Proceedings of Machine Learning Research 133:3–26, 2021 NeurIPS 2020 Competition and Demonstration Track

Bayesian Optimization is Superior to Random Search for Machine Learning Hyperparameter Tuning: Analysis of the Black-Box Optimization Challenge 2020

Ryan Turner RTURNER@TWITTER.COM
Twitter
David Eriksson DERIKSSON@FB.COM
Facebook
Michael McCourt MCCOURT@SIGOPT.COM
SigOpt, an Intel company
Juha Kiili JUHA@VALOHAI.COM
Valohai
Eero Laaksonen EERO@VALOHAI.COM
Valohai
Zhen Xu XUZHEN@4PARADIGM.COM
4Paradigm
Isabelle Guyon GUYON@CHALEARN.ORG
ChaLearn

Editors: Hugo Jair Escalante and Katja Hofmann

Abstract

This paper presents the results and insights from the black-box optimization (BBO) challenge at NeurIPS 2020 which ran from July–October, 2020. The challenge emphasized the importance of evaluating derivative-free optimizers for tuning the hyperparameters of machine learning models. This was the first black-box optimization challenge with a machine learning emphasis. It was based on tuning (validation set) performance of standard machine learning models on real datasets. This competition has widespread impact as black-box optimization (e.g., Bayesian optimization) is relevant for hyperparameter tuning in almost every machine learning project as well as many applications outside of machine learning. The final leaderboard was determined using the optimization performance on held-out (hidden) objective functions, where the optimizers ran without human intervention. Baselines were set using the default settings of several open source black-box optimization packages as well as random search.

Keywords: Black-box optimization, Bayesian optimization, Meta-learning, AutoML, Hyperparameter tuning

1. Introduction

In black-box optimization we aim to solve the problem $\min_{x \in \Omega} f(x)$, where f is a computationally expensive black-box function and the domain Ω is commonly a hyper-rectangle. The fact that evaluations are computationally expensive typically limits the number of evaluations of f to a few hundred in most ML applications. In the black-box setting, no additional information is known about f and we observe no first- or second-order information when evaluating f. This is commonly referred to as derivative-free optimization.

Black-box optimization problems of this form appear everywhere. Most machine learning (ML) models have hyperparameters that require tuning via black-box (i.e., derivative-free) optimization (Snoek et al., 2012). Likewise, black-box optimization has wide applications in closely related areas such as signal processing (Turner and Rasmussen, 2012). These black-box optimization problems are often solved using Bayesian optimization (BO) methods (Frazier, 2018). BO methods rely on a (probabilistic) surrogate model for the objective function that provides a measure of uncertainty. This model is often a Gaussian process (GP) (Rasmussen, 2003), but other models such as Bayesian neural networks are also commonly used as long as they provide a measure of uncertainty. Using this surrogate model, an acquisition function is used to determine the most promising point to evaluate next, where popular options include expected improvement (EI) (Jones et al., 1998), knowledge gradient (KG) (Frazier et al., 2009), and entropy search (ES) (Hennig and Schuler, 2012). There are also other surrogate optimization methods that rely on deterministic surrogate models such as radial basis functions (Wendland, 2004; Fasshauer and McCourt, 2015), see Forrester et al. (2008) for an overview. The choice of surrogate model and acquisition function are both problem-dependent and the goal of this challenge is to compare different approaches over a large number of different problems. This was the first challenge aiming to find the best black-box optimizers specially for ML-related problems.

Moreover, many non-ML problems also benefit from the use of BO. For example, chemical engineering, materials discovery, manufacturing design, control systems and drug discovery are also effective uses of BO (Hernández-Lobato et al., 2017; Ueno et al., 2016; Frazier and Wang, 2016; Negoescu et al., 2011; Ju et al., 2017; Haghanifar et al., 2019; Candelieri et al., 2018; Gramacy et al., 2016; Calandra et al., 2016). Even real-world experiments, such as optimal web interfaces evaluated using A/B tests, have been optimized using BO (Letham et al., 2019). Hyperparameter tuning is such a demanded tool that all the major cloud platforms offer parameter tuning tools (Rodriguez, 2018). Additionally, there are small businesses (e.g., SigOpt) offering hyperparameter optimization as a service (OPTaaS).[1]

Evolutionary algorithms (EAs) (Yu and Gen, 2010) are another popular approach to black-box is optimization; they include methods such as differential evolution (DE), genetic algorithms (GAs), and the covariance matrix adaptation evolution strategy (CMA-ES) (Hansen et al., 2003). However, EAs generally require thousands of evaluations to be competitive with more sample-efficient methods such as BO, which may not be feasible when f is computationally expensive. Other possible options for solving black-box optimization problems include the Nelder-Mead algorithm (Nelder and Mead, 1965) and the quasi-Newton method BFGS (Liu and Nocedal, 1989) with gradients via finite differences.

While BO has gained a lot of traction over the last few years and open source packages have become very mature and robust, the study by Bouthillier and Varoquaux (2020) shows that there is still work to do to convince ML practitioners to use it to tune their algorithms. Surveying authors of papers published at NeurIPS 2019 and ICLR 2020 they found that while 80% of the NeurIPS papers and 88% of the ICLR papers tuned their hyperparameters,

[1] https://sigopt.com/

the vast majority used manual tuning, random search, or grid search (GS). In fact, only 7% of the NeurIPS papers and 6% of the ICLR papers used a different method such as BO.

Motivated by the original NeurIPS 2019 survey we decided to submit a proposal for a competition with the goal of decisively showing that BO and similar methods are superior choices over random search and grid search for tuning hyperparameters of ML models. This competition showed this advantage decisively and also provided guidance on how to select the best performing black-box optimization method.

2. Background

There have been benchmarks and competitions on black-box optimization, such as the annual COCO competition (Hansen et al., 2016). However, the COCO competition focused on synthetic problems like the Ackely and Rastrigin functions. Although being well suited towards unit test-like development cycles, these problems are systematically different from many real-world tasks: They often have highly correlated dimensions and high dynamic range. Many real-world tasks have irrelevant dimensions as well as complex noise patterns. Therefore, the goal of our competition was based on tuning for real-world ML tasks.

In this sense, the AutoML competition series (Liu et al., 2020b) is perhaps the closest competition to our competition. Other similar AutoML competitions were the NIPS 2006 model selection game[2] (Guyon et al., 2011, Sec. 1.5.1) and the "Beat AutoSKLearn" challenge.[3] However, there are some key differences between black-box optimization and AutoML. In AutoML, it is up to the algorithm to determine the *search space* and which ML method to try (e.g., MLP or random forest). To make matters more concrete, Figure 1 gives an example of a search space configuration in a scikit-learn SVM tuning problem that was provided to the optimization algorithms in our competition. Furthermore, an AutoML algorithm is given a training dataset, and must perform as well as possible on the test set without any human ever seeing the training set. By contrast, in black-box optimization, the search space is assigned to the algorithm. The algorithm is given access only to a black-box objective function, which in the general case may not even be ML-related. There is no training (or validation) data given to a black-box optimization algorithm.

There has been a related series of competitions to the COCO benchmark at the GECCO conference: BBComp (Molina et al., 2018; Loshchilov and Glasmachers, 2019). Although the BBComp objective functions are black-box, they have typically been synthetic. The exception is the EMO 2017 edition of BBComp, which used real-world problems. However, this challenge differed in several ways: EMO used only 10 test problems, of which 8 were multi-objective physical simulations and therefore not representative of typical ML use cases. There were also no practice problems given to the participants before evaluating them on the 10 test problems. In the spirit of ML, we think it is important to have a "training set" to iterate on. Finally, the BBComp explicitly allowed closed source submissions and human-in-the-loop optimizers. In this challenge, all submissions had to be open sourced to be eligible for a prize, which is essential as advancing the field is a goal of this challenge.

[2] http://clopinet.com/isabelle/Projects/NIPS2006/

[3] https://worksheets.codalab.org/worksheets/0x18a13ee4b0db4e098679f390bbd97fb2

```
svm_cfg = {
    "C": {"type": "real", "space": "log", "range": (1.0, 1e3)},
    "gamma": {"type": "real", "space": "log", "range": (1e-4, 1e-3)},
    "tol": {"type": "real", "space": "log", "range": (1e-5, 1e-1)},
}
```

Figure 1: An example search space configuration (for an SVM) from the challenge. Each key in the configuration is a hyperparameter to be optimized. Hyperparameters can be of different types: `real`, `int`, `cat`, or `bool`. All variables have a range tuple to specify a range of allowed guesses. Finally, there is a warping configuration to provide a sort of "prior" over what parts of the space need more resolution. These could be: `linear`, `log`, `logit`, or `bilog`.

3. Competition Setup

This is a new competition as there have been no ML-oriented black-box optimization competitions in the past.[4] The most similar competition, the previously mentioned AutoML series, maintains key differences (endemic between any black-box optimization competition and any AutoML competition). In AutoML, the algorithm gets to pick the search space while in black-box optimization, it is assigned to the algorithm by the user. The user often has intuition for reasonable hyperparameter ranges in ML problems, which makes it possible to frame it as a black-box optimization problem. While the survey by Bouthillier and Varoquaux (2020) showed that most ML researchers tune their hyperparameters, they do so using simple methods such as random search, grid search, and manual tuning. In addition, there are a large number of possible algorithms from BO and EAs, so concluding which method is preferable on real-world problems is a clear technical advance.

There were three different sets of problems where the participants' algorithms were evaluated: 1) The *local practice problems*; these problems were run locally on the participants' machines with full visibility into the models and data. 2) The *feedback leaderboard problems*; these problems were used to calculate the leaderboard score on the website during the three month feedback phase of the challenge. These problems were run in a cloud environment and completely hidden from the participant. Participants were limited to five submissions per day on the feedback problems. 3) The *final leaderboard problems*; these problems were used to calculate the final leaderboard score, which was posted on the website after the closing of submissions and used to determine prizes. These problems were also hidden like the feedback problems. To prevent overfitting, the participants' algorithms were only evaluated a single time on the final leaderboard problems. Problems were randomly split between feedback and final. The set of problems is discussed in more detail in Section 3.2.

However, the local practice problems were made from tuning ML models on public scikit-learn datasets, and therefore, not a random split of the other problems.

We provided a *starter kit*[5] to allow the participants to: 1) Locally score their submissions on the local practice problems with a single shell command; and 2) package their submissions

[4] https://bbochallenge.com/

[5] https://github.com/rdturnermtl/bbo_challenge_starter_kit/

into a compliant zip file for submission with a single shell command. All baseline and evaluation code runs on the CPU without the need for a GPU.

3.1. Baselines

The starter kit provided examples using the default configurations of several different optimization packages: Hyperopt (Bergstra et al., 2015), Nevergrad (Rapin and Teytaud, 2018), OpenTuner (Ansel et al., 2014), pySOT (Eriksson et al., 2019a), Scikit-Optimize (Head et al., 2018), and TuRBO (Eriksson et al., 2019b), which serve as baselines. However, the most natural single reference point is the performance of (included) random search.

These baselines were meant to give participants a good starting point, but there are many other possible packages such as Ax/BoTorch (Bakshy et al., 2018; Balandat et al., 2020), Cornell-MOE (Wu and Frazier, 2016), Dragonfly (Kandasamy et al., 2020), Emukit (Paleyes et al., 2019), GPFlowOpt (Knudde et al., 2017), GPyOpt (The GPyOpt authors, 2016), pycma (Hansen et al., 2019), RBFOpt (Costa and Nannicini, 2018), RoBO (Klein et al., 2017), ProBO (Neiswanger et al., 2019), and Spearmint (Snoek et al., 2012).

3.2. The Optimization Problems

The "dataset" for this competition was a collection of optimization problems. Therefore, we followed the same protocols that were followed in the AutoML competition for a "dataset of datasets". A collection of public scikit-learn datasets[6] were provided to the participants in local practice problems.

We obtain novel optimization problems via the Cartesian product of datasets, ML models, and evaluation metrics. For example, the following are all examples of optimization problems,

- Tune a GBDT on MNIST evaluated on accuracy on the validation set.
- Tune logistic regression on MNIST evaluated on log loss on the validation set.
- Tune an MLP on Boston housing evaluated on RMSE on the validation set.

Thus, informally,

$$\{\text{set of opt. problems}\} = \{\text{set of models}\} \times \{\text{set of datasets}\} \times \{\text{set of loss functions}\}\,.$$

The Cartesian product is violated slightly as different loss functions are used for classification and regression problems. Keeping many of these datasets completely hidden allows us to have test (optimization) problems unknown to the participants for both the feedback and final leaderboards. The search space varied by ML model and was provided to the algorithm by the benchmark. We summarize this space of problems across phases in Table 1.

This structure was chosen because in industrial settings we are often more concerned with *wall clock time* than raw CPU time. To keep this wall clock time reasonable, each submission was allowed a budget of 30 minutes per optimization run. Therefore, algorithms that perform well when making parallel suggestions are highly desirable. Much of the BO literature is focused on limiting the number function evaluations rather than iterations.

[6] The example scikit-learn datasets were: `digits`, `iris`, `wine`, `breast`, `boston`, and `diabetes`.

Table 1: A summary of the different model, loss, and dataset combinations that made up the different phases. Note that only the (local) practice problems were visible to the participants; both the feedback and final problems were hidden.

	Practice	Feedback	Final
Models	9	6	6
Loss functions	2	2	2
Datasets	6	5	5

The limitation of 16 rounds of guesses (iterations) is very small in the broader world of optimization.

3.3. Evaluation Metrics

In this challenge we use the open source package Bayesmark (Turner, 2019) to execute all the experiments inside a Docker container (Boettiger, 2015) and for scoring. The Bayesmark package has routines designed to deal with the subtleties of scoring black-box optimization algorithms. Scoring an optimization algorithm on a single problem is easy; simply take the minimum found by the optimizer after n function evaluations. Averaging over repeated trials can be done in noisy settings. Each repeated trial of a particular problem is known as a *study*.

However, averaging over many different problems becomes more subtle. We cannot simply average scores because they are all on different scales (units); such an approach builds in an arbitrary implicit weighting across problems. The Bayesmark package has a scoring system designed to deal with this problem. First, we normalize the performance on each problem so that a single RS suggestion has an average score of 1, and the global optimum has a score of 0.[7] Then, we can average the performance across multiple problems because the units are all the same.

Appendix B contains the equations for scoring taken directly from the Bayesmark documentation. The (feedback and final) leaderboard score is from Equation 7: $100 \times (1 -$ norm-mean-perf). Accordingly, the scoring is normalized such that scores vary from 0 (The optimizer on average is about the same as a single random search guess) to 100 (The optimizer finds the best known optimum every single time). This places the scoring on a normalized, unitless, and intuitive scale.

3.4. Evaluation Environment

The scoring and execution of runs in this challenge was handled using the open source Bayesmark package. Bayesmark is designed around optimizers that use a *suggest-observe* framework; and it provides a Python abstract class with the API. This suggest-observe framework is known as an *open-loop* optimizer. The participant's algorithm suggests $k = 8$

[7] On real problems we often do not know the true global optimum. So, we did many high-budget BO pilot runs on each problem to approximate the global optimum as best we could to use as the zero reference. This means it is possible, but difficult and rare, to get a score below zero.

guesses to be evaluated in parallel via the `suggest` function. The benchmark then evaluates the k different guesses and returns them to the algorithm via the `observe` function. The user just needs to provide a Python file with the Bayesmark `AbstractOptimizer` class implemented. This open-loop setup is desirable as it gives the user more flexibility on how (and if) to evaluate a suggestion. Furthermore, if the black-box being optimized is a real experiment (not a function in code) an open-loop setup is required.

In the context of a black-box optimization competition, each "data point" in the "training" or "test" set is an independent black-box optimization problem. This is similar to the AutoML competition where each "data point" is a dataset. For each optimization problem, the algorithm had access only to a search space specification and a black-box that evaluates the objective function.

For the local practice optimization problems, the evaluation of the objective functions was done on the participants' hardware. However, for the test problems (the feedback and final leaderboards), the objective function had to be hidden, and therefore the participants' submissions were run inside a Docker container in a cloud environment. The optimizers had a total of 640 seconds compute time for making suggestions on each problem (16 iterations with batch size of 8); or 40 seconds per iteration. Optimizers exceeding the time limits were cut off from making further suggestions and the best optima found before being killed was used. The participants were limited to five submissions per day. However, few teams made more than one submission per day.

In this challenge, we used the average score (see Equation 7) over $M = 60$ problems on the feedback leaderboard. A separate set of $M = 60$ problems were used for the final leaderboard. The two sets of problems were split randomly. The feedback leaderboard was run with $N = 10$ repeated trials. The final problems were run with $N = 30$ repeated trials. To ensure the final score was not due to chance, we re-ran the top-20 with $N = 100$ for the final leaderboard ranking.

The submissions were executed in Docker containers hosted on the Valohai platform.[8] Valohai provided a backend where resources could automatically scale. More than 300 worker machines were executing submissions during peak hours, and at quieter moments, not a single one. To prevent data exfiltration about the feedback leaderboard problems, the Docker images had no network/internet access and the participants were not allowed to see the logs. However, the Valohai platform provided easy access for the organizers to inspect the scoring tasks. The organizers could comment on issues through manual channels as they had access to the logs. Post-challenge the evaluation environment has been adapted to CodaLab for use as an "ever-lasting benchmark" in hyperparameter optimization courses.[9]

4. Learnings and Key Results

By the end of the challenge, there were 65 teams after filtering accounts that could not be verified through GitHub or LinkedIn. When testing on the final leaderboard problems, which were not previously available to competitors, most teams saw their gains persist. This implies that the submissions did not simply overfit to the local practice problems or the feedback leaderboard problems: There were actual insights which worked on previously

[8] https://valohai.com/

[9] https://competitions.codalab.org/competitions/28609

unseen ML problems. Out of the 65 total teams whose results appeared in the final leaderboard in Table 2, 61 beat the baseline random search and 23 beat TuRBO which was the strongest baseline in the starter kit. The final results are shown in Table 2 and Figure 2.

4.1. Error Analysis

Just getting a sensible scale for scoring is not enough to gain scientific rigor from this challenge; we need to do an error analysis. Also, for fairness, we wanted to be confident that we did not give prizes to a team due to chance. Our error analysis gave us confidence we did enough repeated trials such that the ranking of the final leaderboard was not due to a "lucky" random seed.

We ran the top-20 participants for $N = 100$ repeated trials to provide statistical confidence in the final results followed by a bootstrap-based analysis to get a "confidence set". Based on this analysis, we are 90% certain in the final top-5 as is shown in Table 3 and our resulting conclusions and learnings. Note that this bootstrap procedure is entirely a post-hoc error analysis and did not determine the final ranking on the leaderboard.

More precisely, we wanted to get error-bars on what the scores would be with an infinite number of repeated trials on these same problems. Note that the final normalized score is a grand-mean across all problems, which means we cannot use a simple t-test. Furthermore, we also wanted to translate that to a "confidence set" on the rankings. So, we opted to use a bootstrap-based analysis of the scores from each study.

To get the bootstrap re-sampled score, we re-sampled with replacement N scores for each problem separately and then took the average of those to get a bootstrap score. We repeated this process $B = 10^4$ times (more than enough for scalar estimation) to get the bootstrap distribution on scores. More formally,

$$S_{upn}^{(i)} \sim \text{Cat}(\{S_{upn}\}_{n=1}^{N}) \,, \quad i \in \{1, \ldots, B\} \,, \tag{1}$$

where Cat is a categorical distribution that is uniform over the elements provided as its argument. In other words, the score on problem p in the nth study of the ith bootstrap replication is re-sampled from the actual scores on problem p. The index u represents the user or team. We then ranked the teams within each bootstrap replication. We got separate parallel rankings for each bootstrap replication. This gives a bootstrap distribution of ranks.

4.2. Performance of the Baselines

In this section we report the score for the baselines provided in the starter kit with the addition of Ax, GPyOpt, and pycma. The score for the different baselines with the default options can be found in Table 4.

We see that TuRBO[10] performs the best with a score of 88.921 followed by pySOT which uses the stochastic RBF (SRBF) method (Regis and Shoemaker, 2007). Both TuRBO and pySOT use trust-region inspired methods, showing that a more local approach is advantageous for ML hyperparameter tuning. Scikit-Optimize, Ax, and GpyOpt, all use Bayesian optimization with a GP model. Scikit-Optimize uses a hedging strategy that uses multiple acquisition functions. Ax uses batch noisy EI (qNEI) (Letham et al., 2019) while

[10] https://github.com/uber-research/TuRBO

Table 2: The final rankings on the final leaderboard for the top-20. We show the final rank, team name, and (mean) score. For completeness, we also show the score using the median instead of the mean to aggregate scores across runs. We provide an analysis comparing this algorithm to the equivalent number of random search iterations ("RS Iters.") needed to obtain the same (mean) score. This is in comparison to the actual number of function evaluations in the challenge: $16 \times 8 = 128$; the ratio yields the "RS Efficiency" factor. The AutoML.org submission would have gotten 92.551 (3rd place) after correcting a minor bug in their submission that prevented their code from executing; so, we present them with rank "*" in this table.

Rank	Team	Score	Median	RS Iters.	RS Efficiency
1	Huawei Noah's Ark Lab	93.519	99.166	15,512	121.188
2	NVIDIA RAPIDS.AI	92.928	98.616	12,089	94.445
*	AutoML.org	92.551	98.693	10,353	80.883
3	JetBrains Research	92.509	99.131	10,179	79.523
4	Duxiaoman DI	92.212	99.027	9,032	70.562
5	Optuna Developers	91.806	99.156	7,698	60.141
6	Ambitious Audemer	91.107	96.668	5,899	46.086
7	jumpshot	91.089	97.056	5,861	45.789
8	KAIST OSI	90.872	98.659	5,409	42.258
9	Able Anteater	90.302	95.954	4,405	34.414
10	Oxford BXL	90.143	98.792	4,165	32.539
11	Innovatrics	90.081	97.062	4,076	31.844
12	IBM AI RBFOpt	90.050	96.117	4,032	31.500
13	Jim Liu	89.996	97.279	3,957	30.914
14	Jzkay	89.969	99.037	3,920	30.625
15	Better call Bayes	89.846	97.395	3,757	29.352
16	dannynguyen	89.706	98.800	3,581	27.977
17	AlexLekov	89.403	99.099	3,232	25.250
18	ABO	89.354	97.893	3,180	24.844
19	a2i2team	89.237	98.781	3,058	23.891
20	Tiny, Shiny & Don	89.229	96.513	3,050	23.828
	`Random Search (RS)`	75.815	88.746	128	1.000

GPyOpt uses EI with local penalization (González et al., 2016), which is the only option in GPyOpt that supports batch evaluations. The hyperopt package uses a tree-structured Parzen estimator (TPE) with EI, so these results indicate that using a GP model leads to better performance than using a TPE. Opentuner and pycma use EAs and are clearly not competitive with the BO packages.

Note that this comparison does not necessarily show that one package is better than another; it rather compares the performance of their default methods. Table 4 shows that the trust-region inspired method TuRBO and SRBF (pySOT) perform best out-of-the box, followed by packages with traditional BO methods as defaults. Both groups of methods perform better than the three packages that rely on EAs.

Table 3: Variation in final rankings (of all teams) across bootstrap replications. Each ranking's frequency in the bootstrap replications is shown in the bottom row. Teams outside these top-5 only appeared in the top-5 in $< 0.1\%$ of bootstrap replications. This analysis gives us near certainty that these top-5 (and the prize amounts) are not due to chance. Duxiaoman DI was the only source of variation. Their solution appears to have a higher variance than the others in the top-5.

Most likely ranking	2nd most likely ranking	3rd most likely ranking
Huawei Noah's Ark Lab NVIDIA RAPIDS.AI JetBrains Research Duxiaoman DI Optuna Developers	Huawei Noah's Ark Lab NVIDIA RAPIDS.AI JetBrains Research Optuna Developers Duxiaoman DI	Huawei Noah's Ark Lab NVIDIA RAPIDS.AI Duxiaoman DI JetBrains Research Optuna Developers
90%	5.2%	4.5%

Table 4: Performance of the example submissions provided to the participants. Like Table 2, we also show the random search equivalents.

Example Submission	Score	Median	RS Iters.	RS Efficiency
TuRBO	88.921	98.927	2,756	21.531
pySOT	88.419	97.324	2,346	18.328
Scikit-Optimize	88.085	96.054	2,114	16.516
Ax	86.977	97.042	1,516	11.844
GpyOpt	85.384	94.443	978	7.641
hyperopt	82.389	93.506	477	3.727
Nevergrad (1+1)	80.012	92.681	288	2.250
pycma	78.658	95.285	220	1.719
OpenTuner	76.854	90.073	156	1.219
`Random Search (RS)`	75.815	88.746	128	1.000

4.3. Bayesian Optimization was Consistently Effective

The submissions immediately bring one significant realization to the forefront: surrogate-assisted optimization is very effective. We discussed the performance of the baselines relative to random search in Section 4.2, but the performance of the participants was even more compelling (see Table 2).

All of the top-20 participants used some form of surrogate-assisted optimization. This is strongly indicative of the value of using a "surrogate model" and that intelligent modeling/decision making can significantly improve the optimization performance. Most top teams used a GP model and at least one of the commonly used acquisition functions, but as we will describe in the next section discovered the need for ensembling.

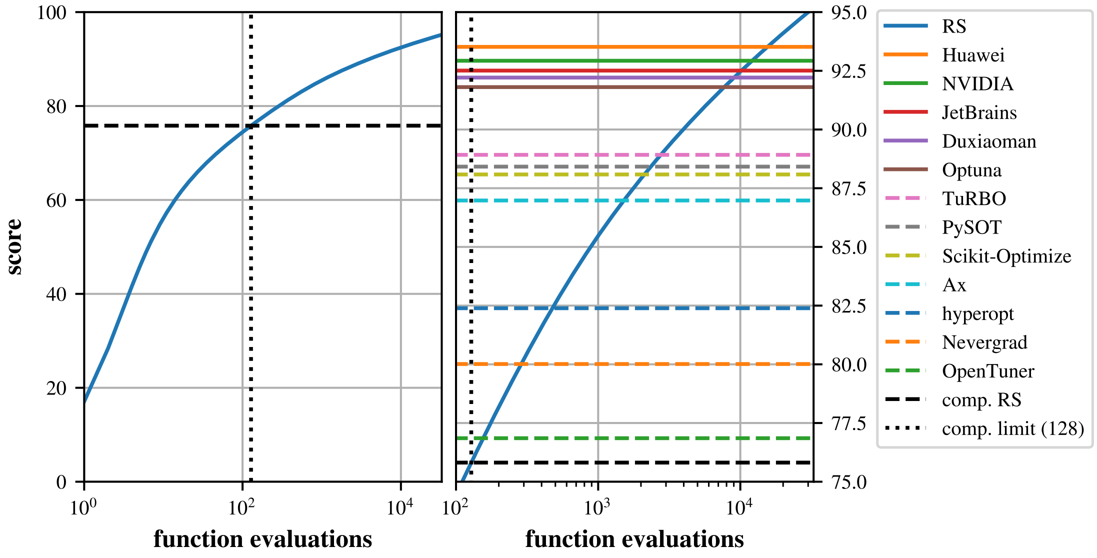

Figure 2: Top methods on the final leaderboard and example submissions vs random search (RS): On the left, we show what RS would have done given more function evaluations than allowed in the challenge (128). This performance curve is based on an unbiased estimate from pooling the data of $N = 256$ RS runs, which gives $256 \times 128 = 32{,}768$ function evaluations for each problem. As a reference we also show the function evaluation limit in the challenge (128) and the performance of RS at 128 function evaluations. On the right, we zoom into the relevant part of the plot and show the performance of the top-5 submissions and the example submissions from the starter kit. The top submissions clearly pull ahead of all the starter kit examples. The NVIDIA submission is a simple ensemble of the TuRBO and Scikit-Optimize examples. We note how much improvement is obtained over each of those solutions individually. Based on the intersection of the curves with the RS curve we can see the "RS Iters." from Table 4. Note that the x-axis is in logarithmic scale. This demonstrates the orders of magnitude more function evaluations that would be required to obtain the same performance as the top submissions using random search. Here, we show the performance on the final (not feedback) leaderboard. Thus the submissions were only evaluated one time on this test suite; this gives us confidence the performance we see here is not simply overfitting to the feedback leaderboard.

Note also that the baseline random search samples uniformly in the *warped space* from the search configuration. Thus, this baseline already outperforms a more naive random search that does not use the warping information, e.g., log scale vs linear scale. Nonetheless, the participants gained orders of magnitude greater search efficiency than random search as seen in Table 2.

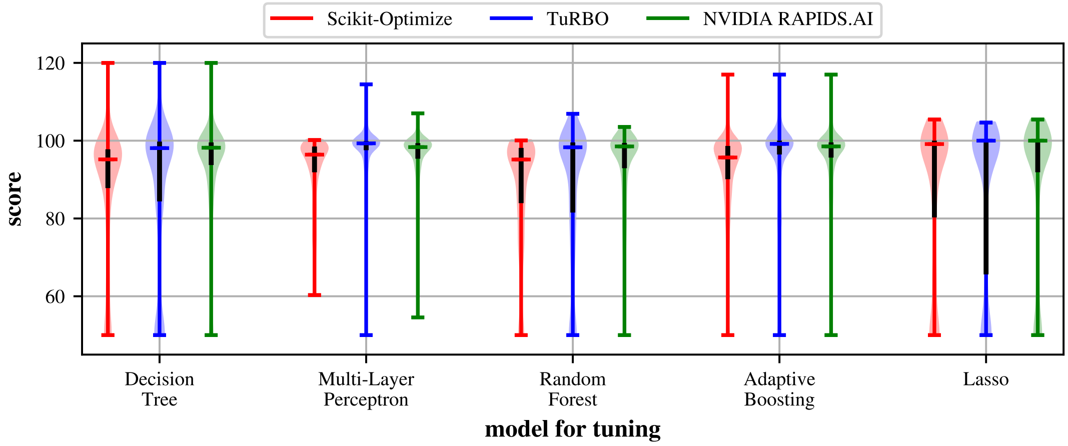

Figure 3: Comparison of the NVIDIA RAPIDS.AI solution and its components (Scikit-Optimize and TuRBO): Distribution of scores split across the different ML models for Scikit-Optimize, TuRBO, and NVIDIA RAPIDS.AI. The thick horizontal line shows the median performance and the vertical black line shows the interquartile range, the difference between 25th and 75th percentiles. While TuRBO often has a similar median to the NVIDIA RAPIDS.AI solution, the 25th percentile is much smaller on Lasso as well as on the decision tree and random forest problems, which affects the mean performance. Ensembling with another high-performance optimizer like Scikit-Optimize helps avoid this failure mode.

4.4. Ensemble Methods

Many published papers on BO propose using only one surrogate model and one acquisition function, despite some prior research having discussed the benefits of ensembling BO methods (Hoffman et al., 2011). Still, teams discovered that the large set of somewhat disparate problems were best treated through a mixture of methods. In fact, all of the methods in the top-10 had some sort of ensembling strategy.

For the purposes of this article we use the term ensemble in a broad sense. Ensembles could be built from multiple surrogates, acquisition functions, or potentially entire optimization algorithms, each of which could independently be used to fully power the optimization. The level at which these ensembling decisions were made varied across each of the teams. The first-placed team Huawei Noah's Ark Lab (Cowen-Rivers et al., 2020) used an elaborate compilation of acquisition functions and incorporated them into a multi-objective optimization strategy to select the next suggestions from the Pareto frontier. Other popular approaches were to alternate the kernel in the GP model or to use multiple surrogate models such as a GP and an XGBoost model.

Evolutionary methods also found their way into some ensembles — 2 of the top-10 submissions incorporated differential evolution into their optimization process. Of particular note, the AutoML team Awad et al. (2020) allocated the final 5 of their 16 batches to differential evolution in order to improve convergence close to the best point found in the first 11 batches. Similarly, Better call Bayes (Biswas et al., 2020) used pySOT and switched to DE for the final batches. This is a great example of how BO methods, while powerful, can be supplemented by other tools to help balance out their weaknesses.

Second-placed NVIDIA RAPIDS.AI (Liu et al., 2020a) and fourth-placed Duxiaoman DI (Wu et al., 2020) lived on the other extreme of simple ensemble methods; they employed straightforward ensembles of TuRBO + Scikit-Optimize and TuRBO + pySOT, respectively. NVIDIA's ensemble, selecting 50% of suggestions from each method, got a score of 92.928, beating both TuRBO (88.921) and Scikit-Optimize (88.085) when used individually by a large margin. Figure 3 shows an analysis comparing the NVIDIA RAPIDS.AI ensemble with its components. This analysis hints that ensembling may be useful in avoiding failed models where an individual BO algorithm makes little progress. The success of ensembles further justifies the use of open-loop optimizers. The implementation of the NVIDIA RAPIDS.AI solution was trivial due to the open-loop nature of the optimizers they were ensembling.

The results of this challenge show that a strong ensemble can be created by combining open source tools without prior knowledge of the underlying components. However, further analysis and ablation studies would be useful for fully understanding the mechanism of why ensembling works so well for BO.

4.5. Building with (and from) Open Source Tools

On the topic of Scikit-Optimize, TuRBO, and open source packages, all of the top-20 submissions had some open source elements present (including NumPy/SciPy). Realistically, nobody worked on a solution entirely independent of existing code: NVIDIA RAPIDS.AI stitched together a solution built entirely from unaltered open source projects; Huawei built on tools like GPy (which are common in the BO and GP community) and built their own strategy using them; AutoML started from open source tools that their research group previously built. This is a testament to the maturity of the open source computational Python community, and in particular, the ML/GP niche of that community.

Six out of the top-10 teams incorporated TuRBO into their solution, showing that trust region-based optimization works well even for the lower dimensional problems represented by these ML hyperparameter tuning tasks. In particular, JetBrains Research (Sazanovich et al., 2020) combined TuRBO with k-means to learn a partitioning of the search space similar to Wang et al. (2020). This prevalence of TuRBO may indicate that the function landscape is often non-smooth and that it can be beneficial to fit a local model rather than a global model. Or, it may indicate that TuRBO was the highest performing baseline provided to participants, and it was a logical starting place.

4.6. Discrete and Categorical Parameters

While surrogate-assisted optimization is very powerful, most literature on the topic deals with only continuous parameters. The Bayesmark tool allowed users to ignore the presence of integer and categorical parameters and computationally treat all parameters as continuous by encoding discrete and categorical parameters in a continuous space. Furthermore, tree based methods such as TPEs are generally considered to more naturally manage categorical parameters, but none of the participants used it in their solutions.

Still, a few participants chose to more actively recognize and manage integer and categorical parameters. The Optuna Developers (Shibata et al., 2020) built on top of TuRBO, but changed the size of the dimensions in the trust region to make sure that at least one value for each discrete parameters was always viable (that the trust region never moved/shrunk so much that none of the points in the trust region represent actionable parameters). KAIST OSI (Kim et al., 2020) took that approach and added in a multi-armed bandit strategy for recovering categorical parameters from their continuous embeddings.

4.7. Meta-Learning and Warm Starting

The competition was divided into a feedback session (which the participants could monitor through a practice leaderboard) and a final testing session (the results of which produced the final leaderboard, as seen in Table 2). The goal of the feedback period was to allow participants to measure their performance on problems which they could not observe and improve through that feedback. Many of the successful participants used this as an opportunity to set tunable elements of their submissions to high performing values; this, in effect, was meta-black-box optimization.

Some participants used this as an opportunity for meta-learning. While this was not the goal of the black-box optimization setting, the participants realized that this meta-learning can further improve the performance by transferring information from hyperparameter configurations applied to similar ML problems. To preserve the black-box nature of the challenge, the final testing was conducted with all anonymized parameter names (e.g., `P1`, `P2`). This negated the benefit of most meta-learning strategies.

But we were so excited by the effort put into meta-learning by these teams that we reran all submissions with full visibility into parameter names. This allowed teams to employ strategies such as making initial guesses using the found optima from problems with the same variable names under the premise that the objective functions are likely similar. Such *warm starting* of the optimization process led to major improvements for AutoML.org, DeepWisdom, dangnguyen, and Tiny, Shiny & Don; participants who ignored this data saw no significant change in performance from this extra information. These results were compiled into an alternate "warm start friendly" leaderboard in Table 5 where AutoML (Awad et al., 2020) emerged victorious. More details can be found in Appendix A.

4.8. Feedback from Participants

There were a few areas where we received requests from participants. Instead of the 8 × 16 design, some participants requested the ability to do 128 serial evaluations. However, we thought rewarding the capability of doing parallel optimization was critical, and therefore did not allow that capability. Other requests included internet connectivity from the Docker containers to be able to use proprietary SaaS-based optimizers, which ran counter to the objectives of the challenge. One participant requested the organizers to agree to a terms of service to be able to run a proprietary solution, which we did not do.

5. Discussion

This competition is only one of many addressing the automation of machine learning model development (Guyon et al., 2019). We hope that future organizers will take the progress made here and continue to develop new competitions which address aspects of automated machine learning which were ignored in this competition.

One point of focus in this competition was the black-box nature of each optimization problem. In other competitions, more knowledge about the machine learning circumstances were made available to the participants, but here we wanted to see how well optimization could be conducted on such problems without any knowledge of the problems. In future competitions, it might be interesting to find a middle ground — perhaps one where the type of model were known (e.g., XGBoost, which would give a benefit similar to what was seen on the warm start leaderboard) or the modeling circumstances were known (e.g., maximizing the F_1 score for a classification problem on imbalanced data). Even without access to the training data, there may still be significant opportunities for improved performance.

Multi-fidelity (or multi-information source) computations were not available in this competition, but they may be common in practical circumstances (Poloczek et al., 2017). Research has observed potential benefits from studying cheaply available (but lower fidelity) information such as through evaluating only a fraction of the training data or a small number of epochs.

Of particular interest is early stopping setups in ML models (Li et al., 2017). Often algorithms can guess a hyperparameter setting will perform poorly based on the start of the learning curve without completing the training algorithm. Such a mechanism could be made available in a black-box setting, which would give competitors the opportunity to more intelligently use their computational budget.

This competition required batches of 8 suggestions to emphasize the need for parallelism which is required for practical circumstances. Some of those circumstances would prefer asynchronous parallelism, where one suggestion is created given the other outstanding 7 suggestions currently being evaluated. Additionally, while we only focused on unconstrained single-objective optimization performance, many relevant problems have additional black-box constraints that need to be satisfied or are more naturally phrased as multi-objective optimization problems (Gardner et al., 2014; Hernández-Lobato et al., 2015; Eriksson and Poloczek, 2021; Knowles, 2006; Daulton et al., 2020).

6. Conclusions

The novelty and importance of the black-box optimization challenge gathered large interest with 65 teams and hundreds of participants; it is also hosted via an ongoing benchmark on CodaLab. It was the first optimization challenge evaluating derivative-free optimizers on ML-related problems. As such, it demonstrated decisively the benefits of Bayesian optimization over random search. The top submissions showed over 100× sample efficiency gains compared to random search. First, all of the top teams used some form of BO ensemble; sometimes with very simple and easy to productionize strategies such as alternating the surrogate, acquisition function, or potentially entire optimization algorithms. Second, the warm start leaderboard demonstrated how warm starting from even loosely related problems often yields large performance gains. Finally, this challenge offers many opportunities for extensions to test and push the boundaries on other aspects of black-box optimization.

Acknowledgments

We would like to thank the NeurIPS 2020 competition track organizers, Hugo Jair Escalante and Katja Hofmann, for their guidance during the challenge. We would also like to thank Newton Le and Serim Park for their feedback on the initial proposal.

References

Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An extensible framework for program autotuning. In *Proceedings of the 23rd international conference on Parallel architectures and compilation*, pages 303–316, 2014.

Noor Awad, Gresa Shala, Difan Deng, Neeratyoy Mallik, Matthias Feurer, Katharina Eggensperger, Andre' Biedenkapp, Diederick Vermetten, Hao Wang, Carola Doerr, Marius Lindauer, and Frank Hutter. Squirrel: A switching hyperparameter optimizer, 2020.

Eytan Bakshy, Lili Dworkin, Brian Karrer, Konstantin Kashin, Benjamin Letham, Ashwin Murthy, and Shaun Singh. AE: A domain-agnostic platform for adaptive experimentation. In *Workshop on Systems for ML*, 2018.

Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. Botorch: A framework for efficient Monte-Carlo Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

James Bergstra, Brent Komer, Chris Eliasmith, Dan Yamins, and David D Cox. Hyperopt: A Python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008, 2015.

Subhodip Biswas, Adam D Cobb, Andreea Sistrunk, Naren Ramakrishnan, and Brian Jalaian. Better call surrogates: A hybrid evolutionary algorithm for hyperparameter optimization, 2020.

Carl Boettiger. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

Xavier Bouthillier and Gaël Varoquaux. Survey of machine-learning experimental methods at NeurIPS2019 and ICLR2020. Research report, Inria Saclay Ile de France, January 2020. URL https://hal.archives-ouvertes.fr/hal-02447823.

Roberto Calandra, André Seyfarth, Jan Peters, and Marc Peter Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence*, 76(1):5–23, 2016.

Antonio Candelieri, Raffaele Perego, and Francesco Archetti. Bayesian optimization of pump operations in water distribution systems. *Journal of Global Optimization*, 71(1): 213–235, 2018.

Alberto Costa and Giacomo Nannicini. RBFOpt: An open-source library for black-box optimization with costly function evaluations. *Mathematical Programming Computation*, 10(4):597–629, 2018.

Alexander I. Cowen-Rivers, Wenlong Lyu, Zhi Wang, Rasul Tutunov, Hao Jianye, Jun Wang, and Haitham Bou Ammar. HEBO: Heteroscedastic evolutionary Bayesian optimisation, 2020.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *Advances in Neural Information Processing Systems 33*, 2020.

David Eriksson and Matthias Poloczek. Scalable constrained Bayesian optimization. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 730–738, 2021.

David Eriksson, David Bindel, and Christine A. Shoemaker. pySOT and POAP: An event-driven asynchronous framework for surrogate optimization, 2019a. arXiv preprint arXiv:1908.00420.

David Eriksson, Michael Pearce, Jacob R. Gardner, Ryan Turner, and Matthias Poloczek. Scalable global optimization via local Bayesian optimization. In *Advances in Neural Information Processing Systems 32*, pages 5497–5508, 2019b.

Gregory E Fasshauer and Michael J McCourt. *Kernel-based Approximation Methods Using MATLAB*, volume 19. World Scientific Publishing Company, 2015.

Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering Design via Surrogate Modelling: A Practical Guide*. John Wiley & Sons, 2008.

Peter Frazier, Warren Powell, and Savas Dayanik. The knowledge-gradient policy for correlated normal beliefs. *INFORMS Journal on Computing*, 21(4):599–613, 2009.

Peter I Frazier. A tutorial on Bayesian optimization. *arXiv preprint arXiv:1807.02811*, 2018.

Peter I Frazier and Jialei Wang. Bayesian optimization for materials design. In *Information Science for Materials Discovery and Design*, pages 45–75. Springer, 2016.

Jacob R. Gardner, Matt J. Kusner, Zhixiang Eddie Xu, Kilian Q. Weinberger, and John P. Cunningham. Bayesian optimization with inequality constraints. In *Proceedings of the 31th International Conference on Machine Learning*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 937–945. JMLR.org, 2014.

Javier González, Zhenwen Dai, Philipp Hennig, and Neil D. Lawrence. Batch Bayesian optimization via local penalization. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, volume 51 of *JMLR Workshop and Conference Proceedings*, pages 648–657. JMLR.org, 2016.

Robert B Gramacy, Genetha A Gray, Sébastien Le Digabel, Herbert KH Lee, Pritam Ranjan, Garth Wells, and Stefan M Wild. Modeling an augmented Lagrangian for blackbox constrained optimization. *Technometrics*, 58(1):1–11, 2016.

Isabelle Guyon, Gavin Cawley, Gideon Dror, and Amir Saffari. *Hands-On Pattern Recognition Challenges in Machine Learning*, volume 1. Microtome Publishing, 2011.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michéle Sebag, Alexander Statnikov, WeiWei Tu, and Evelyne Viegas. Analysis of the AutoML challenge series 2015–2018. In *AutoML*, Springer Series on Challenges in Machine Learning, 2019.

Sajad Haghanifar, Michael McCourt, Bolong Cheng, Jeffrey Wuenschell, Paul Ohodnicki, and Paul W Leu. Creating glasswing butterfly-inspired durable antifogging superomniphobic supertransmissive, superclear nanostructured glass through Bayesian learning and optimization. *Materials Horizons*, 6(8):1632–1642, 2019.

Nikolaus Hansen, Sibylle D Müller, and Petros Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

Nikolaus Hansen, Anne Auger, Olaf Mersmann, Tea Tusar, and Dimo Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *arXiv preprint arXiv:1603.08785*, 2016.

Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. CMA-ES/pycma on Github. *Zenodo, doi*, 10, 2019.

Tim Head, Gilles Louppe MechCoder, Iaroslav Shcherbatyi, et al. scikit-optimize/scikit-optimize: v0. 5.2, 2018.

Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

José Miguel Hernández-Lobato, Michael A. Gelbart, Matthew W. Hoffman, Ryan P. Adams, and Zoubin Ghahramani. Predictive entropy search for Bayesian optimization with unknown constraints. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1699–1707. JMLR.org, 2015.

José Miguel Hernández-Lobato, James Requeima, Edward O. Pyzer-Knapp, and Alán Aspuru-Guzik. Parallel and distributed Thompson sampling for large-scale accelerated exploration of chemical space. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1470–1479. PMLR, 2017.

Matthew Hoffman, Eric Brochu, and Nando de Freitas. Portfolio allocation for Bayesian optimization. In *Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence*, pages 327–336. AUAI Press, 2011.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Shenghong Ju, Takuma Shiga, Lei Feng, Zhufeng Hou, Koji Tsuda, and Junichiro Shiomi. Designing nanostructures for phonon transport via Bayesian optimization. *Physical Review X*, 7(2):021024, 2017.

Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R. Collins, Jeff Schneider, Barnabas Poczos, and Eric P. Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *Journal of Machine Learning Research*, 21(81):1–27, 2020.

Taehyeon Kim, Jaeyeon Ahn, Nakyil Kim, and Seyoung Yun. Adaptive local Bayesian optimization over multiple discrete variables, 2020.

Aaron Klein, Stefan Falkner, Numair Mansur, and Frank Hutter. RoBO: A flexible and robust Bayesian optimization framework in Python. In *NIPS 2017 Bayesian Optimization Workshop*, 2017.

Joshua Knowles. ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006.

Nicolas Knudde, Joachim van der Herten, Tom Dhaene, and Ivo Couckuyt. GPflowOpt: A Bayesian optimization library using TensorFlow. *arXiv preprint arXiv:1711.03845*, 2017. URL https://github.com/GPflow/GPflowOpt/.

Benjamin Letham, Brian Karrer, Guilherme Ottoni, Eytan Bakshy, et al. Constrained Bayesian optimization with noisy experiments. *Bayesian Analysis*, 14(2):495–519, 2019.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research*, 18(1):6765–6816, 2017.

Dong C Liu and Jorge Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528, 1989.

Jiwei Liu, Bojan Tunguz, and Gilberto Titericz. GPU accelerated exhaustive search for optimal ensemble of black-box optimization algorithms, 2020a.

Zhengying Liu, Zhen Xu, Shangeth Rajaa, Meysam Madadi, Julio CS Jacques Junior, Sergio Escalera, Adrien Pavao, Sebastien Treguer, Wei-Wei Tu, and Isabelle Guyon. Towards automated deep learning: Analysis of the AutoDL challenge series 2019. In *NeurIPS 2019 Competition and Demonstration Track*, pages 242–252. PMLR, 2020b.

Ilya Loshchilov and Tobias Glasmachers. Black box optimization competition BBComp, 2019. URL https://www.ini.rub.de/PEOPLE/glasmtbl/projects/bbcomp/index.html.

Daniel Molina, Antonio LaTorre, and Francisco Herrera. An insight into bio-inspired and evolutionary algorithms for global optimization: Review, analysis, and lessons learnt over a decade of competitions. *Cognitive Computation*, 10(4):517–544, 2018.

Diana M Negoescu, Peter I Frazier, and Warren B Powell. The knowledge-gradient algorithm for sequencing experiments in drug discovery. *INFORMS Journal on Computing*, 23(3): 346–363, 2011.

Willie Neiswanger, Kirthevasan Kandasamy, Barnabas Poczos, Jeff Schneider, and Eric Xing. ProBO: A framework for using probabilistic programming in Bayesian optimization. *arXiv preprint arXiv:1901.11515*, 9, 2019.

John A Nelder and Roger Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965.

Andrei Paleyes, Mark Pullin, Maren Mahsereci, Neil Lawrence, and Javier González. Emulation of physical processes with Emukit. In *Second Workshop on Machine Learning and the Physical Sciences, NeurIPS*, 2019.

Matthias Poloczek, Jialei Wang, and Peter I. Frazier. Multi-information source optimization. In *Advances in Neural Information Processing Systems 30*, pages 4288–4298, 2017.

Jérémy Rapin and Olivier Teytaud. Nevergrad - A gradient-free optimization platform. *version 0.2. 0, https://GitHub.com/FacebookResearch/Nevergrad*, 2018.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

Rommel G Regis and Christine A Shoemaker. A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19 (4):497–509, 2007.

Jesus Rodriguez. Hyperparameter tuning platforms are becoming a new market in the deep learning space. *Hacker Noon*, 2018. URL https://hackernoon.com/hyperparameter-tuning-platforms-are-becoming-a-new-market-in-the-deep-learning-space-7106f0ac1689.

Mikita Sazanovich, Anastasiya Nikolskaya, Yury Belousov, and Aleksei Shpilman. Solving black-box optimization challenge via learning search space partition for local Bayesian optimization, 2020.

Masashi Shibata, Toshihiko Yanase, Hideaki Imamura, Masahiro Nomura, Takeru Ohta, Shotaro Sano, and Hiroyuki Vincent Yamazaki. Team Optuna developers' method for black-box optimization challenge 2020, 2020. URL https://valohaichirpprod.blob.core.windows.net/papers/optuna.pdf.

Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems 25*, pages 2960–2968, 2012.

The GPyOpt authors. GPyOpt: A Bayesian optimization framework in Python, 2016. URL http://github.com/SheffieldML/GPyOpt/.

Ryan Turner. The Bayes opt benchmark documentation, 2019. URL https://bayesmark.readthedocs.io/.

Ryan Turner and Carl Edward Rasmussen. Model based learning of sigma points in unscented Kalman filtering. *Neurocomputing*, 80:47–53, 2012.

Tsuyoshi Ueno, Trevor David Rhone, Zhufeng Hou, Teruyasu Mizoguchi, and Koji Tsuda. COMBO: An efficient Bayesian optimization library for materials science. *Materials Discovery*, 4:18–21, 2016.

Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using Monte Carlo tree search. *arXiv preprint arXiv:2007.00708*, 2020.

Holger Wendland. *Scattered Data Approximation*, volume 17. Cambridge University Press, 2004.

Jiajun Wu, Manliang Cao, Liping Shan, and Qing Ying. Higher performance for AutoML: The benefit of various ensemble Bayesian optimization strategy, 2020. URL https://valohaichirpprod.blob.core.windows.net/papers/duxiaoman.pdf.

Jian Wu and Peter I. Frazier. The parallel knowledge gradient method for batch Bayesian optimization. In *Advances in Neural Information Processing Systems 29*, pages 3126–3134, 2016.

Xinjie Yu and Mitsuo Gen. *Introduction to Evolutionary Algorithms*. Springer Science & Business Media, 2010.

Appendix A. Warm Starting

Figure 4 shows the score of the top submissions on the warm start leaderboard compared to random search. Table 5 shows the top-20 leaderboard on the warm start leaderboard with the submission by AutoML.org coming out on top.

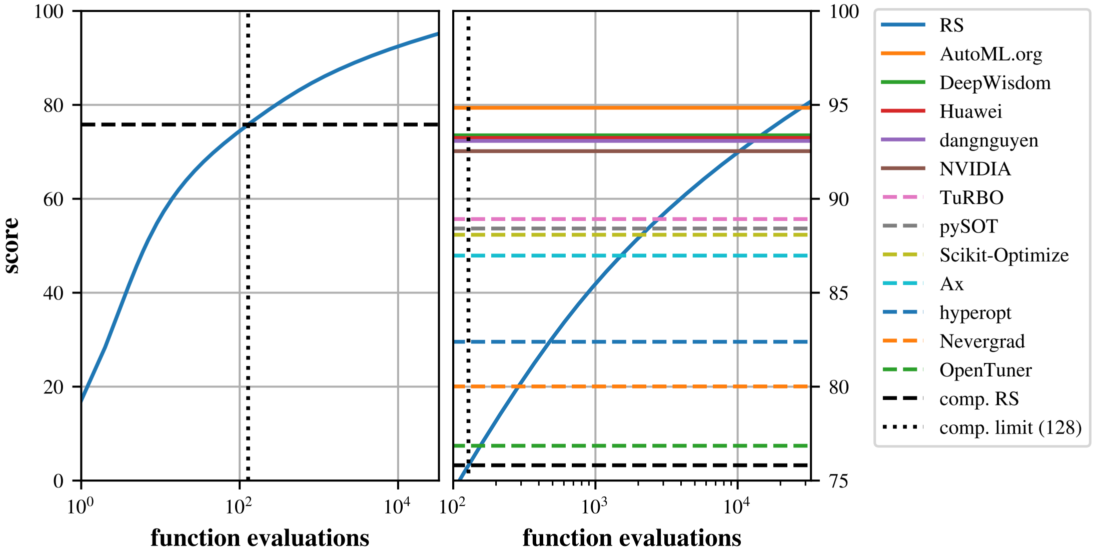

Figure 4: Top methods on the warm start leaderboard and example submissions vs random search (RS). This plot follows the same analysis as Figure 2. The most notable message from this plot is how the intensive use of warm starting by AutoML.org really allows them to "pull ahead of the pack".

Table 5: The final rankings on the warm start leaderboard for the top-20 in the same format as Table 2. We show the final rank, team name, and score. Here, AutoML.org, DeepWisdom, dangnguyen, and Tiny, Shiny & Don make big gains by warm starting from the parameter names, leveraging solutions from *different problems* with the same search space.

Rank	Team	Score	RS Iters.	RS Efficiency
1	AutoML.org	94.845	27,977	218.570
2	DeepWisdom	93.380	14,615	114.180
3	Huawei Noah's Ark Lab	93.241	13,777	107.633
4	dangnguyen	93.082	12,891	100.711
5	NVIDIA RAPIDS.AI	92.537	10,294	80.422
6	JetBrains Research	92.509	10,179	79.523
7	Duxiaoman DI	92.242	9,140	71.406
8	Optuna Developers	92.142	8,785	68.633
9	Tiny, Shiny & Don	92.108	8,667	67.711
10	KAIST OSI	91.272	6,277	49.039
11	jumpshot	91.115	5,917	46.227
12	Ambitious Audemer	90.999	5,669	44.289
13	Innovatrics	90.757	5,186	40.516
14	Jzkay	90.525	4,769	37.258
15	Oxford BXL	90.403	4,566	35.672
16	IBM AI RBFOpt	90.370	4,513	35.258
17	Better call Bayes	90.104	4,108	32.094
18	Able Anteater	90.036	4,012	31.344
19	Jim Liu	89.972	3,924	30.656
20	IBM Research-China	89.834	3,741	29.227
	Random Search (RS)	75.815	128	1.000

Appendix B. How Scoring Works

The scoring system is about aggregating the function evaluations of the optimizers. We represent F_{ptn} as the function evaluation of objective function p (`TEST_CASE`) at batch t (`ITER`) under repeated trial n (`TRIAL`). In the case of batch sizes greater than 1, F_{ptn} is the minimum function evaluation across the suggestions in batch t. The first transformation is that we consider the *cumulative minimum* over batches t as the performance of the optimizer on a particular trial:

$$S_{ptn} = \text{cumm-min}_t F_{ptn} \, . \tag{2}$$

From a decision theoretical perspective it is more sensible to consider the mean (possibly warped) score. Robust measures like the median score can hide a high percentage of runs that completely fail. However, when we look at the mean score we first take the clipped score with a baseline value:

$$S'_{ptn} = \min(S_{ptn}, \, \text{clip}_p) \, . \tag{3}$$

This is largely because there may be a non-zero probability of $F = \infty$ (as in when the objective function crashes), which means that mean random search performance is infinite loss. We set clip_p to the median score after a single function evaluation of random search. The mean performance on a single problem then becomes:

$$\text{mean-perf}_{pt} = \text{mean}_n \, S'_{ptn} \, . \tag{4}$$

Which then becomes a normalized performance (via linear rescaling between optimal and RS performance) of:

$$\text{norm-mean-perf}_{pt} = \frac{\text{mean-perf}_{pt} - \text{opt}_p}{\text{clip}_p - \text{opt}_p} \, . \tag{5}$$

Again, we can aggregate this into all objective function performance with:

$$\text{mean-perf}_t = \text{mean}_p \, \text{norm-mean-perf}_{pt} \, , \tag{6}$$

which is a mean-of-means (or *grand mean*).

For the leaderboard, we use 16 batches ($t = 16$) of batch size 8; and we transform the scores on a 0 to 100 scale. In equations,

$$\text{leaderboard-score} = 100 \times (1 - \text{mean-perf}_{16}) \, . \tag{7}$$

This means that an algorithm whose final solution is only as good as a single guess of random search has score 0. Meanwhile, an algorithm whose final solution *always* finds the best known global optimum has score 100. This scoring system is invariant to independent linear rescaling of any of the objective functions.