Deep learning theory (DRAFT)

Matus Telgarsky

August 24, 2022
Contents

1 Introduction 5
   1.1 Scope of this book ............................................. 6
   1.2 Status and purpose of deep learning theory .................. 8

2 Shallow networks and continuous functions 9

3 Initialization and overparameterization 11

4 Benefits of other architectures 13

I Optimization 15

5 Smooth NTK 17

6 Strongly convex NTK 19

7 Margins and feature learning 21

II Generalization 23

III Other topics 25

8 Distribution modeling 27

A Technical background 31
   A.1 Convexity ................................................... 31
   A.2 Miscellaneous inequalities ............................... 31
   A.3 Probability ................................................ 31
Chapter 1

Introduction

In practice, a family of functions $\mathcal{F}$ is called \textit{neural/deep network} or \textit{neural/deep network architecture} if it has the following characteristics.

1. \textbf{$\mathcal{F}$ is parameterized by $p$ real numbers:} $\mathcal{F} := \{ x \mapsto F(x; w) : w \in \mathbb{R}^p \}$ for some fixed function $F$ and fixed parameter dimension $p$ and input space $x \in \mathcal{X}$;

2. \textbf{Bounded computation:} the number of elementary operations on real numbers needed to compute $\mathcal{F}$ is uniformly bounded over $x$ and $w$.

3. \textbf{Efficient hardware implementation:} standard choices for the construction of $F$ co-evolve with hardware developments.

4. \textbf{Convenient programming interface:} libraries so simple that one hour suffices, with no prior library or even ML experience.

5. \textbf{Amenable to gradient descent:} a single good parameter choice $w \in \mathbb{R}^p$ can be found easily via a first-order descent method.

Items 1 and 2 are usually treated as the standard definition after some restrictions to “elementary operations”. These conditions are satisfied when $F$ is defined by a directed acyclic graph with a single source corresponding to input $x$, a single sink corresponding to the output, and graph nodes perform a bounded amount of computation parametrized by $w \in \mathbb{R}^p$; this graph-based perspective motivates the word “network”. Equivalently, code written in a standard library such as \texttt{pytorch} meets these definitions if there are no loops, no unbounded recursion, or equivalents.

It should also be noted that Items 1 and 2 are not sufficient to characterize deep networks, as they are also satisfied by polynomial classifiers, SVMs, and many other choices. This suggests the importance of further items, specifically Items 3 to 5 which aim to capture what makes deep networks different and successful. This text will not address Items 3 and 4 but will Item 5 as a central concern.

The magic of deep networks is that if we have some data and a corresponding performance measure $\mathcal{R}(w)$ of our selected parameters $w \in \mathbb{R}^p$, then by further tuning $w$ with a simple procedure such as gradient descent (Item 5 above), we can induce $\mathcal{R}(w)$ to also be small, where $\mathcal{R}$ corresponds to our performance on data we have not seen. Here are two examples.
Example 1.0.1 (Prompt-based image generation). The popular software DALL-E 2 takes an english sentence (the \( x \in X \)), passes it through a complicated \( x \mapsto F(x; w) \) where \( w \in \mathbb{R}^p \) with \( p \approx 2^{32} \), and outputs a 1024 \( \times \) 1024 image with 3 color channels [Ramesh et al., 2022]. Even though there are almost \( 10^9 \) training examples, this seemingly large number is dwarfed by the magnitude of joint input and output spaces: we have a vanishingly sparse cover of all reasonable input/output pairs. But of even greater concern, it is not clear how to even define \( \hat{R} \) and \( R \), let alone minimize \( \hat{R} \) and show that \( R \) is also small, and moreover analyze the full procedure.

Example 1.0.2 (Protein folding). AlphaFold uses similarly complicated \( F \) and huge \( w \in \mathbb{R}^p \) to convert the amino acid sequence of a protein (the \( x \in X \)) into a three-dimensional description of the protein (i.e., how the protein is “folded”). [Jumper et al., 2021]. In this case, \( \hat{R} \) and \( R \) can be defined in a variety of reasonable ways, but are very costly: sometimes it requires a large amount of equipment and scientific expertise to produce a few accurate input/output pairs. Due to this expense, the set of labeled examples is not only small, it is biased (e.g., since analyzing certain proteins seems more beneficial to humans), meaning it is not a uniform sampling of the collection of proteins across all species. Despite this, AlphaFold happily outputs folding information for all amino acid sequences it is fed, the shock being its accuracy on \( R \) (which can then be checked) despite the seen and unseen data having different structure.

1.1 Scope of this book

The tools in this book are from being able to analyze Examples 1.0.1 and 1.0.2; the focus is summarized as follows.

1. This book will primarily analyze the simple feed-forward architectures in Definition 1.1.1. A few other architectures will be discussed in (todo).

2. The focus is on arguing why \( R \) can be made small, via a decomposition of \( R \) which has small \( \hat{R} \) as a subproblem, given below in eq. (1.1.2).

3. The goal is to present mathematical material is tools with short proofs and flexible usage, and hopefully to de-emphasize specific dogma. This also means some results are omitted simply because the author could not produce a short proof.

4. Wherever possible, bounds are presented with data- and algorithm-sensitive quantities.

5. Bridging old and new: deep networks have been investigated many times in waves, each time bringing new ideas, but often they are not connected across waves. An explicit example is (fill in the near initialization stuff).

Standard feedforward architectures are as follows.

Definition 1.1.1 (Feedforward networks.). A feedforward architecture has \( w = ((W_1, \ldots, W_L), (b_1, \ldots, b_L)) \), a tuple of matrices and vectors, the weights \( (W_i)_{i=1}^L \) and biases \( (b_i)_{i=1}^L \), and computes

\[
F(x; w) := \sigma_L(W_L \sigma_{L-1}(W_{L-1} \cdots \sigma_1(W_1 x + b_1) \cdots + b_{L-1}) + b_L),
\]

where \( \sigma_l \) is the activation function of the \( l \)-th layer.
where \((\sigma_i)_{i=1}^L\) are fixed nonlinear functions (also called activations or transfer functions). A common example is for each \(\sigma_i\) to apply a single function coordinate-wise, popular examples being the ReLU \(z \mapsto \max\{0, z\}\) and the sigmoid \(z \mapsto 1/(1 + \exp(-z))\). The gates can also be multi-variate, another common choice being the softmax mapping \(v \mapsto \exp(v)/\sum_j \exp(v_j)\), where the numerator is applied coordinate-wise, but the denominator needs information from each input.

A typical simplification is to remove the biases, and also consider only two layers, written as 
\[
x \mapsto a^T \sigma(Vx),
\]
where \(a \in \mathbb{R}^m\) and \(V \in \mathbb{R}^{m \times d}\).

There is significant focus in these notes on this two-layer setup because the higher-layer constructions often have strictly worse analysis (at present), in contrast with practice.

The decomposition of \(\mathcal{R}\) is now given as follows. The starting point is that we would like to minimize \(\mathcal{R}\), over data we have not seen, but only have access to \(\widehat{\mathcal{R}}\), which uses data we have seen. For convenience define a mapping \(\widehat{f} := F(\cdot; w)\), where \(w \in \mathbb{R}^p\) is chosen via some algorithm aiming to minimize \(\widehat{\mathcal{R}}\) (for example, gradient descent). To get a better handle on \(\mathcal{R}\), let \(\bar{f} := F(\cdot; \bar{w})\) denote an ideal mapping which does very well on \(\mathcal{R}\) (and thus we can only handle \(\bar{f}\) mathematically, it is not something our algorithms can access). Using \(\bar{f}\), we can decompose \(\mathcal{R}(\widehat{f})\) as
\[
\mathcal{R}(\widehat{f}) = \mathcal{R}(\widehat{f}) - \widehat{\mathcal{R}}(\widehat{f}) + \widehat{\mathcal{R}}(\widehat{f}) - \widehat{\mathcal{R}}(\bar{f}) + \widehat{\mathcal{R}}(\bar{f}) - \mathcal{R}(\bar{f}) + \mathcal{R}(\bar{f}) - \mathcal{R}(f) + \mathcal{R}(f).
\]

The main content of this book is organized around this decomposition; the idea is that we can show \(\widehat{\mathcal{R}}(\widehat{f})\) is small by showing the four terms in the right hand side are all small. Unfortunately, there is evidence that this classical decomposition is in fact unable to analyze deep learning; as follows is concrete obstruction.

\[\text{Remark 1.1.3 (Standard setup and interpolation).}\] A standard refinement to this setup is statistical learning theory: suppose future and past data are drawn IID from a common distribution, whereby \(\widehat{\mathcal{R}}(\widehat{f}) \to \mathcal{R}(\widehat{f})\) and \(\widehat{\mathcal{R}}(\bar{f}) \to \mathcal{R}(\bar{f})\) as \(n \to \infty\) under a variety of regularity conditions, as discussed in ??.

By contrast, as revealed empirically (Neyshabur et al., 2014; Zhang et al., 2017), it is often the case that \(\widehat{\mathcal{R}}(\widehat{f}) = 0 \ll \mathcal{R}(\bar{f}) \approx \mathcal{R}(\bar{f})\); ostensibly this is outside the purview of ??, which seems to need \(\widehat{\mathcal{R}} \approx \mathcal{R}\) in order to be effective. This setting is sometimes called interpolation and has attracted considerable recent attention (Belkin et al., 2018).

Another standard criticism is that the various components of eq. (1.1.2) or simply that concerns are too compartmentalized, and a tight analysis is impossible. (need to cite a lot more.)

To circumvent these issues, the present perspective again focuses on data- and algorithmic-sensitive complexity terms, and on treating bounds as tools. For instance, as will be seen in (didn’t type yet...: the interpolation setting can be handled via ?? by simply splitting the
problem into noise and signal terms and analyzing each separately. contradiction with each other, and a tight analysis is hopeless. )

1.2 Status and purpose of deep learning theory

Bad news. The practical successes in Examples 1.0.1 and 1.0.2 were not only produced without mathematical understanding of their components, they moreover were evolutions of a sequence of prior architectures and training practice which in turn were also produced without mathematical understanding. As such, is mathematical understanding necessary, and is there any way for this understanding to stop falling further and further behind?

One perspective is that theory must fundamentally change in its approach; this is already evidence by the main volume of works being phenomenological: identifying an empirical phenomenon and then producing mathematical analysis capturing some aspect, possibly after simplification, and not attempt to analyze the full end-to-end training in generality (i.e., showing $R(f) \approx 0$ under practical conditions). This is in fact the perspective of Remark 1.1.3 which to date has only produced mathematical analyses for linear predictors with simplified (usually Gaussian) data. This approach too is without pitfalls, since for instance the applied community has stopped the practice of achieving $R(f) \approx 0$ at all costs, and now is again achieving something closer to $\hat{R}(\hat{f}) \approx R(\hat{f})$.

Good news? With the preceding in mind, why study deep learning theory?

1. Deep networks are deployed throughout the world now, and therefore it is essential — for safety and sanity — to understand them and their weaknesses. Mathematical analysis — even of the phenomological sort — can help with this, even without being a complete characterization.

2. Analyzing deep learning requires new mathematical insights, and this insights can feed back into other areas of machine learning and mathematics. (should list a bunch of examples, not just SVM solvers...)

3. Math is fun.
Chapter 2

Shallow networks and continuous functions
Chapter 3

Initialization and overparameterization
Chapter 4

Benefits of other architectures
Part I

Optimization
Chapter 5

Smooth NTK
Chapter 6

Strongly convex NTK
Chapter 7

Margins and feature learning
Part II

Generalization
Part III

Other topics
Chapter 8

Distribution modeling
Bibliography


Appendix A

Technical background

A.1 Convexity
A.2 Miscellaneous inequalities
A.3 Probability