Chapter 1
Hyperparameter Optimization

Matthias Feurer and Frank Hutter

Abstract Recent interest in complex and computationally expensive machine learning models with many hyperparameters, such as automated machine learning (AutoML) frameworks and deep neural networks, has resulted in a resurgence of research on hyperparameter optimization (HPO). In this chapter, we give an overview of the most prominent approaches for HPO. We first discuss blackbox function optimization methods based on model-free methods and Bayesian optimization. Since the high computational demand of many modern machine learning applications renders pure blackbox optimization extremely costly, we next focus on modern multi-fidelity methods that use (much) cheaper variants of the blackbox function to approximately assess the quality of hyperparameter settings. Lastly, we point to open problems and future research directions.

1.1 Introduction

Every machine learning system has hyperparameters, and the most basic task in automated machine learning (AutoML) is to automatically set these hyperparameters to optimize performance. Especially recent deep neural networks crucially depend on a wide range of hyperparameter choices about the neural network's architecture, regularization, and optimization. Automated hyperparameter optimization (HPO) has several important use cases; it can

- reduce the human effort necessary for applying machine learning. This is particularly important in the context of AutoML.

M. Feurer (✉)
Department of Computer Science, University of Freiburg, Freiburg, Baden-Württemberg, Germany
e-mail: feurerm@informatik.uni-freiburg.de

F. Hutter
Department of Computer Science, University of Freiburg, Freiburg, Germany

F. Hutter et al. (eds.), *Automated Machine Learning*, The Springer Series on Challenges in Machine Learning, https://doi.org/10.1007/978-3-030-05318-5_1

- improve the performance of machine learning algorithms (by tailoring them to the problem at hand); this has led to new state-of-the-art performances for important machine learning benchmarks in several studies (e.g. [105, 140]).
- improve the reproducibility and fairness of scientific studies. Automated HPO is clearly more reproducible than manual search. It facilitates fair comparisons since different methods can only be compared fairly if they all receive the same level of tuning for the problem at hand [14, 133].

The problem of HPO has a long history, dating back to the 1990s (e.g., [77, 82, 107, 126]), and it was also established early that different hyperparameter configurations tend to work best for different datasets [82]. In contrast, it is a rather new insight that HPO can be used to adapt general-purpose pipelines to specific application domains [30]. Nowadays, it is also widely acknowledged that tuned hyperparameters improve over the default setting provided by common machine learning libraries [100, 116, 130, 149].

Because of the increased usage of machine learning in companies, HPO is also of substantial commercial interest and plays an ever larger role there, be it in company-internal tools [45], as part of machine learning cloud services [6, 89], or as a service by itself [137].

HPO faces several challenges which make it a hard problem in practice:

- Function evaluations can be extremely expensive for large models (e.g., in deep learning), complex machine learning pipelines, or large datesets.
- The configuration space is often complex (comprising a mix of continuous, categorical and conditional hyperparameters) and high-dimensional. Furthermore, it is not always clear which of an algorithm's hyperparameters need to be optimized, and in which ranges.
- We usually don't have access to a gradient of the loss function with respect to the hyperparameters. Furthermore, other properties of the target function often used in classical optimization do not typically apply, such as convexity and smoothness.
- One cannot directly optimize for generalization performance as training datasets are of limited size.

We refer the interested reader to other reviews of HPO for further discussions on this topic [64, 94].

This chapter is structured as follows. First, we define the HPO problem formally and discuss its variants (Sect. 1.2). Then, we discuss blackbox optimization algorithms for solving HPO (Sect. 1.3). Next, we focus on modern multi-fidelity methods that enable the use of HPO even for very expensive models, by exploiting approximate performance measures that are cheaper than full model evaluations (Sect. 1.4). We then provide an overview of the most important hyperparameter optimization systems and applications to AutoML (Sect. 1.5) and end the chapter with a discussion of open problems (Sect. 1.6).

1.2 Problem Statement

Let $\mathcal{A}$ denote a machine learning algorithm with N hyperparameters. We denote the domain of the n-th hyperparameter by Λ_n and the overall *hyperparameter configuration space* as $\mathbf{\Lambda} = \Lambda_1 \times \Lambda_2 \times \ldots \Lambda_N$. A vector of hyperparameters is denoted by $\boldsymbol{\lambda} \in \mathbf{\Lambda}$, and $\mathcal{A}$ with its hyperparameters instantiated to $\boldsymbol{\lambda}$ is denoted by $\mathcal{A}_{\boldsymbol{\lambda}}$.

The domain of a hyperparameter can be real-valued (e.g., learning rate), integer-valued (e.g., number of layers), binary (e.g., whether to use early stopping or not), or categorical (e.g., choice of optimizer). For integer and real-valued hyperparameters, the domains are mostly bounded for practical reasons, with only a few exceptions [12, 113, 136].

Furthermore, the configuration space can contain *conditionality*, i.e., a hyperparameter may only be relevant if another hyperparameter (or some combination of hyperparameters) takes on a certain value. Conditional spaces take the form of directed acyclic graphs. Such conditional spaces occur, e.g., in the automated tuning of machine learning pipelines, where the choice between different preprocessing and machine learning algorithms is modeled as a categorical hyperparameter, a problem known as *Full Model Selection* (FMS) or *Combined Algorithm Selection and Hyperparameter optimization problem* (CASH) [30, 34, 83, 149]. They also occur when optimizing the architecture of a neural network: e.g., the number of layers can be an integer hyperparameter and the per-layer hyperparameters of layer i are only active if the network depth is at least i [12, 14, 33].

Given a data set $\mathcal{D}$, our goal is to find

$$\boldsymbol{\lambda}^* = \underset{\boldsymbol{\lambda} \in \mathbf{\Lambda}}{\operatorname{argmin}} \, \mathbb{E}_{(D_{train}, D_{valid}) \sim \mathcal{D}} \mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid}), \tag{1.1}$$

where $\mathbf{V}(\mathcal{L}, \mathcal{A}_{\boldsymbol{\lambda}}, D_{train}, D_{valid})$ measures the loss of a model generated by algorithm $\mathcal{A}$ with hyperparameters $\boldsymbol{\lambda}$ on training data D_{train} and evaluated on validation data D_{valid}. In practice, we only have access to finite data $D \sim \mathcal{D}$ and thus need to approximate the expectation in Eq. 1.1.

Popular choices for the validation protocol $\mathbf{V}(\cdot, \cdot, \cdot, \cdot)$ are the holdout and cross-validation error for a user-given loss function (such as misclassification rate); see Bischl et al. [16] for an overview of validation protocols. Several strategies for reducing the evaluation time have been proposed: It is possible to only test machine learning algorithms on a subset of folds [149], only on a subset of data [78, 102, 147], or for a small amount of iterations; we will discuss some of these strategies in more detail in Sect. 1.4. Recent work on multi-task [147] and multi-source [121] optimization introduced further cheap, auxiliary tasks, which can be queried instead of Eq. 1.1. These can provide cheap information to help HPO, but do not necessarily train a machine learning model on the dataset of interest and therefore do not yield a usable model as a side product.

1.2.1 Alternatives to Optimization: Ensembling and Marginalization

Solving Eq. 1.1 with one of the techniques described in the rest of this chapter usually requires fitting the machine learning algorithm $\mathcal{A}$ with multiple hyperparameter vectors $\boldsymbol{\lambda}_t$. Instead of using the argmin-operator over these, it is possible to either construct an ensemble (which aims to minimize the loss for a given validation protocol) or to integrate out all the hyperparameters (if the model under consideration is a probabilistic model). We refer to Guyon et al. [50] and the references therein for a comparison of frequentist and Bayesian model selection.

Only choosing a single hyperparameter configuration can be wasteful when many good configurations have been identified by HPO, and combining them in an ensemble can improve performance [109]. This is particularly useful in AutoML systems with a large configuration space (e.g., in *FMS* or *CASH*), where good configurations can be very diverse, which increases the potential gains from ensembling [4, 19, 31, 34]. To further improve performance, *Automatic Frankensteining* [155] uses HPO to train a stacking model [156] on the outputs of the models found with HPO; the 2nd level models are then combined using a traditional ensembling strategy.

The methods discussed so far applied ensembling after the HPO procedure. While they improve performance in practice, the base models are not optimized for ensembling. It is, however, also possible to directly optimize for models which would maximally improve an existing ensemble [97].

Finally, when dealing with Bayesian models it is often possible to integrate out the hyperparameters of the machine learning algorithm, for example using *evidence maximization* [98], *Bayesian model averaging* [56], *slice sampling* [111] or *empirical Bayes* [103].

1.2.2 Optimizing for Multiple Objectives

In practical applications it is often necessary to trade off two or more objectives, such as the performance of a model and resource consumption [65] (see also Chap. 3) or multiple loss functions [57]. Potential solutions can be obtained in two ways.

First, if a limit on a secondary performance measure is known (such as the maximal memory consumption), the problem can be formulated as a constrained optimization problem. We will discuss constraint handling in Bayesian optimization in Sect. 1.3.2.4.

Second, and more generally, one can apply multi-objective optimization to search for the Pareto front, a set of configurations which are optimal tradeoffs between the objectives in the sense that, for each configuration on the Pareto front, there is no other configuration which performs better for at least one and at least as well for all other objectives. The user can then choose a configuration from the Pareto front. We refer the interested reader to further literature on this topic [53, 57, 65, 134].

1.3 Blackbox Hyperparameter Optimization

In general, every blackbox optimization method can be applied to HPO. Due to the non-convex nature of the problem, global optimization algorithms are usually preferred, but some locality in the optimization process is useful in order to make progress within the few function evaluations that are usually available. We first discuss model-free blackbox HPO methods and then describe blackbox Bayesian optimization methods.

1.3.1 Model-Free Blackbox Optimization Methods

Grid search is the most basic HPO method, also known as full factorial design [110]. The user specifies a finite set of values for each hyperparameter, and grid search evaluates the Cartesian product of these sets. This suffers from the curse of dimensionality since the required number of function evaluations grows exponentially with the dimensionality of the configuration space. An additional problem of grid search is that increasing the resolution of discretization substantially increases the required number of function evaluations.

A simple alternative to grid search is random search [13].[1] As the name suggests, random search samples configurations at random until a certain budget for the search is exhausted. This works better than grid search when some hyperparameters are much more important than others (a property that holds in many cases [13, 61]). Intuitively, when run with a fixed budget of B function evaluations, the number of different values grid search can afford to evaluate for each of the N hyperparameters is only $B^{1/N}$, whereas random search will explore B different values for each; see Fig. 1.1 for an illustration.

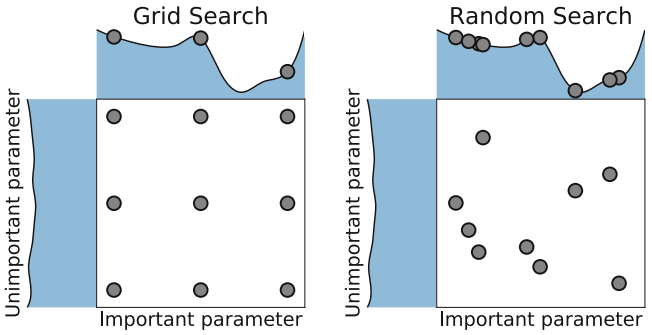

Fig. 1.1 Comparison of grid search and random search for minimizing a function with one important and one unimportant parameter. This figure is based on the illustration in Fig. 1 of Bergstra and Bengio [13]

[1] In some disciplines this is also known as *pure* random search [158].

Further advantages over grid search include easier parallelization (since workers do not need to communicate with each other and failing workers do not leave holes in the design) and flexible resource allocation (since one can add an arbitrary number of random points to a random search design to still yield a random search design; the equivalent does not hold for grid search).

Random search is a useful baseline because it makes no assumptions on the machine learning algorithm being optimized, and, given enough resources, will, in expectation, achieves performance arbitrarily close to the optimum. Interleaving random search with more complex optimization strategies therefore allows to guarantee a minimal rate of convergence and also adds exploration that can improve model-based search [3, 59]. Random search is also a useful method for initializing the search process, as it explores the entire configuration space and thus often finds settings with reasonable performance. However, it is no silver bullet and often takes far longer than guided search methods to identify one of the best performing hyperparameter configurations: e.g., when sampling without replacement from a configuration space with N Boolean hyperparameters with a good and a bad setting each and no interaction effects, it will require an expected 2^{N-1} function evaluations to find the optimum, whereas a guided search could find the optimum in $N + 1$ function evaluations as follows: starting from an arbitrary configuration, loop over the hyperparameters and change one at a time, keeping the resulting configuration if performance improves and reverting the change if it doesn't. Accordingly, the guided search methods we discuss in the following sections usually outperform random search [12, 14, 33, 90, 153].

Population-based methods, such as *genetic algorithms*, *evolutionary algorithms*, *evolutionary strategies*, and *particle swarm optimization* are optimization algorithms that maintain a population, i.e., a set of configurations, and improve this population by applying local perturbations (so-called mutations) and combinations of different members (so-called crossover) to obtain a new generation of better configurations. These methods are conceptually simple, can handle different data types, and are embarrassingly parallel [91] since a population of N members can be evaluated in parallel on N machines.

One of the best known population-based methods is the covariance matrix adaption evolutionary strategy (CMA-ES [51]); this simple evolutionary strategy samples configurations from a multivariate Gaussian whose mean and covariance are updated in each generation based on the success of the population's individuals. CMA-ES is one of the most competitive blackbox optimization algorithms, regularly dominating the *Black-Box Optimization Benchmarking* (BBOB) challenge [11].

For further details on population-based methods, we refer to [28, 138]; we discuss applications to hyperparameter optimization in Sect. 1.5, applications to neural architecture search in Chap. 3, and genetic programming for AutoML pipelines in Chap. 8.

1.3.2 Bayesian Optimization

Bayesian optimization is a state-of-the-art optimization framework for the global optimization of expensive blackbox functions, which recently gained traction in HPO by obtaining new state-of-the-art results in tuning deep neural networks for image classification [140, 141], speech recognition [22] and neural language modeling [105], and by demonstrating wide applicability to different problem settings. For an in-depth introduction to Bayesian optimization, we refer to the excellent tutorials by Shahriari et al. [135] and Brochu et al. [18].

In this section we first give a brief introduction to Bayesian optimization, present alternative surrogate models used in it, describe extensions to conditional and constrained configuration spaces, and then discuss several important applications to hyperparameter optimization.

Many recent advances in Bayesian optimization do not treat HPO as a blackbox any more, for example multi-fidelity HPO (see Sect. 1.4), Bayesian optimization with meta-learning (see Chap. 2), and Bayesian optimization taking the pipeline structure into account [159, 160]. Furthermore, many recent developments in Bayesian optimization do not directly target HPO, but can often be readily applied to HPO, such as new acquisition functions, new models and kernels, and new parallelization schemes.

1.3.2.1 Bayesian Optimization in a Nutshell

Bayesian optimization is an iterative algorithm with two key ingredients: a probabilistic surrogate model and an acquisition function to decide which point to evaluate next. In each iteration, the surrogate model is fitted to all observations of the target function made so far. Then the acquisition function, which uses the predictive distribution of the probabilistic model, determines the utility of different candidate points, trading off exploration and exploitation. Compared to evaluating the expensive blackbox function, the acquisition function is cheap to compute and can therefore be thoroughly optimized.

Although many acquisition functions exist, the *expected improvement* (EI) [72]:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = \mathbb{E}[\max(f_{min} - y, 0)] \tag{1.2}$$

is a common choice since it can be computed in closed form if the model prediction y at configuration $\boldsymbol{\lambda}$ follows a normal distribution:

$$\mathbb{E}[\mathbb{I}(\boldsymbol{\lambda})] = (f_{min} - \mu(\boldsymbol{\lambda}))\,\Phi\left(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}\right) + \sigma\phi\left(\frac{f_{min} - \mu(\boldsymbol{\lambda})}{\sigma}\right), \tag{1.3}$$

where $\phi(\cdot)$ and $\Phi(\cdot)$ are the standard normal density and standard normal distribution function, and f_{min} is the best observed value so far.

Fig. 1.2 illustrates Bayesian optimization optimizing a toy function.

1.3.2.2 Surrogate Models

Traditionally, Bayesian optimization employs Gaussian processes [124] to model the target function because of their expressiveness, smooth and well-calibrated

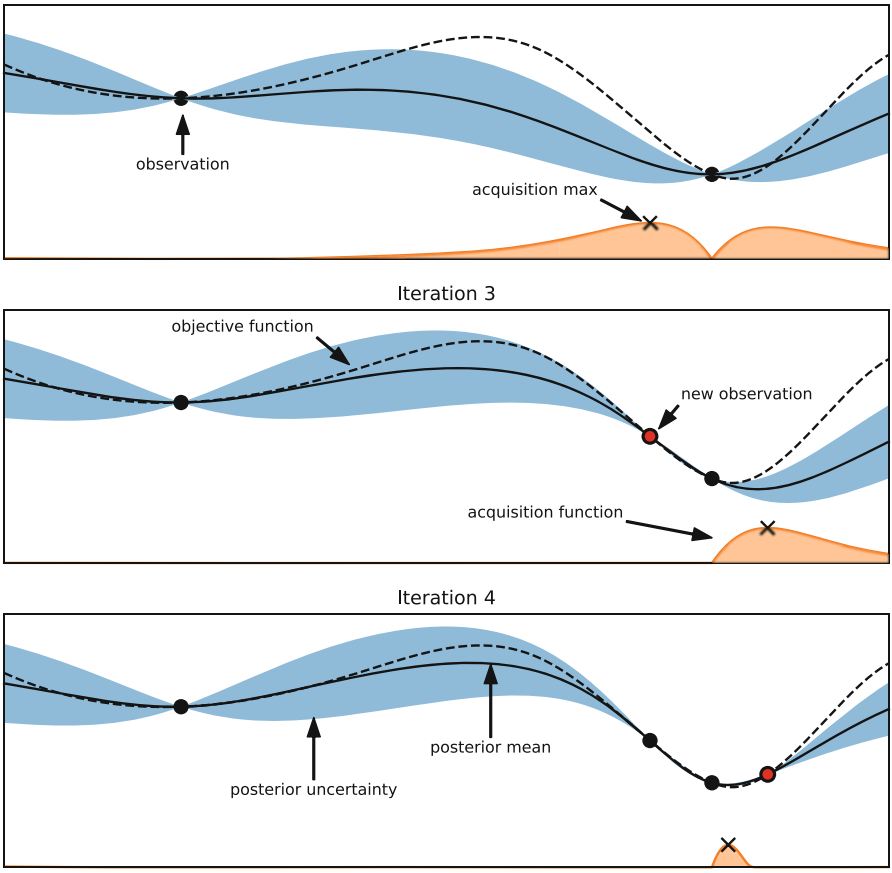

Fig. 1.2 Illustration of Bayesian optimization on a 1-d function. Our goal is to minimize the dashed line using a Gaussian process surrogate (predictions shown as black line, with blue tube representing the uncertainty) by maximizing the acquisition function represented by the lower orange curve. (Top) The acquisition value is low around observations, and the highest acquisition value is at a point where the predicted function value is low and the predictive uncertainty is relatively high. (Middle) While there is still a lot of variance to the left of the new observation, the predicted mean to the right is much lower and the next observation is conducted there. (Bottom) Although there is almost no uncertainty left around the location of the true maximum, the next evaluation is done there due to its expected improvement over the best point so far

uncertainty estimates and closed-form computability of the predictive distribution. A Gaussian process $\mathcal{G}\left(m(\boldsymbol{\lambda}), k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')\right)$ is fully specified by a mean $m(\boldsymbol{\lambda})$ and a covariance function $k(\boldsymbol{\lambda}, \boldsymbol{\lambda}')$, although the mean function is usually assumed to be constant in Bayesian optimization. Mean and variance predictions $\mu(\cdot)$ and $\sigma^2(\cdot)$ for the noise-free case can be obtained by:

$$\mu(\boldsymbol{\lambda}) = \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{y}, \sigma^2(\boldsymbol{\lambda}) = k(\boldsymbol{\lambda}, \boldsymbol{\lambda}) - \mathbf{k}_*^T \mathbf{K}^{-1} \mathbf{k}_*, \tag{1.4}$$

where $\mathbf{k}_*$ denotes the vector of covariances between $\boldsymbol{\lambda}$ and all previous observations, $\mathbf{K}$ is the covariance matrix of all previously evaluated configurations and $\mathbf{y}$ are the observed function values. The quality of the Gaussian process depends solely on the covariance function. A common choice is the Mátern 5/2 kernel, with its hyperparameters integrated out by Markov Chain Monte Carlo [140].

One downside of standard Gaussian processes is that they scale cubically in the number of data points, limiting their applicability when one can afford many function evaluations (e.g., with many parallel workers, or when function evaluations are cheap due to the use of lower fidelities). This cubic scaling can be avoided by scalable Gaussian process approximations, such as sparse Gaussian processes. These approximate the full Gaussian process by using only a subset of the original dataset as *inducing points* to build the kernel matrix $\mathbf{K}$. While they allowed Bayesian optimization with GPs to scale to tens of thousands of datapoints for optimizing the parameters of a randomized SAT solver [62], there are criticism about the calibration of their uncertainty estimates and their applicability to standard HPO has not been tested [104, 154].

Another downside of Gaussian processes with standard kernels is their poor scalability to high dimensions. As a result, many extensions have been proposed to efficiently handle intrinsic properties of configuration spaces with large number of hyperparameters, such as the use of random embeddings [153], using Gaussian processes on partitions of the configuration space [154], cylindric kernels [114], and additive kernels [40, 75].

Since some other machine learning models are more scalable and flexible than Gaussian processes, there is also a large body of research on adapting these models to Bayesian optimization. Firstly, (deep) neural networks are a very flexible and scalable models. The simplest way to apply them to Bayesian optimization is as a feature extractor to preprocess inputs and then use the outputs of the final hidden layer as basis functions for Bayesian linear regression [141]. A more complex, fully Bayesian treatment of the network weights, is also possible by using a Bayesian neural network trained with stochastic gradient Hamiltonian Monte Carlo [144]. Neural networks tend to be faster than Gaussian processes for Bayesian optimization after ~250 function evaluations, which also allows for large-scale parallelism. The flexibility of deep learning can also enable Bayesian optimization on more complex tasks. For example, a variational auto-encoder can be used to embed complex inputs (such as the structured configurations of the automated statistician, see Chap. 9) into a real-valued vector such that a regular Gaussian process can handle it [92]. For multi-source Bayesian optimization, a neural network architecture built on

factorization machines [125] can include information on previous tasks [131] and has also been extended to tackle the CASH problem [132].

Another alternative model for Bayesian optimization are random forests [59]. While GPs perform better than random forests on small, numerical configuration spaces [29], random forests natively handle larger, categorical and conditional configuration spaces where standard GPs do not work well [29, 70, 90]. Furthermore, the computational complexity of random forests scales far better to many data points: while the computational complexity of fitting and predicting variances with GPs for n data points scales as $O(n^3)$ and $O(n^2)$, respectively, for random forests, the scaling in n is only $O(n \log n)$ and $O(\log n)$, respectively. Due to these advantages, the SMAC framework for Bayesian optimization with random forests [59] enabled the prominent AutoML frameworks Auto-WEKA [149] and Auto-sklearn [34] (which are described in Chaps. 4 and 6).

Instead of modeling the probability $p(\mathbf{y}|\boldsymbol{\lambda})$ of observations $\mathbf{y}$ given the configurations $\boldsymbol{\lambda}$, the *Tree Parzen Estimator* (TPE [12, 14]) models density functions $p(\boldsymbol{\lambda}|\mathbf{y} < \alpha)$ and $p(\boldsymbol{\lambda}|\mathbf{y} \geq \alpha)$. Given a percentile α (usually set to 15%), the observations are divided in good observations and bad observations and simple 1-d Parzen windows are used to model the two distributions. The ratio $\frac{p(\boldsymbol{\lambda}|\mathbf{y}<\alpha)}{p(\boldsymbol{\lambda}|\mathbf{y}\geq\alpha)}$ is related to the expected improvement acquisition function and is used to propose new hyperparameter configurations. TPE uses a tree of Parzen estimators for conditional hyperparameters and demonstrated good performance on such structured HPO tasks [12, 14, 29, 33, 143, 149, 160], is conceptually simple, and parallelizes naturally [91]. It is also the workhorse behind the AutoML framework Hyperopt-sklearn [83] (which is described in Chap. 5).

Finally, we note that there are also surrogate-based approaches which do not follow the Bayesian optimization paradigm: Hord [67] uses a deterministic RBF surrogate, and Harmonica [52] uses a compressed sensing technique, both to tune the hyperparameters of deep neural networks.

1.3.2.3 Configuration Space Description

Bayesian optimization was originally designed to optimize box-constrained, real-valued functions. However, for many machine learning hyperparameters, such as the learning rate in neural networks or regularization in support vector machines, it is common to optimize the exponent of an exponential term to describe that changing it, e.g., from 0.001 to 0.01 is expected to have a similarly high impact as changing it from 0.1 to 1. A technique known as *input warping* [142] allows to automatically learn such transformations during the optimization process by replacing each input dimension with the two parameters of a Beta distribution and optimizing these.

One obvious limitation of the box-constraints is that the user needs to define these upfront. To avoid this, it is possible to dynamically expand the configuration space [113, 136]. Alternatively, the estimation-of-distribution-style algorithm TPE [12] is able to deal with infinite spaces on which a (typically Gaussian) prior is placed.

Integers and categorical hyperparameters require special treatment but can be integrated fairly easily into regular Bayesian optimization by small adaptations of the kernel and the optimization procedure (see Sect. 12.1.2 of [58], as well as [42]). Other models, such as factorization machines and random forests, can also naturally handle these data types.

Conditional hyperparameters are still an active area of research (see Chaps. 5 and 6 for depictions of conditional configuration spaces in recent AutoML systems). They can be handled natively by tree-based methods, such as random forests [59] and tree Parzen estimators (TPE) [12], but due to the numerous advantages of Gaussian processes over other models, multiple kernels for structured configuration spaces have also been proposed [4, 12, 63, 70, 92, 96, 146].

1.3.2.4 Constrained Bayesian Optimization

In realistic scenarios it is often necessary to satisfy constraints, such as memory consumption [139, 149], training time [149], prediction time [41, 43], accuracy of a compressed model [41], energy usage [43] or simply to not fail during the training procedure [43].

Constraints can be *hidden* in that only a binary observation (success or failure) is available [88]. Typical examples in AutoML are memory and time constraints to allow training of the algorithms in a shared computing system, and to make sure that a single slow algorithm configuration does not use all the time available for HPO [34, 149] (see also Chaps. 4 and 6).

Constraints can also merely be *unknown*, meaning that we can observe and model an auxiliary constraint function, but only know about a constraint violation after evaluating the target function [46]. An example of this is the prediction time of a support vector machine, which can only be obtained by training it as it depends on the number of support vectors selected during training.

The simplest approach to model violated constraints is to define a penalty value (at least as bad as the worst possible observable loss value) and use it as the observation for failed runs [34, 45, 59, 149]. More advanced approaches model the probability of violating one or more constraints and actively search for configurations with low loss values that are unlikely to violate any of the given constraints [41, 43, 46, 88].

Bayesian optimization frameworks using information theoretic acquisition functions allow decoupling the evaluation of the target function and the constraints to dynamically choose which of them to evaluate next [43, 55]. This becomes advantageous when evaluating the function of interest and the constraints require vastly different amounts of time, such as evaluating a deep neural network's performance and memory consumption [43].

1.4 Multi-fidelity Optimization

Increasing dataset sizes and increasingly complex models are a major hurdle in HPO since they make blackbox performance evaluation more expensive. Training a single hyperparameter configuration on large datasets can nowadays easily exceed several hours and take up to several days [85].

A common technique to speed up manual tuning is therefore to probe an algorithm/hyperparameter configuration on a small subset of the data, by training it only for a few iterations, by running it on a subset of features, by only using one or a few of the cross-validation folds, or by using down-sampled images in computer vision. Multi-fidelity methods cast such manual heuristics into formal algorithms, using so-called low fidelity approximations of the actual loss function to minimize. These approximations introduce a tradeoff between optimization performance and runtime, but in practice, the obtained speedups often outweigh the approximation error.

First, we review methods which model an algorithm's learning curve during training and can stop the training procedure if adding further resources is predicted to not help. Second, we discuss simple selection methods which only choose one of a finite set of given algorithms/hyperparameter configurations. Third, we discuss multi-fidelity methods which can actively decide which fidelity will provide most information about finding the optimal hyperparameters. We also refer to Chap. 2 (which discusses how multi-fidelity methods can be used across datasets) and Chap. 3 (which describes low-fidelity approximations for neural architecture search).

1.4.1 Learning Curve-Based Prediction for Early Stopping

We start this section on multi-fidelity methods in HPO with methods that evaluate and model learning curves during HPO [82, 123] and then decide whether to add further resources or stop the training procedure for a given hyperparameter configuration. Examples of learning curves are the performance of the same configuration trained on increasing dataset subsets, or the performance of an iterative algorithm measured for each iteration (or every i-th iteration if the calculation of the performance is expensive).

Learning curve extrapolation is used in the context of *predictive termination* [26], where a learning curve model is used to extrapolate a partially observed learning curve for a configuration, and the training process is stopped if the configuration is predicted to not reach the performance of the best model trained so far in the optimization process. Each learning curve is modeled as a weighted combination of 11 parametric functions from various scientific areas. These functions' parameters and their weights are sampled via Markov chain Monte Carlo to minimize the loss of fitting the partially observed learning curve. This yields a predictive distribution,

which allows to stop training based on the probability of not beating the best known model. When combined with Bayesian optimization, the predictive termination criterion enabled lower error rates than off-the-shelve blackbox Bayesian optimization for optimizing neural networks. On average, the method sped up the optimization by a factor of two and was able to find a (then) state-of-the-art neural network for CIFAR-10 (without data augmentation) [26].

While the method above is limited by not sharing information across different hyperparameter configurations, this can be achieved by using the basis functions as the output layer of a Bayesian neural network [80]. The parameters and weights of the basis functions, and thus the full learning curve, can thereby be predicted for arbitrary hyperparameter configurations. Alternatively, it is possible to use previous learning curves as basis function extrapolators [21]. While the experimental results are inconclusive on whether the proposed method is superior to pre-specified parametric functions, not having to manually define them is a clear advantage.

Freeze-Thaw Bayesian optimization [148] is a full integration of learning curves into the modeling and selection process of Bayesian optimization. Instead of terminating a configuration, the machine learning models are trained iteratively for a few iterations and then *frozen*. Bayesian optimization can then decide to *thaw* one of the frozen models, which means to continue training it. Alternatively, the method can also decide to start a new configuration. Freeze-Thaw models the performance of a converged algorithm with a regular Gaussian process and introduces a special covariance function corresponding to exponentially decaying functions to model the learning curves with per-learning curve Gaussian processes.

1.4.2 Bandit-Based Algorithm Selection Methods

In this section, we describe methods that try to determine the best algorithm out of a given finite set of algorithms based on low-fidelity approximations of their performance; towards its end, we also discuss potential combinations with adaptive configuration strategies. We focus on variants of the bandit-based strategies *successive halving* and *Hyperband*, since these have shown strong performance, especially for optimizing deep learning algorithms. Strictly speaking, some of the methods which we will discuss in this subsection also model learning curves, but they provide no means of selecting new configurations based on these models.

First, however, we briefly describe the historical evolution of multi-fidelity algorithm selection methods. In 2000, Petrak [120] noted that simply testing various algorithms on a small subset of the data is a powerful and cheap mechanism to select an algorithm. Later approaches used iterative algorithm elimination schemes to drop hyperparameter configurations if they perform badly on subsets of the data [17], if they perform significantly worse than a group of top-performing configurations [86], if they perform worse than the best configuration by a user-specified factor [143], or if even an optimistic performance bound for an algorithm is worse than the best known algorithm [128]. Likewise, it is possible to drop

hyperparameter configurations if they perform badly on one or a few cross-validation folds [149]. Finally, Jamieson and Talwalkar [69] proposed to use the *successive halving* algorithm originally introduced by Karnin et al. [76] for HPO.

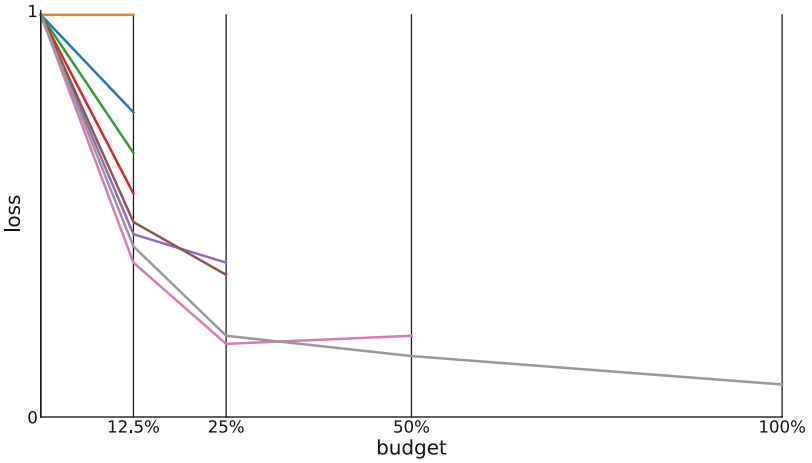

Fig. 1.3 Illustration of successive halving for eight algorithms/configurations. After evaluating all algorithms on $\frac{1}{8}$ of the total budget, half of them are dropped and the budget given to the remaining algorithms is doubled

Successive halving is an extremely simple, yet powerful, and therefore popular strategy for multi-fidelity algorithm selection: for a given initial budget, query all algorithms for that budget; then, remove the half that performed worst, double the budget [2] and successively repeat until only a single algorithm is left. This process is illustrated in Fig. 1.3. Jamieson and Talwalkar [69] benchmarked several common bandit methods and found that successive halving performs well both in terms of the number of required iterations and in the required computation time, that the algorithm theoretically outperforms a uniform budget allocation strategy if the algorithms converge favorably, and that it is preferable to many well-known bandit strategies from the literature, such as *UCB* and *EXP3*.

While successive halving is an efficient approach, it suffers from the budget-vs-number of configurations trade off. Given a total budget, the user has to decide beforehand whether to try many configurations and only assign a small budget to each, or to try only a few and assign them a larger budget. Assigning too small a budget can result in prematurely terminating good configurations, while assigning too large a budget can result in running poor configurations too long and thereby wasting resources.

[2] More precisely, drop the worst fraction $\frac{\eta-1}{\eta}$ of algorithms and multiply the budget for the remaining algorithms by η, where η is a hyperparameter. Its default value was changed from 2 to 3 with the introduction of HyperBand [90].

HyperBand [90] is a hedging strategy designed to combat this problem when selecting from randomly sampled configurations. It divides the total budget into several combinations of number of configurations vs. budget for each, to then call successive halving as a subroutine on each set of random configurations. Due to the hedging strategy which includes running some configurations only on the maximal budget, in the worst case, HyperBand takes at most a constant factor more time than vanilla random search on the maximal budget. In practice, due to its use of cheap low-fidelity evaluations, HyperBand has been shown to improve over vanilla random search and blackbox Bayesian optimization for data subsets, feature subsets and iterative algorithms, such as stochastic gradient descent for deep neural networks.

Despite HyperBand's success for deep neural networks it is very limiting to not adapt the configuration proposal strategy to the function evaluations. To overcome this limitation, the recent approach BOHB [33] combines Bayesian optimization and HyperBand to achieve the best of both worlds: strong anytime performance (quick improvements in the beginning by using low fidelities in HyperBand) and strong final performance (good performance in the long run by replacing HyperBand's random search by Bayesian optimization). BOHB also uses parallel resources effectively and deals with problem domains ranging from a few to many dozen hyperparameters. BOHB's Bayesian optimization component resembles TPE [12], but differs by using multidimensional kernel density estimators. It only fits a model on the highest fidelity for which at least $|\mathbf{\Lambda}| + 1$ evaluations have been performed (the number of hyperparameters, plus one). BOHB's first model is therefore fitted on the lowest fidelity, and over time models trained on higher fidelities take over, while still using the lower fidelities in successive halving. Empirically, BOHB was shown to outperform several state-of-the-art HPO methods for tuning support vector machines, neural networks and reinforcement learning algorithms, including most methods presented in this section [33]. Further approaches to combine HyperBand and Bayesian optimization have also been proposed [15, 151].

Multiple fidelity evaluations can also be combined with HPO in other ways. Instead of switching between lower fidelities and the highest fidelity, it is possible to perform HPO on a subset of the original data and extract the best-performing configurations in order to use them as an initial design for HPO on the full dataset [152]. To speed up solutions to the CASH problem, it is also possible to iteratively remove entire algorithms (and their hyperparameters) from the configuration space based on poor performance on small dataset subsets [159].

1.4.3 Adaptive Choices of Fidelities

All methods in the previous subsection follow a predefined schedule for the fidelities. Alternatively, one might want to actively choose which fidelities to evaluate given previous observations to prevent a misspecification of the schedule.

Multi-task Bayesian optimization [147] uses a multi-task Gaussian process to model the performance of related tasks and to automatically learn the tasks' correlation during the optimization process. This method can dynamically switch between cheaper, low-fidelity tasks and the expensive, high-fidelity target task based on a cost-aware information-theoretic acquisition function. In practice, the proposed method starts exploring the configuration space on the cheaper task and only switches to the more expensive configuration space in later parts of the optimization, approximately halving the time required for HPO. Multi-task Bayesian optimization can also be used to transfer information from previous optimization tasks, and we refer to Chap. 2 for further details.

Multi-task Bayesian optimization (and the methods presented in the previous subsection) requires an upfront specification of a set of fidelities. This can be suboptimal since these can be misspecified [74, 78] and because the number of fidelities that can be handled is low (usually five or less). Therefore, and in order to exploit the typically smooth dependence on the fidelity (such as, e.g., size of the data subset used), it often yields better results to treat the fidelity as continuous (and, e.g., choose a continuous percentage of the full data set to evaluate a configuration on), trading off the information gain and the time required for evaluation [78]. To exploit the domain knowledge that performance typically improves with more data, with diminishing returns, a special kernel can be constructed for the data subsets [78]. This generalization of multi-task Bayesian optimization improves performance and can achieve a 10–100 fold speedup compared to blackbox Bayesian optimization.

Instead of using an information-theoretic acquisition function, Bayesian optimization with the *Upper Confidence Bound* (UCB) acquisition function can also be extended to multiple fidelities [73, 74]. While the first such approach, MF-GP-UCB [73], required upfront fidelity definitions, the later BOCA algorithm [74] dropped that requirement. BOCA has also been applied to optimization with more than one continuous fidelity, and we expect HPO for more than one continuous fidelity to be of further interest in the future.

Generally speaking, methods that can adaptively choose their fidelity are very appealing and more powerful than the conceptually simpler bandit-based methods discussed in Sect. 1.4.2, but in practice we caution that strong models are required to make successful choices about the fidelities. When the models are not strong (since they do not have enough training data yet, or due to model mismatch), these methods may spend too much time evaluating higher fidelities, and the more robust fixed budget schedules discussed in Sect. 1.4.2 might yield better performance given a fixed time limit.

1.5 Applications to AutoML

In this section, we provide a historical overview of the most important hyperparameter optimization systems and applications to automated machine learning.

Grid search has been used for hyperparameter optimization since the 1990s [71, 107] and was already supported by early machine learning tools in 2002 [35]. The first adaptive optimization methods applied to HPO were greedy depth-first search [82] and pattern search [109], both improving over default hyperparameter configurations, and pattern search improving over grid search, too. Genetic algorithms were first applied to tuning the two hyperparameters C and γ of an RBF-SVM in 2004 [119] and resulted in improved classification performance in less time than grid search. In the same year, an evolutionary algorithm was used to learn a composition of three different kernels for an SVM, the kernel hyperparameters and to jointly select a feature subset; the learned combination of kernels was able to outperform every single optimized kernel. Similar in spirit, also in 2004, a genetic algorithm was used to select both the features used by and the hyperparameters of either an SVM or a neural network [129].

CMA-ES was first used for hyperparameter optimization in 2005 [38], in that case to optimize an SVM's hyperparameters C and γ, a kernel lengthscale l_i for each dimension of the input data, and a complete rotation and scaling matrix. Much more recently, CMA-ES has been demonstrated to be an excellent choice for parallel HPO, outperforming state-of-the-art Bayesian optimization tools when optimizing 19 hyperparameters of a deep neural network on 30 GPUs in parallel [91].

In 2009, Escalante et al. [30] extended the HPO problem to the *Full Model Selection* problem, which includes selecting a preprocessing algorithm, a feature selection algorithm, a classifier and all their hyperparameters. By being able to construct a machine learning pipeline from multiple off-the-shelf machine learning algorithms using HPO, the authors empirically found that they can apply their method to any data set as no domain knowledge is required, and demonstrated the applicability of their approach to a variety of domains [32, 49]. Their proposed method, particle swarm model selection (PSMS), uses a modified particle swarm optimizer to handle the conditional configuration space. To avoid overfitting, PSMS was extended with a custom ensembling strategy which combined the best solutions from multiple generations [31]. Since particle swarm optimization was originally designed to work on continuous configuration spaces, PSMS was later also extended to use a genetic algorithm to optimize the pipeline structure and only use particle swarm optimization to optimize the hyperparameters of each pipeline [145].

To the best of our knowledge, the first application of Bayesian optimization to HPO dates back to 2005, when Frohlich and Zell [39] used an online Gaussian process together with EI to optimize the hyperparameters of an SVM, achieving speedups of factor 10 (classification, 2 hyperparameters) and 100 (regression, 3 hyperparameters) over grid search. Tuned Data Mining [84] proposed to tune the hyperparameters of a full machine learning pipeline using Bayesian optimization; specifically, this used a single fixed pipeline and tuned the hyperparameters of the classifier as well as the per-class classification threshold and class weights.

In 2011, Bergstra et al. [12] were the first to apply Bayesian optimization to tune the hyperparameters of a deep neural network, outperforming both manual and random search. Furthermore, they demonstrated that TPE resulted in better

performance than a Gaussian process-based approach. TPE, as well as Bayesian optimization with random forests, were also successful for joint neural architecture search and hyperparameter optimization [14, 106].

Another important step in applying Bayesian optimization to HPO was made by Snoek et al. in the 2012 paper *Practical Bayesian Optimization of Machine Learning Algorithms* [140], which describes several tricks of the trade for Gaussian process-based HPO implemented in the Spearmint system and obtained a new state-of-the-art result for hyperparameter optimization of deep neural networks.

Independently of the Full Model Selection paradigm, Auto-WEKA [149] (see also Chap. 4) introduced the *Combined Algorithm Selection and Hyperparameter Optimization* (CASH) problem, in which the choice of a classification algorithm is modeled as a categorical variable, the algorithm hyperparameters are modeled as conditional hyperparameters, and the random-forest based Bayesian optimization system SMAC [59] is used for joint optimization in the resulting 786-dimensional configuration space.

In recent years, multi-fidelity methods have become very popular, especially in deep learning. Firstly, using low-fidelity approximations based on data subsets, feature subsets and short runs of iterative algorithms, Hyperband [90] was shown to outperform blackbox Bayesian optimization methods that did not take these lower fidelities into account. Finally, most recently, in the 2018 paper *BOHB: Robust and Efficient Hyperparameter Optimization at Scale*, Falkner et al. [33] introduced a robust, flexible, and parallelizable combination of Bayesian optimization and Hyperband that substantially outperformed both Hyperband and blackbox Bayesian optimization for a wide range of problems, including tuning support vector machines, various types of neural networks, and reinforcement learning algorithms.

At the time of writing, we make the following recommendations for which tools we would use in practical applications of HPO:

- If multiple fidelities are applicable (i.e., if it is possible to define substantially cheaper versions of the objective function of interest, such that the performance for these roughly correlates with the performance for the full objective function of interest), we recommend BOHB [33] as a robust, efficient, versatile, and parallelizable default hyperparameter optimization method.
- If multiple fidelities are not applicable:
  - If all hyperparameters are real-valued and one can only afford a few dozen function evaluations, we recommend the use of a Gaussian process-based Bayesian optimization tool, such as Spearmint [140].
  - For large and conditional configuration spaces we suggest either the random forest-based SMAC [59] or TPE [14], due to their proven strong performance on such tasks [29].
  - For purely real-valued spaces and relatively cheap objective functions, for which one can afford more than hundreds of evaluations, we recommend CMA-ES [51].

1.6 Open Problems and Future Research Directions

We conclude this chapter with a discussion of open problems, current research questions and potential further developments we expect to have an impact on HPO in the future. Notably, despite their relevance, we leave out discussions on hyperparameter importance and configuration space definition as these fall under the umbrella of meta-learning and can be found in Chap. 2.

1.6.1 Benchmarks and Comparability

Given the breadth of existing HPO methods, a natural question is what are the strengths and weaknesses of each of them. In order to allow for a fair comparison between different HPO approaches, the community needs to design and agree upon a common set of benchmarks that expands over time, as new HPO variants, such as multi-fidelity optimization, emerge. As a particular example for what this could look like we would like to mention the COCO platform (short for comparing continuous optimizers), which provides benchmark and analysis tools for continuous optimization and is used as a workbench for the yearly Black-Box Optimization Benchmarking (BBOB) challenge [11]. Efforts along similar lines in HPO have already yielded the hyperparameter optimization library (HPOlib [29]) and a benchmark collection specifically for Bayesian optimization methods [25]. However, neither of these has gained similar traction as the COCO platform.

Additionaly, the community needs clearly defined metrics, but currently different works use different metrics. One important dimension in which evaluations differ is whether they report performance on the validation set used for optimization or on a separate test set. The former helps to study the strength of the optimizer in isolation, without the noise that is added in the evaluation when going from validation to test set; on the other hand, some optimizers may lead to more overfitting than others, which can only be diagnosed by using the test set. Another important dimension in which evaluations differ is whether they report performance after a given number of function evaluations or after a given amount of time. The latter accounts for the difference in time between evaluating different hyperparameter configurations and includes optimization overheads, and therefore reflects what is required in practice; however, the former is more convenient and aids reproducibility by yielding the same results irrespective of the hardware used. To aid reproducibility, especially studies that use time should therefore release an implementation.

We note that it is important to compare against strong baselines when using new benchmarks, which is another reason why HPO methods should be published with an accompanying implementation. Unfortunately, there is no common software library as is, for example, available in deep learning research that implements all

the basic building blocks [2, 117]. As a simple, yet effective baseline that can be trivially included in empirical studies, Jamieson and Recht [68] suggest to compare against different parallelization levels of random search to demonstrate the speedups over regular random search. When comparing to other optimization techniques it is important to compare against a solid implementation, since, e.g., simpler versions of Bayesian optimization have been shown to yield inferior performance [79, 140, 142].

1.6.2 *Gradient-Based Optimization*

In some cases (e.g., least-squares support vector machines and neural networks) it is possible to obtain the gradient of the model selection criterion with respect to some of the model hyperparameters. Different to blackbox HPO, in this case each evaluation of the target function results in an entire hypergradient vector instead of a single float value, allowing for faster HPO.

Maclaurin et al. [99] described a procedure to compute the exact gradients of validation performance with respect to all continuous hyperparameters of a neural network by backpropagating through the entire training procedure of stochastic gradient descent with momentum (using a novel, memory-efficient algorithm). Being able to handle many hyperparameters efficiently through gradient-based methods allows for a new paradigm of hyperparametrizing the model to obtain flexibility over model classes, regularization, and training methods. Maclaurin et al. demonstrated the applicability of gradient-based HPO to many high-dimensional HPO problems, such as optimizing the learning rate of a neural network for each iteration and layer separately, optimizing the weight initialization scale hyperparameter for each layer in a neural network, optimizing the $l2$ penalty for each individual parameter in logistic regression, and learning completely new training datasets. As a small downside, backpropagating through the entire training procedure comes at the price of doubling the time complexity of the training procedure. The described method can also be generalized to work with other parameter update algorithms [36]. To overcome the necessity of backpropagating through the complete training procedure, later work allows to perform hyperparameter updates with respect to a separate validation set interleaved with the training process [5, 10, 36, 37, 93].

Recent examples of gradient-based optimization of simple model's hyperparameters [118] and of neural network structures (see Chap. 3) show promising results, outperforming state-of-the-art Bayesian optimization models. Despite being highly model-specific, the fact that gradient-based hyperparemeter optimization allows tuning several hundreds of hyperparameters could allow substantial improvements in HPO.

1.6.3 Scalability

Despite recent successes in multi-fidelity optimization, there are still machine learning problems which have not been directly tackled by HPO due to their scale, and which might require novel approaches. Here, scale can mean both the size of the configuration space and the expense of individual model evaluations. For example, there has not been any work on HPO for deep neural networks on the ImageNet challenge dataset [127] yet, mostly because of the high cost of training even a simple neural network on the dataset. It will be interesting to see whether methods going beyond the blackbox view from Sect. 1.3, such as the multi-fidelity methods described in Sect. 1.4, gradient-based methods, or meta-learning methods (described in Chap. 2) allow to tackle such problems. Chap. 3 describes first successes in learning neural network building blocks on smaller datasets and applying them to ImageNet, but the hyperparameters of the training procedure are still set manually.

Given the necessity of parallel computing, we are looking forward to new methods that fully exploit large-scale compute clusters. While there exists much work on parallel Bayesian optimization [12, 24, 33, 44, 54, 60, 135, 140], except for the neural networks described in Sect. 1.3.2.2 [141], so far no method has demonstrated scalability to hundreds of workers. Despite their popularity, and with a single exception of HPO applied to deep neural networks [91],[3] population-based approaches have not yet been shown to be applicable to hyperparameter optimization on datasets larger than a few thousand data points.

Overall, we expect that more sophisticated and specialized methods, leaving the blackbox view behind, will be needed to further scale hyperparameter to interesting problems.

1.6.4 Overfitting and Generalization

An open problem in HPO is overfitting. As noted in the problem statement (see Sect. 1.2), we usually only have a finite number of data points available for calculating the validation loss to be optimized and thereby do not necessarily optimize for generalization to unseen test datapoints. Similarly to overfitting a machine learning algorithm to training data, this problem is about overfitting the hyperparameters to the finite validation set; this was also demonstrated to happen experimentally [20, 81].

A simple strategy to reduce the amount of overfitting is to employ a different shuffling of the train and validation split for each function evaluation; this was shown to improve generalization performance for SVM tuning, both with a holdout and a cross-validation strategy [95]. The selection of the final configuration can

[3] See also Chap. 3 where population-based methods are applied to Neural Architecture Search problems.

be further robustified by not choosing it according to the lowest observed value, but according to the lowest predictive mean of the Gaussian process model used in Bayesian optimization [95].

Another possibility is to use a separate holdout set to assess configurations found by HPO to avoid bias towards the standard validation set [108, 159]. Different approximations of the generalization performance can lead to different test performances [108], and there have been reports that several resampling strategies can result in measurable performance differences for HPO of support vector machines [150].

A different approach to combat overfitting might be to find *stable optima* instead of *sharp optima* of the objective function [112]. The idea is that for stable optima, the function value around an optimum does not change for slight perturbations of the hyperparameters, whereas it *does* change for sharp optima. Stable optima lead to better generalization when applying the found hyperparameters to a new, unseen set of datapoints (i.e., the test set). An acquisition function built around this was shown to only slightly overfit for support vector machine HPO, while regular Bayesian optimization exhibited strong overfitting [112].

Further approaches to combat overfitting are the ensemble methods and Bayesian methods presented in Sect. 1.2.1. Given all these different techniques, there is no commonly agreed-upon technique for how to best avoid overfitting, though, and it remains up to the user to find out which strategy performs best on their particular HPO problem. We note that the best strategy might actually vary across HPO problems.

1.6.5 Arbitrary-Size Pipeline Construction

All HPO techniques we discussed so far assume a finite set of components for machine learning pipelines or a finite maximum number of layers in neural networks. For machine learning pipelines (see the AutoML systems covered in Part II of this book) it might be helpful to use more than one feature preprocessing algorithm and dynamically add them if necessary for a problem, enlarging the search space by a hyperparameter to select an appropriate preprocessing algorithm and its own hyperparameters. While a search space for standard blackbox optimization tools could easily include several extra such preprocessors (and their hyperparameters) as conditional hyperparameters, an unbounded number of these would be hard to support.

One approach for handling arbitrary-sized pipelines more natively is the tree-structured pipeline optimization toolkit (TPOT [115], see also Chap. 8), which uses genetic programming and describes possible pipelines by a grammar. TPOT uses multi-objective optimization to trade off pipeline complexity with performance to avoid generating unnecessarily complex pipelines.

A different pipeline creation paradigm is the usage of hierarchical planning; the recent ML-Plan [101, 108] uses hierarchical task networks and shows competitive performance compared to Auto-WEKA [149] and Auto-sklearn [34].

So far these approaches are not consistently outperforming AutoML systems with a fixed pipeline length, but larger pipelines may provide more improvement. Similarly, neural architecture search yields complex configuration spaces and we refer to Chap. 3 for a description of methods to tackle them.

Acknowledgements We would like to thank Luca Franceschi, Raghu Rajan, Stefan Falkner and Arlind Kadra for valuable feedback on the manuscript.

Bibliography

1. Proceedings of the International Conference on Learning Representations (ICLR'18) (2018), published online: iclr.cc
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/
3. Ahmed, M., Shahriari, B., Schmidt, M.: Do we need "harmless" Bayesian optimization and "first-order" Bayesian optimization. In: NeurIPS Workshop on Bayesian Optimization (BayesOpt'16) (2016)
4. Alaa, A., van der Schaar, M.: AutoPrognosis: Automated Clinical Prognostic Modeling via Bayesian Optimization with Structured Kernel Learning. In: Dy and Krause [27], pp. 139–148
5. Almeida, L.B., Langlois, T., Amaral, J.D., Plakhov, A.: Parameter Adaptation in Stochastic Optimization, p. 111–134. Cambridge University Press (1999)
6. Amazon: Automatic model tuning (2018), https://docs.aws.amazon.com/sagemaker/latest/dg/automatic-model-tuning.html
7. Bach, F., Blei, D. (eds.): Proceedings of the 32nd International Conference on Machine Learning (ICML'15), vol. 37. Omnipress (2015)
8. Balcan, M., Weinberger, K. (eds.): Proceedings of the 33rd International Conference on Machine Learning (ICML'17), vol. 48. Proceedings of Machine Learning Research (2016)
9. Bartlett, P., Pereira, F., Burges, C., Bottou, L., Weinberger, K. (eds.): Proceedings of the 26th International Conference on Advances in Neural Information Processing Systems (NeurIPS'12) (2012)
10. Baydin, A.G., Cornish, R., Rubio, D.M., Schmidt, M., Wood, F.: Online Learning Rate Adaption with Hypergradient Descent. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
11. BBOBies: Black-box Optimization Benchmarking (BBOB) workshop series (2018), http://numbbo.github.io/workshops/index.html
12. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for hyper-parameter optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger, K. (eds.) Proceedings of the 25th International Conference on Advances in Neural Information Processing Systems (NeurIPS'11). pp. 2546–2554 (2011)
13. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13, 281–305 (2012)

14. Bergstra, J., Yamins, D., Cox, D.: Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In: Dasgupta and McAllester [23], pp. 115–123
15. Bertrand, H., Ardon, R., Perrot, M., Bloch, I.: Hyperparameter optimization of deep neural networks: Combining hyperband with Bayesian model selection. In: Conférence sur l'Apprentissage Automatique (2017)
16. Bischl, B., Mersmann, O., Trautmann, H., Weihs, C.: Resampling methods for meta-model validation with recommendations for evolutionary computation. Evolutionary Computation 20(2), 249–275 (2012)
17. Van den Bosch, A.: Wrapped progressive sampling search for optimizing learning algorithm parameters. In: Proceedings of the sixteenth Belgian-Dutch Conference on Artificial Intelligence. pp. 219–226 (2004)
18. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. arXiv:1012.2599v1 [cs.LG] (2010)
19. Bürger, F., Pauli, J.: A Holistic Classification Optimization Framework with Feature Selection, Preprocessing, Manifold Learning and Classifiers., pp. 52–68. Springer (2015)
20. Cawley, G., Talbot, N.: On Overfitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. Journal of Machine Learning Research 11 (2010)
21. Chandrashekaran, A., Lane, I.: Speeding up Hyper-parameter Optimization by Extrapolation of Learning Curves using Previous Builds. In: Ceci, M., Hollmen, J., Todorovski, L., Vens, C., Džeroski, S. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'17). Lecture Notes in Computer Science, vol. 10534. Springer (2017)
22. Dahl, G., Sainath, T., Hinton, G.: Improving deep neural networks for LVCSR using rectified linear units and dropout. In: Adams, M., Zhao, V. (eds.) International Conference on Acoustics, Speech and Signal Processing (ICASSP'13). pp. 8609–8613. IEEE Computer Society Press (2013)
23. Dasgupta, S., McAllester, D. (eds.): Proceedings of the 30th International Conference on Machine Learning (ICML'13). Omnipress (2014)
24. Desautels, T., Krause, A., Burdick, J.: Parallelizing exploration-exploitation tradeoffs in Gaussian process bandit optimization. Journal of Machine Learning Research 15, 4053–4103 (2014)
25. Dewancker, I., McCourt, M., Clark, S., Hayes, P., Johnson, A., Ke, G.: A stratified analysis of Bayesian optimization methods. arXiv:1603.09441v1 [cs.LG] (2016)
26. Domhan, T., Springenberg, J.T., Hutter, F.: Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In: Yang, Q., Wooldridge, M. (eds.) Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI'15). pp. 3460–3468 (2015)
27. Dy, J., Krause, A. (eds.): Proceedings of the 35th International Conference on Machine Learning (ICML'18), vol. 80. Proceedings of Machine Learning Research (2018)
28. Eberhart, R., Shi, Y.: Comparison between genetic algorithms and particle swarm optimization. In: Porto, V., Saravanan, N., Waagen, D., Eiben, A. (eds.) 7th International conference on evolutionary programming. pp. 611–616. Springer (1998)
29. Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., Leyton-Brown, K.: Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13) (2013)
30. Escalante, H., Montes, M., Sucar, E.: Particle Swarm Model Selection. Journal of Machine Learning Research 10, 405–440 (2009)
31. Escalante, H., Montes, M., Sucar, E.: Ensemble particle swarm model selection. In: Proceedings of the 2010 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1–8. IEEE Computer Society Press (2010)
32. Escalante, H., Montes, M., Villaseñor, L.: Particle swarm model selection for authorship verification. In: Bayro-Corrochano, E., Eklundh, J.O. (eds.) Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications. pp. 563–570 (2009)

33. Falkner, S., Klein, A., Hutter, F.: BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In: Dy and Krause [27], pp. 1437–1446
34. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J.T., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (eds.) Proceedings of the 29th International Conference on Advances in Neural Information Processing Systems (NeurIPS'15). pp. 2962–2970 (2015)
35. Fischer, S., Klinkenberg, R., Mierswa, I., Ritthoff, O.: Yale: Yet another learning environment – tutorial. Tech. rep., University of Dortmund (2002)
36. Franceschi, L., Donini, M., Frasconi, P., Pontil, M.: Forward and Reverse Gradient-Based Hyperparameter Optimization. In: Precup and Teh [122], pp. 1165–1173
37. Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., Pontil, M.: Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In: Dy and Krause [27], pp. 1568–1577
38. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. Neurocomputing 64, 107–117 (2005)
39. Frohlich, H., Zell, A.: Efficient parameter selection for support vector machines in classification and regression via model-based global optimization. In: Prokhorov, D., Levine, D., Ham, F., Howell, W. (eds.) Proceedings of the 2005 IEEE International Joint Conference on Neural Networks (IJCNN). pp. 1431–1436. IEEE Computer Society Press (2005)
40. Gardner, J., Guo, C., Weinberger, K., Garnett, R., Grosse, R.: Discovering and Exploiting Additive Structure for Bayesian Optimization. In: Singh, A., Zhu, J. (eds.) Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 54, pp. 1311–1319. Proceedings of Machine Learning Research (2017)
41. Gardner, J., Kusner, M., Xu, Z., Weinberger, K., Cunningham, J.: Bayesian Optimization with Inequality Constraints. In: Xing and Jebara [157], pp. 937–945
42. Garrido-Merchán, E., Hernández-Lobato, D.: Dealing with integer-valued variables in Bayesian optimization with Gaussian processes. arXiv:1706.03673v2 [stats.ML] (2017)
43. Gelbart, M., Snoek, J., Adams, R.: Bayesian optimization with unknown constraints. In: Zhang, N., Tian, J. (eds.) Proceedings of the 30th conference on Uncertainty in Artificial Intelligence (UAI'14). AUAI Press (2014)
44. Ginsbourger, D., Le Riche, R., Carraro, L.: Kriging Is Well-Suited to Parallelize Optimization. In: Computational Intelligence in Expensive Optimization Problems, pp. 131–162. Springer (2010)
45. Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., Sculley, D.: Google Vizier: A service for black-box optimization. In: Matwin, S., Yu, S., Farooq, F. (eds.) Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 1487–1495. ACM Press (2017)
46. Gramacy, R., Lee, H.: Optimization under unknown constraints. Bayesian Statistics 9(9), 229–246 (2011)
47. Gretton, A., Robert, C. (eds.): Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS), vol. 51. Proceedings of Machine Learning Research (2016)
48. Guyon, I., von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.): Proceedings of the 31st International Conference on Advances in Neural Information Processing Systems (NeurIPS'17) (2017)
49. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Analysis of the IJCNN 2007 agnostic learning vs. prior knowledge challenge. Neural Networks 21(2), 544–550 (2008)
50. Guyon, I., Saffari, A., Dror, G., Cawley, G.: Model Selection: Beyond the Bayesian/Frequentist Divide. Journal of Machine Learning Research 11, 61–87 (2010)
51. Hansen, N.: The CMA evolution strategy: A tutorial. arXiv:1604.00772v1 [cs.LG] (2016)
52. Hazan, E., Klivans, A., Yuan, Y.: Hyperparameter optimization: A spectral approach. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
53. Hernandez-Lobato, D., Hernandez-Lobato, J., Shah, A., Adams, R.: Predictive Entropy Search for Multi-objective Bayesian Optimization. In: Balcan and Weinberger [8], pp. 1492–1501

54. Hernández-Lobato, J., Requeima, J., Pyzer-Knapp, E., Aspuru-Guzik, A.: Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In: Precup and Teh [122], pp. 1470–1479
55. Hernández-Lobato, J., Gelbart, M., Adams, R., Hoffman, M., Ghahramani, Z.: A general framework for constrained Bayesian optimization using information-based search. The Journal of Machine Learning Research 17(1), 5549–5601 (2016)
56. Hoeting, J., Madigan, D., Raftery, A., Volinsky, C.: Bayesian model averaging: a tutorial. Statistical science pp. 382–401 (1999)
57. Horn, D., Bischl, B.: Multi-objective parameter configuration of machine learning algorithms using model-based optimization. In: Likas, A. (ed.) 2016 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE Computer Society Press (2016)
58. Hutter, F.: Automated Configuration of Algorithms for Solving Hard Computational Problems. Ph.D. thesis, University of British Columbia, Department of Computer Science, Vancouver, Canada (2009)
59. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer (2011)
60. Hutter, F., Hoos, H., Leyton-Brown, K.: Parallel algorithm configuration. In: Hamadi, Y., Schoenauer, M. (eds.) Proceedings of the Sixth International Conference on Learning and Intelligent Optimization (LION'12). Lecture Notes in Computer Science, vol. 7219, pp. 55–70. Springer (2012)
61. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Xing and Jebara [157], pp. 754–762
62. Hutter, F., Hoos, H., Leyton-Brown, K., Murphy, K.: Time-bounded sequential parameter optimization. In: Blum, C. (ed.) Proceedings of the Fourth International Conference on Learning and Intelligent Optimization (LION'10). Lecture Notes in Computer Science, vol. 6073, pp. 281–298. Springer (2010)
63. Hutter, F., Osborne, M.: A kernel for hierarchical parameter spaces. arXiv:1310.5738v1 [stats.ML] (2013)
64. Hutter, F., Lücke, J., Schmidt-Thieme, L.: Beyond Manual Tuning of Hyperparameters. KI - Künstliche Intelligenz 29(4), 329–337 (2015)
65. Igel, C.: Multi-objective Model Selection for Support Vector Machines. In: Coello, C., Aguirre, A., Zitzler, E. (eds.) Evolutionary Multi-Criterion Optimization. pp. 534–546. Springer (2005)
66. Ihler, A., Janzing, D. (eds.): Proceedings of the 32nd conference on Uncertainty in Artificial Intelligence (UAI'16). AUAI Press (2016)
67. Ilievski, I., Akhtar, T., Feng, J., Shoemaker, C.: Efficient Hyperparameter Optimization for Deep Learning Algorithms Using Deterministic RBF Surrogates. In: Sierra, C. (ed.) Proceedings of the 27th International Joint Conference on Artificial Intelligence (IJCAI'17) (2017)
68. Jamieson, K., Recht, B.: The news on auto-tuning (2016), http://www.argmin.net/2016/06/20/hypertuning/
69. Jamieson, K., Talwalkar, A.: Non-stochastic best arm identification and hyperparameter optimization. In: Gretton and Robert [47], pp. 240–248
70. Jenatton, R., Archambeau, C., González, J., Seeger, M.: Bayesian Optimization with Tree-structured Dependencies. In: Precup and Teh [122], pp. 1655–1664
71. John, G.: Cross-Validated C4.5: Using Error Estimation for Automatic Parameter Selection. Tech. Rep. STAN-CS-TN-94-12, Stanford University, Stanford University (1994)
72. Jones, D., Schonlau, M., Welch, W.: Efficient global optimization of expensive black box functions. Journal of Global Optimization 13, 455–492 (1998)
73. Kandasamy, K., Dasarathy, G., Oliva, J., Schneider, J., Póczos, B.: Gaussian Process Bandit Optimisation with Multi-fidelity Evaluations. In: Lee et al. [87], pp. 992–1000

74. Kandasamy, K., Dasarathy, G., Schneider, J., Póczos, B.: Multi-fidelity Bayesian Optimisation with Continuous Approximations. In: Precup and Teh [122], pp. 1799–1808
75. Kandasamy, K., Schneider, J., Póczos, B.: High Dimensional Bayesian Optimisation and Bandits via Additive Models. In: Bach and Blei [7], pp. 295–304
76. Karnin, Z., Koren, T., Somekh, O.: Almost optimal exploration in multi-armed bandits. In: Dasgupta and McAllester [23], pp. 1238–1246
77. King, R., Feng, C., Sutherland, A.: Statlog: comparison of classification algorithms on large real-world problems. Applied Artificial Intelligence an International Journal 9(3), 289–333 (1995)
78. Klein, A., Falkner, S., Bartels, S., Hennig, P., Hutter, F.: Fast bayesian hyperparameter optimization on large datasets. In: Electronic Journal of Statistics. vol. 11 (2017)
79. Klein, A., Falkner, S., Mansur, N., Hutter, F.: RoBO: A flexible and robust Bayesian optimization framework in Python. In: NeurIPS workshop on Bayesian Optimization (BayesOpt'17) (2017)
80. Klein, A., Falkner, S., Springenberg, J.T., Hutter, F.: Learning curve prediction with Bayesian neural networks. In: Proceedings of the International Conference on Learning Representations (ICLR'17) (2017), published online: iclr.cc
81. Koch, P., Konen, W., Flasch, O., Bartz-Beielstein, T.: Optimizing support vector machines for stormwater prediction. Tech. Rep. TR10-2-007, Technische Universität Dortmund (2010)
82. Kohavi, R., John, G.: Automatic Parameter Selection by Minimizing Estimated Error. In: Prieditis, A., Russell, S. (eds.) Proceedings of the Twelfth International Conference on Machine Learning, pp. 304–312. Morgan Kaufmann Publishers (1995)
83. Komer, B., Bergstra, J., Eliasmith, C.: Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn. In: Hutter, F., Caruana, R., Bardenet, R., Bilenko, M., Guyon, I., Kégl, B., Larochelle, H. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2014) (2014)
84. Konen, W., Koch, P., Flasch, O., Bartz-Beielstein, T., Friese, M., Naujoks, B.: Tuned data mining: a benchmark study on different tuners. In: Krasnogor, N. (ed.) Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation (GECCO'11). pp. 1995–2002. ACM (2011)
85. Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. In: Bartlett et al. [9], pp. 1097–1105
86. Krueger, T., Panknin, D., Braun, M.: Fast cross-validation via sequential testing. Journal of Machine Learning Research (2015)
87. Lee, D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.): Proceedings of the 30th International Conference on Advances in Neural Information Processing Systems (NeurIPS'16) (2016)
88. Lee, H., Gramacy, R.: Optimization Subject to Hidden Constraints via Statistical Emulation. Pacific Journal of Optimization 7(3), 467–478 (2011)
89. Li, F.F., Li, J.: Cloud AutoML: Making AI accessible to every business (2018), https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/
90. Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., Talwalkar, A.: Hyperband: A novel bandit-based approach to hyperparameter optimization. Journal of Machine Learning Research 18(185), 1–52 (2018)
91. Loshchilov, I., Hutter, F.: CMA-ES for hyperparameter optimization of deep neural networks. In: International Conference on Learning Representations Workshop track (2016), published online: iclr.cc
92. Lu, X., Gonzalez, J., Dai, Z., Lawrence, N.: Structured Variationally Auto-encoded Optimization. In: Dy and Krause [27], pp. 3273–3281
93. Luketina, J., Berglund, M., Greff, K., Raiko, T.: Scalable Gradient-Based Tuning of Continuous Regularization Hyperparameters. In: Balcan and Weinberger [8], pp. 2952–2960
94. Luo, G.: A review of automatic selection methods for machine learning algorithms and hyperparameter values. Network Modeling Analysis in Health Informatics and Bioinformatics 5(1) (2016)

95. Lévesque, J.C.: Bayesian Hyperparameter Optimization: Overfitting, Ensembles and Conditional Spaces. Ph.D. thesis, Université Laval (2018)
96. Lévesque, J.C., Durand, A., Gagné, C., Sabourin, R.: Bayesian optimization for conditional hyperparameter spaces. In: Howell, B. (ed.) 2017 International Joint Conference on Neural Networks (IJCNN). pp. 286–293. IEEE (2017)
97. Lévesque, J.C., Gagné, C., Sabourin, R.: Bayesian Hyperparameter Optimization for Ensemble Learning. In: Ihler and Janzing [66], pp. 437–446
98. MacKay, D.: Hyperparameters: Optimize, or Integrate Out?, pp. 43–59. Springer (1996)
99. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based Hyperparameter Optimization through Reversible Learning. In: Bach and Blei [7], pp. 2113–2122
100. Mantovani, R., Horvath, T., Cerri, R., Vanschoren, J., Carvalho, A.: Hyper-Parameter Tuning of a Decision Tree Induction Algorithm. In: 2016 5th Brazilian Conference on Intelligent Systems (BRACIS). pp. 37–42. IEEE Computer Society Press (2016)
101. Marcel Wever, F.M., Hüllermeier, E.: ML-Plan for unlimited-length machine learning pipelines. In: Garnett, R., Vanschoren, F.H.J., Brazdil, P., Caruana, R., Giraud-Carrier, C., Guyon, I., Kégl, B. (eds.) ICML workshop on Automated Machine Learning (AutoML workshop 2018) (2018)
102. Maron, O., Moore, A.: The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1–5), 193–225 (1997)
103. McInerney, J.: An Empirical Bayes Approach to Optimizing Machine Learning Algorithms. In: Guyon et al. [48], pp. 2712–2721
104. McIntire, M., Ratner, D., Ermon, S.: Sparse Gaussian Processes for Bayesian Optimization. In: Ihler and Janzing [66]
105. Melis, G., Dyer, C., Blunsom, P.: On the state of the art of evaluation in neural language models. In: Proceedings of the International Conference on Learning Representations (ICLR'18) [1], published online: iclr.cc
106. Mendoza, H., Klein, A., Feurer, M., Springenberg, J., Hutter, F.: Towards automatically-tuned neural networks. In: ICML 2016 AutoML Workshop (2016)
107. Michie, D., Spiegelhalter, D., Taylor, C., Campbell, J. (eds.): Machine Learning, Neural and Statistical Classification. Ellis Horwood (1994)
108. Mohr, F., Wever, M., Höllermeier, E.: ML-Plan: Automated machine learning via hierarchical planning. Machine Learning 107(8–10), 1495–1515 (2018)
109. Momma, M., Bennett, K.: A Pattern Search Method for Model Selection of Support Vector Regression. In: Proceedings of the 2002 SIAM International Conference on Data Mining, pp. 261–274 (2002)
110. Montgomery, D.: Design and analysis of experiments. John Wiley & Sons, Inc, eighth edn. (2013)
111. Murray, I., Adams, R.: Slice sampling covariance hyperparameters of latent Gaussian models. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'10). pp. 1732–1740 (2010)
112. Nguyen, T., Gupta, S., Rana, S., Venkatesh, S.: Stable Bayesian Optimization. In: Kim, J., Shim, K., Cao, L., Lee, J.G., Lin, X., Moon, Y.S. (eds.) Advances in Knowledge Discovery and Data Mining (PAKDD'17). Lecture Notes in Artificial Intelligence, vol. 10235, pp. 578–591 (2017)
113. Nguyen, V., Gupta, S., Rana, S., Li, C., Venkatesh, S.: Filtering Bayesian optimization approach in weakly specified search space. Knowledge and Information Systems (2018)
114. Oh, C., Gavves, E., Welling, M.: BOCK: Bayesian Optimization with Cylindrical Kernels. In: Dy and Krause [27], pp. 3865–3874
115. Olson, R., Bartley, N., Urbanowicz, R., Moore, J.: Evaluation of a Tree-based Pipeline Optimization Tool for Automating Data Science. In: Friedrich, T. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'16). pp. 485–492. ACM (2016)
116. Olson, R., La Cava, W., Mustahsan, Z., Varik, A., Moore, J.: Data-driven advice for applying machine learning to bioinformatics problems. In: Proceedings of the Pacific Symposium in Biocomputing 2018. pp. 192–203 (2018)

117. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in PyTorch. In: NeurIPS Autodiff Workshop (2017)
118. Pedregosa, F.: Hyperparameter optimization with approximate gradient. In: Balcan and Weinberger [8], pp. 737–746
119. Peng-Wei Chen, Jung-Ying Wang, Hahn-Ming Lee: Model selection of SVMs using GA approach. In: Proceedings of the 2004 IEEE International Joint Conference on Neural Networks (IJCNN). vol. 3, pp. 2035–2040. IEEE Computer Society Press (2004)
120. Petrak, J.: Fast subsampling performance estimates for classification algorithm selection. Technical Report TR-2000-07, Austrian Research Institute for Artificial Intelligence (2000)
121. Poloczek, M., Wang, J., Frazier, P.: Multi-Information Source Optimization. In: Guyon et al. [48], pp. 4288–4298
122. Precup, D., Teh, Y. (eds.): Proceedings of the 34th International Conference on Machine Learning (ICML'17), vol. 70. Proceedings of Machine Learning Research (2017)
123. Provost, F., Jensen, D., Oates, T.: Efficient progressive sampling. In: Fayyad, U., Chaudhuri, S., Madigan, D. (eds.) The 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'99). pp. 23–32. ACM Press (1999)
124. Rasmussen, C., Williams, C.: Gaussian Processes for Machine Learning. The MIT Press (2006)
125. Rendle, S.: Factorization machines. In: Webb, G., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (eds.) Proceedings of the 10th IEEE International Conference on Data Mining (ICDM'06). pp. 995–1000. IEEE Computer Society Press (2010)
126. Ripley, B.D.: Statistical aspects of neural networks. Networks and chaos—statistical and probabilistic aspects 50, 40–123 (1993)
127. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., Fei-Fei, L.: Imagenet large scale visual recognition challenge. International Journal of Computer Vision 115(3), 211–252 (2015)
128. Sabharwal, A., Samulowitz, H., Tesauro, G.: Selecting Near-Optimal Learners via Incremental Data Allocation. In: Schuurmans, D., Wellman, M. (eds.) Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16). AAAI Press (2016)
129. Samanta, B.: Gear fault detection using artificial neural networks and support vector machines with genetic algorithms. Mechanical Systems and Signal Processing 18(3), 625–644 (2004)
130. Sanders, S., Giraud-Carrier, C.: Informing the Use of Hyperparameter Optimization Through Metalearning. In: Gottumukkala, R., Ning, X., Dong, G., Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X. (eds.) 2017 IEEE International Conference on Big Data (Big Data). IEEE Computer Society Press (2017)
131. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Hyperparameter optimization with factorized multilayer perceptrons. In: Appice, A., Rodrigues, P., Costa, V., Gama, J., Jorge, A., Soares, C. (eds.) Machine Learning and Knowledge Discovery in Databases (ECML/PKDD'15). Lecture Notes in Computer Science, vol. 9285, pp. 87–103. Springer (2015)
132. Schilling, N., Wistuba, M., Drumond, L., Schmidt-Thieme, L.: Joint Model Choice and Hyperparameter Optimization with Factorized Multilayer Perceptrons. In: 2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI). pp. 72–79. IEEE Computer Society Press (2015)
133. Sculley, D., Snoek, J., Wiltschko, A., Rahimi, A.: Winner's curse? on pace, progress, and empirical rigor. In: International Conference on Learning Representations Workshop track (2018), published online: iclr.cc
134. Shah, A., Ghahramani, Z.: Pareto Frontier Learning with Expensive Correlated Objectives. In: Balcan and Weinberger [8], pp. 1919–1927
135. Shahriari, B., Swersky, K., Wang, Z., Adams, R., de Freitas, N.: Taking the human out of the loop: A review of Bayesian optimization. Proceedings of the IEEE 104(1), 148–175 (2016)
136. Shahriari, B., Bouchard-Cote, A., de Freitas, N.: Unbounded Bayesian optimization via regularization. In: Gretton and Robert [47], pp. 1168–1176

137. SIGOPT: Improve ML models 100x faster (2018), https://sigopt.com/
138. Simon, D.: Evolutionary optimization algorithms. John Wiley & Sons (2013)
139. Snoek, J.: Bayesian optimization and semiparametric models with applications to assistive technology. PhD Thesis, University of Toronto (2013)
140. Snoek, J., Larochelle, H., Adams, R.: Practical Bayesian optimization of machine learning algorithms. In: Bartlett et al. [9], pp. 2960–2968
141. Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, Adams, R.: Scalable Bayesian optimization using deep neural networks. In: Bach and Blei [7], pp. 2171–2180
142. Snoek, J., Swersky, K., Zemel, R., Adams, R.: Input warping for Bayesian optimization of non-stationary functions. In: Xing and Jebara [157], pp. 1674–1682
143. Sparks, E., Talwalkar, A., Haas, D., Franklin, M., Jordan, M., Kraska, T.: Automating model search for large scale machine learning. In: Balazinska, M. (ed.) Proceedings of the Sixth ACM Symposium on Cloud Computing - SoCC '15. pp. 368–380. ACM Press (2015)
144. Springenberg, J., Klein, A., Falkner, S., Hutter, F.: Bayesian optimization with robust Bayesian neural networks. In: Lee et al. [87]
145. Sun, Q., Pfahringer, B., Mayo, M.: Towards a Framework for Designing Full Model Selection and Optimization Systems. In: Multiple Classifier Systems, vol. 7872, pp. 259–270. Springer (2013)
146. Swersky, K., Duvenaud, D., Snoek, J., Hutter, F., Osborne, M.: Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In: NeurIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'14) (2014)
147. Swersky, K., Snoek, J., Adams, R.: Multi-task Bayesian optimization. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Proceedings of the 27th International Conference on Advances in Neural Information Processing Systems (NeurIPS'13). pp. 2004–2012 (2013)
148. Swersky, K., Snoek, J., Adams, R.: Freeze-thaw Bayesian optimization arXiv:1406.3896v1 [stats.ML] (2014)
149. Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon, I., Koren, Y., Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (eds.) The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13). pp. 847–855. ACM Press (2013)
150. Wainer, J., Cawley, G.: Empirical Evaluation of Resampling Procedures for Optimising SVM Hyperparameters. Journal of Machine Learning Research 18, 1–35 (2017)
151. Wang, J., Xu, J., Wang, X.: Combination of hyperband and Bayesian optimization for hyperparameter optimization in deep learning. arXiv:1801.01596v1 [cs.CV] (2018)
152. Wang, L., Feng, M., Zhou, B., Xiang, B., Mahadevan, S.: Efficient Hyper-parameter Optimization for NLP Applications. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing. pp. 2112–2117. Association for Computational Linguistics (2015)
153. Wang, Z., Hutter, F., Zoghi, M., Matheson, D., de Feitas, N.: Bayesian optimization in a billion dimensions via random embeddings. Journal of Artificial Intelligence Research 55, 361–387 (2016)
154. Wang, Z., Gehring, C., Kohli, P., Jegelka, S.: Batched Large-scale Bayesian Optimization in High-dimensional Spaces. In: Storkey, A., Perez-Cruz, F. (eds.) Proceedings of the 21st International Conference on Artificial Intelligence and Statistics (AISTATS). vol. 84. Proceedings of Machine Learning Research (2018)
155. Wistuba, M., Schilling, N., Schmidt-Thieme, L.: Automatic Frankensteining: Creating Complex Ensembles Autonomously. In: Proceedings of the 2017 SIAM International Conference on Data Mining (2017)
156. Wolpert, D.: Stacked generalization. Neural Networks 5(2), 241–259 (1992)
157. Xing, E., Jebara, T. (eds.): Proceedings of the 31th International Conference on Machine Learning, (ICML'14). Omnipress (2014)

158. Zabinsky, Z.: Pure Random Search and Pure Adaptive Search. In: Stochastic Adaptive Search for Global Optimization, pp. 25–54. Springer (2003)
159. Zeng, X., Luo, G.: Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection. Health Information Science and Systems 5(1) (2017)
160. Zhang, Y., Bahadori, M.T., Su, H., Sun, J.: FLASH: Fast Bayesian Optimization for Data Analytic Pipelines. In: Krishnapuram, B., Shah, M., Smola, A., Aggarwal, C., Shen, D., Rastogi, R. (eds.) Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). pp. 2065–2074. ACM Press (2016)