscapes can be highly non-convex and rugged. This suggests a need to design neural network architectures that better adhere to the Polyak–Łojasiewicz condition to gain improved convergence guarantees on their training.

Moreover, the Polyak–Łojasiewicz condition serves primarily as a theoretical construct, and there is limited empirical evidence to suggest that the loss landscapes of deep neural networks consistently adhere to this condition. Although our studies indicate that overparameterized neural networks may satisfy the Polyak–Łojasiewicz condition during training, this does not guarantee complete compliance, highlighting the need for further investigation into the practical relevance of the Polyak–Łojasiewicz condition in real-world scenarios.

Finally, while the Polyak–Łojasiewicz condition focuses on convergence to a minimum, it does not address the critical aspect of generalization in neural network training. The relationship between satisfying the Polyak–Łojasiewicz condition and achieving good generalization remains ambiguous. As previously noted, the generalization results in the context of the Polyak–Łojasiewicz condition are notably lacking, presenting a significant challenge that warrants further exploration.

8.2 Future Research Directions

Let us conclude by delineating possible research directions for an extension of this work.

Polyak–Łojasiewicz Conditioning. Future works should consider extending the theory on conditioning to finite networks, thereby including an estimation of the variance due to randomness in finite-width initialization, a reconsideration of the role of normalization (which is more useful in finite-width settings), and focus on alternative ways to bound conditioning during training, where it does not worsen as much as previously thought, as we have shown in Chapter 6.

Moreover, we think that the search for better conditioning can provide a principled justification to weight initialization strategies that make networks train better, instead of focusing on Information Propagation in the Mean Field Theory spirit of Chapter 1. Enhanced initialization could minimize poorly conditioned layers and ensure smoother gradients across layers, reducing both training time and memory requirements..

Polyak–Łojasiewicz Convergence. For what concerns convergence, the theory exposed in Chapter 6 has good predictability at later epochs, while in the initial epochs the networks perform much better than the theoretical predictions. This phenomenon has to be investigated in more depth, as there could be significant gains from having a better model of how networks behave in the first training epochs, e.g. to discard models that exhibit worse PL constants during a neural architecture search.

Polyak–Łojasiewicz Generalization. In contrast with the partial results obtained concerning convergence and conditioning, the tested Polyak–Łojasiewicz generalization theory gives vacuous bounds. We highlight the need to assess the cause of this behavior and to develop alternative approaches.

Alternatives to the Polyak–Łojasiewicz condition. We highlight how future theories should also consider variations of the PL condition like Proxy-PLness [FG21], and the weak-PL condition [CR17], since multiple useful loss functions only satisfy the latter.

Verifications of the Polyak–Łojasiewicz Theory on other Architectures. In Chapter 6 we verified the predictions of the Polyak–Łojasiewicz theory only on fully connected networks. Extensions of the presented theories to CNN, ResNets and Transformers are needed to form a more complete picture of network behavior. Those can be investigated within the same framework, but require different theoretical calculations in Lemma 21, Lemma 23 and an adaptation of the results of Agarwal, Awasthi, and Kale [AAK20] for each architecture, and are thus left to future works.

Networks as Riemannian Manifolds with Kernel Structures on the Tangent Space. Viewing neural networks as Riemannian manifolds, particularly with kernel structures defined on their tangent spaces, could reveal new insights into the geometric properties of network

training landscapes. This approach would allow for studying gradients and curvature in the manifold structure, providing a more nuanced understanding of how network parameters evolve during training and potentially identifying regions of the parameter space associated with better generalization.

Preconditioned and Conjugate Gradient Methods for Hyperparameter-Free Training. Incorporating conjugate gradient techniques with tailored momentum and preconditioning terms holds promise for developing hyperparameter-free optimization algorithms. Such methods could maintain directionality in gradient updates without extensive parameter tuning, possibly offering faster convergence and reducing manual setup.

ADAM's [Kin14] adaptive learning rate mechanism has a natural analogy to Jacobi preconditioning of SGD, effectively rescaling updates based on gradient statistics. This effect is likely enhanced by the random initialization commonly used in networks today. Formalizing this relationship could clarify the mechanics behind Adam's performance in non-convex optimization and lead to new preconditioned variants of SGD that combine the strengths of preconditioning with those of Conjugate Gradient Descent.

Part III

Appendices

Appendix A

Proofs of Deep Kernel Methods

Lemma 6 (Characterization of dependent points)**.** *The points $x_1, \dots, x_m$ are dependent if and only if the matrix $H_{ij,rs} = (K(x_i, x_r)e_j, e_s)$ is singular.*

Proof. $\boxed{\Rightarrow}$ By the definition of point dependence we have $\exists \delta_k \in Y$, which we will write as $\delta_k = \sum_s c_{ks} e_s$ where e_s are the basis vectors of Y, such that

$$\forall f \in \mathcal{H} \quad \sum_k (f(x_k), \delta_k) = 0$$

By letting $f = K(x_i, \cdot)e_j$ in the above equation we obtain $\sum_{ks} c_{ks} K_{ij,ks} = 0$, which can happen with c_{ks} not all zero iff K is singular.

$\boxed{\Leftarrow}$ We know that $\exists \delta_k \in Y$ not all zeros such that $\forall f = K(x_i, \cdot)e_j$ we have $\sum_k (f(x_k), \delta_k) = 0$. To extend this to all functions, we call $H \subseteq \mathcal{H}$ the subspace Span $\{K(x_i, \cdot)e_j\}_{i,j}$ and for every k we have $\forall f in \mathcal{H} \quad f(x_k) = (\pi_H f)(x_k)$ where π_H is the projection on the subspace H (which is closed since it is of finite dimension) and thus

$$\forall f \in \mathcal{H} \quad \sum_k (f(x_k), \delta_k) = \sum_k (\pi_H f(x_k), \delta_k) = 0$$

which constitutes the thesis. □

Definition 10 (Positive Definite Kernel, [MP05])**.** *We say that a kernel $K : X \times X \to \mathcal{L}(Y, Y)$ is positive definite if for every $x_1, \dots, x_m \in X$ and $y_1, \dots, y_m \in Y$ not all zero we have*

$$\sum_{i=1}^{m} \sum_{j=1}^{m} (K(x_i, x_j)y_i, y_j) > 0.$$

Lemma 7 (Independent points in a positive definite kernel)**.** *Let K be a positive definite Kernel and $x_1, \dots, x_m$ be points in X. Then these points are all independent provided that they are distinct.*

Proof. Let $\{x_k\}_k$ be a set of points. Since the kernel K is positive definite, the matrix $H_{ij,kl} = (K(x_i, x_k)e_j, e_l)$ is also positive definite because $\{e_i\}_{i=1,\dots m}$ is a base of Y. Let $a \in \operatorname{Ker} H$ be a

non null vector in its kernel, then we have

$$0 = a^\top (Ha) = a^\top Ha > 0$$

and we obtain a contradiction. □

Lemma 14 (Independent points in CNN Kernels)**.** *A Convolutional Kernel (Definition 3) has at least whc independent points.*

Proof. First suppose $\sigma = \mathrm{Id}$ is the identity, and consider the points x^{abc} defined by $x^{abc}_{ij\gamma} = \delta_{ia}\delta_{jb}\delta_{\gamma c}$ for $a \in \{0, \dots, w\}$, $b \in \{0, \dots, h\}$. By Lemma 6 we must prove the invertibility of the matrix

$$H_{abc\alpha\beta\gamma,def\eta\theta\nu} = \Big(K(x^{abc}, x^{def})e_{\alpha\beta\gamma}, e_{\eta\theta\nu}\Big).$$

By simple calculations we obtain

$$H_{abc\alpha\beta\gamma,def\eta\theta\nu} = \delta_{\gamma c}\delta_{\gamma f}\delta_{(a-\alpha)d}\delta_{(b-\beta)e}\delta_{\alpha\eta}\delta_{\beta\theta}\delta_{\gamma\nu}$$

which is block diagonal with respect to c, f so that we can just study the invertibility of a single block where $c = f = \gamma$.

We notice that $\forall a, b, d, e$ we have a unique pair α, β such that $H_{abc\alpha\beta\gamma,def\eta\theta\nu} \neq 0$, and precisely when $\alpha = a - d$ and $\beta = b - e$. Since $a - d \in \{-w, \dots, w\}$ and $b - e \in \{-h, \dots, h\}$ we have this pair in our indices, and thus we can calculate the determinant and prove that the matrix H is indeed invertible.

Since K is bilinear in its X entries, if we apply an invertible matrix to a set of independent points they remain independent. More formally let $A_{ijk,rst}$ be an invertible matrix. The set of points $z^{ijk} = \sum_{rst} A_{ijk,rst}x^{rst}$ is a set of independent points.

Returning now to the case of a general σ, we know that $K_\sigma(x, x') = K(\sigma(x), \sigma(x'))$ and thus we just need to prove that applying σ entrywise we are able to obtain a set of independent points. Consider then the set of points

$$z^{ab\gamma}_{ijk} = \delta_{\gamma k}\delta_{ia}\delta_{jb}\sigma(r) + (1 - \delta_{\gamma k}\delta_{ia}\delta_{jb})\sigma(c)$$

for which we have $z^{ab\gamma} = \sum_{ijk} z^{ab\gamma}_{ijk}x^{ijk}$, and obviously they are in the image of σ applied to a points with all entries c except for one which is r.

We then just need to prove invertibility of $A_{ab\gamma,ijk} = \delta_{\gamma k}\delta_{ia}\delta_{jb}\sigma(r) + (1 - \delta_{\gamma k}\delta_{ia}\delta_{jb})\sigma(c)$ which, modulo a reordering of the indices, is equivalent to $B_{i,j} = \delta_{ij}\sigma(r) + (1 - \delta_{ij})\sigma(c)$, and thus is invertible by Lemma 8 whenever $(whc - 1)\sigma(c) \neq -\sigma(r)$.

Notice that if σ crosses zero it is much easier to put $\sigma(c) = 0$, so that most calculations become straightforward, and otherwise σ is bound to be either always positive or always negative so that for any two chosen points it cannot happen that $\lambda\sigma(c) = -\sigma(r)$. □

Lemma 15 (Independent points in RNN Kernels)**.** *A Recurrent Kernel (Definition 4) has at least qT independent points.*

Proof. Consider $\Phi_u(z)_{;t}$ and notice that for fixed t and $x \in \mathbb{R}^q$, and for any $z_{;1}, \dots, z_{;t-1}$, there exist a $z_{;t}$ such that $\Phi_u(z)_{;t} = \sigma(z_t + u\Phi_u(z)_{;t-1}) = \sigma(x)$, namely $z_{;t} = x - u\Phi_u(z)_{;t-1}$. This observation practically allows us to reduce the case of recurrent networks to the case of fully connected networks, because we have now decoupled the dependence on time, i.e. for assigned $x_1, \dots, x_T \in \mathbb{R}^q$ there exists a $z \in (\mathbb{R}^q)^T$ such that $\Phi_u(z)_{;t} = \sigma(x_t)$ for each $t = 1, \dots, T$.

We may now observe that a set of independent points can be found, for fixed u, by finding points z that correpond to the set of independent points for FCN Kernels in Lemma 9. $\square$

Appendix B

Additional Details to the Tests of the Polyak–Łojasiewicz Theory

B.1 Additional Theory and Useful Lemmas

In this Appendix we report useful lemmas that complement Chapter 6 about composition of lipschitz and smoothness constants, the theory of minimization of weakly PL functions, and the ideas about the theory of conditioning.

B.1.1 Lemmas on Lipschitz and Smoothness constant

In what follows we define the smoothness and lipschitz constant of a function $f : X \to Y$ by:

$$\|f(x) - f(x')\| \leq G_f \|x - x'\| \tag{B.1}$$
$$\|\nabla f(x) - \nabla f(x')\| \leq L_f \|x - x'\| \tag{B.2}$$

Lemma 16 (Smoothness Constant and Composition)**.** *Let $f : X \to Y$ and $g : Y \to Z$, and define $h(x) = g(f(x))$. Then we have $G_h \leq G_g G_f$ and $L_h \leq G_f^2 L_g + G_g L_f$.*

Proof. For the smoothness constant we have:

$$\|\nabla h_x - \nabla h_{x'}\| = \|\nabla g_{f(x)} \nabla f_x - \nabla g_{f(x')} \nabla f_x + \nabla g_{f(x')} \nabla f_x - \nabla g_{f(x')} \nabla f_{x'}\| \tag{B.3}$$
$$\leq \|\nabla f_x\| \|\nabla g_{f(x)} - \nabla g_{f(x')}\| + \|\nabla g_{f(x')}\| \|\nabla f_x - \nabla f_{x'}\| \tag{B.4}$$
$$\leq \left(G_f^2 L_g + G_g L_f\right) \|x - x'\|. \tag{B.5}$$

The result is on the other hand well-known for the Lipschitz constant. □

Lemma 17. *The mean-squared-error loss has $G \leq 2MSE(x)$, $L \leq N$, where N is the number of dimensions. Moreover it is a 1-strongly convex function.*

Proof. Recall that for the mean-squared-error loss we have:

$$\ell(x,y) = \frac{1}{2}\sum_i (x_i - y_i)^2 \tag{B.6}$$

$$\frac{\partial \ell}{\partial x_n} = x_n - y_n \tag{B.7}$$

$$\frac{\partial^2 \ell}{\partial x_n \partial x_m} = \delta_{nm} \tag{B.8}$$

and thus we obtain that $\sum_n \left|\frac{\partial \ell}{\partial x_n}\right|^2 = 2\ell(x,y)$, $\sum_{n,m}\left|\frac{\partial^2 \ell}{\partial x_n \partial x_m}\right|^2 = N$, and it is a 1-strongly convex function because the Hessian is always positive definite and $H \geq I$. □

Lemma 18. *The cross-entropy loss has $L, G \leq 2$ and is a weakly convex function.*

Proof. Recall that cross-entropy satifyies

$$\ell(x,k) = -x_k + \log\left(\sum_j e^{x_j}\right) \tag{B.9}$$

$$\frac{\partial \ell}{\partial x_n}(x,k) = -\delta_{n,k} + \frac{e^{x_n}}{\sum_j e^{x_j}} \tag{B.10}$$

$$\frac{\partial^2 \ell}{\partial x_m \partial x_n}(x,k) = \frac{\delta_{mn} e^{x_n}\left(\sum_j e^{x_j}\right) - e^{x_m}e^{x_n}}{\left(\sum_j e^{x_j}\right)^2} \tag{B.11}$$

and one easily obtains that $\sum_n \left|\frac{\partial \ell}{\partial x_n}(x,k)\right|^2 \leq 2$, $\sum_{n,m}\left|\frac{\partial^2 \ell}{\partial x_m \partial x_n}(x,k)\right|^2 \leq 2$. Weak convexity stems from the positivity of the hessian matrix: let us call $t_i = \frac{e^{x_i}}{\sum_j e^{x_j}}$, then we have seen that $H_{nm} = \frac{\partial^2 \ell}{\partial x_m \partial x_n}(x,k) = \delta_{mn}t_n - t_n t_m$, and thus we have

$$u^T H u = \sum_{ij} H_{ij}u_i u_j = \sum_i t_i u_i^2 - \sum_{ij} t_i t_j u_i u_j = \sum_i t_i u_i^2 - \left(\sum_j t_j u_j\right)^2 \geq 0 \tag{B.12}$$

by Cauchy-Schwartz inequality applied to $\sqrt{t_i}$ and $\sqrt{t_i}u_i$, since $\sum_i t_i = 1$. □

Definition 11 (Matrix Norm)**.** *Let A be a square matrix and consider the norm induced by the p-vector norm:*

$$\|A\|_{p\to p} := \sup_{v\in V}\frac{\|Av\|_p}{\|v\|_p}. \tag{B.13}$$

Definition 12 (Spectral Radius)**.** *Let A be an $n\times n$ square matrix with eigenvalues $\lambda_1, \ldots, \lambda_n \in \mathbb{C}$. We define its spectral radius $\rho(A)$ by*

$$\rho(A) := \max\left\{,_{i=1}^n |\lambda_i|\right\} = \max\{|\lambda_1|, \ldots, |\lambda_n|\} \tag{B.14}$$

as the maximum absolute value of its eigenvalues.

Lemma 19 (Matrix Norm Facts)**.** *In what follows let $p, q \in \overline{\mathbb{R}}$ be an Hölder conjugate pair, i.e. $\frac{1}{p} + \frac{1}{q} = 1$. Then the matrix norm satisfies the following:*

1. $\|A\|_{p\to p} = \left\|A^T\right\|_{q\to q}$
2. $\|AB\|_{p\to p} \leq \|A\|_{p\to p}\|B\|_{p\to p}$
3. $\rho(A) \leq \|A\|_{p\to p}$
4. $\|A\|_{2\to 2}^2 = \rho(A^T A)$
5. $\|A\|_{\infty\to\infty} = \max_i \left(\sum_j |A_{ij}|\right)$

Proof. Omitted because they are known facts. For a proof see Bhatia [Bha13]. □

Lemma 20 (Interpolation for matrix norms)**.** *Let $p, q \in \overline{\mathbb{R}}$ be an Hölder conjugated pair, i.e. $\frac{1}{p} + \frac{1}{q} = 1$. Then it holds:*

$$\|A\|_{2\to 2}^2 \leq \|A\|_{p\to p}\|A\|_{q\to q}. \tag{B.15}$$

Proof. Using the facts enumerated in Lemma 19 we get

$$\|A\|_{2\to 2}^2 = \rho(A^T A) \leq \left\|A^T A\right\|_{p\to p} \leq \left\|A^T\right\|_{p\to p}\|A\|_{p\to p} = \|A\|_{q\to q}\|A\|_{p\to p}. \tag{B.16}$$

In particular we notice that $\|A\|_{2\to 2}^2 \leq \|A\|_{1\to 1}\|A\|_{\infty\to\infty}$. □

Lemma 21 (Lipschitz and Smoothness constant for a layer)**.** *Let us consider a FCN layer $f(x) = M\phi(x)$ where the activation function ϕ is applied entrywise. Suppose that it receives inputs whose coordinates are always inside the interval I; then its Lipschits and Smoothness constants are:*

$$G_f = \sigma_{max}(M) \cdot \max_{t\in I} \left|\phi'(t)\right| \tag{B.17}$$

$$L_f = \max_{ij} |M_{ij}| \cdot \max_{t\in I} \left|\phi''(t)\right|. \tag{B.18}$$

Proof. It is easy to calculate the layer Lipschitz constant with respect to euclidean norm since it is a composition of a linear layer M and of the non-linear function ϕ, and we obtain the desired result.

The calculation of the Smoothness constant is more involved, and we start by writing out fully the layer-defining equations:

$$f(x)_i = \sum_j M_{ij}\phi(x_j) \tag{B.19}$$

$$\frac{\partial f(x)_i}{\partial x_j} = M_{ij}\phi'(x_j) \tag{B.20}$$

$$\frac{\partial^2 f(x)_i}{\partial x_j\, \partial x_j} = \delta_{jk} M_{ij}\phi''(x_j) \tag{B.21}$$

At this point we can make use of Equation (B.15) to obtain the desired result

$$\left\|\nabla^2 f(x)_i\right\|_{2\to 2} = \sqrt{\|\nabla^2 f(x)_i\|_{1\to 1}\|\nabla^2 f(x)_i\|_{\infty\to\infty}} \tag{B.22}$$

$$= \left\|\nabla^2 f(x)_i\right\| 1 \to 1 \tag{B.23}$$

$$= \max_j |M_{ij}| \cdot \max_{t\in I} \left|\phi''(t)\right| \tag{B.24}$$

where $\left\|\nabla^2 f(x)_i\right\|_{1\to 1} = \left\|\nabla^2 f(x)_i\right\|_{\infty\to\infty}$ because $\nabla^2 f(x)_i$ is symmetric. □

The above lemmas provide a way to compute the worst expected eigenvalue when the network weights are trained, using Equation (4.11), since we can now use Lemma 21 to compute the Lipschitz and Smoothness constants of the layer and Lemma 16 iteratively to bound those of the whole network.

Being based on iterative estimates, it is obvious that the obtained quantities could be off from the real Lipschitz and Smoothness constant by quite a bit, and thus also the obtained eigenvalue and generalization estimates obtained. We leave the question of how to improve those bounds to future works.

B.1.2 Weakly PL Functions

In Chapter 6 we exposed the theory about strongly PL functions for ease of exposition; in this Section we instead present the theory of weak PL functions which originated in the work of Csiba and Richtárik [CR17], and which is needed to analyze the results of the experiments on the cross-entropy loss. We expose the main results about weak PL functions, and refer the interested reader to the cited work.

Definition 13 (Weakly PL function, [CR17])**.** *A function $f : X \to \mathbb{R}$ is weakly μ-PL on $\Omega \subseteq X$ if there $\exists x^* \in X^*$ the set of global minimizer such that*

$$\forall x, y \in \Omega \quad \|\nabla f(x)\|\|x - x^*\| \geq \sqrt{\mu}(f(x) - f^*) \tag{B.25}$$

where $f^* := \min_{x\in\Omega} f(x)$.

It can be easily observed that every weakly convex function is weakly PL with $\mu = 1$. Moreover we have the following lemma on the decrease of the objective during the optimization process:

Lemma 22 (Gradient Descent on Weakly PL functions, [CR17, Lemma 3])**.** *Given an L-smooth, weakly μ-PL function $f : X \to \mathbb{R}$ and chosen an initial point x_0, we let the sequence of iterates evolve according to the rule:*

$$x_{k+1} = x_k - \frac{1}{L}\nabla f(x_k). \tag{B.26}$$

Then the optimality gap decreases following the formula

$$f(x_{k+1}) - f^* \leq \left(1 - \frac{\mu(f(x_k) - f^*)}{2L\|x_k - x_*\|^2}\right)(f(x_k) - f^*). \tag{B.27}$$

Specificities about how the quantities in Equation (B.27) have been measured are discussed in Section 6.2.

B.1.3 Conditioning Ideas

In this Section we detail the ideas in the work of Agarwal, Awasthi, and Kale [AAK20] that allow us to obtain bounds on the conditioning of the various network layers.

In what follows, we will assume that the inputs $(x_i)_{i=1,\dots,n}$ to our neural network are all of unitary norm and their pairwise product satisfies $|x_i \cdot x_j| \leq 1 - \delta$. We being by introducing the definition of the dual activation function, due to Daniely, Frostig, and Singer [DFS16].

Definition 14 (Dual Activation, [DFS16])**.** *Given an activation function σ, we define the dual activation function $\hat{\sigma} : [-1, 1] \to \mathbb{R}$ by*

$$\hat{\sigma}(\rho) = \mathbb{E}_{(X,Y)\sim\Sigma_\rho}[\sigma(X)\sigma(Y)] \qquad \text{where } \Sigma_\rho = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix}. \tag{B.28}$$

Lemma 23 (Dual Activation and Infinite-Width Networks, [DFS16])**.** *$\hat{\sigma}$ corresponds to the function that is calculated by freshly initialized networks layers of infinite-width on a given pair of inputs. In other words, define by*

$$\tilde{K}^m_{ij} := \lim_{t\to\infty} K^{m;\theta}_{ij} \tag{B.29}$$

the NTK deterministic limit, where t is the dimension of the space Θ. Then we have that

$$\tilde{K}^{m+1} = \hat{\sigma}\left(\tilde{K}^m\right) \tag{B.30}$$

where $\hat{\sigma}$ is applied entrywise.

Lemma 23 gives us an idea about what to expect when depth is increased in a wide neural network. Infact, when the space of parameters is so large that $K^{m;\theta}_{ij} \simeq \tilde{K}^m_{ij}$, Lemma 23 gives us a way to know the entries of the K matrix and thus to bound its conditioning.

Eigenvalue Bounding Sketch. We begin by exposing a way to give bounds on the smallest eigenvalue for iterative applications of the dual activation to a given positive definite matrix:

Lemma 24 (Eigenvalue lower bound lemma [AAK20][Lemma 23])**.** *Let $H \in \mathbb{R}^{n\times n}$ a positive-definite matrix, i.e. $H \geq \delta I_n$ and all values $= 1$ on the diagonal. Let $f : \mathbb{R} \to \mathbb{R}$ be an analytic function whose series expansion in zero has only positive coefficients, and let $f[H]$ be the application of f to each entry of the matrix. Then $f[H] \geq (f(1) - f(1 - \delta))I_n$.*

By our assumptions, $K^{0;\theta}$ is positive definite with ones on the diagonal, and we can verify that for all commonly considered activation functions, $\hat{\sigma}$ is analytic on the real line with non-negative coefficients in its power series expansion, so that Lemma 24 above applies.

On the other hand, we can bound the highest eigenvalue using Gershgorin circle theorems if we know how to bound the off-diagonal entries of the K matrix. To this aim we introduce the definition of distortion, that essentially quantifies how much $\hat{\sigma}$ acts as a contraction when its inputs are a bit distant from one.

Definition 15 (Maximal Distortion of Dual Activation, [AAK20])**.** *Given a dual activation function $\hat{\sigma}$, we define its distortion at level δ as:*

$$M_\delta := \sup\{\hat{\sigma}(\rho) \mid |\rho| \leq 1 - \delta\}. \tag{B.31}$$

As the reader can imagine, iteratively applying $\hat{\sigma}$ to the set of pairwise products of inputs

$D_{ij} = \langle x_i, x_j \rangle$ leaves the diagonal terms unchanged since $\hat{\sigma}(1) = 1$, while it increasingly shrinks the off-diagonal terms to zero, in an amount that is quantified by its distortion.

It is then clear that any matrix chain such that $S^{m+1} = \hat{\sigma}(S^m)$ essentially converges toward the identity matrix. Agarwal, Awasthi, and Kale [AAK20] are able to extract the convergence rates from such reasonments and thereby establish the stronger result of exponential convergence of both the $G^{m;\theta}$ and $K^{m;\theta}$ matrices to conditioning of one. For more details we refer the reader to the already cited works [AAK20; DFS16].

Bibliography

[AAK20] Naman Agarwal, Pranjal Awasthi, and Satyen Kale. "A Deep Conditioning Treatment of Neural Networks". In: *arXiv preprint arXiv:2002.01523* (2020).

[Abi+18] Oludare Isaac Abiodun et al. "State-of-the-art in artificial neural network applications: A survey". In: *Heliyon* 4 (2018). DOI: 10.1016/j.heliyon.2018.e00938.

[ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. "Wasserstein generative adversarial networks". In: *International conference on machine learning*. PMLR. 2017, pp. 214–223.

[ALS19] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. "A convergence theory for deep learning via over-parameterization". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 242–252.

[Aro+19] Sanjeev Arora et al. "On exact computation with an infinitely wide neural net". In: *Advances in neural information processing systems* 32 (2019).

[BB22] Dario Balboni and Davide Bacciu. "An Empirical Verification of Wide Networks Theory." In: *BMVC*. 2022, p. 517.

[BB24] Dario Balboni and Davide Bacciu. "ADLER–An efficient Hessian-based strategy for adaptive learning rate". In: (2024).

[BE02] Olivier Bousquet and André Elisseeff. "Stability and generalization". In: *The Journal of Machine Learning Research* 2 (2002), pp. 499–526.

[Ber+19] Christopher Berner et al. "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680* (2019).

[BGBL23] Aleksandr Beznosikov, Eduard Gorbunov, Hugo Berard, and Nicolas Loizou. "Stochastic gradient descent-ascent: Unified theory and new efficient methods". In: *International conference on artificial intelligence and statistics*. PMLR. 2023, pp. 172–235.

[Bha13] Rajendra Bhatia. *Matrix analysis*. Vol. 169. Springer Science & Business Media, 2013.

[BHHH74] E Berndt, BH Hall, RE Hall, and JA Hausman. "Estimation and inference in nonlinear structural models". In: *Annals of economic and social measurement* 3 (1974), pp. 653–666.

[BLLT20] Peter L Bartlett, Philip M Long, Gábor Lugosi, and Alexander Tsigler. "Benign overfitting in linear regression". In: *Proceedings of the National Academy of Sciences* 117.48 (2020), pp. 30063–30070.

[BM02] Peter L Bartlett and Shahar Mendelson. "Rademacher and Gaussian complexities: Risk bounds and structural results". In: *Journal of Machine Learning Research* 3.Nov (2002), pp. 463–482.

[BMM18] Mikhail Belkin, Siyuan Ma, and Soumik Mandal. "To understand deep learning we need to understand kernel learning". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 541–549.

[BMMM] Thomas Bachlechner, Bodhisattwa Prasad Majumder, Huanru Henry Mao, and Julian McAuley. "Deepformers: Training Very Deep Transformers via Dynamical Isometry". In: ().

[Bot+20] Matthew Botvinick et al. "Deep reinforcement learning and its neuroscientific implications". In: *Neuron* 107.4 (2020), pp. 603–616.

[Bre01] Leo Breiman. "Random forests". In: *Machine learning* 45 (2001), pp. 5–32.

[Bre95] Leo Breiman. "Reflections after refereeing papers for NIPS". In: *The Mathematics of Generalization*. CRC Press, 1995, pp. 11–15.

[BRG19] Bastian Bohn, Christian Rieger, and Michael Griebel. "A Representer Theorem for Deep Kernel Learning." In: *J. Mach. Learn. Res.* 20 (2019), pp. 64–1.

[Bro70a] Charles G Broyden. "The convergence of a class of double-rank minimization algorithms: 2. The new algorithm". In: *IMA journal of applied mathematics* 6.3 (1970), pp. 222–231.

[Bro70b] Charles George Broyden. "The convergence of a class of double-rank minimization algorithms 1. general considerations". In: *IMA Journal of Applied Mathematics* 6.1 (1970), pp. 76–90.

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[CBR19] Carlo Ciliberto, Francis Bach, and Alessandro Rudi. "Localized structured prediction". In: *Advances in Neural Information Processing Systems* 32 (2019).

[CGT91] Andrew R Conn, Nicholas IM Gould, and Philippe Toint. "A globally convergent augmented Lagrangian algorithm for optimization with general constraints and simple bounds". In: *SIAM Journal on Numerical Analysis* 28.2 (1991), pp. 545–572.

[CH67] Thomas Cover and Peter Hart. "Nearest neighbor pattern classification". In: *IEEE transactions on information theory* 13.1 (1967), pp. 21–27.

[CMS12] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3642–3649.

[Cor95] Corinna Cortes. "Support-Vector Networks". In: *Machine Learning* (1995).

[CP18] Zachary Charles and Dimitris Papailiopoulos. "Stability and generalization of learning algorithms that converge to global optima". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 745–754.

[CPS18] Minmin Chen, Jeffrey Pennington, and Samuel S Schoenholz. "Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks". In: *arXiv preprint arXiv:1806.05394* (2018).

[CR17] Dominik Csiba and Peter Richtárik. "Global Convergence of Arbitrary-Block Gradient Methods for Generalized Polyak-Łojasiewicz Functions". In: *arXiv preprint arXiv:1709.03014* (2017).

[CYL22] Lesi Chen, Boyuan Yao, and Luo Luo. "Faster stochastic algorithms for minimax optimization under polyak-{\L} ojasiewicz condition". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 13921–13932.

[Dav91] William C Davidon. "Variable metric method for minimization". In: *SIAM Journal on optimization* 1.1 (1991), pp. 1–17.

[DFS16] Amit Daniely, Roy Frostig, and Yoram Singer. "Toward deeper understanding of neural networks: The power of initialization and a dual view on expressivity". In: *arXiv preprint arXiv:1602.05897* (2016).

[Dos+20] Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale". In: *arXiv preprint arXiv:2010.11929* (2020).

[Du+19] Simon Du et al. "Gradient descent finds global minima of deep neural networks". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1675–1685.

[DZPS18] Simon S Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. "Gradient descent provably optimizes over-parameterized neural networks". In: *arXiv preprint arXiv:1810.02054* (2018).

[FG21] Spencer Frei and Quanquan Gu. "Proxy Convexity: A Unified Framework for the Analysis of Neural Networks Trained by Gradient Descent". In: *arXiv preprint arXiv:2106.13792* (2021).

[Fle70] Roger Fletcher. "A new approach to variable metric algorithms". In: *The computer journal* 13.3 (1970), pp. 317–322.

[FP63] Roger Fletcher and Michael JD Powell. "A rapidly convergent descent method for minimization". In: *The computer journal* 6.2 (1963), pp. 163–168.

[FR64] Reeves Fletcher and Colin M Reeves. "Function minimization by conjugate gradients". In: *The computer journal* 7.2 (1964), pp. 149–154.

[Ger31] Semyon Aranovich Gershgorin. "Über die Abgrenzung der Eigenwerte einer Matrix". In: *Izvestiya Rossiiskoi Akademii Nauk, Seriya Matematicheskaya* 6 (1931), pp. 749–754.

[GGGM21] Charles Guille-Escuret, Manuela Girotti, Baptiste Goujaud, and Ioannis Mitliagkas. "A Study of Condition Numbers for First-Order Optimization". In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 1261–1269.

[Gol70] Donald Goldfarb. "A family of variable-metric methods derived by variational means". In: *Mathematics of computation* 24.109 (1970), pp. 23–26.

[Goo+14] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014), pp. 2672–2680.

[Gra+18] Katja Grace et al. "When will AI exceed human performance? Evidence from AI experts". In: *Journal of Artificial Intelligence Research* 62 (2018), pp. 729–754.

[GY22] Eugene Golikov and Greg Yang. “Non-gaussian tensor programs”. In: *Advances in Neural Information Processing Systems*. 2022.

[Haz+16] Elad Hazan et al. “Introduction to online convex optimization”. In: *Foundations and Trends® in Optimization* 2.3-4 (2016), pp. 157–325.

[He+23] Bobby He et al. “Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation”. In: *arXiv preprint arXiv:2302.10322* (2023).

[Her12] Suzana Herculano-Houzel. “The remarkable, yet not extraordinary, human brain as a scaled-up primate brain and its associated cost”. In: *Proceedings of the National Academy of Sciences* 109.supplement_1 (2012), pp. 10661–10668.

[HK70] Arthur E Hoerl and Robert W Kennard. “Ridge regression: Biased estimation for nonorthogonal problems”. In: *Technometrics* 12.1 (1970), pp. 55–67.

[Hot33] Harold Hotelling. “Analysis of a complex of statistical variables into principal components.” In: *Journal of educational psychology* 24.6 (1933), p. 417.

[Hou06] Alston S Householder. *Principles of numerical analysis*. Courier Corporation, 2006.

[HS+52] Magnus Rudolph Hestenes, Eduard Stiefel, et al. *Methods of conjugate gradients for solving linear systems*. Vol. 49. 1. NBS Washington, DC, 1952.

[Hut02] Marcus Hutter. “The fastest and shortest algorithm for all well-defined problems”. In: *International Journal of Foundations of Computer Science* 13.03 (2002), pp. 431–443.

[HZ06] William W Hager and Hongchao Zhang. “A survey of nonlinear conjugate gradient methods”. In: *Pacific journal of Optimization* 2.1 (2006), pp. 35–58.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 1026–1034.

[ITB16] Daniel Jiwoong Im, Michael Tao, and K. Branson. “An Empirical Analysis of Deep Network Loss Surfaces”. In: *ArXiv* abs/1612.04010 (2016).

[JGH18] Arthur Jacot, Franck Gabriel, and Clément Hongler. “Neural tangent kernel: Convergence and generalization in neural networks”. In: *Advances in neural information processing systems*. 2018, pp. 8571–8580.

[Kap+20] Jared Kaplan et al. “Scaling laws for neural language models”. In: *arXiv preprint arXiv:2001.08361* (2020).

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[Kin14] Diederik P Kingma. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

[KNS16] Hamed Karimi, Julie Nutini, and Mark Schmidt. “Linear convergence of gradient and proximal-gradient methods under the polyak-łojasiewicz condition”. In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2016, pp. 795–811.

[LDS18] Dawei Li, Tian Ding, and Ruoyu Sun. "On the benefit of width for neural networks: Disappearance of bad basins". In: *arXiv* (2018), arXiv–1812.

[Lee+19] Jaehoon Lee et al. "Wide neural networks of any depth evolve as linear models under gradient descent". In: *arXiv preprint arXiv:1902.06720* (2019).

[Lee+20] Jaehoon Lee et al. "Finite versus infinite neural networks: an empirical study". In: *arXiv preprint arXiv:2007.15801* (2020).

[Lit+17] Geert Litjens et al. "A survey on deep learning in medical image analysis". In: *Medical image analysis* 42 (2017), pp. 60–88.

[Liu+21] Risheng Liu et al. "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12 (2021), pp. 10045–10067.

[LN89] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[LVL22] Alexandra Sasha Luccioni, Sylvain Viguier, and Anne-Laure Ligozat. "Estimating the Carbon Footprint of BLOOM, a 176B Parameter Language Model". In: *arXiv preprint arXiv:2211.02001* (2022).

[LZB20] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. "Toward a theory of optimization for over-parameterized systems of non-linear equations: the lessons of deep learning". In: *arXiv preprint arXiv:2003.00307* (2020).

[LZB22] Chaoyue Liu, Libin Zhu, and Mikhail Belkin. "Loss landscapes and optimization in over-parameterized non-linear systems and neural networks". In: *Applied and Computational Harmonic Analysis* 59 (2022), pp. 85–116.

[Ma12] Haifeng Ma. "Construction of some generalized inverses of operators between Banach spaces and their selections, perturbations and applications". In: (2012).

[MC89] Michael McCloskey and Neal J Cohen. "Catastrophic interference in connectionist networks: The sequential learning problem". In: *Psychology of learning and motivation*. Vol. 24. Elsevier, 1989, pp. 109–165.

[MCRR20] Giacomo Meanti, Luigi Carratino, Lorenzo Rosasco, and Alessandro Rudi. "Kernel methods through the roof: handling billions of points efficiently". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 14410–14422.

[Mit15] Boris Mityagin. "The zero set of a real analytic function". In: *arXiv preprint arXiv:1512.07276* (2015).

[MP05] Charles A Micchelli and Massimiliano Pontil. "On learning vector-valued functions". In: *Neural computation* 17.1 (2005), pp. 177–204.

[MWHN18] Iacopo Masi, Yue Wu, Tal Hassner, and Prem Natarajan. "Deep face recognition: A survey". In: *2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*. IEEE. 2018, pp. 471–478.

[Nas50] John F Nash Jr. "Equilibrium points in n-person games". In: *Proceedings of the national academy of sciences* 36.1 (1950), pp. 48–49.

[Nes+18] Yurii Nesterov et al. *Lectures on convex optimization*. Vol. 137. Springer, 2018.

[Nes13] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*. Vol. 87. Springer Science & Business Media, 2013.

[Nes83] Y Nesterov. "A method of solving a convex programming problem with convergence rate $O(1/k^2)$". In: *Proceedings of the USSR Academy of Sciences* 269 (1983), p. 3.

[NH18] Quynh Nguyen and Matthias Hein. "Optimization landscape and expressivity of deep cnns". In: *International conference on machine learning*. PMLR. 2018, pp. 3730–3739.

[OS20] Samet Oymak and Mahdi Soltanolkotabi. "Toward Moderate Overparameterization: Global Convergence Guarantees for Training Shallow Neural Networks". In: *IEEE Journal on Selected Areas in Information Theory* 1.1 (2020), pp. 84–105.

[Pol63] Boris Teodorovich Polyak. "Gradient methods for minimizing functionals". In: *Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki* 3.4 (1963), pp. 643–653.

[Poo+16] Ben Poole et al. "Exponential expressivity in deep neural networks through transient chaos". In: *Advances in neural information processing systems* 29 (2016).

[PSG17] Jeffrey Pennington, Samuel Schoenholz, and Surya Ganguli. "Resurrecting the sigmoid in deep learning through dynamical isometry: theory and practice". In: *Advances in neural information processing systems* 30 (2017).

[RCR15] Alessandro Rudi, Raffaello Camoriano, and Lorenzo Rosasco. "Less is more: Nyström computational regularization". In: *Advances in Neural Information Processing Systems* 28 (2015).

[RCR17] Alessandro Rudi, Luigi Carratino, and Lorenzo Rosasco. "Falkon: An optimal large scale kernel method". In: *Advances in neural information processing systems* 30 (2017).

[Red+16] Sashank J Reddi et al. "Stochastic variance reduction for nonconvex optimization". In: *International conference on machine learning*. PMLR. 2016, pp. 314–323.

[RHW86] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536.

[RV09] Mark Rudelson and Roman Vershynin. "Smallest singular value of a random rectangular matrix". In: *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences* 62.12 (2009), pp. 1707–1739.

[RYH22] Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*. Cambridge University Press Cambridge, MA, USA, 2022.

[SA19] Florian Schäfer and Anima Anandkumar. "Competitive gradient descent". In: *Advances in Neural Information Processing Systems* 32 (2019).

[SGGS17] Samuel S Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. "Deep Information Propagation". In: (2017).

[Sha70] David F Shanno. "Conditioning of quasi-Newton methods for function minimization". In: *Mathematics of computation* 24.111 (1970), pp. 647–656.

[Sil+16] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[SMG13] Andrew M Saxe, James L McClelland, and Surya Ganguli. "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks". In: *arXiv preprint arXiv:1312.6120* (2013).

[Smi17] Leslie N Smith. "Cyclical learning rates for training neural networks". In: *2017 IEEE winter conference on applications of computer vision (WACV)*. IEEE. 2017, pp. 464–472.

[SNW11] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2011.

[SS18] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2018.

[SSM98] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. "Nonlinear component analysis as a kernel eigenvalue problem". In: *Neural computation* 10.5 (1998), pp. 1299–1319.

[SSPS21] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. "Reward is enough". In: *Artificial Intelligence* 299 (2021), p. 103535.

[Sut18] Richard S Sutton. "Reinforcement learning: An introduction". In: *A Bradford Book* (2018).

[Tao12] Terence Tao. *Topics in random matrix theory*. Vol. 132. American Mathematical Soc., 2012.

[Tav+18] A. Tavanaei et al. "Deep Learning in Spiking Neural Networks". In: *Neural networks : the official journal of the International Neural Network Society* 111 (2018), pp. 47–63. DOI: 10.1016/j.neunet.2018.12.002.

[TB20] Alexander Tsigler and Peter L Bartlett. "Benign overfitting in ridge regression". In: *arXiv preprint arXiv:2009.14286* (2020).

[Van+10] Jur Van Den Berg et al. "Superhuman performance of surgical tasks by robots using iterative learning from human-guided demonstrations". In: *2010 IEEE International Conference on Robotics and Automation*. IEEE. 2010, pp. 2074–2081.

[Vap95] Vladimir Vapnik. "Support-vector networks". In: *Machine learning* 20 (1995), pp. 273–297.

[Vas+17] Ashish Vaswani et al. "Attention is all you need". In: *arXiv preprint arXiv:1706.03762* (2017).

[WR06] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*. Vol. 2. 3. MIT press Cambridge, MA, 2006.

[Xia+18] Lechao Xiao et al. "Dynamical isometry and a mean field theory of cnns: How to train 10,000-layer vanilla convolutional neural networks". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 5393–5402.

[Yan+20] Zitong Yang et al. "Rethinking bias-variance trade-off for generalization of neural networks". In: *International Conference on Machine Learning*. PMLR. 2020, pp. 10767–10777.

[Yan+22] Greg Yang et al. "Tensor programs v: Tuning large neural networks via zero-shot hyperparameter transfer". In: *arXiv preprint arXiv:2203.03466* (2022).

[Yan19a] Greg Yang. "Scaling limits of wide neural networks with weight sharing: Gaussian process behavior, gradient independence, and neural tangent kernel derivation". In: *arXiv preprint arXiv:1902.04760* (2019).

[Yan19b] Greg Yang. "Wide feedforward or recurrent neural networks of any architecture are gaussian processes". In: *Advances in Neural Information Processing Systems* 32 (2019).

[Yan20a] Greg Yang. "Tensor programs ii: Neural tangent kernel for any architecture". In: *arXiv preprint arXiv:2006.14548* (2020).

[Yan20b] Greg Yang. "Tensor programs ii: Neural tangent kernel for any architecture". In: *arXiv preprint arXiv:2006.14548* (2020).

[Yan20c] Greg Yang. "Tensor programs iii: Neural matrix laws". In: *arXiv preprint arXiv:2009.10685* (2020).

[YH21] Greg Yang and Edward J Hu. "Tensor programs iv: Feature learning in infinite-width neural networks". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11727–11737.

[YKH20] Junchi Yang, Negar Kiyavash, and Niao He. "Global convergence and variance-reduced optimization for a class of nonconvex-nonconcave minimax problems". In: *arXiv preprint arXiv:2002.09621* (2020).

[YL21] Greg Yang and Etai Littwin. "Tensor programs iib: Architectural universality of neural tangent kernel training dynamics". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 11762–11772.

[Yua00] Ya-xiang Yuan. "A review of trust region algorithms for optimization". In: *Iciam*. Vol. 99. 1. 2000, pp. 271–282.

[Zad19] A. Zador. "A critique of pure learning and what artificial neural networks can learn from animal brains". In: *Nature Communications* 10 (2019). DOI: 10.1038/s41467-019-11786-6.

[Zha+16] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *arXiv preprint arXiv:1611.03530* (2016).

[Zha+21] Chiyuan Zhang et al. "Understanding deep learning (still) requires rethinking generalization". In: *Communications of the ACM* 64.3 (2021), pp. 107–115.