Hyper-optimization with Gaussian Process and Differential Evolution Algorithm

Jakub Klus `jakub.klus@innovatrics.com`
Innovatrics s.r.o

Pavel Grunt `pavel.grunt@innovatrics.com`
Innovatrics s.r.o

Martin Dobrovolný `martin.dobrovolny@innovatrics.com`
Innovatrics s.r.o

Abstract

Optimization of problems with high computational power demands is a challenging task. A probabilistic approach to such optimization called Bayesian optimization lowers performance demands by solving mathematically simpler model of the problem. Selected approach, Gaussian Process, models problem using a mixture of Gaussian functions. This paper presents specific modifications of Gaussian Process optimization components from available scientific libraries. Presented modifications were submitted to BlackBox 2020 challenge, where it outperformed some conventionally available optimization libraries.

1 Introduction

The recent popularity of Machine Learning (ML) methods broadens its applicability in many research fields. Selecting the best ML method and tuning its parameters is a common approach to achieve state-of-the-art results. But, with rising data size and method complexity, parameter optimization (so-called hyper-optimization) quickly exhausts available computational power. For example, the systematic optimization of deep learning (DL) training parameters [1, 2] can take several days to perform.

The time required for optimization is proportional to the number of iterations we perform (number of samples taken). The number of samples can be significantly lowered if we model the relation between individual parameters and outputs (so-called objective function). A probabilistic framework for the hyper-optimization is called Bayesian optimization [3]. Bayesian optimization defines a surrogate model of the objective function and loss function, which defines how optimal are future queries.

Shahriari *et al.* [4] provide a comprehensive review of Bayesian optimization applications and available implementations. Moreover, three basic methods of Bayesian optimization are described in [4], namely: Gaussian Process (GP), Tree of Parzen Estimators (TPE), and Random forests (RF).

A GP (also described in [5]) approximates objective function with a mixture of multivariate Gaussian functions. The loss of future queries is then calculated from the predicted mean value and confidence. On the contrary, the objective function in TPE (described thoroughly in [6]) is modelled by two hierarchical processes. One process describes parameters yielding better values, and the second describes parameters yielding worse values. Loss is then evaluated using the ratio of these two hierarchical processes. In RF regression a surrogate model consisting of an ensemble of simpler models made on random subsets of problems is constructed. A GP approach was selected for this

paper since it provides more space for experimental improvements as described in the following sections.

1.1 Kernel function

To estimate Gaussian mixture coefficients efficiently, covariances between individual Gaussian components are calculated using kernel trick [5]. Selection of kernel is one of the crucial parts of GP, individual kernel functions and even their arithmetic are described in [7, 5] Commonly used kernels are squared exponential kernel and Matérn 5/2 kernel [8]. Both aforementioned kernels implement automatic relevance determination (ARD). Briefly, ARD works by maximizing the marginal likelihood of the GP regression model with respect to the kernel length-scale parameters. If the length-scale corresponding to the objective function parameter is large, the model is almost independent of it.

1.2 Non-continuous parameters

A GP needs to be extended to work with non-continuous parameters like categories, integer numbers, or true/false values. Therefore, Garrido-Merchán and Hernández-Lobato in [9] proposed an extension of GP for non-continuous domains. They discourage the naive approach of converting parameters to the real domain and rounding outputs of GP afterwards. To truly model non-continuous parameters they suggest to modify kernel function and apply rounding inside of it. This ensures that interactions between individual parameters will not be extrapolated to undefined values (*e.g.* real values between integers). The difference of these approaches is depicted in Figure 1. The practical implementation is discussed in section 2.1.
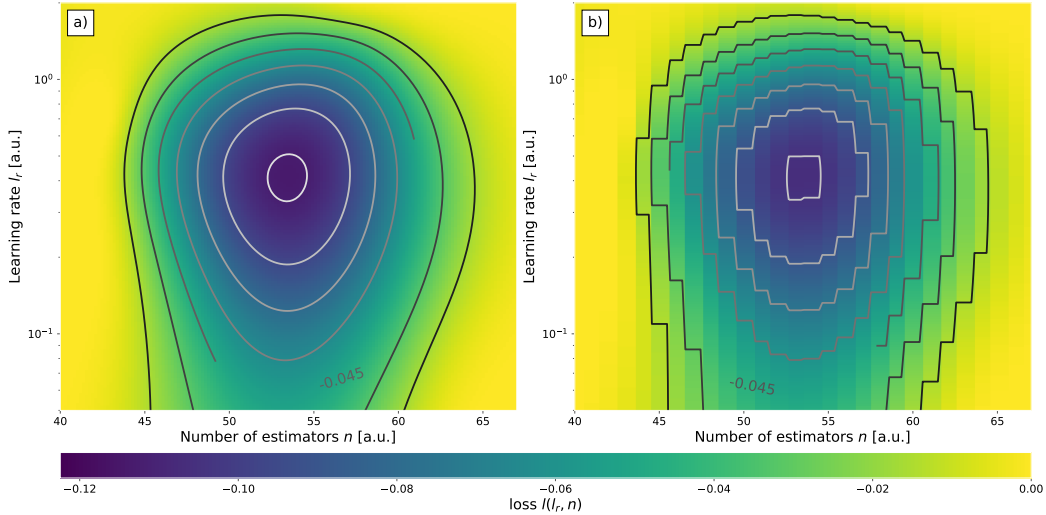

Figure 1: Comparison of loss function for a) kernel function without transformation, b) for kernel function with transformation (see section 2.1 for implementation details).

1.3 Parallelization

Bayesian optimization belongs to a family of sequential-model based optimization (SMBO) methods. In SMBO, a new query is calculated whenever a model prediction is updated with data acquired from the previous step. A GP must be modified to enable parallel evaluation of queries in batches. In [10] two basic methods of parallelization were proposed: *constant liar* and *kriging believer*. In both methods, a batch of queries is calculated sequentially by predicting the result of the first query and inserting it into the GP model. For the *constant liar* approach, known points are expanded by constant L corresponding to the parameters of the previous query. For the *kriging believer* approach, known points are expanded by the current GP model mean corresponding to the parameters of the previous query. The full potential of parallelization was exploited in [11], where authors sampled multiple queries from the loss function directly.

1.4 Challenge details

Algorithms described in this paper were submitted to the Black-Box Optimization for Machine Learning challenge (BlackBox 2020 challenge), a part of NeurIPS 2020 Competition Track. In short, the optimization was performed in 16 batches of 8 queries, and optimizers executing longer than 640 seconds were stopped. Moreover, the BlackBox 2020 challenge organizers defined a test set of assignments and kept a preliminary leaderboard of submissions. Organizers also provided a compilation of example submissions utilizing selected optimization libraries and means of comparison using a defined score.

2 Methods

The proposed solution is a combination of methods described in the previous section built upon a *scikit-learn* library [12]. Scikit-learn provides an implementation of GP regression and basic kernels, including kernel arithmetic. The selected loss function was Expected Improvement (EI) [3] defined as follows:

$$EI = \left(f_{\min} - \hat{y}\right) \Phi \left(\frac{f_{\min} - \hat{y}}{s} \right) + s\phi \left(\frac{f_{\min} - \hat{y}}{s} \right), \tag{1}$$

where $f_{\min}$ is the best observed value, $\hat{y}$ is the value predicted by GP model, s is the standard deviation of the model prediction, $\Phi\left(\cdot\right)$ is a standard normal distribution function, and $\phi\left(\cdot\right)$ is a standard normal density function.

The parallelization was implemented according to the *kriging believer* approach since the *constant liar* is straightforward and sampling of multiple queries is complicated.

2.1 Input space transformations

The nature of objective function parameters can affect the performance of the GP model. Therefore it is advisable to transform the aforementioned parameters into domains where their interactions are balanced. Such domain transforms can incorporate changes proposed in [9]. Numerical values are transformed with respect to their "configuration space" (*e.g.* parameters with "log" configuration space are transformed using $\log_{10}$ function). Categorical values are transformed to the range $[0.0, 1.0]$ using one-hot encoding, thus increasing the input dimension for GP. Boolean true and false values are transformed to 1.0 and 0.0 respectively.

The kernel function transformation is implemented using type coercion. Real numbers corresponding to integer objective function parameters are coerced by rounding. Real numbers from non-linear space are rounded to the nearest transformed integer (*e.g.* parameters from "log" configuration space are coerced by $\log_{10}(\lfloor 10^x \rceil)$). Each real number vector corresponding to a categorical value is modified by replacing its maximal value by 1.0 and the rest of its values by 0.0. Real numbers corresponding to Boolean values are rounded and clipped to range $[0.0, 1.0]$.

2.2 Initialization

The GP has to be primed with at least two queries. With respect to the parallelization, the number of prime queries has to be multiple of batch size. As only the objective function parameter ranges and spaces are known, the simplest option to initialize the GP is a batch of randomly generated parameter values. More advanced initialization mitigating possibilities of sampling nearly identical random parameters is Latin hypercube (LH) initialization [13]. In the LH initialization, we create a set of possible prime queries using a uniform grid over all parameters. The uniform used by the LH is constructed in the transformed input space. Afterwards, we randomly sample unseen points from the set of possible prime queries to create a batch.

2.3 Meta-optimizer

New queries from GP correspond to minimums in loss function (1). We call the optimization inside Bayesian optimization a *meta-optimization* and the selecting optimizer a *meta-optimizer*. We utilized methods provided in the *sckit-learn* library, namely the default optimizer L-BFGS (an approximation of Broyden–Fletcher–Goldfarb–Shanno algorithm [14]) and implementation of a

non-gradient method, Differential Evolution [15]. The Differential Evolution algorithm was chosen according to the investigation of transformed EI loss (see EI loss calculated on an example problem selected from the test set in figure 1b) because L-BFGS can have lower performance in non-continuous spaces. The negative impact of discretization on L-BFGS meta-optimizer is confirmed experimentally, see Table 1.

3   Results and Discussion

Proposed changes were evaluated on a test set given by organizers of BlackBox Challenge 2020. Initially, a problem of domain discretization with respect to meta-optimizers primed by two random batches was investigated.

Results in Table 1 show that Differential Evolution meta-optimizer performs better for complex kernel approximation. In contrast, the score of L-BFGS-B meta-optimizer for complex kernel transformation is significantly lower than the score for naive transformation. In sum, L-BFGS meta-optimizer outperforms Differential Evolution, but cannot beat example submissions.

Table 1: Performance of different meta-optimizers with respect to the discretization regime. Proposed modifications of domain discretizations are evaluated on two meta-optimizers and compared with example submissions (pySOT, turbo, and hyperOpt). The score is calculated from BlackBox Challenge 2020 test set.

Optimizer	Meta-optimizer	Discretization	Score
pySOT			98.385
turbo			97.660
Proposed	L-BFGS-B	naive	97.316
Proposed	L-BFGS-B	complex	96.556
Proposed	Differential Evolution	complex	96.296
hyperOpt			96.147
Proposed	Differential Evolution	naive	95.957

We tried to exploit potential benefits of employing Differential Evolution meta-optimizer by improving its initialization. Apart from the initialization type (discussed in Section 2) a size of initialization batch was concerned. Initialization batch size increment is motivated by the increase of the probability of hitting a promising parameter configuration. The last examined modification was initialization of meta-optimizer, denoted as *meta-initialization*. Random meta-initialization can sustain the same drawbacks as in the case of initialization. Therefore, we proposed a quasi-random meta-initialization that samples objective function parameters from the vicinity of known promising points.

Results in Table 2 show how can the initialization improve the performance of proposed optimizer. It can be stated that a larger number of priming batches helps in most cases, where initialization was not done purely randomly. Also, the non-random initialization improves the resulting score significantly. The combination of the Differential Evolution meta-optimizer, the complex discretization of the kernel, LH initialization, and quasi-random meta-initialization was chosen as the final submission for the BlackBox 2020 challenge.

4   Conclusion

This paper showed several modifications to an available implementation of Gaussian Process optimization procedure. Proposed modifications were evaluated on a test set provided by BlackBox 2020 challenge organizers. The evaluation showed that at least one example submission outperformed selected approach. This was also confirmed by the preliminary BlackBox 2020 challenge leaderboard, where the selected submission landed around 45th place.

However, at the final evaluation (disclosed to BlackBox 2020 challenge participants), the selected submission achieved the 11th place outperforming example submissions greatly. The best explanation we can give as authors is that kernel discretization can play a major role, especially for the case of categorical input parameters. This assumption is further supported by the fact that there were no assignments with categorical inputs in the available test set.

Table 2: Score values for different initialization and meta-initialization options. The evaluated proposal combines the Differential Evolution meta-optimizer and the complex discretization. The quasi-random meta-initialization method samples the objective function parameters from the vicinity of known promising points.

Optimizer	Initialization	Initialization size [batch]	Meta-initialization	Score
pySOT				98.385
Chosen	LH	5	quasi-random	97.871
Proposed	random	5	quasi-random	97.464
Proposed	LH	2	quasi-random	97.309
Proposed	random	2	quasi-random	97.028
Proposed	LH	2	random	95.957
Proposed	random	5	random	96.671
Proposed	LH	5	random	96.450
Proposed	random	2	random	96.296

Broader Impact

Authors believe that this work will motivate contributors and maintainers of black-box optimization scientific packages to incorporate proposed changes into their software. Our optimization framework is available at `https://github.com/Brown-Box/brown-box`.

Acknowledgments and Disclosure of Funding

References

[1] Leslie N. Smith. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *CoRR*, abs/1803.09820, 2018. URL `http://arxiv.org/abs/1803.09820`.

[2] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.

[3] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *J. of Global Optimization*, 13(4):455–492, December 1998. ISSN 0925-5001. doi: 10.1023/A:1008306431147. URL `https://doi.org/10.1023/A:1008306431147`.

[4] B. Shahriari, K. Swersky, Z. Wang, R. P. 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. doi: 10.1109/JPROC.2015.2494218.

[5] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. ISBN 026218253X. URL `http://www.amazon.com/Gaussian-Processes-Learning-Adaptive-Computation/dp/026218253X`.

[6] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[7] David Duvenaud. *Automatic model construction with Gaussian processes*. PhD thesis, University of Cambridge Repository, 2014. URL `https://www.repository.cam.ac.uk/handle/1810/247281`.

[8] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical bayesian optimization of machine learning algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'12, page 2951–2959, Red Hook, NY, USA, 2012. Curran Associates Inc.

[9] Eduardo C. Garrido-Merchán and Daniel Hernández-Lobato. Dealing with categorical and integer-valued variables in bayesian optimization with gaussian processes. *Neurocomputing*, 380:20 – 35, 2020. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2019.11.004. URL `http://www.sciencedirect.com/science/article/pii/S0925231219315619`.

[10] David Ginsbourger, Rodolphe Le Riche, and Laurent Carraro. *Kriging Is Well-Suited to Parallelize Optimization*, pages 131–162. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-10701-6. doi: 10.1007/978-3-642-10701-6_6. URL `https://doi.org/10.1007/978-3-642-10701-6_6`.

[11] Jialei Wang, Scott C. Clark, Eric Liu, and Peter I. Frazier. Parallel bayesian global optimization of expensive functions, 2019.

[12] 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. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[13] Kenny Q. Ye. Orthogonal column latin hypercubes and their application in computer experiments. *Journal of the American Statistical Association*, 93(444):1430–1439, 1998. doi: 10.1080/01621459.1998.10473803. URL `https://www.tandfonline.com/doi/abs/10.1080/01621459.1998.10473803`.

[14] R. Fletcher. *Practical Methods of Optimization; (2nd Ed.)*. Wiley-Interscience, USA, 1987. ISBN 0471915475.

[15] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997. ISSN 0925-5001. doi: 10.1023/A:1008202821328. URL `https://doi.org/10.1023/A:1008202821328`.

[16] Ole Tange. Gnu parallel 20200722 ('privacy shield'), July 2020. URL `https://doi.org/10.5281/zenodo.3956817`. GNU Parallel is a general parallelizer to run multiple serial command line programs in parallel without changing them.