RoBO: A Flexible and Robust Bayesian Optimization Framework in Python

Aaron Klein
Department of Computer Science
University of Freiburg
kleinaa@cs.uni-freiburg.de

Stefan Falkner
Department of Computer Science
University of Freiburg
sfalkner@cs.uni-freiburg.de

Numair Mansur
Department of Computer Science
University of Freiburg
mansurm@cs.uni-freiburg.de

Frank Hutter
Department of Computer Science
University of Freiburg
fh@cs.uni-freiburg.de

Abstract

Bayesian optimization is a powerful approach for the global derivative-free optimization of non-convex expensive functions. Even though there is a rich literature on Bayesian optimization, the source code of advanced methods is rarely available, making it difficult for practitioners to use them and for researchers to compare to and extend them. The BSD-licensed python package RoBO, released with this paper, tackles these problems by facilitating both ease of use and extensibility. Beyond the standard methods in Bayesian optimization, RoBO offers (to the best of our knowledge) the only available implementations of Bayesian optimization with Bayesian neural networks, multi-task optimization, and fast Bayesian hyperparameter optimization on large datasets (Fabolas).

1 Introduction

Bayesian optimization (BO) is a successful method for globally optimizing non-convex, expensive, and potentially noisy functions that do not offer any gradient information [Shahriari et al., 2016, Jones et al., 1998]. Due to its sample efficiency, BO has proven to be particularly useful for applications in which single function evaluations are very expensive.

In its vanilla blackbox version, BO tries to find a global optimizer $\boldsymbol{x} \in \arg\min f(\boldsymbol{x})$ of a blackbox function $f : \mathbb{X} \to \mathbb{R}$, only through noisy observations $y(\boldsymbol{x}) = f(\boldsymbol{x}) + \epsilon$ of the function, where $\epsilon \sim \mathcal{N}(0, \sigma^2_{noise})$. At BO's core is a probabilistic model $p(f|D)$ that captures the current belief of the objective function f given previous observations $D = \{(\boldsymbol{x}_0, y_0), \ldots, (\boldsymbol{x}_i, y_i)\}$. In each iteration i, BO optimizes a so-called acquisition function $a : \mathbb{X} \to \mathbb{R}$ based on $p(f|D)$ that trades off exploration and exploitation to determine the next query point $\boldsymbol{x}_{new} \in \arg\max a(\boldsymbol{x})$. After observing the outcome, $p(f|D)$ is updated and the next iteration begins.

In recent years, the traditional blackbox constraint has been lifted and several extensions of Bayesian optimization have been studied:

- The considerable cost usually associated with an observation can sometimes be reduced by evaluating related *tasks* which can be queried in addition to the function of interest as studied in the multi-task BO framwork of Swersky et al. [2013]. A special case of this is training a machine learning algorithm on only a subset of the data, as studied by Klein et al. [2017a].

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

- If the function to be optimized returns intermediate results, such as, e.g., the trace of an optimizer or the learning curve of a neural network, this additional information can be used to build a better probabilistic model. Handling this additional information effectively requires scalable models; to address this problem Swersky et al. [2014] introduced a special-purpose approximate GP and Klein et al. [2017b] studied a Bayesian neural network approach.
- Previous evaluations on related functions (e.g., other datasets) can accelerate the optimization, as studied for example by Swersky et al. [2013] and Feurer et al. [2015].

To facilitate work on Bayesian optimization that goes beyond blackbox optimization, we introduce RoBO, a new flexible Bayesian optimization framework in Python.

2 RoBO

RoBO is a new Bayesian optimization framework that offers an easy-to-use python interface inspired by the API of SciPy [Jones et al., 2001] to allow users to deploy it easily within their python programs. All code is published under the permissive BSD license and available at https://github.com/automl/RoBO. Tutorials and fully worked examples for using RoBO are available as part of the package's documentation. It provides implementations of different models and acquisition functions and in this way offers some robustness to guard against model mismatch.

2.1 Blackbox Optimization

At RoBO's core is an implementation of the standard Bayesian optimization algorithm that allows for a range of different models and acquisition functions.

Models. While most other BO packages only support Gaussian processes (GPs) [Rasmussen and Williams, 2006, Snoek et al., 2012] to model $p(f|D)$, GPs are not always the best choice. For example, while they tend to yield the best results in low-dimensional, continuous spaces, random forests [Breimann, 2001, Hutter et al., 2011] have been shown to yield better performance in high-dimensional spaces with conditional and categorical choices [Eggensperger et al., 2013]. In terms of computational complexity, vanilla GPs are limited by their cubic scaling in the number of data points, but BO with random forests or approximate GPs [Hutter et al., 2010] easily scales to many data points. A recent popular class of models are Bayesian neural networks [Neal, 1996], which combine well-calibrated uncertainty estimates with strong scalability to high dimensions and many data points; in particular, Snoek et al. [2015] used Bayesian linear regression based on the features in the last layer of a neural network, and Springenberg et al. [2016] used a fully-Bayesian treatment in their Bohamiann method by sampling the network weights using stochastic gradient Hamiltonian Monte-Carlo [Chen et al., 2014].

RoBO implements all of GPs, random forests, and the fully Bayesian neural network from Bohamiann, making it the BO framework that – to the best of our knowledge – supports the largest breadth of models; in particular, we are not aware of another BO framework that supports Bayesian neural networks. Figure 1 visualizes RoBO's different model fits on a small, one-dimensional example to illustrate the different models.

In contrast to random forests, GPs are usually very brittle with respect to their own hyperparameters. RoBO supports two different ways to tune the GP hyperparameters: (1) maximizing the marginal log-likelihood [Rasmussen and Williams, 2006] or (2) marginalizing over hyperparameters by sampling from the marginal log-likelihood [Snoek et al., 2012]. For each hyperparameter we define the same priors as described by Snoek et al. [2012].

Acquisition functions. RoBO supports standard acquisition functions from the literature, such as expected improvement (EI) [Jones et al., 1998], upper confidence bound [Srinivas et al., 2010] and probability of improvement [Jones et al., 1998]. It also supports entropy search [Hennig and Schuler, 2012], which, instead of trying to sample points with low function values, models the distribution of the optimum $p_{\min}(\boldsymbol{x} \mid \mathcal{D}) := p(\boldsymbol{x} \in \arg\min_{\boldsymbol{x}' \in \mathbb{X}} f(\boldsymbol{x}') \mid \mathcal{D})$ and, in each iteration, selects the point $\boldsymbol{x}_{new}$ that minimizes the entropy of this distribution. RoBO contains two different versions of this acquisition function, which use the EPMGP algorithm [Cunningham et al., 2012] or MC sampling to approximate $p_{\min}$, respectively. RoBO supports the use of CMA-ES [Hansen, 2006], DIRECT

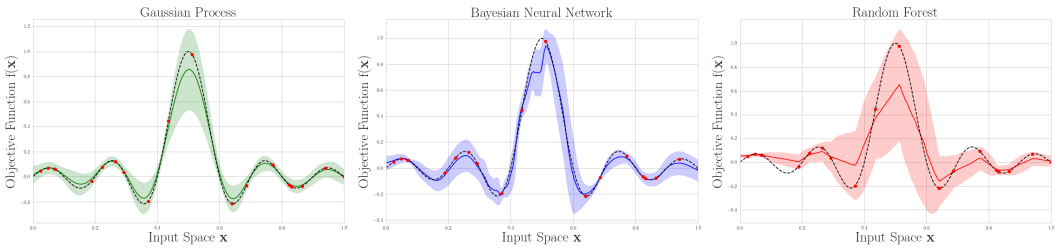

Figure 1: Comparison of different models for the objective function in a one dimensional example. Left: GP, middle: Bayesian neural network; right: random forest. While the Gaussian process is very expressive and allows to incorporate priors easily, the other two scale better to higher dimensions and a large number of observations.

[Jones et al., 1993], random search, and various optimizers from SciPy to optimize its acquisition function.

2.2 Beyond Blackbox Optimization

RoBO also goes beyond the optimization of a blackbox function $f : \mathbb{X} \to \mathbb{R}$ by allowing the specification of a function $g : \mathbb{X} \times \mathcal{T} \to \mathbb{R}$ that includes a task $t \in \mathcal{T}$ as additional input and is linked to f by $f(x) = g(x, t_{target})$. Specifically, RoBO implements two state-of-the-art methods for such problems:

Multi-Task Bayesian Optimization (MTBO). MTBO [Swersky et al., 2013] enables BO to make use of previously sampled function observations on different, but correlated tasks. This instantiates the domain $\mathcal{T}$ of the additional input variable t to a finite set, often just with two elements: an auxiliary task t_{aux} and an target task t_{target}. The auxiliary task can, e.g., describe the same algorithm evaluated on another dataset, a smaller dataset, or intermediate results in case of an iterative algorithm.

Fabolas. To speed up the hyperparameter optimization of machine learning algorithms on large datasets, Klein et al. [2017a] developed a GP-based BO method that can reason over arbitrary subsets of the training data. Their method Fabolas instantiates the domain of task variable t to $\mathbb{R}$, or, in general to $\mathbb{R}^d$. Applied to dataset subsampling, this allows it to quickly learn about the performance with a dataset size without ever evaluating with it. In particular, it can learn about the performance on the full dataset by only evaluating on subsets of various sizes and extrapolating its predictions.

3 Experiments

We now show two experiments that highlight RoBO's flexibility. First, we compare the different models and acquisition functions implemented in RoBO against the GP-based BO tool Spearmint [Snoek et al., 2012] and the random forest based BO tool SMAC [Hutter et al., 2011]. Then, we demonstrate the benefits of going beyond blackbox optimization by exploiting cheaper tasks. We provide code for running all of these experiments (as well as examples and tutorials) in RoBO's repository at https://github.com/automl/RoBO.

In the first experiment, we optimize three different synthetic functions from the blackbox optimization literature. In each iteration, we query every optimizer to return the estimated global optimum $\hat{\boldsymbol{x}}_\star$ (i. e. incumbent) and report the immediate regret $|f(\hat{\boldsymbol{x}}_\star) - f(\boldsymbol{x}_\star)|$ to the true optimum $\boldsymbol{x}_\star$. The methods are the following:

- **GP:** GP model, EI acquisition function, max. likelihood estimate of hyperparameters
- **GP-MCMC:** same as GP, but samples hyperparameters from the marginal likelihood
- **Entropy-Search:** same as GP-MCMC, but information gain as the acquisition function
- **Bohamiann:** Bayesian neural network model, expected improvement acquisition function
- **RF:** Random forest model, expected improvement acquisition function

Figure 2 shows the mean immediate regret of 50 independent runs of each method. As expected, the Gaussian process based optimization methods (GP, GP-MCMC, Entropy Search, and Spearmint)

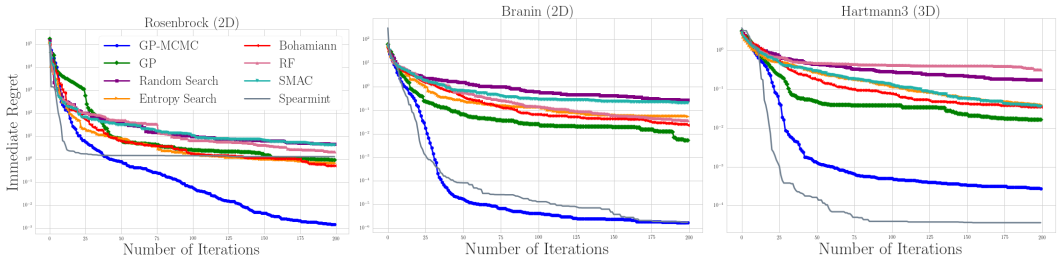

Figure 2: Immediate regret of 50 independent runs for various different optimization methods for Rosenbrock (left), Branin (middle) and Hartmann3 (right).

work very well on these low-dimensional continuous benchmarks, especially when the GP's hyperparameters are properly treated (GP-MCMC and Spearmint). RoBO's performance is very similar to that of Spearmint when using the same model type (GP-MCMC). While the random forest and Bayesian neural network models do not perform as well as the GP in these small-scale experiments, they exhibit better scaling behavior to high dimensions and many function evaluations.

In the second experiment, we showcase the large improvements that are possible by going beyond blackbox optimization. As a benchmark, we optimize the C and γ hyperparameters of a SVM on MNIST, using an additional input s to actively control the size of the dataset used for evaluating each combination of C and γ. Following Eggensperger et al. [2015], we constructed a surrogate benchmark based on performance evaluations for 400 different combinations of C and γ on MNIST collected by Klein et al. [2017a] for dataset sizes ranging in powers of two from $1/512$ of the dataset to the full one, and perform all our experiments on this *surrogate benchmark*.[1]

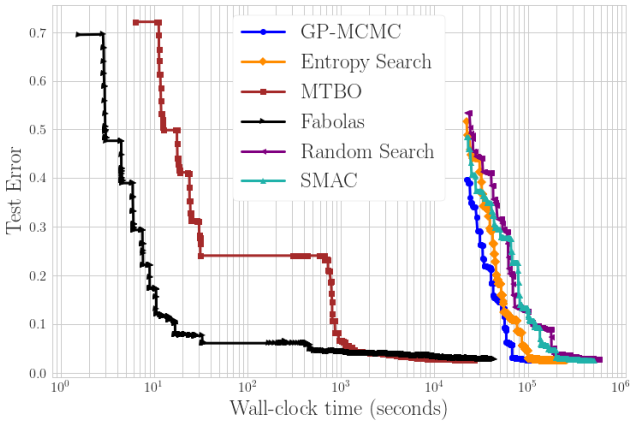

Figure 3: The test error achieved by different methods on the SVM surrogate task. Note the speedup by orders of magnitude achieved by Fabolas and MTBO utilizing subsets of the data for cheaper evaluations.

While standard BO methods are of course applicable for this problem, specialized methods can be dramatically faster by performing most of their optimization on cheap subsets of the data. In particular, we study Fabolas and MTBO, in the latter using a quarter of the total dataset as the auxiliary task. In Figure 3, we report the test error of the incumbent identified and the estimated wall-clock time that was necessary to find this configuration. We note that both MTBO and Fabolas have long converged by the time the blackbox optimization techniques have finished their first function evaluation. This is possible due to the much faster function evaluations when running on subsets of the data and demonstrates why it is crucial to go beyond the limiting blackbox formulation in hyperparameter optimization.

4 Conclusion

We introduced RoBO, a flexible Bayesian optimization framework in python. For standard GP-based blackbox optimization, its performance is on par with Spearmint while using the permissive BSD license. Most importantly, to the best of our knowledge, RoBO is the first BO package that includes Bayesian neural network models and that implements specialized BO methods that go beyond the blackbox paradigm to allow orders of magnitude speedup.

[1] Using a surrogate instead of evaluating an actual SVM in each function evaluation is much cheaper (thus allowing for more repetitions in order to obtain more reliable results) and also allows us to easily make the code for running the entire benchmarking experiment available. The repository also includes a fully-worked example for using Fabloas to tune an actual SVM implemented in scikit-learn [Pedregosa et al., 2011].

Acknowledgment

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 HU 1900/3-1).

References

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. *JGO*, 13: 455–492, 1998.

K. Swersky, J. Snoek, and R. Adams. Multi-task Bayesian optimization. In *Proc. of NIPS'13*, pages 2004–2012, 2013.

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.

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

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

M. Feurer, T. Springenberg, and F. Hutter. Initializing Bayesian hyperparameter optimization via meta-learning. In *Proc. of AAAI'15*, pages 1128–1135, 2015.

E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python. http://www.scipy.org/, 2001.

C. Rasmussen and C. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

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

L. Breimann. Random forests. *MLJ*, 45:5–32, 2001.

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.

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.

F. Hutter, H. Hoos, K. Leyton-Brown, and K. Murphy. Time-bounded sequential parameter optimization. In *Proc. of LION'10*, pages 281–298, 2010.

R. Neal. Bayesian learning for neural networks. *PhD thesis, University of Toronto*, 1996.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, Prabhat, and R. Adams. Scalable Bayesian optimization using deep neural networks. In *Proc. of ICML'15*, pages 2171–2180, 2015.

J. Springenberg, A. Klein, S.Falkner, and F. Hutter. Bayesian optimization with robust bayesian neural networks. In *Proc. of NIPS'16*, 2016.

T. Chen, E.B. Fox, and C. Guestrin. Stochastic gradient Hamiltonian Monte Carlo. In *Proc. of ICML'14*, 2014.

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*, pages 1015–1022, 2010.

P. Hennig and C. Schuler. Entropy search for information-efficient global optimization. *JMLR*, 98888(1): 1809–1837, 2012.

J. Cunningham, P. Hennig, and S. Lacoste-Julien. Approximate gaussian integration using expectation propagation. pages 1–11, January 2012.

N. Hansen. The CMA evolution strategy: a comparing review. In J.A. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, editors, *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, pages 75–102. Springer, 2006.

D. R. Jones, C. D. Perttunen, and B. E. Stuckman. Lipschitzian optimization without the lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181, Oct 1993.

K. Eggensperger, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proc. of AAAI'15*, pages 1114–1120, 2015.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *JMLR*, 12:2825–2830, 2011.