Supplementary material for: BO-HB: Robust and Efficient Hyperparameter Optimization at Scale

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

A. Available Software

To promote reproducible science and enable other researchers to use our method, we provide an open-source implementation of BOHB and Hyperband. It is available under https://github.com/automl/HpBandSter. The benchmarks and our scripts used to produce the data shown in the paper can be found in the *icml_2018* branch.

B. Comparison to other Combinations of Bayesian optimization and Hyperband

Here we discuss the differences between our method and the related approaches of Bertrand et al. (2017) and Wang et al. (2018) in more detail. We note that these works are independent and concurrent; our work extends our preliminary short workshop papers at NIPS 2017 (Falkner et al., 2017) and ICLR 2018 (Falkner et al., 2018).

While the general idea of combining Hyperband and Bayesian optimization by Bertrand et al. (2017) is the same as in our work, they use a Gaussian process for modeling the performance. The budget is modeled like any other dimension of the search space, without any special treatment. Based on our experience with Fabolas (Klein et al., 2017), we expect that the squared exponential kernel might not extrapolate well, which would hinder performance. Also, the small evaluation provided by Bertrand et al. (2017) does not allow strong conclusions about the performance of their method.

Wang et al. (2018) also independently combined TPE and Hyperband, but in a slightly different way than we did. In their method, TPE is used as a subroutine in every iteration of Hyperband. In particular, a new model is built from scratch at the beginning of every SuccessiveHalving run (Algorithm 3, line 8 in Wang et al. (2018)). This means that in later iterations of the algorithm, the model does not benefit from any of the evaluations in previous iterations. In contrast, BOHB collects all evaluations on all budgets and uses the largest budget with enough evaluations (admittedly a heuristic, but we would argue a reasonable one) as a base for future evaluations. This way, BOHB aggregates more knowledge into its models for the different budgets as the optimization progresses. We believe this to be a crucial part of the strong performance of our method. Empirically, Wang et al. (2018) did not achieve the consistent and large speedups across a wide range of applications BOHB achieved in our experiments.

C. Successive Halving

SuccessiveHalving is a simple heuristic to allocate more resources to promising candidates. For completeness, we provide pseudo code for it in Algorithm 1. It is initialized with a set of configurations, a minimum and maximum budget, and a scaling parameter η. In the first *stage* all configurations are evaluated on the smallest budget (line 3). The losses are then sorted and only the best $1/\eta$ configurations are kept in the set C (line 4). For the following stage, the budget is increased by a factor of η (line 5). This is repeated until the maximum budget for a single configuration is reached (line 2). Within Hyperband, the budgets are chosen such that all SuccessiveHalving executions require a similar total budget.

Algorithm 1: Pseudocode for SuccessiveHalving used by Hyperband as a subroutine.

input : initial budget b_0, maximum budget b_{max}, set of n configurations $C = \{c_1, c_2, \dots c_n\}$

1 $b = b_0$
2 **while** $b \leq b_{max}$ **do**
3 $\quad L = \{\tilde{f}(c, b) : c \in C\}$
4 $\quad C = \text{top}_k(C, L, \lfloor |C|/\eta \rfloor)$
5 $\quad b = \eta \cdot b$

[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.

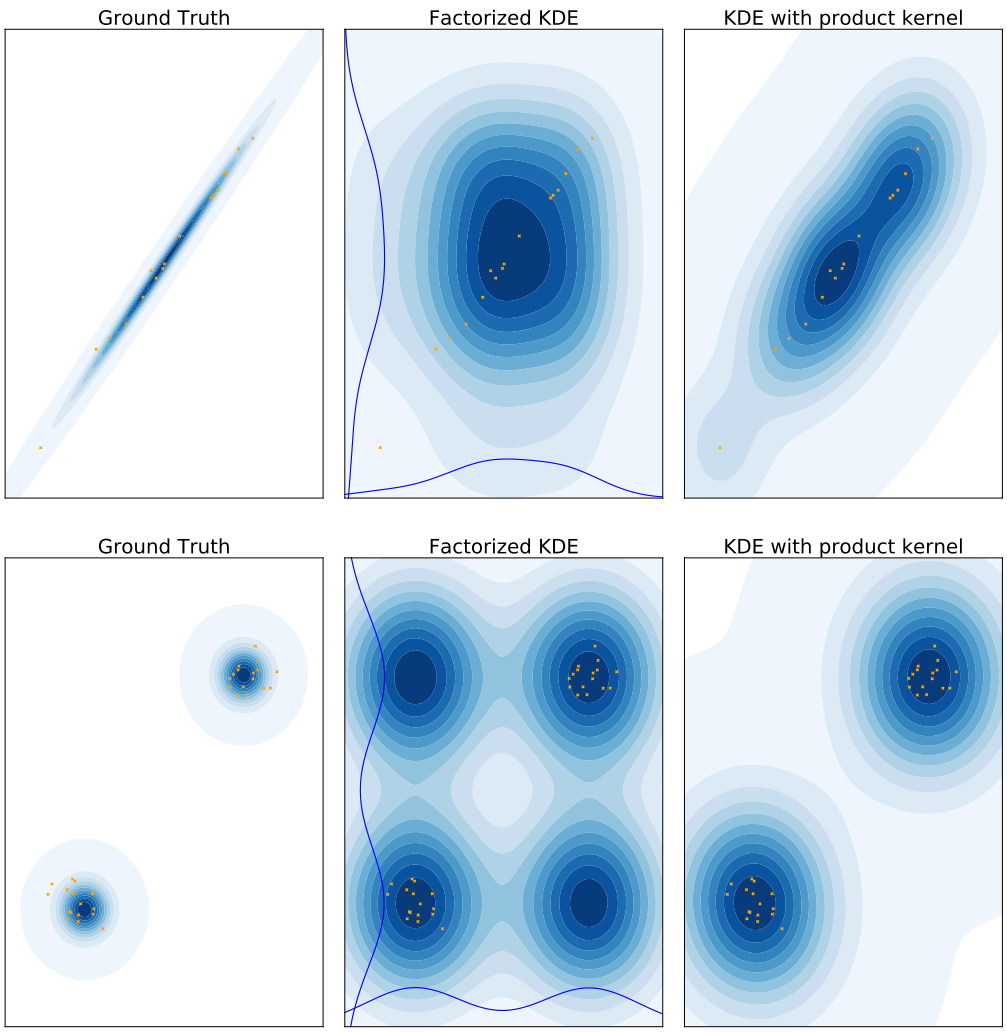

Figure 1. Visualization of the two different KDE approaches. The left column shows the true distribution (blue shaded area) from which 16 samples (orange crosses) were drawn. The middle column shows how a KDE that factorizes the PDF (as in TPE) models the density. The right column demonstrates how the KDE used in BOHB handles the data by factorizing the kernels instead of the PDF. The top row is a single two dimensional Gaussian probability with a strong correlation between the variables. The example in the bottom row is a mixture of two such Gaussians.

D. Details on the Kernel Density Estimator

We used the MultivariateKDE from the statsmodels package (Seabold & Perktold, 2010), which constructs a factorized kernel, with a one-dimensional kernel for each dimension. Note that using this product of 1-d kernels differs from the original TPE, which uses a pdf that is the product of 1-d pdfs. Figure 1 visualizes the differences for two small problems. For the continuous parameters a Gaussian kernel is used, whereas the Aitchison-Aitken kernel is the default for categorical parameters. We used Scott's rule for efficient bandwidth estimation, as preliminary experiments with maximum-likelihood based bandwidth selection did not yield better performance but caused a significant overhead.

E. Performance of all methods on all surrogates

Figure 2 shows the performance of all methods we evaluated on all our surrogate benchmarks. Random search is clearly the worst optimizer across all datasets when the budget is large enough for GP-BO and TPE to leverage their model. Hyperband and the two methods based on it (HB-LCNet) and BOHB improve much more quickly due to the smaller budgets used. On all surrogate benchmarks, BOHB starts to outperform HB after the first couple of iterations (sometimes even earlier, e.g., on dataset letter). The same dataset

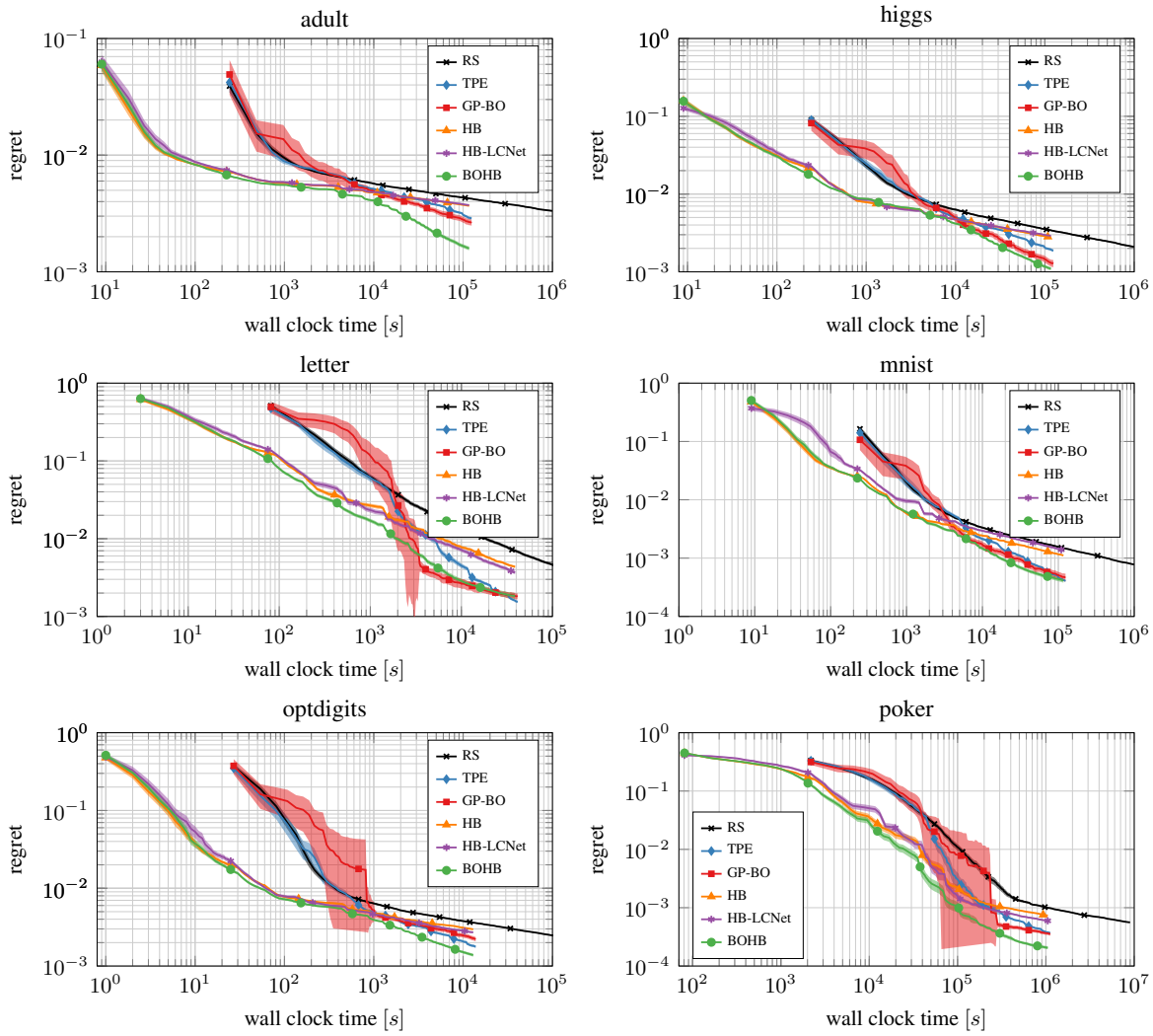

Figure 2. Mean performance on the surrogates for all six datasets. As uncertainties, we show the standard error of the mean based on 512 runs (except for GP-BO, which has only 50 runs).

also shows that traditional BO methods can still have an advantage for very large budgets, since in these late stages of the optimization process the low fidelity evaluations of BOHB can cause a constant overhead without any gain.

F. Performance of parallel runs

Figure 3 shows the performance of BOHB when run in parallel on all our surrogate benchmarks. The speed-ups are quite consistent, and almost linear for a small number of workers (2-8). For more workers, more random configurations are evaluated in parallel before the first model is built, which degrades performance. But even for 32 workers, linear speedups are possible (see, e.g., dataset letter, for reaching a regret of 2×10^{-3}).

We note that in order to carry out this evaluation of parallel performance, we actually simulated the parallel optimization by making each worker wait for the given budget before returning the corresponding performance value of our surrogate benchmark. (The case of one worker is an exception, where we can simply reconstruct the trajectory because all configurations are evaluated serially.) By using this approach in connection with threads, each evaluation of a parallel algorithm still only used 1 CPU, but the run actually ran in real time. For this reason, we decided to not evaluate all possible numbers of workers for dataset poker, for which each run with less than 16 workers would have taken more than a day, and we do not expect any different behavior compared to the other datasets.

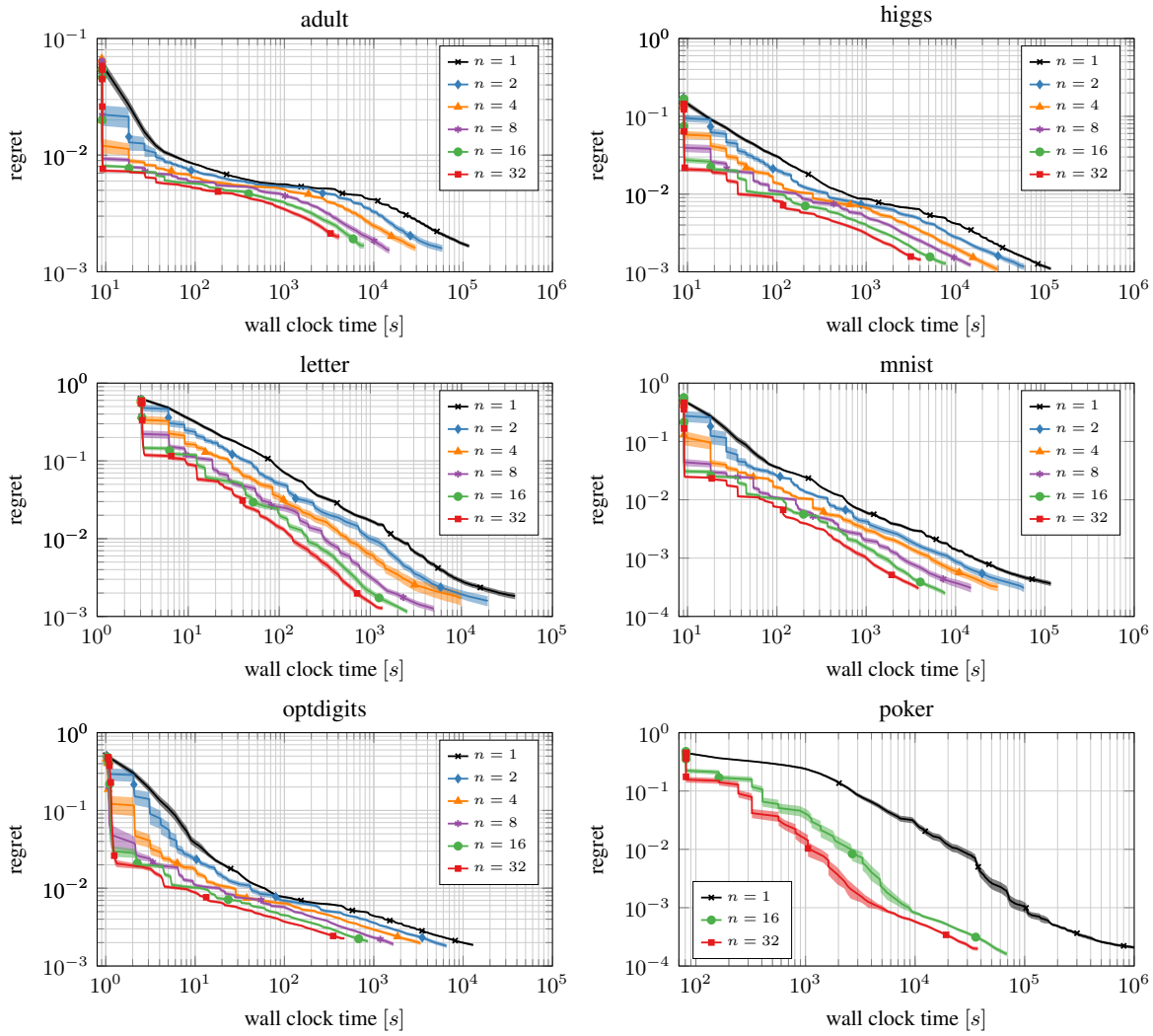

Figure 3. Mean performance on the surrogates for all six datasets with different numbers of workers n. As uncertainties, we show the standard error of the mean based on 128 runs. Because we simulated them in real time to capture the true behavior, poker is too expensive to evaluate with less than 16 workers within a day.

G. Evaluating the hyperparameters of BOHB

In this section, we evaluate the importance of the individual hyperparameters of BOHB, namely the number of samples used to optimize the acquisition function (Figure 4), the fraction of purely random configuration ρ (Figure 5), the scaling parameter η (Figure 6), and the bandwidth factor used to encourage exploration (Figure 7).

Additionally, we want to discuss the importance of η, b_{min} and b_{max} already present in HB. The parameter η controls how aggressively SH cuts down the budget and the number of configurations evaluated. Like HB (Li et al., 2017), BOHB is also quite insensitive to this choice in a reasonable range. For our experiments, we use the same default value ($\eta = 3$) for HB and BOHB.

More important for the optimization are b_{min} and b_{max}, which are problem specific and inputs to both HB and BOHB. While the maximum budget is often naturally defined, or is constrained by compute resources, the situation for the minimum budget is often different. To get substantial speedups, an evaluation with a budget of b_{min} should contain some information about the quality of a configuration with larger budgets; for example, when subsampling the data, the smallest subset should not be one datum, but rather enough points to fit a meaningful model. This requires knowledge about the benchmark and the algorithm being optimized.

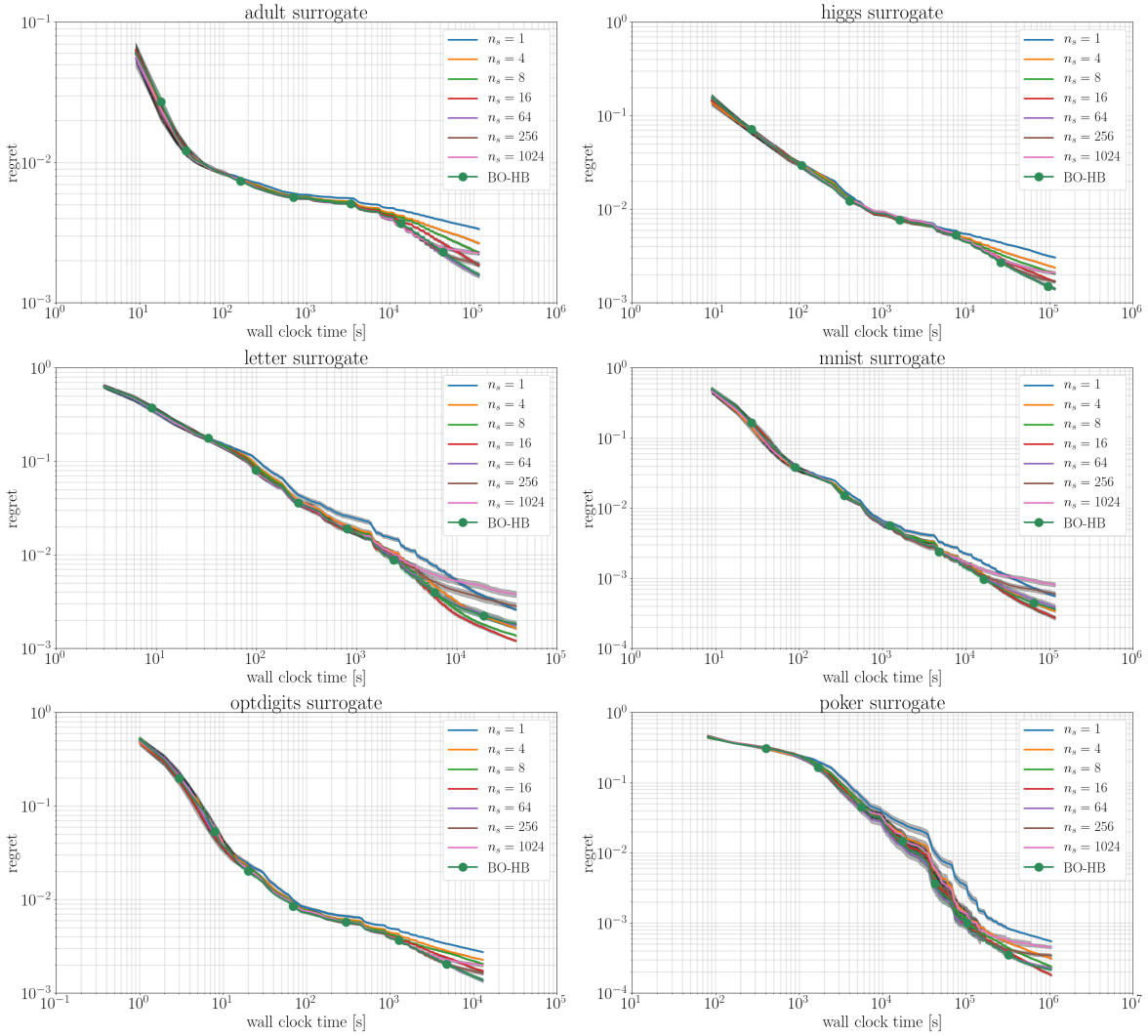

Figure 4. Performance on the surrogates for all six datasets for different number of samples

H. Stochastic Counting Ones

We now formally define the stochastic variant of the counting ones function we use and present the results for different dimensions. The objective function to be minimized can be written as

$$f(\boldsymbol{x}) = -\left(\sum_{x \in X_{cat}} x \;+\; \sum_{x \in X_{cont}} \mathbb{E}_b[(B_{p=x})]\right), \qquad (1)$$

where B_p is the Bernoulli distribution with parameter p, and $\mathbb{E}_b$ denotes the expectation estimated using b independent draws from the distribution.

The problem consists of a deterministic discrete part (the standard counting ones problem), and a stochastic component whose noise is controlled by the budget b. To keep the noise consistent across different dimensions, we chose the budgets such that the total number of samples used remains constant. Specifically, we picked $b_{min} = 576/d$ and $b_{max} = 93312/d$ where $d = N_{cat} + N_{cont}$. For $N_{cat} = N_{cont} = 4$ this results in a minimum of 144 and a maximum of 11664 samples for each Bernoulli distribution. BOHB and HB evaluated between these budgets, where as TPE and SMAC operated always with the maximum budget.

This function has some noteworthy properties:

1. By design, we know the best value, $-d$, and worst value, 0. For easier comparison between different numbers of dimensions, we plot the normalized regret, which in our case is $(f(\boldsymbol{x}) + d)/d$.

2. The optimum $\boldsymbol{x} = [1]^d$ is at the boundary of the search space. This could be problematic for both the random forests in SMAC, and the KDEs in TPE and BOHB.

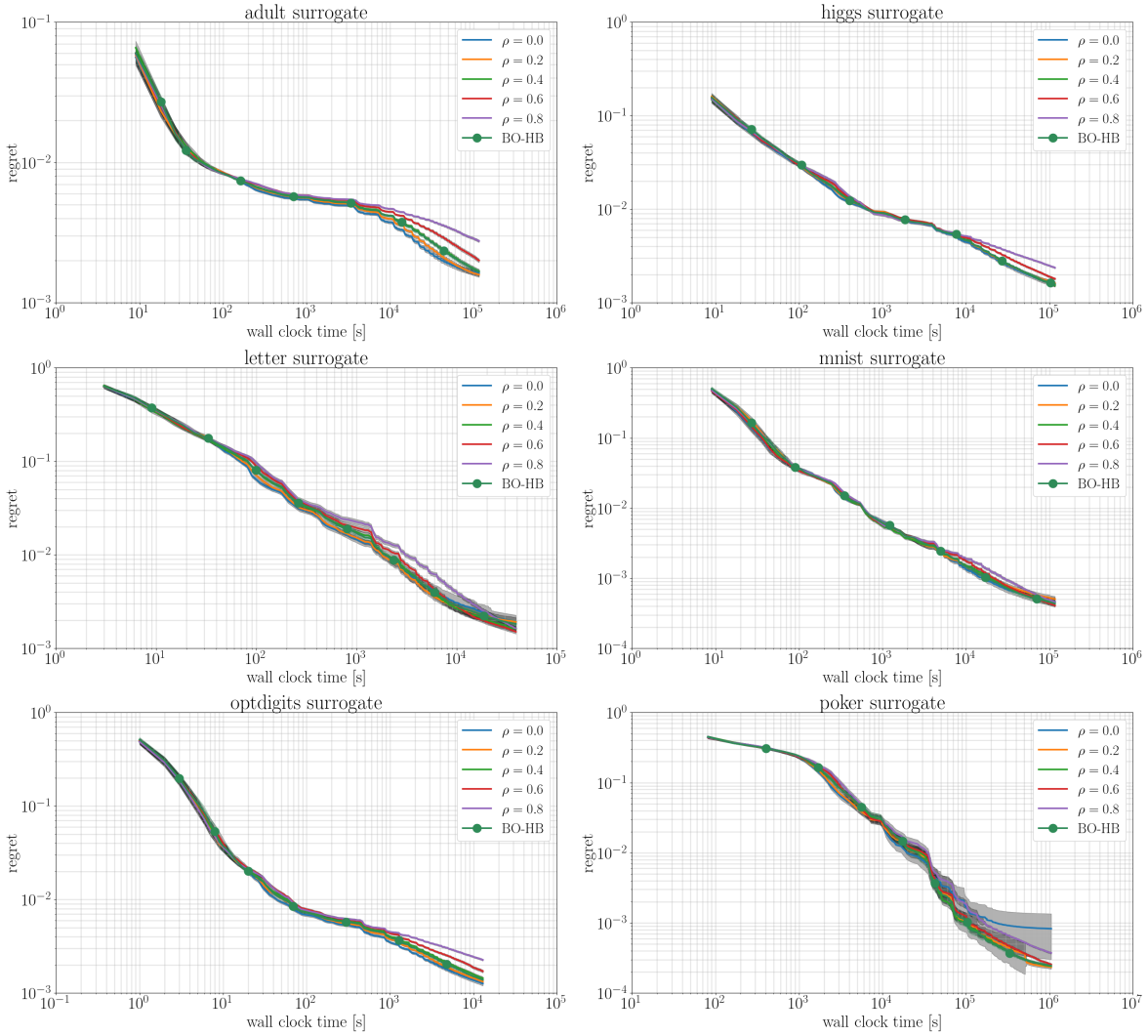

Figure 5. Performance on the surrogates for all six datasets for different random fractions

Figure 8 shows the mean performance of all applicable methods in $d = 8$, 16, 32 and 64 dimensions for a budget of 4000 full function evaluations. The median performances are shown in Figure 9.

We draw the following conclusions from the results:

1. Despite its simple definition, this problem is quite challenging for the methods we applied to it. RS and HB both suffer from the fact that drawing configurations at random performs quite poorly in this space. The model-based approaches SMAC and TPE performed substantially better, especially with large budgets. We would like to mention that SMAC and TPE treated the problem as a blackbox optimization problem; the results for SMAC could likely be improved further by treating individual samples as "instances" and using SMAC's intensification mechanism to reject poor configurations based on few samples and evaluate promising configurations with more samples.

2. BOHB struggles to converge for the eight dimensional example. The most probable reason is the number of samples required to build a model, which was set to 9 here. This led to a consistently slow convergence of BOHB. The median plots (Figure 9) show that there is quite some variability in BOHB's performance for this dimensionality. We attribute this at least in part to the small number of samples used to build the initial model; combined with the poor performance of random configurations, this leads to unstable performance. To investigate this further, we changed two parameters of BOHB: first, we increased the minimum number of points to build a model to 16, which gives the model a higher change to start from slightly better random configurations. Second, to increase exploration, we

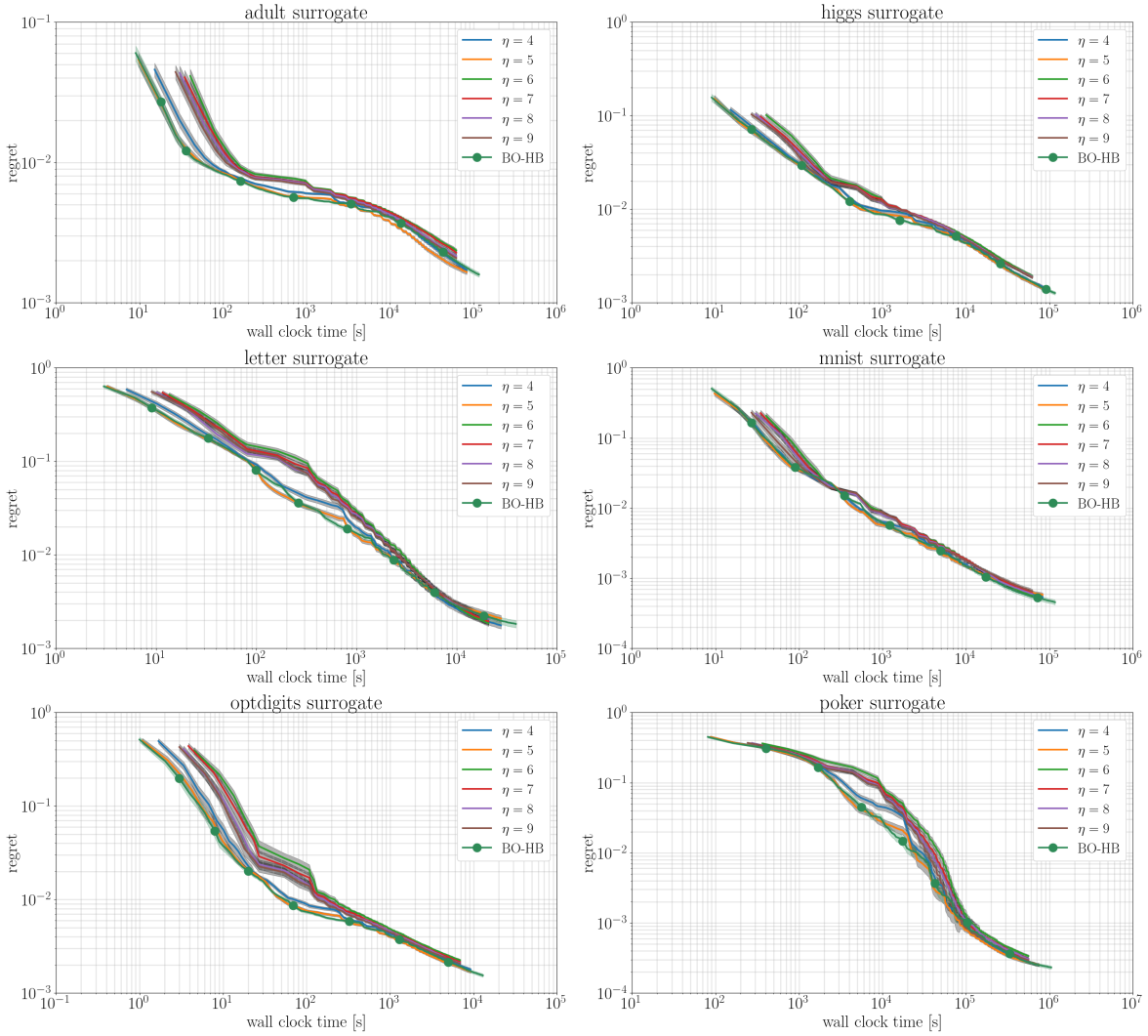

Figure 6. Performance on the surrogates for all six datasets for different values of η.

reduced the number of samples to optimize expected improvement to 16. This increases the chances of sampling a suboptimal (according to the model) point, which can help if the model is poor. The results in Figure 10 show that these simple changes restore BOHB's good performance, which means that BOHB might be robust, but there are still cases where an unlucky combination of budgets and noise in the system can lead to relatively weak convergence. We nevertheless would like to point out, that even this case, it still took TPE 200 full function evaluations to catch up with BOHB (a considerable number for an eight-dimensional problem).

3. Given a large enough budget, BOHB's evaluations on small budgets lead to a constant overhead over only using the more reliable evaluations on larger budgets. This slows down convergence.
4. Since the optimization problem is perfectly separable (there are no interaction effects between any dimensions), we expect TPE's univariate KDE to perform better than BOHB's multivariate one, which might also explain the relatively slow convergence compared to TPE towards the end of the trajectories.

I. Surrogates

I.1. Support Vector Machine

We now analyze the results for the SVM surrogate in more detail. Figure 11 shows the validation error of BOHB and HB when retraining the configurations using the whole training data, as well as the actual loss being optimized (labeled BOHB-loss and HB-loss), i.e. the validation error this configuration achieved being trained on a subset of thet data. Based on the validation error after retraining, HB seems

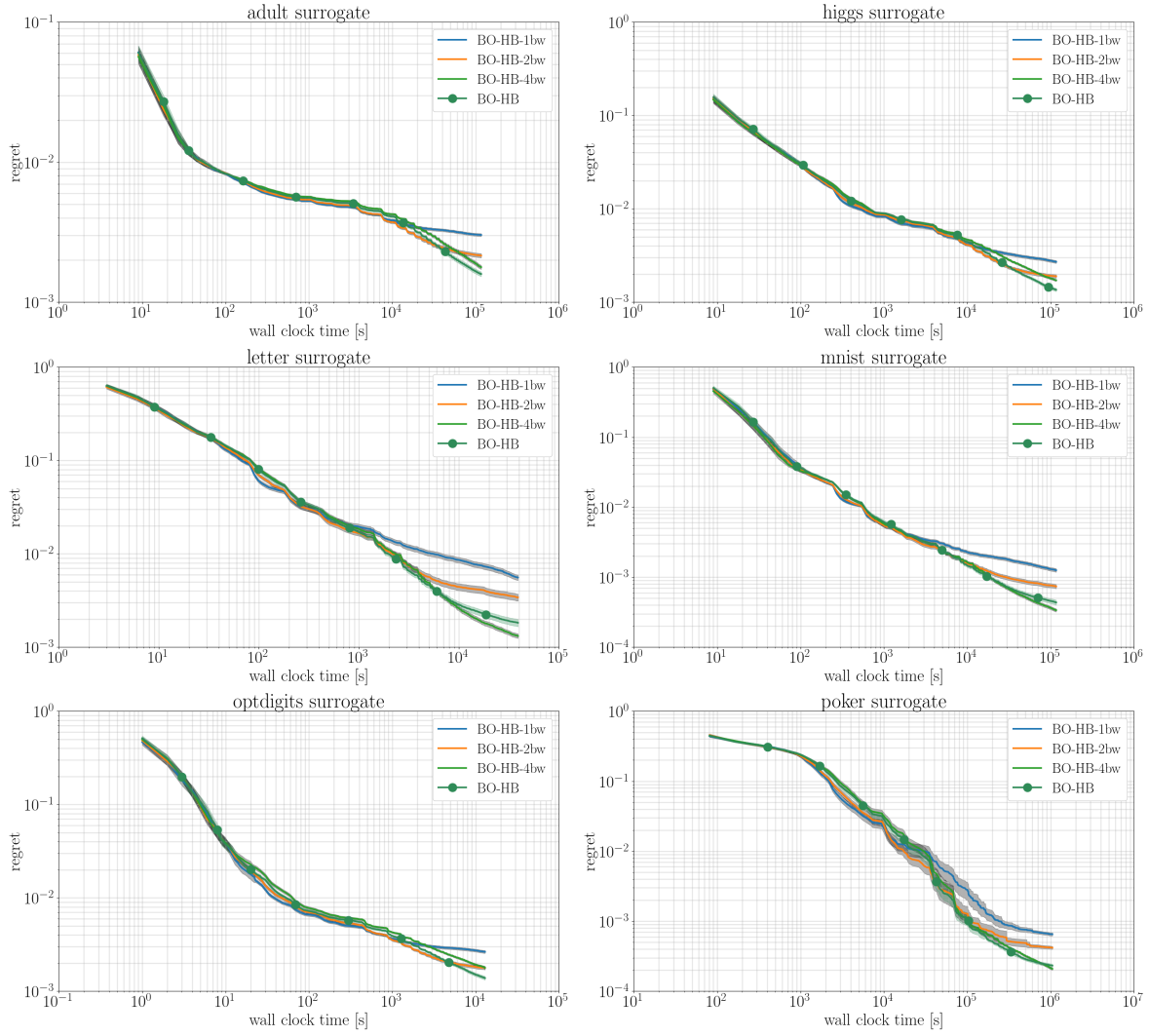

Figure 7. Performance on the surrogates for all six datasets for different bandwidth factors.

to perform slightly better than BOHB, but there is almost no difference when comparing the losses observed during optimization. One can see that the interquartile range of BOHB in the right panel is larger, indicating more variability. HB almost always found a configuration achieving the best error within its first iteration (at around 3000 seconds); BOHB also achieved this in more than half of its runs, but it required a second iteration (around 6000 seconds) in a substantial fraction of runs to find the truly best configuration. This effect can take place when the number of configurations sampled during the first iteration is large enough to fully optimize on the smallest budget. In this particular example, this is due to the low dimensionality of the benchmark (two parameters only) and the relatively large number of configurations that enter the first iteration of Successive Halving (243, given the budgets and η value).

I.2. Feed Forward Network Surrogates

I.2.1. Constructing the Surrogates

To build a surrogate, we sampled 10 000 random configurations for each dataset, trained them for 50 epochs, and recorded their classification error after each epoch, along with their total training time. We fitted two independent random forests that predict these two quantities as a function of the hyperparameter configuration used. This enabled us to predict the classification error as a function of time with sufficient accuracy. As almost all networks converged within the 50 epochs, we extend the curves by the last obtained value if the budget would allow for more epochs.

The surrogates enable cheap benchmarking, allowing us to run each algorithm 256 times. Since evaluating a configuration with the random forest is inexpensive, we used a global

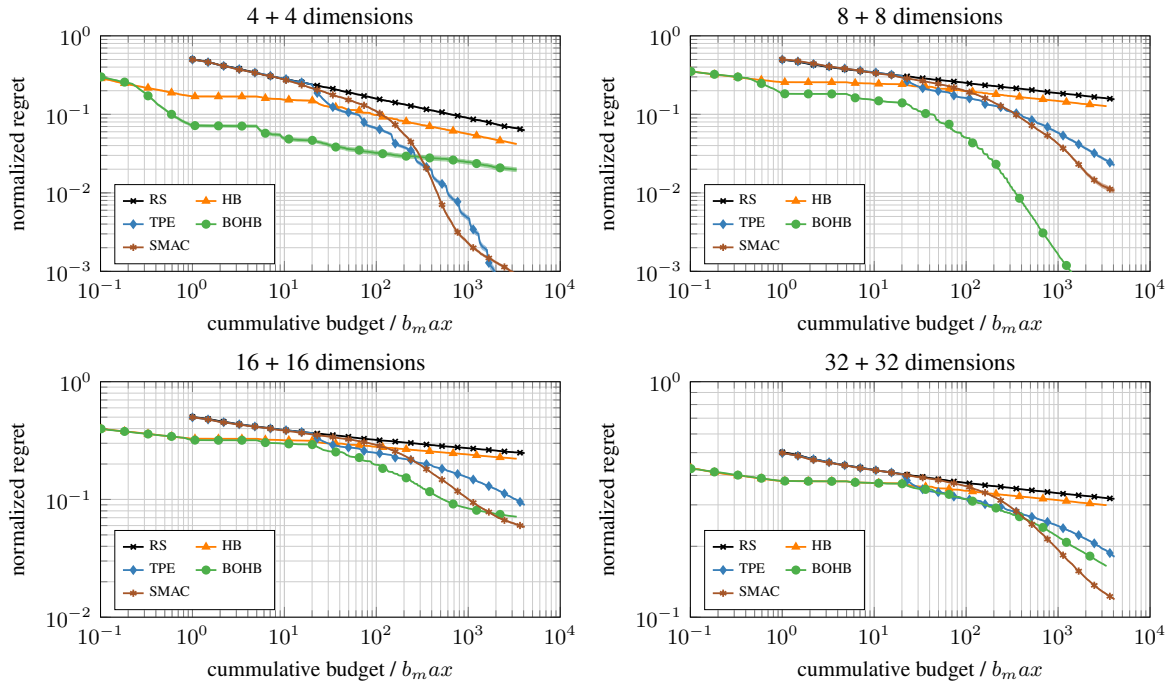

Figure 8. Mean performance of BOHB, HB, TPE, SMAC and RS on the mixed domain counting ones function with different dimensions. As uncertainties, we show the standard error of the mean based on 512 runs.

optimizer (differential evolution) to find the true optimum. We allowed the optimizer 10 000 iterations which should be sufficient to find the true optimum.

Besides these positive aspects of benchmarking with surrogates, there are also some drawbacks that we want to mention explicitly:

(a) There is no guarantee that the surrogate actually reflects the important properties of the true benchmark.

(b) The presented results show the optimized classification error on the validation set used during training. There is no test performance that could indicate overfitting.

(c) Training with stochastic gradient descent is an inherently noisy process, i.e. two evaluations of the same configuration can result in different performances. This is not at all reflected by our surrogates, making them a potentially easier to optimize than the true benchmark they are based on.

(d) By fixing the budgets (see below) and having deterministic surrogates, the global minima might be the result of some small fluctuations in the classification error in the surrogates' training data. That means that the surrogate's minimizer might not be the true minimizer of the real benchmark.

Table 1. The hyperparameters and architecture choices for the fully connected networks.

Hyperparameter	Range	Log-transform
batch size	$[2^3, 2^8]$	yes
dropout rate	$[0, 0.5]$	no
initial learning rate	$[10^{-6}, 10^{-2}]$	yes
exponential decay factor	$[-0.185, 0]$	no
# hidden layers	$\{1, 2, 3, 4, 5\}$	no
# units per layer	$[2^4, 2^8]$	yes

None of these downsides necessarily have substantial implications for comparing different optimizers; they simply show that the surrogate benchmarks are not perfect models for the real benchmark they mimic. Nevertheless, we believe that, especially for development of novel algorithms, the positive aspects outweigh the negative ones.

I.2.2. Determining the budgets

To choose the largest budget for training, we looked at the best configuration as predicted by the surrogate and its training time. We chose the closest power of 3 (because we also used $\eta = 3$ for HB and BOHB) to achieve that performance. We chose the smallest budget for HB such that most configurations had finished at least one epoch. Table 2 lists the budgets used for all datasets.

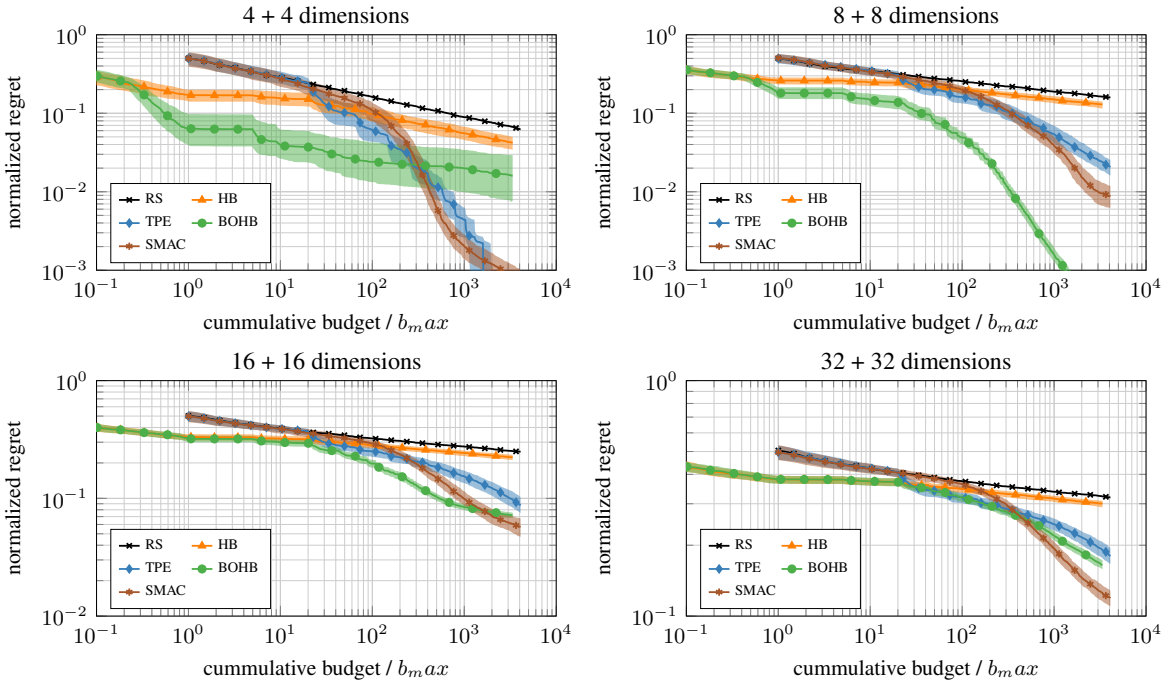

Figure 9. Median performance of BOHB, HB, TPE, SMAC and RS on the mixed domain counting ones function with different dimensions. As uncertanties we show the interquartile range based on 512 runs.

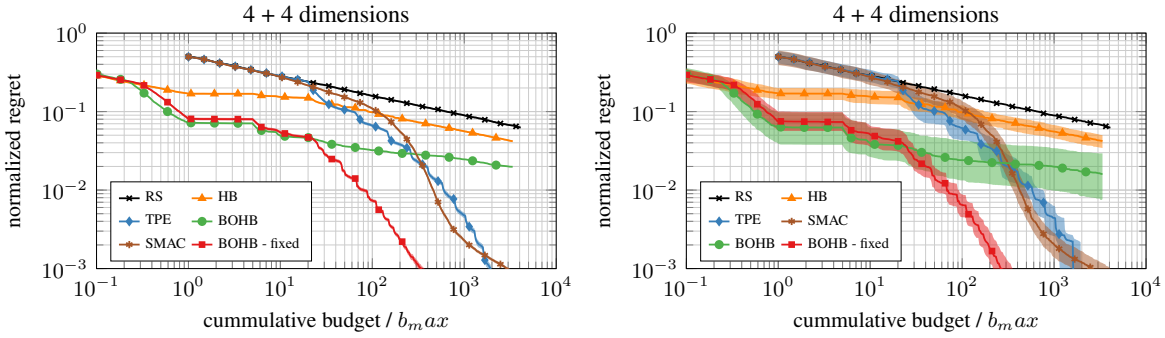

Figure 10. Mean (left) and median (right) performance of BOHB, HB, TPE, SMAC and RS on the mixed domain counting ones function eight dimensions. To show that the poor performance of BOHB is merely product of poor initialization via random search that lead to a overconfident model that does not converge, we reran BOHB (labeled BOHB-fixed) with a slightly changed setting (see text) showing this benchmark required more exploration than the default settings provided.

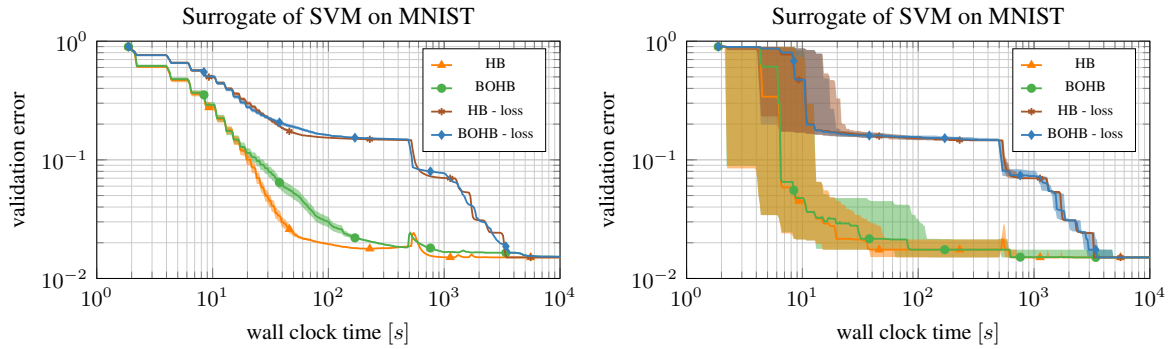

Figure 11. Results for BOHB and HB on the SVM surrogate benchmark. The two panels show the mean (left) and the median (right) of the test error (error when retrained using the whole dataset) and the actual training loss (error rate achieved with the subsets used) labeled HB-loss and BOHB-loss. The apparent advantage of HB over BOHB vanishes when we look the actual losses. This seems to indicate overfitting due to the low dimensionality of the search space and the large number of configurations sampled for the first iteration.

Table 2. The budgets used by HB and BOHB; random search and TPE only used the last budget

Dataset	Budgets in seconds for HB and BOHB
Adult	9, 27, 81, 243
Higgs	9, 27, 81, 243
Letter	3, 9, 27, 81
Poker	81, 243, 729, 2187

Table 3. The hyperparameters for the Bayesian neural network task.

Hyperparameter	Range	Log-transform
# units layer 1	$[2^4, 2^9]$	yes
# units layer 2	$[2^4, 2^9]$	yes
step length	$[10^{-6}, 10^{-1}]$	yes
burn in	$[0, .8]$	no
momentum decay	$[0, 1]$	no

J. Bayesian Neural Networks

We optimized the hyperparameters described in Table 3 for a Bayesian neural network trained with SGHMC on two UCI regression datasets: Boston Housing and Protein Structure. The budget for this benchmark was the number of steps for the MCMC sampler. We set the minimum budget to 500 steps and the maximum budget to 10000 steps. After sampling 100 parameter vectors, we computed the log-likelihood on the validation dataset by averaging the predictive mean and variances of the individual models. The performance of all methods for both datasets is shown in Figure 12.

K. Reinforcement Learning

Table 4 shows the hyperparameters we optimized for the PPO Cartpole task.

Table 4. The hyperparameters for the PPO Cartpole task.

Hyperparameter	Range	Log-transform
# units layer 1	$[2^3, 2^7]$	yes
# units layer 2	$[2^3, 2^7]$	yes
batch size	$[2^3, 2^8]$	yes
learning rate	$[10^{-7}, 10^{-1}]$	yes
discount	$[0, 1]$	no
likelihood ratio clipping	$[0, 1]$	no
entropy regularization	$[0, 1]$	no

References

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.

Falkner, S., Klein, A., and Hutter, F. Combining hyperband and Bayesian optimization. In *NIPS 2017 Bayesian Optimization Workshop*, 2017.

Falkner, S., Klein, A., and Hutter, F. Practical hyperparameter optimization for deep learning. In *ICLR 2018 Workshop Track*, 2018.

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

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.

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

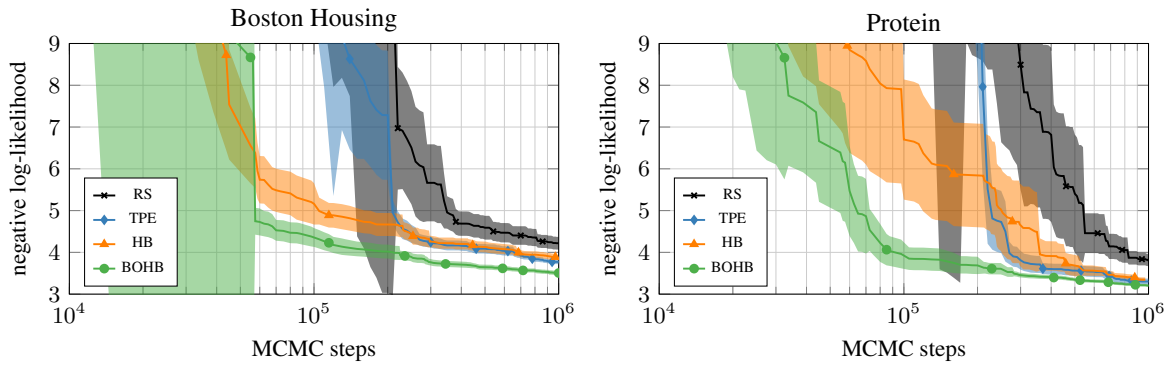

Figure 12. Mean performance of TPE, RS, HB and BOHB for optimizing the 5 hyperparameters of a Bayesian neural network on two different UCI datasets. As uncertainties, we show the stardard error of the mean based on 50 runs.

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.