Combining Hyperband and Bayesian Optimization

Stefan Falkner **Aaron Klein** **Frank Hutter**
Department of Computer Science
University of Freiburg
{sfalkner, kleinaa, fh}@cs.uni-freiburg.de

Abstract

Proper hyperparameter optimization is computationally very costly for expensive machine learning methods, such as deep neural networks; the same holds true for neural architecture search. Recently, the bandit-based strategy Hyperband has shown superior performance to vanilla Bayesian optimization methods that are limited to the traditional problem formulation of expensive blackbox optimization. However, while Hyperband has strong anytime performance for finding configurations with *acceptable* results, it relies on random search and therefore does not find the *best* configurations quickly. We propose to combine Hyperband with Bayesian optimization by maintaining a probabilistic model that captures the density of good configurations in the input space and samples from this model instead of sampling uniformly at random. We empirically show that our new method combines Hyperband's strong anytime performance with the strong eventual performance of Bayesian optimization.

1 Introduction

Finding good hyperparameter settings for a machine learning method often makes the difference between achieving state-of-the-art or quite weak performance. Various methods, such as Bayesian optimization (BO) [Snoek et al., 2012, Hutter et al., 2011, Bergstra et al., 2011] or random search [Bergstra and Bengio, 2012] try to automate the search for good hyperparameter settings by formulating it as a blackbox optimization problem. However, the long training time and the demand for large computational power of contemporary machine learning methods, such as deep neural networks, limit the usefulness of these methods: when single function evaluations require days or weeks it becomes computationally infeasible to apply blackbox optimizers. Recent advances in hyperparameter optimization therefore go beyond this limiting blackbox formulation and consider cheap approximate function evaluations, such as performance when running on a subset of data or optimizing a deep neural network for few epochs [Swersky et al., 2014, Klein et al., 2017a, Li et al., 2017, Klein et al., 2017b].

In particular, when approximate function evaluations are exponentially cheaper than full function evaluations, it is possible to evaluate an exponentially larger number of configurations in the same budget. The recent method Hyperband (HB) [Li et al., 2017] and its building block of successive halving [Jamieson and Talwalkar, 2016] exploit this strategy by evaluating N hyperparameter configurations based on one unit of budget (e.g., a fixed number of epochs, or time interval), continue the best half to two units of budget, the best half thereof to four units, etc. Despite Hyperband's simplicity it was able to outperform Bayesian optimization methods that evaluate hyperparameter configurations on the full budget. However, this only holds for finding *reasonably good* configurations; since Hyperband and successive halving are based on random search, they do not use previous samples to guide their search and therefore are slow in finding the *best* configurations.

The contribution of this paper is to extend Hyperband with a probabilistic model that captures the density of good configurations in the input space. By sampling configurations from this model

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

instead of uniformly at random, our new method combines Hyperband's strong anytime property with Bayesian optimization's guided approach for identifying the best configuration.

2 Background

Bayesian Optimization (BO). In its vanilla version, BO [Shahriari et al., 2016] aims to find a global optimizer $\boldsymbol{x} \in \arg\min f(\boldsymbol{x})$ of a blackbox function $f : \mathbb{X} \to \mathbb{R}$. It iteratively fits a probabilistic model $p(f|D)$ to capture the belief about f based on previous observations $D = \{(\boldsymbol{x}_0, y_0), \ldots, (\boldsymbol{x}_i, y_i)\}$, where we assume only access to noisy observations $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$, with $\epsilon \sim \mathcal{N}(0, \sigma^2_{noise})$. To select the next point $\boldsymbol{x}_{\text{new}}$ to evaluate, BO uses an acquisition function $a(\boldsymbol{x})$ based on $p(f|D)$ and picks a point that maximizes it, i.e. $\boldsymbol{x}_{new} \in \arg\max_{\boldsymbol{x}\in\mathcal{X}} a(\boldsymbol{x})$. It then evaluates f at $(\boldsymbol{x}_{\text{new}})$, obtains y_{new}, updates the probabilistic model and iterates. A popular choice for the acquisition function is the expected improvement (EI) [Jones et al., 1998] over a reference value α under the current model:

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

Tree-of-Parzen-Estimators (TPE). TPE [Bergstra et al., 2011] is a specific instantiation of the general BO method that, instead of having an explicit model for the function f, uses Parzen kernel density estimators (KDE) to approximate the densities

$$l(\boldsymbol{x}) = p(y < \alpha|\boldsymbol{x}) \qquad \text{and} \qquad g(\boldsymbol{x}) = p(y > \alpha|\boldsymbol{x}) \tag{2}$$

that model the probability of the function value being less or greater than a given value α. Bergstra et al. [2011] showed that optimizing Eq. (1) under this model is equivalent to maximizing the ratio $g(\boldsymbol{x})/l(\boldsymbol{x})$. This maximization can be performed efficiently by drawing samples according to $l(\boldsymbol{x})$ and evaluating the ratio. This stochastic optimization can directly be used to parallelize TPE, by relying on the random samples to be different. To ascertain sufficient exploration, TPE allows a user-defined prior over the search space; choosing this as the uniform distribution (as done by default) effectively adds randomly sampled configurations, yielding a 'safe' BO method [Ahmed et al., 2017]. TPE also natively supports conditional hyperparameter spaces, in which not all dimensions are relevant for all configurations.

Hyperband (HB). HB is a bandit-based strategy for hyperparameter optimization that iteratively allocates resources to a set of random configurations. Given a predefined budget B for one iteration, e. g. time or total number of epochs, in each of its iteration i, HB samples N_i configurations uniformly at random and uses successive halving to discard poorly performing configurations. The budget for each configuration increases by a factor of η while the number of continued configurations decreases by it. By automatically exploring different trade-offs between N_i and B, HB is guaranteed to be at most a constant factor slower than random search.

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.

3 Model-based Hyperband

We aim to combine the building blocks described above to obtain a new state-of-the-art hyperparameter optimization method that fulfills the following desiderata:

- **Strong anytime performance:** for short/medium budgets, we aim to do as well as HB.
- **Strong final performance:** for large budgets, we aim to converge as well as BO.
- **Computational efficiency:** to facilitate parallelization, we aim for minimal overhead of model construction and use compared to the shortest possible runs.
- **Conceptual simplicity:** compared to sophisticated BO methods, Hyperband is very simple, allowing quick re-implementation in different frameworks; we thus aim for a simple model.
- **Robustness:** in constrast to existing BO approaches that go beyond blackbox optimization, we aim to avoid any parametric assumptions and to also perform well for high-dimensional and conditional spaces.

We achieve these goals by combining HB with TPE in our new BO-HB method to get the best of both worlds. Our choice is based on previous evaluations of BO methods [Eggensperger et al., 2013] which showed TPE to be comparable to GP-based methods but with substantially less computational overhead. The sampling based optimization by Bergstra et al. [2011] also allows for efficient and effective parallelism, even in complex search spaces with conditionals. TPE is conceptually simpler than traditional BO methods and the properties above also make it more robust than BO approaches that, e.g., rely on GPs.

Our extension replaces the random sampling 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, and we keep track of the performance across all budgets to use as a basis for our models in later iterations.

Algorithm 1 describes the procedure we use to sample new configurations in BO-HB. We construct a model to sample from for each budget B, based on the configurations evaluated so far with that budget. We opted for a single multidimensional KDE compared to the hierarchical one-dimensional KDEs used in TPE. We used the KDE implementation from statsmodels [Seabold and Perktold, 2010], estimating the KDE's bandwidth with the default estimation procedure (Scott's rule of thumb). For a useful KDE, we require a minimum number of configurations N_{min} set to $2 \cdot d$ for our experiments, where d is the number of hyperparameters. Hence, BO-HB starts by sampling N_{min} configurations for the lowest budget at random, followed by configurations proposed by the model. When starting a new iteration with a larger budget, we sample configurations from the model for the largest available budget. In order to keep the theoretical guarantees of HB, we also sample a constant fraction p of the configurations uniformly at random. Because of this, our method is still guaranteed to be only a constant factor slower than random search in the worst case (in which the performance for smaller budgets is negatively correlated with the one for the largest budget).

Algorithm 1: Pseudocode for how to sample a new configuration in BO-HB

input : previous observations D, constant fraction of random configurations p, percentile q, number of samples N_s to optimize EI, minimum number of points N_{min} to build a model
output : next configuration to evaluate
if *rand()* < p **then return** random configuration
find largest budget B with at least N_{min} observations
if *no such* B *exists* **then return** random configuration
$\alpha = q^{\text{th}}$ percentile of all y values in D_B
fit KDE for $l(\boldsymbol{x})$ and $g(\boldsymbol{x})$ on D_B and α, see Eq. (2)
draw N_s samples according to $l(\boldsymbol{x})$
return sample with highest ratio $g(\boldsymbol{x})/l(\boldsymbol{x})$

4 Experiments

We now empirically assess our method's performance for the joint optimization of neural network architectural choices and hyperparameters, a task that naturally lends itself to the different budgets within HB due to the iterative optimization of network weights via stochastic gradient descent.

Benchmarks. We used a simple feed forward network architecture and a few hyperparameters to define our model. The hyperparameters used describe the architecture of the network (number of hidden layers and the number of units in a layer) and properties of the training; all hyperparameters can be found in Table 1, along with their ranges. We trained networks on four datasets from OpenML [Vanschoren et al., 2014]: Adult [Kohavi, 1996], Higgs [Baldi et al., 2014], Letter [Frey and Slate, 1991], and Poker [Cattral et al., 2002], using the Adam optimizer [Kingma and Ba, 2014]. For faster development and to afford more runs of the different optimizers, we constructed surrogate benchmarks based on random evaluations following Eggensperger et al. [2013]; a detailed description of this procedure can be found in the supplementary material. On these surrogates, we evaluated the performance of all optimizers by running them

Table 1: The hyperparameters for the trained 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

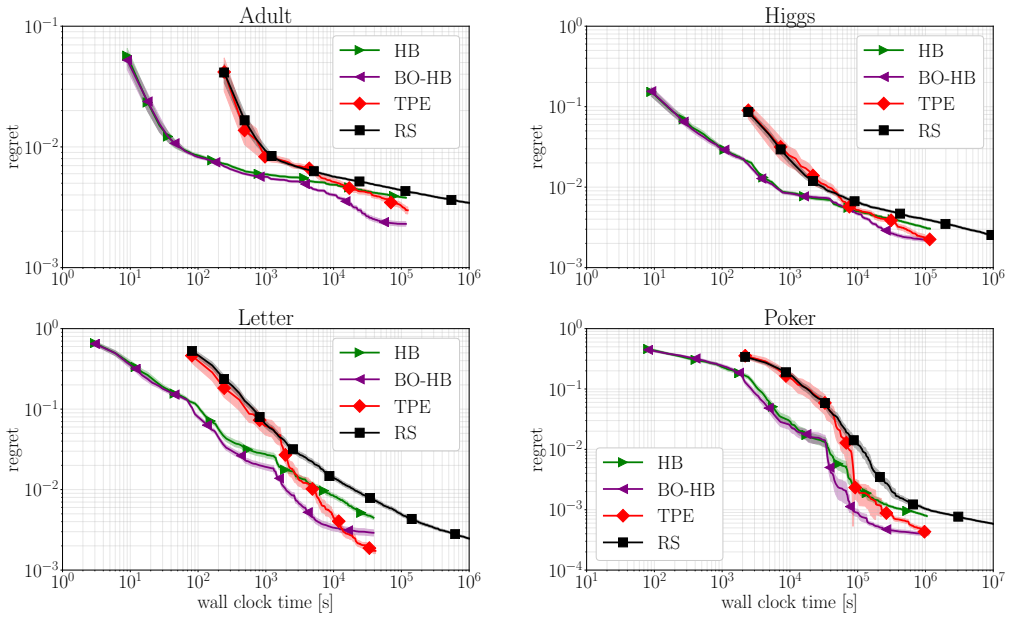

Figure 1: Immediate regret of random search (RS), TPE, HB and BO-HB for optimizing a neural network on the four datasets. We averaged the performance of 256 runs for each method and show twice the standard error of the mean as uncertainty. Note how RS and TPE are indistinguishable for the first 20-30 function evaluations. HB and BO-HB evaluate shorter runs hence their trajectories start earlier. While the latter two clearly dominate the regret for small budgets, TPE can be very competitive and even beat both when run long enough.

repeatedly and averaging the results. To better compare the convergence towards the true optimum, we compute the regret $|f(x) - f(x^*)|$ where $x^* \in \arg\min f(x)$. We specify the training budgets used on the different datasets in the supplementary material.

Results. We ran random search, TPE, HB (with the default $\eta = 3$) and BO-HB on all four surrogates and summarize the results in Figure 1. We note that HB initially performed much better than TPE, but TPE caught up in all cases for large enough budgets. BO-HB started out identical to HB and performed better with larger budgets (with comparable results to TPE); in particular, BO-HB and TPE reached comparable performance for Higgs and Poker, with Adult and Letter representing examples of BO-HB and TPE performing a bit better in the end, respectively.

The results on Letter are a clear indication that for large enough budgets, the overhead of evaluating different budgets can be detrimental to the performance. We see two main reasons for this: (a) the model TPE built for the largest budget contained more data points than the one in BO-HB, thereby impacting performance, and (b) the partial evaluation of new hyperparameter configurations slows down progress in later phases by wasting computational power.

All three methods substantially outperformed random search at the end of their budget. If continued for more than ten times the budget of the other methods, random search also found competitive configurations. We note that the speedups that TPE achieved over random search are comparable to the speedups BO-HB gained over HB.

5 Conclusions

We introduced BO-HB, an extension to Hyperband that, similar to TPE, models the density of good and bad performing configuration in the input space. By sampling from this model instead of uniformly at random, we can cure the drawbacks of random search but keep the performance gains of Hyperband. In future work we will extend our model to work in mixed continuous and discrete space, study the performance when run in parallel and apply it to more realistic benchmarks.

6 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 BO-HB. 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 Euro- pean 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

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, pages 2960–2968, 2012.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION'11*, pages 507–523, 2011.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Proc. of NIPS'11*, pages 2546–2554, 2011.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305, 2012.

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

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast Bayesian optimization of machine learning hyperparameters on large datasets. In *Proc. of AISTATS'17*, 2017a.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. In *Proc. of ICLR'17*, 2017.

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. In *Proc. of ICLR'17*, 2017b.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Proc. of AISTATS'16*, 2016.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

D. Jones, M. Schonlau, and W. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

M. Ahmed, B. Shahriari, and M. Schmidt. Do we need "harmless" bayesian optimization and "first-order" bayesian optimization? In *Proc. of BayesOpt'17*, 2017.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, and K. Leyton-Brown. Towards an empirical foundation for assessing Bayesian optimization of hyperparameters. In *NIPS Workshop on Bayesian Optimization in Theory and Practice (BayesOpt'13)*, 2013.

Skipper Seabold and Josef Perktold. Statsmodels: Econometric and statistical modeling with python. In *9th Python in Science Conference*, 2010.

J. Vanschoren, J. van Rijn, B. Bischl, and L. Torgo. OpenML: Networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.

Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *KDD*, volume 96, pages 202–207, 1996.

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

Peter W. Frey and David J. Slate. Letter recognition using holland-style adaptive classifiers. *Machine Learning*, 6(2):161–182, Mar 1991.

Robert Cattral, Franz Oppacher, and Dwight Deugo. Evolutionary data mining with automatic rule generalization. *Recent Advances in Computers, Computing and Communications*, 1(1):296–300, 2002.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arXiv:1412.6980, 2014.

A 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 random forests to predict both quantities independently given a new hyperparameter configuration. 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 allow cheap evaluations and are the reason we could afford to run each algorithm 256 times. Since evaluating a configuration with the random forest is inexpensive, we used a global 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 about the surrogates, there are also drawback 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 benchmark.

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

None of these downsides have any real 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.

B 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 BO-HB) 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.

Table 2: The budgets used by HB and BO-HB; random search and TPE only used the last budget

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