BOHB: Robust and Efficient Hyperparameter Optimization at Scale

Stefan Falkner [1] **Aaron Klein** [1] **Frank Hutter** [1]

Abstract

Modern deep learning methods are very sensitive to many hyperparameters, and, due to the long training times of state-of-the-art models, vanilla Bayesian hyperparameter optimization is typically computationally infeasible. On the other hand, bandit-based configuration evaluation approaches based on random search lack guidance and do not converge to the best configurations as quickly. Here, we propose to combine the benefits of both Bayesian optimization and bandit-based methods, in order to achieve the best of both worlds: strong anytime performance and fast convergence to optimal configurations. We propose a new practical state-of-the-art hyperparameter optimization method, which consistently outperforms both Bayesian optimization and Hyperband on a wide range of problem types, including high-dimensional toy functions, support vector machines, feed-forward neural networks, Bayesian neural networks, deep reinforcement learning, and convolutional neural networks. Our method is robust and versatile, while at the same time being conceptually simple and easy to implement.

1. Introduction

Machine learning has recently achieved great successes in a wide range of practical applications, but the performance of the most prominent methods depends more strongly than ever on the correct setting of many internal hyperparameters (see, e.g., Henderson et al. (2017); Melis et al. (2017)). The best-performing models for many modern applications of deep learning are getting ever larger and thus more computationally expensive to train, but at the same time both researchers and practitioners desire to set as many hyperparameters automatically as possible. These constraints call for a practical solution to the hyperparameter optimization (HPO) problem that fulfills many desiderata:

1. Strong Anytime Performance. Since large contemporary neural networks often require days or even weeks to train, HPO methods that view performance as a black box function to be optimized require extreme resources. The overall budget that most researchers and practitioners can afford during development is often not much larger than that of fully training a handful of models, and hence practical HPO methods must go beyond this blackbox view to already yield good configurations with such a small budget.

2. Strong Final Performance. On the other hand, what matters most at deployment time is the performance of the best configuration a HPO method can find given a larger budget. Since finding the best configurations in a large space requires guidance, this is where methods based on random search struggle.

3. Effective Use of Parallel Resources. With the rise of parallel computing, large parallel resources are often available (e.g., compute clusters or cloud computing), and practical HPO methods need to be able to use these effectively.

4. Scalability. Modern deep neural networks require the setting of a multitude of hyperparameters, including architectural choices (e.g., the number and width of layers), optimization hyperparameters (e.g., learning rate schedules, momentum, and batch size), and regularization hyperparameters (e.g., weight decay and dropout rates). Practical modern HPO methods therefore must be able to easily handle problems ranging from just a few to many dozens of hyperparameters.

5. Robustness & Flexibility. The challenges for hyperparameter optimization vary substantially across subfields of machine learning; e.g., deep reinforcement learning systems are known to be very noisy (Henderson et al., 2017), while probabilistic deep learning is often very sensitive to a few key hyperparameters. Different hyperparameter optimization problems also give rise to different types of hyperparameters (such as binary, categorical, integer, and continuous), each of which needs to be handled effectively by a practical HPO method.

As we will discuss in Section 2, while there has been a lot of recent progress in the field of hyperparameter optimization,

[1]Department of Computer Science, University of Freiburg, Freiburg, Germany. Correspondence to: Stefan Falkner <sfalkner@informatik.uni-freiburg.de>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

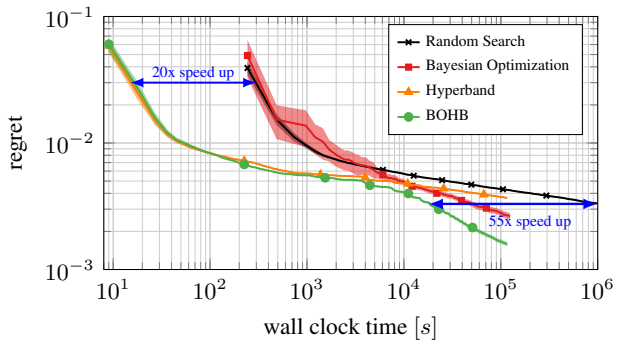

Figure 1. Illustration of typical results obtained, here for optimizing six hyperparameters of a neural network. We show the immediate regret of the best configuration found by 4 methods as a function of time. Hyperband has strong anytime performance, but for larger budgets does not perform much better than random search. In contrast, Bayesian optimization starts slowly (like random search), but given enough time outperforms Hyperband. Our new method BOHB achieves the best of both worlds, starting fast and also converging to the global optimum quickly.

all existing methods have some strengths and weaknesses, but none of them fulfills all of these desiderata. The key contribution of this paper is therefore to combine the strengths of several methods (in particular, Hyperband (Li et al., 2017) and a robust & effective variant (Bergstra et al., 2011) of Bayesian optimization (Brochu et al., 2010; Shahriari et al., 2016)) to propose a practical HPO method that fulfills all of these desiderata. We first describe Bayesian optimization and Hyperband in more detail (Section 3) and then show how to combine them in our new method BOHB, as well as how to effectively parallelize the resulting system (Section 4). Our extensive empirical evaluation (Section 5) demonstrates that our method combines the best aspects of Bayesian optimization and Hyperband: it often finds good solutions over an order of magnitude faster than Bayesian optimization and converges to the best solutions orders of magnitudes faster than Hyperband. Figure 1 illustrates this pattern in a nutshell for optimizing six hyperparameters of a neural network.

In this paper, we focus only on combining Hyperband and Bayesian optimization, but we would like to mention that methods improving BO are potentially applicable to BOHB as well, such as meta learning (Swersky et al., 2013; Feurer et al., 2015; Poloczek et al., 2016; Springenberg et al., 2016), active ensembling to combine models found during the optimization (Lévesque et al., 2016), and using multiple fidelities (Swersky et al., 2013; Kandasamy et al., 2017). The data gathered by BOHB on different budgets could also be used to quantify the importance of hyperparameters (Hutter et al., 2014; Biedenkapp et al., 2017; Golovin et al., 2017; van Rijn & Hutter, 2018). We leave these for future work.

2. Related Work on Model-based Hyperparamter Optimization

Bayesian optimization has been successfully applied to optimize hyperparameters of neural networks in many works: Snoek et al. (2012) obtained state-of-the-art performance on CIFAR-10 by optimizing the hyperparameters of convolutional neural networks; Bergstra et al. (2014) used TPE (Bergstra et al., 2011) to optimize a highly parameterized three layer convolutional neural network; and Mendoza et al. (2016) won 3 datasets in the 2016 AutoML challenge by automatically finding the right architecture and hyperparameters for fully-connected neural networks.

Gaussian processes are the most commonly-used probabilistic model in Bayesian optimization (Shahriari et al., 2016), since they obtain smooth and well-calibrated uncertainty estimates. However, Gaussian processes do not typically scale well to high dimensions and exhibit cubic complexity in the number of data points (scalability); they also do not apply to complex configuration spaces without special kernels (flexibility) and require carefully-set hyperpriors (robustness).

To speed up the hyperparameter optimization of machine learning algorithms, recent methods in Bayesian optimization try to extend the traditional blackbox setting by exploiting cheaper fidelities of the objective function (Swersky et al., 2014; Klein et al., 2017a; Swersky et al., 2013; Kandasamy et al., 2017; Klein et al., 2017c; Poloczek et al., 2017). For instance, multi-task Bayesian optimization (Swersky et al., 2013) exploits correlation between tasks to warm-start the optimization procedure. Fabolas (Klein et al., 2017a) uses similar techniques to evaluate configurations on subsets of the training data and to extrapolate their performance to the full dataset. Even though these methods achieved both good anytime and final performance, they are based on Gaussian processes, which, as described above, do not satisfy all of our desiderata. Alternative models, such as random forests (Hutter et al., 2011) or Bayesian neural networks (Snoek et al., 2015; Springenberg et al., 2016; Perrone et al., 2017), scale better with the number of dimensions, but with the exception of Klein et al. (2017c) have not yet been adopted for multi-fidelity optimization.

Hyperband (Li et al., 2017) is a bandit strategy that dynamically allocates resources to a set of random configurations and uses successive halving (Jamieson & Talwalkar, 2016) to stop poorly performing configurations. We describe this in more detail in Section 3.2. Compared to Bayesian optimization methods that do not use multiple fidelities, Hyperband showed strong anytime performance, as well as flexibility and scalability to higher-dimensional spaces. However, it only samples configurations randomly and does not learn from previously sampled configurations. This can lead to a worse final performance than model-based approaches, as

we show empirically in Section 5.

Concurrently to our work, two other groups (Bertrand et al., 2017; Wang et al., 2018) also attempted to combine Bayesian optimization with Hyperband. However, neither of them achieve the consistent and large speedups our method achieves. Furthermore, the method of Bertrand et al. (2017) is based on Gaussian processes and thus shares the limitations discussed above. We discuss differences between our work and these two papers in more detail in Appendix B.

3. Bayesian Optimization and Hyperband

The validation performance of machine learning algorithms can be modelled as a function $f : \mathcal{X} \to \mathbb{R}$ of their hyperparameters $\boldsymbol{x} \in \mathcal{X}$. We note that the hyperparameter configuration space $\mathcal{X}$ can include both discrete and continuous dimensions. The hyperparameter optimization (HPO) problem is then defined as finding $\boldsymbol{x}_\star \in \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{x})$.

Due to the intrinsic randomness of most machine learning algorithms (e. g. stochastic gradient descent), we assume that we cannot observe $f(\boldsymbol{x})$ directly but rather only trough noisy observations $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2_{noise})$. We now discuss the two methods for tackling this optimization problem in more detail that we will use as components of our new method: Bayesian optimization and Hyperband.

3.1. Bayesian Optimization

In each iteration i, Bayesian optimization (BO) uses a probabilistic model $p(f|D)$ to model the objective function f based on the already observed data points $D = \{(\boldsymbol{x}_0, y_0), \ldots, (\boldsymbol{x}_{i-1}, y_{i-1})\}$. BO uses an acquisition function $a : \mathcal{X} \to \mathbb{R}$ based on the current model $p(f|D)$ that trades off exploration and exploitation. Based on the model and the acquisition function, it iterates the following three steps: (1) select the point that maximizes the acquisition function $\boldsymbol{x}_{new} = \arg\max_{\boldsymbol{x}\in\mathcal{X}} a(\boldsymbol{x})$, (2) evaluate the objective function $y_{new} = f(\boldsymbol{x}_{new}) + \epsilon$, and (3) augment the data $D \leftarrow D \cup (\boldsymbol{x}_{new}, y_{new})$ and refit the model. A common acquisition function is the expected improvement (EI) over the currently best observed value $\alpha = \min\{y_0, \ldots, y_n\}$:

$$a(\boldsymbol{x}) = \int \max(0, \alpha - f(\boldsymbol{x}))dp(f|D). \tag{1}$$

Tree Parzen Estimator. The Tree Parzen Estimator (TPE) (Bergstra et al., 2011) is a Bayesian optimization method that uses a kernel density estimator to model the densities

$$\begin{aligned} l(\boldsymbol{x}) &= p(y < \alpha|\boldsymbol{x}, D) \\ g(\boldsymbol{x}) &= p(y > \alpha|\boldsymbol{x}, D) \end{aligned} \tag{2}$$

over the input configuration space instead of modeling the objective function f directly by $p(f|D)$. To select a new candidate $\boldsymbol{x}_{new}$ to evaluate, it maximizes the ratio $l(\boldsymbol{x})/g(\boldsymbol{x})$; Bergstra et al. (2011) showed that this is equivalent to maximizing EI in Equation (1). Due to the nature of kernel density estimators, TPE easily supports mixed continuous and discrete spaces, and model construction scales linearly in the number of data points (in contrast to the cubic-time Gaussian processes (GPs) predominant in the BO literature).

3.2. Hyperband

While the objective function $f : \mathcal{X} \to \mathbb{R}$ is typically expensive to evaluate (since it requires training a machine learning model with the specified hyperparameters), in most applications it is possible to define cheap-to-evaluate approximate versions $\tilde{f}(\cdot, b)$ of $f(\cdot)$ that are parameterized by a so-called *budget* $b \in [b_{min}, b_{max}]$. With the maximum budget $b = b_{max}$, we have $\tilde{f}(\cdot, b_{max}) = f(\cdot)$, whereas with $b < b_{max}$, $\tilde{f}(\cdot, b)$ is only an approximation of $f(\cdot)$ whose quality typically increases with b. In our experiments, we will use this budget to encode the number of iterations for an iterative algorithm, the number of data points used, the number of steps in an MCMC chain, and the number of trials in deep reinforcement learning.

Hyperband (HB) (Li et al., 2017) is a multi-armed bandit strategy for hyperparameter optimization that takes advantage of these different budgets b by repeatedly calling SuccessiveHalving (SH) (Jamieson & Talwalkar, 2016) to identify the best out of n randomly sampled configurations. It balances very agressive evaluations with many configurations on the smallest budget, and very conservative runs that are directly evaluated on b_{max}. The exact procedure for this trade-off is shown in Algorithm 1 (with pseudocode for SH shown in Appendix C). Line 1 computes the geometrically spaced budget $\in [b_{min}, b_{max}]$. The number of configurations sampled in line 3 is chosen such that every SH run requires the same total budget. SH internally evaluates configurations on a given budget, ranks them by their performance, and continues the top η^{-1} (usually the best-performing third) on a budget η times larger. This is repeated until the maximum budget is reached. In practice, HB works very well and typically outperforms random search and Bayesian optimization methods operating on the full function evaluation budget quite easily for small to medium total budgets. However, its convergence to the global optimum is limited by its reliance on randomly-drawn configurations, and with large budgets its advantage over random search typically diminishes.

4. Model-Based Hyperband

We now introduce our new practical HPO method, which we dub *BOHB* since it combines Bayesian optimization (BO) and Hyperband (HB). We designed BOHB to satisfy all the desiderata described in the introduction. HB already satis-

Algorithm 1: Pseudocode for Hyperband using SuccessiveHalving (SH) as a subroutine.

input : budgets b_{min} and b_{max}, η
$s_{max} = \lfloor \log_\eta \frac{b_{max}}{b_{min}} \rfloor$
for $s \in \{s_{max}, s_{max}-1, \dots, 0\}$ **do**
  sample $n = \lceil \frac{s_{max}+1}{s+1} \cdot \eta^s \rceil$ configurations
  run SH on them with $\eta^s \cdot b_{max}$ as initial budget

fies most of these desiderata (in particular, strong anytime performance, scalability, robustness and flexibility), and we combine it with BO to also satisfy the desideratum of strong final performance in BOHB. We also describe how to extend BOHB to make effective use of parallel resources.

In the design of BOHB's BO component, on top of the five desiderata above, we also followed two additional ones:

6. Simplicity. Simplicity is a virtue, since simple approaches can be easily verified, have less components that can break, and can be easily reimplemented in different frameworks. HB is very simple, but standard GP-BO methods are not: they tend to require complex approximations, complex MCMC sampling over hyperparameters, and for good performance also data-dependent choices of kernel functions and hyperpriors.

7. Computational efficiency. Since our HB component allows us to carry out many function evaluations at small budgets, the cubic complexity of standard GPs, and even the lower complexity of approximate GPs would become problematic. Furthermore, compared to these cheap function evaluations, the complexity of computing sophisticated acquisition functions may also become a bottleneck, especially when parallelization effectively reduces the cost of function evaluations.

For these reasons, along with the reasons of scalability, robustness & flexibility, we based BOHB's BO component on the simple TPE method discussed above. As reliable GP-based BO methods become available that satisfy all the desiderata above, it would be easy to replace TPE with them.

4.1. Algorithm description

BOHB relies on HB to determine how many configurations to evaluate with which budget, but it replaces the random selection of configurations at the beginning of each HB iteration by a model-based search. Once the desired number of configurations for the iteration is reached, the standard successive halving procedure is carried out using these configurations. We keep track of the performance of all function evaluations $g(\boldsymbol{x}, b) + \epsilon$ of configurations $\boldsymbol{x}$ on all budgets b to use as a basis for our models in later iterations.

We follow HB's way of choosing the budgets and continue to use SH, but we replace the random sampling by a BO component to guide the search. We construct a model and use BO to select a new configuration, based on the configurations evaluated so far. In the remainder of this section, we will explain this procedure summarized by the pseudocode in Algorithm 2.

The BO part of BOHB closely resembles TPE, with one major difference: we opted for a single multidimensional KDE compared to the hierarchy of one-dimensional KDEs used in TPE in order to better handle interaction effects in the input space. To fit useful KDEs (in line 4 of Algorithm 2), we require a minimum number of data points N_{min}; this is set to $d+1$ for our experiments, where d is the number of hyperparameters. To build a model as early as possible, we do not wait until $N_b = |D_b|$, the number of observations for budget b, is large enough to satisfy $q \cdot N_b \geq N_{min}$. Instead, after initializing with $N_{min}+2$ random configurations (line 3), we choose the

$$\begin{aligned} N_{b,l} &= \max(N_{min}, q \cdot N_b) \\ N_{b,g} &= \max(N_{min}, N_b - N_{b,l}) \end{aligned} \tag{3}$$

best and worst configurations, respectively, to model the two densities. This ensures that both models have enough datapoints and have the least overlap when only a limited number of observations is available. We used the KDE implementation from statsmodels (Seabold & Perktold, 2010), estimating the KDE's bandwidth with the default estimation procedure (Scott's rule of thumb), which is efficient and performed well in our experience. Details on our KDE are given in Appendix D.

As the optimization progresses, more configurations are evaluated on bigger budgets. Given that the goal is to optimize on the largest budget, BOHB always uses the model for the largest budget for which enough observations are available (line 2). This enables it to overcome wrong conclusions drawn on smaller budgets by eventually relying on results with the highest fidelity only.

To optimize EI (lines 5-6), we sample N_s points from $l'(\boldsymbol{x})$, which is the same KDE as $l(\boldsymbol{x})$ but with all bandwidths multiplied by a factor b_w to encourage more exploration around the promising configurations. We observed that this improves convergence especially in the late stages of the optimization, when the model on the biggest budget is queried frequently but updated rarely.

In order to keep the theoretical guarantees of HB, we also sample a constant fraction ρ of the configurations uniformly at random (line 1). Besides global exploration, this guarantees that after $m \cdot (s_{max}+1)$ SH runs, our method has (on average) evaluated $\rho \cdot m \cdot (s_{max}+1)$ random configurations on b_{max}. As every SH run consumes a budget of at most $(s_{max}+1) \cdot b_{max}$, in the same time random search

evaluates $(\rho^{-1} \cdot (s_{max}+1))$-times as many configuration on the largest budget. This means, that in the worst case (when the lower fidelities are misleading), BOHB is at most this factor times slower than RS, but it is still guaranteed to converge eventually. The same argument holds for HB, but in practice both HB and BOHB substantially outperform RS in our experiments.

No optimizer is free of hyperparameters itself, and their effects have to be studied carefully. We therefore include a detailed empirical analysis of BOHB's hyperparameters in Appendix G that shows each hyperparameter's effect when all others are fixed to their default values (these are also listed there). We find that BOHB is quite insensitive to its hyperparameters, with the default working robustly across different scenarios.

4.2. Parallelization

Modern optimizers must be able to take advantage of parallel resources effectively and efficiently. BOHB achieves that by inheriting properties from both TPE and HB. The parallelism in TPE is achieved by limiting the number of samples to optimize EI, purposefully not optimizing it fully to obtain diversity. This ensures that consecutive suggestions by the model are diverse enough to yield near-linear speedups when evaluationed in parallel. On the other hand, HB can be parallelized by (a) starting different iterations at the same time (a parallel for loop in Alg. 1), and (b) evaluating configurations concurrently within each SH run.

Our parallelization strategy of BOHB is as follows. We start with the first SH run that sequential HB would perform (the most aggressive one, starting from the lowest budget), sampling configurations with the strategy outlined in Algorithm 2 until either (a) all workers are busy, or (b) enough configurations have been sampled for this SH run. In case (a), we simply wait for a worker to free up and then sample a new configuration. In case (b), we start the next SH run in parallel, sampling the configurations to run for it also according to Algorithm 2; observations D (and therefore the resulting models) are shared across all SH runs. BOHB is an anytime algorithm that at each point in time keeps track of the configuration that achieved the best validation performance; it can also be given a maximum budget of SH runs.

Algorithm 2: Pseudocode for sampling in BOHB

input : observations D, fraction of random runs ρ, percentile q, number of samples N_s, minimum number of points N_{min} to build a model, and bandwidth factor b_w
output : next configuration to evaluate

if *rand()* $< \rho$ **then return** random configuration
$b = \arg\max \{D_b : |D_b| \geq N_{min} + 2\}$
if $b = \emptyset$ **then return** random configuration
fit KDEs according to Eqs. (2) and (3)
draw N_s samples according to $l'(\boldsymbol{x})$ (see text)
return sample with highest ratio $l(\boldsymbol{x})/g(\boldsymbol{x})$

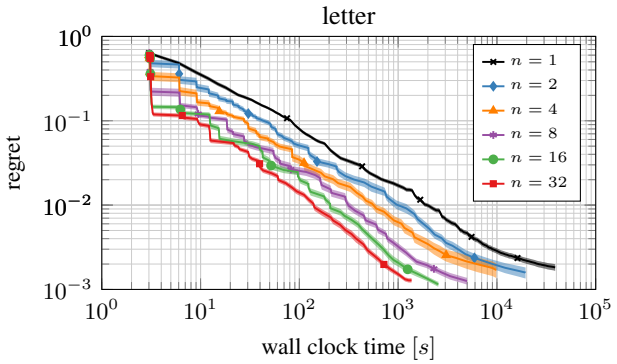

Figure 2. Performance of our method with different number of parallel workers on the letter surrogate benchmark (see Sec. 5) for 128 iterations. The speedup for two and four workers is close to linear, for more workers it becomes sublinear. For example, the speedup to achieve a regret of 10^{-2} for one vs. 32 workers is ca. $2000s/130s \approx 15$. We plot the mean and twice the standard error of the mean over 128 runs.

We note that SH has also been parallelized in (so far unpublished) independent work (Li et al., 2018). Next to parallelizing SH runs (by filling the next free worker with the ready-to-be-executed run with the largest budget), that work mentioned that HB can trivially be parallelized by running its SH runs in parallel. In contrast to this approach of parallelizing HB by having separate pools of workers for each SH run, we rather join all workers into a single pool, and whenever a worker becomes available preferentially execute waiting runs with smaller budgets. New SH runs are only started when the SH runs currently executed are not waiting for a worker to free up. This strategy (a) allows us to achieve better speedups by using all workers in the most aggressive (and often most effective) bracket first, and (b) also takes full advantage of models built on smaller budgets. Figure 2 demonstrates that our method of parallelization can effectively exploit many parallel workers.

5. Experiments

We now comprehensively evaluate BOHB's empirical performance in a wide range of tasks, including a high-dimensional toy function, as well as optimizing the hyperparameters of support vector machines, feed-forward neural networks, Bayesian neural networks, deep reinforcement learning agents and convolutional neural networks. Code for BOHB and our benchmarks is publicly available at https://github.com/automl/HpBandSter

To compare against TPE, we used the Hyperopt package

(Bergstra et al., 2011), and for all GP-BO methods we used the RoBO python package (Klein et al., 2017b). In all experiments we set $\eta = 3$ for HB and BOHB as recommended by Li et al. (2017). If not stated otherwise, for all methods we report the mean performance and the standard error of the mean of the best observed configuration so far (incumbent) at a given budget.

5.1. Artificial Toy Function: Stochastic Counting Ones

In this experiment we investigated BOHB's behavior in high-dimensional mixed continuous / categorical configuration spaces. Since GP-BO methods do not work well on such configuration spaces (Eggensperger et al., 2013) we do not include them in this experiment. However, we do use SMAC (Hutter et al., 2011), since its random forest is known to perform well in high-dimensional categorical spaces (Eggensperger et al., 2013).

Given a set of N_{cat} categorical variables $x_i \in \{0, 1\}$ and N_{cont} continuous variables $x_j \in [0, 1]$, we defined a variant of the counting ones problem as follows: We sum the values of all categoricals x_i and add the sample means of Bernoulli distributions with parameters x_j given by the continuous variables. The number of samples used to estimate the mean represents the budget. Figure 3 shows the result of 512 independent runs of various optimizers in a 16-dimensional space with $N_{cat} = N_{cont} = 8$ parameters. We plot the normalized immediate regret of the noise free function, i.e. $|f(\boldsymbol{x}_{inc}) - d|/d$ where $d = N_{cat} + N_{cont}$ and $\boldsymbol{x}_{inc}$ is the incumbent at a specific time step.

Random search worked very poorly on this benchmark and was quickly dominated by SMAC and TPE. Even though HB worked better in the beginning, SMAC and TPE clearly outperformed it after having obtained a sufficiently informative model. BOHB worked as well as HB in the beginning and then quickly started to perform better.

We obtained similar results for other dimensionalities (see Figures 8 and 9 in the supplementary material), but the picture is not always as clear. In higher dimensions, SMAC seems to outperform TPE, hinting at the limitations of TPE's KDE compared to SMAC's random forest. As BOHB still evaluates configurations on small budgets even in late stages of the optimization, convergence can be slowed down compared to SMAC and TPE. A formal description of the problem, the budgets, and a more detailed discussion of the results can be found in Appendix H.

5.2. Comprehensive Experiments on Surrogate Benchmarks

For the next experiments we constructed a set of surrogate benchmarks based on offline data following Eggensperger et al. (2015). Optimizing a surrogate instead of the real objective function is substantially cheaper, which allows us to afford many independent runs for each optimizer and to draw statistically more meaningful conclusions. A more detailed discussion of how we generated these surrogates can be found in Appendix I in the supplementary material. To better compare the convergence towards the true optimum, we again computed the immediate regret of the incumbent.

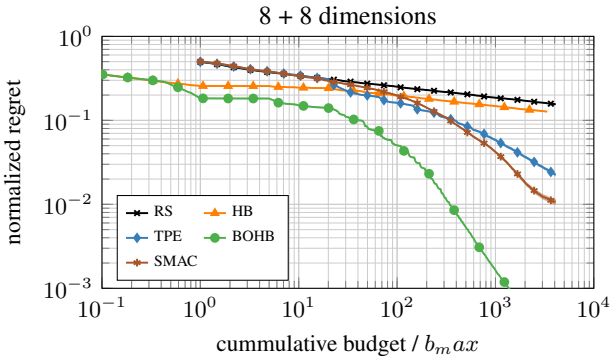

Figure 3. Results for the counting ones problem in 16 dimensional space with 8 categorical and 8 continuous hyperparameters. In higher dimensional spaces RS-based methods need exponentially more samples to find good solutions.

5.2.1. Support Vector Machine on MNIST

To compare against GP-BO, we used the support vector machine on MNIST surrogate from Klein et al. (2017a). This surrogate imitates the hyperparameter optimization of a support vector machine with a RBF kernel with two hyperparameters: the regularization parameter C and the kernel parameter γ. The budget is given by the number of training datapoints, where the minimum budget is $1/512$ of the training data and the maximum budget is the full training data. For further details, we refer to Klein et al. (2017a).

Figure 4 compares BOHB to various BO methods, such as Fabolas (Klein et al., 2017a), multi-task Bayesian optimization (MTBO) (Swersky et al., 2013), GP-BO with expected improvement (Snoek et al., 2012; Klein et al., 2017b), RS and HB. We follow the evaluation protocol of Klein et al. (2017a) and plot the performance of each configuration when retrained using the full dataset. Both BOHB and HB identified the best configuration within their first iterations, making them competitive to Fabolas and MTBO. We note that this is despite the fact that GP-BO methods usually work particularly well on such low-dimensional continuous problems (Eggensperger et al., 2013). A more detailed discussion of the results can be found in Appendix I.

5.2.2. Feed-forward Neural Networks on OpenML Datasets

We optimized six hyperparameters that control the training procedure (initial learning rate, batch size, dropout, exponential decay factor for learning rate) and the architecture (num-

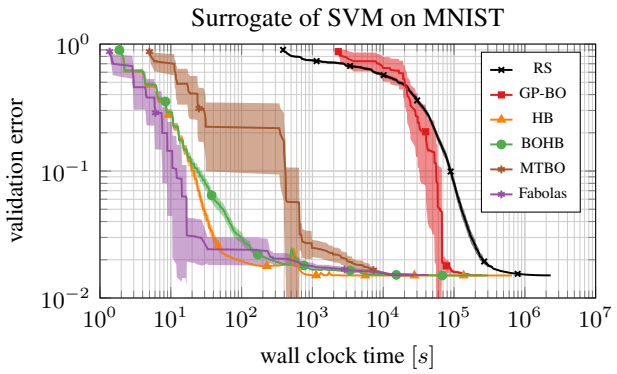

Figure 4. Comparison on the SVM on MNIST surrogates as described in Klein et al. (2017a). BOHB and HB work comparably to Fabolas on this benchmark outperforming MTBO and GP-BO.

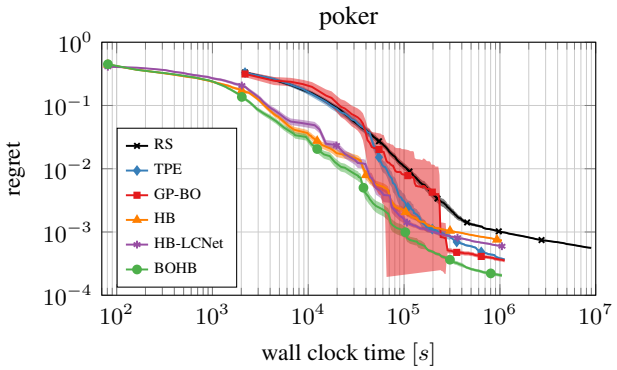

Figure 5. Optimizing six hyperparameter of a feed-forward neural network on featurized datasets; results are based on surrogate benchmarks. Results for the other 5 datasets are qualitatively similar and are shown in Figure 2 in the supplementary material.

ber of layers, units per layer) of a feed forward neural network for six different datasets gathered from OpenML (Vanschoren et al., 2014): Adult (Kohavi, 1996), Higgs (Baldi et al., 2014), Letter (Frey & Slate, 1991), MNIST (LeCun et al., 2001), Optdigits (Dheeru & Karra Taniskidou, 2017), and Poker (Cattral et al., 2002). A detailed description of all hyperparameter ranges and training budgets can be found in Appendix I.

We ran random search (RS), TPE, HB, GP-BO, Hyperband with LC-Net (HB-LCNet, see Klein et al. (2017c)) and BOHB on all six datasets and summarize the results for one of them in Figure 5. Figures for the other datasets are shown in Appendix E.

We note that HB initially performed much better than the vanilla BO methods and achieved a roughly three-fold speedup over RS. However, for large enough budgets TPE and GP-BO caught up in all cases, and in the end found better configurations than HB and RS. HB and BOHB started out identically, but BOHB achieved the same final performance as HB 100 times faster, while at the same time yielding a final result that was better than that of the other BO methods. All model-based methods substantially outperformed RS at the end of their budget, whereas HB approached the same performance. Interestingly, the speedups that TPE and GP-BO achieved over RS are comparable to the speedups that BOHB achieved over HB. Finally, HB-LCNet performed somewhat better than HB alone, but consistently worse than BOHB, even when tuning HB-LCNet. We only compare to HB-LCNet on this benchmark, since it is the only one that includes full learning curves (for which the parametric functions in HB-LCNet were designed). Also, HB-LCNet requires access to performance values for all budgets, which we do not obtain when, e.g., using data subset sizes as a budget, and we thus expect HB-LCNet to perform poorly in the other cases.

5.3. Bayesian Neural Networks

For this experiment we optimized the hyperparameters and the architecture of a two-layer fully connected Bayesian neural network trained with Markov Chain Monte-Carlo (MCMC) sampling. We used stochastic gradient Hamiltonian Monte-Carlo sampling (SGHMC) (Chen et al., 2014) with scale adaption (Springenberg et al., 2016) to sample the parameter vector of the network. Note that to the best of our knowledge this is the first application of hyperparameter optimization for Bayesian neural networks.

As tunable hyperparameters, we exposed the step length, the length of the burn-in period, the number of units in each layer, and the decay parameter of the momentum variable. A detailed description of the configuration space can be found in Appendix J. We used the Bayesian neural network implementation provided in the RoBO python package (Klein et al., 2017b) as described by Springenberg et al. (2016).

We considered two UCI (Dheeru & Karra Taniskidou, 2017) regression datasets, *Boston housing* and *protein structure* as described by Hernández-Lobato & Adams (2015) and report the negative log-likelihood of the validation data. For BOHB and HB, we set the minimum budget to 500 MCMC steps and the maximum budget to 10000 steps. RS and TPE evaluated each configuration on the maximum budget. For each hyperparameter optimization method, we performed 50 independent runs to obtain statistically significant results.

As Figure 6 shows, HB initially performed better than TPE, but TPE caught up given enough time. BOHB converged faster than both HB and TPE and even found a better configuration than the baselines on the Boston housing dataset.

5.4. Reinforcement Learning

Next, we optimized eight hyperparameters of proximal policy optimization (PPO) (Schulman et al., 2017) to learn the *cartpole swing-up* task. For PPO, we used the implementation from the TensorForce framework developed by Schaarschmidt et al. (2017) and we used the implementation

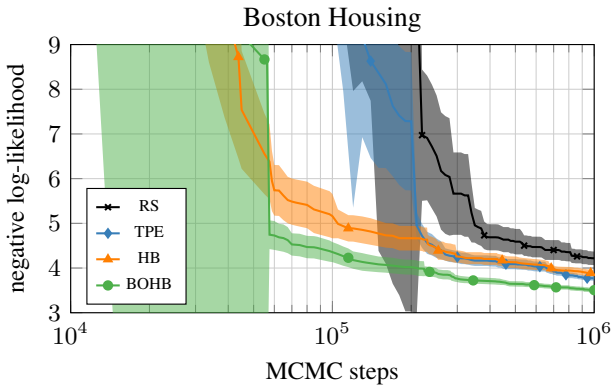

Figure 6. Optimization of 5 hyperparameters of a Bayesian neural network trained with SGHMC. BOHB quickly outperforms both TPE and HB.

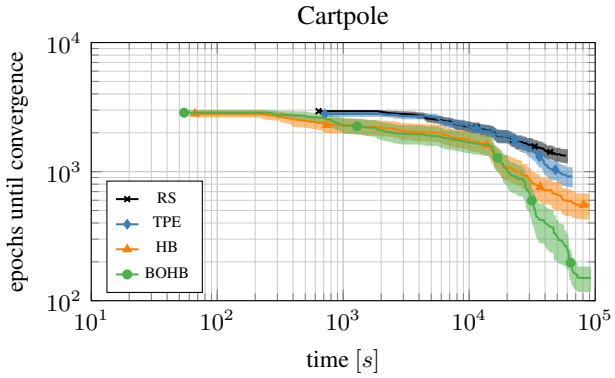

Figure 7. Hyperparameter optimization of 8 hyperparameters of PPO on the cartpole task. BOHB starts as well as HB but converges to a much better configuration.

from OpenAI Gym (Brockman et al., 2016) for the cartpole environment. The configuration space for this experiment can be found in Appendix K.

To find a configuration that not only converges quickly but also works robustly, for each function evaluation we ran a configuration for nine individual trials with a different seed for the random number generator. We returned the average number of episodes until PPO has converged to the optimum, defining convergence to mean that the reinforcement learning agent achieved the highest possible reward for 20 consecutive episodes. For each hyperparameter configuration we stopped training after the agent has either converged or ran for a maximum of 3000 episodes. The minimum budget for BOHB and HB was one trial and the maximum budget were nine trials, and all other methods used a fixed number of nine trials. As in the previous benchmark, for each hyperparameter optimization method we performed 50 independent runs.

Figure 7 shows that HB and BOHB worked equally well in the beginning, but BOHB converged to better configurations in the end. Apparently, the budget for this benchmark was not sufficient for TPE to find the same configuration.

5.5. Convolutional Neural Networks on CIFAR-10

For a final evaluation, we optimized the hyperparameters of a medium-sized residual network (depth 20 and basewidth of 64; roughly 8.5M parameters) with Shake-Shake (Gastaldi, 2017) and Cutout (DeVries & Taylor, 2017) regularization. To perform hyperparameter optimization, we split off 5 000 training images as a validation set. As hyperparameters, we optimized learning rate, momentum, weight decay, and batch size.

We ran BOHB with budgets of 22, 66, 200, and 600 epochs, using 19 parallel workers. Each worker used 2 NVIDIA TI 1080 GPUs for parallel training, which resulted in runs with the longest budget taking approximately 7 hours (on 2 GPUs). The complete BOHB run of 16 iterations required a total of 33 GPU days (corresponding to a cost of less than 3 full function evaluations on each of the 19 workers) and achieved a test error of $2.78\% \pm 0.09\%$ (which is better than the error Gastaldi (2017) obtained with a slightly larger network). While we note that the performance numbers from different papers are not directly comparable due to the use of different optimization and regularization approaches, it is still instructive to compare this result to others in the literature. Our result is better than that of last year's state-of-the-art neural architecture search by reinforcement learning (3.65% (Zoph & Le, 2017)) and the recent paper on progressive neural architecture search (3.41% (Liu et al., 2017)), but it does not quite reach the state-of-the-art performance of 2.4% and 2.1% reported in recent arXiv papers on reinforcement learning (Zoph et al., 2017) and evolutionary search (Real et al., 2018). However, since these approaches used 60 to 95 times more compute resources (2 000 and 3 150 GPU days, respectively!), as well as networks with 3-4 times more parameters, we believe that our results are a strong indication of the practical usefulness of BOHB for resource-constrained optimization.

6. Conclusions

We introduced BOHB, a simple yet effective method for hyperparameter optimization satisfying the desiderata outlined above: it is robust, flexible, scalable (to both high dimensions and parallel resources), and achieves both strong anytime performance and strong final performance. We thoroughly evaluated its performance on a diverse set of benchmarks and demonstrated its improved performance compared to a wide range of other state-of-the-art approaches. Our easy-to-use open-source implementation (available under https://github.com/automl/HpBandSter) should allow the community to effectively use our method on new problems. To further improve BOHB, we will consider an automatic adaptation of the budgets used to alleviate the problem of misspecification by the user while maintaining the versatility and robustness of the current version.

Acknowledgements

We thank Ilya Loshchilov for suggesting to track the best hyperparameter setting across different budgets (already in late 2015), which influenced our thoughts about the problem and ultimately the development of BOHB. This work has partly been supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant no. 716721, by the European Commission under grant no. H2020-ICT-645403-ROBDREAM, and by the German Research Foundation (DFG) under Priority Programme Autonomous Learning (SPP 1527, grant BR 3815/8-1 and HU 1900/3-1) Furthermore, the authors acknowledge support by the state of Baden-Württemberg through bwHPC and the DFG through grant no INST 39/963-1 FUGG.

References

Baldi, P., Sadowski, P., and Whiteson, D. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5, 2014.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, 2011.

Bergstra, J., Yamins, D., and Cox, D. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *International Conference on Machine Learning*, 2014.

Bertrand, H., Ardon, R., Perrot, M., and Bloch, I. Hyperparameter optimization of deep neural networks: Combining hyperband with Bayesian model selection. *Conférence sur l'Apprentissage Automatique*, 2017.

Biedenkapp, A., Lindauer, M., Eggensperger, K., Fawcett, C., Hoos, H., and Hutter, F. Efficient parameter importance analysis via ablation with surrogates. In *National Conference on Artificial Intelligence*, 2017.

Brochu, E., Cora, V., and de Freitas, N. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. `arXiv:1012.2599 [cs.LG]`, 2010.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. `arXiv:1606.01540 [cs.LG]`, 2016.

Cattral, R., Oppacher, F., and Deugo, D. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications*, 1(1), 2002.

Chen, T., Fox, E., and Guestrin, C. Stochastic gradient Hamiltonian Monte Carlo. In Xing, E. and Jebara, T. (eds.), *International Conference on Machine Learning*, 2014.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. `arXiv:1708.04552 [cs.CV]`, 2017.

Dheeru, D. and Karra Taniskidou, E. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Eggensperger, K., Feurer, M., Hutter, F., Bergstra, J., Snoek, J., Hoos, H., and Leyton-Brown, K. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice*, 2013.

Eggensperger, K., Hutter, F., Hoos, H., and Leyton-Brown, K. Efficient benchmarking of hyperparameter optimizers via surrogates. In *National Conference on Artificial Intelligence*, 2015.

Feurer, M., Springenberg, T., and Hutter, F. Initializing Bayesian hyperparameter optimization via meta-learning. In *National Conference on Artificial Intelligence*, 2015.

Frey, P. W. and Slate, D. J. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2), 1991.

Gastaldi, X. Shake-shake regularization. `arXiv:1705.07485 [cs.LG]`, 2017.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *International Conference on Knowledge Discovery and Data Mining*, 2017.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. Deep reinforcement learning that matters. `arXiv:1709.06560 [cs.LG]`, 2017.

Hernández-Lobato, J. and Adams, R. Probabilistic backpropagation for scalable learning of Bayesian neural networks. In *International Conference on Machine Learning*, volume 37, 2015.

Hutter, F., Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, volume 6683, 2011.

Hutter, F., Hoos, H., and Leyton-Brown, K. An efficient approach for assessing hyperparameter importance. In *International Conference on Machine Learning*, 2014.

Jamieson, K. and Talwalkar, A. Non-stochastic best arm identification and hyperparameter optimization. In *Conference on Artificial Intelligence and Statistics*, 2016.

Kandasamy, K., Dasarathy, G., Schneider, J., and Póczos, B. Multi-fidelity bayesian optimisation with continuous approximations. In *International Conference on Machine Learning*, 2017.

Klein, A., Falkner, S., Bartels, S., Hennig, P., and Hutter, F. Fast Bayesian hyperparameter optimization on large datasets. *Electronic Journal of Statistics*, 11(2), 2017a.

Klein, A., Falkner, S., Mansur, N., and Hutter, F. Robo: A flexible and robust bayesian optimization framework in python. In *NIPS 2017 Bayesian Optimization Workshop*, 2017b.

Klein, A., Falkner, S., Springenberg, J. T., and Hutter, F. Learning curve prediction with Bayesian neural networks. In *International Conference on Learning Representations*, 2017c.

Kohavi, R. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, 1996.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In Haykin, S. and Kosko, B. (eds.), *Intelligent Signal Processing*, pp. 306–351. IEEE Press, 2001.

Lévesque, J.-C., Gagné, C., and Sabourin, R. Bayesian hyperparameter optimization for ensemble learning. In *Uncertainty in Artificial Intelligence*, 2016.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *International Conference on Learning Representations*, 2017.

Li, L., Jamieson, K., Rostamizadeh, A., Gonina, K., Hardt, M., Recht, B., and Talwalkar, A. Massively parallel hyperparameter tuning, 2018. URL https://openreview.net/forum?id=S1Y7OOlRZ.

Liu, C., Zoph, B., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. Progressive neural architecture search. arXiv:1712.00559 [cs.CV], 2017.

Melis, G., Dyer, C., and Blunsom, P. On the state of the art of evaluation in neural language models. arXiv:1707.05589 [cs.CL], 2017.

Mendoza, H., Klein, A., Feurer, M., Springenberg, J., and Hutter, F. Towards automatically-tuned neural networks. In *ICML 2016 AutoML Workshop*, 2016.

Perrone, V., Jenatton, R., Seeger, M., and Archambeau, C. Multiple adaptive bayesian linear regression for scalable bayesian optimization with warm start. arXiv:1712.02902 [stat.ML], 2017.

Poloczek, M., Wang, J., and Frazier, P. I. Warm starting bayesian optimization. In *Winter Simulation Conference*, 2016.

Poloczek, M., Wang, J., and Frazier, P. Multi-information source optimization. In *Advances in Neural Information Processing Systems*, 2017.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized Evolution for Image Classifier Architecture Search. 2018.

Schaarschmidt, M., Kuhnle, A., and Fricke, K. Tensorforce: A tensorflow library for applied reinforcement learning. Web page, 2017. URL https://github.com/reinforceio/tensorforce.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. arXiv:1707.06347 [cs.LG], 2017.

Seabold, S. and Perktold, J. Statsmodels: Econometric and statistical modeling with python. In *Python in Science Conference*, 2010.

Shahriari, B., Swersky, K., Wang, Z., Adams, R., and de Freitas, N. Taking the human out of the loop: A review of Bayesian optimization. *IEEE*, 104(1), 2016.

Snoek, J., Larochelle, H., and Adams, R. P. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, and Adams, R. Scalable Bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, volume 37, 2015.

Springenberg, J., Klein, A., Falkner, S., and Hutter, F. Bayesian optimization with robust bayesian neural networks. In *Advances in Neural Information Processing Systems*, 2016.

Swersky, K., Snoek, J., and Adams, R. Multi-task Bayesian optimization. In *Advances in Neural Information Processing Systems*, 2013.

Swersky, K., Snoek, J., and Adams, R. Freeze-thaw bayesian optimization. arXiv:1406.3896 [stat.ML], 2014.

van Rijn, J. N. and Hutter, F. Hyperparameter importance across datasets. In *International Conference on Knowledge Discovery and Data Mining*, 2018.

Vanschoren, J., van Rijn, J., Bischl, B., and Torgo, L. OpenML: Networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2), 2014.

Wang, J., Xu, J., and Wang, X. Combination of hyperband and bayesian optimization for hyperparameter optimization in deep learning. arXiv:1801.01596 [cs.CV], 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. 2017.