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

Michael Sucker MICHAEL.SUCKER@MATH.UNI-TUEBINGEN.DE
Department of Mathematics
University of Tübingen
Tübingen, Germany

Jalal Fadili JALAL.FADILI@ENSICAEN.FR
ENSICAEN
Normandie Université
CNRS, GREYC, France

Peter Ochs OCHS@CS.UNI-SAARLAND.DE
Department of Mathematics and Computer Science
Saarland University
Saarbrücken, Germany

Abstract

We use the PAC-Bayesian theory for the setting of learning-to-optimize. To the best of our knowledge, we present the first framework to learn optimization algorithms with provable generalization guarantees (PAC-Bayesian bounds) and explicit trade-off between convergence guarantees and convergence speed, which contrasts with the typical worst-case analysis. Our learned optimization algorithms provably outperform related ones derived from a worst-case analysis. The results rely on PAC-Bayesian bounds for general, possibly unbounded loss-functions based on exponential families. Further, we provide a concrete algorithmic realization of the framework and new methodologies for learning-to-optimize. Finally, we conduct four practically relevant experiments to support our theory. With this, we showcase that the provided learning framework yields optimization algorithms that provably outperform the state-of-the-art by orders of magnitude.

Keywords: learning-to-optimize, pac-bayes, exponential families, conditioning on convergence, probabilistically constrained sampling

1 Introduction

Typically, optimization algorithms are derived by performing a worst-case analysis on a specific class of problems. Doing so one obtains theoretical convergence guarantees for any instance inside the class. However, the abstract class of problems contains an enormous number beyond the concrete problem of interest, often including also pathological functions that do not occur in practical applications. Furthermore, since the derivation is done "on pen and paper", all modeling steps have to be analytically tractable. This limits the design of algorithms. Both of these restrictions can impair the performance of the resulting method on a concrete problem instance. Nevertheless, both restrictions can be alleviated through learning: Given a concrete application and performance-measure, the algorithm is trained on examples (data) to improve its performance. This enables the automatic adaptation of sophisticated algorithms to this particular setting. However, there is no free lunch: If the

algorithm is explicitly based on quantities that are not analytically tractable, one cannot expect to obtain the same theoretical guarantees as for the worst-case analysis discussed above. Since the practical usefulness of an optimization algorithm without convergence guarantees is at least questionable, this is a major problem and poses the first central question:

What kind of theoretical guarantees can be given for a learned optimization algorithm? Are we able to ensure its usefulness?

One possible alternative to the common guarantees is of statistical nature: Even if we do not know exactly what the algorithm does, we can still observe its performance during training. However, this begs the question whether its performance on the training data is actually representative for the performance on unseen data. Therefore, in the first part of this paper we provide a theoretical framework for learning an optimization algorithm based on its performance on a training set, that is, based on the empirical risk together with a generalization bound for the (true) risk. A popular framework that provides such generalization bounds is the PAC-Bayesian approach to learning, which we apply to the setting of learning-to-optimize. This yields the following informal version of our main theoretical result (compare Example 28). It states that, with high probability we will observe a (training) data set S for which the given bound on the risk of the algorithm's output will hold uniformly in a so-called index $\gamma \in \Gamma$ and distribution $\mathbb{Q}$:

Theorem 1 (Informal) *Under mild boundedness assumptions on the optimization algorithm, the $\mathbb{Q}$-average population loss $\mathcal{R}_\sigma$ of the algorithm's output can be bounded by the $\mathbb{Q}$-average empirical loss $\hat{\mathcal{R}}_\sigma$ of the algorithm's output plus some remainder term r_N that vanishes with the size N of the data set, that is, for all $\varepsilon > 0$:*

$$\mathbb{P}_S \left\{ \forall \gamma \in \Gamma,\ \forall \mathbb{Q} \in \mathcal{M}_1 : \mathbb{Q}[\mathcal{R}_\sigma] \leq \mathbb{Q}[\hat{\mathcal{R}}_\sigma] + r_N(\gamma) \right\} \geq 1 - \varepsilon \,.$$

Especially, the uniformity in $\mathbb{Q}$ allows for *learning* such a distribution. This provides one possible answer to the question about theoretical guarantees for learning-to-optimize. However, while being a generalization bound, such a guarantee is a statement about *relative* and not absolute values, that is, how the true risk *compares to* the empirical risk. Thus, one still has to train the optimization algorithm in such a way that the empirical risk is indeed small enough to be worth the effort. This is particularly important in the area of learning-to-optimize, as there are already algorithms that can provably solve the given problems in a rather short amount of time. Hence, the second central question that arises is of a more practical nature and pertains to the actual training of such an algorithm:

How do we learn an optimization algorithm, such that its performance is clearly superior to the one achieved by a worst-case analysis?

Therefore, in the second part of this work, we develop a concrete algorithmic realization, which allows for learning an optimization algorithm and evaluating the corresponding theoretical guarantee. This involves several key design choices that have not been used before and which are of interest in their own right. Furthermore, as empirical evaluation

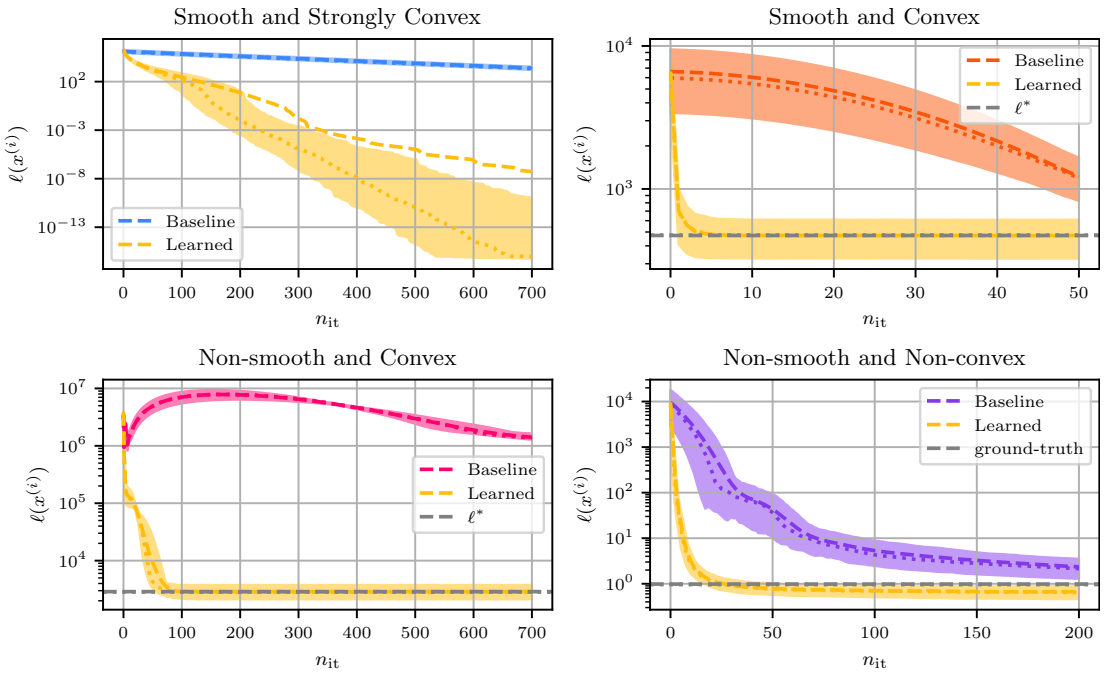

Figure 1: Some numerical results: Loss over iterations (mean as dashed and median as dotted line) of the learned algorithm compared to a standard choice.

of our claims, we conduct four practically relevant experiments, all dealing with very different classes of functions, thereby demonstrating the wide applicability and strong practical performance of our approach. Figure 1 provides a preview of some experimental results, and the details will be provided in Section 7. Each subplot compares the performance of the learned algorithm (yellow) to that of a standard algorithm on different problems ranging from smooth and strongly convex to non-smooth and non-convex. Since the learned algorithm is clearly superior in each case, this provides a possible answer to the question about how to train optimization algorithms. In summary, we provide a complete framework to train optimization algorithms with theoretical guarantees that are (in a certain sense) provably faster than their worst-case optimal counterparts. In particular, this work is a far reaching extension of our conference paper (Sucker and Ochs, 2023) by extending and clarifying the theoretical results in Sections 3, and, in particular, by the algorithmic realization together with its evaluation in Sections 6 and 7, which additionally includes a probabilistic constraining procedure for sampling algorithms in Subsection 5.1. The entire code associated with this paper can be found at `https://github.com/MichiSucker/Learning-to-Optimize-with-PAC-Bayes`.

1.1 Related Work

The literature on both learning-to-optimize and the PAC-Bayes learning approach is vast. Hence, for learning-to-optimize we will mainly focus on approaches that provide some theoretical guarantees. Especially, this excludes many model-free approaches, which replace the

whole update step with a learnable mapping such as a neural network. Chen et al. (2021) provide a good overview about the variety of approaches in learning-to-optimize, and good introductory references for the PAC-Bayesian approach are given by Guedj (2019), Hellström et al. (2023), and Alquier (2024).

1.1.1 Broader Context of Learning-to-Optimize

Optimization is an integral part of machine learning. Thus, learning-to-optimize has significant overlap with the areas of meta-learning (or "learning-to-learn") and AutoML. The first one is a subset of learning-to-optimize, because it is mostly concerned with determining parameters of machine learning models (Vilalta and Drissi, 2002; Hospedales et al., 2021). AutoML, however, more broadly refers to automating all steps necessary to create a machine learning application, which also includes the choice of an optimization algorithm and its hyperparameters (Yao et al., 2018; Hutter et al., 2019; He et al., 2021).

1.1.2 Learning-to-Optimize with Guarantees

Learned optimization methods may lack theoretical guarantees for the sake of convergence speed (Chen et al., 2021). Yet, there are applications where a convergence guarantee is of highest priority: Moeller et al. (2019) provide an example where a purely learning-based approach fails to reconstruct the crucial details in a medical image. Also, they prove convergence of their method by restricting the output to descent directions, for which mathematical guarantees exist. The basic idea is to trace the learned object back to, or constrain it to, a mathematical object with convergence guarantees. Similarly, Sreehari et al. (2016) provide sufficient conditions under which the learned mapping is a proximal mapping. Related schemes, under different assumptions and guarantees, are given by Chan et al. (2016), Teodoro et al. (2017), Tirer and Giryes (2018), Buzzard et al. (2018), Ryu et al. (2019), Sun et al. (2019), Terris et al. (2021) and Cohen et al. (2021). A major advantage of these methods is the fact that the number of iterations is not restricted a priori. However, a major drawback is their restriction to specific algorithms and problems. This contrasts with the approach of unrolling, pioneered by Gregor and LeCun (2010), which limits the number of iterations, yet in principle can be applied to every iterative optimization algorithm. Here, Xin et al. (2016) study the convergence properties of the IHT algorithm, while Chen et al. (2018) consider the unrolled ISTA. However, a difficulty in the theoretical analysis of unrolled algorithms is actually the notion of convergence itself, such that one rather has to consider the generalization performance. Only few works have addressed this: Either directly by means of Rademacher complexity (Chen et al., 2020b), or indirectly in form of a stability analysis (Kobler et al., 2020), as algorithmic stability is linked to generalization (Bousquet and Elisseeff, 2000, 2002; Shalev-Shwartz et al., 2010). *Our theoretical analysis corresponds to the approach of unrolling, that is, a fixed number of iterations. However, in the experiments we stay more closely to the iterative approach of learning an update step that can be applied for an arbitrary number of iterations.*

1.1.3 Design-Choices in Learning-to-Optimize

A major problem of many learned optimization algorithms, especially the ones based on recurrent neural networks (RNN), is their restriction to a certain number of iterations:

They cannot be trained for an arbitrary number of iterations due to instabilities or memory bottlenecks. Further, often they do not generalize well to more iterations than they were trained for (Andrychowicz et al., 2016; Chen et al., 2017; Lv et al., 2017; Chen et al., 2021). A typical way to mitigate this problem is to split the whole trajectory into smaller parts (Andrychowicz et al., 2016; Chen et al., 2017; Metz et al., 2019). However, often this does not lead to fully satisfactory results either, such that other approaches have been proposed: To improve generalization, Lv et al. (2017) introduce random scaling of the coordinates and the addition of a convex function to the objective. 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. Metz et al. (2019) replace the recurrent neural network with a multilayer perceptron (MLP), and they use two unbiased gradient estimators instead of one. Doing so they manage to train algorithms that are faster in wall-clock time than standard ones like Adam. Chen et al. (2020a) 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.
Besides the optimizer, a crucial design choice in learning-to-optimize is that of the loss function. Typically, either the final loss or a weighted sum of the losses along the iterations is used (Chen et al., 2021). *We introduce a new loss function for training optimization algorithms, motivated by an intuitive theoretical argument. Further, we use a single learned update based on MLPs instead of an RNN, and we split the trajectory into subtrajectories and randomize its total length, however, in a new way.*

1.1.4 PAC-Bayesian Bounds through Change-of-Measure

PAC is an acronym for *Probably Approximately Correct*, and PAC-*Bayes* refers to the fact that one considers distributions instead of points (Alquier, 2024). This framework allows for giving high probability bounds on the risk, either as an oracle or as an empirical bound. The key ingredient is a change-of-measure inequality, the choice of which strongly influences the corresponding bound. The one used most often is based on a variational representation of the Kullback–Leibler divergence due to Donsker and Varadhan (1975), employed, for example, by Catoni (2004, 2007). Yet, not all bounds are based on a variational representation, that is, holding uniformly over all posterior distributions (Rivasplata et al., 2020). While many bounds involve the Kullback–Leibler divergence as measure of proximity (McAllester, 2003a,b; Seeger, 2002; Langford and Shawe-Taylor, 2002; Germain et al., 2009), other divergences have been used: Honorio and Jaakkola (2014) prove an inequality for the χ^2-divergence, which is also used by London (2017). Bégin et al. (2016) and Alquier and Guedj (2018) use the Renyi-divergence (α-divergence). Ohnishi and Honorio (2021) propose PAC-bounds based on f-divergences, which include the Kullback–Leibler-, α- and χ^2-divergences. 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 et al., 2009). Motivated by this, Haddouche and Guedj (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. *We give a general PAC-Bayesian theorem based on exponential families. Here, prior, posterior, divergence and data dependence are given naturally. Further, it allows for implementing an abstract learning framework that can be applied to a wide variety of algorithms.*

1.1.5 Boundedness of the Loss Function

A major drawback of many of the existing PAC-Bayes bounds is the assumption of a bounded loss-function. 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: Germain et al. (2009) explicitly include the exponential-moment in the bound, Alquier et al. (2016) use so-called Hoeffding- and Bernstein-assumptions, Catoni (2004) restricts to the sub-Gaussian or sub-Gamma case. Another possibility to ensure the generalization or exponential-moment bounds is to use properties of the algorithm: London (2017) uses algorithmic stability to provide PAC-Bayes bounds for SGD. *We consider suitable properties of optimization algorithms aside from algorithmic stability to ensure the exponential-moment bounds.*

1.1.6 Minimization of the PAC-Bound

PAC-bounds relate the true risk to other terms such as the empirical risk. Yet, they do not directly say anything about the absolute numbers. Thus, learning procedures based on the PAC-Bayesian theory typically aim to minimize this bound: Langford and Caruana (2001) compute non-vacuous generalization bounds through a combination of PAC-bounds with a sensitivity analysis. Dziugaite and Roy (2017) extend this by minimizing the PAC-bound directly. Pérez-Ortiz et al. (2021) also consider learning as minimization of the PAC-Bayes bound and provide tight generalization bounds. Thiemann et al. (2017) are able to solve the minimization problem resulting from their PAC-bound by alternating minimization. *We follow this approach and consider learning as minimization of the PAC-Bayesian bound.*

1.1.7 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 leads to the idea of choosing a data- or distribution-dependent prior (Seeger, 2002; Parrado-Hernández et al., 2012; Lever et al., 2013; Dziugaite and Roy, 2018; Pérez-Ortiz et al., 2021), which, by using an independent subset of the data set, gets optimized to yield a good performance. *The prior distribution strongly influences the performance of our learned algorithms. Thus, we use a data-dependent prior. Further, we show how the prior can be used for preserving*

essential properties during learning: It is key to control the trade-off between convergence guarantee and convergence speed.

1.1.8 More Generalization Bounds

There are more areas of machine learning research that study generalization bounds. Importantly, the field of "stochastic optimization" (SO) provides generalization bounds for specific algorithms. The main differences to our setting are the learning approach and the assumptions made:

- Instead of a distribution over hpyerparameters, the algorithms in SO generate a point estimate, and one studies the properties of this point in terms of the stationarity measure of the true risk functional (Bottou et al., 2018; Davis and Drusvyatskiy, 2022; Bianchi et al., 2022).
- Instead of an abstract algorithm, the setting in SO is more explicit. Thus, more assumptions have to be made. Typical assumptions are (weak) convexity (Shalev-Shwartz et al., 2009; Davis and Drusvyatskiy, 2019), bounded gradients (Défossez et al., 2022), bounded noise (Davis and Drusvyatskiy, 2022), or smoothness (Kavis et al., 2022).

We provide a principled way to learn a distribution over general hyperparameters of an abstract algorithm under weak assumptions and go explicitly beyond analytically tractable quantities. Therefore, the methodology is independent of the chosen implementation.

The rest of the paper is structured as follows: In Section 2 we introduce the notation and provide a formal description of the setting. In Section 3, we derive the general PAC-Bayesian theorem and relate it to other existing bounds. In Section 4, we identify properties of optimization algorithms that allow to apply the PAC-Bayesian theorem. As this strongly relies on assumptions on the prior distribution, we provide a probabilistic constraining procedure that allows to enforce such constraints in Section 5. Then, in Section 6 we describe the learning procedure and our design choices for learning-to-optimize, and in Section 7 we conduct the experiments.

2 Problem Setup & Assumptions

In this section we establish the notation, formalize the setting, and state the main assumptions that are used throughout the remainder of the text.

2.1 Notation

We will endow every topological space $\mathcal{U}$ with the corresponding Borel-σ-algebra $\mathfrak{B}(\mathcal{U})$, and, given a product space $\mathcal{U} \times \mathcal{V}$ of two measurable spaces $(\mathcal{U}, \mathfrak{U})$ and $(\mathcal{V}, \mathfrak{V})$, we endow it with the product-σ-algebra $\mathfrak{U} \otimes \mathfrak{V}$. We will denote the product space of a generic number of spaces $\mathcal{U}_1, ..., \mathcal{U}_n$ by $\prod_{i=1}^n \mathcal{U}_i$, and the product-$\sigma$-algebra by $\bigotimes_{i=1}^n \mathfrak{B}(\mathcal{U}_i)$. If all spaces are equal, this is abbreviated as $\mathcal{U}^n$. For a function $f : \mathcal{U} \times \mathcal{V} \to \mathcal{W}$, $f(u, \cdot) : \mathcal{V} \to \mathcal{W}$ denotes the map $v \mapsto f(u, v)$ with fixed element $u \in \mathcal{U}$. Similarly, for a set $\mathsf{C} \subset \mathcal{U} \times \mathcal{V}$, the section of C for fixed $u \in \mathcal{U}$ is denoted by $\mathsf{C}_u := \{v \in \mathcal{V} \ : \ (u, v) \in \mathsf{C}\}$. In general, generic sets are denoted in typewriter font, for example A, and $\mathbb{1}_{\mathsf{A}}$ denotes the function that is equal to one for $u \in \mathsf{A}$

and zero else, while ι_{A} denotes the function that is equal to zero for $u \in \mathsf{A}$ and $+\infty$ else.[1] Given a measurable space $(\mathcal{U}, \mathfrak{U})$, a measure μ and a measurable function $f \geq 0$, $\mu[f]$ denotes the integral of f w.r.t. μ, while $f \cdot \mu$ denotes the measure given by $(f \cdot \mu)[\mathsf{A}] = \int_{\mathsf{A}} f(u)\ \mu(du)$, that is, $(f \cdot \mu)[\mathcal{U}] = \mu[f]$ and $(f \cdot \mu)[\mathsf{A}] = \mu[f \cdot \mathbb{1}_{\mathsf{A}}]$. Hence, $f \cdot \mu$ is absolutely continuous w.r.t. μ, written as $f \cdot \mu \ll \mu$, with f being the corresponding density. Here, the set of all measures on U will be denoted by $\mathcal{M}(\mathcal{U}) := \{\mu : \mathfrak{U} \to [0, \infty] \ : \ \mu \text{ is a measure}\}$, and the set of all probability measures that are absolutely continuous w.r.t. $\mu \in \mathcal{M}(\mathcal{U})$ are denoted by $\mathcal{M}_1(\mu) := \{\nu \in \mathcal{M}(\mathcal{U}) \ : \ \nu[\mathcal{U}] = 1 \text{ and } \nu \ll \mu\}$. In this context, the Kullback-Leibler divergence between two measures ν and μ is defined as

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

For the rest of the manuscript, we will fix a probability space $\big(\Omega, \mathfrak{F}, \mathbb{P}\big)$, and if $\mu = \mathbb{P}$ is the probability measure, the corresponding expectation is denoted by $\mathbb{E}[f] := \mathbb{P}[f] = \int_{\Omega} f(\omega)\ \mathbb{P}(d\omega)$. Here, we will write random variables in upper-case and corresponding realizations in lower-case with the same symbol, for example $U = u$. Given two random variables $U : \big(\Omega, \mathfrak{F}, \mathbb{P}\big) \to \mathcal{U}$ and $V : \big(\Omega, \mathfrak{F}, \mathbb{P}\big) \to \mathcal{V}$, integration of a measurable function f on $\mathcal{U} \times \mathcal{V}$ w.r.t. the induced probability measure $\mathbb{P}_{(U,V)}$ is specified by the subscript (U, V), that is:

$$\mathbb{E}\left[f(U,V)\right] = \int_{\Omega} f\left(U,V\right)(\omega)\ \mathbb{P}(d\omega) = \int_{\mathcal{U}\times\mathcal{V}} f(u,v)\ \mathbb{P}_{(U,V)}(du,dv) = \mathbb{E}_{(U,V)}\left[f\right]\,.$$

If we have a regular version of the conditional distribution of V, given U, denoted by $\mathbb{P}_{V|U}$, the joint distribution $\mathbb{P}_{(U,V)}$ can be disintegrated into the product $\mathbb{P}_U \otimes \mathbb{P}_{V|U}$ of the marginal $\mathbb{P}_U$ and the probability kernel $(x, \mathsf{B}) \mapsto \mathbb{P}_{V|U=u}[\mathsf{B}]$, which allows us to use the notation:

$$\mathbb{E}[f(U,V)] = \int_{\mathcal{U}} \int_{\mathcal{V}} f(u,v)\ \mathbb{P}_{V|U=u}(dv)\ \mathbb{P}_U(du) = \mathbb{E}_U\left[\mathbb{E}_{V|U=u}\left[f(u,\cdot)\right]\right]\,.$$

Note that changing the order of integration is not allowed in this case. However, if U and V are independent, their joint distribution is given by the product $\mathbb{P}_U \otimes \mathbb{P}_V$ for which Fubini's theorem is applicable, and the iterated integration is clarified by the subscripts U, V:

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

Finally, our theoretical results rely on the notions of probability kernels and exponential families, whose definitions are recalled in Appendix A.

2.2 Main Assumptions and Definitions for Learning Optimization Algorithms

We assume that we are given a distribution over loss-functions with a specific structure, which is modelled by a random variable:

1. We omit the name here, as both $\mathbb{1}_{\mathsf{A}}$ and ι_{A} are called "indicator function". The former in probability theory, the latter in optimization.

Assumption 2 *We are given a Polish space $\mathcal{P}$ (separable and complete metrizable topological space) 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 $P, P_1, ..., P_N : \left(\Omega, \mathfrak{F}, \mathbb{P}\right) \to \mathcal{P}$.*

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

$$\text{Find } x^* : \mathcal{P} \to \mathbb{R}^n, \text{ s.t. } x^*(p) \in \operatorname*{argmin}_{x \in \mathbb{R}^n} \ell(x, p) \quad \mathbb{P}_P - a.s. \tag{1}$$

However, we will only solve a relaxed version of (1) and provide *generalization bounds* for the *average performance* after training on a data set.

Definition 3 *The measurable function $S : \left(\Omega, \mathfrak{F}, \mathbb{P}\right) \longrightarrow \mathcal{P}^N, \quad \omega \mapsto (P_1, ..., P_N)(\omega)$ is called a* data set*, and if the random variables $P_1, ..., P_N$ are i.i.d., that is, $\mathbb{P}_S = \bigotimes_{i=1}^N \mathbb{P}_{P_i} = \bigotimes_{i=1}^N \mathbb{P}_P$, 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 (also called a "stochastic-" or "Markov kernel"):

Definition 4 *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 (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 5 *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:

Assumption 6 *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 $H : \left(\Omega, \mathfrak{F}, \mathbb{P}\right) \longrightarrow \mathcal{H}$, which is independent of S and P. Further, the initialization $x^{(0)} \in \mathbb{R}^n$ is given and fixed.*

Notation 7 *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 $\left(\Omega, \mathfrak{F}, \mathbb{P}\right)$ will not be mentioned in the following.*

Definition 8 *Suppose P and ℓ satisfy Assumption 2, and $\mathcal{A}$ satisfies Assumption 6. The* risk *of $\mathcal{A}$ is defined as the measurable function:*

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

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

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

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

Definition 9 *Let $\emptyset \neq \Gamma$ be an index set, S a data set with data-space $\mathscr{P}^N$, and let $\mathcal{U}$ be a measurable space. A family of probability kernels $(\mathbb{Q}_\gamma)_{\gamma \in \Gamma}$ from $\mathscr{P}^N$ 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 $\eta : \Gamma \to \mathbb{R}^k$, $a : \Gamma \times \mathscr{P}^N \to (0, +\infty)$, and measurable functions $\tau : \mathcal{U} \times \mathscr{P}^N \to \mathbb{R}^k$, $b : \mathcal{U} \to (0, +\infty)$, such that $\mathbb{Q}_\gamma(s) = ba(\gamma, s) \exp\left(\langle \eta(\gamma), \tau(\cdot, s) \rangle\right) \cdot \mu$ for every $\gamma \in \Gamma$, $s \in \mathscr{P}^N$, that is, $\mathbb{Q}_\gamma(s, \mathsf{B}) = \int_{\mathsf{B}} b(u) a(\gamma, s) \exp(\langle \eta(\gamma), \tau(u, s) \rangle) \ \mu(du)$, $\mathsf{B} \in \mathfrak{B}(\mathcal{U})$.*

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 $h \in \mathscr{H}$, we make the following assumption:

Assumption 10 *On the hyperparameter space $\mathscr{H}$, we are given a data-dependent exponential family $(\mathbb{Q}_\gamma)_{\gamma \in \Gamma}$ in η and τ with dominating probability measure $\mu = \mathbb{P}_H$, such that the map $h \mapsto b(h) \exp(\langle \eta(\gamma), \tau(h, s) \rangle)$ is non-trivial and integrable w.r.t. $\mathbb{P}_H$ for every $\gamma \in \Gamma$, $s \in \mathscr{P}^N$, that is, $\mathbb{E}_H\left[b \exp(\langle \eta(\gamma), \tau(\cdot, s) \rangle)\right] \in (0, \infty)$.*

Then, as shown in Lemma 38 in Appendix B, every member of the data-dependent exponential family is indeed a data-dependent distribution on $\mathscr{H}$. In the following, the last integral in Assumption 10 will be of great interest. Here, we will use a similar notation as in Barndorff–Nielsen (2014) and denote

$$\begin{aligned} c(\gamma, s) &:= \int_{\mathscr{H}} b(h) \exp(\langle \eta(\gamma), \tau(h, s) \rangle) \ \mathbb{P}_H(dh) = \mathbb{E}_H\left[b \exp(\langle \eta(\gamma), \tau(\cdot, s) \rangle)\right] \, , \\ \kappa(\gamma, s) &:= \log\left(c(\gamma, s)\right) = \log\left(\mathbb{E}_H\left[b \exp\left(\langle \eta(\gamma), \tau(\cdot, s) \rangle\right)\right]\right) \, . \end{aligned} \tag{2}$$

With this notation, it holds that $a(\gamma, s) = c(\gamma, s)^{-1}$.

Remark 11 *(i) If η describes a lower-dimensional manifold in $\mathbb{R}^k$, $(\mathbb{Q}_\gamma)_{\gamma \in \Gamma}$ is called a* curved *exponential family (Efron, 1975), whose properties might differ from the ones for* linear *exponential families, for example, convexity of the map $\gamma \mapsto a(\gamma, s)$.*

(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 4 we will construct *η and τ such that this holds anyway.*

(iv) In the special case $b \equiv 1$ and $\eta(\gamma) \equiv \gamma$, the map $\gamma \mapsto c(\gamma, s)$ is the moment-generating function of the random variable $\tau(H, s)$. Similarly, in this case $\gamma \mapsto \kappa(\gamma, s)$ is the corresponding cumulant-generating function.

Finally, we will restrict Γ to a compact set. This is needed to get a uniform bound in γ (see Langford and Caruana, 2001; Catoni, 2007; Alquier, 2024).

Assumption 12 *Γ is a compact set with finite covering $\mathcal{O} := \{\mathsf{O}_1, ..., \mathsf{O}_\mathcal{K}\}$, that is, $\Gamma \subset \bigcup_{i=1}^{\mathcal{K}} \mathsf{O}_i$, such that there is a constant $\mathcal{C}_\mathcal{O}$, which, for every $s \in \mathcal{P}^N$, allows for the bound $\max_{i=1,...,\mathcal{K}} \sup_{\gamma,\gamma' \in \mathsf{O}_i} \kappa(\gamma, s) - \kappa(\gamma', s) \leq \mathcal{C}_\mathcal{O}$.*

Remark 13 *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 Γ is a finite set ($\mathcal{K} = |\Gamma|$, $\mathcal{C}_\mathcal{O} = 0$), or, if (Γ, ρ) is a compact metric space and κ is Lipschitz-continuous in γ (uniformly in s) with Lipschitz constant L, such that $\mathcal{C}_\mathcal{O} = L \cdot \max_{i=1,...,\mathcal{K}} \operatorname{diam} \mathsf{O}_i$, where the diameter of a set A is given by $\operatorname{diam} \mathsf{A} = \sup_{x,y \in \mathsf{A}} \rho(x, y)$.*

3 General PAC-Bayesian Theorem

In this section we prove the general PAC-Bayesian bound for data-dependent exponential families, which then can be specialized into a generalization bound of the learned parametric optimization algorithm $\mathcal{A}$. It is based on the following two lemmas, whose proofs can be found in Appendix C and D, respectively. 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 $\gamma \in \Gamma$ by controlling $\gamma \mapsto \kappa(\gamma, s)$ for every $s \in \mathcal{P}^N$.

Lemma 14 *Suppose that Assumption 10 holds and define κ as in (2). Then for every $\gamma \in \Gamma$ and $s \in \mathcal{P}^N$ it holds that $\kappa(\gamma, s) = \sup_{\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)} \mathbb{Q}\left[\langle \eta(\gamma), \tau(\cdot, s)\rangle + \log(b)\right] - D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H)$. Furthermore, for every $\gamma \in \Gamma$, the supremum is attained at $\mathbb{Q}_\gamma(s)$.*

Lemma 15 *Suppose that Assumption 12 holds and assume that $\mathbb{P}\{\kappa(\gamma, S) > t\} \leq \exp(-t)$ for all $t \in \mathbb{R}$ and $\gamma \in \Gamma$. Then $\mathbb{P}\{\sup_{\gamma \in \Gamma} \kappa(\gamma, S) \leq \log(\mathcal{K}/\epsilon) + \mathcal{C}_\mathcal{O}\} \geq 1 - \epsilon$.*

Theorem 16 *Suppose that Assumptions 10 and 12 hold, and assume that $\mathbb{E}_S[c(\gamma, \cdot)] \leq 1$ for all $\gamma \in \Gamma$. Then, it holds that:*

$$\mathbb{P}\Big\{\forall \gamma \in \Gamma,\ \forall \mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H) : \\ \mathbb{Q}\left[\langle \eta(\gamma), \tau(\cdot, s)\rangle + \log(b)\right]|_{s=S} \leq D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_\mathcal{O}\Big\} \geq 1 - \varepsilon\,.$$

Proof Applying Markov's inequality to the non-negative random variable $c(\gamma, S)$ yields for $t \in \mathbb{R}$, $\gamma \in \Gamma$:

$$\mathbb{P}\left\{c(\gamma, S) > \exp(t)\right\} \leq \frac{\mathbb{E}[c(\gamma, S)]}{\exp(t)} \leq \exp(-t)\,.$$

This implies that $\mathbb{P}\{\kappa(\gamma, S) > t\} \leq \exp(-t)$. Hence, Lemma 15 is applicable and gives:

$$\mathbb{P}\left\{\sup_{\gamma\in\Gamma} \kappa(\gamma, S) \leq \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_{\mathcal{O}}\right\} \geq 1 - \varepsilon\,.$$

Using Lemma 14 gives:

$$\mathbb{P}\left\{\sup_{\gamma\in\Gamma}\sup_{\mathbb{Q}\in\mathcal{M}_1(\mathbb{P}_H)} \mathbb{Q}[\langle\eta(\gamma), \tau(\cdot, s)\rangle + \log(b)]|_{s=S} - D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) \leq \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_{\mathcal{O}}\right\} \geq 1 - \varepsilon\,.$$

Simply rearranging and reformulating yields the result. ■

Remark 17 *(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 $\mathbb{E}\left[c(\gamma, (S, S'))\right] \leq 1$.*

(ii) In Section 4 we provide sufficient conditions s.t. $\mathbb{E}[c(\gamma, S)] \leq 1$ holds for all $\gamma > 0$.

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

(iv) Thanks to the reviewers we became aware of the monograph by Hellström et al. (2023), which proposes a similar general PAC-Bayesian theorem. On first sight, it seems like theirs is more general than ours. However, Example 19 clarifies this. Furthermore, we want to remark that the first version of our Theorem 16 appeared in 2022.

For the rest of the paper, we set $b \equiv 1$, such that $\log(b) \equiv 0$. The following corollary shows an example of how to transform Theorem 16 into a high-probability bound on the risk. The proof is given in Appendix E.

Corollary 18 (PAC-Bayesian Generalization Bound) *Denote τ by $\tau = (\tau^{(1)}, \tau^{(r)})$ with $\tau^{(r)} := (\tau^{(2)}, ..., \tau^{(k)})$ and η by $\eta = (\eta^{(1)}, \eta^{(r)})$ with $\eta^{(r)} := (\eta^{(2)}, ..., \eta^{(k)})$. If $\tau^{(1)} = \mathcal{R} - \hat{\mathcal{R}}$ and $\eta^{(1)} > 0$, the following are equivalent for any $\gamma \in \Gamma$, $s \in \mathcal{S}^N$, $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$:*

(i) $\mathbb{Q}\left[\langle\eta(\gamma), \tau(\cdot, s)\rangle\right] \leq D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_{\mathcal{O}}$,

(ii) $\mathbb{Q}[\mathcal{R}] \leq \mathbb{Q}[\hat{\mathcal{R}}(\cdot, s)] + \frac{1}{\eta^{(1)}(\gamma)}\left(D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) + \log\left(\frac{\mathcal{K}}{\varepsilon}\right) + \mathcal{C}_{\mathcal{O}} - \mathbb{Q}\left[\langle\eta^{(r)}(\gamma), \tau^{(r)}(\cdot, s)\rangle\right]\right)$.

In particular, if Theorem 16 applies, we can replace (i) with (ii).

Using similar rearrangements, the following example relates Theorem 16 to other known PAC-Bayesian bounds:

Example 19 *(i) Assume that the loss-function is bounded, that is, $0 \leq \ell \leq C$, and define $\Gamma = \{\gamma\}$, $b \equiv 1$, $\mathcal{C}_{\mathcal{O}} = 0$, $\tau(h, s) := \left(\mathcal{R}(h) - \hat{\mathcal{R}}(h, s),\ C^2\right)$, and $\eta(\gamma) := \left(\gamma, -\frac{\gamma^2}{8}\right)$. Then we recover Catoni's bound (Catoni, 2003; Alquier, 2024):*

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

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

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

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 $f, g : \mathscr{H} \times \mathscr{P}^N \to \mathbb{R}$, and define $\Gamma = \{\gamma\}$, $b \equiv 1$, $\mathcal{C}_\emptyset = 0$, $\tau(h,s) := (f(h,s), g(h,s))$, and $\eta(\gamma) := (\gamma, -\gamma)$. Then our assumption $\mathbb{E}_S[c(\gamma,\cdot)] \leq 1$ reads $\mathbb{E}_{(S,H)}\left[\exp(\gamma(f-g))\right] \leq 1$, which is the same assumption as in Hellström et al. (2023, Thm. 5.1). Similarly, defining $\eta(\gamma) := (1,-1)$ and $T(h,s) := (f(h,s),\ \log(\mathbb{E}_{(S,H)}[\exp(f)]))$, we also get a similar bound as Hellström et al. (2023, Proposition 5.2):*

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

4 Learning-to-Optimize with Guarantees

Here, for our setting in Subsection 2.2, we consider properties of optimization algorithms that assert the necessary condition of Theorem 16, namely $\mathbb{E}[c(\gamma, S)] \leq 1$ for all $\gamma \in \Gamma$, with c defined as in Equation 2, to employ the PAC-Bayesian bound from Section 3.

4.1 Worst-Case Bounds

In the next theorem, the additional assumption on $\mathcal{A}$ is sufficient to ensure the conditions of Theorem 16. 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 4.2, it is too restrictive to achieve a significant acceleration compared to the standard choices from a worst-case analysis. For this, 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 $x^{(0)}$.

Theorem 20 *Suppose that P and ℓ satisfy Assumption 2, and suppose that $\mathcal{A}$ satisfies Assumption 6. Further, assume that there is a measurable function $\rho : \mathscr{H} \longrightarrow [0,\infty)$, such that for every $h \in \mathscr{H}$ 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 $\mathbb{E}\big[\ell(x^{(0)},P)^2\big] < \infty$, and define $\eta : (0,\infty) \longrightarrow \mathbb{R}^2$ and $\tau : \mathscr{H} \times \mathscr{P}^N \longrightarrow \mathbb{R}^2$ through:*

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

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

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

$$\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}(H) - \hat{\mathcal{R}}(H,S))\right)\right] = \mathbb{E}\left[\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S))\right)\right]\Big|_{h=H}\right].$$

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

$$\begin{aligned}\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S))\right)\right] &= \mathbb{E}\left[\exp\left(-\frac{\gamma}{N}\sum_{i=1}^{N}\left(\ell(h,P_i) - \mathbb{E}[\ell(h,P)]\right)\right)\right] \\ &= \prod_{i=1}^{N}\mathbb{E}_P\left[\exp\left(-\frac{\gamma}{N}\left(\ell(h,\cdot) - \mathbb{E}_P[\ell(h,\cdot)]\right)\right)\right].\end{aligned}$$

The loss-function is non-negative and, by assumption on $\mathcal{A}$, can be bounded $\mathbb{P}_P$-a.s. Thus, for every $h \in \mathscr{H}$, $\ell(h,P)$ is a non-negative random variable with finite second-moment, as $\mathbb{E}_P[\ell(h,\cdot)^2] \leq \rho(h)^2\mathbb{E}_P\left[\ell(x^{(0)},\cdot)^2\right] < \infty$. Hence, by Lemma 39, we get:

$$\begin{aligned}\mathbb{E}_P\left[\exp\left(-\frac{\gamma}{N}\left(\ell(h,\cdot) - \mathbb{E}_P[\ell(h,\cdot)]\right)\right)\right] &\leq \exp\Big(\frac{\gamma^2}{2N^2}\mathbb{E}_P[\ell(h,\cdot)^2]\Big) \\ &\leq \exp\Big(\frac{\gamma^2}{2N^2}\rho(h)^2\mathbb{E}_P[\ell(x^{(0)},\cdot)^2]\Big).\end{aligned}$$

Therefore we have the following bound:

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

This can be rearranged into $\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}(h) - \hat{\mathcal{R}}(h,S)) - \frac{\gamma^2}{2N}\rho(h)^2\mathbb{E}_P[\ell(x^{(0)},\cdot)^2]\right)\right] \leq 1$, as the right-hand side does not depend on S. Since H and S are independent, and $h \in \mathscr{H}$ was arbitrary, this inequality does hold $\mathbb{P}_H$-a.s. Therefore, one directly gets the bound $\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}(H) - \hat{\mathcal{R}}(H,S)) - \frac{\gamma^2}{2N}\rho(H)^2\mathbb{E}_P[\ell(x^{(0)},\cdot)^2]\right)\right] \leq 1$. Now, again by Fubini's theorem, one can also switch the order of integration to get:

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

Inserting the definition of η and τ gives $\mathbb{E}\left[\mathbb{E}\left[\exp\left(\langle\eta(\gamma),\tau(H,s)\rangle\right)\right]|_{s=S}\right] \leq 1$. Here, the inner term is the same as $\mathbb{E}\left[\exp\left(\langle\eta(\gamma),\tau(H,s)\rangle\right)\right] = \int_{\mathscr{H}}\exp\left(\langle\eta(\gamma),\tau(h,s)\rangle\right)\,\mathbb{P}_H(dh) = c(\gamma,s)$. Hence, this is the same as $\mathbb{E}[c(\gamma,S)] \leq 1$. ■

Remark 21 *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* $\mathbb{E}\left[f(H,S,S')\right] = \mathbb{E}_{S'}\left[\mathbb{E}_{H|S'=s'}\left[\mathbb{E}_S\left[f(h,\cdot,s')\right]|_{h=H}\right]\right] = \mathbb{E}_{S'}\left[\mathbb{E}_S\left[\mathbb{E}_{H|S'=s'}\left[f(\cdot,s,s')\right]|_{s=S}\right]\right]$, *and the inner term is* ≤ 1 *in any case.*

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

Example 22 *Consider $\ell(x,p) := \frac{p}{2}x^2$ and assume that the chosen algorithm is gradient descent, that is $x^{(k+1)} = x^{(k)} - h\ell'(x^{(k)}, p)$. 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 $(\frac{99}{100})^k$ 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.*

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 5, we develop a technique that allows the user to actually control this probability. Clearly, a stronger guarantee trades for convergence speed.

Definition 23 *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 $\mathsf{L}_\sigma := \{(h,p) \in \mathcal{H}\times\mathcal{P} \ : \ \ell(h,p) \le \sigma(p)\}$. The sections of L_σ for fixed $h \in \mathcal{H}$ will be denoted by $\mathsf{L}_{\sigma,h}$.*

In Lemma 40 we show that L_σ is indeed a measurable set. This is not obvious, as the loss function and the algorithm are composed in a non-standard way. This result further implies that the sections $\mathsf{L}_{\sigma,h}$ are measurable, too. Since $\mathcal{H}$ and $\mathcal{P}$ are Polish spaces, the product $\mathcal{H}\times\mathcal{P}$ is again Polish. Hence, there exists a regular version of the conditional probability of P, given H, that is, a kernel $\mathcal{H} \to \mathcal{P}$, $(h, \mathsf{B}) \mapsto \mathbb{P}_{P|H=h}[\mathsf{B}]$. By Witting (2013, Thm. 1.122, p.124), this determines a regular version of the conditional probability of (H,P), given H, through $\mathcal{H} \to \mathcal{H}\times\mathcal{P}$, $(h,\mathsf{B}) \mapsto \mathbb{P}_{(H,P)|H=h}[\mathsf{B}] := \mathbb{P}_{P|H=h}[\mathsf{B}_h]$, 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 24 *Let L_σ be a parametric sublevel set. Define the* sublevel probability *as the measurable function $h \mapsto \rho(h) := \mathbb{P}_{P|H=h}[\mathsf{L}_{\sigma,h}]$.*

This construction allows us to give a more fine-grained analysis of the algorithm, as it allows to trade 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 41, we define the *sublevel risk* as the expect loss *conditioned* on the sublevel set:

Definition 25 *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 $\mathsf{L}_{\sigma,h}$:*

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

Given a data set $S = (P_1, ..., P_N)$, the empirical sublevel risk $\hat{\mathcal{R}}_\sigma : \mathscr{H} \times \mathscr{P}^N \to [0, +\infty]$ *is defined as $(h, S) \mapsto \hat{\mathcal{R}}_\sigma(h, S) := \frac{1}{\rho(h)}\frac{1}{N}\sum_{i=1}^N \mathbb{1}_{\mathsf{L}_{\sigma,h}}(P_i)\ell(h, P_i)$.*

The following theorem is a direct generalization of Theorem 20. Especially, note that the additional assumption on $\mathcal{A}$ is not needed anymore.

Theorem 26 *Suppose that P and ℓ satisfy Assumption 2, and suppose that $\mathcal{A}$ satisfies Assumption 6. 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 ρ. Assume that $\mathbb{P}_H\{\rho > 0\} = 1$ and $\mathbb{E}_P[\sigma^2] < \infty$. Define $\eta : (0, \infty) \longrightarrow \mathbb{R}^2$ and $\tau : \mathscr{H} \times \mathscr{P}^N \longrightarrow \mathbb{R}^2$ as*

$$\eta(\gamma) := \left(\gamma,\ -\frac{\gamma^2}{2}\right), \quad \tau(h, s) := \left(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h, s),\ \frac{1}{\rho(h)^2 N}\mathbb{E}_P\left[\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\right]\right).$$

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

Proof The proof is very similar to the proof of Theorem 20 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:

$$\mathbb{E}\left[\exp(\gamma(\mathcal{R}_\sigma(H) - \hat{\mathcal{R}}_\sigma(H, S)))\right] = \mathbb{E}\left[\mathbb{E}\left[\exp(\gamma(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h, S)))\right]\Big|_{h=H}\right].$$

Thus, first consider a fixed $h \in \mathscr{H}$ with $p(h) > 0$. Then, by definition and the i.i.d. assumption, it holds that:

$$\begin{aligned}\mathbb{E}\left[\exp(\gamma(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h, S)))\right] &= \mathbb{E}\left[\exp\left(-\frac{\gamma}{N\rho(h)}\sum_{i=1}^N \left(\ell_\sigma(h, P_i) - \mathbb{E}_P[\ell_\sigma(h,\cdot)]\right)\right)\right] \\ &= \prod_{i=1}^N \mathbb{E}_P\left[\exp\left(-\frac{\gamma}{N\rho(h)}\left(\ell_\sigma(h,\cdot) - \mathbb{E}_P[\ell_\sigma(h,\cdot)]\right)\right)\right].\end{aligned}$$

$\ell_\sigma(h,\cdot)$ 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 39 we have the inequality $\mathbb{E}_P\left[\exp\left(-\frac{\gamma}{N\rho(h)}\left(\ell_\sigma(h,\cdot) - \mathbb{E}_P[\ell_\sigma(h,\cdot)]\right)\right)\right] \leq \exp\left(\frac{\gamma^2}{2N^2\rho(h)^2}\mathbb{E}\left[\ell_\sigma(h,\cdot)^2\right]\right)$. Thus:

$$\mathbb{E}\left[\exp(\gamma(\mathcal{R}_\sigma(h,\cdot) - \hat{\mathcal{R}}_\sigma(h, S)))\right] \leq \exp\left(\frac{\gamma^2}{2N\rho(h)^2}\mathbb{E}_P\left[\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\right]\right).$$

This can be rearranged into $\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}_\sigma(h) - \hat{\mathcal{R}}_\sigma(h, S)) - \frac{\gamma^2}{2N\rho(h)^2}\mathbb{E}_P\left[\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\right]\right)\right] \leq 1$, since the right-hand side is independent of S. As this holds for any h with $\rho(h) > 0$, which in turn does hold $\mathbb{P}_H$-a.s., we get

$$\mathbb{E}\left[\exp\left(\gamma(\mathcal{R}_\sigma(H) - \hat{\mathcal{R}}_\sigma(H, S)) - \frac{\gamma^2}{2N\rho(H)^2}\mathbb{E}_P\left[\sigma^2\mathbb{1}_{\mathsf{L}_{\sigma,h}}\right]|_{h=H}\right)\right] \leq 1\,.$$

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

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

Using the definition of η and τ, this is the same as $\mathbb{E}\left[\mathbb{E}\left[\exp(\langle\eta(\gamma),\tau(H,s)\rangle)\right]|_{s=S}\right]\leq 1$. Thus, in total we get $\mathbb{E}\left[c(\gamma,S)\right]\leq 1$, because the inner term can be rewritten as $\mathbb{E}\left[\exp(\langle\eta(\gamma),\tau(H,s)\rangle)\right]=\int_{\mathscr{H}}\exp(\langle\eta(\gamma),\tau(h,s)\rangle)\;\mathbb{P}_H(dh)=c(\gamma,s)$. ■

Remark 27 *The assumption $\mathbb{P}_H\{\rho>0\}=1$ 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 5 provides a construction for achieving this.*

Example 28 *Combining Theorem 26 and Theorem 16, we get that:*

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

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

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

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 $\rho(H)\geq\rho_l$ a.s., this gives rise to:*

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

5 Implementing the Non-divergence – Speed Trade-Off

In Subsection 4.2, 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 $\frac{1}{\rho(h)}$ 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:

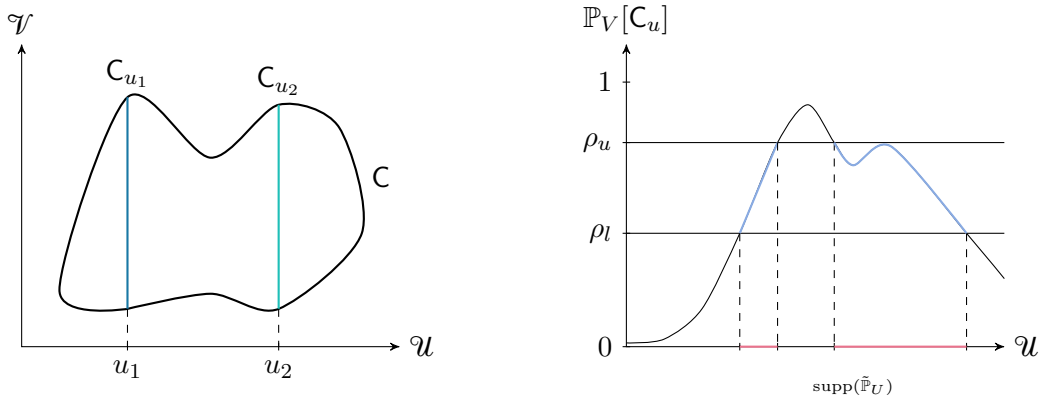

Figure 2: Construction of $\tilde{\mathbb{P}}_U$: On the left, the set $\mathsf{C} \subset \mathcal{U} \times \mathcal{V}$ and two of its sections $\mathsf{C}_{u_1}, \mathsf{C}_{u_2} \subset \mathcal{V}$ are visualized. On the right, the function $\rho(u) = \mathbb{P}_V[\mathsf{C}_u]$, the interval $[\rho_l, \rho_u]$, and the resulting support $\mathrm{supp}(\tilde{\mathbb{P}}_U)$ of $\tilde{\mathbb{P}}_U$ are visualized. Note that, contrary to the visualization here, ρ can actually be *highly discontinuous*.

Lemma 29 *Let $\varepsilon_{\mathrm{conv}} \in [0,1]$ and assume that $\rho(H) \geq \varepsilon_{\mathrm{conv}}$ a.s. Then, for every $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$ we have $\mathbb{Q}\{\rho < \varepsilon_{\mathrm{conv}}\} = 0$.*

Proof By assumption we have $\mathbb{P}_H\{\rho < \varepsilon_{\mathrm{conv}}\} = 0$. 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.

5.1 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 $U : (\Omega, \mathfrak{F}, \mathbb{P}) \to \mathcal{U}$, $V : (\Omega, \mathfrak{F}, \mathbb{P}) \to \mathcal{V}$ 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, 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:

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

This allows us to define the (measurable) function $\rho : \mathcal{U} \to [0,1]$, $u \mapsto \mathbb{P}_{V|U=u}[\mathsf{C}_u]$. By independence of U and V, this is $\mathbb{P}_V$-almost surely the same as $\rho(u) = \mathbb{P}_V[\mathsf{C}_u] \in [\rho_l, \rho_u]$, and we will use the later formulation from now on.[2] Thus, for $\rho_l, \rho_u \in [0,1]$ with $\rho_l < \rho_u$,

2. Note that ρ is still measurable.

Algorithm 1 Iterative estimation of the probability ρ

Require: $q_l, q_u, \varepsilon \in [0,1]$.
  $a, b \leftarrow 1, 1$ ▷ Initialize with uninformative prior.
  while $Q_{a,b}(q_u) - Q_{a,b}(q_l) \geq \varepsilon$ **do** ▷ $Q_{a,b}$ is the quantile function for Beta(a,b).
    Draw $I \sim \mathrm{Ber}(\rho)$
    $a \leftarrow a + I$ and $b \leftarrow b + (1 - I)$
  end while

we can define a measurable set $\mathsf{A} := \left\{u \in \mathcal{U} \ : \ \mathbb{P}_V[\mathsf{C}_u] \in [\rho_l, \rho_u]\right\}$, 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:

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

Therefore, as stated before, we have the following goal:

Goal: Sample from $\tilde{\mathbb{P}}_U$, that is, get $U_1, ..., U_K \sim \mathbb{P}_U$, such that $\mathbb{P}_V[\mathsf{C}_u]|_{u=U_i} \in [\rho_l, \rho_u]$.

This construction is depicted in Figure 2: 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 30 *Consider the random variables P and H from Section 4. By Lemma 29 we want to have $\rho(H) \in [\varepsilon_{\mathrm{conv}}, 1]$, where the sublevel probability is given as $\rho(h) = \mathbb{P}_P[\mathsf{L}_{\sigma,h}]$ (Lemma 41), and the sublevel set $\mathsf{L}_\sigma \subset \mathscr{H} \times \mathscr{P}$ is measurable by Lemma 40. Thus, this corresponds to the identification $\mathcal{U} = \mathscr{H}$, $\mathcal{V} = \mathscr{P}$, and $\rho_l = \varepsilon_{\mathrm{conv}}$, $\rho_u = 1$.*

5.1.1 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 unnormalised 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 $V_1, ..., V_n \sim \mathbb{P}_V$, 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 $\mathbb{P}\{I_n = 1\} = \mathbb{P}\{V_n \in \mathsf{C}_u\} = \mathbb{P}_V[\mathsf{C}_u] = \rho(u)$. Thus, by estimating $\rho(u)$ with an estimator $\hat{\rho}(u)$, we approximate the constraint A with $\hat{\mathsf{A}}$:

$$\mathsf{A} = \{u \in \mathcal{U} \ : \ \rho(u) \in [\rho_l, \rho_u]\} \approx \{u \in \mathcal{U} \ : \ \hat{\rho}(u) \in [\rho_l, \rho_u]\} =: \hat{\mathsf{A}}\,.$$

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 and Casella, 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 $\rho(u)$ in a Bayesian way, as it allows us to balance accuracy against computational complexity through uncertainty-quantification, which we use as a stopping criterion:

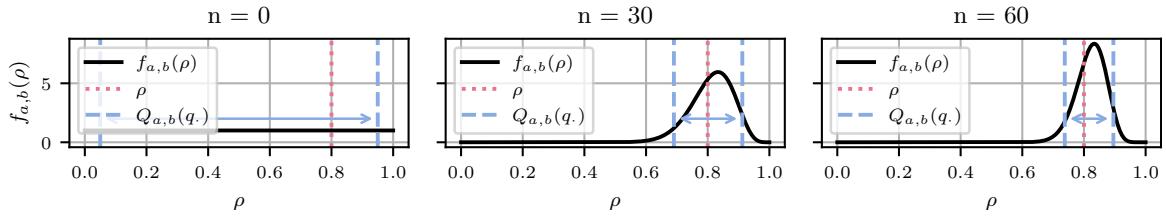

Figure 3: Iterative estimation of $\rho(u)$: The black line shows the density $f_{a^{(n)},b^{(n)}}$ of $\text{Beta}(a^{(n)}, b^{(n)})$ after having observed $I_1, ..., I_n$. The red dotted line indicates the true probability $\rho(u)$, which we are trying to estimate, and the blue dashed lines indicate the lower and upper quantiles corresponding to q_l, q_u. The procedure stops as soon as $Q_{a^{(n)},b^{(n)}}(q_u) - Q_{a^{(n)},b^{(n)}}(q_l) < \varepsilon$, which is indicated by the double-headed arrow. Here, we use $q_l = 0.05, q_u = 0.95, \varepsilon = 0.15$ and $\rho(u) = 0.8$.

We place a Beta-prior $\text{Beta}(a^{(0)}, b^{(0)})$ over the interval $[0, 1]$. As we do not have prior knowledge, and the map $u \mapsto \rho(u)$ can be discontinuous[3], we use a noninformative prior (Berger, 1985, Ch. 3.3), that is, $a_0 = b_0 = 1$. 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 $a^{(k)}, b^{(k)}$ get updated as:

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

This allows us to do the estimation iteratively: We only draw a new sample I_{n+1} as long as $Q^{(n)}(q_u) - Q^{(n)}(q_l) \geq \varepsilon$, where $Q^{(n)}$ denotes the quantile-function of $\text{Beta}(a^{(n)}, b^{(n)})$, and $q_u, q_l, \varepsilon \in [0, 1]$ are parameters that specify the accuracy of the estimation. Finally, one can use the posterior mean $\frac{a^{(n)}}{a^{(n)}+b^{(n)}}$ or posterior mode $\frac{a^{(n)}-1}{a^{(n)}+b^{(n)}-2}$ (provided $a^{(n)}, b^{(n)} > 1$) as point estimate $\hat{\rho}_u$. 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 $\rho(u) \approx 0$ or $\rho(u) \approx 1$, the uncertainty decreases significantly faster than for $\rho(u) \approx 0.5$. This procedure is summarized in Algorithm 1 and depicted in Figure 3.

5.1.2 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 and Adams (2014) introduce an auxiliary binary variable $z_n \in \{0, 1\}$, which allows for querying only a subset of the data for the computation of the exact likelihood. And Quiroz et al. (2018) combine subsampling with a bias-correction strategy to

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

Algorithm 2 Probabilistically constrained sampling

Require: $\rho_l, \rho_u \in [0,1]$, $n_{\max} \in \mathbb{N}$, $u_0 \in \hat{\mathsf{A}}$.
  $n \leftarrow 0$ and $u \leftarrow u_0$
  while $n \leq n_{\max}$ **do**
    1) Draw a proposal u' with SGLD starting from u.
    2) Estimate $\rho(u') = \mathbb{P}_V[\mathsf{C}_{u'}]$ by $\hat{\rho}(u')$ with Algorithm 1.
    if $\hat{\rho}(u') \in [\rho_l, \rho_u]$ **then**
      $u \leftarrow u'$
    else
      Reject u'.
    end if
  end while

speed-up the sampling procedure. A summary of different approaches is given by Bardenet et al. (2017). We leave the analysis for our proposed approximation to future work.

5.1.3 Choice of the Sampling Procedure

Often, the hyperparameters $h \in \mathscr{H}$ are high-dimensional. Thus, we use *stochastic gradient Langevin dynamics* (Welling and Teh, 2011) (SGLD) 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 31 *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 4: The upper row shows the function $u \mapsto \rho(u)$, and the potential from which we want to sample with the constraint $\rho(u) \in [0.6, 1]$. 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.

6 Learning Procedure

This section deals with the implementation of the learning procedure, and translates the abstract framework discussed in Sections 3 and 4 into concrete design choices. Thus, this marks the beginning of the second part of the paper, which is less theoretical. The resulting learning procedure is visualized in Figure 5 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

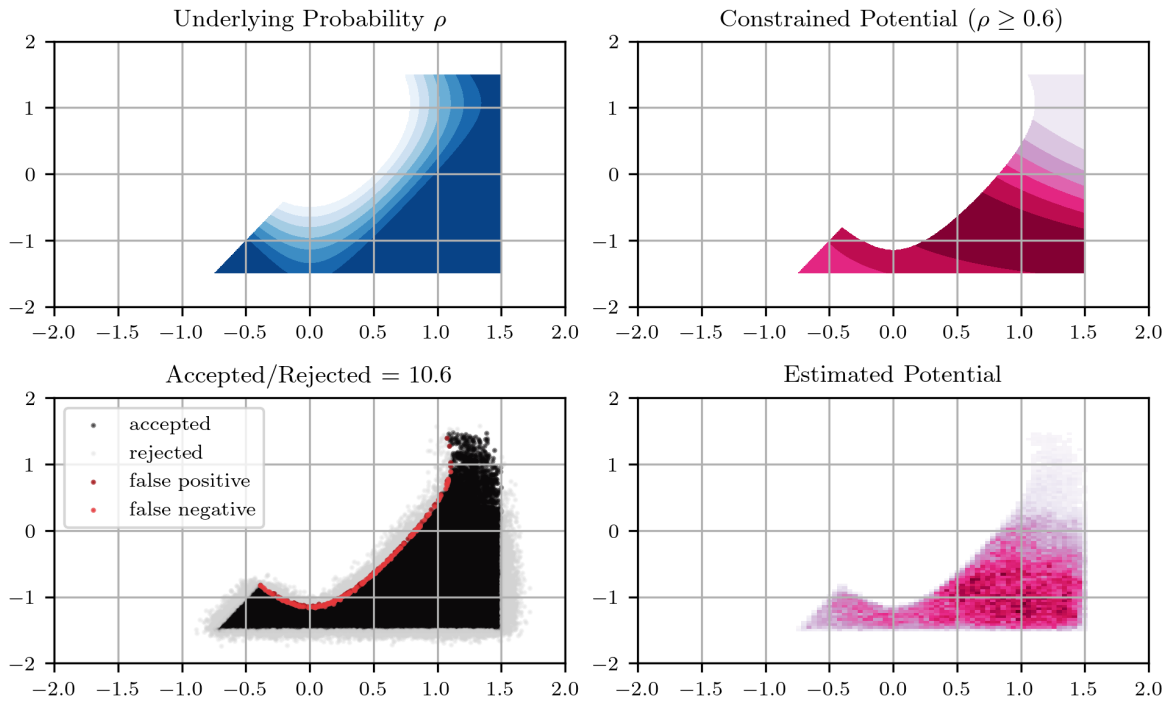

Figure 4: Example for probabilistically constrained sampling: The upper left plot shows the underlying function $\rho(u)$. It is discontinuous and defines a non-convex set A. The upper right plot shows the probabilistically constrained potential ($\rho(u) \in [0.6, 1]$), from which we want to sample. The lower left plot shows the accepted (black) and the rejected (gray) samples (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 $\rho(u) \approx 0.6$, where the remaining uncertainty can easily lead to a wrong decision. Here, we have chosen the $q_l = 0.01$, $q_u = 0.99$, and $\varepsilon = 0.05$ in Algorithm 1. Finally, the lower right plot shows the estimated potential.

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 $h^{(0)} \in \mathscr{H}$ that a) satisfies the constraint in Subsection 4.2 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 $\gamma^* \in \Gamma$, which allows for computing the optimal posterior distribution $\mathbb{Q}_{\gamma^*}$ in closed-form.

The outline of this section is as follows: In Subsection 6.1 we identify the optimal posterior $\mathbb{Q}^*$ in the abstract setting. In Subsection 6.2, we describe the pre-computation phase in (i).

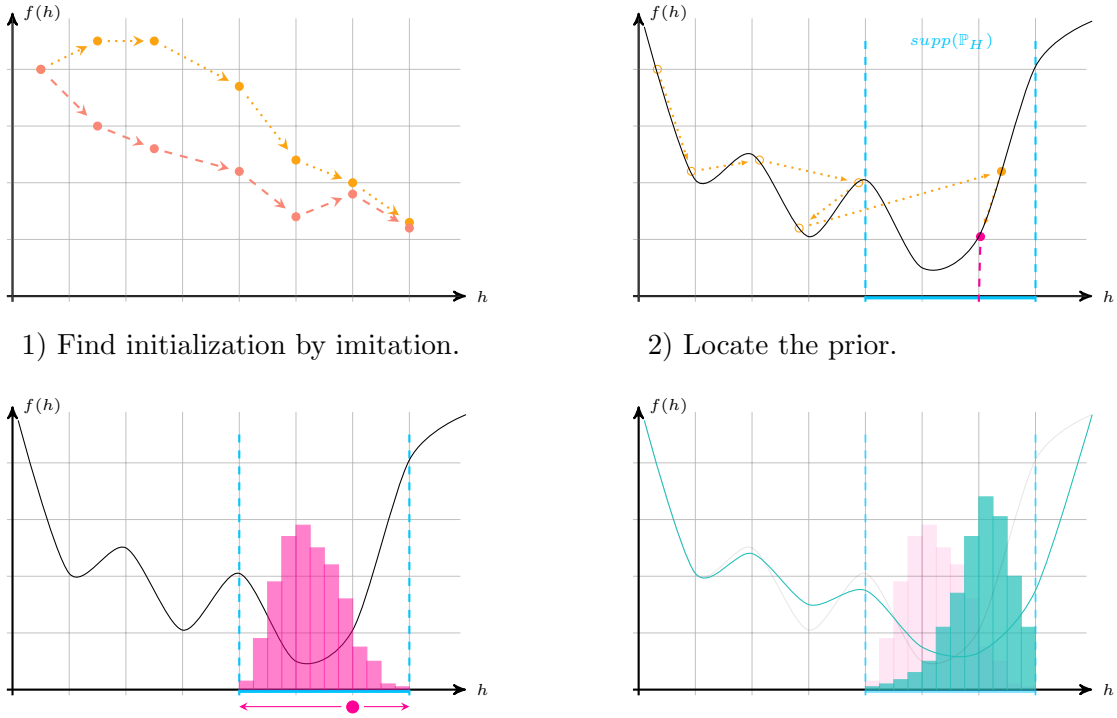

Figure 5: Learning procedure: **1)** Imitation learning. **2)** Probabilistically constrained stochastic empirical risk minimization. **3)** Construct prior through sampling. **4)** Compute posterior by performing the PAC-Bayesian learning step.

Subsections 6.3 and 6.4 deal with the concrete design choices in (ii) and (iii) to construct the prior, and Subsection 6.5 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 use the generic S, implicitly assuming the above partitioning.

Remark 32 *Through the choice of the sampling algorithm, the concrete learning procedure described here mainly applies to the case $\mathscr{H} = \mathbb{R}^d$, $d \in \mathbb{N}$. Nevertheless, the general methodology is still applicable to other Polish spaces, if this choice can be adjusted accordingly.*

6.1 Minimization of the PAC-Bound

Learning is phrased as minimizing the PAC-Bayesian upper-bound. Hence, in this subsection we consider η, τ and item (ii) from Corollary 18, and we seek for $\gamma \in \Gamma$ and $\mathbb{Q} \in \mathcal{M}_1(\mathbb{P}_H)$

that minimize the upper-bound, that is, we want to solve:

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

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

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

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 14 shows that the term inside the brackets is given by $\tilde{\kappa}(\gamma,s) - \log(K/\varepsilon) - C_{\mathcal{O}}$, where $\tilde{\kappa}$ corresponds to the exponential family $(\tilde{\mathbb{Q}}_\gamma)_{\gamma\in\Gamma}$ 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 $\tilde{\mathbb{Q}}_\gamma(s) \propto \exp(\langle\eta(\gamma),\tilde{\tau}(\cdot,s)\rangle)\cdot\mathbb{P}_H$, usually called the Gibbs posterior (Alquier, 2024). By denoting $F(\gamma,s) := -\frac{1}{\eta^{(1)}(\gamma)}(\tilde{\kappa}(\gamma,s) - \log(\mathcal{K}/\varepsilon) - \mathcal{C}_{\mathcal{O}})$, one is left with solving the following problem:

$$\inf_{\gamma\in\Gamma}\ F(\gamma,s), \tag{3}$$

which for $\Gamma\subset\mathbb{R}$ is one-dimensional. Based on Theorem 26, we restrict to $\Gamma\subset(0,+\infty)$, such that the solution to (3) can be seen as an approximation to the global minimum $\inf_{\gamma>0}F(\gamma,s)$. For the latter one, one can show that the solution set lies in a compact interval $[\Gamma_{\min},\Gamma_{\max}]$, since $F(\gamma,s)\to\infty$ as $\gamma\to 0$ or $\gamma\to\infty$. Under our assumptions, $F(\cdot,s)$ is continuously differentiable. Hence, since Γ is compact, $F(\cdot,s)$ is Lipschitz-continuous on Γ and the minimum in (3) is attained. For a finite set $\Gamma=\{\gamma_1,...,\gamma_K\}\subset[\Gamma_{\min},\Gamma_{\max}]$, the optimization reduces to grid search. For $\Gamma=[\Gamma_{\min},\Gamma_{\max}]$, we employ grid search as initialization for gradient-based optimization. Here, the computational bottleneck is given by evaluating $\gamma\mapsto\tilde{\kappa}(\gamma,s)$. In Sections 6.4 and 6.5 we will ensure that this is cheap.

6.2 Finding a Trainable Initialization

To increase numerical stability, we start with "imitation learning" (Chen et al., 2020a), 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 iteration number $t\in\mathbb{N}$, and a parameter $p\in\mathcal{P}$, denote the first t iterates of $\mathcal{A}(h,p,x^{(0)})$ by $x^{(1)},...,x^{(t)}\in\mathbb{R}^n$ and the ones of $\mathcal{A}'(x^{(0)},p)$ by $y^{(1)},...,y^{(t)}\in\mathbb{R}^n$. Then, define the loss as the mean squared error over these iterations:

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

In each iteration, that is, each prediction of tuples $(x^{(1)},y^{(1)}),...,(x^{(t)},y^{(t)})$, the parameters p, $x^{(0)}$ and t are randomized as described in Section 6.3. 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.

Algorithm 3 Procedure to find an initialization

Require: Data set s_{prior}, $x^{(0)} \in \mathbb{R}^n$, $t, n_{\text{init}} \in \mathbb{N}$ and $\varepsilon > 0$.
  $m \leftarrow +\infty$ and sample $p \sim U_{s_{\text{prior}}}$.
  while $\frac{1}{n_{\text{init}}} m > \varepsilon$ **do**
    $m \leftarrow 0$
    for $i = 1, ..., n_{\text{init}}$ **do**
      1) Compute $(x^{(1)}, y^{(1)}), ..., (x^{(t)}, y^{(t)})$ with $\mathcal{A}(h, p, x^{(0)})$ and $\mathcal{A}'(p, x^{(0)})$, resp.
      2) Compute $\ell_{\text{init}}(h, p, x^{(0)}, t) = \frac{1}{t} \sum_{k=1}^{t} \|x^{(k)} - y^{(k)}\|_2^2$.
      3) Update $m \leftarrow m + \ell_{\text{init}}(h, p, x^{(0)}, t)$
      4) Update h by backpropagation and Adam. ▷ Other algorithms possible.
      5) Update p, $x^{(0)}$ and t based on Section 6.3.
    end for
  end while

6.3 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:

6.3.1 Ratio of Losses

The canonical loss function to be minimized is the empirical risk $\hat{\mathcal{R}}(h, s) = \frac{1}{N} \sum_{i=1}^{N} \ell(h, p_i)$, and, if $\mathscr{H}$ is high-dimensional or if N is large, one resorts to stochastic empirical risk minimization. While this kind of loss was used extensively before, for learning-to-optimize it has a strong disadvantage: Only the overall outcome after $n_{\max}$ 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 $\tilde{\ell}_{\text{train}}(h, p, x^{(0)}) := \sum_{i=1}^{n} \ell(x^{(i)}, p)$. Again, 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*:

$$\ell_{\text{train}}(h, p, x^{(0)}, t) := \sum_{i=1}^{t} \mathbb{1}_{\{\ell(x^{(i-1)}, p) > 0\}} \frac{\ell(x^{(i)}, p)}{\ell(x^{(i-1)}, p)}, \qquad t \in \mathbb{N},\ t \leq n\,.$$

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

Algorithm 4 Procedure to locate the prior

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

Set $x \leftarrow x^{(0)}$, $b \leftarrow$ **false**, and sample $p \sim \mathrm{U}_{s_{\text{prior}}}$ ▷ b = Point inside constraint?

for $i = 1, ..., n_{\max}$ **do** ▷ Other stopping criteria possible.

1.a) Compute $x^{(1)}, ..., x^{(t)}$ with $\mathcal{A}(h, p, x^{(0)})$.

1.b) Compute $\ell_{\text{train}}(h, p, x^{(0)}, t) = \sum_{i=1}^{t} \mathbb{1}_{\{\ell(x^{(i-1)},p)>0\}} \frac{\ell(x^{(i)},p)}{\ell(x^{(i-1)},p)}$.

1.c) Construct a proposal $\tilde{h}$ by using backpropagation and Adam.

2) Estimate $\rho(\tilde{h})$ by $\hat{\rho}(\tilde{h})$ with Algorithm 1 on s_{val}.

if $\hat{\rho}(\tilde{h}) \in [\rho_l, \rho_u]$ **then** ▷ If point inside constraint, just update.

$h \leftarrow \tilde{h}$ and $b \leftarrow$ **true**

else ▷ If not...

if $b =$ **true then** ▷ ...reject moving outside constraint.

Reject $\tilde{h}$, set $x^{(0)} \leftarrow x$, sample $p \sim \mathrm{U}_{s_{\text{prior}}}$, and continue with 1).

else ▷ ...accept, if constraint has not been found yet.

$h \leftarrow \tilde{h}$

end if

end if

3) Draw $R \sim \mathrm{Ber}(\frac{t}{n_{\text{train}}})$.

if $R = 0$ **then**

$x^{(0)} \leftarrow x^{(t)}$

else $R = 1$

$x^{(0)} \leftarrow x$ and sample $p \sim \mathrm{U}_{s_{\text{prior}}}$

end if

end for

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.

6.3.2 Randomized Trajectory Length

Training $\mathcal{A}$ with fixed initialization $x^{(0)}$ and fixed trajectory length leads to overfitting: Applying it at another starting 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 $t \leq n_{\text{train}}$ and set $y := x^{(0)}$.

0) Sample a parameter p uniformly at random from s.

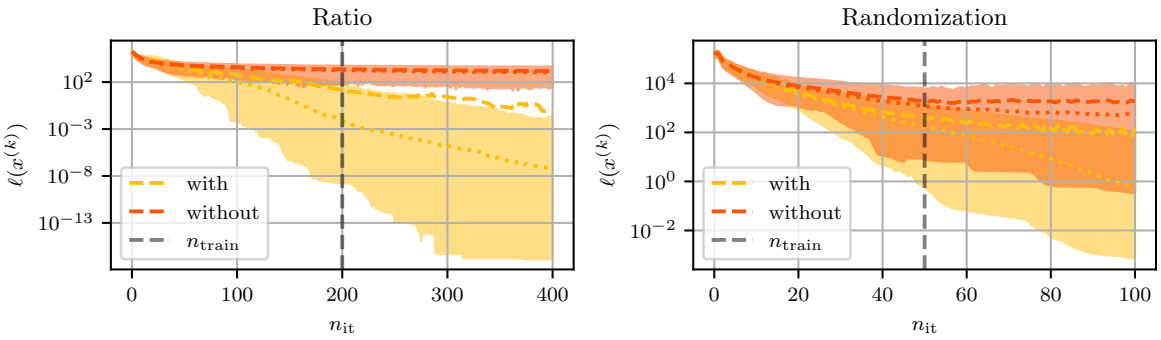

Figure 6: Effect of our design choices: Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded region represents 95% of the data. The yellow algorithm was trained with our design choice and the orange one without. Besides that, everything else was kept the same. 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 n_{train} *and* an overall better performance.

1) Compute $x^{(1)}, ..., x^{(t)}$ with $\mathcal{A}(h, p, y)$ and the loss $\ell_{\text{train}}(h, p, y, t)$, and update h.

2) Sample $R^{(k)} \sim \text{Ber}(\frac{t}{n_{\text{train}}})$. If $R^{(k)} = 0$, set $y := x^{(t)}$ 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 continuous the current trajectory. The choice $\frac{t}{n_{\text{train}}}$ ensures that the expected trajectory-length equals n_{train}: Define $Z := \inf\{k \in \mathbb{N} \ : \ R^{(k)} = 1\}$. Then, $Z \sim \text{Geo}(\frac{t}{n_{\text{train}}})$ is a geometrically distributed with expectation $\mathbb{E}[Z] = \frac{n_{\text{train}}}{t}$. Therefore, for the actual length $L = t \cdot Z$ of the trajectory we get $\mathbb{E}[L] = t\mathbb{E}[Z] = n_{\text{train}}$.

Remark 33 *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 $t = 1$. That 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 6 shows the effect of these two design choices: The left plot shows the effect of using the ration 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 (yellow), and one time without (orange). 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

Algorithm 5 Procedure to construct the prior

Require: Data sets s_{prior} (sampling) and s_{val} (constraint), $n_{\text{sam}} \in \mathbb{N}$ and $h \in \text{supp}(\tilde{\mathbb{P}}_h)$.
1) Starting from h, run Algorithm 2 (with ℓ_{train}) to get the points $h_1, ..., h_{n_{\text{sam}}} \in \mathscr{H}$.
2) Evaluate φ_{prior} on $\{h_1, ..., h_{n_{\text{sam}}}\}$ by evaluating $\hat{\mathcal{R}}_g$ corresponding to s_{prior}.
3) Compute $\mathbb{P}_H\{h_j\}$, that is, $\mathbb{P}_H\{h_j\} = \sigma\left(\varphi_{\text{prior}}(h_1), ..., \varphi_{\text{prior}}(h_{n_{\text{sam}}})\right)_j$.

procedure. Further details can be found in the GitHub-repository. The overall procedure is summarized in Algorithm 4.

6.4 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, by Lemma 14 the functional form of the posterior is fully specified, namely it is of the form:

$$\mathbb{Q}_\gamma(s) \propto \exp(\varphi_\gamma(\cdot, s)) \cdot \mathbb{P}_H, \qquad \gamma \in \Gamma\,, \tag{4}$$

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

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

where μ is a measure on $\mathscr{H}$, which allows to sample from $\mathbb{P}'$ (possibly unnormalized). In our experiments it holds $\mathscr{H} = \mathbb{R}^d$ 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 $h_1, ..., h_{n_{\text{sam}}} \in \mathscr{H}$, $n_{\text{sam}} \in \mathbb{N}$, 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 $\mathscr{H}$ corresponding to $h_1, ..., h_{n_{\text{sam}}}$, where the normalization constant is given by $Z = \sum_{i=1}^{n_{\text{sam}}} w_i$ with $w_i = \exp\left(-\hat{\mathcal{R}}_{\sigma,\text{prior}}(h_i) - \iota_{[\rho_l,\rho_u]}\left(\hat{\rho}(h_i)\right)\right)$. When $h_1, ..., h_{n_{\text{sam}}} \in \mathscr{H}$ are given, the corresponding probabilities can equivalently be expressed with the so-called *softmax* function $\sigma(x_1, ..., x_n)_j = \frac{\exp(x_j)}{\sum_{i=1}^n \exp(x_i)}$ and the potential $\varphi_{\text{prior}}(h) = -\hat{\mathcal{R}}_{\sigma,\text{prior}}(h) - \iota_{[\rho_l,\rho_u]}\left(\hat{\rho}(h)\right)$:

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

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

Remark 34 *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

Require: 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) Evaluate $\tilde{T}(h_i, s)$, $i = 1, ..., n_{\text{sam}}$.
2) Setup $\{\varphi_\gamma(h_1, s), ..., \varphi_\gamma(h_{n_{\text{sam}}}, s)\}$ as functions in γ.
3) Solve $\gamma^* \in \operatorname{argmin}_{\gamma \in \Gamma} \; F(\gamma, s)$. ▷ $F(\gamma^*, s)$ is the predicted PAC-bound.
4) Compute $\mathbb{Q}_{\gamma^*}(s, \{h_j\}) = \sigma\left(\varphi_{\gamma^*}(h_1, s), ..., \varphi_{\gamma^*}(h_{n_{\text{sam}}}, s)\right)_j$, $j = 1, ..., n_{\text{sam}}$.
5) Optional: Choose $h^* = \operatorname{argmax}_{i=1,...,n_{\text{sam}}} \mathbb{Q}_{\gamma^*}(s, \{h_i\})$ as final point-estimate.

6.5 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 (4), for every $\gamma \in \Gamma$ the optimal posterior $\mathbb{Q}_\gamma(s)$ is given by:

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

with the potential $\varphi_\gamma(h, s) = \langle \eta(\gamma), \tilde{\tau}(h, s)\rangle + \varphi_{\text{prior}}(h)$. Thus, to get the distribution $\mathbb{Q}_\gamma(s)$ as a function of γ, 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 $\gamma^* \in \Gamma$, which is found as described in Section 6.1. After evaluating the potentials $\varphi_\gamma(\cdot, s)$, which has to be done anyway, evaluating $\tilde{\kappa}(\cdot, s)$ in γ is cheap. The process is summarized in Algorithm 6.

7 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, the *non-smooth* Lasso problem, and the *non-smooth and non-convex* problem of training a neural network. More details on the implementation, especially a detailed description of the architectures of the algorithms and how we construct the parameters for each problem, is given in Appendix F. 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.

7.1 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:

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

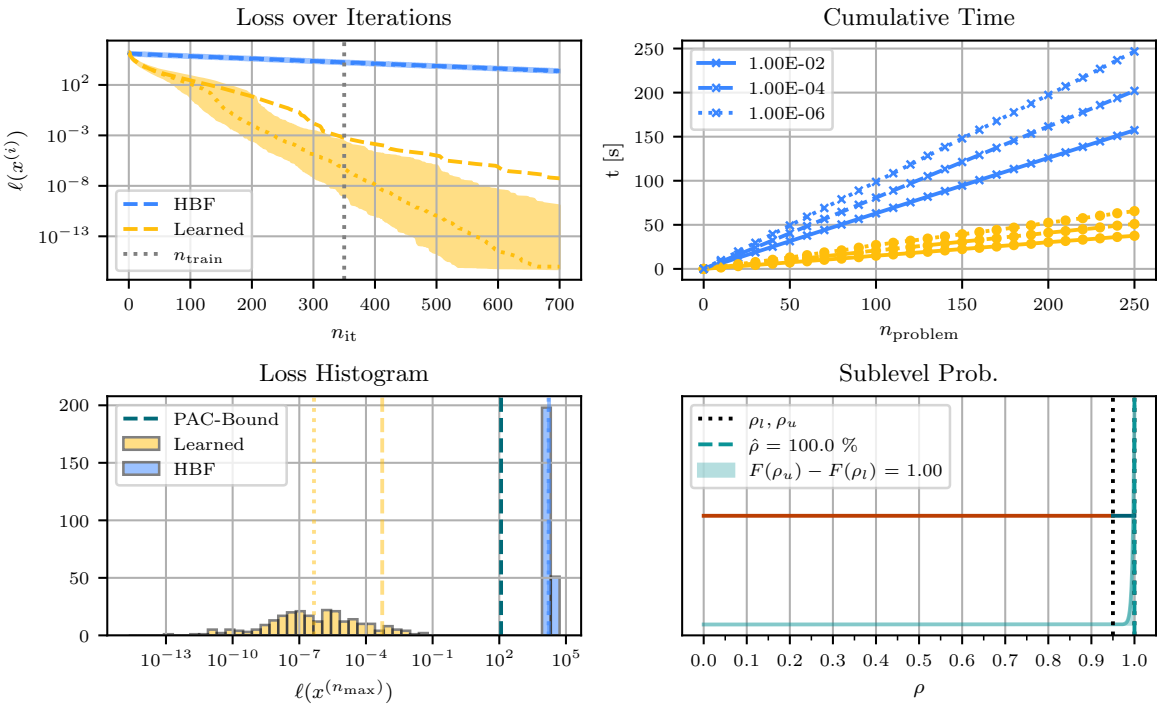

Figure 7: **Upper left:** Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions represent the 10th to 90th percentile. The learned algorithm $\mathcal{A}$ is shown in yellow, while *heavy-ball with friction* (HBF) is shown in blue. **Upper right:** The different lines indicate the cumulative computation time the algorithms need to solve all the test problems up to a certain accuracy (in function values) measured by $\ell(x^{(i)}, p) < \varepsilon$. However, note that both algorithms are run for maximally $n_{\max} = 1\text{e}4$ iterations. **Lower left:** Loss histogram after $n_{\text{train}} = 350$ iterations with predicted PAC-bound. **Lower right:** The teal dashed line shows the point estimate for the sublevel probability, while the teal solid line shows the Beta-posterior. The black dotted lines indicate the constraints ρ_l, ρ_u and show the feasible region as dark teal line.

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 $x^{(k+1)} = x^{(k)} - \alpha \nabla f(x^{(k)}) + \beta \left(x^{(k)} - x^{(k-1)}\right)$, 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). Further details can be found in Appendix F.1. Figure 7 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 $n_{\text{train}} = 350$ 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" 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 p_l and p_u in all test cases.

7.2 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:

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

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 $y^{(k+1)} = x^{(k)} + \frac{t_k - 1}{t_{k+1}}(x^{(k)} - x^{(k-1)})$ followed by setting $x^{(k+1)} = y^{(k)} - \alpha \nabla f(y^{(k+1)})$. We use the optimal choices $t_{k+1} = \frac{1}{2}\left(1 + \sqrt{1 + 4t_k^2}\right)$ and $\alpha = \frac{1}{L}$. Here, the smoothness constant L is given by the largest eigenvalue of $A^TA + \frac{\lambda}{\varepsilon} D^T D$, where $D \in \mathbb{R}^{2n\times n}$ is given by "stacking" D_h and D_w, that is, $D = \begin{pmatrix} D_h & D_w \end{pmatrix}^T$. Further details can be found in Appendix F.2. The results of this experiment are summarized in Figure 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. Finally, the lower right plot shows that, while the algorithm did not reach the sublevel set in all of the test cases, the probabilistically constraint optimization/sampling procedure did work correctly.

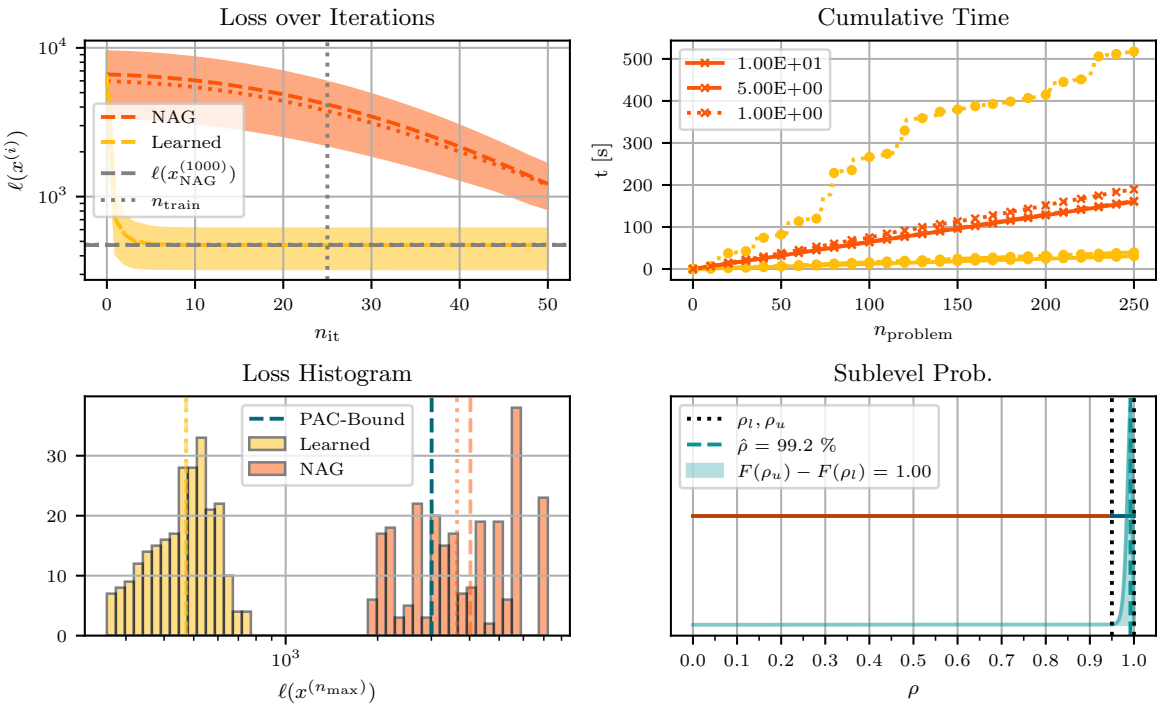

Figure 8: **Upper left:** Dashed lines represent mean losses, dotted lines show median losses, and the shaded regions represent the 10th to 90th percentile. The learned algorithm $\mathcal{A}$ is shown in yellow, while Nesterovs *accelerated gradient descent* (NAG) is shown in orange. **Upper right:** The different lines show the cumulative computation time the algorithms need to solve the test problems up to a certain accuracy (in function values) measured by $\ell(x^{(i)}, p) - \ell(x^{(1000)}_{\text{std}}, p) < \varepsilon$. However, note that both algorithms are run for maximally $n_{\max} = 1000$ iterations. **Lower left:** Loss histogram after $n_{\text{train}} = 50$ iterations with predicted PAC-bound. **Lower right:** The teal dashed line shows the point estimate for the sublevel probability, while the teal solid line shows the Beta-posterior. The black dotted lines indicate the constraints ρ_l, ρ_u and show the feasible region as dark teal line.

7.3 Lasso-Problem

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 and Teboulle (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 $y^{(k)} = x^{(k)} + \beta^{(k)}\left(x^{(k)} - x^{(k-1)}\right)$ followed by setting $x^{(k+1)} = \operatorname{prox}_{\alpha g}\left(y^{(k)} - \alpha\nabla h(y^{(k)})\right)$. Here, the proximal mapping is defined as $\hat{x} = \operatorname{prox}_{\alpha g}(\bar{x}) = \operatorname{argmin}_{x\in\mathbb{R}^n}\ \lambda\|x\|_1 + \frac{1}{2\alpha}\|x - \bar{x}\|_2^2$, and can be computed efficiently in closed-form yielding the so-called *soft-thresholding operator*:

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

We choose $\alpha = 1/L$, where L is the largest eigenvalue of A^TA, that is, the smoothness parameter of h, while $\beta^{(k)}$ is set to $\beta^{(k)} := (t_k - 1)/t_{k+1}$ with $t_{k+1} = (1 + \sqrt{1 + 4t_k^2})/2$. Further details about the experiment can be found in Appendix F.3. The results of this experiment are summarized in Figure 9: 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. And the lower right plot indicates that the algorithm did reach the sublevel set in all of the test cases.

7.4 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 $\beta \in \mathbb{R}^m$ of a neural network $\mathtt{N}_\beta$, which then is used to predict a function $g : \mathbb{R} \to \mathbb{R}$. Hence, the optimization variable is given by $\beta \in \mathbb{R}^m$. As baseline we use Adam (Kingma and Ba, 2015) (as it is implemented in PyTorch), 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 n_{train} iterations. Note that originally Adam was introduced for stochastic optimization, while we use it in the "full-batch setting" here. Further details can be found in Appendix F.4. Figure 10 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.

8 Discussion and Limitations

The motivation for this paper was to use more structure in a given problem than is analytically tractable. For this, we considered a distribution over parametric loss functions and

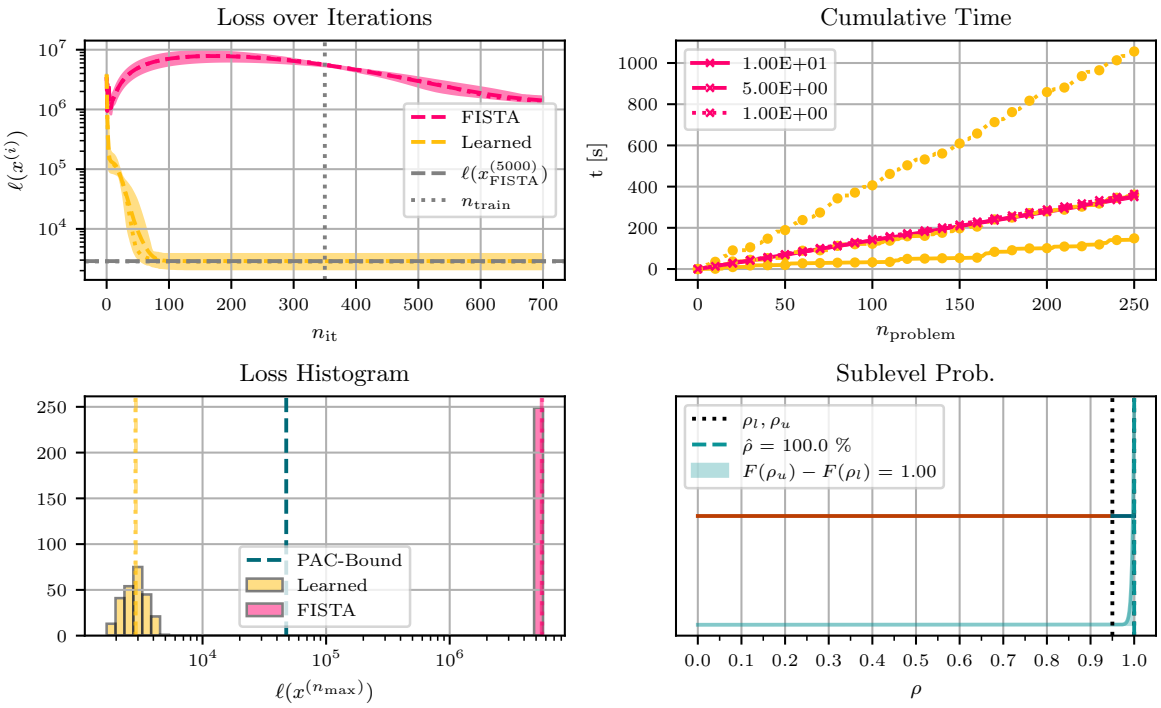

Figure 9: **Upper left:** Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded region represents the 10th to 90th percentile. Here, the *fast iterative shrinkage-thresholding algorithm* (FISTA) is shown in pink and the learned algorithm in yellow. The gray horizontal lines represent the loss achieved by FISTA after 5000 iterations, which serves as approximation for the solution. **Upper right:** The different lines show the cumulative computation time the algorithms need to solve all the test problems up to a certain accuracy (in function-values) measured by $\ell(x^{(i)}, p) - \ell(x^{(5000)}_{\text{std}}, p) < \varepsilon$. However, note that both algorithms are run for maximally $n_{\max} = 5000$ iterations. **Lower left:** Loss histogram (after $n_{\text{train}} = 350$ iterations) with the predicted PAC-bound. **Lower right:** The teal dashed line shows the point estimate for the sublevel probability, while the teal solid line shows the Beta-posterior. The black dotted lines indicate the constraints ρ_l, ρ_u and show the feasible region as dark teal line.

formulated the (ultimate) goal in (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. 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,

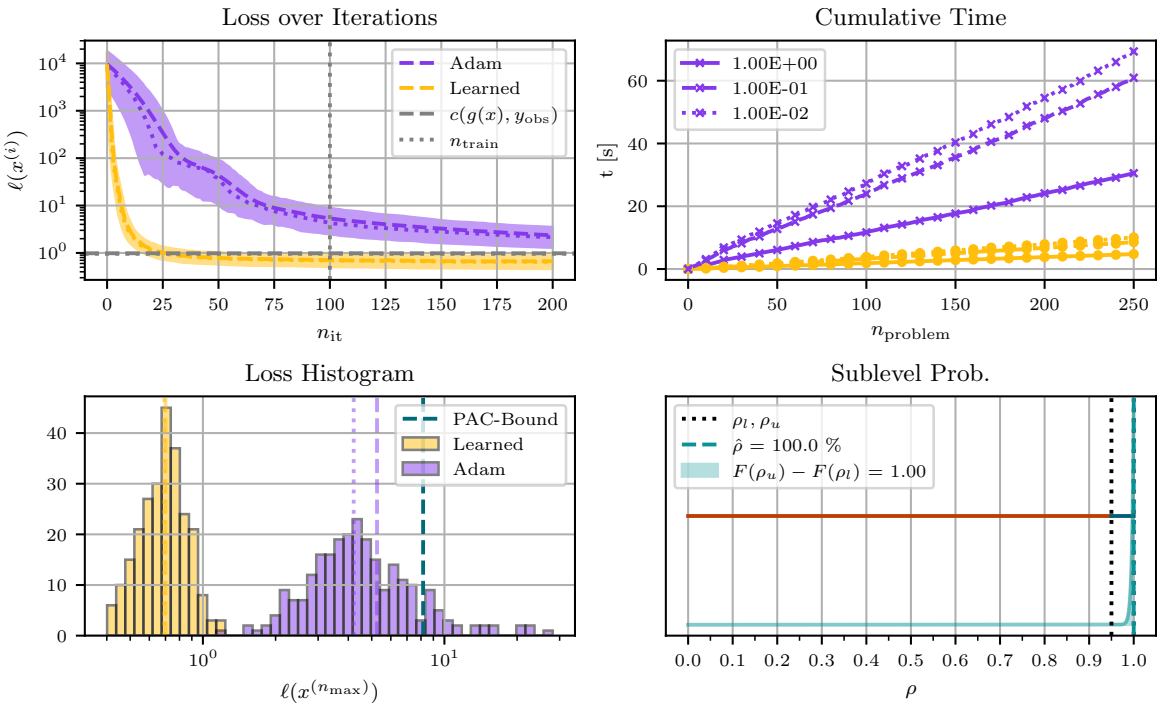

Figure 10: **Upper left:** Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions indicate the 10th to 90th percentile. The vertical dotted line shows n_{train}, and the horizontal gray dashed line represents the average loss of the ground-truth function g (equal to one, as we added standard Gaussian noise). Here, Adam is shown in purple and the learned algorithm in yellow. **Upper right:** The different lines show the cumulative computation time of the algorithms to solve all the test problems up to a certain accuracy (in function values) measured by $\ell(\beta^{(i)}, p) - c(X_i, Y_i) < \varepsilon$. However, note that both algorithms are run for maximally $n_{\max} = 5000$ iterations. **Lower left:** Loss histogram (after n_{train} iterations) and PAC-bound. **Lower right:** The teal dashed line shows the point estimate for the sublevel probability, while the teal solid line shows the Beta-posterior. Here, the black dotted lines indicate the constraints ρ_l, ρ_u and show the feasible region as dark teal line.

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. All these are promising directions of research that we leave to a future work.

Acknowledgments and Disclosure of Funding

M. Sucker and P. Ochs acknowledge funding by the German Research Foundation under Germany's Excellence Strategy – EXC number 2064/1 – 390727645. Furthermore, J. Fadili and P. Ochs are supported by the ANR-DFG joint project TRINOM-DS under the numbers ANR-20-CE92-0037-01 and OC150/5-1.

Appendix A. Supplementary Definitions

Definition 35 (Probability Kernel) *Let $(\mathcal{U}, \mathfrak{U})$, $(\mathcal{V}, \mathfrak{V})$ be measurable spaces. A function $\mu : \mathcal{U} \times \mathfrak{V} \to [0, \infty]$, $(u, \mathsf{A}) \mapsto \mu(u, \mathsf{A})$ is called a* kernel *from $\mathcal{U}$ to $\mathcal{V}$, written as $\mu : \mathcal{U} \to \mathcal{V}$, if for every set $\mathsf{A} \in \mathfrak{V}$, the map $u \mapsto \mu(u, \mathsf{A})$ is measurable, and for every point $u \in \mathcal{U}$, the map $\mathsf{A} \mapsto \mu(u, \mathsf{A})$ is a measure. Furthermore, μ is called a* probability kernel *from $\mathcal{U}$ to $\mathcal{V}$, if $\mu(u, \mathcal{V}) = 1$ for every $u \in \mathcal{U}$.*

Definition 36 (Exponential Family) *Let Γ be a non-empty index set. A family of probability measures $(\mathbb{Q}_\gamma)_{\gamma \in \Gamma}$ on a measurable space $\mathcal{U}$ is called an* exponential family *(in η and τ), if there is a dominating probability measure μ, that is, $(\mathbb{Q}_\gamma)_{\gamma \in \Gamma} \subset \mathcal{M}_1(\mu)$, functions $\eta : \Gamma \to \mathbb{R}^k$, $a : \Gamma \to (0, +\infty)$, and measurable functions $\tau : \mathcal{U} \to \mathbb{R}^k$, $b : \mathcal{U} \to (0, +\infty)$, such that for every $\gamma \in \Gamma$ we have $\mathbb{Q}_\gamma = ba(\gamma) \exp(\langle \eta(\gamma), \tau \rangle) \cdot \mu$, that is, $\mathbb{Q}_\gamma[\mathsf{B}] = \int_{\mathsf{B}} b(u) a(\gamma) \exp(\langle \eta(\gamma), \tau(u) \rangle)\ \mu(du)$, $\mathsf{B} \in \mathfrak{B}(\mathcal{U})$.*

Definition 37 (Support of a Measure) *Let $\mathcal{U}$ be a topological space, and let μ be a measure on $\mathcal{U}$. The* support *of μ is defined as:*

$$\operatorname{supp}(\mu) := \{u \in \mathcal{U} \ : \ \mu[\mathsf{B}] > 0 \text{ for every neighborhood } \mathsf{B} \text{ of } u\}.$$

Appendix B. Supplementary Lemmas

Lemma 38 *Under Assumption 10, $\mathbb{Q}_\gamma$ is a data-dependent distribution for every $\gamma \in \Gamma$.*

Proof Denote the density of $\mathbb{Q}_\gamma$ w.r.t. $\mathbb{P}_H$ by $f_\gamma(h, s) := \frac{b(h)}{c(\gamma, s)} \exp(\langle \eta(\gamma), \tau(h, s) \rangle)$. The map $c(\gamma, \cdot) : \mathcal{P}^N \to [0, \infty)$ is $\mathfrak{B}(\mathcal{P}^N)$ measurable, as τ is measurable w.r.t. the product-σ-algebra and $\mathbb{P}_H$ is a finite measure (Kallenberg, 2021, Lemma 1.28, p.25). Hence, f_γ is measurable w.r.t. $\mathfrak{B}(\mathscr{H}) \otimes \mathfrak{B}(\mathcal{P}^N)$, since $c(\gamma, s) \in (0, \infty)$. Thus, it holds that $\mathbb{Q}_\gamma = f_\gamma \cdot \mathbb{P}_H$ is a kernel from from $\mathcal{P}^N$ to $\mathscr{H}$ (Kallenberg, 2021, Lem. 3.2, p.56). Finally, $\mathbb{Q}_\gamma : \mathcal{P}^N \to \mathscr{H}$ is actually a probability kernel, since $c(\gamma, s)$ is the corresponding normalization constant. ■

The following result states that non-negative random variables with finite second moment satisfy a one-sided sub-Gaussian inequality (Boucheron et al., 2013, p.47).

Lemma 39 *Let U be a non-negative random variable with finite second moment. Then, for every $\gamma > 0$ it holds $\mathbb{E}[\exp(-\gamma(U - \mathbb{E}[U]))] \leq \exp\left(\frac{\gamma^2}{2} \mathbb{E}[U^2]\right)$.*

Lemma 40 *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, $\ell \circ \mathcal{A} : \mathscr{H} \times \mathcal{P} \to [0, +\infty]$, $(h, p) \mapsto \ell(\mathcal{A}(h, p), p)$ is measurable w.r.t. $\mathfrak{B}(\mathscr{H}) \otimes \mathfrak{B}(\mathcal{P})$ and $\mathfrak{B}([0, +\infty])$. Since $\ell \geq 0$ is measurable, there exists a sequence of simple[4] 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 characteristic functions of

4. A function is called simple, if it is of the form $\ell_n = \sum_{i=1}^K a_n^i \mathbb{1}_{\mathsf{A}_n^i}$ with disjoint sets A_n^i.

the form $\mathbb{1}_{\mathsf{A}}$ for a measurable set $\mathsf{A} \in \mathfrak{B}(\mathbb{R}^n) \otimes \mathfrak{B}(\mathscr{P})$. 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. ■

Lemma 41 *Suppose Assumption 6 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) $\rho(h) = \mathbb{P}_P[\mathsf{L}_{\sigma,h}]$,

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

Proof By the independence of P and H, we have $\mathbb{E}\left\{\ell(H,P)\mathbb{1}_{\mathsf{L}_\sigma}(H,P) \mid H = h\right\} = \int_{\mathscr{P}} \ell(h,p)\mathbb{1}_{\mathsf{L}_{\sigma,h}}(p)\ \mathbb{P}_P(dp) = \mathbb{E}_P[\ell(h,\cdot)\mathbb{1}_{\mathsf{L}_{\sigma,h}}]\ \mathbb{P}_H$-a.s., 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 $\mathbb{P}_P[\mathsf{L}_{\sigma,h}] = \mathbb{P}_{P|H=h}[\mathsf{L}_{\sigma,h}] = \rho(h)$, which shows (i) and the second equality of (ii). ■

Appendix C. Proof of Lemma 14

Proof Take any $\gamma \in \Gamma$ and $s \in \mathscr{P}^N$. 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:

$$D_{\mathrm{KL}}(\mathbb{Q} \parallel \mathbb{P}_H) = \mathbb{Q}\left[\log(f)\right] = \mathbb{P}_H\left[f\log(f)\right] = \mathbb{P}_H\left[\mathbb{1}_{\{f>0\}} f \log(\mathbb{1}_{\{f>0\}} f)\right] .$$

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

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

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

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

Rearranging yields $\kappa(\gamma,s) = \int_{\mathscr{H}} \log(b(h)) + \langle \eta(\gamma), \tau(h,s)\rangle\ \mathbb{Q}_\gamma(s,dh) - D_{\mathrm{KL}}(\mathbb{Q}_\gamma(s) \parallel \mathbb{P}_H)$. ■

Appendix D. Proof of Lemma 15

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

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

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

$$\begin{aligned}\mathbb{P}\left\{ \sup_{\gamma\in\Gamma} \kappa(\gamma, S) > t \right\} &\leq \mathbb{P}\left\{ \max_{i=1,...,\mathcal{K}} \kappa(\gamma_i, S) + \mathcal{C}_{\mathcal{O}} > t \right\} \leq \sum_{i=1}^{\mathcal{K}} \mathbb{P}\left\{ \kappa(\gamma_i, S) + \mathcal{C}_{\mathcal{O}} > t \right\} \\ &\leq \sum_{i=1}^{\mathcal{K}} \exp(\mathcal{C}_{\mathcal{O}} - t) = \mathcal{K} \exp(\mathcal{C}_{\mathcal{O}} - t) \,.\end{aligned}$$

Taking $t = \log\left(\frac{\mathcal{K}}{\epsilon}\right) + \mathcal{C}_{\mathcal{O}}$ gives $\mathbb{P}\left\{\sup_{\gamma\in\Gamma} \kappa(\gamma, S) > \log\left(\frac{\mathcal{K}}{\epsilon}\right) + \mathcal{C}_{\mathcal{O}}\right\} \leq \epsilon$. ■

Appendix E. Proof of Corollary 18

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

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

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

Appendix F. Implementation Details

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 $g(p) := \alpha\ell(x^{(0)}, p)^\beta$, $\alpha, \beta > 0$, to define the sublevel set L_σ. In Algorithm 1, we use $\rho_l = 0.95$, $\rho_u = 1.0$, $q_l = 0.01$, $q_u = 0.99$, 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. 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 with an initial step-size of 10^{-4}, which gets reduced by a factor of 0.5 every $2 \cdot 10^4$ iterations, for a total of $n_{\max} = 2 \cdot 10^5$ iterations. We use a trajectory length of $t = 1$, 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 Γ with $|\Gamma| = 75 \cdot 10^3$, and an accuracy (of the PAC-bound) of $\varepsilon = 0.05$. 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 $\|\nabla \ell(x^{(k)}, p)\|$ and the corresponding unit vector $d_1^{(k)}$. Further, the norm is transformed to $n_1^{(k)} := \log(1 + \|\nabla \ell(x^{(k)}, p)\|)$ to be less scale-sensitive. The iterates $x^{(k)}, x^{(k-1)}$ are combined into the momentum term $m^{(k)} := x^{(k)} - x^{(k-1)}$, which also is split into the unit vector $d_2^{(k)}$ and the transformed norm $n_2^{(k)}$. Similarly, we also transform the function values into $\ell_1^{(k)} = \log(1 + \ell(x^{(k)}, p))$ and $\ell_2^{(k)} = \log(1 + \ell(x^{(k-1)}, p))$.

Remark 42 *(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 $\varphi_\gamma(\cdot, s)$ on a single sample $h \in \mathscr{H}$, one has to compute all losses $\ell(h, p_i)$, $i = 1, ..., N_{\text{prior}} + N_{\text{train}}$, that is, "solving" $N_{\text{prior}} + N_{\text{train}}$ optimization problems.

F.1 Details for the Experiment on Quadratic Functions

This subsection describes the missing details for the experiment on quadratic functions.

F.1.1 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 $a_{ii}^j = \sqrt{m_j} + i \cdot \frac{\sqrt{L_j} - \sqrt{m_j}}{n}$, $i = 1, ..., n$, 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 $b_1, ..., b_N \overset{iid}{\sim} \mathcal{N}(\mu, \Sigma)$, where we create μ and $\Sigma = C^T C$ by sampling $\mu_i, C_{i,k} \overset{iid}{\sim} \mathrm{U}_{[-5,5]}$, $i, k = 1, ..., n$.

F.1.2 Algorithm

The algorithmic update of the learned algorithm $\mathcal{A}$ is visualized in Figure 11 and consists of two blocks: The update-block combines the gradient direction $d_1^{(k)}$, the momentum direction $d_2^{(k)}$, and their "interaction" $d_1^{(k)} \odot d_2^{(k)}$ into the new update-direction $d^{(k)}$, while the other block computes a step-size based on the corresponding logarithmically transformed norms $n_1^{(k)}$ and $n_2^{(k)}$, and the logarithmically transformed function values $\ell_1^{(k)}$ and $\ell_2^{(k)}$.

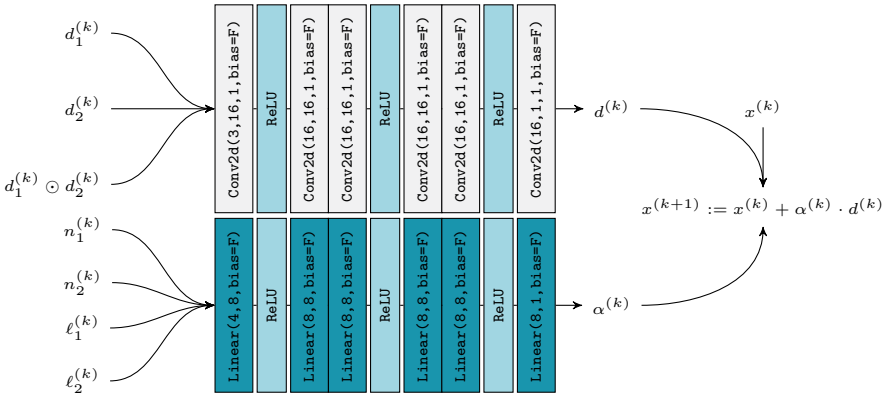

Figure 11: Update step of $\mathcal{A}$ for quadratic problems: The directions $d_1^{(k)}$, $d_2^{(k)}$ and $d_1^{(k)} \odot d_2^{(k)}$ are inserted as different channels into the `Conv2d`-block, which performs 1×1 "convolutions", that is, the algorithm acts coordinate-wise on the input, and yields an new update-direction $d^{(k)}$. The scales $n^{(1)}$, $n^{(2)}$, and the function values $\ell_1^{(k)}$, $\ell_2^{(k)}$ get transformed separately by the fully-connected block to yield the step-size $\alpha^{(k)}$.

F.2 Details for the Image-Processing Experiment

This subsection describes the missing details for the image-processing experiment.

F.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 $\varepsilon_{i,j} \overset{iid}{\sim} \mathcal{N}(0, \sigma)$, $i = 1, ..., N_h$, $j = 1, ..., N_w$. The regularization parameters $\lambda_i \in \mathbb{R}$, $i = 1, ..., N$, are given by sampling uniformly, that is, $\lambda_i \overset{iid}{\sim} \mathrm{U}_{[\lambda_-, \lambda_+]}$, where we use $\lambda_- = 0.05$ and $\lambda_+ = 0.5$.

F.2.2 Algorithm

The algorithmic update of $\mathcal{A}$ is visualized in Figure 12 and consists of an update-block, which combines $d_1^{(k)}$, $d_2^{(k)}$ and their "interaction" $d_1^{(k)} \odot d_2^{(k)}$ into the new update direction $d^{(k)}$, 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 about an image-processing problem into the design of the optimization algorithm.

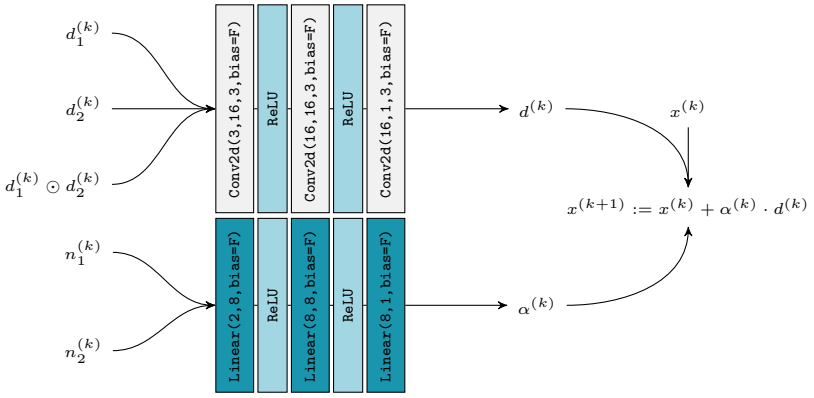

Figure 12: Update step of $\mathcal{A}$ for the image-processing problems: The directions $d_1^{(k)}$, $d_2^{(k)}$ and $d_1^{(k)} \odot d_2^{(k)}$ are inserted as different channels (in the shape of the image) into the `Conv2d`-block, which performs a 3×3-convolution. The scales $n^{(1)}$, $n^{(2)}$ get transformed separately by the fully-connected block.

F.3 Details for the LASSO Experiment

This subsection describes the missing details for the LASSO experiment.

F.3.1 Construction of the Parameters

The same matrix $A \in \mathbb{R}^{p \times n}$ with dimensions $n = 350$ and $m = 70$ is used for all problem instances. Here, we sample each entry uniformly, that is, $a_{i,j} \overset{iid}{\sim} \mathrm{U}_{[-0.5,0.5]}$, $i = 1, ..., m$, $j = 1, ..., n$. 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, $\lambda_i \overset{iid}{\sim} \mathrm{U}_{[\lambda_-, \lambda_+]}$, $i = 1, ..., N$, with $\lambda_- = 5$ and $\lambda_+ = 10$, while the right-hand side is sampled from a multivariate normal distribution, that is, $b_i \overset{iid}{\sim} \mathcal{N}(\mu, \Sigma)$, $i = 1, ..., N$, where we first construct μ and $\Sigma = C^T C$ by sampling each entry of μ and C uniformly at random in $[-5, 5]$.

F.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 $x^{(k)}$ (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, $x_{\neq}^{(k)}$ and $x_0^{(k)}$.Then, first, we compute weights $w_1, ..., w_8$ with a fully-connected block with ReLU-activation functions, where we use the features $n_1^{(k)} = \log(1 + \|\nabla \ell(x^{(k)}, p)\|)$, $n_2^{(k)} = \log(1 + \|x^{(k)} - x^{(k-1)}\|)$, $n_3^{(k)} = \log(1 + \|p^{(k)}\|)$, where $p^{(k)} = \mathrm{prox}_{\beta g}\left(x^{(k)} - \beta \nabla \ell(x^{(k)}, p)\right)$, $\Delta \ell^{(k)} := \ell(x^{(k)}, p) - \ell(x^{(k-1)}, p)$, $\Delta g^{(k)} := g(x^{(k)}) - g(x^{(k-1)})$, $\Delta h^{(k)} := h(x^{(k)}, p) - h(x^{(k-1)}, p)$, the scalar products $s_{\neq}^{(k)}$ and $s_0^{(k)}$ between the (normalized) gradient and (normalized) momentum, and the regularization parameter λ. Then, we use these weights to perform a

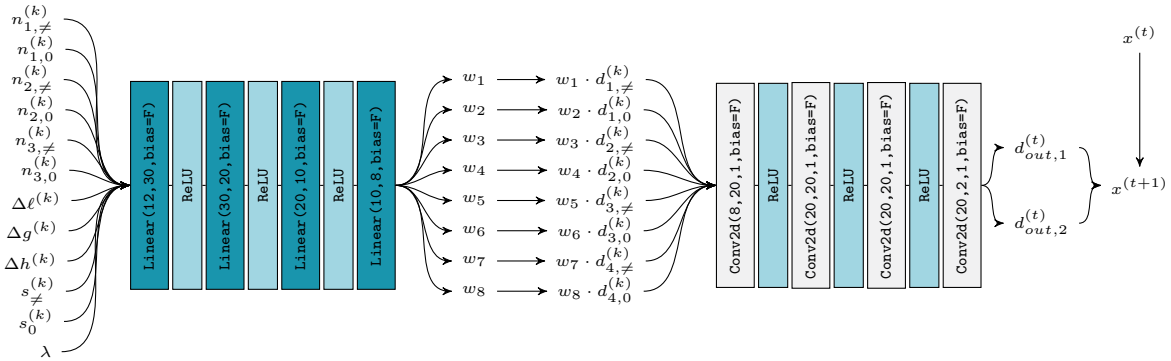

Figure 13: Algorithmic update for the LASSO problem: Based on the given features (split into zero and non-zero parts), the first block computes eight weights, which are used to perform a weighting of the different directions, which then get used in the second block. This second block predicts two directions $d_{out,1}, d_{out,2}$, where $d_{out,1}$ only acts on the non-zero entries, and $d_{out,2}$ acts on the zero entries. These are used in the update $x^{(k+1)} := \mathrm{prox}_{\beta g}\left(x^{(k)} + \left(d^{(k)}_{out,1,\neq} - \nabla\ell(x^{(k)}, p) + \|x^{(k)} - x^{(k-1)}\| \cdot d^{(k)}_{out,2,0}\right)/L\right)$.

reweighting of the directions $d^{(k)}_1, ..., d^{(k)}_4$, which are the normalized gradient, the normalized momentum, the normalized residual $x^{(k)} - p^{(k)}$, and the coordinate-wise product between (normalized) gradient and (normalized) momentum. Then, these reweighted directions get fed into a 1x1-convolutional block, which predicts the two directions $d^{(k)}_{out,1}$ and $d^{(k)}_{out,2}$, which we use to compute the final update with the proximal mapping, given by

$$x^{(k+1)} := \mathrm{prox}_{\beta g}\left(x^{(k)} + \left(d^{(k)}_{out,1,\neq} - \nabla\ell(x^{(k)}, p) + \|x^{(k)} - x^{(k-1)}\| \cdot d^{(k)}_{out,2,0}\right)/L\right) .$$

F.4 Details for the Neural-Network-Training Experiment

This subsection describes the missing details for the neural-network-training experiment.

F.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, $i = 1, ..., N$, 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, $x_{i,j} \overset{iid}{\sim} \mathrm{U}[-2, 2]$, $i = 1, ..., N$, $j = 1, ..., K$. Then, we sample the coefficients $(c_{i,0}, ..., c_{i,5})$ of g_i uniformly in $[-5, 5]$, that is, $c_{i,l} \overset{iid}{\sim} \mathrm{U}[-5, 5]$, $i = 1, ..., N$, $l = 0, ..., 5$. Lastly, we get the values $y_{i,j}$ as:

$$y_{i,j} = g_i(x_{i,j}) + \varepsilon_{i,j} \quad \text{with} \quad \varepsilon_{i,j} \overset{iid}{\sim} \mathcal{N}(0, 1), \quad i = 1, ..., N,\ j = 1, ..., 50\,.$$

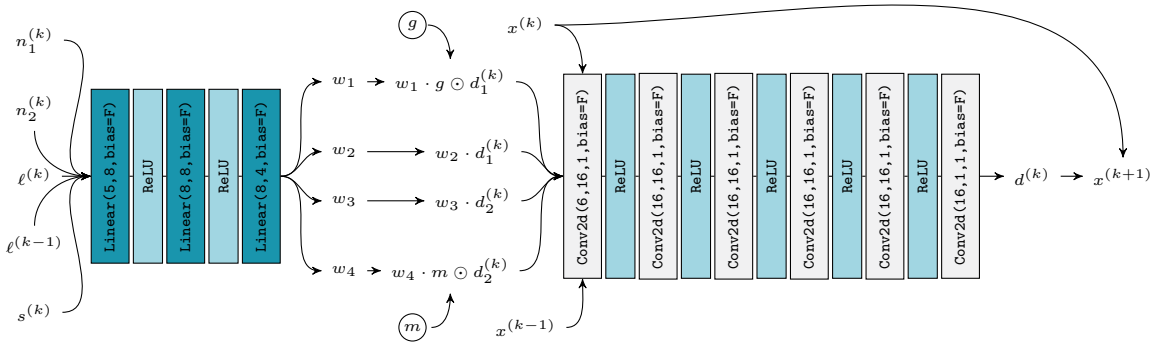

Figure 14: Algorithmic update for training the neural network: Based on the two norms $n_1^{(k)}$ and $n_2^{(k)}$, the scalar product $s^{(k)} := \langle d_1^{(k)}, d_2^{(k)} \rangle$, and the current and previous (logarithmically scaled) loss $\ell^{(k)}$, $\ell^{(k-1)}$, we compute four scalars $w_1, ..., w_4$, which are used for weighting $d_1^{(k)}$, $d_2^{(k)}$, and their corresponding pre-conditioned versions $g \odot d_1^{(k)}$ and $m \odot d_2^{(k)}$. Together with the current and previous point, they get fed (as separate channels) into the first layer of a 1x1-convolutional block, which computes an update direction $d^{(k)}$. Then, we update $x^{(k+1)} := x^{(k)} + d^{(k)}$.

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 $X_i = (x_{i,1}, ..., x_{i,K}) \in \mathbb{R}^K$ and $Y_i = (y_{i,1}, ..., y_{i,K}) \in \mathbb{R}^K$. 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, $\mathcal{P} = \mathbb{R}^{K \times 2}$.

F.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 $\ell(\beta, p_i) := c(\mathtt{N}(\beta, X_i), Y_i) := \frac{1}{K} \sum_{j=1}^{K} (\mathtt{N}_\beta(x_{i,j}) - y_{i,j})^2$.
As model 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 $(x, x^2, ..., x^5)$. Hence, the parameters $\beta \in \mathbb{R}^m$ 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 $m = (5 \cdot 50) + (1 \cdot 50) + 50 + 1 = 351$.

F.4.3 Algorithm

The algorithmic update in Figure 14 consists of two blocks: A weighting block, which computes four weights $w_1, ..., w_4$ based on the norms $n_1^{(k)}, n_2^{(k)}$, the losses $\ell(x^{(k)}, p), \ell(x^{(k-1)}, p)$, and the scalar product $\langle d_1^{(k)}, d_2^{(k)} \rangle$. Each of these gets multiplied with $d_1^{(k)}, d_2^{(k)}$, or the "pre-conditioned" versions, which we compute by pointwise multiplication with the learned vectors g and m. Then, additionally to the $x^{(k)}$ and $x^{(k-1)}$, these weighted directions get fed into an update-block, which computes the final update direction $d^{(k)}$.

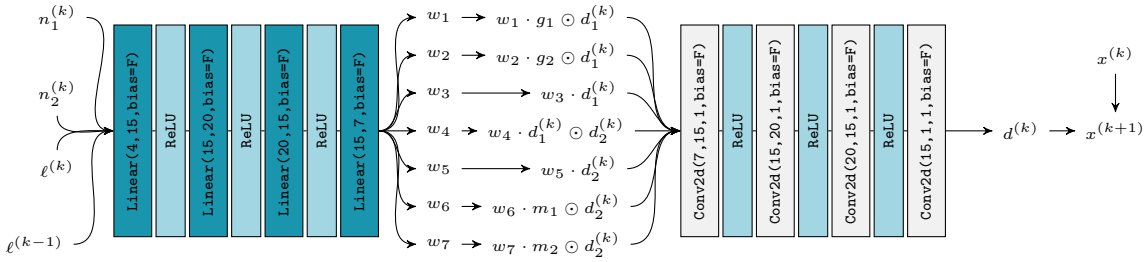

Figure 15: Algorithmic update for the MNIST experiment: Based on the two norms $n_1^{(k)}$ and $n_2^{(k)}$, and the current and previous (logarithmically scaled) loss $\ell^{(k)}$, $\ell^{(k-1)}$, we compute seven scalars $w_1, ..., w_7$, which are used for weighting $d_1^{(k)}$, $d_2^{(k)}$, their corresponding preconditioned versions $g_i \odot d_1^{(k)}$, $m_i \odot d_2^{(k)}$, and their coordinate-wise product $d_1^{(k)} \odot d_2^{(k)}$. Then, they get fed (as separate channels) into the first layer of a 1x1-convolutional block, which computes an update direction $d^{(k)}$. Finally, we update $x^{(k+1)} := x^{(k)} + d^{(k)}$.

Appendix G. 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 $\beta \in \mathbb{R}^m$ of a neural network $\mathtt{N}_\beta$, 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 $\beta \in \mathbb{R}^m$. 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 β has dimension $m = 13090$.

Remark 43 *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, $p \in \mathbb{R}^{250\times(20\times20)\times1} = \mathcal{P}$, and as baseline we use Adam. The architecture of the learnt algorithm is shown in Figure 15 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 weigh the input-directions of the second block. It 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 16 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

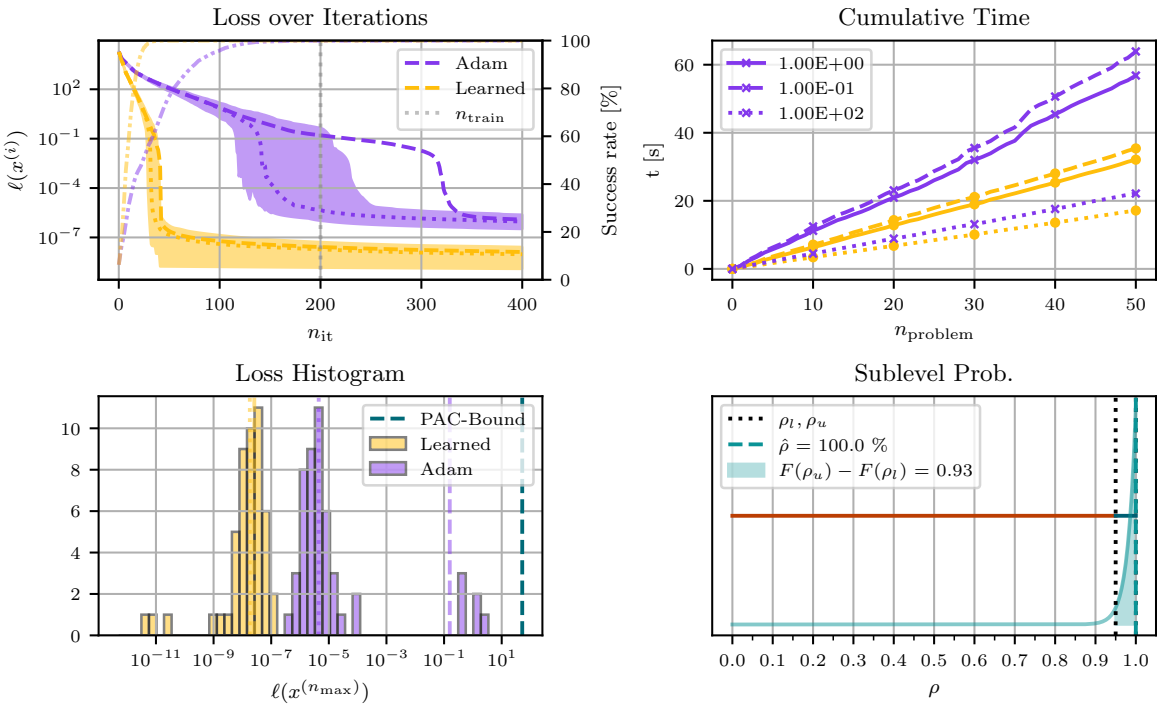

Figure 16: **Upper left:** Dashed lines represent the mean losses, dotted lines represent the median losses, and the shaded regions indicate the 10th to 90th percentile. Further, dashdotted-lines represent the classification-accuracy, which is shown on the right y-axis. Here, Adam is shown in purple and the learned algorithm in yellow. **Upper right:** The different lines show the cumulative computation time of the algorithms to solve all the test problems up to a certain accuracy (in function values) measured by $\ell(\beta^{(i)}, p) - c(X_i, Y_i) < \varepsilon$. However, note that both algorithms are run for maximally $n_{\max} = 5000$ iterations. **Lower left:** Loss histogram (after n_{train} iterations) and PAC-bound. **Lower right:** The teal dashed line shows the point estimate for the sublevel probability, while the teal solid line shows the Beta-posterior. Here, the black dotted lines indicate the constraints ρ_l, ρ_u and show the feasible region as dark teal line.

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.

References

Pierre Alquier. User-friendly introduction to PAC-Bayes bounds. *Foundations and Trends® in Machine Learning*, 17(2):174–303, 2024.

Pierre Alquier and Benjamin Guedj. Simpler PAC-Bayesian bounds for hostile data. *Machine Learning*, 107(5):887–902, 2018.

Pierre Alquier, James Ridgway, and Nicolas Chopin. On the properties of variational approximations of Gibbs posteriors. *Journal of Machine Learning Research*, 17(1):8374–8414, 2016.

Ron Amit, Baruch Epstein, Shay Moran, and Ron Meir. Integral probability metrics PAC-Bayes bounds. *Advances in Neural Information Processing Systems*, 35:3123–3136, 2022.

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Rémi Bardenet, Arnaud Doucet, and Chris Holmes. Towards scaling up markov chain monte carlo: an adaptive subsampling approach. In *International conference on machine learning*, pages 405–413. PMLR, 2014.

Rémi Bardenet, Arnaud Doucet, and Chris Holmes. On Markov chain Monte Carlo methods for tall data. *Journal of Machine Learning Research*, 18(47), 2017.

Ole Barndorff–Nielsen. *Information and exponential families: in statistical theory*. John Wiley & Sons, 2014.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009.

Luc Bégin, Pascal Germain, François Laviolette, and Jean-Francis Roy. Pac-bayesian theory for transductive learning. In *Artificial Intelligence and Statistics*, pages 105–113. PMLR, 2014.

Luc Bégin, Pascal Germain, François Laviolette, and Jean-Francis Roy. PAC-Bayesian bounds based on the Rényi divergence. In *Artificial Intelligence and Statistics*, pages 435–444. PMLR, 2016.

James O. Berger. *Statistical decision theory and Bayesian analysis*. Springer-Verlag, 1985.

Pascal Bianchi, Walid Hachem, and Sholom Schechtman. Convergence of constant step stochastic gradient descent for non-smooth non-convex functions. *Set-Valued and Variational Analysis*, pages 1–31, 2022.

Leon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

Stephane Boucheron, Gabor Lugosi, and Pascal Massart. *Concentration inequalities: A nonasymptotic theory of independence*. Oxford university press, 2013.

Olivier Bousquet and André Elisseeff. Algorithmic stability and generalization performance. *Advances in Neural Information Processing Systems*, 13, 2000.

Olivier Bousquet and André Elisseeff. Stability and generalization. *Journal of Machine Learning Research*, 2:499–526, 2002.

Gregery T Buzzard, Stanley H Chan, Suhas Sreehari, and Charles A Bouman. Plug-and-play unplugged: Optimization-free reconstruction using consensus equilibrium. *SIAM Journal on Imaging Sciences*, 11(3):2001–2020, 2018.

Olivier Catoni. A pac-bayesian approach to adaptive classification. *preprint*, 840(2):6, 2003.

Olivier Catoni. *Statistical learning theory and stochastic optimization: Ecole d'Eté de Probabilités de Saint-Flour, XXXI-2001*, volume 1851. Springer Science & Business Media, 2004.

Olivier Catoni. PAC-Bayesian supervised classification: The thermodynamics of statistical learning. *Lecture Notes-Monograph Series*, 56:i–163, 2007.

Stanley H Chan, Xiran Wang, and Omar A Elgendy. Plug-and-play ADMM for image restoration: Fixed-point convergence and applications. *IEEE Transactions on Computational Imaging*, 3(1):84–98, 2016.

Tianlong Chen, Weiyi Zhang, Zhou Jingyang, Shiyu Chang, Sijia Liu, Lisa Amini, and Zhangyang Wang. Training stronger baselines for learning to optimize. *Advances in Neural Information Processing Systems*, 33:7332–7343, 2020a.

Tianlong Chen, Xiaohan Chen, Wuyang Chen, Howard Heaton, Jialin Liu, Zhangyang Wang, and Wotao Yin. Learning to optimize: A primer and a benchmark. *arXiv preprint arXiv:2103.12828*, 2021.

Xiaohan Chen, Jialin Liu, Zhangyang Wang, and Wotao Yin. Theoretical linear convergence of unfolded ISTA and its practical weights and thresholds. *Advances in Neural Information Processing Systems*, 31, 2018.

Xinshi Chen, Yufei Zhang, Christoph Reisinger, and Le Song. Understanding deep architecture with reasoning layer. *Advances in Neural Information Processing Systems*, 33: 1240–1252, 2020b.

Yutian Chen, Matthew W Hoffman, Sergio Gómez Colmenarejo, Misha Denil, Timothy P Lillicrap, Matt Botvinick, and Nando Freitas. Learning to learn without gradient descent by gradient descent. In *International Conference on Machine Learning*, pages 748–756. PMLR, 2017.

Regev Cohen, Michael Elad, and Peyman Milanfar. Regularization by denoising via fixed-point projection. *SIAM Journal on Imaging Sciences*, 14(3):1374–1406, 2021.

Damek Davis and Dmitriy Drusvyatskiy. Stochastic model-based minimization of weakly convex functions. *SIAM Journal on Optimization*, 29(1):207–239, 2019.

Damek Davis and Dmitriy Drusvyatskiy. Graphical convergence of subgradients in nonconvex optimization and learning. *Mathematics of Operations Research*, 47(1):209–231, 2022.

Alexandre Défossez, Leon Bottou, Francis Bach, and Nicolas Usunier. A simple convergence proof of adam and adagrad. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856.

Monroe D Donsker and SR Srinivasa Varadhan. Asymptotic evaluation of certain Markov process expectations for large time, i. *Communications on Pure and Applied Mathematics*, 28(1):1–47, 1975.

Gintare Karolina Dziugaite and Daniel M. Roy. 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, UAI 2017, Sydney, Australia, August 11-15, 2017*. AUAI Press, 2017.

Gintare Karolina Dziugaite and Daniel M Roy. Data-dependent PAC-Bayes priors via differential privacy. *Advances in neural information processing systems*, 31, 2018.

Gintare Karolina Dziugaite, Kyle Hsu, Waseem Gharbieh, Gabriel Arpino, and Daniel Roy. On the role of data in PAC-Bayes bounds. In *International Conference on Artificial Intelligence and Statistics*, pages 604–612. PMLR, 2021.

Bradley Efron. Defining the curvature of a statistical problem (with applications to second order efficiency). *The Annals of Statistics*, pages 1189–1242, 1975.

Pascal Germain, Alexandre Lacasse, François Laviolette, and Mario Marchand. PAC-Bayesian learning of linear classifiers. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 353–360, 2009.

Karol Gregor and Yann LeCun. Learning fast approximations of sparse coding. In *Proceedings of the 27th international conference on international conference on machine learning*, pages 399–406, 2010.

Benjamin Guedj. A primer on PAC-Bayesian learning. In *Proceedings of the second congress of the French Mathematical Society*, volume 33, 2019.

Maxime Haddouche and Benjamin Guedj. Wasserstein PAC-Bayes learning: Exploiting optimisation guarantees to explain generalisation, 2023.

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

Fredrik Hellström, Giuseppe Durisi, Benjamin Guedj, and Maxim Raginsky. Generalization bounds: Perspectives from information theory and pac-bayes. *arXiv preprint arXiv:2309.04381*, 2023.

Jean Honorio and Tommi Jaakkola. Tight bounds for the expected risk of linear classifiers and PAC-Bayes finite-sample guarantees. In *Artificial Intelligence and Statistics*, pages 384–392. PMLR, 2014.

Timothy Hospedales, Antreas Antoniou, Paul Micaelli, and Amos Storkey. Meta-learning in neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5149–5169, 2021.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

O. Kallenberg. *Foundations of Modern Probability*. Probability theory and stochastic modelling. Springer, 2021. ISBN 9783030618728.

Ali Kavis, Kfir Yehuda Levy, and Volkan Cevher. High probability bounds for a class of nonconvex algorithms with adagrad stepsize. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=dSw0QtRMJkO`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

Erich Kobler, Alexander Effland, Karl Kunisch, and Thomas Pock. Total deep variation: A stable regularizer for inverse problems. *arXiv preprint arXiv:2006.08789*, 2020.

Anoop Korattikara, Yutian Chen, and Max Welling. Austerity in MCMC land: Cutting the Metropolis-Hastings budget. In *International conference on machine learning*, pages 181–189. PMLR, 2014.

John Langford and Rich Caruana. (Not) bounding the true error. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001.

John Langford and John Shawe-Taylor. PAC-Bayes and margins. *Advances in neural information processing systems*, 15, 2002.

Guy Lever, François Laviolette, and John Shawe-Taylor. Tighter PAC-Bayes bounds through distribution-dependent priors. *Theoretical Computer Science*, 473:4–28, 2013.

Ben London. A PAC-Bayesian analysis of randomized learning with application to stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

Kaifeng Lv, Shunhua Jiang, and Jian Li. Learning gradient descent: Better generalization and longer horizons. In *International Conference on Machine Learning*, pages 2247–2255. PMLR, 2017.

Dougal Maclaurin and Ryan P Adams. Firefly Monte Carlo: Exact MCMC with subsets of data. In *30th Conference on Uncertainty in Artificial Intelligence, UAI 2014*, pages 543–552. AUAI Press, 2014.

David McAllester. Simplified PAC-Bayesian margin bounds. In *Learning theory and Kernel machines*, pages 203–215. Springer, 2003a.

David McAllester. PAC-Bayesian stochastic model selection. *Machine Learning*, 51(1):5–21, 2003b.

Luke Metz, Niru Maheswaranathan, Jeremy Nixon, Daniel Freeman, and Jascha Sohl-Dickstein. Understanding and correcting pathologies in the training of learned optimizers. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4556–4565. PMLR, 2019.

Michael Moeller, Thomas Mollenhoff, and Daniel Cremers. Controlling neural networks via energy dissipation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3256–3265, 2019.

Yurii Nesterov. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Proceedings of the USSR Academy of Sciences*, 269:543–547, 1983.

Yurii Nesterov. *Lectures on convex optimization*, volume 137. Springer, 2018.

Yuki Ohnishi and Jean Honorio. Novel change of measure inequalities with applications to PAC-Bayesian bounds and Monte Carlo estimation. In *International Conference on Artificial Intelligence and Statistics*, pages 1711–1719. PMLR, 2021.

Emilio Parrado-Hernández, Amiran Ambroladze, John Shawe-Taylor, and Shiliang Sun. PAC-Bayes bounds with data dependent priors. *Journal of Machine Learning Research*, 13(1):3507–3531, 2012.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

María Pérez-Ortiz, Omar Rivasplata, John Shawe-Taylor, and Csaba Szepesvári. Tighter risk certificates for neural networks. *Journal of Machine Learning Research*, 22(227): 1–40, 2021.

Boris T Polyak. Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Matias Quiroz, Robert Kohn, Mattias Villani, and Minh-Ngoc Tran. Speeding up MCMC by efficient data subsampling. *Journal of the American Statistical Association*, 2018.

Omar Rivasplata, Ilja Kuzborskij, Csaba Szepesvári, and John Shawe-Taylor. PAC-Bayes analysis beyond the usual bounds. *Advances in Neural Information Processing Systems*, 33:16833–16845, 2020.

C.P. Robert and G. Casella. *Monte Carlo statistical methods*. Springer New York, 2004.

Ernest Ryu, Jialin Liu, Sicheng Wang, Xiaohan Chen, Zhangyang Wang, and Wotao Yin. Plug-and-play methods provably converge with properly trained denoisers. In *International Conference on Machine Learning*, pages 5546–5557. PMLR, 2019.

Matthias Seeger. PAC-Bayesian generalisation error bounds for Gaussian process classification. *Journal of Machine Learning Research*, 3:233–269, 2002.

Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Stochastic convex optimization. In *COLT*, volume 2, page 5, 2009.

Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Learnability, stability and uniform convergence. *Journal of Machine Learning Research*, 11:2635–2670, 2010.

Suhas Sreehari, S Venkat Venkatakrishnan, Brendt Wohlberg, Gregery T Buzzard, Lawrence F Drummy, Jeffrey P Simmons, and Charles A Bouman. Plug-and-play priors for bright field electron tomography and sparse interpolation. *IEEE Transactions on Computational Imaging*, 2(4):408–423, 2016.

Michael Sucker and Peter Ochs. PAC-Bayesian learning of optimization algorithms. In *International Conference on Artificial Intelligence and Statistics*, pages 8145–8164. PMLR, 2023.

Yu Sun, Brendt Wohlberg, and Ulugbek S Kamilov. An online plug-and-play algorithm for regularized image reconstruction. *IEEE Transactions on Computational Imaging*, 5(3): 395–408, 2019.

Afonso M Teodoro, José M Bioucas-Dias, and Mário AT Figueiredo. Scene-adapted plug-and-play algorithm with convergence guarantees. In *2017 IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2017.

Matthieu Terris, Audrey Repetti, Jean-Christophe Pesquet, and Yves Wiaux. Enhanced convergent PNP algorithms for image restoration. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 1684–1688. IEEE, 2021.

Niklas Thiemann, Christian Igel, Olivier Wintenberger, and Yevgeny Seldin. A strongly quasiconvex PAC-Bayesian bound. In *International Conference on Algorithmic Learning Theory*, pages 466–492. PMLR, 2017.

Robert Tibshirani. Regression shrinkage and selection via the LASSO. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

Tom Tirer and Raja Giryes. Image restoration by iterative denoising and backward projections. *IEEE Transactions on Image Processing*, 28(3):1220–1234, 2018.

Ricardo Vilalta and Youssef Drissi. A perspective view and survey of meta-learning. *Artificial intelligence review*, 18:77–95, 2002.

Cédric Villani et al. *Optimal transport: old and new*, volume 338. Springer, 2009.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML)*, pages 681–688, 2011.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pages 3751–3760. PMLR, 2017.

Hermann Witting. *Mathematische Statistik I: Parametrische Verfahren bei festem Stichprobenumfang*. Springer-Verlag, 2013.

Bo Xin, Yizhou Wang, Wen Gao, David Wipf, and Baoyuan Wang. Maximal sparsity with deep networks? *Advances in Neural Information Processing Systems*, 29, 2016.

Quanming Yao, Mengshuo Wang, Yuqiang Chen, Wenyuan Dai, Yu-Feng Li, Wei-Wei Tu, Qiang Yang, and Yang Yu. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306*, 2018.