Solving Black-Box Optimization Challenge via Learning Search Space Partition for Local Bayesian Optimization

Mikita Sazanovich[*]
JetBrains Research
St Petersburg, Russia
mikita.sazanovich@jetbrains.com

Anastasiya Nikolskaya[*]
JetBrains Research
St Petersburg, Russia
nastia.nikolskaya@gmail.com

Yury Belousov[*]
JetBrains Research
St Petersburg, Russia
yury-belousov@outlook.com

Aleksei Shpilman
JetBrains Research
St Petersburg, Russia
alexey@shpilman.com

Abstract

In this paper, we describe our approach to solving the black-box optimization challenge through learning search space partition for local Bayesian optimization. We develop an algorithm for low budget optimization. We further optimize the hyper-parameters of our algorithm using Bayesian optimization. Our approach ranks 3rd in the competition.

1 Introduction

Optimization of hyper-parameters for machine learning models is a common practice. Sometimes, it is done manually, but it could also be automated. Optimizing machine learning models while treating them as a black-box function is a part of black-box optimization. It has been successfully used for many different tasks such as hyper-parameter tuning for convolutional neural networks (Snoek et al. [2012]), policy optimization in reinforcement learning (Wang et al. [2020]), neural architecture search (Wang et al. [2019]).

The black-box optimization challenge focuses on the application of Bayesian optimization to tuning the hyper-parameters of machine learning models. In this competition, the participants are asked to optimize the hyper-parameters of an unknown objective function f. The algorithm is provided with the hyper-parameter configuration space: the number of hyper-parameters, their types (integer, real, categorical, or boolean), their spaces (linear, logarithmic, logit, or bi-logarithmic), and the lists or the ranges of possible values. The algorithm runs for $K = 16$ iterations and suggests $B = 8$ hyper-parameter sets (or points) $x_{k1}, ..., x_{kB}$ per iteration. It receives the value of the objective function for each of them, i.e., $y_{k1} = f(x_{k1}), ..., y_{kB} = f(x_{kB})$. The algorithm is expected to better understand how the objective function value depends on the hyper-parameter values by using the evaluation results from previous iterations. The final goal of the algorithm is to minimize the objective function value f.

[*]Equal contribution

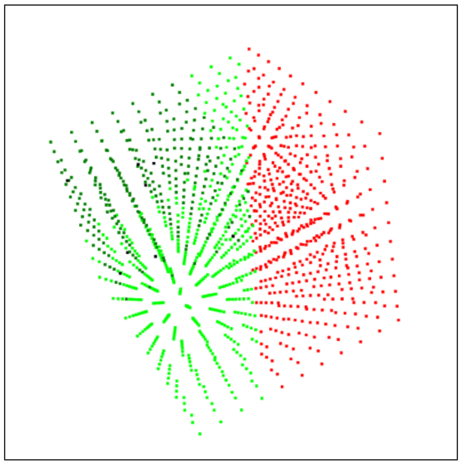

Figure 1: An example space partition for a 3D hyper-parameter space. The red points lie outside the selected region from the partition. The light green points are inside it but outside the region of the local Bayesian optimization model. The dark green points are some of the points which could be suggested next. In the end, the algorithm selects the black points for the next iteration.

2 Algorithm

2.1 High-level overview

To tackle this challenge, we develop the following method. The algorithm consists of several parts, including the initial sampling method, local Bayesian optimization, and learning search space partition for it.

First, we run the initial points generator to provide n_{init} points for our model to start. The n_{init} value could depend on the number of hyper-parameters or be a predetermined number. The initialization usually runs from 1 to 4 iterations. After the initialization, the space partitioning is built. The space partitioning is rebuilt every $n_{rebuild}$ iterations after its first construction. When the space partitioning is built or rebuilt, the local Bayesian optimization model runs in the space region with the lowest average objective function value. It is initialized from all previously evaluated points in the region, and it runs for $n_{rebuild}$ iterations. We set $n_{rebuild}$ to 4.

Furthermore, we reset the algorithm to its initial state (i.e., remove all accumulated points, its space partitioning, and begin from the initialization) every n_{reset} iterations if no progress has been made. To measure the progress, we consider the minimum function objective before and after the last n_{reset} iterations. If the minimum value is the same, we say that no progress has been made and reset our algorithm. It helps with getting out of the local minimum and making progress in the global optimization task. We set n_{reset} to 8.

2.2 Initial sampling method

One of the approaches for initial point generation is to generate them completely randomly. The downside of this is that there is no guarantee that these points are spread well enough across all the dimensions. Sampling methods such as Latin hypercube, Sobol, Halton, Hammersly (Greenhill et al. [2020]) and MaxPro (Joseph et al. [2015]) take advantage of the fact that we know beforehand how many random points we want to sample. Then the initial points can be sampled in a way that each dimension is explored. We experiment with these methods in our algorithm.

2.3 Local Bayesian optimization

We use the trust region Bayesian optimization (TuRBO) algorithm from Eriksson et al. [2019] as our local Bayesian optimization model. Considering the low budget for the number of iterations, we modify it with a decay factor, which shrinks the trust region. The policy we use is to decay the

Table 1: Hyper-parameters of finals candidates.

Configuration	Initial Sampling	n_{init}	Split model	Split kernel	Split regularization	$decay$
Candidate 1	Latin hypercube	8	SVM	rbf	0.002762	0.700
Candidate 2	Latin hypercube	24	SVM	poly	745.322745	0.499
Candidate 3	Latin hypercube	24	SVM	rbf	145.415497	0.416
Candidate 4	Latin hypercube	24	SVM	rbf	165.066908	0.549
Candidate 5	Latin hypercube	24	SVM	rbf	76.7041709	0.677

Table 2: Local and remote evaluation of finals candidates.

Configuration	Local scores mean	Local scores stddev	Remote scores mean	Remote scores stddev
Candidate 1	98.239	0.609	96.939	0.300
Candidate 2	**98.960**	**0.305**	**97.557**	0.281
Candidate 3	98.733	0.423	97.451	0.257
Candidate 4	98.828	0.599	97.345	0.167
Candidate 5	98.711	0.338	97.505	**0.117**

region side lengths by a constant factor $decay$ with each iteration if we have already used half of our iterations budget, i.e., past 8 iterations.

2.4 Learning search space partition

We intend on learning the space partition into regions with high/low objective function values, similar to Wang et al. [2020]. Using the space partition, we then select the region with the lowest average objective function value and run the local Bayesian optimization algorithm described above. Fig. 1 shows an example of such a space partition during a run of the algorithm.

More formally, when we construct a space partitioning at an iteration t (from 1 to K), we have a dataset D_t which consists of previously evaluated points (x_1, y_1), ..., (x_{n_t}, y_{n_t}), where $n_t = t * B$. We recursively split the current set of points into a left and a right sub-tree. The split is built as follows:

1. We run the KMeans algorithm to group the points into 2 clusters based on their objective function values y_i. The left sub-tree is formed from a set of points with a lower average function objective value.

2. Using KMeans algorithm labels as ground-truth labels, we train a split model to predict whether a set of hyper-parameters would fall into the first or the second cluster. We consider SVM with different kernels and k-nearest neighbors algorithm for the split model.

3. The split model filters the current set of points so that only the points which are predicted to be in the left sub-tree remain.

We continue to split the set of points until we reach the maximum depth $max_{depth} = 5$, or the new set of points is not large enough for the initialization of the local Bayesian optimization model.

2.5 Optimization

To set the right in the context of the competition hyper-parameters for our algorithm, we use the multi-task Bayesian optimization method from Letham and Bakshy [2019] to build a multi-task Gaussian process to combine the local and remote evaluation results. Using it, we generate 5 candidates for the pool of finals candidates. Table 1 shows the hyper-parameters of the generated set of candidates.

3 Results

3.1 Metric

The score given to a run of an algorithm on an objective function is the minimum value received by an algorithm normalized by the expected minimum and maximum objective function values. The score is then multiplied by 100. More formally, let f_a be a minimum value received by an algorithm, f_{min} be the expected minimum and f_{max} the expected maximum values. The score is computed as follows: $s_a = 100 * (1.0 - \frac{f_a - f_{min}}{f_{max} - f_{min}}) = 100 * \frac{f_{max} - f_a}{f_{max} - f_{min}}$.

3.2 Local evaluation

We evaluate the set of candidates from Sec. 2.5 on a number of tasks locally. We follow the suggestion from the competition authors and run experiments using bayesmark library[2]. The left side of Table 2 shows the results. We run each candidate locally 19 times.

3.3 Remote evaluation

We send the same set of candidates from Sec. 2.5 to a remote evaluation server. The right side of Table 2 presents the results. We evaluate each candidate remotely 3 times.

3.4 Competition finals

We select Candidate 2 for the competition finals based on the evaluation scores from all runs using the Wilcoxon signed-rank test with p-value less than 0.05. Our approach scores 92.509 in the finals and ranks 3rd overall.

4 Conclusion

In this paper, we present our approach to the black-box optimization challenge. The algorithm includes local Bayesian optimization and learning search space partition. We optimize the hyper-parameters of our algorithm by using a black-box optimization algorithm. Our selected algorithm ranks 3rd in the competition finals.

References

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2951–2959, 2012.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using monte carlo tree search. *ArXiv*, abs/2007.00708, 2020.

Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *ArXiv*, abs/1906.06832, 2019.

Stewart Greenhill, S. Rana, Sunil Gupta, Pratibha Vellanki, and S. Venkatesh. Bayesian optimization for adaptive experimental design: A review. *IEEE Access*, 8:13937–13948, 2020.

V. Joseph, E. Gul, and S. Ba. Maximum projection designs for computer experiments. *Biometrika*, 102:371–380, 2015.

David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Benjamin Letham and Eytan Bakshy. Bayesian optimization for policy search via online-offline experimentation. *Journal of Machine Learning Research*, 20(145):1–30, 2019.

[2]`https://github.com/uber/bayesmark`