Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters

Katharina Eggensperger, Matthias Feurer, Frank Hutter
Freiburg University
{eggenspk,feurerm,fh}@informatik.uni-freiburg.de

James Bergstra
University of Waterloo
james.bergstra@uwaterloo.ca

Jasper Snoek
Harvard University
jsnoek@seas.harvard.edu

Holger H. Hoos and Kevin Leyton-Brown
University of British Columbia
{hoos,kevinlb}@cs.ubc.ca

Abstract

Progress in practical Bayesian optimization (BO) is hampered by the fact that the only available standard benchmarks are artificial test functions that are not representative of practical applications. To alleviate this problem, we introduce a library of benchmarks from the prominent application of hyperparameter optimization and use it to compare Spearmint, TPE, and SMAC, three recent BO methods for hyperparameter optimization.

1 Introduction

The performance of many machine learning (ML) methods depends crucially on hyperparameter settings and thus on the method used to set hyperparameters. Recently, Bayesian optimization (BO) methods have been shown to outperform established methods for this problem (such as grid search and random search [1]) and to rival—and in some cases surpass—human domain experts in finding good hyperparameter settings [2, 3, 4]. As a result, hyperparameter optimization has become an active research area within BO, with characteristics such as low effective dimensionality [1, 5, 6] and problem variants, such as optimization across different data sets [7] being explored.

One obstacle to further progress in this nascent field is a dearth of hyperparameter optimization benchmarks and comparative empirical studies. To date, a typical paper introducing a new hyperparameter optimizer also introduces a new set of hyperparameter optimization benchmarks, on which the optimizer is demonstrated to achieve state-of-the-art performance as compared to, e.g., human domain experts. It can be difficult to evaluate a new optimizer on benchmarks used in previous papers because (1) optimizers are written in different programming languages and use different search space representations and file formats; (2) hyperparameter optimization benchmarks that have been developed jointly with an optimizer are not typically packaged as black boxes (including the respective machine learning algorithm and its input data) that can be used with other optimizers. These problems represent a considerable barrier to anyone aiming to develop a new hyperparameter optimization algorithm and objectively measure its performance. As a result, so far it has been unknown how recent hyperparameter optimizers compare across benchmarks.

To alleviate these problems, we have collected and made available a library of hyperparameter optimization benchmarks from the recent literature and used it to empirically evaluate the respective strengths and weaknesses of three prominent BO methods for hyperparameter optimization: Spearmint [2], TPE [8], and SMAC [9]. We thereby hope to provide an empirical foundation to facilitate the development and evaluation of future methods for this problem.

2 Bayesian Optimization Methods for Hyperparameter Optimization

Given a machine learning algorithm A having hyperparameters $\lambda_1, \ldots, \lambda_n$ with respective domains $\Lambda_1, \ldots, \Lambda_n$, we define its hyperparameter space $\boldsymbol{\Lambda} = \Lambda_1 \times \cdots \times \Lambda_n$. Hyperparameters can be continuous, integer-valued, or categorical. Following [10] and [8], we say that a hyperparameter λ_i is *conditional* on another hyperparameter λ_j if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subsetneq \Lambda_j$. Let $\mathcal{L}(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$ denote the validation loss (e.g., misclassification rate) that A achieves on data $\mathcal{D}_{\text{valid}}$ when trained on $\mathcal{D}_{\text{train}}$. The hyperparameter optimization problem under k-fold cross-validation is then to minimize the blackbox function

$$f(\boldsymbol{\lambda}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}^{(i)}_{\text{train}}, \mathcal{D}^{(i)}_{\text{valid}}). \tag{1}$$

Bayesian optimization (see [11] for a detailed tutorial) constructs a probabilistic model $\mathcal{M}$ of f based on point evaluations of f and any available prior information, and uses that model to select subsequent configurations $\boldsymbol{\lambda}$ to evaluate. In order to select its next hyperparameter configuration $\boldsymbol{\lambda}$ using model $\mathcal{M}$, BO uses an *acquisition function* $a_{\mathcal{M}} : \boldsymbol{\Lambda} \to \mathbb{R}$, which uses the predictive distribution of model $\mathcal{M}$ at arbitrary hyperparameter configurations $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ to quantify how useful knowledge about $\boldsymbol{\lambda}$ would be. This function is then maximized over $\boldsymbol{\Lambda}$ to select the most useful configuration $\boldsymbol{\lambda}$ to evaluate next. Several well-studied acquisition functions exist [12, 13, 14]; all aim to trade off exploitation (locally optimizing hyperparameters in regions known to perform well) versus exploration (trying hyperparameters in a relatively unexplored region of the space). The most popular acquisition function is the *expected improvement ($EI(\boldsymbol{\lambda})$)* [13] over an existing given error rate f_{min} attainable at a hyperparameter configuration $\boldsymbol{\lambda}$, where expectations are taken over predictions with the current model $\mathcal{M}$:

$$\mathbb{E}_{\mathcal{M}}[I_{f_{min}}(\boldsymbol{\lambda})] = \int_{-\infty}^{f_{min}} \max\{f_{min} - f, 0\} \cdot p_{\mathcal{M}}(f \mid \lambda)\, df. \tag{2}$$

One main difference between existing BO algorithms lies in the model classes they employ. In this paper, we empirically compare three popular BO algorithms for hyperparameter optimization that are based on different model types.

Spearmint [2, 15]. Spearmint uses Gaussian process (GP) models and performs slice sampling over the GP's hyperparameters for a fully Bayesian treatment. Spearmint supports both continuous and discrete parameters (by rounding). It does not exploit knowledge about the conditionality of parameters. It draws new points by calculating expected improvement over a Sobol grid. Due to limitations in the Sobol library, it does not support more than 40 hyperparameters.

Sequential Model-based Algorithm Configuration (SMAC) [9, 16]. SMAC is based on random forest models. Uncertainty estimates are obtained as the empirical variance over the predictions of the trees in the given forest. For hyperparameter optimization problems with cross-validation, SMAC evaluates the loss of configurations at single folds at a time in order to save time. Different hyperparameter configurations are compared based only on the folds evaluated for both. SMAC supports continuous, categorical, and conditional parameters. It was the best-performing optimizer for Auto-WEKA [3] and has also been used to configure many combinatorial optimization algorithms.

Tree Parzen Estimator (TPE) [8, 17]. TPE is a non-standard BO algorithm. While Spearmint and SMAC model $p(f \mid \lambda)$ directly, TPE models $p(f < f*)$, $p(\boldsymbol{\lambda} \mid f < f*)$, and $p(\boldsymbol{\lambda} \mid f \geq f*)$, where $f*$ is defined as a fixed quantile of the losses observed so far, and the latter two probabilities are defined by tree-structured Parzen density estimators. With these distributions defined, a term proportional to the expected improvement from Equation (2) can be computed in closed form [8]. TPE supports continuous, categorical, and conditional parameters, as well as priors for each hyperparameter over which values are expected to perform best. TPE has been used succesfully in several papers beyond the one in which it was introduced [4, 18, 3].

3 Hyperparameter Optimization Benchmarks

Table 1 summarizes all of the benchmarks we collected. These include simple test functions (used for convenience in many papers) as well as benchmarks collected from recent work on hyperparameter optimization using the optimizers we aim to evaluate [8, 2, 3]. We make these benchmarks available at `www.automl.org/benchmarks.html` in various formats.

Table 1: Hyperparameter optimization benchmarks used. The loss function is misclassification rate for all but LDA, where it is perplexity.

Algorithm	#hyp.params(conditional)	continuous/discrete	Dataset	Size (Train/Valid/Test)	Citation	runtime
Branin	2(-)	2/-	-	-	[19]	$< 1s$
Hartmann 6d	6(-)	6/-	-	-	[19]	$< 1s$
Log. Reg.	4(-)	4/-	MNIST	50k/10k/10k	[2, 20]	$< 5m$
Log. Reg.	4(-)	4/-	MNIST	5 fold cv, 60k Train, 10k Test	[2, 20]	$< 25m$
LDA ongrid	3(-)	-/3	Wikipedia articles	200k/24560/25k	[2, 21]	$< 1s$
SVM ongrid	3(-)	-/3	UniPROBE	$\approx$ 20k/-/$\approx$ 20k	[2, 22]	$< 1s$
HP-NNET	14(4)	7/7	MRBI	10k/2k/50k	[8, 23]	$\approx 6m$
HP-NNET	14(4)	7/7	MRBI	5 fold cv, 12k Train, 50k Test	[8, 23]	$\approx 25m$
HP-NNET	14(4)	7/7	convex	6.5k/1.5k/50k	[8, 23]	$\approx 7m$
HP-NNET	14(4)	7/7	convex	5 fold cv, 8k Train, 50k Test	[8, 23]	$\approx 25m$
HP-DBNET	38(29)	19/17	convex	6.5k/1.5k/50k	[8, 23]	$\approx 10m$
Auto-WEKA	786(784)	296/490	convex	6.5k/1.5k/50k	[3, 23]	$\approx 15m$

Low-dimensional benchmarks. We collected three hyperparameter optimization benchmarks with small numbers of parameters from [2]: simple LOGISTIC REGRESSION to classify the popular MNIST dataset; ONLINE LATENT DIRICHLET ALLOCATION (LDA) for Wikipedia articles, and STRUCTURED SUPPORT VECTOR MACHINES (SVM). The latter two benchmarks are defined on a grid of hyperparameter values: for each of the grid points (288 for LDA; 1 400 for SVM), algorithm performance data (on a validation set; no cross-validation) has been precomputed by [2] to allow for very rapid experiments. Another advantage of such precomputed data is that anyone can use these benchmarks without having to compile and run the respective ML algorithms.

Medium-dimensional benchmarks. We collected two types of hyperparameter optimization benchmarks of intermediate dimensionality from [8]. HP-NNET and HP-DBNET are implementations of a simple neural network and a deep neural network, respectively. Both run dramatically faster on GPUs than CPUs. Both include continuous and categorical parameters, some of which are conditional. Both use preprocessing methods that (at least for HP-NNET) need to be set carefully to achieve high performance. For each hyperparameter in these benchmarks, an expert-defined prior over good values is defined. Since these priors cannot be expressed in SMAC's and Spearmint's formats, they get lost in translation, as does conditionality information in the case of Spearmint.

High-dimensional benchmarks. To test the limits of current optimizers, we also investigated the AUTO-WEKA framework [3], which encodes combined model selection and hyperparameter optimization into an enormous hierarchical space with 768 hyperparameters.

Cross-validation is essential to avoid overfitting, especially as hyperparameter optimization methods improve. Nevertheless, out of the methods above, only Auto-WEKA used cross-validation. To study the impact of cross-validation, we considered additional versions of the LOGISTIC REGRESSION and HP-NNET benchmarks with 5-fold cross-validation.

4 Results

We ran each optimizer with its default settings 10 times on each benchmark for the number of function evaluations used in the paper introducing the benchmark. Individual runs (if cross-validation is used, individual folds) of all machine learning methods were subjected to a timeout of one hour, but this only took effect for a few runs on the HP-DBNET. The HP-NNET and HP-DBNET experiments required GPUs (we used NVIDIA Tesla M2070s for HP-NNET and a NVIDIA GeForce GTX 780 for HP-DBNET). For benchmarks including cross-validation, it is possible to evaluate one fold at a time; SMAC took advantage of this while the other methods do not support it yet and thus evaluated all 5 cross-validation folds for a selected configuration at once.

Table 2 summarizes the results. Overall, Spearmint performed best for the low-dimensional continuous problems, followed by SMAC and then TPE (which does not model interactions between parameters). SMAC showed solid all-round performance, likely owing to its robust random forest models. Finally, TPE performed well on the discrete low-dimensional benchmarks and best for HP-DBNET, likely due to its ability to exploit the expert-defined priors about good parameter values available for this benchmark.

We also encountered a few surprises. Firstly, Spearmint had some robustness problems, e.g., crashing on 3 of 10 runs for the HARTMANN-6 function because of a singular covariance matrix. It also had some problems with discrete parameter values: by maximizing EI over a dense sobol grid instead of the discrete input grid, in some cases it repeatedly chose values that were rounded to the same

Table 2: Losses obtained for all optimizers and benchmarks. We report means and standard deviation across up to 10 runs of each optimizer (except for the entries with superscripts, which show the number of runs completed[1]). For each benchmark, bold face indicates the best mean value, and underlined values are not statistically significantly different from the best according to a Mann-Whitney-U test (with p=0.05). The absolute values with and without cross-validation are incomparable as they are based on different data set splits.

		SMAC		**Spearmint**		**TPE**	
Experiment	#evals	Valid.Error	Best Value	Valid.Error	Best Value	Valid.Error	Best Value
branin (0.398)	200	0.427 ± 0.019	0.400	$\underline{\mathbf{0.398}}\pm0.000$	**0.398**	0.510 ± 0.077	0.406
har6 (-3.32237)	200	$\underline{\mathbf{-2.960}}\pm0.200$	-3.165	-2.623 ± 1.026	**-3.322**	$\underline{-2.920}\pm0.164$	-3.115
Log.Regression	100	0.086 ± 0.006	0.077	$\underline{\mathbf{0.073}}\pm0.001$	**0.070**	0.082 ± 0.003	0.75
Log.Regression 5CV	100	$\underline{\mathbf{0.081}}\pm0.003$	**0.078**	$\underline{0.083}\pm0.001$	0.080	0.089 ± 0.007	0.081
LDA ongrid	50	$\underline{\mathbf{1269.6}}\pm2.9$	**1266.2**	$\underline{1272.6}\pm10.2$	**1266.2**	$\underline{1271.5}\pm3.5$	**1266.2**
SVM ongrid	100	$\underline{\mathbf{0.241}}\pm0.001$	**0.241**	$\underline{0.246}\pm0.009$	**0.241**	$\underline{\mathbf{0.241}}\pm0.000$	**0.241**
HP-NNET convex	100	$\underline{\mathbf{0.19}}\pm0.014^{(8)}$	**0.175**	$\underline{0.209}\pm0.003^{(6)}$	0.020	$\underline{0.202}\pm0.011^{(9)}$	0.179
HP-NNET convex 5CV	50	$\underline{\mathbf{0.208}}\pm0.010^{(4)}$	0.198	$\underline{0.230}\pm0.015^{(2)}$	0.215	$\underline{0.210}\pm0.012^{(4)}$	**0.189**
HP-NNET MRBI	100	$\underline{0.527}\pm0.019^{(8)}$	0.504	$\underline{\mathbf{0.501}}\pm0.032^{(5)}$	**0.467**	$\underline{0.503}\pm0.021$	0.478
HP-NNET MRBI 5CV	50	$\underline{0.512}\pm0.032^{(4)}$	0.478	$\underline{\mathbf{0.497}}\pm0.020^{(3)}$	**0.476**	$\underline{0.518}\pm0.000^{(1)}$	0.518
HP-DBNET convex	200	$\underline{0.159}^{(1)}$		$\underline{0.138}^{(1)}$		$\underline{\mathbf{0.135}}^{(1)}$	
Auto-WEKA[2]	30h	$\underline{\mathbf{0.221}}$		N/A		0.255	

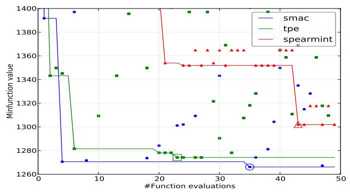

Figure 1: Response values obtained & incumbent over time for one run of each optimizer on LDA ONGRID.

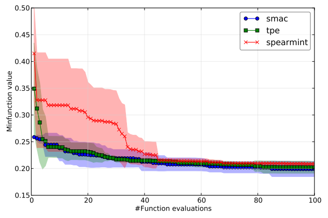

Figure 2: Mean ± stddev of best loss over time, HP-NNET convex

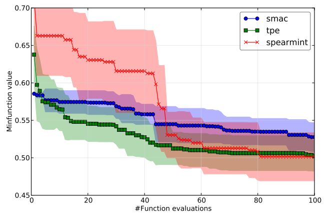

Figure 3: Mean ± stddev of best loss over time, HP-NNET MRBI

discrete values for evaluation (leading to repetitions in the samples). This problem is demonstrated in Figure 1.

Secondly, performance did not only depend on the algorithm but also on the dataset: SMAC performed best for the HP-NNET on one dataset (convex, see Figure 2) and worst on the other (MRBI, see Figure 3), and exactly the opposite held for Spearmint. A more detailed analysis of the data shows that it was crucial to switch on PCA preprocessing with its conditional strength parameter set to around 0.7. SMAC and Spearmint consistently failed to do so for the datasets they performed poorly on. It was surprising to see this, but further investigation is necessary to find out why it occurred. We pause here to note that these potential problems in SMAC and Spearmint could not have been identified without a competitive comparison against other methods; otherwise, one would have simply assumed that a good configuration was found.

We also studied the CPU time required by the optimizers. SMAC'S and TPE'S overhead was negligible (< 1 second), but Spearmint required > 42 seconds to select the next data point after 200 evaluations. This is of course due to the cubic scaling behaviour of Spearmint's GPs, which would prohibit its use for the optimization of cheap functions. However, in our setting this overhead was dominated by the expensive function evalations.

5 Conclusion and future work

This work provides the first extensive comparison of three hyperparameter optimizers, showing that each has their advantages and disadvantages (mostly in terms of robustness and search space definitions supported). To support further research, our software package and benchmarks are publicly available at `www.automl.org/benchmarks.html`. It offers a common interface for the three optimization packages utilized in this paper and allows the easy integration of new ones. Our benchmark library is only a first step, but we are committed to making it easy for other researchers to use it and to contribute their own benchmarks and optimization packages.

[1]Unfortunately, not all of our 10×3 optimizer runs for the GPU experiments have finished, due to an unscheduled 2-week outage of the GPU cluster we used. We will update the results for the final version.

[2]Spearmint does not apply for Auto-WEKA (since it has more than 40 hyperparameters). Since these runs are computationally expensive, we reuse results from [3], which already compared SMAC and TPE.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

[2] J. Snoek, H. Larochelle, and R.P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, 2012.

[3] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD'13*, 2013.

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proc. of ICML'12*, 2013.

[5] B. Chen, R.M. Castro, and A. Krause. Joint optimization and variable selection of high-dimensional Gaussian processes. In *Proc. of ICML'12*, 2012.

[6] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas. Bayesian optimization in high dimensions via random embeddings. In *Proc. of IJCAI'13*, 2013.

[7] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *Proc. of ICML'13*, 2013.

[8] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *Proc. of NIPS'11*, 2011.

[9] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, 2011.

[10] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *JAIR*, 36(1):267–306, 2009.

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

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

[13] M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. In *New Developments and Applications in Experimental Design*, volume 34, pages 11–25. 1998.

[14] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. of ICML'10*, 2010.

[15] J. Snoek. Spearmint source code. Version from September 1, 2013.

[16] S. Ramage and F. Hutter. SMAC source code, java version v2.06.01.

[17] J Bergstra. TPE source code, part of hyperopt implementation, github version from 08/30/2013.

[18] J. Bergstra and D.D. Cox. Hyperparameter optimization and boosting for classifying facial expressions: How good can a "null" model be? *CoRR*, abs/1306.3476, 2013.

[19] A. Hedar. Test functions for unconstrained global optimization. `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm`. Last accessed on Oct 16, 2013.

[20] Y. Lecun, l. Bottou, Y. Bengio, and P.Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[21] M. D. Hoffman, D. M. Blei, and F. R. Bach. Online learning for latent dirichlet allocation. In *Proc. of NIPS'10*, 2010.

[22] K. Miller, M. P. Kumer, B. Packer, D. Goodman, and D. Koller. Max-margin min-entropy models. In *Proc. of AISTATS'12*, 2012.

[23] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proc. of ICML'07*, 2007.