Learning Optimal Algorithms for Parametric Optimization Problems

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät
der Eberhard Karls Universität Tübingen
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)

vorgelegt von
Michael Sucker
aus Gießen

Tübingen
2025

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	27.06.2025
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter	Prof. Dr. Peter Ochs
2. Berichterstatter	Prof. Dr. Andreas Prohl

Disclaimer: This thesis adapts the `kaobook`-template by F. Marotta, which is based on K. A. Ohori's doctoral thesis and the Tufte-LaTeX class.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dr. Peter Ochs for granting me the opportunity to pursue the PhD with him. If you would not have been in Tübingen at exactly this moment in time, I probably would have never pursued a PhD, and I'm very glad that you patiently waited for me to figure out whether I should try it or not. The shear amount of what I have learned during this process, in terms of math, in terms of programming, in terms of doing research, in terms of writing papers – it is close to incredible. There are two more things that I'm particularly grateful for: First, you gave me a lot of freedom in following my own ideas. On the one hand, this made the last three years extremely enjoyable, because it allowed me to dive deeply into creative thinking and consider the problems from my own point of view, which, I guess, is sometimes quite different from yours. On the other hand, through this process I learned to put more trust into my own ideas, because I can see now that they can work out quite well. Second, it seems to me that you are very good at figuring out when your students are indeed stuck and need some guidance, or when they just need to be pushed a little more because they have not tried hard enough yet. In both ways, this is very helpful. Thank you!

Second, I would like to thank my colleague Dr. Camille Castera, with whom I shared the office for most of this time. Your guidance and help with so many different things, especially during the early stages of my PhD, was very helpful and probably also saved me a lot of time. Similarly, I would also like to thank Sheheryar Mehmood, our PyTorch-expert, who I could always come to ask if my code was not working as expected, or just to discuss some ideas. In general, I am very glad that you two stayed here in Tübingen with me, because otherwise I'm not sure if I would have made it until the end, and I really enjoyed all the discussions that we had, in particular if the simple-looking problems turned out to be a bit thornier than expected. Thank you!

I would also like to take the opportunity to express gratitude towards my family. Especially, I would like to thank my mother Gabriele and my father Jens for their unwavering support throughout all this time. Even when I, after finishing my first bachelors, proposed to start all over again to study mathematics and computer science, you did not hesitate for a second. Another big "Thank you" goes out to my (little) sister Marie: Whenever we talked about the doctoral thesis, recent progress in it, or just mathematics in general, and you tried to understand what I was babbling, I at least felt that I could see fascination in your eyes. And that was always motivating for me. As one of my best friends remarked once: "She's clearly the best of you three", and I can just agree with that. Lastly, I would also like to thank my brother Matthias, who is the real academic of us both. I'm always impressed by how much you know, and it is always a joy when we're together at home.

Finally, I would like to express my deepest gratitude towards my wife, Jule. We have been together way over a decade by now, and we have gone through easy times and through tougher times, but we have mastered all of them without a hitch, because we work so well as a team. With you by my side, I can face the future with optimism and I look forward to what lies ahead of us.

Michael Sucker
Tübingen, March 31, 2025

Abstract

From an abstract point of view, the problem of optimally selecting a variable or arranging certain things in the best possible way occurs frequently and in a wide variety of situations. It is therefore not surprising that so-called optimization problems permanently have to be solved in science and industry. At the same time, however, nowadays these problems get increasingly complex and their solution more time- and energy-consuming. Thus, there is a constant need for ever more efficient optimization algorithms to solve such problems.
"Learning-to-optimize" is a recent line of research that leverages machine learning techniques to automatically build and accelerated optimization algorithms. While the empirical results can be impressive, theoretical guarantees are mostly lacking, such that the application of learned optimization methods is still questionable and thus not widely adopted. This applies in particular to safety-critical applications with their need for guarantees, not least because there are already the conventional algorithms that can provably solve these problems.
One of the reasons why the theoretical understanding of learned optimization algorithms lags so far behind is that the respective established mathematical language and the usual chains of reasoning in mathematical optimization and in machine learning are quite different: Mathematical optimization most often relies on geometric arguments and deductive reasoning, while machine learning models learn from observational data, that is, the results are data-dependent, such that the guarantees are of statistical nature. This results in the problem that most known proof-strategies fail sooner or later or, even worse, are not applicable at all.

Therefore, this thesis provides a series of contributions towards new guarantees and a better theoretical understanding of learned optimization algorithms, as well as a way for bridging the gap between conventional optimization theory and learning-to-optimize.
The first part considers a simplified model of an optimization algorithm and provides generalization guarantees for its performance based on the observations during training. Furthermore, it also introduces a corresponding learning procedure as well as several design choices for learning optimization algorithms.
Since many limitations encountered in the first part are based on the inadequacy of the underlying model, the second part introduces a more faithful model of optimization algorithms and how practitioners use them. This construction is done in a bottom-up approach, which allows for minimal assumptions and wide applicability, and effectively solves the problems of the first part all at once. Even more so, this model offers the possibility for deriving generalization guarantees for most conceivable performance measures rather easily.
Then, based on this new model, the third part provides a new proof-strategy with which the gap between conventional optimization theory and learning-to-optimize can be closed to a large extent. Here, the underlying idea is exemplified for the problem of proving convergence to stationary points of non-smooth and non-convex loss functions. This results in a generalization guarantee for the learned optimization algorithm which certifies that, roughly speaking, the probability that the learned algorithm will converge to a stationary point of the loss function is lower-bounded by corresponding quantities that are observable during training.
Finally, the last part summarizes the contributions of this thesis and discusses its limitations. Further, it shows how the remaining problems can be explained based on the proposed model, which of these problems can potentially be solved based on statistical approaches and which will probably always have to be solved in the conventional way. Lastly, some potential directions for future research are discussed.

Taken together, these results collectively contribute to a better understanding of learning-to-optimize and the overarching objective of obtaining more efficient optimization algorithms through the application of machine learning while keeping the needed theoretical guarantees.

Zusammenfassung

Abstrakt betrachtet tritt das Problem, eine Variable optimal auszuwählen oder gewisse Dinge bestmöglich anzuordnen, häufig und in den unterschiedlichsten Situationen auf. Es ist daher nicht verwunderlich, dass sogenannte Optimierungsprobleme in Wissenschaft und Industrie andauernd gelöst werden müssen. Gleichzeitig werden diese Probleme heutzutage aber immer komplexer und ihre Lösung immer zeit- und energieaufwändiger. Es besteht daher ein ständiger Bedarf an immer effizienteren Optimierungsalgorithmen zur Lösung solcher Probleme.
"Lernen-zu-optimieren" ist ein relativ junger Forschungszweig, der Techniken des maschinellen Lernens zur automatischen Erstellung und Beschleunigung von Optimierungsalgorithmen einsetzt. Obgleich die empirischen Ergebnisse beeindruckend sein können, fehlt es zumeist an theoretischen Garantien, sodass die Anwendung solcher Methoden immer noch fragwürdig und dementsprechend nicht weit verbreitet ist. Dies gilt insbesondere für sicherheitskritische Anwendungen mit ihrem Bedarf an Garantien, nicht zuletzt weil es bereits herkömmliche Algorithmen gibt, die diese Probleme bewiesenermaßen lösen können.
Einer der Gründe, warum das theoretische Verständnis von erlernten Optimierungsalgorithmen so weit hinterherhinkt, ist, dass die jeweils etablierte mathematische Sprache und die üblichen Argumentationsketten in der mathematischen Optimierung und im maschinellen Lernen sehr unterschiedlich sind: Die mathematische Optimierung stützt sich meist auf geometrische Argumente und deduktives Denken, während maschinelle Lernmodelle aus Beobachtungsdaten lernen, d.h. die Ergebnisse sind datenabhängig, so dass die Garantien statistischer Natur sind. Daraus ergibt sich das Problem, dass die meisten bekannten Beweisstrategien früher oder später versagen oder, was noch schlimmer ist, überhaupt nicht anwendbar sind.

Daher liefert diese Arbeit eine Reihe von Beiträgen zu neuen Garantien und einem besseren theoretischen Verständnis von gelernten Optimierungsalgorithmen sowie einen Weg zur Überbrückung der Lücke zwischen konventioneller Optimierungstheorie und dem Lernen von Optimierungsalgorithmen.
Im ersten Teil wird ein vereinfachtes Modell für Optimierungsalgorithmen betrachtet und es werden Verallgemeinerungsgarantien für deren Leistung basierend auf Beobachtungen während des Trainings gegeben. Darüber hinaus werden ein entsprechendes Lernverfahren sowie mehrere Design-Entscheidungen für das Lernen von Optimierungsalgorithmen vorgestellt.
Da viele der im ersten Teil genannten Einschränkungen auf der Unzulänglichkeit des zugrundeliegenden Modells beruhen, wird im zweiten Teil ein realistischeres Modell für Optimierungsalgorithmen und deren Anwendung in der Praxis hergeleitet. Die Konstruktion erfolgt dabei in einem Bottom-up-Ansatz, wodurch minimale Annahmen und eine breite Anwendbarkeit möglich sind und die Probleme des ersten Teils effektiv auf einen Schlag gelöst werden. Ferner bietet dieses Modell die Möglichkeit, verhältnismäßig einfach Verallgemeinerungsgarantien für die meisten denkbaren Leistungskennzahlen herzuleiten.
Basierend auf diesem neuen Modell wird im dritten Teil eine Beweisstrategie vorgestellt, mit der die Lücke zwischen der herkömmlichen Optimierungstheorie und dem Lernen von Optimierungsalgorithmen zu großen Teilen geschlossen werden kann. Hierbei wird die zugrundeliegende Idee anhand des Problems der Konvergenz zu stationären Punkten von nicht-differenzierbaren und nicht-konvexen Verlustfunktionen veranschaulicht. Daraus ergibt sich eine Generalisierungsschranke für den gelernten Algorithmus, die vereinfacht gesagt garantiert, dass die Wahrscheinlichkeit, dass dieser zu einem stationären Punkt der Verlustfunktion konvergiert, durch entsprechende, während des Trainings beobachtbare Größen nach unten begrenzt ist.
Abschließend werden im letzten Teil die Beiträge dieser Arbeit zusammengefasst und ihre Einschränkungen erörtert. Außerem wird dargelegt, wie sich auch die verbleibenden Probleme auf der Grundlage des vorgeschlagenen Modells erklären lassen, welche dieser Probleme potenziell auf der Grundlage statistischer Ansätze gelöst werden können und welche wahrscheinlich immer auf herkömmliche Weise gelöst werden müssen. Zu guter Letzt werden einige mögliche Richtungen für die zukünftige Forschung diskutiert.

Zusammengenommen tragen diese Ergebnisse zu einem besseren Verständnis des Lernens von Optimierungsalgorithmen und dem übergeordneten Ziel bei, durch die Anwendung von maschinellem Lernen effizientere Optimierungsalgorithmen zu erhalten und dabei die erforderlichen theoretischen Garantien zu wahren.

Contents

1 Introduction

There is a well-known legend about the founding of Carthage: The Phoenician princess Dido had to flee from her brother Pygmalion and eventually arrived on the coast of modern-day Tunisia, where she asked the ruling king Iarbas for land. Thereupon he promised her that she would receive as much land as she could enclose with an oxhide. Agreeing to the deal, she then cut the oxhide into fine strips, tied them together and thus obtained a long ribbon that she wanted to stretch around the perimeter of her land. Consequently, the question arose as to what shape the ribbon should take so that it would span a country with the *largest possible area* (for example, see Blåsjö, 2005; Taschner, 2020).

Such optimization problems, that is, to arrange things in the best possible way, arise naturally in many applications. As a result, the history of solving optimization problems dates back to antiquity and covers various fields inside and outside of mathematics. For example, in geometry, the above optimization problem is known as the *isoperimetric problem* and its solution was already known to the ancient greek (Goldstine, 1980; Blåsjö, 2005). Related to physics, Pierre de Fermat studied the refraction of light, resulting in his principle which states that the path taken by a ray is the one that can be traveled in the least time. Similarly, Galileo, as well as Johann and Jacob Bernoulli, studied *brachistochrone curves*, that is, curves on which a (frictionless) bead slides in the shortest amount of time. A solution to this problem was also found by Leonhard Euler, who, more generally, studied the problem of finding curves that minimize or maximize a certain functional, resulting in the *calculus of variations*. Later, this approach was simplified and extended by Joseph-Louis Lagrange, resulting in the Euler-Lagrange equations, which are used extensively in classical mechanics (Goldstine, 1980). Related to statistics, the problem of fitting linear equations to observations was studied by Adrien-Marie Legendre and Carl Friedrich Gauß, resulting in the method of *least squares*, which, for example, was also used in the rediscovery of the dwarf-planet Ceres (Stigler, 1981; Teets et al., 1999).

Nowadays, optimization problems are ubiquitous in science and industry: They range from logistics (finding the best route for a fleet of vehicles), finance (minimizing the risk of a portfolio while maximizing its return) and production (minimizing production costs under constraints), over healthcare (allocation of resources and treatments to patients), telecommunication (optimizing network resources to ensure consistent utilization and high service quality) and marketing (placement of advertisement), up to engineering (best weight-to-strength ratio of components), economics (optimal pricing models while taking market conditions into account) and machine learning (determining the parameters of a model, for example, training a neural network), just to name a few (see, for example, Hillier et al., 2015). In contrast to many rather traditional optimization problems, however, it is usually not possible to find a closed-form solution for these more modern problems, such that they are solved iteratively with a suitable algorithm. At the same time, these problems are also becoming increasingly high-dimensional (for example, see Sevilla et al., 2022; Villalobos et al., 2022), which makes their iterative solution expensive and

time-consuming. Thus, there is a constant need for ever more efficient algorithms. However, there is a crucial problem: Some of the known algorithms are already *optimal*, that is, it seems like we cannot improve on them any further, so to speak. If this is not already surprising enough, at the same time these algorithms are typically build from a few basic ingredients only. So one might wonder how such simple algorithms can actually be the best out there, and whether there *really* isn't a better way? The solution to this conundrum can be found by considering what is actually meant when an algorithm is labeled as being "optimal". Basically, it certifies that, among all the possible functions under consideration, there is one on which no hypothetical other algorithm with the same information can be considerably faster, and that its general performance on all other functions under consideration is not worse than that. Thus, whether this yields a reasonable "ranking" of algorithms strongly depends on which and how many functions we are considering: The more functions there are to choose from, the more similar the performance of algorithms becomes, culminating in the limiting case in which random guessing *is* the "optimal" strategy. On the flip side, however, this also means that we get a more differentiated picture (and faster algorithms) if we make the setting more concrete. Even more so, one could argue that whenever we have a specific problem to solve, we are not in fact interested in whether the algorithm could potentially also solve all other kinds of problems as long as it can solve the given problem at hand. Given this, we might wonder:

What prevents us from developing algorithms that work particularly well in a specific scenario?

1.1 Analytical Tractability

We ought then to consider the present state of the universe as the effect of its previous state and as the cause of that which is to follow. An intelligence that, at a given instant, could comprehend all the forces by which nature is animated and the respective situation of the beings that make it up, if moreover it were vast enough to submit these data to analysis, would encompass in the same formula the movements of the greatest bodies of the universe and those of the lightest atoms. For such an intelligence nothing would be uncertain, and the future, like the past, would be open to its eyes.

–Pierre-Simon Laplace, A Philosophical Essay on Probabilities

For example, see Dale et al. (2012).

To answer this question, we have to consider the whole process of solving an optimization problem, described, for example, by Nocedal et al. (2006) and, in even more detail, by Hillier et al. (2015). A summary of this process is depicted in Figure 1.1: In the beginning, we have a specific, yet often only vaguely defined, real-world problem that we want to solve. This has to be translated into a mathematical model, that is, an idealized representation of the real world, and then, based on this model, we can derive an algorithm to solve it. The crux of the matter is that, in each step, our choices are restricted by *analytical tractability*, that is, we have to be able to analyze the interplay between our model and the algorithm. For example, a detailed and accurate model of the real-world problem is obviously desirable. However, the more detailed this model is the more complex the analysis will be, such that the mathematical model will always stay an abstract description of the real-world, with some details being lost along the way. While this makes the model more

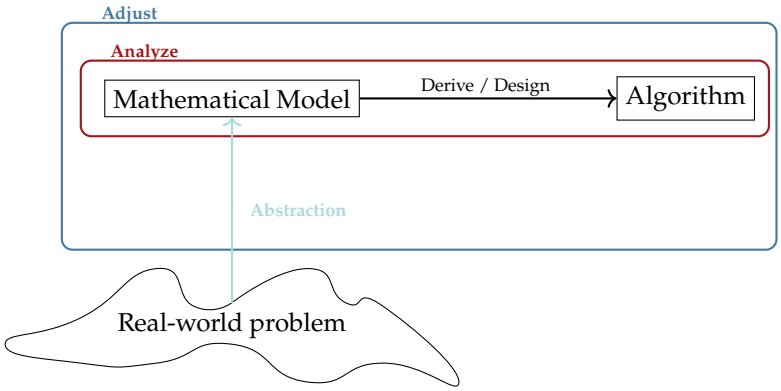

Figure 1.1: Abstract process of solving an optimization problem.

widely applicable, it comes at the price of an inherent loss of information for each particular problem. Similarly, a fast algorithm is desirable. To achieve this, one could try to incorporate highly customized components into its update step. Yet again, more and more detailed components make the analysis increasingly difficult, so that one might end up with a sophisticated algorithm that is practically useless for a given problem, just because there are a few parameters that cannot be chosen well enough since the setting is too difficult to analyze. Additionally, there is in fact another problem: These modeling steps are themselves tedious and time-consuming. Thus, if they have to be performed for each single problem separately, the time that is won by a faster algorithm is more than compensated by the time that is lost for its derivation. Given these problems, we might wonder:

Do we really have to be able to analyze every single step? Isn't there an automated way?

1.2 Learning-to-Optimize

You can observe a lot by just watching.

–Lawrence Peter „Yogi" Berra

Automatically adapting algorithms to a given setting based on observational data is where machine learning excels. Roughly speaking, its goal is to develop algorithms that can learn from observations to solve a task *without* us providing explicit instructions. Hence, it could be the way to design algorithms that automatically learn from data how to optimize a given quantity of interest. Even more so, since these algorithms get trained directly on the problem at hand, they can identify which information is actually relevant for solving it. Thus, machine learning can *alleviate* the bounds of analytical tractability:

- *Information*: During training, the algorithm can potentially access all the structure that is inherent in the problem at hand instead of just the one that we are able to model.
- *Automation*: Instead of deriving the correct algorithm on pen-and-paper for each problem separately, the algorithm gets automatically adapted to the given setting through training.

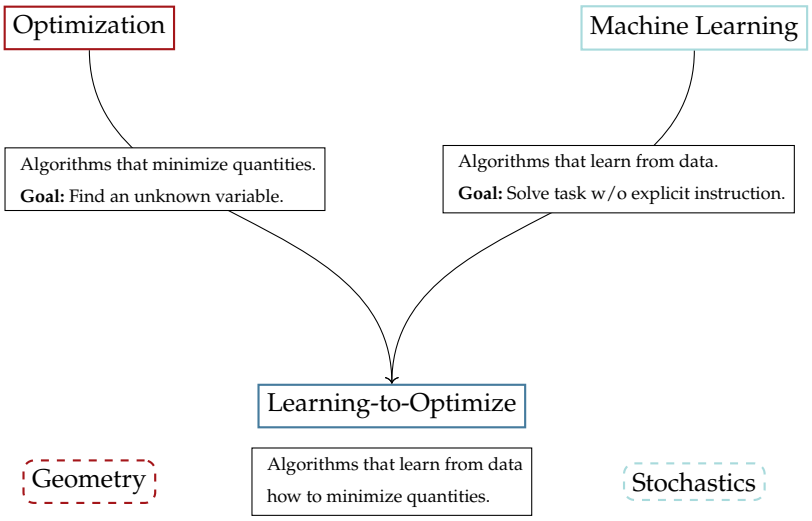

Figure 1.2: Intuitive description of learning-to-optimize: It combines the goal of an optimization algorithm with the goal of a machine learning algorithm. One of the reasons that make this endeavor so complicated (from a theoretical point of view) is the fact that the established mathematical language in these two fields is quite different: the arguments in mathematical optimization are usually based on geometric notions, whereas machine learning tends to approach problems from the perspective of stochastics.

- *Possibilities*: Instead of deriving all free parameters analytically, these get chosen numerically during training. As a result, the choice of building blocks is much less restricted when designing an algorithm, which drastically increases the number of possibilities.

Thus, there is a huge potential upside of applying machine learning techniques in optimization and this could in fact be the way to completely new algorithms that break the barriers of conventional ones. However, going *explicitly beyond* analytical tractability is also the key problem: In most instances, we cannot analyze the algorithms in the same way that we are used to and existing theorems do not apply. This is because we do not really know which structure the algorithm actually uses for solving the problem or, if the design of the algorithm is too complicated, we might not even be able to write down the update step correctly. Further, even if the setting would allow for an analytical treatment, we might not know (beforehand) which parameters get chosen during training, so we cannot make a statement about how the algorithm behaves on unseen data. To make matters worse, since we train the algorithm on a given data set, the final algorithm is *data-dependent*. Thus, we cannot guarantee the performance of an algorithm in the way we are used to. However, this is a major problem: The usefulness of an optimization algorithm without guarantees is at least questionable, especially given the fact that we already have functioning optimization algorithms as baseline. Therefore, if we want to leverage the potential of machine learning in optimization, we have to find new ways to guarantee that the resulting algorithms work in the intended way. Here, we might wonder:

How could such guarantees look like?

1.3 Learning from Observation

If we be, therefore, engaged by arguments to put trust in past experience, and make it the standard of our future judgement, these arguments must be probable only [...].

–Daivd Hume, An Enquiry Concerning Human Understanding

One way to new kinds of guarantees is proposed in this manuscript and it is based on the following fundamental insight, which describes the essential difference to traditional optimization:

Instead of *knowing* how the algorithm behaves, we are actually able to *observe* it.

While this simple observation might sound like a truism at first, it is indeed the crucial difference to traditional optimization, so that new approaches are most likely to be found there. Especially, as will be layed out during this manuscript, it is the key for deriving new kinds of guarantees and actually also allows for transferring known theoretical results into this new setup. However, there is one thing to bear in mind: Ultimately, this approach allows us to *learn from observations*, that is, instead of mathematical certainty, the end result is a statement about probabilities. Even more so, since the algorithm is adapted based on these observations, there is the potential of being "fooled by randomness", that is, adapting to structure that is only present in the given sample and not in the overall population. Therefore, assuming for now that we are in fact able to train an optimization algorithm in such a way that its performance is compelling during training, a central question to be answered is whether this observable performance during training is actually *representative* for the performance on unseen data. Ultimately, this will result in generalization guarantees, which certify that, given enough observations during training, it is highly likely that the performance on unseen data is approximately the same.

To summarize, this manuscript basically deals with one central goal:

Central Goal

Learning optimization algorithms *with* theoretical guarantees for their performance.

This, however, bears several questions in itself:

- What is a suitable model for optimization algorithms that allows for learning?
- How do we actually learn an optimization algorithm?
- How could such new guarantees look like?
- Can we also make a statement about asymptotic convergence?

Furthermore, since the ultimate goal is to automate the design, the training, and the computation of the guarantees of an optimization algorithm for as many areas of application as possible, we formulate the following guiding principle, that we try to follow throughout the remainder of this text:

Guiding Principle

The model and the procedure should be widely applicable.

- Both should rely on as few assumptions as possible.
- Both should be largely independent of the implementation of the algorithm.

1.4 Outline and Originality

Chapter 2 summarizes the background that is necessary for a proper understanding of this work. In particular, this involves background on probability theory in Section 2.1, the PAC-Bayesian approach to learning in Section 2.2, background on mathematical optimization in Section 2.3, and learning-to-optimize in Section 2.4. At the same time, Chapter 2 also introduces a concise notation and summarizes related work. Then, Chapter 3 presents the first new results for learning-to-optimize. It is based on the paper "PAC-Bayesian Learning of Optimization Algorithms" (Sucker et al., 2023), which got published at the *International Conference on Artificial Intelligence and Statistics* (AISTATS), and its extension "Learning-to-Optimize with PAC-Bayesian Guarantees: Theoretical Considerations and Practical Implementation" (Sucker et al., 2024a), which has been accepted (after minor revision) by the *Journal of Machine Learning Research* (JMLR). The co-author contributions to these papers are as follows:

PAC-Bayesian Learning of Optimization Algorithms

	Ideas	Analysis	Experiments	Writing
Michael Sucker	75%	90%	90%	80%
Peter Ochs	25%	10%	10%	20%

Learning-to-Optimize with PAC-Bayesian Guarantees: Theoretical Considerations and Practical Implementation

	Ideas	Analysis	Experiments	Writing
Michael Sucker	70%	80%	90%	70%
Jalal Fadili	15%	10%	0%	20%
Peter Ochs	15%	10%	10%	10%

Based on the limitations of this initial approach, Chapter 4 presents a new and more faithful model of optimization algorithms, and how to use it to get better guarantees. This chapter is based on the preprint "A Markovian Model for Learning-to-Optimize" (Sucker et al., 2024c), which at the time of writing is still under review, and the co-author contributions to this paper are as follows:

A Markovian Model for Learning-to-Optimize

	Ideas	Analysis	Experiments	Writing
Michael Sucker	95%	90%	90%	80%
Peter Ochs	5%	10%	10%	20%

Finally, Chapter 5 treats the problem of convergence of learned optimization algorithms. It is based on the paper "A Generalization Result for

Convergence in Learning-to-Optimize" (Sucker et al., 2024b), which got published at the *International Conference of Machine Learning* (ICML), and the co-author contributions to this paper are as follows:

A Generalization Result for Convergence in Learning-to-Optimize

	Ideas	**Analysis**	**Experiments**	**Writing**
Michael Sucker	55%	80%	90%	80%
Peter Ochs	45%	20%	10%	20%

To conclude, Chapter 6 discusses the results and their limitations, and provides a few ideas for future research.

Part I

Background and Related Work

Mathematical Background 2

This work is located at the intersection of stochastics, learning theory, and mathematical optimization, and for some parts it actually requires rather advanced techniques from every single one of them: ideas from the theory of Markov processes, non-smooth and non-convex optimization, and PAC-Bayesian generalization bounds. All these will be combined to provide a new point of view onto the actual topic of research, which is to learn optimization algorithms with theoretical guarantees on their performance. However, this interdisciplinarity comes at a price: It requires an extended preliminary section to cover the corresponding background material and it requires an extended introduction of a concise notation which, as the author can tell from personal experience, is way harder to find than it might seem, especially if the established notations in the respective fields differ and, at the same time, the number of different objects and variables increases.

Therefore, this section serves three purposes: First, except for the most elementary results, basically all background material that is needed for a proper understanding of this work is covered. Second, the notation that is used throughout the rest of this manuscript is introduced. Third, existing approaches in learning-to-optimize and the PAC-Bayesian approach to learning are discussed, which puts the present manuscript into context.

Because it allows to start at the very basis of mathematics, namely *sets*, the discussion starts with the background material on probability theory.

2.1 Background on Probability Theory

"Probability theory is nothing but common sense reduced to calculations." - *Pierre-Simon Laplace*

The results summarized here are standard in the corresponding literature on measure- and probability theory, and can be found, for example, in the (highly recommended) book by Kallenberg (2021) or in the book by Klenke (2013), and the following discussion is based on these two.

Given some space $\mathcal{U}$, typically denoted in script-font, we will write *subsets* in typewriter font, for example, $\mathsf{A} \subset \mathcal{U}$, and we denote the corresponding *indicator function*, that is, the function that is constant equal to one on A and zero otherwise, by $\mathbb{1}_{\mathsf{A}}$. Then, we can form collections of subsets of $\mathcal{U}$, most importantly, a *σ-algebra*, which is defined as a non-empty collection of subsets of $\mathcal{U}$ that is closed under complementation and countable unions/intersections. In general, σ-algebras will be written in Fraktur-font, for example, $\mathfrak{U}$, and the tuple $(\mathcal{U}, \mathfrak{U})$ is called a *measurable space*. However, there are other collections of sets that are important for this manuscript, namely *π-systems* and *λ-systems*:

Definition 2.1.1 *A collection $\mathcal{C}$ of subsets of $\mathcal{U}$ is called a* π-system, *if it is closed under finite intersection, that is:*

$$\mathsf{A}, \mathsf{B} \in \mathcal{C} \quad \Longrightarrow \quad \mathsf{A} \cap \mathsf{B} \in \mathcal{C}\,.$$

Further, $\mathcal{C}$ is called a λ-system, *if*

(i) $\mathcal{U} \in \mathcal{C}$,

(ii) $\mathsf{A}, \mathsf{B} \in \mathcal{C}$ *with* $\mathsf{A} \subset \mathsf{B}$ *implies* $\mathsf{B} \setminus \mathsf{A} \in \mathcal{C}$,
(iii) $\mathsf{A}_1, \mathsf{A}_2, \ldots \in \mathcal{C}$ *with* $\mathsf{A}_n \uparrow \mathsf{A}$ *implies* $\mathsf{A} \in \mathcal{C}$.

Their importance is based on the following *monotone-class theorem*, which underlies one of the most widely used proof-strategies in measure-theory, sometimes called "measure-theoretic induction", as it allows for extending a property from $\mathcal{C}$ to $\sigma(\mathcal{C})$, the σ-algebra that is *generated* by $\mathcal{C}$:

$\sigma(\mathcal{C})$ is the smallest σ-algebra on $\mathcal{U}$ that contains $\mathcal{C}$.

Theorem 1.1 of Kallenberg (2021).

Theorem 2.1.1 *For any π-system $\mathcal{C}$ and λ-system $\mathcal{D}$ in a space $\mathcal{U}$, we have*

$$\mathcal{C} \subset \mathcal{D} \implies \sigma(\mathcal{C}) \subset \mathcal{D}\,.$$

Given two spaces $\mathcal{U}$ and $\mathcal{V}$, their *(Cartesian) product* is denoted by $\mathcal{U} \times \mathcal{V}$, and the product of a generic number of spaces $\mathcal{U}_1, \ldots, \mathcal{U}_n$ is denoted by $\prod_{i=1}^n \mathcal{U}_i$. In this case, if all spaces are the same, that is, $\mathcal{U}_1 = \ldots = \mathcal{U}_n = \mathcal{U}$, this is abbreviated as $\mathcal{U}^n$. Similarly, if $\mathcal{U}$ and $\mathcal{V}$ are in fact endowed with σ-algebras $\mathfrak{U}$ and $\mathfrak{V}$, the corresponding *product-σ-algebra* on $\mathcal{U} \times \mathcal{V}$ is denoted by $\mathfrak{U} \otimes \mathfrak{V}$, while the product-$\sigma$-algebra on a generic product-space $\prod_{i=1}^n \mathcal{U}_i$ is denoted by $\bigotimes_{i=1}^n \mathfrak{U}_i$. Here, if all σ-algebras are the same, that is, $\mathfrak{U}_1 = \ldots = \mathfrak{U}_n = \mathfrak{U}$, this is abbreviated as $\mathfrak{U}^{\otimes n}$. However, most of the considered spaces in this work are also endowed with a *topology*, in which case we will use the *Borel-σ-algebra*, that is, the σ-algebra that is generated by the topology, and denote it by $\mathfrak{B}(\mathcal{U})$. Here, the topology will basically always be *Polish*, that is, it is *separable* and admits a *complete metrization*. Similarly, if not specified any further, we will endow product-spaces with the *product-topology*. These choices are especially useful in the case of at most countably many spaces, because then the Borel-σ-algebra corresponding to the product-topology and the product σ-algebra corresponding to the Borel-σ-algebras coincide:

The product-σ-algebra is generated by the *cylinder sets*.

Lemma 2.1 of Kallenberg (2021) or Theorem 14.8 of Klenke (2013).

Lemma 2.1.2 *For a countable number of separable metric spaces $\mathcal{U}_n$, $n \in \mathbb{N}$, we have*

$$\mathfrak{B}\left(\prod_{i\in\mathbb{N}} \mathcal{U}_i\right) = \bigotimes_{i\in\mathbb{N}} \mathfrak{B}(\mathcal{U}_i)\,.$$

Thus, they can be used interchangeably and this fact will be used without any further mentioning.
Similar to the definition of *continuity* of functions between topological spaces, a function $f : (\mathcal{U}, \mathfrak{U}) \to (\mathcal{V}, \mathfrak{V})$ is said to be *measurable*, if $f^{-1}(\mathsf{B}) \in \mathfrak{U}$ for every $\mathsf{B} \in \mathfrak{V}$. Here, one can show that it suffices to check measurability of f on a generator of $\mathfrak{V}$. Then, since the product-σ-algebra is generated by the cylinder sets, one can relate measurability of vector-valued functions to the measurability of their coordinates:

More precisely, f is said to be $\mathfrak{U}$-$\mathfrak{V}$-measurable. However, the σ-algebras will usually be clear from the context, such that they are not mentioned any further.

Lemma 1.9 of Kallenberg (2021).

Lemma 2.1.3 *Let I be an index set, and let $(\mathcal{U}, \mathfrak{U})$, $(\mathcal{V}_i, \mathfrak{V}_i)$, $i \in I$, be measurable spaces. Further, let $f_i : \mathcal{U} \to \mathcal{V}_i$, $i \in I$, be functions and define $f : \mathcal{U} \to \prod_{i\in I} \mathcal{V}_i$, $u \mapsto (f_i(u))_{i\in I}$. Then the following are equivalent:*

(i) f is measurable w.r.t. $\mathfrak{U}$ and $\bigotimes_{i\in I} \mathfrak{V}_i$,
(ii) for every $i \in I$, f_i is measurable w.r.t. $\mathfrak{U}$ and $\mathfrak{V}_i$.

The notation with curly brackets differs from the standard notation. We use it to emphasize the fact that measures are acting on sets. Further, if the set is itself defined with curly brackets, we write these brackets only once instead of twice.

Given a measurable space $(\mathcal{U}, \mathfrak{U})$, a function $\mu : \mathfrak{U} \to [0, \infty]$ is called a *measure*, if it is *countably additive* and satisfies $\mu\{\emptyset\} = 0$. In this case, the triple $(\mathcal{U}, \mathfrak{U}, \mu)$ is called a *measure space*. Here, μ is called *finite*, if $\mu\{\mathcal{U}\} < \infty$, it is called *s-finite* if it is a countable sum of finite measures, and it is called *σ-finite*, if $\mathcal{U}$ can be written as a countable union of disjoint

sets $\mathsf{A}_n \in \mathfrak{U}$ with $\mu\{\mathsf{A}_n\} < \infty$. Given a measure μ on $\mathcal{U}$ and a measurable function $f : \mathcal{U} \to \mathcal{V}$, we can define the *push-forward measure* $\mu \circ f^{-1}$ on $\mathcal{V}$ through $\left(\mu \circ f^{-1}\right)\{\mathsf{A}\} := \mu\{f \in \mathsf{A}\} := \mu\{u \in \mathcal{U} : f(u) \in \mathsf{A}\}$ for $\mathsf{A} \in \mathfrak{V}$. Similarly, if $\mathcal{V} = \mathbb{R}$ and $f \geq 0$, we can define a new measure $f \cdot \mu$ on $\mathcal{U}$, given by $\left(f \cdot \mu\right)\{\mathsf{A}\} = \int_{\mathsf{A}} f(u)\, \mu(du)$. In this case, $f \cdot \mu$ is *absolutely continuous* w.r.t. μ, written as $f \cdot \mu \ll \mu$, and f is the corresponding *Radon-Nikodym derivative* (or *density*). For two such (finite) measures μ and ν, the discrepancy between them can be quantified in terms of the *Kullback-Leibler divergence*, which is defined[1] as:

$$D_{\mathrm{KL}}(\nu \parallel \mu) = \begin{cases} \int_{\mathcal{U}} \log(f(u))\, \nu(du), & \nu \ll \mu \text{ with density } f\,, \\ +\infty, & \text{otherwise}\,. \end{cases}$$

1: In order to avoid the effect of a differing total mass, the Kullback-Leibler divergence is usually used when both measures have the same (finite) total mass, that is, in the case of probability measures, which are introduced below.

Here, for a measurable function $f : \mathcal{U} \to \mathbb{R}$ and a measure μ on $\mathcal{U}$, the integral of f w.r.t. μ is often abbreviated in operator-notation, that is, $\mu\{f\} = \int_{\mathcal{U}} f(u)\, \mu(du)$. In particular, this applies when having multiple iterated integrals at once, that is, in the case of *product-measures*, in which case this has to be read from right to left instead from inside to outside:

Theorem 2.1.4 (Fubini's theorem) *Let $(\mathcal{U}, \mathfrak{U}, \mu)$ and $(\mathcal{V}, \mathfrak{V}, \nu)$ be σ-finite measure spaces. Then it holds:*

(i) There exists a unique measure $\mu \otimes \nu$ on $(\mathcal{U} \times \mathcal{V}, \mathfrak{U} \otimes \mathfrak{V})$, such that

$$(\mu \otimes \nu)\{\mathsf{A} \times \mathsf{B}\} = \mu\{\mathsf{A}\} \cdot \nu\{\mathsf{B}\}, \quad \mathsf{A} \in \mathfrak{U},\ \mathsf{B} \in \mathfrak{V}\,.$$

(ii) For any measurable function $f : \mathcal{U} \times \mathcal{V} \to \mathbb{R}$, that is either integrable w.r.t. $\mu \otimes \nu$ or non-negative, we have

$$\begin{aligned} (\mu \otimes \nu)\{f\} &= \int_{\mathcal{U}\times\mathcal{V}} f(u,v)\, (\mu \otimes \nu)(du, dv) \\ &= \int_{\mathcal{U}} \mu(du) \int_{\mathcal{V}} \nu(dv)\, f(u,v) = \mu\big\{\nu\{f\}\big\} \\ &= \int_{\mathcal{V}} \nu(dv) \int_{\mathcal{U}} \mu(du)\, f(u,v) = \nu\big\{\mu\{f\}\big\}\,. \end{aligned}$$

Theorem 1.29 of Kallenberg (2021), or Theorem 14.14 together with Theorem 14.16 of Klenke (2013).

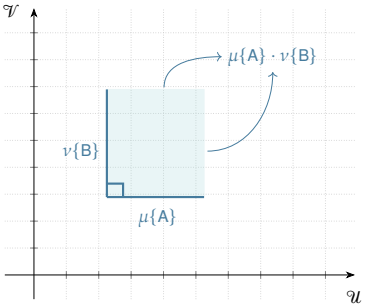

Figure 2.1: The product measure describes a form of orthogonality. More details about this interpretation are provided by Kallenberg (2021, p.173).

As the theorem indicates, the notation for product-measures is consistent with the one for σ-algebras: the product-measure of μ and ν on $\mathfrak{U} \otimes \mathfrak{V}$ is denoted by $\mu \otimes \nu$, the product-measure of $\mu_1, \ldots, \mu_n$ on $\bigotimes_{i=1}^n \mathfrak{U}_i$ is denoted by $\bigotimes_{i=1}^n \mu_i$, and, if all measures are the same, by $\mu^{\otimes n}$. This operator-notation for integrals is especially useful when dealing with so-called *kernels*, which are of fundamental importance for this work:

Definition 2.1.2 *Let $(\mathcal{U}, \mathfrak{U})$ and $(\mathcal{V}, \mathfrak{V})$ be measurable spaces. A mapping $\kappa : \mathcal{U} \times \mathfrak{V} \to [0, \infty]$ is called a* kernel *from $\mathcal{U}$ to $\mathcal{V}$, if the map $u \mapsto \kappa(u)\{\mathsf{A}\}$ is measurable for every fixed $\mathsf{A} \in \mathfrak{V}$, and the map $\mathsf{A} \mapsto \kappa(u)\{\mathsf{A}\}$ is a measure for every fixed $u \in \mathcal{U}$. Furthermore, κ is called* finite, *if $\kappa(u)\{\mathcal{V}\} < \infty$ for all $u \in \mathcal{U}$, it is called* s-finite, *if it is a countable sum of finite kernels, and it is called a* probability kernel, *if $\kappa(u)\{\mathcal{V}\} = 1$ for every $u \in \mathcal{U}$.*

Definition 8.25 of Klenke (2013) or pages 56-57 of Kallenberg (2021).
By abuse of notation, $\kappa : \mathcal{U} \times \mathfrak{V} \to [0, \infty]$ is usually written as $\kappa : \mathcal{U} \to \mathcal{V}$. Further, we use the notation $\kappa(u)\{\mathsf{A}\}$, that is, we separate the variables u and A, to stress the fact that they are typically varied separately: κ is a measure if we *fix* u, and it is a mapping, if we *fix* A.

Kernels are a powerful tool for modeling. For example, by considering the constant kernel $(u, \mathsf{A}) \mapsto \mu\{\mathsf{A}\}$ they generalize measures, by considering $(u, \mathsf{A}) \mapsto \delta_{f(u)}\{\mathsf{A}\}$, where δ is the *Dirac measure*, they allow to transform measurable functions into measures, and they can be used to model stochastic dependencies and transitions. The following lemma provides two important ways of generating a new kernel from given ones. Later

on, this will be used to describe the transition of (stochastic) optimization algorithms in measure-theoretic terms:

Lemma 3.2 of Kallenberg (2021).

Lemma 2.1.5 *Let $\kappa : \mathcal{U} \to \mathcal{V}$ be s-finite. Then*

(i) for any measurable function $f : \mathcal{U} \times \mathcal{V} \to [0, \infty)$, the function $u \mapsto \kappa(u)\{f(u,\cdot)\} = \int_{\mathcal{V}} f(u,v)\, \kappa(u)(dv)$ is measurable, and γ, defined as $\gamma(u)\{\mathsf{A}\} := \big(f(u,\cdot) \cdot \kappa(u)\big)\{\mathsf{A}\} = \int_{\mathsf{A}} f(u,v)\, \kappa(u)(dv)$ is again an s-finite kernel from $\mathcal{U}$ to $\mathcal{V}$,

(ii) for any measurable function $f : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$, we have that $\gamma(u) := \kappa(u) \circ f(u,\cdot)^{-1}$ is an s-finite kernel from $\mathcal{U}$ to $\mathcal{W}$.

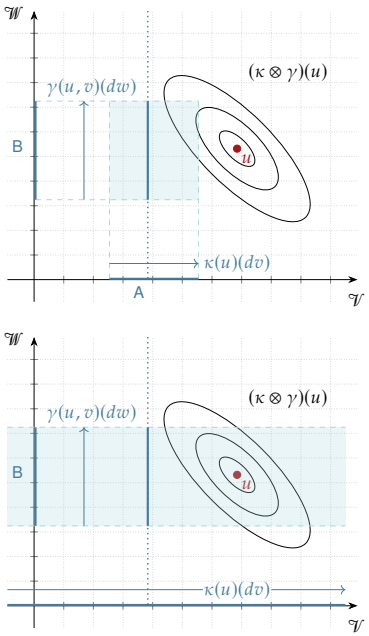

Figure 2.2: Top: Product of kernels. Bottom: Concatenation of kernels.

Similarly, the next definition provides the two fundamental operations on kernels:

Definition 2.1.3 *Let $\kappa : \mathcal{U} \to \mathcal{V}$ and $\gamma : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$ be s-finite kernels. For every measurable function $f : \mathcal{V} \times \mathcal{W} \to [0, \infty]$, the* product *of κ and γ is defined as the kernel $\kappa \otimes \gamma : \mathcal{U} \to \mathcal{V} \times \mathcal{W}$, given by:*

$$(\kappa \otimes \gamma)(u)\{f\} = \int_{\mathcal{V}} \kappa(u)(dv) \int_{\mathcal{W}} \gamma(u,v)(dw)\, f(v,w)\,.$$

Similarly, for every measurable function $f : \mathcal{W} \to [0, \infty]$, the concatenation *of κ and γ is defined as the kernel $\kappa \cdot \gamma : \mathcal{U} \to \mathcal{W}$, given by:*

$$(\kappa \cdot \gamma)(u)\{f\} = \int_{\mathcal{V}} \kappa(u)(dv) \int_{\mathcal{W}} \gamma(u,v)(dw)\, f(w)\,.$$

Here, the following lemma certifies that these operations do indeed yield kernels. Thus, starting from some initial measure, kernels can be used to "construct" a measure on a higher-dimensional space or to transform the initial measure into another one on a possibly different space. Ultimately, this result will be used iteratively to describe the so-called *(joint) distribution* of the iterates generated by the algorithm.

Lemma 3.3 of Kallenberg (2021).

Lemma 2.1.6 *Let $\kappa : \mathcal{U} \to \mathcal{V}$ and $\gamma : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$ be s-finite kernels. Then $\kappa \otimes \gamma$ and $\kappa \cdot \gamma$ are s-finite kernels from $\mathcal{U}$ to $\mathcal{V} \times \mathcal{W}$ and $\mathcal{W}$, respectively.*

Conversely, kernels also allow for *disintegrating* a measure on the product space, which, in the context of probability theory later on, allows for proving existence of so-called *regular conditional distributions*:

Theorem 3.4 of Kallenberg (2021).

Theorem 2.1.7 *Let ρ be a σ-finite measure on $\mathcal{U} \times \mathcal{V}$, where $\mathcal{V}$ is Polish. Then*

(i) $\rho = \mu \otimes \kappa$ for a σ-finite measure μ on $\mathcal{U}$ and a σ-finite kernel $\kappa : \mathcal{U} \to \mathcal{V}$,

(ii) the measures $\kappa(u)$ are μ-a.e. unique up to normalization.

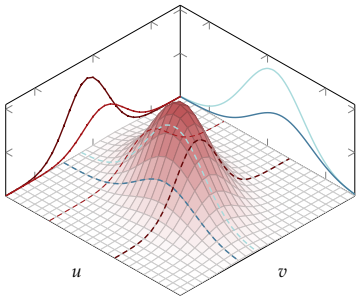

Figure 2.3: Disintegration of a measure on the product space.

Here, one can think of this as if the measure ρ gets decomposed into the "sections" $(\delta_u \cdot \kappa) = (\delta_u \otimes \kappa)\{\mathcal{U} \times \cdot\} = \kappa(u)$.

A *probability space* is a measure space $(\mathcal{U}, \mathfrak{U}, \mu)$, such that $\mu\{\mathcal{U}\} = 1$. In this case, μ is called a *probability measure* and we will denote the set of all probability measures on $\mathcal{U}$ by $\mathcal{M}_1(\mathcal{U})$. Similarly, given a reference measure $\mu \in \mathcal{M}_1(\mathcal{U})$, the set of all probability measures that are absolutely

continuous w.r.t. μ are denoted by $\mathcal{M}_1(\mu)$. Furthermore, we will use the standard notation and denote a generic probability space by $(\Omega, \mathfrak{A}, \mathbb{P})$. Then, a measurable map $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to (\mathcal{U}, \mathfrak{U})$ is called a *random element in* $\mathcal{U}$ and, in the special case of $\mathcal{U} = \mathbb{R}$, it is also called a *random variable*. In this work, we will denote random elements in upper-case letters and their corresponding realizations, that is, elements in the image-space, by lower-case letters, and we will try to match these letters to the corresponding image space, for example, $U = u \in \mathcal{U}$ or $V = v \in \mathcal{V}$. Given a random variable $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathbb{R}$, and assuming the integral exists, the *expected value* of U is defined as the integral w.r.t. $\mathbb{P}$:

$$\mathbb{E}\{U\} := \int_\Omega U(\omega)\, \mathbb{P}(d\omega) = \int_{\mathcal{U}} u\ \left(\mathbb{P} \circ U^{-1}\right)(du) =: \int_{\mathcal{U}} u\, \mathbb{P}_U(du)\,,$$

Since the expectation is the integral w.r.t. the probability measure, we also write it with curly brackets.

where, in this case, the push-forward $\mathbb{P}_U := \mathbb{P} \circ U^{-1}$ is called the *distribution* of U. Then, given a random element V in $\mathcal{V}$ and a measurable function $f : \mathcal{V} \to \mathbb{R}$, the expected value of $f \circ V$ can be expressed more compactly as:

$$\begin{aligned}\mathbb{E}\{f(V)\} = \mathbb{E}\{f \circ V\} &= \int_\Omega (f \circ V)(\omega)\, \mathbb{P}(d\omega) = \int_{\mathcal{V}} f(v)\ \left(\mathbb{P} \circ V^{-1}\right)(dv) \\ &= \int_{\mathcal{V}} f(v)\, \mathbb{P}_V(dv) =: \mathbb{E}_V\{f\}\,.\end{aligned}$$

Furthermore, given two random elements $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{U}$ and $V : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{V}$, the distribution of (U, V) on $\mathcal{U} \times \mathcal{V}$ is called the *joint distribution*, while $\mathbb{P}_U$ and $\mathbb{P}_V$ are called the *marginals* of U and V, respectively.

Given an index set I, and events $\mathsf{A}_i \in \mathfrak{A}$, $i \in I$, we say that the events A_i are *independent*, if for any finite number of distinct indices $i_1, \ldots, i_n \in I$, we have $\mathbb{P}\{\bigcap_{j=1}^n \mathsf{A}_{i_j}\} = \prod_{j=1}^n \mathbb{P}\{\mathsf{A}_{i_j}\}$. Similarly, the collections $\mathcal{C}_i \subset \mathfrak{A}$, $i \in I$, are said to be independent, if independence holds for any finite number of distinct indices $i_1, \ldots, i_n \in I$ and any choice of $\mathsf{A}_{i_j} \in \mathcal{C}_{i_j}$. Using this, random elements $U_i : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{U}_i$, $i \in I$, are said to be independent, if the σ-algebras $\sigma(U_i) \subset \mathfrak{A}$, $i \in I$, are independent. This can equivalently be characterized by their joint distribution:

Lemma 2.1.8 *Let $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{U}$ and $V : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{V}$ be two random elements. Then the following are equivalent:*

(i) U and V are independent,
(ii) the joint distribution of U and V is given by the product of their marginals, that is, $\mathbb{P}_{(U,V)} = \mathbb{P}_U \otimes \mathbb{P}_V$.

Lemma 4.10 of Kallenberg (2021).

Here, using Fubini's theorem, the iterated integration of a measurable function $f : \mathcal{U} \times \mathcal{V} \to \mathbb{R}$ is indicated by the corresponding subscript:

$$\begin{aligned}\mathbb{E}\{f(U,V)\} &= \int_{\mathcal{U}\times\mathcal{V}} f(u,v)\, \mathbb{P}_{(U,V)}(du, dv) \\ &= \int_{\mathcal{U}} \int_{\mathcal{V}} f(u,v)\, \mathbb{P}_V(dv)\, \mathbb{P}_U(du) = \mathbb{E}_U\left\{\mathbb{E}_V\{f(u,\cdot)\}|_{u=U}\right\}\,.\end{aligned}$$

The ordering $u = U$ is intentional: u in the inner integral is fixed to U, not the other way around.

In particular, for any independent random variables $U_1, \ldots, U_n$, we get $\mathbb{E}\left\{\prod_{i=1}^n U_i\right\} = \prod_{i=1}^n \mathbb{E}\{U_i\}$. Conversely, if U and V are not independent, the joint distribution cannot be represented as product of the marginals, such that Fubini's theorem is not applicable. Rather, one has to consider

conditional expectations. Typically, these are introduced by means of the *Radon-Nikodym equation* (or *averaging property*), a more intuitive[2] approach, however, relies on projections in Hilbert-spaces: Consider a σ-field $\mathfrak{F} \subset \mathfrak{A}$, and define

$$M := \left\{ W \in L^2(\mathfrak{A}) \,:\, \exists \tilde{W} \in L^2(\mathfrak{F}) \text{ such that } W = \tilde{W} \; a.s. \right\} .$$

Since addition and scalar multiplication are measurable, M defines a linear subspace of the Hilbert space $L^2(\mathfrak{A})$ and, since limits of measurable functions are again measurable, it is also closed. Thus, by the projection theorem in Hilbert spaces, every $V \in L^2(\mathfrak{A})$ has an a.s. unique decomposition $V = W + R$ with $W \in M$ and $R \perp M$. Thus, one can define the conditional expectation of V, given $\mathfrak{F}$, as an arbitrary $\mathfrak{F}$-measurable *version* of W:

$$\mathbb{E}\{V \mid \mathfrak{F}\} := \tilde{W} .$$

Therefore, the conditional expectation of V, given $\mathfrak{F}$, is defined as the L^2-orthogonal projection onto M. In particular, $\mathbb{E}\{V \mid \mathfrak{F}\}$ is a $\mathfrak{F}$-measurable random variable and only defined almost surely. The following theorem extends this construction to L^1 and establishes the averaging property, which often serves as defining property:

2: Assume that we are trying to "identify" a random variable by collecting information about it in a σ-algebra. In the beginning, we only have access to its average behavior, that is, all random variables that obey the same average behavior are possible candidates. Then, after having observed a certain event, we want to incorporate this new information into our estimates. Thus, only random variables that are consistent with these observations are of interest. At the same time, however, they still need to obey the same average behavior. Hence, while incorporating the new information, we want to stay as close as possible to our established theory, such that we update our estimates solely in the "direction" of the new information and not in any other.

Theorem 8.1 of Kallenberg, 2021.

Theorem 2.1.9 *Let $(\Omega, \mathfrak{A}, \mathbb{P})$ be a probability space. Then, for any σ-field $\mathfrak{F} \subset \mathfrak{A}$, there exists an a.s. unique linear operator $\mathbb{E}\{\cdot \mid \mathfrak{F}\} : L^1(\mathfrak{A}) \to L^1(\mathfrak{F})$, such that, for every $V \in L^1(\mathfrak{A})$ and every $\mathsf{A} \in \mathfrak{F}$, we have*

$$\mathbb{E}\left\{\mathbb{1}_{\mathsf{A}} \mathbb{E}\{V \mid \mathfrak{F}\}\right\} = \mathbb{E}\{\mathbb{1}_{\mathsf{A}} V\} .$$

Then, based on the conditional expectation one can introduce the conditional probability of an event B, given $\mathfrak{F}$, as $\mathbb{P}\{\mathsf{B} \mid \mathfrak{F}\} := \mathbb{E}\{\mathbb{1}_{\mathsf{B}} \mid \mathfrak{F}\}$. By the averaging property, $\mathbb{P}\{\mathsf{B} \mid \mathfrak{F}\}$ is the a.s. unique random variable satisfying

$$\mathbb{E}\left\{\mathbb{1}_{\mathsf{A}} \mathbb{P}\{\mathsf{B} \mid \mathfrak{F}\}\right\} = \mathbb{P}\{\mathsf{A} \cap \mathsf{B}\} .$$

This construction of conditional expectations directly extends to conditioning on random elements U by considering the σ-algebra generated by U:

$$\mathbb{E}\{V \mid U\} := \mathbb{E}\{V \mid \sigma(U)\} .$$

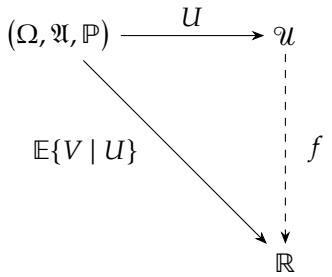

Figure 2.4: Factorization lemma, for example, see Corollary 1.97 of Klenke (2013) or Lemma 1.14 of Kallenberg (2021).

Hence, by definition, $\mathbb{E}\{V \mid U\}$ is an $\sigma(U)$-measurable random variable, such that *Doob's factorization lemma* yields the existence of a measurable function $f : \mathcal{U} \to \mathbb{R}$ that allows for the decomposition:

$$\mathbb{E}\{V \mid U\} = f \circ U .$$

This motivates to define the conditional expectation of V, given $U = u$, as $\mathbb{E}\{V \mid U = u\} := f(u)$, and, as before, the conditional probability of an event B, given $U = u$, as $\mathbb{P}\{\mathsf{B} \mid U = u\} = \mathbb{E}\{\mathbb{1}_{\mathsf{B}} \mid U = u\}$. However, since the expression $\mathbb{P}\{\mathsf{B} \mid U = u\}$ is only defined almost surely, and the corresponding null set N_{B} might depend on B, it is not a given that the map $\mathsf{B} \mapsto \mathbb{P}\{\mathsf{B} \mid U = u\}$ is indeed a measure. Especially, this is problematic in case of *conditional distributions*, and motivates the usage of kernels. Thus, one can define a *regular conditional distribution* of V, given $\mathfrak{F}$, as an $\mathfrak{F}$-measurable probability kernel $\kappa : \Omega \to \mathcal{V}$, such that the following equality holds for $\mathbb{P}$-almost all $\omega \in \Omega$ and all $\mathsf{B} \in \mathfrak{V}$:

$$\kappa(\omega)\{\mathsf{B}\} = \mathbb{P}\{V \in \mathsf{B} \mid \mathfrak{F}\}(\omega) .$$

More generally, given another random element U in a measurable space

$(\mathcal{U}, \mathfrak{U})$, a *regular conditional distribution* of V, given U, is a kernel from $(\Omega, \mathfrak{A}, \mathbb{P})$ to $\mathcal{V}$, of the form

$$\kappa(U)\{\mathsf{B}\} = \mathbb{P}\{V \in \mathsf{B} \mid U\} \ a.s., \quad \mathsf{B} \in \mathfrak{V},$$

where κ is a probability kernel from $\mathcal{U}$ to $\mathcal{V}$. Doing so, however, one is left with the question about the existence of such regular conditional distributions. Since σ-algebras are closed under countable unions and measures are σ-subadditive, the countable union of null sets is a null set again. Therefore, if the underlying space $\mathcal{V}$ allows for a sufficiently good *countable* approximation, one might hope to combine the null sets N_B, $\mathsf{B} \in \mathfrak{V}$, to a null set again. This line of though is one of the main reasons why Polish spaces are used ubiquitously in probability theory, and the following theorem summarizes the result, which is a specialization of Theorem 2.1.7 to probability theory:

Still, $\kappa(\cdot)\{\mathsf{B}\}$ is basically the measurable function given by the factorization lemma.

Theorem 2.1.10 *Let U and V be random elements in $\mathcal{U}$ and $\mathcal{V}$, respectively, where $\mathcal{V}$ is Polish. Then there exists a probability kernel $\kappa : \mathcal{U} \to \mathcal{V}$, such that $\mathbb{P}_{(U,V)} = \mathbb{P}_U \otimes \kappa$. Furthermore, κ has the following properties:*

(i) κ is $\mathbb{P}_U$-a.s. unique,
(ii) $\mathbb{P}\{V \in \cdot \mid U\} = \kappa(U)$ a.s.,
(iii) for every non-negative measurable function f on $\mathcal{U} \times \mathcal{V}$, it holds:

$$\mathbb{E}\{f(U,V) \mid U\} = \int_{\mathcal{V}} f(U,v)\, \kappa(U)(dv) \quad a.s.$$

Theorem 8.5 of Kallenberg (2021) or Theorem 8.37 together with Theorem 8.38 of Klenke (2013).

In the following, we will denote the kernel from Theorem 2.1.10 as $(u, \mathsf{B}) \mapsto \mathbb{P}_{V|U=u}\{\mathsf{B}\}$. Then, the iterative integration of a function f on $\mathcal{U} \times \mathcal{V}$ is again clarified by subscripts:

$$\begin{aligned}\mathbb{E}\{f(U,V)\} &= \int_{\mathcal{U}\times\mathcal{V}} f(u,v)\, \mathbb{P}_{(U,V)}(du, dv) \\ &= \int_{\mathcal{U}} \int_{\mathcal{V}} f(u,v)\, \mathbb{P}_{V|U=u}(dv)\, \mathbb{P}_U(du) = \mathbb{E}_U\left\{\mathbb{E}_{V|U=u}\{f(u,\cdot)\}\right\}.\end{aligned}$$

Lastly, because we will be concerned with sequences that are generated by the optimization algorithm and with the procedure of stopping it, the following results from the theory of stochastic processes are needed. Let T be an index set, and let $\mathcal{U}_t$, $t \in T$, be sets. For $t \in T$ we denote by

$$\pi_t := \pi_t^T : \prod_{t' \in T} \mathcal{U}_{t'} \to \mathcal{U}_t, \quad (u_{t'})_{t' \in T} \mapsto u_t$$

the *projection* onto the t-th coordinate. More generally, for subsets $J \subset I \subset T$, the map $\pi_J^I : \prod_{i\in I} \mathcal{U}_i \to \prod_{j\in J} \mathcal{U}_j$, $(u_i)_{i\in I} \mapsto (u_j)_{j\in J}$ denotes the canonical projection from the coordinates in I onto the coordinates in J. Then, if $(\mathcal{U}_t, \mathfrak{U}_t)$, $t \in T$, are actually measurable spaces, the product-σ-algebra $\bigotimes_{t\in T} \mathfrak{U}_t$ is the smallest σ-algebra on $\prod_{t\in T} \mathcal{U}_t$, such that all coordinate-projections π_t are measurable, because the preimages under the projection are exactly the cylinder sets. If all spaces $\mathcal{U}_t$ are the same, we have the following characterization of measurability:

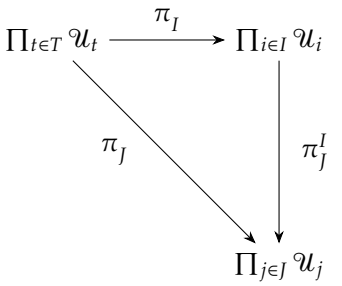

Figure 2.5: Coordinate projections for $J \subset I \subset T$.

Lemma 2.1.11 *For any measurable space $(\mathcal{U}, \mathfrak{U})$, index set T, subset $\mathsf{F} \subset \mathcal{U}^T$, and function $X : \Omega \to \mathsf{F}$, the following are equivalent:*

Lemma 4.1 of Kallenberg (2021).

(i) X *is measurable w.r.t.* $\mathsf{F} \cap \mathfrak{U}^{\otimes T}$,
(ii) $X_t = \pi_t \circ X : \Omega \to \mathcal{U}$ *is measurable w.r.t.* $\mathfrak{U}$ *for every* $t \in T$.

In this case, X is called a $\mathcal{U}$-valued *stochastic process* on T with *paths* in F, and Lemma 2.1.11 shows that one can regard X also as a collection of random elements $(X_t)_{t\in T}$ taking values in the *state space* $\mathcal{U}$. In the following, the so-called *canonical process* will be of particular importance:

Page 239 of Kallenberg (2021) or Definition 14.6 of Klenke (2013).

Definition 2.1.4 *Let* $T \neq \emptyset$ *be an index set,* $(\mathcal{U}, \mathfrak{U})$ *a measurable space, and set* $(\Omega, \mathfrak{A}) := (\mathcal{U}^T, \mathfrak{U}^{\otimes T})$. *Then, the identity map* $X := id : \Omega \to \Omega$ *is called the* canonical process *on* $(\Omega, \mathfrak{A})$.

The importance of the canonical process is due to the fact that, if we are only interested in the distribution of a process (or random variable), we can fix it as the identity and just exchange the underlying measures, which makes the treatment more flexible. In this case, we also have that $X_t = \pi_t \circ X = \pi_t$, that is, the single coordinates of the canonical process are given by the corresponding projection.

We have seen above that we can either start with a measure on the product space (joint distribution) and decompose it into a measure (marginal) and a kernel (conditional distribution), or we can construct a measure on the product space from a measure and a kernel. The following results extend this logic to countable products, that is, to the distribution of discrete-time stochastic processes.

Definition 14.40 of Klenke (2013).

Definition 2.1.5 *Let* $\mathcal{U}$ *be a Polish space, and let* $T \subset [0, \infty)$ *be a (additive) semi-group. A family of probability kernels* $(\kappa_t)_{t\in T}$ *from* $\mathcal{U}$ *to* $\mathcal{U}$ *is called a* Markovian semi-group, *if it holds* $\kappa_0(u) = \delta_u$ *for all* $u \in \mathcal{U}$, *and the* Chapman-Kolmogorov relation *does hold, that is,*

$$\kappa_s \cdot \kappa_t = \kappa_{s+t} \quad \forall s, t \in T\,.$$

Then, by iteratively applying the product-operation of kernels to all members of the Markovian semi-group, we can build a *projective family* of probability measures on the collection of product spaces $\mathcal{U}^J$, $J \subset T$ finite, that is, a family of probability measures that is "compatible" under projections. This in turn allows to apply *Kolmogorov's extension theorem* to get a unique limiting measure on the space $\mathcal{U}^T$:

Corollary 14.44 of Klenke (2013).

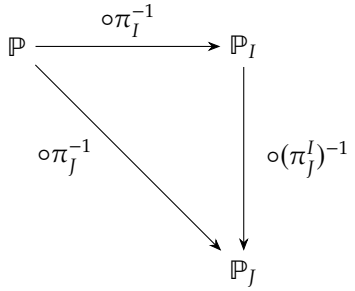

Figure 2.6: Projective family of probability measures.

Theorem 2.1.12 *Let* $(\kappa_t)_{t\in T}$ *be a Markovian semi-group on the Polish space* $\mathcal{U}$. *Then there is a unique probability kernel* γ *from* $(\mathcal{U}, \mathfrak{B}(\mathcal{U}))$ *to* $(\mathcal{U}^T, \mathfrak{B}(\mathcal{U})^{\otimes T})$ *with the following property: For every* $u \in \mathcal{U}$, *and every finite number of timepoints* $0 = t_0 < t_1 < \ldots < t_n$ *in* T, *we have for* $J := \{t_0, \ldots, t_n\}$:

$$\gamma(u) \circ \pi_J^{-1} = \delta_u \otimes \bigotimes_{k=0}^{n-1} \kappa_{t_{k+1}-t_k}\,.$$

In particular, for every probability measure μ *on* $\mathcal{U}$, *there exists a unique probability measure* $\mathbb{P}_\mu$ *on* $(\mathcal{U}^T, \mathfrak{B}(\mathcal{U})^{\otimes T})$ *with the following property: For every finite number of timepoints* $0 = t_0 < t_1 < \ldots < t_n$ *in* T, *we have for* $J := \{t_0, \ldots, t_n\}$:

$$\mathbb{P}_\mu \circ \pi_J^{-1} = \mu \otimes \bigotimes_{k=0}^{n-1} \kappa_{t_{k+1}-t_k}\,.$$

The last result addresses the problem of when a stochastic process gets stopped. A fundamental question in this context is always whether the information that got collected up to a given point in time is sufficient to decide whether the process can be stopped or not. For this, we need a filtration: Given an index set $T \subset \mathbb{R}$, a *filtration* $\mathfrak{F} = (\mathfrak{F}^{(t)})_{t\in T}$ on T is defined as a non-decreasing family of σ-algebras $\mathfrak{F}^{(t)} \subset \mathfrak{A}$, $t \in T$, that is, $\mathfrak{F}^{(s)} \subset \mathfrak{F}^{(t)}$ for $s \leq t$. Then, a stochastic process X on T is *adapted* to $\mathfrak{F}$, if X_t is $\mathfrak{F}^{(t)}$-measurable for every $t \in T$, and the smallest filtration on T with this property is given by the *induced filtration* defined through $\mathfrak{F}^{(t)} := \sigma(X_s;\ s \leq t)$. Furthermore, a random element τ in $T \cup \{\infty\}$ is called a *random time* and if the process $\mathbb{1}\{\tau \leq t\}$, $t \in T$, is adapted to the filtration $\mathfrak{F}$, that is, if $\{\tau \leq t\} \in \mathfrak{F}^{(t)}$ for every $t \in T$, it is called a *stopping time* (w.r.t. $\mathfrak{F}$).

Lemma 2.1.13 *Let τ, σ be stopping times w.r.t. a filtration $\mathfrak{F}$ on T. Then we have:*

(i) $\sigma \wedge \tau := \min\{\sigma, \tau\}$ and $\sigma \vee \tau := \max\{\sigma, \tau\}$ are stopping times.
(ii) If $\tau \equiv t \in T \cup \{\infty\}$, then τ is a stopping time.

Furthermore, in the case $T = \mathbb{N}_0$, we have: If X is an $\mathfrak{F}$-adapted process with values in a measurable space $(\mathcal{U}, \mathfrak{U})$, and $\mathsf{B} \in \mathfrak{U}$, then τ_{B}, defined as

$$\tau_{\mathsf{B}} := \inf\{t \in \mathbb{N}_0 \ :\ X_t \in \mathsf{B}\}$$

is a stopping time, the so-called hitting time *of X for the set B.*

Lemma 9.1 and Lemma 9.6 of Kallenberg (2021).

Now, being acquainted with some probability theory, we can continue with the background material on the fundamental ideas of machine learning and, especially, with the problem of generalization, which will be a major theme for the rest of this work.

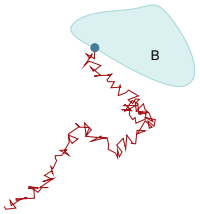

Figure 2.7: Hitting time τ_{B}.

2.2 Background on Machine Learning

As discussed, for example, by Richter (2019), approaches in machine learning are traditionally divided into three categories:

(i) *Supervised Learning*: It is assumed that there is a probability space $(\Omega, \mathfrak{A}, \mathbb{P})$ and some random variables $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{U}$, the *features*, which serve as input variable, and $V : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{V}$, the *labels*, which serve as target variable. The goal is to determine a relationship between U and V, ideally in terms of a (deterministic) map[3] $f^* : \mathcal{U} \to \mathcal{V}$, such that $f^*(U) = V$. To do this, a *loss-function* $\ell : \mathcal{V} \times \mathcal{V} \to [0, \infty)$ and a *model* $\mathcal{F} \subset \{f : \mathcal{U} \to \mathcal{V} \ :\ f \text{ is measurable}\}$ have to be specified: The model determines which functions are considered as possible candidates for the solution, and the loss-function measures the discrepancy between the true labels and the corresponding prediction. Then, one tries to solve $\inf_{f\in\mathcal{F}} \mathcal{R}(f)$, where the so-called *risk* is given by $\mathcal{R}(f) = \mathbb{E}\{\ell(f(U), V)\}$. However, almost always this is infeasible, such that one resorts to solving the approximate problem $\inf_{f\in\mathcal{F}} \hat{\mathcal{R}}(f, S)$, where the *empirical risk* $\hat{\mathcal{R}}(f, S) = \frac{1}{N}\sum_{i=1}^{N} \ell(f(U_i), V_i)$ is computed on a *training data set* of labeled input-output pairs $S = ((U_1, V_1), \ldots, (U_N, V_N))$, typically assumed to be i.i.d. from $\mathbb{P}_{(U,V)}$.

(ii) *Unsupervised Learning*: Instead of having input-output pairs, only unlabeled training data $U_1, \ldots, U_N \overset{iid}{\sim} \mathbb{P}_U$ is given, and the goal is

3: More generally, one can try to find the conditional distribution $\mathbb{P}_{V|U}$, and the ideas stay mostly the same.

to gain *structural knowledge* about $\mathbb{P}_U$ from these observations, that is, structure that cannot be attributed to random noise alone.

(iii) *Reinforcement Learning*: While supervised and unsupervised learning are quite similar in their overall modeling, reinforcement learning takes a fundamentally different point of view: It considers a so-called *agent*, that interacts with a given *environment* and tries to learn how to make *decisions* in order to achieve a specified goal, typically formulated in terms of a *reward*. This generates a sequence of states and corresponding actions of the agent, which can be summarized into a *Markov decision process*. Formally, this can be described by a tuple $(\mathcal{X}, \mathcal{A}, \gamma, r, \mathbb{P}_{Z^{(0)}})$ consisting of a *state space* $\mathcal{X}$, an *action space* $\mathcal{A}$, a *transition kernel* $\gamma : \mathcal{X} \times \mathcal{A} \times \mathfrak{B}(\mathcal{X}) \to [0,1]$, a *reward* function $r : \mathcal{X} \times \mathcal{A} \times \mathcal{X} \to \mathbb{R}$, and an *initial distribution* $\mathbb{P}_{Z^{(0)}}$.

Nowadays, however, there is a growing level of specialization, with fields becoming increasingly nuanced, yet showing considerable overlap. One of these more nuanced areas, namely *self-supervised learning*, is also of interest for this study: Instead of relying on the labels V_i, the unlabeled training data U_i is used to generated a supervisory signal.

Remark 2.2.1 Often, the model $\mathcal{F}$ is *parametric*, that is, there is a bivariate function $f : \mathcal{U} \times \mathcal{P} \to \mathcal{V}$, where $\mathcal{P}$ is a set of parameters, such that

$$\mathcal{F} = \{f(\cdot, p) \,:\, p \in \mathcal{P}\}\,,$$

and we will also make use of this parametric setting later on.

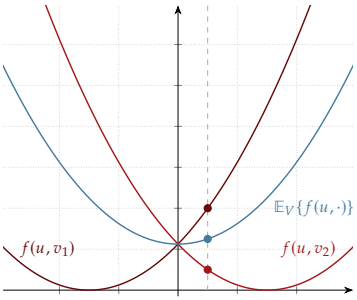

Figure 2.8: Visualization for the empirical risk as unbiased estimator in the setting of $f(u,v) = (u-v)^2$ and $V \sim 0.5\delta_{v_1} + 0.5\delta_{v_2}$.

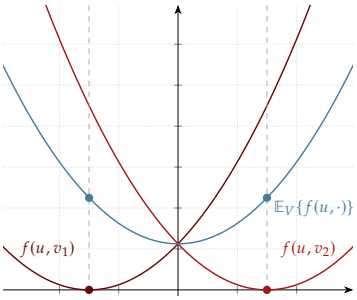

Figure 2.9: Visualization for the empirical risk as biased estimator, that is, after training, in the setting of $f(u,v) = (u-v)^2$ and $V \sim 0.5\delta_{v_1} + 0.5\delta_{v_2}$. In this case, we would have $\mathbb{E}\{\hat{\mathcal{R}}(\hat{f}(S), S)\} \equiv 0$, while the corresponding risk $\mathcal{R}(\hat{f})$ would be strictly greater than zero.

The idea of using the empirical risk as approximation to the true risk is based on the fact that it is an *unbiased estimator* for the risk, that is:

$$\begin{aligned}\mathbb{E}\{\hat{\mathcal{R}}(f,S)\} &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\{\ell(f(U_i),V_i)\} \\ &= \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}\{\ell(f(U),V)\} = \mathcal{R}(f)\,.\end{aligned}$$

However, this kind of reasoning is only valid, as long as f is independent of S. Thus, it cannot be applied for the function selected during training: Assuming that $\hat{f}(S) = \arg\min_{f\in\mathcal{F}} \hat{\mathcal{R}}(f,S)$, clearly it holds that:

$$\mathbb{E}\{\hat{\mathcal{R}}(\hat{f}(S),S)\} \le \mathbb{E}\{\hat{\mathcal{R}}(f,S)\} = \mathcal{R}(f)\,,$$

and, depending on the given setup, the gap between $\mathbb{E}\{\hat{\mathcal{R}}(\hat{f}(S),S)\}$ and $\mathcal{R}(f)$ can be arbitrarily large. Thus, it is important to know whether the observed performance during training is actually *representative* for the performance on unseen problems, that is, whether it *generalizes*, and this is where the PAC-Bayesian framework comes into play.

2.2.1 The PAC-Bayesian Approach to Learning

PAC is an acronym for *Probably Approximately Correct* and refers to the fact that it allows to give high-probability bounds on the risk which (should) get tight if the size of the training data set increases. The original PAC-framework is based on an *a priori* worst-case analysis of the model $\mathcal{F}$, such that it often leads to poor estimates of the generalization performance, that is, loose bounds, since $\mathcal{F}$ has to be chosen large enough

to yield a satisfactory performance at all[4] (Shawe-Taylor et al., 1997). This motivated the original works on a PAC-*Bayesian* analysis due to Shawe-Taylor et al. (1997), McAllester (1998), and McAllester (1999), which, in contrast, provides *a posteriori* estimates based on the model *and* the training data. Since then, the PAC-Bayesian framework has been applied to various problems in machine learning. Some examples include general classification problems (Langford et al., 2001; Seeger, 2002; Langford et al., 2002; Catoni, 2004; Catoni, 2007), support-vector machines (Ambroladze et al., 2006; Parrado-Hernández et al., 2012), majority-vote algorithms (Lacasse et al., 2006; Laviolette et al., 2011; Masegosa et al., 2020), density estimation (Higgs et al., 2010), least-squares estimation and linear classifiers (Audibert et al., 2011; Honorio et al., 2014), multi-armed bandits and martingales (Seldin et al., 2012a; Seldin et al., 2012b), life-long and online learning (Pentina et al., 2014; Haddouche et al., 2022), stochastic optimization (London, 2017), meta-learning (Amit et al., 2018; Rothfuss et al., 2021), random forests (Lorenzen et al., 2019), representation learning (Nozawa et al., 2020), Monte-Carlo estimation (Ohnishi et al., 2021), variational auto-encoders (Chérief-Abdellatif et al., 2022), stochastic neural networks (Biggs et al., 2021; Clerico et al., 2023), and generative models (Mbacke et al., 2023). At the same time, also many general results about the PAC-Bayesian learning approach or how to resolve some of its initial limitations have been published. For example, this includes the relation of PAC-Bayesian bounds to other generalization bounds in statistical learning theory and how to combine it with generic chaining (Audibert et al., 2007), more abstract PAC-Bayesian bounds comprising many (at the time) known bounds (Germain et al., 2009), computationally less expensive variational approximations of the Gibbs posterior (Alquier et al., 2016), the connection between PAC-Bayesian and standard Bayesian inference (Germain et al., 2016), its extension from bounded loss-functions to heavy-tailed losses (Holland, 2019), the case of training data that has temporal dependencies modeled by a Markov chain (Banerjee et al., 2021), the use of data-dependent priors to get tighter bounds (Parrado-Hernández et al., 2012; Dziugaite et al., 2021), loss-functions that can be bounded in a parameter-dependent way (Haddouche et al., 2021), and training with small data sets (Foong et al., 2021). For a more detailed discussion we refer to the review papers by Guedj (2019), Hellström et al. (2025), and Alquier (2024).

4: This is basically the same dilemma as discussed in the introduction.

Change-of-Measure Inequalities

As the name "Bayesian" indicates, these bounds deal with measures instead of points. In particular, to be applicable at all, an additional distribution is needed: The so-called *prior* characterizes how likely each candidate in the model $\mathcal{F}$ is before training. Then, the key ingredient for the resulting generalization bound is a *change-of-measure inequality*, which quantifies the discrepancy between the initial prior and the resulting *posterior*, which gets chosen based on the observations during training. Thus, this change-of-measure inequality implicitly determines how much importance should be given to the observations during training, such that its choice strongly influences the corresponding bound. The one used most often is based on a variational representation of the Kullback-Leibler divergence due to Donsker et al. (1975), employed, for example, by Catoni (2004) and Catoni (2007). Yet, not all bounds are based on a variational representation, that is, holding uniformly over all posterior distributions (Rivasplata et al., 2020). Similarly, while many bounds involve the Kullback-Leibler divergence as measure of proximity (McAllester, 2003b;

McAllester, 2003a; Seeger, 2002; Langford et al., 2002; Germain et al., 2009), other divergences have been used: Honorio et al. (2014) prove an inequality for the χ^2-divergence, which is also used by London (2017). Bégin et al. (2016) and Alquier et al. (2018) use the Rényi-divergence (α-divergence). Ohnishi et al. (2021) propose PAC-Bayesian bounds based on general f-divergences, which include the Kullback-Leibler-, α- and χ^2-divergence. More recently, Amit et al. (2022) propose to replace the Kullback-Leibler divergence by so-called "integral probability metrics", which encompass, for example, the Wasserstein distance that obeys many favorable properties and also captures the geometry of the underlying space (see Villani, 2009). Motivated by this, Haddouche et al. (2023) also investigate PAC-Bayesian generalization bounds for the Wasserstein distance and their interplay with the output of optimization algorithms. A major advantage of using the Wasserstein distances instead of the Kullback-Leibler divergence is the fact that it does not constrain the support of the distribution a-priori through the choice of the prior. On the other hand, it demands assumptions on the loss function, which are not necessarily satisfied in learning-to-optimize.

Bounded Loss-Functions

A major drawback of most of the existing PAC-Bayesian bounds is the assumption that the loss-function is bounded, typically normalized to the interval $[0, 1]$. Historically, this is due to the fact that most of the time classification problems have been considered. Nowadays, however, this assumption is mainly used to apply some exponential-moment inequality like the Hoeffding- or Bernstein-inequality (Rivasplata et al., 2020; Alquier, 2024) and several ways have been developed to circumvent this problem: Catoni (2004) restricts to the case of sub-Gaussian or sub-Gamma random variables, Germain et al. (2009) explicitly include the exponential-moment in the bound, and Alquier et al. (2016) use so-called Hoeffding- and Bernstein-assumptions. Alternatively, if the procedure is applied to a specific kind of algorithm, another possibility to ensure the generalization or exponential-moment bounds is to use specific properties of exactly this algorithm. For example, London (2017) uses algorithmic stability to provide PAC-Bayesian bounds for stochastic gradient descent.

Learning with PAC-Bayesian Bounds

PAC(-Bayesian) bounds relate the true risk to other terms such as the empirical risk, that is, they make a statement about how the risk *compares* to the empirical risk rather than stating whether these terms are actually small. Thus, learning procedures based on the PAC-Bayesian theory typically aim to minimize this upper bound: Langford et al. (2001) compute non-vacuous generalization bounds through a combination of PAC-bounds with a sensitivity analysis. Dziugaite et al. (2017) extend this by minimizing the PAC-bound directly. Pérez-Ortiz et al. (2021) also consider learning as minimization of the PAC-Bayesian bound and provide tight generalization bounds for neural-network based classification. Thiemann et al. (2017) are able to solve the minimization problem resulting from their PAC-bound by alternating minimization.

That is, the idea is basically that of a majorization-minimization procedure.

The Choice of the Prior

A common difficulty in learning with PAC-Bayesian bounds is the choice of the prior distribution, as it heavily influences the performance of the learned models and the generalization bound (Catoni, 2004; Dziugaite et al., 2021; Pérez-Ortiz et al., 2021). In part, and especially for the Kullback-Leibler divergence, this is due to the fact that the divergence term can dominate the bound, keeping the posterior close to the prior. This lead to the idea of choosing a data- or distribution-dependent prior (Seeger, 2002; Parrado-Hernández et al., 2012; Lever et al., 2013; Dziugaite et al., 2018; Pérez-Ortiz et al., 2021), which, by using an independent subset of the data set, gets optimized to yield a good performance.

Recalling Figure 1.2, by now we have covered the background material for the right branch, that is, the background on stochastics and learning theory. Next, we discuss the left branch, which deals with mathematical optimization and its primarily geometric point of view.

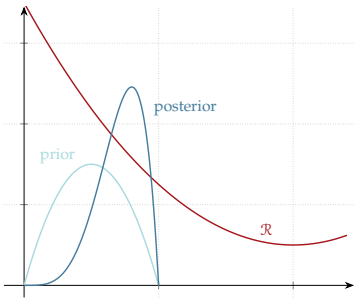

Figure 2.10: Visualization for why the choice of the prior distribution is crucial: To have a tight generalization bound, the posterior has to stay close to the prior. However, if the support of the prior is not chosen well-enough, the performance of the learned algorithm will not be good.

2.3 Background on Mathematical Optimization

"Nothing takes place in the world whose meaning is not that of some maximum or minimum." - *Leonhard Euler*

This section summarizes the main notions from the field of mathematical optimization. Again, these are standard in the corresponding literature and can be found, for example, in the books by Nocedal et al. (2006), Bertsekas (1999), or Nesterov (2018). Similarly, the material from variational analysis is based on the book by Rockafellar et al. (1998).

Mathematical optimization is the science of maximizing of minimizing functions, possibly under a given set of constraints. Since minimizing a function f is equivalent to maximizing $-f$, we will focus only on minimization problems:

$$\min_{x \in \mathsf{C}} f(x) .$$

Here, the function $f : \mathcal{X} \to \mathbb{R}$ is called the *objective function*, $\mathcal{X}$ is called the *optimization space*, $x \in \mathcal{X}$ is the optimization variable, and C is the *constraint set*. If $\mathsf{C} = \mathcal{X}$, the problem is said to be *unconstrained*.
A *global solution* to such a minimization problem is given by a point $x^* \in \mathsf{C}$ that attains the *optimal value* $f(x^*) = \inf f := \inf_{x \in \mathsf{C}} f(x)$, that is, $f(x^*) \leq f(x)$ for all $x \in \mathsf{C}$, and we denote the set of all globally optimal solutions by $\arg\min f := \arg\min_{x \in \mathsf{C}} f(x)$. If the condition $f(x^*) \leq f(x)$ only does hold locally, x^* is called a *local solution*.
To solve optimization problems, an *optimization algorithm* $\mathcal{A}$ is used. Typically, this is an iterative method that, starting from an initial guess $x^{(0)}$, generates a sequence $(x^{(t)})_{t \in \mathbb{N}_0} \subset \mathcal{X}$, which should approximate some $x^* \in \arg\min f$. However, in full generality, optimization is an unsolvable problem: There is not a single algorithm that can solve all optimization problems, and there are optimization problems on which no algorithm performs better than random guessing. Such *no-free-lunch* results are known in many areas of mathematics and computer science, and a common remedy is to provide more information by considering a more restrictive setting, that is, by making more assumptions. In the case of optimization, this is done by specifying a *class of problems*, which consists of a *model*, an *oracle*, and a *stopping criterion*. Here, the model specifies the common properties of problems within the class (and therefore also excludes pathological cases), the oracle specifies the information that the algorithm can access during run-time, and the stopping criterion specifies the goal of the algorithm, that is, the properties a given point

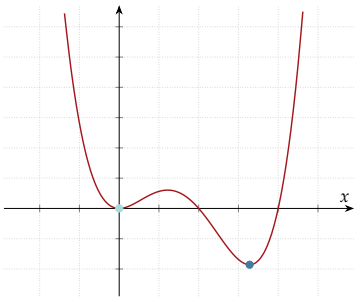

Figure 2.11: Local and global solution.

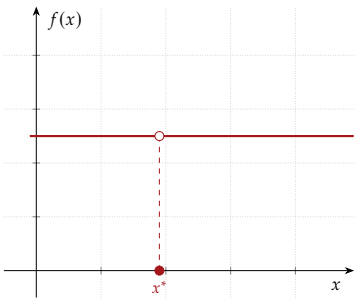

Figure 2.12: Without any knowledge about the solution x^*, no algorithm performs better than random guessing.

should satisfy in order to qualify as a solution to the problem.
The specification of a model also allows for comparing different algorithms based on their *performance*: A typical performance-measure is given through the *analytical complexity* of $\mathcal{A}$, that is, how often the oracle has to be called to find a point satisfying the stopping criterion. Using this, the performance of an algorithm is determined by means of a *worst-case analysis*, which provides an upper-bound on the number of calls to the oracle over all problem-instances inside the class. Here, if the stopping criterion is specified in terms of a *residual* $r^{(k)}$, complexity can also be expressed in terms of a *convergence rate*, which quantifies how fast (in the worst case) the residual converges to zero. However, the performance of an algorithm in itself does not directly indicate whether the result is actually satisfactory for the respective class. For this, also the *lower complexity bound* is needed, which is the best analytic complexity that any algorithm[5] can achieve on the given class. Only if the analytic complexity of $\mathcal{A}$ coincides (up to constant factors) with the lower complexity bound, the algorithm $\mathcal{A}$ is said to be *optimal*.

5: Note that the class also determines which information an algorithm can access.

2.3.1 Smooth Unconstrained Optimization

In many cases, finding a global solution to the minimization problem is not feasible. Actually, verifying whether a given point is a solution or not can already be problematic. Hence, easily verifiable conditions are needed. In the case of unconstrained minimization over $\mathcal{X} = \mathbb{R}^d$, such conditions can be characterized in terms of the (higher-order) *differential*:

A detailed discussion of the differential is given by Königsberger (2006).

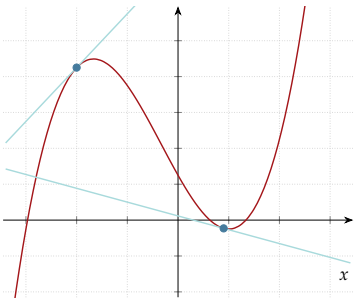

Figure 2.13: The function $x \mapsto f(\bar{x}) + \langle \nabla f(\bar{x}), x - \bar{x} \rangle$ provides the (locally) best linear approximation of f.

Definition 2.3.1 *A function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be* differentiable *in $\bar{x}$, if there is a linear map $L : \mathbb{R}^d \to \mathbb{R}$, such that*

$$\lim_{h \to 0} \frac{f(\bar{x} + h) - f(\bar{x}) - L(h)}{\|h\|} = 0 \,.$$

The uniquely defined linear map L is called the differential *of f at $\bar{x}$, and it is written as $\mathrm{d}f(\bar{x})$.*

The defining condition can equivalently be represented as

$$f(\bar{x} + h) - f(\bar{x}) = \mathrm{d}f(\bar{x})h + r(h)$$

with the condition that $\lim_{h \to 0} \frac{r(h)}{\|h\|} = 0$, usually abbreviated as $r(h) \in o(\|h\|)$ or simply $o(\|h\|)$. Now, if $\mathbb{R}^d$ is endowed with a scalar product $\langle \cdot, \cdot \rangle$, every *linear form* $L : \mathbb{R}^d \to \mathbb{R}$ can uniquely be represented as $L(h) = \langle g, h \rangle$, $h \in \mathbb{R}^d$. Thus, if L is the differential of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ in $\bar{x}$, the vector g is called the *gradient* of f in $\bar{x}$ w.r.t. $\langle \cdot, \cdot \rangle$, written as $\mathrm{grad} f(\bar{x})$. Therefore, the gradient of f in $\bar{x}$ is uniquely determined by the condition $\mathrm{d}f(\bar{x})h = \langle \mathrm{grad} f(\bar{x}), h \rangle$, and in case of the (standard) Euclidean scalar product, one can show that it is given by the vector of *partial derivatives* $\partial_i f(\bar{x})$, $i = 1, \ldots, p$, written as $\nabla f(\bar{x})$. More generally, if f is two-times continuously differentiable in $\bar{x}$, we can generalize the differential to the bilinear form $\mathrm{d}^{(2)} f(\bar{x})(x, y) := \partial_x(\partial_y f)(\bar{x})$, and one can show that it can be represented through the *Hessian matrix* $\nabla^2 f(\bar{x}) := (\partial_{ij} f(\bar{x}))_{i,j=1,\ldots,p}$, satisfying the equality $\mathrm{d}^{(2)} f(\bar{x})(x, y) = x^T \nabla^2 f(\bar{x}) y$. Then, both a simple necessary and a simple sufficient condition for optimality can be given:

Lemma 2.3.1 *Let x^* be a local solution to the problem $\min_{x\in\mathbb{R}^d} f(x)$. If f is differentiable at x^*, then it holds that $\nabla f(x^*) = 0$. Conversely, if f is twice continuously differentiable at a point x^* with $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ being positive definite, then x^* is a local minimizer of f.*

For example, see Theorem 1.2.1 and Theorem 1.2.3 of Nesterov (2018).

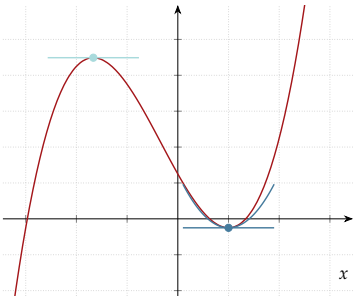

Figure 2.14: Visualization of optimality conditions.

Especially, the set of solutions is contained in the set of *stationary* or *critical points*, that is, points $x \in \mathbb{R}^d$ with $\nabla f(x) = 0$.
To find such a stationary point, an optimization algorithm is employed and, as stated before, usually this is an iterative method that generates a sequence by updating the current estimate of the solution step by step. Thus, from an abstract point of view many optimization algorithms can be written in the following simple form:

$$x^{(t+1)} = x^{(t)} + \alpha^{(t)} d^{(t)} .$$

In this context, $d^{(t)} \in \mathbb{R}^d$ is the *update-direction* and $\alpha^{(t)} > 0$ is referred to as the *step-size* (or *learning-rate*), which is often considered to be a so-called *hyperparameter*, that is, a parameter that has to be chosen by the user to optimize the performance of the method.
The most prominent example of such an algorithm is *gradient descent* , which chooses $d^{(t)} = -\nabla f(x^{(t)})$ and, in the following, we will use it to showcase how the theoretical analysis, and therefore also the provided guarantees for an optimization algorithm, is influenced by properties of the objective function, and how this could potentially change, if more information through observation would be accessible. The starting-point are quadratic functions of the form:

Based on its simplicity, we will come back to gradient descent over and over throughout the remainder of the text to illustrate ideas or problems.

$$f(x) := \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + c ,$$

where $A \in \mathbb{R}^{d\times d}$ is assumed to be a *symmetric* and *positive definite* matrix, $b \in \mathbb{R}^d$, and $c \in \mathbb{R}$. In this case, the gradient of f is given by $\nabla f(x) = Ax - b$, such that gradient descent can be written as $x^{(k+1)} = x^{(k)} - \alpha(Ax^{(k)} - b)$. Since A is symmetric positive definite, it is invertible and the unique solution to the optimization problem can be found through $\nabla f(x) = 0$, which is satisfied for $x^* = A^{-1}b$. Thus, $Ax^* = b$ and we get the following recursion for the residual:

$$r^{(k+1)} := x^{(k+1)} - x^* = x^{(k)} - x^* - \alpha A(x^{(k)} - x^*) = (\mathbb{1} - \alpha A)(x^{(k+1)} - x^*) .$$

Therefore, $\|x^{(k+1)} - x^*\| \leq \|\mathbb{1} - \alpha A\| \|x^{(k+1)} - x^*\|$, and $\mathcal{A}$ converges with a *linear rate*, that is, the relative gain in accuracy per iteration is constant along the iterations, as soon as there exists $\rho \in (0, 1)$ with $\|\mathbb{1} - \alpha A\| \leq \rho$. By analyzing the *spectrum* of A, one can show that this is the case for $\alpha \in (0, \frac{2}{L})$ and that the best rate gets attained at $\alpha = \frac{2}{L+\mu}$, where L and μ are the largest and smallest eigenvalue of A, respectively.
However, it is important to note that this choice is only optimal in terms of the worst-case analysis and, if more information is available, the truly optimal choice might differ substantially. For example, if the algorithm would only do a single step, exact line search would almost always be better. Similarly, if we knew more about the initial point $x^{(0)}$, other choices can potentially yield convergence in one step, as Figure 2.15 illustrates. Furthermore, if there would be more information about the matrix A, the design of the algorithm could also be adapted. For example, if A is actually a (block) diagonal matrix, incorporating a *preconditioner* into the update might improve the performance tremendously.
While these examples are somewhat pathological, they still highlight that

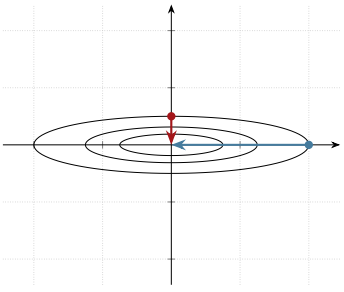

Figure 2.15: Hypothetical example of how the step-size would change, if we would know more about the initial point: While both points are on the same level-line, the respective gradients differ strongly, such that the truly optimal step-sizes differ by a factor of four. Similarly, the worst-case optimal step-size is actually suboptimal in both cases.

there can be a substantial difference between (truly) optimal and worst-case optimal choices, depending on how much additional information is present. Nevertheless, without such additional information, to prevent *divergence* of the algorithm in any case, the step-size has to be chosen small enough related to the *smoothness parameter* of f, defined as the *Lipschitz-constant* of the gradient:

Definition 2.3.2 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be continuously differentiable. If ∇f is L-Lipschitz continuous, that is, $\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|$ for all $x, y \in \mathbb{R}^d$, f is said to be* L-smooth.

The following result, know as the *descent lemma*, is a widely-used tool in the convergence analysis of optimization algorithms on L-smooth functions, because it provides a quadratic *minorant* and a quadratic *majorant* around every point:

See, for example, Theorem 2.1.5 of Nesterov (2018).

Lemma 2.3.2 *Let f be L-smooth. Then we have for every $\bar{x} \in \mathbb{R}^d$:*

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

This property can be used in so-called *majorization-minimization* techniques, that is, in each iteration a majorant to the objective function is minimized, such that the value of the objective function has to decrease, too. Similarly, *strong convexity* provides a global quadratic minorant to the objective function around every point:

Definition 2.1.3 of Nesterov (2018).

Definition 2.3.3 *A continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be* strongly convex *with modulus $\gamma > 0$, if*

$$f(x) \geq f(\bar{x}) + \langle \nabla f(\bar{x}), x - \bar{x} \rangle + \frac{\gamma}{2}\|x - \bar{x}\|^2, \quad x, \bar{x} \in \mathbb{R}^d .$$

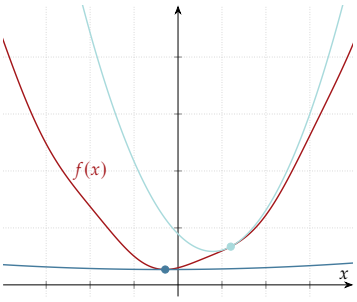

Figure 2.16: The function $f(x) = 2x^2 + 3.9\sin(x) + 15$ is 7.9-smooth and 0.1-strongly convex.

Especially, if x^* is a stationary point of f, we have $f(x) \geq f(x^*) + \frac{\gamma}{2}\|x - x^*\|^2$, which implies that x^* is the unique global minimizer of f. Then, if $f : \mathbb{R}^d \to \mathbb{R}$ is not quadratic, yet still μ-strongly convex and L-smooth, it can be shown that the iterates of gradient descent converge linearly to the global optimum with the same rate parameter $\rho = \left(\frac{L-\mu}{L+\mu}\right)$. However, despite converging linearly, gradient descent is not an optimal *first-order* method for this class of problems: In terms of the rate, the lower complexity bound for first-order methods on μ-strongly convex and L-smooth functions is given by $\rho^* = \left(\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)$, and it is attained by the *heavy-ball algorithm*[6] (HBF; for example, see Polyak, 1964; Polyak, 1987), which performs an update of the following form:

$$x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}) + \beta(x^{(t)} - x^{(t-1)}) .$$

Thus, the heavy-ball algorithm is an optimal first-method for μ-strongly convex and L-smooth functions, which, however, comes with an additional memory overhead through the *momentum term* $m^{(t)} := x^{(t)} - x^{(t-1)}$. Nevertheless, while HBF is worst-case optimal and gradient descent is not, by considering the example of Figure 2.15 again, we can see that gradient descent can still be faster than HBF: If both algorithms would start on one of the two axes (with the corresponding choice of the step-size

6: Sometimes also called *heavy-ball with friction*.

parameter), gradient descent would converge in a single step to the global minimum, while HBF would continue to move through the existence of the momentum term and thus converge only later. While, again, this is a pathological example, it shows that the ranking of algorithms can change as soon as more information is present.

Intuitively, the reason why gradient descent is still able to obtain a linear rate is given by the fact that the strong-convexity parameter prevents the objective function from becoming arbitrarily flat. This is not the case, if the objective function is merely *convex*:

Definition 2.3.4 *A continuously differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is said to be* convex, *if*

$$f(x) \geq f(\bar{x}) + \langle \nabla f(\bar{x}), x - \bar{x} \rangle, \quad x, \bar{x} \in \mathbb{R}^d .$$

Definition 2.1.2 of Nesterov (2018). More generally, one can define convex functions via their *epigraph* (see, for example, Theorem 3.1.2 of Nesterov, 2018). Since this is not needed in the following, we restrict to this more simple setting here.

Again, if x^* is a stationary point, we have $f(x^*) \leq f(x)$ for all $x \in \mathbb{R}^d$, such that x^* is a global solution. However, uniqueness cannot be deduced anymore.
In general, if the objective f is L-smooth, a gradient step with step-size $\alpha = \frac{1}{L}$ leads to a decrease in function-values:

$$\begin{aligned} f(x^{(t+1)}) &\leq f(x^{(t)}) + \langle \nabla f(x^{(t)}), x^{(t+1)} - x^{(t)} \rangle + \frac{L}{2} \|x^{(t+1)} - x^{(t)}\|^2 \\ &= f(x^{(t)}) - \frac{1}{2L} \|\nabla f(x^{(t)})\|^2 . \end{aligned}$$

Thus, by summing these inequalities, we find that the norm of the gradient actually converges to zero:

$$\frac{1}{2L} \sum_{t=0}^{T} \|\nabla f(x^{(t)})\|^2 \leq f(x^{(0)}) - f(x^{(T+1)}) < \infty .$$

However, in general, one can only show a *sublinear rate* of convergence, that is, the gain in accuracy per iteration decreases over time, as follows: Consider the function $g(x) := f(x^{(t)}) + \langle \nabla f(x^{(t)}), x - x^{(t)} \rangle + \frac{L}{2} \|x - x^{(t)}\|^2$. It is L-strongly convex and attains its unique minimizer at

$$\nabla g(x) = 0 \iff x = x^{(t+1)} = x^{(t)} - \frac{1}{L} \nabla f(x^{(t)}) .$$

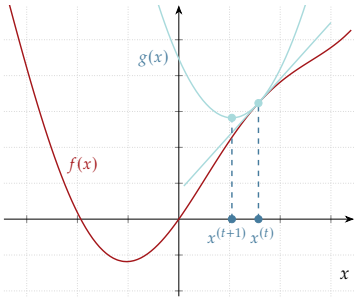

Figure 2.17: One step of gradient descent minimizes the quadratic majorant g.

Thus, we have that $g(x^{(t+1)}) + \frac{L}{2} \|x - x^{(t+1)}\|^2 \leq g(x)$ for all $x \in \mathbb{R}^d$, which is equivalent to

$$\begin{aligned} &f(x^{(t)}) + \langle \nabla f(x^{(t)}), x^{(t+1)} - x^{(t)} \rangle + \frac{L}{2} \|x^{(t+1)} - x^{(t)}\|^2 + \frac{L}{2} \|x - x^{(t+1)}\|^2 \\ &\leq f(x^{(t)}) + \langle \nabla f(x^{(t)}), x - x^{(t)} \rangle + \frac{L}{2} \|x - x^{(t)}\|^2 . \end{aligned}$$

Here, the left-hand side can be bounded by the descent lemma and, if additionally f is convex, the right-hand side by convexity. Thus, in this case we get:

$$f(x^{(t+1)}) + \frac{L}{2} \|x - x^{(t+1)}\|^2 \leq f(x) + \frac{L}{2} \|x - x^{(t)}\|^2 .$$

Therefore, if there exists $x^* \in \arg\min f$, we conclude that

$$f(x^{(t+1)}) - f(x^*) \leq \frac{L}{2} \left(\|x^* - x^{(t)}\|^2 - \|x^* - x^{(t+1)}\|^2 \right) .$$

Again, by summing these terms and using the fact that $(f(x^{(t)}))_{t\in\mathbb{N}_0}$ is non-increasing, we find that

$$0 \leq T\left(f(x^{(T)}) - f(x^*)\right) \leq \sum_{t=0}^{T-1}(f(x^{(t)}) - f(x^*)) \leq \frac{L}{2}\|x^{(0)} - x^*\|^2\,,$$

which shows that gradient descent converges (in function values) with a sublinear rate. Yet, also here, while it is not possible to prove that gradient descent converges with a linear rate on convex functions, it might be possible to observe it during execution: If the region in which gradient descent moves is actually strongly convex, it does not really matter if the function globally is merely convex and, if we would have this additional information, it would influence our choice of the corresponding hyperparameters such as the step-size. Furthermore, the analysis also shows that the whole procedure relies strongly on the knowledge of the smoothness-constant L. If it is not known, however, choosing the correct step-size α is much harder, such that a more conservative choice has to be used, which makes the algorithm even slower.

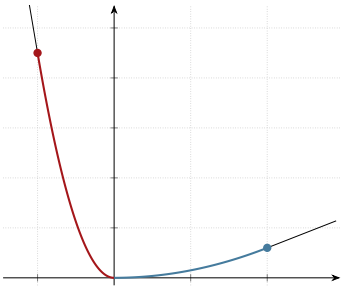

Figure 2.18: Hypothetical example of linear convergence of gradient descent on a convex function: While the whole function is merely convex, the regions between the starting points and the minimizer are both times strongly convex. Thus, while we can only prove a sublinear rate, we could *observe* a linear rate.

Again, gradient descent is not an optimal first-method for the class of L-smooth and convex functions, because the lower complexity bound (in terms of the rate) is given by $\frac{1}{t^2}$ and is attained by *Nesterov's accelerated gradient descent* (NAG; see Nesterov, 1983), which performs the following two-step update:

$$y^{(t)} = x^{(t)} + \frac{a^{(t)} - 1}{a^{(t+1)}}(x^{(t)} - x^{(t-1)})\,,$$
$$x^{(t+1)} = y^{(t)} - \alpha\nabla f(y^{(t)})\,.$$

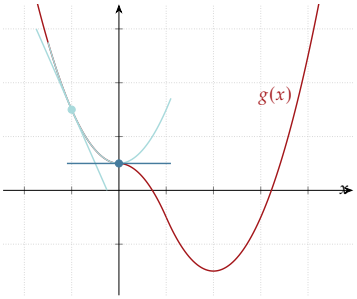

Figure 2.19: Gradient descent on the L-smooth function $g(x)$: Starting at $x^{(0)} = -1$, it can converge in one step. However, the resulting point is only stationary, and not even a local minimum.

Nevertheless, as Figure 2.18 exemplifies, the actual *observable* performance might differ strongly from the provable, and the standard choice $\alpha = \frac{1}{L}$ might in reality be far from optimal, depending on the region in the optimization space that is actually of interest.
Finally, if the objective is merely L-smooth, we can use the same reasoning as before to deduce that the function values are non-increasing, and that the norm of the gradient converges to zero. However, we can neither deduce that the sequence of iterates converges, nor that the function values actually converge to the optimal value.

2.3.2 Variational Analysis and Non-Smooth Optimization

For now, the discussion was focused on minimizing unconstrained differentiable objectives. However, not all minimization problems are unconstrained nor sufficiently often differentiable. A common "trick" to (analytically) deal with constrained optimization problems is to consider the *extended real numbers* $\overline{\mathbb{R}} := [-\infty, +\infty]$, equipped with the usual calculation rules.[7] The idea is to simply assign a value of $+\infty$ to all points outside the constraint set, such that, if the objective function does attain finite values inside the constraint, any point outside cannot be a minimizer. The following function is useful in this context:

7: Two conventions have to be kept in mind: First, $0 \cdot \pm\infty = 0$, which is consistent with measure-theory. Second, expressions of the form "$\infty - \infty$" have to be avoided, as they are not defined.

Definition 2.3.5 *For a set* $\mathsf{C} \subset \mathbb{R}^d$*, the corresponding* indicator function ι_{C} *is defined as*

$$\iota_{\mathsf{C}}(x) = \begin{cases} 0, & x \in \mathsf{C}\,, \\ +\infty, & \textit{otherwise}\,. \end{cases}$$

This definition can be found on page 6 of Rockafellar et al. (1998).

Remark 2.3.1 We decided to keep the name "indicator function" for both optimization theory and probability theory, despite the fact that the definitions differ slightly. This is due to the fact that these are well-established names in their corresponding fields, and a confusion is not to be expected. Basically, they are related through $\mathbb{1}_{\mathsf{C}} = \exp(-\iota_{\mathsf{C}})$.

As outlined above, instead of minimizing a function $\bar{f}$ over a set $\mathsf{C} \subset \mathbb{R}^d$, one can then also minimize the function

$$f(x) := \bar{f}(x) + \iota_{\mathsf{C}}(x) = \begin{cases} \bar{f}(x), & x \in \mathsf{C}\,, \\ +\infty, & \text{otherwise} \end{cases}$$

over the whole space $\mathbb{R}^d$. Thus, in the following, we will consider only unconstrained minimization problems for functions f taking values in the extended real numbers. In this context, the *(effective) domain* of a function $f : \mathbb{R}^d \to \mathbb{R}$ is given by $\operatorname{dom} f := \{x \in \mathbb{R}^d \,:\, f(x) < +\infty\}$. Further, f is said to be *proper*, if $\operatorname{dom} f \neq \emptyset$ and we have $f(x) > -\infty$ for all $x \in \mathbb{R}^d$. Figure 2.20 shows that, whether such a function f actually attains a minimizer now also depends crucially on the set C: Only if the left boundary point x_L of C does belong to C again, the minimum will be attained, such that it holds $f(x_L) \leq \liminf_{x \to x_L} f(x)$. More generally, *lower semi-continuity* is important for the attainment of minimizers:

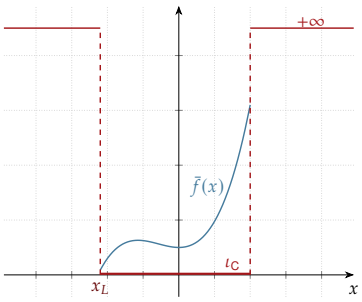

Figure 2.20: The function $f = \bar{f} + \iota_{\mathsf{C}}$.

Definition 2.3.6 *A function* $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ *is* lower semi-continuous *at* $\bar{x}$*, if* $\liminf_{x \to \bar{x}} f(x) \geq f(\bar{x})$.

Definition 1.5 of Rockafellar et al. (1998).

Furthermore, while $\bar{f}$ is smooth over C, the function f, defined on all of $\mathbb{R}^d$, is not even continuous. Hence, the previously stated optimality conditions cannot be employed anymore.

In the following, given a function $f : \mathbb{R}^d \to \overline{\mathbb{R}}$, a sequence $(x^{(t)})_{t \in \mathbb{N}_0}$ and a point $\bar{x}$, *f-attentive convergence* of $x^{(t)}$ to $\bar{x}$, that is, the convergence $x^{(t)} \to \bar{x}$ with $f(x^{(t)}) \to f(\bar{x})$, is denoted by $x^{(t)} \xrightarrow{f} \bar{x}$.

To motivate the upcoming definition, consider the function $f(x) = |x|$. Since f is linear on each of the intervals $(-\infty, 0)$ and $(0, +\infty)$, the differential should evaluate as follows:

$$\mathrm{d}f(x) = \begin{cases} +1, & x > 0\,, \\ -1, & x < 0\,. \end{cases}$$

However, for $x = 0$ many options are possible: Every $v \in [-1, 1]$ satisfies

$$\langle v, x \rangle \leq |\langle v, x \rangle| = |v||x| \leq |x| = f(x)\,,$$

such that $f(x) \geq f(0) + \langle v, x - 0 \rangle$ holds for all $x \in \mathbb{R}$. Especially, for $\bar{x} = 0$, this can be written in the more general local version:

$$f(x) \geq f(\bar{x}) + \langle v, x - \bar{x} \rangle + o(|x - \bar{x}|)$$

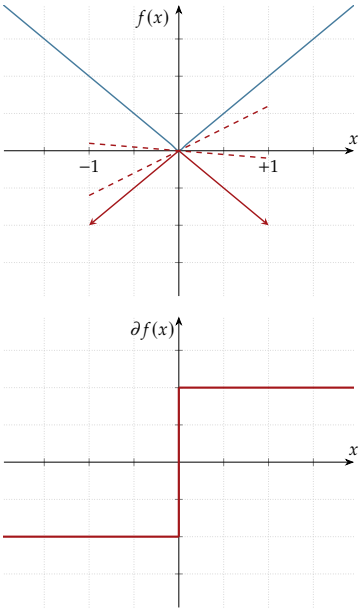

Figure 2.21: The function $f(x) = |x|$ and the corresponding subdifferential.

which yields the following definition:

Definition 8.3 of Rockafellar et al. (1998)

Definition 2.3.7 *Let $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ and $\bar{x} \in \operatorname{dom} f$.*

(i) The regular subdifferential *of f at $\bar{x}$ is defined as:*

$$\hat{\partial} f(\bar{x}) := \{v \in \mathbb{R}^d \ : \ f(x) \geq f(\bar{x}) + \langle v, x - \bar{x} \rangle + o(\|x - \bar{x}\|)\} \,,$$

and $v \in \hat{\partial} f(\bar{x})$ is called a regular subgradient *of f at $\bar{x}$.*

(ii) The (limiting) subdifferential *of f at $\bar{x}$ is defined as:*

$$\partial f(\bar{x}) := \{v \in \mathbb{R}^d \ : \exists x^{(t)} \xrightarrow{f} \bar{x},\, v^{(t)} \to v \text{ with } v^{(t)} \in \hat{\partial} f(x^{(t)})\} \,,$$

and $v \in \partial f(\bar{x})$ is called a (limiting) subgradient *of f at $\bar{x}$.*

Here, we have the following relations between the regular and the limiting subdifferential and how they correspond to the gradient of a differentiable function:

Theorem 8.6 and Exercise 8.8 of Rockafellar et al. (1998).

Lemma 2.3.3 *Let $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ and $\bar{x} \in \operatorname{dom} f$. Then we have:*

(i) $\hat{\partial} f(\bar{x})$ is convex, both $\hat{\partial} f(\bar{x})$ and $\partial f(\bar{x})$ are closed, and we have the inclusion $\hat{\partial} f(\bar{x}) \subset \partial f(\bar{x})$.

(ii) If f is differentiable at $\bar{x}$, we have $\hat{\partial} f(\bar{x}) = \{\nabla f(\bar{x})\}$. Similarly, if f is continuously differentiable on a neighborhood of $\bar{x}$, we even have $\partial f(\bar{x}) = \{\nabla f(\bar{x})\}$.

Hence, if $\bar{x}$ is a local minimizer of f, we have $f(x) \geq f(\bar{x})$ for all x in a neighborhood of $\bar{x}$, which implies the inequality

$$f(x) \geq f(\bar{x}) + \langle 0, x - \bar{x} \rangle + o(\|x - \bar{x}\|) \,,$$

such that $0 \in \hat{\partial} f(\bar{x}) \subset \partial f(\bar{x})$. This necessary optimality condition is known as *Fermat's rule*:

Theorem 10.1 of Rockafellar et al. (1998).

Theorem 2.3.4 *If a proper function $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ has a local minimum at $\bar{x}$, then we have $0 \in \partial f(\bar{x})$.*

Correspondingly, in this context a point $\bar{x} \in \mathbb{R}^d$ will be called *stationary*, if $0 \in \partial f(\bar{x})$.

By definition, the subdifferential of f at x is a set. Hence, ∂f can be considered as a *set-valued mapping*: A set-valued mapping $S : \mathbb{R}^d \rightrightarrows \mathbb{R}^p$ associates to each point $x \in \mathbb{R}^d$ a subset of $\mathbb{R}^p$. Its *graph* is given by $\operatorname{gph} S = \{(x, v) \in \mathbb{R}^d \times \mathbb{R}^p \ : \ v \in S(x)\}$, while its *(effective) domain* is given by $\operatorname{dom} S = \{x \in \mathbb{R}^d \ : \ S(x) \neq \emptyset\}$. The *outer limit* of S at $\bar{x}$ is defined as

$$\limsup_{x \to \bar{x}} S(x) := \{v \in \mathbb{R}^p \ : \ \exists x^{(t)} \to \bar{x},\, v^{(t)} \to v \text{ with } v^{(t)} \in S(x^{(t)})\} \,.$$

Based on this, we get the following notion of continuity for set-valued mappings:

Definition 5.4 of Rockafellar et al. (1998).

Definition 2.3.8 *A set-valued mapping $S : \mathbb{R}^d \rightrightarrows \mathbb{R}^p$ is said to be* outer semi-continuous *at $\bar{x}$, if $\limsup_{x \to \bar{x}} S(x) \subset S(\bar{x})$.*

Finally, to work with such set-valued mappings also from the perspective of measure theory later on, we need a notion of measurability: A set-valued map $S : T \rightrightarrows \mathbb{R}^d$ is said to be measurable, if for every open set $\mathsf{O} \subset \mathbb{R}^d$ the set $S^{-1}(\mathsf{O}) \subset T$ is measurable. In particular, dom S has to be measurable. Here, the following criterion gives a useful condition for measurability of set-valued mappings.

Lemma 2.3.5 *Suppose T is a Borel subset of $\mathbb{R}^p$, and let $S : T \rightrightarrows \mathbb{R}^d$ be closed-valued. If S is outer semi-continuous relative to T, then S is a measurable mapping w.r.t. $\mathfrak{B}(T)$.*

Exercise 14.9 of Rockafellar et al. (1998).

In this context, the last question that needs to be clarified is under which conditions it can be guaranteed that the selection of individual elements from such sets can be done in a measurable way, too. The following result provides an answer:

Lemma 2.3.6 *A closed-valued, measurable mapping $S : T \rightrightarrows \mathbb{R}^d$ always admits a measurable selection: There exists a measurable function $v : \operatorname{dom} S \to \mathbb{R}^d$, such that $v(t) \in S(t)$ for all $t \in T$.*

Corollary 14.6 of Rockafellar et al. (1998).

2.3.3 Convergence of the Trajectory

As shown above, proving convergence to local or global minimizer is often infeasible. However, even proving convergence of the trajectory *at all* can be a delicate problem, especially for non-smooth functions. In general, as shown by Absil et al. (2005), convergence of the trajectory might fail even for simple algorithms like gradient descent on highly smooth functions. However, as pointed out by the same authors, convergence of the sequence can in fact be deduced, if the objective function satisfies the *Łojasiewicz inequality*, which holds for real analytic functions (Bierstone et al., 1988):

Lemma 2.3.7 *Let $f : \mathbb{R}^d \to \mathbb{R}$ be a real analytic function on a neighborhood of $\bar{x} \in \mathbb{R}^d$. Then there are constants $\varepsilon > 0$, $\sigma > 0$ and $\rho \in [0, 1)$, such that*

$$\|\nabla f(x)\| \geq \sigma |f(x) - f(\bar{x})|^\rho \quad \forall x \in \mathsf{B}_\varepsilon(\bar{x}) .$$

An intuitive interpretation of this inequality is given in terms of a *desingularization function*, that is, a function that reparametrizes f in such a way that it becomes sharp around $\bar{x}$, in the sense that the gradient does not become arbitrarily small (Attouch et al., 2013): For simplicity, assume that $f \geq 0$ and $f(\bar{x}) = 0$. Then, on the set $\{f > 0\} \cap \mathsf{B}_\varepsilon(\bar{x})$ the Łojasiewicz gradient inequality can be rewritten as:

$$\begin{aligned} \|\nabla f(x)\| \geq \sigma f(x)^\rho &\iff \|\frac{1}{\sigma} f(x)^{-\rho} \nabla f(x)\| \geq 1 \\ &\iff \|\nabla(\varphi \circ f)(x)\| \geq 1 \quad \text{with} \quad \varphi(x) = \frac{1}{\sigma} \frac{x^{1-\rho}}{1-\rho} . \end{aligned}$$

Unfortunately, the original Łojasiewicz inequality is not applicable in the non-smooth setting. Nevertheless, many pathological failure cases are also excluded by the large class of *semi-algebraic* functions or, more generally, functions that are *definable in an o-minimal structure* or that are *tame*[8] (see, for example, Dries, 1998; Attouch et al., 2010), and extensions

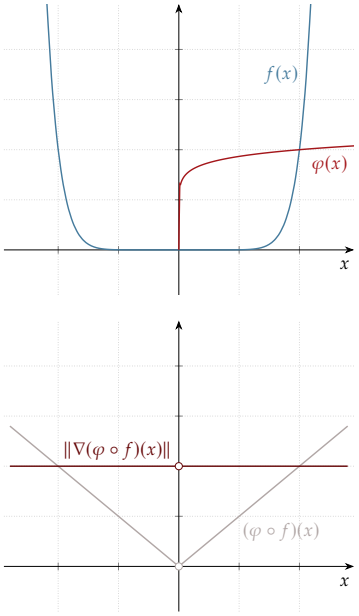

Figure 2.22: Reparametrization of $f(x) = x^{10}$ with the desingularization function $\varphi(x) = x^{\frac{1}{10}}$, that is, $\sigma = 10$, $\rho = 0.9$. The figure is adapted from P. Ochs script for the lecture "Non-smooth Analysis and Optimization", winter term 2022/23.

8: Tame is again *slightly* more general than definable.

of the Łojasiewicz inequality to the case of smooth definable functions are provided by Kurdyka (1998), whereas extensions to the nonsmooth subanalytic or definable setting are shown by Bolte et al. (2007b), Bolte et al. (2007a), and Attouch et al. (2009). This yields the *Kurdyka–Łojasiewicz inequality*:

Definition 2.4 of Attouch et al. (2013).

Definition 2.3.9 *Let $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ be a proper lower semi-continuous function. f is said to have the* Kurdyka-Łojasiewicz property *at $\bar{x} \in \operatorname{dom} f$, if there exists $\eta \in (0, +\infty]$, a neighborhood U of $\bar{x}$, and a continuous concave function $\varphi : [0, \eta) \to [0, \infty)$, such that:*

(i) $\varphi(0) = 0$
(ii) *φ is continuously differentiable on $(0, \eta)$,*
(iii) *for all $s \in (0, \eta)$ we have $\varphi'(s) > 0$,*
(iv) *for all $x \in \mathsf{U} \cap \{f(\bar{x}) < f < f(\bar{x}) + \eta\}$, the* Kurdyka-Łojasiewicz inequality *holds:*

$$\varphi'\big(f(x) - f(\bar{x})\big)\,\operatorname{dist}\big(0, \partial f(x)\big) \geq 1\,.$$

Further, if f satisfies the Kurdyka-Łojasiewicz inequality for every $x \in \operatorname{dom} f$, it is called a Kurdyka-Łojasiewicz function.

As stated above, typical examples for Kurdyka-Łojasiewicz functions are semi-algebraic functions or, more generally, functions that are definable in an o-minimal structure (Attouch et al., 2010). Importantly, while these definitions are highly abstract and sometimes not fully intuitive, most functions that are encountered in optimization problems in practice are definable, such that they satisfy the Kurdyka–Łojasiewicz inequality automatically. Using this, several algorithms have been shown to converge even for nonconvex functions (Attouch et al., 2009; Attouch et al., 2010; Attouch et al., 2013; Bolte et al., 2014; Ochs et al., 2014; Ochs, 2019).

In the remainder of this manuscript, we will learn optimization algorithms. For this, we will have to specify the architecture of the update-step, which can be a highly non-trivial task. Thus, to find a suitable architecture for each individual setting, we also have to be able to test several ones, such that our mathematical model for the algorithm should not depend too much on a specific structure. Therefore, instead of analyzing each single architecture separately, we will consider an abstract algorithm and its trajectory. Then, however, the update step cannot be analyzed in as much detail as could be done, for example, in the case of gradient descent. Hence, we also have to consider abstract properties of the trajectory, which allow, for example, to deduce convergence. Here, the following properties will prove to be useful:

This definition can be found on page 8 of the paper due to Attouch et al. (2013).

Definition 2.3.10 *Let $f : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ be a proper lower semi-continuous function, and let $(x^{(t)})_{t \in \mathbb{N}_0} \subset \mathbb{R}^d$ be a sequence.*

(i) *The sequence $(x^{(t)})_{t \in \mathbb{N}_0}$ satisfies the* sufficient-descent condition *for f, if there exists a constant $a > 0$, such that*

$$f(x^{(t+1)}) + a\|x^{(t+1)} - x^{(t)}\|^2 \leq f(x^{(t)}), \quad \forall t \in \mathbb{N}_0\,.$$

(ii) *The sequence $(x^{(t)})_{t \in \mathbb{N}_0}$ satisfies the* relative-error condition *for f, if there exist a constant $b > 0$ and subgradients $v^{(t)} \in \partial f(x^{(t)})$, $t \in \mathbb{N}$,*

such that

$$\|v^{(t+1)}\| \leq b\|x^{(t+1)} - x^{(t)}\|\,, \quad \forall t \in \mathbb{N}_0\,.$$

(iii) The sequence $(x^{(t)})_{t\in\mathbb{N}_0}$ *satisfies the* continuity condition *for* f*, if for any convergent subsequence* $x^{(t_j)} \overset{j\to\infty}{\to} \bar{x}$ *we have* $f(x^{(t_j)}) \overset{j\to\infty}{\to} f(\bar{x})$.

The importance of these properties is clarified by the following theorem due to Attouch et al. (2013):

Theorem 2.3.8 *Let* $f : \mathbb{R}^d \to \mathbb{R}\cup\{+\infty\}$ *be a proper lower semi-continuous function, and let* $(x^{(t)})_{t\in\mathbb{N}_0} \subset \mathbb{R}^d$ *be a sequence that satisfies the sufficient-descent condition, the relative-error condition, and the continuity condition. If, additionally,* $(x^{(t)})_{t\in\mathbb{N}_0}$ *stays bounded and* f *is a Kurdyka-Łojasiewicz function, then the sequence* $(x^{(t)})_{t\in\mathbb{N}_0}$ *converges to a stationary point of* f.

Actually, Theorem 2.9 of Attouch et al. (2013) is stated slightly differently: They assume existence of a convergent subsequence (instead of boundedness) with corresponding f-attentive convergence towards a cluster point around which f has the KL-property. However, the boundedness assumption together with the generally more restrictive setting here implies existence of such a subsequence.

Clearly, if $(x^{(t)})_{t\in\mathbb{N}_0}$ does not possess the sufficient-descent condition, it is trivial to construct counterexamples for Theorem 2.3.8. As the next two examples show, we also cannot do without the relative-error condition or boundedness:

Example 2.3.1 Starting from $x^{(1)} := 1$, define the sequence for $t \geq 2$ by $x^{(t)} := x^{(t-1)} + \frac{1}{t}$, and consider the positive and convex function $f(x) := \exp(-x)$. We show that the sequence $(x^{(t)})_{t\in\mathbb{N}}$ does satisfy the sufficient-descent condition for f: By definition, we have the recursive formula $f(x^{(t+1)}) = f(x^{(t)})\exp\left(-\frac{1}{t+1}\right)$, which allows for rewriting the sufficient-descent condition as:

$$\frac{a}{(t+1)^2} \leq \left(1 - \exp\left(-\frac{1}{t+1}\right)\right) f(x^{(t)})\,.$$

Then, we have to find $a > 0$ satisfying this inequality for all $t \in \mathbb{N}$. For $t = 1$, the right-hand side is greater than $\frac{1}{9}$, such that we can choose any $a \in (0, \frac{4}{9}]$ (rough estimate). Thus, take $a \in (0, \frac{4}{9}]$, such that $\tilde{a} := a \cdot 2e \in (0, \frac{4}{9}]$, and proceed by induction (note that $\tilde{a}$ satisfies the stated condition for $t = 1$). Assuming that the inequality holds true for all $k \leq t$, we get by the induction hypothesis:

$$\frac{\tilde{a}}{(t+2)^2} \leq \left(\frac{t+1}{t+2}\right)^2 \left(1 - \exp\left(-\frac{1}{t+1}\right)\right) f(x^{(t)})$$

By inserting a trivial 1 three-times, the right-hand side can be written as:

$$\left(\frac{t+1}{t+2}\right)^2 \cdot \exp\left(\frac{1}{t+2}\right) \cdot \frac{\exp\left(\frac{1}{t+1}\right) - 1}{\exp\left(\frac{1}{t+2}\right) - 1} \cdot \left(1 - \exp\left(-\frac{1}{t+2}\right)\right) f(x^{(t+1)})\,.$$

Here, the first term is bounded by 1, the second by e, and the third by 2. Hence, dividing both sides by $2e$, we get:

$$\frac{a}{(t+2)^2} \leq \left(1 - \exp\left(-\frac{1}{t+2}\right)\right) f(x^{(t+1)})\,,$$

such that $(x^{(t)})_{t\in\mathbb{N}}$ satisfies the sufficient-descent condition for f. Nevertheless, we have $x^{(t)} = x^{(t-1)} + \frac{1}{t} = \ldots = \sum_{k=1}^{t} \frac{1}{k}$, such that $|x^{(t)}| \overset{t\to\infty}{\to} \infty$, that is, the sequence is unbounded and does not converge.

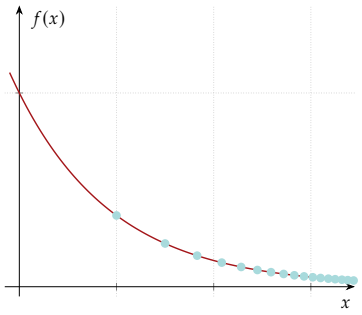

Figure 2.23: Counterexample for the case that the sequence is unbounded (and, actually, there is no minimizer).

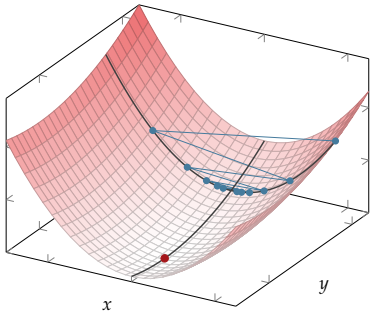

Figure 2.24: Idea of the counterexample for the case that the relative-error condition fails.

Example 2.3.2 Consider the smooth and strongly convex function $f(x,y) := \frac{1}{2}x^2 + \frac{1}{2}y^2$, and define the sequence $((x^{(t)}, y^{(t)}))_{t\in\mathbb{N}_0} \subset \mathbb{R}^2$ through

$$(x^{(t+1)}, y^{(t+1)}) := (x^{(t)} - 0.1x^{(t)}, y^{(t)})\,.$$

Then we have $\|(x^{(t+1)}, y^{(t+1)}) - (x^{(t)}, y^{(t)})\|^2 = \left(0.1x^{(t)}\right)^2$, and it holds:

$$\begin{aligned} f(x^{(t+1)}, y^{(t+1)}) &= \frac{1}{2}\left(x^{(t)} - 0.1x^{(t)}\right)^2 + \frac{1}{2}\left(y^{(t)}\right)^2 \\ &= f(x^{(t)}, y^{(t)}) - 0.1\left(x^{(t)}\right)^2 + \frac{1}{2}\left(0.1x^{(t)}\right)^2 \\ &= f(x^{(t)}, y^{(t)}) - 9.5\|(x^{(t+1)}, y^{(t+1)}) - (x^{(t)}, y^{(t)})\|^2\,, \end{aligned}$$

such that $(x^{(t+1)}, y^{(t+1)})$ satisfies the sufficient-descent condition. However, for $y^{(0)} \neq 0$, the sequence converges to $(0, y^{(0)})$, which is not a stationary point of f.

2.4 Background on Learning-to-Optimize

Learning-to-optimize leverages machine learning techniques to develop new and faster optimization algorithms. The main underlying assumption is that we are given a family of optimization problems with similar structure, and we repeatedly have to solve problems of this kind. Therefore, we want to have algorithms that are specifically tailored towards this family of problems and that can solve them particularly fast. One assumption that generally goes along with this is that we train the algorithm in a preliminary phase that is allowed to be time-consuming, as long as the resulting method is saving time during application. Given this generic formulation of ideas and wishes, several question arise:

(i) How do we actually train an optimization algorithm?
(ii) How do such learned optimization algorithms look like?
(ii) Can we still provide guarantees for their performance?

How these questions have been answered so far is outlined in the following subsections.

2.4.1 General Framework and Broader Context

There are several areas of research that have significant overlap with learning-to-optimize. First, learning-to-optimize is basically the same as *amortized optimization*, which, by using machine learning methods, tries to predict the solution to parametric optimization problems, either directly ("fully-amortized") or iteratively ("semi-amortized") (Amos, 2023). Second, if the optimization problem is that of determining the parameters of a machine learning model, learning-to-optimize overlaps with the area of *meta learning* (or *learning-to-learn*), which generally deals with the problem of enabling faster learning and adaption of machine learning models, especially in deep learning (Vilalta et al., 2002; Huisman et al., 2021; Hospedales et al., 2022; Vettoruzzo et al., 2024). Finally, the field of *automated machine learning*, or AutoML, more generally refers to automating as many steps as possible that are necessary to develop a machine learning application and thus also involves the design and

choice of a corresponding optimization algorithm (Yao et al., 2018; Hutter et al., 2019).

Since to date there is no well-established framework for how to classify different approaches in learning-to-optimize, we adopt the distinction into *model-free* and *model-based* approaches proposed by T. Chen et al. (2022). Therefore, also the upcoming discussion of several approaches in learning-to-optimize is based on their work.

2.4.2 Model-Free Approaches

The abstract work-flow of model-free approaches is similar to traditional optimization: a parametric update is applied iteratively to the internal state of the algorithm. However, the name "model-free" is due to the fact that, in this case, the update should explicitly not be inspired by a traditional optimization algorithm. Rather, it is more like a black-box that observes some inputs and predicts the new iterate.

Recurrent Neural Networks

Due to the iterative nature of optimization algorithms, many model-free approaches rely on *recurrent neural networks* (RNNs), especially in form of the *long short-term memory* (LSTM) architecture (Hochreiter et al., 2001; Andrychowicz et al., 2016; Wichrowska et al., 2017; Y. Chen et al., 2017; Lv et al., 2017; Cao et al., 2019). While the idea of using recurrent neural networks to model optimization algorithms comes naturally, the resulting methods suffer from the typical problems of RNNs, especially vanishing/exploding gradients and memory overheads (Bengio et al., 1994; Pascanu et al., 2013; Orvieto et al., 2023). Thus, these models cannot be trained for an arbitrary number of iterations, which forces to subdivide the entire trajectory into smaller pieces. This, however, often leads to unsatisfactory results, especially in terms of generalization to more iterations, and several approaches to overcome these difficulties have been proposed: Wichrowska et al. (2017) introduce a hierarchical RNN architecture and additionally draw the number of unrollings and the unrolling length from a heavy-tailed exponential distribution. While achieving the needed generalization, this approach does not achieve the same wall-clock time as simple optimization algorithms. Thus, Metz et al. (2019) replace the recurrent neural network with a multilayer perceptron. Doing so, they manage to train algorithms that are faster in wall-clock time than standard ones like Adam (see also Metz et al., 2022). Finally, T. Chen et al. (2020) consider training techniques in general and introduce a progressive scheme that gradually increases the unrolling length, as well as an imitation learning approach to learn to mimic analytic optimizers.

Reinforcement Learning

From an abstract point of view, another natural way to model optimization algorithms is through the lens of reinforcement learning, that is, the optimization algorithm is considered to be an agent that acts in the optimization space in search for the minimizer. This idea was first proposed by Li et al. (2017a), and further specialized to the training of neural networks (Li et al., 2017b). In their framework, an optimization algorithm is equivalent to a so-called *policy*, such that finding a suitable optimization algorithm becomes a *policy search problem*. Besides this work,

other works that consider using reinforcement learning in learning-to-optimize rather use it as a way to have adaptive learning-rate schedulers for existing algorithms (Daniel et al., 2016; C. Xu et al., 2017; Z. Xu et al., 2019; Almeida et al., 2021).

Program Search

Based on the common problems of learned optimizers, especially generalization to more iterations and/or other objective functions, Bello et al. (2017) propose to predict the mathematical update of an algorithm in a given programming language instead of learning a parameterized update. Similarly, Zheng et al. (2022) introduce *symbolic learning-to-optimize*, which leverages symbolic regression to get scalable and interpretable updates, and X. Chen et al. (2023) formulate the discovery of new optimization algorithms as a program search problem.

2.4.3 Model-Based Approaches

As counterpart to model-free approaches, model-based approaches try to combine traditional optimization algorithms with machine learning techniques, that is, the algorithmic update is inspired by hand-crafted algorithms and usually only some part of it is learned. Here, the majority of approaches can be characterized into the following two branches, namely *plug-and-play* and *unrolling*.

Plug-and-Play

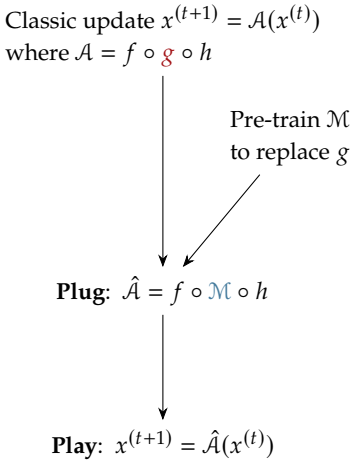

Figure 2.25: High-level idea of plug-and-play methods: A model $\mathcal{M}$ gets pre-trained to replace a certain part of the classic update. Then, we just insert this model into the update (plug) and iterate the algorithm (play).

The workflow of plug-and-play methods is depicted in Figure 2.25: Given the update of a traditional optimization algorithm, a model is trained to replace one of its operations. Then, this pre-trained model is inserted into the algorithm, that is, the original operation is replaced by the learned one. Subsequently, instead of using the original update, the new one is applied iteratively.
Originally, this idea was introduced for inverse problems in image reconstruction by Venkatakrishnan et al. (2013), where the authors replace one of the proximal maps of the *alternating direction method of multipliers* algorithm (ADMM; see for example Boyd et al., 2011) by a learned map. Based on the given promising empirical performance, subsequent works have also considered its theoretical properties: Sreehari et al. (2016) provide sufficient conditions (especially non-expansiveness), such that the learned map is a proximal map, which allows to establish global convergence. Similarly, Chan et al. (2017) consider a slightly weaker assumption (certain kind of boundedness instead of non-expansiveness), which is at least sufficient to prove the fixed-point convergence of the method.
The overall idea has been adopted several times for different applications and/or algorithms, mostly in the image-reconstruction literature (for example, see He et al., 2019; Gupta et al., 2018; Brifman et al., 2016; Ono, 2017; Kamilov et al., 2017; Meinhardt et al., 2017; Chang et al., 2017; Ahmad et al., 2020; Gärtner et al., 2023), and, while generally most of the works in learning-to-optimize are empirical, also some theoretical results have been provided (Teodoro et al., 2017; Buzzard et al., 2018; Tirer et al., 2019; Sun et al., 2019; Ryu et al., 2019; Gavaskar et al., 2020; Terris et al., 2021; Sun et al., 2021; Cohen et al., 2021). In these works, the underlying proof-idea is to trace the learned object back to, or constrain

it to, a mathematical object with existing convergence guarantees. As a consequence, the convergence results themselves are often not that surprising, rather it is the way in which the conditions are enforced in each situation. This highlights one major drawback of plug-and-play methods: They are strongly tailored to their particular setting, that is, they often exploit very specific properties of the algorithm and the problem at hand, such that the results are not easily transferable to other problems. On the other hand, one of the main advantages is the fact that these methods can indeed provide convergence guarantees and that they can be applied for an arbitrary number of iterations. This contrasts with algorithm unrolling, which is discussed next.

Unrolling

Abstractly, neural networks can be considered as a concatenation of finitely many parameterized functions, where each function comprises a layer of the neural network. Similarly, in optimization the T-th iterate can be computed by applying the update step of the algorithm T-times iteratively. Given this, the idea of unrolling is straightforward (see Figure 2.26): Each update-step of the algorithm is treated as a layer in a neural network, which then is trained in an end-to-end-fashion (Monga et al., 2021). This idea was first proposed by Gregor et al. (2010) in the signal-processing literature for the *iterative shrinkage and thresholding algorithm* (ISTA; for example, see Daubechies et al., 2004) and, based on its empirical success, has been widely adopted in the sparse-recovery literature (Sprechmann et al., 2015; Sprechmann et al., 2013; Ablin et al., 2019; Sulam et al., 2020). Furthermore, it has also been studied extensively from a theoretical perspective (Moreau et al., 2017; Giryes et al., 2018; X. Chen et al., 2018; J. Liu et al., 2019). However, due to the sparsity constraints, most of these results focus on either ISTA or its accelerated variant, namely the *fast iterative shrinkage and thresholding algorithm* (FISTA; see Beck et al., 2009). Nevertheless, by now, many other optimization algorithms have been unrolled, too: Domke (2012) considers the effect of truncating optimization algorithms on implicit differentiation for several standard algorithms like gradient-descent, heavy-ball, etc. Xin et al. (2016), as well as Wang et al. (2016), treat the ℓ_0 sparse approximation problem with the *iterative hard thresholding* algorithm, and Kobler et al. (2017) propose their so-called "variational networks" for image reconstruction based on unrolling *proximal gradient descent*. Similarly, Adler et al. (2018) unroll a *proximal primal-dual algorithm* for tomographic reconstruction, and Wadayama et al. (2019) as well as Wu et al. (2019) consider *projected gradient descent*. Other algorithms that have been considered are ADMM (Yang et al., 2016; Xie et al., 2019), a *proximal interior point method* (Corbineau et al., 2019), and the *Frank-Wolfe algorithm* (D. Liu et al., 2020).
While there are theoretical results about convergence of unrolled methods, these have to be interpreted with some caution: In practice, an algorithm can be unrolled only for a finite number of iterations and the resulting network gets trained for exactly this number of iterations. Thus, in almost all cases convergence will not be observed[9] and the network might actually overfit to the specific setting. This motivates an alternative point of view, namely to consider the generalization performance of such an algorithm. Before the works that this manuscript is based on, only few works have addressed this: Either directly by means of Rademacher complexity (X. Chen et al., 2020), or indirectly in form of a stability analysis (Kobler et al., 2022), as algorithmic stability is linked to generalization (Bousquet et al., 2000; Bousquet et al., 2002; Shalev-Shwartz et al., 2010).

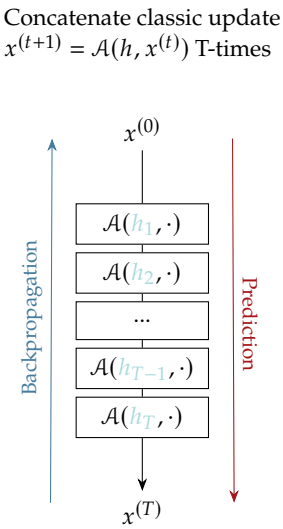

Figure 2.26: High-level idea of unrolling methods: The (parameterized) update $\mathcal{A}(h, \cdot)$ gets treated as a layer in a neural network, which in turn is amenable to end-to-end training.

9: This problem will be a recurring theme in later parts of this manuscript.

2.4.4 Convergence Guarantees

As stressed by T. Chen et al. (2022), learned optimization methods may lack theoretical guarantees for the sake of convergence speed. Yet, there are applications where a convergence guarantee is of highest priority: Möller et al. (2019) provide an example of how a purely learning-based approach for medical image reconstruction produces the visually more appealing images, *while failing to recreate the crucial details of malign tissue*. To avoid such a failure, they restrict the output of their method to so-called descent directions (for which mathematical guarantees exist), which in turn allows them to prove convergence. Similarly, Banert et al. (2020) and Banert et al. (2024) consider parameterized algorithms and use learning to choose the corresponding parameters, although they restrict the set of possible coefficients to those that guarantee convergence. Closely related safe-guarding techniques for learning-to-optimize were presented by R. Liu et al. (2020), Prémont-Schwarz et al. (2022), and Heaton et al. (2023), and analogous ideas also appeared in the context of *operator splitting techniques* (Themelis et al., 2019) or *Anderson acceleration* (Zhang et al., 2020). The main difference is that, instead of restricting the method a priori, in each iteration the predicted update gets checked for a convergence-ensuring condition, and the predicted update will only be accepted, if this condition is met. Otherwise, it is replaced by the output of a traditional optimization algorithm.
While using such restrictions allows for analyzing learned optimization algorithms in the same way as traditional ones, this comes at a price: Not only the analysis of the learned algorithm, also its performance is restricted and eventually similar to hand-crafted algorithms. In this case, however, it is hard to justify why we should prefer the learned algorithm over the hand-crafted one, that is, why we should learn optimization algorithms in the first place.

2.4.5 Design of Learned Optimization Algorithms

The design of learned optimization algorithms, especially for model-free approaches based on RNNs, their training, as well as common pitfalls and how to avoid them, has been studied (from an empirical perspective) by Metz et al. (2019), Wichrowska et al. (2017), and Metz et al. (2022). In contrast, motivated by traditional optimization algorithms, J. Liu et al. (2023) advocate for more mathematical structure in the design of learned optimization algorithms, because their analysis reveals that, if we require properties like asymptotic stationarity or global convergence for whole problem classes, the possible update-steps are more or less limited to versions of preconditioned gradient descent. Similarly, Castera et al. (2024) propose several design principles for learning-to-optimize that traditional optimization algorithms usually obey and analyze how they affect generalization. Then, based on their findings, they propose a novel learned BFGS-algorithm. Similarly, Liao et al. (2023) proposed to use learning-to-optimize in the context of quasi-Newton methods.

Part II

Learning-to-Optimize with Generalization Guarantees

PAC-Bayesian Learning of Optimization Algorithms

3

The underlying idea of this initial work is the following: In practice, an optimization algorithm has to be stopped at some point in time. Thus, only a finite number of iterations can be performed anyway. Therefore, by concatenating the algorithmic update sufficiently often with itself, that is, by unrolling it, we can simplify the setting tremendously and treat the whole optimization process as a fixed map in the optimization space. This allows for considering it from the perspective of supervised or self-supervised learning, depending on which performance measure is considered: If the algorithm is trained to approximate the solution of an optimization problem, supervised methods seem appropriate. If the algorithm is trained to yield a small function-value, the supervisory signal (the loss) can be computed from the input alone, that is, self-supervised methods are applicable. However, one question remains:

What is the actual input data on which the algorithm is trained?

Here, clearly, optimization algorithms should minimize *functions*. Thus, the underlying idea is that there should be a "distribution of functions with similar structure". We formalize this in the following way.

3.1 Problem Setup

We assume that we are given a distribution over loss-functions, which is modeled by a *parametric function* ℓ together with a random variable P, representing the *parameter*:

Assumption 3.1.1 *We are given a Polish space $\mathcal{P}$ and a non-negative and measurable loss-function $\ell : \mathbb{R}^n \times \mathcal{P} \to [0, +\infty]$. Further, for some $N \in \mathbb{N}$, we are given i.i.d. random variables $P, P_1, ..., P_N : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{P}$.*

Then, ideally, we would like to find a solution to each realization of the random objective:

$$\text{Find } x^* : \mathcal{P} \to \mathbb{R}^n, \text{ s.t. } x^*(p) \in \arg\min_{x \in \mathbb{R}^n} \ell(x, p) \quad \mathbb{P}_P - a.s. \tag{3.1}$$

However, we will only solve a relaxed version of (3.1) and provide *generalization bounds* for the *average performance* after training on a data set.

Definition 3.1.1 *The measurable function $S : (\Omega, \mathfrak{A}, \mathbb{P}) \longrightarrow \mathcal{P}^N, \quad \omega \mapsto (P_1, ..., P_N)(\omega)$ is called a* data set *and, if the random variables $P_1, ..., P_N$ are i.i.d., that is, $\mathbb{P}_S = \bigotimes_{i=1}^N \mathbb{P}_{P_i} = \mathbb{P}_P^{\otimes N}$, it is called an* i.i.d. data set. *Further, $\mathcal{P}^N$ is called the* data-space.

PAC-Bayesian generalization bounds involve a so-called posterior distribution, which usually is a "data-dependent distribution". As also pointed out by Rivasplata et al. (2020), this is an instance of a probability kernel:

Definition 3.1.2 *Let S be a data set with data-space $\mathcal{P}^N$, and let $\mathcal{U}$ be a measurable space. A probability kernel from $\mathcal{P}^N$ to $\mathcal{U}$ is called a* data-dependent distribution *on $\mathcal{U}$.*

For solving problem (3.1), for every realization p of P, we apply an optimization algorithm $\mathcal{A}$ to $\ell(\cdot, p)$. For this, we consider a similar setting as London (2017), that is, randomized algorithms are considered as deterministic algorithms with randomized hyperparameters:

Definition 3.1.3 *Let $\mathcal{H}$ be a Polish space and $n \in \mathbb{N}$. A measurable function*

$$\mathcal{A} : \mathcal{H} \times \mathbb{R}^n \times \mathcal{P} \longrightarrow \mathbb{R}^n, \quad (h, x^{(0)}, p) \mapsto \mathcal{A}(h, x^{(0)}, p)\,,$$

is called a parametric algorithm. *$\mathbb{R}^n$ is the space of the optimization variable, $\mathcal{P}$ the space of the parameters of the loss function, and $\mathcal{H}$ the space of the hyperparameters of the algorithm.*

Please note that $\mathcal{A}$ corresponds to the *whole* algorithm, that is, for an iterative algorithm its output is the final iterate.
In the PAC-Bayesian approach, learning $\mathcal{A}$ refers to finding a distribution $\mathbb{Q}$ on $\mathcal{H}$ based on its performance on a data set S. For this, one needs a reference distribution, called the *prior*, which can (and should) encode prior knowledge about suitable choices of hyperparameters:

Alternatively, one could also consider the product-measure $\mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N}$ on $\mathcal{H} \times \mathcal{P}^N$ and denote the corresponding projection by H and $P_1, ..., P_N$. While this has not been checked in detail, probably P itself would not be needed in this case.

Assumption 3.1.2 *We are given a parametric algorithm $\mathcal{A}$ with Polish hyperparameter space $\mathcal{H}$, and a (prior) distribution $\mathbb{P}_H$ on $\mathcal{H}$ that is induced by a random variable $H : (\Omega, \mathfrak{A}, \mathbb{P}) \longrightarrow \mathcal{H}$, which is independent of S and P. Further, the initialization $x^{(0)} \in \mathbb{R}^n$ is given and fixed.*

Notation 3.1.1 To simplify the notation, we use the short-hand $\ell(h, p) := \ell(\mathcal{A}(h, p), p)$. Furthermore, if not needed explicitly, $x^{(0)}$ and $(\Omega, \mathfrak{A}, \mathbb{P})$ will not be mentioned in the following.

One could also define the risk as $h \mapsto \mathbb{E}\{\ell(H, P) \mid H = h\}$. However, since P and H are independent, this is $\mathbb{P}_H$-almost surely the same as the definition used here.

Definition 3.1.4 *Suppose P and ℓ satisfy Assumption 3.1.1, and $\mathcal{A}$ satisfies Assumption 3.1.2. The* risk *of $\mathcal{A}$ is defined as the measurable function:*

$$\mathcal{R} : \mathcal{H} \longrightarrow [0, +\infty], \; h \mapsto \mathbb{E}\{\ell(h, P)\} = \mathbb{E}_P\{\ell(h, \cdot)\}\,.$$

Similarly, for an i.i.d. data set $S = (P_1, ..., P_N)$ the empirical risk *is defined as:*

$$\hat{\mathcal{R}} : \mathcal{H} \times \mathcal{P}^N \longrightarrow [0, +\infty], \quad (h, S) \mapsto \hat{\mathcal{R}}(h, S) = \frac{1}{N} \sum_{i=1}^{N} \ell\,(h, P_i)\,.$$

The following theory is based on exponential families, which is a very flexible class of distributions. We highlight the dependency on the data set in the following adjusted definition:

Definition 3.1.5 *Let $\emptyset \neq \Lambda$ be an index set, S a data set with data-space $\mathcal{P}^N$, and let $\mathcal{U}$ be a measurable space. A family of probability kernels $(\mathbb{Q}_\lambda)_{\lambda\in\Lambda}$ from $\mathcal{P}^N$ to $\mathcal{U}$ is called a* data-dependent exponential family *(in η and τ), if there is a probability measure μ on $\mathcal{U}$, functions $\eta : \Lambda \to \mathbb{R}^k$, $a : \Lambda \times \mathcal{P}^N \to (0, +\infty)$, and measurable functions $\tau : \mathcal{U} \times \mathcal{P}^N \to \mathbb{R}^k$, $b : \mathcal{U} \to (0, +\infty)$, such that $\mathbb{Q}_\lambda(s) = ba(\lambda, s) \exp\left(\langle \eta(\lambda), \tau(\cdot, s)\rangle\right) \cdot \mu$ for every $\lambda \in \Lambda$ and $s \in \mathcal{P}^N$, that is,*

$$\mathbb{Q}_\lambda(s)\{\mathsf{B}\} = \int_{\mathsf{B}} b(u)a(\lambda, s) \exp(\langle \eta(\lambda), \tau(u, s)\rangle)\, \mu(du), \quad \mathsf{B} \in \mathfrak{B}(\mathcal{U})\,.$$

We introduce data-dependency through τ, since it strongly affects the shape of the distribution and, contrary to η, is defined on the underlying space $\mathcal{U}$. Since we want to learn a distribution over hyperparameters $h \in \mathcal{H}$, we make the following assumption:

Assumption 3.1.3 *On the hyperparameter space $\mathcal{H}$, we are given a data-dependent exponential family $(\mathbb{Q}_\lambda)_{\lambda\in\Lambda}$ in η and τ with dominating probability measure $\mu = \mathbb{P}_H$, such that the map $h \mapsto b(h) \exp(\langle \eta(\lambda), \tau(h, s)\rangle)$ is non-trivial and integrable w.r.t. $\mathbb{P}_H$ for every $\lambda \in \Lambda$, $s \in \mathcal{P}^N$, that is, $\mathbb{E}_H \left\{b \exp(\langle \eta(\lambda), \tau(\cdot, s)\rangle)\right\} \in (0, \infty)$.*

In the following, the last integral in Assumption 3.1.3 will be of great interest. Here, we will use a similar notation as in the book by Barndorff–Nielsen (2014) and denote

$$\begin{aligned} c(\lambda, s) &:= \mathbb{E}_H \left\{b \exp(\langle \eta(\lambda), \tau(\cdot, s)\rangle)\right\}\,, \\ \kappa(\lambda, s) &:= \log\left(c(\lambda, s)\right) = \log\left(\mathbb{E}_H \left\{b \exp\left(\langle \eta(\lambda), \tau(\cdot, s)\rangle\right)\right\}\right)\,. \end{aligned} \tag{3.2}$$

With this notation, it holds that $a(\lambda, s) = c(\lambda, s)^{-1}$ is the normalization constant of $\mathbb{Q}_\lambda(s)$.

Remark 3.1.1
(i) If η describes a lower-dimensional manifold in $\mathbb{R}^k$, $(\mathbb{Q}_\lambda)_{\lambda\in\Lambda}$ is called a *curved* exponential family (Efron, 1975), whose properties might differ from those of *linear* exponential families, for example, convexity of the map $\lambda \mapsto a(\lambda, s)$.
(ii) In PAC-Bayes, the dominating measure $\mathbb{P}_H$ is usually referred to as *prior* and every distribution $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ is referred to as a *posterior*. This deviates from the standard definitions of prior and posterior in Bayesian statistics.
(iii) In general, the integrability assumption is restrictive, as it affects the choice of b, η and τ. However, in Section 3.3 we will *construct* η and τ such that this holds.
(iv) In the special case $b \equiv 1$ and $\eta(\lambda) \equiv \lambda$, the map $\lambda \mapsto c(\lambda, s)$ is the moment-generating function of the random variable $\tau(H, s)$. Similarly, in this case $\lambda \mapsto \kappa(\lambda, s)$ is the corresponding cumulant-generating function.

Finally, we will restrict Λ to a compact set. This is needed to get a uniform bound in λ (see Langford et al., 2001; Catoni, 2007; Alquier, 2024).

Assumption 3.1.4 *Λ is a compact set with finite covering $\mathcal{O} := \{\mathsf{O}_1, \ldots, \mathsf{O}_\mathcal{K}\}$, that is, $\Lambda \subset \bigcup_{i=1}^{\mathcal{K}} \mathsf{O}_i$, such that there is a constant $\mathcal{C}_\mathcal{O}$, which, for every $s \in \mathcal{P}^N$, allows for the bound $\max_{i=1,\ldots,\mathcal{K}} \sup_{\lambda,\lambda'\in\mathsf{O}_i} \kappa(\lambda, s) - \kappa(\lambda', s) \leq \mathcal{C}_\mathcal{O}$.*

Remark 3.1.2 The non-trivial part of this assumption is the existence of the constant $\mathcal{C}_{\mathcal{O}}$ for the given finite covering. It does hold, for example, if Λ is a finite set ($\mathcal{K} = |\Lambda|$, $\mathcal{C}_{\mathcal{O}} = 0$), or, if (Λ, ρ) is a compact metric space and κ is Lipschitz-continuous in λ (uniformly in s) with Lipschitz constant L, such that $\mathcal{C}_{\mathcal{O}} = L \cdot \max_{i=1,\dots,\mathcal{K}} \operatorname{diam} \mathsf{O}_i$, where the diameter of a set A is given by $\operatorname{diam} \mathsf{A} = \sup_{x,y\in\mathsf{A}} \rho(x,y)$.

3.2 General PAC-Bayesian Theorem

In this section we prove a general PAC-Bayesian bound for data-dependent exponential families, which then can be specialized into a generalization bound for the learned parametric optimization algorithm $\mathcal{A}$. It is based on the following two lemmas: The first lemma is a form of the Donsker-Varadhan variational formulation and yields uniformity in the distributions $\mathbb{Q}$, while the second lemma yields uniformity in $\lambda \in \Lambda$ by controlling $\lambda \mapsto \kappa(\lambda, s)$ for every $s \in \mathcal{P}^N$.

Lemma 3.2.1 *Suppose that Assumption 3.1.3 holds and define κ as in (3.2). Then for every $\lambda \in \Lambda$ and $s \in \mathcal{P}^N$ it holds that*

$$\kappa(\lambda, s) = \sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)} \mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle + \log(b)\} - D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H)\,.$$

Furthermore, for every $\lambda \in \Lambda$, the supremum is attained at $\mathbb{Q}_\lambda(s)$.

Here, we use the operator-notation for a more compact representation.

Proof. Take any $\lambda \in \Lambda$ and $s \in \mathcal{P}^N$. First, let $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ be arbitrary. By the Radon-Nikodym theorem, there exists a measurable function $f \geq 0$, s.t. $\mathbb{Q} = f \cdot \mathbb{P}_H$. Since the convention $0 \cdot \infty = 0$ applies throughout measure theory, one has:

$$\begin{aligned} D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) &= \mathbb{Q}\{\log(f)\} = \mathbb{P}_H\{f \log(f)\} \\ &= \mathbb{P}_H\{\mathbb{1}_{\{f>0\}} f \log(\mathbb{1}_{\{f>0\}} f)\}\,. \end{aligned}$$

Hence, w.l.o.g. we can assume that $f > 0$ $\mathbb{P}_H$-a.s. Then, by Jensen's inequality, one gets:

$$\begin{aligned} &\mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle + \log(b)\} - \mathbb{Q}\{\log(f)\} \\ &= \mathbb{Q}\Big\{\log\Big(\frac{b}{f}\exp\big(\langle \eta(\lambda), \tau(\cdot, s)\rangle\big)\Big)\Big\} \\ &\leq \log\Big(\mathbb{Q}\Big\{\frac{b}{f}\exp\big(\langle \eta(\lambda), \tau(\cdot, s)\rangle\big)\Big\}\Big) \\ &= \log\Big((f\cdot\mathbb{P}_H)\Big\{\frac{b}{f}\exp\big(\langle \eta(\lambda), \tau(\cdot, s)\rangle\big)\Big\}\Big) \\ &= \log\big(\mathbb{P}_H\big\{b\exp\big(\langle \eta(\lambda), \tau(\cdot, s)\rangle\big)\big\}\big) = \kappa(\lambda, s)\,. \end{aligned}$$

Hence, we have $\kappa(\lambda, s) \geq \mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle + \log(b)\} - D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H)$ for any probability measure $\mathbb{Q} \ll \mathbb{P}_H$. Now consider a member of the exponential family. By definition we have:

$$\begin{aligned} &D_{\mathrm{KL}}(\mathbb{Q}_\lambda(s) \parallel \mathbb{P}_H) \\ &= \int_{\mathcal{H}} \log\left(b(h)a(\lambda, s)\exp(\langle \eta(\lambda), \tau(h, s)\rangle)\right)\ \mathbb{Q}_\lambda(s)(dh)\,. \end{aligned}$$

By the properties of the logarithm, this is the same as:

$$
\begin{aligned}
&= \int_{\mathcal{H}} \log(b(h)) + \langle \eta(\lambda), \tau(h,s) \rangle \, \mathbb{Q}_\lambda(s)(dh) - \log\left(c(\lambda, s)\right) \\
&= \int_{\mathcal{H}} \log(b(h)) + \langle \eta(\lambda), \tau(h,s) \rangle \, \mathbb{Q}_\lambda(s)(dh) - \kappa(\lambda, s) \\
&= \mathbb{Q}_\lambda(s)\{\langle \eta(\lambda), \tau(\cdot, s) \rangle + \log(b)\} - \kappa(\lambda, s)\,.
\end{aligned}
$$

Rearranging yields

$$
\kappa(\lambda, s) = \mathbb{Q}_\lambda(s)\{\langle \eta(\lambda), \tau(\cdot, s) \rangle + \log(b)\} - D_{\mathrm{KL}}(\mathbb{Q}_\lambda(s) \,\|\, \mathbb{P}_H)\,.
$$

□

Lemma 3.2.2 *Suppose that Assumption 3.1.4 holds and assume that, for all $r \in \mathbb{R}$ and $\lambda \in \Lambda$, we have $\mathbb{P}\{\kappa(\lambda, S) > r\} \leq \exp(-r)$. Then*

$$
\mathbb{P}\{\sup_{\lambda \in \Lambda} \kappa(\lambda, S) \leq \log(\mathcal{K}/\epsilon) + \mathcal{C}_{\mathcal{O}}\} \geq 1 - \epsilon\,.
$$

Proof. W.l.o.g. assume that $\mathsf{O}_i \neq \emptyset$ and choose $\lambda_i \in \mathsf{O}_i$, $i = 1, \ldots, \mathcal{K}$. Then, for every $s \in \mathcal{P}^N$, it holds that:

$$
\begin{aligned}
\sup_{\lambda \in \Lambda} \kappa(\lambda, s) &\leq \max_{i=1,\ldots,\mathcal{K}} \sup_{\lambda \in \mathsf{O}_i} \kappa(\lambda, s) \\
&= \max_{i=1,\ldots,\mathcal{K}} \Big(\kappa(\lambda_i, s) + \sup_{\lambda \in \mathsf{O}_i} \left(\kappa(\lambda, s) - \kappa(\lambda_i, s)\right) \Big) \\
&\leq \max_{i=1,\ldots,\mathcal{K}} \kappa(\lambda_i, s) + \mathcal{C}_{\mathcal{O}}\,.
\end{aligned}
$$

Thus, in total one gets for $r \in \mathbb{R}$:

$$
\begin{aligned}
\mathbb{P}\Big\{\sup_{\lambda \in \Lambda} \kappa(\lambda, S) > r\Big\} &\leq \mathbb{P}\Big\{\max_{i=1,\ldots,\mathcal{K}} \kappa(\lambda_i, S) + \mathcal{C}_{\mathcal{O}} > r\Big\} \\
&\leq \sum_{i=1}^{\mathcal{K}} \mathbb{P}\left\{\kappa(\lambda_i, S) + \mathcal{C}_{\mathcal{O}} > r\right\} \\
&\leq \sum_{i=1}^{\mathcal{K}} \exp(\mathcal{C}_{\mathcal{O}} - r) = \mathcal{K} \exp(\mathcal{C}_{\mathcal{O}} - r)\,.
\end{aligned}
$$

Taking $r = \log\left(\frac{\mathcal{K}}{\epsilon}\right) + \mathcal{C}_{\mathcal{O}}$ gives

$$
\mathbb{P}\Big\{\sup_{\lambda \in \Lambda} \kappa(\lambda, S) > \log\Big(\frac{\mathcal{K}}{\epsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big\} \leq \epsilon\,.
$$

□

Theorem 3.2.3 *Suppose that Assumptions 3.1.3 and 3.1.4 hold, and assume that $\mathbb{E}_S\{c(\lambda, \cdot)\} \leq 1$ for all $\lambda \in \Lambda$. Then, it holds that:*

$$
\begin{aligned}
\mathbb{P}\Big\{\forall \lambda \in \Lambda,\ \forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)\ &: \mathbb{Q}\big\{\langle \eta(\lambda), \tau(\cdot, s) \rangle + \log(b)\big\}|_{s=S} \leq \\
&D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big\} \geq 1 - \varepsilon\,.
\end{aligned}
$$

Proof. Applying Markov's inequality to the non-negative random variable $c(\lambda, S)$ yields for $r \in \mathbb{R}$, $\lambda \in \Lambda$:

$$\mathbb{P}\big\{c(\lambda, S) > \exp(r)\big\} \leq \frac{\mathbb{E}\{c(\lambda, S)\}}{\exp(r)} \leq \exp(-r)\,.$$

This implies that $\mathbb{P}\{\kappa(\lambda, S) > r\} \leq \exp(-r)$. Hence, Lemma 3.2.2 is applicable and gives:

$$\mathbb{P}\Big\{\sup_{\lambda\in\Lambda} \kappa(\lambda, S) \leq \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big\} \geq 1 - \varepsilon\,.$$

Using Lemma 3.2.1 gives:

$$\mathbb{P}\Big\{\sup_{\lambda\in\Lambda} \sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)} \mathbb{Q}\big\{\langle \eta(\lambda), \tau(\cdot, s)\rangle + \log(b)\big\}|_{s=S} - D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) \\ \leq \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big\} \geq 1 - \varepsilon\,.$$

Simply rearranging and reformulating yields the result. □

Remark 3.2.1 (i) Note that the statement is still true for a data-dependent prior $\mathbb{P}_H$: Given another independent data set S', one needs to assume that $\mathbb{E}\{c(\lambda, (S, S'))\} \leq 1$.

(ii) In Section 3.3 we provide sufficient conditions s.t. $\mathbb{E}\{c(\lambda, S)\} \leq 1$ holds for all $\lambda > 0$.

(iii) Typically, $\mathcal{K}$ is (related to) the covering-number of Λ, and $\log(\mathcal{K})$ bears the intrinsic dimension of Λ. Thus, in full generality, it might be large. For us, however, it only has a minor influence, since $\Lambda \subset \mathbb{R}$, and the empirical risk is typically much larger.

(iv) The monograph of Hellström et al. (2025) proposes a similar general PAC-Bayesian theorem. On first sight, it seems like theirs is more general than ours. However, Example 3.2.1 clarifies this. Furthermore, we want to remark that the first version of our Theorem 3.2.3 appeared in 2022.

From now on we set $b \equiv 1$, such that $\log(b) \equiv 0$. The following corollary shows an example of how to transform Theorem 3.2.3 into a high-probability bound on the risk.

Corollary 3.2.4 (PAC-Bayesian Generalization Bound) *Denote τ by $\tau = (\tau^{(1)}, \tau^{(r)})$ with $\tau^{(r)} := (\tau^{(2)}, \ldots, \tau^{(k)})$ and η by $\eta = (\eta^{(1)}, \eta^{(r)})$ with $\eta^{(r)} := (\eta^{(2)}, \ldots, \eta^{(k)})$. If $\tau^{(1)} = \mathcal{R} - \hat{\mathcal{R}}$ and $\eta^{(1)} > 0$, for any $\lambda \in \Lambda$, $s \in \mathcal{P}^N$ and $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$, the condition*

$$\mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle\} \leq D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_{\mathcal{O}}\,, \tag{A}$$

is equivalent to

$$\mathbb{Q}\{\mathcal{R}\} \leq \mathbb{Q}\{\hat{\mathcal{R}}(\cdot, s)\} + \frac{1}{\eta^{(1)}(\lambda)}\Big(D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big)\Big) \\ + \frac{1}{\eta^{(1)}(\lambda)}\Big(\mathcal{C}_{\mathcal{O}} - \mathbb{Q}\{\langle \eta^{(r)}(\lambda), \tau^{(r)}(\cdot, s)\rangle\}\Big)\,. \tag{B}$$

In particular, if Theorem 3.2.3 applies, we can replace (A) with (B).

Proof. The two formulas are simply rewritings of each other: By assumption, bilinearity and definition of the Euclidean scalar product, and linearity of the integral, the term $\mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle\}$ can be split up as:

$$\begin{aligned}
&\mathbb{Q}\{\langle \eta(\lambda), \tau(\cdot, s)\rangle\} \\
&= \mathbb{Q}\{\eta^{(1)}(\lambda)(\mathcal{R} - \hat{\mathcal{R}}(\cdot, s))\} + \mathbb{Q}\{\langle \eta^{(r)}(\lambda), \tau^{(r)}(\cdot, s)\rangle\} \\
&= \eta^{(1)}(\lambda)\mathbb{Q}\{\mathcal{R}\} - \eta^{(1)}(\lambda)\mathbb{Q}\{\hat{\mathcal{R}}(\cdot, s)\} + \mathbb{Q}\{\langle \eta^{(r)}(\lambda), \tau^{(r)}(\cdot, s)\rangle\} .
\end{aligned}$$

Simply rearranging the terms then yields the result, as $\eta^{(1)} > 0$. □

Using similar rearrangements, the following example relates Theorem 3.2.3 to other known PAC-Bayesian bounds:

Example 3.2.1 (i) Assume that the loss-function is bounded, that is, $0 \leq \ell \leq C$, and define $\Lambda = \{\lambda\}$, $b \equiv 1$, $\mathcal{C}_{\mathcal{O}} = 0$, $\tau(h, s) := (\mathcal{R}(h) - \hat{\mathcal{R}}(h, s),\ C^2)$, and $\eta(\lambda) := (\lambda, -\frac{\lambda^2}{8})$. Then we recover Catoni's bound (Catoni, 2003; Alquier, 2024):

$$\mathbb{P}\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)\ :\ \mathbb{Q}\{\mathcal{R}\} \leq \mathbb{Q}\{\hat{\mathcal{R}}(\cdot, s)\}|_{s=S} + \frac{1}{\lambda}\Big(D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\Big(\frac{1}{\varepsilon}\Big) + \frac{\lambda^2 C^2}{8N}\Big)\Big\} \geq 1 - \varepsilon .$$

(ii) Assume that ℓ takes values in $[0, 1]$ and let $D : [0, 1]^2 \to \mathbb{R}$ be convex. Further, define $\Lambda = \{1\}$, $b \equiv 1$, $\mathcal{C}_{\mathcal{O}} = 0$, $\tau(h, s) := \big(nD(\mathcal{R}(h), \hat{\mathcal{R}}(h, s)),\ \log(\mathbb{E}_{(S,H)}\{\exp(nD(\mathcal{R}, \hat{\mathcal{R}}))\})\big)$, and $\eta(\lambda) := (1, -1)$. Then we get:

$$\mathbb{P}\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) : \mathbb{Q}\{nD(\mathcal{R}, \hat{\mathcal{R}}(\cdot, s))\}|_{s=S} \leq D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\Big(\frac{1}{\varepsilon}\Big) + \log(\mathbb{E}_{(S,H)}\{\exp(nD(\mathcal{R}, \hat{\mathcal{R}}))\})\Big\} \geq 1 - \varepsilon .$$

Dividing both sides by $n \geq 1$ and applying Jensen's inequality to the left term, we get the bound of Germain et al. (2009). Similarly, one can obtain the bound of Bégin et al. (2014).

(iii) Consider two measurable functions $f, g : \mathcal{H} \times \mathcal{P}^N \to \mathbb{R}$, and define $\Lambda = \{\lambda\}$, $b \equiv 1$, $\mathcal{C}_{\mathcal{O}} = 0$, $\tau(h, s) := (f(h, s), g(h, s))$, and $\eta(\lambda) := (\lambda, -\lambda)$. Then our assumption $\mathbb{E}_S\{c(\lambda, \cdot)\} \leq 1$ reads $\mathbb{E}_{(S,H)}\{\exp(\lambda(f - g))\} \leq 1$, which is the same assumption as in Theorem 5.1 of Hellström et al. (2025). Similarly, defining $\eta(\lambda) := (1, -1)$ and $\tau(h, s) := \big(f(h, s),\ \log(\mathbb{E}_{(S,H)}\{\exp(f)\})\big)$, we also get a similar bound as in Proposition 5.2 of Hellström et al. (2025):

$$\mathbb{P}\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) : \mathbb{Q}\{f(\cdot, s)\}|_{s=S} \leq D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\Big(\frac{1}{\varepsilon}\Big) + \log\big(\mathbb{E}_{(S,H)}\{\exp(f)\}\big)\Big\} \geq 1 - \varepsilon .$$

3.3 Application to Learning-to-Optimize

Here, for our setting in Subsection 3.1, we consider properties of optimization algorithms that assert the necessary condition $\mathbb{E}\{c(\lambda, S)\} \leq 1$, for all $\lambda \in \Lambda$, to employ the PAC-Bayesian bound from Section 3.2.

3.3.1 Using Worst-Case Bounds

In the next theorem, the additional assumption on $\mathcal{A}$ is sufficient to ensure the conditions of Theorem 3.2.3. Essentially, it requires the loss of the algorithm's output to be bounded. It can be used, for example, if one wants to combine the learning procedure with existing worst-case guarantees. Yet, as shown in Section 3.3.2, it is too restrictive to achieve a significant acceleration compared to the standard choices from a worst-case analysis.

Since one of the steps in the proof is used later on again, we formulate it as a short lemma:

Lemma 3.3.1 *Let $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathbb{R}$ be a random variable, such that there exists a constant $b \in \mathbb{R}$ with $\mathbb{E}\{\exp(U)\} \leq \exp(b)$. Then it holds that:*

$$\mathbb{E}\{\exp(U - b)\} \leq 1 \,.$$

Proof. This follows immediately from the properties of the exponential function and the fact that b is a constant:

$$\begin{aligned}\mathbb{E}\{\exp(U - b)\} &= \mathbb{E}\{\exp(U)\exp(-b)\} = \mathbb{E}\{\exp(U)\}\exp(-b) \\ &\leq \exp(b)\exp(-b) \leq 1 \,.\end{aligned}$$

□

Furthermore, we nee the following result, which states that non-negative random variables with finite second moment satisfy a one-sided sub-Gaussian inequality.

This result can be found on p.47 in the book by Boucheron et al. (2013).

Lemma 3.3.2 *Let U be a non-negative random variable with finite second moment. Then, for every $\lambda > 0$ it holds*

$$\mathbb{E}\{\exp(-\lambda(U - \mathbb{E}\{U\}))\} \leq \exp\Big(\frac{\lambda^2}{2}\mathbb{E}\{U^2\}\Big) \,.$$

Theorem 3.3.3 *Suppose that P and ℓ satisfy Assumption 3.1.1, and suppose that $\mathcal{A}$ satisfies Assumption 3.1.2. Further, assume that there is a measurable function $\rho : \mathcal{H} \to [0, \infty)$, such that for every $h \in \mathcal{H}$ it holds that $\ell(h, \cdot) \leq \rho(h)\ell(x^{(0)}, \cdot)$ $\mathbb{P}_P$-a.s. Furthermore, let S be a corresponding i.i.d. data set of size $N \in \mathbb{N}$. Finally, assume that $\mathbb{E}\{\ell(x^{(0)}, P)^2\} < \infty$, and define $\eta : (0, \infty) \to \mathbb{R}^2$ and $\tau : \mathcal{H} \times \mathcal{P}^N \to \mathbb{R}^2$ through:*

$$\eta(\lambda) := \Big(\lambda, \, -\frac{\lambda^2}{2}\Big) \,, \quad \tau(h, s) := \Big(\mathcal{R}(h) - \hat{\mathcal{R}}(h, s), \, \frac{\rho^2(h)}{N}\mathbb{E}\{\ell(x^{(0)}, P)^2\}\Big) \,.$$

Then it holds that $\mathbb{E}\{c(\lambda, S)\} \leq 1$ for all $\lambda > 0$.

Please recall our short-hand notation $\ell(h, p) = \ell(\mathcal{A}(h, p), p) = \ell(\mathcal{A}(h, x^{(0)}, p), p)$, that is, $\ell(x^{(0)}, p)$ evaluates the loss function at $x^{(0)}$, while $\ell(h, p)$ evaluates the loss function at the output of the algorithm with hyperparameters h and starting from $x^{(0)}$.

Proof. Since H and S are independent, their joint distribution is given by the product measure, that is, $\mathbb{P}_{(S,H)} = \mathbb{P}_S \otimes \mathbb{P}_H$. Thus, by Fubini's theorem we get:

$$\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(H) - \hat{\mathcal{R}}(H, S))\big)\Big\} = \mathbb{E}\Big\{\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(h) - \hat{\mathcal{R}}(h, S))\big)\Big\}\Big|_{h=H}\Big\} \,.$$

Hence, first consider the inner integral for a fixed $h \in \mathcal{H}$. Then, by definition and the i.i.d. assumption it holds:

$$\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S))\big)\Big\} = \mathbb{E}\Big\{\exp\Big(-\frac{\lambda}{N}\sum_{i=1}^{N}\big(\ell(h,P_i) - \mathbb{E}\{\ell(h,P)\}\big)\Big)\Big\}$$
$$= \prod_{i=1}^{N}\mathbb{E}_P\Big\{\exp\Big(-\frac{\lambda}{N}\big(\ell(h,\cdot) - \mathbb{E}_P\{\ell(h,\cdot)\}\big)\Big)\Big\}.$$

The loss-function is non-negative and, by assumption on $\mathcal{A}$, can be bounded $\mathbb{P}_P$-a.s., such that $\mathbb{E}_P\{\ell(h,\cdot)^2\} \leq \rho(h)^2\mathbb{E}_P\big\{\ell(x^{(0)},\cdot)^2\big\} < \infty$. Thus, we have that $\ell(h,P)$ is a non-negative random variable with finite second-moment for every $h \in \mathcal{H}$. Applying Lemma 3.3.2, we get that:

$$\mathbb{E}_P\Big\{\exp\big(-\frac{\lambda}{N}(\ell(h,\cdot) - \mathbb{E}_P\{\ell(h,\cdot)\})\big)\Big\} \leq \exp\Big(\frac{\lambda^2}{2N^2}\mathbb{E}_P\{\ell(h,\cdot)^2\}\Big)$$
$$\leq \exp\Big(\frac{\lambda^2}{2N^2}\rho(h)^2\mathbb{E}_P\{\ell(x^{(0)},\cdot)^2\}\Big).$$

Therefore we have the following bound:

$$\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S))\big)\Big\} \leq \exp\Big(\frac{\lambda^2}{2N}\rho(h)^2\mathbb{E}_P\big\{\ell(x^{(0)},\cdot)^2\big\}\Big).$$

Since the right-hand side does not depend on S, applying Lemma 3.3.1 yields $\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S)) - \frac{\lambda^2}{2N}\rho(h)^2\mathbb{E}_P\{\ell(x^{(0)},\cdot)^2\}\big)\Big\} \leq 1$. Since H and S are independent, and $h \in \mathcal{H}$ was arbitrary, this inequality does hold $\mathbb{P}_H$-a.s, and we directly get the bound $\mathbb{E}\big\{\exp\big(\lambda(\mathcal{R}(H) - \hat{\mathcal{R}}(H,S)) - \frac{\lambda^2}{2N}\rho(H)^2\mathbb{E}_P\{\ell(x^{(0)},\cdot)^2\}\big)\big\} \leq 1$. Now, again by Fubini's theorem, one can also switch the order of integration to get:

$$\mathbb{E}\Big\{\mathbb{E}\Big\{\exp\big(\lambda(\mathcal{R}(H) - \hat{\mathcal{R}}(H,s)) - \frac{\lambda^2}{2N}\rho(H)^2\mathbb{E}_P\{\ell(x^{(0)},\cdot)^2\}\big)\Big\}\Big|_{s=S}\Big\} \leq 1.$$

Inserting the definition of η and τ gives $\mathbb{E}\big\{\mathbb{E}\big\{\exp\big(\langle\eta(\lambda),\tau(H,s)\rangle\big)\big\}\big|_{s=S}\big\} \leq 1$. Here, the inner term is the same as $\mathbb{E}\big\{\exp\big(\langle\eta(\lambda),\tau(H,s)\rangle\big)\big\} = c(\lambda,s)$. In summary, this is the same as $\mathbb{E}\{c(\lambda,S)\} \leq 1$. □

Remark 3.3.1 The argument still works for a data-dependent prior, if the corresponding data sets S' and S are independent: While interchanging the integration w.r.t. S' and H is not allowed, an interchange w.r.t. H and S is still valid (under the integral), that is, for a function f it would hold $\mathbb{E}\{f(H,S,S')\} = \mathbb{E}_{S'}\big\{\mathbb{E}_{H|S'=s'}\big\{\mathbb{E}_S\big\{f(h,\cdot,s')\big\}|_{h=H}\big\}\big\} = \mathbb{E}_{S'}\big\{\mathbb{E}_S\big\{\mathbb{E}_{H|S'=s'}\big\{f(\cdot,s,s')\big\}|_{s=S}\big\}\big\}$, and the inner term is ≤ 1 in any case.

3.3.2 Conditional Boundedness

Typically, the previous approach is too restrictive, because the boundedness assumption on $\mathcal{A}$ already requires theoretical worst-case estimates *almost surely*. For example, if $(\ell(\cdot,p))_{p\in\mathcal{P}}$ is a family of quadratic functions, and one tries to learn the step-size of gradient descent, the boundedness prevents step-size parameters that lie outside the worst-case convergence regime, as they would lead to a diverging behavior, which increases the

incurred empirical risk dramatically. Thus, to motivate the upcoming discussion, consider the following thought-experiment:

Example 3.3.1 Consider $\ell(x,p) := \frac{p}{2}x^2$ and assume that the chosen algorithm is gradient descent, that is $x^{(t+1)} = x^{(t)} - h\ell'(x^{(t)},p)$. For a given p, the optimal step-size is $h = \frac{1}{p}$, which gives convergence in one step. Then, if p is given by samples from the distribution $\mathbb{P}_p = 0.99\delta_1 + 0.01\delta_{100}$, a worst-case analysis would suggest to take $h_w = \frac{1}{100}$. In this case, we would have an algorithm that converges in a single step for 1% of the problem instances, while having a linear convergence rate (of the iterates) of $(\frac{99}{100})^t$ for the other 99%. Another choice is to take $h_d = 1$, which leads to an algorithm that does converge in a single step for 99% of the problem instances, but diverges in 1% of the cases. By restricting to the 99% of the cases where convergence does occur, the overall difference in speed is drastic.

Remark 3.3.2 Note that this example also highlights an important aspect: As long as we insist on convergence for all instances, we simply cannot choose a step-size that is substantially larger[1] than h_w! Even if we consider the average case, the resulting step-size has to lie within the interval $(0, \frac{2}{100})$, such that the average-case performance and the worst-case performance are roughly the same.

1: To be precise: The chosen step-size would still have to lie in the interval $(0, \frac{2}{L})$, which determines the standard choice for L-smooth functions. Thus, the rate for 99% of the problems could actually be improved from $(\frac{99}{100})^t$ to roughly $(\frac{98}{100})^t$.

Hence, in this section, a different approach is taken: We actually allow for divergence, if it only occurs in rare cases with a controllable probability, that is, "almost surely" is relaxed to "with a sufficiently large probability". Essentially, we only consider the loss for all those hyperparameters, where the loss is bounded by a certain constant, as well as the probability for that to occur. Then, in Section 3.4, we develop a technique that allows the user to actually control this probability. Clearly, a stronger guarantee trades for convergence speed.

Definition 3.3.1 *Given a measurable function $\sigma : \mathcal{P} \to \mathbb{R}$, the (parametric)* sublevel set $\mathsf{L}_\sigma \subset \mathcal{H} \times \mathcal{P}$ *is defined as* $\mathsf{L}_\sigma := \{(h,p) \in \mathcal{H} \times \mathcal{P} : \ell(h,p) \leq \sigma(p)\}$. *The sections of L_σ for fixed $h \in \mathcal{H}$ will be denoted by $\mathsf{L}_{\sigma,h}$.*

In order to use this set in a measure-theoretic framework, we first show that L_σ is indeed measurable. This is not obvious, as the loss function and the algorithm are composed in a non-standard way.

Lemma 3.3.4 *The sublevel set L_σ is measurable.*

Proof. As σ is assumed to be measurable, it suffices to show that the specific composition of ℓ and $\mathcal{A}$ is measurable, that is, $\ell \circ \mathcal{A} : \mathcal{H} \times \mathcal{P} \to [0,+\infty]$, $(h,p) \mapsto \ell(\mathcal{A}(h,p),p)$ is measurable w.r.t. $\mathfrak{B}(\mathcal{H}) \otimes \mathfrak{B}(\mathcal{P})$ and $\mathfrak{B}([0,+\infty])$. Since $\ell \geq 0$ is measurable, there exists a sequence of simple functions ℓ_n with $\ell = \lim_{n\to\infty} \ell_n$. Thus, since limits of measurable functions are measurable, it suffices to consider the case of a simple function $\ell : \mathbb{R}^n \times \mathcal{P} \to \mathbb{R}$. Then, however, it suffices to consider indicator functions of the form $\mathbb{1}_{\mathsf{A}}$ for a measurable set $\mathsf{A} \in \mathfrak{B}(\mathbb{R}^n) \otimes \mathfrak{B}(\mathcal{P})$. Since the product-$\sigma$-algebra is generated by cylinder sets, it actually suffices to consider the case $\ell = \mathbb{1}_{\mathsf{B}\times\mathsf{D}}$, that is, $(\ell \circ \mathcal{A})(h,p) = \mathbb{1}_{\mathsf{B}\times\mathsf{D}}(\mathcal{A}(h,p),p) = \mathbb{1}_{\mathsf{B}}(\mathcal{A}(h,p))\mathbb{1}_{\mathsf{D}}(p)$. The second term is obviously measurable, and the first term is measurable as a composition of two measurable functions. □

Basically, a function f is called simple if it is a finite weighted sum of indicator functions, that is, if there are disjoint sets A_i, $i = 1,\dots,K$, such that $f = \sum_{i=1}^K a_i \mathbb{1}_{\mathsf{A}_i}$.

More formally, one would use a monotone-class argument to show that it is sufficient to considers cylinder sets.

This result further implies that the sections $\mathsf{L}_{\sigma,h}$ are measurable, too. Since $\mathscr{H}$ and $\mathscr{P}$ are Polish spaces, the product $\mathscr{H}\times\mathscr{P}$ is again Polish. Hence, there exists a regular version of the conditional distribution of P, given H, that is, a kernel $\mathscr{H} \to \mathscr{P}$, $(h, \mathsf{B}) \mapsto \mathbb{P}_{P|H=h}\{\mathsf{B}\}$. By Witting (1985, Thm. 1.122, p.124), this determines a regular version of the conditional distribution of (H, P), given H, through $\mathscr{H} \to \mathscr{H} \times \mathscr{P}$, $(h, \mathsf{B}) \mapsto \mathbb{P}_{(H,P)|H=h}\{\mathsf{B}\} := \mathbb{P}_{P|H=h}\{\mathsf{B}_h\}$, and we have $\mathbb{P}_H$-a.s. the equality $\mathbb{P}\{(H, P) \in \mathsf{B} \mid H = h\} = \mathbb{P}_{P|H=h}\{\mathsf{B}_h\}$. In particular, this applies to the sublevel set L_σ, and the map $h \mapsto \mathbb{P}_{P|H=h}\{\mathsf{L}_{\sigma,h}\}$ is measurable.

Definition 3.3.2 *Let L_σ be a parametric sublevel set. Define the* sublevel probability *as the measurable function $h \mapsto \mathbb{p}(h) := \mathbb{P}_{P|H=h}\{\mathsf{L}_{\sigma,h}\}$.*

Lemma 3.3.5 *Suppose Assumption 3.1.2 holds, and let $\mathbb{P}_P\{\mathsf{L}_{\sigma,h}\} > 0$ for every $h \in \mathscr{H}$. Then we have $\mathbb{P}_H$-a.s.:*

(i) $\mathbb{p}(h) = \mathbb{P}_P\{\mathsf{L}_{\sigma,h}\}$,

(ii) $\mathbb{E}\{\ell(H, P)\mathbb{1}_{\mathsf{L}_\sigma}(H, P) \mid H = h\} = \mathbb{E}_P\{\ell(h, \cdot)\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}$
$= \mathbb{p}(h) \cdot \mathbb{E}_P\{\ell(h, \cdot) \mid \mathsf{L}_{\sigma,h}\}$.

Proof. By the independence of P and H, we have $\mathbb{P}_H$-a.s.

$$\begin{aligned}\mathbb{E}\{\ell(H, P)\mathbb{1}_{\mathsf{L}_\sigma}(H, P) \mid H = h\} &= \int_{\mathscr{P}} \ell(h, p)\mathbb{1}_{\mathsf{L}_{\sigma,h}}(p)\, \mathbb{P}_P(dp) \\ &= \mathbb{E}_P\{\ell(h, \cdot)\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}\,,\end{aligned}$$

which shows the first equality of (ii). Since $\mathbb{P}_P\{\mathsf{L}_{\sigma,h}\} > 0$, the elementary conditional expectation is defined as

$$\mathbb{E}_P\{\ell(h, \cdot) \mid \mathsf{L}_{\sigma,h}\} = \frac{\mathbb{E}_P\{\ell(h, \cdot)\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}}{\mathbb{P}_P\{\mathsf{L}_{\sigma,h}\}}\,.$$

Again by independence we have $\mathbb{P}_H$-a.s. the equality

$$\mathbb{p}(h) = \mathbb{P}_{P|H=h}\{\mathsf{L}_{\sigma,h}\} = \mathbb{P}_P\{\mathsf{L}_{\sigma,h}\}\,,$$

which shows (i) and the second equality of (ii). □

This construction allows us to give a more fine-grained analysis of the algorithm, as it allows for trading the boundedness assumption for the sublevel probability. This basically extends a worst-case analysis, which would correspond to an uniform upper bound. Motivated by Lemma 3.3.5, we define the *sublevel risk* as the expect loss *conditioned* on the sublevel set:

Definition 3.3.3 *Let L_σ be a parametric sublevel set. Then the* sublevel risk *$\mathcal{R}_\sigma : \mathscr{H} \to [0, +\infty]$ is defined as the conditional expectation of the loss given $\mathsf{L}_{\sigma,h}$:*

$$\mathcal{R}_\sigma(h) := \mathbb{E}_P\{\ell(h, \cdot) \mid \mathsf{L}_{\sigma,h}\} = \begin{cases} \frac{1}{\mathbb{p}(h)}\mathbb{E}_P\{\ell(h, \cdot)\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}, & \text{if } \mathbb{p}(h) > 0\,; \\ 0, & \text{otherwise}\,. \end{cases}$$

Given a data set $S = (P_1, \ldots, P_N)$, the empirical sublevel risk *$\hat{\mathcal{R}}_\sigma : \mathscr{H} \times$*

$\mathcal{P}^N \to [0,+\infty]$ *is defined as*

$$(h,S) \mapsto \hat{\mathcal{R}}_\sigma(h,S) := \frac{1}{\mathbb{p}(h)}\frac{1}{N}\sum_{i=1}^{N} \mathbb{1}_{\mathsf{L}_{\sigma,h}}(P_i)\ell(h,P_i)\,.$$

Example 3.3.2 Consider again the setting from Example 3.3.1, define $\sigma(p) := \ell(x^{(0)},p)$, and assume that the algorithm gets stopped after $T \in \mathbb{N}$ iterations. Then the choice $h_w = \frac{1}{100}$ yields $\mathbb{p}(h_w) = 1$ and we have $\mathbb{P}_P$-a.s. the equality $\mathsf{L}_{\sigma,h_w} = \mathcal{P}$, such that we get:

$$\begin{aligned}\mathcal{R}_\sigma(h_w) &= \mathbb{E}\left\{\frac{P}{2}\ell(h_w,P)\right\} = \mathbb{E}\left\{\left(1-\frac{P}{100}\right)^{2T}\ell(x^{(0)},P)\right\}\\ &= 0.01\cdot\left(1-\frac{100}{100}\right)^{2T}\ell(x^{(0)},100) + 0.99\cdot\left(1-\frac{1}{100}\right)^{2T}\ell(x^{(0)},1)\\ &= \left(\frac{99}{100}\right)^{2T}\frac{0.99}{2}(x^{(0)})^2\end{aligned}$$

On the other hand, for the choice $h_d = 1$ we have that $\mathbb{p}(h_w) = 0.99$ and $\mathsf{L}_{\sigma,h_d} = \{P=1\}$ $\mathbb{P}_P$-a.s., such that it holds:

$$\begin{aligned}\mathcal{R}_\sigma(h_d) &= \frac{1}{0.99}\mathbb{E}\left\{\frac{P}{2}\ell(h_d,P)\mathbb{1}_{\mathsf{L}_{\sigma,h}}\right\} = \frac{1}{0.99}\mathbb{E}\left\{(1-P)^{2T}\ell(x^{(0)},P)\mathbb{1}_{\{P=1\}}\right\}\\ &= 0\,.\end{aligned}$$

This example shows that, if we change the perspective and allow for divergence on a small, rather unlikely subset of problem instances, the worst-case optimal choice is not optimal anymore. However, it also highlights that we have to be very cautious in the interpretation of the sublevel risk, because it *ignores* all instances for which divergence occurs.

The following theorem is a direct generalization of Theorem 3.3.3. Especially, note that the additional assumption on $\mathcal{A}$ is not needed anymore.

Theorem 3.3.6 *Suppose that P and ℓ satisfy Assumption 3.1.1, and suppose that $\mathcal{A}$ satisfies Assumption 3.1.2. Further, let S be a corresponding i.i.d. data set of size $N \in \mathbb{N}$, and let L_σ be a parametric sublevel set with sublevel probability $\mathbb{p}$. Assume that $\mathbb{P}_H\{\mathbb{p} > 0\} = 1$ and $\mathbb{E}_P\{\sigma^2\} < \infty$. Define $\eta : (0,\infty) \longrightarrow \mathbb{R}^2$ and $\tau : \mathcal{H}\times\mathcal{P}^N \longrightarrow \mathbb{R}^2$ as*

$$\begin{aligned}\eta(\lambda) &:= \left(\lambda,\ -\frac{\lambda^2}{2}\right),\\ \tau(h,s) &:= \left(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h,s),\ \frac{1}{\mathbb{p}(h)^2 N}\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}\right).\end{aligned}$$

Then, for all $\lambda > 0$, it holds that $\mathbb{E}\{c(\lambda,S)\} \leq 1$.

Proof. The proof is very similar to the proof of Theorem 3.3.3 and basically uses the same reasoning. Let $\ell_\sigma(h,p) := \mathbb{1}_{\mathsf{L}_{\sigma,h}}(p)\ell(h,p)$. Since H and S are independent, one gets from Fubini's theorem:

$$\mathbb{E}\{\exp(\lambda(\mathcal{R}_\sigma(H) - \hat{\mathcal{R}}_\sigma(H,S)))\} = \mathbb{E}\{\mathbb{E}\{\exp(\lambda(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h,S)))\}|_{h=H}\}\,.$$

Thus, first consider a fixed $h \in \mathcal{H}$ with $\mathbb{p}(h) > 0$. Then, by definition and

the i.i.d. assumption, it holds that:

$$\begin{aligned}
&\mathbb{E}\{\exp(\lambda(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h,S)))\} \\
&= \mathbb{E}\Big\{\exp\Big(-\frac{\lambda}{N\mathbb{p}(h)}\sum_{i=1}^{N}(\ell_\sigma(h,P_i) - \mathbb{E}_P\{\ell_\sigma(h,\cdot)\})\Big)\Big\} \\
&= \prod_{i=1}^{N}\mathbb{E}_P\Big\{\exp\Big(-\frac{\lambda}{N\mathbb{p}(h)}(\ell_\sigma(h,\cdot) - \mathbb{E}_P\{\ell_\sigma(h,\cdot)\})\Big)\Big\}\,.
\end{aligned}$$

$\ell_\sigma(h,\cdot)$ is non-negative, and by definition of the parametric sublevel set has a finite second-moment, that is $\mathbb{E}_P\{\ell_\sigma(h,\cdot)^2\} \leq \mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\} < \infty$. Hence, by Lemma 3.3.2 we get that $\mathbb{E}_P\big\{\exp\big(-\frac{\lambda}{N\mathbb{p}(h)}(\ell_\sigma(h,\cdot) - \mathbb{E}_P\{\ell_\sigma(h,\cdot)\})\big)\big\} \leq \exp\big(\frac{\lambda^2}{2N^2\mathbb{p}(h)^2}\mathbb{E}\{\ell_\sigma(h,\cdot)^2\}\big)$. Thus:

$$\mathbb{E}\big\{\exp(\lambda(\mathcal{R}_\sigma(h,\cdot) - \hat{\mathcal{R}}_\sigma(h,S)))\big\} \leq \exp\Big(\frac{\lambda^2}{2N\mathbb{p}(h)^2}\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}\Big)\,.$$

The right-hand side does not depend on S. Thus, by applying Lemma 3.3.1 we get that $\mathbb{E}\big\{\exp\big(\lambda(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h,S)) - \frac{\lambda^2}{2N\mathbb{p}(h)^2}\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}\big)\big\} \leq 1$. As this holds for any h with $\mathbb{p}(h) > 0$, which in turn does hold $\mathbb{P}_H$-a.s., we get

$$\mathbb{E}\Big\{\exp\Big(\lambda(\mathcal{R}_\sigma(H) - \hat{\mathcal{R}}_\sigma(H,S)) - \frac{\lambda^2}{2N\mathbb{p}(H)^2}\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}|_{h=H}\Big)\Big\} \leq 1\,.$$

Changing the order of integration with Fubini's theorem, we get:

$$\mathbb{E}\Big\{\mathbb{E}\Big\{\exp\Big(\lambda(\mathcal{R}_\sigma(H) - \hat{\mathcal{R}}_\sigma(H,s)) - \frac{\lambda^2}{2N\mathbb{p}(H)^2}\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}|_{h=H}\Big)\Big\}\Big|_{s=S}\Big\} \leq 1\,.$$

By inserting the definition of η and τ, this can equivalently be written as $\mathbb{E}\big\{\mathbb{E}\{\exp(\langle\eta(\lambda),\tau(H,s)\rangle)\}\big|_{s=S}\big\} \leq 1$. Thus, in total we get $\mathbb{E}\{c(\lambda,S)\} \leq 1$, because the inner term can be rewritten as $\mathbb{E}\{\exp(\langle\eta(\lambda),\tau(H,s)\rangle)\} = \int_{\mathcal{H}}\exp(\langle\eta(\lambda),\tau(h,s)\rangle)\,\mathbb{P}_H(dh) = c(\lambda,s)$. □

Remark 3.3.3 The assumption $\mathbb{P}_H\{\mathbb{p} > 0\} = 1$ states that, under the prior, the algorithm should be able to "reach" the sublevel set. This is a constraint on the *support* of $\mathbb{P}_H$, which is not satisfied without further ado. Section 3.4 provides a construction for achieving this.

Example 3.3.3 Combining Theorem 3.3.6 and Theorem 3.2.3, we get that:

$$\mathbb{P}\Big\{\forall\lambda\in\Lambda,\ \forall\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H) : \mathbb{Q}\{\mathcal{R}_\sigma\} \leq \mathbb{Q}\{\hat{\mathcal{R}}_\sigma(s)\}|_{s=S} + \\ + \frac{D_{\mathrm{KL}}(\mathbb{Q}\,\|\,\mathbb{P}_H) + \log\big(\frac{\mathcal{K}}{\varepsilon}\big) + \mathcal{C}_{\mathcal{O}} + \frac{\lambda^2}{2N}\mathbb{Q}\Big\{\frac{\mathbb{E}_P\{\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\}|_{h=\cdot}}{\mathbb{p}^2}\Big\}}{\lambda}\Big\} \geq 1-\varepsilon\,.$$

For every fixed $\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)$, optimizing over λ (assuming that λ^* is

We bascially have a term $f(\lambda) = \frac{A}{\lambda} + \lambda B$, such that $f'(\lambda) = -\frac{A}{\lambda^2} + B$. This vanishes for $\lambda = \pm\sqrt{\frac{A}{B}}$ and, if we insert $\lambda^* := \sqrt{\frac{A}{B}}$, we get $f(\lambda^*) = 2\sqrt{AB}$.

attained in Λ), gives:

$$\mathbb{P}\Bigg\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) : \mathbb{Q}\{\mathcal{R}_\sigma\} \leq \mathbb{Q}\{\hat{\mathcal{R}}_\sigma(s)\}|_{s=S} + \\ + \sqrt{\frac{2\Big(D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big)\mathbb{Q}\Big\{\frac{\mathbb{E}_P[\sigma^2 \mathbb{1}_{\mathrm{L}_{\sigma,h}}]|_{h=\cdot}}{\mathbb{p}(\cdot)^2}\Big\}}{N}}\Bigg\} \geq 1 - \varepsilon\,.$$

Now, a typical performance-measure in optimization is *complexity*, that is, how many iterations are needed to reach a loss smaller or equal to ϑ. Thus, specifying $\sigma \equiv \vartheta$ and assuming that $\mathbb{p}(H) \geq \mathbb{p}_I$ a.s., this gives rise to:

$$\mathbb{P}\Bigg\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) : \mathbb{Q}\{\mathcal{R}_\sigma\} \leq \mathbb{Q}\{\hat{\mathcal{R}}_\sigma(s)\}|_{s=S} + \\ + \frac{\vartheta}{\mathbb{p}_I}\sqrt{\frac{2\Big(D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\Big(\frac{\mathcal{K}}{\varepsilon}\Big) + \mathcal{C}_{\mathcal{O}}\Big)}{N}}\Bigg\} \geq 1 - \varepsilon\,.$$

3.4 Implementing the Non-Divergence – Speed Trade-Off

In Subsection 3.3.2, and as stressed in Example 3.3.2, care has to be taken in the choice of the prior $\mathbb{P}_H$: Just minimizing the upper bound as much as possible can lead to a neglect of a high sublevel probability, that is, the algorithm is especially fast on a small subset of the parameters, while it diverges for the rest. This is due to the fact that the term $\frac{1}{\mathbb{p}(h)}$ might not compensate for the smaller sublevel risk. Thus, if a certain sublevel probability $\varepsilon_{\mathrm{conv}} \in [0, 1]$ has to be ensured, one has to enforce it. In the case of PAC-Bayesian learning with absolutely continuous distributions, it suffices to have this property for the prior:

Lemma 3.4.1 *Let $\varepsilon_{\mathrm{conv}} \in [0, 1]$ and assume that $\mathbb{p}(H) \geq \varepsilon_{\mathrm{conv}}$ a.s. Then, for every $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ we have $\mathbb{Q}\{\mathbb{p} < \varepsilon_{\mathrm{conv}}\} = 0$.*

Proof. By assumption we have $\mathbb{P}_H\{\mathbb{p} < \varepsilon_{\mathrm{conv}}\} = 0$. Thus, the result follows directly by definition of absolute continuity. □

Though the proof is trivial, this lemma has a very important consequence, which we want to stress: If one can guarantee that a required property is satisfied for the prior, it will be *preserved* during the PAC-Bayesian learning process, that is, if the prior only puts mass on hyperparameters that ensure a certain sublevel probability, the posterior will do the same. How to enforce such constraints during construction of the prior is discussed next.

3.4.1 Sampling under Probabilistic Constraints

In this section, we describe a methodology that allows for sampling from a distribution that is *probabilistically constrained* in the following sense: We are given two independent random variables $U : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{U}$, $V : (\Omega, \mathfrak{A}, \mathbb{P}) \to \mathcal{V}$ taking values in the Polish spaces $\mathcal{U}$ and $\mathcal{V}$, with joint and marginal distributions $\mathbb{P}_{(U,V)}$, $\mathbb{P}_U$ and $\mathbb{P}_V$, respectively. Further,

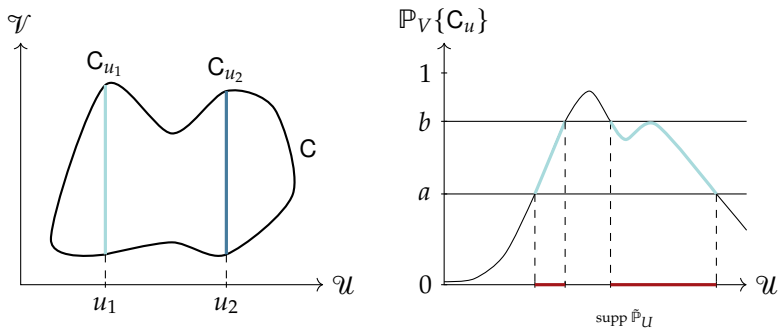

Figure 3.1: Construction of $\tilde{\mathbb{P}}_U$: On the left, the set $\mathsf{C} \subset \mathcal{U} \times \mathcal{V}$ and two of its sections $\mathsf{C}_{u_1}, \mathsf{C}_{u_2} \subset \mathcal{V}$ are visualized. On the right, the function $\mathbb{p}(u) = \mathbb{P}_V\{\mathsf{C}_u\}$, the interval $[a, b]$, and the resulting support $\operatorname{supp} \tilde{\mathbb{P}}_U$ are visualized. Note that, contrary to the visualization here, $\mathbb{p}$ can actually be *highly discontinuous*.

we consider a measurable set $\mathsf{C} \subset \mathcal{U} \times \mathcal{V}$, and we want to generate samples $U = u \in \mathcal{U}$, such that the probability of (U, V) lying in C, given $U = u$, takes values in a certain interval:

$$\mathbb{P}_{(U,V)|U=u}\{\mathsf{C}\} = \mathbb{P}_{V|U=u}\{\mathsf{C}_u\} \in [a, b] \subset [0, 1] .$$

This allows us to define the (measurable) function $\mathbb{p} : \mathcal{U} \to [0, 1]$, $u \mapsto \mathbb{P}_{V|U=u}\{\mathsf{C}_u\}$. By independence of U and V, this is $\mathbb{P}_V$-almost surely the same as $\mathbb{p}(u) = \mathbb{P}_V\{\mathsf{C}_u\} \in [a, b]$, and we will use the later formulation[2] from now on. Thus, for $a, b \in [0, 1]$ with $a < b$, we can define a measurable set $\mathsf{A} := \left\{u \in \mathcal{U} : \mathbb{P}_V\{\mathsf{C}_u\} \in [a, b]\right\}$, which yields a new measure $\tilde{\mathbb{P}}_U$ on $\mathcal{U}$ by restricting to A, that is, for a measurable set $\mathsf{B} \subset \mathcal{U}$ it holds:

$$\tilde{\mathbb{P}}_U\{\mathsf{B}\} := \left((\mathbb{1}_{[a,b]} \circ \mathbb{p}) \cdot \mathbb{P}_U\right)\{\mathsf{B}\} = (\mathbb{1}_{\mathsf{A}} \cdot \mathbb{P}_U)\{\mathsf{B}\} = \mathbb{P}_U\{\mathsf{A} \cap \mathsf{B}\} .$$

Therefore, the goal is to sample from $\tilde{\mathbb{P}}_U$, which equivalently can be stated as:

$$\text{Goal: Get } U_1, \ldots, U_K \sim \mathbb{P}_U, \text{ such that } \mathbb{P}_V\{\mathsf{C}_u\}|_{u=U_i} \in [a, b].$$

This construction is depicted in Figure 3.1: The left figure visualizes the sections $\{\mathsf{C}_u\}_{u \in \mathcal{U}}$ of the set C, while the right figure shows the corresponding construction of the support of $\tilde{\mathbb{P}}_U$. In the following, we implicitly assume that the imposed constraint is realizable, that is, $\tilde{\mathbb{P}}_U$ has a non-empty support.

Example 3.4.1 Consider the random variables P and H from Section 3.3. By Lemma 3.4.1 we want to have $\mathbb{p}(H) \in [\varepsilon_{\text{conv}}, 1]$, where the sublevel probability is given as $\mathbb{p}(h) = \mathbb{P}_P\{\mathsf{L}_{\sigma,h}\}$ (Lemma 3.3.5), and the sublevel set $\mathsf{L}_\sigma \subset \mathcal{H} \times \mathcal{P}$ is measurable by Lemma 3.3.4. Thus, this corresponds to the identification $\mathcal{U} = \mathcal{H}$, $\mathcal{V} = \mathcal{P}$, and $a = \varepsilon_{\text{conv}}$, $b = 1$.

Incorporation into a Sampling Procedure

The only distinction between samples from $\tilde{\mathbb{P}}_U$ and samples from $\mathbb{P}_U$ is the restriction to A. Since many sampling algorithms access the unnormalized density anyway, it suffices to be able to sample from $\mathbb{P}_U$, if the restriction to A can be satisfied differently. Thus, we have to integrate this constraint into a sampling procedure for $\mathbb{P}_U$. Because we do not have any geometrical or topological information about the set C, we resort to statistical information: Given i.i.d. samples $V_1, \ldots, V_n \sim \mathbb{P}_V$, for a given $u \in \mathcal{U}$ we are able to evaluate the Bernoulli random variables $I_n := \mathbb{1}\{V_n \in \mathsf{C}_u\}$, $n \in \mathbb{N}$. These have the parameter $\mathbb{P}\{I_n = 1\} = \mathbb{P}\{V_n \in \mathsf{C}_u\} = \mathbb{P}_V\{\mathsf{C}_u\} = \mathbb{p}(u)$. Thus,

2: Note that $\mathbb{p}$ is still measurable.

Algorithm 1: Iterative estimation of the probability $\mathbb{p}$

Data: $q_l, q_u, \varepsilon \in [0,1]$.

```
// Initialize with uninformative prior.
```

$\alpha, \beta \leftarrow 1, 1$

// $Q_{\alpha,\beta}$ is the quantile function of Beta(α,β).

while $Q_{\alpha,\beta}(q_u) - Q_{\alpha,\beta}(q_l) \geq \varepsilon$ **do**

  Draw $I \sim \text{Ber}(\mathbb{p})$.

  Set $\alpha \leftarrow \alpha + I$ and $\beta \leftarrow \beta + (1 - I)$.

by estimating $\mathbb{p}(u)$ with an estimator $\hat{\mathbb{p}}(u)$, we approximate the constraint A with $\hat{\mathsf{A}}$:

$$\mathsf{A} = \{u \in \mathcal{U} \,:\, \mathbb{p}(u) \in [a,b]\} \approx \{u \in \mathcal{U} \,:\, \hat{\mathbb{p}}(u) \in [a,b]\} =: \hat{\mathsf{A}}\,.$$

To decide whether a given sample $U_i \sim \mathbb{P}_U$ does lie in A, that is, whether U_i can actually be regarded as a sample from $\tilde{\mathbb{P}}_U$, we resort to a simple accept-reject mechanism as in *Metropolis-Hastings*-type algorithms (Robert et al., 2004). Note that this allows to keep an algorithm *inside* $\hat{\mathsf{A}}$. However, it does not provide a way *into* $\hat{\mathsf{A}}$, let alone A.

We estimate $\mathbb{p}(u)$ in a Bayesian way, as it allows us to balance accuracy against computational complexity through uncertainty-quantification, which we use as a stopping criterion: We place a Beta-prior Beta$(\alpha^{(0)}, \beta^{(0)})$ over the interval $[0,1]$. As we do not have prior knowledge, and the map $u \mapsto \mathbb{p}(u)$ can be discontinuous[3], we use a noninformative prior (Berger, 1985, Ch. 3.3), that is, $\alpha^{(0)} = \beta^{(0)} = 1$. Since the Beta distribution is the conjugate prior for the Bernoulli distribution (Berger, 1985, p.130), that is, the posterior is again a Beta-distribution, after observing a sample I_{k+1} the parameters $\alpha^{(k)}, \beta^{(k)}$ get updated as:

$$\alpha^{(k+1)} = \alpha^{(k)} + I_{k+1}, \qquad \beta^{(k+1)} = \beta^{(k)} + (1 - I_{k+1})\,.$$

This allows us to do the estimation iteratively: We only draw a new sample I_{k+1} as long as $Q^{(k)}(q_u) - Q^{(k)}(q_l) \geq \varepsilon$, where $Q^{(k)}$ denotes the quantile-function of Beta$(\alpha^{(k)}, \beta^{(k)})$, and $q_u, q_l, \varepsilon \in [0,1]$ are parameters that specify the accuracy of the estimation by specifying how much mass (through q_l, q_u) has to be concentrated to which extend (through ε). Finally, one can use the posterior mean $\frac{\alpha^{(k)}}{\alpha^{(k)}+\beta^{(k)}}$ or posterior mode $\frac{\alpha^{(k)}-1}{\alpha^{(k)}+\beta^{(k)}-2}$ (provided $\alpha^{(k)}, \beta^{(k)} > 1$) as point estimate $\hat{\mathbb{p}}_u$. By adjusting q_l, q_u or ε, one can balance between accuracy and computational complexity. However, the number of iterations needed also depends on the true probability: For $\mathbb{p}(u) \approx 0$ or $\mathbb{p}(u) \approx 1$, the uncertainty decreases significantly faster[4] than for $\mathbb{p}(u) \approx 0.5$. This procedure is summarized in Algorithm 1 and depicted in Figure 3.2.

3: Consider learning the step-size parameter $h > 0$ for gradient descent on quadratic functions with largest eigenvalue L: The algorithm converges for $h < \frac{2}{L}$ ($\mathbb{p}(h) = 1$) and diverges for $h > \frac{2}{L}$ ($\mathbb{p}(h) = 0$) .

4: For example, the variance of a Beta-distributed random variable with parameters α and β is given by $\frac{\alpha\beta}{(\alpha+\beta+1)(\alpha+\beta)^2}$. Thus, after N draws the variance can be calculated as $\frac{\alpha^{(N)}\beta^{(N)}}{(N+3)(N+2)^2}$. Therefore, if $\alpha^{(N)} = 1$, we have $\beta^{(N)} = N+1$, such that we get $\frac{\alpha^{(N)}\beta^{(N)}}{(N+3)(N+2)^2} = \frac{N+1}{(N+3)(N+2)^2} \approx \frac{1}{N^2}$. Conversely, if $\alpha^{(N)} = \beta^{(N)} = \frac{N}{2} + 1$, we get $\frac{\alpha^{(N)}\beta^{(N)}}{(N+3)(N+2)^2} = \frac{(\frac{N}{2}+1)^2}{(N+3)(N+2)^2} \approx \frac{1}{N}$.

Broader Context

Different, yet conceptually similar ideas for how to cut the computational cost of Bayesian Markov-Chain-Monte-Carlo algorithms through subsampling have been proposed: Korattikara et al. (2014) use sequential hypothesis tests to reach the binary accept-reject decision in the Metropolis-Hastings algorithm. Bardenet et al. (2014) estimate the accept-reject step in such a way that it coincides with the true accept-reject step with a user-specified probability. Maclaurin et al. (2014) introduce an auxiliary binary variable $z_n \in \{0,1\}$, which allows for querying only

Algorithm 2: Probabilistically constrained sampling

Data: $a, b \in [0,1]$, $n_{\max} \in \mathbb{N}$, $u_0 \in \hat{\mathsf{A}}$.

$u \leftarrow u_0$
for $i = 1, \dots, n_{\max}$ **do**
  1) Draw a proposal u' with SGLD starting from u.
  2) Estimate $\mathbb{p}(u') = \mathbb{P}_V\{\mathsf{C}_{u'}\}$ by $\hat{\mathbb{p}}(u')$ with Algorithm 1.
  if $\hat{\mathbb{p}}(u') \in [a, b]$ **then**
    $u \leftarrow u'$
  else
    Reject u'.

a subset of the data for the computation of the exact likelihood. And Quiroz et al. (2019) combine subsampling with a bias-correction strategy to speed-up the sampling procedure. A summary of different approaches is given by Bardenet et al. (2017).

Choice of the Sampling Procedure

Often, the hyperparameters $h \in \mathcal{H}$ are high-dimensional, for example, if the algorithm is modeled by a neural network. Thus, we use *stochastic gradient Langevin dynamics* (SGLD; see Welling et al., 2011) as the underlying sampling algorithm, and constrain it to the set $\hat{\mathsf{A}}$ by use of the previously described procedure. This is summarized in Algorithm 2. However, if it fits the application, other sampling algorithms can be used, too. The computational overhead of the additional estimation depends on the cost of evaluating $\mathbb{1}\{V_n \in \mathsf{C}_u\}$. In our case it is expensive: Every sample I_n requires to run the algorithm $\mathcal{A}$, which corresponds to approximating the solution of a minimization problem.

Remark 3.4.1 Algorithm 2 requires to start in the set $\hat{\mathsf{A}}$. If such a point is not known, one can still run the algorithm and just "start" the accept-reject mechanism as soon as one has found a point $u \in \hat{\mathsf{A}}$. However, it is not guaranteed that such a point will actually be found.

The results of applying this procedure for a two-dimensional toy example are shown in Figure 3.3: The upper row shows the function $u \mapsto \mathbb{p}(u)$, and the potential from which we want to sample with the constraint $\mathbb{p}(u) \in [0.6, 1]$. The lower row shows the accepted (black) and rejected (gray) samples, and the final estimate of the constrained potential. While most samples get accepted/rejected correctly, some are actually false-positives (dark red) or false-negatives (red). Yet, this is to be expected. Note that, for simplicity, we did use full gradients here.

We are now in a position to describe the whole learning procedure.

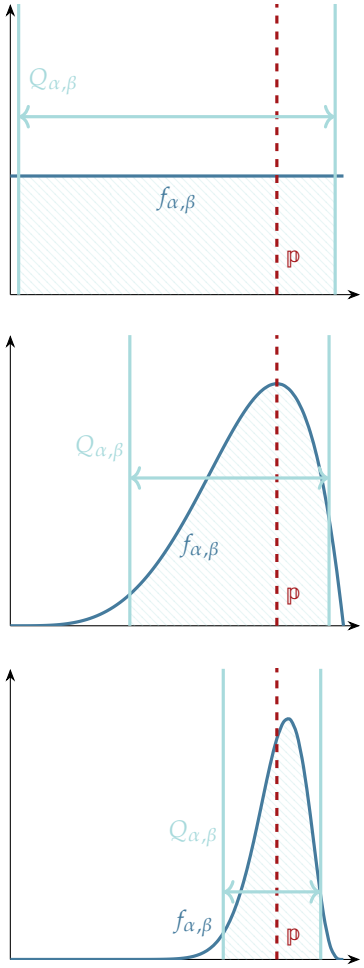

Figure 3.2: Estimation of $\mathbb{p}$ with a Beta-prior.

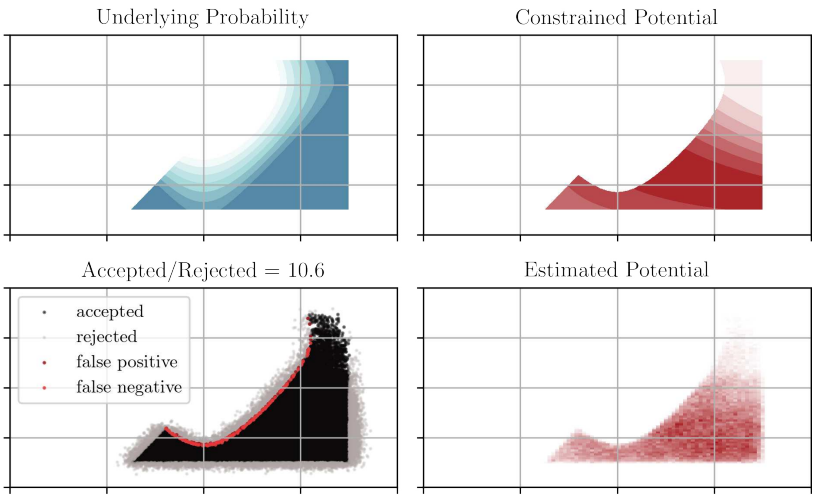

Figure 3.3: Probabilistically constrained sampling: The upper left plot shows the underlying function $\mathbb{p}(u)$. It is discontinuous and defines a non-convex set A. The upper right plot shows the probabilistically constrained potential ($\mathbb{p}(u) \in [0.6, 1]$), from which we want to sample. The lower left plot shows the accepted (black) and the rejected (gray) samples, which appear in a ratio of about 10:1. Further, we can see that some of them are false-positives (dark red) or false-negatives (red). Especially, this happens for $\mathbb{p}(u) \approx 0.6$, where the remaining uncertainty can easily lead to a wrong decision. Here, we have chosen $q_l = 0.01$, $q_u = 0.99$, and $\varepsilon = 0.05$ in Algorithm 1. Finally, the lower right plot shows the estimated potential.

3.5 Learning Procedure

This section deals with the implementation of the learning procedure, and translates the abstract framework discussed in Sections 3.2 and 3.3 into concrete design choices. Thus, this marks the beginning of the second part of this chapter, which is less theoretical. The resulting learning procedure is visualized in Figure 3.4 and consists of four steps:

(i) **Step one:** Train the algorithm to "mimic" another algorithm $\mathcal{A}'$. This is needed only, if one cannot choose stable initial hyperparameters directly, for example, when the update includes a neural network. Otherwise, the algorithm might predict points that are so far off that one encounters numerical instabilities.

(ii) **Step two:** Find a point $h^{(0)} \in \mathcal{H}$ that a) satisfies the constraint in Subsection 3.3.2 and b) yields a good performance. For this, we perform a constrained version of stochastic empirical risk minimization with a new, specifically designed loss function.

(iii) **Step three:** Starting from $h^{(0)}$, construct the prior distribution by running a constrained version of a sampling algorithm.

(iv) **Step four:** Find the optimal $\lambda^* \in \Lambda$, which allows for computing the optimal posterior distribution $\mathbb{Q}_{\lambda^*}$ in closed-form.

The outline of this section is as follows: In Subsection 3.5.1 we identify the optimal posterior $\mathbb{Q}_{\lambda^*}$ in the abstract setting. In Subsection 3.5.2, we describe the pre-computation phase in (i). Subsections 3.5.3 and 3.5.4 deal with the concrete design choices in (ii) and (iii) to construct the prior, and Subsection 3.5.5 yields the posterior distribution in (iv). Since the prior has to be independent of the data set that is used in the PAC-Bayesian step, we split the data set S into independent parts S_{prior}, S_{val}, S_{train} and S_{test}, where S_{prior} and S_{val} are used for the construction of the prior distribution, S_{train} is used for the PAC-Bayesian learning step, and S_{test} is the test set which is only needed for the experiments. Nevertheless, for notational simplicity, we will still use the generic S, implicitly assuming the above partitioning.

Remark 3.5.1 Through the choice of the sampling algorithm, the concrete learning procedure described here mainly applies to the case $\mathcal{H} = \mathbb{R}^d$, $d \in \mathbb{N}$. Nevertheless, the general methodology is still applicable to other Polish spaces, if this choice can be adjusted accordingly.

3.5.1 Minimization of the PAC-Bayesian Bound

Learning is phrased as minimizing the PAC-Bayesian upper-bound. Hence, in this subsection we consider η, τ and Equation (B) from Corollary 3.2.4, and we seek for $\lambda \in \Lambda$ and $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ that minimize the upper-bound, that is, we want to solve:

$$\inf_{\lambda\in\Lambda} \inf_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)} \left(\mathbb{Q}\{\hat{\mathcal{R}}(\cdot,s)\} + \frac{1}{\eta^{(1)}(\lambda)} \left(D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{\mathcal{K}}{\varepsilon}\right)\right) + \frac{1}{\eta^{(1)}(\lambda)} \left(\mathcal{C}_{\mathcal{O}} - \mathbb{Q}\{\langle \eta^{(r)}(\lambda), \tau^{(r)}(\cdot,s)\rangle\}\right)\right).$$

By factoring out $-\frac{1}{\eta^{(1)}(\lambda)}$ again, this is actually the same as:

$$\inf_{\lambda\in\Lambda} -\frac{\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)} \mathbb{Q}\{\langle \eta(\lambda), \tilde{\tau}(\cdot,s)\rangle\} - D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) - \log\left(\frac{\mathcal{K}}{\varepsilon}\right) - \mathcal{C}_{\mathcal{O}}}{\eta^{(1)}(\lambda)},$$

where $\tilde{\tau}(h,s) := \left(-\hat{\mathcal{R}}(h,s),\ \tau^{(r)}(h,s)\right)$. Since $\log(\mathcal{K}/\varepsilon) + \mathcal{C}_{\mathcal{O}}$ is a constant, Lemma 3.2.1 shows that the numerator is given by $\tilde{\kappa}(\lambda,s) - \log(K/\varepsilon) - \mathcal{C}_{\mathcal{O}}$, where $\tilde{\kappa}$ corresponds to the exponential family $(\tilde{\mathbb{Q}}_\lambda)_{\lambda\in\Lambda}$ built upon $\tilde{\tau}$ and η (with $b \equiv 1$). Furthermore, the optimal posterior distribution $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ is given by the corresponding member of the data-dependent exponential family $\tilde{\mathbb{Q}}_\lambda(s) \propto \exp(\langle \eta(\lambda), \tilde{\tau}(\cdot,s)\rangle)\cdot\mathbb{P}_H$, usually called the Gibbs posterior (Alquier, 2024). By denoting $F(\lambda,s) := -\frac{1}{\eta^{(1)}(\lambda)}(\tilde{\kappa}(\lambda,s) - \log(\mathcal{K}/\varepsilon) - \mathcal{C}_{\mathcal{O}})$, one is left with solving the following problem:

$$\inf_{\lambda\in\Lambda} F(\lambda,s), \tag{3.3}$$

which for $\Lambda \subset \mathbb{R}$ is one-dimensional. Based on Theorem 3.3.6, we restrict to $\Lambda \subset (0,+\infty)$, such that the solution to (3.3) can be seen as an approximation to the global minimum $\inf_{\lambda>0} F(\lambda,s)$. For the latter one, it can be shown that the solution set lies in a compact interval $[\Lambda_{\min}, \Lambda_{\max}]$, since $F(\lambda,s) \to +\infty$ as $\lambda \to 0$ or $\lambda \to +\infty$. Under our assumptions, $F(\cdot,s)$ is continuously differentiable. Hence, since Λ is compact, $F(\cdot,s)$ is Lipschitz-continuous on Λ and the minimum in (3.3) is attained. For a finite set $\Lambda = \{\lambda_1,\dots,\lambda_K\} \subset [\Lambda_{\min}, \Lambda_{\max}]$, the optimization reduces to grid search. For $\Lambda = [\Lambda_{\min}, \Lambda_{\max}]$, we employ grid search as initialization for gradient-based optimization. Here, the computational bottleneck is given by evaluating $\lambda \mapsto \tilde{\kappa}(\lambda,s)$. In Sections 3.5.4 and 3.5.5 we will ensure that this is cheap.

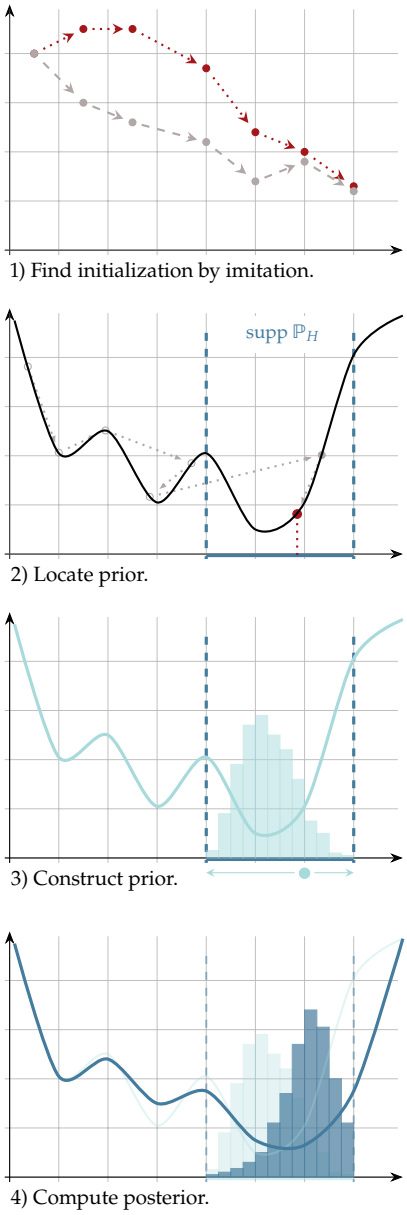

Figure 3.4: Visualization of the four main steps of the learning procedure.

3.5.2 Finding a Trainable Initialization

To increase numerical stability, we start with "imitation learning" (T. Chen et al., 2020), that is, the algorithm $\mathcal{A}$ should "follow" another algorithm $\mathcal{A}'$, for example, gradient descent. For this, we minimize the mean squared error between the iterates of the two algorithms: Given a starting point $x^{(0)} \in \mathbb{R}^n$, an integer $t_{\mathrm{len}} \in \mathbb{N}$ specifying the number of iterations, and a parameter $p \in \mathcal{P}$, denote the first t_{len} iterates of $\mathcal{A}(h,p,x^{(0)})$ by $x^{(1)},\dots,x^{(t_{\mathrm{len}})} \in \mathbb{R}^n$ and the ones of $\mathcal{A}'(x^{(0)},p)$ by $y^{(1)},\dots,y^{(t_{\mathrm{len}})} \in \mathbb{R}^n$. Then, define the loss as the mean squared error over these iterations:

$$\ell_{\mathrm{init}}(h,p,x^{(0)},t_{\mathrm{len}}) := \frac{1}{t_{\mathrm{len}}} \sum_{k=1}^{t_{\mathrm{len}}} \|x^{(k)} - y^{(k)}\|_2^2.$$

Algorithm 3: Procedure to find an initialization

Data: Data set s_{prior}, $x^{(0)} \in \mathbb{R}^n$, $t_{\text{len}}, n_{\text{init}} \in \mathbb{N}$ and $\varepsilon > 0$.

Set $m \leftarrow +\infty$ and sample $p \sim \mathrm{U}_{s_{\text{prior}}}$.

while $\frac{1}{n_{\text{init}}} m > \varepsilon$ **do**

  $m \leftarrow 0$

  for $i = 1, \ldots, n_{\text{init}}$ **do**

    1) Compute $(x^{(1)}, y^{(1)}), \ldots, (x^{(t_{\text{len}})}, y^{(t_{\text{len}})})$ with $\mathcal{A}(h, p, x^{(0)})$ and $\mathcal{A}'(p, x^{(0)})$, resp.

    2) Compute $\ell_{\text{init}}(h, p, x^{(0)}, t_{\text{len}}) = \frac{1}{t_{\text{len}}} \sum_{k=1}^{t_{\text{len}}} \|x^{(k)} - y^{(k)}\|_2^2$.

    3) Update $m \leftarrow m + \ell_{\text{init}}(h, p, x^{(0)}, t_{\text{len}})$.

    // Also other algorithms than Adam are possible.

    4) Update h by backpropagation and Adam.

    5) Update p, $x^{(0)}$ and t_{len} based on Section 3.5.3.

In each iteration, that is, for each prediction of tuples $(x^{(1)}, y^{(1)}), \ldots, (x^{(t_{\text{len}})}, y^{(t_{\text{len}})})$, the parameters p, $x^{(0)}$ and t_{len} are randomized as described in Section 3.5.3. Please note that it is not necessary to reach a high accuracy here, as the purpose is to prevent divergence and not actual imitation of $\mathcal{A}'$. The procedure is summarized in Algorithm 3.

3.5.3 Locating the Prior

Empirically, the performance of the learned algorithm is significantly improved by the following two design choices. The motivation is to prevent overfitting and to learn a scale-independent contraction of the loss:

Ratio of Losses

The canonical loss function to be minimized is the empirical risk $\hat{\mathfrak{R}}(h, s) = \frac{1}{N} \sum_{i=1}^{N} \ell(h, p_i)$, and, if $\mathcal{H}$ is high-dimensional or if N is large, one resorts to stochastic empirical risk minimization. While this kind of loss is used extensively in machine learning, for learning-to-optimize it has a strong disadvantage: Only the overall outcome after t_{train} iterations gets penalized. Thus, it does not take the performance along the trajectory into account. Further, often it is hard to minimize (due to training instabilities) and does not lead to the desired performance. To circumvent this, Andrychowicz et al. (2016) proposed to use $\tilde{\ell}_{\text{train}}(h, p, x^{(0)}) := \sum_{i=1}^{t_{\text{train}}} \ell(x^{(i)}, p)$. However, also this formulation has a decisive flaw: Under most objectives, if the algorithm performs reasonably well, the loss at the beginning is *several orders* of magnitude larger than the loss at the end. Hence, $\tilde{\ell}_{\text{train}}$ mainly penalizes the loss at the beginning, leading to an algorithm that minimizes the loss very fast in early iterations, yet slows down a lot in later iterations. This is due to $\tilde{\ell}_{\text{train}}$ being *scale-sensitive*. Additionally, the incurred loss might vary strongly with the initialization $x^{(0)}$ alone, thereby introducing ambiguity into the incurred losses. We propose to use the *ratio of consecutive losses*:

$$\ell_{\text{train}}(h, p, x^{(0)}, t_{\text{len}}) := \sum_{i=1}^{t_{\text{len}}} \mathbb{1}_{\{\ell(x^{(i-1)}, p) > 0\}} \frac{\ell(x^{(i)}, p)}{\ell(x^{(i-1)}, p)},$$

Algorithm 4: Procedure to locate the prior

Data: Data sets $s_{\text{prior}}, s_{\text{val}}$, numbers $n_{\max}, t_{\text{train}}, t_{\text{len}} \in \mathbb{N}$ with $t_{\text{len}} \leq t_{\text{train}}$, initialization $x^{(0)}$ and thresholds $a, b \in [0,1]$ with $a < b$.

Set $x \leftarrow x^{(0)}$ and sample $p \sim \mathrm{U}_{s_{\text{prior}}}$.
$b \leftarrow$ **False** `// b = Point inside constraint?`
for $i = 1, \ldots, n_{\max}$ **do** `// Other stopping criteria possible.`

  `// Compute a new proposal for the hyperparameters.`
  1.a) Compute $x^{(1)}, \ldots, x^{(t_{\text{len}})}$ with $\mathcal{A}(h, p, x^{(0)})$.
  1.b) Compute $\ell_{\text{train}}(h, p, x^{(0)}, t_{\text{len}}) = \sum_{k=1}^{t_{\text{len}}} \mathbb{1}_{\{\ell(x^{(k-1)},p)>0\}} \frac{\ell(x^{(k)},p)}{\ell(x^{(k-1)},p)}$.
  1.c) Construct a proposal $\tilde{h}$ by using backpropagation and Adam.

  `// Check the probabilistic constraints.`
  2) Estimate $\mathbb{p}(\tilde{h})$ by $\hat{\mathbb{p}}(\tilde{h})$ with Algorithm 1 on s_{val}.
  if $\hat{\mathbb{p}}(\tilde{h}) \in [a, b]$ **then** `// If point inside constraint, update.`
    $h \leftarrow \tilde{h}$
    $b \leftarrow$ **True**
  else `// If not...`
    if $b =$ **True then** `// ...reject moving outside constraint.`
      Reject $\tilde{h}$.
      Set $x^{(0)} \leftarrow x$.
      Sample $p \sim \mathrm{U}_{s_{\text{prior}}}$.
      Continue with 1).
    else `// ...accept, if constraint has not been found yet.`
      $h \leftarrow \tilde{h}$

  `// Randomize the trajectory.`
  3) Draw $R \sim \mathrm{Ber}(\frac{t_{\text{len}}}{t_{\text{train}}})$.
  if $R = 0$ **then**
    $x^{(0)} \leftarrow x^{(t_{\text{len}})}$
  else
    $x^{(0)} \leftarrow x$
    Sample $p \sim \mathrm{U}_{s_{\text{prior}}}$.

where $t_{\text{len}} \in \mathbb{N}$ with $t_{\text{len}} \leq t_{\text{train}}$. This has several advantages: First, the loss is not scale-sensitive anymore, such that it favors hyperparameters that yield a good performance in each iteration. Second, there is no ambiguity in the observed loss through the initialization, as the only criterion is a strong contraction of the loss (instead of a small loss). Third, the incurred losses do not vary too much, which empirically makes it easier to choose hyperparameters of the learning procedure. However, it also has a disadvantage: If the function values do indeed converge in a setting where the optimal loss is strictly greater than zero, this gets fully penalized, as then $\frac{\ell(x^{(i)},p)}{\ell(x^{(i-1)},p)} \equiv 1$. For now, we do not know how to avoid this problem (apart from just stopping the iterations in case of convergence) while keeping the advantages.

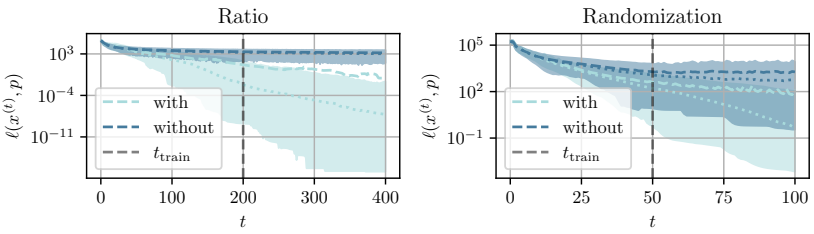

Figure 3.5: Effect of our design choices: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded region represents 95% of the data. The light blue algorithm was trained with our design choice and the dark blue one without. Besides that, everything else was kept the same. In the left plot we can see that using the ratio of consecutive losses strongly improves the performance, and in the right plot we can see that the randomization procedure yields generalization beyond t_{train} *and* an overall better performance.

Randomized Trajectory Length

Training $\mathcal{A}$ with fixed initialization $x^{(0)}$ and fixed trajectory length leads to overfitting, that is, starting the algorithm at another point $\tilde{x}^{(0)}$ or applying it for more iterations typically does not work, or even leads to divergence. To avoid this, we propose the following randomization: Fix $t_{\text{len}} \leq t_{\text{train}}$ and set $y := x^{(0)}$.

0) Sample a parameter p uniformly at random from s.
1) Compute $x^{(1)}, \ldots, x^{(t_{\text{len}})}$ with $\mathcal{A}(h, p, y)$ and the loss $\ell_{\text{train}}(h, p, y, t_{\text{len}})$, and update h.
2) Sample $R^{(k)} \sim \text{Ber}(\frac{t_{\text{len}}}{t_{\text{train}}})$. If $R^{(k)} = 0$, set $y := x^{(t_{\text{len}})}$ and go to step 1). If $R^{(k)} = 1$, set $y := x^{(0)}$ and go to step 0).

The random variable $R^{(k)}$ decides whether the algorithm gets restarted from $x^{(0)}$ with a new parameter $\tilde{p}$, or if one continues the current trajectory. The choice $\frac{t_{\text{len}}}{t_{\text{train}}}$ ensures that the expected trajectory-length equals t_{train}: Define $T := \inf\{k \in \mathbb{N} \,:\, R^{(k)} = 1\}$. Then, $T \sim \text{Geo}(\frac{t_{\text{len}}}{t_{\text{train}}})$ is geometrically distributed with expectation $\mathbb{E}\{T\} = \frac{t_{\text{train}}}{t_{\text{len}}}$. Therefore, for the actual length $L = t_{\text{len}} \cdot T$ of the trajectory we get $\mathbb{E}\{L\} = t_{\text{len}}\mathbb{E}\{T\} = t_{\text{train}}$.

Remark 3.5.2 Similarly to Andrychowicz et al. (2016), we omit the computation of second-order derivatives during training. Additionally, and surprisingly, it usually suffices to consider single iterates, that is $t_{\text{len}} = 1$. This amounts to learning an update step that is agnostic to the recurrent nature of the optimization algorithm and just learns to adapt to the local geometry of the loss function *along the iterations*.

Figure 3.5 shows the effect of these two design choices: The left plot shows the effect of using the ratio of consecutive losses and the right plot shows the effect of randomizing the trajectory. In both cases, we train two times the same algorithm: One time with our proposed choice (light blue), and one time without (dark blue). Everything else is kept the same, that is, both were trained with Algorithm 4. In the left plot one can see that the ratio of losses strongly improves the performance compared to using normal function-values, and in the right plot one can see that the randomization procedure improves the generalization to more iterations *and* its performance. However, please note that there might be some bias: The architecture of the algorithm is one that we have found using our proposed training procedure. Further details can be found in the GitHub-repository (see Appendix A). The overall procedure is summarized in Algorithm 4.

Algorithm 5: Procedure to construct the prior

Data: Data sets s_{prior} (sampling) and s_{val} (constraint), $n_{\text{sam}} \in \mathbb{N}$ and $h \in \operatorname{supp} \tilde{\mathbb{P}}_h$.

// Starting from h, construct samples inside the constraint.

1 1) Run Algorithm 2 (with ℓ_{train}) to get the points $h_1, \ldots, h_{n_{\text{sam}}} \in \mathcal{H}$.

// Evaluate potential on s_{prior} (compute $\hat{\mathcal{R}}_{\sigma,\text{prior}}$).

2 2) Compute $\varphi_{\text{prior}}(h_1), \ldots, \varphi_{\text{prior}}(h_{n_{\text{sam}}})$.

// Prior is given by evaluating the softmax-function.

3 3) Evaluate $\mathbb{P}_H = \sigma\left(\varphi_{\text{prior}}(h_1), \ldots, \varphi_{\text{prior}}(h_{n_{\text{sam}}})\right)$.

3.5.4 Constructing the Prior

Besides the performance and the sublevel guarantees, the only assumption on the prior $\mathbb{P}_H$ is its independence of S_{train}. Further, the functional form of the posterior is fully specified by Lemma 3.2.1, namely it is of the form:

$$\mathbb{Q}_\lambda(s) \propto \exp(\varphi_\lambda(\cdot, s)) \cdot \mathbb{P}_H, \qquad \lambda \in \Lambda, \tag{3.4}$$

where the potential is given by $\varphi_\lambda(h, s) = \langle \eta(\lambda), \tilde{\tau}(h, s) \rangle$. Hence, for mathematical convenience, we will construct $\mathbb{P}_H$ by approximating the distribution $\mathbb{P}'$ given by

$$\mathbb{P}' \propto \exp\left(-\hat{\mathcal{R}}_{\sigma,\text{prior}} - \iota_{[a,b]} \circ \mathbb{p}\right) \cdot \mu,$$

where μ is a measure on $\mathcal{H}$, which allows to sample from $\mathbb{P}'$ (possibly unnormalized). In our experiments it holds $\mathcal{H} = \mathbb{R}^d$ and we choose $\mu = \lambda^d$, where λ^d is the d-dimensional Lebesgue measure. For sampling, we use the *stochastic gradient Langevin dynamics* algorithm, where we use the output of the backpropagation algorithm as proxy for the (sub)gradient. Finally, since anyway we have to resort to a sampling algorithm to get points $h_1, \ldots, h_{n_{\text{sam}}} \in \mathcal{H}$, $n_{\text{sam}} \in \mathbb{N}$, we define the prior distribution directly as a discrete distribution, that is $\mathbb{P}_H\{h\} := \frac{1}{Z} \sum_{i=1}^{n_{\text{sam}}} w_i \delta_{h_i}\{h\}$. Thus, $\mathbb{P}_H$ is the suitably normalized discrete measure on $\mathcal{H}$ corresponding to $h_1, \ldots, h_{n_{\text{sam}}}$, where the normalization constant is given by $Z = \sum_{i=1}^{n_{\text{sam}}} w_i$ with $w_i = \exp\left(-\hat{\mathcal{R}}_{\sigma,\text{prior}}(h_i) - \iota_{[a,b]}\left(\hat{\mathbb{p}}(h_i)\right)\right)$. When $h_1, \ldots, h_{n_{\text{sam}}} \in \mathcal{H}$ are given, the corresponding probabilities can equivalently be expressed with the so-called *softmax* function $\sigma(x_1, \ldots, x_n)_j = \frac{\exp(x_j)}{\sum_{i=1}^n \exp(x_i)}$ and the potential $\varphi_{\text{prior}}(h) = -\hat{\mathcal{R}}_{\sigma,\text{prior}}(h) - \iota_{[a,b]}\left(\hat{\mathbb{p}}(h)\right)$:

$$\mathbb{P}_H\{h_j\} = \frac{\exp\left(\varphi_{\text{prior}}(h_j)\right)}{\sum_{i=1}^{n_{\text{sam}}} \exp\left(\varphi_{\text{prior}}(h_i)\right)} = \sigma\left(\varphi_{\text{prior}}(h_1), \ldots, \varphi_{\text{prior}}(h_{n_{\text{sam}}})\right)_j .$$

Here, the potentials φ_{prior} have to be computed *only once* for every h_i, $i = 1, \ldots, n_{\text{sam}}$. This is summarized in Algorithm 5.

Remark 3.5.3 As one would approximate the intractable integrals with Monte-Carlo estimates anyway, choosing a discrete measure is less restrictive than it seems, and it has the additional advantage of allowing for *exact* instead of approximate inference.

Algorithm 6: Procedure to construct the posterior

Data: Points $\{h_1, ..., h_{n_{\text{sam}}}\}$, values $\{\varphi_{\text{prior}}(h_1), ..., \varphi_{\text{prior}}(h_{n_{\text{sam}}})\}$, data set $s = s_{\text{train}}$.

1 1) Compute $\tilde{T}(h_1, s), ..., \tilde{T}(h_{n_{\text{sam}}}, s)$.
2 2) Setup $\{\varphi_\lambda(h_1, s), ..., \varphi_\lambda(h_{n_{\text{sam}}}, s)\}$ as functions in λ.
3 3) Solve $\lambda^* \in \arg\min_{\lambda \in \Lambda} F(\lambda, s)$. `// F(λ*, s) is the PAC-bound.`
`// Posterior is given by evaluating the softmax-function.`
4 4) Evaluate $\mathbb{Q}_{\lambda^*}(s) = \sigma\left(\varphi_{\lambda^*}(h_1, s), ..., \varphi_{\lambda^*}(h_{n_{\text{sam}}}, s)\right)$.
`// Optional`
5 5) Choose $h^* = \arg\max_{i=1,...,n_{\text{sam}}} \mathbb{Q}_{\lambda^*}(s)\{h_i\}$ as final point-estimate.

3.5.5 Computing the Posterior

Given a discrete prior $\mathbb{P}_H$, every posterior $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ is also discrete with the same support $\{h_1, ..., h_{n_{\text{sam}}}\}$. Then, by the closed-form solution (3.4), for every $\lambda \in \Lambda$ the optimal posterior $\mathbb{Q}_\lambda(s)$ is given by:

$$\begin{aligned}\mathbb{Q}_\lambda(s)\{h_j\} &= \frac{\exp\left(\langle \eta(\lambda), \tilde{\tau}(h_j, s)\rangle + \varphi_{\text{prior}}(h_j)\right)}{\sum_{i=1}^{n_{\text{sam}}} \exp\left(\langle \eta(\lambda), \tilde{\tau}(h_i, s)\rangle + \varphi_{\text{prior}}(h_j)\right)} \\ &= \sigma\left(\varphi_\lambda(h_1, s), ..., \varphi_\lambda(h_{n_{\text{sam}}}, s)\right)_j ,\end{aligned}$$

with the potential $\varphi_\lambda(h, s) = \langle \eta(\lambda), \tilde{\tau}(h, s)\rangle + \varphi_{\text{prior}}(h)$. Thus, to get the distribution $\mathbb{Q}_\lambda(s)$ as a function of λ, one has to compute $\tilde{\tau}(h_i, s)$ *only once* for every $i = 1, ..., n_{\text{sam}}$, such that it can be evaluated with the softmax function. Hence, the only missing ingredient is the optimal $\lambda^* \in \Lambda$, which is found as described in Section 3.5.1. After evaluating the potentials $\varphi_\lambda(\cdot, s)$, which has to be done anyway, evaluating $\tilde{\kappa}(\cdot, s)$ in λ is cheap. The process is summarized in Algorithm 6.

3.6 Experiments

We consider the smooth and strongly convex problem of minimizing quadratic functions with varying strong convexity and smoothness constants, a high-dimensional image processing problem (convex and smooth), the non-smooth LASSO problem, and the non-smooth and non-convex problem of training a neural network. More details on the implementation, especially how we construct the parameters for each problem, are given in Appendix A. Alternatively, the code can be found in the GitHub-repository.

In the evaluation, we will always show the loss over the iterations in the upper left plot, the performance in terms of computation time in the upper right plot, the loss histogram with predicted PAC-bound in the lower left plot, and the final estimate for the sublevel probability, that is, whether the probabilistically constrained optimization/sampling procedure did work correctly, in the lower right plot. Finally, note that we always show the performance of a single sample h^* (mode of the posterior), and not the mean performance.

As we contrast the learned algorithm to first-order methods, in each iteration $\mathcal{A}$ has access to iterates, (sub)gradients, and function values, and the update is solely based on these. Here, we perform preprocessing: The (sub)gradient is split into its norm $\|\nabla \ell(x^{(t)}, p)\|$ and the corresponding unit vector $d_1^{(t)}$. Further, the norm is transformed to $n_1^{(t)} := \log(1 + \|\nabla \ell(x^{(t)}, p)\|)$ to be less scale-sensitive. The iterates $x^{(t)}, x^{(t-1)}$ are

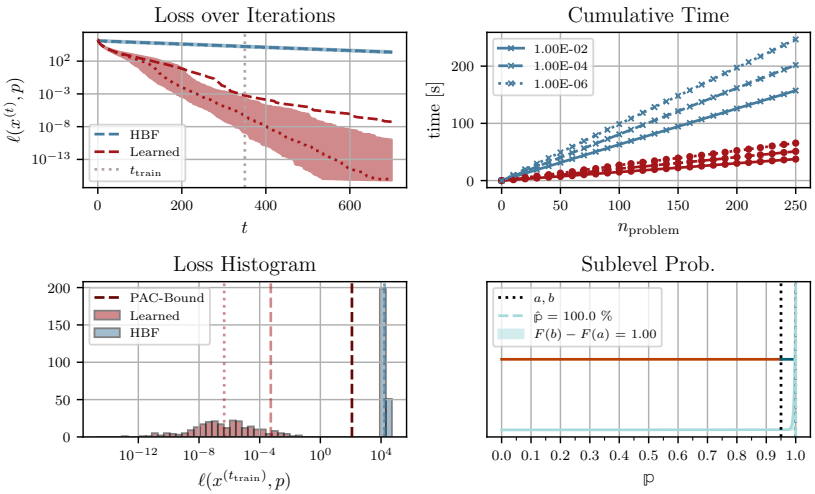

Figure 3.6: Upper left: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions represent the 10th to 90th percentile. The learned algorithm $\mathcal{A}$ is shown in red, while *heavy-ball with friction* (HBF) is shown in blue. **Upper right:** Cumulative computation time to solve all the test problems up to a certain accuracy measured by $\ell(x^{(t)}, p) < \varepsilon$ (both algorithms are run for at most $t_{\text{max}} = 1\text{e}4$ iterations). **Lower left:** Loss histogram after $t_{\text{train}} = 350$ iterations with predicted PAC-bound. **Lower right:** Bayesian estimate for the sublevel probability (Beta-posterior; light blue line) and the imposed constraints a, b (black dotted line).

combined into the momentum term $m^{(t)} := x^{(t)} - x^{(t-1)}$, which also is split into the unit vector $d_2^{(t)}$ and the transformed norm $n_2^{(t)}$. Similarly, we also transform the function values into $\ell_1^{(t)} = \log(1 + \ell(x^{(t)}, p))$ and $\ell_2^{(t)} = \log(1 + \ell(x^{(t-1)}, p))$.

3.6.1 Quadratics

As first problem we consider strongly convex quadratic functions with varying strong convexity, varying smoothness and varying right-hand side, that is, each optimization problem is of the form:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2, \qquad A \in \mathbb{R}^{n \times n},\ b \in \mathbb{R}^n.$$

Thus, the parameters are given by $p = (A, b) \in \mathbb{R}^{n^2+n} =: \mathcal{P}$, while the optimization variable is $x \in \mathbb{R}^n$, where we use $n = 200$. By construction, each of these functions is L-smooth and m-strongly convex, with $L \in [L_-, L_+]$ and $m \in [m_-, m_+]$. Hence, assuming that it is not feasible to compute the smoothness and strong-convexity constants for each problem separately, the given class of functions is L_+-smooth and m_--strongly convex. Therefore, we use *heavy-ball with friction* (HBF) due to Polyak (1964) as baseline. Its update is given by $x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}) + \beta\left(x^{(t)} - x^{(t-1)}\right)$, where the optimal worst-case convergence rate is attained for $\alpha = \left(\frac{2}{\sqrt{L_+}+\sqrt{\mu_-}}\right)^2$ and $\beta = \left(\frac{\sqrt{L_+}-\sqrt{\mu_-}}{\sqrt{L_+}+\sqrt{\mu_-}}\right)^2$ (Nesterov, 2018). On the other hand, the algorithmic update of the learned algorithm $\mathcal{A}$ is visualized in Figure 3.7 and consists of two blocks: The update-block combines the gradient direction $d_1^{(t)}$, the momentum direction $d_2^{(t)}$, and their "interaction" $d_1^{(t)} \odot d_2^{(t)}$ into the new update-direction $d^{(t)}$, while the other block computes a step-size based on the corresponding logarithmically transformed norms $n_1^{(t)}$ and $n_2^{(t)}$, and the logarithmically transformed function values $\ell_1^{(t)}$ and $\ell_2^{(t)}$. Further details can be found in Appendix A.1.
Figure 3.6 shows the results of this experiment: The upper left plot shows that the learned algorithm outperforms HBF by orders of magnitude and, despite being trained for $t_{\text{train}} = 350$ iterations, one can use it until convergence. Here, the mean indicates that there are single instances for which instabilities occur, and, by comparing it to the median, one observes that the mean is far from being representative of the "typical"

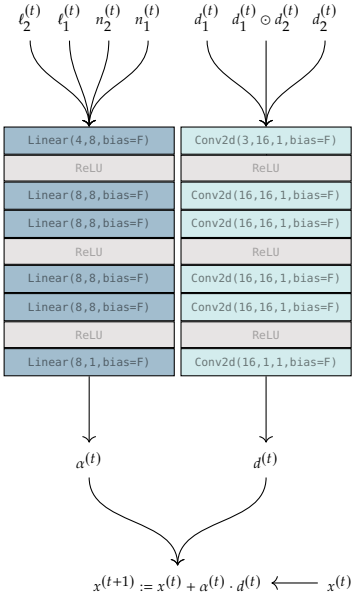

Figure 3.7: Update step of $\mathcal{A}$ for quadratic problems: The directions $d_1^{(t)}$, $d_2^{(t)}$ and $d_1^{(t)} \odot d_2^{(t)}$ are inserted as different channels into the `Conv2d`-block, which performs 1×1 "convolutions", that is, the algorithm acts coordinate-wise on the input, and yields an new update-direction $d^{(t)}$. The scales $n^{(1)}$, $n^{(2)}$, and the function values $\ell_1^{(t)}$, $\ell_2^{(t)}$ get transformed separately by the fully-connected block to yield the step-size $\alpha^{(t)}$.

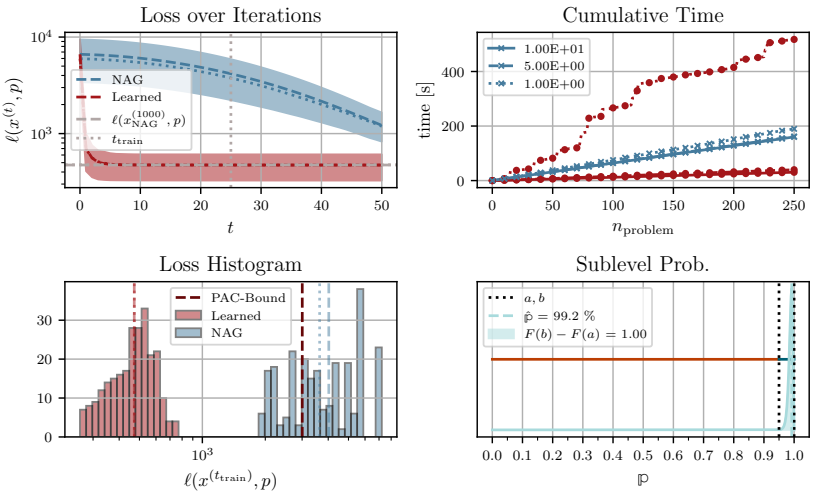

Figure 3.8: Upper left: Dashed lines represent mean losses, dotted lines show median losses, and the shaded regions represent the 10th to 90th percentile. The learned algorithm $\mathcal{A}$ is shown in red, while Nesterov's *accelerated gradient descent* (NAG) is shown in blue. **Upper right:** Cumulative computation time to solve the test problems up to a certain accuracy measured by $\ell(x^{(t)}, p) - \ell(x^{(1000)}_{\text{NAG}}, p) < \varepsilon$ (both algorithms are run for at most $t_{\max} = 1000$ iterations). **Lower left:** Loss histogram after $t_{\text{train}} = 50$ iterations with predicted PAC-bound. **Lower right:** Bayesian estimate for the sublevel probability (Beta-posterior; light blue line) and the imposed constraints a, b (black dotted line).

performance. Further, the algorithm performs well on very different orders of magnitude, ranging from about 1e5 to 1e−15. The upper right plot confirms that also in terms of computation time the learned algorithm is way faster than HBF, and the lower left plot shows that while the predicted PAC-bound is not tight, it still provides the guarantee to outperform HBF. Lastly, the lower right plot shows that the algorithm did satisfy the specified constraints a and b in all test cases.

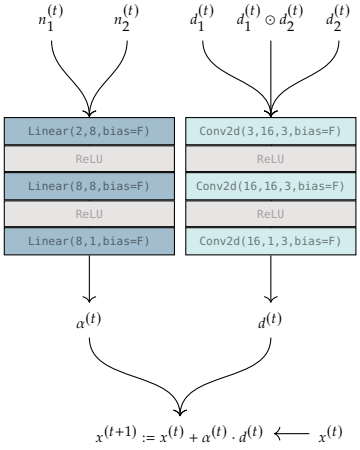

Figure 3.9: Update step of $\mathcal{A}$ for the image-processing experiment: The directions $d_1^{(t)}$, $d_2^{(t)}$ and $d_1^{(t)} \odot d_2^{(t)}$ are inserted as different channels (in the shape of an image) into the `Conv2d`-block, which performs a 3×3-convolution. The scales $n^{(1)}, n^{(2)}$ get transformed separately by the fully-connected block.

3.6.2 Image Processing

We consider (gray-scale) *image denoising/deblurring* with a regularizer given as smooth approximation to the L_1-norm of the image derivative, that is, problems of the form:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|^2 + \lambda \sum_{i,j=1}^{n} \sqrt{(D_h x)_{i,j}^2 + (D_w x)_{i,j}^2 + \varepsilon^2}\,,$$

where $\lambda \in \mathbb{R}$, $A, D_h, D_w \in \mathbb{R}^{n \times n}$, and $b \in \mathbb{R}^n$. The matrix A describes the "blurring" of the image, while D_h and D_w are the discrete image derivatives in h- and w-direction, respectively, which are used to penalize local changes in the image. We use images of height $N_h = 250$ and width $N_w = \texttt{int}(0.75 \cdot N_h) = 187$. Thus, the dimension n of the optimization space is given by $n = 46750$. Further, as parameters p we use the observed image and the regularization parameter, that is, $p = (b, \lambda) \in \mathbb{R}^{n+1} =: \mathcal{P}$. Since the problem is smooth and convex, yet not strongly convex, the baseline algorithm is given by the *accelerated gradient descent* (NAG) algorithm due to Nesterov (1983). Its update is given by first computing $y^{(t+1)} = x^{(t)} + \frac{a^{(t)}-1}{a^{(t+1)}}(x^{(t)} - x^{(t-1)})$ followed by setting $x^{(t+1)} = y^{(t)} - \alpha \nabla f(y^{(t+1)})$. We use the optimal choices $a^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(a^{(t)})^2}\right)$ and $\alpha = \frac{1}{L}$. Here, the smoothness constant L is given by the largest eigenvalue of $A^T A + \frac{\lambda}{\varepsilon} D^T D$, where $D \in \mathbb{R}^{2n \times n}$ is given by "stacking" D_h and D_w, that is, $D = \begin{pmatrix} D_h & D_w \end{pmatrix}^T$. On the other hand, the algorithmic update of $\mathcal{A}$ is visualized in Figure 3.9 and consists of an update-block, which combines the directions $d_1^{(t)}$, $d_2^{(t)}$ and their "interaction" $d_1^{(t)} \odot d_2^{(t)}$ into the new update direction $d^{(t)}$, and a block to compute a step-size from the norms of the gradient and momentum term. Note that we use 3×3-convolutions this time, that is, we incorporate the knowledge

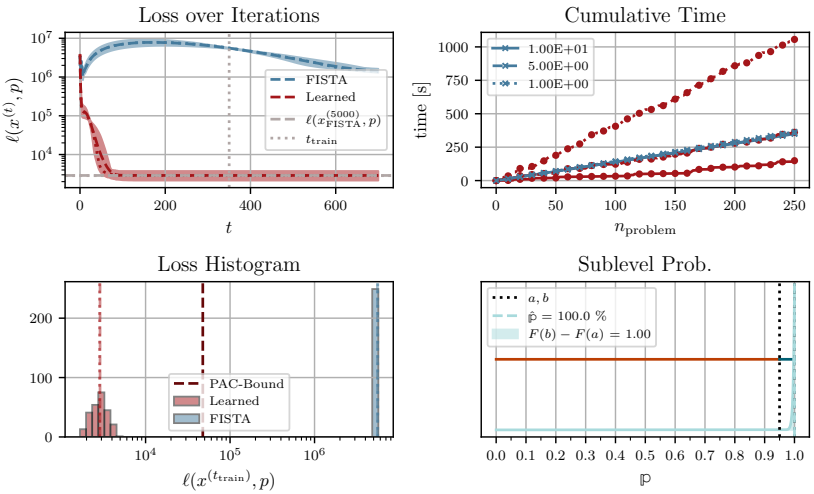

Figure 3.11: Upper left: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded region represents the 10th to 90th percentile. The learned algorithm is shown in red and the *fast iterative shrinkage-thresholding algorithm* (FISTA) is shown in blue. **Upper right:** Cumulative computation time to solve the test problems up to a certain accuracy measured by $\ell(x^{(t)}, p) - \ell(x^{(5000)}_{\text{FISTA}}, p) < \varepsilon$ (both algorithms are run for at most $t_{\max} = 5000$ iterations). **Lower left:** Loss histogram (after $t_{\text{train}} = 350$ iterations) with predicted PAC-bound. **Lower right:** Bayesian estimate for the sublevel probability (Beta-posterior; light blue line) and the imposed constraints a, b (black dotted line).

about an image-processing problem into the design of the optimization algorithm. Further details can be found in Appendix A.2. The results of this experiment are summarized in Figure 3.8: The upper left plot shows that the learned algorithm is much faster than NAG in terms of the loss over the iterations, reaching a loss close to the ground-truth after only 5 iterations. The upper right plot confirms this finding also in terms of computation time. Yet, one can observe a strong increase in computation time for the dotted line (loss per pixel of about $\frac{1}{46750}$), indicating that the learned algorithm often is not able to reach this accuracy. In the lower left plot, one can observe that the predicted PAC-bound is not perfectly tight, yet provides the guarantee to outperform NAG in this setting. Finally, the lower right plot shows that, while the algorithm did not reach the sublevel set in all of the test cases, the probabilistically constrained optimization/sampling procedure did work correctly.

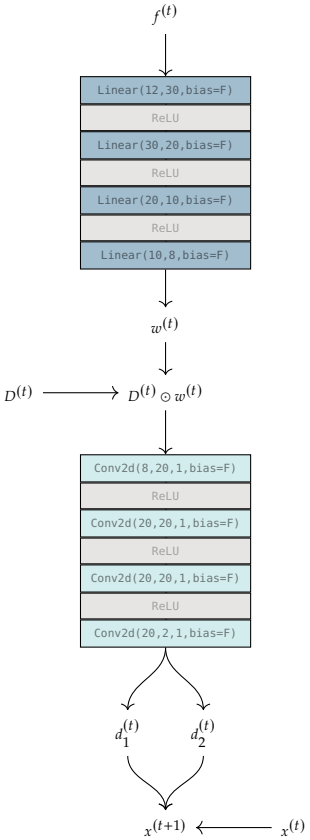

Figure 3.10: Update step of $\mathcal{A}$ for the LASSO-experiment: Based on the given features $f^{(t)} \in \mathbb{R}^{12}$, the first block computes a weighting vector, which is used to weigh the different columns (directions) in the matrix $D^{(t)}$. Here, the notation $D^{(t)} \odot w^{(t)}$ should be understood in the sense that each direction in $D^{(t)}$ gets multiplied by one scalar in $w^{(t)}$. Then, based on $D^{(t)} \odot w^{(t)}$, the second block predicts two directions $d_1^{(t)}, d_2^{(t)}$, where $d_1^{(t)}$ only acts on the zero entries, and $d_2^{(t)}$ acts on the non-zero entries. Finally, we update $x^{(t+1)}$ based on Equation (3.5).

3.6.3 LASSO

Here we consider the LASSO problem (Tibshirani, 1996), that is, a non-smooth problem of the form:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1 \qquad A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m,$$

with $m \leq n$. Thus, we are solving an underdetermined system of linear equations with an additional ℓ_1-regularization term, which promotes sparsity in the solution (see Hastie et al., 2009). Hence, the optimization variable is given by $x \in \mathbb{R}^n$. As baseline we use the *fast iterative shrinkage-thresholding algorithm* (FISTA) by Beck et al. (2009), which performs an extrapolation step followed by a proximal gradient step, that is, abbreviating $h(x) := \frac{1}{2}\|Ax - b\|^2$ and $g(x) := \lambda \|x\|_1$, the update is given by first computing $y^{(t)} = x^{(t)} + \frac{a^{(t)}-1}{a^{(t+1)}}\left(x^{(t)} - x^{(t-1)}\right)$ followed by setting $x^{(t+1)} = \text{prox}_{\alpha g}\left(y^{(t)} - \alpha \nabla h(y^{(t)})\right)$. Here, the proximal mapping is defined as $\hat{x} = \text{prox}_{\alpha g}(\bar{x}) = \arg\min_{x \in \mathbb{R}^n} \lambda \|x\|_1 + \frac{1}{2\alpha}\|x - \bar{x}\|_2^2$, and can be computed efficiently in closed-form yielding the so-called *soft-thresholding operator*:

$$\hat{x}_i = \begin{cases} \bar{x}_i - \alpha\lambda \frac{\bar{x}_i}{|\bar{x}_i|}, & |\bar{x}_i| > \alpha\lambda\,; \\ 0, & \text{otherwise}, \end{cases}, \quad i = 1, \ldots, n\,.$$

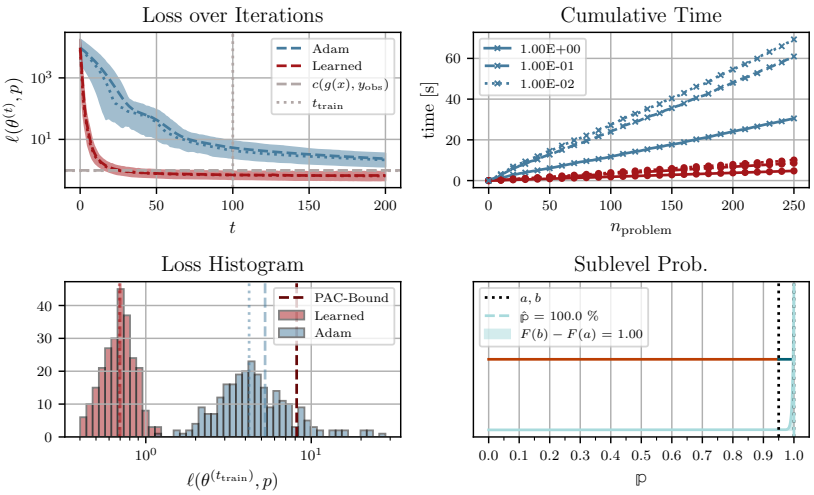

Figure 3.12: Upper left: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions indicate the 10th to 90th percentile. The learned algorithm is shown in red and Adam in blue. **Upper right:** Cumulative computation time to solve the test problems up to a certain accuracy measured by $\ell(\theta^{(t)}, p) - c(X_i, Y_i) < \varepsilon$ (both algorithms are run for at most $t_{\max} = 5000$ iterations). **Lower left:** Loss histogram (after t_{train} iterations) and predicted PAC-bound. **Lower right:** Bayesian estimate for the sublevel probability (Beta-posterior; light blue line) and the imposed constraints a, b (black dotted line).

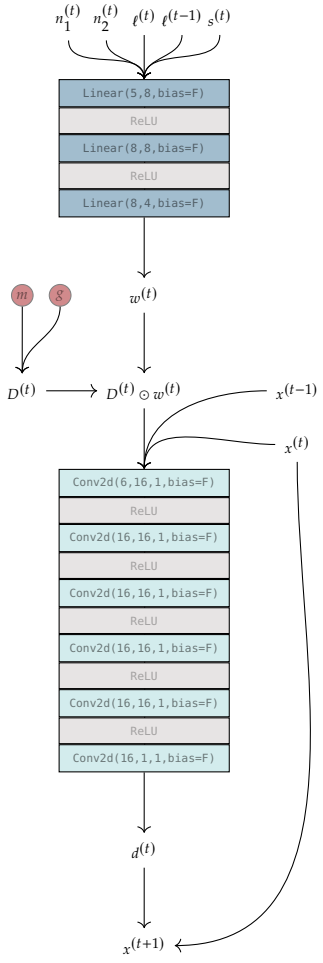

Figure 3.13: Update step of $\mathcal{A}$ for the neural-network-training experiment: The first block (fully-connected) computes a weighting vector, which is used to weigh the different columns (directions) in the matrix $D^{(t)}$. Here, the notation $D^{(t)} \odot w^{(t)}$ should be understood in the sense that each direction in $D^{(t)}$ gets multiplied by one scalar in $w^{(t)}$. Based on these, the second block predicts the new update.

We choose $\alpha = 1/L$, where L is the largest eigenvalue of $A^T A$, that is, the smoothness parameter of h, while $a^{(t)}$ is updated iteratively as $a^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(a^{(t)})^2}\right)$. On the other hand, the algorithmic update of the learned algorithm $\mathcal{A}$ consists of two blocks, that are used consecutively: The first block consists of linear layers with ReLU-activations functions and gets a feature-vector $f^{(t)}$ as input. From this, it computes a weighting-vector $w^{(t)}$, which is used to weigh the different columns of the matrix $D^{(t)}$, which are given by several unit-vectors (directions), such as the normalized gradient. Then, this reweighted matrix $D^{(t)} \odot w^{(t)}$ gets inserted into the second block, which consists of 1x1 convolutional layers with ReLU-activation functions and predicts the two directions $d_1^{(t)}$ and $d_2^{(t)}$. Finally, these get used to compute the final update with the proximal mapping, given by

$$x^{(t+1)} := \operatorname{prox}_{\alpha g}\left(x^{(t)} + \frac{d_1^{(t)} - \nabla \ell(x^{(t)}, p) + \|x^{(t)} - x^{(t-1)}\| \cdot d_2^{(t)}}{L}\right). \quad (3.5)$$

Further details about the experiment, especially the features $f^{(t)}$ and directions $D^{(t)}$, can be found in Appendix A.3. The results of this experiment are summarized in Figure 3.11: The upper left plot shows that the learned algorithm outperforms FISTA by several orders of magnitude, achieving a loss that is similar to the one of $x_{\text{FISTA}}^{(5000)}$ after only 100 iterations, and one can observe that the learned algorithm can be used for more iterations than it was trained for. The upper right plot shows that, up to a certain accuracy, it is also way faster in terms of computation time. Yet, it seems that $\mathcal{A}$ does not reach arbitrary levels of accuracy. The lower left plot shows that the predicted PAC-bound is not perfectly tight, yet guarantees that $\mathcal{A}$ will outperform FISTA for the given number of iterations. Finally, the lower right plot indicates that the algorithm did reach the sublevel set in all of the test cases.

3.6.4 Training Neural Networks

This experiment considers the problem of training a neural network on a regression problem, that is, $\mathcal{A}$ is trained to predict the parameters $\theta \in \mathbb{R}^m$ of a neural network $\mathtt{N}_\theta$, which then is used to predict a function $g : \mathbb{R} \to \mathbb{R}$. Hence, the optimization variable is given by $\theta \in \mathbb{R}^m$. As baseline we use Adam due to Kingma et al. (2015) as it is implemented

in PyTorch (Paszke et al., 2019), which is a widely used optimization algorithm for training neural networks. For tuning, we perform a grid search over 100 step-size parameters, such that its performance is best for the given t_{train} iterations. Note that originally Adam was introduced for stochastic optimization, while we use it in the "full-batch setting" here. On the other hand, the algorithmic update of the learned algorithm $\mathcal{A}$ in Figure 3.13 consists of two blocks: A fully-connected block, which computes a weighting-vector $w^{(t)}$ based on the norms $n_1^{(t)}, n_2^{(t)}$, the losses $\ell(x^{(t)}, p), \ell(x^{(t-1)}, p)$, and the scalar product $s^{(t)} := \langle d_1^{(t)}, d_2^{(t)} \rangle$. Then, this vector gets used to weigh the different directions stored in the matrix $D^{(t)}$, which are given by $d_1^{(t)}, d_2^{(t)}$, and their "pre-conditioned" versions, which we compute by point-wise multiplication with the learned vectors g and m. After that, these weighted directions and the iterates $x^{(t)}$ and $x^{(t-1)}$ serve as different input channels to the update-block, which computes the final update direction $d^{(t)}$ through 1x1 convolutional layers. Further details can be found in Appendix A.4. Figure 3.12 shows the results of this experiment: The upper left plot shows that the learned algorithm clearly outperforms Adam, reaching the ground-truth loss after about 25 iterations, while Adam is not able to reach it within 200 iterations. Further, while the algorithm was trained for 100 iterations, it can be applied for more. The upper right plot confirms that, also in terms of computation time, $\mathcal{A}$ is way faster in training the neural network than Adam. The lower left plot shows that the predicted PAC-bound is not perfectly tight, yet yields a reasonable bound on the average performance, and guarantees to perform roughly as good as Adam (on average). The lower right plot indicates that the algorithm did reach the sublevel set in all test cases.

3.7 Discussion and Limitations

The motivation for this initial work was to use more structure in a given problem than is analytically tractable. For this, we considered a distribution over parametric loss functions and formulated the (ultimate) goal in (3.1), that is, to find a solution to each realization from this distribution. Under reasonable assumptions, this problem is too general to be solved. This led to the formulation of the performance of an algorithm in terms of its risk, that is, the expected loss after a given and fixed number of iterations. However, since this is intractable, we derived PAC-Bayesian generalization bounds relating the risk to the empirically observable performance on a data set. This resulted in the formulation of a training objective, which relies heavily on the existence of a prior distribution satisfying our assumptions and yielding a good performance. As such a distribution is typically not known, we derived a procedure to construct it. This involved several key design choices, such as the loss-function, specific randomization steps, and, especially, the probabilistic constraints. Finally, we validated the resulting learning procedure on four practically relevant problems and showed that it yields a superior performance. While these experimental results are promising, we nevertheless see five main limitations of our work. First, the only guarantee that is provided by the PAC-Bayesian bound is an upper bound on the function value after a specified number of iterations. In particular, it does not guarantee that the function values, the iterates, or the gradient norm actually do converge. Second, our learning procedure is *not guaranteed to work* and still involves many design choices. Third, one still has to find a good architecture for each given problem, which

can be time-consuming. Fourth, the presented algorithmic procedure has a high computational cost (offline), which however, at least in part, is due to the nature of learning-to-optimize. Finally, the procedure only applies to deterministic algorithms.

3.7.1 Thoughts

In summary, these limitations are in large part based on the inadequacy of the model: The setting is restricted to such an extend that the algorithm can actually be treated as a fixed mapping. While this idea can be motivated from a practical perspective, it bears one major problem:

This is simply not a faithful model of optimization algorithms.

Rather, an optimization algorithm is a procedure that is applied iteratively and, in this way, generates a whole sequence. Furthermore, many optimization algorithms in practice, especially in large-scale applications like deep learning, are actually stochastic algorithms, that is, the resulting sequence of iterates is a stochastic process. Therefore, the next section introduces a new model for optimization algorithms, which will remove these limitations. Ultimately, this will also result in new kinds of convergence guarantees for (learned) optimization algorithms and additionally allows for considering completely new kinds of performance measures.

A Markovian Model for Learning-to-Optimize

4

Many guarantees of interest in optimization rely on the *whole trajectory* of the optimization algorithm, that is, the whole sequence of iterates $(x^{(t)})_{t\in\mathbb{N}_0}$. For example, consider the case of proving a linear rate of convergence: The sequence of iterates $(x^{(t)})_{t\in\mathbb{N}_0}$ converges linearly to x^* with rate $\rho < 1$, if $\|x^{(t+1)} - x^*\| \leq \rho\|x^{(t)} - x^*\|$ *for all* $t \in \mathbb{N}_0$. Unfortunately, the previous model is not capable of deriving such guarantees, because it assumes, right from the start, that the algorithm will perform only a finite number of iterations.

Therefore, in this chapter, we introduce a new model for learning-to-optimize, which allows for considering the whole trajectory of an optimization algorithm and, additionally, also allows for stochastic algorithms. It is based on the following simple line of thought: To generate a whole sequence, typically the optimization algorithm is applied iteratively, that is, the new iterate is computed based on the current iterate. Thus, at least, the update has to be of the form:

$$x^{(t+1)} = \mathcal{A}(x^{(t)})\,.$$

However, in this case the algorithm would perform the same update-steps independently of the loss-function $\ell(\cdot, p)$ it is applied to. Therefore, to distinguish between different loss-functions, the algorithm should also depend on the parameters $p \in \mathcal{P}$, such that the update has to be extended to:

$$x^{(t+1)} = \mathcal{A}(p, x^{(t)})\,.$$

While this allows for applying the algorithm to different realizations of the loss-function, it does neither allow for adjusting the algorithm nor for learning it. Thus, we have to extend the algorithmic update even further, and introduce additional hyperparameters $h \in \mathcal{H}$:

$$x^{(t+1)} = \mathcal{A}(h, p, x^{(t)})\,.$$

Now, given an update like this, the algorithm can adjust points in the optimization space, it can be adapted to different loss-functions, and it can be learned. Nevertheless, it is still deterministic. Hence, to be able to model stochastic algorithms, we also have to include a way to randomize each iteration:

$$x^{(t+1)} = \mathcal{A}(h, p, x^{(t)}, r^{(t+1)})\,,$$

where we use the index $t + 1$ because, conditioned on the information at time t, the next iterate should not be predictable. Finally, there is one remaining problem: For now, the algorithm can only update points *in the optimization space* and it does this solely based on the previous iterate. However, many optimization algorithms either update one or more auxiliary variables before actually updating the optimization variable or they rely on more than just one previous iterate. Hence, instead of solely "acting" on the optimization variable x, we introduce a higher dimensional *state variable* $z \in \mathcal{Z}$, such that the optimization space $\mathcal{X}$ is a sub-space of the *state space* $\mathcal{Z}$ and there exists a projection $\pi_{\mathcal{X}} : \mathcal{Z} \to \mathcal{X}$ with $\pi_{\mathcal{X}}(z) = x$. To summarize, in this chapter we consider an iterative update of the form:

$$z^{(t+1)} = \mathcal{A}(h, p, z^{(t)}, r^{(t+1)})\,, \tag{4.1}$$

Note that this problem does not occur when treating the whole optimization algorithm as a fixed map.

and, as before, the hyperparameters $h \in \mathcal{H}$ allow for adjusting the algorithm, that is, eventually they are learned, and the parameters $p \in \mathcal{P}$ specify the objective function $\ell(\cdot, p)$ the algorithm is applied to. However, the new variable $r^{(t+1)}$ models the *internal randomness* of $\mathcal{A}$ and the state $z^{(t)}$ comprises the optimization variable, as well as any other auxiliary variable that is needed to compute the next iterate. Therefore, we make the following assumptions:

Assumption 4.0.1 *We are given four Polish probability spaces: the state space* $(\mathcal{Z}, \mathfrak{B}(\mathcal{Z}), \mathbb{P}_{Z^{(0)}})$*, the parameter space* $(\mathcal{P}, \mathfrak{B}(\mathcal{P}), \mathbb{P}_P)$*, the hyperparameter space* $(\mathcal{H}, \mathfrak{B}(\mathcal{H}), \mathbb{P}_H)$*, and the randomization space* $(\mathcal{R}, \mathfrak{B}(\mathcal{R}), \mathbb{P}_R)$*.*

This assumption will actually not really be needed until the next chapter.

Assumption 4.0.2 *$\mathcal{Z}$ is the product of the* optimization space *$\mathcal{X}$ and the* memory space *$\mathcal{M}$, that is, $\mathcal{Z} = \mathcal{X} \times \mathcal{M}$, where both $\mathcal{X}$ and $\mathcal{M}$ are Polish spaces with metrics $d_{\mathcal{X}}$ and $d_{\mathcal{M}}$.*

Note that we can always "lift" ℓ to $\mathcal{Z}$ by concatenating it with the projection $\pi_{\mathcal{X}}$. More generally, and as is done in the underlying paper, one could also directly consider the more general case of a loss-function $\ell : \mathcal{Z} \times \mathcal{P} \to [0, +\infty]$, which, for example, also allows for considering Lyapunov functions as "loss". As this is not really needed in the following, we omit doing it here to highlight the difference between optimization variable and state variable.

Assumption 4.0.3 *We are given a measurable function $\ell : \mathcal{X} \times \mathcal{P} \to [0, +\infty]$, the* loss function*, and a measurable map $\mathcal{A} : \mathcal{H} \times \mathcal{P} \times \mathcal{Z} \times \mathcal{R} \to \mathcal{Z}$, the* algorithmic update*.*

Based on these assumptions, we will derive a widely applicable model for optimization algorithms that can be learned on a data set of problem instances. However, before doing that, we provide some concrete examples of the proposed setting:

Example 4.0.1 (i) Consider stochastic gradient descent to minimize the parametric empirical risk $\ell(x, p) := \frac{1}{m} \sum_{i=1}^{m} f_i(x, p)$. In each iteration, the algorithm samples an index j uniformly in $\{1, \dots, m\}$ and performs the update:

$$x^{(t+1)} = x^{(t)} - h \nabla f_j(x^{(t)}, p)\,,$$

where $h > 0$ is a step-size. This can be summarized into a single mapping $\mathcal{A}$ as:

$$\mathcal{A}(h, p, x^{(t)}, r^{(t+1)}) := x^{(t)} - h \sum_{i=1}^{m} \mathbb{1}_{\{i\}}(r^{(t+1)}) \nabla f_i(x^{(t)}, p)\,,$$

where $r^{(t+1)} \sim \mathrm{U}_{\{1,\dots,m\}}$. Thus, it holds $\mathcal{Z} = \mathcal{X} = \mathbb{R}^n$, $\mathcal{H} = [0, \infty)$, and $\mathcal{R} = \{1, \dots, m\}$.

(ii) Consider the heavy-ball algorithm, which performs the following update:

$$x^{(t+1)} = x^{(t)} - \alpha \nabla \ell(x^{(t)}, p) + \beta(x^{(t)} - x^{(t-1)})\,.$$

Thus, the hyperparameters are given by $h = (\alpha, \beta) \in \mathbb{R}^2$ and, if we define the state as $z^{(t)} = (x^{(t)}, x^{(t-1)})$, the state space is given by $\mathcal{Z} = \mathbb{R}^{2n}$, such that the optimization and memory space are given by $\mathcal{X} = \mathcal{M} = \mathbb{R}^n$ with corresponding projection $\pi_{\mathcal{X}} : \mathbb{R}^{2n} \to \mathbb{R}^n$.

(iii) Nesterov's accelerated gradient descent performs the following

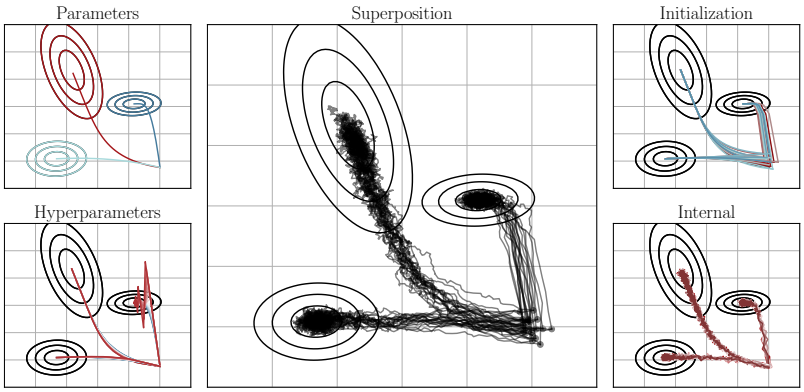

Figure 4.1: Middle: Superposition of different sources of randomness. **Upper left:** The algorithm can be applied to several problem instances coming from a common distribution. **Upper right:** The algorithm might be started from different, randomly chosen initializations. **Lower left:** There might be randomness (or uncertainty) in the choice of the hyperparameters of the algorithm. **Lower right:** The algorithmic update might be inherently stochastic.

update:

$$y^{(t)} = x^{(t)} + \frac{a^{(t)} - 1}{a^{(t+1)}}(x^{(t)} - x^{(t-1)}),$$
$$x^{(t+1)} = y^{(t)} - \tau \nabla f(y^{(t)}),$$

where we assume that $a^{(t+1)} = g_1(a^{(t)})$, which, for example, is satisfied for the worst-case optimal choice. Thus, by including $x^{(t)}, x^{(t-1)}$ and $a^{(t)}$ in the state variable $z^{(t)}$, the algorithmic update can be defined as:

$$z^{(t+1)} := \begin{pmatrix} a^{(t+1)} \\ x^{(t)} \\ x^{(t+1)} \end{pmatrix} := \begin{pmatrix} g_1(a^{(t)}) \\ x^{(t)} \\ (g_3 \circ g_2)(x^{(t)}, x^{(t-1)}, a^{(t)}) \end{pmatrix},$$

where g_2 computes the auxiliary variable $y^{(t)}$, and g_3 performs the gradient-step at $y^{(t)}$. In this case, the state space is given by $\mathcal{Z} = \mathbb{R}^{2n+1}$, the optimization space is given by $\mathcal{X} = \mathbb{R}^n$ and the memory space is given by $\mathcal{M} = \mathbb{R}^{n+1}$. Especially, also algorithms that explicitly depend on a variable considered as "time" can be treated in this setting, if the corresponding parameter is included in the state.

(iv) Consider an update of the form:

$$z^{(t+1)} := \begin{pmatrix} y^{(t+1)} \\ x^{(t+1)} \end{pmatrix} := \begin{pmatrix} \mathcal{N}_1(h_1, p, y^{(t)}, r^{(t+1)}) \\ x^{(t)} - \mathcal{N}_2(h_2, p, z^{(t)}, r^{(t+1)}) \end{pmatrix},$$

where, additionally to updating the iterates $x^{(t)} \in \mathbb{R}^n$ with a neural network $\mathcal{N}_2$, one updates a hidden state $y^{(t)} \in \mathbb{R}^m$ with another neural network $\mathcal{N}_1$. In this case, the state would consist of $z^{(t)} = (x^{(t)}, y^{(t)}) \in \mathbb{R}^{m+n}$, $\pi_{\mathcal{X}}$ would be the projection from $\mathbb{R}^{m+n}$ onto $\mathbb{R}^n$ with $\pi_{\mathcal{X}}(z^{(t)}) = x^{(t)}$, and the hyperparameters h would be given by the parameters of these two networks, that is, the tuple $h = (h_1, h_2)$.

As the example shows, this model is widely applicable and accounts for many optimization algorithms. However, this also begs a fundamental difficulty: The resulting iterates form a rather generic stochastic process, such that it is infeasible to derive generalization guarantees for each single run of the optimization algorithm in this general setting, that is, guarantees that hold almost-surely. On the other hand, this process is actually driven by a single simple equation, namely Equation (4.1), which might allow for "disentangling the randomness": The parameters,

the hyperparameters, the initialization, and the internal randomness all influence different aspects of the *distribution of the trajectory*, and some of these effects are actually quite intuitive (for a visualization of this idea, see Figure 4.1). Thus, the average behavior of the algorithm might in fact be tractable. Since the expected value of a random variable is defined as the integral w.r.t. its distribution, we therefore need to model the distribution of the trajectories generated by the algorithm and the way in which they evolve with the parameters and hyperparameters in the correct way. The key ingredient for doing this is the so-called *transition kernel*.

4.1 The Transition Kernel

By rearranging one can also think of this as $\mathcal{H} \times (\mathcal{P} \times \mathcal{I} \times \mathcal{R}^{\mathbb{N}})^N$, that is, we have one hyperparameter (one algorithm) that we apply to N different problems.

Since we eventually want to learn the hyperparameters of the algorithm on a data set of size N, we define the measurable space $(\Omega_{\text{pre}}, \mathfrak{A}_{\text{pre}})$ as:

$$\Omega_{\text{pre}} := \mathcal{H} \times \mathcal{P}^N \times \mathcal{I}^N \times (\mathcal{R}^{\mathbb{N}})^N, \quad \mathfrak{A}_{\text{pre}} := \mathfrak{B}\left(\Omega_{\text{pre}}\right),$$

and endow it with the probability measure

$$\mathbb{P}_{\text{pre}} := \mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N} \otimes \mathbb{P}_{Z^{(0)}}^{\otimes N} \otimes \bigotimes_{i=1}^{N} \mathbb{P}_R^{\otimes \mathbb{N}}.$$

Further, we denote the canonical process on Ω_{pre}, that is, the coordinate projections, by:

$$X := \left(H, P_{[N]}, Z_{[N]}^{(0)}, (R^{(t)})_{t \in \mathbb{N}, [N]}\right),$$

where we use the compact notation $P_{[N]}$ to denote the vector with N coordinates, that is, $P_{[N]} = (P_1, ..., P_N)$ (analogously for other variables). Thus, it holds that $H \sim \mathbb{P}_H$, $P_1, ..., P_N \overset{iid}{\sim} \mathbb{P}_P$, $Z_1^{(0)}, ..., Z_N^{(0)} \overset{iid}{\sim} \mathbb{P}_{Z^{(0)}}$, and all $R_n^{(t)} \sim \mathbb{P}_R$, $n = 1, ..., N$, $t \in \mathbb{N}$, are i.i.d. Then, given a cylinder set $\mathsf{B}_1 \times ... \times \mathsf{B}_N \in \mathfrak{B}(\mathcal{I})^{\otimes N}$, Fubini's theorem yields the following factorization:

$$\begin{aligned}
&\mathbb{P}_{\text{pre}}\left\{\left(\mathcal{A}(H, P_1, Z_1^{(0)}, R_1^{(0)}), ..., \mathcal{A}(H, P_N, Z_N^{(0)}, R_N^{(0)})\right) \in \mathsf{B}_1 \times ... \times \mathsf{B}_N\right\} \\
&= \int_{\mathcal{H} \times \mathcal{P}^N \times \mathcal{I}^N} \prod_{n=1}^{N} \mathbb{P}_R\left\{\mathcal{A}(h, p_n, z_n^{(0)}, \cdot) \in \mathsf{B}_n\right\} \left(\mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N} \otimes \mathbb{P}_{Z^{(0)}}^{\otimes N}\right)(dh, dp_{[N]}, dz_{[N]}^{(0)})
\end{aligned}$$

Here, we have the following simple, yet fundamental result:

Lemma 4.1.1 *Suppose that $\mathcal{A}$ satisfies Assumption 4.0.3. Then the map*

$$\left((h, p, z), \mathsf{B}\right) \mapsto \left(\mathbb{P}_R \circ \mathcal{A}(h, p, z, \cdot)^{-1}\right)\{\mathsf{B}\}$$

is a probability kernel from $\mathcal{H} \times \mathcal{P} \times \mathcal{I}$ to $\mathcal{I}$. Similarly, the map

$$\left((h, p_{[N]}, z_{[N]}), \mathsf{A}\right) \mapsto \bigotimes_{n=1}^{N} \left(\mathbb{P}_R \circ \mathcal{A}(h, p_n, z_n, \cdot)^{-1}\right)\{\mathsf{A}\}$$

is a probability kernel from $\mathcal{H} \times \mathcal{P}^N \times \mathcal{I}^N$ to $\mathcal{I}^N$.

Proof. We use the following notation:

$$\gamma(h,p,z)\{\mathsf{B}\} := \left(\mathbb{P}_R \circ \mathcal{A}(h,p,z,\cdot)^{-1}\right)\{\mathsf{B}\}$$
$$\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\} := \bigotimes_{n=1}^{N}\left(\mathbb{P}_R \circ \mathcal{A}(h,p_n,z_n,\cdot)^{-1}\right)\{\mathsf{A}\}$$

Since $\mathcal{A} : \mathscr{H}\times\mathscr{P}\times\mathfrak{Z}\times\mathscr{R}\to\mathfrak{Z}$ is measurable, and $\mathbb{P}_R$ can be seen as the constant kernel from $\mathscr{H}\times\mathscr{P}\times\mathfrak{Z}\to\mathscr{R}$, the first statement follows directly from Lemma 2.1.5 with $\mathscr{U} := \mathscr{H}\times\mathscr{P}\times\mathfrak{Z}$, $\mathscr{V} := \mathscr{R}$ and $\mathscr{W} := \mathfrak{Z}$. Then, however, if $(h,p_{[N]},z_{[N]})$ is given, we also get that $\Gamma(h,p_{[N]},z_{[N]})$ is a measure on $\mathfrak{Z}^N$. Therefore, it remains to show measurability of $(h,p_{[N]},z_{[N]})\mapsto\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\}$ for fixed $\mathsf{A}\in\mathfrak{B}(\mathfrak{Z})^{\otimes N}$. We do this by a monotone-class argument, that is, based on Theorem 2.1.1. For this, define the following two classes of sets:

$$\mathcal{C} := \{\mathsf{A}^0\times\ldots\times\mathsf{A}^N \;:\; \mathsf{A}^0,\ldots,\mathsf{A}^K\in\mathfrak{B}(\mathfrak{Z})\},$$
$$\mathcal{D} := \{\mathsf{A}\in\mathfrak{B}(\mathfrak{Z})^{\otimes N} : (h,p_{[N]},z_{[N]})\mapsto\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\}\text{ is measurable}\}.$$

$\mathcal{C}$ is the class of cylinder sets, which is a $\cap$-stable generator of the product-σ-field by definition. Thus, $\mathcal{C}$ is a π-system with $\sigma(\mathcal{C}) = \mathfrak{B}(\mathfrak{Z})^{\otimes N}$. Furthermore, for any $\mathsf{B}_1\times\ldots\times\mathsf{B}_N\in\mathcal{C}$, it holds that:

$$\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{B}_1\times\ldots\times\mathsf{B}_N\} = \prod_{n=1}^{N}\gamma(h,p_n,z_n)\{\mathsf{B}_n\}.$$

Since γ is a probability kernel, that is, measurable in (h,p_n,z_n), it follows that $\mathcal{C}\subset\mathcal{D}$. Thus, it remains to show that $\mathcal{D}$ is a λ-system. Clearly, it holds that $\mathfrak{Z}^N\in\mathcal{C}\subset\mathcal{D}$. Thus, take $\mathsf{A},\mathsf{B}\in\mathcal{D}$ with $\mathsf{A}\supset\mathsf{B}$. Then, since $\Gamma(h,p_{[N]},z_{[N]})$ is a probability measure for each fixed $(h,p_{[N]},z_{[N]})$, we have the following pointwise equality:

$$\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\setminus\mathsf{B}\} = \Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\} - \Gamma(h,p_{[N]},z_{[N]})\{\mathsf{B}\}.$$

Since $\mathsf{A},\mathsf{B}\in\mathcal{D}$, we have that the right-hand side is measurable, which implies that $\mathsf{A}\setminus\mathsf{B}\in\mathcal{D}$. Finally, for $\mathsf{A}_1,\mathsf{A}_2,\ldots\in\mathcal{D}$ with $\mathsf{A}_n\uparrow\mathsf{A}$, by the continuity of measures, we have the pointwise equality:

$$\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\} = \lim_{n\to\infty}\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}_n\}.$$

Since limits of measurable functions are measurable, it follows that $\mathsf{A}\in\mathcal{D}$. Therefore, $\mathcal{D}$ is a λ-system and Theorem 2.1.1 yields $\mathfrak{B}(\mathfrak{Z})^{\otimes N}\subset\mathcal{D}$. Thus, Γ is a probability kernel from $\mathscr{H}\times\mathscr{P}^N\times\mathfrak{Z}^N$ to $\mathfrak{Z}^N$. □

This motivates the following definition:

Definition 4.1.1 *The* transition kernel *of $\mathcal{A}$ is given by the probability kernel $\gamma : \mathscr{H}\times\mathscr{P}\times\mathfrak{Z}\to\mathfrak{Z}$, defined as:*

$$\gamma(h,p,z)\{\mathsf{B}\} := \left(\mathbb{P}_R \circ \mathcal{A}(h,p,z,\cdot)^{-1}\right)\{\mathsf{B}\}.$$

The joint transition kernel *of $\mathcal{A}$ is given be the probability kernel $\Gamma : \mathscr{H}\times\mathscr{P}^N\times\mathfrak{Z}^N\to\mathfrak{Z}^N$, defined as:*

$$\Gamma(h,p_{[N]},z_{[N]})\{\mathsf{A}\} := \bigotimes_{n=1}^{N}\left(\mathbb{P}_R \circ \mathcal{A}(h,p_n,z_n,\cdot)^{-1}\right)\{\mathsf{A}\}.$$

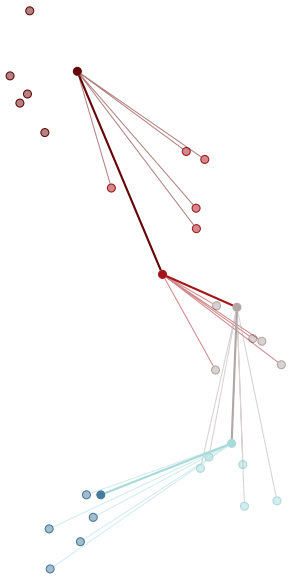

Figure 4.2: Visualization of the transition kernel γ: At each point, there are several possibilities to which the algorithm could move.

Example 4.1.1 (i) The transition kernel of stochastic gradient descent from Example 4.0.1 is given by:

$$\mathbb{P}_R\left\{\mathcal{A}(h,p,x,\cdot)\in \mathsf{B}\right\} := \mathrm{U}_{\{1,\dots,m\}}\left\{x - h\sum_{i=1}^{m}\mathbb{1}_{\{i\}}(\cdot)\nabla f_i(x,p)\in \mathsf{B}\right\}.$$

Basically, this is the transition kernel used by Bianchi et al. (2022), and our definition is a direct generalization of it.

(ii) The transition kernel can be seen as a direct generalization of the usual algorithmic update from points to measures. To see this, consider a deterministic algorithm. Then, we get:

$$\mathbb{P}_R\left\{\mathcal{A}(h,p,z)\in \mathsf{B}\right\} = \mathbb{1}_{\mathsf{B}}(\mathcal{A}(h,p,z)) = \delta_{\mathcal{A}(h,p,z)}\{\mathsf{B}\}.$$

Thus, integrating w.r.t. γ just yields the new iterate in this case. Taking this approach, one can analyze algorithms in the usual way. Additionally, we also recover the average-case setting of Pedregosa et al. (2020) and Scieur et al. (2020).

For a deterministic algorithm, given an initial point $z^{(0)} \in \mathcal{Z}$, we can compute the t-th state by iteratively applying $\mathcal{A}$ t-times to $z^{(0)}$. With the transition kernel, this is generalized to distributions: Given an initial distribution $\mathbb{P}_{Z^{(0)}}$, we can compute the distribution of the t-th state by iteratively applying γ t-times to it. This can be expressed more compactly in terms of the *transition semi-group*:

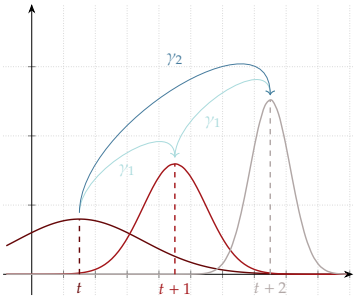

Figure 4.3: Visualization of the transition semi-group $(\gamma_t)_{t\in\mathbb{N}_0}$.

Definition 4.1.2 *Given the transition kernel γ of $\mathcal{A}$, we define the* transition semi-group $(\gamma_t)_{t\in\mathbb{N}_0}$ *of $\mathcal{A}$ recursively through:*

$$\gamma_0(h,p,z) := \delta_z,$$
$$\gamma_t(h,p,z) := \gamma_{t-1}(h,p,z)\cdot\gamma(h,p,\cdot).$$

Similarly, given the joint transition kernel Γ of $\mathcal{A}$, the joint transition semi-group *of $\mathcal{A}$ is defined recursively through:*

$$\Gamma_0(h,p_{[N]},z_{[N]}) := \delta_{z_{[N]}},$$
$$\Gamma_t(h,p_{[N]},z_{[N]}) := \Gamma_{t-1}(h,p_{[N]},z_{[N]})\cdot\Gamma(h,p_{[N]},\cdot).$$

As shown next, through this recursive definition, the factorization of Γ extends to the joint transition semi-group $(\Gamma_t)_{t\in\mathbb{N}_0}$, which will ultimately result in a corresponding factorization of the joint distribution of the processes $Z_1,\dots,Z_N$ corresponding to the parameters $P_1,\dots,P_N$.

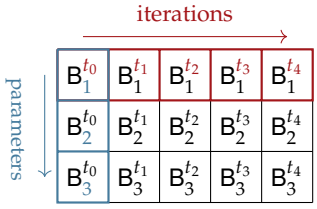

Figure 4.4: Visual representation of the involved sets for the case $N = 3$ and $K = 4$. The product with the joint transition semi-group is applied to the whole set, while the product with the transition semi-group is applied "row-wise".

Lemma 4.1.2 *For every $(h,p_{[N]},z_{[N]}) \in \mathcal{H}\times\mathcal{P}^N\times\mathcal{Z}^N$, $K\in\mathbb{N}$, $t_0 < t_1 < \dots < t_K \in \mathbb{N}_0$, and any set $\left(\mathsf{B}_1^{t_0}\times\dots\times\mathsf{B}_N^{t_0}\right)\times\dots\times\left(\mathsf{B}_1^{t_K}\times\dots\times\mathsf{B}_N^{t_K}\right)$ with $\mathsf{B}_j^{t_k}\in\mathfrak{B}(\mathcal{Z})$, it holds that:*

$$\left(\delta_{z_{[N]}}\otimes\bigotimes_{k=0}^{K-1}\Gamma_{t_{k+1}-t_k}(h,p_{[N]},\cdot)\right)\left\{\bigtimes_{n=1}^{N}\mathsf{B}_n^{t_0}\times\dots\times\bigtimes_{n=1}^{N}\mathsf{B}_n^{t_K}\right\}$$
$$=\bigotimes_{n=1}^{N}\left(\delta_{z_n}\otimes\bigotimes_{k=0}^{K-1}\gamma_{t_{k+1}-t_k}(h,p_n,\cdot)\right)\left\{\bigtimes_{i=0}^{K}\mathsf{B}_1^{t_i}\times\dots\times\bigtimes_{i=0}^{K}\mathsf{B}_N^{t_i}\right\}$$

Proof. Inserting $B_j^{t_k} = \mathcal{Z}$ if necessary, w.l.o.g. we can restrict to the case $t_k = k$, that is, $t_{k+1} - t_k = 1$. Then, we prove the result by induction. Thus,

first, consider the case $K = 1$:

$$\begin{aligned}&\Big(\delta_{z_{[N]}} \otimes \Gamma(h, p_{[N]}, \cdot)\Big)\{(\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_N^0) \times (\mathsf{B}_1^1 \times \ldots \times \mathsf{B}_N^1)\} \\ &= \int_{\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_N^0} \Gamma(h, p_{[N]}, y_{[N]})\{\mathsf{B}_1^1 \times \ldots \times \mathsf{B}_N^1\}\ \delta_{z_{[N]}}(dy_{[N]})\,.\end{aligned}$$

By definition of Γ, this is the same as:

$$= \int_{\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_N^0} \prod_{n=1}^{N} \gamma(h, p_n, y_n)\{\mathsf{B}_n^1\}\ \delta_{z_{[N]}}(dy_{[N]})\,.$$

By using the fact that $\delta_{z_{[N]}} = \bigotimes_{n=1}^{N} \delta_{z_n}$, Fubini's theorem yields:

$$\begin{aligned}&= \prod_{n=1}^{N} \int_{\mathsf{B}_n^0} \gamma(h, p_n, y_n)\{\mathsf{B}_n^1\}\ \delta_{z_n}(dy_n) \\ &= \prod_{n=1}^{N} (\delta_{z_n} \otimes \gamma(h, p_n, \cdot))\{\mathsf{B}_n^0 \times \mathsf{B}_n^1\} \\ &= \bigotimes_{n=1}^{N} (\delta_{z_n} \otimes \gamma(h, p_n, \cdot))\{(\mathsf{B}_1^0 \times \mathsf{B}_1^1) \times \ldots \times (\mathsf{B}_N^0 \times \mathsf{B}_N^1)\}\,.\end{aligned}$$

Thus, let the statement be true for $K \in \mathbb{N}$ and consider the case $K + 1$. Using the recursive definition $\bigotimes_{k=0}^{K-1} \Gamma(h, p_{[N]}, \cdot) = \Big(\bigotimes_{k=0}^{K-2} \Gamma(h, p_{[N]}, \cdot)\Big) \otimes \Gamma(h, p_{[N]}, \cdot)$, we can write:

Here and below we use the operator-notation for a more compact representation.

$$\begin{aligned}&\Big(\delta_{z_{[N]}} \otimes \bigotimes_{k=0}^{K} \Gamma(h, p_{[N]}, \cdot)\Big)\{(\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_N^0) \times \ldots \times (\mathsf{B}_1^{K+1} \times \ldots \times \mathsf{B}_N^{K+1})\} \\ &= \Big(\delta_{z_{[N]}} \otimes \bigotimes_{k=0}^{K-1} \Gamma(h, p_{[N]}, \cdot)\Big)\Big\{\mathbb{1}_{\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_N^K} \Gamma(h, p_{[N]}, \cdot)\{\mathsf{B}_1^{K+1} \times \ldots \times \mathsf{B}_N^{K+1}\}\Big\}\,.\end{aligned}$$

By the factorization of Γ, this is the same as:

$$= \Big(\delta_{z_{[N]}} \otimes \bigotimes_{k=0}^{K-1} \Gamma(h, p_{[N]}, \cdot)\Big)\Big\{\prod_{n=1}^{N} \mathbb{1}_{\mathsf{B}_n^0 \times \ldots \times \mathsf{B}_n^K} \gamma(h, p_n, \cdot)\{\mathsf{B}_n^{K+1}\}\Big\}$$

By Fubini's theorem and the induction hypothesis, this is the same as:

$$= \prod_{n=1}^{N} \Big(\delta_{z_n} \otimes \bigotimes_{k=0}^{K-1} \gamma(h, p_n, \cdot)\Big)\Big\{\mathbb{1}_{\mathsf{B}_n^0 \times \ldots \times \mathsf{B}_n^K} \gamma(h, p_n, \cdot)\{\mathsf{B}_n^{K+1}\}\Big\}\,.$$

Using again the recursive definition of the product of kernels, this is the same as:

$$\begin{aligned}&= \prod_{n=1}^{N} \Big(\delta_{z_n} \otimes \bigotimes_{k=0}^{K} \gamma(h, p_n, \cdot)\Big)\{\mathsf{B}_n^0 \times \ldots \times \mathsf{B}_n^{K+1}\} \\ &= \bigotimes_{n=1}^{N} \Big(\delta_{z_n} \otimes \bigotimes_{k=0}^{K} \gamma(h, p_n, \cdot)\Big)\Big\{(\mathsf{B}_1^0 \times \ldots \times \mathsf{B}_1^{K+1}) \times \ldots \times (\mathsf{B}_N^0 \times \ldots \times \mathsf{B}_N^{K+1})\Big\}\,.\end{aligned}$$

□

4.2 Distribution of the Trajectories

For every fixed $(h,p) \in \mathscr{H} \times \mathscr{P}$ we have that $(\gamma_t(h,p,\cdot))_{t\in\mathbb{N}_0}$ is a Markovian semi-group on $\mathfrak{X}$ (by definition). Thus, by Theorem 2.1.12 there exist a unique probability measure $\psi_{h,p}$ on $\mathfrak{X}^{\mathbb{N}_0}$, such that for any natural numbers $0 = t_0 < t_1 < \ldots < t_K$, it holds that:

$$\psi_{h,p} \circ \pi_J^{-1} = \mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma_{t_{k+1}-t_k}(h,p,\cdot)\,, \tag{4.2}$$

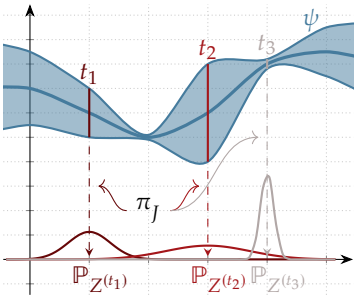

Figure 4.5: Finite-dimensional distributions of ψ.

where $J := \{t_0, \ldots, t_K\}$ and $\pi_J : \mathfrak{X}^{\mathbb{N}_0} \to \mathfrak{X}^{K+1}$ denotes the corresponding coordinate projection. This means that the finite-dimensional distribution of $\psi_{h,p}$ corresponding to the time-points $t_0, \ldots, t_K$ is given by the distribution of the iterates $\left(Z^{(t_0)}, \ldots, Z^{(t_K)}\right)$ generated by $\mathcal{A}(h,p,\cdot,\cdot)$ (with initial distribution $\mathbb{P}_{Z^{(0)}}$). Since the distribution of a stochastic process is uniquely determined by its finite-dimensional distributions (Kallenberg, 2021, Prop. 4.2, p.84), this implies that, for every $(h,p) \in \mathscr{H} \times \mathscr{P}$, there is exactly one distribution on the space of trajectories $\mathfrak{X}^{\mathbb{N}_0}$, namely $\psi_{h,p}$, that describes the iterates corresponding to $\mathcal{A}(h,p,\cdot,\cdot)$. Similarly, there exists a unique probability measure $\Psi_{h,p_{[N]}}$ on $\left(\mathfrak{X}^N\right)^{\mathbb{N}_0}$, such that:

$$\Psi_{h,p_{[N]}} \circ \pi_{J,[N]}^{-1} = \mathbb{P}_{Z^{(0)}}^{\otimes N} \otimes \bigotimes_{k=0}^{K-1} \Gamma_{t_{k+1}-t_k}(h,p_{[N]},\cdot)\,, \tag{4.3}$$

where $\pi_{J,[N]} : \left(\mathfrak{X}^N\right)^{\mathbb{N}_0} \to \left(\mathfrak{X}^N\right)^{K+1}$ denotes the corresponding coordinate projections on $\left(\mathfrak{X}^N\right)^{\mathbb{N}_0}$. Similar to before, $\Psi_{h,p_{[N]}}$ is the unique measure on $\left(\mathfrak{X}^N\right)^{\mathbb{N}_0}$ that describes the joint distribution of the trajectories $Z_{[N]} = (Z_1, \ldots, Z_N)$ in $\mathfrak{X}^N$, which are generated by the collection $\left(\mathcal{A}(h,p_1,\cdot,\cdot), \ldots, \mathcal{A}(h,p_N,\cdot,\cdot)\right)$. Intuitively, and as will be shown later on, this is (up to reordering) the same as considering N individual processes $Z_1, \ldots, Z_N$ on $\mathfrak{X}$ generated by $\mathcal{A}(h,\cdot,\cdot,\cdot)$ on the problem instances $p_1, \ldots, p_N$. First, however, we show that both $\psi_{h,p}$ and $\Psi_{h,p_{[N]}}$ can be "summarized" into a unique kernel:

Note the notation: $Z^{(t)}$ is the t-th random iterate and $z^{(t)}$ is its realization in $\mathfrak{X}$. On the other hand, $Z = (Z^{(t)})_{t\in\mathbb{N}_0}$ is the whole random trajectory, and $\zeta = (z^{(t)})_{t\in\mathbb{N}_0}$ is its realization in $\mathfrak{X}^{\mathbb{N}_0}$.

Theorem 4.2.1 *Suppose that Assumptions 4.0.1 and 4.0.3 hold. Then, the map*

$$\begin{aligned}\psi : (\mathscr{H} \times \mathscr{P}) \times \mathfrak{B}(\mathfrak{X})^{\otimes\mathbb{N}_0} &\to [0,1]\,,\\ \left((h,p),\mathsf{B}\right) &\mapsto \psi(h,p)\{\mathsf{B}\} := \psi_{h,p}\{\mathsf{B}\}\end{aligned}$$

is the unique probability kernel from $\mathscr{H} \times \mathscr{P}$ to $\mathfrak{X}^{\mathbb{N}_0}$, such that (4.2) holds. Similarly, the map

$$\begin{aligned}\Psi : (\mathscr{H} \times \mathscr{P}^N) \times \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes\mathbb{N}_0} &\to [0,1]\,,\\ \left((h,p_{[N]}),\mathsf{B}\right) &\mapsto \Psi(h,p_{[N]})\{\mathsf{B}\} := \Psi_{h,p_{[N]}}\{\mathsf{B}\}\end{aligned}$$

is the unique probability kernel from $\mathscr{H} \times \mathscr{P}^N$ to $\left(\mathfrak{X}^N\right)^{\mathbb{N}_0}$, such that (4.3) holds.

Proof. By construction, we have that $\psi(h,p)$ is a probability measure on $\mathfrak{X}^{\mathbb{N}_0}$ for each $(h,p) \in \mathscr{H} \times \mathscr{P}$. Therefore, we only have to show measurability. Again, we do this by a monotone-class argument based

on Theorem 2.1.1. For this, define the following two classes of sets:

$$\mathcal{C} := \{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K \times \prod_{k>K} \mathcal{Z} \ : \ K \in \mathbb{N}_0, \quad \mathsf{A}^0, \ldots, \mathsf{A}^K \in \mathfrak{B}(\mathcal{Z})\} \,,$$
$$\mathcal{D} := \{\mathsf{A} \in \mathfrak{B}(\mathcal{Z})^{\otimes \mathbb{N}_0} \ : \ (h,p) \mapsto \psi(h,p)\{\mathsf{A}\} \text{ is measurable}\} \,.$$

Also here, $\mathcal{C}$ is the class of cylinder sets, which is $\cap$-stable, that is, it is a π-system, and it generates the product σ-algebra on $\mathcal{Z}^{\mathbb{N}_0}$. Thus, by Theorem 2.1.1, we have to show that $\mathcal{C} \subset \mathcal{D}$, and that $\mathcal{D}$ is a λ-system. First, we show that $\mathcal{C} \subset \mathcal{D}$. It holds:

$$\begin{aligned}
\psi(h,p)\Big\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K \times \prod_{k>K} \mathcal{Z}\Big\} &= \psi_{h,p}\Big\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K \times \prod_{k>K} \mathcal{Z}\Big\} \\
&= \left(\psi_{h,p} \circ \pi^{-1}_{\{0,\ldots,K\}}\right)\left\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K\right\} \\
&= \left(\mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma(h,p,\cdot)\right)\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K\} \,.
\end{aligned}$$

Since γ is a probability kernel from $\mathcal{H} \times \mathcal{P} \times \mathcal{Z}$ to $\mathcal{Z}$, by Lemma 2.1.6 it holds that $\mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma(h,p,\cdot)$ is a probability kernel from $\mathcal{H} \times \mathcal{P}$ to $\mathcal{Z}^{K+1}$. Therefore, the map $(h,p) \mapsto \mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma(h,p,\cdot)\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K\}$ is measurable, which implies that also the map $(h,p) \mapsto \psi(h,p)\left\{\mathsf{A}^0 \times \ldots \times \mathsf{A}^K \times \prod_{k>K} \mathcal{Z}\right\}$ is measurable. This shows that

$$\mathcal{C} \subset \mathcal{D} \,.$$

Now we show that $\mathcal{D}$ is a λ-system. Since $\mathcal{Z}^{\mathbb{N}_0} \in \mathcal{C} \subset \mathcal{D}$, we have $\mathcal{Z}^{\mathbb{N}_0} \in \mathcal{D}$. Hence, take $\mathsf{A}, \mathsf{B} \in \mathcal{D}$ with $\mathsf{A} \supset \mathsf{B}$. By definition of $\mathcal{D}$, we have that both $(h,p) \mapsto \psi(h,p)\{\mathsf{A}\}$ and $(h,p) \mapsto \psi(h,p)\{\mathsf{B}\}$ are measurable. Thus, this implies that the map $(h,p) \mapsto \psi(h,p)\{\mathsf{A} \setminus \mathsf{B}\} = \psi(h,p)\{\mathsf{A}\} - \psi(h,p)\{\mathsf{B}\}$ is measurable, where the (point-wise) equality follows from the fact that $\psi(h,p) = \psi_{h,p}$ is a probability measure for each $(h,p) \in \mathcal{H} \times \mathcal{P}$. Therefore, we have that $\mathsf{A} \setminus \mathsf{B} \in \mathcal{D}$, and $\mathcal{D}$ is closed under proper differences. Finally, take $\mathsf{A}_1, \mathsf{A}_2, \ldots \in \mathcal{D}$ with $\mathsf{A}_n \uparrow \mathsf{A}$. Then, since $\psi(h,p)$ is a measure for each $(h,p) \in \mathcal{H} \times \mathcal{P}$, we have the equality:

$$\psi(h,p)\{\mathsf{A}\} = \psi_{h,p}\{\mathsf{A}\} = \lim_{n\to\infty} \psi_{h,p}\{\mathsf{A}_n\} = \lim_{n\to\infty} \psi(h,p)\{\mathsf{A}_n\} \,.$$

By definition of $\mathcal{D}$, we have that $(h,p) \mapsto \psi(h,p)\{\mathsf{A}_n\}$ is measurable for each $n \in \mathbb{N}$. Thus, the map $(h,p) \mapsto \psi(h,p)\{\mathsf{A}\}$ is the pointwise limit of measurable functions, and therefore measurable itself. Hence, we have $\mathsf{A} \in \mathcal{D}$, such that $\mathcal{D}$ is a λ-system and, by applying Theorem 2.1.1, we get

$$\mathfrak{B}(\mathcal{Z})^{\otimes \mathbb{N}_0} = \sigma(\mathcal{C}) \subset \mathcal{D} \,,$$

that is, the map $(h,p) \mapsto \psi(h,p)\{\mathsf{A}\}$ is measurable for each $\mathsf{A} \in \mathfrak{B}(\mathcal{Z})^{\otimes \mathbb{N}_0}$. It follows that ψ is a probability kernel from $\mathcal{H} \times \mathcal{P}$ to $\mathcal{Z}^{\mathbb{N}_0}$. The uniqueness of ψ follows from the fact that $\psi_{h,p}$ is unique for every $(h,p) \in \mathcal{H} \times \mathcal{P}$, which concludes the proof for ψ. The statement for Ψ follows by the same arguments. □

The whole construction is summarized visually in Figure 4.6. Theorem 4.2.1 states that the distribution of the trajectory on $\mathcal{Z}^{\mathbb{N}_0}$ depends *measurably* on the parameters of the problem and the hyperparameters of the algorithm. This allows us to define a new probability space, which describes these three directly. However, before doing this, the next lemma states that the factorization of Γ actually extends to a factorization of Ψ,

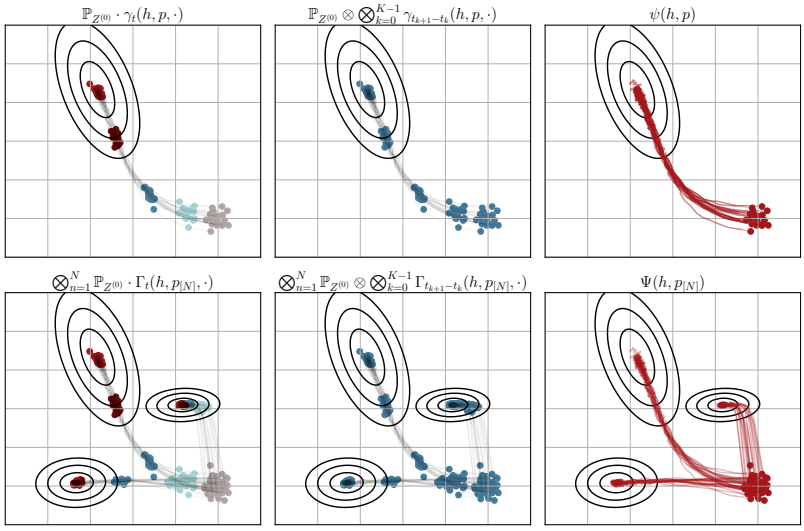

Figure 4.6: Visualization of the (joint) transition kernel. The upper row shows how the kernel $\gamma(h,p,\cdot)$ acts on the initial distribution: The iterative concatenation (upper left) transforms the initial distribution of $Z^{(0)}$ on $\mathfrak{X}$ (gray) into the distributions of $Z^{(t_1)}$ (light blue), $Z^{(t_2)}$ (dark blue), $Z^{(t_3)}$ (brown), and $Z^{(t_4)}$ (red). Similarly, the iterative product (upper middle) transforms the initial distribution on $\mathfrak{X}$ into a distribution on $\mathfrak{X}^5$, namely the joint distribution of $(Z^{(0)}, Z^{(t_1)}, ..., Z^{(t_4)})$. Then, this yields the unique distribution $\psi(h,p)$ on $\mathfrak{X}^{\mathbb{N}_0}$ (upper right) for the whole trajectory (red lines). The lower row shows basically the same thing for the joint transition kernel $\Gamma(h,p_{[N]},\cdot)$ on the space $\mathfrak{X}^N$.

	C^0	C^1	C^2	C^3	C^4	
A_1	B_1^0	B_1^1	B_1^2	B_1^3	B_1^4	...
A_2	B_2^0	B_2^1	B_2^2	B_2^3	B_2^4	...
A_3	B_3^0	B_3^1	B_3^2	B_3^3	B_3^4	...

Figure 4.7: Visual representation of the involved sets: Such a set can be expressed "row-wise" like $A = A_1 \times ... \times A_N$ or "column-wise" like $C^0 \times ... \times C^K \times \prod_{k>K} \mathfrak{X}^N$.

that is, the resulting processes $(Z_1, ..., Z_N)$ are conditionally independent. Here, for a lack of a better notation, we will also write cylinder sets $A \in \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$ with their corresponding coordinates $A = A_1 \times ... \times A_N$, where each $A_i \in \mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}$. This is justified by the fact that $\left(\mathfrak{X}^N\right)^{\mathbb{N}_0} \cong \left(\mathfrak{X}^{\mathbb{N}_0}\right)^N$ and $\mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0} \cong \left(\mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}\right)^{\otimes N}$. Further, $\mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$ is generated by cylinder sets of the form $\left(B_1^0 \times ... \times B_N^0\right) \times ... \times \left(B_1^K \times ... \times B_N^K\right) \times \prod_{k>K} \mathfrak{X}^N$, while $\left(\mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}\right)^{\otimes N}$ is generated by cylinder sets of the form $\left(B_1^0 \times ... \times B_1^K \times \prod_{k>K} \mathfrak{X}\right) \times ... \times \left(B_N^0 \times ... \times B_N^K \times \prod_{k>K} \mathfrak{X}\right)$, which are just a reordering of each other.

Lemma 4.2.2 *Suppose that Assumptions 4.0.1 and 4.0.3 hold. Then, for any $(h, p_{[N]}) \in \mathcal{H} \times \mathcal{P}^N$, and any set $A_1 \times ... \times A_N \in \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$, we have the following factorization:*

$$\Psi(h, p_{[N]})\{A_1 \times ... \times A_N\} = \prod_{i=1}^{N} \psi(h, p_n)\{A_n\} .$$

Thus, it holds that $\Psi(h, p_{[N]}) \cong \bigotimes_{n=1}^{N} \psi(h, p_n)$.

Proof. $\Psi(h, p_{[N]})$ is uniquely defined by its values on a $\cap$-stable generator of $\mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$. As stated above, such a generator is given by cylinder sets of the form:

$$\underbrace{\left(B_1^0 \times ... \times B_N^0\right)}_{=:C^0} \times ... \times \underbrace{\left(B_1^K \times ... \times B_N^K\right)}_{=:C^K} \times \prod_{k>K} \mathfrak{X}^N .$$

Thus, by denoting $J := \{0, ..., K\}$, we get by definition:

$$\Psi(h, p_{[N]})\left\{C^0 \times ... \times C^K \times \prod_{k>K} \mathfrak{X}^N\right\} = \left(\Psi_{h,p_{[N]}} \circ \pi^{-1}_{J,[N]}\right)\left\{C^0 \times ... \times C^K\right\}$$

$$= \left(\mathbb{P}_{Z^{(0)}}^{\otimes N} \otimes \bigotimes_{k=0}^{K-1} \Gamma(h, p_{[N]}, \cdot)\right)\left\{C^0 \times ... \times C^K\right\} .$$

This can equivalently be written as:

$$= \int_{\mathfrak{Z}^N} \mathbb{P}_{Z^{(0)}}^{\otimes N}(dz_{[N]}) \left(\delta_{z_{[N]}} \otimes \bigotimes_{k=0}^{K-1} \Gamma(h, p_{[N]}, \cdot)\right) \left\{\mathsf{C}^0 \times ... \times \mathsf{C}^K\right\} .$$

Thus, by Lemma 4.1.2 and Fubini's theorem, this is the same as:

$$\begin{aligned}
&= \int_{\mathfrak{Z}^N} \mathbb{P}_{Z^{(0)}}^{\otimes N}(dz_{[N]}) \prod_{i=1}^{N} \left(\delta_{z_n} \otimes \bigotimes_{k=0}^{K-1} \gamma(h, p_n, \cdot)\right) \left\{\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K\right\} \\
&= \prod_{i=1}^{N} \int_{\mathfrak{Z}} \mathbb{P}_{Z^{(0)}}(dz_n) \left(\delta_{z_n} \otimes \bigotimes_{k=0}^{K-1} \gamma(h, p_n, \cdot)\right) \left\{\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K\right\} \\
&= \prod_{i=1}^{N} \left(\mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma(h, p_n, \cdot)\right) \left\{\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K\right\} \\
&= \prod_{i=1}^{N} \left(\psi_{h,p_n} \circ \pi_J^{-1}\right) \left\{\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K\right\} = \prod_{i=1}^{N} \psi_{h,p_n} \left\{\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K \times \prod_{k>K} \mathfrak{Z}\right\} .
\end{aligned}$$

Since $\mathsf{B}_n^0 \times ... \times \mathsf{B}_n^K \times \prod_{k>K} \mathfrak{Z}$ is the n-th "coordinate" of the set $\mathsf{C}^0 \times .... \times \mathsf{C}^K \times \prod_{k>K} \mathfrak{Z}^N$, this shows that all finite-dimensional marginals of $\Psi(h, p_{[N]})$ coincide with the corresponding ones of $\otimes_{n=1}^N \psi(h, p_n)$. Thus, by the Kolmogorov Extension Theorem (Klenke, 2013, Thm. 14.36, p.295), that is, the uniqueness of the projective limit, we get that

$$\Psi(h, p_{[N]}) \cong \bigotimes_{n=1}^{N} \psi(h, p_n) .$$

Therefore, up to reordering, the measure $\Psi(h, p_{[N]})$ is given by the product of the measures $\psi(h, p_n)$, $n = 1, ..., N$. □

4.3 A New Probability Space

In the following, we will only be interested in the distribution of the trajectories on $\mathfrak{Z}^{\mathbb{N}_0}$ or $\left(\mathfrak{Z}^N\right)^{\mathbb{N}_0} \cong \left(\mathfrak{Z}^{\mathbb{N}_0}\right)^N$ respectively, and how they evolve with h and p. Hence, we define the measurable space $(\Omega, \mathfrak{A})$ as:

$$\Omega := \mathscr{H} \times \mathscr{P}^N \times \left(\mathfrak{Z}^N\right)^{\mathbb{N}_0} , \quad \mathfrak{A} := \mathfrak{B}(\Omega) ,$$

and endow it with the probability measure $\mathbb{P}$, given by:

$$\mathbb{P} := \left(\mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N}\right) \otimes \Psi .$$

Further, as before, we denote the canonical process by $X := (H, P_{[N]}, Z_{[N]})$, that is, $H \sim \mathbb{P}_H, P_1, ..., P_N \overset{iid}{\sim} \mathbb{P}_P$, and $Z_{[N]} := (Z_1, ..., Z_N) \sim (\mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N}) \cdot \Psi$.

Remark 4.3.1 (i) Since we define $\mathbb{P}$ through the probability kernel Ψ, it is assumed implicitly that Assumptions 4.0.1 and 4.0.3 do hold all the time, as they were needed for its construction.
(ii) Additionally to $P_{[N]}$, and depending on which notation seems more helpful, we will also denote the data set by $S := P_{[N]}$.

The following lemma summarizes results about regular versions of the

conditional distributions arising from the construction above. Additionally, it is also meant to fix the notation.

Lemma 4.3.1 *It holds that:*

(i) Ψ *is a regular version of the conditional distribution of* $Z_{[N]}$*, given* H *and* $P_{[N]}$*, that is, for* $\mathsf{A} \in \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$ *it holds that:*

$$\mathbb{P}_{Z_{[N]}|H,P_{[N]}}\{\mathsf{A}\} = \Psi(H, P_{[N]})\{\mathsf{A}\}, \quad \mathbb{P}_{(H,P_{[N]})}\text{–a.s.}$$

(ii) $\mathbb{P}_{P_{[N]}} \otimes \Psi$ *is a regular version of the conditional distribution of* $(P_{[N]}, Z_{[N]})$*, given* H*, that is, for* $\mathsf{B} \in \mathfrak{B}(\mathscr{P}) \otimes \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$ *it holds that:*

$$\mathbb{P}_{(P_{[N]},Z_{[N]})|H}\{\mathsf{B}\} = \left(\mathbb{P}_{P_{[N]}} \otimes \Psi(H, \cdot)\right)\{\mathsf{B}\}, \quad \mathbb{P}_H\text{–a.s.}$$

(iii) ψ *is a regular version of the conditional distribution of* Z_n*, given* H *and* P_n*, that is, for a set* $\mathsf{C} \in \mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}$ *it holds that:*

$$\mathbb{P}_{Z_n|H,P_n}\{\mathsf{C}\} = \psi(H, P_n)\{\mathsf{C}\}, \quad \mathbb{P}_{(H,P_n)}\text{–a.s.}$$

(iv) $\mathbb{P}_P \otimes \psi$ *is a regular version of the conditional distribution of* (P_n, Z_n)*, given* H*, that is, for a set* $\mathsf{D} \in \mathfrak{B}(\mathscr{P}) \otimes \mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}$ *it holds that:*

$$\begin{aligned}\mathbb{P}_{(P_n,Z_n)|H}\{\mathsf{D}\} &= \left(\mathbb{P}_{P_n} \otimes \psi(H, \cdot)\right)\{\mathsf{D}\} \\ &= \left(\mathbb{P}_P \otimes \psi(H, \cdot)\right)\{\mathsf{D}\}, \quad \mathbb{P}_H\text{–a.s.}\,.\end{aligned}$$

Proof. (i) We have to show that for any $\mathsf{B} \in \mathfrak{B}\left(\mathfrak{X}^N\right)^{\otimes \mathbb{N}_0}$ and $\mathsf{A} \in \mathfrak{B}\left(\mathscr{H} \times \mathscr{P}^N\right)$ it holds that:

$$\mathbb{E}_{(H,P_{[N]})}\left\{\mathbb{1}_{\mathsf{A}}\Psi(\cdot,\cdot)\{\mathsf{B}\}\right\} = \mathbb{P}_{(H,P_{[N]},Z_{[N]})}\{\mathsf{A} \times \mathsf{B}\}\,.$$

Since $\mathbb{P}_{(H,P_{[N]},Z_{[N]})} = \mathbb{P}$ (coordinate projections), this holds by construction of $\mathbb{P}$.

(ii) Since $\mathbb{P}_{(H,P_{[N]})} = \mathbb{P}_H \otimes \mathbb{P}_{P_{[N]}} = \mathbb{P}_H \otimes \mathbb{P}_P^{\otimes N}$, this follows from (i).

(iii) We have to show that for any $\mathsf{B} \in \mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}$ and $\mathsf{A} \in \mathfrak{B}(\mathscr{H} \times \mathscr{P})$ it holds that:

$$\mathbb{E}_{(H,P_n)}\left\{\mathbb{1}_{\mathsf{A}}\psi(\cdot,\cdot)\{\mathsf{B}\}\right\} = \mathbb{P}_{(H,P_n,Z_n)}\{\mathsf{A} \times \mathsf{B}\}\,.$$

Since $\mathfrak{B}(H \times P)$ is generated by the cylinder sets, it suffices to consider A of the form $\mathsf{A} = \mathsf{C} \times \mathsf{D}$. Here, one gets:

$$\begin{aligned}&\mathbb{E}_{(H,P_n)}\left\{\mathbb{1}_{\mathsf{C}\times\mathsf{D}}\psi(\cdot,\cdot)\{\mathsf{B}\}\right\} \\ &= \int_{\mathscr{H}} \mathbb{P}_H(dh)\, \mathbb{1}_{\mathsf{C}}(h) \int_{\mathscr{P}} \mathbb{P}_{P_n}(dp_n)\, \mathbb{1}_{\mathsf{D}}(p_n)\psi(h, p_n)\{\mathsf{B}\}\,.\end{aligned}$$

By inserting $1 = \prod_{i\neq n}^N \mathbb{1}_{\mathscr{P}}(p_i)\psi(h, p_i)\{\mathfrak{X}^{\mathbb{N}_0}\}$ and using Lemma 4.2.2, this is the same as:

$$= \mathbb{P}\left\{H \in \mathsf{C},\ P_n \in \mathsf{D},\ Z_n \in \mathsf{B}\right\}\,.$$

(iv) Since $\mathbb{P}_{(H,P_n)} = \mathbb{P}_H \otimes \mathbb{P}_{P_n} = \mathbb{P}_H \otimes \mathbb{P}_P$, this follows from (iii).

□

Remark 4.3.2 In the following, we denote the kernel $(h, \mathsf{A}) \mapsto \big(\mathbb{P}_P \otimes \psi(h, \cdot)\big)\{\mathsf{A}\}$ by $(h, \mathsf{A}) \mapsto \mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}$.

Corollary 4.3.2 *Given H and $P_{[N]}$, the processes $Z_{[N]} = (Z_1, \ldots, Z_N)$ are independent, and Z_i is independent of P_j for $i \neq j$. That is, for any set $\mathsf{A}_1 \times \ldots \times \mathsf{A}_N \in \big(\mathfrak{B}(\mathfrak{X})^{\otimes \mathbb{N}_0}\big)^{\otimes N}$ and any $(h, p_{[N]}) \in \mathcal{H} \times \mathcal{P}^N$ it holds that:*

$$\begin{aligned}&\mathbb{P}\left\{Z_{[N]} \in \mathsf{A}_1 \times \ldots \times \mathsf{A}_N \mid H = h, P_{[N]} = p_{[N]}\right\}\\ &= \prod_{i=1}^{N} \mathbb{P}\left\{Z_n \in \mathsf{A}_n \mid H = h, P_n = p_n\right\},\end{aligned}$$

that is, $\mathbb{P}_{Z_{[N]}|H,P_{[N]}} = \bigotimes_{n=1}^{N} \mathbb{P}_{Z_n|H,P_n}$.

Proof. This follows directly from Lemma 4.2.2 and Lemma 4.3.1. □

Corollary 4.3.3 *Let $f_1, \ldots, f_N : \mathfrak{X}^{\mathbb{N}_0} \to \mathbb{R}_{\geq 0}$ be measurable functions. Then it holds that:*

$$\mathbb{E}\Big\{\sum_{n=1}^{N} f_n(Z_n) \,\Big|\, H, P_{[N]}\Big\} = \sum_{n=1}^{N} \mathbb{E}\{f_n(Z_n) \mid H, P_n\}.$$

Proof. By definition, it holds that:

$$\mathbb{E}\Big\{\sum_{n=1}^{N} f_n(Z_n) \,\Big|\, H, P_{[N]}\Big\} = \int_{(\mathfrak{X}^{\mathbb{N}_0})^N} \sum_{n=1}^{N} f_n(\zeta_n)\, \Psi(H, P_{[N]})(d\zeta_{[N]}).$$

By Lemma 4.3.1, this is the same as:

$$= \sum_{n=1}^{N} \int_{(\mathfrak{X}^{\mathbb{N}_0})^N} f_n(\zeta_n)\, \Psi(H, P_{[N]})(d\zeta_{[N]}).$$

By Lemma 4.2.2 and Fubini's theorem, this is the same as:

$$\begin{aligned}&= \sum_{n=1}^{N} \int_{\mathfrak{X}^{\mathbb{N}_0}} f_n(\zeta_n)\, \psi(H, P_n)(d\zeta_n) \cdot \prod_{i \neq n}^{N} \underbrace{\int_{\mathfrak{X}^{\mathbb{N}_0}} \psi(H, P_i)(d\zeta_i)}_{=1}\\ &= \sum_{n=1}^{N} \int_{\mathfrak{X}^{\mathbb{N}_0}} f_n(\zeta_n)\, \psi(H, P_n)(d\zeta_n) = \sum_{n=1}^{N} \mathbb{E}\{f_n(Z_n) \mid H, P_n\}.\end{aligned}$$

where the last step follows from applying Lemma 4.3.1 again. □

4.3.1 Stopping the Algorithm

Ultimately, the algorithm is stopped at some point. Typically, this is the case as soon as some convergence criterion is met or as soon as the maximal computational budget is reached. Here, the set of points $z \in \mathfrak{X}$ for which $\mathcal{A}$ satisfies the convergence criterion can be represented

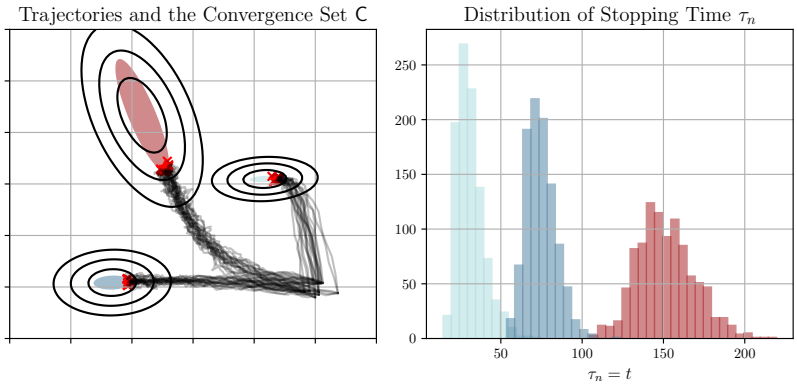

Figure 4.9: Visualization of the stopping time τ: In the left plot, the convergence set for each problem is shown as shaded region. As soon as a trajectory enters this region (red crosses), the algorithm is stopped. This yields the distribution of τ depending on the parameter p, as shown in the right plot.

as a subset $\tilde{\mathsf{C}} \subset \mathfrak{X}$. However, since we are considering parametric loss functions, we also have to use a parametric set $\mathsf{C} \subset \mathcal{P} \times \mathfrak{X}$:

Definition 4.3.1 *The* convergence set $\mathsf{C} \subset \mathcal{P} \times \mathfrak{X}$ *is defined as:*

$$\mathsf{C} := \{(p, z) \in \mathcal{P} \times \mathfrak{X} \ : \ z \text{ satisfies the convergence criterion for } \ell(\cdot, p)\} .$$

Since the convergence set is defined in terms of a not further specified convergence criterion, we need the following assumption:

Assumption 4.3.1 *The convergence set* C *is measurable.*

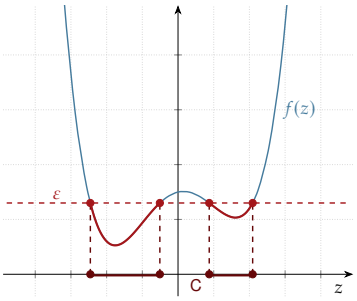

Figure 4.8: The convergence set C for the criterion $f(z) < \varepsilon$.

Example 4.3.1
(i) For convergence in terms of the loss function, one could use $\mathsf{C} := \{(p, z) \in \mathcal{P} \times \mathfrak{X} \ : \ \ell(\pi_{\mathfrak{X}}(z), p) \leq \varepsilon\}$, which is measurable, since ℓ is measurable.
(ii) If $\ell(\cdot, p)$ has a unique minimizer x_p^*, convergence in terms of the iterates could be written as $\mathsf{C} := \{(p, z) \in \mathcal{P} \times \mathfrak{X} \ : \ d(\pi_{\mathfrak{X}}(z), x_p^*) \leq \varepsilon\}$. If $p \mapsto x_p^*$ is measurable, C is measurable, because the distance d is continuous.
(iii) For convergence to a stationary point, one could use $\mathsf{C} := \{(p, z) \in \mathcal{P} \times \mathfrak{X} \ : \ \|\nabla_1 \ell(\pi_{\mathfrak{X}}(z), p)\| \leq \varepsilon\}$. For example, if $\nabla_1 \ell$ is (jointly) continuous, C is measurable.
(iv) In the previous examples, the convergence set could have been also defined as a subset of $\mathfrak{X} \times \mathcal{P}$. In general, however, this might be insufficient as this example highlights: If the update includes a momentum term $m^{(t)} = x^{(t)} - x^{(t-1)}$ as, for example, in the heavy-ball algorithm, one could define the convergence set also as $\mathsf{C} := \{(p, z) \in \mathcal{P} \times \mathfrak{X} \ : \ \|\nabla_1 \ell(\pi_{\mathfrak{X}}(z), p)\| \leq \varepsilon \text{ and } \|m^{(t)}\| \leq \varepsilon\}$.

Based on this, we introduce the random times $\tau_{\max} := t_{\max} \in \mathbb{N}$ and $\tau_{\text{conv},n} := \inf\{t \in \mathbb{N}_0 \ : \ (P_n, Z_n^{(t)}) \in \mathsf{C}\}$, $n = 1, \ldots, N$. Then, we combine them into the random times τ_n:

$$\tau_n := \tau_{\max} \wedge \tau_{\text{conv},n} := \min\{\tau_{\max}, \tau_{\text{conv},n}\}, \quad n = 1, \ldots, N .$$

This idea is visualized in Figure 4.9 and the following lemma shows that the random times τ_n are indeed $\mathfrak{F}_n$*-stopping times*, that is, $\{\tau_n \leq t\} \in \mathfrak{F}_n^{(t)}$ for each $t \in \mathbb{N}_0$, $n = 1, \ldots, N$. Intuitively, this tells us that, at time t, the information collected in $\mathfrak{F}_n^{(t)}$ is indeed enough to decide whether the algorithm did reach the convergence set or not:

Proposition 4.3.4 *Suppose that Assumption 4.3.1 holds. Then, for each $n \in \{1, \dots, N\}$, τ_n is a stopping time w.r.t. the filtration $\mathfrak{F}_n = (\mathfrak{F}_n^{(t)})_{t\in\mathbb{N}_0}$, where $\mathfrak{F}_n^{(t)} := \sigma(P_n, Z_n^{(0)}, \dots, Z_n^{(t)})$.*

Proof. If $\tau_{\max}$ and $\tau_{\text{conv},n}$ are $\mathfrak{F}_n$-optional, so is τ_n by Lemma 2.1.13. By the same result, we have that $\tau_{\max}$ is $\mathfrak{F}_n$-optional, because it is constant. Hence, we only have to show that $\tau_{\text{conv},n}$ is $\mathfrak{F}_n$-optional. By definition, $\tau_{\text{conv},n}$ is a so-called *hitting time* for the process $U_n = (U_n^{(t)})_{t\in\mathbb{N}_0}$ given by $U_n^{(t)} := (P_n, Z_n^{(t)})$, that is, it is of the form

$$\tau_{\text{conv},n} = \inf\left\{t \in \mathbb{N}_0 \,:\, U_n^{(t)} \in \mathsf{C}\right\}.$$

Further, we have that:

$$\begin{aligned}\mathfrak{F}_n^{(t)} &= \sigma(P_n, Z_n^{(0)}, \dots, Z_n^{(t)}) \\ &= \sigma((P_n, Z_n^{(0)}), \dots, (P_n, Z_n^{(t)})) = \sigma(U_n^{(0)}, \dots, U_n^{(t)}),\end{aligned}$$

that is, U_n is adapted to $\mathfrak{F}_n$. Therefore, the conclusion follows from the second part of Lemma 2.1.13. □

4.4 Generalization Results

In this section, we use the previously constructed probability space to derive the generalization results. To this end, we first give a PAC-Bayesian generalization result for the case of a bounded parametric function defined on the space of trajectories $\mathfrak{X}^{\mathbb{N}_0}$ in Theorem 4.4.4. Subsequently, we specialize it to the case of the convergence time in Corollary 4.4.5 and to the case of the convergence rate in Corollary 4.4.6. Both results are immediate consequences of Theorem 4.4.4. Here, we need the following well-known lemma[1] due to Hoeffding, which can be found, for example, in the book by Boucheron et al. (2013, Lemma 2.2 and p.33) or in the book by Massart (2007, Lemma 2.6):

1: Not to be confused with *Hoeffding's inequality*, which is a corollary of this.

Lemma 4.4.1 (Hoeffding's lemma) *Let U be a centered random variable taking values in the interval $[a, b]$. Then, for any $\lambda \in \mathbb{R}$,*

$$\mathbb{E}\left\{\exp(\lambda U)\right\} \leq \exp\left(\frac{\lambda^2(b-a)^2}{8}\right).$$

This allows to derive the following statement, from which the PAC-Bayesian generalization theorem follows immediately.

Lemma 4.4.2 *Let $f : \mathscr{P} \times \mathfrak{X}^{\mathbb{N}_0} \to [0, +\infty)$ be a measurable function that is bounded from above by $f_{\max} \in \mathbb{R}$. Then, for every $\lambda \in \mathbb{R}$ it holds that:*

$$\mathbb{E}\left\{\exp\left(\lambda\left(\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\{f(P_n, Z_n) \mid H, P_n\} - \mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\right\} \leq \exp\left(\frac{\lambda^2}{2N} f_{\max}^2\right).$$

Proof. By Corollary 4.3.3 it holds that:

$$\mathbb{E}\left\{\exp\left(\lambda\left(\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\{f(P_n,Z_n)\mid H,P_n\}-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\right\}$$
$$=\mathbb{E}\left\{\exp\left(\lambda\left(\mathbb{E}\left\{\frac{1}{N}\sum_{n=1}^{N}f(P_n,Z_n)\,\middle|\,H,P_{[N]}\right\}-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\right\}.$$

Since the other terms are constant w.r.t. Ψ, this is the same as:

$$=\mathbb{E}\left\{\exp\left(\mathbb{E}\left\{\lambda\left(\frac{1}{N}\sum_{n=1}^{N}f(P_n,Z_n)-\mathbb{E}_{(P,Z)|H}\{f\}\right)\,\middle|\,H,P_{[N]}\right\}\right)\right\}.$$

By Jensen's inequality, this can be bounded by:

$$\leq\mathbb{E}\left\{\mathbb{E}\left\{\exp\left(\lambda\left(\frac{1}{N}\sum_{n=1}^{N}f(P_n,Z_n)-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\,\middle|\,H,P_{[N]}\right\}\right\}.$$

By the properties of the expectation and the exponential function, this is the same as:

$$=\mathbb{E}\left\{\mathbb{E}\left\{\exp\left(\lambda\left(\frac{1}{N}\sum_{n=1}^{N}f(P_n,Z_n)-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\,\middle|\,H\right\}\right\}$$
$$=\mathbb{E}\left\{\mathbb{E}\left\{\prod_{n=1}^{N}\exp\left(\frac{\lambda}{N}\left(f(P_n,Z_n)-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\,\middle|\,H\right\}\right\}.$$

By Lemma 4.2.2, the independence of the parameters $P_1,\ldots,P_N$, and Fubini's theorem, this is the same as:

$$=\mathbb{E}\left\{\prod_{n=1}^{N}\mathbb{E}\left\{\exp\left(\frac{\lambda}{N}\left(f(P_n,Z_n)-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\,\middle|\,H\right\}\right\}$$
$$=\mathbb{E}\left\{\prod_{n=1}^{N}\mathbb{E}_{(P_n,Z_n)|H}\left\{\exp\left(\frac{\lambda}{N}\left(f-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\right\}\right\}.$$

Since the parameters $P_1,\ldots,P_N$ are i.i.d., by Lemma 4.4.1, this can be bounded by:

$$=\mathbb{E}\left\{\prod_{n=1}^{N}\mathbb{E}_{(P,Z)|H}\left\{\exp\left(\frac{\lambda}{N}\left(f-\mathbb{E}_{(P,Z)|H}\{f\}\right)\right)\right\}\right\}$$
$$\leq\mathbb{E}\left\{\prod_{n=1}^{N}\exp\left(\frac{\lambda^2}{8N^2}(2f_{\max})^2\right)\right\}=\exp\left(\frac{\lambda^2}{2N}f_{\max}^2\right).$$

□

For the next result we need the following variational formulation due to Donsker et al. (1975):

Lemma 4.4.3 *Let μ be a probability measure on $\mathcal{U}$. Then, for any measurable*

and bounded function $f : \mathcal{U} \to \mathbb{R}$, it holds that:

$$\log\left(\int_{\mathcal{U}} \exp(f(u))\,\mu(du)\right) = \sup_{\nu \in \mathcal{M}_1(\mu)} \left\{\int_{\mathcal{U}} f(u)\,\nu(du) - D_{\mathrm{KL}}(\nu \,\|\, \mu)\right\}.$$

Now, please recall that the map $h \mapsto \mathbb{E}_{(P,Z)|H=h}\{f\} = \left(\mathbb{P}_P \otimes \psi(h,\cdot)\right)\{f\}$ is indeed a measurable and bounded function on $\mathcal{H}$, that is, it can be integrated w.r.t. an arbitrary (probability) measure $\mathbb{Q} \in \mathcal{M}_1(\mathcal{H})$. Furthermore, note that, by projecting from $\mathfrak{Z}^{\mathbb{N}_0}$ onto the corresponding coordinate, this result can be applied to single fixed iterates, too.

Theorem 4.4.4 *Let $f : \mathcal{P} \times \mathfrak{Z}^{\mathbb{N}_0} \to [0, +\infty)$ be a measurable function that is bounded from above by $f_{\max} \in \mathbb{R}$. Then, for every $\lambda \in (0, +\infty)$ and $\varepsilon > 0$ it holds that:*

$$\mathbb{P}_S\left\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) \,:\, \mathbb{Q}\left\{\mathbb{E}_{(P,Z)|H}\{f\}\right\} \leq \frac{1}{N}\sum_{n=1}^{N} \mathbb{Q}\left\{\mathbb{E}_{(P_n,Z_n)|H,P_n}\{f\}\right\} + \frac{D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \frac{\lambda^2}{2N} f_{\max}^2 - \log(\varepsilon)}{\lambda}\right\} \geq 1 - \varepsilon\,.$$

Proof. Abbreviate $\bar{f} := \mathbb{E}_{(P,Z)|H}\{f\}$ and $\hat{f} := \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}_{(P_n,Z_n)|H,P_n}\{f\}$. Applying Lemma 4.4.2 together with Lemma 3.3.1, we get that:

$$\mathbb{E}_{(S,H)}\left\{\exp\left(\lambda(\bar{f} - \hat{f}) - \frac{\lambda^2}{2N} f_{\max}^2\right)\right\} \leq 1\,.$$

By Fubini's theorem applied to $\mathbb{P}_{(S,H)} = \mathbb{P}_S \otimes \mathbb{P}_H$, as well as the variational formulation of Donsker-Varadhan and the linearity of the integral, this is the same as:

$$\mathbb{E}_S\left\{\exp\left(\sup_{\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)} \lambda\left(\mathbb{Q}\{\bar{f}\} - \mathbb{Q}\{\hat{f}\}\right) - D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) - \frac{\lambda^2}{2N} f_{\max}^2\right)\right\} \leq 1\,.$$

Then, for any $c \in \mathbb{R}$ we get from Markov's inequality:

$$\mathbb{P}_S\left\{\sup_{\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)} \lambda\left(\mathbb{Q}\{\bar{f}\} - \mathbb{Q}\{\hat{f}\}\right) - D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) - \frac{\lambda^2}{2N} f_{\max}^2 \geq c\right\} \leq \exp(-c)\,.$$

Using $c = \log(1/\varepsilon)$ yields:

$$\mathbb{P}_S\left\{\sup_{\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)} \lambda\left(\mathbb{Q}\{\bar{f}\} - \mathbb{Q}\{\hat{f}\}\right) - D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) - \frac{\lambda^2}{2N} f_{\max}^2 \geq \log\left(\frac{1}{\varepsilon}\right)\right\} \leq \varepsilon\,.$$

Restricting to $\lambda > 0$, reformulating, and taking the complementary event yields the result. □

4.4.1 Guarantees for the Convergence Time

Please recall the definition of $\tau_n = \tau_{\mathrm{conv},n} \wedge \tau_{\max}$. Then, note that $\tau_{\mathrm{conv},n}$ can be written as $\tilde{\tau} \circ (P_n, Z_n)$, where $\tilde{\tau} : \mathcal{P} \times \mathfrak{Z}^{\mathbb{N}_0} \to \mathbb{N}_0 \cup \{+\infty\}$ is given by:

$$\begin{aligned}(p, (z^{(t)})_{t \in \mathbb{N}_0}) &\mapsto \inf\{k \in \mathbb{N}_0 \,:\, (p, z^{(k)}) \in \mathsf{C}\} \\ &= \inf_{k \in \mathbb{N}_0} k \cdot (1 + \iota_{\mathsf{C}}(p, z^{(k)}))\,.\end{aligned}$$

Since C is measurable by Assumption 4.3.1, and ι_{C} only takes the values $\{0, +\infty\}$, the map $(p, (z^{(t)})_{t\in\mathbb{N}_0}) \mapsto k \cdot (1 + \iota_{\mathsf{C}}(p, z^{(k)}))$ is measurable. Thus, $\tilde{\tau}$ is measurable as infimum of countably many measurable functions. Therefore, the map $T : \mathcal{P} \times \mathcal{X}^{\mathbb{N}_0} \to \mathbb{N}_0$, defined through

$$T(p, (z^{(t)})_{t\in\mathbb{N}_0}) := t_{\max} \wedge \tilde{\tau}(p, (z^{(t)})_{t\in\mathbb{N}_0}),$$

is measurable[2] and bounded by $t_{\max}$, and we can write τ_n as $T \circ (P_n, Z_n)$. Hence, by defining the average expected convergence time as $\bar{\tau} := \mathbb{E}_{(P,Z)|H}\{T\}$, we get the following result:

2: T is measurable w.r.t. $\mathfrak{B}(\mathcal{P}) \otimes \mathfrak{B}(\mathcal{X})^{\otimes\mathbb{N}_0}$ and $\mathfrak{B}([0, +\infty))$, because for $A \in \mathfrak{B}([0, +\infty))$ we have $T^{-1}(\mathsf{A}) = T^{-1}(\mathsf{A} \cap \mathbb{N}_0) \in \mathfrak{B}(\mathcal{P}) \otimes \mathfrak{B}(\mathcal{X})^{\otimes\mathbb{N}_0}$.

Corollary 4.4.5 *Suppose that Assumption 4.3.1 holds. Then, for every $\lambda \in (0, +\infty)$ and $\varepsilon > 0$ it holds that:*

$$\mathbb{P}_S\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) \,:\, \mathbb{Q}\{\bar{\tau}\} \le \frac{1}{N}\sum_{n=1}^{N} \mathbb{Q}\{\mathbb{E}\{\tau_n \mid H, P_n\}\} + \frac{D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \frac{\lambda^2}{2N} t_{\max}^2 - \log(\varepsilon)}{\lambda}\Big\} \ge 1 - \varepsilon\,.$$

Proof. Since T is measurable and bounded by $t_{\max}$, we can apply Theorem 4.4.4 with $f = T$ and $f_{\max} = t_{\max}$. By noting that $\mathbb{E}\{\tau_n \mid H, P_n\} = \mathbb{E}_{(P_n,Z_n)|H,P_n}\{T\}$, this directly yields the result. □

4.4.2 Guarantees for the Convergence Rate

The convergence rate of an algorithm describes how fast a given residual error, for example the loss, decreases to zero and, in the case of a linear rate of convergence, this can be summarized in the corresponding contraction-factor. Assuming that $\ell^*(\cdot, p) = 0$, in each iteration the ratio of the new and the previous loss is bounded from above by some constant $\rho < 1$:

$$\frac{\ell(\pi_{\mathcal{X}}(z^{(t+1)}), p)}{\ell(\pi_{\mathcal{X}}(z^{(t)}), p)} \le \rho < 1\,.$$

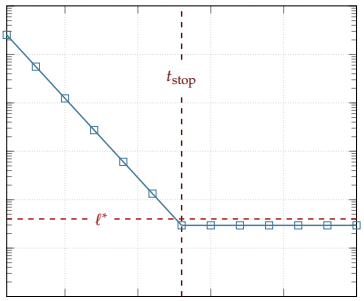

Figure 4.10: To estimate the (linear) rate correctly, we have to stop as soon as the algorithm does: In this hypothetical example, the algorithm gets stopped as soon as it reaches a loss $\le \ell^*$.

Applying this reasoning iteratively, after T steps the loss has contracted by at least

$$\frac{\ell(\pi_{\mathcal{X}}(z^{(T)}), p)}{\ell(\pi_{\mathcal{X}}(z^{(0)}), p)} = \prod_{t=0}^{T-1} \frac{\ell(\pi_{\mathcal{X}}(z^{(t+1)}), p)}{\ell(\pi_{\mathcal{X}}(z^{(t)}), p)} \le \rho^T\,.$$

Hence, to estimate ρ, we need to consider the term $\left(\frac{\ell(\pi_{\mathcal{X}}(z^{(T)}),p)}{\ell(\pi_{\mathcal{X}}(z^{(0)}),p)}\right)^{\frac{1}{T}}$. However, we also have to take care of the fact that the algorithm gets stopped as soon as it hits the convergence set, because we have $\frac{\ell(\pi_{\mathcal{X}}(z^{(t+1)}),p)}{\ell(\pi_{\mathcal{X}}(z^{(t)}),p)} = 1$ in this case, which could falsify the estimation: Assume that the algorithm contracts the loss exactly with a rate of ρ until it gets stopped at time t_{stop} and we estimate ρ with the loss at time $t_{\max} = t_{\text{stop}} + t_{\text{add}}$. Then our estimate for ρ is given by:

$$\left(\frac{\ell(\pi_{\mathcal{X}}(z^{(t_{\max})}), p)}{\ell(\pi_{\mathcal{X}}(z^{(0)}), p)}\right)^{\frac{1}{t_{\max}}} = \left(\frac{\ell(\pi_{\mathcal{X}}(z^{(t_{\text{stop}})}), p)}{\ell(\pi_{\mathcal{X}}(z^{(0)}), p)}\right)^{\frac{1}{t_{\text{stop}}+t_{\text{add}}}} = \rho^{\frac{t_{\text{stop}}}{t_{\text{stop}}+t_{\text{add}}}}\,.$$

Thus, to estimate ρ correctly we have to have $t_{\text{add}} = 0$. Otherwise, we will inevitably overestimate it. This motivates the following definition:

Definition 4.4.1 *The* contraction function *is defined as:*

$$c : \mathcal{P}\times\mathcal{Z}\times\mathcal{Z} \to \mathbb{R}_{\geq 0},\ (p, z_1, z_2) \mapsto \frac{\ell(\pi_{\mathcal{X}}(z_1), p)}{\ell(\pi_{\mathcal{X}}(z_2), p)} \cdot \mathbb{1}\{\ell(\pi_{\mathcal{X}}(z_2), p) > 0\}\,.$$

Abbreviate $T := T(p, (z^{(t)})_{t\in\mathbb{N}_0})$. *Then, the* rate function *is defined as:*

$$r : \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0} \to \mathbb{R}_{\geq 0}, (p, (z^{(t)})_{t\in\mathbb{N}_0}) \mapsto (c(p, z^{(T)}, z^{(0)}))^{\frac{1}{T}} \cdot \mathbb{1}\{T \geq 1\}\,,$$

and the expected rate function *is defined as* $\bar{r} := \mathbb{E}_{(P,Z)|H}\{r\}$. *Similarly, for some* $r_{\max} \in \mathbb{R}_{\geq 0}$, *we define the* bounded rate function *and* expected bounded rate function *as* $r_b := r \cdot \mathbb{1}\{r \leq r_{\max}\}$, *and* $\bar{r}_b = \mathbb{E}_{(P,Z)|H}\{r_b\}$.

T was defined as $T(p, (z^{(t)})_{t\in\mathbb{N}_0}) := t_{\max} \wedge \tilde{\tau}(p, (z^{(t)})_{t\in\mathbb{N}_0})$

Remark 4.4.1 (i) If $\ell_p^* > 0$ and one has access to it during training, one would change the definition to

$$\frac{\ell(\pi_{\mathcal{X}}(z_1), p) - \ell_p^*}{\ell(\pi_{\mathcal{X}}(z_2), p) - \ell_p^*} \cdot \mathbb{1}\{\ell(\pi_{\mathcal{X}}(z_2), p) - \ell_p^* > 0\}\,.$$

Empirically, however, training with just the ratio of consecutive losses works similarly good, as long as ℓ_p^* is not too large.

(ii) Note that, in contrast to defining the rate function in terms of the supremum of the contraction function over the iterations, this definition is applicable in the stochastic case, too, as it provides the *discounted total contraction* over all T iterations.

Since ℓ is measurable, we have that c is measurable. Further, since T is measurable and only takes countably many values, we get that $z^{(T)}$ is measurable[3]. Therefore, also r is measurable. Hence, r_b is measurable and bounded, and we get the following result:

3: Basically, we use that $\{z^{(T)} \in \mathsf{A}\} = \bigcup_{t\in\mathbb{N}_0}\{z^{(t)} \in \mathsf{A},\ T = t\}$.

Corollary 4.4.6 *Suppose that Assumption 4.3.1 holds. Then, for every* $\lambda \in (0, +\infty)$ *and* $\varepsilon > 0$ *it holds that:*

$$\mathbb{P}_S\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)\ :\ \mathbb{Q}\{\bar{r}_b\} \leq \frac{1}{N}\sum_{n=1}^{N} \mathbb{Q}\left\{\mathbb{E}_{(P_n,Z_n)|H,P_n}\{r_b\}\right\} + \frac{D_{\mathrm{KL}}(\mathbb{Q} \,\|\, \mathbb{P}_H) + \frac{\lambda^2}{2N} r_{\max}^2 - \log(\varepsilon)}{\lambda}\Big\} \geq 1 - \varepsilon\,.$$

Proof. Again, this follows directly from Theorem 4.4.4 with $f = r_b$ and $f_{\max} = r_{\max}$. □

4.4.3 Properties of the Trajectory

The last theoretical result concerns the probability to observe a trajectory that obeys a certain property, for example, to converge with at least a rate of $r \leq r_{\max}$. Such properties can be encoded in a measurable set $\mathsf{A} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$, and we have to consider the function $f := \mathbb{1}_{\mathsf{A}}$. Since $\mathbb{1}_{\mathsf{A}}$ is measurable and bounded, one could directly apply the results from above. Yet, in this case, one can actually compute the integral in closed-form

"Probability theory has a right and a left hand. On the right is the rigorous foundational work using the tools of measure theory. The left hand 'thinks probabilistically', reduces problems to gambling situations, coin-tossing, motions of a physical particle." - *Leo Breiman*

and get tighter results. Since $\mathbb{1}_{\mathsf{A}}$ only takes values in $\{0,1\}$, that is, it can serve as a Bernoulli random variable, the following function will be useful:

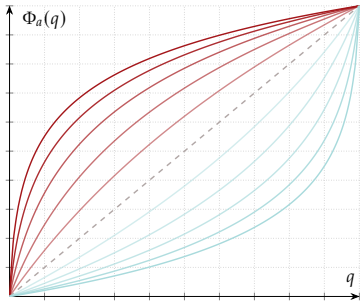

Figure 4.11: The function Φ_a for $a \in \{-5,-4,\ldots,-1,1,\ldots,4,5\}$.

Lemma 4.4.7 (Catoni (2007)) *Define the function*

$$\Phi_a(q) := -\frac{1}{a}\log\left(1-\left[1-\exp(-a)\right]q\right) .$$

Then it holds that:

(i) Φ_a *is an increasing one-to-one mapping of the unit-interval onto itself,*
(ii) Φ_a *is strictly convex for* $a>0$ *and strictly concave for* $a<0$,
(iii) Φ_a^{-1} *is given by:* $\Phi_a^{-1}(q) := \frac{1-\exp(-aq)}{1-\exp(-a)}$.

Proof. By validating $\Phi_a \circ \Phi_a^{-1} = id = \Phi_a^{-1} \circ \Phi_a$ with the formula given in (iii), one finds that Φ_a is one-to-one, which at the same time also shows (iii). The first derivative of Φ_a is given by:

$$\frac{\partial}{\partial q}\Phi_a(q) = \frac{1}{a}\frac{1-\exp(-a)}{1-[1-\exp(-a)]q} .$$

If $a>0$, we have that $\frac{1}{a}>0$ and $0<1-\exp(-a)<1$, such that $1-(1-\exp(-a))q>0$, which shows that $\frac{\partial}{\partial q}\Phi_a(q)>0$. Conversely, if $a<0$, we have that $\frac{1}{a}<0$ and $1-\exp(-a)<0$, such that $1-(1-\exp(-a))q>0$, which shows that $\frac{\partial}{\partial q}\Phi_a(q)>0$. Thus, Φ_a is increasing. Similarly, the second derivative of Φ_a is given by

$$\frac{\partial^2}{\partial^2 q}\Phi_a(q) = \frac{1}{a}\left(\frac{1-\exp(-a)}{1-[1-\exp(-a)]q}\right)^2 ,$$

which is strictly negative/positive whenever a is and thus shows (ii). □

Using the function Φ_a, we get the following compact representation:

Lemma 4.4.8 *Let* $\mathsf{A} \subset \mathcal{P}\times\mathcal{Z}^{\mathbb{N}_0}$ *be measurable. Then, for any* $\lambda\in\mathbb{R}$, *it holds that:*

$$\mathbb{E}\left\{\exp\left(-\frac{\lambda}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right\} = \mathbb{E}\left\{\exp(-\lambda\Phi_{\frac{1}{N}}(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}))\right\} .$$

Proof. By the same arguments as before, we get from Lemma 4.3.1:

$$\mathbb{E}\left\{\exp\left(-\frac{\lambda}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right\} = \mathbb{E}_H\left\{\left(\mathbb{E}_{(P,Z)|H}\left\{\exp\left(-\frac{\lambda}{N}\mathbb{1}_{\mathsf{A}}\right)\right\}\right)^N\right\} .$$

Then, for any $h\in\mathcal{H}$, the inner integral is given by:

$$\mathbb{E}_{(P,Z)|H=h}\left\{\exp\left(-\frac{\lambda}{N}\mathbb{1}_{\mathsf{A}}\right)\right\} = \mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}^c\} + \exp\left(-\frac{\lambda}{N}\right)\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\} .$$

Using the properties of a probability measure and rearranging yields:

$$= 1 - \mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\} + \exp\left(-\frac{\lambda}{N}\right)\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}$$
$$= 1 - \left(1 - \exp\left(-\frac{\lambda}{N}\right)\right)\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}\,.$$

Further, it holds:

$$\left(1 - \left(1 - \exp\left(-\frac{\lambda}{N}\right)\right)\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}\right)^N$$
$$= \exp\left(N \log\left(1 - \left(1 - \exp\left(-\frac{\lambda}{N}\right)\right)\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}\right)\right)$$
$$= \exp(-\lambda \Phi_{\frac{\lambda}{N}}(\mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}))$$

Inserting this into the first equation concludes the proof. □

This yields the following corollary, which is needed to apply the PAC-Bayesian argument:

Corollary 4.4.9 *Let $\mathsf{A} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ be measurable. Then, for any $\lambda \in \mathbb{R}$ it holds that:*

$$\mathbb{E}_S\left\{\mathbb{E}_H\left\{\exp\left(\lambda\left[\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}\}\right]\right)\right\}\right\} \leq 1\,.$$

Proof. By Fubini's theorem we have the following equality:

$$\mathbb{E}_S\left\{\mathbb{E}_H\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}\}\right)\right)\right\}\right\}$$
$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}\}\right)\right)\right\}$$
$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\{\mathbb{1}_{\mathsf{A}}(P_n,Z_n) \mid H,P_n\}\right)\right)\right\}.$$

Please recall that H, P, etc. are defined as the corresponding projections on Ω, such that we do not have to distinguish between conditional probabilities and conditional distributions.

By Corollary 4.3.3, this is the same as:

$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \mathbb{E}\left\{\frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\,\middle|\,H,P_{[N]}\right\}\right)\right)\right\}$$
$$= \mathbb{E}\left\{\exp\left(\mathbb{E}\left\{\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\,\middle|\,H,P_{[N]}\right\}\right)\right\}.$$

By Jensen's inequality, this can be bounded by:

$$\leq \mathbb{E}\left\{\mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right)\,\middle|\,H,P_{[N]}\right\}\right\}$$
$$= \mathbb{E}\left\{\mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right) - \frac{1}{N}\sum_{n=1}^{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right)\,\middle|\,H\right\}\right\}.$$

Since the first term is constant given H, this is the same as:

$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right)\right)\mathbb{E}\left\{\prod_{i=1}^{N}\exp\left(-\frac{\lambda}{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right)\,\middle|\,H\right\}\right\}.$$

By conditional independence given H, and the fact that $(P_1,Z_1),\ldots,(P_N,Z_N)$ have the same conditional distribution (Lemma 4.3.1), this is the same as:

$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right)\right)\prod_{i=1}^{N}\mathbb{E}\left\{\exp\left(-\frac{\lambda}{N}\mathbb{1}_{\mathsf{A}}(P_n,Z_n)\right)\right)\,\middle|\,H\right\}\right\}$$

$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right)\right)\left(\mathbb{E}\left\{\exp\left(-\frac{\lambda}{N}\mathbb{1}_{\mathsf{A}}(P,Z)\right)\right)\,\middle|\,H\right\}\right)^{N}\right\}.$$

As before, we find that this is the same as:

$$= \mathbb{E}\left\{\exp\left(\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right)\right)\exp\left(-\lambda\left(\Phi_{\frac{\lambda}{N}}\left(\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\right)\right)\right\} = 1\,.$$

□

Thus, this yields the following PAC-Bayesian generalization bound.

O. Catoni proved this PAC-Bayesian generalization bound for Bernoulli random variables. We arrive at the same result for the properties of the trajectories, since they are formulated as indicator functions, which can be considered as Bernoulli random variables.

Theorem 4.4.10 (Catoni (2007)) *Let $\mathsf{A} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ be measurable. Then, for $\lambda \in (0,+\infty)$, it holds that:*

$$\mathbb{P}_S\left\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)\,:\,\mathbb{Q}\{\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}\} \leq \Phi^{-1}_{\frac{\lambda}{N}}\left(\frac{1}{N}\sum_{n=1}^{N}\mathbb{Q}\left\{\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}\}\right\} + \frac{D_{\mathrm{KL}}(\mathbb{Q}\parallel\mathbb{P}_H) + \log\left(\frac{1}{\varepsilon}\right)}{\lambda}\right)\right\} \geq 1-\varepsilon\,.$$

Proof. Abbreviate $\mathbb{p} := \mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ and $\hat{\mathbb{p}} := \frac{1}{N}\sum_{n=1}^{N}\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}\}$. Then, by Corollay 4.4.9 it holds that:

$$\mathbb{E}_S\left\{\mathbb{E}_H\left\{\exp\left(\lambda(\Phi_{\frac{\lambda}{N}}\circ\mathbb{p} - \hat{\mathbb{p}}(\cdot,s))\right)\right\}\Big|_{s=S}\right\} \leq 1\,.$$

Therefore, by the Donsker-Varadhan variational formulation, we have:

$$1 \geq \mathbb{E}_S\left\{\exp\left(\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)}\lambda\mathbb{Q}\{\Phi_{\frac{\lambda}{N}}\circ\mathbb{p} - \hat{\mathbb{p}}(\cdot,s)\}|_{s=S} - D_{\mathrm{KL}}(\mathbb{Q}\parallel\mathbb{P}_H)\right)\right\}$$

$$= \mathbb{E}_S\left\{\exp\left(\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)}\lambda\left(\mathbb{Q}\{\Phi_{\frac{\lambda}{N}}\circ\mathbb{p}\} - \mathbb{Q}\{\hat{\mathbb{p}}(\cdot,s)\}|_{s=S}\right) - D_{\mathrm{KL}}(\mathbb{Q}\parallel\mathbb{P}_H)\right)\right\}.$$

Since $\lambda > 0$, Φ is convex by Lemma 4.4.7. Thus, applying Jensen's inequality yields:

$$\geq \mathbb{E}_S\left\{\exp\left(\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)}\lambda\left(\Phi_{\frac{\lambda}{N}}(\mathbb{Q}\{\mathbb{p}\}) - \mathbb{Q}\{\hat{\mathbb{p}}(\cdot,s)\}|_{s=S}\right) - D_{\mathrm{KL}}(\mathbb{Q}\parallel\mathbb{P}_H)\right)\right\}$$

Then, applying Markov's inequality yields:

$$\mathbb{P}_S\left\{\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)}\lambda\left(\Phi_{\frac{\lambda}{N}}(\mathbb{Q}\{\mathbb{p}\}) - \mathbb{Q}\{\hat{\mathbb{p}}(\cdot,s)\}|_{s=S}\right) - D_{\mathrm{KL}}(\mathbb{Q}\parallel\mathbb{P}_H) \geq \log\left(\frac{1}{\varepsilon}\right)\right\}$$

$$\leq \varepsilon\,,$$

from which the conclusion follows. □

4.5 Experiments

Similar to before, we consider the following five experiments: a strongly convex and smooth quadratic problem, a convex and smooth image processing problem, the convex and non-smooth LASSO problem, the non-convex and non-smooth problem of training a neural network, and a non-convex and non-smooth stochastic empirical risk minimization problem. The training procedure is mostly the same as in Section 3.5. The main difference, however, is that we replace

$$\ell_{\text{train}}(h, p, x^{(0)}, t_{\text{len}}) = \sum_{i=1}^{t_{\text{len}}} \mathbb{1}\{\ell(x^{(i-1)}, p) > 0\} \frac{\ell(x^{(i)}, p)}{\ell(x^{(i-1)}, p)}$$

by

$$\ell_{\text{train}}(h, p, z^{(t)}) = \mathbb{1}\{\ell(\pi_{\mathfrak{X}}(z^{(t)}), p) > 0\} \frac{\ell(\pi_{\mathfrak{X}}(z^{(t+1)}), p)}{\ell(\pi_{\mathfrak{X}}(z^{(t)}), p)} \cdot \mathbb{1}_{\mathsf{C}^c}(z^{(t)}, p)\,,$$

where $\mathsf{C} \subset \mathcal{P} \times \mathcal{Z}$ is the convergence set. Thus, we fix $t_{\text{len}} = 1$ and the algorithm "observes" a loss only as long as it did not reach the convergence set. This effectively solves the problem mentioned before that the algorithm might observe a "full loss" in the case of convergence. For completeness, we briefly summarize the training procedure again: In the outer loop, we sample a loss-function randomly from the training set. Then, in the inner loop, we train the algorithm on this loss-function with ℓ_{train}, that is, in each iteration the algorithm computes a new point and observes the loss ℓ_{train}, which is used to update its hyperparameters. This finally yields some hyperparameters h_0. Then, starting from h_0, we construct the *discrete* prior distribution $\mathbb{P}_H$ over points $h_1, \ldots, h_{n_{\text{sam}}} \in \mathcal{H}$, by a sampling procedure. Finally, we perform the (closed-form) PAC-Bayesian optimization step, which yields the posterior $\mathbb{Q}^* \in \mathcal{M}_1(\mathbb{P}_H)$. In the end, for simplicity, we set the hyperparameters to

$$h^* = \operatorname*{arg\,max}_{i=1,\ldots,n_{\text{sam}}} \mathbb{Q}^*\{h_i\}\,.$$

In the description below, we use $x^{(t)}$, $t \in \mathbb{N}_0$, to denote the iterates of the algorithm in the optimization space, that is, $x^{(t)} = \pi_{\mathfrak{X}}(z^{(t)})$, and, typically, we have $x^{(t)} \in \mathbb{R}^d$, $d \in \mathbb{N}$.

4.5.1 Quadratics

In this subsection, we consider strongly convex quadratic functions with varying strong convexity, varying smoothness and varying right-hand side, that is, each optimization problem is of the form:

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}\|Ax - b\|^2\,, \qquad A \in \mathbb{R}^{d \times d},\ b \in \mathbb{R}^d\,.$$

Thus, the parameters are given by $p = (A, b) \in \mathbb{R}^{d^2+d} =: \mathcal{P}$, while the optimization variable is $x \in \mathbb{R}^d$, and we use $d = 200$. We control the strong-convexity and smoothness of ℓ by sampling them randomly in the intervals $[m_-, m_+], [L_-, L_+] \subset (0, +\infty)$, and define the matrix A_j,

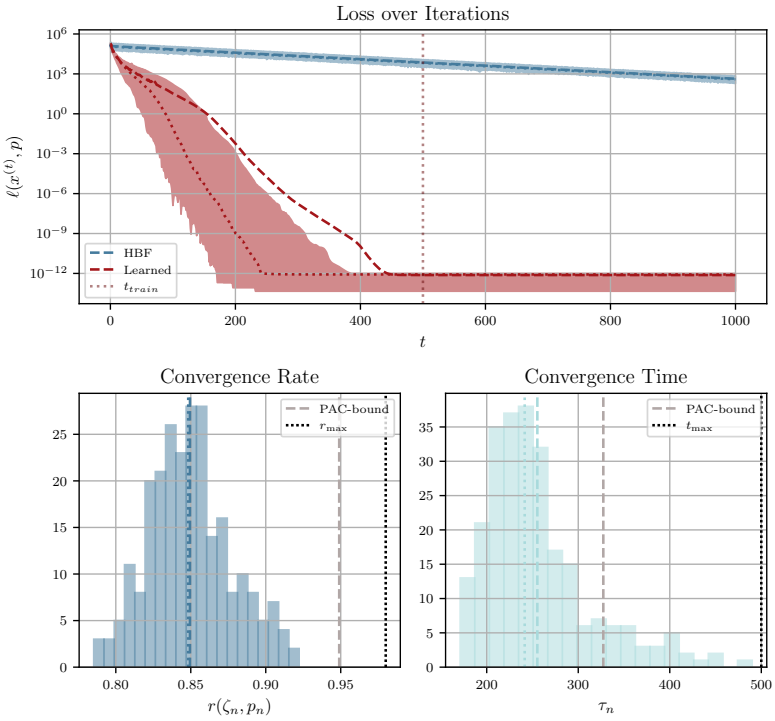

Figure 4.12: Quadratic: The **top figure** shows the loss over the iterations, where HBF is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. We can see from the figure that the learned algorithm reaches the convergence criterion way faster than HBF. The **lower left plot** shows the convergence rate of the learned algorithm. Here, the dashed line represent the empirical mean and the PAC-bound, respectively, and we can see that the bound is not vacuous, but also not really tight. Similarly, the **lower right figure** shows the convergence time of the learned algorithm. Again, the dashed lines represent the empirical mean and the corresponding PAC-bound.

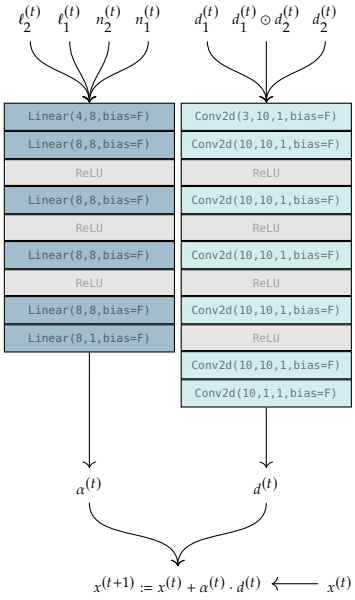

Figure 4.13: Update step of $\mathcal{A}$ for the experiment on quadratic problems (similar to before): The directions $d_1^{(t)}$, $d_2^{(t)}$ and $d_1^{(t)} \odot d_2^{(t)}$ are inserted as different channels into the `Conv2d`-block, which performs 1×1 "convolutions", that is, the algorithm acts coordinate-wise on the input. The scales $n_1^{(t)}$, ..., $n_4^{(t)}$ get transformed separately by the fully-connected block.

$j = 1, ..., N$, as a *diagonal matrix* with entries $a_{ii}^j = \sqrt{m_j} + i \cdot \frac{\sqrt{L_j} - \sqrt{m_j}}{d}$, $i = 1, ..., d$. While, in principle, this is restrictive, we do not use this knowledge explicitly, and, later on, achieve a similar performance in the image-processing and LASSO problems, which both include a non-diagonal quadratic term. Finally, we define the convergence set $\mathsf{C}_{\text{quad}} \subset \mathcal{P} \times \mathcal{Z}$ as

$$\mathsf{C}_{\text{quad}} := \{\ell(\pi_{\mathcal{X}}(z), p) < 10^{-12}\}.$$

Since the given class of functions is L_+-smooth and m_--strongly convex, we use *heavy-ball with friction* (HBF; Polyak, 1964) as baseline. Its update is given by $x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}) + \beta \left(x^{(t)} - x^{(t-1)}\right)$, where the optimal worst-case convergence rate is attained for $\alpha = \left(\frac{2}{\sqrt{L_+} + \sqrt{\mu_-}}\right)^2$, $\beta = \left(\frac{\sqrt{L_+} - \sqrt{\mu_-}}{\sqrt{L_+} + \sqrt{\mu_-}}\right)^2$ (Nesterov, 2018). On the other hand, the learned algorithm $\mathcal{A}$ performs an update of the form $x^{(t+1)} = x^{(t)} + \alpha^{(t)} \cdot d^{(t)}$, where $\alpha^{(t)}$ and $d^{(t)}$ are predicted by separate blocks of a neural network, which is visualized in Figure 4.13. Here, the step-size $\alpha^{(t)}$ is computed by a fully-connected block (without bias) and ReLU activation functions based on the the inputs $n_1^{(t)} = \log(1 + \|\nabla_1 \ell(x^{(t)}, p)\|)$, $n_2^{(t)} = \log(1 + \|x^{(t)} - x^{(t-1)}\|)$, $\ell_1^{(t)} = \log(1 + \ell(x^{(t)}, p))$ and $\ell_2^{(t)} = \log(1 + \ell(x^{(t-1)}, p))$, whereas the direction $d^{(t)}$ is computed by a 1×1-convolutional block, where the input-channels are given by the normalized gradient $d_1 := \frac{\nabla_1 \ell(x^{(t)}, p)}{\|\nabla_1 \ell(x^{(t)}, p)\|}$, the normalized momentum term $d_2 := \frac{x^{(t)} - x^{(t-1)}}{\|x^{(t)} - x^{(t-1)}\|}$, and their point-wise product $d_1 \odot d_2$, and we use 10 channels in each hidden layer.

The upper plot of Figure 4.12 shows that the learned algorithm outperforms HBF by orders of magnitude. The median is shown as dotted line, while the mean is shown as dashed line. The shaded region indicates the area up to the quantile $q = 0.95$, that is, 95% of the test data. We can observe that the mean is not representative for the typical performance

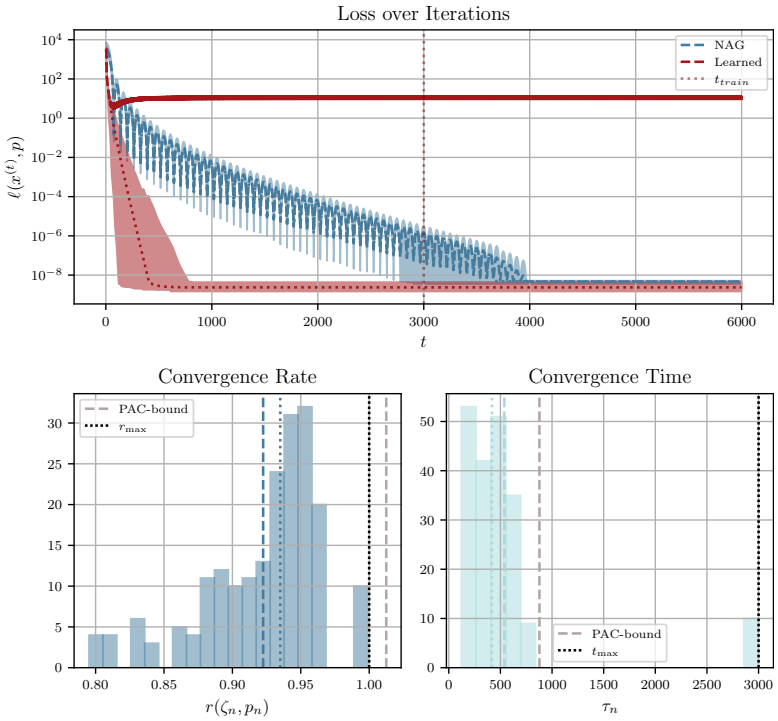

Figure 4.14: Image processing: The **top figure** shows the loss over the iterations, where NAG is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. The **lower left plot** shows the convergence rate of the learned algorithm. The dashed lines represent the empirical mean and the PAC-bound, and we can see that the bound is vacuous here. Similarly, the **lower right figure** shows the convergence time of the learned algorithm. Again, the dashed lines represent the empirical mean and the corresponding PAC-bound, which, in this case, is quite tight.

of the algorithm, and is strongly influenced by a few problem instances for which the learned algorithm does not work as good. In the lower right plot, the convergence time is shown. We can see that for most of the problems the stopping criterion is reached before $t_{\max} = 500$ iterations, and, on average, the learned algorithms needs about 250 iterations to solve such a problem. Further, the provided PAC-bound yields a reasonable estimate of the true (average) convergence time. Similarly, the lower left plot shows the estimated convergence rate: On average, the learned algorithm contracts the loss by a factor of less than 0.85 per iteration. However, while the given PAC-bound is not vacuous, it is also not tight.

4.5.2 Image Processing

In this subsection, we consider a (gray-scale) *image denoising/deblurring* problem with a smooth approximation to the L_1-norm of the image derivative as regularizer, that is, problems of the form:

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}\|Ax - b\|^2 + \lambda \sum_{i,j=1}^{d} \sqrt{(D_h x)_{i,j}^2 + (D_w x)_{i,j}^2 + \varepsilon^2},$$

where $\lambda \in \mathbb{R}$, $A, D_h, D_w \in \mathbb{R}^{d \times d}$, and $b \in \mathbb{R}^d$. Here, the matrix A describes the "blurring" of the image, while D_h and D_w are the discrete image derivatives in h- and w-direction, respectively, which are used to penalize local changes in the image. We use images of height $N_h = 200$ and width $N_w = \lfloor 0.75 \cdot N_h \rfloor = 150$. Thus, the dimension d of the optimization space is given by $d = 30000$. Further, as parameters p we use the observed image and the regularization parameter, that is, $p = (b, \lambda) \in \mathbb{R}^{d+1} =: \mathcal{P}$. Throughout, we use $\varepsilon = 0.01$. For computational efficiency, the matrices A, D_h, D_w are implemented through the convolution of the image x with a corresponding kernel (with reflective boundary conditions). Additionally, after blurring an image with A, we add centered Gaussian noise $\varepsilon_{i,j}$ with standard deviation $\sigma = \frac{25}{256}$ to each pixel. The regularization parameters

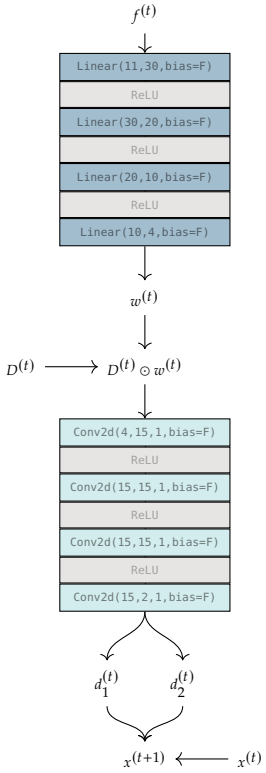

Figure 4.15: Algorithmic update for the image-processing problem: Based on the features $f^{(t)}$, the first block computes a weight vector $w^{(t)}$, which is used to perform a weighting of the different directions in the matrix $D^{(t)}$. Then, the second block consists of a 1x1-convolutional layers, which computes two update direction $d_1^{(t)}$ and $d_2^{(t)}$, which we use to update $x^{(t)}$ based on Equation (4.4).

$\lambda_i \in \mathbb{R}$, $i = 1, \dots, N$, are sampled uniformly from the interval $[0.05, 0.5]$. Finally, we define the convergence set as

$$\mathsf{C}_{\text{img}} = \{(p, z) \in \mathcal{P} \times \mathcal{Z} \ : \ \|\nabla \ell(\pi_{\mathcal{X}}(z), p)\| < 10^{-4}\} \, .$$

Since the problem is smooth and convex, yet not strongly convex, the baseline algorithm is given by the *accelerated gradient descent* algorithm due to Nesterov (1983). Its update is given by first computing $y^{(t+1)} = x^{(t)} + \frac{a^{(t)}-1}{a^{(t+1)}}(x^{(t)} - x^{(t-1)})$ followed by setting $x^{(t+1)} = y^{(t)} - \alpha \nabla f(y^{(t+1)})$. We use the optimal choices $a^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(a^{(t)})^2}\right)$ and $\alpha = \frac{1}{L}$. Here, the smoothness constant L is given by the largest eigenvalue of $A^T A + \frac{\lambda}{\varepsilon} D^T D$, where $D \in \mathbb{R}^{2d \times d}$ is given by "stacking" D_h and D_w, that is, $D = \begin{pmatrix} D_h & D_w \end{pmatrix}^T$. On the other hand, the learned algorithm $\mathcal{A}$ performs an update of the form

$$x^{(t+1)} = x^{(t)} + \frac{\|\nabla_1 \ell(x^{(t)}, p)\|}{L} d_1^{(t)} + \|x^{(t)} - x^{(t-1)}\| \, d_2^{(t)} - \frac{1}{L} \nabla_1 \ell(x^{(t)}, p) \, , \quad (4.4)$$

where the directions $d_1^{(t)}$ and $d_2^{(t)}$ are predicted by a neural network, which is visualized in Figure 4.15. For more details on the update, especially on the vector $f^{(t)}$ and the matrix $D^{(t)}$, we refer to Appendix B.1.
The results of this experiment are summarized in Figure 4.14. We can see that, on a typical problem from this distribution, the algorithm clearly outperforms NAG and reaches the convergence set in about 500 iterations. However, as the mean (dashed line) strongly deviates from the remaining test problems (shaded area), we observe that the learned algorithm does not converge for all of the problems. This is also validated by the lower right plot, which shows the convergence time: For some problems, the algorithm does not reach the convergence set before $t_{\max} = 3000$. Furthermore, the lower left plot shows, unfortunately, that the PAC-bound for the convergence rate is vacuous here. This might be due to the fact that the rate function already yields a value close to $r_{\max}$. Thus, the PAC-bound, which has to be greater or equal, will exceed $r_{\max}$ easily. Therefore, we actually expect this behavior to happen whenever the actual rate is close to $r_{\max}$. Nevertheless, with more training data the bound would get more tight, and, additionally, we still have the guarantee for the convergence time, which is actually quite tight here.

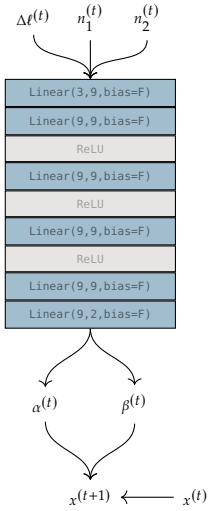

Figure 4.16: Alternative architecture for the image processing experiment: Based on the difference in the observed loss, the norm of the gradient and the norm of the momentum term (all logarithmically transformed), the fully-connected block predicts a step-size $\alpha^{(t)}$ and a momentum parameter $\beta^{(t)}$. Then, we use these to update the iterate as in the heavy-ball algorithm, that is $x^{(t+1)} = x^{(t)} - \alpha^{(t)} \nabla_1 \ell(x^{(t)}, p) + \beta^{(t)}(x^{(t)} - x^{(t-1)})$.

A second architecture. Based on the fact that the previous architecture did not work for some problem instances (the mean stayed constant at 10^1), we devised another architecture and repeated the experiment. This new architecture (see Figure 4.16) is actually quite simple: it just consists of one fully-connected block with ReLU-activation functions to predict a step-size $\alpha^{(t)}$ and a momentum-parameter $\beta^{(t)}$. Then, we use these to update the iterate in the same way as the heavy-ball algorithm, that is:

$$x^{(t+1)} = x^{(t)} - \alpha^{(t)} \nabla_1 \ell(x^{(t)}, p) + \beta^{(t)}(x^{(t)} - x^{(t-1)}) \, .$$

This has two key advantages: First, the update is "computationally lightweight", that is, it actually has a similar computational cost as NAG. This is due to the fact that the matrices in the linear layers are small, such that the additional cost through the matrix multiplications in the fully-connected block is negligible compared to the overall processing of the images. Second, the update is highly interpretable, because it resembles the heavy-ball algorithm and thus might allow for further

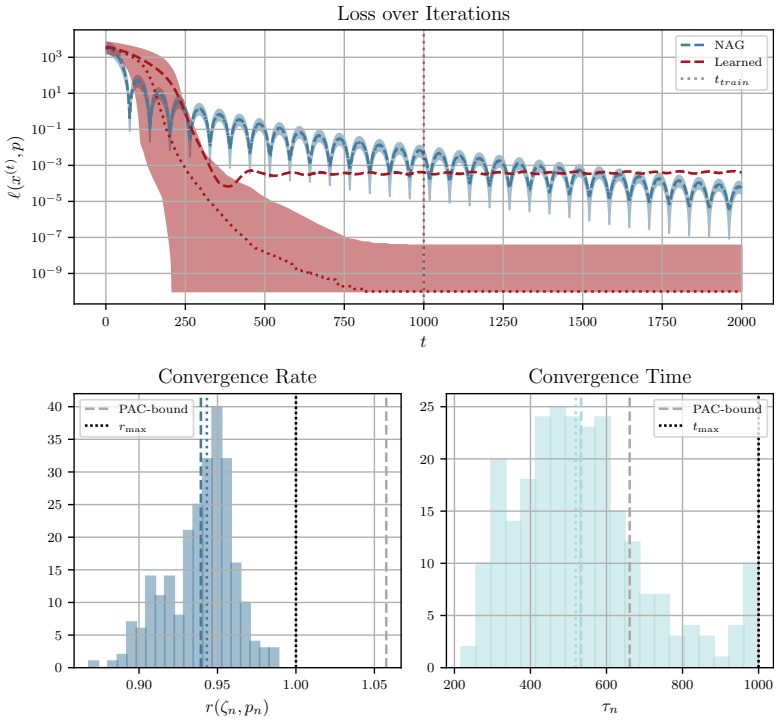

Figure 4.17: Image processing with the alternative architecture: The **top figure** shows the loss over the iterations, where NAG is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. The **lower left plot** shows the convergence rate of the learned algorithm. The dashed lines represent the empirical mean and the PAC-bound, and, unfortunately, we can see that also here the bound is vacuous. Similarly, the **lower right figure** shows the convergence time of the learned algorithm. Again, the dashed lines represent the empirical mean and the corresponding PAC-bound, which, in this case, is quite tight.

analytic considerations.
The results of this experiment are shown in Figure 4.17: We can see that the learned algorithm significantly outperforms NAG and converges in less than 1000 iterations for most problem instances. However, there are still single problems for which it does not reach the stopping criterion. Nevertheless, this time the mean plateaus at a loss of less than 10^{-3}, which, given the size of the optimization variable, is a significant improvement. Therefore, overall, we conclude that this new architecture is clearly preferable.

4.5.3 LASSO

In this subsection, we consider the LASSO problem (Tibshirani, 1996), that is, a non-smooth problem of the form:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|_2^2 + \lambda \|x\|_1 \qquad A \in \mathbb{R}^{m \times n},\ b \in \mathbb{R}^m\,,$$

with $m \leq n$. Hence, the optimization variable is given by $x \in \mathbb{R}^n$, and we use the same matrix $A \in \mathbb{R}^{m \times n}$ with dimensions $n = 70$ and $m = 35$ for all problem instances, where we sample each entry uniformly in $[-0.5, 0.5]$. Thus, the parameters p are given by the right-hand side and the regularization parameter, that is, $p = (b, \lambda) \in \mathbb{R}^{m+1} =: \mathcal{P}$. For this, the regularization parameter λ is sampled uniformly from $[5, 10]$, while the right-hand side is sampled from a multivariate normal distribution. Since the problem is convex and non-smooth, we use the *fast iterative shrinkage-thresholding algorithm* algorithm (FISTA; Beck et al., 2009) as baseline, which performs an extrapolation step followed by a proximal gradient step, that is, abbreviating $h(x) := \frac{1}{2}\|Ax - b\|^2$ and $g(x) := \lambda \|x\|_1$, the update is given by first computing $y^{(t)} = x^{(t)} + \frac{a^{(t)}-1}{a^{(t+1)}}\left(x^{(t)} - x^{(t-1)}\right)$ followed by setting $x^{(t+1)} = \operatorname{prox}_{\alpha g}\left(y^{(t)} - \alpha \nabla h(y^{(t)})\right)$. Here, the proximal mapping can be computed in closed-form yielding the *soft-thresholding*

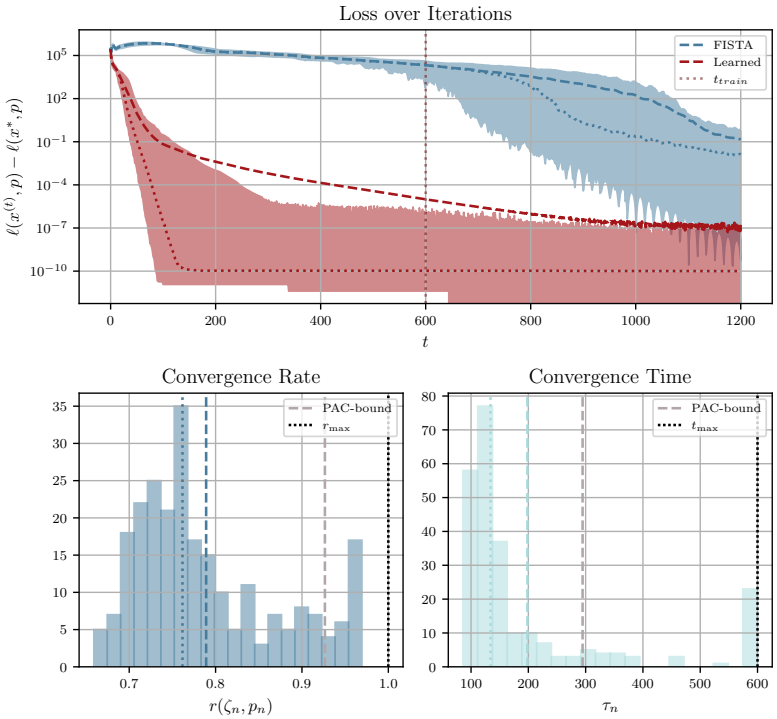

Figure 4.19: LASSO-Problem: The **top figure** shows the loss over the iterations, where the *fast iterative shrinkage-thresholding algorithm* (FISTA) is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. We can see that, on a *typical* problem from this distribution, the learned algorithm reaches the stopping criterion in less than 200 iterations, and outperforms FISTA by several orders of magnitude. However, as the mean indicates, there are problem instances on which also the learned algorithm is rather slow. As the **lower left plot** shows, the learned algorithm strongly contracts the loss (on average) by a factor of 0.8 in each iteration. Similarly, the **lower right figure** shows that, typically, the learned algorithm reaches the convergence set in about 200 iterations. In both cases, the predicted PAC-bound is reasonable tight.

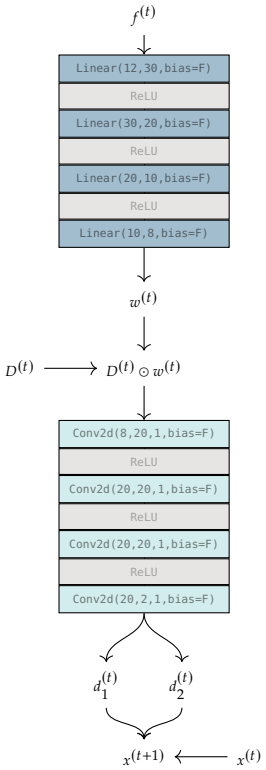

Figure 4.18: Algorithmic update $\mathcal{A}$ for the LASSO-experiment (same as before): Based on the features $f^{(t)}$, the first block computes a weighting vector $w^{(t)}$, which is used to scale the different columns (directions) in the matrix $D^{(t)}$. Then, based on $D^{(t)} \odot w^{(t)}$, the second block predicts two directions $d_1^{(t)}, d_2^{(t)}$, where $d_1^{(t)}$ only acts on the zero entries, and $d_2^{(t)}$ acts on the non-zero entries. Then, we update $x^{(t+1)}$ based on Equation (4.5).

operator $\hat{x}_i = \mathbb{1}\{|\bar{x}_i| > \alpha\lambda\} \cdot \left(\bar{x}_i - \alpha\lambda \frac{\bar{x}_i}{|\bar{x}_i|}\right), i = 1, \dots, d$. We choose $\alpha = 1/L$, where L is the largest eigenvalue of $A^T A$, while $a^{(t)}$ is updated iteratively as $a^{(t+1)} = \frac{1}{2}\left(1 + \sqrt{1 + 4(a^{(t)})^2}\right)$. Since the problem is non-smooth, the norm of the gradient cannot be used as stopping criterion. Thus, we define the convergence set $\mathsf{C}_{\text{lasso}} \subset \mathcal{P} \times \mathcal{Z}$ as

$$\mathsf{C}_{\text{lasso}} := \left\{ \| \pi_{\mathcal{X}}(z) - \text{prox}_{\alpha g}\left(\pi_{\mathcal{X}}(z) - \alpha \nabla h(\pi_{\mathcal{X}}(z))\right) \| < 10^{-6} \right\}.$$

On the other hand, the learned algorithm $\mathcal{A}$ performs an update of the form

$$x^{(t+1)} = \text{prox}_{g/L}\left(x^{(t)} + \frac{1}{L}\left(d_1^{(t)} - \nabla h(x^{(t)}) + \|x^{(t)} - x^{(t-1)}\| \cdot d_2^{(t)}\right)\right), \quad (4.5)$$

where $d_1^{(t)}$ and $d_2^{(t)}$ are predicted by a neural network, which is visualized in Figure 4.18. For more details on the update, especially on the features $f^{(t)}$ and the directions $D^{(t)}$, we refer to the Appendix B.2.
The results are summarized in Figure 4.19: The upper plot shows that, on a typical example, the learned algorithm reaches the convergence criterion in about 200 iterations, and outperforms FISTA by many orders of magnitude. However, since the mean (dashed line) strongly deviates from the median (dotted line), we can observe that there are single problem instances on which the algorithm does not perform as good. Especially, it seems that there are single problem instances, for which instabilities occur. However, this happens only at a small loss. The two lower plots show the convergence rate and time, respectively, together with the predicted PAC-bounds. In both cases, they are reasonable tight.

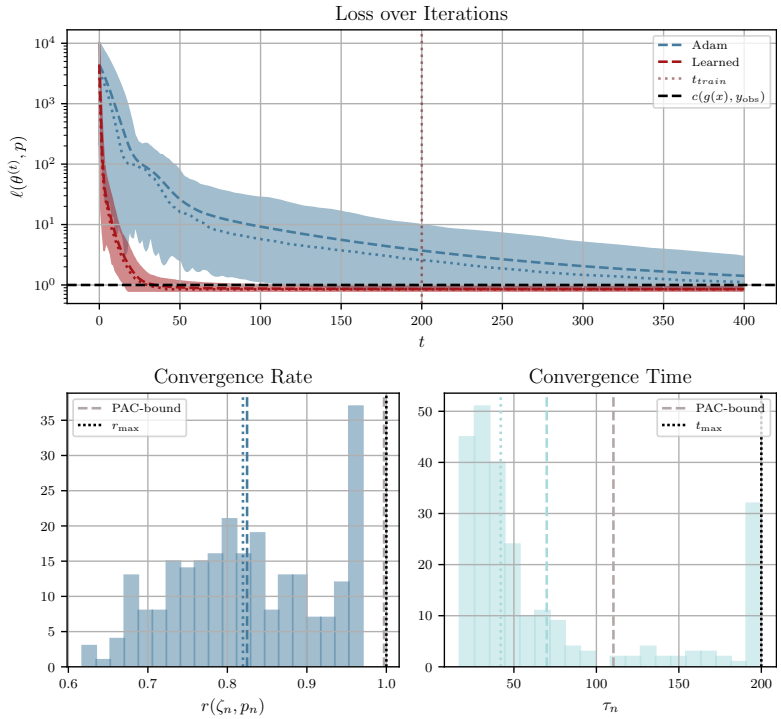

Figure 4.20: Training the neural network: The **top figure** shows the loss over the iterations, where Adam is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. As can be seen, the learned algorithm reaches the ground-truth loss after about 25 iterations, and clearly outperforms Adam. The **lower left plot** shows the convergence rate of the learned algorithm, and we can see that it varies strongly. This is due to the fact that the loss plateaus quickly, but does not necessarily reach the convergence set, which can also be observed in the **lower right figure**, as there are about 30 problem instances, where the learned algorithm does not reach the convergence criterion in the given 200 iterations. However, on average, the learned algorithm reaches the convergence set in about 75 iterations.

4.5.4 Training a Neural Network

In this subsection, we consider the problem of training a neural network on a regression problem, that is, $\mathcal{A}$ is trained to predict the parameters $\theta \in \mathbb{R}^n$ of a neural network $\mathtt{N}(\theta, \cdot)$, which then is used to predict a function $g : \mathbb{R} \to \mathbb{R}$. Hence, the optimization variable is given by $\theta \in \mathbb{R}^n$. We assume that the neural network should learn the function $g : \mathbb{R} \to \mathbb{R}$ from noisy observations $y_j = g(x_j) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, 1)$. For this, we construct polynomials g_i, $i = 1, \dots, N$, of degree $d = 5$ by sampling points $\{x_{i,j}\}_{j=1}^K$ (here: $K = 50$) uniformly in $[-2, 2]$ and the coefficients $(c_{i,0}, \dots, c_{i,5})$ of g_i uniformly in $[-5, 5]$. For every function $g_i : \mathbb{R} \to \mathbb{R}$ the neural network is trained on the data set $p_i := \{X_i, Y_i\}$ with $X_i = (x_{i,1}, \dots, x_{i,K}) \in \mathbb{R}^K$ and $Y_i = (y_{i,1}, \dots, y_{i,K}) \in \mathbb{R}^K$. Hence, the data set $\{X_i, Y_i\}$ will serve as the parameter of the loss function, such that the parameter space $\mathcal{P}$ can be identified as the space of these data sets, that is, $\mathcal{P} = \mathbb{R}^{K \times 2}$. Since the mean square error is the standard choice for training models on regression tasks, the loss is given by $\ell(\theta, p_i) := c(\mathtt{N}(\theta, X_i), Y_i) := \frac{1}{K} \sum_{j=1}^K (\mathtt{N}(\theta, x_{i,j}) - y_{i,j})^2$, and for $\mathtt{N}$ we use a fully-connected two layer neural network with ReLU-activation functions. To have more features in the input layer, the input x is transformed into the vector $(x, x^2, \dots, x^5)$. Hence, the parameters $\theta \in \mathbb{R}^n$ of the neural network are given by the weights $A_1 \in \mathbb{R}^{50 \times 5}$, $A_2 \in \mathbb{R}^{1 \times 50}$ and biases $b_1 \in \mathbb{R}^{50}$, $b_2 \in \mathbb{R}$ of the two fully-connected layers. Therefore, the optimization space is of dimension $n = (5 \cdot 50) + (1 \cdot 50) + 50 + 1 = 351$. As baseline we use Adam (Kingma et al., 2015) as it is implemented in PyTorch (Paszke et al., 2019), which is a widely used optimization algorithm for training neural networks. For tuning, we perform a grid search over 100 step-size parameters in $[10^{-4}, 10^{-2}]$, such that its performance is best for the given $t_{\text{train}} = 200$ iterations, which yields the value $\kappa = 0.008$. Note that, originally, Adam was introduced for stochastic optimization, while we use it in the "full-batch setting" here, that is, without stochasticity. We

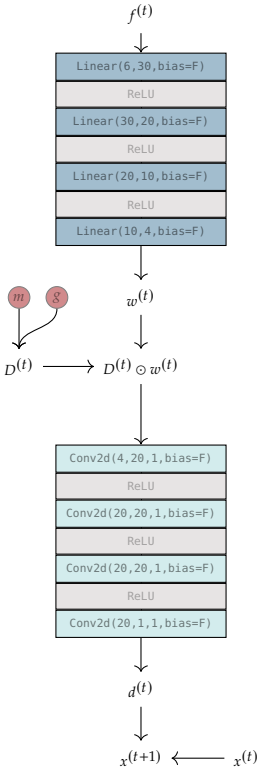

Figure 4.21: Algorithmic update for training the neural network: Based on the given six features, the first block computes a weight vector $w^{(t)}$, which is used to scale the different directions in $D^{(t)}$. Then, this reweighted matrix gets inserted as channels into the second block, which computes an update direction $d^{(t)}$. Finally, we update $x^{(t+1)} := x^{(t)} + d^{(t)}$.

define the convergence set as

$$\mathsf{C}_{\text{nn}} := \{(p, z) \in \mathcal{P} \times \mathcal{Z} \ : \ \ell(\pi_{\mathcal{X}}(z), p) < 0.85\} \, .$$

This is based on the fact that, since we add standard normal noise, the expected loss of the ground truth takes the value 1.0, that is, a loss below 1.0 can be regarded as overfitting. The learned algorithm simply performs the update $x^{(t+1)} = x^{(t)} + d^{(t)}$, where $d^{(t)}$ is predicted by a neural network, which is visualized in Figure 4.21. For more details on the architecture, we refer to Appendix B.3.
The upper plot of Figure 4.20 shows that the learned algorithm clearly outperforms Adam, reaching the ground-truth loss already after about 25 iterations, whereas Adam is not able to reach it within 400 iterations (on average), as the mean is still above 1.0. The lower left plot shows that the learned algorithm contracts the loss in each iteration, on average, by a factor of 0.82, and the lower right plot shows that it typically needs about 75 iterations to reach the convergence set. However, one can also observe that there is a significant amount of problems that do not reach the convergence set in 200 iterations, and that do not have a "fast convergence rate". These two findings are strongly correlated, because, if the loss plateaus without reaching the convergence set, the estimated rate-function deteriorates. In this case, that is, a non-smooth and non-convex problem, it might be due to the fact that our definition of the convergence set is somewhat arbitrary. Nevertheless, in both cases, the provided PAC-bound is reasonably tight.

Remark 4.5.1 Another definition that one could use is given by the procedure of *early stopping*, that is, one stops the algorithm as soon as the validation loss deviates from the training loss by a certain amount. Through this, one could get a guarantee for how long the network has to be trained until it starts to overfit to the training data.

4.5.5 Stochastic Empirical Risk Minimization

Lastly, we consider the problem of stochastic empirical risk minimization. For this, we use the same problem and setup as in Subsection 4.5.4 for training a neural network. However, this time, in each iteration the algorithm only has access to a randomly selected minibatch (of size $m = 5$) instead of the full-batch setting considered before. Therefore, we have to deal with a stochastic, non-convex, and non-smooth optimization problem. Since we cannot access the full gradient anymore, we define the convergence set as

$$\mathsf{C}_{\text{stoch}} := \{(p, z) \in \mathcal{P} \times \mathcal{Z} \ : \ \ell(\pi_{\mathcal{X}}(z), p) < 0.9\} \, .$$

As before, the value 0.9 is chosen due to our construction of the data set. As baseline, Adam is used again, where we perform a grid-search over 100 step-sizes in the interval $[10^{-6}, 10^{-1}]$ in such a way that its average performance is best after $t_{\text{train}} = 2500$ iterations, which yields the value of $\kappa = 4 \cdot 10^{-3}$. Here, the (empirical) average is taken over 25 problem instances, where we perform 10 runs per problem instance. As learned algorithm, we use a "preconditioned" version of Adam, that is, we update $x^{(t)}$ as follows:

$$x^{(t)} = x^{(t-1)} - (s^{(t)} \cdot \alpha \cdot p_1^{(t)} \odot \hat{m}^{(t)}) / (0.001 \cdot |p_2^{(t)}| \odot \sqrt{\hat{v}^{(t)}} + \varepsilon) \, , \qquad (4.6)$$

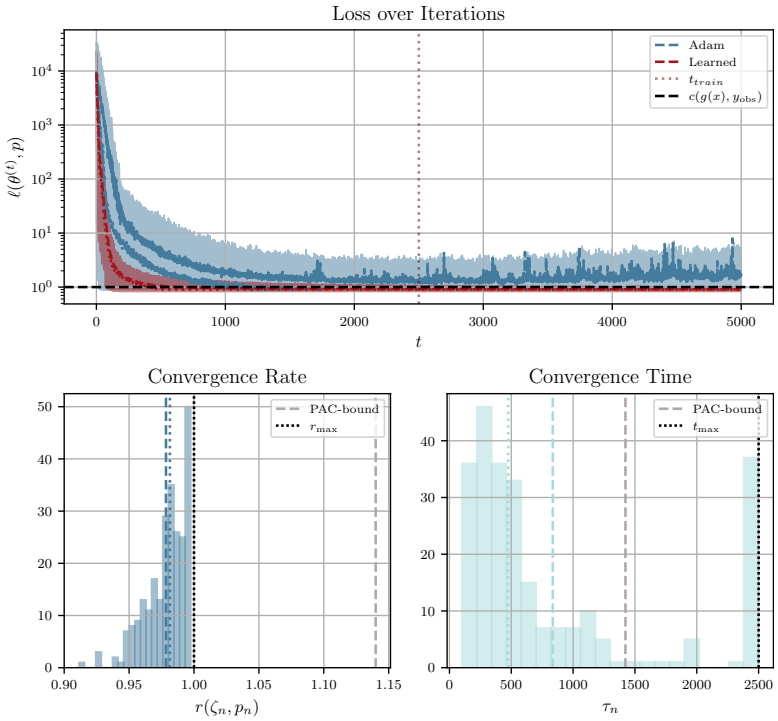

Figure 4.22: Stochastic empirical risk minimization: The **top figure** shows the loss (emp. risk) over the iterations, where Adam is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents the test data up to the quantile $q = 0.95$, that is, 95% of the test data. Here, the learned algorithm reaches the ground-truth faster than Adam and its performance varies way less. The **lower left plot** shows the convergence rate of the learned algorithm and we can see that, unfortunately, the provided PAC-bound is vacuous. This is due to the fact that the loss plateaus quickly, while the algorithm still needs many iterations to reach the convergence set. However, the **lower right figure** shows that, on average, the learned algorithm reaches the convergence set in about 800 iterations and the PAC-bound provides a reasonable estimate for this.

where $\hat{m}^{(t)}, \hat{v}^{(t)}$ are computed in the same way as for Adam, and $s^{(t)}, p_1^{(t)}, p_2^{(t)}$ are predicted by a neural network, which is visualized in Figure 4.23. More details about the update, especially the features $f^{(t)}$ and the directions in $D^{(t)}$, are given in Appendix B.4.
The upper plot of Figure 4.22 shows the empirical risk over the iterations, where we have performed one run per problem. The plot shows that the learned algorithm still outperforms Adam, although not as clearly as in the full-batch case. The lower right plot shows that, on average, the learned algorithm reaches the convergence set in about 800 iterations, and we can see that the provided PAC-bound is reasonably tight. Here, the median shows that more than 50% of the problems actually need less than 500 iterations, while the median of Adam (in the upper plot) indicates that the median convergence time for Adam is at about 1300 iterations. Thus, on a majority of instances, the learned algorithm needs way less iterations to reach the convergence set. On the other hand, however, the lower left plot shows that the provided bound for the convergence rate is vacuous. Again, this can be attributed to the fact that the algorithm has to perform many iterations to reach the convergence set, which in turn yields a value (for the convergence rate) close to 1.0, which then gives a vacuous PAC-bound.

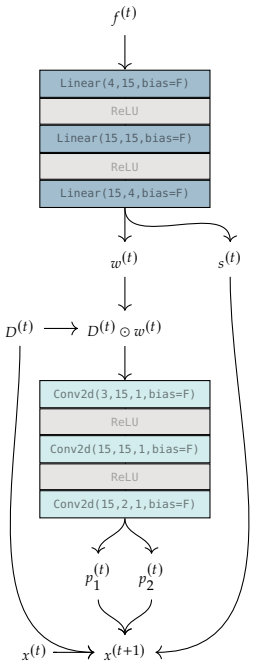

Figure 4.23: Algorithmic update $\mathcal{A}$ for stochastic empirical risk minimization: Based on the features $f^{(t)}$, the first block computes a weight vector $w^{(t)}$ and a step-size $s^{(t)}$. Then, $w^{(t)}$ is used to scale the different directions in $D^{(t)}$ before they get inserted as channels into the second block, which consists of 1x1-convolutional layers, and computes two vectors $p_1^{(t)}, p_2^{(t)}$. Finally, we update $x^{(t)}$ based on Equation (4.6).

4.6 Discussion and Limitations

The goal of this chapter was to model optimization algorithms in learning-to-optimize more faithfully, so that it is possible to give generalization guarantees for (nearly) any kind of interesting statistics. In particular, this involves convergence rates and stopping times of the algorithm, which both depend on the whole trajectory of the algorithm instead of single iterates. Therefore, we introduced a probabilistic model for the distribution of the trajectory of an abstract iterative optimization algorithm

and, based on this model, we provided generalization bounds for the convergence rate and the convergence time of the learned algorithm. However, both results are still *non-asymptotic*: The stopping time has to be finite and the provided rate function is only an approximation to a "true" convergence rate, which, by construction of the algorithm, would be valid for any iterate. While the provided framework does theoretically allow for non-asymptotic results as it allows for accessing the whole trajectory of the algorithm, they might not be within reach for *practical* generalization results, simply because of the fact that asymptotic events are inherently non-observable. Thus, the results presented here are still insufficient to derive the convergence of the learned algorithm. While one might think that they are actually sufficient, because one could define the convergence set as the set of stationary points, this only holds true in theory. In practice, however, it is typically not the case that one does indeed find a stationary point in a finite number of iterations, such that the corresponding guarantee on the convergence time would be vacuous[4]. Therefore, the next section deals with the problem of convergence of the algorithm.

4: In each run, the algorithm would be stopped after $t_{\max}$ iterations, resulting in the guarantee that the expected stopping time is $\geq t_{\max}$.

Convergence to Stationary Points 5

In this chapter, we provide a new strategy for proving convergence of learned algorithms to stationary points of the loss function. As before, we have the problem that we do not *know* what the algorithm will do, that is, how the update looks like or which conditions it satisfies. However, compared to traditional optimization, our key advantage is the fact that we can *observe* the algorithm during training. While this might sound like a trivial remark, it is indeed the key difference and we will leverage on it in the following. Even more so, we can actually train the algorithm with regard to certain properties. Ultimately, this will result in a generalization result for the probability that the learned algorithm generates a sequence that converges to a stationary point of the loss function.

5.1 The Main Idea

The underlying idea of this section is straightforward. However, it might get obscured by the technical details, such that we shortly, and informally, present it here: Mathematical reasoning is *deductive*, that is, given a set of premises, say a and b, the conclusion c has to follow logically from them. If this is the case, we are *sure* that, given a and b, c holds true. For example, given an object x and properties a, b, and c, we could be interested in the implication:

$$x \text{ satisfies } a \wedge b \implies x \text{ satisfies } c\,.$$

If the implication is valid, we know that, as soon as x satisfies a and b, it inevitably also satisfies c. Now, given a collection of objects x, we could be wondering whether they possess the property c. This is surely the case, if a and b hold true for all of them. However, if a and b hold true only for some objects x, this kind of reasoning cannot be employed directly, and the language of probability theory is more appropriate. Here, however, the properties (or premises) have to be phrased as sets $\mathsf{A} = \{x \,:\, x \text{ satisfies } a\}$, $\mathsf{B} = \{x \,:\, x \text{ satisfies } b\}$, and $\mathsf{C} = \{x \,:\, x \text{ satisfies } c\}$, in which case the implication translates into an inclusion:

$$\mathsf{A} \cap \mathsf{B} = \{x \,:\, x \text{ satisfies } a \wedge b\} \subset \{x \,:\, x \text{ satisfies } c\} = \mathsf{C}\,.$$

Although this is a simple conversion of a logical statement into a statement of measure theory, it in fact allows for a more fine-grained result: If we are given a probability measure μ over objects x, we can always conclude that $\mu\{\mathsf{A} \cap \mathsf{B}\} \leq \mu\{\mathsf{C}\}$, that is, as the implication tells us, it is more likely to observe an object x that satisfies property c than to observe an object x that satisfies both properties a and b. Further, if $\mu\{\mathsf{A} \cap \mathsf{B}\} = 1$, we can deduce that c holds true *almost surely*. In the following, the objects x to be considered are whole sequences, the property c is convergence to stationary points, and μ will be the distribution of the trajectories on $\mathfrak{X}^{\mathbb{N}_0}$. However, we want to stress that this idea is by no means limited to this scenario. Rather, it applies way more generally to all sorts of "desirable properties".

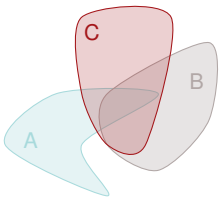

Figure 5.1: Simple Venn diagram for the implication: If x lies in the intersection of A and B, we can directly conclude that it also lies in C.

Note that, as long as $\mu\{\mathsf{A} \cap \mathsf{B}\} > 0$, this case also covers the conditional probability $\mu\{\cdot \mid \mathsf{A} \cap \mathsf{B}\}$, in which case we would get $\mu\{\mathsf{C} \mid \mathsf{A} \cap \mathsf{B}\} = \frac{\mu\{\mathsf{A} \cap \mathsf{B} \cap \mathsf{C}\}}{\mu\{\mathsf{A} \cap \mathsf{B}\}} = 1$.

5.2 Translating Geometry into Measure Theory

"A mathematician who argues from probabilities in geometry is not worth an ace." - *Socrates*

In this section, we combine Theorem 4.4.10 with Theorem 2.3.8 to get a generalization result for the convergence of learned algorithms to stationary points. In doing so, we bring together advanced tools from learning theory and optimization. We show that the probability to observe a parameter p and a corresponding trajectory ζ, which converges to a stationary point of $\ell(\cdot, p)$, generalizes. For this, we formulate the sufficient-descent condition, the relative-error condition, and the boundedness assumption as measurable sets in $\mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$, such that their intersection is exactly the sequences satisfying the properties of Theorem 2.3.8. Since this relies on the results of Chapter 4, we recall the corresponding assumptions:

Assumption 5.2.1 *The state space* $(\mathcal{Z}, \mathfrak{B}(\mathcal{Z}), \mathbb{P}_{Z^{(0)}})$*, the parameter space* $(\mathcal{P}, \mathfrak{B}(\mathcal{P}), \mathbb{P}_P)$*, the hyperparameter space* $(\mathcal{H}, \mathfrak{B}(\mathcal{H}), \mathbb{P}_H)$*, and the randomization space* $(\mathcal{R}, \mathfrak{B}(\mathcal{R}), \mathbb{P}_R)$ *are Polish probability spaces.*

Assumption 5.2.2 $\mathcal{Z}$ *is the product of the* optimization space $\mathcal{X}$ *and the* memory space $\mathcal{M}$*, that is,* $\mathcal{Z} = \mathcal{X} \times \mathcal{M}$*, where both* $\mathcal{X}$ *and* $\mathcal{M}$ *are Polish spaces with metrics* $d_{\mathcal{X}}$ *and* $d_{\mathcal{M}}$*.*

Assumption 5.2.3 *The (possibly extended-valued)* loss-function $\ell : \mathcal{X} \times \mathcal{P} \to [0, +\infty]$ *and the* algorithmic update $\mathcal{A} : \mathcal{H} \times \mathcal{P} \times \mathcal{Z} \times \mathcal{R} \to \mathcal{Z}$ *are both measurable.*

Additionally, since these results also rely on Theorem 2.3.8 and, more generally, variational analysis, we need to impose the following:

Assumption 5.2.4 *We have* $\mathcal{X} = \mathbb{R}^d$ *and* $\mathcal{P} = \mathbb{R}^q$*, and the function* $\ell : \mathcal{X} \times \mathcal{P} \to [0, \infty]$ *is proper, lower semi-continuous, and continuous on* $\operatorname{dom} \ell$*. Furthermore, the map* $(x, p) \mapsto \partial_1 \ell(x, p)$ *is outer semi-continuous.*

Finally, please recall the notation: Given a trajectory $\zeta = (z^{(t)})_{t \in \mathbb{N}_0} \subset \mathcal{Z}$, we denote the projection of $z^{(t)}$ onto the optimization space by $x^{(t)} := \pi_{\mathcal{X}}(z^{(t)}) \in \mathcal{X}$.

5.2.1 Measurability

In this section we first encode the needed properties of the trajectory as sets in $\mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ and then, to apply Theorem 4.4.10, we show that they are actually measurable w.r.t. $\mathfrak{B}(\mathcal{P}) \otimes \mathfrak{B}(\mathcal{Z})^{\otimes \mathbb{N}_0}$.

Lemma 5.2.1 *Suppose that Assumption 5.2.4 holds and denote the (parametric) set of stationary points of* ℓ *by*

$$\mathsf{A}_{\text{stat}} := \{(p, x) \in \mathcal{P} \times \mathcal{X} \ : \ 0 \in \partial_1 \ell(x, p)\} .$$

Then A_{stat} *is closed.*

Proof. Take $(p^{(t)}, x^{(t)})_{t \in \mathbb{N}} \subset \mathsf{A}_{\text{stat}}$ with $(p^{(t)}, x^{(t)}) \to (\bar{p}, \bar{x}) \in \mathcal{P} \times \mathcal{X}$. Then we need to show that $(\bar{p}, \bar{x}) \in \mathsf{A}_{\text{stat}}$. Since the map $(x, p) \mapsto \partial_1 \ell(x, p)$ is

outer semi-continuous, we have:

$$\limsup_{(x,p)\to(\bar{x},\bar{p})} \partial_1 \ell(x,p) \subset \partial_1 \ell(\bar{x},\bar{p})\,.$$

By definition of the outer limit, this is the same as:

$$\{v \in \mathcal{X} \,:\, \exists (x^{(t)},p^{(t)}) \to (\bar{x},\bar{p}),\ \exists v^{(t)} \to v \text{ with } v^{(t)} \in \partial_1 \ell(x^{(t)},p^{(t)})\} \\ \subset \partial_1 \ell(\bar{x},\bar{p})\,.$$

In particular, we have that $(x^{(t)},p^{(t)}) \to (\bar{x},\bar{p})$ and that $0 \in \partial_1 \ell(x^{(t)},p^{(t)})$ for all $t \in \mathbb{N}$. Thus, setting $v^{(t)} := 0$, $t \in \mathbb{N}$, and $v := 0$, we conclude that $0 \in \partial_1 \ell(\bar{x},\bar{p})$, such that $(\bar{p},\bar{x}) \in \mathsf{A}_{\text{stat}}$. Therefore A_{stat} is closed. □

With this, we can show that the (parametric) set of trajectories that converge to a stationary point of the loss-function is measurable w.r.t. $\mathfrak{B}(\mathcal{P}) \otimes \mathfrak{B}(\mathcal{Z})^{\otimes \mathbb{N}_0}$. Here, the section $\mathsf{A}_{\text{stat},p} := \{x \in \mathcal{X} \,:\, (p,x) \in \mathsf{A}_{\text{stat}}\}$ is the set of stationary points of $\ell(\cdot,p)$.

Lemma 5.2.2 *Suppose that Assumptions 5.2.1, 5.2.2, 5.2.3 and 5.2.4 hold. Define the (parametric) set of sequences that converge to a stationary point of ℓ as*

$$\mathsf{A}_{\text{conv}} := \{(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0} \,: \\ \exists x^* \in \mathsf{A}_{\text{stat},p}\ s.t.\ \lim_{t\to\infty} \|x^{(t)} - x^*\| = 0\}\,.$$

Then A_{conv} is measurable.

Proof. Since the proof does not get more complicated by considering general Polish spaces $\mathcal{P}$ and $\mathcal{X}$ instead of $\mathbb{R}^q$ and $\mathbb{R}^d$, we prove the result in this more general setting. Here, we have that $d_{\mathcal{P}\times\mathcal{X}} := d_{\mathcal{P}} + d_{\mathcal{X}}$ is a metric on $\mathcal{P} \times \mathcal{X}$ that metrizes the product-topology, that is, it induces the same product-σ-algebra. Similarly, if we denote the countable dense subset of $\mathcal{P}$ by $\hat{\mathcal{P}}$, and the one of $\mathcal{Z}$ by $\hat{\mathcal{Z}}$, then we have that $\mathcal{D} := \hat{\mathcal{P}} \times \hat{\mathcal{Z}}$ is dense in $\mathcal{P} \times \mathcal{Z}$.
First, we adopt the notation for the *limes inferior* for sets from probability theory: For $\varepsilon > 0$ and $(p,x) \in \mathcal{P} \times \mathcal{X}$, denote by $\{\mathsf{B}_\varepsilon(p,x)\ \text{ult.}\}$ the (parametric) set of trajectories for which the optimization variable ultimately lies in the ball with radius ε around (p,x), that is:

$$\{\mathsf{B}_\varepsilon(p,x)\ \text{ult.}\} := \bigcup_{n\in\mathbb{N}_0} \bigcap_{k\geq n} \{(p', \pi_{\mathcal{X}}(z^{(k)})) \in \mathsf{B}_\varepsilon(p,x)\} \in \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}\,,$$

where the notation $\{(p', \pi_{\mathcal{X}}(z^{(k)})) \in \mathsf{B}_\varepsilon(p,x)\}$ is an abbreviation for $\{(p',(z^{(t)})_{t\in\mathbb{N}_0}) \in \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0} \,:\, (p', \pi_{\mathcal{X}}(z^{(k)})) \in \mathsf{B}_\varepsilon(p,x)\}$. This set is measurable, because we have the following equality:

$$\{(p', \pi_{\mathcal{X}}(z^{(k)})) \in \mathsf{B}_\varepsilon(p,x)\} = \left(g \circ (id, \pi_{\mathcal{X},k})\right)^{-1} [0,\varepsilon)\,,$$

where $g : \mathcal{P}\times\mathcal{X} \to [0,+\infty)$ is defined as $g(p',x') = d_{\mathcal{P}\times\mathcal{X}}((p',x'),(p,x))$, which is continuous and therefore measurable, and $\pi_{\mathcal{X},k} = \pi_{\mathcal{X}} \circ \pi_k$ is the projection onto the k-th iterate of the optimization variable. Hence, $\{\mathsf{B}_\varepsilon(p,x)\ \text{ult.}\} \in \mathfrak{B}(\mathcal{P}) \otimes \mathfrak{B}(\mathcal{Z})^{\otimes\mathbb{N}_0}$.
W.l.o.g. we can assume that $\mathsf{A}_{\text{stat}} \neq \emptyset$, because if it were empty, A_{conv} would be empty too, which is measurable. Thus, we claim that we have

the following equality:

$$\mathsf{A}_{\text{conv}} = \mathsf{L} := \bigcap_{k\in\mathbb{N}} \bigcup_{\substack{(p,z)\in\mathcal{D} \\ \mathsf{A}_{\text{stat}}\cap\mathsf{B}_{1/k}(p,\pi_{\mathfrak{X}}(z))\neq\emptyset}} \{\mathsf{B}_{1/k}(p,\pi_{\mathfrak{X}}(z)) \text{ ult.}\}\,.$$

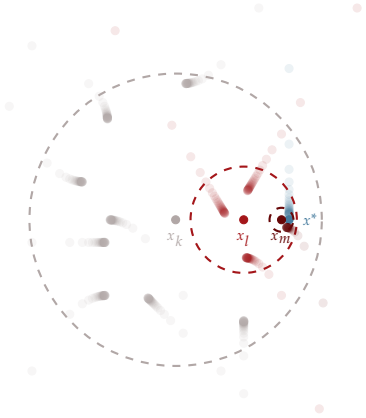

Figure 5.2: Visualization for the sets $\{\mathsf{B}_{1/k}(p_k,x_k) \text{ ult.}\}$, $\{\mathsf{B}_{1/l}(p_l,x_l) \text{ ult.}\}$ and $\{\mathsf{B}_{1/m}(p_m,x_m) \text{ ult.}\}$ (for simplicity, only the variable x is shown): A sequence belongs to $\{\mathsf{B}_{1/k}(p_k,x_k) \text{ ult.}\}$, if it *ultimately* lies in the ball with radius $\frac{1}{k}$ around x_k. If we decrease the radius, the corresponding midpoints converge to x^*, and only sequences that also converge to x^* can ultimately lie in such a ball.

If this equality holds, A_{conv} is measurable as a countable intersection/union of measurable sets, and we show the equality by showing both inclusions. Hence, take $(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{A}_{\text{conv}}$. By definition, there exists $x^* \in \mathfrak{X}$, such that $(p,x^*) \in \mathsf{A}_{\text{stat}}$ and $\lim_{t\to\infty} d_{\mathfrak{X}}(x^{(t)},x^*) = 0$. Hence, for any $k \in \mathbb{N}$, there exists $N_k \in \mathbb{N}$, such that $x^{(t)} \in \mathsf{B}_{1/3k}(x^*)$ for all $t \geq N_k$. Now, take $(p_k,z_k) \in \mathcal{D}$, such that $p_k \in \mathsf{B}_{1/3k}(p)$ and $x_k := \pi_{\mathfrak{X}}(z_k) \in \mathsf{B}_{1/3k}(x^*)$, which exists, since $\mathcal{D}$ is dense. Then we have for all $t \geq N_k$:

$$\begin{aligned} d_{\mathcal{P}\times\mathfrak{X}}((p,x^{(t)}),(p_k,x_k)) &= d_{\mathcal{P}}(p,p_k) + d_{\mathfrak{X}}(x^{(t)},x_k) \\ &\leq d_{\mathcal{P}}(p,p_k) + d_{\mathfrak{X}}(x^{(t)},x^*) + d_{\mathfrak{X}}(x^*,x_k) < \frac{1}{k}\,. \end{aligned}$$

Therefore, we have $(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \{\mathsf{B}_{1/k}(p_k,x_k) \text{ ult.}\}$. Further, it holds:

$$d_{\mathcal{P}\times\mathfrak{X}}((p,x^*),(p_k,x_k)) < \frac{2}{3k} < \frac{1}{k}\,,$$

such that $(p_k,z_k) \in \mathcal{D}$ with $\mathsf{A}_{\text{stat}} \cap \mathsf{B}_{1/k}(p_k,\pi_{\mathfrak{X}}(z_k)) \neq \emptyset$. Since $k \in \mathbb{N}$ was arbitrary, such a tuple can be found for every $k \in \mathbb{N}$, and we conclude that

$$(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \bigcup_{\substack{(p',z')\in\mathcal{D} \\ \mathsf{A}_{\text{stat}}\cap\mathsf{B}_{1/k}(p',\pi_{\mathfrak{X}}(z'))\neq\emptyset}} \{\mathsf{B}_{1/k}(p',\pi_{\mathfrak{X}}(z')) \text{ ult.}\} \quad \forall k \in \mathbb{N}\,.$$

This implies that $(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{L}$, which shows the inclusion $\mathsf{A}_{\text{conv}} \subset \mathsf{L}$. Now, conversely, let $(p,(z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{L}$. Then, for every $k \in \mathbb{N}$ there exists a $(p_k,z_k) \in \mathcal{D}$ and a $N_k \in \mathbb{N}$, such that $\mathsf{A}_{\text{stat}} \cap \mathsf{B}_{1/k}(p_k,\pi_{\mathfrak{X}}(z_k)) \neq \emptyset$ and

$$(p,x^{(t)}) \in \mathsf{B}_{1/k}(p_k,\pi_{\mathfrak{X}}(z_k)), \quad \forall t \geq N_k\,.$$

Denoting $x_k := \pi_{\mathfrak{X}}(z_k)$, we have that the resulting sequence of midpoints $(p_k,x_k)_{k\in\mathbb{N}}$ is Cauchy in $\mathcal{P}\times\mathfrak{X}$, because: Given $k,l \in \mathbb{N}$, we have that $(p,x^{(t)}) \in \mathsf{B}_{1/k}(p_k,x_k)$ for all $t \geq N_k$, and $(p,x^{(t)}) \in \mathsf{B}_{1/l}(p_l,x_l)$ for all $t \geq N_l$. Therefore, we have that $(p,x^{(t)}) \in \mathsf{B}_{1/k}(p_k,x_k) \cap \mathsf{B}_{1/l}(p_l,x_l)$ for all $t \geq N := \max\{N_k,N_l\}$, which implies the following bound:

$$\begin{aligned} d_{\mathcal{P}\times\mathfrak{X}}((p_k,x_k),(p_l,x_l)) &\leq d_{\mathcal{P}\times\mathfrak{X}}((p_k,x_k),(p,x^{(N_k)})) \\ &\quad + d_{\mathcal{P}\times\mathfrak{X}}((p,x^{(N_k)}),(p,x^{(N)})) \\ &\quad + d_{\mathcal{P}\times\mathfrak{X}}((p,x^{(N)}),(p,x^{(N_l)})) \\ &\quad + d_{\mathcal{P}\times\mathfrak{X}}((p,x^{(N_l)}),(p_l,x_l)) \\ &\leq \frac{1}{k} + \frac{2}{k} + \frac{2}{l} + \frac{1}{l} \leq \frac{3}{k} + \frac{3}{l} \overset{k,l\to\infty}{\to} 0\,. \end{aligned}$$

Hence, by completeness of $\mathcal{P}\times\mathfrak{X}$, the sequence $(p_k,x_k)_{k\in\mathbb{N}}$ has a limit $(p^*,x^*) \in \mathcal{P}\times\mathfrak{X}$. First, we show that $p^* = p$: Since $(p,x^{(N_k)}) \in \mathsf{B}_{1/k}(p_k,x_k)$ for all $k \in \mathbb{N}$, we have by continuity of the metric:

$$\begin{aligned} d_{\mathcal{P}}(p,p^*) &= \lim_{k\to\infty} d_{\mathcal{P}}(p,p_k) \leq \lim_{k\to\infty} d_{\mathcal{P}\times\mathfrak{X}}((p,x^{(N_k)}),(p_k,x_k)) \\ &\leq \lim_{k\to\infty} \frac{1}{k} = 0\,. \end{aligned}$$

Therefore, $(p_k, x_k) \to (p, x^*)$. Second, we show that $(p, x^*) \in \mathsf{A}_{\text{stat}}$, that is, $x^* \in \mathsf{A}_{\text{stat},p}$. For this, assume the contrary, that is, assume that $(p, x^*) \in \mathsf{A}_{\text{stat}}^c$. By Lemma 5.2.1, the set A_{stat} is closed, such that $\mathsf{A}_{\text{stat}}^c$ is open. Hence, there exists $\varepsilon > 0$ with $\mathsf{B}_\varepsilon(p, x^*) \subset \mathsf{A}_{\text{stat}}^c$, and we have $\mathsf{B}_\varepsilon(p, x^*) \cap \mathsf{A}_{\text{stat}} = \emptyset$. Then, since $(p_k, x_k) \to (p, x^*)$, there exists $N \in \mathbb{N}$, such that $d_{\mathcal{P}\times\mathcal{X}}((p_k, x_k), (p, x^*)) < \frac{\varepsilon}{3}$ for all $k \geq N$. Now, taking $k \geq N$ large enough, such that $\frac{1}{k} < \frac{\varepsilon}{3}$, we conclude that

$$\mathsf{B}_{1/k}(p_k, x_k) \cap \mathsf{A}_{\text{stat}} = \emptyset \,.$$

By definition of the sequence $(p_k, x_k)_{k\in\mathbb{N}}$, this is a contradiction, such that $(p, x^*) \in \mathsf{A}_{\text{stat}}$. It remains to show that the sequence $(x^{(t)})_{t\in\mathbb{N}_0}$ converges to x^*. For this, assume the contrary again. Then there exists an $\varepsilon > 0$ with the property that for all $N \in \mathbb{N}$, one can find a $T \geq N$, such that $d_{\mathcal{X}}(x^{(T)}, x^*) \geq \varepsilon$. Then, choose $k \in \mathbb{N}$ large enough, such that $d_{\mathcal{X}}(x_k, x^*) \leq \frac{\varepsilon}{3}$ and $\frac{1}{k} < \frac{\varepsilon}{3}$. Since $(p, x^{(t)}) \in \mathsf{B}_{1/k}(p_k, x_k)$ for all $t \geq N_k$, we get that:

$$d_{\mathcal{X}}(x^{(t)}, x^*) \leq d_{\mathcal{X}}(x^{(t)}, x_k) + d_{\mathcal{X}}(x_k, x^*) \leq \frac{2\varepsilon}{3} < \varepsilon \quad \forall t \geq N_k \,.$$

Again, this is a contradiction and such an $\varepsilon > 0$ cannot exist. Thus $(x^{(t)})_{t\in\mathbb{N}_0}$ converges to $x^* \in \mathsf{A}_{\text{stat},p}$, and we have that $(p, (z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{A}_{\text{conv}}$, which concludes the proof. □

Lemma 5.2.3 *Suppose that Assumptions 5.2.1, 5.2.2, 5.2.3 and 5.2.4 hold. Define the (parametric) set of sequences that satisfy the sufficient-descent condition as*

$$\begin{aligned}\mathsf{A}_{\text{desc}} := \Big\{ &(p, (z^{(t)})_{t\in\mathbb{N}_0}) \in \mathcal{P} \times \mathcal{X}^{\mathbb{N}_0} : \\ &(x^{(t)})_{t\in\mathbb{N}_0} \subset \operatorname{dom} \ell(\cdot, p) \text{ and } \exists a > 0 \text{ s.t. } \forall t \in \mathbb{N}_0 \\ &\ell(x^{(t+1)}, p) + a\|x^{(t+1)} - x^{(t)}\|^2 \leq \ell(x^{(t)}, p)\Big\} \,.\end{aligned}$$

Then A_{desc} *is measurable.*

Proof. Since $\mathbb{Q}$ is dense in $\mathbb{R}$, we can restrict to $a \in (0, +\infty) \cap \mathbb{Q} =: \mathbb{Q}_+$. Then A_{desc} can be written as

$$\left(\bigcup_{a\in\mathbb{Q}_+} \bigcap_{t\in\mathbb{N}_0} \mathsf{A}_{a,t} \right) \cap \left(\bigcap_{t\in\mathbb{N}_0} \left\{ \ell(x^{(t)}, p) < +\infty \right\} \right) ,$$

where $\mathsf{A}_{a,t}$ is given by:

$$\left\{ \ell(x^{(t+1)}, p) + a\|x^{(t+1)} - x^{(t)}\|^2 \leq \ell(x^{(t)}, p) \right\} .$$

In the definition of $\mathsf{A}_{a,t}$, we can w.l.o.g. restrict to $(p, (z^{(t)})_{t\in\mathbb{N}_0})$ with $\ell(x^{(t)}, p) < +\infty$ and $\ell(x^{(t+1)}, p) < +\infty$, since we intersect with the corresponding (measurable) sets anyway.

Since σ-algebras are stable under countable unions/intersection, it suffices to show that the sets $\{\ell(x^{(t)}, p) < +\infty\}$ and $\mathsf{A}_{a,t}$ are measurable. Here, the set $\{\ell(x^{(t)}, p) < +\infty\}$ can be written as:

$$\left\{ \ell(x^{(t)}, p) < +\infty \right\} = \Big(\ell \circ \chi \circ (id, \pi_{\mathcal{X},t}) \Big)^{-1} [0, +\infty) \,,$$

where $\chi : \mathcal{P} \times \mathcal{X} \to \mathcal{X} \times \mathcal{P}$ just interchanges the coordinates (which is measurable), and id is the identity on $\mathcal{P}$. Since $[0, +\infty)$ is a measurable set and ℓ is assumed to be measurable, we have that $\{\ell(x^{(t)}, p) < +\infty\}$

is measurable for each $t \in \mathbb{N}_0$. To show that $\mathsf{A}_{a,t}$ is measurable, we define the function $g_a : (\operatorname{dom} \ell)^2 \to \mathbb{R}$ through $((x_1, p_1), (x_2, p_2)) \mapsto \ell(x_2, p_2) - \ell(x_1, p_1) + a\|x_2 - x_1\|^2$. Then, g_a is measurable and $\mathsf{A}_{a,t}$ can be written as:

$$\begin{aligned}\mathsf{A}_{a,t} &= \left\{g_a(x^{(t)}, p, x^{(t+1)}, p) \leq 0\right\} \\ &= \left\{\left(g_a \circ (\pi_{\mathfrak{X},t}, id, \pi_{\mathfrak{X},t+1}, id) \circ d\right)\left(p, (z^{(t')})_{t' \in \mathbb{N}_0}\right) \leq 0\right\} \\ &= \left(g_a \circ (\pi_{\mathfrak{X},t}, id, \pi_{\mathfrak{X},t+1}, id) \circ d\right)^{-1}(-\infty, 0]\,,\end{aligned}$$

where $d : \mathcal{P} \times \mathfrak{X}^{\mathbb{N}_0} \to (\mathfrak{X}^{\mathbb{N}_0} \times \mathcal{P})^2$ is the diagonal inclusion $(p, z) \mapsto ((z, p), (z, p))$, which is measurable w.r.t. to the product-σ-algebra on $(\mathfrak{X}^{\mathbb{N}_0} \times \mathcal{P})^2$, since $d^{-1}(\mathsf{B}_1 \times \mathsf{B}_2) = \tilde{\mathsf{B}}_1 \cap \tilde{\mathsf{B}}_2$, where $\tilde{\mathsf{B}}_i$ is the same as B_i up to an interchange of the coordinates. Thus, the set $\mathsf{A}_{a,t}$ is measurable, which concludes the proof. □

We proceed with the relative error condition. It involves a union over all subgradients, and therefore might not be measurable. Hence, we restrict to subgradients given through a *measurable selection*, that is, a measurable function $v : \operatorname{dom} \partial_1 \ell \to \mathfrak{X}$, such that $v(x, p) \in \partial_1 \ell(x, p)$ for every $(x, p) \in \operatorname{dom} \partial_1 \ell$. To guarantee its existence, we want to use Lemma 2.3.6. Thus, we need the following result:

Lemma 5.2.4 *Suppose Assumption 5.2.4 holds. Then $(x, p) \mapsto \partial_1 \ell(x, p)$ is closed-valued and measurable.*

Proof. Since $\partial_1 \ell(x, p)$ is the subdifferential of $\ell(\cdot, p)$ at x, by Lemma 2.3.3 we have that, for every $\bar{p} \in \mathcal{P}$ and every $\bar{x} \in \operatorname{dom} \ell(\cdot, \bar{p})$, the set $\partial_1 \ell(\bar{x}, \bar{p})$ is closed. Therefore, $\partial_1 \ell(\bar{x}, \bar{p})$ is closed for every $(\bar{x}, \bar{p}) \in \operatorname{dom} \ell$. Further, for $(\bar{x}, \bar{p}) \notin \operatorname{dom} \ell$, we have $\partial_1 \ell(\bar{x}, \bar{p}) = \emptyset$, which is closed, too. Therefore, $(\bar{x}, \bar{p}) \mapsto \partial_1 \ell(\bar{x}, \bar{p})$ is closed-valued. Finally, since $(\bar{x}, \bar{p}) \mapsto \partial_1 \ell(\bar{x}, \bar{p})$ is also outer semi-continuous, Lemma 2.3.5 implies that $\partial_1 \ell$ is measurable w.r.t. $\mathfrak{B}(\mathfrak{X} \times \mathcal{P})$. □

Corollary 5.2.5 *Suppose Assumption 5.2.4 holds. Then there exists a measurable selection for $\partial_1 \ell$, that is, a measurable map $v : \operatorname{dom} \partial_1 \ell \to \mathfrak{X}$, such that $v(x, p) \in \partial_1 \ell(x, p)$ for all $(x, p) \in \mathfrak{X} \times \mathcal{P}$.*

Proof. The result follows directly by combining Lemma 5.2.4 with Lemma 2.3.6. □

Lemma 5.2.6 *Suppose that Assumptions 5.2.1, 5.2.2, 5.2.3 and 5.2.4 hold. Denote the measurable selection from Corollary 5.2.5 by v, and define the (parametric) set of sequences that satisfy the relative-error condition as*

$$\begin{aligned}\mathsf{A}_{\text{err}} := \Big\{&(p, (z^{(t)})_{t \in \mathbb{N}_0}) \in \mathcal{P} \times \mathfrak{Z}^{\mathbb{N}_0} : \\ &(p, x^{(t)}) \in \operatorname{dom} \partial_1 \ell \ \forall t \in \mathbb{N}_0 \text{ and } \exists b > 0 \text{ s.t. } \forall t \in \mathbb{N}_0 \\ &\|v(x^{(t+1)}, p)\| \leq b\|x^{(t+1)} - x^{(t)}\|\Big\}\,.\end{aligned}$$

Then A_{err} is measurable.

Proof. Again, we can restrict to $b \in \mathbb{Q} \cap (0, +\infty) =: \mathbb{Q}_+$. Thus, A_{err} can be written as:

$$\mathsf{A}_{\text{err}} = \left(\bigcup_{b \in \mathbb{Q}_+} \bigcap_{t \in \mathbb{N}_0} \mathsf{B}_{b,t} \right) \cap \left(\bigcap_{t \in \mathbb{N}_0} \{(x^{(t)}, p) \in \operatorname{dom} \partial_1 \ell\} \right),$$

where $\mathsf{B}_{b,t}$ is given by:

$$\mathsf{B}_{b,t} := \{ \|v(x^{(t+1)}, p)\| \leq b \|x^{(t+1)} - x^{(t)}\| \}.$$

Hence, since σ-algebras are stable under countable unions/intersections, we only have to show measurability of the sets $\mathsf{B}_{b,t}$ and $\{(x^{(t)}, p) \in \operatorname{dom} \partial_1 \ell\}$. Here, we have:

$$\{(x^{(t)}, p) \in \operatorname{dom} \partial_1 \ell\} = \left(\chi \circ (id, \pi_{\mathfrak{X},t}) \right)^{-1} (\operatorname{dom} \partial_1 \ell),$$

where, again, $\chi : \mathcal{P} \times \mathfrak{X} \to \mathfrak{X} \times \mathcal{P}$ just interchanges the coordinates (which is measurable), and id is the identity on $\mathcal{P}$. By Lemma 5.2.4, $\operatorname{dom} \partial_1 \ell$ is measurable, such that $\{(x^{(t)}, p) \in \operatorname{dom} \partial_1 \ell\}$ is measurable for each $t \in \mathbb{N}_0$. Thus, it remains to show the measurability of the sets $\mathsf{B}_{b,t}$. For this, introduce the function $g_b : (\operatorname{dom} \partial_1 \ell)^2 \to \mathbb{R}$, $((x_1, p_1), (x_2, p_2)) \mapsto \|v(x_2, p_2)\| - b\|x_2 - x_1\|$. Since v is measurable and the norm is continuous, we have that g_b is measurable. Then, we can write the set $\mathsf{B}_{b,t}$ as:

Similar to before, in the definition of $\mathsf{B}_{b,t}$ we can restrict to $(p, (z^{(t)})_{t \in \mathbb{N}_0})$ such that $(x^{(t)}, p), (x^{(t+1)}, p) \in \operatorname{dom} \partial_1 \ell$.

$$\begin{aligned} \mathsf{B}_{b,t} &= \{ g_b(x^{(t)}, p, x^{(t+1)}, p) \leq 0 \} \\ &= \{ (g_b \circ (\pi_{\mathfrak{X},t}, id, \pi_{\mathfrak{X},t+1}, id) \circ d)(p, (z^{(t')})_{t' \in \mathbb{N}_0}) \leq 0 \} \\ &= (g_b \circ (\pi_{\mathfrak{X},t}, id, \pi_{\mathfrak{X},t+1}, id) \circ d)^{-1}(-\infty, 0], \end{aligned}$$

where $d : \mathcal{P} \times \mathfrak{Z}^{\mathbb{N}_0} \to (\mathfrak{Z}^{\mathbb{N}_0} \times \mathcal{P})^2$ is again the diagonal inclusion. Hence, also the sets $\mathsf{B}_{b,t}$ are measurable for each $t \in \mathbb{N}_0$ and $b \in \mathbb{Q}_+$, which concludes the proof. □

Lemma 5.2.7 *Assume that Assumptions 5.2.1, 5.2.2 and 5.2.4 hold. Define the set of bounded sequences as:*

$$\tilde{\mathsf{A}}_{\text{bound}} = \left\{ (z^{(t)})_{t \in \mathbb{N}_0} \in \mathfrak{Z}^{\mathbb{N}_0} \ : \exists c \geq 0 \ s.t. \ \|x^{(t)}\| \leq c \ \forall t \in \mathbb{N}_0 \right\}.$$

Then $\mathsf{A}_{\text{bound}} := \mathcal{P} \times \tilde{\mathsf{A}}_{\text{bound}}$ *is measurable.*

Proof. By definition of the product-σ-algebra, it suffices to show that $\tilde{\mathsf{A}}_{\text{bound}}$ is measurable. Then, again, it suffices to consider $c \in [0, +\infty) \cap \mathbb{Q} := \mathbb{Q}_+$, and we can write $\tilde{\mathsf{A}}_{\text{bound}}$ as:

$$\tilde{\mathsf{A}}_{\text{bound}} = \bigcup_{c \in \mathbb{Q}_+} \underbrace{\bigcap_{t \in \mathbb{N}_0} \{(z^{(t)})_{t \in \mathbb{N}_0} \in \mathfrak{Z}^{\mathbb{N}_0} \ : \ \|x^{(t)}\| \leq c\}}_{=: \mathsf{C}_{c,t}}.$$

Hence, it suffices to show that the sets $\mathsf{C}_{c,t}, c \in \mathbb{Q}_+, t \in \mathbb{N}_0$, are measurable. By defining $g(x) := \|x\|$, this follows directly from the equality $\mathsf{C}_{c,t} = (g \circ \pi_{\mathfrak{X},t})^{-1}[0, c]$. □

5.2.2 Convergence to Stationary Points

We are now in a position to derive the main result of this chapter.

Corollary 5.2.8 *Suppose that Assumptions 5.2.1, 5.2.2, 5.2.3, and 5.2.4 hold. Furthermore, assume that $\ell(\cdot, p)$ is a Kurdyka-Łojasiewicz function for every $p \in \mathcal{P}$. Then the sets $\mathsf{A}_{\text{desc}} \cap \mathsf{A}_{\text{err}} \cap \mathsf{A}_{\text{bound}} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ and $\mathsf{A}_{\text{conv}} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ are measurable, and it holds that:*

$$\mathsf{A}_{\text{desc}} \cap \mathsf{A}_{\text{err}} \cap \mathsf{A}_{\text{bound}} \subset \mathsf{A}_{\text{conv}} .$$

Proof. Take $(p, (z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{A}_{\text{desc}} \cap \mathsf{A}_{\text{err}} \cap \mathsf{A}_{\text{bound}}$. Then, we have that the corresponding sequence of iterates $(x^{(t)})_{t\in\mathbb{N}_0}$ satisfies the sufficient-descent condition and the relative-error condition for $\ell(\cdot, p)$, and it is bounded. Further, $(x^{(t)})_{t\in\mathbb{N}_0}$ also satisfies the continuity condition, since we have $(x^{(t)})_{t\in\mathbb{N}_0} \subset \operatorname{dom} \ell(\cdot, p)$ and ℓ is assumed to be continuous on its domain. Therefore, Theorem 2.3.8 implies that $(x^{(t)})_{t\in\mathbb{N}_0}$ converges to a stationary point of $\ell(\cdot, p)$. Hence, there exists $x^* \in \mathsf{A}_{\text{stat},p}$, such that $\lim_{t\to\infty} \|x^{(t)} - x^*\| = 0$, and we conclude that $(p, (z^{(t)})_{t\in\mathbb{N}_0}) \in \mathsf{A}_{\text{conv}}$. □

In particular, if μ is a (probability) measure on $\mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$, for example, $\mu = \mathbb{P}_{(P,Z)|H=h}$ for a given $h \in \mathcal{H}$, by the monotonicity of measures it holds that:

$$\mu\{\mathsf{A}_{\text{desc}} \cap \mathsf{A}_{\text{err}} \cap \mathsf{A}_{\text{bound}}\} \leq \mu\{\mathsf{A}_{\text{conv}}\} .$$

This idea yields our main theorem for this chapter:

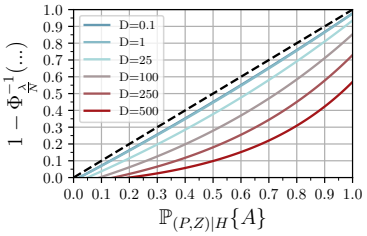

Figure 5.3: Lower bound on the convergence probability for $N = 1000$ and $\varepsilon = 0.01$ depending on the divergence term $D := D_{\text{KL}}(\mathbb{Q} \parallel \mathbb{P}_H)$.

Theorem 5.2.9 *Suppose that Assumptions 5.2.1, 5.2.2, 5.2.3, and 5.2.4 hold. Further, assume that $\ell(\cdot, p)$ is a Kurdyka–Łojasiewicz function for every $p \in \mathcal{P}$. Abbreviate $\mathsf{A} := \mathsf{A}_{\text{desc}} \cap \mathsf{A}_{\text{err}} \cap \mathsf{A}_{\text{bound}}$. Then, for $\lambda \in (0, +\infty)$, it holds that:*

$$\mathbb{P}_S\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) \;:\; \mathbb{Q}\left\{\mathbb{P}_{(P,Z)|H}\left\{\mathsf{A}_{\text{conv}}\right\}\right\} \geq 1 - \Phi^{-1}_{\frac{\lambda}{N}}\Big(\frac{1}{N}\sum_{n=1}^{N} \mathbb{Q}\left\{\mathbb{P}_{(P_n,Z_n)|H,P_n}\{\mathsf{A}^c\}\right\} + \frac{D_{\text{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{1}{\varepsilon}\right)}{\lambda}\Big)\Big\} \geq 1 - \varepsilon .$$

Proof. By taking the complementary events in Corollary 5.2.8, we have $\mathbb{P}_H$-a.s.:

$$\mathbb{P}_{(P,Z)|H}\{\mathsf{A}^c\} \geq 1 - \mathbb{P}_{(P,Z)|H}\left\{\mathsf{A}_{\text{conv}}\right\} .$$

Additionally, for any measurable set $\mathsf{B} \subset \mathcal{P} \times \mathcal{Z}^{\mathbb{N}_0}$ and $\lambda \in (0, +\infty)$, we have by Theorem 4.4.10:

$$\mathbb{P}_S\Big\{\forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) \;:\; \mathbb{Q}\{\mathbb{P}_{(P,Z)|H}\left\{\mathsf{B}\right\}\} \leq \Phi^{-1}_{\frac{\lambda}{N}}\Big(\frac{1}{N}\sum_{n=1}^{N} \mathbb{Q}\left\{\mathbb{P}_{(P_n,Z_n)|H,P_n}\left\{\mathsf{B}\right\}\right\} + \frac{D_{\text{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{1}{\varepsilon}\right)}{\lambda}\Big)\Big\} \geq 1 - \varepsilon .$$

Hence, using $\mathsf{B} := \mathsf{A}^c$, inserting the inequality above, and rearranging the terms yields the result. □

Remark 5.2.1 (i) Actually, the lower bound holds for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$. Further, since we do not know the difference $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}} \setminus \mathsf{A}\}$,

we do not know how tight this bound is for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}$.

(ii) We want to stress the following: We do not assume that the conditions of Theorem 2.3.8, for example, the sufficient-descent condition, do hold *per se*. This theorem is rather about the fact that, based on our observations during training, we can deduce *how often* these conditions will hold on unseen problem instances.

(iii) For large N, if λ is chosen correctly, we can approximate $\Phi^{-1}_{\frac{\lambda}{N}}(p) \approx p$. Assuming this holds, and abbreviating the empirical approximation as $\hat{\mathbb{P}}_{(P,Z)|H} := \frac{1}{N}\sum_{n=1}^{N} \mathbb{P}_{(P_n,Z_n)|H,P_n}$, the inequality reads:

$$\mathbb{Q}\left\{\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}\right\} \geq \mathbb{Q}\left\{\hat{\mathbb{P}}_{(P,Z)|H}\{\mathsf{A}\}\right\} + \frac{D_{\text{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{1}{\varepsilon}\right)}{\lambda}.$$

This is intuitive: For larger N, we have more confidence in our estimate $\hat{\mathbb{P}}_{(P,Z)|H}\{\mathsf{A}\}$, so we can choose a larger λ which dampens the last term and tightens the lower bound for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}$.

5.3 Experiments

In this section, we conduct two experiments: The strongly convex and smooth problem of minimizing quadratic functions with varying strong convexity, varying smoothness, and varying right-hand side, and the non-smooth non-convex problem of training a neural network on different data sets.

5.3.1 Training of the Algorithm

For training, we mainly use the procedure proposed in Chapters 3 and 4. For completeness, we briefly summarize it here again: In the outer loop, we sample a loss-function $\ell(\cdot, p)$ randomly from the training set. Then, in the inner loop, we train the algorithm on this loss-function with ℓ_{train} given by

$$\ell_{\text{train}}(h, p, z^{(t)}) = \mathbb{1}\{\ell(\pi_{\mathfrak{X}}(z^{(t)}), p) > 0\} \frac{\ell(\pi_{\mathfrak{X}}(z^{(t+1)}), p)}{\ell(\pi_{\mathfrak{X}}(z^{(t)}), p)} \cdot \mathbb{1}_{\mathsf{C}^c}(p, z^{(t)}),$$

where C is the convergence set, which we define as $\mathsf{C}_{\text{quad}} := \{(p, z) \in \mathcal{P} \times \mathcal{Z} \;:\; \ell(\pi_{\mathfrak{X}}(z), p) < 10^{-16}\}$ for the quadratic functions and as $\mathsf{C}_{\text{nn}} := \{(p, z) \in \mathcal{P} \times \mathcal{Z} \;:\; \ell(\pi_{\mathfrak{X}}(z), p) < 0.75\}$ for the neural networks. Thus, in each iteration the algorithm computes a new point and observes the loss ℓ_{train}, which is used to update its hyperparameters. We run this procedure for $150 \cdot 10^3$ iterations. This yields hyperparameters $h^{(0)} \in \mathcal{H}$, such that $\mathcal{A}(h^{(0)}, \cdot, \cdot)$ has a good performance. However, typically, it is not a descent method yet, that is, $\mathbb{P}_{(P,Z)|H=h^{(0)}}\{\mathsf{A}\}$ is small, such that the predicted bound would not be meaningful. Therefore, we employ the probabilistic constraining procedure of Chapter 3 in a progressive way: Starting from $h^{(0)}$, we try to find a sequence of hyperparameters $h^{(1)}, h^{(2)}, \ldots$, such that

$$\mathbb{P}_{(P,Z)|H=h^{(0)}}\{\mathsf{A}\} < \mathbb{P}_{(P,Z)|H=h^{(1)}}\{\mathsf{A}\} < \mathbb{P}_{(P,Z)|H=h^{(2)}}\{\mathsf{A}\} < \ldots.$$

Since we use a deterministic algorithm, we omit the variable r here.

Remark 5.3.1 Please recall that $(h, \mathsf{A}) \mapsto \mathbb{P}_{(P,Z)|H=h}\{\mathsf{A}\}$ is a given probability kernel and H is the identity on $\mathscr{H}$, such that this notation is correct, even if we did not construct the final prior distribution yet.

For this, we test the probabilistic constraint every 1000 iterations, that is: Given $h^{(i)}$, we train the algorithm (as before) for another 1000 iterations, which yields a candidate $\tilde{h}^{(i+1)}$. If $\mathbb{P}_{(P,Z)|H=h^{(i)}}\{\mathsf{A}\} < \mathbb{P}_{(P,Z)|H=\tilde{h}^{(i+1)}}\{\mathsf{A}\}$, we accept $h^{(i+1)} := \tilde{h}^{(i+1)}$, otherwise we reject it and start again from $h^{(i)}$. Finally, this yields some hyperparameters h_0 that have a good performance and such that $\mathbb{P}_{(P,Z)|H=h_0}\{\mathsf{A}\}$ is large enough (here: about 90%). Then, starting from h_0, we construct the *actual* discrete prior distribution $\mathbb{P}_H$ over points $h_1, \ldots, h_{n_{\text{sample}}} \in \mathscr{H}$, by a sampling procedure (as before). Finally, we perform the (closed-form) PAC-Bayesian optimization step, which yields the posterior $\mathbb{Q}^* \in \mathcal{M}_1(\mathbb{P}_H)$. In the end, for simplicity, we set the hyperparameters to

$$h^* = \underset{i=1,\ldots,n_{\text{sample}}}{\arg\max} \; \mathbb{Q}^*\{h_i\} \, .$$

For the construction of the prior, we use $N_{\text{prior}} = 500$ functions, for the probabilistic constraint we use $N_{\text{val}} = 500$ functions, and for the PAC-Bayesian optimization step we use $N_{\text{train}} = 250$ functions, all of which are sampled i.i.d., that is, the data sets are independent of each other.

Remark 5.3.2 Training the algorithm to yield a good performance is comparably easy. On the other hand, turning it into an algorithm, such that $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ is large enough (in our case: a descent method without enforcing it) is challenging and, unfortunately, not guaranteed to work. Nevertheless, it is key to get a meaningful guarantee.

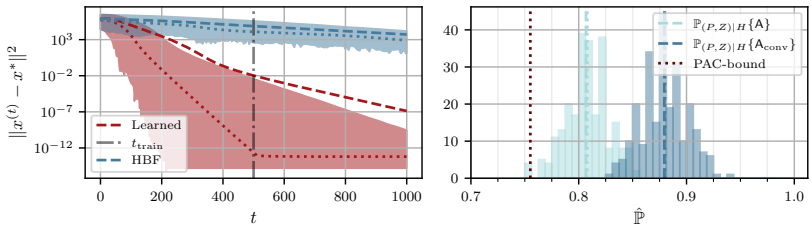

Figure 5.4: The left figure shows the distance to the minimizer. Here, dashed lines represent the mean, dotted lines represent the median, and the shaded regions indicates 95% of the test data. The learned algorithm is shown in red and *heavy-ball with friction* (HBF) in blue. The right plot shows the estimates (dashed lines) for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ (light blue), $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}$ (dark blue), and the PAC-bound (brown). One can see that the predicted chain of inequalities $1 - \Phi^{-1}(...) \leq \mathbb{P}_{(P,Z)|H}\{\mathsf{A}\} \leq \mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}$ does hold true.

5.3.2 Quadratic Problems

First, we train the algorithm $\mathcal{A}$ to solve quadratic problems. Thus, each optimization problem $\ell(\cdot, p)$ is of the form

$$\min_{x \in \mathbb{R}^d} \frac{1}{2}\|Ax - b\|^2, \quad A \in \mathbb{R}^{d \times d},\ b \in \mathbb{R}^d,$$

such that the parameters are given by $p = (A, b) \in \mathbb{R}^{d^2+d} =: \mathcal{P}$, and the optimization variable is $x \in \mathbb{R}^d$, $d = 200$. The strong-convexity and smoothness constants of ℓ are sampled randomly in the intervals $[m_-, m_+], [L_-, L_+] \subset (0, +\infty)$, and we define the matrix A_j, $j = 1, ..., N$, as *diagonal matrix* with entries $a_{ii}^j = \sqrt{m_j} + i(\sqrt{L_j} - \sqrt{m_j})/d$, $i = 1, ..., d$. In principle, this is a severe restriction. However, we do not use this knowledge explicitly in the design of our algorithm $\mathcal{A}$, that is, if the algorithm "finds" this structure during training by itself, it can leverage on it. Like this, the given class of functions is L_+-smooth and m_--strongly convex, such that we use *heavy-ball with friction* (HBF; Polyak, 1964) as worst-case optimal baseline. Its update is given by $x^{(t+1)} = x^{(t)} - \alpha \nabla f(x^{(t)}) + \beta\left(x^{(t)} - x^{(t-1)}\right)$, where the optimal worst-case convergence rate is attained for $\alpha = \left(\frac{2}{\sqrt{L_+}+\sqrt{\mu_-}}\right)^2, \beta = \left(\frac{\sqrt{L_+}-\sqrt{\mu_-}}{\sqrt{L_+}+\sqrt{\mu_-}}\right)^2$ (Nesterov, 2018). Similarly, the learned algorithm $\mathcal{A}$ performs an update of the form $x^{(t+1)} = x^{(t)} + \alpha^{(t)} d^{(t)}$, where $\alpha^{(t)}$ and $d^{(t)}$ are predicted by separate blocks of a neural network, which is visualized in Figure 5.5. Here, we stress that the update is not constrained in any way. For more details on the update step, especially the used features $d_1^{(t)}, d_2^{(t)}, s_1^{(t)}, ..., s_4^{(t)}$, we refer to Appendix C.1. Since the functions are smooth and strongly convex, we only have to check the sufficient-descent condition and the relative-error condition. Obviously, in practice it is impossible to check them for all $t \in \mathbb{N}_0$. Thus, we restrict to $t_{\text{train}} = 500$ iterations. Then, given a measurable selection $v(x, p) \in \partial_1 \ell(x, p)$, the relative-error condition is trivially satisfied with $b := \max_{t \leq t_{\text{train}}}\{\|v(x^{(t)}, p)\|\} / \min_{t \leq t_{\text{train}}} \|x^{(t)} - x^{(t-1)}\|$, such that we only have to check the sufficient-descent condition during training. Finally, we consider ζ to be converged, if the loss is smaller than 10^{-16}.

The results are shown in Figure 5.4: The left plot shows the distance to the minimizer x^* over the iterations, where HBF is shown in blue and the learned algorithm in red, and we can see that the learned algorithm is clearly superior. The right plot shows the estimated probabilities $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ (light blue dashed line), $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}}\}$ (dark blue dashed line), and the PAC-bound (brown dotted line) on 250 test sets of size $N = 250$. We can see that the PAC-bound is quite tight for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$, while there is a substantial gap $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}_{\text{conv}} \setminus \mathsf{A}\}$. Nevertheless, it *guarantees* convergence of the algorithm in about 75% of the test problems.

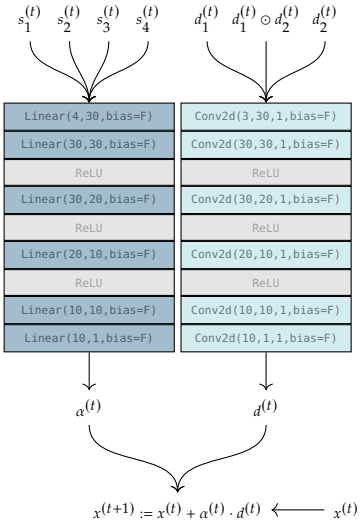

Figure 5.5: Algorithmic update $\mathcal{A}$ for the experiment on quadratic functions: The directions $d_1^{(t)}$, $d_2^{(t)}$ and $d_1^{(t)} \odot d_2^{(t)}$ are inserted as different channels into the `Conv2d`-block, which performs 1×1 "convolutions", that is, the algorithm acts coordinate-wise on the input. The scales $s_1^{(t)}, ..., s_4^{(t)}$ get transformed separately by the fully-connected block.

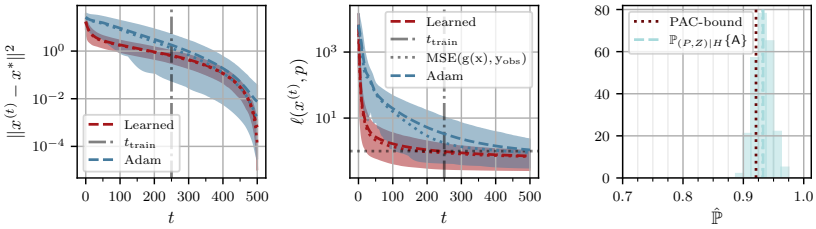

Figure 5.6: The left figure shows the distance to the estimated critical point and the figure in the middle shows the loss. Adam is shown in blue and the learned algorithm in red. The mean and median are shown as dashed and dotted lines, respectively, while the shaded region represents 95% of the test data. We see that the learned algorithm minimizes the loss faster than Adam, and seems to converge to a critical point. The right plot shows the estimate for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ (light blue dashed line) and the PAC-bound (brown).

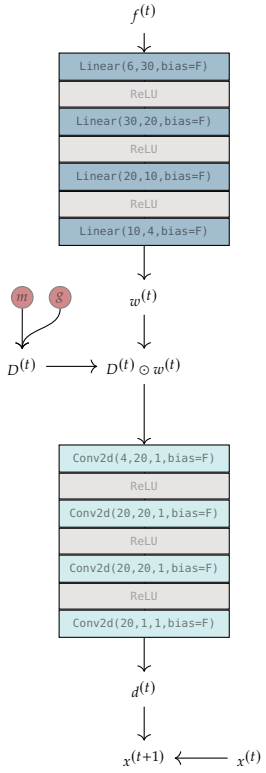

Figure 5.7: Algorithmic update $\mathcal{A}$ for training the neural network (similar to before): Based on the features $f^{(t)}$, the first block computes a weight vector $w^{(t)}$ and a step-size $s^{(t)}$. Then, $w^{(t)}$ is used to weigh the different directions in $D^{(t)}$ before they get inserted as channels into the second block, which consists of 1x1-convolutional layers, and computes the update direction $d^{(t)}$. Finally, we update $x^{(t+1)} = x^{(t)} + d^{(t)}/\sqrt{t}$.

5.3.3 Training a Neural Network

As second experiment, we train the algorithm $\mathcal{A}$ to train a neural network $\mathtt{N}$ on a regression problem. Thus, the algorithm $\mathcal{A}$ predicts parameters $\theta \in \mathbb{R}^m$, such that $\mathtt{N}(\theta, \cdot)$ estimates a function $g : \mathbb{R} \to \mathbb{R}$ from noisy observations $y_{i,j} = g_i(x_j) + \varepsilon_{i,j}$, $i = 1, ..., N$, $j = 1, ..., K$ ($K = 50$), with $\varepsilon_{i,j} \overset{iid}{\sim} \mathcal{N}(0, 1)$. Here, we use the mean square error as loss for the neural network, and for $\mathtt{N}$ we use a fully-connected two layer neural network with ReLU-activation functions. Then, by using the data sets as parameters, that is, $\mathcal{P} = \mathbb{R}^{K\times 2}$ and $p_i = \{(x_{i,j}, y_{i,j})\}_{j=1}^K$, the loss functions for the algorithm are given by $\ell(\theta, p_i) := \frac{1}{K}\sum_{j=1}^K (\mathtt{N}(\theta, x_{i,j}) - y_{i,j})^2$, which are non-smooth and non-convex in θ. Here, the input x is transformed into the vector $(x, x^2, ..., x^5)$, such that the parameters $\theta \in \mathbb{R}^m$ of the neural network are given by the weights $A_1 \in \mathbb{R}^{50\times 5}$, $A_2 \in \mathbb{R}^{1\times 50}$ and biases $b_1 \in \mathbb{R}^{50}$, $b_2 \in \mathbb{R}$ of the two fully-connected layers. Thus, the optimization space is of dimension $m = 351$. For the functions g_i we use polynomials of degree 5, where we sample the coefficients $(c_{i,0}, ..., c_{i,5})$ uniformly in $[-5, 5]$. Similarly, we sample the points $\{x_{i,j}\}_{j=1}^K$ uniformly in $[-2, 2]$.
As baseline we use Adam (Kingma et al., 2015) as it is implemented in PyTorch (Paszke et al., 2019), and we tune its step-size with a simple grid search over 100 values in $[10^{-4}, 10^{-2}]$, such that its performance is best for the given $t_{\text{train}} = 250$ iterations. This yields the value $\kappa = 0.008$. Note that we use Adam in the "full-batch setting" here, while, originally, it was introduced for the stochastic case. On the other hand, the learned algorithm performs the update $x^{(t+1)} = x^{(t)} + d^{(t)}/\sqrt{t}$, where $d^{(t)}$ is predicted by a neural network, which is visualized in Figure 5.7. Again, we stress that $d^{(t)}$ is not constrained in any way. For more details on the update, especially the features $f^{(t)}$ and the directions $D^{(t)}$, we refer to Appendix C.2. Finally, as we cannot access the critical points directly, we approximate them by running gradient descent for $5 \cdot 10^4$ iterations with a step-size of $1 \cdot 10^{-6}$, starting for each problem and algorithm from the last iterate ($t = 500$). Similarly, we cannot estimate the convergence probability in this case, only the probability for the event A.
The results of this experiment are shown in Figure 5.6: The left plot shows the distance to the critical point and the plot in the middle shows the loss. Here, Adam is shown in blue, while the learned algorithm is shown in red. Finally, the right plot shows the estimate for $\mathbb{P}_{(P,Z)|H}\{\mathsf{A}\}$ and the predicted PAC-bound. We can see that the learned algorithm does indeed seem to converge to a critical point and it minimizes the loss faster than Adam. Further, the PAC-bound is quite tight, and it *guarantees* that the learned algorithm will converge in about 92% of the problems.

5.4 Discussion and Limitations

We presented a novel method for deducing convergence of generic learned algorithms that exhibit a Markovian structure with high probability. To showcase the idea, we derived a new convergence result for learned optimization algorithms on (possibly) non-smooth non-convex loss-functions based on generalization. This was based on the fundamental insight that, contrary to traditional optimization, in learning-to-optimize we can actually *observe* the algorithm during training. While the approach is theoretically sound, practically it has at least four drawbacks, on which we shortly want to comment: First, and foremost, one simply cannot observe the *whole* trajectory in practice. Thus, one can only obtain an approximation to this result, that is, whether the used conditions do hold up to a certain number of iterations. Nevertheless, by using sufficiently many iterations, one can guarantee that the algorithm gets sufficiently close to a critical point. Second, instead of verifying the conditions used here, one could alternatively try to observe the final result directly, for example by looking at the norm of the gradient. However, when checking the proposed conditions one is guaranteed to get arbitrary close to a critical point, while, in the other case, one could end up with a small gradient norm that is arbitrary far away from a critical point. This applies in particular to the non-smooth setting, where the subdifferential does not necessarily tell anything about the distance to a critical point[1], or to applications where one simply cannot access critical points during training. Third, for now, training the algorithm in such a way that it actually does satisfy the proposed properties on a majority of problems is quite difficult and time-consuming. Lastly, due to the sufficient-descent condition, Theorem 2.3.8 is not well-suited for stochastic optimization. Nevertheless, Theorem 4.4.10 and the proposed approach can directly be transferred to the stochastic setting, which we leave for future work.

1: For example, consider $f(x) = |x|$.

Part III

Conclusion

Discussion and Outlook 6

This chapter summarizes the main contributions of this thesis and discusses a few open problems in the context of our proposed model. Finally, it concludes this manuscript by proposing possible directions for future research.

6.1 Summary

Applying machine learning techniques in the field of optimization has great potential and is just at the beginning of its development. In the course of this development, however, we should not lose sight of the fact that optimization algorithms have exactly one task:

They should solve optimization problems.

And similar to how large language models can fool the practitioner into believing a well-formulated yet hallucinated answer, a learned optimization algorithm might only seem to work and, again, might fool the practitioner into believing the final output. This concern was perfectly illustrated in the field of medical image reconstruction by Möller et al. (2019) when they showed that, although the trained optimization algorithm produced the visually more pleasing images, it neglected the crucial details, such that the potentially malign cancerous tissue was not even visible in the final result. Therefore:

Optimization algorithms have to come with theoretical guarantees.

This thesis has made several contributions to improve the theoretical understanding of learned optimization methods, to provide new kinds of theoretical guarantees for them, and to bridge the gap between conventional optimization theory and learning-to-optimize.

A Markovian Model

We proposed an abstract, yet precise model for optimization algorithms and how practitioners use them. This model rests on a single simple equation, which applies to many, if not most, optimization algorithms in practice. Similarly, it requires minimal assumptions, which makes it applicable to a wide variety of applications. The main underlying idea is that optimization algorithms are applied iteratively, such that, if we summarize the update step into a single mapping on a possibly higher-dimensional state space, we obtain the functional equation of a Markov process, which opens the door to this rich theory. Likewise, since optimization algorithms are usually stopped based on a local condition, we formulated the procedure of stopping an optimization algorithm as the process of entering a corresponding set. This in turn allows for treating the time when the algorithm gets stopped as the corresponding hitting time, which is another well-studied object in the theory of stochastic processes. Taken together, this model allows us to

consider a variety of performance measures and procedures to stop the algorithm, and all of them based on the whole trajectory generated by the algorithm. However, while this model covers a wide range of optimization algorithms and problem instances, it obviously will not cover all of them. Nevertheless, as we have built our model in a bottom-up manner, it should be comparably straightforward to incorporate the corresponding changes in many cases.

PAC-Bayesian Guarantees

We have provided PAC-Bayesian generalization guarantees for several different interesting performance measures: the loss after a fixed number of iterations, the convergence time, the (estimated) convergence rate, and general properties of the trajectory. In doing so, we have extended the fields of application of the PAC-Bayesian theory to also include learning-to-optimize, and we have shown how numerous aspects of learning-to-optimize are amenable to generalization guarantees. Especially, most of these guarantees rely on the same mild assumptions as our model for the algorithm, such that they are widely applicable. Even more so, these results showcase that we are not as much restricted by analytical tractability in learning-to-optimize as we are in conventional optimization theory, which opens the door for new and interesting ways to quantify the performance of an optimization algorithm.

A New Learning Procedure

Based on the presented theory, we have developed and subsequently refined a learning procedure that allows for learning optimization algorithms with PAC-Bayesian generalization guarantees. In doing so, we have proposed several design choices that make the procedure flexible and empirically improve the performance quite significantly. We highlight two of these ideas:

(i) First, and foremost, we have proposed a new loss-function for training optimization algorithms: the ratio of the objective values of consecutive iterates or, if available, the ratio of the corresponding sub-optimality gaps.[1] Empirically, this new loss-function greatly improves the performance of the algorithm and also allows for learning algorithms that work well on many different orders of magnitude, that is, they do not plateau just because they have reached comparably small objective values. This idea is based on the fact that the algorithm should perform updates that contract the loss-function independently of its current value, that is, the progress is measured in relative instead of absolute terms, and that the corresponding feedback, the loss during training, should not be obscured by other external changes like a differing initialization.

(ii) This new loss-function also enabled us to train the algorithm in a different, highly efficient, and largely independent way: Similar to reinforcement learning, the algorithm is treated as an agent that moves in the optimization space in search for the minimizer. At a given point, it performs an update and observes as loss the ratio of the previous and the new objective value (or, if available, the ratio of the sub-optimality gap), which therefore quantifies how well the performed update at the given point actually turned out to be. Thus, except for the objective function and the algorithm, in many cases no other external signal or resource is needed, and

1: Just as the motivational quote teaches us: *Compare yourself to who you were yesterday, not to someone else today.*

also the loss for training can be computed on-the-fly, that is, in a self-supervised manner.

Bridging the Gap to Conventional Optimization Theory

Finally, we have proposed a way to transfer known convergence theorems from conventional optimization theory into learning-to-optimize by treating the properties, which are necessary to deduce the result, as measurable sets in the space of trajectories of the algorithm. This change of perspective is based on the fundamental insight that the key difference between conventional optimization theory and learning-to-optimize lies in the different amounts of what we know (or can assume) compared to what we can observe. Doing so, the logical conclusion of a theorem gets transformed into a corresponding inclusion of sets, which enables to treat such statements with probabilistic methods. This resulted in a generalization guarantee for the learned optimization algorithm which certifies that, roughly speaking, the probability that the learned algorithm will converge to a stationary point of the loss function is lower-bounded by corresponding quantities that are observable during training. However, we want to stress again that the underlying idea is not restricted to optimization algorithms and convergence, rather it applies more generally to sequential prediction algorithms with a Markovian structure and all sorts of "desirable properties".

6.2 Problems

While changing the perspective from strong assumptions and deductive reasoning to mild assumptions and observational data enables to build many new algorithms with fundamentally different properties and guarantees, it also highlights one of the main limitations of this statistical approach, which we coin "observational tractability": In some cases it is simply not possible to observe the needed quantities in practice. In fact, most of the problems discusses below are related to the data and can be explained based on our model.

Some Statistics are Intractable

First, and foremost, we simply cannot observe the whole trajectory in practice, that is, we will always have to stop the algorithm after some time $T \in \mathbb{N}_0$. Therefore, some guarantees can only be approximated and some others are completely infeasible. For example, convergence to a stationary point can in some cases be approximated by a sufficiently small gradient norm, which is observable in finite time, whereas full convergence of the sequence of iterates is simply not observable, because it is an asymptotic notion and thus belongs to the so-called *terminal events* (sometimes also called limiting or asymptotic events). In general, all events that belong to the *terminal σ-algebra* corresponding to the iterates $(Z^{(t)})_{t\in\mathbb{N}_0}$ are not directly observable in practice, because, by definition, these are events whose outcome does not change by changing any finite number of iterates. Thus, whenever we observe only a finite number of iterates, we cannot judge about such asymptotic events in a straightforward way. This subtle problem, however, underlines again why our new proof-strategy is so crucial.

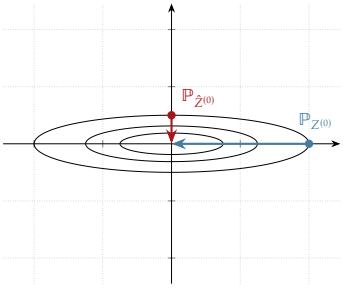

Figure 6.1: Counterexample for changing the initialization: Consider the problem of learning the step-size of gradient descent on quadratic functions. If the initial distribution $\mathbb{P}_{Z^{(0)}}$ lies on the axis corresponding to the eigenvector of the smallest eigenvalue μ, the optimal choice is $h^* = \frac{1}{\mu}$. Then, however, changing the initial distribution to $\mathbb{P}_{\hat{Z}^{(0)}}$ is not possible while keeping the guarantees. In fact, nearly any other initial point would lead to divergence.

Changing the Initialization

Changing the initialization after training while keeping the guarantees is not straightforward to do. To see this, please recall Equation (4.2), which describes the finite-dimensional distributions of the process Z for a given hyperparameter $h \in \mathcal{H}$ and parameter $p \in \mathcal{P}$:

$$\psi_{h,p} \circ \pi_J^{-1} = \mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{K-1} \gamma_{t_{k+1}-t_k}(h, p, \cdot)\,.$$

In particular, it shows that the distribution of the trajectory depends on the initial distribution $\mathbb{P}_{Z^{(0)}}$. Therefore, if we have trained the algorithm with a certain initial distribution $\mathbb{P}_{Z^{(0)}}$, we cannot simply change it to another initial distribution $\mathbb{P}_{\hat{Z}^{(0)}}$ and expect that the corresponding guarantees still hold true, because this would correspond to making predictions about a distribution from which we have not observed a single sample.

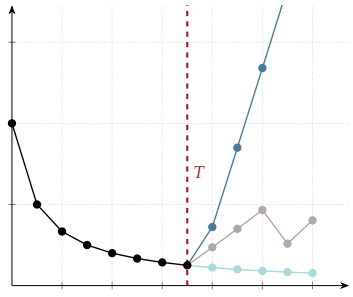

Figure 6.2: Illustration for changing the time-horizon: If we observe a sequence only up to time T, we do not know how the sequence behaves later on.

Changing the Time-Horizon

Extending the time-horizon after training to more iterations while keeping the guarantees is also not straightforward to do. As discussed above, during training we typically observe the time-points $J = \{0, \dots, T\}$ for some $T \in \mathbb{N}_0$. Hence, for a given $(h, p) \in \mathcal{H} \times \mathcal{P}$, the distribution of the observable part of the trajectory is given by:

$$\psi_{h,p} \circ \pi_{\{0,\dots,T\}}^{-1} = \mathbb{P}_{Z^{(0)}} \otimes \bigotimes_{k=0}^{T-1} \gamma(h, p, \cdot)\,. \tag{6.1}$$

Thus, this part of the trajectory (obviously) depends on the time-horizon T, that is, the maximal number of iterations. Therefore, if we have trained the algorithm with a certain upper bound on the number of iterations, we cannot simply run the algorithm for more iterations and expect that the corresponding guarantees still hold true also for these later iterations. Again, this would correspond to making predictions about a distribution from which we have not observed a single sample.

The Data is Expensive

Since we can only observe a finite number of iterations yet the goal is to solve optimization problems, most of the guarantees that we are interested in need a large amount of iterations. For example, if we are interested in the loss after a fixed amount of iterations, say T, we have to perform T iterations of the algorithm for each single data point. However, as Equation (6.1) shows, the distribution of the trajectory also depends on the hyperparameters h and the parameters p. Therefore, if we have N different objective functions to train on and K hyperparameters, we have to perform at least $T \cdot N \cdot K$ iterations to compute the guarantee, which scales up quite fast.[2] For example, if we have $T = 3000$, $N = 1000$, and $K = 100$, we have to perform 300.000.000 iterations. Hence, even if we can do 3000 iterations per second, this still amounts to a computation time of roughly 28 hours.

2: Even more so, if we have a stochastic algorithm, we might want to compute I trajectories per configuration, such that everything gets multiplied additionally by I.

Training is Difficult

While we have made significant progress, the overall process of training an optimization algorithm is nevertheless difficult. Especially, finding an architecture that works well for a given set of problems often is time-consuming. Although a certain structural similarity between the models has emerged by now, they still need to be adapted to the specific problems. Additionally, the features that serve as input to the model strongly influence the performance and are, at least in part, problem-dependent. One thing that makes this endeavor even more difficult is the fact that there are already the conventional algorithms which are "computationally lightweight", that is, they are fast in their computation and can perform many iterations per second. Thus, to win the race in actual computation time, the learned algorithm has to have a better ratio of progress per iteration compared to cost per iteration, which often limits the design of the architecture to small models.

6.3 Future Research

There is a plethora of potential directions for future research, such that we only briefly touch on a few general ideas. However, we would like to point out that these ideas are still more or less vague and have not been investigated any further, so they might turn out to be wishful thinking or dead ends.

Generalization to more objective functions. While this is one of the main underlying assumptions of the approach presented here, it might be of concern that the learned algorithm is only applicable to realizations of the random objective $\ell(\cdot, P)$. Thus, future research could address the problem of generalizing these results to more objective functions. Similarly, being able to use different initializations or longer time-horizons is also of great interest. In general, it might be fruitful to combine the approaches of conventional optimization theory and the ones from learning-to-optimize, such that we (hopefully) end up in a middle-ground between analytical and observational tractability.

Using different divergences. The results presented here all rely on the Kullback-Leibler divergence as measure of proximity between the prior and the posterior. However, there are many different divergences and distances that also might be of interest for learning-to-optimize. Especially, the quite flexible class of "integral probability metrics" might be worth investigating, as it allows to consider different classes of test-functions in a unified framework.

Investigating the learning procedure. Based on the empirical findings in this thesis, training the optimization algorithm in a way that combines self-supervision and ideas from reinforcement learning allows for a lot of flexibility and additionally seems to be quite effective in many cases. Most of it, however, is based on simple heuristics. Thus, the theoretical properties of this learning approach might be interesting. For example, are these choices actually good from a theoretical perspective? Or do they have to be adjusted? Similarly, are there conditions under which

the hyperparameters actually converge? If yes, does the limit inherit desirable properties?

Variational inference. Instead of constructing the prior and posterior as it is done in this manuscript, one could also try to use techniques from variational inference, which often allows for fast approximations of intractable distributions with known ones. This would have the advantage that the prior and posterior would stem from well-understood classes of distributions, which might allow for further analytic considerations. Additionally, there might be some benefits in terms of computation time. However, the probabilistic constraining procedure could potentially not be used in the way as it is done here. Similarly, the main computational cost of the presented procedure stems from computing the guarantees, which could (probably) not be circumvented in this way.

Other generalization guarantees. While it has proven quite successful, there is no obvious reason why we do have to stick to *PAC-Bayesian* generalization guarantees. Especially, constructing a whole distribution over hyperparameters when we actually only need a single one that works is counterintuitive and might in fact turn out to be unnecessary. Here, it might be interesting to adapt the underlying idea of using convex conjugates or other objects that are defined in terms of a supremum, for example, the dual norm of a continuous linear functional, to obtain a suitable uniform bound.

Continuous-time dynamics. Similar to how conventional optimization algorithms are often studied through the lens of their continuous-time dynamics, that is, in terms of a corresponding differential equation, one could try to adapt this approach also for learning-to-optimize. In this context, the rich theory of Feller processes or, more generally, contraction semi-groups with their generators, might be of help. Actually, interpreting optimization algorithms through this lens might be interesting in general. Similarly, the rich theory around the Koopman-operator allows for handling nonlinear dynamical systems through a globally linear representation, which allows for a new point of view and could also yield valuable insights.

Adaptive algorithms. As stressed above, the loss-function for training the algorithm that was proposed in this thesis, that is, the ratio of the objective values of consecutive iterates, allows to train an algorithm without needing any further ingredients. Thus, it also provides a means for adapting an algorithm on-the-fly to each given loss-function separately. This could have a few potential benefits:

(i) The algorithm can "overfit" to each function separately, which could alleviate the need for generalization.
(ii) A pre-training phase might not be necessary anymore or, at least, less important, as the algorithm gets adapted to each function on-the-fly.
(iii) We would have adaptive algorithms also for objective functions that were intractable before, because the adaptation is based on backpropagation for a general architecture instead of a pre-defined update.

(iv) For example, when training a neural network, this approach allows to train the model and the algorithm jointly, that is, the learnable optimization algorithm is simply placed on top of the neural network so that its parameters then form the new leaf nodes in the computational graph.

Combining the model with conventional optimization theory. Using the Markovian model in the conventional optimization theory might be interesting, because it allows for investigating functions through the lens of the trajectory of the algorithm: Instead of imposing a property for the objective function globally, it could be imposed only $\mathbb{P}_{Z|H,P}$-a.s., that is, only in the region of the trajectories of the algorithm.

6.4 Personal Comment

The journey changes you; it should change you. [...] You take something with you. Hopefully, you leave something good behind.

–Anthony Bourdain, No Reservations: Around the World on an Empty Stomach

This marks the end of this thesis, and if you have made it this far, I would like to thank you for your time and interest. I hope this work has been insightful and did provide a valuable new perspective on learning-to-optimize.

Part IV

Appendix

A

Implementation Details for Chapter 3

The code can be found at: https://github.com/MichiSucker/Learning-to-Optimize-with-PAC-Bayes

We use the following training procedure in all experiments: $N = N_{\text{prior}} + N_{\text{train}} + N_{\text{val}} + N_{\text{test}}$ denotes the total number of datapoints, and we use $N_{\text{prior}} = ... = N_{\text{test}} = 250$. (Sub)Gradients are defined by the output of backpropagation as it is implemented in PyTorch (Paszke et al., 2019), and we use $\sigma(p) := \alpha \ell(x^{(0)}, p)^{\beta}$, $\alpha, \beta > 0$, to define the sublevel set L_{σ}. In Algorithms 1 and 2, we use $a = 0.95$, $b = 1.0$, $q_l = 0.01$, $q_u = 0.99$, and $\varepsilon = 0.075$. Thus, the algorithm should reach L_{σ} in at least 95% of the cases, and for the estimation of the sublevel probability it should concentrate 98% of the mass within a distance of 0.075. Further, in Algorithm 2 we use *stochastic gradient Langevin dynamics* to draw 10^2 samples, where we decay the step-size starting from 10^{-6}. In Algorithm 3, we use Adam with an initial step-size of 10^{-3}, which gets reduced by a factor of 0.5 every 200 iterations until an accuracy of $\varepsilon = 10^{-2}$ is reached, or for at most $n_{\text{init}} = 10^3$ iterations. In Algorithm 4, we use Adam for a total of $n_{\max} = 2 \cdot 10^5$ iterations with an initial step-size of 10^{-4}, which gets reduced by a factor of 0.5 every $2 \cdot 10^4$ iterations. We use a trajectory length of $t_{\text{len}} = 1$, that is, only single points, and update the constraint only every $2 \cdot 10^4$ iterations (with a reset to previous iterates, if we have left the set $\hat{\mathsf{A}}$). In Algorithm 6, we use a finite Λ with $|\Lambda| = 75 \cdot 10^3$, and an accuracy (of the PAC-bound) of $\varepsilon = 0.05$.

Remark A.0.1 (i) We always use the output of the backpropagation algorithm instead of exact (sub-)gradients, that is, the learned algorithms *do not rely on smoothness*.

(ii) We use 100 samples only, as they are very costly: To evaluate the potentials φ_{prior} and $\varphi_{\lambda}(\cdot, s)$ on a single sample $h \in \mathcal{H}$, one has to compute all losses $\ell(h, p_i)$, $i = 1, ..., N_{\text{prior}} + N_{\text{train}}$, that is, (approximately) solving $N_{\text{prior}} + N_{\text{train}}$ optimization problems.

A.1 Details for the Experiment on Quadratic Functions

This section describes the missing details for the experiment on quadratic functions in Subsection 3.6.1.

Construction of the Parameters

To control the strong-convexity and smoothness of ℓ, we specify intervals $[m_-, m_+], [L_-, L_+] \subset (0, +\infty)$, and sample $m_1, ..., m_N \overset{iid}{\sim} \mathrm{U}_{[m_-, m_+]}$, $L_1, ..., L_N \overset{iid}{\sim} \mathrm{U}_{[L_-, L_+]}$. Then, the matrices A_j, $j = 1, ..., N$, are created as *diagonal matrices* with entries $a^j_{ii} = \sqrt{m_j} + i \cdot \frac{\sqrt{L_j} - \sqrt{m_j}}{n}$, $i = 1, ..., n$, that is, we

linearly interpolate from $\sqrt{m_j}$ to $\sqrt{L_j}$. Hence, the matrix $A_j^T A_j$ has smallest and largest eigenvalue m_j and L_j, respectively. To change the relative position between the ellipsoid of the quadratic and the initialization, we randomize the right-hand side by sampling $b_1, \ldots, b_N \overset{iid}{\sim} \mathcal{N}(\mu, \Sigma)$, where we create μ and $\Sigma = C^T C$ by sampling $\mu_i, C_{i,k} \overset{iid}{\sim} \mathrm{U}_{[-5,5]}$, $i, k = 1, \ldots, n$.

A.2 Details for the Image-Processing Experiment

This section describes the missing details for the image-processing experiment in Subsection 3.6.2.

A.2.1 Construction of the Parameters

Throughout, we use $\varepsilon = 0.01$. For computational efficiency, the matrices A, D_h, D_w are implemented through the convolution of the image x with a corresponding kernel (with reflective boundary conditions). For A, we use a standard 5×5-Gaussian kernel, while D_h and D_w are given through the kernels:

$$k_h = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \in \mathbb{R}^{3\times 3} \qquad \text{and} \qquad k_w = \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{pmatrix} \in \mathbb{R}^{3\times 3}.$$

Additionally, after blurring an image with A, we add centered Gaussian noise $\varepsilon_{i,j}$ with standard deviation $\sigma = \frac{25}{256}$ to each pixel, that is, $b_{i,j} = (Ax^*)_{i,j} + \varepsilon_{i,j}$ with $\varepsilon_{i,j} \overset{iid}{\sim} \mathcal{N}(0, \sigma)$, $i = 1, \ldots, N_h$, $j = 1, \ldots, N_w$. The regularization parameters $\lambda_i \in \mathbb{R}$, $i = 1, \ldots, N$, are given by sampling uniformly, that is, $\lambda_i \overset{iid}{\sim} \mathrm{U}_{[\lambda_-, \lambda_+]}$, where we use $\lambda_- = 0.05$ and $\lambda_+ = 0.5$.

A.3 Details for the LASSO Experiment

This section describes the missing details for the LASSO experiment in Subsection 3.6.3.

A.3.1 Construction of the Parameters

The same matrix $A \in \mathbb{R}^{m\times n}$ with dimensions $n = 350$ and $m = 70$ is used for all problem instances. Here, we sample each entry uniformly, that is, $a_{i,j} \overset{iid}{\sim} \mathrm{U}_{[-0.5,0.5]}$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Thus, the parameters p are given by the right-hand side and the regularization parameter, that is, $p = (b, \lambda) \in \mathbb{R}^{m+1} =: \mathcal{P}$. For this, the regularization parameter λ is also sampled uniformly, that is, $\lambda_i \overset{iid}{\sim} \mathrm{U}_{[\lambda_-, \lambda_+]}$, $i = 1, \ldots, N$, with $\lambda_- = 5$ and $\lambda_+ = 10$, while the right-hand side is sampled from a multivariate normal distribution, that is, $b_i \overset{iid}{\sim} \mathcal{N}(\mu, \Sigma)$, $i = 1, \ldots, N$, where we first construct μ and $\Sigma = C^T C$ by sampling each entry of μ and C uniformly at random in $[-5, 5]$.

A.3.2 Algorithm

The solutions of the LASSO problem are typically sparse. To achieve this, the algorithm has to identify the coordinates which are non-zero. Therefore, in each iteration, we treat the zero and non-zero entries of $x^{(t)}$ (and derived quantities) separately. Here, non-zero entries are written with a $\neq$-subscript, while zero entries are written with a 0-subscript, for example, $x^{(t)}_{\neq}$ and $x^{(t)}_{0}$.Then, first, we compute weights $w_1, \dots, w_8$ with a fully-connected block with ReLU-activation functions, where we use the features $n^{(t)}_1 = \log(1 + \|\nabla \ell(x^{(t)}, p)\|)$, $n^{(t)}_2 = \log(1 + \|x^{(t)} - x^{(t-1)}\|)$, $n^{(t)}_3 = \log(1 + \|p^{(t)}\|)$, where $p^{(t)} = \operatorname{prox}_{\alpha g}\left(x^{(t)} - \alpha \nabla \ell(x^{(t)}, p)\right)$, $\Delta \ell^{(t)} := \ell(x^{(t)}, p) - \ell(x^{(t-1)}, p)$, $\Delta g^{(t)} := g(x^{(t)}) - g(x^{(t-1)})$, $\Delta h^{(t)} := h(x^{(t)}, p) - h(x^{(t-1)}, p)$, the scalar products $s^{(t)}_{\neq}$ and $s^{(t)}_{0}$ between the (normalized) gradient and (normalized) momentum, and the regularization parameter λ. Then, we use these weights to perform a reweighting of the directions $d^{(t)}_1, \dots, d^{(t)}_4$, which are the normalized gradient, the normalized momentum, the normalized residual $x^{(t)} - p^{(t)}$, and the coordinate-wise product between (normalized) gradient and (normalized) momentum. Then, these reweighted directions get inserted into a 1x1-convolutional block, which predicts the two directions $d^{(t)}_1$ and $d^{(t)}_2$.

A.4 Details for the Neural-Network-Training Experiment

This section describes the missing details for the neural-network-training experiment in Subsection 3.6.4.

A.4.1 Construction of the Parameters

We assume that the neural network should learn a function $g : \mathbb{R} \to \mathbb{R}$ from noisy observations y_j, that is $y_j = g(x_j) + \varepsilon$ with $\varepsilon \sim \mathcal{N}(0, 1)$. For this, we construct polynomials g_i, $i = 1, \dots, N$, of degree $d = 5$ as follows: For every function g_i, we sample points $\{x_{i,j}\}_{j=1}^{K}$ (here: $K = 50$) uniformly in $[-2, 2]$, that is, $x_{i,j} \overset{iid}{\sim} \mathrm{U}_{[-2,2]}$, $i = 1, \dots, N$, $j = 1, \dots, K$. Then, we sample the coefficients $(c_{i,0}, \dots, c_{i,5})$ of g_i uniformly in $[-5, 5]$, that is, $c_{i,l} \overset{iid}{\sim} \mathrm{U}_{[-5,5]}$, $i = 1, \dots, N$, $l = 0, \dots, 5$. Lastly, we get the values $y_{i,j}$ as:

$$y_{i,j} = g_i(x_{i,j}) + \varepsilon_{i,j} \quad \text{with} \quad \varepsilon_{i,j} \overset{iid}{\sim} \mathcal{N}(0, 1), \quad i = 1, \dots, N, \; j = 1, \dots, 50\,.$$

For every function $g_i : \mathbb{R} \to \mathbb{R}$ the neural network is trained on the data set $p_i := \{X_i, Y_i\}$ with $X_i = (x_{i,1}, \dots, x_{i,K}) \in \mathbb{R}^K$ and $Y_i = (y_{i,1}, \dots, y_{i,K}) \in \mathbb{R}^K$. Hence, the data set will serve as the parameter p of the loss function $\ell : \mathbb{R}^p \times \mathcal{P} \to \mathbb{R}_{\geq 0}$, such that the parameter space $\mathcal{P}$ can be identified as the space of these data sets, that is, $\mathcal{P} = \mathbb{R}^{K \times 2}$.

A.4.2 Loss Function and Architecture

Since the mean square error is the standard choice for training models on regression tasks, the loss is given by $\ell(\theta, p_i) := c(\mathtt{N}(\theta, X_i), Y_i) := \frac{1}{K} \sum_{j=1}^{K} (\mathtt{N}_\theta(x_{i,j}) - y_{i,j})^2$. As model $\mathtt{N}_\theta$ we use a fully-connected two layer

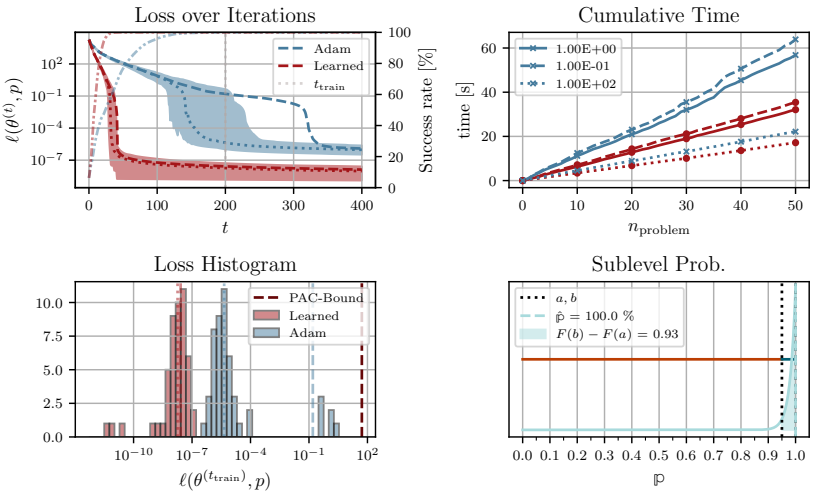

Figure A.1: Upper left: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions indicate the 10th to 90th percentile. Further, dashdotted-lines represent the classification-accuracy, which is shown on the right y-axis. Here, Adam is shown in blue and the learned algorithm in red. **Upper right:** Cumulative computation time to solve all the test problems up to a certain accuracy measured by $\ell(\theta^{(t)}, p) - c(X_i, Y_i) < \varepsilon$ (both algorithms are run for at most $t_{\text{max}} = 5000$ iterations). **Lower left:** Loss histogram (after t_{train} iterations) and PAC-bound. **Lower right:** Bayesian estimate for the sublevel probability (Beta-posterior; light blue line) and the imposed constraints a, b (black dotted line).

neural network with `ReLU`-activation functions. To have more features in the input layer, the input x is transformed into the vector $(x, x^2, ..., x^5)$. Hence, the parameters $\theta \in \mathbb{R}^m$ are given by the weights $A_1 \in \mathbb{R}^{50\times 5}$, $A_2 \in \mathbb{R}^{1\times 50}$ and biases $b_1 \in \mathbb{R}^{50}, b_2 \in \mathbb{R}$ of the two fully-connected layers. Therefore, the optimization space is of dimension $m = (5 \cdot 50) + (1 \cdot 50) + 50 + 1 = 351$.

A.5 Additional Experiment on MNIST

This experiment considers the problem of training a neural network to do classification on the MNIST data set, that is, $\mathcal{A}$ is trained to predict the parameters $\theta \in \mathbb{R}^m$ of a neural network $\mathtt{N}_\theta$, which then is used to predict a class-label $k \in \{0, ..., 9\}$ based on an input image. Hence, the optimization variable is given by $\theta \in \mathbb{R}^m$. Here, the model consists of two convolutional layers with ReLU-activation functions and Max-Pooling, followed by three linear layers with ReLU-activation functions. Through this, the optimization variable θ has dimension $m = 13090$.

Remark A.5.1 Note that, as the theory does not apply to stochastic algorithms, we have to compute full gradients. This limits the amount of images (20x20) per data set to 250.

As loss-function, we use a penalized cross-entropy loss to enforce higher classification-accuracy, as parameters p of the loss-function we use the data sets consisting of input images and ground-truth labels, that is, $p \in \mathbb{R}^{250\times(20\times 20)\times 1} = \mathcal{P}$, and as baseline we use Adam. The architecture of the learned algorithm is shown in Figure A.2 and consists of two blocks: The first block uses linear layers with ReLU-activation functions and computes seven weights based on the gradient-norm, the norm of the momentum-term, and the incurred losses. Then, these weights are used to scale the input-directions of the second block. This second block consists of 1×1-convolutional layers with ReLU-activation functions, and computes an update-direction based on the gradient, the momentum, their coordinate-wise product, and four additional directions, which are computed from the gradient and the momentum-term by coordinate-wise preconditioning. For more details we refer to the GitHub repository. Figure A.1 shows the results of this experiment: The upper left plot shows that the learned algorithm outperforms Adam, classifying all images

correctly after about 50 iterations, while Adam needs about 200 iterations to reach the same classification-accuracy. The upper right plot confirms that, also in terms of computation time, $\mathcal{A}$ is faster in training the neural network than Adam. However, based on the higher computational cost per iteration, the gap is not as large as for the function values. The lower left plot shows that the predicted PAC-bound is not tight here. This can be attributed to the fact that we had to use a smaller amount of data, due to the high computational cost in each iteration. Finally, the lower right plot indicates that the algorithm did reach the sublevel set in all test cases.

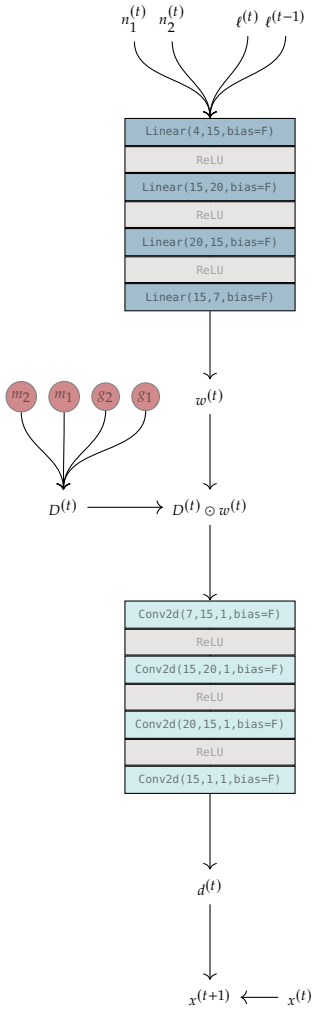

Figure A.2: Algorithmic update for the MNIST experiment: Based on the two norms $n_1^{(t)}$ and $n_2^{(t)}$, and the current and previous (logarithmically scaled) loss $\ell^{(t)}$, $\ell^{(t-1)}$, we compute a weighting-vector $w^{(t)}$, which is used to scale the different directions in $D^{(t)}$, which are given by $d_1^{(t)}$, $d_2^{(t)}$, their corresponding preconditioned versions $g_i \odot d_1^{(t)}$, $m_i \odot d_2^{(t)}$, and their coordinate-wise product $d_1^{(t)} \odot d_2^{(t)}$. Then, these get inserted (as separate channels) into the a 1x1-convolutional block, which computes the new update direction $d^{(t)}$. Finally, we update $x^{(t+1)} := x^{(t)} + d^{(t)}$.

B

Implementation Details for Chapter 4

This chapter provides a few more details for the experiments corresponding to Chapter 4, especially a detailed description of the used features.

The code can be found at: https://github.com/MichiSucker/Markovian-Model-for-L2O

B.1 Details for the Image-Processing Experiment

This section summarizes the missing details for the experiment in Subsection 4.5.2.

The update of the learned algorithm reads:

$$x^{(t+1)} := x^{(t)} + \frac{\|\nabla_1 \ell(x^{(t)}, p)\|}{L} d_1^{(t)} - \frac{1}{L}\nabla_1 \ell(x^{(t)}, p) + \|x^{(t)} - x^{(t-1)}\| \, d_2^{(t)} \, .$$

Here, the directions $d_1^{(t)}$ and $d_2^{(t)}$ are predicted by a 1x1-convolutional block with ReLU-activation functions based on the reweighted[1] direction-matrix $D^{(t)}$, which contains the normalized gradient, the normalized momentum, the normalized gradient of the data-fidelity term, and the normalized gradient of the regularization term. Here, the weights $w^{(t)}$ for the reweighting are predicted by a fully-connected block with ReLU-activation functions based on the following eleven features: $n_1^{(t)} = \log(1 + \|\nabla \ell(x^{(t)}, p)\|)$, $n_2^{(t)} = \log(1 + \|x^{(t)} - x^{(t-1)}\|)$, $\Delta \ell^{(t)} := \ell(x^{(t)}, p) - \ell(x^{(t-1)}, p)$, $\Delta r^{(t)} := r(x^{(t)}, p) - r(x^{(t-1)}, p)$, where r is the regularization term, $\Delta h^{(t)} := h(x^{(t)}, p) - h(x^{(t-1)}, p)$, where h is the data-fidelity term, $g^{(t)} := \max_{i=1,\dots,n} |\nabla_1 \ell(x^{(t)}, p)|_i$, the scalar products $s_1^{(t)}, \dots, s_4^{(t)}$ between the (normalized) gradient and the (normalized) momentum, between the (normalized) gradient of the regularization term and the (normalized) momentum, between the (normalized) gradient of the data-fidelity term and the (normalized) momentum, between the (normalized) gradient of the regularization term and the (normalized) gradient of the data-fidelity term, and, finally, the regularization parameter λ.

1: Each column is reweighted by one coordinate of $w^{(t)}$. Then, these directions get inserted into the convolutional block as separate channels.

B.2 Details for the LASSO Experiment

This section summarizes the missing details for the experiment in Subsection 4.5.3.

Since in the LASSO problem the algorithm has to identify the support of the solution, that is, the coordinates which are non-zero, we also treat the zero and non-zero entries of $x^{(t)}$ (and derived quantities) separately. Here, we denote the non-zero entries by $x_{\neq}^{(t)}$ and

2: Note that, while these entries are zero for $x^{(t)}$, this does not necessarily apply to derived quantities.

the zero entries by $x^{(t)}_{=}$, and similarly for all other quantities[2]. First, we compute a weight vector $w^{(t)}$ with a fully-connected block with ReLU-activation functions. For this, we use the features $n_1^{(t)} = \log(1 + \|\nabla_1 \ell(x^{(t)}, p)\|)$, $n_2^{(t)} = \log(1 + \|x^{(t)} - x^{(t-1)}\|)$, $n_3^{(t)} = \log(1 + \|p^{(t)}\|)$, where $p^{(t)} = \operatorname{prox}_{\alpha g}\left(x^{(t)} - \alpha \nabla_1 \ell(x^{(t)}, p)\right)$, $\Delta\ell^{(t)} := \ell(x^{(t)}, p) - \ell(x^{(t-1)}, p)$, $\Delta g^{(t)} := g(x^{(t)}) - g(x^{(t-1)})$, $\Delta h^{(t)} := h(x^{(t)}, p) - h(x^{(t-1)}, p)$, the scalar product $s^{(t)}$ between the (normalized) gradient and (normalized) momentum, and the regularization parameter λ. Then, these weights are used to perform a reweighting of the directions in the matrix $D^{(t)}$, which are given by the normalized gradient, the normalized momentum, the normalized residual $x^{(t)} - p^{(t)}$, and the coordinate-wise product between (normalized) gradient and (normalized) momentum. Afterwards, these reweighted directions get inserted into a 1x1-convolutional block, which predicts the two directions $d_1^{(t)}$ and $d_2^{(t)}$, which in turn are used to compute the final update with the proximal mapping, given by

$$x^{(t+1)} := \operatorname{prox}_{g/L}\left(x^{(t)} + \left(d_1^{(t)} - \nabla_1 \ell(x^{(t)}) + \|x^{(t)} - x^{(t-1)}\| \cdot d_2^{(t)}\right)/L\right).$$

B.3 Details for the Neural-Network-Training Experiment

This section summarizes the missing details for the experiment in Subsection 4.5.4.

To compute the weight vector $w^{(t)}$ with the first block, we use the following six features: $n_1^{(t)} = \log(1+\|\nabla_1 \ell(x^{(t)}, p)\|)$, $n_2^{(t)} = \log(1+\|x^{(t)} - x^{(t-1)}\|)$, $\Delta\ell^{(t)} := \ell(x^{(t)}, p) - \ell(x^{(t-1)}, p)$, $g^{(t)} := \max_{i=1,\dots,n} |\nabla_1 \ell(x^{(t)}, p)_i|$, the scalar product $s^{(t)}$ between the (normalized) gradient and the (normalized) momentum, and the time index t. Then, these weights are used to perform a weighting of the directions in $D^{(t)}$, which are given by the normalized gradient, the normalized momentum term, and their preconditioned version (diagonal preconditioners given by g and m). Then, these directions are inserted as different channels into a 1×1-convolutional block, which predicts the direction $d^{(t)}$, which in turn is used to update the iterate as $x^{(t+1)} = x^{(t)} + d^{(t)}$.

B.4 Details for the Experiment on Stochastic Empirical Risk Minimization

This section summarizes the missing details for the experiment in Subsection 4.5.5.

First, we compute the same features as Adam, that is,

$$\begin{aligned} m^{(t)} &= \beta_1 m^{(t-1)} + (1-\beta_1)\hat{\nabla}_1 \ell(x^{(t)}, p) \\ v^{(t)} &= \beta_2 v^{(t-1)} + (1-\beta_2)\hat{\nabla}_1 \ell(x^{(t)}, p) \odot \hat{\nabla}_1 \ell(x^{(t)}, p), \end{aligned}$$

where $\hat{\nabla}_1 \ell(x^{(t)}, p)$ denotes the stochastic gradient. Then, as for Adam, we set $\hat{m}^{(t)} = \frac{m^{(t)}}{1-\beta_1^t}$ and $\hat{v}^{(t)} = \frac{v^{(t)}}{1-\beta_2^t}$, and split $\hat{m}^{(t)}$ and $\hat{v}^{(t)}$ into the

corresponding (logarithmically transformed) norms $n_1^{(t)}$, $n_2^{(t)}$ and unit-vectors $d_1^{(t)}$, $d_2^{(t)}$, respectively. The norms, together with the current (stochastic) loss and the time-index t, get inserted as features $f^{(t)}$ into a fully-connected block to compute a weight vector $w^{(t)}$ and an additional scalar $s^{(t)}$. Here, $s^{(t)}$ is used as a step-size in the final update, while $w^{(t)}$ is used to scale the vectors $d_1^{(t)}$, $d_2^{(t)}$, and $d_1^{(t)} \odot d_2^{(t)}$ (columns of the matrix $D^{(t)}$) before they get inserted into the 1×1-convolutional block, which outputs vectors $p_1^{(t)}$ and $p_2^{(t)}$. These, in turn, are used as diagonal preconditioners for the update of Adam, that is, the final output is given by the formula

$$x^{(t)} = x^{(t-1)} - (s^{(t)} \cdot \alpha \cdot p_1^{(t)} \odot \hat{m}^{(t)})/(0.001 \cdot |p_2^{(t)}| \odot \sqrt{\hat{v}^{(t)}} + \varepsilon)\,.$$

Here, the constant 0.001 is inserted to stabilize the training in the beginning, and for the other constants we use the default values in PyTorch, that is, we set $\kappa = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1 \cdot 10^{-8}$.

C Implementation Details for Chapter 5

This chapter provides a few more details for the experiments corresponding to Chapter 5, especially a detailed description of the used features.

The code can be found at: https://github.com/MichiSucker/COLA_2024

C.1 Details for the Experiment on Quadratic Functions

This section summarizes the missing details for the experiment in Subsection 5.3.2.

The algorithmic update is similar to before and consists of two blocks:

1) The first block consists of 1×1-convolutional layers with ReLU-activation functions and computes the update direction $d^{(t)}$. As features, we use the normalized gradient $d_1^{(t)} := \frac{\nabla_1 \ell(x^{(t)},p)}{\|\nabla_1 \ell(x^{(t)},p)\|}$, the normalized momentum term $d_2^{(t)} := \frac{x^{(t)}-x^{(t-1)}}{\|x^{(t)}-x^{(t-1)}\|}$, and their coordinate-wise product $d_1^{(t)} \odot d_2^{(t)}$. The normalization is done to stabilize the training.
2) The second block consists of linear layers with ReLU-activation functions and computes the step-size $\alpha^{(t)}$. As features, we use the (logarithmically transformed) norm of the gradient $s_1^{(t)} = \log(1 + \|\nabla_1 \ell(x^{(t)}, p)\|)$ and of the momentum $s_2^{(t)} = \log\left(1 + \|x^{(t)} - x^{(t-1)}\|\right)$, and the current and previous losses $s_3^{(t)} = \log\left(1 + \ell(x^{(t)}, p)\right)$, $s_4^{(t)} = \log\left(1 + \ell(x^{(t-1)}, p)\right)$. Again, the logarithmic scaling is done to stabilize training. Here, the term "+1" is added to map zero onto zero.

Importantly, we want to stress that the algorithmic update is not constrained in any way: the algorithm just predicts a direction and a step-size, and we do not enforce any specific properties.

C.2 Details for the Neural-Network-Training Experiment

This section summarizes the missing details for the experiment in Subsection 5.3.3.

The algorithmic update is similar to before and consists of two blocks:

1) The first block consists of linear layers with ReLU-activation functions and computes a weight vector $w^{(t)}$. As features, we use the (logarithmically transformed) gradient norm $s_1^{(t)} := \log(1 + \|\nabla_1 \ell(x^{(t)}, p)\|)$, the (logarithmically transformed) norm of the momentum term $s_2^{(t)} := \log(1 + \|x^{(t)} - x^{(t-1)}\|)$, the difference between the current and previous loss $s_3^{(t)} := \ell(x^{(t)}, p) - \ell(x^{(t-1)}, p)$, the scalar product between the (normalized) gradient and the (normalized) momentum term $s_4^{(t)}$, the maximal absolute value of the coordinates of the gradient $s_5^{(t)}$, and the iteration counter t.
2) The second block consists of 1×1-convolutional layers with ReLU-activation functions and computes the update direction $d^{(t)}$. As features, we use the normalized gradient $d_1^{(t)} := \frac{\nabla_1 \ell(x^{(t)}, p)}{\|\nabla_1 \ell(x^{(t)}, p)\|}$, the normalized momentum term $d_2^{(t)} := \frac{x^{(t)} - x^{(t-1)}}{\|x^{(t)} - x^{(t-1)}\|}$, and their "preconditioned" versions $g \odot d_1^{(t)}$ and $m \odot d_2^{(t)}$, where the weights $m, d \in \mathbb{R}^d$ are learned, too.

Again, we want to stress that the algorithmic update is not constrained in any way: the algorithm just predicts a direction, and we do not restrict it in any way.

Bibliography

Ablin, P., T. Moreau, M. Massias, and A. Gramfort (2019). 'Learning step sizes for unfolded sparse coding'. In: *Advances in Neural Information Processing Systems*. Vol. 32.

Absil, P.-A., R. Mahony, and B. Andrews (2005). 'Convergence of the Iterates of Descent Methods for Analytic Cost Functions'. In: *SIAM Journal on Optimization* 16.2, pp. 531–547. DOI: 10.1137/040605266.

Adler, J. and O. Öktem (2018). 'Learned Primal-Dual Reconstruction'. In: *IEEE Transactions on Medical Imaging* 37.6, pp. 1322–1332. DOI: 10.1109/TMI.2018.2799231.

Ahmad, R., C. A. Bouman, G. T. Buzzard, S. Chan, S. Liu, E. T. Reehorst, and P. Schniter (2020). 'Plug-and-Play Methods for Magnetic Resonance Imaging: Using Denoisers for Image Recovery'. In: *IEEE Signal Processing Magazine* 37.1, pp. 105–116. DOI: 10.1109/MSP.2019.2949470.

Almeida, D., C. Winter, J. Tang, and W. Zaremba (2021). 'A Generalizable Approach to Learning Optimizers'. In: *arXiv preprint arXiv:2106.00958*.

Alquier, P. (2024). 'User-friendly Introduction to PAC-Bayes Bounds'. In: *Foundations and Trends® in Machine Learning* 17.2, pp. 174–303. DOI: 10.1561/2200000100.

Alquier, P. and B. Guedj (2018). 'Simpler PAC-Bayesian bounds for hostile data'. In: *Machine Learning* 107.5, pp. 887–902. DOI: 10.1007/s10994-017-5690-0.

Alquier, P., J. Ridgway, and N. Chopin (2016). 'On the properties of variational approximations of Gibbs posteriors'. In: *Journal of Machine Learning Research* 17.236, pp. 1–41.

Ambroladze, A., E. Parrado-Hernández, and J. Shawe-Taylor (2006). 'Tighter PAC-Bayes Bounds'. In: *Advances in Neural Information Processing Systems*. Vol. 19.

Amit, R., B. Epstein, S. Moran, and R. Meir (2022). 'Integral Probability Metrics PAC-Bayes Bounds'. In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 3123–3136.

Amit, R. and R. Meir (2018). 'Meta-learning by Adjusting Priors Based on Extended PAC-Bayes Theory'. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, pp. 205–214.

Amos, B. (2023). 'Tutorial on Amortized Optimization'. In: *Foundations and Trends® in Machine Learning* 16.5, pp. 592–732. DOI: 10.1561/2200000102.

Andrychowicz, M., M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas (2016). 'Learning to learn by gradient descent by gradient descent'. In: *Advances in Neural Information Processing Systems*. Vol. 29.

Attouch, H. and J. Bolte (2009). 'On the convergence of the proximal algorithm for nonsmooth functions involving analytic features'. In: *Mathematical Programming* 116, pp. 5–16. DOI: 10.1007/s10107-007-0133-5.

Attouch, H., J. Bolte, and B. F. Svaiter (2013). 'Convergence of descent methods for semi-algebraic and tame problems: proximal algorithms, forward–backward splitting, and regularized Gauss–Seidel methods'. In: *Mathematical Programming* 137, pp. 91–129. DOI: 10.1007/s10107-011-0484-9.

Attouch, H., J. Bolte, P. Redont, and A. Soubeyran (2010). 'Proximal Alternating Minimization and Projection Methods for Nonconvex Problems: An Approach Based on the Kurdyka-Łojasiewicz Inequality'. In: *Mathematics of Operations Research* 35.2, pp. 438–457. DOI: 10.1287/moor.1100.0449.

Audibert, J.-Y. and O. Bousquet (2007). 'Combining PAC-Bayesian and Generic Chaining Bounds'. In: *Journal of Machine Learning Research* 8.32, pp. 863–889.

Audibert, J.-Y. and O. Catoni (2011). 'Robust linear least squares regression'. In: *Annals of Statistics* 39.5, pp. 2766–2794. DOI: 10.1214/11-AOS918.

Banerjee, I., V. A. Rao, and H. Honnappa (2021). 'PAC-Bayes Bounds on Variational Tempered Posteriors for Markov Models'. In: *Entropy* 23.3. DOI: 10.3390/e23030313.

Banert, S., A. Ringh, J. Adler, J. Karlsson, and O. Öktem (2020). 'Data-Driven Nonsmooth Optimization'. In: *SIAM Journal on Optimization* 30.1, pp. 102–131. DOI: 10.1137/18M1207685.

Banert, S., J. Rudzusika, O. Öktem, and J. Adler (2024). 'Accelerated Forward-Backward Optimization Using Deep Learning'. In: *SIAM Journal on Optimization* 34.2, pp. 1236–1263. DOI: 10.1137/22M1532548.

Bardenet, R., A. Doucet, and C. Holmes (2014). 'Towards scaling up Markov chain Monte Carlo: an adaptive subsampling approach'. In: *Proceedings of the 31st International Conference on Machine Learning*. Vol. 32. Proceedings of Machine Learning Research, pp. 405–413.

– (2017). 'On Markov chain Monte Carlo methods for tall data'. In: *Journal of Machine Learning Research* 18.47, pp. 1–43.

Barndorff–Nielsen, O. (2014). *Information and Exponential Families in Statistical Theory*. John Wiley & Sons.

Beck, A. and M. Teboulle (2009). 'A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems'. In: *SIAM Journal on Imaging Sciences* 2.1, pp. 183–202. DOI: 10.1137/080716542.

Bégin, L., P. Germain, F. Laviolette, and J.-F. Roy (2014). 'PAC-Bayesian Theory for Transductive Learning'. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Vol. 33. Proceedings of Machine Learning Research. PMLR, pp. 105–113.

– (2016). 'PAC-Bayesian Bounds based on the Rényi Divergence'. In: *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*. Vol. 51. Proceedings of Machine Learning Research. PMLR, pp. 435–444.

Bello, I., B. Zoph, V. Vasudevan, and Q. V. Le (2017). 'Neural Optimizer Search with Reinforcement Learning'. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 459–468.

Bengio, Y., P. Simard, and P. Frasconi (1994). 'Learning long-term dependencies with gradient descent is difficult'. In: *IEEE Transactions on Neural Networks* 5.2, pp. 157–166. DOI: 10.1109/72.279181.

Berger, J. O. (1985). *Statistical Decision Theory and Bayesian Analysis*. Springer New York, NY.

Bertsekas, D. (1999). *Nonlinear Programming*. Athena Scientific.

Bianchi, P., W. Hachem, and S. Schechtman (2022). 'Convergence of Constant Step Stochastic Gradient Descent for Non-Smooth Non-Convex Functions'. In: *Set-Valued and Variational Analysis* 30.3, pp. 1117–1147. DOI: 10.1007/s11228-022-00638-z.

Bierstone, E. and P. D. Milman (1988). 'Semianalytic and subanalytic sets'. In: *Publications Mathématiques de l'Institut des Hautes Études Scientifiques* 67, pp. 5–42. DOI: 10.1007/BF02699126.

Biggs, F. and B. Guedj (2021). 'Differentiable PAC–Bayes Objectives with Partially Aggregated Neural Networks'. In: *Entropy* 23.10. DOI: 10.3390/e23101280.

Blåsjö, V. (2005). 'The Isoperimetric Problem'. In: *The American Mathematical Monthly* 112.6, pp. 526–566. DOI: 10.1080/00029890.2005.11920227.

Bolte, J., A. Daniilidis, and A. Lewis (2007a). 'The Łojasiewicz Inequality for Nonsmooth Subanalytic Functions with Applications to Subgradient Dynamical Systems'. In: *SIAM Journal on Optimization* 17.4, pp. 1205–1223. DOI: 10.1137/050644641.

Bolte, J., A. Daniilidis, A. Lewis, and M. Shiota (2007b). 'Clarke Subgradients of Stratifiable Functions'. In: *SIAM Journal on Optimization* 18.2, pp. 556–572. DOI: 10.1137/060670080.

Bolte, J., S. Sabach, and M. Teboulle (2014). 'Proximal alternating linearized minimization for nonconvex and nonsmooth problems'. In: *Mathematical Programming* 146.1, pp. 459–494. DOI: 10.1007/s10107-013-0701-9.

Boucheron, S., G. Lugosi, and P. Massart (2013). *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press.

Bousquet, O. and A. Elisseeff (2000). 'Algorithmic Stability and Generalization Performance'. In: *Advances in Neural Information Processing Systems*. Vol. 13.

– (2002). 'Stability and Generalization'. In: *Journal of Machine Learning Research* 2, pp. 499–526.

Boyd, S., N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. (2011). 'Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers'. In: *Foundations and Trends® in Machine learning* 3.1, pp. 1–122. DOI: 10.1561/2200000016.

Brifman, A., Y. Romano, and M. Elad (2016). 'Turning a denoiser into a super-resolver using plug and play priors'. In: *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 1404–1408. DOI: 10.1109/ICIP.2016.7532589.

Buzzard, G. T., S. H. Chan, S. Sreehari, and C. A. Bouman (2018). 'Plug-and-play Unplugged: Optimization-Free Reconstruction Using Consensus Equilibrium'. In: *SIAM Journal on Imaging Sciences* 11.3, pp. 2001–2020. DOI: 10.1137/17M1122451.

Cao, Y., T. Chen, Z. Wang, and Y. Shen (2019). 'Learning to Optimize in Swarms'. In: *Advances in Neural Information Processing Systems*. Vol. 32.

Castera, C. and P. Ochs (2024). 'From Learning to Optimize to Learning Optimization Algorithms'. In: *arXiv preprint arXiv:2405.18222*.

Catoni, O. (2003). 'A PAC-Bayesian approach to adaptive classification'. In: *preprint*.

– (2004). *Statistical Learning Theory and Stochastic Optimization. Ecole d'Eté de Probabilités de Saint-Flour, XXXI-2001*. Springer Berlin, Heidelberg.

– (2007). 'PAC-Bayesian Supervised Classification: The Thermodynamics of Statistical Learning'. In: *Lecture Notes-Monograph Series* 56. DOI: 10.1214/074921707000000391.

Chan, S. H., X. Wang, and O. A. Elgendy (2017). 'Plug-and-Play ADMM for Image Restoration: Fixed-Point Convergence and Applications'. In: *IEEE Transactions on Computational Imaging* 3.1, pp. 84–98. DOI: 10.1109/TCI.2016.2629286.

Chang, J. R., C.-L. Li, B. Póczos, B. Vijaya Kumar, and A. C. Sankaranarayanan (2017). 'One Network to Solve Them All — Solving Linear Inverse Problems Using Deep Projection Models'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 5889–5898. DOI: 10.1109/ICCV.2017.627.

Chen, T., X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin (2022). 'Learning to Optimize: A Primer and A Benchmark'. In: *Journal of Machine Learning Research* 23.189, pp. 1–59.

Chen, T., W. Zhang, Z. Jingyang, S. Chang, S. Liu, L. Amini, and Z. Wang (2020). 'Training Stronger Baselines for Learning to Optimize'. In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 7332–7343.

Chen, X., C. Liang, D. Huang, E. Real, K. Wang, H. Pham, X. Dong, T. Luong, C.-J. Hsieh, Y. Lu, and Q. V. Le (2023). 'Symbolic Discovery of Optimization Algorithms'. In: *Advances in Neural Information Processing Systems*. Vol. 36, pp. 49205–49233.

Chen, X., J. Liu, Z. Wang, and W. Yin (2018). 'Theoretical Linear Convergence of Unfolded ISTA and Its Practical Weights and Thresholds'. In: *Advances in Neural Information Processing Systems*. Vol. 31.

Chen, X., Y. Zhang, C. Reisinger, and L. Song (2020). 'Understanding Deep Architecture with Reasoning Layer'. In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 1240–1252.

Chen, Y., M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. de Freitas (2017). 'Learning to Learn without Gradient Descent by Gradient Descent'. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 748–756.

Chérief-Abdellatif, B.-E., Y. Shi, A. Doucet, and B. Guedj (2022). 'On PAC-Bayesian reconstruction guarantees for VAEs'. In: *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*. Vol. 151. Proceedings of Machine Learning Research. PMLR, pp. 3066–3079.

Clerico, E., G. Deligiannidis, and A. Doucet (2023). 'Wide stochastic networks: Gaussian limit and PAC-Bayesian training'. In: *Proceedings of The 34th International Conference on Algorithmic Learning Theory*. Vol. 201. Proceedings of Machine Learning Research. PMLR, pp. 447–470.

Cohen, R., M. Elad, and P. Milanfar (2021). 'Regularization by Denoising via Fixed-Point Projection (RED-PRO)'. In: *SIAM Journal on Imaging Sciences* 14.3, pp. 1374–1406. DOI: 10.1137/20M1337168.

Corbineau, M.-C., C. Bertocchi, E. Chouzenoux, M. Prato, and J.-C. Pesquet (2019). 'Learned Image Deblurring by Unfolding a Proximal Interior Point Algorithm'. In: *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 4664–4668. DOI: 10.1109/ICIP.2019.8803438.

Dale, A. I. and P.-S. Laplace (2012). *Pierre-Simon Laplace Philosophical Essay on Probabilities. Translated from the fifth French edition of 1825 With Notes by the Translator*. 1st ed. Springer New York, NY.

Daniel, C., J. Taylor, and S. Nowozin (2016). 'Learning Step Size Controllers for Robust Neural Network Training'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 30.1. DOI: 10.1609/aaai.v30i1.10187.

Daubechies, I., M. Defrise, and C. De Mol (2004). 'An iterative thresholding algorithm for linear inverse problems with a sparsity constraint'. In: *Communications on Pure and Applied Mathematics* 57.11, pp. 1413–1457. DOI: https://doi.org/10.1002/cpa.20042.

Domke, J. (2012). 'Generic Methods for Optimization-Based Modeling'. In: *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*. Vol. 22. Proceedings of Machine Learning Research. PMLR, pp. 318–326.

Donsker, M. D. and S. S. Varadhan (1975). 'Asymptotic evaluation of certain Markov process expectations for large time, I'. In: *Communications on Pure and Applied Mathematics* 28.1, pp. 1–47.

Dries, L. P. D. v. d. (1998). *Tame Topology and O-minimal Structures*. Cambridge University Press.

Dziugaite, G. K., K. Hsu, W. Gharbieh, G. Arpino, and D. Roy (2021). 'On the Role of Data in PAC-Bayes Bounds'. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 604–612.

Dziugaite, G. K. and D. M. Roy (2017). 'Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data'. In: *Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence*. AUAI Press.

– (2018). 'Data-dependent PAC-Bayes priors via differential privacy'. In: *Advances in Neural Information Processing Systems*. Vol. 31.

Efron, B. (1975). 'Defining the Curvature of a Statistical Problem (with Applications to Second Order Efficiency)'. In: *The Annals of Statistics* 3.6, pp. 1189–1242. DOI: 10.1214/aos/1176343282.

Foong, A., W. Bruinsma, D. Burt, and R. Turner (2021). 'How Tight Can PAC-Bayes be in the Small Data Regime?' In: *Advances in Neural Information Processing Systems*. Vol. 34, pp. 4093–4105.

Gärtner, E., L. Metz, M. Andriluka, C. D. Freeman, and C. Sminchisescu (2023). 'Transformer-Based Learned Optimization'. In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11970–11979. DOI: 10.1109/CVPR52729.2023.01152.

Gavaskar, R. G. and K. N. Chaudhury (2020). 'Plug-and-Play ISTA Converges With Kernel Denoisers'. In: *IEEE Signal Processing Letters* 27, pp. 610–614. DOI: 10.1109/LSP.2020.2986643.

Germain, P., F. Bach, A. Lacoste, and S. Lacoste-Julien (2016). 'PAC-Bayesian Theory Meets Bayesian Inference'. In: *Advances in Neural Information Processing Systems*. Vol. 29.

Germain, P., A. Lacasse, F. Laviolette, and M. Marchand (2009). 'PAC-Bayesian Learning of Linear Classifiers'. In: *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 353–360.

Giryes, R., Y. C. Eldar, A. M. Bronstein, and G. Sapiro (2018). 'Tradeoffs Between Convergence Speed and Reconstruction Accuracy in Inverse Problems'. In: *IEEE Transactions on Signal Processing* 66.7, pp. 1676–1690. DOI: 10.1109/TSP.2018.2791945.

Goldstine, H. H. (1980). *A History of the Calculus of Variations from the 17th through the 19th Century*. Springer New York, NY.

Gregor, K. and Y. LeCun (2010). 'Learning Fast Approximations of Sparse Coding'. In: *Proceedings of the 27th International Conference on Machine Learning*, pp. 399–406.

Guedj, B. (2019). 'A Primer on PAC-Bayesian Learning'. In: *Proceedings of the Second Congress of the French Mathematical Society*. Vol. 33.

Gupta, H., K. H. Jin, H. Q. Nguyen, M. T. McCann, and M. Unser (2018). 'CNN-Based Projected Gradient Descent for Consistent CT Image Reconstruction'. In: *IEEE Transactions on Medical Imaging* 37.6, pp. 1440–1453. DOI: 10.1109/TMI.2018.2832656.

Haddouche, M. and B. Guedj (2022). 'Online PAC-Bayes Learning'. In: *Advances in Neural Information Processing Systems*. Vol. 35, pp. 25725–25738.

– (2023). 'Wasserstein PAC-Bayes Learning: Exploiting Optimisation Guarantees to Explain Generalisation'. In: *arXiv preprint arXiv:2304.07048*.

Haddouche, M., B. Guedj, O. Rivasplata, and J. Shawe-Taylor (2021). 'PAC-Bayes Unleashed: Generalisation Bounds with Unbounded Losses'. In: *Entropy* 23.10. DOI: 10.3390/e23101330.

Hastie, T., R. Tibshirani, and J. Friedman (2009). *The Elements of Statistical Learning. Data Mining, Inference, and Prediction*. 2nd ed. Springer New York, NY.

He, J., Y. Yang, Y. Wang, D. Zeng, Z. Bian, H. Zhang, J. Sun, Z. Xu, and J. Ma (2019). 'Optimizing a Parameterized Plug-and-Play ADMM for Iterative Low-Dose CT Reconstruction'. In: *IEEE Transactions on Medical Imaging* 38.2, pp. 371–382. DOI: 10.1109/TMI.2018.2865202.

Heaton, H., X. Chen, Z. Wang, and W. Yin (2023). 'Safeguarded Learned Convex Optimization'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 37.6, pp. 7848–7855. DOI: 10.1609/aaai.v37i6.25950.

Hellström, F., G. Durisi, B. Guedj, and M. Raginsky (2025). 'Generalization Bounds: Perspectives from Information Theory and PAC-Bayes'. In: *Foundations and Trends® in Machine Learning* 18.1, pp. 1–223. DOI: 10.1561/2200000112.

Higgs, M. and J. Shawe-Taylor (2010). 'A PAC-Bayes Bound for Tailored Density Estimation'. In: *Algorithmic Learning Theory*. Springer Berlin Heidelberg, pp. 148–162. DOI: https://doi.org/10.1007/978-3-642-16108-7_15.

Hillier, F. S. and G. J. Lieberman (2015). *Introduction to Operations Research*. McGraw-Hill.

Hochreiter, S., A. S. Younger, and P. R. Conwell (2001). 'Learning to Learn Using Gradient Descent'. In: *Artificial Neural Networks — ICANN 2001*. Springer Berlin Heidelberg, pp. 87–94. DOI: https://doi.org/10.1007/3-540-44668-0_13.

Holland, M. (2019). 'PAC-Bayes under potentially heavy tails'. In: *Advances in Neural Information Processing Systems*. Vol. 32.

Honorio, J. and T. Jaakkola (2014). 'Tight Bounds for the Expected Risk of Linear Classifiers and PAC-Bayes Finite-Sample Guarantees'. In: *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*. Vol. 33. Proceedings of Machine Learning Research. PMLR, pp. 384–392.

Hospedales, T., A. Antoniou, P. Micaelli, and A. Storkey (2022). 'Meta-Learning in Neural Networks: A Survey'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9, pp. 5149–5169. DOI: 10.1109/TPAMI.2021.3079209.

Huisman, M., J. N. Van Rijn, and A. Plaat (2021). 'A survey of deep meta-learning'. In: *Artificial Intelligence Review* 54.6, pp. 4483–4541. DOI: https://doi.org/10.1007/s10462-021-10004-4.

Hutter, F., L. Kotthoff, and J. Vanschoren (2019). *Automated Machine Learning. Methods, Systems, Challenges*. Springer Cham.

Kallenberg, O. (2021). *Foundations of Modern Probability*. Springer Cham.

Kamilov, U. S., H. Mansour, and B. Wohlberg (2017). 'A Plug-and-Play Priors Approach for Solving Nonlinear Imaging Inverse Problems'. In: *IEEE Signal Processing Letters* 24.12, pp. 1872–1876. DOI: 10.1109/LSP.2017.2763583.

Kingma, D. P. and J. Ba (2015). 'Adam: A Method for Stochastic Optimization'. In: *3rd International Conference on Learning Representations, ICLR 2015*.

Klenke, A. (2013). *Wahrscheinlichkeitstheorie*. Springer Spektrum Berlin, Heidelberg.

Kobler, E., A. Effland, K. Kunisch, and T. Pock (2022). 'Total Deep Variation: A Stable Regularization Method for Inverse Problems'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.12, pp. 9163–9180. DOI: 10.1109/TPAMI.2021.3124086.

Kobler, E., T. Klatzer, K. Hammernik, and T. Pock (2017). 'Variational Networks: Connecting Variational Methods and Deep Learning'. In: *Pattern Recognition*. Springer International Publishing, pp. 281–293.

Königsberger, K. (2006). *Analysis 2*. Springer Berlin, Heidelberg.

Korattikara, A., Y. Chen, and M. Welling (2014). 'Austerity in MCMC Land: Cutting the Metropolis-Hastings Budget'. In: *Proceedings of the 31st International Conference on Machine Learning*. Vol. 32. Proceedings of Machine Learning Research 1. PMLR, pp. 181–189.

Kurdyka, K. (1998). 'On gradients of functions definable in o-minimal structures'. In: *Annales de l'institut Fourier*. Vol. 48. 3, pp. 769–783.

Lacasse, A., F. Laviolette, M. Marchand, P. Germain, and N. Usunier (2006). 'PAC-Bayes Bounds for the Risk of the Majority Vote and the Variance of the Gibbs Classifier'. In: *Advances in Neural Information Processing Systems*. Vol. 19.

Langford, J. and R. Caruana (2001). '(Not) Bounding the True Error'. In: *Advances in Neural Information Processing Systems*. Vol. 14.

Langford, J. and J. Shawe-Taylor (2002). 'PAC-Bayes & Margins'. In: *Advances in Neural Information Processing Systems*. Vol. 15.

Laviolette, F., M. Marchand, and J.-F. Roy (2011). 'From PAC-Bayes Bounds to Quadratic Programs for Majority Votes'. In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*, pp. 649–656.

Lever, G., F. Laviolette, and J. Shawe-Taylor (2013). 'Tighter PAC-Bayes bounds through distribution-dependent priors'. In: *Theoretical Computer Science* 473, pp. 4–28. DOI: https://doi.org/10.1016/j.tcs.2012.10.013.

Li, K. and J. Malik (2017a). 'Learning to Optimize'. In: *International Conference on Learning Representations*.

– (2017b). 'Learning to Optimize Neural Nets'. In: *arXiv preprint arXiv:1703.00441*.

Liao, I., R. Dangovski, J. N. Foerster, and M. Soljacic (2023). 'Learning to Optimize Quasi-Newton Methods'. In: *Transactions on Machine Learning Research*.

Liu, D., K. Sun, Z. Wang, R. Liu, and Z.-J. Zha (2020). 'Frank-Wolfe Network: An Interpretable Deep Structure for Non-Sparse Coding'. In: *IEEE Transactions on Circuits and Systems for Video Technology* 30.9, pp. 3068–3080. DOI: 10.1109/TCSVT.2019.2936135.

Liu, J., X. Chen, Z. Wang, and W. Yin (2019). 'ALISTA: Analytic Weights Are As Good As Learned Weights in LISTA'. In: *International Conference on Learning Representations (ICLR)*.

Liu, J., X. Chen, Z. Wang, W. Yin, and H. Cai (2023). 'Towards Constituting Mathematical Structures for Learning to Optimize'. In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 21426–21449.

Liu, R., S. Cheng, Y. He, X. Fan, Z. Lin, and Z. Luo (2020). 'On the Convergence of Learning-Based Iterative Methods for Nonconvex Inverse Problems'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.12, pp. 3027–3039. DOI: 10.1109/TPAMI.2019.2920591.

London, B. (2017). 'A PAC-Bayesian Analysis of Randomized Learning with Application to Stochastic Gradient Descent'. In: *Advances in Neural Information Processing Systems*. Vol. 30.

Lorenzen, S. S., C. Igel, and Y. Seldin (2019). 'On PAC-Bayesian bounds for random forests'. In: *Machine Learning* 108.8, pp. 1503–1522. DOI: 10.1007/s10994-019-05803-4.

Lv, K., S. Jiang, and J. Li (2017). 'Learning Gradient Descent: Better Generalization and Longer Horizons'. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 2247–2255.

Maclaurin, D. and R. P. Adams (2014). 'Firefly Monte Carlo: Exact MCMC with subsets of data'. In: *30th Conference on Uncertainty in Artificial Intelligence, UAI 2014*. AUAI Press, pp. 543–552.

Masegosa, A., S. Lorenzen, C. Igel, and Y. Seldin (2020). 'Second Order PAC-Bayesian Bounds for the Weighted Majority Vote'. In: vol. 33, pp. 5263–5273.

Massart, P. (2007). *Concentration Inequalities and Model Selection. Ecole d'Eté de Probabilités de Saint-Flour XXXIII-2003*. Springer Berlin, Heidelberg.

Mbacke, S. D., F. Clerc, and P. Germain (2023). 'PAC-Bayesian Generalization Bounds for Adversarial Generative Models'. In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 24271–24290.

McAllester, D. A. (1998). 'Some PAC-Bayesian Theorems'. In: *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, pp. 230–234. DOI: 10.1145/279943.279989.

– (1999). 'PAC-Bayesian Model Averaging'. In: *Proceedings of the Twelfth Annual Conference on Computational Learning Theory*, pp. 164–170. DOI: 10.1145/307400.307435.

– (2003a). 'PAC-Bayesian Stochastic Model Selection'. In: *Machine Learning* 51.1, pp. 5–21. DOI: 10.1023/A:1021840411064.

– (2003b). 'Simplified PAC-Bayesian Margin Bounds'. In: *Learning Theory and Kernel Machines*. Springer Berlin, Heidelberg, pp. 203–215. DOI: https://doi.org/10.1007/978-3-540-45167-9_16.

Meinhardt, T., M. Möller, C. Hazirbas, and D. Cremers (2017). 'Learning Proximal Operators: Using Denoising Networks for Regularizing Inverse Imaging Problems'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1799–1808. DOI: 10.1109/ICCV.2017.198.

Metz, L., C. D. Freeman, J. Harrison, N. Maheswaranathan, and J. Sohl-Dickstein (2022). 'Practical Tradeoffs between Memory, Compute, and Performance in Learned Optimizers'. In: *Proceedings of The 1st Conference on Lifelong Learning Agents*. Vol. 199. Proceedings of Machine Learning Research. PMLR, pp. 142–164.

Metz, L., N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein (2019). 'Understanding and correcting pathologies in the training of learned optimizers'. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 4556–4565.

Möller, M., T. Möllenhoff, and D. Cremers (2019). 'Controlling Neural Networks via Energy Dissipation'. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3255–3264. DOI: 10.1109/ICCV.2019.00335.

Monga, V., Y. Li, and Y. C. Eldar (2021). 'Algorithm Unrolling: Interpretable, Efficient Deep Learning for Signal and Image Processing'. In: *IEEE Signal Processing Magazine* 38.2, pp. 18–44. DOI: 10.1109/MSP.2020.3016905.

Moreau, T. and J. Bruna (2017). 'Understanding Trainable Sparse Coding with Matrix Factorization'. In: *International Conference on Learning Representations*.

Nesterov, Y. (2018). *Lectures on Convex Optimization*. Springer Cham.

Nesterov, Y. (1983). 'A method for solving the convex programming problem with convergence rate $O(1/k^2)$'. In: *Proceedings of the USSR Academy of Sciences* 269, pp. 543–547.

Nocedal, J. and S. J. Wright (2006). *Numerical Optimization*. Springer New York, NY.

Nozawa, K., P. Germain, and B. Guedj (2020). 'PAC-Bayesian Contrastive Unsupervised Representation Learning'. In: *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*. Vol. 124. Proceedings of Machine Learning Research. PMLR, pp. 21–30.

Ochs, P. (2019). 'Unifying Abstract Inexact Convergence Theorems and Block Coordinate Variable Metric iPiano'. In: *SIAM Journal on Optimization* 29.1, pp. 541–570. DOI: 10.1137/17M1124085.

Ochs, P., Y. Chen, T. Brox, and T. Pock (2014). 'iPiano: Inertial Proximal Algorithm for Nonconvex Optimization'. In: *SIAM Journal on Imaging Sciences* 7.2, pp. 1388–1419. DOI: 10.1137/130942954.

Ohnishi, Y. and J. Honorio (2021). 'Novel Change of Measure Inequalities with Applications to PAC-Bayesian Bounds and Monte Carlo Estimation'. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*. Vol. 130. Proceedings of Machine Learning Research. PMLR, pp. 1711–1719.

Ono, S. (2017). 'Primal-Dual Plug-and-Play Image Restoration'. In: *IEEE Signal Processing Letters* 24.8, pp. 1108–1112. DOI: 10.1109/LSP.2017.2710233.

Orvieto, A., S. L. Smith, A. Gu, A. Fernando, C. Gulcehre, R. Pascanu, and S. De (2023). 'Resurrecting Recurrent Neural Networks for Long Sequences'. In: *Proceedings of the 40th International Conference on Machine Learning*. Vol. 202. Proceedings of Machine Learning Research. PMLR, pp. 26670–26698.

Parrado-Hernández, E., A. Ambroladze, J. Shawe-Taylor, and S. Sun (2012). 'PAC-Bayes Bounds with Data Dependent Priors'. In: *Journal of Machine Learning Research* 13.112, pp. 3507–3531.

Pascanu, R., T. Mikolov, and Y. Bengio (2013). 'On the difficulty of training recurrent neural networks'. In: *Proceedings of the 30th International Conference on Machine Learning*. Vol. 28. Proceedings of Machine Learning Research 3. PMLR, pp. 1310–1318.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala (2019). 'PyTorch: An Imperative Style, High-Performance Deep Learning Library'. In: *Advances in Neural Information Processing Systems*. Vol. 32.

Pedregosa, F. and D. Scieur (2020). 'Average-Case Acceleration Through Spectral Density Estimation'. In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 7553–7562.

Pentina, A. and C. Lampert (2014). 'A PAC-Bayesian Bound for Lifelong Learning'. In: *Proceedings of the 31st International Conference on Machine Learning*. Vol. 32. Proceedings of Machine Learning Research 2. PMLR, pp. 991–999.

Pérez-Ortiz, M., O. Rivasplata, J. Shawe-Taylor, and C. Szepesvári (2021). 'Tighter Risk Certificates for Neural Networks'. In: *Journal of Machine Learning Research* 22.227, pp. 1–40.

Polyak, B. T. (1964). 'Some methods of speeding up the convergence of iteration methods'. In: *USSR Computational Mathematics and Mathematical Physics* 4.5, pp. 1–17. DOI: https://doi.org/10.1016/0041-5553(64)90137-5.

– (1987). *Introduction to Optimization*. New York, Optimization Software.

Prémont-Schwarz, I., J. Vítků, and J. Feyereisl (2022). 'A Simple Guard for Learned Optimizers'. In: *Proceedings of the 39th International Conference on Machine Learning*. Vol. 162. Proceedings of Machine Learning Research. PMLR, pp. 17910–17925.

Quiroz, M., R. Kohn, M. Villani, and M.-N. Tran (2019). 'Speeding Up MCMC by Efficient Data Subsampling'. In: *Journal of the American Statistical Association* 114.526, pp. 831–843. DOI: 10.1080/01621459.2018.1448827.

Richter, S. (2019). *Statistisches und maschinelles Lernen. Gängige Verfahren im Überblick*. Springer Spektrum Berlin, Heidelberg.

Rivasplata, O., I. Kuzborskij, C. Szepesvári, and J. Shawe-Taylor (2020). 'PAC-Bayes Analysis Beyond the Usual Bounds'. In: *Advances in Neural Information Processing Systems*. Vol. 33, pp. 16833–16845.

Robert, C. and G. Casella (2004). *Monte Carlo Statistical Methods*. Springer New York, NY.

Rockafellar, R. T. and R. J.-B. Wets (1998). *Variational Analysis*. Springer Berlin, Heidelberg.

Rothfuss, J., V. Fortuin, M. Josifoski, and A. Krause (2021). 'PACOH: Bayes-Optimal Meta-Learning with PAC-Guarantees'. In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 9116–9126.

Ryu, E., J. Liu, S. Wang, X. Chen, Z. Wang, and W. Yin (2019). 'Plug-and-Play Methods Provably Converge with Properly Trained Denoisers'. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 5546–5557.

Scieur, D. and F. Pedregosa (2020). 'Universal Average-Case Optimality of Polyak Momentum'. In: *Proceedings of the 37th International Conference on Machine Learning*. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 8565–8572.

Seeger, M. (2002). 'PAC-Bayesian Generalisation Error Bounds for Gaussian Process Classification'. In: *Journal of Machine Learning Research* 3, pp. 233–269.

Seldin, Y., N. Cesa-Bianchi, P. Auer, F. Laviolette, and J. Shawe-Taylor (2012a). 'PAC-Bayes-Bernstein Inequality for Martingales and its Application to Multiarmed Bandits'. In: *Proceedings of the Workshop on On-line Trading of Exploration and Exploitation 2*. Vol. 26. Proceedings of Machine Learning Research. PMLR, pp. 98–111.

Seldin, Y., F. Laviolette, N. Cesa-Bianchi, J. Shawe-Taylor, and P. Auer (2012b). 'PAC-Bayesian Inequalities for Martingales'. In: *IEEE Transactions on Information Theory* 58.12, pp. 7086–7093. DOI: 10.1109/TIT.2012.2211334.

Sevilla, J., L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos (2022). 'Compute Trends Across Three Eras of Machine Learning'. In: *2022 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8. DOI: 10.1109/IJCNN55064.2022.9891914.

Shalev-Shwartz, S., O. Shamir, N. Srebro, and K. Sridharan (2010). 'Learnability, Stability and Uniform Convergence'. In: *Journal of Machine Learning Research* 11.90, pp. 2635–2670.

Shawe-Taylor, J. and R. C. Williamson (1997). 'A PAC Analysis of a Bayesian Estimator'. In: *Proceedings of the Tenth Annual Conference on Computational Learning Theory*, pp. 2–9. DOI: 10.1145/267460.267466.

Sprechmann, P., A. M. Bronstein, and G. Sapiro (2015). 'Learning Efficient Sparse and Low Rank Models'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9, pp. 1821–1833. DOI: 10.1109/TPAMI.2015.2392779.

Sprechmann, P., R. Litman, T. Ben Yakar, A. M. Bronstein, and G. Sapiro (2013). 'Supervised Sparse Analysis and Synthesis Operators'. In: *Advances in Neural Information Processing Systems*. Vol. 26.

Sreehari, S., S. V. Venkatakrishnan, B. Wohlberg, G. T. Buzzard, L. F. Drummy, J. P. Simmons, and C. A. Bouman (2016). 'Plug-and-Play Priors for Bright Field Electron Tomography and Sparse Interpolation'. In: *IEEE Transactions on Computational Imaging* 2.4, pp. 408–423. DOI: 10.1109/TCI.2016.2599778.

Stigler, S. M. (1981). 'Gauss and the Invention of Least Squares'. In: *The Annals of Statistics* 9.3, pp. 465–474. DOI: 10.1214/aos/1176345451.

Sucker, M., J. Fadili, and P. Ochs (2024a). 'Learning-to-Optimize with PAC-Bayesian Guarantees: Theoretical Considerations and Practical Implementation'. In: *arXiv preprint arXiv:2404.03290*.

Sucker, M. and P. Ochs (2023). 'PAC-Bayesian Learning of Optimization Algorithms'. In: *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*. Vol. 206. Proceedings of Machine Learning Research. PMLR, pp. 8145–8164.

– (2024b). 'A Generalization Result for Convergence in Learning-to-Optimize'. In: *arXiv preprint arXiv:2410.07704*.

– (2024c). 'A Markovian model for learning-to-optimize'. In: *arXiv preprint arXiv:2408.11629*.

Sulam, J., A. Aberdam, A. Beck, and M. Elad (2020). 'On Multi-Layer Basis Pursuit, Efficient Algorithms and Convolutional Neural Networks'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.8, pp. 1968–1980. DOI: 10.1109/TPAMI.2019.2904255.

Sun, Y., B. Wohlberg, and U. S. Kamilov (2019). 'An Online Plug-and-Play Algorithm for Regularized Image Reconstruction'. In: *IEEE Transactions on Computational Imaging* 5.3, pp. 395–408. DOI: 10.1109/TCI.2019.2893568.

Sun, Y., Z. Wu, X. Xu, B. Wohlberg, and U. S. Kamilov (2021). 'Scalable Plug-and-Play ADMM With Convergence Guarantees'. In: *IEEE Transactions on Computational Imaging* 7, pp. 849–863. DOI: 10.1109/TCI.2021.3094062.

Taschner, R. (2020). 'Das Problem der Dido'. In: *77-mal Mathematik für zwischendurch. Unterhaltsame Kuriositäten und unorthodoxe Anwendungen*. Ed. by G. Glaeser. Springer Berlin, Heidelberg, pp. 47–48.

Teets, D. and K. Whitehead (1999). 'The Discovery of Ceres: How Gauss Became Famous'. In: *Mathematics Magazine* 72.2, pp. 83–93.

Teodoro, A. M., J. M. Bioucas-Dias, and M. A. T. Figueiredo (2017). 'Scene-Adapted plug-and-play algorithm with convergence guarantees'. In: *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, pp. 1–6. DOI: 10.1109/MLSP.2017.8168194.

Terris, M., A. Repetti, J.-C. Pesquet, and Y. Wiaux (2021). 'Enhanced Convergent PNP Algorithms For Image Restoration'. In: *2021 IEEE International Conference on Image Processing (ICIP)*. IEEE, pp. 1684–1688. DOI: 10.1109/ICIP42928.2021.9506226.

Themelis, A. and P. Patrinos (2019). 'SuperMann: A Superlinearly Convergent Algorithm for Finding Fixed Points of Nonexpansive Operators'. In: *IEEE Transactions on Automatic Control* 64.12, pp. 4875–4890. DOI: 10.1109/TAC.2019.2906393.

Thiemann, N., C. Igel, O. Wintenberger, and Y. Seldin (2017). 'A Strongly Quasiconvex PAC-Bayesian Bound'. In: *Proceedings of the 28th International Conference on Algorithmic Learning Theory*. Vol. 76. Proceedings of Machine Learning Research. PMLR, pp. 466–492.

Tibshirani, R. (1996). 'Regression Shrinkage and Selection via the Lasso'. In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1, pp. 267–288.

Tirer, T. and R. Giryes (2019). 'Image Restoration by Iterative Denoising and Backward Projections'. In: *IEEE Transactions on Image Processing* 28.3, pp. 1220–1234. DOI: 10.1109/TIP.2018.2875569.

Venkatakrishnan, S. V., C. A. Bouman, and B. Wohlberg (2013). 'Plug-and-Play priors for model based reconstruction'. In: *2013 IEEE Global Conference on Signal and Information Processing*. IEEE, pp. 945–948. DOI: 10.1109/GlobalSIP.2013.6737048.

Vettoruzzo, A., M.-R. Bouguelia, J. Vanschoren, T. Rögnvaldsson, and K. Santosh (2024). 'Advances and Challenges in Meta-Learning: A Technical Review'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.7, pp. 4763–4779. DOI: 10.1109/TPAMI.2024.3357847.

Vilalta, R. and Y. Drissi (2002). 'A Perspective View and Survey of Meta-Learning'. In: *Artificial Intelligence Review* 18.2, pp. 77–95. DOI: 10.1023/A:1019956318069.

Villalobos, P., J. Sevilla, T. Besiroglu, L. Heim, A. Ho, and M. Hobbhahn (2022). 'Machine Learning Model Sizes and the Parameter Gap'. In: *arXiv preprint arXiv:2207.02852*.

Villani, C. (2009). *Optimal Transport. Old and New*. Springer Berlin, Heidelberg.

Wadayama, T. and S. Takabe (2019). 'Deep Learning-Aided Trainable Projected Gradient Decoding for LDPC Codes'. In: *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, pp. 2444–2448. DOI: 10.1109/ISIT.2019.8849215.

Wang, Z., Q. Ling, and T. S. Huang (2016). 'Learning Deep $\ell 0$ Encoders'. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. DOI: 10.1609/aaai.v30i1.10198.

Welling, M. and Y. W. Teh (2011). 'Bayesian Learning via Stochastic Gradient Langevin Dynamics'. In: *Proceedings of the 28th International Conference on Machine Learning*, pp. 681–688.

Wichrowska, O., N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein (2017). 'Learned Optimizers that Scale and Generalize'. In: *Proceedings of the 34th International Conference on Machine Learning*. Vol. 70. Proceedings of Machine Learning Research. PMLR, pp. 3751–3760.

Witting, H. (1985). *Mathematische Statistik I. Parametrische Verfahren bei festem Stichprobenumfang*. Vieweg+Teubner Verlag Wiesbaden.

Wu, S., A. Dimakis, S. Sanghavi, F. Yu, D. Holtmann-Rice, D. Storcheus, A. Rostamizadeh, and S. Kumar (2019). 'Learning a Compressed Sensing Measurement Matrix via Gradient Unrolling'. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6828–6839.

Xie, X., J. Wu, G. Liu, Z. Zhong, and Z. Lin (2019). 'Differentiable Linearized ADMM'. In: *Proceedings of the 36th International Conference on Machine Learning*. Vol. 97. Proceedings of Machine Learning Research. PMLR, pp. 6902–6911.

Xin, B., Y. Wang, W. Gao, D. Wipf, and B. Wang (2016). 'Maximal Sparsity with Deep Networks?' In: *Advances in Neural Information Processing Systems*. Vol. 29.

Xu, C., T. Qin, G. Wang, and T.-Y. Liu (2017). 'Reinforcement Learning for Learning Rate Control'. In: *arXiv preprint arXiv:1705.11159*.

Xu, Z., A. M. Dai, J. Kemp, and L. Metz (2019). 'Learning an Adaptive Learning Rate Schedule'. In: *arXiv preprint arXiv:1909.09712*.

Yang, Y., J. Sun, H. Li, and Z. Xu (2016). 'Deep ADMM-Net for Compressive Sensing MRI'. In: *Advances in Neural Information Processing Systems*. Vol. 29.

Yao, Q., M. Wang, Y. Chen, W. Dai, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu (2018). 'Taking Human out of Learning Applications: A Survey on Automated Machine Learning'. In: *arXiv preprint arXiv:1810.13306*.

Zhang, J., B. O'Donoghue, and S. Boyd (2020). 'Globally Convergent Type-I Anderson Acceleration for Nonsmooth Fixed-Point Iterations'. In: *SIAM Journal on Optimization* 30.4, pp. 3170–3197. DOI: 10.1137/18M1232772.

Zheng, W., T. Chen, T.-K. Hu, and Z. Wang (2022). 'Symbolic Learning to Optimize: Towards Interpretability and Scalability'. In: *International Conference on Learning Representations*.