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

Mikita Sazanovich*, Anastasiya Nikolskaya*, Yury Belousov*, Aleksei Shpilman

December 11, 2020

34th Conference on Neural Information Processing Systems

Introduction: Challenge

The black-box optimization challenge[1] focuses on the application of Bayesian optimization to tuning the hyper-parameters of machine learning models.

- An unknown objective function f.
- The hyper-parameter configuration space.
- The algorithm runs for $K = 16$ iterations.
- It suggests $B = 8$ hyper-parameter sets (or points) $x_{k1}, ..., x_{kB}$ per iteration and receives the value of the objective function for each of them, i.e., $y_{k1} = f(x_{k1}), ..., y_{kB} = f(x_{kB})$.
- The goal is to minimize the objective function value f.

[1] *https://bbochallenge.com*

Introduction: Scoring

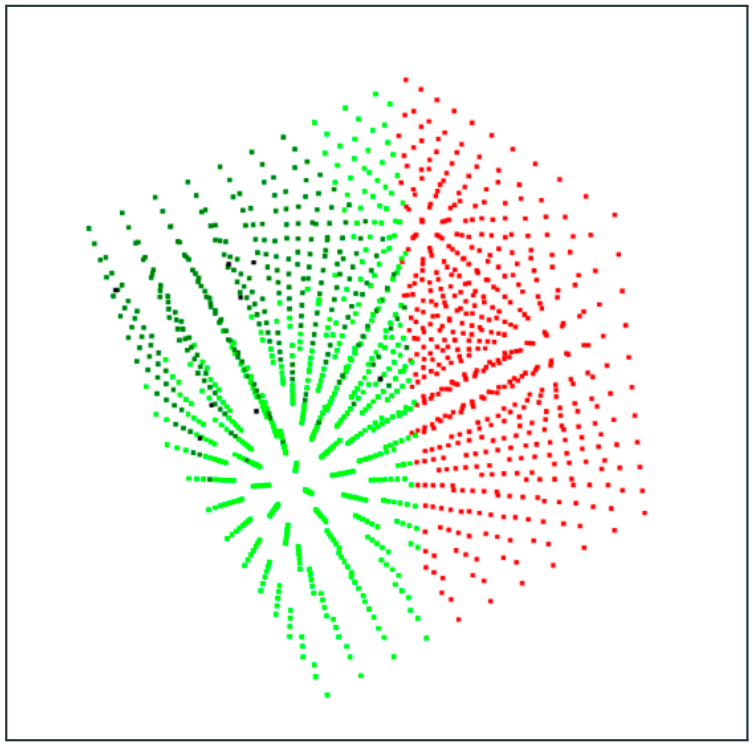
Let f_a be a minimum value received by an algorithm, f_{min} be the expected minimum and f_{max} the expected maximum values.

The score is computed as follows:
$s_a = 100 * (1.0 - \frac{f_a - f_{min}}{f_{max} - f_{min}}) = 100 * \frac{f_{max} - f_a}{f_{max} - f_{min}}$.

Algorithm: High-level Overview

The algorithm consists of several parts, including the initial sampling method, local Bayesian optimization, and learning search space partition for it.

Figure 1: An example space partition for a 3D hyper-parameter space.



We get n_{init} points, rebuild the partition every $n_{rebuild} = 4$ iterations, and reset the algorithm every $n_{reset} = 8$ if no progress is made.

Algorithm: Learning Search Space Partition

If we construct a space partition at an iteration t (from 1 to K), we have a dataset $D_t = \{(x_1, y_1), ..., (x_{n_t}, y_{n_t})\}$, where $n_t = t * B$. The split into a left and a right sub-tree is built as follows:

1. Run the KMeans algorithm for 2 clusters base on y_i. Lower value to the left sub-tree, higher – to the right.
2. Using the cluster assignments, we train SVM or k-nearest methods to predict the assignments.
3. Use the split model to filters the current set of points.

Do it recursively until we reach the maximum depth $max_{depth} = 5$, or there will not be enough points. Then we select the leftmost leaf.

The process is similar to Wang et al. [2020].

Algorithm: Local Bayesian Optimization

We use the trust region Bayesian optimization (TURBO) algorithm from Eriksson et al. [2019] as our local Bayesian optimization model.

Modifications:

1. Add a decay component which shrinks the trust region.
2. The policy is to decay the region side lengths by a constant factor *decay*.
3. Start the process if we have already used the half of our iterations budget, i.e., past 8 iterations.

Algorithm: Initial Sampling Method

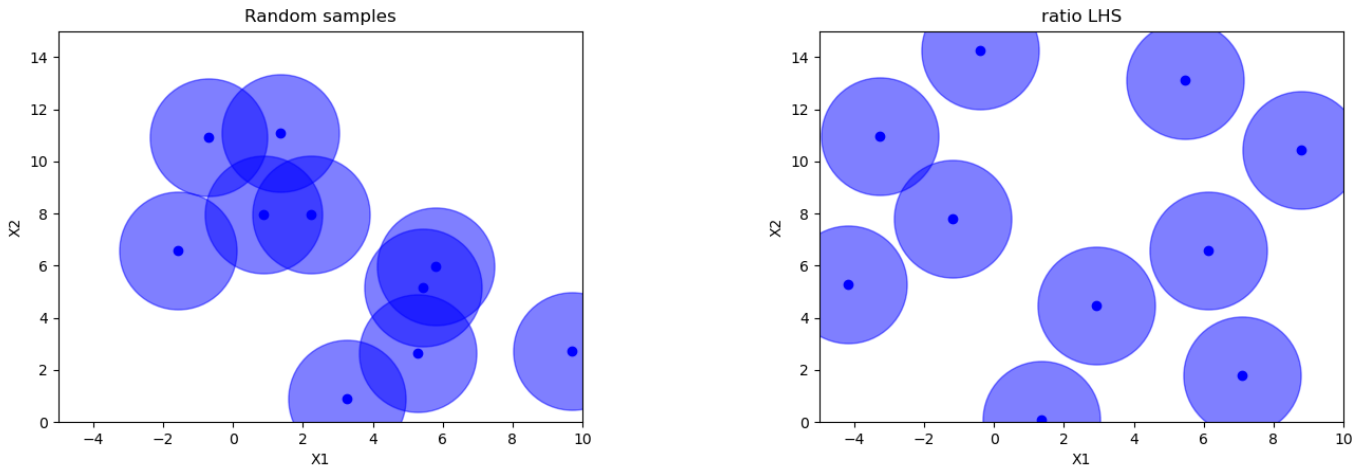
To overcome the problems with random initialization for points, we consider sampling methods such as:

- Space filling designs: Latin hypercube, Sobol, Halton, Hammersly (Greenhill et al. [2020]);
- MaxPro (Joseph et al. [2015]).

Why Random Initialization Is Not Optimal?

There is no guarantee that these points are spread well enough across all the dimensions

Figure 2: Initial samples over toy 2D example [2]



[2] Comparing initial sampling methods

Impact of Sampling Methods

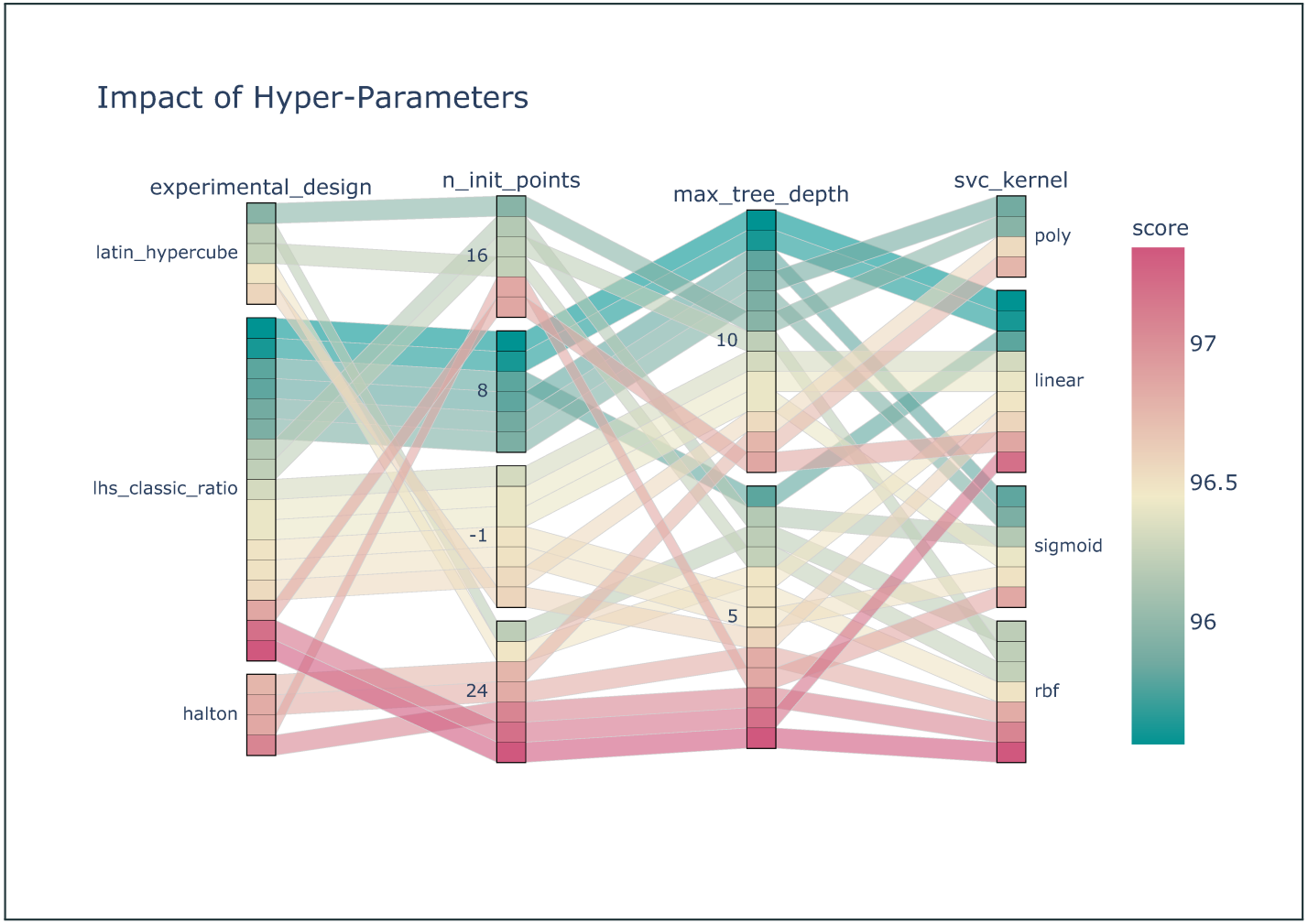
By taking into account the fact that we know beforehand how many initial points we want to sample, we can boost score by more than 0.5.

Table 1: Local and remote evaluation of sampling methods.

Method	Local mean	Local stddev	Remote mean	Remote stddev
random	95.586	1.273	91.749	0.366
halton	96.215	1.057	92.465	**0.132**
lhs	**96.939**	**0.662**	**92.537**	0.446

Impact of Hyper-Parameters

Figure 3: Parallel Categories Diagram Between Hyper-Parameters and Remote Score.



Algorithm: Optimization

So, how do we choose hyper-parameters for our algorithm? By black-box optimization!

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.

Algorithm: Candidates

Table 2: Hyper-parameters of finals candidates.

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

All candidates use Latin hypercube as the initial sampling method.

Results: Evaluation

Table 3: Local and remote evaluation of finals candidates.

Configuration	Local mean	Local stddev	Remote mean	Remote 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**

Candidate 2 uses SVM with polynomial kernel and regularization parameter of 745.322745.

Results: Competition Finals

We select the Candidate 2 for the competition finals based on the mean scores 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!

Results: What Didn't Work Well For Us

- Ensemble of algorithms.
- Custom kernels (*Matérn kernel 5/2* worked best).
- Custom acquisition functions.

Thanks! Questions?

References

D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek. Scalable global optimization via local bayesian optimization. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

S. Greenhill, S. Rana, S. Gupta, P. 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.

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

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